ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Decentralized Search

CS-438: Decentralized Systems Engineering

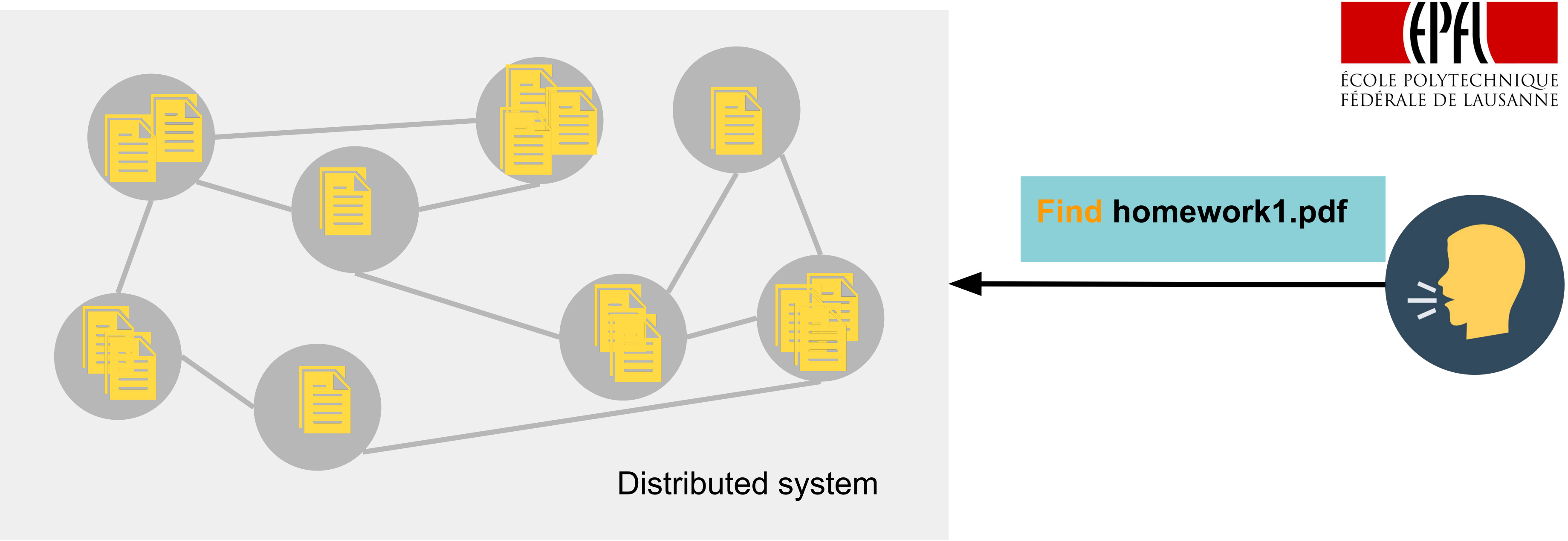
(slide credits: Cristina Basescu)

Decentralized search in systems around us

- **Techniques discussed here frequently appear as a sub-system**
  - Dynamo (Amazon's key value store) uses DHTs
  - Akamai's content delivery network (CDN) uses consistent hashing

- **Sometimes they a a user-facing system**
  - File sharing through torrents

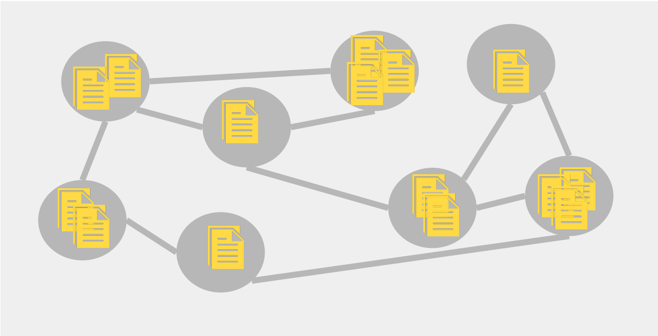
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE



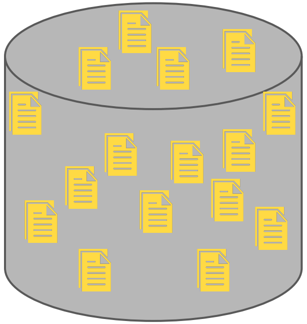
- Peers collectively store data & collaborate to retrieve data on request
- **Main challenge: find who stores a file**

Peer-to-peer vs centralized storage

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE



vs



- **Scalable**
  - Cheap to accommodate load variance
- **Fault-tolerant**
- **Dynamic**
  - Peers come and go (churn)
- **Self-organizing and adaptive**
  - Data replication, data locality, routes to find data
- **Resistant to censorship**

- **Easy to implement**
- **More difficult to scale**
  - overprovisioning
- **Single point of failure**

Peer-to-peer storage classification

- **Data organization: which node stores a particular file**
  - Structured: predetermined mapping between nodes and files
  - Unstructured: mapping fully flexible, nodes may dynamically become specialized

- **Degree of decentralization**
  - Fully decentralized: all nodes have the same capabilities
  - Partially centralized: some nodes have more capabilities than others

	Unstructured	**Structured**
Partially centralized	Napster	
Fully decentralized	Gnutella, Freenet, BubbleStorm	DIstributed Hash Tables

Roadmap

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

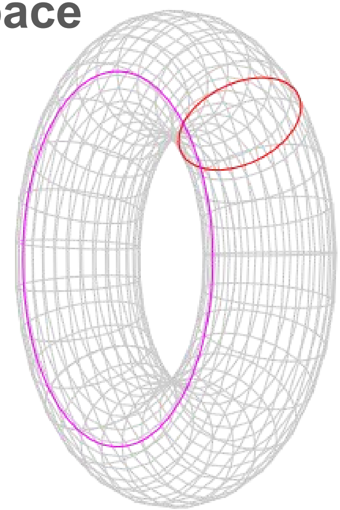
- **Structured Search: DHTs**
  - Place data cleverly to make it easier to find
  - CAN, Chord
- **Unstructured search**
  - Napster, Gnutella (brief), Freenet, BubbleStorm

Distributed Hash Tables

- **Same API as a hash table, but running on multiple peers**
  - Insert (key, item)
  - Item = lookup(id)
  - The item can be a file, data object, document, etc
- **Goals**
  - Existing item is always found
  - Scales to many nodes
  - Handles churn well
- **Many proposals, we'll look at two basic ones**
  - Content Addressable Network (CAN)
  - Chord

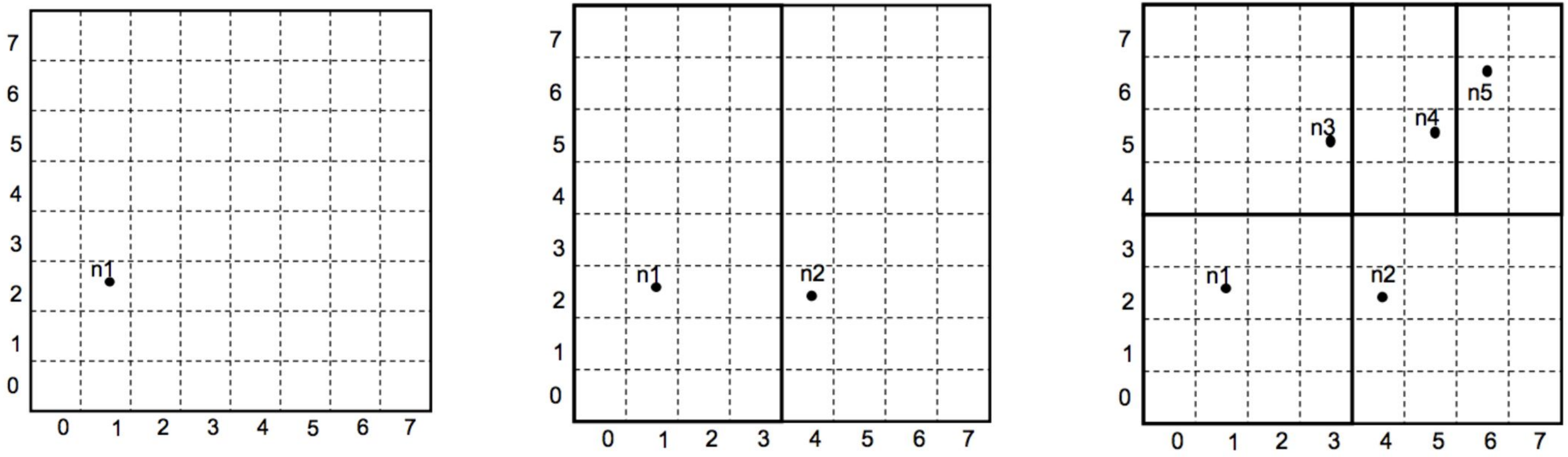
Content Addressable Network (CAN)

- **Each peer gets an identifier in a d-dimensional Cartesian space**
  - N peers
  - Peers are responsible for part of the space
- **Item gets an identifier in the same d-dimensional space**
  - Item stored on closest peer in space
- **Property**
  - Item retrieval in at most $d * N^{1/d}$ steps
  - Routing table at each peer is O(d)



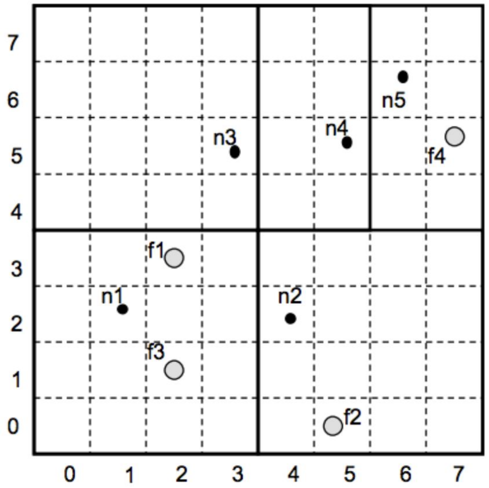
CAN Example: 2-dim space

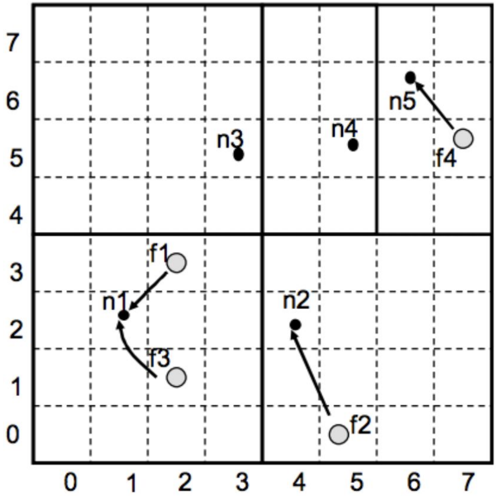
- **All peers cover the entire space**
  - Each covers either a square or a rectangle, depending on other peers' ids
- **Example**
  - N1 (1,2) joins: covers the entire space
  - Then N2 (4,2) joins: space divided between them
  - Then N3 (3,5), N4 (5,5), N5 (6,6)



CAN Example (cont)

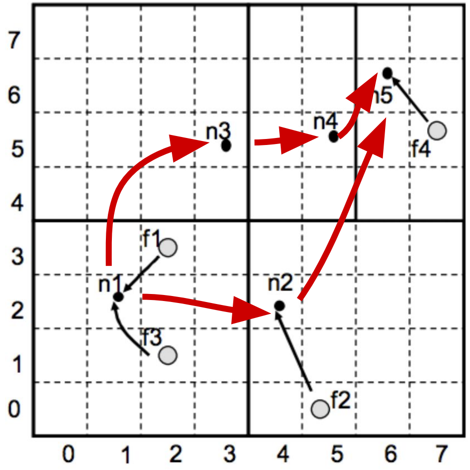
- **Items**
  - f1 (2,3), f2 (5,1), f3 (2,1), f4 (7,5)





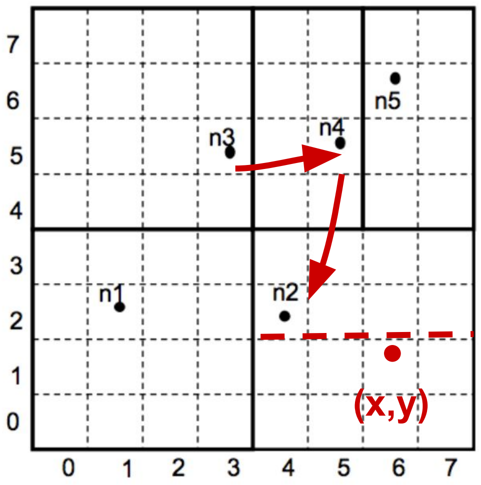
CAN Routing

- **Each node knows its immediate neighbors in d-space**
  - Here it's 2-dim space
  - n1 for example knows n2 and n3
  - What peers does n4 know? **n2, n3 and n5**
  - What about n5? **n4 and n2**
- **Lookup example**
  - n1 queries f4
  - Can route around **some failures**



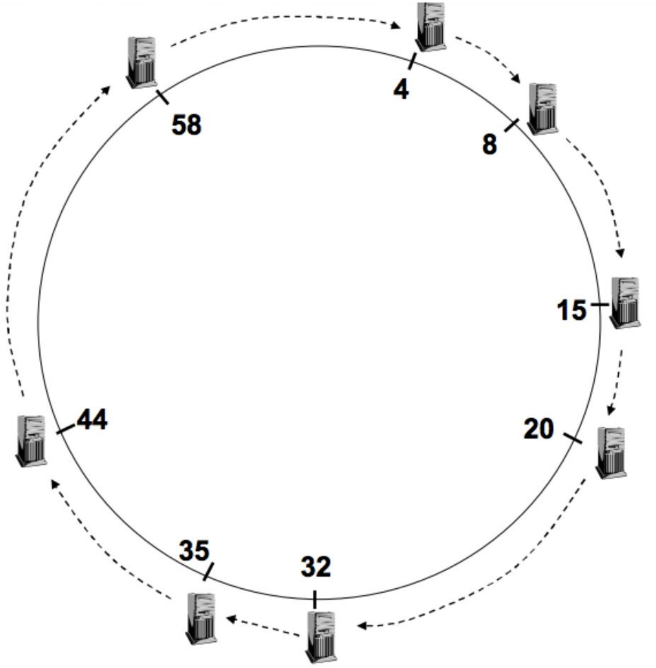
CAN joins and departures

- **Joins**
  - New peer A picks random id (x,y)
  - Bootstrap: A gets to know one peer in CAN, e,g, n3
  - n3 routes to peer that owns the point (x,y), here n2
  - n2 splits its space and offloads corresponding data to A
- **Departures**
  - Peer B leaves
  - B gives its data to neighbor peer with same space size
  - The neighbor peer thus doubles its space
- **Failures**
  - How to avoid data loss?
  - Replicate data on neighboring peers



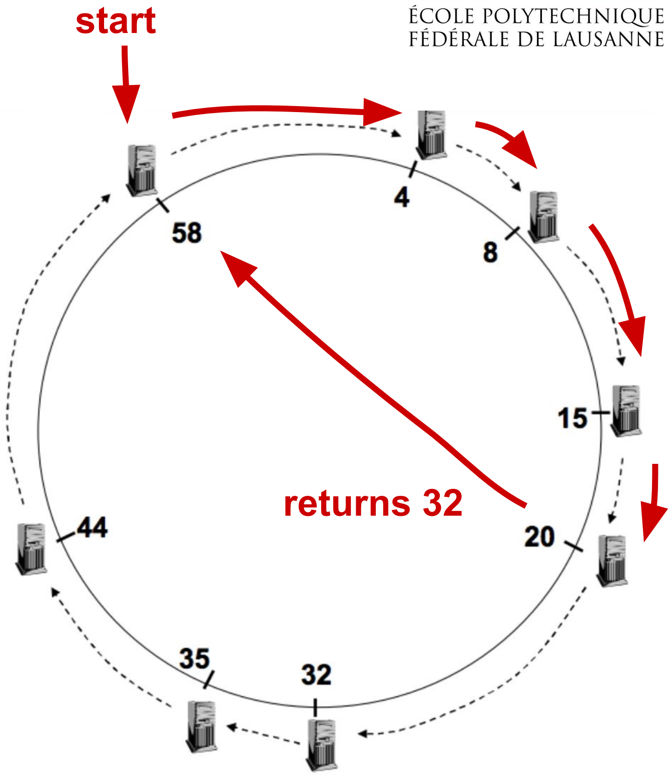
Chord

- **Different mapping of peers to item space**
  - N peers
  - Chord uses a ring, instead of a Cartesian space
  - Each peer gets an id in a uni-dimensional space $0..2^m$-1
  - Each item also gets such an id
  - A peer stores items with ids between the id of its predecessor peer and its own id, e,g, peer 8 stores items [5,..,8]
- **Properties**
  - Routing table at each node of size O(log N)
  - Guarantees an existing item found in O(log N) steps



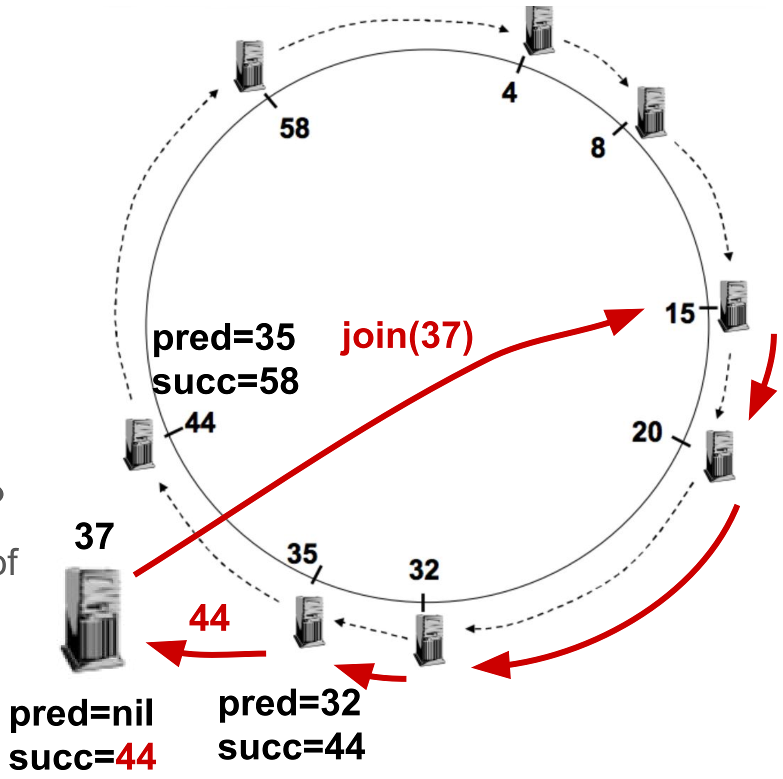
Chord: correctness

- **Key design decision**
  - Decouple correctness from efficiency
- **Let's first look at correctness**
- **Routing**
  - Each peer stores a link to its ring successor
- **Lookup of item with id_x**
  - Start at any node
  - Follow the successor link until encountering the first successor node whose id ≥ id_x
- **Eg: lookup item 25**
  - Starting at node 58
  - Stop at node 32: this node should store item 25
  - How do we know the item does not exist?



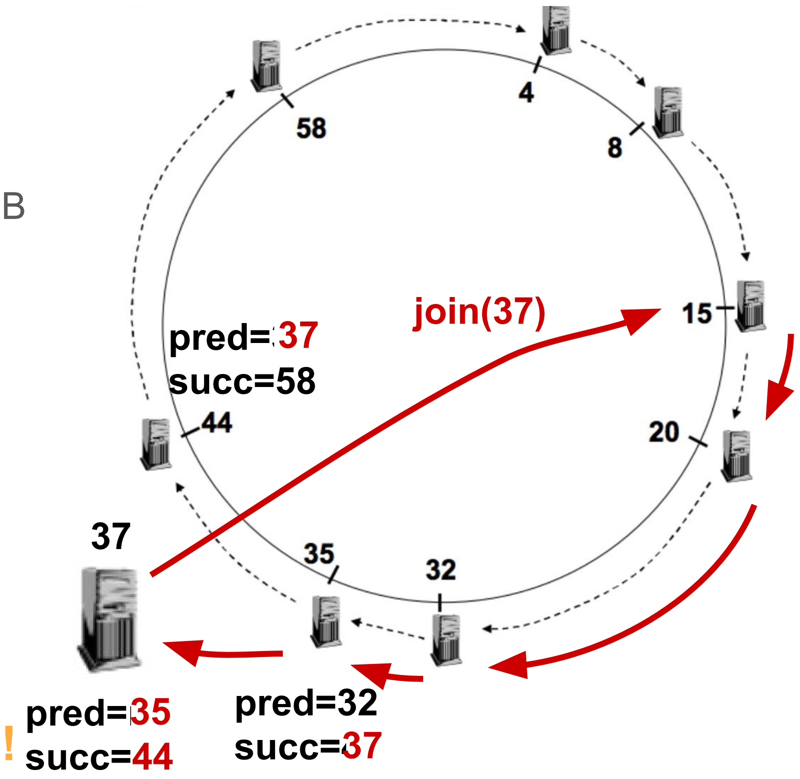
Chord: node join (1)

- **Node with id_A joins**
  - Bootstrap: needs to know at least one node id_K
  - Send to id_K join(id_A)
  - id_K performs a lookup $id_S \leftarrow$ lookup(id_A)
  - id_S becomes the successor of id_A
- **Eg node 37 joins, knows node 15**
  - Note that $pred_{44}$ and $succ_{35}$ do not change yet
- **Succ not up-to-date, what can go wrong?**
  - Everything still works correctly, as if 37 is not part of the system



Chord: node join (2)

- **Periodically nodes perform stabilize()**
  - A sends stabilize() to its succ C
  - C returns B ← $pred_C$ to A in a notify(B) message
  - C updates its $pred_C$ ← A if B < A
  - If B is between A and C, then A updates $succ_A$ ← B
- **Whose stabilize() corrects the val in eg?**
  - 37 sends stabilize() to its succ 44
  - 44 sends notify(35) back, updates its $pred_{44}$ ← 37
  - 35 sends stabilize() to its succ 44
  - 44 sends notify(37) back
  - 35 updates its $succ_{35}$ ← 37
  - 35 sends stabilize() to its succ 37
  - 37 sends notify(nil) back, updates its $pred_{37}$ ← 35
- **Order or updates crucial for correctness!**

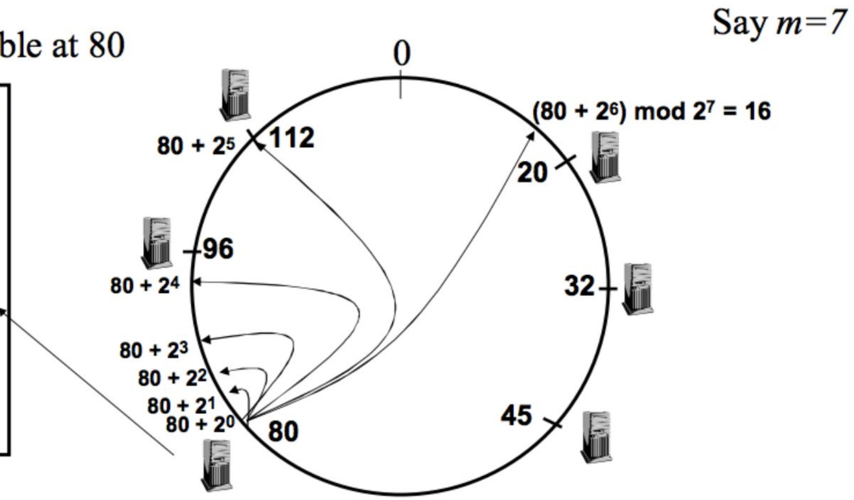


Chord efficiency: finger tables

- **Routing table of size O(log N)**
- **Item lookup of size O(log N)**

Finger Table at 80

i	$ft[i]$
0	96
1	96
2	96
3	96
4	96
5	112
6	20



ith entry at peer with id n is first peer with id >= $n + 2^i (\text{mod}\ 2^m)$

Chord: robustness

- **Each node maintains k immediate successors, instead of just 1**
  - Why? **If successor of new node dies, new node needs to restart join**
  - stabilize() and notify() change accordingly for k successors

CAN / Chord optimizations

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

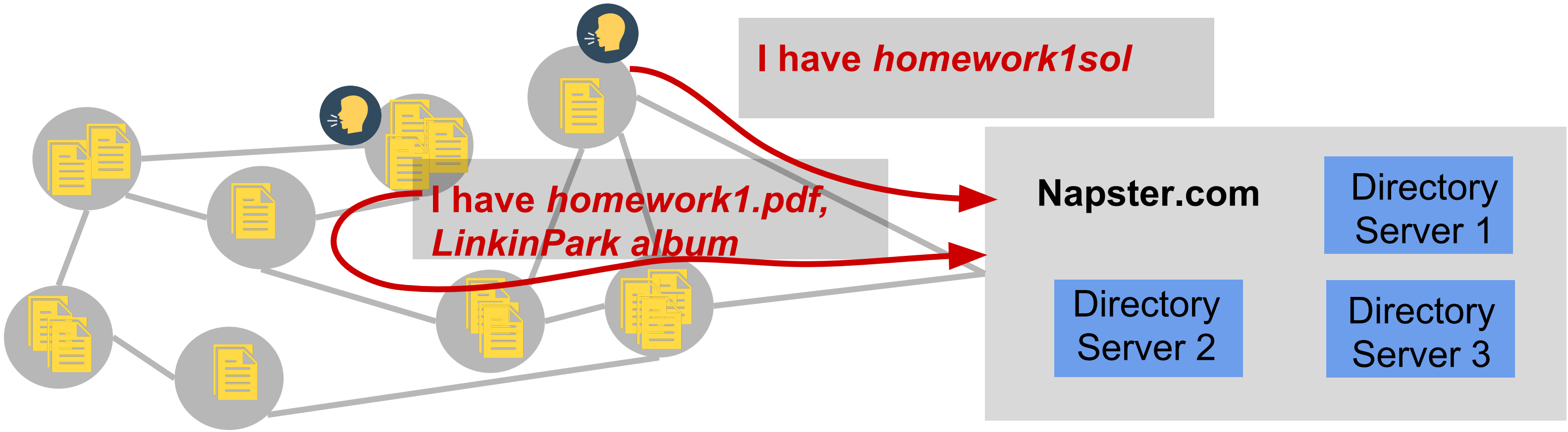
- **Reduce latency**
  - Route through next peer that reduces time to reach destination
  - Choose as successor the closest node from range $[N+2^{i-1}, N+2^{i})$
- **Accommodate heterogeneous systems**
  - Systems with different storage / bandwidth
  - Multiple virtual nodes per physical node

Roadmap

- **Structured Search: DHTs**
  - CAN, Chord
- **Unstructured search**
  - Place data somewhere, find it later somehow
  - Napster, Gnutella (brief), Freenet, BubbleStorm

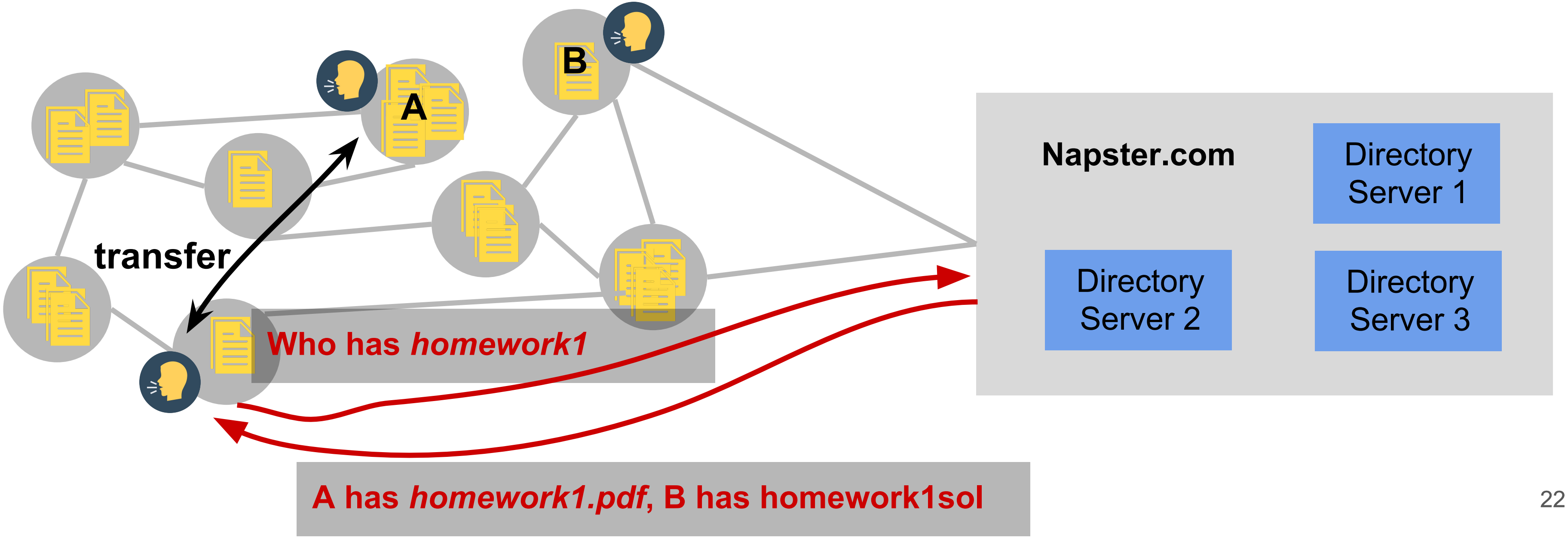
Napster: Overview

- **That's how it started**
  - Free music over the internet 1999-2001, peaked at 1.5 million users
- **Peers (home users) store files**
  - Each peer announces its local files to a centralized directory
- **Simple centralized directory**
  - Stores mapping between peers and files they store
  - Centralized means the only Napster.com has the directory, but Napster.com itself can store the directory on multiple servers



Napster: Architecture

- **File retrieval**
  - Ask directory for peers that stores files matching a pattern (ideally nearby / less loaded peers)
  - Contact the peers directly for file transfer - transfer is peer to peer
- **Peers implicitly share storage & bandwidth**

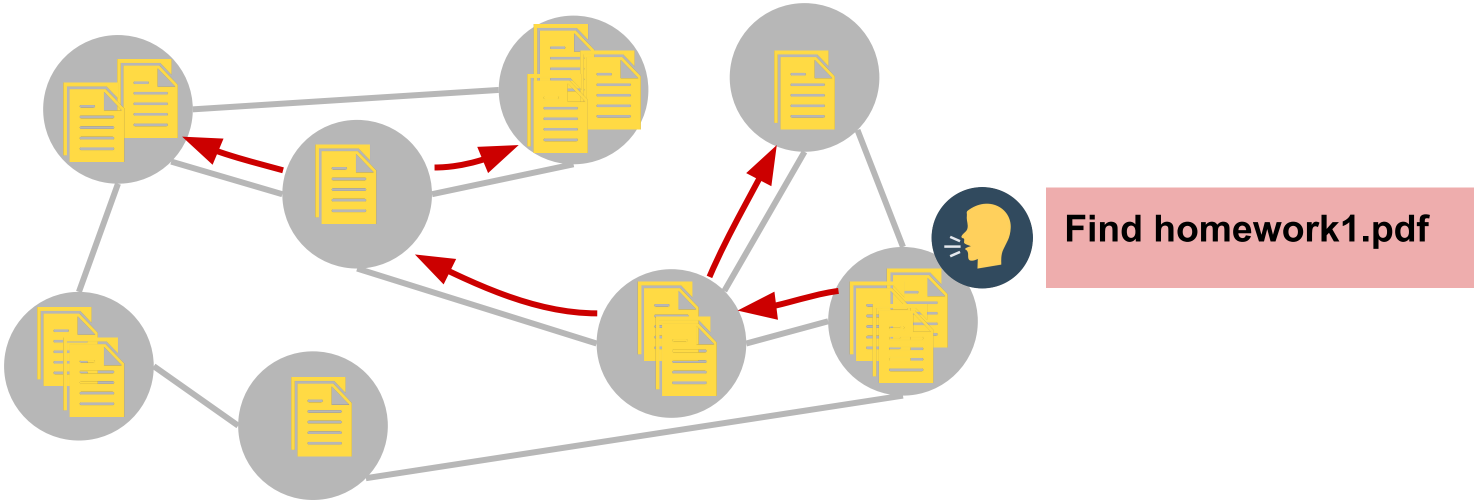


Napster: Analysis

- **Single-level filesystem**
  - Flat naming
- **Centralized directory**
  - **Advantage:** easy to build sophisticated search engines on top of index system
  - **Disadvantage**: potential bottleneck (scaling problem) & single point of failure
- **What if peers fail / go offline?**
  - Keepalive mechanism - directory servers ping them periodically
  - Any issues with keepalive? Packetstorms
- **Social, not technical**
  - Successful due to building an online community
  - Ethics built-in: tit for tat

Gnutella: main idea

- **No centralized component**
  - Ad-hoc network
- **Find a file by recursively flooding request to neighbors**
  - Nr of hops bounded using TTL



Gnutella: Analysis

- **Advantages**
  - Decentralized
  - Robust
- **Diasadvantages**
  - A single request can flood part of the network
  - No guarantee on actually finding the file (depends on TTL, start point and file location)

Freenet: Principles

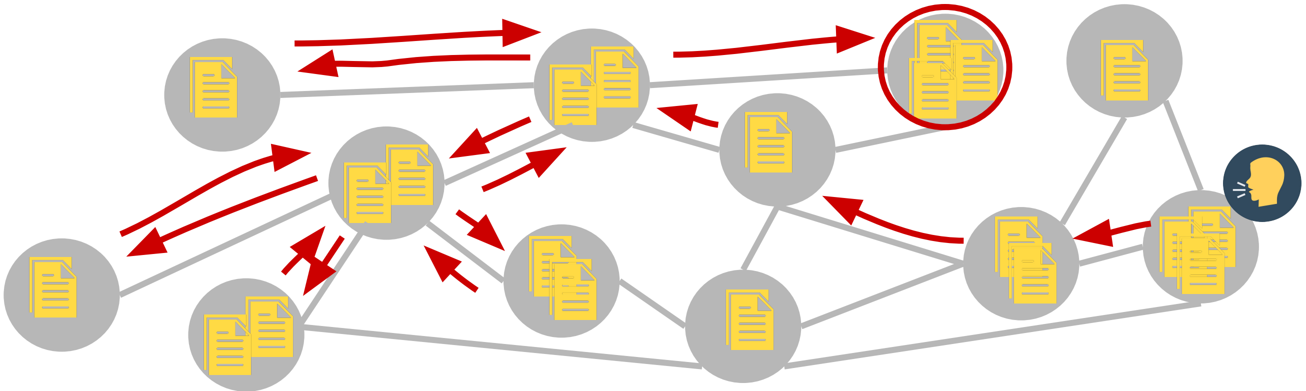
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

- **Distributed filesystem**
  - Peers insert in the system files they want to share (origin peers)
  - Origin peer does not necessarily store the files he inserts
- **Peers dynamically become specialized in storing certain files**
  - Specialization by file name
  - They are not “born specialized” as in DHTs
  - Specialization driven by routing
- **Anonymity component**
  - Files have pseudonyms
  - Peers cannot know - and cannot be held accountable for - what they store
  - File source anonymized *to some extent*

Freenet: File Search

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

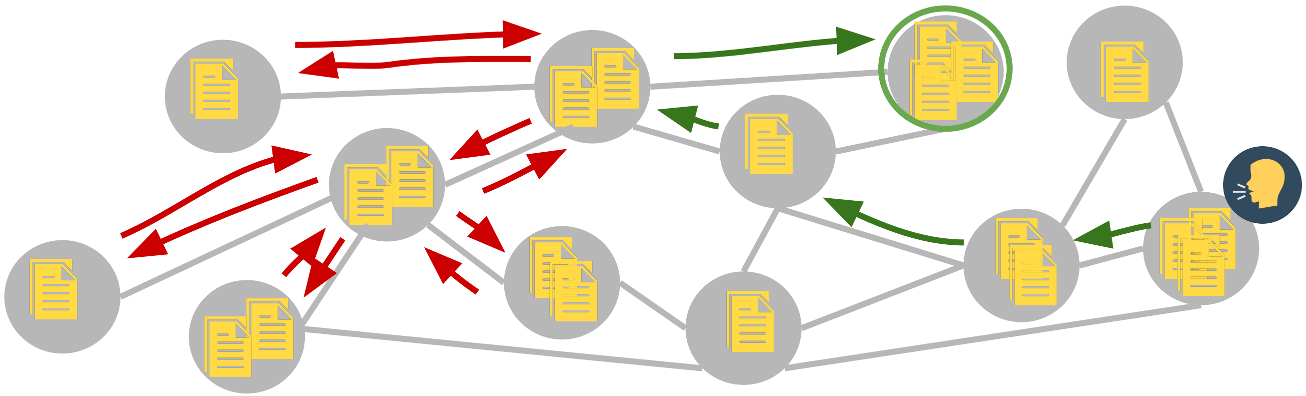
- **Each peer maintains a routing table**
  - Maps routing keys to destination peers (more on routing keys soon)
  - Request for file **ad082om34pwu: send to destination peer** whose routing key is **lexicographically closest match to searched key** (without sending back)
  - IP routing underneath: match peer to next hop
- **Response includes peer that stored the file**
- **Search can fail**
  - Hops-to-live limit. **Why?**
  - Retry by backtracking at each step, retry with bigger hops-to-live limit



Find ad082om34pwu
Hops-to-live 5

Freenet: Routing table (RT)

- **Routing tables lead to node specialization**
  - A peer P likely receives search requests for keys similar to the ones his neighbors have in their RT tables for P
  - Thus P becomes specialized in such keys
  - Failed requests should lead to adjusting RTs
- **Peers along return path following a search replicate searched file**
  - Eviction strategy: LRU
  - Consequence of replication? **Replicate files close to source. Popular files have more replicas.**



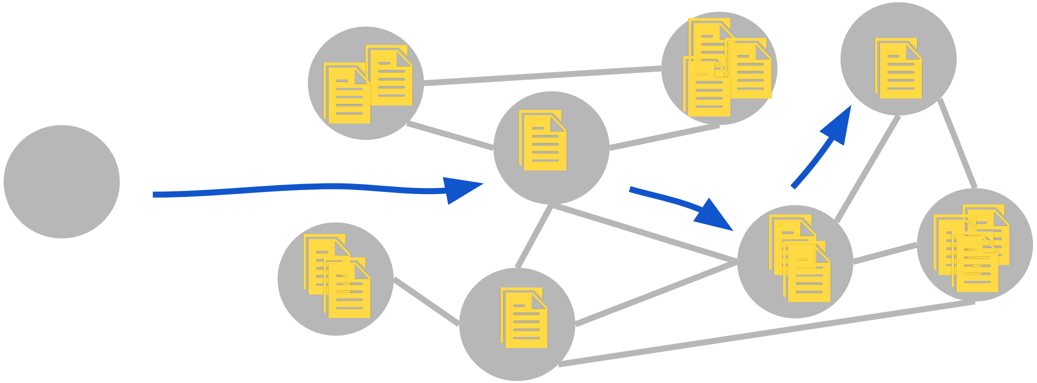
Find ad082om34pwu
Hops-to-live 5

Freenet: Storing data

- **Store file with name xyz**
  - Perform file search for xyz
  - If found, try with different file name
  - If not found, store file along forward search path
  - Peers along path associate fille origin peer with the file name
- **No file lifetime guarantee**
  - Peer can erase own routing table entries
  - Peers can erase data they store

Freenet: Node join

- **Peers that join get a routing key**
  - Routing key used in routing tables
  - For routing consistency & key clustering: routing key for a node the same in all routing tables
  - Each new peer could pick its own routing key and announce presence
  - But adversary could change it in the announcement and defeat routing table consistency
- **Routing key decided by all nodes on the announcement path**
  - Init peer picks rnd $seed_1$, computes commitment $c_1 \leftarrow$ **H($seed_1$)**, sends **<addr, c_1>** to rnd node
  - Receiving node picks rnd $seed_2$, computes $c_2 \leftarrow$ H($seed_2$ XOR c_1), sends to rnd node from RT etc
  - Nodes reveal seeds, routing_key $\leftarrow$ $seed_1$ XOR $seed_2$ etc, each node can check commitment



Freenet: Peer anonymity w.r.t files

- **Peers can replace peer addresses in inserts / replies with their own**
  - Adversary controlling single peer cannot know the real source
  - What if adversary controls multiple peers? **Traffic correlation**
- **Anonymized filenames**

Freenet: Anonymized Filenames (1)

- **Keyword-signed key (KSK)**
  - S = String that describes the file
  - Deterministic fct F, K_Pub,K_Priv ← F(S)
  - File_key ← Hash(K_Pub)
  - File signed with K_Priv, encrypted using S as encryption key. Issues?
  - Peer published S on public bulletin board
  - Keys form a flat global namespace. Issues? **Attacker inserts junk files using popular name**
- **Signed-subspace key (SSK)**
  - User creates own namespace and associates K_Pub, K_Priv
  - File_key ← Hash (Hash(K_Pub) XOR Hash(S))
  - File signed with K_Priv, encrypted using S as encryption key
  - Peer publishes S together with K_Pub on public bulletin board
  - Can an attacker insert files to someone else's namespace? **No, because he cannot sign them**

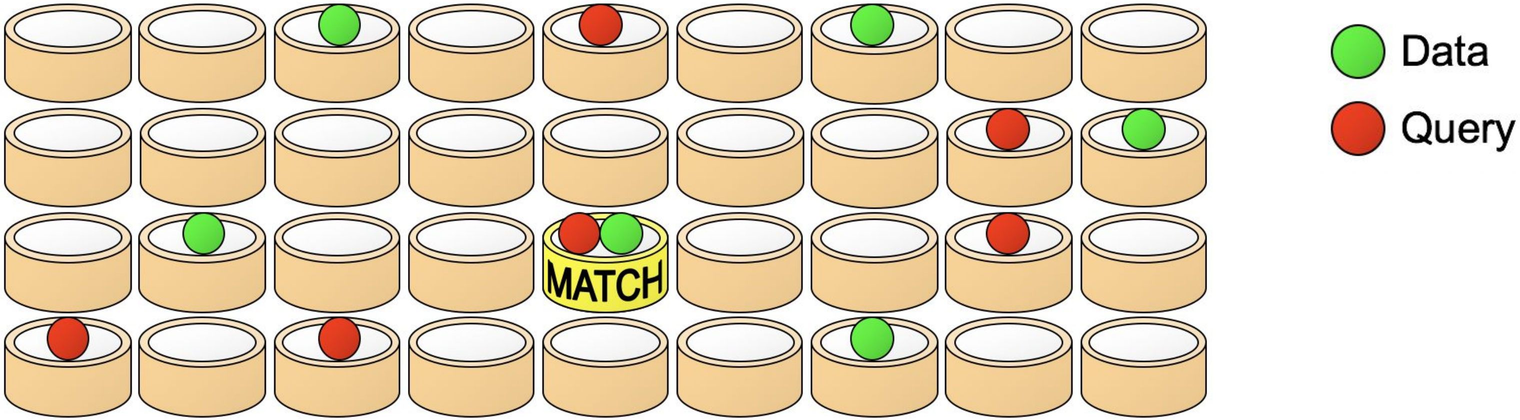
Freenet: Anonymized Filenames (1)

- **Content-hashed key (CHK)**
  - Used to implement file updates and for splitting files in chunks
  - File_key ← Hash(File)
  - Generate K_Pub,K_Priv
  - File encrypted with K_priv
  - Peer publishes File_key, K_Pub
- **File update**
  - Indirection mechanism: store CHK File_Key in a namespace using an SSK name
  - Update contents of File and replace contents of SSK name
  - SSK name stays the same!
- **File splitting**
  - Useful for large files
  - Create several chunks each with its own CHK file_kley and store these keys under a single SSK name

BubbleStorm: Intuition

- **Replicate both queries and data on random nodes**
  - $O(\sqrt{n})$ copies each
  - Higher bandwidth peers naturally receive more traffic
  - Maintain replication factor in face of churn

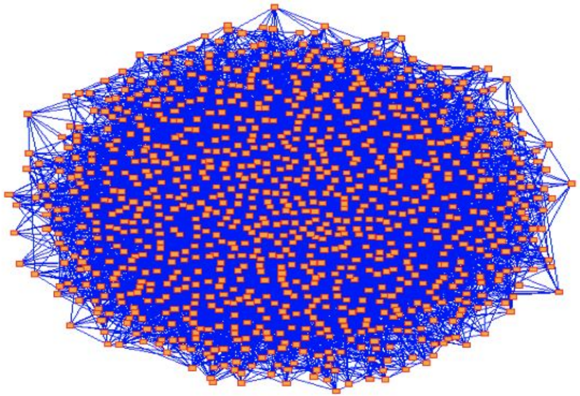
- **Data and queries intersect on some nodes**
  - Due to birthday paradox
  - Nodes evaluate queries on all their stored data



BubbleStorm: Replication on random nodes

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

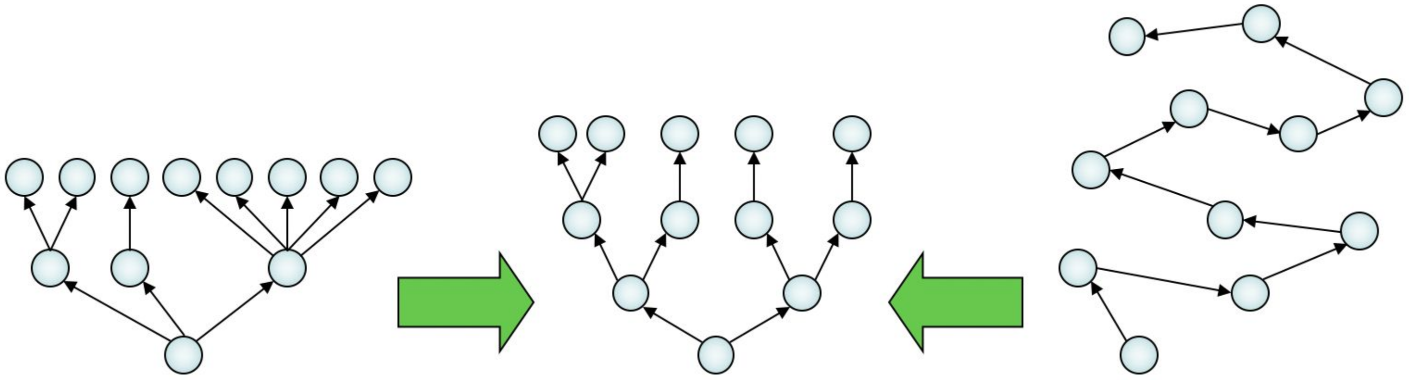
- **Arrange peers in a random graph overlay**
  - Exploring an edge leads to randomly-sampled peer
  - Creation of random subset (bubble) is cheap
- **Node degree chosen proportional to bandwidth**
  - Random walks follow edges with equal probability
  - Utilization balanced for heterogeneity
- **Topology modified only when nodes join / leave**
  - Neighbors' degree remains unchanged



BubbleStorm: Node joins and leaves

- **Node join: inserts itself on random edge**
  - Random walk from bootstrap peer
  - Pick random edge to split and insert in between
- **Node leave**
  - Splices neighboring peers together

BubbleStorm: Fast replication (1)



Flooding

- Low latency
- Reliable
- Imprecise #replicas
- Unbalanced link load

BubbleCast

- **Low latency**
- **Reliable**
- **Precise #replicas**
- **Balanced link load**

Random walk

- High latency
- Unreliable
- Precise #replicas
- Balanced link load

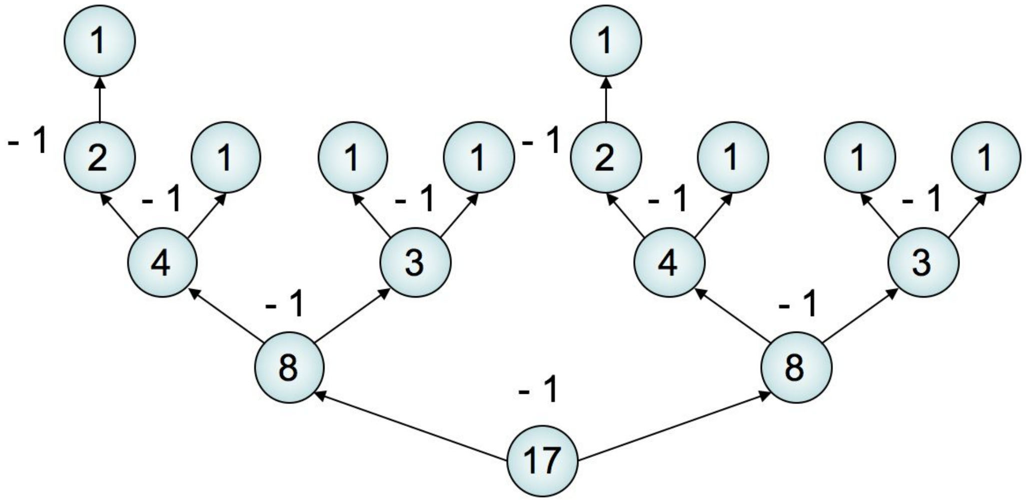
Node counter instead of hop counter

Branch at every step

BubbleStorm: Fast Replication (2)

- **Counter specifies #replicas to create**
  a. Create replicate
  b. Split updated counter equally between neighbors

- **Replication depth between branches differs by at most 1**
  a. Logarithmic routing depth



Unstructured vs structured search

- **Unstructured search: support any selective query**
  - Peers simply deploy a query language locally
- **DHTs: Perform multiple lookups**
  - Must transform query into multiple key-value parts
  - Higher latency due to executing each part separately

Conclusions

- **Search in peer-to-peer systems**
  - Churn
  - Replication
  - Routing
- **Structured search: DHTs**
  - Key-value store primitive
  - Peers and items given random id in id space
  - Predefined mapping of keys to values
  - CAN, Chord
- **Unstructured search**
  - Support more variate query primitives
  - Freenet: Dynamic specialization of peers
  - BubbleStorm: random replication of data and queries, probabilistic intersection on same node