Hadoop Distributed File System

Big Data Management

Knowledge objectives

1. Recognize the need of persistent storage
2. Enumerate the design goals of GFS
3. Explain the structural components of HDFS
4. Name three file formats in HDFS and explain their differences
5. Recognize the importance of choosing the file format depending on workload
6. Explain the actions of the coordinator node in front of chunkserver failure
7. Explain a mechanism to avoid overloading the master node in HDFS
8. Explain how data is partitioned and replicated in HDFS
9. Recognize the relevance of sequential read

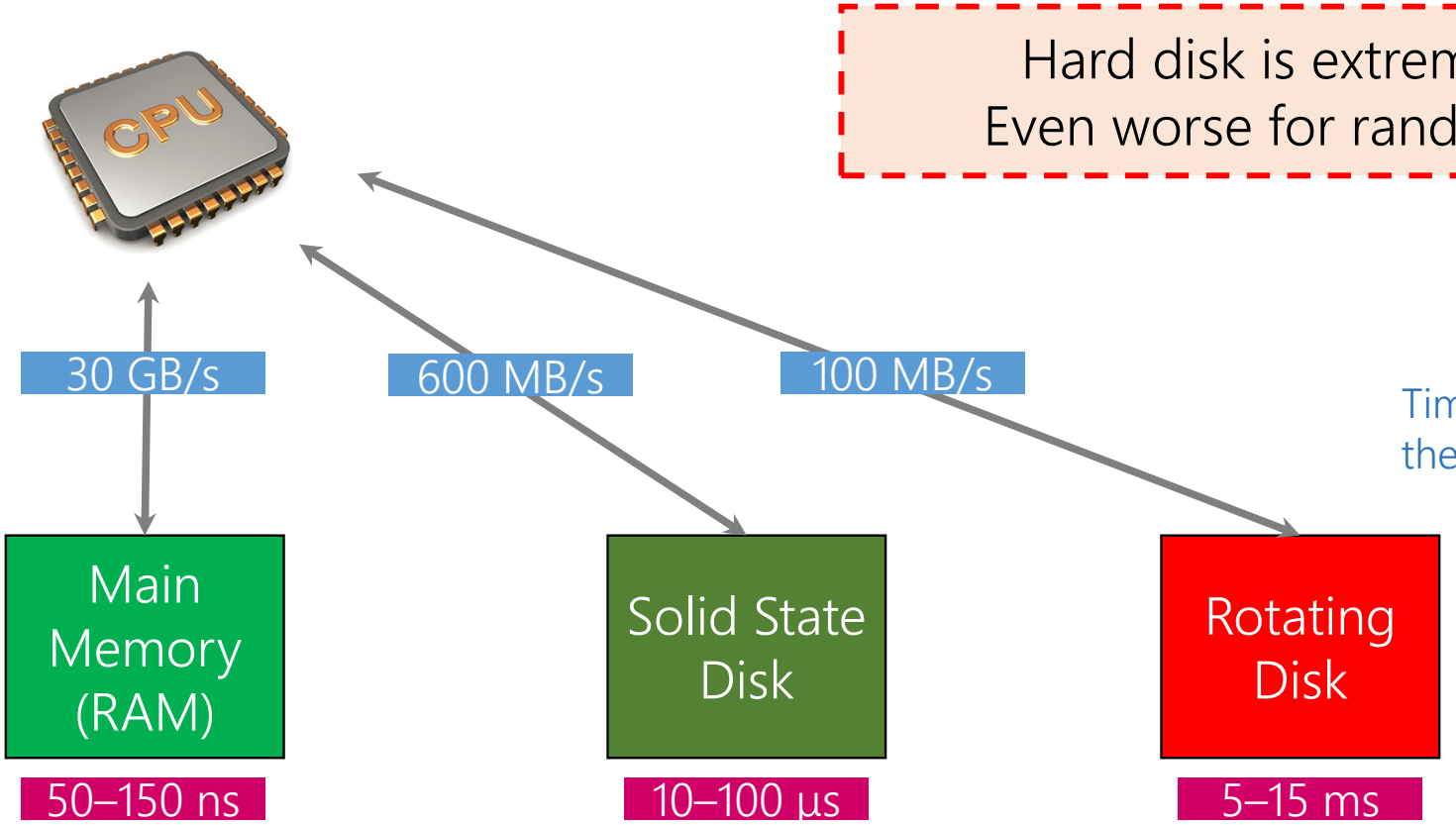
Understanding objectives

1. Choose the format for an HDFS file based on heuristics
2. Estimate the data retrieved by scan, projection and selection operations in SequenceFile, Avro and Parquet

(Distributed) File Systems

Google File System

Time to bring data (approximations)



Hard disk is extremely slow!
Even worse for random access.

Data transfer

Time to move the data from the disk to the processor

Latency

Time to start an I/O (i.e., access a block)

Reasons to keep using HDDs

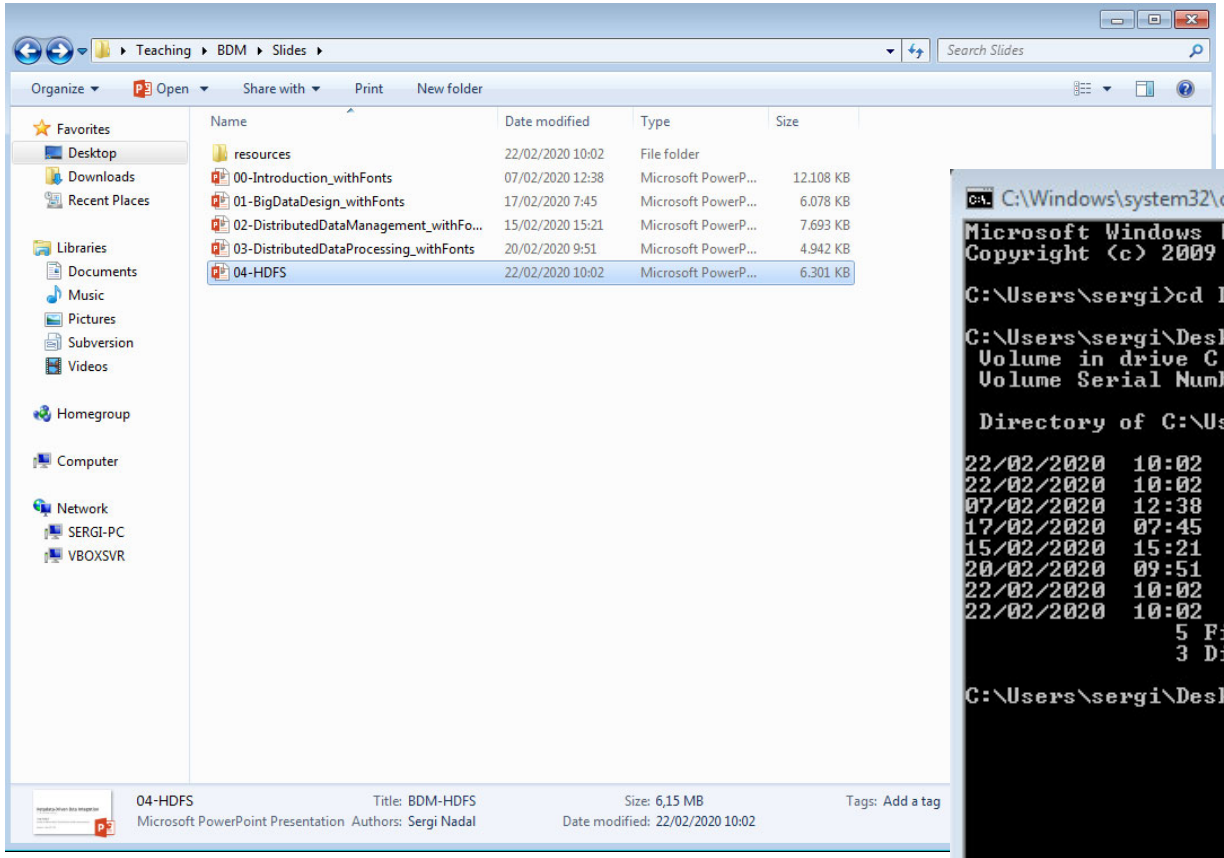
Main Memory

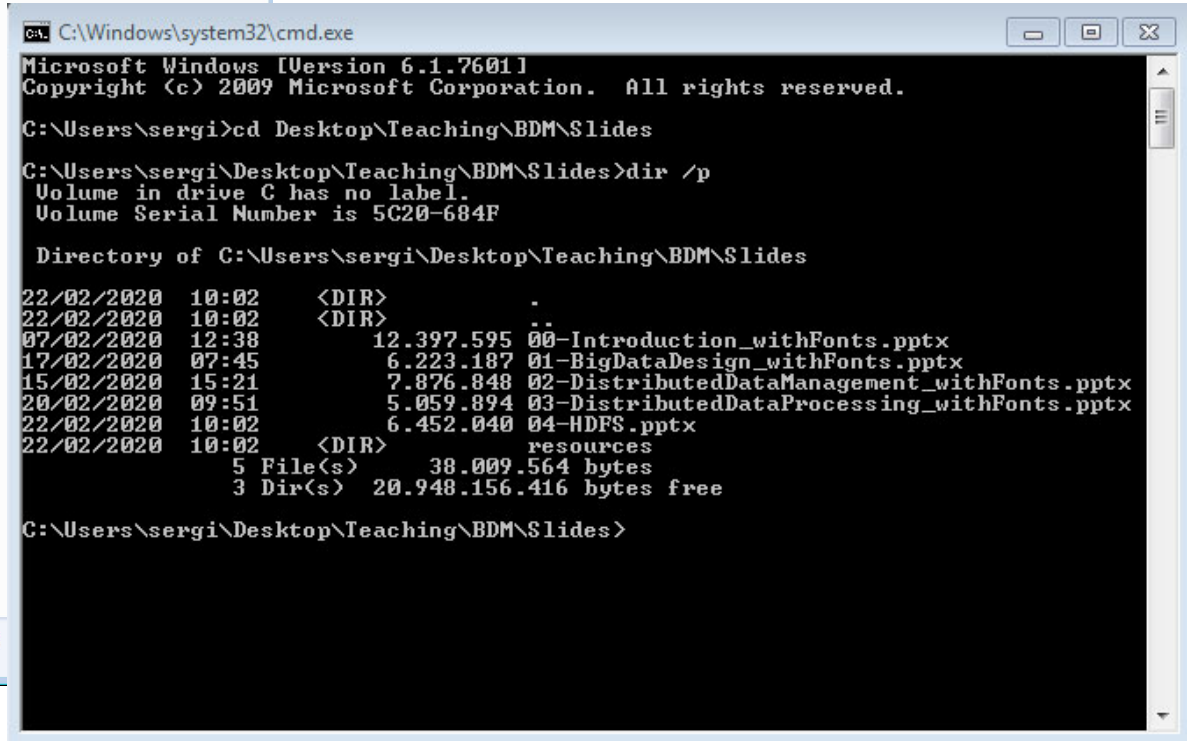
Faster ✔

✔ Cheaper
✔ Persistent

Hard Drive Disk

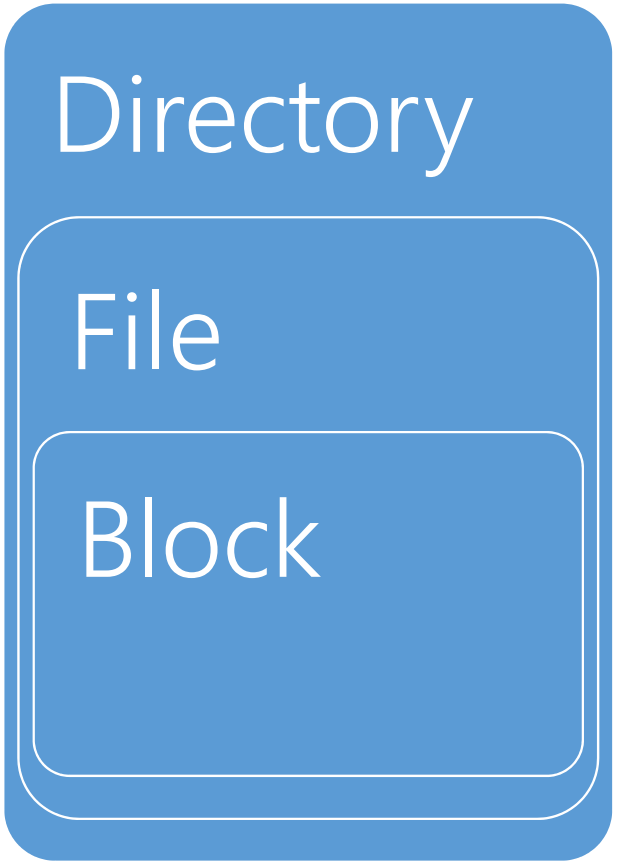
What is a file system?





Functionalities provided by a FS

- Creates a hierarchical structure of data
  - Splits and stores data into files and blocks
- Provides interfaces to read/write/delete
- Maintains directories/files metadata
  - Size, date of creation, permissions, …



Distributed File Systems

- Same requirements, different setting
  1. Files are huge for traditional standards
  2. Most files are updated by appending data rather than overwriting
     - Write Once and Read Many times (WORM)
  3. Component failures are the norm rather than the exception
- Google File System (GFS)
  - The first large-scale distributed file system
  - Capacity of a GFS cluster

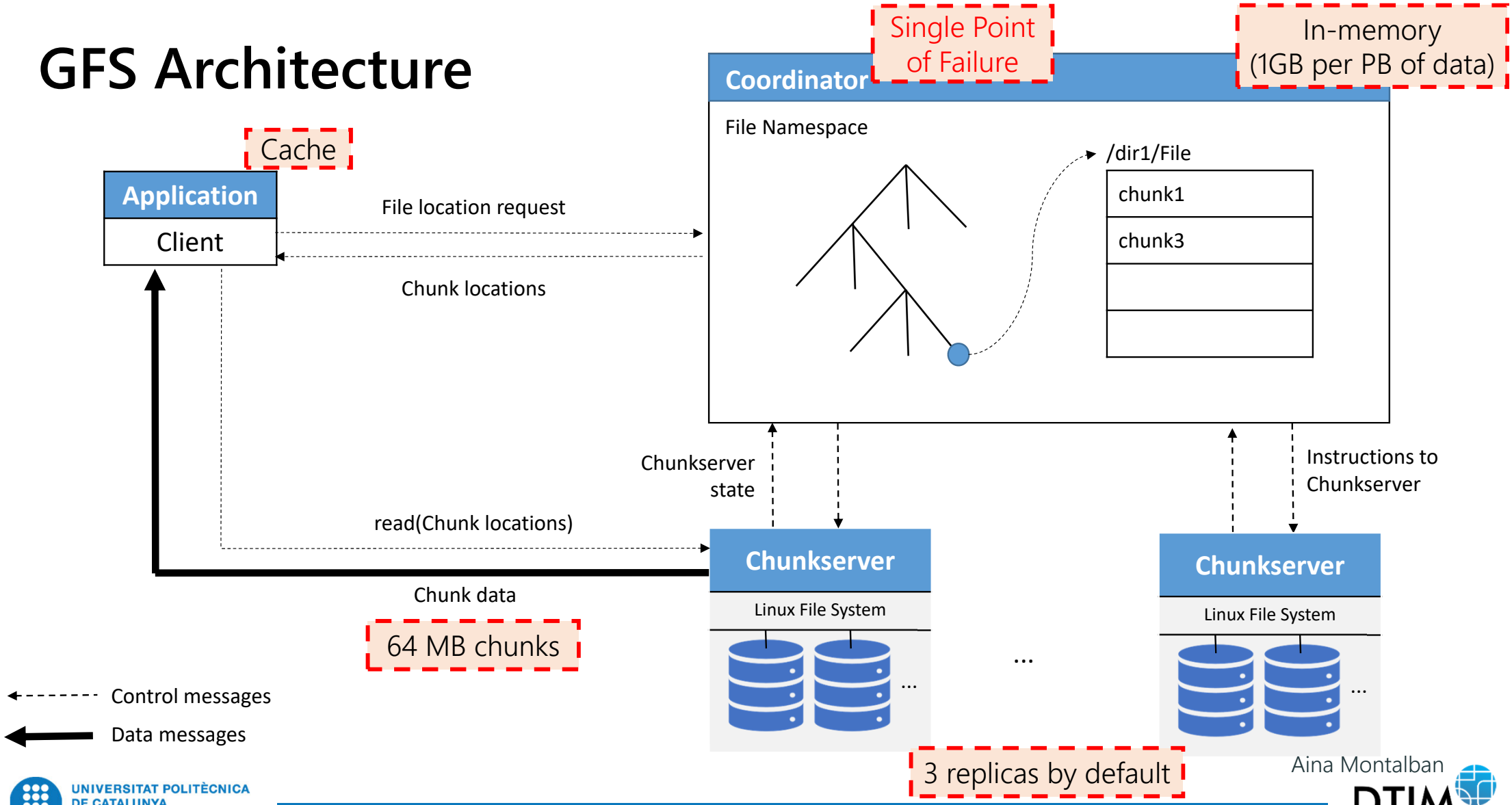
Capacity	Nodes	Clients	Files
10 PB	10.000	100.000	100.000.000

Design goals of GFS

- Efficient management of files
  - Optimized for very large files (GBs to TBs)
- Efficiently append data to the end of files
  - Allow concurrency
- Tolerance to failures
  - Clusters are composed of many inexpensive machines that fail often
    - Failure probability (2-3/1.000 per day)
- Optimize sequential scans
  - Overcome high latency of HDDs (5-15ms) compared to main memory (50-150ns)

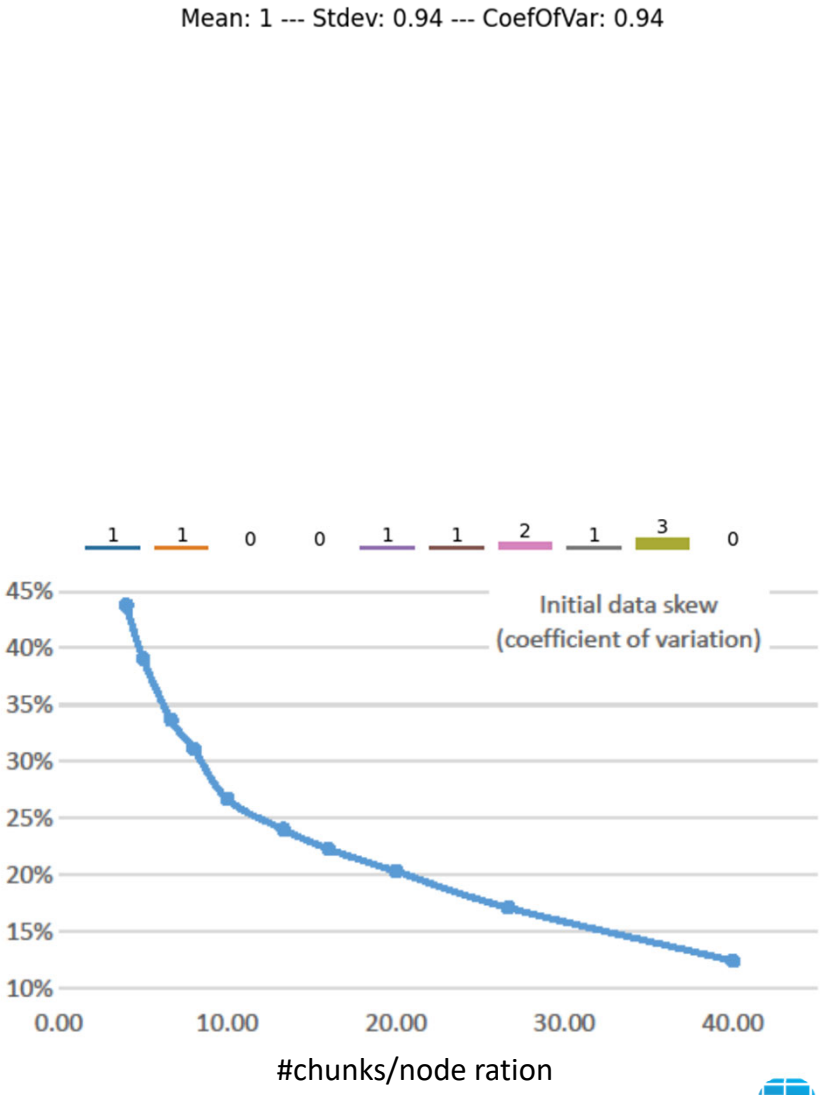
GFS Architecture

GFS Architecture



Other features

- Rebalance
  - Avoids skewness in the distribution of chunks
- Deletion
  - Moves a file to the trash (hidden)
    - Kept for 6h
  - *expunge* to force the trash to be emptied
- Management of stale replicas
  - Coordinator maintains versioning information about all chunks



(Distributed) Data Design

Challenge I

Storage layouts

"Jack of all trades, master of none"

- Different workloads require different layout
  - Horizontal
    - For scan-based workloads
  - Vertical
    - For projection-based workloads (reads a subset of columns)
  - Hybrid
    - For projection- and predicate-based workloads (reads a subset of columns or rows)

Horizontal layout – Sequence File

- Records of binary key-value pairs

Table 1

A	B	C	D
101	201	301	401
102	202	302	402
103	203	303	403

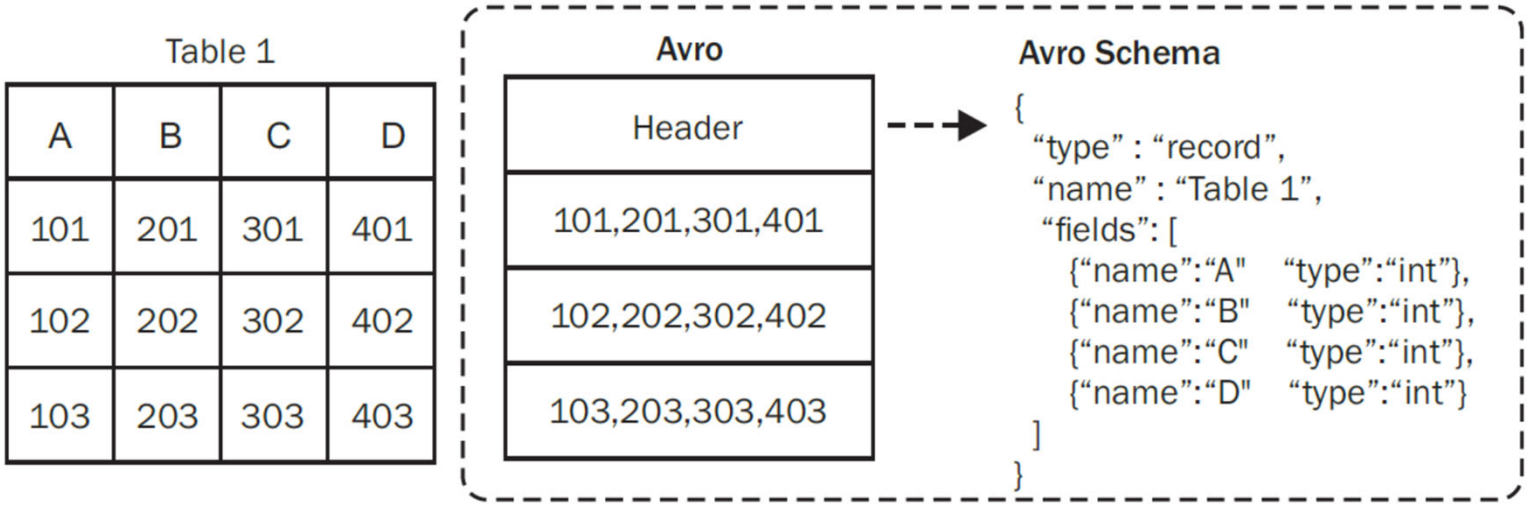
Sequence File

Header
Key : 101 Value : 201,301,401
Key : 102 Value : 202,302,402
Key : 103 Value : 203,303,403

- Compression
  - Uncompressed
  - Record-compressed
  - Block-compressed
    - "block" is the compression unit – a block of records (not a chunk)
      - 1 MB default

Horizontal layout - Avro

- Binary encoding of (compressed) rows
- The header contains a schema encoded in JSON

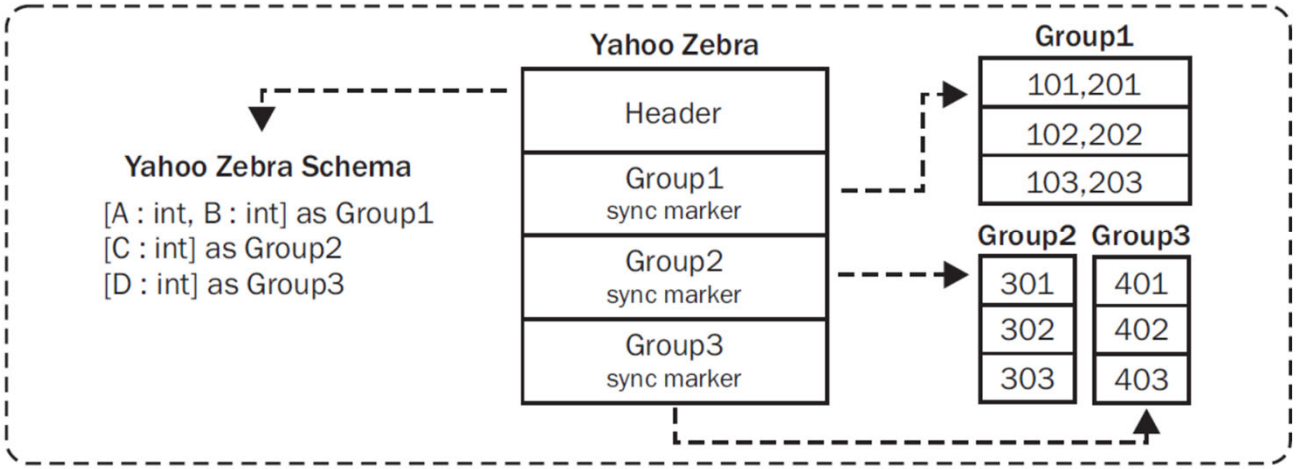


Vertical layout - Zebra

- The header contains the definition of groups
  - Each group contains a set of columns
  - Widely benefits from compression
- Not really used in practice

Table 1

A	B	C	D
101	201	301	401
102	202	302	402
103	203	303	403

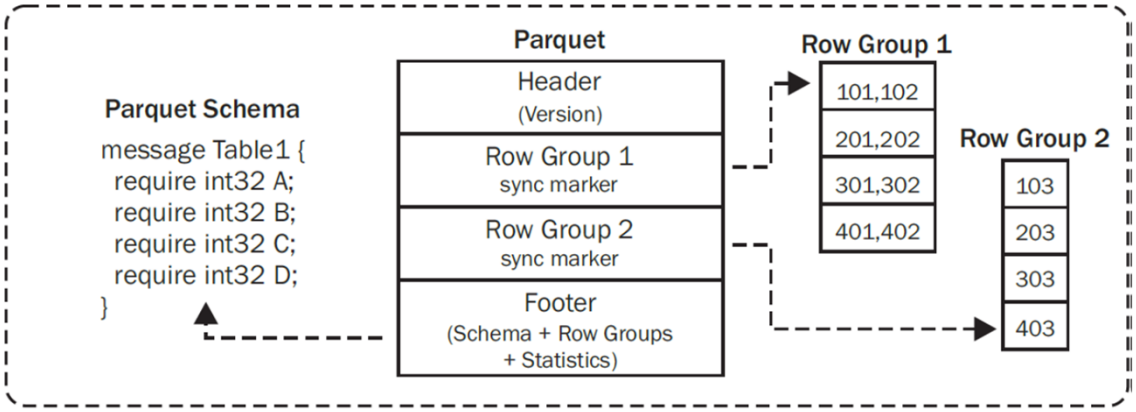


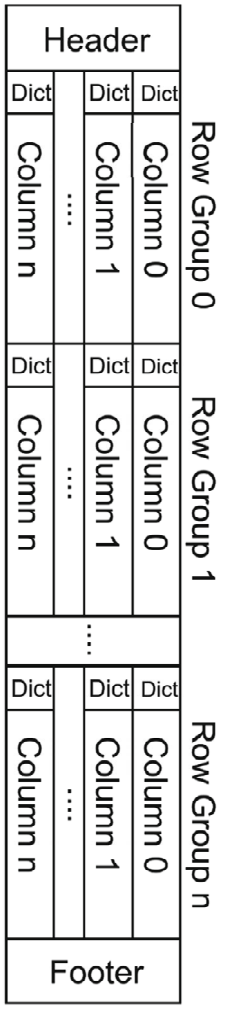
Hybrid layout - Parquet

- Row groups (RG) - horizontal partitions
  - Data vertically partitioned within RGs
- Statistics per row group (aid filtering)
  - E.g., min-max

Table 1

A	B	C	D
101	201	301	401
102	202	302	402
103	203	303	403





Parquet encoding definitions

- Plain
- Valid for any data type
  - Dictionary encoding
  - Run-length encoding with dictionary
- Specific for some kind of data types
  - Delta encoding
  - Delta byte array / Delta-length byte array
  - Byte Stream Split

Comparison of data formats

Features	Horizontal		Vertical	Hybrid	
	Sequence Files	Avro	Yahoo Zebra	ORC	Parquet
Schema	No	Yes	Yes	Yes	Yes
Column Pruning	No	No	Yes	Yes	Yes
Predicate Pushdown	No	No	No	Yes	Yes
Indexing Information	No	No	No	Yes	Yes
Statistics Information	No	No	No	Yes	Yes
Nested Records	No	No	Yes	Yes	Yes



Rule-based choice (heuristic)

Given a flow represented as DAG(V, E)

- SequenceFile
  - size(getCol(v)) = 2
- Parquet
  - ∃e ∈ O(v), getType(e) = {AggregationOps}
  - ∃e ∈ O(v), getCol(getOP (e)) ⊂ getCol(v)
- Avro
  - ∀e ∈ O(v), getCol(getOP (e)) = getCol(v)
  - ∃e ∈ O(v), getType(e) ∈ {Join, CartesianProduct, GroupAll, Distinct}

Cost-based choice

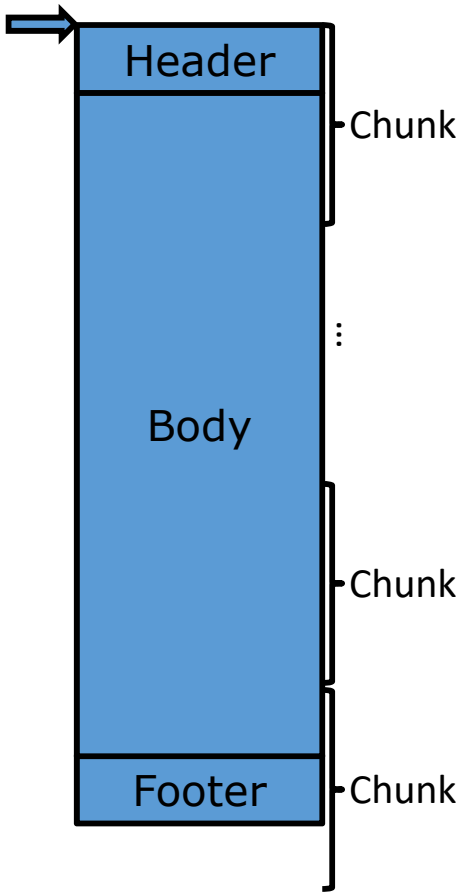
- Helps in choosing the right storage layout based on the workloads
- Costs to Consider
  - Write cost
  - Read Cost
    - Scan Operation
    - Projection Operation
    - Selection Operation
- Costs ignored
  - Block compression
  - Dictionary encoding (in Parquet)

General formulas for size estimation

$$Size(Layout) = Size(Header_{Layout}) + Size(Body_{Layout}) + Size(FooterLayout)$$

$$UsedChunks(Layout) = \frac{Size(Layout)}{Size(chunk)}$$

$$Seeks(Layout) = \lceil UsedChunks(Layout) \rceil$$

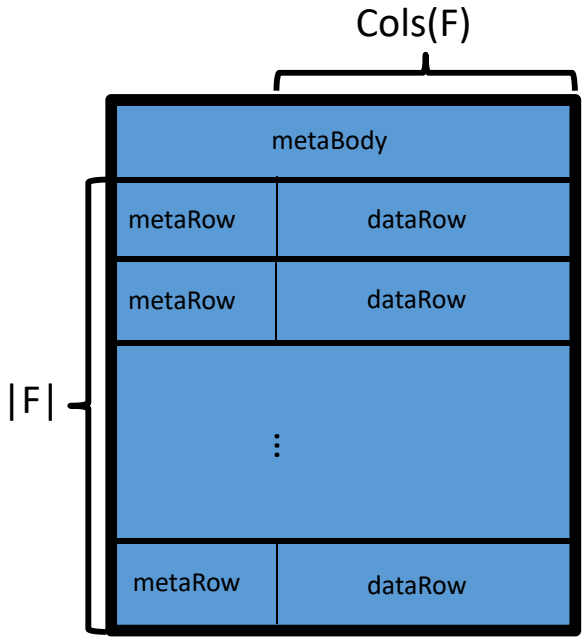


"Layout" can be either "Horizontal", "Vertical" or "Hybrid"

Horizontal layout size estimation

$$Size(dataRow) = Cols(F) * Size(cell)$$

$$Size(Body_{Horizontal}) = Size(metaBody) + |F| * (Size(metaRow) + Size(dataRow))$$

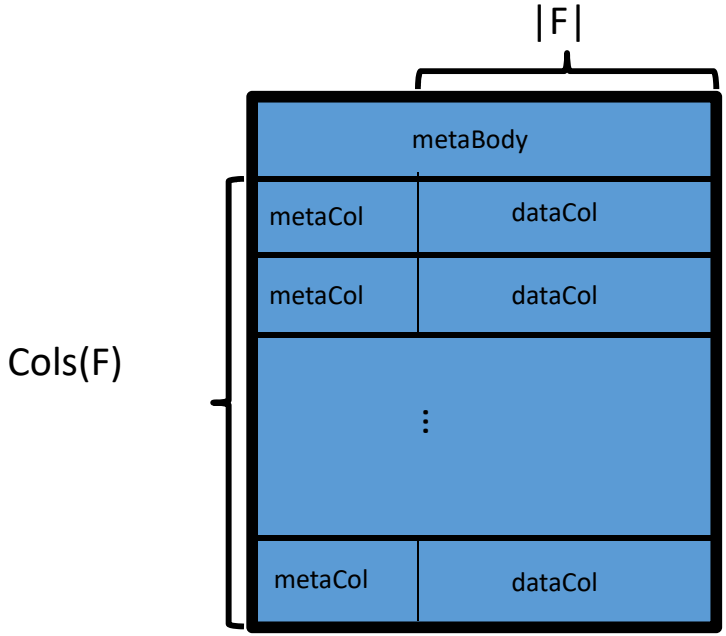


"F" is the content of the file

Vertical layout size estimation

$$Size(dataCol) = |F| * Size(cell)$$

$$Size(Body_{Vertical}) = Size(metaBody) + Cols(F) * (Size(metaCol) + Size(dataCol))$$



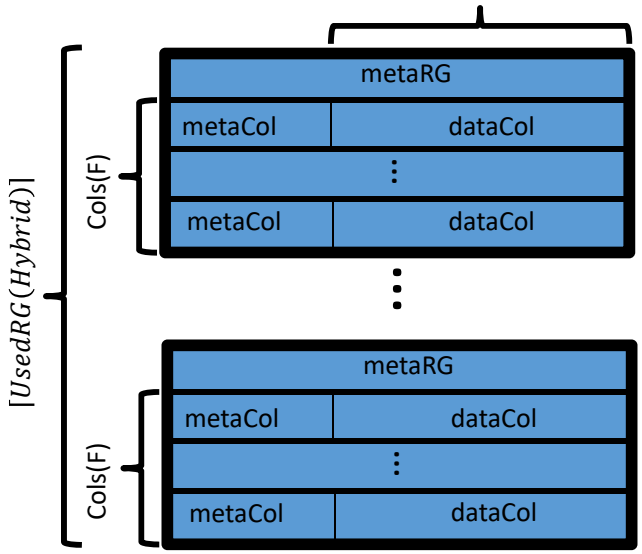
"F" is the content of the file

Hybrid layout size estimation

$$UsedRG(Hybrid) = \frac{Cols(F) * |F| * Size(cell)}{(Size(RG) - Size(metaRG) - Cols(F) * Size(metaCol))}$$

$$Size(Body_{Hybrid}) = \lceil UsedRG(Hybrid) \rceil * (Size(metaRG) + Cols(F) * Size(metaCol)) + Cols(F) * |F| * Size(cell)$$

$$UsedRows(RG) = \frac{|F|}{UsedRG(Hybrid)}$$



"F" is the content of the file

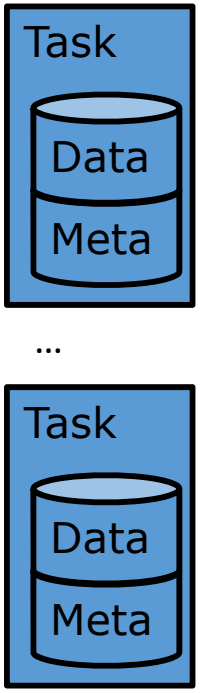
General formulas for cost estimation

$$Cost(Write_{Layout}) = \text{UsedChunks(Layout)*W}_{\text{WriteTransfer}} + \text{Seeks(Layout)*(1-W}_{\text{WriteTransfer}})$$

$$Size(Scan_{Layout}) = Size(Layout) + (\lceil UsedChunks(Layout) \rceil * Size(Meta_{Layout}))$$

$$UsedChunks(Scan_{Layout}) = \frac{Size(Scan_{Layout})}{Size(chunk)}$$

$$Cost(Scan_{Layout}) = \text{UsedChunks(Scan}_{\text{Layout}}\text{)*W}_{\text{ReadTransfer}} + \text{Seeks(Layout)*(1-W}_{\text{ReadTransfer}})$$



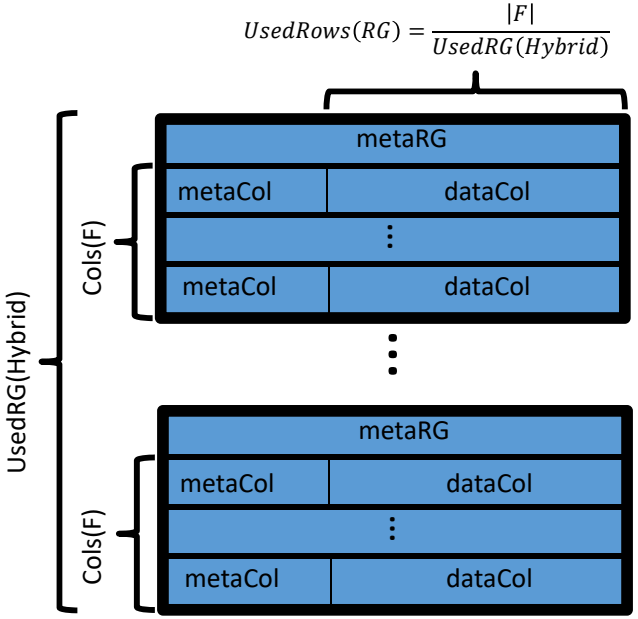
"Layout" can be either "Horizontal", "Vertical" or "Hybrid"

Projection in hybrid layouts

$$Size(projCols) = Proj(F) * UsedRows(RG) * Size(cell)$$

$$Size(Project_{Hybrid}) = Size(Header_{Hybrid}) + Size(Footer_{Hybrid}) + \lceil UsedRG(Hybrid) \rceil * (Size(metaRG) + proj(F) * Size(metaCol)) + UsedRG(Hybrid) * Size(projCols)$$

$$Cost(Project_{Hybrid}) = \text{UsedChunks}(\text{Project}_{\text{Hybrid}})*\text{W}_{\text{ReadTransfer}} + \text{Seeks}(\text{Hybrid})*(1-\text{W}_{\text{ReadTransfer}})$$



"Proj(F)" is the number of projected columns

Probability of retrieving a RowGroup

- Probability of a row fulfilling P (a.k.a. selectivity factor)
  SF

- Probability of a row NOT fulfilling P
  1-SF

- Probability of none of the rows in a RowGroup fulfilling P
  $(1-SF)\cdot(1-SF)\cdot \ldots \cdot(1-SF) = (1-SF)^{UsedRows(RG)}$

- Probability of some row in a RowGroup fulfilling P
  $1-(1-SF)^{UsedRows(RG)}$

Selection in hybrid layouts

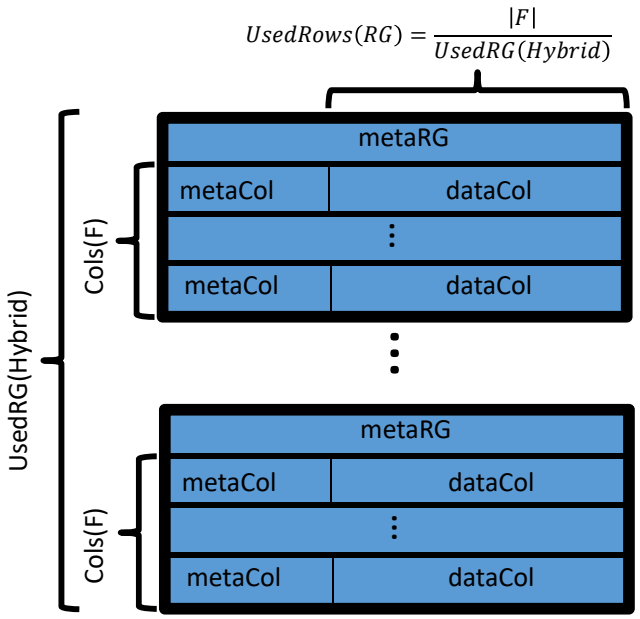
$$P(RGSelected) = 1 - (1 - SF)^{UsedRows(RG)}$$

$$Size(RowsSelected) = \left\lceil \frac{SF * |F|}{UsedRows(RG)} \right\rceil \left(Size(metaRG) + Cols(F) * Size(metaCol)\right) + SF * |F| * Cols(F) * Size(cell)$$

$$UsedRG\left(Select_{Hybrid}\right) = \begin{cases} if\ unsorted:\ P(RGSelected) * UsedRG(Hybrid) \\ if\ sorted: \left\lceil \frac{Size(RowsSelected)}{Size(RG)} \right\rceil \end{cases}$$

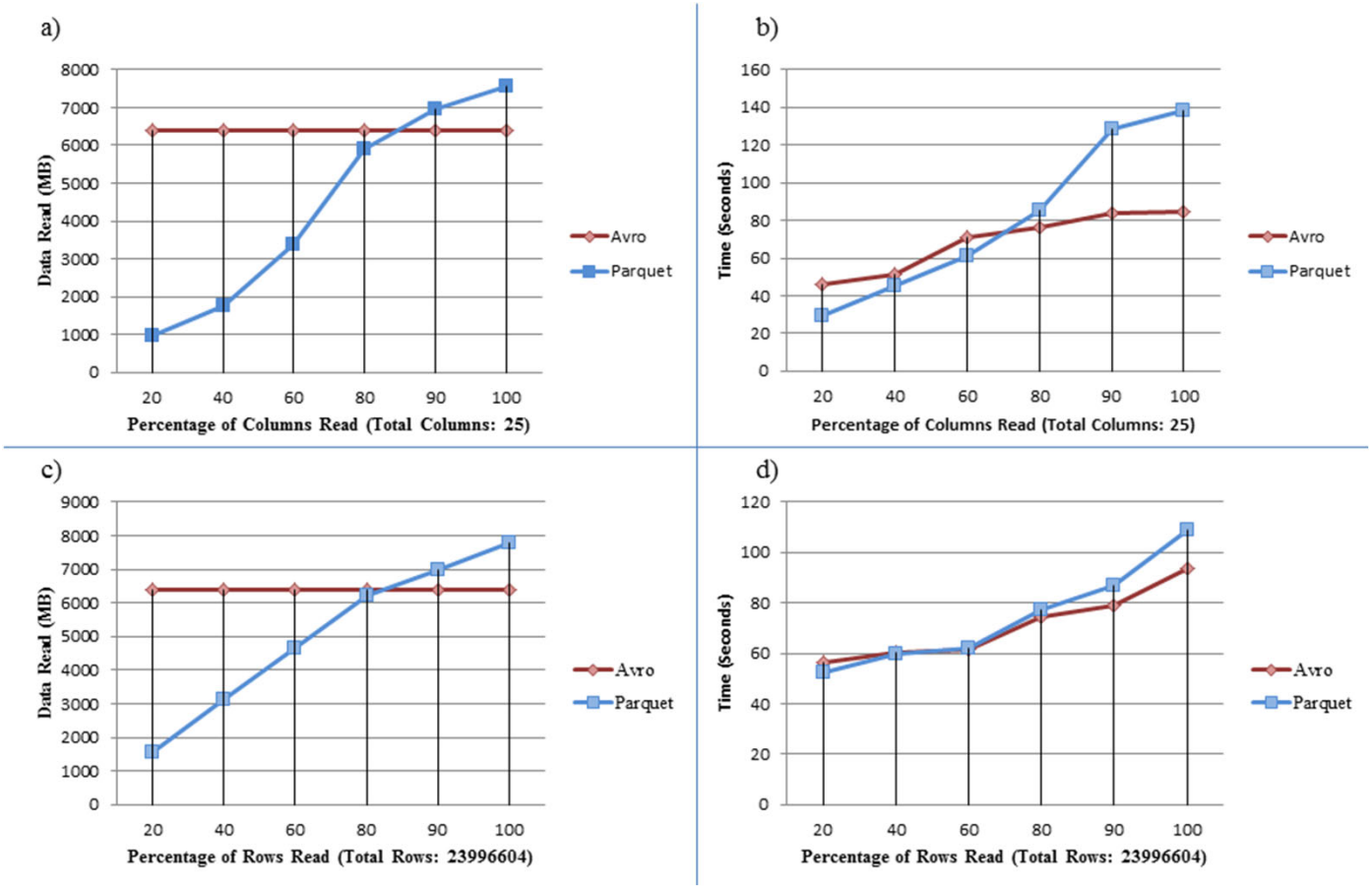
$$Size(Select_{Hybrid}) = Size\left(Header_{Hybrid}\right) + Size\left(Footer_{Hybrid}\right) + UsedRG(Select_{Hybrid}) * Size(RG)$$

$$Cost(Select_{Hybrid}) = \text{UsedChunks}(\text{Select}_{\text{Hybrid}}) * \text{W}_{\text{ReadTransfer}} + \text{Seeks}(\text{SelectHybrid}) * (1 - \text{WReadTransfer})$$



"F" is the content of the file

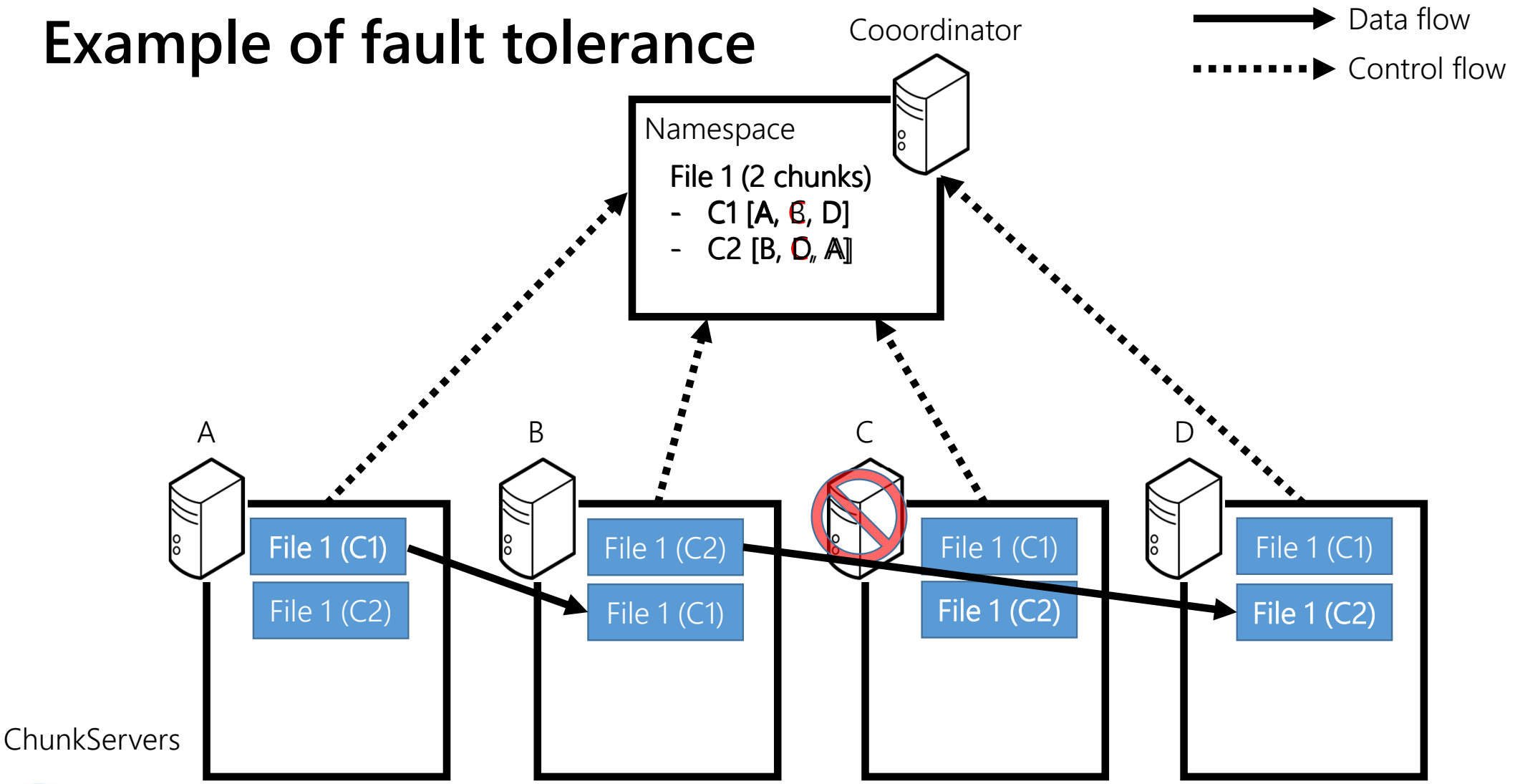
Comparison of selection and projection



Fault tolerance

- Managed from the coordinator
  - It expects to receive every 3 seconds a *heartbeat* message from chunkservers
- Chunkserver not sending a heartbeat for 60 seconds, a fault is declared
- Corrective actions
  - Update the namespace
  - Copy one of the replicas to a new chunkserver
    - Potentially electing a new primary replica

Example of fault tolerance



(Distributed) Catalog Management

Challenge II

Client caching

Cache miss

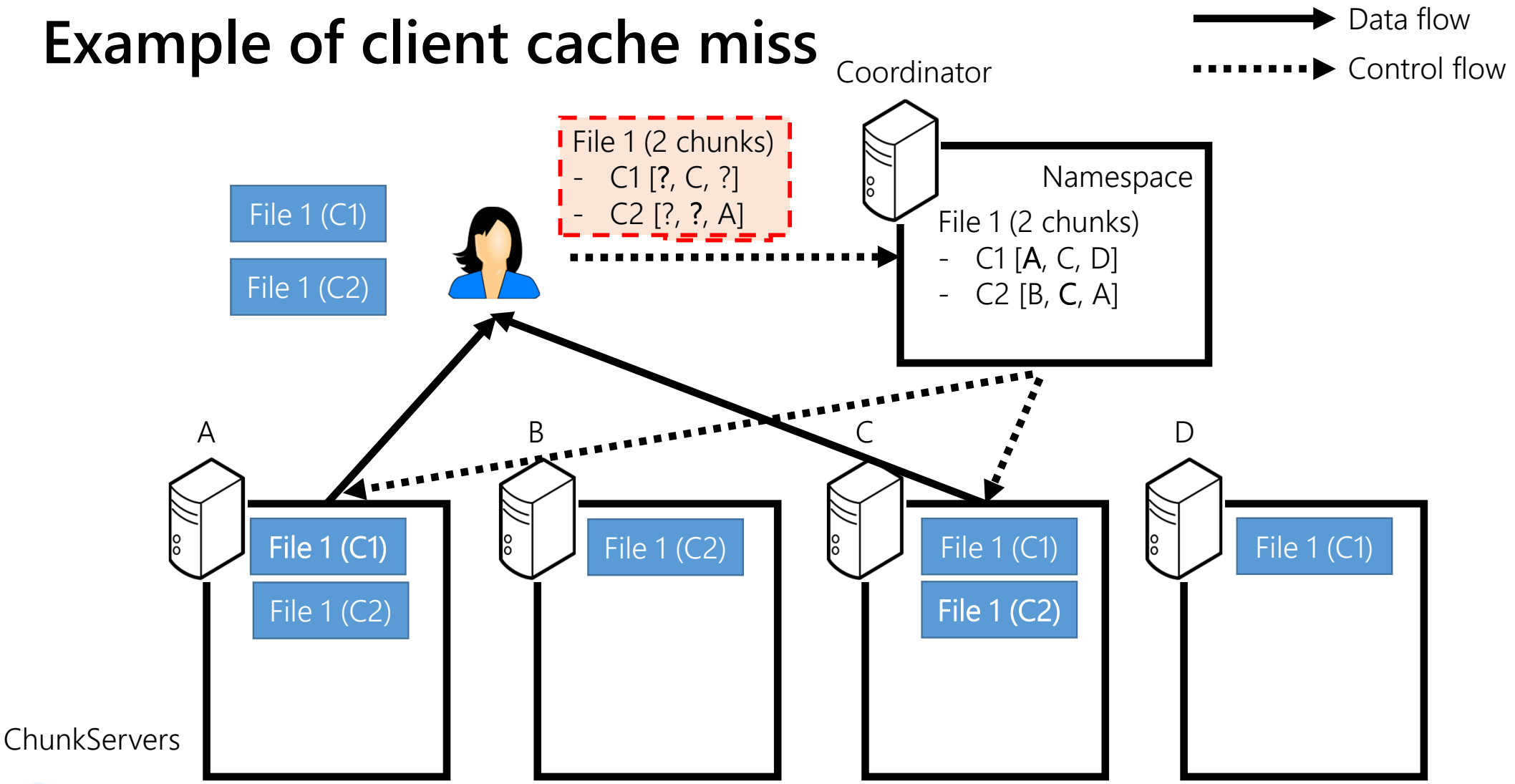
1. The client sends a READ command to the coordinator
2. The coordinator requests chunkservers to send the chunks to the client
   - Ranked according to the closeness in the network
3. The list of locations is cached in the client
   - Not a complete view of all chunks

Cache hit

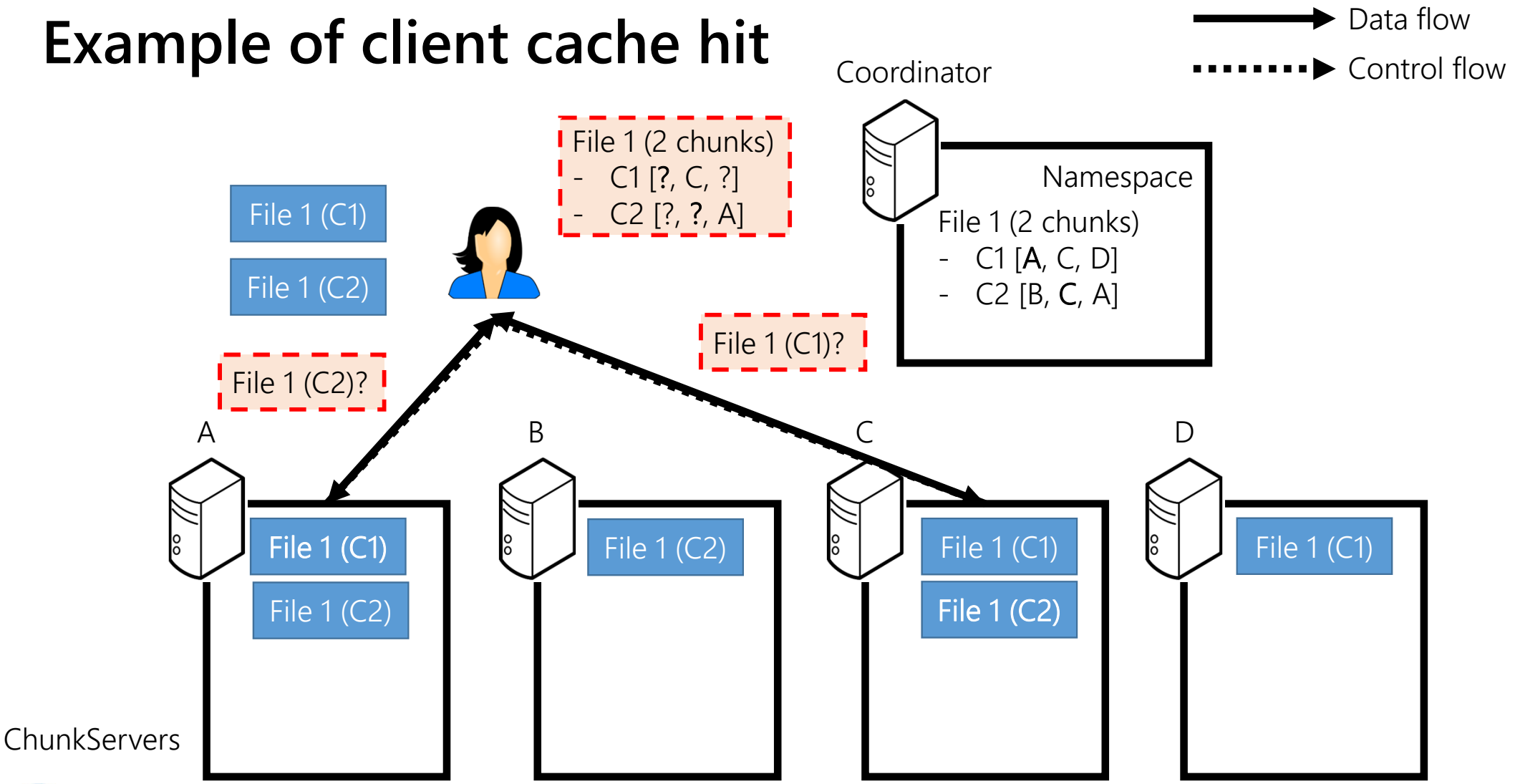
1. The client reads the cache and requests the chunkservers to send the chunks

Avoid coordinator bottleneck
+
One communication step is saved

Example of client cache miss



Example of client cache hit

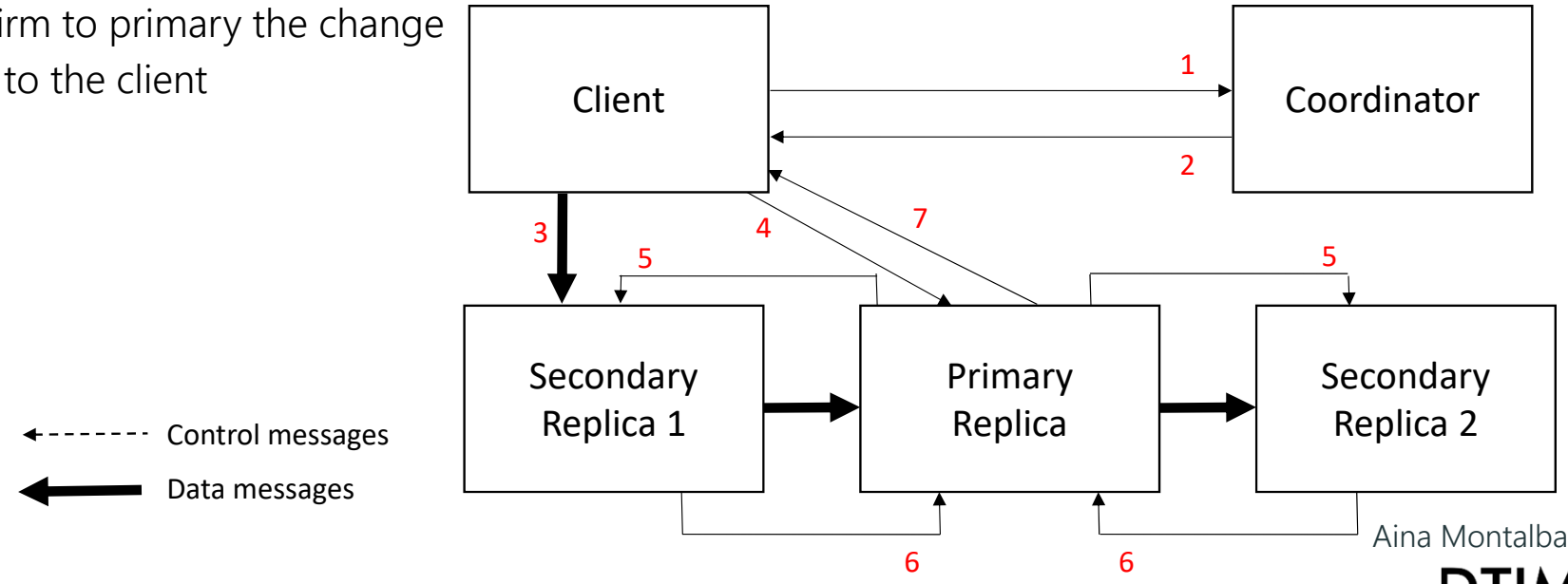


(Distributed) Transaction Management

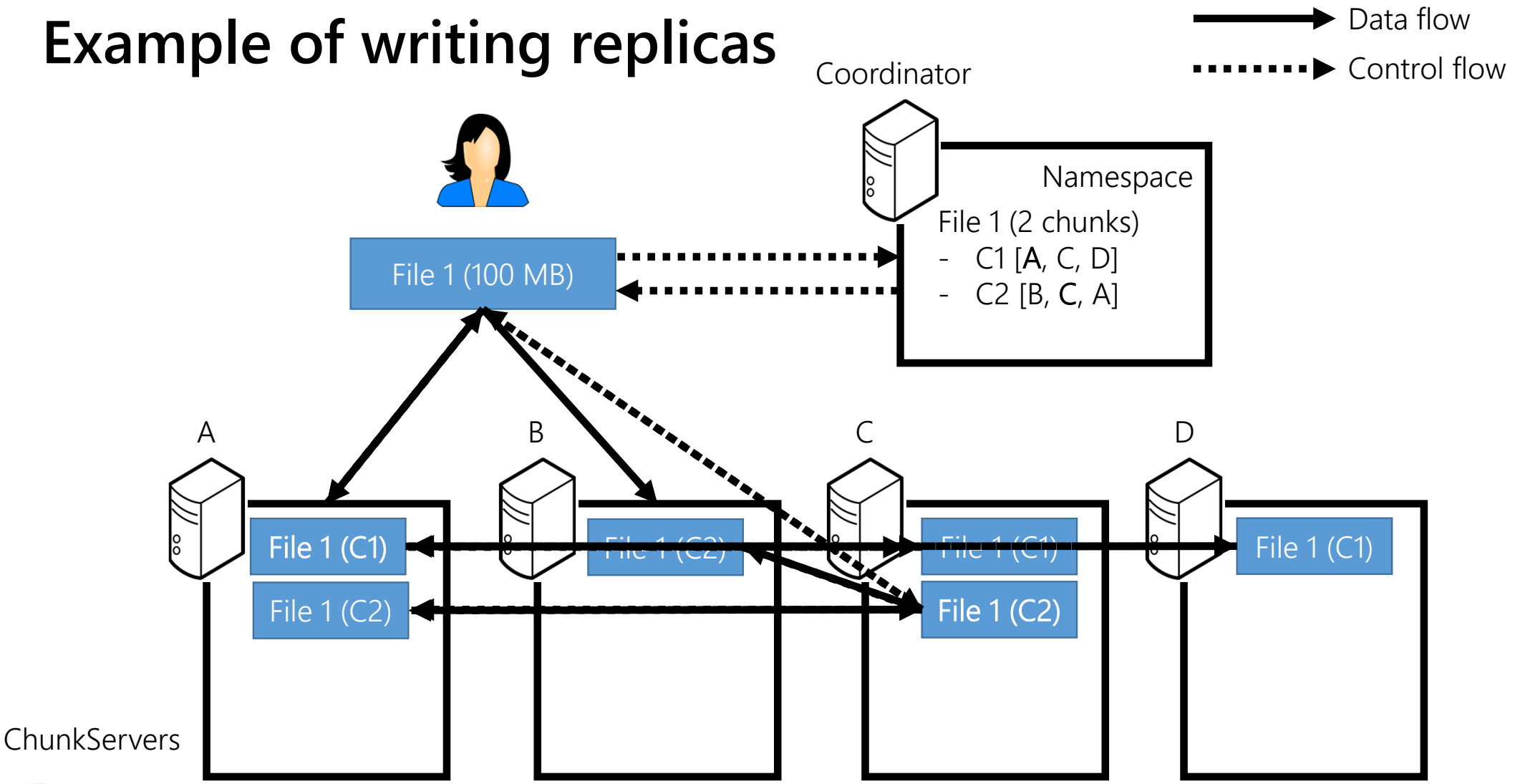
Challenge III

Writing replicas

1. The client requests the list of the replicas of a file
2. Coordinator returns metadata
3. Client sends a chunk to the closest chunkserver in the network
   - This chunk is pipelined to the other chunkservers in the order defined by the master (leases)
4. Client sends WRITE command to primary replica
5. Primary replica sends WRITE command to secondary replicas
6. Secondaries confirm to primary the change
7. Primary confirms to the client



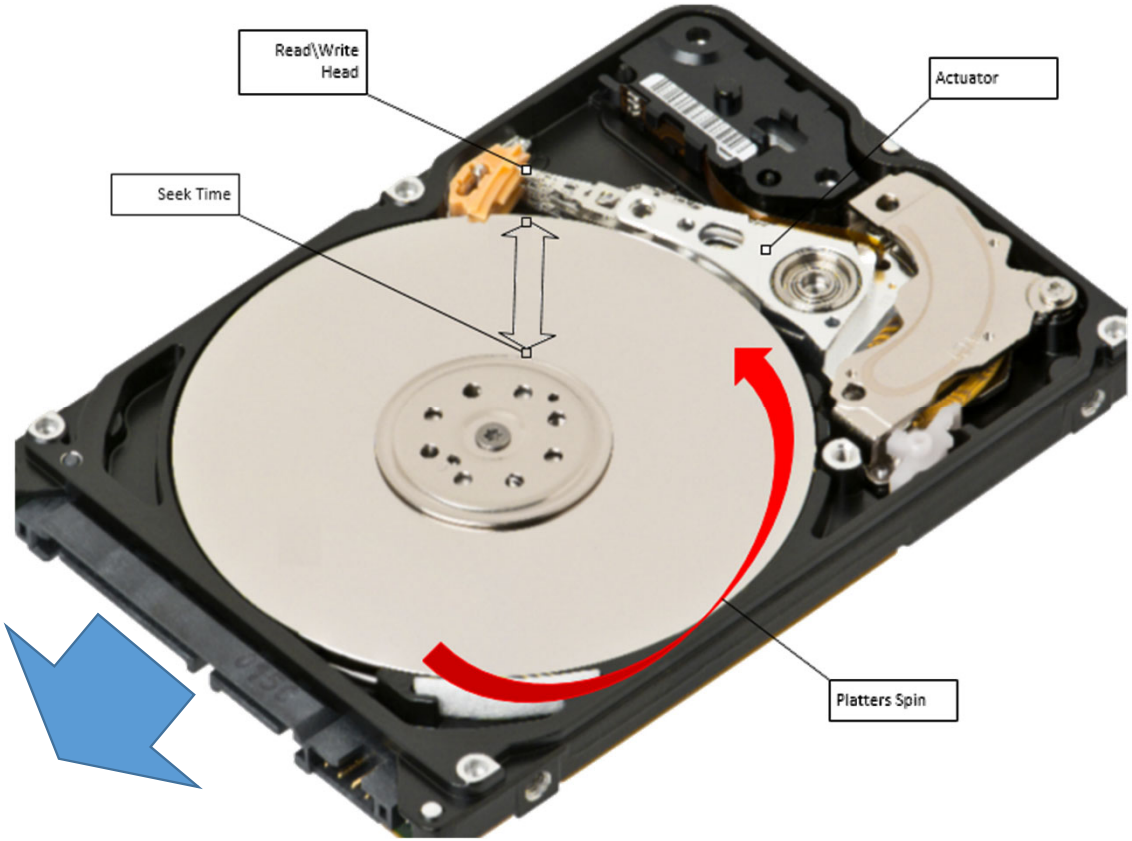
Example of writing replicas



(Distributed) Query processing

Challenge IV

HDDs costs



Rotational Latency

The amount of time taken for the platters to spin the data under the head (measured in RPM)

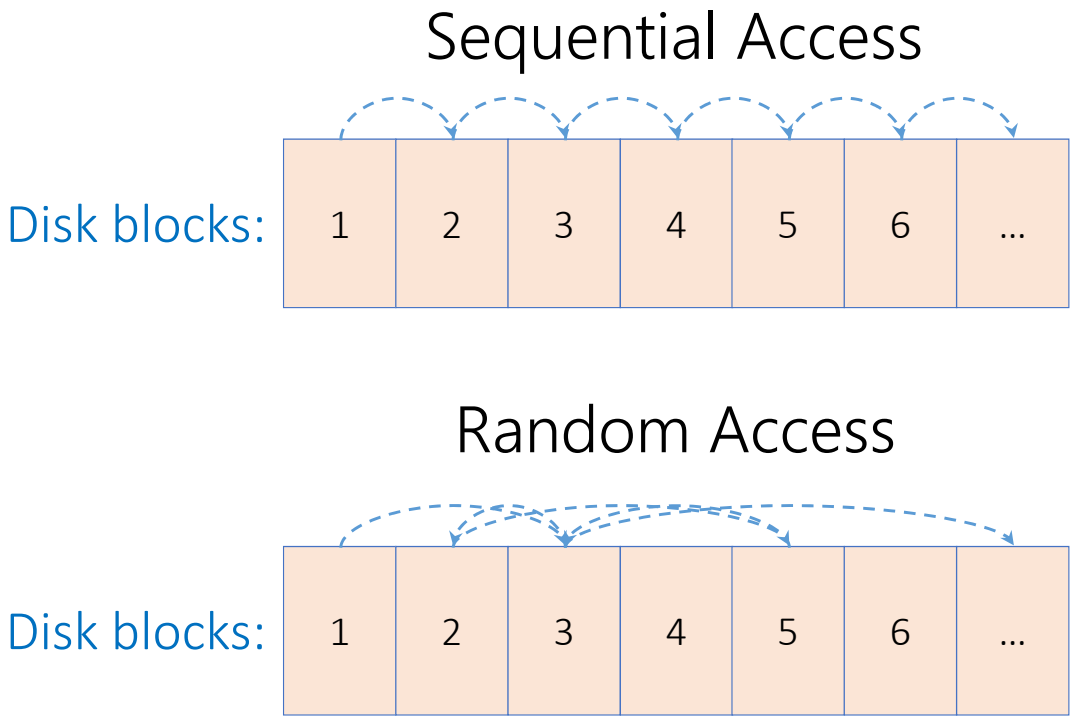
Seek Time

Time taken for the ReadWrite head (mechanical arm) to move between cylinders on the disk

Transfer Time

Time taken for requests to get from the system to the disk (depends on the block size, e.g., 8KB)

Sequential vs. Random access



Cost of accessing data (approximations)

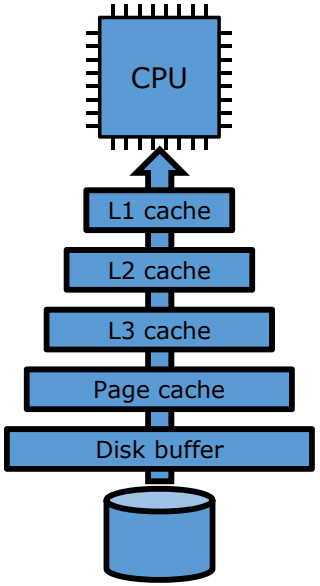
- Sequential reads
  - Option to maximize the effective read ratio
    - Depends on DB design
  - Enables pre-fetching

  Cost = seek+rotation+n*transfer
- Random Access
  - Requires indexing structures
  - Ignores data locality

  $Cost_{single\ cylinder\ files}$ = seek+n*(rotation+transfer)

  $Cost_{multi\text{-}cylinder\ files}$ = n*(seek+rotation+transfer)

seek	~12ms
rotation	~3ms
transfer (8KB)	~0.03ms



Closing

Summary

- GFS architecture and components
- GFS main operations
  - Fault tolerance
  - Writing files and maintenance of replicas
  - Reading files
- HDFS file formats
  - Horizontal
  - Vertical
  - Hybrid

References

- S. Ghemawat et al. *The Google File System*. OSDI'03
- K. V. Shvachko. *HDFS scalability: the limits to growth*. 2010
- S. Abiteboul et al. *Web data management*. Cambridge University Press, 2011
- A. Jindal et al. *Trojan data layouts: right shoes for a running elephant*. SOCC, 2011
- F. Färber et al. *SAP HANA database: data management for modern business applications*. SIGMOD, 2011
- V. Raman et al. *DB2 with BLU Acceleration: So Much More than Just a Column Store*. VLDB, 2013
- D. Abadi, et al. *Column-stores vs. row-stores: how different are they really?* SIGMOD Conference, 2008
- M. Stonebraker et al. *C-Store: A Column-oriented DBMS*. VLDB, 2005
- G. Copeland and S. Khoshafian. *A Decomposition Storage Model*. SIGMOD Conference, 1985
- F. Munir. *Storage Format Selection and Optimization of Materialized Intermediate Results in Data-Intensive Flows*. PhD Thesis (UPC), 2019
- A. Hogan. *Procesado de Datos Masivos* (U. Chile)