Programming of Distributed Systems

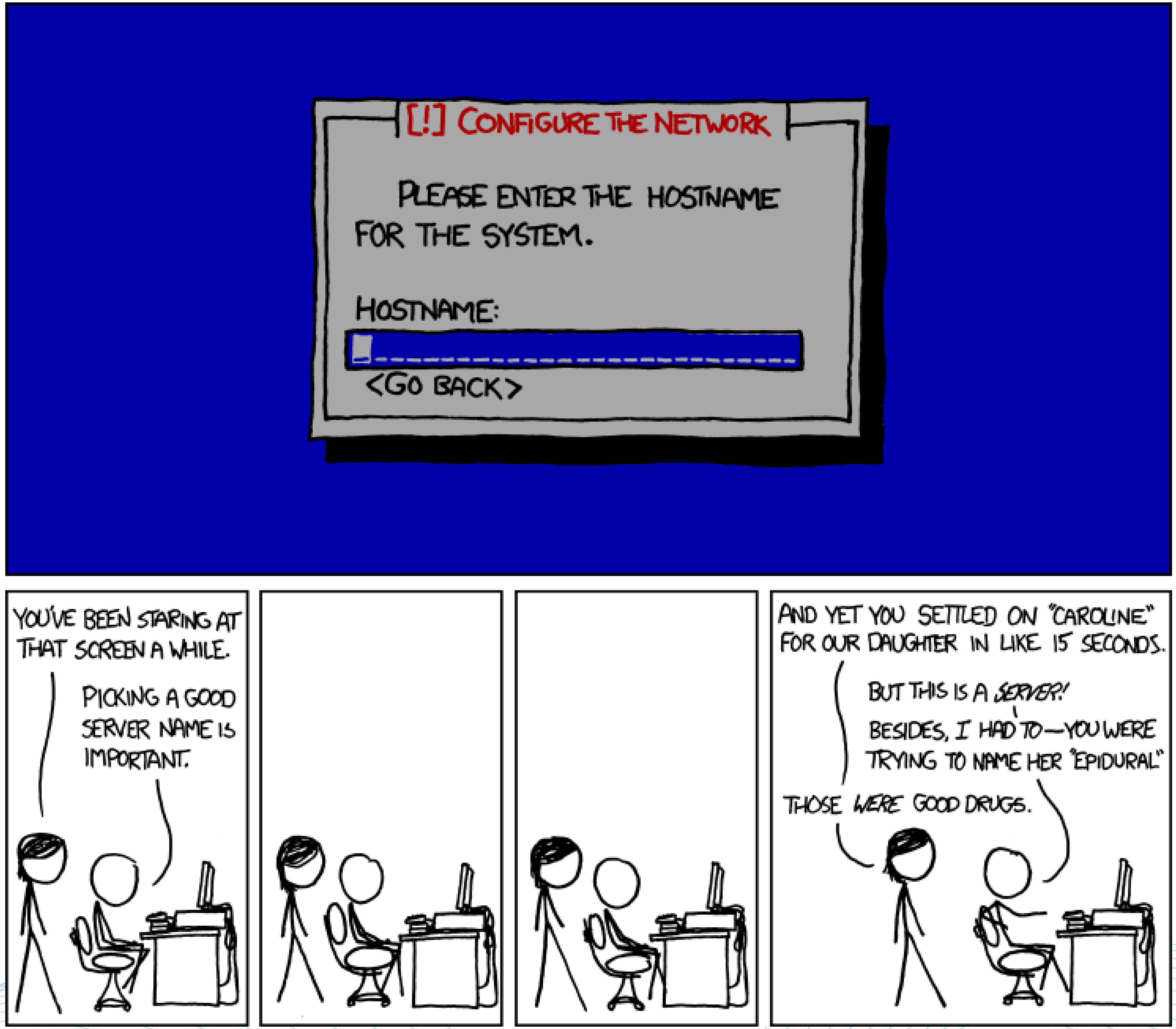
Topic III – Naming

Dr.-Ing. Dipl.-Inf. Erik Schaffernicht

Reading Remarks

Reading Task:
Chapter 5
DHTs / CHORD example in 5.2 and decentralized implementations in 5.4 is nice to know, but not examination relevant.

Pay attention to the DNS (repetition) and NFS examples in 5.3.



Caption Text:
This hostname is going in dozens of remote config files. Changing a kid's name is comparatively easy!

https://xkcd.com/910/

Names

> A **name** in a distributed system is a string of bits or characters that is used to refer to an **entity**.

From Steen, Tanenbaum. Distributed Systems (2017), p.238

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.

> A **location-independent** name for an entity, is independent from the addresses of the access points offer by that entity.

From Steen, Tanenbaum. Distributed Systems (2017), p.238

Identifiers & Naming Systems

1. An **identifier** is a name that 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.

From Steen, Tanenbaum. Distributed Systems (2017), p.239

Central question:

How to resolve names and identifiers into addresses?

Name-to-address binding → table of `<name, address>` pairs

Three Naming Systems

1. Flat naming
2. Structured naming
3. Attribute-based naming

Flat naming

→ Name is a random string and does not contain any information on how to locate the process!

First idea: Broadcasting

Example: Address Resolution Protocol

Broadcast: Who has the IP address 130.243.103.44?

Answer with Ethernet address, if the node is listening.

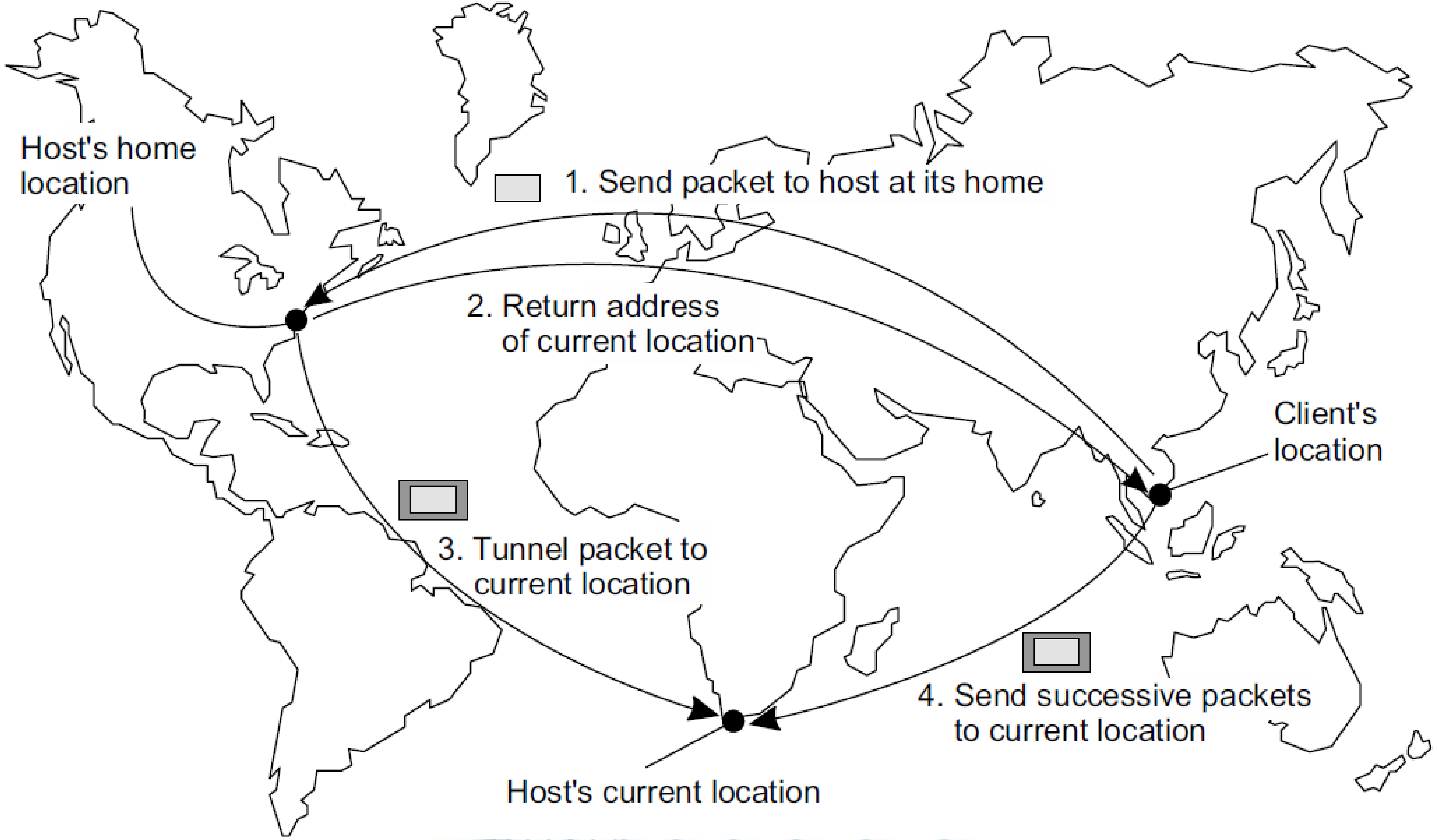
Home-based approaches

Idea:

Home keeps track of where the entity is

Entity's home address registered at a naming service

Home registers the foreign c/o address of the entity



Forwarding pointers

Idea

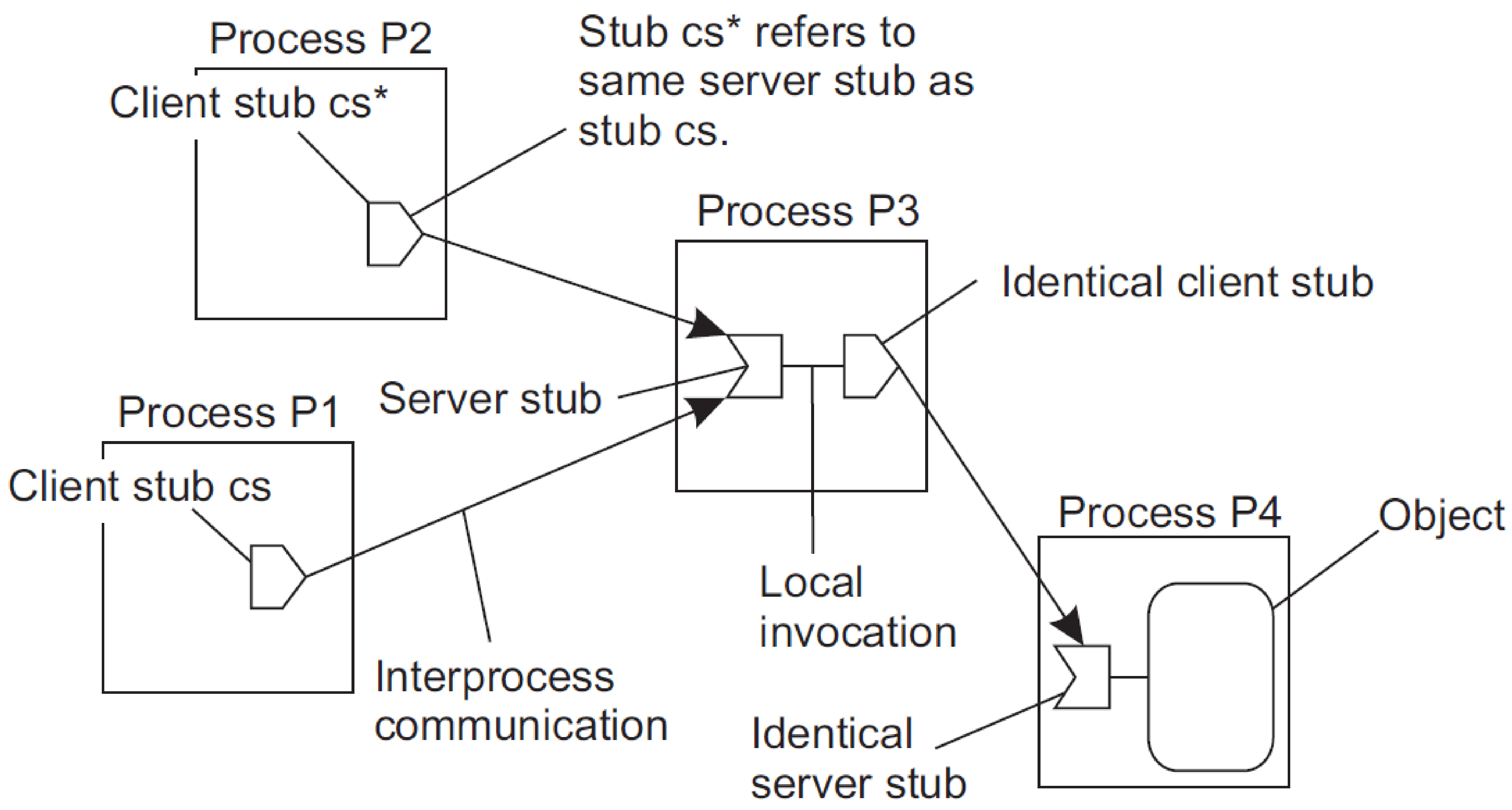
When an entity moves, it leaves behind a pointer to its next location and update a client's reference when present location is found

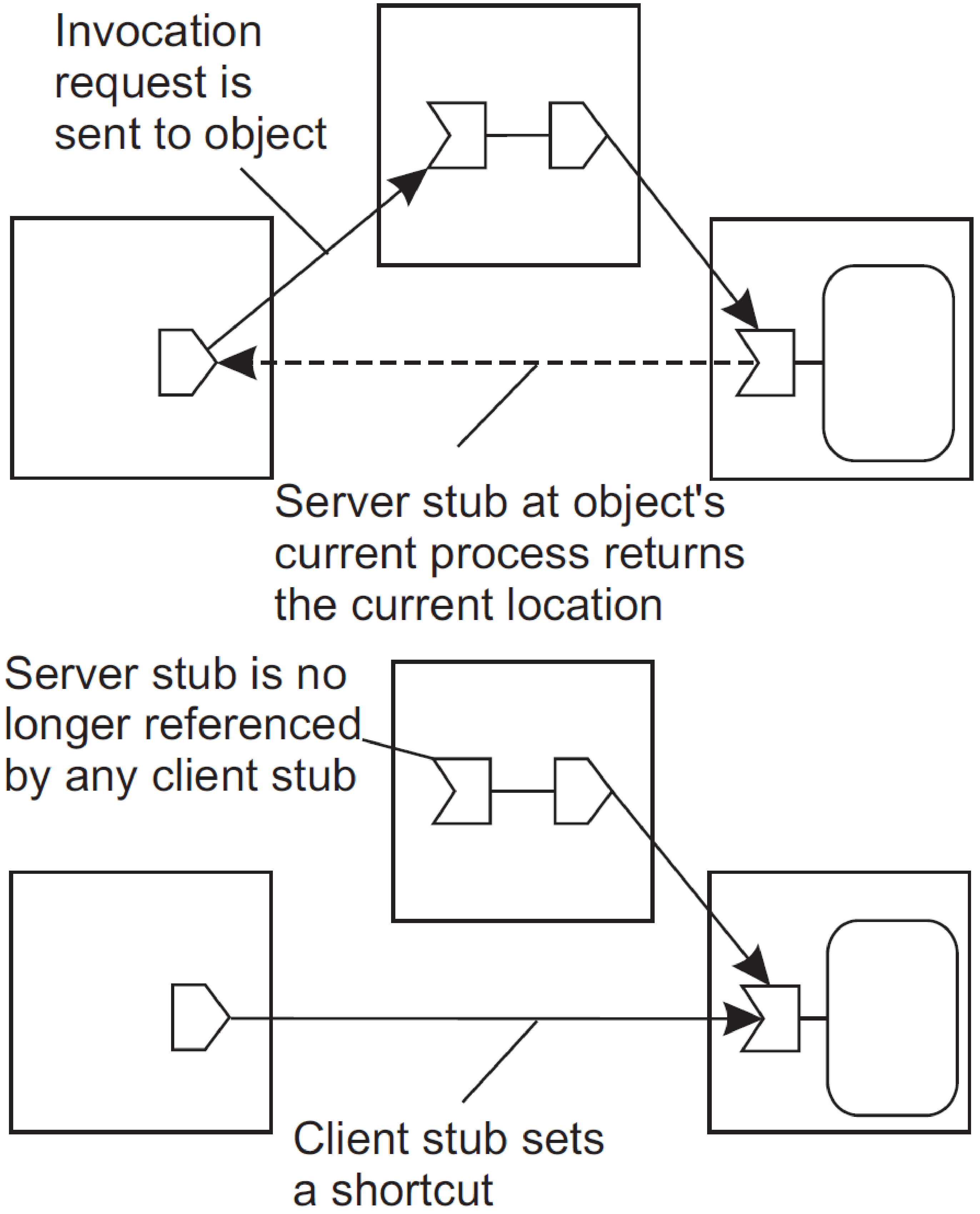
Geographical scalability problems

- Long chains are not fault tolerant
- Increased network latency at dereferencing

ÖREBRO UNIVERSITY

SSP chains - Forwarding using stubs

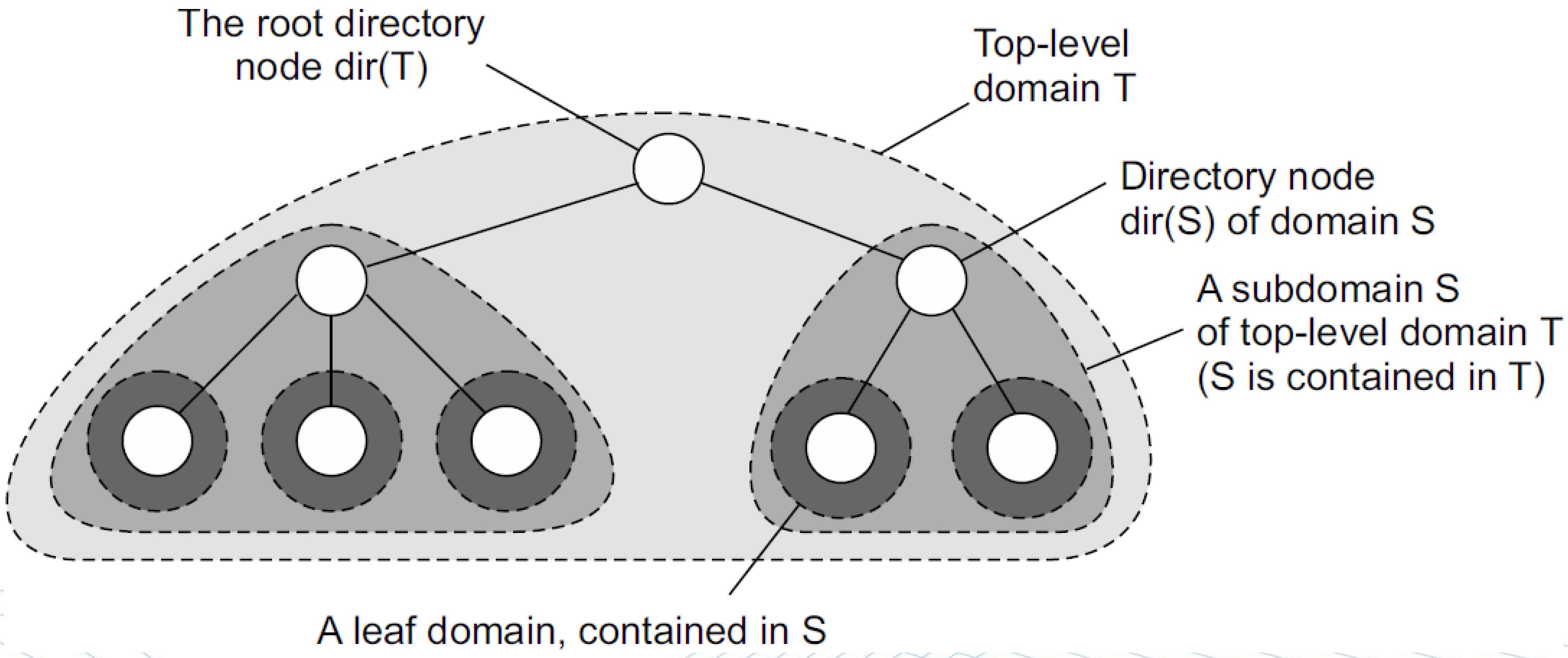




Hierarchical Location Services

Idea

Large-scale search tree dividing the network into hierachical domains



Hierarchical Location Services

Concepts

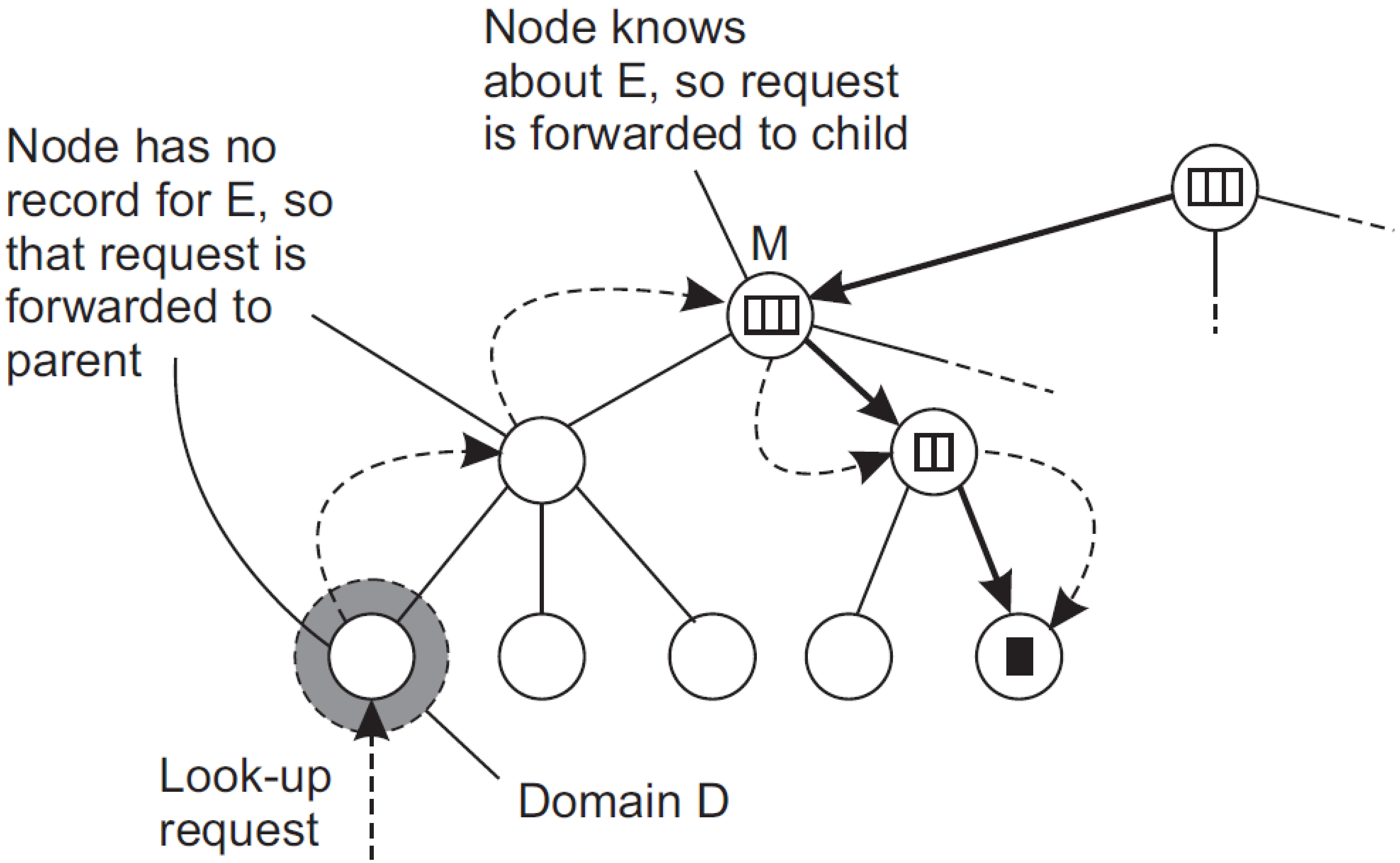
- Address of entity E is stored in a leaf or intermediate node
- Intermediate nodes contain a pointer to a child if and only if the subtree rooted at the child stores an address of the entity
- The root knows about all entities

Reading task:
Check chapter 5.2 for how to build such a tree.

Hierarchical Location Services

Lookup operation

- Start lookup at local leaf node
- Node knows about E ➔ follow downward pointer, else go up
- Upward lookup always stops at root



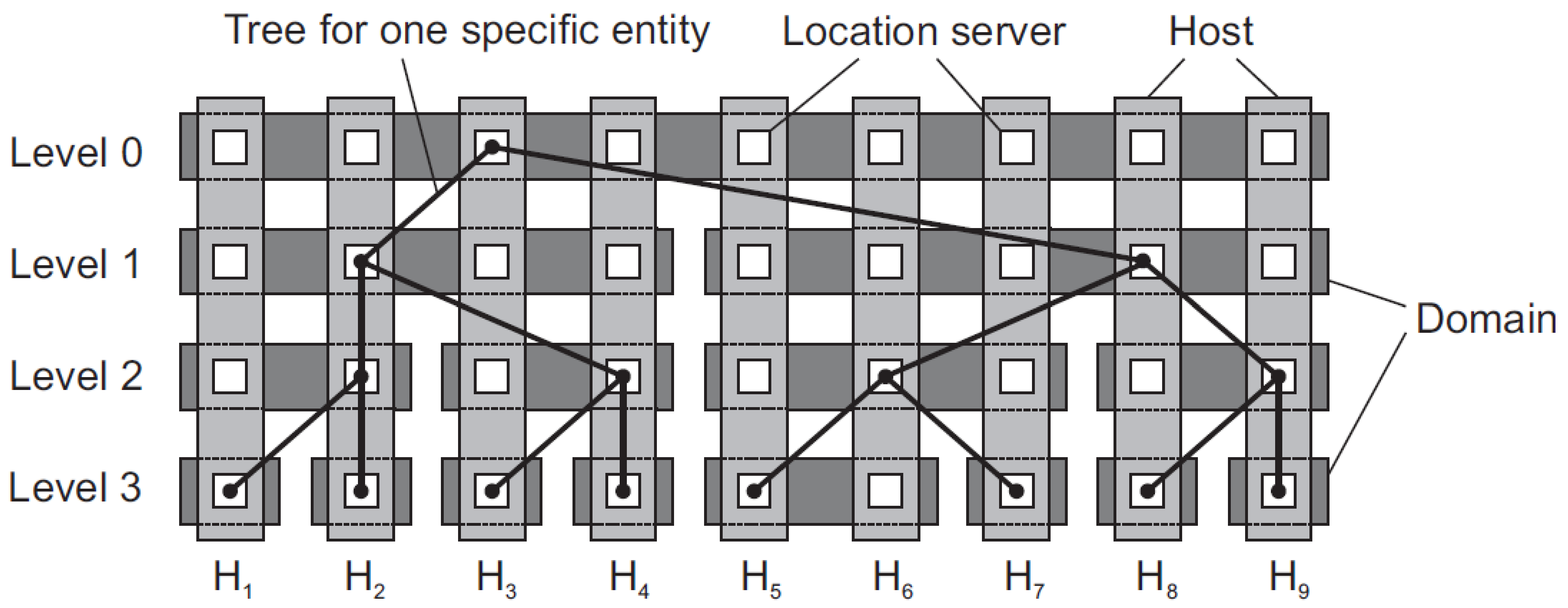
Hierarchical Location Services

Size Scalabilty

Problem: root node needs to keep track of all identifiers

Physical implementations of HLS are very different from their logical design

➔ mapping logical servers to physical ones



Overlay networks

Overlay networks are virtual **networks** of nodes and logical links, which are built on top of an existing **network** using the underlying infrastructure.

Examples:

- Telephone ←→ Internet
- Resilient Overlay Network
- Virtual Private Networks / Darknet

The beginning of Skype – A Peer-to-Peer overlay network

Three entities:

1) Login-server

- User authenification & guarantees Username uniqueness

- Only central component of the system
- Online/Offline user information is stored decentralized

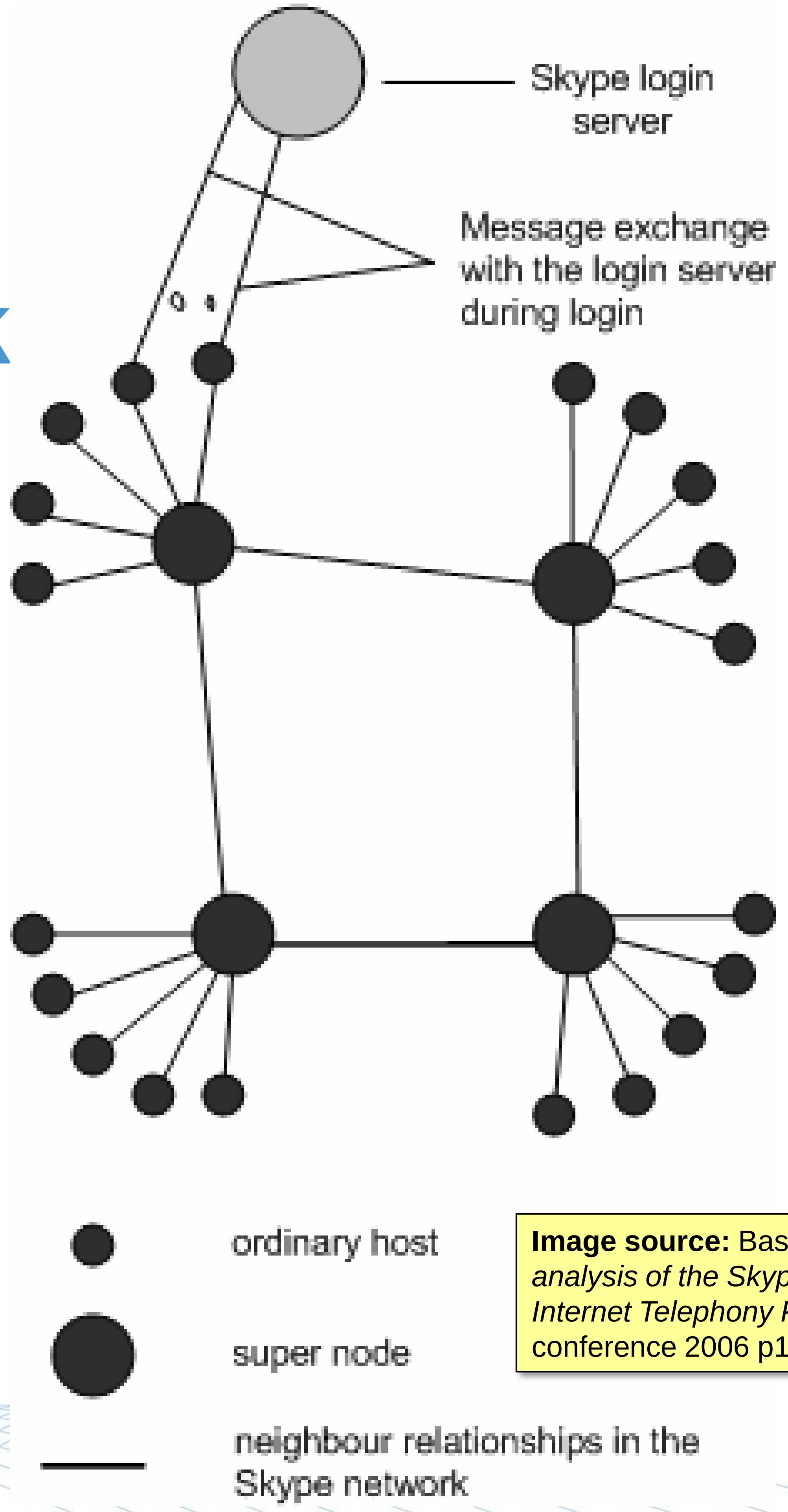


Image source: Baset & Schulzrinne.*An analysis of the Skype Peer-to-Peer Internet Telephony Protocol*. 25th INFOCOM conference 2006 p1-11

ÖREBRO UNIVERSITY

The beginning of Skype –
A Peer-to-Peer overlay network

Three entities:

2) Ordinary host

- end-user / Skype application

3) Super node

- Any ordinary host with a public IP and enough resources (CPU, memory, bandwidth) can become a super node

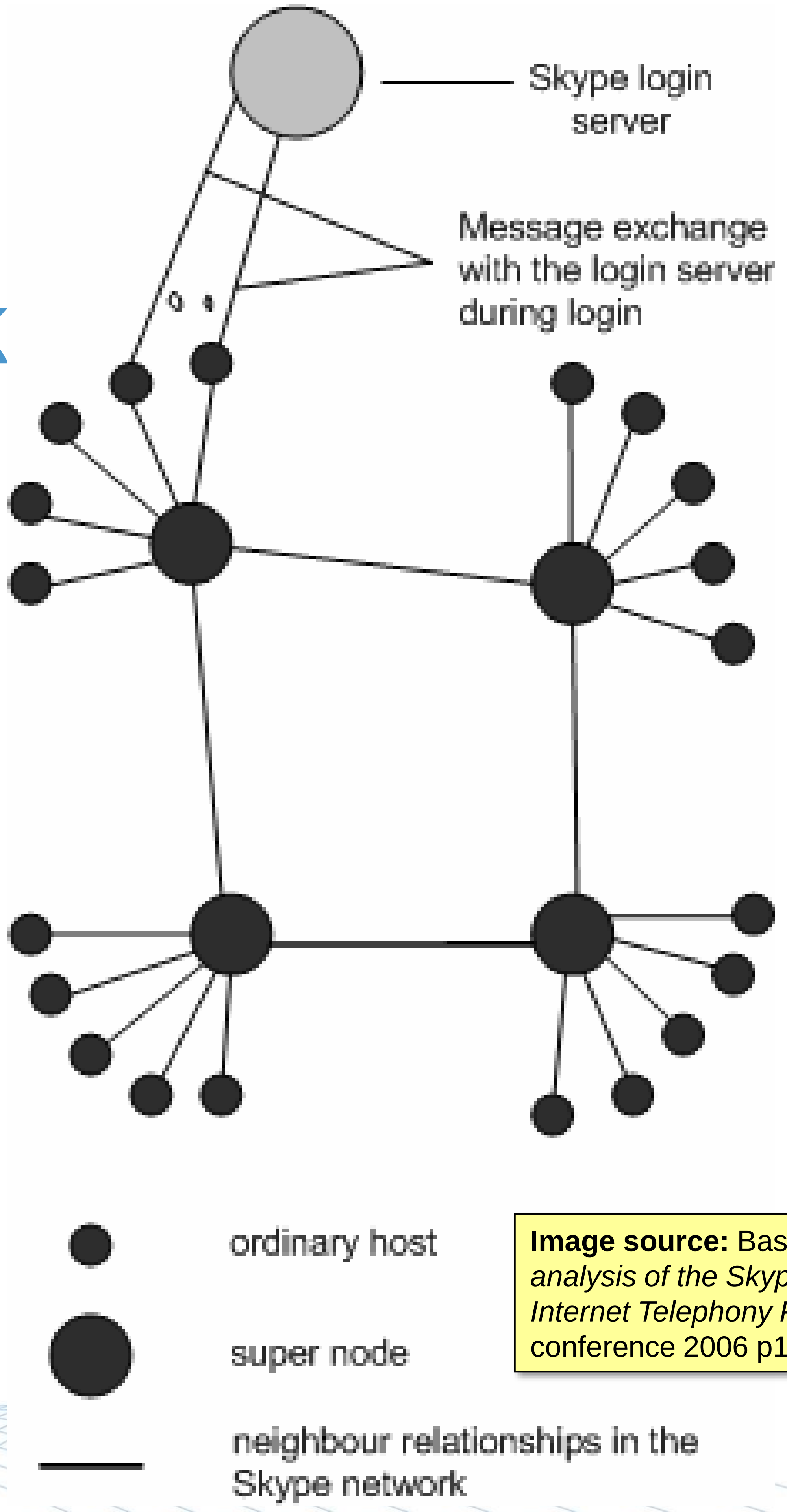


Image source: Baset & Schulzrinne.*An analysis of the Skype Peer-to-Peer Internet Telephony Protocol*. 25th INFOCOM conference 2006 p1-11

The beginning of Skype – A Peer-to-Peer overlay network

Login process:

1) Connect via TCP to a super node

- each client builds and maintains a host cache of reachable super nodes
- client stays connected to one super node all the time, if the super node fails it connects to another super node
- get address of the login server

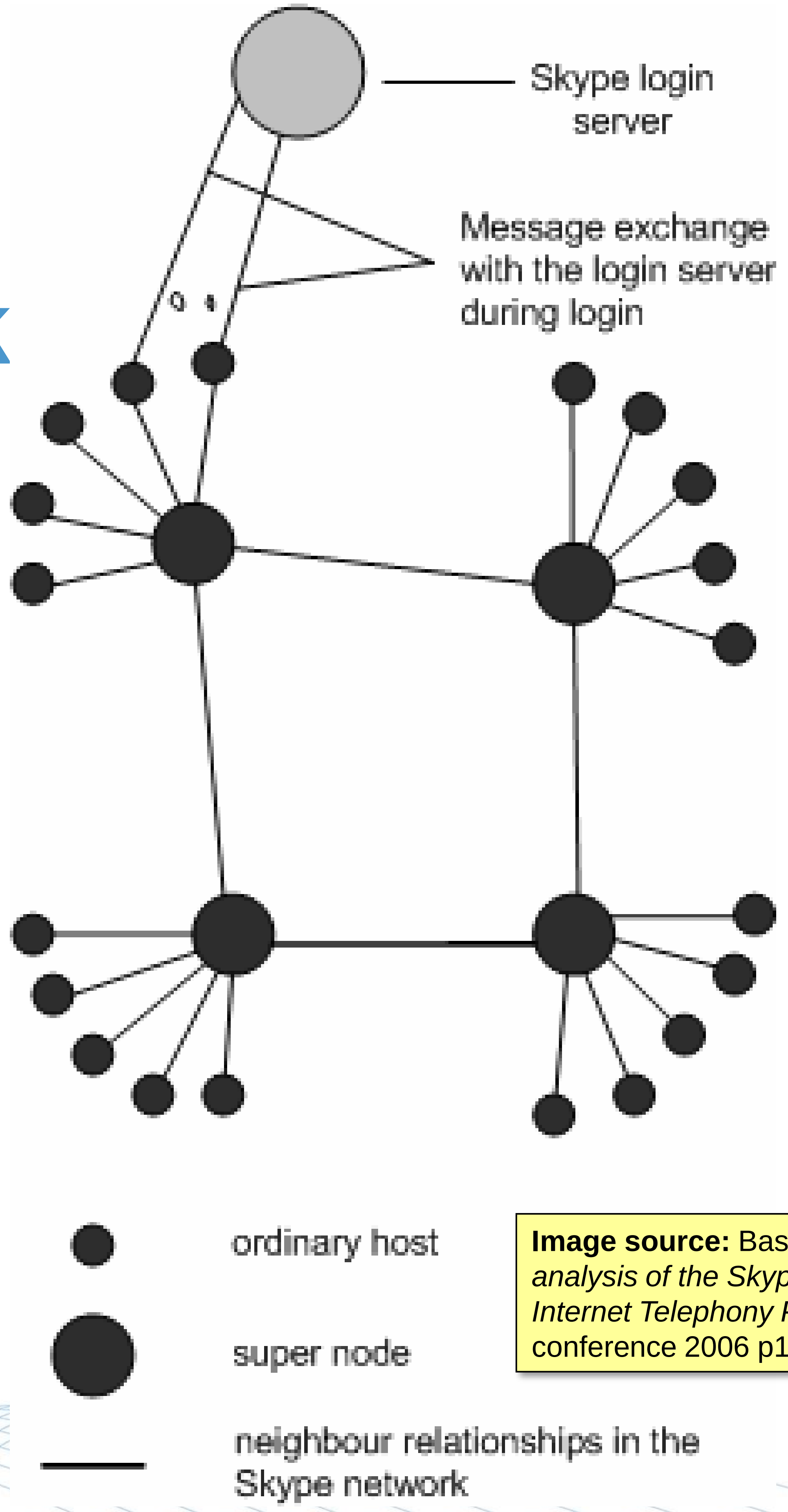


Image source: Baset & Schulzrinne.*An analysis of the Skype Peer-to-Peer Internet Telephony Protocol*. 25th INFOCOM conference 2006 p1-11

ÖREBRO UNIVERSITY

The beginning of Skype – A Peer-to-Peer overlay network

Login process:

2) Connect via TCP to the login server, authenticate the user and close the connection again

Task: If you need a refresher on TCP/UDP, the OSI model and other topics from Datorkommunikation, you can start with Chapter 4.1 of the course book!

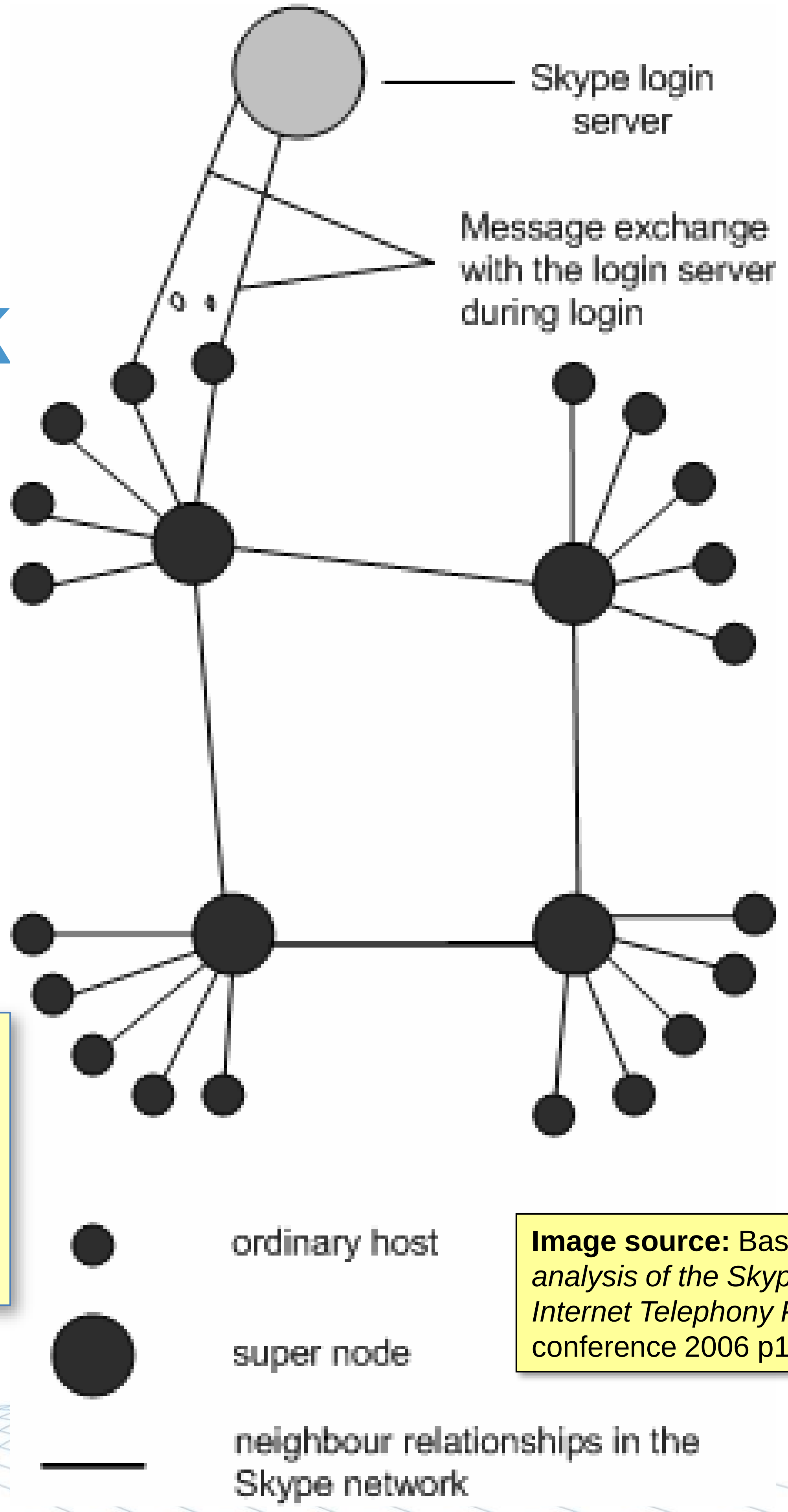


Image source: Baset & Schulzrinne.*An analysis of the Skype Peer-to-Peer Internet Telephony Protocol*. 25th INFOCOM conference 2006 p1-11

The beginning of Skype –
A Peer-to-Peer overlay network

Login process:

3) Client determines Firewall/NAT status and advertises online status to the super node

➔ Login complete

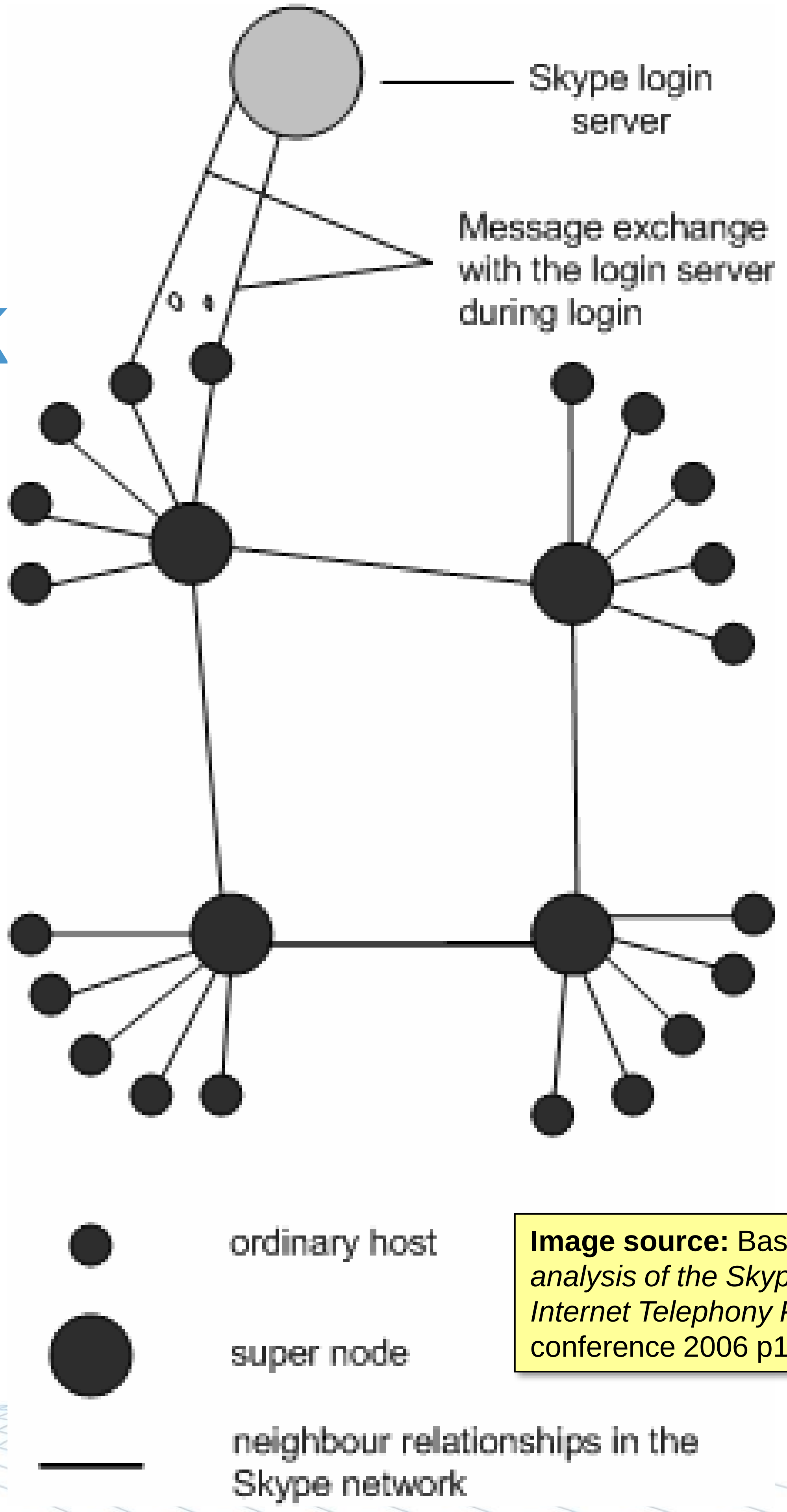
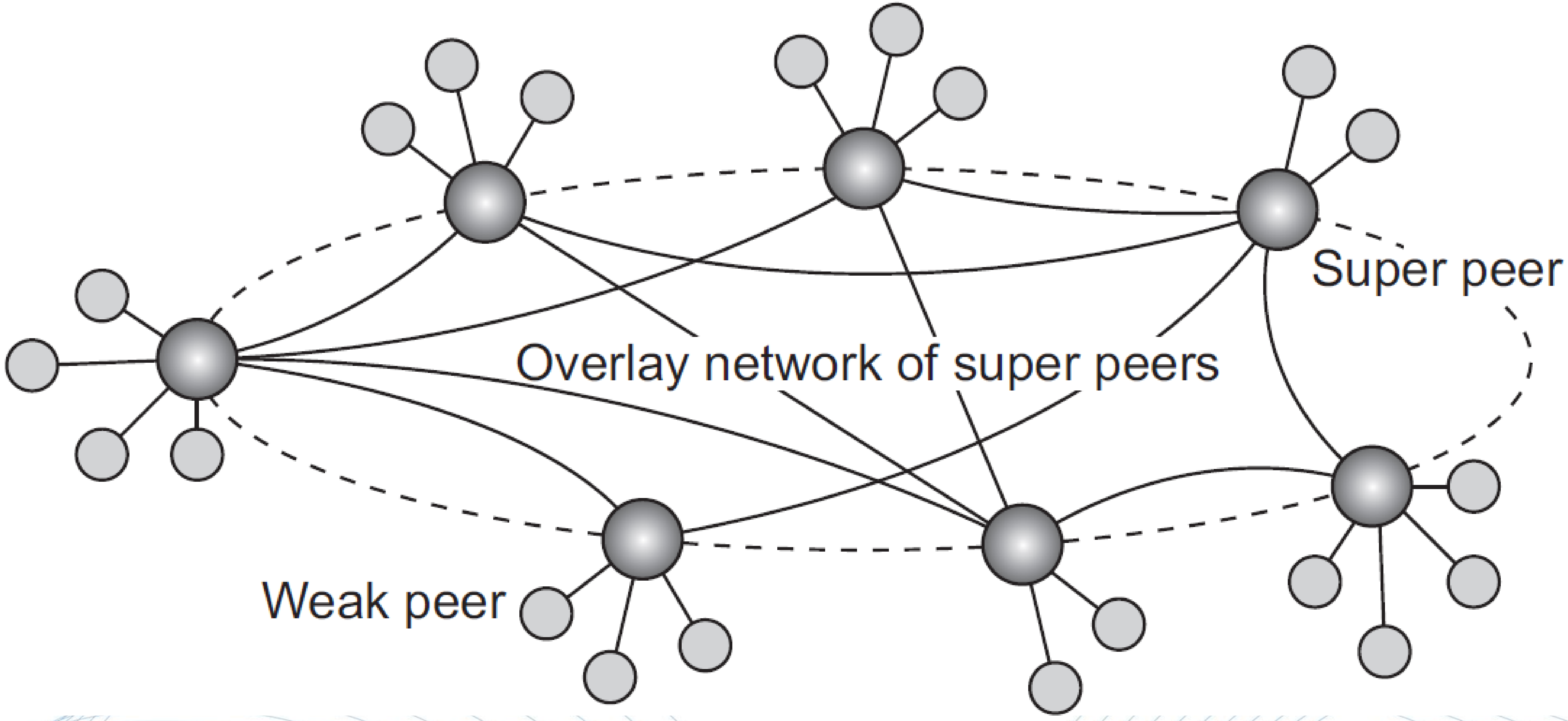


Image source: Baset & Schulzrinne.*An analysis of the Skype Peer-to-Peer Internet Telephony Protocol*. 25th INFOCOM conference 2006 p1-11

The beginning of Skype – A Peer-to-Peer overlay network

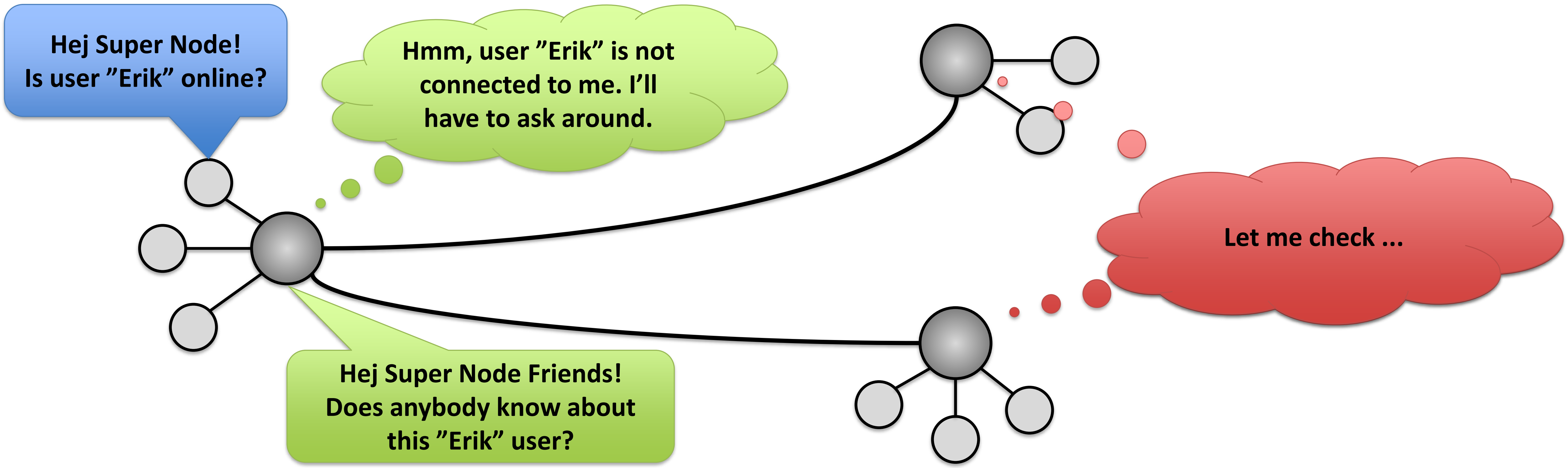
Super nodes form a super peer network



Searching for users and their addresses via the super peer network

A call itself is then setup between the clients itself (UDP, no firewalls) or with the help of super peers (UDP or TCP, at least on client behind a firewall)

Searching in unstructured P2P networks



Searching in unstructured P2P networks (2)

In unstructured P2P networks each node maintains an ad hoc list of neighbors. A link between two arbitrary nodes in the overlay network only exists with a certain probability. ➔ Random Graph

Whom to ask in the network?

- Random walk → Ask one of your neighbors
- Flooding → Ask all of your neighbors
- Middle ground → Ask some of your neighbors

Searching in unstructured P2P networks (3)

Random walk

- Choose one random neighbor and pass the request.
- If the chosen neighbor does not have the information, it forwards request to one of its randomly chosen neighbors, and so on.
- Stopping criteria:
    1. User/Data found
    2. TTL (time-to-live) counter that is reduced each hop
    3. ....

  ➔ Contact origin node with failure/sucess message

Searching in unstructured P2P networks (3)

Flooding

- Pass request to all neighbors.
- Request is ignored, if receiving node has seen it before.
- If the node does not have the information, send forward request to all neighbors (kinda recursive behaviour)
- Stopping criteria:
    1. TTL (time-to-live) counter that is reduced each hop
    2. Check with origin node whether the search is still ongoing

Searching in unstructured P2P networks (4)

Middle ground

- Initiate n random walks
- Policy / Knowledge based approaches

Comparison

- Flooding: Many messages, many nodes contacted, expensive transport costs, but fast
- Random walk: Fewer messages and nodes contacted to find the information, but slow

TASK:
Flooding vs. Random walk
(p. 86, Note 2.6)

Follow the numerical example for the random walk vs. flooding argument.

Interlude: Hash functions

A hash function h is a function that maps input x of arbitrary finite length to an output $h(x)$ of fixed bit length n.

- the above property is also called *compression*
- often assumed:
  Given h and x it is *easy* to compute $h(x)$

$h(username) \rightarrow \{0000, \ldots, 1111\}$

For example:
$h(Erik) \rightarrow 0110$

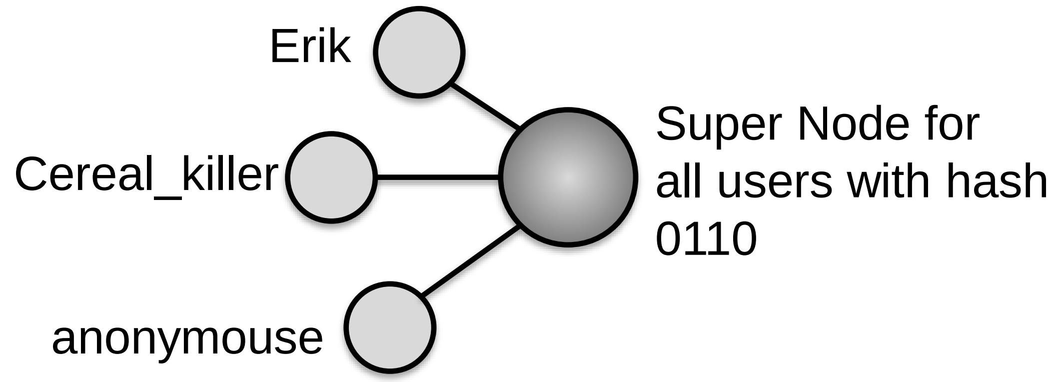
$h(GuineaPigDoc) \rightarrow 1111$

Python
```
>>>hash(”Distributed Systems”)
```

Structured P2P systems

Structured P2P networks arrange nodes in the overlay network in deterministic way with known existing links.

Toy Example: hypercube structure

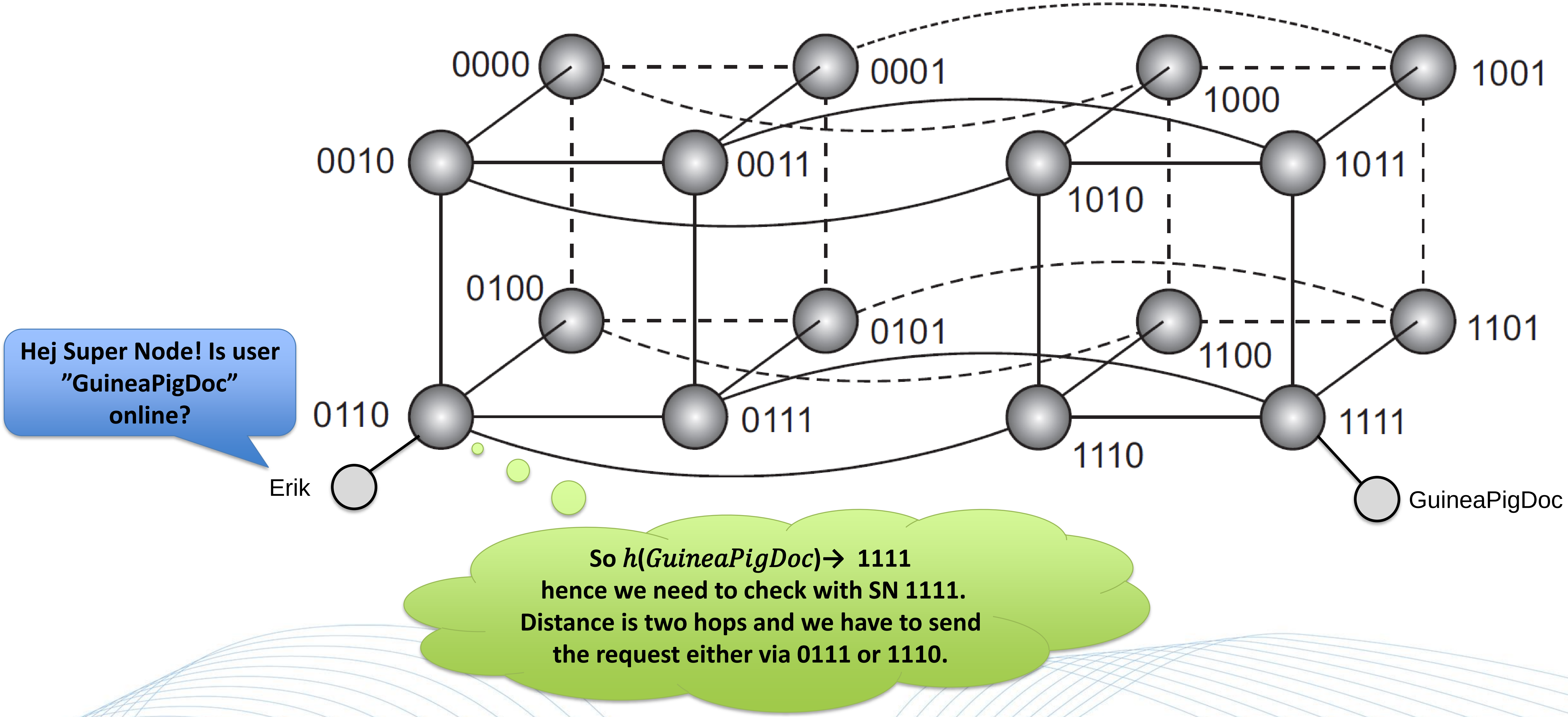


Idea:

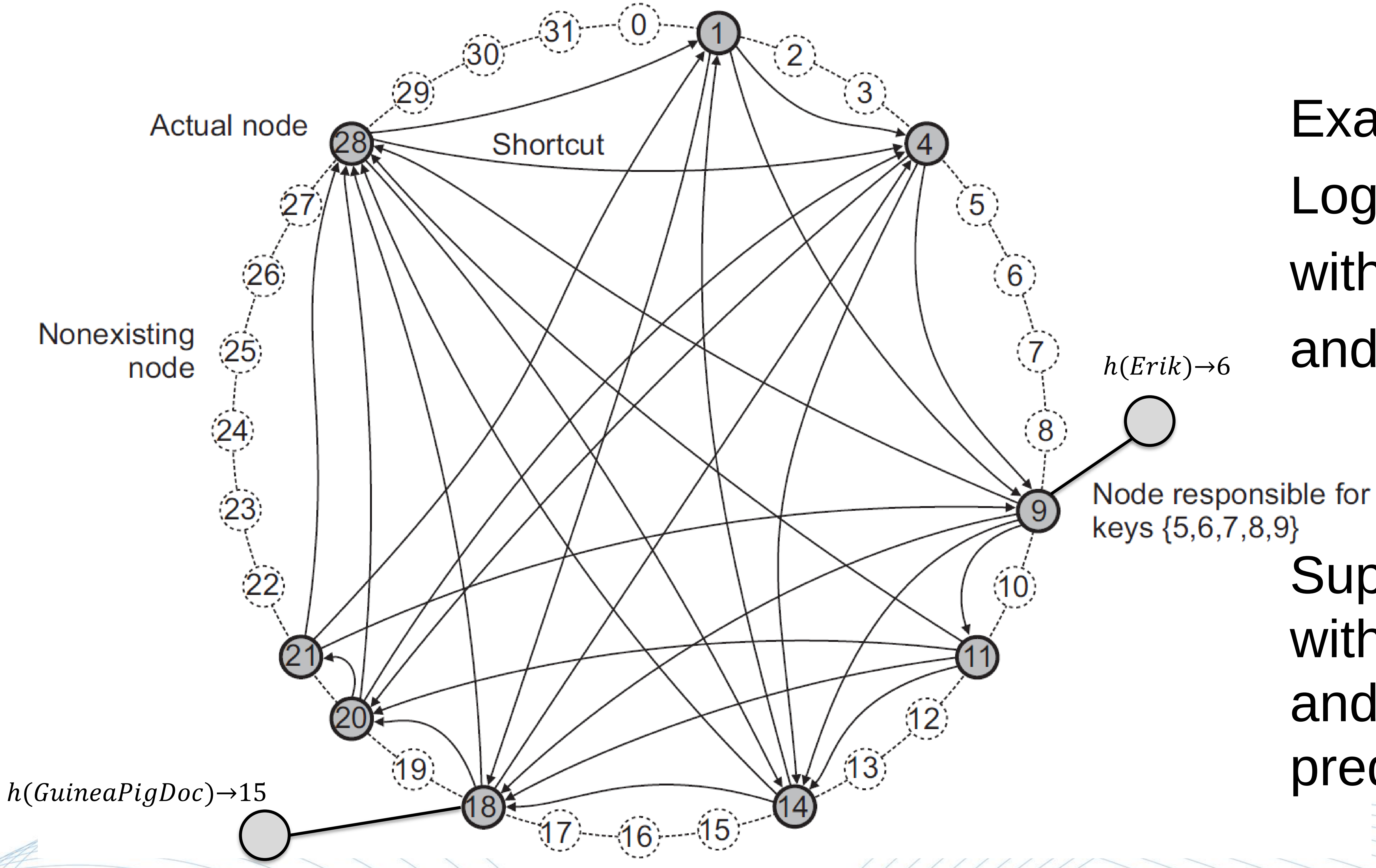
Use a hash function to assign values (usernames, files, etc) to certain nodes

➔ Distributed Hash Table

Searching in structured P2P networks



Searching in structured P2P networks



Example: CHORD

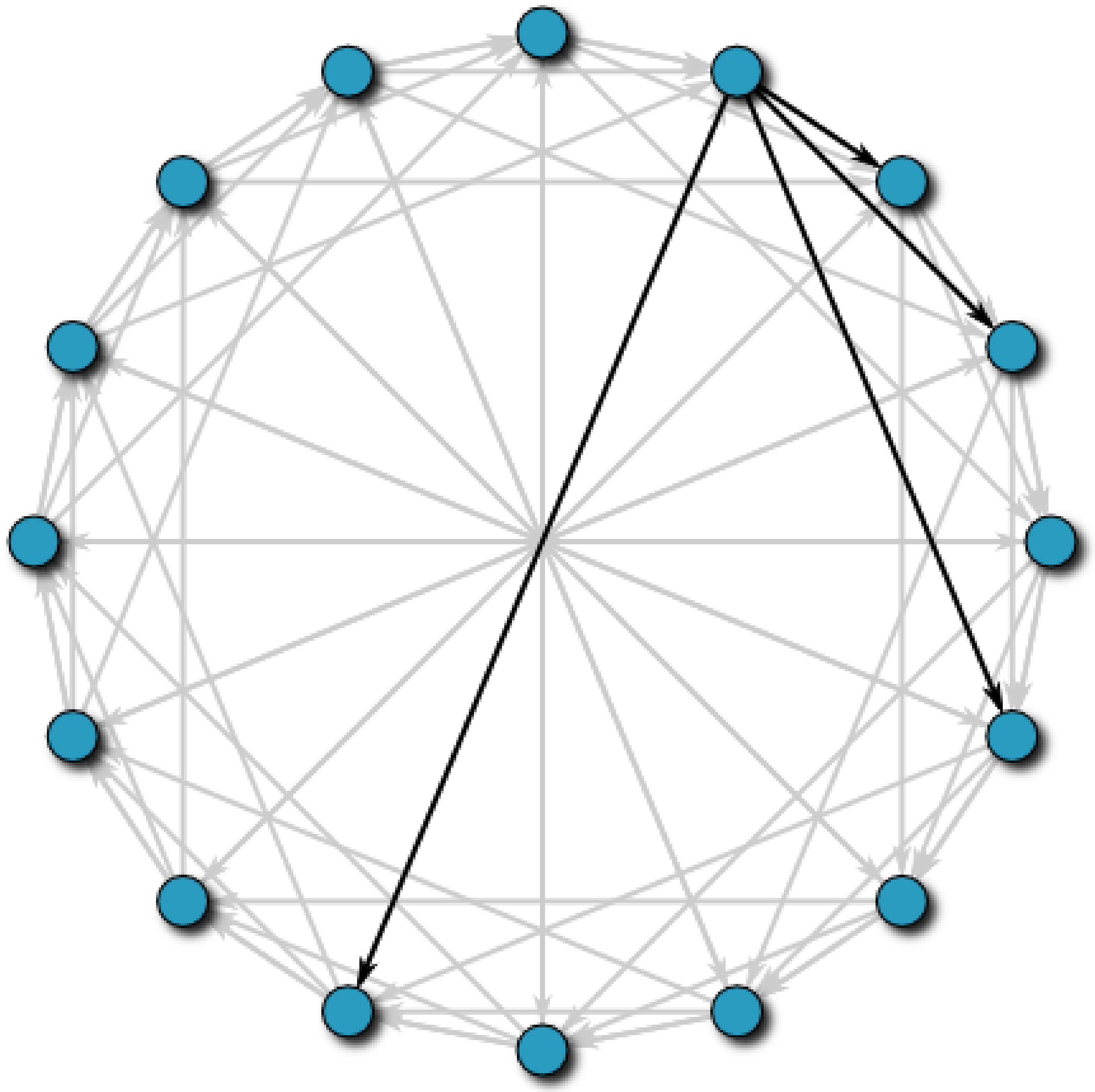
Logical ring structure with **unidirectional** links and 5-bit hash function

Super node connected with all its *own* hashes and all inactive predecessors

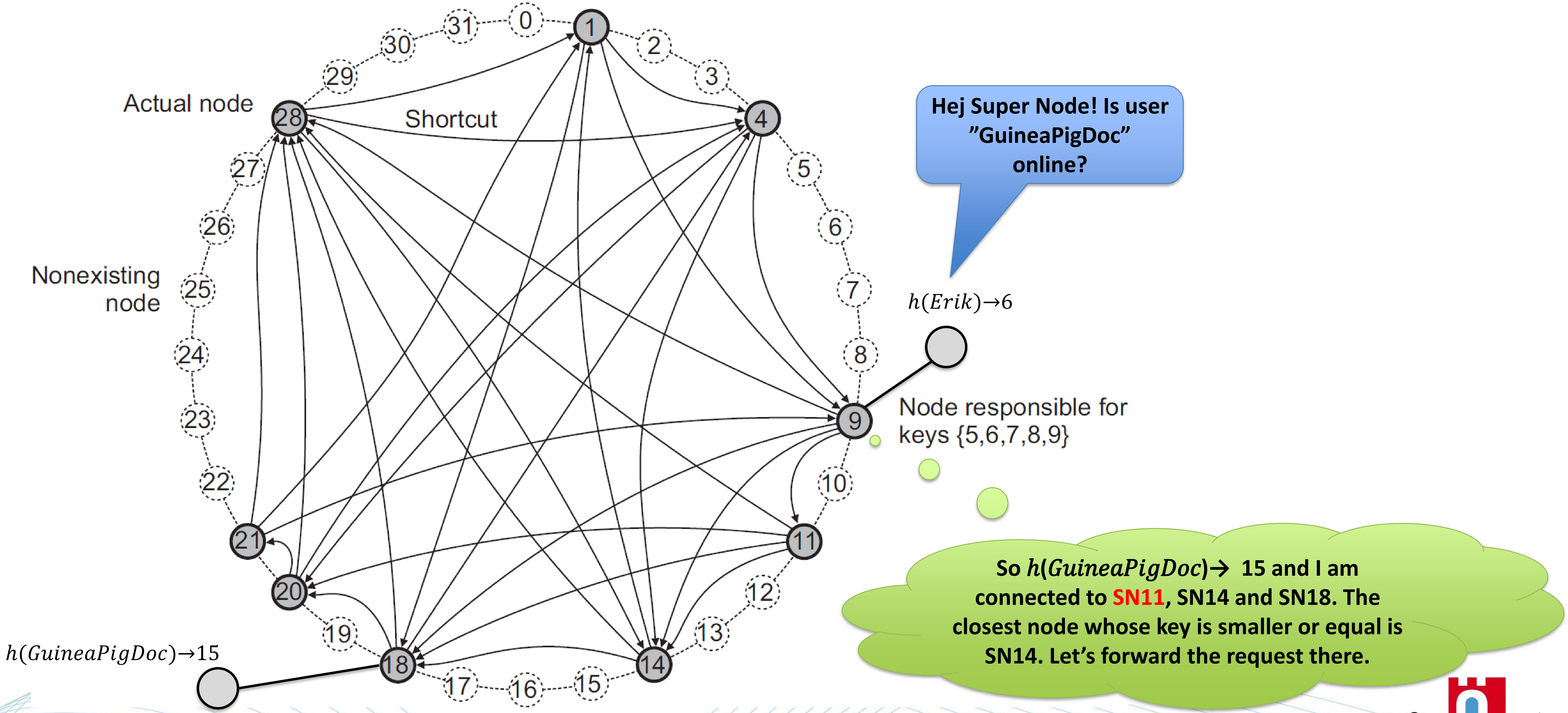
Ideal CHORD structure

- Avoid linear search $O(N)$ by using short-cuts (*fingers*)

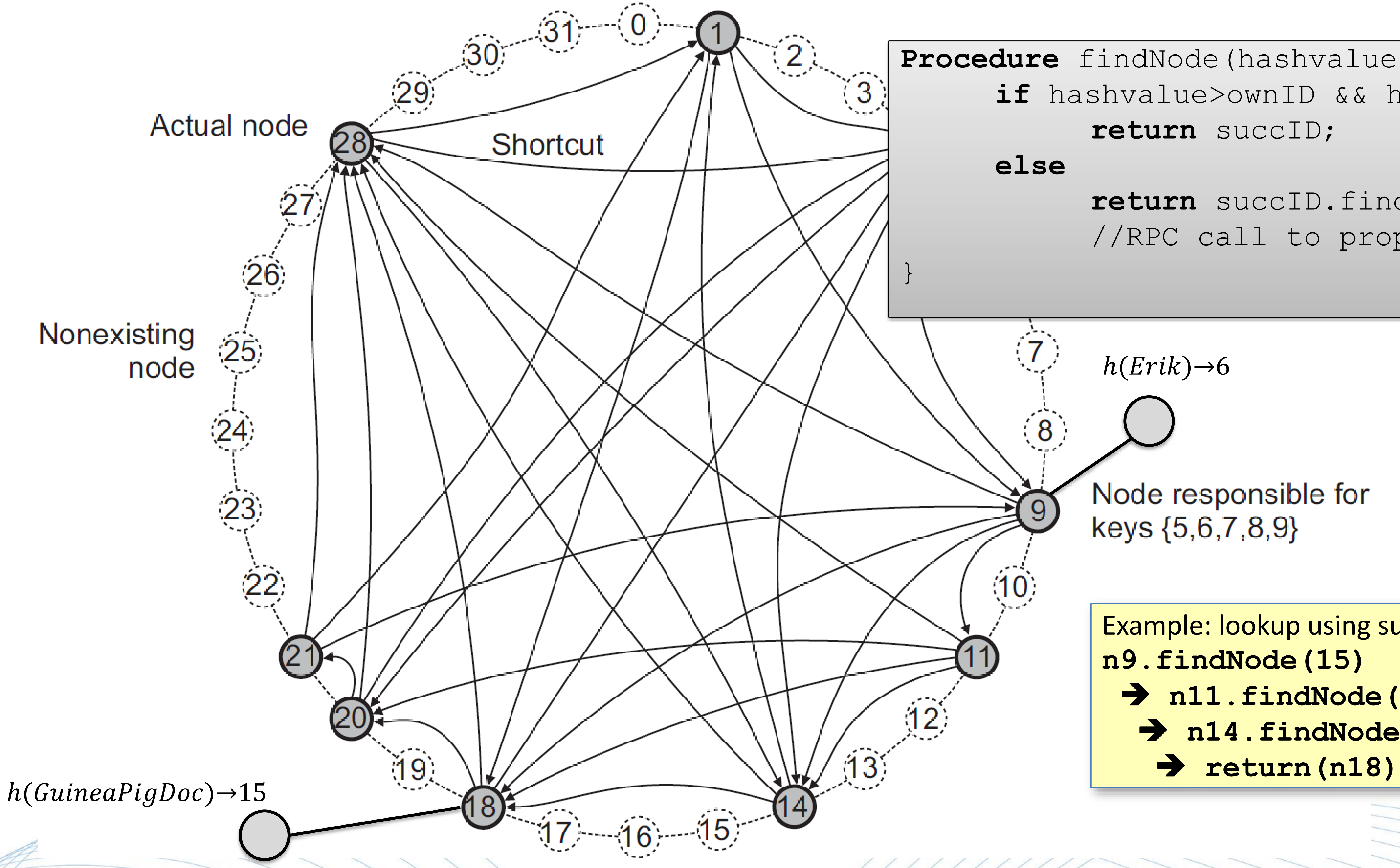
- binary-search behavior $O(\log(N))$ by creating connections to the 2^k-*th* predecessor (with $k = \{0,1,2,\ldots\}$)



Searching in structured P2P networks



CHORD search as an RPC




```
Procedure findNode(hashvalue) {
      if hashvalue>ownID && hashvalue<=succID
            return succID;
      else
            return succID.findNode(hashvalue);
            //RPC call to propagate search in the ring
}
```

Example: lookup using succesor links only

```
n9.findNode(15)
 ➔ n11.findNode(15)
  ➔ n14.findNode(15)
   ➔ return(n18)
```

Structured naming

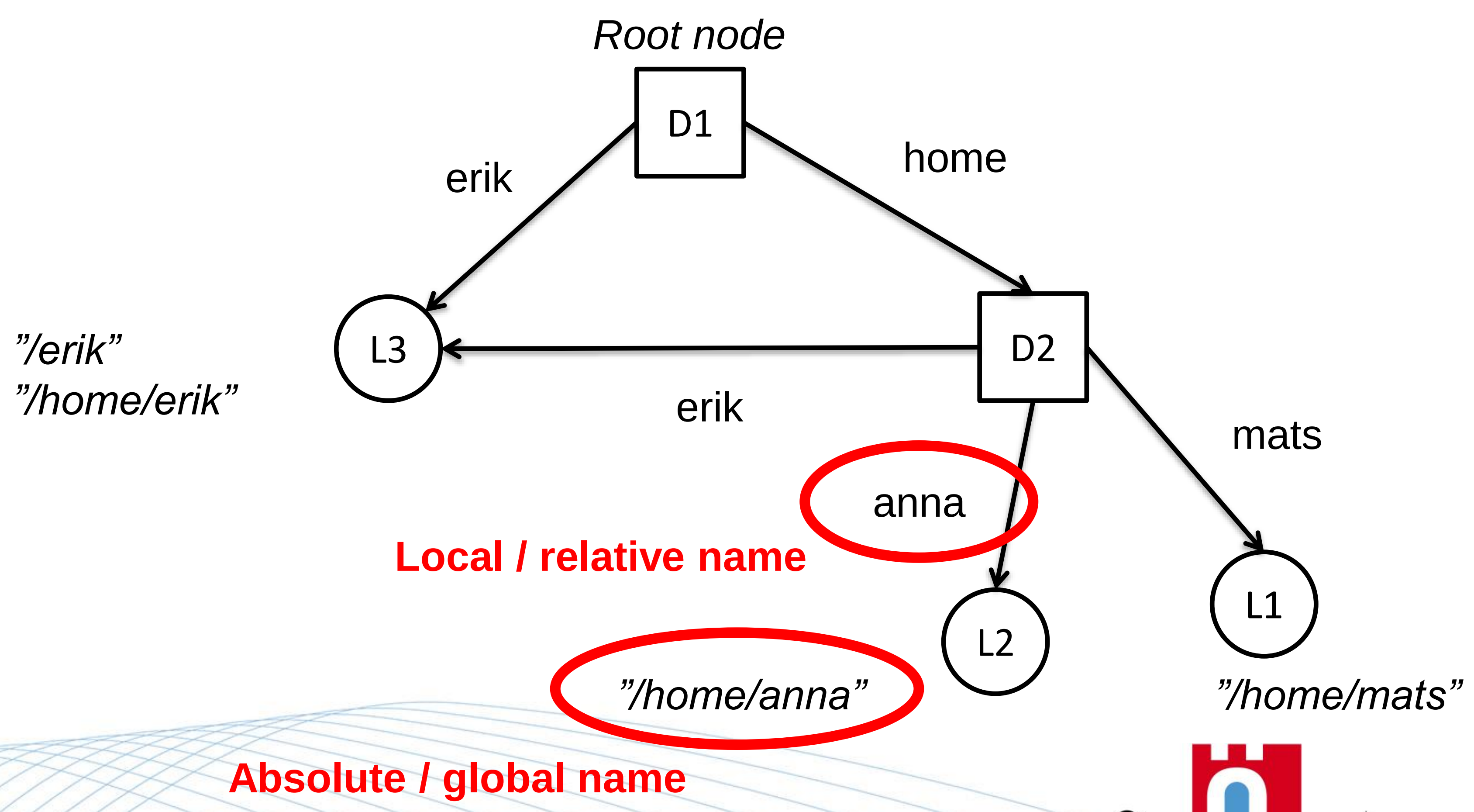
A **name space** is a labelled, directed graph consisting of **leaf nodes** and **directory nodes**.

Leaf node

represents a named entity,
e.g. by storing its address

Directory node

An entity that refers to other nodes



Name resolution

Problem

To resolve a name, we need to find and start at a directory node.

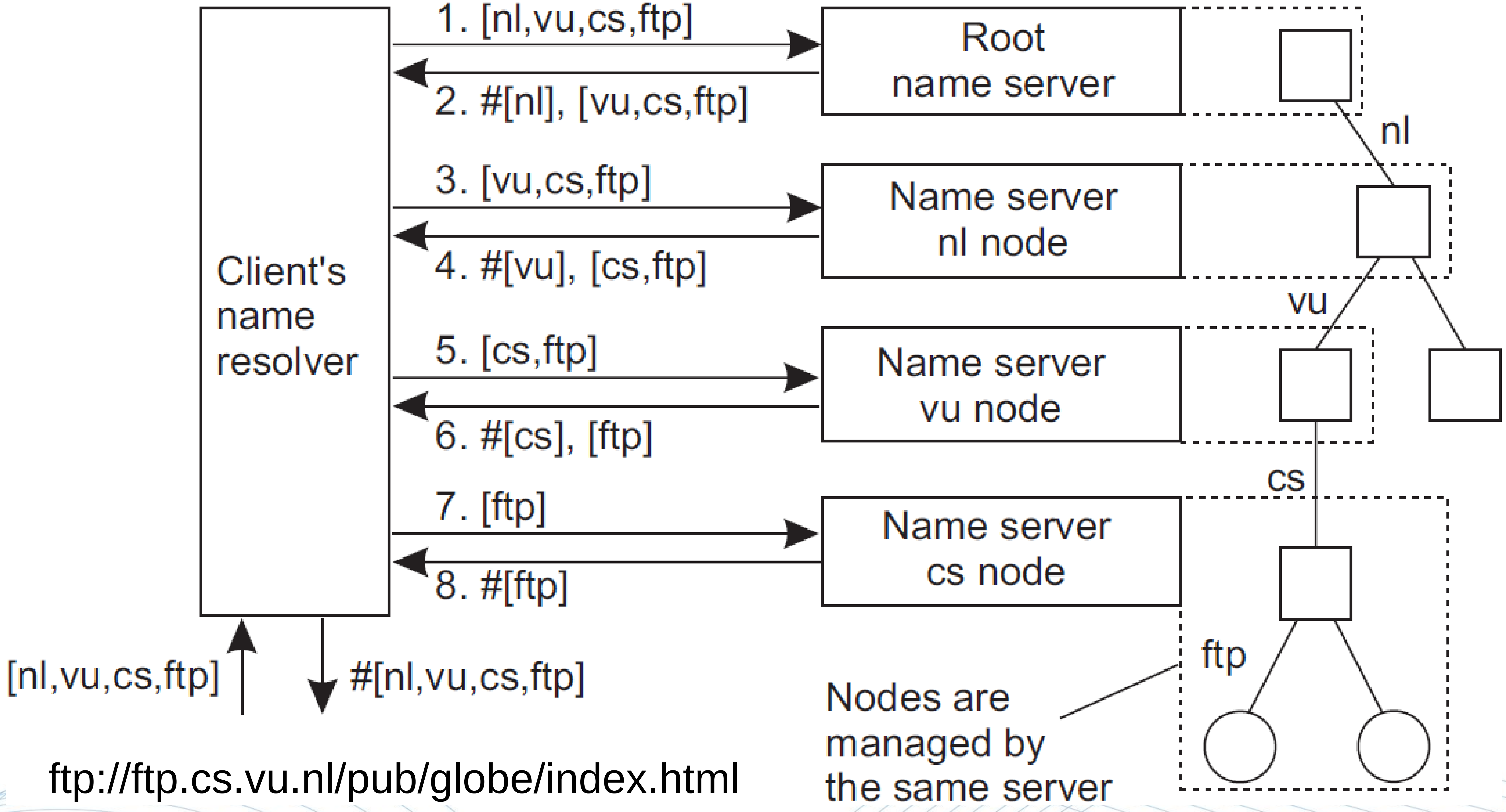
Closure mechanism

Select from the context where to start name resolution

- www.oru.se → start at a DNS server
- /home/erik/teaching → start at the local NFS file server
- 0046 19 30 3227 → dial a phone number
- 130.243.103.44 → route to that specific IP address

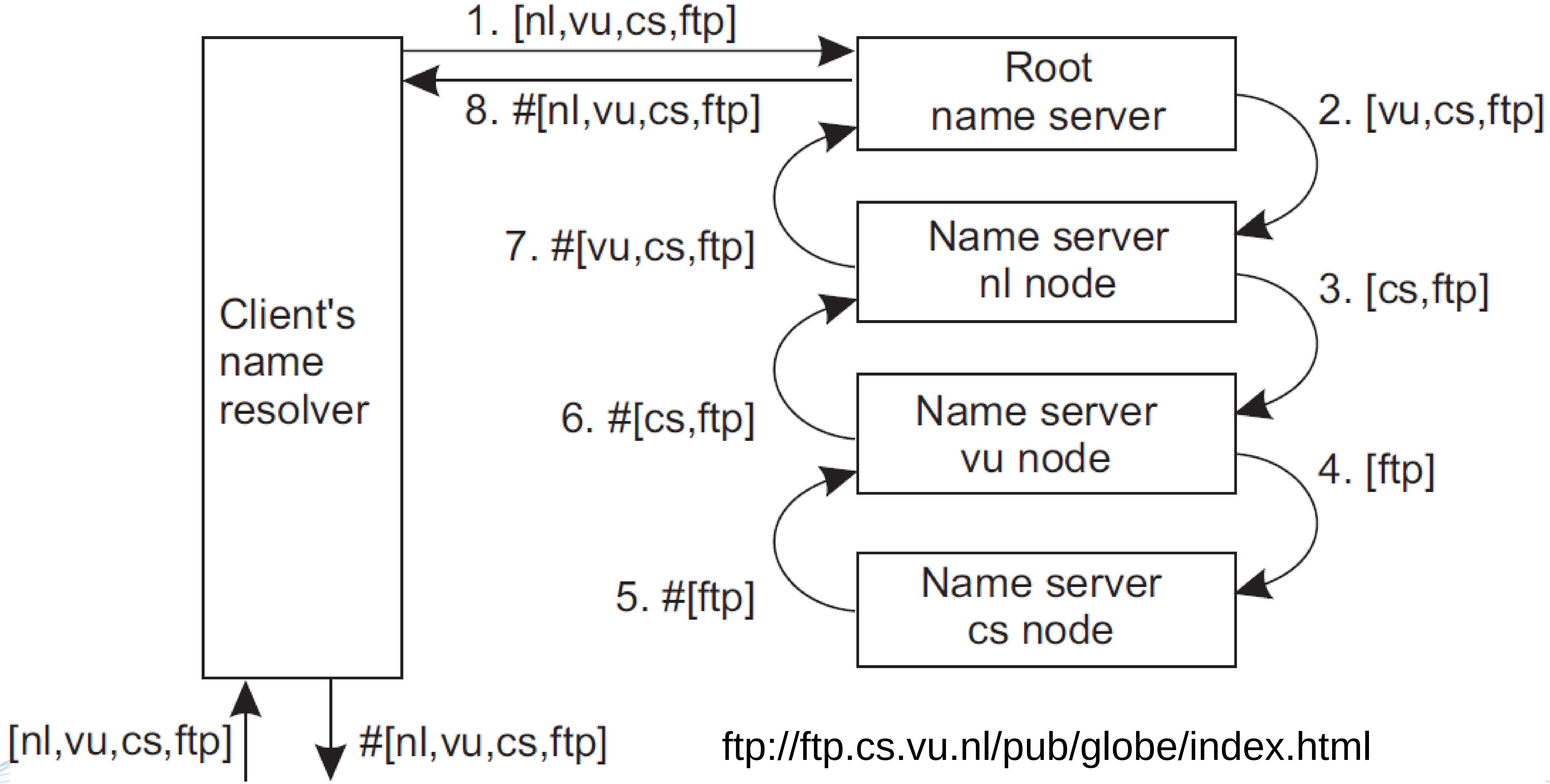
ÖREBRO UNIVERSITY

Iterative name resolution



ftp://ftp.cs.vu.nl/pub/globe/index.html

Recursive name resolution



ftp://ftp.cs.vu.nl/pub/globe/index.html

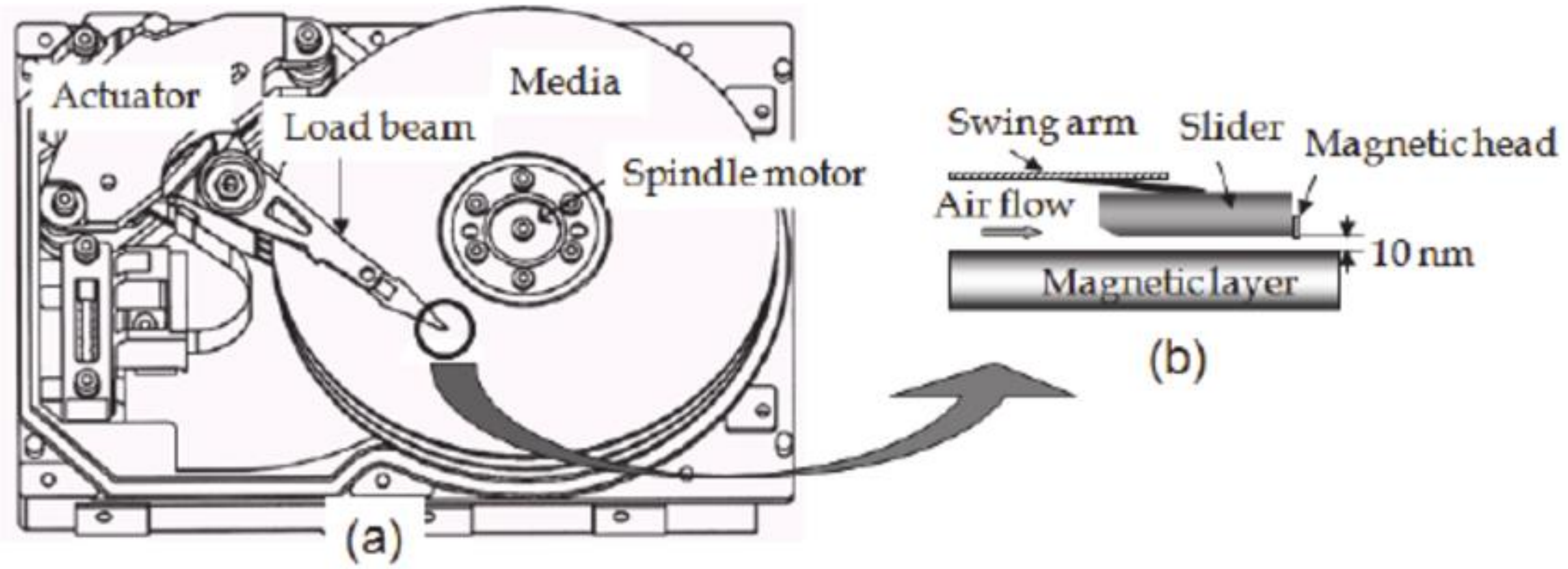
Size scalability

We need to ensure that servers can handle a large number of requests per time unit ➔ high-level servers are in big trouble.

Solution: Replication

- Top-level server is heavily replicated
  → start at the closest server
- Node content on the top-level rarely changes

Excursion: File systems



Purpose

Component of the OS that bridges the
hardware (surfaces, cylinder, sectors)
or a logical array of blocks with the a logical structure for storing data (directories, files)

Tasks

disk management, naming, access protection

Excursion: File systems

Files

- data with properties (content, size, owner, rights, etc.)
- can have a type relevant for the file system
  - block, device, link, directory etc.
- can have a type relevant for other parts of the OS
  - executable, library, text, image
- basic file operations (dependent on OS)
  - `create`, `open`, `close`, `read`, `write`, `move`, etc.

Excursion: File systems

Directories

- provides the user with a way to organize files
- provides the FS with a structured naming interface to separate logical files from physical file placement on the disk
- naming hierarchies (/, /home/, /home/erik/)
- table of `<name, location>` pairs

ÖREBRO UNIVERSITY

Excursion: File systems

Path Name Translation

Example: Open `/home/erik/mysteriousfile`

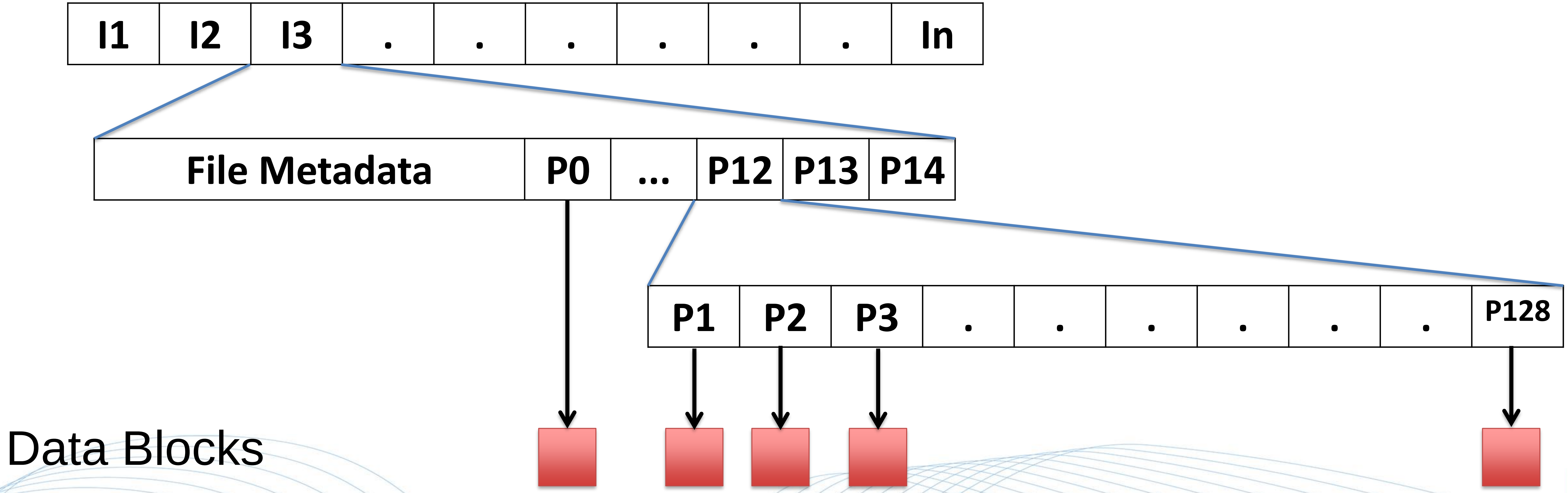
- Open file ”`/`” ➔ *root* position is always known
- Search for entry ”`home/`” and get its location
- Open file ”`home/`”, search for ”`erik/`” and get its location
- Open file ”`erik/`”, search for ”`mysteriousfile`” and get its location
- Open file ” `mysteriousfile`”

➔ walking along the directory paths

Excursion: File systems

Example: Inodes in UNIX file systems (e.g. extX family)

- roughly 1% of the space is metadata
- Inode array on disk at a well known locations

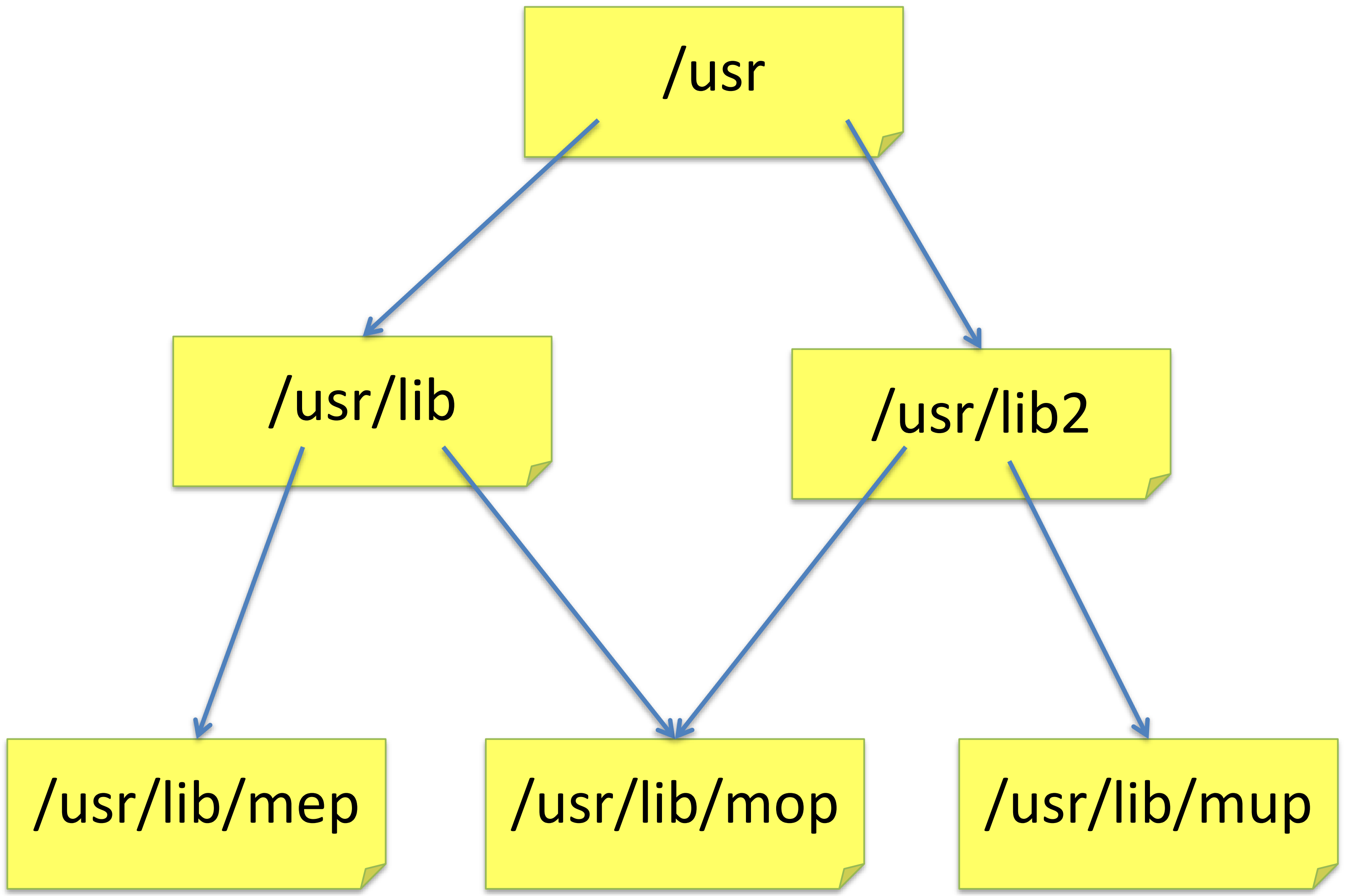


Excursion: File systems

Hard Links

- pointer to the file number on the disk or actual data on the disk

```
usr/lib:                 usr/lib2:
     (mep, #5064)             (mup, #1928)
     (mop, #3227)             (mop, #3227)
```

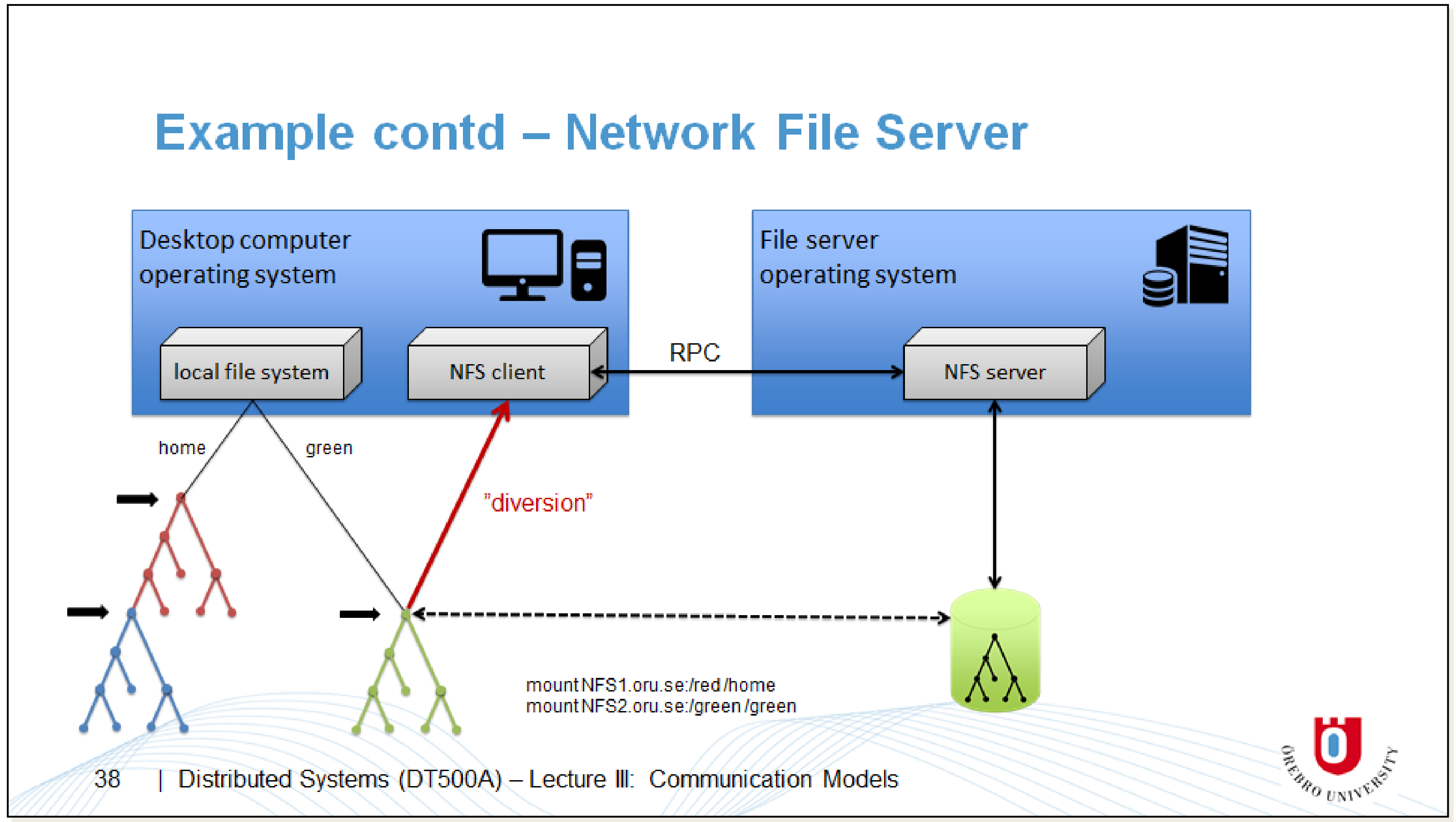


Excursion: File systems

Soft or Symbolic Links

- different directory entry
  instead of: (*sourceFilename, #file*)
  have: (*sourceFilename, destinationFilename*)
- OS looks up *destinationFilename* each time *sourceFilename* is accessed
  - lookup might fail
  - can link files on a different disk

Mounting



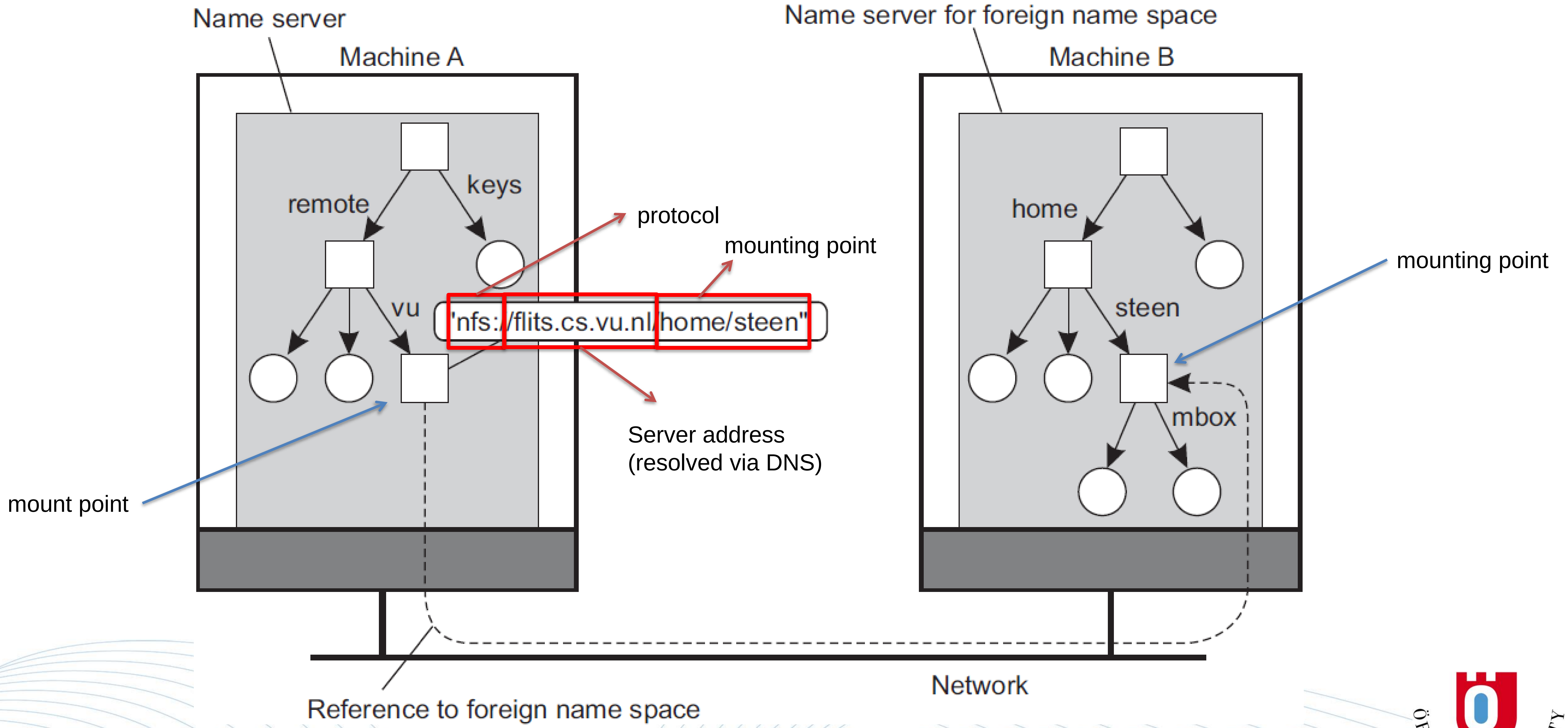
Idea

Merge different name spaces transparently by connecting the node identifier of a foreign name space with a node in the current name space

Required Information

- Used access protocol
- Server name
- Mounting point in the foreign name space

Mounting in distributed systems



Attribute-based naming

Idea

Naming and look-up of entities by means of their attributes

➔ Directory services

Problem

Comparison of requested and actual attributes is expensive

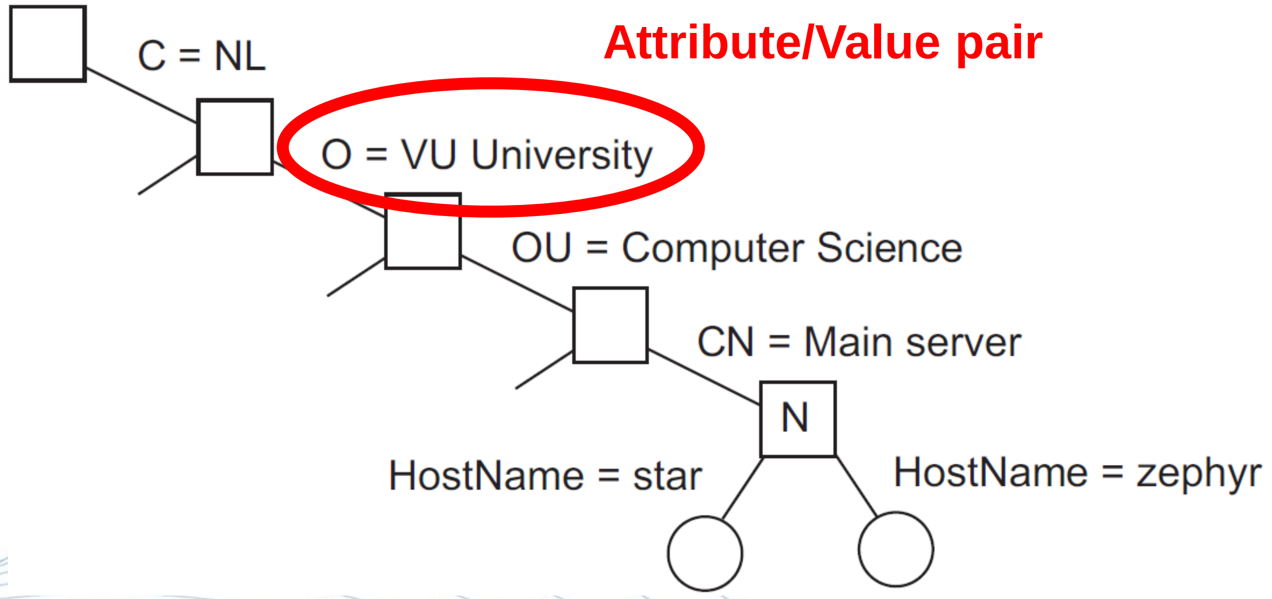
➔ Inspection of all entities required

Attribute-based naming

Solution

Combination of structured naming with a database implementing the directory service

Example: LDAP – Lightweight directory access protocol



Attribute	Value	Attribute	Value
Locality	*Amsterdam*	*Locality*	*Amsterdam*
Organization	*VU University*	*Organization*	*VU University*
OrganizationalUnit	*Computer Science*	*OrganizationalUnit*	*Computer Science*
CommonName	*Main server*	*CommonName*	*Main server*
HostName	*star*	*HostName*	*zephyr*
HostAddress	*192.31.231.42*	*HostAddress*	*137.37.20.10*

Result of `search(''(C=NL)(O=VU University)(OU=*)(CN=Main server)'')`