Reti P2P per file distribution: BitTorrent

Corso di ***Applicazioni Telematiche***

A.A. 2008-09– Lezione n.22

Prof. Roberto Canonico

Università degli Studi di Napoli Federico II

Facoltà di Ingegneria

P2p file-sharing

- Quickly grown in popularity
  - Dozens or hundreds of file sharing applications
  - 35 million American use P2P networks -- 29% of all Internet users in US!
  - Audio/Video transfer now dominates traffic on the Internet

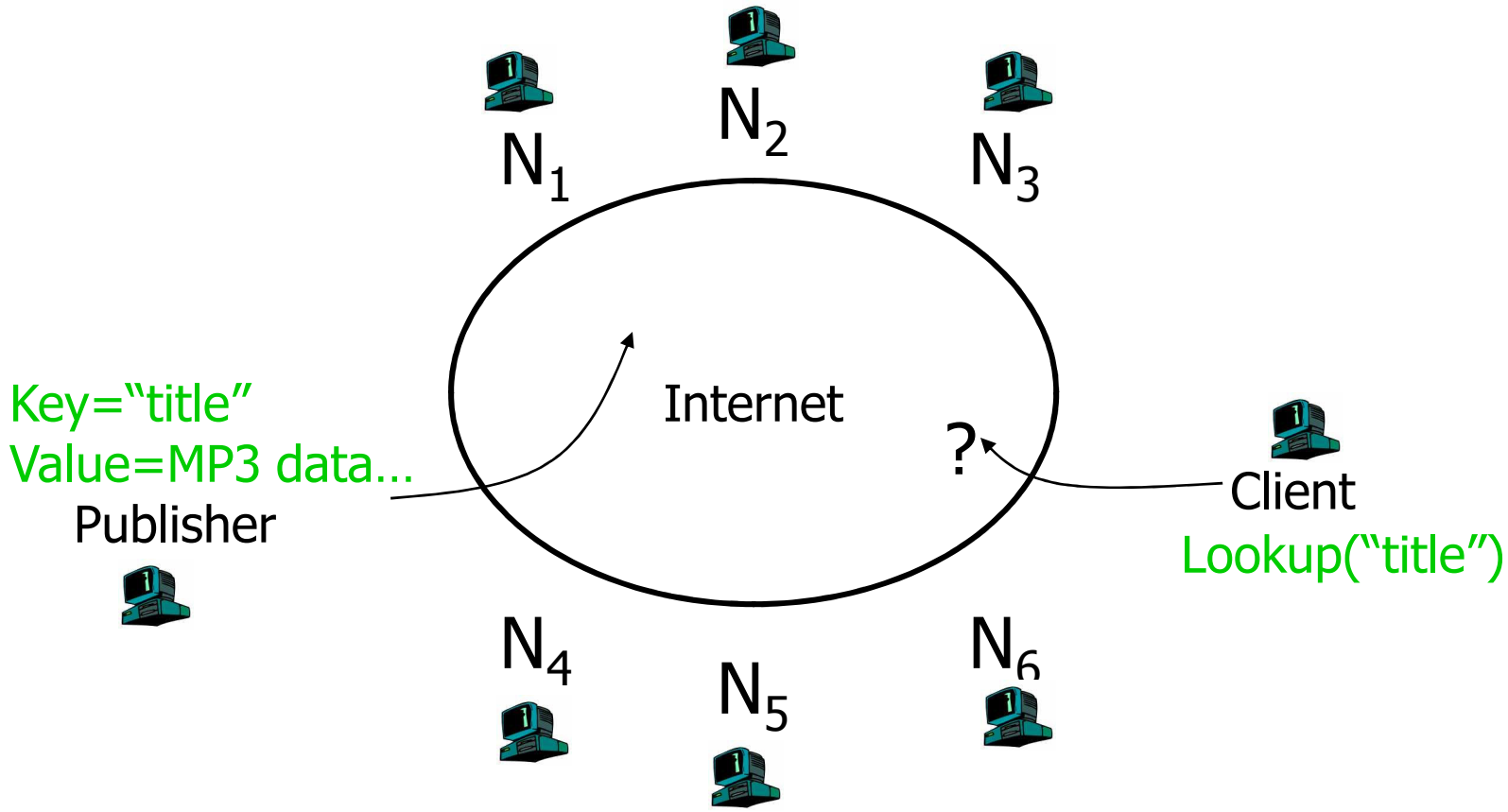
The p2p challenge

- C1: Search(human's goals) -> file
  - Given keywords / human description, find a specific file
- C2: Fetch(file) -> bits

What's out there?

	Central	Flood	Super-node flood	Route
Whole File	Napster	Gnutella		Freenet
Chunk Based	BitTorrent		KaZaA (bytes, not chunks)	DHTs eDonkey2000 New BT

Searching


N1
N2
N3
N4
N5
N6
Internet
Key="title"
Value=MP3 data…
Publisher
?
Client
Lookup("title")

Searching 2

- Needles vs. Haystacks
  - Searching for top 40, or an obscure punk track from 1981 that nobody's heard of?
- Search expressiveness
  - Whole word? Regular expressions? File names? Attributes? Whole-text search?
    - (e.g., p2p gnutella or p2p google?)

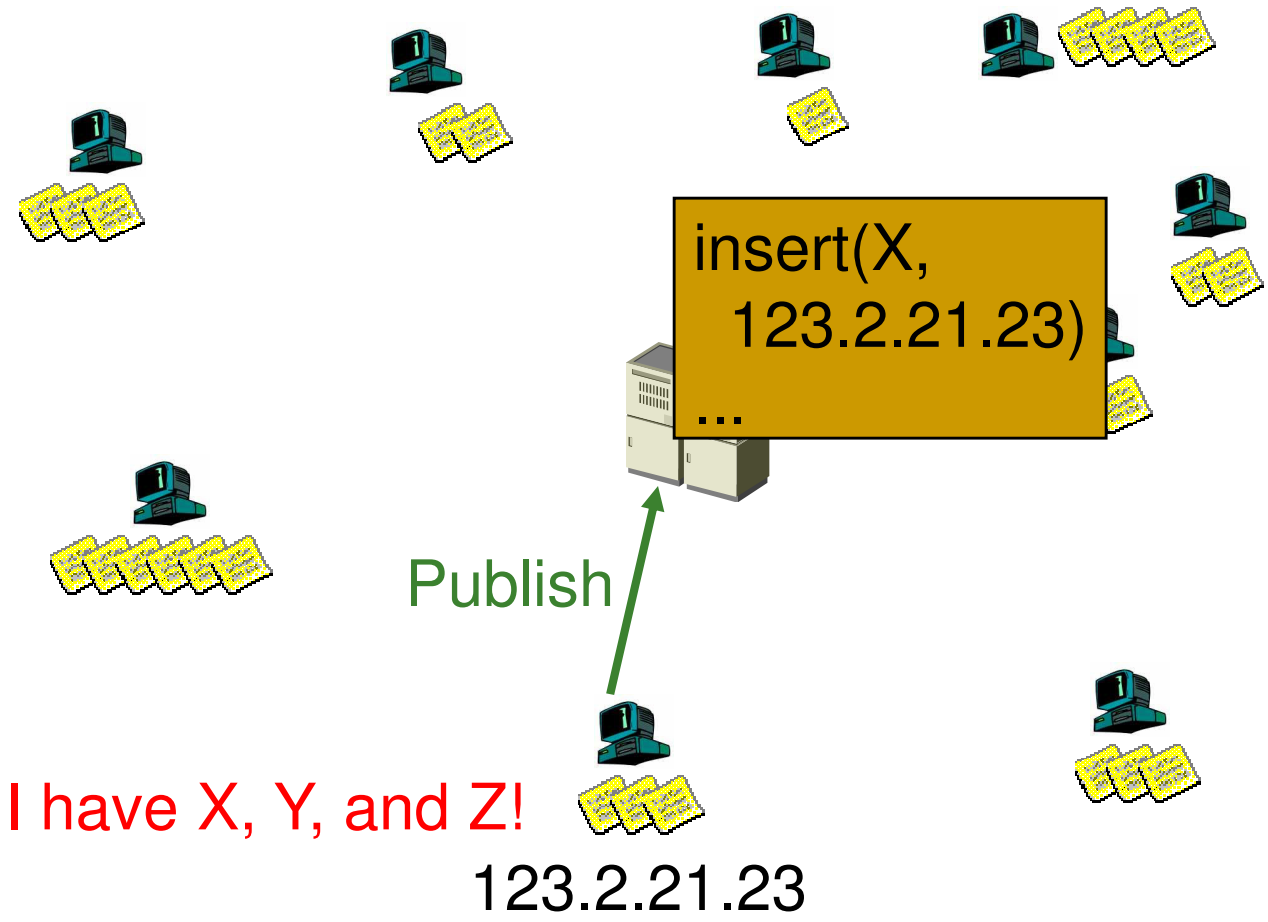
Framework

- 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?

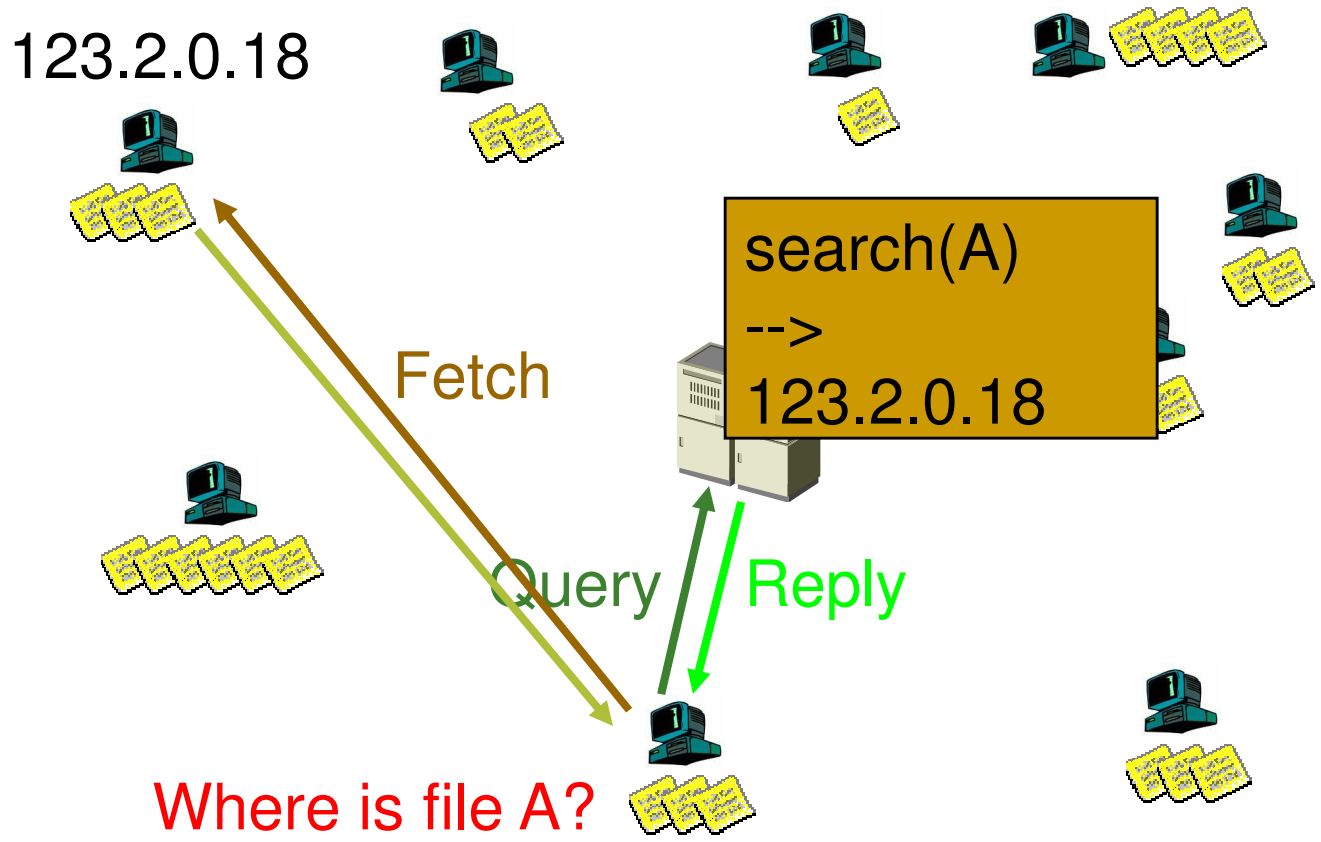
Centralized Database: Napster

- **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



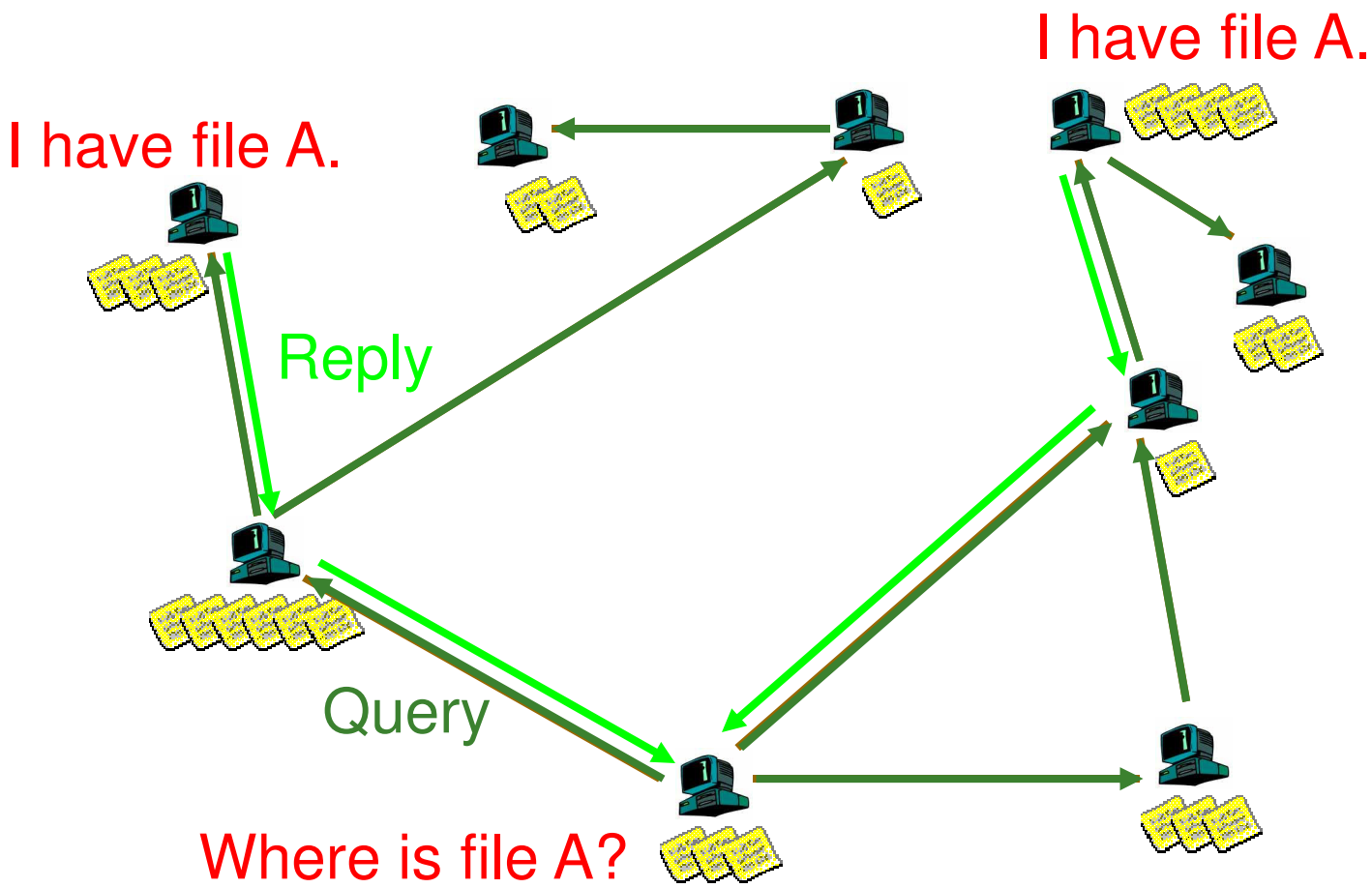
Napster: discussion

- 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

Query Flooding

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

Search in Query Flooding



Flooding Discussion

- Pros:
  - Fully de-centralized
  - Search cost distributed
  - Processing @ each node permits powerful search semantics
- Cons:
  - Search scope is O(*N*)
  - Search time is O(???)
  - Nodes leave often, network unstable
- TTL-limited search works well for haystacks.
  - For scalability, does NOT search every node. May have to re-issue query later

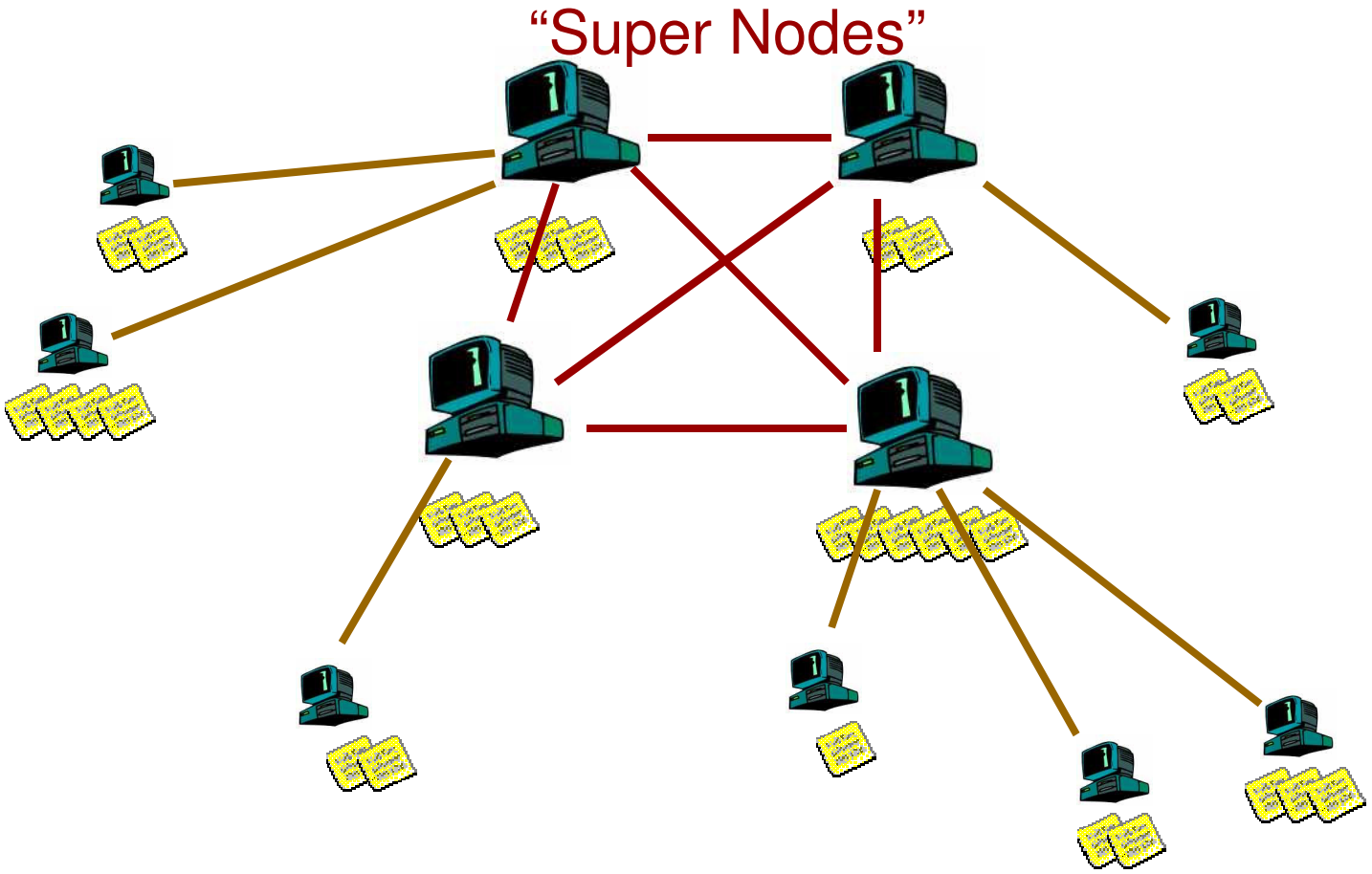
Query Floding: Gnutella

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

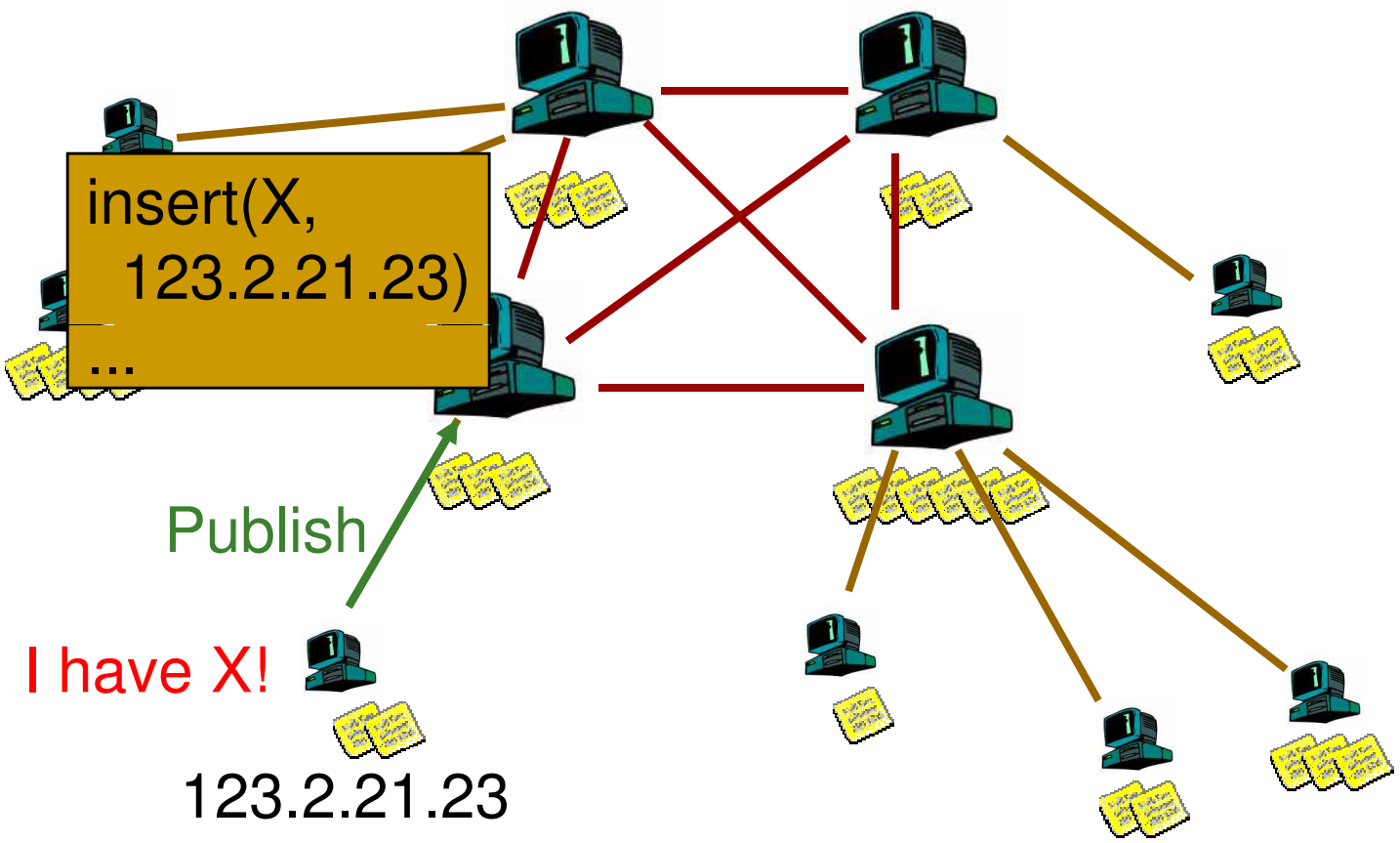
Flooding with Supernodes

- "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

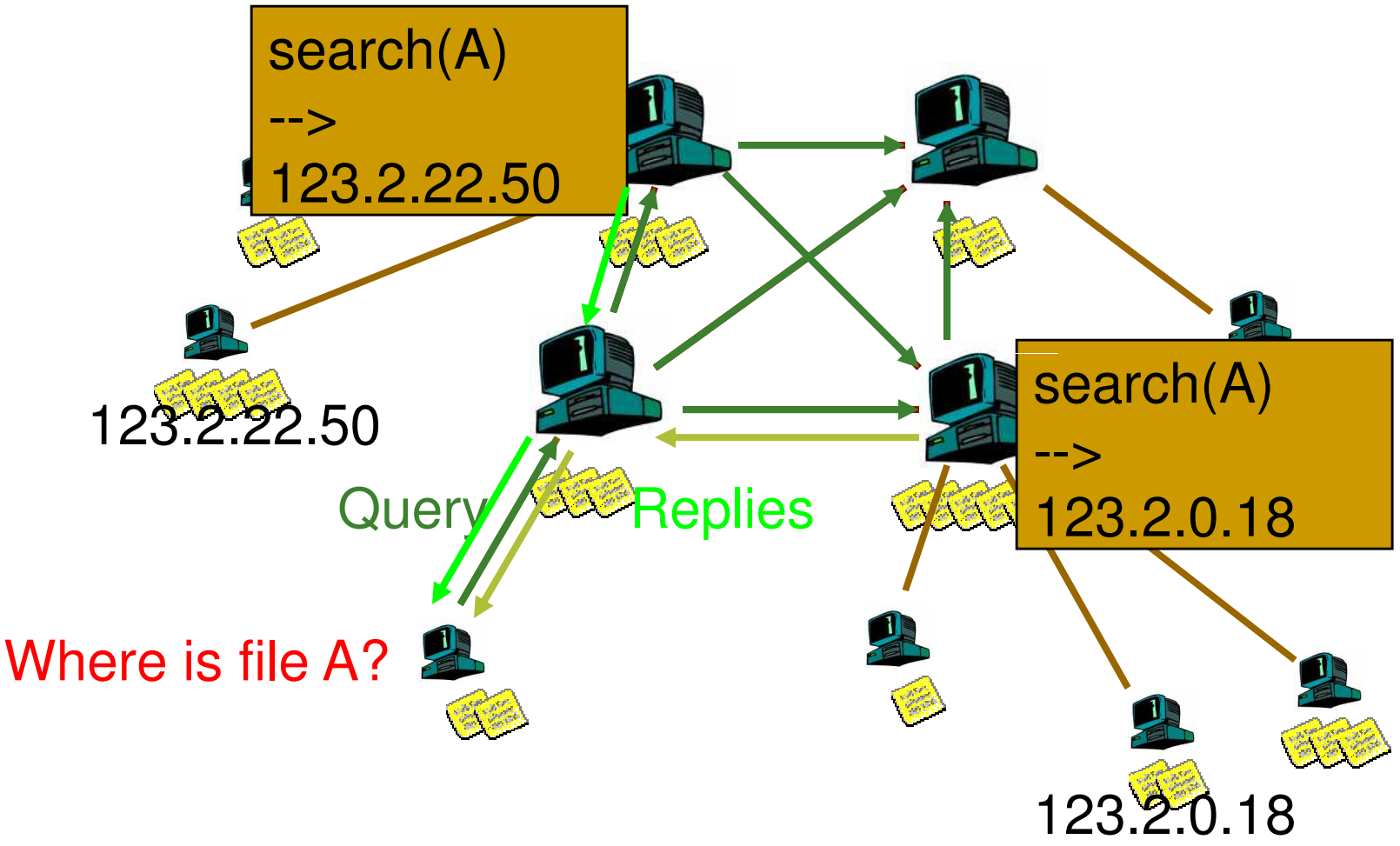
Supernodes Network Design



Supernodes: File Insert



Supernodes: File Search



Supernodes: Fetching (And use of hashes…)

- More than one node may have the 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

Supernode Flooding Discussion

- Pros:
  - Tries to take into account node heterogeneity:
    - Bandwidth
    - Host Computational Resources
    - Host Availability (?)
  - Rumored to take into account network locality
  - Scales better
- Cons:
  - Mechanisms easy to circumvent
  - Still no real guarantees on search scope or search time
- Similar behavior to plain flooding, but better.

Stability and Superpeers

- Why superpeers?
  - Query consolidation
    - Many connected nodes may have only a few files
    - Propagating a query to a sub-node would take more b/w than answering it yourself
  - Caching effect
    - Requires network stability
- Superpeer selection is time-based
  - How long you've been on is a good predictor of how long you'll be around.

Superpeers: KaZaA

- In 2001, KaZaA created by Dutch company Kazaa BV
- 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 in 2005 with >10 million users (number varies)

Searching & Fetching

- Query flooding finds:
  - An object
    - Filename?
    - Hash?
  - A host that serves that object
- In QF systems, d/l from the host that answered your query
- Generally uses only one source…

Fetching

- When you have an object ID,
- Get a list of peers serving that ID
  - Easier than the keyword lookup
  - Queries are structured
- Download in parallel from multiple peers
- “Swarming”
  - Download from others downloading same object at same time

Swarming: BitTorrent

- 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.
- Big differences from Napster:
  - Chunk based downloading (sound familiar? :)
  - “few large files” focus
  - Anti-freeloading mechanisms

BitTorrent: Sharing Strategy

- Employ "Tit-for-tat" sharing strategy
  - A is downloading from some other people
    - A will let the fastest N of those download from him
  - 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
  - Let N peop
- Goal: Pareto Efficiency
  - Game Theory: "No change can make anyone better off without making others worse off"
  - Does it get there? No, but it's reasonable

BitTorrent: Summary

- Pros:
  - Works reasonably well in practice
  - Gives peers incentive to share resources; avoids freeloaders
- Cons:
  - Pareto Efficiency relative weak condition
  - Central tracker server needed to bootstrap swarm
  - Tracker is a design choice, not a requirement. Newer BT variants use a “distributed tracker” - a Distributed Hash Table

Distributed Hash Tables

- Academic answer to p2p
- Goals
  - Guatanteed lookup success
  - Provable bounds on search time
  - Provable scalability
- Makes some things harder
  - Fuzzy queries / full-text search / etc.
- Read-write, not read-only
- Hot Topic in networking since introduction in ~2000/2001

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)

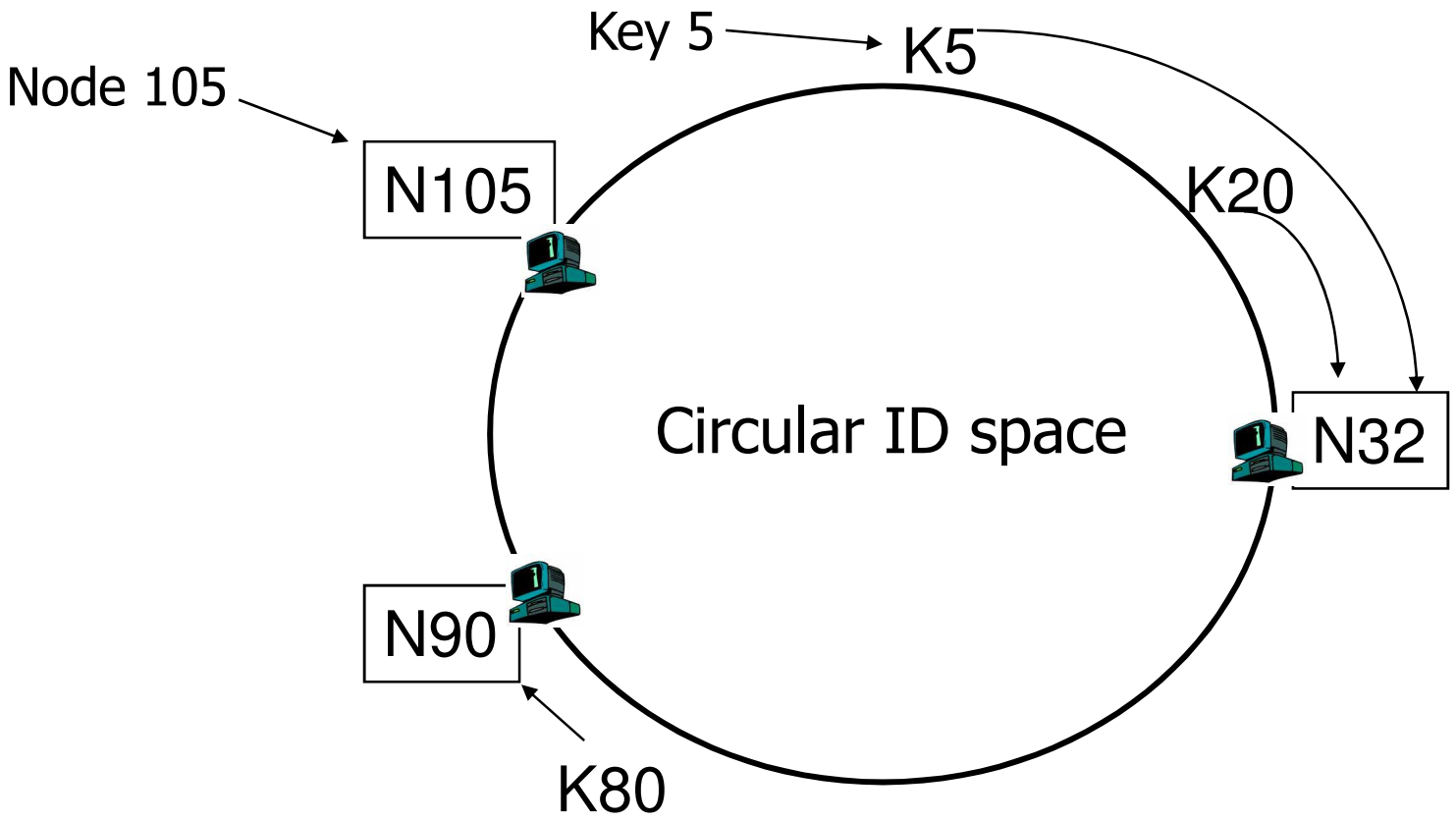
- Structured Overlay Routing:
  - **Join**: On startup, contact a "bootstrap" node and integrate yourself into the distributed data structure; get a *node id*
  - **Publish**: Route publication for *file id* toward a close *node id* along the data structure
  - **Search**: Route a query for file id toward 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* in an *uni*-dimensional space (a Ring)
  - 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

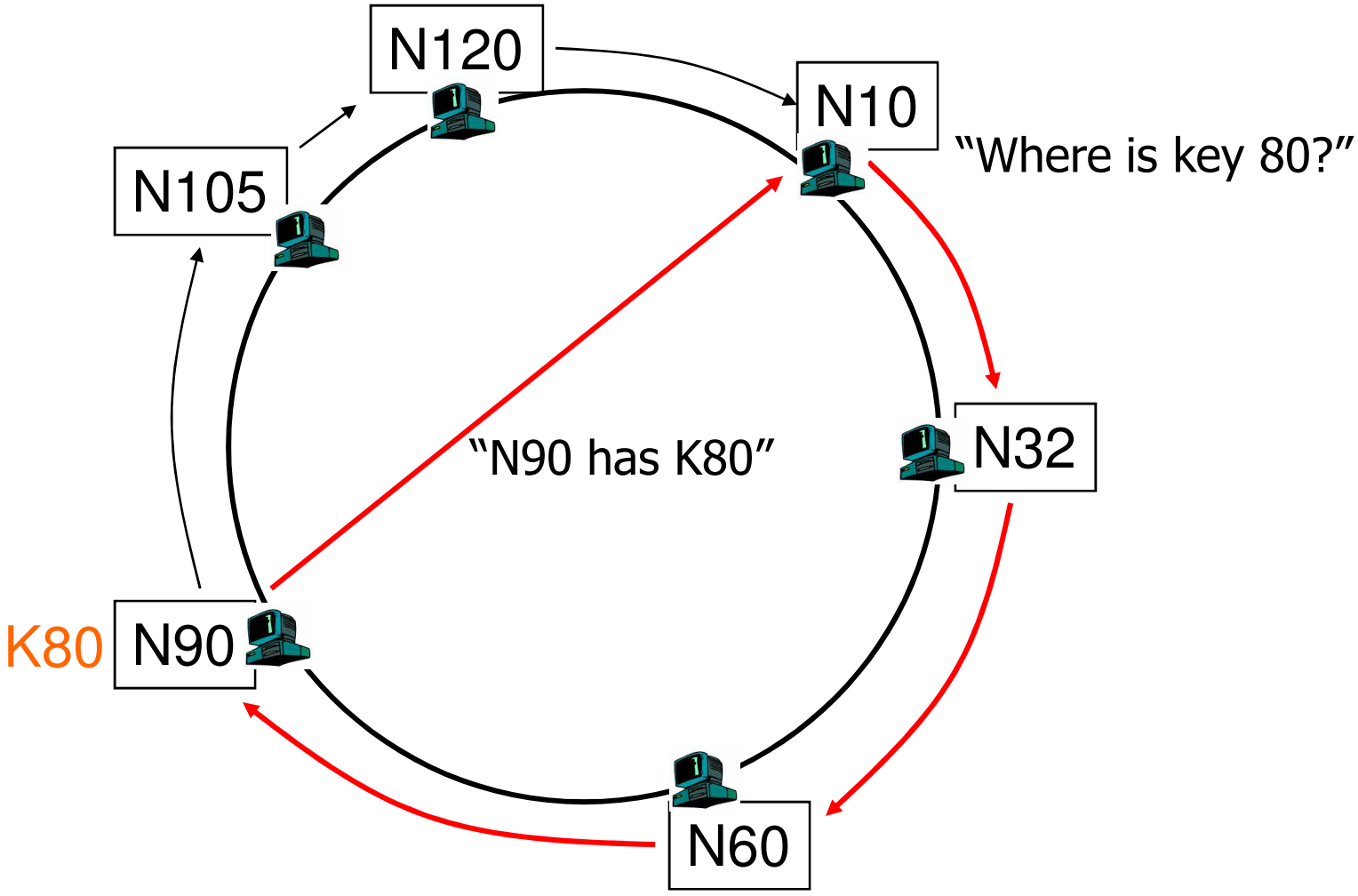
from MIT in 2001

DHT: Consistent Hashing

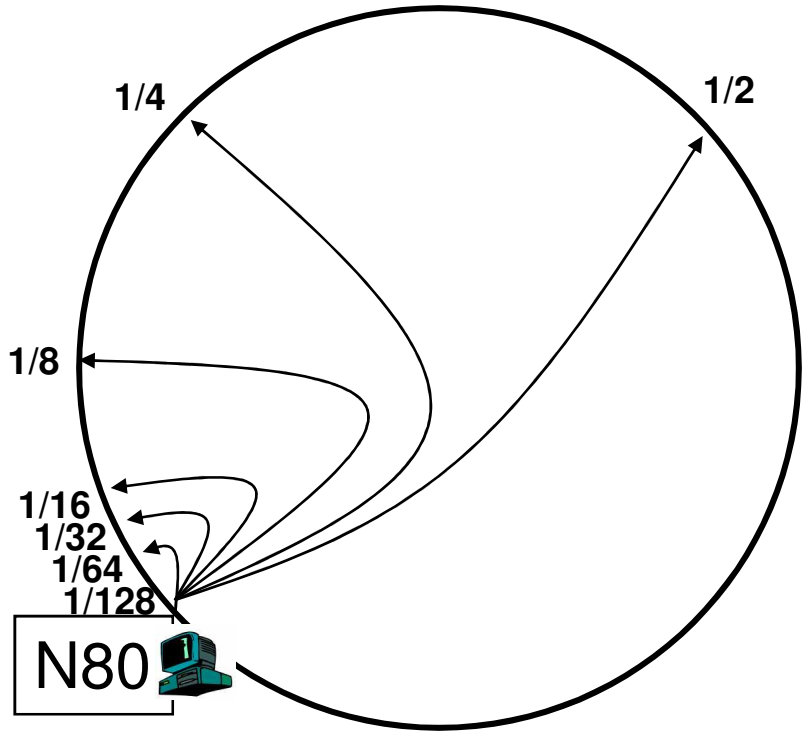


A key is stored at its successor: node with next higher ID

DHT: Chord Basic Lookup



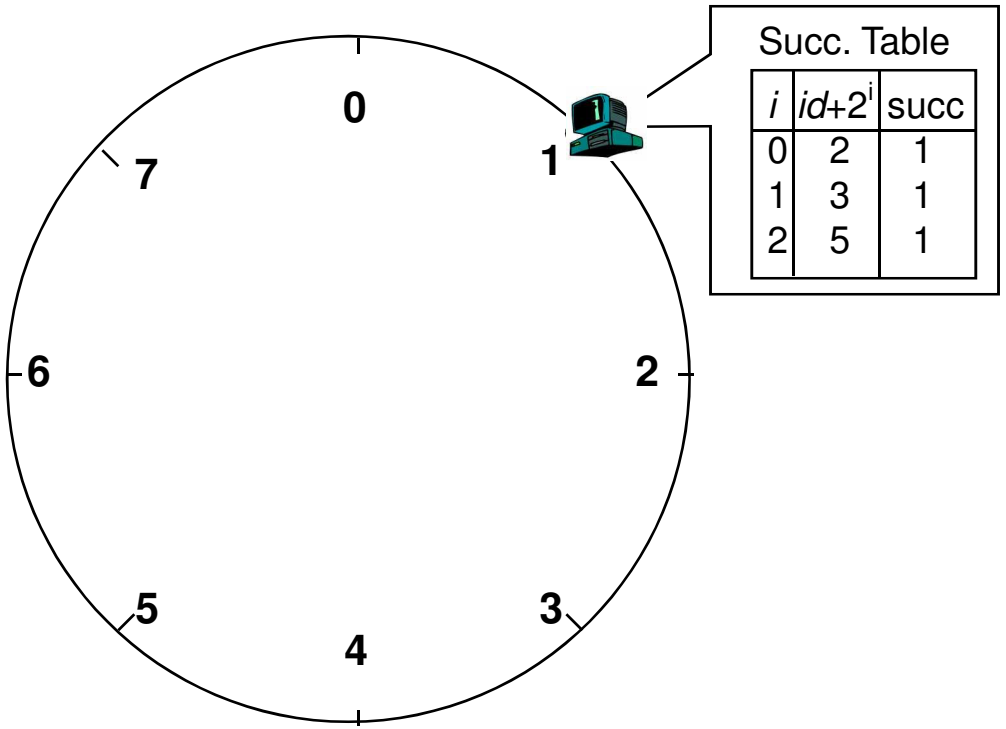
DHT: Chord "Finger Table"



- 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

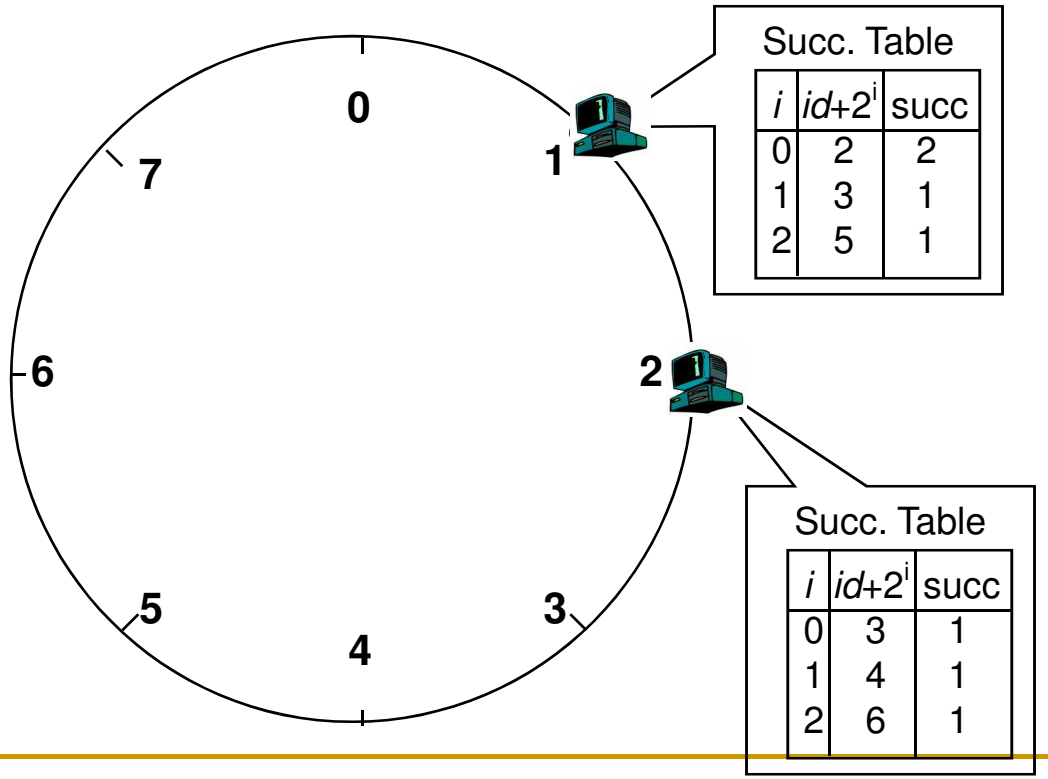
DHT: Chord Join

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



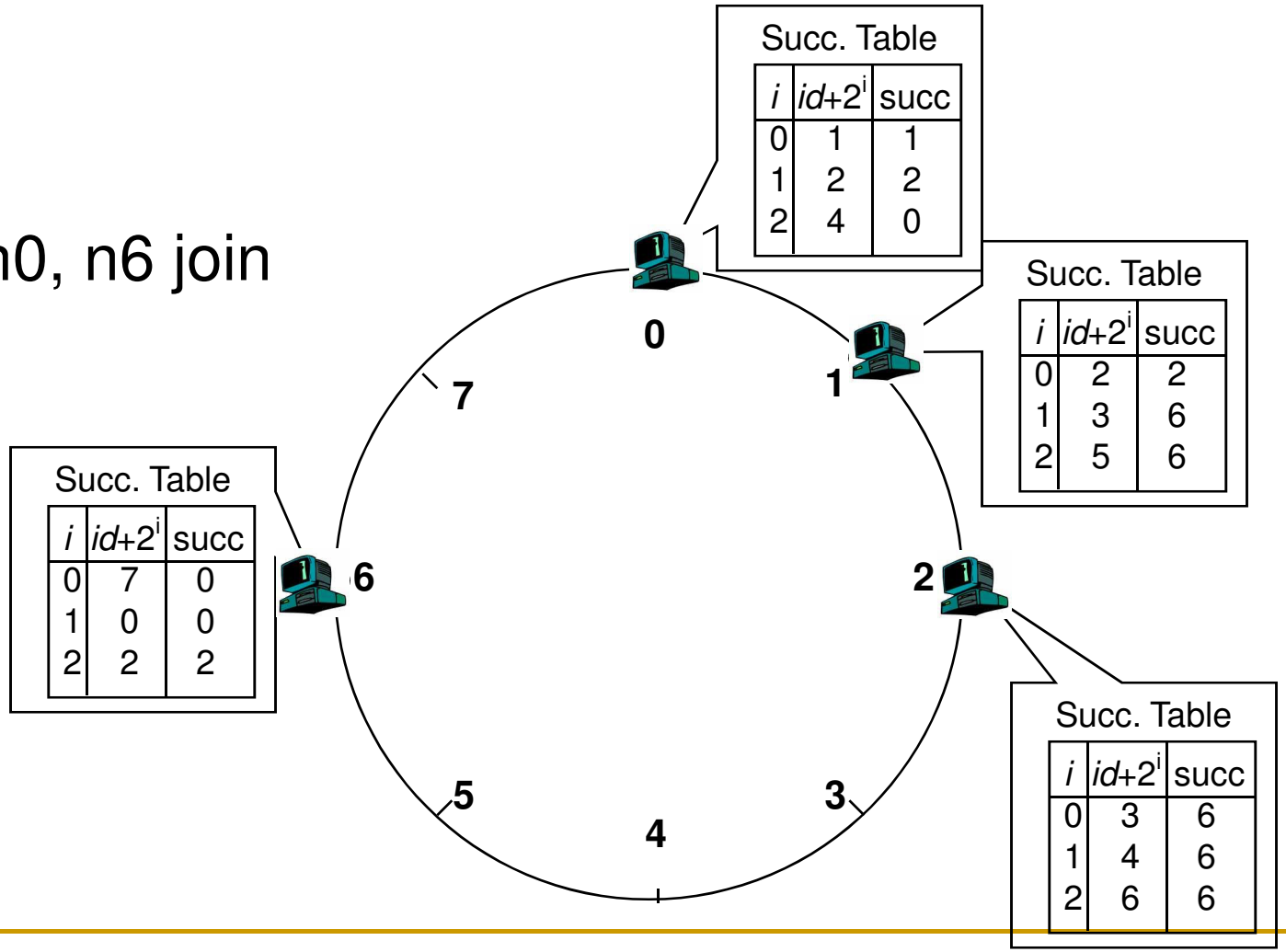
DHT: Chord Join

- Node n2 joins



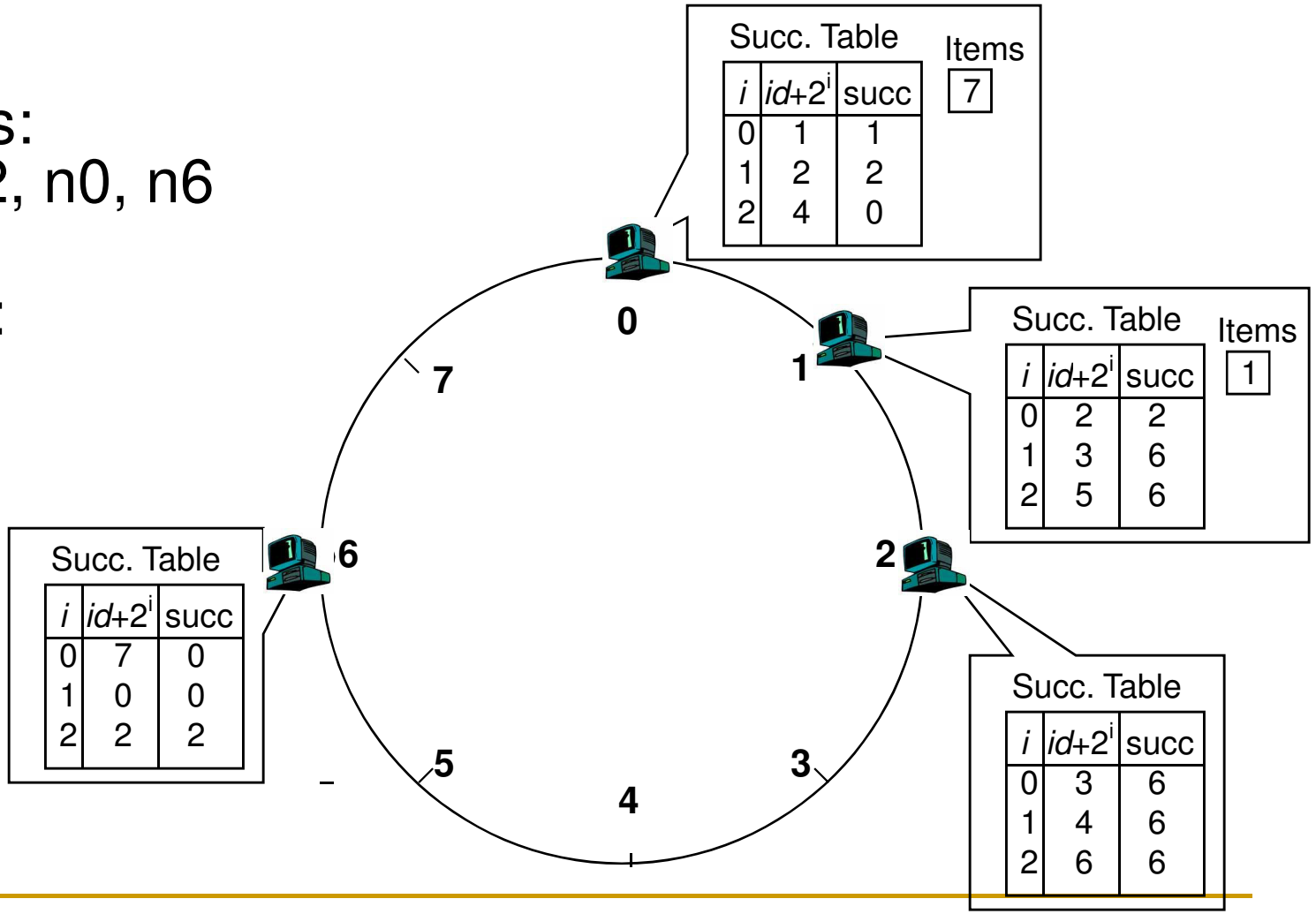
DHT: Chord Join

- Nodes n0, n6 join



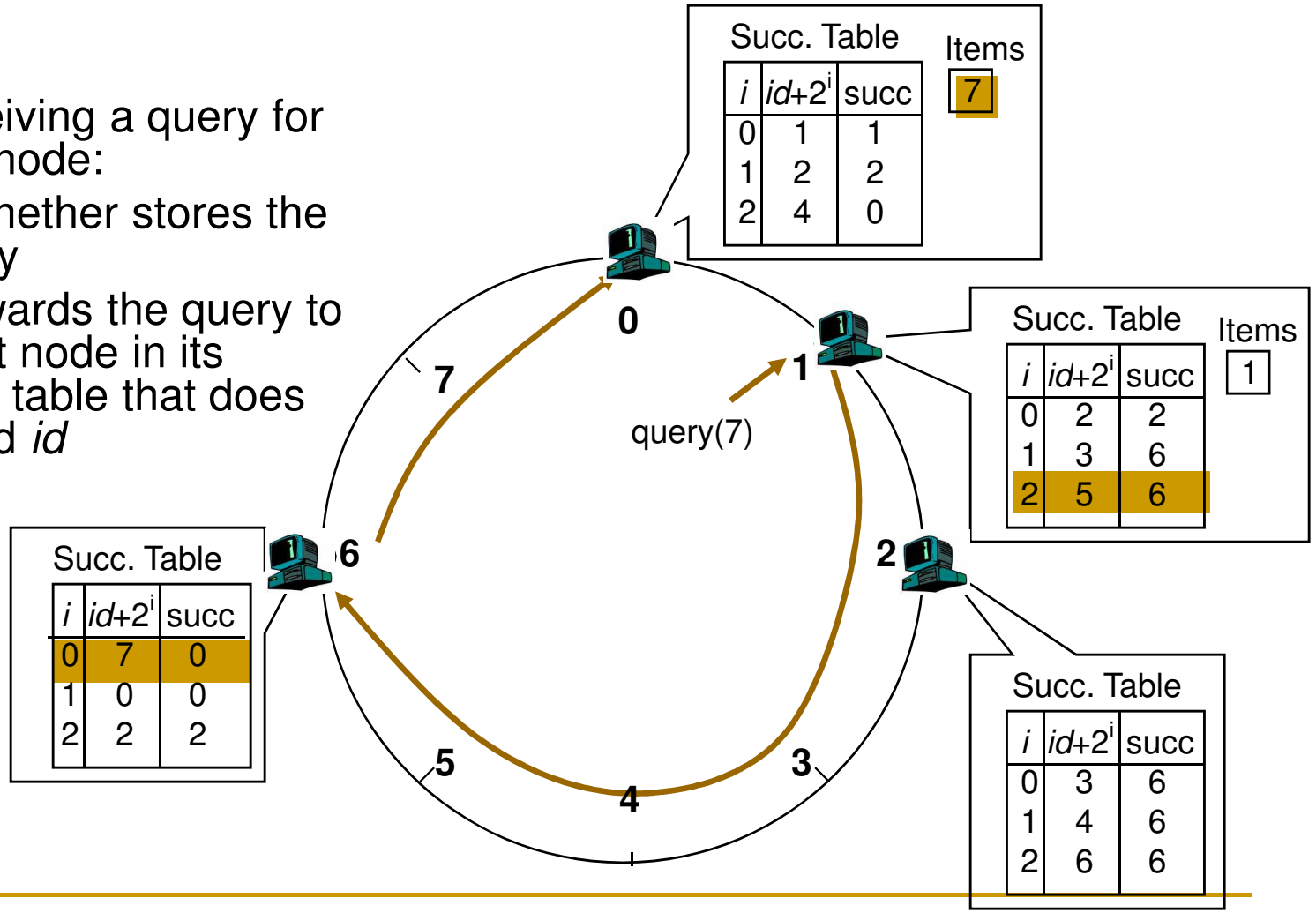
DHT: Chord Join

- Nodes:
  n1, n2, n0, n6
- Items:
  f7, f2



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

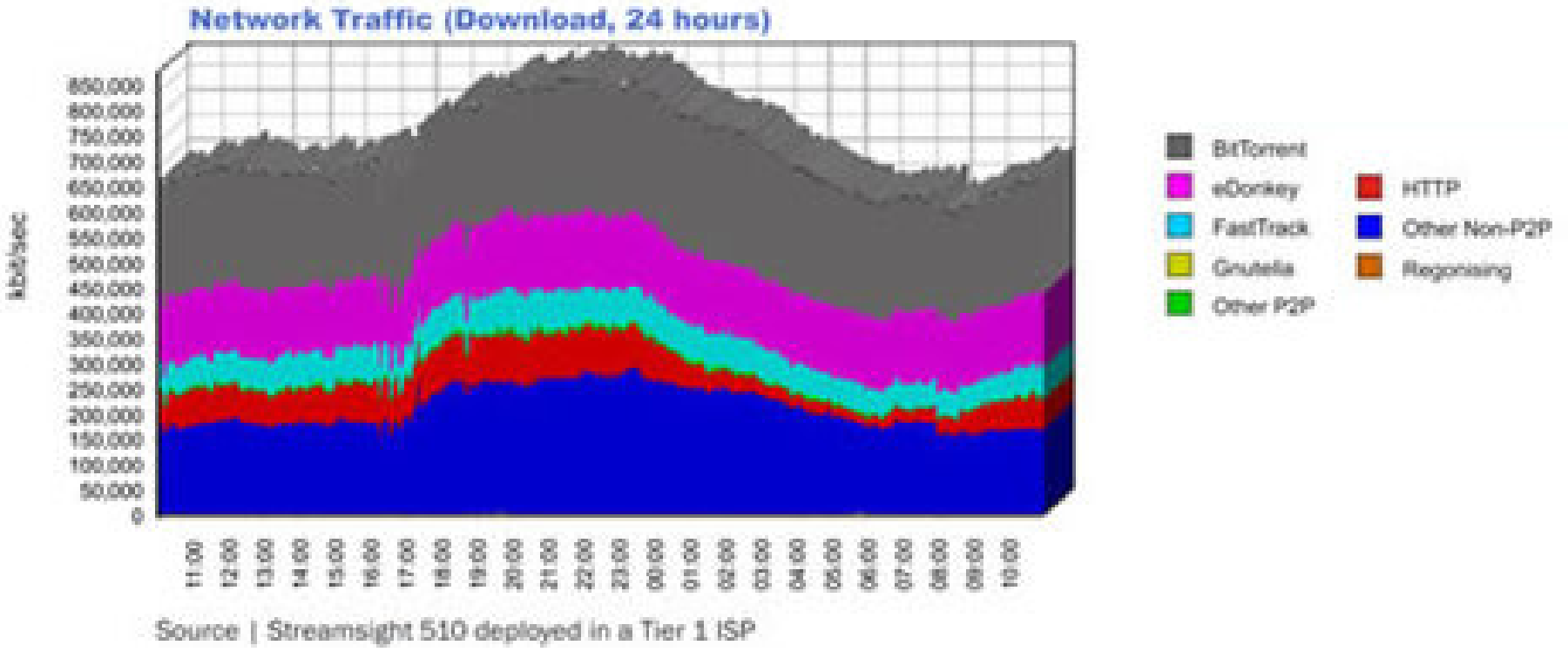
BitTorrent

- Written by Bram Cohen (in Python) in 2001
- “Pull-based” “swarming” approach
  - Each file split into smaller pieces
  - Nodes request desired pieces from neighbors
    - As opposed to parents pushing data that they receive
  - Pieces not downloaded in sequential order
  - Previous multicast schemes aimed to support “streaming”; BitTorrent does not
- Encourages contribution by all nodes

Why is (studying) BitTorrent important?

- BitTorrent consumes significant amount of internet traffic today
  - In 2004, BitTorrent accounted for 30% of all internet traffic (Total P2P was 60%), according to CacheLogic
  - Slightly lower share in 2005 (possibly because of legal action), but still significant
  - BT always used for legal software (linux iso) distribution too
  - Recently: legal media downloads (Fox)

Why is (studying) BitTorrent important?



(From CacheLogic, 2004)

BitTorrent: scenario

- Millions want to download the same popular huge files (for free)
  - ISO's
  - Media (the real example!)
- Client-server model fails
  - Single server fails
  - Can't afford to deploy enough servers

Basic Idea

- Chop file into many pieces
- Replicate different pieces on different peers as soon as possible
- As soon as a peer has a complete piece, it can trade it with other peers
- Hopefully, we will be able to assemble the entire file at the end

Pieces and Sub-Pieces

- A piece is broken into sub-pieces ... typically 16KB in size
- Policy: Until a piece is assembled, only download sub-pieces for that piece
- This policy lets complete pieces assemble quickly

Pipelining

- When transferring data over TCP, it is critical to always have several requests pending at once, to avoid a delay between pieces being sent.
- At any point in time, some number, typically 5, are requested simultaneously.
- Every time a sub-piece arrives, a new request is sent

Terminology

- Seed: peer with the entire file
  - Original Seed: The first seed
- Leech: peer that's downloading the file
  - Fairer term might have been "downloader"
- Sub-piece: Further subdivision of a piece
  - The "unit for requests" is a subpiece
  - But a peer uploads only after assembling complete piece

BitTorrent Swarm

- Swarm
  - Set of peers all downloading the same file
  - Organized as a random mesh
- Each node knows list of pieces downloaded by neighbors
- Node requests pieces it does not own from neighbors
  - Exact method explained later

.torrent file

To share a file or group of files, a peer first creates a **.torrent** file, a small file that contains

(1) metadata about the files to be shared, and

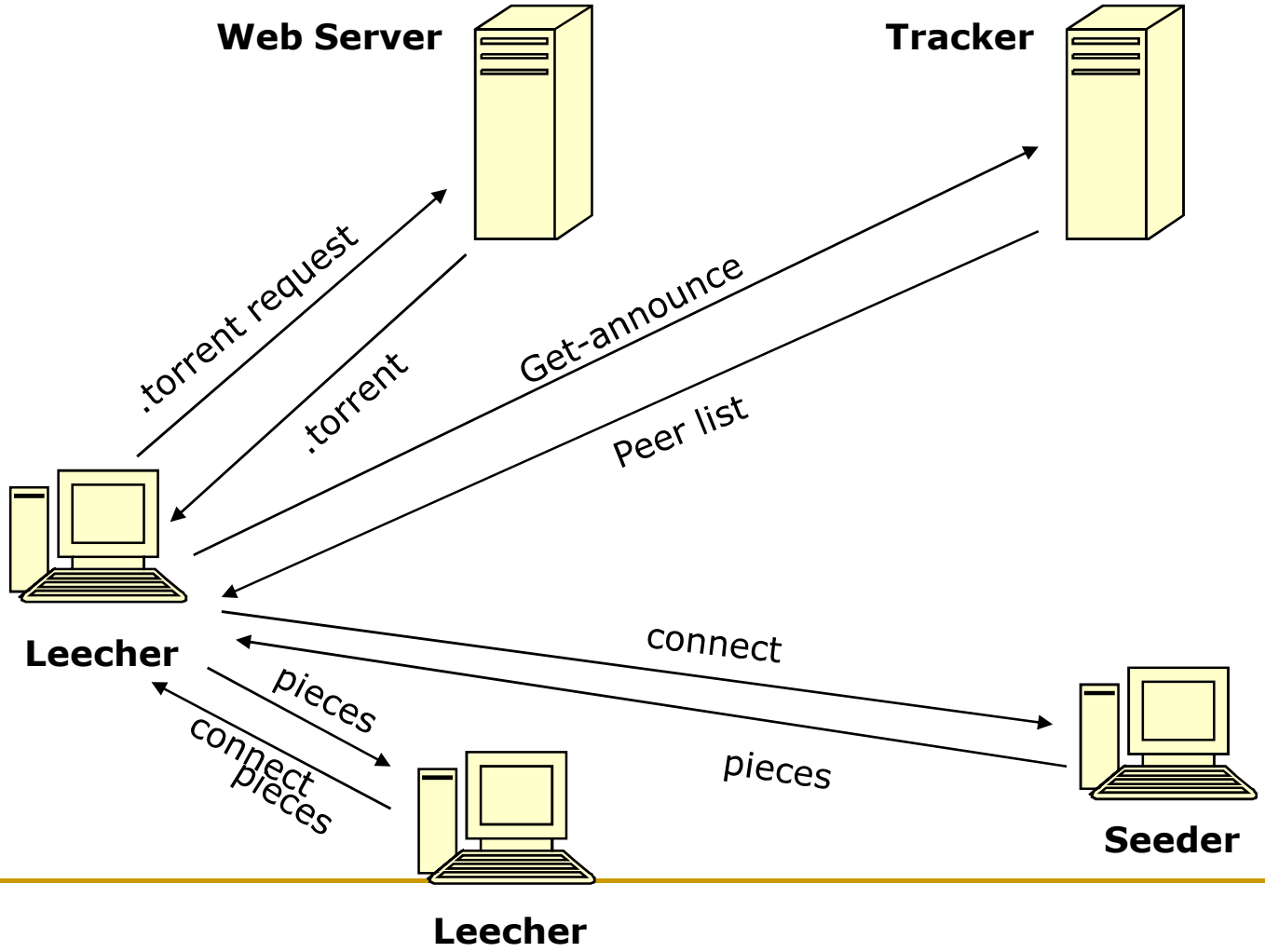
(2) Information about the **tracker**, the computer that coordinates the file distribution.

Peers first obtain a .torrent file for it, and connect to the specified tracker which tells them from which other peers to download the pieces of the file.

Contents of .torrent file

- URL of tracker
- Piece length – Usually 256 KB
- SHA-1 hashes of each piece in file
  - For reliability
- “files” – allows download of multiple files

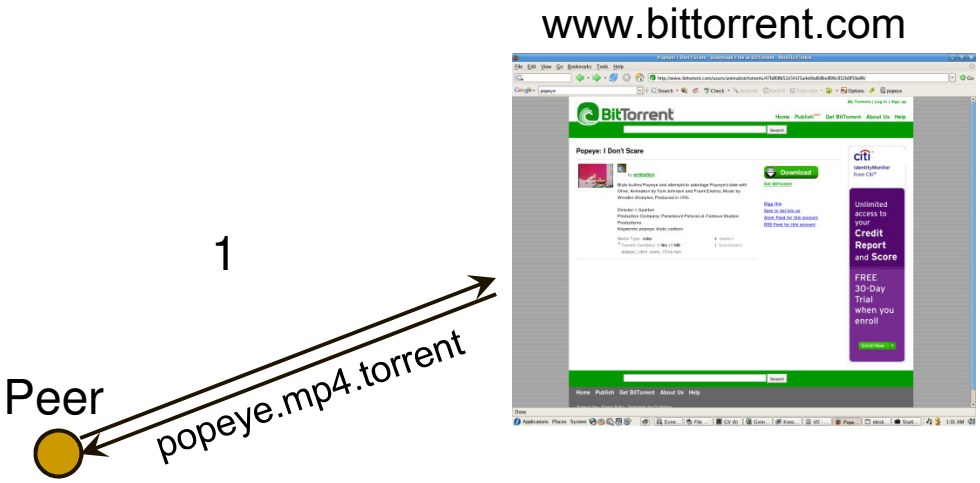
Bittorrent: interactions



How a node enters a swarm for file "popeye.mp4"

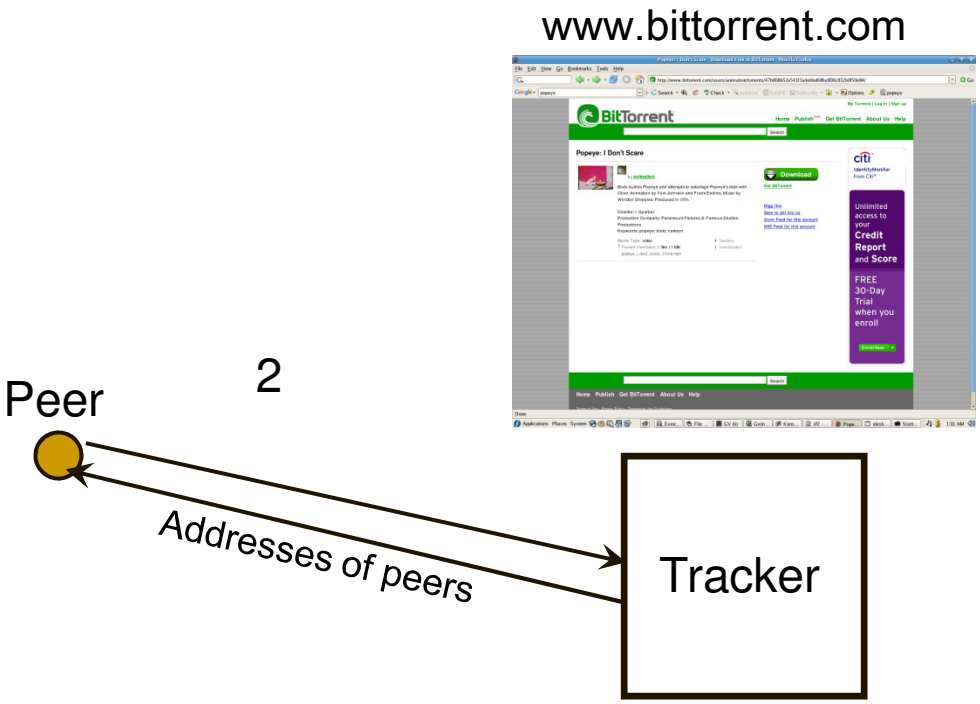
- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of tracker for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

How a node enters a swarm for file "popeye.mp4"



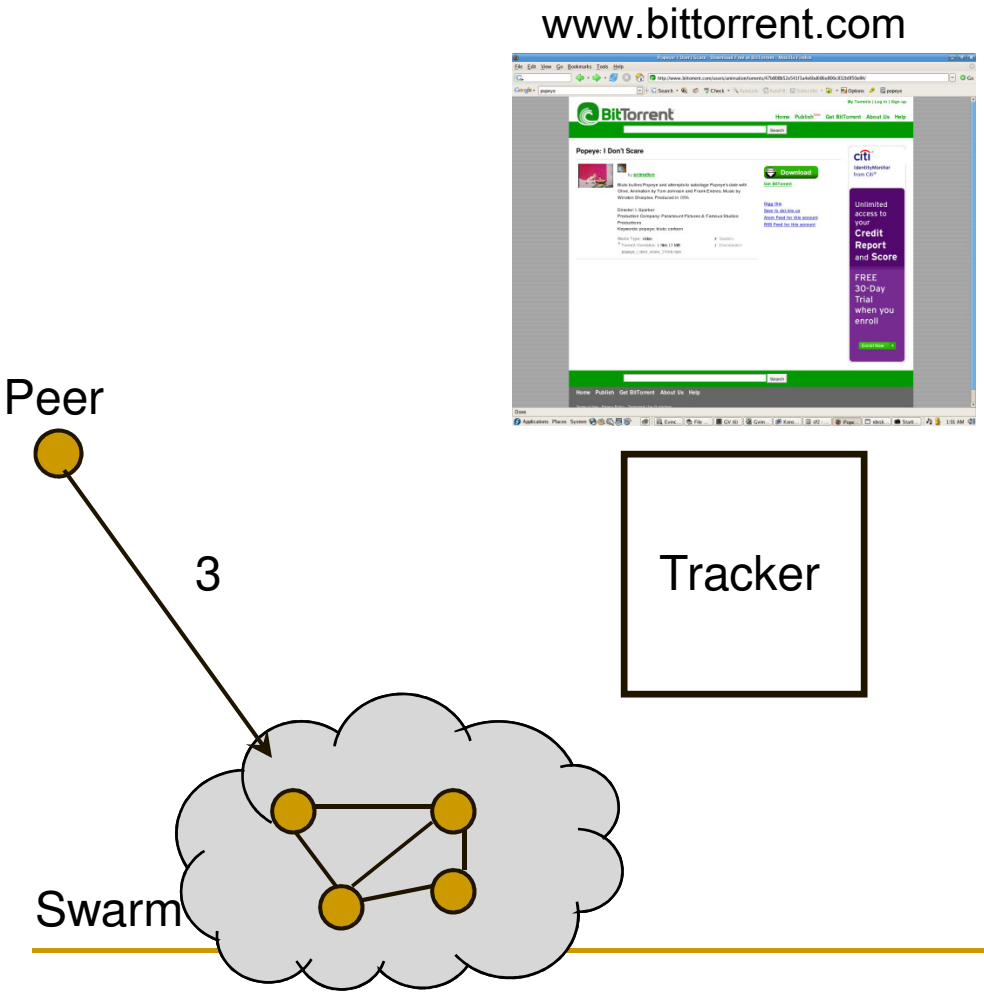
- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of tracker for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

How a node enters a swarm for file "popeye.mp4"



- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of tracker for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

How a node enters a swarm for file "popeye.mp4"



- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of tracker for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

Piece Selection

- The order in which pieces are selected by different peers is critical for good performance
- If a bad algorithm is used, we could end up in a situation where every peer has all the pieces that are currently available and none of the missing ones
- If the original seed is prematurely taken down, then the file cannot be completely downloaded!

Choosing pieces to request

- Random First Piece:
  - When peer starts to download, request random piece
    - Initially, a peer has nothing to trade
    - Important to get a complete piece ASAP
    - Then participate in uploads
  - When first complete piece assembled, switch to rarest-first

Choosing pieces to request

- Rarest-first: Look at all pieces at all peers, and request piece that's owned by fewest peers
  - Increases diversity in the pieces downloaded
    - avoids case where a node and each of its peers have exactly the same pieces; increases throughput
  - Increases likelihood all pieces still available even if original seed leaves before any one node has downloaded entire file

Choosing pieces to request

- End-game mode:
  - When requests sent for all sub-pieces, (re)send requests to all peers.
  - To speed up completion of download
  - Cancel request for downloaded sub-pieces

Endgame Mode

- When all the sub-pieces that a peer doesn't have are actively being requested, these are requested from EVERY peer. When the sub-piece arrives, the replicated requests are cancelled.
- This ensures that a download doesn't get prevented from completion due to a single peer with a slow transfer rate.
- Some bandwidth is wasted, but in practice, this is not too much

Tit-for-tat as incentive to upload

- Want to encourage all peers to contribute
- Peer *A* said to choke peer *B* if it (*A*) decides not to upload to *B*
- Each peer (say *A*) unchokes at most 4 *interested* peers at any time
  - The three with the largest upload rates to *A*
    - Where the tit-for-tat comes in
  - Another randomly chosen (Optimistic Unchoke)
    - To periodically look for better choices

Choking

- **Choking** is a temporary refusal to upload. Downloading occurs as normal
- It is one of BitTorrent's most powerful idea to deal with free riders
- Cooperation involves uploading pieces (that you have) to your peer *(Connection is kept open to avoid setup costs)*
- Based on game-theoretic concepts.
  (Tit-for-tat strategy)

Choking Algorithm

- A good choking algorithm should cap the number of simultaneous uploads for good TCP performance.
- It should avoid choking and unchoking quickly, (known as fibrillation). It should reciprocate to peers who let it download.
- Finally, it should try out unused connections once in a while to find out if they might be better than the currently used ones, known as optimistic unchoking.

Anti-Snubbing

When over a minute goes by without getting a single piece from a particular peer, BitTorrent assumes it is "snubbed" by that peer and doesn't upload to it except as an optimistic unchoke. This frequently results in more than one concurrent optimistic unchoke, which causes download rates to recover much more quickly when they falter.

Anti-snubbing

- A peer is said to be snubbed if each of its peers chokes it
- To handle this, snubbed peer stops uploading to its peers
- Optimistic unchoking done more often
  - Hope is that will discover a new peer that will upload to us

Upload-Only mode

- Once download is complete, a peer has no download rates to use for comparison nor has any need to use them. The question is, which nodes to upload to?
- Policy: Upload to those with the best upload rate.
- This ensures that pieces get replicated faster

Why BitTorrent took off

- Practical Reasons (perhaps more important!)
  - Working implementation (Bram Cohen) with simple well-defined interfaces for plugging in new content
  - Many recent competitors got sued / shut down
    - Napster, Kazaa
  - Doesn't do "search" per se. Users use well-known, trusted sources to locate content
    - Avoids the pollution problem, where garbage is passed off as authentic content

Pros and cons of BitTorrent

- Pros
  - Proficient in utilizing partially downloaded files
  - Discourages “freeloading”
    - By rewarding fastest uploaders
  - Encourages diversity through “rarest-first”
    - Extends lifetime of swarm
- Works well for “hot content”

Pros and cons of BitTorrent

- Cons
  - Assumes all interested peers active at same time; performance deteriorates if swarm “cools off”
  - Even worse: no trackers for obscure content

Pros and cons of BitTorrent

- Dependence on centralized tracker: pro/con?
  - ☹ Single point of failure: New nodes can't enter swarm if tracker goes down
  - Lack of a search feature
    - ☺ Prevents pollution attacks
    - ☹ Users need to resort to out-of-band search: well known torrent-hosting sites / plain old web-search

Free-Riding Problem in P2P Networks

- Vast majority of users are free-riders
  - Most share no files and answer no queries
  - Others limit # of connections or upload speed
- A few “peers” essentially act as servers
  - A few individuals contributing to the public good
  - Making them hubs that basically act as a server
- BitTorrent prevent free riding
  - Allow the fastest peers to download from you
  - Occasionally let some free loaders download

“Trackerless” BitTorrent

- BitTorrent also supports "trackerless" torrents, featuring a DHT implementation that allows the client to download torrents that have been created without using a BitTorrent tracker.
- To be more precise, “BitTorrent without a centralized-tracker”
  - E.g.: Azureus
- Uses a Distributed Hash Table (Kademlia DHT)
- Tracker run by a normal end-host (not a web-server anymore)
  - The original seeder could itself be the tracker
  - Or have a node in the DHT randomly picked to act as the tracker