Distributed System Architectures

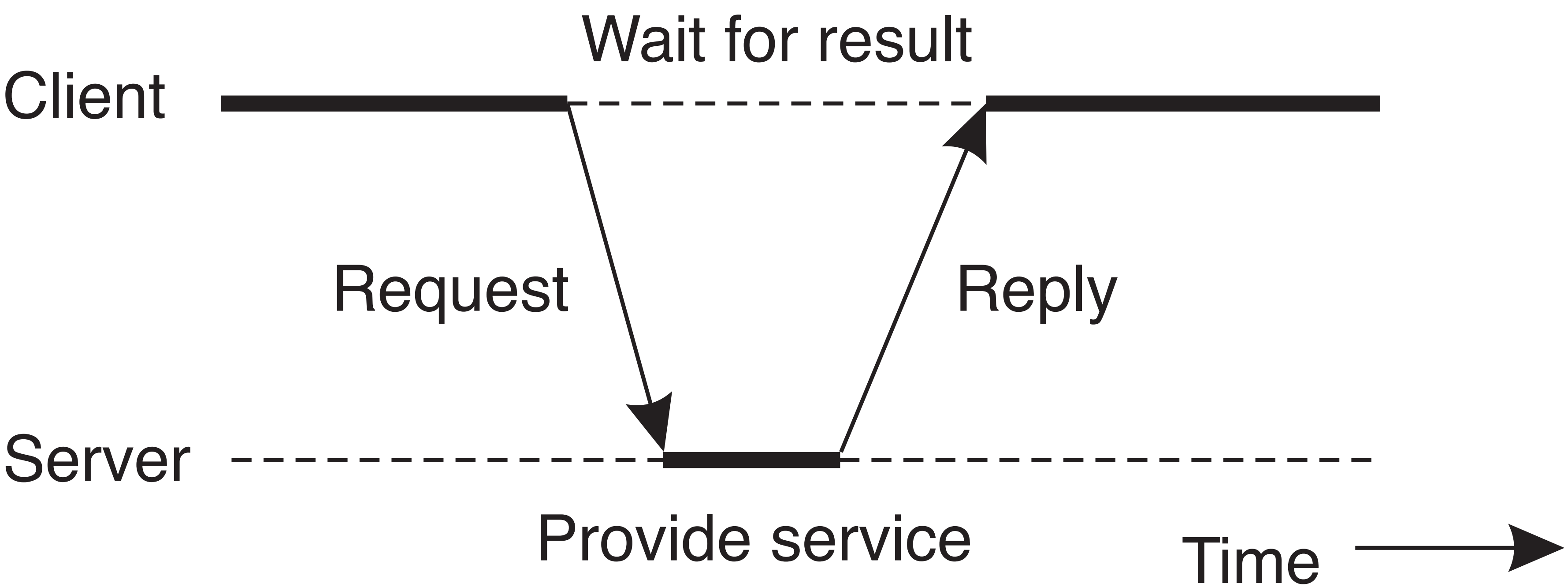
Yao Liu

System architectures

- Centralized architectures
  - Client-server applications
- Decentralized architectures
  - Peer-to-peer applications
- Hybrid architectures

Centralized architectures

- Request-reply behavior



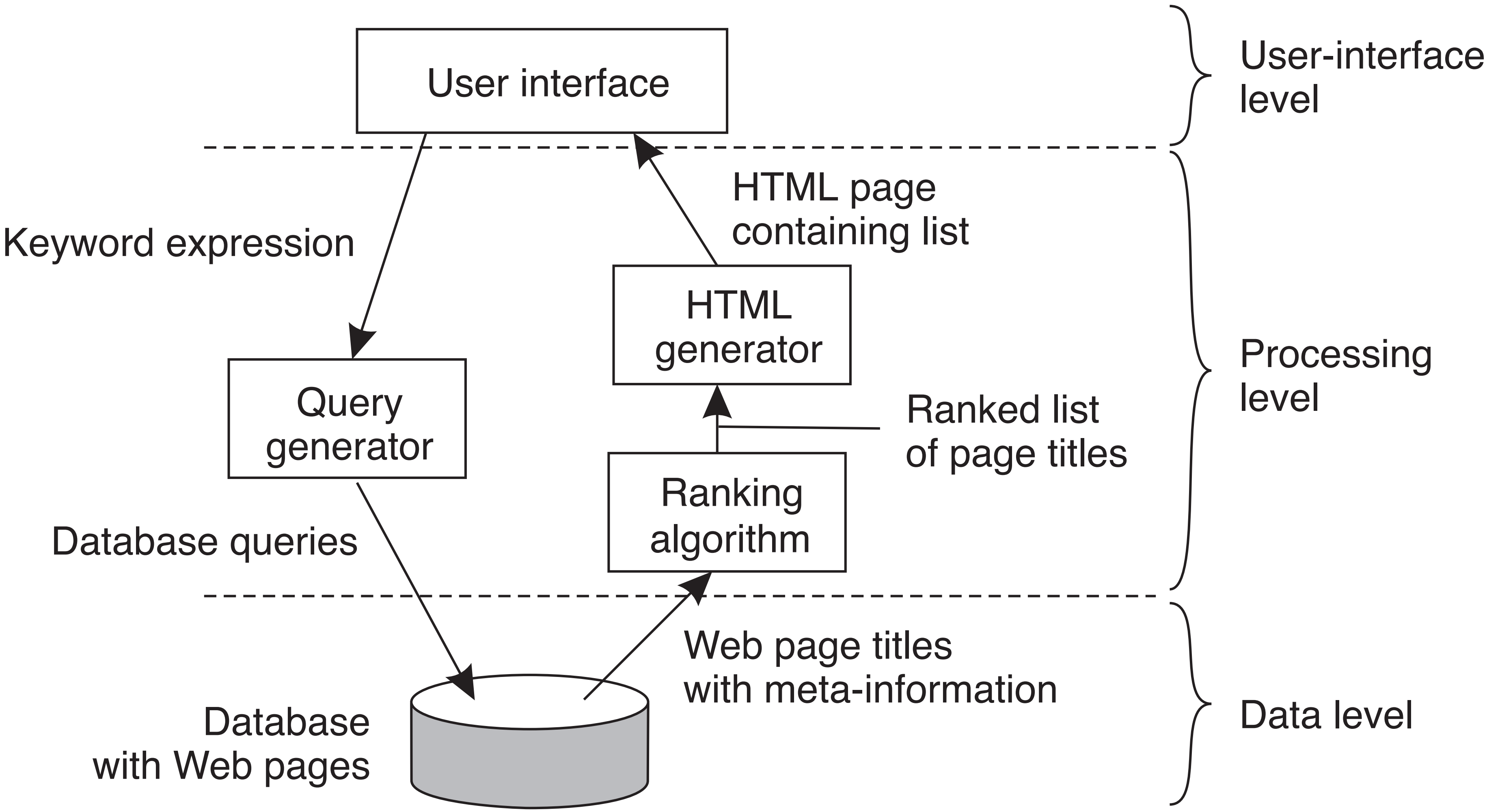
Application software architectures for client-server systems

- Many applications can be considered to be made up of three software components or logical layers
  - user interface
  - processing layer
  - data layer

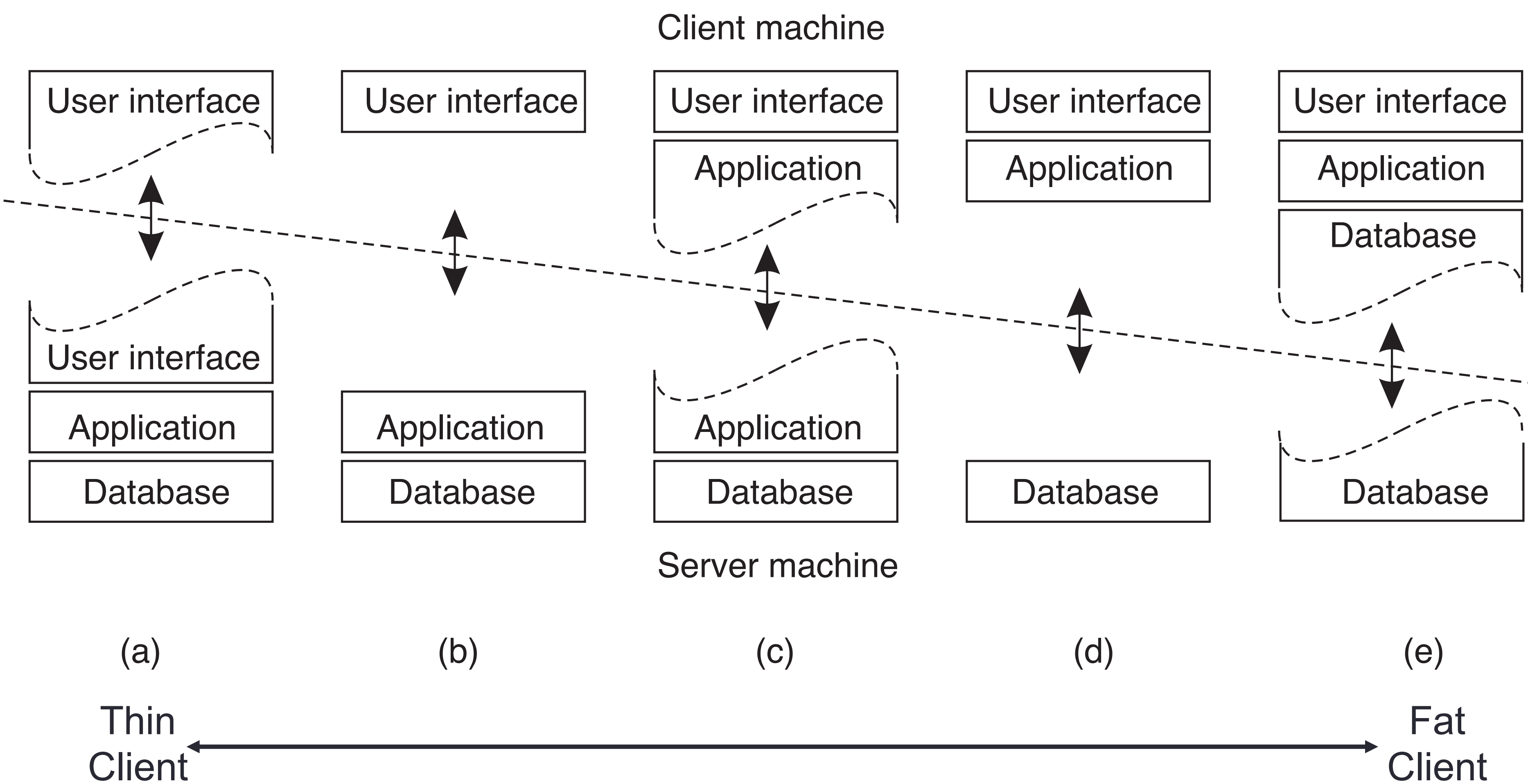
Example

- Web search engine
  - Interface: type in a keyword string
  - Processing: processes to generate DB queries, rank replies, format response
  - Data: database of web pages

Application layering

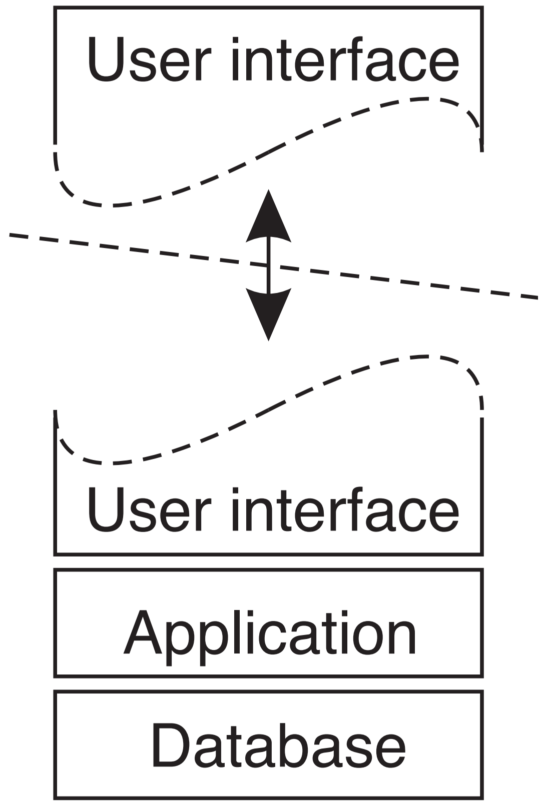


Two-tiered architecture



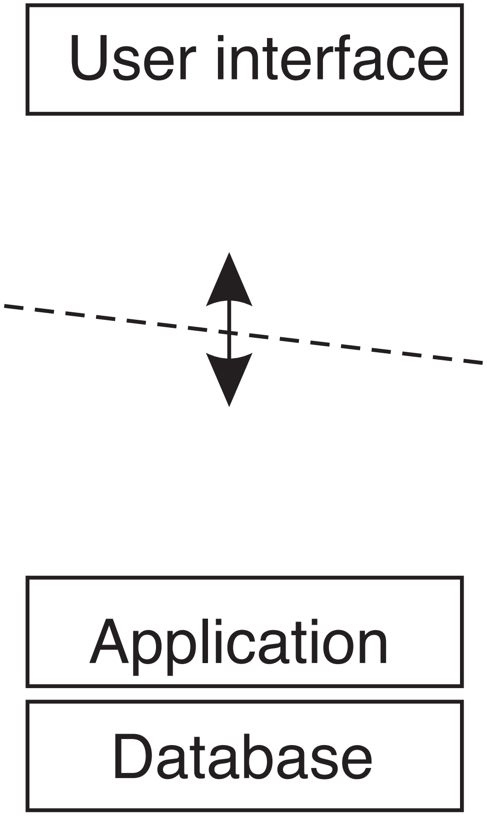
Distributed presentation

- Example: Remote Desktop



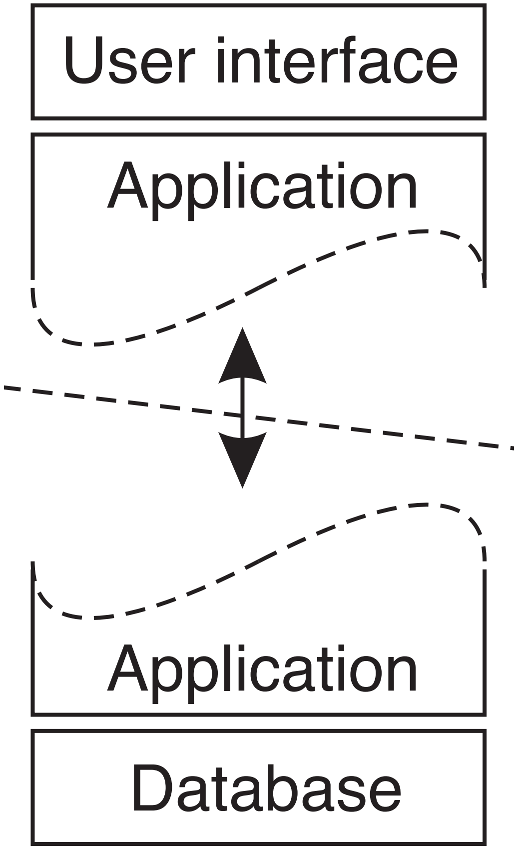
Remote presentation

- Example: telnet



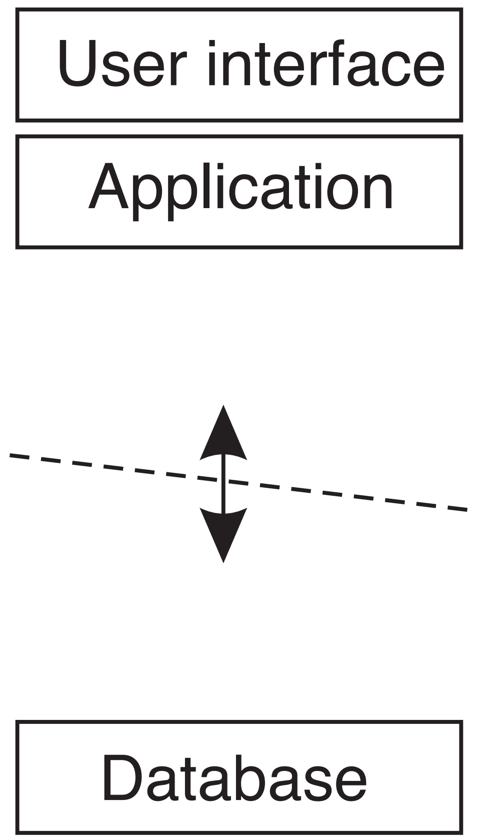
Distributed programs

- Example: World Wide Web



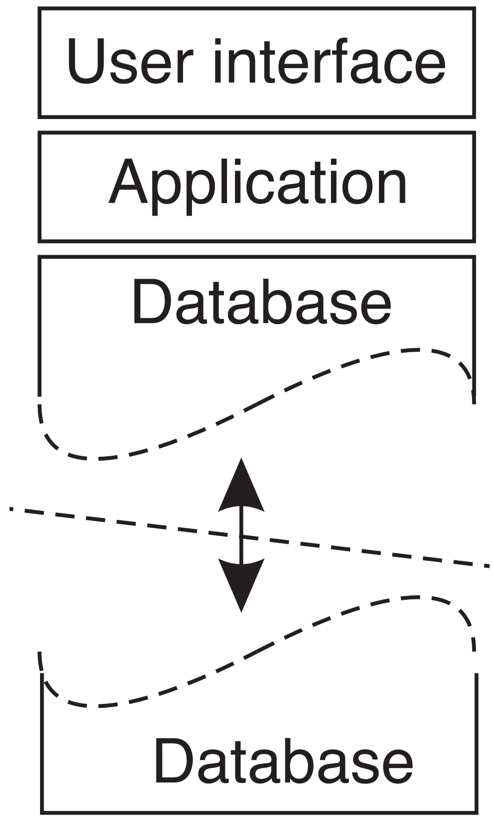
Remote data

- Example: Network File Systems (NFS)

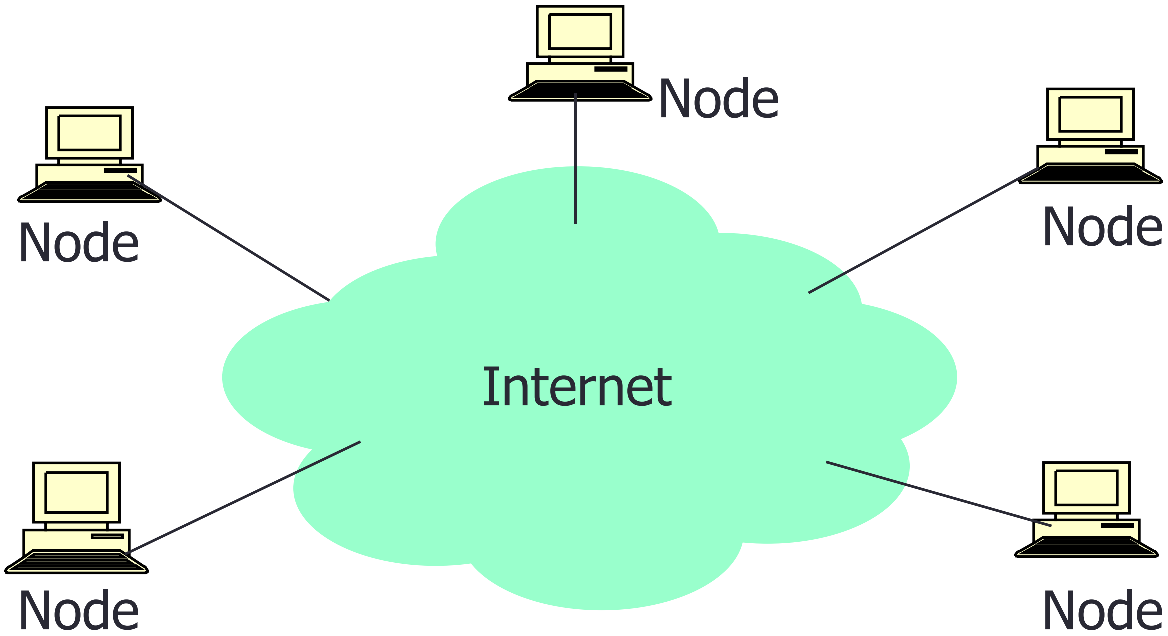


Distributed data

- Example: Browser with cache, Andrew File System (AFS), Dropbox



Decentralized architectures



- A decentralized system architecture:
  - No centralized control
  - Nodes are symmetric in function
- Nodes are unreliable
- Nodes form an overlay network

Centralized vs. decentralized architectures

- Traditional client-server architectures exhibit **vertical distribution**. Each level serves a different purpose in the system.
  - Logically different components reside on different nodes
- **Horizontal distribution** (P2P): each node has roughly the same processing capabilities and stores/manages part of the total system data.
  - Better load balancing, more resistant to denial-of-service attacks, but harder to manage than C/S
  - Communication & control is not hierarchical; all about equal

Peer-to-peer

- Nodes act as **both** client and server; interaction is symmetric
- Each node acts as a server for part of the total system data
- **Overlay networks** connect nodes in the P2P system
  - Nodes in the overlay use their own addressing system for storing and retrieving data in the system
  - Nodes can route requests to locations that may not be known by the requester.

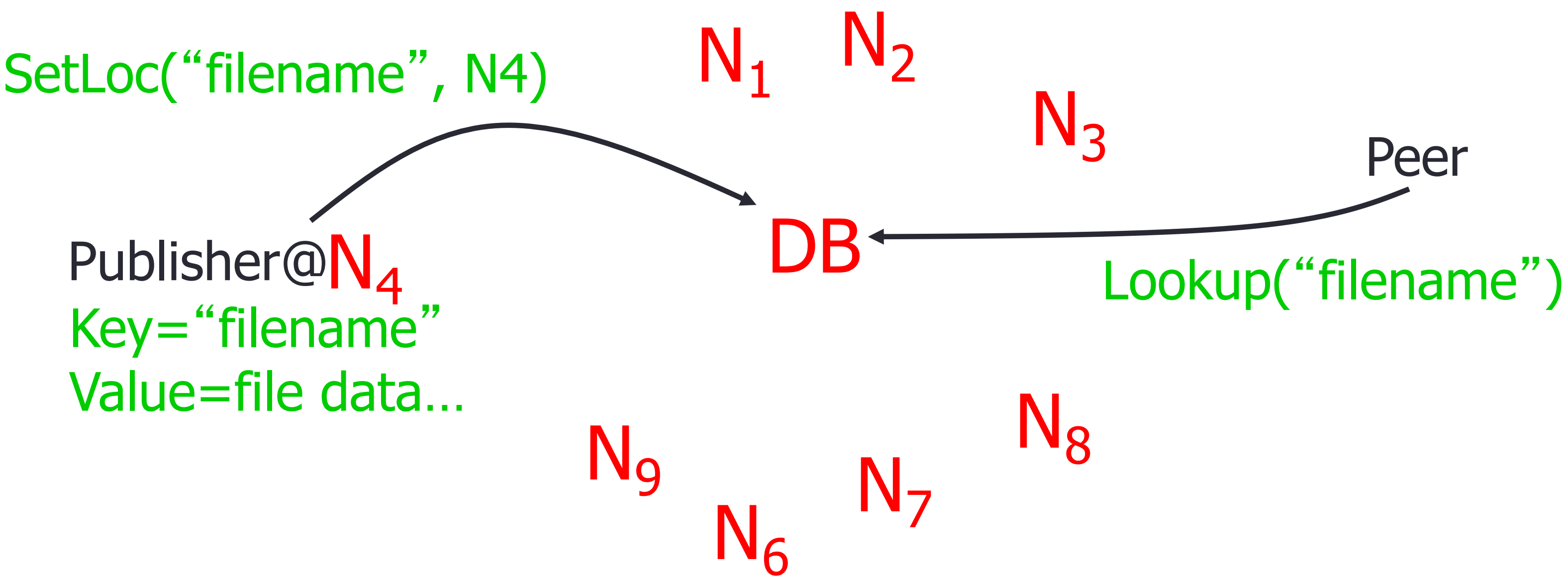
Overlay networks

- Are logical or virtual networks, built on top of a physical network
- A link between two nodes in the overlay may consist of several physical links.
- Messages in the overlay are sent to logical addresses, not physical (IP) addresses
- Various approaches used to resolve logical addresses to physical.

Organization of nodes in P2P systems

- Centralized directory
  - Original Napster
- Unstructured P2P systems
  - Gnutella and its successors
- Structured P2P systems
  - Based on Distributed Hash Tables (DHTs)
  - Chord, CAN, Tapestry, …

Centralized lookup (Napster)



Simple, but hard to keep state up to date and a single point of failure

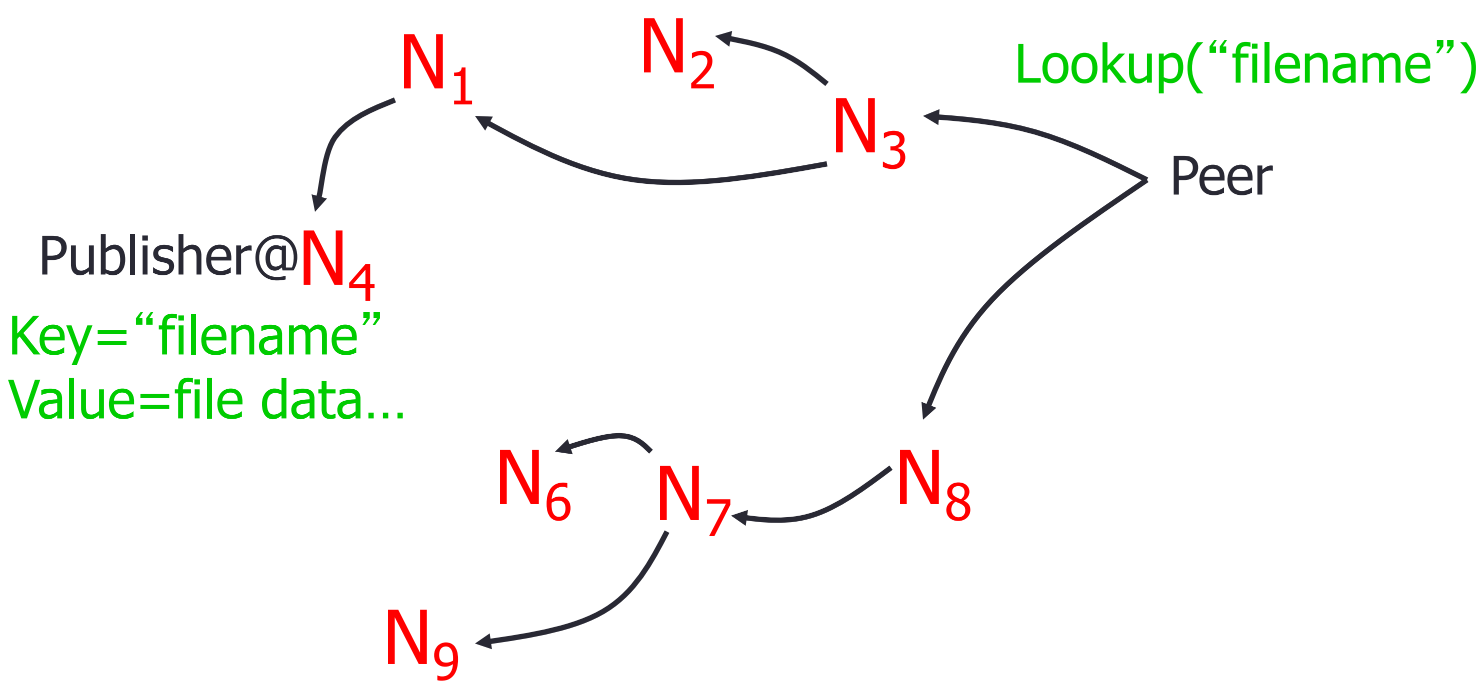
Unstructured P2P systems

- Overlay network resembles a random graph
- Searching for content based upon query flooding
  - Gnutella
- Each node knows about a subset of nodes, its “neighbors”
- Data items are randomly mapped to some node in the system & lookup is random
- Second generation P2P file sharing systems (e.g., Kazaa) introduced some structure in the form of superpeers

Locating a data object by flooding

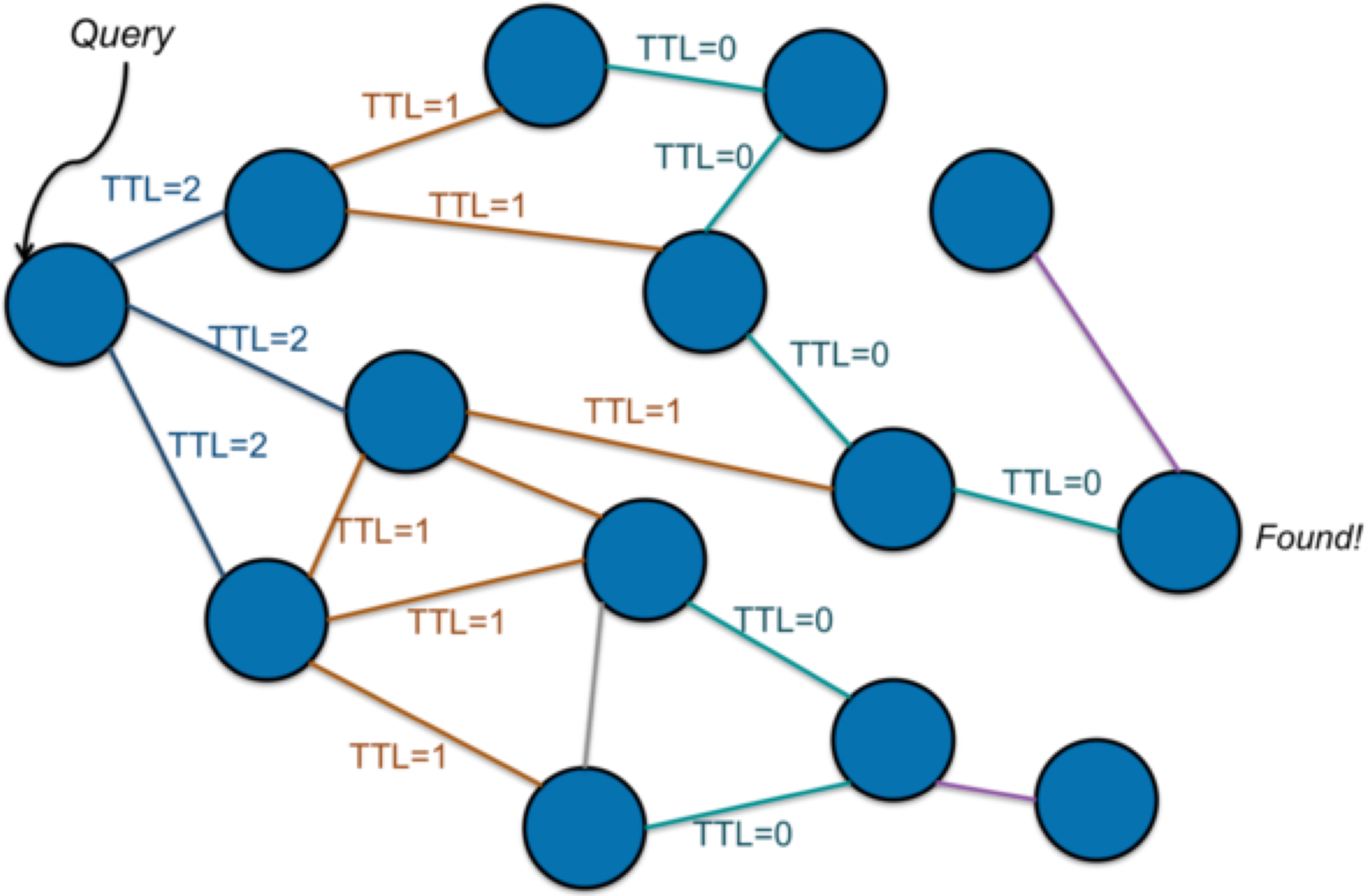
- Send a request to all known neighbors
  - If not found, neighbors forward the request to their neighbors
- Works well in small to medium sized networks, doesn’t scale well
- “Time-to-live” counter can be used to control number of hops
- Example system: Gnutella

Flooded queries (Gnutella)

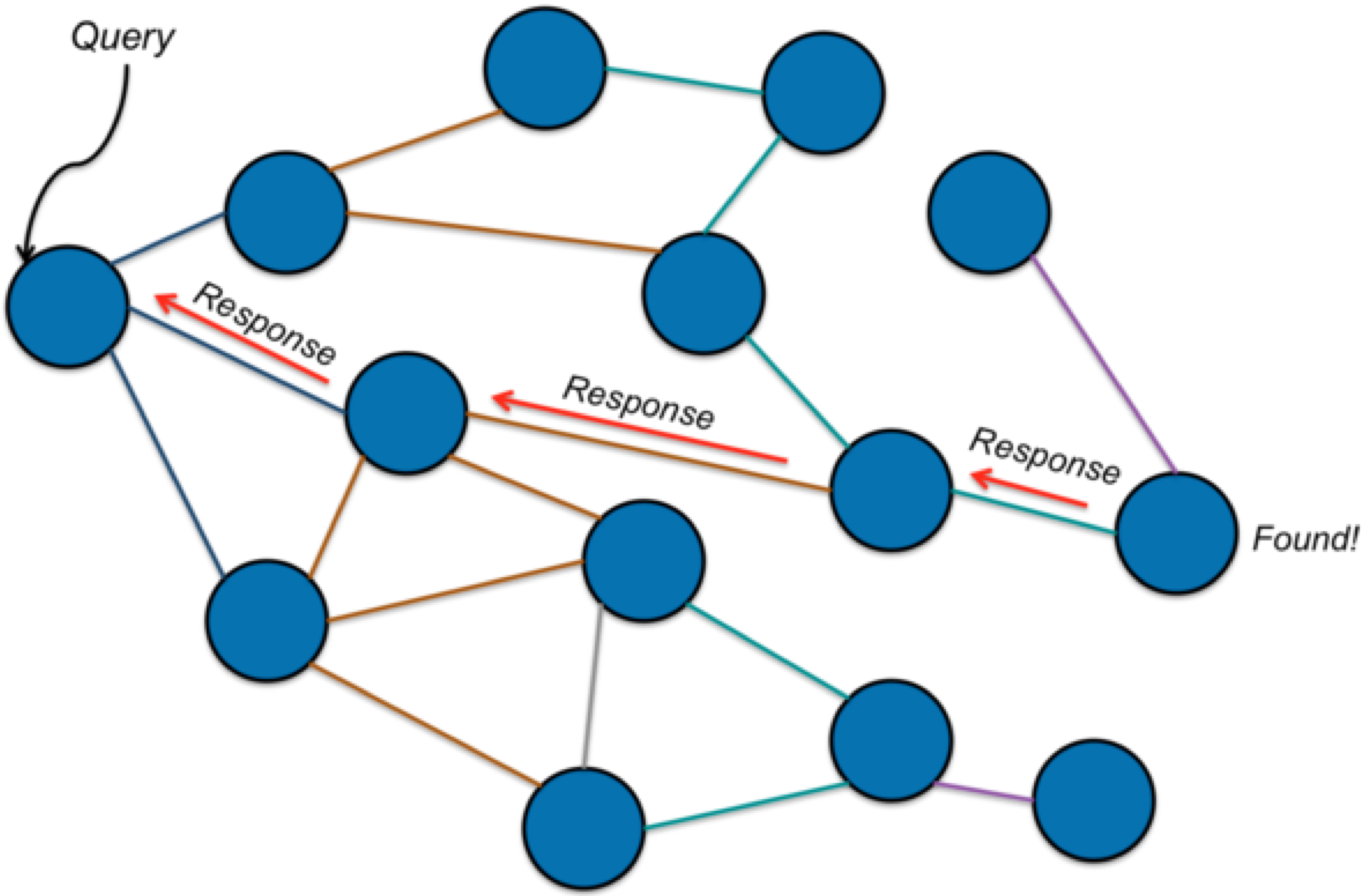


Robust, but worst case O(*N*) messages per lookup

Flooded queries

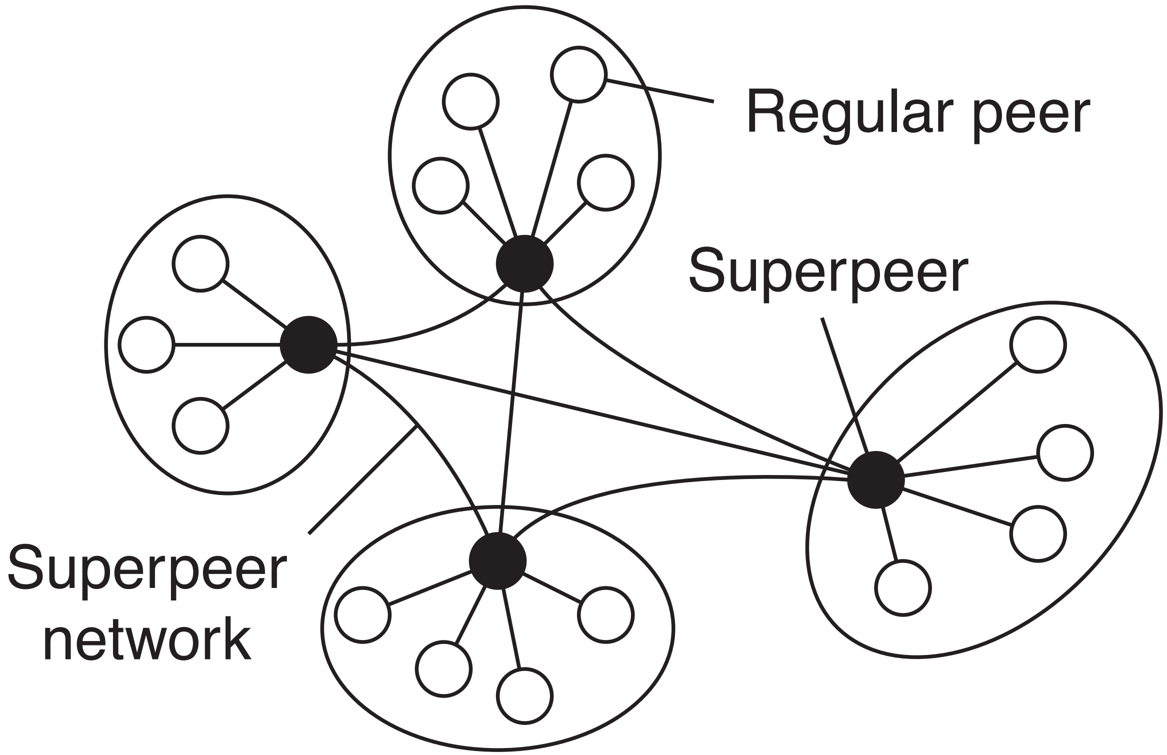


Back propagation



Superpeers

- A hierarchical organization of nodes into a superpeer network (e.g., Kazaa)



Superpeers

- Maintain index to some or all nodes in the system
- Supports resource discovery
- Act as servers to regular peers, act as peers to other superpeers
- Improve scalability by controlling floods
- Can also monitor state of network

Structured P2P systems

- A common approach is to use a **distributed hash table** (DHT) to organize the nodes
- Single-node hash table:
  - key = hash(name)
  - put(key, value)
  - get(key) → value
- How do I do this across millions of hosts on the Internet?
  - DHT

What is a DHT?

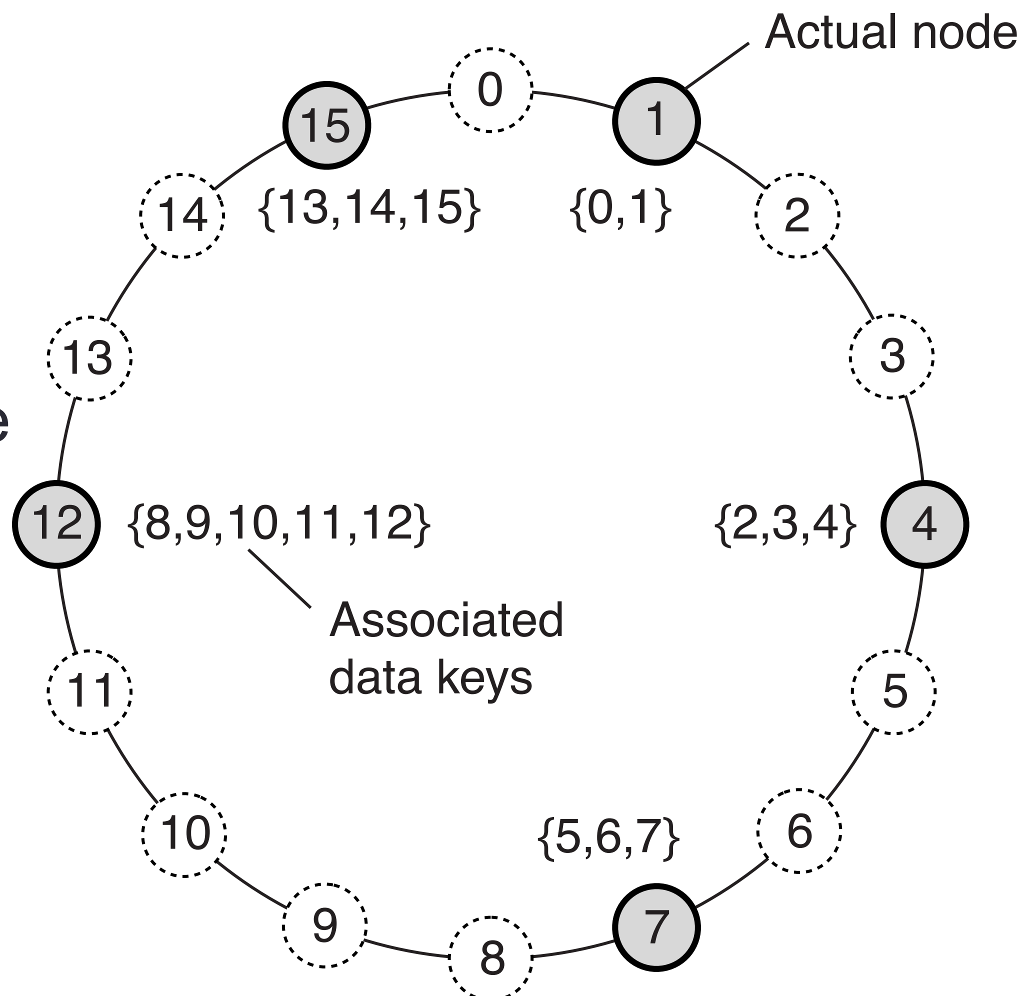
- Distributed Hash Table:
  - key = hash(data)
  - lookup(key) → node IP address (Chord)
  - send-RPC(IP address, PUT, key, value)
  - send-RPC(IP address, GET, key) → value
- Possibly a first step towards truly large-scale distributed systems
  - a tuple in a global database engine
  - a data block in a global file system
  - rare.mp3 in a P2P file-sharing system

Characteristics of DHT

- **Scalable** – to thousands, even millions of network nodes
  - Search time increases more slowly than size; usually *O*(log(N))
- **Fault tolerant** – able to re-organize itself when nodes fail
- **Decentralized** – no central coordinator (example of decentralized algorithms)

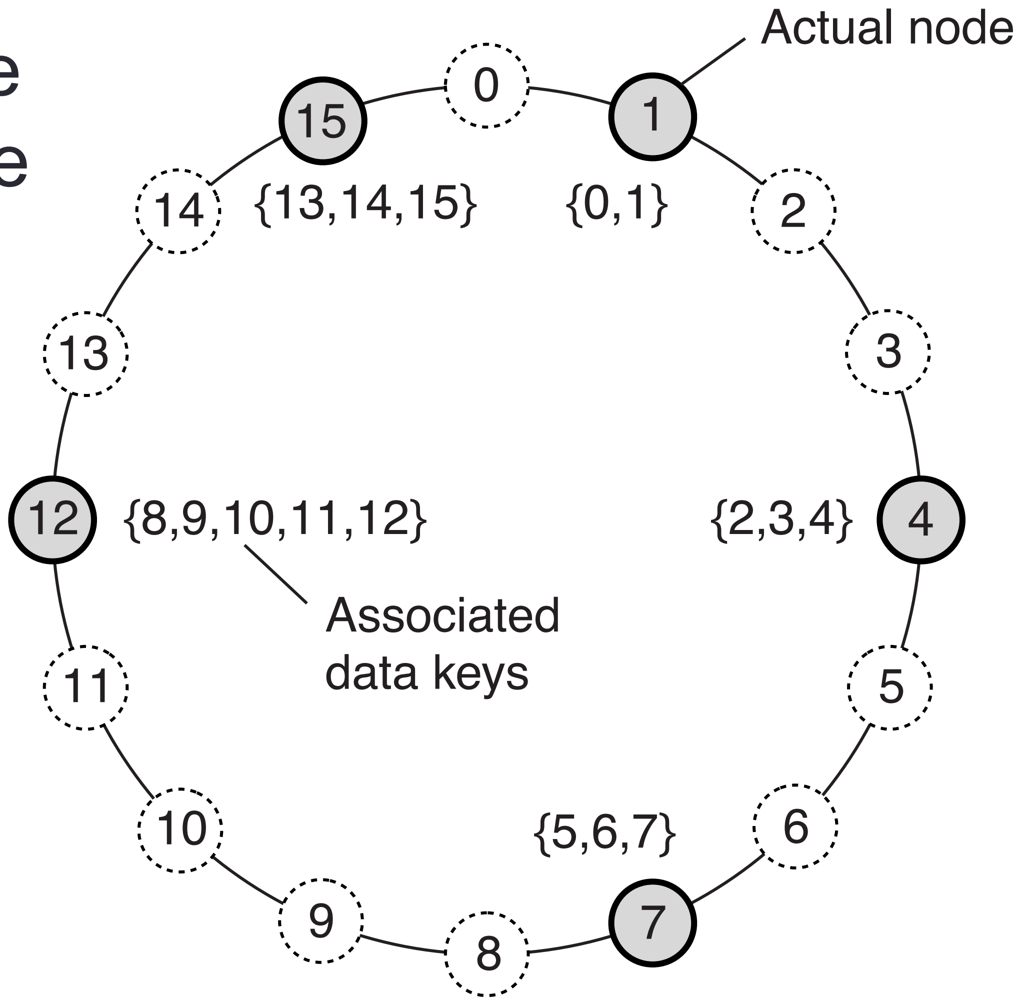
Distributed Hash Table

- Chord:
  - Map nodes to a large circular space
  - Map keys (e.g., hash(data)) to the same circular space
  - Key *k* belong to the first node whose identifier is equal to or follows *k* in the identifier space (successor(*k*))



Lookup in Chord

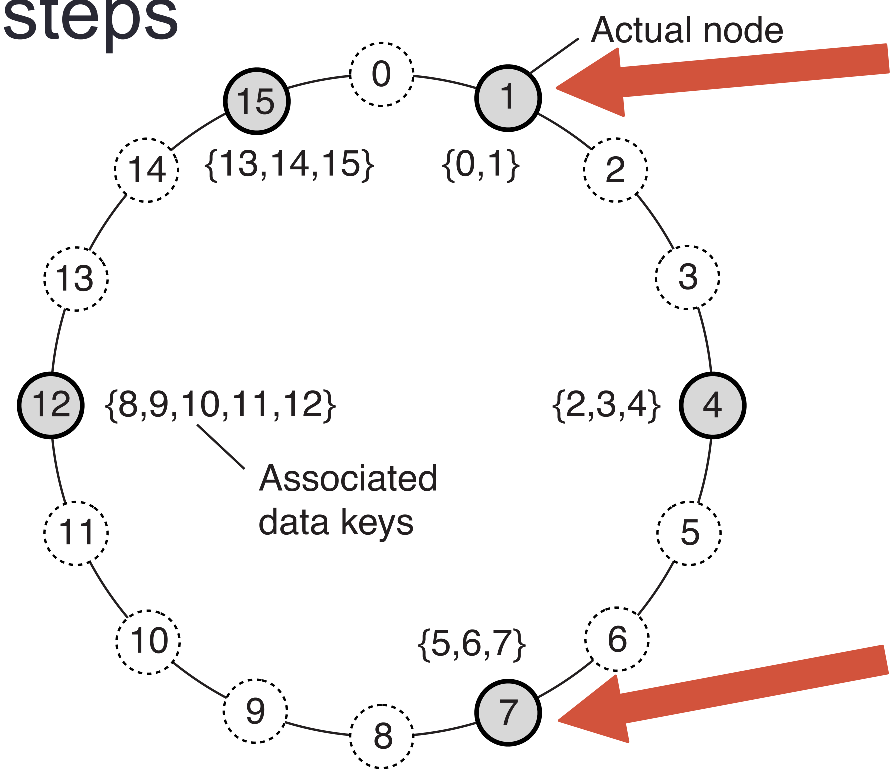
- Every node need to be aware of the next node on the ring
- May traverse all N nodes to find the Key
- O(N) steps



Lookup in Chord: finger table

- The i^{th} 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 circle.
- O(log*N*) steps

A client contacts node 1 to find the node that succeeds key 8

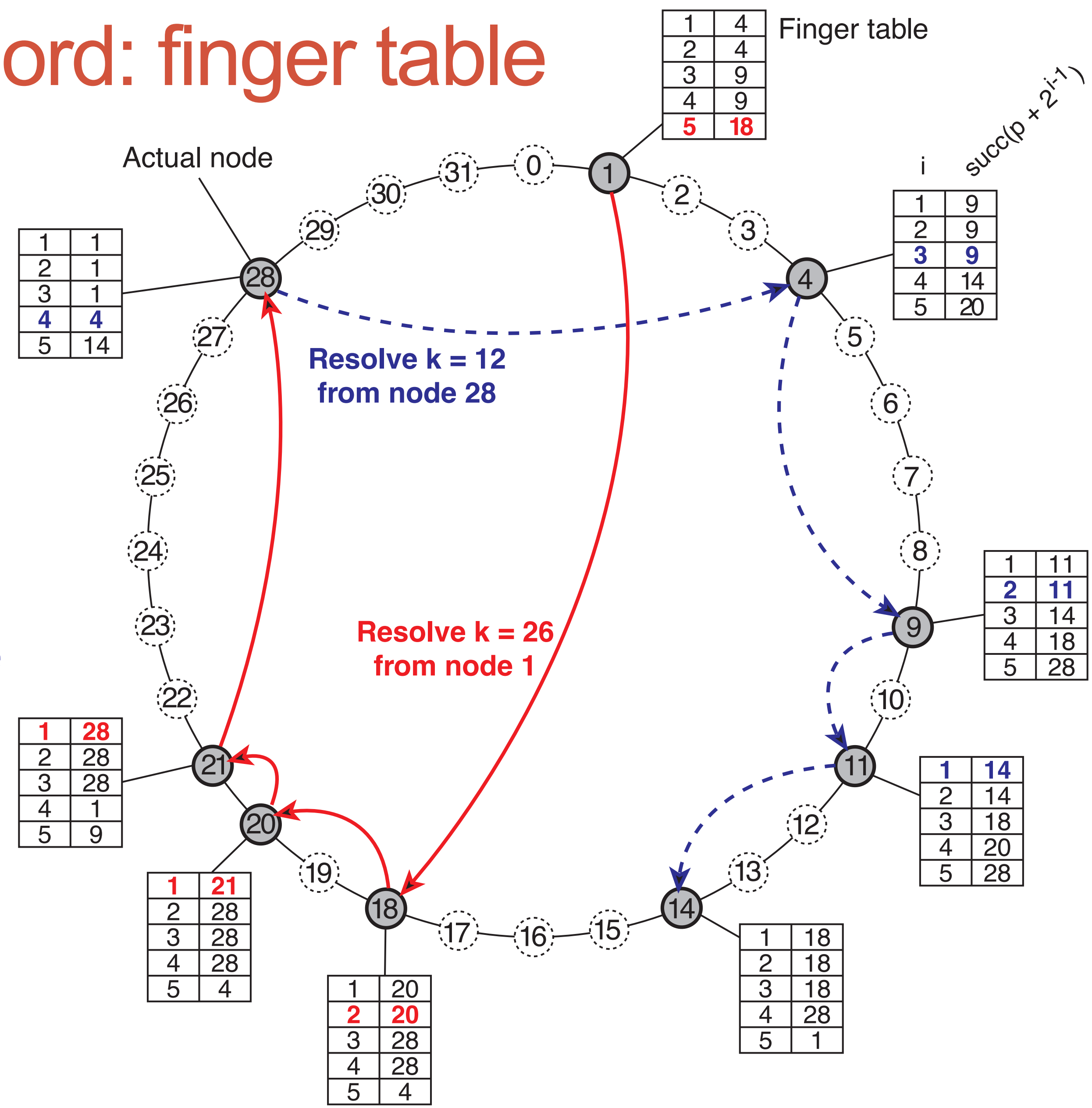


i	Start	Succ.
1	2	4
2	3	4
3	5	7
4	9	12

i	Start	Succ.
1	8	12
2	9	12
3	11	12
4	15	15

Lookup in Chord: finger table

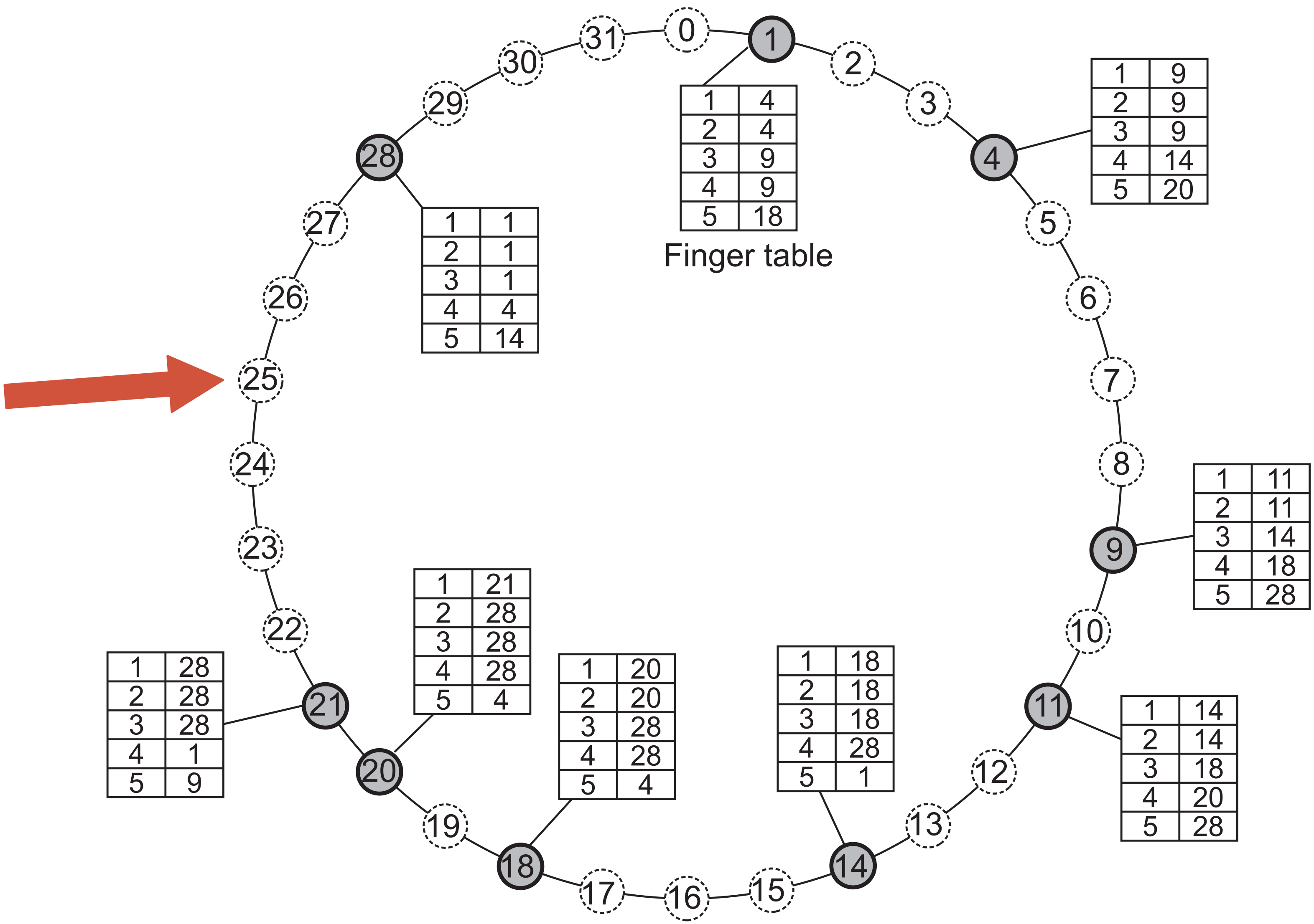
Resolving key 26 from node 1 and key 12 from node 28 in a Chord system.



Join & leave Chord

- Join
  - Generate the node's random identifier, `id`, using the hash function
  - Initialize its fingertable and its predecessor
  - Update fingertables of existing nodes
  - Assume data items from `succ(id)`
- Leave (normally)
  - Update fingertables of nodes that are affected
  - Move data to `succ(id)`
- Leave (due to failure)
  - Periodically, nodes can run "self-healing" algorithms

Node 25 joins the Chord DHT



Create a new fingertable for Node 25

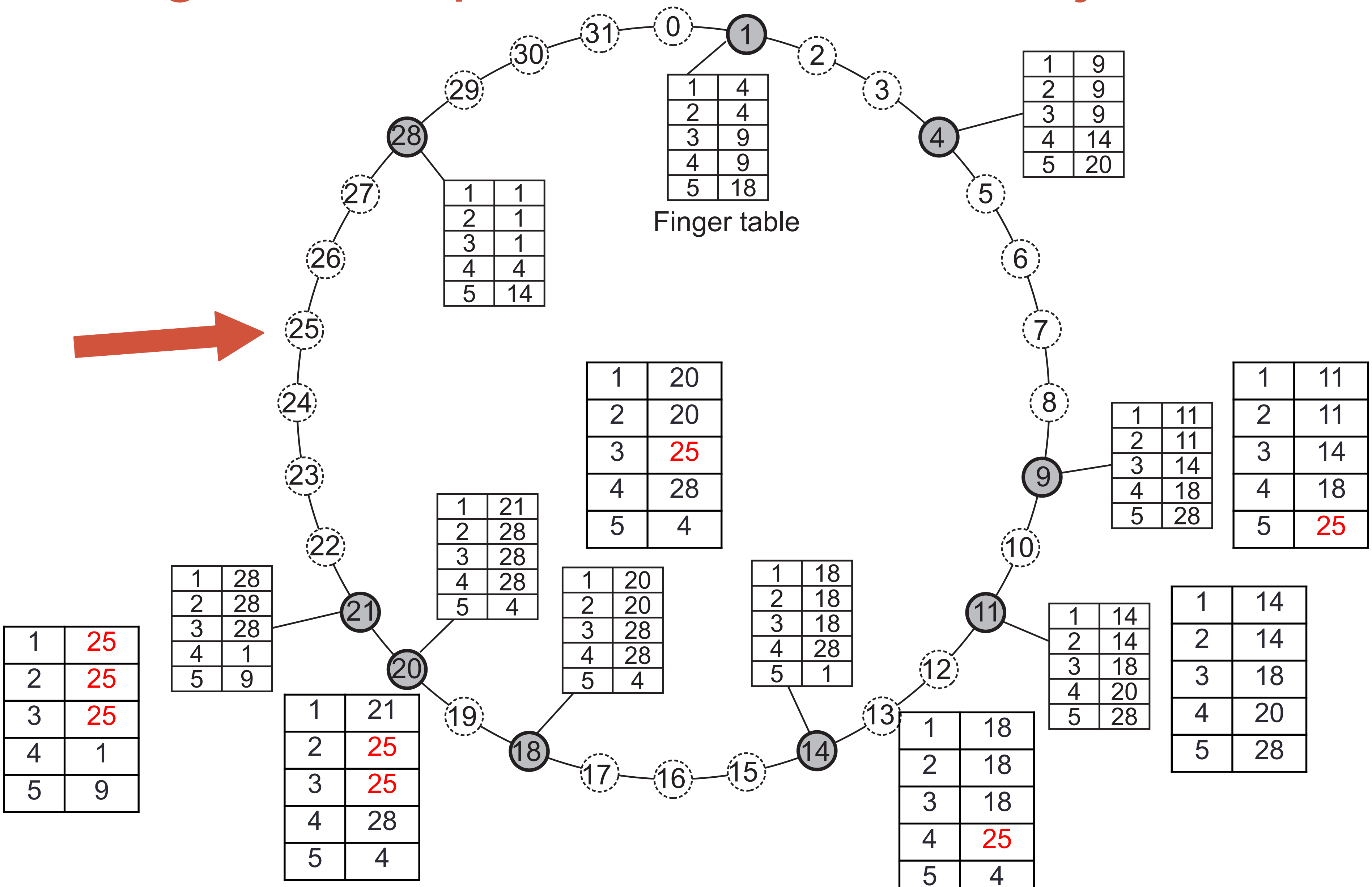
- Node 25 can contact an arbitrary node that already exists in the DHT, and ask this node to compute fingertable entries for it.

1	28
2	28
3	1
4	1
5	9

Fingertable updates

- For each node *p* that $p+2^{i-1}$ belongs to the interval (pred($node_{new}$), $node_{new}$], the new node will update node *p*'s i^{th} entry in the fingertable.
- A new node affects *O(log(N))* other nodes' fingertable entries in the system, on average
- Finding and updating these nodes takes: *O(log(N)*log(N))*

Fingertable updates after Node 25 joins



1	4
2	4
3	9
4	9
5	18

1	9
2	9
3	9
4	14
5	20

1	1
2	1
3	1
4	4
5	14

1	20
2	20
3	25
4	28
5	4

1	11
2	11
3	14
4	18
5	28

1	11
2	11
3	14
4	18
5	25

1	21
2	28
3	28
4	28
5	4

1	20
2	20
3	28
4	28
5	4

1	18
2	18
3	18
4	28
5	1

1	28
2	28
3	28
4	1
5	9

1	14
2	14
3	18
4	20
5	28

1	14
2	14
3	18
4	20
5	28

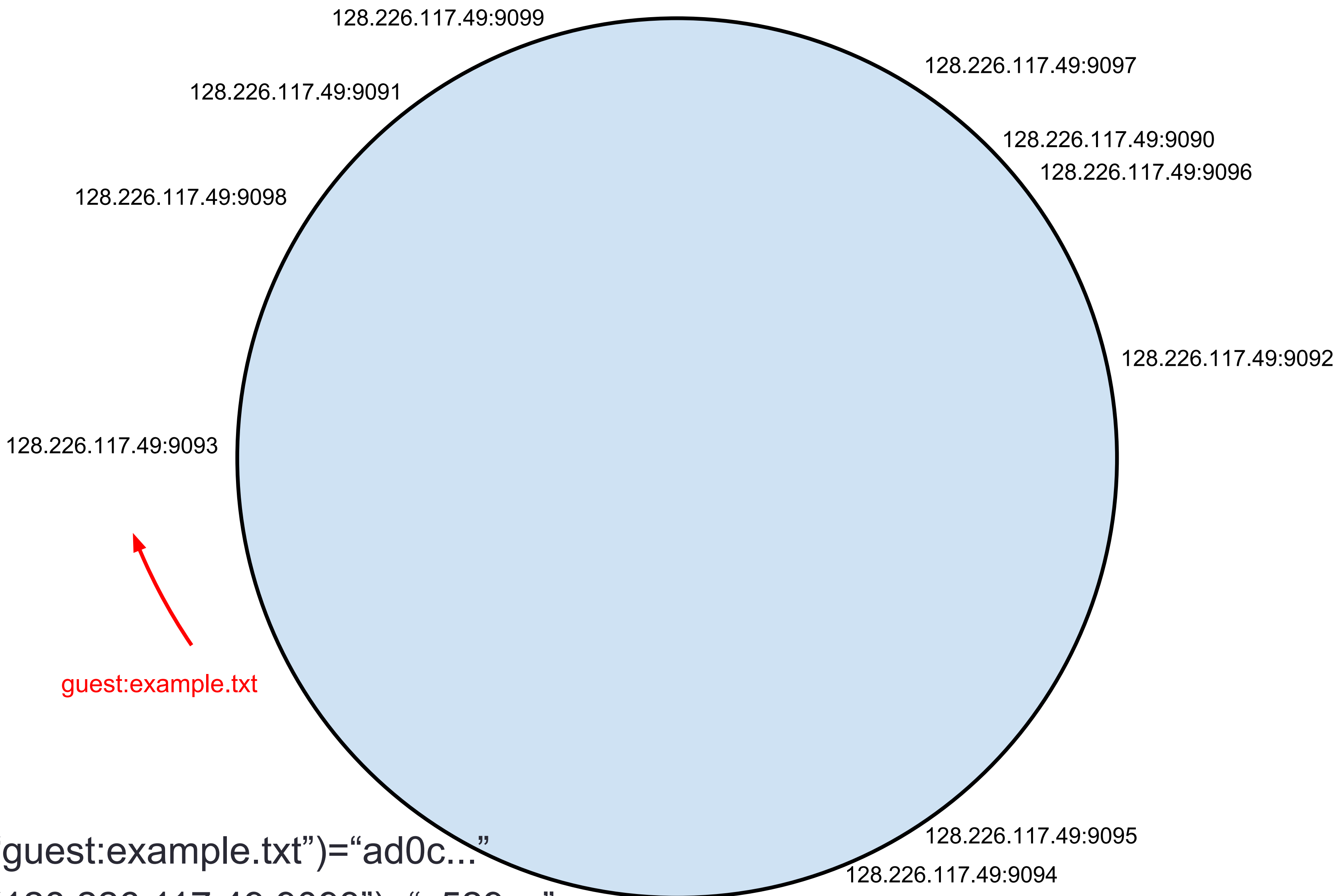
1	25
2	25
3	25
4	1
5	9

1	21
2	25
3	25
4	28
5	4

1	18
2	18
3	18
4	25
5	4

256-bit ID space

Use SHA256 to generate Node IDs and keys



SHA256(“guest:example.txt”)=“ad0c…”

SHA256(“128.226.117.49:9093”)=“c529…”

More details can be found in the Chord paper

- “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”
  - https://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

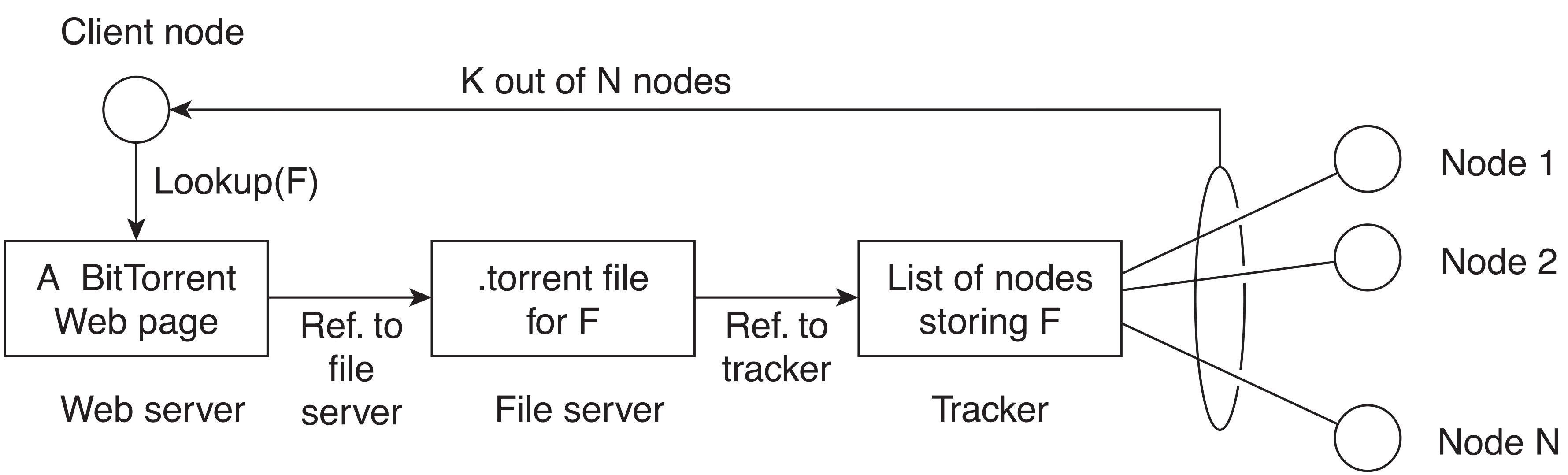
Hybrid architectures

- Client-server combined with decentralized architectures
  - Collaborative distributed systems: e.g., BitTorrent, which supports parallel downloading and uploading of chunks of a file. First, interact with C/S system, then operate in decentralized manner.

  - Edge-server systems: e.g., Content Delivery Network (CDN), edge servers at ISPs act as servers to their clients, but cooperate with other edge servers to host shared content

BitTorrent

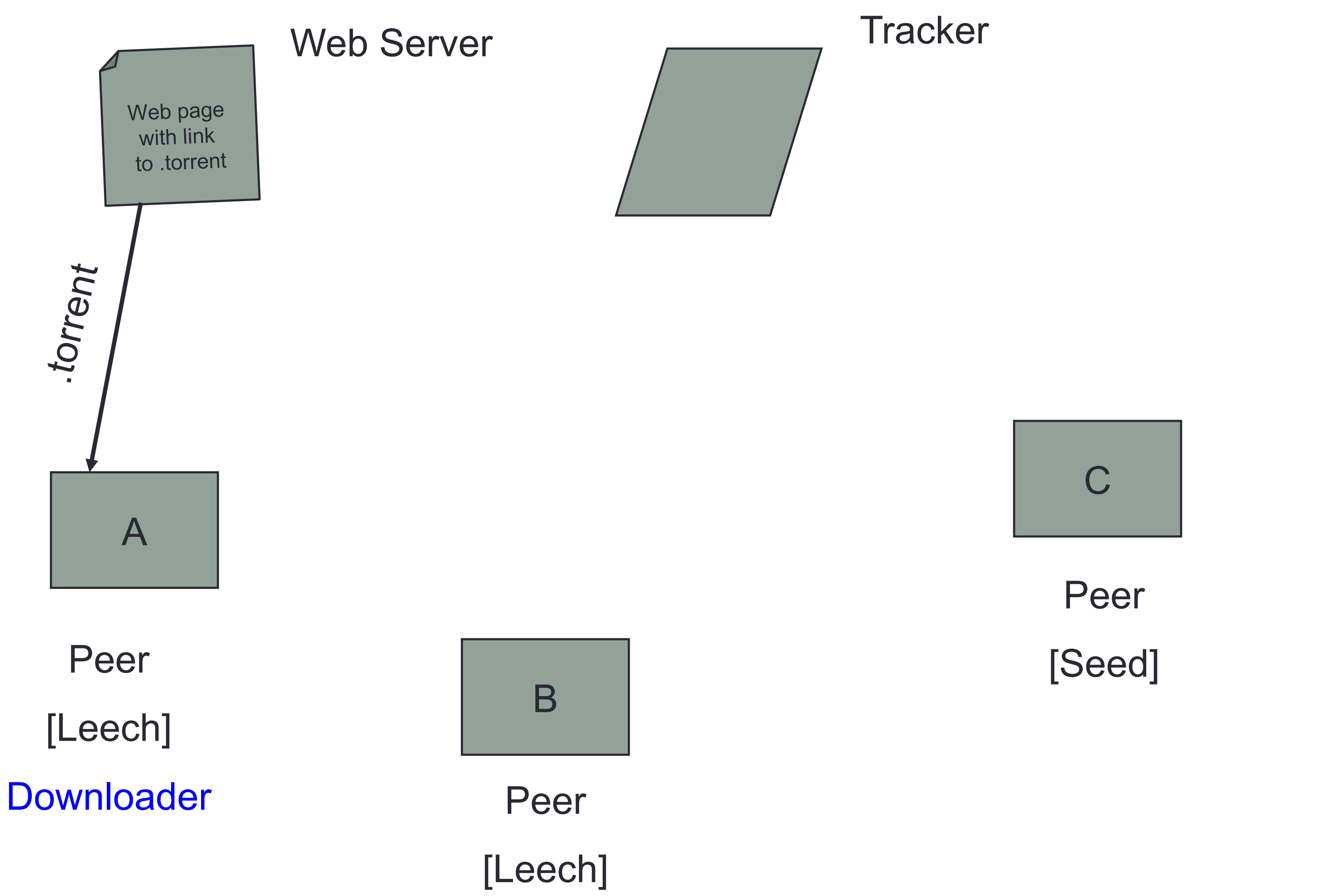
- A node joins a swarm of downloaders, who in parallel get file chunks from the source, but also distributed these chunks amongst each other.



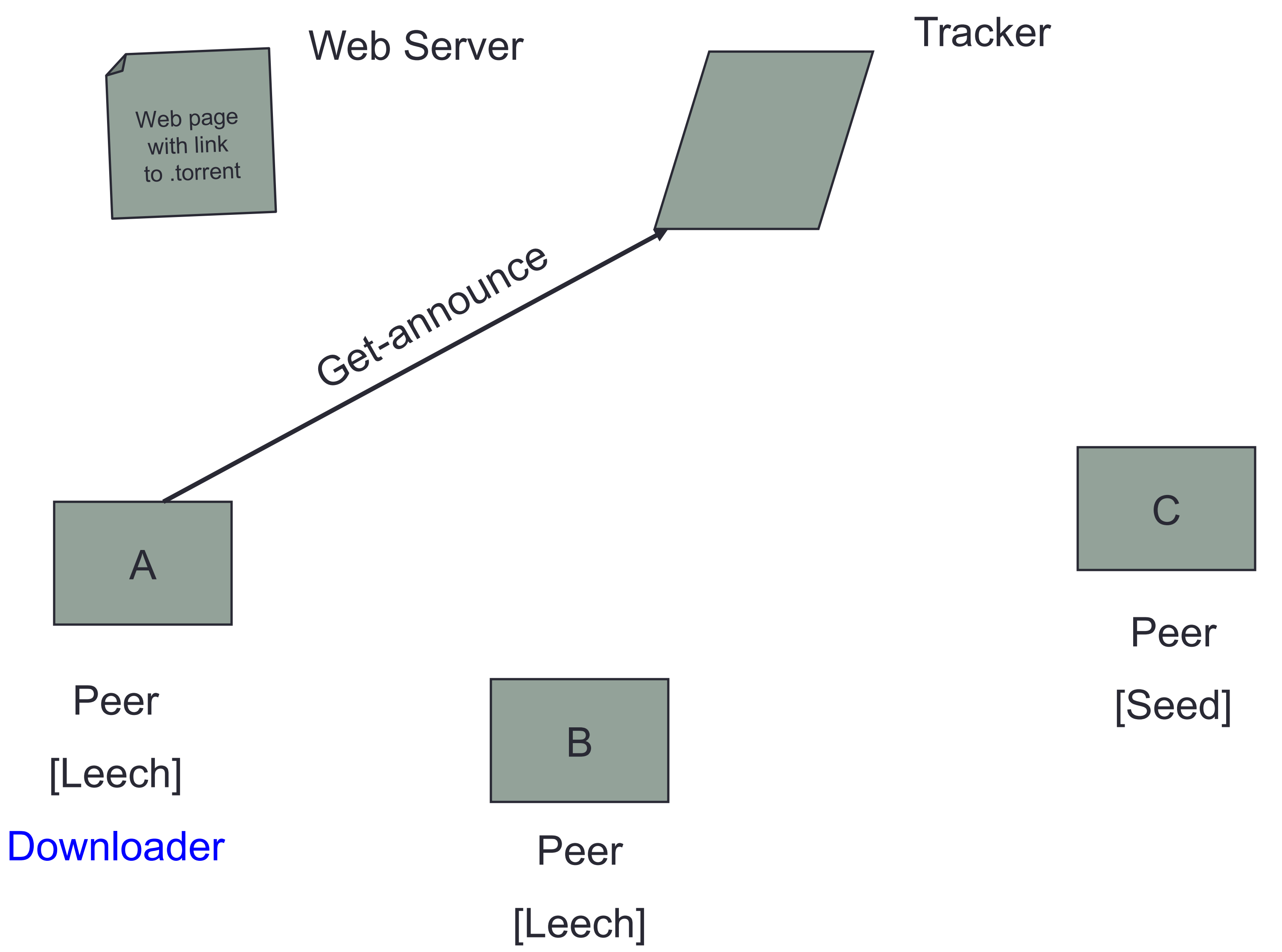
BitTorrent

- File divided into chunks, e.g., 256 KB.
- Peers contact a global directory (web server) to locate a *.torrent* file with the information needed to locate a **tracker**
- The tracker supplies a list of active peers that have chunks of the desired file.
- Using information from the tracker, peers can download the file in chunks from multiple peers in the network. Peers must also provide file chunks to other users.

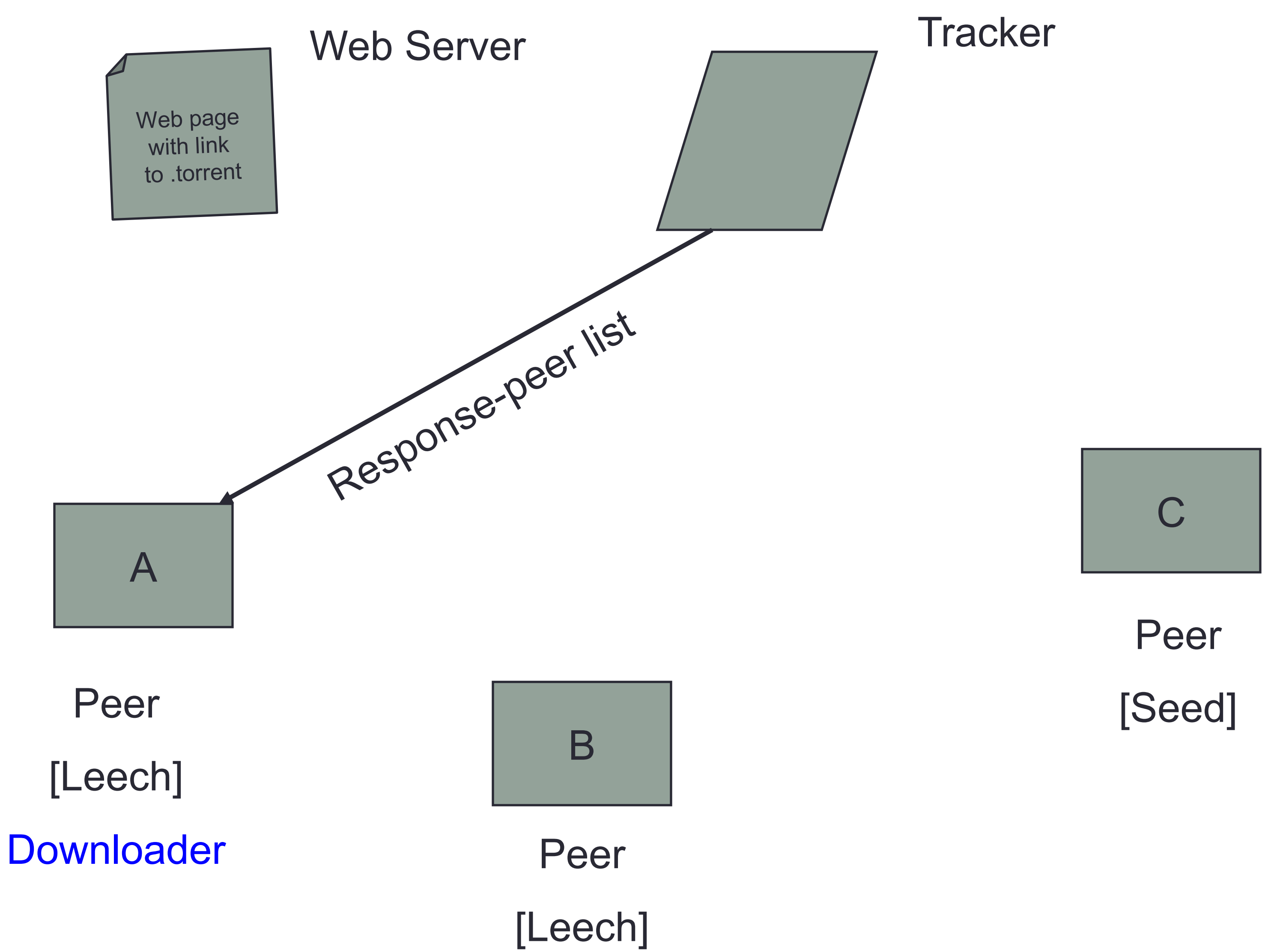
BitTorrent


Web Server
Web page with link to .torrent
Tracker
.torrent
A
Peer
[Leech]
Downloader
B
Peer
[Leech]
C
Peer
[Seed]

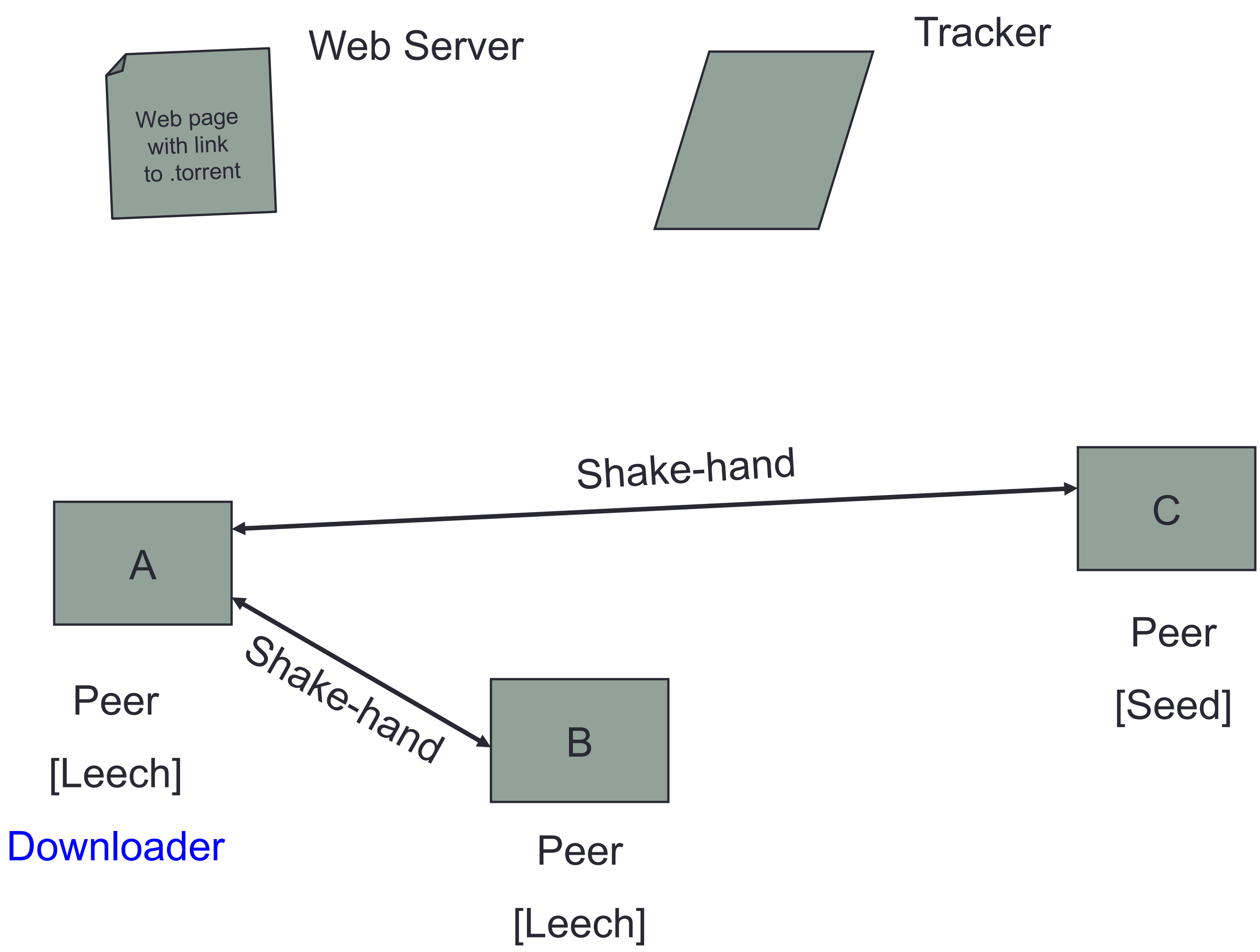
BitTorrent


Web Server
Web page
with link
to .torrent
Tracker
Get-announce
A
Peer
[Leech]
Downloader
B
Peer
[Leech]
C
Peer
[Seed]

BitTorrent

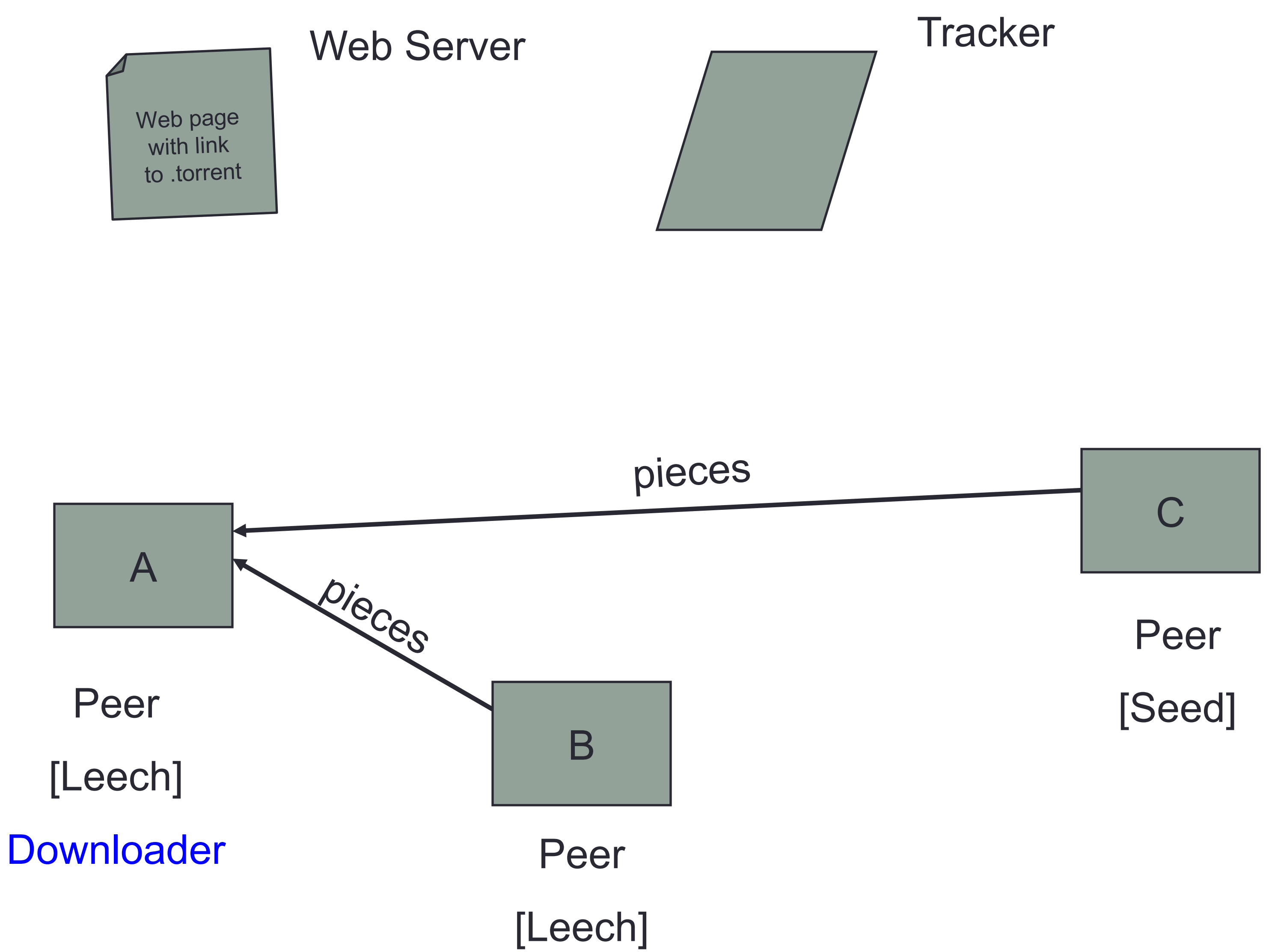

Web page
with link
to .torrent
Web Server
Tracker
Response-peer list
A
Peer
[Leech]
Downloader
B
Peer
[Leech]
C
Peer
[Seed]

BitTorrent



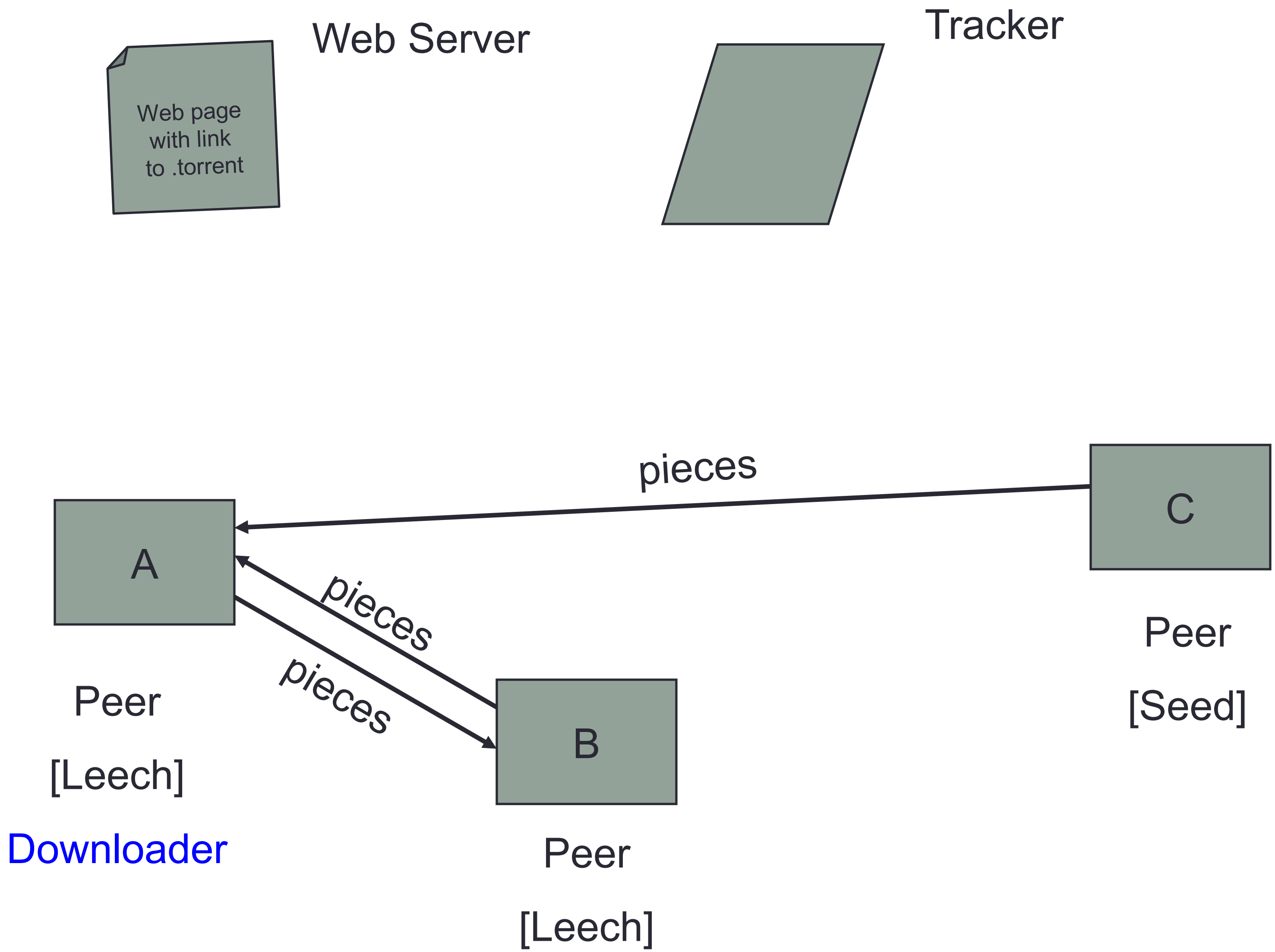
Web Server
Web page with link to .torrent
Tracker
Shake-hand
C
A
Shake-hand
B
Peer
[Seed]
Peer
[Leech]
Downloader
Peer
[Leech]

BitTorrent

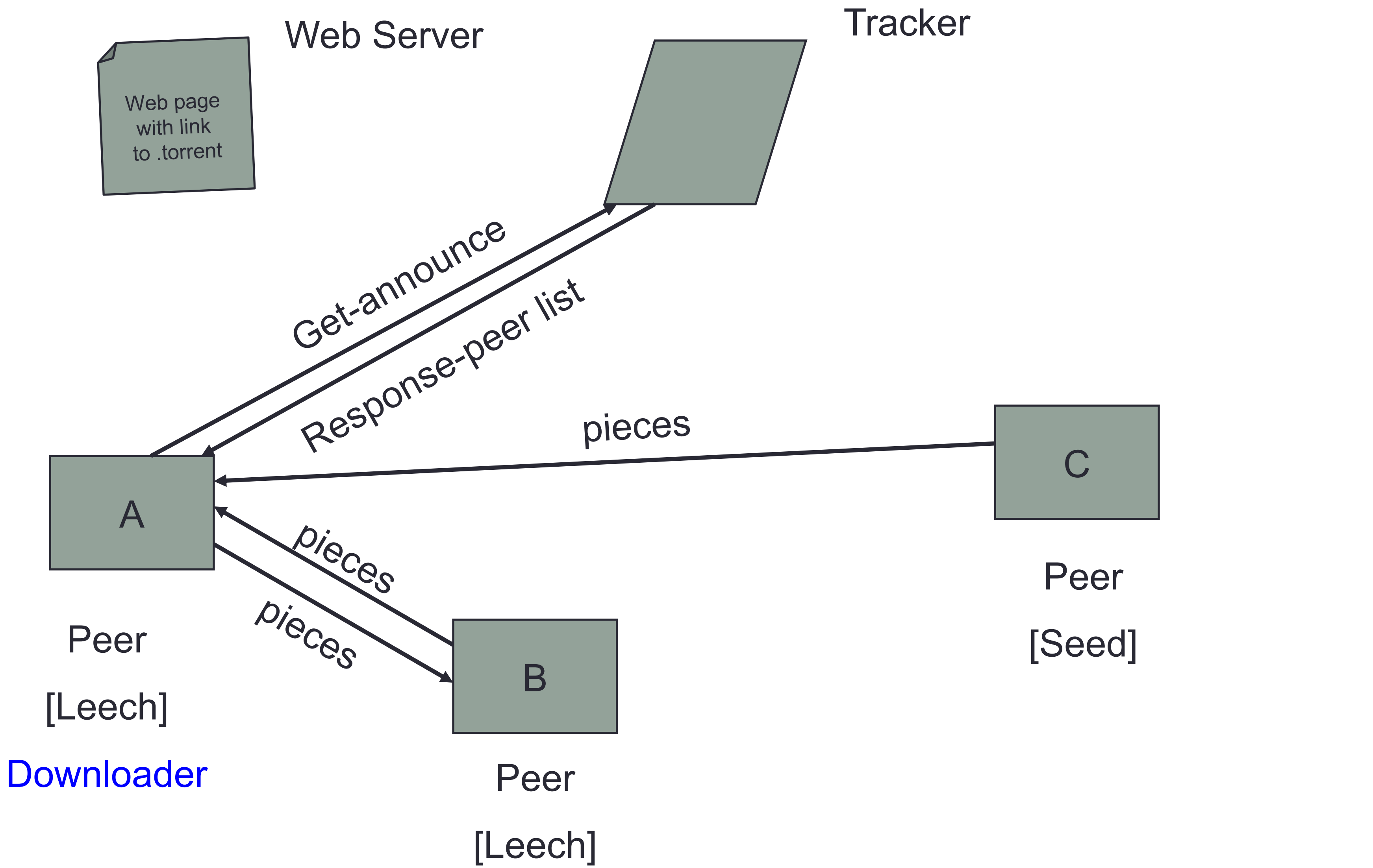


Web Server
Web page with link to .torrent
Tracker
pieces
pieces
A
C
B
Peer
[Leech]
Downloader
Peer
[Seed]
Peer
[Leech]

BitTorrent


Web page with link to .torrent
Web Server
Tracker
pieces
C
A
pieces
pieces
B
Peer
[Seed]
Peer
[Leech]
Downloader
Peer
[Leech]

BitTorrent


Web page with link to .torrent
Web Server
Tracker
Get-announce
Response-peer list
pieces
A
C
pieces
pieces
B
Peer
[Seed]
Peer
[Leech]
Downloader
Peer
[Leech]

Requesting chunks in BitTorrent

- At any given time, different peers have different subsets of file chunks
- Periodically, Alice asks all known peers for list of chunks that they have
- Alice requests missing chunks from peers, **rarest chunks first**

Sending chunks in BitTorrent

Tit-for-Tat:

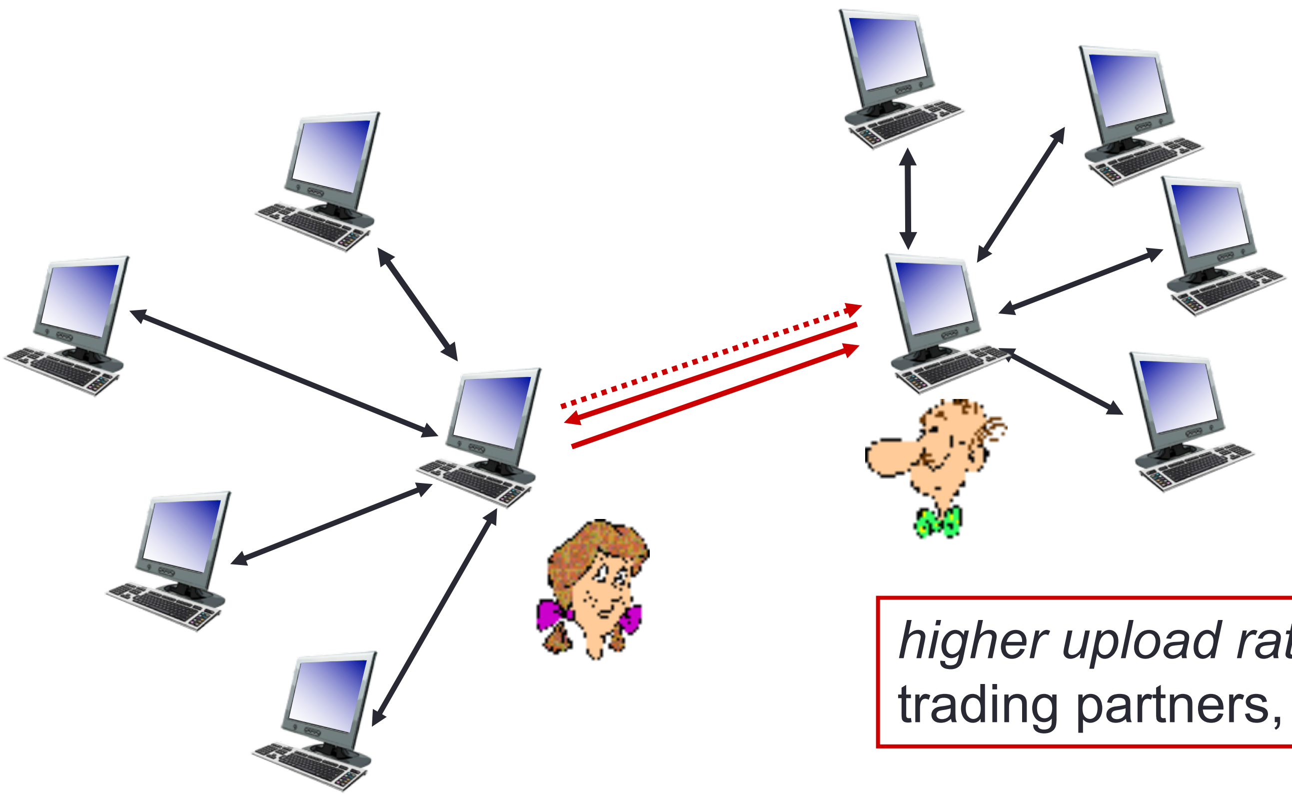
- Alice sends chunks to those four peers currently sending her chunks at highest rate
- Other peers are choked by Alice (do not receive chunks from her)
- Re-evaluate top 4 every 10 secs
- Every 30 secs: randomly select another peer, starts sending chunks -- “optimistically un-choke” this peer
- Newly chosen peer may join top 4

BitTorrent: Tit-for-Tat

(1) Alice "optimistically un-chokes" Bob

(2) Alice becomes one of Bob's top-4 providers; Bob reciprocates

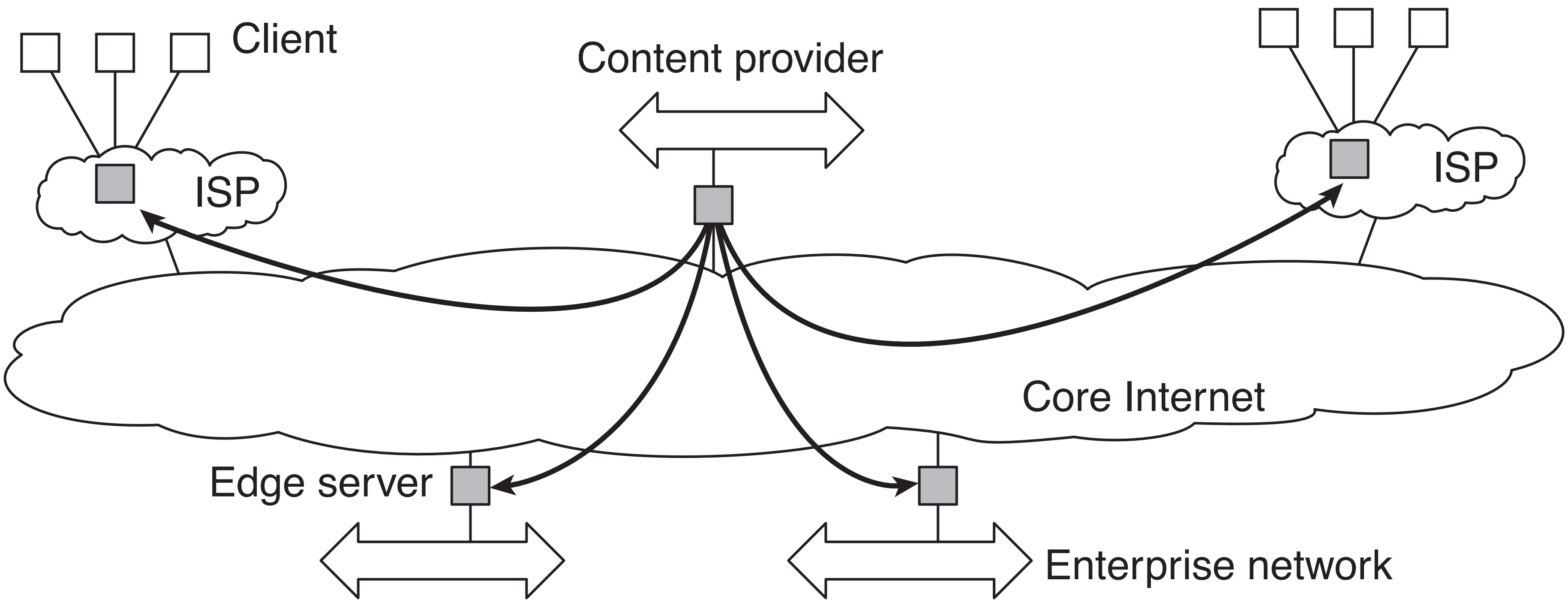
(3) Bob becomes one of Alice's top-4 providers



higher upload rate: find better trading partners, get file faster !

Hybrid architectures: edge-server systems

- Cooperative caching



P2P vs. Client/Server

- P2P computing allows end users to communicate without a dedicated server.
- There is less likelihood of performance bottlenecks since communication is more distributed.
  - Data distribution leads to workload distribution.
- Resource discovery is more difficult than in centralized client-server computing & look-up/retrieval is slower
- P2P can be more fault tolerant, more resistant to denial of service attacks because network content is distributed.
  - Individual hosts may be unreliable, but overall, the system should maintain a consistent level of service

Reading

- Chapter 2 and Section 5.2.3 of TBook
- Articles on P2P systems on myCourses