



# Sparse Indexing: Large-Scale, Inline Deduplication Using Sampling and Locality

Mark Lillibridge, Kave Eshghi, Deepavali Bhagwat, Vinay Deolalikar, Greg Trezise, and Peter Camble

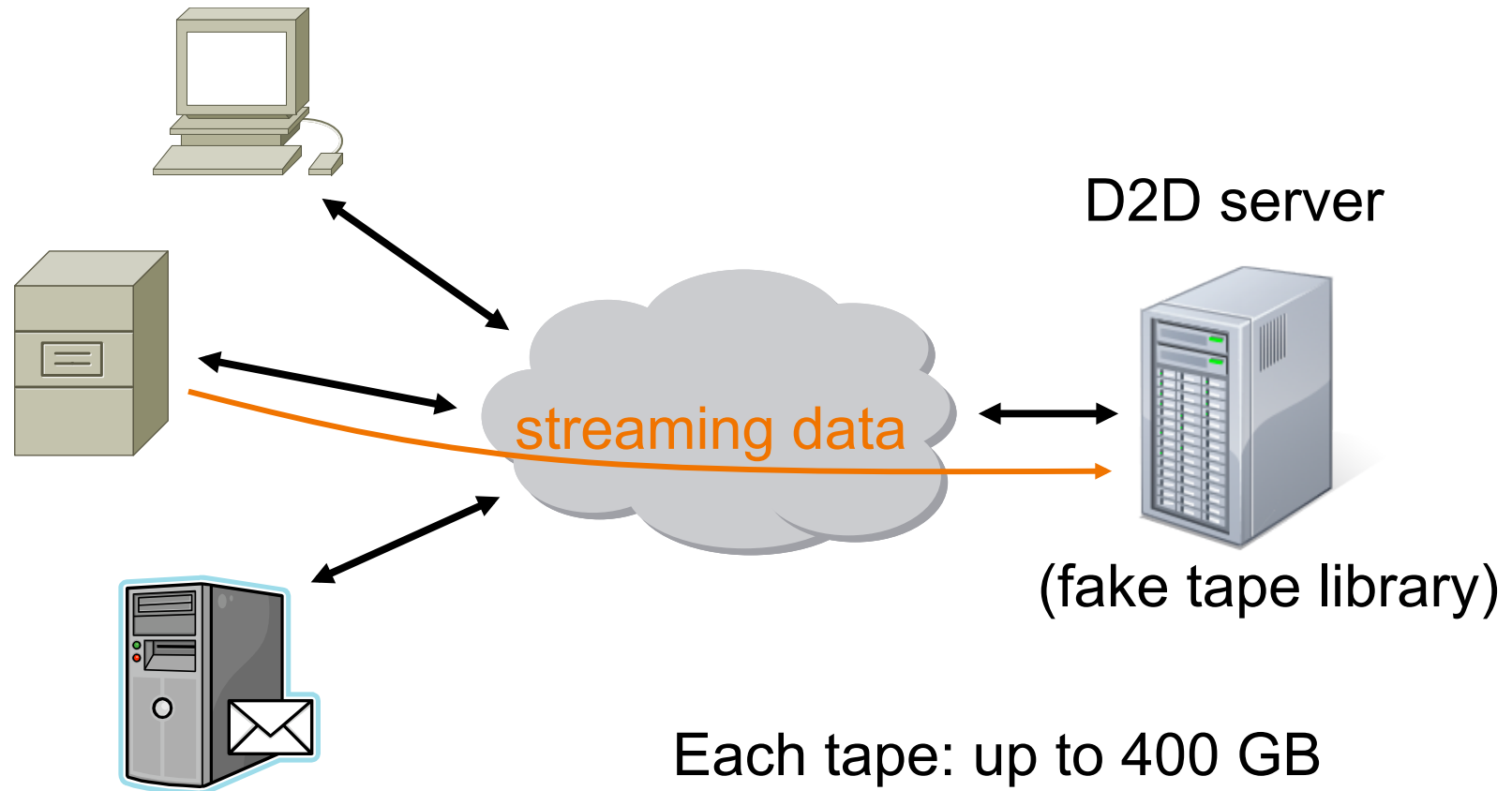
Work done at Hewlett-Packard laboratories



# The Problem:

deduplication at scale  
for disk-to-disk backup

# A Disk-to-Disk Backup Scenario



Each tape: up to 400 GB

Total: 100 TB – 10 PB

# Example backup streams

Monday:



Tuesday:



Wednesday:



# Little changes from day-to-day

Monday:



Tuesday:



Wednesday:



# After ideal deduplication

Monday:



Tuesday:



Wednesday:



# Chunk-based deduplication

Monday:



Tuesday:



Wednesday:



# Chunk-based deduplication

Monday:



Tuesday:



Wednesday:





# Chunk-based deduplication

Monday:



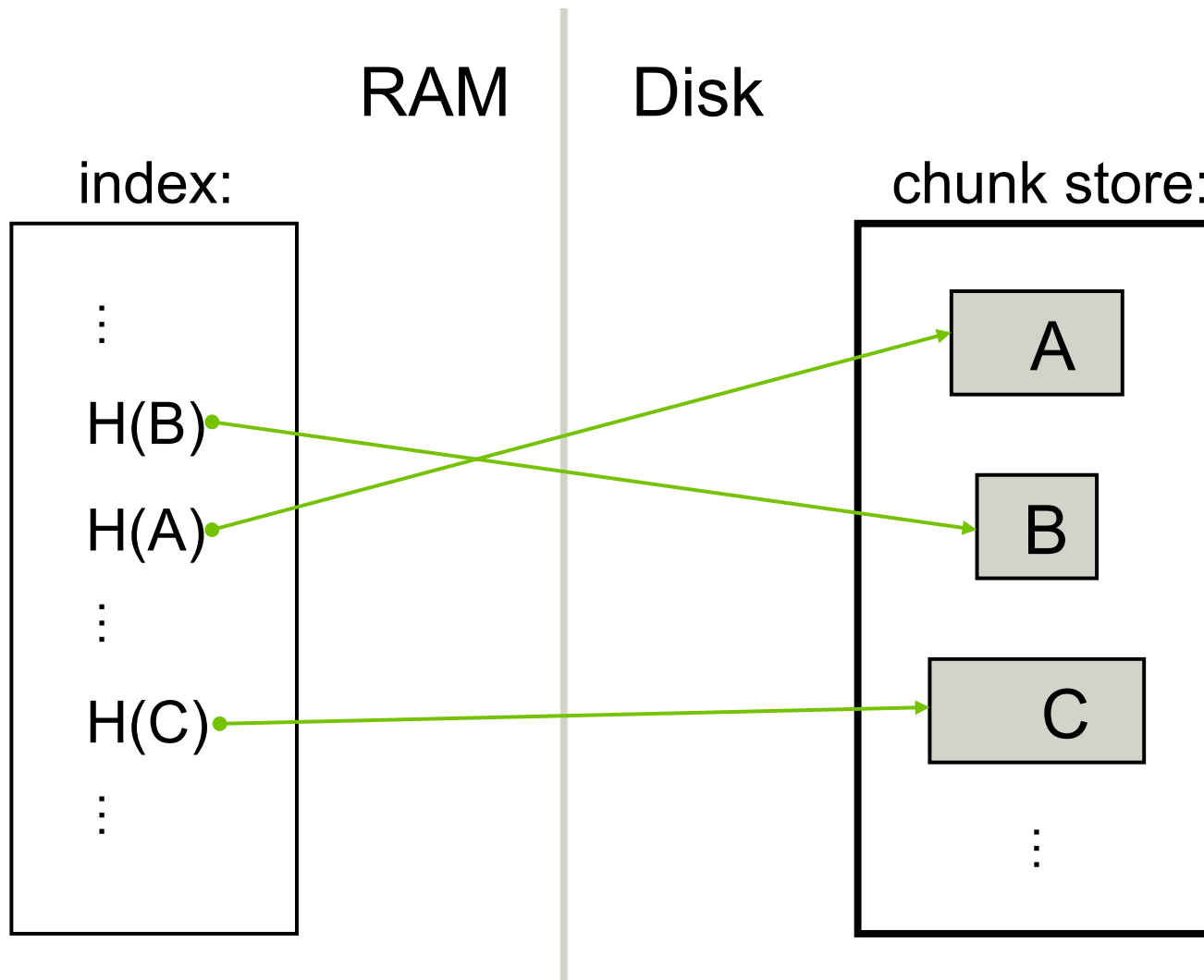
Tuesday:



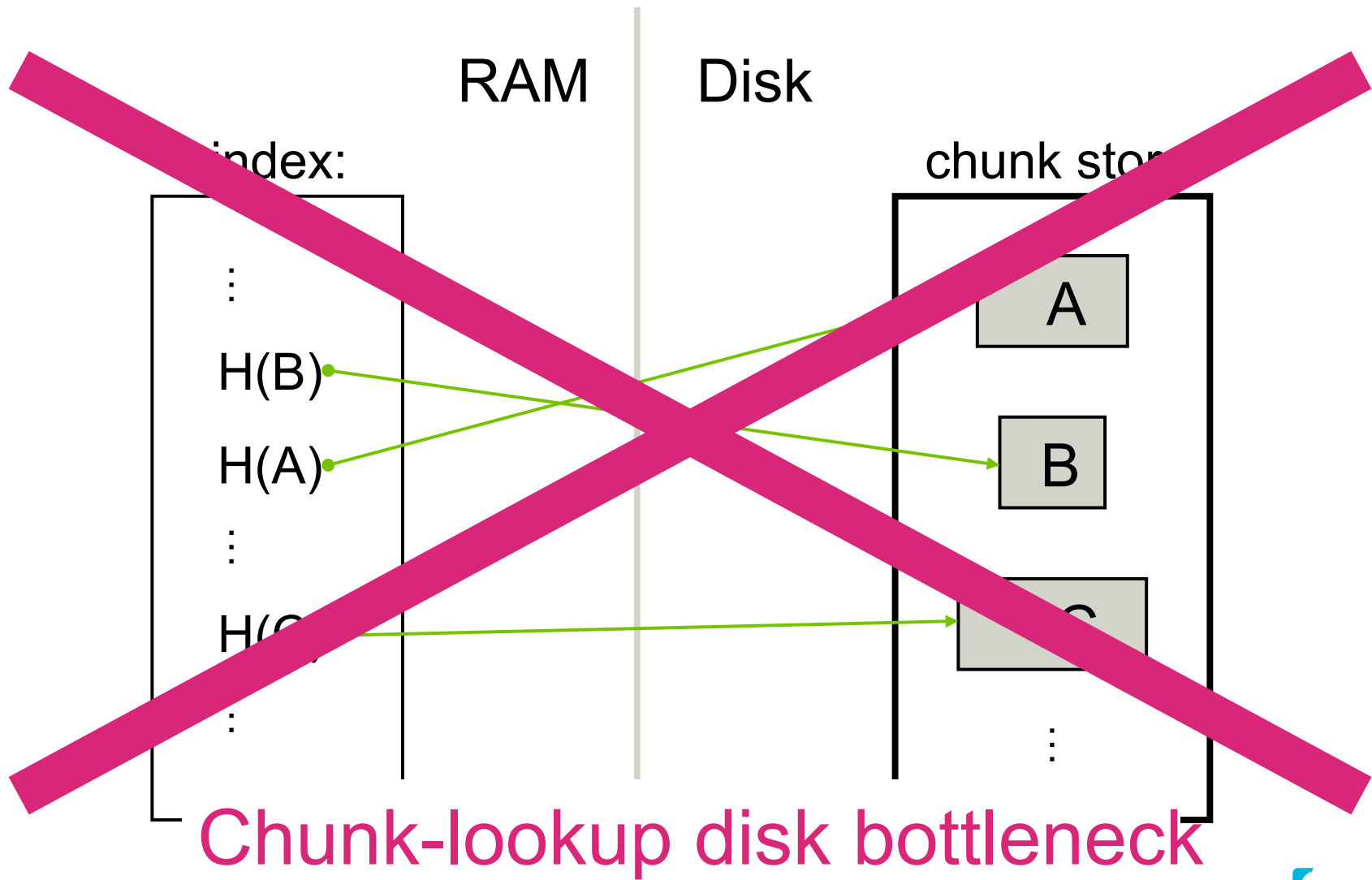
Wednesday:



# The standard implementation



# The standard implementation



# One existing solution

- **Avoiding the Disk Bottleneck in the Data Domain Deduplication File System.** Benjamin Zhu, *Data Domain, Inc.*; Kai Li, *Data Domain, Inc.*, and Princeton University; Hugo Patterson, *Data Domain, Inc.* FAST'08.
- Today: a new approach that
  - uses significantly less RAM
  - provides a guaranteed minimum throughput

Our Approach:

Sparse indexing

# Sparse indexing

- Key ideas:
  - Chunk locality
  - Sampling

# No temporal locality

Monday:



Tuesday:



Wednesday:



# Large sections of data reappear mostly intact → chunk locality

Monday:



Tuesday:



Wednesday:

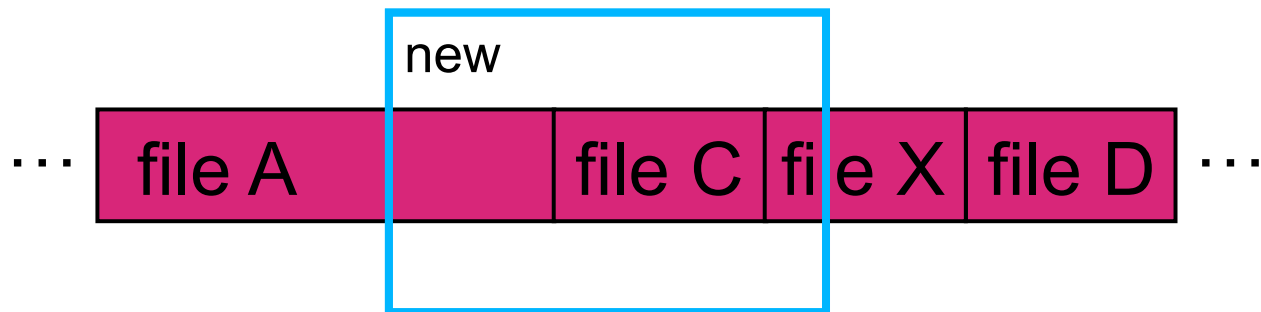
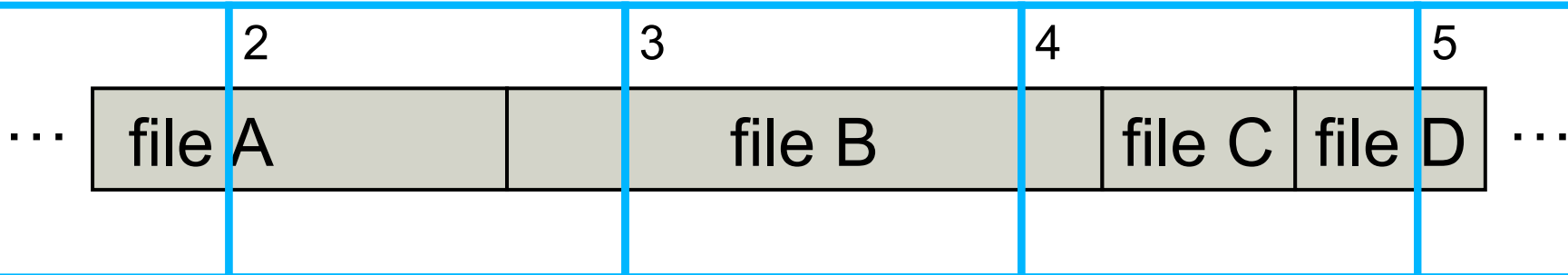




# Exploiting chunk locality

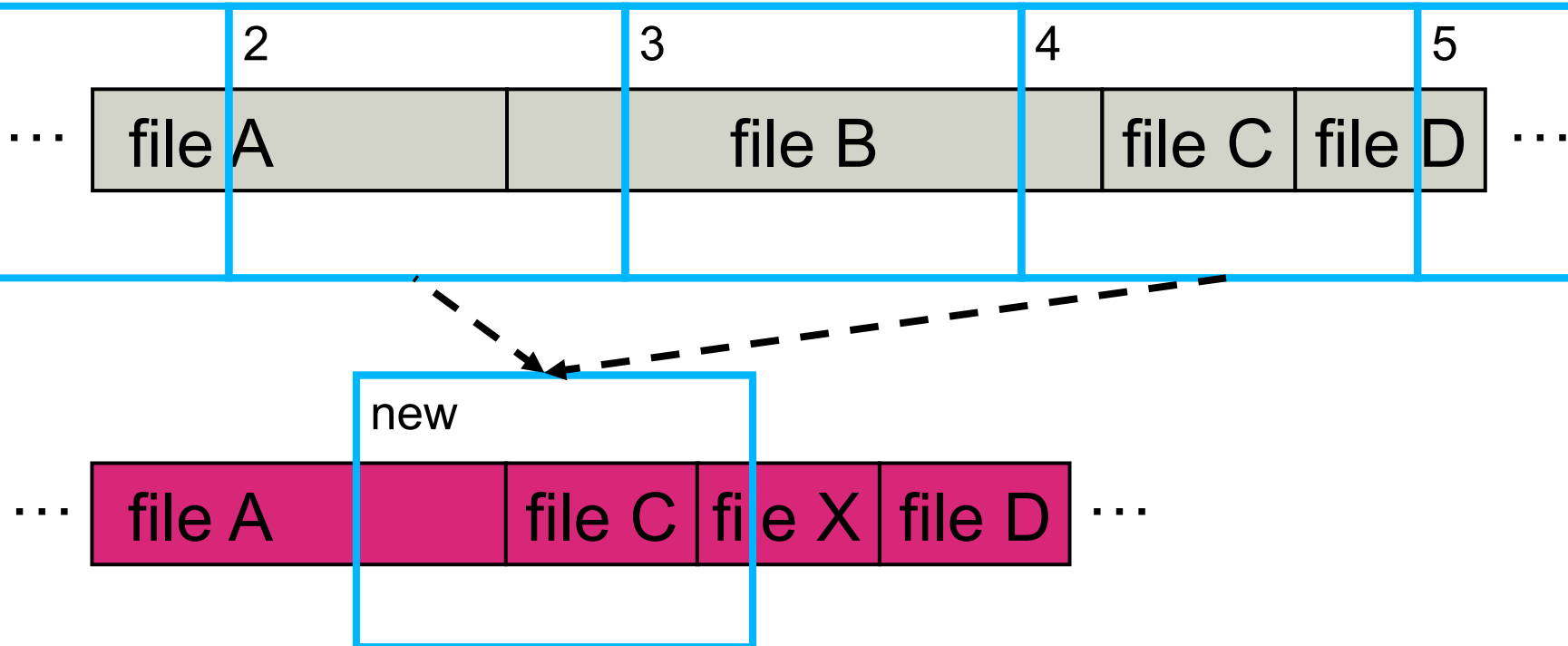


# Divide into *segments*



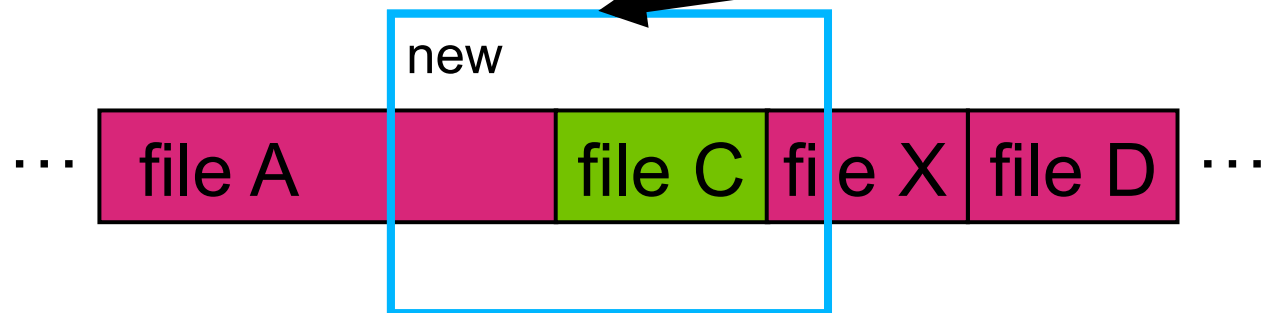
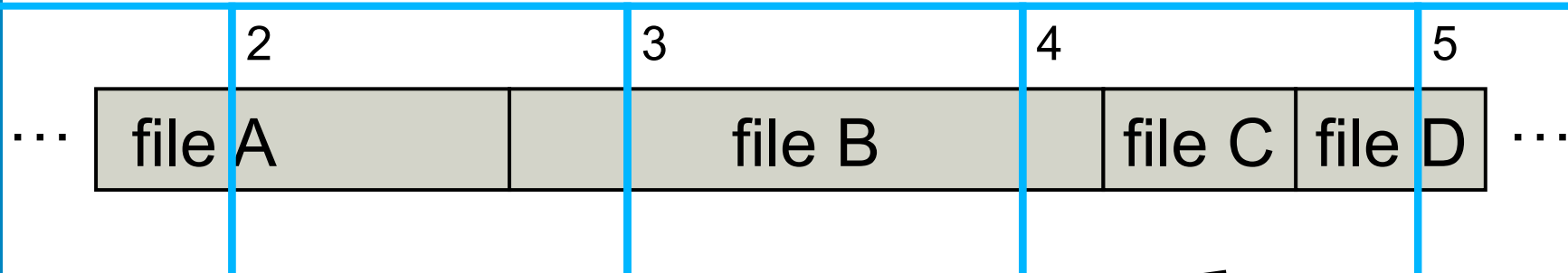
Chunks not shown, real segments much longer

# Deduplicate one segment at a time

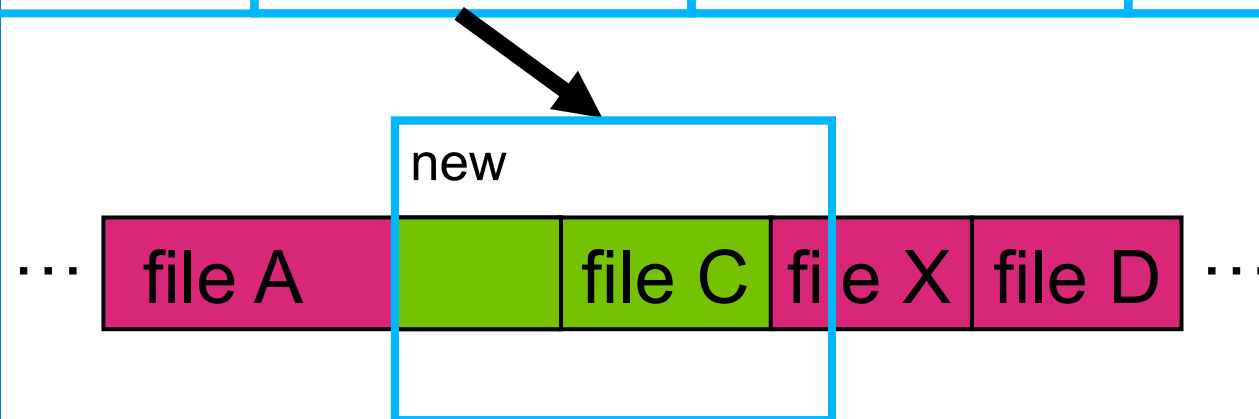
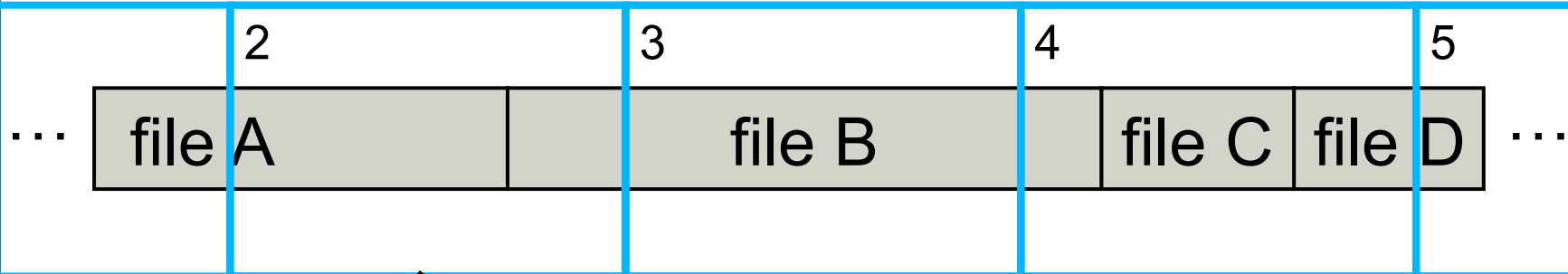


Against a few carefully chosen champion segments

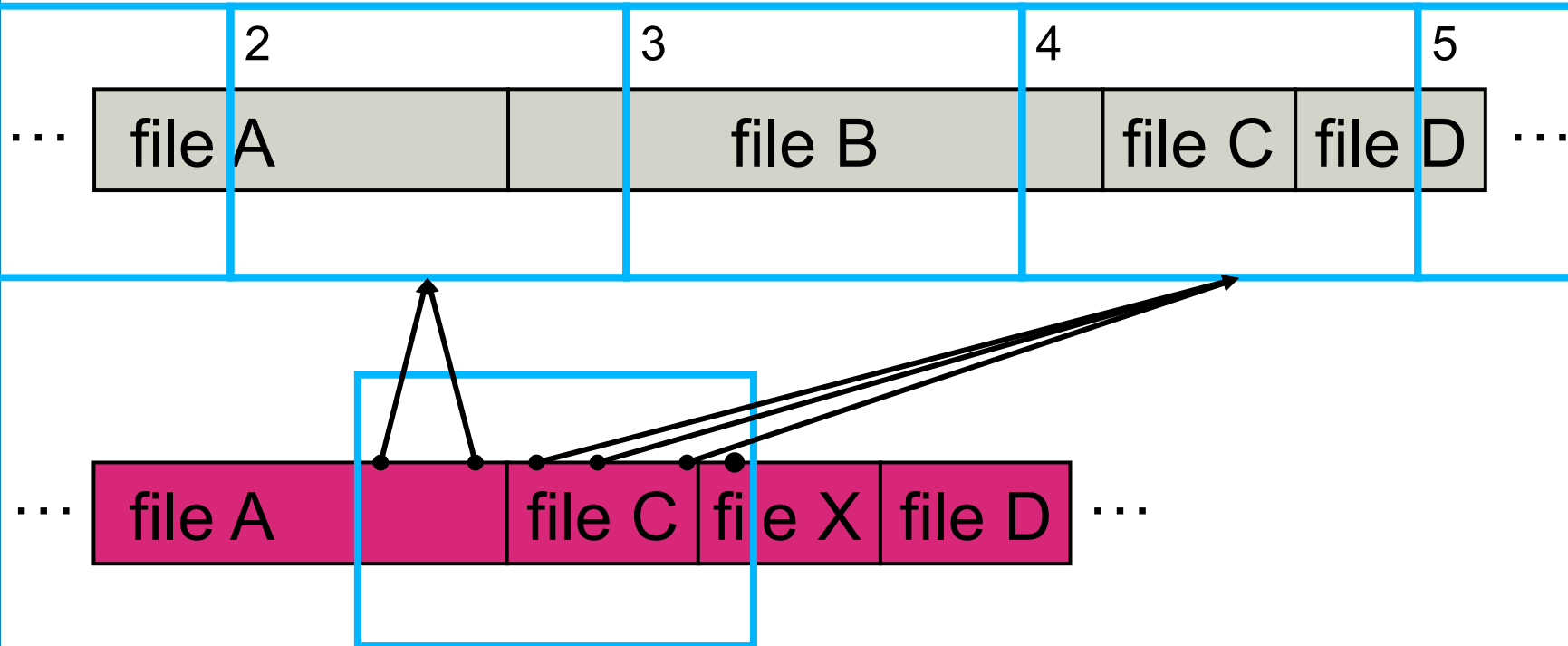
# Champion #1: the most similar segment



# Champion #2: most similar to remainder



# Finding similar segments by sampling

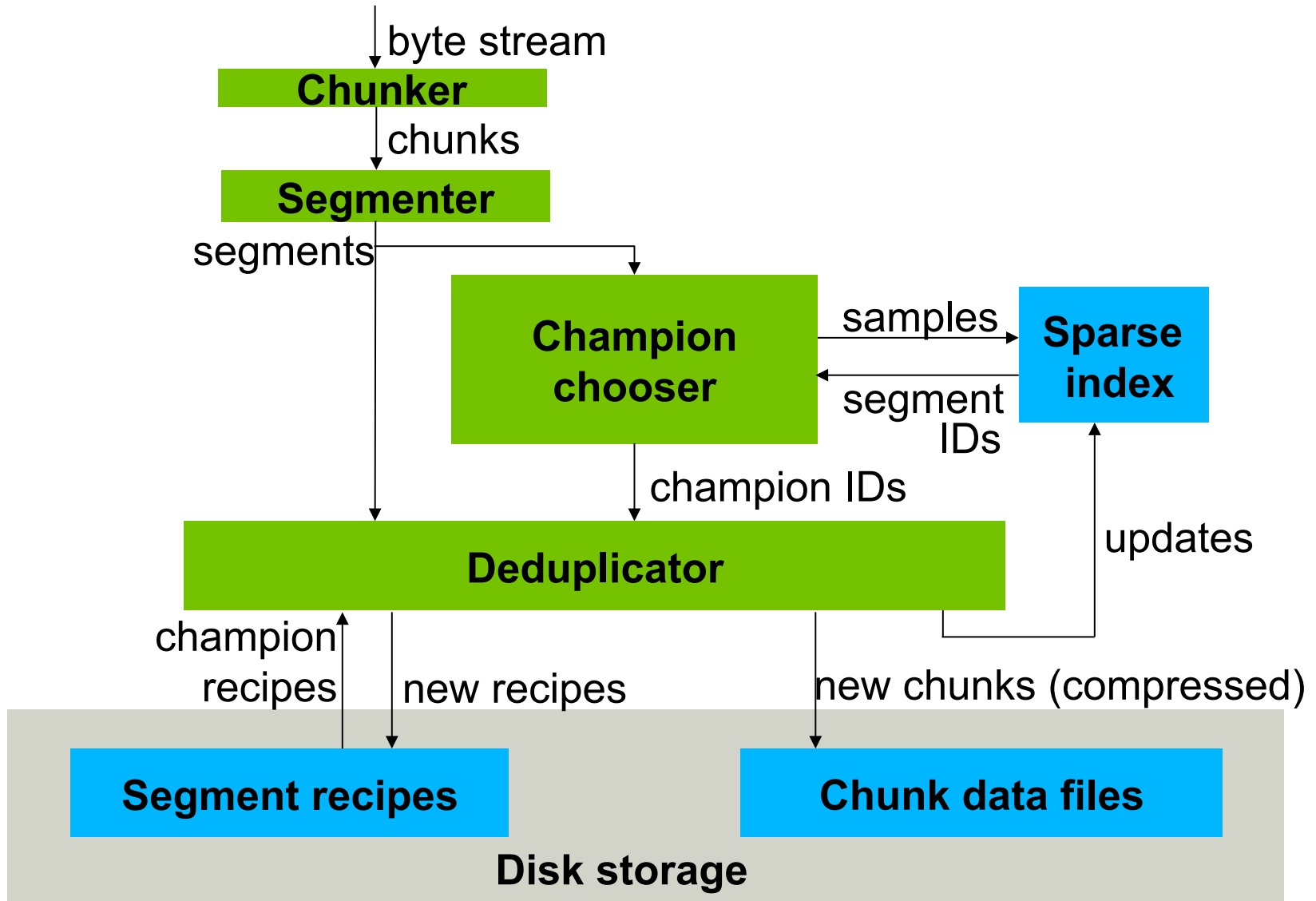


Sparse Index: samples  $\bullet \rightarrow$  containing segment(s)

# A few details

- Also keep segment *recipes*:
  - list of pointers to a segment's chunks
- Actually deduplicate against champion recipes
- Better with variable-sized segments
  - boundaries based on landmarks (“superchunks”)
  - reduces number of champions required

# Putting it all together





# Results

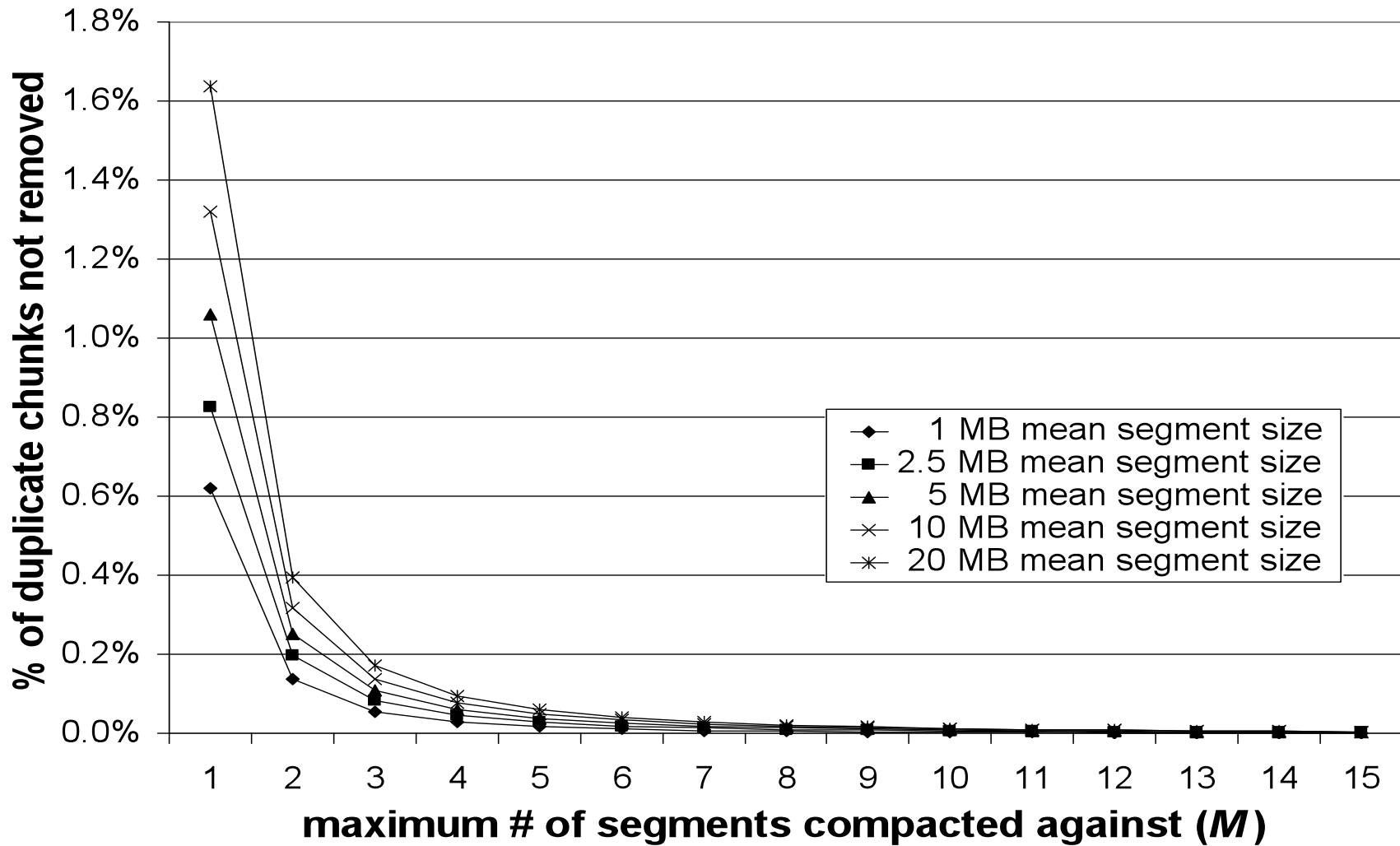
# Methodology

- Built a simulator
- Fixed parameters:
  - 4 KB mean chunk size
  - variable-size segments
  - maximum of 1 segment ID kept per sample
- Varying parameters:
  - mean segment size
  - sampling rate
  - maximum number of champions per segment (M)

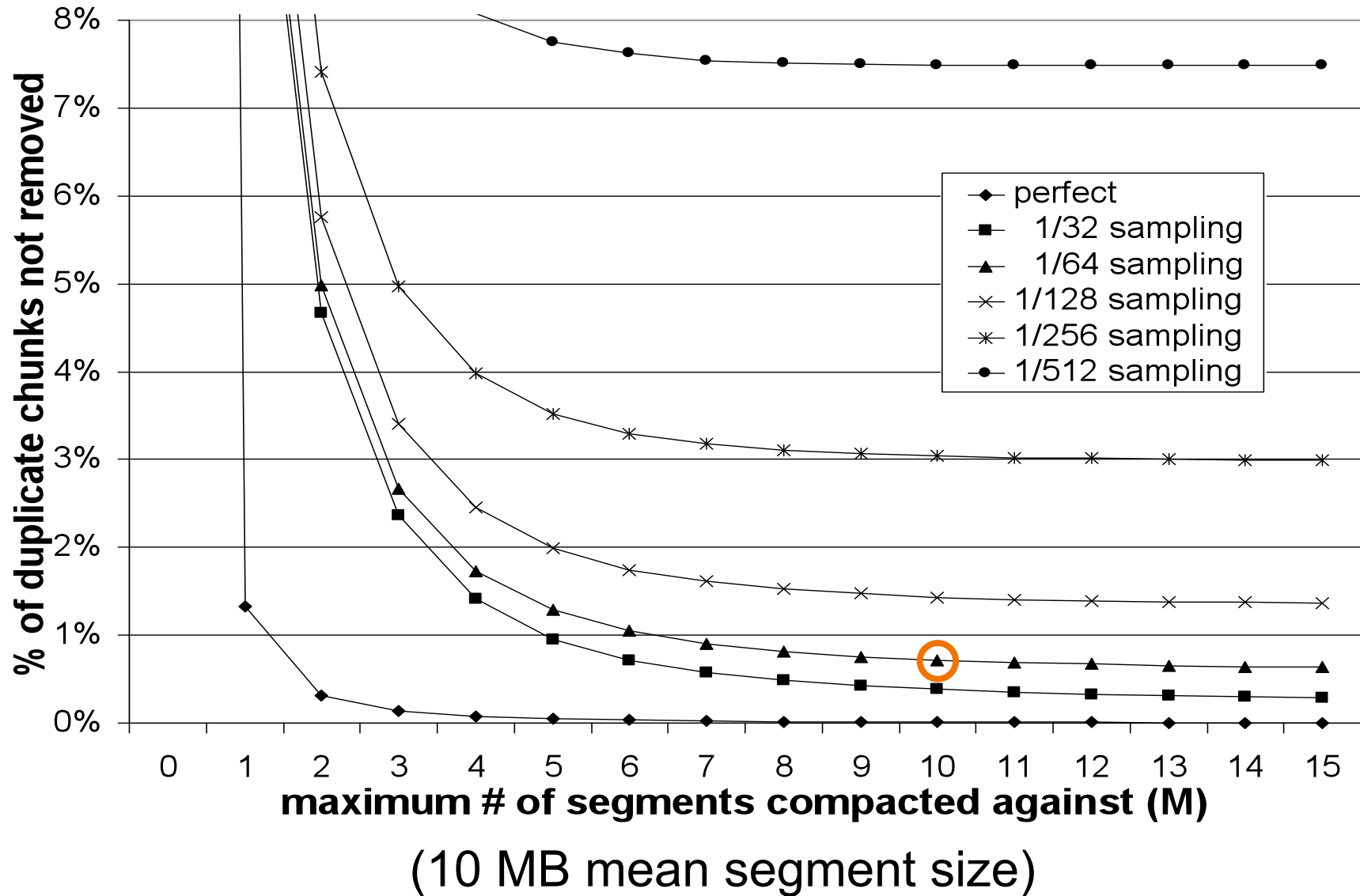
# The data sets

- **Workgroup** [this talk] 3.8 TB
  - backups of 20 desktop PCs belonging to engineers
  - semi-regular backups over 3 months via tar
  - 154 full backups and 392 incremental backups
  - end-of-week full backups are synthetic
  
- **SMB** [see paper] 0.6 TB
  - backups of a server with
    - real Oracle data
    - synthetic Microsoft Exchange data
  - two weeks

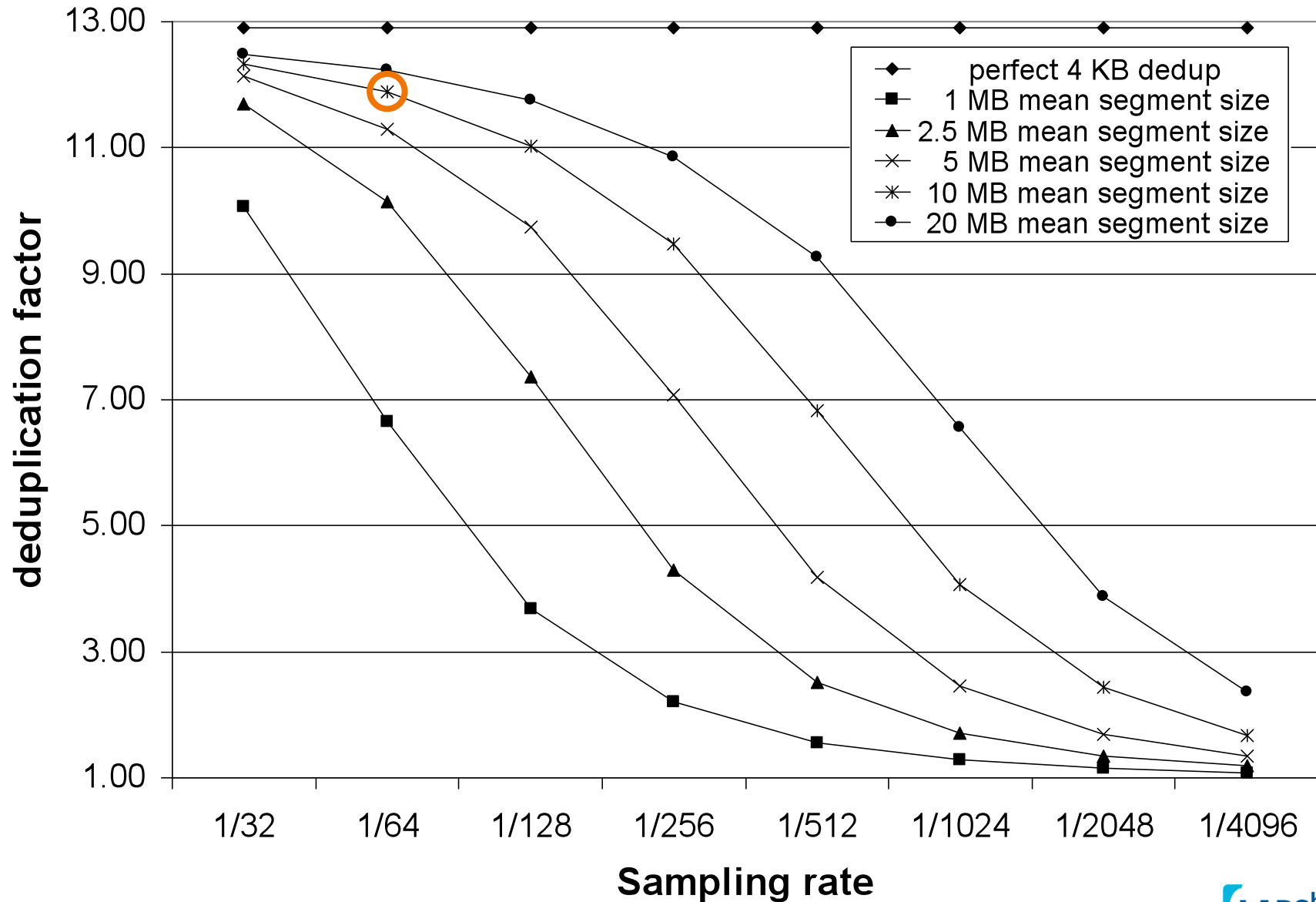
# Chunk locality exists...



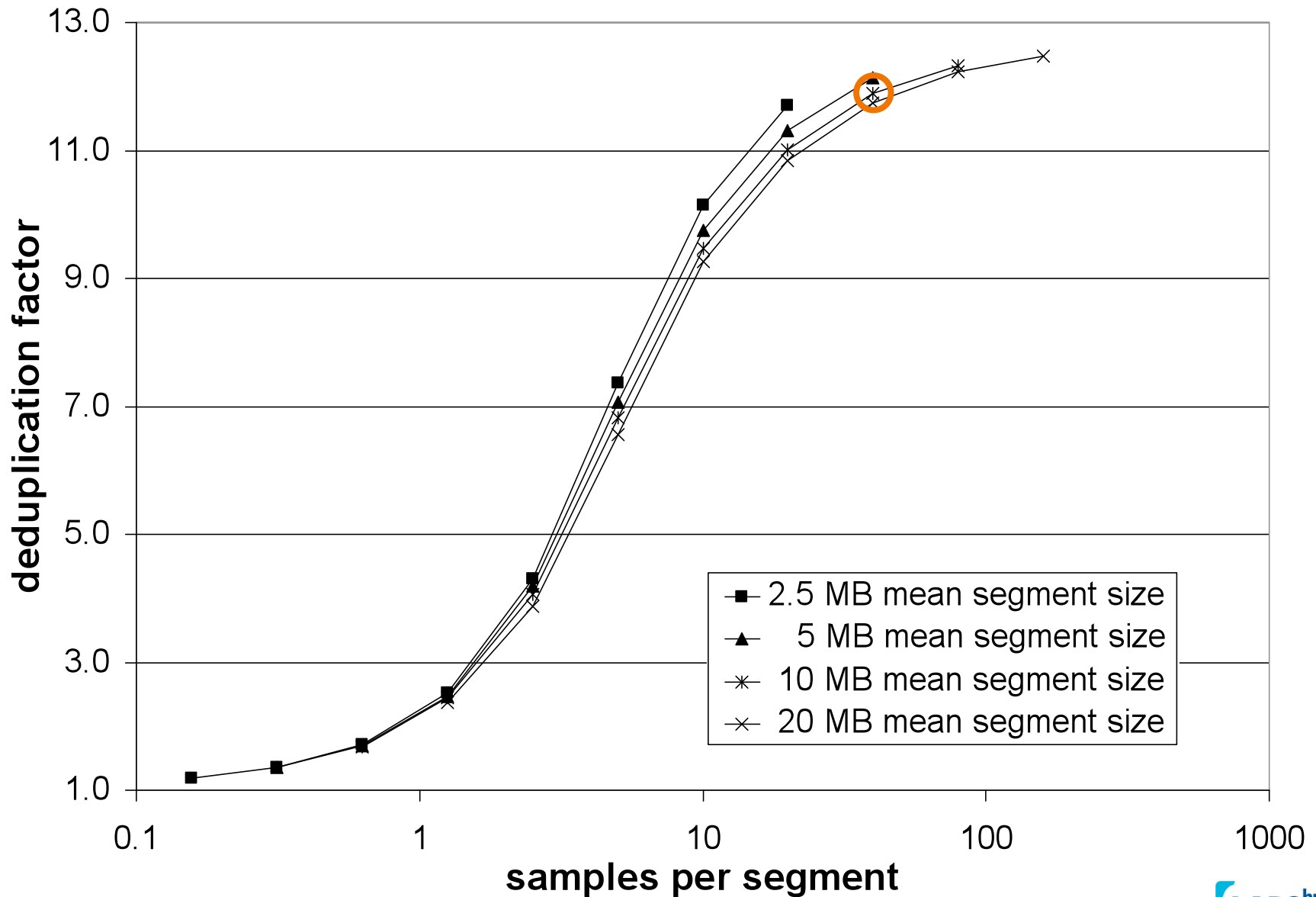
# Sampling can exploit most of it...



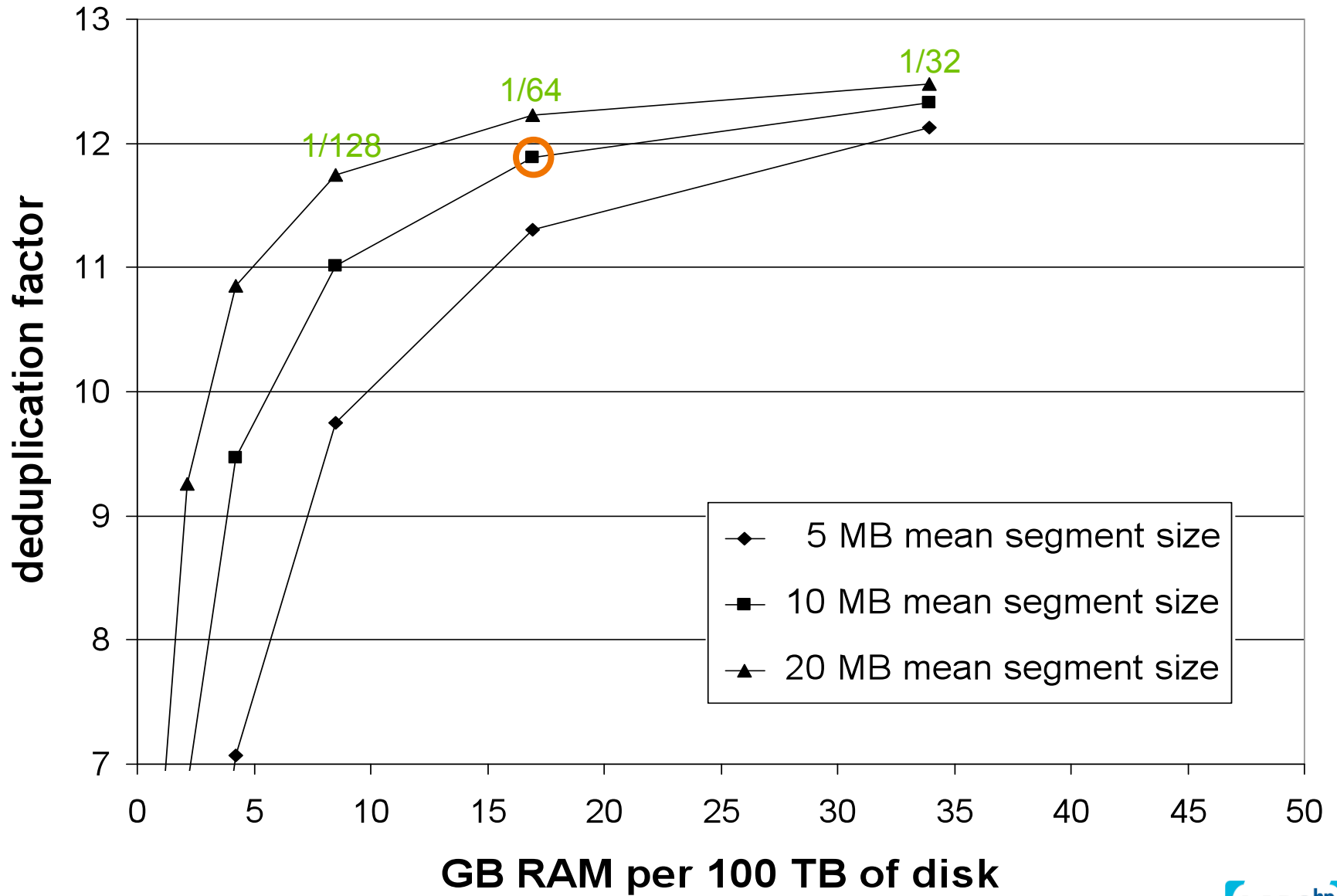
# Deduplication with at most 10 champions



# Deduplication depends primarily on...



# Index RAM usage





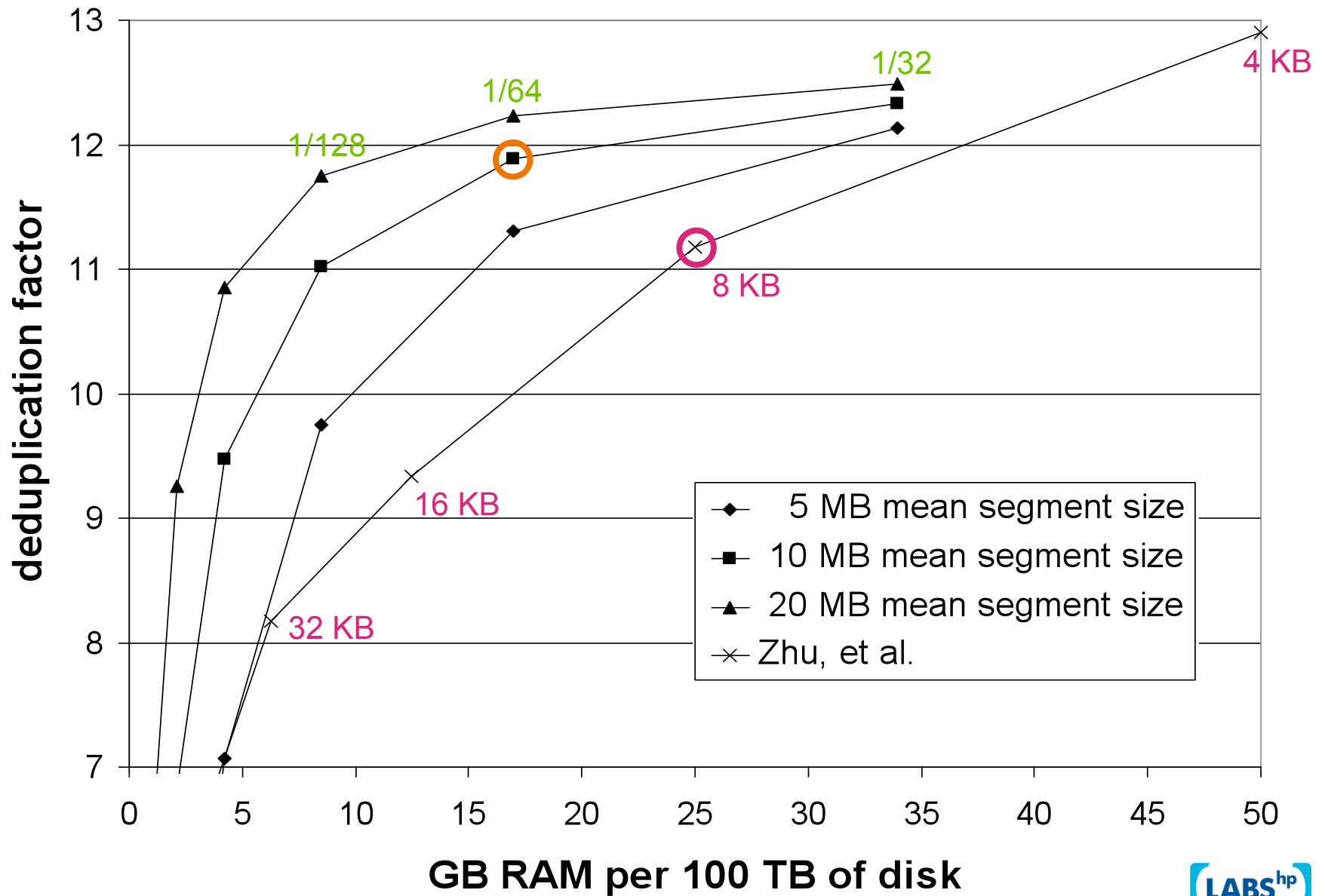
# Comparison with Zhu, et al.

- Their chunk lookup:
  - bloom filter: might the store have a copy?
  - cache of chunk container indexes
  - full on disk index

# Comparison with Zhu, et al.

- Their chunk lookup:
  - bloom filter: might the store have a copy?
  - cache of chunk container indexes
  - full on disk index
- When chunk locality is poor,
  - deduplication quality remains constant
  - but throughput degrades
- Find all duplicate chunks
  - but larger chunk size

# Ram usage comparison



# What about all those disk accesses?

- Infrequent due to batch processing
- **Example:**
  - load at most **10** champions per **10 MB** segment
  - average of 1.7 champions per 10 MB segment
  - = 0.17 champions/MB
  - = 1 seek per 5 MB
- I/O burden:
  - 20 ms to load a champion recipe (~100 KB)
  - → 1 drive can handle > 250 MB/s ingestion rate

Thank You