Distributed Systems - Overview

- some systems background/context
- some legal/social context
- development of technology – DS evolution
- ** DS fundamental characteristics **
- software structure for a node
- model/architecture/engineering for a DS
- architectures for large-scale DS
  - federated administration domains
  - integrated domain-independent services
  - detached, ad-hoc groups

Costly Failures in Large-Scale Systems

- UK Stock Exchange - share trading system
  - abandoned 1993, cost £400M
- CA automated childcare support
  - pended 1997, cost $300M
- US tax system modernisation
  - scrapped 1997, cost $4B
- UK ASSIST, statistics on welfare benefits
  - terminated 1994, cost £3.5M
- London Ambulance Service Computer Aided Despatching (LASCAD) scrapped 1992, cost £7.5M, 20 lives lost in 2 days

Why high public expectation?

Web experience

- e.g. information services: trains, postcodes, phone numbers
- e.g. online banking
- e.g. airline reservation
- e.g. conference management
- e.g. online shopping and auction

Properties: read mostly, server model, client-server paradigm, closely coupled, synchronous interaction (request-reply), single-purpose, (often) private sector

Public-Sector Systems

healthcare, police, social services, immigration, passports, DVLA (driver + vehicle licensing), court-case workflow, tax, independent living for the aged and disabled, …

- bespoke and complex
- large scale
- many types of client (many roles)
- web portal interface, but often not web-service model
- long timescale, high cost
- ubiquitous and mobile computing – still under research
- former policy of competition and independent procurement
- current policy requiring interoperation
- legislation and government policy

Some Legal/Policy Requirements - 1

"*patients may specify who may see, and not see, their electronic health records (EHRs)*" - exclusions

"*only the doctor with whom the patient is registered (for treatment) may e.g. prescribe drugs, read the patient's EHR, etc.*" - relationships

"*the existence of certain sensitive components of EHRs must be invisible, except to explicitly authorised roles*"

Some Legal/Policy Requirements - 2

"*buses should run to time and bus operators will be punished if published timetables are not met*."

so bus operators refuse to cooperate in traffic monitoring, even though monitoring could show that delay is often not their fault.

Data Protection Legislation

Gathered data that identifies individuals must not be stored:

CCTV cameras: software must not *recognise* people and store identities with images

(*thermal imaging (infra-red) - just monitor/count*)

Vehicle number plate recognition: must not be associated with people then stored with identities

(*only police allowed to look up*)

Police records: accusations that are not upheld?

Sally Geeson murder - previous army records of LC Atkinson

Soham murders – previous police records of Huntley;

Govt. now require interaction between counties

UK Freedom of Information Act: Jan 2005

Rapidly Developing and New Technology

- can't ever design a "*second system*", it's always possible to do more next time

- rapid obsolescence - incremental *growth* not sustainable long-term (unlike e.g. telephone system)

  - a current software engineering research area

- but *big-bang* deployment is a bad idea

  design for incremental *deployment*

- mobile workers in healthcare, police, utilities etc.

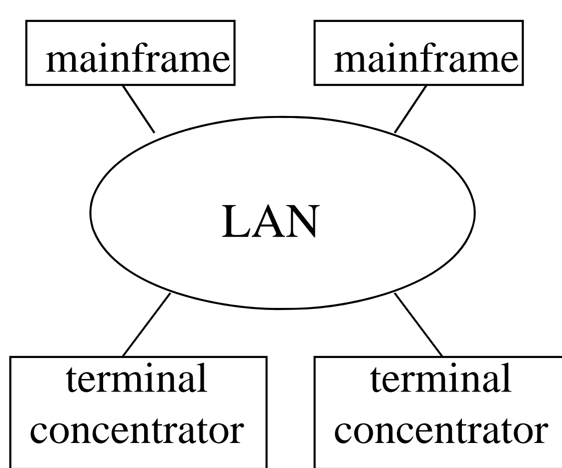
  - integration of wired and wireless networks

- ubiquitous computing: integration of camera and sensor data

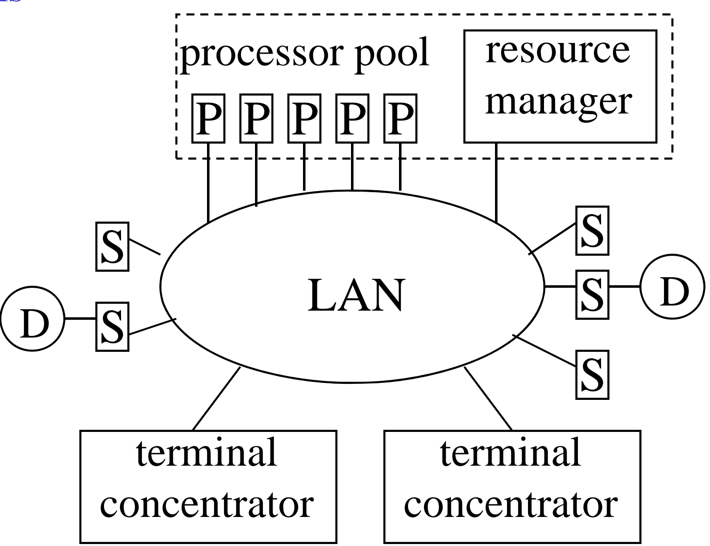
DS history: technology-driven evolution

- Fast, reliable (interconnected) LANs (e.g. Ethernet, Cambridge Ring) made DS possible in 1980s
- Early research was on distribution of OS functionality
  1. terminals + multiaccess systems
  2. terminals + pool of processors + dedicated servers (Cambridge CDCS)
  3. Diskless workstations + servers (Stanford)
  4. Workstations + servers (Xerox PARC)
- Now WANs are fast and reliable, …….

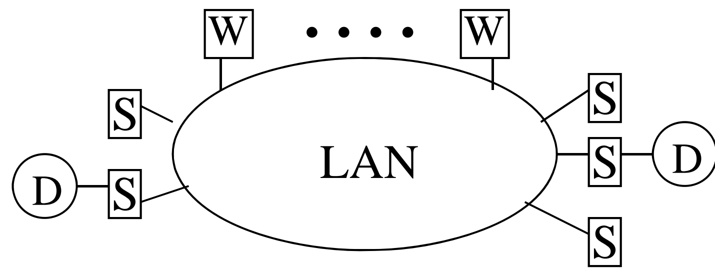
1. LAN as terminal switch to multiaccess systems



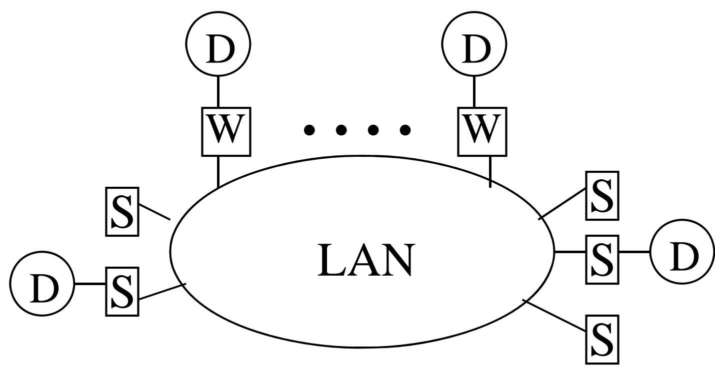
2 terminals + processor pool + servers



3 diskless workstations + servers



4 workstations + servers



Technology-driven evolution – comms.

- WANs *quickly* became as high bandwidth and reliable as LANs
- Distributed database research such as data-shipping -vs- query-shipping became obsolete in the 1990s
- Web services created new problems – flash crowds
- Bandwidth had become high, but latency was and remains a problem, due to end-system processing time for huge numbers of clients
- See lecture DS-7: content storage and delivery for web servers

How to think about Distributed Systems

- fundamental characteristics
- software structure for a node
- model/architecture/engineering for a system

DS fundamental characteristics

1. Concurrent execution of components
2. Independent failure modes
3. Transmission delay
4. No global time

Implications:

2, 3 - can't know why there's no reply – node/comms. failure and/or node/comms. congestion

4 - can't use locally generated timestamps for ordering distributed events

1, 3 - inconsistent views of state/data when it's distributed

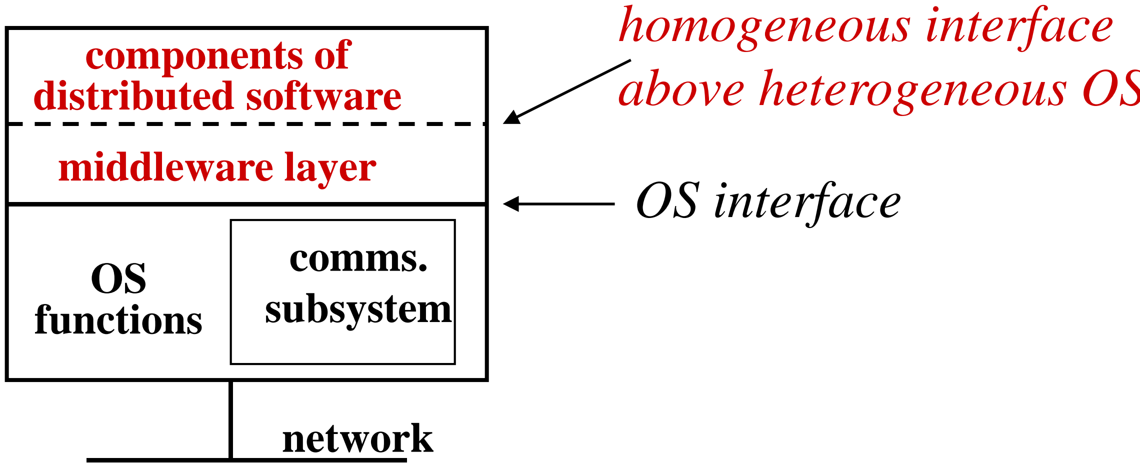
1 - can't wait for quiescence to resolve inconsistencies

single node - software structure

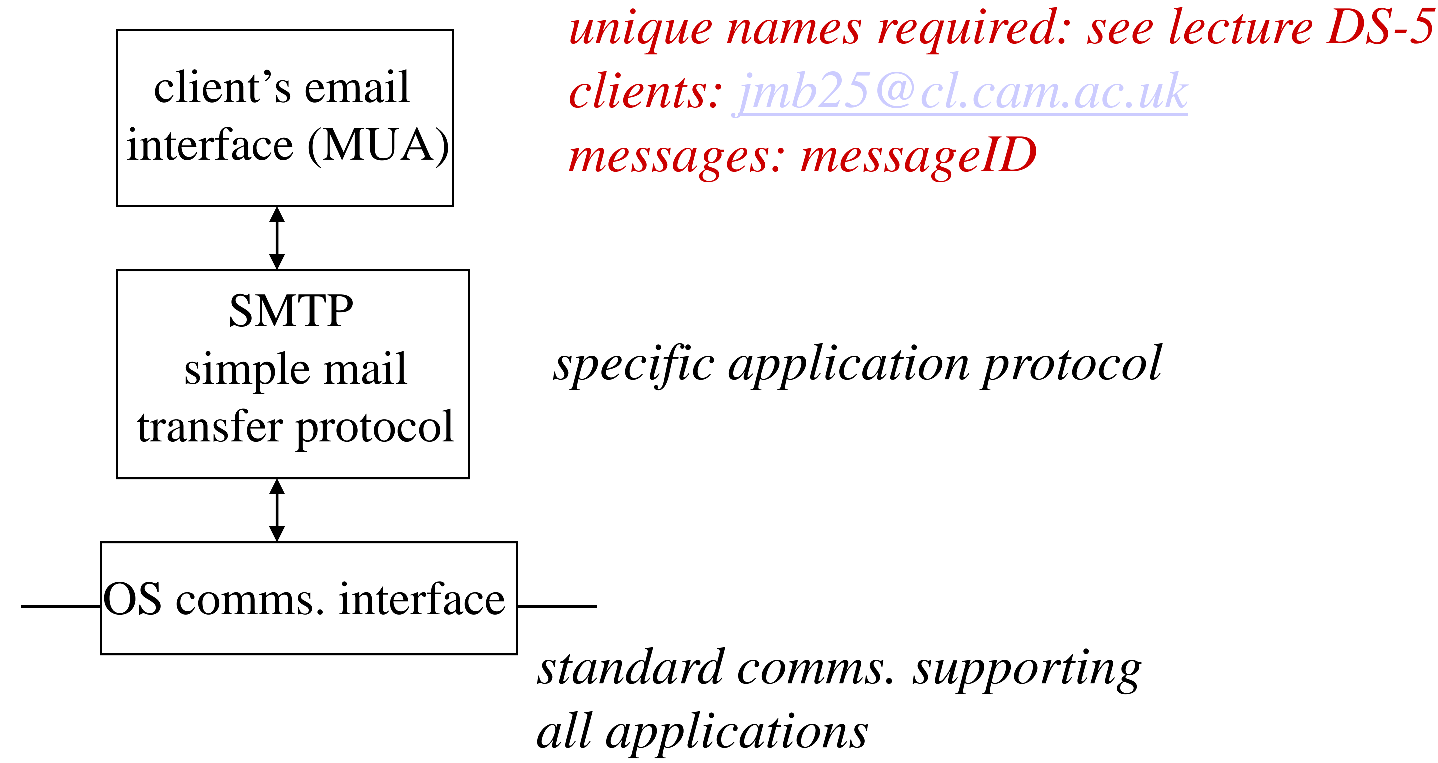
Support for distributed software may be:

special case: directly by OS in a *homogeneous* cluster (distributed OS design) – not the focus of this course

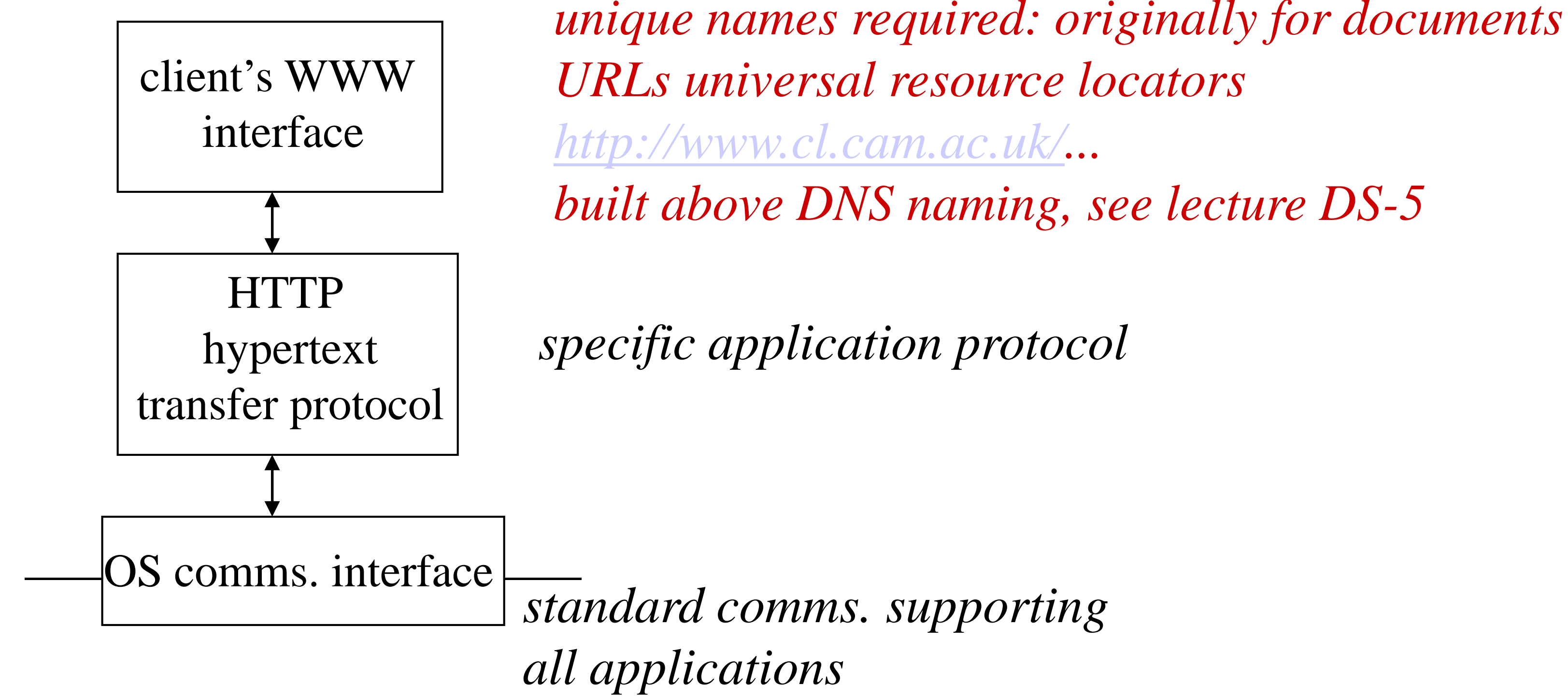
general case: by a software layer (middleware) above potentially heterogeneous OS



Distributed application structure – email, news, ftp



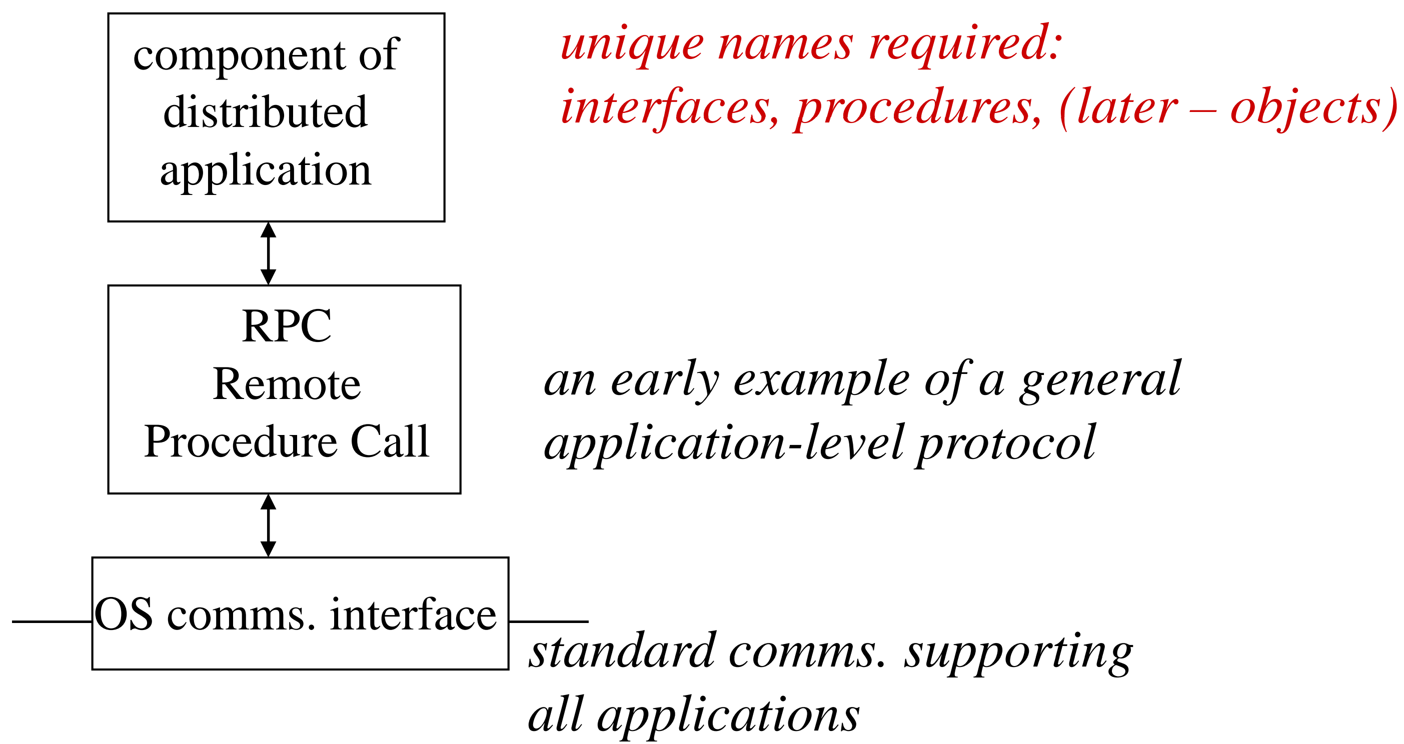
Distributed application structure – web documents



A browser interface came to be used for **general** distributed applications

W3C standards for Web Services – see lectures DS-4, DS-7

Distributed application structure – general support example



RPC is an early example of a protocol above which distributed applications may be developed. RPC examples: ISO-ODP, OSF-DCE

A middleware also includes services above the RPC layer

Open and proprietary middleware

- Open: evolution is controlled by standards bodies (e.g. ISO, NIST) or consortia (e.g. OMG, W3C). Requests for proposals (RFPs) are issued, draft specifications published with RFCs (requests for comments). Compromise is common.
- Closed, proprietary: can be changed by the owner (clients need to buy a new release). Consistency across versions is not guaranteed. Plus *"embrace, enhance, extinguish"*.

Related issues:

- single/multi language: can components be written in different languages and interoperate?
- open interoperability: is desirable across middlewares (including different implementations of the same MW)

DS Design: model, architecture, engineering

Programming model of distributed computation:

- What are the named entities? objects, components, services,..

- How is communication achieved?
  - synchronous/blocking (request-response) invocation
    e.g. client-server model
  - asynchronous messages e.g. event notification model
  - one-to-one, one-to-many?

- Are the communicating entities closely or loosely coupled?
  - must they share a programming context?
  - must they be running at the same time?

System architecture: the framework within which the entities in the model interoperate

- Naming
- Location of named objects
- Security of communication, as required by applications
- Authentication of participants
- Access control / authorisation
- Replication to meet requirements for reliability, availability

May be defined within *administration domains*.

Need to consider *multi-domain systems* and interoperation within and between domains

System engineering: implementation decisions

- Placement of functionality: client libraries, user agents, servers, wrappers/interception
- Replication for failure tolerance, performance, load balancing –> consistency issues
- Optimisations e.g. caching, batching
- Selection of standards e.g. XML, X.509
- What *"transparencies"* to provide at what level:

  (transparent = hidden from application developer: needn't be programmed for, can't be detected when running).

  distribution transparency: location? failure? migration?

  may not be achievable or may be too costly

Architectures for Large-Scale, Networked Systems

Individual user using globally available service

Single administration domain

Federated administration domains

Independent, external services - to be integrated

Detached, ad-hoc, anonymous groups;
anonymous principals, issues of trust and risk

Federated administration domains: Examples

- national healthcare services:
  many hospitals, clinics, primary care practices.
- national police services:
  43 county police forces (52 with Scotland),
- global company:
  branches in London, Tokyo, New York, Berlin, Paris ..
- transport
  County Councils responsible for cities, some roads
- active city:
  fire, police, ambulance, healthcare services.
  mobile workers
  sensor networks e.g. for traffic/pollution monitoring

Federated domains - characteristics

- names: administered per domain (users, roles, services, data-types, messages, sensors, ….)
- authentication: users administered within a domain
- communication: needed *within* and *between* domains
- security: per-domain firewall-protection
- policies: specified per domain e.g. for communication, access control *intra and inter-domain*, plus some external policies to satisfy government, legal and institutional requirements
- high trust and accountability within a domain,

  known trust between domains

Independent, External Services - Examples

- commercial web-based services
  e.g. online banking, airline booking etc.

- national services used by police and others
  e.g. DVLA, court-case workflow

- national health services
  e.g. national Electronic Health Record (EHR) service

- e-science (grid) databases and generic services
  e.g. astronomical, transport, medical *databases*
  for *computation* (e.g. XenoServices), or *storage*

- e-science may support "virtual organisations" –
  collaborating groups across several domains

Independent, external services - characteristics

- naming and authentication

  may be of individuals via trusted third parties (TTPs)

  and/or via home domain of client

- access control policies

  related to client roles in domains and/or individuals

  support for *"virtual organisations"* spanning domains

- need for: accounting, charging, audit

  these may be done by trusted third parties

  a basis for mutual trust (service done, client paid)

- trust

  based on evidence of behaviour

  clients exchange experiences, services monitor and record

  assume full connectivity, e.g. TTPs can authenticate/identify

Examples of detached ad hoc groups and the need for trust

- Commuters regularly play cards on the train
- Auctions – build up trust of an ID through small honoured purchases, then default on a big one
- E-purse purchases – trust in system
- Recommendations: e.g. in a tourist scenario - restaurants, places to visit. Recommendations of people and their skills.
- Wireless routing via peers:

  routing of messages P2P rather than by dedicated brokers – reliability, confidentiality, altruism
- Trust has a context – skills may not transfer

  e.g. drivers of cars, trains, planes ...

3. Detached, ad-hoc, anonymous groups

- e.g. connected by wireless
- can’t assume trusted third-parties (CAs) accessible
- can’t assume knowledge of names and roles, identity likely to be by key/pseudonym
- new identities can be generated (by detected villains)

- parties need to decide whether to interact
- each has a trust policy and a trust engine
- each computes whether to proceed – policy is based on:

- accumulated trust information
  (from recommendations and evidence from monitoring)
- risk (resource-cost) and likelihood of possible outcomes

Promising Approaches for Large-Scale Systems

- Roles for scalability
- Parametrised roles for expressiveness, scalability, simplicity
- RBAC for services, service-managed objects, including the communication service
- Policy specification and change management
- Policy-driven system management

- Asynchronous, loosely-coupled communication
  publish/subscribe for scalability
  event-driven paradigm for ubiquitous computing
- Database integration – how best to achieve it?

And don't forget:

- Mobile users
- Sensor network integration

Opera Group – research themes

(**objects policy events roles access control**)

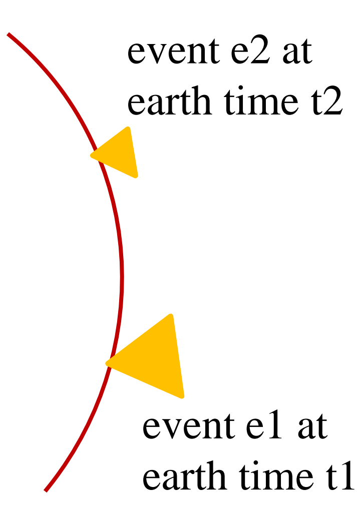
- Access Control (OASIS RBAC)
  Open Architecture for Securely Interworking Services
- Policy expression and management
- Event-driven systems (CEA, Hermes)
  EDSAC21: event-driven, secure application control for the 21st Century
- Trust and risk in global computing (EU SECURE)
  secure **c**ollaboration among **u**biquitous **r**oaming **e**ntities
- TIME: Traffic Information Monitoring Environment
  TIME-EACM event architecture and context management
- CareGrid: dynamic trust domains for healthcare applications
- SmartFlow: Extendable event-based middleware
- PAL: personal and social communications services for health and lifestyle monitoring.

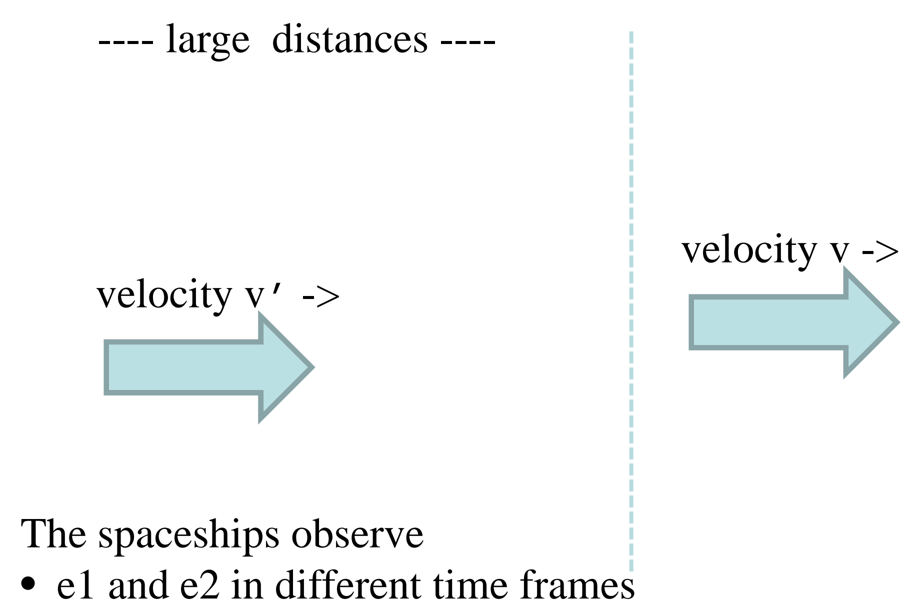
see: *www.cl.cam.ac.uk/research/srg/opera*

for people, projects, publications for download

Time in Distributed Systems

There is no common universal time (Einstein) but the speed of light is constant for all observers irrespective of their velocity

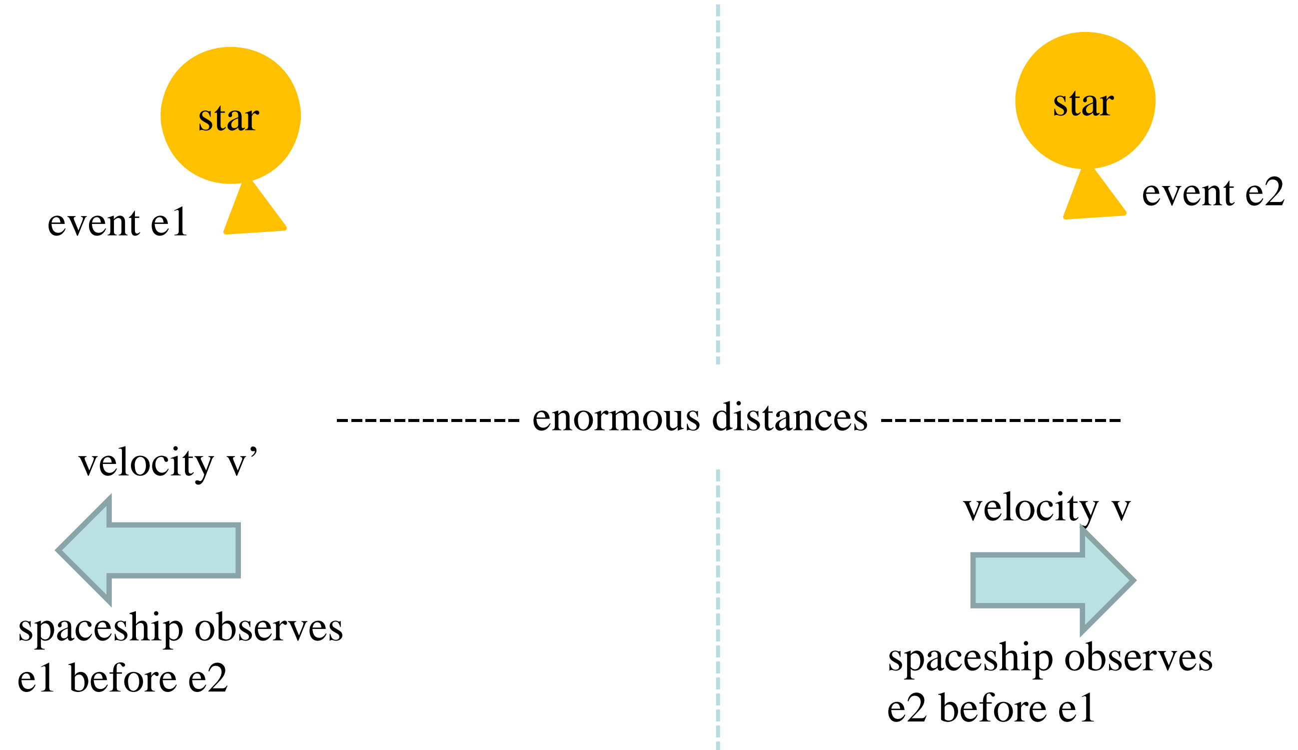




The spaceships observe

- e1 and e2 in different time frames
- different values for time elapsed between e1 and e2
- e1 and e2 (from the same source) in the same order.

Event ordering in space



Time in Distributed Systems

Assume our distributed system is earth-based

Earth time is defined w.r.t. the earth's rotation

- solar year is constant
- solar day is lengthening (earth slowing)

From 1948, earth time has been based on atomically-defined caesium clocks
(atomic second = solar second)

There are now about 50 such clocks, average value = TIA (International Atomic Time)

BIH (Intn'l Bureau de l'Heure) announces leap seconds to keep in phase with the sun
---- about 30 so far, most recently Jan 1999, Jan 2006

Time in Distributed Systems - UTC

UTC (Universal Coordinated Time) is corrected TIA

UTC services are offered by radio stations and satellites
– receivers are available commercially
Accuracy varies with weather conditions
– stated bounds are 1ms – 10ms
radio 10ms
UK: Rugby since 1927, Anthorn Cumbria 2007,
US: Fort Collins Colorado
satellite: GOES 0.5ms, GPS 1ms

UTC signals take time to propagate – UTC can't be known exactly
For a given receiver we can estimate a time interval during which an event has happened w.r.t. UTC, see later "interval timestamps"

Timers in computers

Based on frequency of oscillation of a quartz crystal

Each computer has a timer that interrupts periodically
Clock drift: in practice, the number of interrupts per hour varies slightly
in the fabricated devices, also with temperature, so clocks may drift,
typically $1/10^6$ (1 sec in 11.6 days)

Timers can be set from transmitted UTC
We have already seen that we cannot know accurately the time at which an event occurs,
but can only specify an interval
We now have to increase that interval to allow for clock drift
as well as other sources of inaccuracy

Note that computer systems tag events with timestamps, usually a local clock reading.
Preferably, interval timestamps should be used.

Does accurate time matter?

Important questions:
How accurate does time need to be?
How is time used in a distributed system?
What does "A happened before B" mean in a distributed system?

Sometimes we CAN'T SAY in which order two events occurred:

- if the events have point timestamps that differ by less than some value*
- if the events have interval timestamps, and the intervals overlap

*we prefer intervals because for point timestamps we need to know the characteristics of the originator in order to determine the tolerance

Use of time in distributed systems: examples 1

1. Any source of resource contention e.g. Airline booking

   POLICY: if the reservation requests of two transactions may each be satisfied separately but there are not enough seats left for both, then the transaction with the earlier timestamp wins

   Note that no causality is involved, the requests are independent.
   We don't need accurate time but just an ordering convention so all agree who won.

   On a tie (equal timestamps) use an agreed tie-breaker
   e.g. IP address / processID

Use of time in distributed systems: examples 2

2 Programming environments e.g. UNIX ***make*** (compile and link)

Suppose a ***make*** involves many components that are edited on distributed computers.
Suppose a component is edited immediately after a ***make***,
but on a computer with a slow clock
so that the recorded time of the edit is before the recorded time of the ***make***.
On the next ***make*** this component is not recompiled.

This can be made unlikely to happen, if we ensure that clocks are initialised accurately e.g. not from the operator's watch, but from a "time server" – see below.

This is an example of correctness depending on correct event ordering:
did the edit take place before or after the last ***make****?*

of course – it's a bad idea to use a timestamp as a version number in a distributed system. ***make*** was designed for a centralised UNIX system.

Use of time in distributed systems: examples 3

3. Did a credit/debit transaction take place before or after midnight?
   This affects daily calculation of interest.

4. The value of shares at the time of buying/selling.

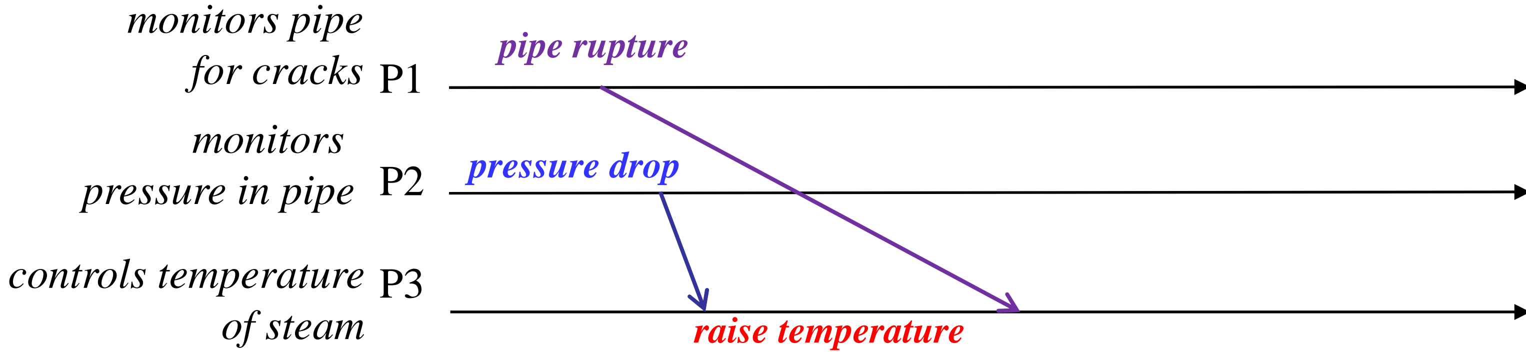
5. Insider dealing? Did X read Y before buying/selling?

Note that some of the above examples require only a means of agreement, so that all participants in the algorithm make the same decision.

Others require accurate time, or the order of events in the real world, when causality is at issue.

Physical causality in the environment

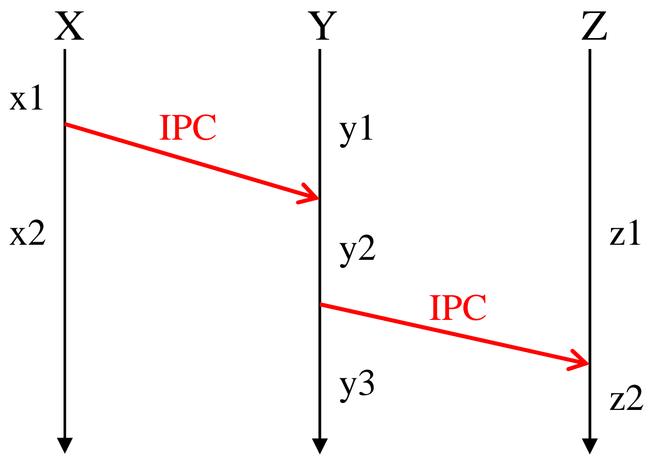
Causality may be absolute and physical – outside the scope of the message transport service



1. The pipe ruptures which causes a drop in pressure
2. P1 send a message to controller P3 to notify rupture
3. P2 sends a message to controller P3 to notify pressure drop
4. P3 receives P2's message (before P1's) and increases temperature
5. P3 receives P1s message .....
6. AUDIT may infer (wrongly) that temperature increase caused the pipe to rupture

The controller's algorithm must take delay and physical timestamps into account
AUDIT of system failure may have to report "***can't say***" for close timestamps

Event ordering in distributed systems



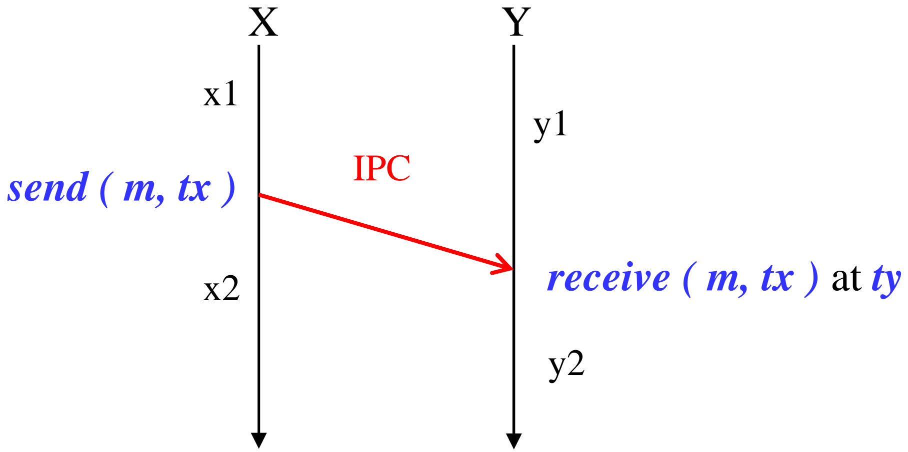
Define < to mean "happened before"
Events in a single system are assumed to be ordered

IPC: ***send*** is before ***receive,*** this is TRUE *whatever the local clocks of X, Y and Z indicate*
IPC imposes a partial order on events:

events in region x1 < events in regions y2 and y3
events in region x1 < events in region z2
events in regions y1 and y2 < events in region z2
for events in other regions we can't say what the order is
unless we know the precise accuracy of all physical clock values

Time

Local clocks must respect true event orderings



Note that X's ***send*** caused Y's receive

Suppose Y's local clock reads ***ty*** on ***receive (m, tx)***

if ***ty*** > ***tx*** OK

if ***ty*** < = ***tx*** reset ***ty*** to ***tx*** plus one increment

This imposes logical time on the system

BUT system time adjusted in this way will drift ahead of UTC

- could use counters rather than timestamps if all we need is event ordering
- so-called "Lamport Time"

How can we generate timestamps that are reasonably close to UTC and preserve causal ordering?

Protocols for synchronizing physical clocks - 1

Cristian's algorithm 1989

- Assume one computer has a UTC receiver (call it a time server)
- Each computer polls the time server periodically
  (period depends on maximum clock drift and accuracy required).
- Server sends back its value of the time
- Client receives this value and may: use it as it is,
  add the known minimum network delay,
  add half the round trip time for this request/response

Client/receiver resets its clock from this value T:

if T > local time
use it to set the clock, or adjust the interrupt rate for a while to speed up the clock
e.g. 10ms -> 9ms

if T < local time
time cannot be put back or event ordering within the local system would be violated
so adjust the interrupt rate to slow down the clock e.g. 10ms –> 11ms

Protocols for synchronizing physical clocks - 2

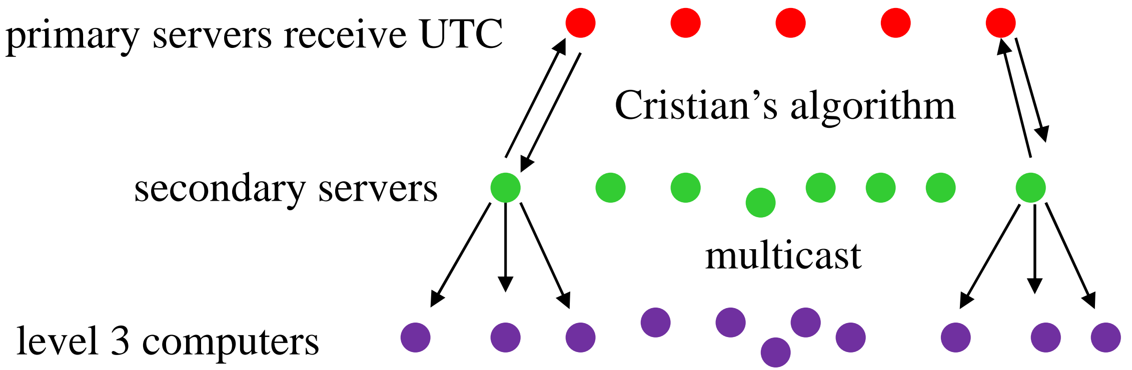
In the above, the time server is a single point of failure.

A number of time servers can be used to increase reliability
each computer multicasts to all time servers,
takes the average of the returned values then proceeds as above.

If there is no time server,
a nominated component can multicast to all, requesting their time
then multicast the average value to all (*Berkeley UNIX 1989*).

NTP Network Time Protocol

For the Internet as a hierarchy of computers:



- uses UDP
- allow for network delay and adjust clocks as described for Cristian's algorithm
- accurate to a few tens of milliseconds

Time servers also exist as web servers for explicit query from individual computers

Point timestamps and interval timestamps

For any computer we can *estimate* how long UTC takes to reach it, taking into account:
- atmospheric pressure
- network(s) transmission time
- software overhead e.g. in local OS

To tag a message with a timestamp:
The local clock reading could be used as a point timestamp and a tolerance could be estimated.
Note that this should be *source-specific* – hardly ever taken into account.

An interval timestamp, in which the UTC is estimated to lie, captures the uncertainty over measuring time, taking into account local conditions
i.e. interval width should be source-dependent.

Use of point and interval timestamps

If events are to be ordered,
Point timestamps closer than their associated tolerances and overlapping interval timestamps indicate that this cannot be done reliably.

The application may be told that a strong ordering is impossible (CAN'T SAY).
A weak ordering may be formed on the basis of e.g. the point timestamps taken literally, or the upper interval bounds, but it should be made explicit to the application that this is not correct/reliable
e.g. it cannot be used as an audit of the possibility or otherwise of cause and effect.

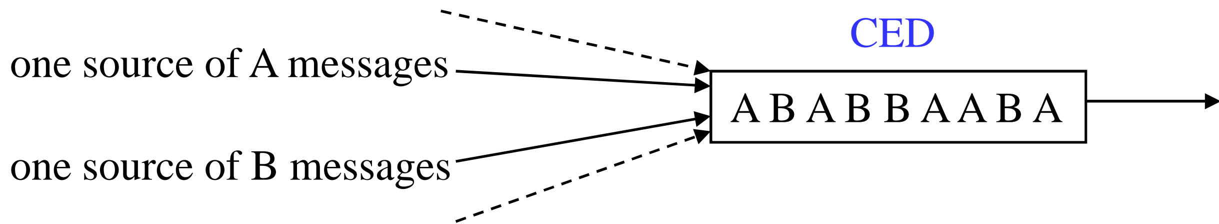
This is the nature of distributed systems – we have to live with it.
Ref: Fundamental Properties, introduction slide 13

Applications that abstract above distributed time should be aware that they are doing this
e.g. arrival time of a request at a server may be used to order requests.
Source timestamps may indicate a different order or may be indeterminate.
Database and stream processing applications tend to use arrival time at the server.

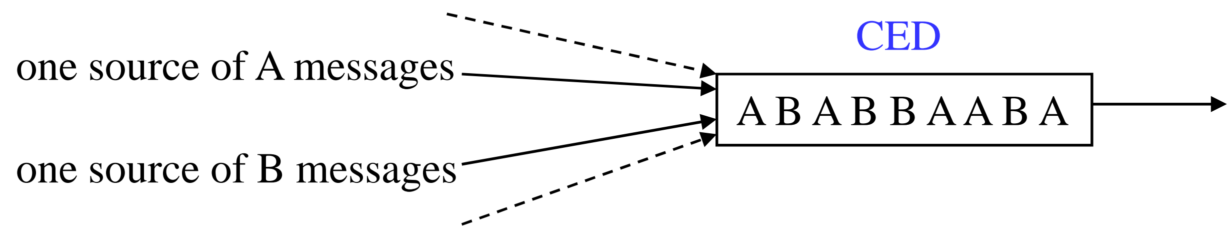
Composition of events (sent as messages)

Applications are often interested in patterns of events, perhaps discovered through data mining
- fraud detection
- fault detection
- raising alarms – medical, environmental, ...
- controlling the volume of events propagated, e.g. from sensors, from faulty components

A Composite Event Detector (CED) receives streams of events from distributed sources and notifies a stream of composite events. An example showing two event types A and B:



Composition of events – composition algebra



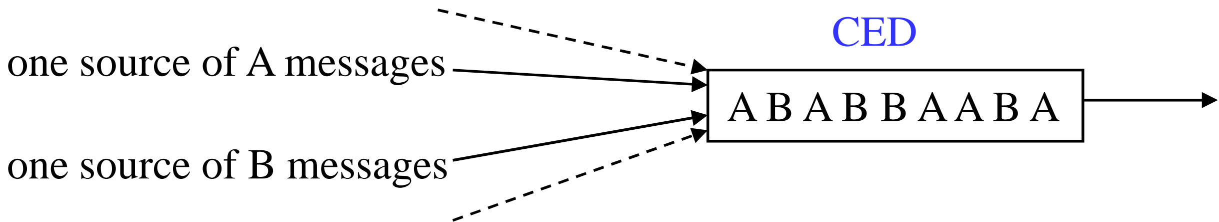
An event algebra defines composition operators: e.g.

AND, OR, SEQ (before/after),
UNTIL (stream with a terminator),
AFTER (stream with a starter),
NOT? (difficult to decide)

Recall fundamental uncertainty over time if event ordering (SEQ, AFTER, UNTIL) is offered.
perhaps offer choice to application of strong and weak ordering, or tag whether strong or weak

Timestamp of composite event?
– the interval spanning all component events (easy/natural with interval timestamps on events)
– or the timestamp of the latest component event? – when did the CE complete?

CED engineering issues



Engineering issues:

- are all the event sources registered with the CED, and the connections to them, operational?
  use a heartbeat protocol with each source
  should processing be delayed if lack of a heartbeat indicates an event may have been delayed ?
  the NOT operator makes this problem explicit

- buffer size and garbage collection?

- consumption policy (in this example, which As with which Bs?) *historical? most recent?*

A CED may take as input primitive and/or composite events
CED components (subtrees) may be distributed
e.g. placed close to event sources, optimising communication

Process groups and message ordering

If processes belong to groups, certain algorithms can be used that depend on group properties

- membership
  create (name), kill (name)
  join (name, process), leave (name, process)

- internal structure?
  NO (peer structure) – failure tolerant, complex protocols
  YES (a single coordinator and point of failure) – simpler protocols
  e.g. all join requests must go to the coordinator – concurrent joins avoided

- closed or open?
  OPEN – a non-member can send a message to the group
  CLOSED – only members can send to the group

- failures?
  a failed process leaves the group without executing leave

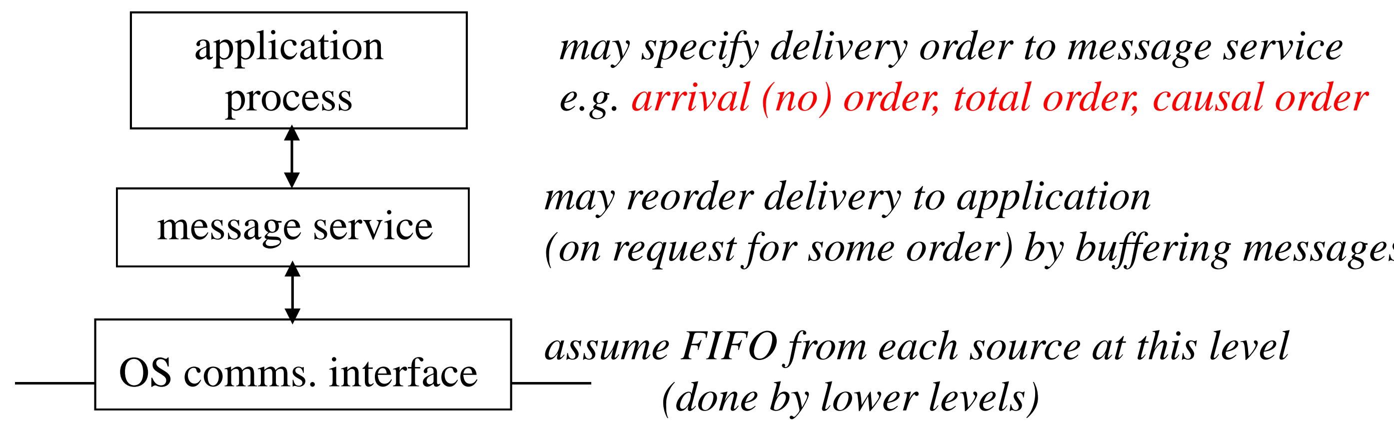
- robustness
  leave, join and failures happen during normal operation – algorithms must be robust

Message delivery for a process group - assumptions

ASSUMPTIONS

- messages are multicast to named process groups
- reliable channels: a given message is delivered reliably to all members of the group
- FIFO from a given source to a given destination
- processes don't crash (failure and restart not considered)
- processes behave as specified e.g. send the same values to all processes
  - we are not considering so-called Byzantine behaviour
    (when malicious or erroneous processes do not behave according to their specifications
    see Lamport's Byzantine Generals problem).

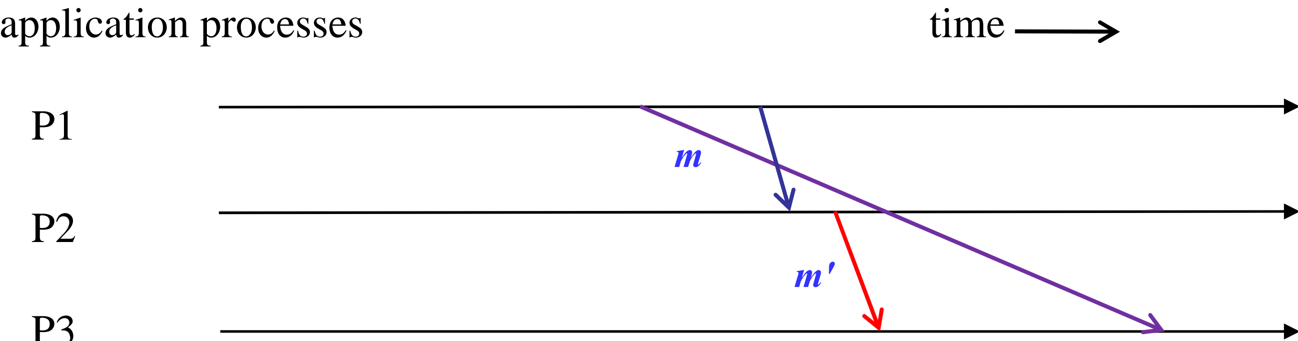
Ordering message delivery



total order = every process receives all messages in the same order (including its own).
We first consider *causal order*

Message delivery – causal order

First, define causal order in terms of one-to-one messages; later, multicast to a process group



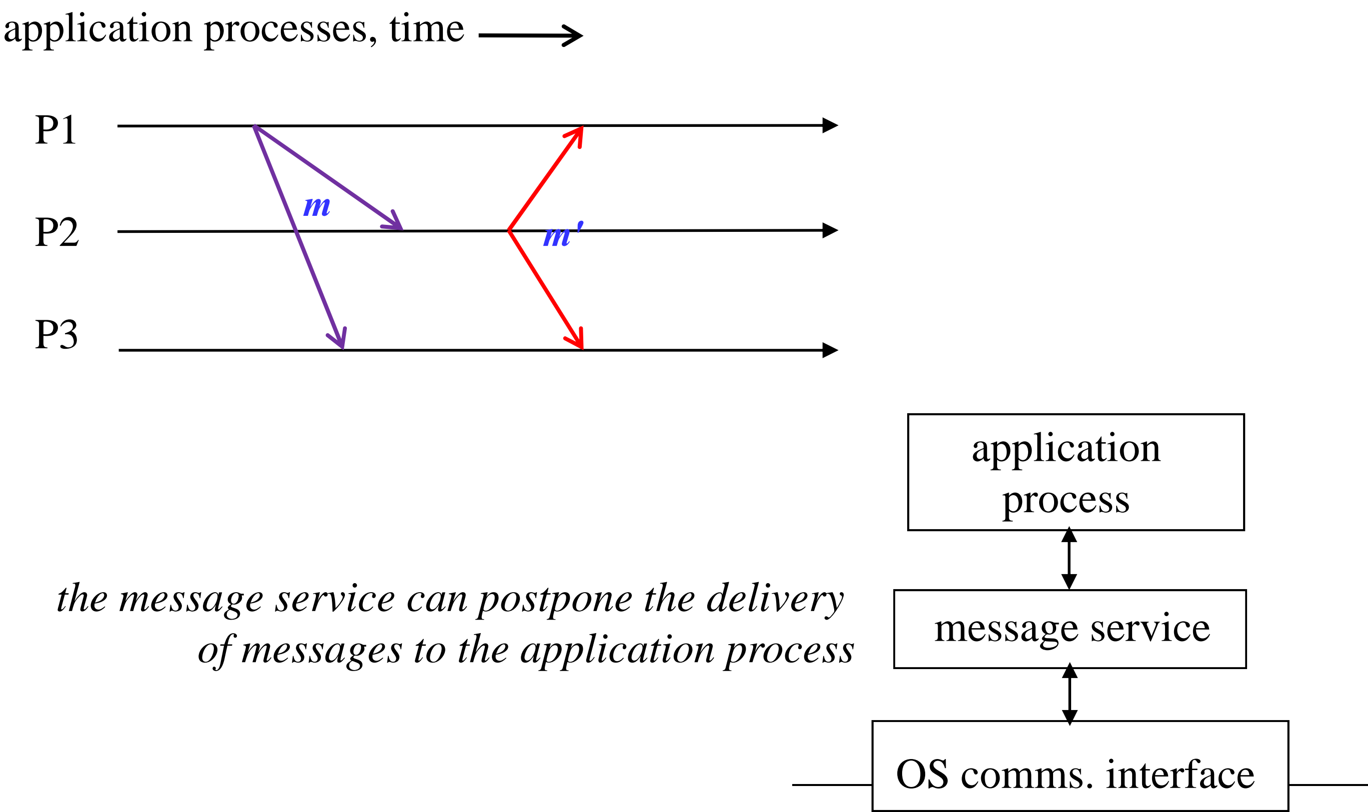
P1 sent message ***m*** *provably before* P2 sent message ***m'***
The above diagram shows a violation of causal delivery order
Causal delivery order requires that, at P3, ***m*** is delivered before ***m'***
The definition relates to POTENTIAL causality, not application semantics

DEFINITION of causal delivery order (where < means "happened before")

$$send_i (m) < send_j (m') \Rightarrow deliver_k (m) < deliver_k (m')$$

Message delivery – causal order for a process group

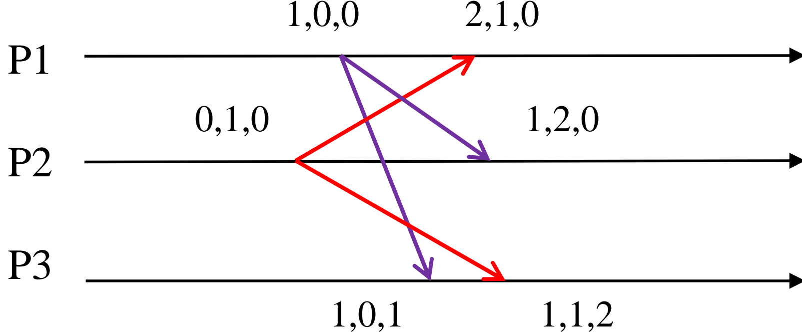
If we know that all processes in a group receive all messages, the message delivery service can implement causal delivery order (for total order, see later)



Message delivery – causal order using Vector Clocks

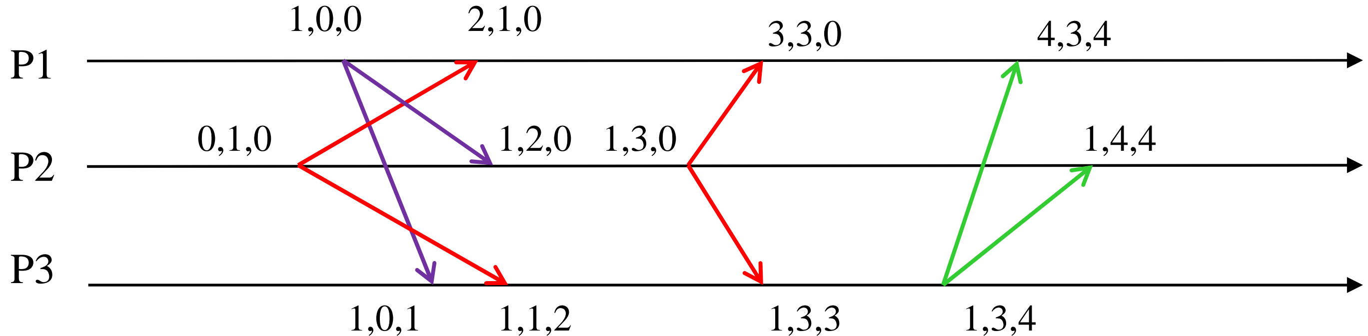
A vector clock is maintained by the message service at each node for each process:

application processes



vector notation:
- fixed number of processes N
- each process's message service keeps a vector of dimension N
- for each process, each entry records the most up-to-date value of the state counter delivered to the application process, for the process at that position

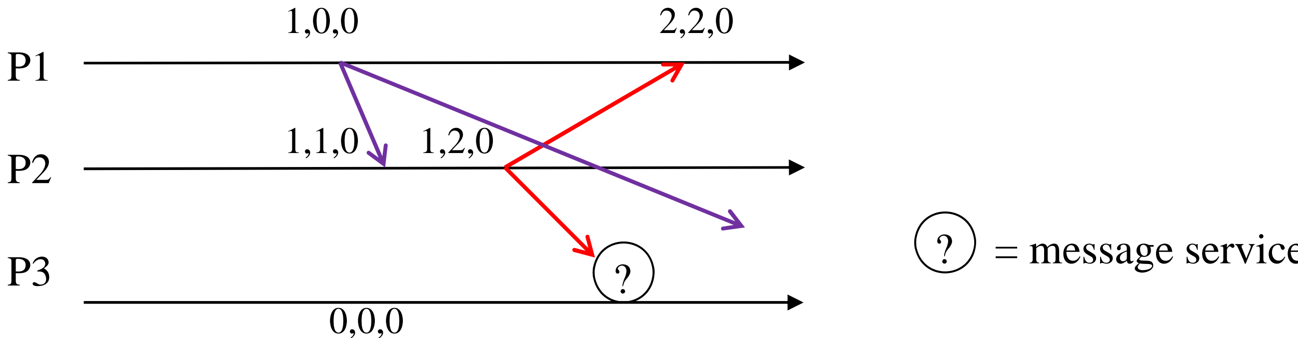
Vector Clocks – message service operation



Message service operation:

- before ***send*** increment local process state value in local vector
- on ***send***, timestamp message with sending process's local vector
- on ***receive*** *by message service* – see below
- on ***deliver*** *to receiving application process*, increment receiving process's state value in its local vector and update the other fields of the vector by comparing its values with the incoming vector (timestamp) and recording the higher value in each field, thus updating this process's knowledge of system state

Implementing causal order using Vector Clocks



P3's vector is at (0,0,0) and a message with timestamp (1,2,0) arrives from P2
i.e. P2 has received a message from P1 that P3 hasn't seen.

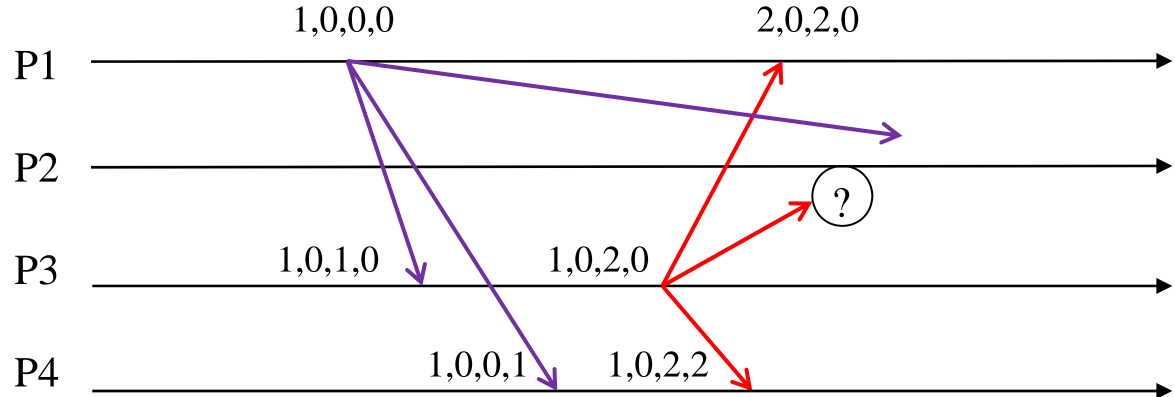
More detail of P3's message service:

receiver vector	sender	sender vector	decision	new receiver vector
0,0,0	P2	1,2,0	buffer	0,0,0
	P3 *is missing a message from* P1 *that sender* P2 *has already received*			
0,0,0	P1	1,0,0	deliver	1,0,1
1,0,1	P2	1,2,0	deliver	1,2,2

In each case: do the *sender and receiver agree on the state of all other processes*?
If the sender has a higher state value for any of these others, the receiver is missing a message, so buffer the message

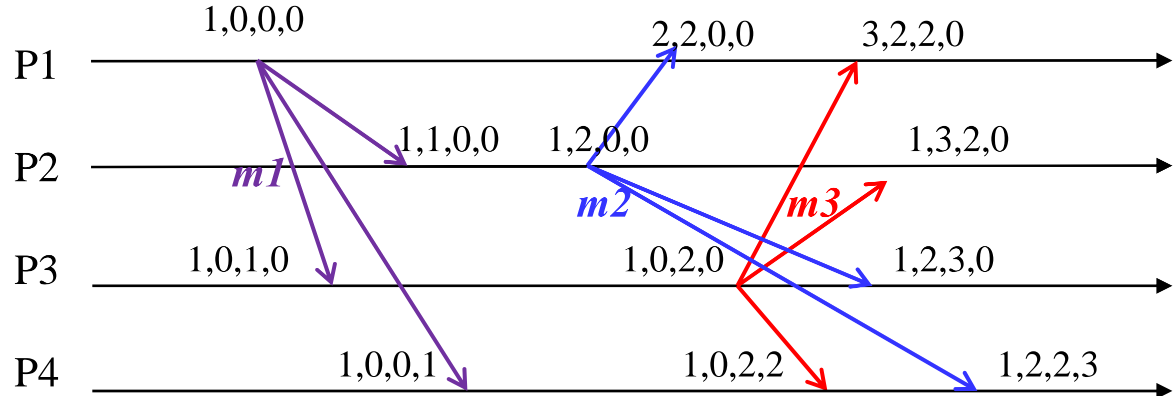
Vector Clocks - example

application processes



sender	sender vector	receiver	receiver vector	decision	new receiver vector
P3	1,0,2,0	P1	1,0,0,0	deliver	P1 -> 2,0,2,0
	same states for P2 *and* P4				
P3	1,0,2,0	P4	1,0,0,1	deliver	P4 -> 1,0,2,2
	same states for P1 *and* P2				
P3	1,0,2,0	P2	0,0,0,0	buffer	P2 -> 0,0,0,0
	same state for P4, *different for* P1- *missing message*				
P1	1,0,0,0	P2	0,0,0,0	deliver	P2 -> 1,1,0,0
	reconsider buffered message:				
P3	1,0,2,0	P2	1,1,0,0	deliver	P2 -> 1,2,2,0
	same states for P1 *and* P4				

Total order is not enforced by the vector clocks algorithm



m2 and ***m3*** are not causally related

P1 receives ***m1***, ***m2***, ***m3***
P2 receives ***m1***, ***m2***, ***m3***
P3 receives ***m1***, ***m3***, ***m2***
P4 receives ***m1***, ***m3***, ***m2***

If the application requires total order this could be enforced by modifying the vector clock algorithm to include ACKs and delivery to self.

Totally ordered multicast

The vectors can be a large overhead on message transmission and a simpler algorithm can be used if only total order is required.

Recall the ASSUMPTIONS

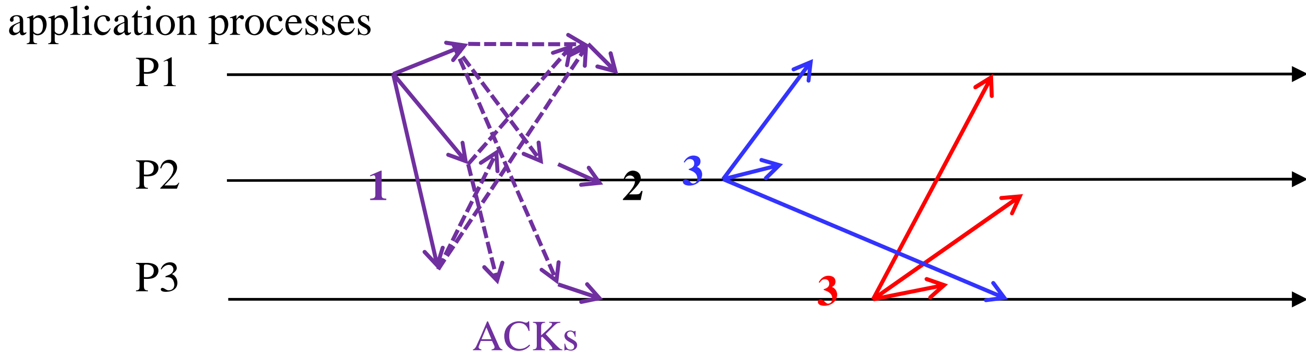
- messages are multicast to named process groups
- reliable channels: a given message is delivered reliably to all members of the group
- FIFO from a given source to a given destination
- processes don't crash (failure and restart not considered)
- no Byzantine behaviour

total order algorithm

- sender multicasts to all including itself
- all acknowledge receipt as a multicast message
- message is delivered in timestamp order after all ACKs have been received

If the delivery system must support both, so that applications can choose, vector clocks can achieve both causal and total ordering.

Total ordered multicast – outline of approach



P1 increments its clock to 1 and multicasts a message with timestamp 1
All delivery systems collect the message, multicast ACK and collect all ACKs
 - no contention – deliver message to application processes
and increment local clocks to 2.

P2 and P3 both multicast messages with timestamp 3
All delivery systems collect messages, multicast ACKs and collect ACKs.
 - contention – so use a tie-breaker (e.g. lowest process ID wins)
and deliver P2s message before P3s

This is just a sketch of an approach. In practice, timeouts would have to be used to take account of long delays due to congestion and/or failure of components and/or communication links

Consistency in Distributed Systems

Recall the fundamental DS properties – DS may be large in scale and widely distributed

1. concurrent execution of components
2. independent failure modes
3. transmission delay
4. no global time

Consistency is an issue for both:

- replicated objects
- transactions involving related updates to different objects (recall ACID properties)

We first study replication and later distributed transactions

Objects may be replicated for a number of reasons:

- reliability/availability - to avoid a single point-of-failure
- performance - to avoid overload of a single "bottleneck"
- to give fast access to local copies – to avoid communications delays and failures

Examples of replicated objects:

Naming data

for name-to-location mapping, name-to-attribute mapping in general

Web pages

mirror sites world-wide of heavily used sites

Maintaining Consistency of Replicas

Weak consistency – for when the "fast access" requirement dominates.

- update some replica, e.g. the closest or some designated replica
- the updated replica sends update messages to all other replicas.
- different replicas can return different values for the queried attribute of the object
  the value should be returned, or "not known", with a timestamp
- in the long term all updates must propagate to all replicas ....
  - consider failure and restart procedures,
  - consider order of arrival,
  - consider possible conflicting updates

Strong consistency – ensures that only consistent state can be seen.

All replicas return the same value when queried for the attribute of an object.

This may be achieved at a cost – high latency.

Engineering weak consistency of replicas

- Simple approach
  all updates are made to a PRIMARY COPY
  the primary copy propagates updates to a number of backup copies
  note that updates have been *serialised* through the primary copy
  queries can be made to any copy
  the primary copy can be queried for the most up-to-date value
  example: DNS domains have a distinguished name server per domain plus a few replicas
  performance?
  for other than small-scale systems, the primary copy will become a bottleneck
  and update access will be slow - to the location of the primary copy from everywhere.
  So have different primary copy sites for different data items.
  reliability? availability?
  the primary copy could fail!
  have a HOT STANDBY to which updates are made synchronously with the primary copy

- General, scalable approach:
  Distribute a number of first-class replicas.
  We now have to be aware that *concurrent updates and queries* can be made.

Weak consistency of replicas - issues

The system must converge to a consistent state as the updates propagate.

Consider DS properties:
1. concurrent execution of components
2. independent failure modes
3. transmission delay
4. no global time

1 and 3: concurrent updates and communications delay

- the updates do not, in general, reach all replicas in the same (total) order
- the order of *conflicting* updates matters
- conflicts must be resolved, semantics are application-specific, see also below re. timestamps

2: failures of replicas

Restart procedures must be in place to query for missed updates

....

Weak consistency of replicas – issues (contd.)

The system must converge to a consistent state as the updates propagate.

Consider DS properties:

1. concurrent execution of components
2. independent failure modes
3. transmission delay
4. no global time

....

4: no global time – are clocks synchronised?
but we need at least an ordering convention for arbitrating between conflicting updates
- e.g. conflicting values for the same named entry – change of password or privileges
- e.g. add/remove item from list – DL, ACL, hot list
- e.g. tracking a moving object – times must make physical sense
- e.g. processing an audit log – times must reflect physical causality

In practice, systems will not rely solely on message propagation but also compare state from time to time, e.g. Name servers – Grapevine, GNS, DNS

Further reading:
Y Saito and M Shapiro “Optimistic replication” ACM Computing Surveys 37(1) pp.42-81, March 2005

Strong Consistency of Replicas (and in Transactions)

Transactional semantics: ACID properties (Atomicity, Consistency, Isolation, Durability)

start transaction

make the same update to all *replicas*

or make *related updates* to a number of *different objects*

end transaction

(either: ***commit*** – all changes are made, are visible and persist

or: ***abort*** – no changes are made to any object)

First consider implementation of strongly consistent replication

See later for distributed transactions.

First thoughts – ***update***: lock all objects, make update, unlock all objects ?

query: read from any replica

Strong Consistency of **Replicas**

Problems with locking all objects to make an update:

- Some replicas may be at the end of slow communication lines
- Some replicas may fail, or be slow or overloaded
- So: Lack of availability of the system (a reason for replication)
  i.e. delay in responding to queries.
  This is because of the slowness of the update protocol due to
  communications failures or delays,
  replica failure or delays
- Intolerable if no-one can update or query
  because one (distant, difficult-to-access) replica fails

So we try a majority voting scheme - QUORUM ASSEMBLY

A solution for strong consistency of **replicas**.

Quorum Assembly for replicas

Assume n copies. Define a read quorum QR and a write quorum QW,
Where QR must be locked for reading and QW must be locked for writing, such that:

$QW > n/2$

$QW + QR > n$

These ensure that

only one write quorum can successfully be assembled at any time ($QW > n/2$)

every QW and QR contain at least one up-to-date replica ($QW + QR > n$)

After assembling a (write) quorum QW, bring all replicas up-to-date then make the update.

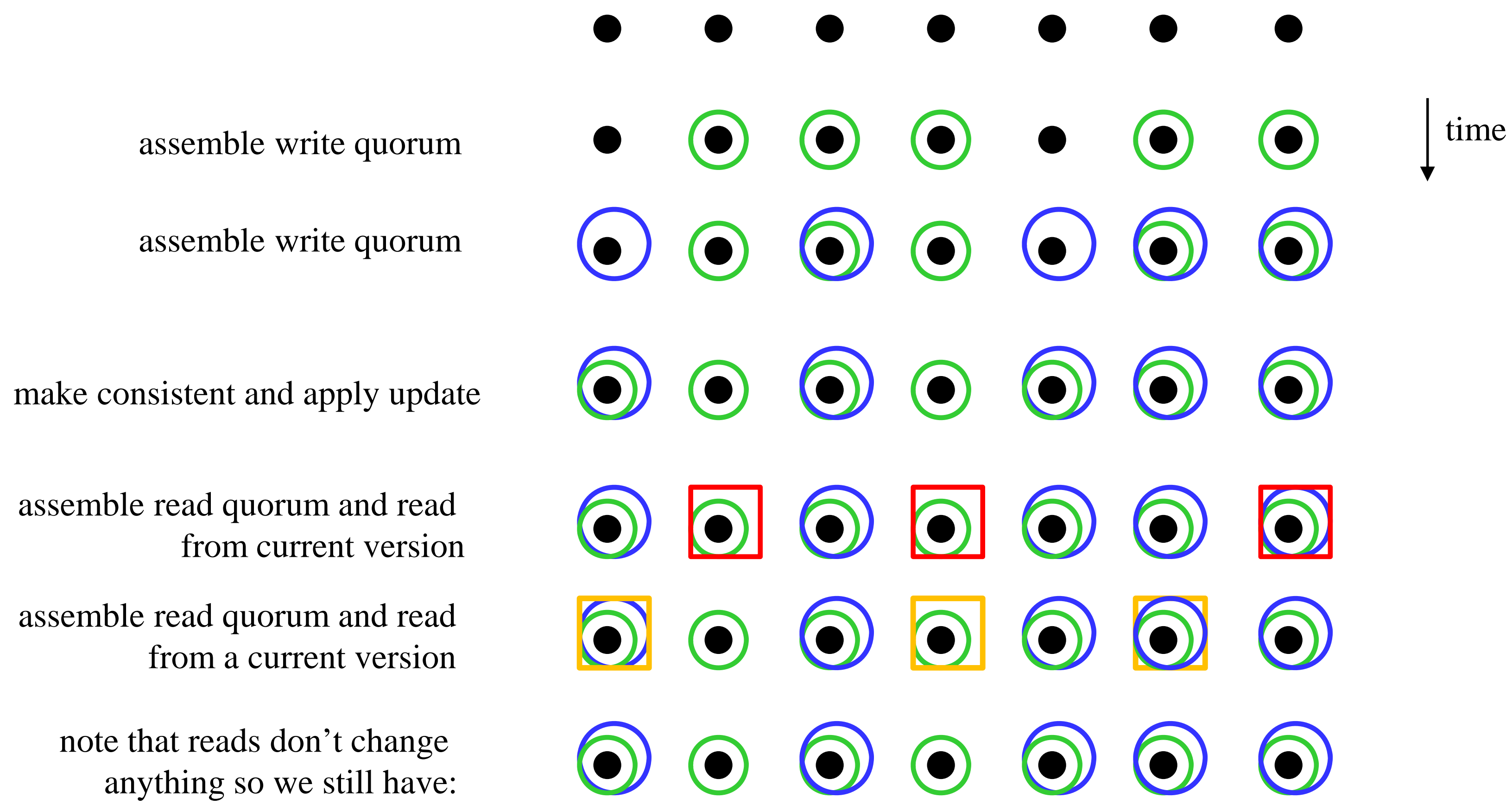
e.g. QW = n, QR = 1 is lock all copies for writing, read from any

e.g. n = 7, QW = 5 ○ QR = 3 □

time

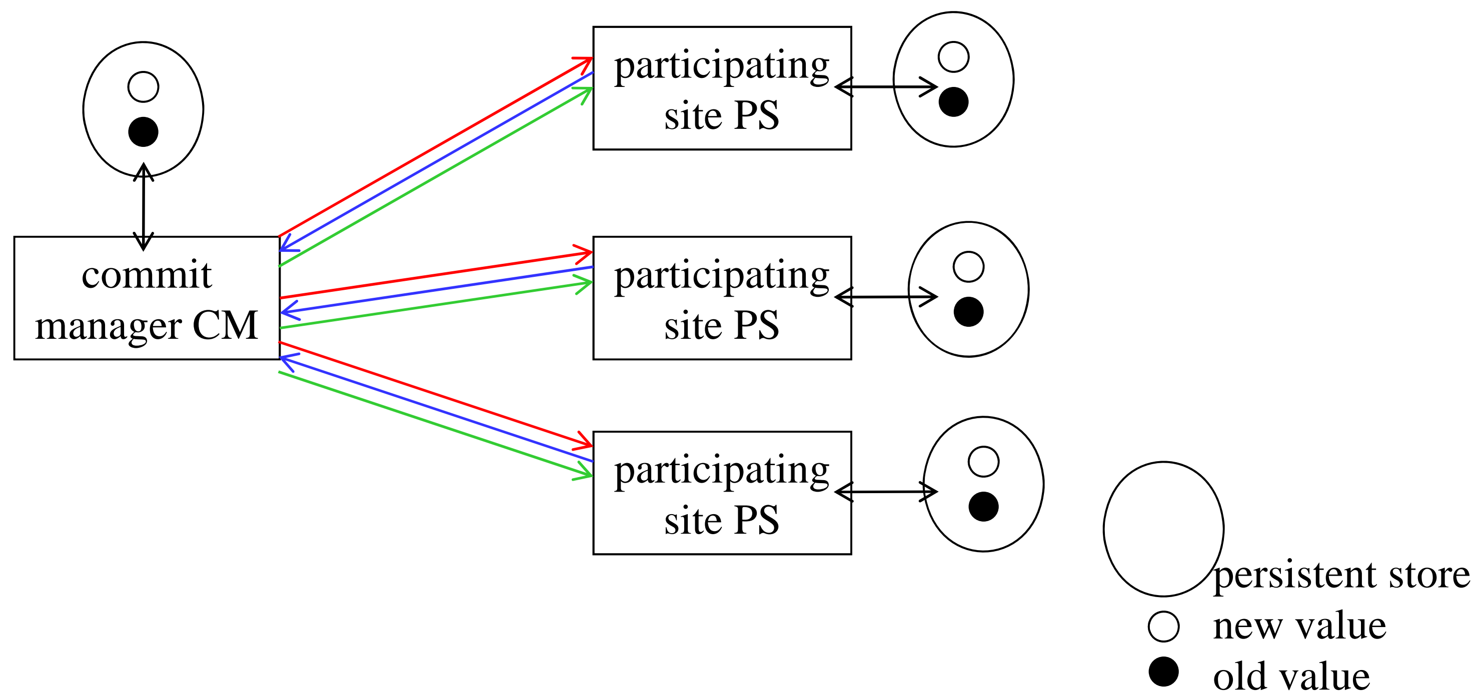
read from a
current version

Example continued with n = 7, QW = 5, QR = 3



Distributed atomic update for replicas and transactions

For both write quora of replicas and related objects being updated under a transaction
we need atomic commitment – all make the update or none does
This is achieved by an atomic commitment protocol, such as **two-phase commit (2PC)**



For a group of processes, one functions as commit manager CM, and runs the 2PC protocol,
the others are participating sites PS, and participate in the 2PC protocol

Two-phase commit

Phase 1

1. CM requests votes from all (PS and CM) – all secure data to persistent store then vote
2. CM assembles votes including its own

Phase 2

CM decides on commit (if all have voted commit) or abort (if any have voted abort)
This is the single point of decision of the algorithm

3. CM propagates decision to all PSs

Failures during 2PC, recall DS independent failure modes
Before voting commit each PS must

- record in persistent store that 2PC is in progress
- save the update to persistent store

.....crash .... at this point

- on restart, find out from CM what the decision was

Before deciding commit, CM must

- record in persistent store that 2PC is in progress
- record its own update to persistent store
- collect all votes, including its own

On deciding commit, CM must:

- record the decision in persistent store ..... crash before propagation ....
- propagate the decision ..... crash during propagation ....

On restart, lookup the decision and propagate it to all PSs

2PC some detail from the CM and PS algorithms

PS algorithm

Either send abort vote and exit protocol or send commit and await decision (set timer)

Timer expires

- CM could be waiting for slow PSs
- CM crash before deciding commit
- CM crash after deciding commit
  - before propagating to any PSs
  - after propagating to some PSs
    - optimise – CM sends list of PSs – ask any for decision

CM algorithm

- send vote request and await replies – set timer
- if any PS does not reply, decide abort and propagate decision

2PC for quorum update

After abort, contact more replicas and start 2PC again

Perhaps assemble more than the required write quorum and commit if a quorum vote commit.

Concurrency in Quorum Assembly

Consider a process group, each member managing a replica
Assume the group is open and unstructured
– any member can take an external update request

Suppose two or more different update requests are made at different replica managers

Each attempts to assemble a write quorum
– if successful, will run a 2PC protocol as CM
either: one succeeds in assembling a write quorum, the other(s) fail
or: both/all fail to assemble a quorum
– e.g. each of two lock half the replicas
we have DEADLOCK

Concurrency in Quorum Assembly – Deadlock prevention

Can deadlock be prevented or detected?

Assume all quorum assembly requests are multicast to all group members

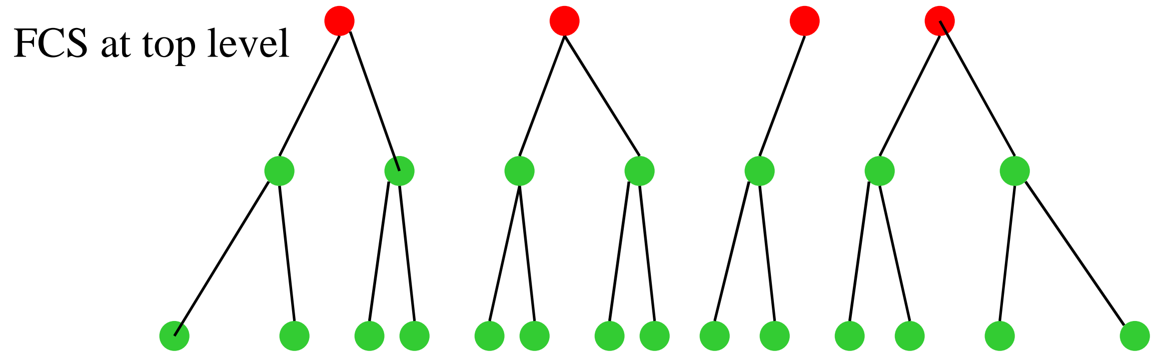
1. The quorum assembler's timeout expires, waiting for enough replica managers to join. It could release locked replicas and restart after backing off for a time.

2. Some replica manager has become part of a quorum (request had a timestamp) then receives a request from a different quorum assembler (with a timestamp) All replica managers agree who wins e.g. based on earliest timestamp wins. Members of the losing quorum can send abort, and exit, then join the winning quorum.

3. Use an open, structured* group so update requests are forwarded to the manager, which is always the CM.
   (* recall lecture DS-2: process groups and message ordering).

Quorum Assembly in large-scale systems

It is difficult to assemble a quorum from a *large number of widely distributed replicas*.
- manage the *scale* by using a *hierarchy*:
  define a small set of first class servers (FCS) each above a subtree of servers



There are various approaches, based on application requirements such as:
speed of response, consistency etc.

Example:

- Update requests must be made to a FCS
- FCSs use quorum assembly and 2PC among FCSs then propagate the update to all FCSs
- Each FCS propagates down its own subtree
- Correct read is from a FCS which assembles a read quorum
- Fast read (value + timestamp) is from any server – with the risk of missing most recent updates

Concurrency Control for Distributed **Transactions**

- Transactions comprise related updates to (distributed) objects
- Any object may fail independently at any time in a DS.
- Distributed atomic commitment must be achieved e.g. by two-phase commit (2PC)
- Concurrent transactions may have objects in common.

Recall pessimistic concurrency control (lecture CC-8)
(strict) two-phase locking (2PL)
(strict) timestamp ordering (TSO)

Recall optimistic concurrency control (OCC) (lecture CC-8)
- take shadow copies of objects, apply updates to shadows, request commit of the validator
- the validator implements commit or abort (do nothing)

For pessimistic CC, atomic commitment is achieved by a protocol e.g. 2PC
note that strict pessimistic CC must be used (objects locked until after commit) – see below

If a fully optimistic approach is taken we do not lock objects for commitment,
since a validator commits new object versions atomically – see below.

Pessimistic Concurrency Control for Distributed Transactions

Strict two-phase locking 2PL

Phase 1:

For objects involved in the transaction, attempt to lock object and apply update

- locks are held while others are acquired, to avoid cycles in the serialisation graph
- so susceptible to DEADLOCK (indicating a SG cycle would have occurred)

Phase 2:

(for strict 2PL locks are held until after commit)

Commit update using e.g. 2PC protocol

Strict timestamp ordering TSO

Each transaction is given a timestamp

For objects involved in the transaction, attempt to lock object and apply update

The object compares the timestamp of the requesting transaction with that of its most recent update. If later – OK. If earlier REJECT as TOO LATE – the transaction aborts and restarts with a later timestamp.

(for strict TSO, locks are held until commit)

Commit update using e.g. 2PC

Implementation of OCC

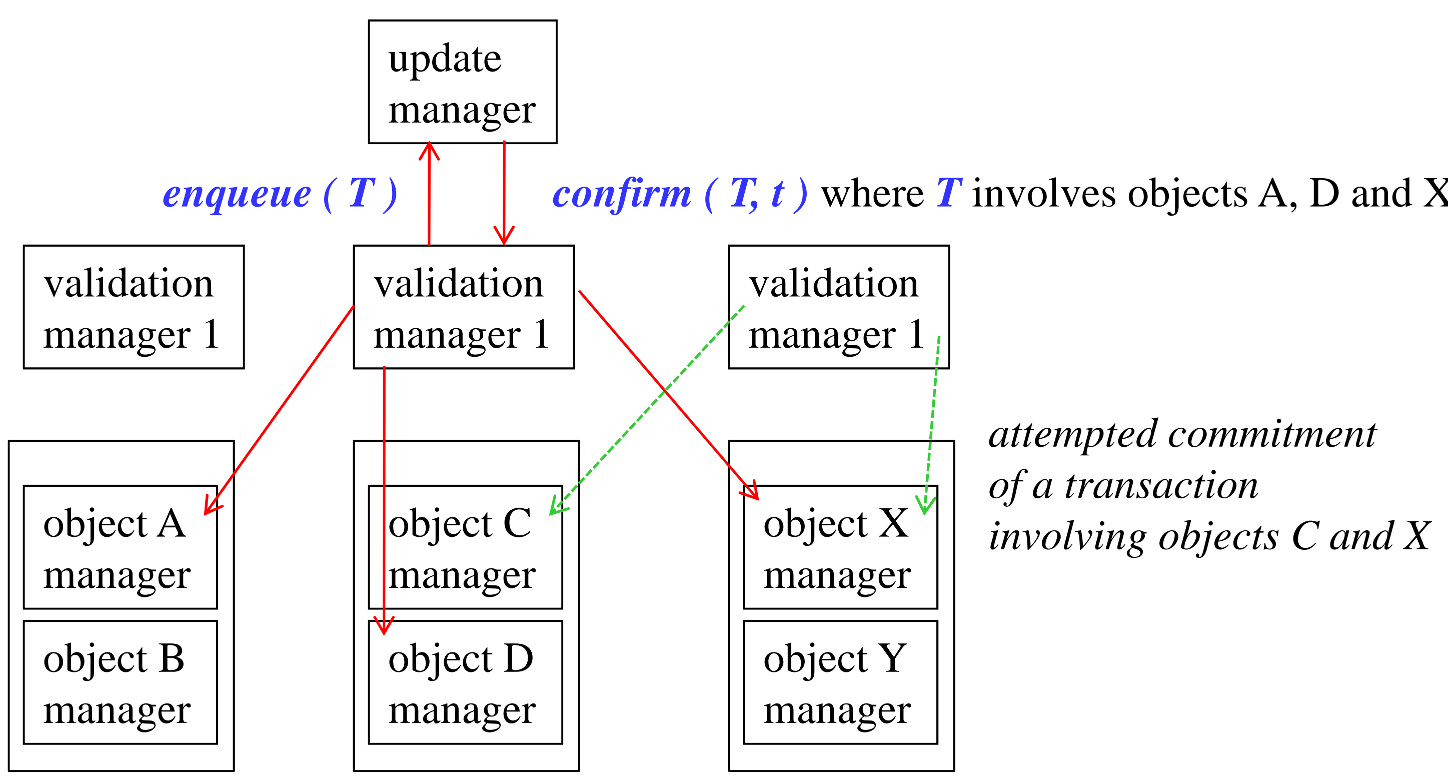
If a fully optimistic approach is taken we do not lock objects until after commitment, since a validator commits new object versions atomically.

The next three slides show single-threading of transactions' commitment through requesting ***commit*** of the validator.

These are included for completeness and further study. They will not be lectured or examined.

Two-phase validation for Optimistic Concurrency Control

Assumptions:
A validator per node – the validator at the coordinating node runs the commitment protocol.
A single, system-wide update manager, responsible for the queue of transactions validated for update.
Each object votes independently ***accept, reject*** on whether there has been a conflict; ***accept*** is sent with the version timestamp of the shadow object.



Two-phase validation for OCC – notes 1

Differences from atomic commitment

1. an object may participate in several protocols concurrently

2. if all objects vote ***accept*** in the first phase and the transaction is validated successfully, they do not need to apply the updates in the second phase. Instead, the VM applies for a timestamp from the update manager
   – at this point the serialisation order is determined.
   This interaction is atomic.
   The VM must then inform each participating object of the decision.

Two-phase validation for OCC – notes 2

Phase 1 outline

The VM requests and assembles votes for ***accept*** or ***reject*** from each participating object,
except that some may say ***busy*** – try again later,
if they are involved in a concurrent transaction.
If any vote is ***reject***, the transaction is aborted.
All votes must be ***accept*** before ***commit*** can be accomplished.
If any vote is ***busy***, all objects are told to suspend validation for a subsequent retry.
Objects that voted ***accept*** are then free to validate other transactions.
On retry, objects that originally voted ***accept*** may vote ***reject***.

Phase 2 outline

The VM decides to ***commit, abort*** or ***retry*** on the basis of the votes,
taking account of shadow object consistency.
If the decision is ***commit***, the VM applies to the update manager for a timestamp.
The decision is propagated to participating objects, with the timestamp.

Algorithms and protocols for distributed systems

We have defined process groups as having peer or hierarchical structure
and have seen that a coordinator may be needed to run a protocol such as 2PC.

With peer structure, an external process may send an update request to any group member, which then functions as coordinator. We have seen that deadlock may occur.

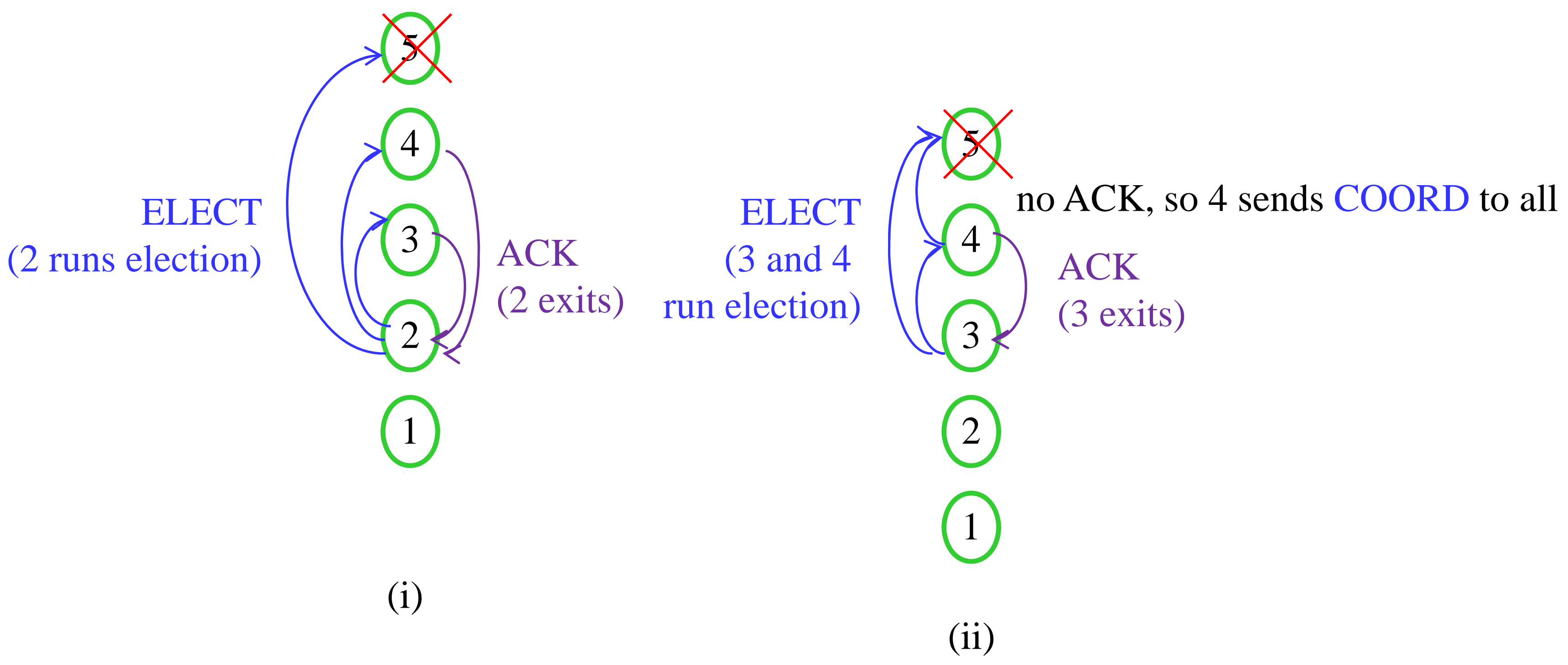
If the group has hierarchical structure, one member is elected as coordinator.
That member must manage group protocols, and external requests must be sent on to it.
Note that this solves the potential deadlock problem of concurrent updates.
But a *single point of failure* is created, and a potential *bottleneck*, so this is only suitable for small groups.

If the coordinator fails, a new one must be elected.
For the election algorithm:

assume that: - each process has a unique ID known to all members
- the process with highest ID is coordinator

Election algorithm - Bully

P notices no reply from coordinator
P sends ELECT message to all processes with higher IDs
If any reply, P exits. Any process that replies must itself run an election.
If none reply, P wins – gets any required state from persistent storage
– P sends COORD message to the group



Election algorithm - Ring

Processes are ordered into a ring structure that is known to all.
A failed process can be bypassed, provided that the ordering around the ring is known and that messages are acknowledged, so that a no-ACK can be detected.

The election algorithm:
P notices that the coordinator is not functioning.

P sends an ELECT message to the next process in the ring, tagged with its own (P's) ID

On receipt of ELECT by any process:

- without receiver's (its own) ID:
  append receiver's ID to message and send to next process in ring
- with receiver's ID (it's been all round):
  look for highest ID in list of appended IDs and send COORD (highest ID) message

Many election algorithms can run concurrently
All should agree on the same highest ID

Distributed mutual exclusion

Suppose N processes hold an object replica and it is required that
only one-at-a-time may access its replica.

Examples: - ensuring coherence of distributed shared memory
- distributed games
- distributed whiteboard

i.e. for use by *simultaneously running, tightly coupled components* managing object replicas in main memory. We have already seen the approach of LOCKING *persistent object replicas* for consistent, transactional update.

Assume that - processes update in place
- then the update is propagated (not shown as part of the algorithm)

Each process executes code of the form:

entry protocol
access object under exclusion, in a critical region
exit protocol

Distributed mutex 1. centralised algorithm

One process is the elected coordinator

entry protocol

send ***request*** message to coordinator

wait for ***reply (OK-to- enter)*** from coordinator

access object under exclusion

exit protocol

send ***finished*** message to coordinator

+ fair FCFS or priority if coordinator re-orders
+ economical (3 messages in basic protocol, but need more ....)
– coordinator is single point of failure (need to elect a new one if it fails)
– what does no reply mean? waiting for exclusive access – OK
but coordinator could have failed

Improve/solve by using extra messages:

- coordinator ACKs request and sends again when process can proceed?
- heartbeat protocol between coordinator and processes awaiting object access

Distributed mutex 2 – Token Ring

A token giving permission to access the object circulates indefinitely

entry protocol

wait for token to arrive

access object under exclusion

exit protocol

pass on token

- Not controllable – ring order, not request order or priority order
- Quite efficient, but token circulates when no process wants object access
- Must handle loss of token and regeneration, ensuring one token only
- crashes? Use ACKs, reconfigure, bypass failed processes

3. Distributed peer-to-peer algorithm

entry protocol

send a timestamped ***request*** to all processes including oneself
(there is a convention for ordering timestamps: earliest timestamp wins + tiebreaker)
only when ALL processes have sent ***reply*** can the object be accessed
Any process executing the protocol, so wanting access, sends a ***reply*** to
all processes whose request messages have earlier timestamps than its own message

access object under exclusion

exit protocol

reply to any deferred requests

+ fair FCFS
– not economical, 2N messages + any ACKs
– N points of failure
– N bottlenecks
– no reply? Failure or deferring?

A bad idea – requiring ALL processes to act before any one can proceed.

Introduction to Middleware I

- What is Middleware?
  - Layer between OS and distributed applications
  - Hides complexity and heterogeneity of distributed system
  - Bridges gap between low-level OS communications and programming language abstractions
  - Provides common programming abstraction and infrastructure for distributed applications
  - Overview at: *http://www.middleware.org*

Distributed Applications

Middleware (remote calls, object invocation, messages, ...)

Operating System Comms (sockets, IP, TCP, UDP, ...)

Network (packets, bits...)

Introduction to Middleware II

- Middleware provides support for (some of):
  - Naming, Location, Service discovery, Replication
  - Protocol handling, Communication faults, QoS
  - Synchronisation, Concurrency, Transactions, Storage
  - Access control, Authentication

- Middleware dimensions:
  - Request/Reply vs. Asynchronous Messaging
  - Language-specific vs. Language-independent
  - Proprietary vs. Standards-based
  - Small-scale vs. Large-scale
  - Tightly-coupled vs. Loosely-coupled components

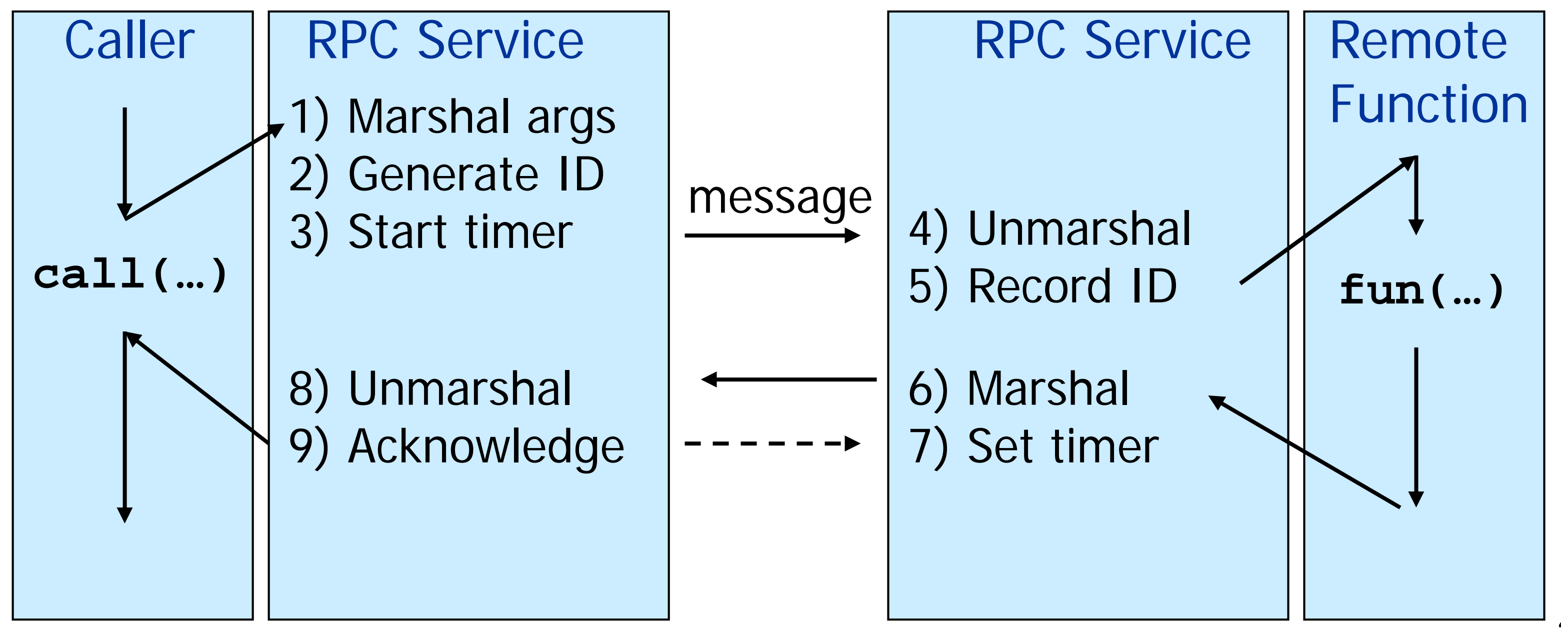
Outline

- Part I: Remote Procedure Call (RPC)
  - Historic interest
- Part II: Object-Oriented Middleware (OOM)
  - Java RMI
  - CORBA
  - Reflective Middleware

- Part III: Message-Oriented Middleware (MOM)
  - Java Message Service
  - IBM MQSeries
  - Web Services
- Part IV: Event-Based Middleware
  - Cambridge Event Architecture
  - Hermes

Part I: Remote Procedure Call (RPC)

- Masks remote function calls as being local
- Client/server model
- Request/reply paradigm usually implemented with message passing in RPC service
- Marshalling of function parameters and return value



Properties of RPC

Language-level pattern of **function call**

- easy to understand for programmer

Synchronous request/reply interaction

- natural from a programming language point-of-view
- matches replies to requests
- built in synchronisation of requests and replies

Distribution transparency (in the no-failure case)

- hides the complexity of a distributed system

Various **reliability** guarantees

- deals with some distributed systems aspects of failure

Failure Modes of RPC

- Invocation semantics supported by RPC in the light of:
  - network and/or server congestion,
  - client, network and/or server failure
    - note DS independent failure modes
- RPC systems differ, many examples, local was Mayflower

Maybe or at most once (RPC system tries once)

- Error return – programmer may retry

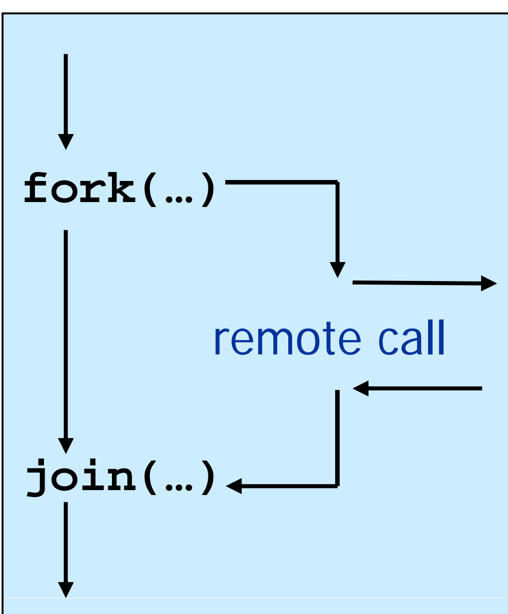
Exactly once (RPC system retries a few times)

- Hard error return – some failure most likely

  note that "exactly once" cannot be guaranteed

Disadvantages of RPC

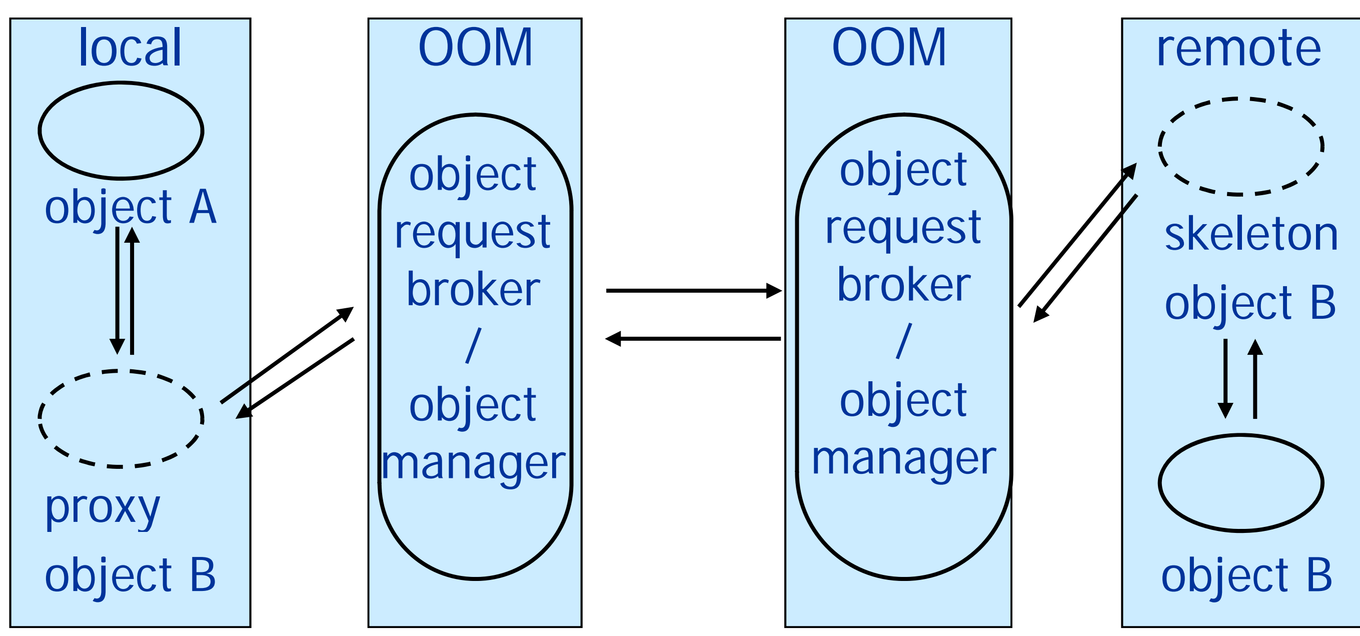
- ✖ Synchronous request/reply interaction
  - tight coupling between client and server
  - client may block for a long time if server loaded
    leads to multi-threaded programming at client
  - slow/failed clients may delay servers when replying
    multi-threading essential at servers



- ✖ Distribution Transparency
  - Not possible to mask all problems
- ✖ RPC paradigm is not object-oriented
  - invoke functions on servers as opposed to methods on objects

Part II: Object-Oriented Middleware (OOM)

- **Objects** can be *local* or *remote*
- **Object references** can be *local* or *remote*
- Remote objects have visible **remote interfaces**
- Masks remote objects as being local using **proxy objects**
- **Remote method invocation**



Properties of OOM

Support for object-oriented programming model

- objects, methods, interfaces, encapsulation, …
- exceptions (were also in some RPC systems e.g. Mayflower)

Synchronous request/reply interaction

- same as RPC

Location Transparency

- system (ORB) maps object references to locations

Services comprising multiple servers are easier to build with OOM

- RPC programming is in terms of ***server-interface (operation)***
- RPC system looks up server address in a location service

Java Remote Method Invocation (RMI)

- Covered in 1B Advanced Java programming
- Distributed objects in Java

```
public interface PrintService extends Remote {
  int print(Vector printJob) throws RemoteException;
}
```

- RMI compiler creates proxies and skeletons
- RMI registry used for interface lookup
- Entire system written in Java (single-language system)

CORBA

- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
  - Open standard by the OMG (Version 3.0)
  - Language- and platform independent
- **Object Request Broker** (ORB)
  - General Inter-ORB Protocol (GIOP) for communication
  - Interoperable Object References (IOR) contain object location
  - CORBA **Interface Definition Language** (IDL)
    - Stubs (proxies) and skeletons created by IDL compiler
  - Dynamic remote method invocation
- **Interface Repository**
  - Querying existing remote interfaces
- **Implementation Repository**
  - Activating remote objects on demand

CORBA IDL

- Definition of language-independent remote interfaces
  - **Language mappings** to C++, Java, Smalltalk, …
  - Translation by IDL compiler
- Type system
  - *basic types*: long (32 bit), long long (64 bit), short, float, char, boolean, octet, any, …
  - *constructed types*: struct, union, sequence, array, enum
  - *objects* (common super type `Object`)

```
typedef sequence<string> Files;
interface PrintService : Server {
  void print(in Files printJob);
};
```

- Parameter passing
  - `in`, `out`, `inout`
  - basic & constructed types passed by value
  - objects passed by reference

CORBA Services (selection)

- Naming Service
  - Names → remote object references
- Trading Service
  - Attributes (properties) → remote object references
- Persistent Object Service
  - Implementation of persistent CORBA objects
- Transaction Service
  - Making object invocation part of transactions
- Event Service and Notification Service
  - In response to applications‘ need for asynchronous communication
  - built above synchronous communication with *push* or *pull* options
  - *not* an integrated programming model with general IDL messages

Disadvantages of OOM

✖ Synchronous request/reply interaction only

- So CORBA **`oneway`** semantics added and -
- Asynchronous Method Invocation (AMI)
- But *implementations* may not be loosely coupled

✖ Distributed garbage collection

- Releasing memory for unused remote objects

✖ OOM rather static and heavy-weight

- Bad for ubiquitous systems and embedded devices

OOM experience

Keynote address at Middleware 2009

Steve Vinoski

From Middleware Implementor to Middleware User

(There and back again)

Available from the course materials page and the MW09 program on the website

Reflective Middleware

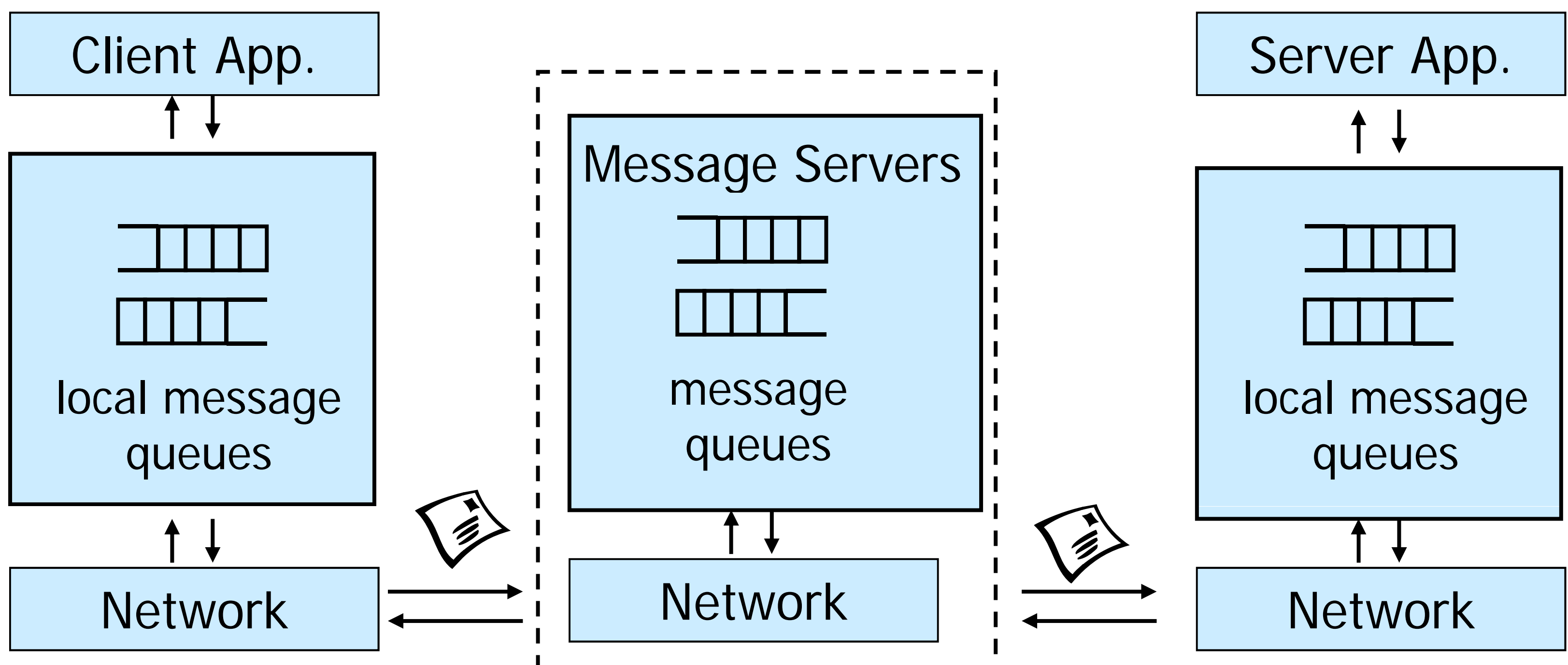
- Flexible middleware (OOM) for mobile and context-aware applications – **adaptation** to context through *monitoring* and *substitution* of components

- Interfaces for **reflection**
  - Objects can inspect middleware behaviour

- Interfaces for **customisability**
  - Dynamic reconfiguration depending on environment
  - Different protocols, QoS, ...
  - e.g. use different marshalling strategy over unreliable wireless link

Part III: Message-Oriented Middleware (MOM)

- Communication using **messages**
- Messages stored in **message queues**
- **message servers** decouple client and server
- Various assumptions about **message content**



Properties of MOM

- **Asynchronous** interaction
  - Client and server are only **loosely coupled**
  - Messages are queued
  - Good for application integration
- Support for **reliable** delivery service
  - Keep queues in persistent storage
- Processing of messages by intermediate message server(s)
  - May do filtering, transforming, logging, …
  - Networks of message servers
- Natural for database integration

IBM MQSeries

- One-to-one reliable message passing using queues
  - Persistent and non-persistent messages
  - Message priorities, message notification
- **Queue Managers**
  - Responsible for queues
  - Transfer messages from input to output queues
  - Keep routing tables
- **Message Channels**
  - Reliable connections between queue managers
- Messaging API:

MQopen	Open a queue
MQclose	Close a queue
MQput	Put message into opened queue
MQget	Get message from local queue

Java Message Service (JMS)

- API **specification** to access MOM implementations
- Two modes of operation *specified*:
  - **Point-to-point**
    - one-to-one communication using queues
  - **Publish/Subscribe**
    - cf. Event-Based Middleware

- **JMS Server** implements JMS API
- JMS Clients connect to JMS servers
- Java objects can be serialised to JMS messages
- A JMS interface has been provided for MQ
- pub/sub (one-to-many) - just a specification?

Disadvantages of MOM

✖ Poor programming abstraction (but has evolved)

- Rather low-level (cf. Packets)
- Request/reply more difficult to achieve, but can be done

✖ Message formats originally unknown to middleware

- No type checking (JMS addresses this – implementation?)

✖ Queue abstraction only gives one-to-one communication

- Limits scalability (JMS pub/sub – implementation?)

Web Services

- Use well-known web standards for distributed computing

Communication

- Message content expressed in **XML**
- **S**imple **O**bject **A**ccess **P**rotocol (**SOAP**)
  - Lightweight protocol for sync/async communication

Service Description

- **W**eb **S**ervices **D**escription **L**anguage (**WSDL**)
  - Interface description for web services

Service Discovery

- **U**niversal **D**escription **D**iscovery and **I**ntegration (**UDDI**)
  - Directory with web service description in WSDL

Properties of Web Services

Language-independent and open standard

SOAP offers OOM and MOM-style communication:

- Synchronous request/reply like OOM
- Asynchronous messaging like MOM
- Supports internet transports (http, smtp, ...)
- Uses XML Schema for marshalling types to/from programming language types

WSDL says how to use a web service

http://api.google.com/GoogleSearch.wsdl

UDDI helps to find the right web service

- Exports SOAP API for access

Disadvantages of Web Services

✖ Low-level abstraction

- leaves a lot to be implemented

✖ Interaction patterns have to be built

- one-to-one and request-reply provided
- one-to-many?
- still synchronous service invocation, rather than notification
- No nested/grouped invocations, transactions, ...

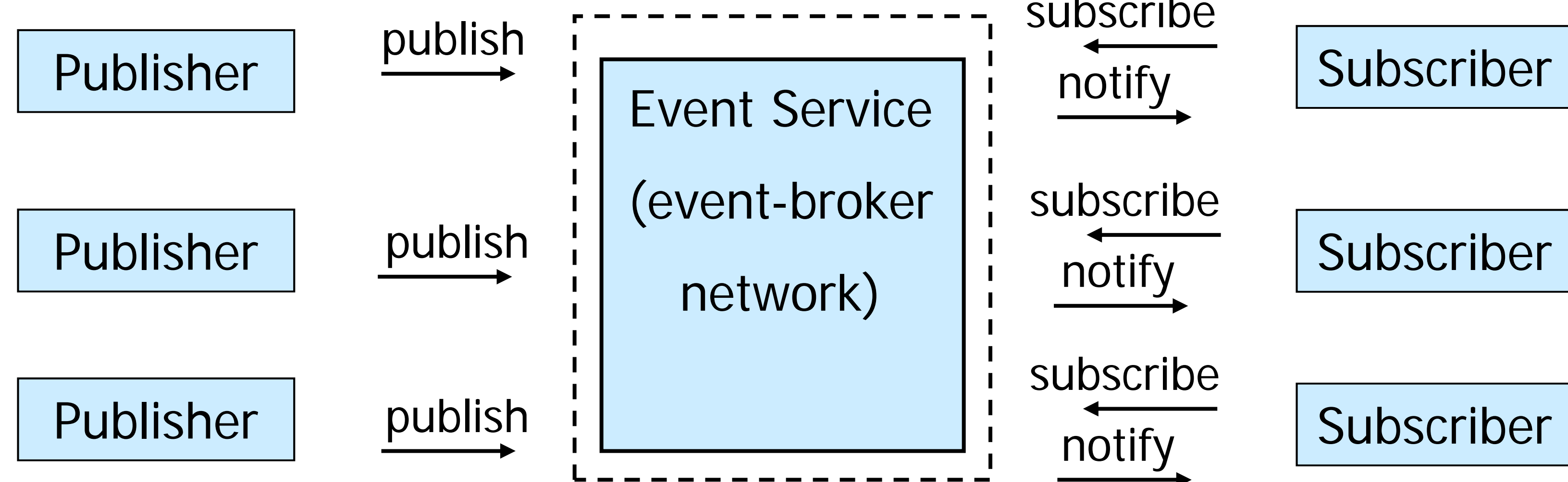
✖ No location transparency

What we lack, so far

- ✖ General interaction patterns
  - we have one-to-one and request-reply
  - one-to-many? many to many?
  - notification?
  - dynamic joining and leaving?
- ✖ Location transparency
  - anonymity of communicating entities
- ✖ Support for pervasive computing
  - data values from sensors
  - lightweight software

Part IV: Event-Based Middleware a.k.a. Publish/Subscribe

- **Publishers** (*advertise* and) *publish* **events** (messages)
- **Subscribers** express interest in events with *subscriptions*
- **Event Service** *notifies* interested subscribers of published events
- Events can have arbitrary content (typed) or name/value pairs



Topic-Based and Content-Based Pub/Sub

- Event Service matches events against subscriptions
- What do subscriptions look like?

Topic-Based Publish/Subscribe

- Publishers publish events belonging to a **topic** or **subject**
- Subscribers subscribe to a **topic**

```
subscribe(PrintJobFinishedTopic, …)
```

(Topic and) Content-Based Publish/Subscribe

- Publishers publish events belonging to **topic**s and
- Subscribers provide a **filter** based on ***content*** of events

```
subscribe(type=printjobfinished, printer='aspen', …)
```

Properties of Publish/Subscribe

Asynchronous communication

- Publishers and subscribers are loosely coupled

Many-to-many interaction between pubs. and subs.

- Scalable scheme for large-scale systems
- Publishers do not need to know subscribers, and vice-versa
- Dynamic join and leave of pubs, subs, (brokers - see lecture DS-8)

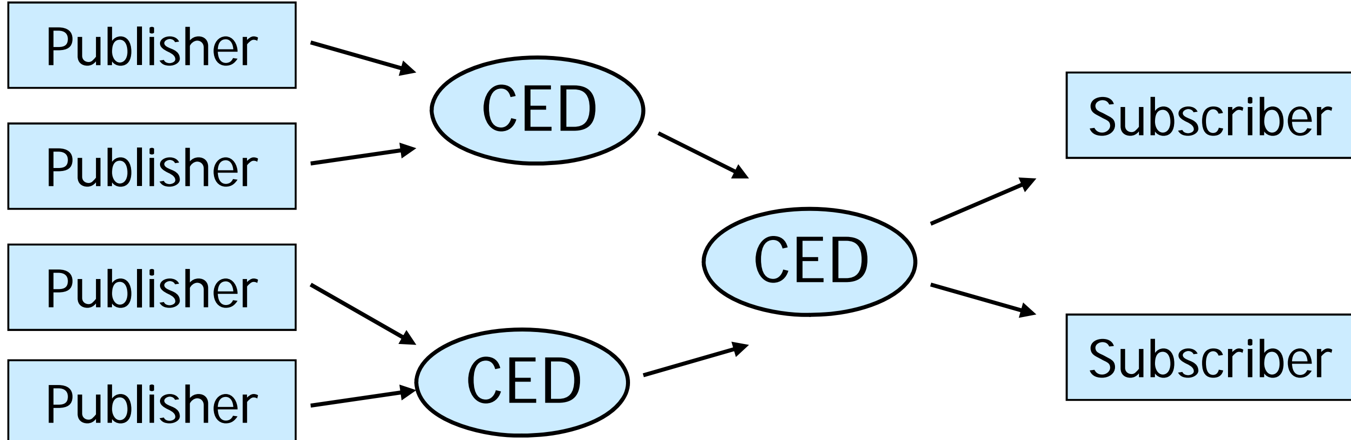
(Topic and) Content-based pub/sub very expressive

- Filtered information delivered only to interested parties
- Efficient content-based routing through a broker network

Composite Event Detection (CED)

- Content-based pub/sub may not be expressive enough
  - Potentially thousands of event types (primitive events)
  - Subscribers interest: event *patterns* (define *high-level* events, ref DS-2)
- **Event Patterns**

  `PrinterOutOfPaperEvent` or `PrinterOutOfTonerEvent`
- **C**omposite **E**vent **D**etectors (CED)
  - Subscribe to primitive events and publish composite events



Summary

- Middleware is an important abstraction for building distributed systems

1. Remote Procedure Call
2. Object-Oriented Middleware
3. Message-Oriented Middleware
4. Event-Based Middleware

- Synchronous vs. asynchronous communication
- Scalability, many-to-many communication
- Language integration
- Ubiquitous systems, mobile systems

Naming in Distributed Systems

Unique identifiers **UIDs** e.g. 128 bits

- are never reused
- refer to the same thing at all times, or to nothing at all

UIDs should be location-independent! Can the named object be moved?

Pure and impure names (Needham)

pure names

- the name itself yields no information, and commits the system to nothing
- it can only be used to compare with other similar names e.g. in table look-up

impure names

- the name yields information,
- and commits the system to maintaining the context in which it can be resolved

Examples of impure names

jmb25@cl.cam.ac.uk name of a person, registered in a DNS domain
jeanb@lcs.mit.edu name of a person, registered in another DNS domain
another or the same person?

puccini.cl.cam.ac.uk name of a machine, registered in a DNS domain

(disc-pack-ID, object-ID) Bad idea from history of naming files and directories.
Seemed efficient until the objects had to be moved

(host-ID, object-ID) OK, it's impure .... but how are pure names generated in a DS?
We must not have centralised name allocation.
(***host-ID, object-ID)*** has been used (badly) in middleware,
and has made the objects unmoveable.
It could be used to generate pure names,
if we do not make use of the separate fields. Typical example:

32-bit host-ID	96 bit object-ID

Unique names

Both pure and impure names can be unique.
Uniqueness can be achieved by using:

for impure names:

- hierarchical names: scope of uniqueness is level in hierarchy
  (uniqueness is within the names in the directory in which the name is recorded)

for pure names

- a bit pattern: flat, system-wide uniqueness,

Problems with pure names:

- where to look them up to find out information about them?
- how do you know that an object does not exist? How can a global search be avoided?
- how to engineer uniqueness reliably in a distributed system?
  centralised creation of names? As discussed above, ***(host-ID, object-ID)***?

Problems with impure names:

- how to restructure the namespace
  e.g. when objects move about or when companies restructure

Examples of (pure/impure) names - unique identification

UK national insurance - allocated on employment
US Social Security - allocated on employment
Passport
Driving licence
Services: RAC, AA, AAA(US), AAA(Aus)
Credit cards
Bank accounts
Utilities' customer numbers: gas/electricity/water/phone
Charity members
Loyalty card members

For the above examples:

- Is the structure explicit or implicit?
- Is allocation centralised or distributed?
- What is the resolution context?

More examples of names - unique identification?

UK health service (NHS) ID – allocated to every citizen at birth

but hospitals still use local patient numbers per treatment, with names recorded at the time
e.g. is J. Ken Moody the same person as John K. Moody?
can medication information be used interchangeably?

Professional societies: BCS, ACM, IEEE

aside: A database key MUST be unique, e.g. Jean Bacon is not unique
Jean Bacon in Connecticut USA, and I were allocated the same IEEE membership number
– our records had been merged: Cambridge work, Connecticut home. I got renewal notice.
IEEE membership would not believe me until I found her and we emailed them together.

e.g. from the Computer Lab, Jatinder Singh is held up at immigration
because he shares the name and date of birth of another Jatinder Singh

Middleware 2009 keynote address:
Professor Hector Garcia-Molina, Stanford
Entity Resolution – Glue for Middleware

available from the MW09 program, and the course materials page

Telephone company analogy – wired service

Geographically partitioned distributed naming database.

Electronic version is current, paper directories are an official cache
Frequency of update (some years ago):
Cambridge area 1,000,000 entries, 5,000 updates a week

Given a name e.g. (Yudel Luke), or (Yudel Luke, 3 Acacia Drive) which directory to use?

- don't know where to look up pure names

Lookup doesn't yield useable information:
Call# **->** unobtainable, where # is from the official cache (paper directories)
we detect out-of-date values, call directory enquiries, cache unofficially until new directory
Call# **->** unobtainable, for a number that we know and use often, or from personal address book
redial, report fault, check official cache, ask social network if X has moved phone

Can't find an entry in the official cache (exact matching required)
e.g. Phillips - check spelling – Philips
e.g. try acronyms S.S. for Social Services

BT offer a web service *www.thephonebook.com* (name and address -> number)
only offers exact matching e.g. Philips not suggested for Phillips (do you mean?)
increasingly, search engine approaches are used to augment directory lookup

Name spaces and naming domains

In general, provide clients with values of attributes of named objects

Name space

- the collection of valid names recognised by a name service
- a precise specification is required, giving the structure of names

e.g. ISBN:1-234567-89-1 namespace identifier, namespace-specific string
/a/b/c/d file system pathname, variable length, hierarchical
puccini.cl.cam.ac.uk DNS machine name – see case study below
128 bit system-wide OS port name for Mach OS, system-wide UID

Naming domain

a name space for which there exists a single overall administrative authority
for assigning names within it
this authority may delegate name assignment for nested sub-domains (see DNS below)

Name resolution - binding

Name resolution or binding

obtaining a value for an attribute of the named object that allows the object to be used

Late binding is considered good practice
Programs should contain names, not addresses

A machine may fail and the service moved to another machine.
Your local agent may cache resolved names for subsequent use,
and may expire values based on timestamps (TTL time-to-live)

Cached values are always used "at your own risk".
They should not be embedded in programs.

If cached values don't work, the lookup has to be repeated.
Lookup may be iterative for large-scale systems – see later.

Names, attributes and values stored in a name service

example:

object type	attribute list
user	login-name, mailbox-hosts(s)
computer	architecture, OS, network-address, owner
service	network-address, version#, protocol
group	list of names of members
alias	canonical name
directory?	list of hosts holding the directory (may be held in a separate structure rather than as a type of name, as here)

Directories are likely to be replicated for scalability, fault-tolerance, efficiency, availability

Directory names resolve to a list of hosts plus their addresses to avoid an extra lookup per host

Attribute-based (inverse) lookup may be offered –
A YELLOW PAGES style of service for object discovery e.g. X.500, LDAP

Too much information can be dangerous – useful for hackers

Names, attributes and values - examples

object type -> list of attribute names

name service holds:
object-type, object-name -> list of attribute values

You can acquire a standard directory service e.g. LDAP and use it to store whatever your service/application needs

querying:
object-type, object-name, attribute-name -> attribute value

computer, puccini.cl.cam.ac.uk, location **->** IP-address
user,   some-user-name, public-key **->** PK bit pattern

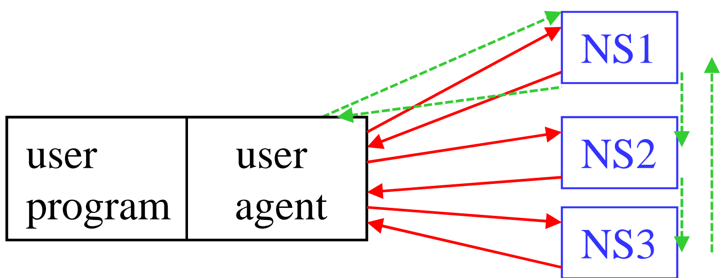
checking:
object-type, object-name, attribute-name, attribute value -> yes/no

ACL,   filename, some-user-name, write-access   **->** yes/no

Attribute-based (inverse) lookup:
object-type, attribute-name, attribute value -> list of object-names

computer, OS version#, OS version# value -> list of computers

Iterative name resolution



user agent (UA) starts from the root address of the name service, or tries some well-known sub-tree root, e.g. the location of the ***uk*** directory may be used by agents in the UK

To resolve ***cl.cam.ac.uk*** the UA, as in

looks up ***ac***'s address in uk, then looks up ***cl***'s address in ***ac***

The UA will cache resolved names as hints for future use, see slide 6

Alternatively, any name server will take a name, resolve it and return the resolved value, as in

The client may be able to choose, e.g. select "recursive" in DNS

Engineering optimisations:

use of caching at UA and at directories

try cached values first

Name services - examples

DNS   Internet Domain Name Service, see below

Grapevine: Xerox PARC early 1980's, *Birrell, Levin, Needham, Schroeder CACM 25(1) 1982*
two-level naming hierarchy e.g. ***name@registry***   ***birrell@pa***
primarily for email, but also gave primitive authentication and access control
(check password as attribute of user, check ACL)
any Grapevine server would take any request from a GV user agent

Clearinghouse, Xeroc PARC, Oppen and Dalal 1983
ISO standard based on an extension of Grapevine
three-level hierarchy

GNS Global Name Service, DEC SRC, *Lampson et al. 1986* - see below
full hierarchical naming
support for namespace restructuring

X.500 and LDAP, see below

Name services in Middleware
CORBA: naming and (interface) trading services, Java JNDI, Web W3C: UDDI
Allow registration of names/interfaces of externally invocable components with interface references and attributes such as location
May offer separate services for: ***name –> object-reference,***   ***object-reference –> location***

Case Study: DNS - Internet Domain Name Service

Before 1987 the whole naming database was held centrally and copied to selected servers periodically
The Internet had become too *large scale* and a *distributed, hierarchical scheme* was needed
(*Paul V.Mockapetris, 1987*)

What does DNS name?

In practice, the objects are:
- computers
- servers such as mail hosts

The directory structure (resolution context) is captured as:
- domains

DNS – Definition of names

Definition of names

hierarchy of components (labels), highest level in hierarchy is last component, total max 255 chars

label: max 63 chars, case insensitive, must start with a letter, only letters, digits, hyphens allowed

final label of a fully qualified name can be:

3-letter code: type of hosting organisation

edu, gov, mil are still US-based, others, e.g. ***com, net, org, int***, can be anywhere

2-letter code: country of origin defined by ISO e.g. ***uk, fr, ie, de***, ...

final 2-letter label doesn't always imply country of location of host , but where the host was registered

e.g. www.yahoo.co.uk is in S germany

e.g. ISO have defined many small "country of origin" domains such as ***to, cc, bv***, ...

arpa: for inverse lookup, e.g. ***128.232.56.7.in-addr.arpa***

Examples of domain names:

mit.edu

cl.cam.ac.uk

cs.tcd.ie

tu-darmstadt.de

DNS

Computers using DNS are grouped into zones e.g. ***uk, cam***
Within a zone, management of nested sub-domains can be delegated
e.g. ***cl*** is managed locally by the domain manager who adds names to a local file
Each zone has a primary name server that holds the master list for the zone. Secondary name servers hold replicas for the zone.

Queries can relate to individual hosts or zones/domains, examples:

query			response
A	computer name	->	IPv4 address
AAAA	computer names	->	IPv6 address
MX	mail host for domain	->	list < host, preference, IP address >
	includes mail hosts for detached computers		
NS	DNS servers for a domain	->	list < host, authority?Y/N, IP address >

DNS - Unix examples

```
$  /usr/bin/nslookup
>   set q = A
>   cosi.cl.cam.ac.uk
Address: 128.232.8.106

>    www.cl.cam.ac.uk
 Address:  128.232.0.20


>   set q = MX
>   cl.cam.ac.uk
cl.cam.ac.uk      mail exchanger = 5  mx.cam.ac.uk
>    set q=A
>    mx.cam.ac.uk
Address: 131.111.8.145


>   set q = NS
>   cl.cam.ac.uk
cl.cam.ac.uk    nameserver = dns1.cl.cam.ac.uk
                        .......                ........
cl.cam.ac.uk    nameserver = dns0.eng.cam.ac.uk
>   set q = A
>   dns0.eng.cam.ac.uk
Address: 129.169.8.8
```

DNS name servers – note the large scale

The domain database is partitioned into directories that form a distributed namespace

e.g. the directory ***ac.uk*** is held on the computer ***nsl.cs.ucl.ac.uk***

We need a starting point for name resolution – e.g. the above for domains within the uk

DNS directory addresses can be looked up for a domain

yielding: IP address, well-known port

directory: ***ac.uk***
cam.ac.uk
cl.cam.ac.uk
....

directory: ***cam.ac.uk***
cl.cam.ac.uk
....

name server: ***cl.cam.ac.uk***
names -> attributes/values

redundant link (or cached value)
for frequently used name

Directories are replicated for availability and good response (primary and secondaries per domain)

Authorised name server for domain is distinguished - weak consistency of secondaries with primary

Resolved queries are cached with a TTL (time to live)

(by user agents and directories) – works because *naming data tends to be stable*

Queries and responses may be batched into composite query messages

DNS design assumptions and future issues

DNS and other name services were designed, in the days of desktops,
on the assumption that objects are static, so that cached values continue to work,
update rates are low, etc.

With huge data centres and high bandwidth available, is it time to recentralise to a few first-class servers? See Tim Deegan's thesis (examined by Paul Mockapetris) and:
T Deegan, J Crowcroft and A Warfield,
"The MAIN name system, an exercise in centralized computing"
ACM SIGCOMM 35(5), Oct 2005.

New issues relate to mobile devices and myriad small devices including sensors

Mobile devices may attach anywhere worldwide
device's MAC address is a UID, IP address?
see comms. courses for details of protocols

Name service design: Replication and Consistency

Directories are replicated for scalability, availability, reliability , ...

How should propagation of updates between replicas be managed?

lookup (arguments) **< - >** is the most recent value known, system wide, guaranteed to be returned?

If system-wide (strong) consistency were guaranteed this would imply:

- delay on update
- delay on lookup

It is essential to have fast access to naming data

– so we relax the consistency requirement

Is this justified?

Name services – assumptions to justify weak consistency

Design assumptions were as below, but new issues have arisen

- naming data change rarely,
- changes propagate quickly,
- inconsistencies will be rare

YES – information on users and machines
NO – distribution lists (see analysis of how Grapevine outgrew its specification)
NEW – mobile users, computers and small devices e.g. Internet-enabled phones
NEW – huge numbers of devices to be named – does the design rely on low update traffic?

- we detect obsolete naming data when it doesn't work
  YES – users
  NO – distribution lists

- If it works it doesn't matter that it's out of date
  - you might have made the request a little earlier
  - recall uncertainties over time in DS

Consistency – vs - Availability

We have argued that availability must be chosen for name services, so weak consistency
When only weak consistency is supported:

lookup (arguments) **- >** returns either: value, version# / timestamp
or: not known at time of last update

Examples:

Service on failed machine, restart at new IP address – update directory(s) – rare event
User changes company – coarse time grain
Companies merge – coarse time grain
Change of password – takes time to propagate – insecurity during propagation
Changes to ACLs and DLs – insecurity during propagation
Revocation of users' credentials – may have been used for authentication/authorisation at session start – can the effect be made instantaneous?
Hot lists e.g. stolen credit cards – must propagate fast – push rather than pull model

Lessons:

Note design assumptions
Take care what data the name service is being used for
Does the service offer notification of change, on registration of interest, as in active databases?

Long-term Consistency

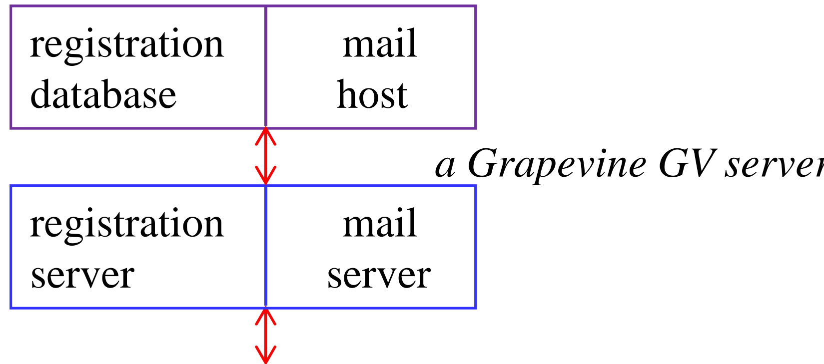
Requirement:
If updates stopped there would be consistency when all updates had propagated
Note that failure and restart behaviour must be specified for updates to propagate reliably

This requirement cannot be tested in a distributed system
- no guarantee that there will be periods of quiescence (no activity)

Updates are propagated by the message transport system
- conflicting updates might arrive out of order, from different sources
- need an arbitration policy based on timestamps
- but recall unreliability of source timestamps, so outcomes of an agreement protocol may not meet the *external* requirements.

Name services typically exchange whole directories periodically and compare them.
The directory is tagged with a new version# after this consistency check
e.g. GNS declares a new “epoch”

Example: Grapevine – the first?



Note that small scale allows a simple design with rapid navigation
2D names ***name@registry***
Every GV server contains the GV registry that contains, for all GV registries worldwide,
registry-name -> list of addresses where registry is held
2 types of name within a registry
group-name -> list of members for distribution lists, also used for ACLs
individual-name -> attributes such as password, mail-host-list, ...

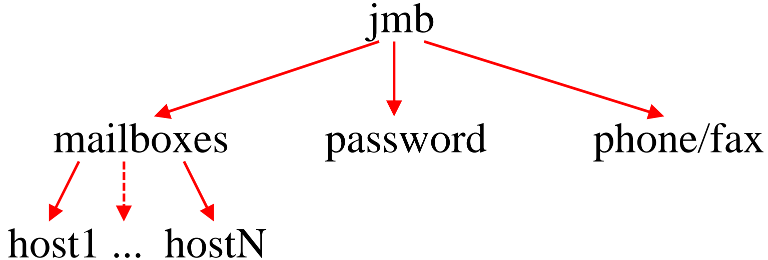
Problems – soon outgrew its specification of #servers, #clients
huge distribution lists were not foreseen
client mail transport protocol used for system updates – could be held up

Global Name service GNS (DEC – 1986)

Lampson, Designing a Global Name service, Proc 5th ACM PODC, 1986

Aims: long life – allowing many changes in the organisation of the namespace
large scale – an arbitrary number of names and domains

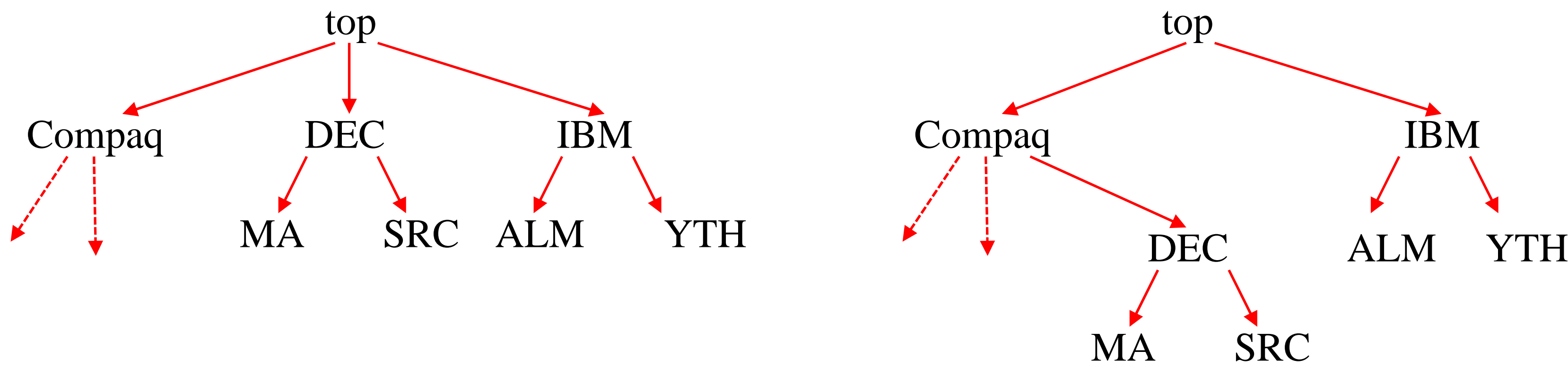
Names
Define 2D names of the form < directory-name, value-name >
where value-names may be a tree such as:



The GNS directory structure is hierarchical
Every directory has a Directory Identifier (DI), a UID *the novel design aspect of GNS*
A full name is any name starting with a DI
- doesn't require a root directory
- doesn't rely on the availability of some root directory

GNS namespace reconfiguration

If the directory hierarchy is reconfigured, a directory may still be found via its DI
Names starting with that DI do not change if the reconfiguration is above that DI



old names still work below the DEC directory

Support is needed by the directory service to locate a directory from its DI
  a DI is a pure name – where do we look it up?
Directories usually map directory pathnames to IP addresses
In addition, top level GNS directories store DIs with directory names, e.g.

GNS namespace reconfiguration – directory updates

Names starting from DEC: ***311/SRC, birrell*** do not change. DEC is always 311
Names starting above DEC: ***999/DEC/SRC, birrell -> 999/Compaq/DEC/SRC, birrell***

Directory entries include – DIs with directory names
– pathnames from root

552 = 999/Compaq 311 = 999/DEC n = 999/DEC/SRC	IP addresses

552 = 999/Compaq 311 = 999/Compaq/DEC n = 999/Compaq/DEC/SRC	IP addresses

X.500 Directory Service (White and yellow pages)

ISO and CCITT standard, above OSI protocol stack
More general than most name services where names must be known precisely
and are resolved to locations
Components:

- DIT directory information tree
- DSA directory service agent
- DUA directory user agent
- DAP directory access protocol

X.500 is resource-consuming and difficult to use

1993 major revision including replication, access control, schema management,
But X.500 was not accepted as a generic name service
X.509 certificates for authentication and attributes/authorisation have been successful

LDAP Lightweight Directory Access Protocol, *Howes, Kille, Yeong, Robins, 1993*
accepted by IETF – can download free and deploy – widely used

- access protocol built on TCP/IP
- heavy use of strings, instead of ASN.1 data-types
- simplification of server and client
- current status V3
- LDUP duplication and update protocol being developed

Naming Postlude

Naming for the Internet, see DNS
Naming for companies, worldwide, motivated Grapevine, see also GNS
Standard name services, X.500, LDAP

Naming for the web – document names are based on internet naming
scheme://host-name:port/pathname at host
scheme = protocol: http, ftp, local file
host-name = web server's DNS address, default port 80
pathname is in web server's filing system of file containing data for web page
e.g. http://www.cl.cam.ac.uk/research/

Also, W3C have defined standards for web services (see Middleware)
with message content expressed in XML
SOAP – simple object access protocol
WSDL – web service description language
UDDI – universal description, discovery and integration
(directory service with web service descriptions in WSDL)

Access control (authorisation) in distributed systems

recall lecture 9 - Introduction to DS: slides 21 to 27
for access control within the overall system architecture:

- as an individual e.g. from home
- within a single administration domain e.g. CL
- using external services from a domain as an individual or group member
- federated domains: inter-domain authorisation

We are concerned with authorisation for service use and/or object access
How is access control policy expressed and enforced?

Authorisation and authentication

Authorisation is built above authentication (proof of identity – proof that you are who you say you are – will someone/something vouch for you?).

Within an administration domain, principals are named and registered as individuals and members of groups.

Principals authenticate in their home domain by means of e.g. passwords.

The aim is to avoid having to have a ***username/password*** for every service.

(.... *How does one remember them all? ......*

....Use the same one for all? No, break one break all .....)

A Single Sign On service is needed.

Authentication is covered in Security courses.

For background reading, slides 29-36 outline some single sign on systems for cross-domain service use: Raven, Shibboleth, OpenID

Access control – from first principles

Model: access matrix A (i, j) rows represent principals, columns objects
entry (i, j) contains the rights principal i has to object j

Implementation: since the matrix is sparse (most entries are null)

1. Using access control lists (ACLs): keep non-null entries of column j with object j
   ACL entry = principal name + access rights
   optimisation: group name = list of principals

2. Using capabilities: keep non-null entries of row i with principal i
   a capability (capability list entry) = object name + access rights

Assume managers for the various types of object

On an access request, the manager must check that the requesting principal has the appropriate right to access the object

1. check that the ACL list contains an entry for that principal with the right
2. check that the capability passed by the principal with the request contains the right

ACLs – cf. - capabilities

ACLs

Expressiveness: subtle expression of policy – entries may be for individual principals and groups with individual exceptions.

Revocation: easy to revoke – but ACL changes may not have immediate effect (because the ACL may not be checked on every access once an object is open)

BUT: slow to check - scalability problem

- if expressiveness exploited e.g. negatives and exceptions allowed
- if there are many principals and large groups
- generalisation? multi-domain operation? names outside domain of registration?

AND: awkward to delegate rights e.g. For a file to a printer for a single print job

In a distributed system many services are not part of privileged OSs.

Capabilities

Quick to check – like a ticket – so scale well

Anonymous – knowledge of names not needed – may generalise to multiple domains.

- anonymity may be wanted by some applications for privacy reasons

Problems/issues ... because they are associated with the process rather than the object ...

Capability-based access control - issues

as defined so far, a capability is an object name and some rights

1. protection

   must prevent unauthorised creation, tampering, theft

2. control of propagation

   can principals pass on copies?

   must they ask the object manager? How can this be enforced?

3. delegation

   is an example of propagation

   often with restricted rights for a limited time or action

4. revocation

   - if the access control policy changes, and certain principals should lose their rights, their capabilities should ideally be revoked. Can this be done without revoking all capabilities for the service/object?
   - if a capability is known to have been stolen or tampered with it should be revoked instantly. Will this invalidate all capabilities for this service/object?

     Anonymity has created revocation problems.

Capabilities in centralised and distributed systems

centralised

Several capability architectures were designed and built e.g. Plessey PP250, CAP

Capabilities can be protected by the hardware and/or the OS

A capability is named via an index into a segment or an OS table.

- held in protected OS space per process
- held in typed capability segments in user space with operations such as
  insert, delete, use-as-argument

distributed

Can't be protected by hardware/OS

Have to be transferred across networks and pass through user space
so must be encryption-protected

Security terminology and implementation:
capability = signed certificate
e.g. X.509 authentication and attribute certificates

Capabilities in distributed systems - design

Must be protected by encryption

The object manager (certificate issuer) keeps a ***SECRET*** (random number, private key) and uses a well-known function ***f***, a one-way function.

A capability is constructed using

check digits = f (SECRET, protected fields)

protected fields	***check digits*** (signature)

When a capability is presented with an operation invocation, the manager checks that:

f (SECRET, protected fields) = check digits

If not, the invocation is rejected.

More generally, the invoked service may not be the capability issuer. The service can check back with the issuer (cf. Certification Authority)

Encryption-protected capabilities – issues?

1. protection

   protect against tampering – adding rights,
   NOT against theft – eavesdropping on the network and replay attacks

2. control of propagation

   still no control over propagation

3. delegation

   object manager must be asked to create a capability with reduced rights to pass to another principal for delegated authority.
   Works indefinitely – duration is not controlled – nor further transfer

4. revocation

   (recall: needed when access control policy changes as well as for stolen capabilities)
   - expiry time as a protected field (like X.509) – crude mechanism
   - hot list of invalid invoking principals per service/object (spoofing? check overhead?)
   - change the SECRET – not selective – all old capabilities will not work and authorised principals will have to request new capabilities.

Principal-specific capabilities

include the name of the principal in the capability for generation and checking as an argument to ***f***, perhaps as a protected field in the capability.

1. protection

   from tampering – YES,   from theft – YES: authenticate presenting principal

2. control of propagation

   YES – a capability for the receiving principal can only be created by the object manager

3. delegation

   YES – a capability for the receiving principal must be created by the object manager

4. revocation

   can be more selective – still involves overhead of checking hot list
   e.g. revocation list of principals excluded by policy change
   stolen capabilities should be detected on authenticating the presenting principal
   unless the presenter is successfully masquerading as the owner.

All the above raise the question of the structure and scope of principal names and how and where principals are authenticated.

ACLs in distributed systems

We first followed the capability thread after slide 11.

We have discussed principal-specific capabilities

Now, return to consider ACLs.

ACLs comprise lists of principals (or groups)

ACL entry = principal (or group) name, rights (from slide 3)

Where principals and groups are defined and registered within some administration domain.

Without group names ACLs may become unmanageable, long lists of principals.

Within the administration domain where a group and its constituent principals are registered, a group name can be expanded into a list of principals, for checking.

How can group names be used outside the domain where the group is registered?

We generalise groups to roles and consider role-based access control (RBAC)

Role-based access control (RBAC)

Services may classify their clients into named roles e.g.

login service: ***logged-in-user*** (after authentication)

patient monitoring service: ***surgeon, doctor, nurse, patient***

online exam service: ***candidate, examiner, chief examiner***

digital library service: ***reader, librarian, administrator***

Access rights (privileges) are assigned to roles for use of services (method invocation) or more fine-grained access to individual objects or broad categories of object managed by a service

Scope of role names may be the local domain of the service, or some role names may be organisation-wide, across federated domains

e.g. ***sales-manager*** used in all branches of a world-wide company

police-sergeant used in all of the 52 UK county police forces

NHS-doctor used throughout the UK NHS

RBAC - 2

Administration: note the separation:

principals –> ***roles***, ***roles*** –> ***privileges***

Service developers need only specify authorisation in terms of roles, independently of the administration of principals

e.g. annual student cohort, staff leaving and joining

Principals are authenticated, as always, and must also prove their right to acquire/activate a role. They thus prove they are authorised to use a service

Compare with ACLs – like ACLs containing only group names.

Compare with capabilities – can a capability that proves role membership be engineered?

RBAC seems promising for fast authorisation checking.

RBAC – 3: Parametrised roles

Roles may be parametrised for fine-grained access control to capture:

- relationships between principals:

Policy: "*only the doctor treating a patient may access the medical record*"

e.g. ***treating-doctor (hospital-ID, doctor-ID, patient-ID)***

- patients and others may express exclusions as authorisation policy

e.g. ***doctor (doctor-ID)***

Policy: "*where doctor is not Shipman*", "*where doctor is not <x> (a relative)*"

Compare with ACLs containing only groups, with exclusions of individual members

– semantics of precedence of evaluation in ACLs has always been a difficult area.

RBAC – 4: Role hierarchies

Some RBAC systems define role hierarchies with privilege inheritance up the hierarchy.

The hierarchy may mirror organisational structure, which reflects power and responsibility rather than functional competence.

Privilege inheritance is even less defensible for functional roles.

Also: privilege inheritance violates the *principle of minimum necessary privilege* and makes reasoning about privileges difficult
– see many ACM SACMAT papers)

Role hierarchies are defined in the later NIST RBAC standards.

Our work has avoided privilege inheritance (see OASIS case study).

RBAC – 5: Inter-domain authorisation

RBAC eases authorisation outside principals' home domains, because:

- Roles change less frequently than principals leave and join them
- Administration of users and role membership is separate from service development and use.
- Negotiation on use of services external to domains can be in terms of roles, e.g. payment for a role to use a service
- Federated domains may contain agreed role names in each domain.
  Makes policy easier to negotiate and express.
  e.g. ***sales-department-staff, sales-manager, salesman***

RBAC – 6: Authorisation context

Authorisation policy could include other constraints on use of a role
e.g. time of day, as well as relationships and exclusions.
see OASIS case study – environmental constraints

The privileges associated with a role might not be static.
e.g. ***student (course-ID, student-ID)*** may read solutions to exercises
only after marked work has been returned.

e.g. Conference management system – a small-scale example follows
of use of an external service from a number of domains.

Example: conference management (e.g. Easychair, CMT, EDAS, ...) selection from workflow and policy

Program chair registers names, email addresses, initial password, and roles of the programme committee: roles ***PC-chair(s), PC-member***

all are sent an email asking them to register their account, change their password

Authors submit papers, acquiring role ***contact-author***, returned a UID for the paper

contact-author may submit new versions up to the deadline

PC-members are assigned papers to review. They may delegate some reviews:

role ***reviewer*** per paper, separate from ***PC-member***

Conflicts of interest must be expressed by submitting-authors and PC members, and enforced by the system

PC members must never be able to know the reviewers and see the reviews of their own papers

PC members can see only their own reviews until after the review deadline.

After this, in a discussion phase, PC members may be able to see the ranked order and other reviews (except for their own papers). Systems vary in this respect.

Note: small scale example e.g. 50 PC members, 200 papers

Note: rights will change after deadlines (an example of context)

Design of capabilities/certificates can incorporate RBAC

Traditional capabilities in centralised systems:

object-ID	***rights***

proves the presenter has the rights to the object

RBAC

role	***parameters***

proves the presenter holds the role + parameters
must be checked against access control policy

Capabilities/certificates in distributed systems

check digits = ***f*** (SECRET, protected fields)

protected fields (***object-ID, rights***)	***check digits*** (signature)

RBAC in distributed systems

check digits = ***f*** (SECRET, protected fields)

protected fields (***role, parameters***)	***check digits*** (signature)

RBAC - discussion

RBAC provides:

1. Expressiveness
   - subtle expression of access control policy.
   - if roles are parametrised, exclusions and relationships can be captured.
   - environmental/context checks (time/place) can also be included.

2. Efficiency
   - checking faster than ACLs
   - use of certificate technology comparable with capabilities
   - or use a secure channel and role authentication in source domain

3 Cross-domain interworking
   - easy to negotiate
   - authorisation policy expressible and enforceable
   - heterogeneity of certificates – can check back with issuing domain

OASIS RBAC

Open Architecture for Securely Interworking Services
Case study from Opera Group research

- OASIS services name their clients in terms of **roles**

- OASIS services specify **policy** in terms of **roles**
  - for **role entry** (activation)
  - for **service invocation** (authorisation, access control)

  both in Horn clause form

see: *www.cl.cam.ac.uk/Research/SRG/opera*
for people, projects, publications for download

OASIS model of **role activation**

a role activation rule is of the form:

condition1, condition2, ….. |- target role

where the conditions can be

- prerequisite role
- appointment credential
- environmental constraint

all are parametrised

OASIS (continued) **membership** rules

as we have seen, a role activation rule:

cond1*, cond2, cond3*, ….. |- target role

role membership rule:

the role activation conditions that must **remain true**, e.g.*

for the principal to remain active in the role

monitored using **event-based middleware**

another contributor to an **active security environment**

OASIS model of **authorisation**

An authorisation rule is of the form:

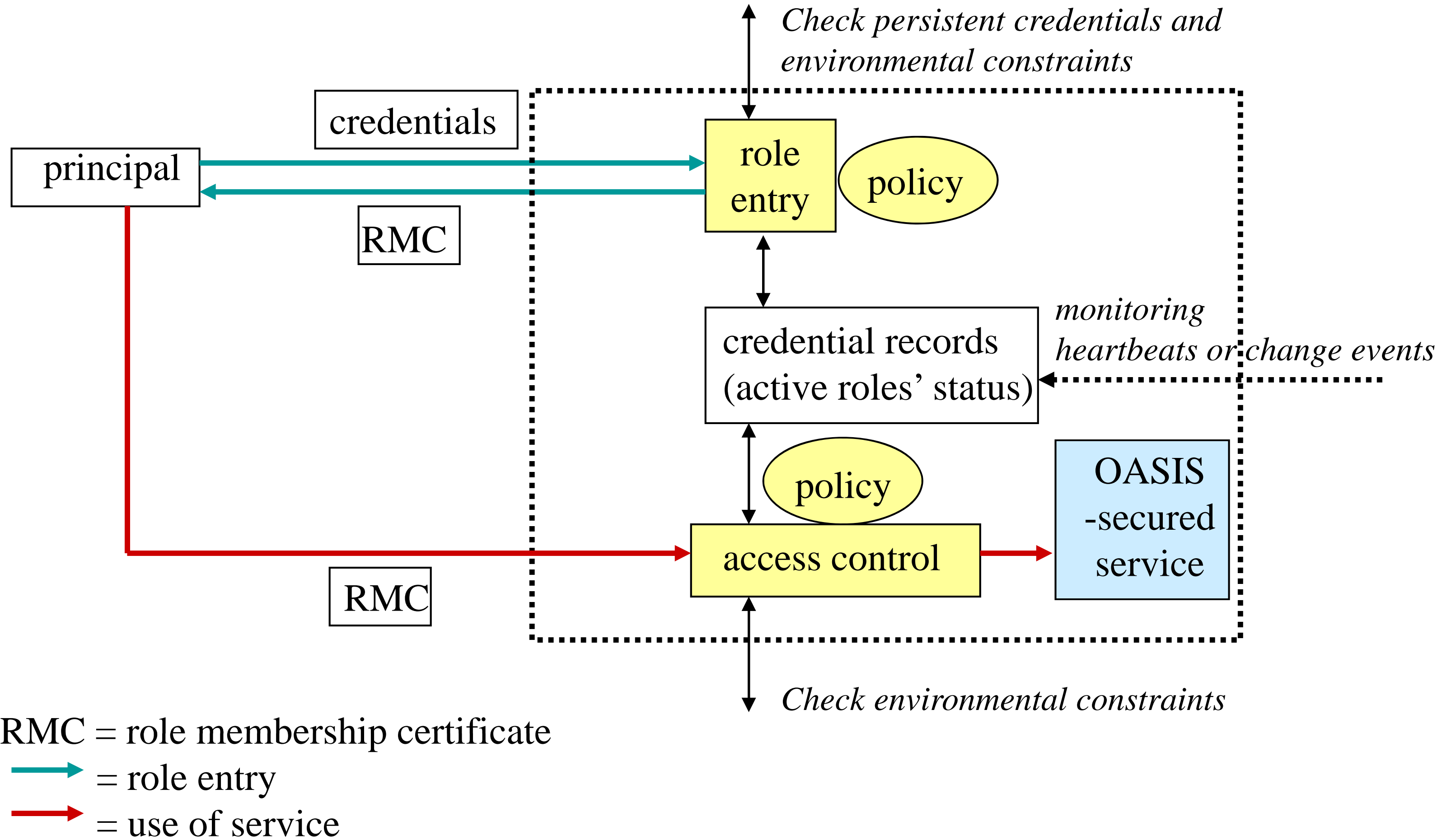
condition1, condition2, ….. |- access

where the conditions can be

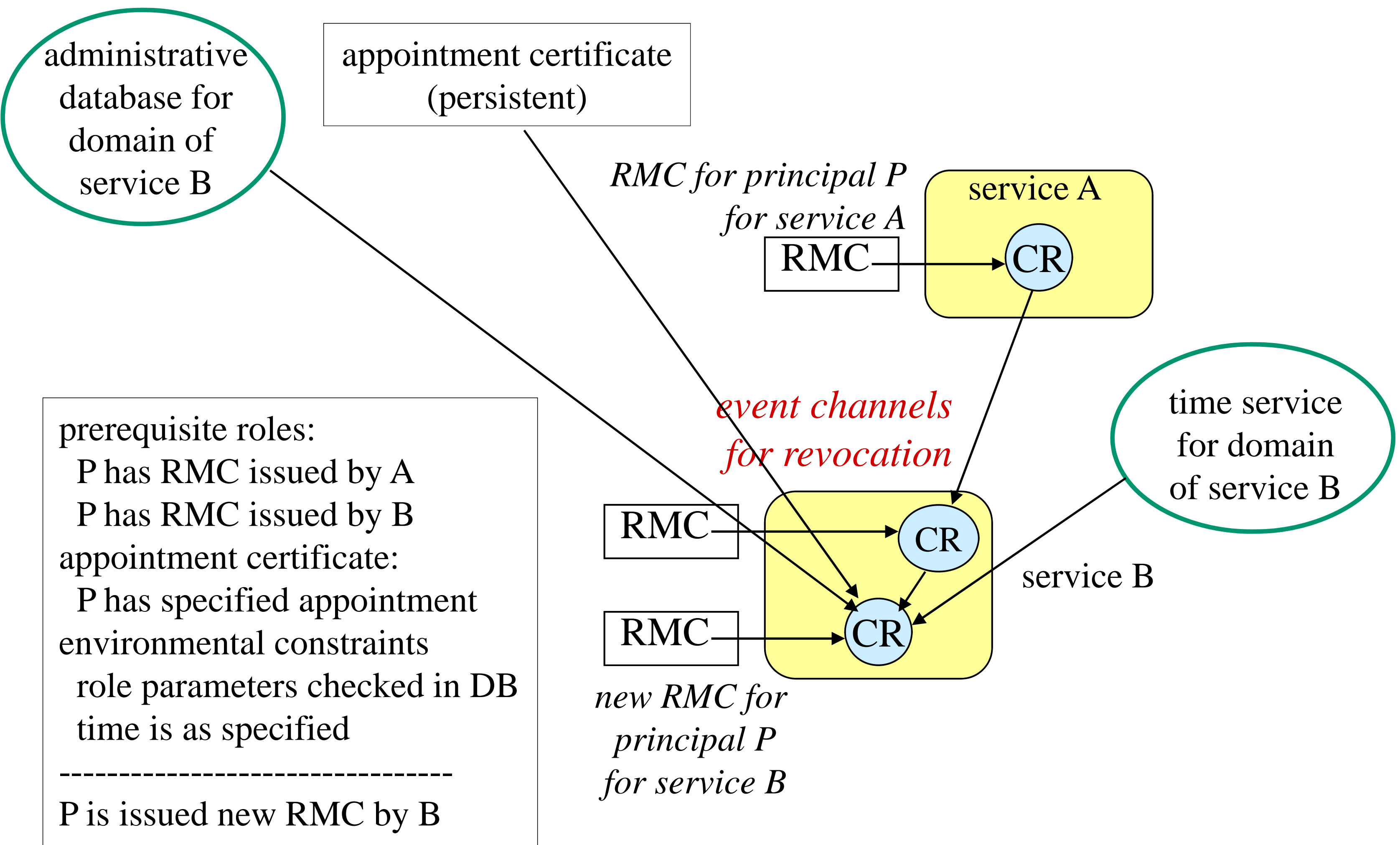
- an active role
- an environmental constraint

all are parametrised

A Service Secured by OASIS Access Control

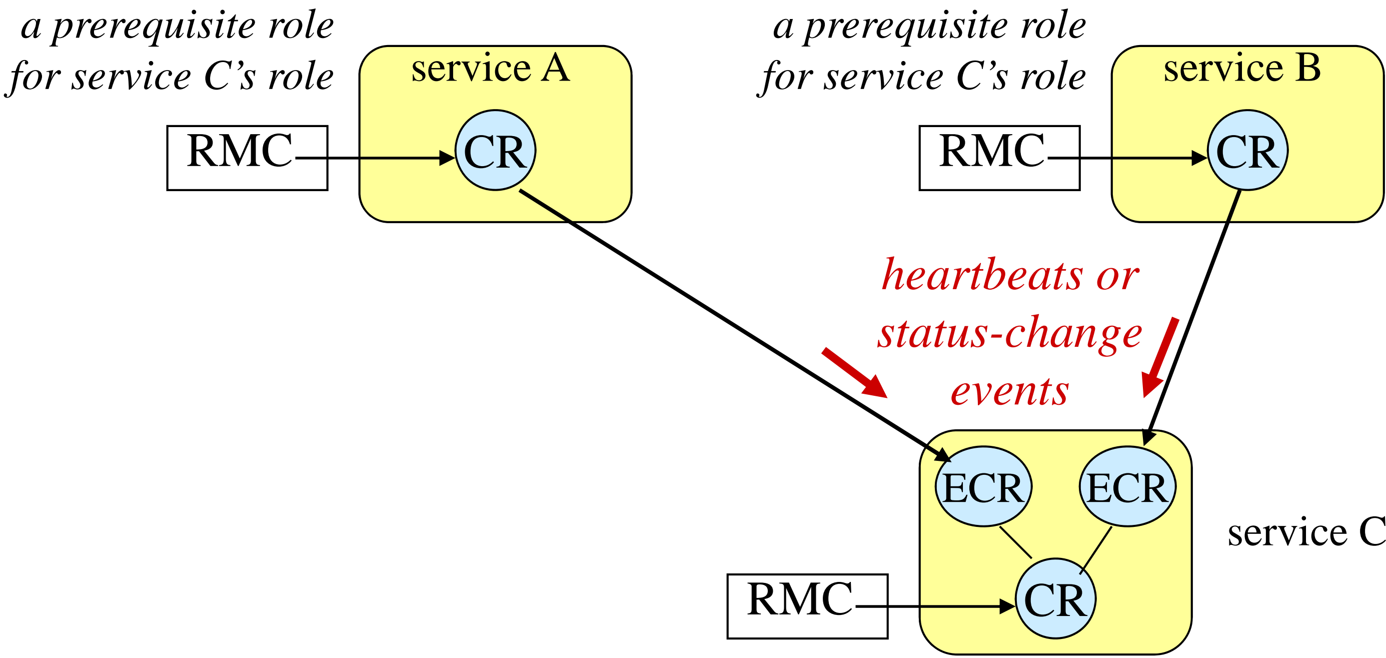


OASIS role activation illustrated



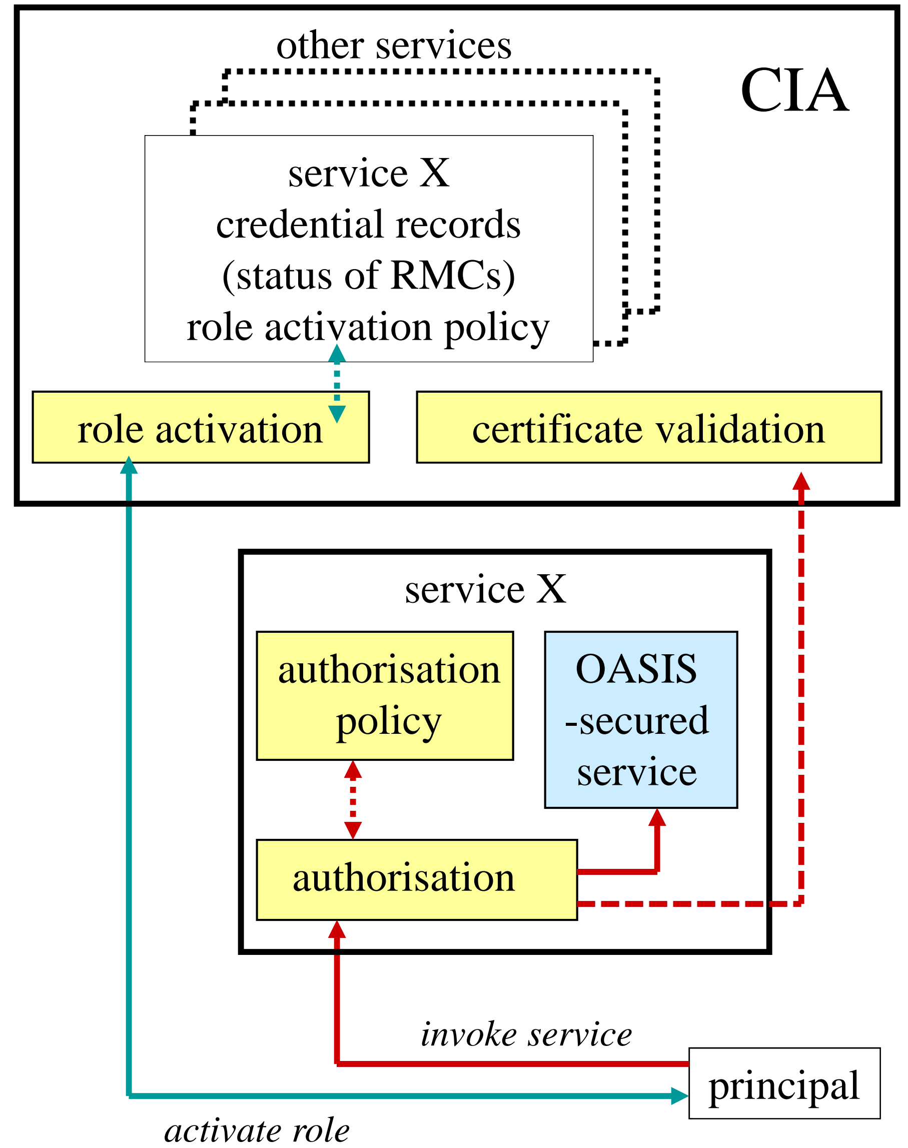
role entry policy specification of service B, in Horn clause form
conditions for principal P to activate some role

Active Security Environment
Monitoring membership rules of active roles



RMC = role membership certificate
CR = credential record
ECR = external credential record

Engineering per-domain certificate issuing and authentication



It is not realistic for every service to manage secrets and issue certificates

The CIA service, for services in its domain:
- keeps the activation polices
- activates roles
- issues and validates certificates
- maintains credential record structures for active roles
-handles revocation via event channels

The CIA service, for services in other domains:
-validates certificates it has issued
- handles revocation of its certificates

OASIS philosophy and characteristics

- Distributed architecture, not a single organisation. Incremental deployment of independently developed services in independent administration domains.
- RBAC for scalability, parametrised roles for expressiveness of policy (e.g. exclusion of values, relationships between parameters).
- Policy expression is per service, per domain
- Roles are activated within sessions. Persistent credentials may be required for role activation.
- Independent designs of RMCs may coexist – service at which RMC is presented checks back with issuer for RMC validation
- Service (domain) level agreements on use of others' RMCs
- Anonymity if and when required
- Immediate revocation on an individual basis
- No role hierarchies with inheritance of privileges

Background on cross-domain authentication

(From slide 2) – here is an outline of some single sign on systems

Raven for use of websites across all the domains of Cambridge University
- common naming of principals (CRSIDs, nested domains)
- authentication is sufficient for authorisation

Shibboleth organisation-centric.
organisation negotiates use by its members of external services

OpenID user-centric
used by many large websites (BBC, Google, MySpace, PayPal, ...)

Raven

- Aim: avoid proliferation of passwords for UCam web services
  - Raven is a Ucam-webauth Single Sign On system instance
  - Developed within Cambridge (by Jon Warbrick)

- Three parties in the Ucam_webauth protocol:
  - User's web-browser
  - Target web-server
  - Raven web-server

- Authentication token passed as an HTTP cookie
  - Thus should be passed using HTTPS… but often isn't

Example Raven dialogue

- User requests protected page
- Target web-server checks for Ucam-WLS-Session cookie
- If found, and decodes correctly, page is returned. Done.

- Otherwise, redirect client browser to Raven server
  - Encodes information about the requested page in the URL
- Raven inputs and checks credentials
  - (Also permits users to "cancel")
- Raven redirects client browser to the protected page. Done.
  - (An HTTP 401 error will be generated if users cancelled)

Raven coordinates participants using time

- Target web-server verifies Ucam-WLS-Session cookie
  - Public-key of Raven server pre-loaded on target web-server
- Target web-server and Raven do not interact directly
  - Client browser receives, stores and resends cookies
- What about malicious client behaviour or interception?
  - e.g. replay attacks?
- Raven requires time-synchronisation
  - A site-specific clock-skew margin can be configured

Shibboleth provides federated authentication

- System for federated authentication and authorisation
  - Internet2 middleware group standard
  - Implements SAML: Security Assertion Markup Language
  - Facilitates single-sign-on across administrative domains

- Raven actually speaks both Ucam-webauth and Shibboleth
  - Shibboleth has the advantage of wider software support

- Identity providers (IdPs) supply user information
- Service providers (SPs) consume this information and get access to secure content

Shibboleth exchange

- Similar to Raven, but with some extra indirection
  - User requests protected resource from SP
  - SP crafts authentication request
  - User redirected to IdP or 'Where Are You From' service
    - E.g. UK Federation WAYF service
  - User authenticates (external to Shibboleth)
  - Shibboleth generates SAML authentication assertion handle
  - User redirected to SP
  - SP may issue AttributeQuery to IdP's attribute service
  - SP can make access control decision

OpenID

- Another cross-domain single-sign-on system
- Shibboleth is organisation-centric
  - Organisations must agree to accept other organisations' statements regarding foreign users
  - Lots of support within the UK Joint Information Systems Committee (JISC) for accessing electronic resources
- OpenID is user-centric
  - Primarily about identity
  - OpenIDs are permanent URI or XRI structures

OpenID (cont)

- User provides their ID to relying party web site
  - OpenID 1.0 retrieves URL, learns identity provider
  - OpenID 2.0 retrieves XRDS, learns identity provider
    - XRDS/Yadis indirection affords greater flexibility

- Many big commercial players offer OpenID assertions
- Lots of open source software support for OpenID also

- In terms of responsibility, consider use for:
  - Access to a web resource
  - Access to a wireless network

Storage services

Consider various computing environments and scenarios

Professional, academic, commercial, home – *based on traditional wired networks*

Mobile users with computing devices – may be *internet-connected and/or using wireless/ad.hoc*

Pervasive/active environments – sensor networks' logs/databases - *wired and wireless networks*

Some scenarios
(consider domain architecture, naming, location, authentication, authorisation, communication)

1. Single domain behind firewall – local files served by network-based file service
   plus accessing remote files and services.

2. Digital libraries, copyright, professional societies, publishers: scientific archive.
   issues: persistence of data through technology change; persistence of scientific archive
   who guarantees long term persistence?

3. Internet-based, cooperative P2P file-storage

4. GRID/cloud: storage for e-science applications

5. Commercial data centres

Examples of requirements for storage

Traditional environments

- program/document storage and development. Loading and running programs.
  Run-time data access and storage.

- application services (local and remote) need storage. Databases, CAD, versioning systems (SVN), email, newsgroups, naming directories, applications for download.

Mobile users

- detached operation: copy, disconnect, remote-work, reconnect, synchronise files – conflicts?
- access to files from remote locations – secure connection to home domain
- or use an internet-based service to place files close to where they will be used?

Peer-to-Peer (P2P)

- use spare capacity across the Internet for file storage, backup/archive
  e.g. Ocean Store from Berkeley, built on Planet Lab.
- Cooperative model e.g. music and film "sharing" e.g. Gnutella, Napster

Storage for e-science (grid -> cloud computing)

- e.g. petabytes of astronomic or genomic data and storage for computations on such data
- e.g. public data such as EHRs (security/trust is critical)

File structure, media types, indexing and retrieval,

Should a storage service provide support for structure representation, indexing and retrieval?

e.g. web-pages are composite documents, containing links to images.
Should general file services support structured files, as opposed to the byte-sequence abstraction?
Aim to avoid storing multiple copies of large image objects.
Issue: persistence of objects linked to – "40% of URLs fail after 2 years" (dangling references).

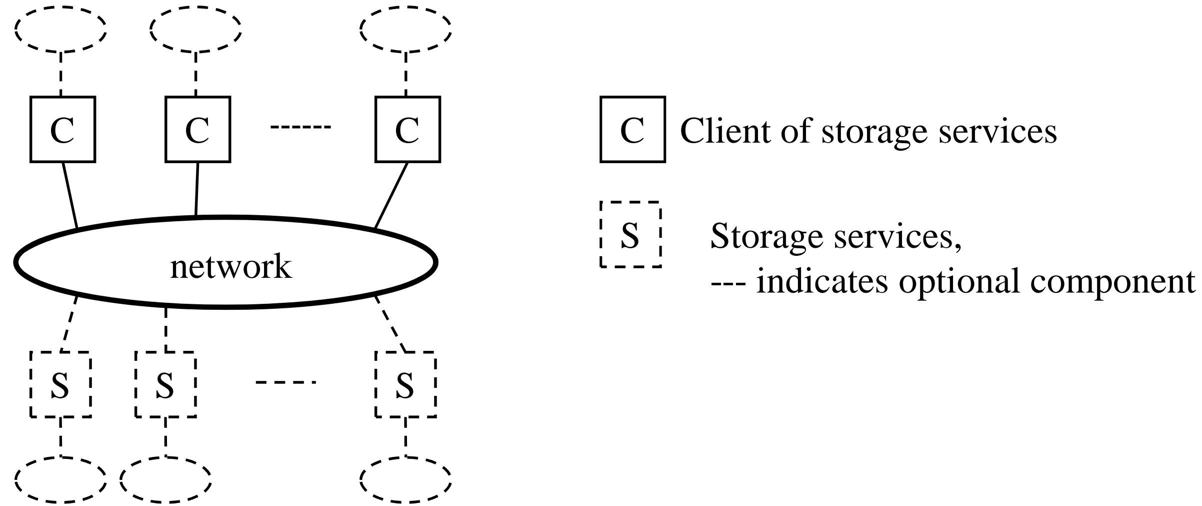
e.g. Cooperative work / versions e.g. SVN, records structure above the basic file abstraction
- aids synchronisation of updates and retrieval of any version.

e.g. Video/film stores, and content-delivery networks, deal in unstructured files but partition into blocks for transmission and reassembly.
e.g. Large collections e.g. photographs, e.g. videos of "my day" for memory-loss patients,
e.g. Audits of professional caring (NHS, SS) or business activities, process for suspicious or dangerous behaviour. Determine patterns that may imply fraud.
e.g. Logs of sensor data recording traffic, pollution, building projects (tunnels (Arup CTRL))
The data is analysed statistically to extract behaviour in context
– to determine which factors are significant

Should we use a database if we need to capture structure? e.g. for data mining?
We may only need to know external links

Storage services from first principles

First consider a professional, network-based environment in a single domain



1. (thin) clients have no local storage. System provides shared storage servers.
   e.g. early V system at Stanford, e.g. network computers

2. Clients have local storage. There are no dedicated storage services.
   would need e.g. Unix ***mount*** to achieve a shared filing system

3. Clients have local storage and there are also shared storage servers.
   Client discs used for?
   - Private desktop e.g. Xerox, Apple Mac, Windows
   - system files for bootstrapping
   - cached files – first-class copy is in shared service
   - temporary files not backed up by sys-admin

Storage service functionality

- open or closed? Is it bound into a single OS file system model e.g. pathname format

- how is functionality distributed?

  where is directory service for pathname resolution and access control?
  existence control / garbage collection? (reachability is via the directory graph)
  concurrency control? (based on knowledge (state) of what is open and mode)

- level of interface?
  - remote block server

    e.g. early RVD (remote virtual disc)
    client system may do block allocation within an allocated partition (for minimal overhead at server) or server does allocation
    e.g. current video servers distribute blocks to achieve low latency
  - remote, UID-named files. Interactions may involve whole files or parts of files.

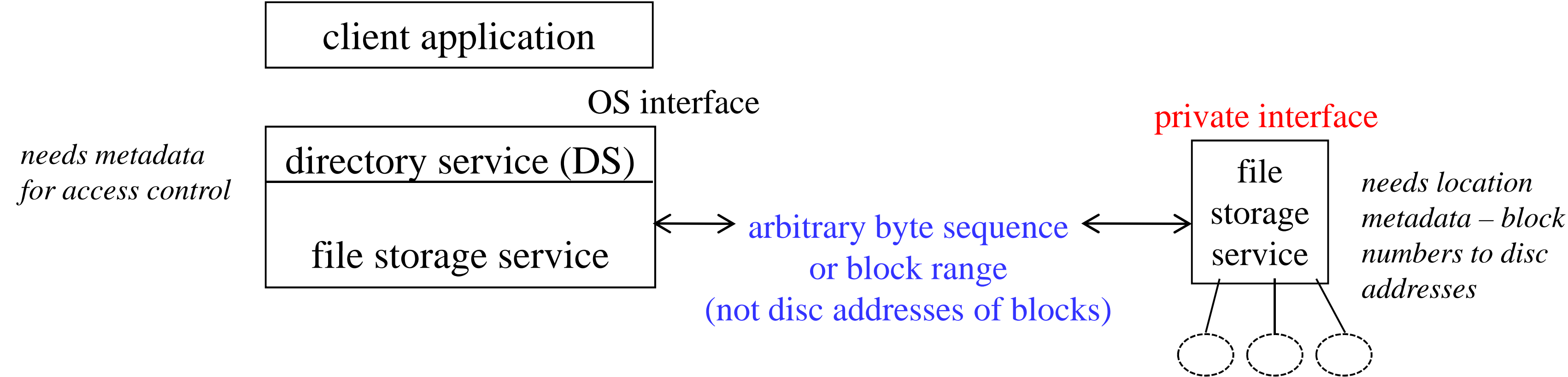
    server does block allocation – server overhead.
  - remote path-named files bound into a single OS naming scheme

- caching and replication

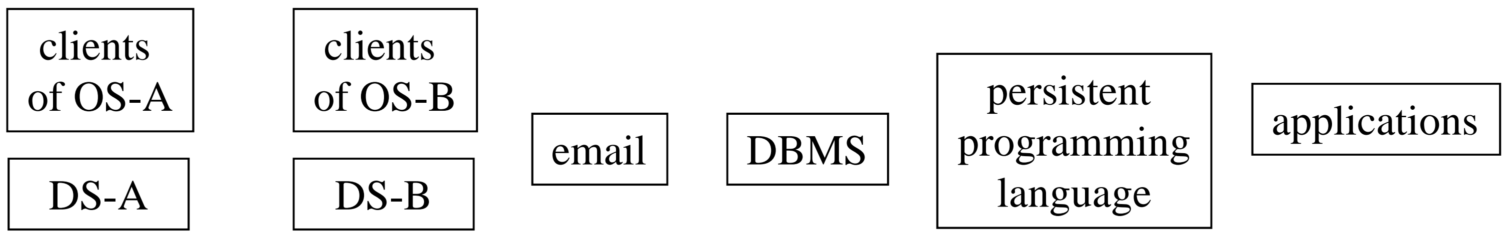
  is the service responsible for managing, or assisting with
  - multiple cached copies of a file at different clients?
  - replicas of a file (replicated by servers for reliability)?

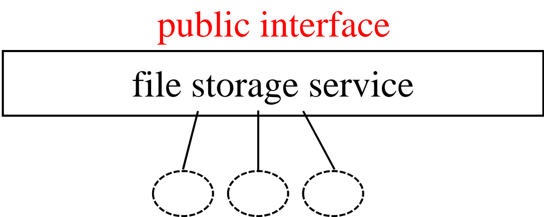
Storage service architectures

a) Closed storage architecture (single OS accesses storage service (SS))



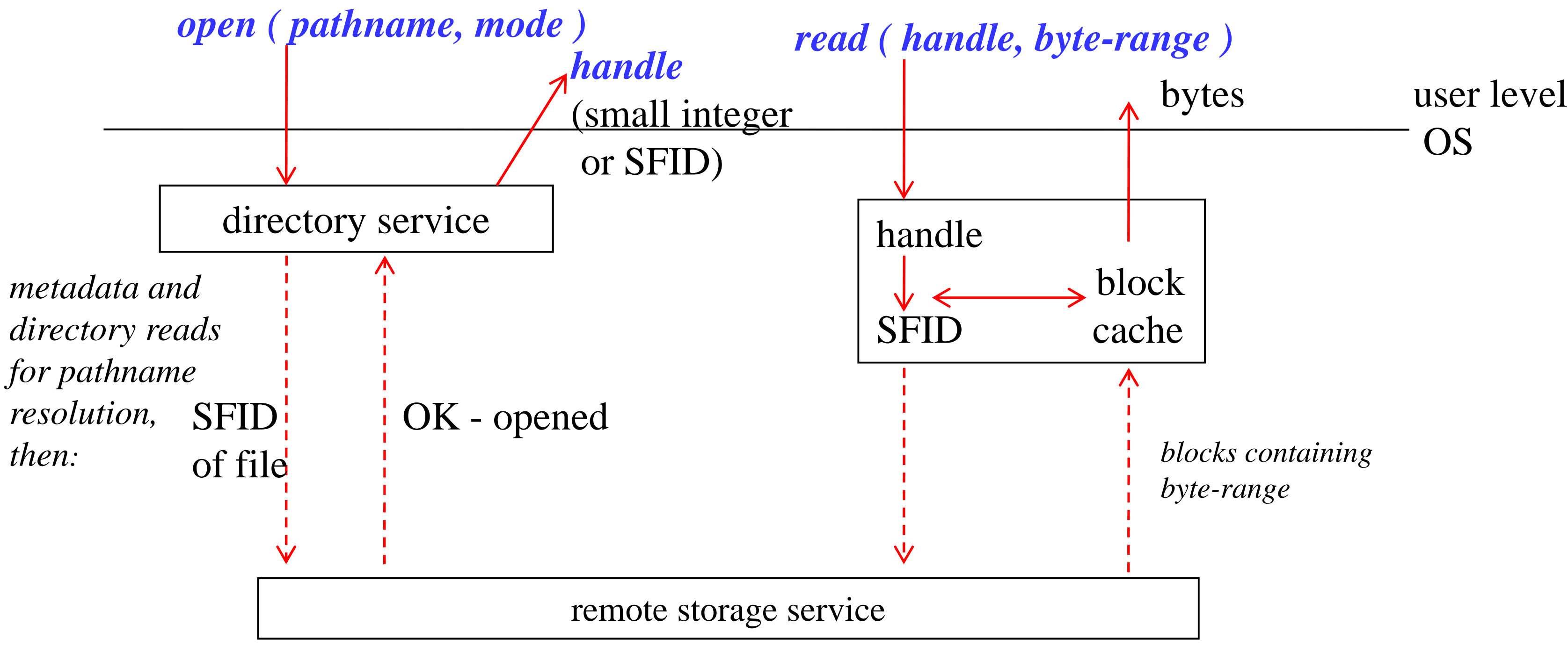
b) Open storage architecture





Remote SS interface at file storage (byte-sequence) level

SFID = system file identifier



*If the SS is stateless it does not support **open** at its interface*
and holds no info' on files in use.
Pathname resolution is still required, to obtain the SSID (hard link) of the file.

Operations in remote storage service interface

SFID ← ***create***

read	***(SFID, byte-range)***		*assumes interaction at byte-sequence level*
write	***(SFID, byte-range)***		*as above, rather than whole file*
delete	***(SFID)***	?	
lock	***(SFID)***	?	
unlock	***(SFID)***	?	? *are design decisions*
open	***(SFID)***	?	
close	***(SFID)***	?	

Does the service hold state?

- NO – specified as stateless
  - simple crash recovery
  - can't help with concurrency control
  - can't help with cache management

- YES – interface supports ***open/close***, records who has files open and access mode
  - server crash recovery – needs to interact with clients to rebuild its state
  - support for concurrency control by client OSs
    exclusive/shared locks better than single-writer/multiple-reader
  - support for cache management if clients store whole-file copies locally
    can *notify* holders of copies when a new version is written
    otherwise high traffic from clients requesting status of cached items

Existence control and garbage collection

A file should stay in existence for as long as it is reachable from the root of the directory naming graph

- Lost object problem ***SFID*** ⟵ ***create (...)***
  server allocates metadata in persistent store
  either: **server crash**
  or: reply (***SFID***) reaches client's main memory only
  client crash
  on server or client restart, client repeats ***create (...)*** and gets a new ***SFID***

- Storage service at file level can't help – doesn't see naming graphs

- A directory service can do existence control for its own objects.
  Multiple instances of the DS would have to cooperate to traverse the graph.
  This would work for a closed architecture and for a single OS's files within an open architecture.

- What about - objects shared across systems e.g. video clip in document?
  - objects not stored in directories?

- Consider a ***touch*** operation provided by the storage service. All its clients i.e. services, not users, must traverse their naming graphs and ***touch*** all their files periodically.
  Untouched files are deleted (archived).

An early case study: The Cambridge File Server (CFS)

Developed as part of the Cambridge Distributed Computing System (CDCS) in the late 1970s
CDCS was used as the Lab's research environment throughout the 1980s.

http://www.research.microsoft.com/NeedhamBook/cmds.pdf

Andrew Birrell and Roger Needham "A Universal File Server"
IEEE Trans SE 6 (5), pp 450-453, May 1980

CFS design features:

- open architecture, many OS clients e.g. Tripos, CAP, research file systems
- minimal support for structure without enforcing path-naming
- some operations with transactional semantics to avoid lost objects
- no delete operation
- garbage collection run from any processor bank machine

CFS basic concepts

CFS provides

two primitive types: ***byte*** and ***UID***

(***PUID*** = persistent ***UID, TUID*** = transient ***UID*** for open objects)

two abstractions: ***file*** – an uninterpreted sequence of bytes

named persistently by a ***PUID*** with a random component

index – a sequence of ***PUID***s, itself named by a ***PUID***

Index – used by CFS's clients to mirror their directory structures.

all index operations are transactional (failure-atomic – all-or-nothing)

- Existence control

  indexes form a general naming graph starting from a specific root index

  objects are preserved while they are reachable from the root

  - reference counts are used, recording the number of times a ***PUID*** is preserved in an index

  - an asynchronous garbage collector is used to detect cyclic structures

- Concurrency control – just MRSW

some CFS operations

file operations:

PUID ← create-file (index-PUID, entry, ...) % must store new ***PUID*** in existing index
% transaction avoids lost object problem
TUID ← open-file (PUID, read/write) % ***TUID*** = temporary ***UID*** for open file
data ← read (TUID, offset, amount)
done ← write (TUID, offset, amount, data)
done ← close-file (TUID)
NOTE – no ***delete-file*** operation, garbage collection instead

index operations:

The index operations are used by OS clients to mirror the directory operations they offer their users.

PUID ← create-index (index-PUID, entry) % must store new ***PUID*** in existing index
% transaction avoids lost object problem
TUID ← open-index (PUID, read/write) % ***TUID*** = temporary ***UID*** for open index
done ← close-index (TUID)
done ← preserve (index-PUID, entry, object-PUID) % put a link to an object in an index
PUID ← retrieve (index-PUID, entry) % extract a link from an index
done ← delete-entry (index-PUID, entry) % remove a link from an index
NOTE – no ***delete-index*** operation, garbage collection instead

From Internet Data Centers to Data Centers in the Cloud

This case study is a short extract from a keynote address given to the Doctoral Symposium at Middleware 2009

by Lucy Cherkasova of HP Research Labs Palo Alto.

The full keynote is on the course materials page

The keynote focus is ***performance modelling***

- Data Centers Evolution
  - Internet Data Center
  - Enterprise Data Centers
  - Web 2.0 Mega Data Centers

Data Center Evolution

- Internet Data Centers (IDCs first generation – per company)
  - Data Center boom started during the dot-com bubble
  - Companies needed fast Internet connectivity and an established Internet presence
  - Web hosting and co-location facilities for company's services
  - Challenges in service scalability, dealing with flash crowds, and dynamic resource provisioning
    - New paradigm: everyone on the Internet can come to your web site!
  - Mostly static web content
    - Many results on improving web server performance, web caching, and request distribution
  - Web interface for configuring and managing devices (products sold by company)
  - New pioneering architectures such as
    - Content Distribution Network (CDN),
    - Overlay networks for delivering media content

Content Delivery Network (CDN)

- High availability and responsiveness are key factors for business Web sites
- "Flash Crowd" problem
- Main goal of CDN's solution is
  - overcome server overload problem for popular sites,
  - minimize the network impact in the content delivery path.
- CDN: large-scale distributed network of servers,
  - Surrogate servers (proxy caches) are located closer to the edges of the Internet a.k.a. edge servers
- Akamai is one of the largest CDNs
  - 56,000 servers in 950 networks in 70 countries
  - Deliver 20% of all Web traffic

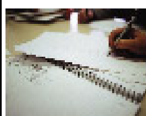
Retrieving a Web Page

Support Information

Global Support Organizations

Support for Radix applications is provided by several different support teams in each of the regions. Some applications are supported by a virtual global team, while others are supported by teams within each region.

More



Radix Applications and GIO Support Models

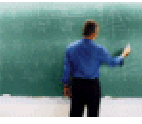
The GIO support teams provide support for the Radix applications and infrastructure. Some applications and infrastructure are supported by a virtual global team and/or by teams within each region.

More

Data Administration

The GIO Data Administration team is responsible for the maintenance of data within the Radix environment. They support the MS Regional Delivery and Customer organizations to load and modify configuration information and data used by the Radix applications.

More



Radix Tool Component Definitions

The Radix tool component description documents are overviews of the components of Radix from a service and support perspective. A high level service description, a break down of the infrastructure support dependencies and a list of the service requests are included. The primary audience for these documents are the GIO Radix support teams.

More

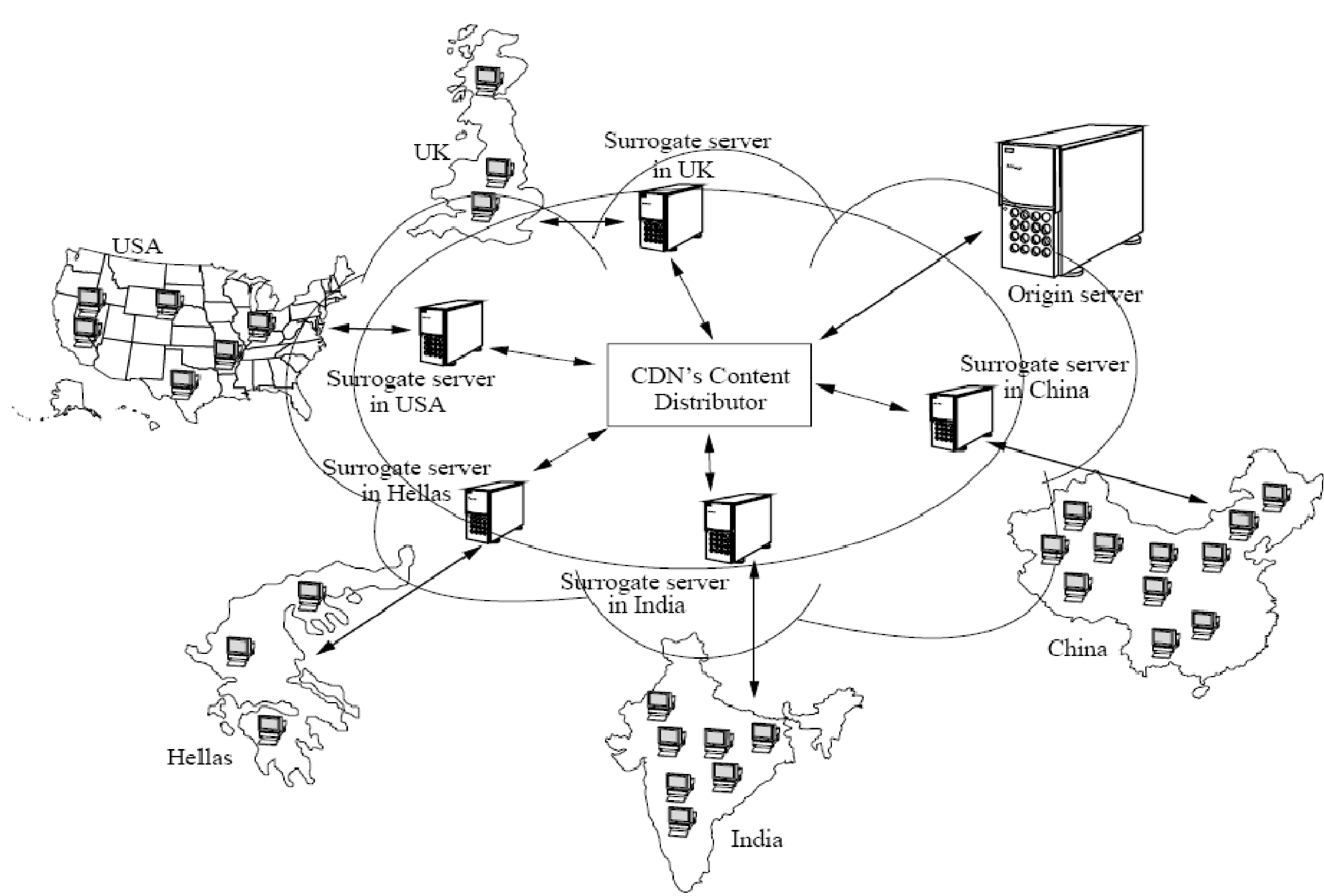
Web page is a composite object:

- HTML file is delivered first
- Client browser parses it for embedded objects
- Send a set of requests for these embedded objects
- Typically, 80% or more of bytes of a web page are images
- 80% of the page can be served by a CDN.

CDN’s Design

- Two main mechanisms
  - URL rewriting
    - <img src =http://www.xyz.com/images/foo.jpg>
    - <img src =http://akamai.xyz.com/images/foo.jpg>
  - DNS redirection
    - Transparent, does not require content modification
    - Typically employs two-level DNS lookup to choose most appropriate edge server (***name -> list of edge servers, selected list item -> IP address***)

CDN Architecture

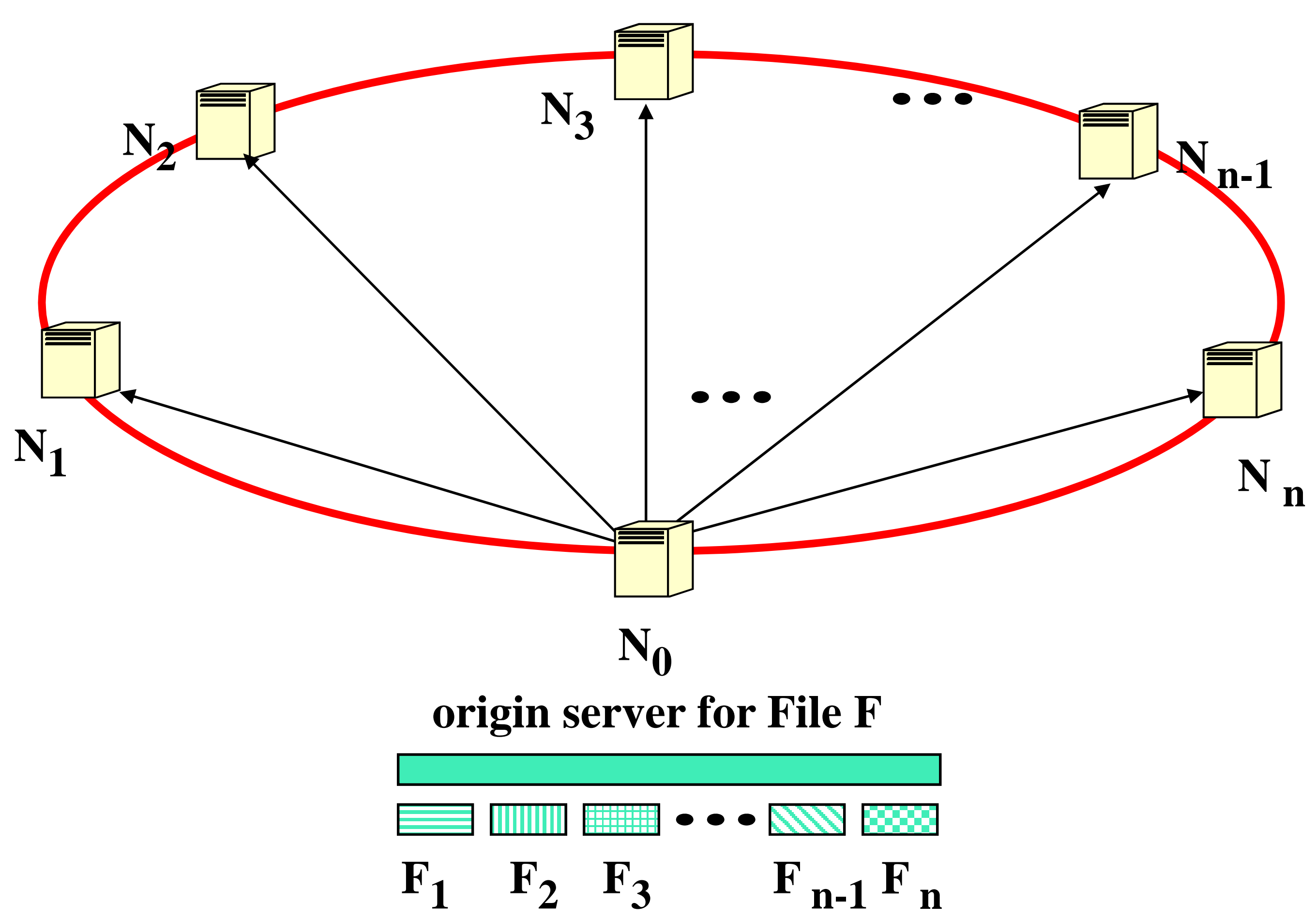


CDN Research Problems

- Efficient large-scale content distribution
  - large files,
  - video on demand, streaming media

    low latency, real-time requirement
    - FastReplica for CDNs
    - BitTorrent (general purpose)
    - SplitStream (multicast, video streaming)

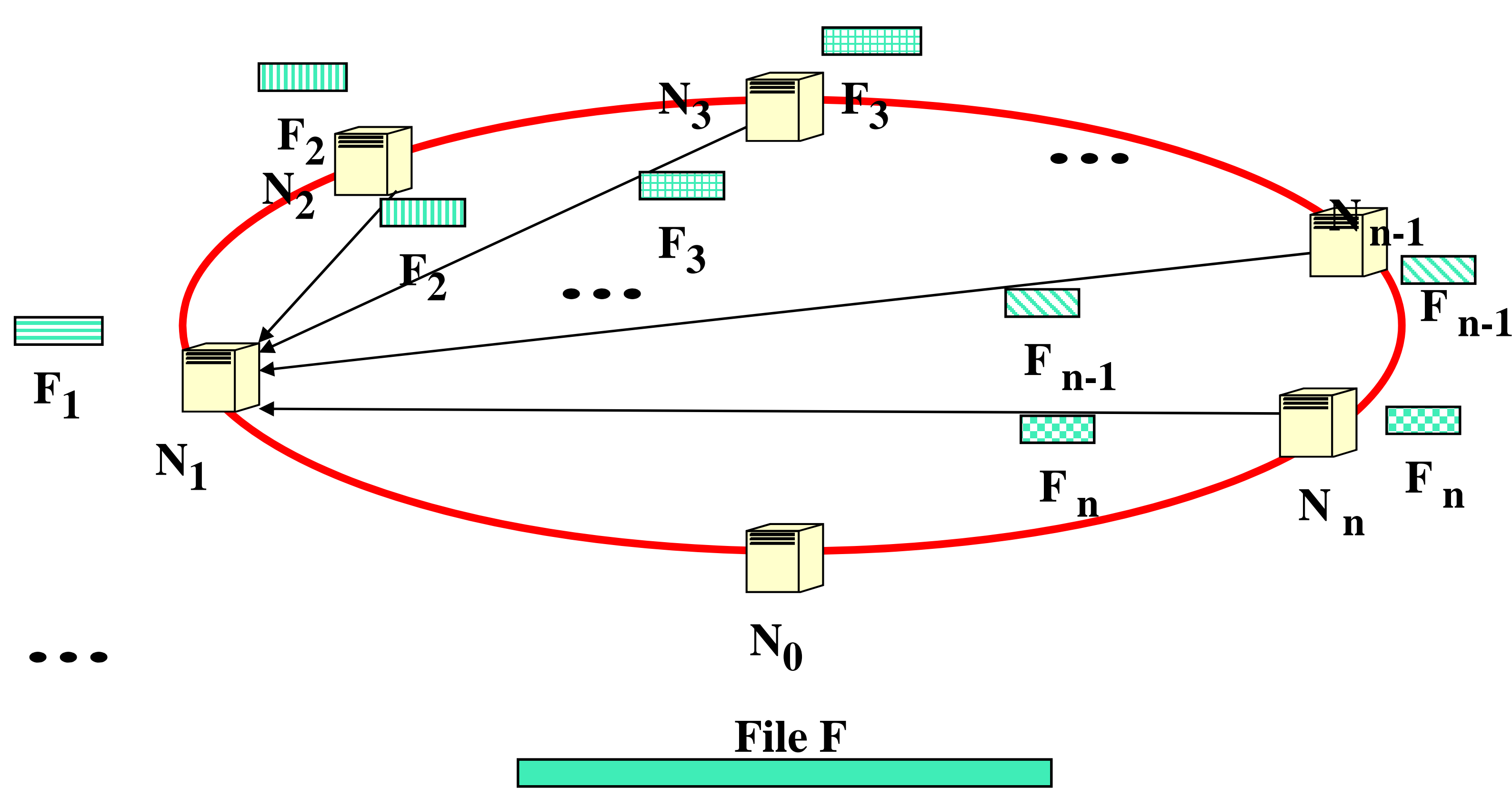
FastReplica: Distribution Step



L. Cherkasova, J. Lee. *FastReplica: Efficient Large File Distribution within Content Delivery Networks*

Proc. of the 4th USENIX Symp. on Internet Technologies and Systems (USITS'2003).

FastReplica: Collection Step



Remaining Research Problems

Some (2009) open questions:

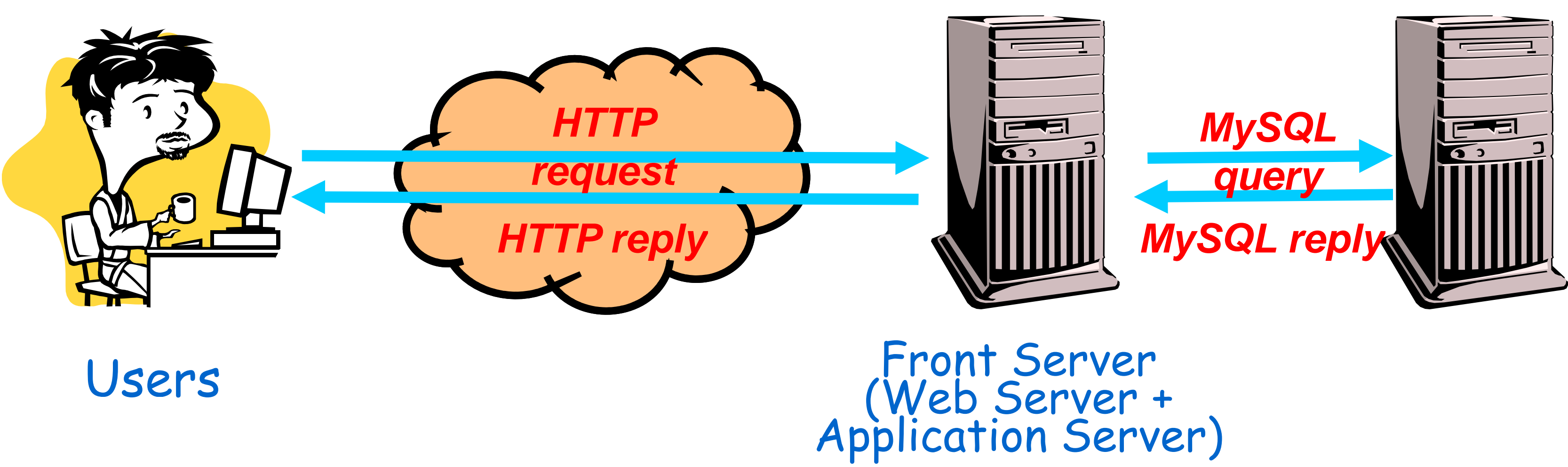
- Optimal number of edge servers and their placement
  - Two different approaches:
    - *Co-location*: placing servers closer to the edge (Akamai)
    - *Network core*: server clusters in large data centers near the main network backbones (Limelight and AT&T)
- Content placement
- Large-scale system monitoring and management
  - to gather evidence as a basis for design decisions

Data Center Evolution

- Enterprise Data Centers
  - New application design: multi-tier applications - database integration, see next slide
  - Many traditional applications, e.g. HR, payroll, financial, supply-chain, call-desk, etc, are re-written using this paradigm.
  - Many different and complex applications
  - *Trend: Everything as a Service*
    - Service oriented Architecture (SOA)
  - Dynamic resource provisioning within a large cluster
  - Virtualization (datacenter middleware)
  - Dream of Utility Computing:
    - Computing-on-demand (IBM)
    - Adaptive Enterprise (HP)

Multi-tier Applications

- Enterprise applications:
  - Multi-tier architecture is a standard building block



Example: Units of Client/Server Activity

- Session:

  A sequence of individual transactions issued by the same client

- Concurrent Sessions = Concurrent Clients

- Think time:

  The interval from a client receiving a response to the client sending the next transaction

Data Growth

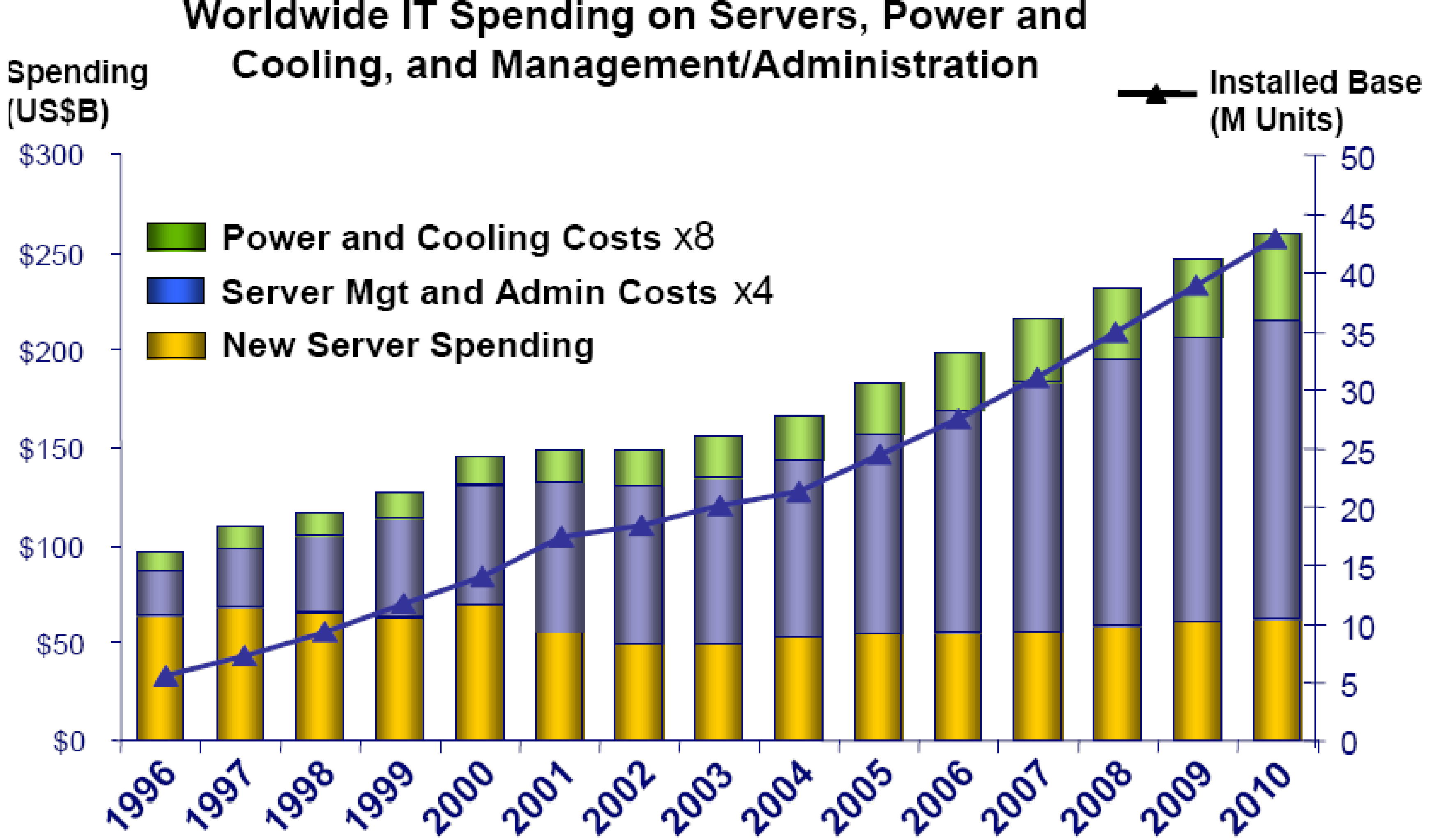
- Unprecedented data growth:
  - The amount of data managed by today's Data Centers quadruples every 18 months
- New York Stock Exchange generates about 1 TB of new trade data each day.
- Facebook hosts ~10 billion photos (1 PB of storage).
- The Internet Archive stores around 2PB, and it is growing at 20TB per month
- The Large Hadron Collider (CERN) will produce ~15 PB of data per year.

Big Data

- IDC estimate the size of "digital universe" :
  - 0.18 zettabytes in 2006;
  - 1.8 zettabytes in 2011 (10 times growth);
- A zettabyte is 10^{21} bytes, i.e.,
  - 1,000 exabytes or
  - 1,000,000 petabytes
- Big Data is here
  - Machine logs, RFID readers, sensors networks, retail and enterprise transactions
  - Rich media
  - Publicly available data from different sources
- New challenges for storing, managing, and processing large-scale data in the enterprise (information and content management)
  - Performance modeling of new applications

Worldwide Server Market:

Cost of Management and Power Ramps Dramatically



Source: IDC, 2008

Data Center Evolution

- Data Center in the Cloud
  - Web 2.0 Mega-Datacenters: Google, Amazon, Yahoo
  - Amazon Elastic Compute Cloud (EC2)
  - Amazon Web Services (AWS) and Google AppEngine
  - New class of applications related to ***parallel processing*** of large data
  - Google's Map-Reduce framework (with the open source implementation Apache Hadoop)
    - *Mappers* do the work on data slices, *Reducers* process the results
    - Handle node failures and restart failed work
  - One can rent ones own Data Center in the Cloud on a "pay-per-use" basis
  - Cloud Computing: Software as a Service (SaaS) + Utility Computing

Existing and New Technologies

Existing Technology New Applications Google Scholar Facebook	New Technology New Applications Middleware for multi-tier apps Virtualization, Map-Reduce Existing Applications
Existing Technology Existing Applications html, http Web Servers/Web Browsers Google, E-commerce	New Technology Existing Applications html, http Web Servers/Web Browsers Google, E-commerce

Event-driven communication paradigm

- asynchronous message-passing rather than request-reply
- advertise, subscribe, publish/notify for scalability

  e.g. subscribe to and be notified of:

  bus-seen-event (busID=uni4., location=*)*
- event-driven paradigm for ubiquitous computing: sensors generate data, notified as events
- compose/correlate events for higher level semantics

  e.g. traffic congestion, pollution and traffic
- database integration – how best to achieve it?

Event-Driven Systems CEA

Cambridge Event Architecture (CEA), 1992 -

- extension of O-O middleware, typed events
  - "advertise, subscribe, publish/notify", direct or mediated,
  - publishers (or mediators if >1 publisher for a type) process subscription filters and multicast to relevant subscribers
- federated event systems:
  - gateways/contracts/XML
- applications:
  - multimedia presentation control
  - pervasive environments (active house, active city, active office)
  - tracking mobile entities (active badge technology)
  - telecommunications monitoring and control

Event-Driven Systems - Hermes

- Hermes large-scale event service, 2001-4
  - PhD work of Peter Pietzuch at CL

- loosely-coupled
- publish/subscribe
- widely distributed event-broker network
- via a P2P overlay network (DHT e.g. Pastry), see slide 5
- distributed filtering (optimise use of comms.)
- rendezvous nodes for advertisers/subscribers

Use of P2P/DHT substrate

- Broker IP addresses hashed into 128-bit space
- Event topics hashed into 128-bit space. Topic is managed by broker with nearest value > topic hash
- Brokers keep tables of nearest neighbours (for different common prefixes) in 128-bit space – see next slide
- Event messages from pubs and subs routed to broker nearest to event topic's hash value in O(logN) hops – called the "rendezvous node" for that topic
- Paths to same destination converge quickly
- Paths shared to nearby destinations – late fanout
- Resilient to join/leave/failure of nodes
- Scales to millions of nodes

Pastry node 2030xx…'s routing table starts:

0* Id,a	1* Id,a	2*	3* Id,a
20*	21* Id,a	22* Id,a	23* Id,a
200* Id,a	201* Id,a	202* Id,a	203*
2030* etc.	2031* Id,a	2032* Id,a	2033* Id,a

e.g. route to 1xxxx…

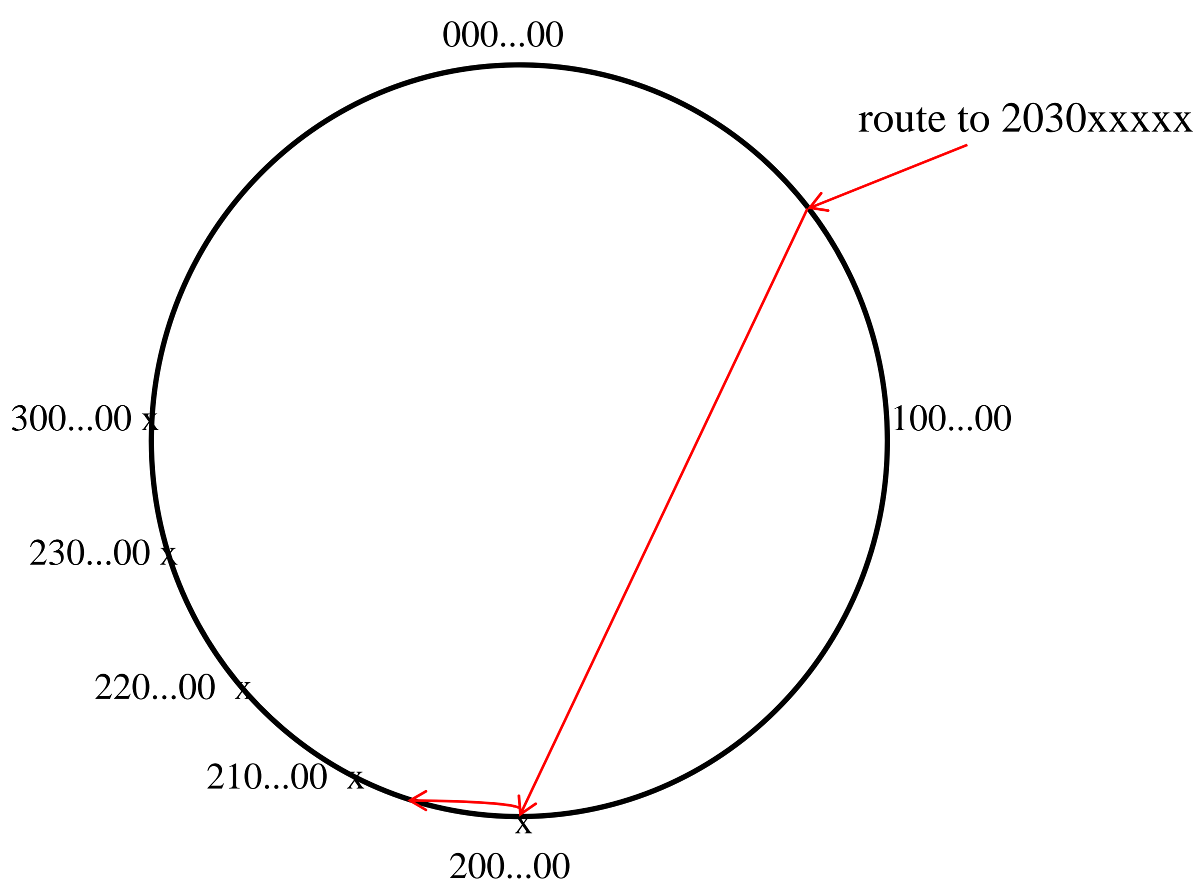
e.g. route to **22**xxx…

e.g. route to **200**xxx**…**

e.g. route to **2032**xx**…**

- nodeIds and keys are in some base 2^b (e.g. b=2 here)
- each entry, except those for itself, contains the 'Id' and IP address 'a' of another node

Pastry route convergence, e.g.using base 2

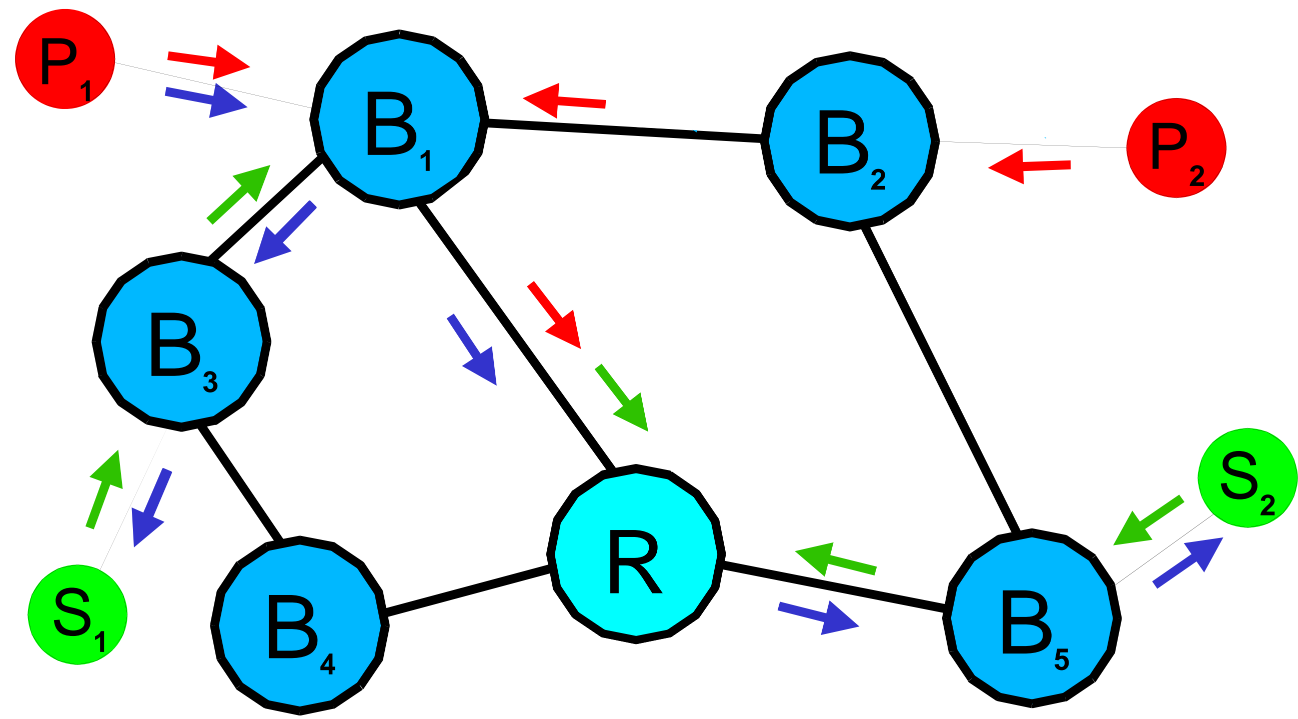


Hermes Pub/Sub Design

- Event Brokers (B)
  - provide middleware functionality
  - logical overlay P2P network: content-based routing+filtering
  - easily extensible
- Event Clients: Publishers (P), Subscribers (S)
  - connect to any Event Broker
    - publishers **advertise**,
    - subscribers **subscribe** (brokers set up routing state),
    - publishers **publish**,
    - brokers route messages and **notify** publications to subscribers
  - lightweight, language-independent

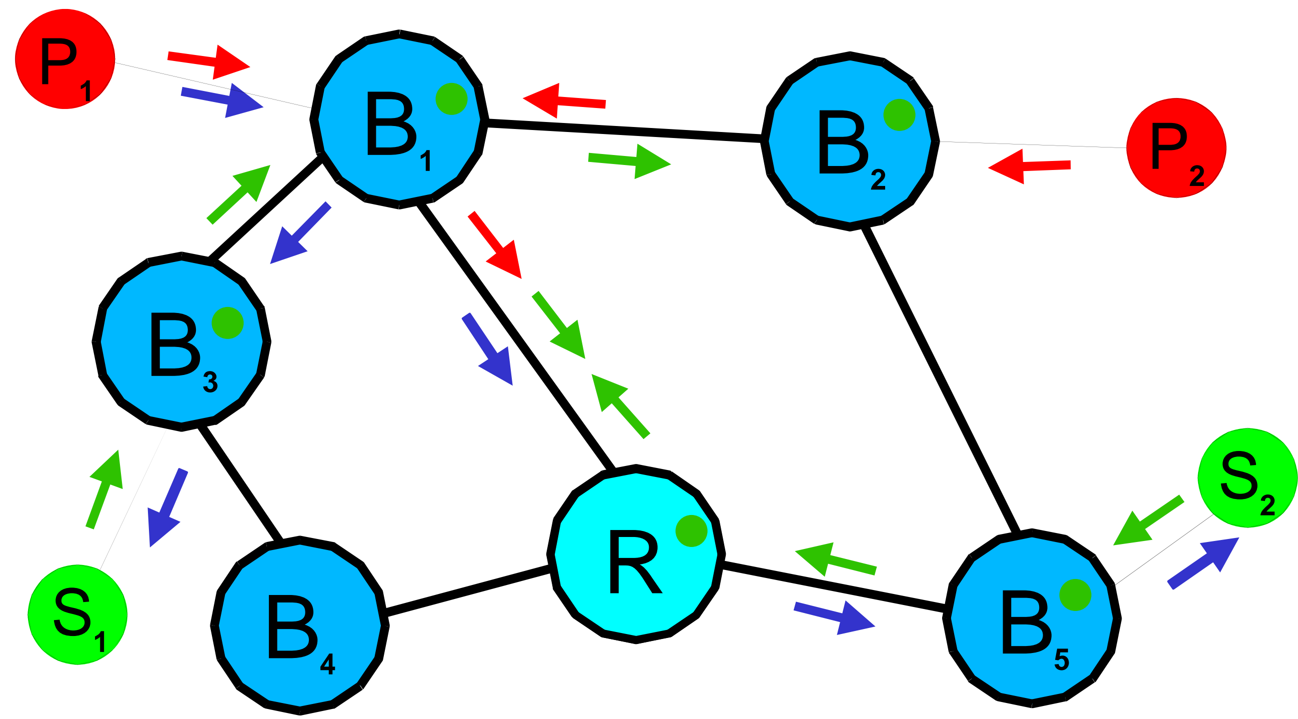
Algorithms I – Topic-Based Pub/Sub

- Type Msg, Advertisements, Subscriptions, Notifications
- **Rendezvous Nodes**
- **Reverse Path Forwarding**
  - Notifications follow Ads then the reverse path of Subs



Algorithms II – Content-Based Pub/Sub

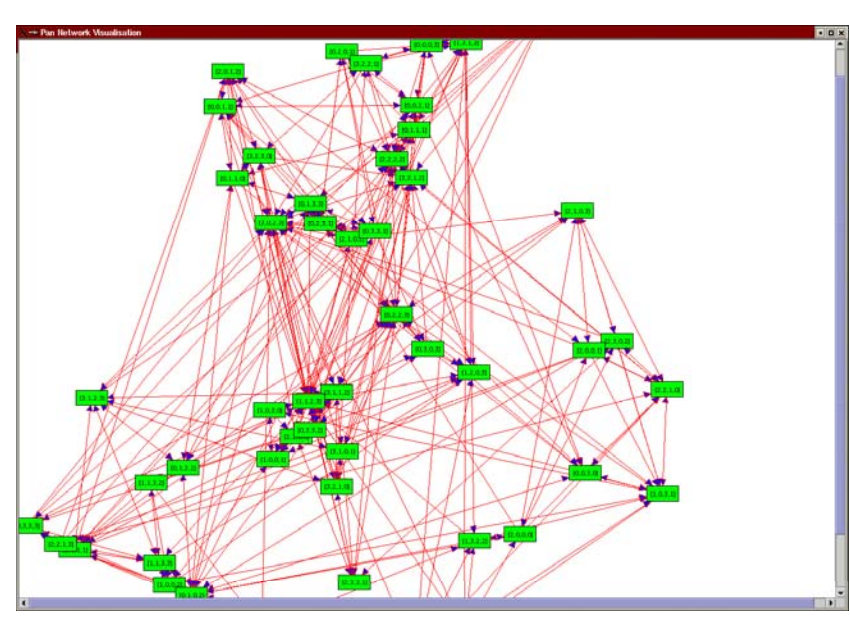
- Filtering State
- Notifications follow reverse paths of subscriptions
- Subscriptions consolidated by covering and merging for path sharing before late fanout



Hermes Implementation

- Actual implementation
  - Java implementation of event broker and event clients
  - Event types defined in XML Schema
  - Java language binding for events using reflection

- Implementation within a simulator
  - Large-scale, Internet-like topologies
  - over 100 nodes
  - used in later projects



Pub/sub not sufficient for general applications

- decouples publishers and subscribers
  - pubs and/or subs need not be running at the same time
- publishers are anonymous to subscribers
  - subs need to know topic (attributes), not pubs' names and locations
  - but receivers may **need** to know the sender or sender's role
- only multicast, one-to-many communication
  - may also need one-to-one and request-reply
- can't reply
  - either anonymously, e.g. to vote, or identified (can be fixed)
- efficient notification for large-scale systems using CBR
  - but content-based routing may violate privacy of information
  - subscriptions may also be confidential

Event-Driven systems – Composite Events

- Event composition (correlation)
  - Pietzuch, Shand, Bacon, Middleware 2003, and IEEE Network, Jan/Feb 2004

- composite event service above event brokers
- service instances placed to optimise communication
- FSM recognisers – parallel evaluation
- events have source-specific interval timestamps
- simulations of large-scale systems…

Bottom-up and/or Top-Down?

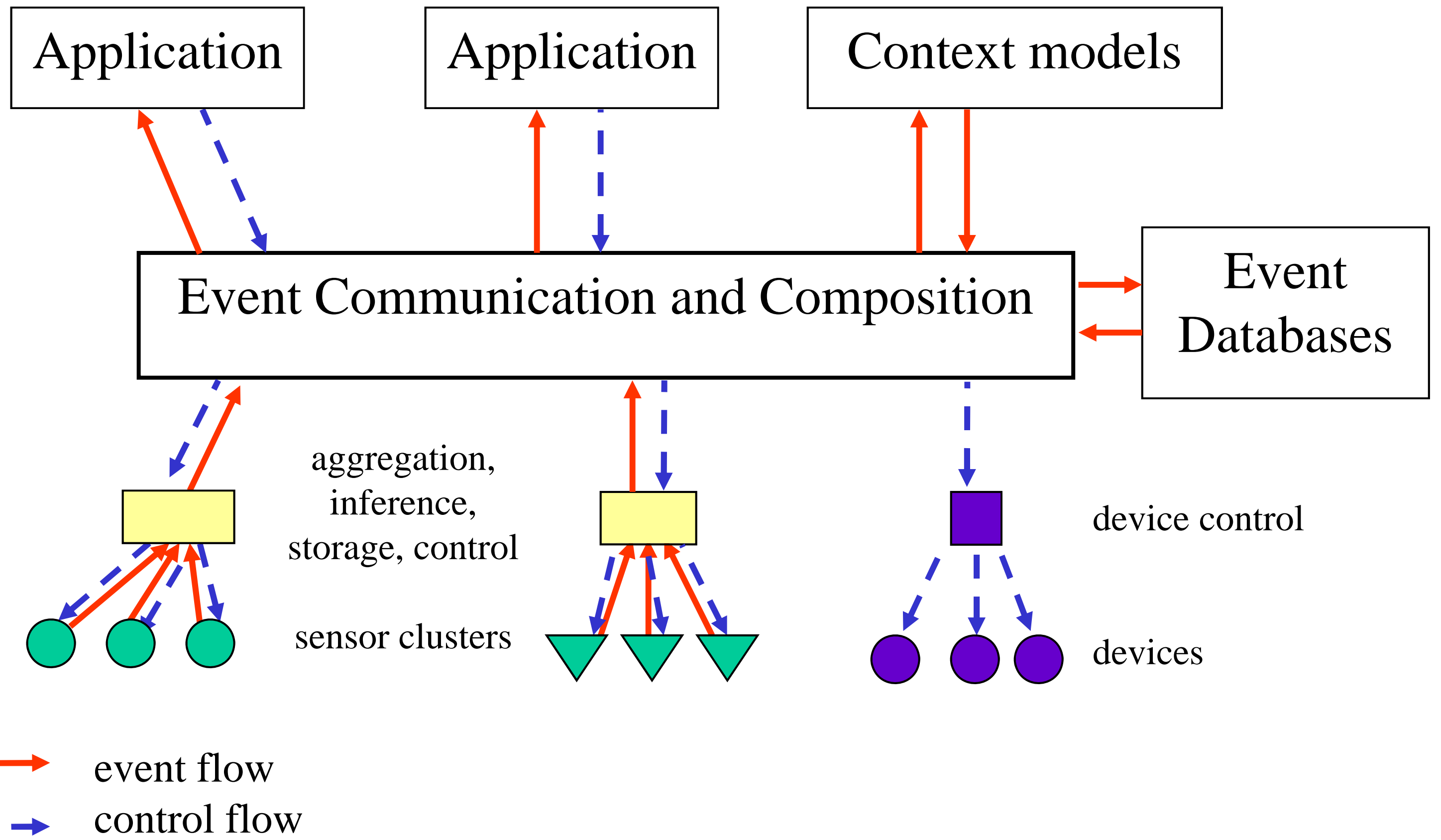
- Can we express all we require by bottom-up composition of primitive events?
- Can we take advantage of high-level models of context?
  - e.g. maps, plans, mathematical models, GIS
- What can users be expected to express?
- How is the *top-down, bottom-up gap* bridged and high-level requirements converted into event subscriptions?

"nearest empty meeting room?", "turn off the lights if the room is empty", "quickest way to get to Stansted airport?"

Work-in-Progress

Dagstuhl seminar in May, DEBS in Cambridge in July

Integrating sensor networks (1)



Integrating sensor networks (2)

- ***Data*:**
  - sensor-ID, data value, timestamp, location
  - value aggregation from densely deployed sensors
  - inaccuracies masked – data cleansing
  - heterogeneous sensor data correlated (fused)
- ***Information/semantics*:**
  - events defined, to present sensor data to applications including context models
  - events correlated, higher-level events generated
  - real-time delivery may be required
  - level of data logging required (keeping all sensor data)?

Traffic monitoring applications

- ***sensors:***
  - SCOOT loops for counting,
  - video cameras – extract and transmit anonymised data
  - thermal imaging (infra-red detectors),
  - acoustic detectors
  - bus location data
  - car-park occupancy detection
  - ANPR automatic number-plate recognition

- I subscribe to
  - *bus-seen-event (busID=*uni*4.*, location=MadingleyP&R)*

  and my desktop is pinged when the bus is detected.

Traffic monitoring applications (cont'd)

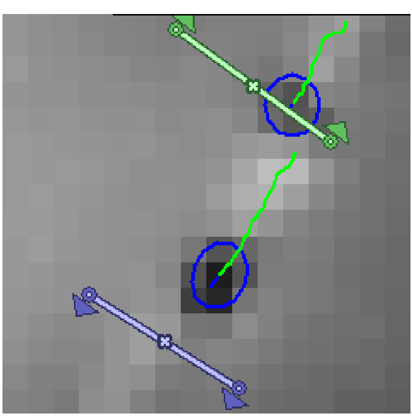
- Route-advice service: on entering my car I indicate my destination on a map – route is shown, car monitored and route updated dynamically as conditions change.

  Research on sharing of data within SatNav systems.
- Easy to do with bespoke systems and/or coupling applications with sensors in "vertical silos" e.g. car park data only sent to displays on radial roads into cities
- Sharing of (public) data: application developers (and public) can build services by subscribing to advertised events

Work-in-Progress: TIME-EACM project

Irisys infra-red cameras + motion detection

- combined with video for validation
  - (and banal tasks like positioning the lines)
- privacy-preserving
- Testing carried out: Dept. Eng. roof, Fen Causeway, 2006
  ~ 90% accuracy cf. video
- wired communication via Engineering Dept.



Irisys – mirroring annual manual count

- carried out on Cambridge radial roads annually
  - we did Huntingdon Rd, 9th Oct 2006, 8am – 7pm
  - using one of DTG's sentient vans
  - amateur positioning (by us)
  - incoming and outgoing traffic
  - validated against video
  - over 90% accuracy cf. video if cycles excluded
- County Council haven't told us how their manual count compares with video

Irisys – ongoing monitoring

- mounted on a lamp post on Madingley Road
- connection to CL via Wifi

Stagecoach/ACIS bus monitoring

- GPS location of buses on some Stagecoach routes
- radio transfers data back to base
  - some GPRS, some custom
- bus-stop displays
  - timetables and expected arrival times
  - Minibus project – mobile phone selection of bus stops and display of information
- live and historical data from ACIS since Aug 2007
  - for project use - under a NDA
- this data allows journey times and congestion to be analysed and predicted

Healthcare monitoring application

sensors: body sensors for blood-pressure, blood-sugar, etc.
cameras / thermal imaging in smart homes, tag objects

- Emergency detection based on sensor values and image analysis – how to decide when to summon help?
- Smart homes: monitoring for falls, visitors, …
  - (guide-dogs–vs–people?) (visitors–vs–burglars?)
- Tagging objects: “where did I leave …?”, (*pull* model)
  or to build a world model for navigation avoiding obstacles
- Economic model? cost of technology–vs–more people? risks/costs of false positives and false negatives?

- **Work-in-Progress: CareGrid and PAL projects**

Integrating databases with pub/sub

- DB world: continuous queries require recording of individual queries and individual response, one-to-one.
- instead, Event-Based world: databases advertise events
  - *event type* (*<attribute-type>*)

  e.g."cars-for-sale(maker, model, colour, automatic?, …)"

  advertised by many databases e.g. in the Cambridge area
- clients subscribe and are notified of occurrences
- the pub/sub service does the filtering – not the database
- we have used PostgreSQL – active predicate store

DB Motivating Example – Police IT

Bill Hayden is suspected of masterminding a nationwide terrorist organisation.

- As well as looking up his past database records, the investigators (special terrorism unit) **subscribe**, in all 43 police counties, to **advertised** database update events specifying his name as an attribute.
  - *Note inter-domain naming and access control.*
- Triggers are set in the databases so that any future records that are made, relating to his movements and activities, will be **published** and **notified** automatically and immediately to *those authorised* to investigate him.

Securing pub/sub using RBAC

- At the event client level – use RBAC
  - domain-level authorisation policy indicates, for event types and attributes, the roles that can advertise/publish and subscribe
  - inter-domain subscription is negotiated, as for any other service
    - note that spamming is prevented – only authenticated roles can use the pub/sub service to advertise/publish
- At the event-broker level – use encryption
  - are all the event brokers trusted?
  - if not, some may not be allowed to see (decrypt) some (attributes of) some messages.
    - this affects content-based routing

Work-in-Progress – SmartFlow project