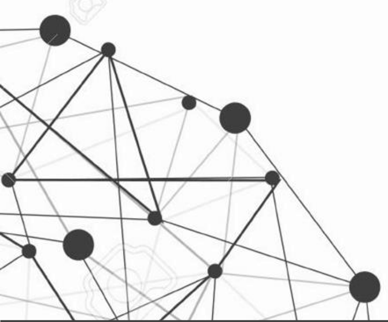
DedupSearch: Two-Phase Deduplication Aware Keyword Search

Nadav Elias (Computer Science, Technion)

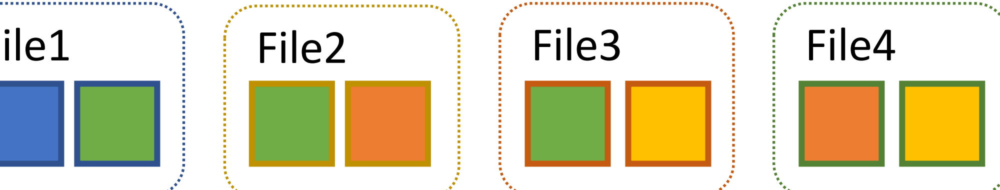
Philip Shilane (Dell Technologies)

Sarai Sheinvald (ORT Braude College of Engineering)

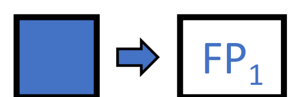
Gala Yadgar (Computer Science, Technion)

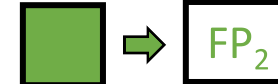


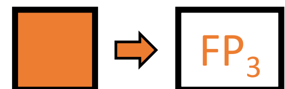
Reminder: Deduplication

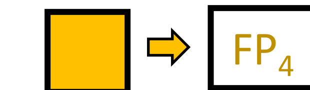


- Fingerprints
  - Cryptographic hash of block content
  - Low collision probability

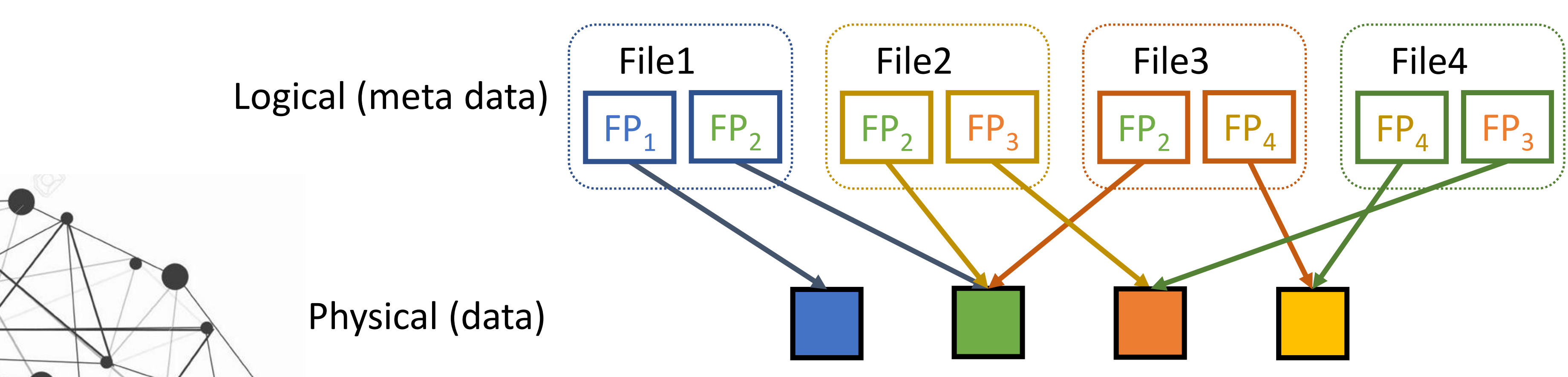




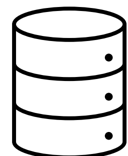




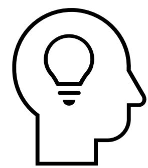
- File recipe stores the list of fingerprints of a file



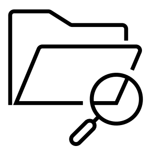
Motivation



Deduplication is everywhere



Rethink storage functions

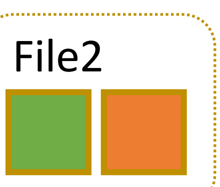


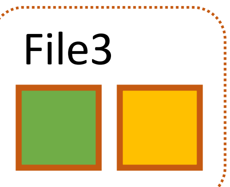
Specifically, a search for a byte string

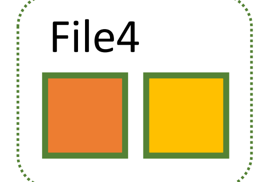
- Find a document containing particular terms
- Machine learning preprocessing
- Offline: no index

Naïve Search

File1

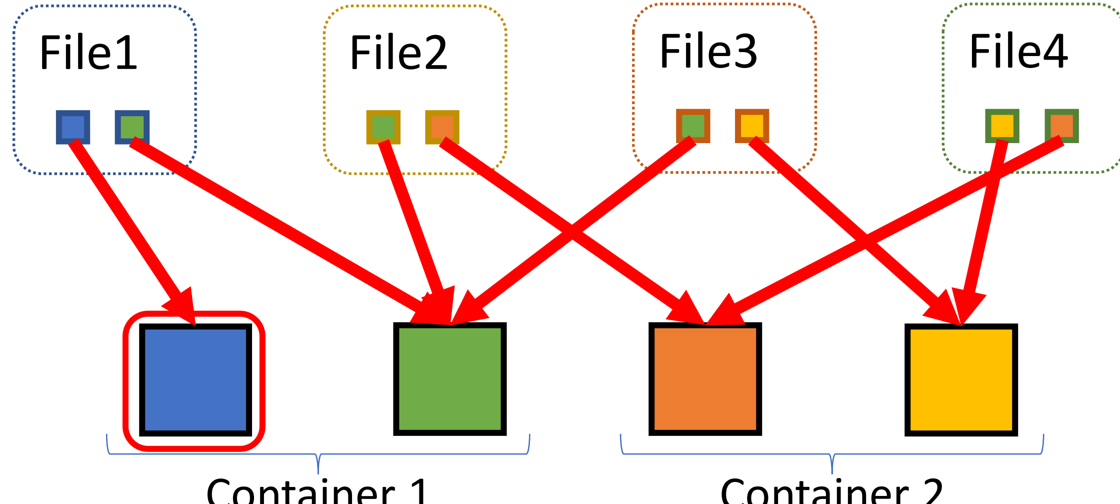


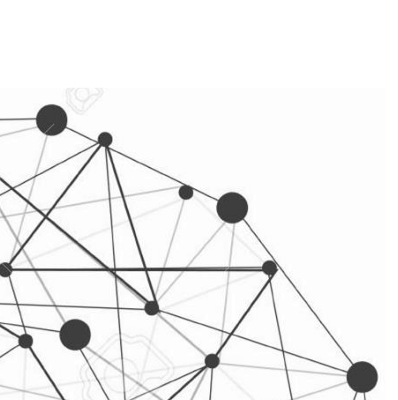




- Scan each files' chunks in order
  - 8 chunk reads

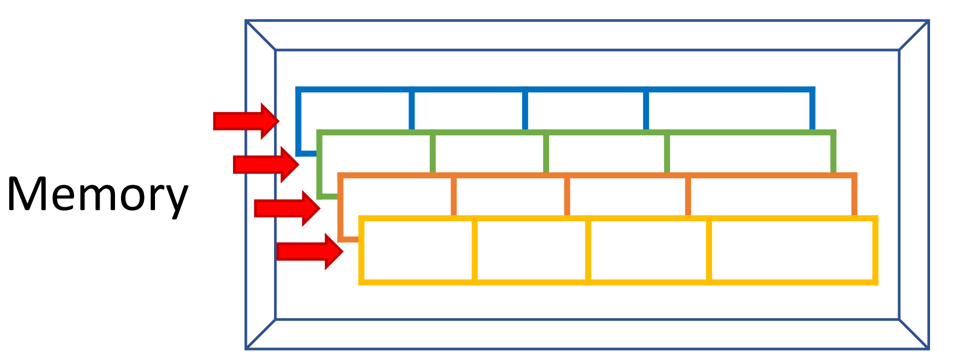
- Chunks are processed multiple times
- Random access
- Entire containers might be fetched several times

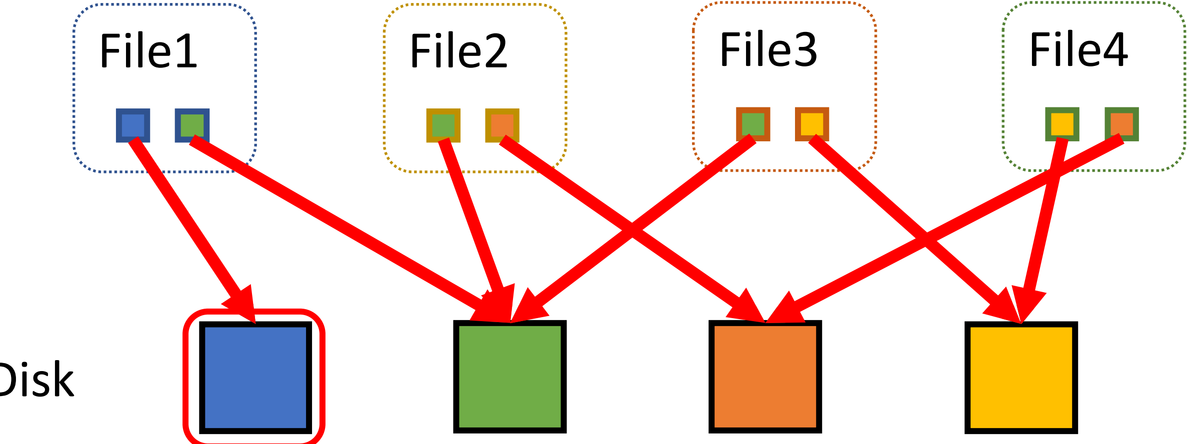




DedupSearch

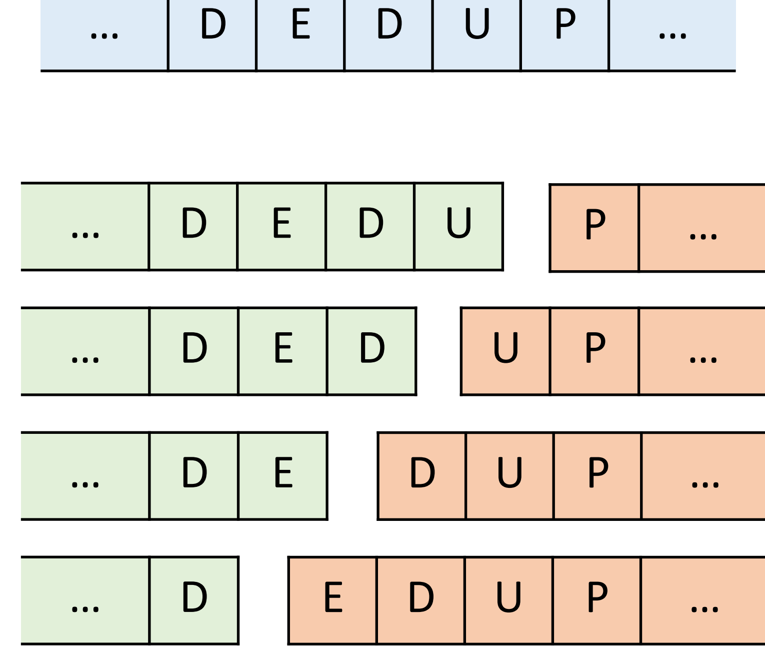
- Physical phase
  - Search in the physical data
  - 4 chunks read
- Logical phase
  - Use recipe to match search results and files





Challenges

Keyword: “DEDUP”



Keyword split

- Search also prefixes and suffixes
- *Aho-Corasick, 1975*

Tiny results

… T

E …

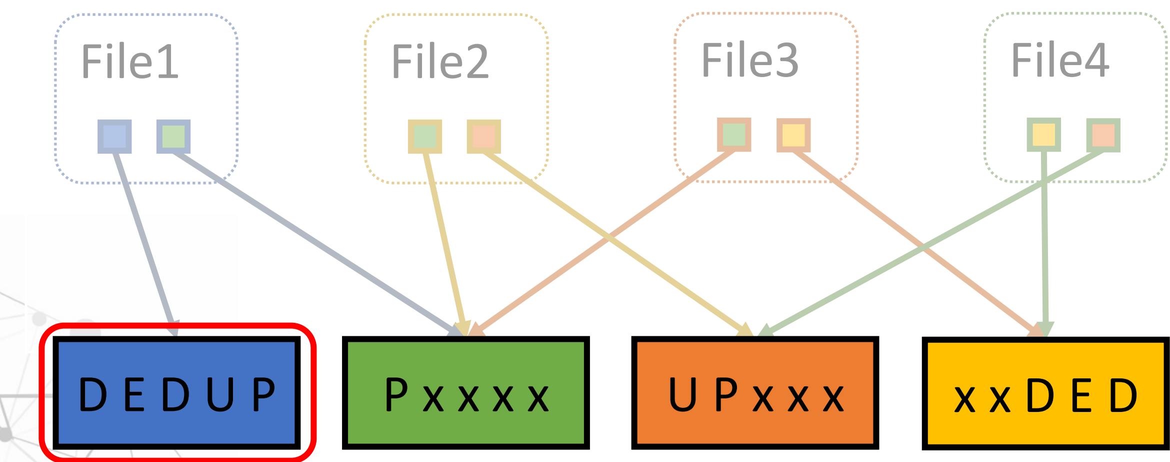
- What about “TASTE”?
  - 12% of text letters are “E”
  - 9% of text letters are “T”

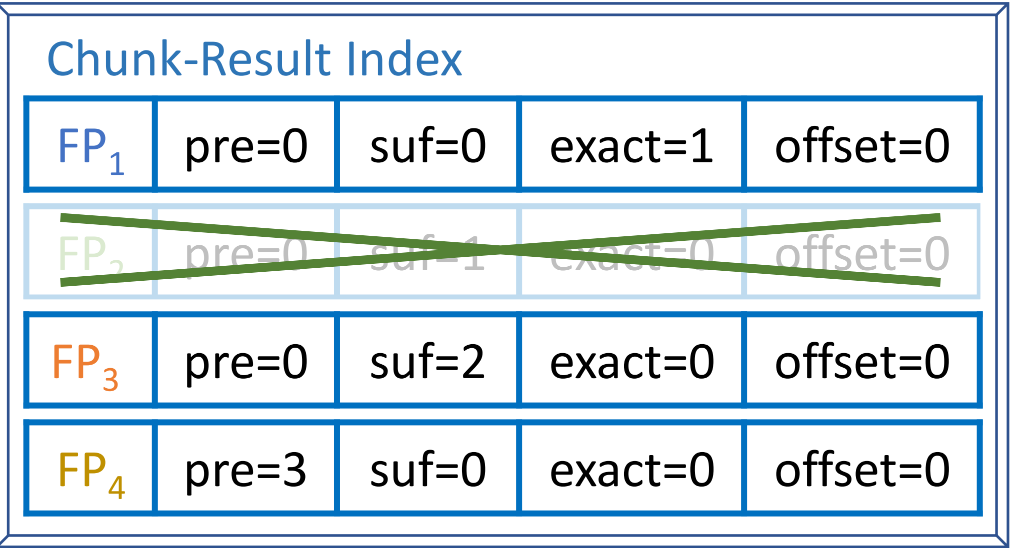
Physical Phase

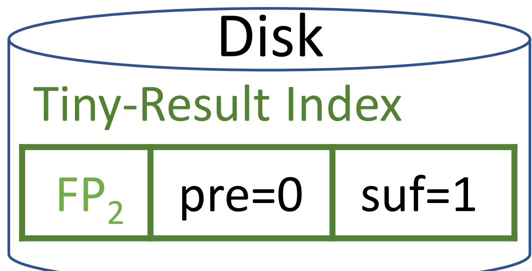
1. Read chunks
2. Search keyword in chunk

parallel threads

Problem: tiny prefix/suffix
→ huge in-memory index



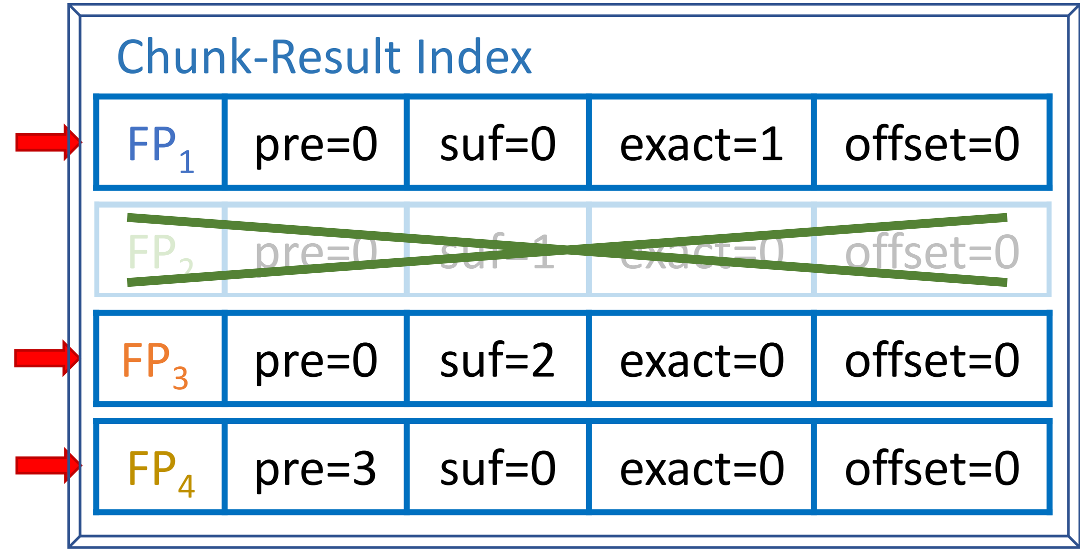


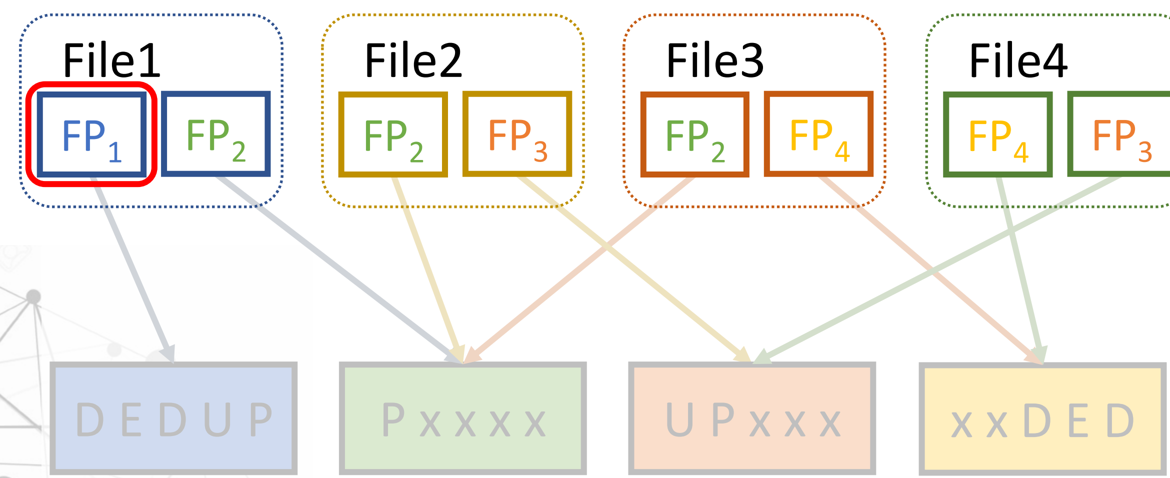


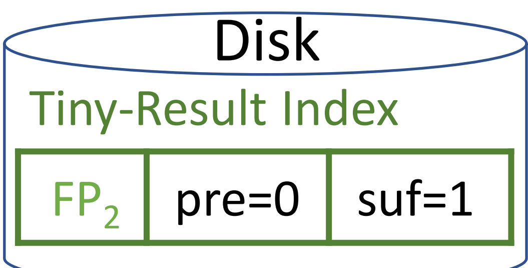
Logical Phase

1. Read recipe
2. Fetch chunk-result object
3. Collect matches
   - including tiny-result fetch

parallel threads





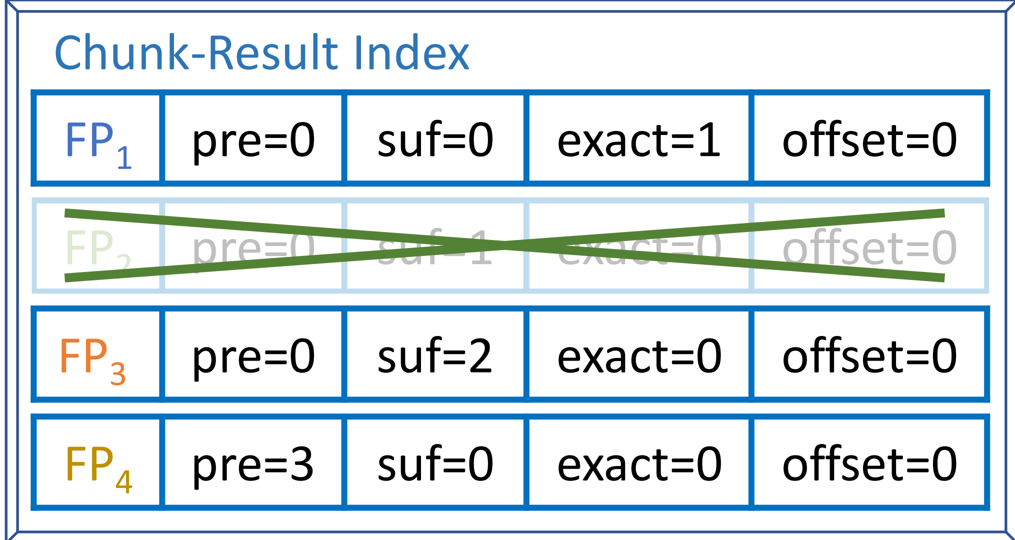


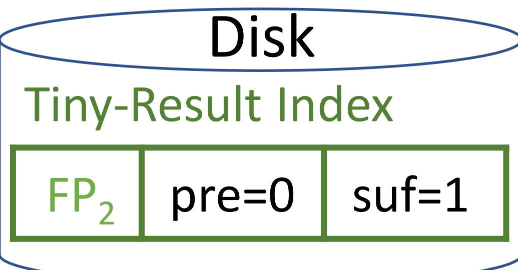
output:
File1: offset=0
File4: offset=2

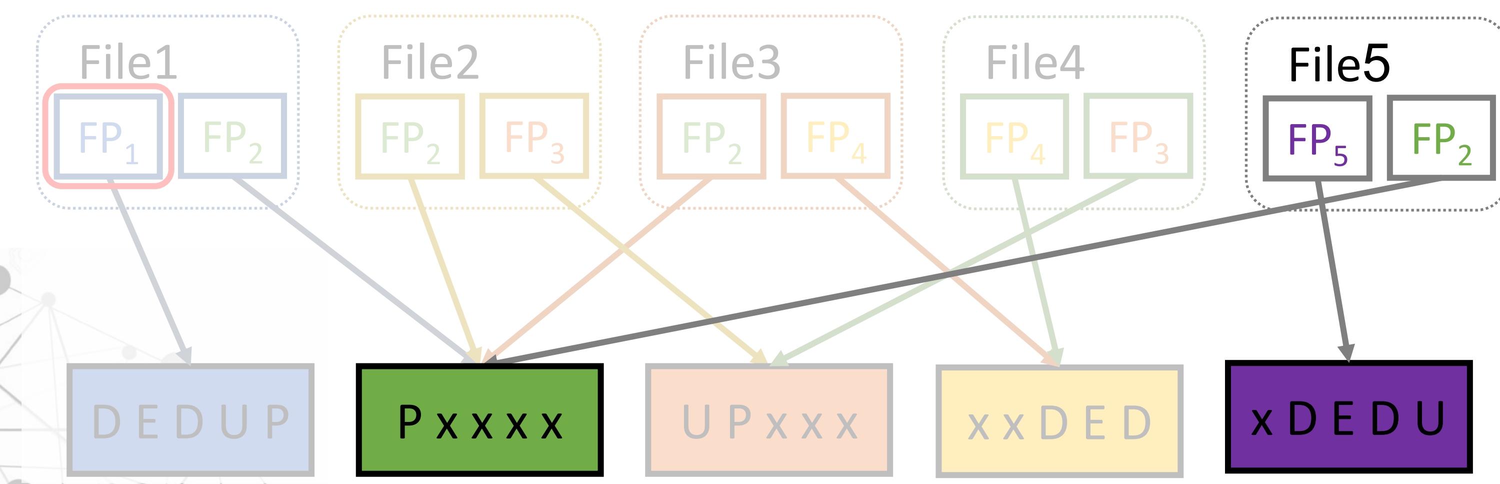
Logical Phase

1. Read recipe
2. Fetch chunk-result object
3. Collect matches
   - including tiny-result fetch

parallel threads







output:
File1: offset=0
File4: offset=2

Experimental Setup

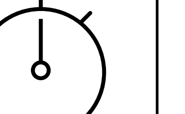
$Dataset$	$Backups\ \#$	$Logical\ Size$	$Physical\ Size$	$Recipe\ Size$	$\frac{Physical}{Logical}$
Wikipedia The Free Encyclopedia	41	2.59 TB	0.84 TB	12 GB	32.8 %
Linux™	408	204 GB	15 GB	2 GB	7.4 %

See paper for: up to 37 VM backups, up to 2703 Linux backups, up to 41 Wikipedia backups, up to 128 concurrent searches
Implemention based on *Destor, M. Fu. FAST 15.*

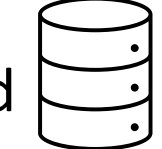
Naïve vs. DedupSearch

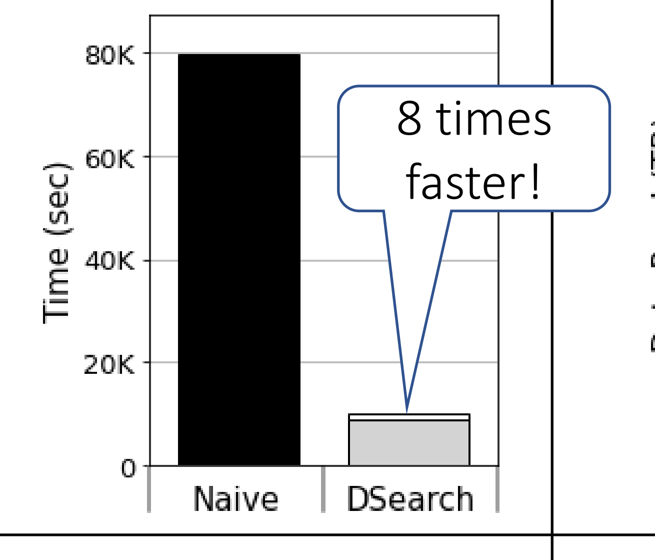
- DedupSearch outperforms *Naïve*
  - Less time, less data read

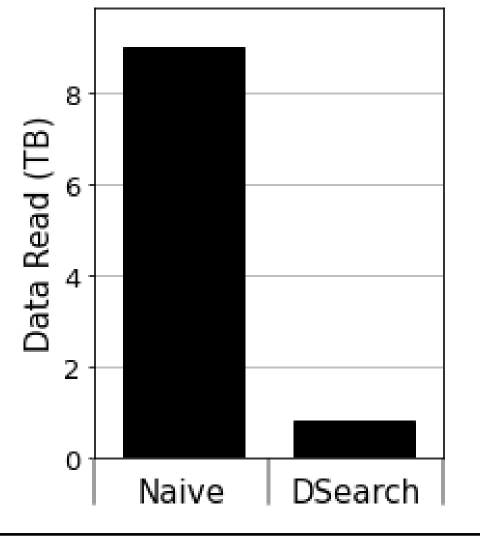
Time

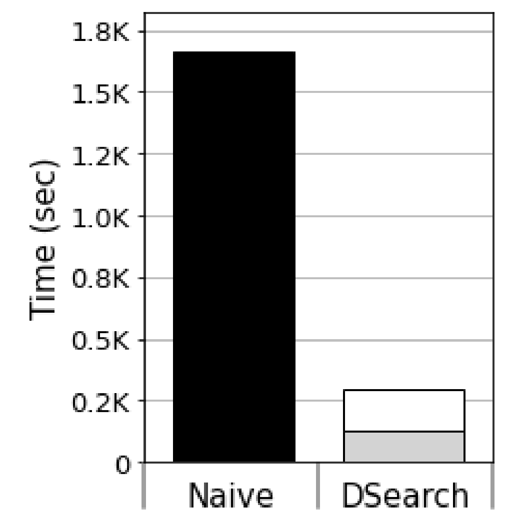


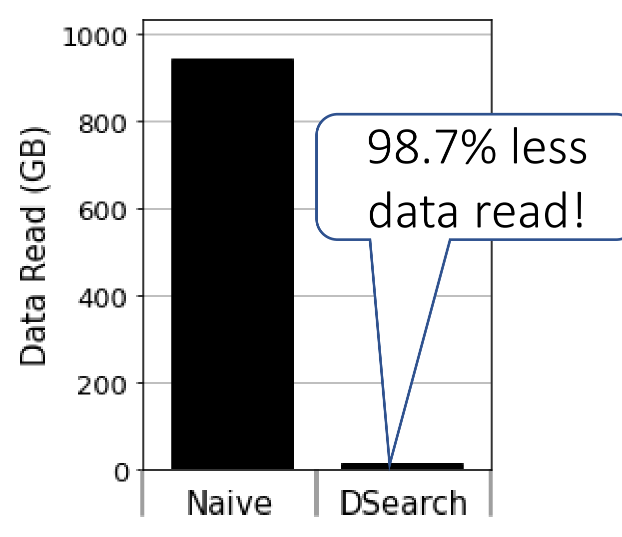
Data Read

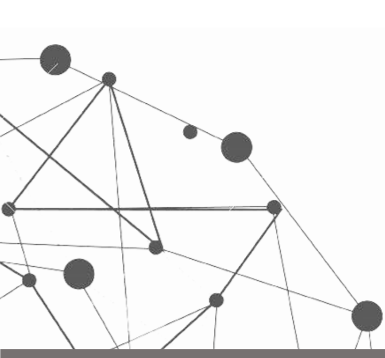






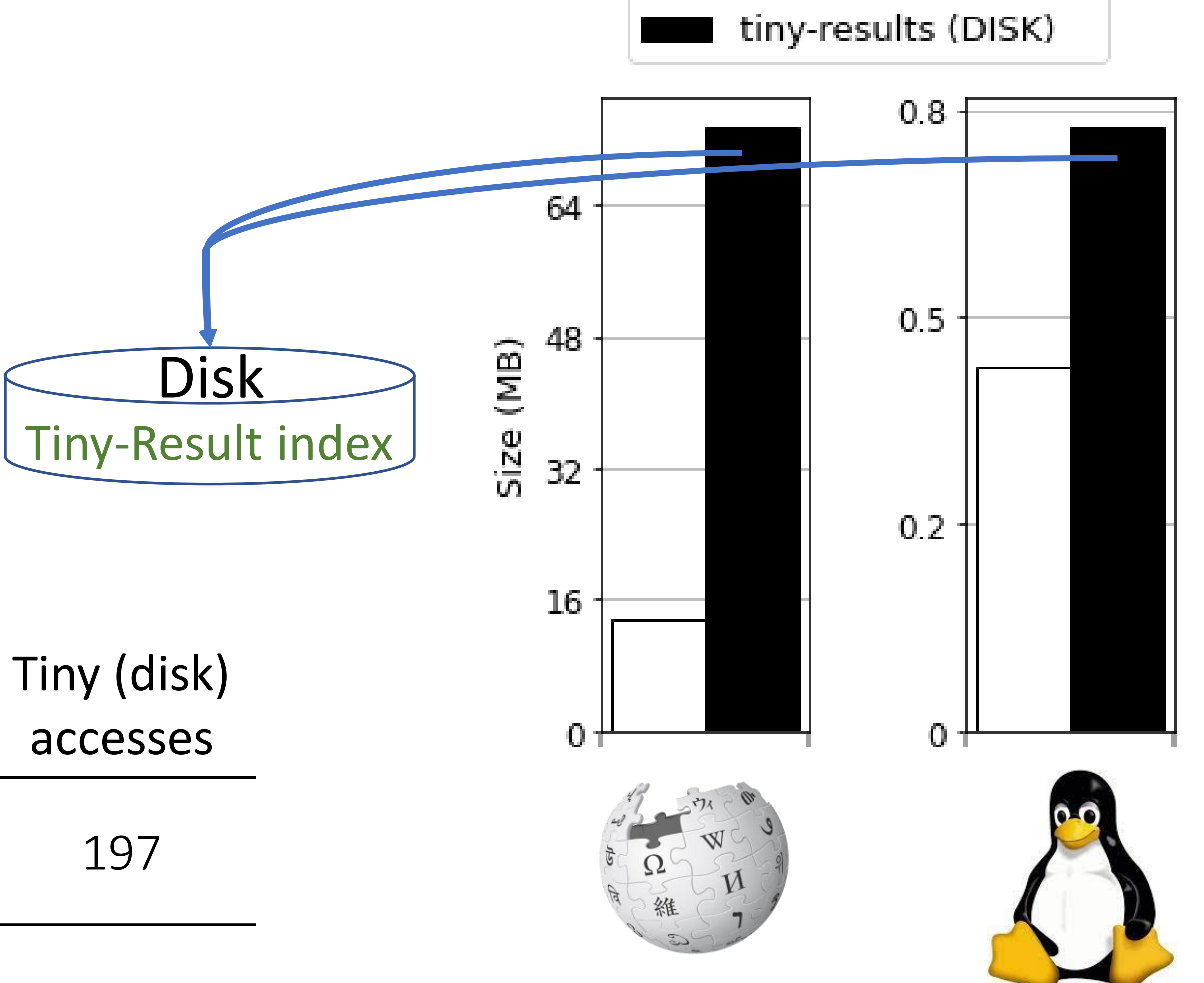






Database size

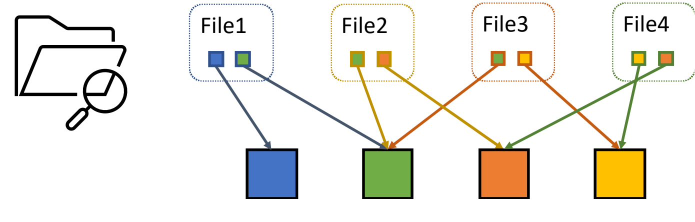
- Most intermediate results are tiny
- Little use of tiny



Dataset	Results #	matches split	Tiny (disk) accesses
Linux (204GB)	120K	228	197
Wiki (2.59TB)	2.34M	1170	1780

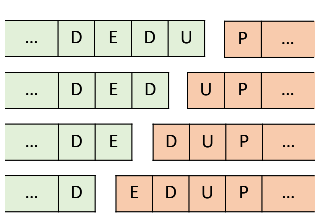
Conclusions

➢Deduplication-aware keyword search is more efficient than traditional keyword search



Challenges:

- Split keywords
- Lots of partial results



Our results:

- Up to 10 times faster
- Up to 99.9% less data read
- Compact database
- Minimum disk accesses