[544] Hadoop Ecosystem

Tyler Caraza-Harter

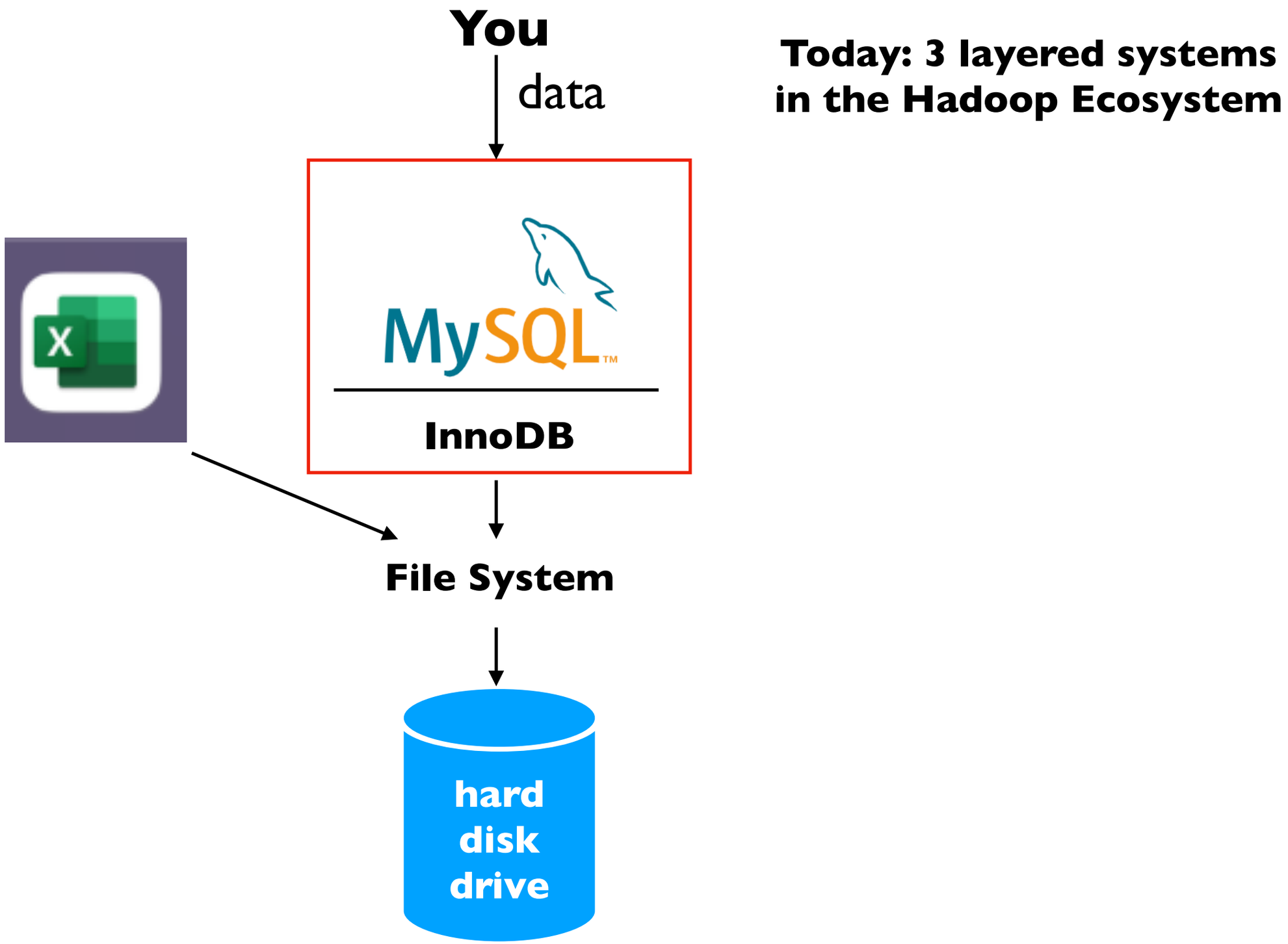
Outline: Hadoop Ecosystem

Architecture: Layered Data Systems

Hadoop File System

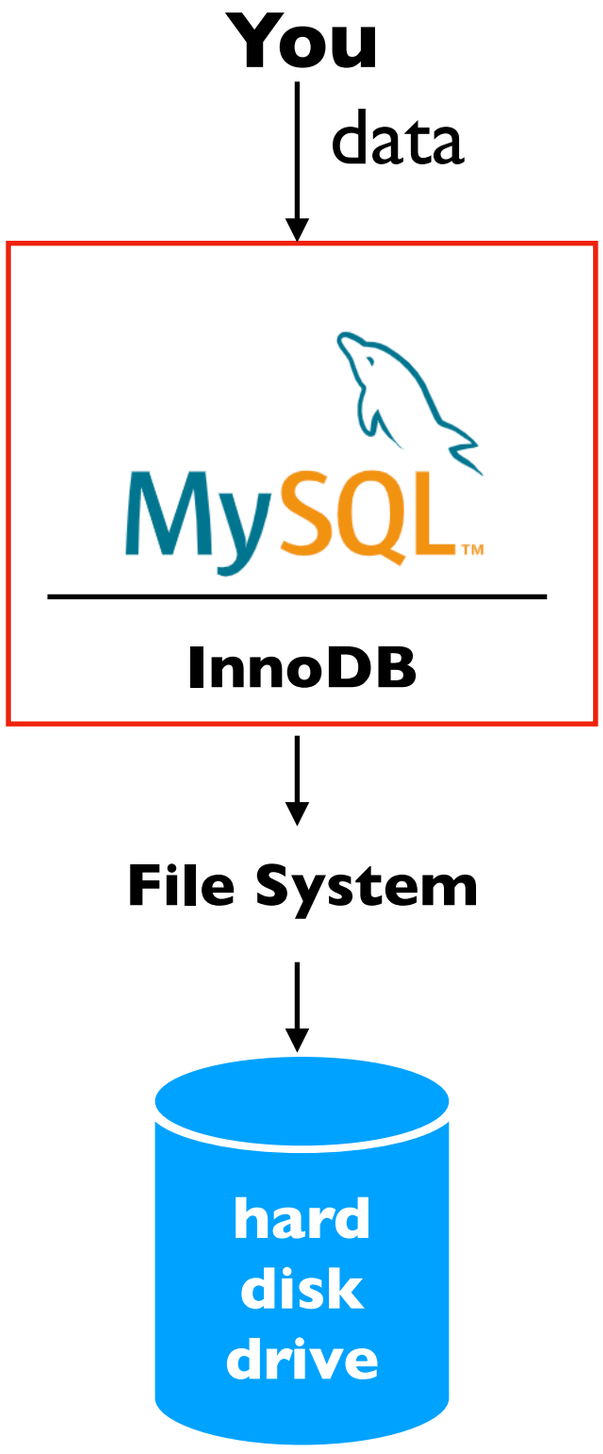
Hadoop MapReduce

Design: storage systems are generally built as a composition of layered subsystems

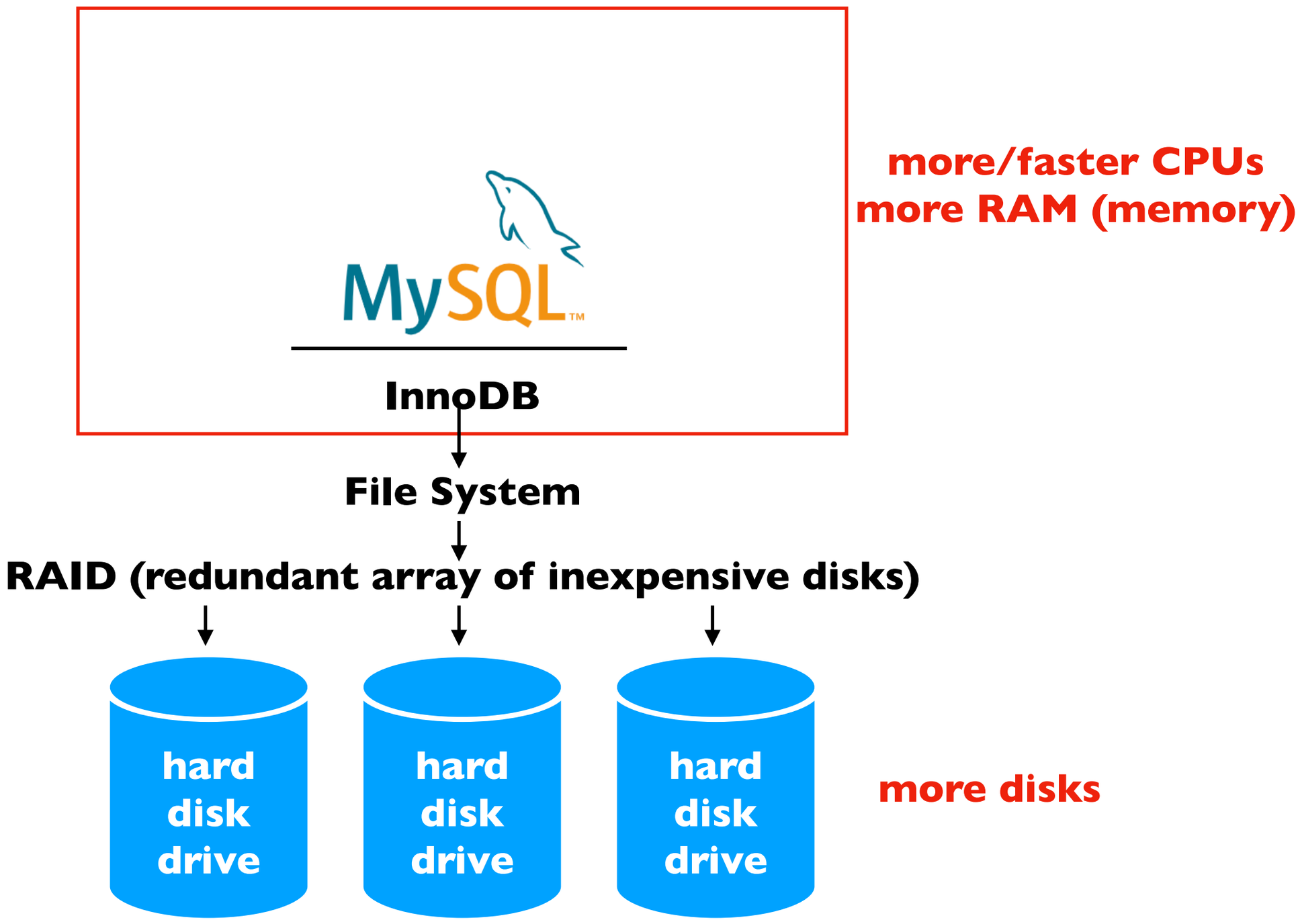


Today: 3 layered systems in the Hadoop Ecosystem

What if your data is too big for your server?

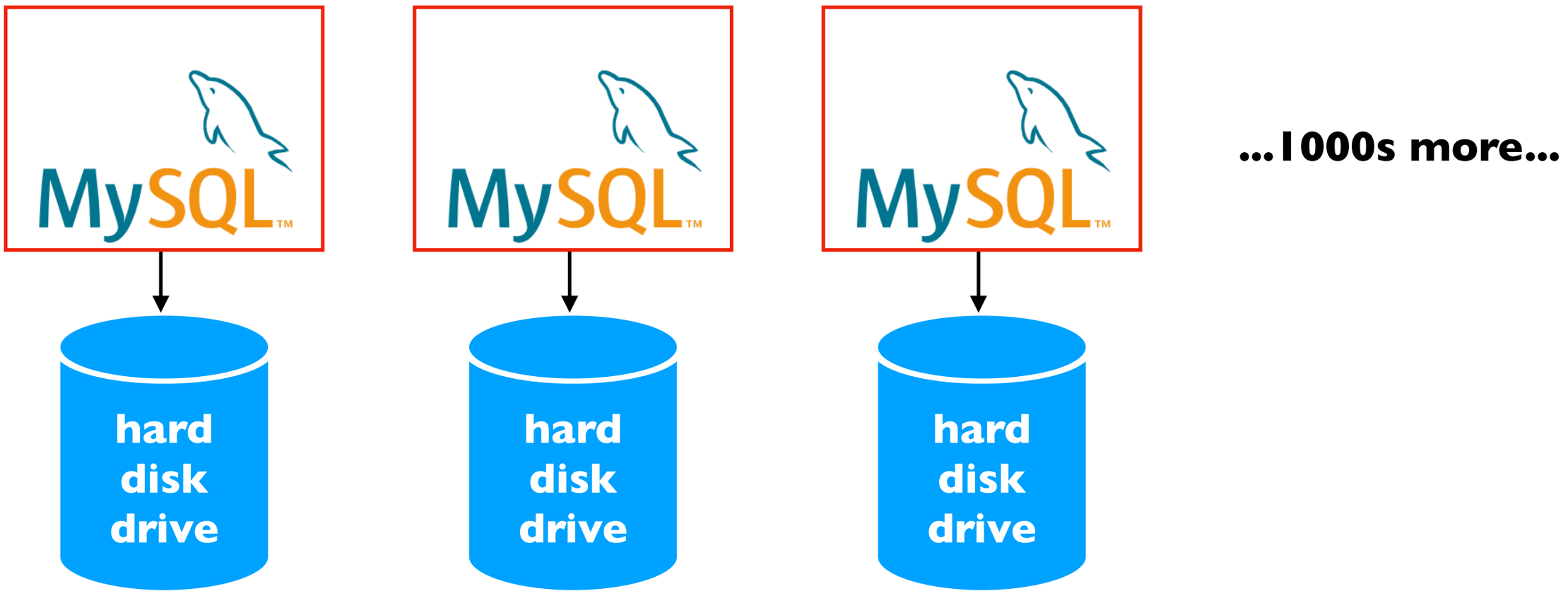


What if your data is too big for your server?
Option 1: scale up (buy better hardware)



What if your data is too big for your server?
Option 2: scale out (more machines)

where does the data actually go?



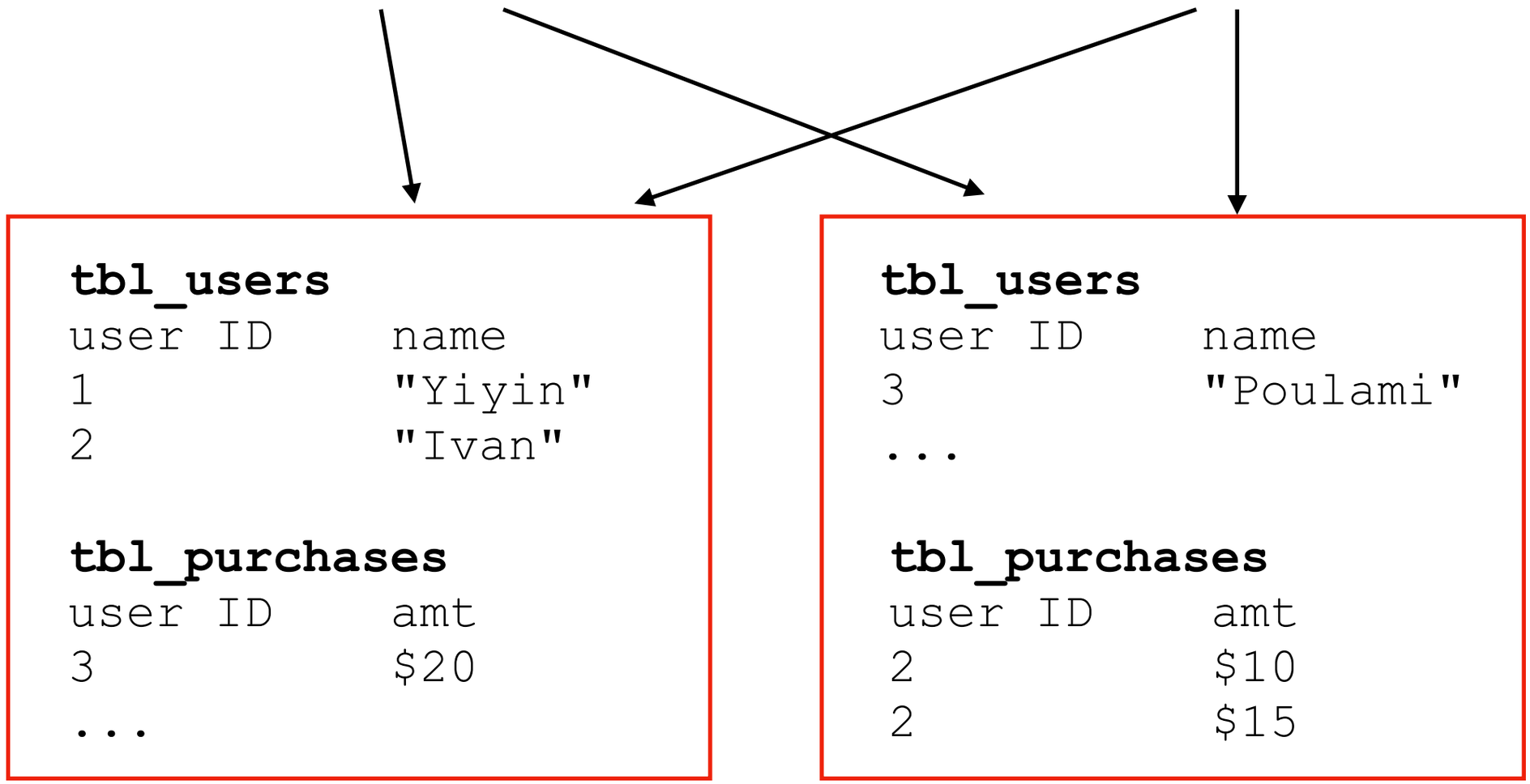
Approach: partition the tables

tbl_users

user ID	name
1	"Yiyin"
2	"Ivan"
3	"Poulami"
...	

tbl_purchases

user ID	amt
2	$10
2	$15
3	$20
...	



tbl_users

user ID	name
1	"Yiyin"
2	"Ivan"

tbl_purchases

user ID	amt
3	$20
...	

tbl_users

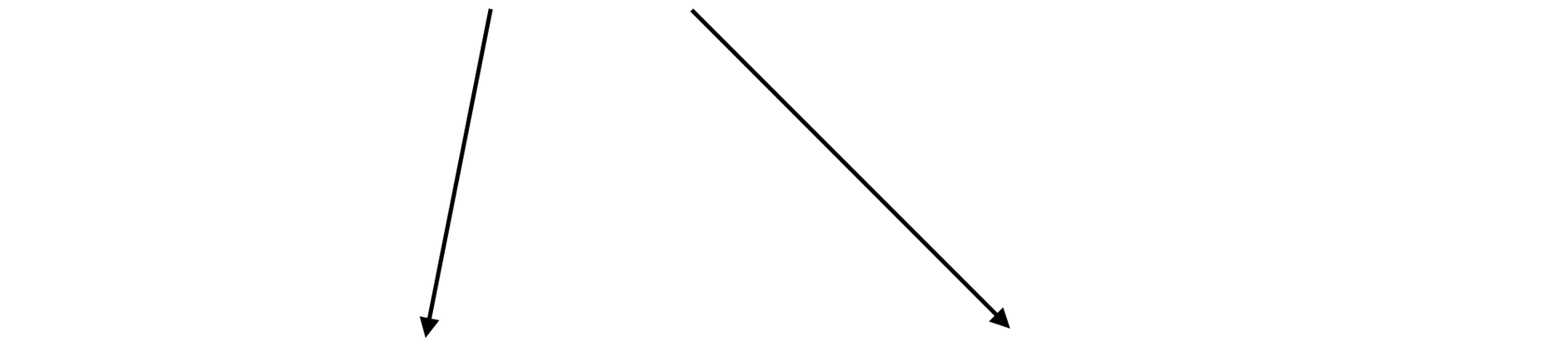
user ID	name
3	"Poulami"
...	

tbl_purchases

user ID	amt
2	$10
2	$15

Approach: send queries to multiple DBs...

```
SELECT * FROM tbl_purchase WHERE amt > 12
```



tbl_users

user ID	name
1	"Yiyin"
2	"Ivan"

tbl_purchases

user ID	amt
3	$20
...	

tbl_users

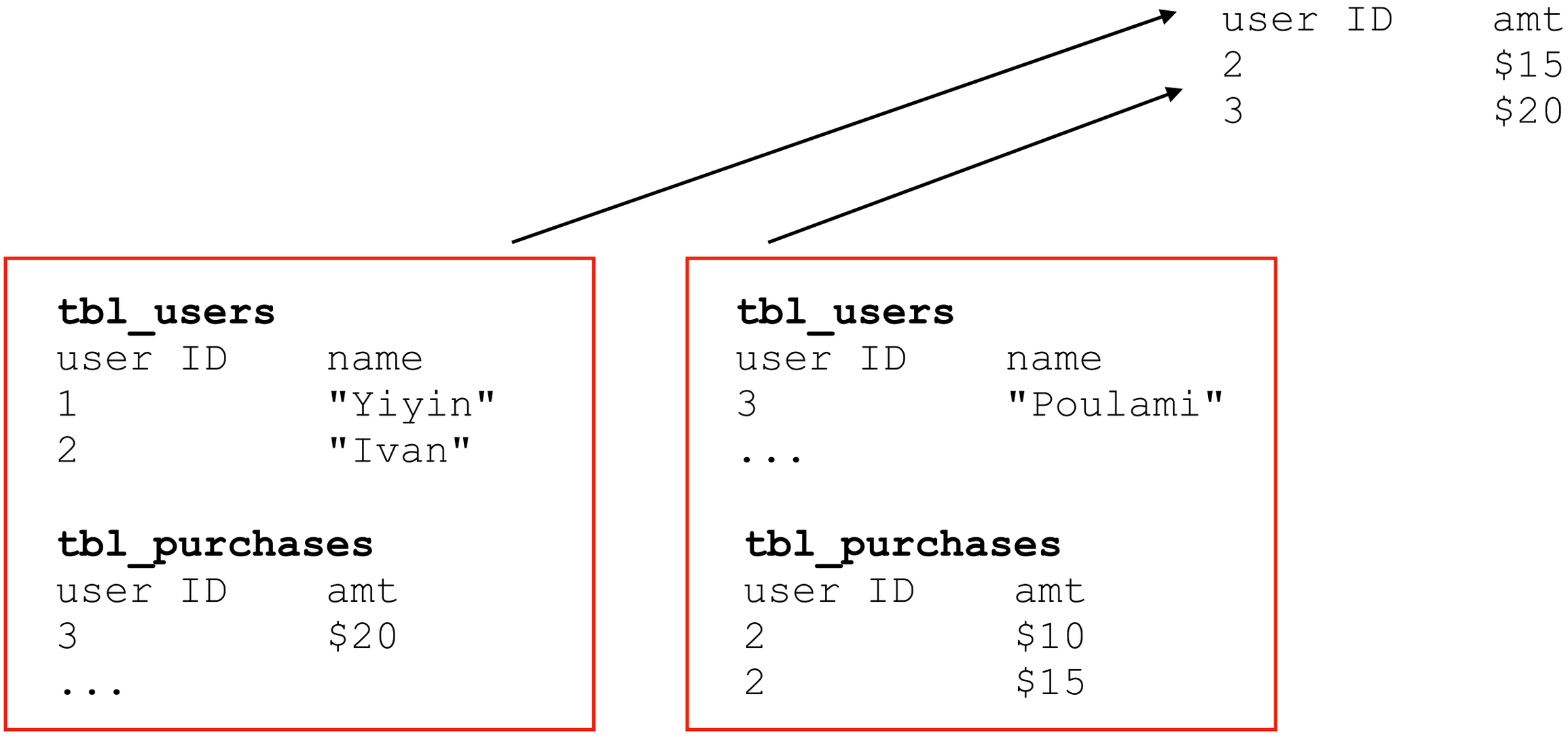
user ID	name
3	"Poulami"
...	

tbl_purchases

user ID	amt
2	$10
2	$15

...combine results

```
SELECT * FROM tbl_purchase WHERE amt > 12
```



tbl_purchases

user ID	amt
2	$15
3	$20

tbl_users

user ID	name
1	"Yiyin"
2	"Ivan"

tbl_purchases

user ID	amt
3	$20
...	

tbl_users

user ID	name
3	"Poulami"
...	

tbl_purchases

user ID	amt
2	$10
2	$15

What is a query that would break things?

```
SELECT ...
```

tbl_users

user ID	name
1	"Yiyin"
2	"Ivan"

tbl_purchases

user ID	amt
3	$20
...	

tbl_users

user ID	name
3	"Poulami"
...	

tbl_purchases

user ID	amt
2	$10
2	$15

What is a query that would break things?

```
SELECT * FROM tbl_users
INNER JOIN tbl_purchases
ON tbl_users.user_id = tbl_purchases.user_id
```

tbl_users

user ID	name
1	"Yiyin"
2	"Ivan"

tbl_purchases

user ID	amt
3	$20
...	

tbl_users

user ID	name
3	"Poulami"
...	

tbl_purchases

user ID	amt
2	$10
2	$15

Why use a traditional/relational DB if basic things like JOIN don't work right at scale?

example: Cassandra documentation

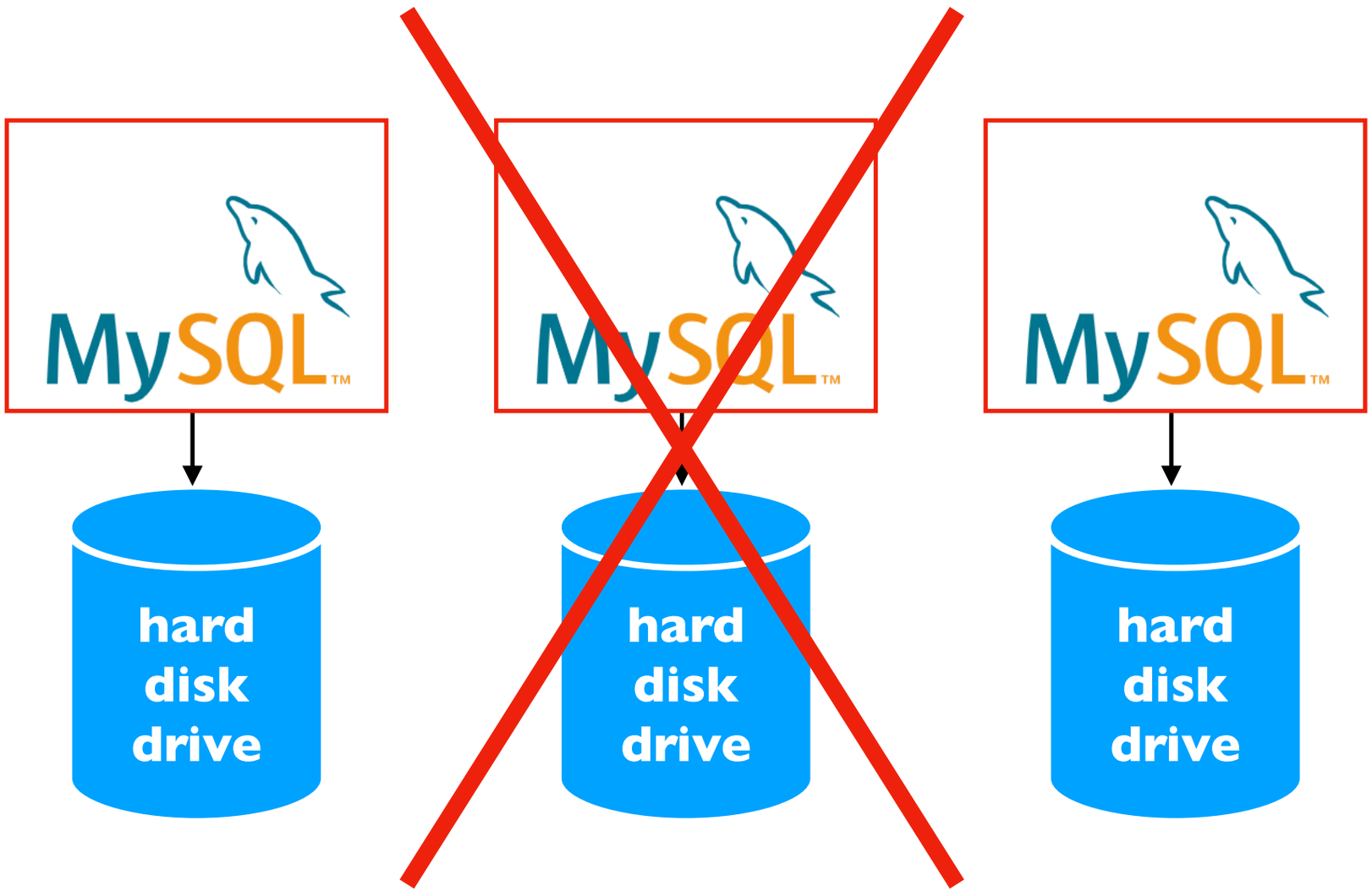
STEP 3: CREATE FILES

The Cassandra Query Language (CQL) is very similar to SQL but suited for the JOINless structure of Cassandra.

https://cassandra.apache.org/_/quickstart.html

What if a server dies?

happens all the time when you have 1000s of machines



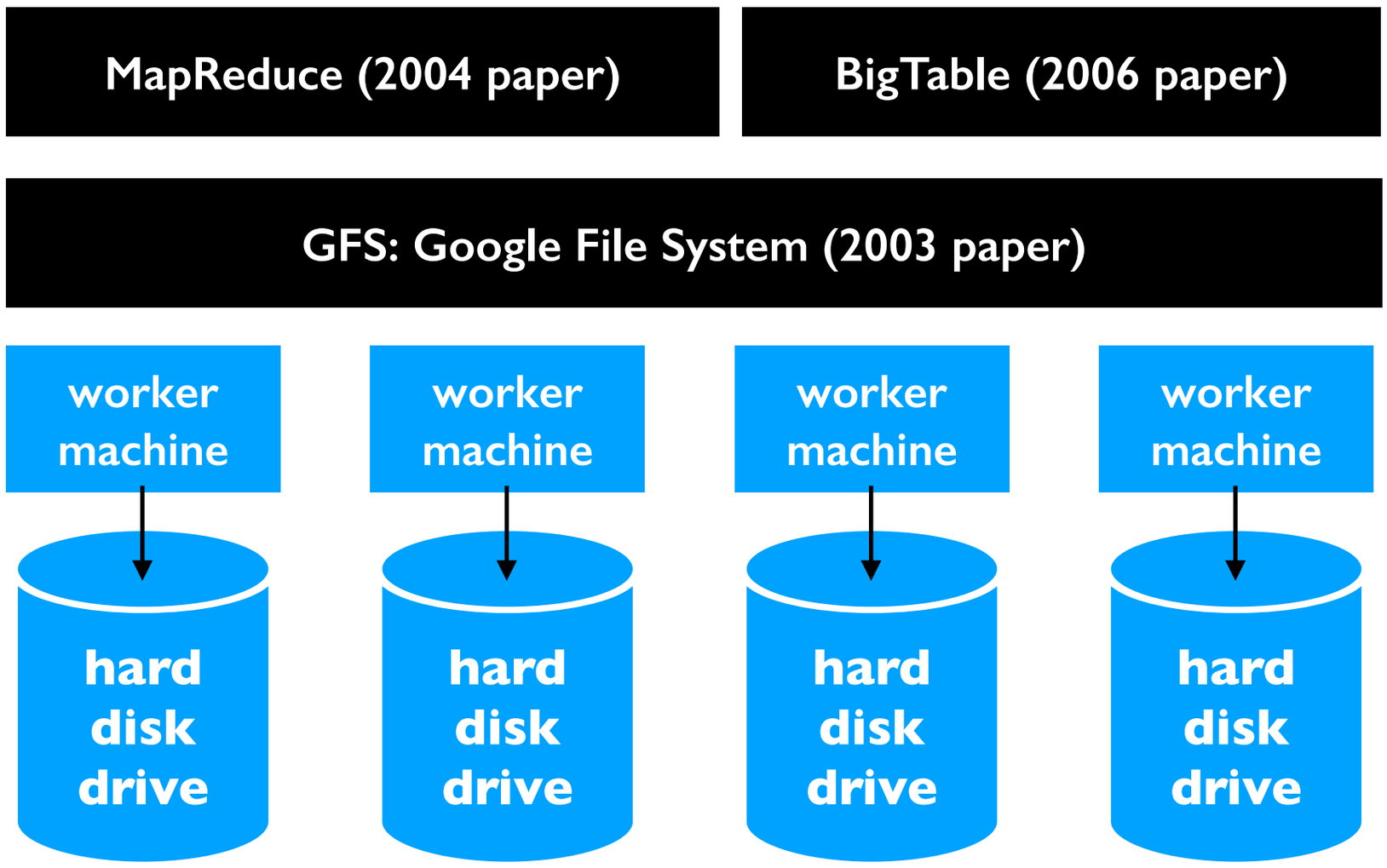
...1000s more...

Motivation for System Redesign

Features

- some classic features (like JOINS) may not be essential
- scaling to many machines is essential
- fault tolerance is essential

Google Architecture



radical idea: base everything on lots of cheap, commodity hardware

Hadoop Ecosystem

Yahoo, Facebook, Cloudera, and others developed open-source Hadoop ecosystem, mirroring Google's systems

	Google (paper only)	Hadoop, 1st gen (open source)		Modern Hadoop
Distributed File System	GFS	HDFS	covered today	
Distributed Analytics	MapReduce	Hadoop MapReduce		Spark
Distributed Database	BigTable	HBase		Cassandra

Outline: Hadoop Ecosystem

Architecture: Layered Data Systems

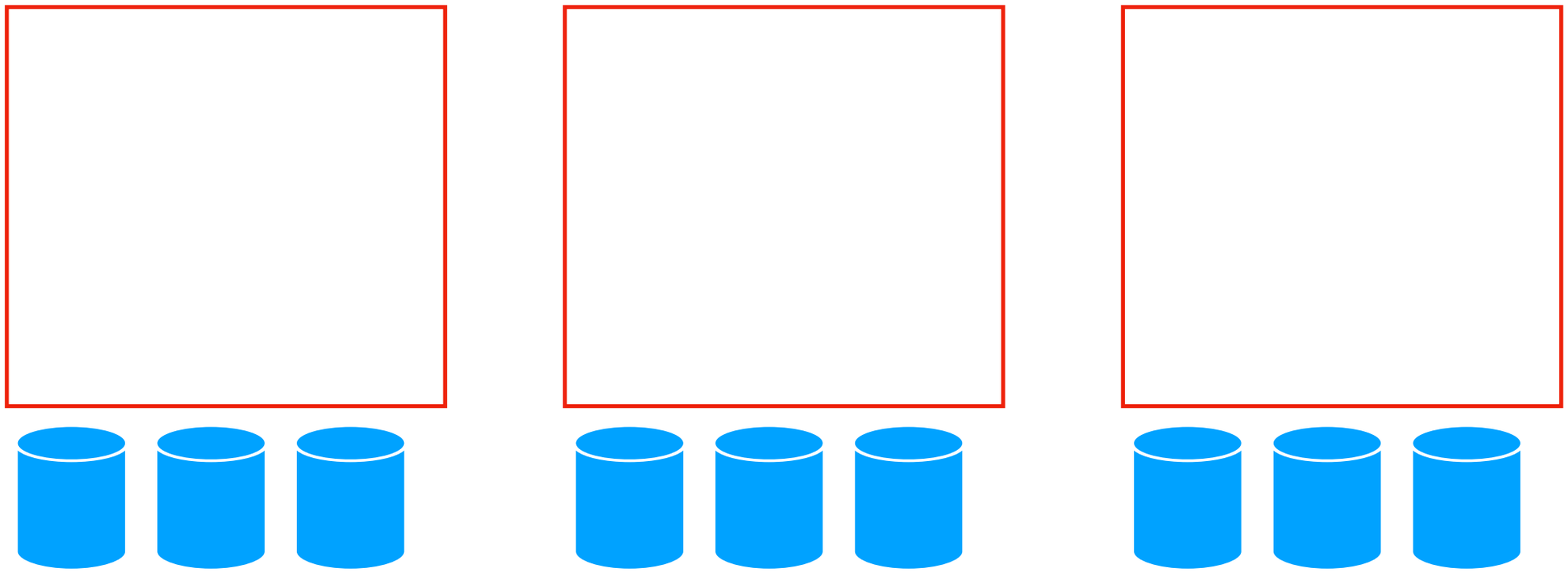
Hadoop File System

Hadoop MapReduce

HDFS: DataNodes store File Blocks

```
F1: "ABCD"                F2: "EFGHIJKL"
```

DataNode Computers

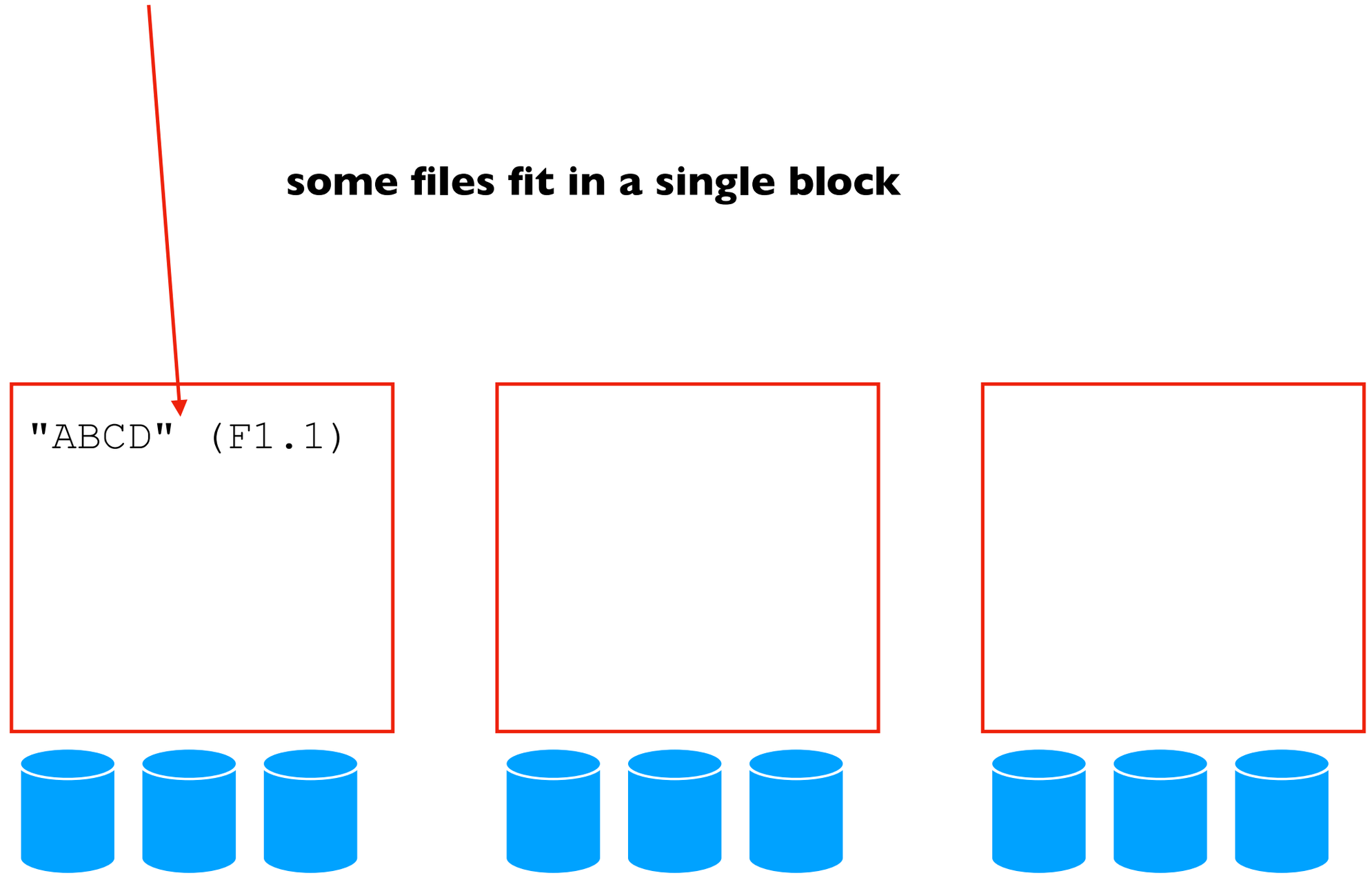


HDFS: DataNodes store File Blocks

F1: "ABCD"

F2: "EFGHIJKL"

some files fit in a single block



DataNode Computers

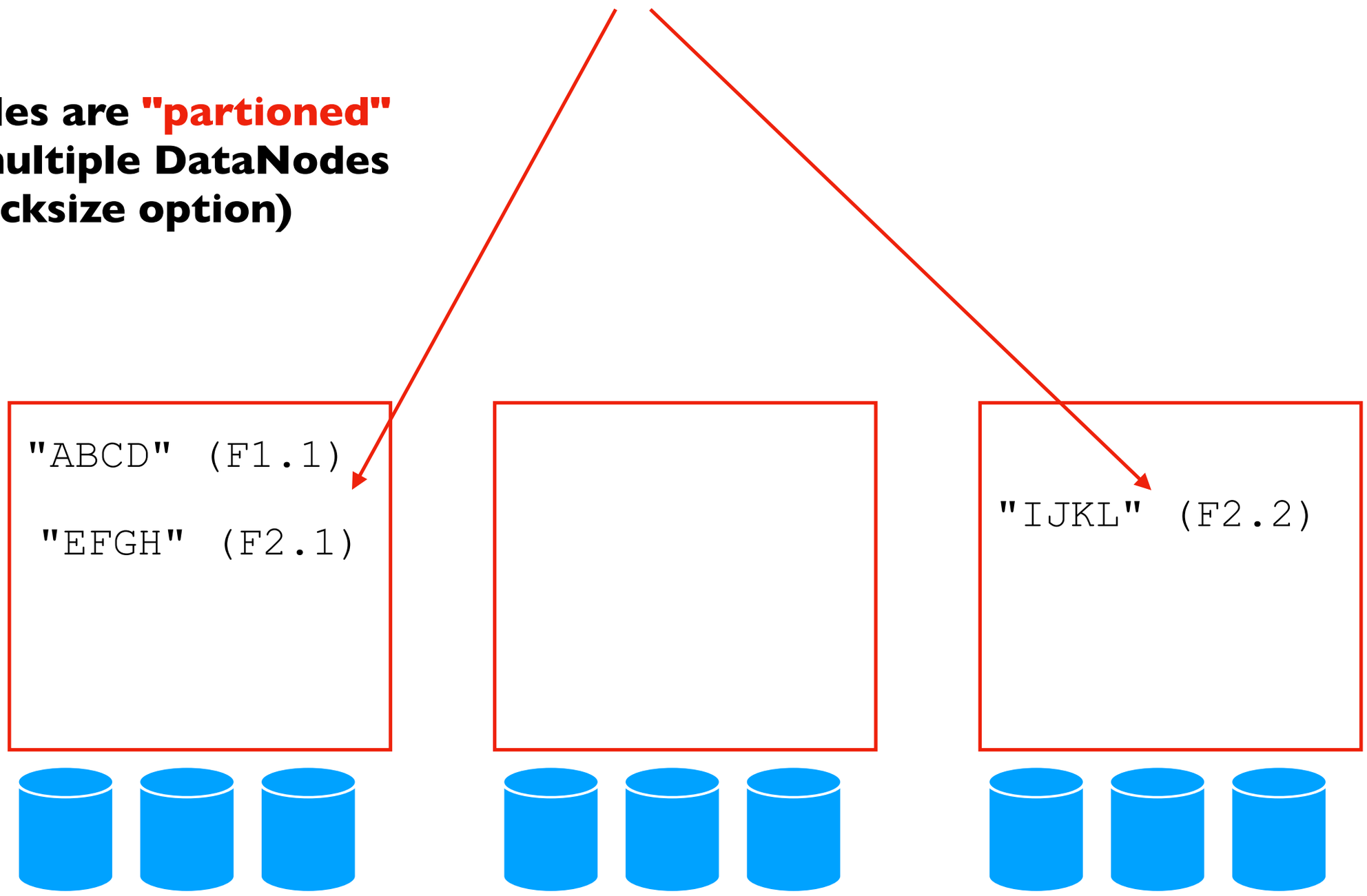
Partitioning Across DataNodes

F1: "ABCD"

F2: "EFGHIJKL"

bigger files are "partioned" across multiple DataNodes (blocksize option)

DataNode Computers



Replication Across DataNodes

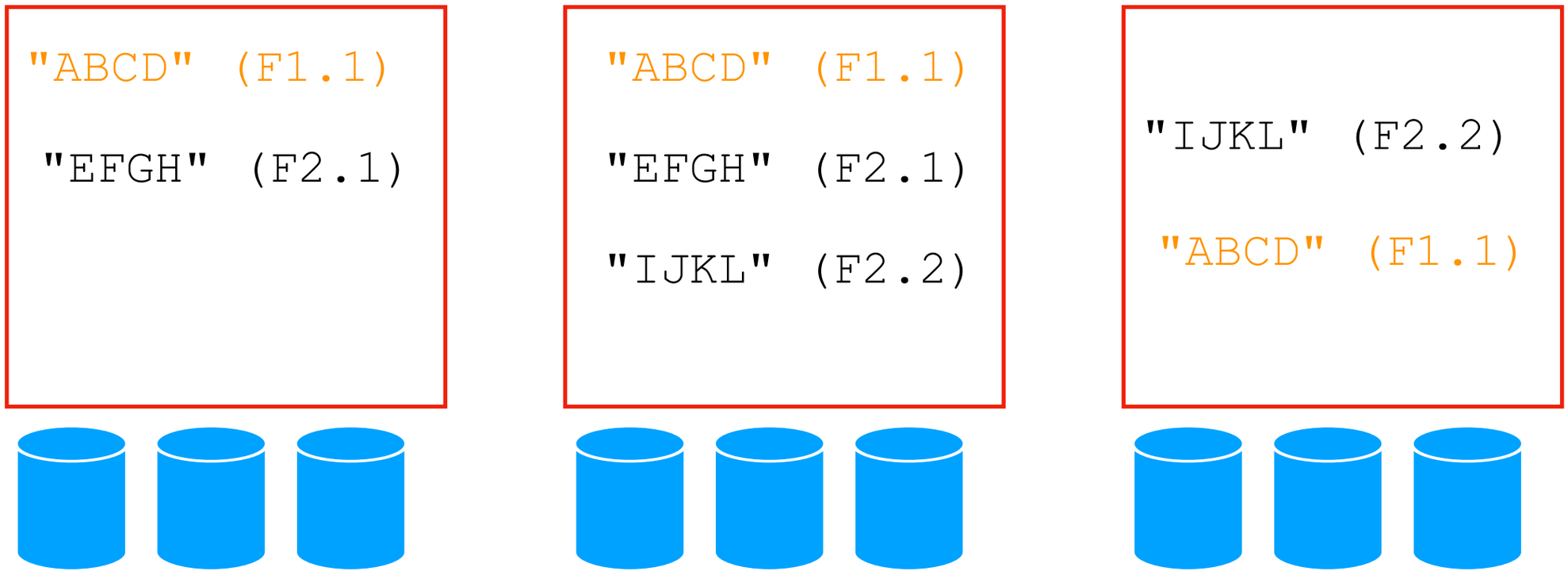
F1: "ABCD"

3x replication

F2: "EFGHIJKL"

2x relpication

DataNode Computers



Replication Across DataNodes

F1: "ABCD"

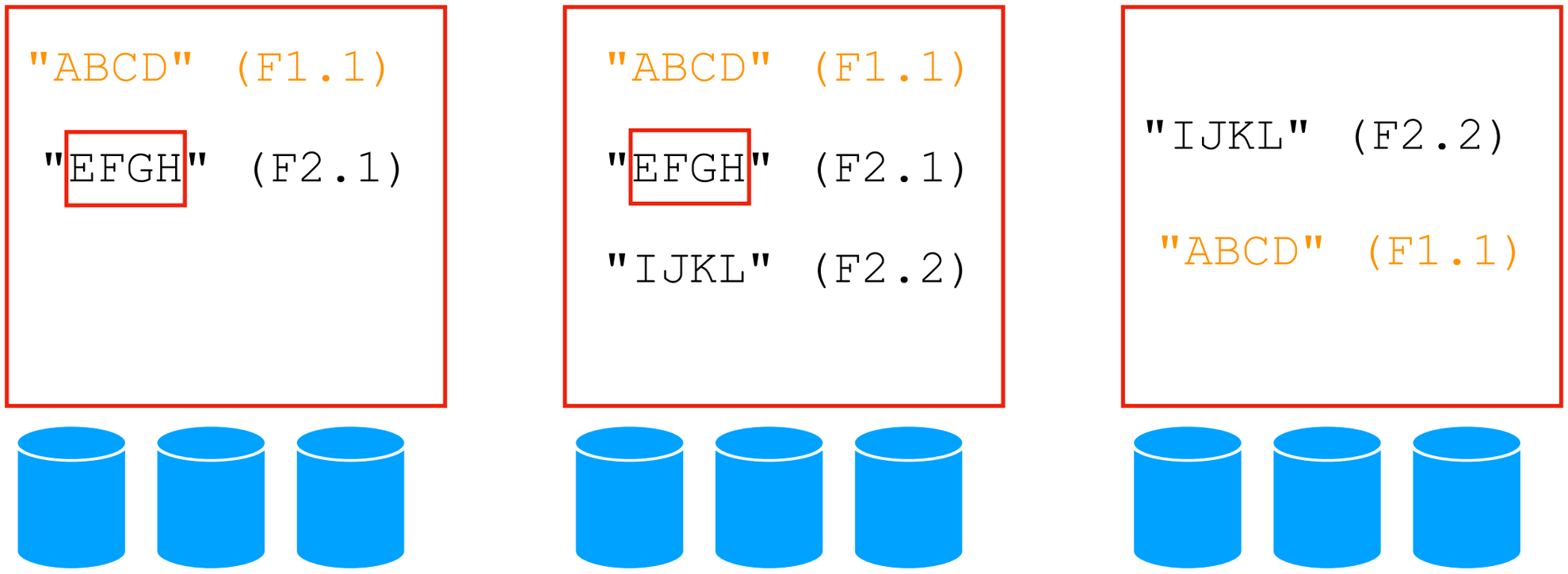
3x replication

F2: "EFGHIJKL"

2x relpication

logical vs. physical blocks

DataNode Computers



Replication Across DataNodes

F1: "ABCD"

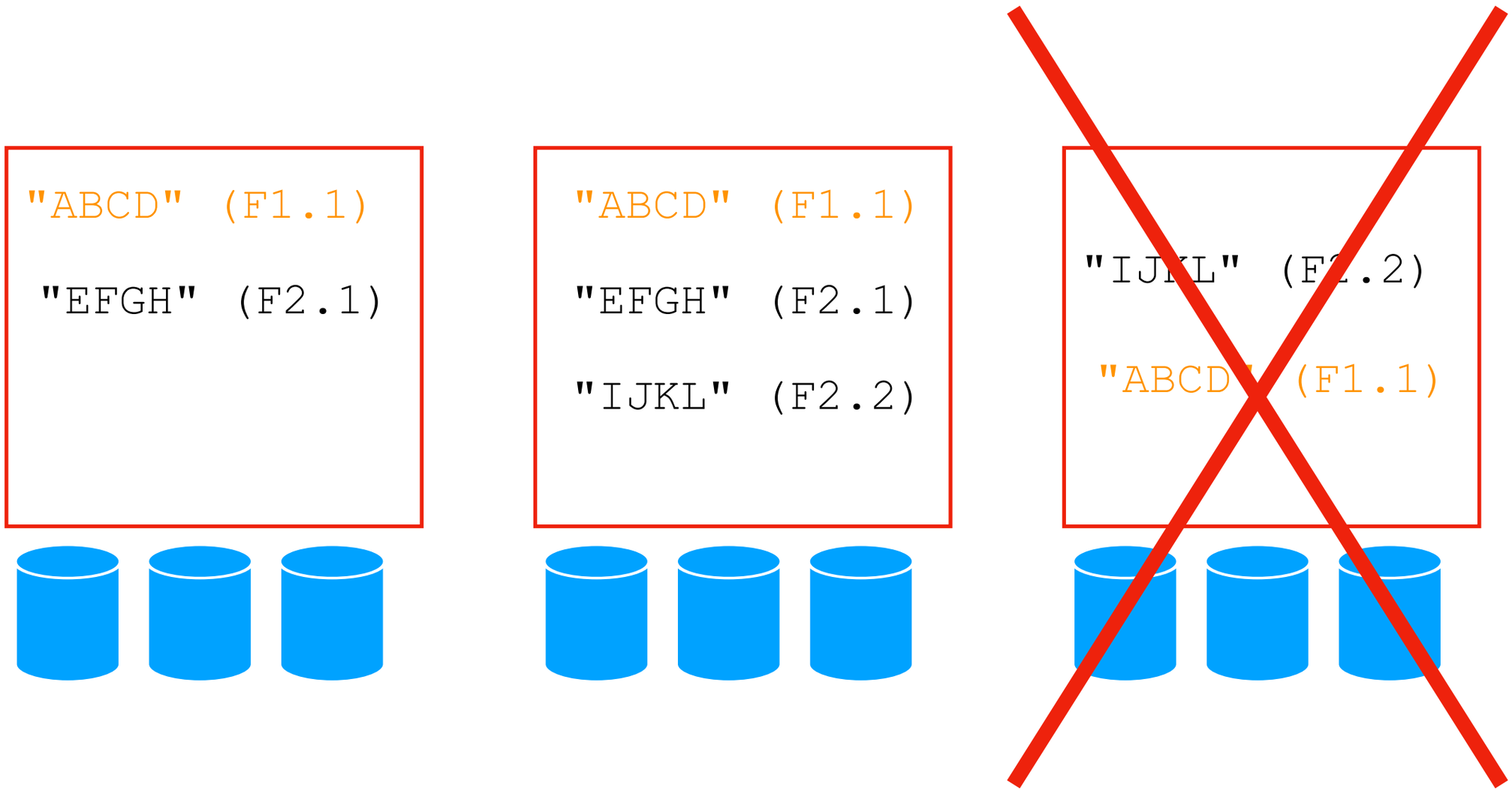
3x replication

F2: "EFGHIJKL"

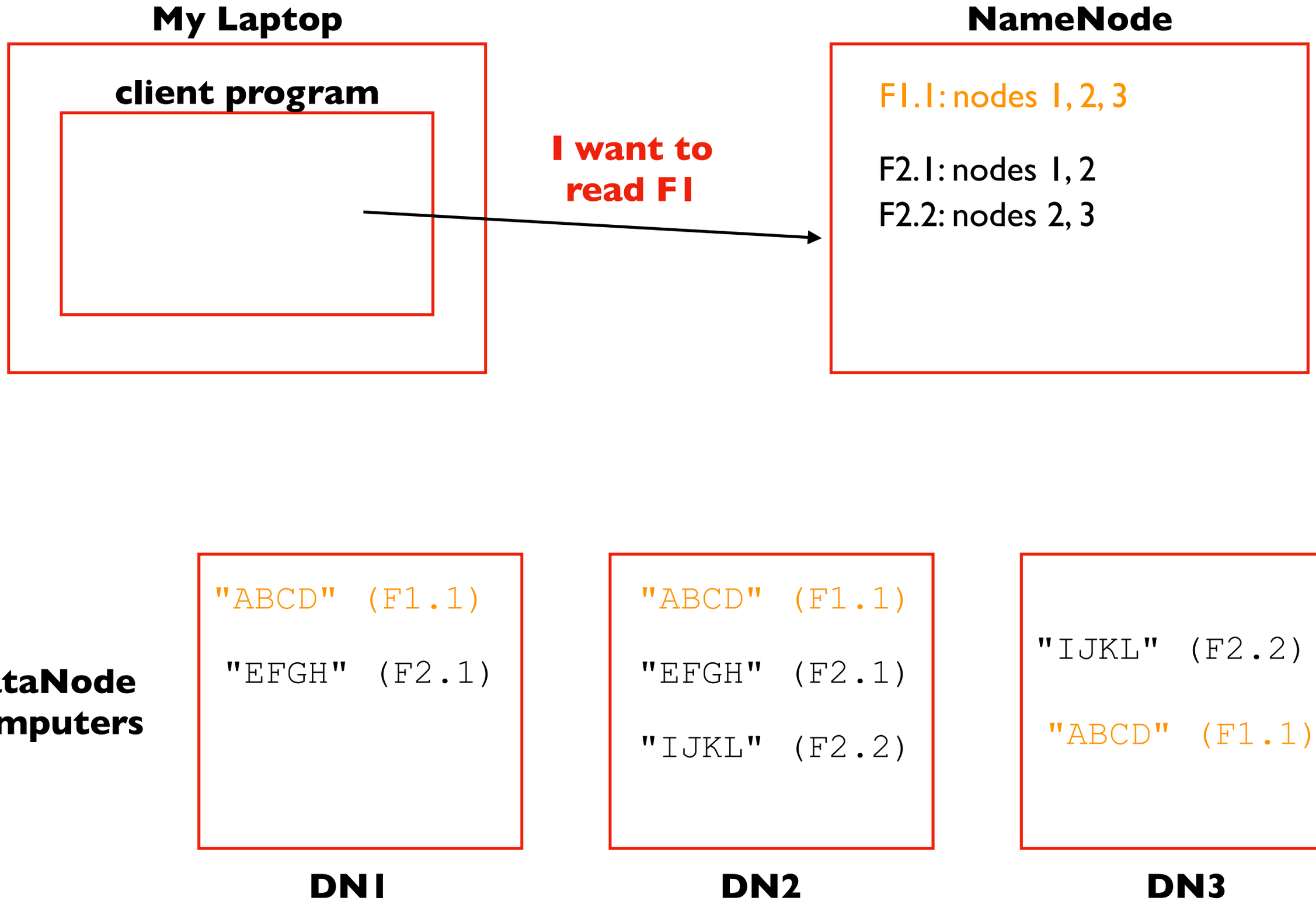
2x relpication

if a DataNode dies, we still have all the data.
Which file (F1 or F2) is safer in general?

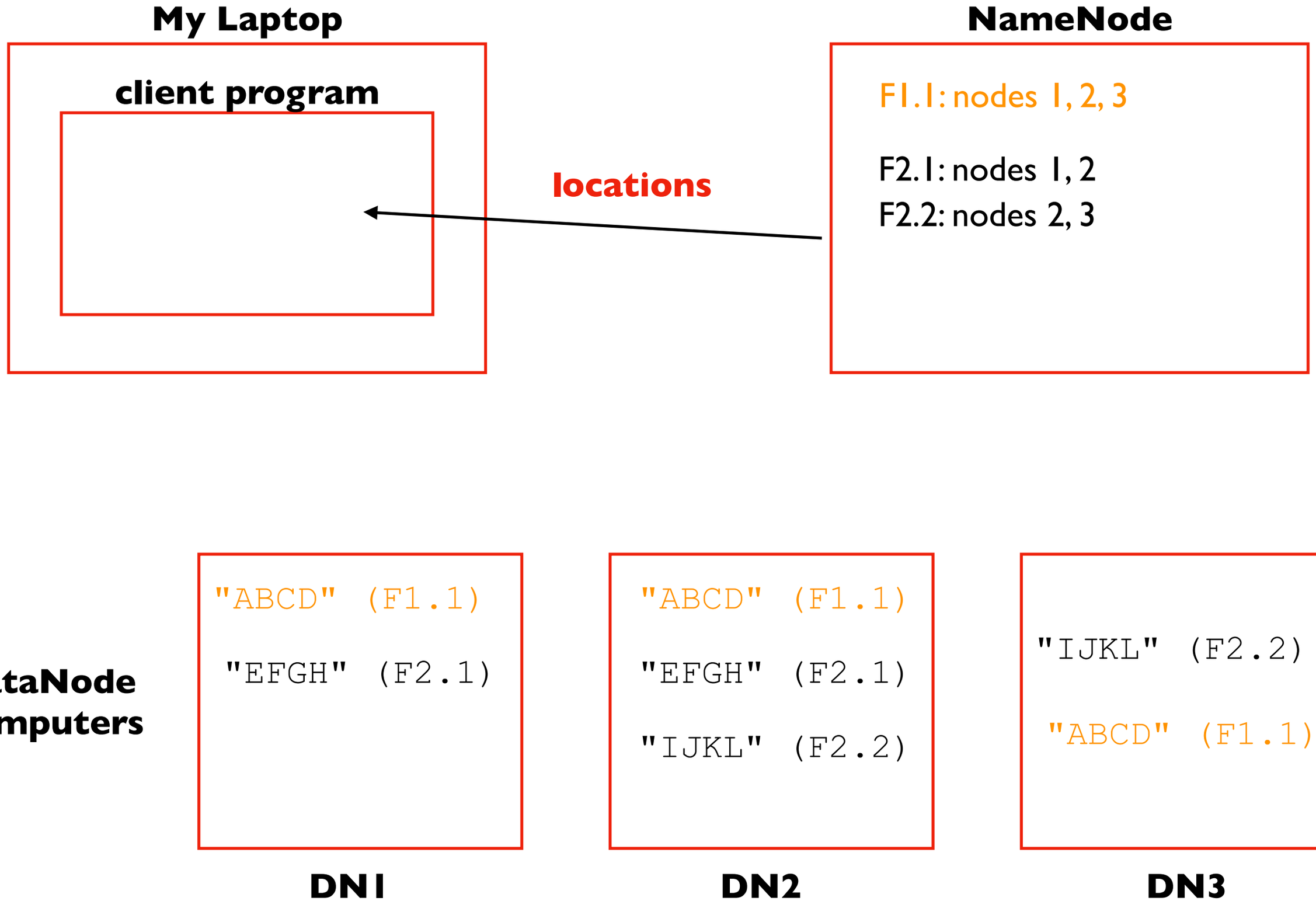
DataNode Computers



Boss (NameNode)/Worker Architecture



Boss (NameNode)/Worker Architecture



Boss (NameNode)/Worker Architecture

My Laptop

client program

NameNode

F1.1: nodes 1, 2, 3

F2.1: nodes 1, 2

F2.2: nodes 2, 3

network transfer

DataNode Computers

DN1

"ABCD" (F1.1)

"EFGH" (F2.1)

DN2

"ABCD" (F1.1)

"EFGH" (F2.1)

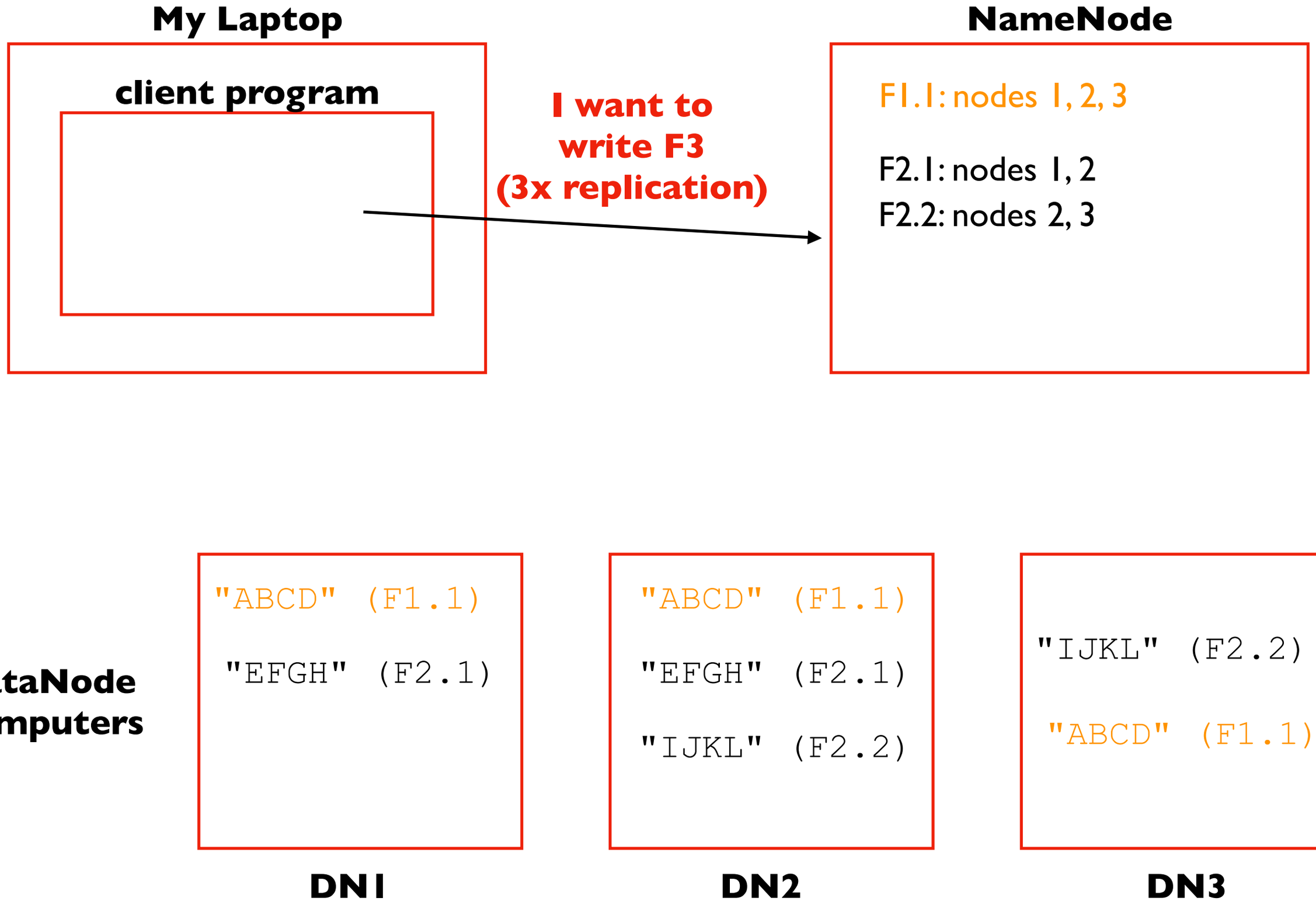
"IJKL" (F2.2)

DN3

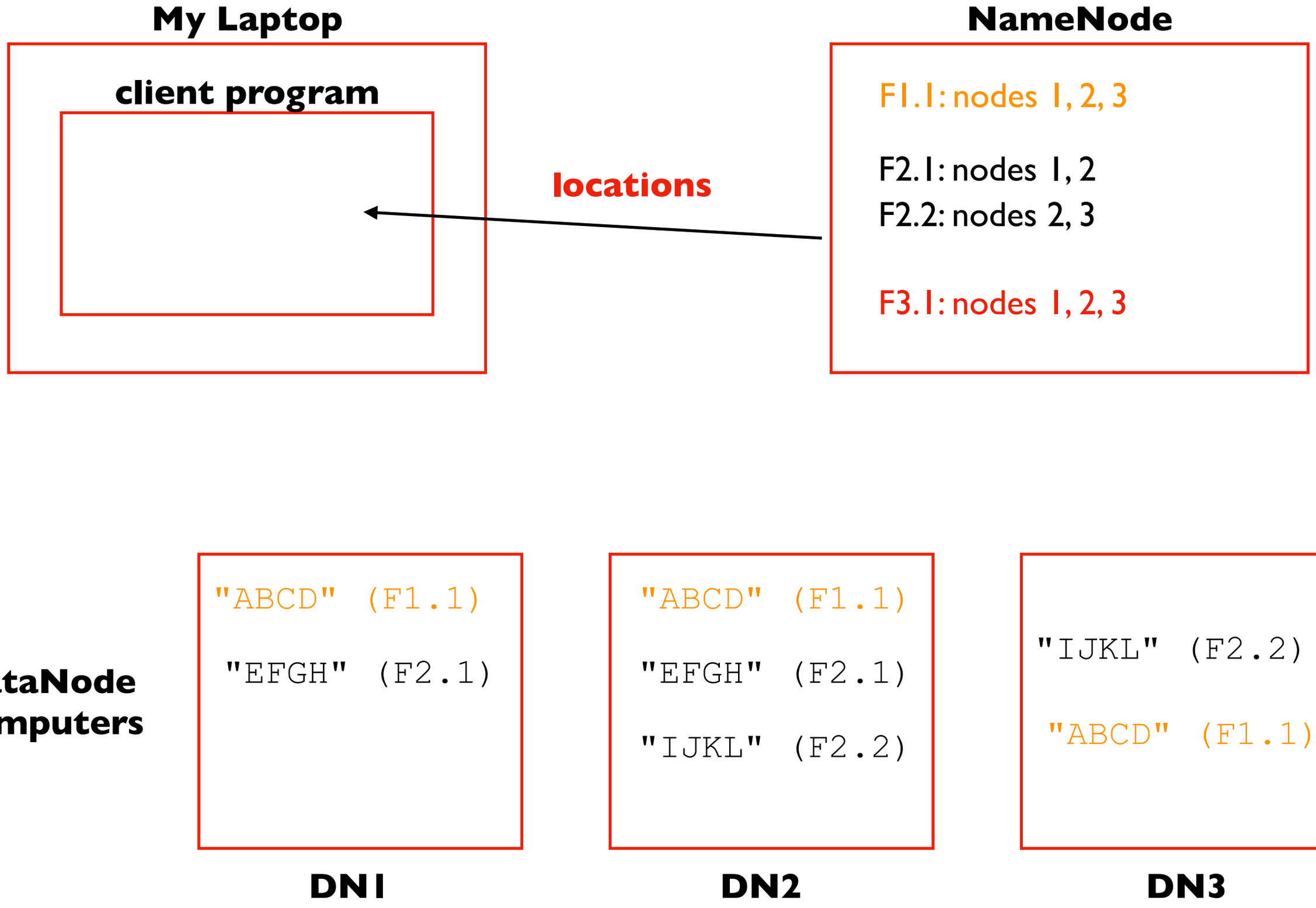
"IJKL" (F2.2)

"ABCD" (F1.1)

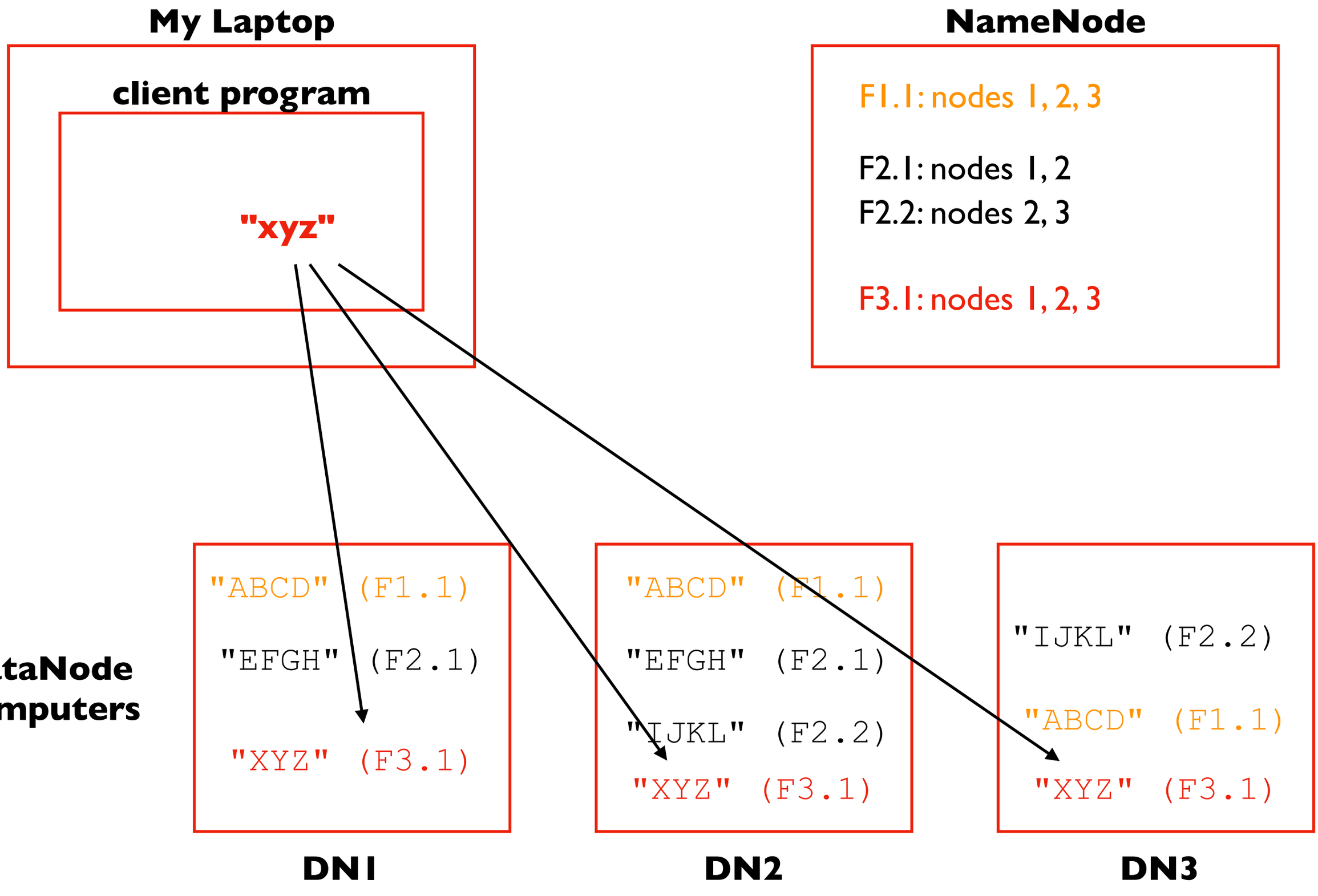
Boss (NameNode)/Worker Architecture



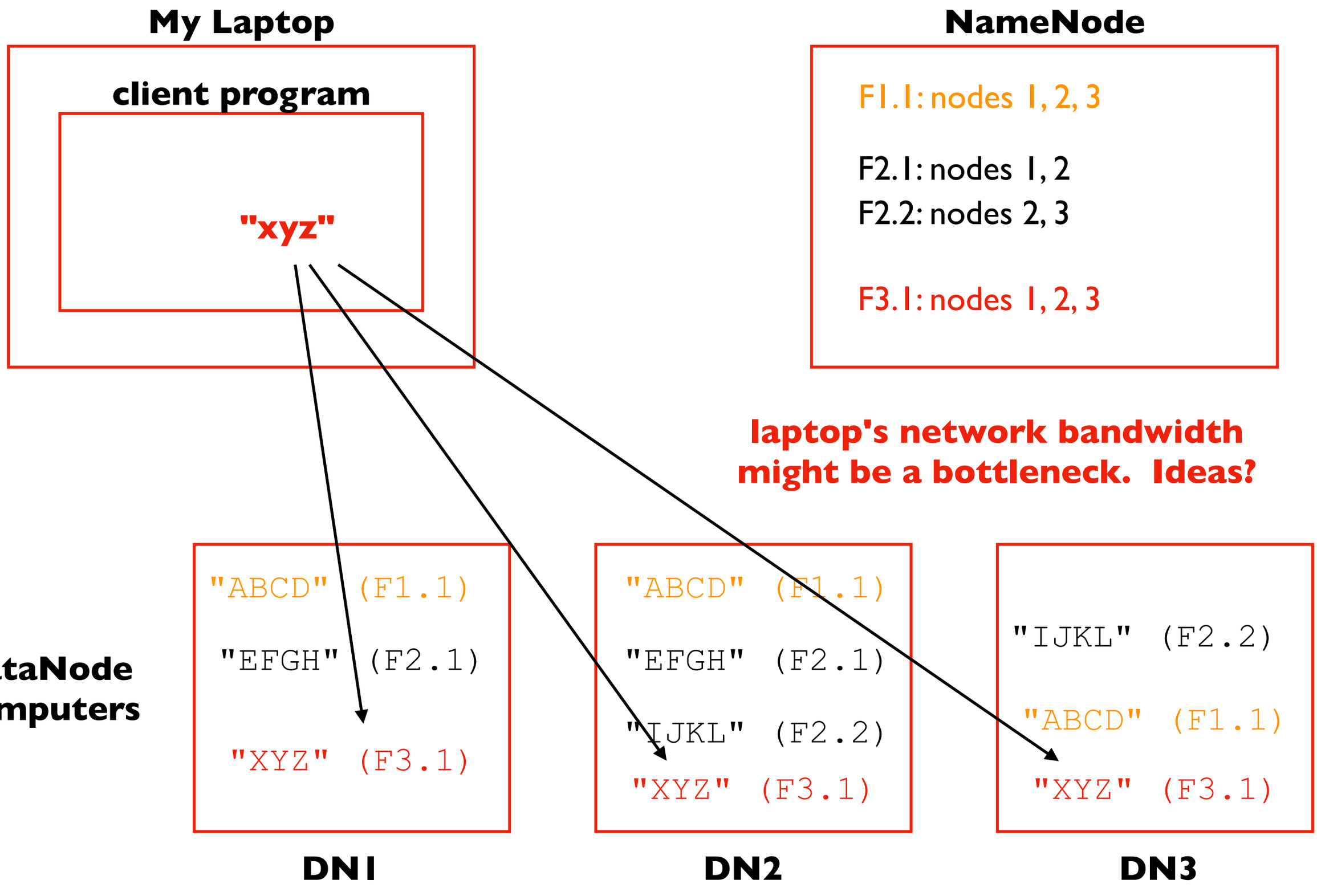
Boss (NameNode)/Worker Architecture



Boss (NameNode)/Worker Architecture



Boss (NameNode)/Worker Architecture



Pipeline Writes

My Laptop

client program

"xyz"

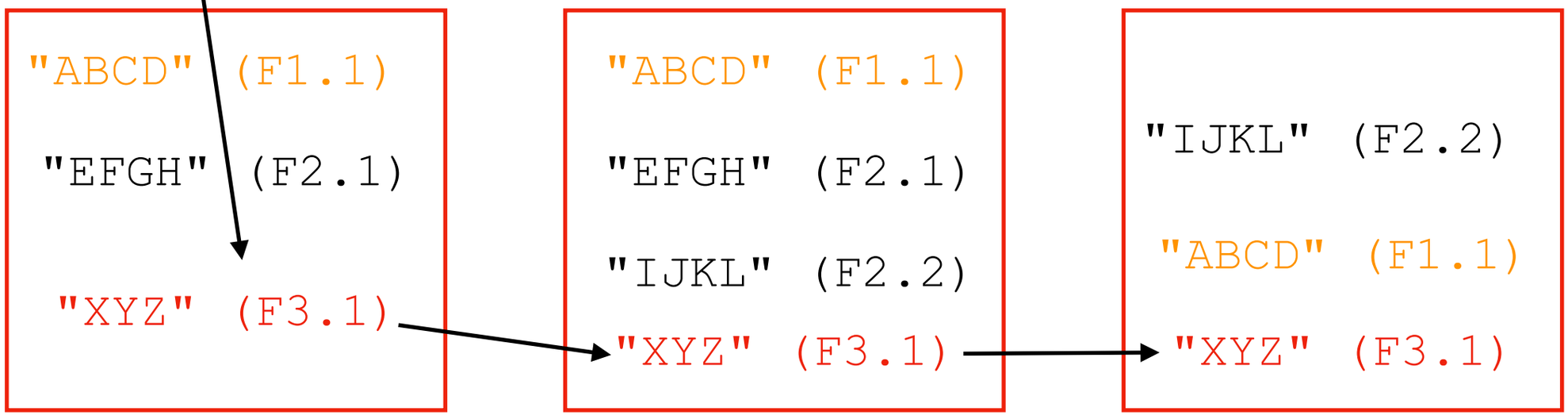
NameNode

F1.1: nodes 1, 2, 3

F2.1: nodes 1, 2
F2.2: nodes 2, 3

F3.1: nodes 1, 2, 3

DataNode Computers



DN1 **DN2** **DN3**

How are reads/writes amplified at disk level?

if a client **writes** 4 MB to a 2x replicated file, how much data do we **write** to hard drives?

if a client **reads** 2 MB to a 3x replicated file, how much data do we **read** from hard drives?

NameNode

F1.1: nodes 1, 2, 3

F2.1: nodes 1, 2
F2.2: nodes 2, 3

F3.1: nodes 1, 2, 3

DataNode Computers

DN1	DN2	DN3
"ABCD" (F1.1)	"ABCD" (F1.1)	"IJKL" (F2.2)
"EFGH" (F2.1)	"EFGH" (F2.1)	"ABCD" (F1.1)
"XYZ" (F3.1)	"IJKL" (F2.2)	"XYZ" (F3.1)
	"XYZ" (F3.1)	

What are the tradeoffs of replication factor and block size?

benefits of high replication?

benefits of low replication?

benefits of large block size?

benefits of small block size?

NameNode

F1.1: nodes 1, 2, 3

F2.1: nodes 1, 2
F2.2: nodes 2, 3

F3.1: nodes 1, 2, 3

DataNode Computers

DN1

"ABCD" (F1.1)

"EFGH" (F2.1)

"XYZ" (F3.1)

DN2

"ABCD" (F1.1)

"EFGH" (F2.1)

"IJKL" (F2.2)

"XYZ" (F3.1)

DN3

"IJKL" (F2.2)

"ABCD" (F1.1)

"XYZ" (F3.1)

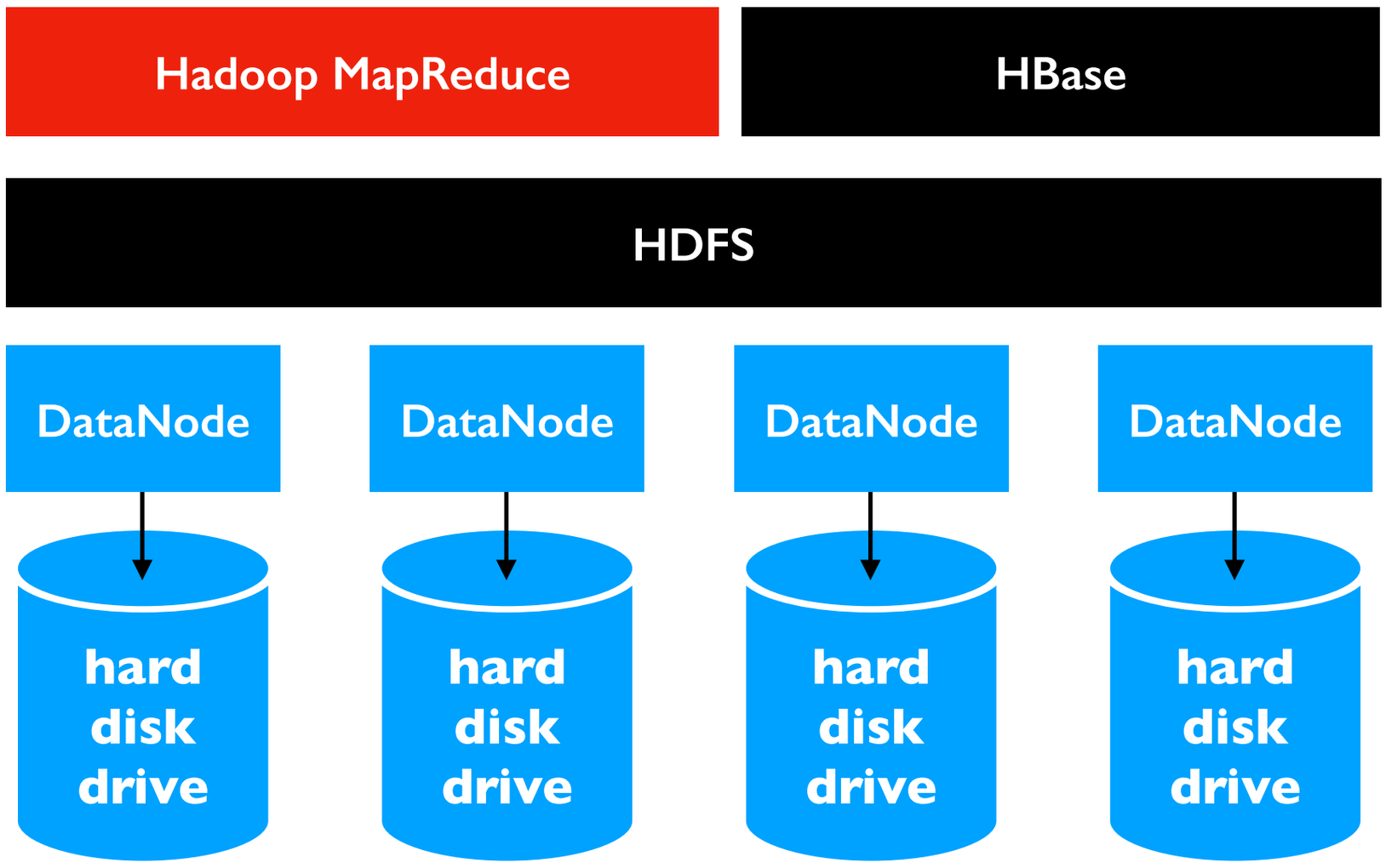
Outline: Hadoop Ecosystem

Architecture: Layered Data Systems

Hadoop File System

Hadoop MapReduce

MapReduce


Hadoop MapReduce
HBase
HDFS
DataNode
DataNode
DataNode
DataNode
hard disk drive
hard disk drive
hard disk drive
hard disk drive

How do we answer questions?

SQL:

a query, "SELECT * FROM ..." → **Database** → **results**

MapReduce

map function code
reduce function code
mapper/reducer counts
a file (or many)

→ **MapReduce** → **output files**

How do we answer questions?

SQL:

a query, "SELECT * FROM ..." → Database → **results**

MapReduce

map function code
reduce function code
mapper/reducer counts
a file (or many)

→ MapReduce → **output files**

input/output files are generally in HDFS

How do we answer questions?

SQL:

a query, "SELECT * FROM ..." → Database → **results**

MapReduce

map function code
reduce function code
mapper/reducer counts
a file (or many)

→ MapReduce → **output files**

Mappers by example: what are the colors of the squares?

input.csv (in HDFS):

```
color, shape,   size
red,   circle,  3
red,   square,  5
blue,  oval,    1
green, square,  3
```

```
def map(key, value):
    ...
```

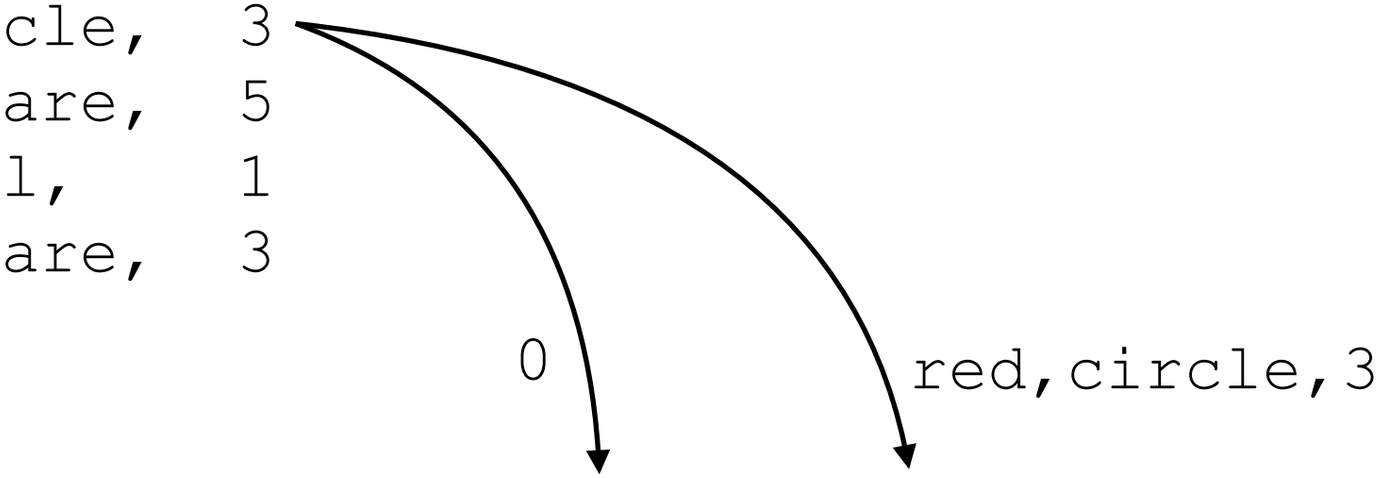
In SQL:

SELECT color FROM table WHERE shape = "square";

Mappers by example: what are the colors of the squares?

input.csv (in HDFS):

```
color,  shape,   size
red,    circle,  3
red,    square,  5
blue,   oval,    1
green,  square,  3
```



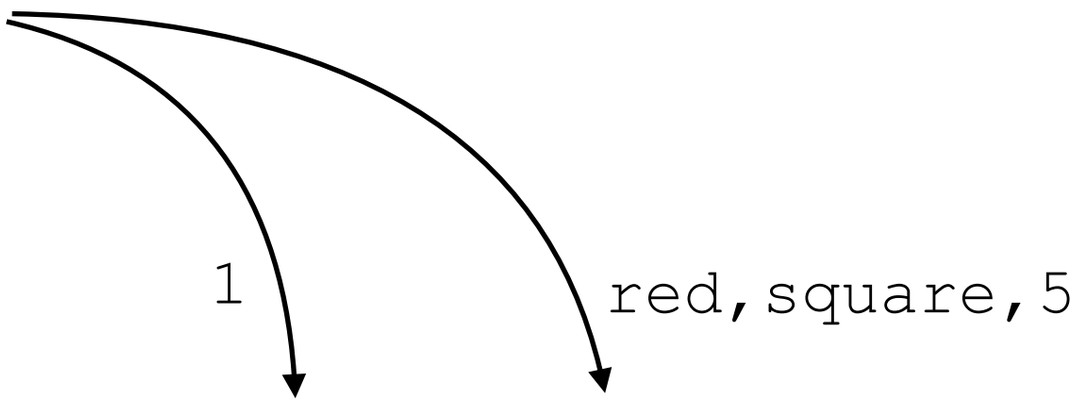

```
def map(key, value):
    ...
```

zero or more output key/value pairs

Mappers by example: what are the colors of the squares?

input.csv (in HDFS):

```
color,  shape,   size
red,    circle,  3
red,    square,  5
blue,   oval,    1
green,  square,  3
```




```
def map(key, value):
    ...
```

zero or more output key/value pairs

Mappers by example: what are the colors of the squares?

input.csv (in HDFS):

color,	shape,	size
red,	circle,	3
red,	square,	5
blue,	oval,	1
green,	square,	3

1

red,square,5

```
def map(key, value):
    if value.shape = square:
        emit(key, value.color)
```

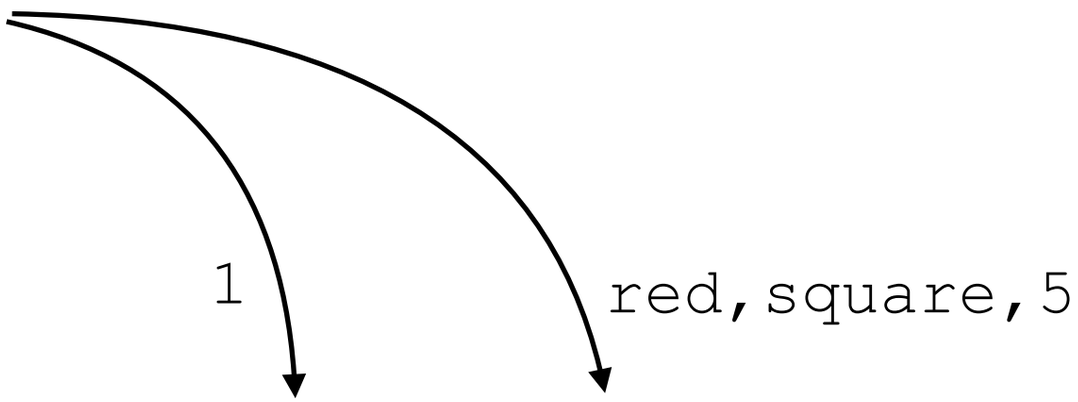
key	value
1	red
3	green

Mappers by example: what are the colors of the squares?

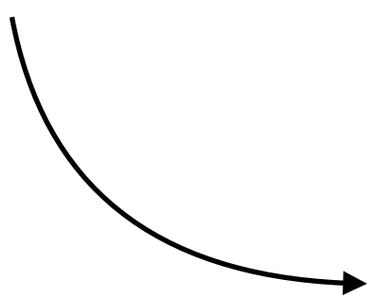
what if the data is huge?

input.csv (in HDFS):

```
color,  shape,   size
red,    circle,  3
red,    square,  5
blue,   oval,    1
green,  square,  3
```




```
def map(key, value):
    if value.shape = square:
        emit(key, value.color)
```



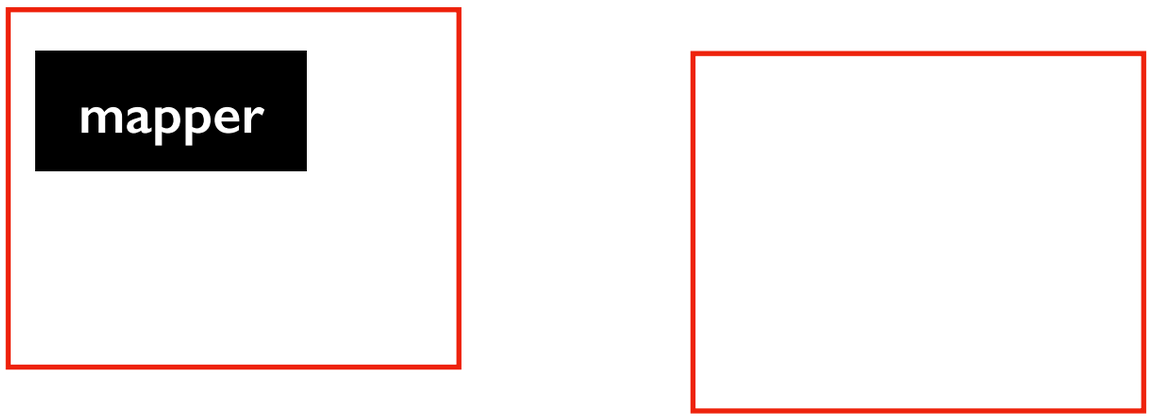
key	value
1	red
3	green

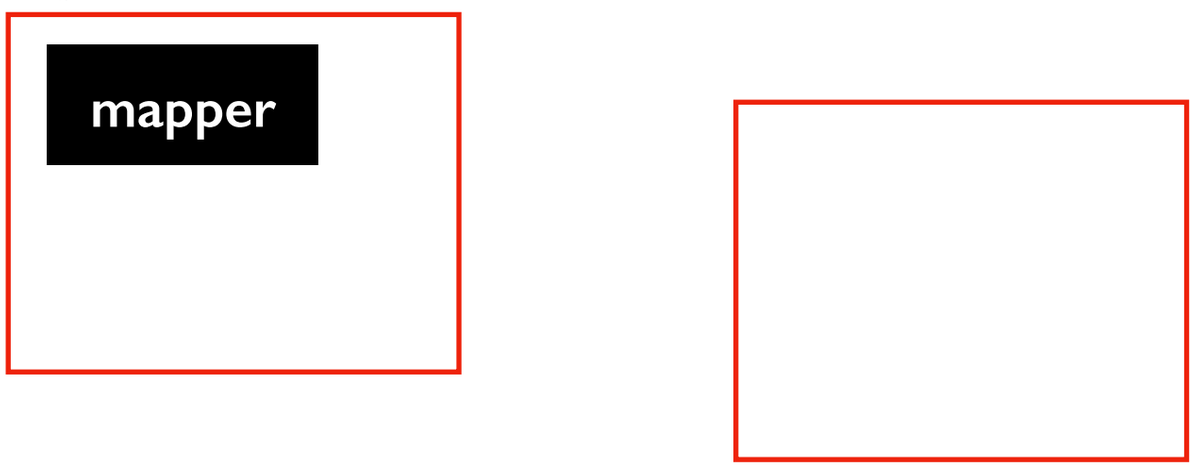
Mappers Run on Multiple Machines at Once

cluster of machines

input.csv (in HDFS):

```
color,  shape,    size
red,    circle,   3
red,    square,   5
blue,   oval,     1
green,  square,   3
```



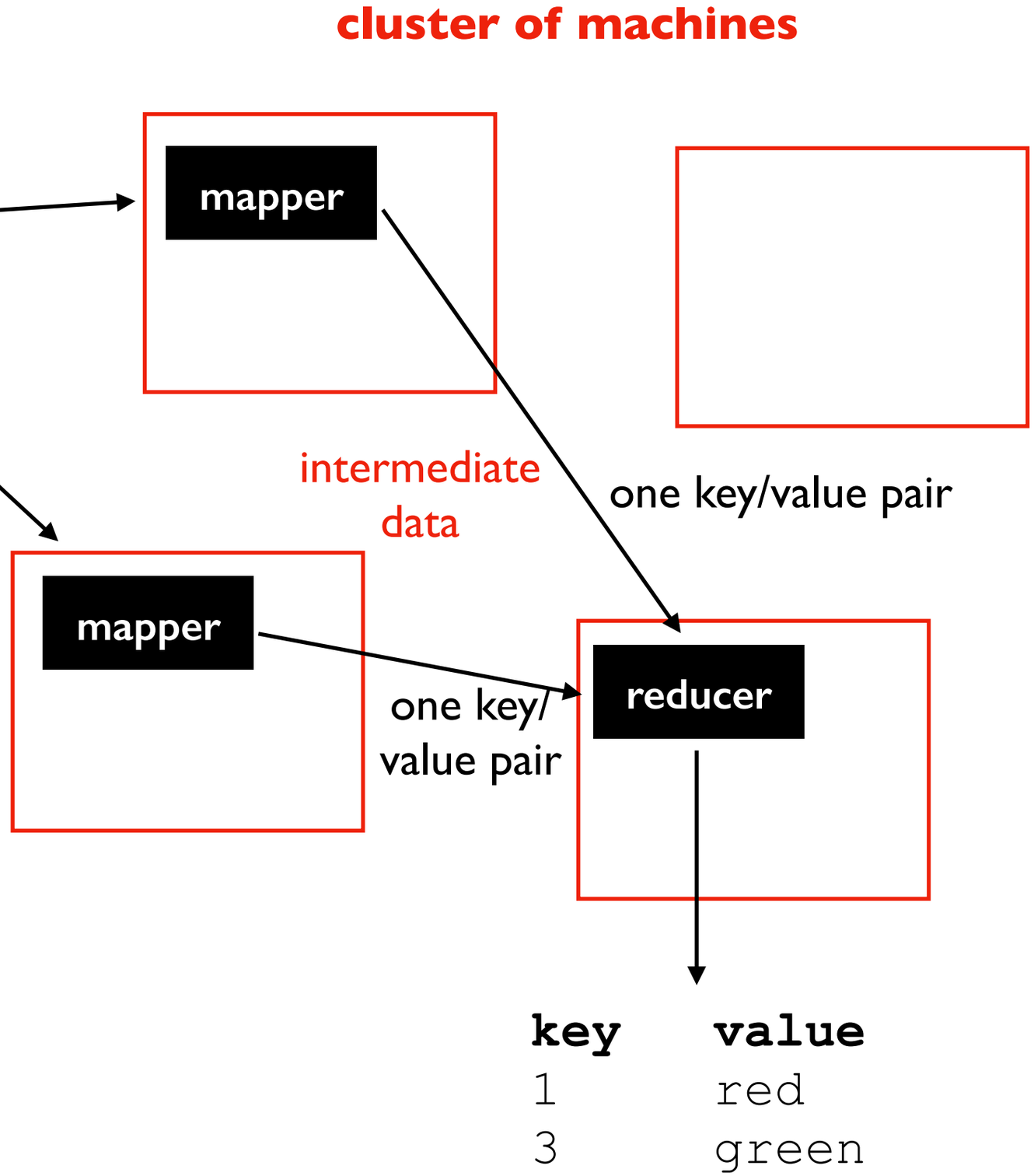


Reducers

input.csv (in HDFS):

```
color,  shape,    size
red,    circle,   3
red,    square,   5
blue,   oval,     1
green,  square,   3
```

a simple (default) reduce task can combine output of multiple mappers to a single file



Reducers

**reducers can output exactly their input,
OR have further computation**

```
def reduce(key, values):
    for row in values:
        emit(key, row)
```

Reducers

color,	shape,	size
red,	circle,	3
red,	square,	5
blue,	oval,	1
green,	square,	3

```
def map(key, value):
    emit(value.color, value)
```

key	value		
blue	blue,	oval,	1
green	green,	square,	3
red	red,	circle,	3
red,	red,	square,	5

```
def reduce(key, values):
    count = 0
    for row in values:
        count = count + 1
    emit(key, count)
```

intermediate data is grouped and sorted by

Reducers

color,	shape,	size
red,	circle,	3
red,	square,	5
blue,	oval,	1
green,	square,	3

```
def map(key, value):
    emit(value.color, value)
```

key	value
blue	blue, oval, 1
green	green, square, 3
red	red, circle, 3
red,	red, square, 5

intermediate data is grouped and sorted by

```
def reduce(key, values):
    count = 0
    for row in values:
        count = count + 1
    emit(key, count)
```

key	value
blue	1

Reducers

color,	shape,	size
red,	circle,	3
red,	square,	5
blue,	oval,	1
green,	square,	3

```
def map(key, value):
    emit(value.color, value)
```

key	value		
blue	blue,	oval,	1
green	green,	square,	3
red	red,	circle,	3
red,	red,	square,	5

intermediate data is grouped and sorted by

```
def reduce(key, values):
    count = 0
    for row in values:
        count = count + 1
    emit(key, count)
```

key	value
blue	1
green	1

Reducers

color,	shape,	size
red,	circle,	3
red,	square,	5
blue,	oval,	1
green,	square,	3

```
def map(key, value):
    emit(value.color, value)
```

key	value		
blue	blue,	oval,	1
green	green,	square,	3
red	red,	circle,	3
red,	red,	square,	5

intermediate data is grouped and sorted by

```
def reduce(key, values):
    count = 0
    for row in values:
        count = count + 1
    emit(key, count)
```

key	value
blue	1
green	1
red	2

What is the SQL equivalent of this MapReduce program?

color,	shape,	size
red,	circle,	3
red,	square,	5
blue,	oval,	1
green,	square,	3

```
def map(key, value):
    emit(value.color, value)
```

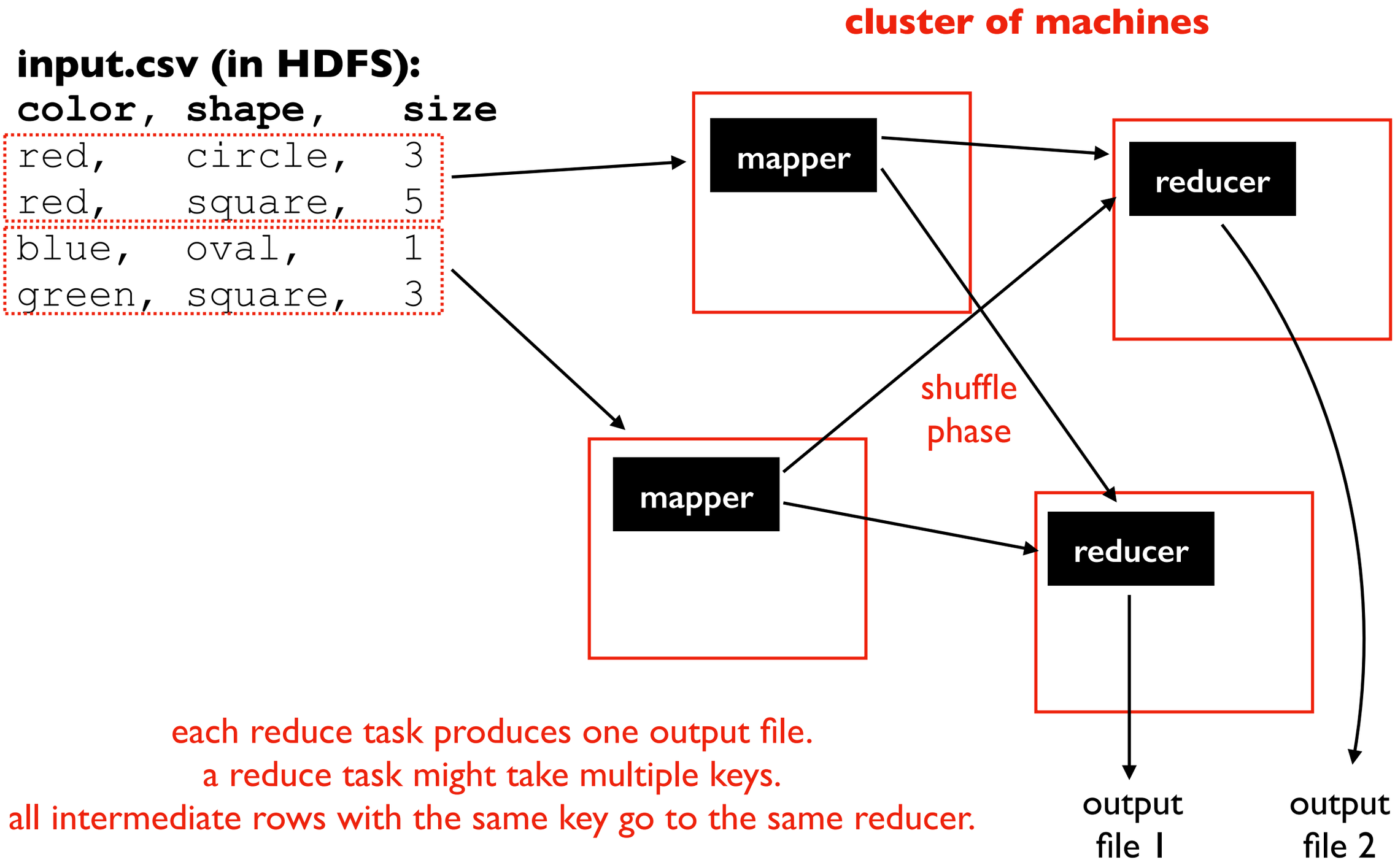
key	value		
blue	blue,	oval,	1
green	green,	square,	3
red	red,	circle,	3
red,	red,	square,	5

intermediate data is grouped and sorted by

```
def reduce(key, values):
    count = 0
    for row in values:
        count = count + 1
    emit(key, count)
```

key	value
blue	1
green	1
red	2

Multiple Reducers (for big intermediate data)



each reduce task produces one output file.
a reduce task might take multiple keys.
all intermediate rows with the same key go to the same reducer.

SQL => MapReduce

Map Phase

- SELECT, WHERE, JOIN

Shuffle Phase

- ORDER BY

Reduce Phase

- GROUPBY/AGGREGATE, HAVING, JOIN

MapReduce is more flexible. (for example, how to do a GROUP BY where one row goes to mutliple groups in SQL?)

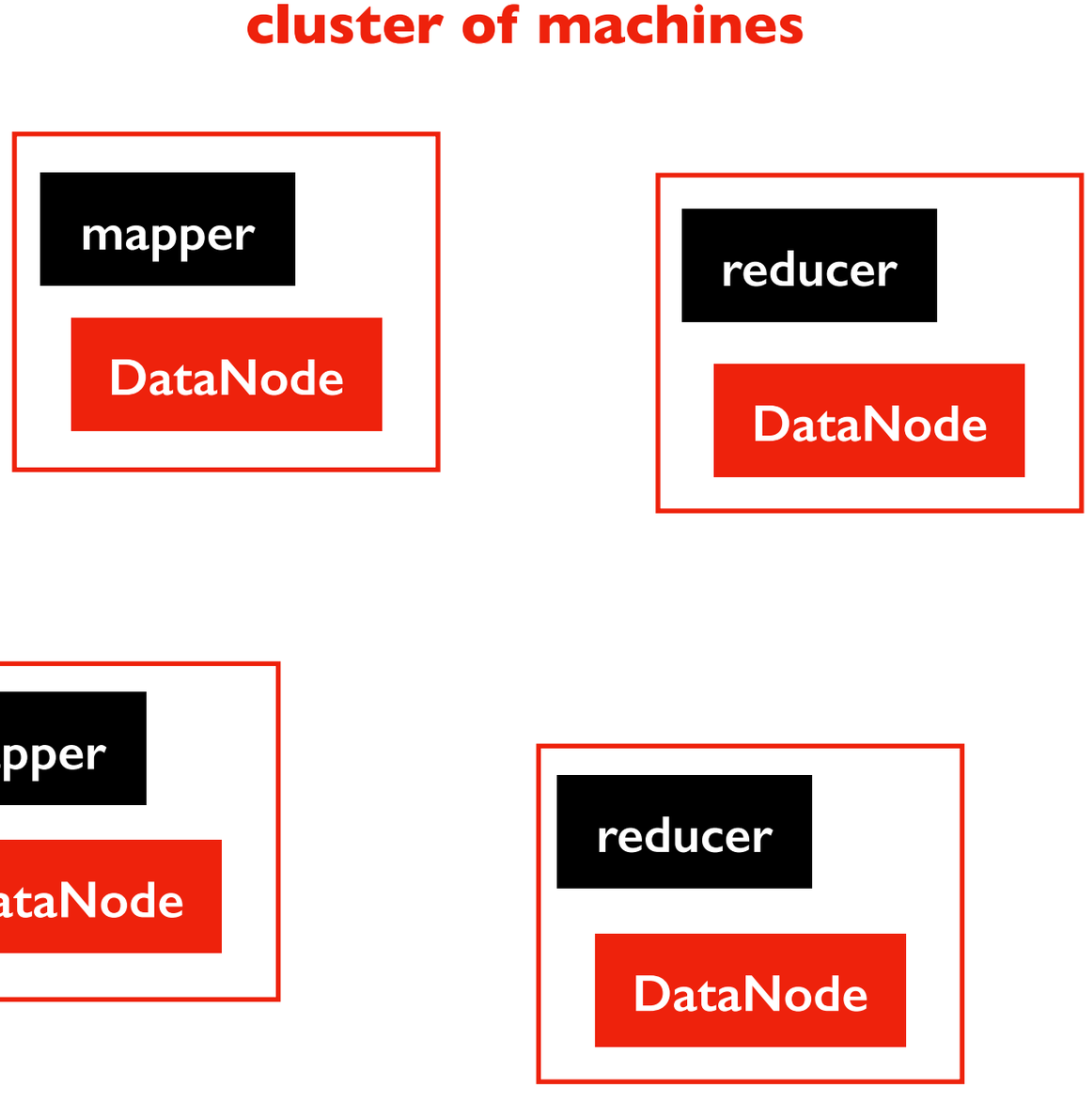
Projects like **HiveQL** try to make MapReduce more accessible.

Data Locality: Avoid Network Transfers

Run on same machines

- HDFS DataNodes
- MapReduce executor

Try to run mappers on machine where DataNode has needed data. Uses disk but not network.



Summary: Some Key Ideas

To build complex systems...

- compose layers of subsystems

To scale out...

- partition your data

To handle faults...

- replicate your data

To optimize I/O...

- pipeline writes
- co-locate computation (MapReduce) with data (HDFS)