Distributed Systems

Naming (2)

Lecture 10

June 6 2005

Gerd Liefländer

System Architecture Group

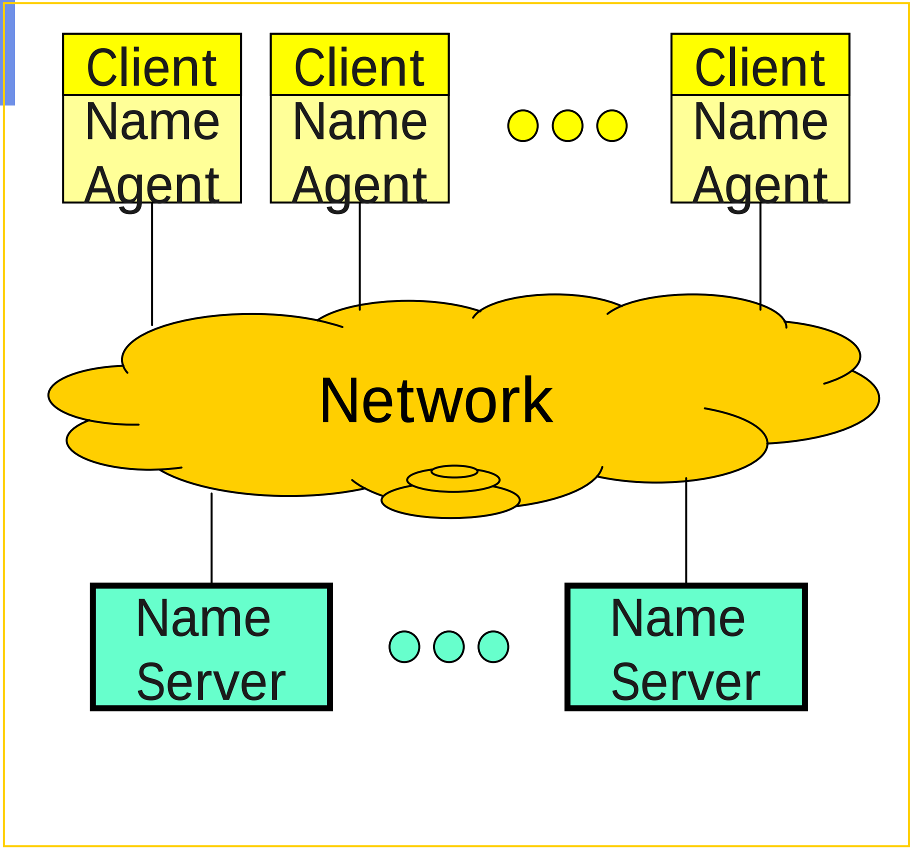
© 2004 Universität Karlsruhe, System Architecture Group

Schedule of Today

- Implementation of a Name Space
- Examples
  - DNS
  - X.500
  - ...
- Locating Mobile Entities
  - Naming versus Locating Entities
  - Simple Resolution
  - Home-based Hierarchical Approaches
- Removing Unreferenced Entities
  - Unreferenced Objects
  - Reference Counting and Listing
  - Identifying Unreachable Entities

© 2004 Universität Karlsruhe, System Architecture Group

Components of Name Service Architecture



Name agent:

- Supplies interface between name service and client
- Cooperates with name servers in order to *generate, resolve, …* names
- Caches the results

Name server:

- Manages context information and implements mapping function
- Sometimes cooperate with other name servers
- Supplies an interface for the name agent

Name Service Operations

Manipulation of context information:

- ADD: Create a reference: Name $\rightarrow$ Object
- DELETE: Delete a reference
- MODIFY: Change a reference

Queries:

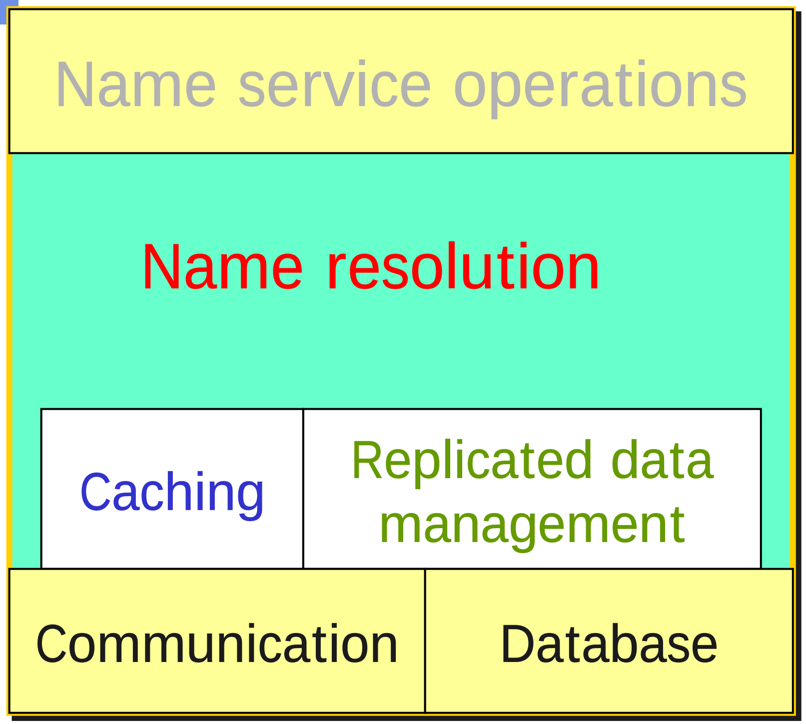
- READ: Resolve a name into an object
- SEARCH: Find a name or object based on some attributes
- LIST: List all names

Administration:

- Assignment of access privileges
- Authentication
- Extension of the name space

© 2004 Universität Karlsruhe, System Architecture Group

Name Server Architecture



Name Resolution
- mapping of names to objects

Caching
- of remote context information for efficient access

Replicated data management
- management of replicated context information

Communication
- between agents and name servers
- between name servers for cooperation purposes

Database
- management of local context information

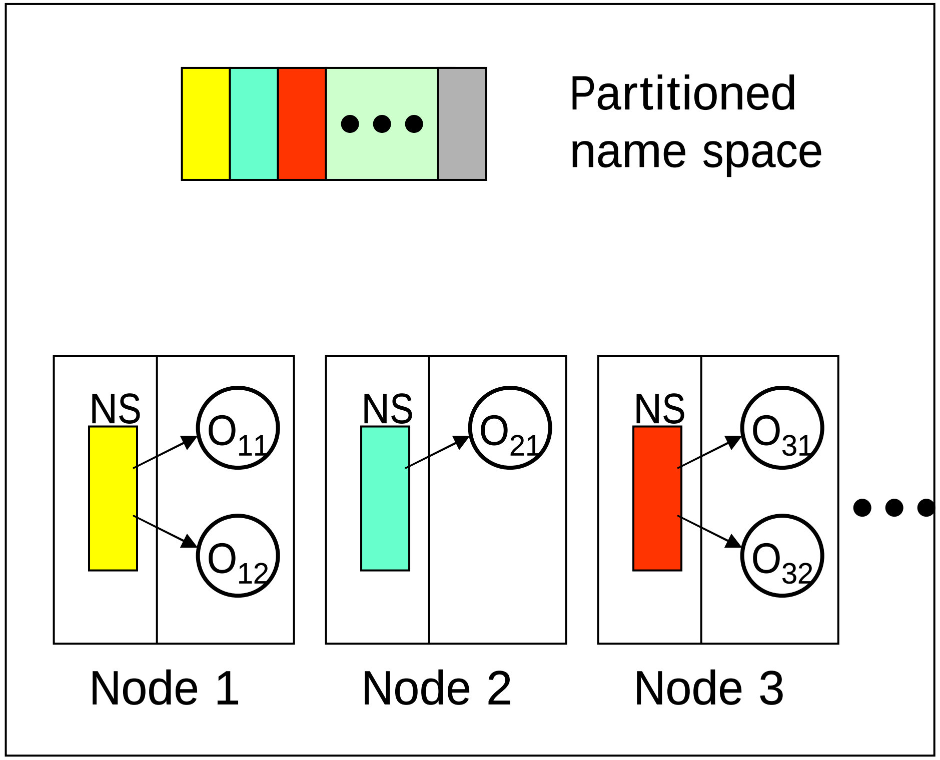
© 2004 Universität Karlsruhe, System Architecture Group

Resolution of Structured Names

- Resolution mechanism depends on type of name
- Location-dependent names
  - Name contains the location of the object it identifies
- Authority-dependent names
  - Name contains location of name server being responsible for name-resolution (the *authority*)
- Location-independent names
  - Name contains no location information

© 2004 Universität Karlsruhe, System Architecture Group

Location-Dependent Names (1)



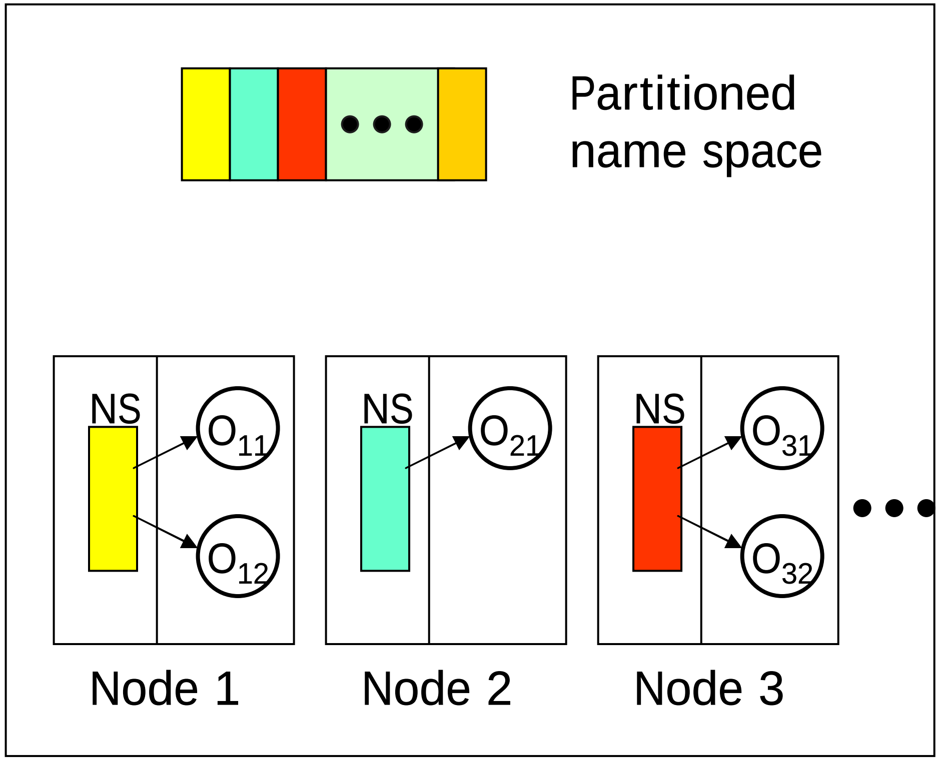
Structure of name space:

- Each node has its own name server
- Name server only manages the names of local objects
- Name structure:
  `NodeID.ObjectID`

Names are generated:

- by the local name server
- ... through concatenation of NodeID and local ObjectID

Location-Dependent Names (2)



Name resolution (of **N.O**):

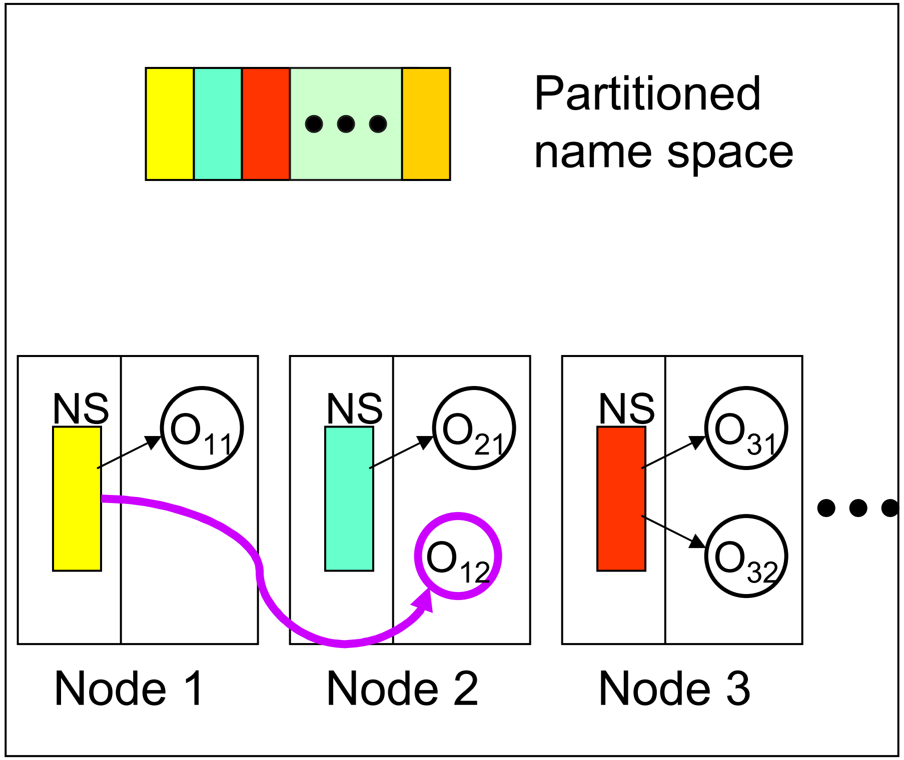
- Name agent sends query to node N.
- Name server of N maps O to a local object

Properties:

+ Nodes are highly autonomous
+ Name authority can be found in a simple and efficient way

- Objects can not migrate

Authority-Dependent Names (1)



Structure of the name space:

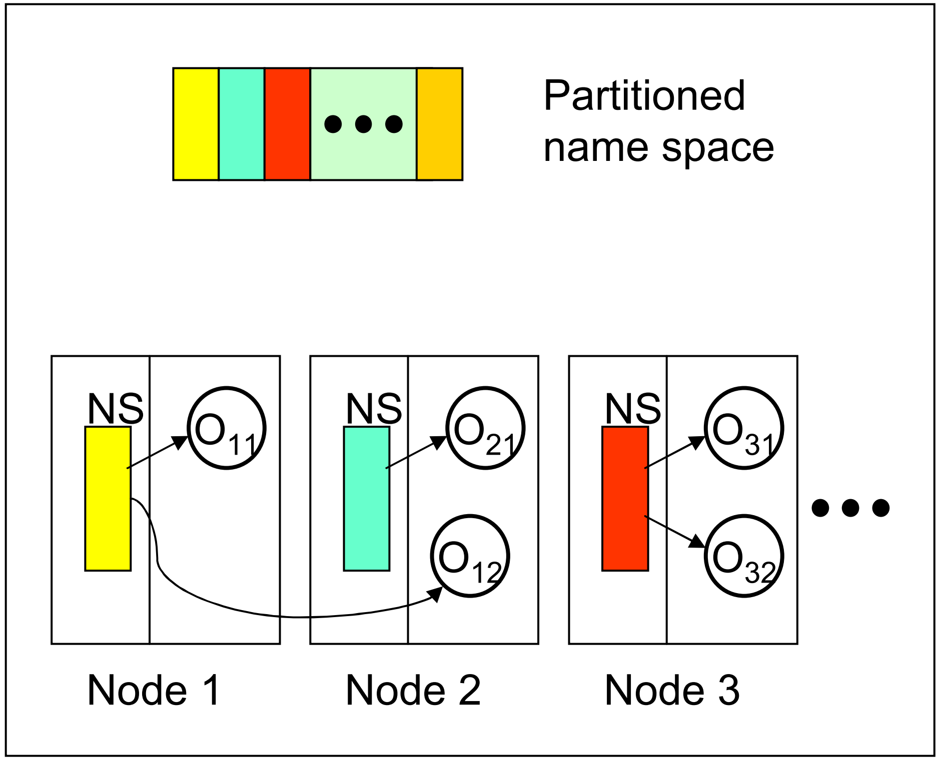
- Every node has a name server
- Name server manages the names of all locally-created objects
- Objects may migrate
- Name server must be aware of the locations of all migrated objets
- Name structure: NodeID.ObjectID

Names are generated:

- ... by local name server
- ... through concatenation of the NodeID and local unique ObjectID

© 2004 Universität Karlsruhe, System Architecture Group

Authority-Dependent Names (2)



Name resolution (of N.O):

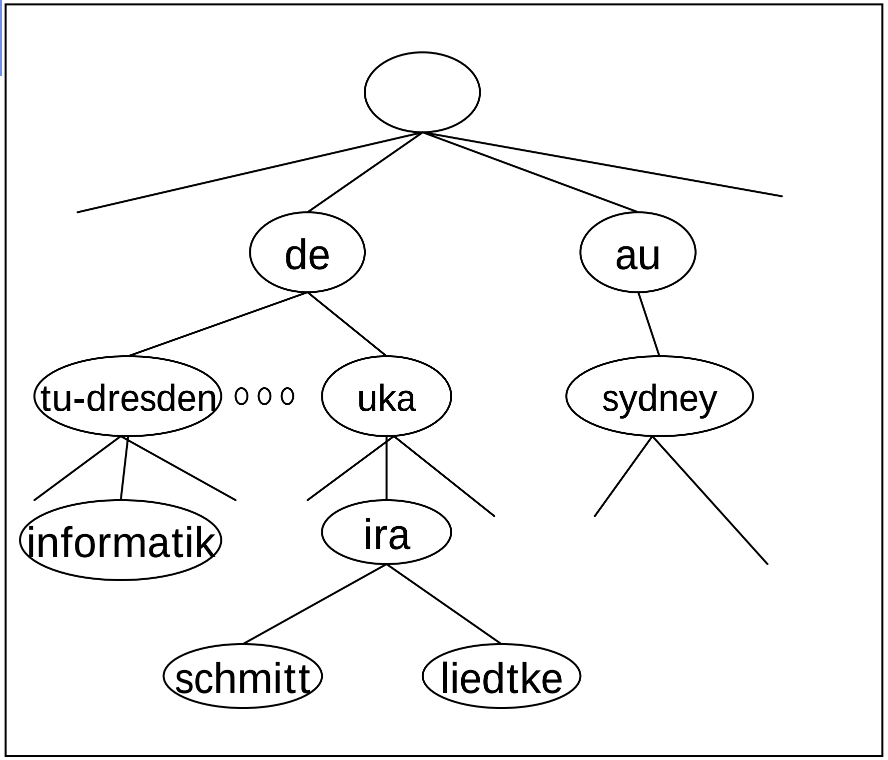
- Name agent sends query to node N.
- Name server of N maps O to a local object

Properties:

+ Name generation is simple
+ Name authority can be found in a simple and efficient way
+ Objects can migrate
- Additional overhead for the monitoring of migrated objects
- Only one (fixed) authority per object

© 2004 Universität Karlsruhe, System Architecture Group

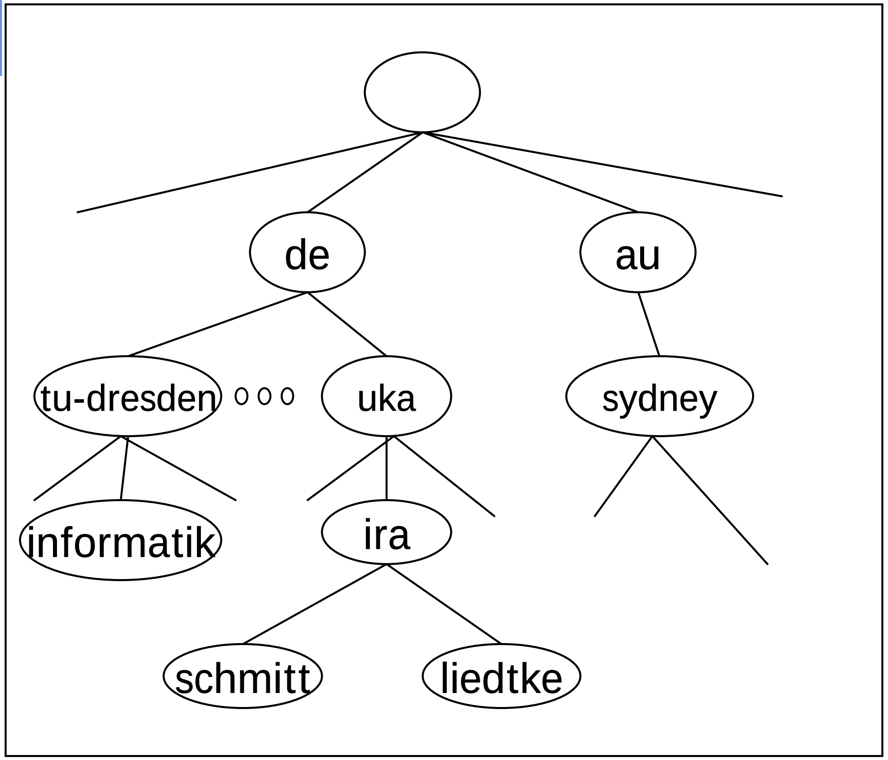
Location-Independent Names (1)



Structure of the name space:

- Hierarchically structured NS
- Hierarchy can resemble:
  - organizational structure
  - system topology
  - geographical distribution
  - etc.

Location-Independent Names (2)



Relative name of an object:

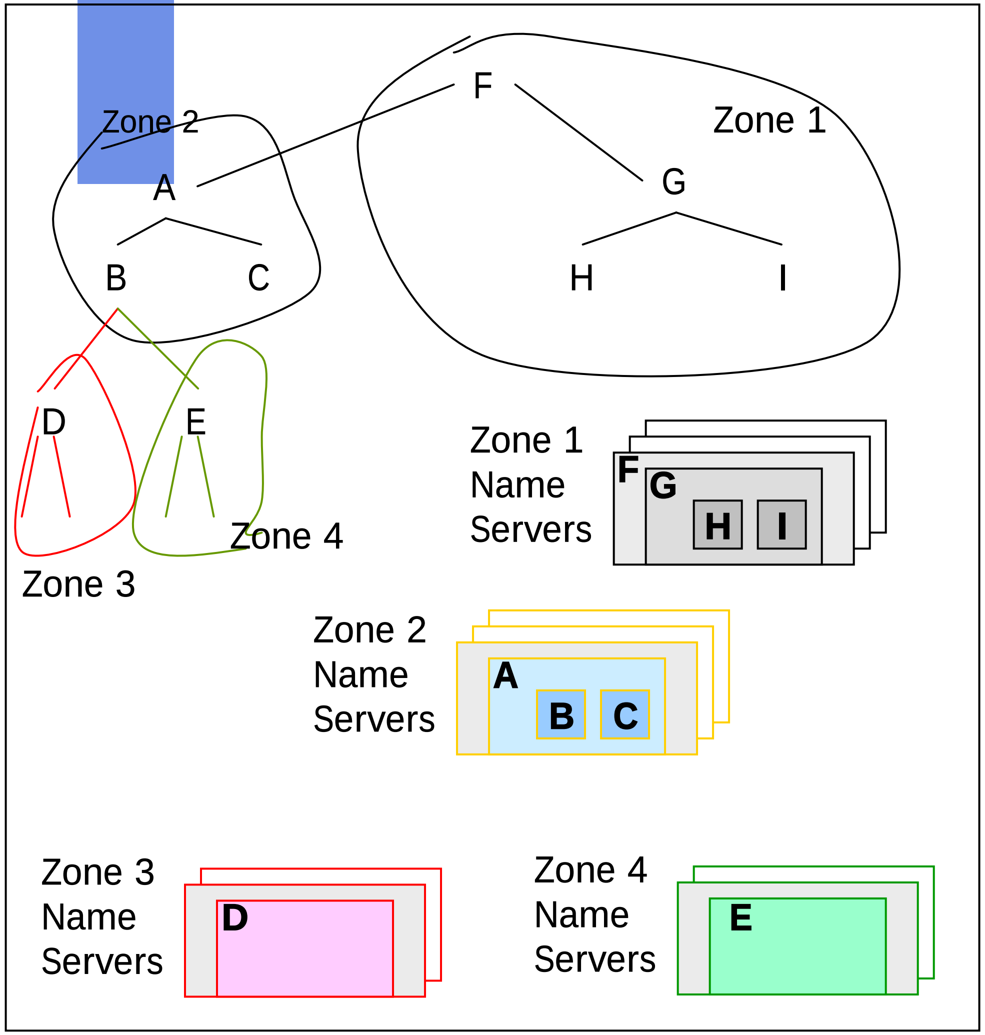
- Identifies a name uniquely in context of its parent object
- Example:
  uka/ira/schmitt (parent = de)

Absolute name of an object:

- Globally unique identifier for each object
- Concatenation of the relative names of all predecessors
- Example:
  /de/uka/ira/liedtke

© 2004 Universität Karlsruhe, System Architecture Group

Location-Independent Names (3)

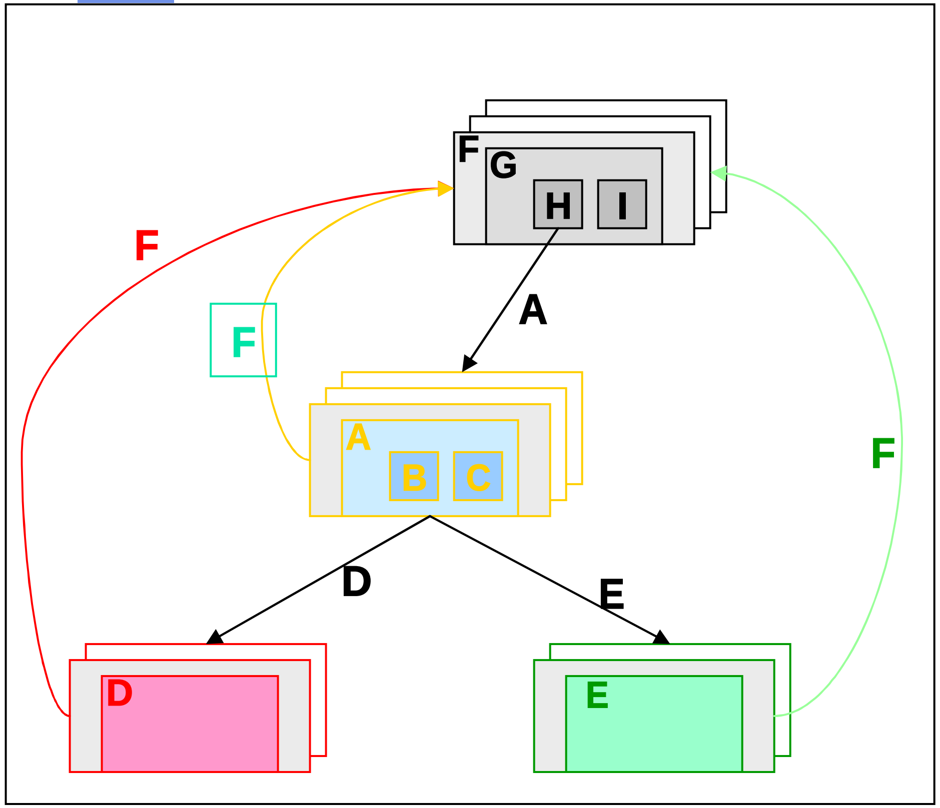


Partitioning of name space:

- Hierarchical name space is split up into zones
- Each zone has at least one name server
- Name server manages name space covered by the zone
- Replication achieved by having multiple name servers per zone

© 2004 Universität Karlsruhe, System Architecture Group

Location-Independent Names (4)



Every name server knows

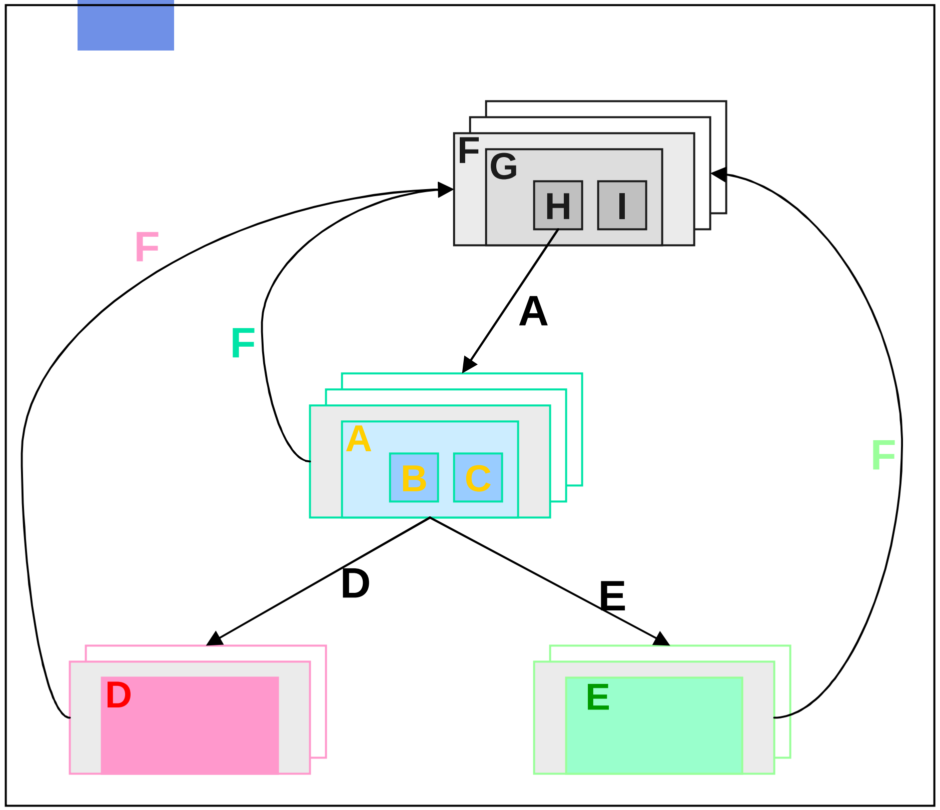
- ... at least one server of root zone
- ... the root zone of every child zone and at least one name server of it
- Name server manages the name

Name resolution:

- Agent asks arbitrary name server N
- If N knows the name
  ⇒ local resolution
- If N does not know the name
  ⇒ Resolution starts with a root server
  ⇒ ... and works its way down the server hierarchy until the name has been completely resolved

© 2004 Universität Karlsruhe, System Architecture Group

Location-Independent Names (5)



Name resolution

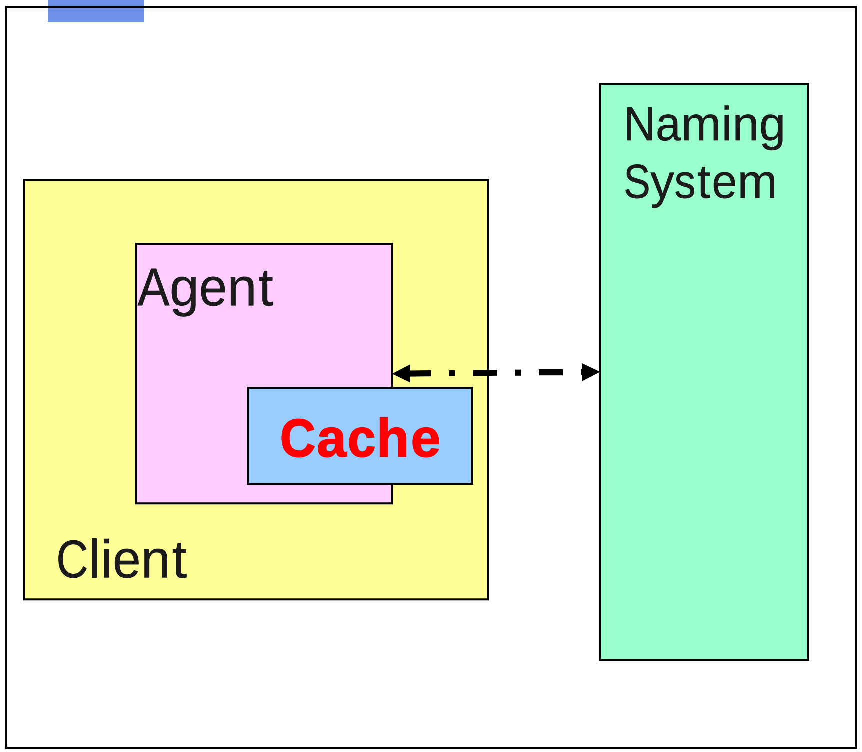
- ... is done by the authority responsible for the object

Properties:

+ Objects can migrate
+ Authority for an object may change
+ Multiple authorities possible (replication)

- Resolution may be expensive (Efficiency improved by caching)
- Name generation may require communication

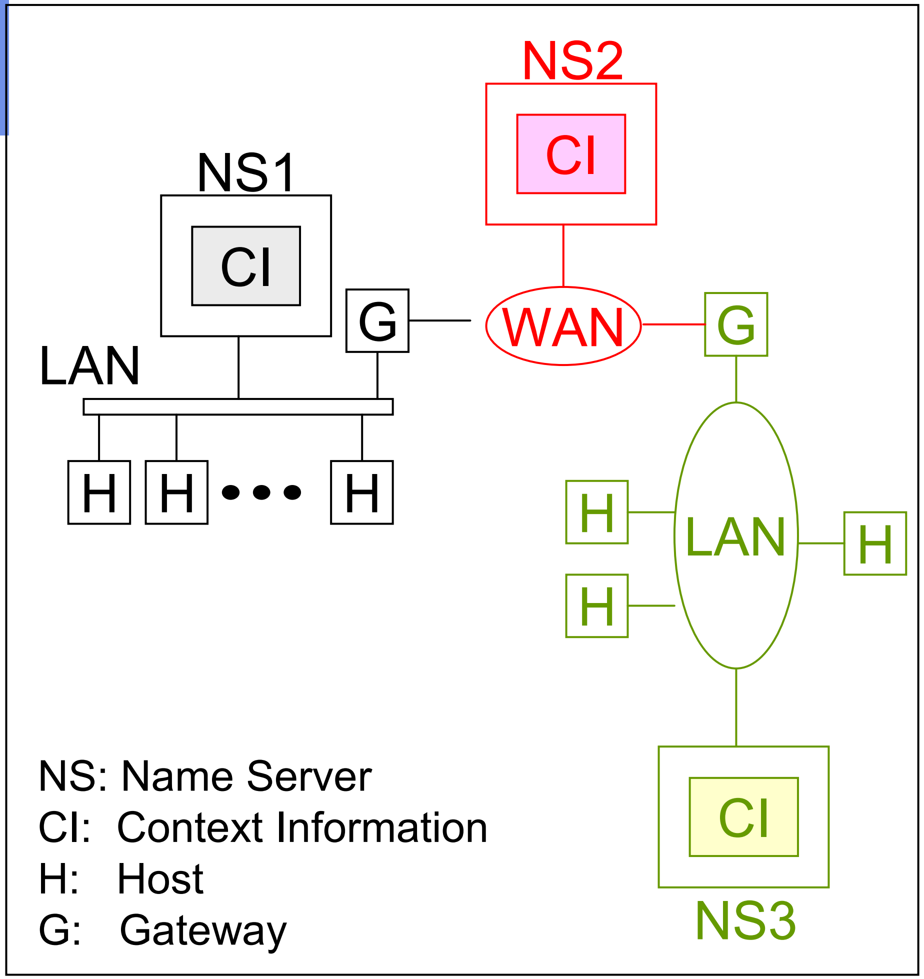
Caching



- Cache contains mapping information
- On receipt of a request for resolution, agent checks if required information is already or still in the cache
  - Yes: Resolution is done locally
  - No: Agent sends a query to the naming system; its result is stored in the cache
- A cache entry is invalidated when after a resolution it is found to be stale

© 2004 Universität Karlsruhe, System Architecture Group

Replication (1)



Goal:

- High availability, fast access

Pessimistic methods:

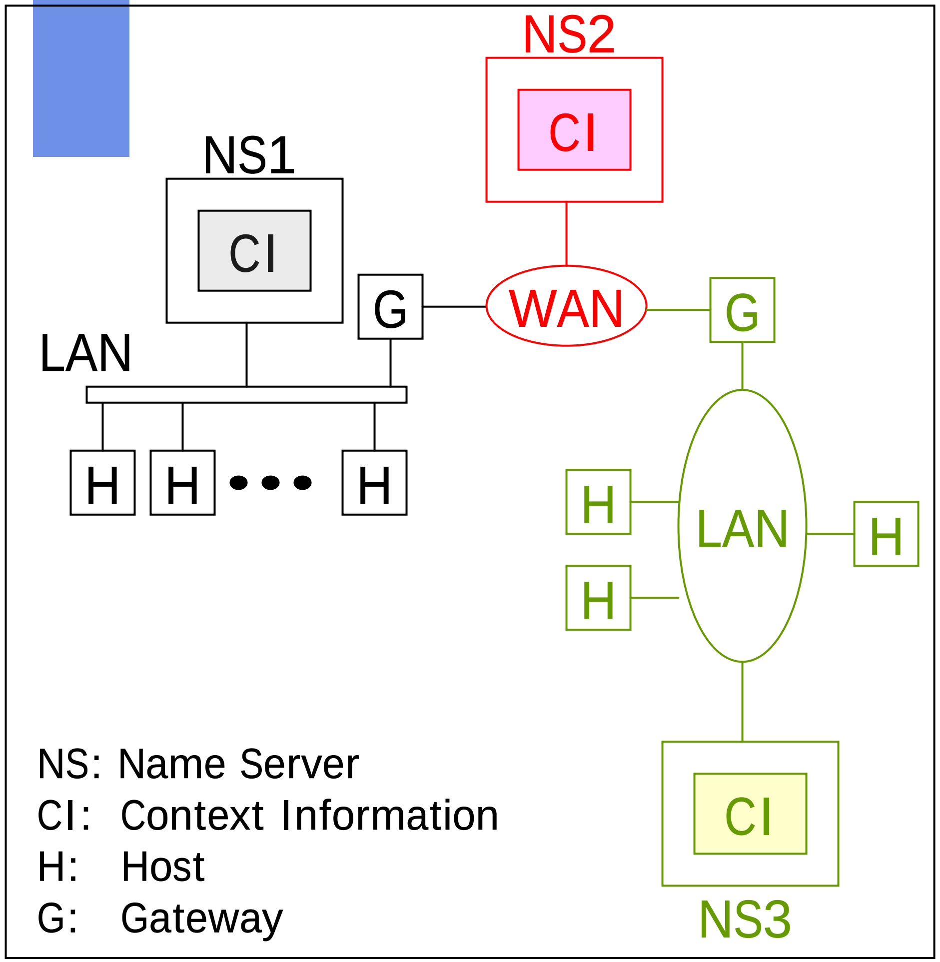
- One-copy serializability
- Limited availability if the network is partitioned

Optimistic methods:

- Weak consistency
- But: Available even if the network is partitioned

© 2004 Universität Karlsruhe, System Architecture Group

Replication (2)



Weak consistency is sufficient:

- If naming information is changed *infrequently*
- Even if stale information is used, in many cases no harm will be done
- If use of the information will reveal that it is stale

Some Examples

- DNS: The Internet Domain Name System
- Jini: Network Technology
- GNS: Global Navigation System
- NIS - Network Information Service
- X.500
- LDAP - Lightweight Directory Protocol

© 2004 Universität Karlsruhe, System Architecture Group

Domain Name System*

DNS = distributed database used by TCP/IP applications
to map between hostnames and IP addresses +
to provide electronic mail routing information

Primary task:

- Mapping from a symbolic name to the 32 bit Internet(IP) address, e.g.

  `mcculler.uni-karlsruhe.de` → **129……**

- General functionality:
- Mapping from hierarchical names to objects

*Paul Mockapetris (1984, standard in the Internet since 1987)

© 2004 Universität Karlsruhe, System Architecture Group

Domain Name System

- Name space: tree, edges labeled
- Name of incoming edge identifies a node
- Each subtree is called a *domain*
- Sequence of edge labels to the root is a *domain name*
- Each node contains a *resource record*

Domain Name System

Name space:

manages names for: computer, email-server, ...

Name structuring:

- hierarchical name space
- pathnames start at the leave and end at the root, e.g. i30www.ira.uka.de
- Top level domain has fixed names (can be changed only by an official Internet-office) below top level domain you can have as many levels as you need

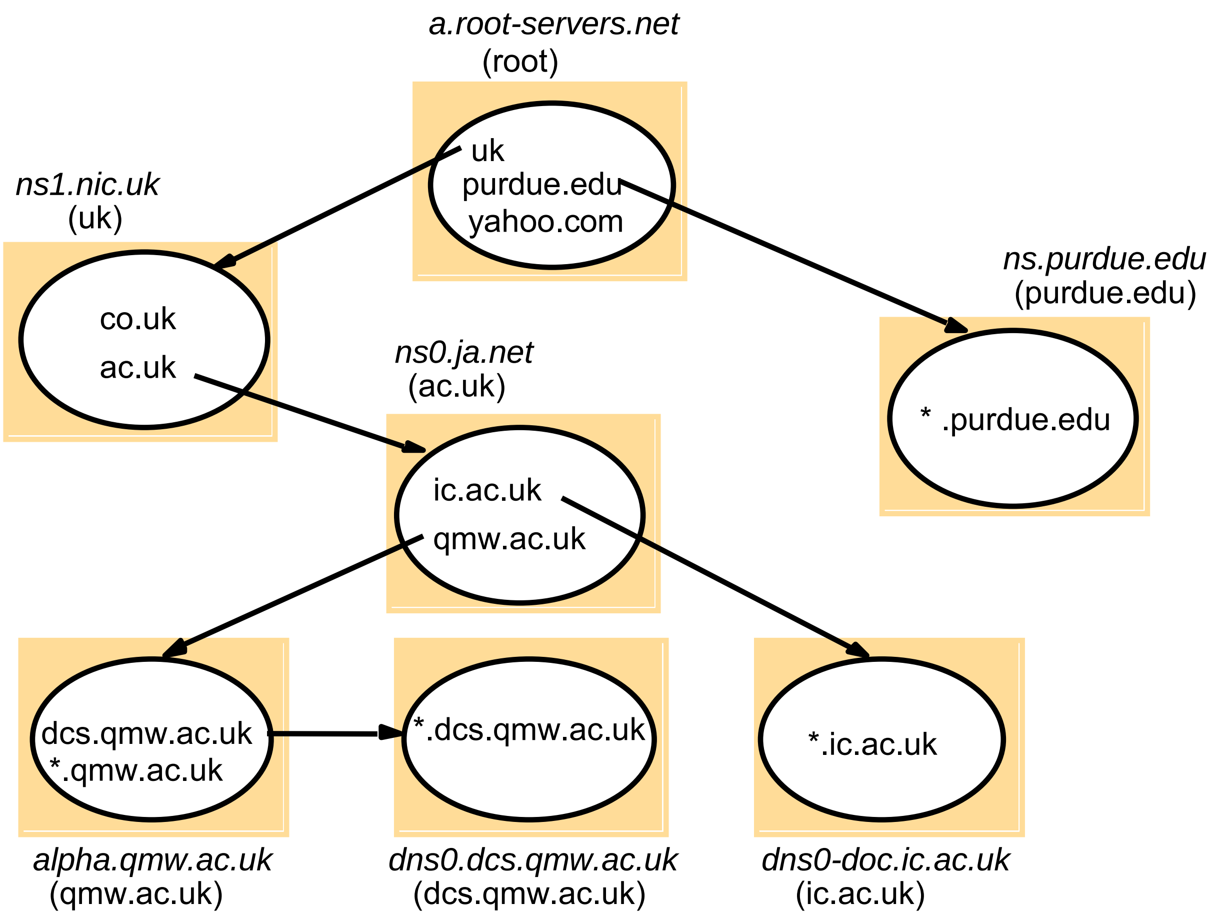
© 2004 Universität Karlsruhe, System Architecture Group

DNS Top-Level Domain

Domain Name	Meaning
com	Commercial bussiness
edu	Universities (colleges) in USA
gov	Government departments(USA)
mil	Military institutions
net	Netprovider
org	All other business
arpa	Temporal ARPA-domain
int	International organisations
Zip code of Country(e.g. de)	Abbreviations of all countries

© 2004 Universität Karlsruhe, System Architecture Group

DNS Name Servers



Note: Name server names are in italics, and the corresponding domains are in parentheses.

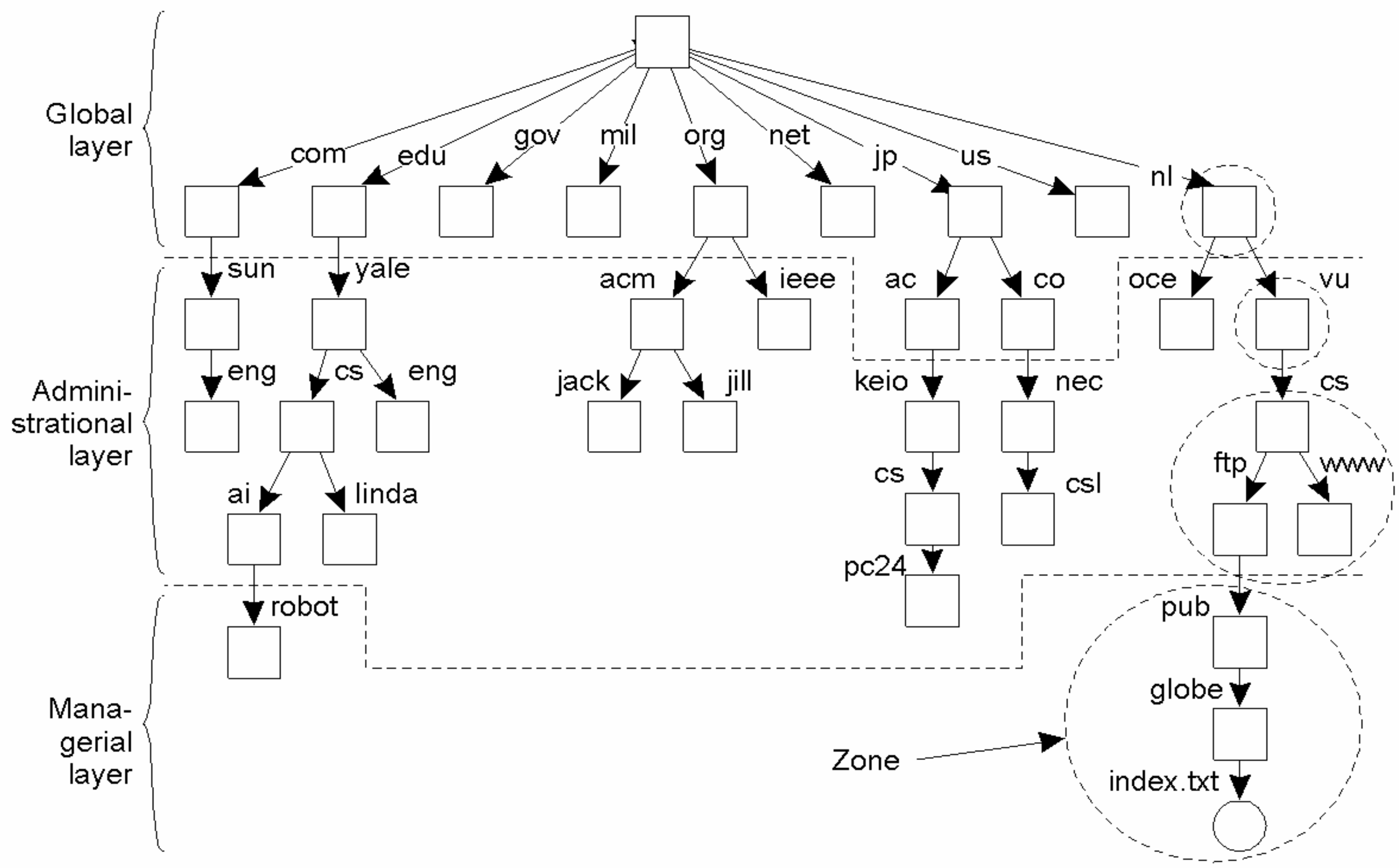
DNS Resource Records

Type of record	Associated entity	Description
SOA	Zone	Holds information on the represented zone
A	Host	Contains an IP address of the host this node represents
MX	Domain	Refers to a mail server to handle mail addressed to this node
SRV	Domain	Refers to a server handling a specific service
NS	Zone	Refers to a name server that implements the represented zone
CNAME	Node	Symbolic link with the primary name of the represented node
PTR	Host	Contains the canonical name of a host
HINFO	Host	Holds information on the host (OS + HW-type) this node represents
TXT	Any kind	Contains any entity-specific information considered useful

- Most important types of resource records forming the contents of nodes in the Internet’s DNS name space.

© 2004 Universität Karlsruhe, System Architecture Group

DNS Name Space



DNS Implementation

- An excerpt from the DNS database for the zone *cs.vu.nl*.

Name	Record type	Record value
cs.vu.nl	SOA	star (1999121502,7200,3600,2419200,86400)
cs.vu.nl	NS	star.cs.vu.nl
cs.vu.nl	NS	top.cs.vu.nl
cs.vu.nl	NS	solo.cs.vu.nl
cs.vu.nl	TXT	"Vrije Universiteit - Math. & Comp. Sc."
cs.vu.nl	MX	1 zephyr.cs.vu.nl
cs.vu.nl	MX	2 tornado.cs.vu.nl
cs.vu.nl	MX	3 star.cs.vu.nl
star.cs.vu.nl	HINFO	Sun Unix
star.cs.vu.nl	MX	1 star.cs.vu.nl
star.cs.vu.nl	MX	10 zephyr.cs.vu.nl
star.cs.vu.nl	A	130.37.24.6
star.cs.vu.nl	A	192.31.231.42
zephyr.cs.vu.nl	HINFO	Sun Unix
zephyr.cs.vu.nl	MX	1 zephyr.cs.vu.nl
zephyr.cs.vu.nl	MX	2 tornado.cs.vu.nl
zephyr.cs.vu.nl	A	192.31.231.66
www.cs.vu.nl	CNAME	soling.cs.vu.nl
ftp.cs.vu.nl	CNAME	soling.cs.vu.nl
soling.cs.vu.nl	HINFO	Sun Unix
soling.cs.vu.nl	MX	1 soling.cs.vu.nl
soling.cs.vu.nl	MX	10 zephyr.cs.vu.nl
soling.cs.vu.nl	A	130.37.24.11
laser.cs.vu.nl	HINFO	PC MS-DOS
laser.cs.vu.nl	A	130.37.30.32
vucs-das.cs.vu.nl	PTR	0.26.37.130.in-addr.arpa
vucs-das.cs.vu.nl	A	130.37.26.0

© 2004 Universität Karlsruhe, System Architecture Group

DNS Implementation

Name	Record type	Record value
cs.vu.nl	NIS	solo.cs.vu.nl
solo.cs.vu.nl	A	130.37.21.1

- Part of the description for the *vu.nl* domain which contains the *cs.vu.nl* domain.

Replication and Caching in DNS

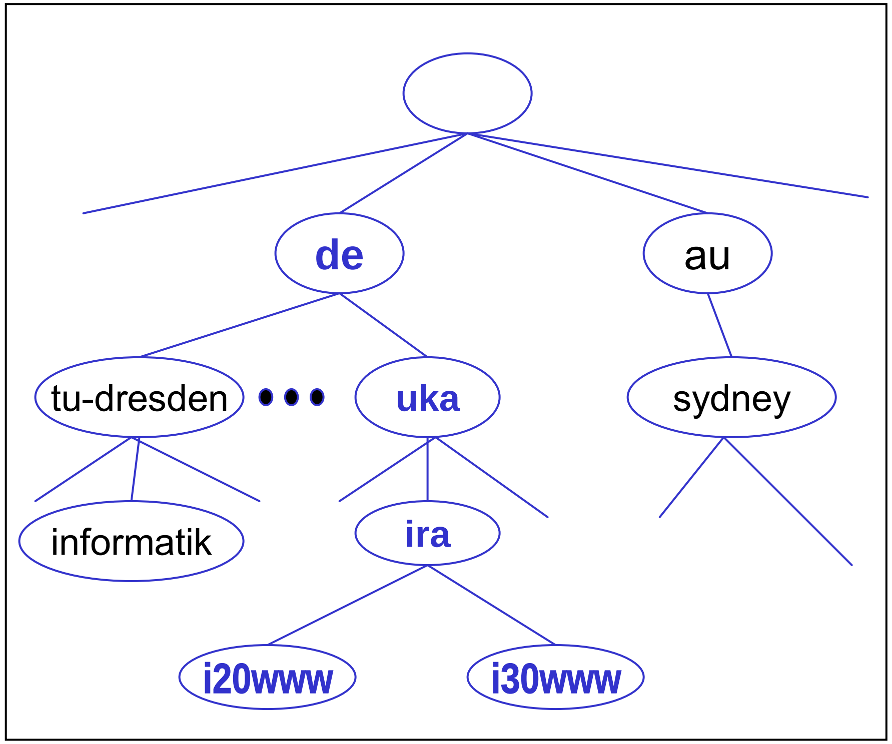
- Replication
  - for every root server there are at least 2 replicas
    - primary/backup principle
    - backup servers update their state periodically via primary server
- Caching
  - each name server implements caching

Further reading:
F. Halsall: “Data Communications, Computer Networks and Open System”, Addison-Wesley 1992

D. Comer: “Computernetzwerke und Internets”, Prentice Hall 1997

© 2004 Universität Karlsruhe, System Architecture Group

DNS Name Space

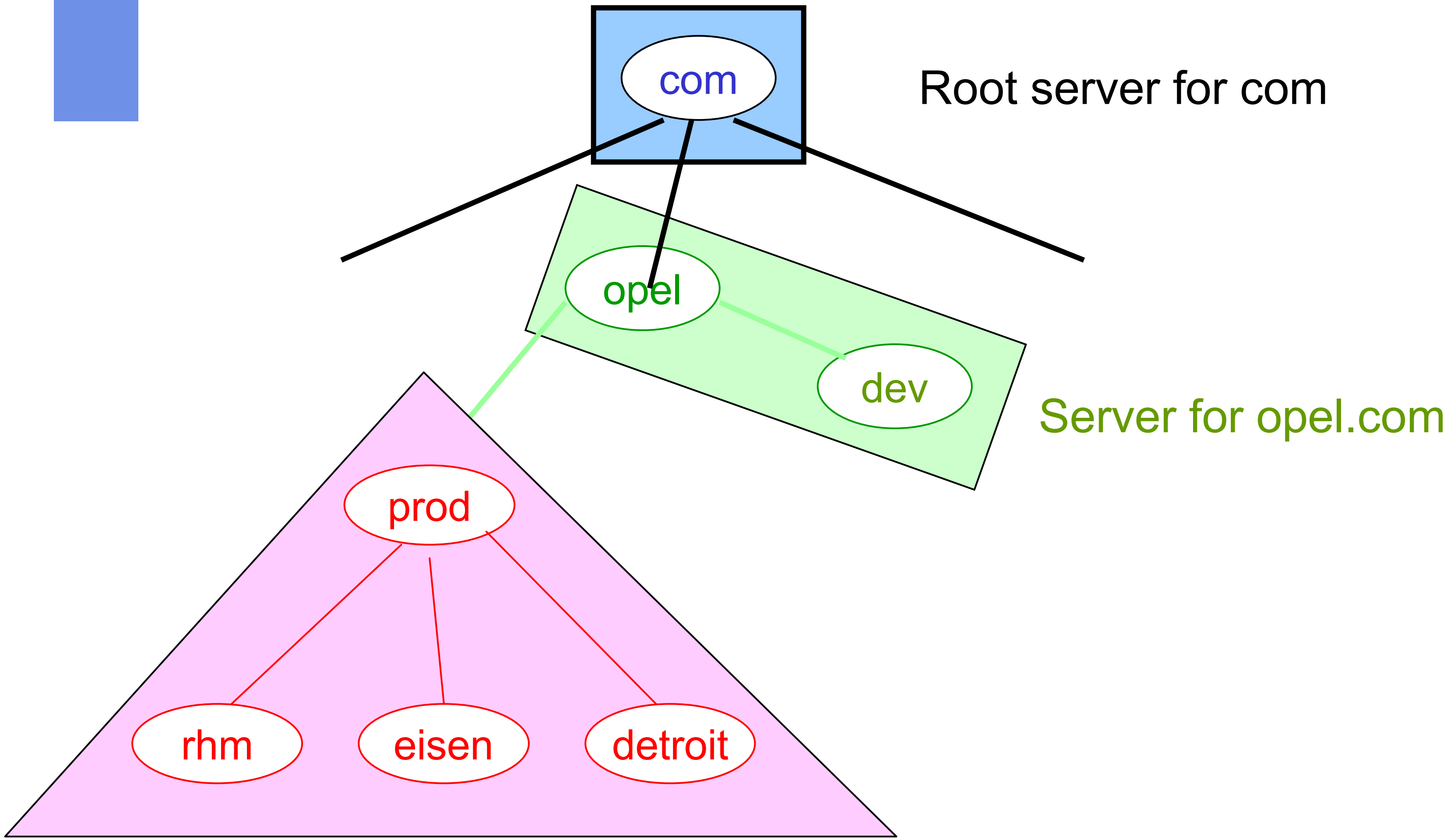


- Hierarchical, location-independent names
- Name space is split up into domains being ordered in a *tree-like fashion*
- Each domain receives a domain name being *unique* among its sibling nodes
- Absolute name of a domain is obtained by concatenating the relative names on the path from this domain to the root of the tree

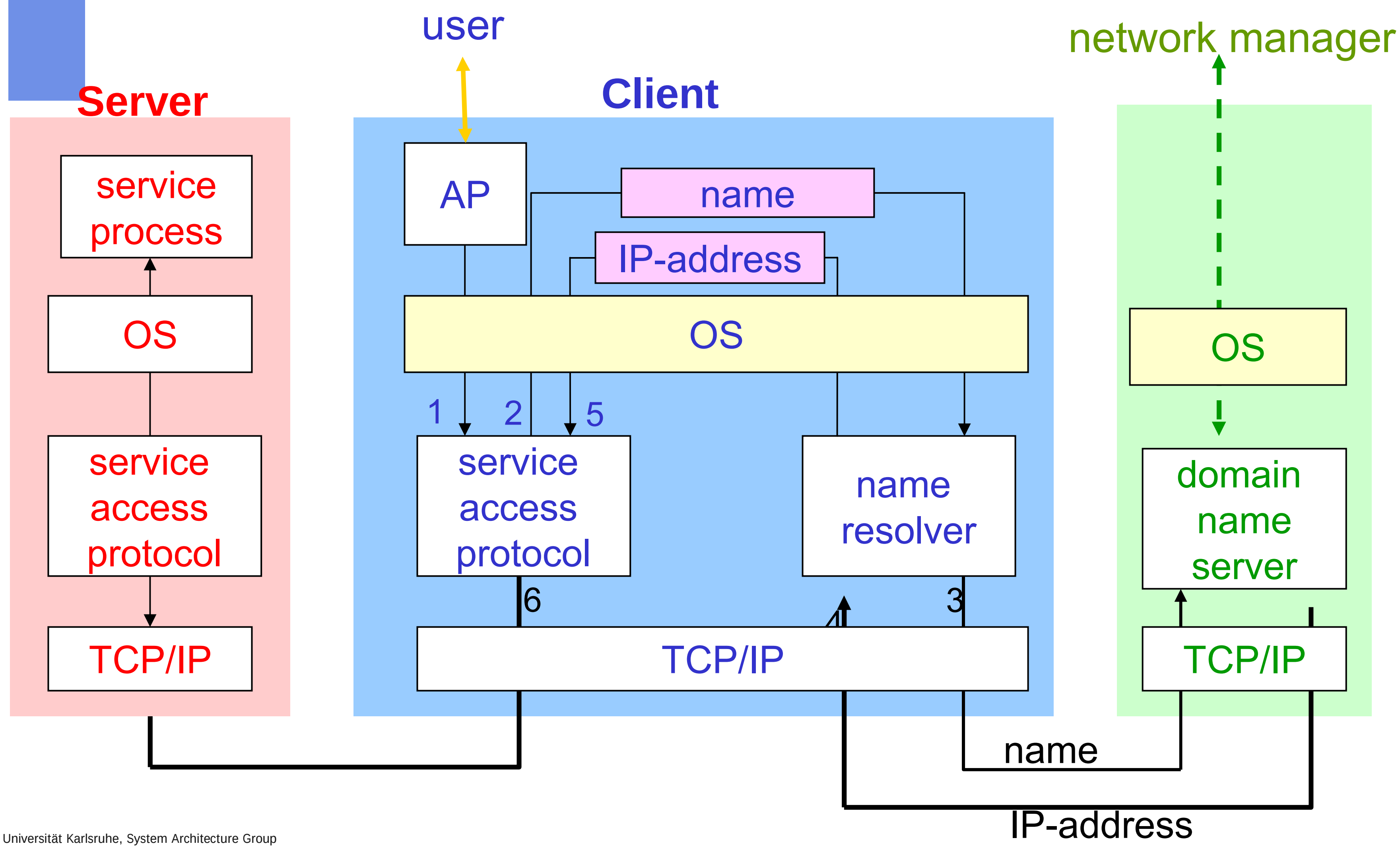
Example: i30www.ira.uka.de

© 2004 Universität Karlsruhe, System Architecture Group

DNS Name Server Hierarchy

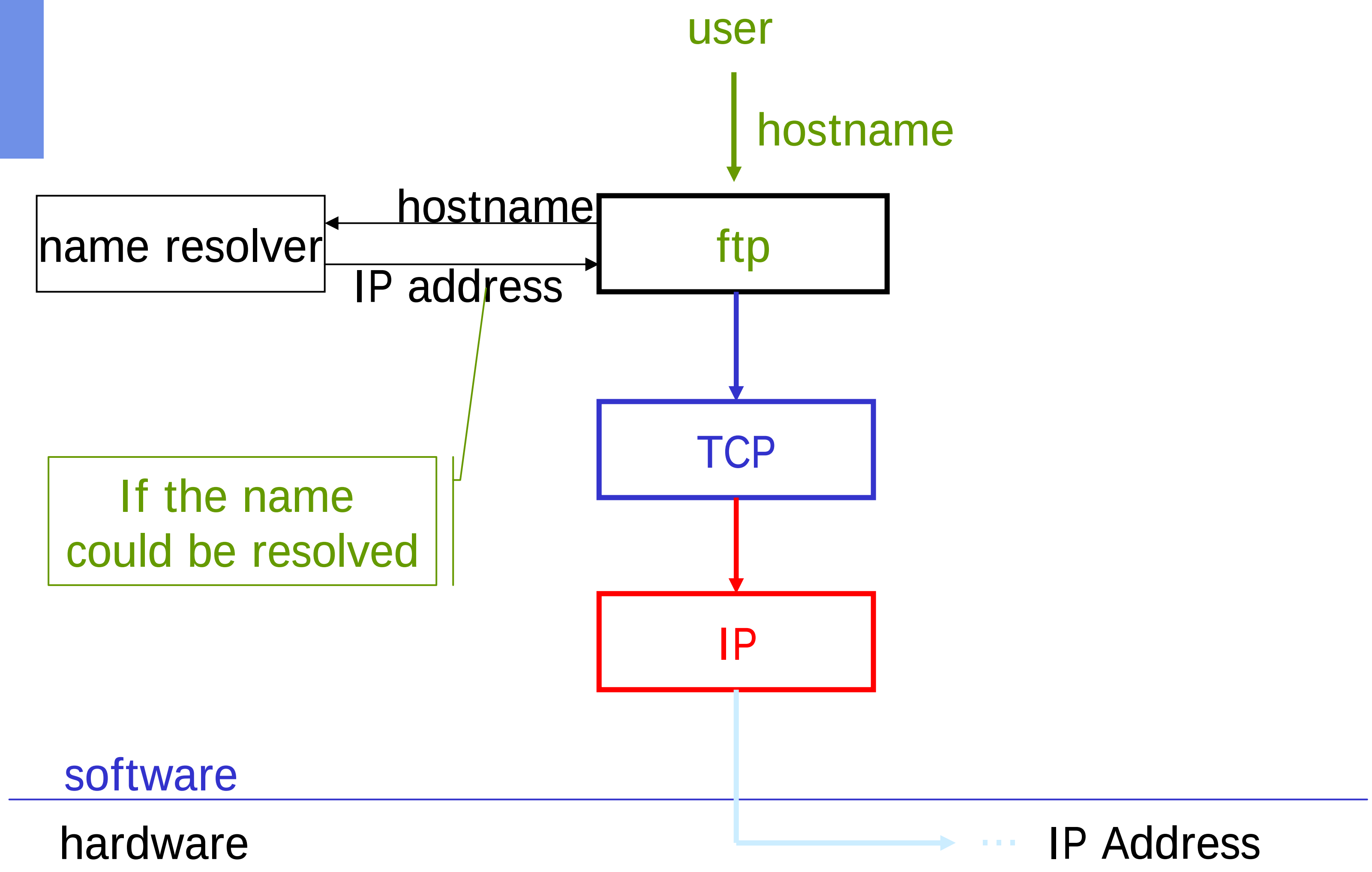


DNS System Architecture

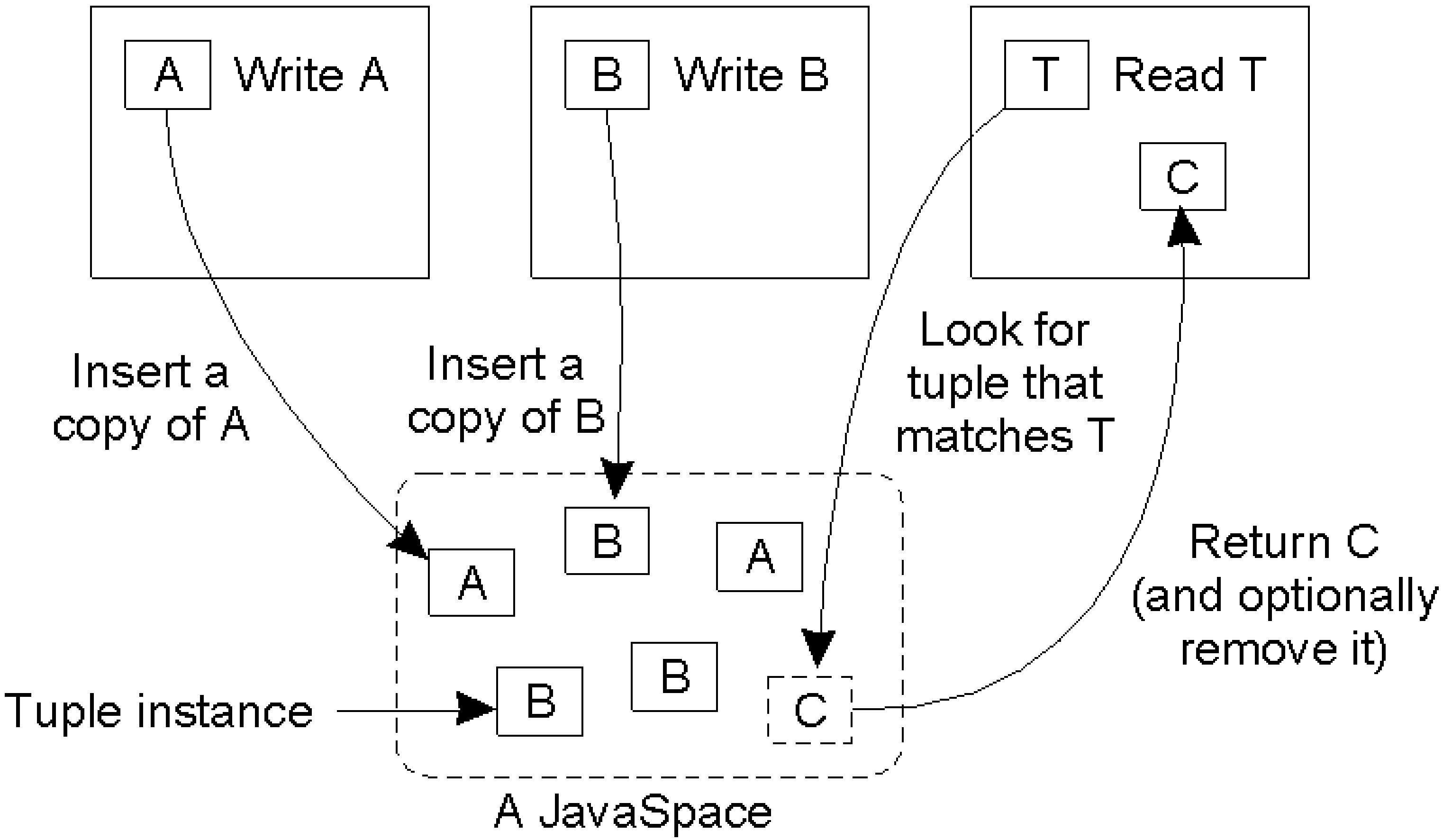


© 2004 Universität Karlsruhe, System Architecture Group

Example

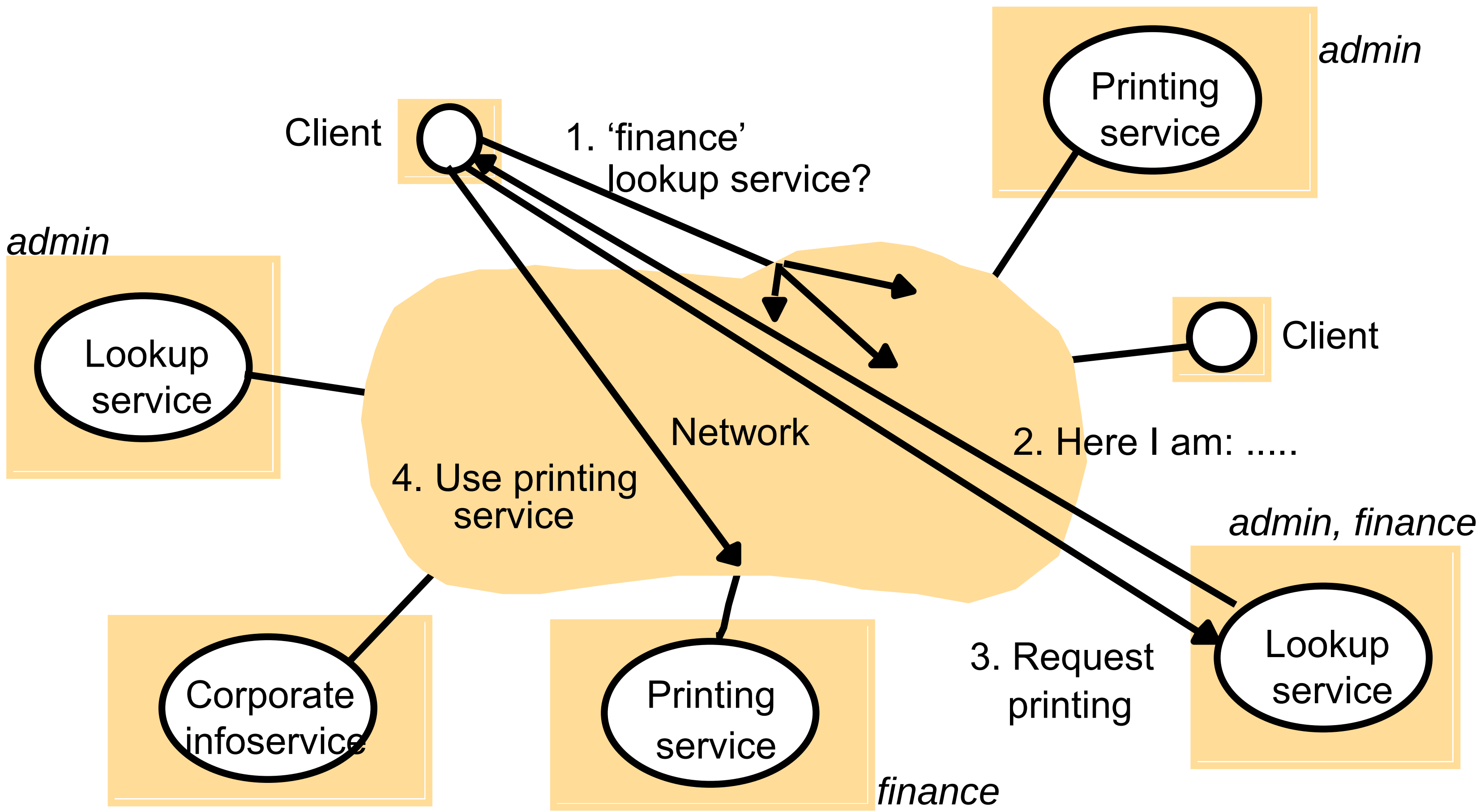

user
hostname
name resolver
hostname
IP address
ftp
If the name could be resolved
TCP
IP
software
hardware
... IP Address

Overview on Jini



- The general organization of a JavaSpace in Jini.

Service Discovery in Jini

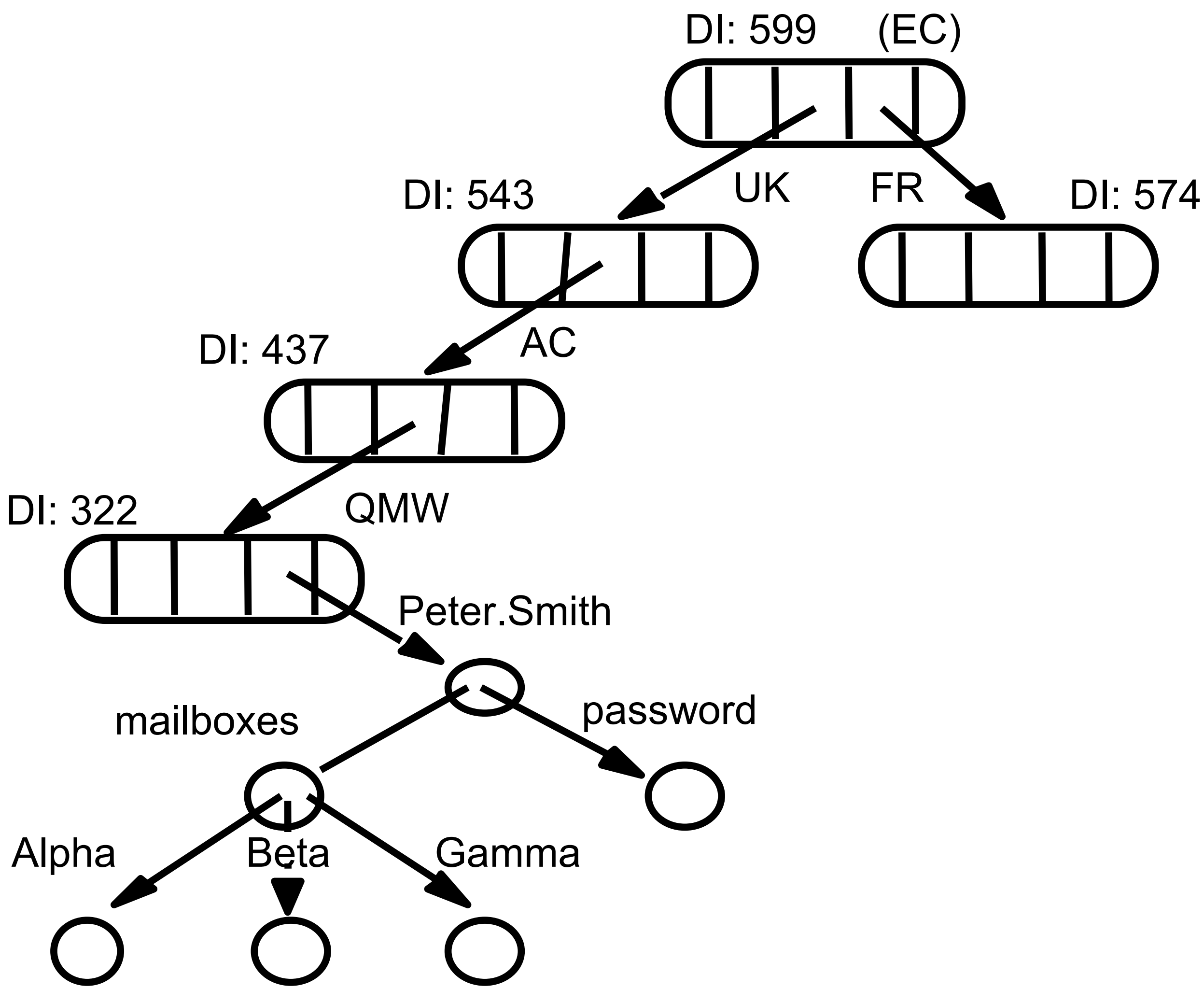


Books on Jini

- “Jini Technology: An Overview” by Ilango Kumaran Prentice Hall PTR (November 2001),
  "For all managers, architects and consultants seeking to evaluate Jini from a technological and architectural perspective, and compare it with other work."

- “ Developing Jini Applications Using Java 2 “ by Hinkmond Wong
  Addison Wesley Longman (September 2001),
  "This book is your key to understanding and avoiding common traps and pitfalls that await developers approaching Jini and J2ME technology for the first time. “

Global Name Service (GNS)



Directory Service: X 500

CCITT and ISO standard (1988):

Names

- List of tuples <attribute = value)
- Attributes
  - country "c"
  - organization "o"
  - organizational unit "ou"
  - surname "sn"

  …
  - telephone number "telephone"

Example:

/c=de/o=uni-karlsruhe/ou=rz/sn=zoller/telephone=+49 721 608 405

© 2004 Universität Karlsruhe, System Architecture Group

Directory Service: X.500

- A *Directory Service* supports lookup based on a set of attribute values (*yellow pages*)
- Directory entries contain <attrib, value> pairs
- Set of entries forms Directory Information Base (DIB).
- *Naming attributes* of an entry jointly identify an entry uniquely.
- Canonical sequences of naming attributes form the Directory Information Tree (DIT)
  - Edges are labeled with <attrib, value> pairs
- Each name attribute is a so called RDN (relative distinguished name)

© 2004 Universität Karlsruhe, System Architecture Group

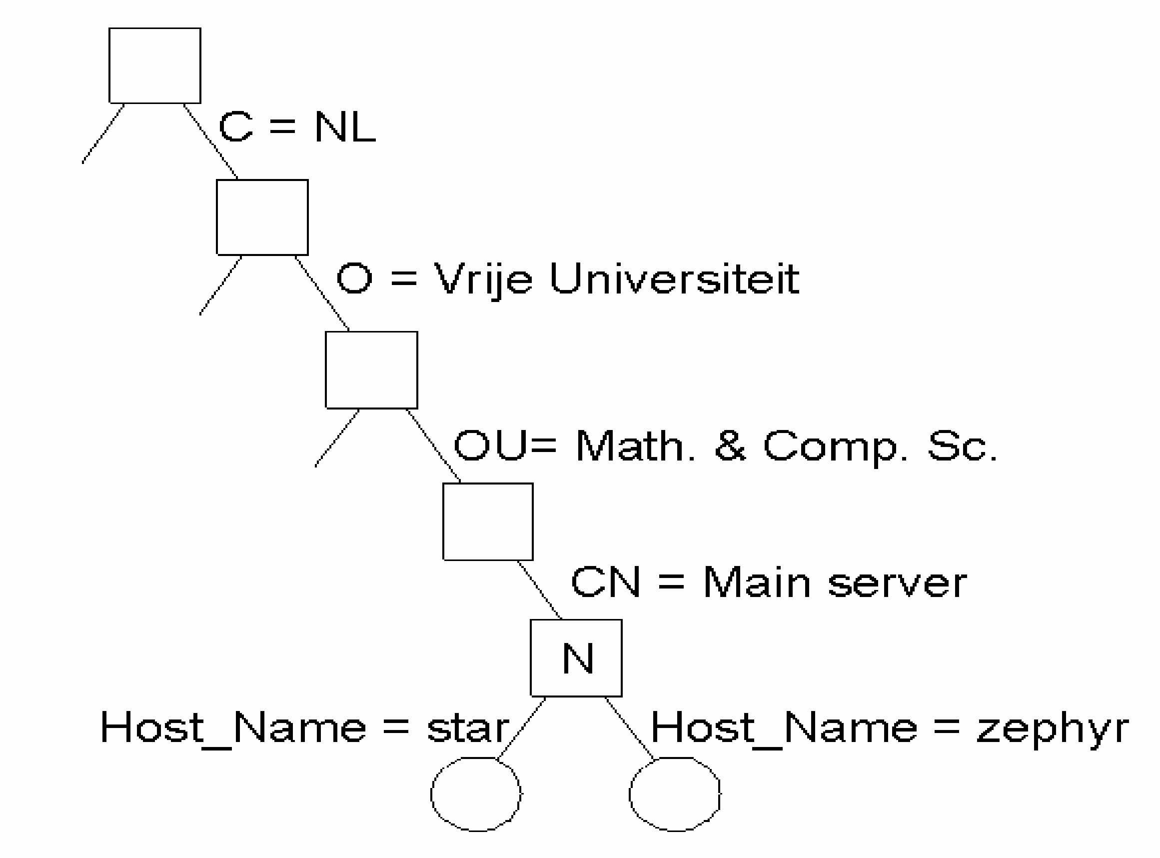
The X.500 Directory Entries

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Math. & Comp. Sc.
CommonName	CN	Main server
Mail_Servers	--	130.37.24.6, 192.31.231,192.31.231.66
FTP_Server	--	130.37.21.11
WWW_Server	--	130.37.21.11

- A simple example of a X.500 directory entry using X.500 naming conventions.

© 2004 Universität Karlsruhe, System Architecture Group

The X.500 DIB



- Part of the directory information tree

© 2004 Universität Karlsruhe, System Architecture Group

The X.500 Name Space

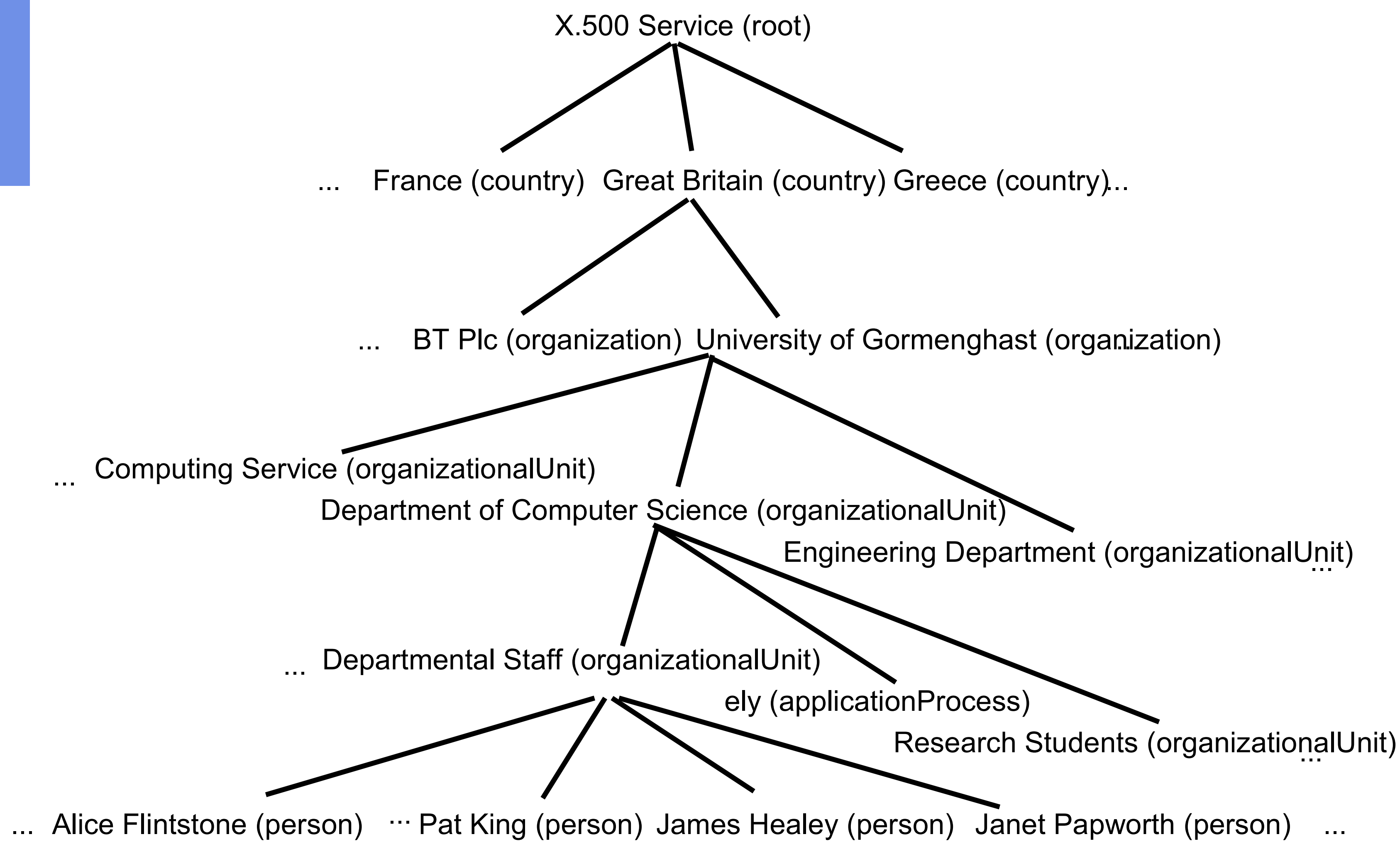
Attribute	**Value**
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Math. & Comp. Sc.
CommonName	Main server
Host_Name	**star**
Host_Address	**192.31.231.42**

Attribute	**Value**
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Math. & Comp. Sc.
CommonName	Main server
Host_Name	**zephyr**
Host_Address	**192.31.231.66**

- Two directory entries having *Host_Name* as RDN

© 2004 Universität Karlsruhe, System Architecture Group

X.500 Directory Information Tree



X.500 Lookup

Name lookup

```
list(/C=NL/O=Vrije Universiteit/
   OU=Math.&Comp.Sci./CN=Main server)
```

returns corresponding names

```
star zephyr
```

Directory lookup

```
search &(C=NL)(O=Vrije Universiteit)
       (OU=*)(CN=Main server)
```

returns all entries with matching attributes

Locating Mobile Entities

- Up to now we always had a mapping from:
  User friendly names → node addresses

- Mobile systems often change either their names or their addresses in a DS, we need a better solution for looking up a specific node:
  - Name → identifier
  - Identifier → address

© 2004 Universität Karlsruhe, System Architecture Group

Migration of Service

Example:

Assume we move ftp.cs.vu.nl to a new machine in a very far away domain, suppose: ftp.cs.unisa.edu.au

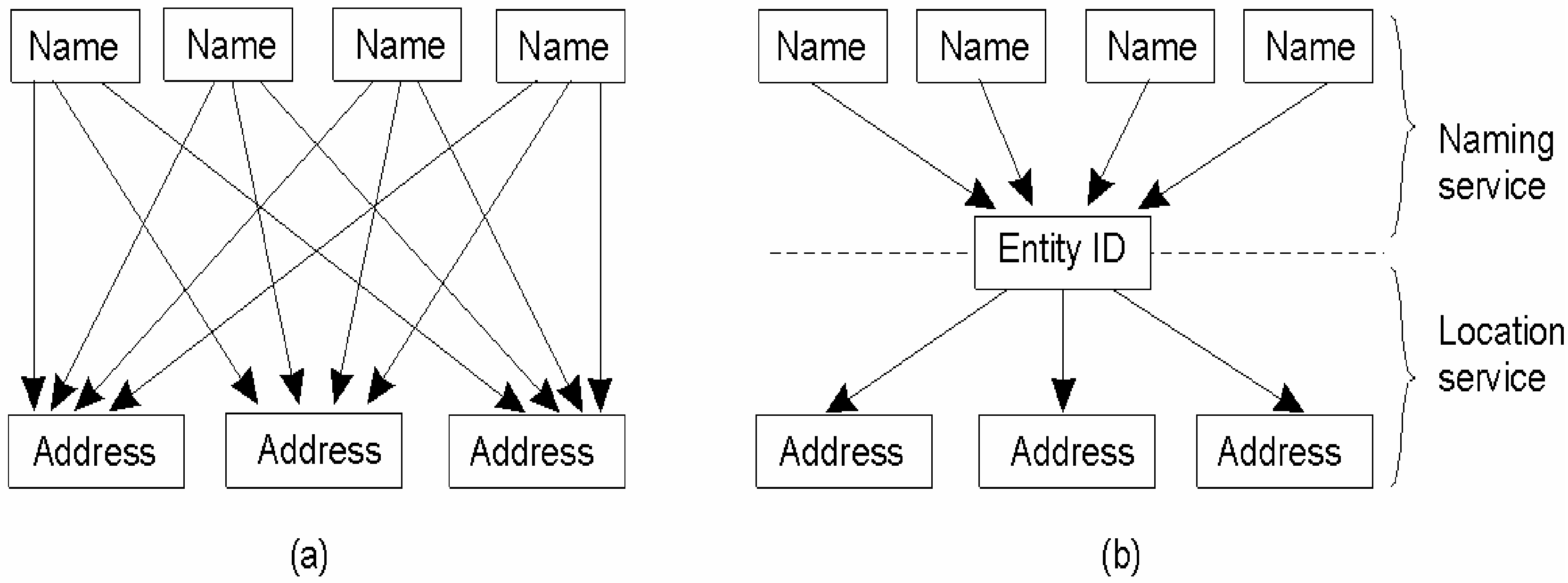
However, we would like to use ftp.cs.vu.nl , because many applications might contains program fragments using this name a as symbolic link, i.e. this name will be used as an ID, thus hanging this ID, all links to it will be invalid.

© 2004 Universität Karlsruhe, System Architecture Group

Two Principle Solutions

1. Insert the address of the new machin in the DND data base for the entry cs.vu.nl
   - No implications for look ups
   - However, if this ftp-service will migrate again to some other place, also its entry in the DNS data base has to be updated
2. Insert the name of the new machione instead of its address (thus ftp.cs.vu.nl becomes a symbolic name link).
   - Loop ups are less efficient due to its 2 steps
     1. Look for the name of the new machine
     2. Look for the address related to this new name

Naming versus Locating Entities



a) Single level mapping between names and addresses

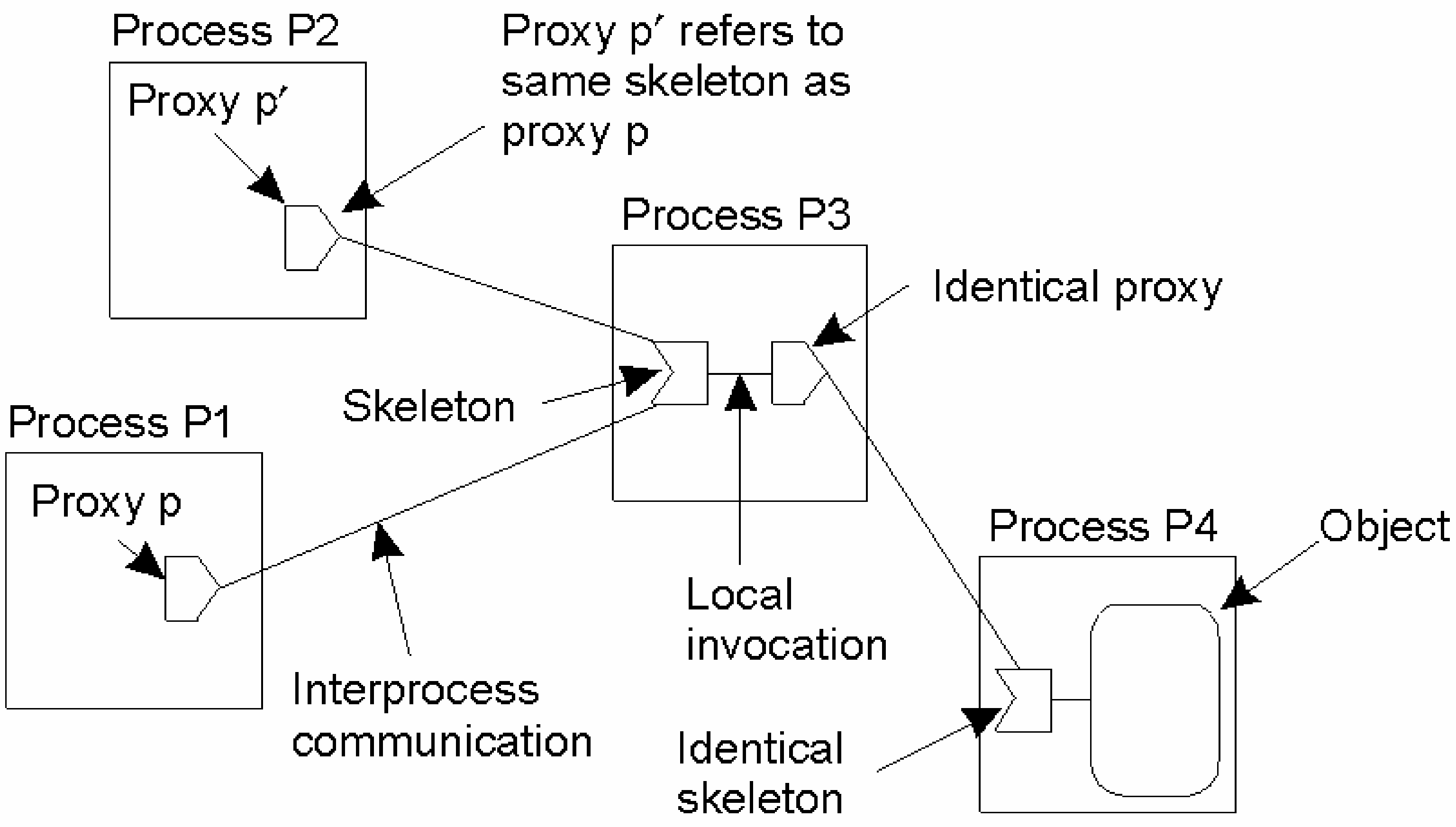
b) T-level mapping using identities.

Location Services

- Simple approaches (in general, LAN only)
- Broadcast id, corresponding node responds with its current address (ex.: ARP)
- Multicast id to a well-known group, all mobile nodes subscribe to that group

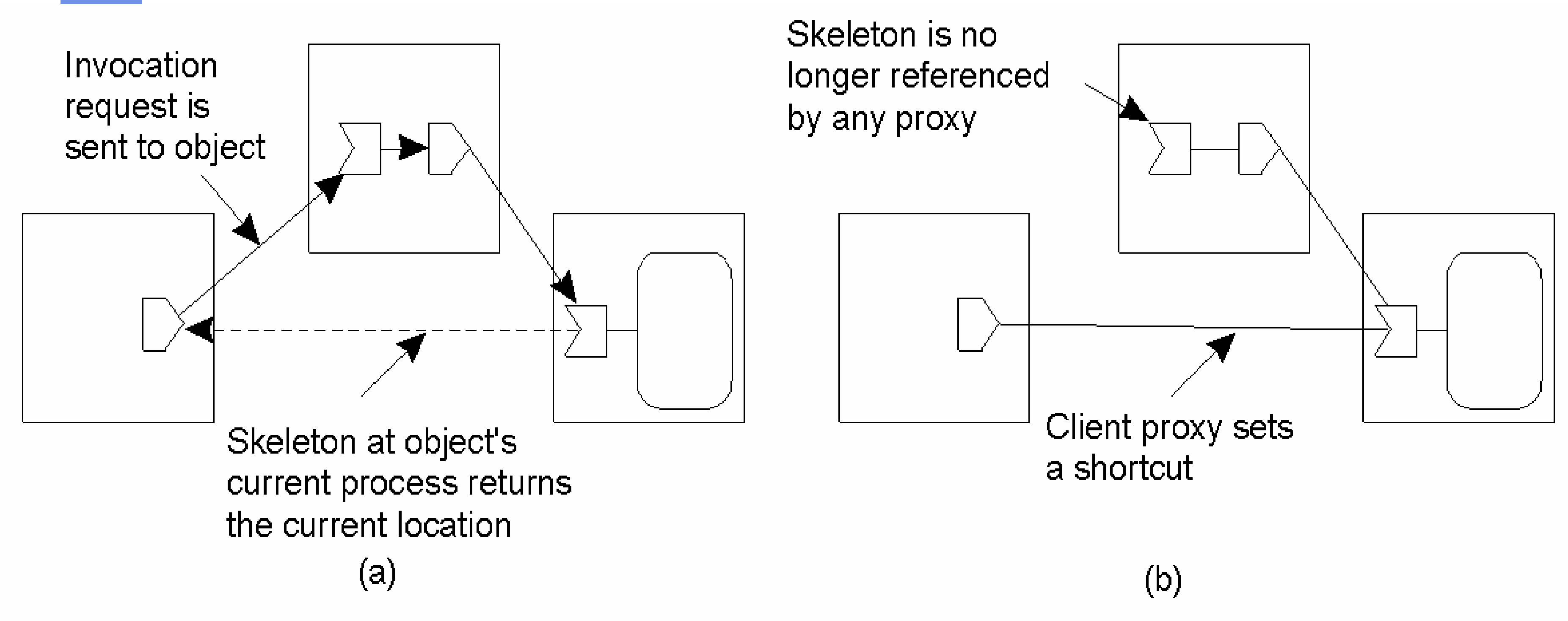
© 2004 Universität Karlsruhe, System Architecture Group

Forwarding Pointers (1)



- Principle of forwarding pointers using (*proxy, skeleton*) pairs

Forwarding Pointers (2)



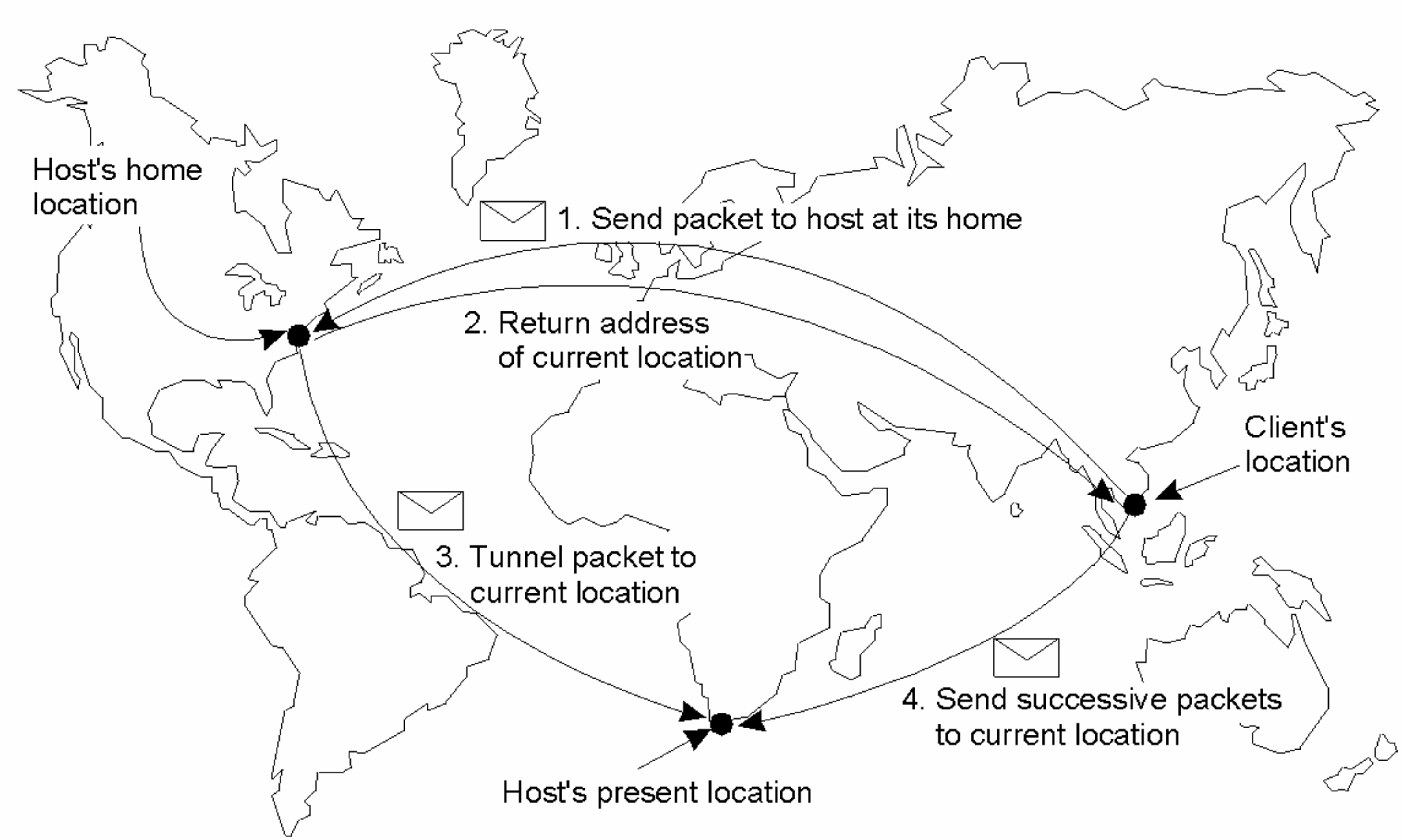
Redirecting a forwarding pointer, by storing a shortcut in a proxy

© 2004 Universität Karlsruhe, System Architecture Group

Location Home-Based Approach

- Can be used as a fall-back mechanism with forwarding pointers
- A dedicated *home node* always keeps an up-to-date pointer to the mobile object
- Home node is typically node where an object originates
- Make home node *fault-tolerant* to ensure object can always be located

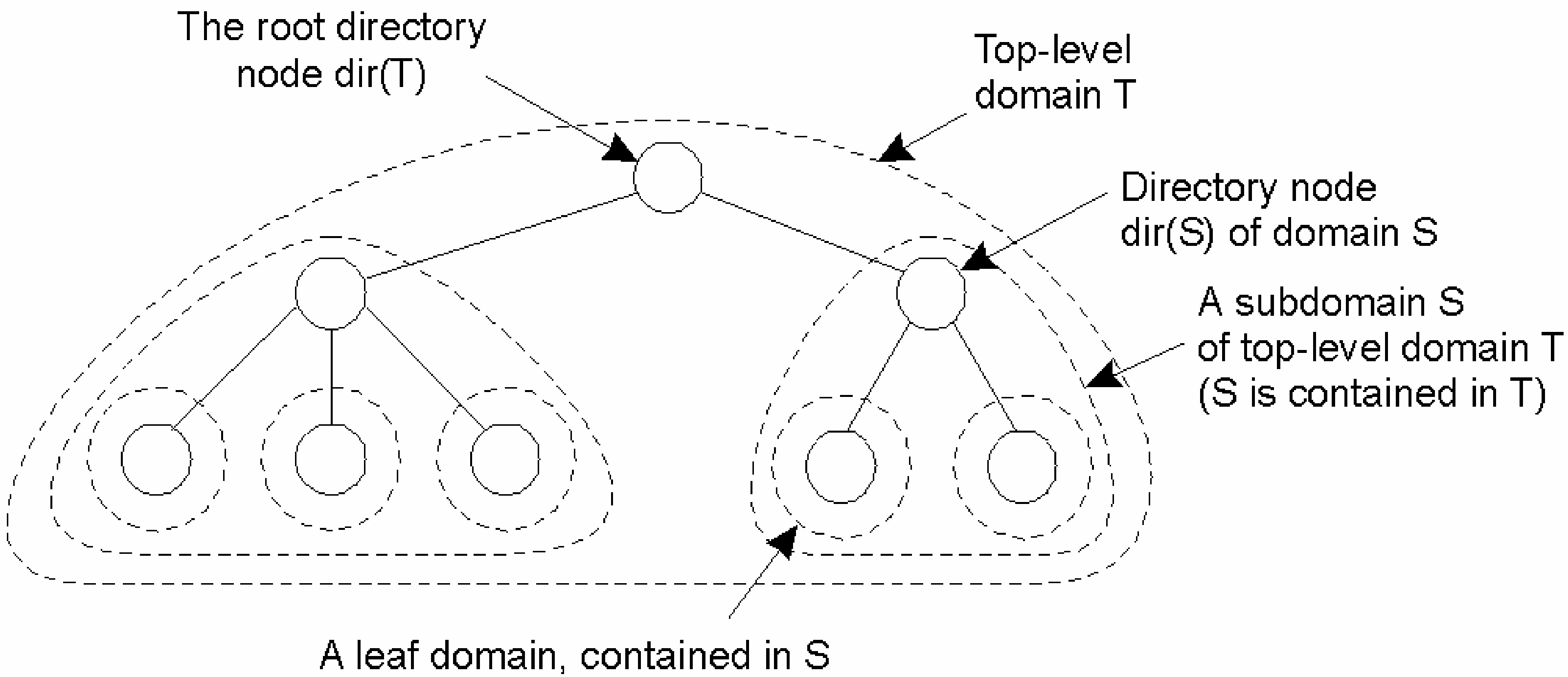
Home-Based Approaches



- The principle of Mobile IP

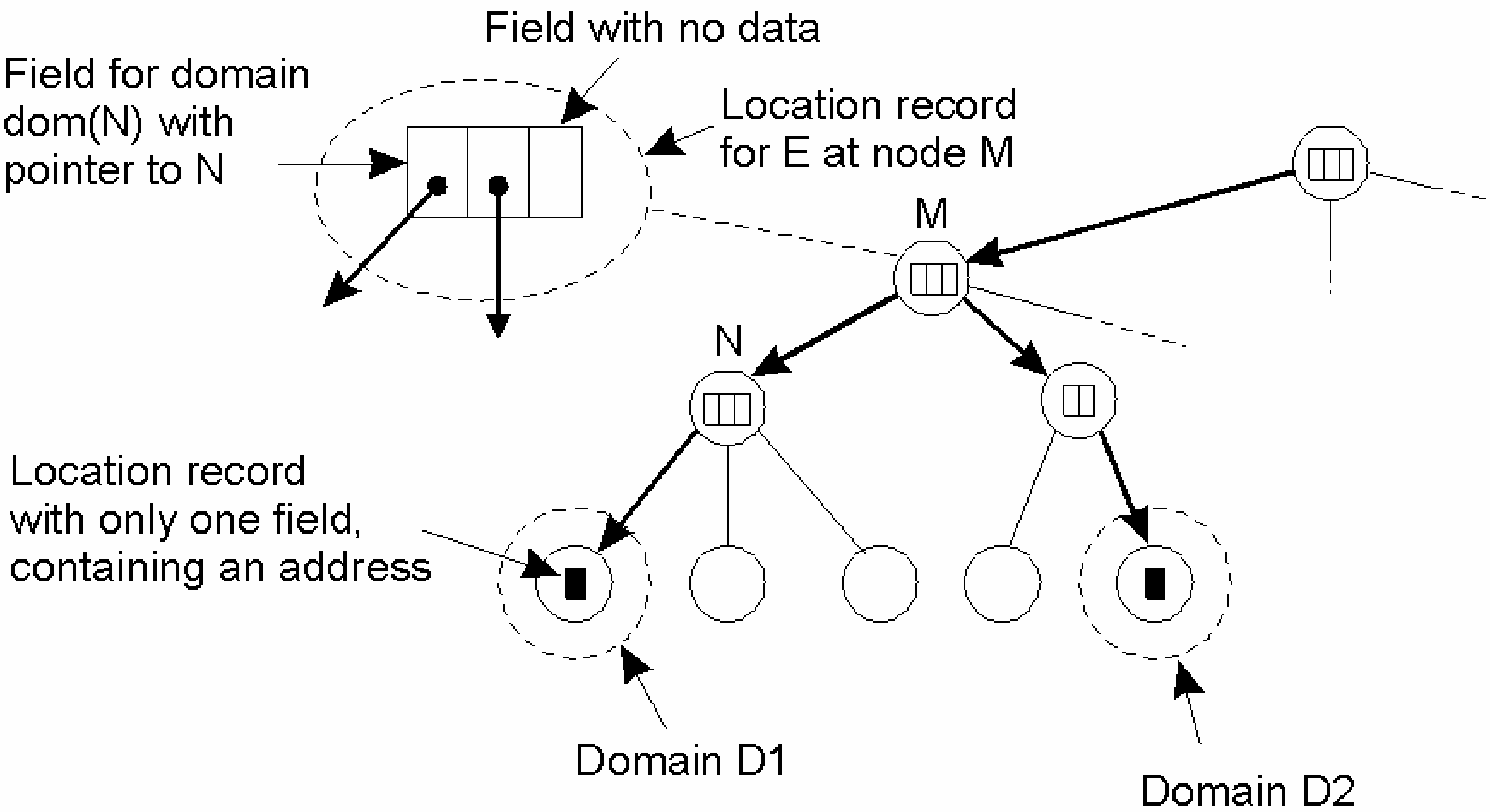
© 2004 Universität Karlsruhe, System Architecture Group

Hierarchical Approaches (1)



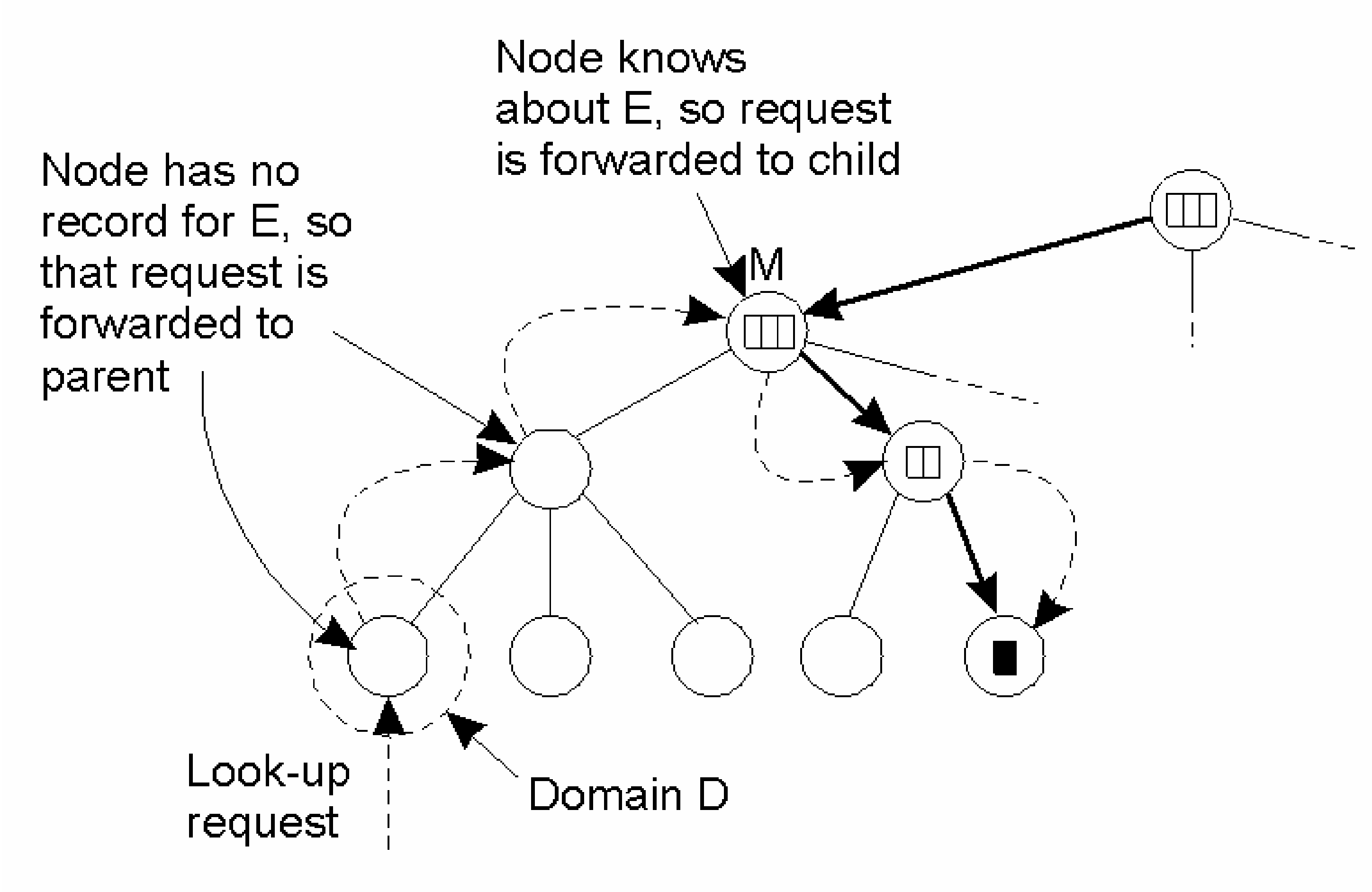
- Hierarchical organization of a location service into domains, each having an associated directory node.

Hierarchical Approaches (2)



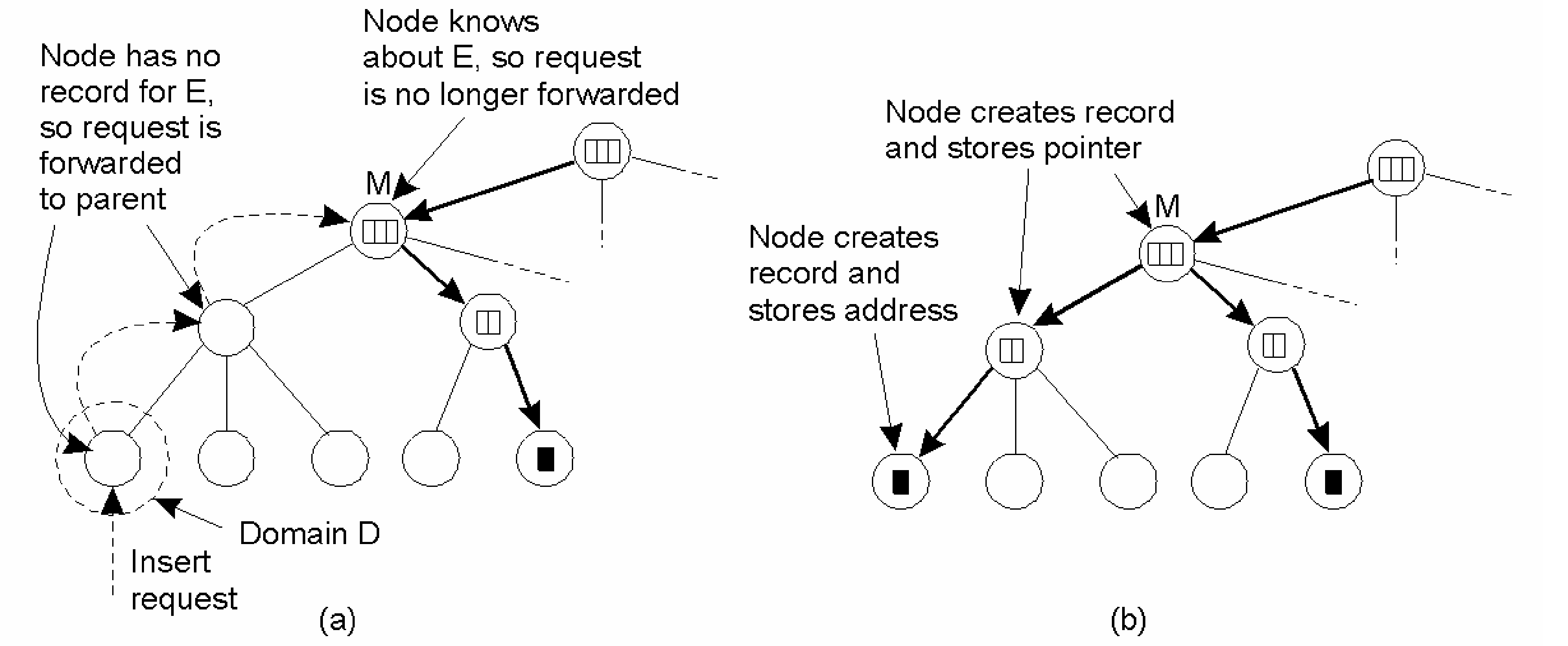
- An example of storing information of an entity having two addresses in different leaf domains.

Hierarchical Approaches (3)



- Looking up a location in a hierarchically organized location service

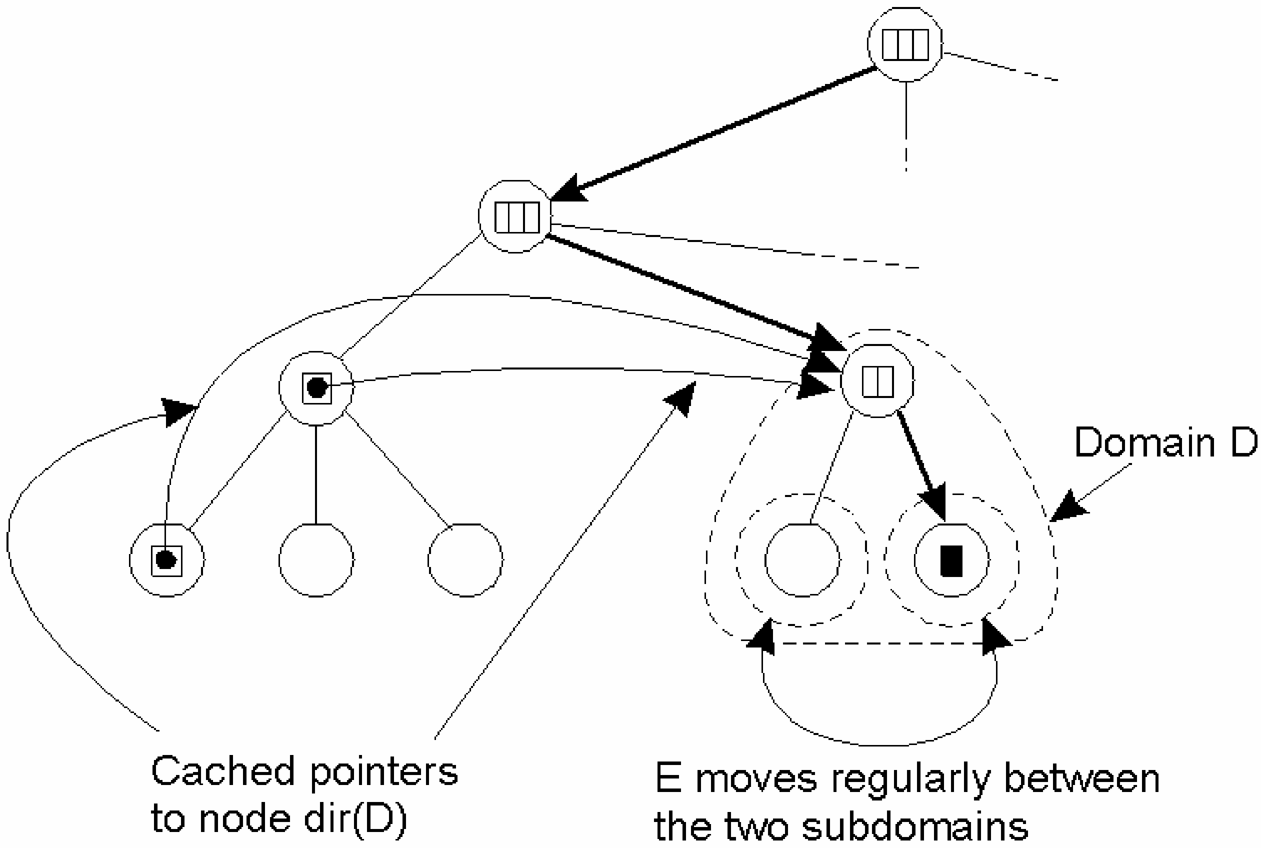
Hierarchical Approaches (4)



a) An insert request is forwarded to the first node that knows about entity *E*

b) A chain of forwarding pointers to the leaf node is created

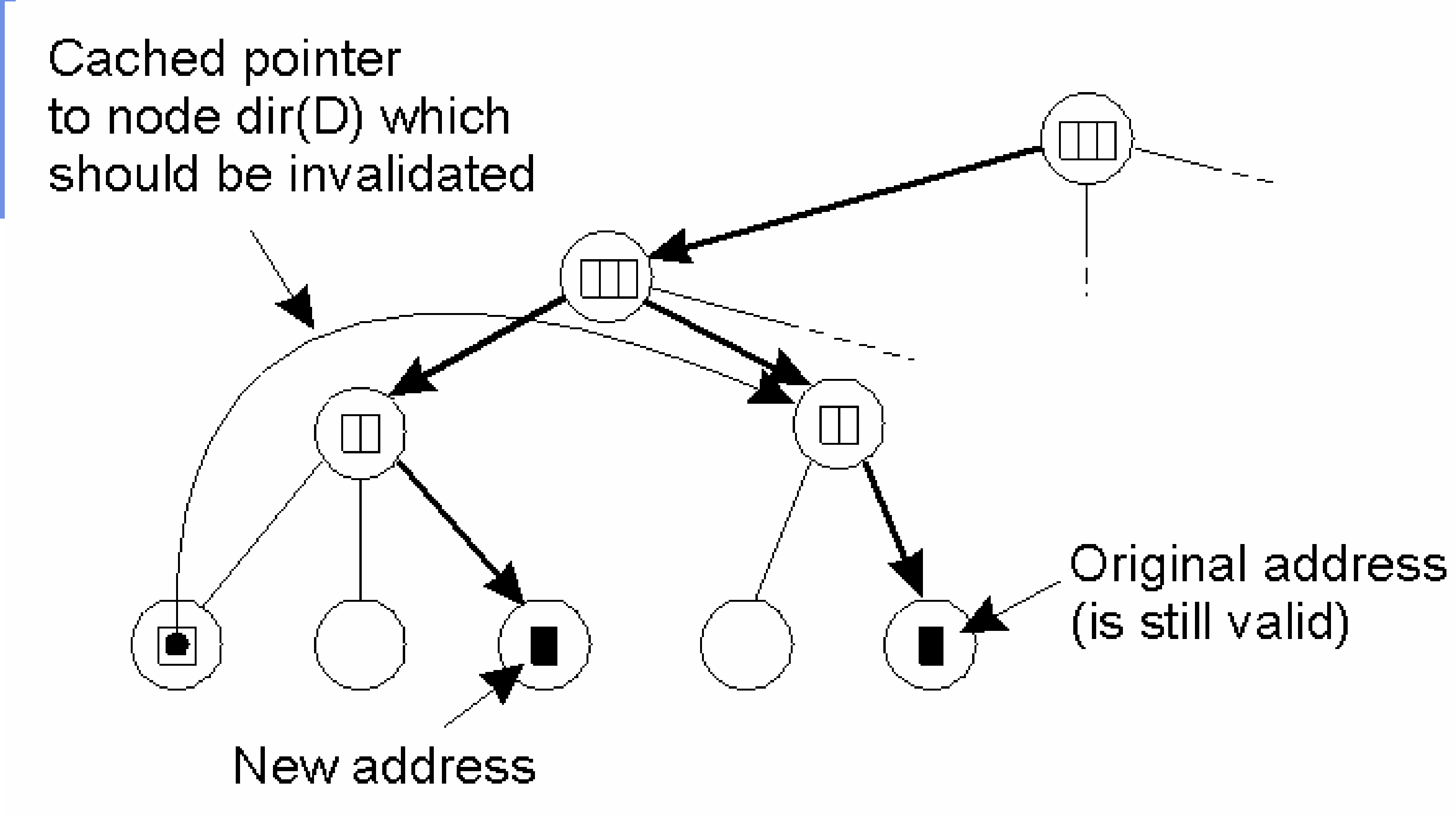
Pointer Caches (1)



- Caching a reference to a directory node of the lowest-level domain in which an entity will reside most of the time

© 2004 Universität Karlsruhe, System Architecture Group

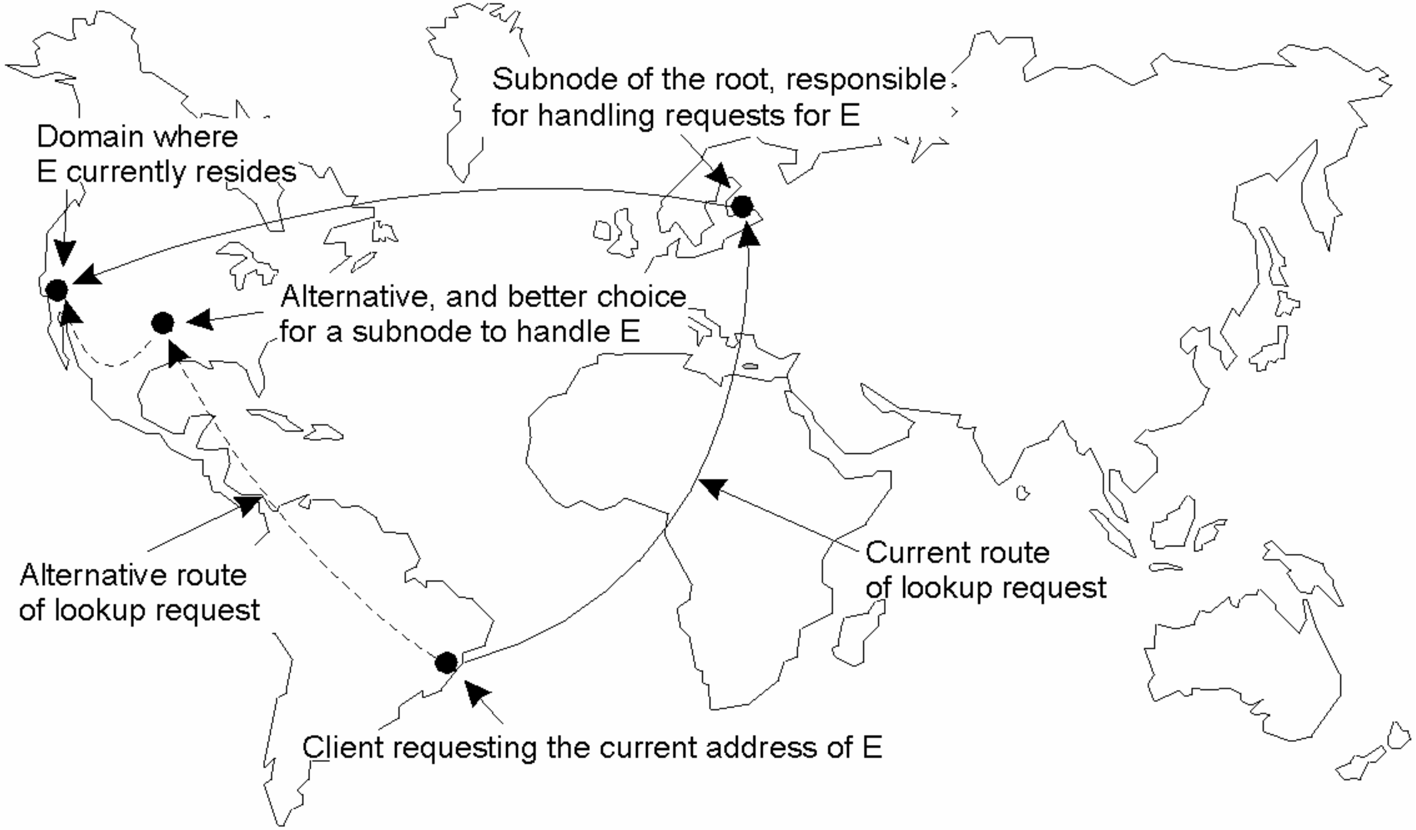
Pointer Caches (2)



- Cache entry that needs to be invalidated because it returns a nonlocal address, while such an address is available.

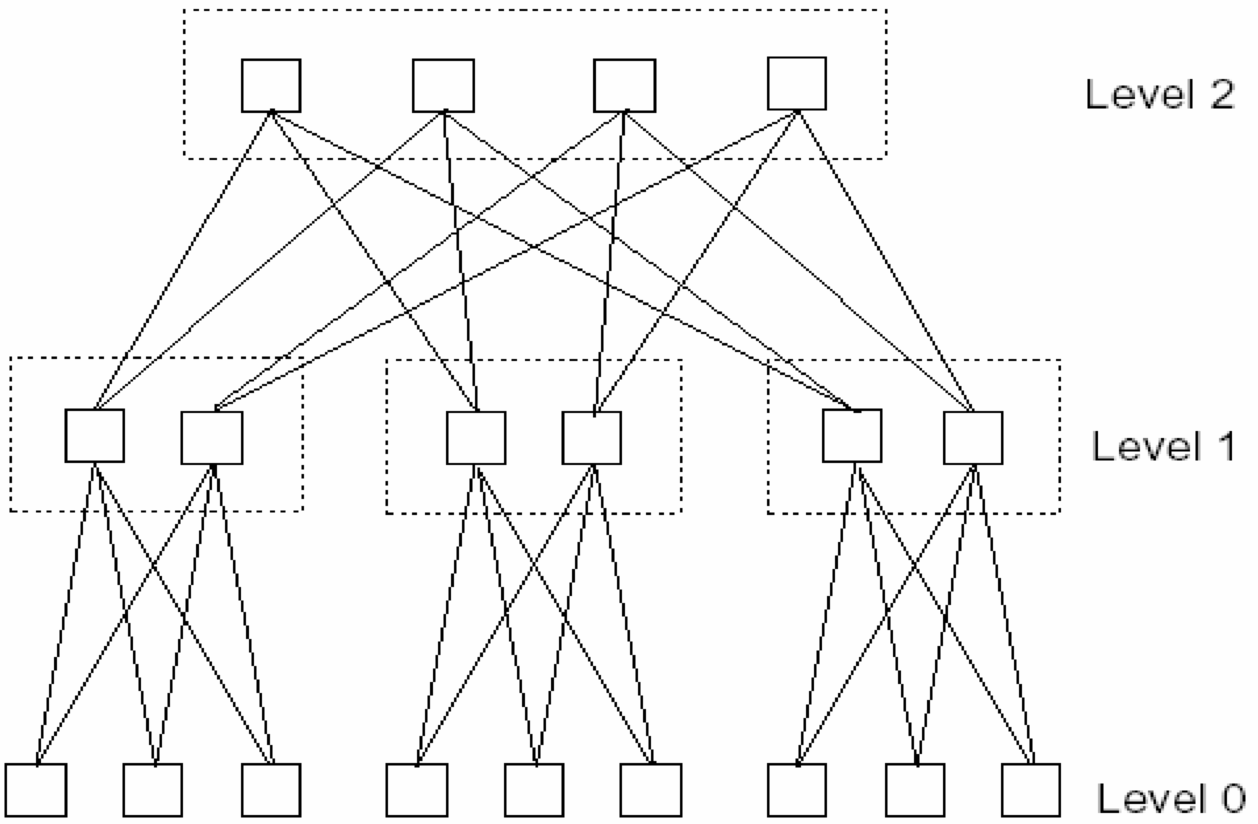
© 2004 Universität Karlsruhe, System Architecture Group

Scalability Issues



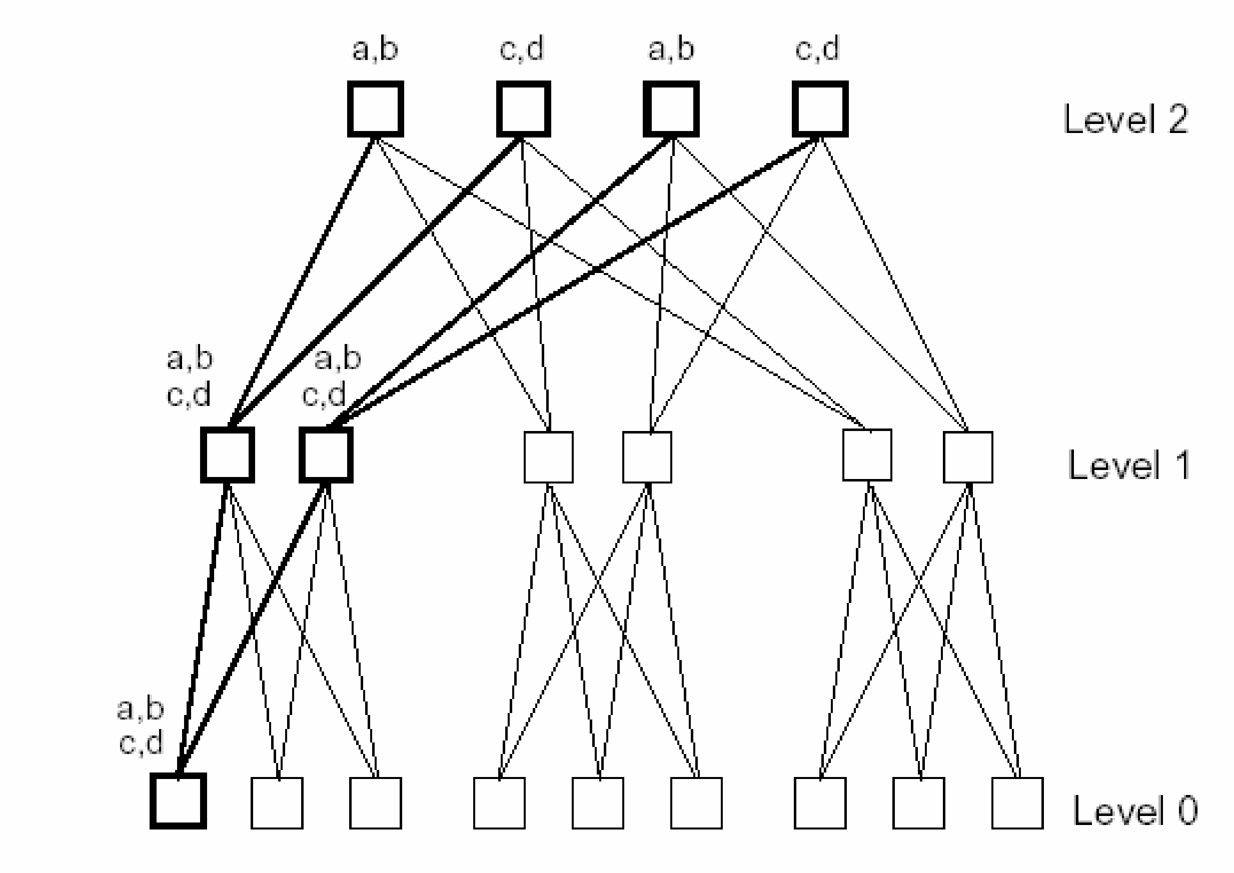
Scalability issues related to uniformly placing subnodes of a partitioned root node across the network covered by a location service.

Scalability Solution



Use of a *fat tree* to increase the number of servers at upper levels of the tree

Scalability Solution



- Replicating and partitioning binding information among the multiple parents in the fat tree

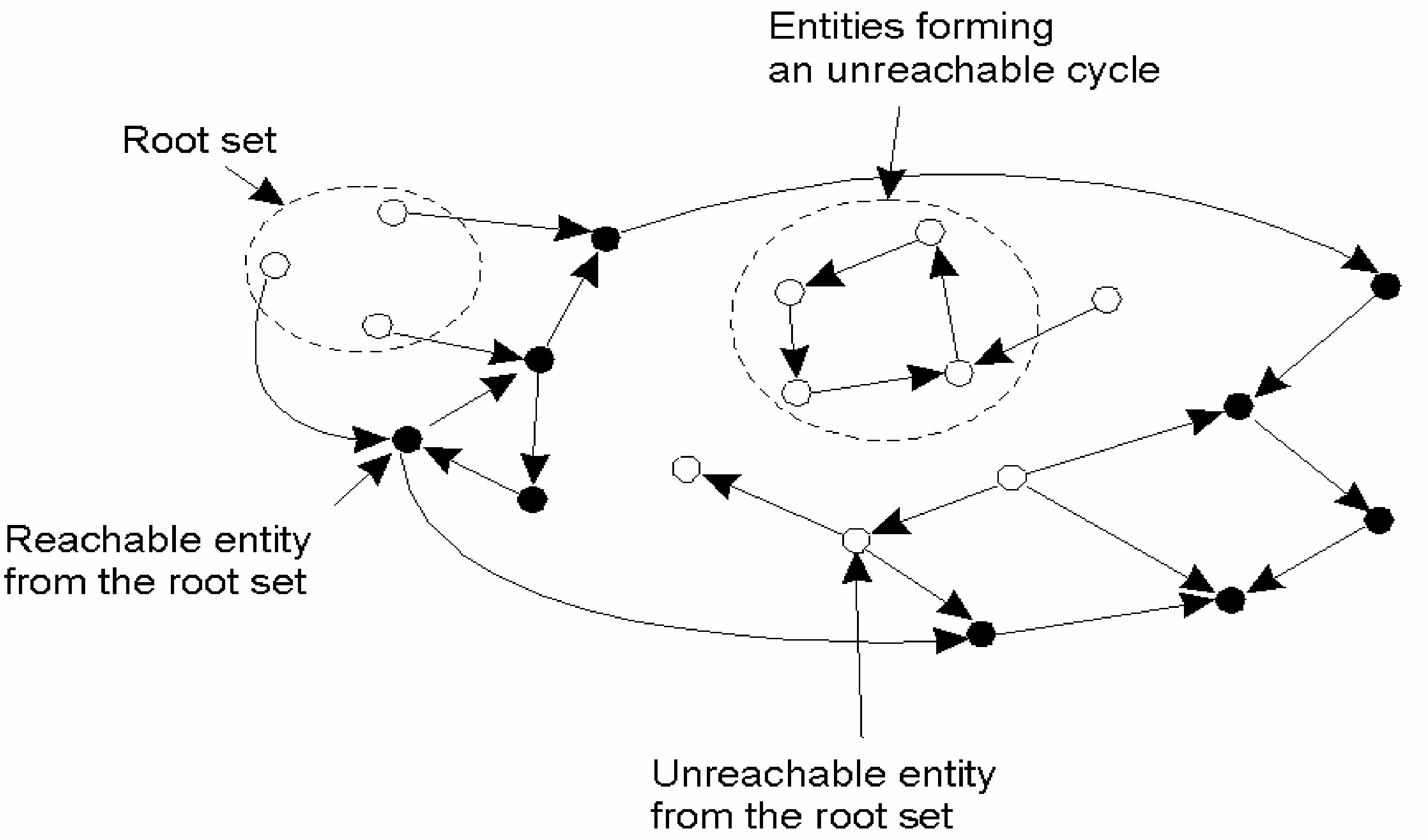
Deleting of Objects

- *How to manage, that an object can be deleted without any dangling reference?*
- However, we want to able to delete objects whether there might be any valid refernces to that object
- *What to do with objects without any current reference ?* (it's wasting resource, may be never reused again)

⇒ we need a distributed garbage collection

© 2004 Universität Karlsruhe, System Architecture Group

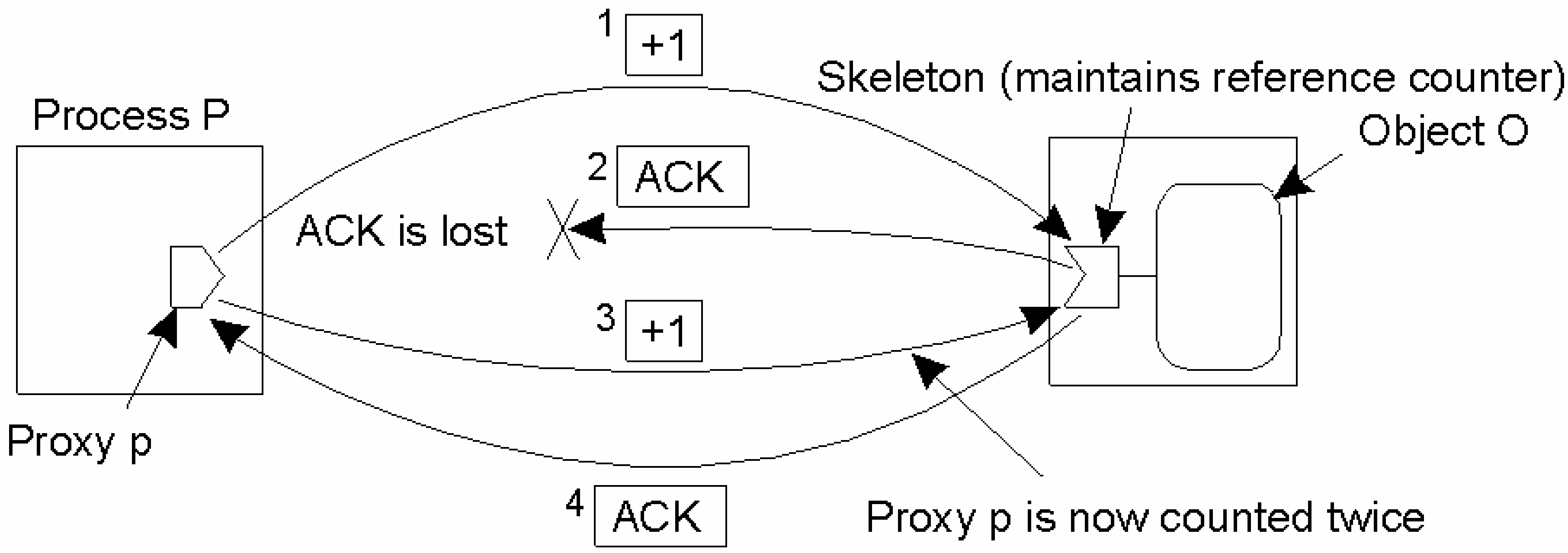
Problem of Unreferenced Objects



- An example of a graph representing objects containing references to each other.

© 2004 Universität Karlsruhe, System Architecture Group

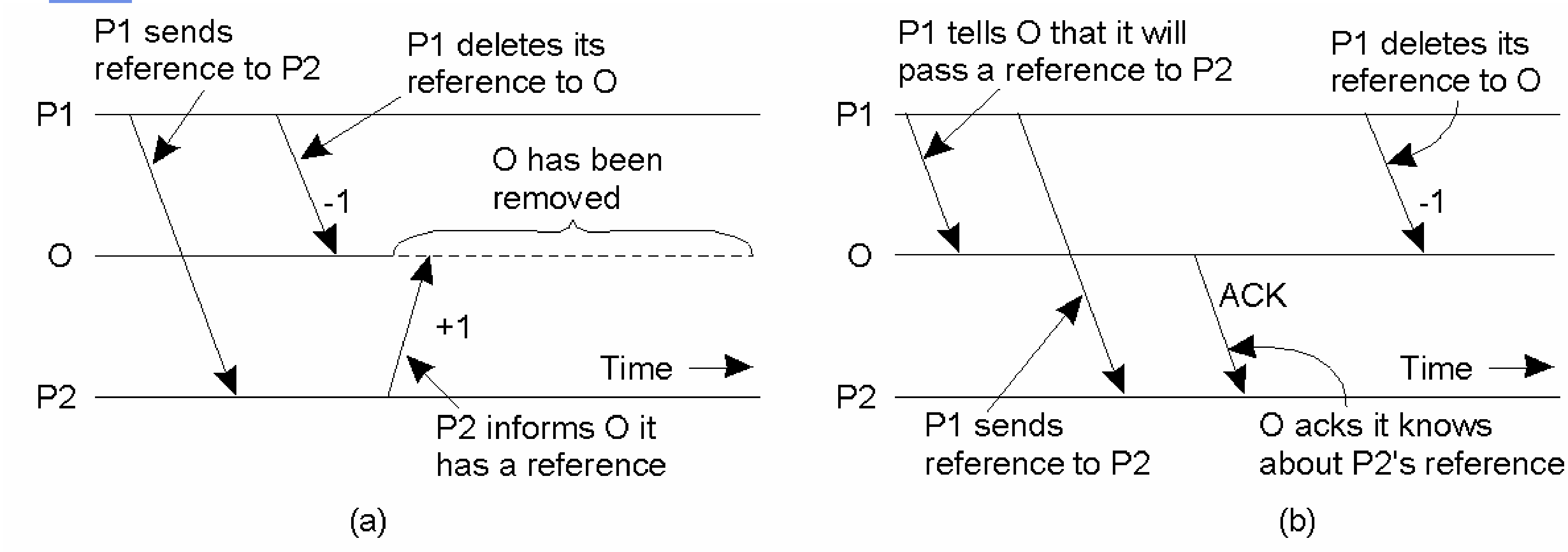
Reference Counting (1)



- The problem of maintaining a proper reference count in the presence of unreliable communication

© 2004 Universität Karlsruhe, System Architecture Group

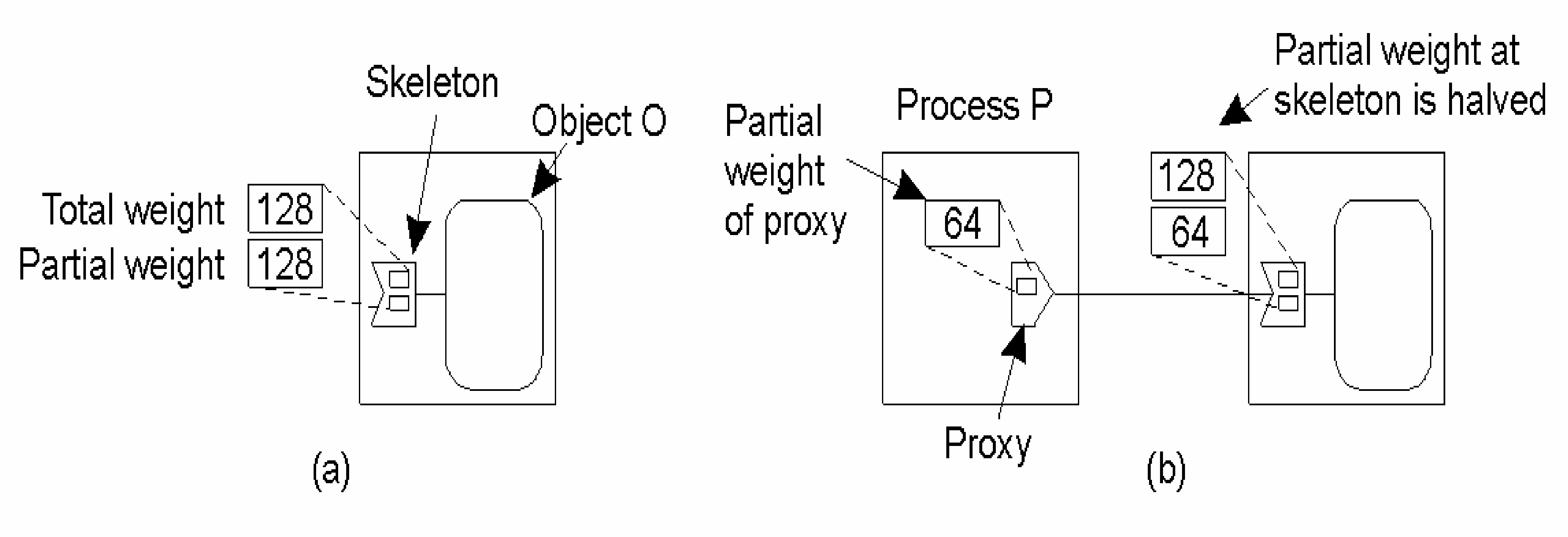
Reference Counting (2)



a) Copying a reference to another process and incrementing the counter too late

b) A solution.

Advanced Referencing Counting (1)

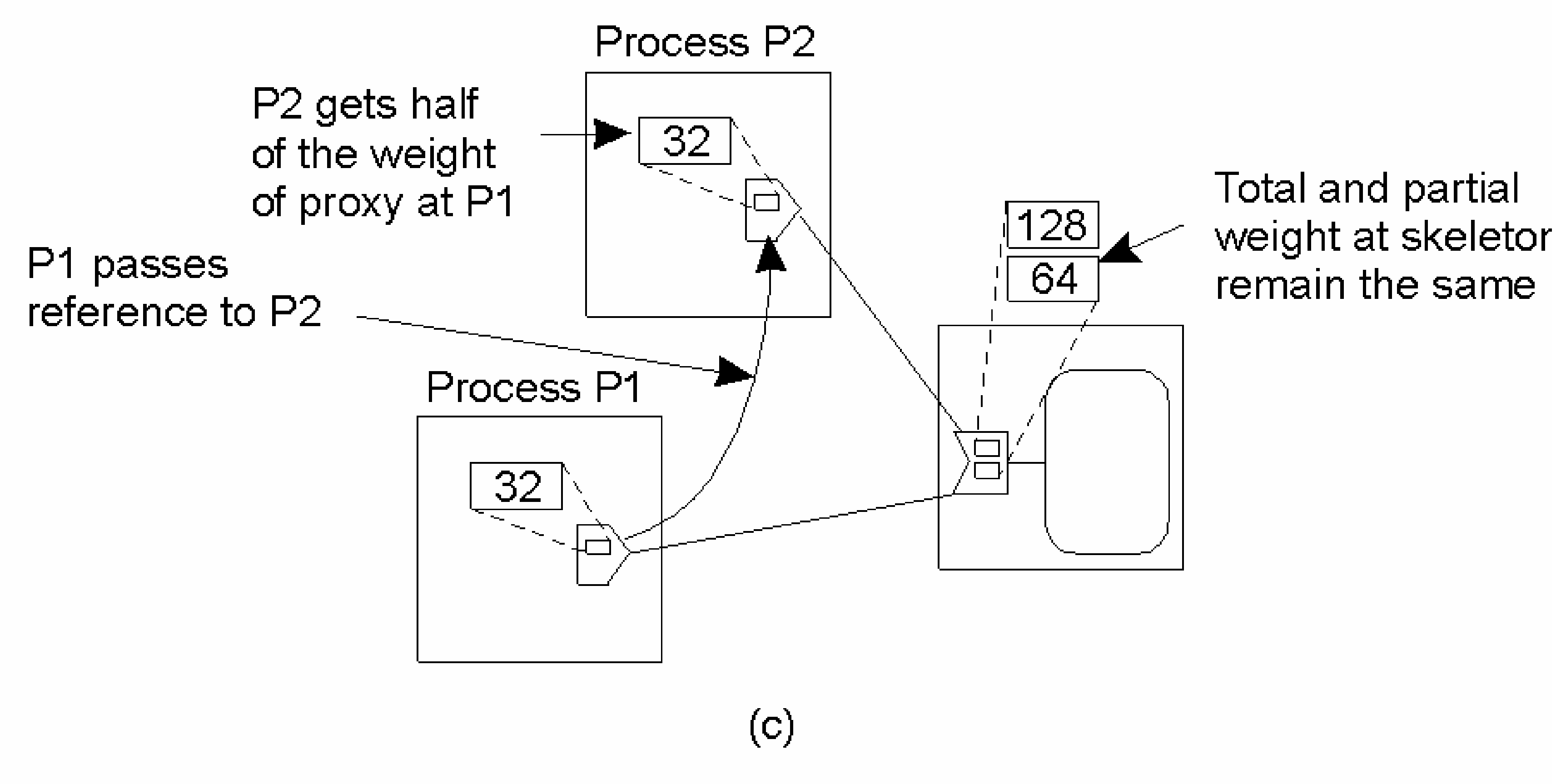


a) The initial assignment of weights in weighted reference counting

b) Weight assignment when creating a new reference.

© 2004 Universität Karlsruhe, System Architecture Group

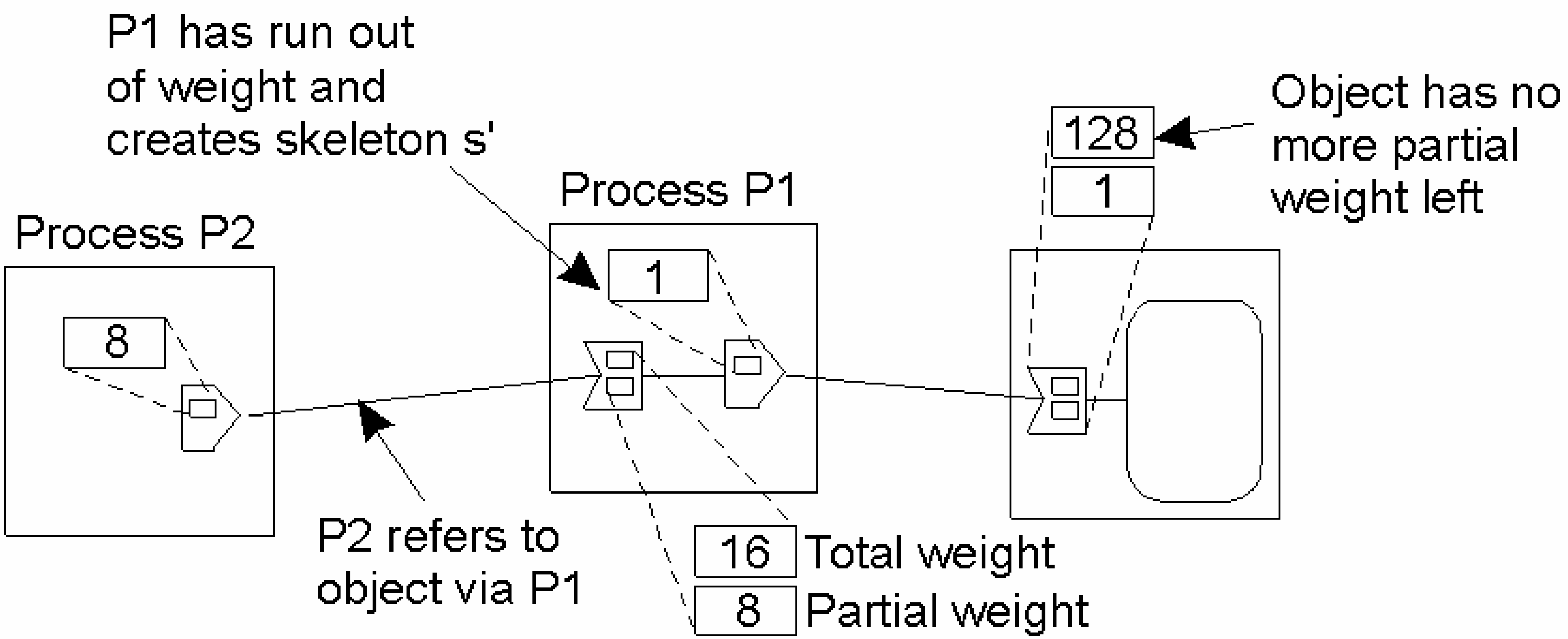
Advanced Referencing Counting (2)



c) Weight assignment when copying a reference.

© 2004 Universität Karlsruhe, System Architecture Group

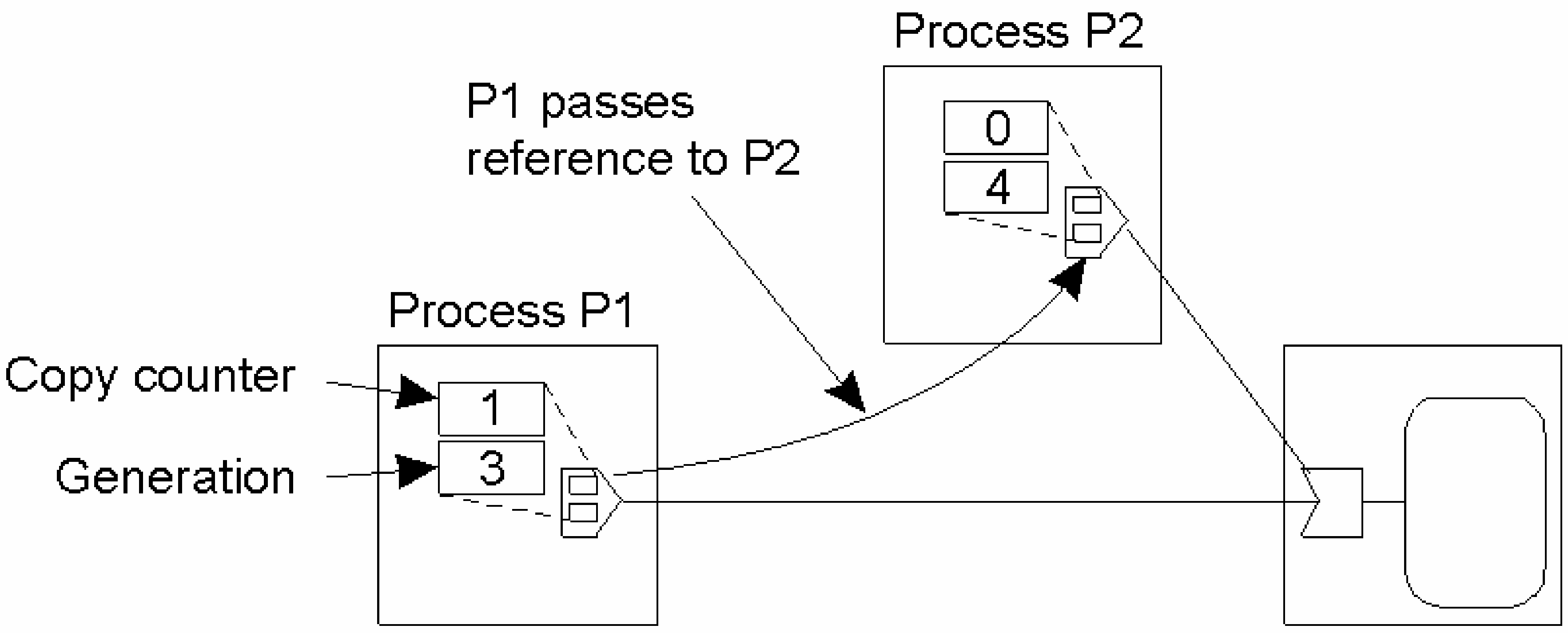
Advanced Referencing Counting (3)



- Creating an indirection when the partial weight of a reference has reached 1.

© 2004 Universität Karlsruhe, System Architecture Group

Advanced Referencing Counting (4)



- Creating and copying a remote reference in generation reference counting.

© 2004 Universität Karlsruhe, System Architecture Group

Generation Referencing Counting

Skeleton maintains *G*

- *G[k]* is #proxies in generation *k*

When a proxy is removed

- it sends its generation *k* and copy count *n* to the skeleton
- skeleton does
  - *G[k] -= 1*
  - *G[k+1] += n*

Remaining problems:

- Node failures, need reliable communication

© 2004 Universität Karlsruhe, System Architecture Group

Identifying Unreachable Objects

- *Mark-and-sweep tracing* (e.g., in Emerald)
- Local collectors mark objects reachable from a local root, or a marked object
- A marked proxy notifies its skeleton, thus marking it
- When all local collectors are done marking, collect unmarked objects

Problem:

- Requires execution to stop during marking
- Incremental approaches may cause excessive communication

Reference Lists

- Adding and deleting proxies from a reference list in the skeleton of the object server are idem potent operations $\Rightarrow$

  do not require *reliable communication*
- However, we will use acks to be sure that adding or deleting have been done
- Skeleton is able to control the consistency of its reference list pinging to all proxies
- Scalability is low

© 2004 Universität Karlsruhe, System Architecture Group

Hierarchical Tracing

- Hierarchical approach to GC
- Nodes are recursively partitioned into groups
- Each group does internal GC
- References among groups are GC'ed in next higher group

Hierarchical Tracing

Skeleton marks:

- Soft: reachable only from proxies inside group
- Hard: reachable from proxy outside group, or reachable from a root inside the group

Proxy marks:

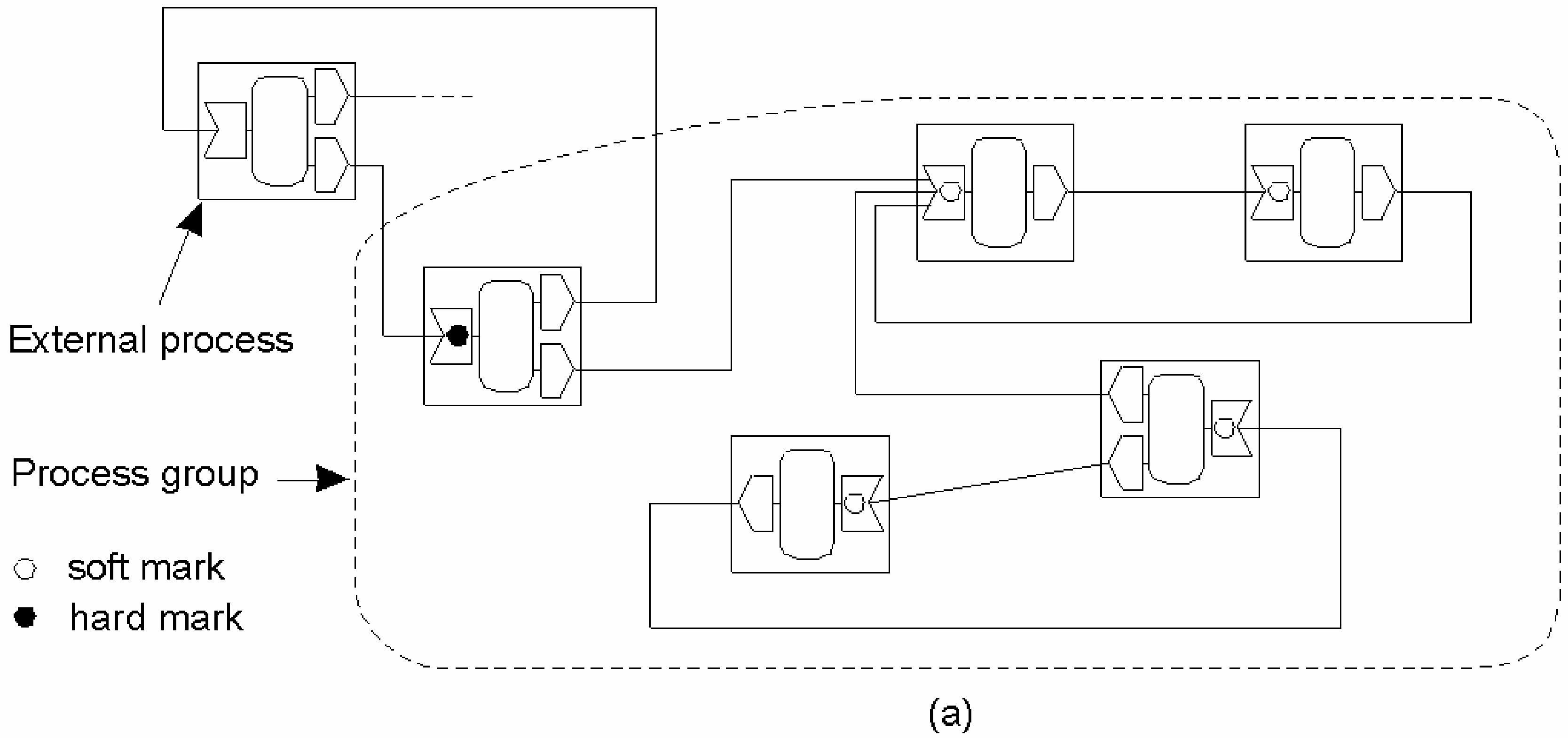
- None: unreachable
- Soft: reachable from a skeleton marked soft
- Hard: reachable from a root

Hierarchical Tracing

GC within a group proceeds as follows:

1. Initial marking of skeletons (soft or hard)
2. Intraprocess propagation of marks from skeletons to proxies (local GC)
3. Interprocess, intragroup propagation of hard marks from proxies to skeletons
4. Stabilization (iterate steps 2 and 3)
5. Garbage reclamation (e.g., mark soft skeletons as garbage, run local GC)

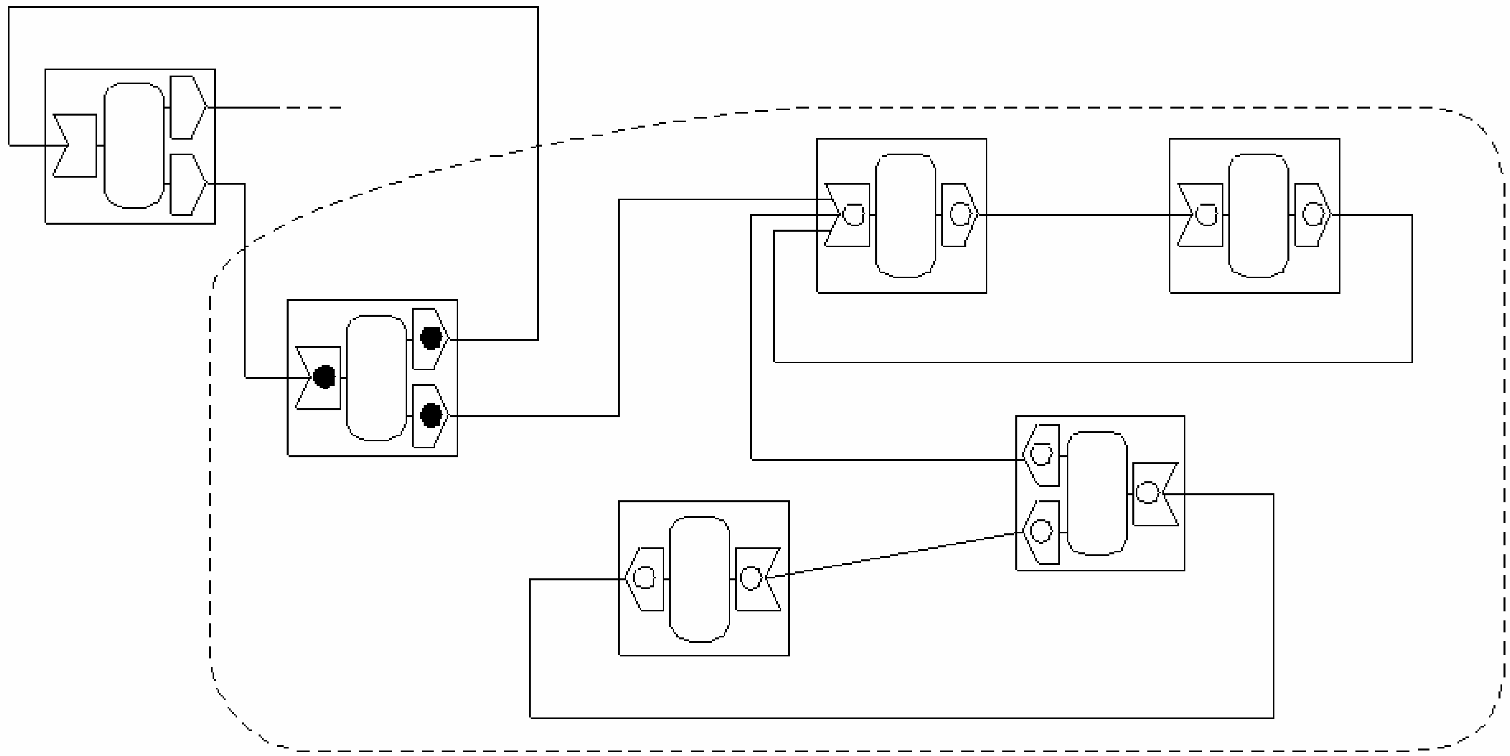
Tracing in Groups (1)



- Initial marking of skeletons

© 2004 Universität Karlsruhe, System Architecture Group

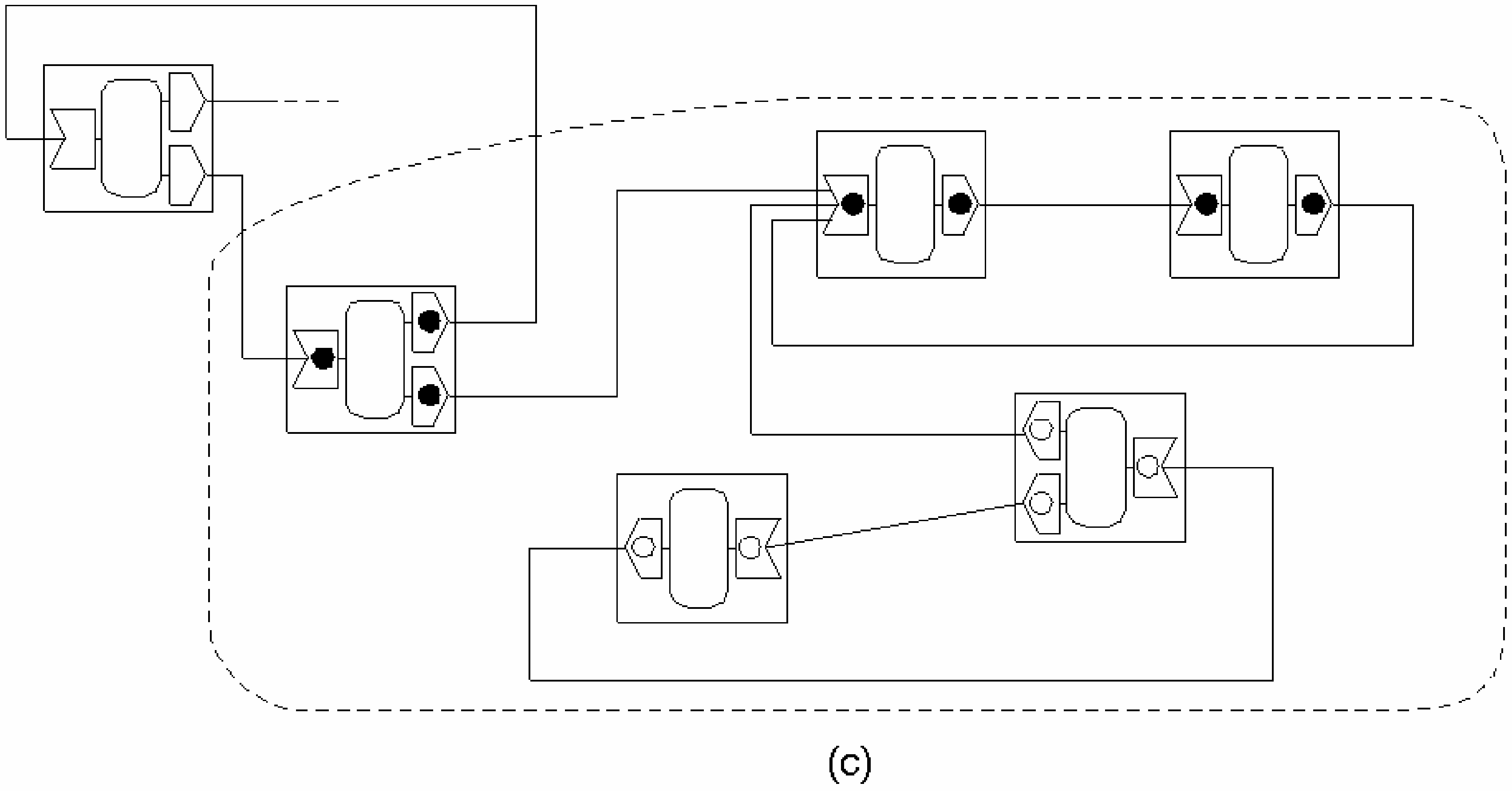
Tracing in Groups (2)



(b)

- After local propagation in each process

Tracing in Groups (3)



- Final marking