Arpeggio:

Metadata Searching and Content Sharing in Structured Peer-to-Peer Networks

Dan R. K. Ports

drkp@mit.edu

MIT Computer Science and Artificial Intelligence Laboratory

Joint work with Austin T. Clements and David R. Karger

CSAIL

Outline

- **Overview**
- Searching
- Index Gateways
- Content Distribution
- Conclusion

The Content-Sharing Problem

- Goals
  - Find files matching a search query
  - Identify sources for a file
  - Want full decentralization
- Assumptions
  - Only searching *metadata*
  - Metadata is small (compared to actual data)
  - Highly dynamic and unstable network topology, content, and sources

CSAIL

P2P Evolution

- Centralized index systems (Napster)
  - Content shared peer-to-peer, but index centralized
  - Scale poorly, single point of failure
- Unstructured overlay networks (Gnutella)
  - Queries by flooding or random walks
  - Large network impact
  - Sacrifices perfect recall

CSAIL

DHTs to the Rescue?

- Useful building block: *distributed hash tables* (DHTs)
  - e.g. **Chord**, Pastry, Tapestry, Kademlia, ...
- LOOKUP abstraction: locate peer responsible for key
- GET-BLOCK/PUT-BLOCK hash table operations

- Efficient: $O(\log n)$ messages per lookup
- Scalable: $O(\log n)$ routing state per node
- Robust: maintains correctness despite node joins & failures

CSAIL

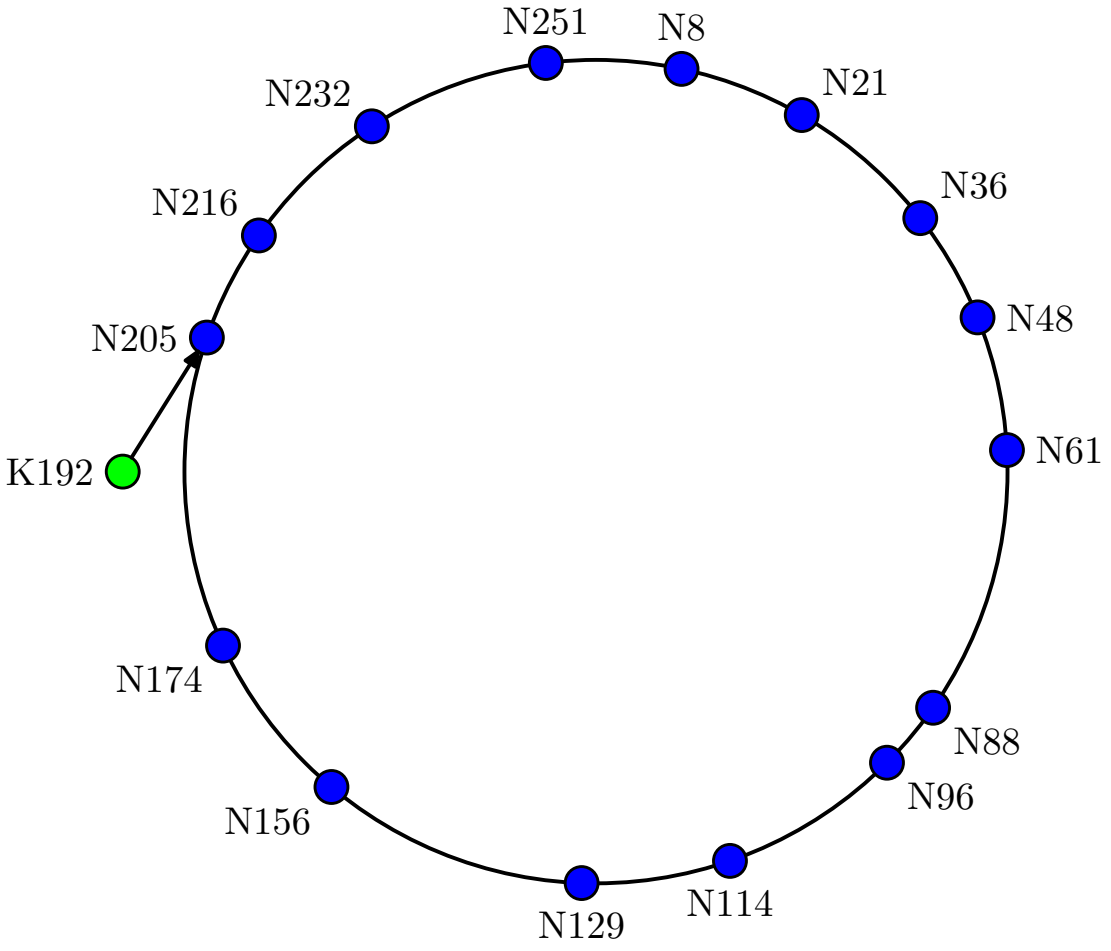
Building a Distributed Hash Table

- **Consistent hashing:** which node stores an object?

CSAIL

Building a Distributed Hash Table

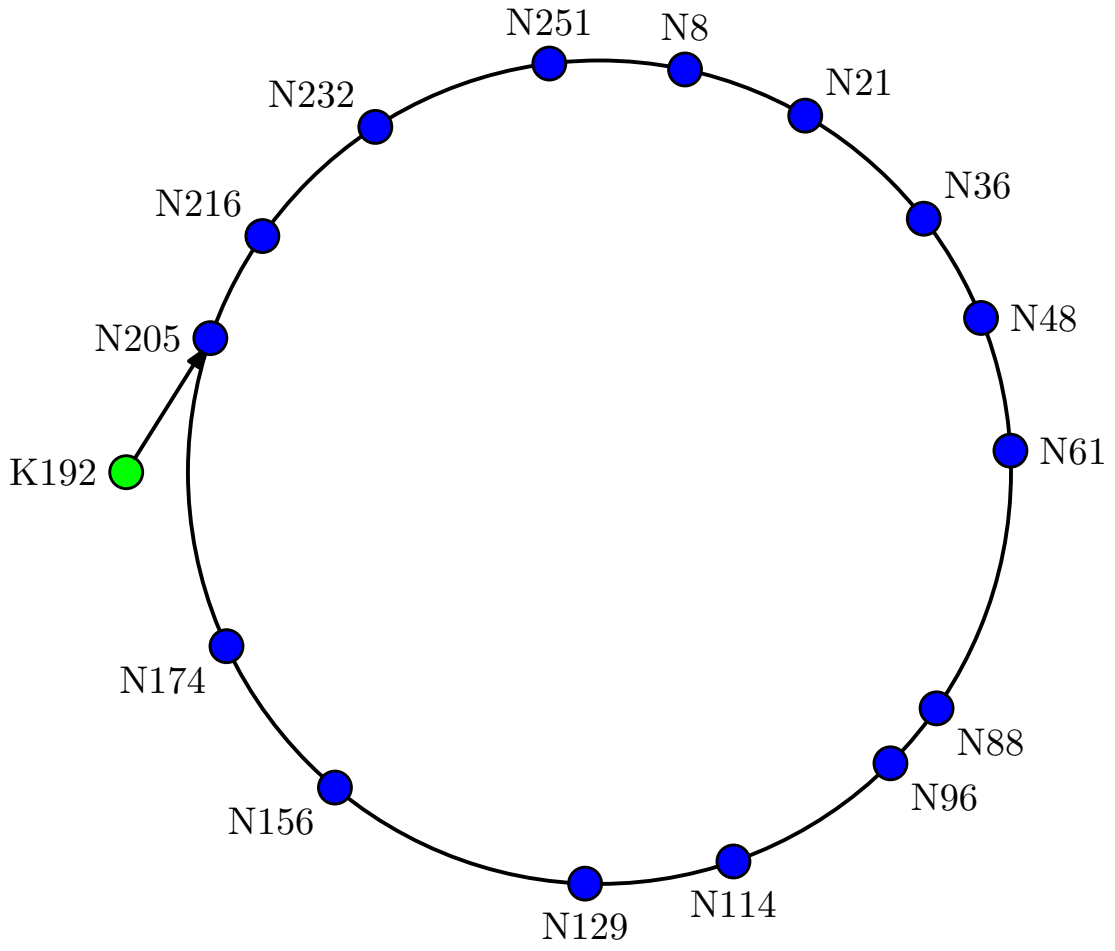
- Map nodes and objects to same circular keyspace



CSAIL

Building a Distributed Hash Table

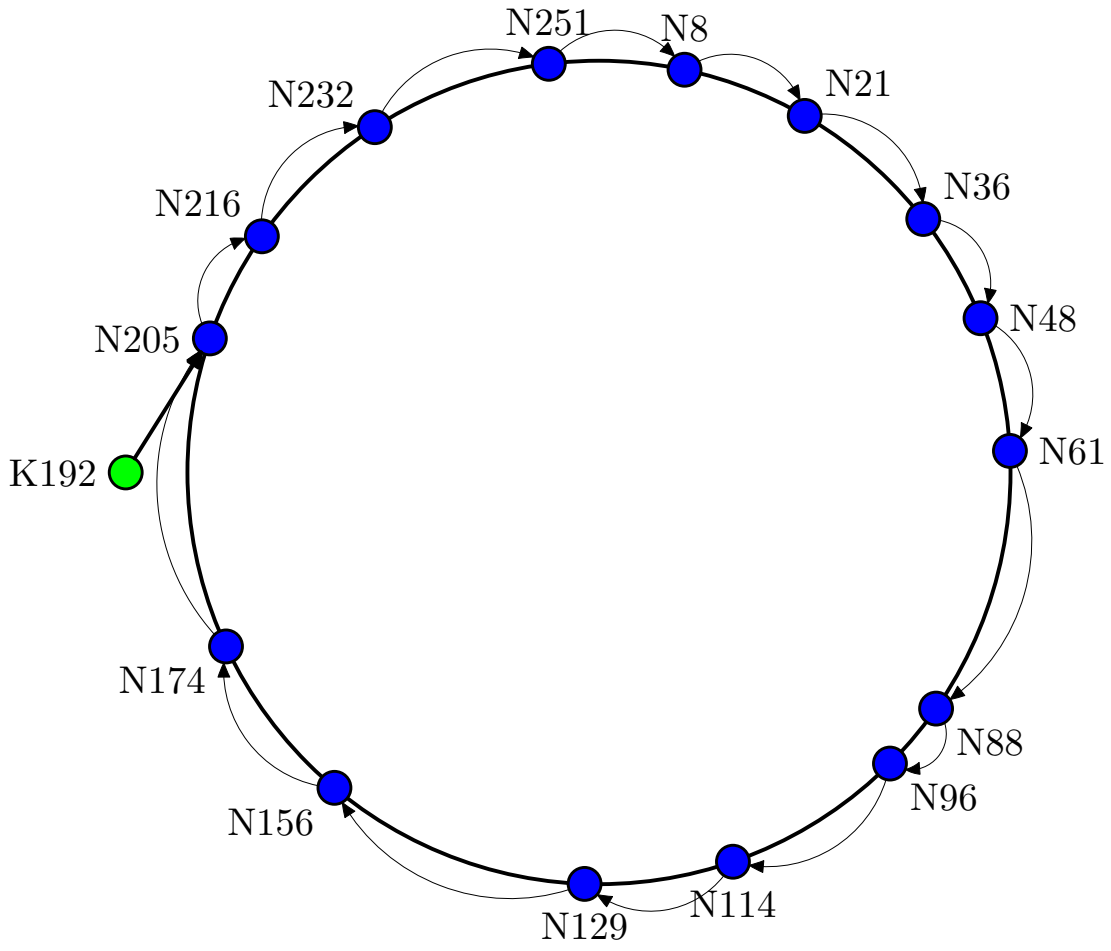
- Successor node is responsible for each key



CSAIL

Building a Distributed Hash Table

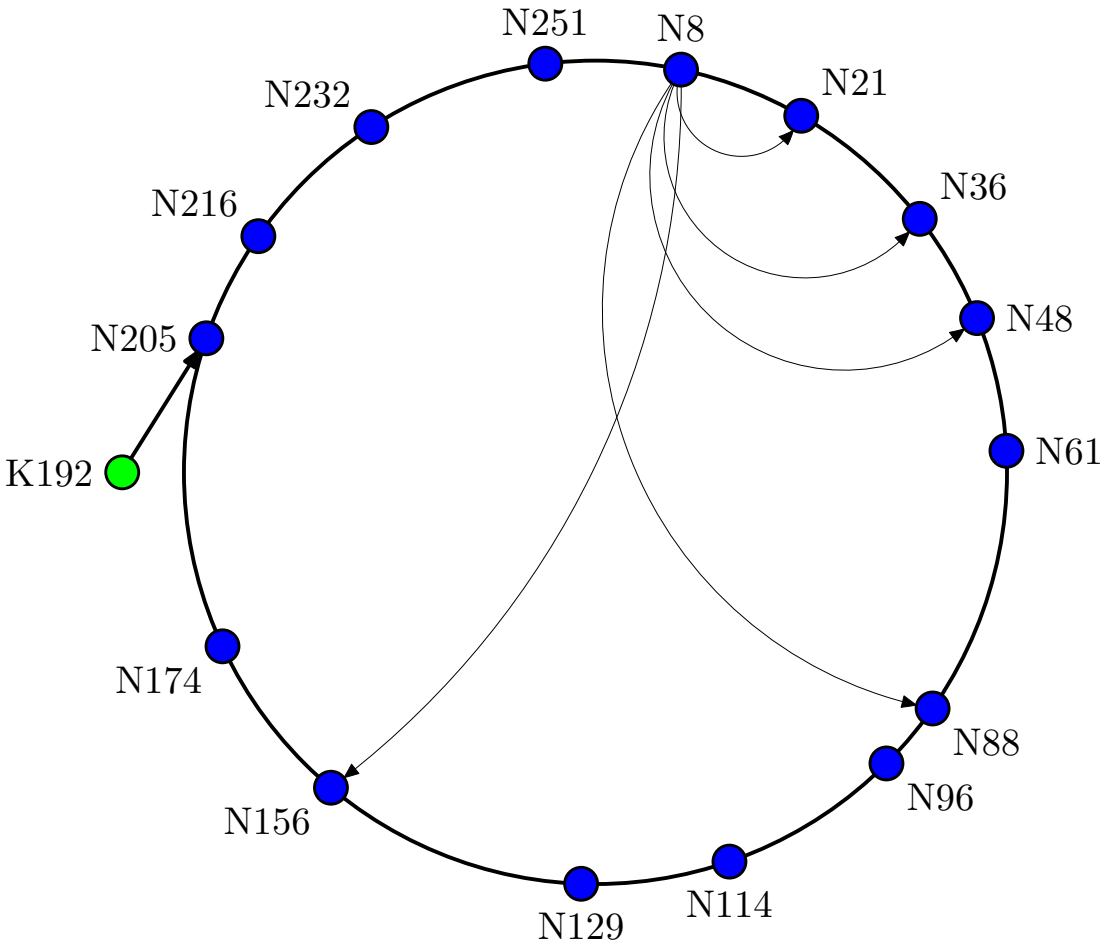
- Pointers to successors — slow routing possible



CSAIL

Building a Distributed Hash Table

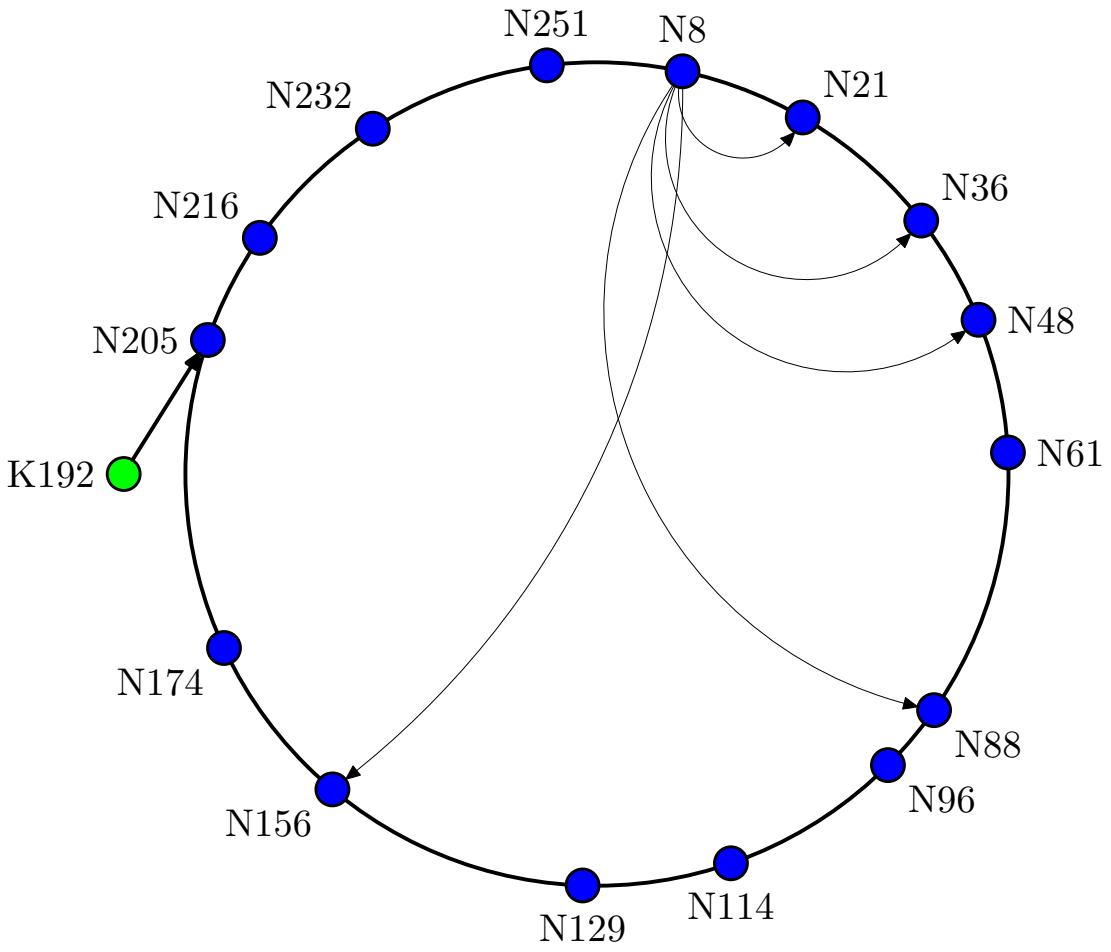
- Exponentially spaced *fingers* — $O(\log n)$ hop routing



CSAIL

Building a Distributed Hash Table

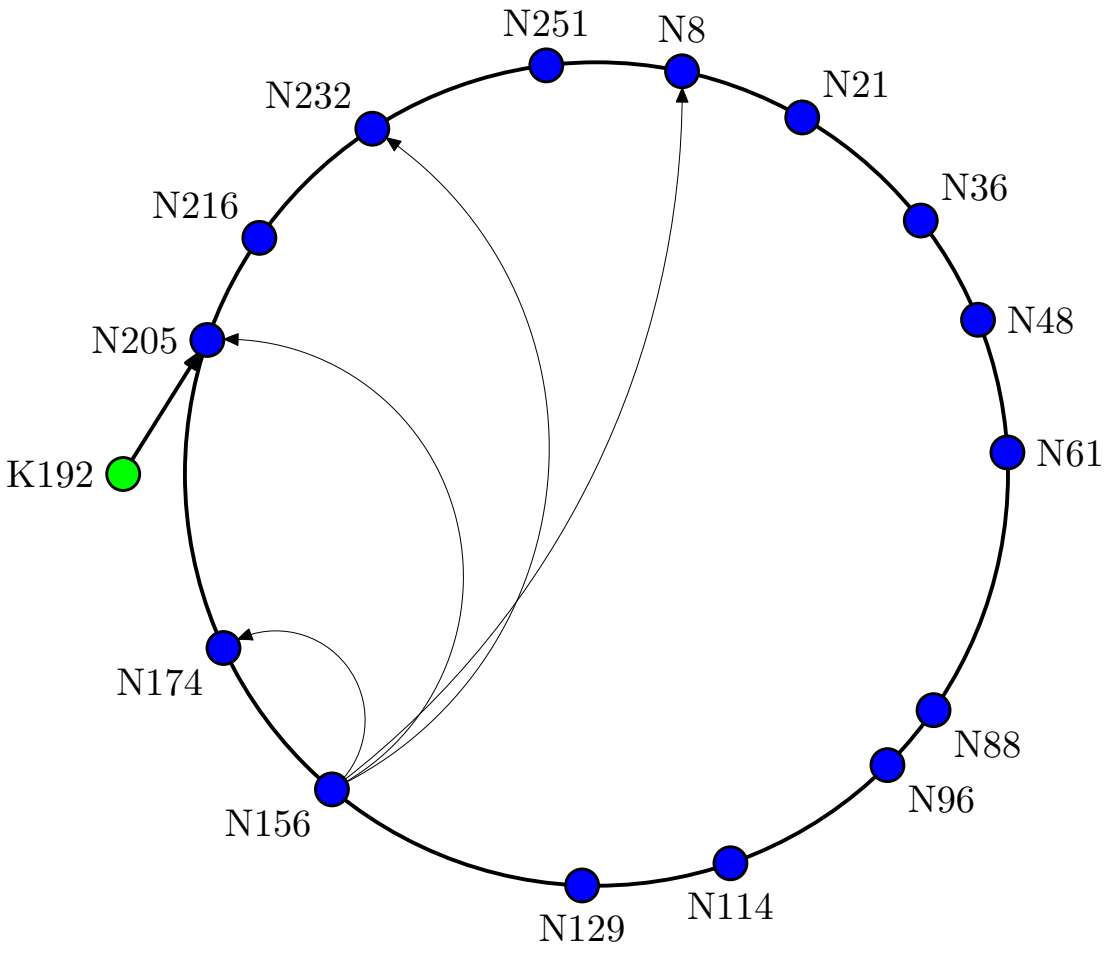
- N8: LOOKUP(K192) — N156 is closest finger



CSAIL

Building a Distributed Hash Table

- N156 knows N205 is responsible, forwards message



CSAIL

DHTs for File Sharing?

- Can we just store everything in a DHT?
  - DHT is great for *lookup by name*,
    but usually we don't know the filename!
  - Need complex *search by keyword* queries instead

- **Arpeggio:** indexing based on DHT techniques
  - Uses Chord LOOKUP primitive
  - Maintains a *distributed keyword-set index*
  - Indirect storage of file data

CSAIL

Outline

- Overview
- **Searching**
- Index Gateways
- Content Distribution
- Conclusion

CSAIL

Index Entries

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f39b1940ed5...
size	586MB
type	application/x-iso9660-image
⋮	

CSAIL

Distributed Inverted Indexing

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

CSAIL

Distributed Inverted Indexing

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

CSAIL

Distributed Inverted Indexing

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

CSAIL

Distributed Inverted Indexing

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

(disk1, freebsd, iso)	
name	Freebsd Disk1.iso
file ID	fbcbfdff31f27de...

CSAIL

Distributed Inverted Indexing

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

(disk1, freebsd, iso)	
name	Freebsd Disk1.iso
file ID	fbcbfdff31f27de...

CSAIL

Distributed Inverted Indexing

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

(disk1, freebsd, iso)	
name	Freebsd Disk1.iso
file ID	fbcbfdff31f27de...

CSAIL

Distributed Inverted Indexing

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

(disk1, freebsd, iso)	
name	Freebsd Disk1.iso
file ID	fbcbfdff31f27de...

Problem: Network traffic for large indexes

CSAIL

Index-Side Filtering

- Keywords are small, so store keywords in index
- Pick one index node
- Send full query
- Index node performs filtering and returns only relevant results
- Can also include other *filterable metadata*, e.g. file size, MP3 bitrate, etc.

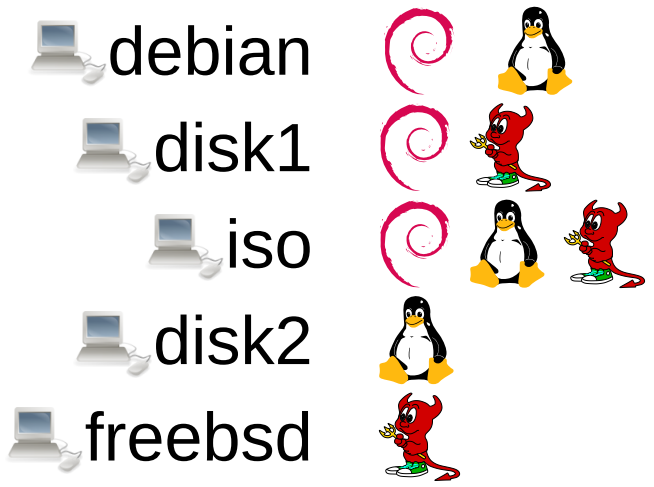
CSAIL

Index-Side Filtering

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

(disk1, freebsd, iso)	
name	Freebsd Disk1.iso
file ID	fbcbfdff31f27de...



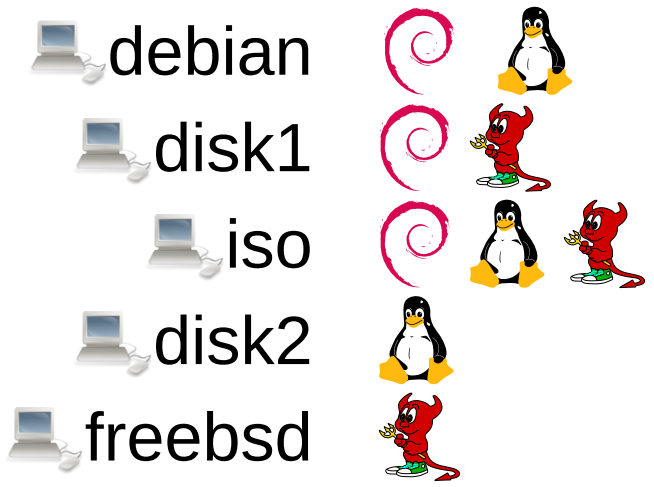
CSAIL

Index-Side Filtering

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

(disk1, freebsd, iso)	
name	Freebsd Disk1.iso
file ID	fbcbfdff31f27de...



“debian disk1?”

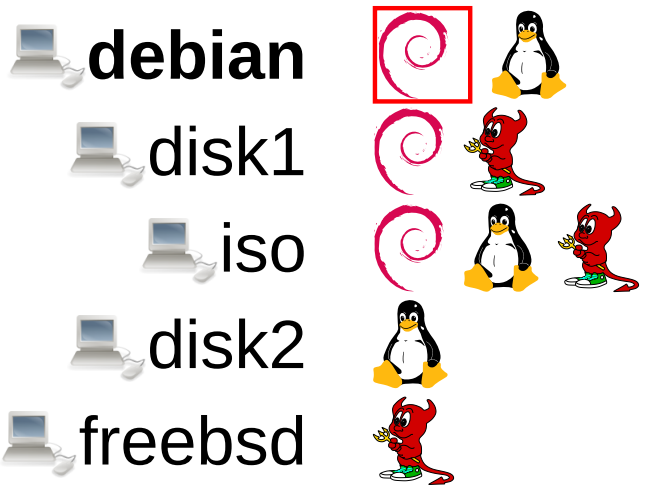
CSAIL

Index-Side Filtering

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

(disk1, freebsd, iso)	
name	Freebsd Disk1.iso
file ID	fbcbfdff31f27de...



“debian disk1?”

{ }

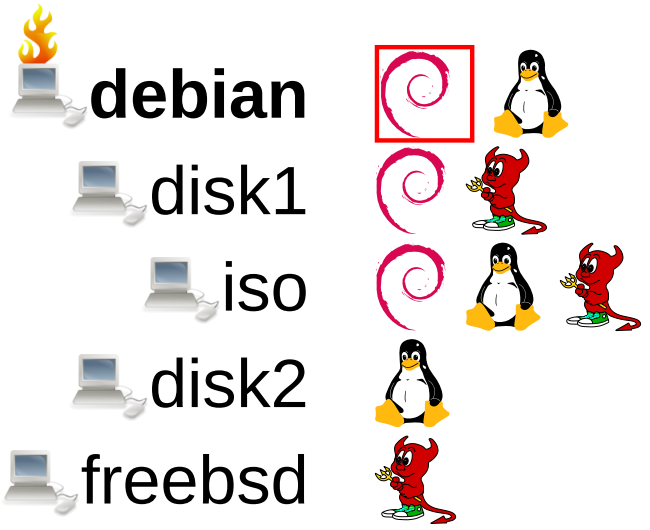
CSAIL

Index-Side Filtering

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

(disk1, freebsd, iso)	
name	Freebsd Disk1.iso
file ID	fbcbfdff31f27de...




“debian disk1?”

{ }

Problem: Poor query load-balancing

CSAIL

Keyword-Set Indexing

- Build index on *keyword sets* rather than keywords
- Store subsets of size $\leq K$
- More keyword-set indexes, but each is shorter
- Single-keyword indexes are less important, so can be truncated
  - < 29% of web searches have only 1 keyword. [Reynolds & Vahdat 2003]
- To search: send filtered query to any K-size subset index

CSAIL

Keyword-Set Indexing

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

$(K = 2)$

CSAIL

Keyword-Set Indexing

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

$(K = 2)$

CSAIL

Keyword-Set Indexing

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

(disk1, freebsd, iso)	
name	Freebsd Disk1.iso
file ID	fbcbfdff31f27de...

$(K = 2)$

debian

disk1

iso

debian disk1

debian iso

disk1 iso

disk2

debian disk2

disk2 iso

⋮

Keyword-Set Indexing

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

(disk1, freebsd, iso)	
name	Freebsd Disk1.iso
file ID	fbcbfdff31f27de...

$(K = 2)$

debian

disk1

iso

debian disk1

debian iso

disk1 iso

disk2

debian disk2

disk2 iso

⋮

“debian disk1?”

CSAIL

Keyword-Set Indexing

(debian, disk1, iso)	
name	Debian Disk1.iso
file ID	cdb79ca3db1f3...

(debian, disk2, iso)	
name	Debian Disk2.iso
file ID	5ccbf54d7e502...

(disk1, freebsd, iso)	
name	Freebsd Disk1.iso
file ID	fbcbfdff31f27de...

$(K = 2)$

debian

disk1

iso

debian disk1

debian iso

disk1 iso

disk2

debian disk2

disk2 iso

⋮

“debian disk1?”

CSAIL

Indexing Cost

$m =$ metadata keywords

$K =$ maximum subset size parameter

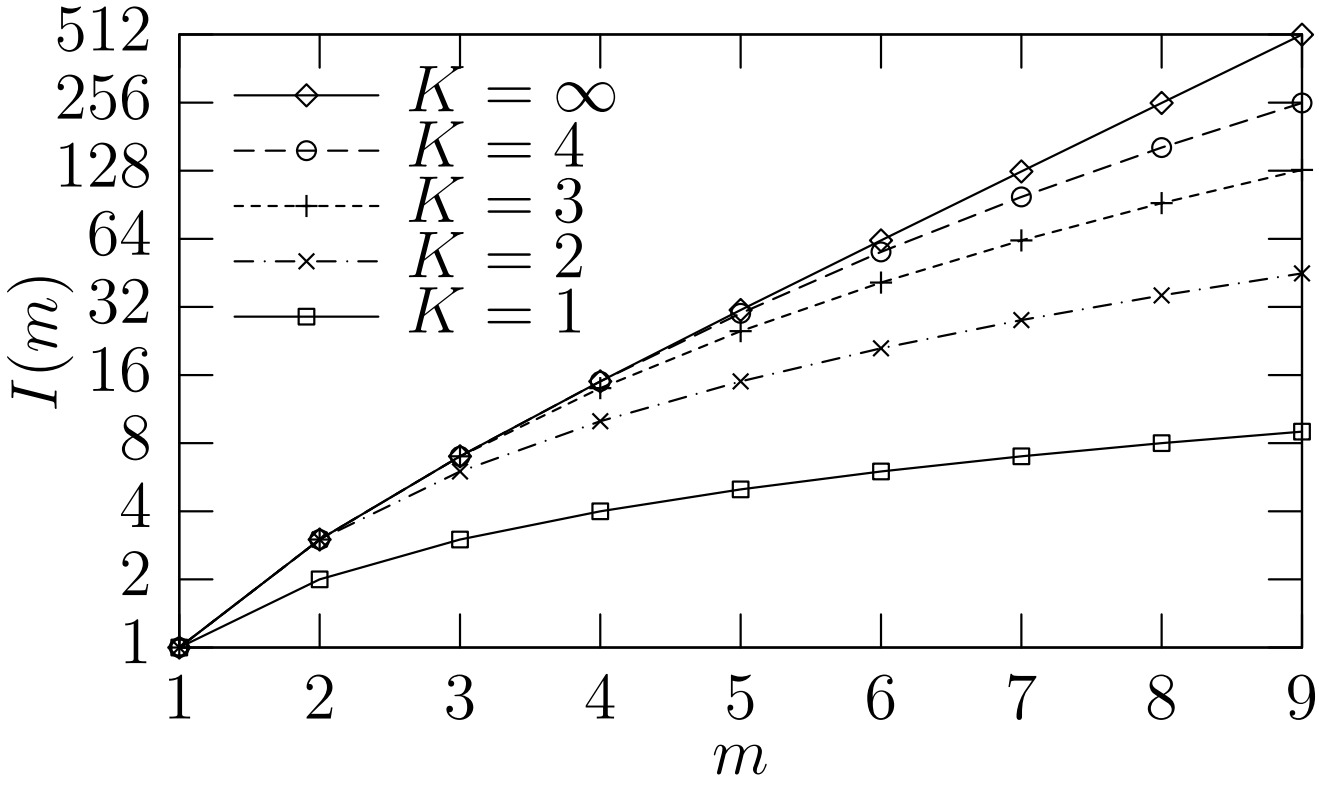
$I(m) =$ index entries

$$= \sum_{i=1}^{K} \binom{m}{i} = \begin{cases} 2^m - 1 & \text{if } m \leq K \\ O(m^K) & \text{if } m > K \end{cases}$$

CSAIL

Indexing Cost

$$I(m) = \sum_{i=1}^{K} \binom{m}{i} = \begin{cases} 2^m - 1 & \text{if } m \leq K \\ O(m^K) & \text{if } m > K \end{cases}$$



CSAIL

Indexing Cost

$$I(m) = \sum_{i=1}^{K} \binom{m}{i} = \begin{cases} 2^m - 1 & \text{if } m \leq K \\ O(m^K) & \text{if } m > K \end{cases}$$

For files with many metadata keywords, $I(m)$ is polynomial in m.

CSAIL

Storage Costs (FreeDB)

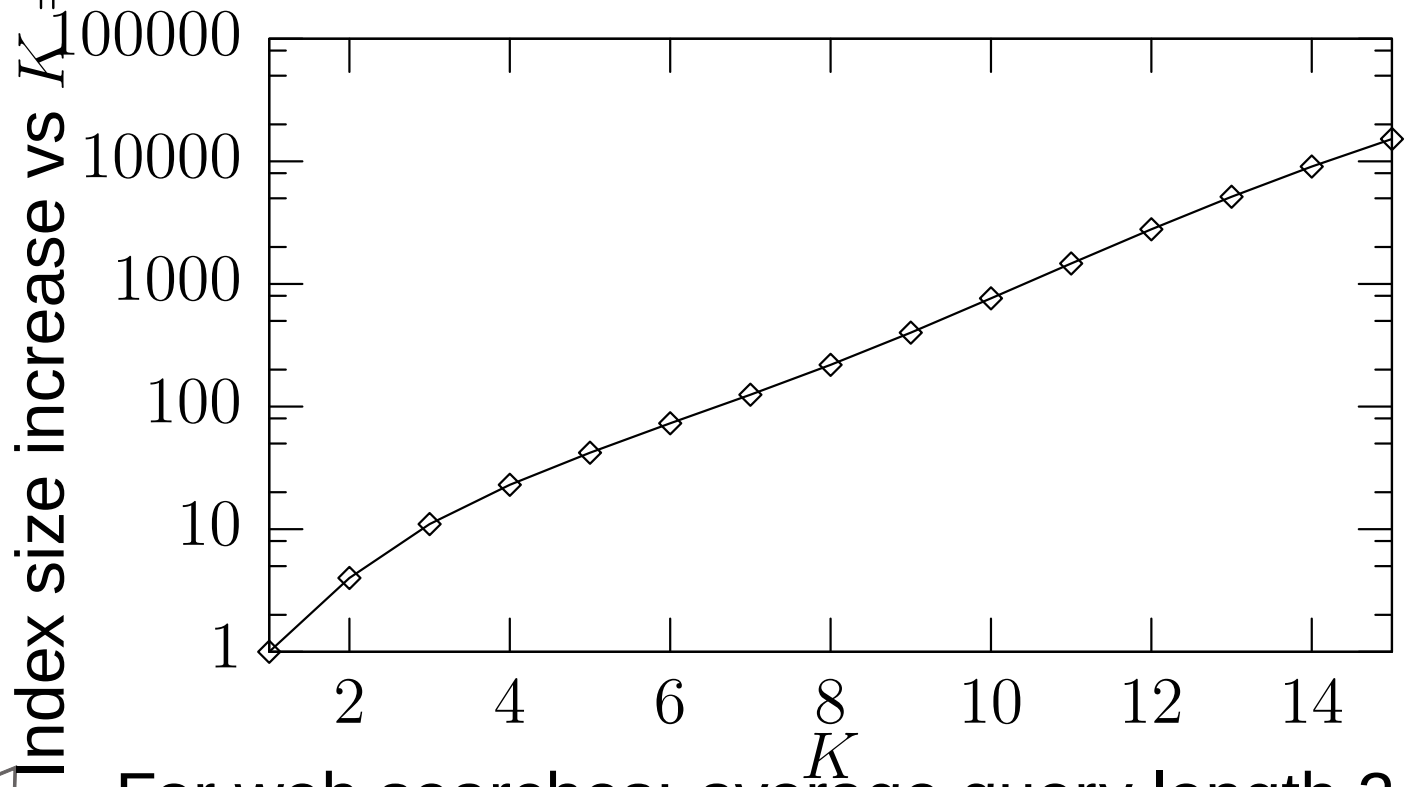
Number of songs	21,195,244
Total index entries ($K = 1$)	134,403,379
Index entries per song ($K = 1$)	6.274406
Total index entries ($K = 3$)	1,494,688,373
Index entries per song ($K = 3$)	66.078093

$\Rightarrow$ Total storage cost only an order of magnitude more than required for $K = 1$ inverted index.

CSAIL

Choosing K

- Larger K improves query load distribution, increases indexing costs



For web searches: average query length 2.53

CSAIL

Index Maintenance

- Expiration
  - File availability constantly changes as peers join/leave
  - Expiration rather than polling
- Replication
  - Replication instead of erasure coding
  - Only weak consistency required
  - Can periodically propagate updates in batch

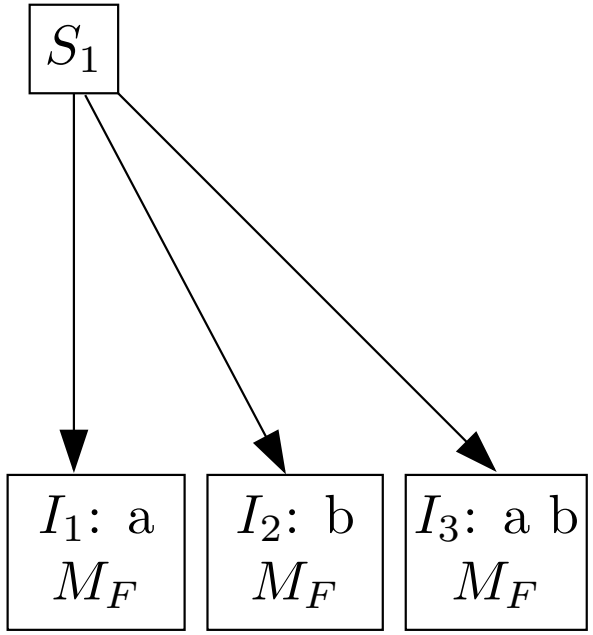
CSAIL

Outline

- Overview
- Searching
- **Index Gateways**
- Content Distribution
- Conclusion

CSAIL

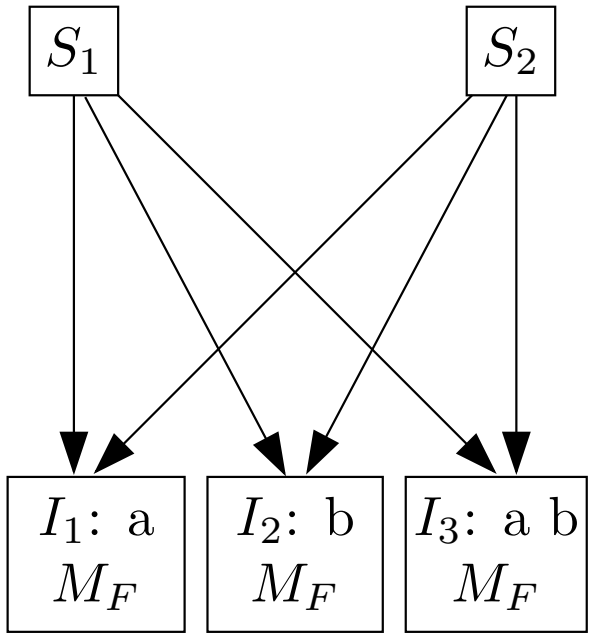
Index Gateways



- Each file has one metadata block, stored in $I(m)$ indexes.

CSAIL

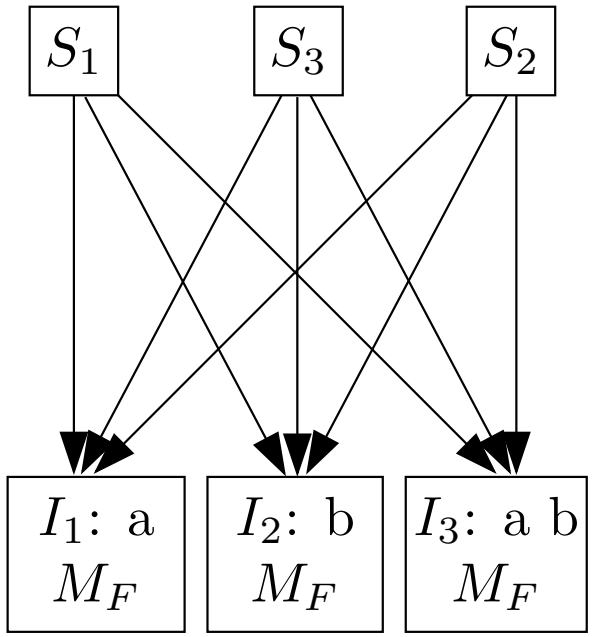
Index Gateways



- Each peer sharing the file will insert the *same* metadata block into each index.

CSAIL

Index Gateways



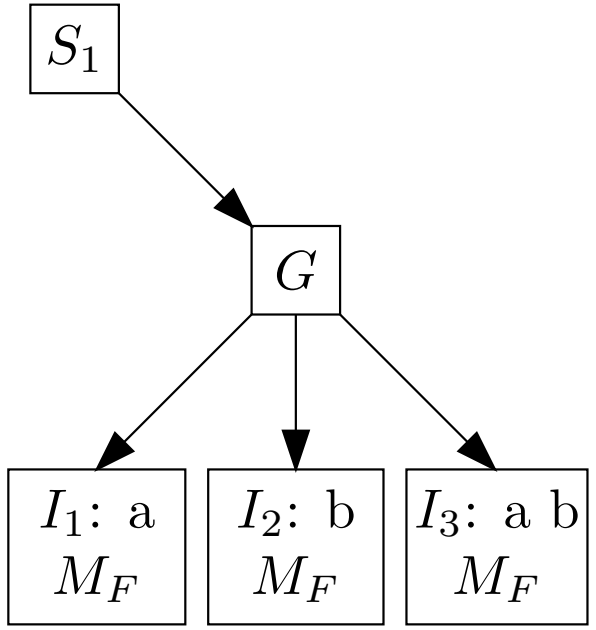
- Total insertion cost for m metadata keywords and s source peers: $sI(m)$ messages.

Problem: expensive and redundant

CSAIL

Index Gateways

- Solution: aggregate updates at an *index gateway*
- Receives metadata blocks from sources and sends to indexes only when necessary

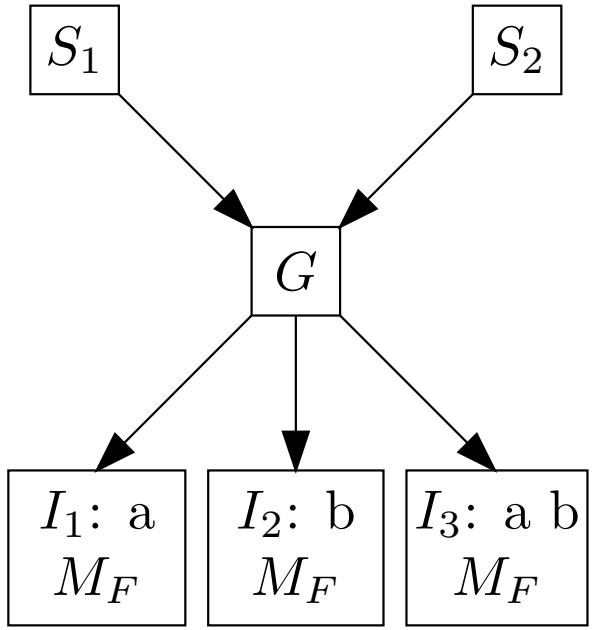


Insertion cost is now $s + I(m)$ (vs. $sI(m)$)!

CSAIL

Index Gateways

- Solution: aggregate updates at an *index gateway*
- Receives metadata blocks from sources and sends to indexes only when necessary

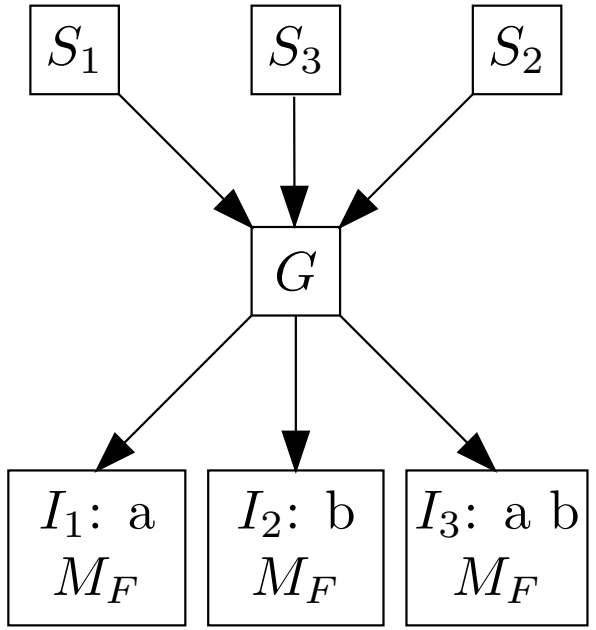


Insertion cost is now $s + I(m)$ (vs. $sI(m)$)!

CSAIL

Index Gateways

- Solution: aggregate updates at an *index gateway*
- Receives metadata blocks from sources and sends to indexes only when necessary



Insertion cost is now $s + I(m)$ (vs. $sI(m)$)!

CSAIL

Outline

- Overview
- Searching
- Index Gateways
- **Content Distribution**
- Conclusion

CSAIL

Direct Storage?

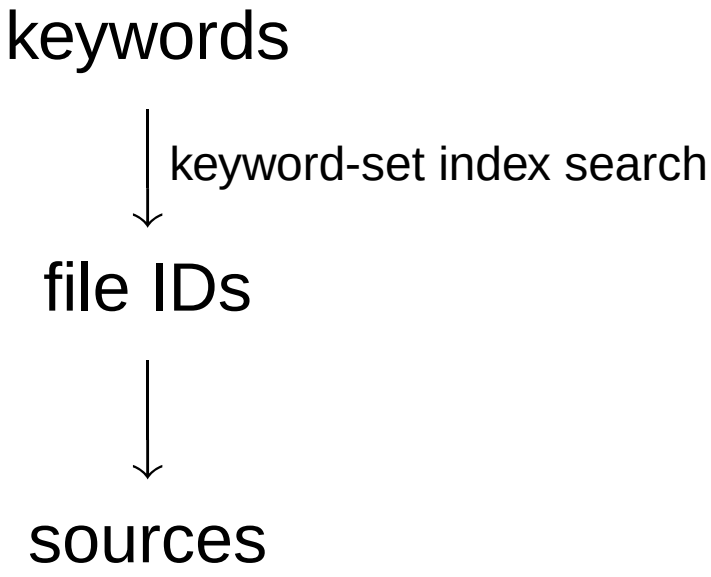
- Content is large
- Network has churn
  - Kazaa median session length 2.4 minutes [Gummadi et al. 2003]

Problem: DHT storage of content is impractical

CSAIL

Indirect Storage

- Add indirection
- Store only small pointers in the network



CSAIL

Content-Sharing Subrings

- How to identify sources for a file?
- Simple solution: “tracker node”
- Instead, file sources form subrings
  - To find source, LOOKUP random ID in file’s subring
  - Search and maintenance costs same as with tracker, but distributed over network
  - No single point of failure

CSAIL

Postfetching

- Indirect storage leads to unavailable content
- *Postfetching:* actively increase the availability of rare but popular files
  - Delay metadata expiration to track request for unavailable content
  - Requesting node inserts *request block*
  - Source node later rejoins and locates request block
  - Content replicated on caches on random nodes

CSAIL

Outline

- Overview
- Searching
- Index Gateways
- Content Distribution
- **Conclusion**

Conclusion

- Supports search with distributed keyword-set index
- Extends standard DHT interface with network-side processing
  - Index-side filtering
  - Index gateways
- Content distribution with indirect storage
  - Indexing via subrings
  - Postfetching to increase availability

CSAIL

Conclusion

- Supports search with distributed keyword-set index
- Extends standard DHT interface with network-side processing
  - Index-side filtering
  - Index gateways
- Content distribution with indirect storage
  - Indexing via subrings
  - Postfetching to increase availability

- **Questions?**

CSAIL


CSAIL

Bonus Section

Bonus 1 – Segmentation

- At this level, content is atomic
- Can’t share partially downloaded content

Problem: Doesn’t utilize upload bandwidth

CSAIL

Bonus 1 – Segmentation

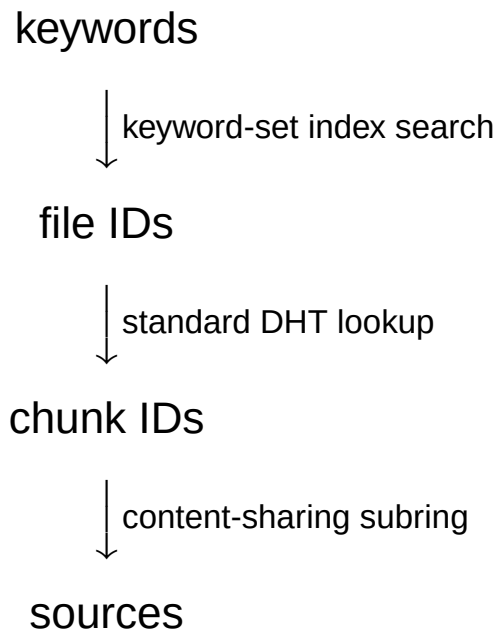
- Split content into chunks and share on the chunk level
- Typical file sharing networks contain many similar files

Problem: Underutilization of content sources

CSAIL

Bonus 1 – Segmentation

- Place chunk boundaries based on content data
- Identify chunks by hash of their contents
- Files are now just a list of their constituent chunk IDs



CSAIL