Deduplication

CSCI 333

Spring 2019

Logistics

- Lab 2a/b
- Final Project
- Final Exam
- Grades

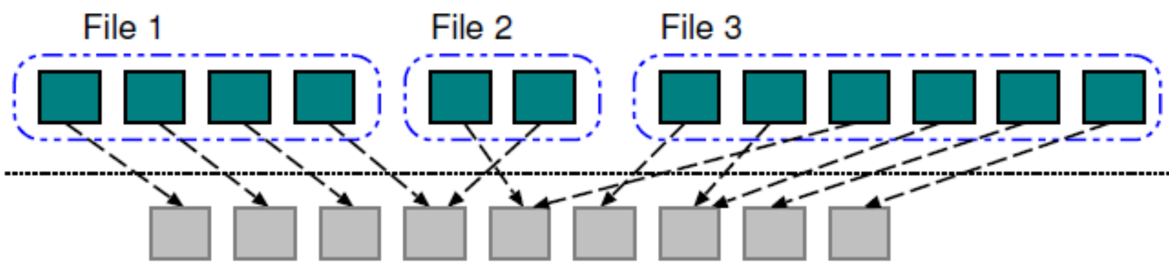
Last Class

- BetrFS [FAST '15]
  - Linux file system using B^e-trees
    - Metadata B^e-tree: path -> struct stat
    - Data in B^e-tree: path|{block#} -> 4KiB block
  - Schema maps VFS operations to efficient B^e-tree operations
    - Upserts, Range queries
  - Next iteration [FAST '16] : fixed slowest operations
    - Rangecast delete messages
    - "Zones"
    - Late-binding journal

This Class

- Introduction to Deduplication
  - Big picture idea
  - Design choices and tradeoffs
  - Open questions
- Slides from Gala Yadgar & Geoff Kuenning, presented at Dagstuhl
- I’ve added new slides (slides without borders) for extra context

Deduplication



Geoff Kuenning

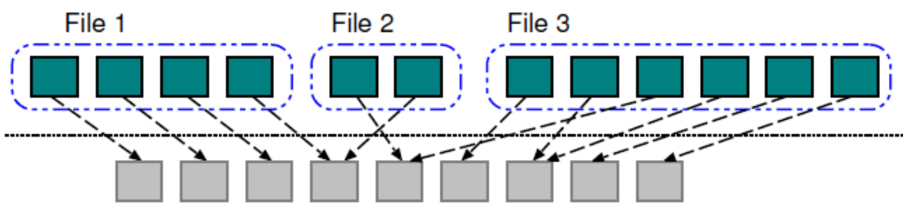
HARVEY MUDD COLLEGE

Gala Yadgar

TECHNION Israel Institute of Technology

Sources of Duplicates

- Different people store the same files
  - Shared documents, code development
  - Popular photos, videos, etc.



- May also share blocks
  - Attachments
  - Configuration files
  - Company logo and other headers

→ Deduplication!

Deduplication

- Dedup(e) is one form of compression
- High-level goal: identify duplicate objects and eliminate redundant copies
  - How should we define a duplicate object?
  - What makes a copy "redundant"?

- Answers are application-dependent and some of the more interesting research questions!

857 Desktops at Microsoft

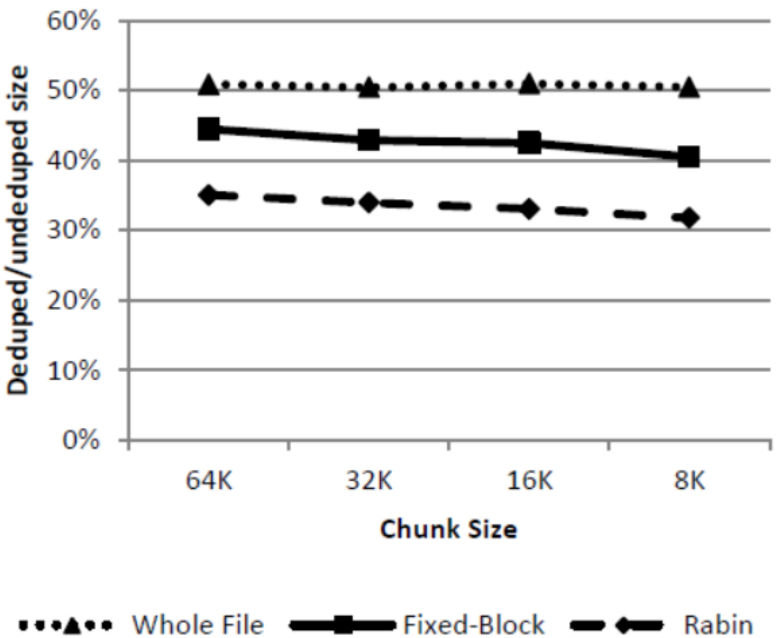


Figure 1: Deduplication vs. Chunk Size for Various Algorithms

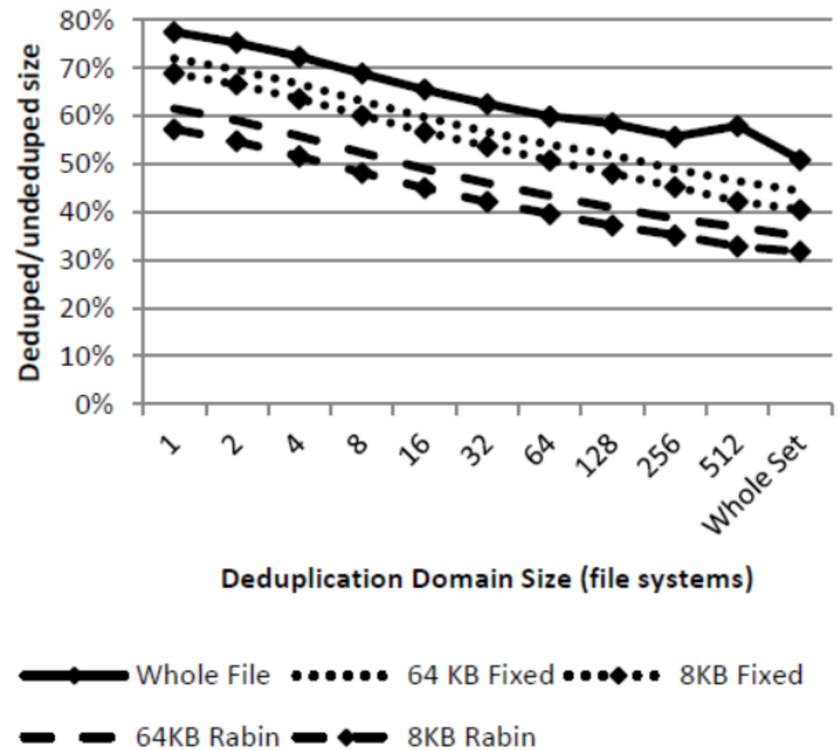


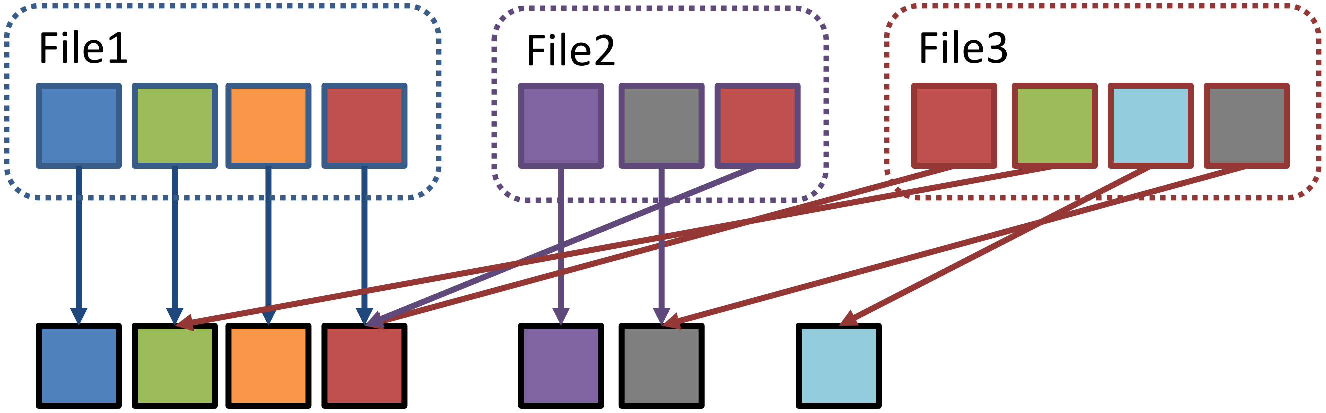
Figure 2: Deduplication vs. Deduplication Domain Size

D. Meyer, W. Bolosky. **A Study of Practical Deduplication.** FAST 2011

"Naïve" Deduplication

For each new file

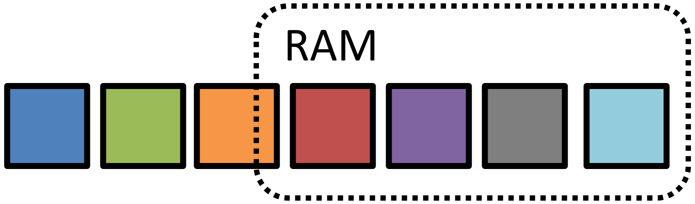
- Compare each block to all existing blocks
    - If new, write block and add pointer
    - If duplicate, add pointer to existing copy



Are we done?

Identifying Duplicates

- It's unreasonable to "Compare each block to all existing blocks"



→Fingerprints

Cryptographic hash of block content

Low collision probability

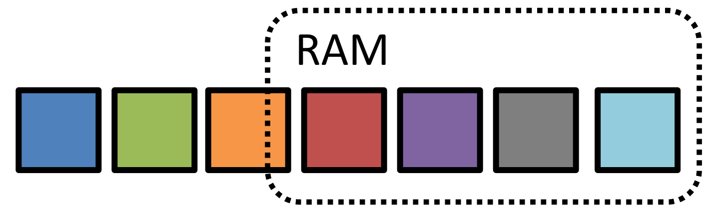
Dedup Fingerprints

- Goal: uniquely identify an object's contents
- How big should a fingerprint be?
  - Ideally, large enough that the probability of a collision is lower than the probability of a hardware error
    - MD5: 16-byte hash
    - SHA-1: 20-byte hash
- Technique: system stores a map (index) between each object's fingerprint and each object's location
  - Compare a new object's fingerprint against all existing fingerprints, looking for a match
  - Scales with *number* of unique objects, not *size* of objects

Identifying Duplicates

- It's unreasonable to "Compare each block to all existing blocks"

→Fingerprints

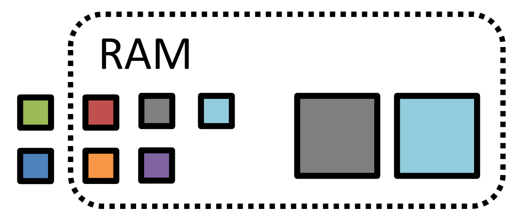


Cryptographic hash of block content

Low collision probability

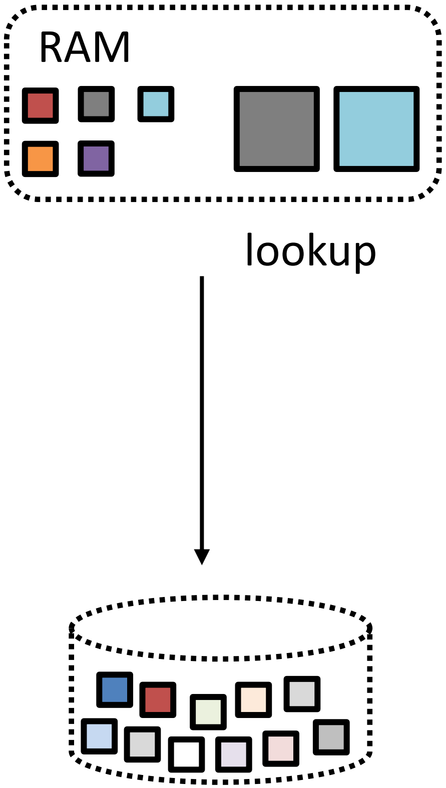
- It's also unreasonable to compare to all fingerprints...

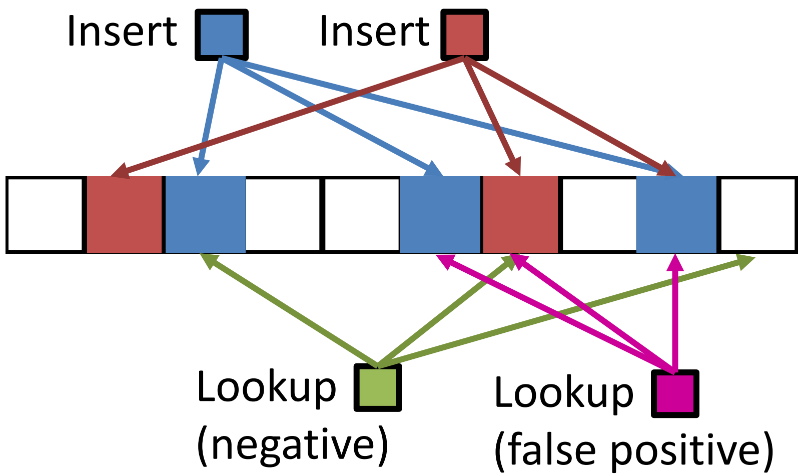
→Fingerprint cache



Fingerprint Lookup

- How should we store the fingerprints?
- Every unique block is a miss → miss rate ≥ 40%
- One solution: Bloom filter



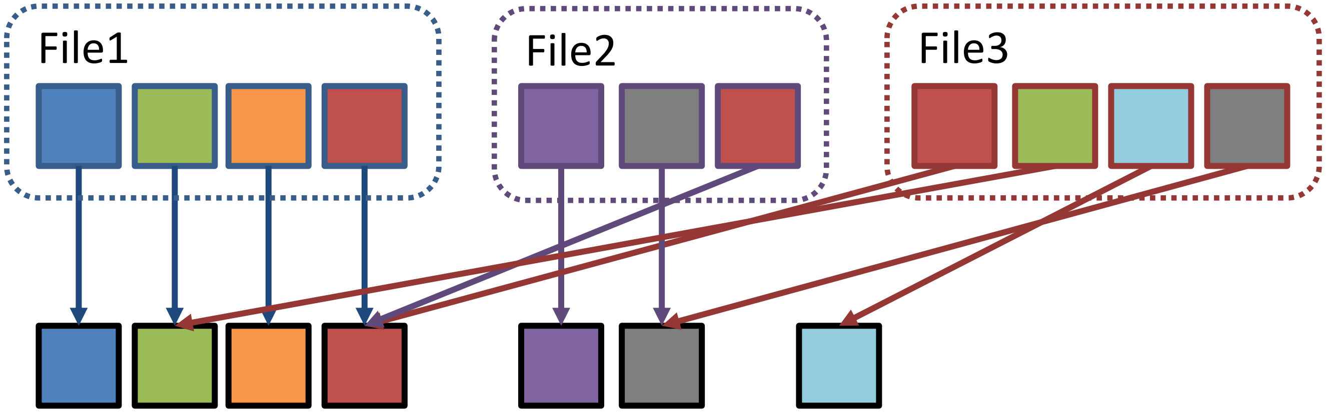


- Challenge: 2% false positive rate → 1TB for 4PB of data

How To Implement a Cache?

- (Bloom) Filters help us determine if a fingerprint exists
  - We still need to do an I/O to find the mapping
- Locality in fingerprints?
  - If we sort our index by fingerprint: cryptographic hash destroys all notions of locality
  - What if we grouped fingerprints by temporal locality of writes?

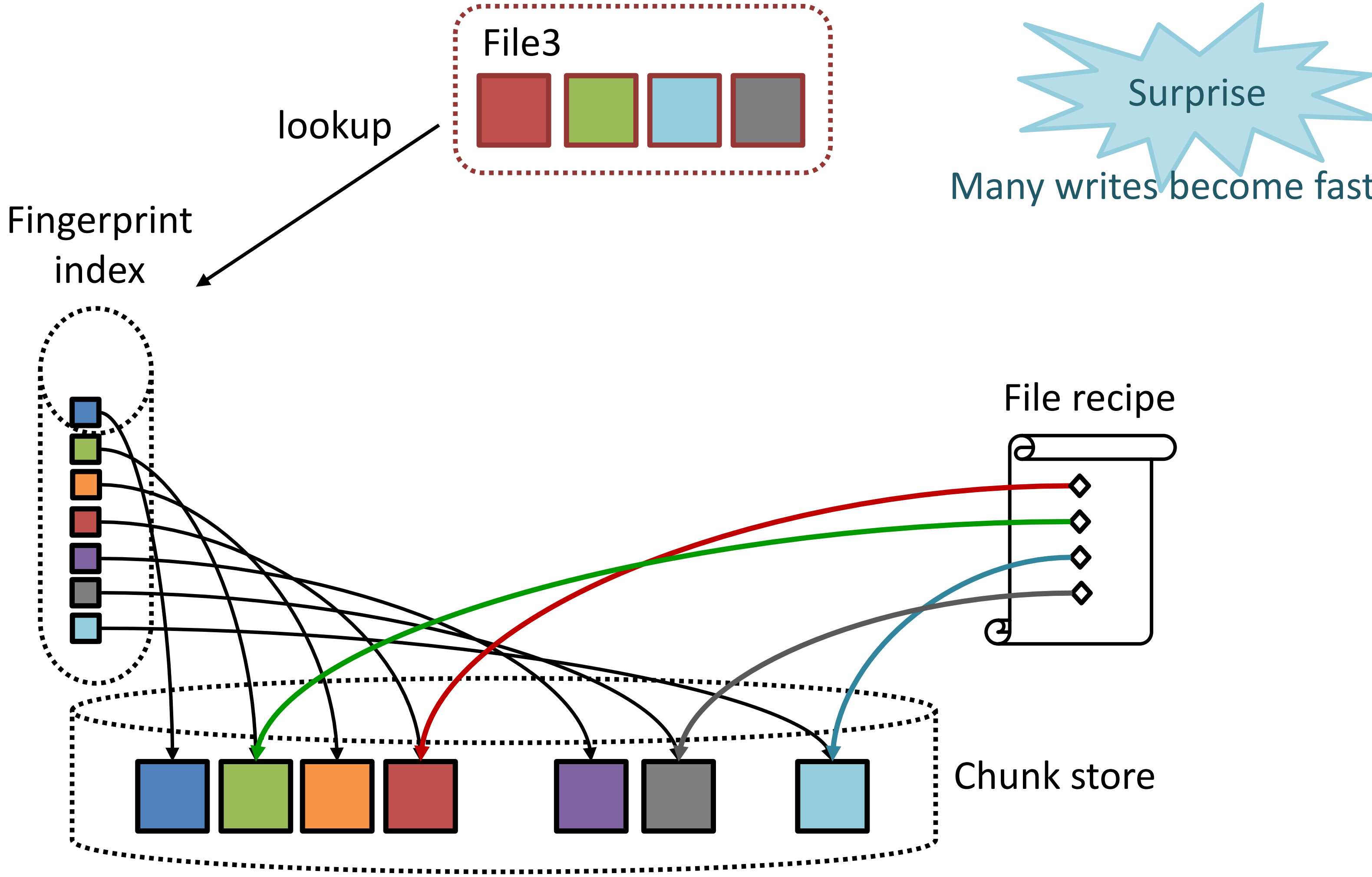
Reading and Restoring



- How long does it take to read File1?
- How long does it take to read File3?

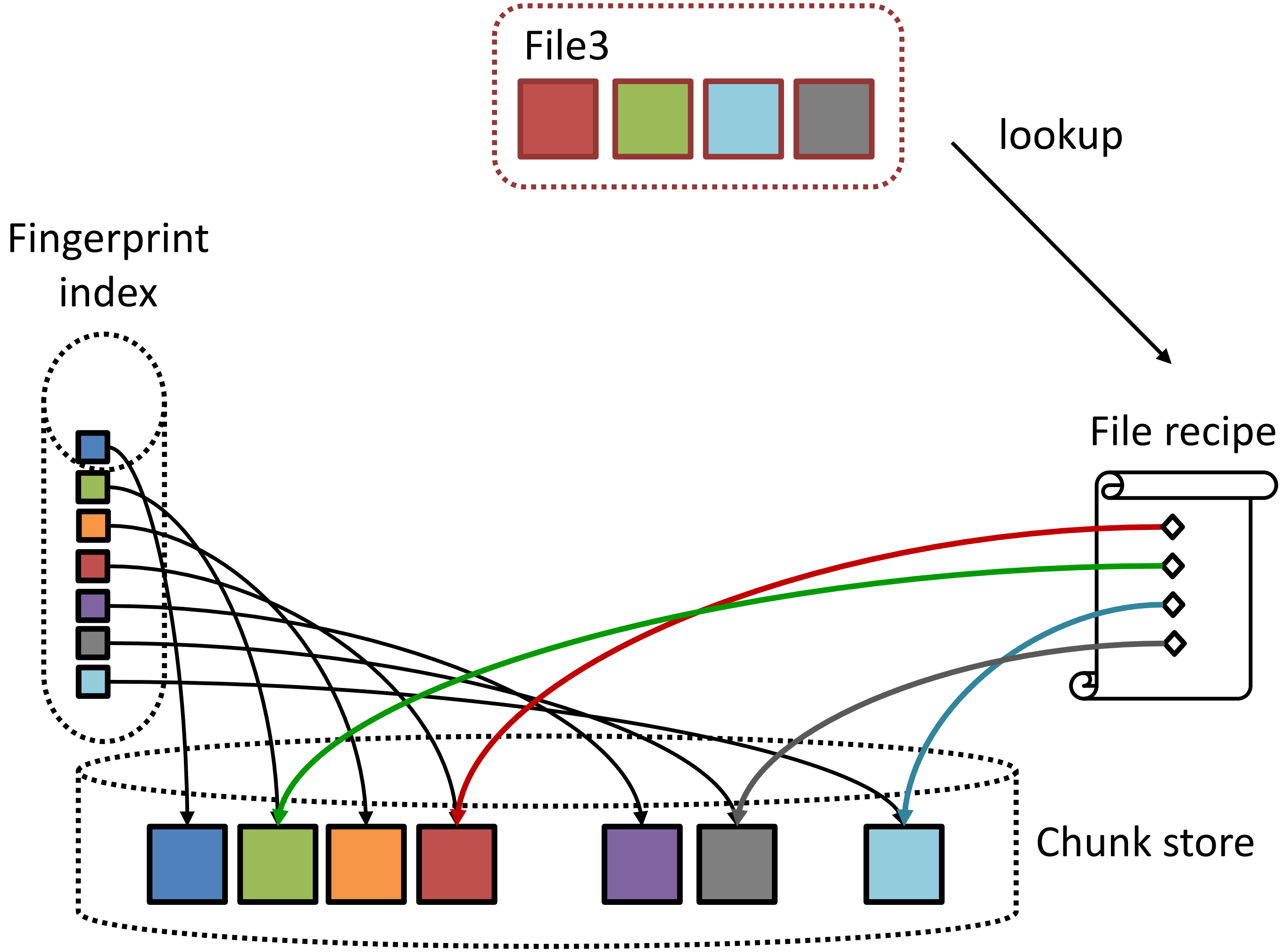
- Challenge: when is it better to store the duplicates?

Write Path



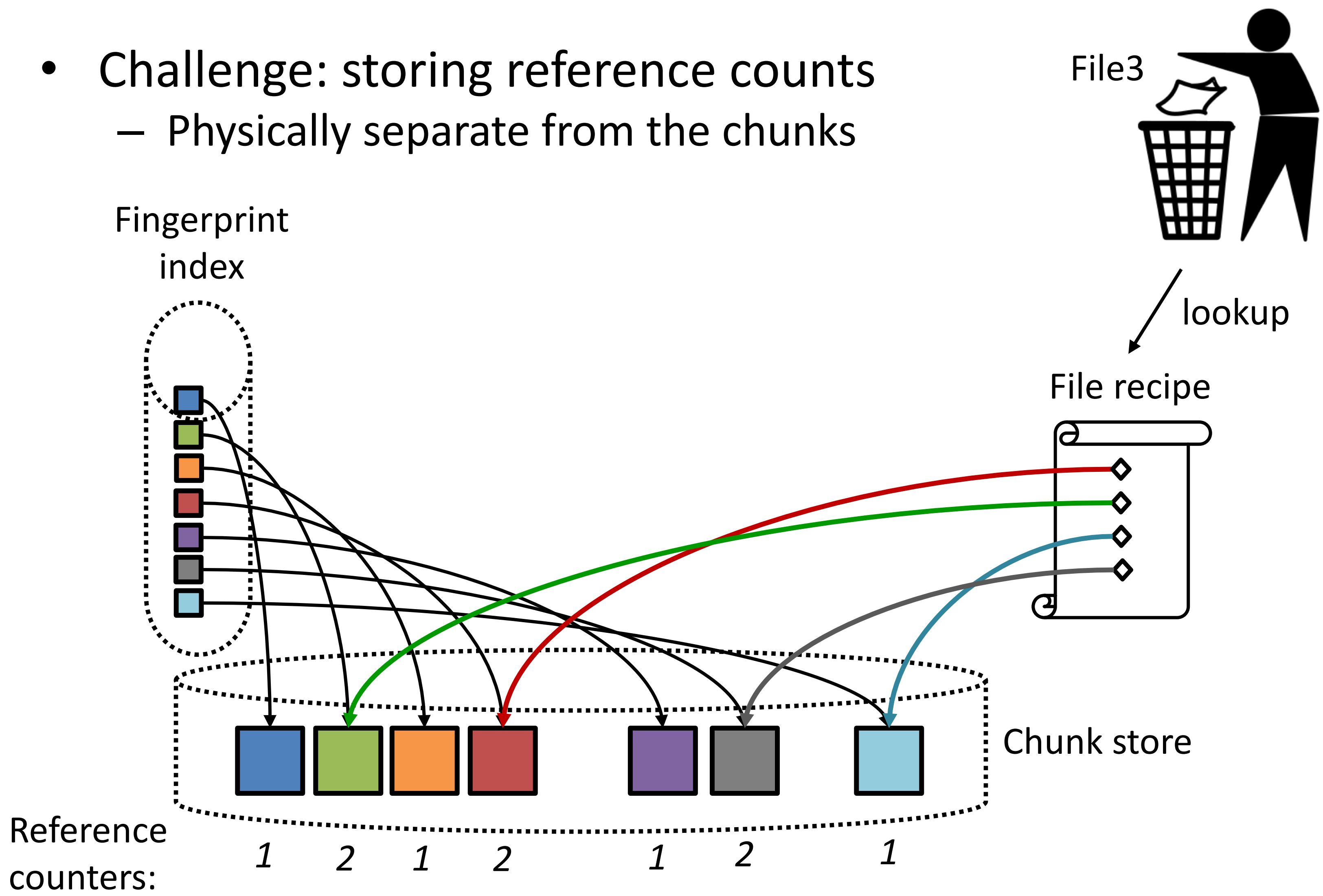
Many writes become faster!

Read Path



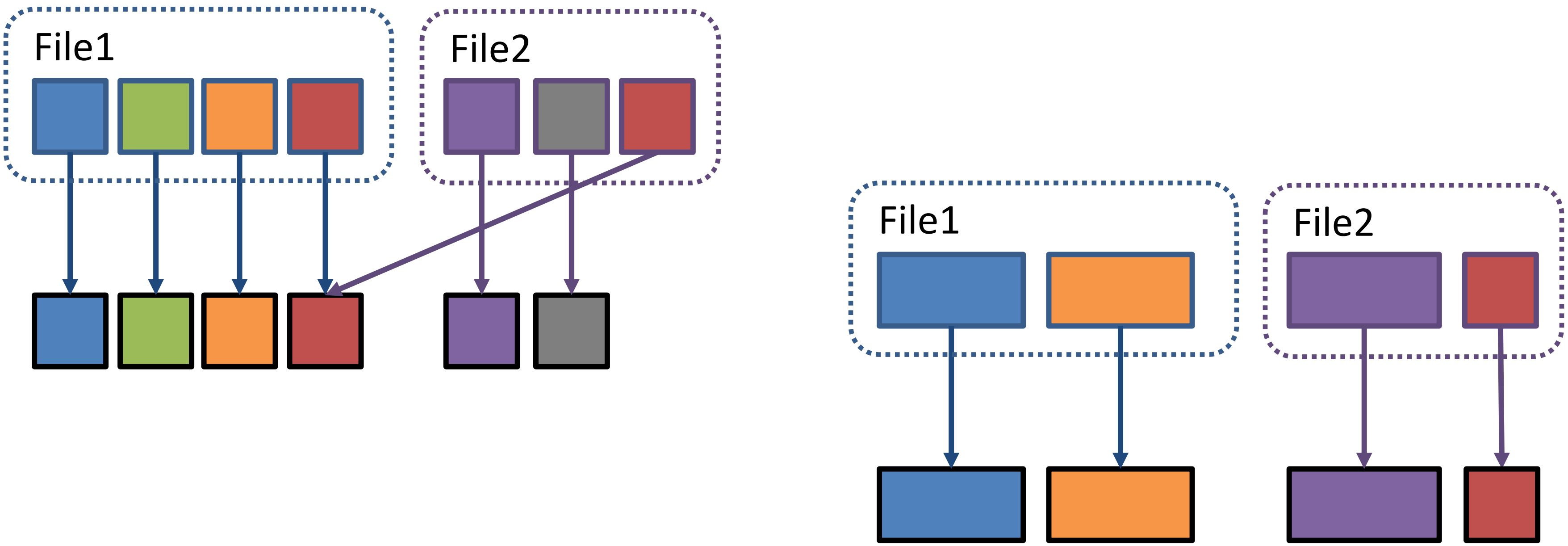
Delete Path

- Challenge: storing reference counts
  - Physically separate from the chunks



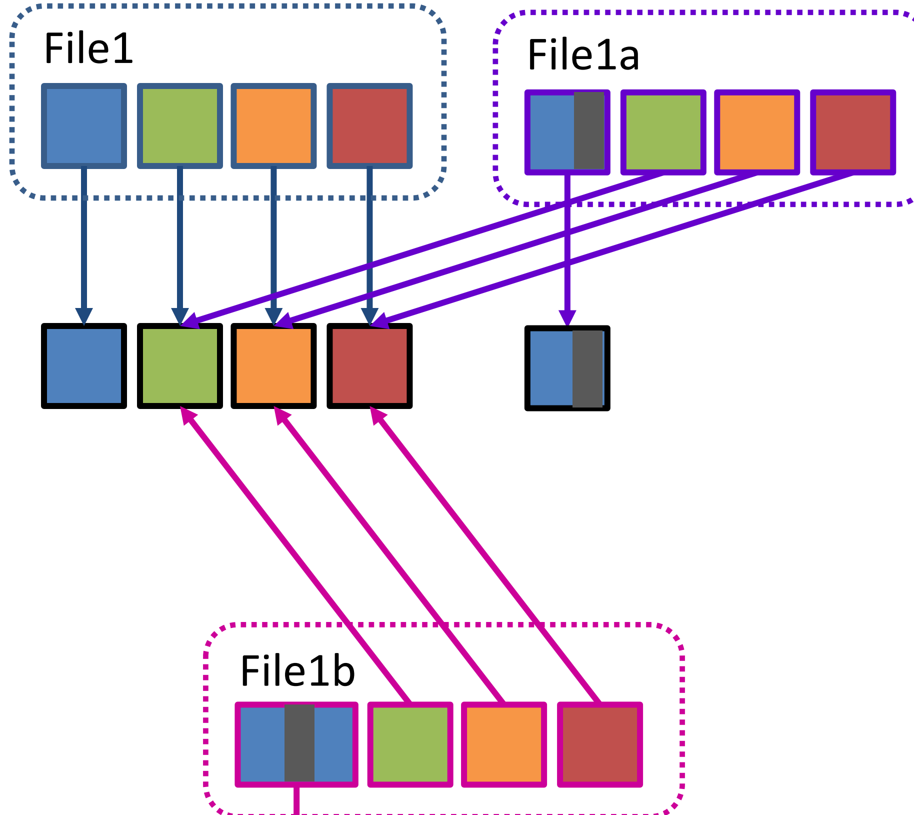
Chunking

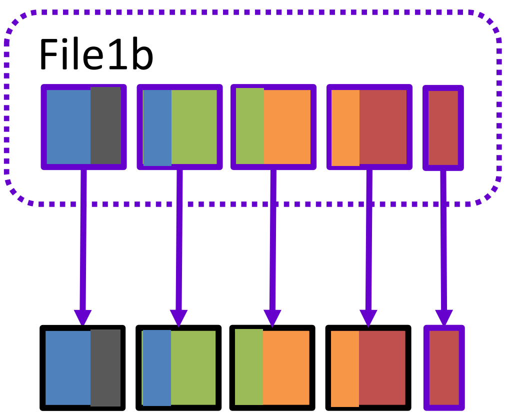
- Chunking: splitting files into blocks
- Fixed-size chunks: usually aligned to device blocks
- What is the best chunk size?

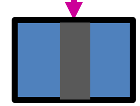


Updates and Versions

- Best case:
  aabbccdd
  → a**A**bbccdd
- Worst case:
  aabbccdd
  → a**A**abbccdd





Ideally… 

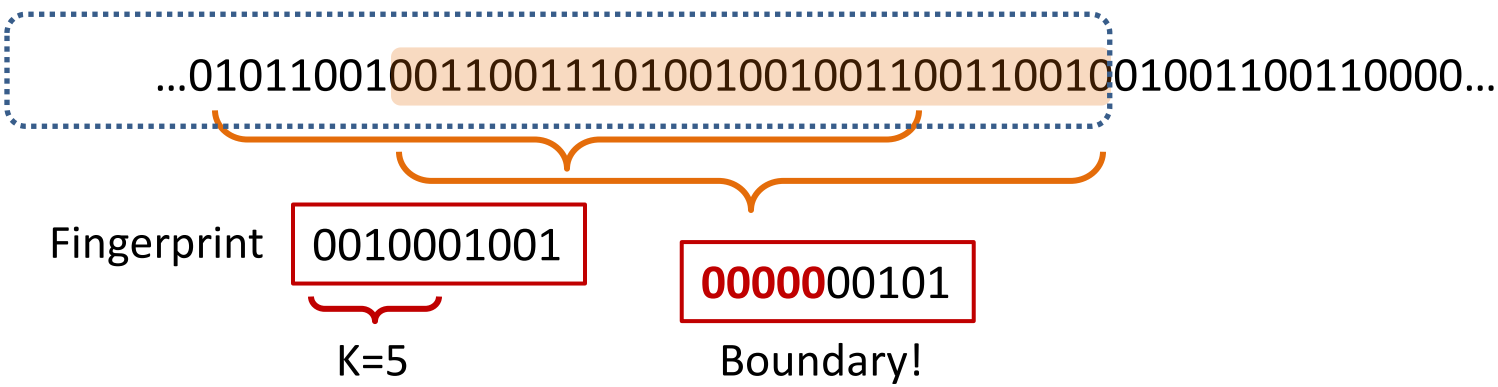
Variable-Size Chunks

- Basic idea: chunk boundary is triggered by a random string
- For example: 010
- aa010|bb010|cc010|dd → aAa010|bb010|cc010|dd

- Triggers should be:
  - Not too short/long
  - Not too popular (000000…)
  - Easy to identify

Identifying Chunk Boundaries

- 48-byte triggers (empirically, this works)
- Define a set of possible triggers
  - → K highest bits of the hash are == 0
  - → Rabin fingerprints do this efficiently
  - → "systems" solutions for corner cases



- Challenge: parallelize this process

Rabin Fingerprints

- "The polynomial representation of the data modulo a predetermined irreducible polynomial" [LBFS sosp01]
- What/why Rabin fingerprints?
  - Calculates a rolling hash
  - "Slide the window" in a constant number of operations (intuition: we "add" a new byte and "subtract" an old byte to slide the window by one)
  - Define a "chunk" once our window's hash matches our target value (i.e., we hit a trigger)

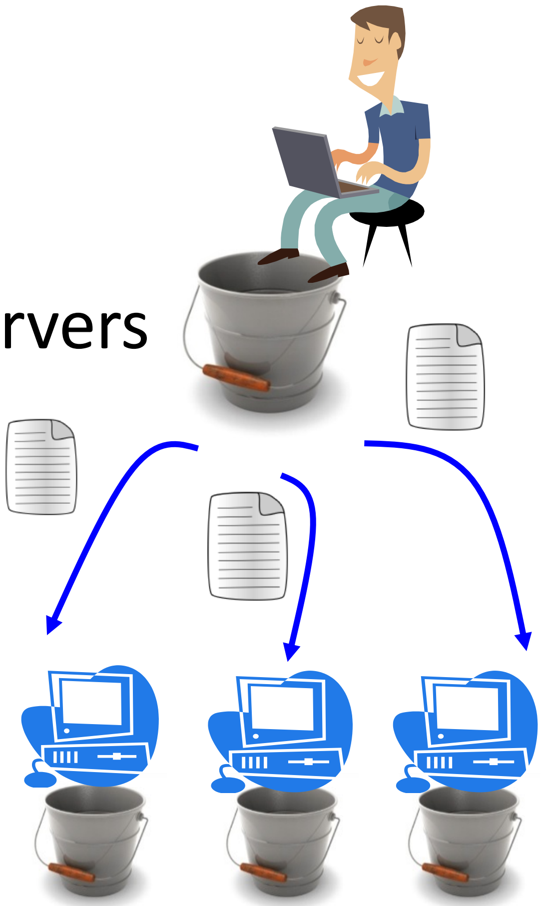
Defining chunk boundaries

- Tradeoff between small and large chunks?
  - Finer granularity of sharing vs. metadata overhead
- With process just described, how might we:
  - Produce a very small chunk?
  - Produce a very large chunk?
- How might we modify our chunking algorithm to give us "reasonable" chunk sizes?
  - To avoid small chunks: don't consider boundaries until minimum size threshold
  - To avoid large chunks: as soon as we reach a maximum threshold, insert a chunk boundary

Distributed Storage

Increase storage capacity and performance with multiple storage servers

- Each server is a separate machine (CPU,RAM,HDD/SSD)
- Data access is distributed between servers



☝ **Scalability**

Increase capacity with data growth

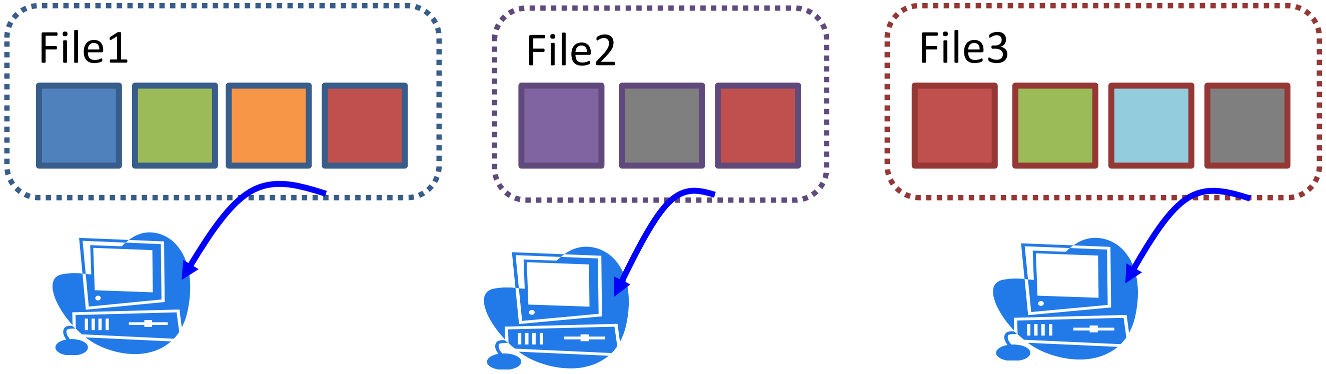
☝ **Load balancing**

Independent of workload

☝ **Failure handling**

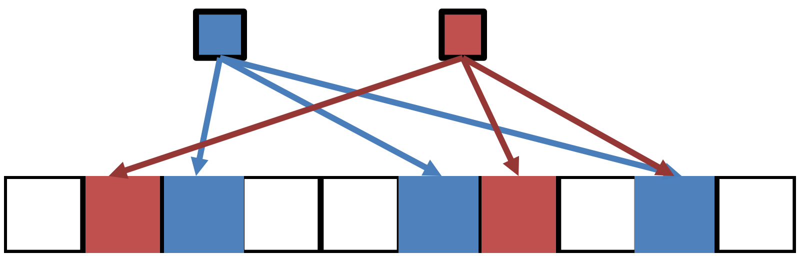
Network, nodes and devices always fail

Distributed Deduplication



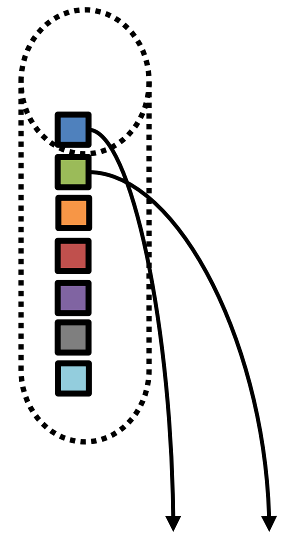
- Where/when should we look for duplicates?
- Where should we store each file?

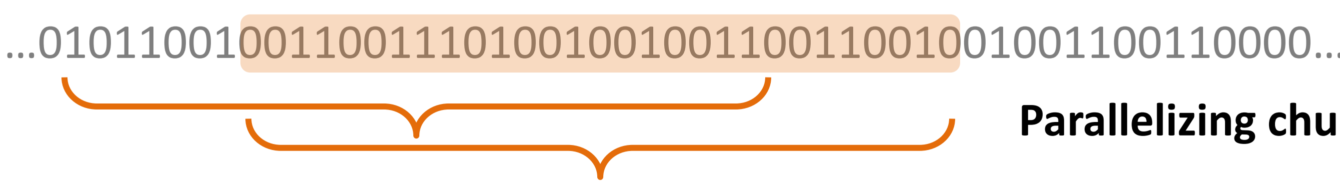
Challenges (aka Summary)



Approximate membership query structures (AMQ)

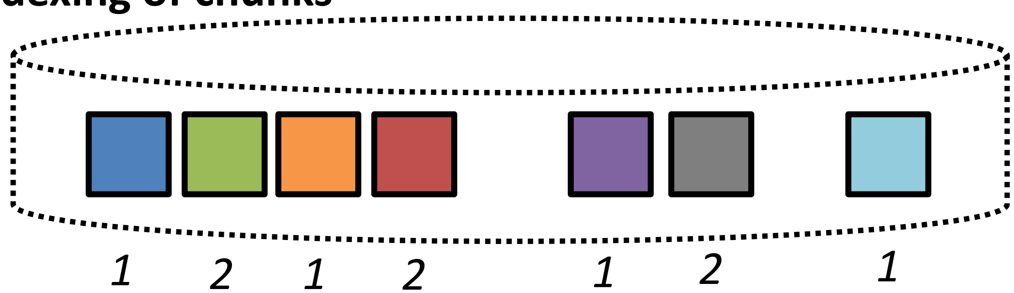
Size of fingerprint dictionary





Parallelizing chunking

Bidirectional indexing of chunks



→ Wonderful theory problems!

Next Class?

- ~~Specific dedup system(s) (4)~~
- Mapreduce (+ write-optimized) (2)
- Google file system (1)
- RAID (3)

Final Project Discussion

- Get with your group
- Find another group
- Pitch your project / show them your proposal
  - React/revise