Anchor-driven Subchunk Deduplication

Bartłomiej Romański
Łukasz Heldt
Wojciech Kilian
Krzysztof Lichota
Cezary Dubnicki

9LivesData LLC

Who we are

- 9LivesData
  - R&D company based in Warsaw, Poland
  - 50+ scientists and software engineers
  - designers/coders of HYDRAstor backend for NEC

- HYDRAstor
  - scalable, content-addressable backup storage
  - global dedup, self-healing
  - owned by NEC, on sale in the USA and Japan
  - started by 9LivesData founder in Princeton, NJ
  - fastest and largest dedup system (Curtis W. Preston analysis)

HYDRAstor

Problem statement

- System model
  - Block store
  - Clients writing data streams (backup)
- Goals
  - Maximize amount of data kept in the system
  - Measured using duplicate elimination ratio (DER)
    - DER = data written/data physically stored

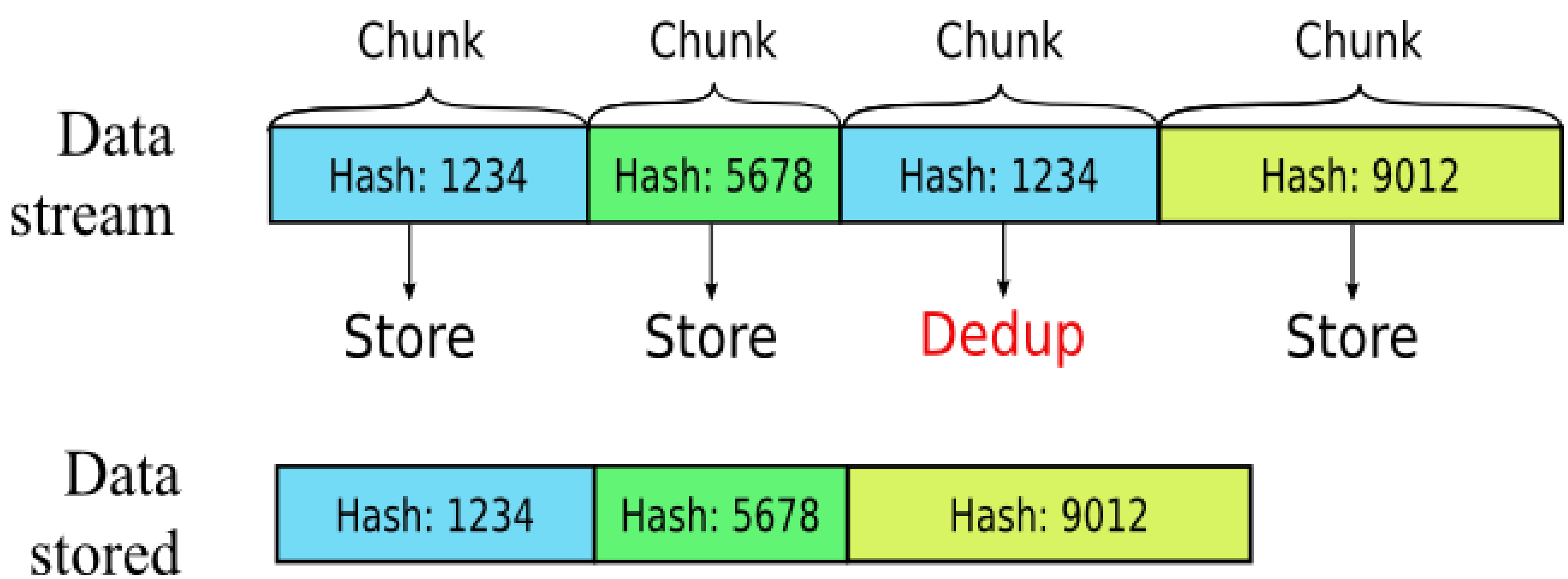
Subchunk deduplication aims at maximizing DER.

Outline

- Quick introduction to deduplication and chunking
- Subchunk deduplication
- Results of simulations
- Conclusions

Content-based deduplication

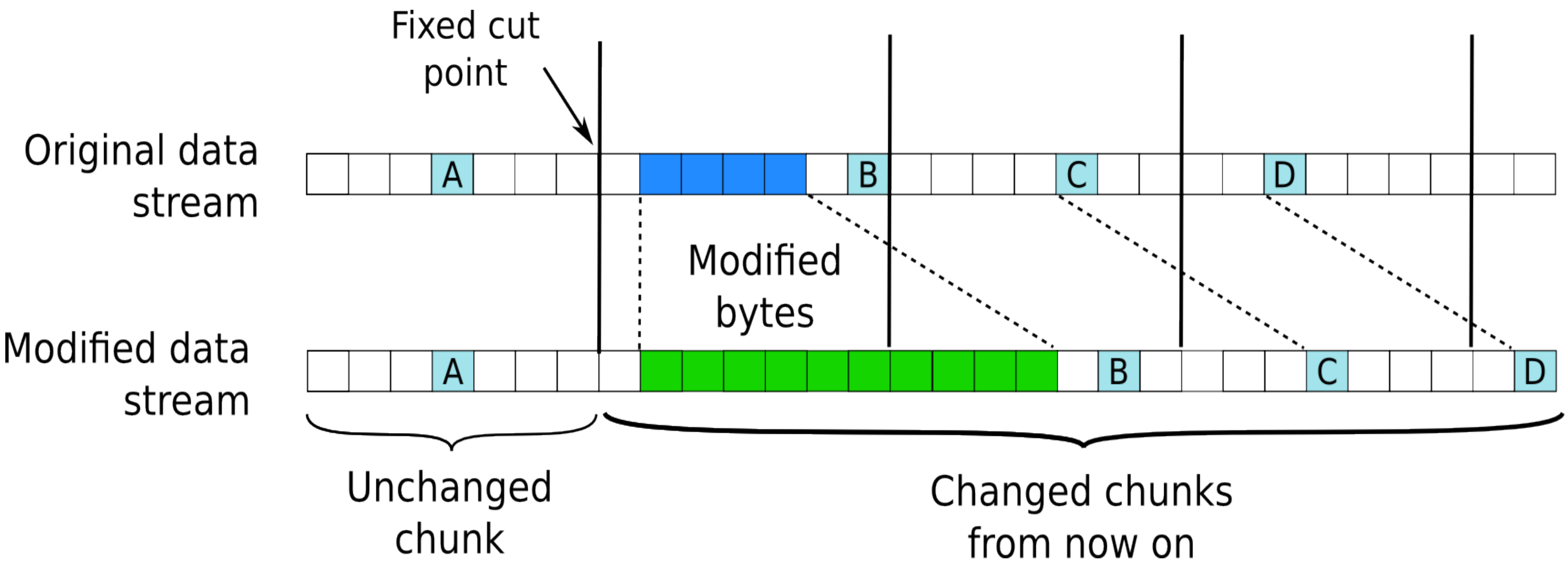
- Cut the data into chunks (sequences of bytes)
- Compute hash (e.g. SHA-1) on each chunk
- Check if hash exists in block store
  - Exists – deduplication
  - Otherwise – store



HYDRAstor

Fixed-size chunking problem

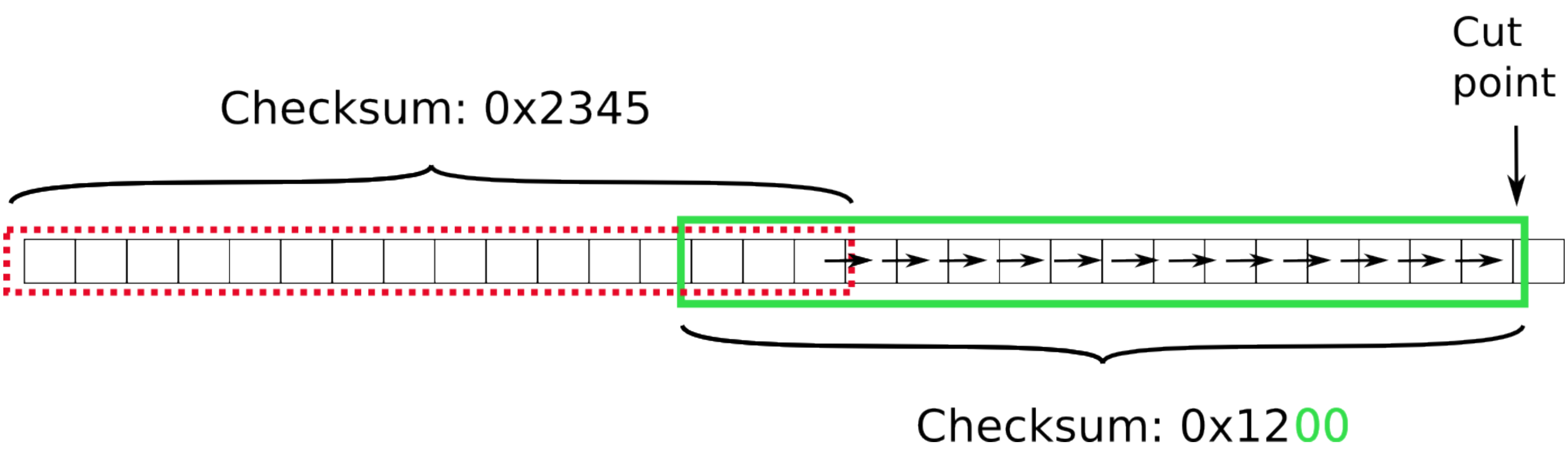
- Fixed-size chunks have problems
  - Insertions/deletions break dedup



- Standard solution: content-defined chunking (CDC)

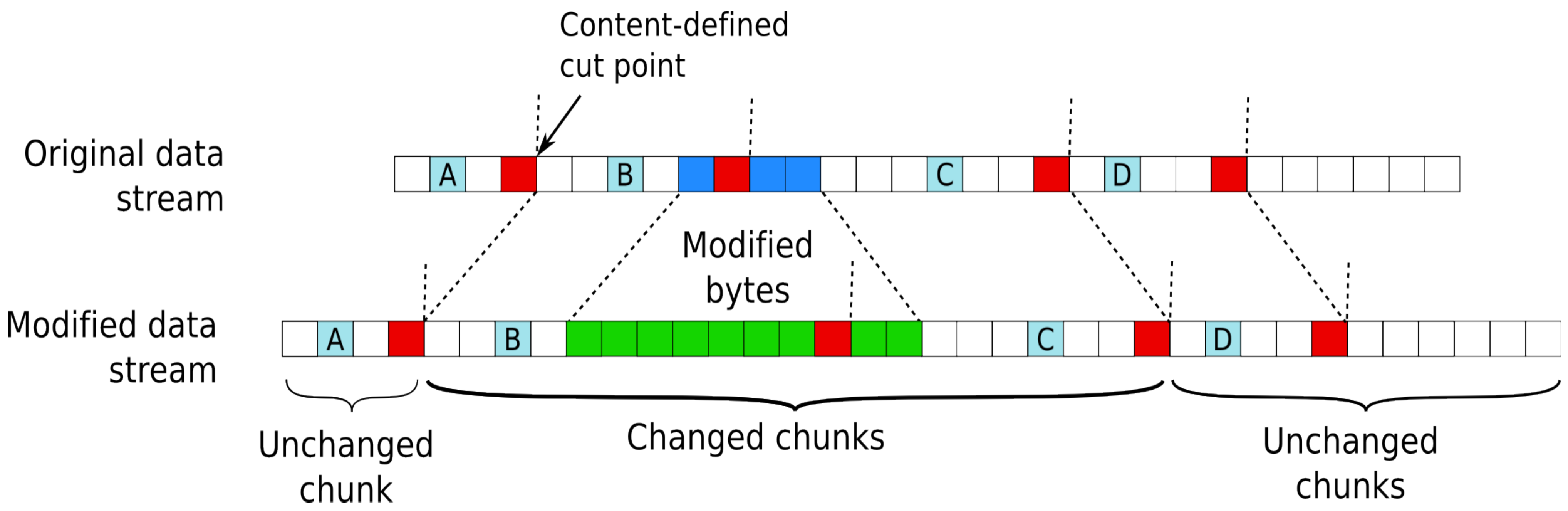
Content-defined chunking (CDC)

- Move sliding fixed-size window over input bytes
- Compute checksum over window bytes
- If checksum's last *X* bits are zeroes – cut at this point



Content-defined chunking (CDC)

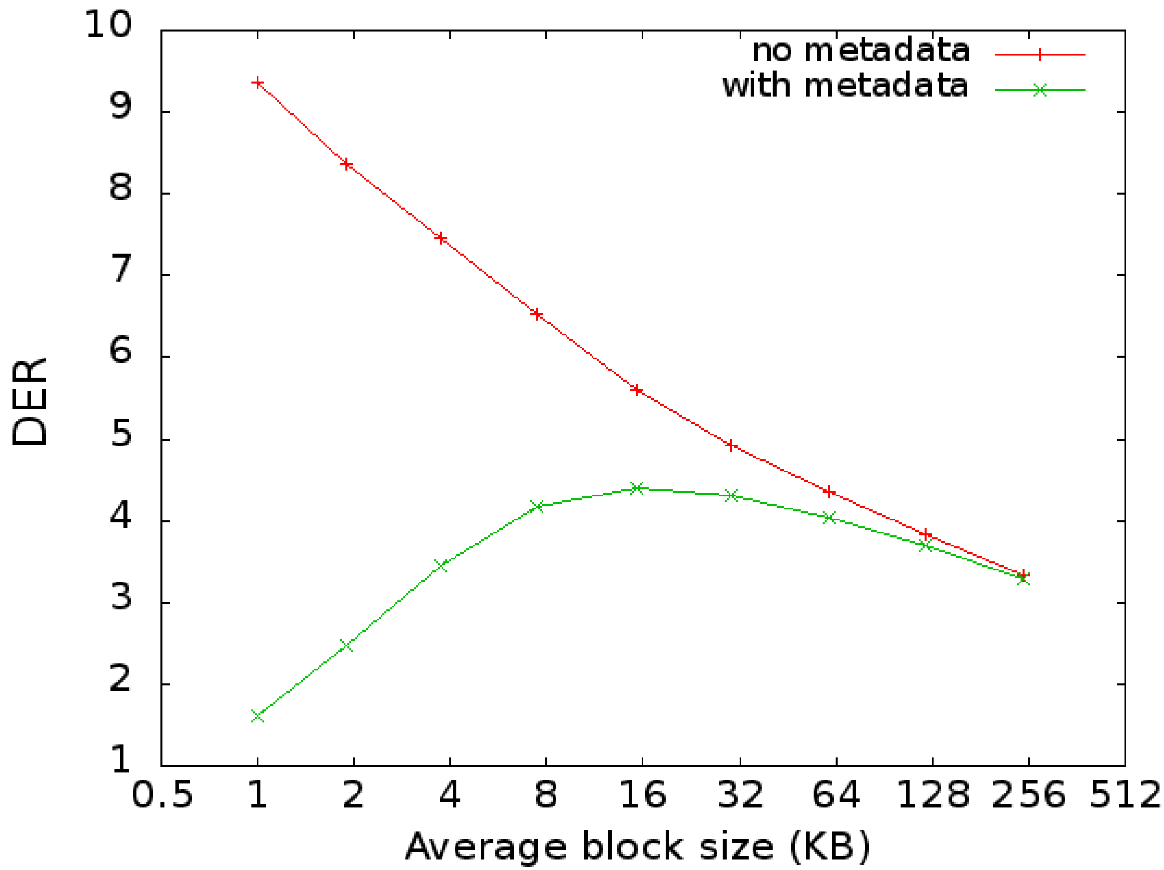
- Cut points happen every 2^X bytes on average (expected value for random data)
- Cut points usually preserved by insertions/deletions



HYDRAstor

Deduplication vs chunk size

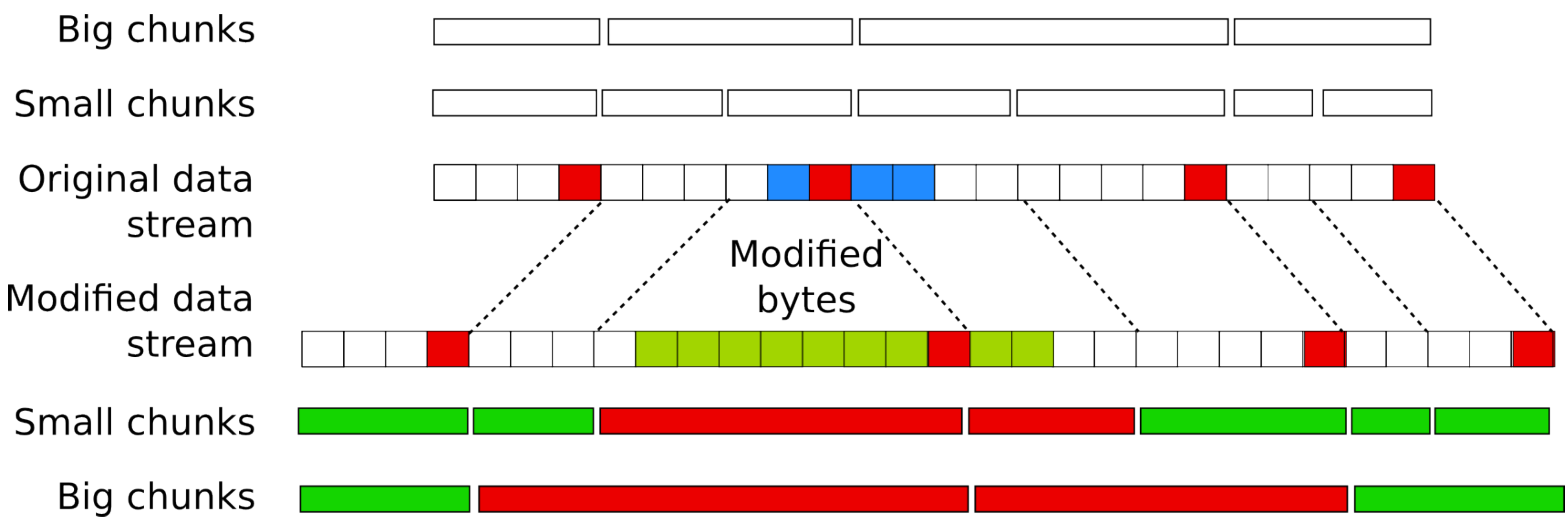
- The smaller the chunk size, the better deduplication
- But: short chunk size impractical due to metadata overhead and other reasons



HYDRAstor

Conclusions from CDC – use 2 chunk sizes

- Big chunks – smaller overhead, worse raw dedup
- Small chunks – bigger overhead, better raw dedup
  - Use big chunks where possible
  - Use small chunks to improve dedup in areas of change



HYDRAstor

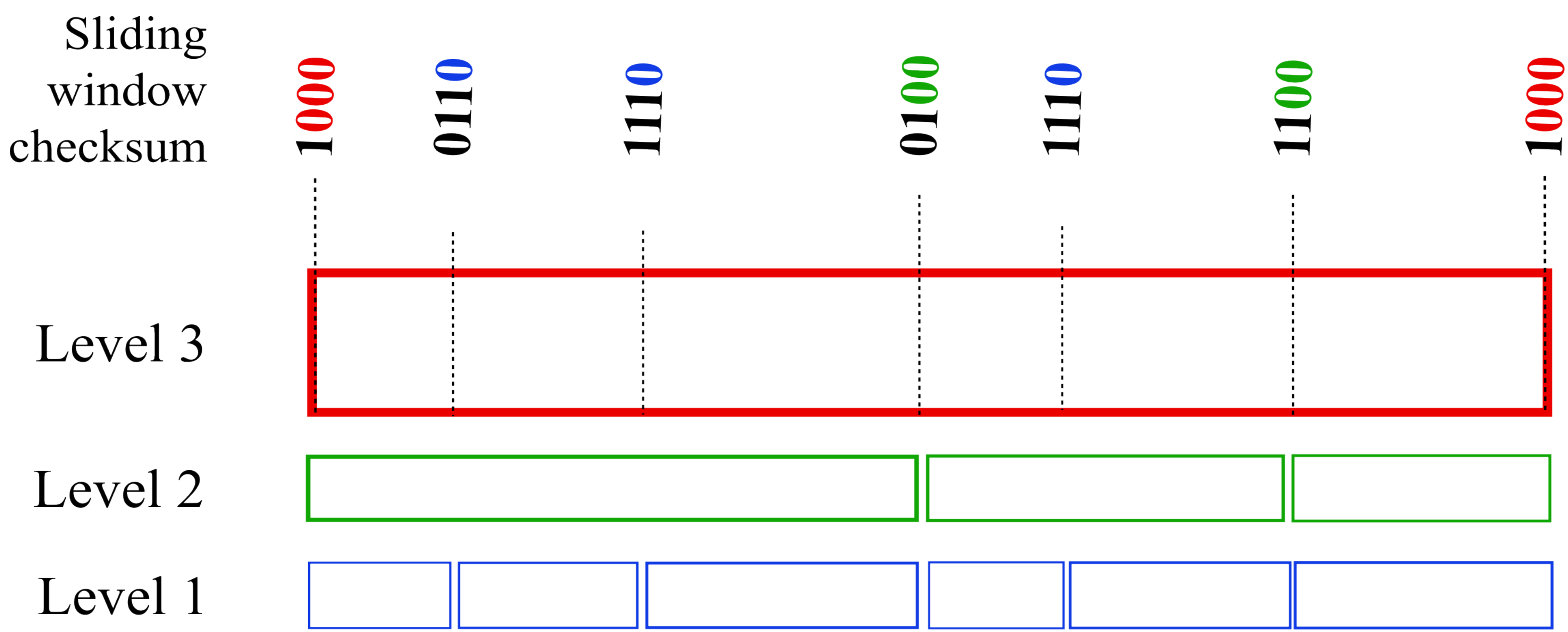
Conclusions from CDC – avoid small chunks overhead

- Small chunks have higher metadata overhead
  - Per-chunk metadata is constant
  - Metadata overhead spoils dedup ratio
- Small chunks cause worse performance
  - Per-chunk processing has constant factor
- Conclusions
  - Avoid small chunks metadata overhead
  - Process big chunks not small chunks

Subchunk deduplication

Subchunk definition

- Observation: all chunks created with X+1 trailing zeroes are also chunks of level X (i.e. cut points for avg. 64 KB chunks are also cut points for 32 KB, 16 KB, 8 KB, ...)
- A chunk can be split into subchunks in deterministic way



Main idea of subchunk dedup

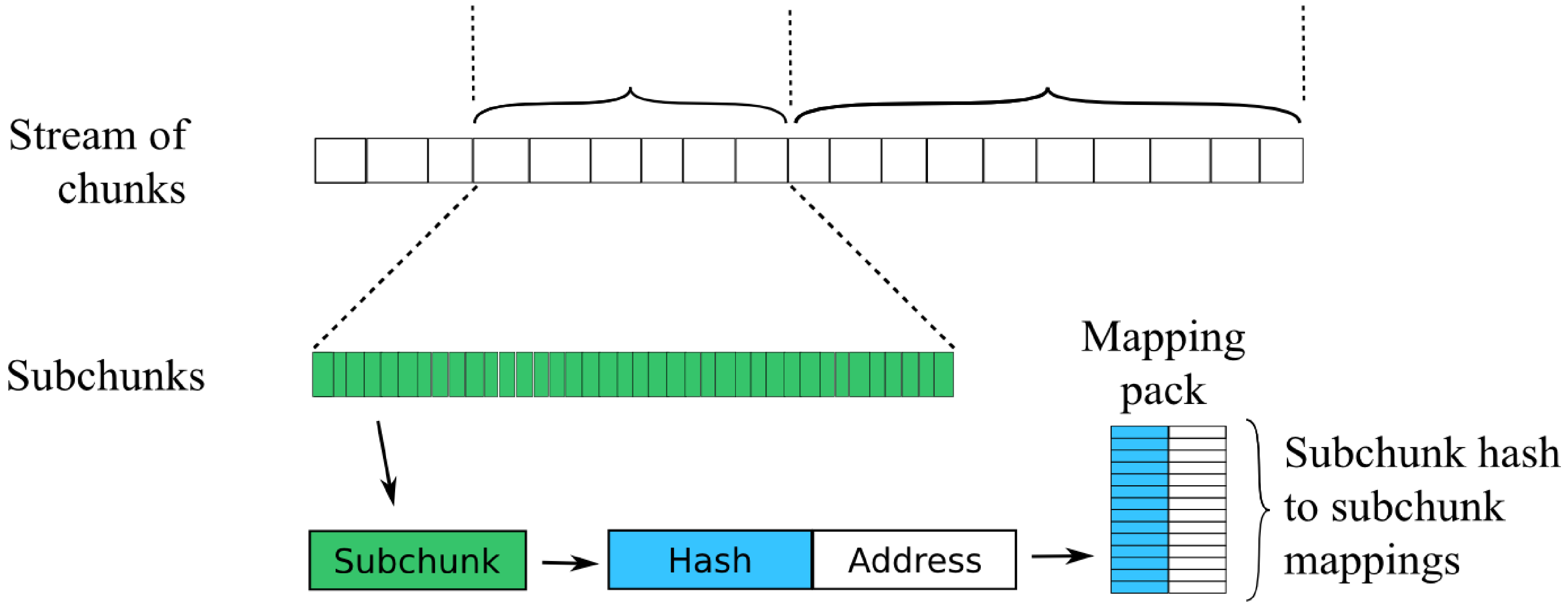
- Use global index to locate big chunks
  - Dedup against all data in the system
- Use subchunks instead of small chunks
  - Subchunk share metadata with container chunk
- Use additional structure to locate subchunks

HYDRAstor

Locating subchunks

- Deduplication against all subchunks costly
  - Too many subchunks
- Duplicates are usually local to data stream
- Solution
  - Split subchunks index into parts (***mapping packs***)
  - Use only parts relevant to current data stream
  - Load proper index parts dynamically during dedup (build ***dedup context*** for current data stream)

Splitting subchunk index into mapping packs



- Mapping packs are stored in block store

Subchunk deduplication algorithm

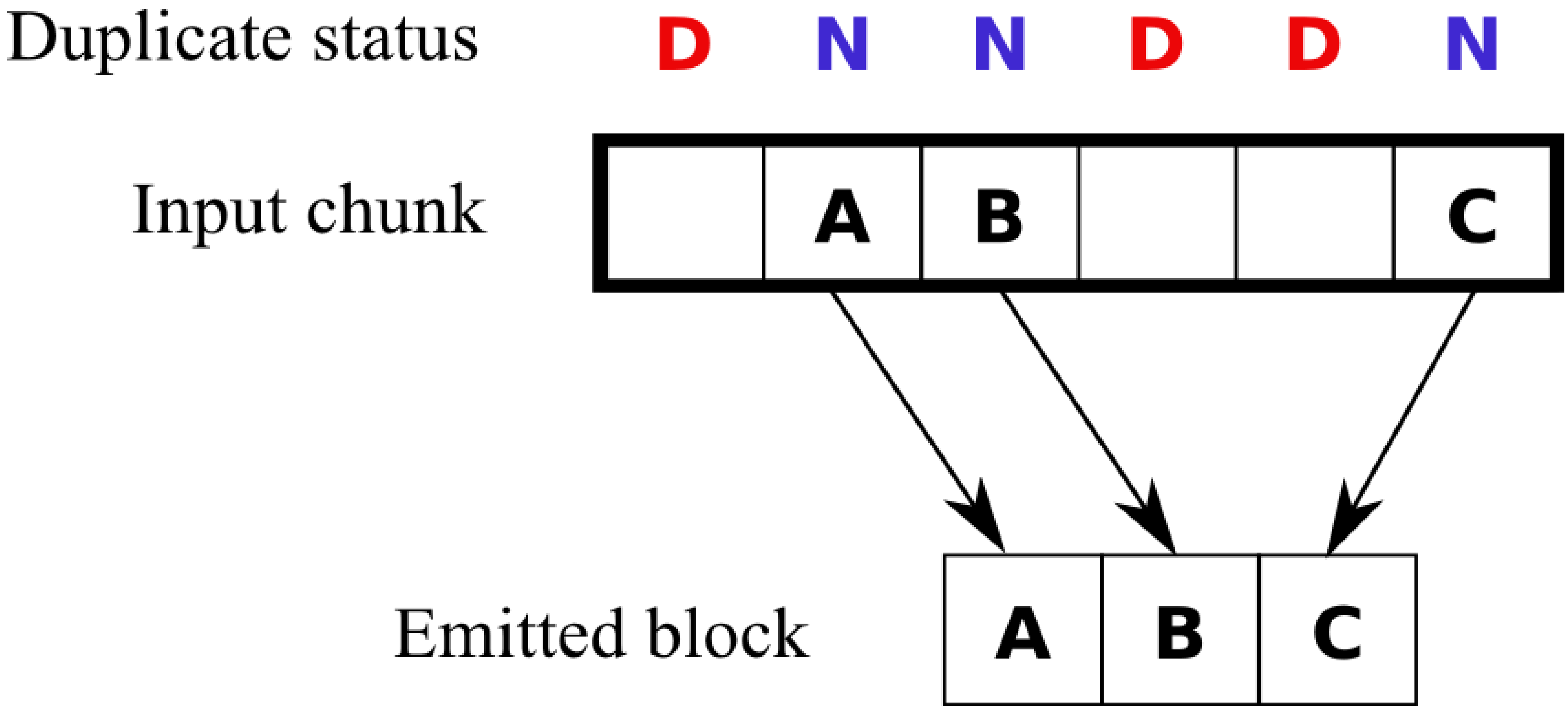
1. Chunk the input stream into big chunks and each chunk into subchunks

2. Store hashes of subchunks in *mapping pack* for future dedup

3. Using global index check if big chunk exists, if not:
   - 3.1. Check if each subchunk exists in *dedup context*
   - 3.2. Emit non-duplicate subchunks as one block

Note: algorithm works with base dedup even when subchunk mappings do not exist, so mapping packs are disposable

Subchunk emission



Subchunk deduplication context

- Runtime cache of subchunk hashes to subchunks
- Stored in RAM
- Constant size
  - LRU per mapping
- Updated by loading mapping packs
- Should keep mappings relevant for incoming backup stream

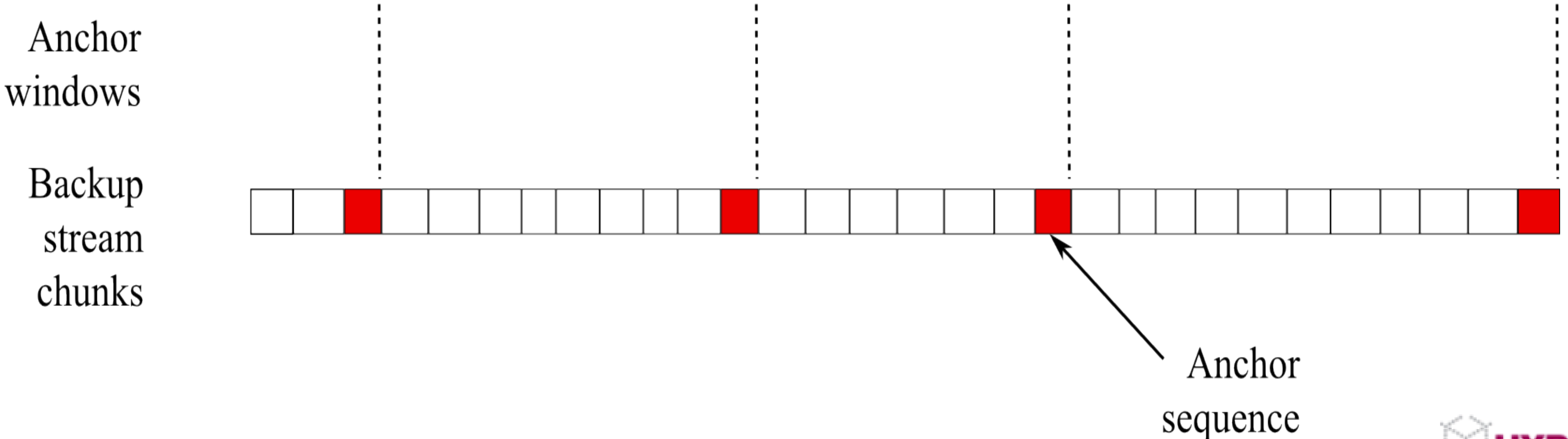
HYDRAstor

Locating mapping packs

- Problem: when writing a stream, how to find mapping packs which likely contain mappings for incoming data?
- We do not assume knowledge of data streams relations
- We need to be able to handle changes in data streams

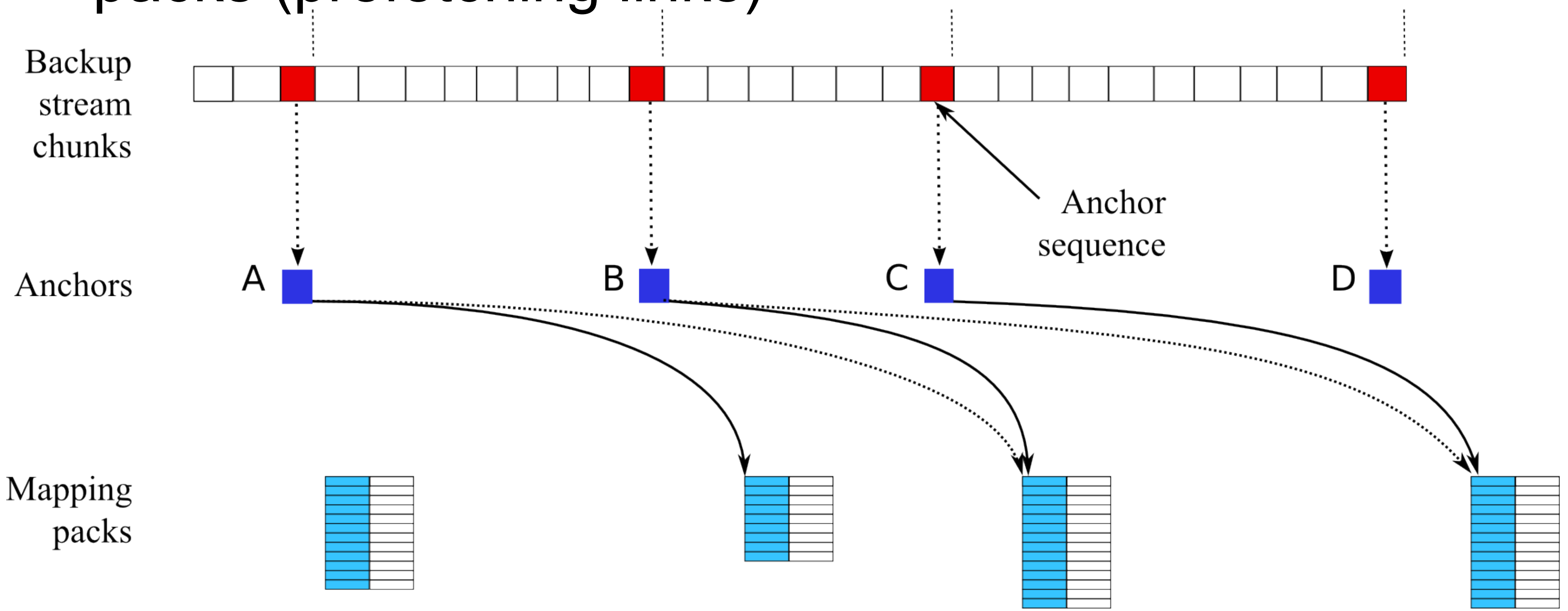
Splitting stream into windows

- Apply content-defined chunking to chunk **hashes,** instead of bytes (with window size = 1)
- Anchor sequence – block whose hash has X trailing 0 bits
- Anchor window – data chunks between 2 anchor sequences
- Anchor sequences usually kept in case of insertions/deletions



Locating mapping packs using anchors

- Anchor - special block corresponding to anchor sequence
  - addressed with anchor sequence address
- Each anchor keeps pointers to multiple (N) mapping packs (prefetching links)

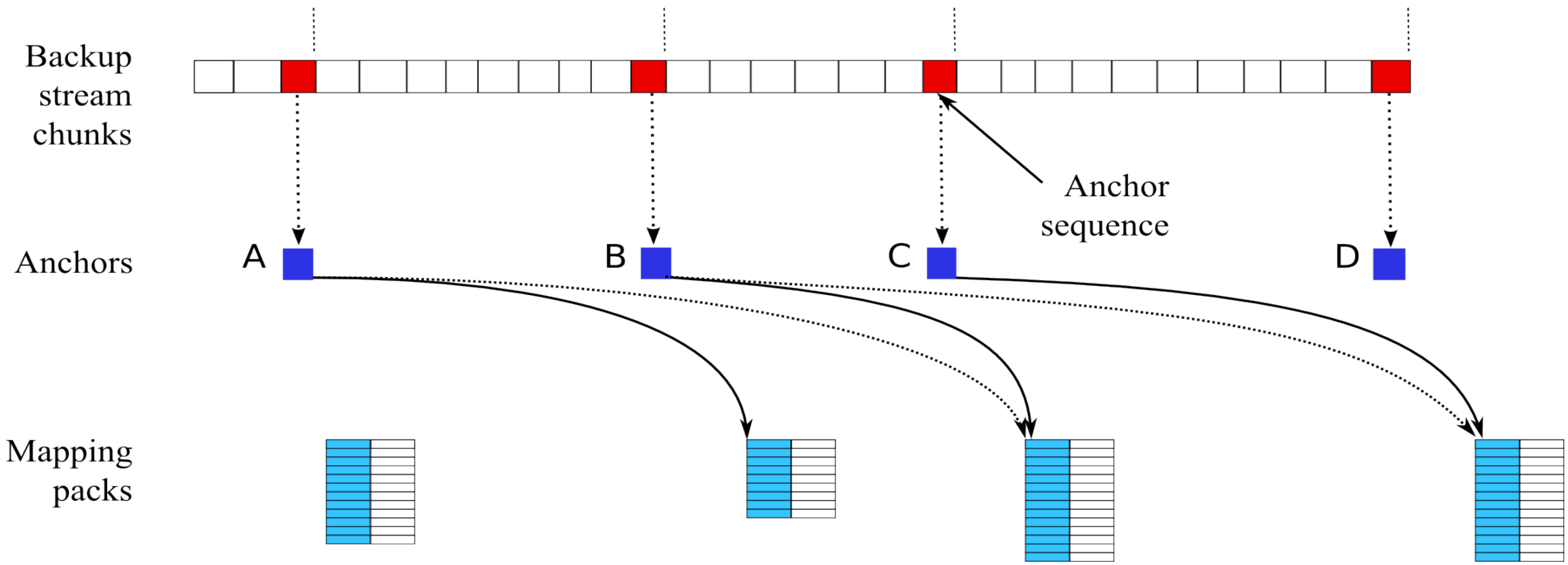


HYDRAstor

Mapping packs and dedup context update

When anchor sequence is spotted in data stream:

1. Finish writing current mapping pack to block store
   - store pointer to pack with the previous anchors
   - emit anchors with sufficient prefetching pointers
2. Prefetch mapping packs for anchor into dedup context
   - remove old mappings from dedup context (LRU)



Simulation results

Results of simulations

- Datasets
  - Netware (backups)
  - Wikipedia snapshots
  - Mailboxes
  - Total

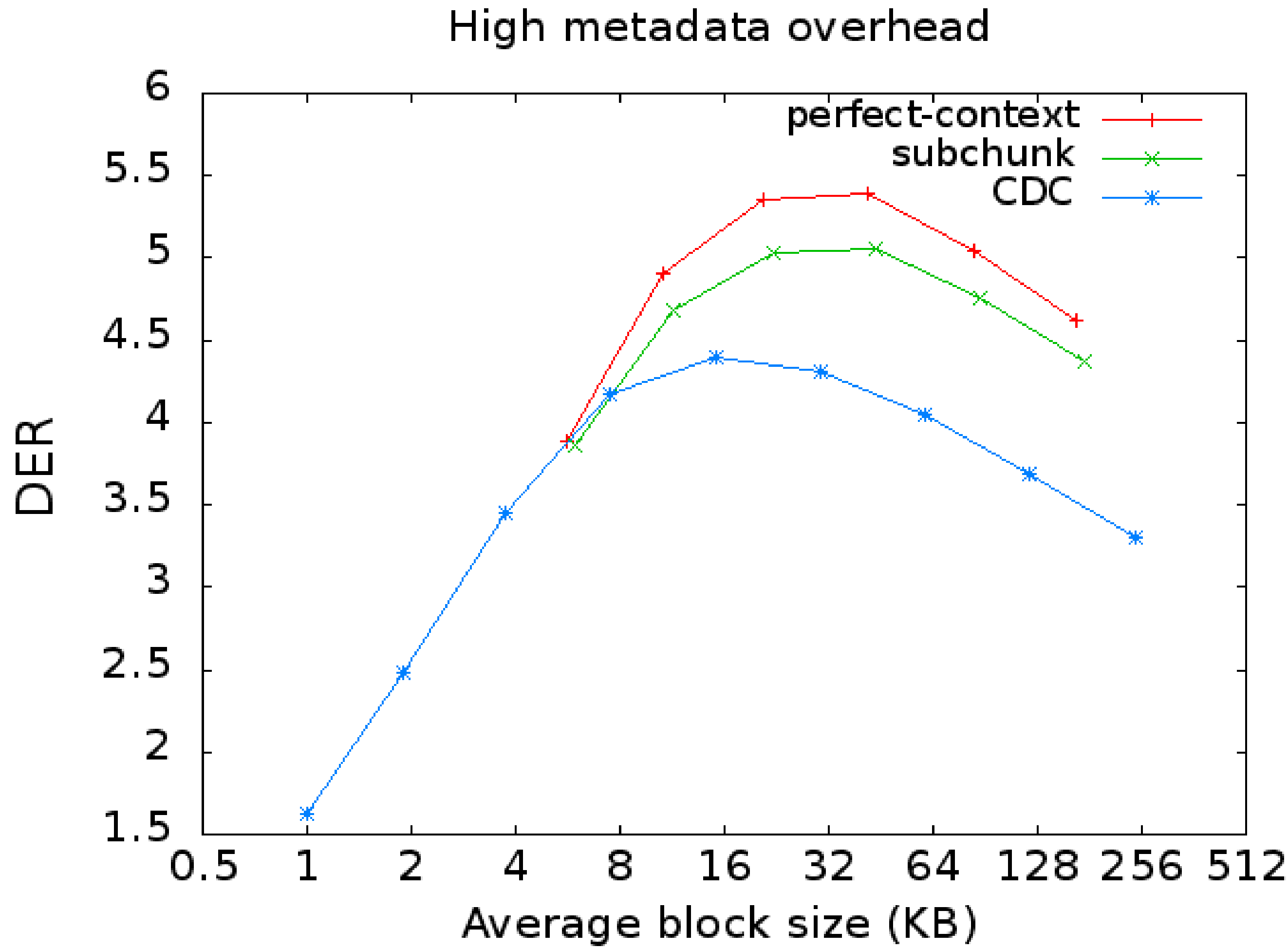
- Metadata
  - Low metadata overhead
  - High metadata overhead

Reasons for high metadata overhead

- High resiliency - distributed system must survive many node failures

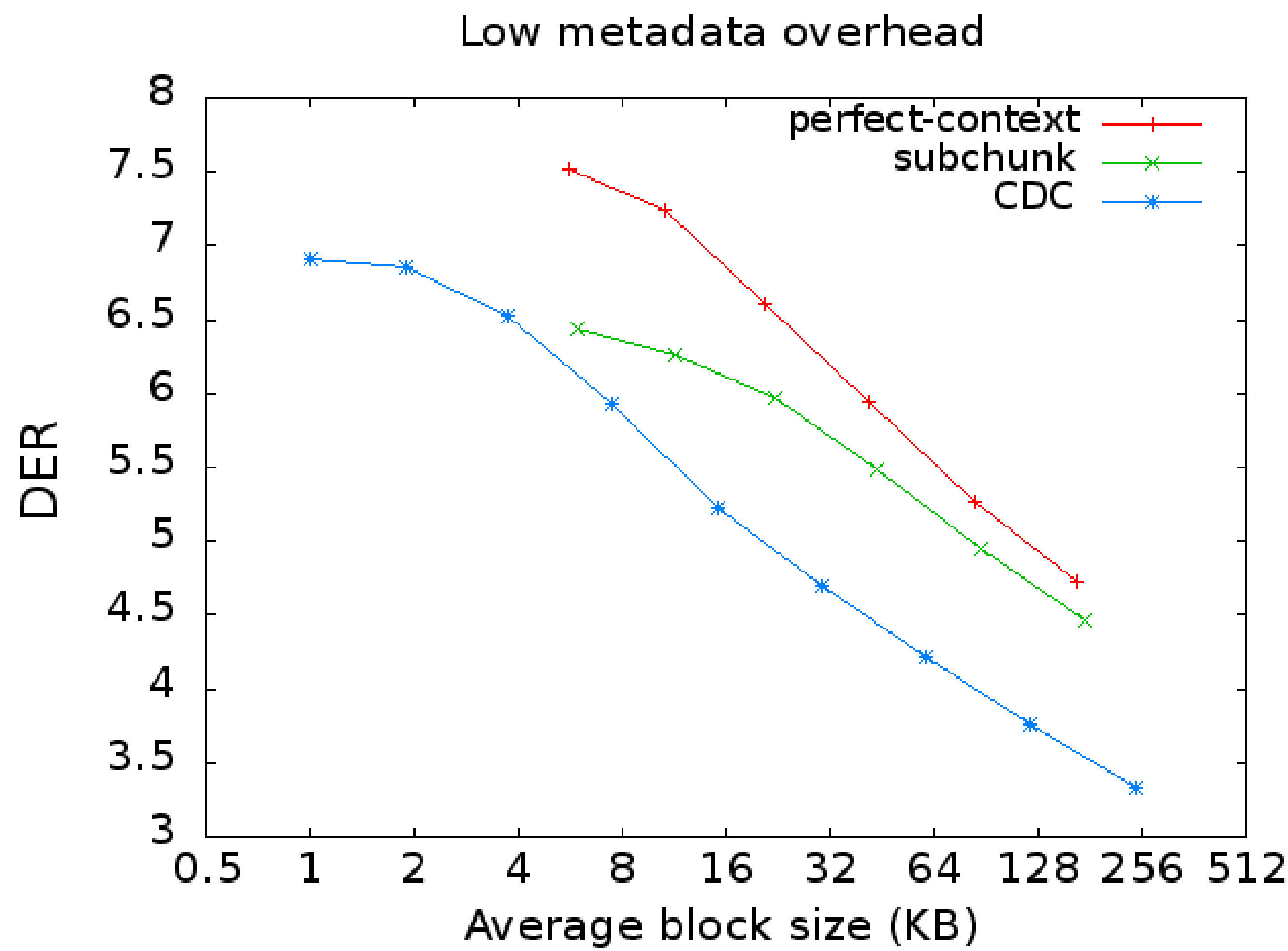
- High availability – many copies of metadata
  - critical operations like deletion need complete metadata

Results for high metadata overhead system



- Expected subchunk size is 1/8 of chunk size
- Subchunk 64/8KB is better than CDC 8KB (by 20%)

Results for low metadata overhead system



- Expected subchunk size is 1/8 of chunk size
- Subchunk 16KB/2KB is better than CDC 8KB (by 6%)

Conclusions

- Better effective deduplication ratio
  - high metadata overhead: +20% vs CDC 8KB
  - low metadata overhead: +6% vs CDC 8KB
- Higher average block size
  - better for performance
- Low metadata overhead for subchunks
- Disposable subchunk structures
  - can be kept with low resiliency
  - only affect deduplication ratio gain
- Good tradeoff between fragmentation and deduplication ratio (details in paper)

Questions?

Thank you!