Hadoop Distributed File System

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

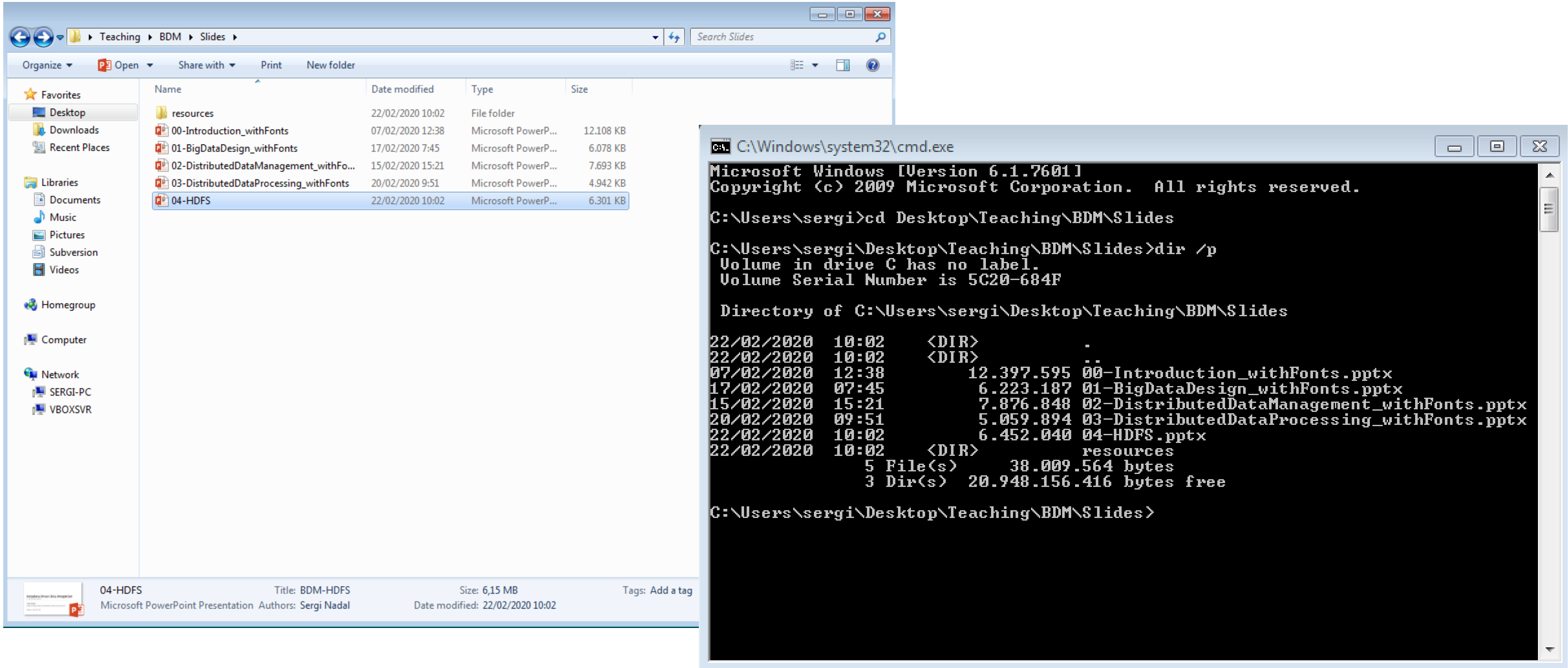
Understanding objectives

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

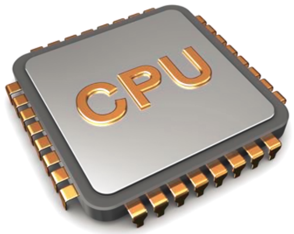
Google File System

What is a file system?



Time to bring data (approximations)

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



	Main Memory (RAM)	Solid State Disk	Rotating Disk
Data transfer	30 GB/s	600 MB/s	100 MB/s
Latency	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

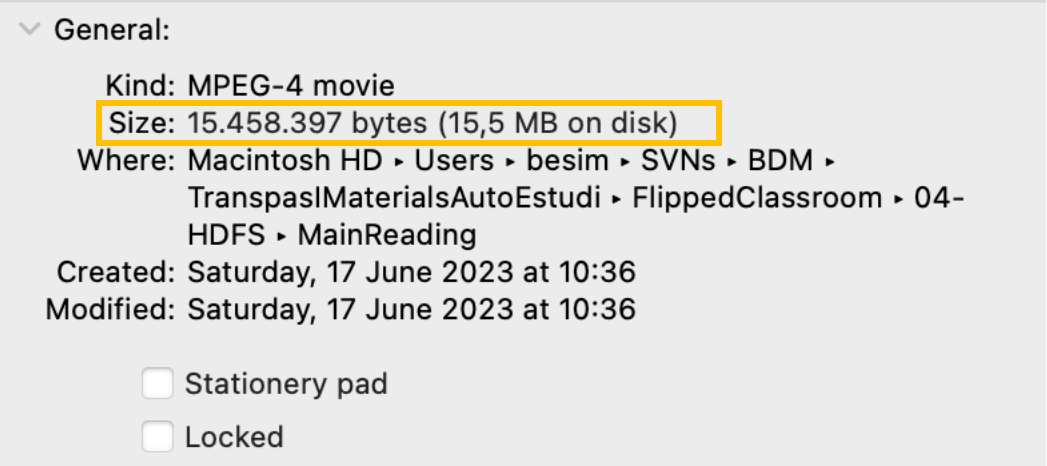
Faster ✓

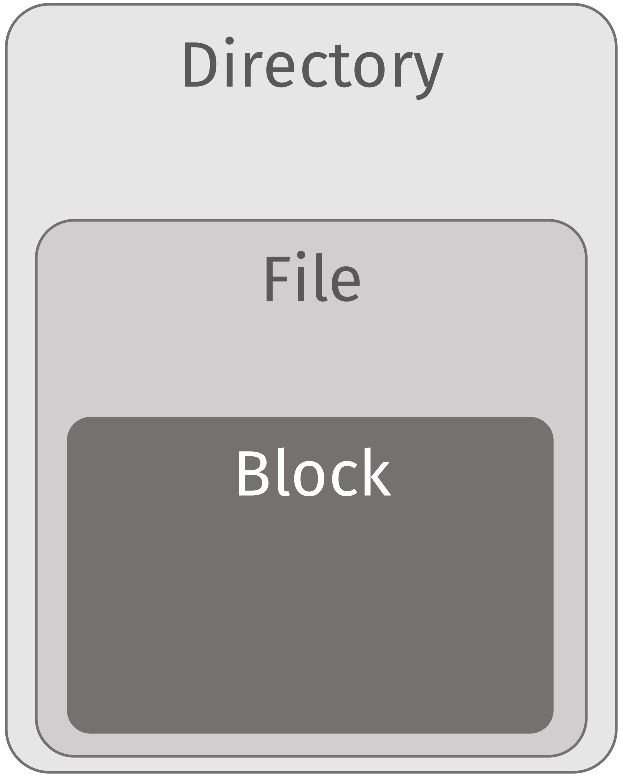
✓ Cheaper
✓ Persistent

Hard Drive Disk

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
    - Expected node failure rate: 2-3 node failures per 1.000 per day
- Sequential scans optimized
  - Overcome high latency of HDDs (5-15ms) compared to main memory (50-150ns)

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require

1. INTRODUCTION

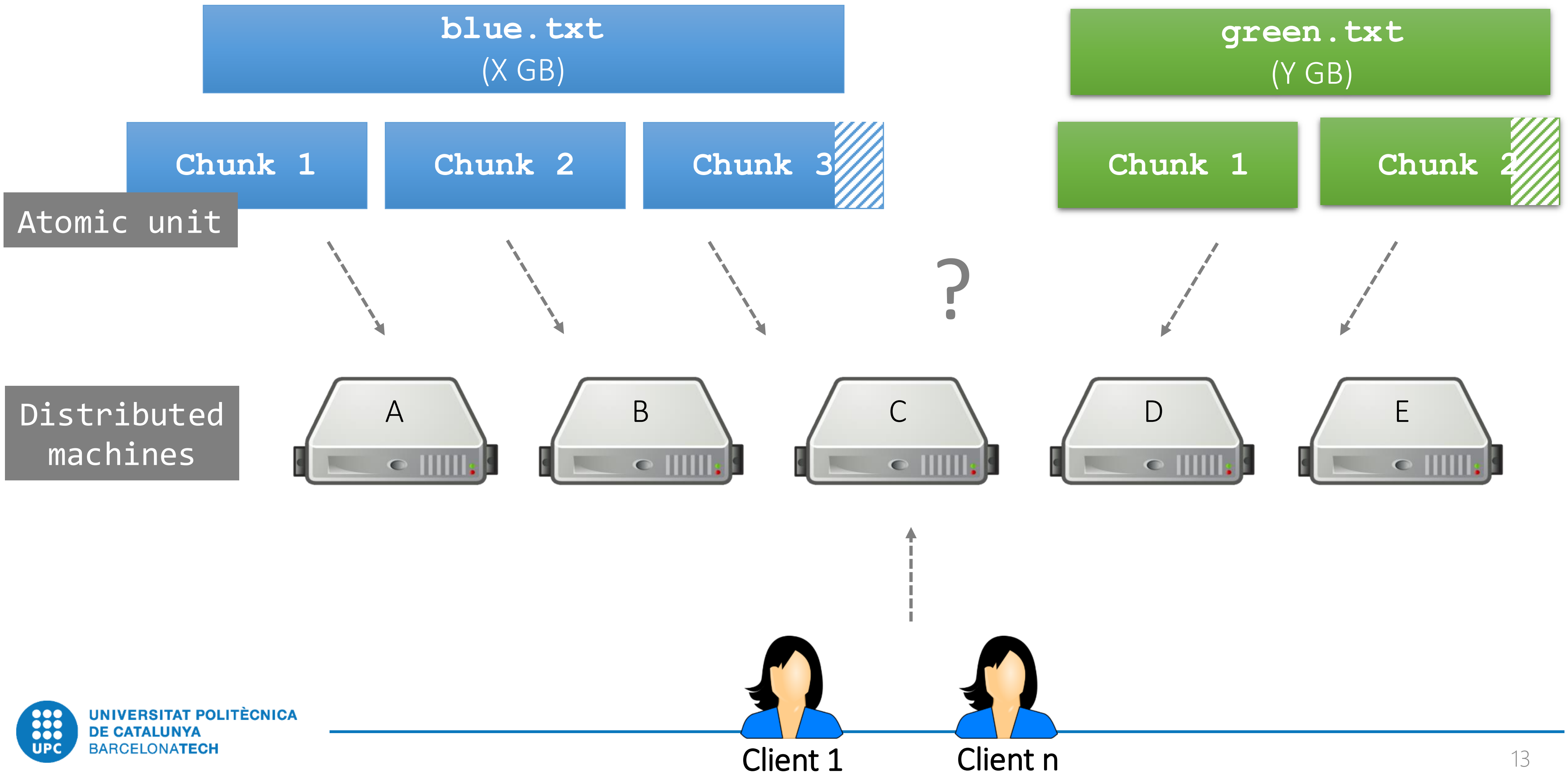
We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the compo-

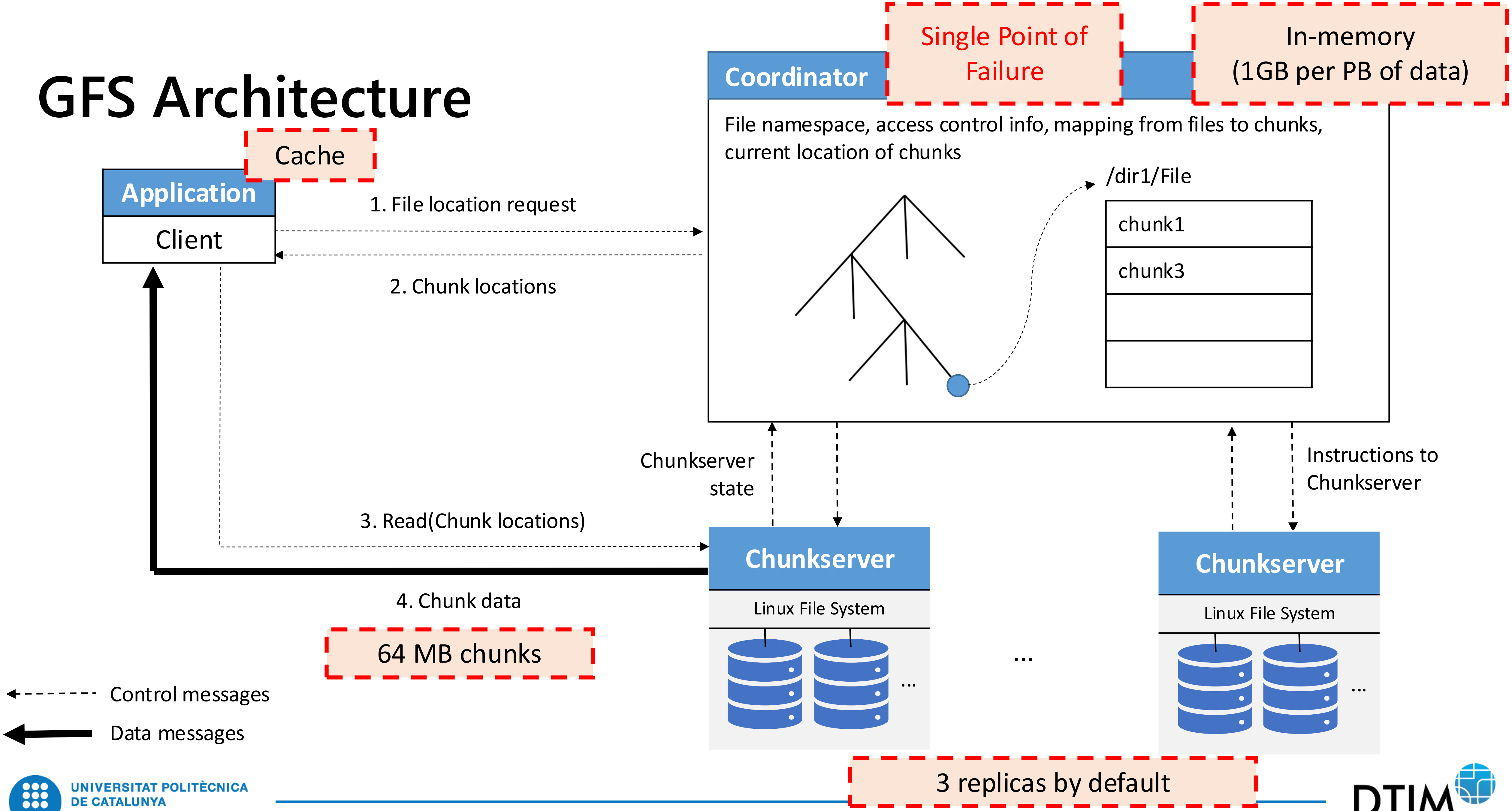
GFS Architecture

SOSP '03

Distributed files: fragmentation into fixed-size chunks

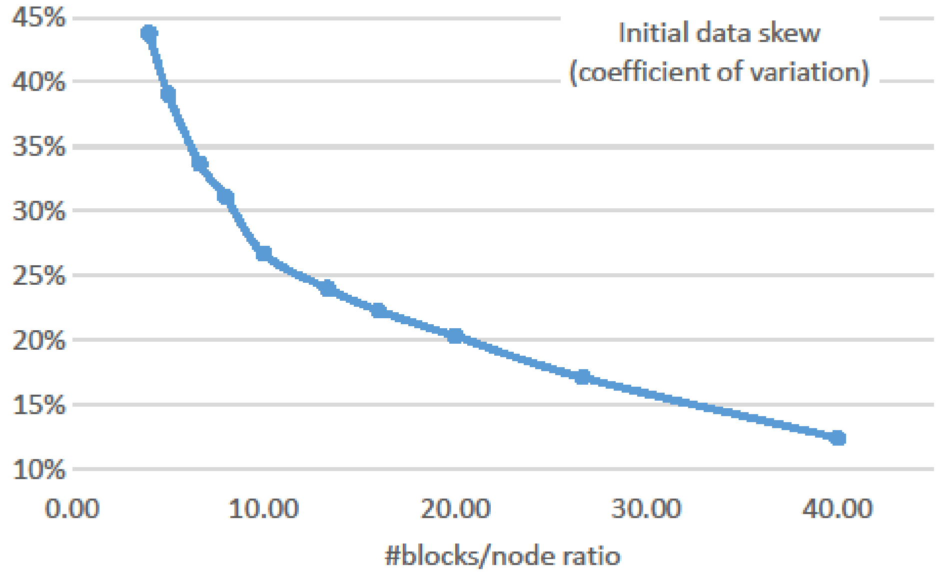


GFS Architecture



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

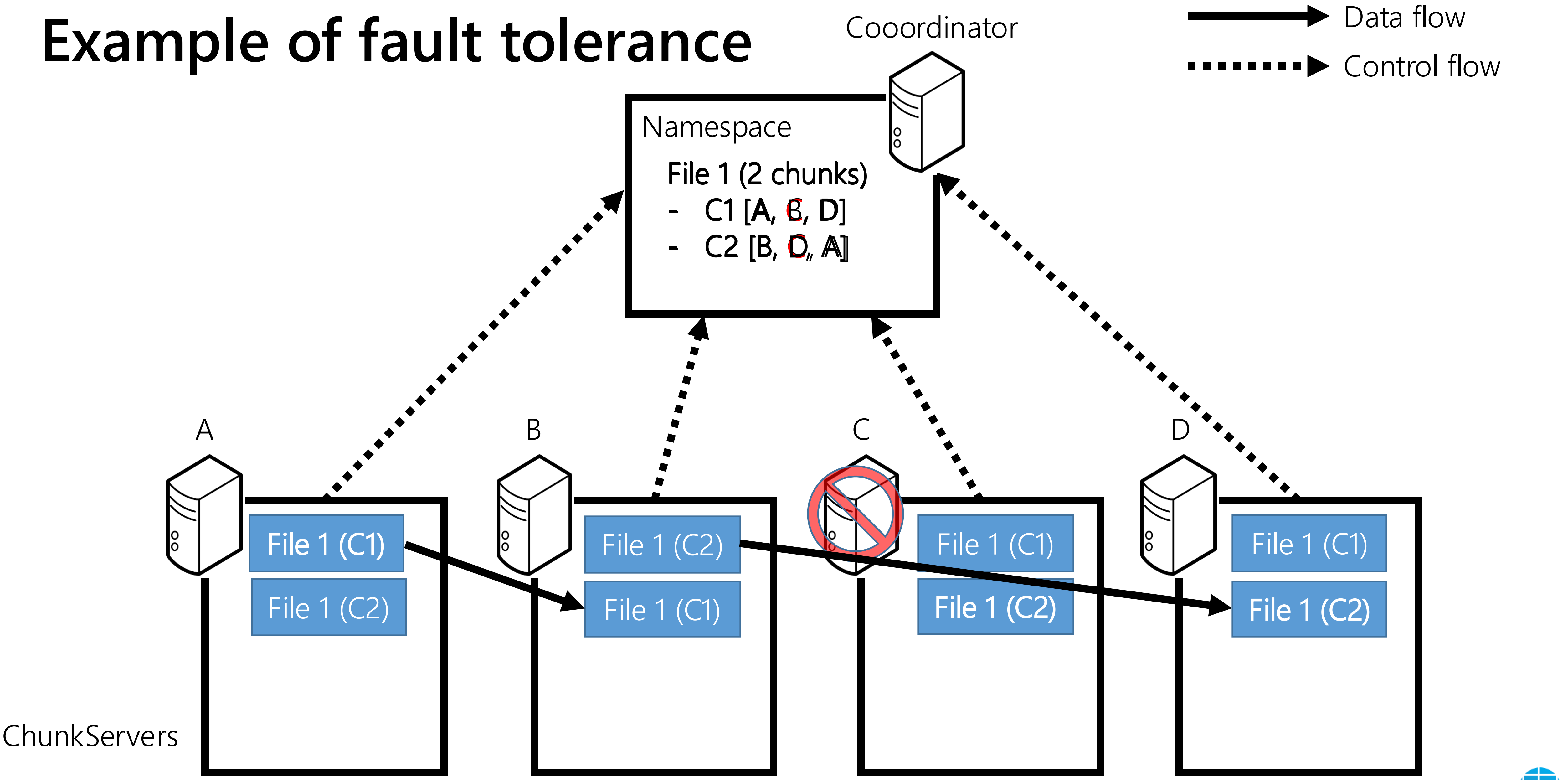


#chunks/node ratio

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

Cash miss

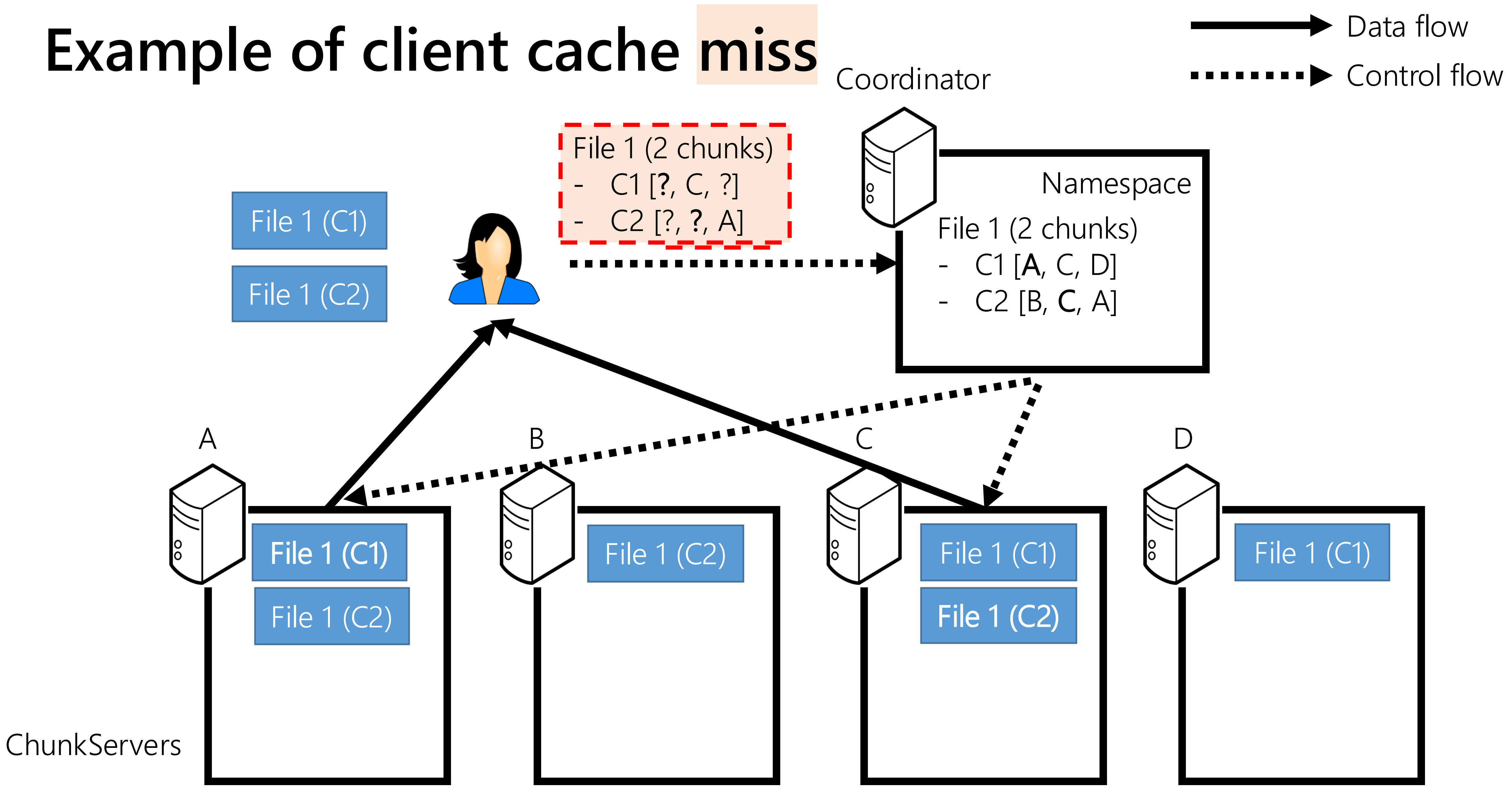
1. The client sends a READ command to the coordinator
2. The coordinator requests chunkservers to send the chunks to the client
   - Ranked according to the closeness in the network
3. The list of locations is cached in the **client**
   - Not a complete view of all chunks

Cash hit

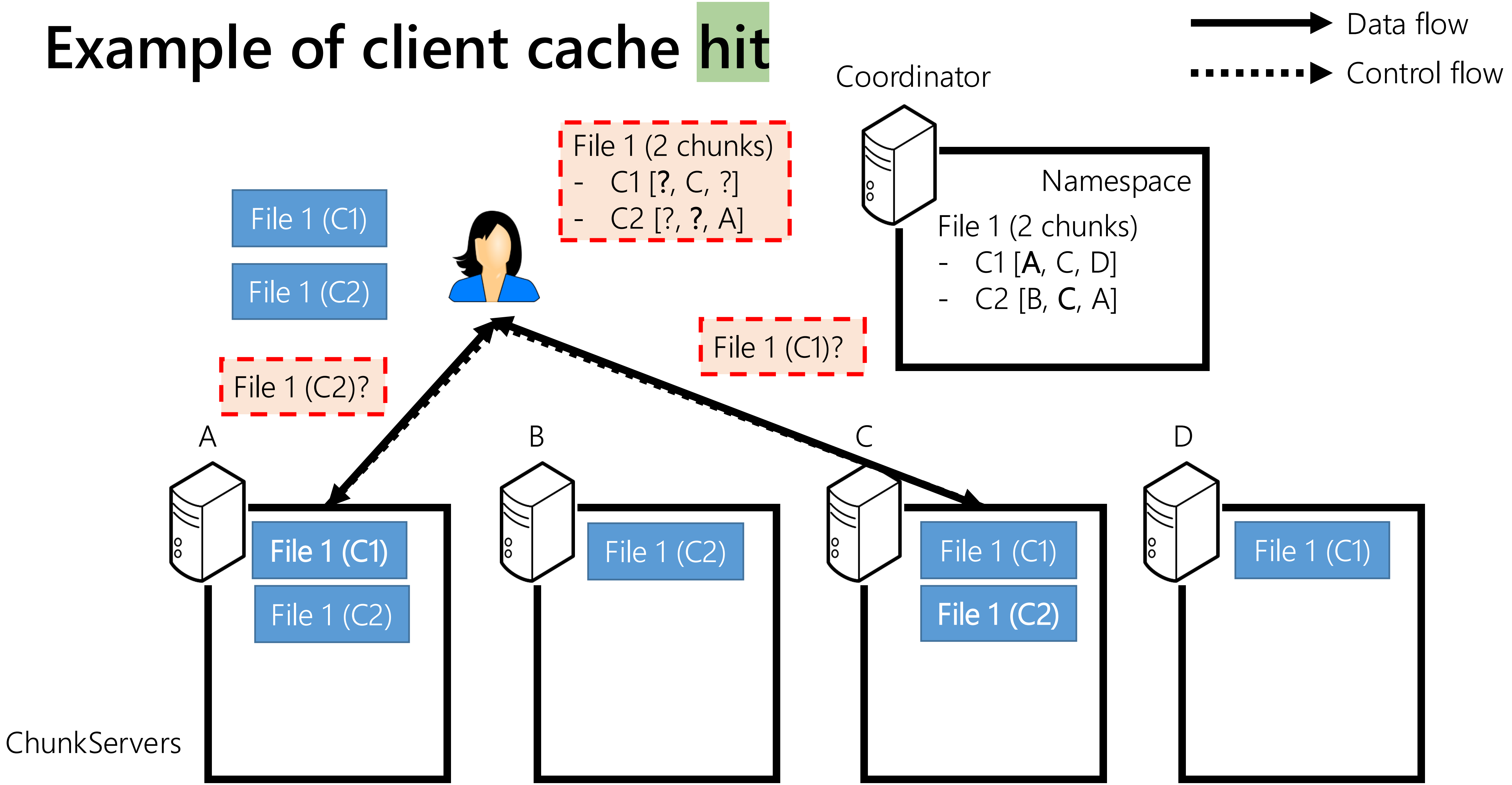
1. The client reads the **cache (locally stored)** and requests the chunkservers to send the chunks

Avoid coordinator bottleneck
+
One communication step is saved

Example of client cache miss



Example of client cache hit

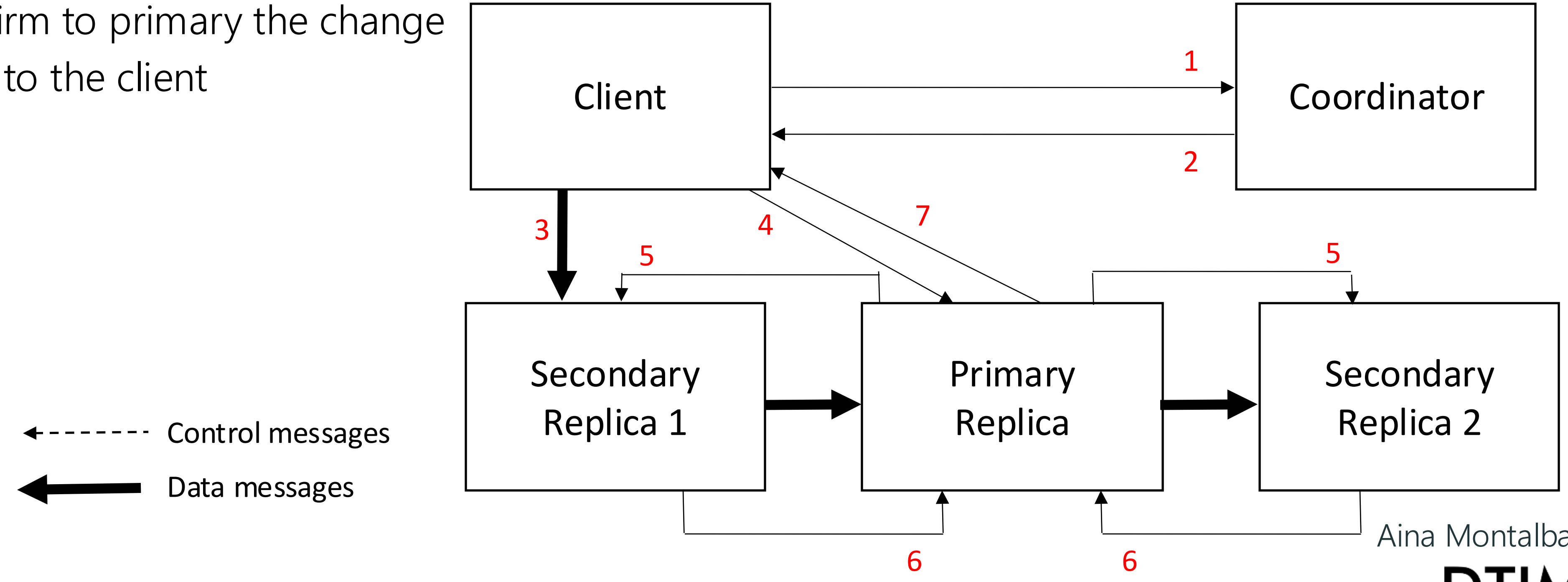


(Distributed) Transaction Management

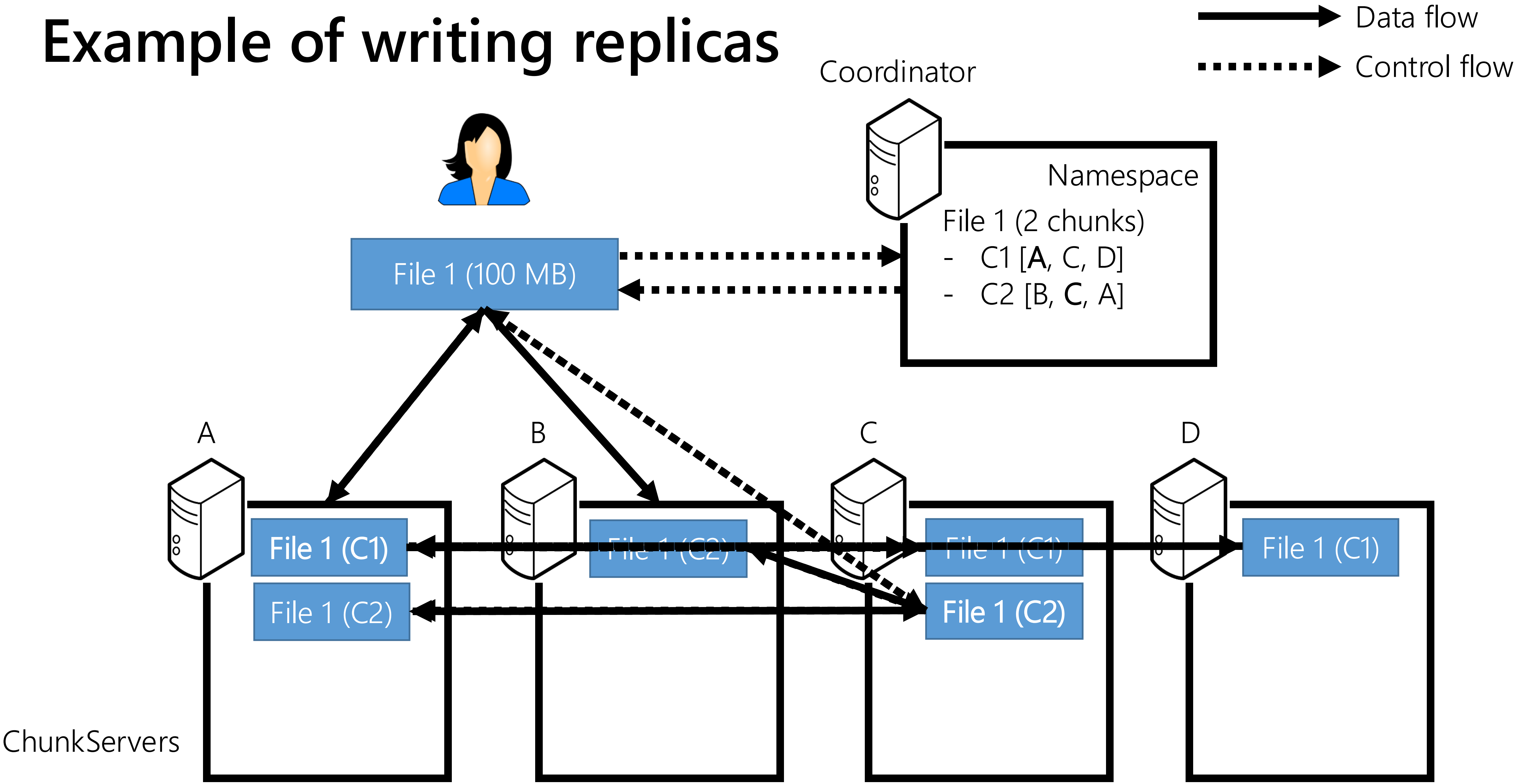
Challenge III

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



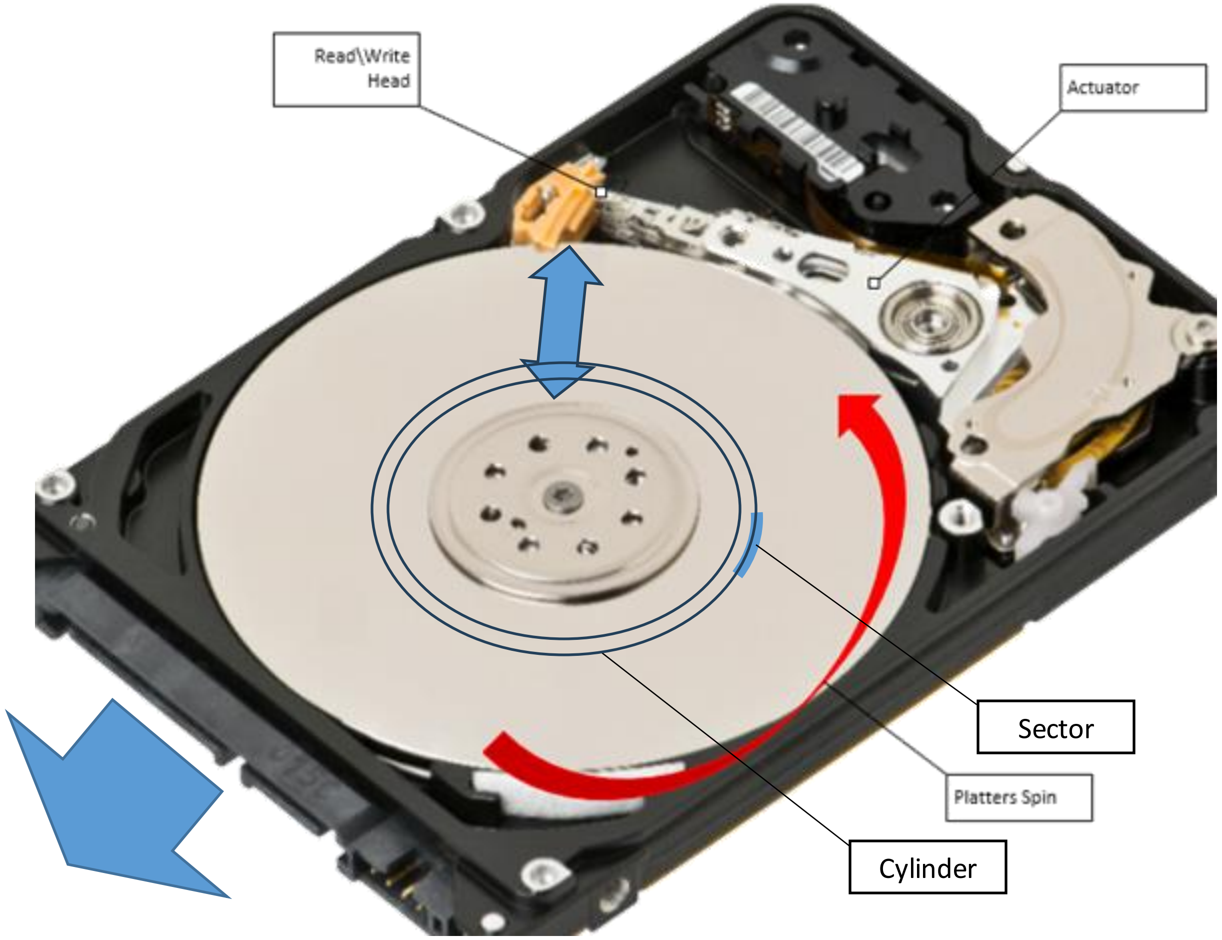
Example of writing replicas



(Distributed) Query processing

Challenge IV

HDDs costs



<u>Rotational Latency</u>

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

<u>Seek Time</u>

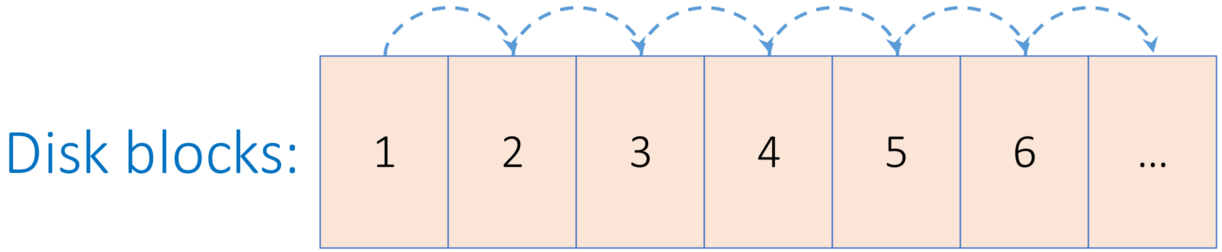
Time taken for the ReadWrite head (mechanical arm) to move between cylinders on the disk

<u>Transfer Time</u>

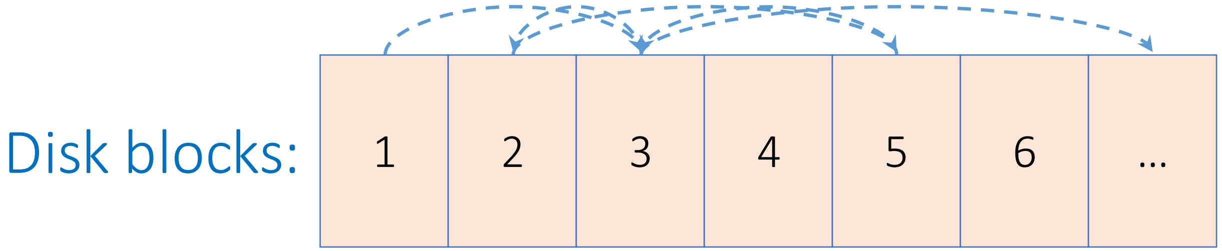
Time taken for requests to get from the system to the disk (depends on the block size, e.g., 8KB)

Sequential vs. Random access

Sequential Access



Random Access



Cost of accessing data (approximations)

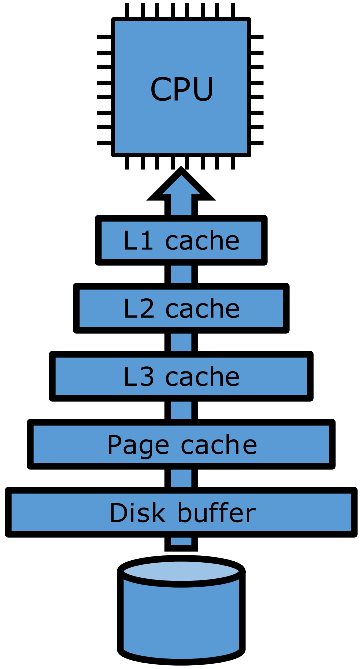
- Sequential reads
  - Option to maximize the effective read ratio
    - Depends on DB design
  - Enables pre-fetching

  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

(Distributed) Data Design

Challenge V

Storing CSV files in HDFS

```
ID,Name,Age,City,Salary
1,John Doe,28,New York,55000
2,Jane Smith,34,Los Angeles,62000
3,Michael Johnson,40,Chicago,75000
4,Emily Davis,26,Houston,48000
5,Wil Wilson,31,San Francisco,72000
6,Sophia Martinez,29,Miami,51000
7,James Brown,38,Seattle,68000
8,Olivia Taylor,27,Denver,53000
9,William Anderson,45,Boston,82000
10,Ava Thomas,33,Atlanta,60000
11,Benjamin White,36,Dallas,67000
12,Amy Harris,30,Philadelphia,59000
13,Mason Martin,32,Phoenix,64000
14,Emma Thompson,25,San Diego,47000
15,Liam Robinson,41,Las Vegas,77000
```

```
ID,Name,Age,City,Salary
1,John Doe,28,New York,55000
2,Jane Smith,34,Los Angeles,62000
3,Michael Johnson,40,Chicago,75000
4,Emily Davis,26,Houston,48000
```

```
5,Wil Wilson,31,San Francisco,72000
6,Sophia Martinez,29,Miami,51000
7,James Brown,38,Seattle,68000
8,Olivia Taylor,27,Denver,53000
9,William Anderson,45,Boston,82000
```

```
10,Ava Thomas,33,Atlanta,60000
11,Benjamin White,36,Dallas,67000
12,Amy Harris,30,Philadelphia,59000
13,Mason Martin,32,Phoenix,64000
14,Emma Thompson,25,San Diego,47000
```

```
15,Liam Robinson,41,Las Vegas,77000
```

CSV is a row-based text format (horizontal)

- Easy to use, but inefficient

- **No built-in schema**
  - May have inconsistent structures
  - Extra processing required (e.g., parsing)
- **No built-in compression**
  - Each field stored as plain text, resulting in larger file sizes
- **No metadata**
  - Queries that need only a few columns must still scan the entire block

Storage layouts

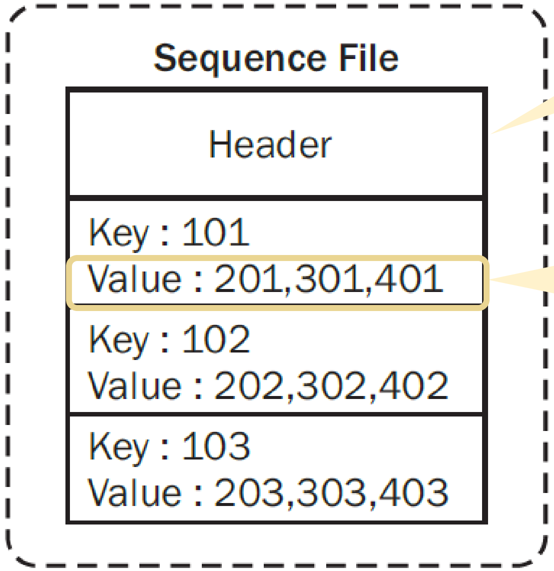
- Different workloads require different layouts
  - Horizontal (e.g., SequenceFile, Avro)
    - For scan-based workloads
  - Vertical (e.g., Zebra)
    - For projection-based workloads (reads a subset of columns)
  - Hybrid (e.g., Parquet)
    - For projection- and predicate-based workloads (reads a subset of columns or rows)

Horizontal layout – Sequence File

- Records of <u>binary key-value</u> pairs

Table 1

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



Stores **metadata** like: Key and Value Class type, Compression algorithm

Everything stored together as a single value

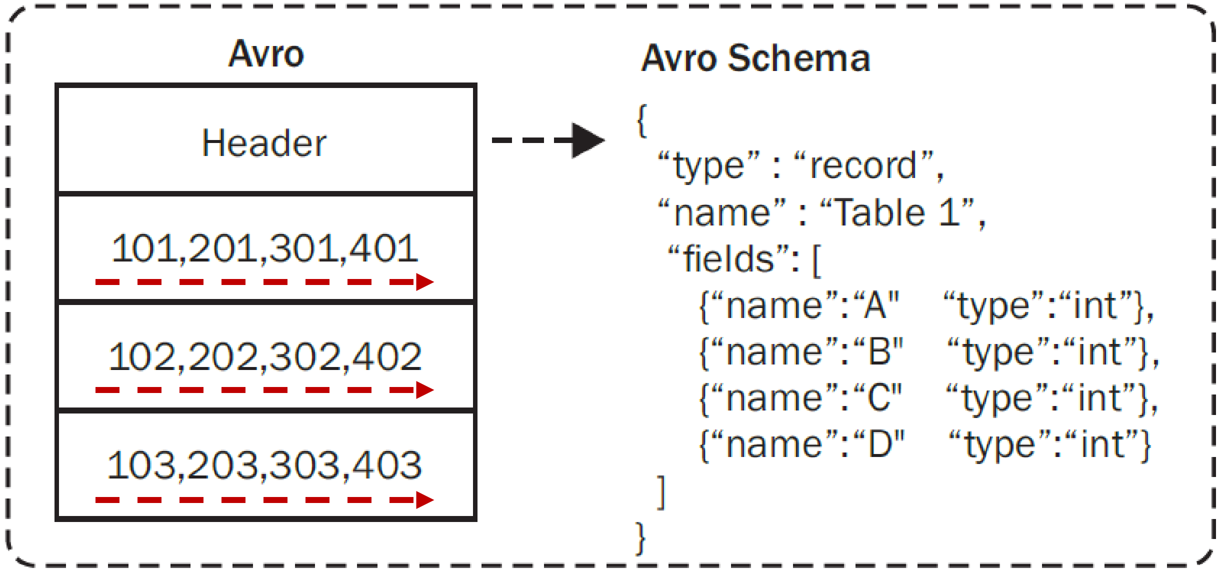
- Compression
  - Uncompressed
  - Record-compressed
  - Block-compressed
    - "block" is the compression unit – a block of records (not a chunk)
      - 1 MB default

Horizontal layout - Avro

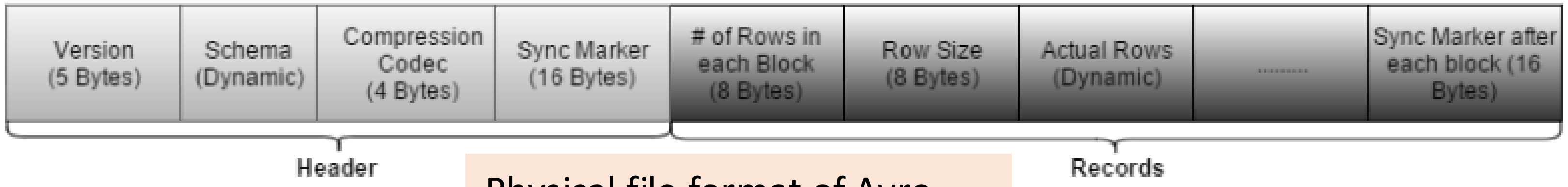
- Binary encoding of (compressed) rows
- The header contains a schema encoded in JSON

Table 1

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



There is also meta-information for each row



Physical file format of Avro

Horizontal layout - Avro

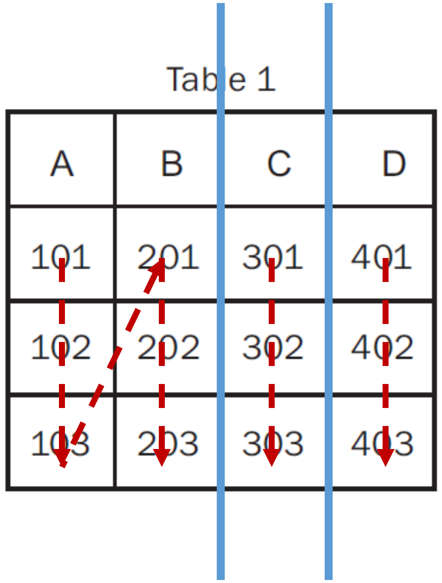
```
{"namespace": "example.avro",
 "type": "record",
 "name": "User",
 "fields": [
     {"name": "name", "type": "string"},
     {"name": "favorite_number",  "type": ["int", "null"]},
     {"name": "favorite_color", "type": ["string", "null"]}
 ]
}
```

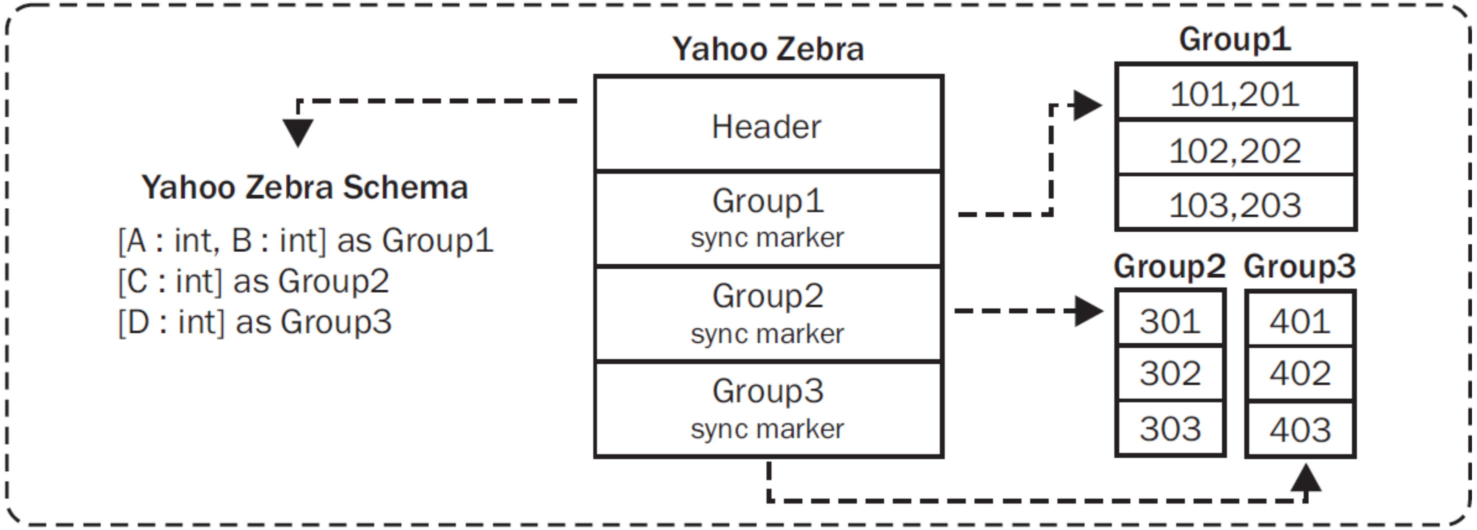
```python
import avro.schema
from avro.datafile import DataFileReader, DataFileWriter
from avro.io import DatumReader, DatumWriter

schema = avro.schema.parse(open("user.avsc", "rb").read())

writer = DataFileWriter(open("users.avro", "wb"), DatumWriter(), schema)
writer.append({"name": "Alyssa", "favorite_number": 256})
writer.append({"name": "Ben", "favorite_number": 7, "favorite_color": "red"})
writer.close()

reader = DataFileReader(open("users.avro", "rb"), DatumReader())
for user in reader:
    print user
reader.close()
```

```
{u'favorite_color': None, u'favorite_number': 256, u'name': u'Alyssa'}
{u'favorite_color': u'red', u'favorite_number': 7, u'name': u'Ben'}
```

Vertical layout - Zebra

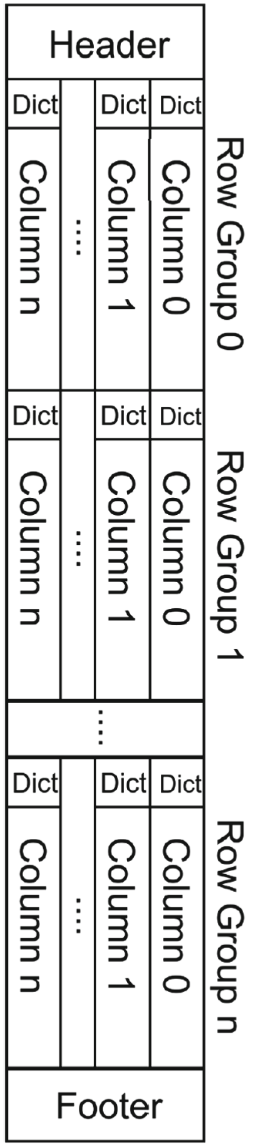
- The header contains the definition of groups
  - Each group contains a set of columns
  - Widely benefits from compression
- Not really used in practice

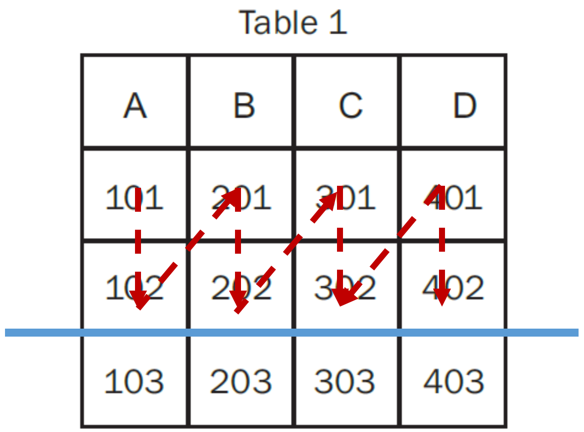


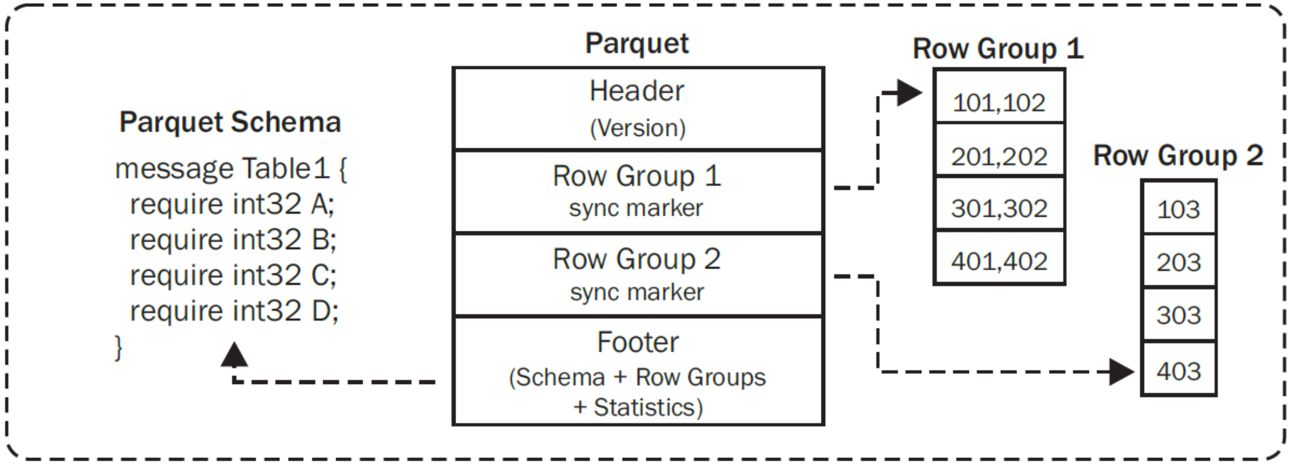


Hybrid layout - Parquet

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







Comparison of data formats

Helps during the data serialization and deserialization phases by **avoiding the need to cast** the data at the app. level – which is a costly operation.

	l		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

APACHE hadoop

Comparison of data formats

Read only the required columns (e.g., `SELECT a1, a2 from Cities`) and avoid performing unnecessary reads.

	tal		Vertical	Hybrid	
	Sequence Files	Avro	Yahoo Zebra	ORC	Parquet
Sch	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

Comparison of data formats

Push down the selection predicates (e.g., `where city = “Barcelona”`) into the **storage layer**, because they store indexing information that helps in filtering the records while reading. Avoid unnecessary reads.

			Vertical	Hybrid	
	[illegible]s	Avro	Yahoo Zebra	ORC	Parquet
[illegible]	[illegible]	Yes	Yes	Yes	Yes
Col[illegible]uning	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

Comparison of data formats

Index info like ColumnIndex are stored as metadata. This allows **navigation to the pages of a column**, based on column values and is used to locate data pages that contain matching values for a scan predicate

[illegible]	[illegible]		Vertical	Hybrid	
[illegible]	[illegible]es	Avro	Yahoo Zebra	ORC	Parquet
[illegible]	[illegible]	Yes	Yes	Yes	Yes
[illegible]	[illegible]	No	Yes	Yes	Yes
Pre[illegible] 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

Comparison of data formats

Statistical information for each column are pre-computed and allow to determine whether a page can be skipped, and also enables easier computation of aggregates.

	al		Vertical	Hybrid	
	Files	Avro	Yahoo Zebra	ORC	Parquet
		Yes	Yes	Yes	Yes
		No	Yes	Yes	Yes
		No	No	Yes	Yes
Ind Information	No	No	No	Yes	Yes
Statistics Information	No	No	No	Yes	Yes
Nested Records	No	No	Yes	Yes	Yes

Comparison of data formats

Features	Horizontal		Vertical	Hybrid	
	Sequence Files	Avro	Yahoo Zebra	ORC	Parquet
		Yes	Yes	Yes	Yes
		No	Yes	Yes	Yes
		No	No	Yes	Yes
		No	No	Yes	Yes
St ics Information	No	No	No	Yes	Yes
Nested Records	No	No	Yes	Yes	Yes

Allows to store bag, maps and custom user data types.

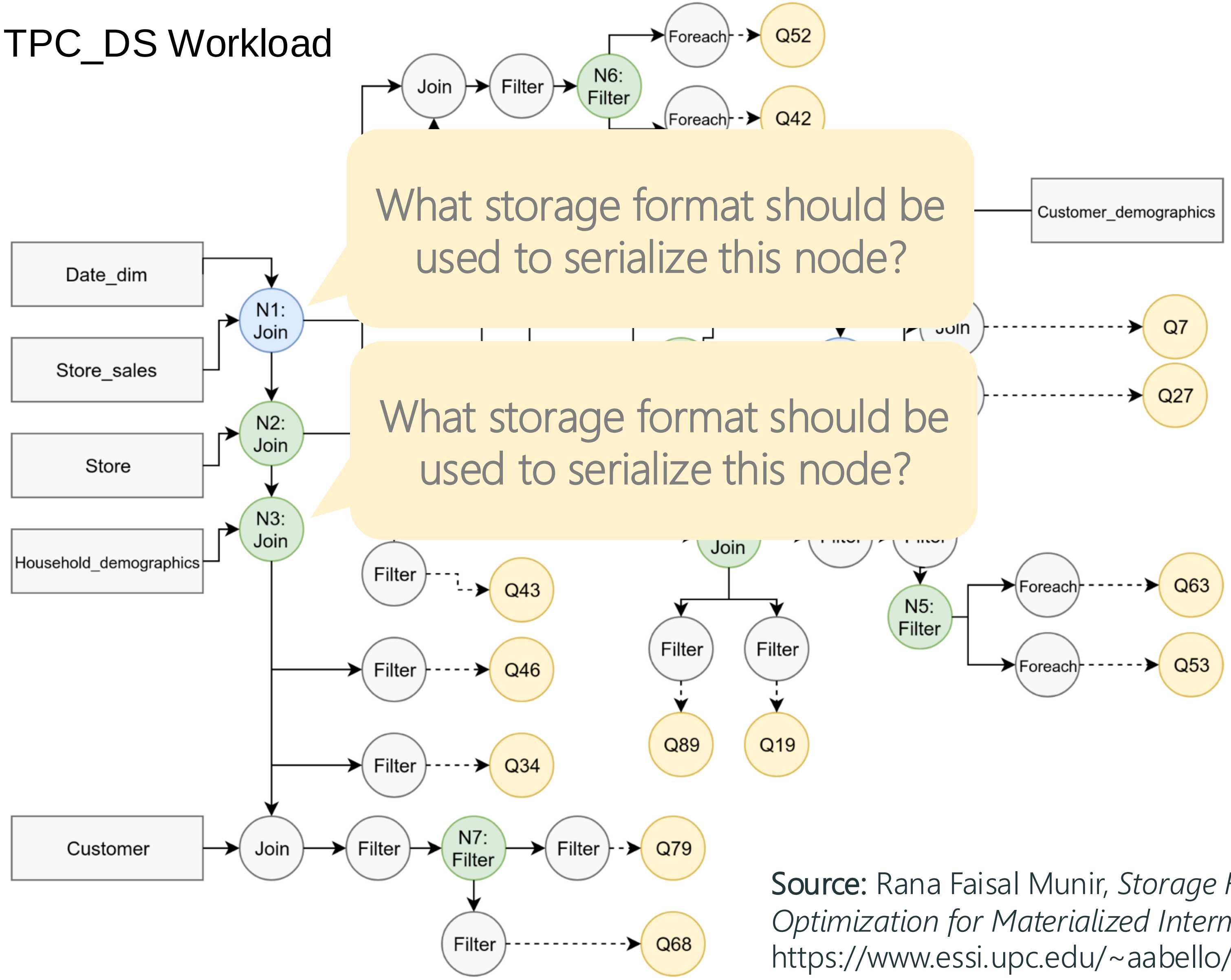
In summary, each format provides a different set of features that will affect the overall performance when retrieving the intermediate results from the disk. Generally, **hybrid layouts perform well if a subset of data is read.** Alternatively, **horizontal layouts perform well if all, or most of the data is read.**

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

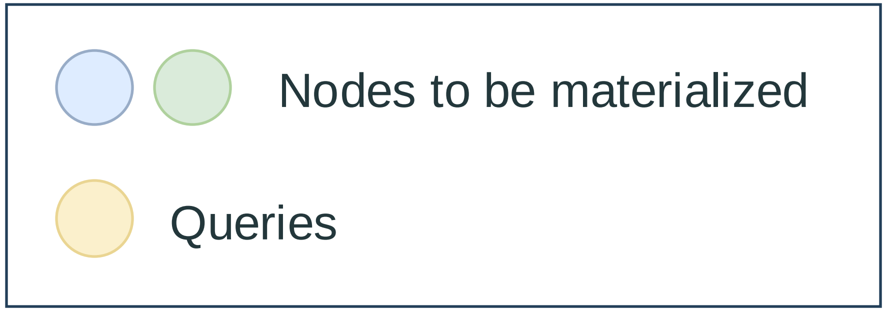
Vertical layouts are subsumed by Hybrid layouts, as Hybrid layouts support all their features.

How to choose the right storage format?

TPC_DS Workload



Given a flow represented as DAG(V, E)



Source: Rana Faisal Munir, *Storage Format Selection and Optimization for Materialized Intermediate Results in DIFs*
https://www.essi.upc.edu/~aabello/publications/19.thesis_Rana.pdf

Rule-based choice (heuristic)

Given a flow repr

Only for key value pairs, because SF stores data as kv pairs. Otherwise, several columns would need to be combined (e.g., with a separator marker "-") and parsed at the application level.

- 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}

1. When performing aggregations on data (statistical information helps)
2. When subset of data is read

1. When the operator affects all the columns (not subset)
2. When all the data is read without filtering

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)

Cost model in relational databases:
Enumerate the set of possible plans for a query. Generate an estimate of the cost of executing a particular query plan for the current state of the database. Given the estimated cost decide the algorithms to use for a given query.

Statistics are required to estimate the cost of executing a particular query plan (e.g., cardinality of data, the cost of scanning a row, the cost of sorting the data …).

Parameters of the cost model

We can assume the existence of the variables shown in the table.

They may be **given** (e.g., system constants) or **calculated** once the data and workloads are available (e.g., data statistics, workload statistics, layout vars).

Source: Rana Faisal Munir, *Storage Format Selection and Optimization for Materialized Intermediate Results in DIFs*
https://www.essi.upc.edu/~aabello/publications/19.thesis_Rana.pdf

Variable	Description
	System Constants
R	Replication factor
p	Probability of accessed replica being local
$Chunk_{Size}$	Block size in the DFS
BW_{Disk}	Disk bandwidth
BW_{Net}	Network bandwidth
$Time_{Seek}$	Disk seek time
$Time_{Disk}$	$\frac{Chunk_{Size}}{BW_{Disk}}$
$Time_{Net}$	$\frac{Chunk_{Size}}{BW_{Net}}$
	Data Statistics
$\lvert IR \rvert$	Number of Rows in IR
Row_{Size}	Average Row Size of IR
$ColValue_{Size}$	Average Column Size[1] of IR
$\#Cols$	Columns of IR
	Workload Statistics
Ref_{Cols}	Number of columns used in an operation
SF	Selectivity factor of an operation
	Layout Variables
RG_{Size}	Row group size of hybrid layouts
$Meta_{Size_{Layout}}$	Metadata size for a given layout
$Body_{Size_{Layout}}$	Size of the body of a layout
$Header_{Size_{Layout}}$	Size of the header of a layout
$Footer_{Size_{Layout}}$	Size of the footer of a layout
$Used_{Chunks_{Layout}}$	Number of chunks of a layout
$Used_{RowGroups_{Layout}}$	Number of row group of hybrid layouts
$\lvert RG \rvert$	Number of rows of a row group
$Total_{Seeks_{Layout}}$	Total number of seeks for a given layout

[1] Extra 4 bytes are considered for variable length columns

Table 4.2: Parameters of the Cost Model

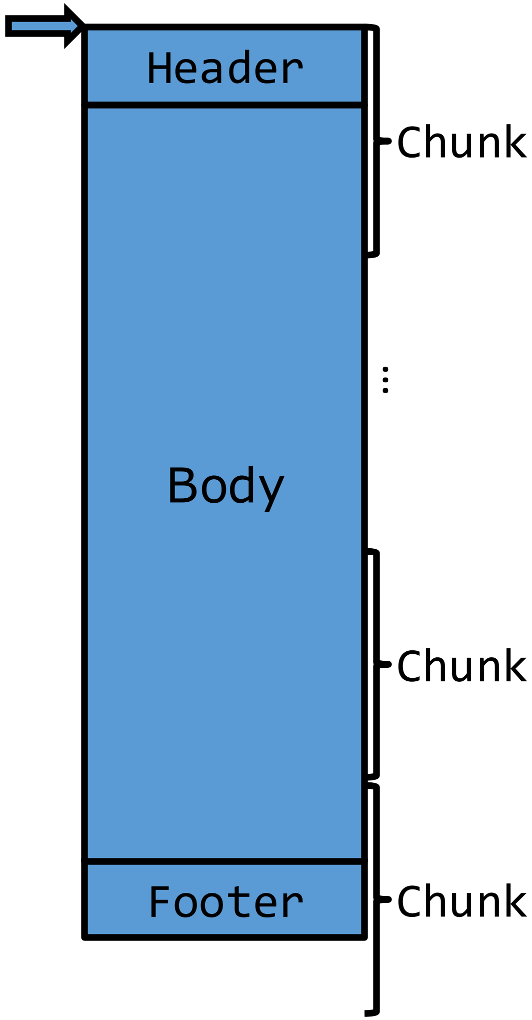
UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

General formulas for size estimation

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

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

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



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

Horizontal layout size estimation

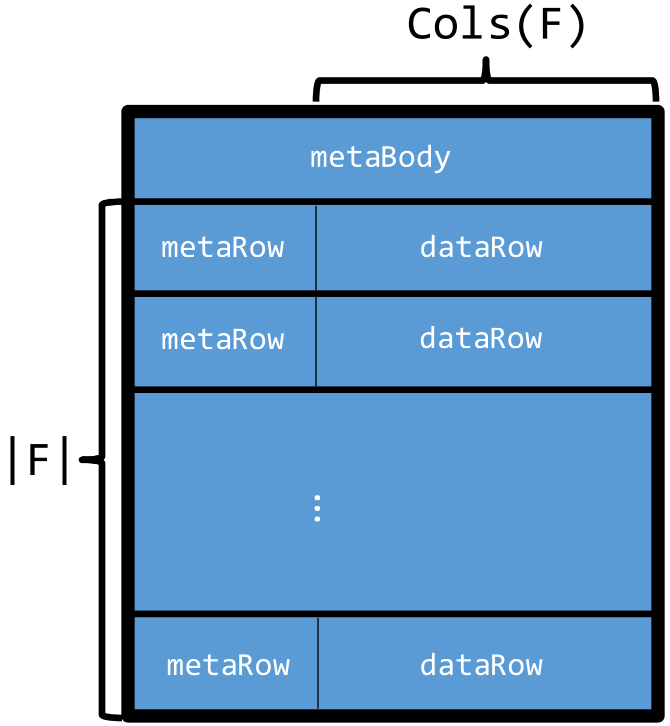
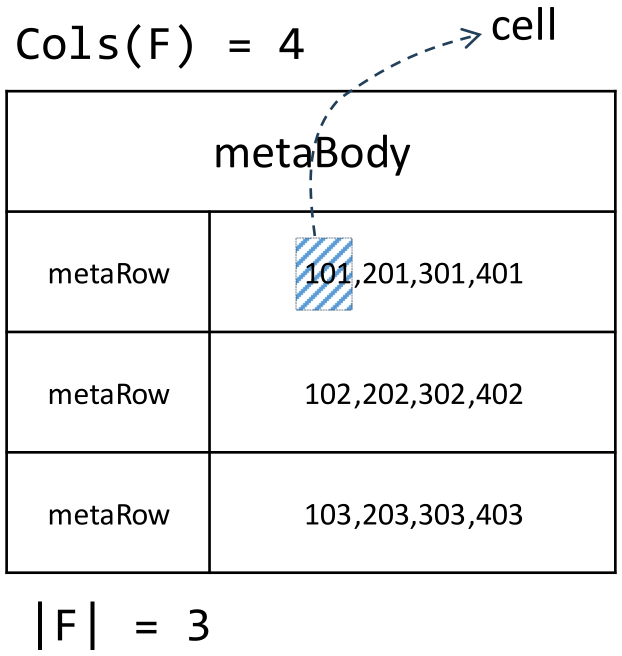


Table 1

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



What is the size of the body in Horizontal layouts?

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

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

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

“F” is the content of the file

|F| is the cardinality (number of rows)

www.essi.upc.edu/dtim

Vertical layout size estimation

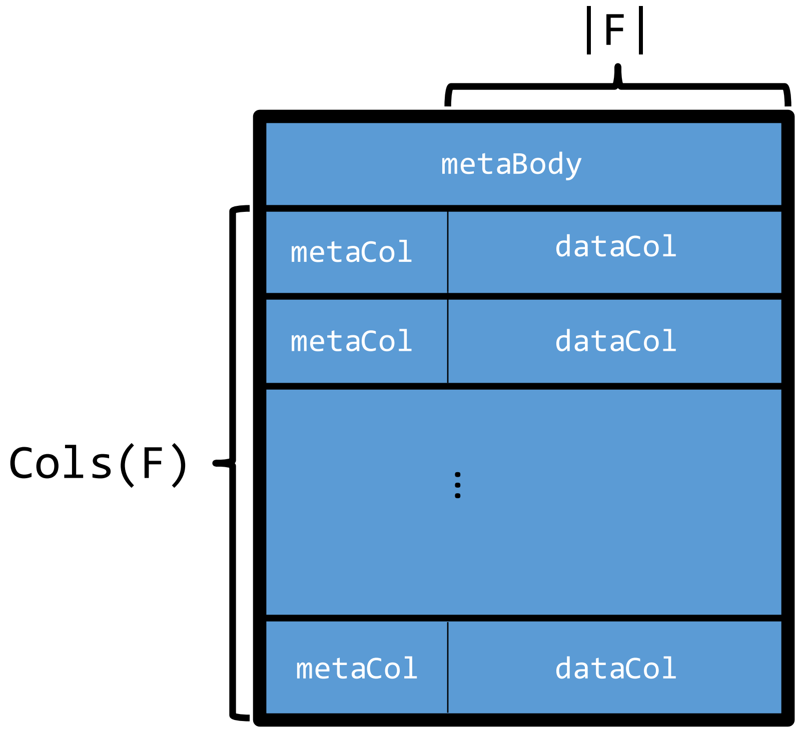
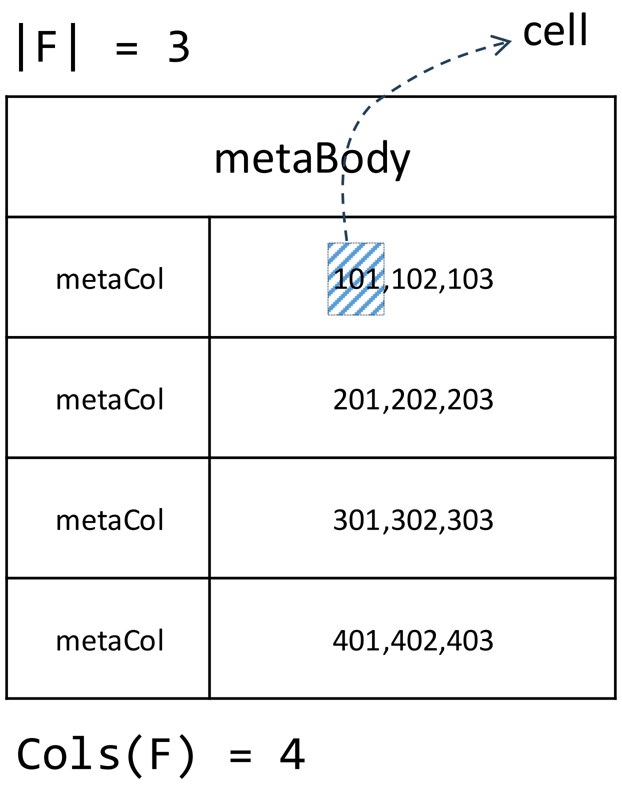


Table 1

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



What is the size of the row and body in Vertical layouts?

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

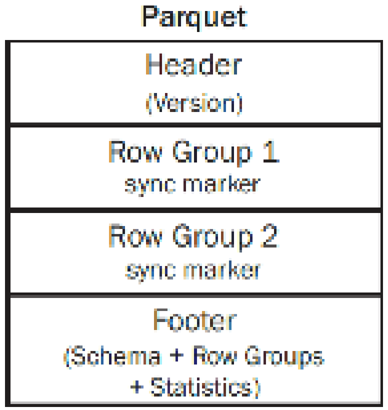
$$Size(Body_{Vertical}) = Size(metaBody) + Cols(F) * (Size(metaCol) + Size(dataCol))$$

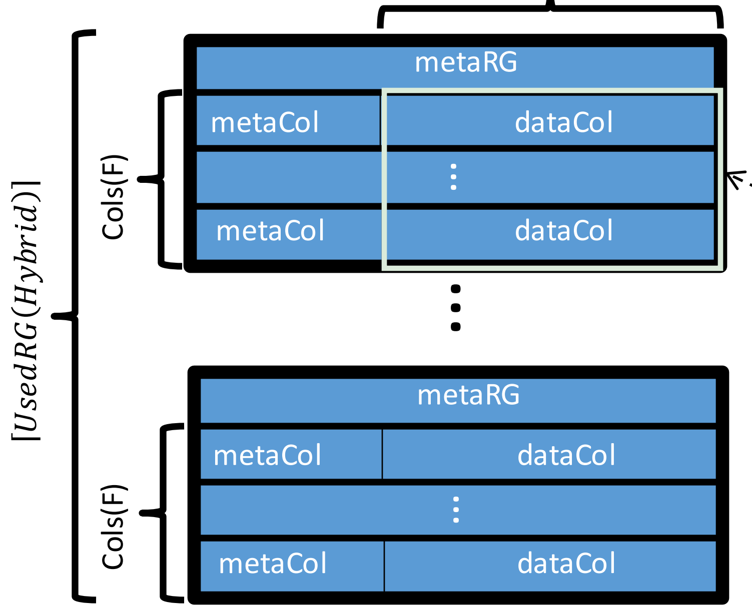
"F" is the content of the file

|F| is the cardinality (number of rows)

Hybrid layout size estimation

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





What is the size of the body in Vertical layouts?

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

The size of all the raw data (without metadata)

The size of an RG without metadata

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

Multiply each RG with the metadata it stores

Plus the size of the raw data

General formulas for cost estimation

What is the cost of **writing**? Since it is distributed, number of chunks plus the seek cost to locate the positon.

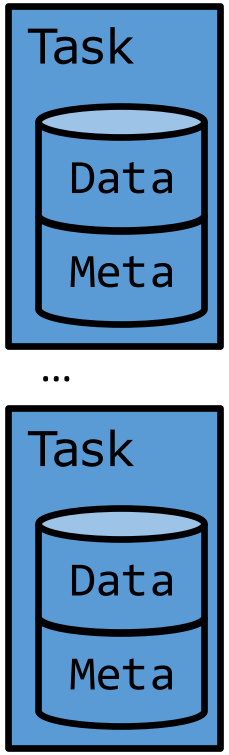
$$
\begin{aligned}
Cost(Write_{Layout}) = & \ \mathrm{UsedChunks(Layout)}*\mathrm{W}_{\mathrm{WriteTransfer}} \\
& +\mathrm{Seeks(Layout)}*(1\text{-}\mathrm{W}_{\mathrm{WriteTransfer}})
\end{aligned}
$$

What is the cost of **reading** (full scan)? Thers is an extra cost: the metadata is trasnfered to each task.

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

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

$$
\begin{aligned}
Cost(Scan_{Layout}) = & \ \mathrm{UsedChunks(Scan_{Layout})}*\mathrm{W}_{\mathrm{ReadTransfer}} \\
& +\mathrm{Seeks(Layout)}*(1\text{-}\mathrm{W}_{\mathrm{ReadTransfer}})
\end{aligned}
$$



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

Cost of projection in hybrid layouts

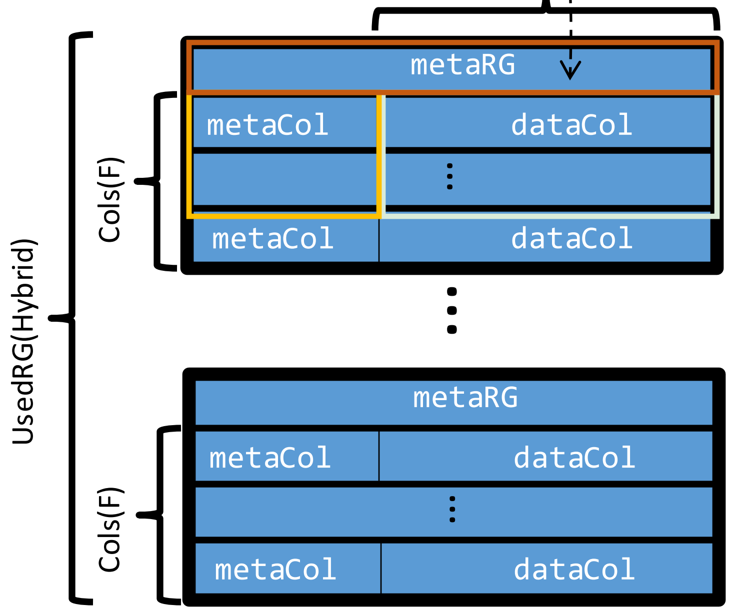
$$Size(projCols) = Proj(F) * UsedRows(RG) * Size(cell)$$

If UsedRows=dataCol
and Proj(F) = 2

$$Size(Project_{Hybrid}) = Size(Header_{Hybrid}) + Size(Footer_{Hybrid})$$
$$+[UsedRG(Hybrid)] * (Size(metaRG) + proj(F) * Size(metaCol))$$
$$+[UsedRG(Hybrid)] * Size(projCols)$$

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

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



What about selection?

We have to access the entire RG, even if only some data are actually requested by the user (e.g., there is a filter).

So, we need to compute how many RGs we need to access based on the rows that are defined by the predicate!

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

Probability of retrieving a RowGroup

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

Retrieve all RGs except the ones where none of their rows fulfill P (if none of the rows in the RG fulfills P, we don't read that RG)

Selection in hybrid layouts

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

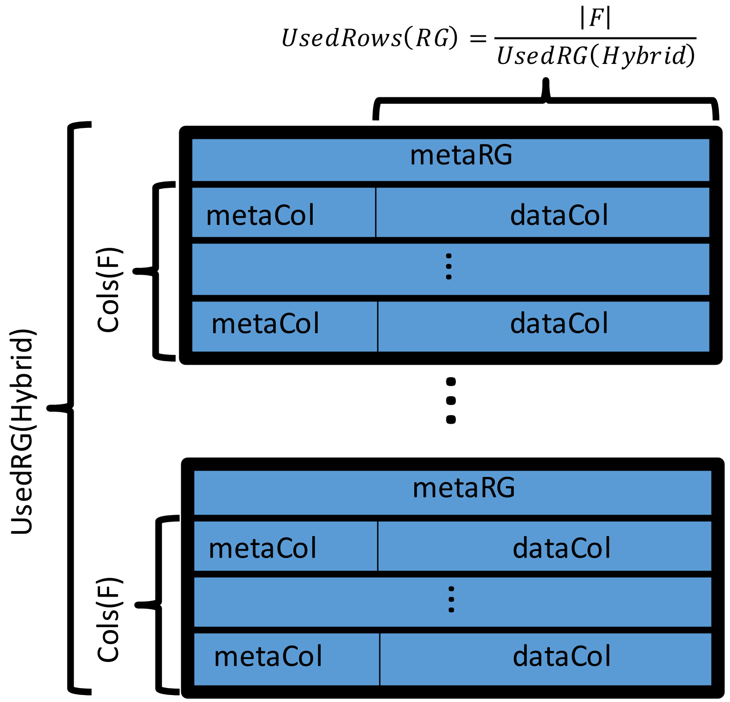
$$Size(RowsSelected) = \left\lceil \frac{SF * |F|}{UsedRows(RG)} \right\rceil * \left(Size(metaRG) + Cols(F) * Size(metaCol)\right)$$ The size of the metadata depending on SF

$$+SF * |F| * Cols(F) * Size(cell)$$ The size of the raw data multiplied by SF

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

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

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



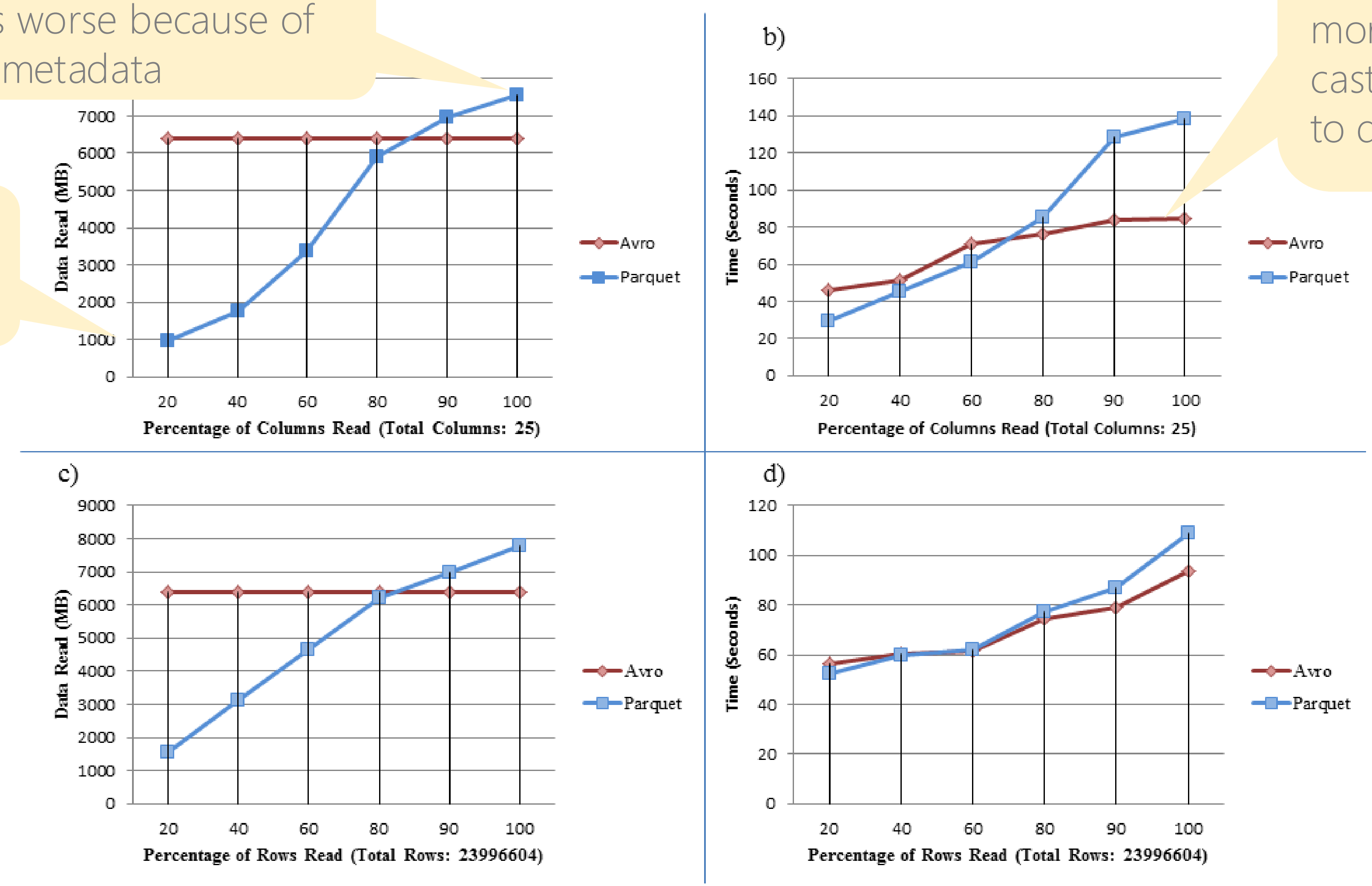
“F” is the content of the file

Comparison of selection and projection

If you read all the columns, Parquet is worse because of the extra metadata

If you read few columns, Parquet is better

It is not flat, because we have a schema and the more columns, the more casting you need to do to de-serialize the data



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)