Example topic 1

- Peer-to-peer

P2P file sharing

Notes based on notes by K.W. Ross, J. Kurose, D. Rubenstein, and others

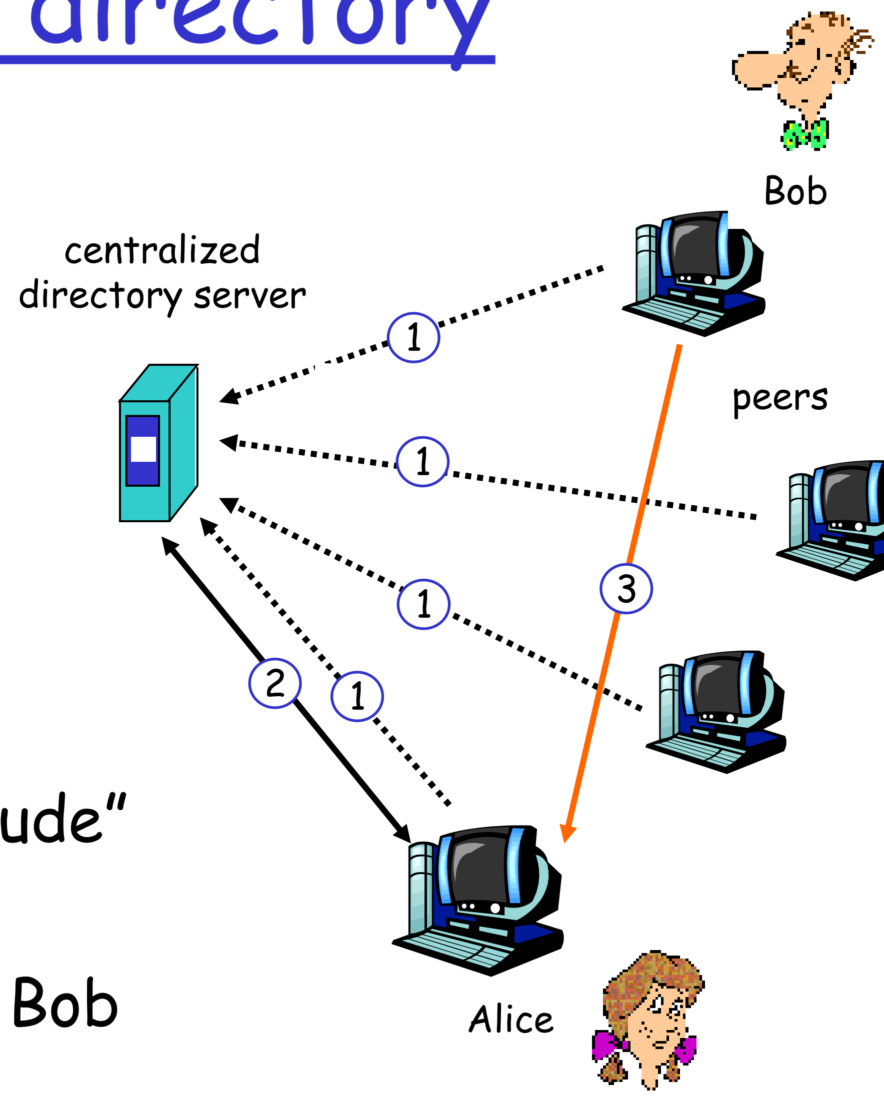
P2P: centralized directory

Original "Napster" design

1) When peer connects, it informs central server:
   - IP address
   - content

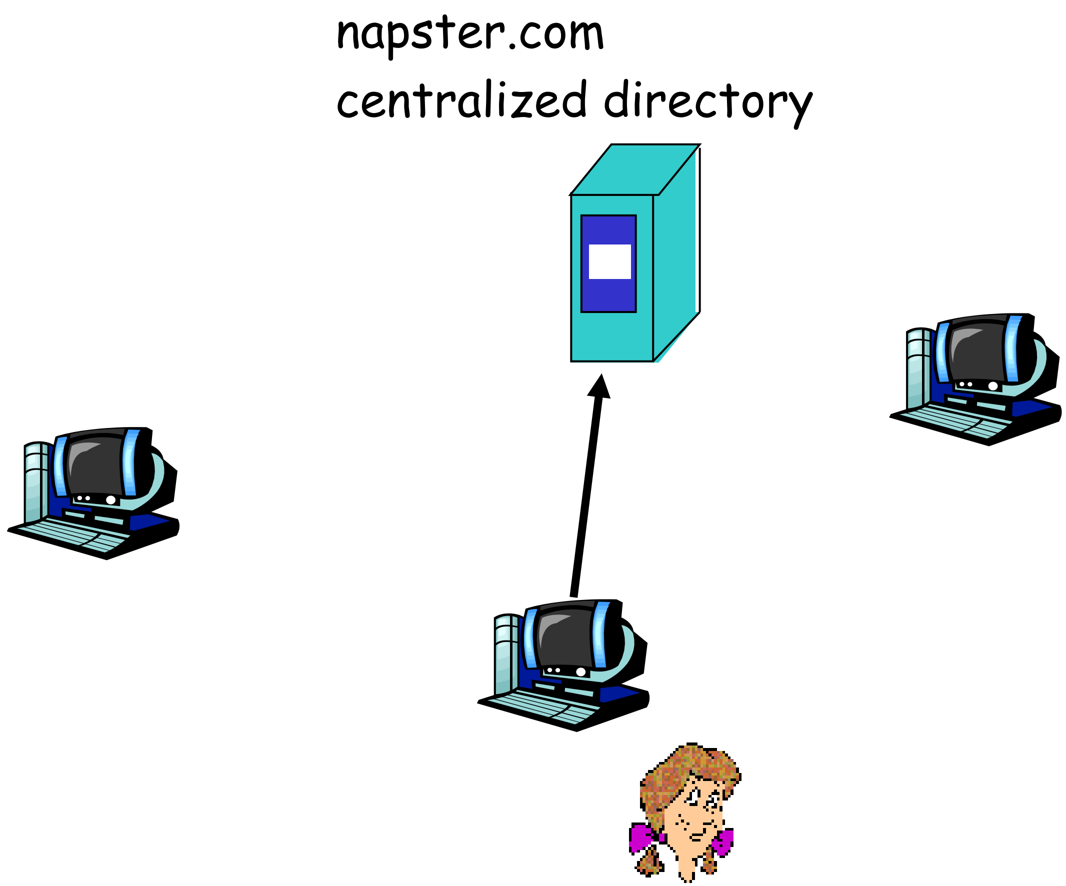
2) Alice queries for "Hey Jude"

3) Alice requests file from Bob



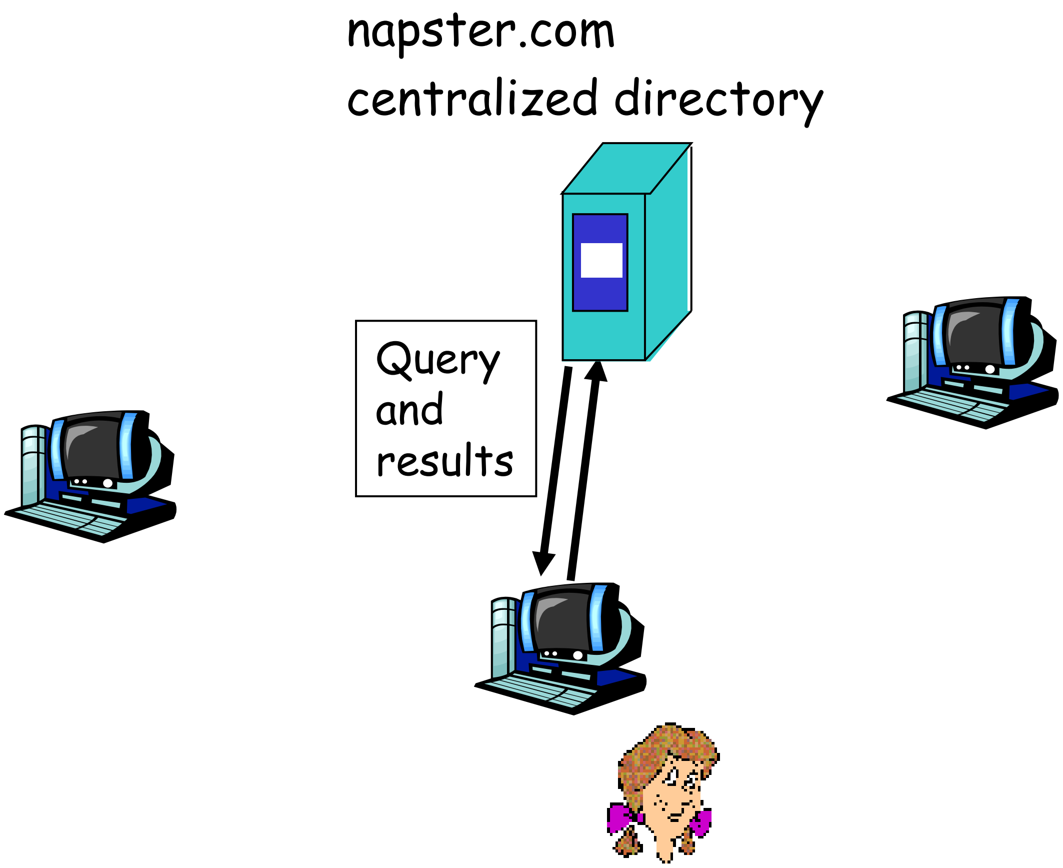
Napster

1. File list and IP address is uploaded



Napster

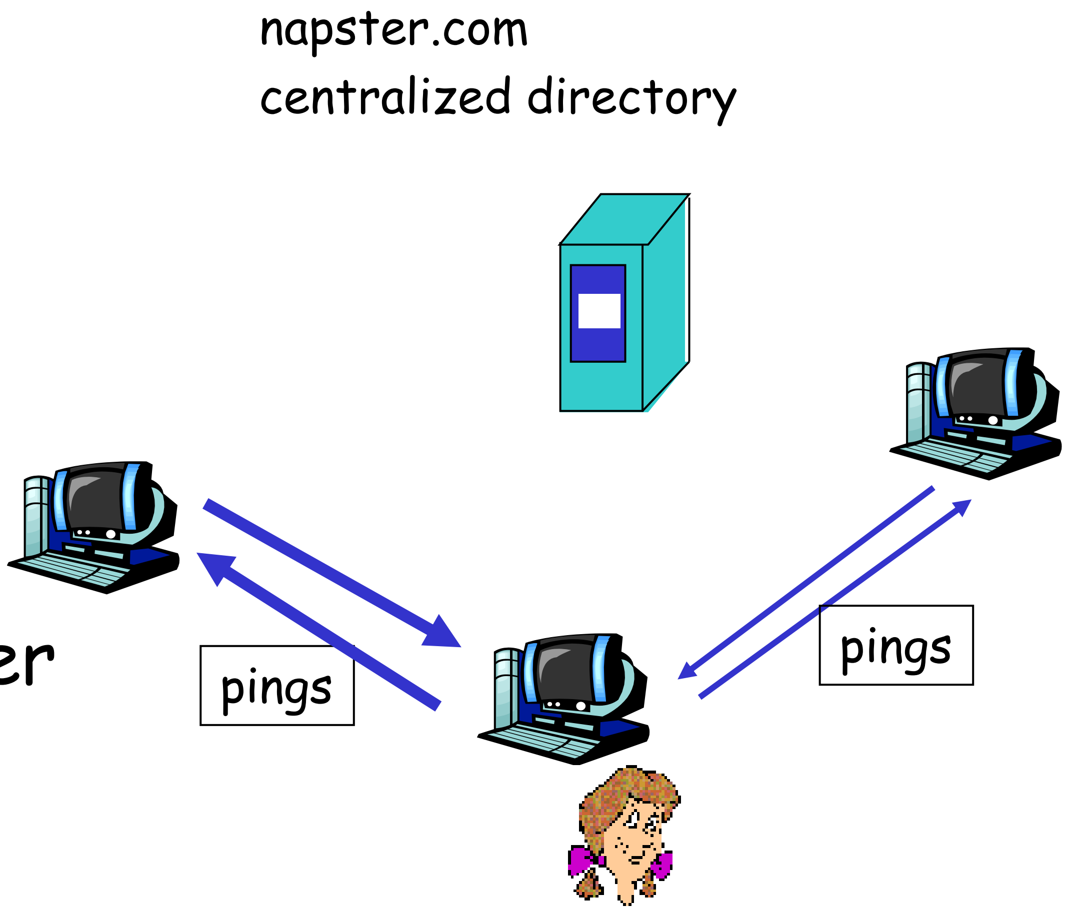
2. User requests search at server.



Napster

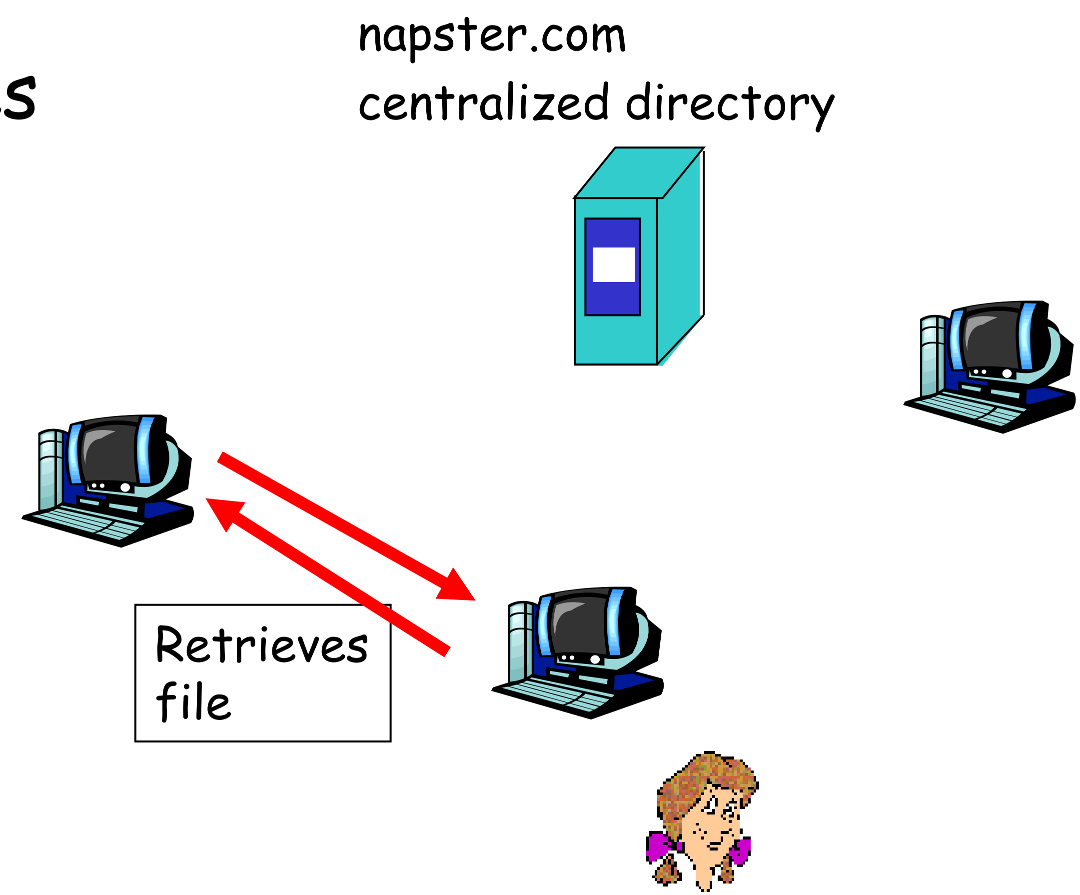
3. User pings hosts that apparently have data.

   Looks for ***best*** transfer rate.



Napster

4. User chooses server



Napster's centralized server farm had difficult time keeping up with traffic

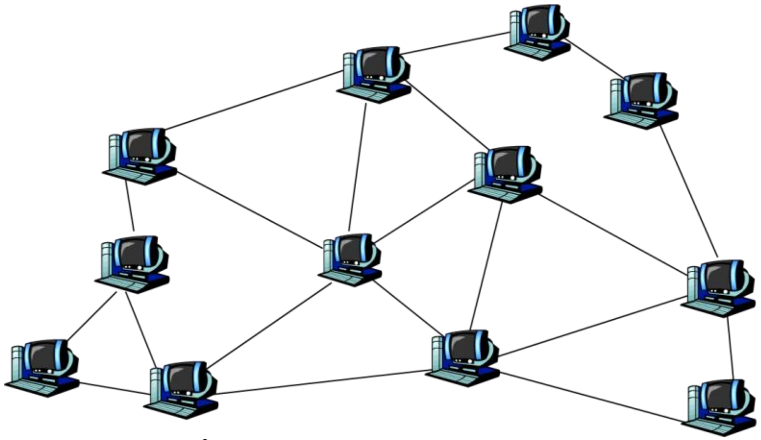
P2P: problems with centralized directory

- single point of failure
- performance bottleneck
- copyright infringement: “target” of lawsuit is obvious

file transfer is decentralized, but locating content is highly centralized

Unstructured P2P: Gnutella

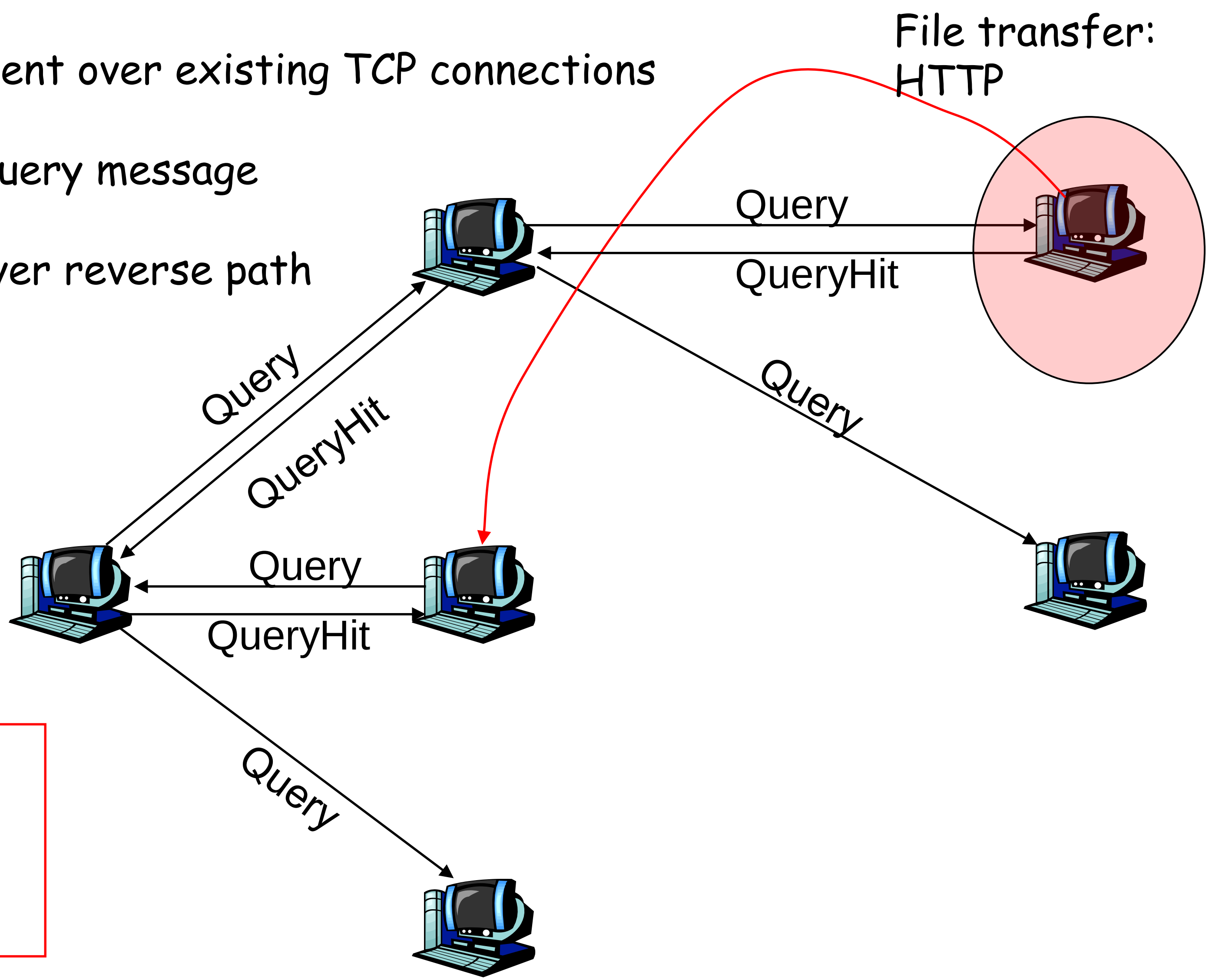
- **Focus:** decentralized method searching for files
  - central directory server no longer the bottleneck
  - more difficult to "pull plug"



- Each application instance serves to:
  - store selected files
  - route queries from and to its neighboring peers
  - respond to queries if file stored locally
  - serve files

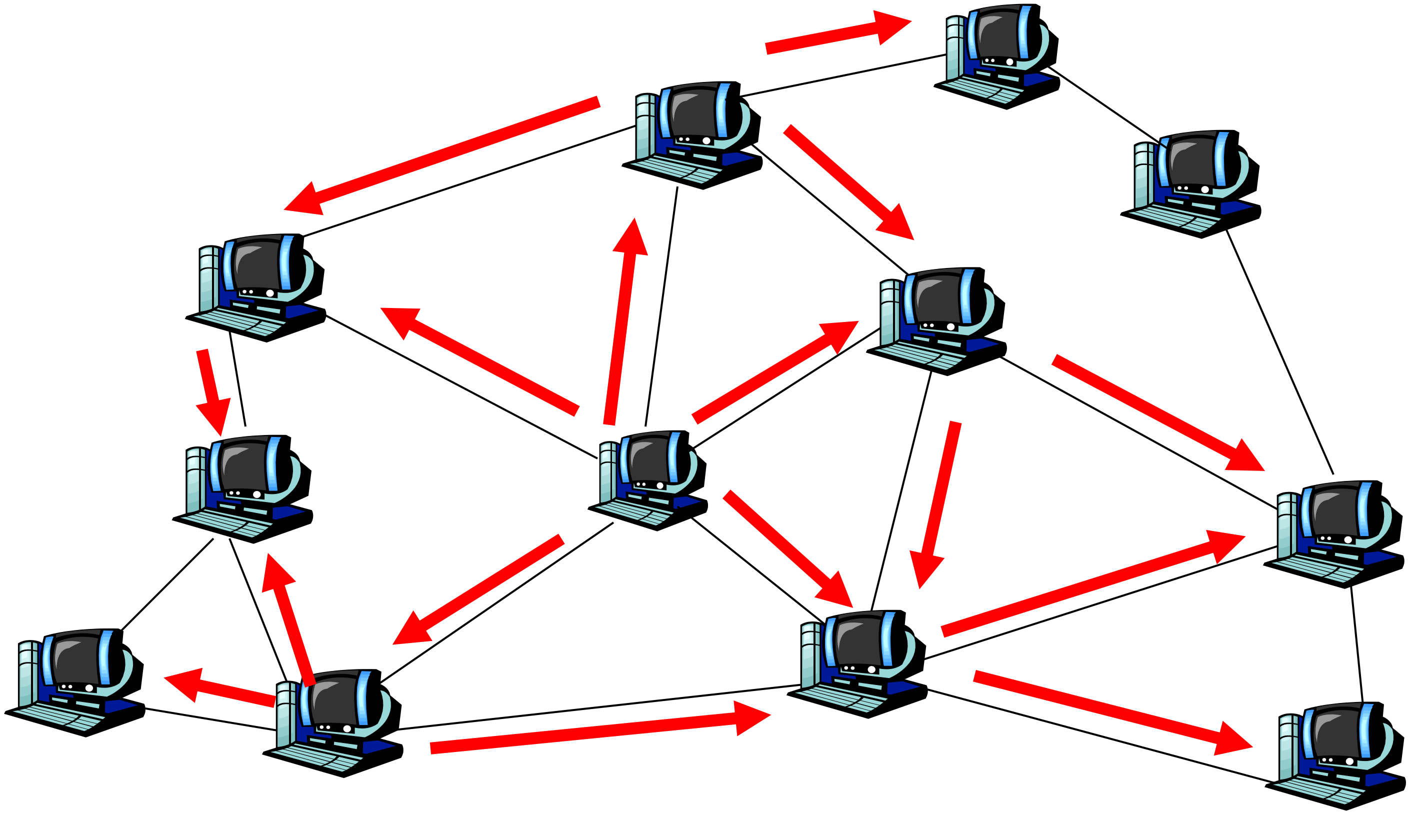
Gnutella: protocol

- Query message sent over existing TCP connections
- Peers forward Query message
- QueryHit sent over reverse path

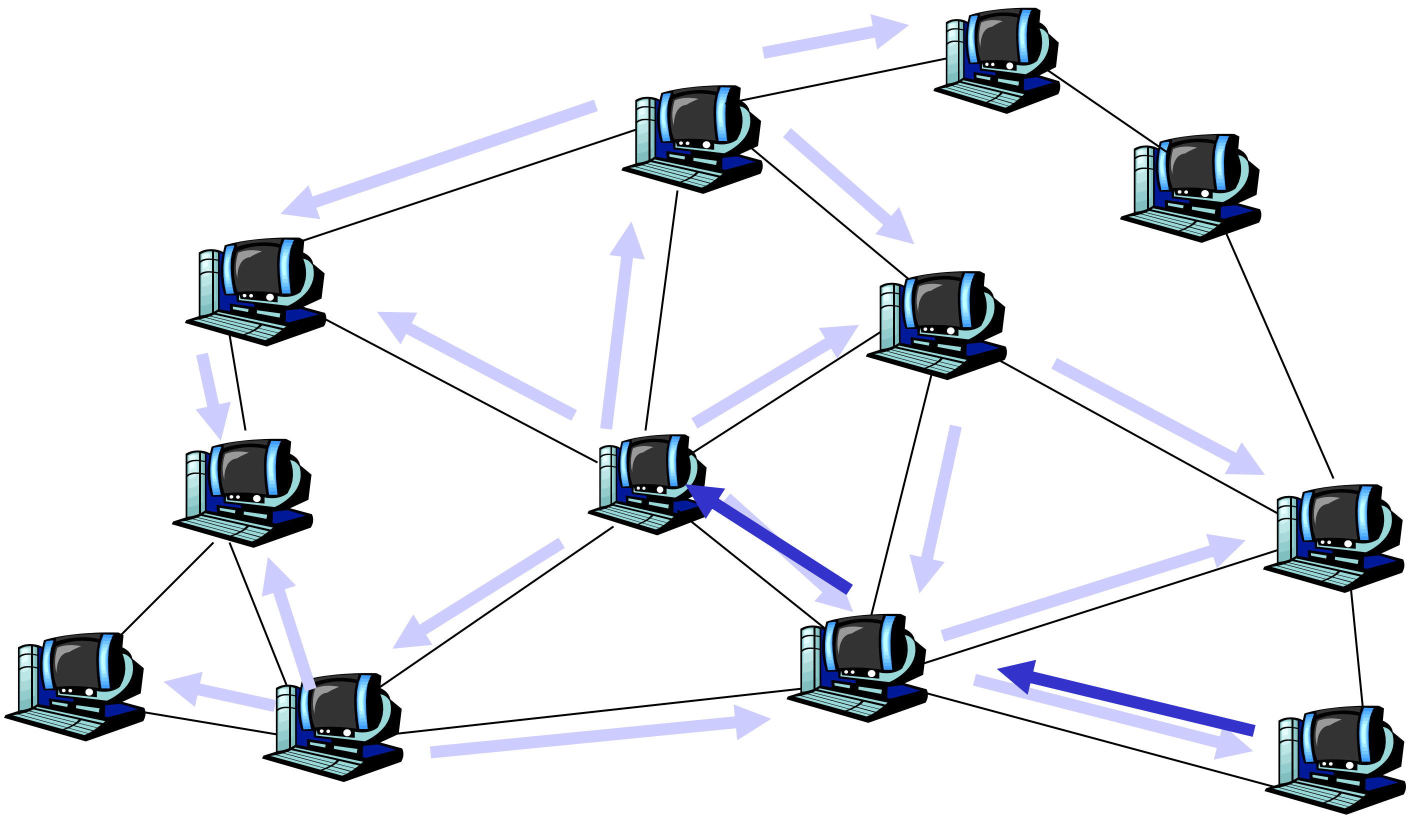


Scalability:
limited scope
flooding

Distributed Search/Flooding

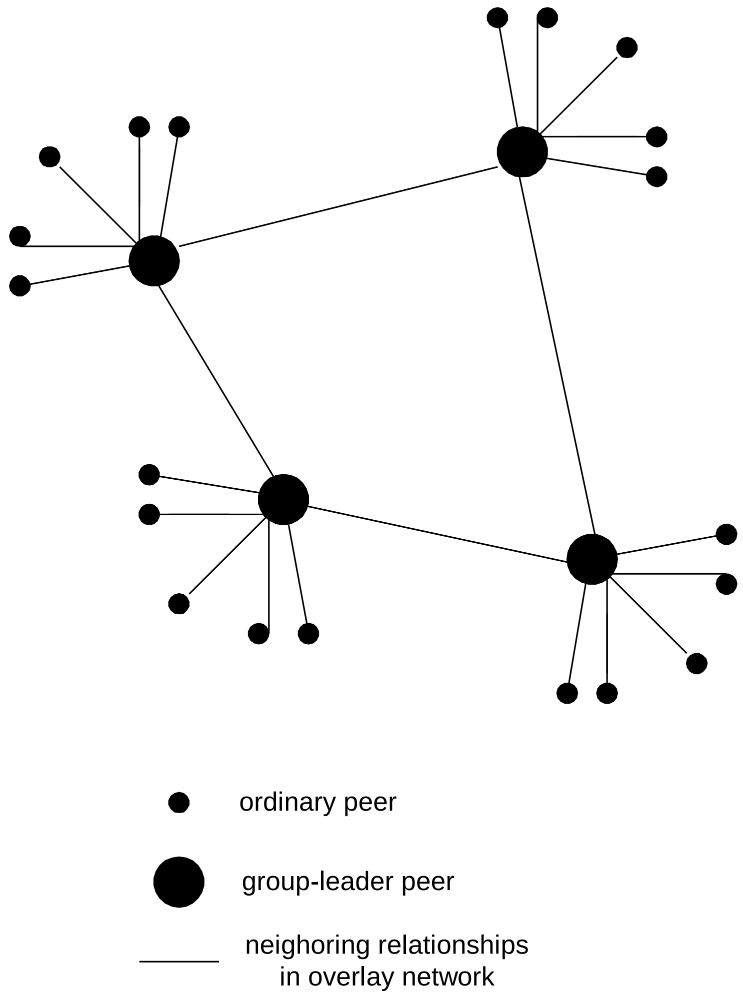


Distributed Search/Flooding

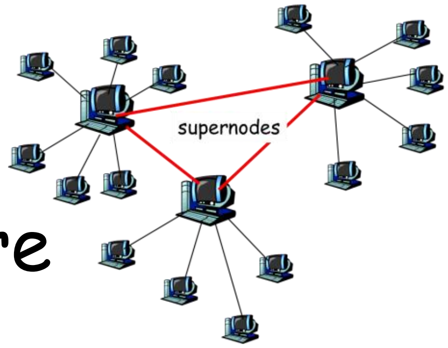


Hierarchical Overlay

- Between centralized index, query flooding approaches
- Each peer is either a *group leader* or assigned to a group leader
  - TCP connection between peer and its group leader
  - TCP connections between some pairs of group leaders
- Group leader tracks content in its children



Example: KaZaA Architecture (2)



- Nodes that have more connection bandwidth and are more available are designated as "supernodes"
- Each supernode acts as a mini-Napster hub, tracking the content and IP addresses of its descendants
-
-

Parallel Downloading; Recovery

- If file is found in multiple nodes, user can select parallel downloading
- Most likely HTTP byte-range header used to request different portions of the file from different nodes

- Automatic recovery when server peer stops sending file

KaZaA Corporate Structure

- Software developed by FastTrack in Amsterdam
- FastTrack also deploys KaZaA service
- FastTrack licenses software to Music City (Morpheus) and Grokster
- Later, FastTrack terminates license, leaves only KaZaA with killer service
- Summer 2001, Sharman networks, founded in Vanuatu (small island in Pacific), acquires FastTrack
  - Board of directors, investors: secret
- Employees spread around, hard to locate
- Code in Estonia

Lessons learned from KaZaA

KaZaA provides powerful file search and transfer service <u>without</u> server infrastructure

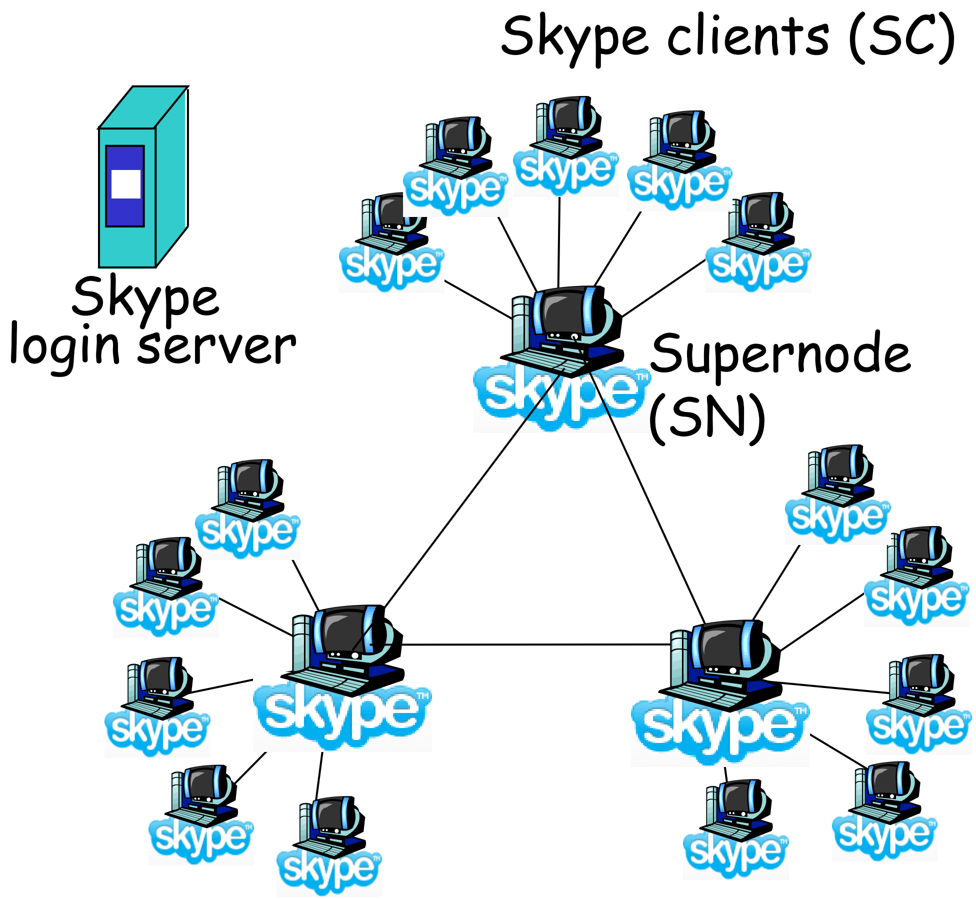
- <u>Exploit heterogeneity</u>
- Provide automatic recovery for interrupted downloads
- Powerful, intuitive user interface

<u>Copyright infringement</u>

- International cat-and-mouse game
- With distributed, serverless architecture, can the plug be pulled?
- Prosecute users?
- Launch DoS attack on supernodes?
- Pollute?

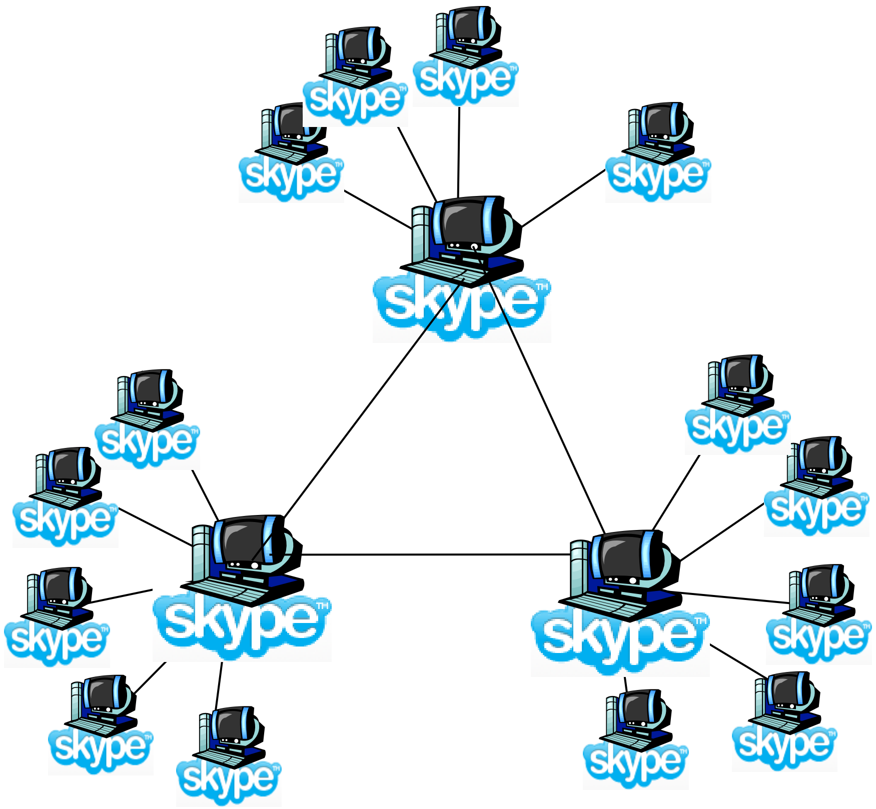
P2P Case study: Skype

- Inherently P2P: pairs of users communicate.
- Proprietary application-layer protocol (inferred via reverse engineering)
- Hierarchical overlay with Supernodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs



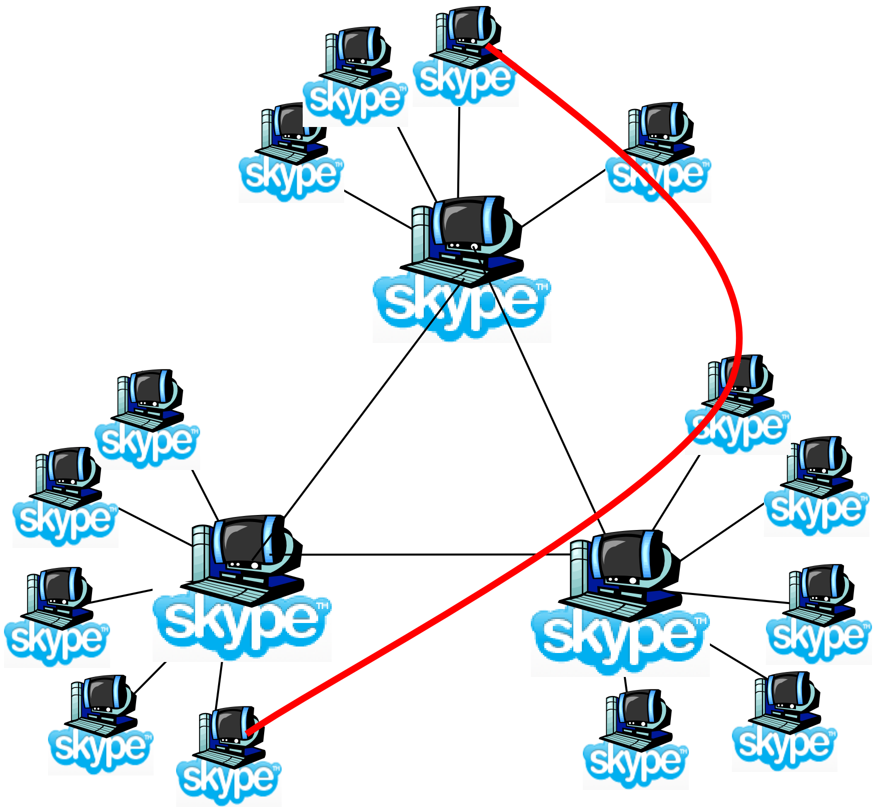
Peers as relays

- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer



Peers as relays

- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer
- Solution:
  - Using Alice's and Bob's SNs, Relay is chosen
  - Each peer initiates session with relay.
  - Peers can now communicate through NATs via relay



Structured p2p systems

Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) pairs

Key	Value
00	
01	
10	
11	

DHT Identifiers

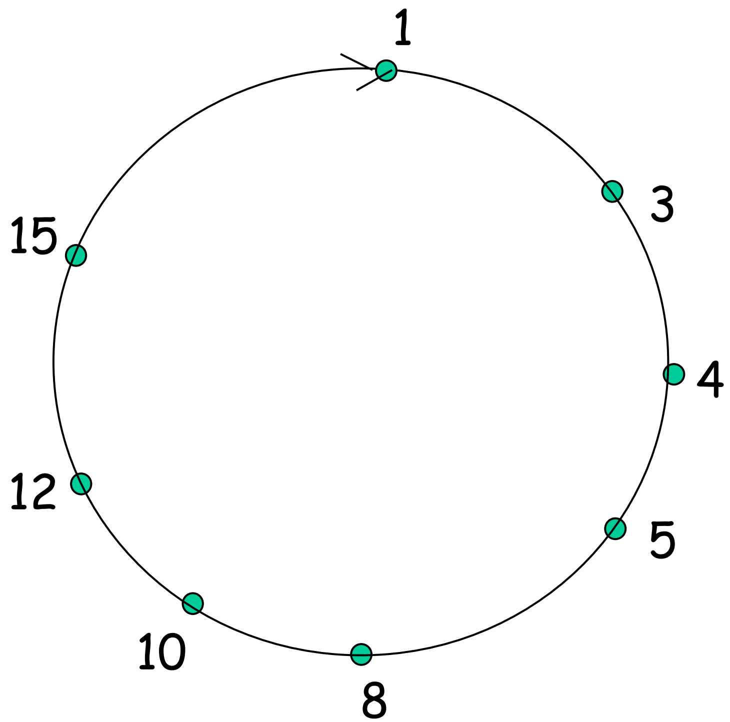
Key	Value
000000	
000001	
000002	
…	
ffffff	

- Assign integer identifier to each peer in range $[0,2^n-1]$
    - Each identifier can be represented by n bits.
- Require each key to be an integer in same range.
- To get integer keys, hash original key.
    - E.g., key = h("Led Zeppelin IV")
    - This is why they call it a distributed "hash" table

How to assign keys to peers?

- Central issue:
  - Assigning (key, value) pairs to peers.
- Rule: Assign key to the peer that has the closest ID.
- Convention in lecture: closest is the closest successor of the key.
- Ex: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

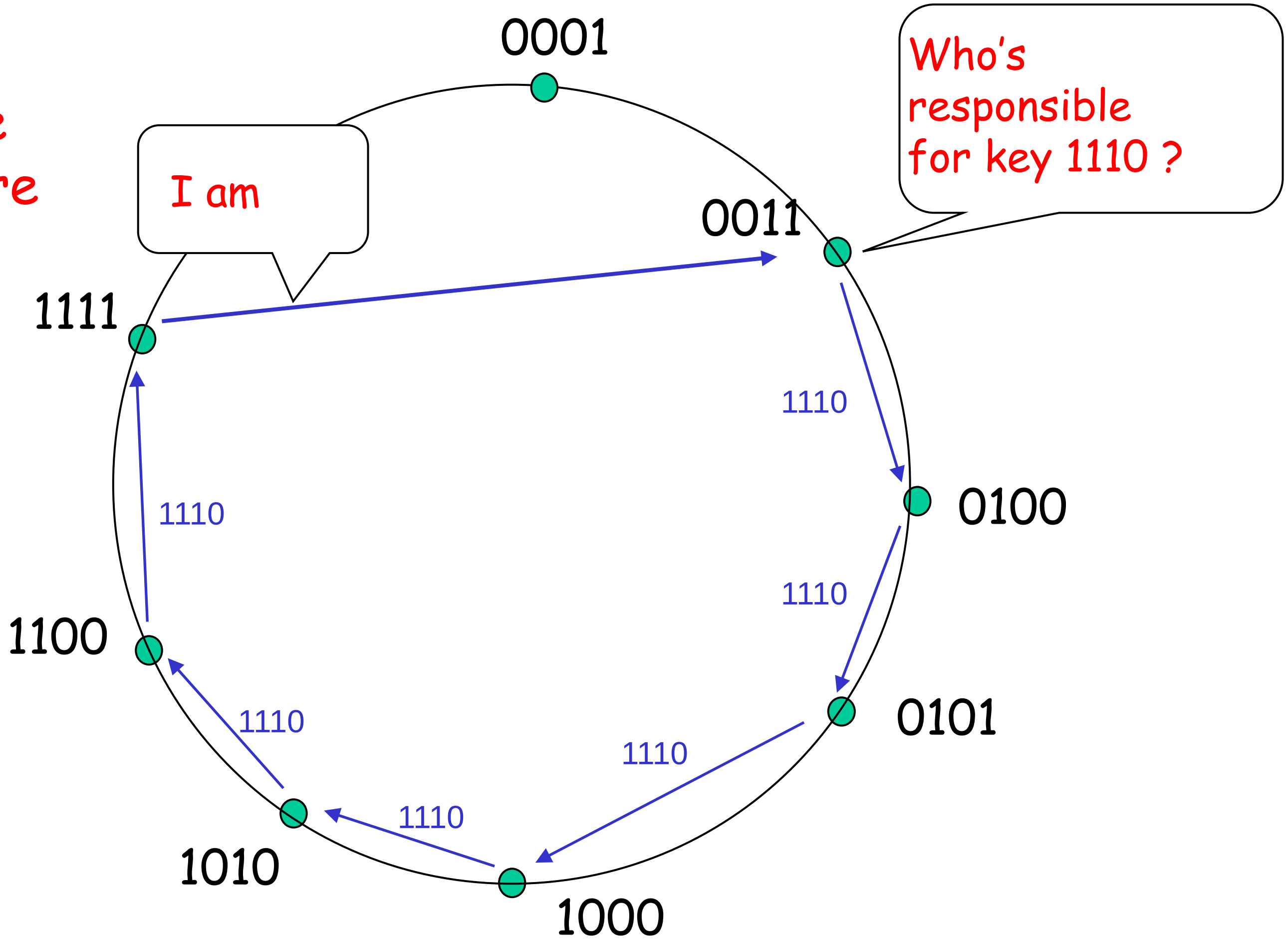
Circular DHT (1)



- Each peer *only* aware of immediate successor and predecessor.
- “Overlay network”

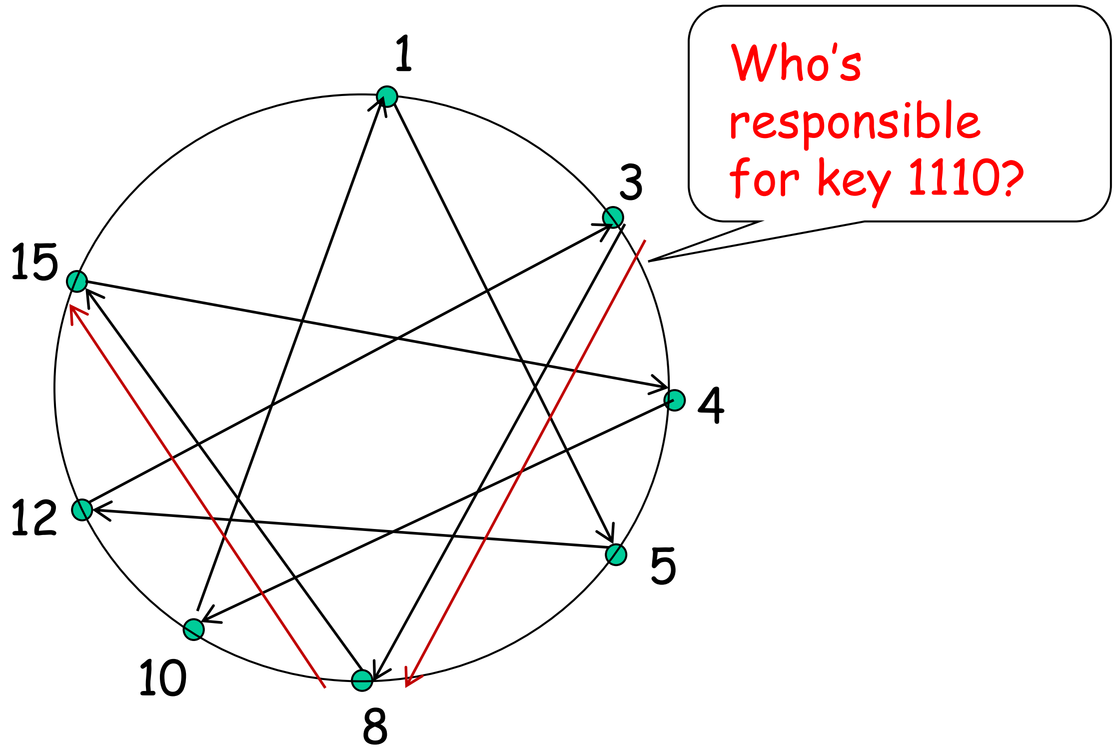
Circle DHT (2)

O(N) messages on avg to resolve query, when there are N peers



Define closest as closest successor

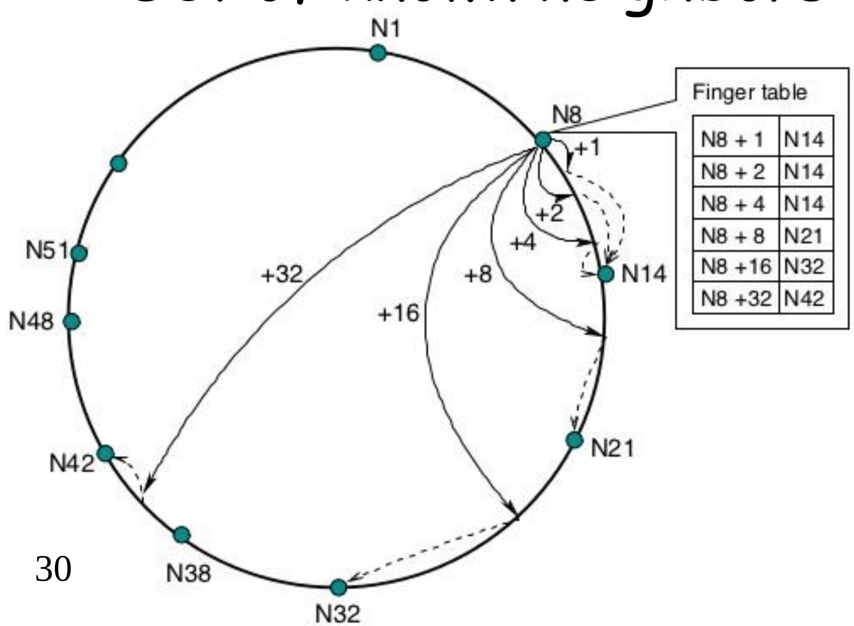
Circular DHT with Shortcuts

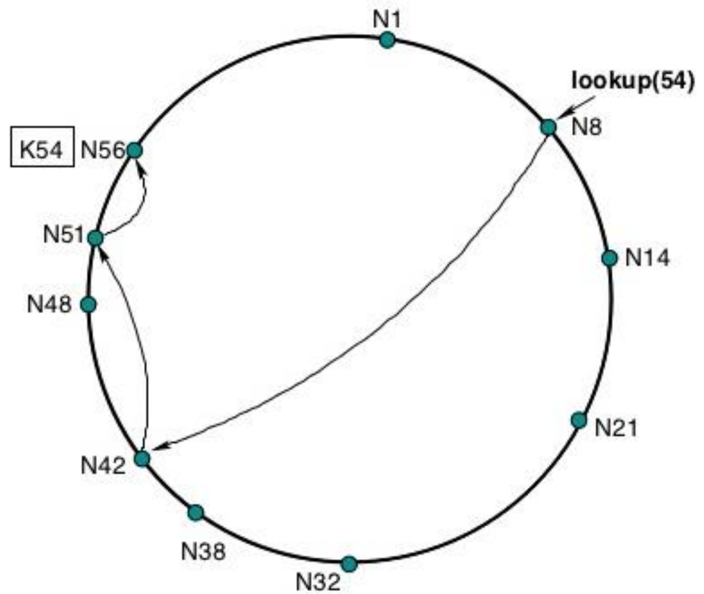


- Each peer keeps track of IP addresses of predecessor, successor, short cuts.
  - E.g., Example above reduced from 6 to 2 messages.
- Possible to design shortcuts so O(log N) neighbors, O(log N) messages in query

Example: Chord Routing [see paper]

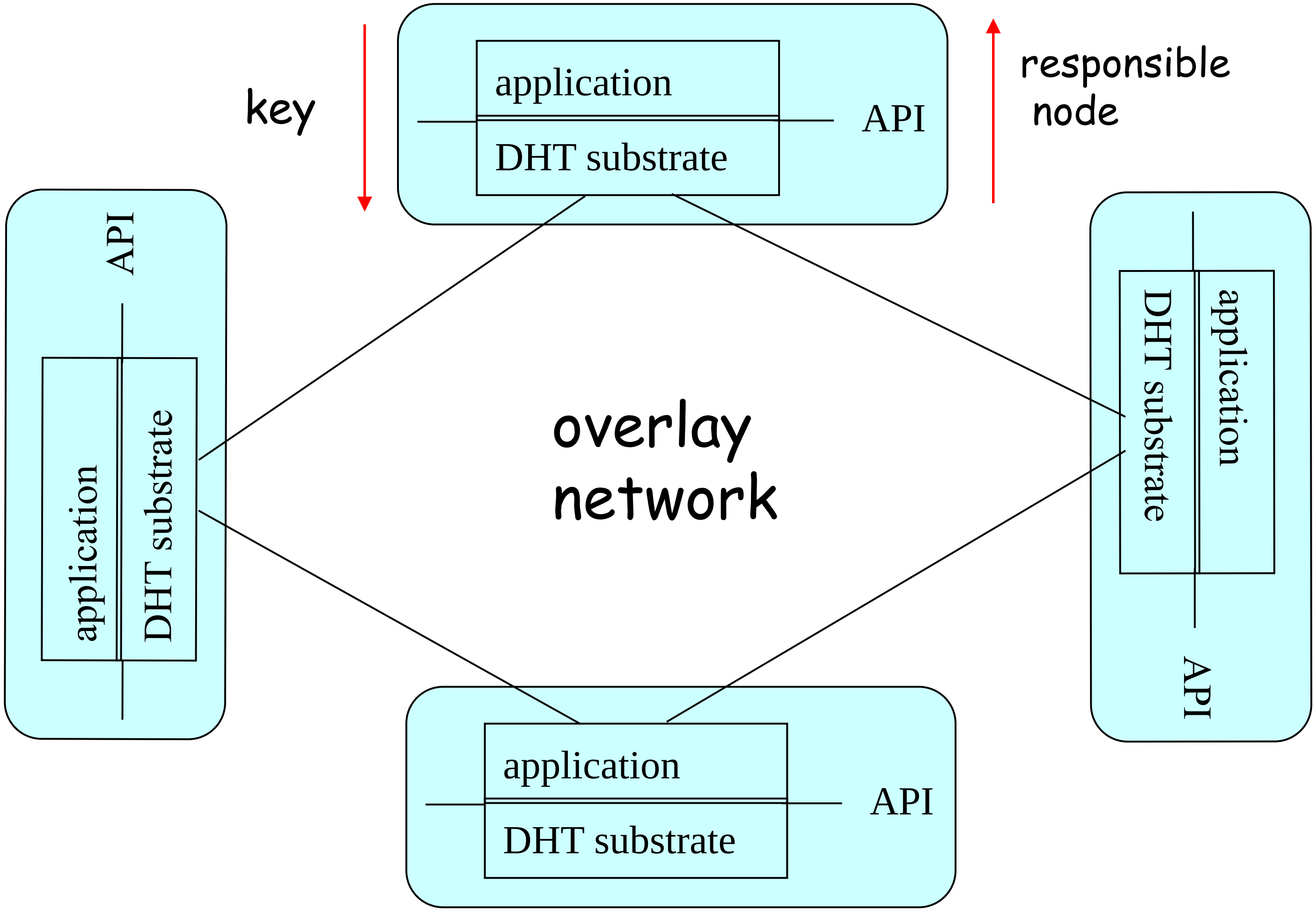
- A node s's i^{th} neighbor has the ID that is equal to $s+2^i$ or is the next largest ID (mod ID space), i≥0
- To reach the node handling ID t, send the message to neighbor $\#\log_2(t-s)$
- Requirement: each node s must know about the next node that exists clockwise on the Chord (0^{th} neighbor)
- Set of known neighbors called a finger table



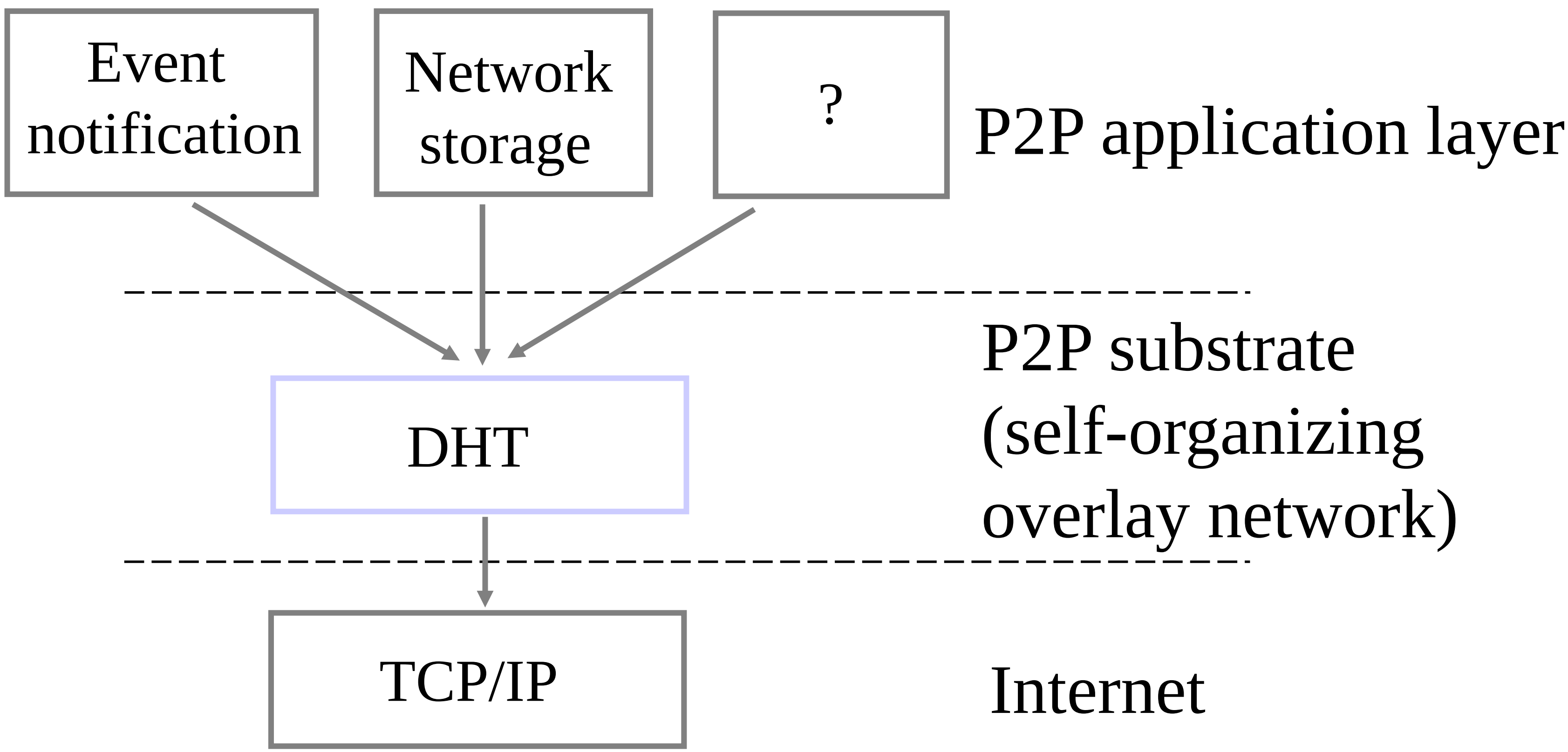


DHT API

each data item (e.g., file or metadata pointing to file copies) has a key

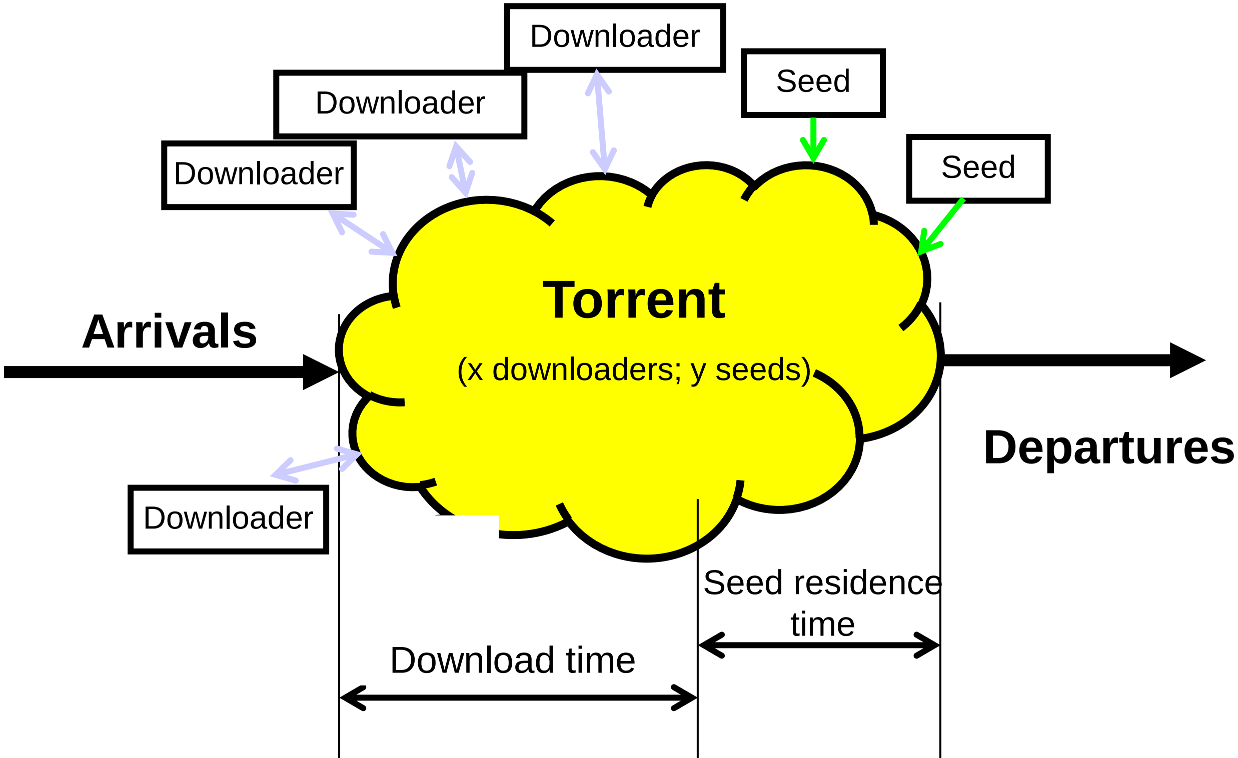


DHT Layered Architecture



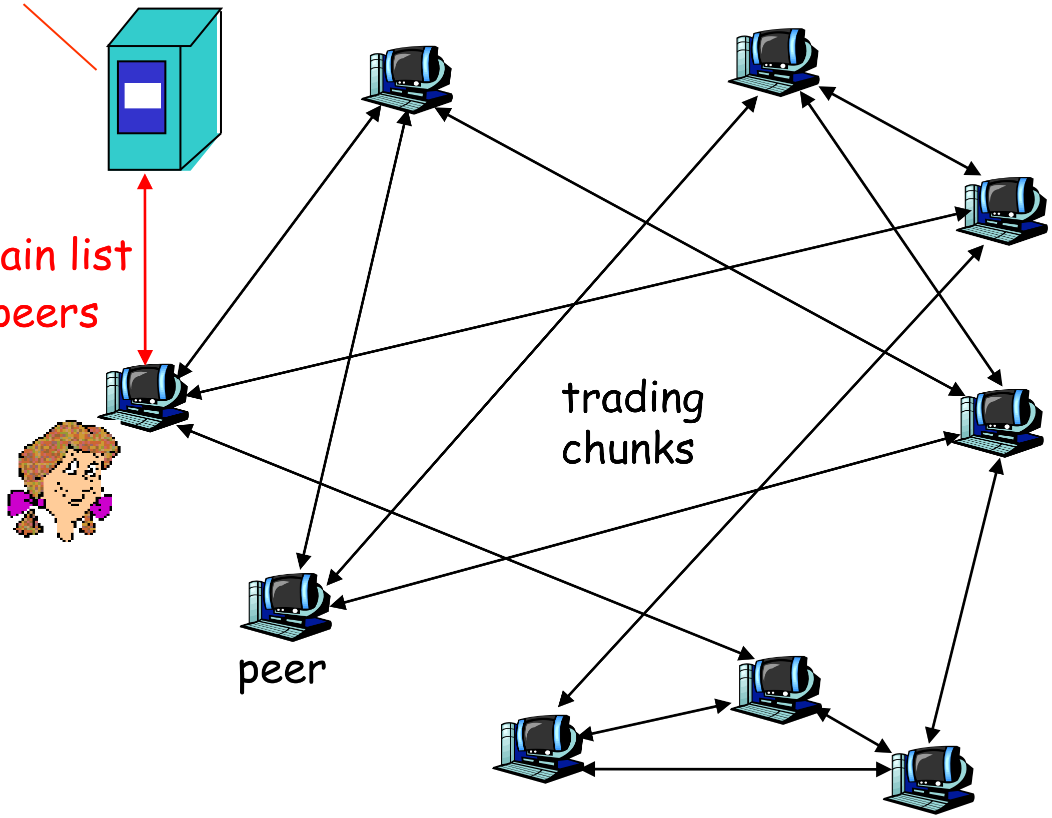
BitTorrent-like systems

- File split into many smaller pieces
- Pieces are downloaded from both seeds and downloaders
- Distribution paths are dynamically determined
  - Based on data availability



File distribution: BitTorrent

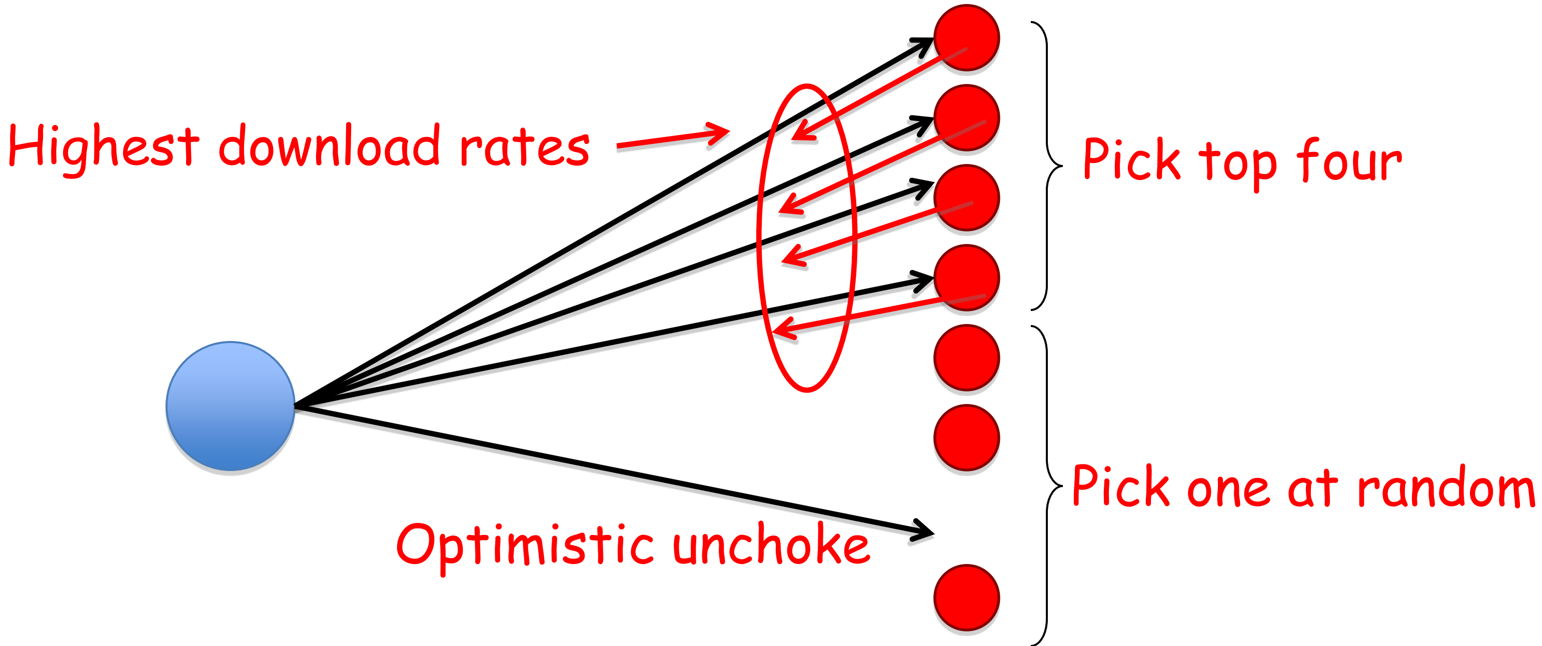
- P2P file distribution



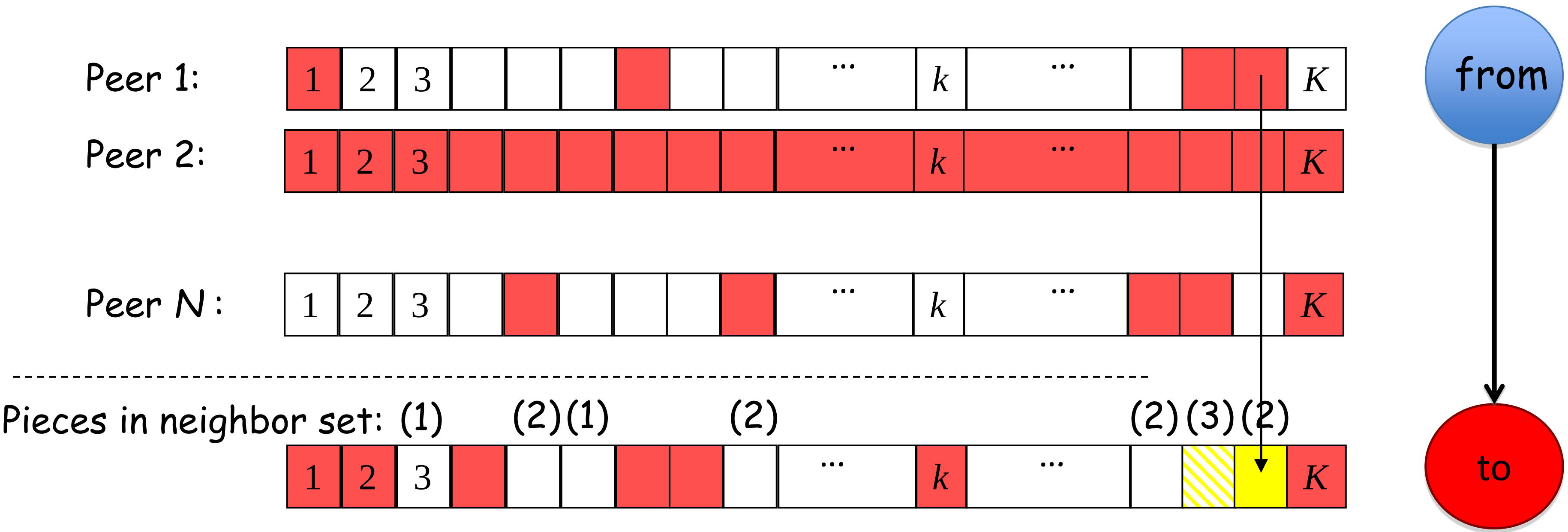
Download using BitTorrent

Background: Incentive mechanism

- Establish connections to large set of peers
  - At each time, only upload to a small (changing) set of peers
- Rate-based tit-for-tat policy
  - Downloaders give upload preference to the downloaders that provide the highest download rates



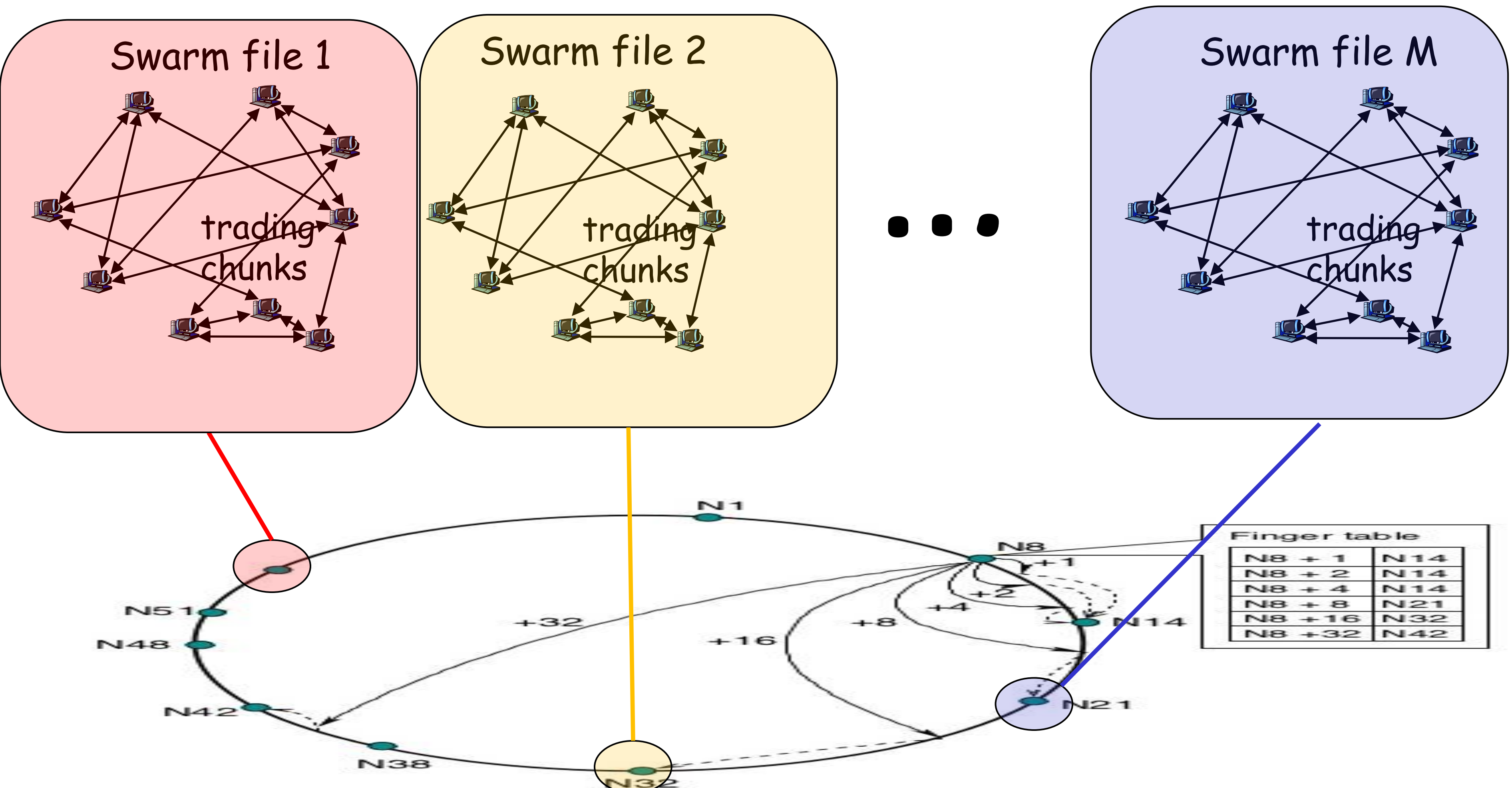
Download using BitTorrent
Background: Piece selection



- Rarest first piece selection policy
  - Achieves high piece diversity
- Request pieces that
  - the uploader has;
  - the downloader is interested (wants); and
  - is the rarest among this set of pieces

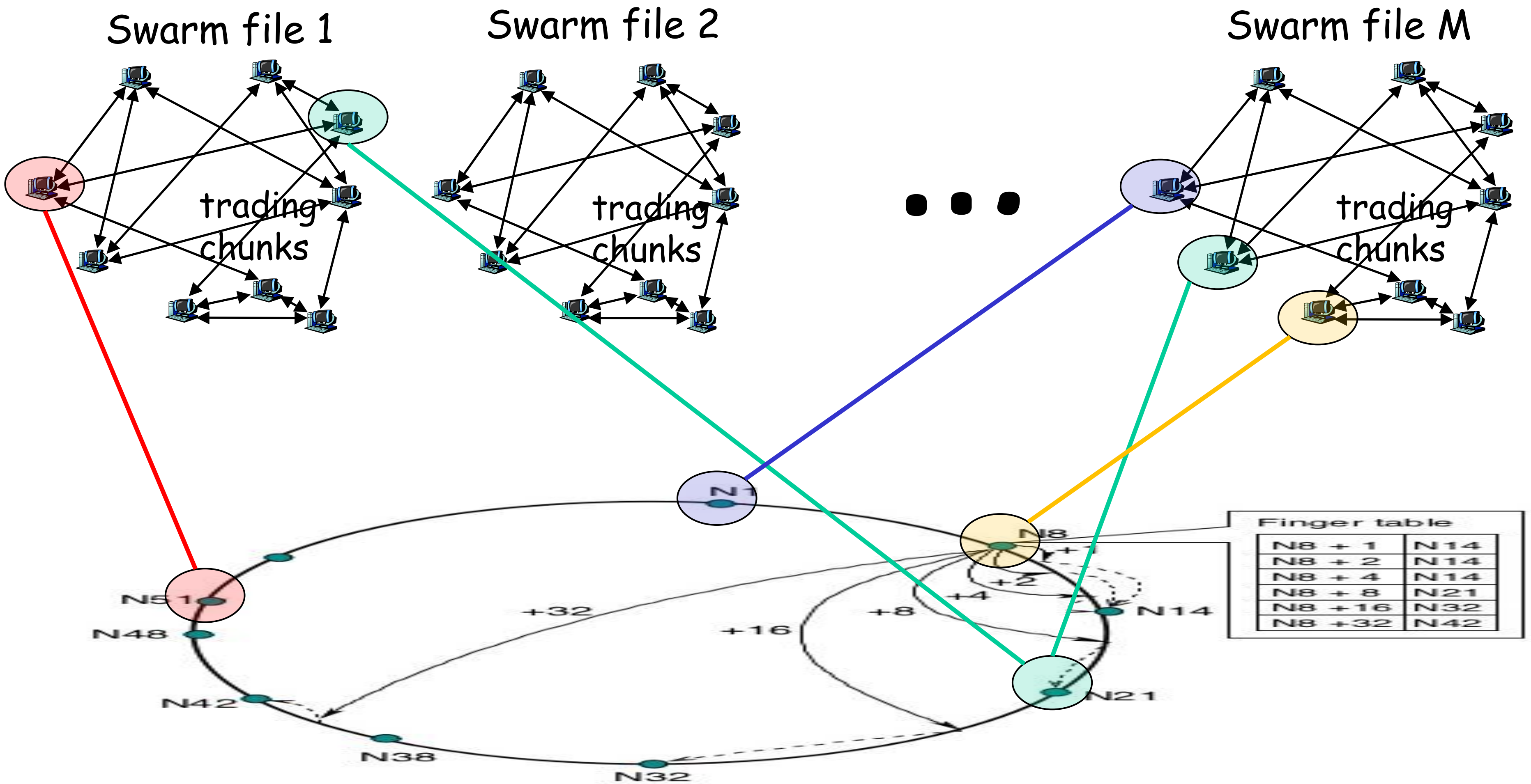
Tracker-less torrents

- Combine DHTs and BT …



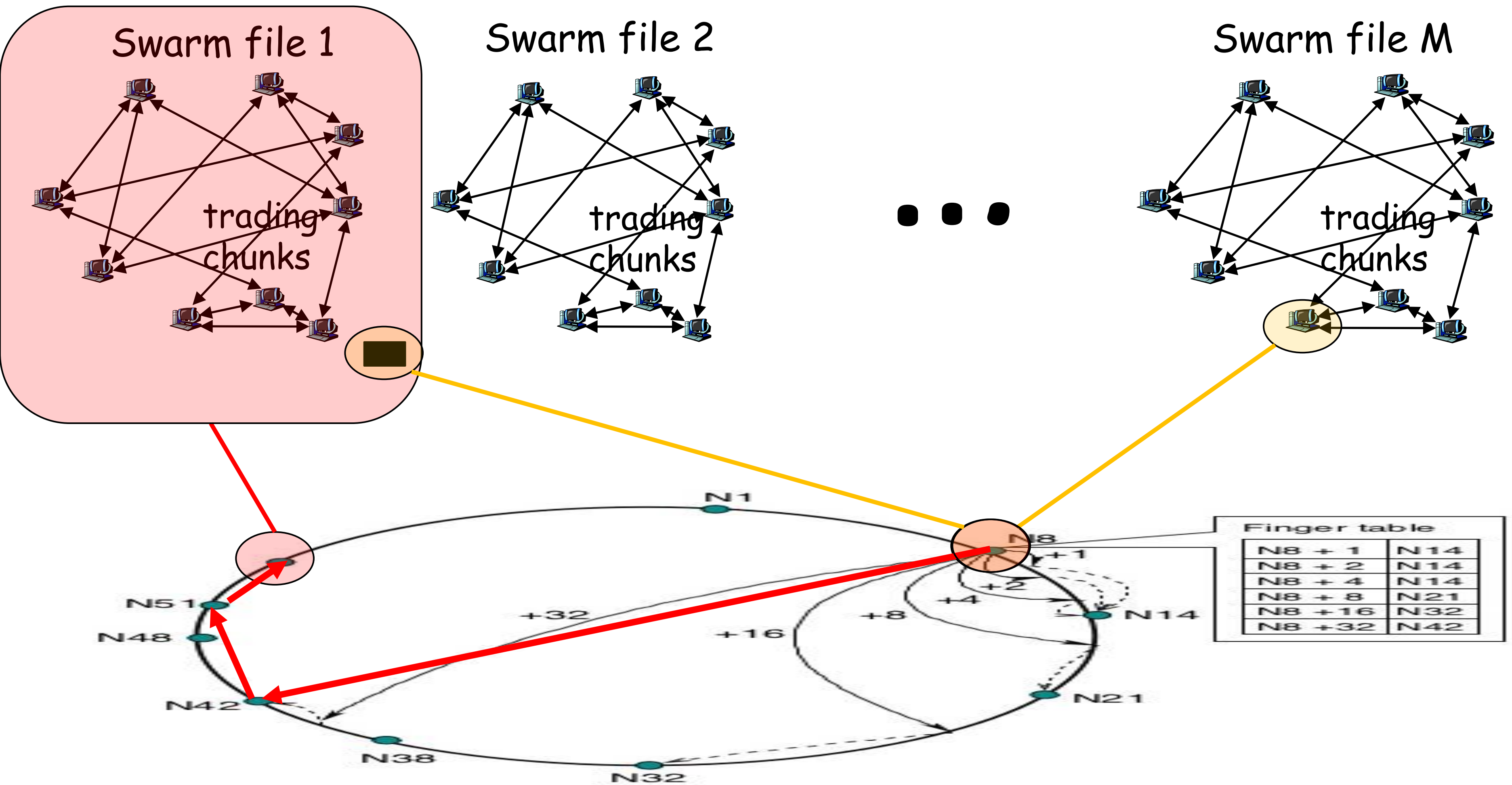
Tracker-less torrents

- Combine DHTs and BT …



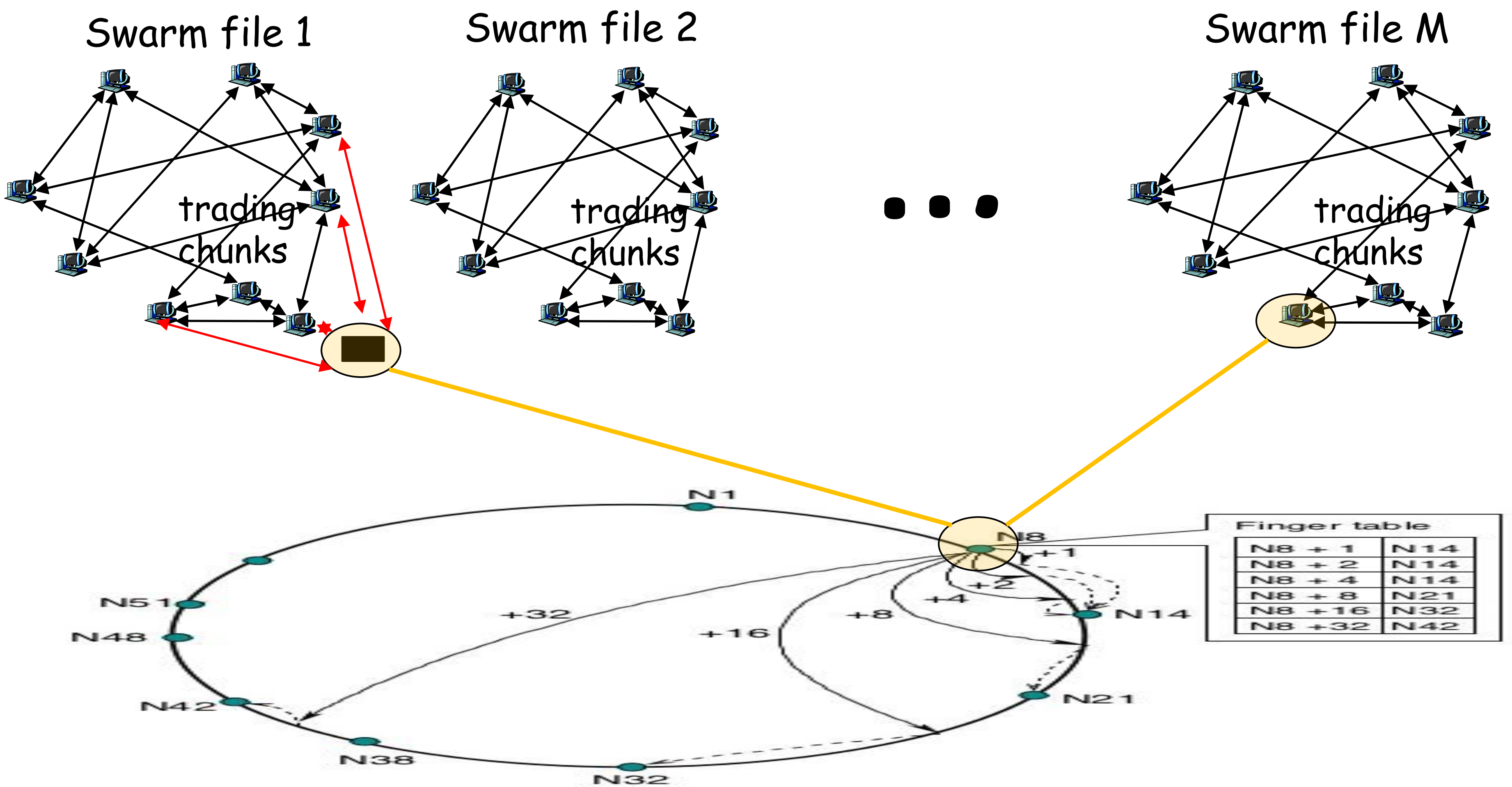
Tracker-less torrents

- Combine DHTs and BT ...



Tracker-less torrents

- Combine DHTs and BT …



Example topic 2

- MapReduce

Motivation

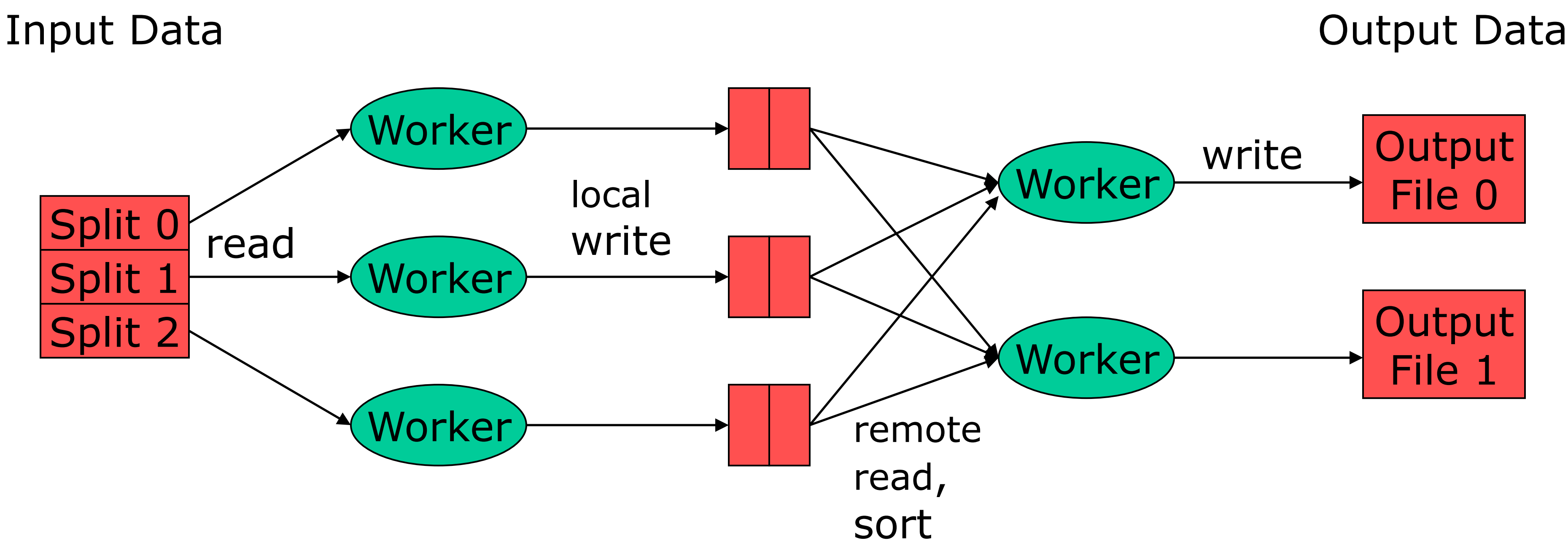
- Process lots of data
  - Google processed about 24 petabytes of data per day in 2009.
- **A single machine** cannot serve all the data
  - You need a distributed system to store and process **in parallel**

MapReduce

- MapReduce [OSDI'04] provides
  - Automatic parallelization, distribution
  - I/O scheduling
    - Load balancing
    - Network and data transfer optimization
  - Fault tolerance
    - Handling of machine failures
- **Need more power: Scale out, not up!**
    - Large number of **commodity servers** as opposed to some high-end specialized servers

Apache Hadoop:
Open source implementation of MapReduce

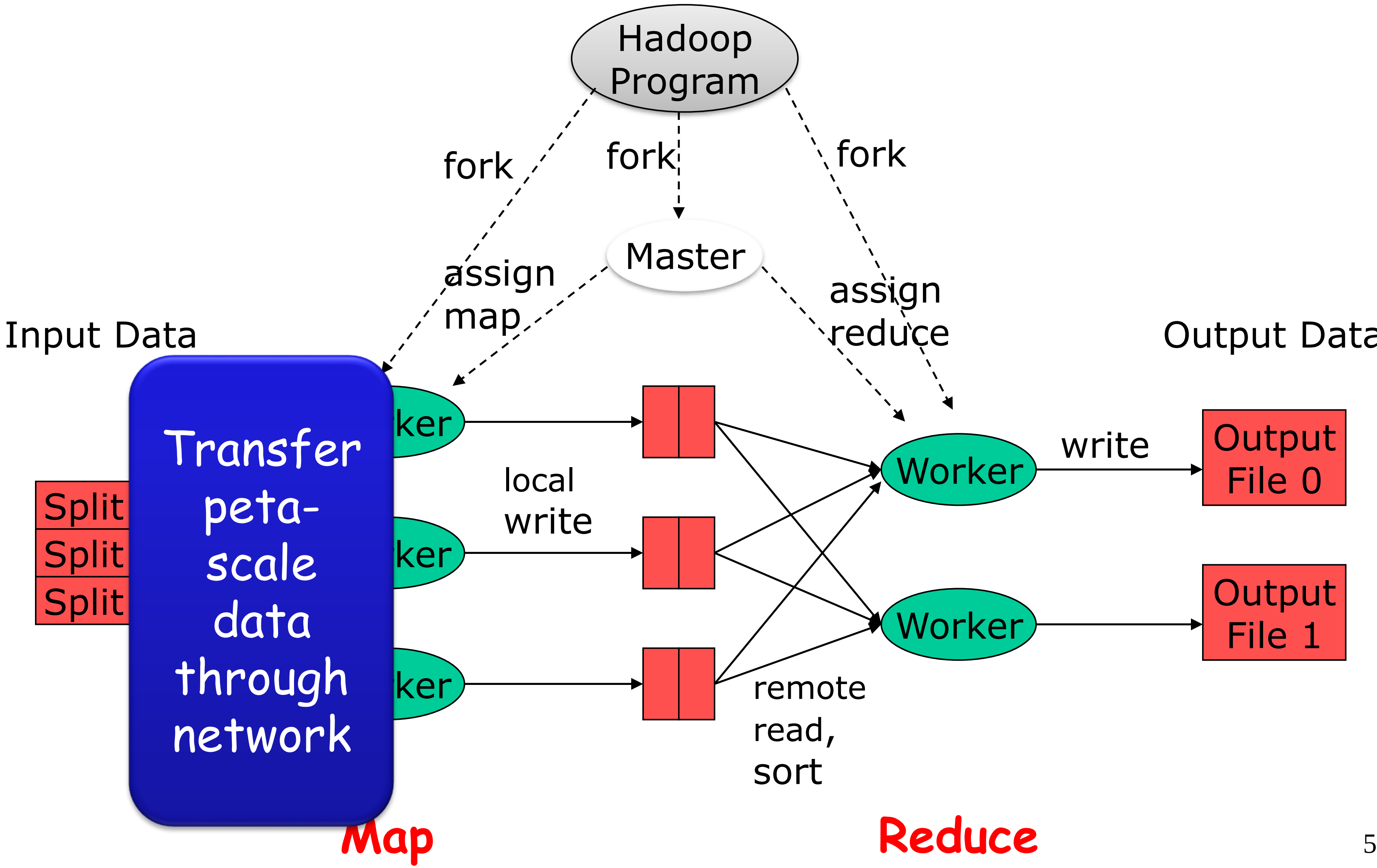
MapReduce workflow



Map
extract something
you care about from
each record

Reduce
aggregate,
summarize,
filter, or
transform

MapReduce



Hadoop Program
fork
fork
fork
Master
assign map
assign reduce
Input Data
Output Data
Split
Split
Split
ker
ker
ker
Transfer peta-scale data through network
local write
Worker
Worker
write
Output File 0
Output File 1
remote read, sort
Map
Reduce

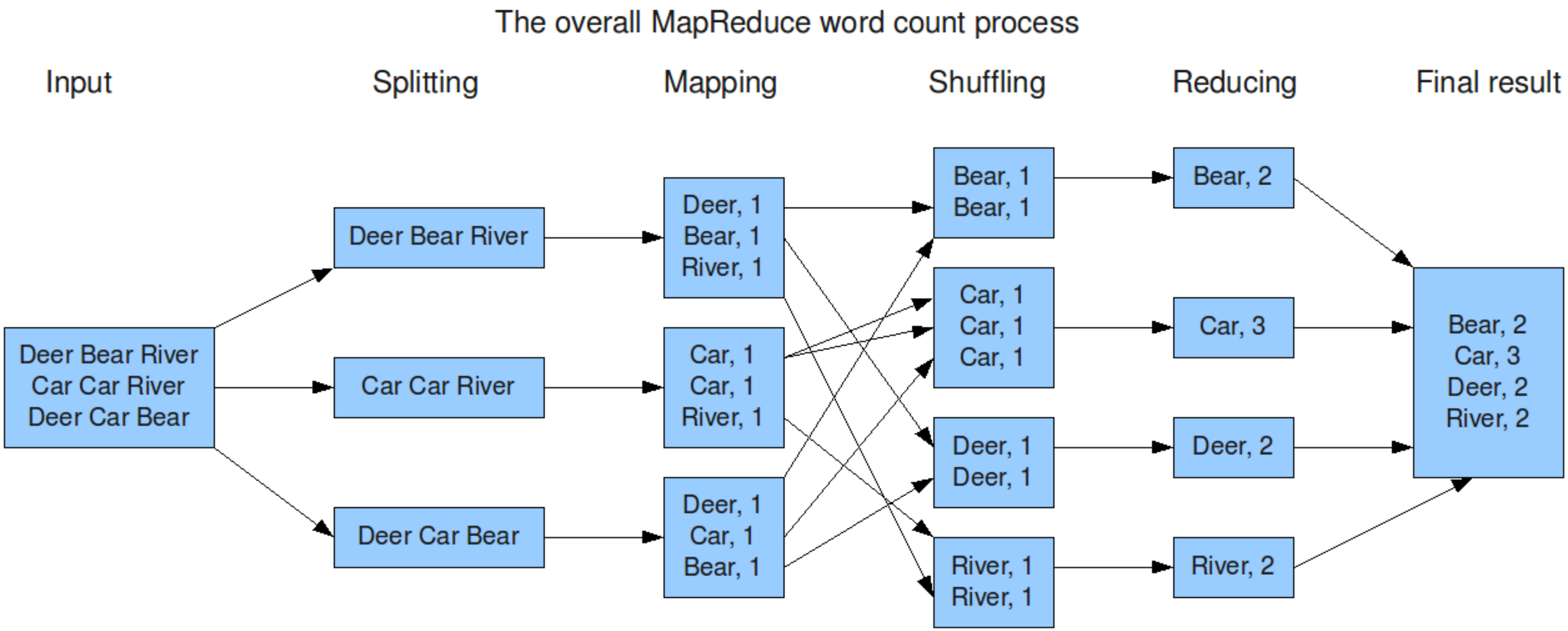
Failure in MapReduce

- Failures are norm in commodity hardware
- Worker failure
  - Detect failure via periodic heartbeats
  - Re-execute in-progress map/reduce tasks
- Master failure
  - Single point of failure; Resume from Execution Log
- Data stored on multiple nodes
- Robust
  - Google's experience: lost 1600 of 1800 machines once!, but finished fine

Example: Word Count

http://kickstarthadoop.blogspot.ca/2011/04/word-count-hadoop-map-reduce-example.html

MapReduce: map, shuffle, reduce

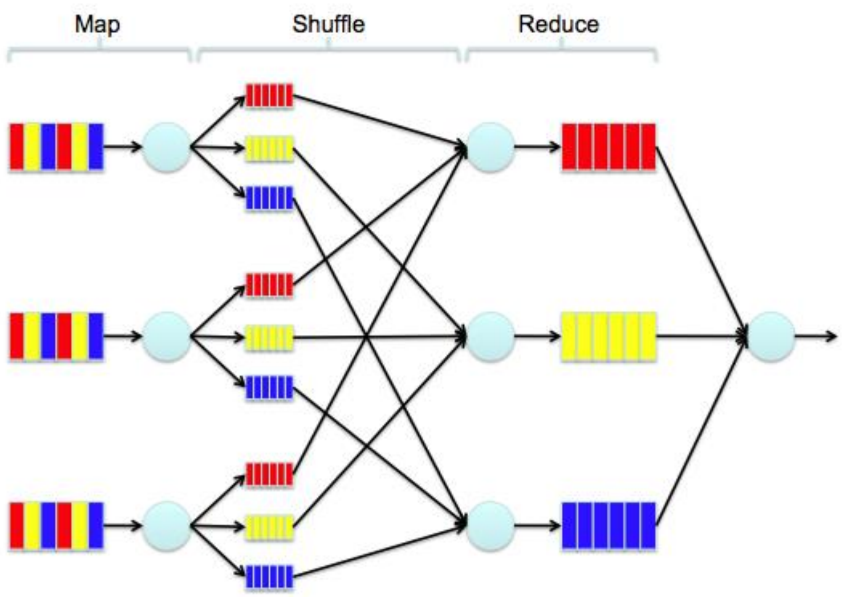


Map: Each worker applies the map function to local data + writes the output to a temporary storage. Master ensures only one copy of the redundant input data is processed.

Shuffle: Workers redistribute data based on the output keys (produced by the map function) such that all data belonging to one key is located on the same worker node

Reduce: Workers process each group of output data, per key, in parallel.

MapReduce: map, shuffle, reduce



Map: Each worker applies the map function to local data + writes the output to a temporary storage. Master ensures only one copy of the redundant input data is processed.

Shuffle: Workers redistribute data based on the output keys (produced by the map function) such that all data belonging to one key is located on the same worker node

Reduce: Workers process each group of output data, per key, in parallel.

Summary

- MapReduce
  - Programming paradigm for data-intensive computing
  - Distributed & parallel execution model
  - Simple to program