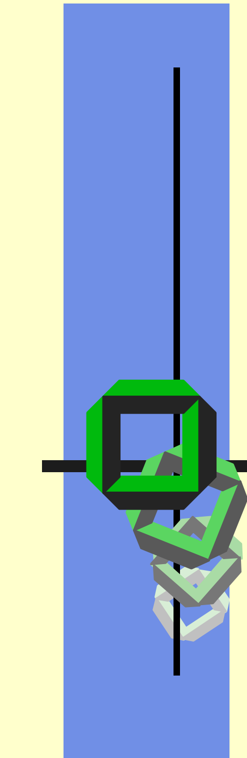


Distributed Systems

9 Naming

June-08-2009

Gerd Liefländer

System Architecture Group

© 2009 Universität Karlsruhe (TH), System Architecture Group

Schedule of the Week

- Motivation & Introduction
- Basic Terms
- Naming System
  - Flat Naming
  - Hierarchical Location Service
  - Structured Naming
- Implementation of a Name Service
  - Name Space Distribution
  - Name Resolution
- Examples
  - Domain Name Service
  - GNS
- Attribute Naming

© 2009 Universität Karlsruhe (TH), System Architecture Group

Motivation

"Any problem in computer science can be solved with another layer of indirection"

David Wheeler

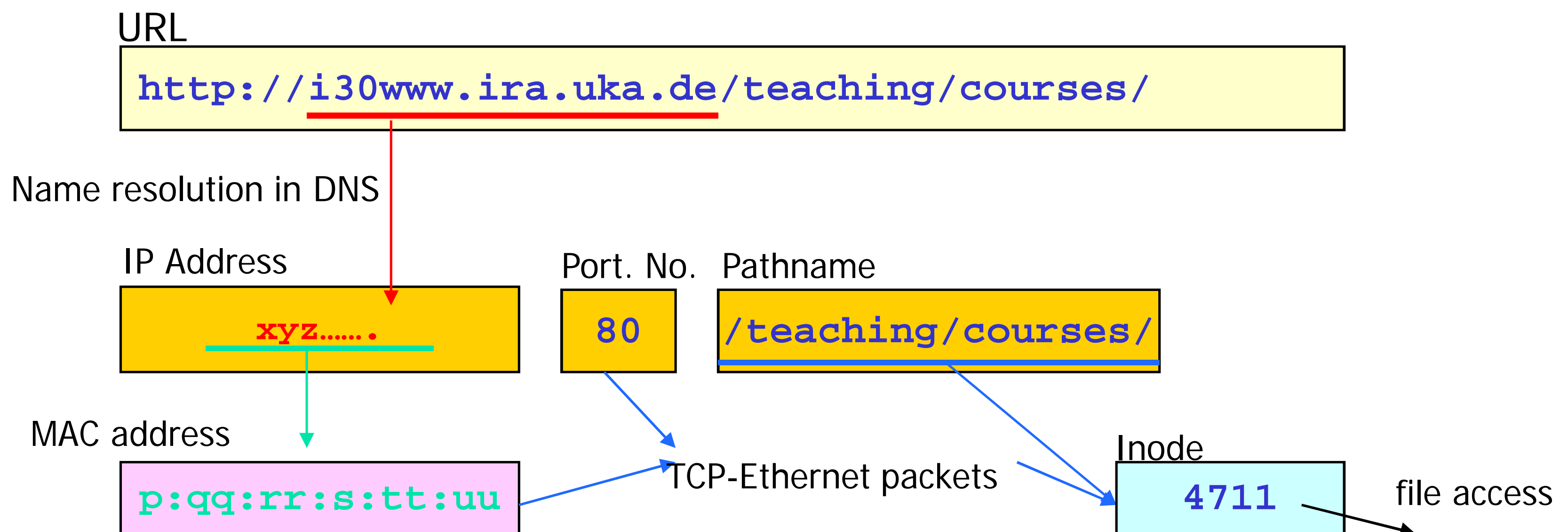
Naming is a Layer of Indirection

- *What problems does it solve?*
  - Makes objects human readable
  - Hides complexity and dynamics
    - Multiple lower-layer objects can have one name
    - Changes in lower-layer objects
  - Allows an object to be found in different ways
    - One object, e.g. a file can have multiple names
    - You can navigate to the target object via absolute or relative pathnames

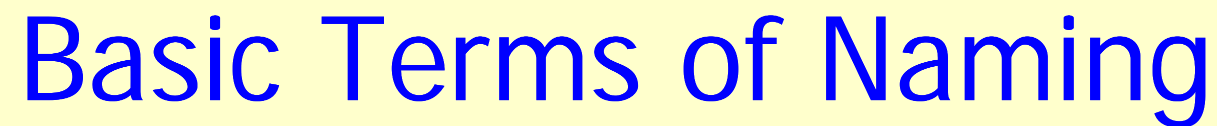
Motivation

- People prefer names
  - do not like to remember the Inode number of their files
- Programs (machines) use
  - identifiers, e.g. PIDs 0815 or
  - addresses, e.g. `0xffffffff`
- Needed a mapping of

  names → identifiers or addresses



Basic Terms of Naming



Names

Identifiers

Addresses

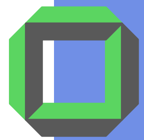
Attributes

© 2009 Universität Karlsruhe (TH), System Architecture Group

Names

- Names are helpful to ...
  - explain
    - "talking" using names helps understanding
    - e.g. mail server
  - identify
    - A name should reference an entity unambiguously
    - e.g. `lief@ira.uka.de`
  - address, i.e. locate
    - A name hides the actual location of an entity or of an access point of a server
    - e.g. `i30www.ira.uka.de` hides the IP address of our web server
- A name is valid only in its context, e.g. identical names can reference different entities, e.g.
  - "4711" may reference an Inode, a UID, a PID, a phone number, a product name

© 2009 Universität Karlsruhe (TH), System Architecture Group



Names

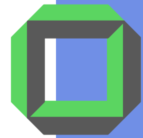
- Name: string of characters used to refer to an entity
  - Pure names with no additional info of the entity, e.g. **`4711`**
  - Impure names with additional info of the entity, e.g. **`i30www.ira.uka.de`**
- Entity: any resource/object in a DS (another name)
- To work with an entity, you need its access point(s), i.e. you must use its address(es)
- If an entity moves to another location it needs a new access point (e.g. IP address of host + port number)
- Alias: One of several names for the same entity

© 2009 Universität Karlsruhe (TH), System Architecture Group

Naming Terms

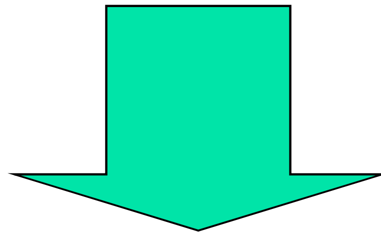
- Name service: Service that maps a name to its entity. It can be centralized or distributed
- Name resolution: Act of resolving a name to its entity
- Location independent: refers to same entity regardless of the location at which it is resolved
- Location transparent: does not indicate the location of the entity it refers to
  - Precondition for object migration
- Human-friendly: A possible character string that helps to identify the content of the named object

© 2009 Universität Karlsruhe (TH), System Architecture Group



Names → Objects …

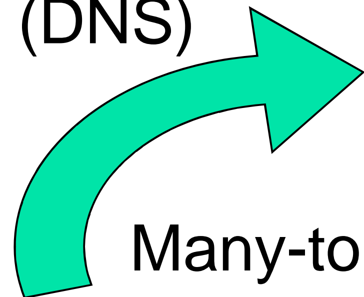
- Names map to objects through a resolution service



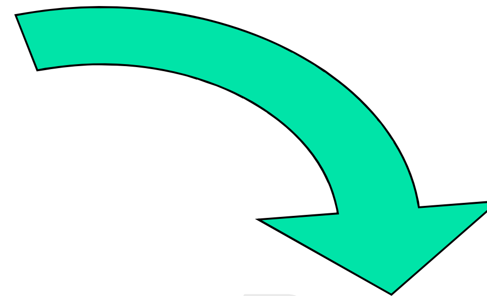
Naming in Networks

DNS Names map into Addresses

Domain Name System (DNS)

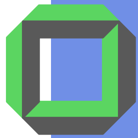


Many-to-many



Domain Name (www.cnn.com)

- Hierarchical
- User-friendly
- Location independent
- But not org independent



Addresses map into Routes

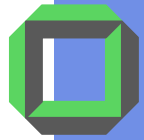
IP address
(128.94.2.17)

Routing algorithm
(BGP, OSPF, RIP)

Name → Address → Route

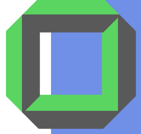
One-to-many

- Hierarchical
- Location Dependent
- Non-unique
- Can change often
- Refers to an interface, not a host



Routes get Packets to Interfaces

- A path
- Source dependent
- Can change often



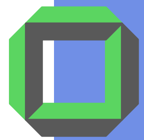
Identifiers and Locators

- A name is always an identifier to a greater or lesser extent
  - Can be persistent or non-persistent
  - Can be globally unique, locally unique, or even non-unique
- If a name has structure that helps the resolution service, then the name is also a locator

Name $\leftrightarrow$ Identifier

- Names can be easily interpreted by humans, e.g.
  - file name index.html
  - Internet domain www.peterandthewolves.de
  - URL http://www.jimmy-world.de
- Identifiers are easily interpreted by programs
  - Communication endpoint of an IPC
  - File descriptor in NFS
  - Reference within an RPC
- Every process accessing a resource must either know its name or its identifier, e.g. typically, an application opens a file via its filename and further works with that file using its file identifier (handle)

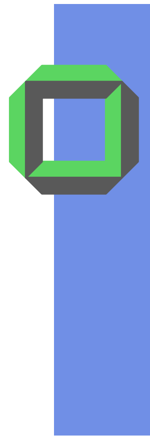
© 2009 Universität Karlsruhe (TH), System Architecture Group



Identifier & Address

- True identifier: A “name” that
  1. refers to at most one entity (unique in space)
  2. always refers to the same entity (unique in time)
  3. each entity is referred to by at most one identifier
  - identifiers with all three properties are sometimes called GUIDs (globally unique identifier)
- Address: A “special name” that refers to a special entity called an access point
  - provides access to another entity, e.g. Jimmy’s office desktop, Jimmy’s cellular phone

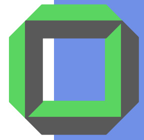
© 2009 Universität Karlsruhe (TH), System Architecture Group



Uniform Resource Identifier

- URL (Uniform Resource Locator) = address of a web-resource
  - Efficient and scalable for an unlimited set of web-resources
  - If resource migrates ⇒ problem of dangling reference
- URN (Uniform Resource Name) solves the problem of dangling references
  - Migration transparent persistent name of the web resource
  - Saved together with **current URL** in a URN-Lookup service
  - Example: **`urn:ISBN:0-13-183369-3`**

© 2009 Universität Karlsruhe (TH), System Architecture Group



Uniform Resource Identifier

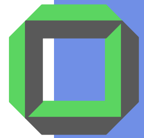
- URC (Uniform Resource Characteristic) is completely based on attributes of a web-resource
- The (abstract) notation may serve as a search key:

  “author =Jochen Liedtke”

  “title =Towards Real Micro Kernels”

  “keywords =micro kernels, ….”
- Is a URN-subset
  - Is saved with (URL and URN) in a URC-look up service

© 2009 Universität Karlsruhe (TH), System Architecture Group



Attribute

- Attribute
  - Additional information of an entity
  - Bind a name to its attributes
- Example attributes of a machine
  - IP address
  - HW architecture …
- Example attributes of a user
  - UID
  - User's family name, Christian name, …
  - Encrypted password

© 2009 Universität Karlsruhe (TH), System Architecture Group

Flat Naming

- *Given an identifier how to locate the entity?*

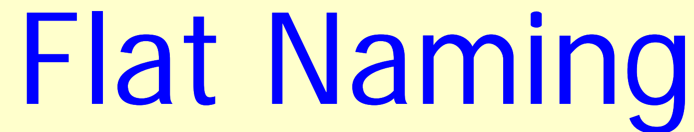
Structured Naming

- *How to map human friendly names to addresses of entities?*

Descriptive Naming

- *How to locate an entity given some of its attributes?*

Flat Naming



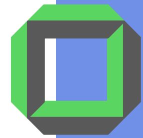
Mapping: *Identifiers* to *Addresses*

<u>Assumption:</u>

Identifiers are jus a bit string with no additional information on how to locate the desired entity

Goal: Find an/the appropriate access point of the desired entity

© 2009 Universität Karlsruhe (TH), System Architecture Group



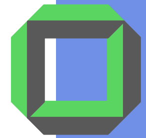
Application of Flat Naming

- Simple Solutions
  - Broadcast or Multicast
  - Forwarding Pointers
- Home Based Approach
- P2P Application
- Flooding
- Distributed Hash Tables
- Hierarchical Approaches
  - Example: Globe

Only in LANs with few nodes

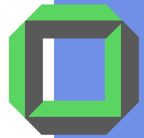
Scalable

© 2009 Universität Karlsruhe (TH), System Architecture Group



Requirement for Simple Solutions

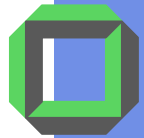
- Local area network (LAN)
  - (Fast) Ethernet
  - IBM Token Ring
  - …
  - Wireless local network
- *Simple solution? How?*
  - Just broadcast/multicast a message with object identifier
  - Machine hosting the identified entity will answer with its access point, e.g. with its Ethernet address



Analysis of Broadcast/Multicast

- Broadcast identifier to every node of the DS
- Solution does not scale with number of nodes
- Superfluous message overhead at unrelated sites & interruption of current work
- Instead of a broadcast you can use a multicast to a subset of the nodes
  - With HW support (e.g. Ethernets support data-link multicasting) network overhead can be reduced
- Multicasting is also used to get the access point of the nearest/best replica in case of replicated entities
  - Simply take the one who's reply arrives first
  - *Better policies?* See later

© 2009 Universität Karlsruhe (TH), System Architecture Group



Multicast Group/Address

- Internet supports network-level multicasting by allowing hosts to join a specific multi-cast group
- Such groups are identified by a multi-cast address
- If a host uses such a multi-cast address, the network-layer provides a best-effort service to deliver the message to all group members
- Use of a multicast address can also be to locate a nearby server of a set of replicated servers

© 2009 Universität Karlsruhe (TH), System Architecture Group

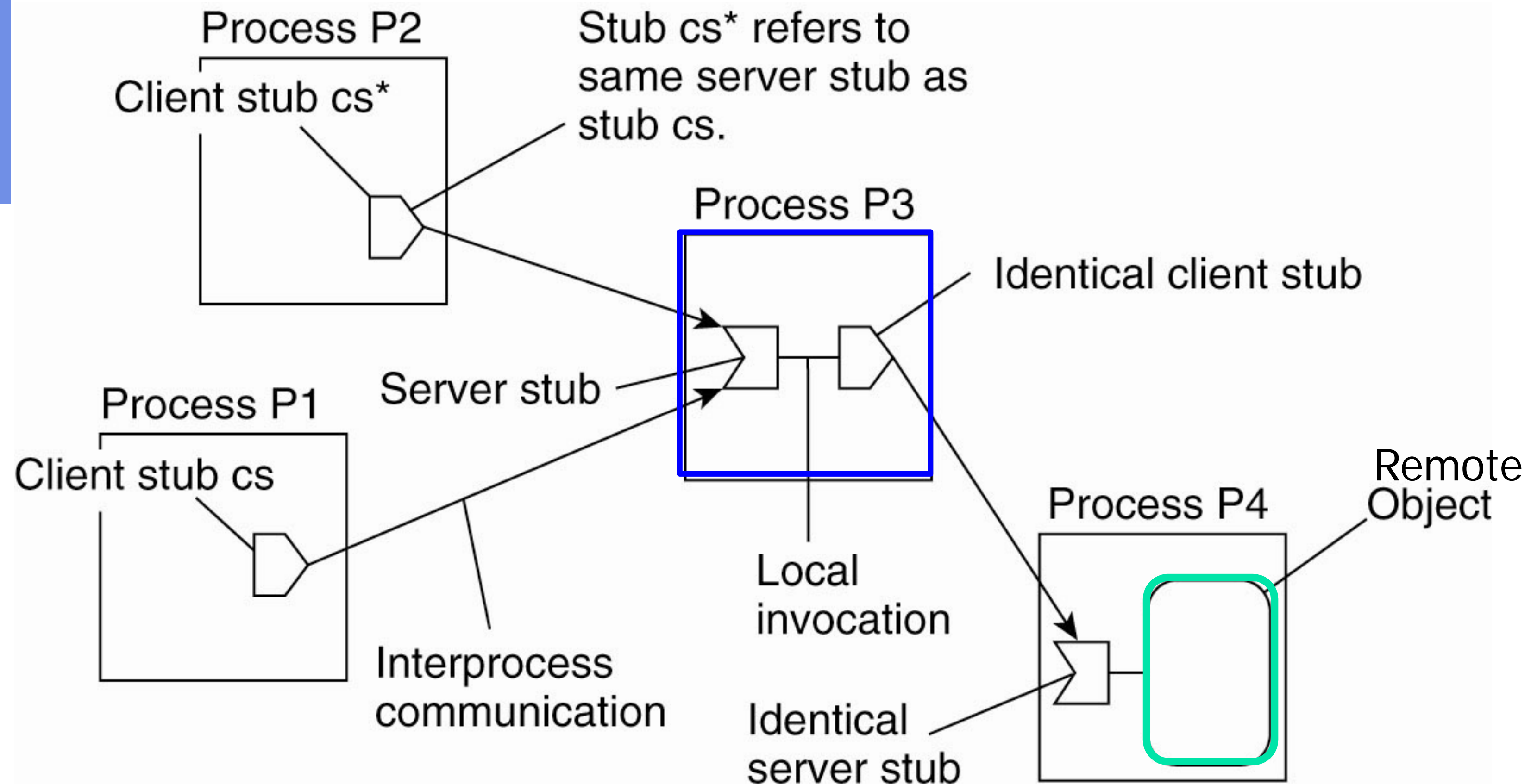
Forwarding Pointers[1]

- When an entity moves from site A to site B, it leaves behind in A a reference to its new location at site B
- Whenever a name has been resolved it will reference to the first location, from where a pointer chain can be traveled to find the current location of an entity
- Simple, however, chain must not be broken
- In case of a highly mobile entity the chain might become too long

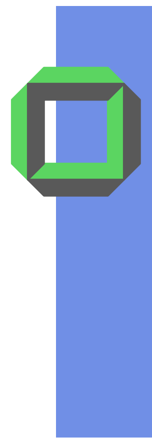
[1]Marc Shapiro et al.: SSP Chains: Robust, Distributed References Supporting Acyclic Garbage Collection, 1992 see: **http://www-sor.inria.fr/projects/sspc/**

© 2009 Universität Karlsruhe (TH), System Architecture Group

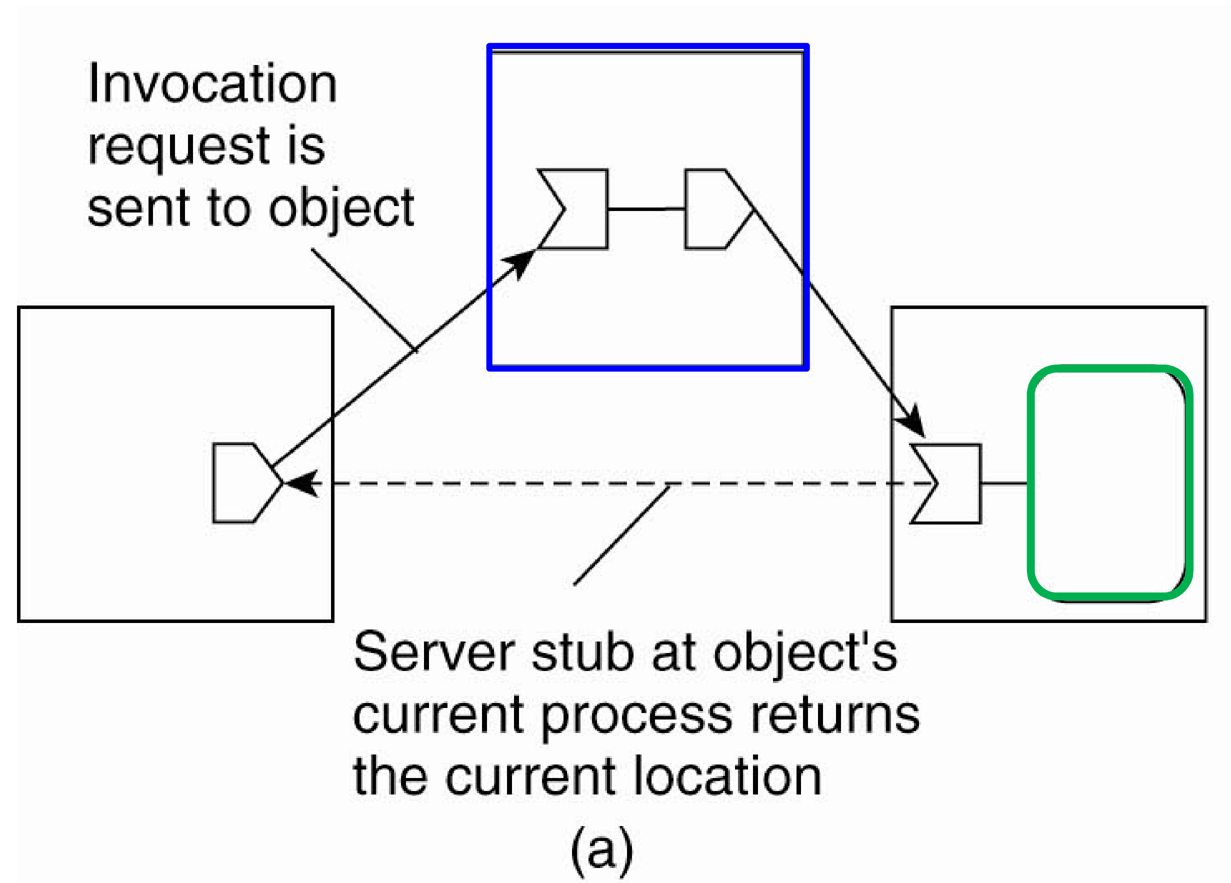
Forwarding Pointers (1)



- Principle of forwarding pointers using **(client stub, server stub)** pairs

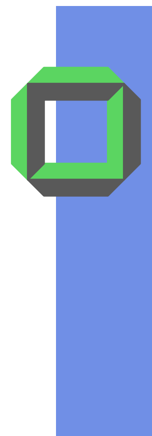


Forwarding Pointers (2)

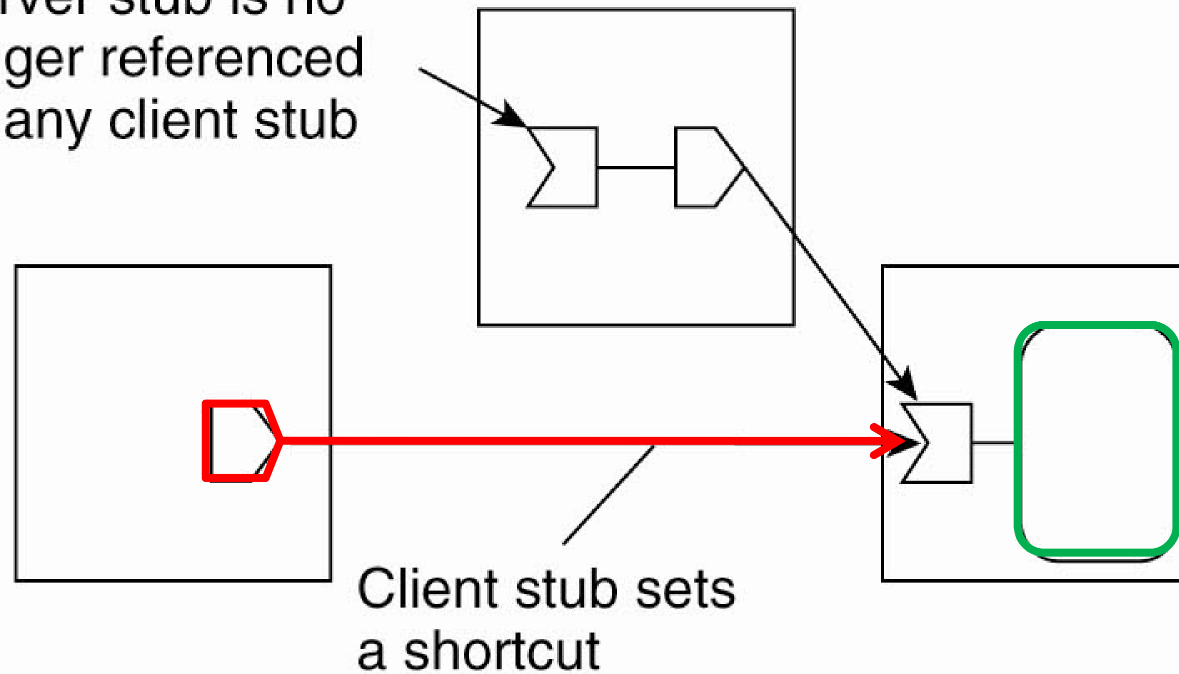


- Redirecting a forwarding pointer by storing a shortcut in a client stub.

© 2009 Universität Karlsruhe (TH), System Architecture Group



Forwarding Pointers (3)



- Redirecting a forwarding pointer by storing a shortcut in the client stub.

Home-Based Approaches

Host's home location

1. Send packet to host at its home

2. Return address of current location

Client's location

3. Tunnel packet to current location

4. Send successive packets to current location

Host's present location

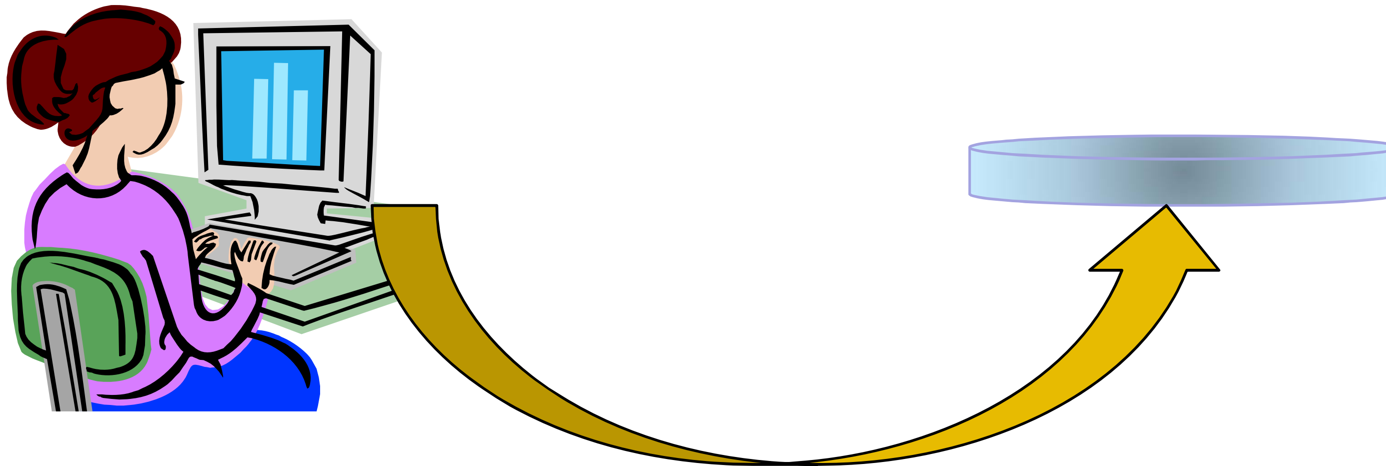
- The principle of Mobile IP

© 2009 Universität Karlsruhe (TH), System Architecture Group

Peer to Peer Systems

© 2009 Universität Karlsruhe (TH), System Architecture Group

P2P Application



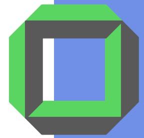
- Alice wants to find a specific music video (e.g. John Lennon's Imagine) via a key (identifier)

© 2009 Universität Karlsruhe (TH), System Architecture Group

Lookup via Flooding

- Forward the query to all immediate neighbors (except the one that has sent the query)
- Duplicates to the same target can be skipped, however this requires that each query gets an unambiguous sequence number
- Analysis:
  - Very simple approach
  - Used in early Gnutella
  - Entities are found only if they are within the search horizon
  - Lookups are fast
  - Huge message overhead in the network

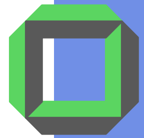
© 2009 Universität Karlsruhe (TH), System Architecture Group



Flooding

- You get the n.th neighbor within the n.th step
- How are the nodes distributed concerning the distance from the initiator of the query?
  - Tree: number of neighbors increases exponentially with the distance L, i.e. k^L, if k ist the node degree
  - Mesh: number of neighbors increases with L^{d-1}, if d is the dimension of the mesh

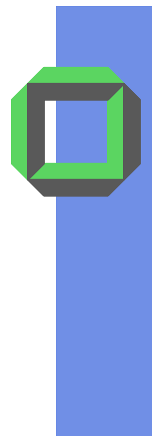
© 2009 Universität Karlsruhe (TH), System Architecture Group



Random Walk

- Principle of depth first search
- The query is only forwarded to one single neighbor
- A query is forwarded to a node at most once
- Each node knows its neighbors of first and second degree
- Analysis:
  - Few message overhead, but long searching
  - Complicated implementation
  - Complete lookup not always possible

© 2009 Universität Karlsruhe (TH), System Architecture Group

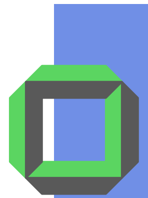


Distributed Hash-Tables (DHT)

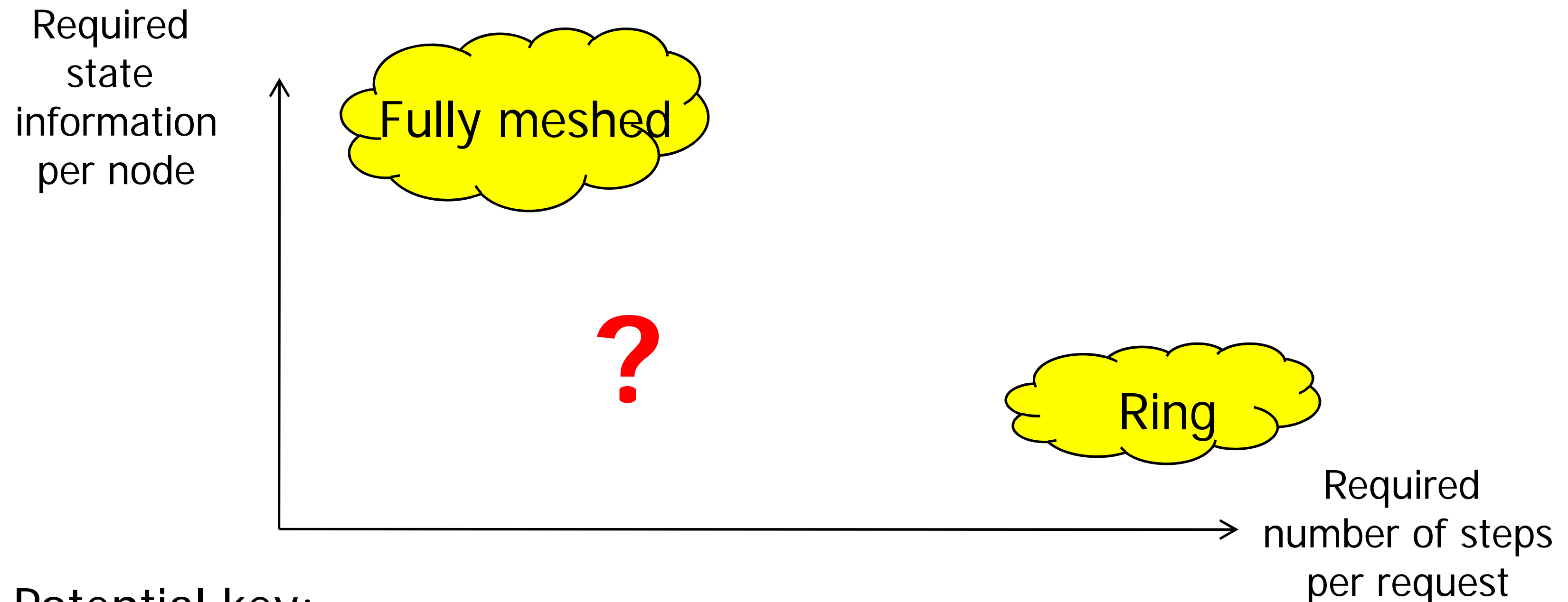
Hash key space, i.e. $[0,2^{128})$

How to locate entities according to their hash keys?

© 2009 Universität Karlsruhe (TH), System Architecture Group



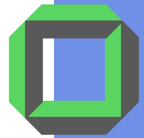
Tradeoff with DHT



Potential key:

- File name or title or composer etc. of some song
- hashed via an appropriate hash-function to a hash-key of length m
  m ~ 128, i.e. no unambiguousness, but collisions might be rare
- Nodes store the entities, whose hash keys fall into a given interval
- Nodes tend to be equally distributed over the hash-key space

© 2009 Universität Karlsruhe (TH), System Architecture Group



Distributed Hash Table

- Generic application interface:

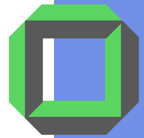
```
void publish (key_t, value_t)
value_t lookup(key_t)
```

- An extended functionality would be nice, e.g.

```
void withdraw(key_t)
```

- Key:
  - Can be a file name, a song title, ….
  - Are mapped via a hash-function to a 128 bit number, i.e. unambiguousness is not guaranteed, but collisions are rare

© 2009 Universität Karlsruhe (TH), System Architecture Group



Chord

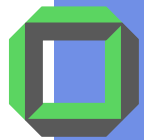
- All the hash-keys (the identifiers) and the node-ids are mapped to a logical ring
- Each node stores the entities with hash-keys between its ID and the ID of its predecessor
- When a new node enters Chord, the new node and his successor divide their responsibility for storing entities according to the ID of the new node
- When a node leaves Chord it gives all its stored entities to its successor

© 2009 Universität Karlsruhe (TH), System Architecture Group

Consistent Hashing

- Consistent hash function assigns each node and key an m-bit identifier.
- SHA-1 is used as a base hash function.
- A node's identifier is defined by hashing the node's IP address.
- A key identifier is produced by hashing the key (chord doesn't define this. Depends on the application).
  - ID(node) = hash(IP, Port)
  - ID(key) = hash(key)

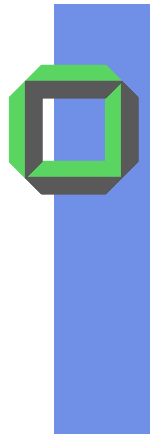
© 2009 Universität Karlsruhe (TH), System Architecture Group



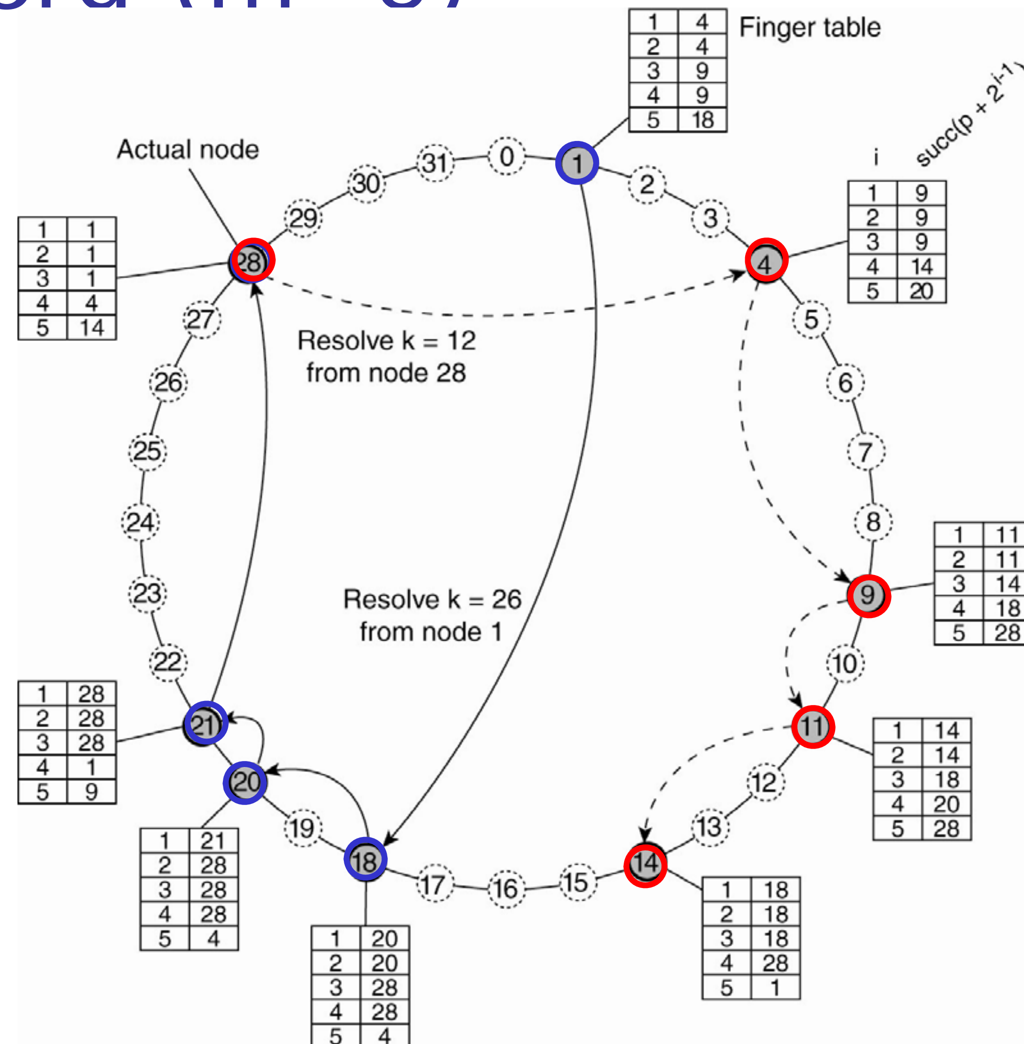
Chord

- m-bit identifier (m usually 128 or 160)
- Example m=5, i.e. set of nodes with
  - NIDs $\in \{0, 1, \ldots, 31\}$
- An entity with key k is managed by that node with the smallest identifier NID $\geq$ k
- This responsible node NID is called succ(k)
- Each node p maintains a finger table FT of m entries
- To lookup key k at node p, node p will forward request to node q with index j in FT_p:
  - $q = FT_p[j] < k \leq FT_p[j+1]$

© 2009 Universität Karlsruhe (TH), System Architecture Group



Example Chord (m=5)



Address of node 28 is returned to node 1 and the key k(=26) is resolved

The assumed logical ring is an overlay, i.e. often large hops can take place

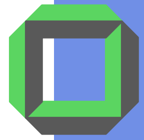
- Resolving **key 26** at node **1** in Chord
- Resolving key 12 at node 28 dashed lines

© 2009 Universität Karlsruhe (TH), System Architecture Group

Finger Tables

- Each node n′ maintains a routing table with up to m entries (which is in fact the number of bits in identifiers), called finger table.
- The ith entry in the table at node n contains the identity of the first node s that succeeds n by at least 2^{i-1} on the identifier circle.
- $s = successor(n+2^{i-1})$.
- s is called the ith finger of node n, denoted by n.finger(i)

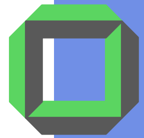
© 2009 Universität Karlsruhe (TH), System Architecture Group



Finger Tables

- A finger table entry includes both the Chord identifier and the IP address (+ port number) of the relevant node
- The first finger in each finger table is always the current immediate successor in the Chord ring
- As long as the Chord misses many „ring nodes“ some finger table entries point to the same successor, see the entries of the previous example

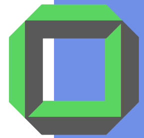
© 2009 Universität Karlsruhe (TH), System Architecture Group



Scalable Key Location

- Since each node has finger entries at power of two intervals around the identifier circle, each node can forward a query at least halfway along the remaining distance between the node and the target identifier. From this intuition follows a theorem:

  Theorem: With high probability, the number of nodes that must be contacted to find a successor in an N-node network is O(logN).



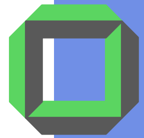
Analysis Chord

- Disadvantage: If a node storing a couple of entities crashes, it is hard to repair the routing structure
- However, the nodes are physically not neighbored, i.e. successors and predecessors in Chord are randomly distributed, there might be avoidable latencies if hops are long distance hops
- In proximity routing each node has multiple successors per finger table entry, e.g. containing the r most neighbored nodes

© 2009 Universität Karlsruhe (TH), System Architecture Group

Node Joins & Stabilization

- The most important thing is the successor pointer.
- If the successor pointer is ensured to be up to date, which is sufficient to guarantee correctness of lookups, then finger table can always be verified.
- Each node runs a “stabilization” protocol periodically in the background to update successor pointer and finger table.
- Besides the finger table each node in Chord also maintains a successor-list of its r nearest successors on the ring to support recovers from node crashes

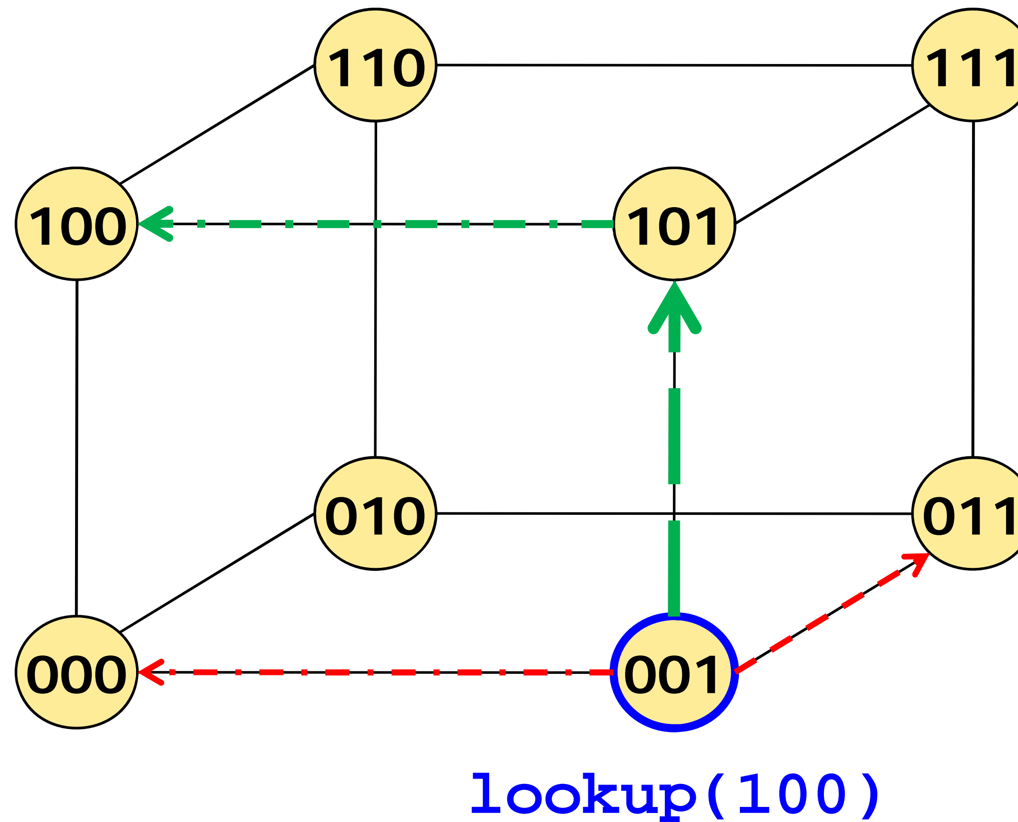


Pastry

- Keys and node-IDs are interpreted as a number sequence of the 2^k-number-system and mapped to a hypercube
- Each position of the number sequence corresponds to a coordinate of a dimension of the hyper cube
- Each node is responsible for the numbers with the same initial number-sequence
- Lookup queries are forwarded to those nodes with better fitting initial number-sequences

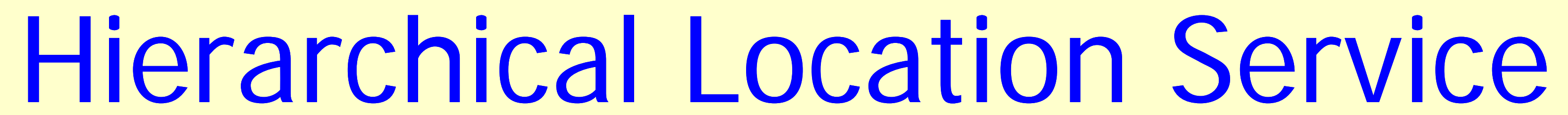
© 2009 Universität Karlsruhe (TH), System Architecture Group

Pastry Example

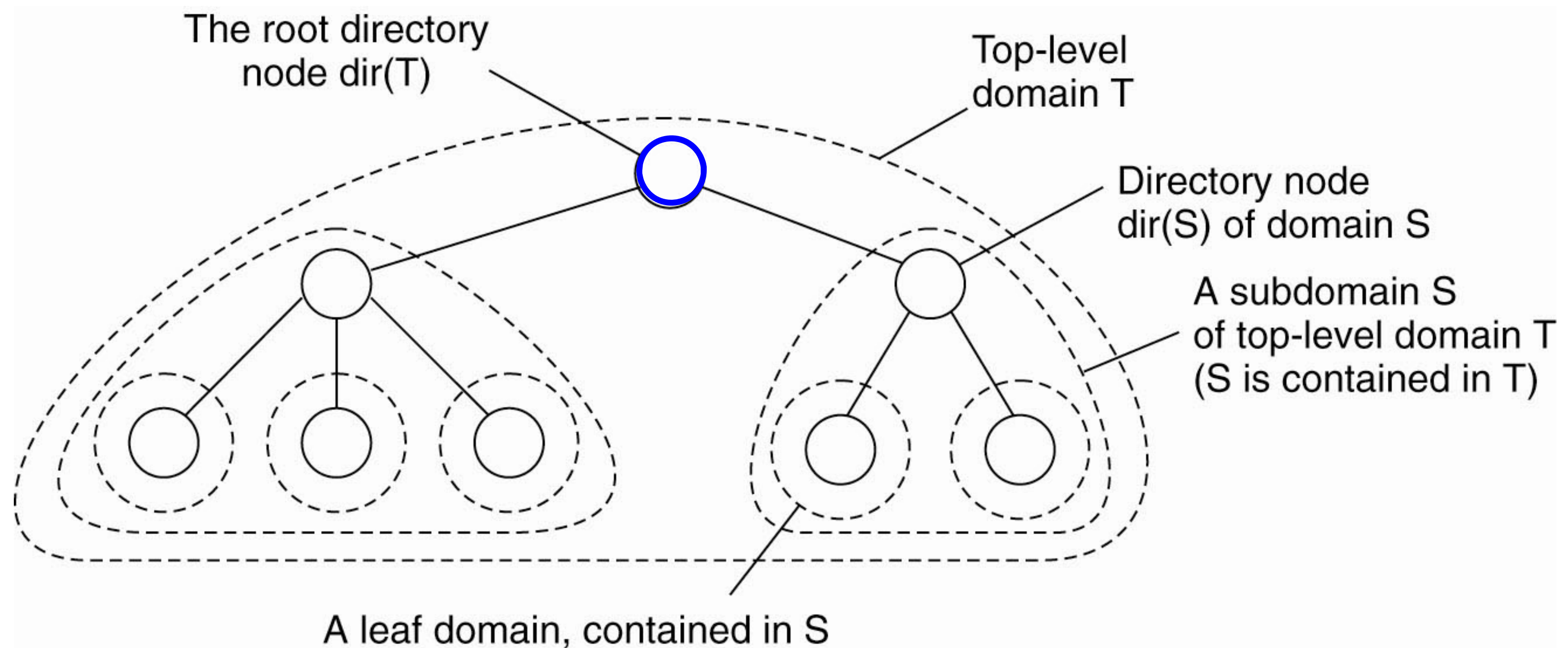


- Evaluate the major advantages and disadvantages of Chord versus Pastry

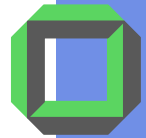
© 2009 Universität Karlsruhe (TH), System Architecture Group


Hierarchical Location Service

Hierarchical Approaches (1)



- Hierarchical organization of a location service into domains, each having an associated directory node
- Each domain D has a directory node dir(d) keeping track of all entities in that domain
- Lowest level domain (leaf) corresponds to a LAN domain



Location Record

- In a leaf domain directory service the location record contains

  <entity identifier, address of entity>

- In the higher directory services the location record contains:

  <entity identifier, pointer to next lower directory service containing entity or sub-domain with entity>

© 2009 Universität Karlsruhe (TH), System Architecture Group

Hierarchical Approaches (2)

Field with no data

Field for domain dom(N) with pointer to N

Location record for E at node M

M

N

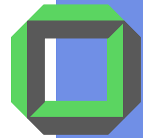
Location record with only one field, containing an address

Domain D1

Domain D2

- Each entity, currently located in a domain is represented by a location record in its directory node
- An example of storing information of a replicated entity having two addresses in different leaf domains

© 2009 Universität Karlsruhe (TH), System Architecture Group



Hierarchical Approaches (3)

Node knows about E, so request is forwarded to child

Node has no record for E, so that request is forwarded to parent

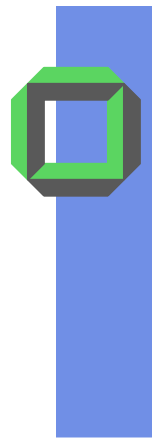
M

Look-up request

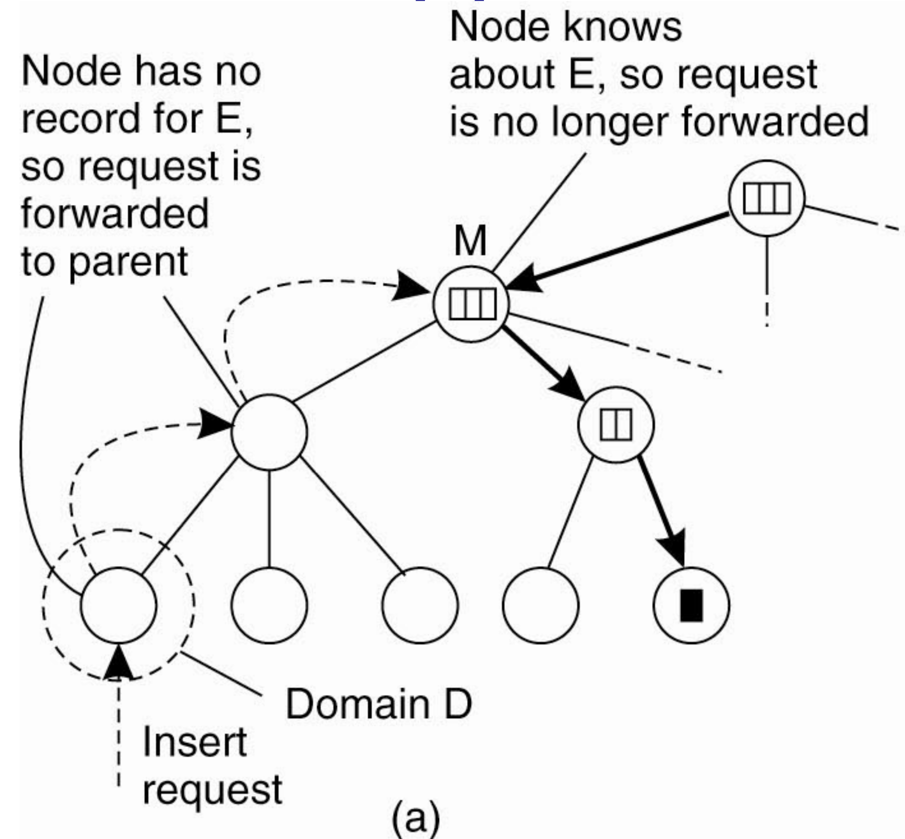
Domain D

- Looking up a location in a hierarchically organized location service.

© 2009 Universität Karlsruhe (TH), System Architecture Group

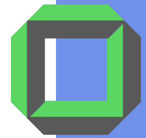


Hierarchical Approaches (4)

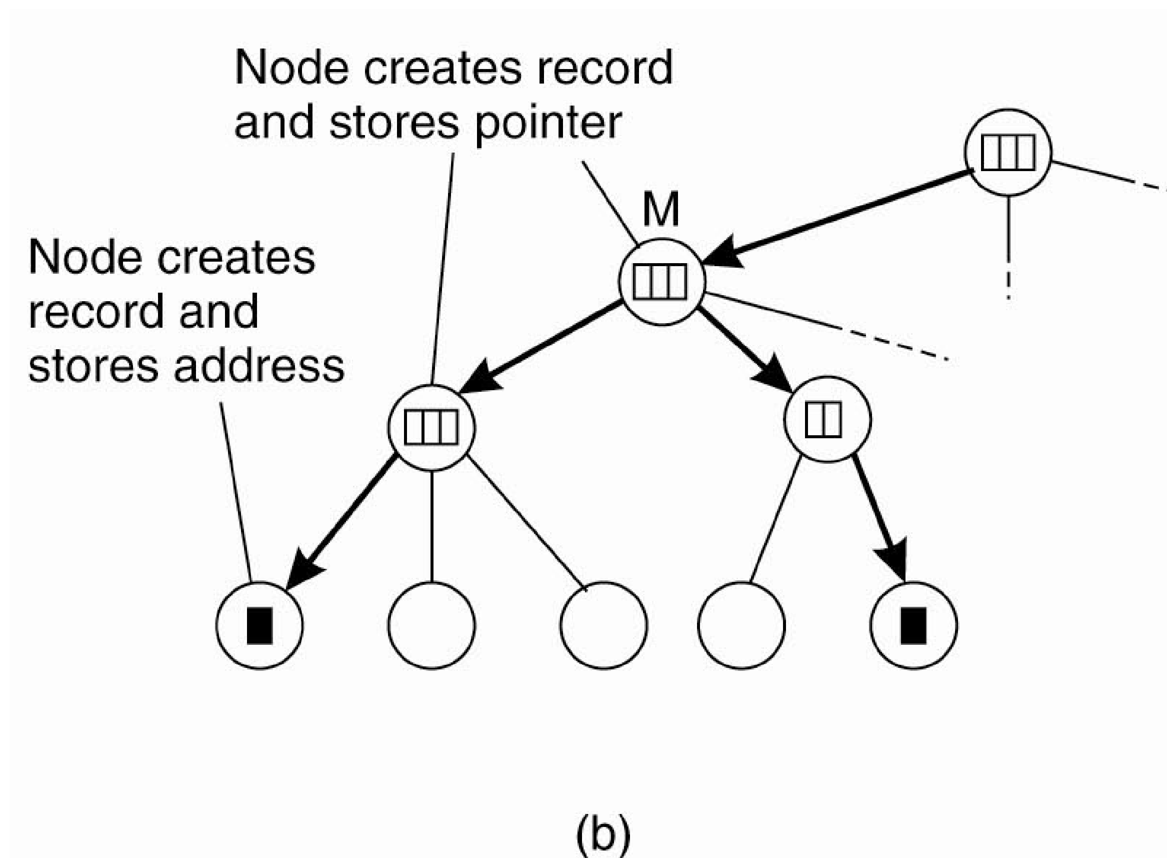


- A client has created a replica of E in domain D
- Client wants to insert address of D
- An insert request is forwarded to the first node M that knows about entity E

© 2009 Universität Karlsruhe (TH), System Architecture Group

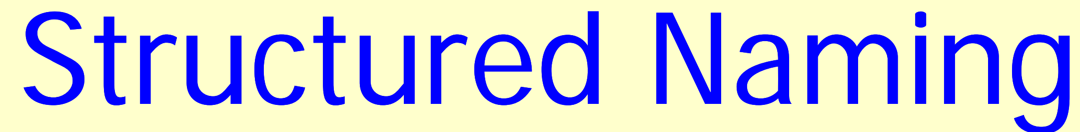


Hierarchical Approaches (5)



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

© 2009 Universität Karlsruhe (TH), System Architecture Group



Structured Naming

Mapping: structured names to addresses

Human friendly names, i.e. readable

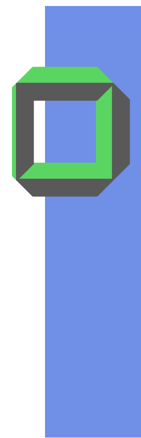
Examples. File names, host names

© 2009 Universität Karlsruhe (TH), System Architecture Group

Name Space

- Name space of structured names can be represented as a labeled, directed root-tree[1] with two types of nodes:
  - Leaf node represents a named entity (no outgoing edge)
  - Directory nodes have some outgoing edges, each labeled with a name
  - Directory node stores a (catalogue) table containing pairs of <edge label, node identifier>
- Root of graph is the starting point of name resolution
  - A global name denotes the same entity no matter where this name in the distributed system is used
  - A local name depends on the location where it is used

[1]As long as there are no aliases



Hierarchical Name Space

node 1

files

john

imagine

paul

yesterday

users

node 2

files

users

Hold On: 33344
How do you sleep: 81111
Imagine: 4711
Jealous Guy: 888888
Mother: 88899
Woman: 10555
Working Class Hero: 1999

Example path name:

/node1/files/john/imagine

© 2009 Universität Karlsruhe (TH), System Architecture Group

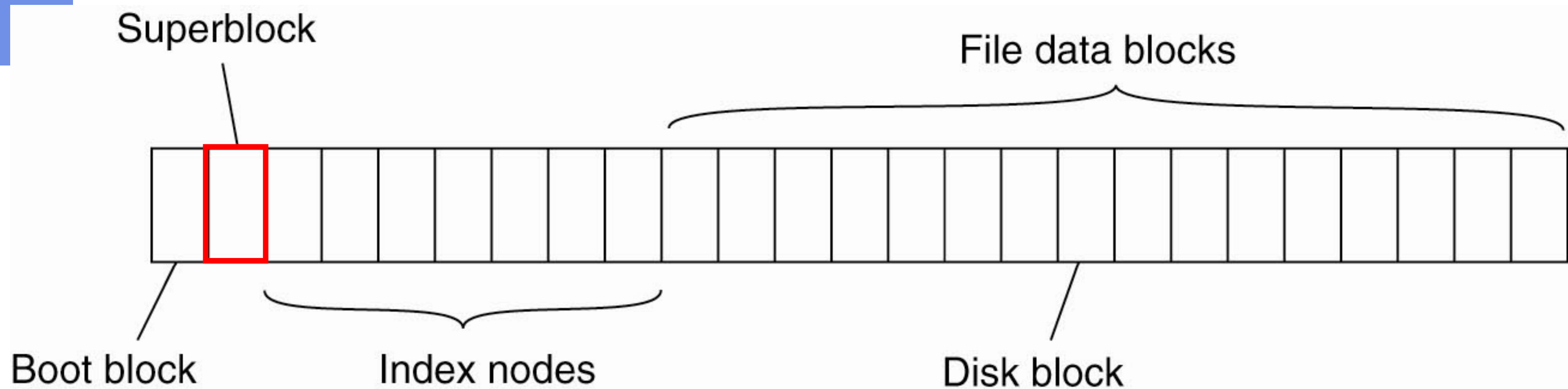
Name Resolution

- Given a pathname:

  N:<$label_1$, $label_2$, …,$label_n$>

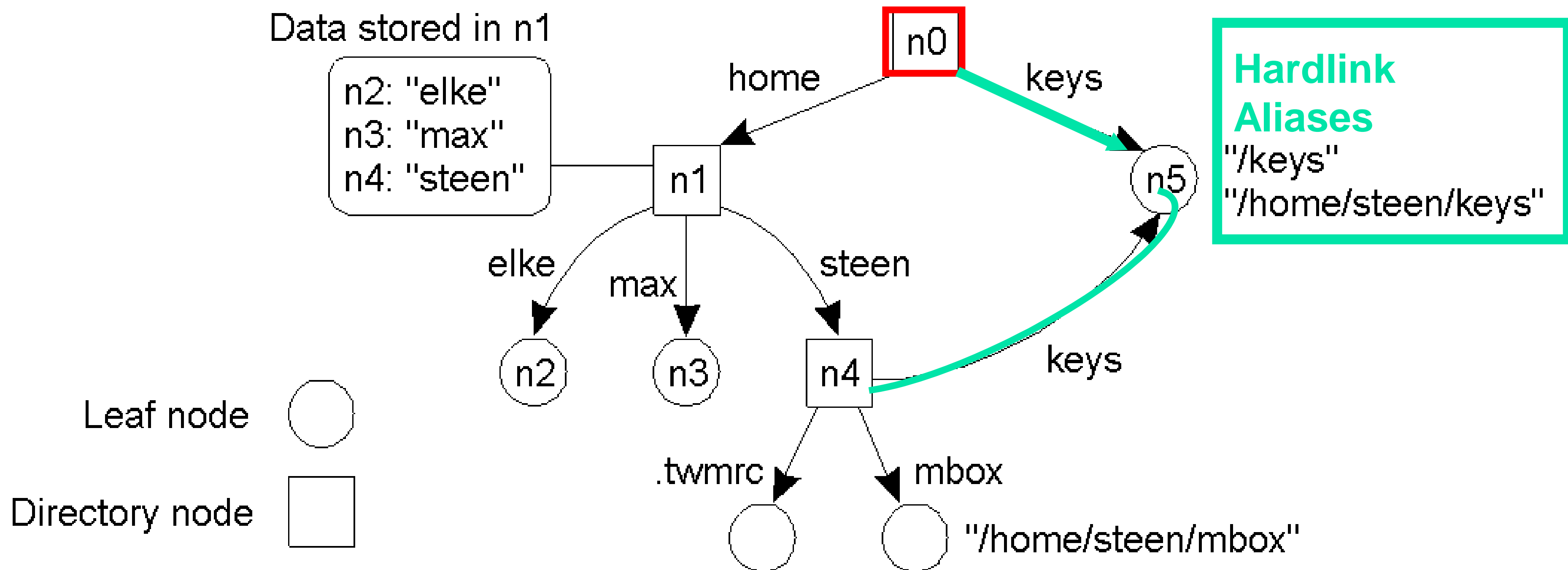
- Name resolution starts at node N of the naming graph, looking up if $label_1$ is part of its directory table
- If so, it returns the identifier of the directory containing $label_2$, …
- Resolution stops at the last node returning the content of that node (e.g. the inode of a file)

Name Spaces (2)



- General organization of the UNIX FS implementation on a logical disk of contiguous disk blocks
- Inode 0 reserved for root directory

Name Spaces (1)



- A general naming graph[1] with a single root node n_0
- Path PN := <$label_1$, $label_2$, ... , $label_n$>

[1] See Unix or Linux file system

Linking and Mounting

- Linking: creating aliases. Two approaches:
  - Allow multiple namespace paths to reference the same leaf (hard link)
    - take care not to create cycles
  - Special leaf nodes that contain a path name (soft link)
    - name resolution continues after appending the new name to the original name
- Mounting: Allows a leaf node to refer to a directory in a different name space (a mount point)
  - name resolution continues in new name space

© 2009 Universität Karlsruhe (TH), System Architecture Group

Name Space (2)

Data stored in n1

n2: "elke"
n3: "max"
n4: "steen"

n0
home
keys
n1
n5 "/keys"
elke
max
steen
n2
n3
n4
Leaf node
Directory node
.twmrc
mbox
keys
Data stored in n6
"/keys"
n6 "/home/steen/keys"

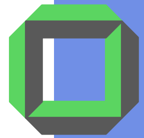
- Concept of a symbolic link explained in a naming graph

© 2009 Universität Karlsruhe (TH), System Architecture Group

Name Resolution

- *Where to start?*
- Needed a closure mechanism
- Examples:
  - Inode 0 in a Unix FS at a specific byte offset on the root partition
  - Environment variables in Unix
    - The closure mechanism of all environments variables enables where to start with the look up
    - Each PCB contains a table of all current valid environment variable

© 2009 Universität Karlsruhe (TH), System Architecture Group

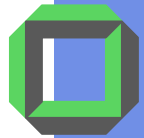


Closure Mechanism

Name resolution can only take place if you know how and where to start

Examples:

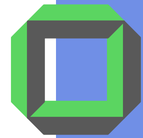
1. Name resolution in a Unix FS is easy, because the root directory is first inode on logical disk, i.e. **`inode 0`** representing that FS.
2. 497216083837 -a commonly used string- but without knowing that the above string represents a phone-number, you can not use it ⇒ convention for submitting a phone number (+49)-721-608-3837 is Gerd's phone number in his office



Combined Name Space

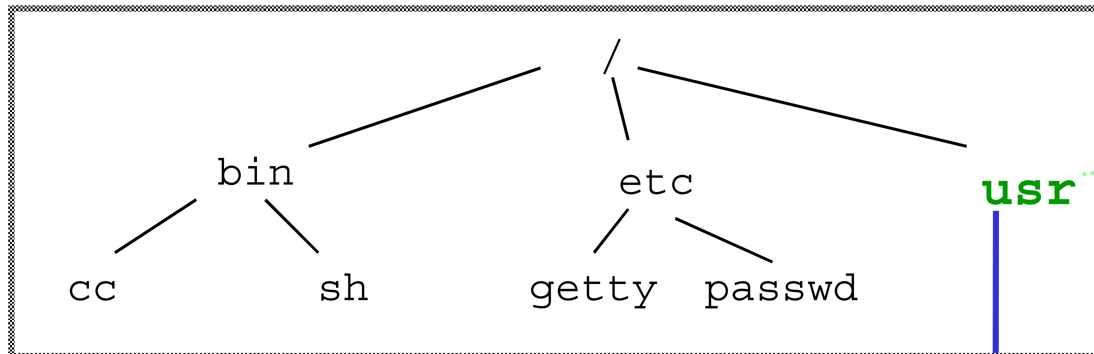
- Name space may be totally or partly be incorporated within another name space
  - e.g. mounting of Unix file systems
- Name can be root directory of another name space
  - With potential different separators (i.e. / instead of .)
  - Implemented via a problem oriented lookup service

© 2009 Universität Karlsruhe (TH), System Architecture Group



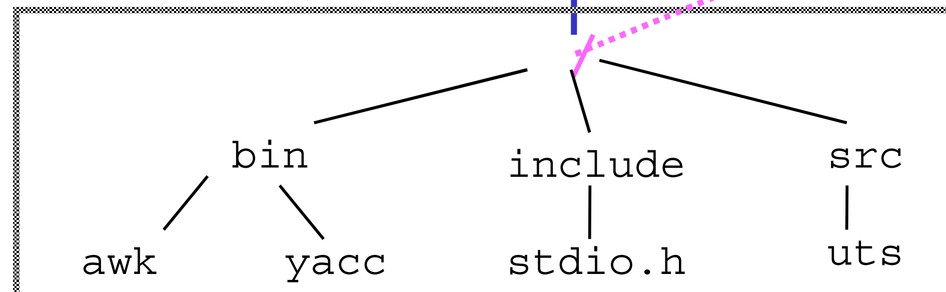
Linking of Name Spaces (Mounting)

root file system



mount-point

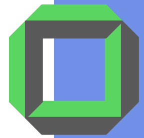
mounting-point



mountable file system

- Name resolution can be used to combine different name spaces transparently

© 2009 Universität Karlsruhe (TH), System Architecture Group



Linking of Distr. Name Spaces

Given n>1 name spaces on different machines, i.e. each name space may belong to a different server:

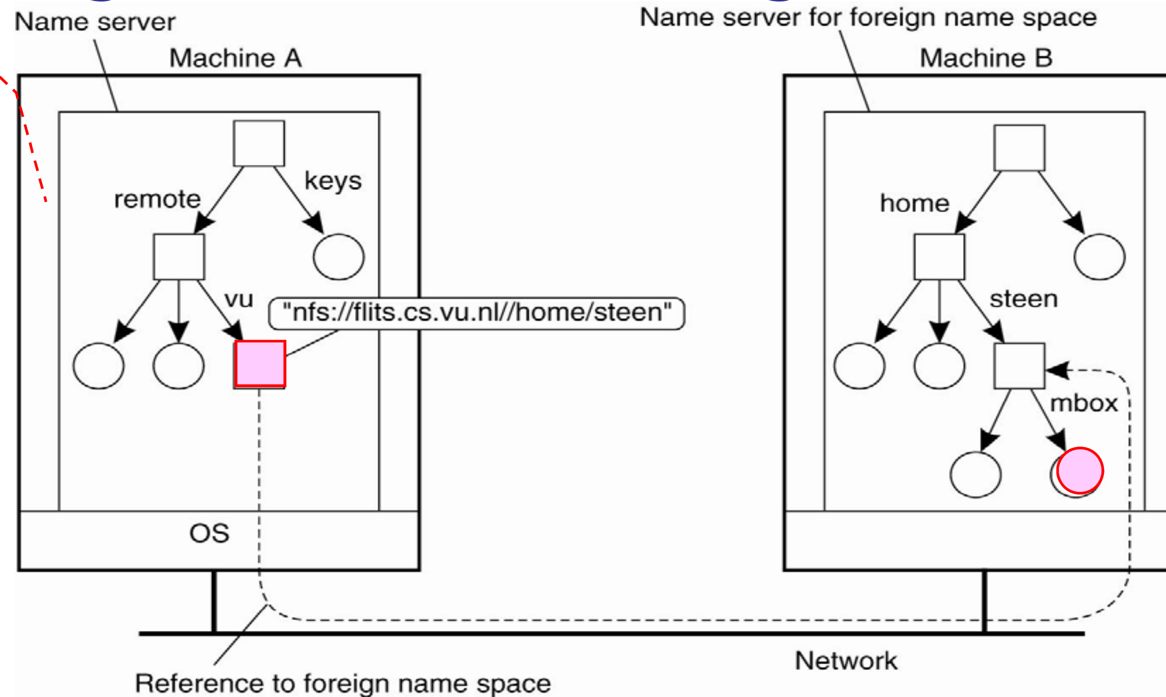
<u>Goal:</u>
To mount a foreign name space NS_2 into the name space NS_1, the mount point in NS_1 must contain:

1. Name of an access protocol for server S_2
2. Name of the server S_2
3. Name of the mounting point in the foreign name space N2

Each of the names must to be resolved, e.g. using URLs

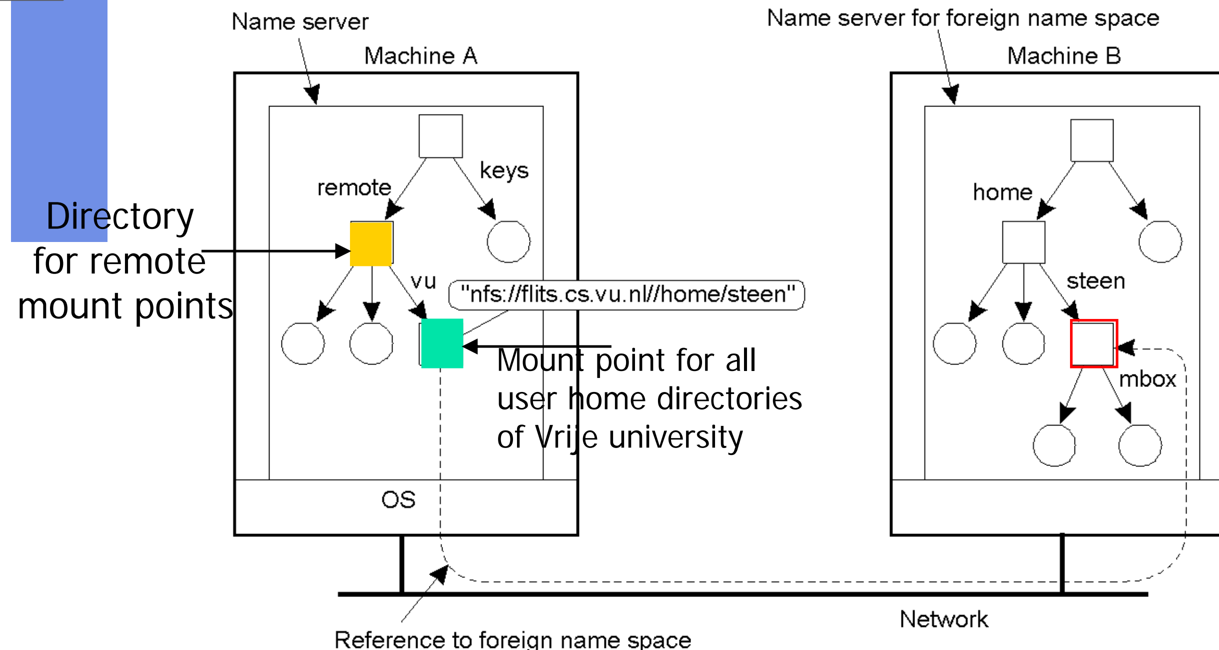
© 2009 Universität Karlsruhe (TH), System Architecture Group

Linking and Mounting



- Mounting remote name spaces via a specific access protocol (NFS), i.e. on both machines NFS is configured
- Client wants to access his mailbox located on machine B
- Client offers pathname **`/remote/vu/mbox`** at machine A
- Name resolution starts at root of machine A

Linking and Mounting (2)



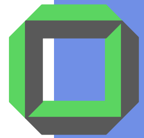
An authorized user on Machine A uses:
`/remote/vu/mbox`

What happens?

1. On A `/remote/vu` is resolved
2. Reply with the result: `nfs://flits.cs.vu.nl/home/steen`
3. Client contacts via B, i.e. `flits.cs.vu.nl`
4. B resolves `/home/steen/mbox`

- Both machines contain Sun's NFS
- Client specifies "`nfs://flits.cs.vs.vl/home/steen`" in the directory `/remote/vu` to be used as an NFS URL
- Server name `flits.cs.vs.nl` is resolved using DNS
- `/home/steen` is resolved using the remote server on machine B

© 2009 Universität Karlsruhe (TH), System Architecture Group



Linking & Mounting (3)

- Linking remote name server via mount points is one way
- Alternative:
  - Insert a new root-node above the two roots of the current name spaces (file systems)
  - Done by GNS of DEC

© 2009 Universität Karlsruhe (TH), System Architecture Group