Hadoop Distributed File System

Big Data Management

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

DTIM www.essi.upc.edu/dtim

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

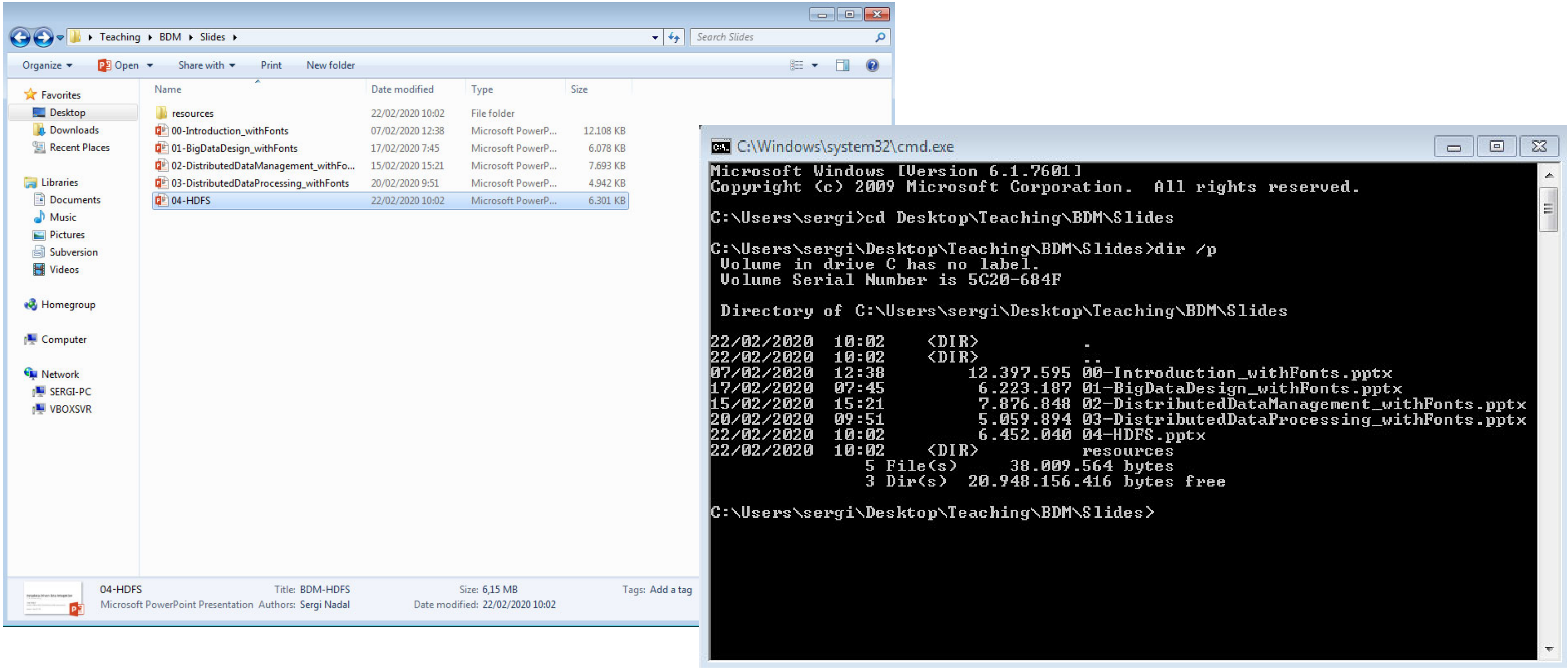
Understanding objectives

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

What is a file system?



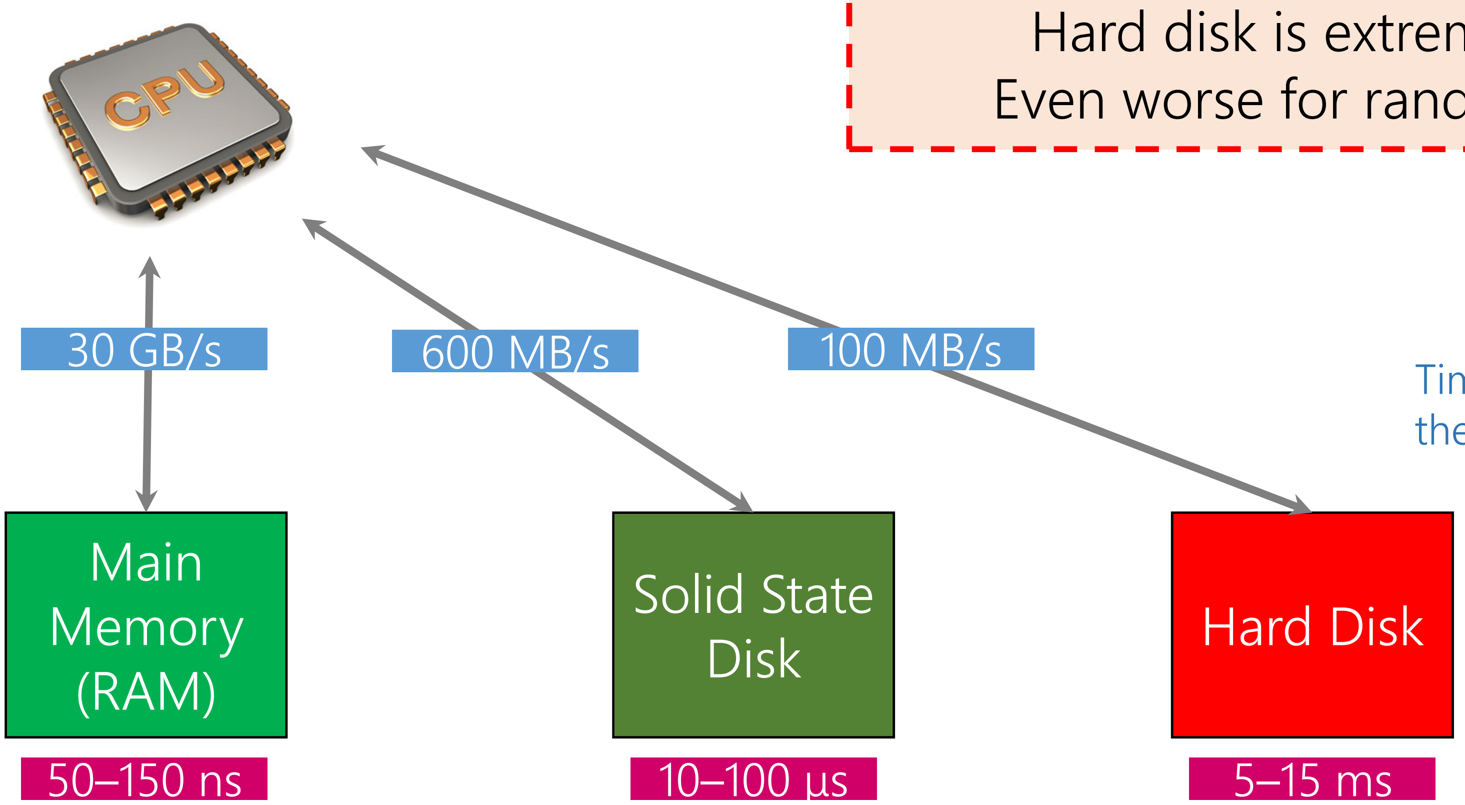
Name	Date modified	Type	Size
resources	22/02/2020 10:02	File folder	
00-Introduction_withFonts	07/02/2020 12:38	Microsoft PowerP...	12.108 KB
01-BigDataDesign_withFonts	17/02/2020 7:45	Microsoft PowerP...	6.078 KB
02-DistributedDataManagement_withFo...	15/02/2020 15:21	Microsoft PowerP...	7.693 KB
03-DistributedDataProcessing_withFonts	20/02/2020 9:51	Microsoft PowerP...	4.942 KB
04-HDFS	22/02/2020 10:02	Microsoft PowerP...	6.301 KB

04-HDFS
Microsoft PowerPoint Presentation
Title: BDM-HDFS
Authors: Sergi Nadal
Size: 6,15 MB
Date modified: 22/02/2020 10:02
Tags: Add a tag

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\sergi>cd Desktop\Teaching\BDM\Slides

C:\Users\sergi\Desktop\Teaching\BDM\Slides>dir /p
 Volume in drive C has no label.
 Volume Serial Number is 5C20-684F

 Directory of C:\Users\sergi\Desktop\Teaching\BDM\Slides

22/02/2020  10:02    <DIR>          .
22/02/2020  10:02    <DIR>          ..
07/02/2020  12:38        12.397.595 00-Introduction_withFonts.pptx
17/02/2020  07:45         6.223.187 01-BigDataDesign_withFonts.pptx
15/02/2020  15:21         7.876.848 02-DistributedDataManagement_withFonts.pptx
20/02/2020  09:51         5.059.894 03-DistributedDataProcessing_withFonts.pptx
22/02/2020  10:02         6.452.040 04-HDFS.pptx
22/02/2020  10:02    <DIR>          resources
               5 File(s)     38.009.564 bytes
               3 Dir(s)  20.948.156.416 bytes free

C:\Users\sergi\Desktop\Teaching\BDM\Slides>
```

Time to bring data (approximations)

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



30 GB/s

600 MB/s

100 MB/s

Main Memory (RAM)

Solid State Disk

Hard Disk

50–150 ns

10–100 µs

5–15 ms

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

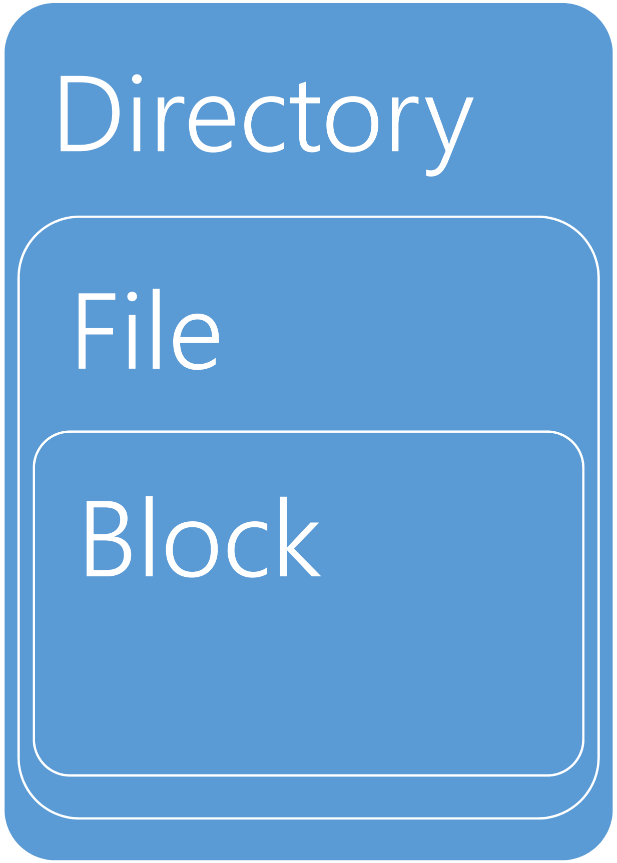
Faster ✓

✓ Cheaper
✓ Persistant

Hard Drive Disk

Functionalities provided by a FS

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



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
  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

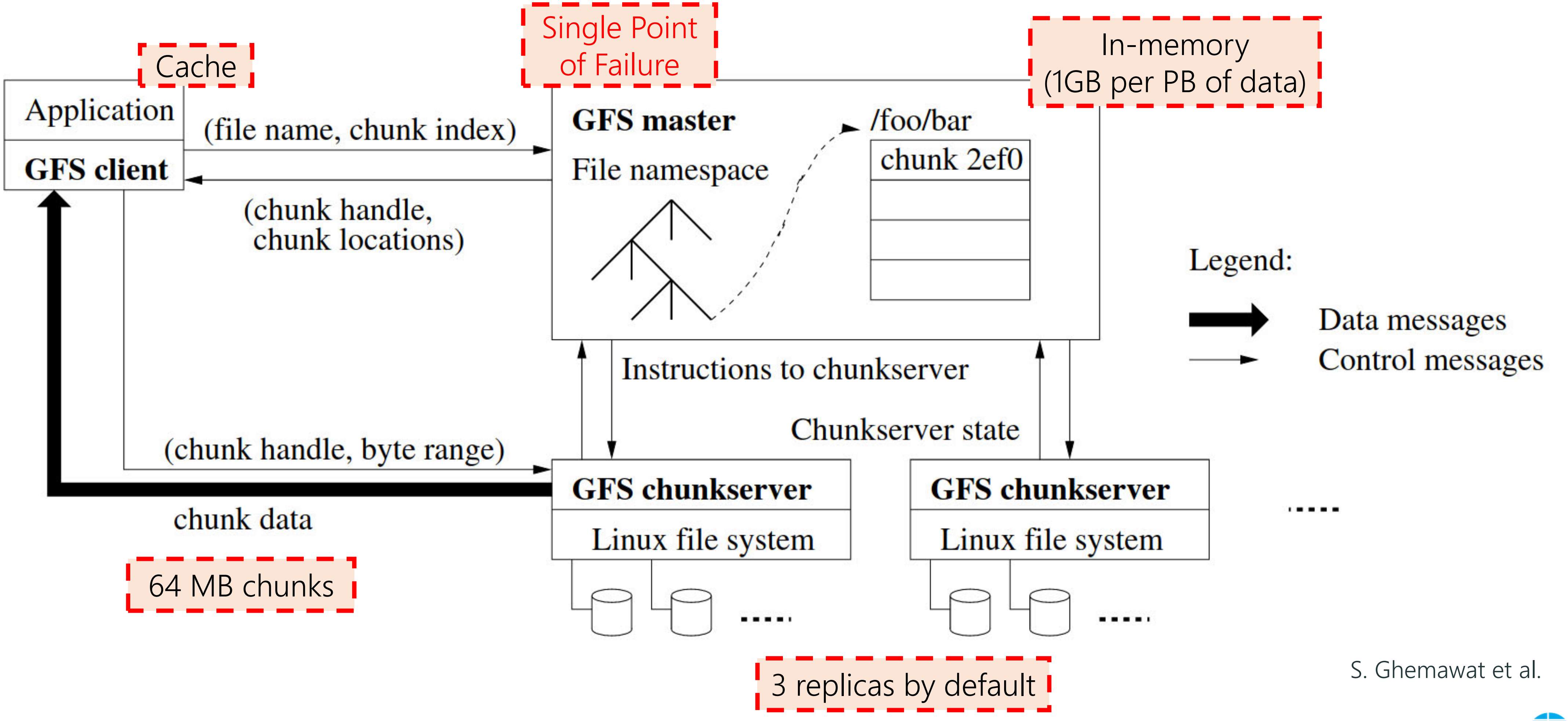
Capacity	Nodes	Clients	File
10 PB	10.000	100.000	100.000.000

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)
- Sequential scans optimized
  - Overcome the high latency of HDDs (5-15ms) compared to main memory (50-150ns)

GFS Architecture

HDFS Architecture



Cache

Single Point of Failure

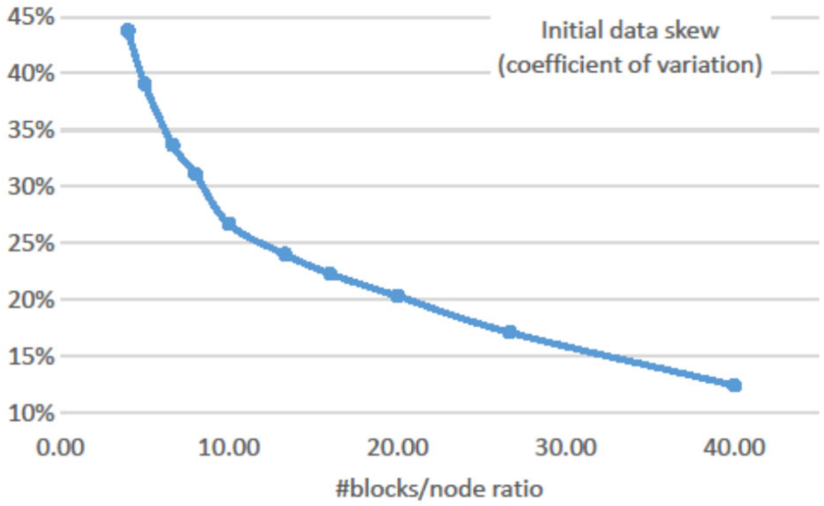
In-memory (1GB per PB of data)

64 MB chunks

3 replicas by default

S. Ghemawat et al.

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

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

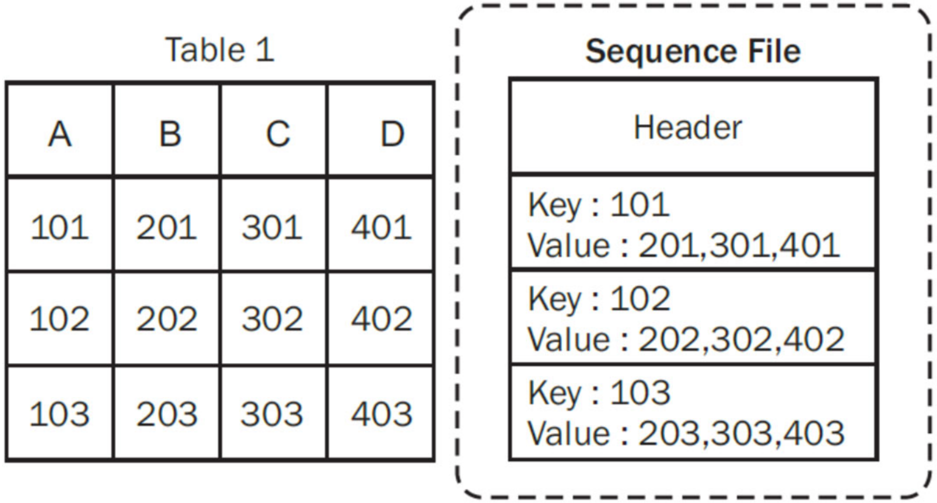


Table 1

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

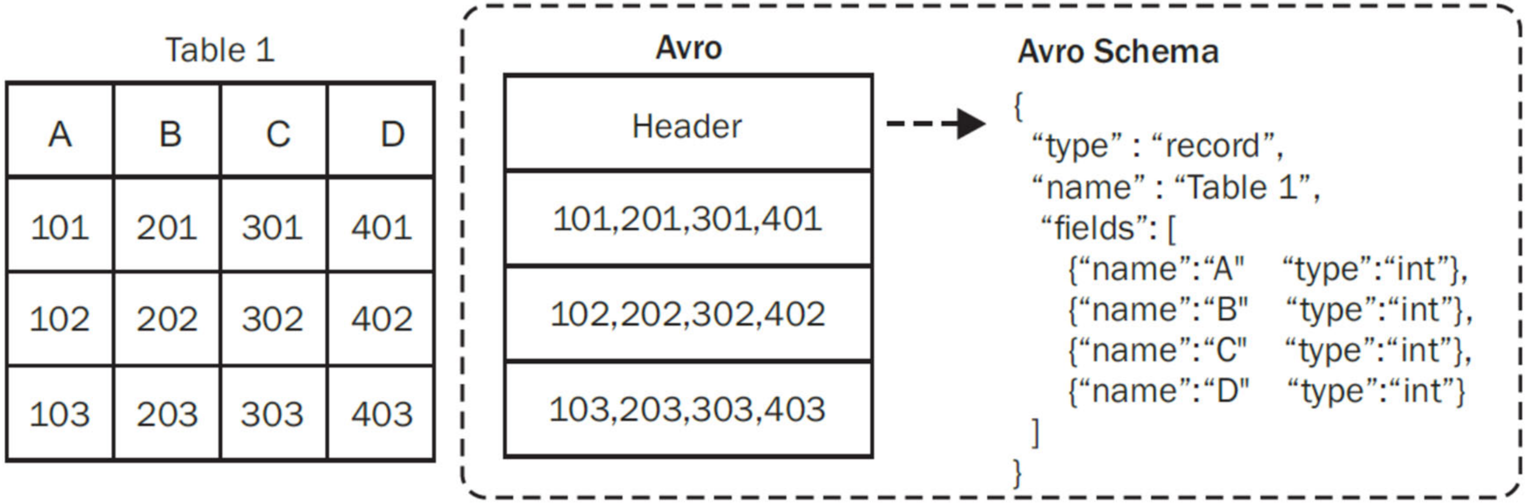


Table 1

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

Avro

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

Avro Schema

```
{
 “type” : “record”,
 “name” : “Table 1”,
  “fields”: [
    {“name”:“A"   “type”:“int”},
    {“name”:“B"   “type”:“int”},
    {“name”:“C"   “type”:“int”},
    {“name”:“D"   “type”:“int”}
 ]
}
```

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

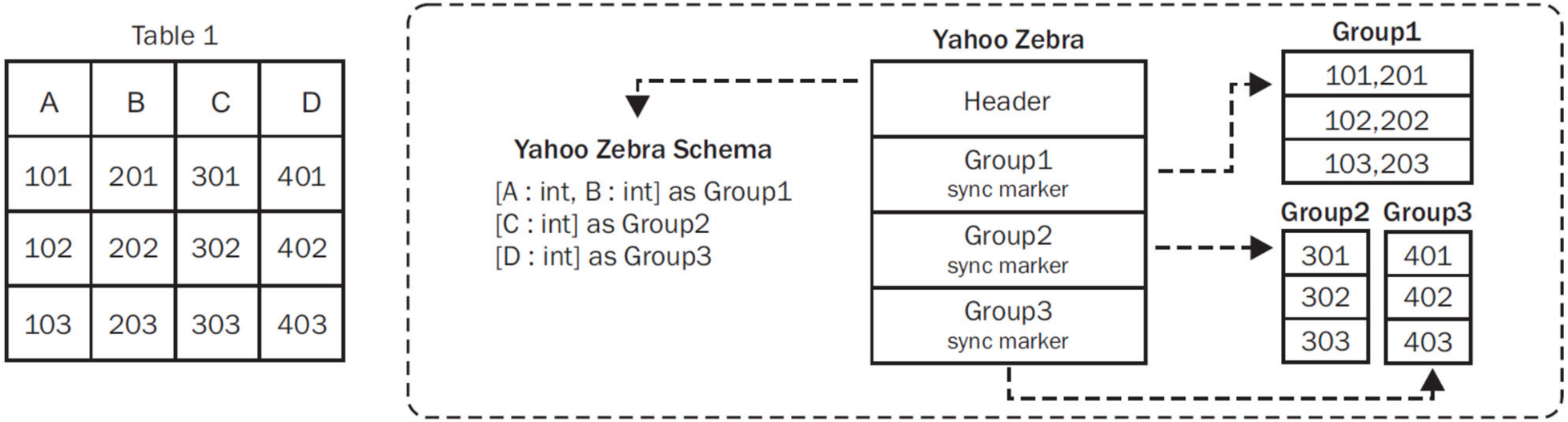


Table 1

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

Yahoo Zebra Schema

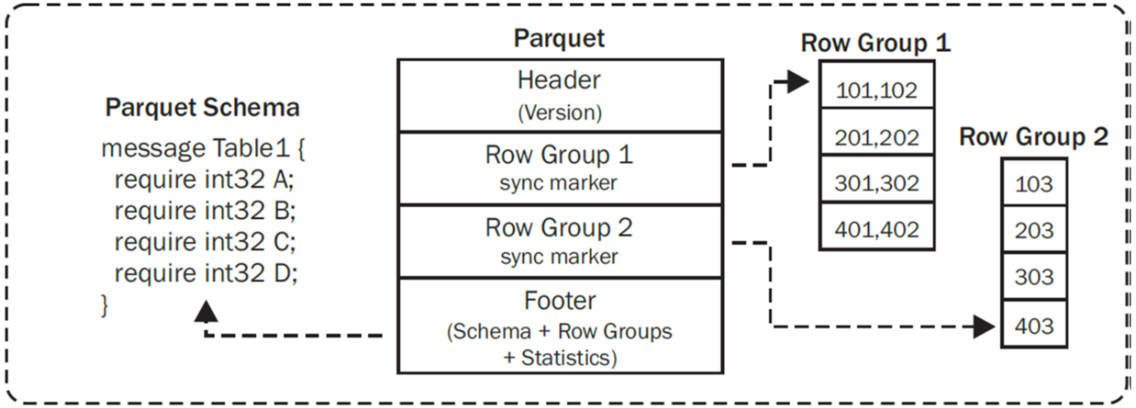
[A : int, B : int] as Group1
[C : int] as Group2
[D : int] as Group3

Hybrid layout - Parquet

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

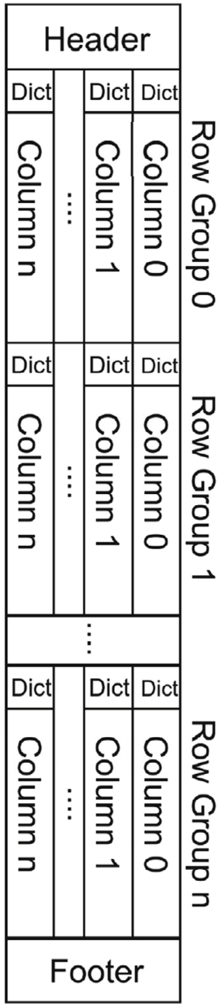
Table 1

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



Parquet Schema

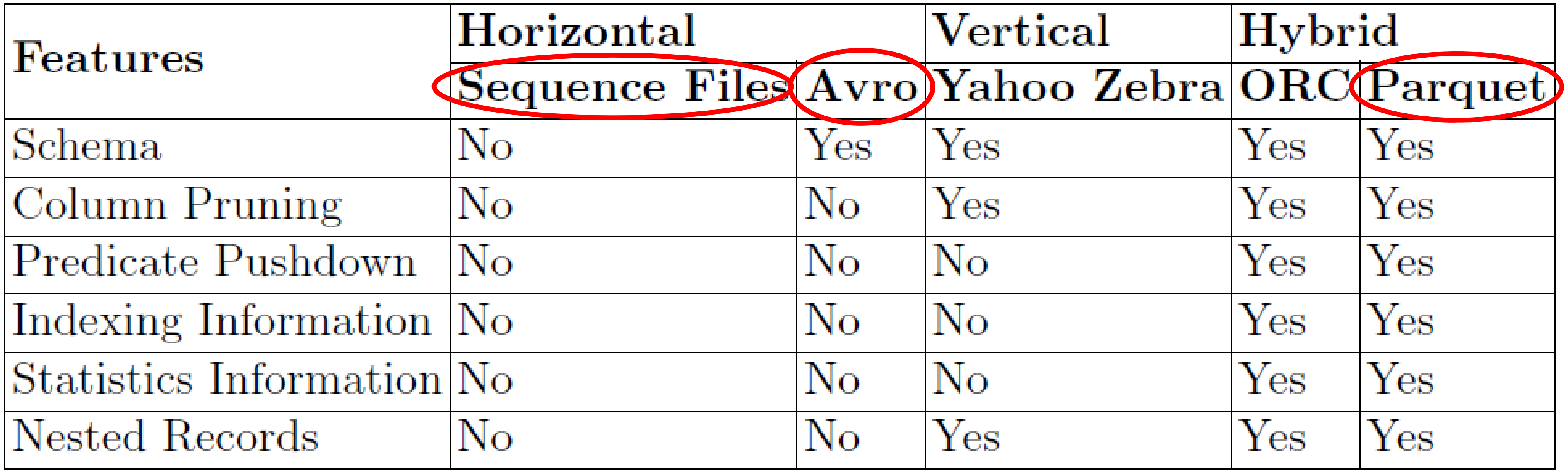
```
message Table1 {
  require int32 A;
  require int32 B;
  require int32 C;
  require int32 D;
}
```



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
  - $\exists e \in O(v)$, getType(e) = {AggregationOps}
  - $\exists e \in O(v)$, getCol(getOP (e)) $\subset$ getCol(v)
- Avro
  - $\forall e \in O(v)$, getCol(getOP (e)) = getCol(v)
  - $\exists e \in O(v)$, getType(e) $\in$ {Join, CartesianProduct, GroupALL, Distinct}

Cost-based choice

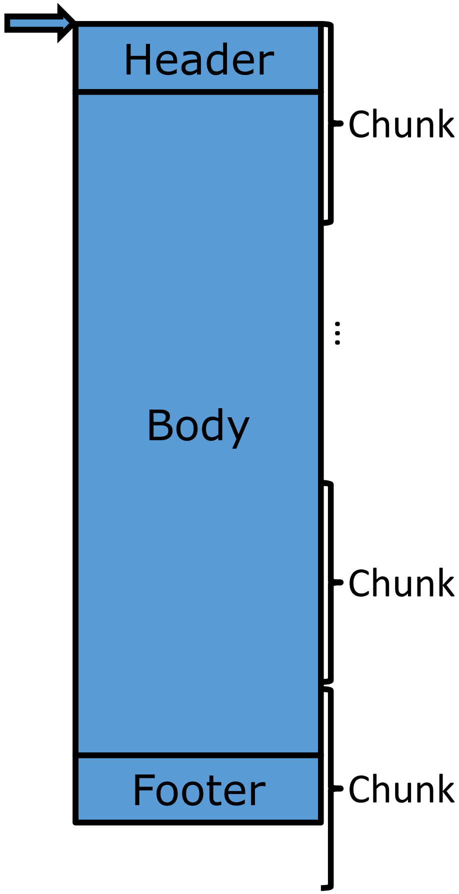
- Helps in choosing the right storage layout based on the workloads
- Costs to Consider
  - Write cost
  - Read Cost
    - Scan Operation
    - Projection Operation
    - Selection Operation
- Costs ignored
  - Block compression
  - Dictionary encoding (in Parquet)

General formulas for size estimation

$$\begin{aligned} Size(Layout) = {} & Size(Header_{Layout}) \\ & + Size(Body_{Layout}) \\ & + Size(FooterLayout) \end{aligned}$$

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

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

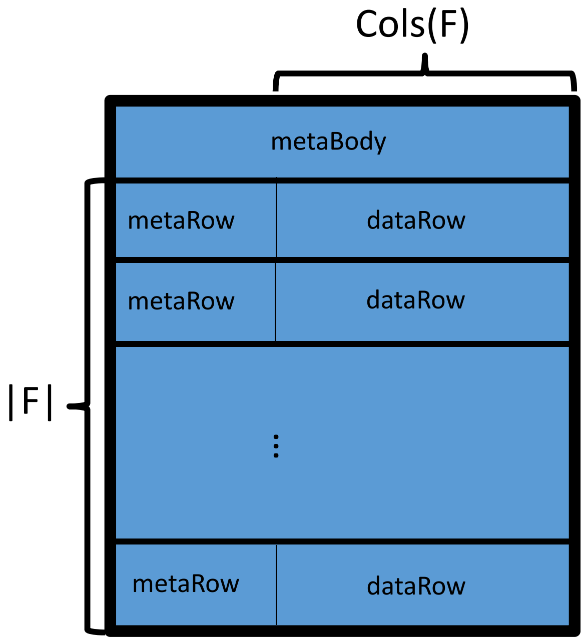


“Layout” can be either “Horizontal”, “Vertical” or “Hybrid”

Horizontal layout size estimation

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

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

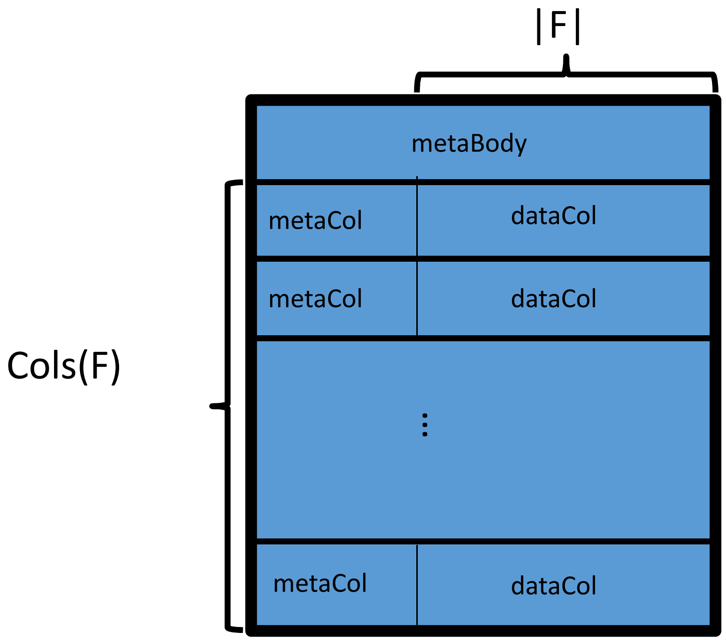


“F” is the content of the file

Vertical layout size estimation

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

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



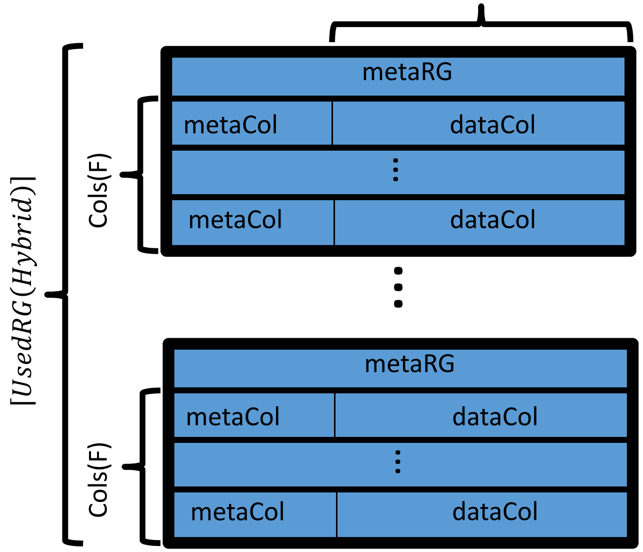
“F” is the content of the file

Hybrid layout size estimation

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

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

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



“F” is the content of the file

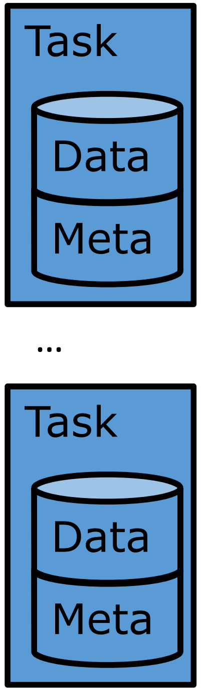
General formulas for cost estimation

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

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

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

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



“Layout” can be either “Horizontal”, “Vertical” or “Hybrid”

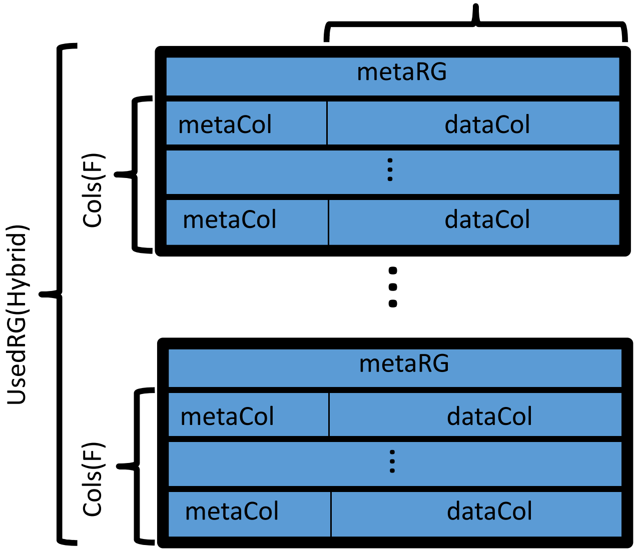
Projection in hybrid layouts

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

$$\begin{aligned} Size(Project_{Hybrid}) = {} & Size\left(Header_{Hybrid}\right) + Size\left(Footer_{Hybrid}\right) \\ & + \lceil UsedRG(Hybrid) \rceil * (Size(metaRG) + proj(F) * Size(metaCol)) \\ & + UsedRG(Hybrid) * Size(projCols) \end{aligned}$$

$$\begin{aligned} Cost(Project_{Hybrid}) = {} & UsedChunks(Project_{Hybrid}) * W_{ReadTransfer} \\ & + Seeks(Hybrid) * (1 - W_{ReadTransfer}) \end{aligned}$$

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



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

Probability of retrieving a RowGroup

- Probability of a row fulfilling P
  SF

- Probability of a row NOT fulfilling P
  1-SF

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

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

Selection in hybrid layouts

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

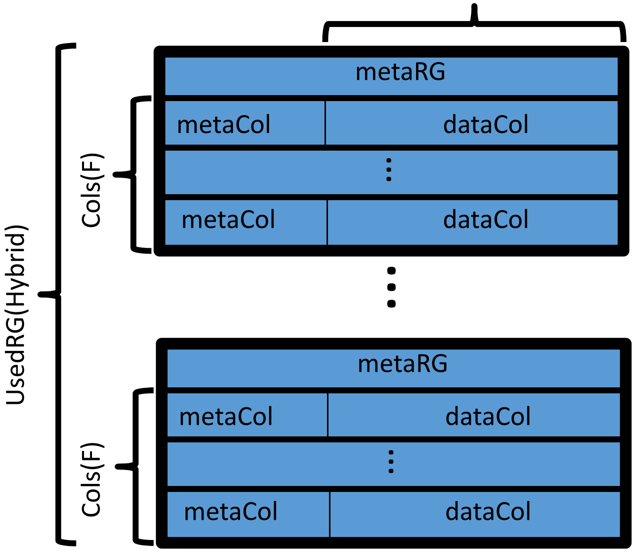
$$Size(RowsSelected) = \left\lceil \frac{SF * |F|}{UsedRows(RG)} \right\rceil \big(Size(metaRG) + Cols(F) * Size(metaCol)\big)$$
$$+ SF * |F| * Cols(F) * Size(cell)))$$

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

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

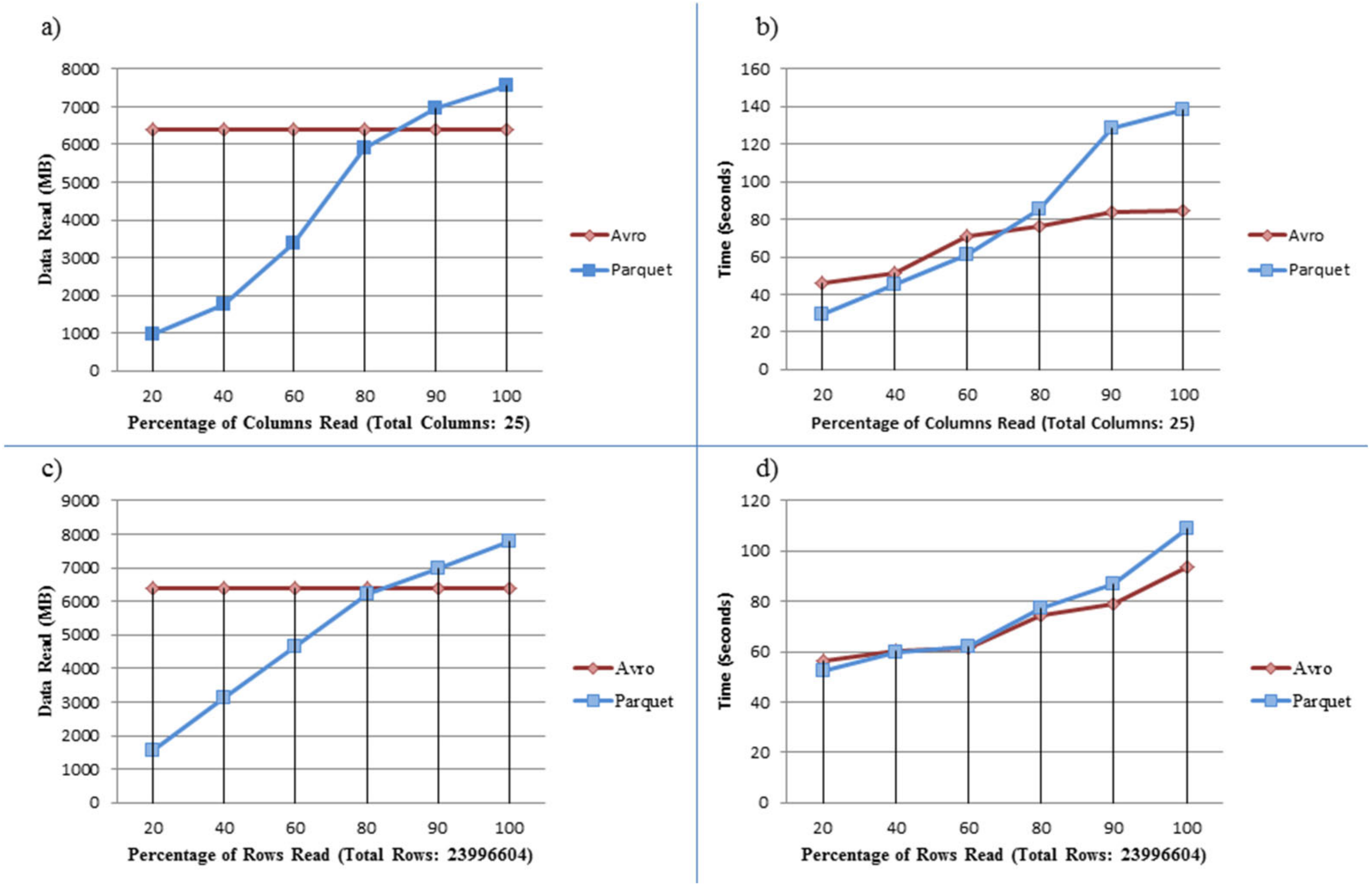
$$Cost(Select_{Hybrid}) = UsedChunks(Select_{Hybrid}) * W_{ReadTransfer}$$
$$+ Seeks(Select_{Hybrid}) * (1 - W_{ReadTransfer})$$

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



“F” is the content of the file

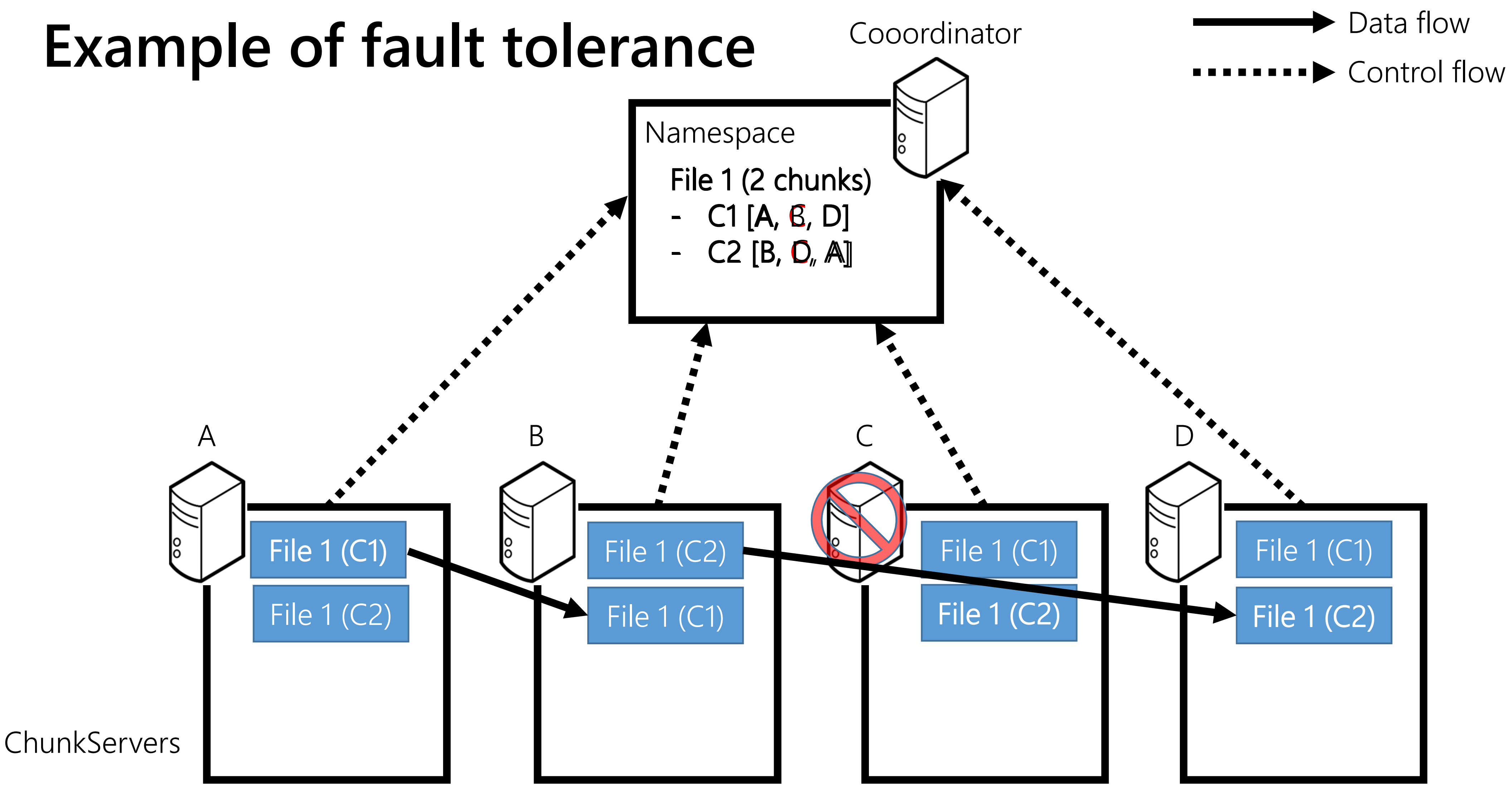
Comparison of selection and projection



Fault tolerance

- Managed from the coordinator
  - It expects to receive every 3 seconds a *heartbeat* message from chunkservers
- If a chunkserver does not send 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



File 1 (2 chunks)

- C1 [A, B, D]
- C2 [B, D, A]

Catalog Management

Challenge II

Client caching

Cache miss

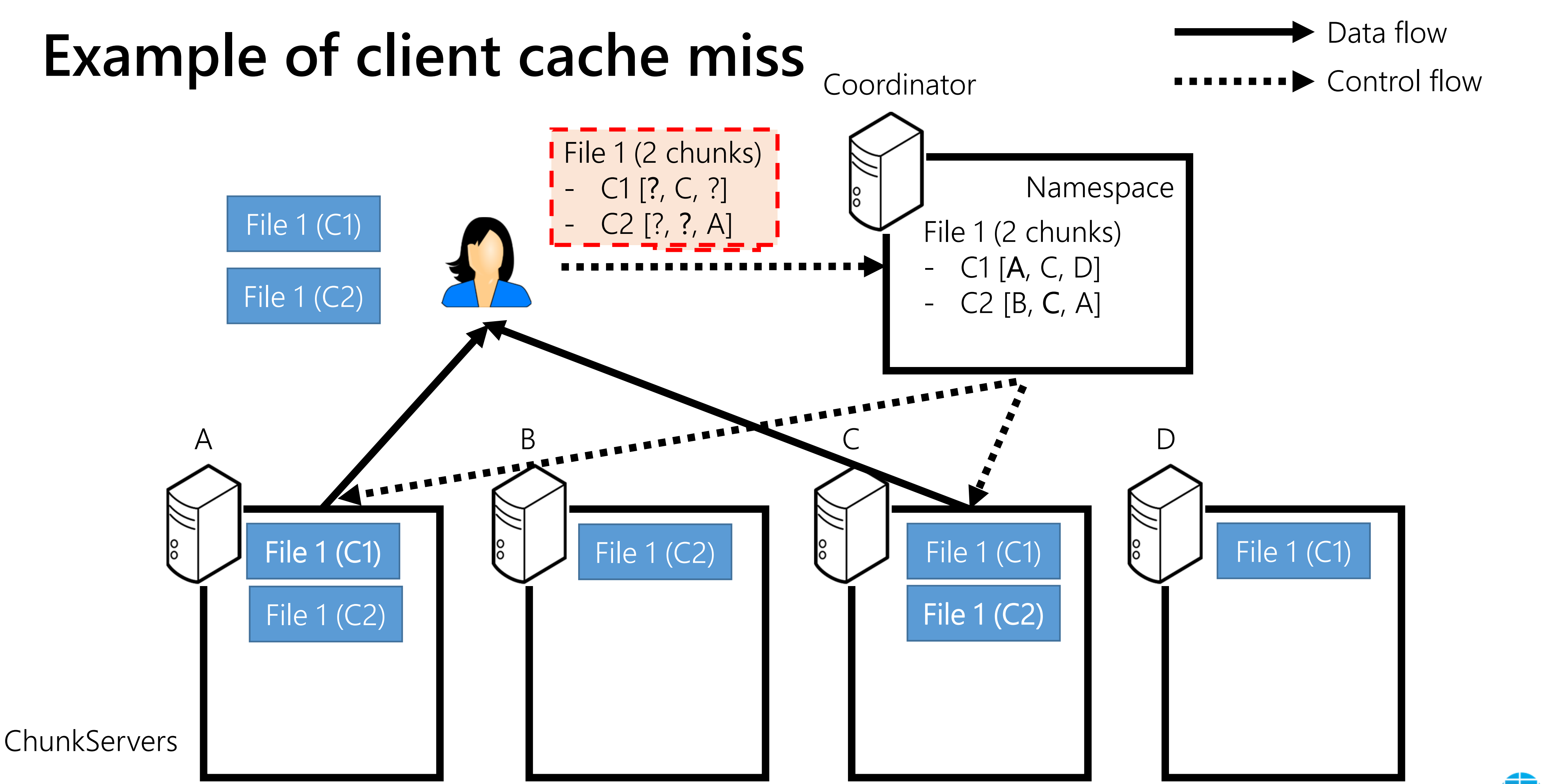
1. The client sends a READ command to the coordinator
2. The coordinator requests chunkservers to send the chunks to the client
   - Ranked according to the closeness in the network
3. The list is cached in the client
   - Not a complete view of all partitions

Cache hit

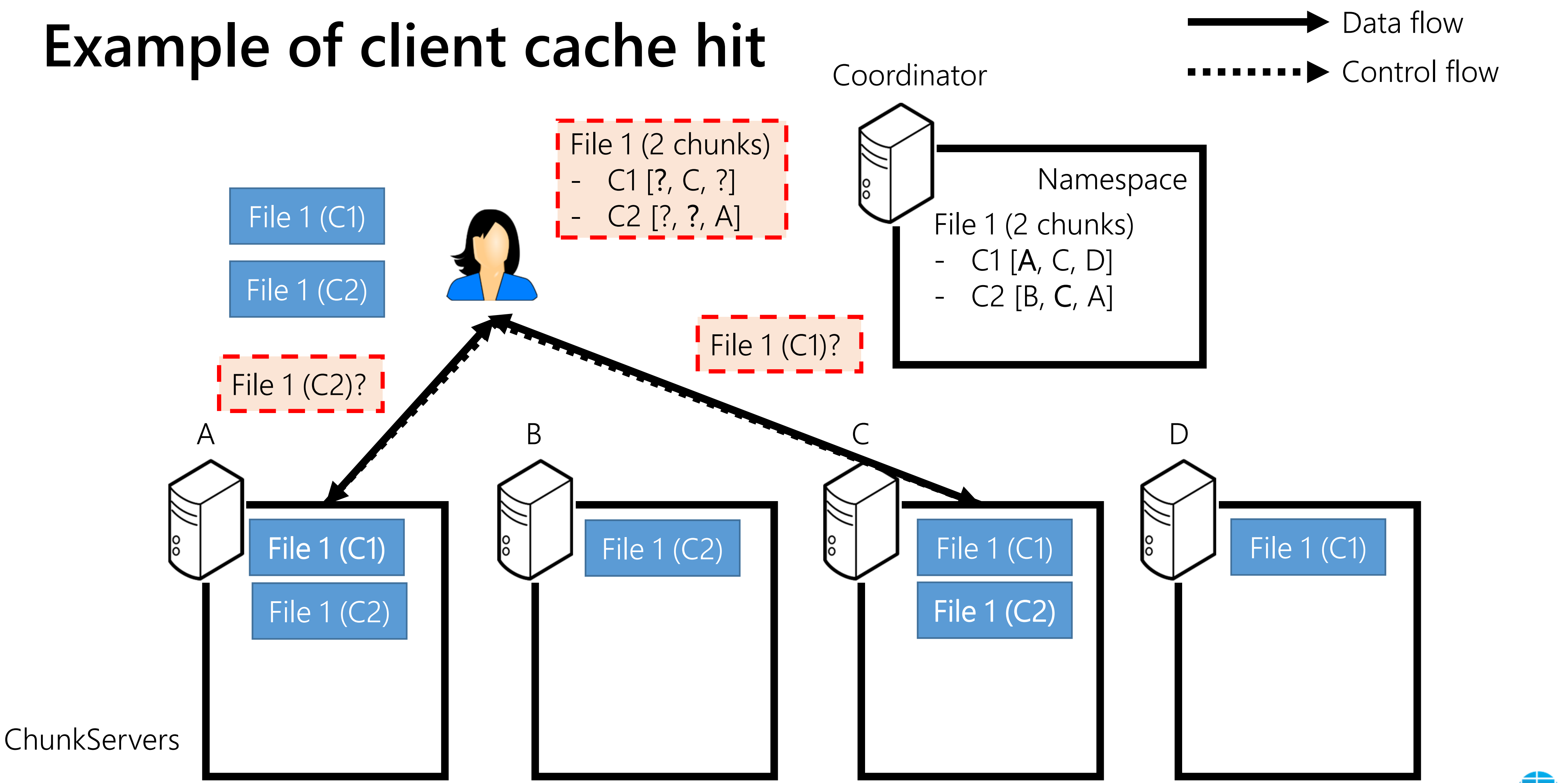
1. The client reads the cache and requests the chunkservers to send the chunks

Avoid coordinator bottleneck
+
One communication step is saved

Example of client cache miss



Example of client cache hit



Data flow

Control flow

Coordinator

Namespace

File 1 (2 chunks)
- C1 [**A**, C, D]
- C2 [B, **C**, A]

File 1 (2 chunks)
- C1 [**?**, C, ?]
- C2 [?, **?**, A]

File 1 (C1)

File 1 (C2)

File 1 (C2)?

File 1 (C1)?

A: **File 1 (C1)**, File 1 (C2)

B: File 1 (C2)

C: File 1 (C1), **File 1 (C2)**

D: File 1 (C1)

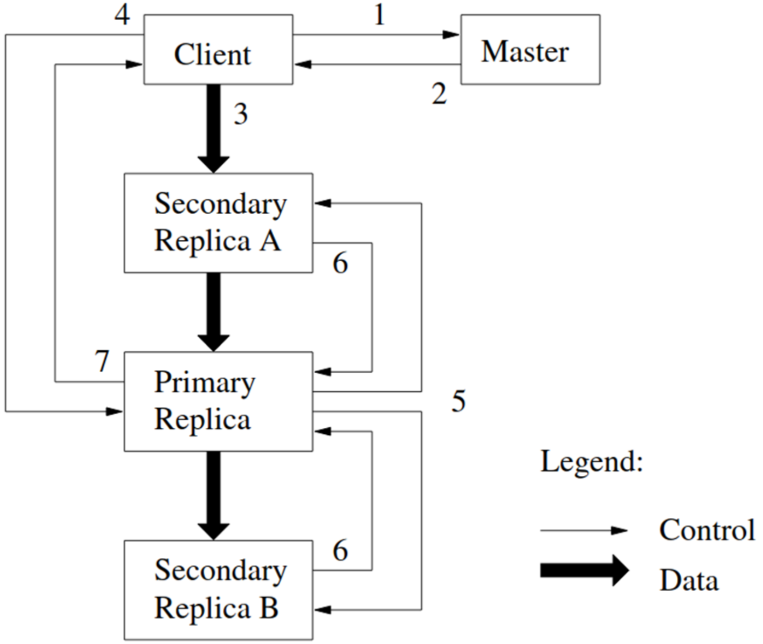
ChunkServers

Transaction Management

Challenge III

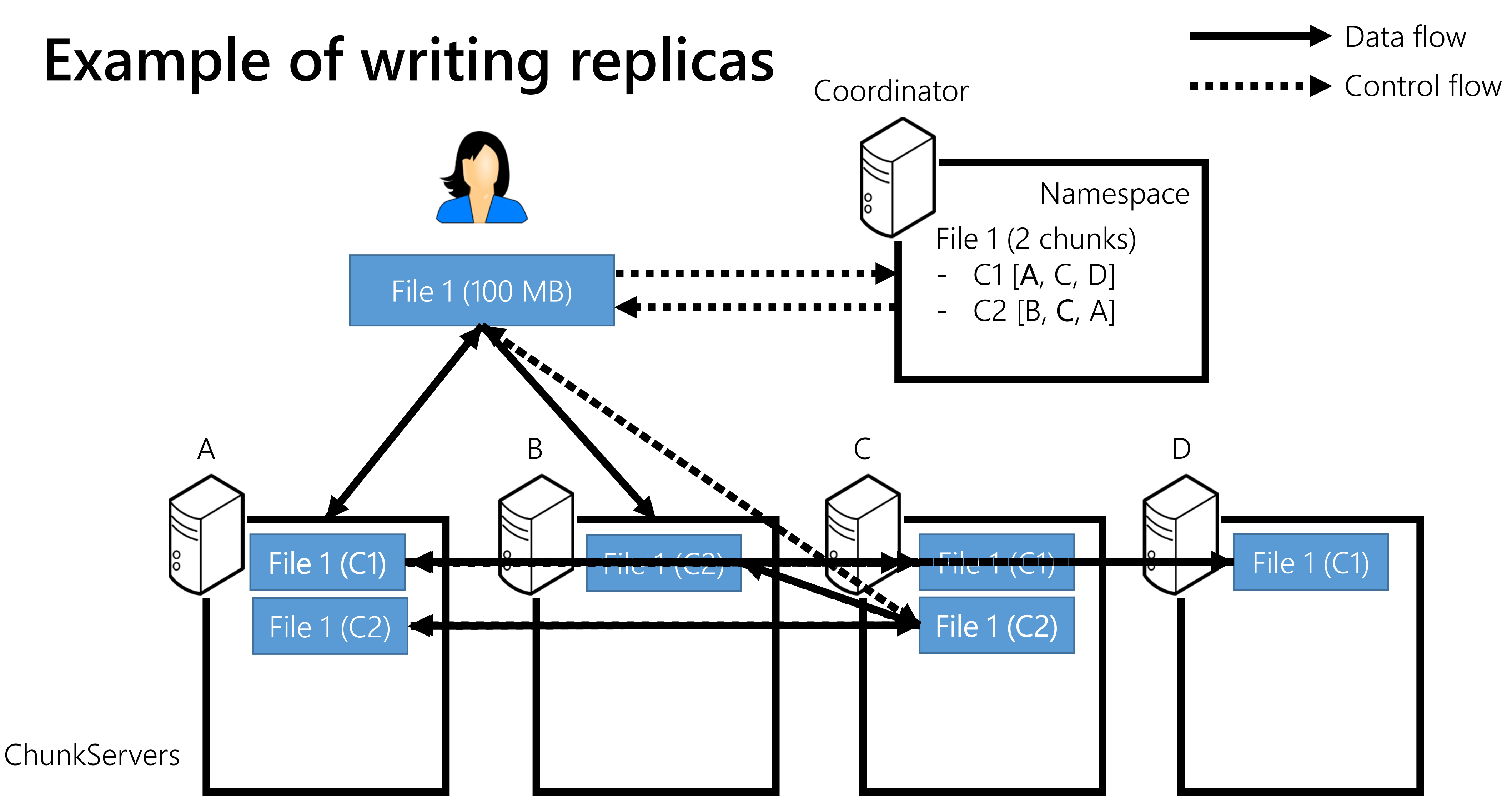
Writing replicas

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 order to primary replica
5. Primary replica sends WRITE order to secondary replicas
6. Secondaries confirm to primary the change
7. Primary confirms to the client



S. Ghemawat et al.

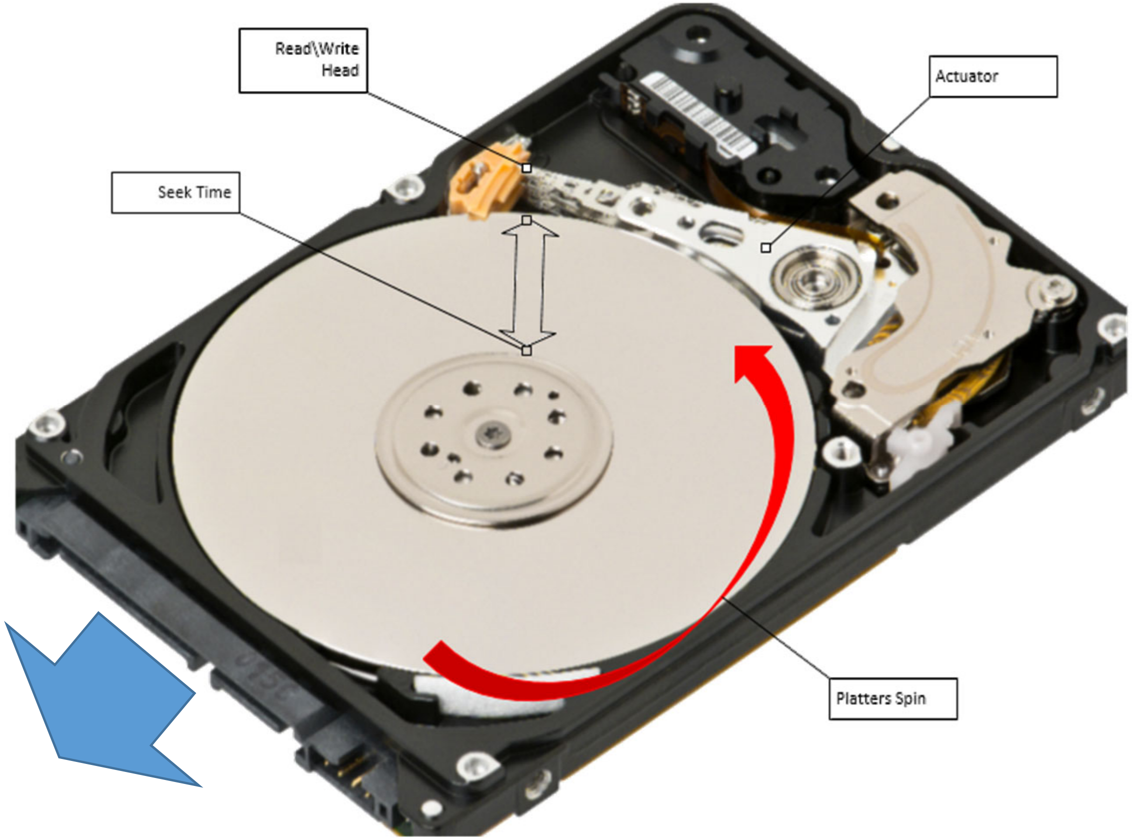
Example of writing replicas



Query processing

Challenge IV

HDDs costs



Rotational Latency

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

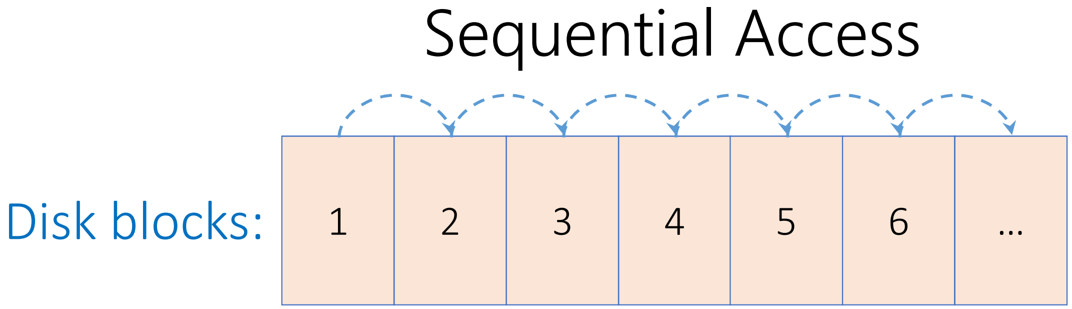
Seek Time

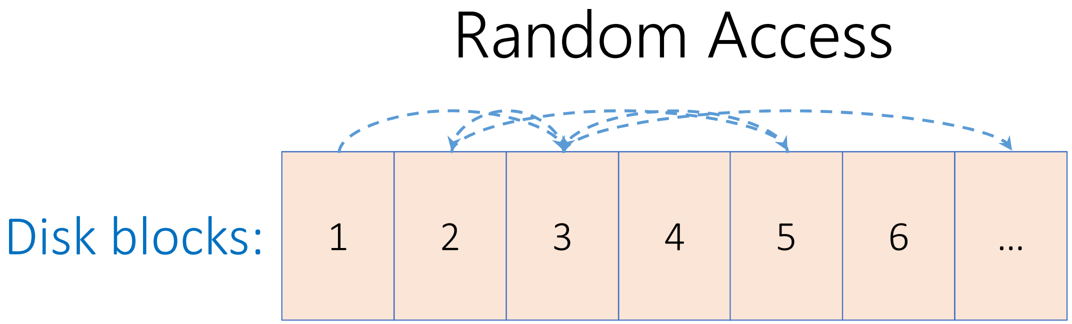
Time it takes for the ReadWrite head (mechanical arm) to move between sectors on the disk

Transfer Time

Time it takes for requests to get from the system to the disk (depends on the block size, e.g., 4KB)

Sequential vs. Random access





Cost of accessing data (approximations)

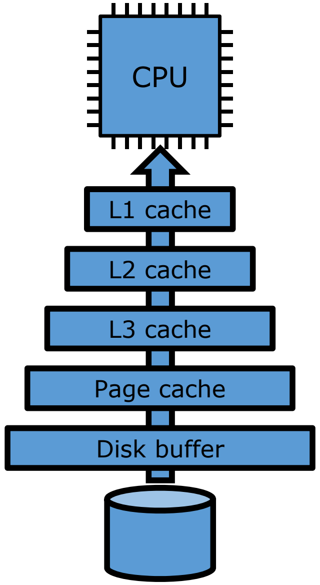
seek	~12ms
rotation	~3ms
transfer (8KB)	~0.03ms

- Random Access
  - Requires indexing structures
  - Ignores data locality

  Cost = n*(seek+rotation+transfer)

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

  Cost = seek+rotation+n*transfer



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
- F. Munir et al. *ResilientStore: A Heuristic-Based Data Format Selector for Intermediate Results*. MEDI, 2016
- 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*: PhD Thesis (UPC)
- *A.* Hogan: Procesado de Datos Masivos (U. Chile)