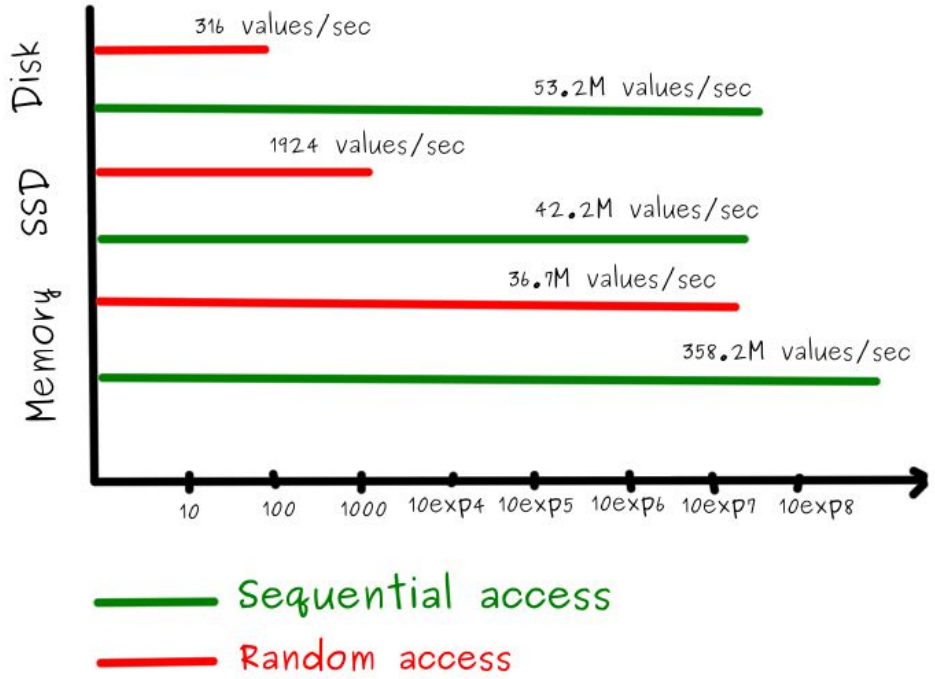
O-O-O-O-O-O-O

HEXAGON

Persistence Axioms

- *Sequential reads and writes are typically faster* than random
- *Random record access on collections of fixed-size records (eg. get(index)) can be very efficient* when based on computing access offsets using mathematical formulas (eg. offset = index * record-size) rather than scanning the data - (O1)
- Data retrieval optimization is typically a matter of *I/O optimization* (eg. minimize # of blocks loaded to resolve a given query)
- Data publishing optimization is typically a matter of *I/O optimization* (eg. minimize # of blocks to upload). Typically data publishing !== data retrieval optimization.
- *Index-free adjacency is the key differentiator* of native graph processing (eg. edges to store direct pointers to adjacent nodes so that fast traversals are possible w/o the help of indices)

Sequential Access vs. Random Access

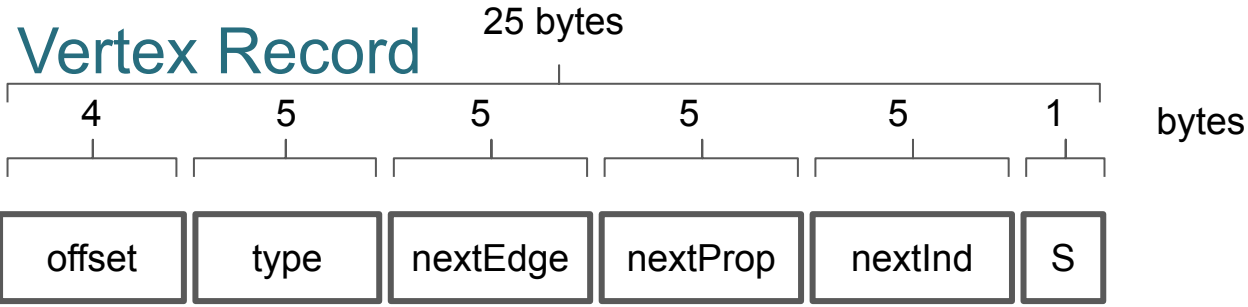


Benchmark published by Adam Jacobs, *The Pathologies of Big Data* in ACM Newsletter

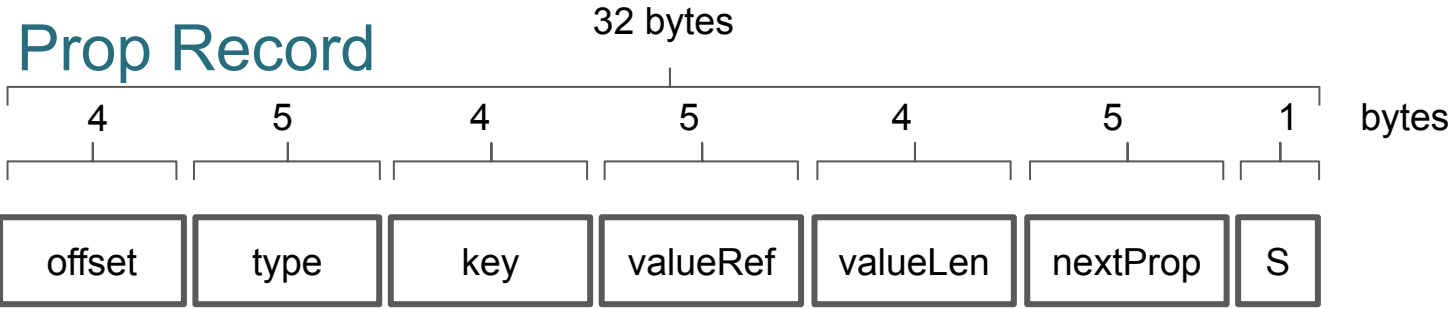
HEXAGON

Persistence Format

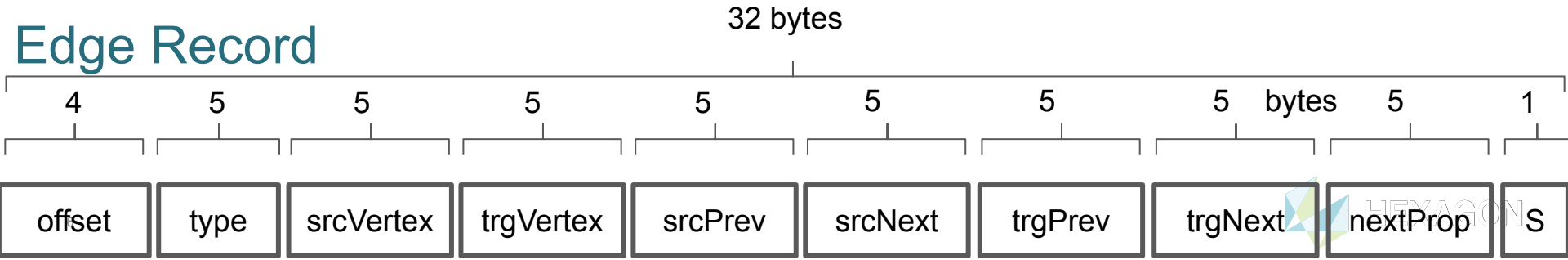
Vertex Record



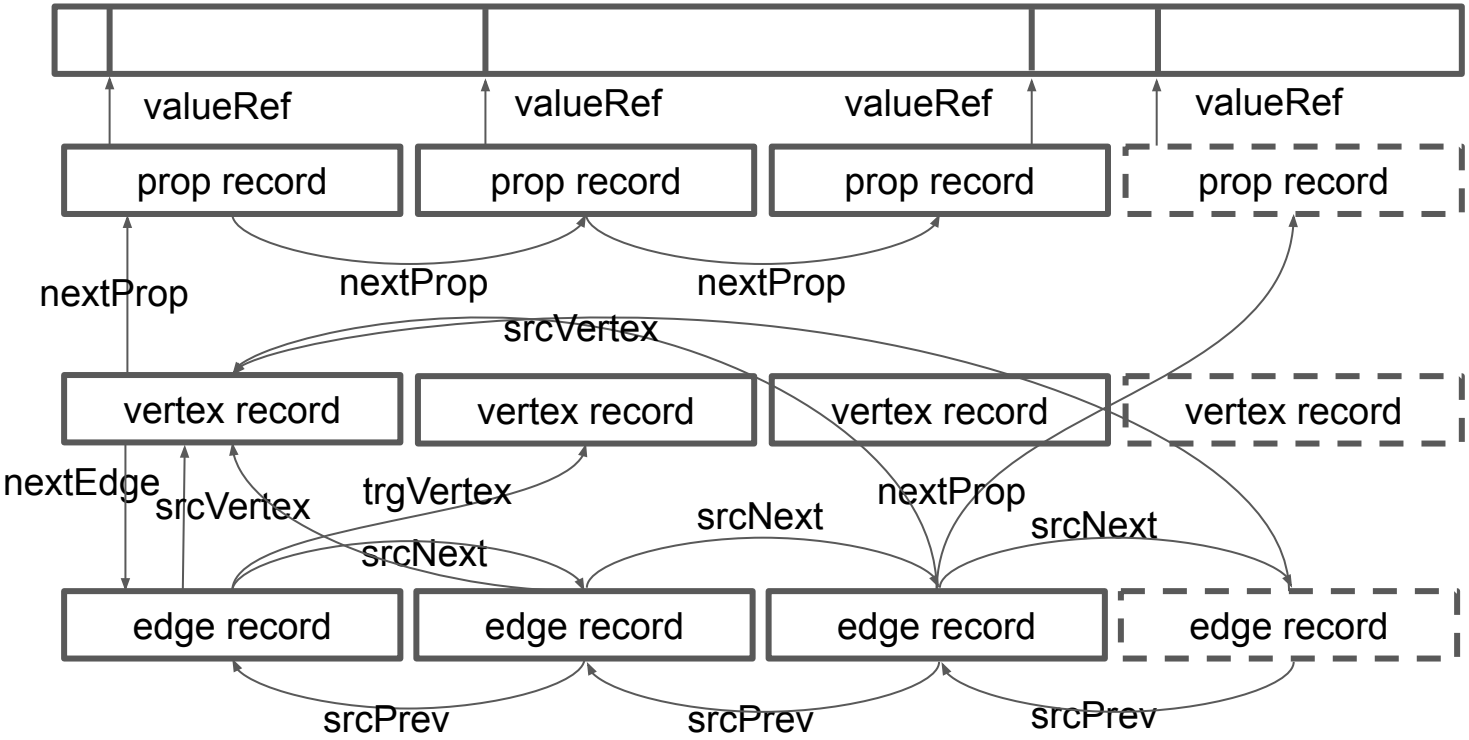
Prop Record



Edge Record



Arrays of Records



Prop values byte array

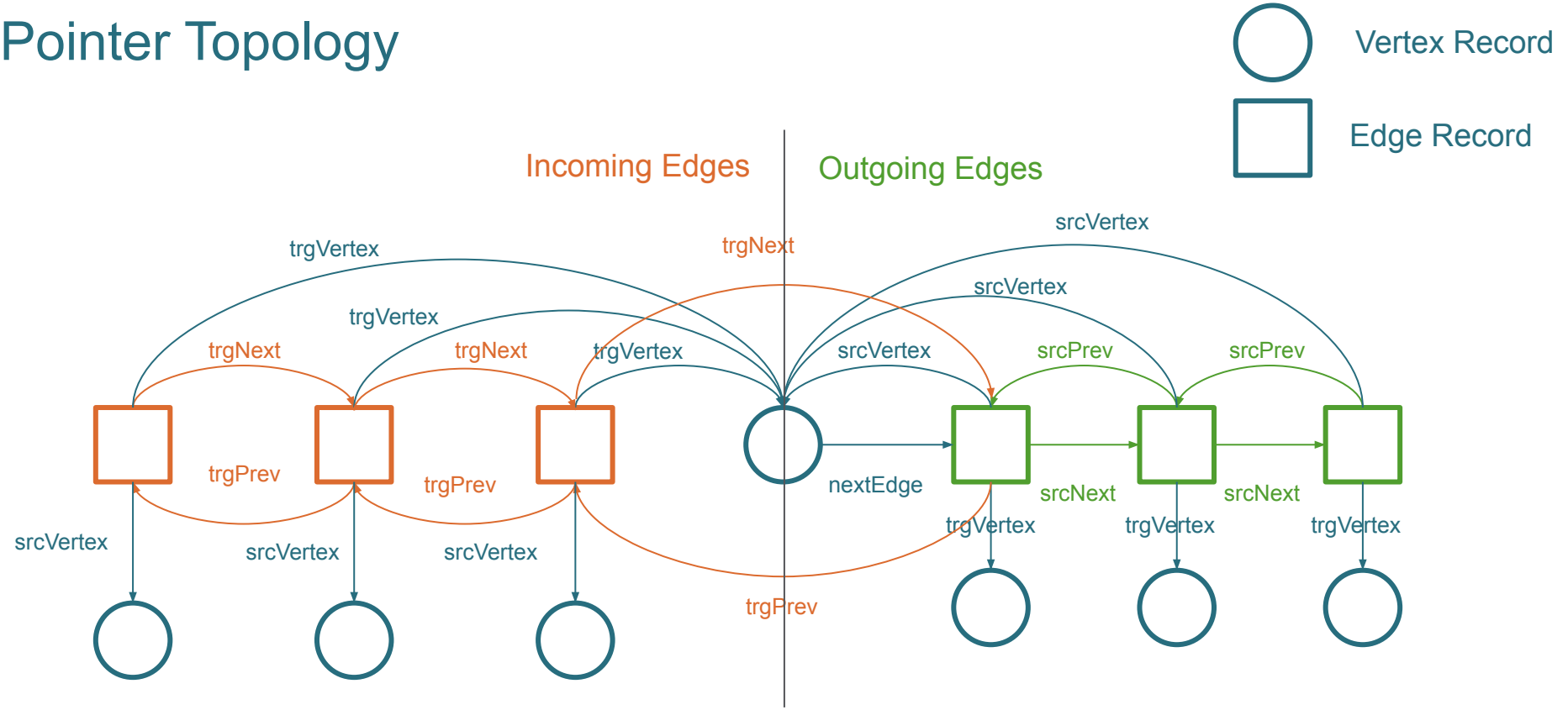
Props byte array

Vertices byte array

Edges byte array

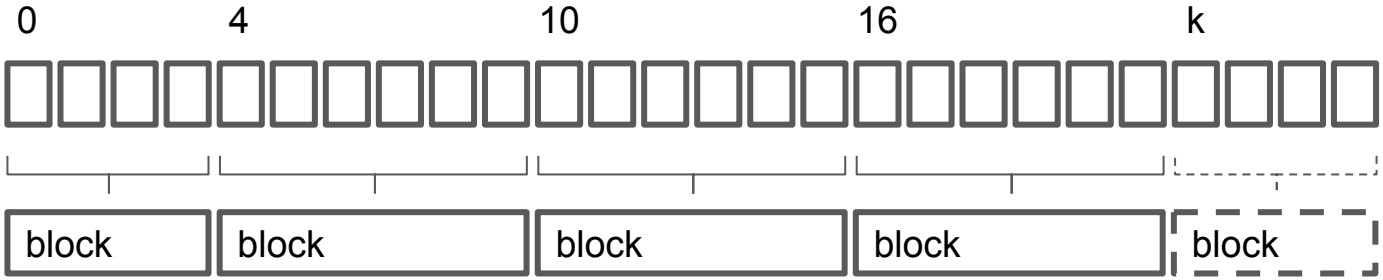
HEXAGON

Pointer Topology



HEXAGON

Chunking



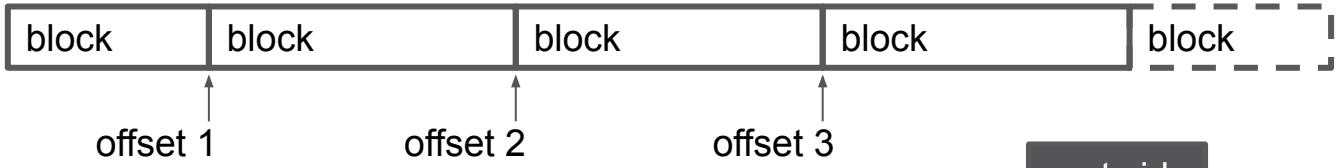
Stable Chunking: A method to partition any array of records to **support incrementality** (ie. generate stable chunks):

- Re-applying chunking alg. on the same data *should yield same chunks*
- Re-applying chunking alg. on a modified data set should *maximize reuse of older chunks and minimize the creation of new chunks*
- A modified chunk is considered new chunk (as it needs publishing)

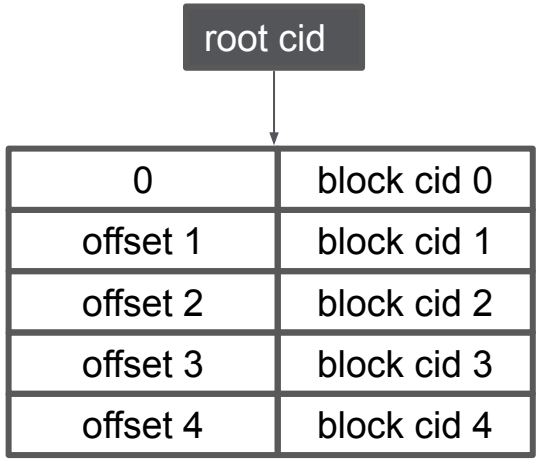
Algorithm Choices:

- Fixed size chunking, on arrays of fixed size records (eg. chunk-size = k * record-size, where k = number of records per chunk)
- Content-based chunking, works on any data*.

Index blocks by record offset



- The index describes unambiguously the array of records
- The index itself is an array of records which can be chunked recursively
- The index is immutable
- The index is content addressable



HEXAGON

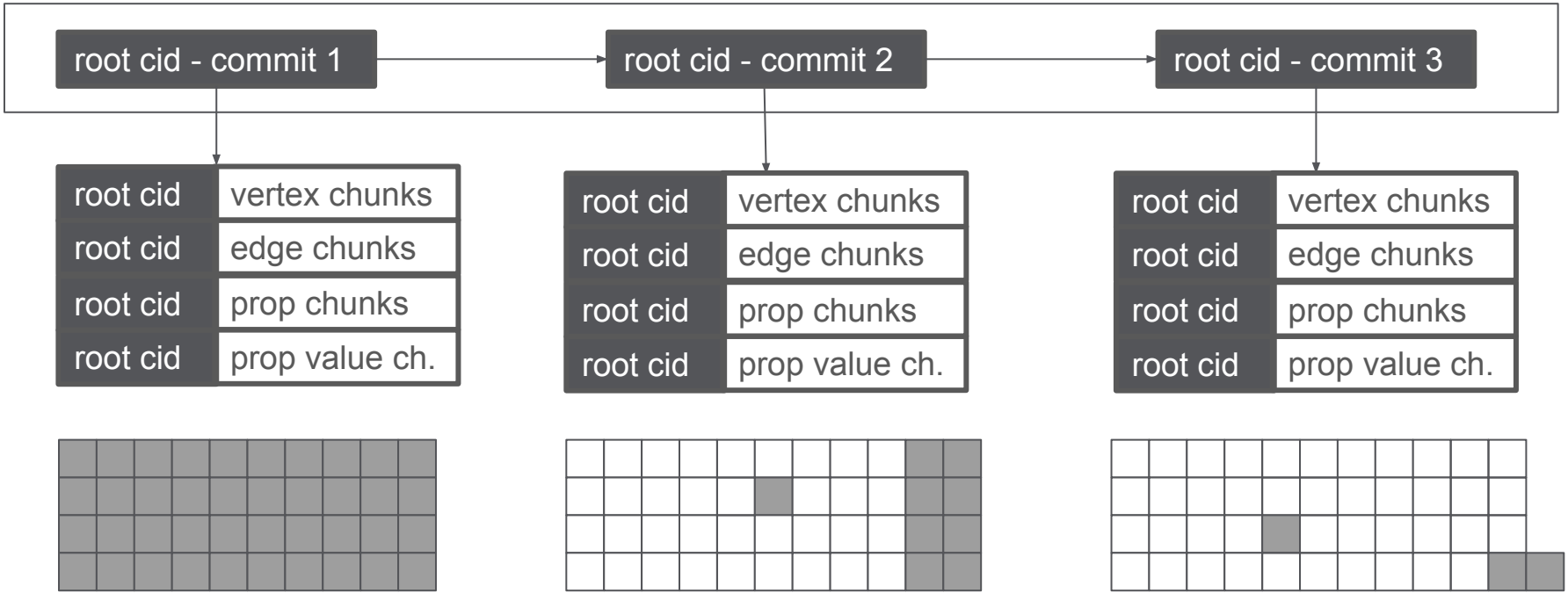
More Axioms

- *Content addressing* favors location independent storage, ie simplified replication across storage providers (ie. AZ blob storage, S3, browser local, etc.), supportive to collab. scenarios
- References represented as content identifiers *guarantee referred data integrity*: uniqueness, immutability and authenticity
- *Simplified & efficient block equality checks* w/o accessing referred data (eg. leverage the chunk index to identify changes to a given baseline)
- Supportive for chunk deduplication, supportive for versioning

Create Graph

Add new sub-tree

Modify many Property Values on single Vertex



Reused chunk

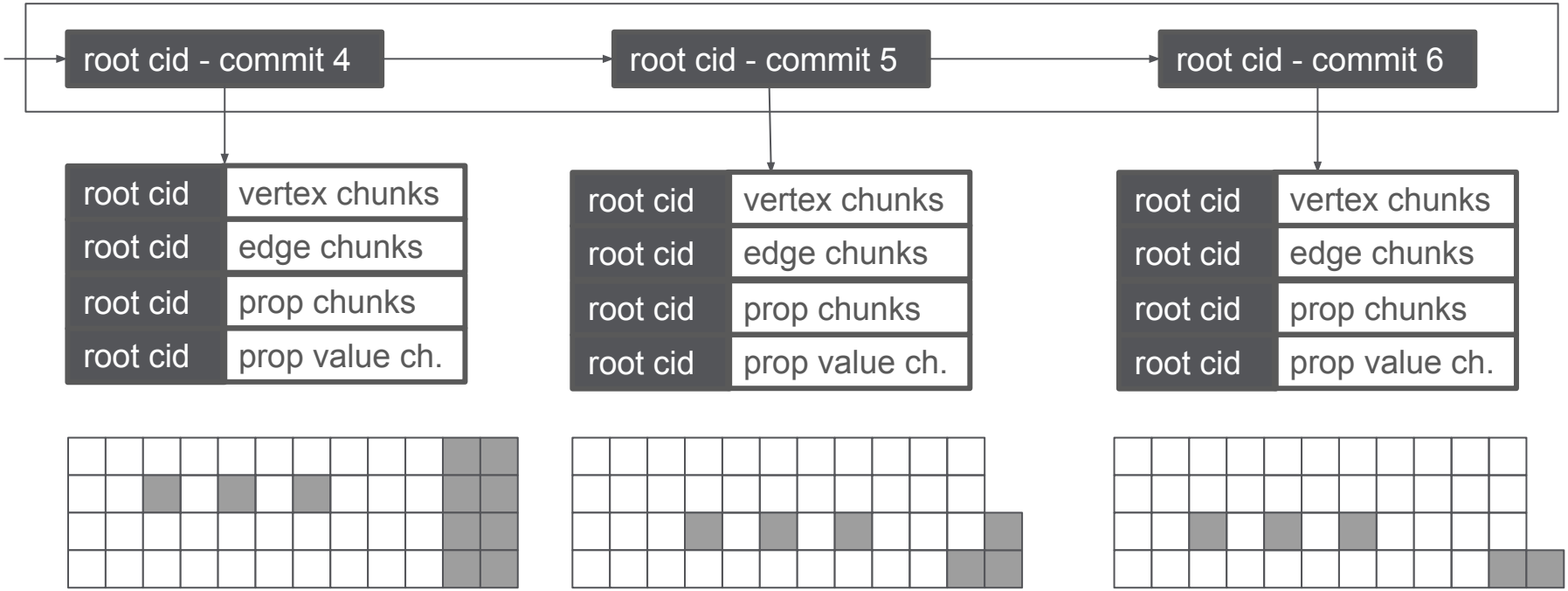
New / Modified chunk

HEXAGON

Add many new sub-trees

Add many Properties to many Vertices

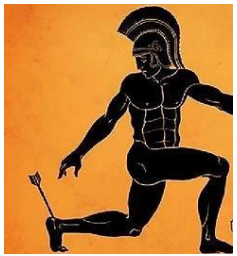
Modify many Property Values to many Vertices



Reused chunk

New / Modified chunk

Challenges



- **Incrementality vs. Fragmentation**
  - Incrementality requires to collocate data according to edits and preserve historical blocks (minimize writes), but adds fragmentation to the actual graph structure (possibly scattered read). CQRS?
- **Read**
  - Fragmentation can be a performance killer
  - Use-case based optimizer (batching blocks at retrieval time)
  - Caching blocks locally (relevant also for offline support)
- **Delete**
  - Logical only, will benefit from a strategy to reuse offsets from deleted records
- **Compaction**
  - To correct fragmentation, prune deleted records
  - Invasive (potentially replace many offsets and associated refs, introduce logical offsets?),
  - Complex to execute online