



# Introdução ao Big Data



# Data warehouse

Data warehouse é um sistema que **extrai, limpa, padroniza e entrega** os dados das fontes de dados em um armazenamento dimensional dos dados permitindo implementar consultas e análises para o propósito de tomada de decisões.

*Fonte: Ralph Kimball, Joe Caserta: The Data Warehouse ETL Toolkit; Wiley 2004*

A parte mais **visível** para os clientes é “**Consultar e Analisar**”

A parte mais **complexa e demorada** é “extrair, limpar, padronizar e entregar”

# Arquitetura padrão



**GRANDE** quantidade  
de dados  
estão sendo geradas  
em real-time

O que representa 60 segundos de dados?



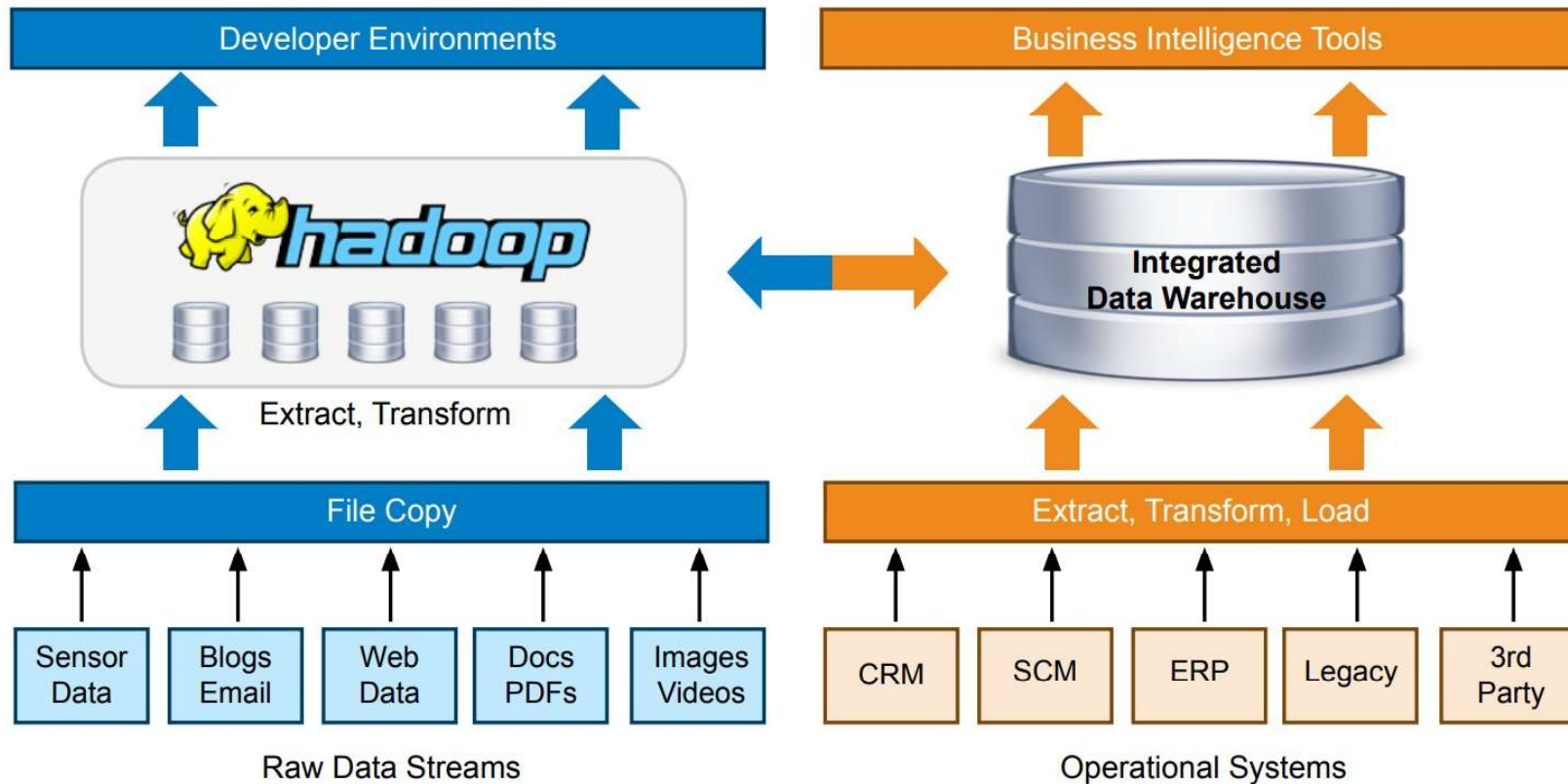
# 2018 *This Is What Happens In An Internet Minute*



# 2019 *This Is What Happens In An Internet Minute*



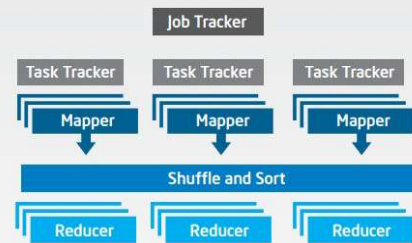
# Big Data



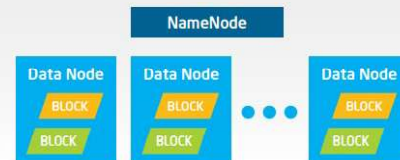
# Hadoop

## LOGICAL ARCHITECTURE

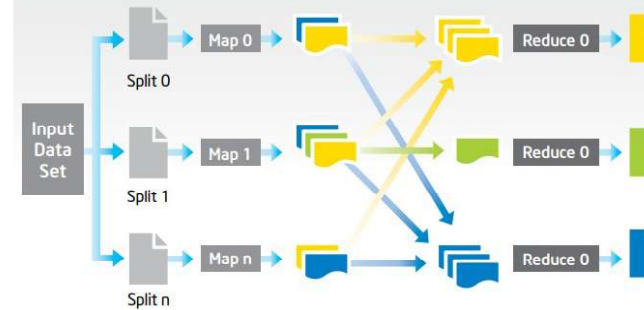
### Processing: MapReduce



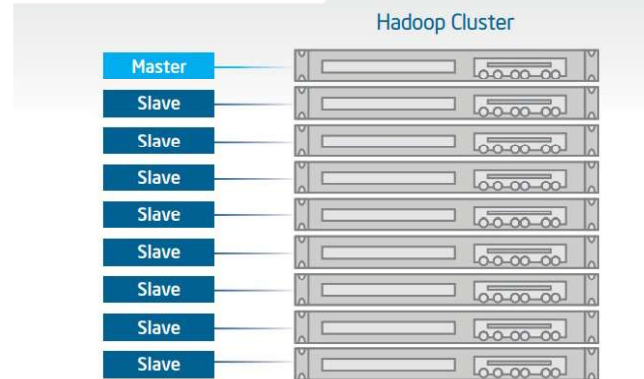
### Storage: HDFS



## PROCESS FLOW

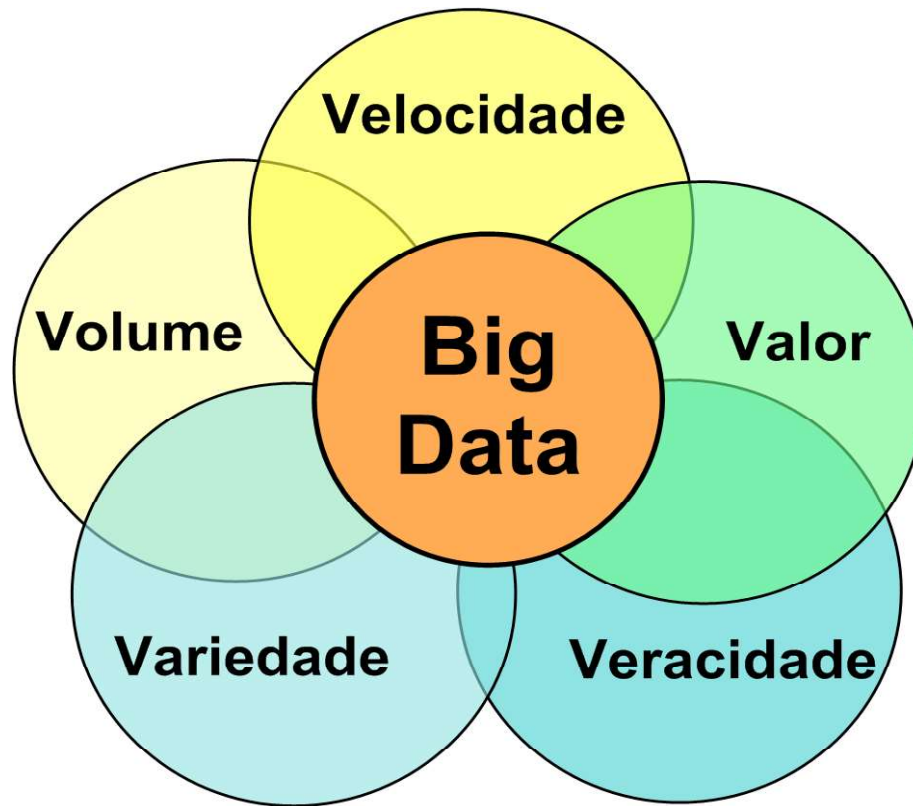


## PHYSICAL ARCHITECTURE





# 5 Vs de Big Data



# Data Lake

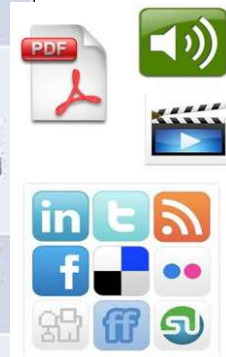


# Data Lake

	Data Warehouse	Data Lake
Dados	<ul style="list-style-type: none"><li>· Estruturados</li><li>· Processados</li></ul>	<ul style="list-style-type: none"><li>· Estruturados / Semi-estruturados / Não estruturados</li><li>· Não processados (em estado bruto)</li></ul>
Processamento	<ul style="list-style-type: none"><li>· Esquema de dados gerado no momento da escrita</li></ul>	<ul style="list-style-type: none"><li>· Esquema de dados gerado no momento da leitura</li></ul>
Armazenamento	<ul style="list-style-type: none"><li>· Alto custo para alto volume de dados</li></ul>	<ul style="list-style-type: none"><li>· Criado para ser de baixo custo, independente do volume de dados</li></ul>
Agilidade	<ul style="list-style-type: none"><li>· Pouco ágil, configuração fixa</li></ul>	<ul style="list-style-type: none"><li>· Bastante ágil, pode ser configurado e reconfigurado conforme necessário</li></ul>
Segurança	<ul style="list-style-type: none"><li>· Estratégias de segurança bastante maduras</li></ul>	<ul style="list-style-type: none"><li>· Ainda precisa aperfeiçoar o modelo de segurança e acesso aos dados</li></ul>
Usuários	<ul style="list-style-type: none"><li>· Analistas de Negócios</li></ul>	<ul style="list-style-type: none"><li>· Cientistas e Analistas de Dados</li></ul>

# Estruturas de dados em ETI

Dados Estruturados	Dados Semiestruturados	Dados Não Estruturados
Esquema pré-definido	Nem sempre há um esquema	Não há esquema
Estrutura regular	Estrutura irregular	Estrutura irregular
Estrutura independente dos dados	Estrutura embutida nos dados	Pode não ter estrutura alguma
Estrutura reduzida	Estrutura extensa (particularidades de cada dado, visto que cada um pode ter uma organização própria)	Estrutura extensa (particularidades de cada dado, visto que cada um pode ter uma organização própria)
Fracamente evolutiva	Fortemente evolutiva (estrutura modifica-se com frequência)	Fortemente evolutiva (estrutura modifica-se com frequência)
Prescritiva (esquemas fechados e restrições de integridade)	Estrutura descritiva	Estrutura descritiva
Distinção entre estrutura e dados é clara	Distinção entre estrutura e dados não é clara	Distinção entre estrutura e dados não é clara





# Algumas aplicações



# Monitoramento de tráfego

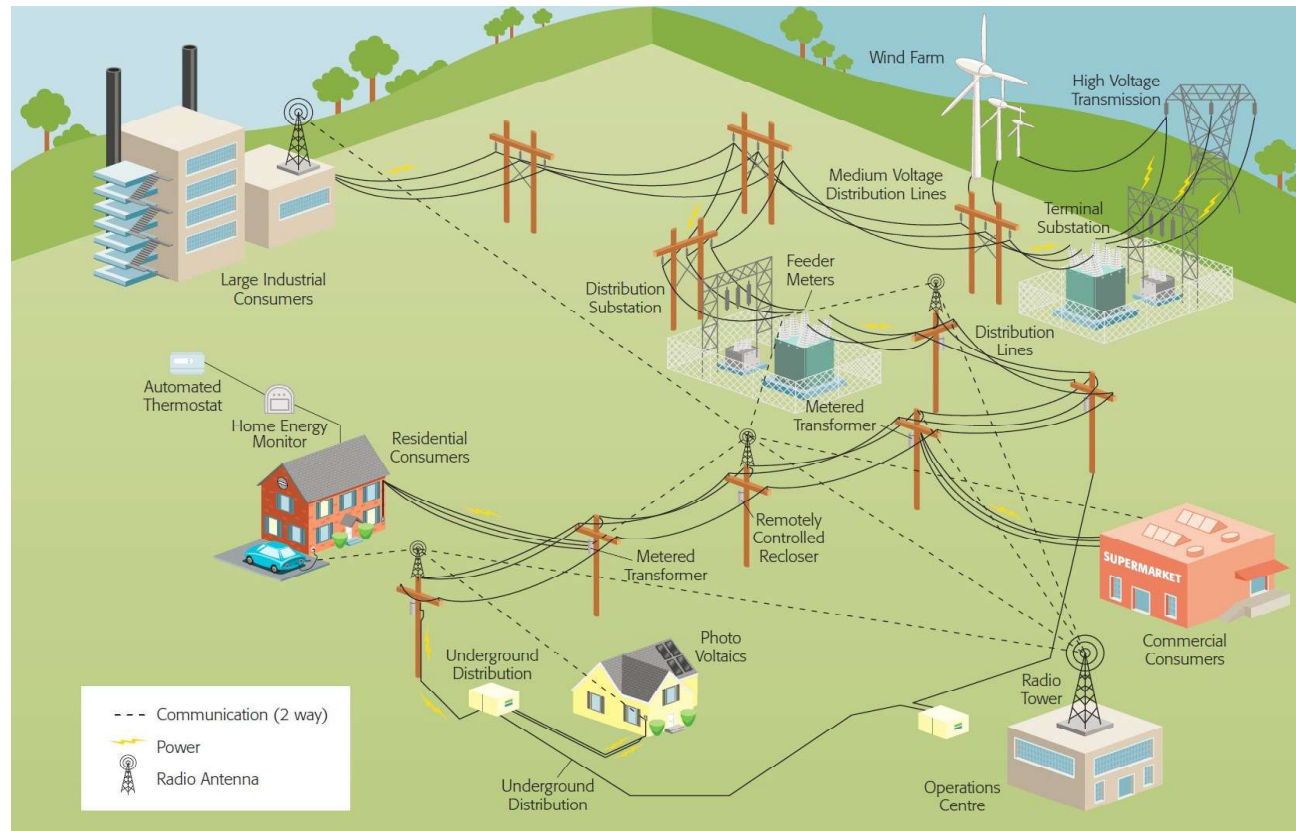


# Monitoramento de

- **tráfego** Arquitetura para *Stream* e *complex event processing* (CEP *processing*).
- Entrada baseada em sensores.
- Usa informação colaborativa (crowdsourcing) para resolver problemas de insegurança da fonte de dados.
- Dataset de 13GB por mês da cidade de Dublin (Irlanda).



# Smart Grid



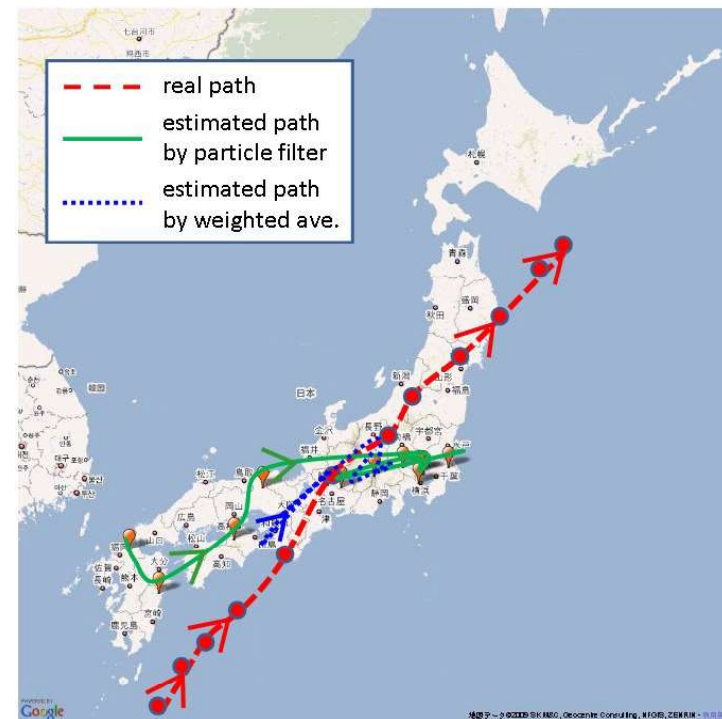
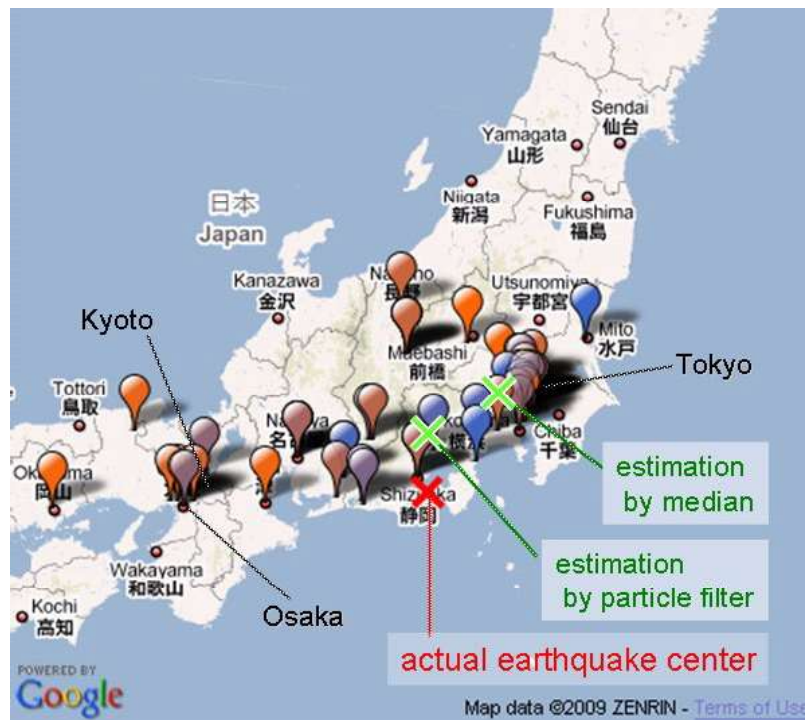


# Projeto de Smart Grid em L.A.

- 1.4 milhões de consumidores
- Otimização de demanda energética
- Predição de pico de demanda
- Fonte de dados:
  - AMIs ( Advanced Metering Infrastruc
- 3 TB de dados por dia



# Detecção de eventos em tempo real usando o Twitter



**Agência de meteorologia do Japão**



# Computação nas nuvens



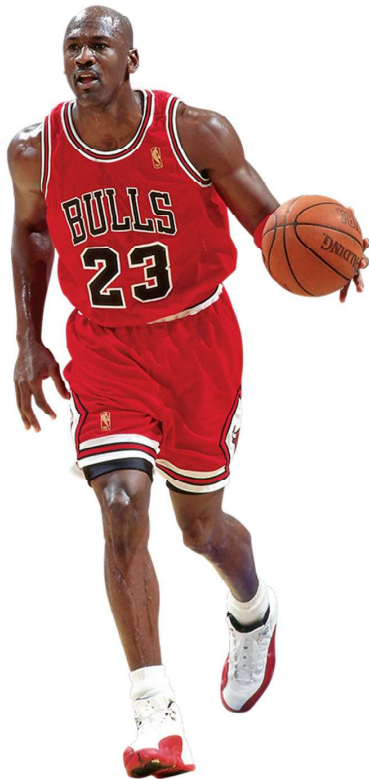
# Por que Cloud Computing?

DADOS SÃO GERADOS  
CADA VEZ MAIS EM  
GRANDES VOLUMES  
POR DIFERENTES  
FONTES E EM  
DIFERENTES FORMATOS

O MUNDO ESTÁ CADA  
VEZ MENOR

EM GERAL, NADA CABE NA MEMÓRIA DE UMA ÚNICA MÁQUINA  
QUANDO SE TRATA DE BIG DATA!

## Big Data, Mundo pequeno

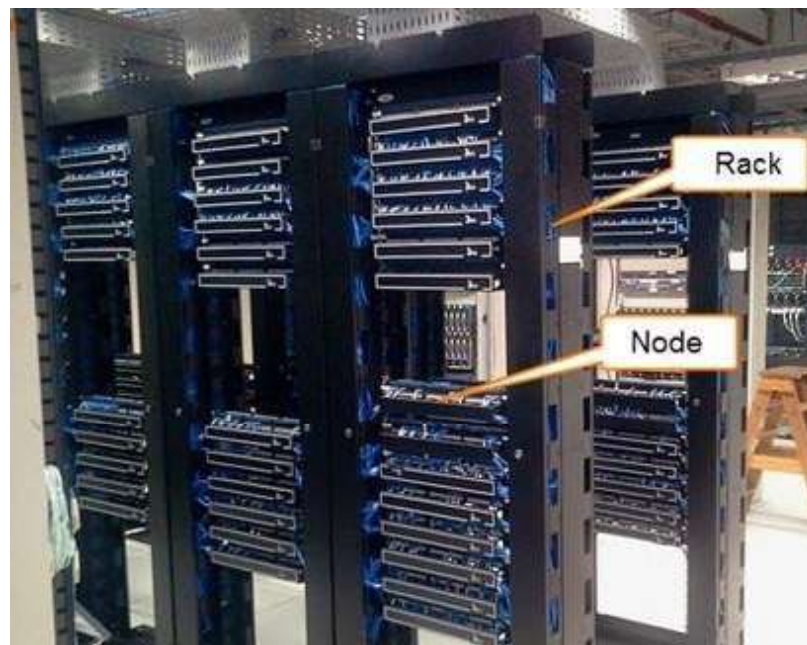


SUPERCOMPUTADOR  
MONOLÍTICO

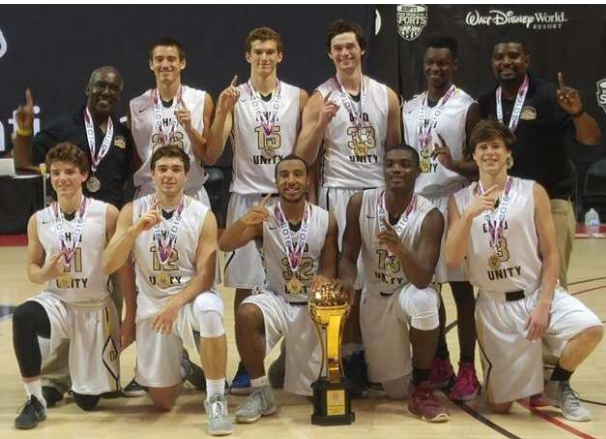


CLUSTER DE COMPUTADORES GENÉRICOS  
DISTRIBUÍDO

# Arquitetura do Cluster Hadoop



## Big Data, Mundo pequeno



DISTRIBUÍDO

- Muito hardware barato (HDFS)
- Replicação e Tolerância a falha (YARN)
- Computação Distribuída (Map Reduce)



# Cloudera Data Platform



DATA CENTER &  
PRIVATE CLOUD



HYBRID  
CLOUD



MULTI  
PUBLIC CLOUD

CLUSTERA  
**SDX**

METADATA / SCHEMA / MIGRATION / SECURITY / GOVERNANCE



DATA  
HUB



DATA FLOW &  
STREAMING



DATA  
ENGINEERING



DATA  
WAREHOUSE



OPERATIONAL  
DATABASE



MACHINE  
LEARNING

CLUSTERA RUNTIME



CONTROL  
PLANE



DATA  
CATALOG



REPLICATION  
MANAGER



WORKLOAD  
MANAGER



MANAGEMENT  
CONSOLE



# Hadoop

HDFS

MAPREDUCE

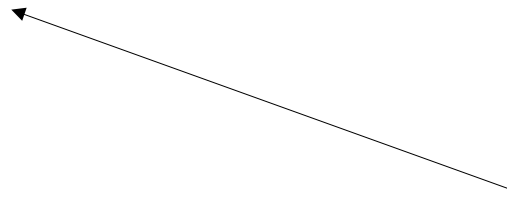
YARN

# Hadoop

MAPREDUCE

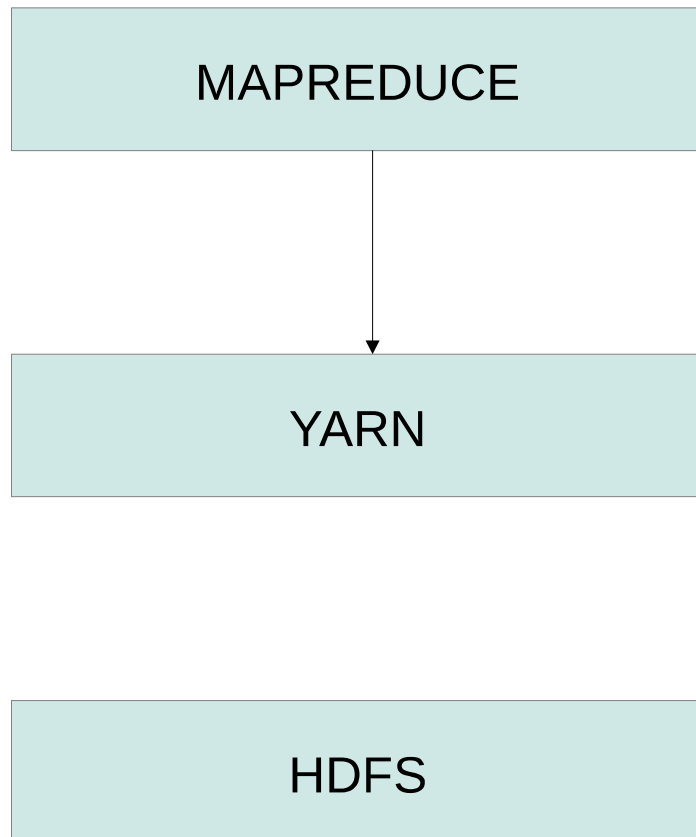
YARN

HDFS



O USUÁRIO DEFINE AS  
TAREFAS  
DE MAP E REDUCE POR MEIO  
DE UMA API MAPREDUCE

# Hadoop



UM JOB É LANÇADO NO  
CLUSTER

# Hadoop

MAPREDUCE

YARN



HDFS

O YARN VERIFICA ONDE E COMO  
DEVE EXECUTAR O JOB, E  
ARMAZENA O RESULTADO NO  
HDFS



# HDFS

## Hadoop Distributed File System



# HDFS



- Construído em hardware “commodity”
- Altamente tolerante a falha
- Projetado para processamento em lote - Acesso a dados tem alto “throughput” ao invés de baixa latência (ocultar latência)
- Suporta datasets muito grandes

## HDFS

- Gerencia o armazenamento de arquivos entre múltiplos discos



# HDFS

- Cada disco em uma máquina diferente em um cluster





# HDFS

- Um cluster de máquinas



# HDFS

- Um nó do cluster



# HDFS



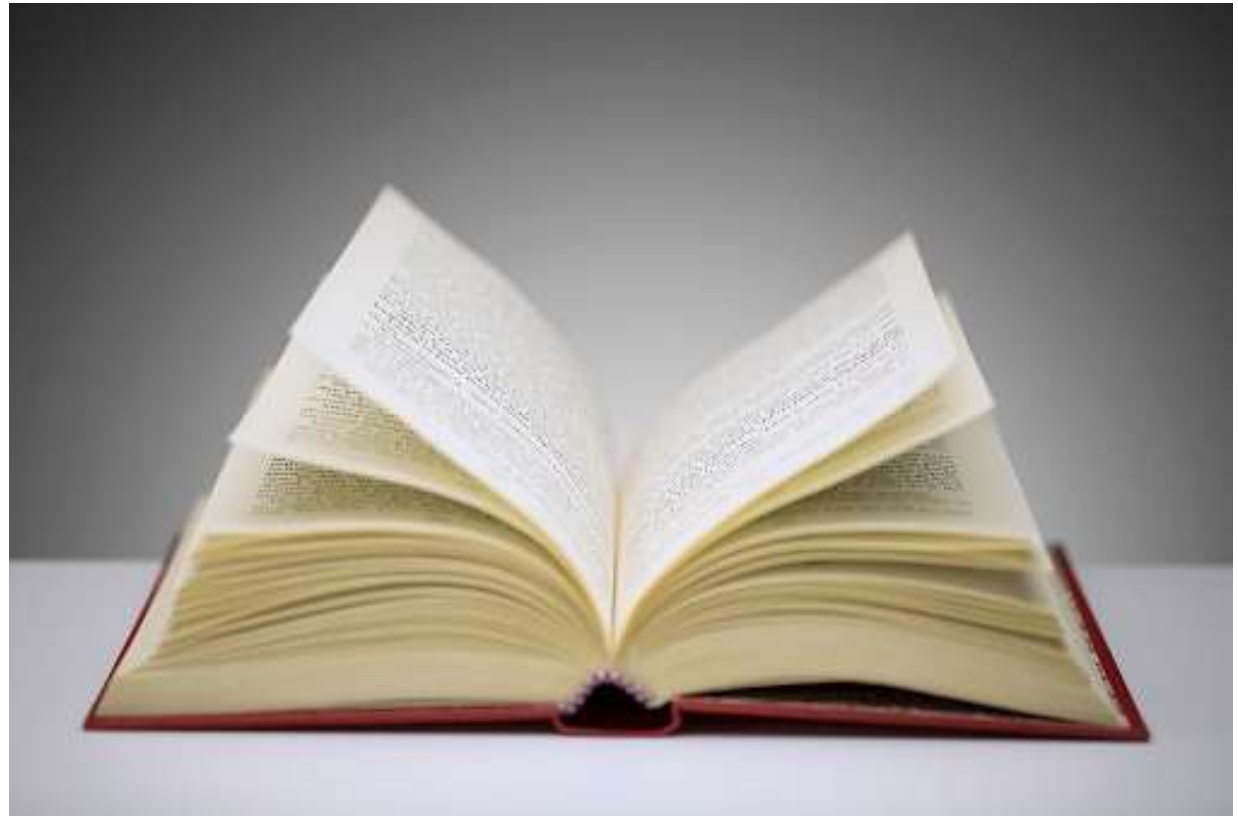
**Namenode**



**Datanodes**



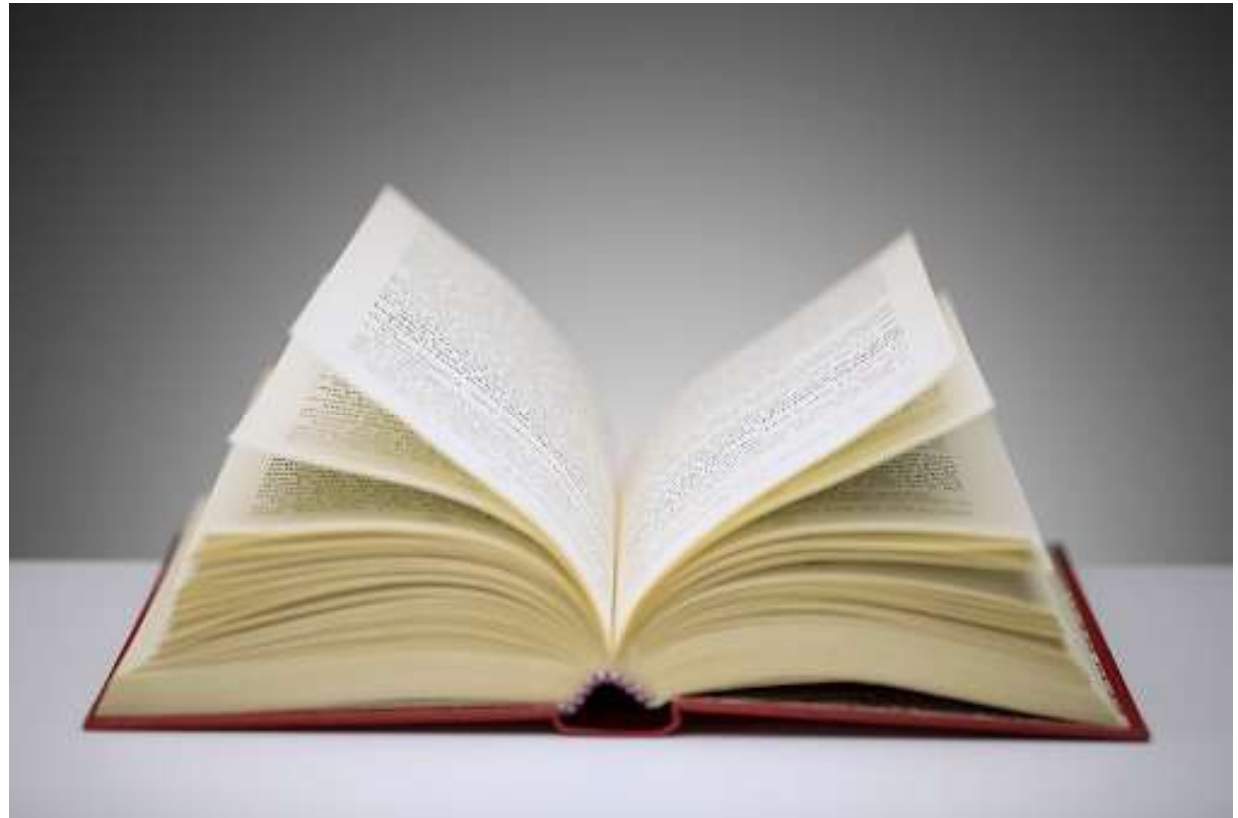
# HDFS



SE OS DADOS EM UM SISTEMA DISTRIBUÍDO PODE SER CONSIDERADO UM LIVRO...

# HDFS

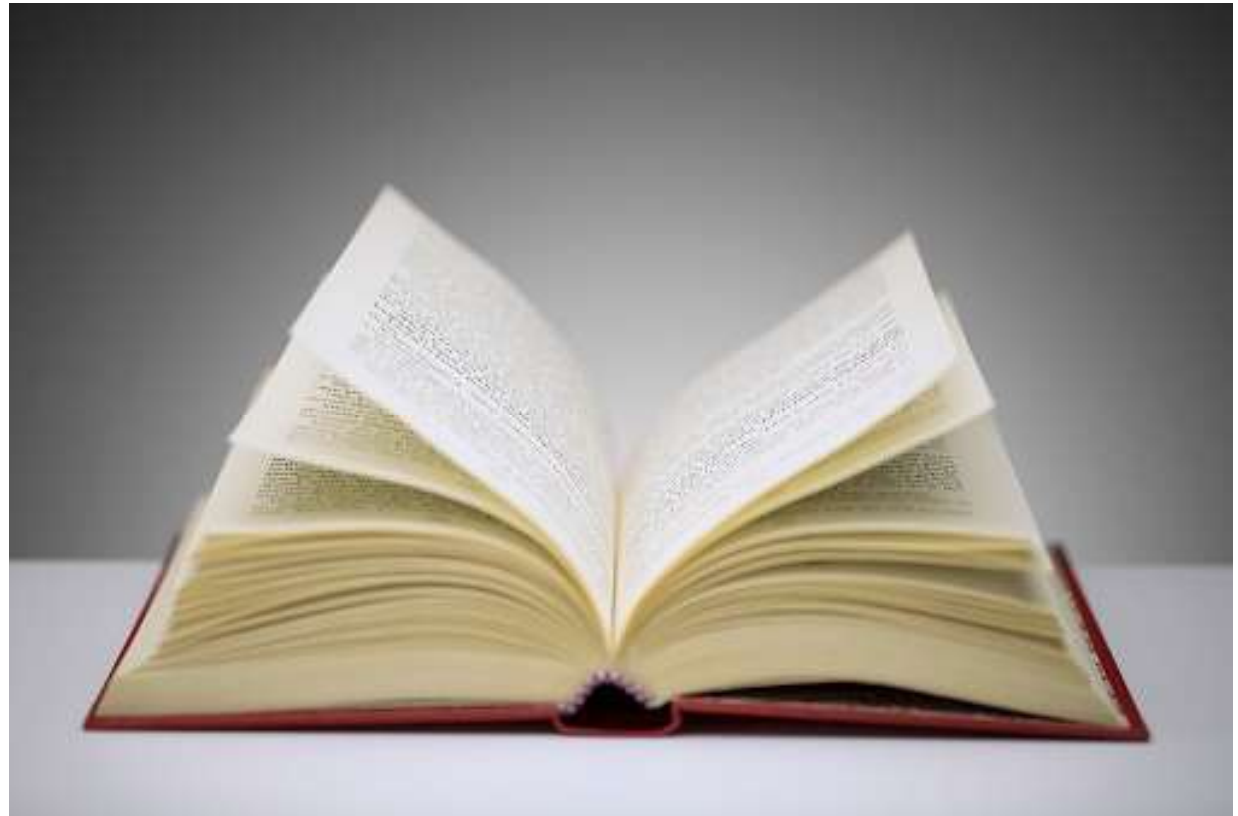
## Namenode



O NAMENODE É O ÍNDICE DO LIVRO

# HDFS

## Datanodes



CADA DATANODE GUARDA O TEXTO ATUAL  
EM CADA PÁGINA DO LIVRO



# HDFS



Next: up previous contents index  
Next: Dynamic indexing Up: Index construction Previous: Single-pass in-memory indexing Contents Index

## Distributed indexing

Collections are often so large that we cannot perform index construction efficiently on a single machine. This is particularly true of the World Wide Web for which we need large computer clusters [4] to construct any reasonably sized web index. Web search engines, therefore, use distributed indexing algorithms for index construction. The result of the construction process is a distributed index that is partitioned across several machines – either according to term or according to document. In this section, we describe distributed indexing for a term-partitioned index. Most large search engines prefer a document-partitioned index (which can be easily generated from a term-partitioned index). We discuss this topic further in Section 20.3 (page [4]).

The distributed index construction method we describe in this section is an application of MapReduce, a general architecture for distributed computing. MapReduce is designed for large computer clusters. The point of a cluster is to solve large computing problems on cheap commodity machines or nodes that are built from standard parts (processor, memory, disk) as opposed to on a supercomputer with specialized hardware. Although hundreds or thousands of machines are available in such clusters, individual machines can fail at any time. One requirement for robust distributed indexing is, therefore, that we divide the work up into chunks that we can easily assign and – in case of failure – reassign. A master node directs the process of assigning and reassigning tasks to individual worker nodes.

The map and reduce phases of MapReduce split up the computing job into chunks that standard machines can process in a short time. The various steps of MapReduce are shown in Figure 4.5 and an example on a collection consisting of two documents is shown in Figure 4.6. First, the input data, in our case a collection of web pages, are split into  $n$  splits where the size of the split is chosen to ensure that the work can be distributed evenly (chunks should not be too large) and efficiently (the total number of chunks we need to manage should not be too large); 16 or 64 MB are good sizes in distributed indexing. Splits are not preassigned to machines, but are instead assigned by the master node on an ongoing basis: As a machine finishes processing one split, it is assigned the next one. If a machine dies or becomes a laggard due to hardware problems, the split it is working on is simply reassigned to another machine.

Figure 4.5: An example of distributed indexing with MapReduce. Adapted from Dean and Ghemawat (2004).

`\includegraphics[width=11.5cm]{art/mapreduce2.eps}`

In general, MapReduce breaks a large computing problem into smaller parts by recasting it in terms of manipulation of key-value pairs. For indexing, a key-value pair has the form (termID, docID). In distributed indexing, the mapping from terms to termIDs is also distributed and therefore more complex than in single-machine indexing. A simple solution is to maintain a (perhaps precomputed) mapping for frequent terms that is copied to all nodes and to use terms directly (instead of termIDs) for infrequent terms. We do not address this problem here and assume that all nodes share a consistent term  $\rightarrow$  termID mapping.

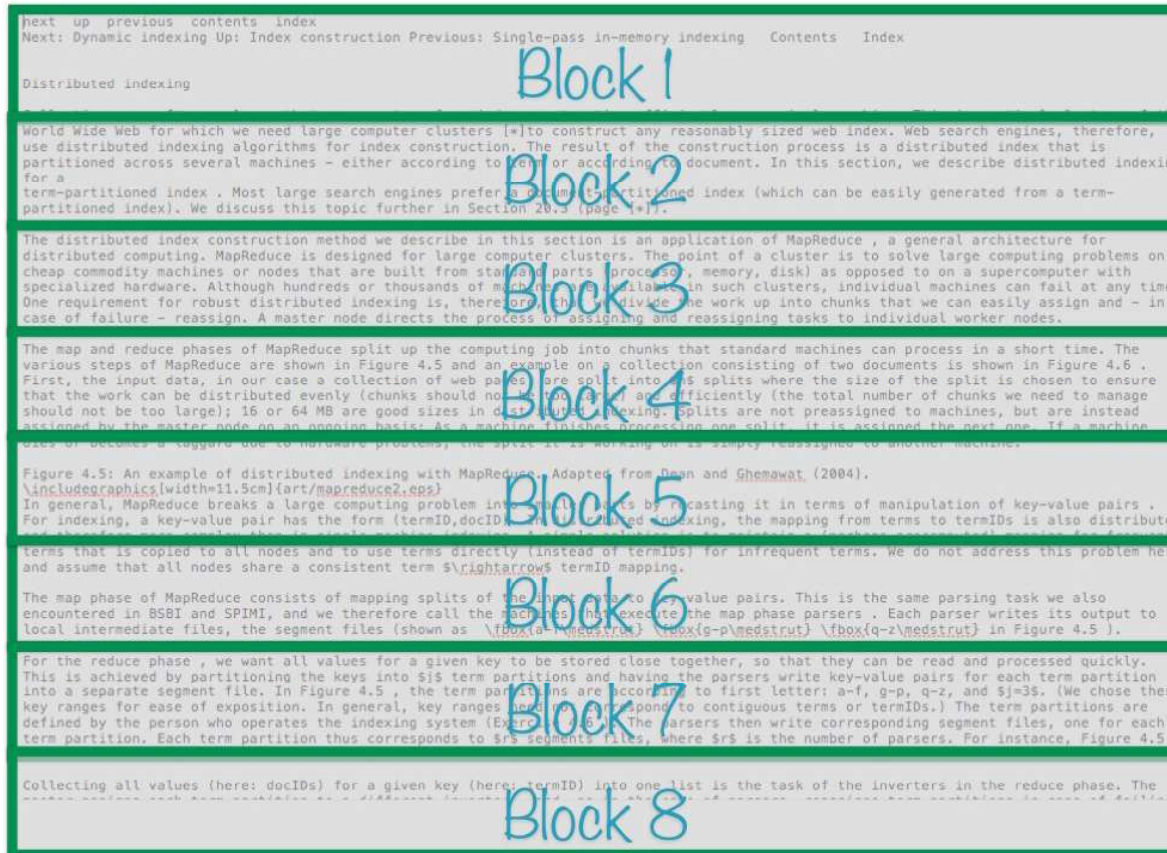
The map phase of MapReduce consists of mapping splits of the input data to key-value pairs. This is the same parsing task we also encountered in BSBI and SPIMI, and we therefore call the machines that execute the map phase parsers. Each parser writes its output to local intermediate files, the segment files (shown as `\fbox{a-f\medstrut}` `\fbox{g-p\medstrut}` `\fbox{q-z\medstrut}` in Figure 4.5).

For the reduce phase, we want all values for a given key to be stored close together, so that they can be read and processed quickly. This is achieved by partitioning the keys into  $j$  term partitions and having the parsers write key-value pairs for each term partition into a separate segment file. In Figure 4.5, the term partitions are according to first letter: a-f, g-p, q-z, and  $j=3$ . (We chose these key ranges for ease of exposition. In general, key ranges need not correspond to contiguous terms or termIDs.) The term partitions are defined by the person who operates the indexing system (Exercise 4.6). The parsers then write corresponding segment files, one for each term partition. Each term partition thus corresponds to  $s$  segments files, where  $s$  is the number of parsers. For instance, Figure 4.5 shows three a-f segment files of the a-f partition, corresponding to the three parsers shown in the figure.

Collecting all values (here: docIDs) for a given key (here: termID) into one list is the task of the inverters in the reduce phase. The

Imagine um grande arquivo de texto contendo todas as palavras da Wikipedia em inglês...

# HDFS



- Diferentes tamanhos de arquivos são processados da mesma forma;
- O armazenamento é simplificado;
- Unidade para replicação e tolerância a falhas;
- Tamanho de bloco padrão é de 128 MB;

Vamos quebrar o texto em blocos de sentenças...



## HDFS

O tamanho do bloco é um “trade off”



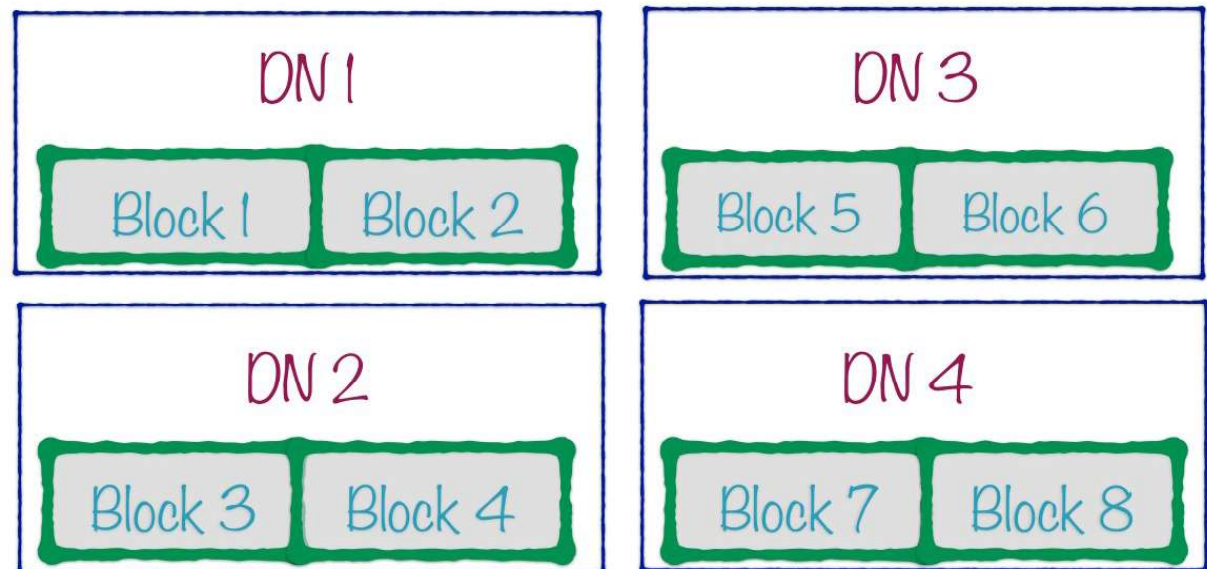
REDUZ O  
PARALELISMO



AUMENTA O  
OVERHEAD

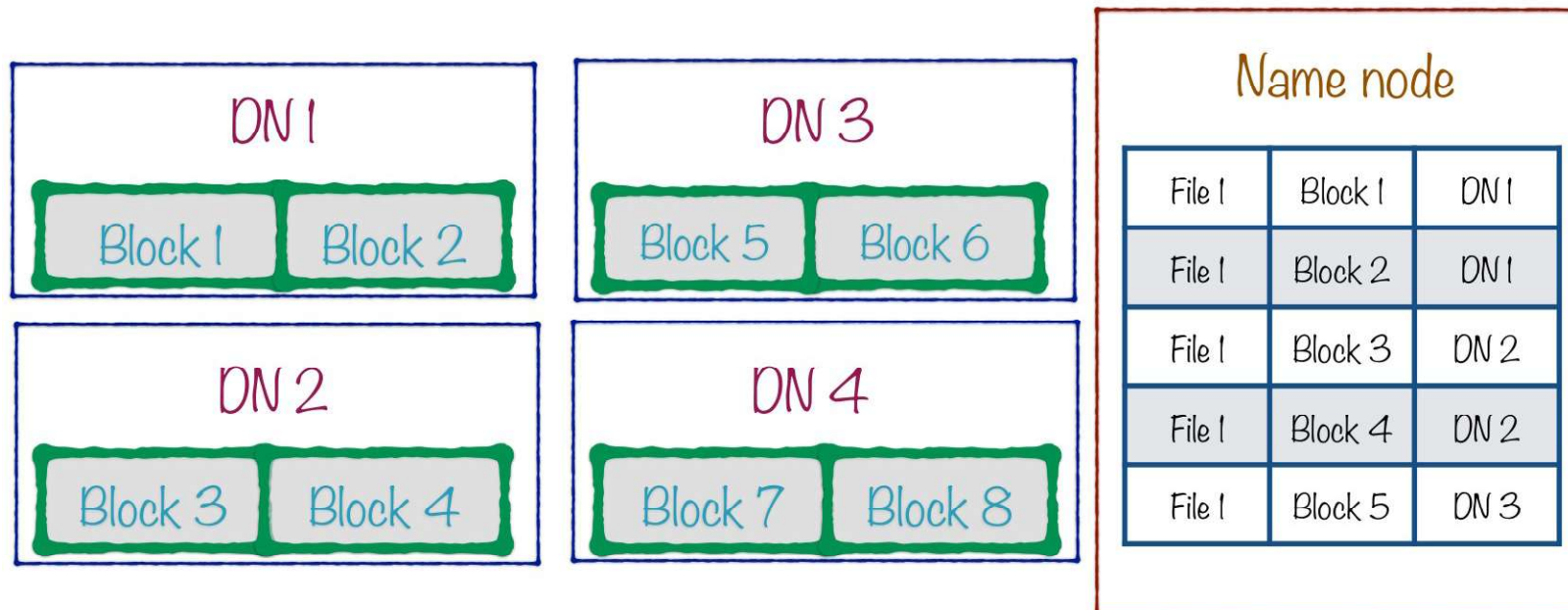
O tamanho ajuda a minimizar o  
tempo

## Armazenando um arquivo no HDFS



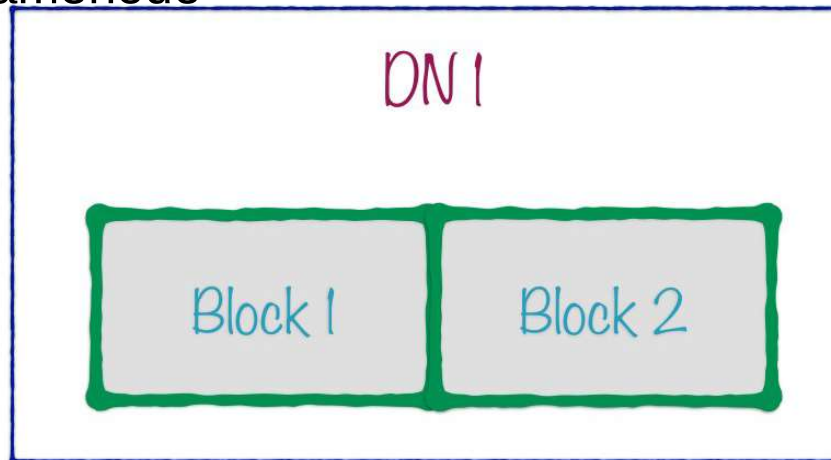
Como saber onde os fragmentos de um arquivo em particular estão???

## Armazenando um arquivo no HDFS



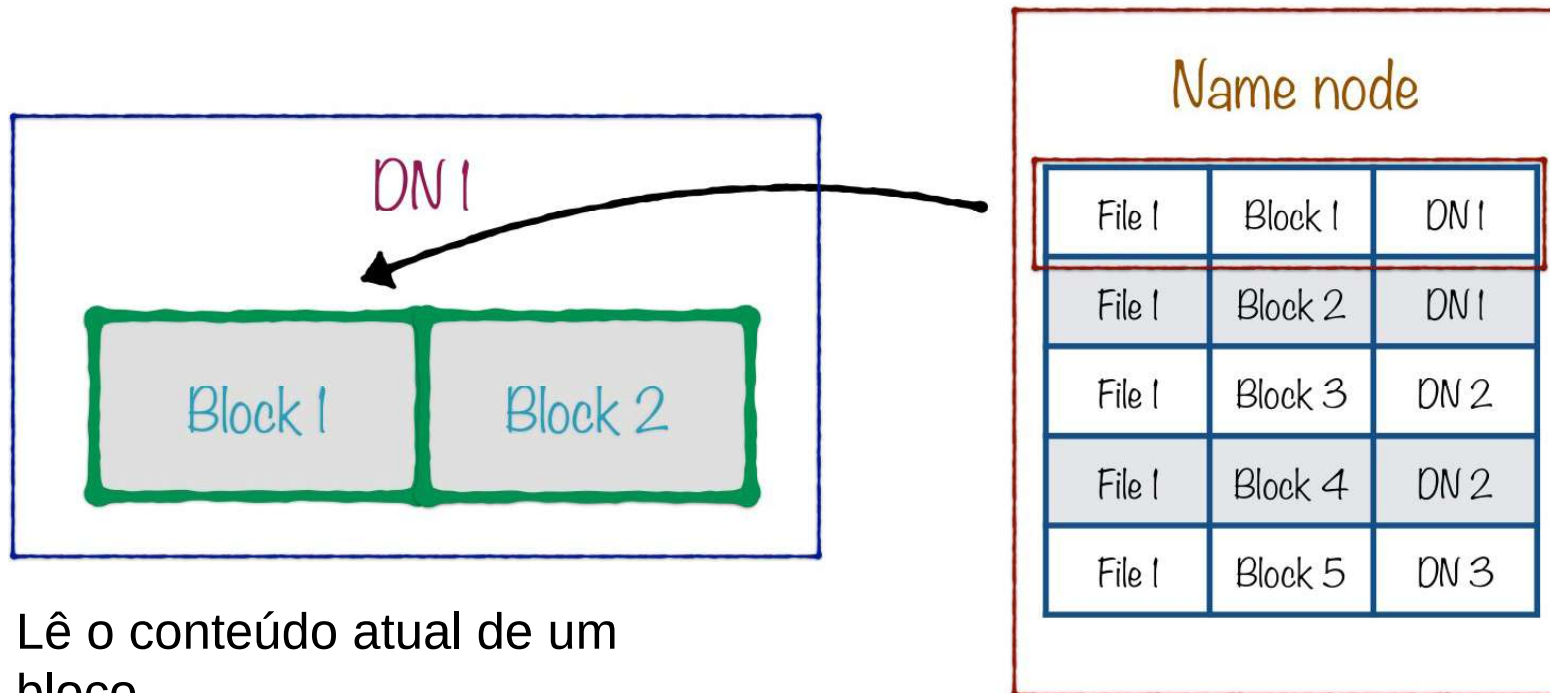
# Lendo um arquivo no HDFS

Requisição para o  
Namenode

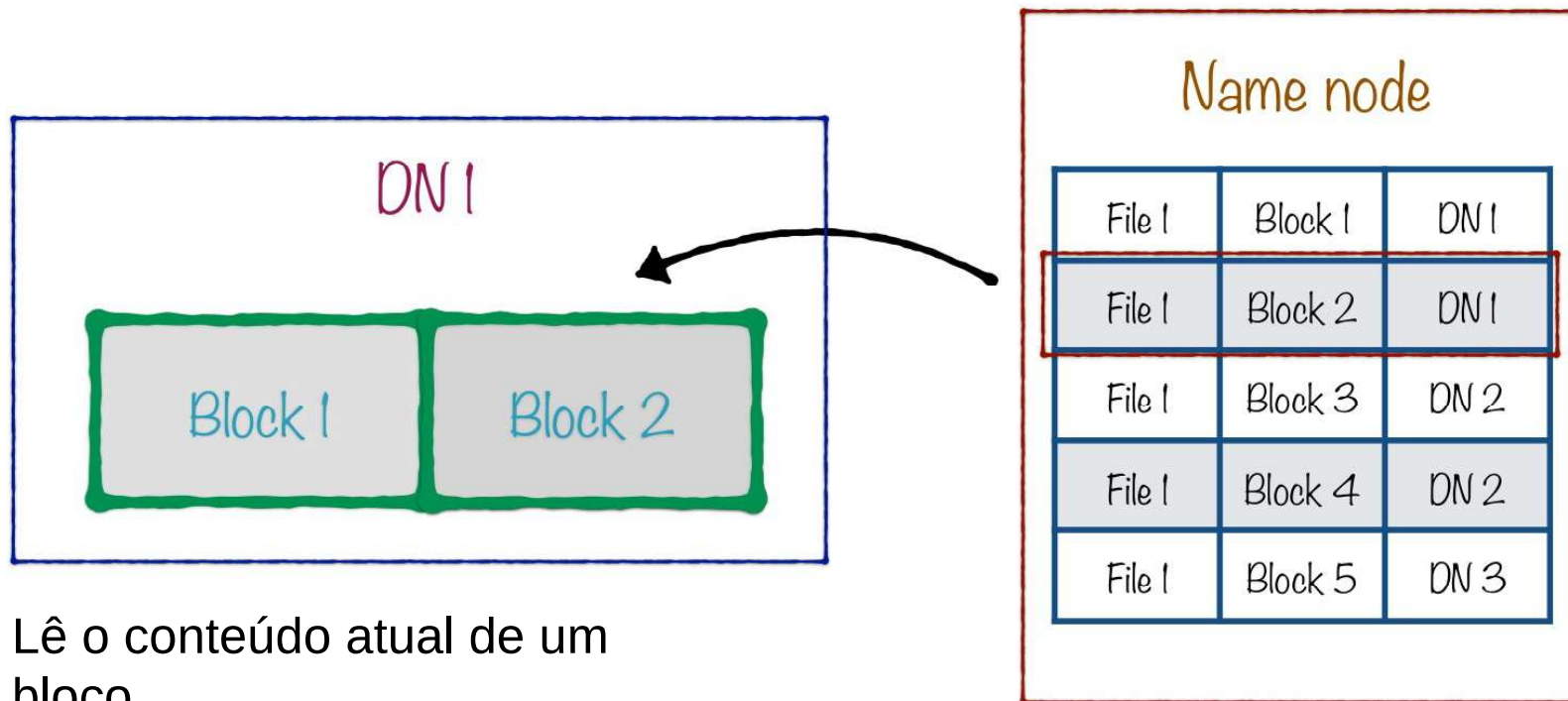


Name node		
File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3

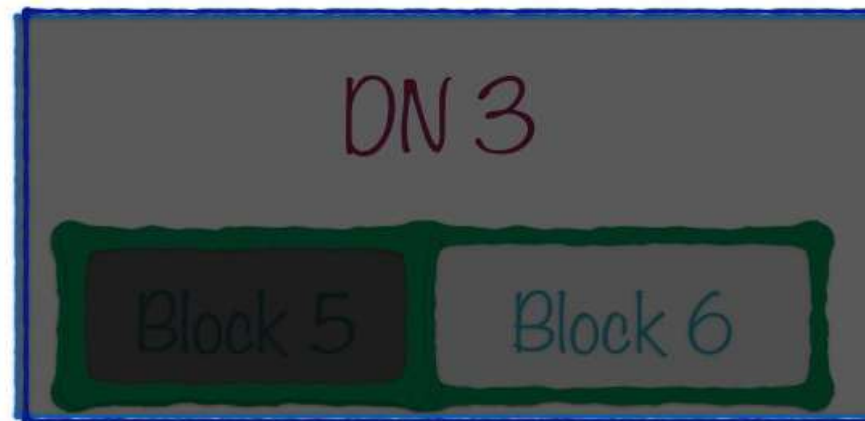
## Lendo um arquivo no HDFS



## Lendo um arquivo no HDFS

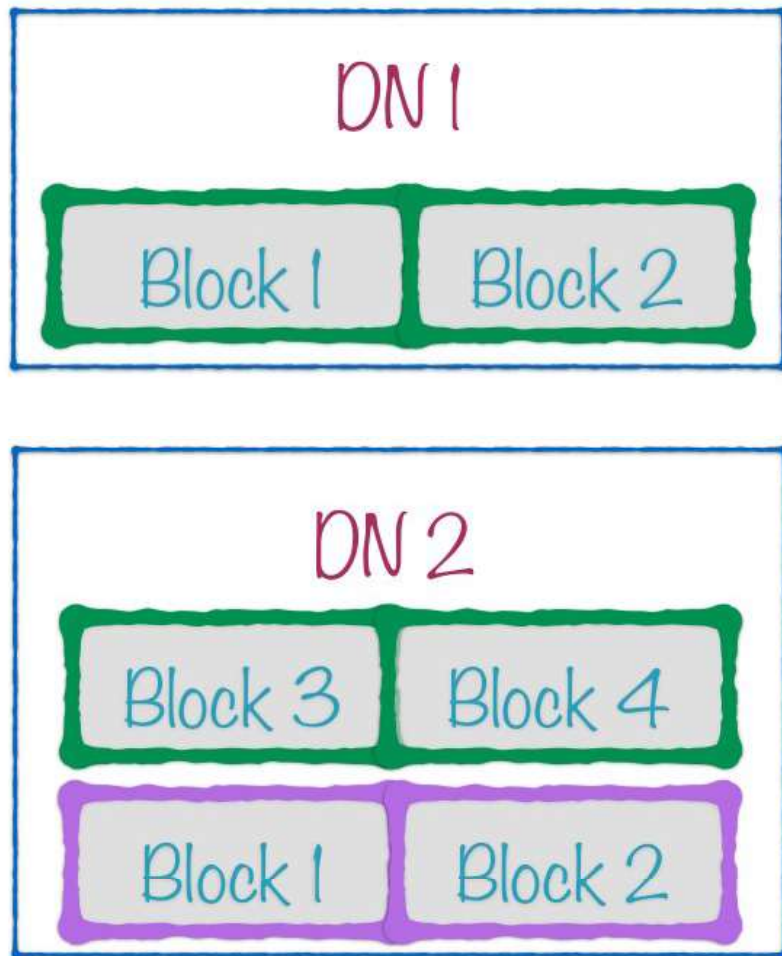


## Replicação no HDFS



O que é acontece se um dos blocos for corrompido???

## Replicação no HDFS



1. Replicar blocos baseados em um fator de replicação;
2. Armazenar replicas em diferentes locais;



## Replicação no HDFS

### Name node

File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3
File 1	Block 1	DN 2
File 1	Block 2	DN 2


A localização das réplicas  
são armazenadas no  
Namenode!

## Definindo o fator de replicação no HDFS

Definir o fator de replicação é um “trade off”



MAXIMIZAR  
REDUNDÂNCIA

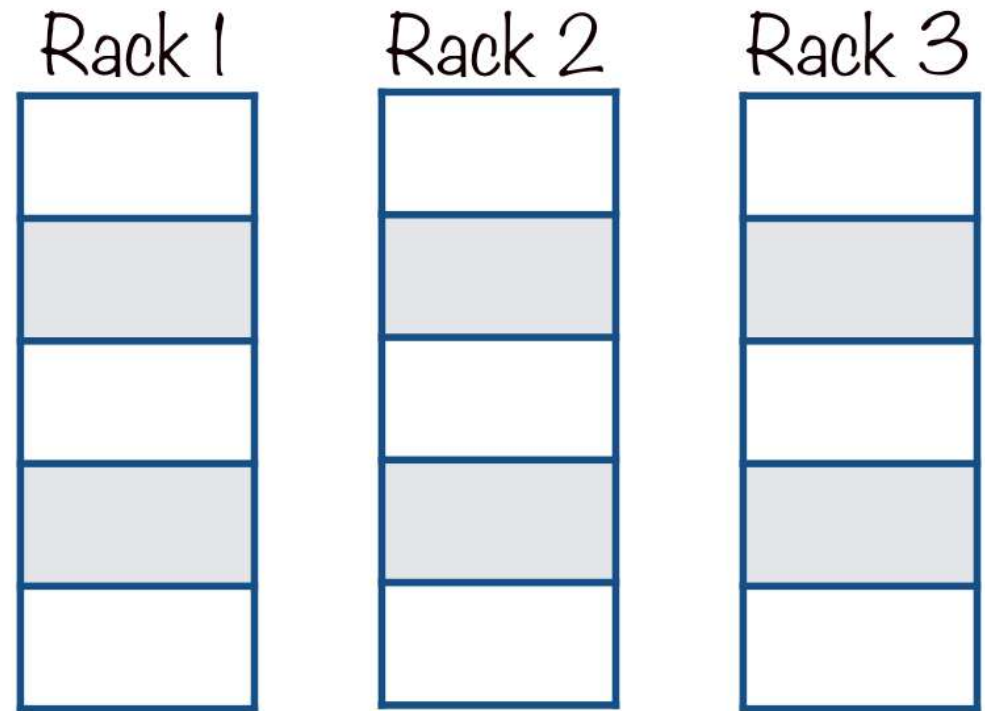


MINIMIZAR A  
LARGURA  
DE BANDA DE  
GRAVAÇÃO

## Definindo o fator de replicação no HDFS



MAXIMIZAR  
REDUNDÂNCIA

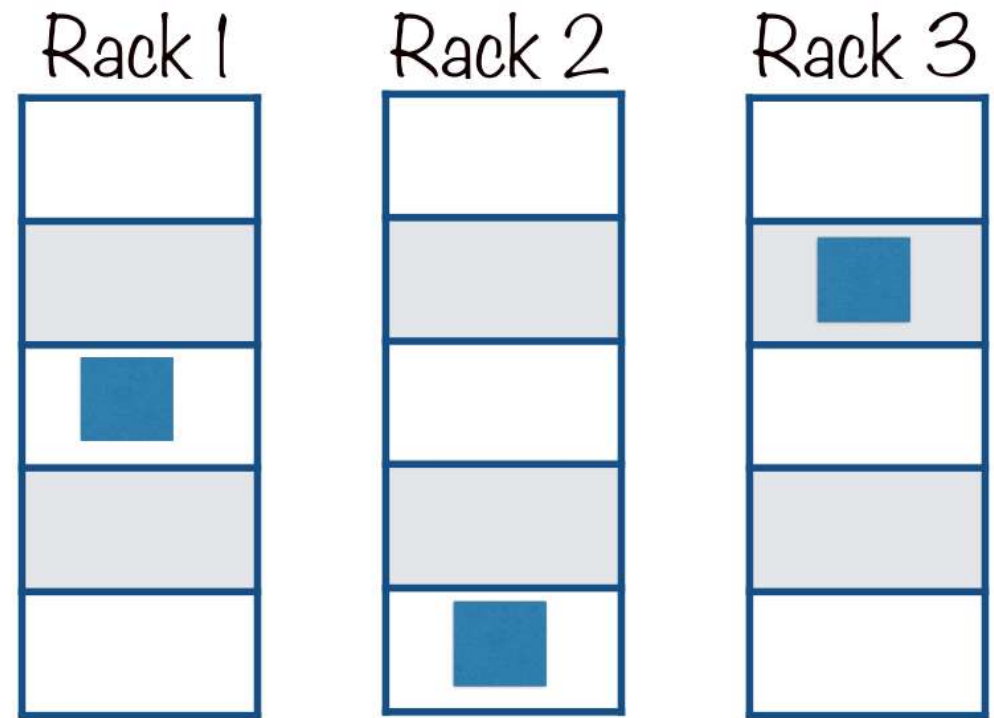


Servidores em um data center

## Definindo o fator de replicação no HDFS




MAXIMIZAR  
REDUNDÂNCIA



Armazena réplicas em  
diferentes nós

## Definindo o fator de replicação no HDFS



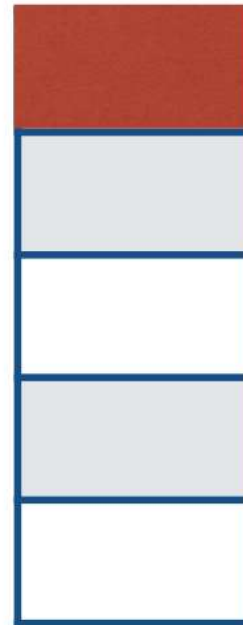
MINIMIZAR A  
LARGURA  
DE BANDA DE  
GRAVAÇÃO

Para minimizar a largura de banda de gravação é necessário que as réplicas fiquem próximas uma das outras.

## Definindo o fator de replicação no HDFS

MINIMIZAR A  
LARGURA  
DE BANDA DE  
GRAVAÇÃO

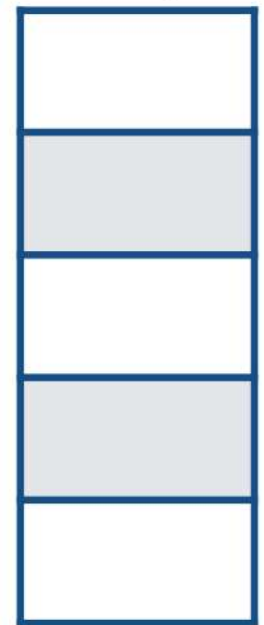
Rack 1



Rack 2



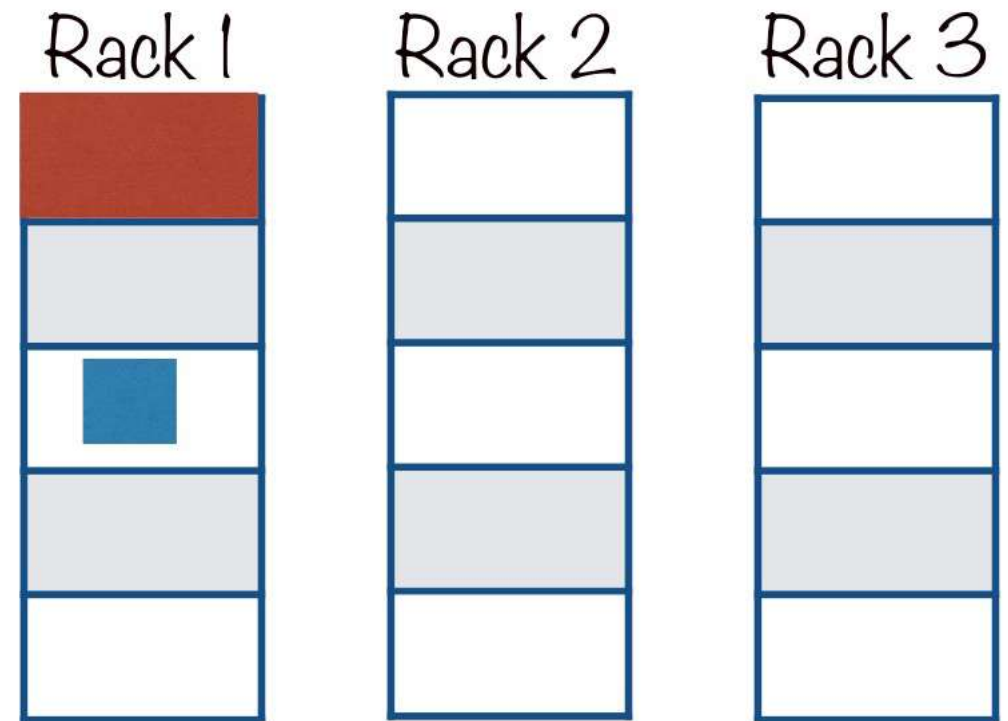
Rack 3



Nó que executa o pipeline de  
replicação

## Definindo o fator de replicação no HDFS

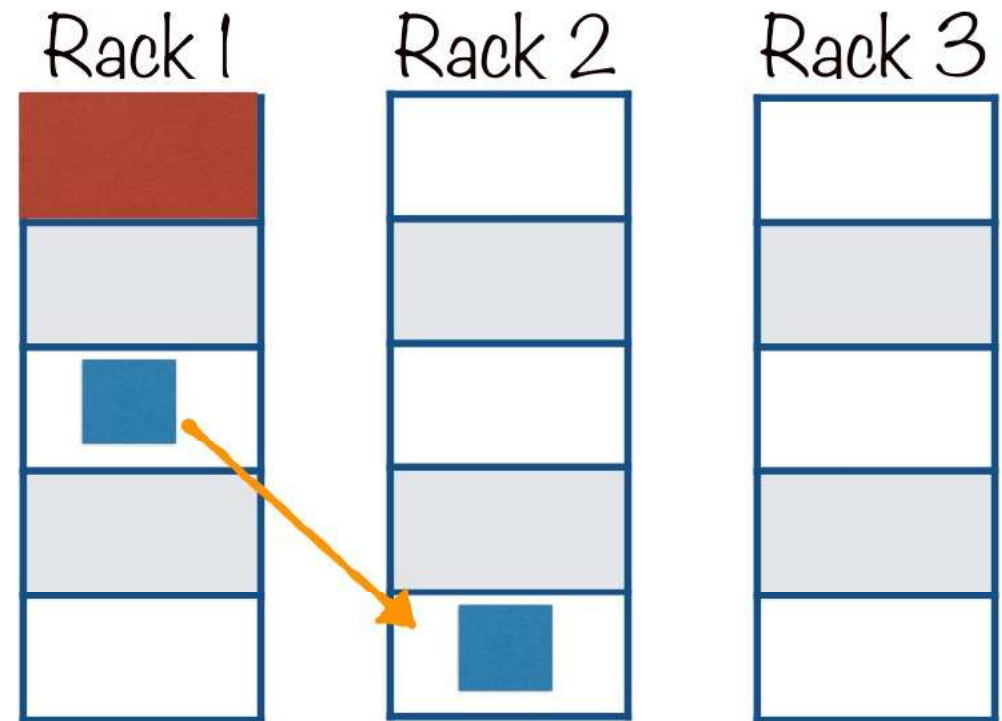
MINIMIZAR A  
LARGURA  
DE BANDA DE  
GRAVAÇÃO



Este nó escolhe qual será o  
local para a primeira réplica

## Definindo o fator de replicação no HDFS

MINIMIZAR A  
LARGURA  
DE BANDA DE  
GRAVAÇÃO

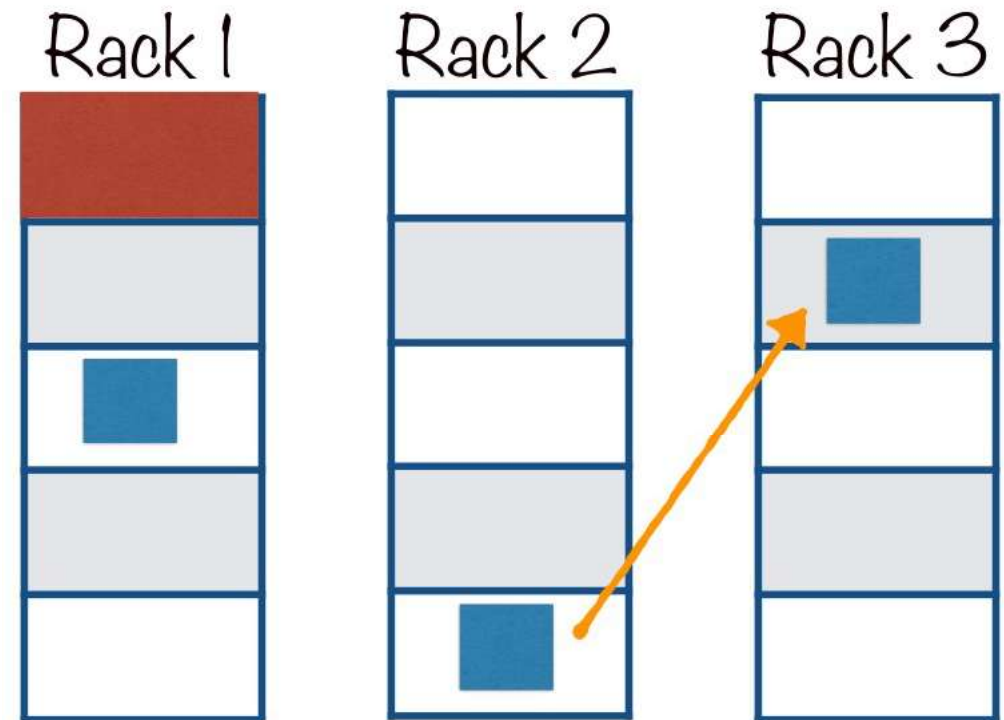


A réplica é encaminhada para  
um nó próximo...



## Definindo o fator de replicação no HDFS

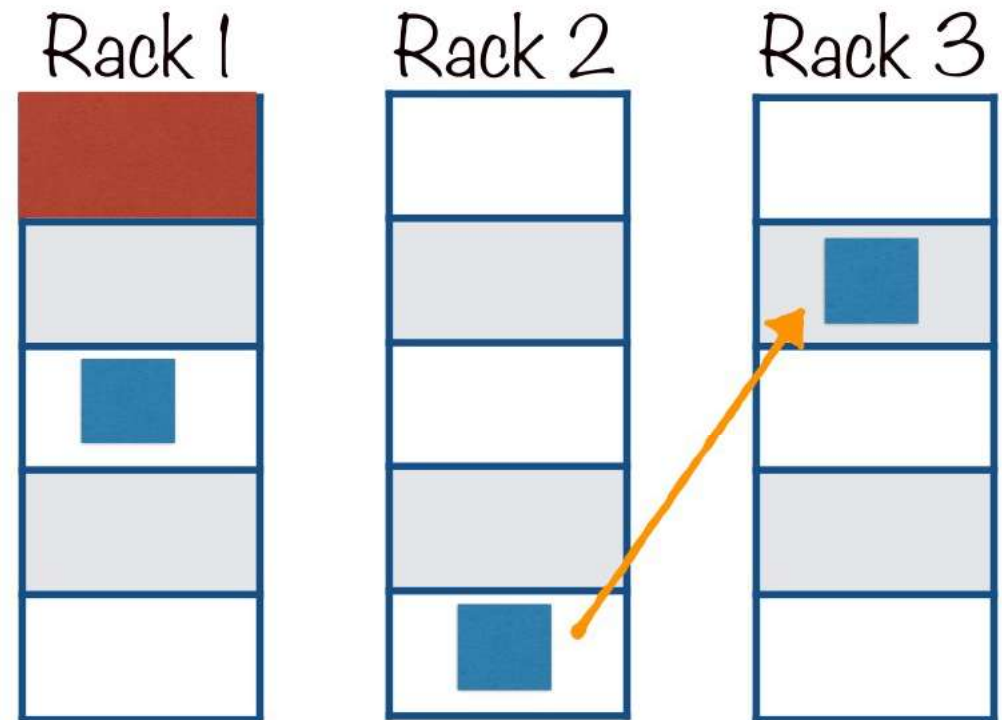
MINIMIZAR A  
LARGURA  
DE BANDA DE  
GRAVAÇÃO



Uma outra réplica é encaminhada  
para um nó próximo...

## Definindo o fator de replicação no HDFS

MINIMIZAR A  
LARGURA  
DE BANDA DE  
GRAVAÇÃO



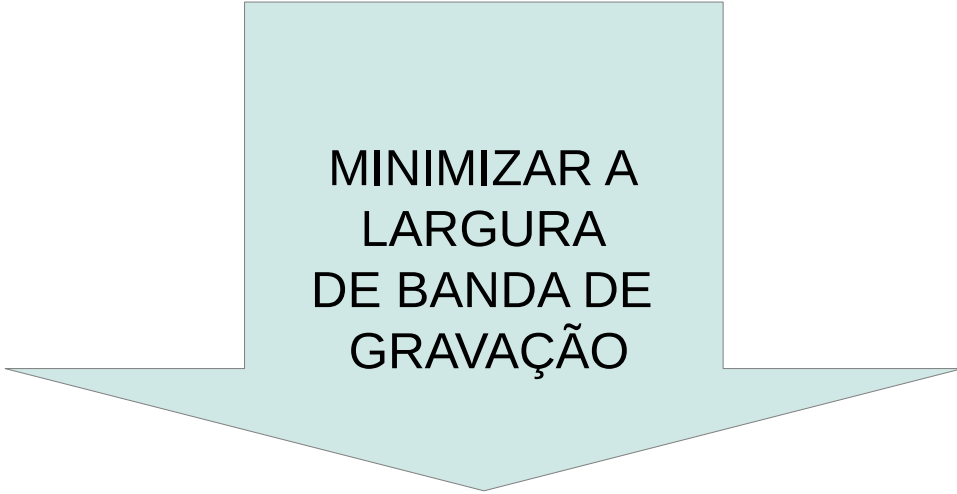
O encaminhamento requer uma  
largura de banda alta e aumenta o

## Definindo o fator de replicação no HDFS

É necessário, então, equilibrar as duas necessidades!

A light blue arrow pointing upwards, with a wide base and a narrow tip.

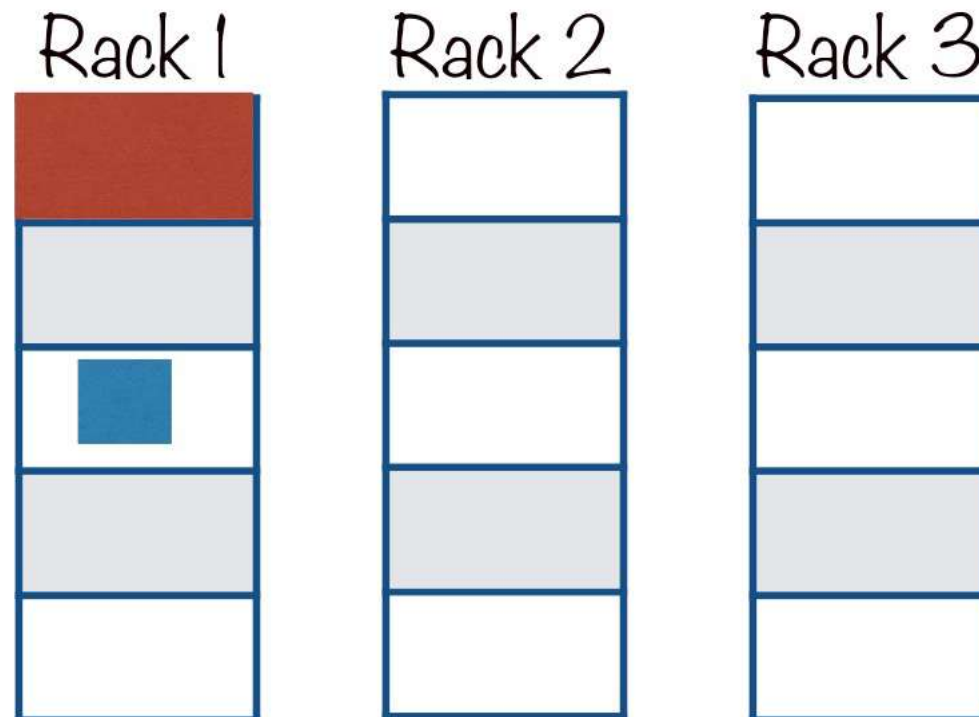
MAXIMIZAR  
REDUNDÂNCIA

A light blue arrow pointing downwards, with a wide base and a narrow tip.

MINIMIZAR A  
LARGURA  
DE BANDA DE  
GRAVAÇÃO

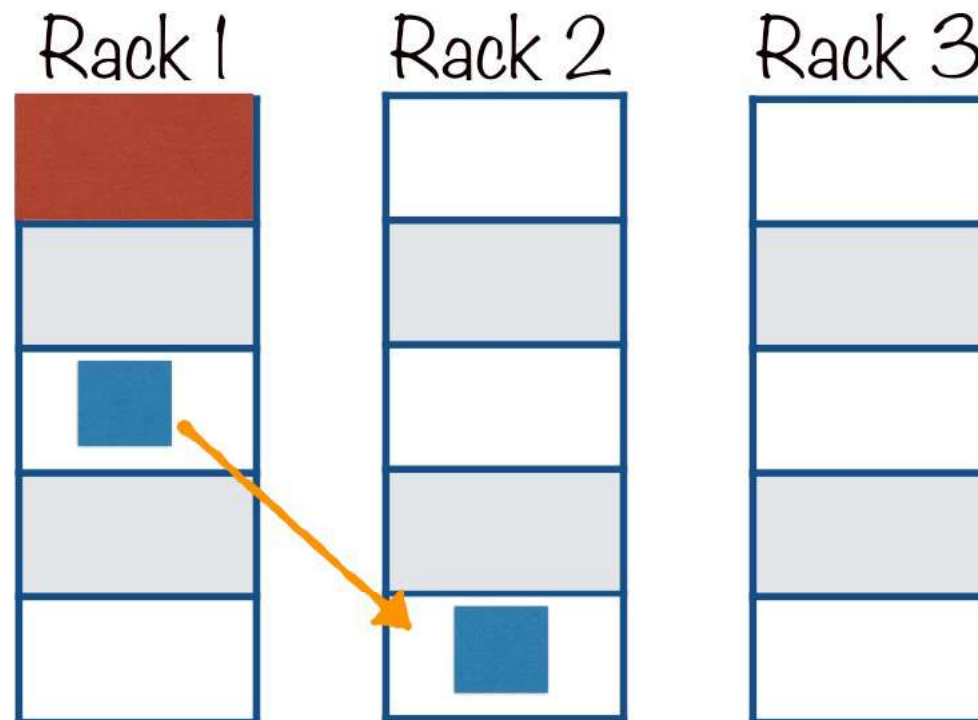
## Definindo o fator de replicação no HDFS

Escolher um local de armazenamento aleatoriamente!



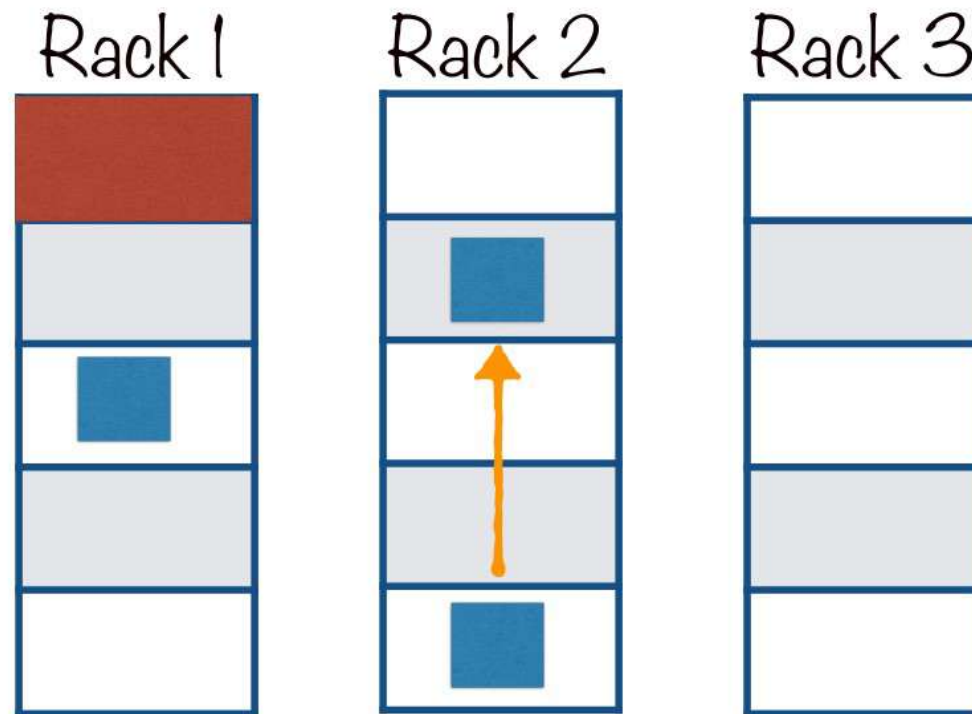
## Definindo o fator de replicação no HDFS

A segunda localização é preciso que fique em um rack diferente (se possível for...)

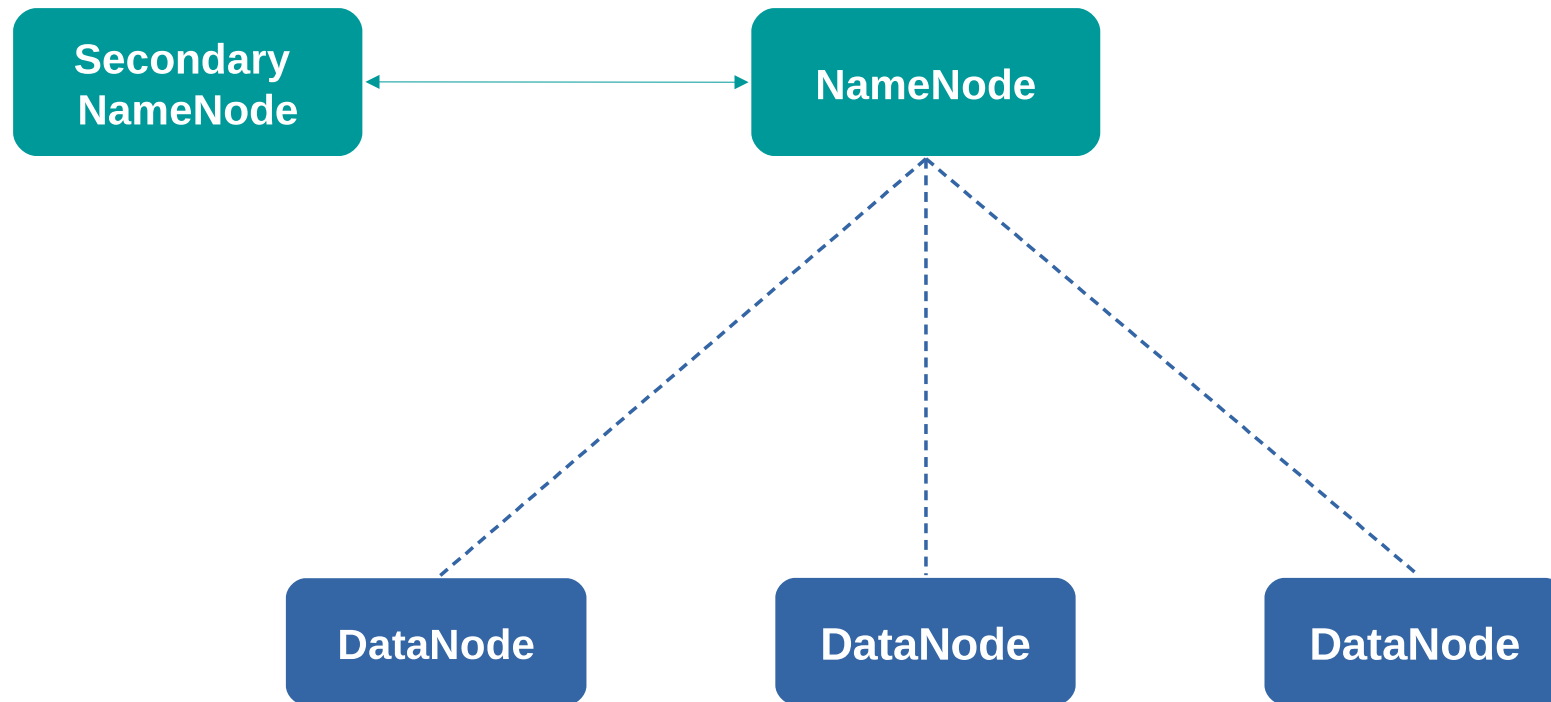


## Definindo o fator de replicação no HDFS

A terceira localização é interessante que fique no mesmo rack do segundo mas em um nó diferente...

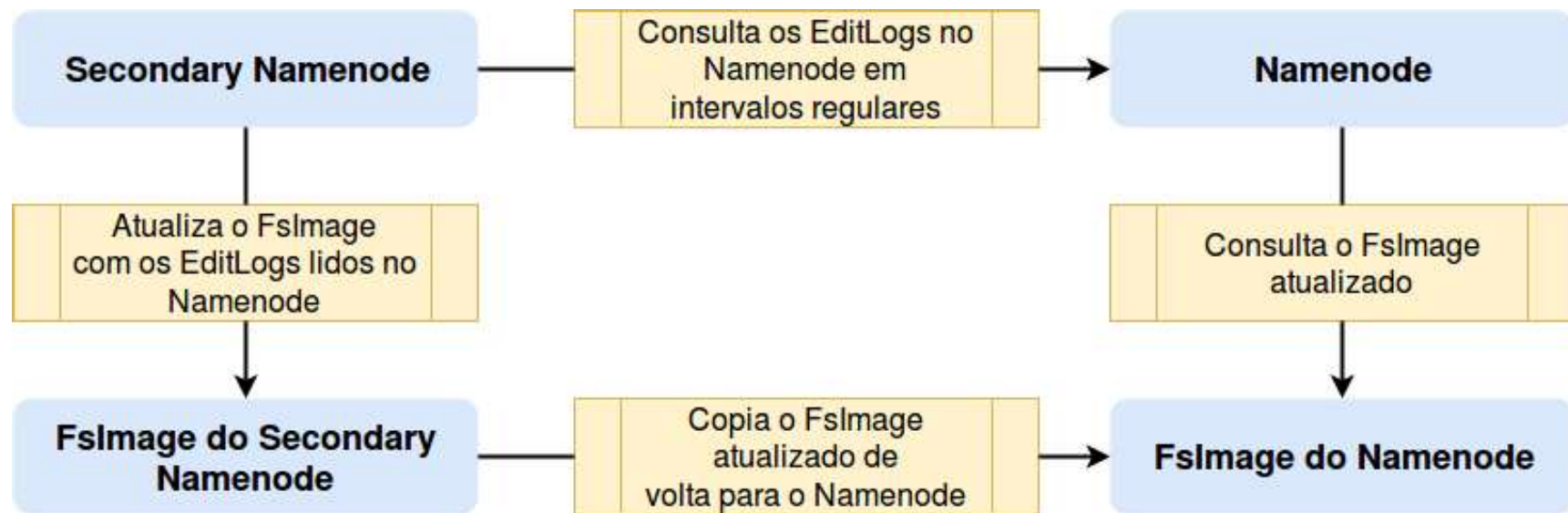


# Arquitetura HDFS





# Arquitetura HDFS





# MapReduce



# Workflow do MapReduce

**1**

Os dados são divididos em pequenos blocos e distribuídos pelo cluster Hadoop.

**2**

O MapReduce analisa os dados baseado nos pares de chave-valor.

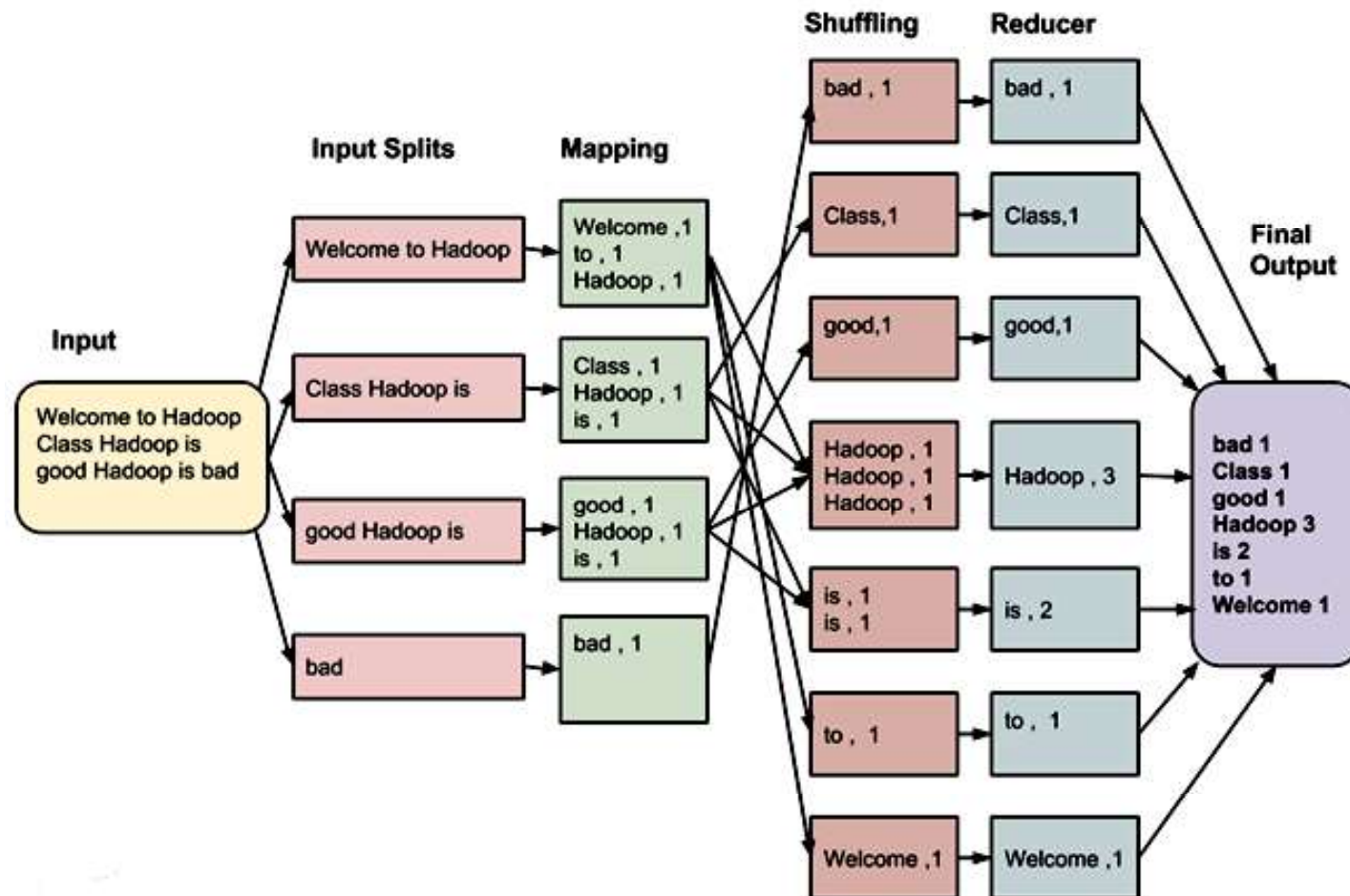
**3**

Os resultados são colocados em pequenos blocos através do cluster Hadoop.

**4**

Os resultados podem ser lidos no cluster.

# MapReduce



# MapReduce

## Funcionamento do MapReduce

### Mapper

NOME	IDADE	GENERO
guilherme	18	masculino
gabriela	22	feminino
antonio	30	masculino
guilherme	25	masculino
guilherme	27	masculino
gabriela	16	feminino

**MAPPER**

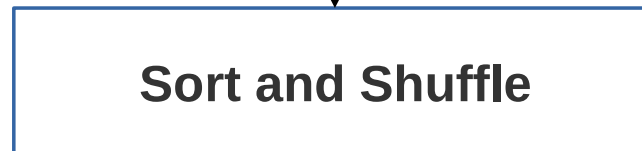
guilherme:18 gabriela:22 antonio:30 guilherme:25 gabriela:16 guilherme:27

# MapReduce

## Funcionamento do MapReduce

### Sort and Shuffle

guilherme:18 gabriela:22 antonio:30 guilherme:25 gabriela:16  
guilherme:27



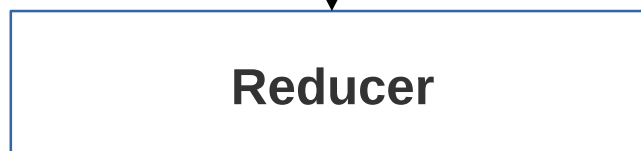
antonio:30 gabriela:22,16 guilherme:18,25,27

# MapReduce

## Funcionamento do MapReduce

### Reduce

antonio:30 gabriela:22,16 guilherme:18,25,27



antonio:1 gabriela:2  
guilherme:3



# MapReduce

## Vantagens do MapReduce

### **Processamento Paralelo**

No MapReduce, estamos dividindo o trabalho entre vários nós e cada nó funciona com uma unidade de processamento simultâneo. Assim, o MapReduce baseia-se no paradigma de programação “Dividir para conquistar” que nos ajuda a processar os dados usando diferentes máquinas ou processadores.

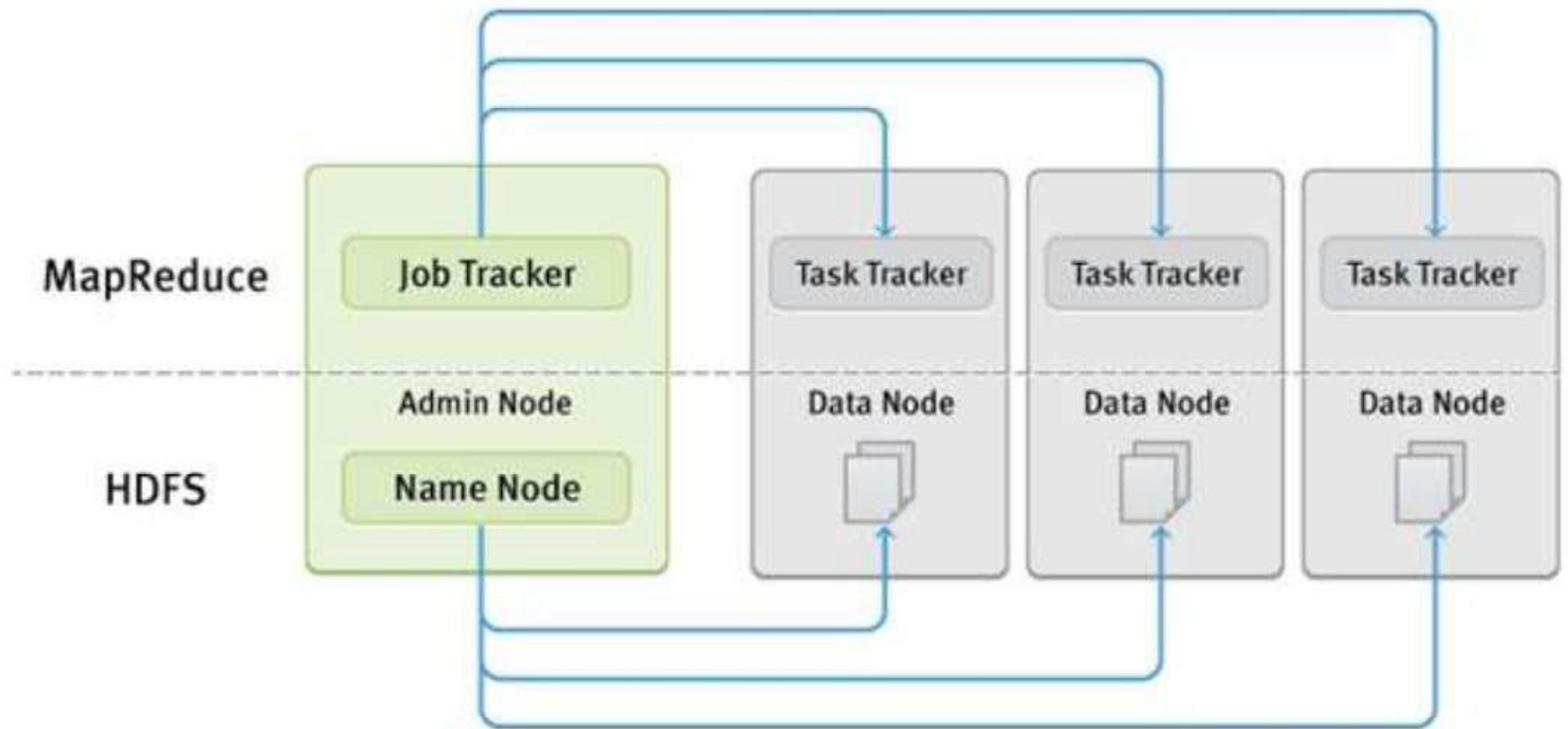
# MapReduce

## Vantagens do MapReduce

### Localidade de Dados

Em vez de mover dados para a unidade de processamento, estamos agora movendo a unidade de processamento para onde os dados estão. No sistema tradicional, costumávamos trazer dados para a unidade de processamento e, então, o processamento ocorria.

# Arquitetura HDFS e MapReduce



**Data Node:** Armazena/Recupera Dados

**TaskTracker:** Executa Jobs de MapReduce

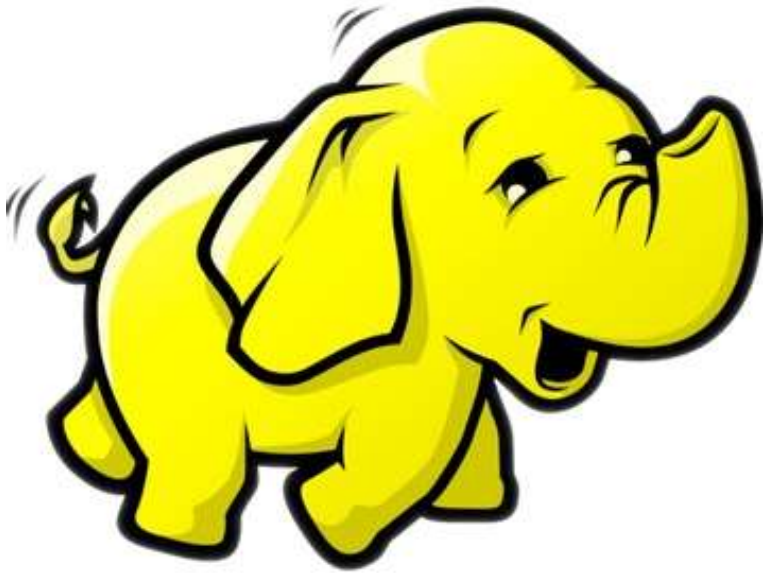


# YARN

## Yet Another Resource Negotiator



# YARN



- Coordena as tarefas em execução no cluster;
- Atribui novos nós em caso de falha;

# YARN

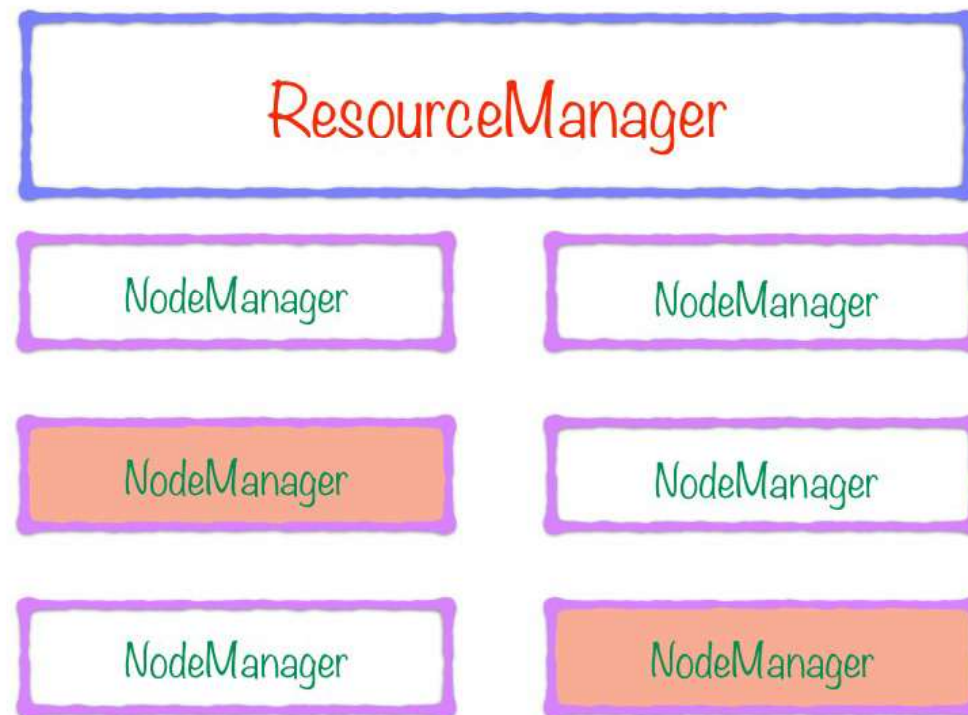
## RESOURCE MANAGER

- Executa em um único nó Master;
- Agenda tarefas nos nós;

## NODE MANAGER

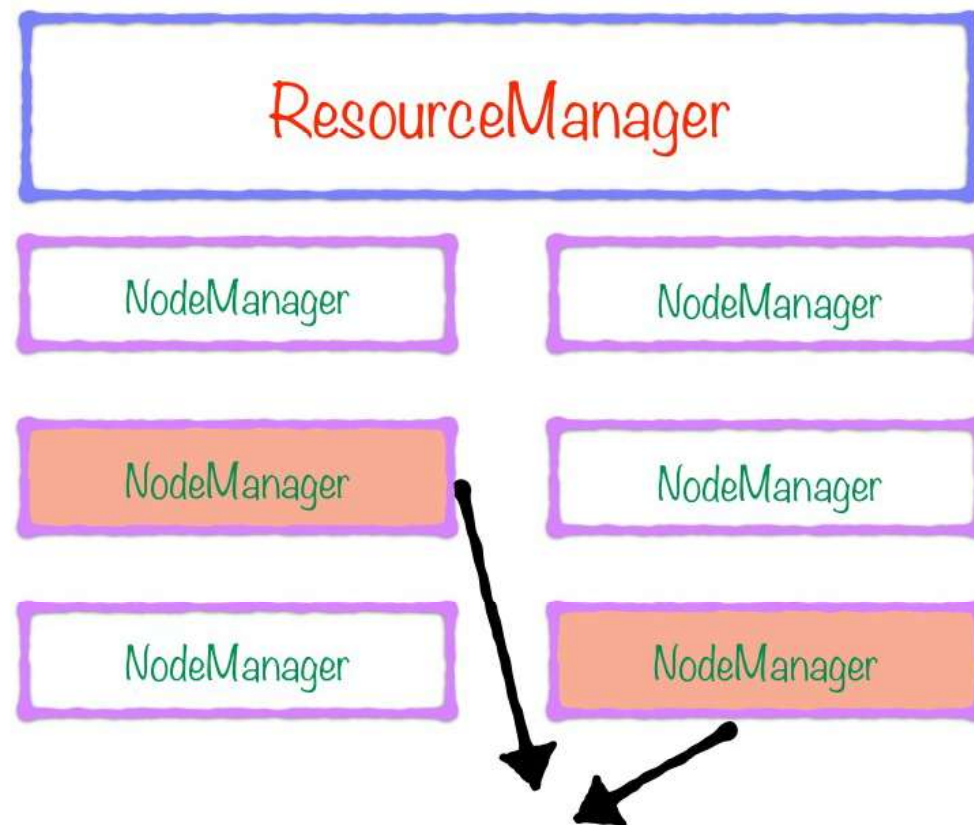
- Executa em todos os nós;
- Gerencia tarefas dentro de um nó;

# YARN



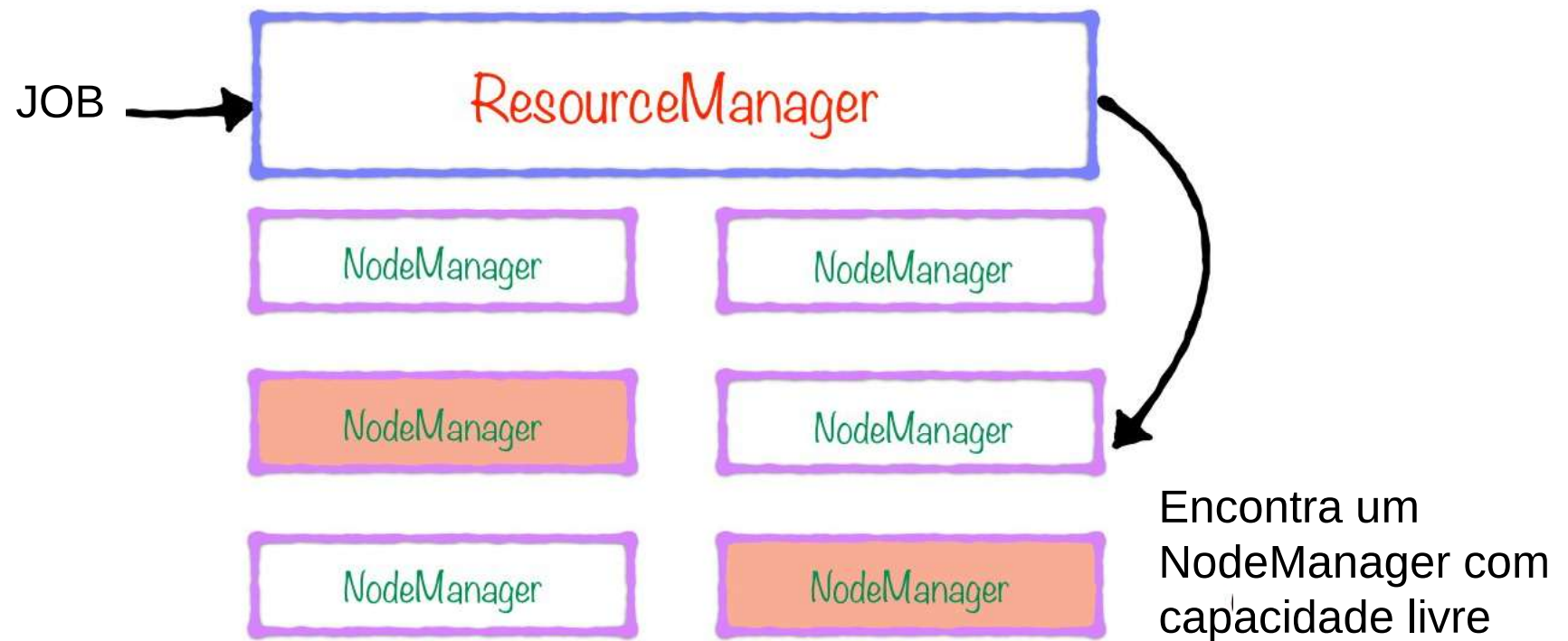


# YARN

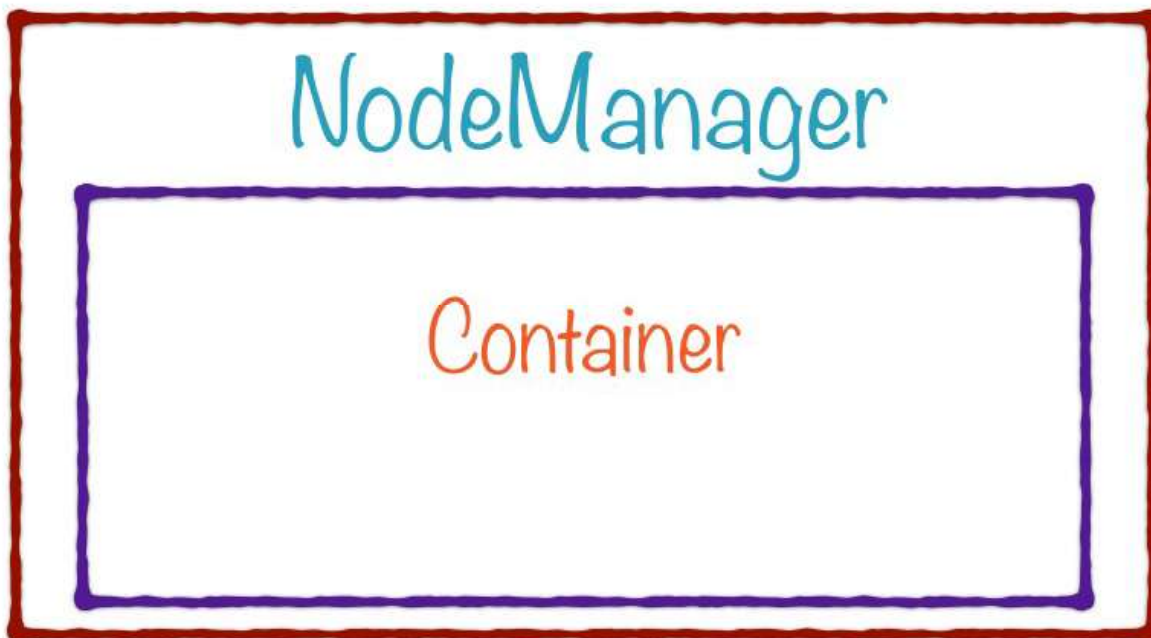


Já estão executando  
tarefas

# YARN



## YARN – Application Master Process



Todos os processos em um nó são executados dentro de um Container!

Esta é a unidade lógica para recursos que um processo necessita (memória, CPU etc);

Um container executa uma aplicação específica;

Um NodeManager pode ter múltiplos containers;

## YARN – Application Master Process



NodeManager

Container

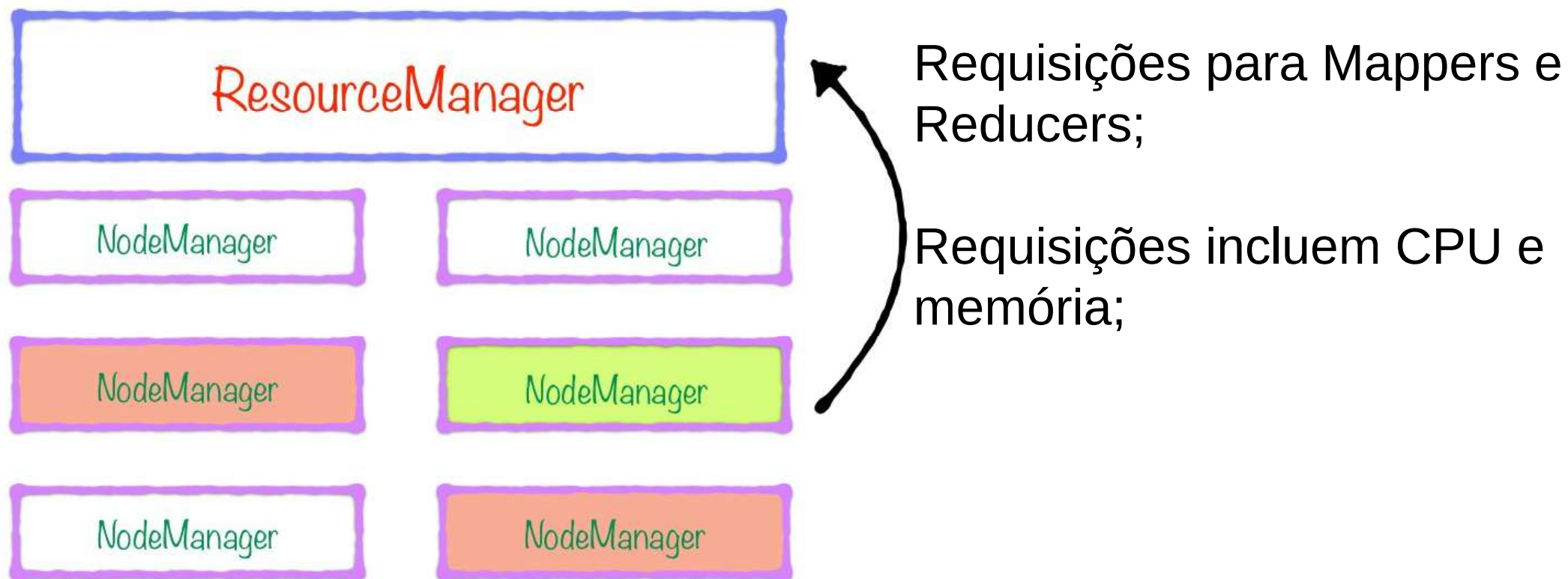
Application Master Process

O ResourceManager inicia a aplicação Master dentro de um container;

Executa a computação Requerida para a tarefa;

Se recursos adicionais são necessários, o Application Master faz a requisição.

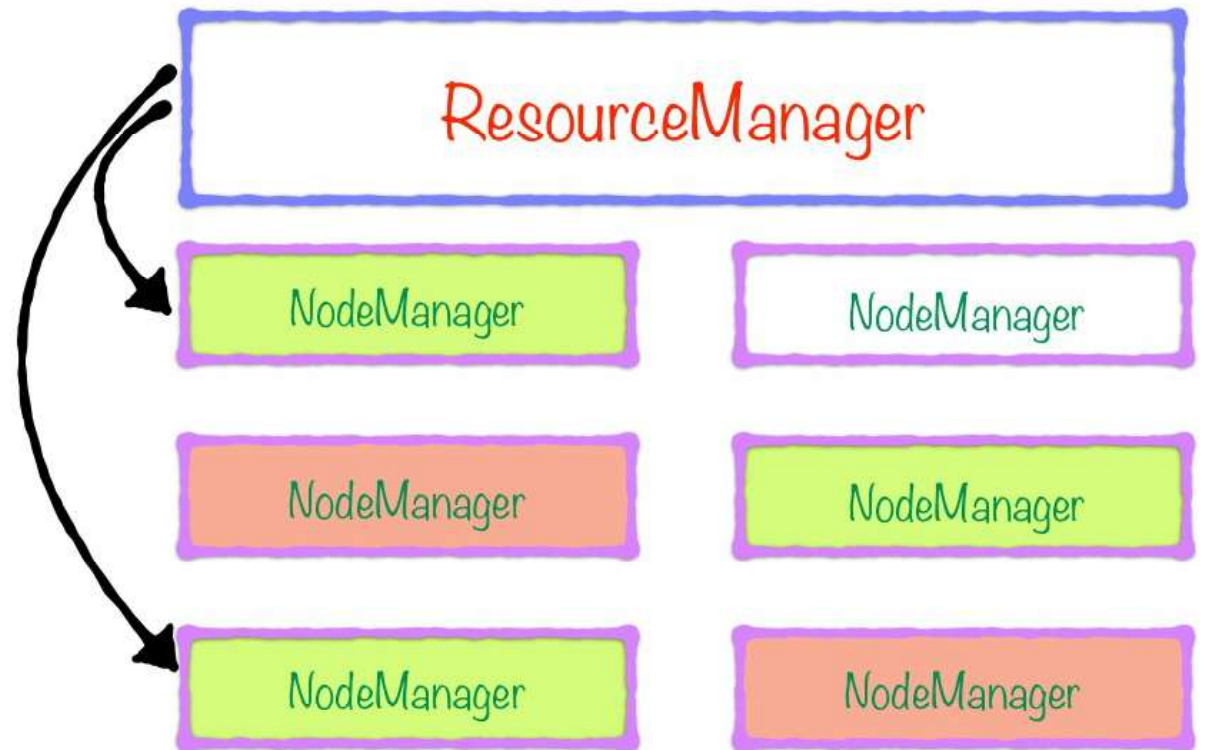
# YARN



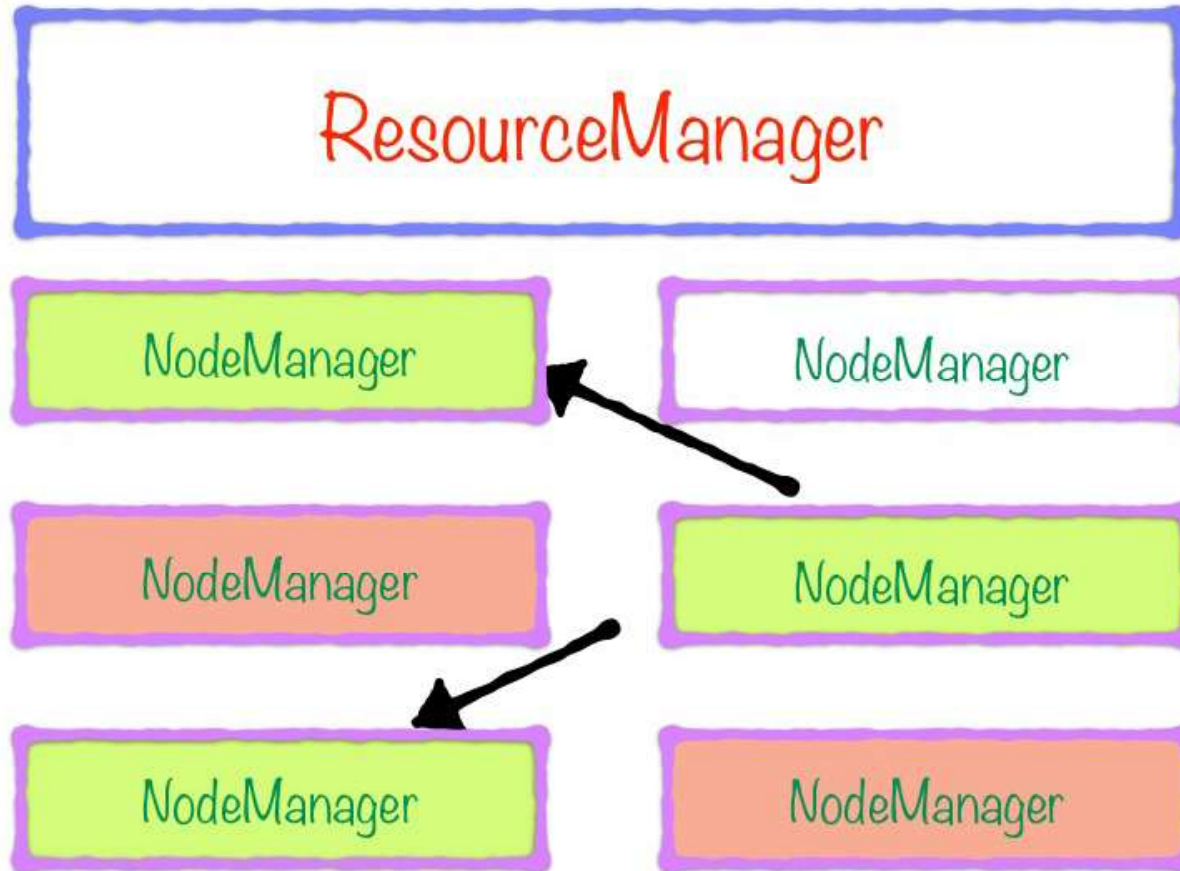
# YARN

Atribui nós adicionais;

Notifica o Application Master  
que fez a requisição



# YARN



O Application Master no  
nó original inicia os  
Applications Masters em  
novos  
nós atribuídos;

**Fim**