Peer-to-Peer Networks

Jeff Pang

Intro

- Quickly grown in popularity
  - Dozens or hundreds of file sharing applications
  - 35 million American adults use P2P networks -- 29% of all Internet users in US!
  - Audio/Video transfer now dominates traffic on the Internet
- But what is P2P?
  - Computers “Peering”? -- Server Clusters, IRC Networks, Internet Routing!
  - Clients with no servers? -- Doom, Quake!

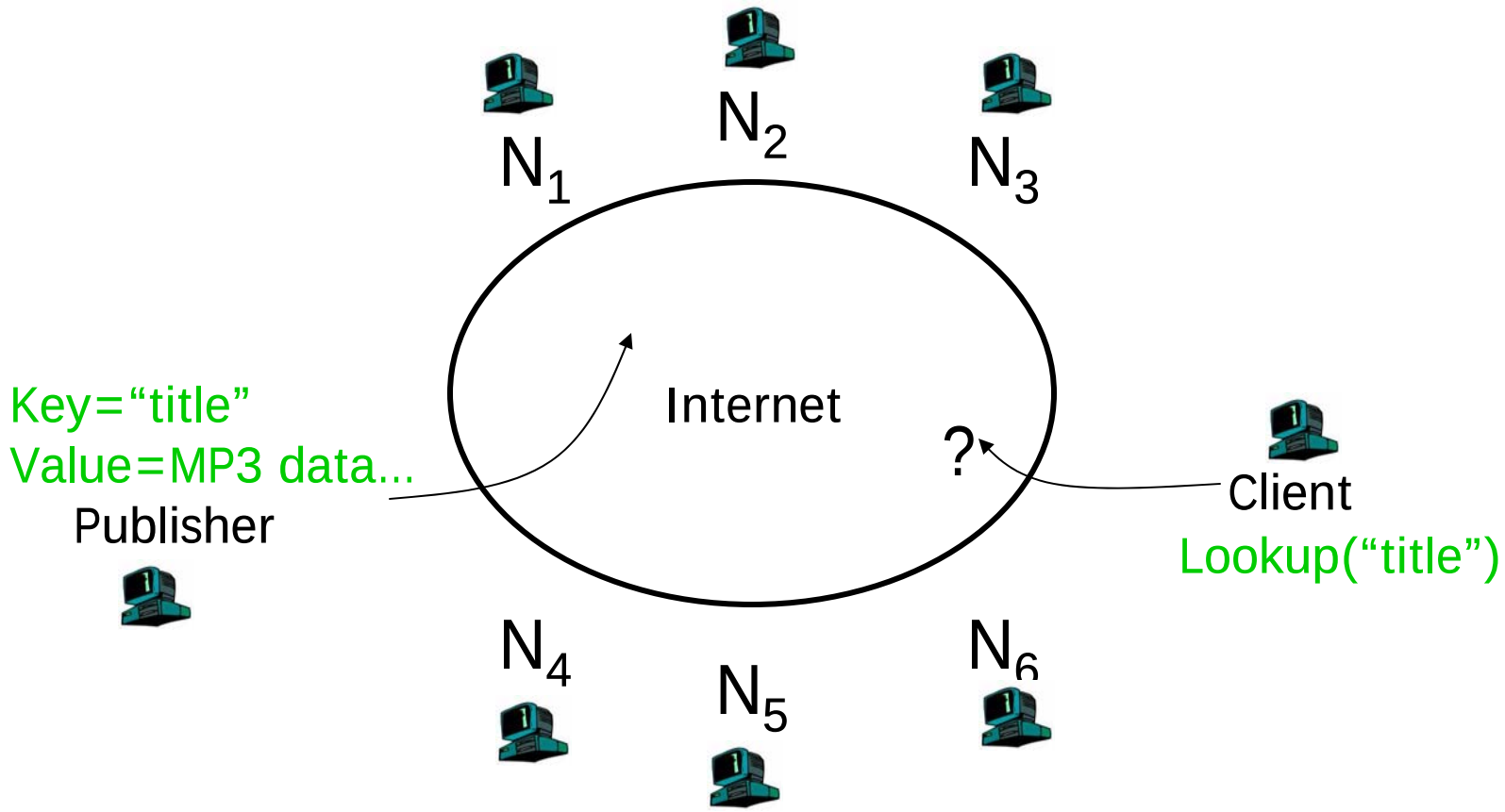
Intro (2)

- Fundamental difference: Take advantage of resources at the *edges* of the network
- What's changed:
  - End-host resources have increased dramatically
  - Broadband connectivity now common
- What hasn't:
  - Deploying infrastructure still expensive

Overview

- **Centralized Database**
  - Napster
- **Query Flooding**
  - Gnutella
- **Intelligent Query Flooding**
  - KaZaA
- **Swarming**
  - BitTorrent
- **Unstructured Overlay Routing**
  - Freenet
- **Structured Overlay Routing**
  - Distributed Hash Tables (Chord)

The Lookup Problem



The Lookup Problem (2)

- Common Primitives:
  - **Join**: how to I begin participating?
  - **Publish**: how do I advertise my file?
  - **Search**: how to I find a file?
  - **Fetch**: how to I retrieve a file?

Next Topic...

- **Centralized Database**
  - Napster
- **Query Flooding**
  - Gnutella
- **Intelligent Query Flooding**
  - KaZaA
- **Swarming**
  - BitTorrent
- **Unstructured Overlay Routing**
  - Freenet
- **Structured Overlay Routing**
  - Distributed Hash Tables

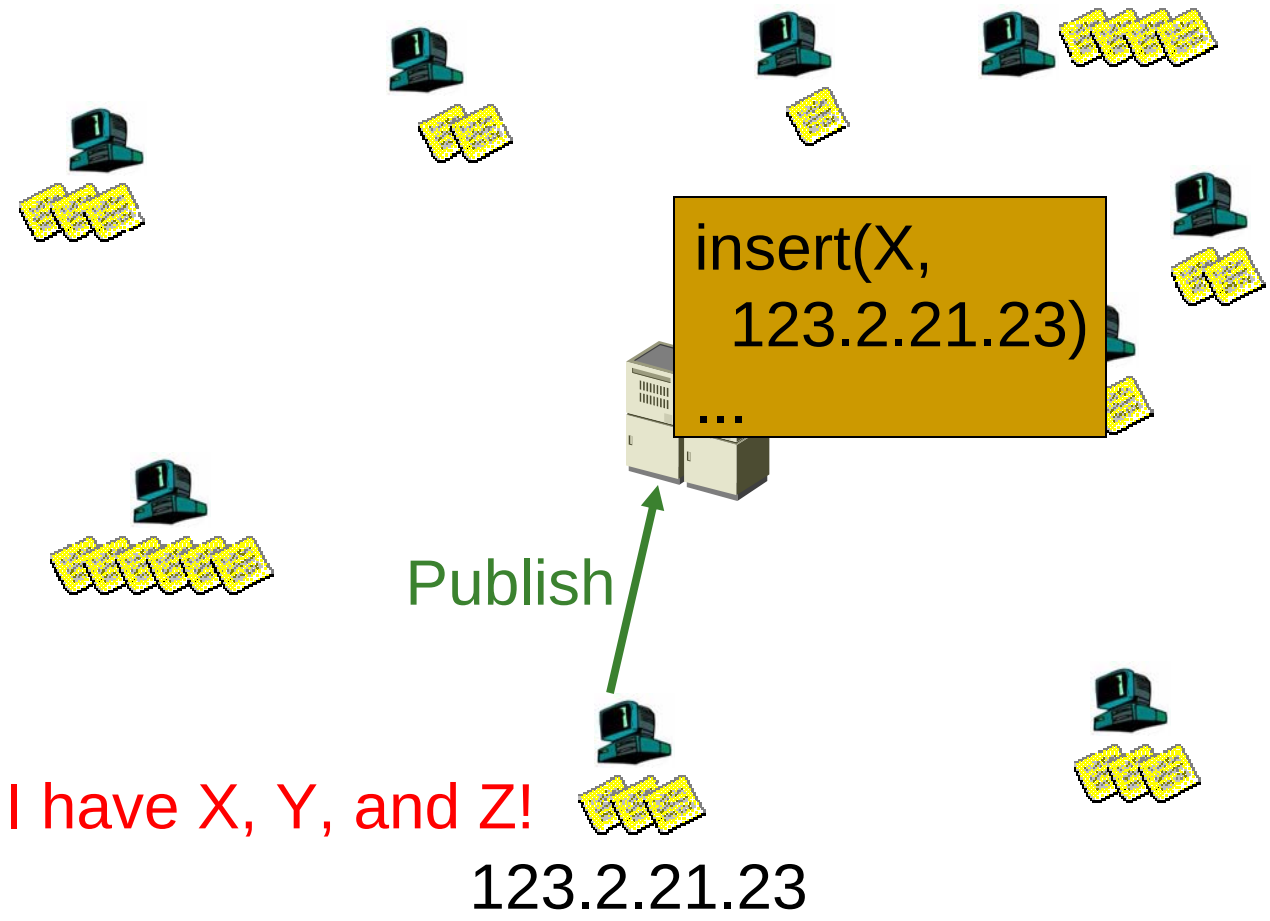
Napster: History

- In 1999, S. Fanning launches Napster
- Peaked at 1.5 million simultaneous users
- Jul 2001, Napster shuts down

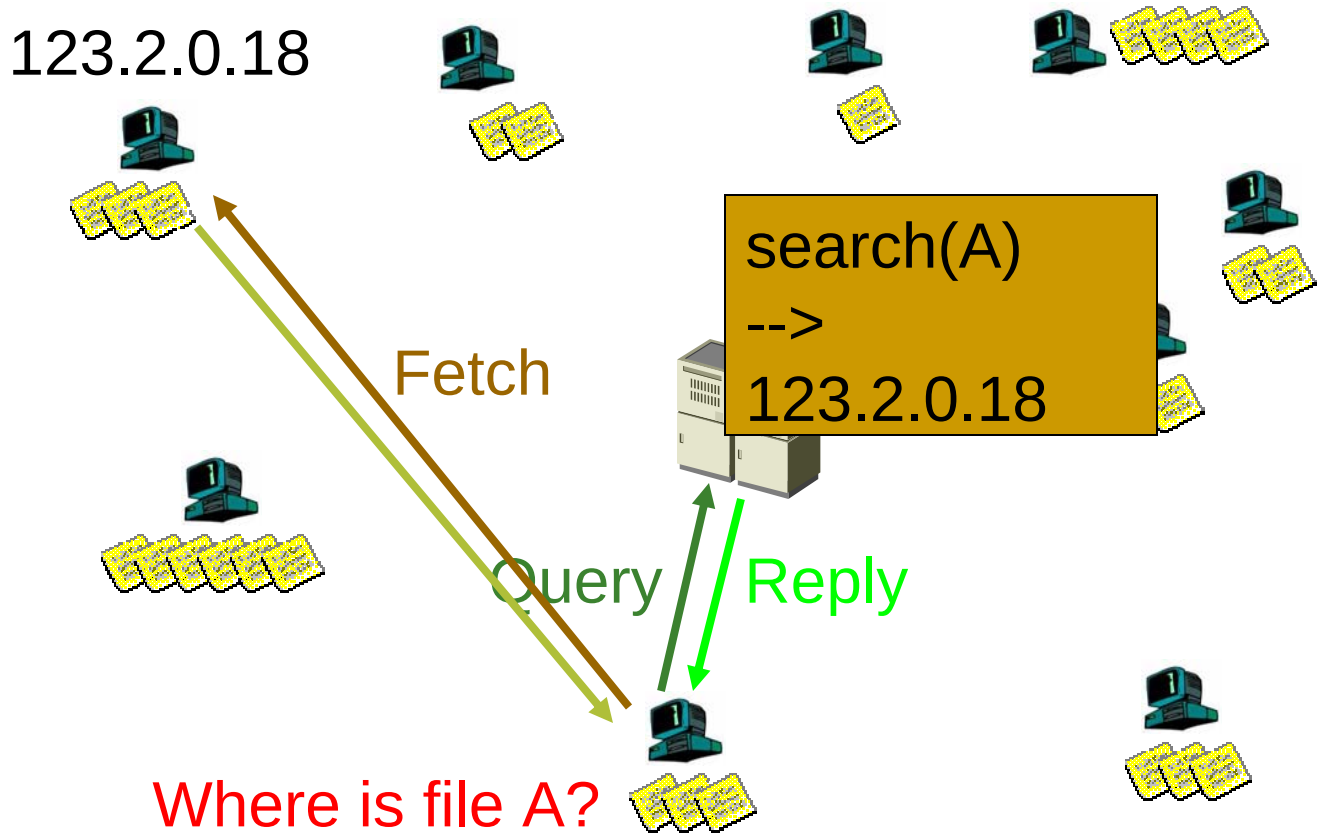
Napster: Overiew

- Centralized Database:
  - **Join**: on startup, client contacts central server
  - **Publish**: reports list of files to central server
  - **Search**: query the server => return someone that stores the requested file
  - **Fetch**: get the file directly from peer

Napster: Publish



Napster: Search



Napster: Discussion

- Pros:
  - Simple
  - Search scope is O(1)
  - Controllable (pro or con?)
- Cons:
  - Server maintains O(N) State
  - Server does all processing
  - Single point of failure

Next Topic...

- **Centralized Database**
  - Napster
- **Query Flooding**
  - Gnutella
- **Intelligent Query Flooding**
  - KaZaA
- **Swarming**
  - BitTorrent
- **Unstructured Overlay Routing**
  - Freenet
- **Structured Overlay Routing**
  - Distributed Hash Tables

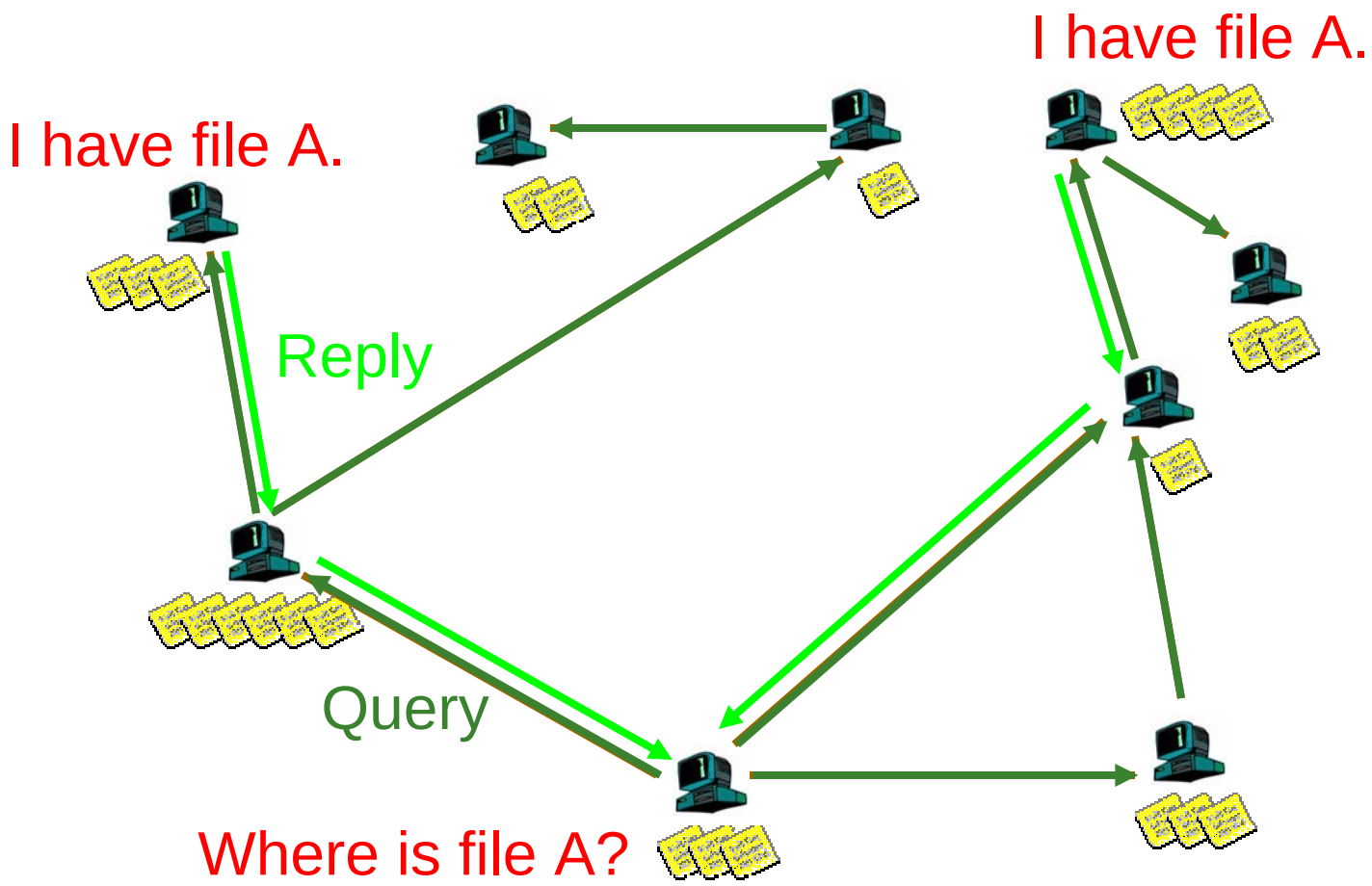
Gnutella: History

- In 2000, Justin. Frankel and Tom. Pepper from Nullsoft released Gnutella
- Soon many other clients: Bearshare, Morpheus, LimeWire, etc.
- In 2001, many protocol enhancements including “ultrapeers”

Gnutella: Overview

- Query Flooding:
  - **Join**: on startup, client contacts a few other nodes; these become its “neighbors”
  - **Publish**: no need
  - **Search**: ask neighbors, who as their neighbors, and so on... when/if found, reply to sender.
  - **Fetch**: get the file directly from peer

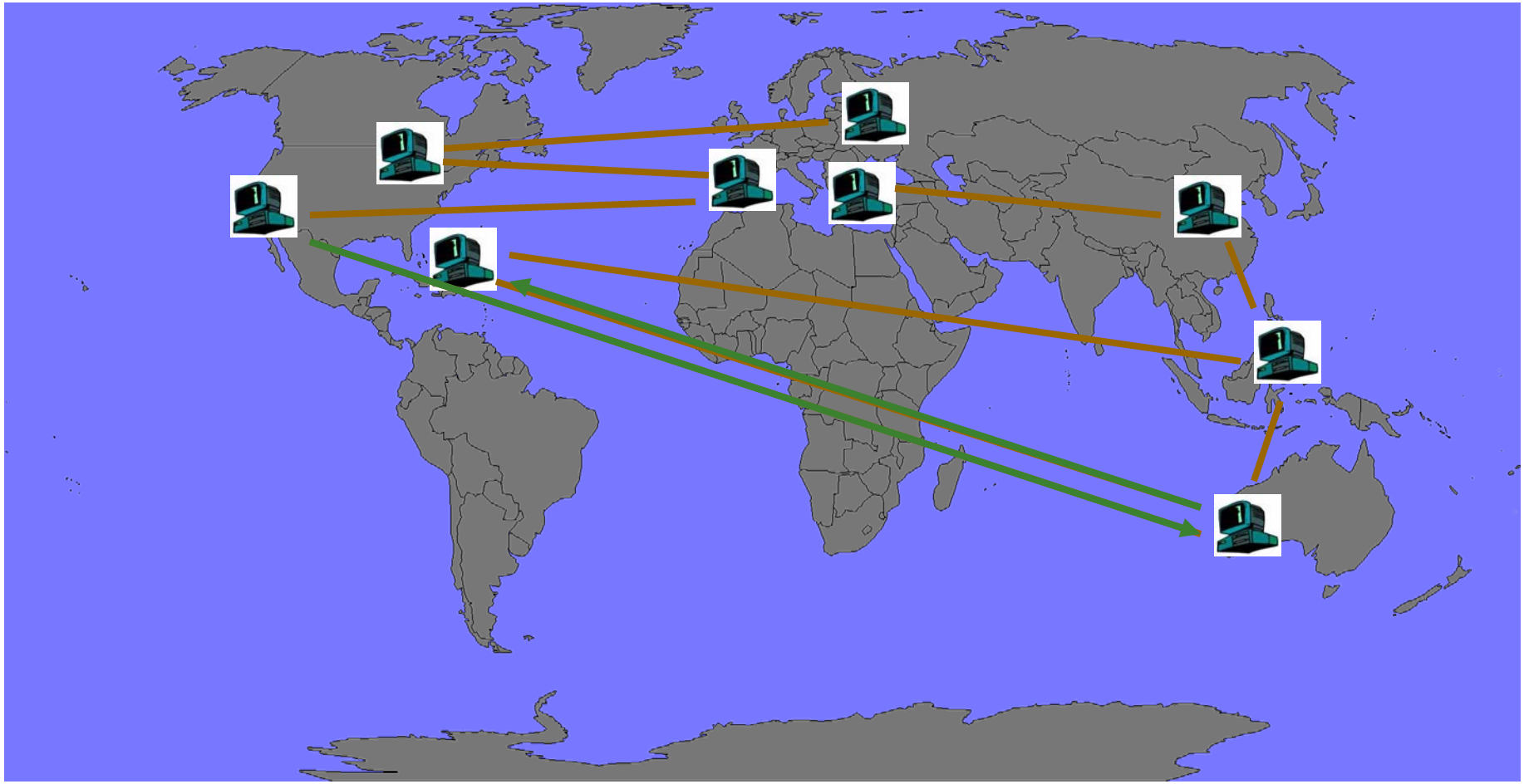
Gnutella: Search



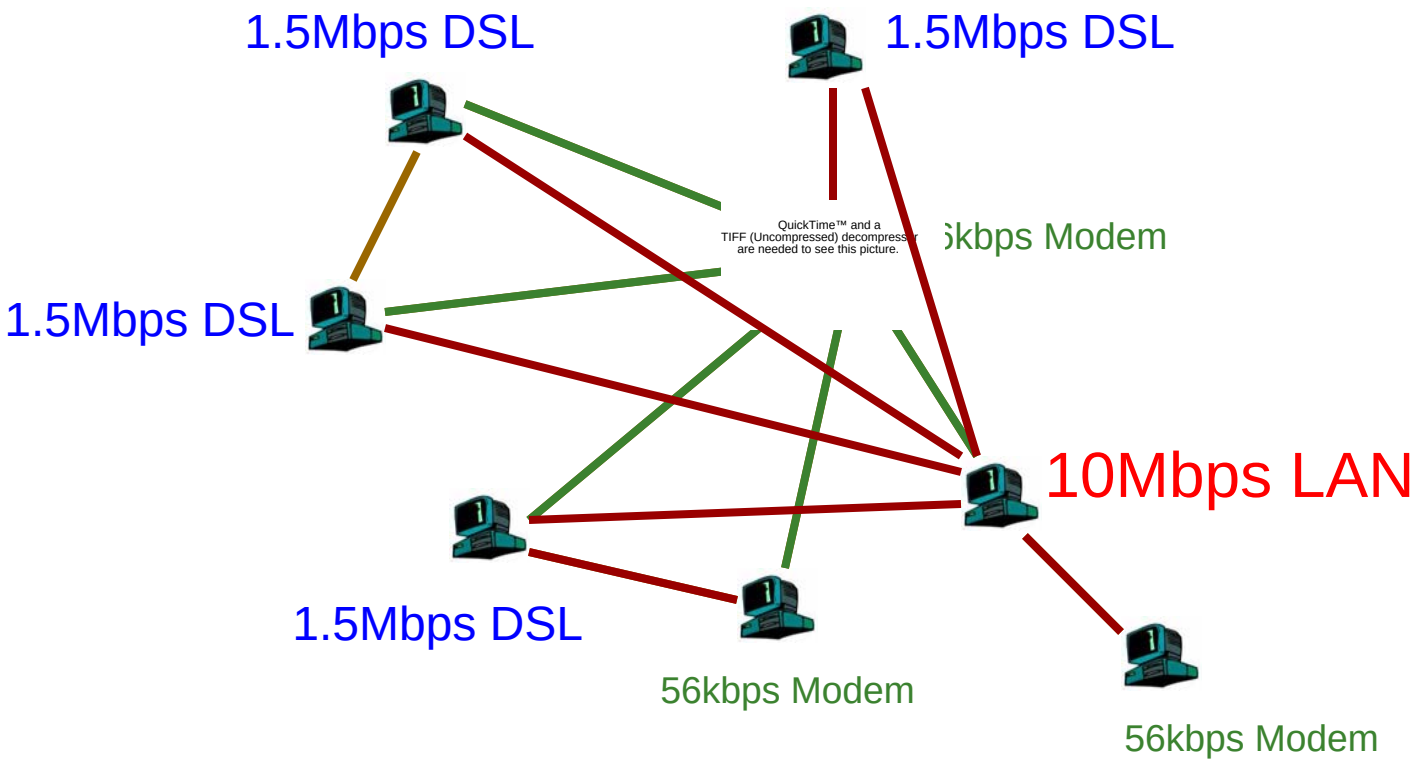
Gnutella: Discussion

- Pros:
  - Fully de-centralized
  - Search cost distributed
- Cons:
  - Search scope is O(*N*)
  - Search time is $O(Log_b(N))$
    - b: the average outdegree
  - Nodes leave often, network unstable

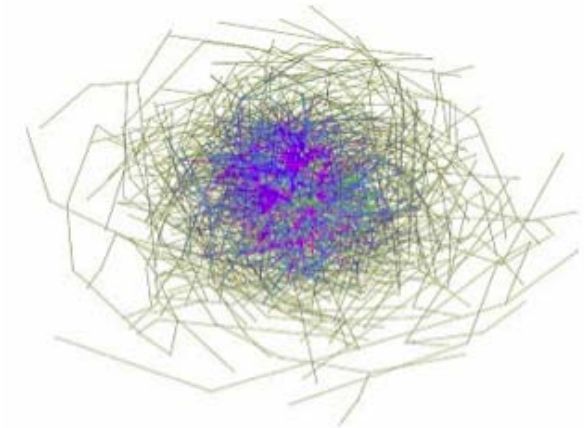
Aside: Search Time?



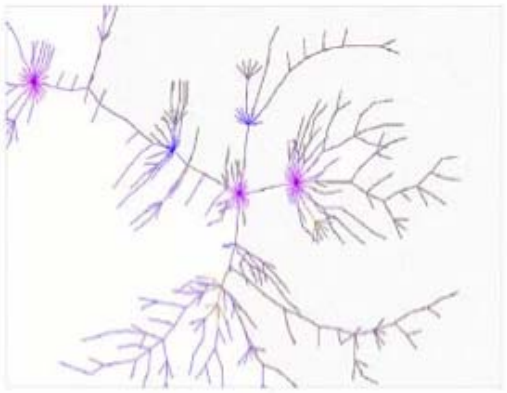
Aside: All Peers Equal?



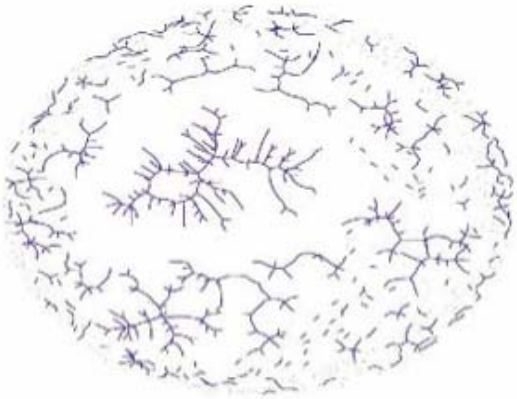
Aside: Network Resilience



Partial Topology



Random 30% die



Targeted 4% die

from Saroiu *et al*., *MMCN* 2002

Next Topic...

- **Centralized Database**
  - Napster
- **Query Flooding**
  - Gnutella
- **Intelligent Query Flooding**
  - KaZaA
- **Swarming**
  - BitTorrent
- **Unstructured Overlay Routing**
  - Freenet
- **Structured Overlay Routing**
  - Distributed Hash Tables

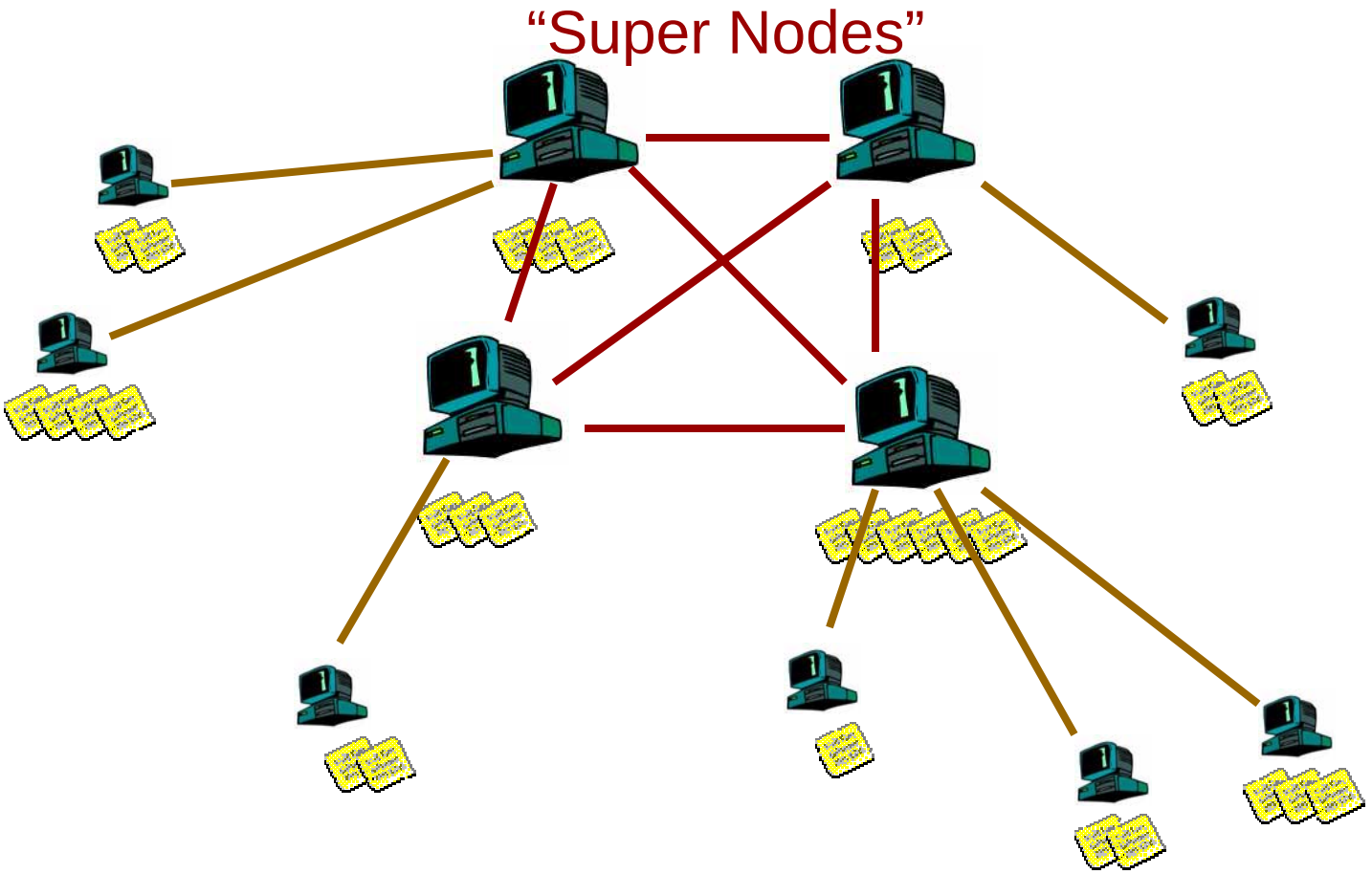
KaZaA: History

- In 2001, KaZaA created by Dutch company Kazaa BV.
- Founders: Niklas Zennstrom and Janus Friis. (heard of skype? They wrote it also.)
- Single network called FastTrack used by other clients as well: Morpheus, giFT, etc.
- Eventually protocol changed so other clients could no longer talk to it.
- Most popular file sharing network today with >10 million users (number varies)

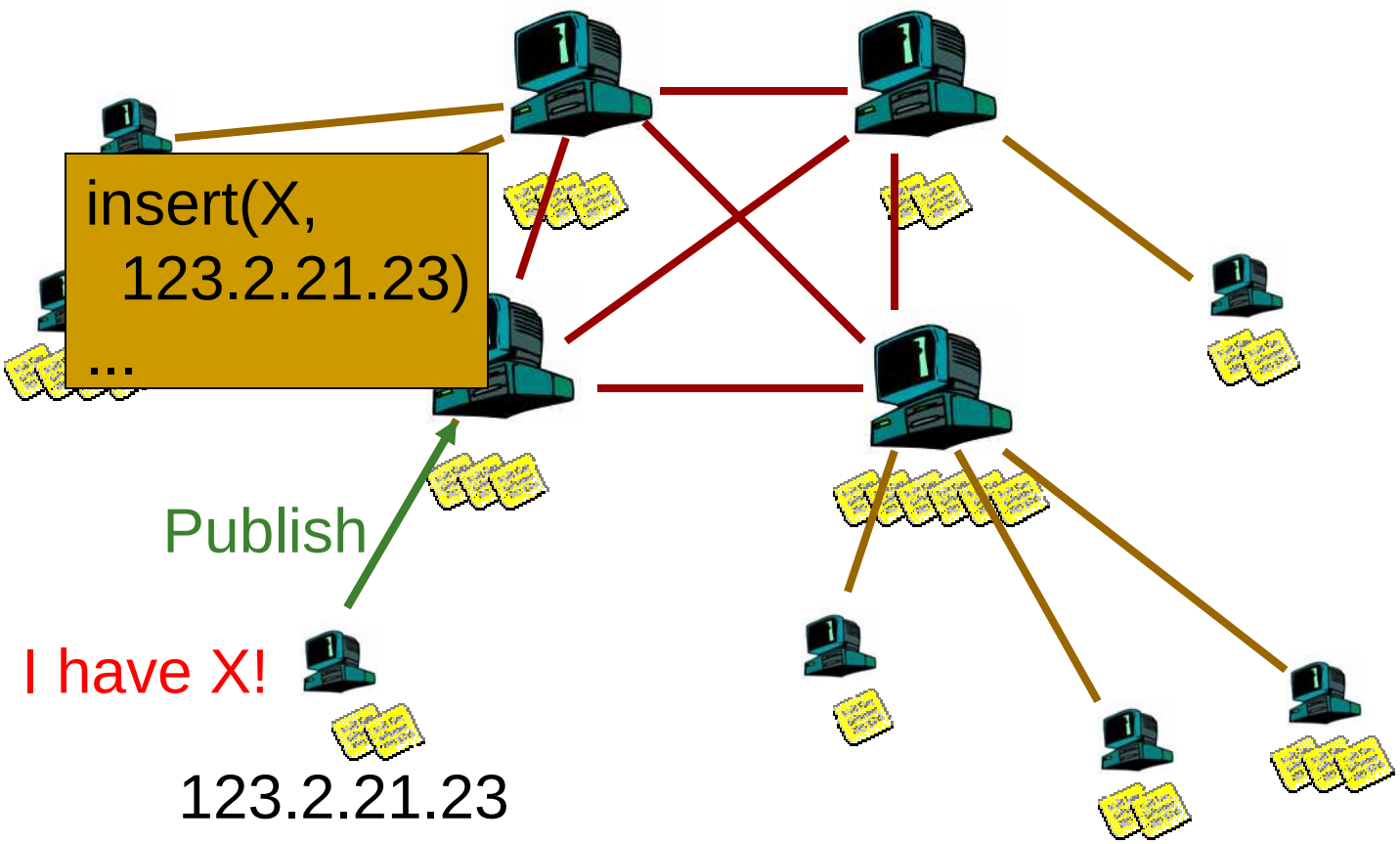
KaZaA: Overview

- "Smart" Query Flooding:
  - **Join**: on startup, client contacts a "supernode" ... may at some point become one itself
  - **Publish**: send list of files to supernode
  - **Search**: send query to supernode, supernodes flood query amongst themselves.
  - **Fetch**: get the file directly from peer(s); can fetch simultaneously from multiple peers

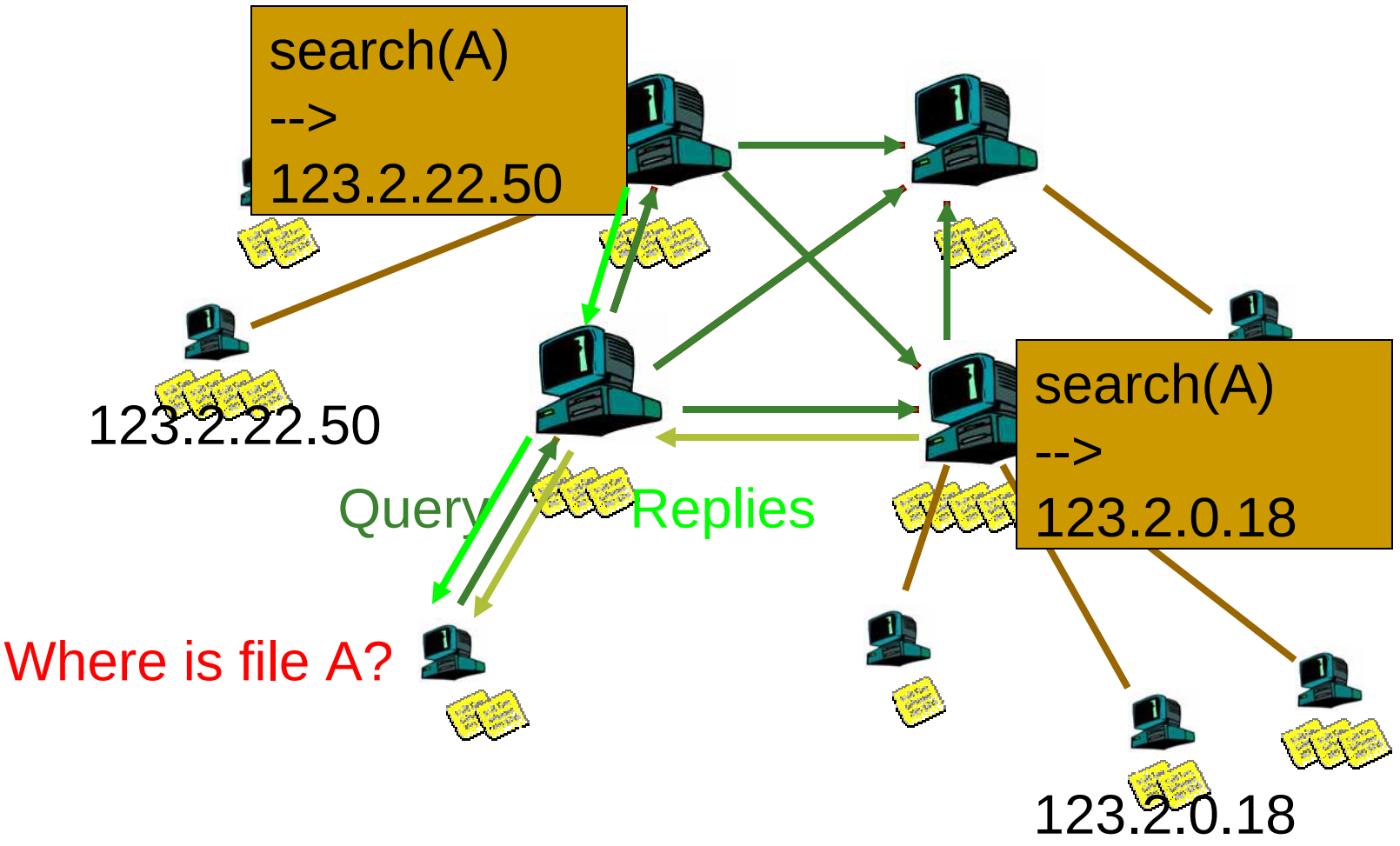
KaZaA: Network Design



KaZaA: File Insert



KaZaA: File Search



KaZaA: Fetching

- More than one node may have requested file...
- How to tell?
  - Must be able to distinguish identical files
  - Not necessarily same filename
  - Same filename not necessarily same file...
- Use Hash of file
  - KaZaA uses UUHash: fast, but not secure
  - Alternatives: MD5, SHA-1
- How to fetch?
  - Get bytes [0..1000] from A, [1001...2000] from B
  - Alternative: Erasure Codes

KaZaA: Discussion

- Pros:
  - Tries to take into account node heterogeneity:
    - Bandwidth
    - Host Computational Resources
    - Host Availability (?)
  - Rumored to take into account network locality
- Cons:
  - Mechanisms easy to circumvent
  - Still no real guarantees on search scope or search time

Next Topic...

- **Centralized Database**
  - Napster
- **Query Flooding**
  - Gnutella
- **Intelligent Query Flooding**
  - KaZaA
- **Swarming**
  - BitTorrent
- **Unstructured Overlay Routing**
  - Freenet
- **Structured Overlay Routing**
  - Distributed Hash Tables

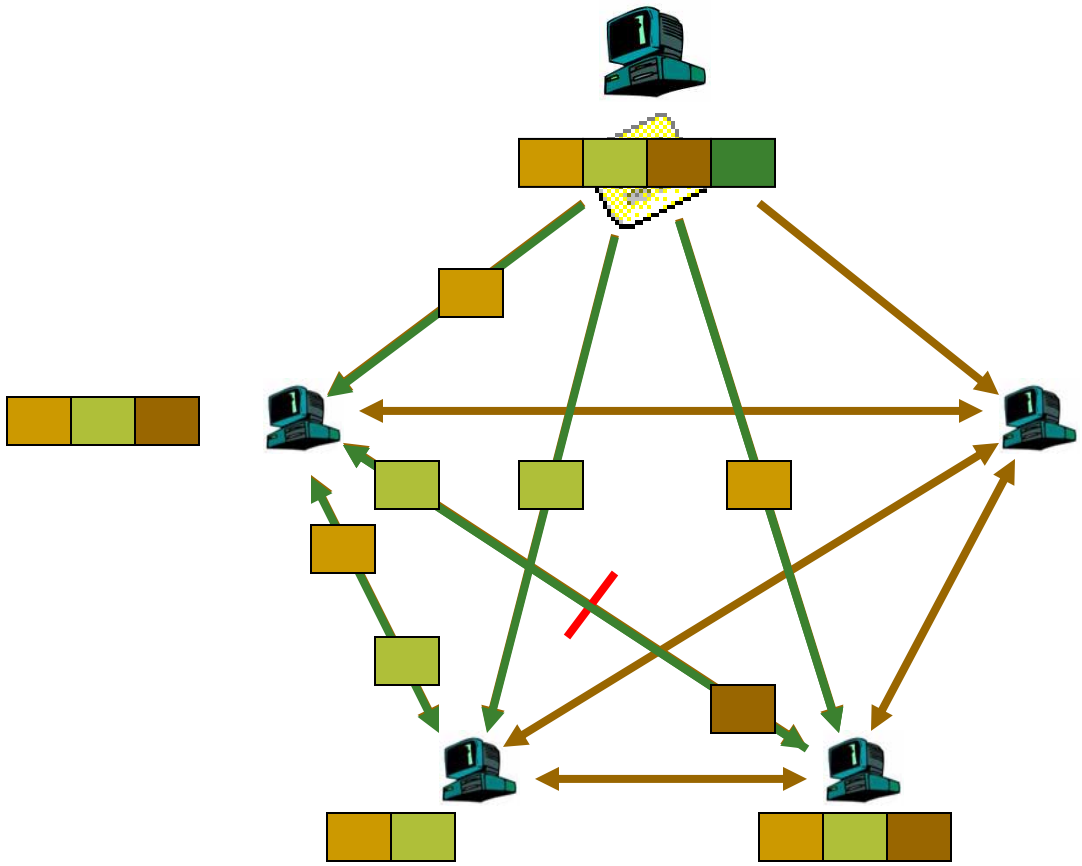
BitTorrent: History

- In 2002, B. Cohen debuted BitTorrent
- Key Motivation:
  - Popularity exhibits temporal locality (Flash Crowds)
  - E.g., Slashdot effect, CNN on 9/11, new movie/game release
- Focused on Efficient *Fetching*, not *Searching*:
  - Distribute the *same* file to all peers
  - Single publisher, multiple downloaders
- Has some “real” publishers:
  - Blizzard Entertainment using it to distribute the beta of their new game

BitTorrent: Overview

- Swarming:
  - **Join**: contact centralized “tracker” server, get a list of peers.
  - **Publish**: Run a tracker server.
  - **Search**: Out-of-band. E.g., use Google to find a tracker for the file you want.
  - **Fetch**: Download chunks of the file from your peers. Upload chunks you have to them.

BitTorrent: Fetch



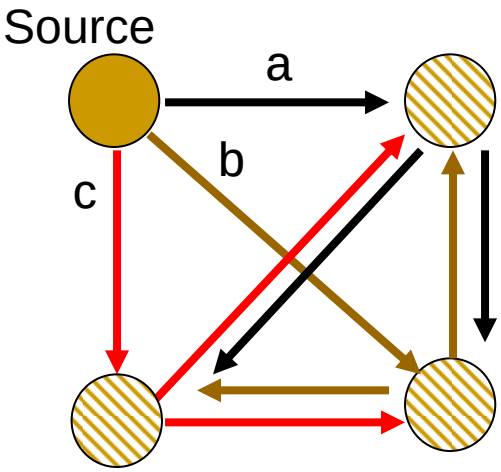
BitTorrent: Sharing Strategy

- Employ "Tit-for-tat" sharing strategy
  - "I'll share with you if you share with me"
  - Be optimistic: occasionally let freeloaders download
    - Otherwise no one would ever start!
    - Also allows you to discover better peers to download from when they reciprocate
- Approximates Pareto Efficiency
  - Game Theory: The optimal strategy is the strategy such that "no change can make anyone better off without making others worse off"

BitTorrent: Summary

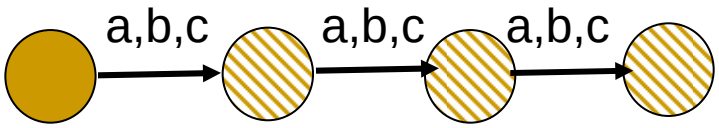
- Pros:
  - Works reasonably well in practice
  - Gives peers incentive to share resources; avoids freeloaders
- Cons:
  - Pareto Efficiency relative weak condition

Aside: Static BitTorrent-Like Networks

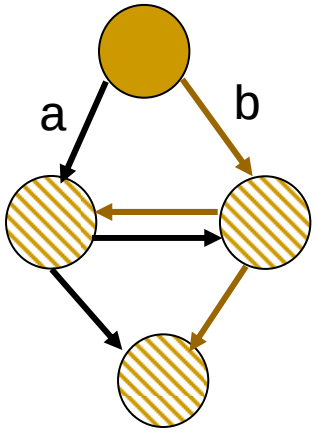


Scenario 1: All nodes with same capacities.

Convince yourself that partioning packets equally among nodes as above is optimal, i.e. minimize the time to download the file for all the nodes.

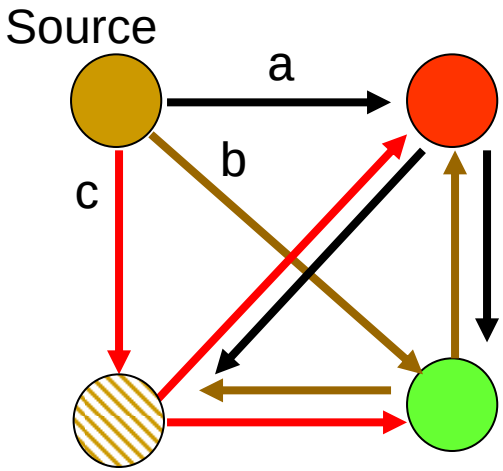


Is this scheme optimal?



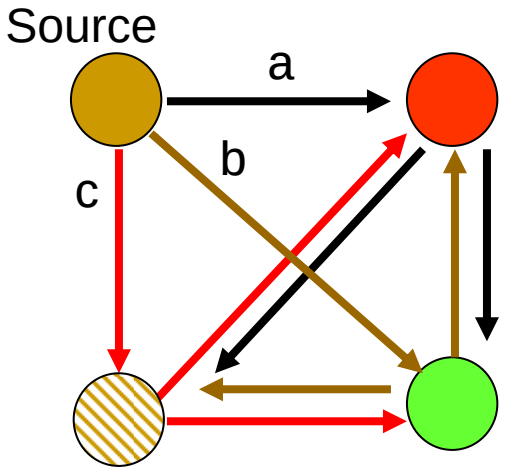
Is this scheme optimal?

Aside: Static BitTorrent-Like Networks



Nodes have different capacities.

Optimal Solution?



Nodes joins at different times!

Optimal Solution?

Default final project: Devise a scheme to approximate the optimal solution for a BitTorrent like network when node joins at differrent time!

Next Topic...

- **Centralized Database**
  - Napster
- **Query Flooding**
  - Gnutella
- **Intelligent Query Flooding**
  - KaZaA
- **Swarming**
  - BitTorrent
- **Unstructured Overlay Routing**
  - Freenet
- **Structured Overlay Routing**
  - Distributed Hash Tables

Freenet: History

- In 1999, I. Clarke started the Freenet project
- Basic Idea:
  - Employ Internet-like routing on the overlay network to publish and locate files
- Addition goals:
  - Provide anonymity and security
  - Make censorship difficult

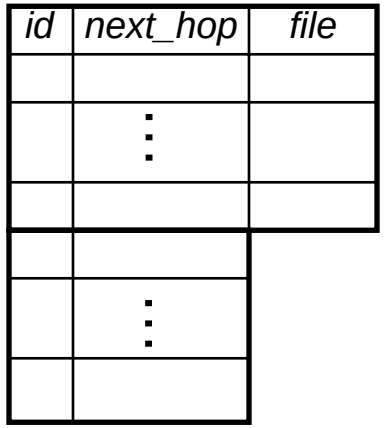
Freenet: Overview
http://freenetproject.org/

- Routed Queries:
  - **Join**: on startup, client contacts a few other nodes it knows about; gets a unique *node id*
  - **Publish**: route file contents toward the *file id*. File is stored at node with *id* closest to *file id*
  - **Search**: route query for *file id* toward the closest *node id*
  - **Fetch**: when query reaches a node containing *file id*, it returns the file to the sender

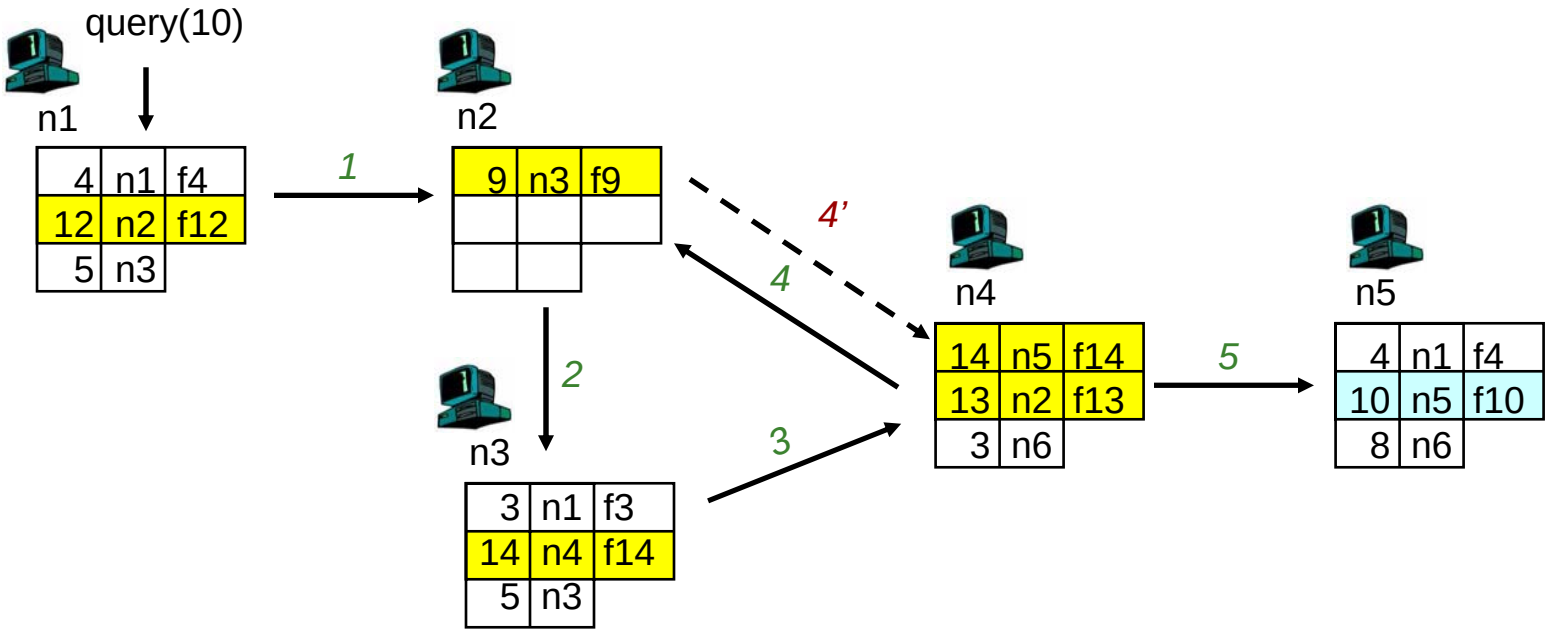
Freenet: Routing Tables

- ***id*** – file identifier (e.g., hash of file)
- ***next_hop*** – another node that stores the file id
- ***file*** – file identified by *id* being stored on the local node

- Forwarding of query for file *id*
  - If file *id* stored locally, then stop
    - Forward data back to upstream requestor
  - If not, search for the "closest" *id* in the table, and forward the message to the corresponding *next_hop*
  - If data is not found, failure is reported back
    - Requestor then tries next closest match in routing table



Freenet: Routing



Freenet: Routing Properties

- "Close" file ids tend to be stored on the same node
  - Why? Publications of similar file ids route toward the same place
- Network tend to be a "small world"
  - Small number of nodes have large number of neighbors (i.e., ~ "six-degrees of separation")
- Consequence:
  - Most queries only traverse a small number of hops to find the file

Freenet: Anonymity & Security

- Anonymity
  - Randomly modify source of packet as it traverses the network
  - Can use “mix-nets” or onion-routing
- Security & Censorship resistance
  - No constraints on how to choose *ids* for files => easy to have to files collide, creating “denial of service” (censorship)
  - Solution: have a *id* type that requires a private key signature that is verified when updating the file

Freenet: Discussion

- Pros:
  - Intelligent routing makes queries relatively short
  - Search scope small (only nodes along search path involved); no flooding
  - Anonymity properties may give you “plausible deniability”
- Cons:
  - Still no provable guarantees!
  - Anonymity features make it hard to measure, debug

Next Topic...

- **Centralized Database**
  - Napster
- **Query Flooding**
  - Gnutella
- **Intelligent Query Flooding**
  - KaZaA
- **Swarming**
  - BitTorrent
- **Unstructured Overlay Routing**
  - Freenet
- **Structured Overlay Routing**
  - Distributed Hash Tables (DHT)

DHT: History

- In 2000-2001, academic researchers said “we want to play too!”
- Motivation:
  - Frustrated by popularity of all these “half-baked” P2P apps :)
  - We can do better! (so we said)
  - Guaranteed lookup success for files in system
  - Provable bounds on search time
  - Provable scalability to millions of node
- Hot Topic in networking ever since

DHT: Overview

- **Abstraction**: a distributed “hash-table” (DHT) data structure:
  - put(id, item);
  - item = get(id);
- **Implementation**: nodes in system form a distributed data structure
  - Can be Ring, Tree, Hypercube, Skip List, Butterfly Network, ...

DHT: Overview (2)

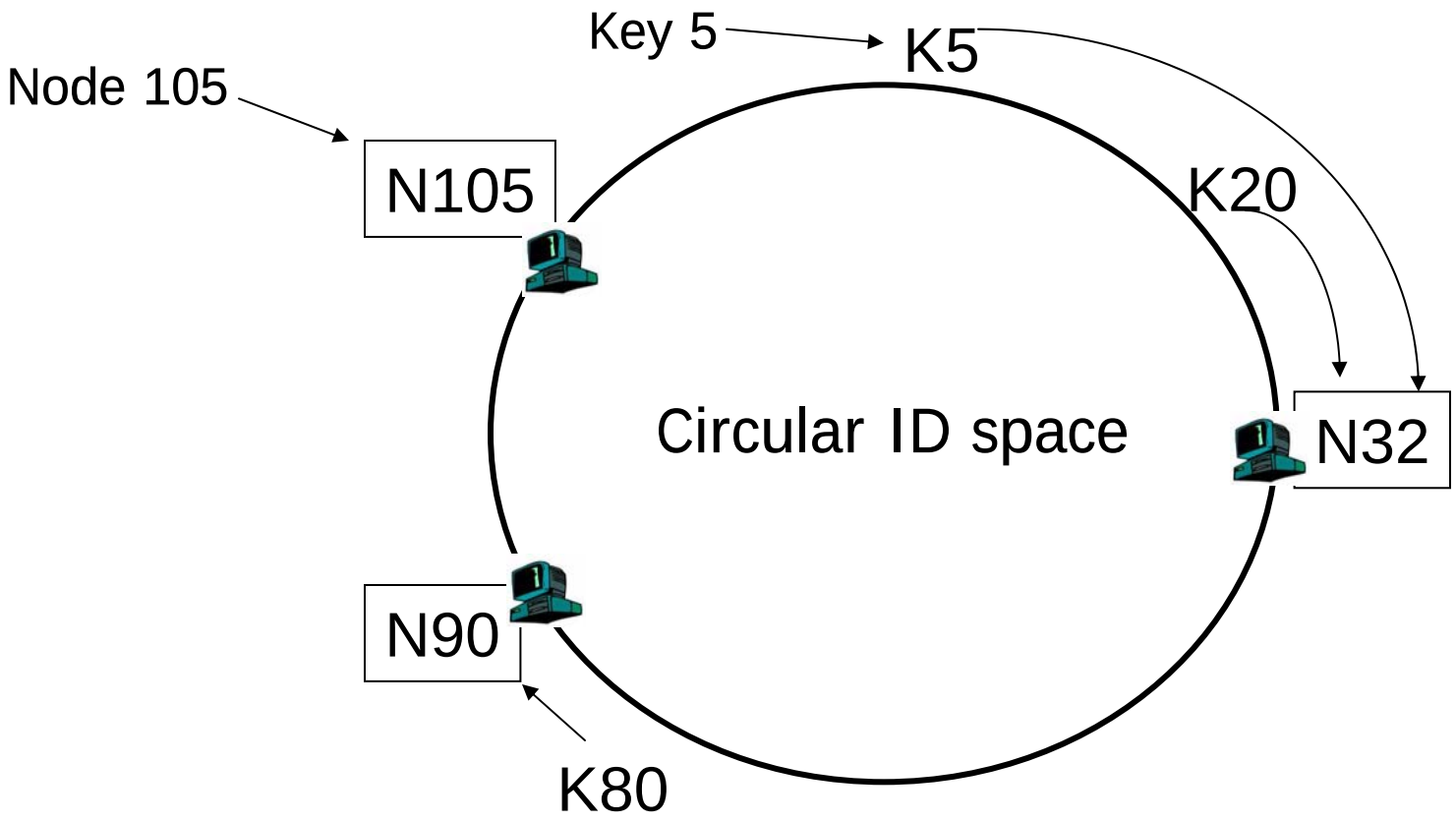
- Structured Overlay Routing:
  - **Join**: On startup, contact a "bootstrap" node and integrate yourself into the distributed data structure; get a *node id*
  - **Publish**: Route publication for *file id* toward a close *node id* along the data structure
  - **Search**: Route a query for file id toward a close node id. Data structure guarantees that query will meet the publication.
  - **Fetch**: Two options:
    - Publication contains actual file => fetch from where query stops
    - Publication says "I have file X" => query tells you 128.2.1.3 has X, use IP routing to get X from 128.2.1.3

DHT: Example - Chord

- Associate to each node and file a unique *id*, e.g., pick from the range [0...2^m]
  - Usually the hash of the file or IP address
- Properties:
  - Routing table size is O(log *N*) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log *N*) hops

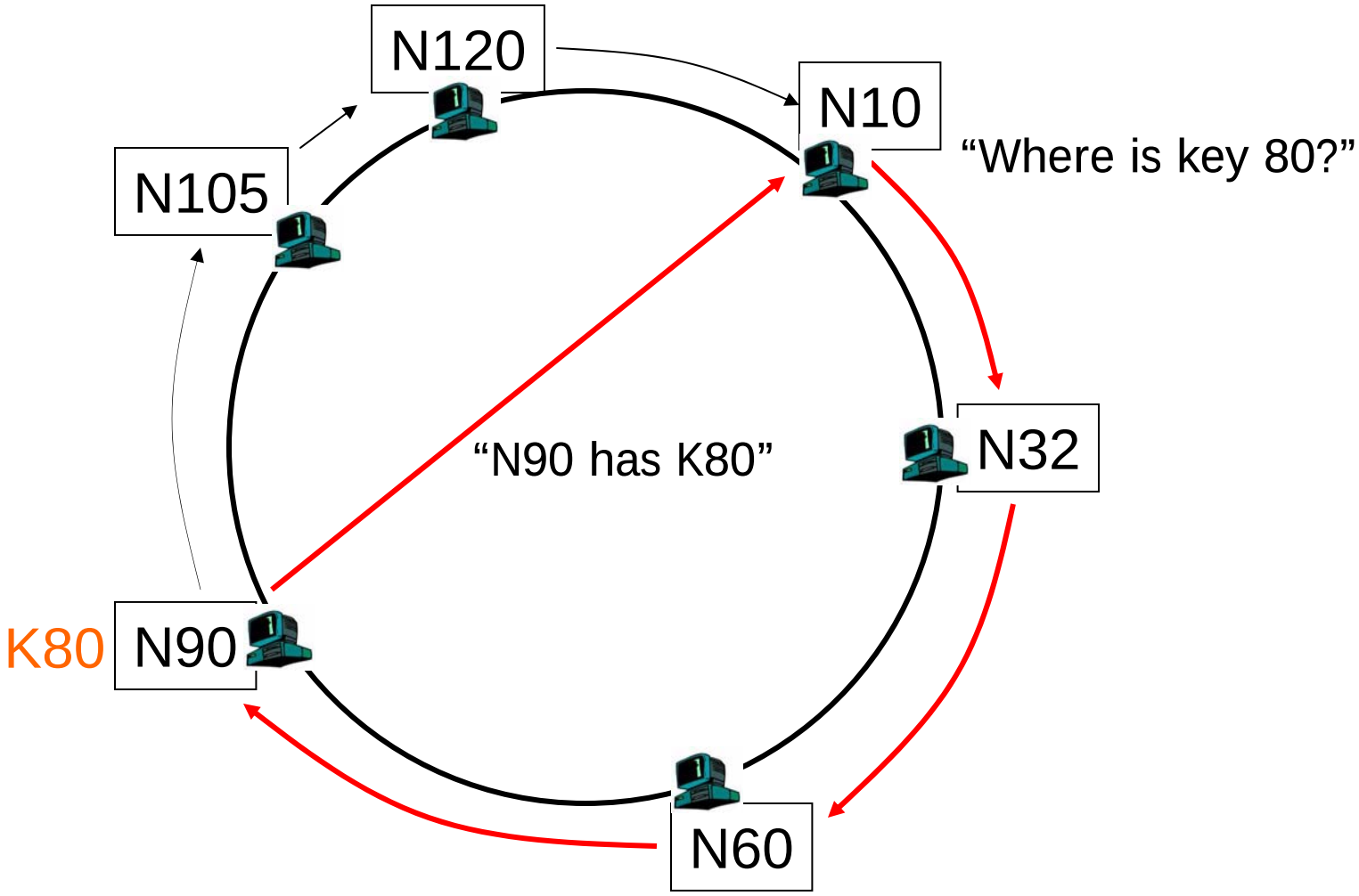
from MIT in 2001

DHT: Consistent Hashing

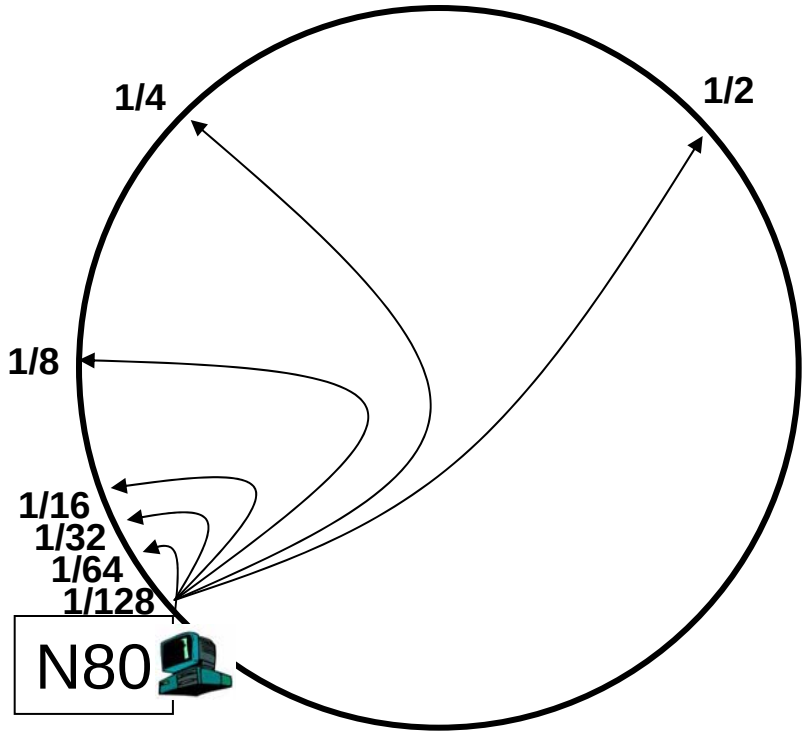


A key is stored at its successor: node with next higher ID

DHT: Chord Basic Lookup



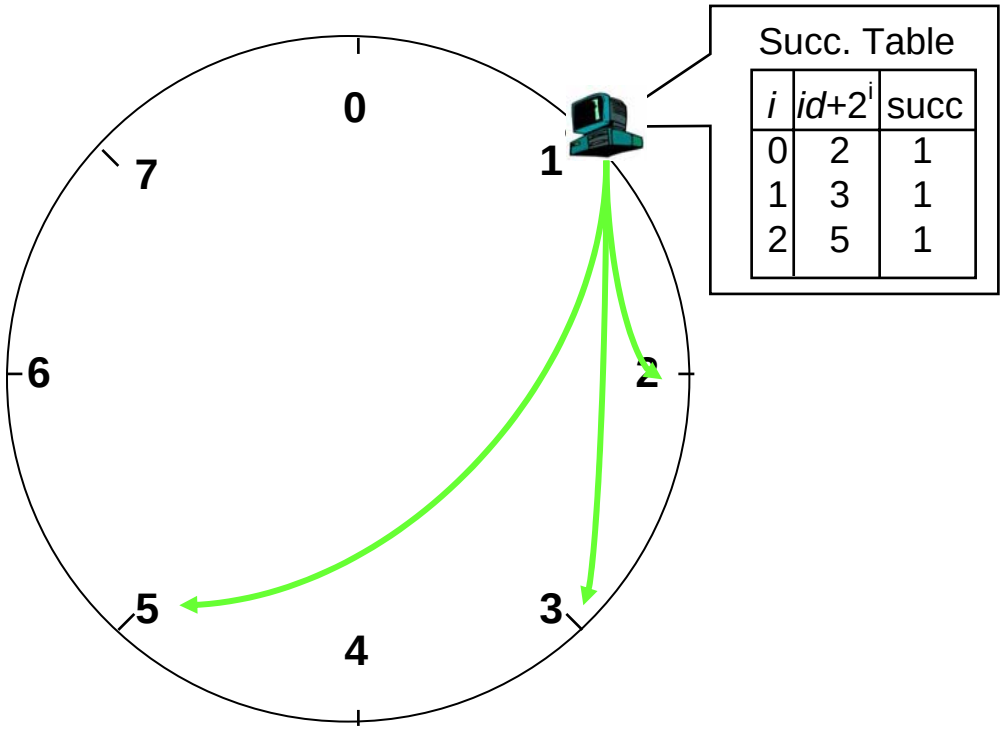
DHT: Chord "Finger Table"



- Entry *i* in the finger table of node *n* is the first node that succeeds or equals $n + 2^i$
- In other words, the ith finger points $1/2^{n-i}$ way around the ring

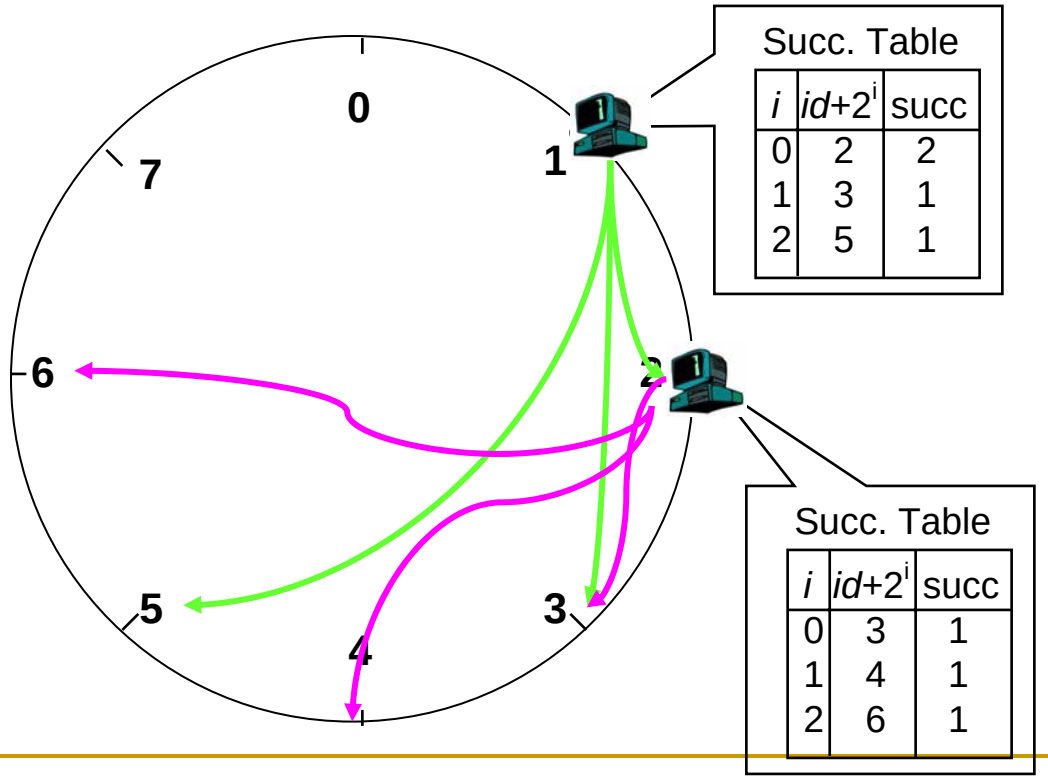
DHT: Chord Join

- Assume an identifier space [0..8]
- Node n1 joins



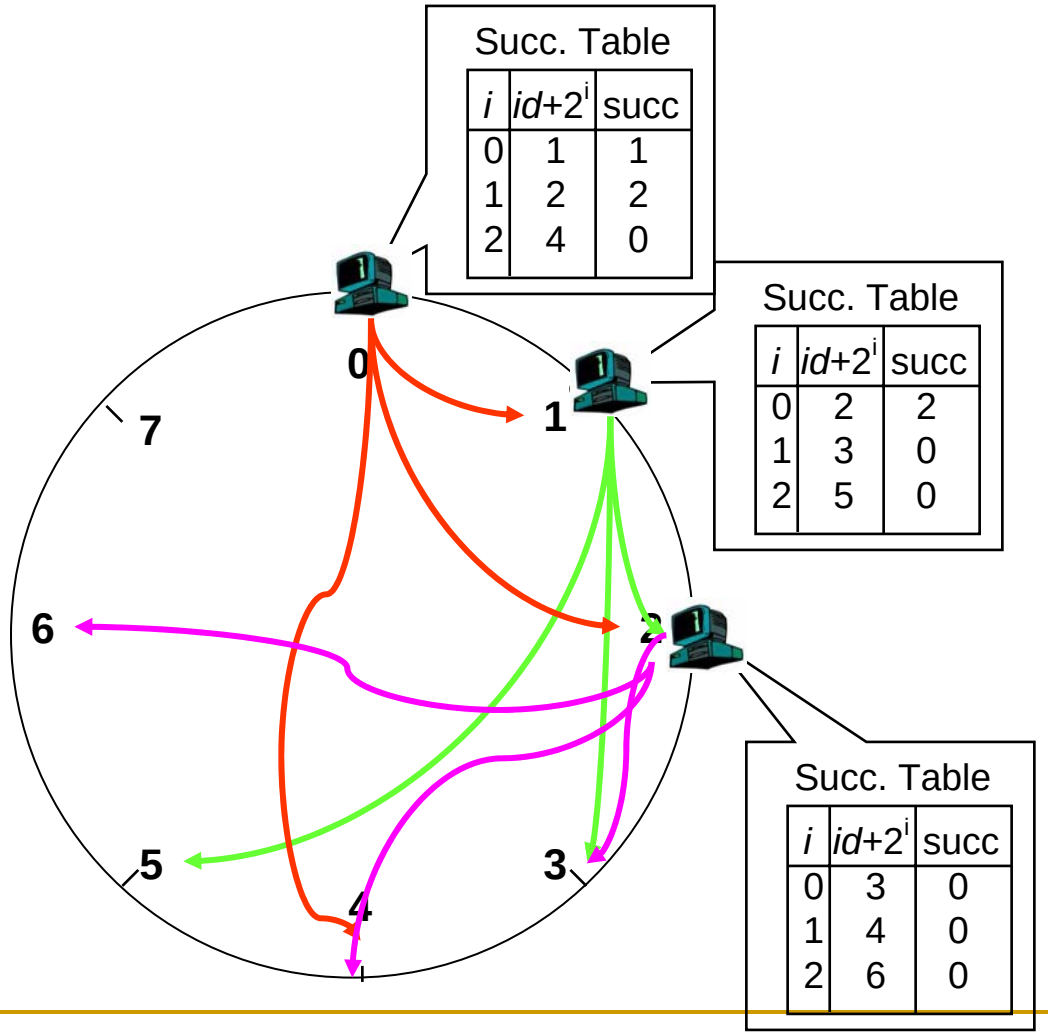
DHT: Chord Join

- Node n2 joins



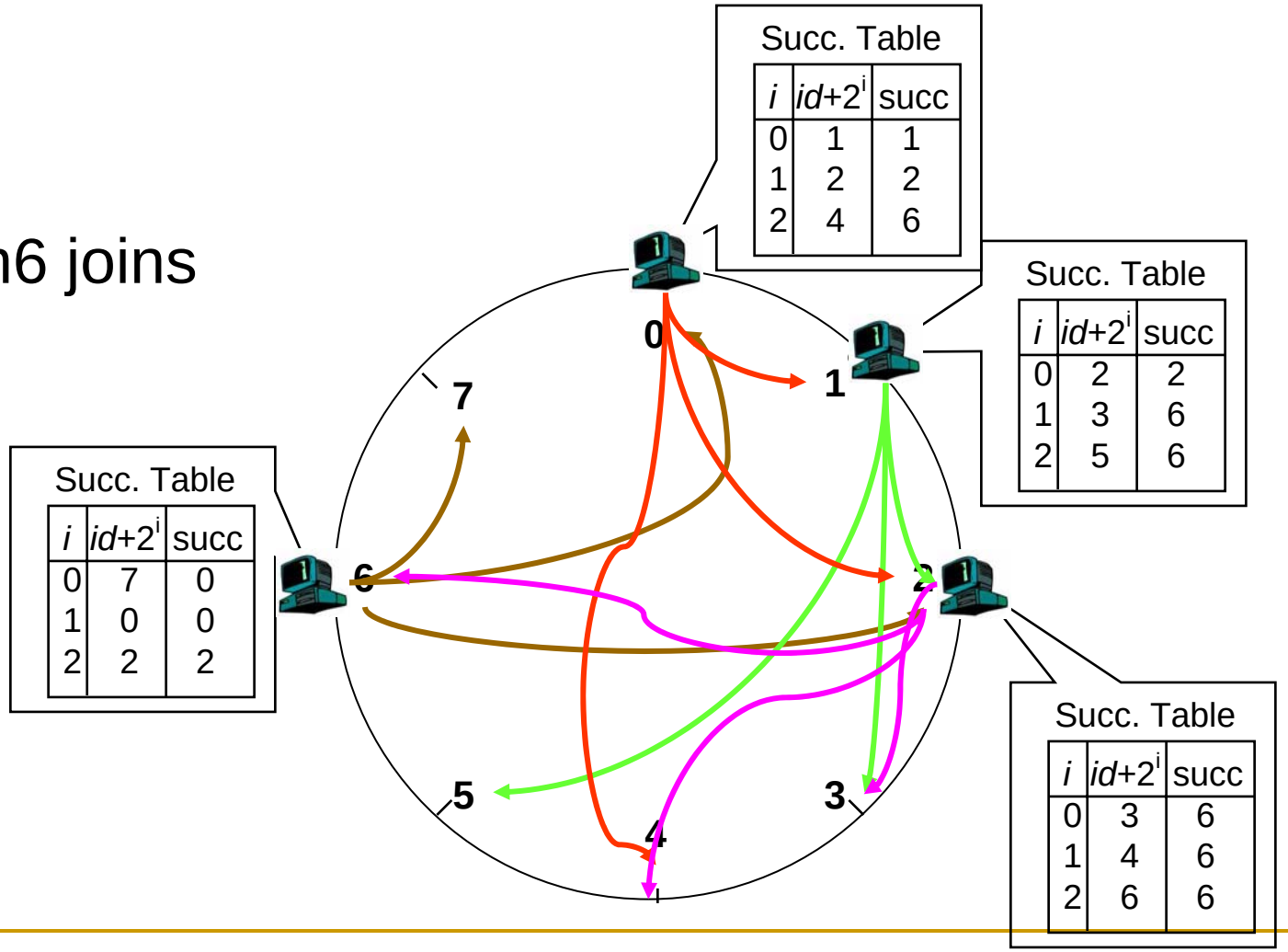
DHT: Chord Join

- Nodes n0 joins



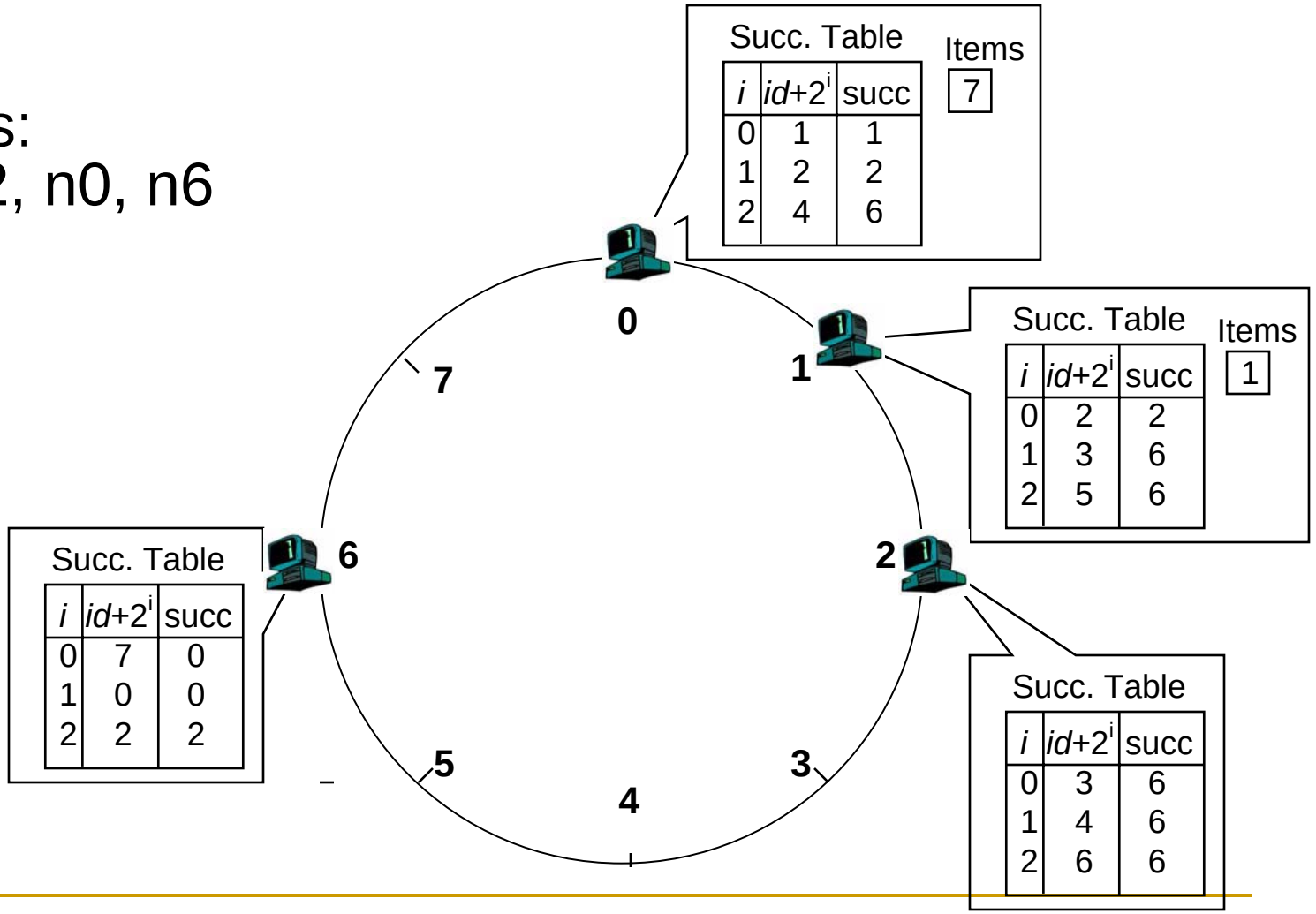
DHT: Chord Join

- Nodes n6 joins



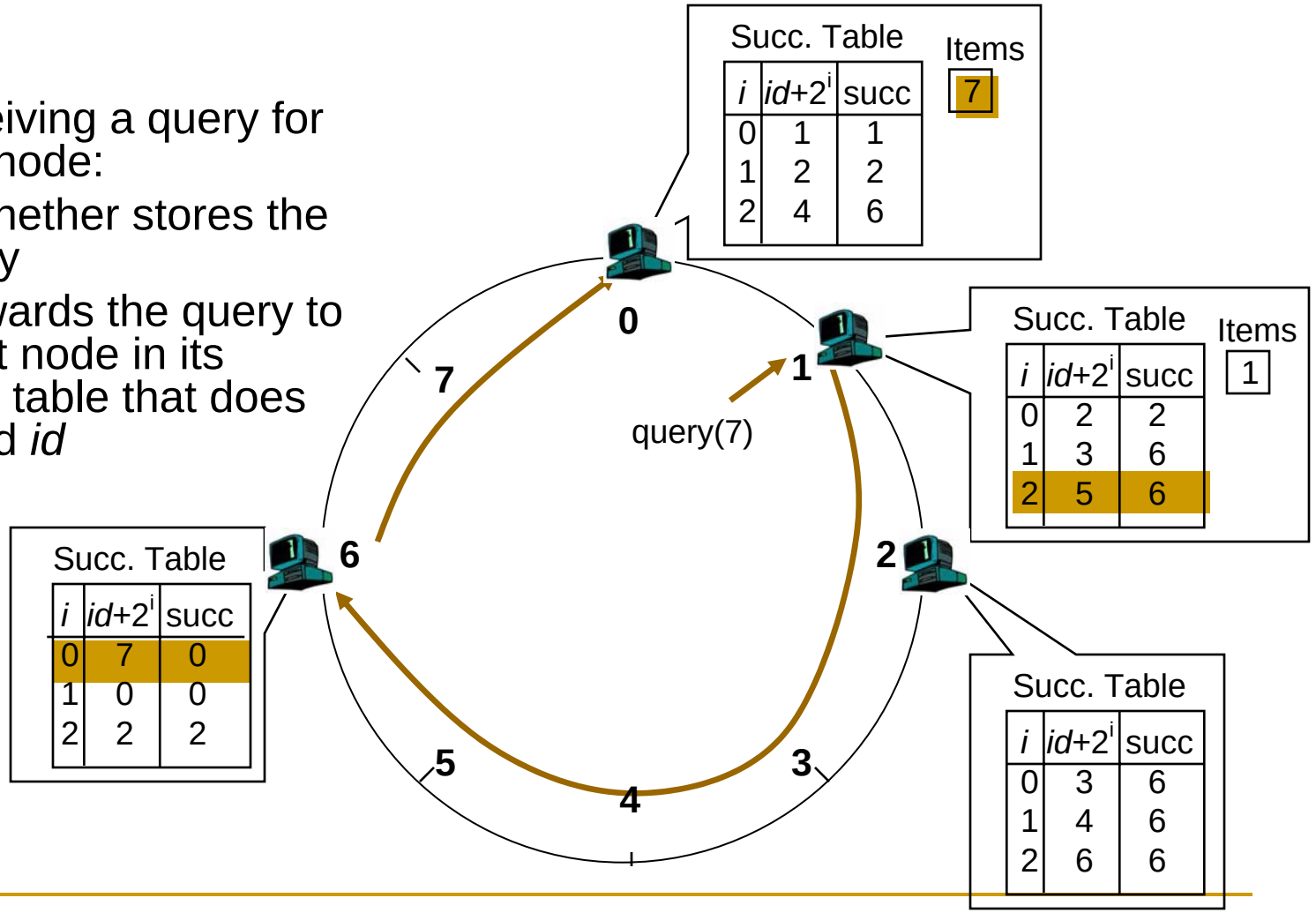
DHT: Chord Join

- Nodes:
  n1, n2, n0, n6
- Items:
  f7, f1



DHT: Chord Routing

- Upon receiving a query for item *id*, a node:
- Checks whether stores the item locally
- If not, forwards the query to the largest node in its successor table that does not exceed *id*



DHT: Chord Summary

- Routing table size?
  - Log *N* fingers
- Routing time?
  - Each hop expects to 1/2 the distance to the desired id => expect O(log *N*) hops.

DHT: Discussion

- Pros:
  - Guaranteed Lookup
  - O(log *N*) per node state and search scope
- Cons:
  - No one uses them? (only one file sharing app)
  - Supporting non-exact match search is hard

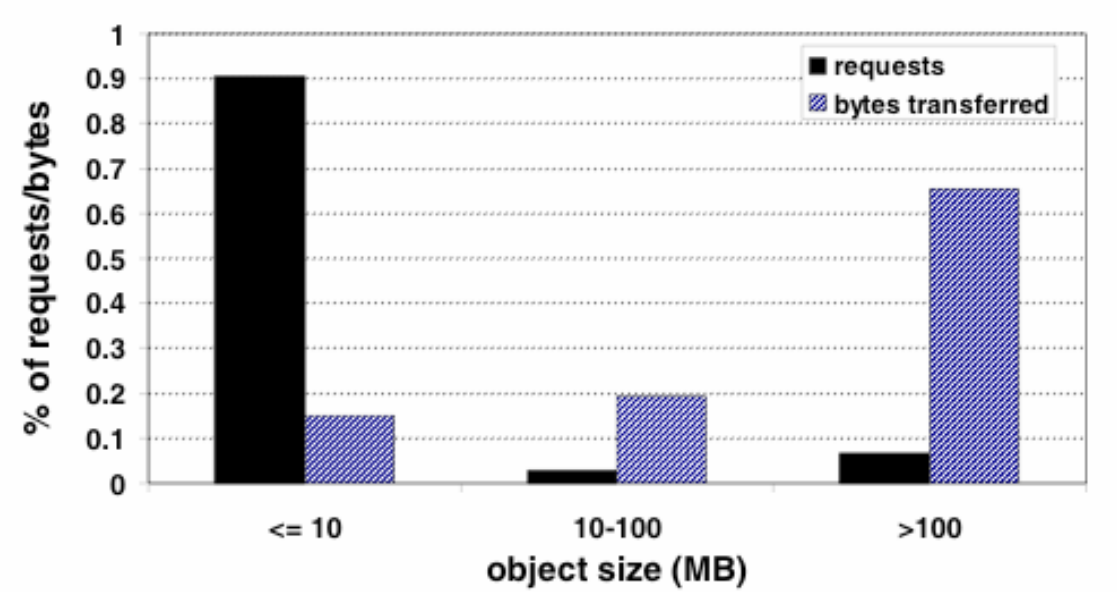
P2P: Summary

- Many different styles; remember pros and cons of each
  - centralized, flooding, swarming, unstructured and structured routing
- Lessons learned:
  - Single points of failure are very bad
  - Flooding messages to everyone is bad
  - Underlying network topology is important
  - Not all nodes are equal
  - Need incentives to discourage freeloading
  - Privacy and security are important
  - Structure can provide theoretical bounds and guarantees

Extra Slides

KaZaA: Usage Patterns

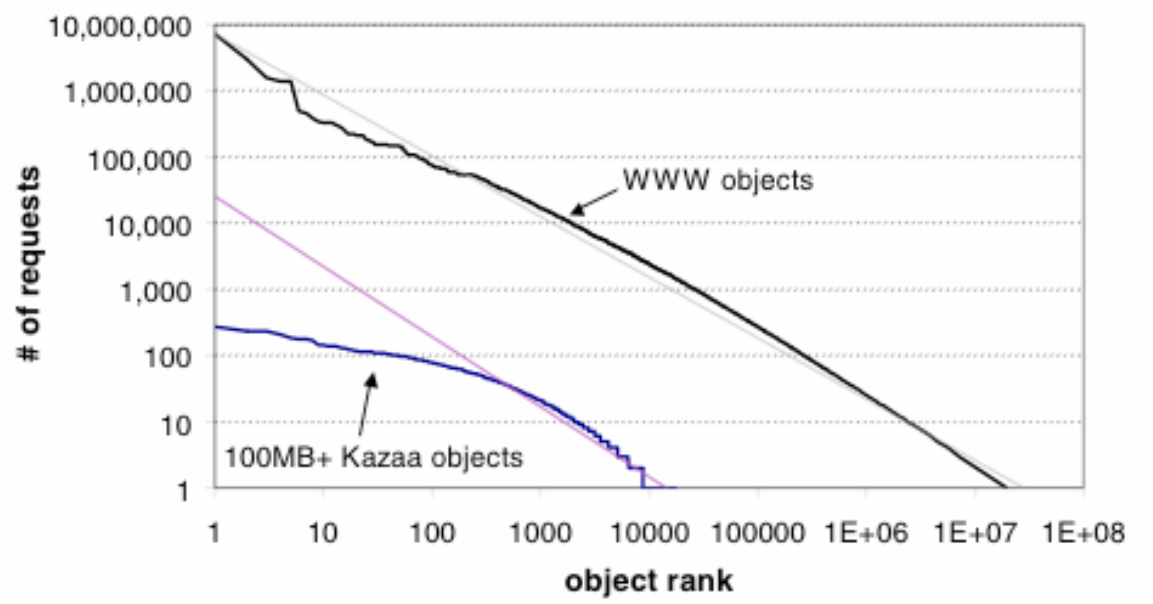
- KaZaA is more than one workload!
  - Many files < 10MB (e.g., Audio Files)
  - Many files > 100MB (e.g., Movies)



from Gummadi *et al.*, *SOSP* 2003

KaZaA: Usage Patterns (2)

- KaZaA is not Zipf!
  - FileSharing: "Request-once"
  - Web: "Request-repeatedly"



from Gummadi *et al.*, *SOSP* 2003