Naming: Intro and Chord

1 December 2024
Lecture 5

Some Slide Credits: Maarten van Steen

Topics for Today

- Naming
  - Theory: Naming Entities
  - Flat Naming
    - DHTs – Chord
    - HLS

Source: TvS 6.1 - 6.5

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 from 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 the following properties:

P1 Each identifier refers to at most one entity

P2 Each entity is referred to by at most one identifier

P3 An identifier always refers to the same entity (prohibits reusing an identifier)

Observation: An identifier need not necessarily be a pure name, i.e., it may have content.

So Far

- Naming
  - Theory: Naming Entities
  - Flat Naming
    - DHTs – Chord
    - HLS

Flat Naming

Problem: Given an essentially **unstructured name** (e.g., an identifier), how can we locate its associated **access point**?

Simple solutions (broadcasting)

Home-based approaches

Distributed Hash Tables (structured P2P)

Hierarchical location service

Simple Solutions

Broadcasting: Simply broadcast the ID, requesting the entity to return its current address.

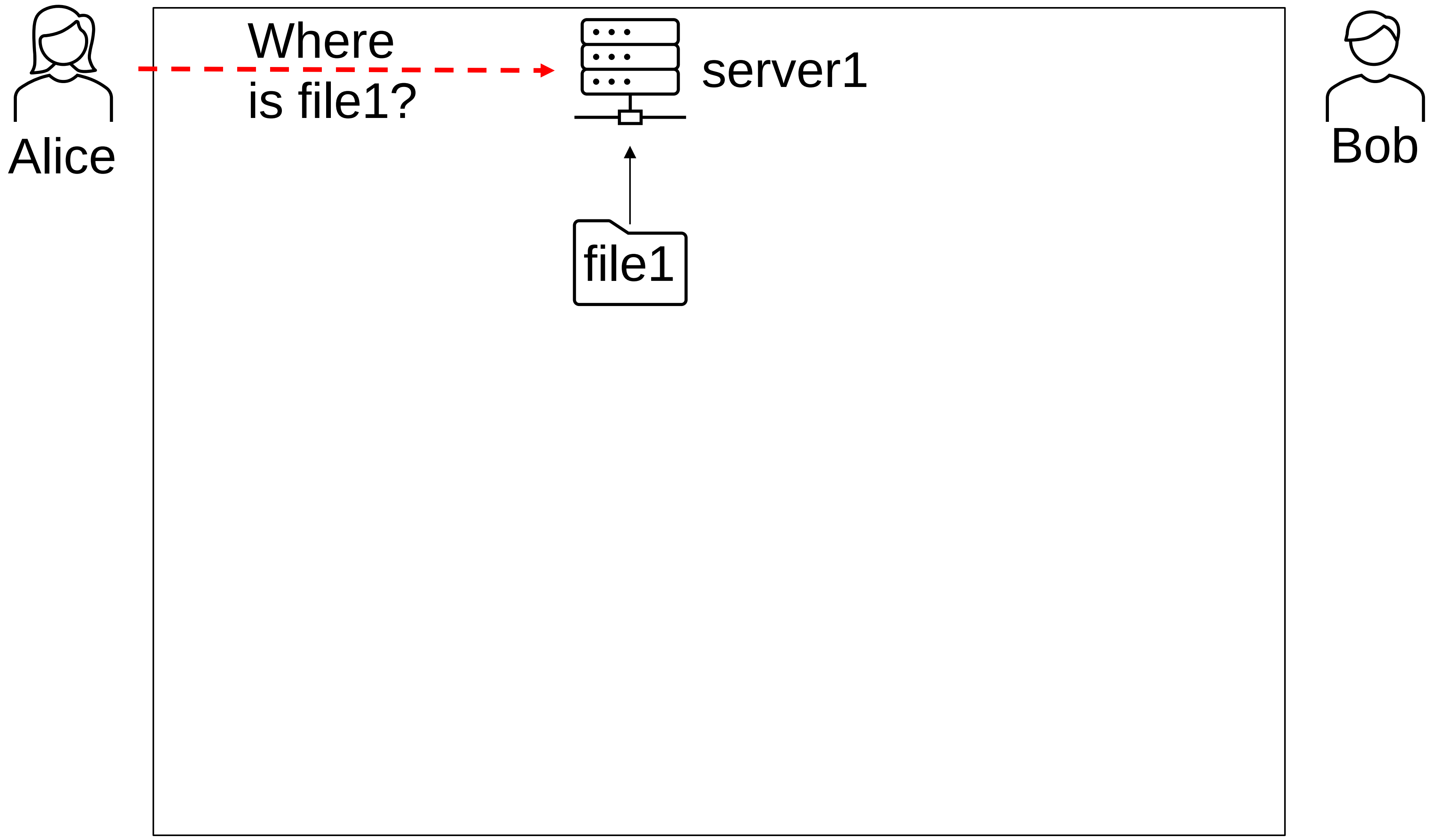
- Can never scale beyond local-area networks (think of ARP/RARP)
- Requires all processes to listen to incoming location requests

Forwarding pointers: Each time an entity moves, it leaves behind a pointer telling where it has gone to.

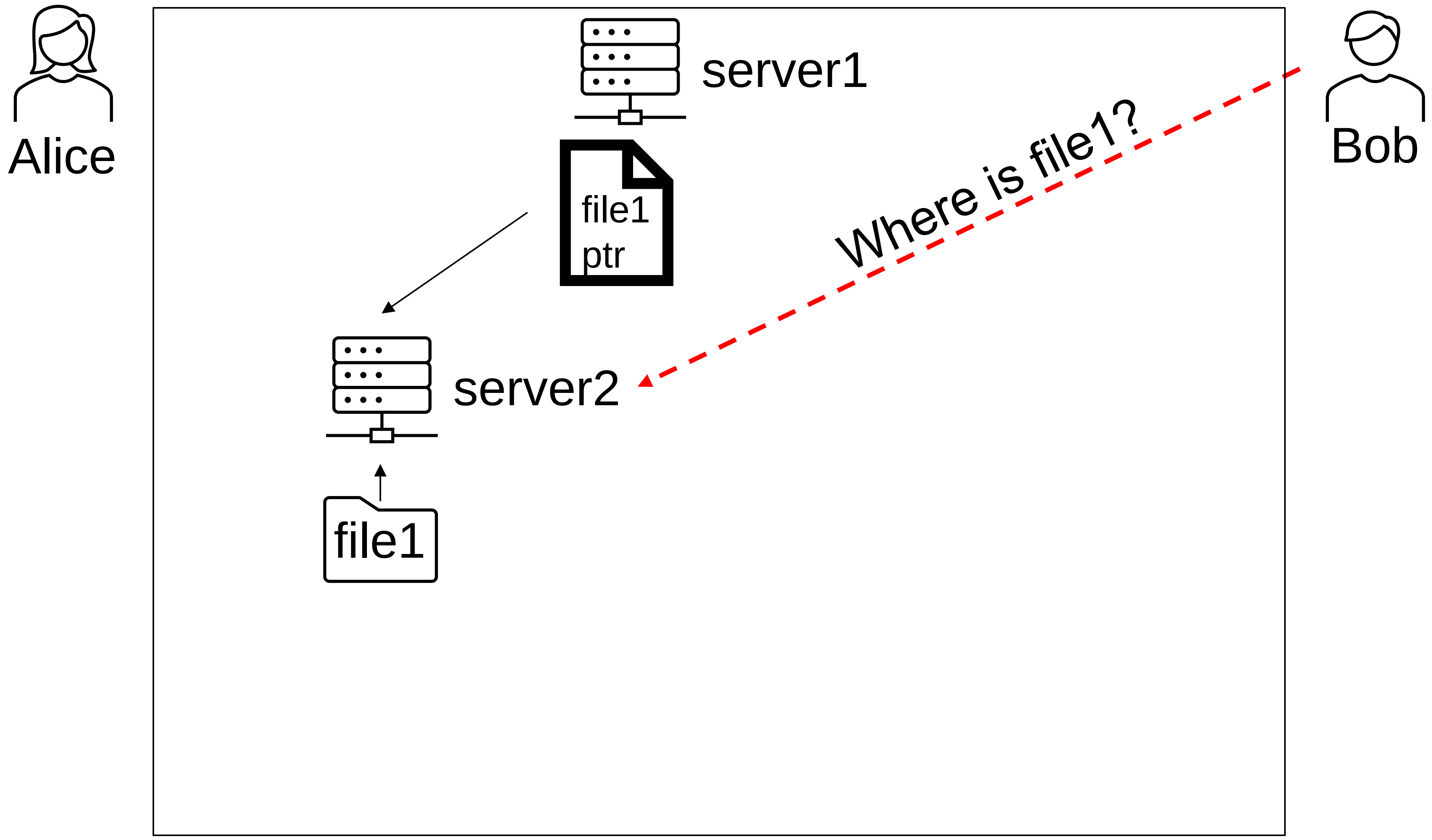
- Dereferencing can be made entirely transparent to clients by simply following the chain of pointers
- Update a client’s reference as soon as present location has been found
- Geographical scalability problems:
  - Long chains are not fault tolerant
  - Increased network latency at dereferencing

Essential to have separate chain reduction mechanisms

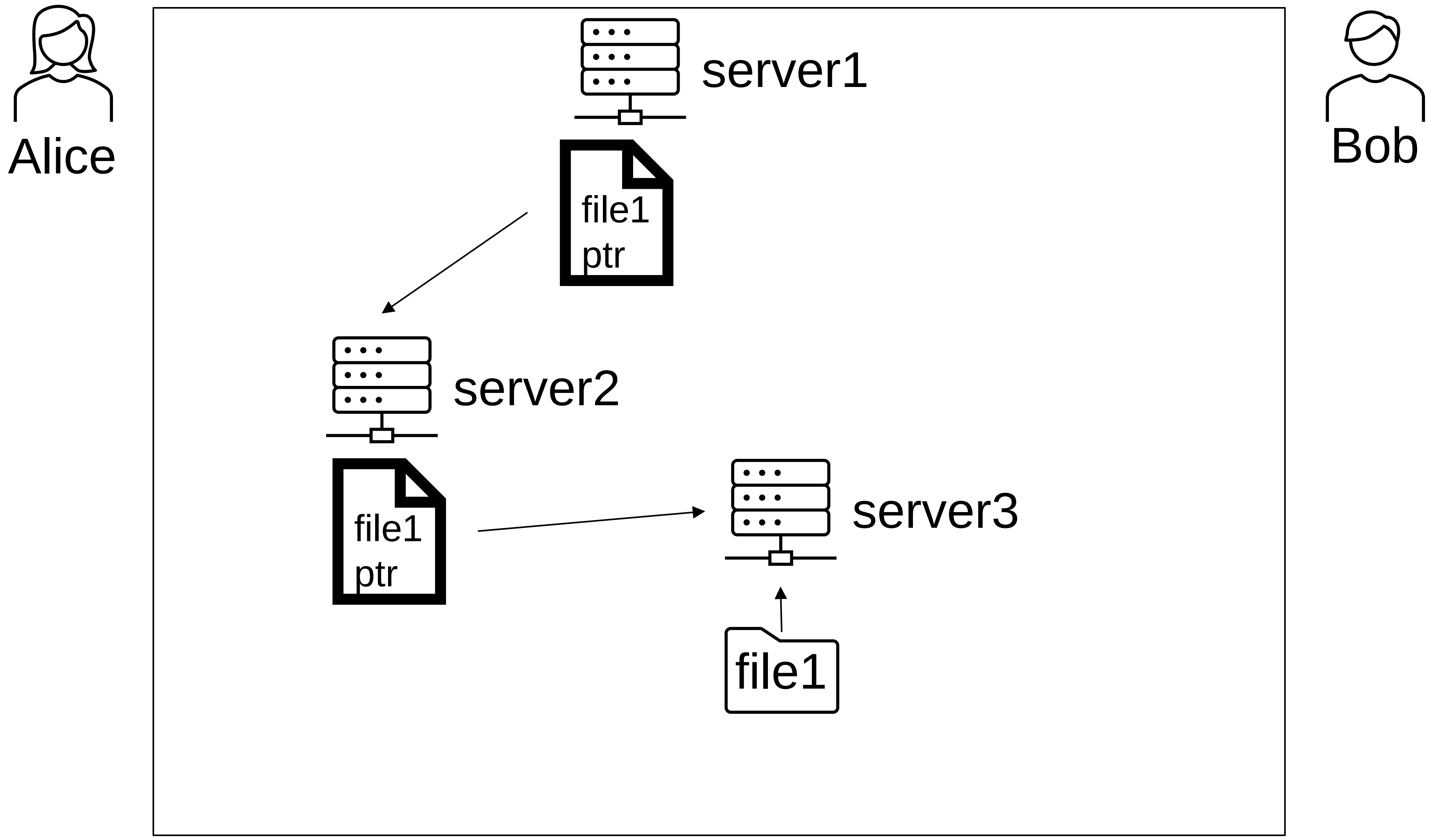
Forwarding Pointers



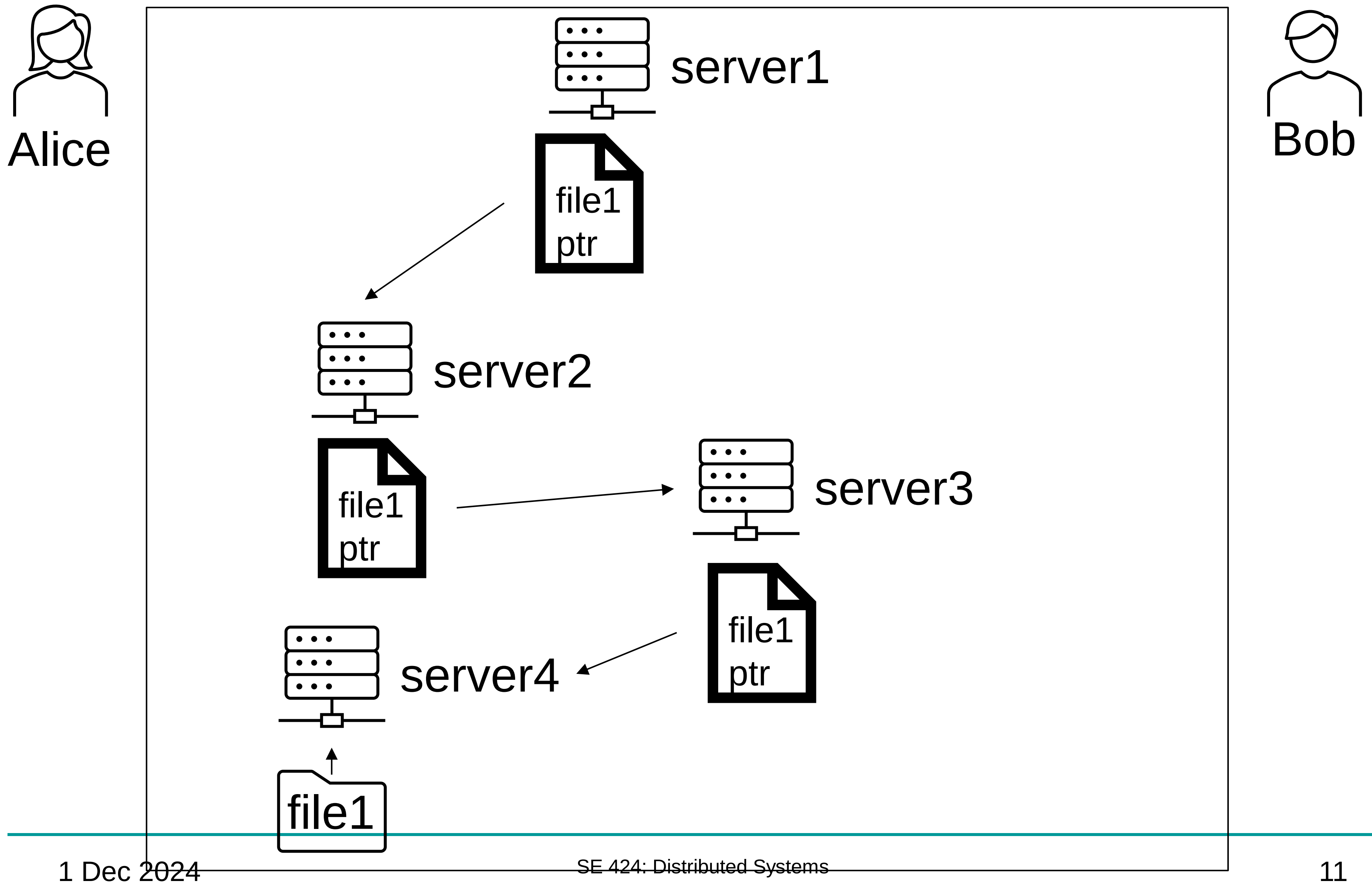
Forwarding Pointers



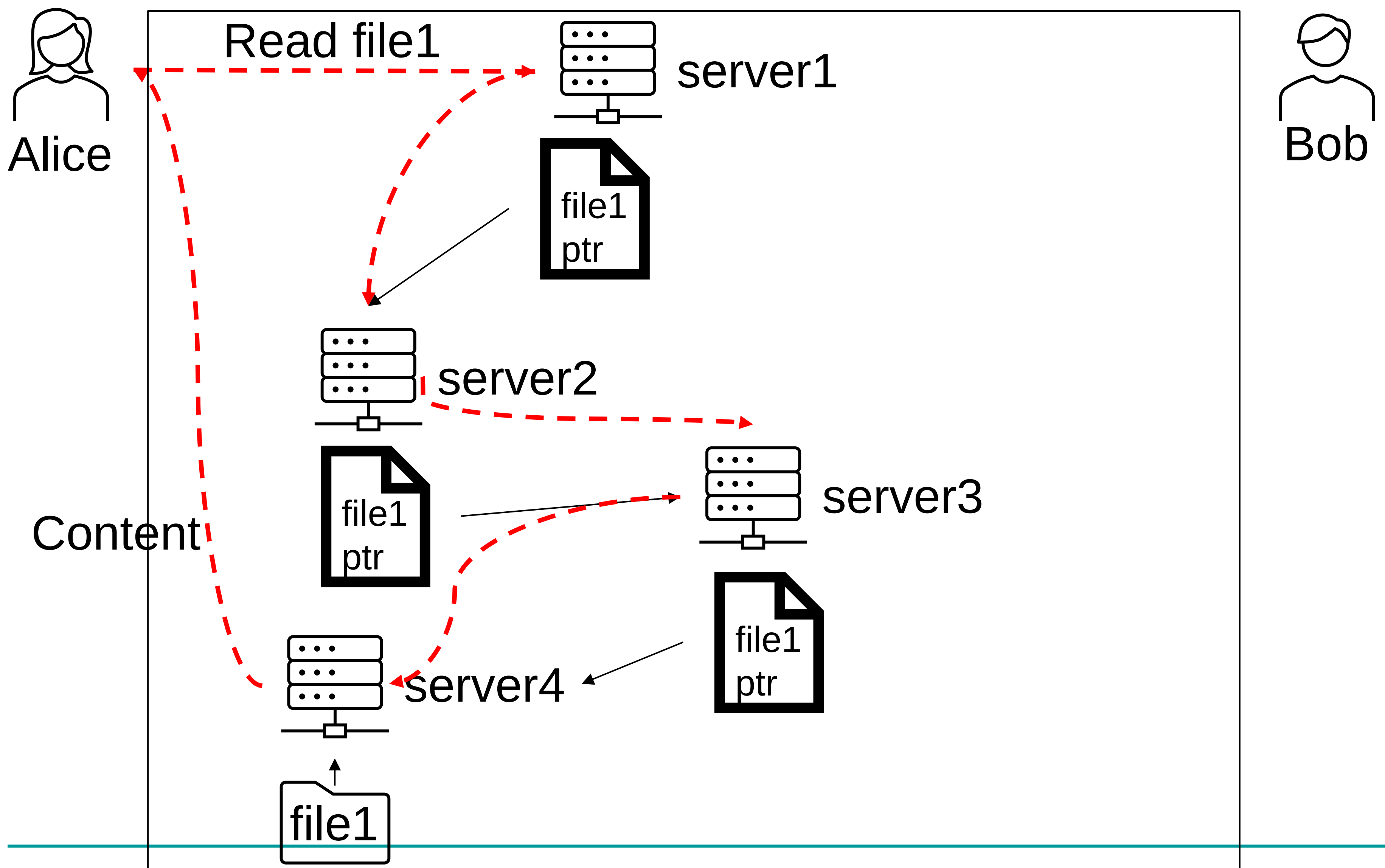
Forwarding Pointers



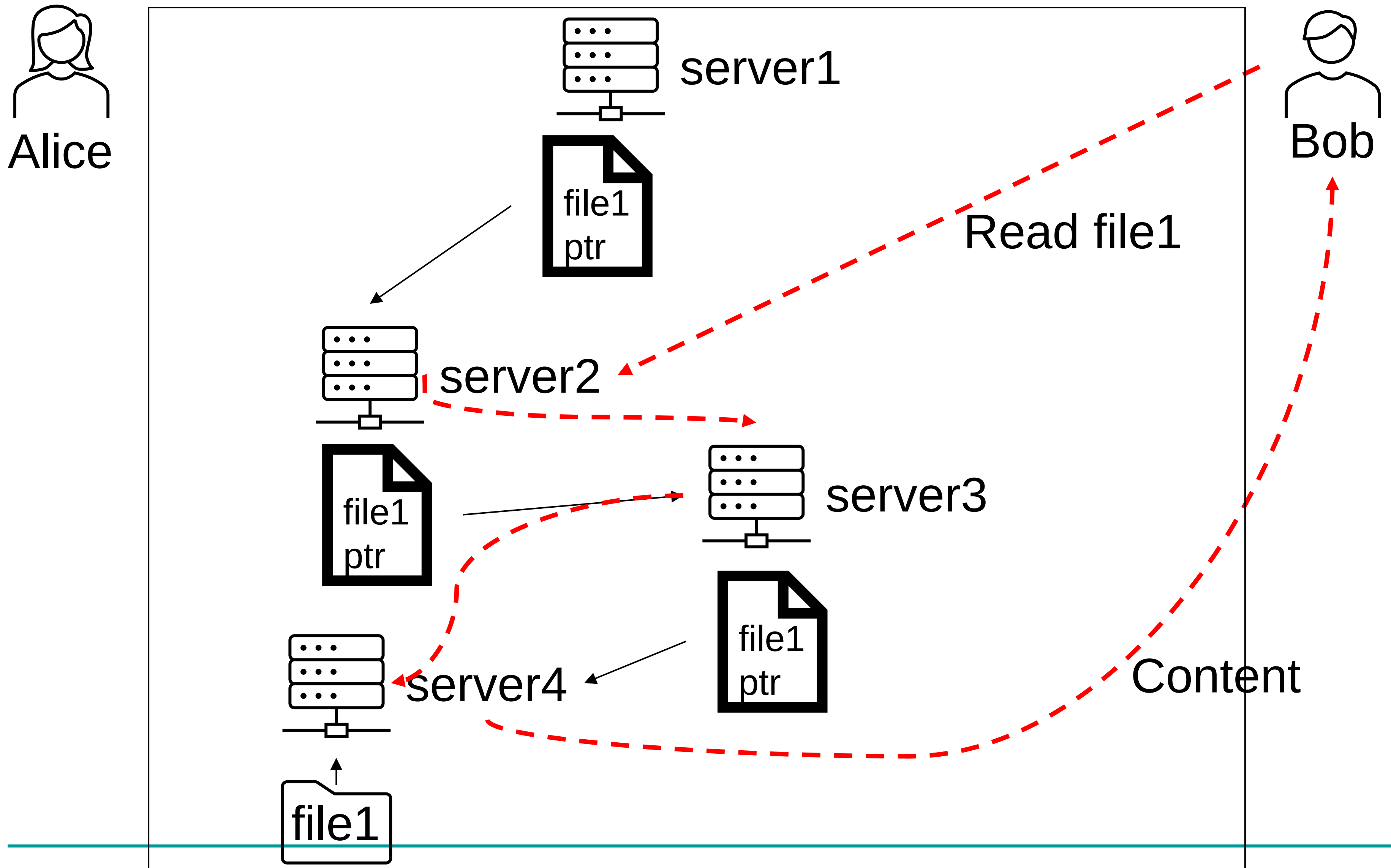
Forwarding Pointers



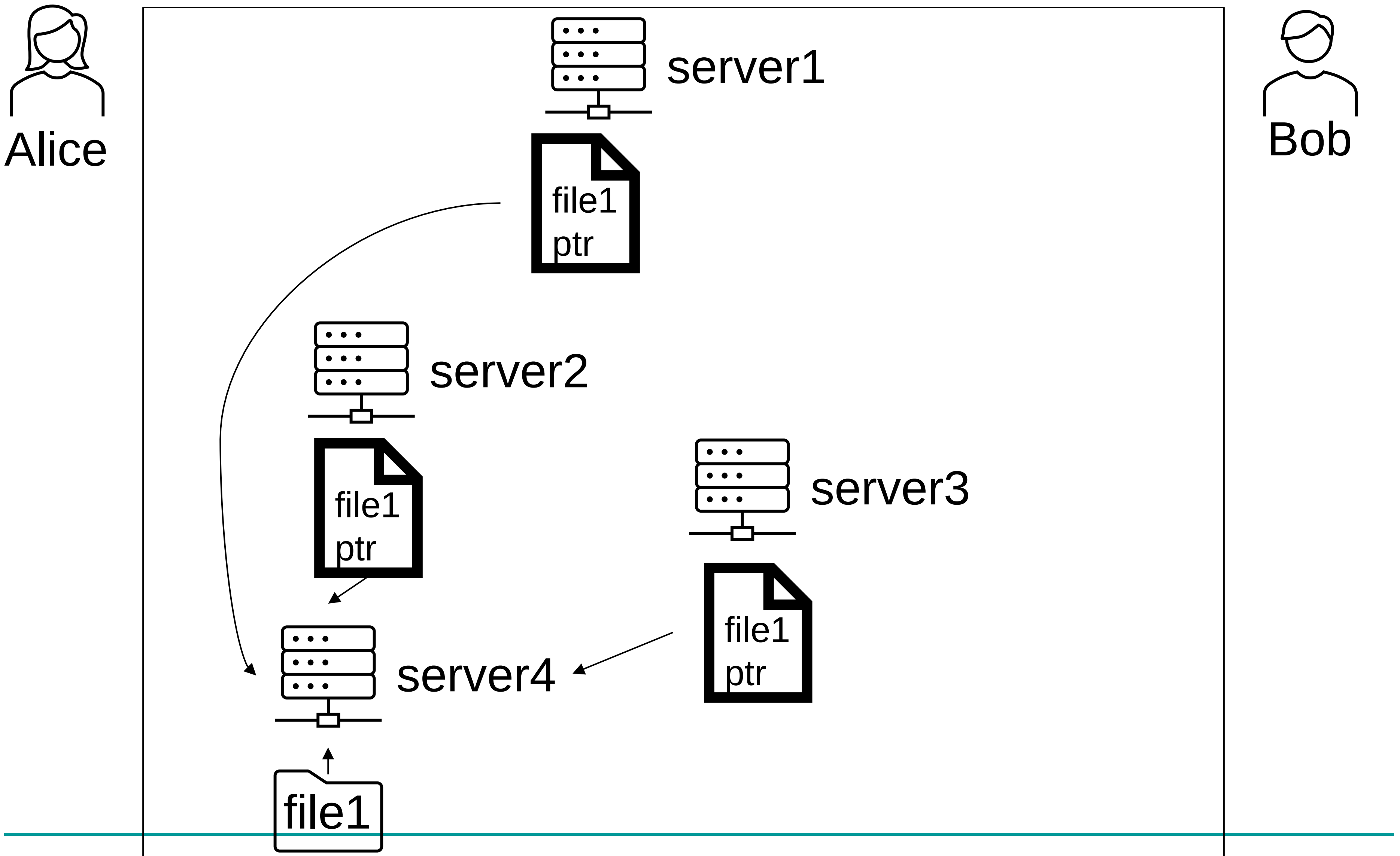
Forwarding Pointers



Forwarding Pointers



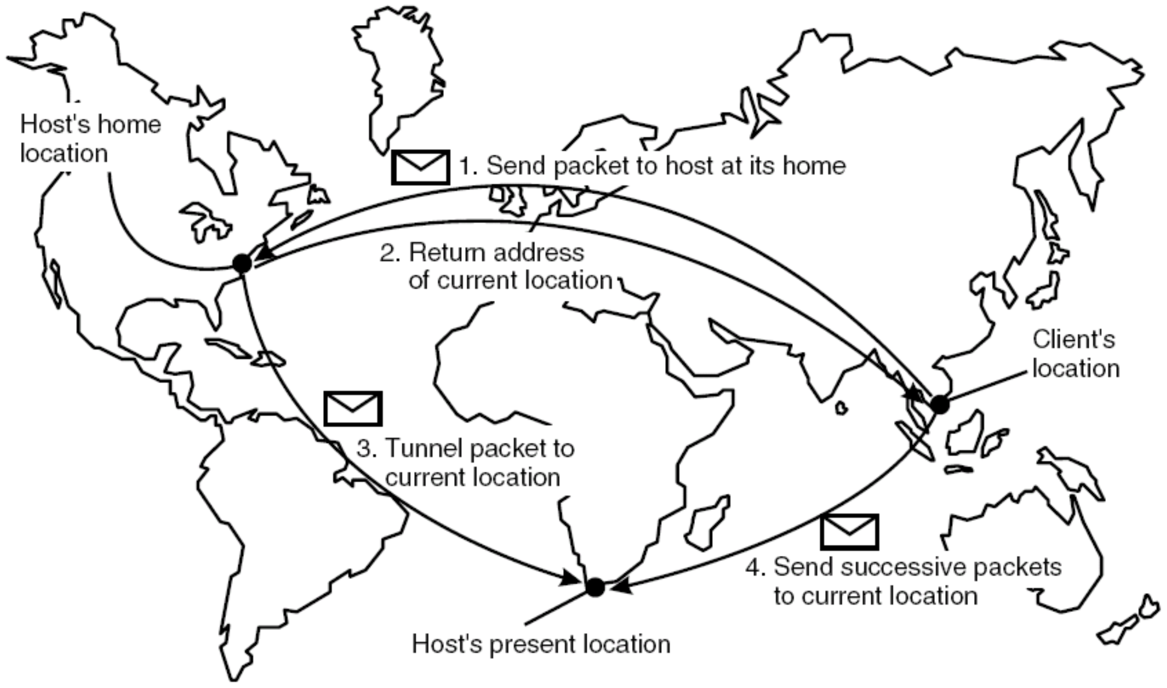
Shorten Forwarding Pointers



Home-Based Approaches (1/2)

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

- An entity's home address is registered at a naming service
- The home registers the foreign address of the entity
- Clients always contact the home first, and then continues with the foreign location



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

Home-Based Approaches (2/2)

Two-tiered scheme: Keep track of **visiting** entities:

- Check local visitor register first
- Fall back to home location if local lookup fails
- Example: Cellular network HLR + VLR

Problems with home-based approaches:

- The home address must be supported as long as the entity lives.
- The home address is fixed, which means an unnecessary burden when the entity permanently moves to another location
- Poor geographical scalability (the entity may be next to the client)

Question: How can we solve the “permanent move” problem?

Distributed Hash Tables

Example: Consider the organization of many nodes into a logical ring (Chord)

- Each node is assigned a random m-bit identifier.
- Every entity is assigned a unique m-bit key.
- Entity with key k falls under jurisdiction of node with smallest $id \geq k$ (called its successor).

Nonsolution: Let node id keep track of $succ\ (id)$ and start linear search along the ring.

DHTs: Finger Tables (1/3)

Each node *p* maintains a **finger table** $FT_p[]$ with at most *m* entries:

$$FT_p[i] = succ(p + 2^{\{i-1\}})$$

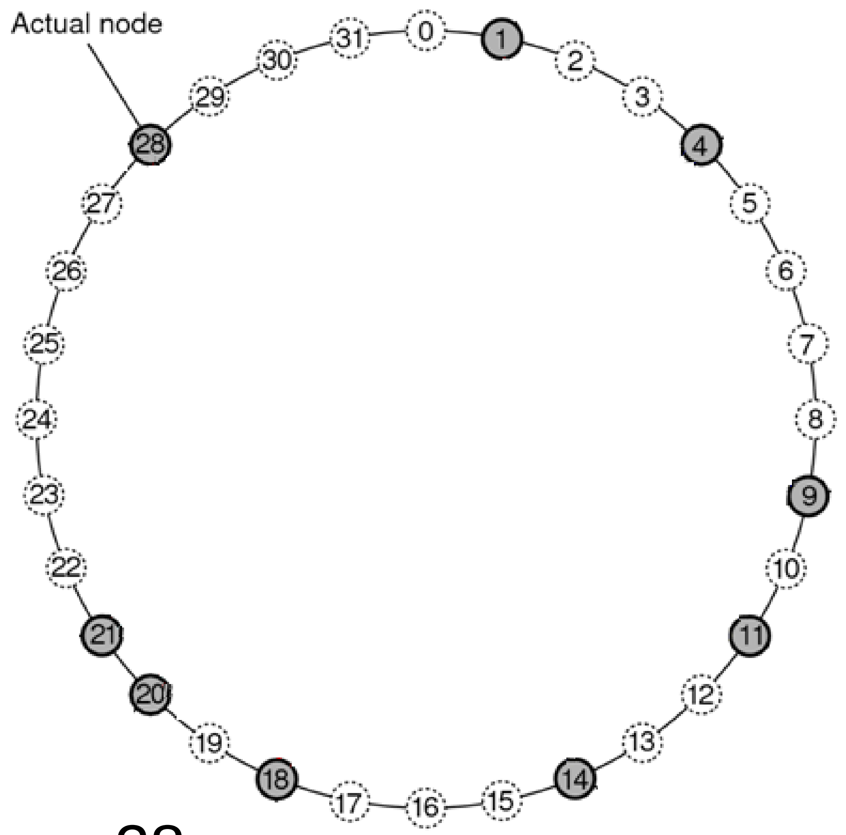
Note: $FT_p[i]$ points to the first node succeeding p by at least $2^{\{i-1\}}$.

To look up a key k, node p forwards the request to node with index j satisfying

$$q = FT_p[j] \leq k < FT_p[j+1]$$

- If $p < k < FT_p[1]$ the request is also forwarded to $FT_p[1]$.

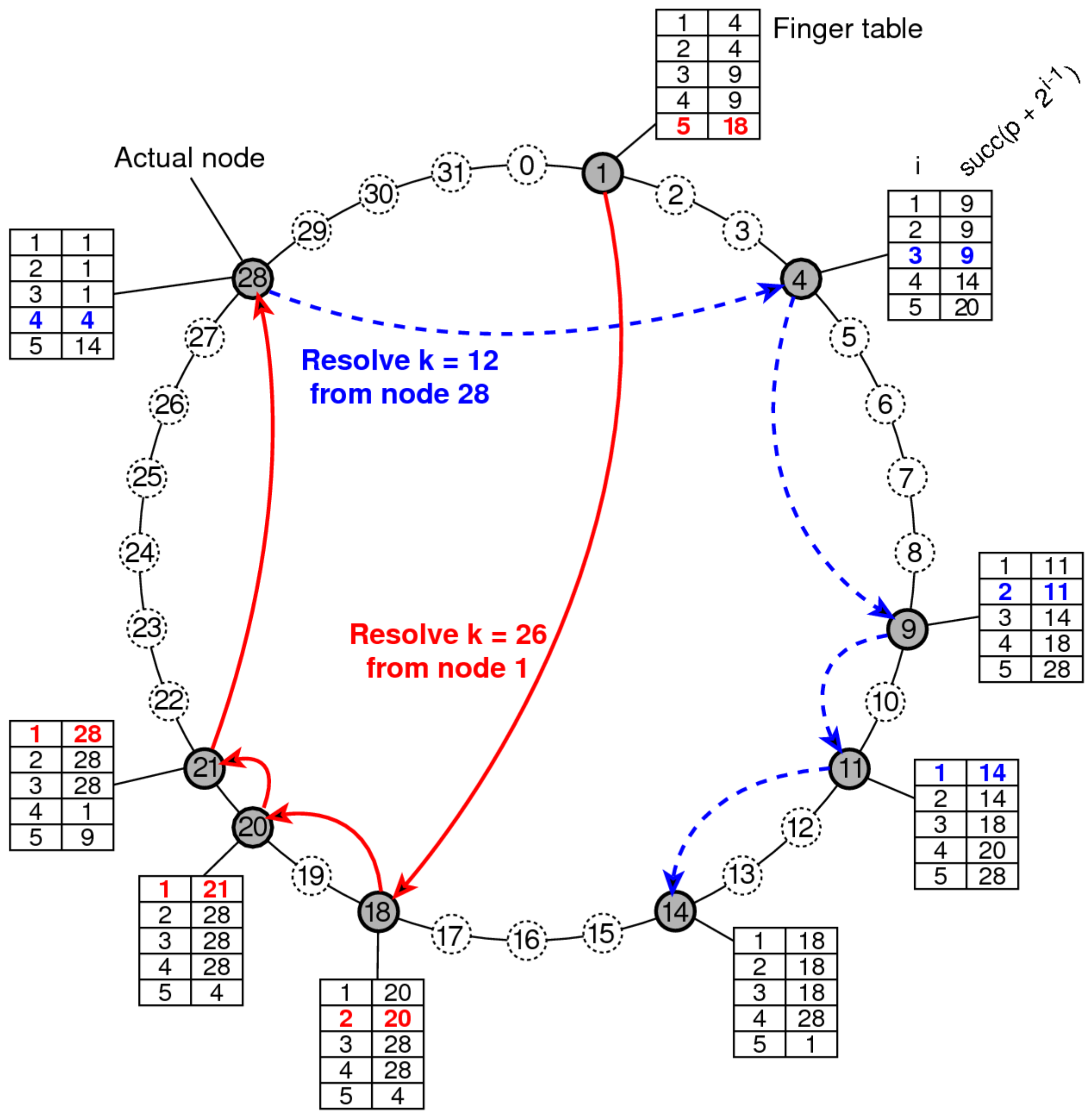
DHTs: Finger Tables (2/3)



Examples:

1. Resolve 12 from 28
2. Resolve 26 from 1

DHTs: Finger Tables (3/3)



1	4
2	4
3	9
4	9
5	**18**

i	succ(p + 2^{i-1})
1	9
2	9
3	**9**
4	14
5	20

1	11
2	**11**
3	14
4	18
5	28

1	**14**
2	14
3	18
4	20
5	28

1	18
2	18
3	18
4	28
5	1

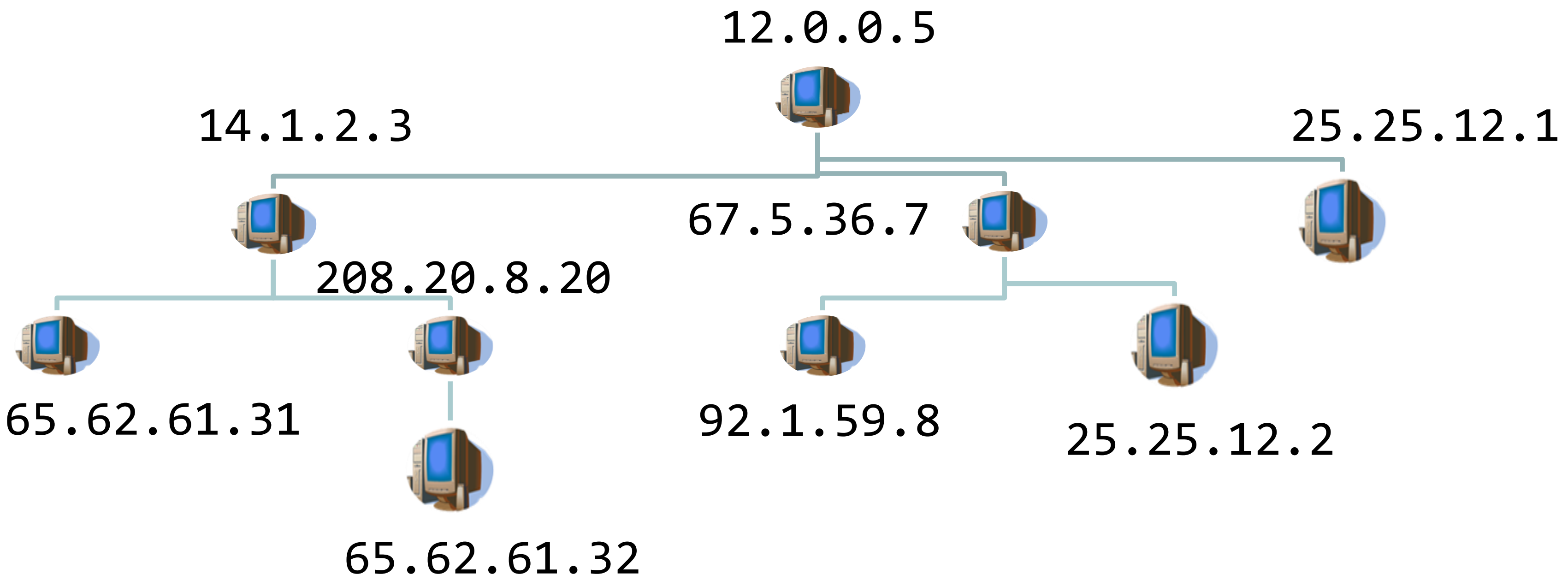
1	20
2	**20**
3	28
4	28
5	4

1	**21**
2	28
3	28
4	28
5	4

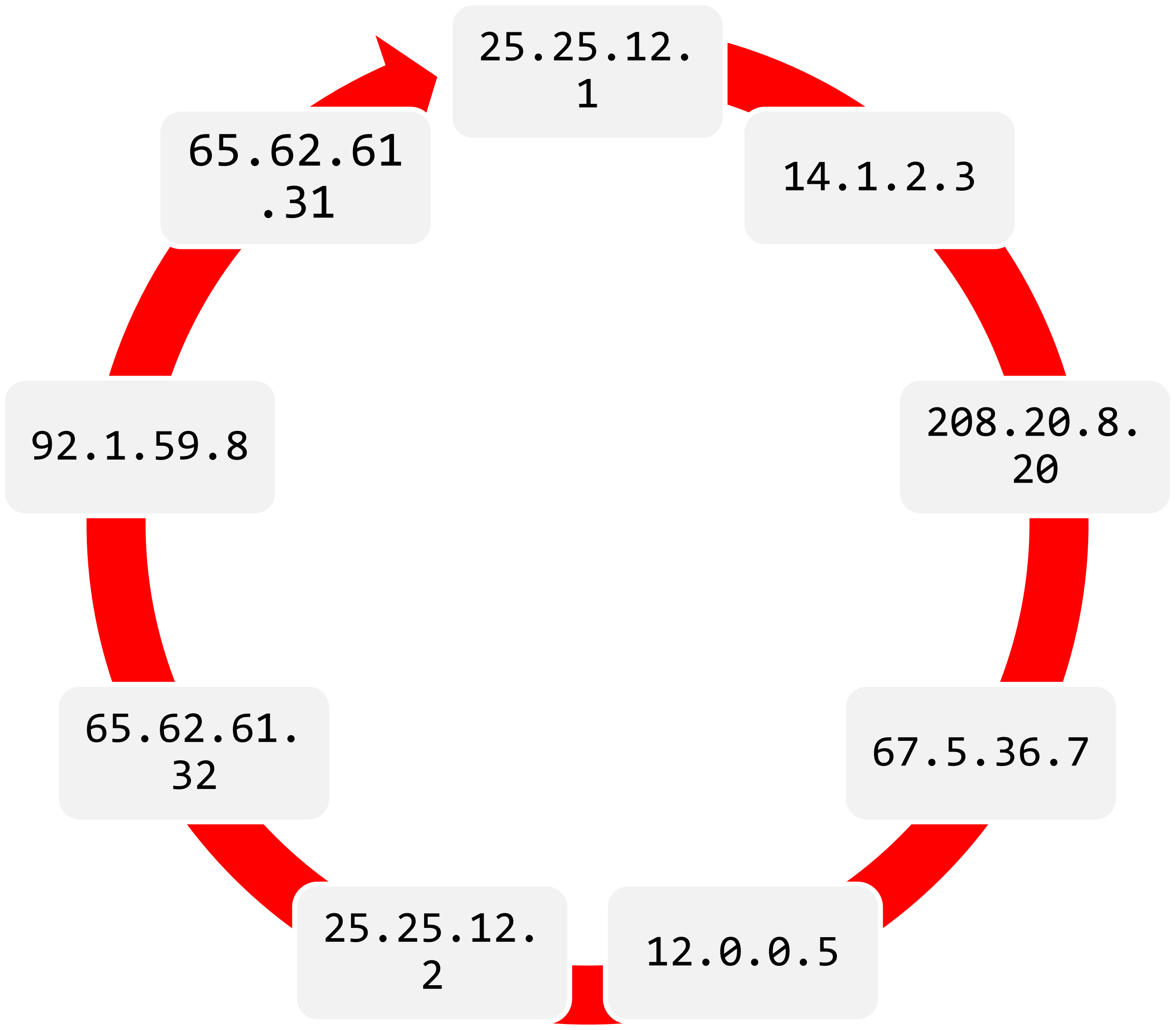
1	**28**
2	28
3	28
4	1
5	9

1	1
2	1
3	1
4	**4**
5	14

Multicast→Chord



Multicast→Chord



Exploiting Network Proximity

Problem: The logical organization of nodes in the overlay may lead to erratic message transfers in the underlying Internet: node k and node $succ(k+1)$ may be very far apart.

Topology-aware node assignment: When assigning an ID to a node, make sure that nodes close in the ID space are also close in the network. Can be very difficult.

Proximity routing: Maintain more than one possible successor, and forward to the closest.

Example: in Chord $FT_p[i]$ points to first node in $INT = [p + 2^{\{i-1\}}, p + 2^i - 1]$.
Node p can also store pointers to other nodes in INT.

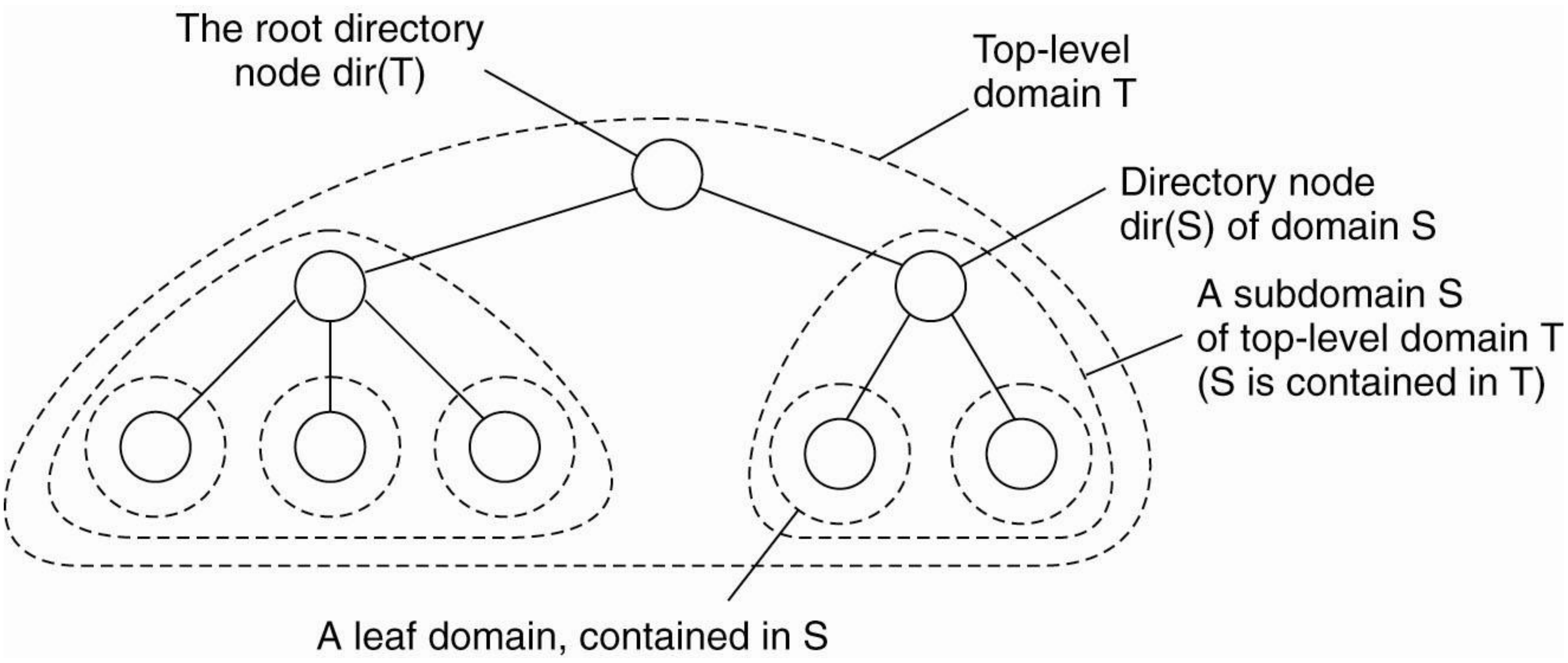
Proximity neighbor selection: When there is a choice of selecting who your neighbor will be (not in Chord), pick the closest one.

So Far

- Naming
  - Theory: Naming Entities
  - Flat Naming
    - DHTs – Chord
    - HLS

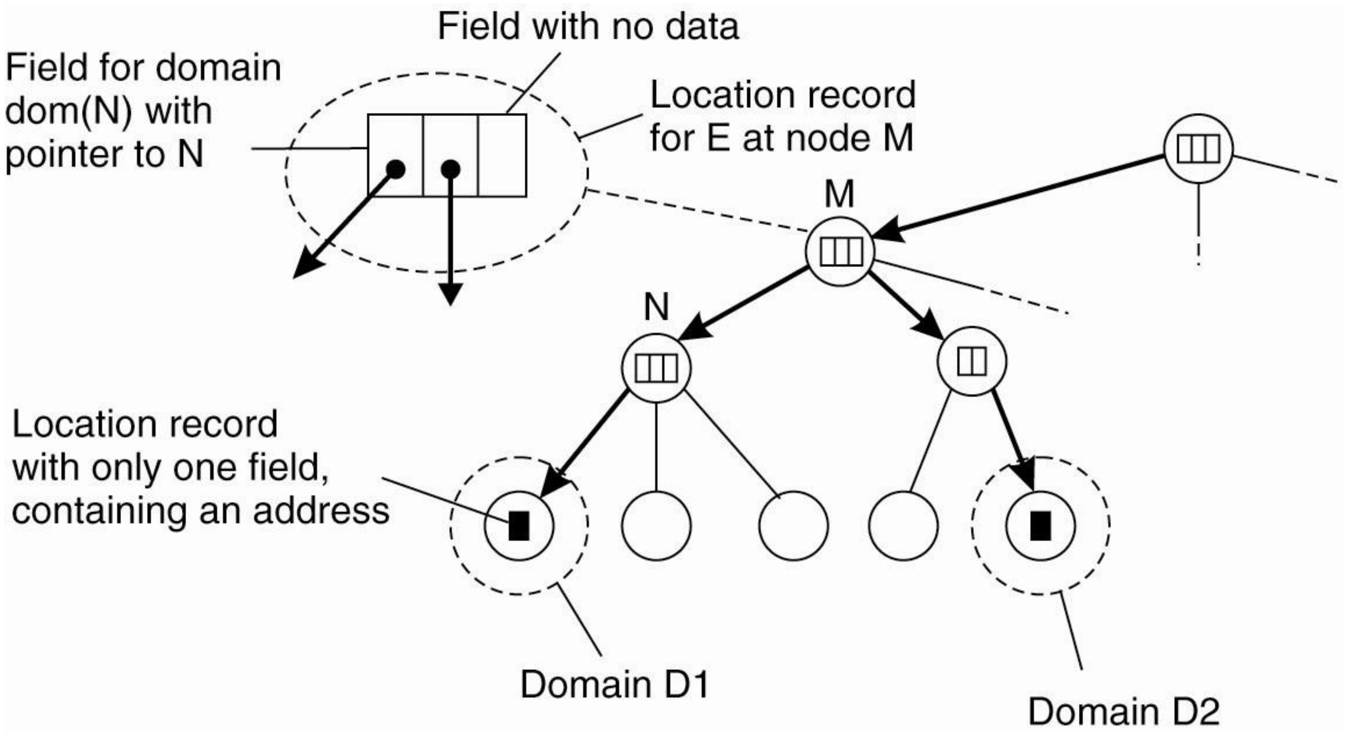
Hierarchical Location Services (HLS)

Basic idea: Build a large-scale search tree for which the underlying network is divided into hierarchical domains. Each domain is represented by a separate directory node.


The root directory node dir(T)
Top-level domain T
Directory node dir(S) of domain S
A subdomain S of top-level domain T (S is contained in T)
A leaf domain, contained in S

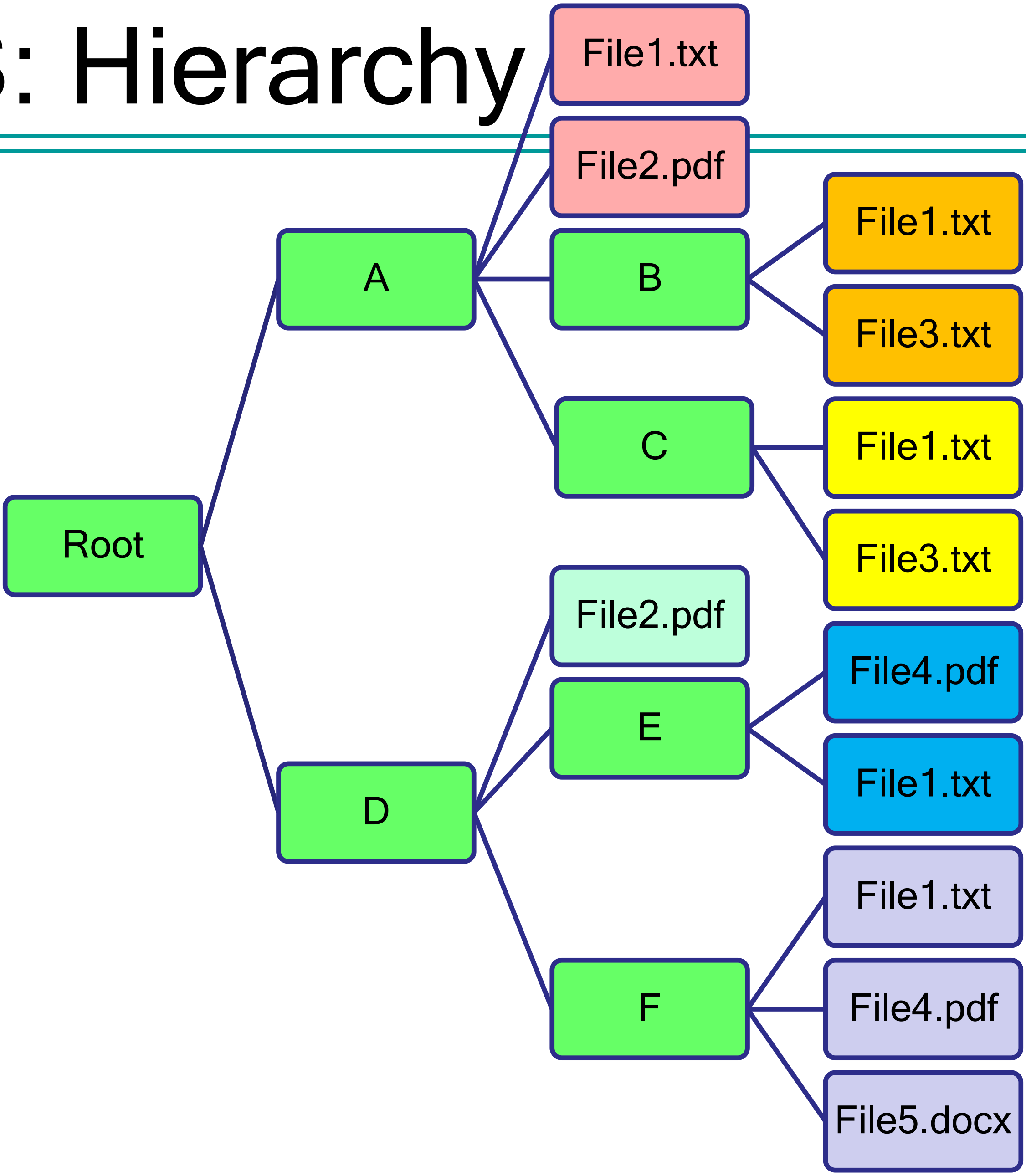
HLS: Tree Organization

- The address of an entity is stored in a leaf node, or in an 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



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

HLS: Hierarchy



HLS: Routing Tables

Root Table	
Resource	Found At
File1.txt	A, D
File2.pdf	A, D
File3.txt	A
File4.pdf	D
File5.docx	D

A Table	
Resource	Found At
File1.txt	ME, B,C
File2.pdf	ME
File3.txt	B, C

D's Table	
Resource	Found At
File1.txt	E, F
File2.pdf	ME
File4.pdf	E, F
File5.docx	F

B's Table	
Resource	Found At
File1.txt	ME
File3.txt	ME

C's Table	
Resource	Found At
File1.txt	ME
File3.txt	ME

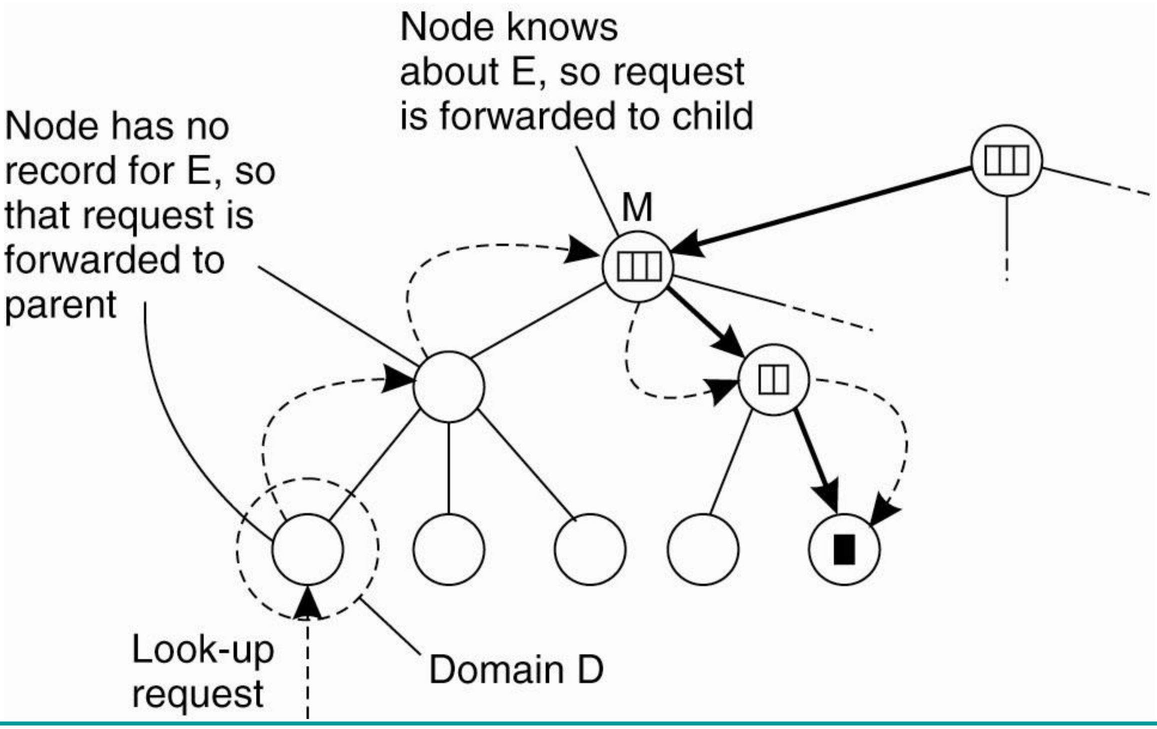
E's Table	
Resource	Found At
File1.txt	ME
File4.pdf	ME

F's Table	
Resource	Found At
File1.txt	ME
File4.pdf	ME
File5.docx	ME

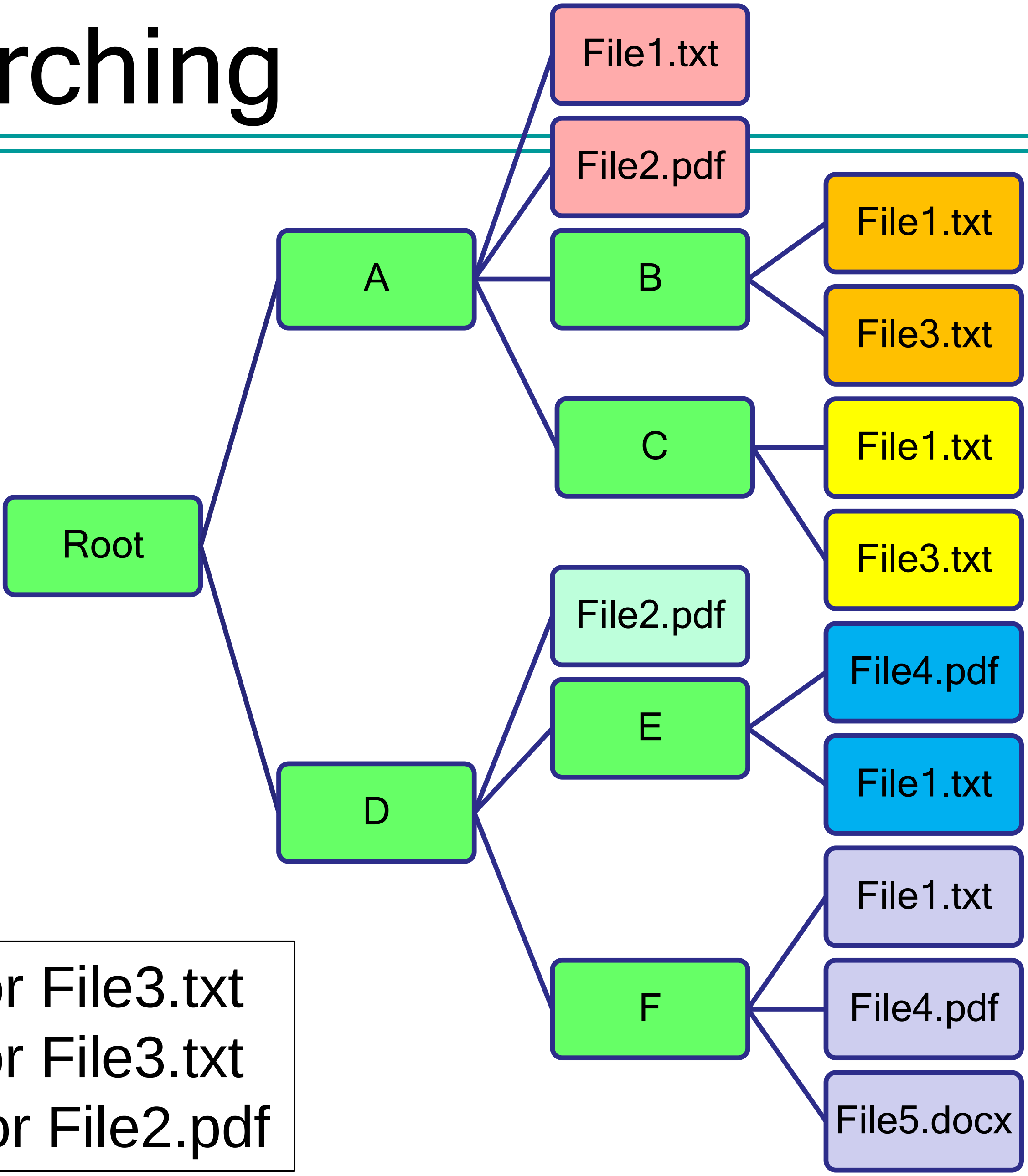
HLS: Lookup Operation

Basic principles:

- Start lookup at local leaf node
- If node knows about the entity, follow downward pointer, otherwise go one level up
- Upward lookup always stops at root


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

Searching



1. F looks for File3.txt
2. A looks for File3.txt
3. C looks for File2.pdf

HLS Insert and Delete

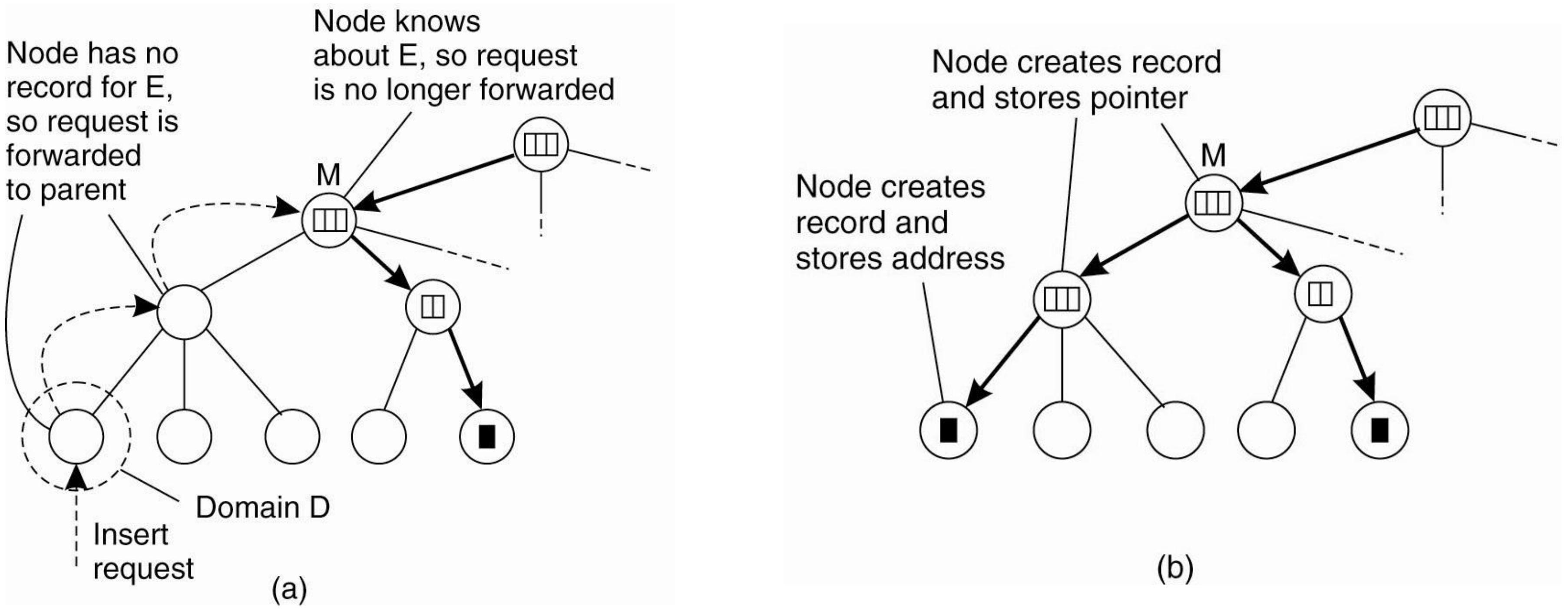
Insert ⊕

1. Node adds record to its own table
2. If this was the first record for the file at this node, inform the node's father.
3. Father adds a record to its own table, pointing to the child node
4. If this was the first record for the file, go to step 2

Delete ⊖

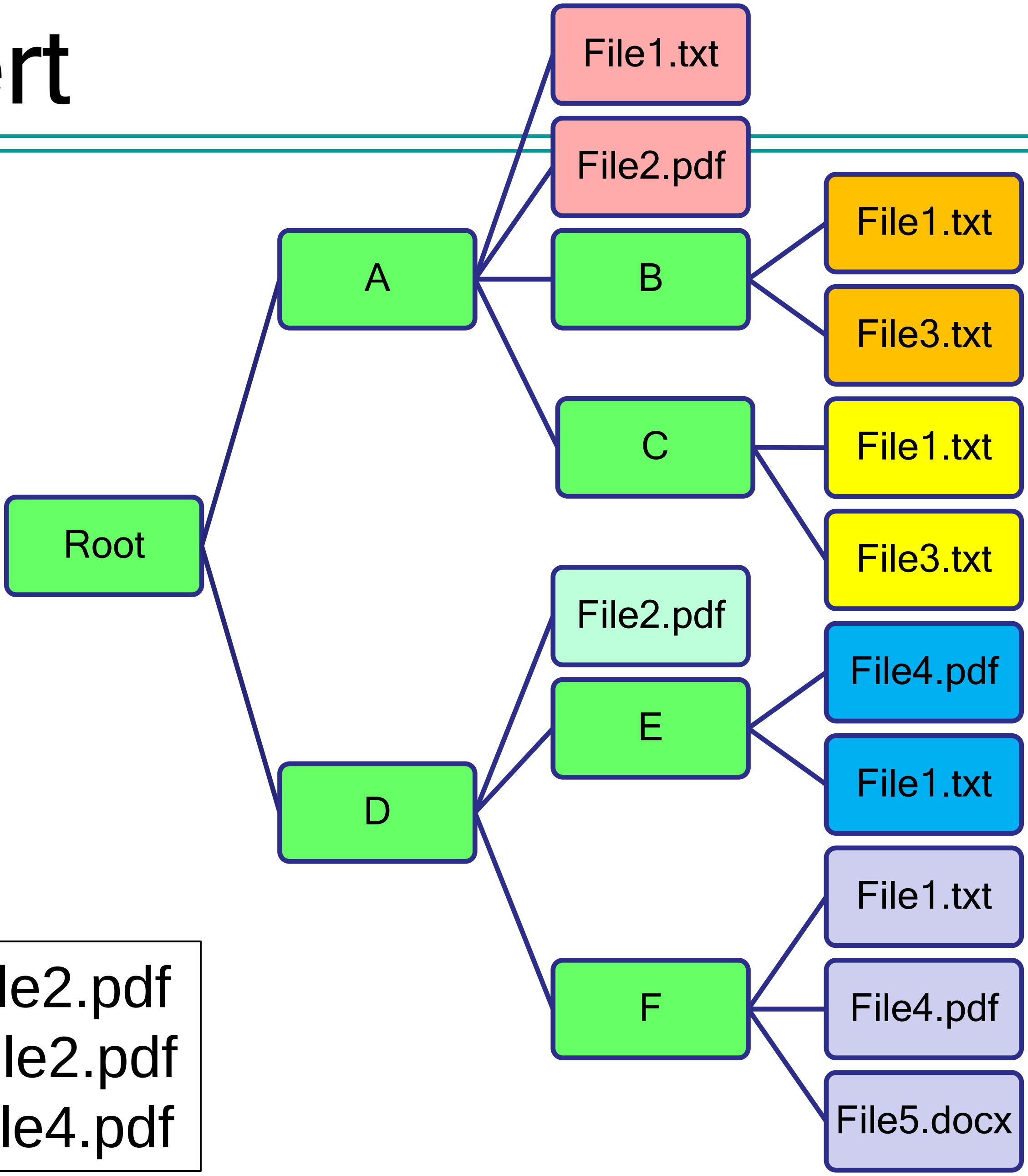
1. Node removes the record from its own table
2. If this was the last record for the file at this node, inform the node's father
3. Father removes the record pointing to the file from its table.
4. If this was the last record for the file, go to step 2.

HLS: Insert Operation



Node has no record for E, so request is forwarded to parent
Node knows about E, so request is no longer forwarded
M
Domain D
Insert request
(a)
Node creates record and stores pointer
M
Node creates record and stores address
(b)

Insert



1. F adds File2.pdf
2. C adds File2.pdf
3. B adds File4.pdf

HLS: Routing Tables

Root Table	
Resource	Found At
File1.txt	A, D
File2.pdf	A, D
File3.txt	A
File4.pdf	D
File5.docx	D

A Table	
Resource	Found At
File1.txt	ME, B,C
File2.pdf	ME
File3.txt	B, C

D's Table	
Resource	Found At
File1.txt	E, F
File2.pdf	ME
File4.pdf	E, F
File5.docx	F

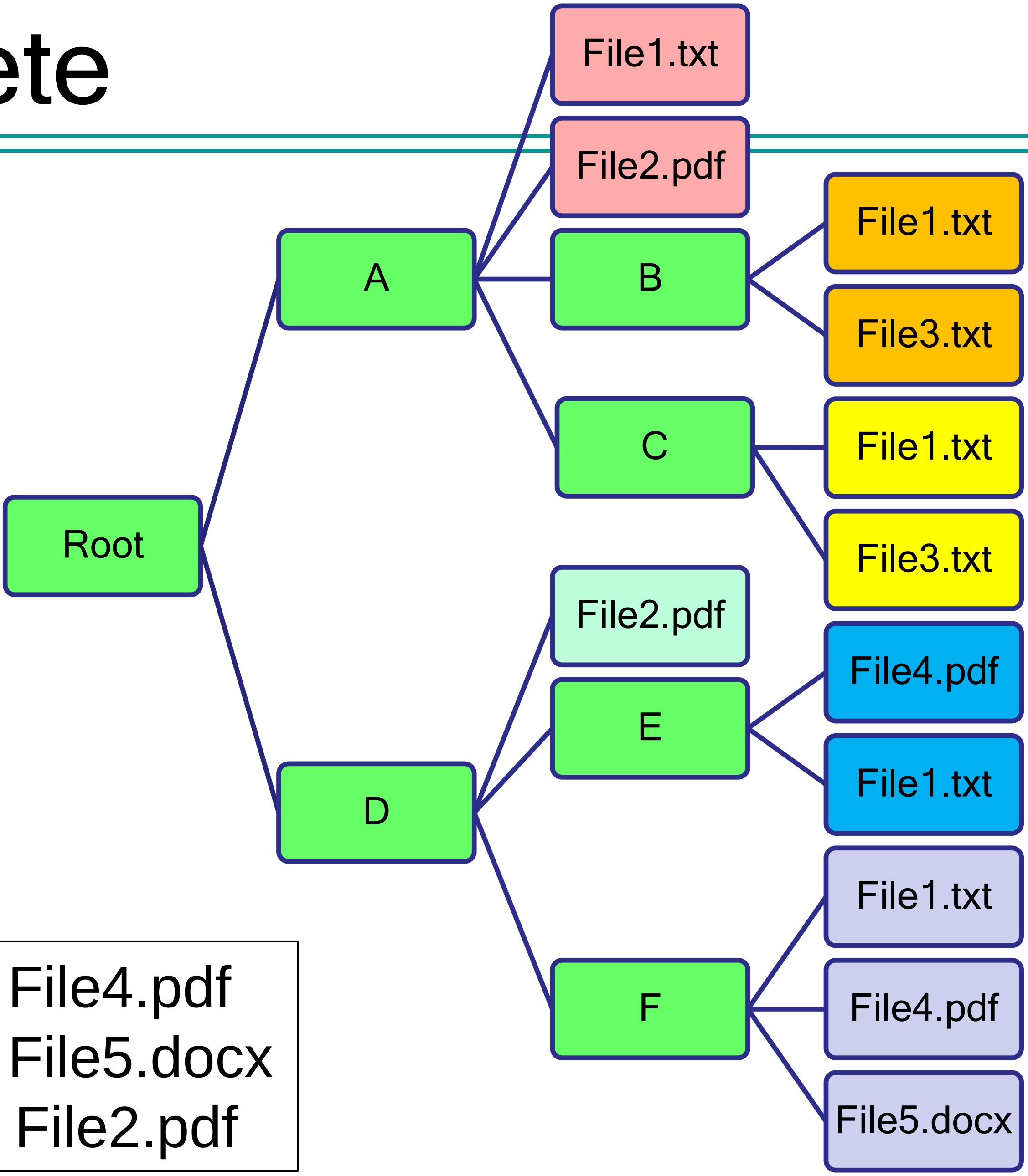
B's Table	
Resource	Found At
File1.txt	ME
File3.txt	ME

C's Table	
Resource	Found At
File1.txt	ME
File3.txt	ME

E's Table	
Resource	Found At
File1.txt	ME
File4.pdf	ME

F's Table	
Resource	Found At
File1.txt	ME
File4.pdf	ME
File5.docx	ME

Delete



1. F deletes File4.pdf
2. F deletes File5.docx
3. D deletes File2.pdf

Can we reduce load on the root?

Conclusion

- Naming
  - Theory: Naming Entities
  - Flat Naming
    - DHTs – Chord
    - HLS