

**INF5071 – Performance in distributed systems:**



# **Distribution – Part III**

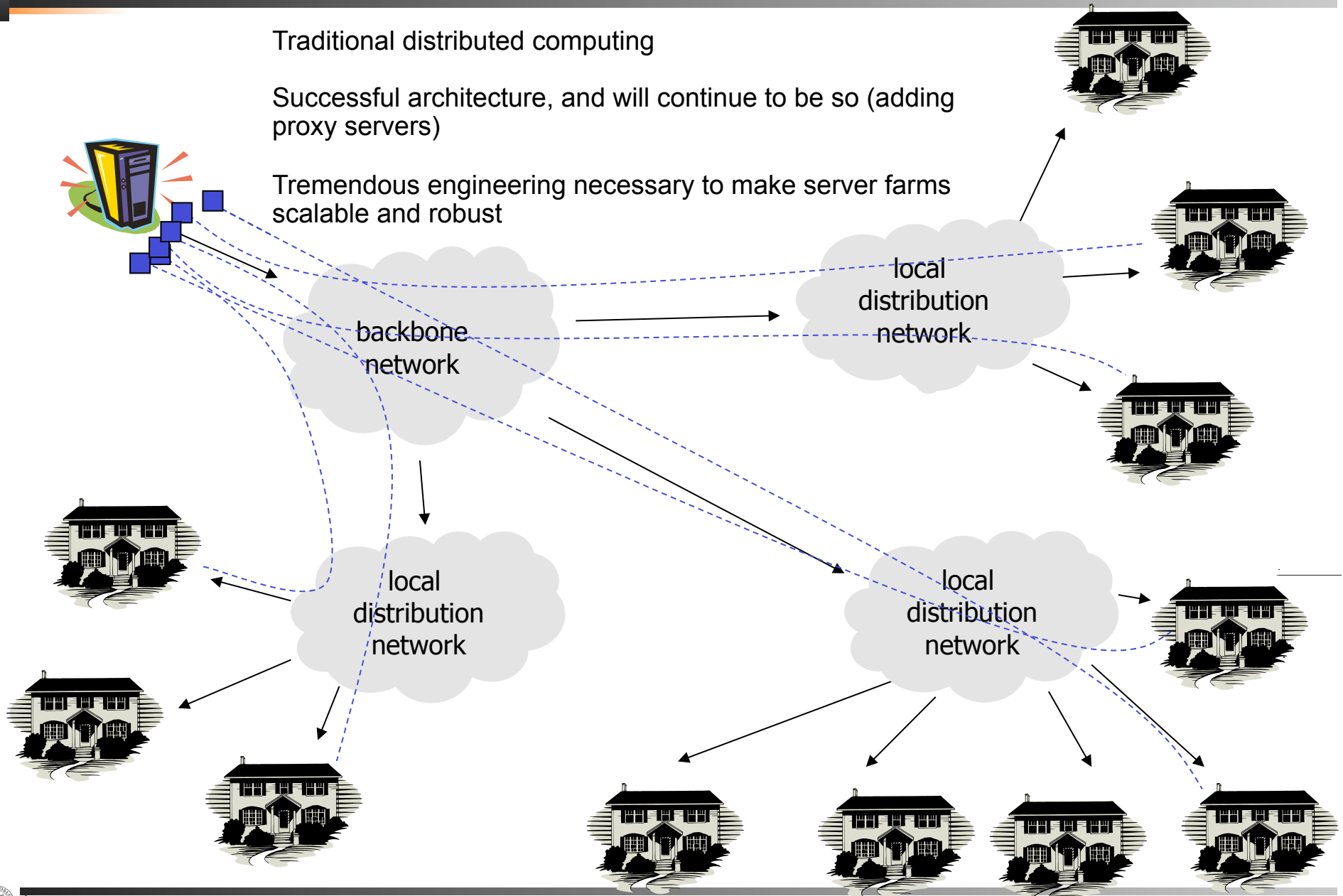
November 11, 09

# Client-Server

Traditional distributed computing

Successful architecture, and will continue to be so (adding proxy servers)

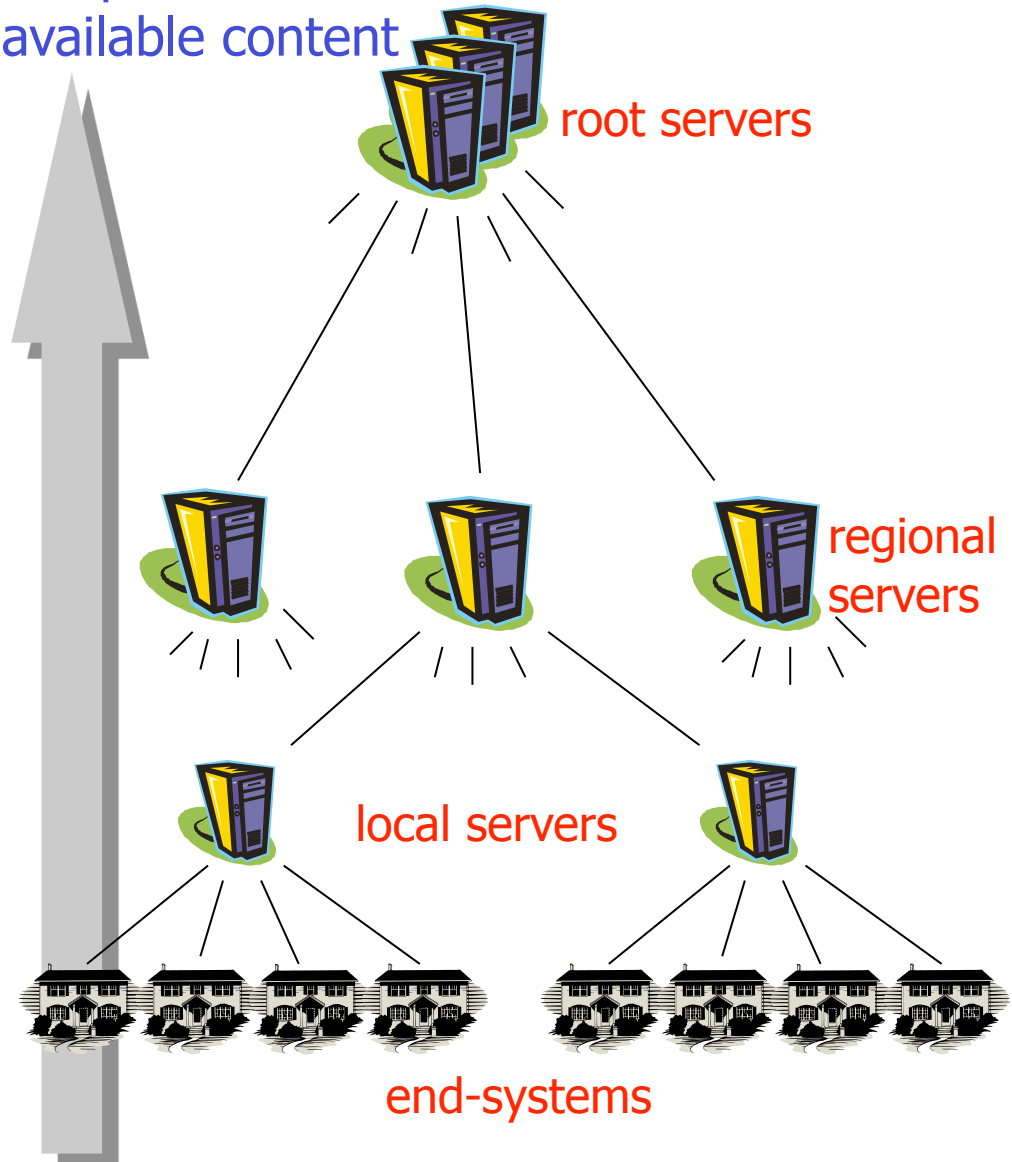
Tremendous engineering necessary to make server farms scalable and robust



# Distribution with proxies

- Hierarchical distribution system
  - E.g. proxy caches that consider popularity
- Popular data replicated more frequently and kept close to clients
- Unpopular ones close to the root servers

completeness of available content

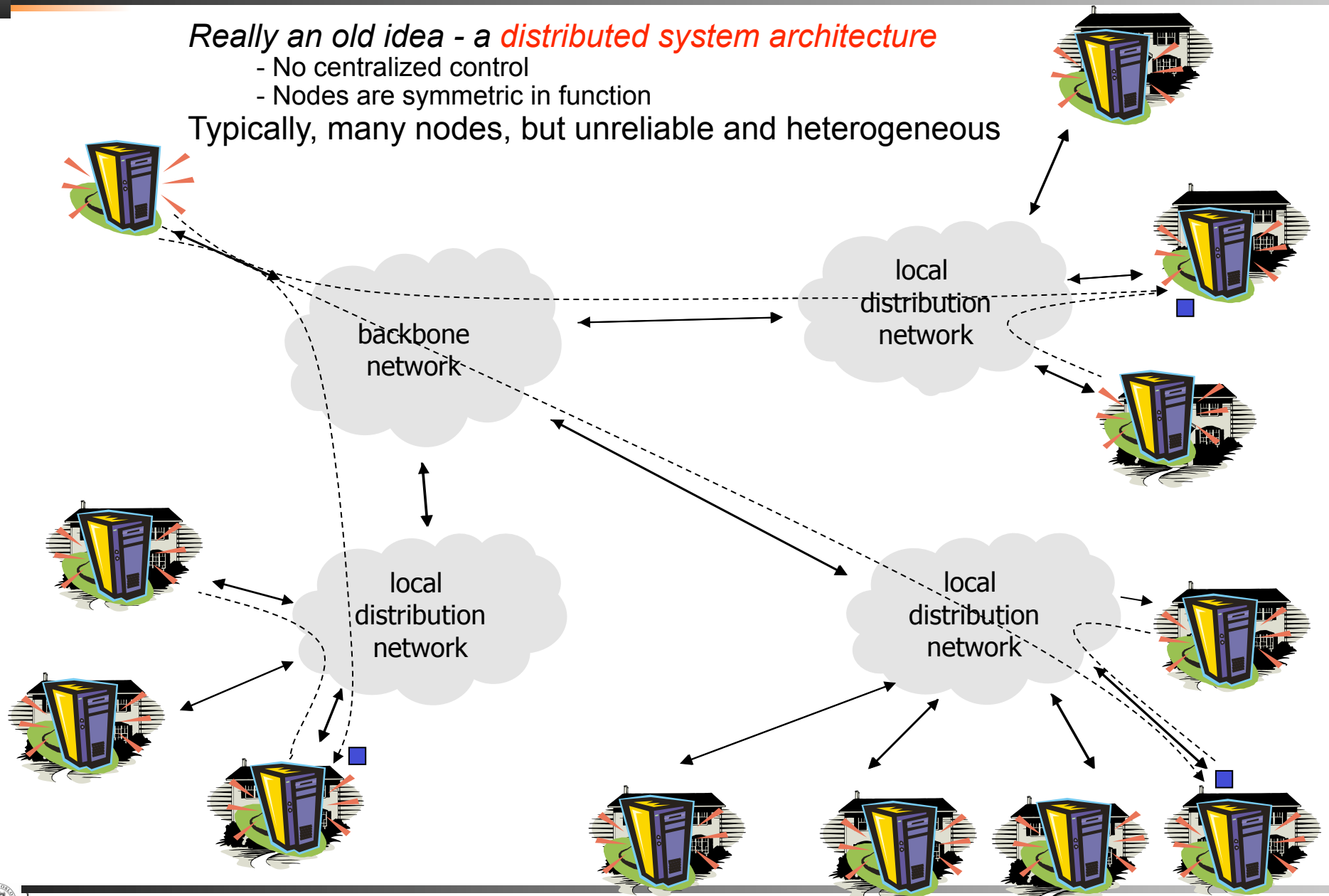


# Peer-to-Peer (P2P)

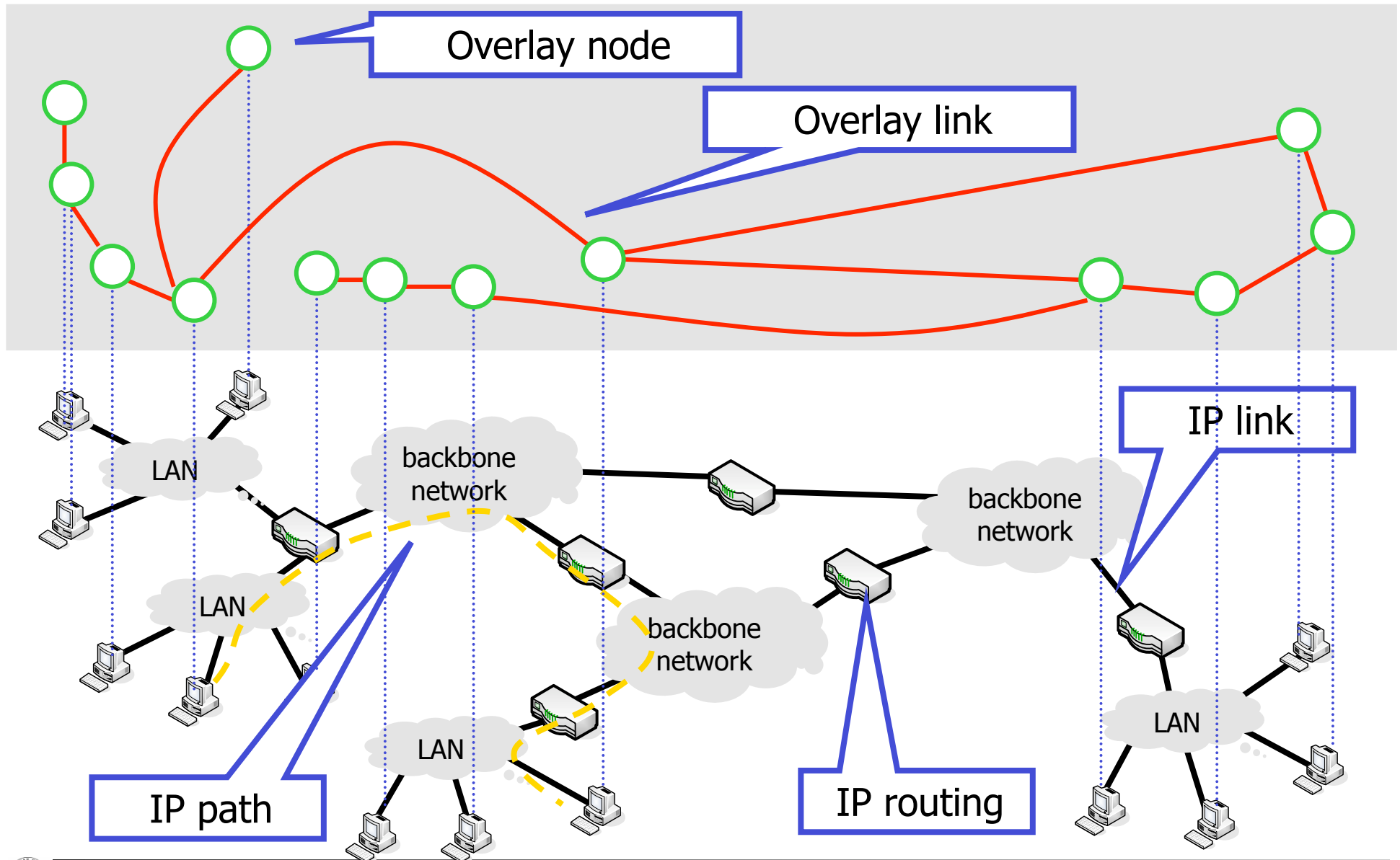
Really an old idea - a *distributed system architecture*

- No centralized control
- Nodes are symmetric in function

Typically, many nodes, but unreliable and heterogeneous



# Overlay networks



# P2P

---

- Many aspects similar to proxy caches
  - Nodes act as clients and servers
  - Distributed storage
  - Bring content closer to clients
  - Storage limitation of each node
  - Number of copies often related to content popularity
  - Necessary to make replication and de-replication decisions
  - Redirection
  
- But
  - No distinguished roles
  - No generic hierarchical relationship
    - At most hierarchy per data item
  - Clients do not know where the content is
    - May need a *discovery protocol*
  - All clients may act as roots (origin servers)
  - Members of the P2P network come and go (churn)

# P2P Systems

---

- Peer-to-peer systems
  - New considerations for distribution systems
- Considered here
  - Scalability, fairness, load balancing
  - Content location
  - Failure resilience
  - Routing
    - Application layer routing
    - Content routing
    - Request routing
- Not considered here
  - Copyright
  - Privacy
  - Trading





Examples: Napster



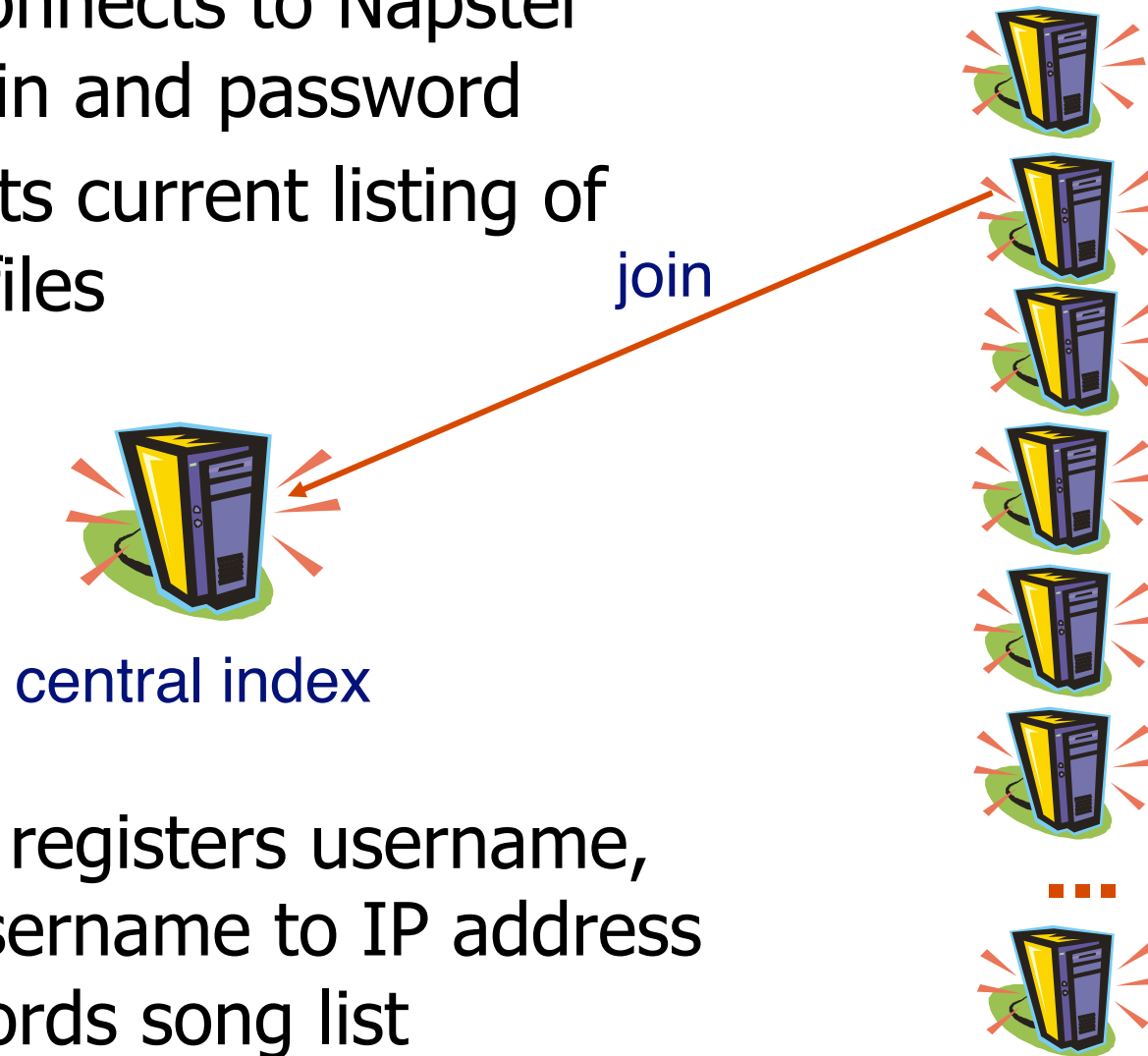
# Napster

---

- Program for sharing (music) files over the Internet
- Approach taken
  - Central index
  - Distributed storage and download
  - All downloads are shared
- P2P aspects
  - Client nodes act also as file servers
- Shut down July 2001, but Roxio Inc. has rebrand the pressplay music service as Napster 2.0, sold for \$121M 2008, Napster 4 available in UK...

# Napster

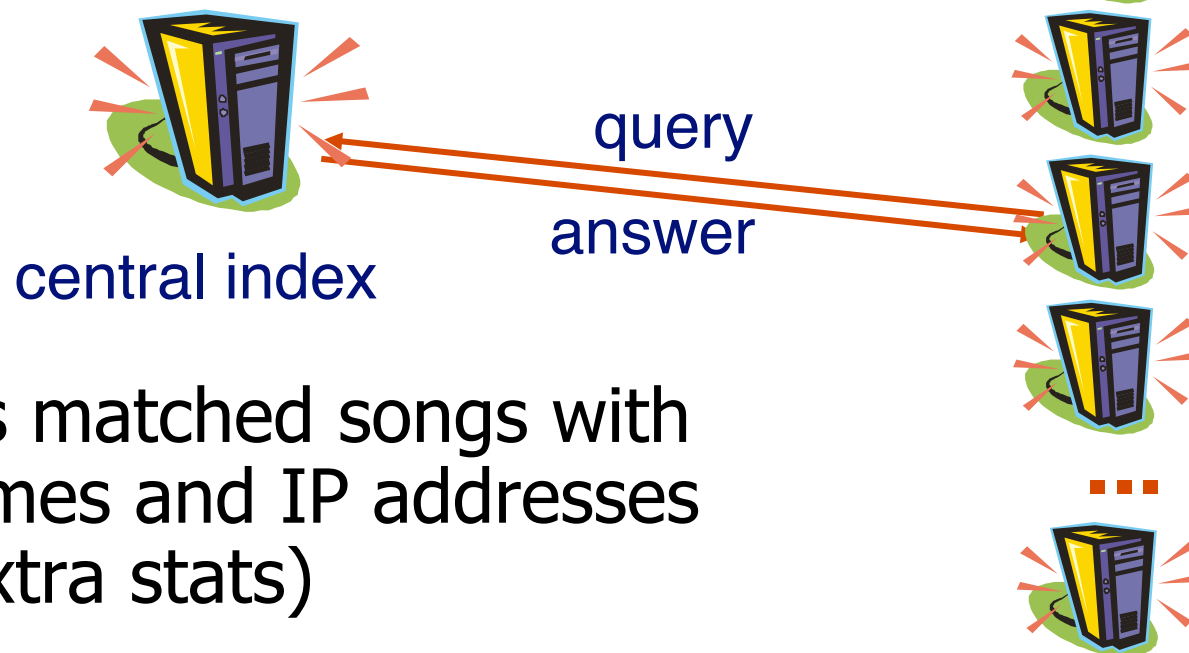
- Client connects to Napster with login and password
- Transmits current listing of shared files



- Napster registers username, maps username to IP address and records song list

# Napster

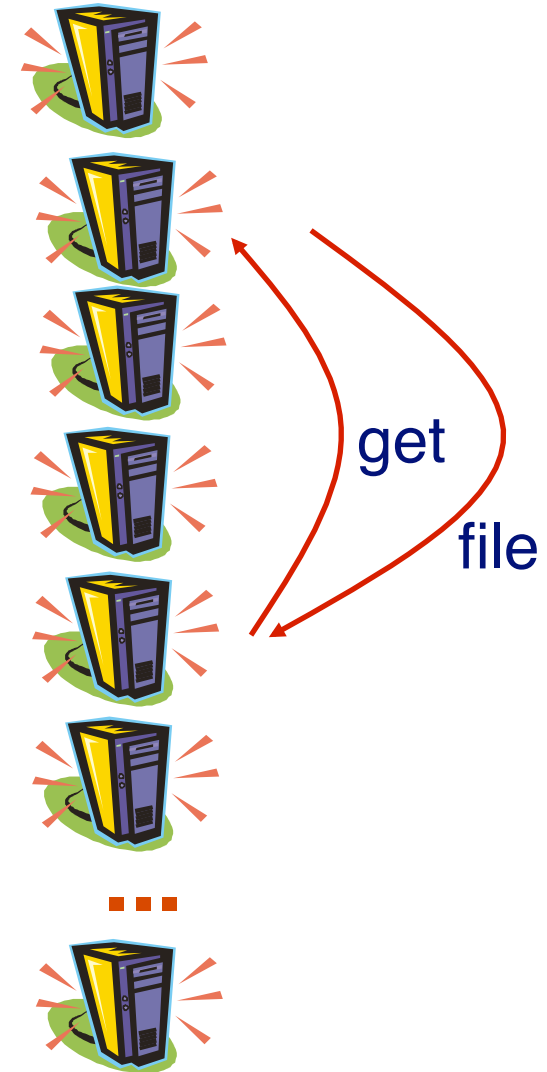
- Client sends song request to Napster server
- Napster checks song database



- Returns matched songs with usernames and IP addresses (plus extra stats)

# Napster

- User selects a song, download request sent straight to user
- Machine contacted if available



# Napster: Assessment

---

- Scalability, fairness, load balancing
  - Large distributed storage
  - Replication to querying nodes
    - Number of copies increases with popularity
  - Unavailability of files with low popularity
  - Network topology is not accounted for at all, i.e., latency may be increased and bandwidth may be low
- Content location
  - Simple, centralized search/location mechanism
  - Can only query by index terms
- Failure resilience
  - No dependencies among normal peers
  - Index server as single point of failure





Examples: Gnutella

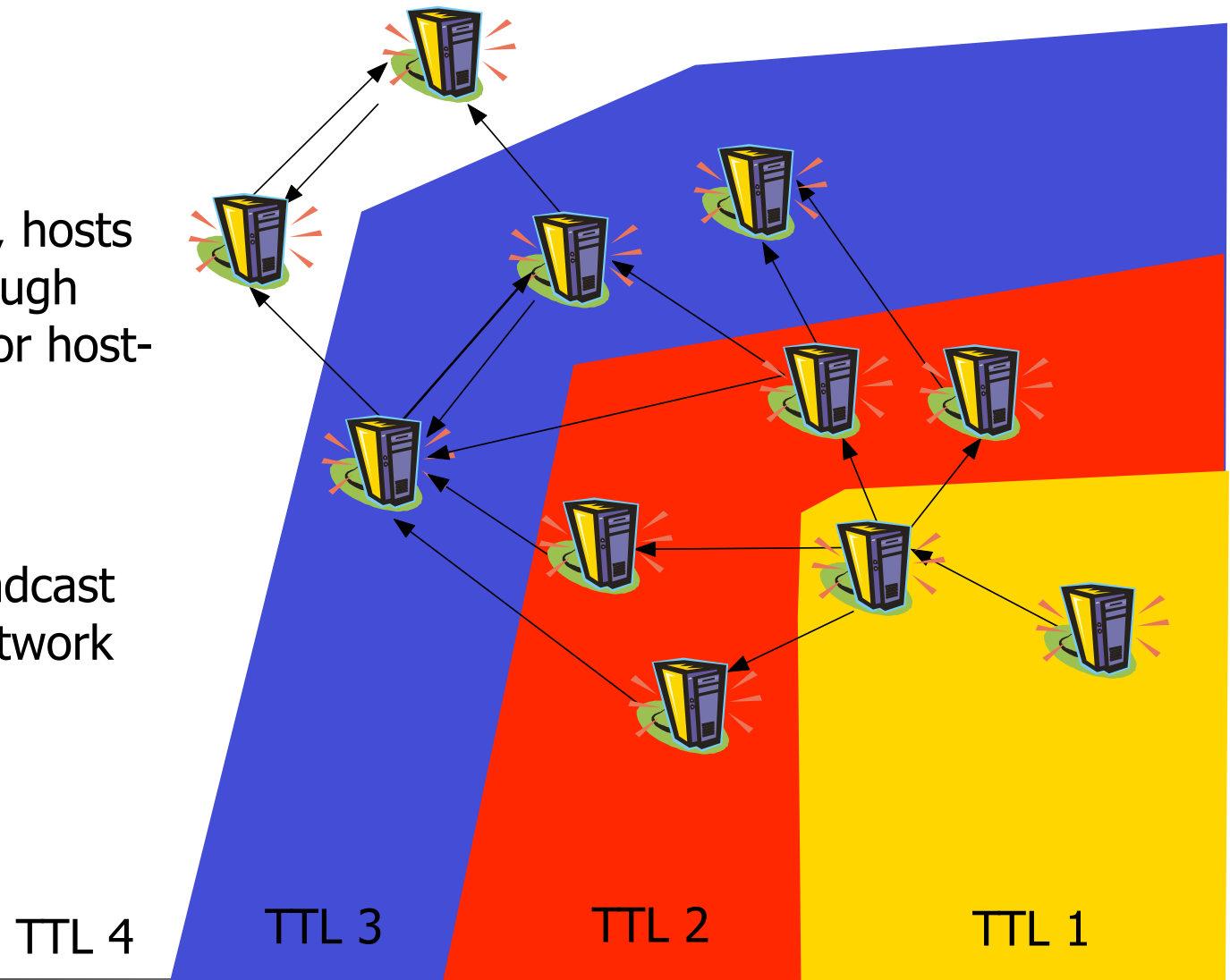
# Gnutella

- Program for sharing files over the Internet
- Approach taken
  - Pure P2P system, centralized nothing (each peer typically connect to 5 other peers)
  - Dynamically built overlay network
  - Query for content by overlay broadcast
  - No index maintenance
- P2P aspects
  - Peer-to-peer file sharing
  - Peer-to-peer querying
  - Entirely decentralized architecture
- Many iterations to fix poor initial design (lack of scalability)
  - introduction of leaves and ultrapeers to find shorter paths
  - each leaf connected to few (typically 3) ultrapeers which again connected to many (typically 32) other ultrapeers
  - ...



# Gnutella: Joining

- Connect to one known host and send a *broadcast ping*
  - Can be any host, hosts transmitted through word-of-mouth or host-caches
  - Use overlay broadcast ping through network with TTL of 7 (later 4)

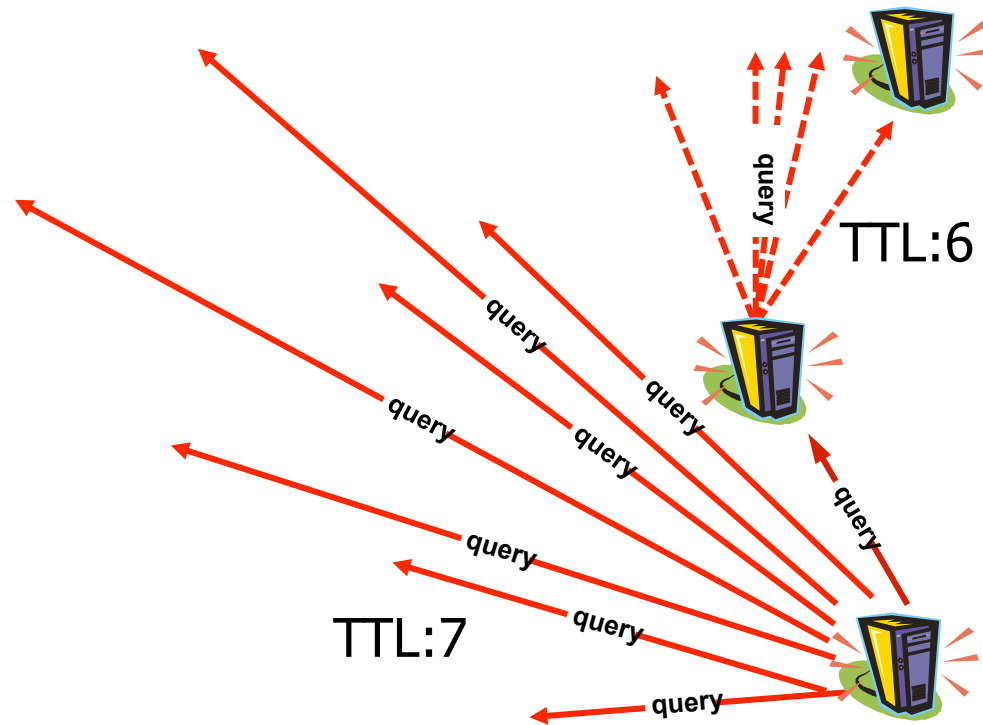






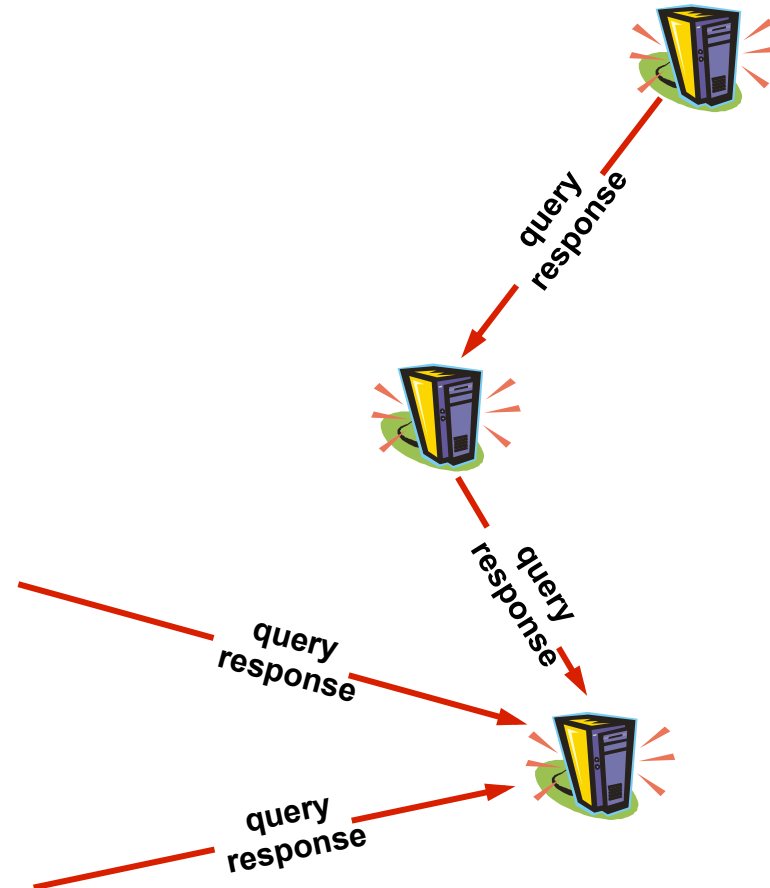
# Gnutella: Query

- Query by broadcasting in the overlay
  - Send query to all overlay neighbors
  - Overlay neighbors forward query to all their neighbors
  - Up to 7 layers deep (TTL 7), and later 4



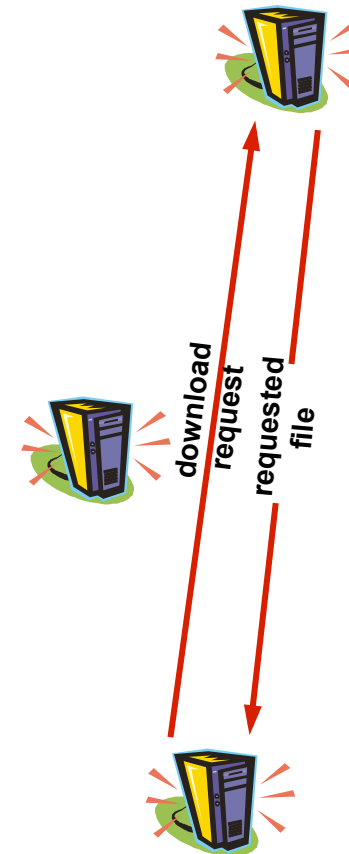
# Gnutella: Query

- Send routed responses
  - To the overlay node that was the source of the broadcast query
  - Querying client receives several responses
  - User receives a list of files that matched the query and a corresponding IP address
- Later revised to directly deliver responses



# Gnutella: Transfer

- File transfer
  - Using direct communication
  - File transfer protocol not part of the Gnutella specification



# Gnutella: Assessment

---

- Scalability, fairness, load balancing
  - Replication to querying nodes
    - Number of copies increases with popularity
  - Large distributed storage
  - Unavailability of files with low popularity
  - Bad scalability, uses flooding approach
  - Network topology is not accounted for at all, latency may be increased
- Content location
  - No limits to query formulation
  - Less popular files may be outside TTL
- Failure resilience
  - No single point of failure
  - Many known neighbors
  - Assumes quite stable relationships





# Examples: Freenet

# Freenet

---

- Program for sharing files over the Internet
  - Focus on anonymity
- Approach taken
  - Purely P2P, centralized nothing
  - Dynamically built overlay network
  - Query for content by hashed query and best-first-search
  - Caching of hash values and content along the route
  - *Content forwarding in the overlay*
- P2P aspects
  - Peer-to-peer file sharing
  - Peer-to-peer querying
  - Entirely decentralized architecture
  - Anonymity



# Freenet: Nodes and Data

---

- Nodes
  - Routing tables - contain IP addresses of other nodes and the hash values they hold (resp. held)
- Data is indexed with a hash values
  - “Identifiers” are hashed
  - Identifiers may be keywords, author ids, or the content itself
  - Secure Hash Algorithm (SHA-1) produces a “one-way” 160-bit key
  - Content-hash key (CHK) =  $\text{SHA-1}(\text{content})$



# Freenet: Storing and Retrieving Data

## ■ Storing Data

- Data is moved to a node with arithmetically close keys
  1. The key and data are sent to the local node
  2. The key and data is forwarded to the node with the nearest keyRepeat 2 until maximum number of hops is reached

## ■ Retrieving data

### — Best First Search

1. An identifier is hashed into a key
  2. The key is sent to the local node
  3. If data is not in local store, the request is forwarded to the best neighbor
- Repeat 3 with next best neighbor until data found, or request times out
4. If data is found, or hop-count reaches zero, return the data or error along the chain of nodes (if data found, intermediary nodes create entries in their routing tables)

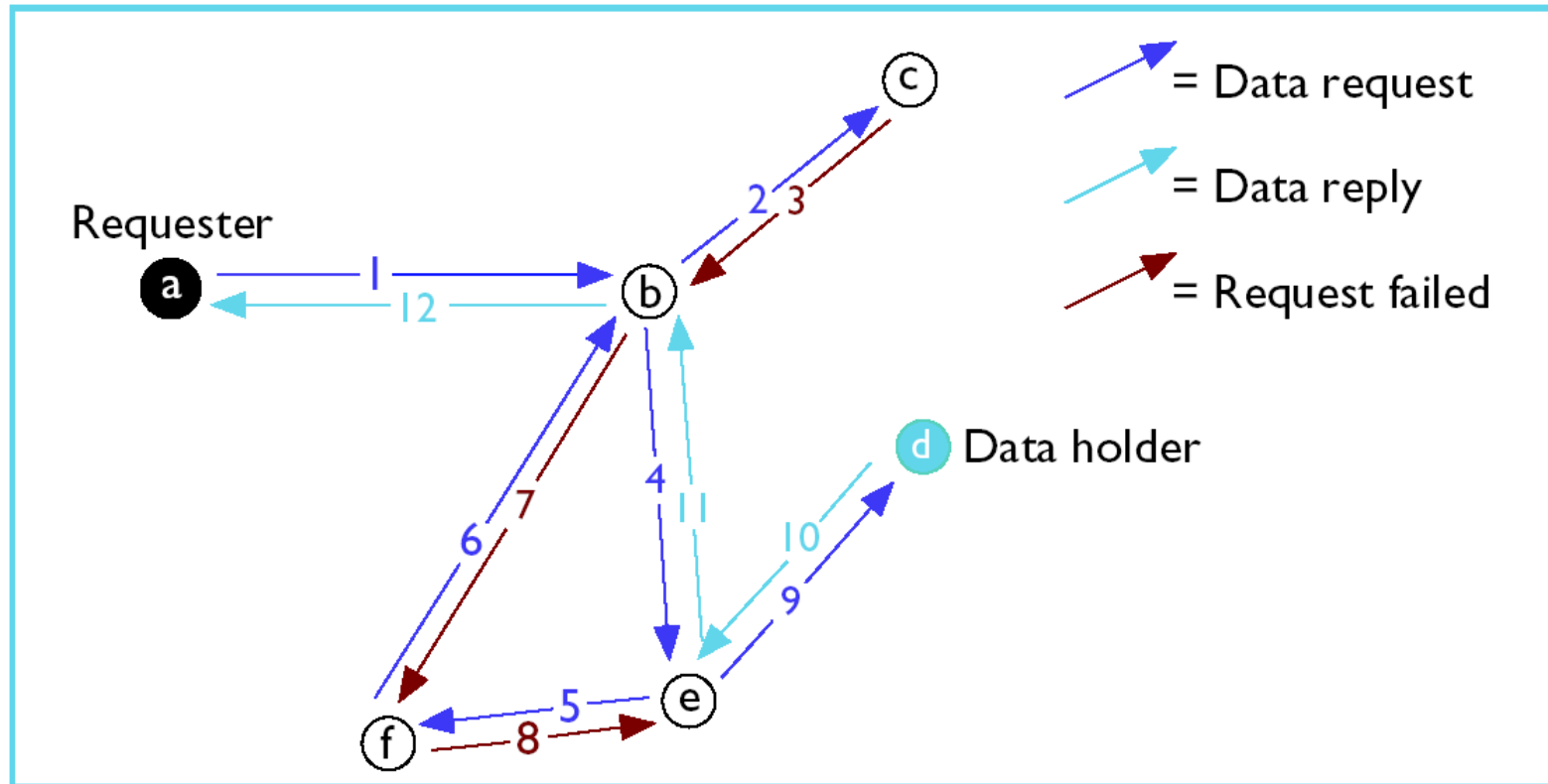


# Freenet: Best First Search



- Heuristics for Selecting Direction
  - > **RES**: Returned most results
  - < **TIME**: Shortest satisfaction time
  - < **HOPS**: Min hops for results
  - > **MSG**: Sent us most messages (all types)
  - < **QLEN**: Shortest queue
  - < **LAT**: Shortest latency
  - > **DEG**: Highest degree

# Freenet: Routing Algorithm



# Freenet: Assessment

---

- Scalability, fairness, load balancing
  - Caching in the overlay network
    - Access latency decreases with popularity
  - Large distributed storage
  - Fast removal of files with low popularity
    - A lot of storage wasted on highly popular files
  - Network topology is not accounted for
- Content location
  - Search by hash key: limited ways to formulate queries
  - Content placement changes to fit search pattern
  - Less popular files may be outside TTL
- Failure resilience
  - No single point of failure





Examples: FastTrack,  
Morpheus, OpenFT

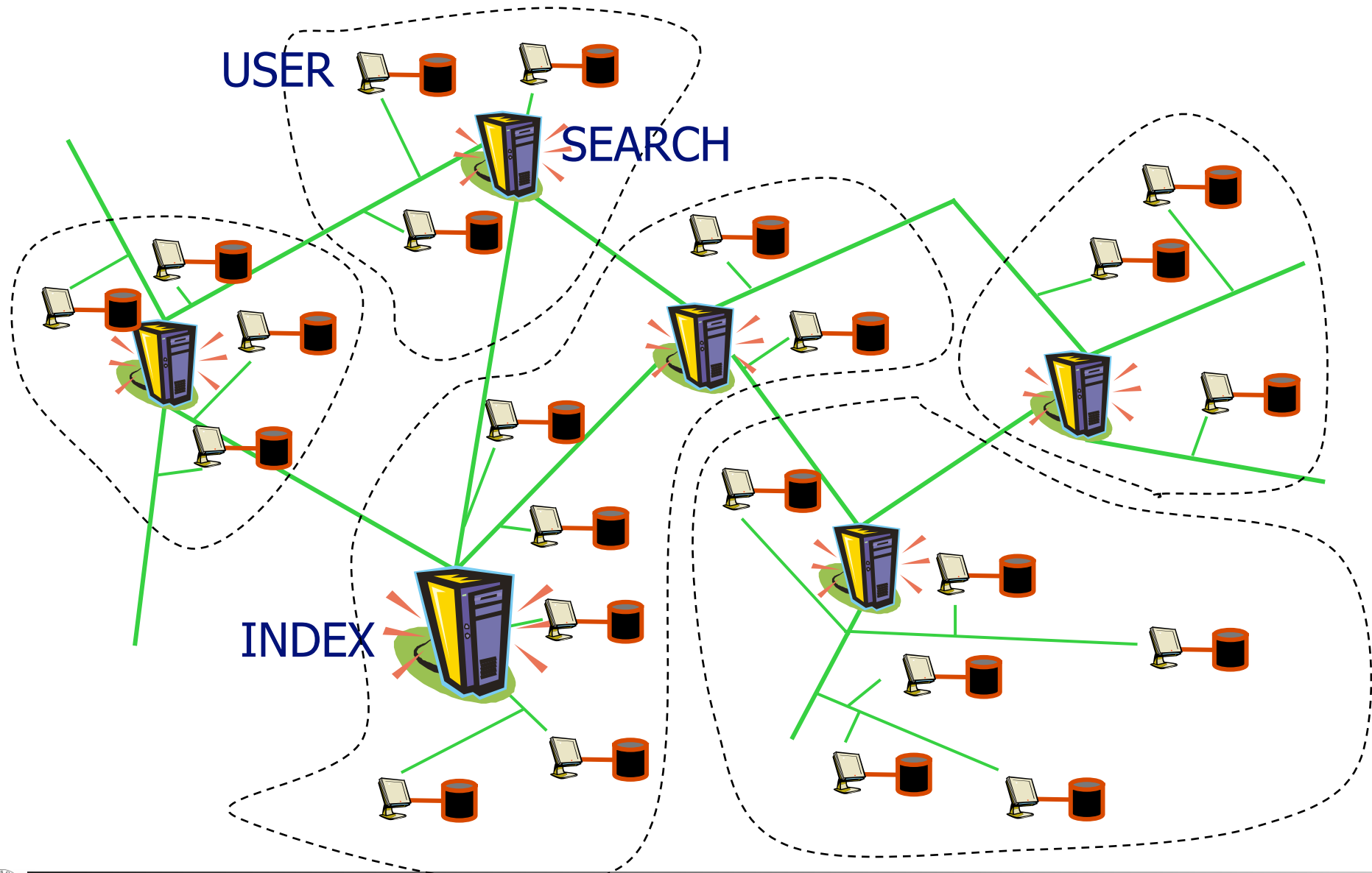
# FastTrack, Morpheus, OpenFT

---

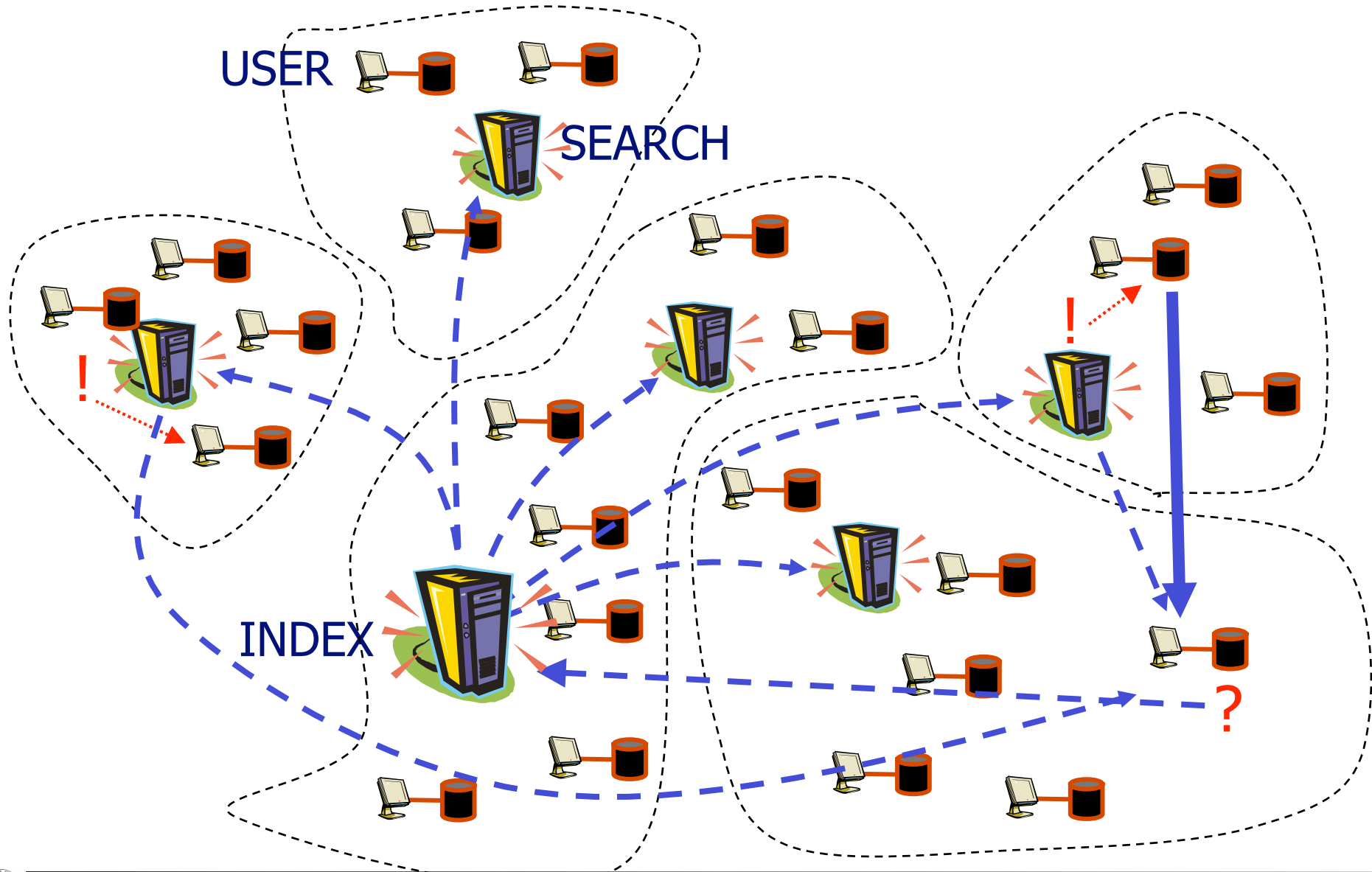
- Peer-to-peer file sharing protocol
- Three different nodes
  - USER
    - Normal nodes
  - SEARCH
    - Keep an index of “their” normal nodes
    - Answer search requests
  - INDEX
    - Keep an index of search nodes
    - Redistribute search requests



# FastTrack, Morpheus, OpenFT



# FastTrack, Morpheus, OpenFT





# FastTrack, Morpheus, OpenFT: Assessment

- Scalability, fairness, load balancing
  - Large distributed storage
  - Avoids broadcasts
  - Load concentrated on super nodes (index and search)
  - Network topology is partially accounted for
  - Efficient structure development
- Content location
  - Search by hash key: limited ways to formulate queries
  - All indexed files are reachable
  - Can only query by index terms
- Failure resilience
  - No single point of failure but overlay networks of index servers (and search servers) reduces resilience
  - Relies on very stable relationship / Content is registered at search nodes
  - Relies on a partially static infrastructure



Examples: BitTorrent

# BitTorrent

---

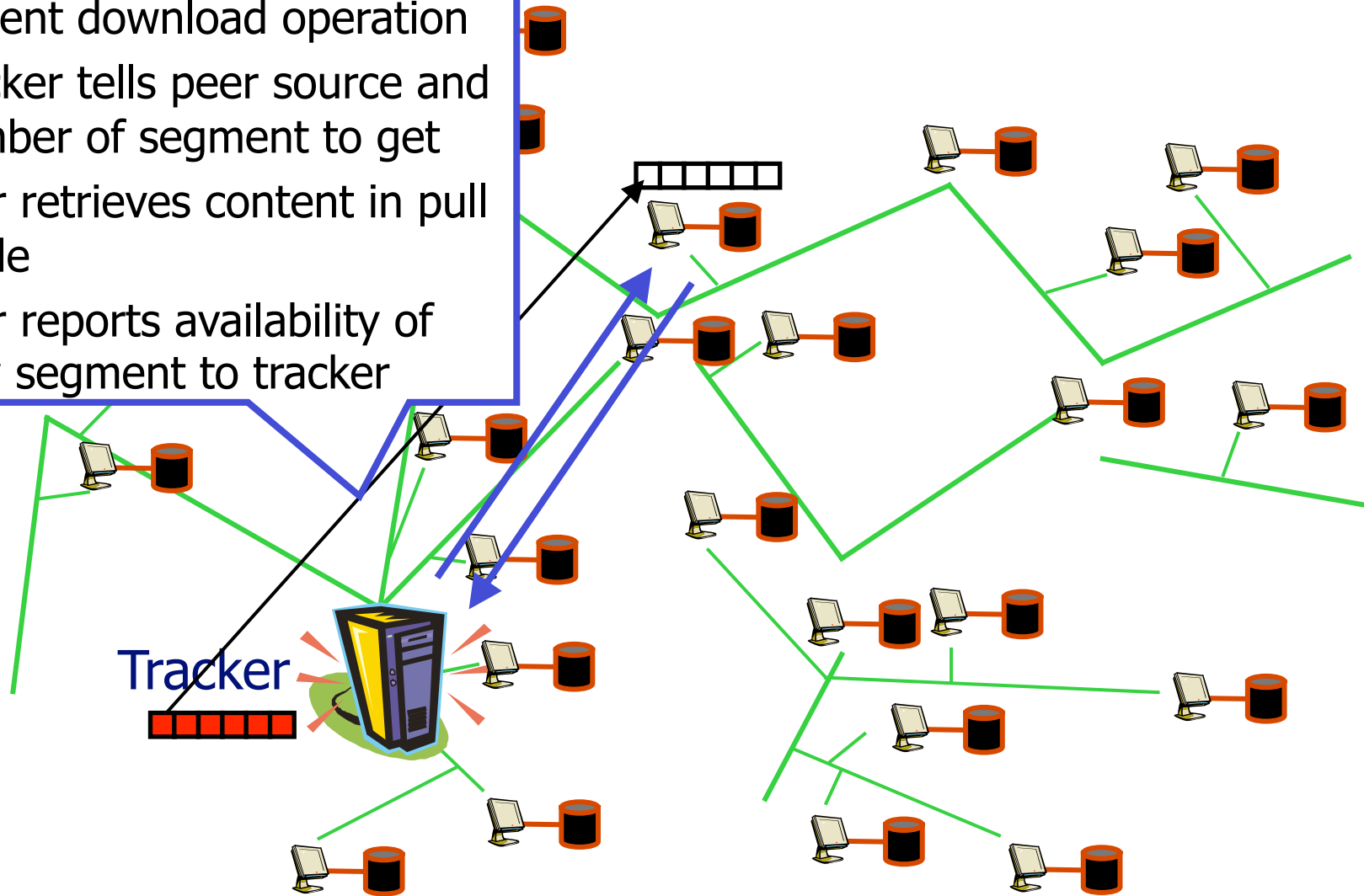
- Distributed download system
- Content is distributed in segments
- Tracker
  - One central download server per content
  - Approach to fairness (tit-for-tat) per content
  - No approach for finding the tracker
- No content transfer protocol included



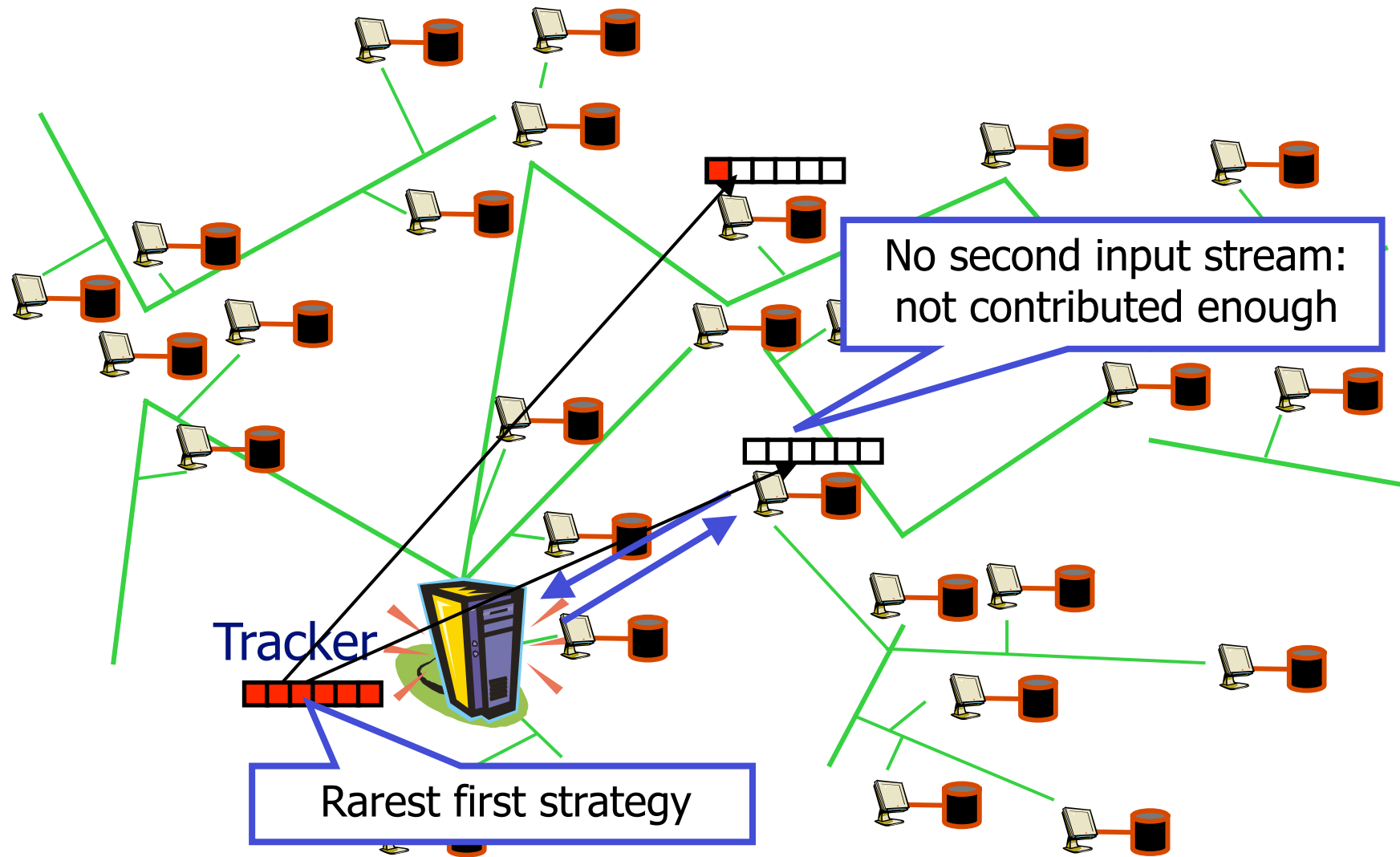
# BitTorrent

## Segment download operation

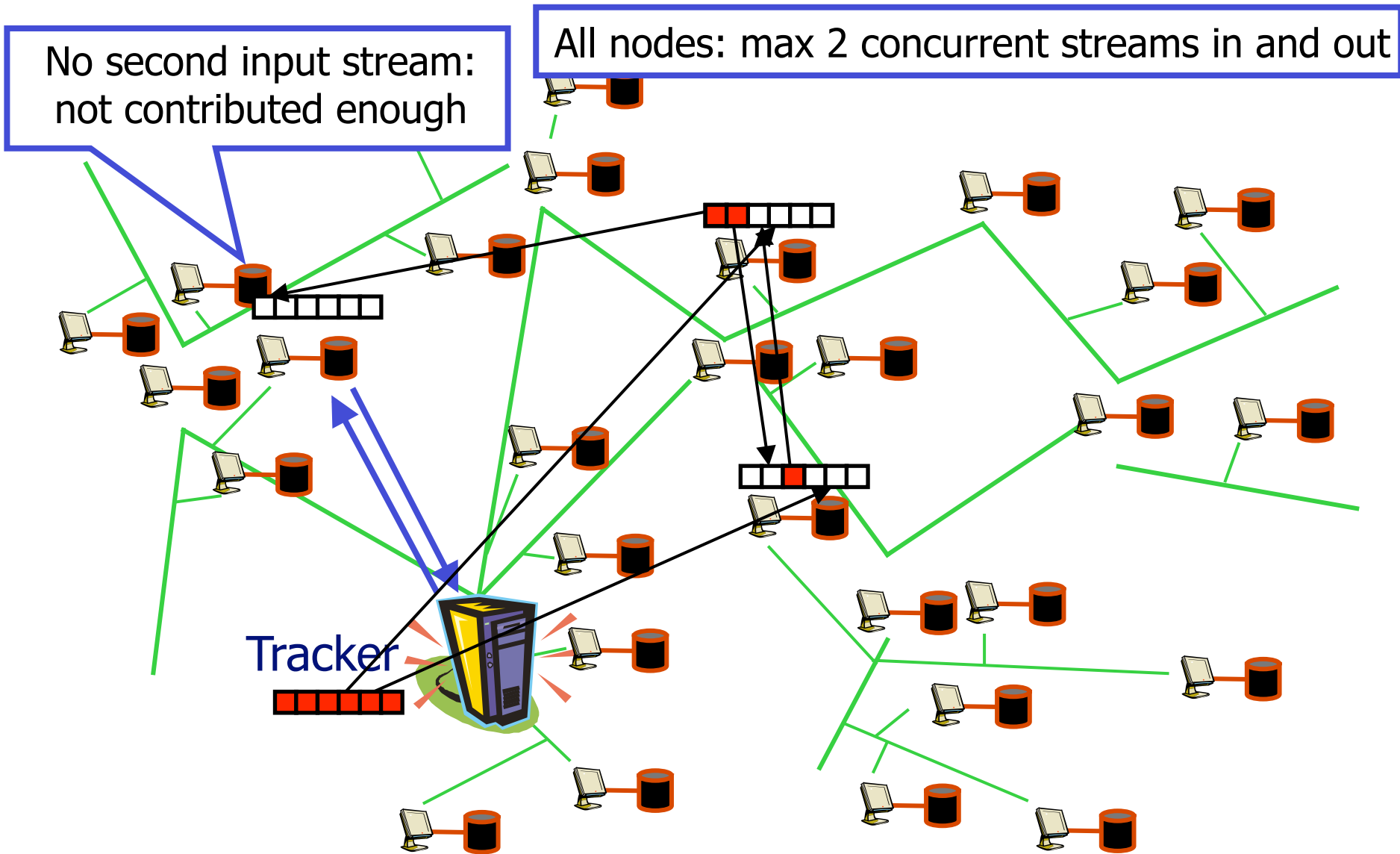
- Tracker tells peer source and number of segment to get
- Peer retrieves content in pull mode
- Peer reports availability of new segment to tracker



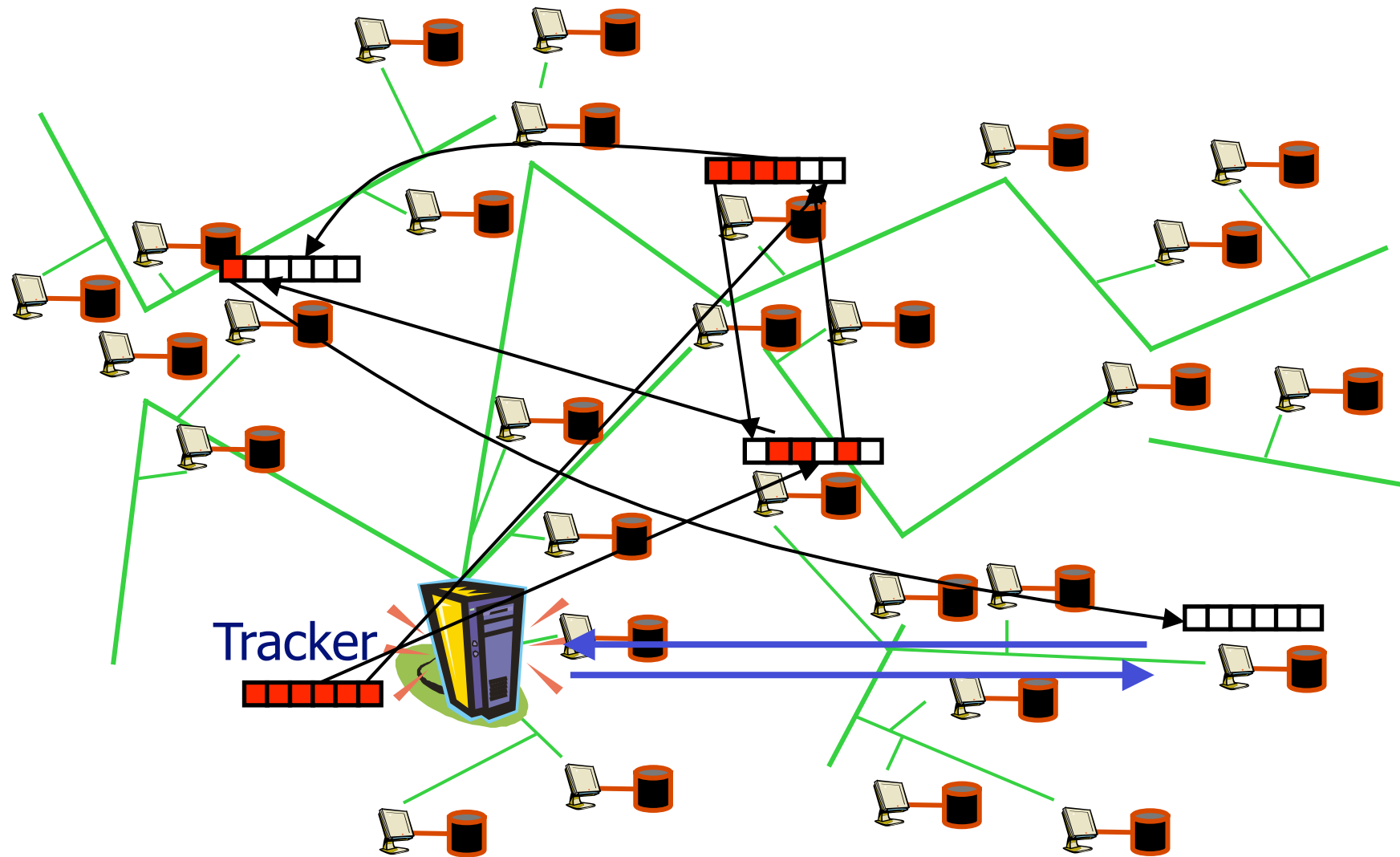
# BitTorrent



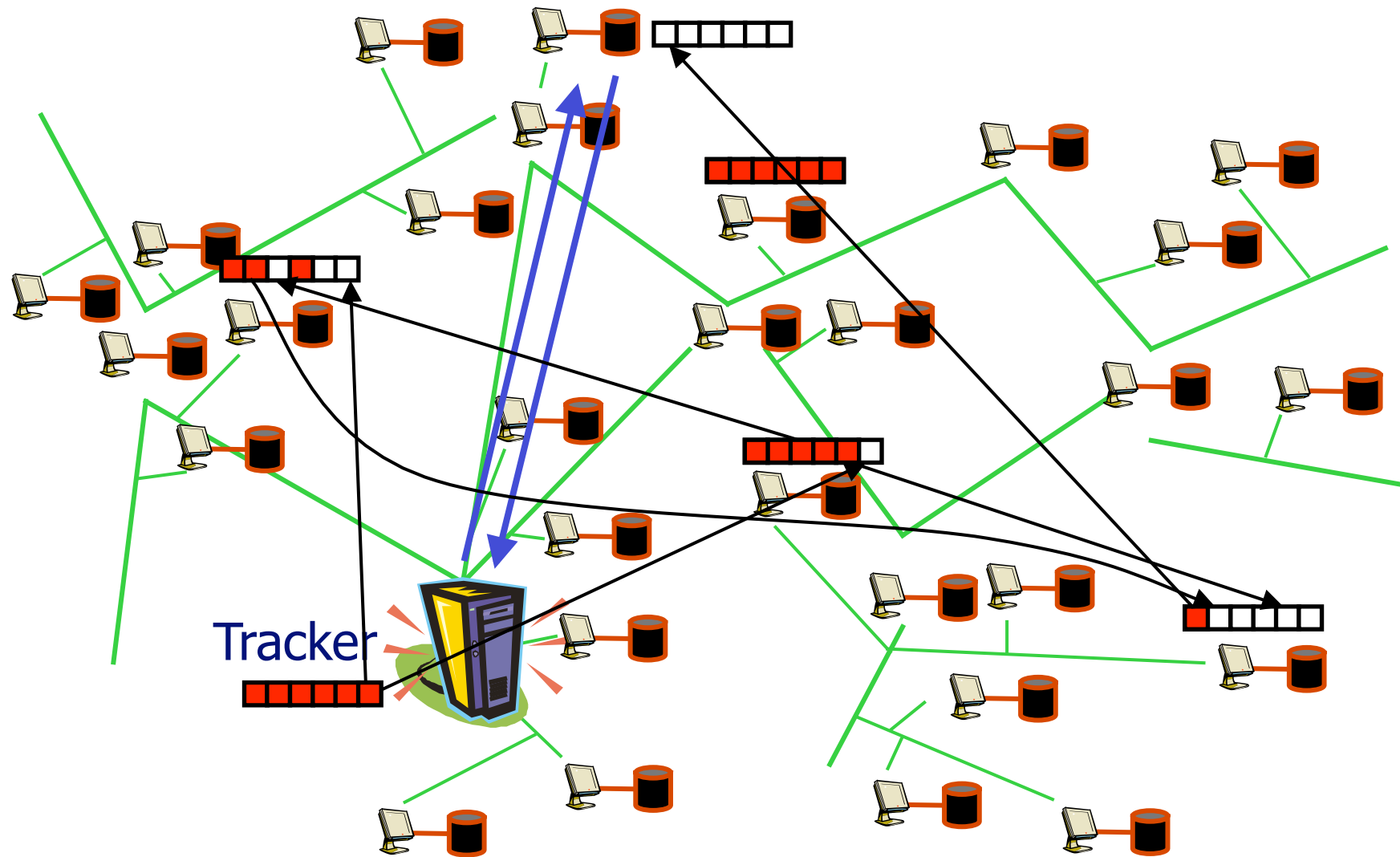
# BitTorrent



# BitTorrent

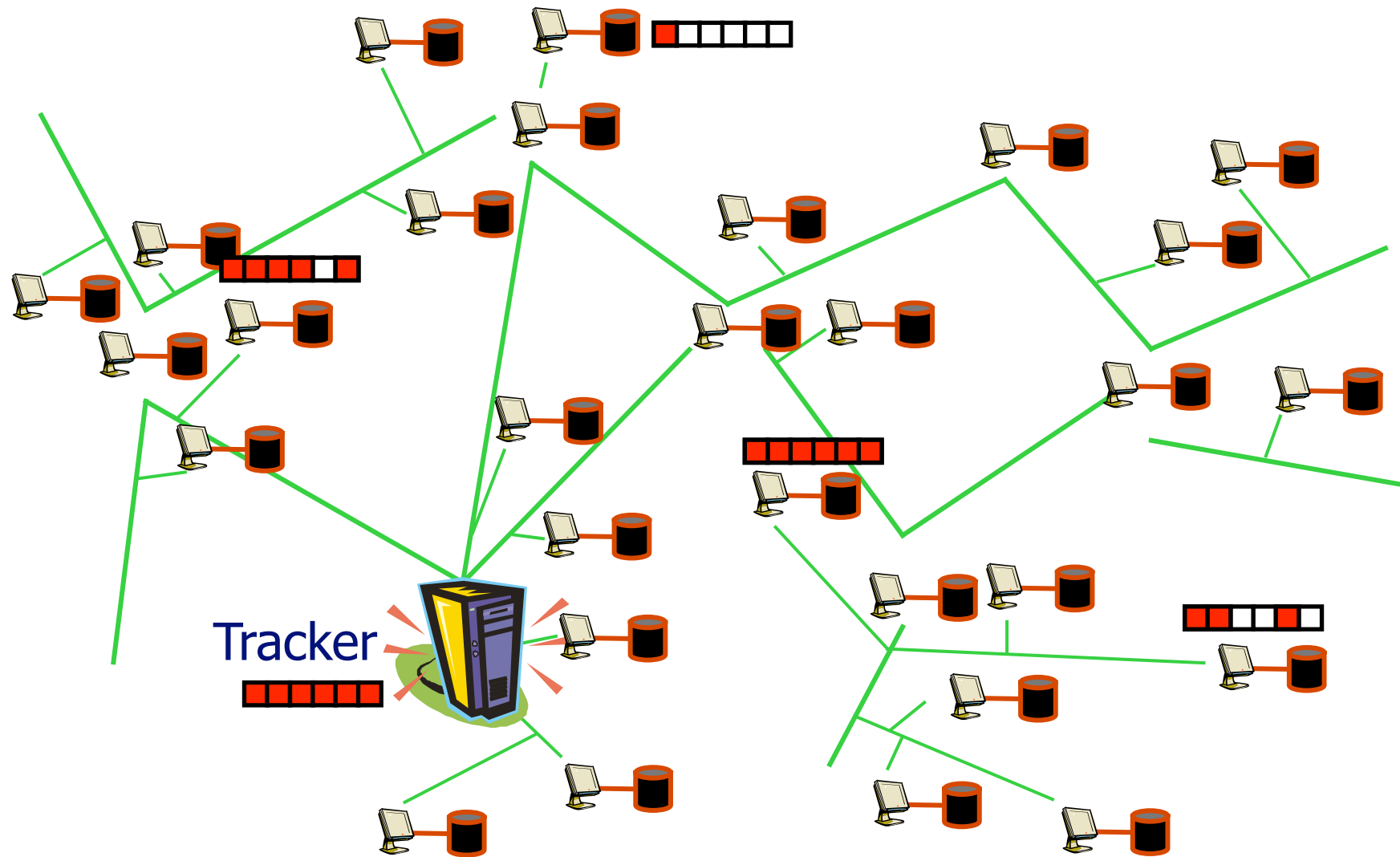


# BitTorrent





# BitTorrent



# BitTorrent Assessment

---

- Scalability, fairness, load balancing
  - Large distributed storage
  - Avoids broadcasts
  - Transfer content segments rather than complete content
  - Does not rely on clients staying online after download completion
  - Contributors are allowed to download more
- Content location
  - Central server approach
- Failure resilience
  - Tracker is single point of failure



# Comparison

	Napster	Gnutella	FreeNet	FastTrack	BitTorrent
Scalability		Limited by flooding	Uses caching		Separate overlays per file
Routing information	One central server	Neighbour list		Index server	One tracker per file
Lookup cost	$O(1)$	$O(\log(\#\text{nodes}))$	$O(\#\text{nodes})$	$O(1)$	$O(\#\text{blocks})$
Physical locality				By search server assignment	



# Comparison

	Napster	Gnutella	FreeNet	FastTrack	BitTorrent
Load balancing	Many replicas of popular content		Content placement changes to fit search	Load concentrated on supernodes	Rarest first copying
Content location	All files reachable	Unpopular files may be outside TTL		All files reachable Search by hash	External issue
	Uses index server Search by index term	Uses flooding	Search by hash		
Failure resilience	Index server as single point of failure	No single point of failure		Overlay network of index servers	Tracker as single point of failure



# Peer-to-Peer Lookup Protocols

---

Distributed Hash Tables (DHTs)

# Challenge: Fast, efficient lookups

---

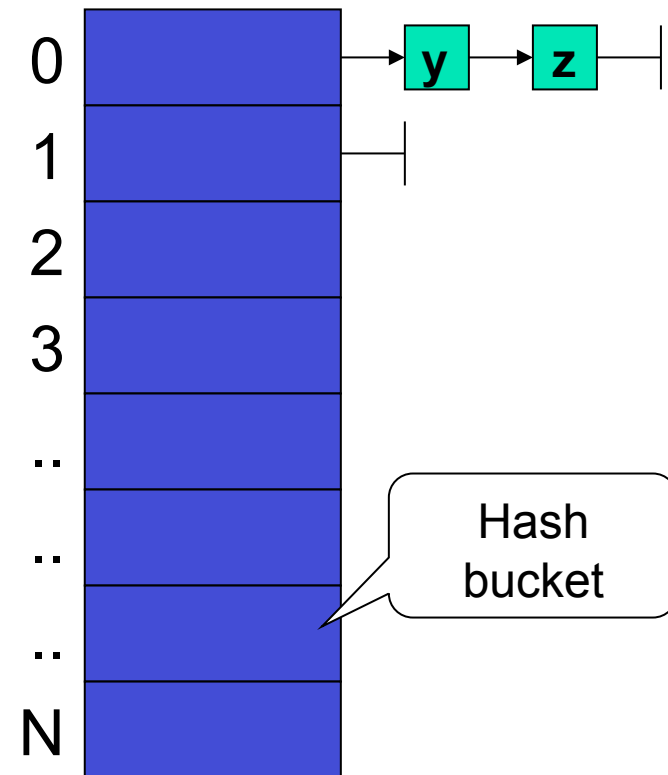
- Napster has one central server
- Gnutella uses flooding
- Freenet uses a distributed approach, but their key based heuristics make items with similar keys to cluster
- Distributed Hash Tables (DHTs) use a more structured key based routing
  - distribution properties of Gnutella and Freenet
  - efficiency and lookup guarantee of Napster

# Lookup Based on Hash Tables

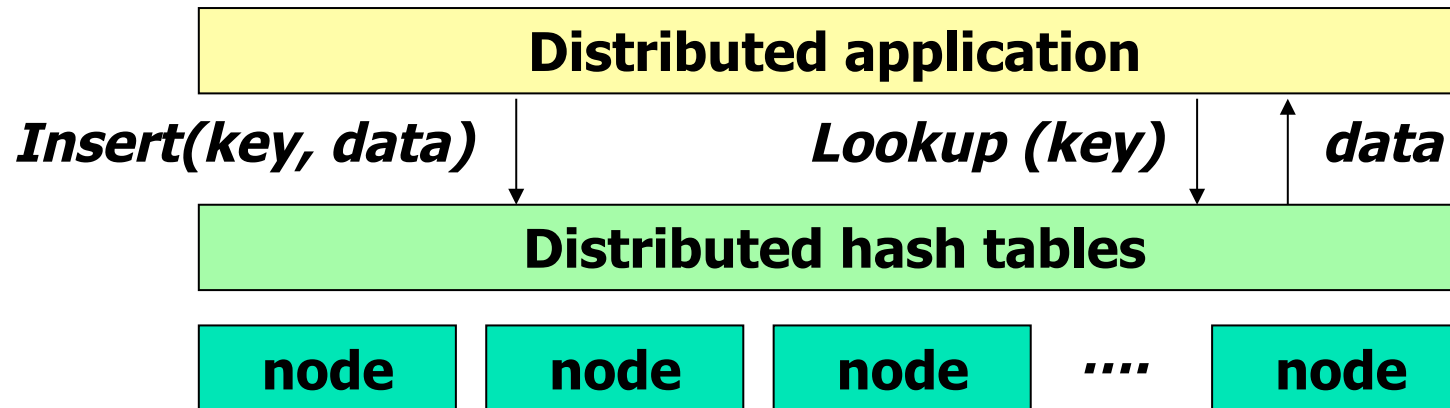
*lookup(key) → data*  
*Insert(key, data)*



*hash table*



# Distributed Hash Tables (DHTs)



- Key identifies data uniquely
- Nodes are the hash buckets
- The keyspace is partitioned
  - usually a node with ID = X has elements with keys close to X
  - must define a useful key nearness metric
  - DHT should balance keys and data across nodes
- Keep the hop count small
- Keep the routing tables "right size"
- Stay robust despite rapid changes in membership





# Examples: Chord

---

# Chord

---

- Approach taken
  - Only concerned with efficient indexing
  - Distributed index - decentralized lookup service
  - Inspired by consistent hashing: SHA-1 hash
  - Content handling is an external problem entirely
    - No relation to content
    - No included replication or caching
- P2P aspects
  - Every node must maintain keys
  - Adaptive to membership changes
  - Client nodes act also as file servers



# Chord IDs & Consistent Hashing

- $m$  bit identifier space for both keys and nodes
  - Key identifier = SHA-1(key)

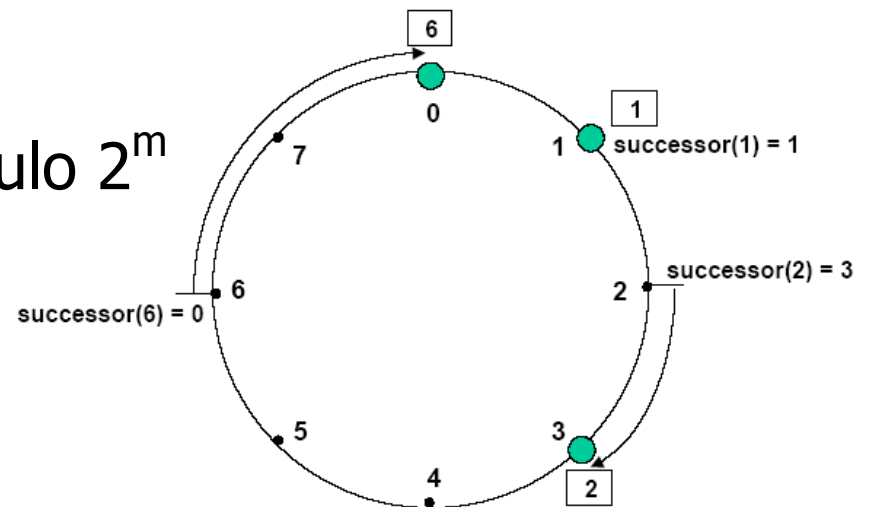
Key="LetItBe"  $\xrightarrow{\text{SHA-1}}$  ID=54

- Node identifier = SHA-1(IP address)

IP="198.10.10.1"  $\xrightarrow{\text{SHA-1}}$  ID=123

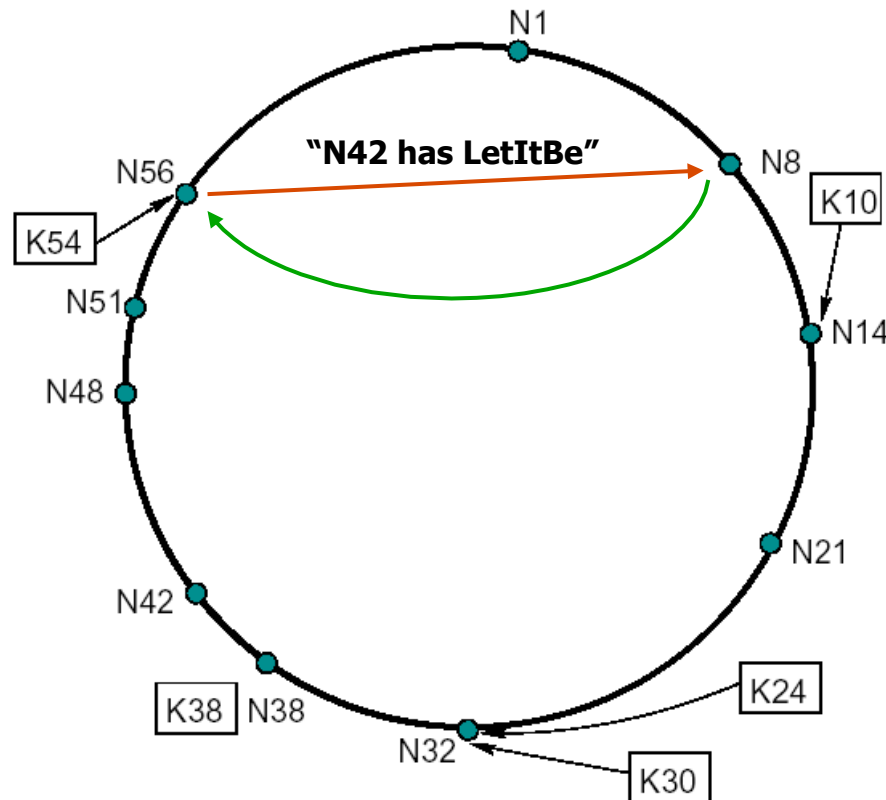
- Both are uniformly distributed

- Identifiers ordered in a circle modulo  $2^m$
- A key is mapped to the first node whose id is equal to or follows the key id



# Routing: Everyone-Knows-Everyone

- Every node knows of every other node - requires global information
- Routing tables are large –  $N$



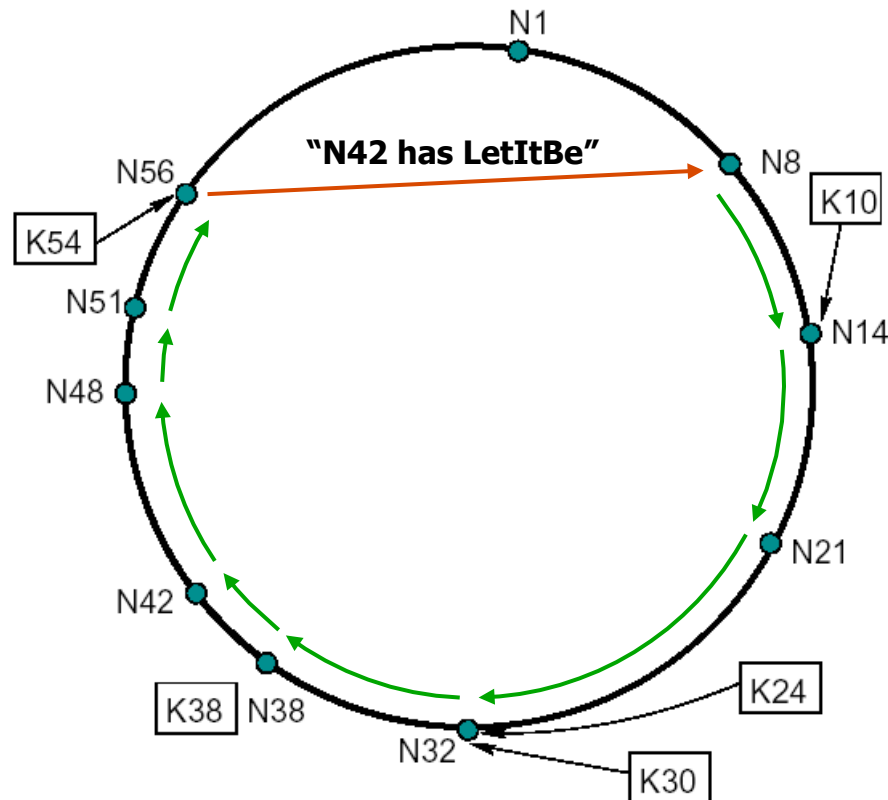
Where is "LetItBe"?

Hash("LetItBe") = K54

Requires **O(1)** hops

# Routing: All Know Their Successor

- Every node only knows its successor in the ring
- Small routing table – 1



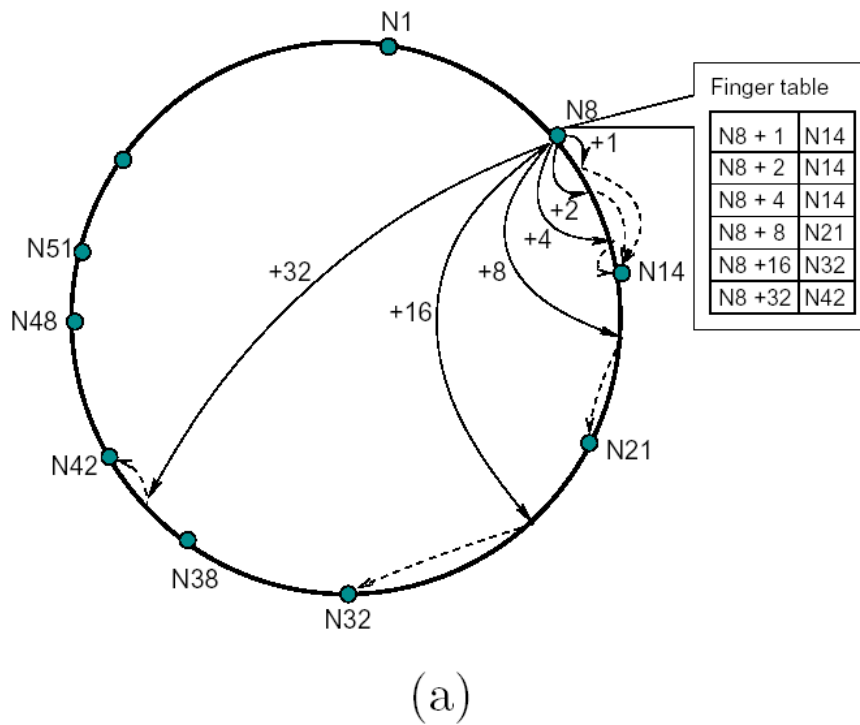
Where is “LetItBe”?

Hash(“LetItBe”) = K54

Requires  $O(N)$  hops

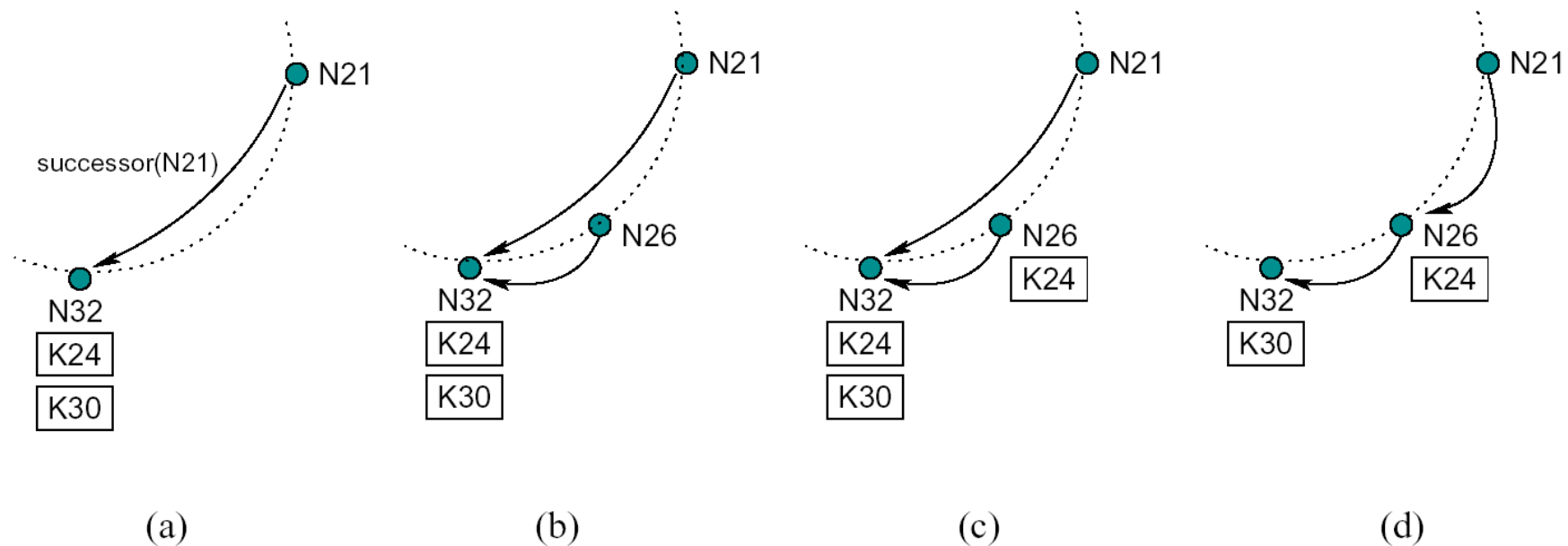
# Routing: "Finger Tables"

- Every node knows  $m$  other nodes in the ring
- Increase distance exponentially
- Finger  $i$  points to **successor** of  $n+2^i$



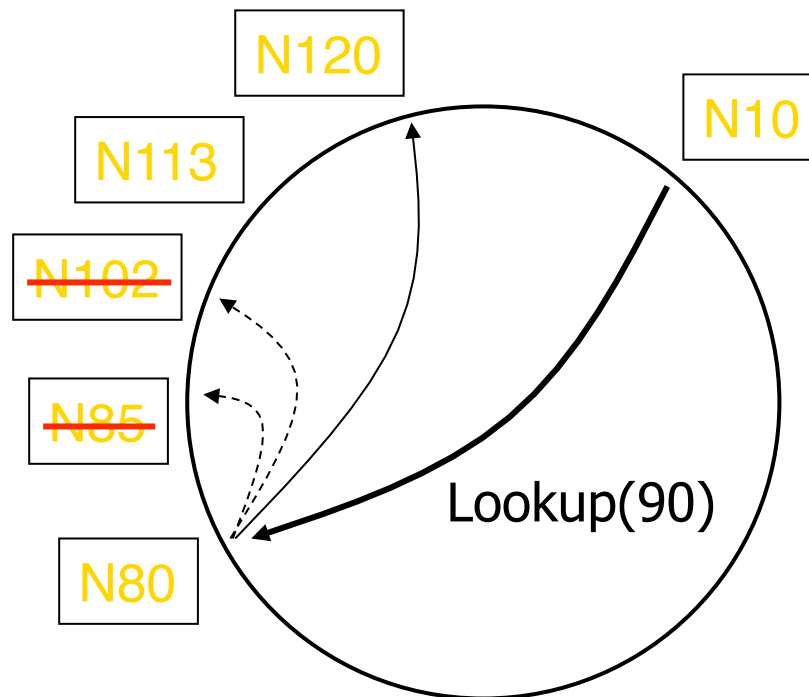
# Joining the Ring

- Three step process:
  - Initialize all fingers of new node - by asking another node for help
  - Update fingers of existing nodes
  - Transfer keys from successor to new node



# Handling Failures

- Failure of nodes might cause incorrect lookup



- N80 doesn't know correct successor, so lookup fails
- One approach: successor lists
  - Each node knows  $r$  immediate successors
  - After failure find first known live successor
  - Increased routing table size



# Chord Assessment

---

- Scalability, fairness, load balancing
  - Large distributed index
  - Logarithmic search effort
  - Network topology is **not** accounted for
  - Routing tables contain  $\log(\#nodes)$
  - Quick lookup in large systems, low variation in lookup costs
- Content location
  - Search by hash key: limited ways to formulate queries
  - All indexed files are reachable
  - $\log(\#nodes)$  lookup steps
  - Not restricted to file location
- Failure resilience
  - No single point of failure
  - Not in basic approach
    - Successor lists allow use of neighbors to failed nodes
    - Salted hashes allow multiple indexes
  - Relies on well-known relationships, but fast awareness of disruption and rebuilding



# Examples: Pastry

---

# Pastry

---

- Approach taken
  - Only concerned with efficient indexing
  - Distributed index - decentralized lookup service
  - Uses DHTs
  - Content handling is an external problem entirely
    - No relation to content
    - No included replication or caching
  
- P2P aspects
  - Every node must maintain keys
  - Adaptive to membership changes
  - Leaf nodes are special
  - Client nodes act also as file servers

# Pastry

- DHT approach
  - Each node has unique 128-bit nodeId
    - Assigned when node joins
    - Used for routing
  - Each message has a key
  - NodeIds and keys are in base  $2^b$ 
    - $b$  is configuration parameter with typical value 4 (base = 16, hexadecimal digits)
  - Pastry node routes the message to the node with the closest nodeId to the key
  - Number of routing steps is  $O(\log N)$
  - Pastry takes into account network locality
  
- Each node maintains
  - *Routing table* is organized into  $\lceil \log_{2^b} N \rceil$  rows with  $2^b - 1$  entry each
  - *Neighborhood set  $M$*  — nodeId's, IP addresses of  $|M|$  closest nodes, useful to maintain locality properties
  - *Leaf set  $L$*  — set of  $|L|$  nodes with closest nodeId

# Pastry Routing

b=2, so nodeid is base 4

Contains the nodes that are numerically closest to local node

Nodeid 10233102			
Leaf set	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232
Routing table			
-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	
Neighborhood set $2^b-1$ entries per row			
13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

$\lceil \log_{2^b} N \rceil$  rows

Entries in the  $m^{th}$  column have  $m$  as  $n^{th}$  row digit

Entries in the  $n^{th}$  row share the first  $n-1$  digits with current node  
**common prefix – next digit – rest**

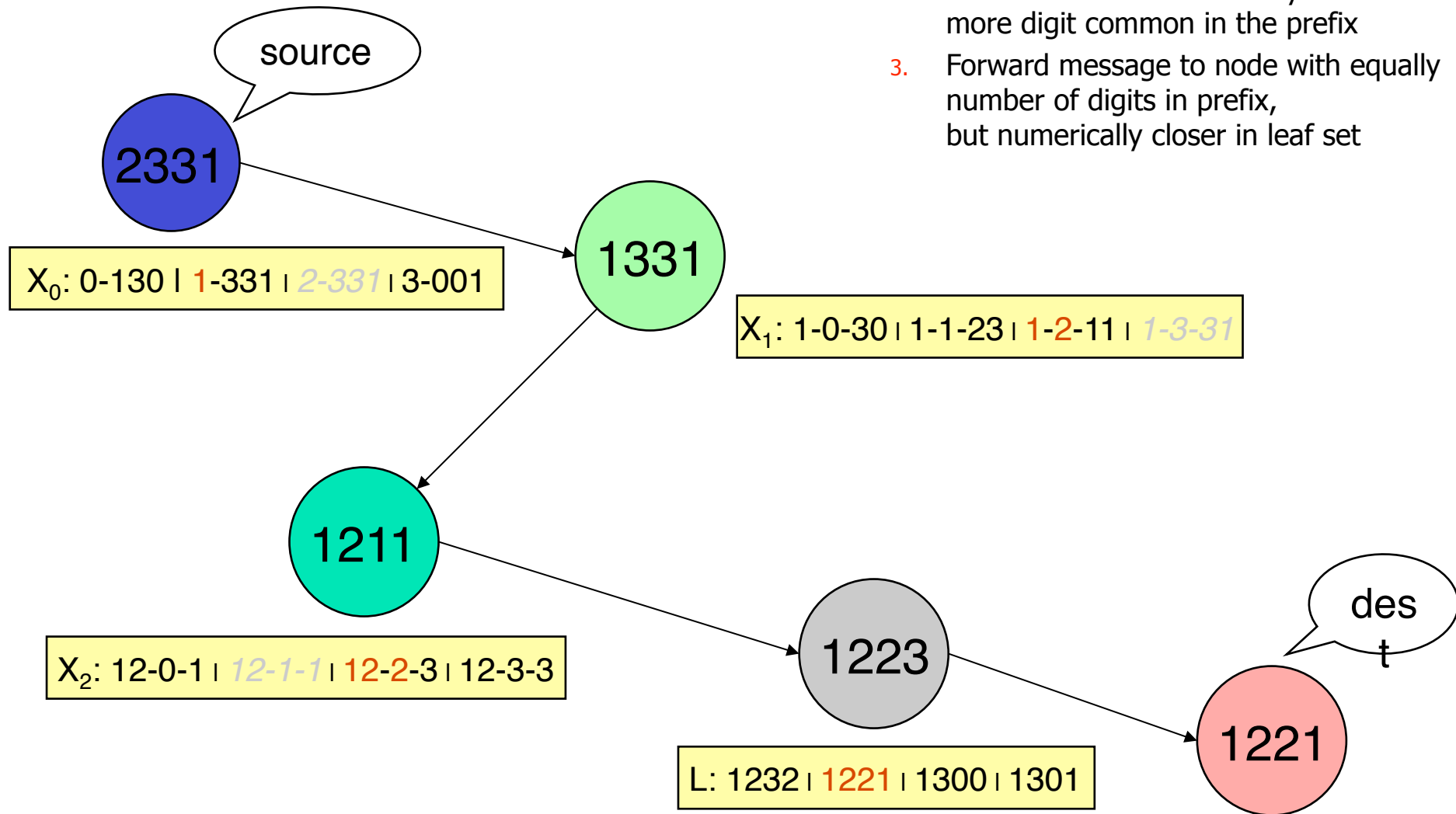
Entries with no suitable nodeid are left empty

Contains the nodes that are closest to local node according to proximity metric



# Pastry Routing

1. Search leaf set for exact match
2. Search route table for entry with at one more digit common in the prefix
3. Forward message to node with equally number of digits in prefix, but numerically closer in leaf set



# Pastry Assessment

---

- Scalability, fairness, load balancing
  - Distributed index of arbitrary size
  - Support for physical locality and locality by hash value
  - Stochastically logarithmic search effort
  - Network topology is partially accounted for, given an additional metric for physical locality
  - Stochastically logarithmic lookup in large systems, variable lookup costs
- Content location
  - Search by hash key: limited ways to formulate queries
  - All indexed files are reachable
  - Not restricted to file location
- Failure resilience
  - No single point of failure
  - Several possibilities for backup routes





# Examples: Tapestry

---



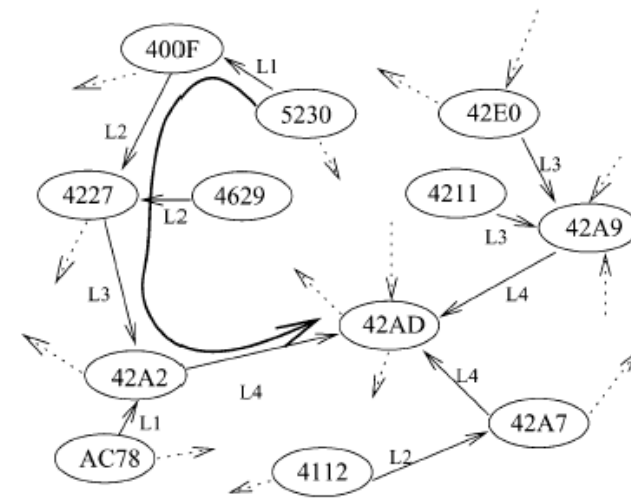
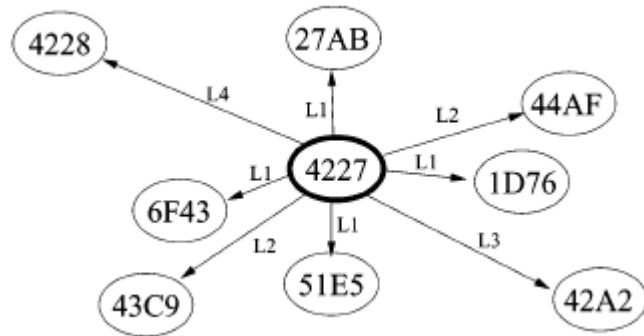
# Tapestry

---

- Approach taken
  - Only concerned with self-organizing indexing
  - Distributed index - decentralized lookup service
  - Uses DHTs
  - Content handling is an external problem entirely
    - No relation to content
    - No included replication or caching
- P2P aspects
  - Every node must maintain keys
  - Adaptive to changes in membership and value change

# Routing and Location

- Namespace (nodes and objects)
  - SHA-1 hash: 160 bits length
  - Each object has its own hierarchy rooted at  $\text{RootID} = \text{hash}(\text{ObjectID})$
- Prefix-routing [JSAC 2004]
  - Router at  $h^{\text{th}}$  hop shares *prefix* of length  $\geq h$  digits with destination
  - local tables at each node (neighbor maps)
  - route digit by digit:  $4^{***} \rightarrow 42^{**} \rightarrow 42A^* \rightarrow 42AD$
  - neighbor links in levels



# Routing and Location

- Suffix routing [tech report 2001]
  - Router at  $h^{th}$  hop shares *suffix* of length  $\geq h$  digits with destination
  - Example: 5324 routes to 0629 via  
5324  $\rightarrow$  2349  $\rightarrow$  1429  $\rightarrow$  7629  $\rightarrow$  0629
- Tapestry routing
  - Cache pointers to all copies
  - Caches are soft-state
  - UDP Heartbeat and TCP timeout to verify route availability
  - Each node has 2 backup neighbors
  - Failing primary neighbors are kept for some time (days)
  - Multiple `root` nodes possible, identified via hash functions
    - Search `value` in a `root` if its hash is that of the `root`
    - Choosing a `root` node
      - Choose a random address
      - Route towards that address
      - If no route exists, choose **deterministically**, a surrogate



# Routing and Location

- Object location
  - Root responsible for storing object's location (but not the object)
  - Publish / search both routes incrementally to root
- Locates objects
  - Object: `key/value pair`
    - E.g. `filename/file`
  - Automatic replication of `keys`
  - No automatic replication of `values`
- Values
  - May be replicated
  - May be stored in erasure-coded fragments



# Tapestry

---

- Location service for mobile objects
  - Location of and access to mobile objects
  - Self-organizing, scalable, robust, wide-area infrastructure
  - Self-administering, fault-tolerant, resilient under load
  - Point-to-point communication, no centralized resources
- Locates objects
  - Object: key/value pair
    - E.g. filename/file
  - Automatic replication of keys
  - No automatic replication of values
- Values
  - May be replicated
  - May be stored in erasure-coded fragments

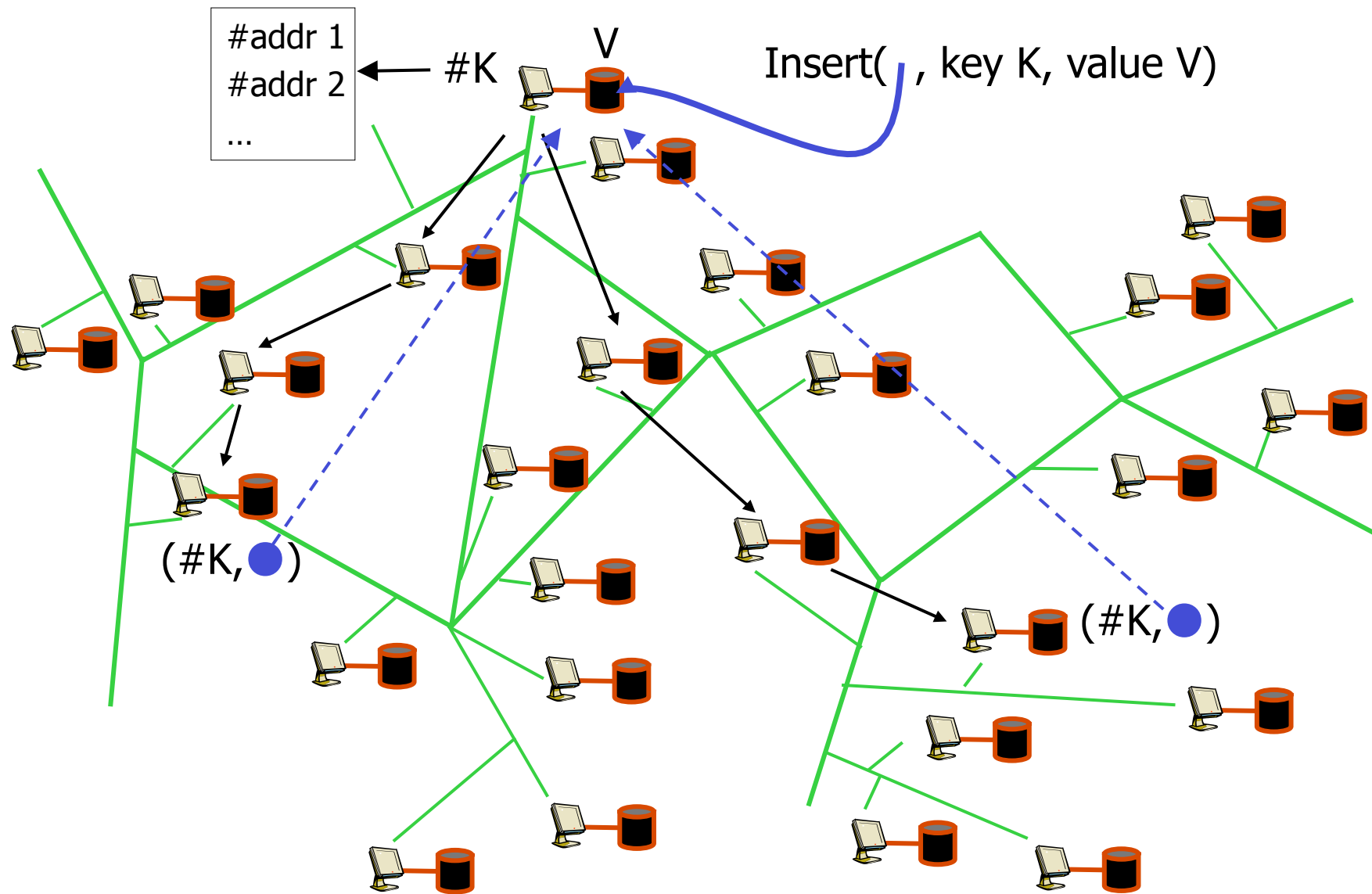
# Tapestry

---

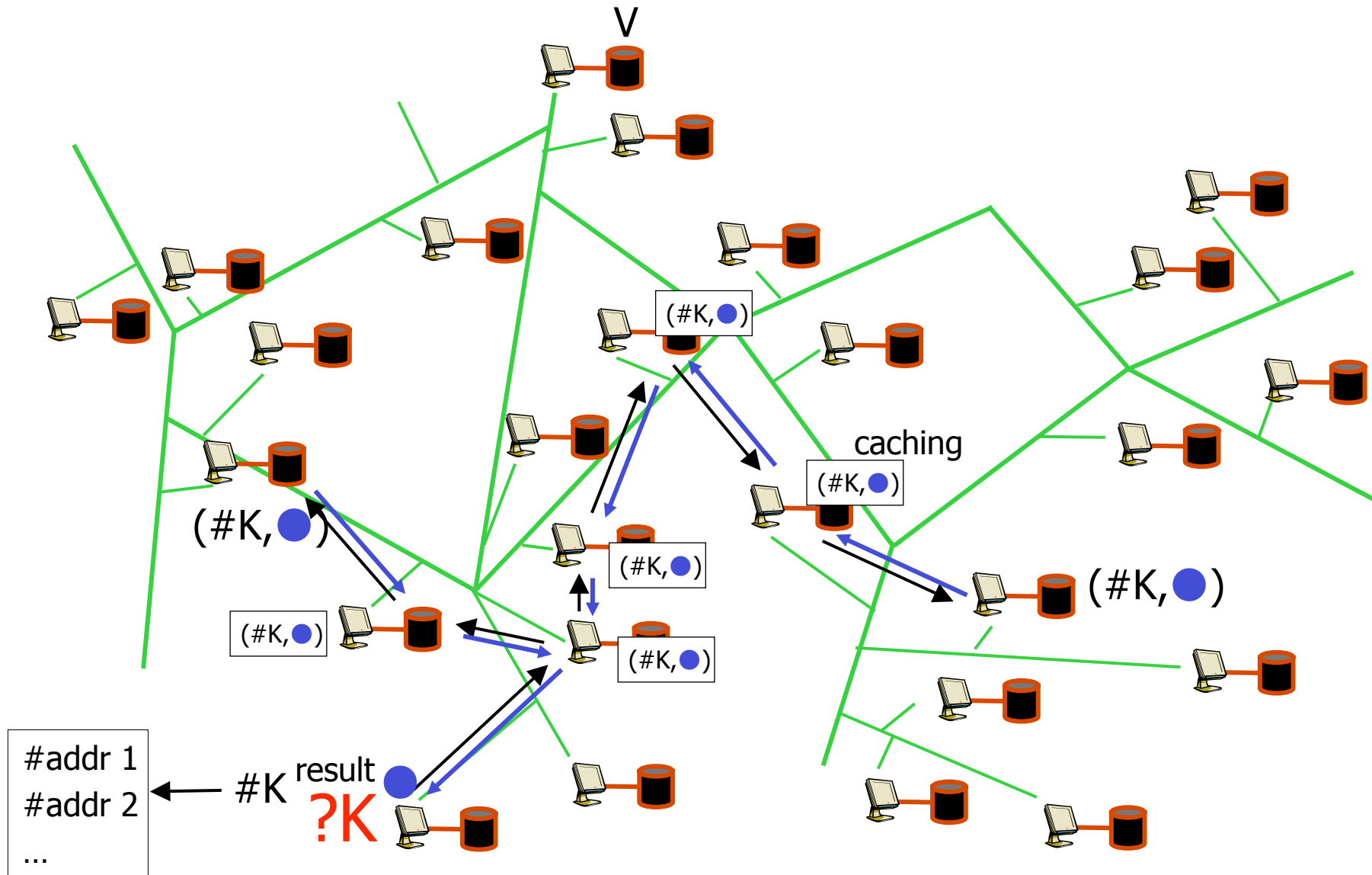
- Routing and directory service
- Goal
  - Find route to closest copy
- Routing
  - Forwarding of messages in the overlay network of nodes
- Nodes
  - Act as servers, routers and clients
- Routing
  - Each object has a unique `root`, identified by its `key`
  - Hash value of `key` **is** source route prefix to object's `root`
  - `Root` answers with address of `value`'s location
  - Routers cache the response



# Tapestry

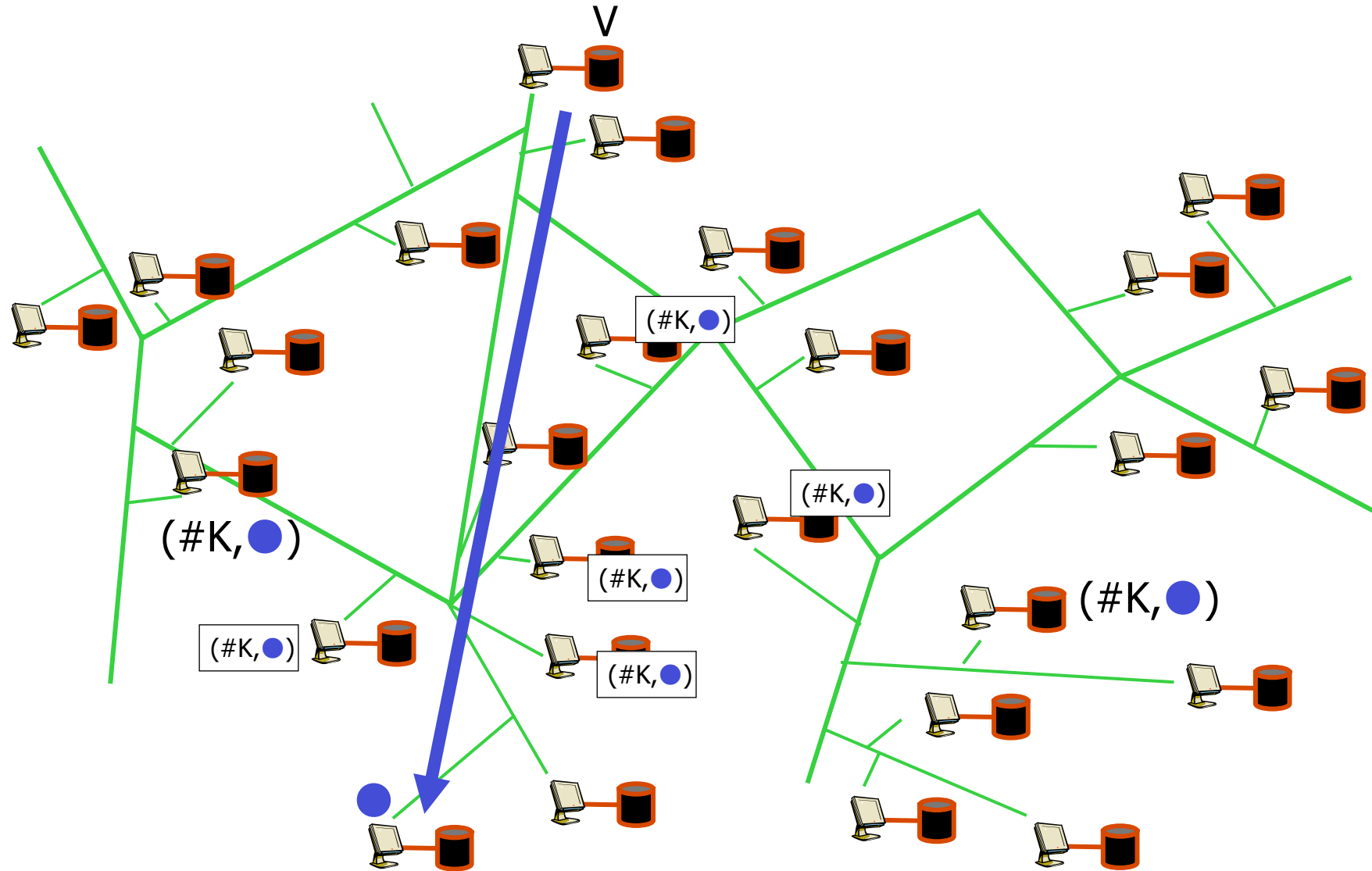


# Tapestry

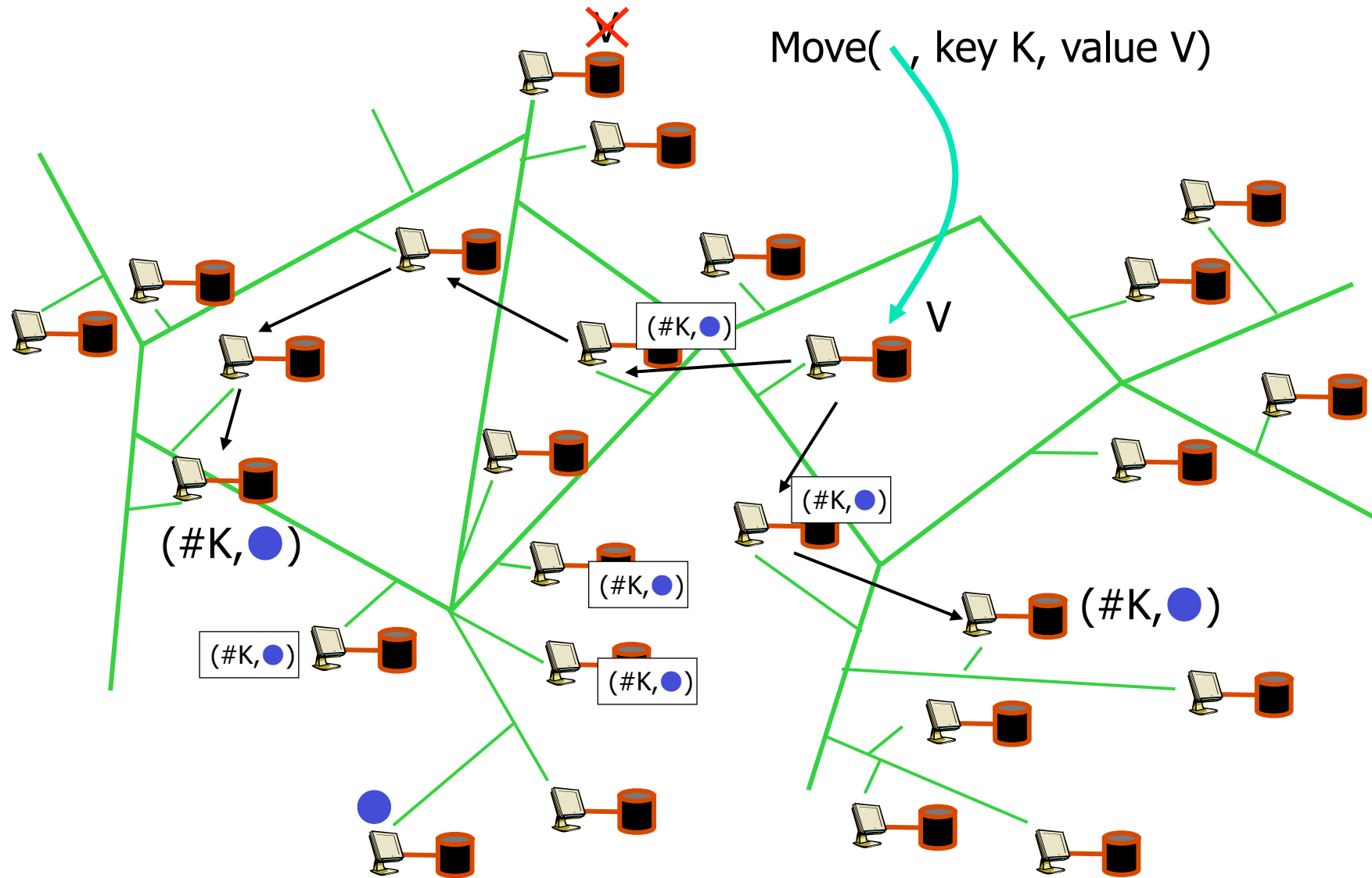




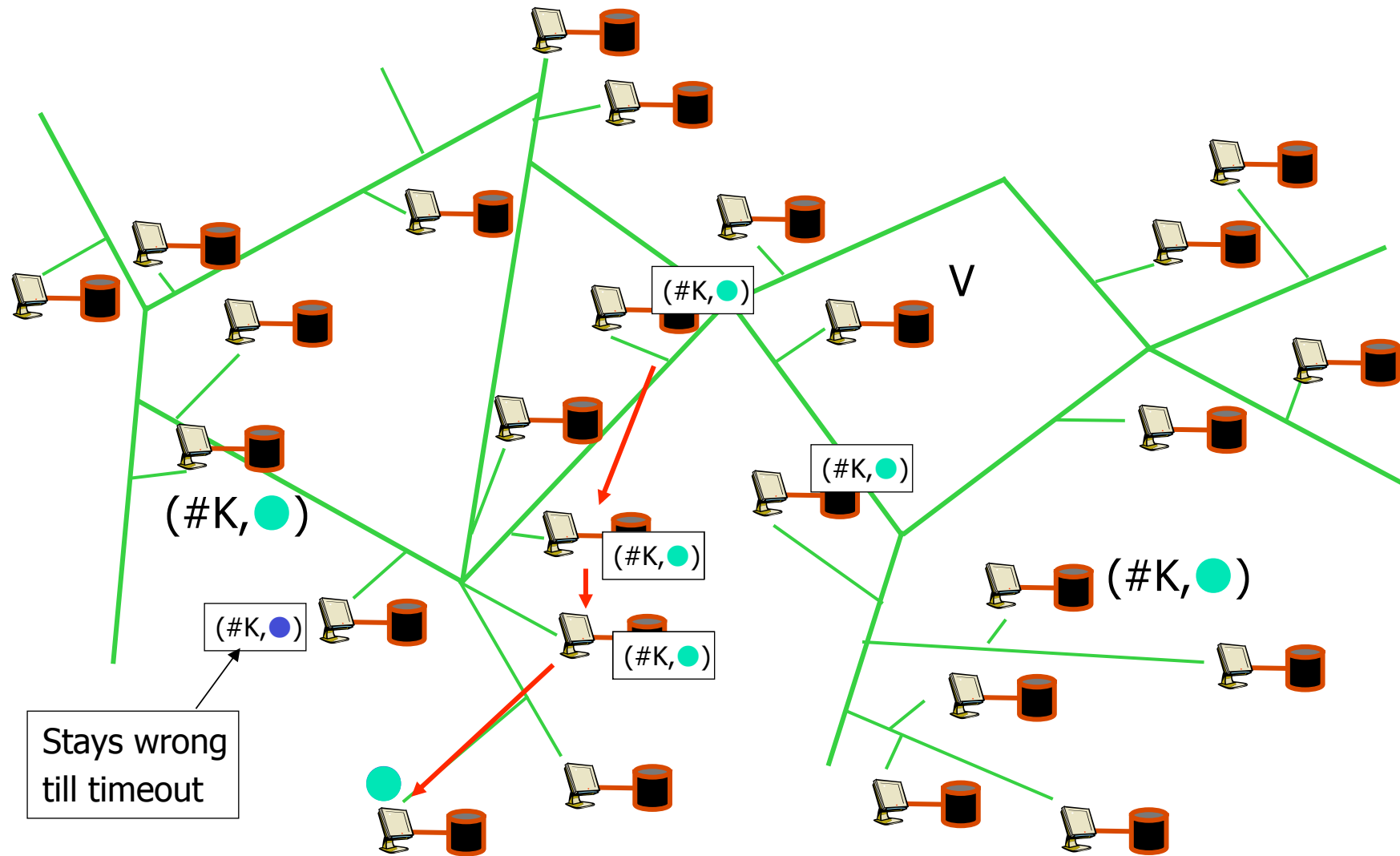
# Tapestry



# Tapestry



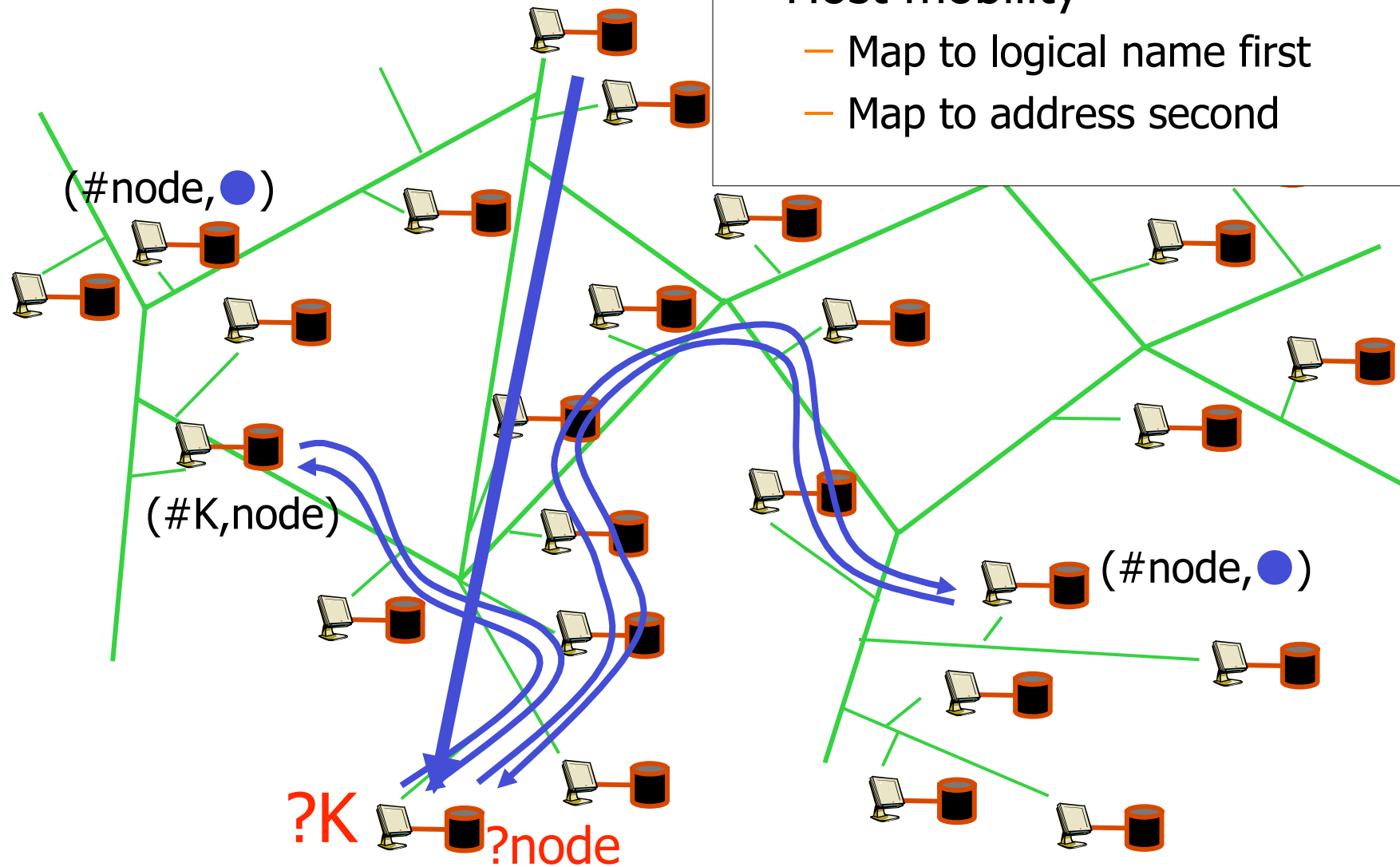
# Tapestry



# Mobile Tapestry

- Host mobility

- Map to logical name first
- Map to address second



# Tapestry Assessment

---

- Scalability, fairness, load balancing
  - Distributed index(es) of arbitrary size
  - Limited physical locality of key access by caching and nodeId selection
  - Variable lookup costs
  - Independent of content scalability
- Content location
  - Search by hash key: limited ways to formulate queries
  - All indexed files are reachable
  - Not restricted to file location
- Failure resilience
  - No single point of failure
  - Several possibilities for backup routes
  - Caching of key resolutions
  - Use of hash values with several salt values





# Comparison

---

# Comparison

	Chord	Pastry	Tapestry
Routing information	Log(#nodes) routing table size	Log(#nodes) x (2 <sup>b</sup> - 1) routing table size	At least log(#nodes) routing table size
Lookup cost	Log(#nodes) lookup cost	Approx. log(#nodes) lookup cost	Variable lookup cost
Physical locality		By neighbor list	In mobile tapestry
Failure resilience	No resilience in basic version  Additional successor lists provide resilience	No single point of failure Several backup route	No single point of failure Several backup route Alternative hierarchies



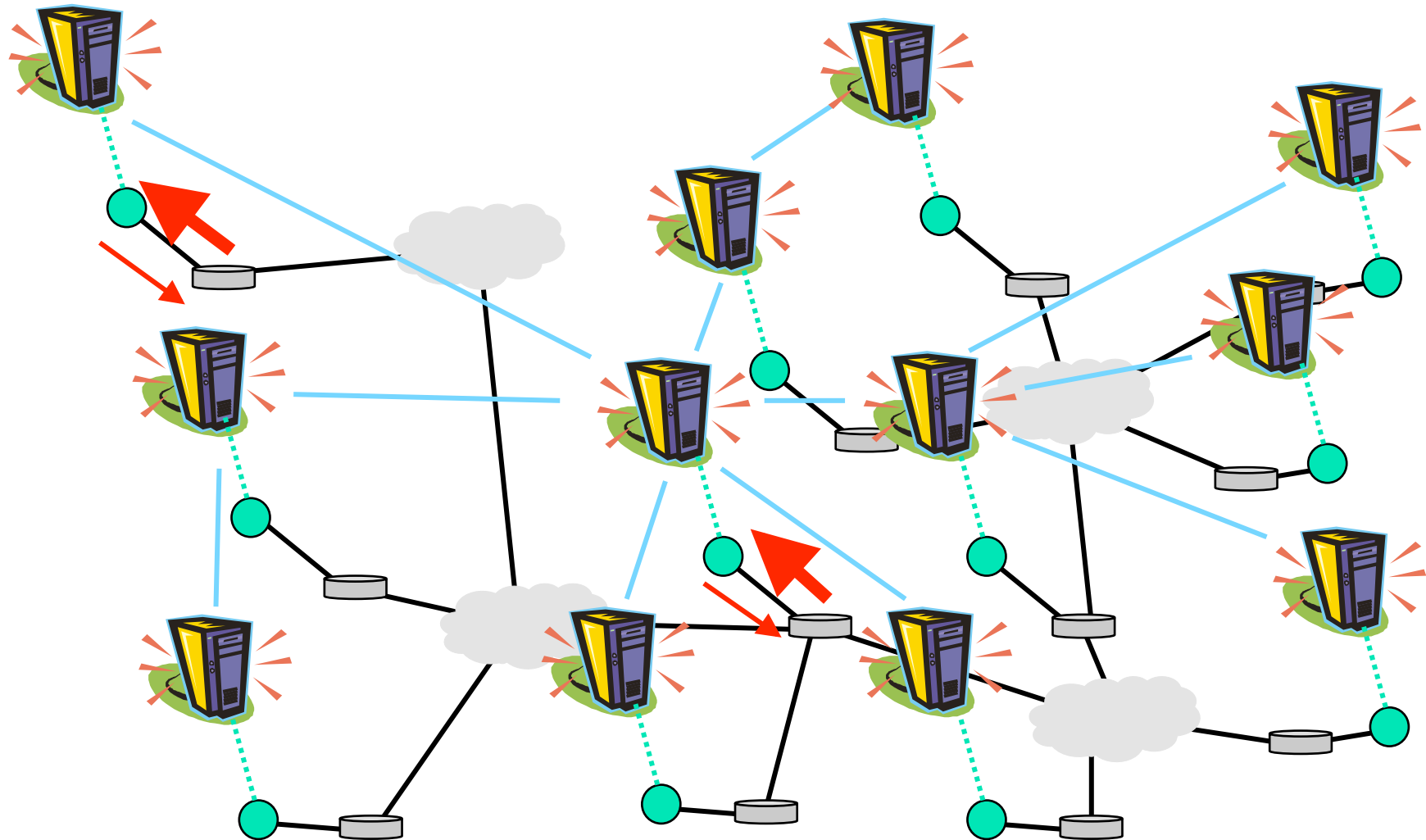
# **Applications:**

Streaming in Peer-to-peer networks

---



# Peer-to-peer network





Promise

---

# Promise

---

- Video streaming in Peer-to-Peer systems
  - Video segmentation into many small segments
  - Pull operation
  - Pull from several sources at once
  
- Based on Pastry and CollectCast
  
- CollectCast
  - Adds rate/data assignment
  - Evaluates
    - Node capabilities
    - Overlay route capabilities
  - Uses topology inference
    - Detects shared path segments - using ICMP similar to traceroute
    - Tries to avoid shared path segments
    - Labels segments with quality (or goodness)

# Promise

[Hafeeda et. al. 03]

**Each active sender:**

- receives a control packet specifying which data segments, data rate, etc.,
- pushes data to receiver as long as no new control packet is received

active sender



standby sender



standby sender



## The receiver:

- sends a lookup request using DHT
- selects some active senders, control packet
- receives data as long as no errors/changes occur
- if a change/error is detected, new active senders may be selected



Thus, **Promise** is a multiple sender to one receiver P2P media streaming system which 1) accounts for different capabilities, 2) matches senders to achieve best quality, and 3) dynamically adapts to network fluctuations and peer failure

active sender





# SplitStream

---

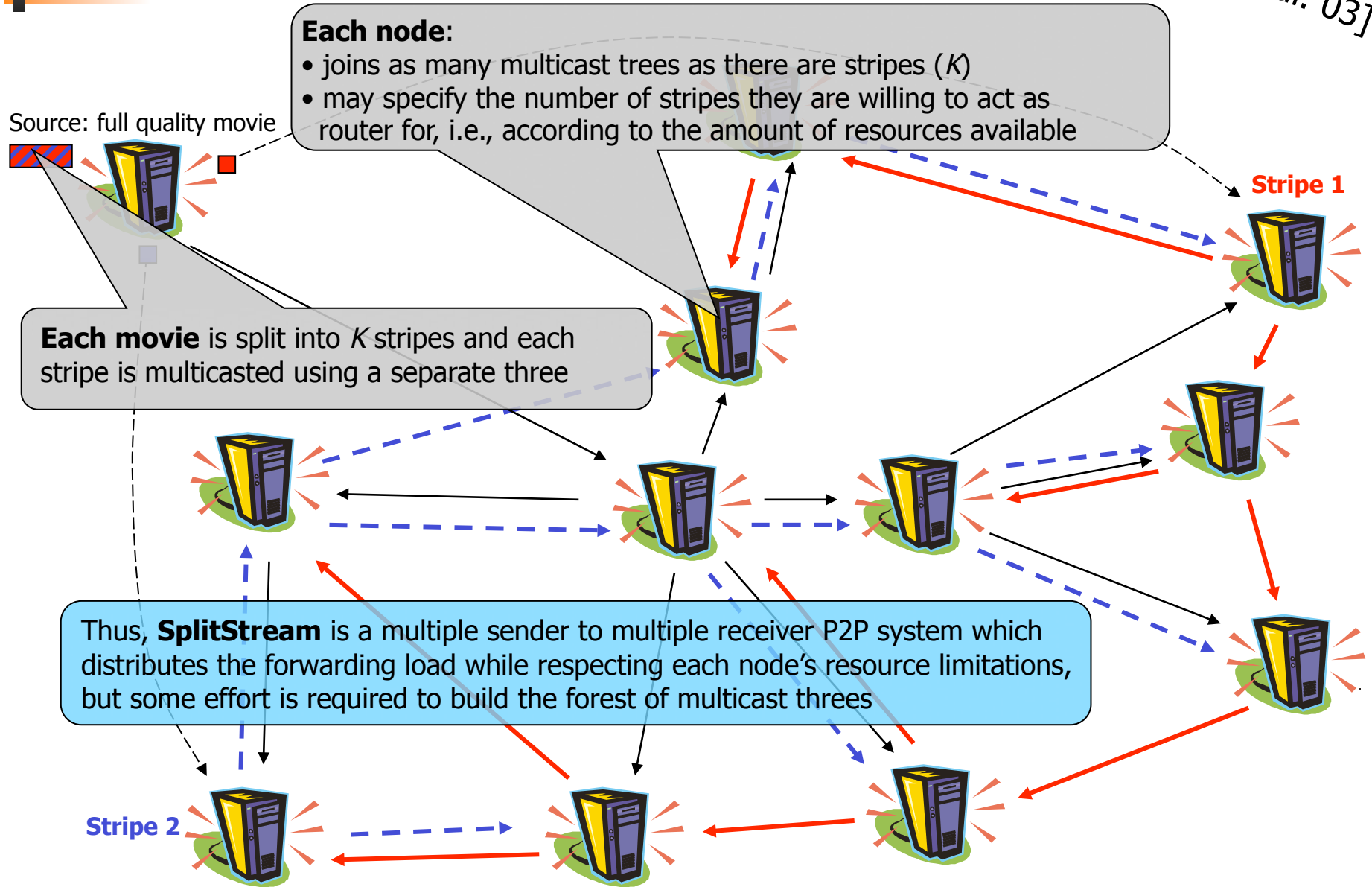
# SplitStream

---

- Video streaming in Peer-to-Peer systems
  - Uses layered video
  - Uses overlay multicast
  - Push operation
  - Build disjoint overlay multicast trees
  
- Based on Pastry

# SplitStream

[Castro et. al. 03]



# Some References

---

1. M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment", SOSP'03, Lake Bolton, New York, October 2003
2. Mohamed Hefeeda, Ahsan Habib, Boyan Botev, Dongyan Xu, Bharat Bhargava, "Promise: Peer-to-Peer Media Streaming Using Collectcast", ACM MM'03, Berkeley, CA, November 2003
3. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek and Hari Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications", ACM SIGCOMM'01
4. Ben Y. Zhao, John Kubiawicz and Anthony Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing", UCB Technical Report CSD-01-1141, 1996
5. John Kubiawicz, "Extracting Guarantees from Chaos", Comm. ACM, 46(2), February 2003
6. Antony Rowstron and Peter Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", Middleware'01, November 2001

