[544] Hadoop Ecosystem

Tyler Caraza-Harter

Learning Objectives

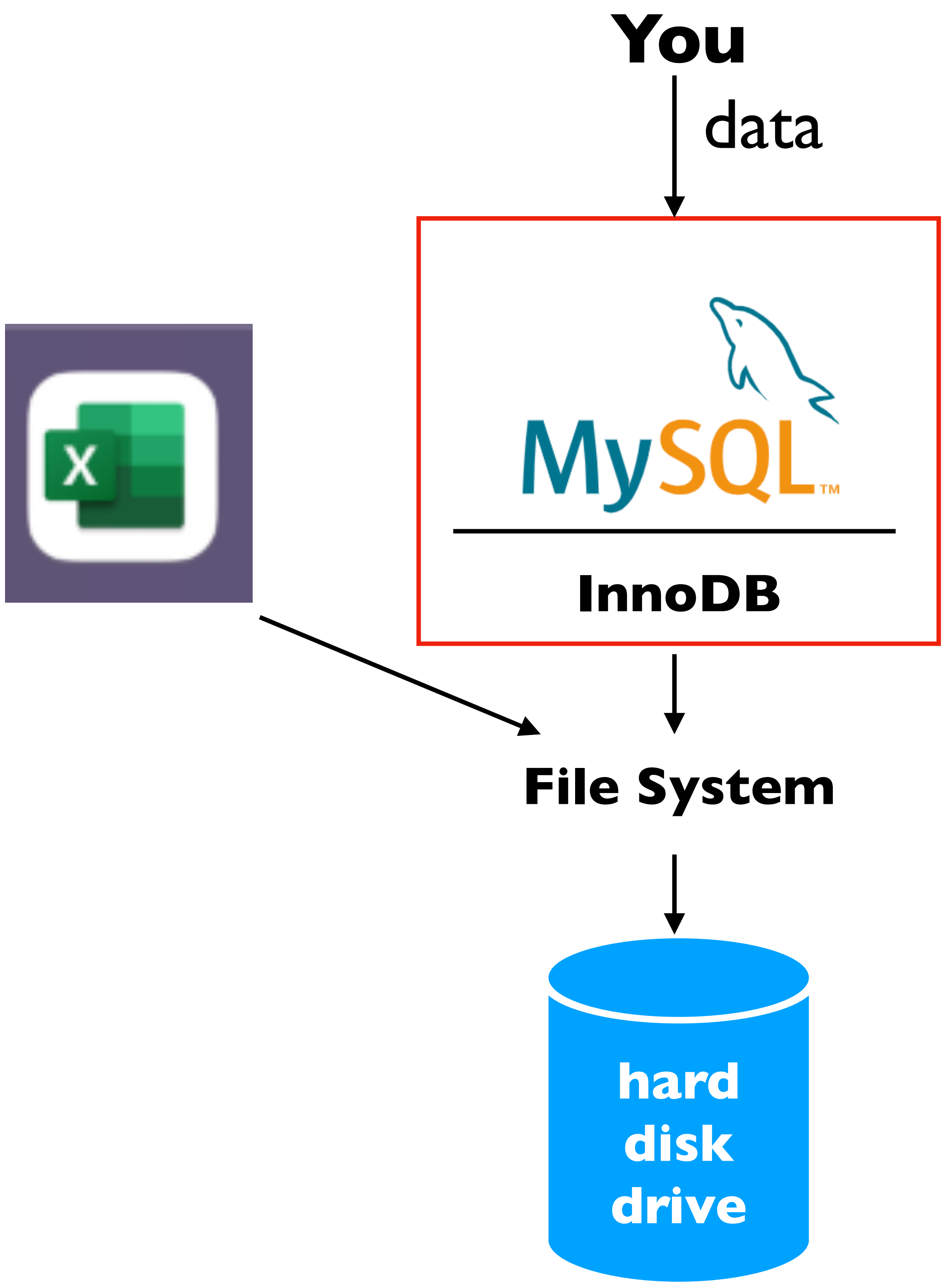
- describe the purpose of GFS, MapReduce, and BigTable (at a high level), and similar Hadoop systems (HDFS, Spark, and Cassandra)
- describe partitioning and replication and the motivation for each technique
- identify the role that clients, NameNodes, and DataNodes play for HDFS reads and writes

Outline: Hadoop Ecosystem

Motivation, Hadoop Ecosystem

Hadoop File System (HDFS)

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



You

data

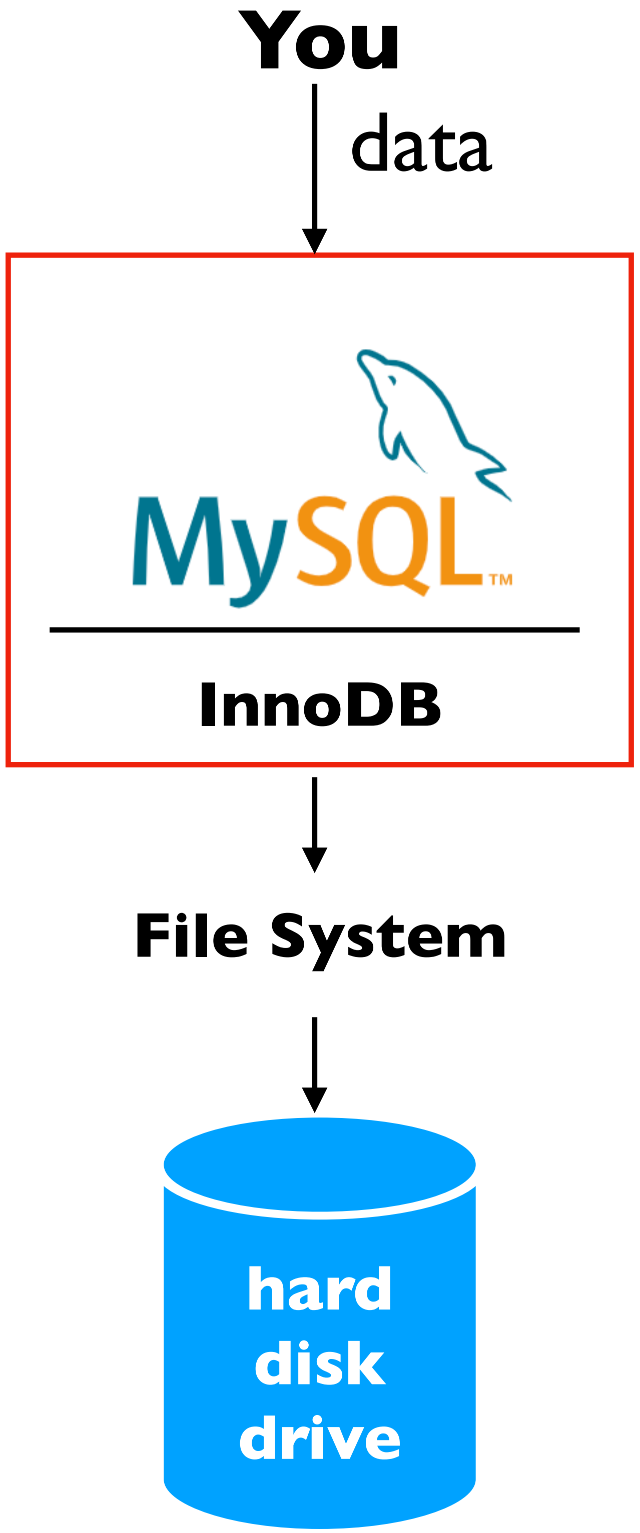
InnoDB

File System

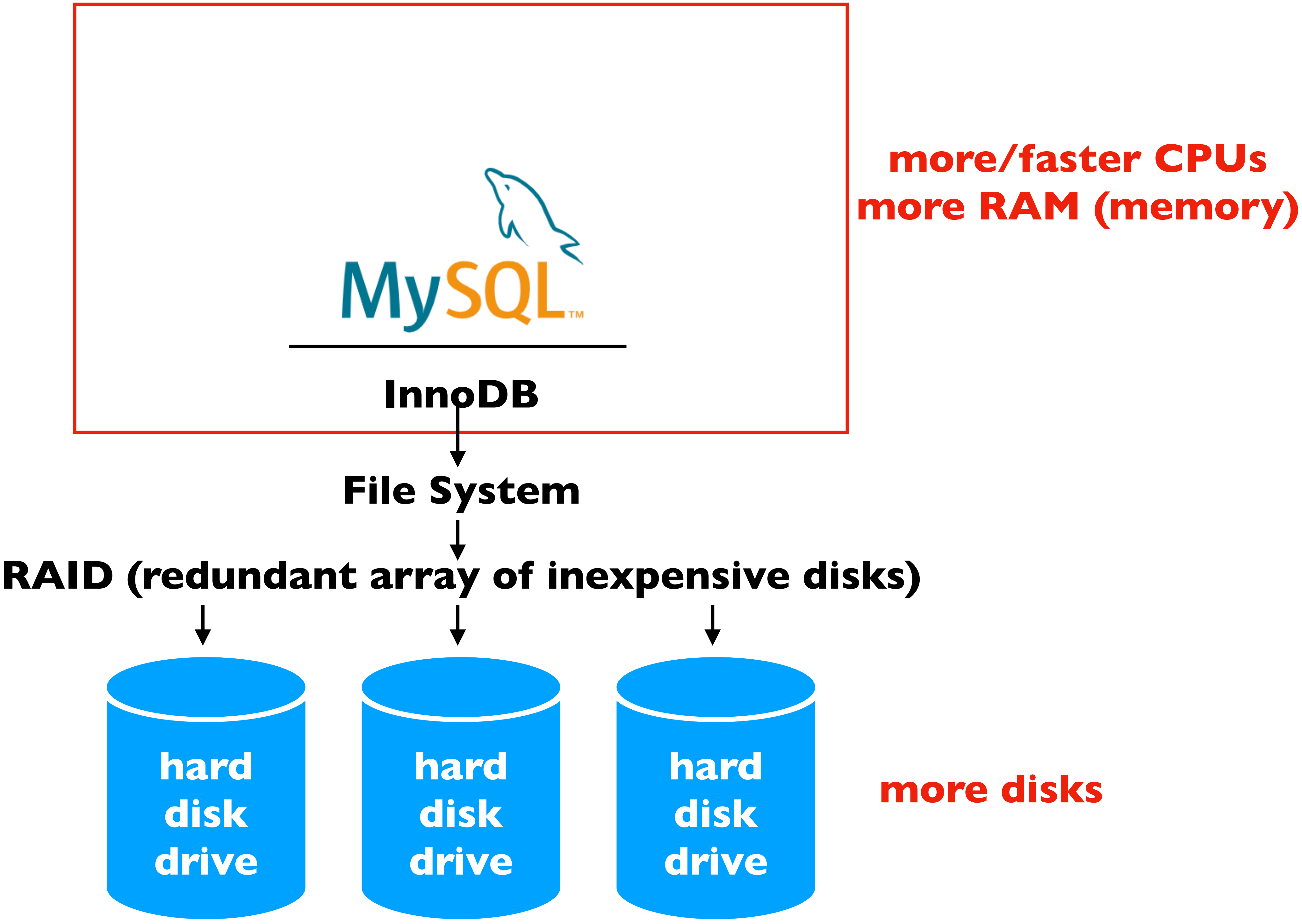
hard disk drive

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)



MySQL

InnoDB

File System

RAID (redundant array of inexpensive disks)

hard disk drive

hard disk drive

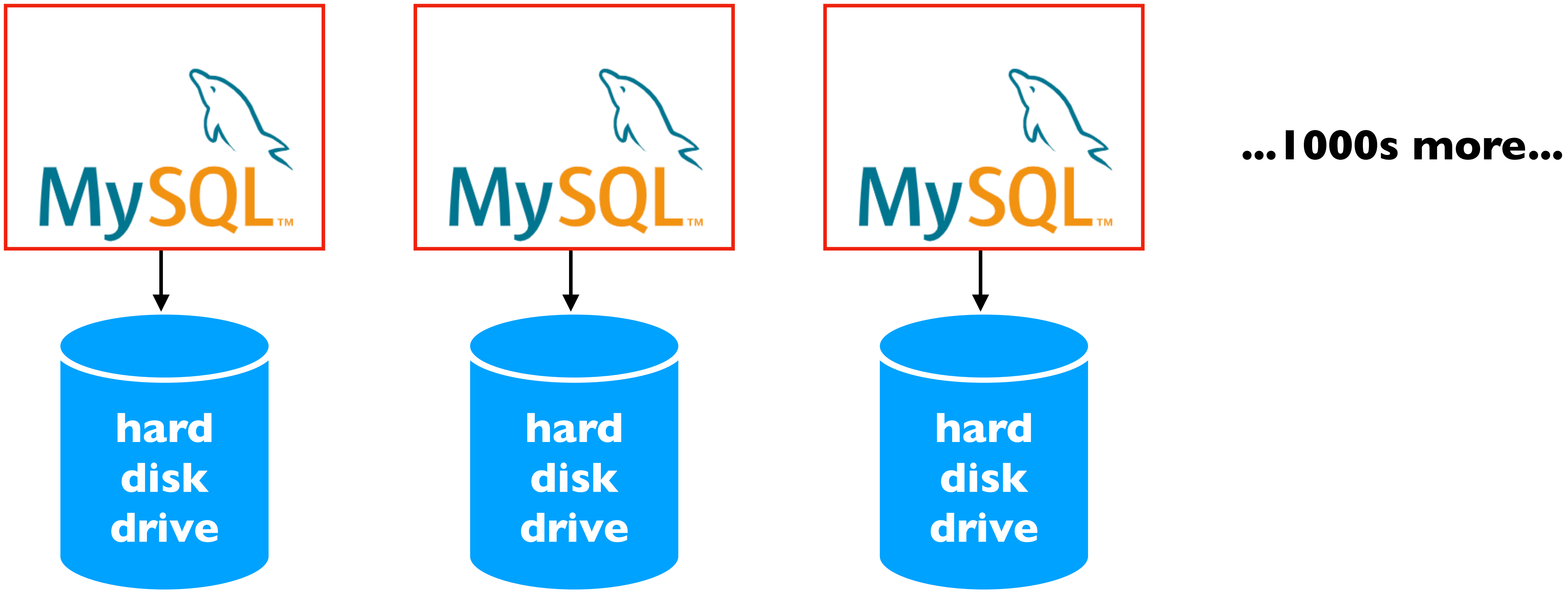
hard disk drive

more/faster CPUs
more RAM (memory)

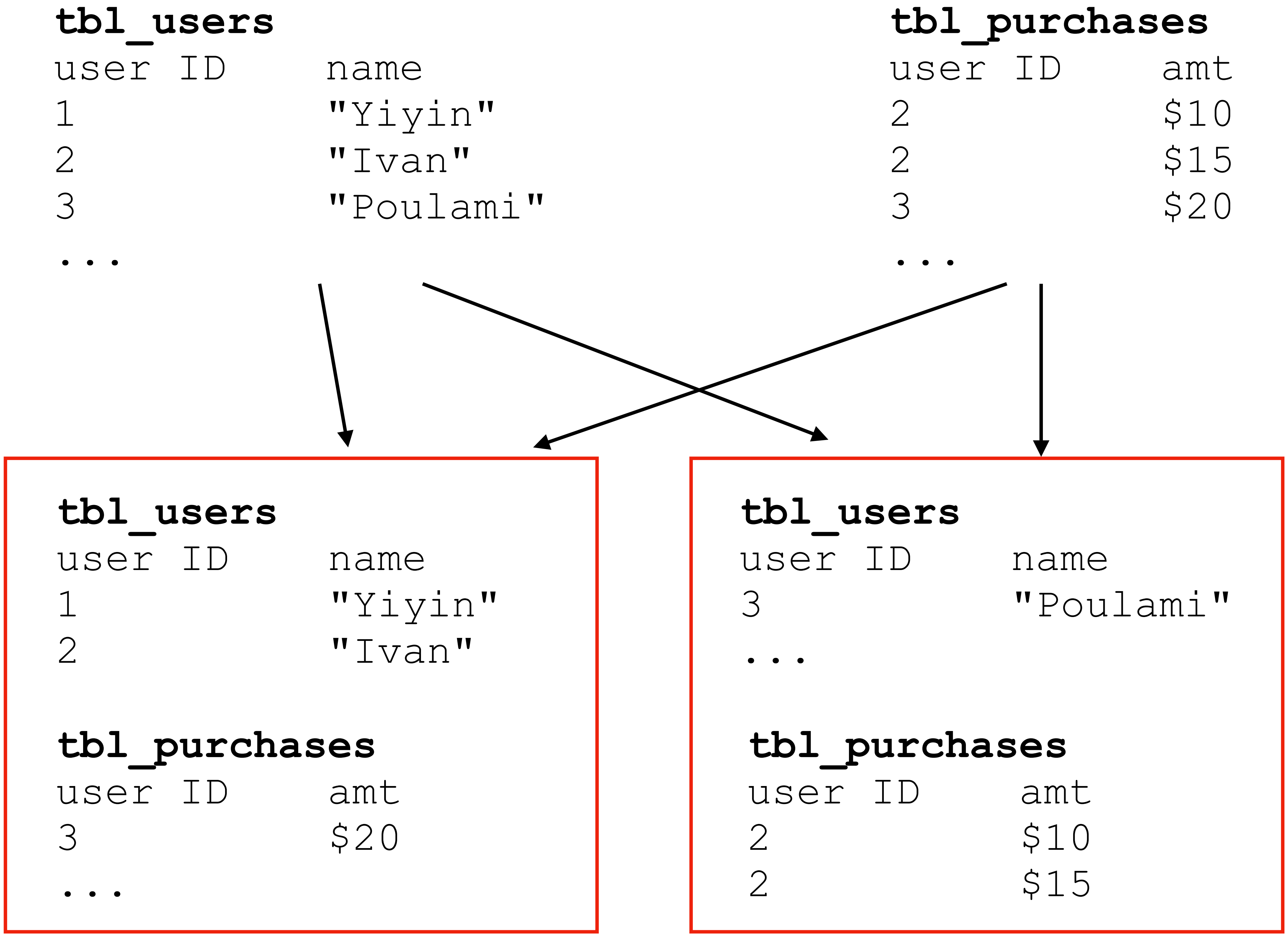
more disks

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

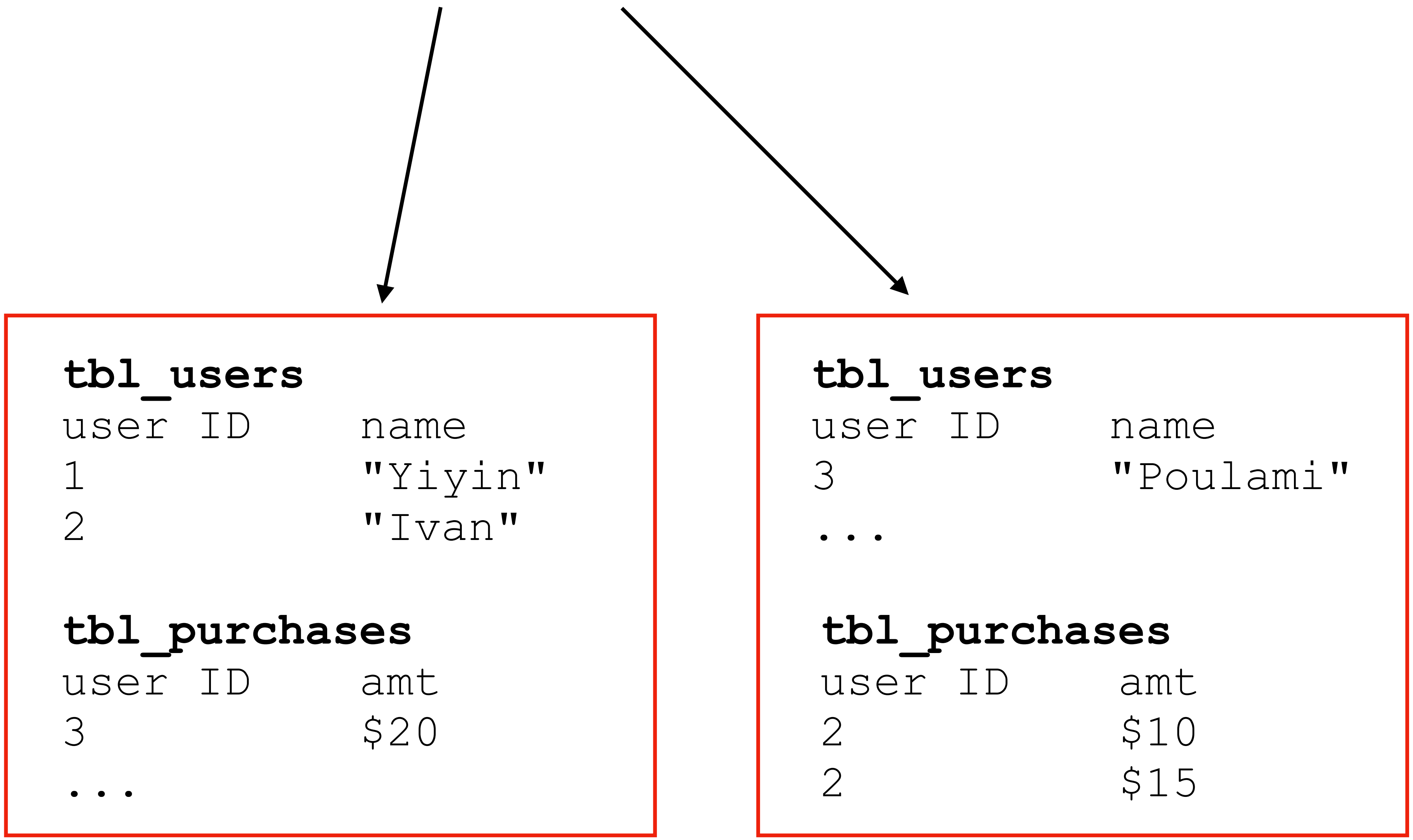
user ID	name
3	"Poulami"
...	

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

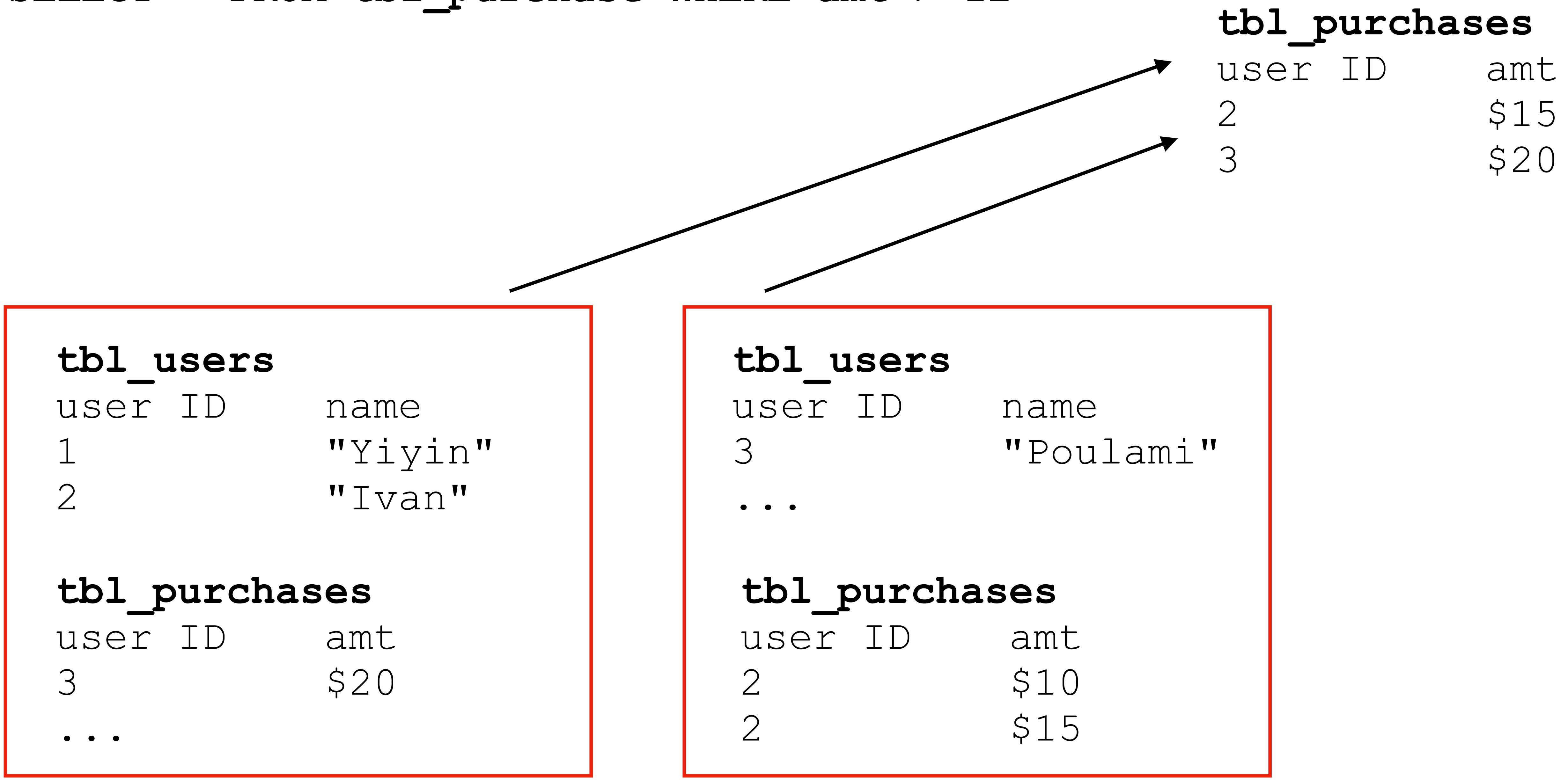
user ID	name
3	"Poulami"
...	

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

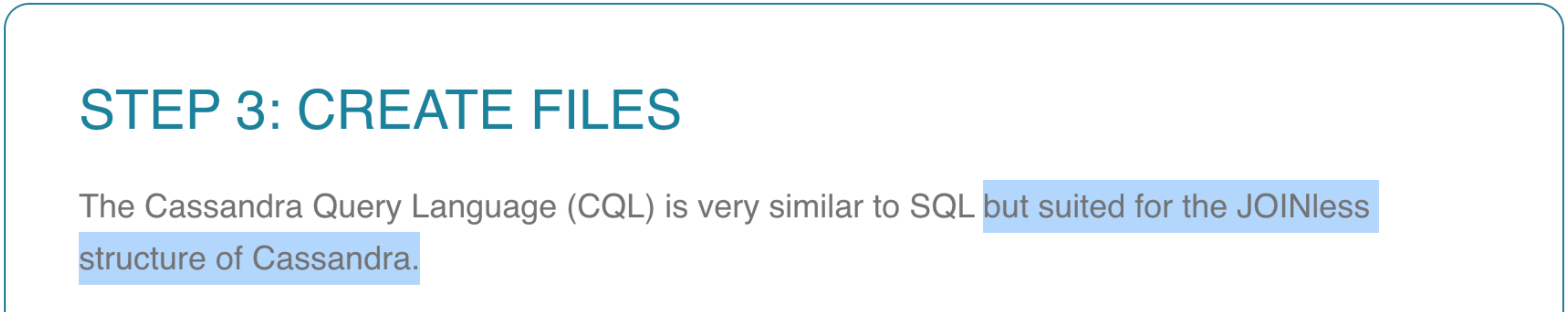
user ID	name
3	"Poulami"
...	

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



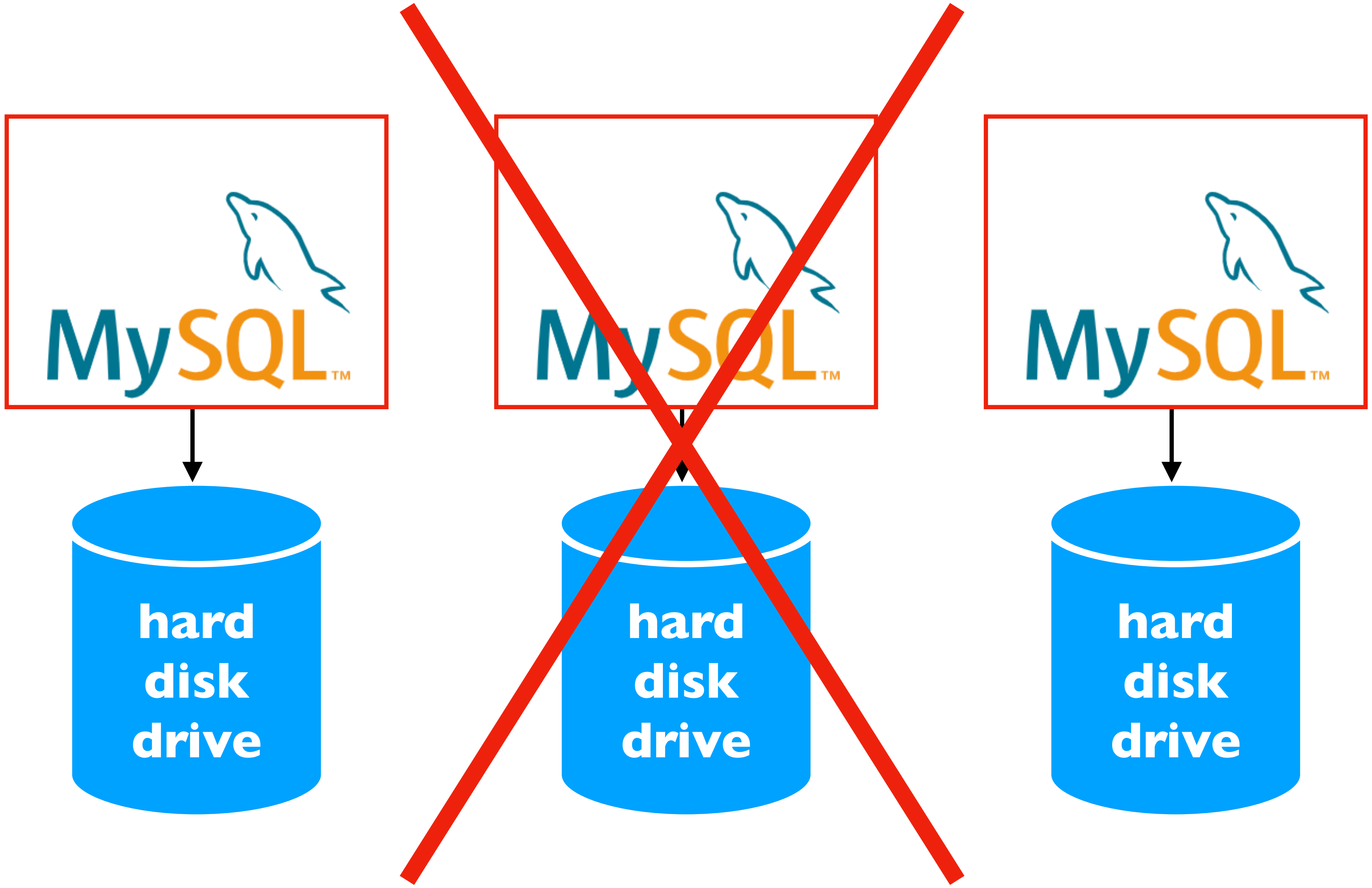
STEP 3: CREATE FILES

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



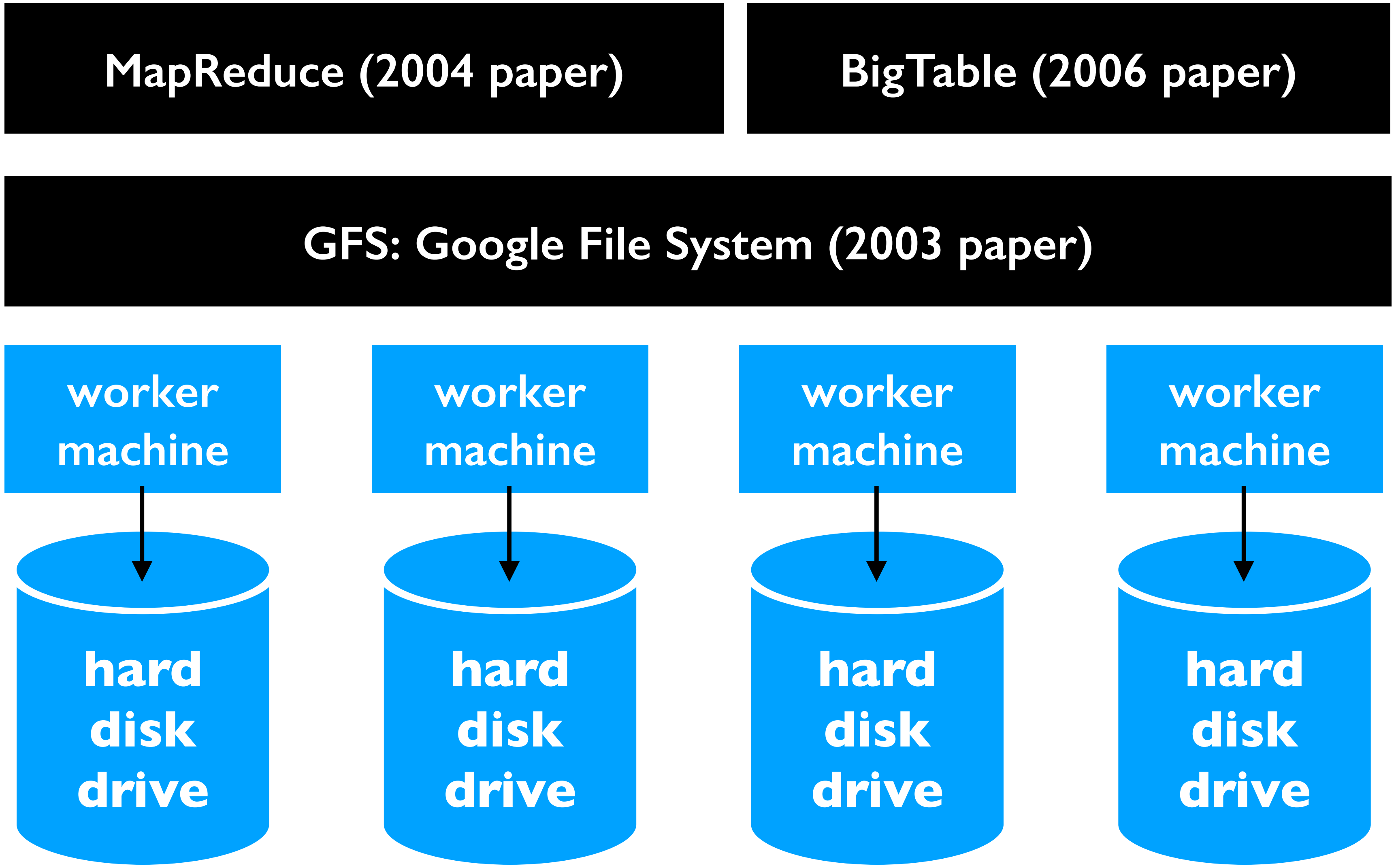
...1000s more...

Motivation for System Redesign

Features

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

Google Architecture



MapReduce (2004 paper)

BigTable (2006 paper)

GFS: Google File System (2003 paper)

worker machine

worker machine

worker machine

worker machine

hard disk drive

hard disk drive

hard disk drive

hard disk drive

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	
Distributed Analytics	MapReduce	Hadoop MapReduce	Spark
Distributed Database	BigTable	HBase	Cassandra

Ecosystem: Ambari, Avro, Cassandra, Chukwa, HBase, Hive, Mahout, Ozone, Pig, Spark, Submarine, Tez, ZooKeeper

https://hadoop.apache.org/

Outline: Hadoop Ecosystem

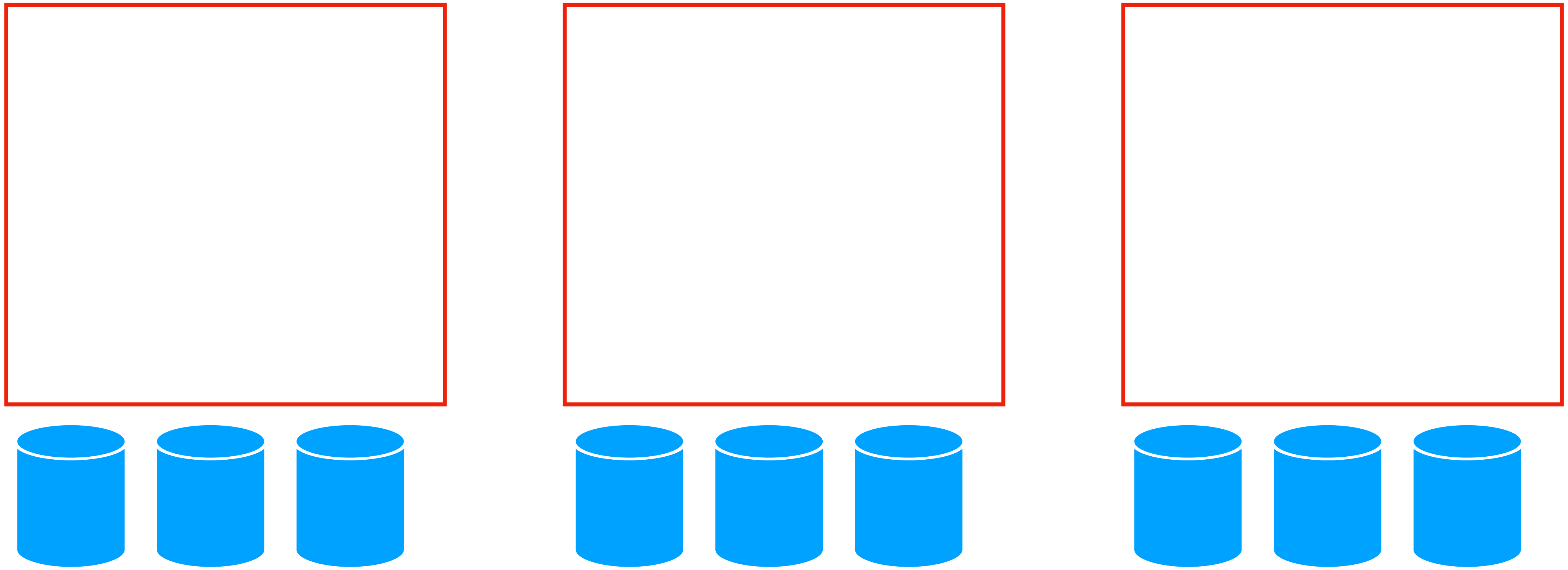
Motivation, Hadoop Ecosystem

Hadoop File System (HDFS)

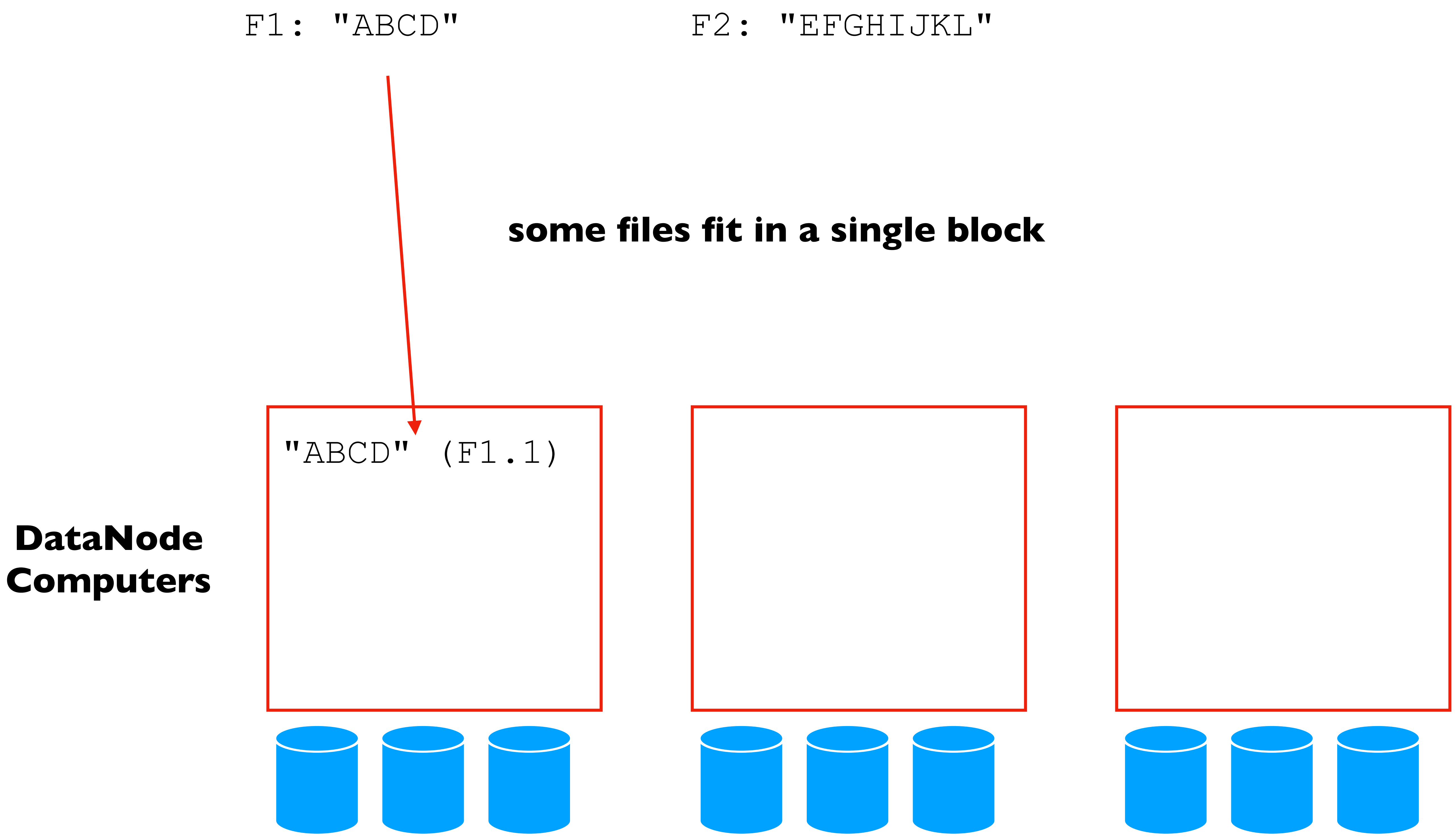
HDFS: DataNodes store File Blocks

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

DataNode Computers



HDFS: DataNodes store File Blocks



F1: "ABCD"

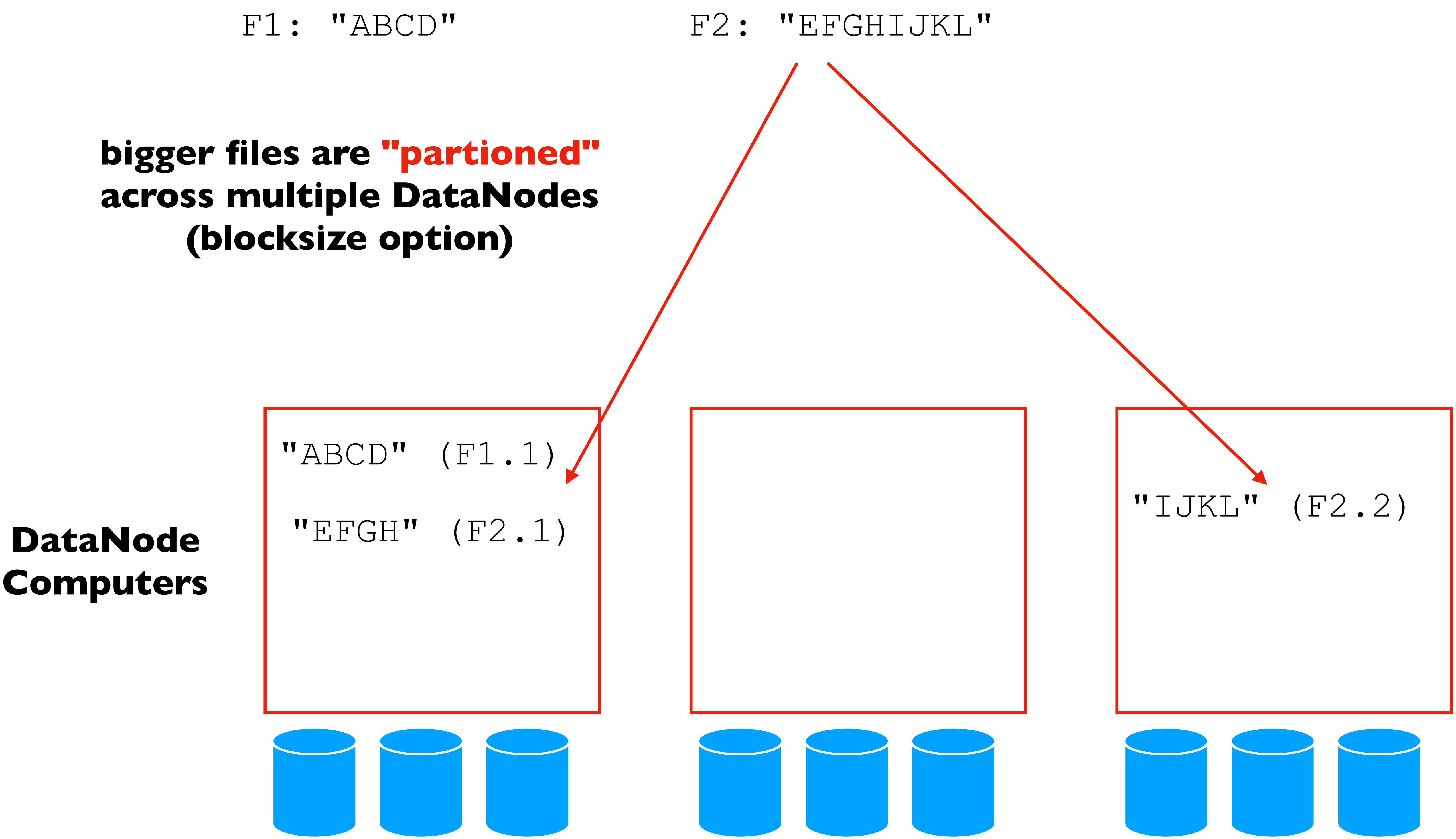
F2: "EFGHIJKL"

some files fit in a single block

"ABCD" (F1.1)

DataNode Computers

Partitioning Across DataNodes



F1: "ABCD"

F2: "EFGHIJKL"

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

DataNode Computers

"ABCD" (F1.1)

"EFGH" (F2.1)

"IJKL" (F2.2)

Replication Across DataNodes

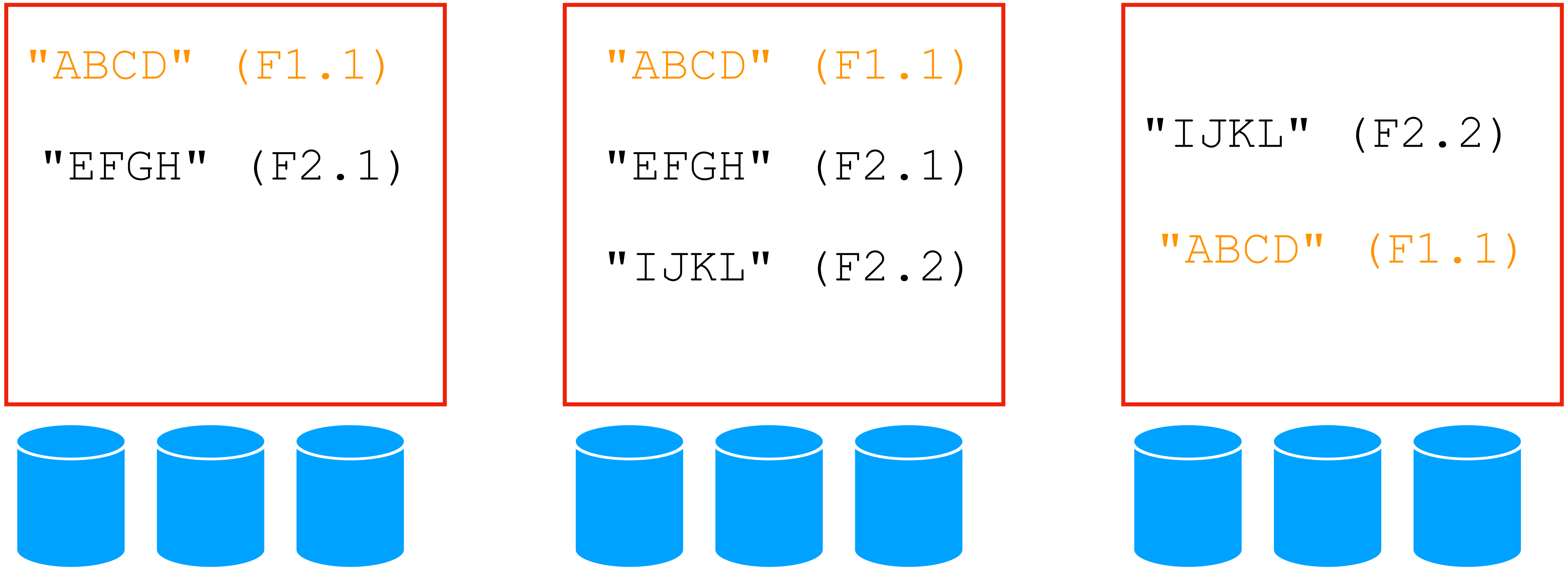
F1: "ABCD"

3x replication

F2: "EFGHIJKL"

2x relpication

DataNode Computers



"ABCD" (F1.1)

"EFGH" (F2.1)

"ABCD" (F1.1)

"EFGH" (F2.1)

"IJKL" (F2.2)

"IJKL" (F2.2)

"ABCD" (F1.1)

Replication Across DataNodes

F1: "ABCD"

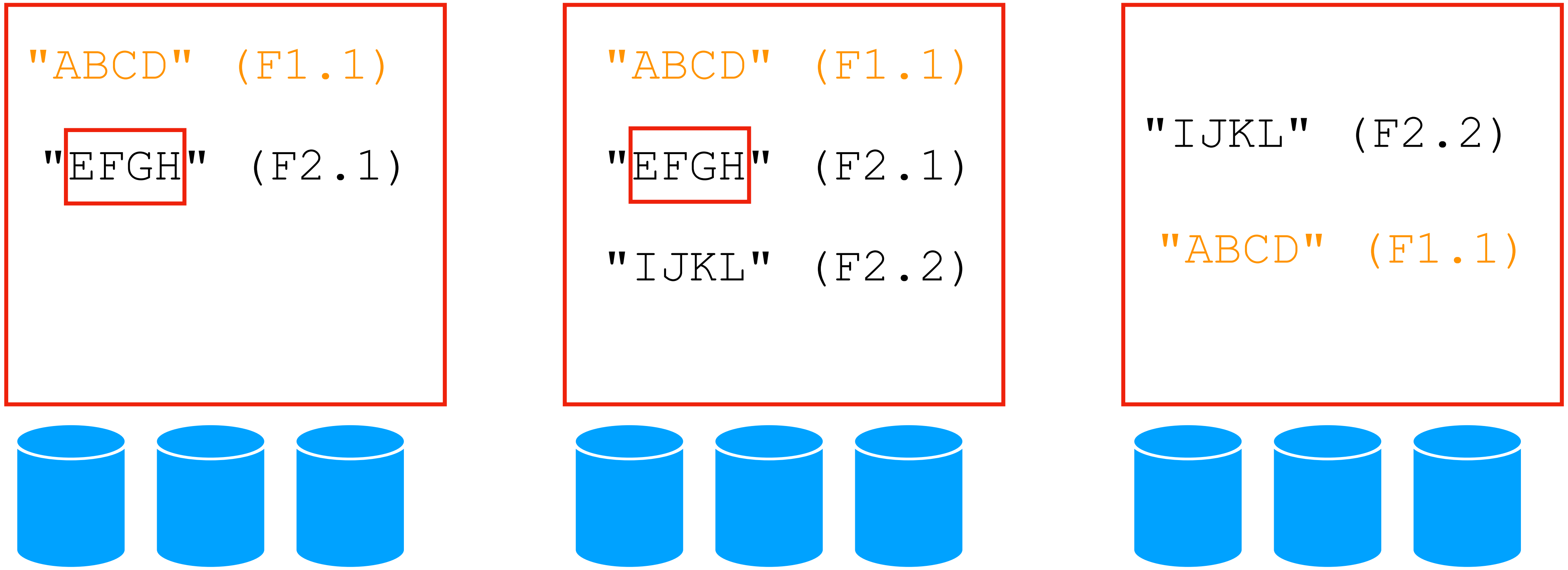
3x replication

F2: "EFGHIJKL"

2x relpication

logical vs. physical blocks

DataNode Computers



"ABCD" (F1.1)

"EFGH" (F2.1)

"ABCD" (F1.1)

"EFGH" (F2.1)

"IJKL" (F2.2)

"IJKL" (F2.2)

"ABCD" (F1.1)

Replication Across DataNodes

`F1: "ABCD"`

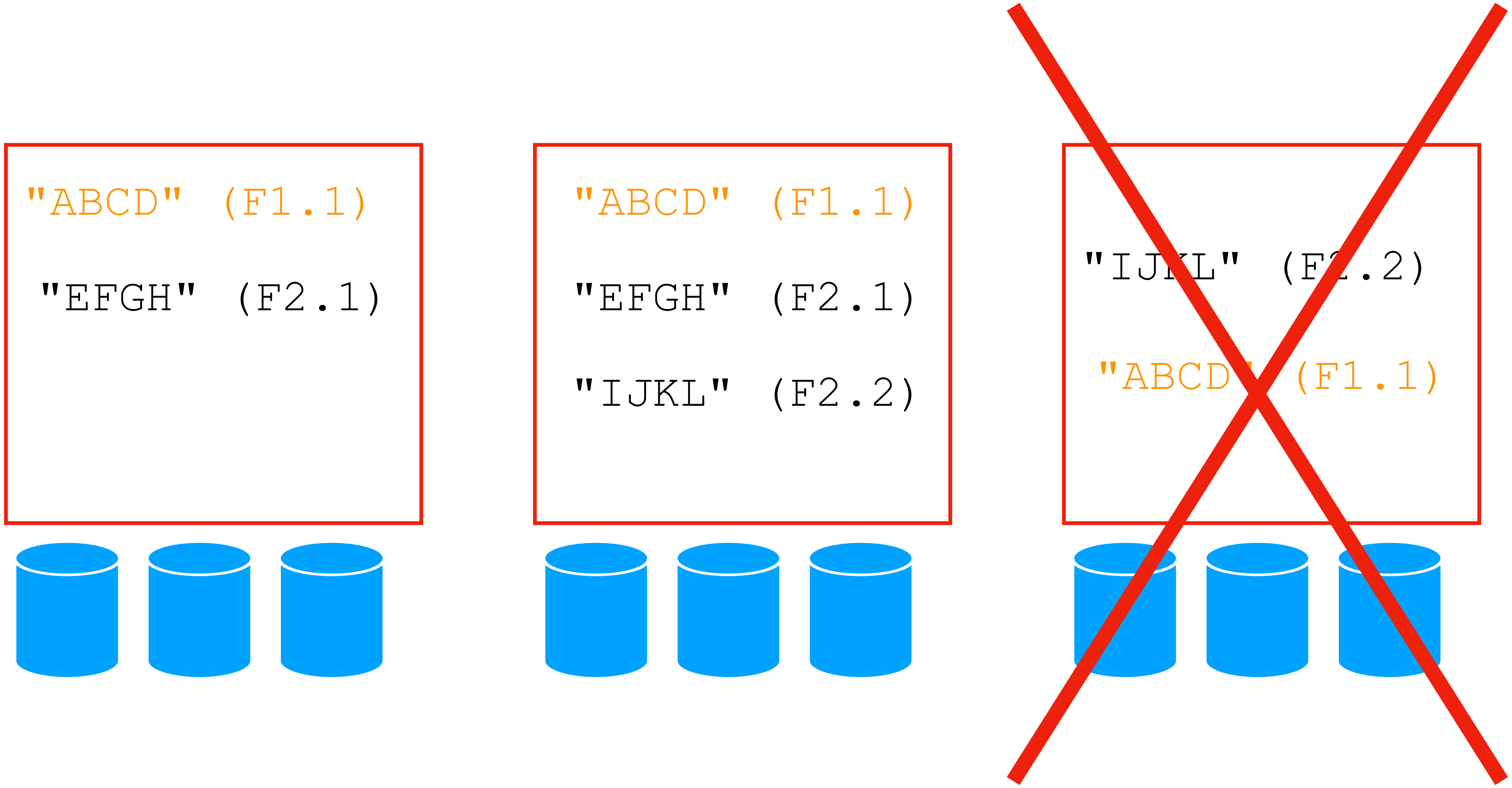
3x replication

`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



```
"ABCD" (F1.1)
"EFGH" (F2.1)
```

```
"ABCD" (F1.1)
"EFGH" (F2.1)
"IJKL" (F2.2)
```

```
"IJKL" (F2.2)
"ABCD" (F1.1)
```

Aside: Replication vs. Erasure Encoding

HDFS Strategies for handling node failure

Replication

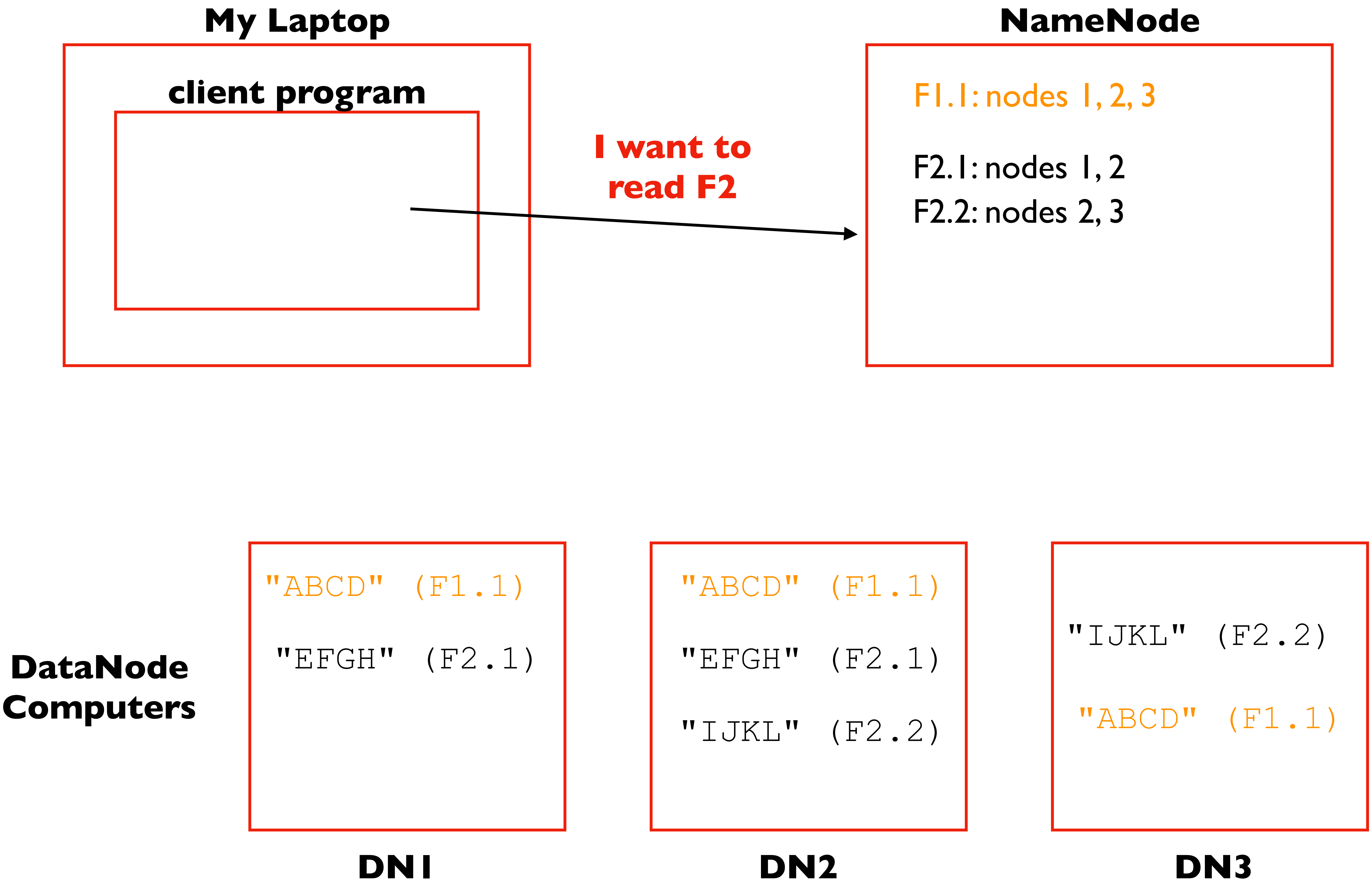
- original strategy, used for new/hot data
- covered in CS 544

Erasure Encoding

- more space efficient, less I/O efficient
- recent HDFS feature used for cold data (NOT covered in CS 544)

https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html

Boss (NameNode)/Worker Architecture



My Laptop

client program

I want to read F2

NameNode

F1.1: nodes 1, 2, 3

F2.1: nodes 1, 2

F2.2: nodes 2, 3

DataNode Computers

"ABCD" (F1.1)

"EFGH" (F2.1)

DN1

"ABCD" (F1.1)

"EFGH" (F2.1)

"IJKL" (F2.2)

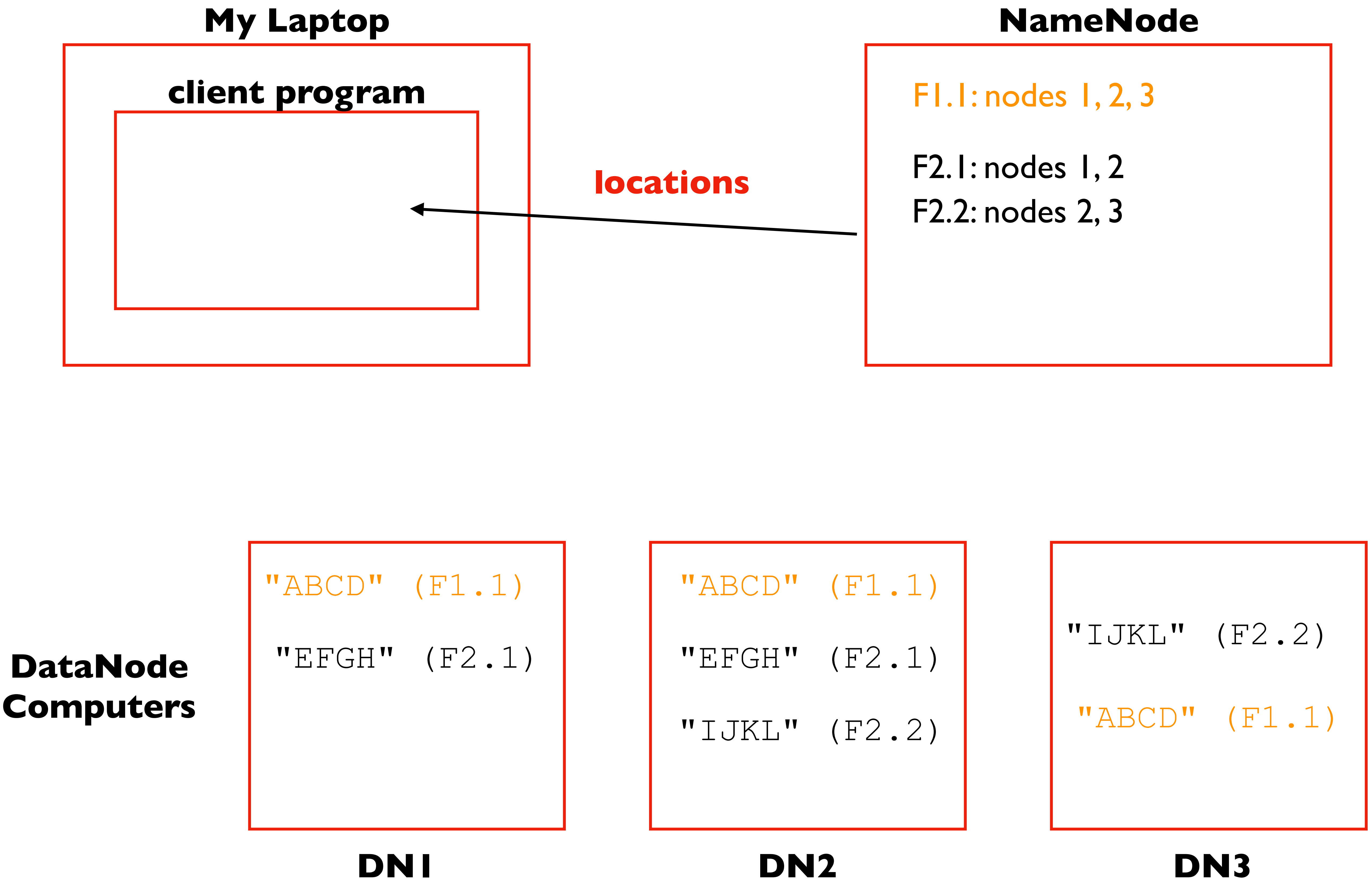
DN2

"IJKL" (F2.2)

"ABCD" (F1.1)

DN3

Boss (NameNode)/Worker Architecture



My Laptop

client program

locations

NameNode

F1.1: nodes 1, 2, 3

F2.1: nodes 1, 2

F2.2: nodes 2, 3

DataNode Computers

"ABCD" (F1.1)

"EFGH" (F2.1)

DN1

"ABCD" (F1.1)

"EFGH" (F2.1)

"IJKL" (F2.2)

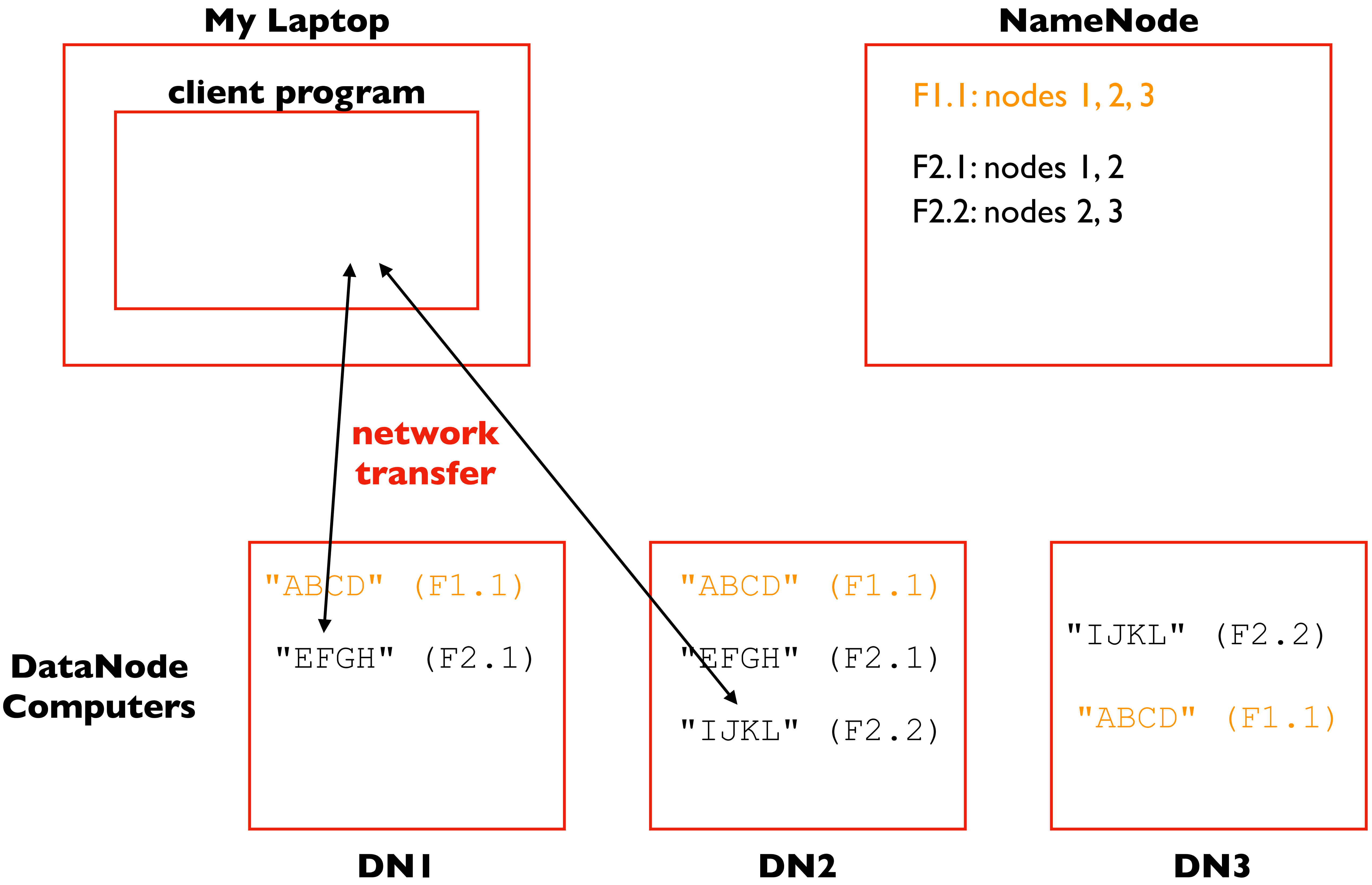
DN2

"IJKL" (F2.2)

"ABCD" (F1.1)

DN3

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

"ABCD" (F1.1)

"EFGH" (F2.1)

DN1

"ABCD" (F1.1)

"EFGH" (F2.1)

"IJKL" (F2.2)

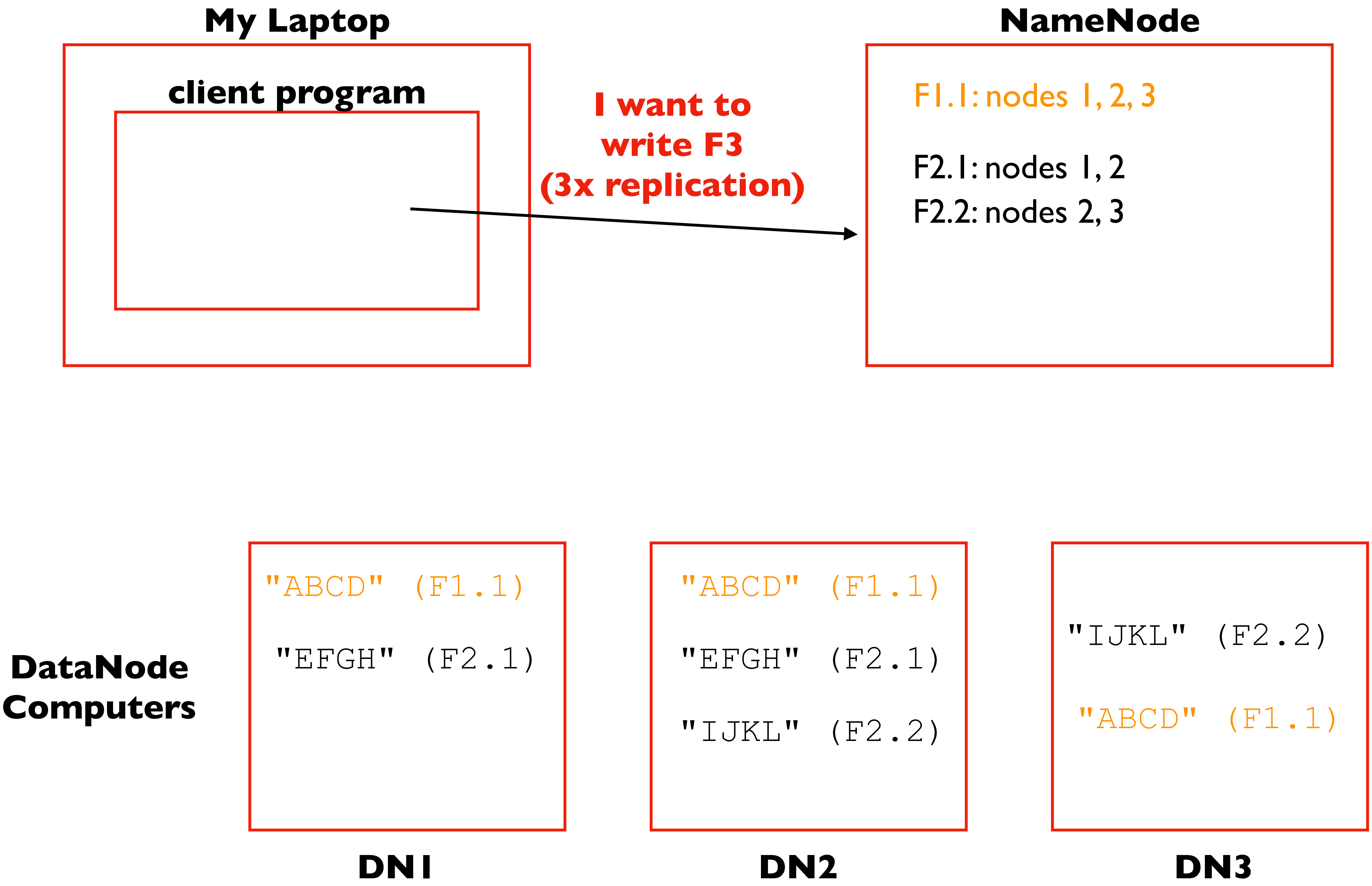
DN2

"IJKL" (F2.2)

"ABCD" (F1.1)

DN3

Boss (NameNode)/Worker Architecture



My Laptop

client program

I want to write F3 (3x replication)

NameNode

F1.1: nodes 1, 2, 3

F2.1: nodes 1, 2

F2.2: nodes 2, 3

DataNode Computers

DN1

```
"ABCD" (F1.1)
"EFGH" (F2.1)
```

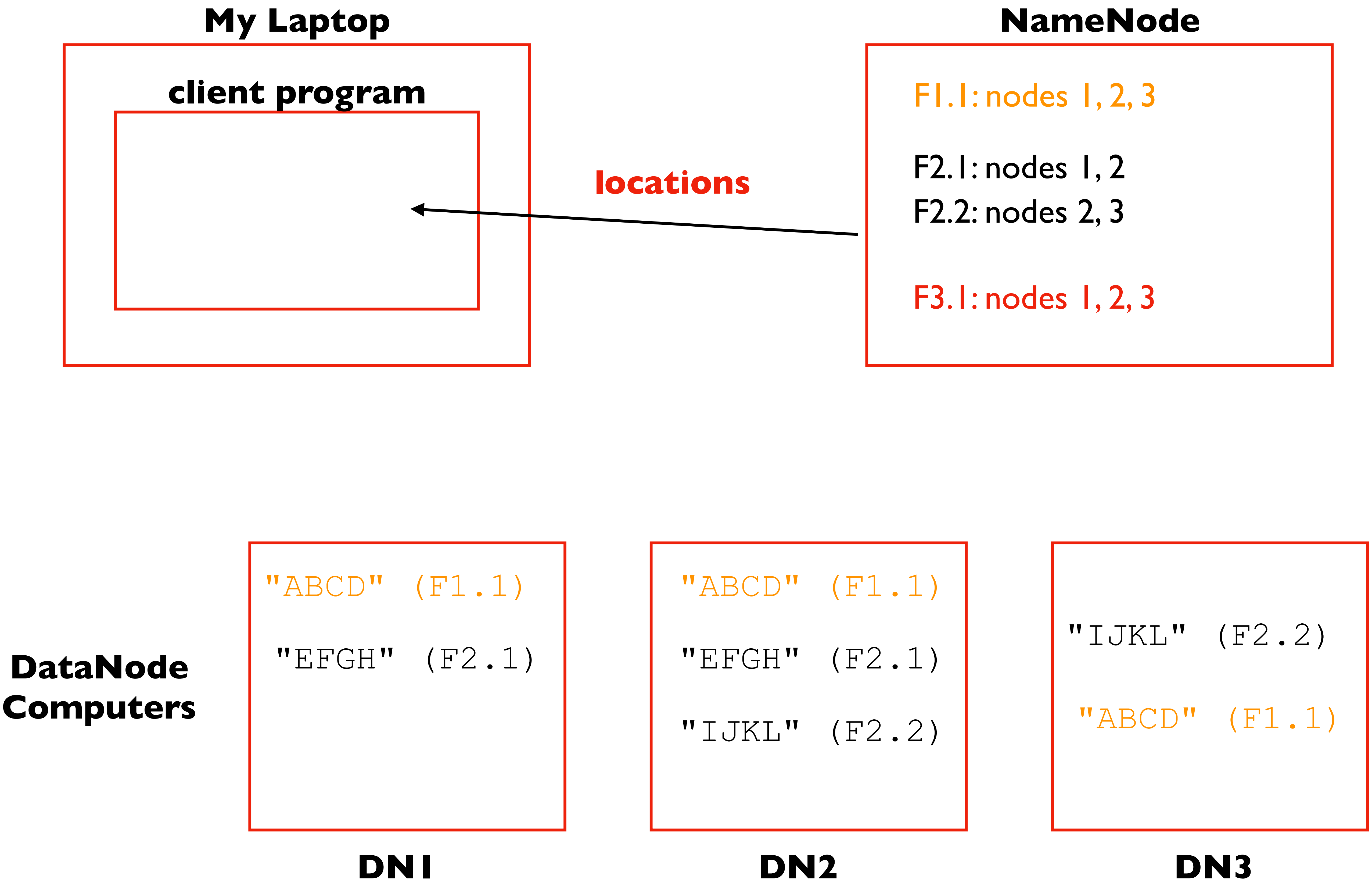
DN2

```
"ABCD" (F1.1)
"EFGH" (F2.1)
"IJKL" (F2.2)
```

DN3

```
"IJKL" (F2.2)
"ABCD" (F1.1)
```

Boss (NameNode)/Worker Architecture



My Laptop

client program

locations

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

"ABCD" (F1.1)

"EFGH" (F2.1)

DN1

"ABCD" (F1.1)

"EFGH" (F2.1)

"IJKL" (F2.2)

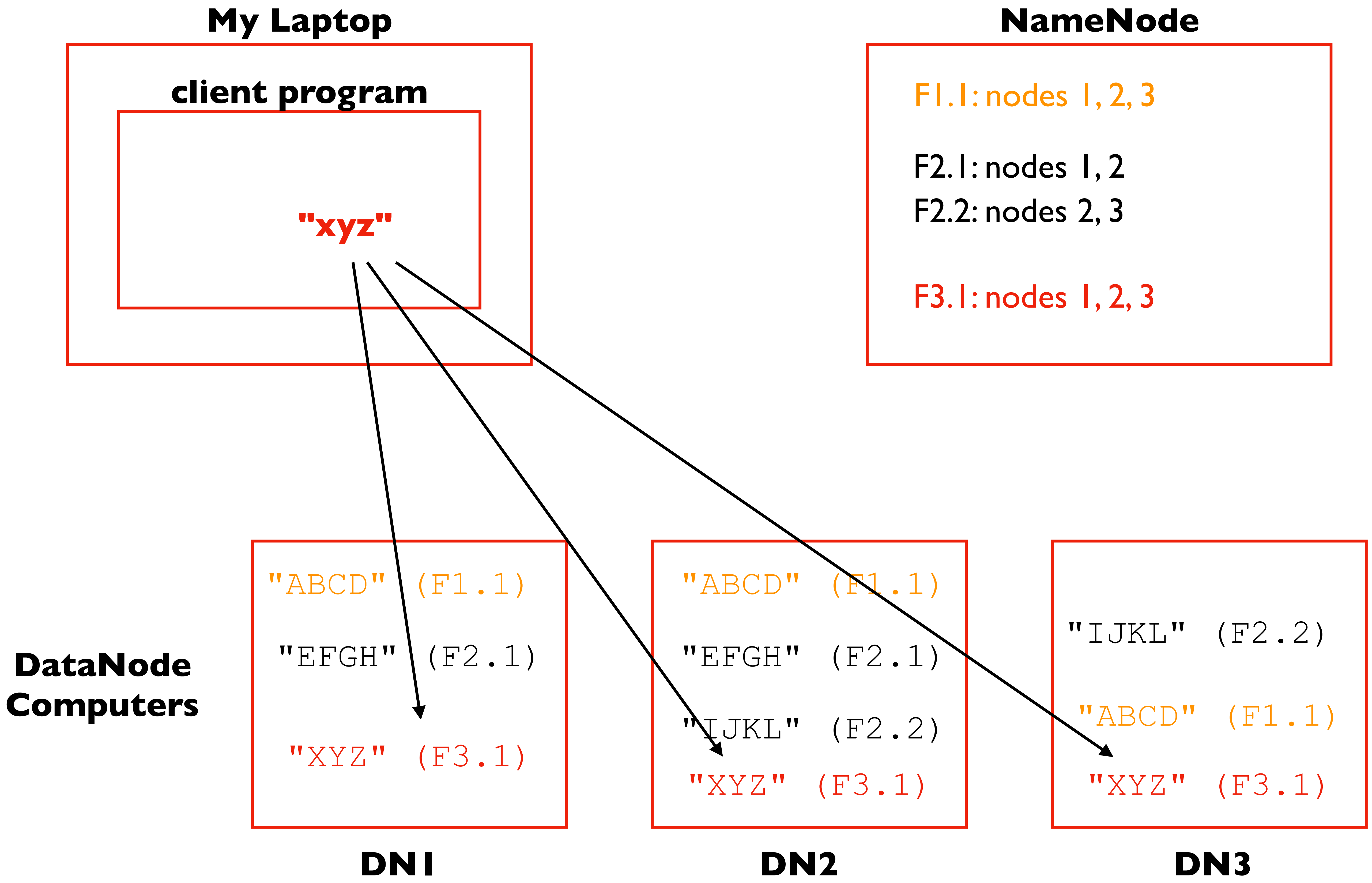
DN2

"IJKL" (F2.2)

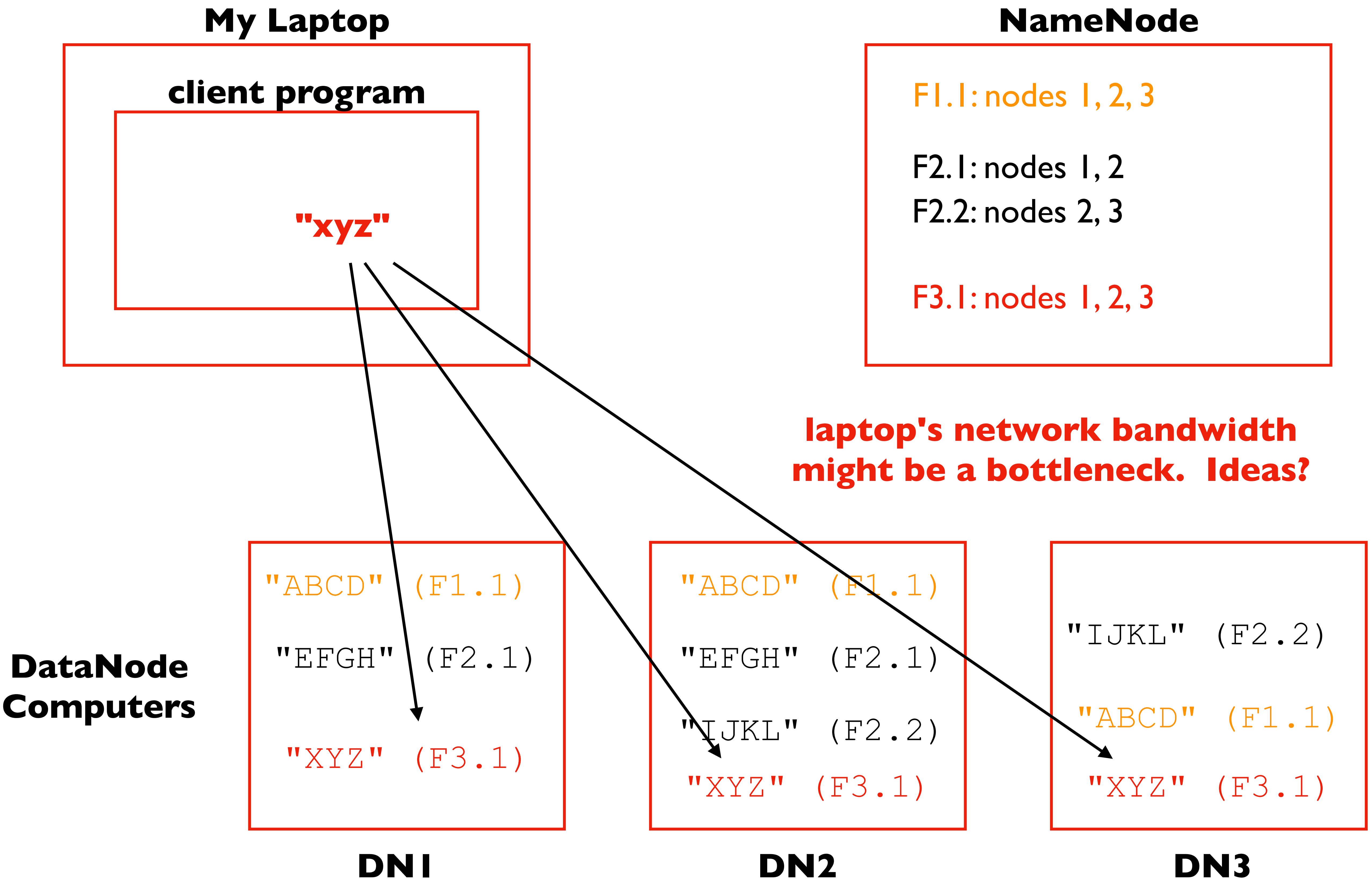
"ABCD" (F1.1)

DN3

Boss (NameNode)/Worker Architecture



Boss (NameNode)/Worker Architecture



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

"ABCD" (F1.1)
"EFGH" (F2.1)
"XYZ" (F3.1)

DN2

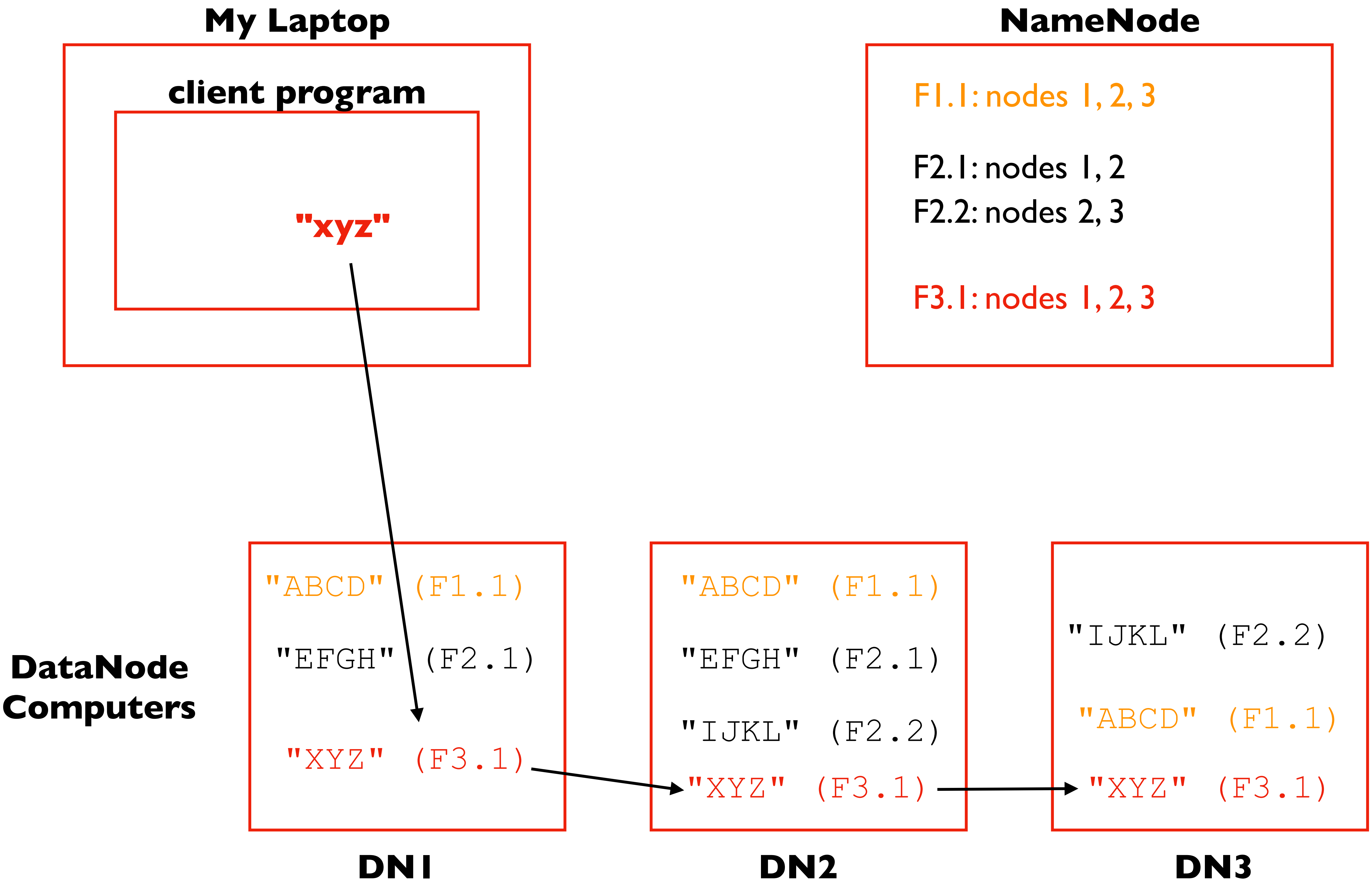
"ABCD" (F1.1)
"EFGH" (F2.1)
"IJKL" (F2.2)
"XYZ" (F3.1)

DN3

"IJKL" (F2.2)
"ABCD" (F1.1)
"XYZ" (F3.1)

laptop's network bandwidth might be a bottleneck. Ideas?

Pipelined 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

```
"ABCD"  (F1.1)
"EFGH"  (F2.1)
"XYZ"  (F3.1)
```

DN1

```
"ABCD"  (F1.1)
"EFGH"  (F2.1)
"IJKL"  (F2.2)
"XYZ"  (F3.1)
```

DN2

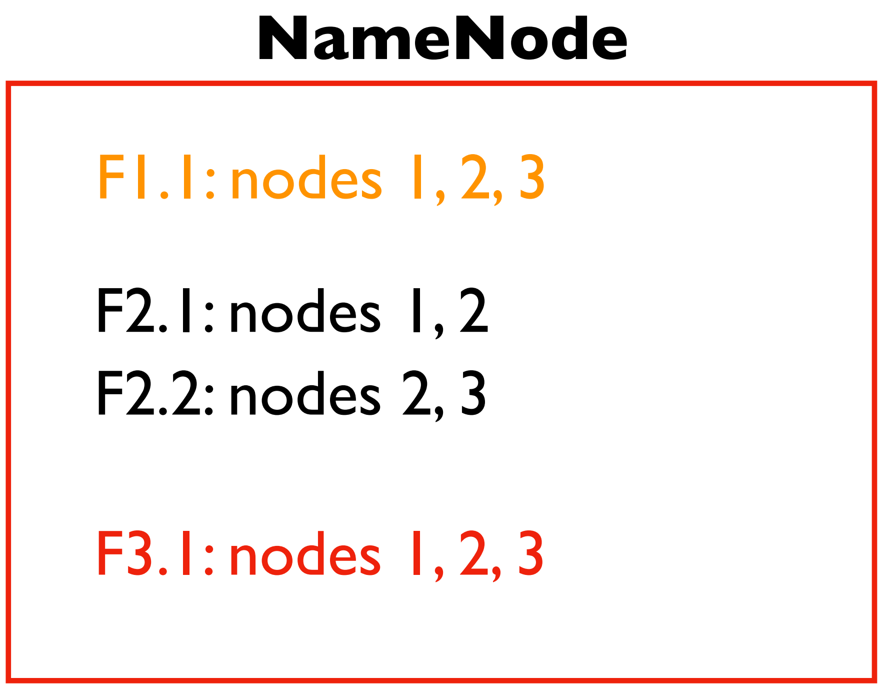
```
"IJKL"  (F2.2)
"ABCD"  (F1.1)
"XYZ"  (F3.1)
```

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?

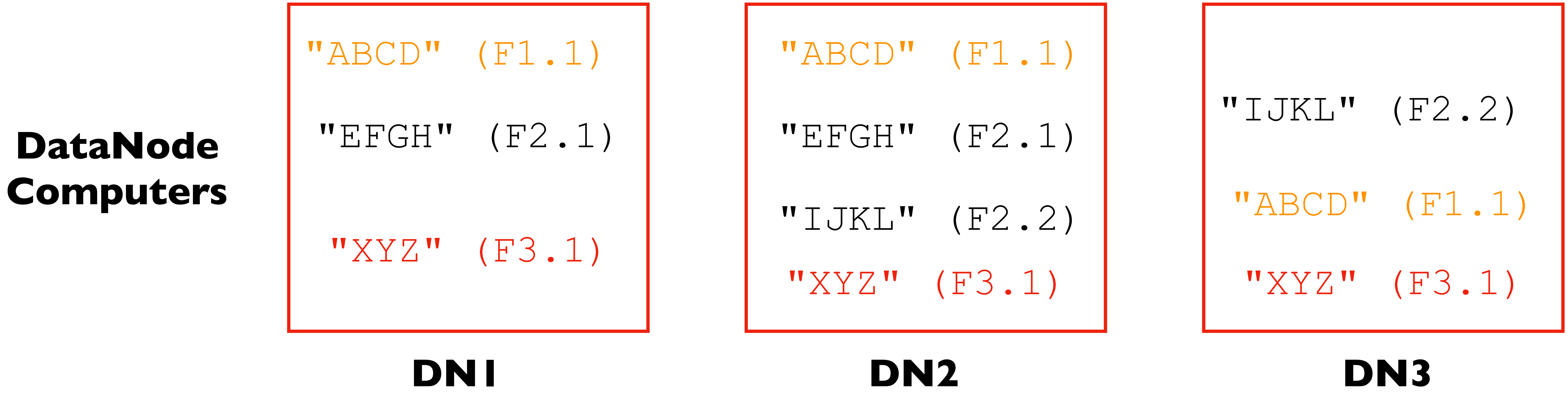


NameNode

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

```
"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)
```

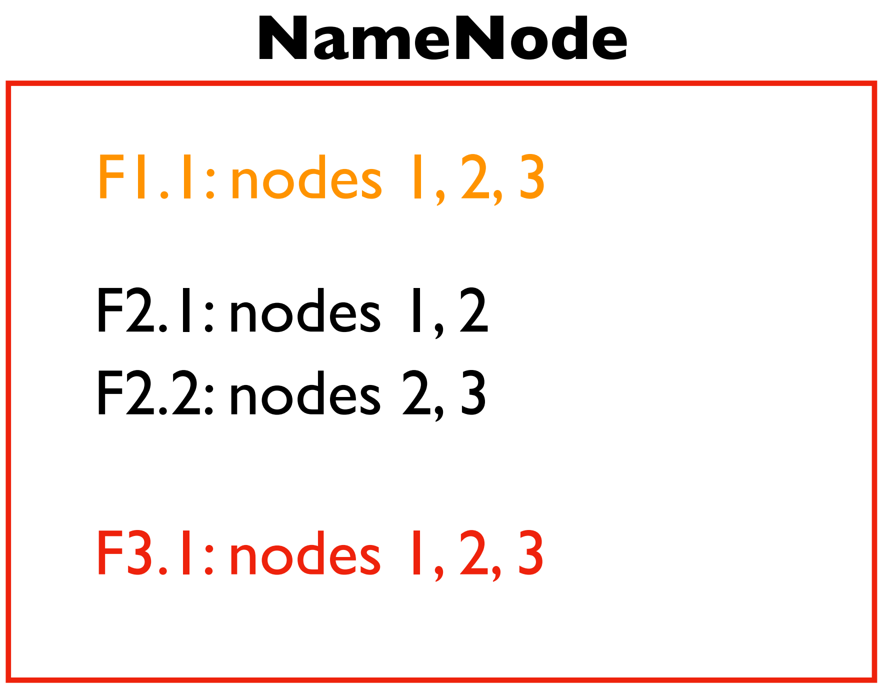
What are the tradeoffs of replication factor and block size?

benefits of high replication?

benefits of low replication?

benefits of large block size?

benefits of small block size?

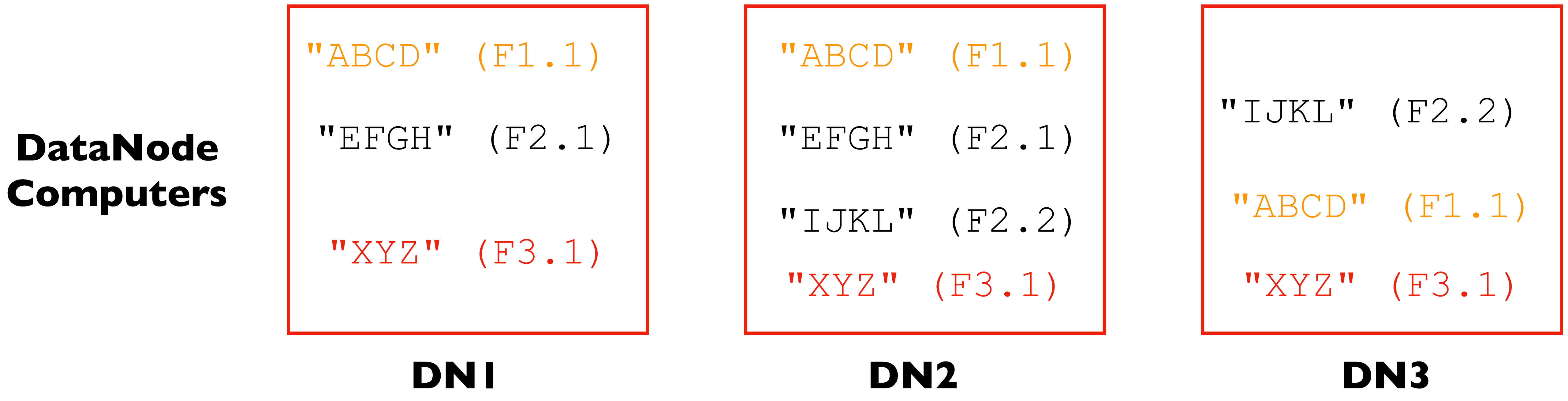


NameNode

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)

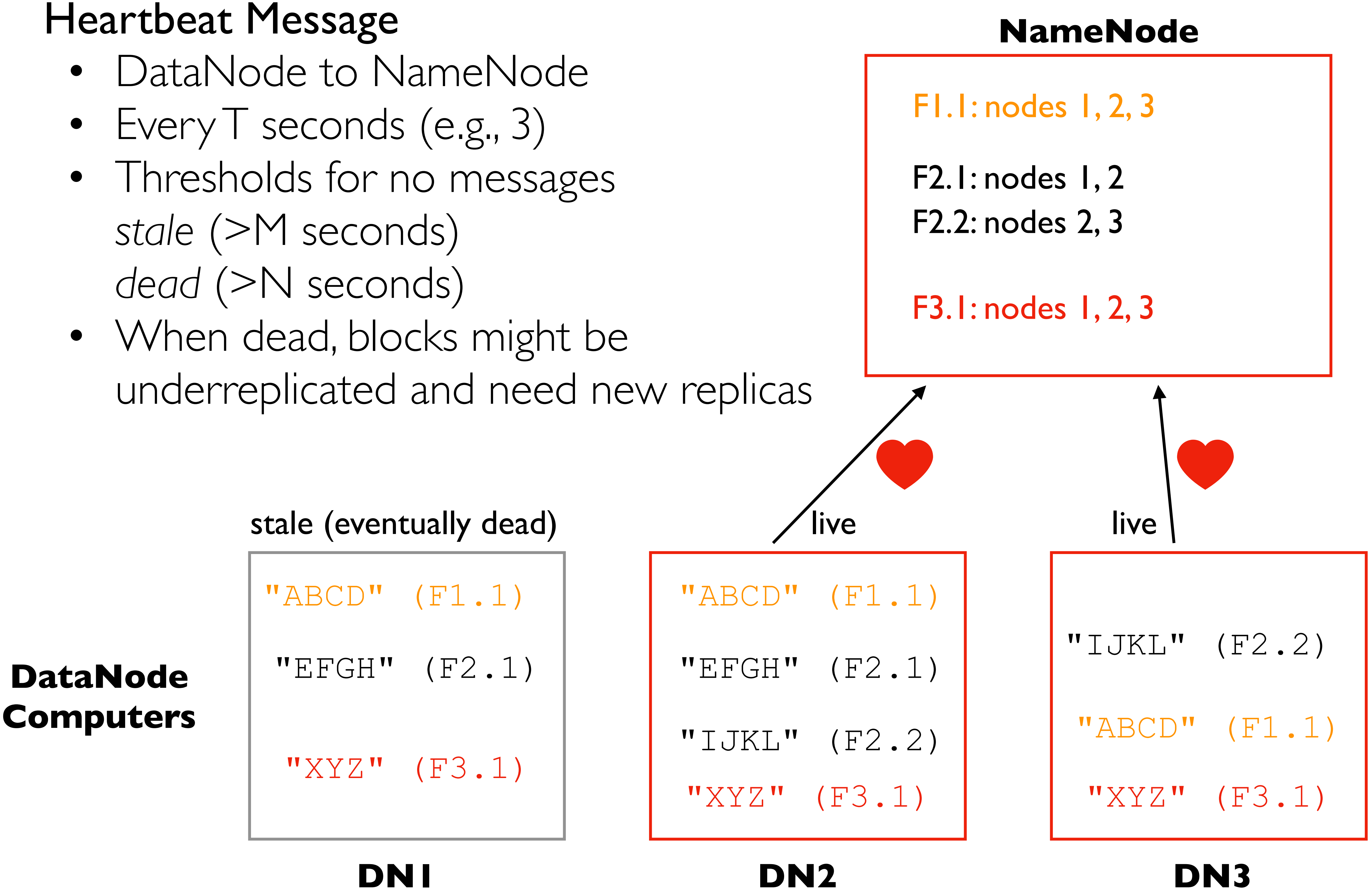
DN3

"IJKL" (F2.2)
"ABCD" (F1.1)
"XYZ" (F3.1)

How do we know when a DataNode fails?

Heartbeat Message

- DataNode to NameNode
- Every T seconds (e.g., 3)
- Thresholds for no messages
  stale (>M seconds)
  dead (>N seconds)
- When dead, blocks might be underreplicated and need new replicas



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

stale (eventually dead)

"ABCD" (F1.1)
"EFGH" (F2.1)
"XYZ" (F3.1)

DN1

live

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

DN2

live

"IJKL" (F2.2)
"ABCD" (F1.1)
"XYZ" (F3.1)

DN3

Summary: Some Key Ideas Applied to HDFS

To build complex systems...

- compose layers of subsystems

To scale out...

- partition your data

To handle faults...

- replicate your data

To detect faults...

- send heartbeats

To optimize I/O...

- pipeline writes