

# Many ways to store data

**Sébastien Ponce**

sebastien.ponce@cern.ch

**CERN**

Thematic CERN School of Computing 2015

## In the previous episode...

- We've discussed data formats
- And especially data structures
- We've talked about organising our data

## In the previous episode...

- We've discussed data formats
- And especially data structures
- We've talked about organising our data

## Today

Let's see how we store our data !

# Outline

## 1 Storage devices

- Existing devices
- Hierarchical storage

## 2 Distributed storage

- Data distribution
- Data federation

## 3 Parallelizing files' storage

- Striping
- Introduction to Map/Reduce

## 4 Conclusion

# Storage devices

- 1 Storage devices
  - Existing devices
  - Hierarchical storage
- 2 Distributed storage
- 3 Parallelizing files' storage
- 4 Conclusion

# A variety of storage devices

## Main differences

- Capacities from 1 GB to 10 TB per unit
- Prices from 1 to 300 for the same capacity
- Very different reliability
- Very different speeds

# A variety of storage devices

## Main differences

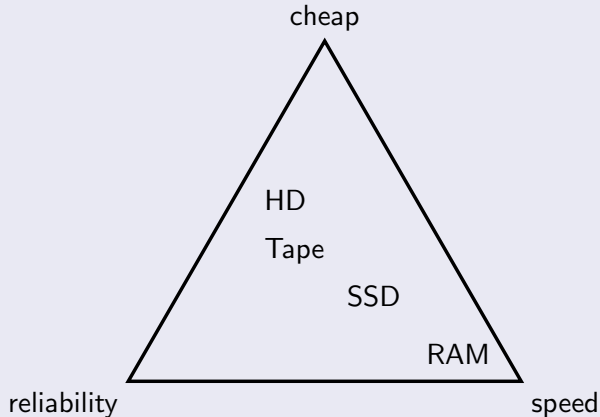
- Capacities from 1 GB to 10 TB per unit
- Prices from 1 to 300 for the same capacity
- Very different reliability
- Very different speeds

## Typical numbers in 2015

	Capacity per unit	Latency	\$/TB	Speed	reliability
RAM	16 GB	5 ns	5500 \$	10 GB s <sup>-1</sup>	volatile
SSD	500 GB	10 μs	400 \$	550 MB s <sup>-1</sup>	poor
HD	6 TB	3 ms	30 \$	150 MB s <sup>-1</sup>	average
Tape	10 TB	100 s	25 \$	450 MB s <sup>-1</sup>	good

# A variety of storage devices

You cannot have everything





# Reliability in real world (CERN)

## For disks

- probability of losing a disk per year : few %, up to 10%
  - with 60K disks, it's around 10 per day
  - and all files are lost
- one unrecoverable bit error in  $10^{14}$  bits read/written
  - for 10GB files, that's one file corrupted per 1000 files written

# Reliability in real world (CERN)

## For disks

- probability of losing a disk per year : few %, up to 10%
  - with 60K disks, it's around 10 per day
  - and all files are lost
- one unrecoverable bit error in  $10^{14}$  bits read/written
  - for 10GB files, that's one file corrupted per 1000 files written

## For tapes

- probability of losing a tape per year :  $10^{-4}$ 
  - and you recover most of the data on it
  - net result is  $10^{-7}$  file loss per year
- one unrecoverable bit error in  $10^{19}$  bits read/written
  - for 10GB files, that's one file corrupted per 100M files written

# Practical Mass Storage - Real Big Data

when you count in 100s of PetaBytes...

## The constraints

- disks or tapes are the only possible solutions
- disks are unreliable at that scale, and need redundancy
  - we'll see that extensively
- tapes are cheaper long term storage by factor 2-2.5
- tape latency imposes data access on disk

# Specificities of tape storage

## Key points

- 450MB/s in sequential read/write
  - 3x the speed of a disk
  - who said tape is slow ?
- latency/seek time in the order of minutes !
  - due to mount time and robot arm moving
  - due to positioning
- storage is cheap, I/O is not
  - 20\$/TB for storage capacity
  - 25K\$ for each drive

# Tape efficiency

## Computation

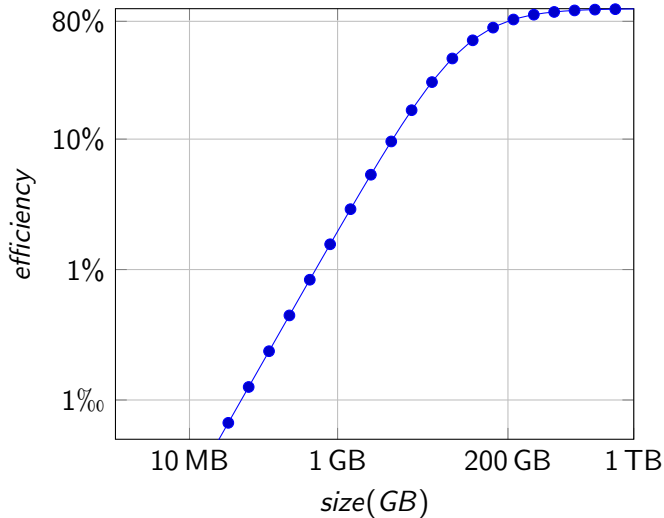
$$\text{efficiency} = \frac{I/O \text{ time}}{\text{mount time} + I/O \text{ time}}$$

$$\text{mount size} = \text{mount time} * \text{drive speed}$$

$$\text{efficiency} = \frac{1}{1 + \frac{\text{mount size}}{\text{data size}}}$$

$$\text{mount size} \simeq 50 \text{ GB}$$

# Tape efficiency



**No mount for less than 100 GB !**

# Hierarchical storage

## Layers

- tape as primary storage
- disks used as a cache in front of the tape system
- SSD used as cache in front of disks (or inside them)

## CERN's case (2015)

- tape capacity : 220 PB
- tape usage : 190 PB
- disk raw capacity : 200 PB
- disk usable space : 105 PB

# Some consequences

## Disk cache management is needed

- the disk cache needs garbage collection
- different algorithm used depending on usage
  - FIFO - First In First Out
  - LRU - Least Recently Used

## User need to adapt their usage

- data need to be prefetched from tape before access
- preferably in bulk



# Distributed storage

- 1 Storage devices
- 2 Distributed storage
  - Data distribution
  - Data federation
- 3 Parallelizing files' storage
- 4 Conclusion

# Handling large distributed storage

## Standard issues

- failures are very common
- data distribution is hard to balance
- congestions are frequent

# Keeping balanced data distribution

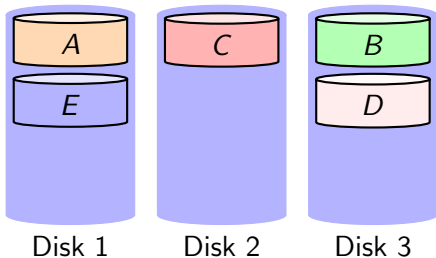
sounds initially easy

- standard load balancing
- taking benefit of the numerous clients

# Keeping balanced data distribution

sounds initially easy

- standard load balancing
- taking benefit of the numerous clients



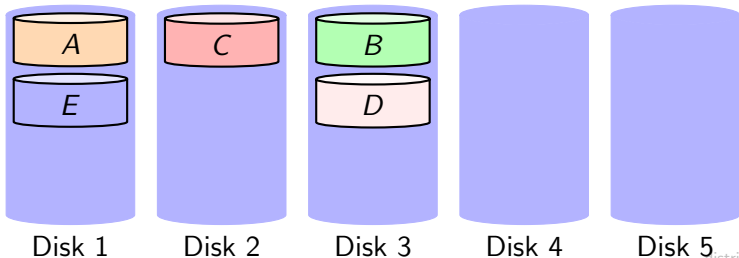
# Keeping balanced data distribution

## sounds initially easy

- standard load balancing
- taking benefit of the numerous clients

## Growing the cluster

- when you add disks, their are empty
- data need to be rebalanced



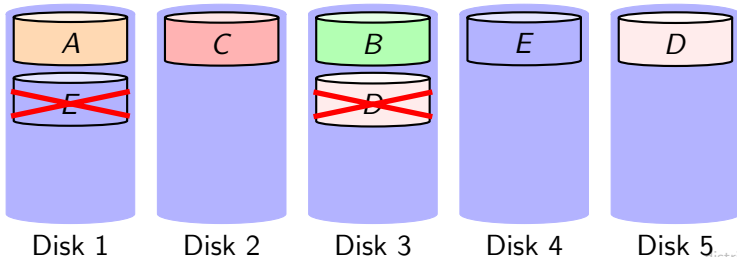
# Keeping balanced data distribution

## sounds initially easy

- standard load balancing
- taking benefit of the numerous clients

## Growing the cluster

- when you add disks, their are empty
- data need to be rebalanced



# Data congestions

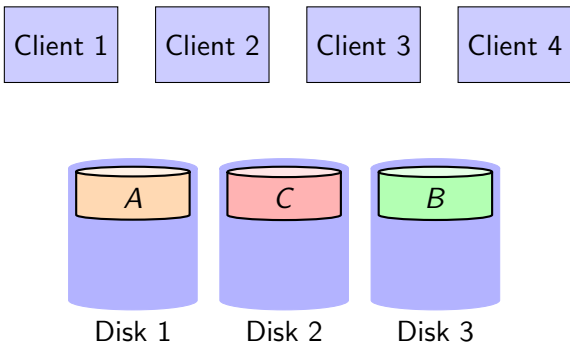
## The problem of popular files

- on writes, you can send a piece of data where you like
- on reads, you have to send the client where the data is
- but for popular pieces of data, the device holding it may become congested

# Data congestions

## The problem of popular files

- on writes, you can send a piece of data where you like
- on reads, you have to send the client where the data is
- but for popular pieces of data, the device holding it may become congested

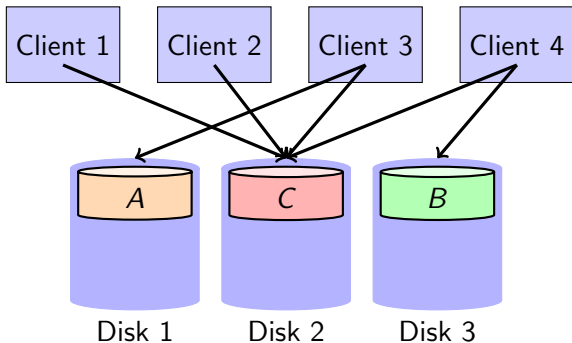




# Data congestions

## The problem of popular files

- on writes, you can send a piece of data where you like
- on reads, you have to send the client where the data is
- but for popular pieces of data, the device holding it may become congested



# Can be solved using data temperature

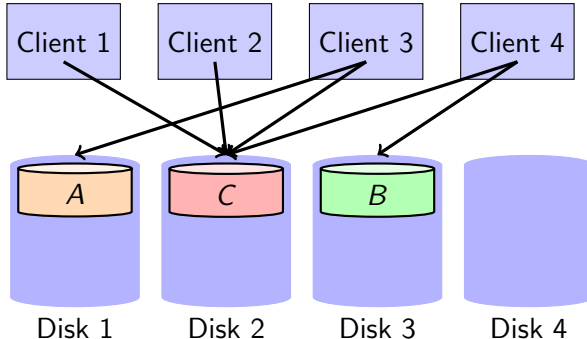
## Principle

- temperature is data popularity
- “hot” data is accessed more often
- data getting hotter than given threshold get replicated
- on cool down, replicas should be dropped

# Can be solved using data temperature

## Principle

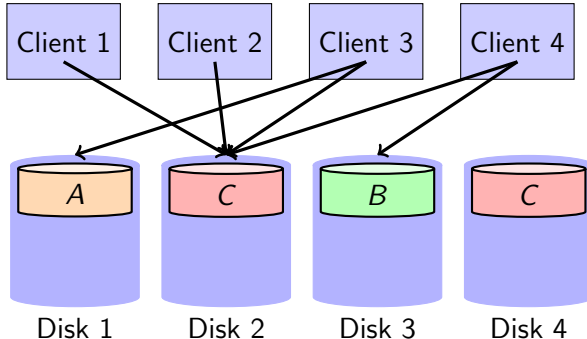
- temperature is data popularity
- “hot” data is accessed more often
- data getting hotter than given threshold get replicated
- on cool down, replicas should be dropped



# Can be solved using data temperature

## Principle

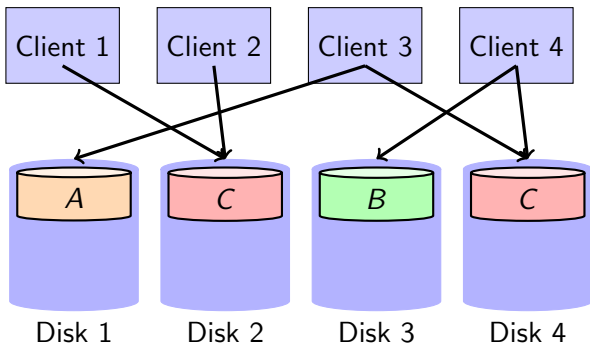
- temperature is data popularity
- “hot” data is accessed more often
- data getting hotter than given threshold get replicated
- on cool down, replicas should be dropped



# Can be solved using data temperature

## Principle

- temperature is data popularity
- “hot” data is accessed more often
- data getting hotter than given threshold get replicated
- on cool down, replicas should be dropped



# The global view

## A World distributed storage

- large experiments' data are distributed around the world
- usually replicating the data at least once
- networks are now fast and you can access remote data through wide area network when local data is not available
- but you do not want to do that by default
- and you want to try the “closest” replica first

# The global view

## A World distributed storage

- large experiments' data are distributed around the world
- usually replicating the data at least once
- networks are now fast and you can access remote data through wide area network when local data is not available
- but you do not want to do that by default
- and you want to try the “closest” replica first

## Data geolocalization

- a global central catalog lists all available replicas
- and their “physical” location
- with information on the topology of the network to reach them

# Data federation

## Hiding the underlying complexity

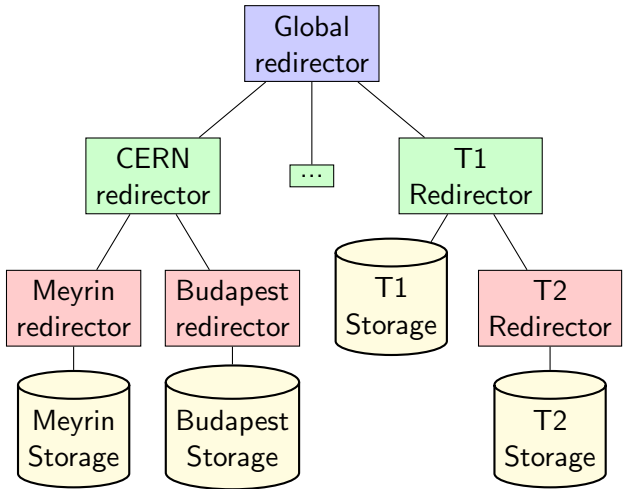
- some interfaces that hide this world wide distribution of data
- by redirecting clients dynamically to the closest data available

## Xrootd federation

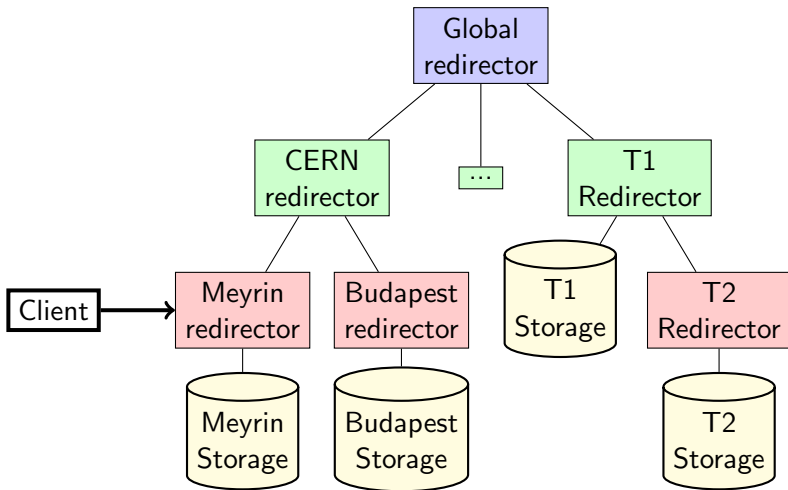
- widely used by LHC experiments across the world



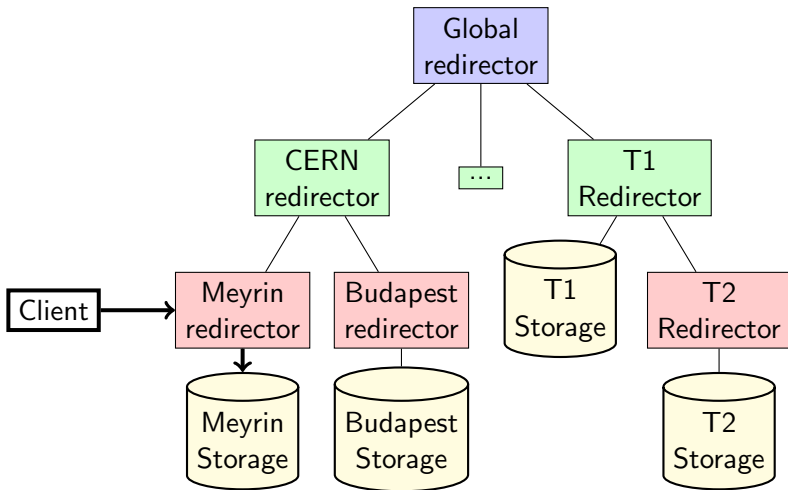
# Typical federation in an LHC experiment



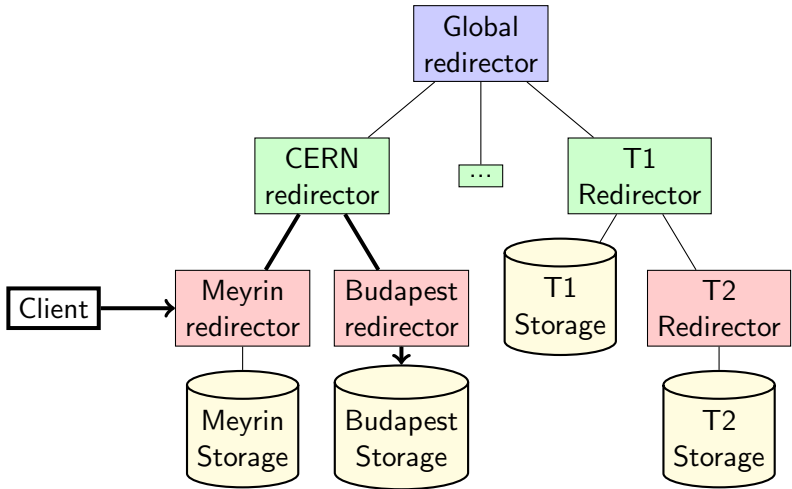
# Typical federation in an LHC experiment



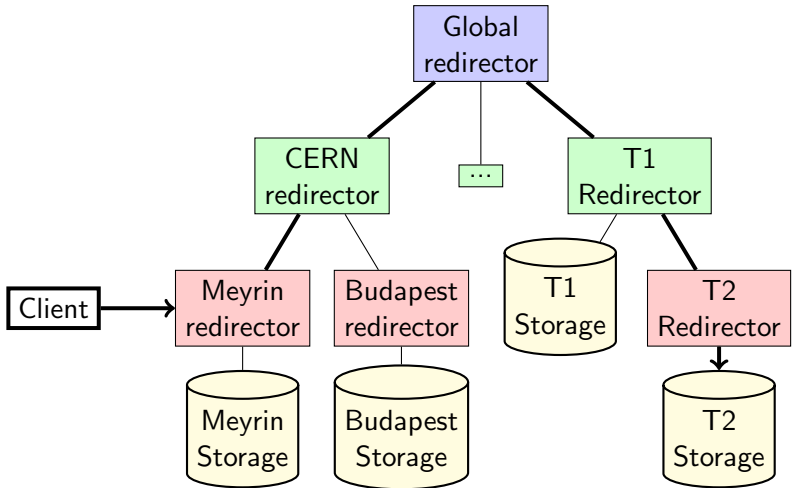
# Typical federation in an LHC experiment



# Typical federation in an LHC experiment



# Typical federation in an LHC experiment



# Parallelizing files' storage

- 1 Storage devices
- 2 Distributed storage
- 3 Parallelizing files' storage
  - Striping
  - Introduction to Map/Reduce
- 4 Conclusion

# Why to parallelize storage ?

## to work around limitations

- individual device speed (think disk)
  - a file is typically stored on a single device
- network cards' speed
  - 1 Gbit network still present
  - network congestion on a node reduces bandwidth per stream
- core network throughput
  - switches / routers are expensive
  - machines may have less throughput than their card(s) allow(s)
- hot data congestions
  - and the black hole it can generate
  - as slower transfers allow to accumulate more transfers

# Parallelizing through striping

## Main idea

- use several devices in parallel for a single stream
- moving the limitations up by summing performances

## Basic striping : Divide and conquer for storage

- split data into chunks aka stripes on different devices
- access in parallel



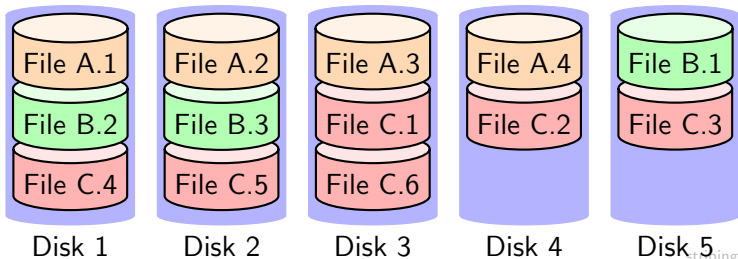
# Parallelizing through striping

## Main idea

- use several devices in parallel for a single stream
- moving the limitations up by summing performances

## Basic striping : Divide and conquer for storage

- split data into chunks aka stripes on different devices
- access in parallel



# RAID 0

## RAID

- stands to “Redundant Array of Inexpensive Disks”
- set of configurations that employ the techniques of striping, mirroring, or parity to create large reliable data stores from multiple general-purpose computer hard disk drives (Wikipedia)

## Useful RAID levels

RAID 0 striping

RAID 1 mirroring

RAID 5 parity

RAID 6 double parity

Can be implemented in hardware or software

# RAID 0

## RAID

- stands to “Redundant Array of Inexpensive Disks”
- set of configurations that employ the techniques of striping, mirroring, or parity to create large reliable data stores from multiple general-purpose computer hard disk drives (Wikipedia)

## Useful RAID levels

RAID 0 striping

RAID 1 mirroring

RAID 5 parity

RAID 6 double parity

} See data preservation talk

Can be implemented in hardware or software

# RAID versus RAIN

## RAIN

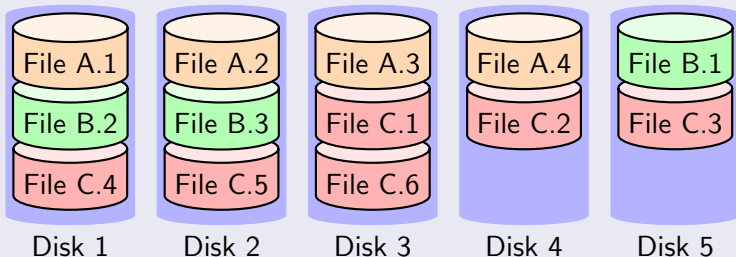
- Redundant Array of Inexpensive Nodes
- similar to RAID but across nodes

## Main interest

- tackle also the network limitations
- when used for redundancy, improves reliability
- more on this in subsequent lecture

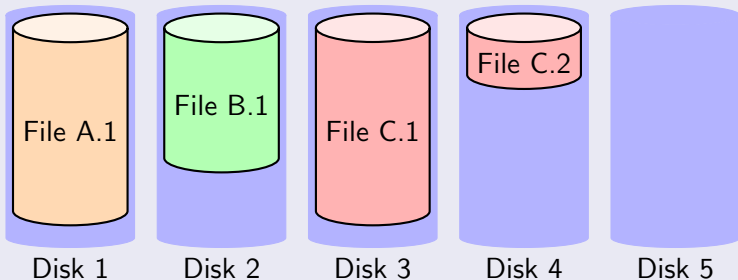
# Practical striping - the stripe size

## Desired picture



# Practical striping - the stripe size

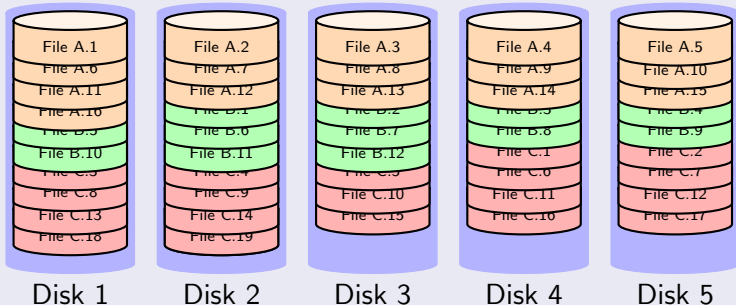
## Stripes too big



RAID has practically not effect

# Practical striping - the stripe size

## Striped too small



RAID will only kill performance by forcing disk to seek far too often

# How to choose the stripe size

## size of the stripe

- must be as small as possible to let small reads benefit from parallelization
- must not be too small
  - to avoid having to deal with too much metadata
  - to avoid too much disk seeking

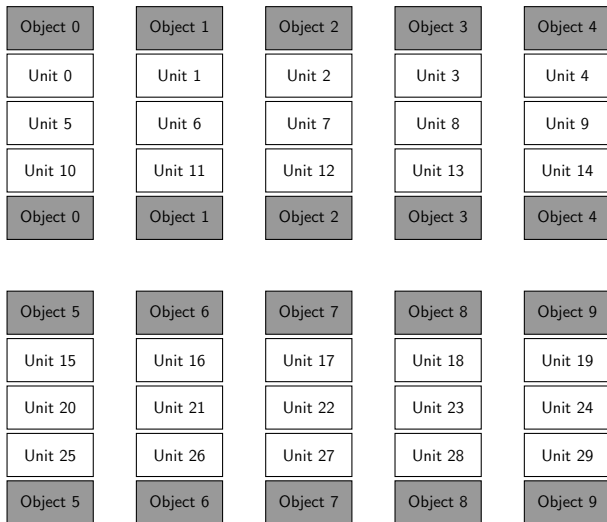


# A generic solution for the stripe size

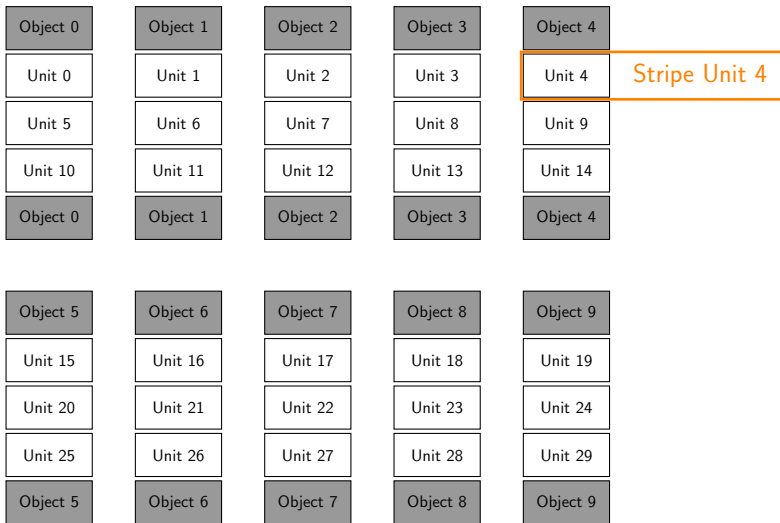
## Idea

- disentangle “stripe size” from “object size”
- “stripe size” is the size of one slice of data
- “object size” is the size of one block of data on disk
- several stripes are put together into one bigger object

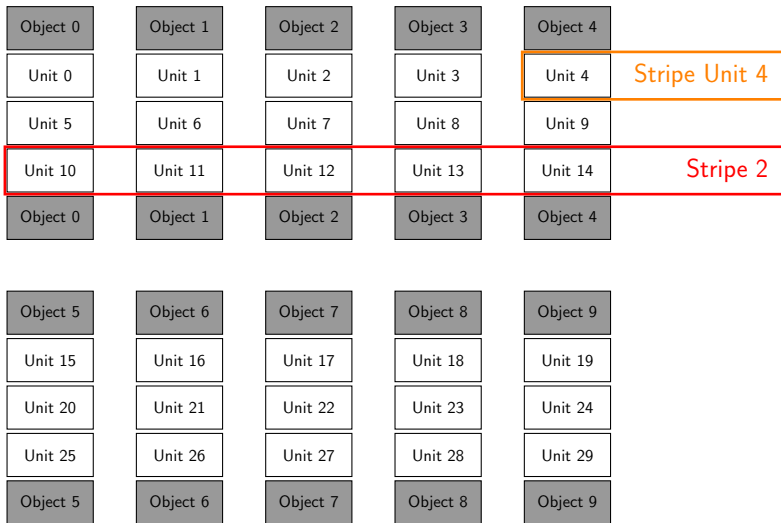
# Ceph striping



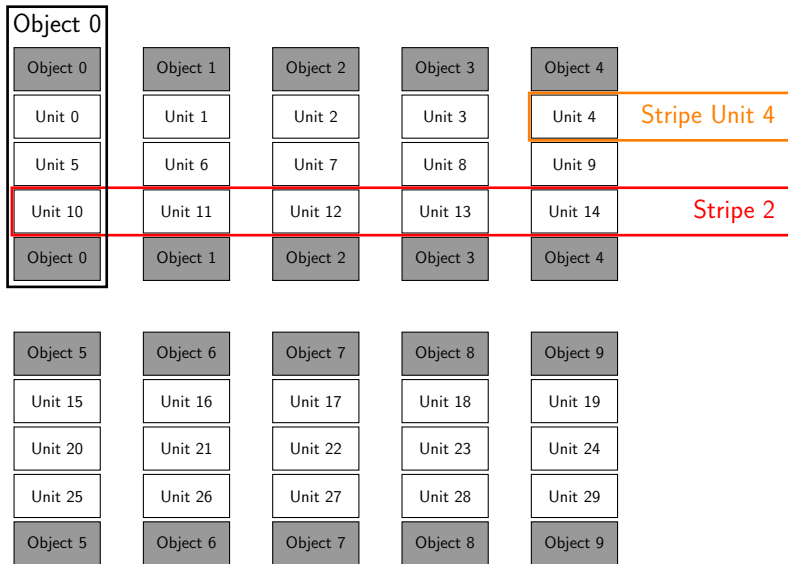
# Ceph striping



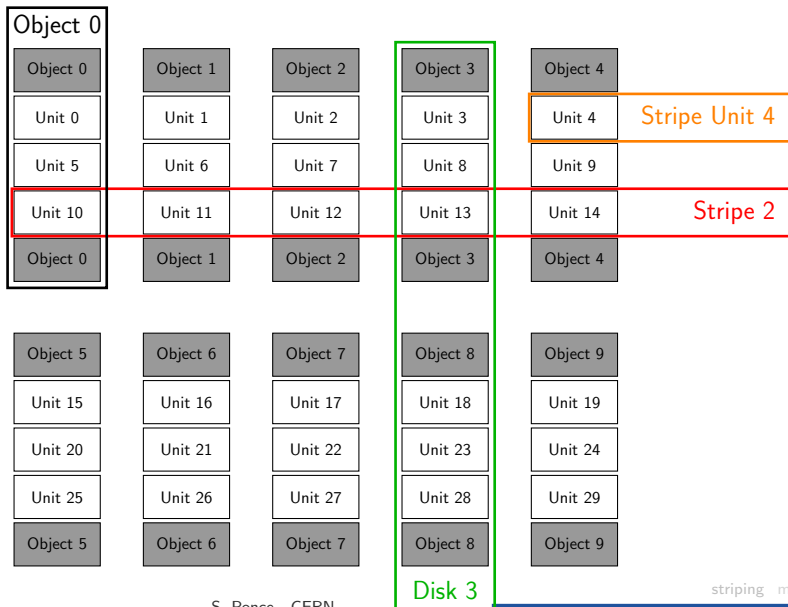
# Ceph striping



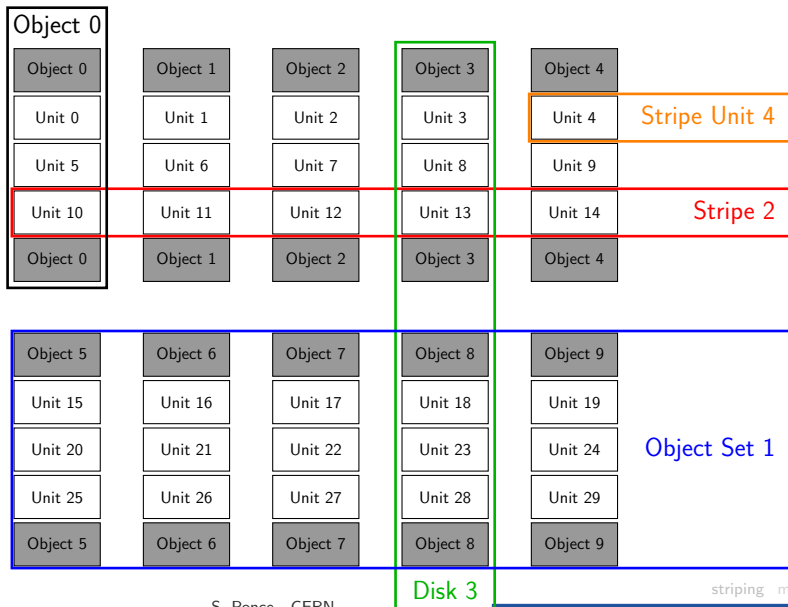
# Ceph striping



# Ceph striping



# Ceph striping



# Practical striping - number of disks

## Why to have many

- to increase parallelism
- to get better performances

## Why to have few

- to limit the risk of losing files
- as losing a disk now means losing all files of all disks
- if  $p$  is the probability to lose a disk  
the probability to lose one in  $n$  is  $p_n = np(1 - p)^{n-1} \sim np$



# A generic solution for the number of disks

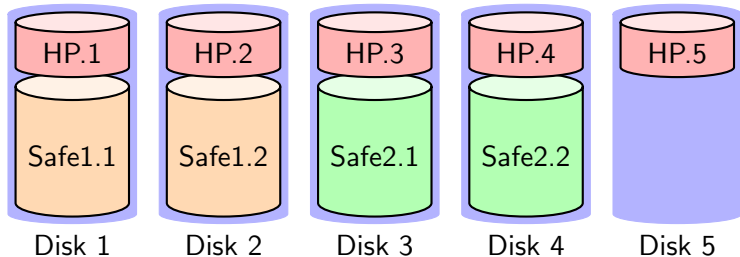
## Idea

- disentangle “nb disks” from “nb stripes”
- do not use all disks for all files
- adapt your number of disks to each file
  - more disks for high performance files
  - less disks for more safety

# A generic solution for the number of disks

## Idea

- disentangle “nb disks” from “nb stripes”
- do not use all disks for all files
- adapt your number of disks to each file
  - more disks for high performance files
  - less disks for more safety



# Going further : Map/Reduce

## What do we have with striping ?

- striping allows to distribute server I/O on several devices
- but client still faces the total I/O
- and CPU is not distributed

## Map/Reduce Idea

- send computation to the data nodes
- “the most efficient network I/O is the one you don’t do”

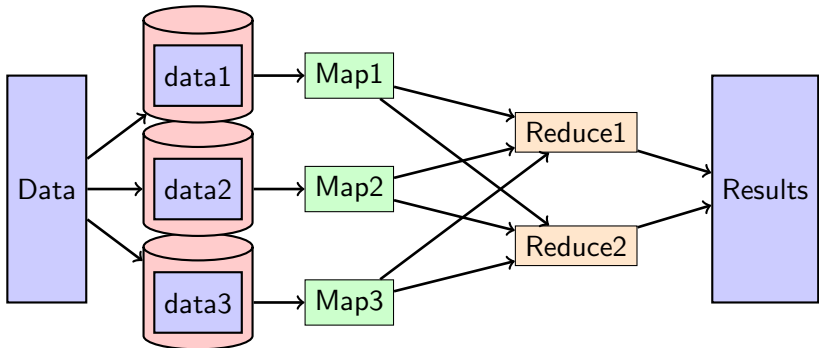
# Map/Reduce Introduction

## Schema

**Map** processes data locally and returns key/value pairs

**Shuffle** sends output to *Reduce* step based on output key

**Reduce** merges partial results to final output

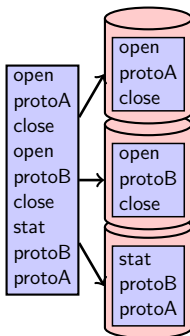


# Example : usage of protocols from logs

```
open  
protoA  
close  
open  
protoB  
close  
stat  
protoB  
protoA
```

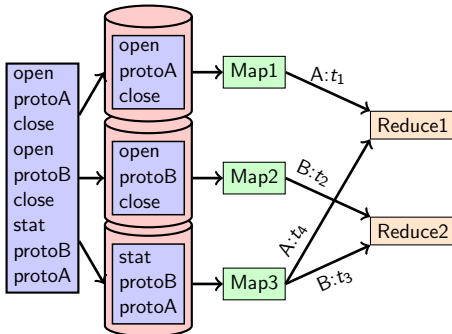
# Example : usage of protocols from logs

- 1 split logs on different nodes



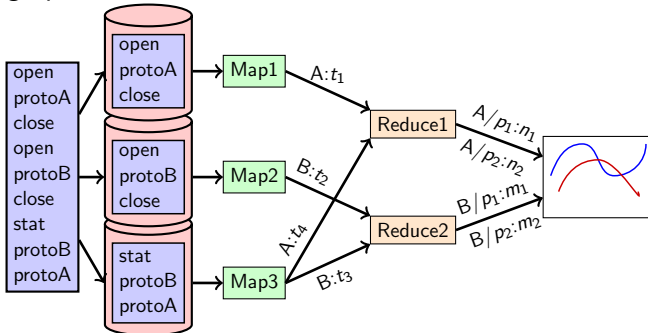
# Example : usage of protocols from logs

- 1 split logs on different nodes
- 2 parse and extract key/value (*Protocol/Date*) in Map



# Example : usage of protocols from logs

- ① split logs on different nodes
- ② parse and extract key/value (*Protocol/Date*) in Map
- ③ count accesses and output *Date/nb accesses* in Reduce
- ④ build graph



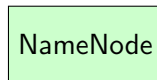


# The case of Hadoop, HDFS

## HDFS : the Hadoop Distributed FileSystem

- Distributed
  - files split in blocks spread over the cluster
  - blocks are typically 128MB
- Redundant
  - mirroring, 3 copies by default
  - erasure coding coming in 3.0 (still alpha)
- Hardware aware

# Parallel Read/Write in HDFS



DataNode 1



DataNode 2

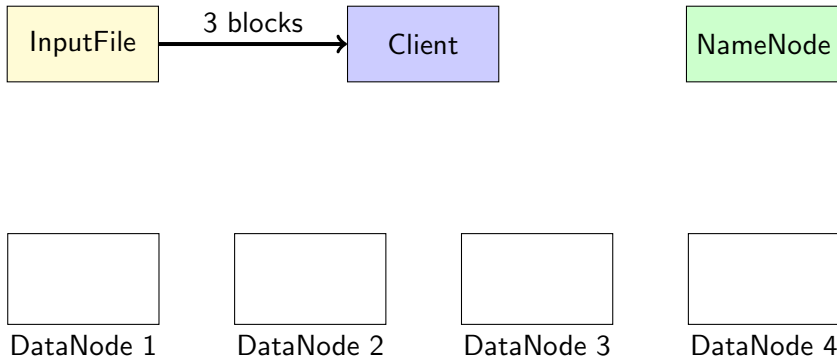


DataNode 3



DataNode 4

# Parallel Read/Write in HDFS



# Parallel Read/Write in HDFS



DataNode 1



DataNode 2

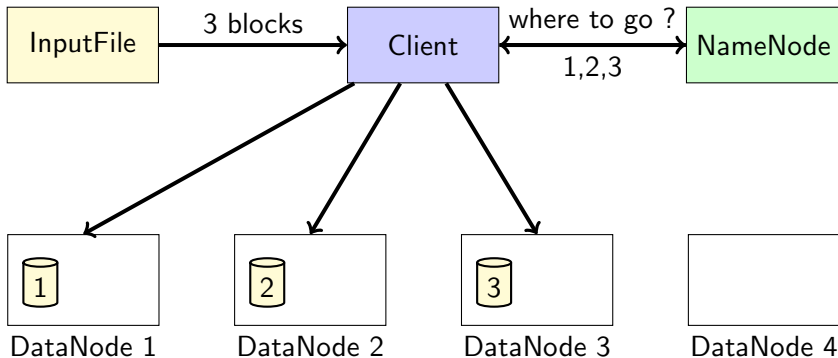


DataNode 3

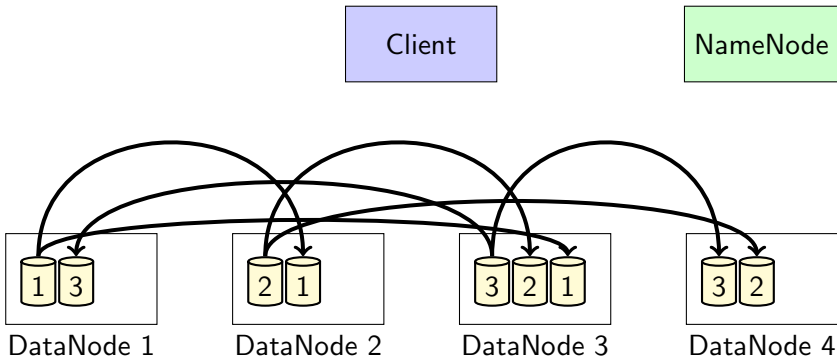


DataNode 4

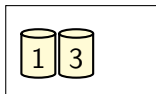
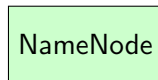
# Parallel Read/Write in HDFS



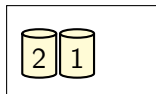
# Parallel Read/Write in HDFS



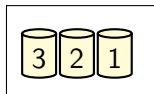
# Parallel Read/Write in HDFS



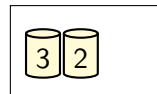
DataNode 1



DataNode 2

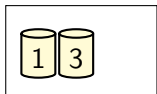
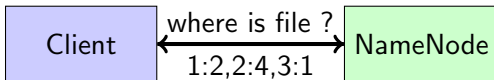


DataNode 3

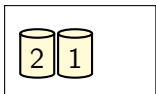


DataNode 4

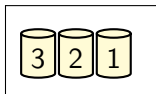
# Parallel Read/Write in HDFS



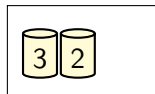
DataNode 1



DataNode 2



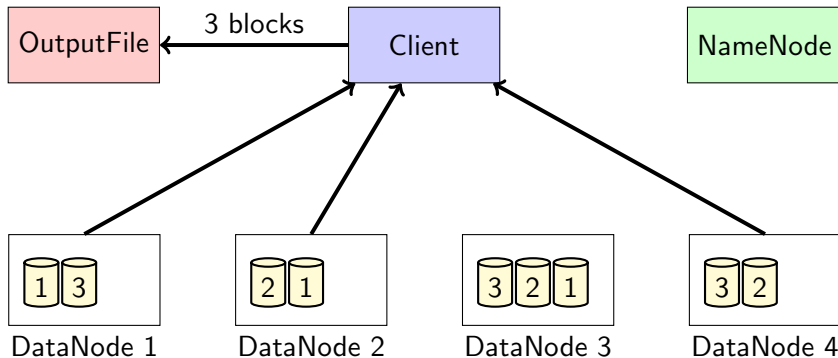
DataNode 3



DataNode 4



# Parallel Read/Write in HDFS



# Parallel Map/Reduce

Client

JobTracker

NameNode



DataNode 1



DataNode 2

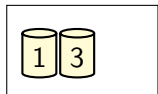
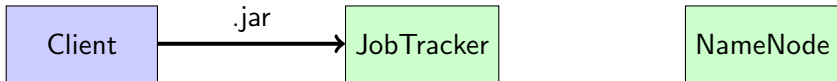


DataNode 3

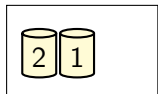


DataNode 4

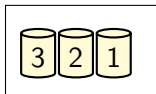
# Parallel Map/Reduce



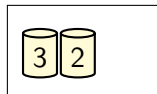
DataNode 1



DataNode 2

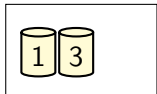
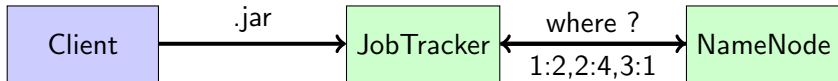


DataNode 3

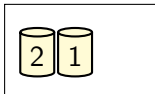


DataNode 4

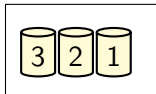
# Parallel Map/Reduce



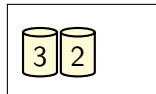
DataNode 1



DataNode 2

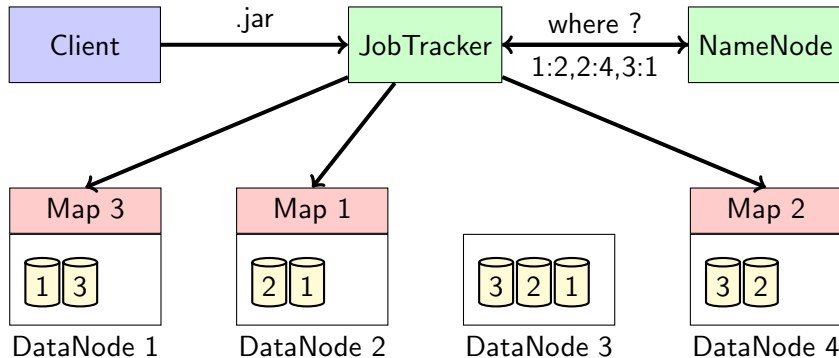


DataNode 3

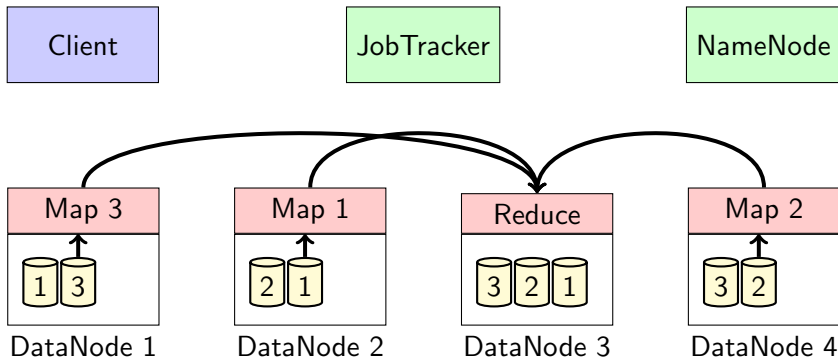


DataNode 4

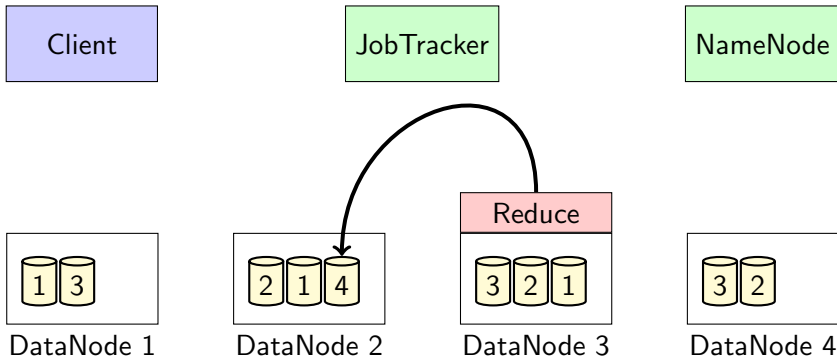
# Parallel Map/Reduce



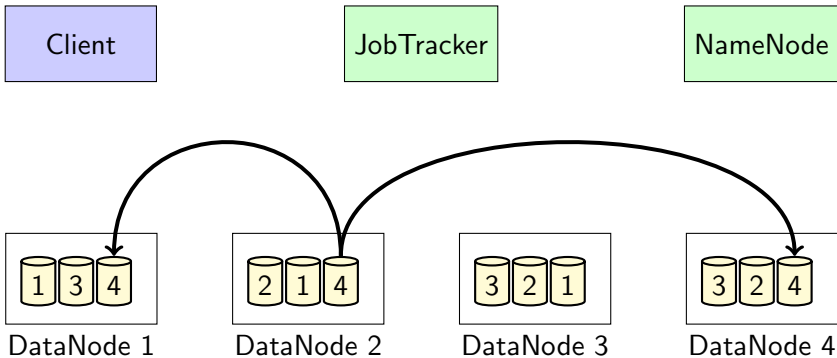
# Parallel Map/Reduce



# Parallel Map/Reduce



# Parallel Map/Reduce





# Conclusion

- 1 Storage devices
- 2 Distributed storage
- 3 Parallelizing files' storage
- 4 Conclusion**

# Conclusion

## Key messages of the day

- Take benefit of all existing devices to optimize your efficiency
- Do not underestimate the problem of data distribution
- Striping solves many problems. Use it but think twice
- Map/Reduce avoids a lot of I/O when results are small