Distributed Systems

Coordination and naming

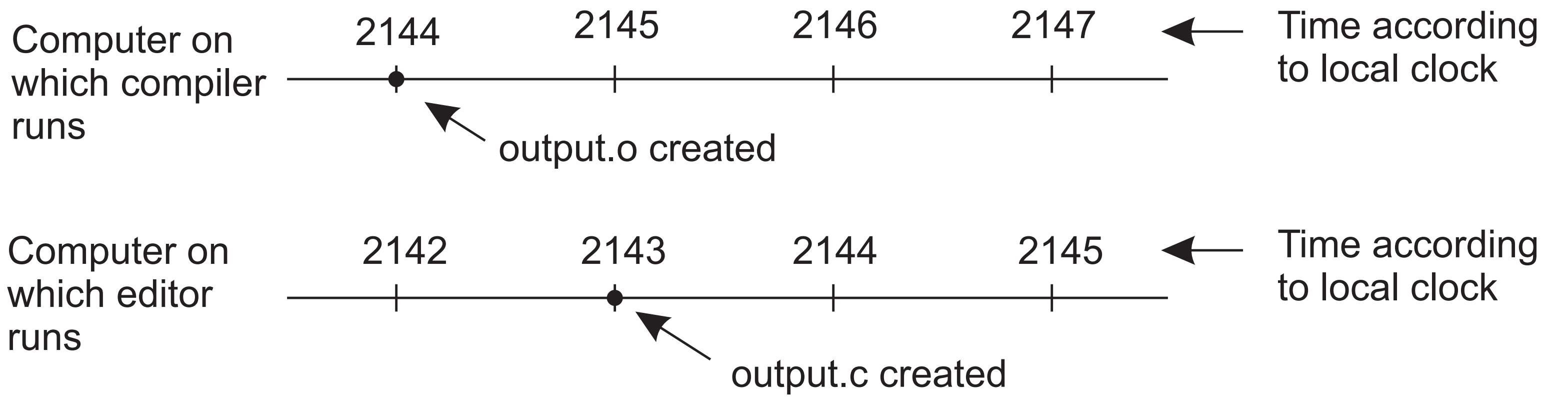
TDTS04 – Computer Networks and Distributed Systems
Carl Magnus Bruhner, ADIT/IDA

LINKÖPING UNIVERSITY

Based on slides by M. van Steen and A. S. Tanenbaum,
Distributed Systems, 4th ed., distributed-systems.net

Coordination in distributed systems

Challenges of coordination



Physical clocks

Problem

Sometimes we simply need the exact time, not just an ordering.

Solution: Universal Coordinated Time (UTC)

- Based on the number of transitions per second of the cesium 133 atom (pretty accurate).
- At present, the real time is taken as the average of some 50 cesium clocks around the world.
- Introduces a leap second from time to time to compensate that days are getting longer.

Note

UTC is broadcast through short-wave radio and satellite. Satellites can give an accuracy of about ±0.5 ms.

Clock synchronization

Precision

The goal is to keep the deviation between two clocks on any two machines within a specified bound, known as the precision π:

$$\forall t, \forall p, q : |C_p(t) - C_q(t)| \leq \pi$$

with $C_p(t)$ the computed clock time of machine p at UTC time t.

Accuracy

In the case of accuracy, we aim to keep the clock bound to a value α:

$$\forall t, \forall p : |C_p(t) - t| \leq \alpha$$

Synchronization

- Internal synchronization: keep clocks precise
- External synchronization: keep clocks accurate

Clock drift

Clock specifications

- A clock comes specified with its maximum clock drift rate ρ.
- $F(t)$ denotes oscillator frequency of the hardware clock at time t
- F is the clock's ideal (constant) frequency ⇒ living up to specifications:

$$\forall t : (1-\rho) \leq \frac{F(t)}{F} \leq (1+\rho)$$

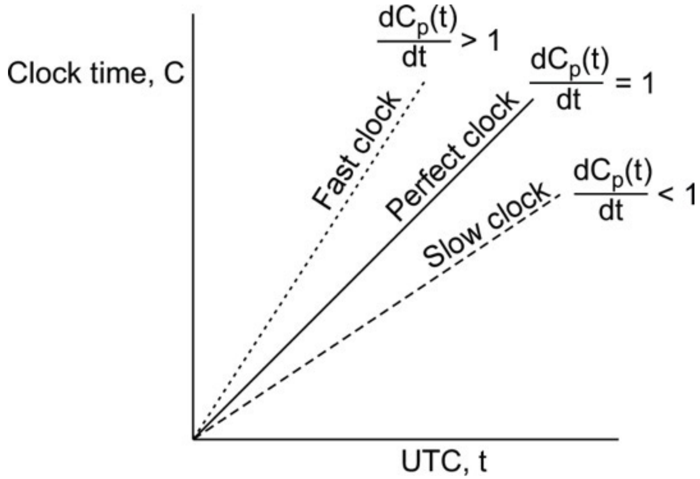
Observation

By using hardware interrupts we couple a software clock to the hardware clock, and thus also its clock drift rate:

$$C_p(t) = \frac{1}{F}\int_0^t F(t)dt \Rightarrow \frac{dC_p(t)}{dt} = \frac{F(t)}{F}$$

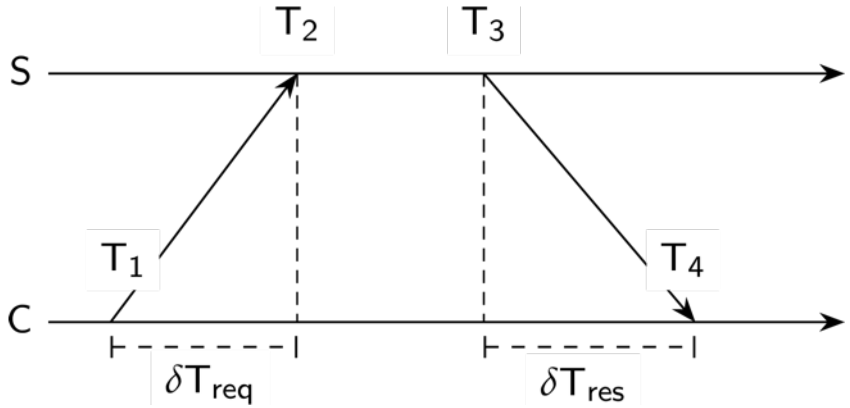
$$\Rightarrow \forall t : 1-\rho \leq \frac{dC_p(t)}{dt} \leq 1+\rho$$

Fast, perfect, slow clocks



Detecting and adjusting incorrect times

Getting the current time from a timeserver



Computing the relative offset θ and delay δ

Assumption: $\delta T_{req} = T_2 - T_1 \approx T_4 - T_3 = \delta T_{res}$

$$\theta = T_3 + ((T_2 - T_1) + (T_4 - T_3))/2 - T_4 = ((T_2 - T_1) + (T_3 - T_4))/2$$

$$\delta = ((T_4 - T_1) - (T_3 - T_2))/2$$

Network Time Protocol

Collect (θ, δ) pairs. Choose θ for which associated delay δ was minimal.

The Happened-before relationship

Issue

What usually matters is not that all processes agree on exactly what time it is, but that they agree on the order in which events occur. Requires a notion of ordering.

The happened-before relation

- If *a* and *b* are two events in the same process, and *a* comes before *b*, then $a \rightarrow b$.
- If *a* is the sending of a message, and *b* is the receipt of that message, then $a \rightarrow b$
- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

Note

This introduces a partial ordering of events in a system with concurrently operating processes.

Logical clocks & Lamport's clock

Problem

How do we maintain a global view of the system's behavior that is consistent with the happened-before relation?

Attach a timestamp $C(e)$ to each event e, satisfying the following properties:

- P1 If a and b are two events in the same process, and $a \rightarrow b$, then we demand that $C(a) < C(b)$.
- P2 If a corresponds to sending a message m, and b to the receipt of that message, then also $C(a) < C(b)$.

Problem

How to attach a timestamp to an event when there's no global clock ⇒ maintain a consistent set of logical clocks, one per process.

Logical clocks: solution

Each process P_i maintains a local counter C_i and adjusts this counter

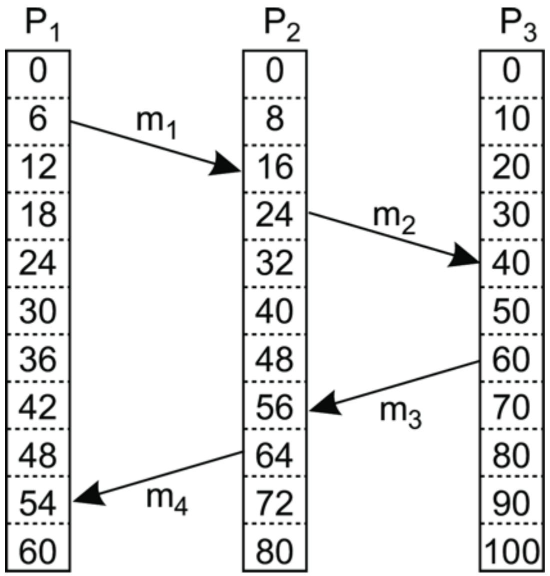
1. For each new event that takes place within P_i, C_i is incremented by 1.
2. Each time a message m is sent by process P_i, the message receives a timestamp $ts(m) = C_i$.
3. Whenever a message m is received by a process P_j, P_j adjusts its local counter C_j to $\max\{C_j, ts(m)\}$; then executes step 1 before passing m to the application.

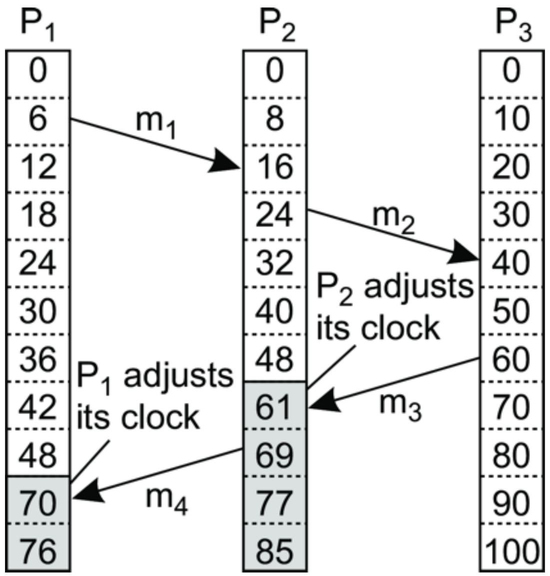
Notes

- Property P1 is satisfied by (1); Property P2 by (2) and (3).
- It can still occur that two events happen at the same time. Avoid this by breaking ties through process IDs.

Logical clocks: example (Lamport’s timestamps)

Consider three processes with event counters operating at different rates

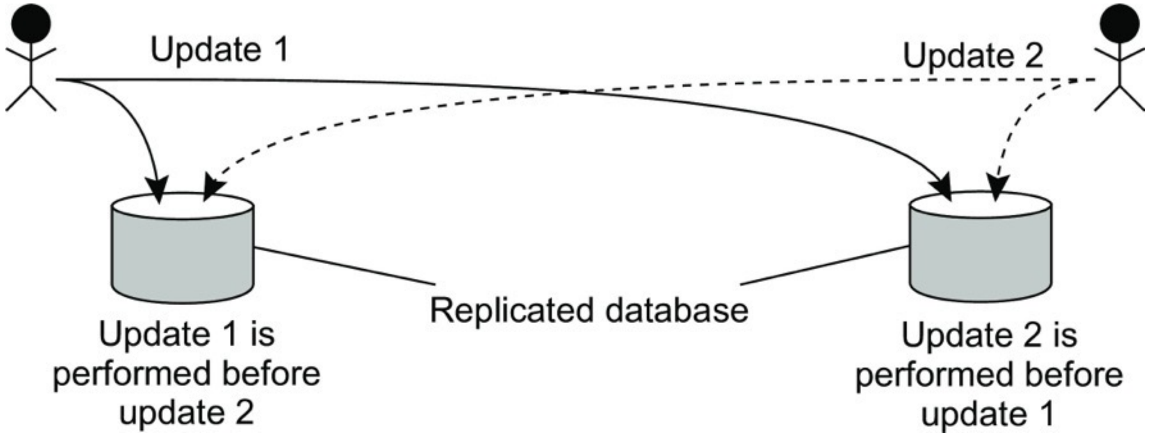




Example: Totally ordered multicast

Concurrent updates on a replicated database are seen in the same order everywhere

- P_1 adds \$100 to an account (initial value: \$1000)
- P_2 increments account by 1%
- There are two replicas



Result

In absence of proper synchronization:
replica #1 ← \$1111, while replica #2 ← \$1110.

Example: Totally ordered multicast

Solution

- Process P_i sends timestamped message m_i to all others. The message itself is put in a local queue $queue_i$.
- Any incoming message at P_j is queued in $queue_j$, according to its timestamp, and acknowledged to every other process.

P_j passes a message m_i to its application if:

(1) m_i is at the head of $queue_j$

(2) for each process P_k, there is a message m_k in $queue_j$ with a larger timestamp.

Note

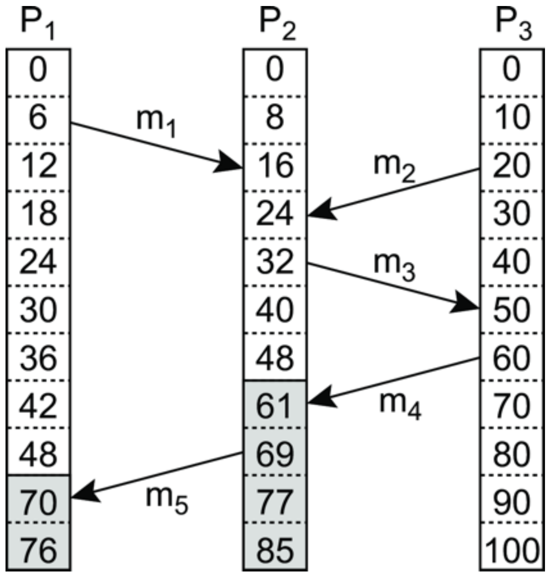
We are assuming that communication is reliable and FIFO ordered.

Vector clocks

Observation

Lamport's clocks do not guarantee that if $C(a) < C(b)$ that a causally preceded b.

Concurrent message transmission using logical clocks



Observation

Event a: m_1 is received at $T = 16$;

Event b: m_2 is sent at $T = 20$.

Note

We cannot conclude that a causally precedes b.

Capturing potential causality

Solution: each P_i maintains a vector VC_i

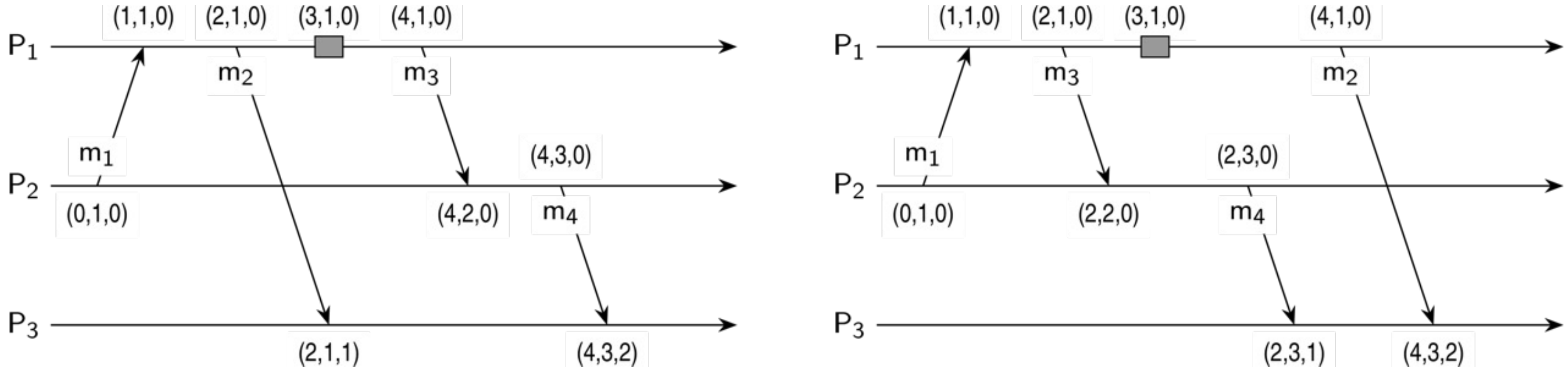
- $VC_i[i]$ is the local logical clock at process P_i.
- If $VC_i[j] = k$ then P_i knows that k events have occurred at P_j.

Maintaining vector clocks

1. Before executing an event, P_i executes $VC_i[i] \leftarrow VC_i[i] + 1$.
2. When process P_i sends a message m to P_j, it sets m's (vector) timestamp $ts(m)$ equal to VC_i after having executed step 1.
3. Upon the receipt of a message m, process P_j sets $VC_j[k] \leftarrow \max\{VC_j[k], ts(m)[k]\}$ for each k, after which it executes step 1 and then delivers the message to the application.

Vector clocks: Example

Capturing potential causality when exchanging messages



(a) (b)

Analysis

Situation	$ts(m_2)$	$ts(m_4)$	$ts(m_2) < ts(m_4)$	$ts(m_2) > ts(m_4)$	**Conclusion**
(a)	(2, 1, 0)	(4, 3, 0)	Yes	No	m_2 may causally precede m_4
(b)	(4, 1, 0)	(2, 3, 0)	No	No	m_2 and m_4 may conflict

Causally ordered multicasting

Observation

We can now ensure that a message is delivered only if all causally preceding messages have already been delivered.

Adjustment

P_i increments $VC_i[i]$ only when sending a message, and P_j “adjusts” VC_j when receiving a message (i.e., effectively does not change $VC_j[j]$).

P_j postpones delivery of m until:

1. $ts(m)[i] = VC_j[i] + 1$
2. $ts(m)[k] \leq VC_j[k]$ for all $k \neq i$

Mutual exclusion

Problem

Several processes in a distributed system want exclusive access to some resource.

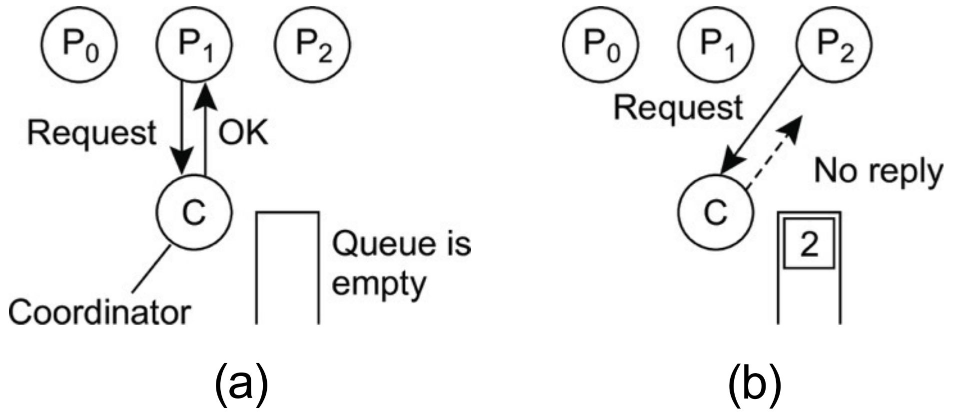
Basic solutions

Permission-based: A process wanting to enter its critical region, or access a resource, needs permission from other processes.

Token-based: A token is passed between processes. The one who has the token may proceed in its critical region, or pass it on when not interested.

Permission-based, centralized

Simply use a coordinator



(a) Process P_1 asks the coordinator for permission to access a shared resource. Permission is granted.

(b) Process P_2 then asks permission to access the same resource. The coordinator does not reply.

(c) When P_1 releases the resource, it tells the coordinator, which then replies to P_2.

Election algorithms

Principle

An algorithm requires that some process acts as a coordinator. The question is how to select this special process dynamically.

Note

In many systems, the coordinator is chosen manually (e.g., file servers). This leads to centralized solutions ⇒ single point of failure.

Basic assumptions

- All processes have unique id’s
- All processes know id’s of all processes in the system (but not if they are up or down)
- Election means identifying the process with the highest id that is up

Election by bullying

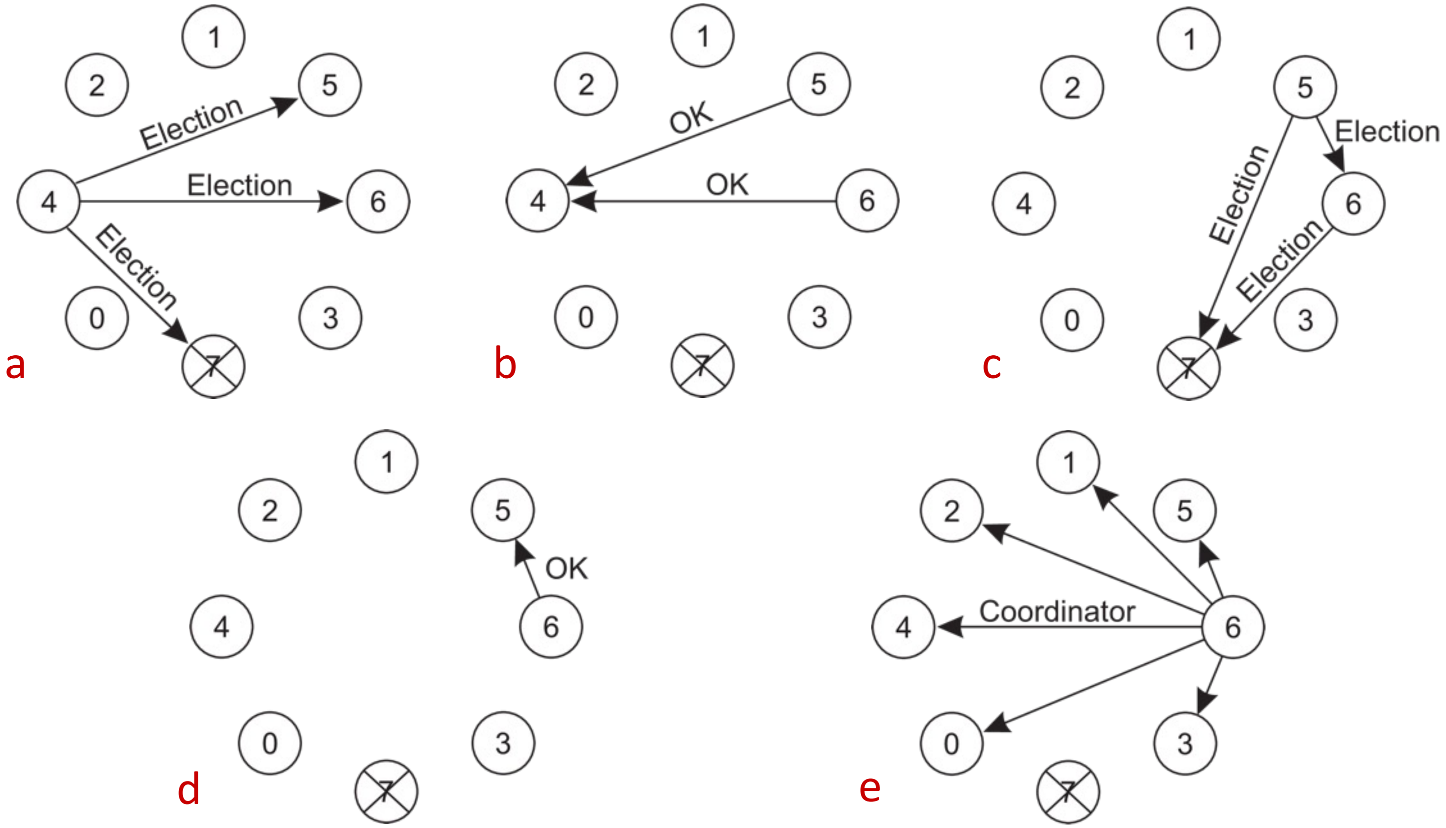
Principle

Consider N processes $\{P_0, \ldots, P_{N-1}\}$ and let $id(P_k) = k$. When a process P_k notices that the coordinator is no longer responding to requests, it initiates an election:

1. P_k sends an *ELECTION* message to all processes with higher identifiers: $P_{k+1}, P_{k+2}, \ldots, P_{N-1}$.
2. If no one responds, P_k wins the election and becomes coordinator.
3. If one of the higher-ups answers, it takes over and P_k's job is done.

Election by bullying

The bully election algorithm



Election in a ring

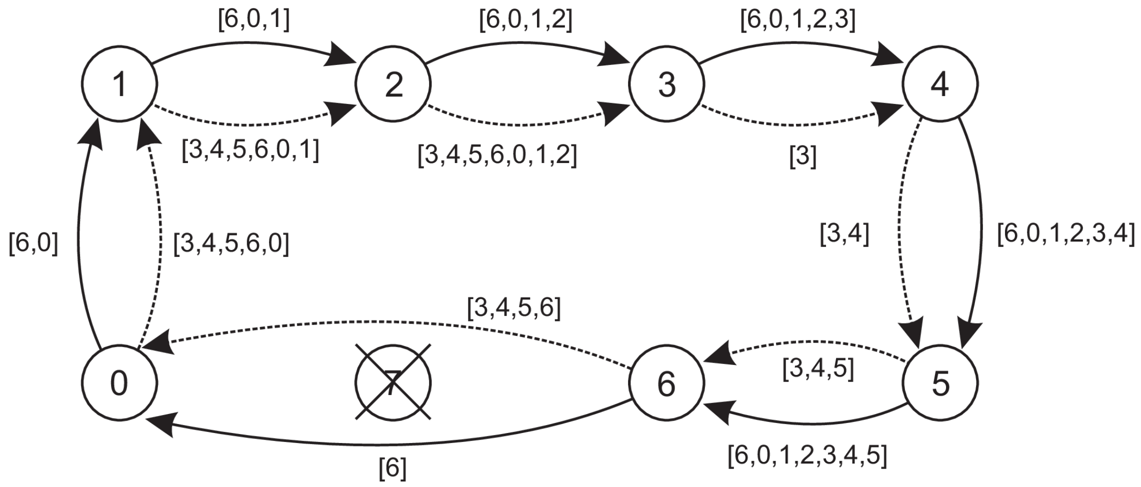
Principle

Process priority is obtained by organizing processes into a (logical) ring. The process with the highest priority should be elected as coordinator.

- Any process can start an election by sending an election message to its successor. If a successor is down, the message is passed on to the next successor.
- If a message is passed on, the sender adds itself to the list. When it gets back to the initiator, everyone had a chance to make its presence known.
- The initiator sends a coordinator message around the ring containing a list of all living processes. The one with the highest priority is elected as coordinator.

Election in a ring

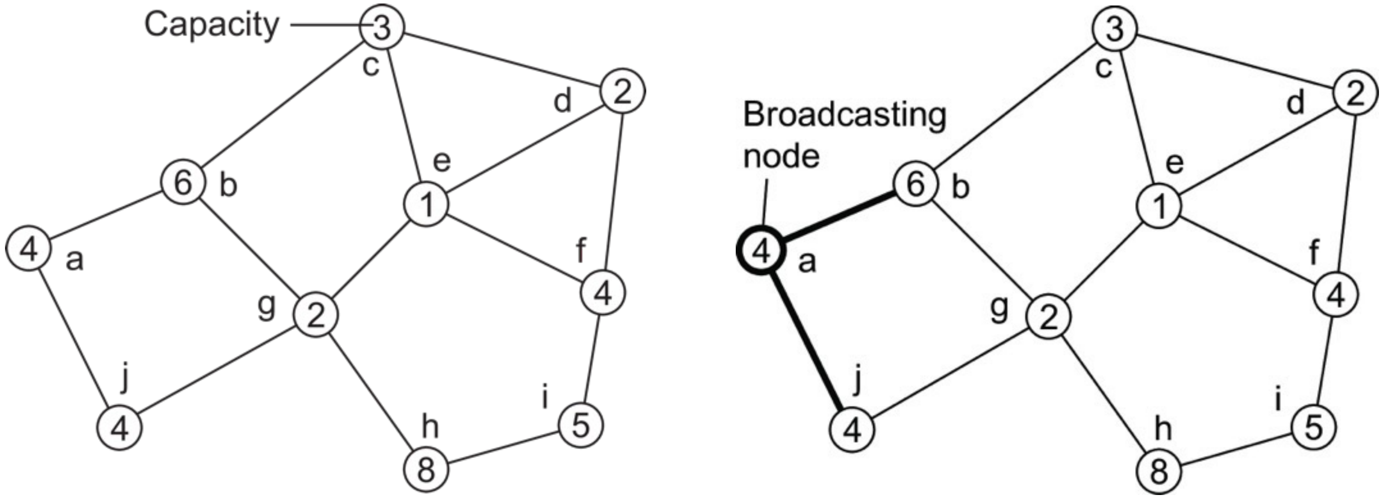
Election algorithm using a ring



- The solid line shows the election messages initiated by P_6
- The dashed one, the messages by P_3

A solution for wireless networks

A sample network

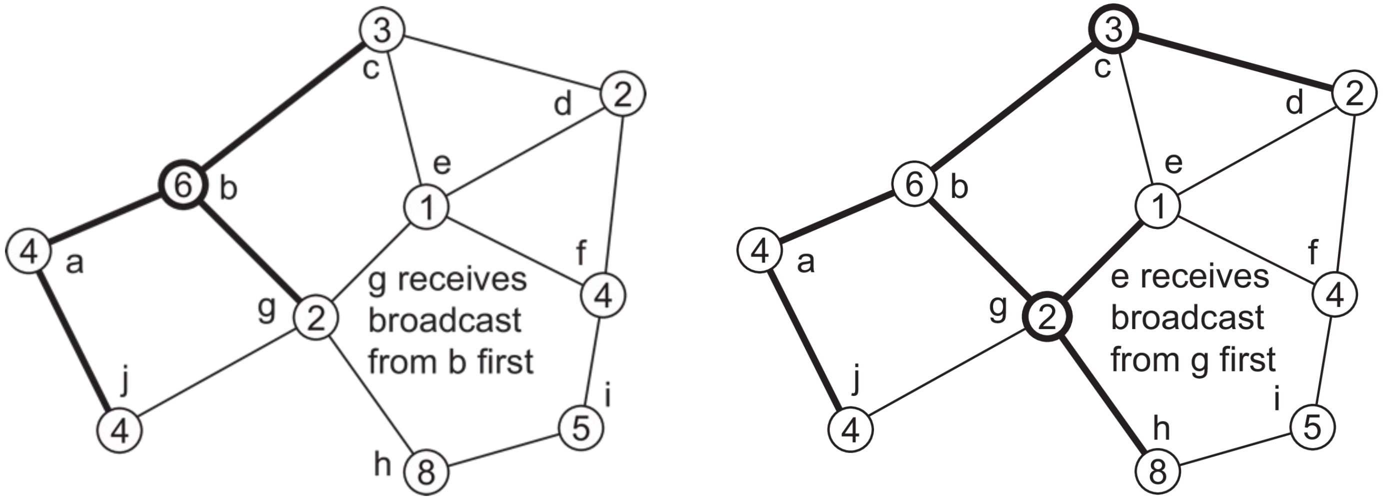


Essence

Find the node with the highest capacity to select as the next leader.

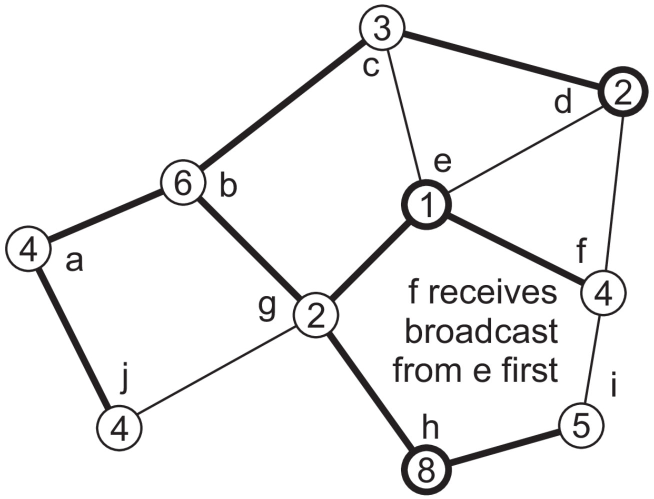
A solution for wireless networks

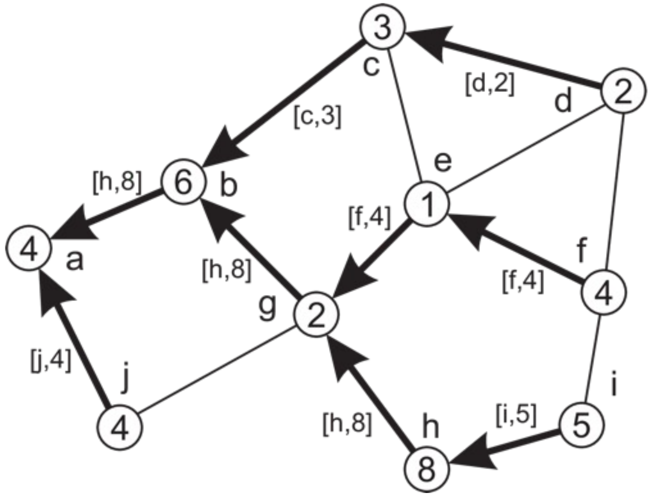
A sample network



A solution for wireless networks

A sample network





Essence

A node reports back only the node that it found to have the highest capacity.

Naming in distributed systems

Naming

Essence

Names are used to denote entities in a distributed system. To operate on an entity, we need to access it at an access point. Access points are entities that are named by means of an address.

Note

A location-independent name for an entity E, is independent of the addresses of the access points offered by E.

Identifiers

Pure name

A name that has no meaning at all; it is just a random string. Pure names can be used for comparison only.

Identifier: A name having some specific properties

1. An identifier refers to at most one entity.
2. Each entity is referred to by at most one identifier.
3. An identifier always refers to the same entity (i.e., it is never reused).

Broadcasting

Broadcast the ID, requesting the entity to return its current address

- Can never scale beyond local-area networks
- Requires all processes to listen to incoming location requests

Address Resolution Protocol (ARP)

To find out which MAC address is associated with an IP address, broadcast the query “who has this IP address”?

Forwarding pointers

When an entity moves, it leaves behind a pointer to its next location

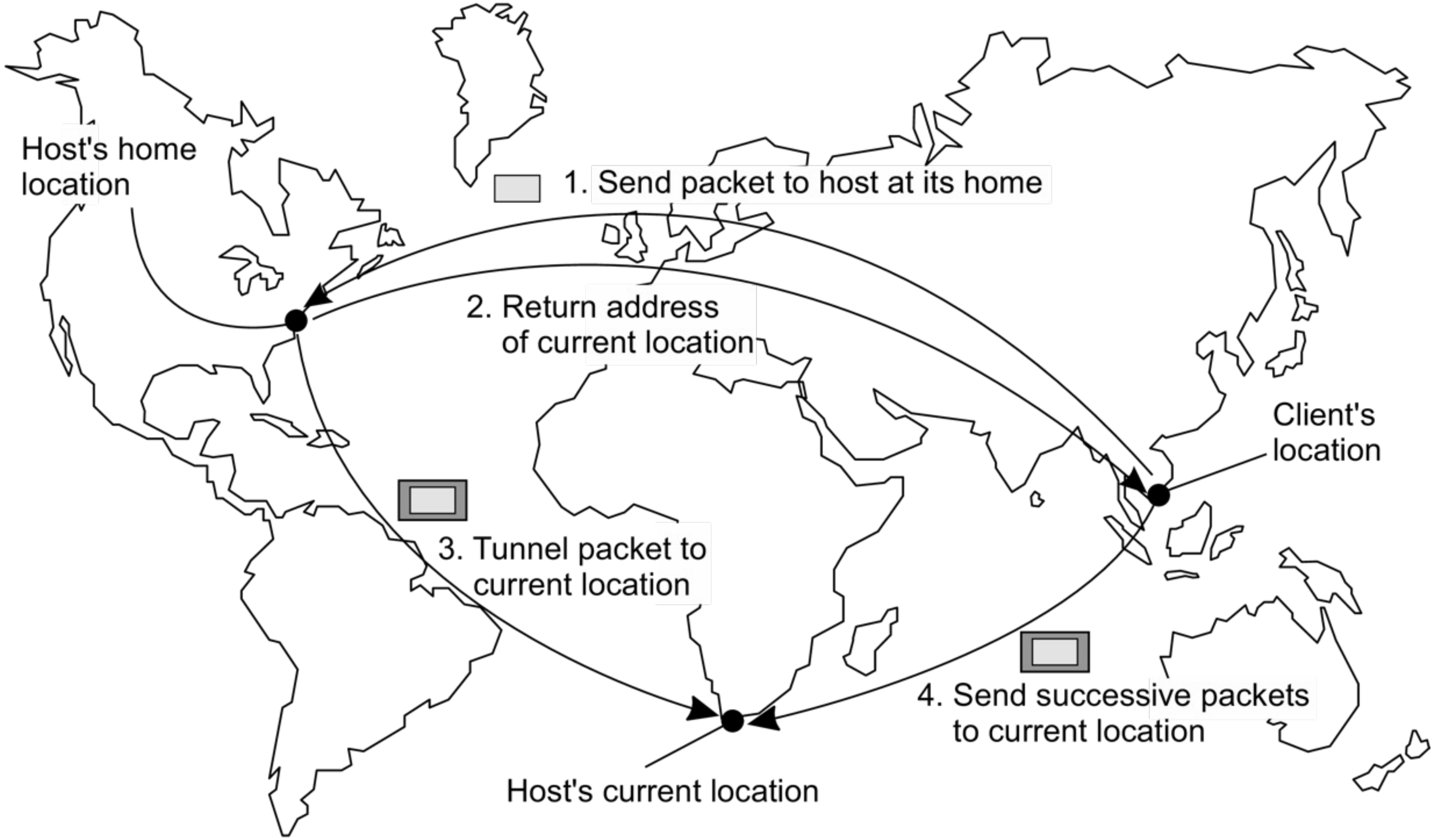
- Dereferencing can be made entirely transparent to clients by simply following the chain of pointers
- Update a client’s reference when present location is found
- Geographical scalability problems (for which separate chain reduction mechanisms are needed):
    - Long chains are not fault tolerant
    - Increased network latency at dereferencing

Home-based approaches

Single-tiered scheme: Let a home keep track of where the entity is

- Entity’s home address registered at a naming service
- The home registers the foreign address of the entity
- Client contacts the home first, and then continues with foreign location

The principle of mobile IP



Home-based approaches

Problems with home-based approaches

- Home address has to be supported for entity’s lifetime
- Home address is fixed ⇒ unnecessary burden when the entity permanently moves
- Poor geographical scalability (entity may be next to client)

Note

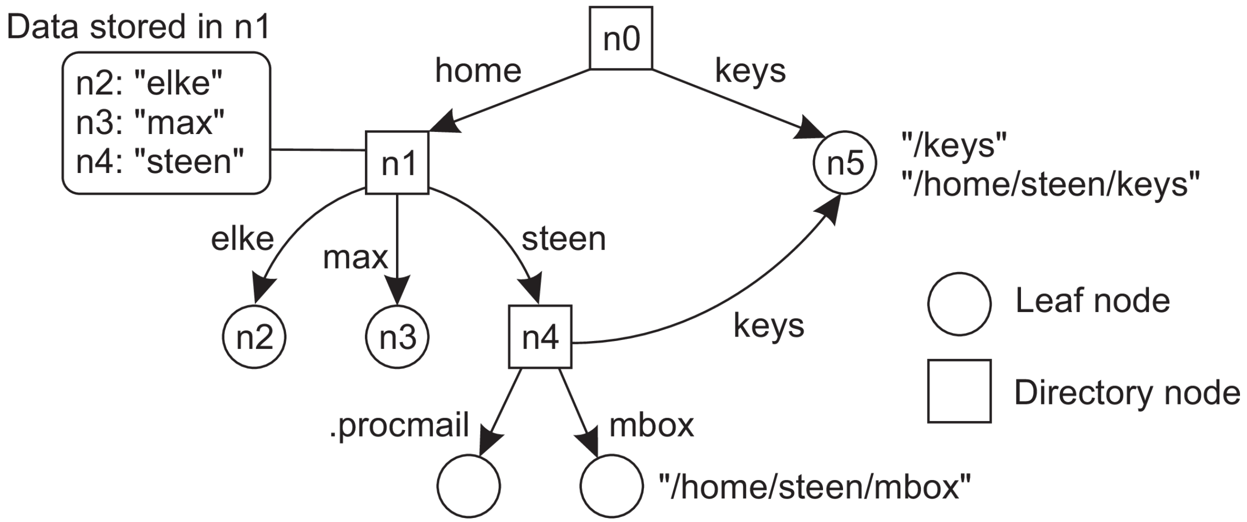
Permanent moves may be tackled with another level of naming (DNS)

Name space

Naming graph

A graph in which a leaf node represents a (named) entity. A directory node is an entity that refers to other nodes.

A general naming graph with a single root node



Note

A directory node contains a table of *(node identifier, edge label)* pairs.

Name space

We can easily store all kinds of attributes in a node

- Type of the entity
- An identifier for that entity
- Address of the entity’s location
- Nicknames
- ...

Note

Directory nodes can also have attributes, besides just storing a directory table with *(identifier, label)* pairs.

Name resolution

Problem

To resolve a name, we need a directory node. How do we actually find that (initial) node?

Closure mechanism: The mechanism to select the implicit context from which to start name resolution

- www.distributed-systems.net: start at a DNS name server
- /home/maarten/mbox: start at the local NFS file server (possible recursive search)
- 0031 20 598 7784: dial a phone number
- 77.167.55.6: route message to a specific IP address

Name linking

Hard link

What we have described so far as a path name: a name that is resolved by following a specific path in a naming graph from one node to another.

Soft link: Allow a node *N* to contain a name of another node

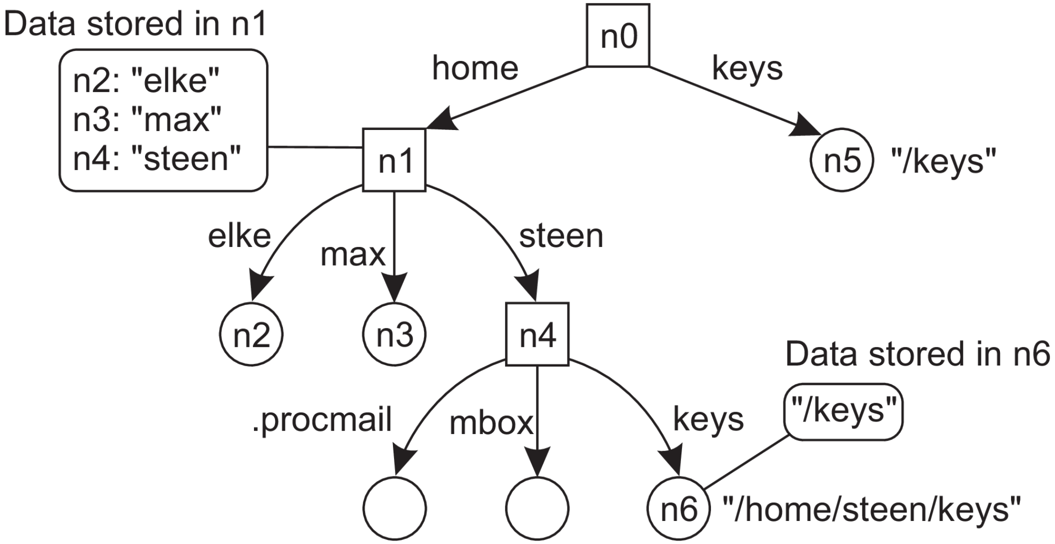
- First resolve *N*'s name (leading to *N*)
- Read the content of *N*, yielding *name*
- Name resolution continues with *name*

Observations

- The name resolution process determines that we read the content of a node, in particular, the name in the other node that we need to go to.
- One way or the other, we know where and how to start name resolution given *name*

Name linking

The concept of a symbolic link explained in a naming graph



Observation

Node *n5* has only one name

Mounting

Issue

Name resolution can also be used to merge different name spaces transparently through mounting: associating a node identifier of another name space with a node in a current name space.

Terminology

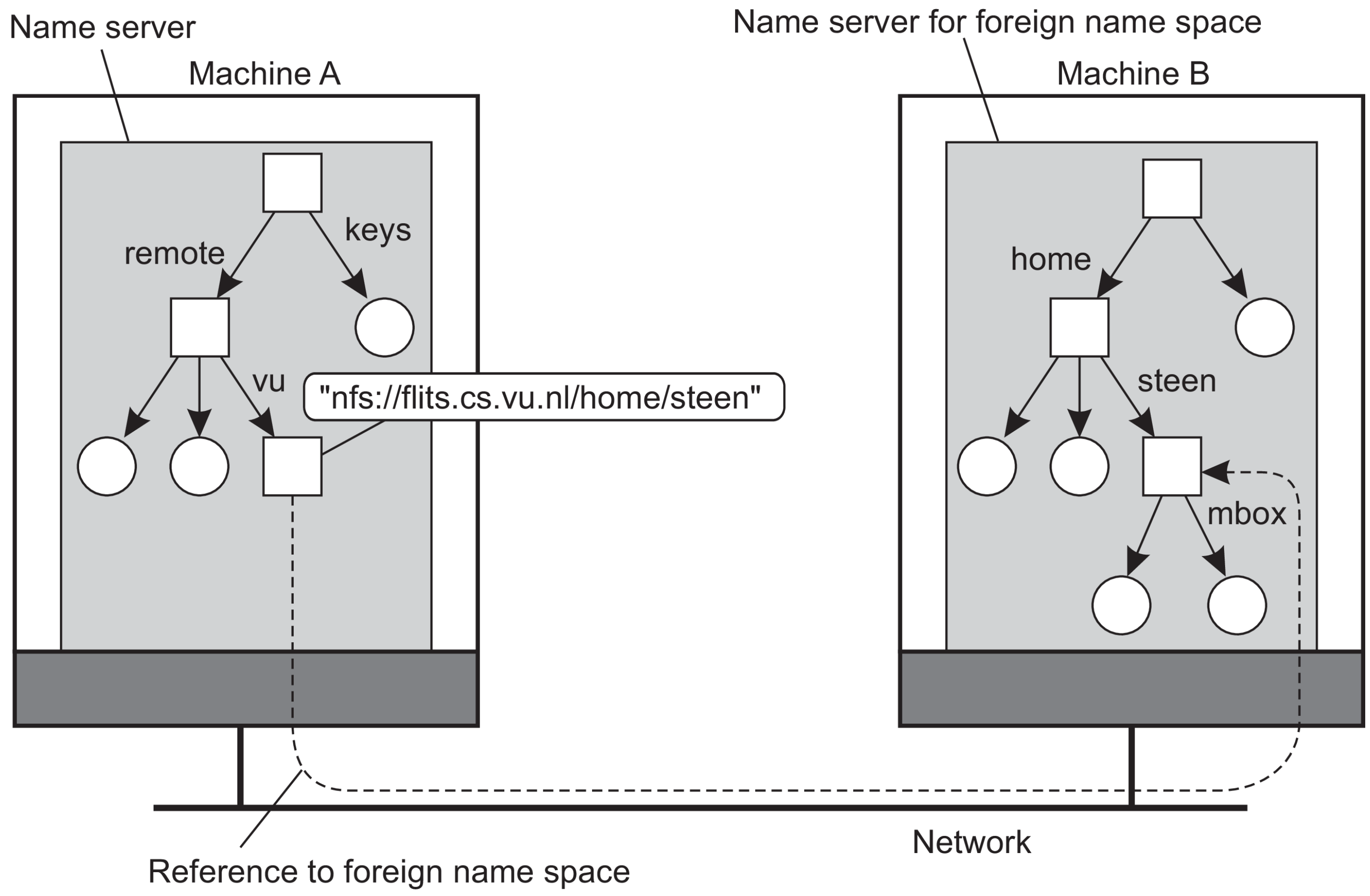
- Foreign name space: the name space that needs to be accessed
- Mount point: the node in the current name space containing the node identifier of the foreign name space
- Mounting point: the node in the foreign name space where to continue name resolution

Mounting across a network

1. The name of an access protocol.
2. The name of the server.
3. The name of the mounting point in the foreign name space.

Mounting in distributed systems

Mounting remote name spaces through a specific access protocol



Name-space implementation

Basic issue

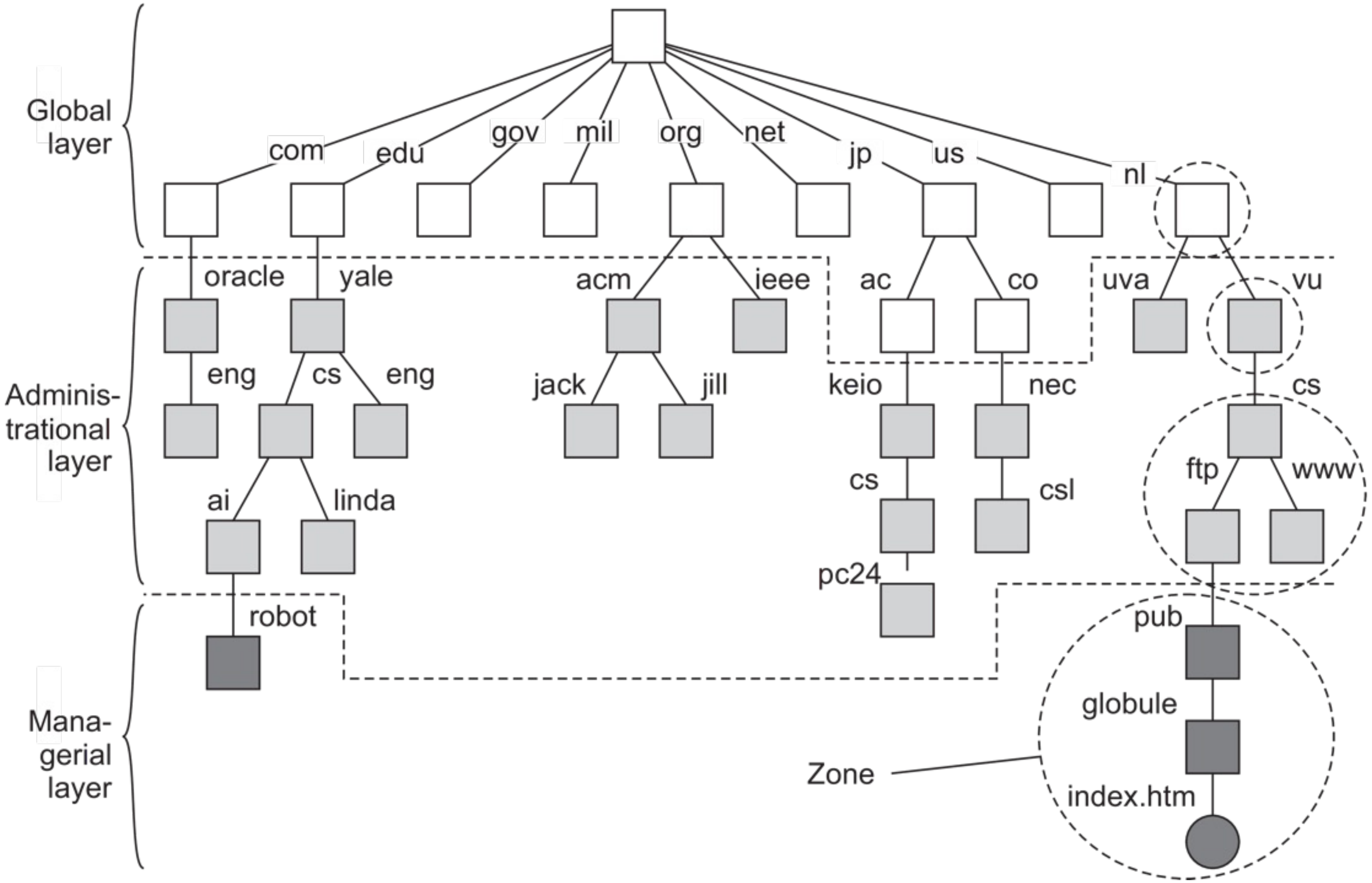
Distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.

Distinguish three levels

- Global level: Consists of the high-level directory nodes. Main aspect is that these directory nodes have to be jointly managed by different administrations
- Administrational level: Contains mid-level directory nodes that can be grouped in such a way that each group can be assigned to a separate administration.
- Managerial level: Consists of low-level directory nodes within a single administration. Main issue is effectively mapping directory nodes to local name servers.

Name-space implementation

An example partitioning of the DNS name space, including network files



Name-space implementation

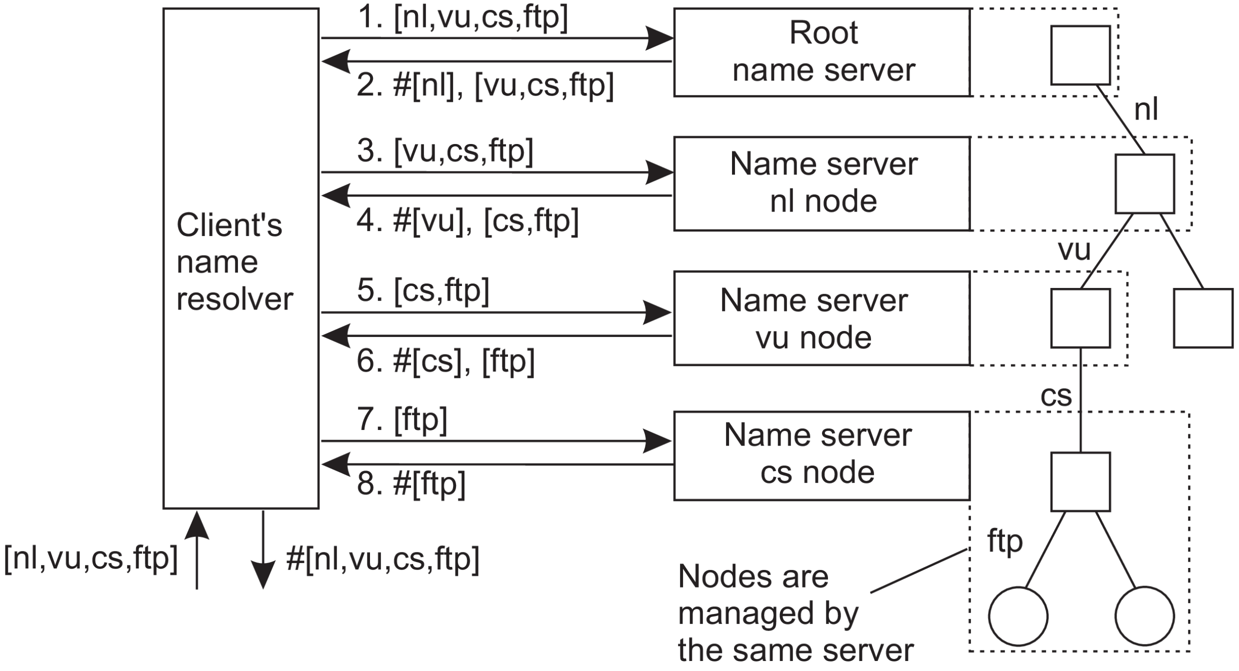
A comparison between name servers for implementing nodes in a name space

Item	Global	Administrational	Managerial
Geographical scale	Worldwide	Organization	Department
# Nodes	Few	Many	Vast numbers
Responsiveness	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
# Replicas	Many	None or few	None
Client-side caching?	Yes	Yes	Sometimes

Iterative name resolution

Principle

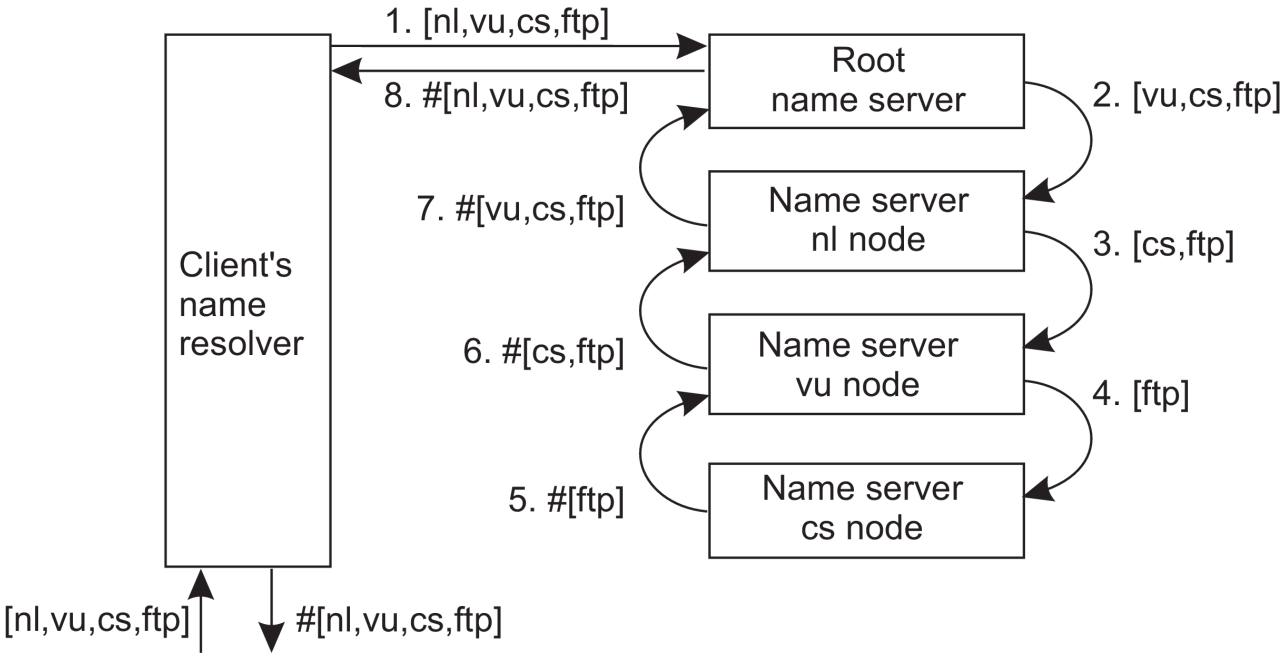
1. *resolve*(*dir*, [$name_1, \ldots, name_K$]) sent to $Server_0$ responsible for *dir*
2. $Server_0$ resolves *resolve*(*dir*, $name_1$) → dir_1, returning the identification (address) of $Server_1$, which stores dir_1.
3. Client sends *resolve*(dir_1, [$name_2, \ldots, name_K$]) to $Server_1$, etc.



Recursive name resolution

Principle

1. *resolve*(*dir*, [$name_1$,..., $name_K$]) sent to $Server_0$ responsible for *dir*
2. $Server_0$ resolves *resolve*(*dir*, $name_1$) → dir_1, and sends *resolve*(dir_1, [$name_2$, ..., $name_K$]) to $Server_1$, which stores dir_1.
3. $Server_0$ waits for result from $Server_1$, and returns it to client.



➥ **Next lecture:** Consistency, replication, and fault tolerance

Questions?

Questions/feedback: carl.magnus.bruhner@liu.se

LINKÖPING
UNIVERSITY