With great [computational] power comes great [algorithmic] responsibility
—David Keyes

Big Data e Computação em Nuvem

Aula 02

Map Reduce

Introdução ao Spark

Prof. Michel Fornaciali, PhD.
Prof. Thanuci Silva, PhD.

Contatos:
MichelSF@insper.edu.br
thanucis@insper.edu.br

Insper

Na aula anterior...

- Apresentação da disciplina

- Escalabilidade
- Paralelização
- Programação distribuída (Dask)

Agenda

- Estratégia MapReduce
- Funcionamento do Spark
- RDDs (*Resilient Distributed Dataset*)

Processamento em Larga Escala

Aspectos Teóricos, Técnicos e a Estratégia Map Reduce

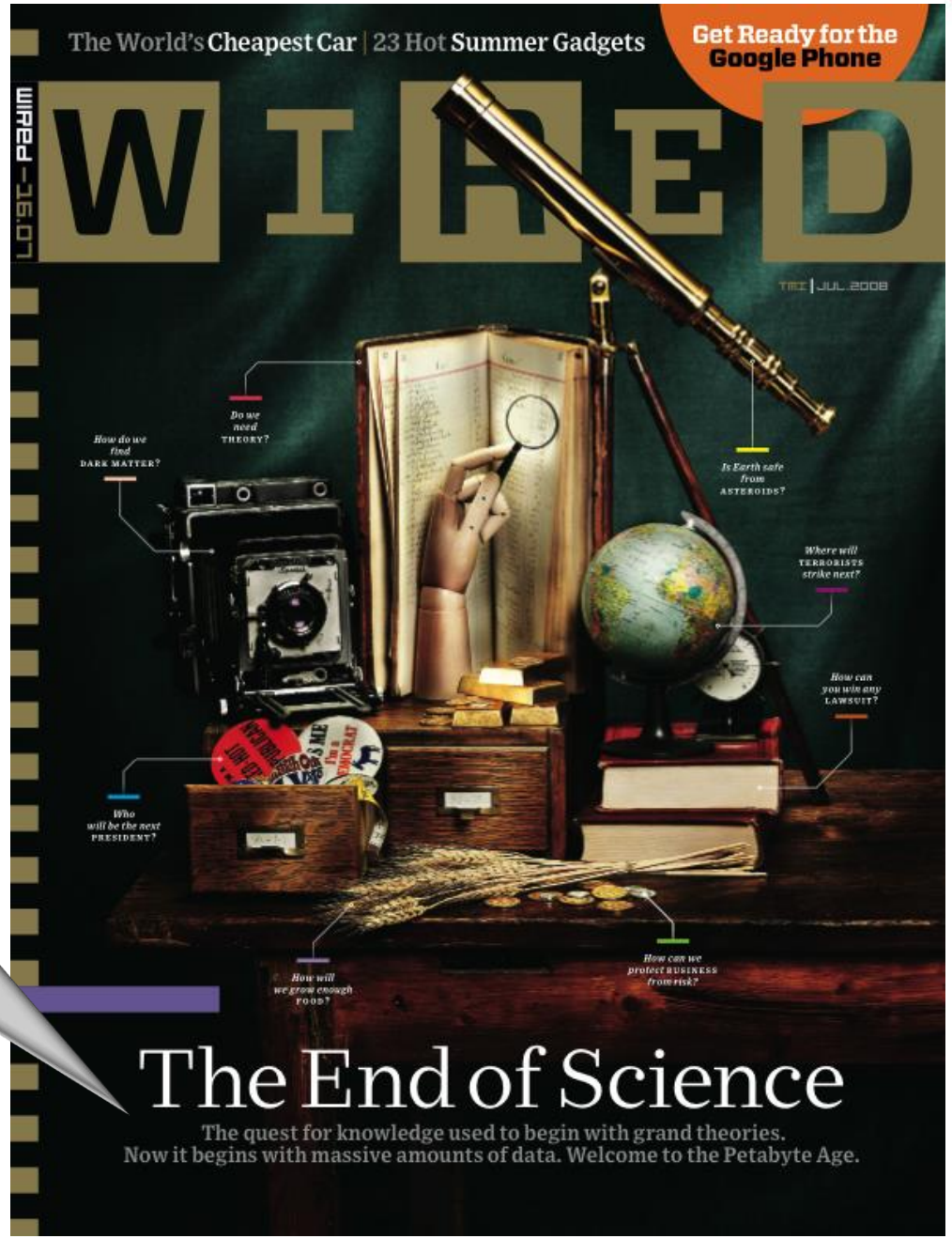
Introdução ao Apache Spark

Contextualização

The quest for knowledge used to begin with grand theories.

Now it begins with massive amounts of data.

Welcome to the Petabyte Age.



Insper

O que discutimos anteriormente

- Uma única máquina não consegue executar a computação necessária
  - Desempenho
  - Tempo
  - Memória

- Vimos também uma forma de **dividir o problema em partes menores**, preferencialmente paralelizáveis, que possam ser executadas de uma maneira **distribuída** (i.e., múltiplas máquinas), possibilitando um aumento do desempenho computacional (speed-up)

- A essa estratégia dá-se o nome de Dividir para Conquistar!

Dividir para Conquistar!

Lemuel Gulliver is welcomed to the Land of the Lilliputians

Imagem: https://gulliverslastvoyage.com/2017/02/22/about-the-last-voyage-of-lemuel-gulliver/

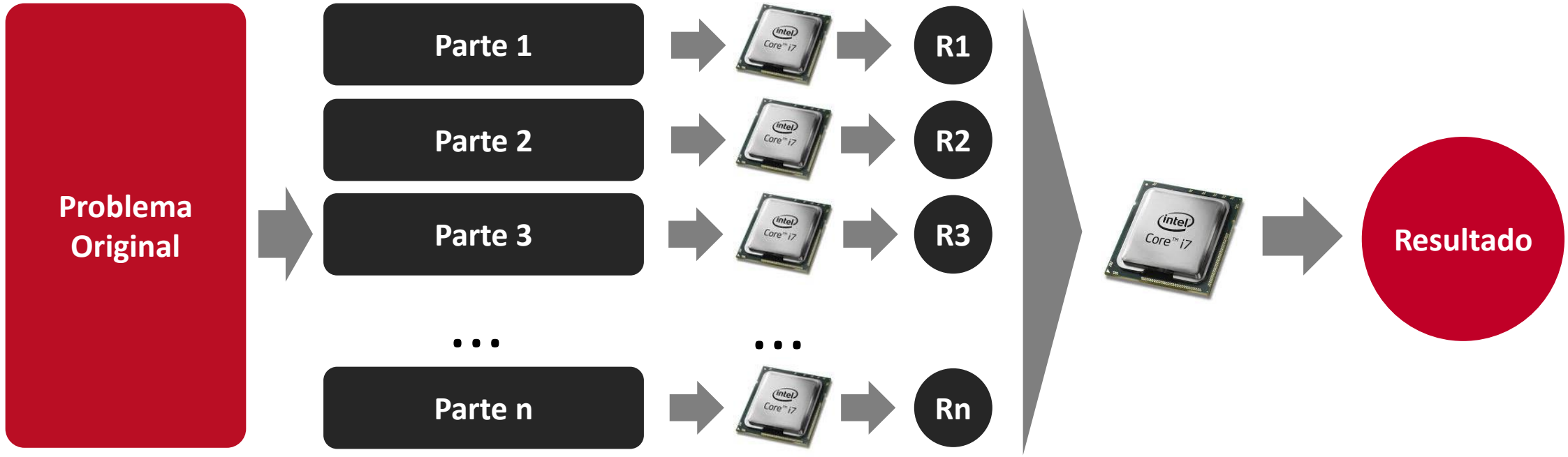
Insper

Questões que surgem

- Habitualmente quando iniciamos a trabalhar com computação distribuída temos as seguintes dúvidas:
  - Como eu faço a distribuição de um algoritmo?
  - Como eu particiono os meus dados?
  - Como eu mantenho uma visão consistente sobre a computação em execução?
  - Como posso recuperar o processamento se uma máquina falhar?
  - Como eu aloco os recursos de maneira ótima?

Ao paralelizar o processamento com Python, utilizamos uma estratégia de escalabilidade horizontal.

Estratégia de Paralelização



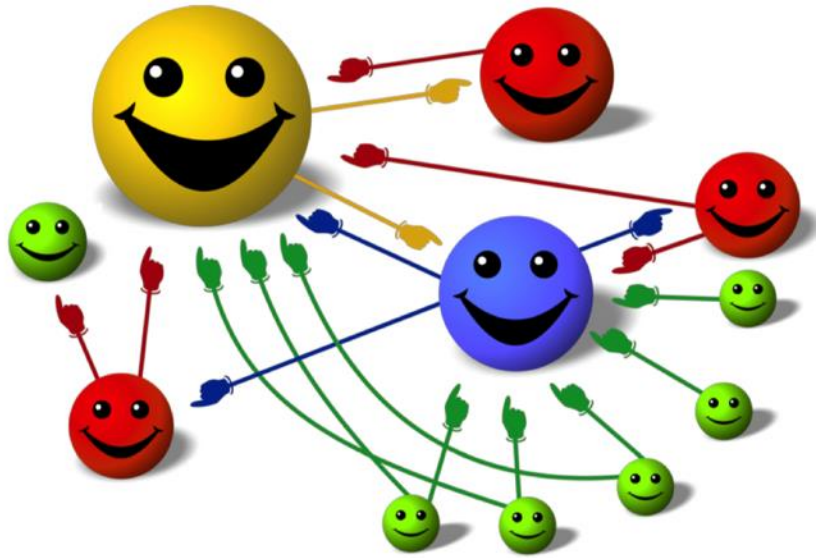
Planejamento	Mapeamento	Redução
$P \rightarrow p_1, p_2, p_3, ..., p_n$	$R_n = map(p_n)$	$R = reduce(R_1,R_2,...,R_n)$

Insper

MapReduce

Onde tudo começou: Google

- Algoritmo desenvolvido pelos fundadores da Google (Larry Page, Sergey Brin)
- Base para o surgimento do motor de buscas Google
- Trata-se de uma nota dada pelo Google a páginas indexadas em sua base

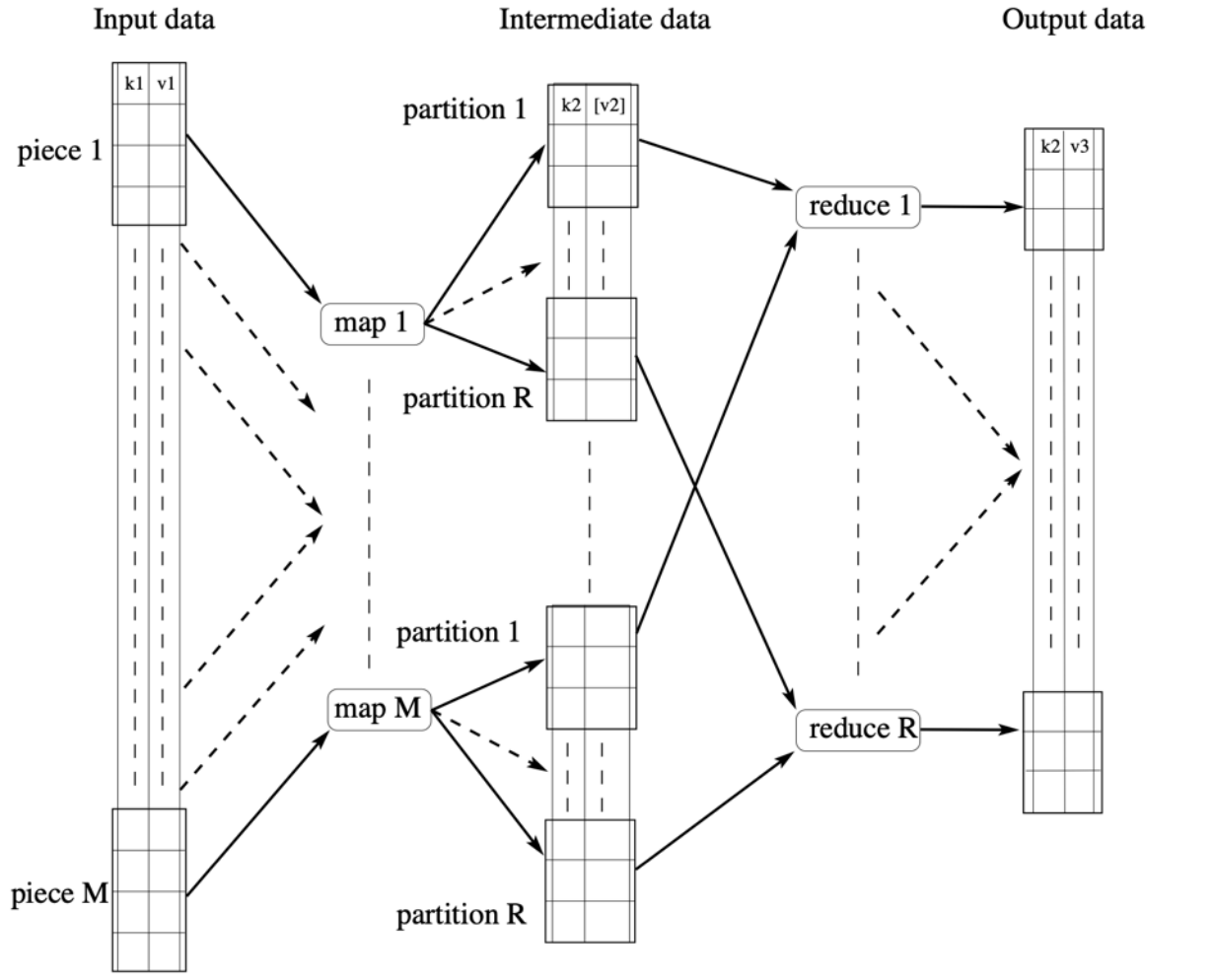


- O conceito é muito simples:
  - Se uma página é referenciada por outras páginas, maior será o seu score
  - Se uma página for referenciada por uma outra página de alto score, maior será o seu score

- A indexação do Google tem como um de seus passos iniciais a contagem das palavras nas páginas

MapReduce idealizado pela Google

- Modelo inicial proposto pela Google
- Consiste na paralização da computação em um aglomerado de máquinas comuns
- Modelo de programação:
  1. Leia uma grande quantidade dados
  2. Aplique uma função MAP
  3. Fase intermediária: Shuffle & Sort
  4. Aplique a função REDUCE
  5. Grave os resultados

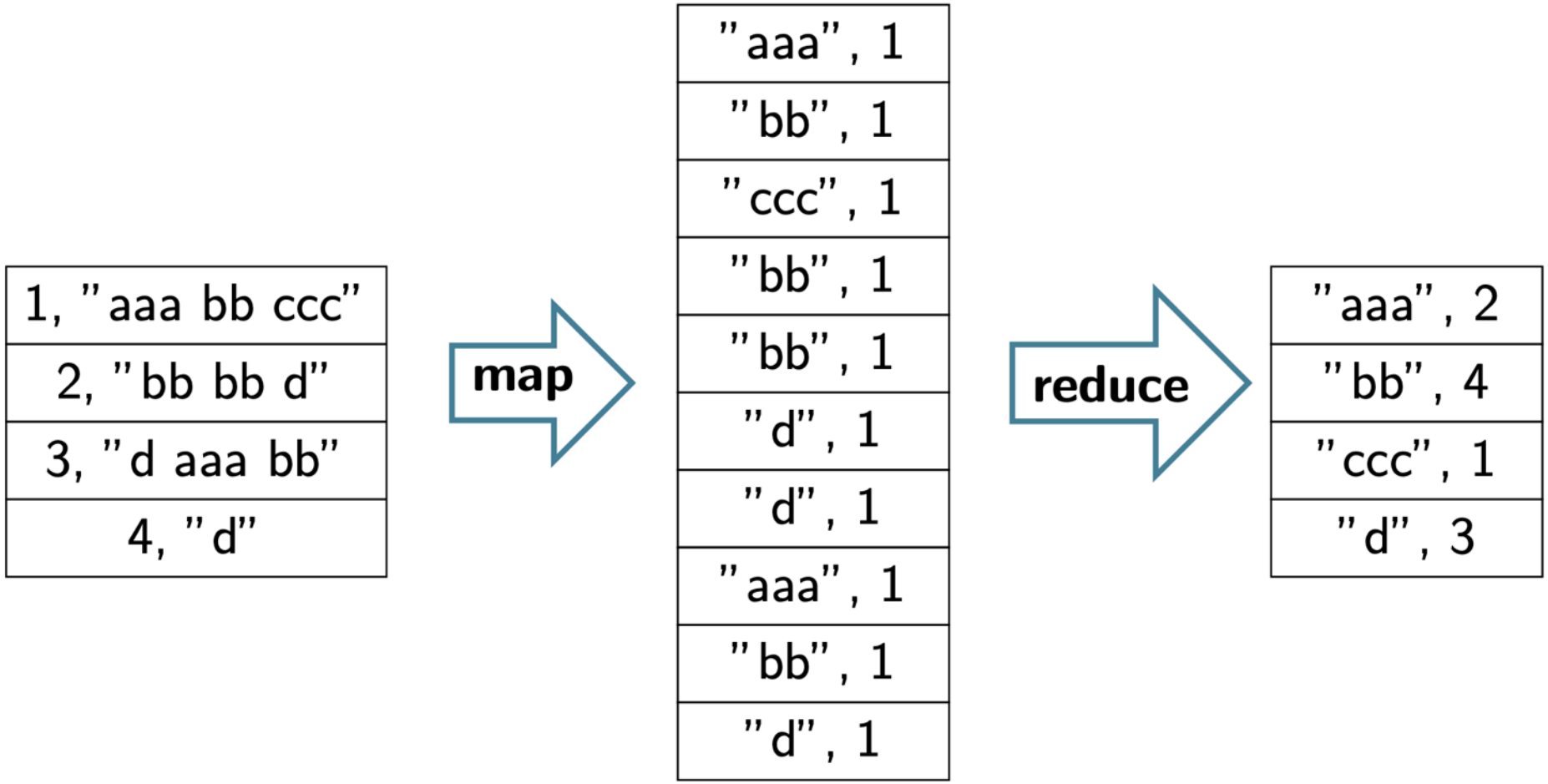


No MapReduce, o programa vai até os dados, e não o contrário.

Insper

MapReduce: Word Count

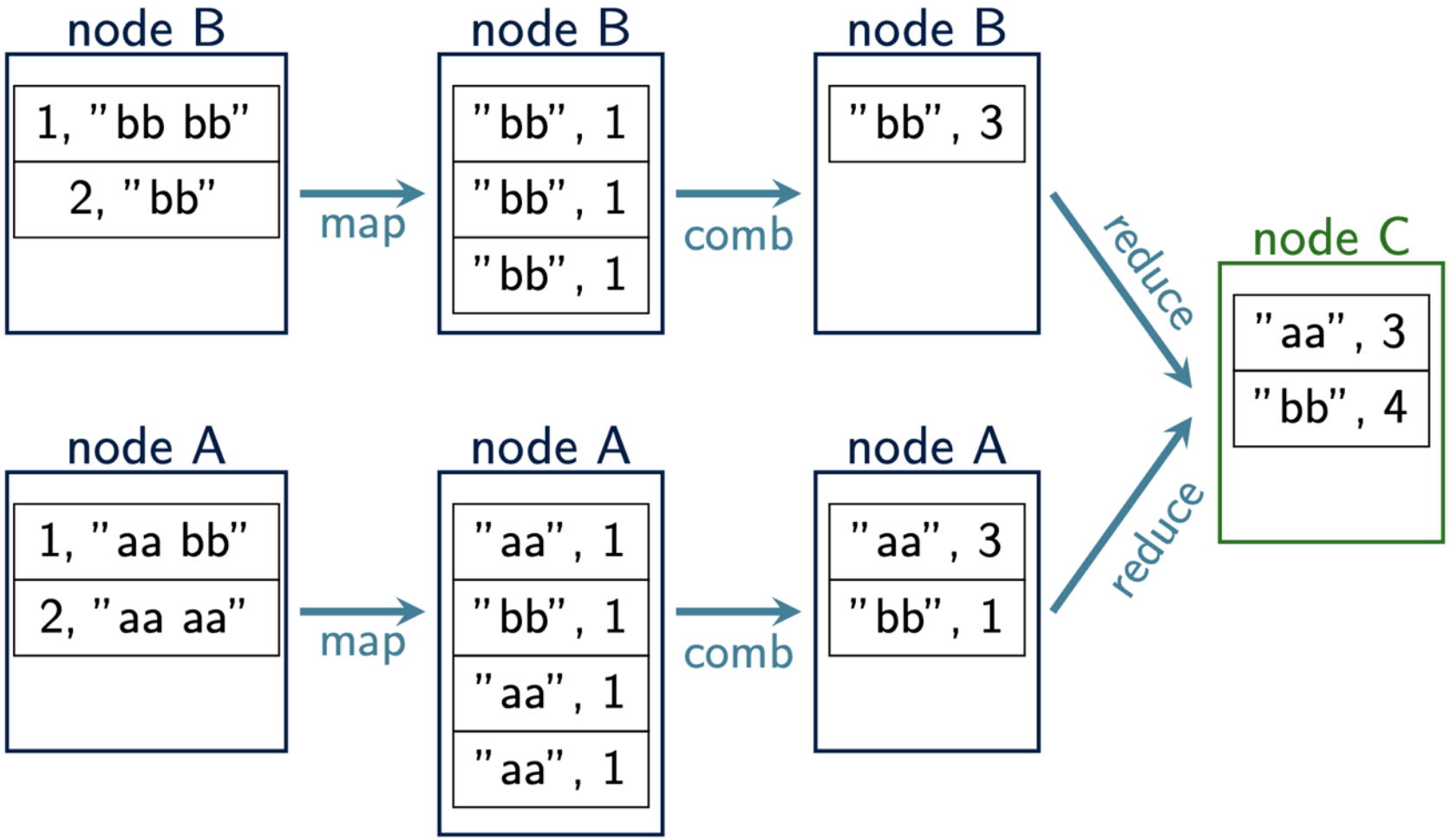
Representação Lógica



Insper

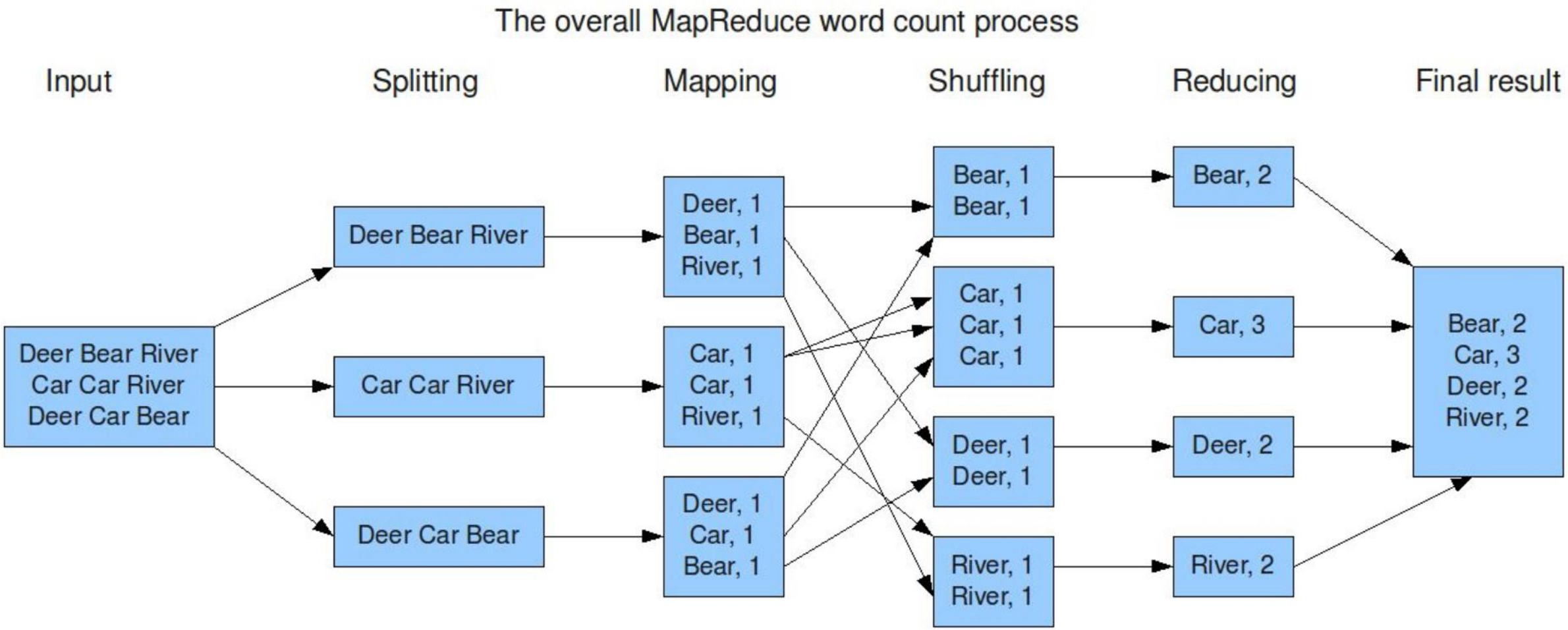
MapReduce: Word Count

Representação Distribuída



Insper

MapReduce – Processo Completo



Insper

A estratégia Map Reduce não é nova

Presente há mais de 40 anos em linguagens funcionais

- Map em programação funcional:

map ({1, 2, 3, 4} , x^2) → {2, 4, 9, 16}

```
1 lista = [1, 2, 3, 4]
2 m = map(lambda x : x*x, lista)
3
4 print(lista, '-->', list(m))
```

```
[1, 2, 3, 4] --> [1, 4, 9, 16]
```

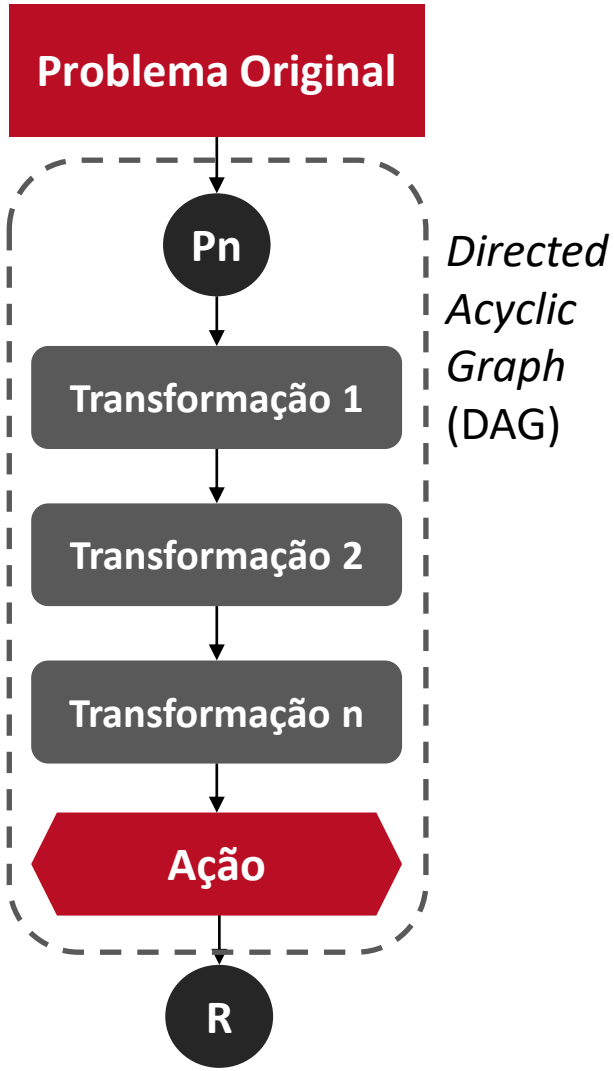
reduce ({1, 2, 3, 4} , x + y) → {10}

```
1 lista = [1, 2, 3, 4]
2 reduce(lambda x,y : x + y, lista)
```

```
10
```

A distribuição resiliente de um algoritmo exige a manutenção das propriedades comutativa e associativa nas ações aplicadas.

Requisitos para o Processamento Paralelo



- **Propriedade Comutativa:** a ordem dos fatores não altera o produto (resultado):
  - D = A + B + C = B + C + A
- **Propriedade Associativa:** a associação dos fatores não modifica o produto:
  - D = (A + B) + C = A + (B + C)
- **Fechamento da programação funcional**:
  - Além das propriedades acima, exige-se que as funções utilizadas sejam fechadas, isso é, contenham seu escopo total encapsulado.

Insper

Exemplo Prático

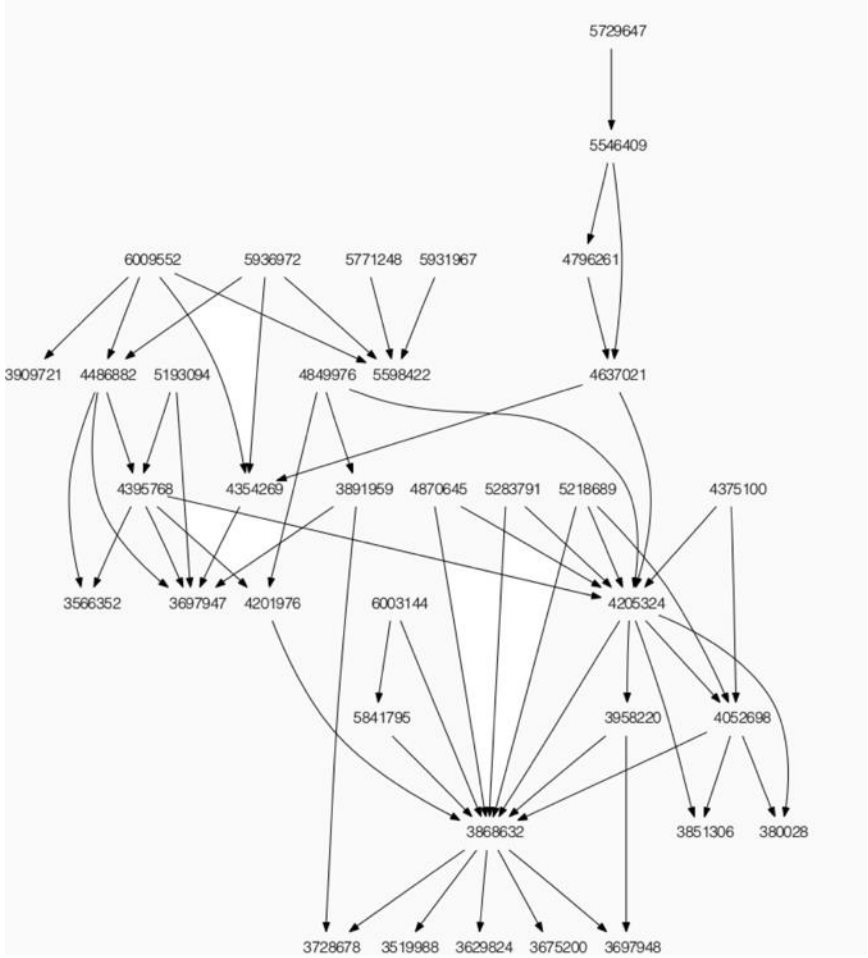
Citações em patentes

- Dados do US National Bureau of Economic Research (www.nber.org/patents - Cite75_99.txt)
- Lista os IDs das patentes e outras patentes citadas por esta:

"CITING", "CITED"
3858241, 956203
3858241, 1324234
3858242, 1515701
3858244, 956203

- Objetivo: contar quantas vezes uma patente foi citada
- Saída desejada:

956203, 2
1515701, 1
1324234, 1



Insper

Exemplo Prático

Citações em patentes

- Map: <inteiro, inteiro> → Lista(<inteiro, inteiro>)

Map Entrada	Map Saída
<3858241, 956203>	[<956203, 1>]
<3858241, 1324234>	[<1324234,1>]

- Reduce: <inteiro, Lista(inteiros)> → List(<inteiro, inteiro>)

Map Entrada	Map Saída
<956203, [1, 1] >	[<956203, 2>]
<1324234, [1] >	[<1324234,1>]

"CITING", "CITED"
3858241, 956203
3858241, 1324234
3858242, 1515701
3858244, 956203

956203, 2
1515701, 1
1324234, 1

Hadoop

Timeline

- 2003 – Google publica artigo do Google File System
- 2004 – Google publica artigo do MapReduce
- 2005 – Doug Cutting cria uma versão do MapReduce denominada Hadoop
- 2006 - Hadoop se torna um subprojeto do Apache Lucene
- 2007 – Yahoo! se torna o maior patrocinador e utilizador do projeto
- 2008 – Hadoop deixa a tutela do Lucene e torna-se um projeto top-level
- 2010 – Facebook anuncia maior aglomerado de Hadoop do mundo (2.900 nós e 30 petabytes de dados)

Hadoop

Case – The New York Times

- Em 2007, o jornal The New York Times converteu para PDF todos os artigos publicados entre 1951 e 1980
- Cada artigo é composto por várias imavens previamente digitalizadas que devem ser redimensionadas de forma coerente para criação do PDF
- 4 TB de imagens TIFF em 11 milhões de arquivos PDF
- 100 instâncias EC2 da Amazon foram utilizadas durante 24 horas para gerar 1,5 TB de arquivos PDF, a um custo de aproximadamente US$ 240,00.

A software system programmer at the days, thought this was a perfect chance to use the Amazon web Services (AWS) and Hadoop. Storing and serving the final set of PDFs from Amazon's straightforward Storage Service (S3) was already deemed a less expensive approach than scaling up the storage back-end of the web site. Why not method the PDFs within the AWS cloud as well?

The 4 TB of run-in pictures into S3. He "started writing code to drag all the components that structure an article out of S3, generate a PDF from them and store the PDF back in S3. This was straightforward enough using the JetS3t —Open supply Java toolkit for S3, iText PDF Library and putting in the Java Advanced Image Extension ."1 when tweaking his code to figure among the Hadoop framework, Derek deployed it to Hadoop running on 100 nodes in Amazon's Elastic cipher Cloud (EC2). the duty ran for 24 hours and generated another 1.5 TB of data to be hold on in S3.

At 10 cents per instance per hour, the full job complete up cost accounting only $240 (100 instances x 24 hours x $0.10) in computation. The storage value for S3 was additional, however because the Times had set to archive its files in S3 anyway, that value was already amortized. Data transfer between S3 and EC2 being free, the Hadoop job didn't incur any bandwidth value in the least.

BIG DATA & HADOOP
Learn by Example

by
Mayank Bhushan

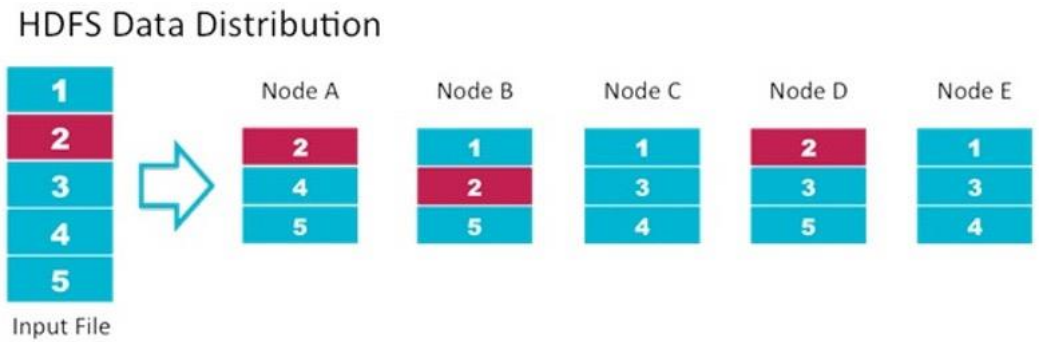
Créditos: Adaptado de Daniel Cordeiro / USP

BPB PUBLICATIONS

Insper

Hadoop

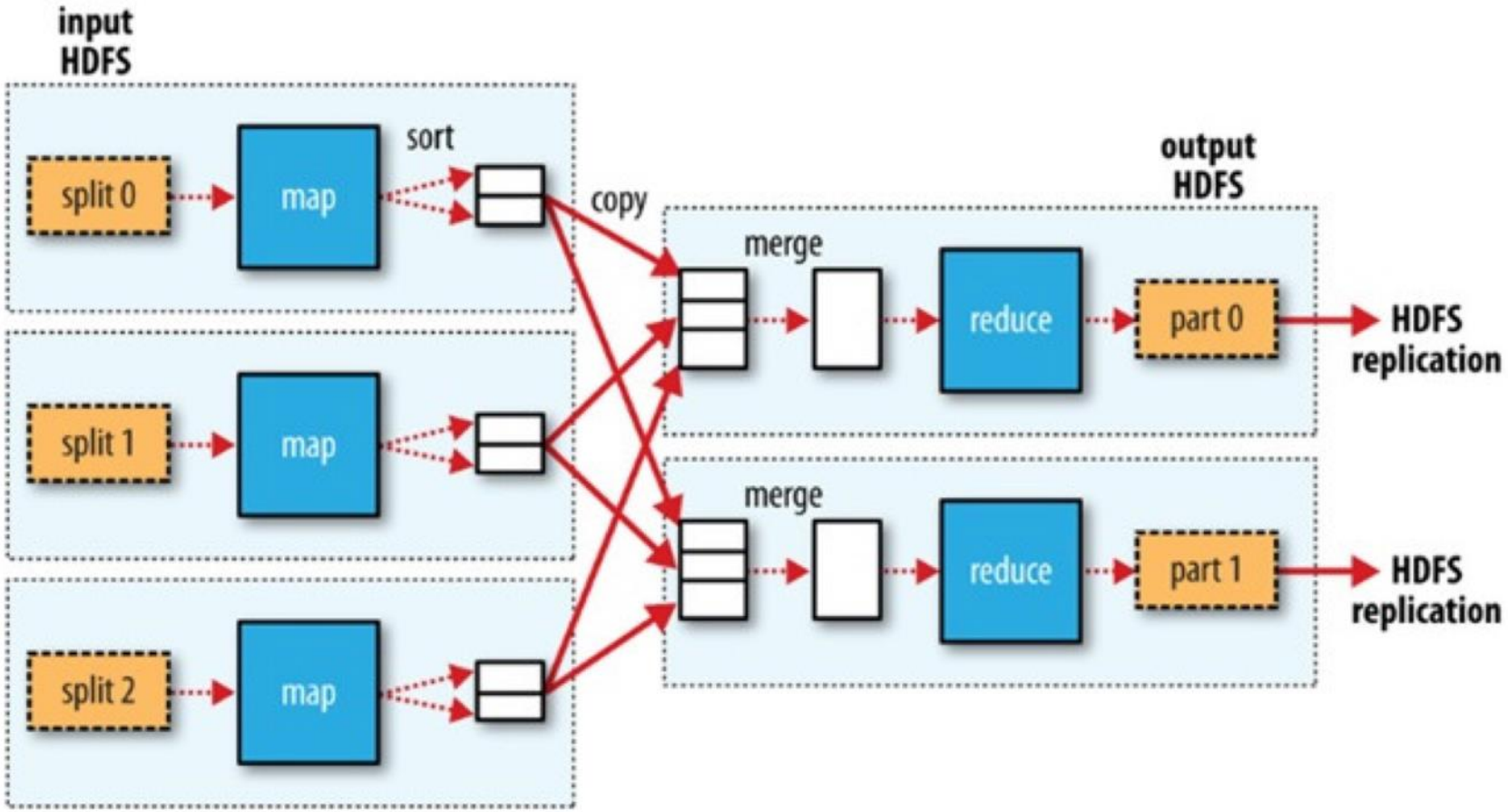
- Composto por dois componentes principais:
  - HDFS – sistema de arquivos distribuído



  - MapReduce
    - Responsável pelo processamento de dados armazenados nesse sistema de arquivo

Insper

MapReduce no Hadoop com HDFS



Insper

Hadoop

Arquitetura clássica

- Muitas leituras/escritas no disco enquanto processa

Insper

Hadoop

Arquitetura clássica

- Desejo:
  - Será que é possível manter o máximo possível os dados na memória?

  - Discussão: essa abordagem é mais eficiente?

Insper

Spark

- Originalmente desenvolvido na Universidade da California
  - *Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, M. Zaharia et al. NSDI, 2012.*
- Um dos projetos mais populares de Big Data na atualidade

- Sua principal vantagem é buscar realizar um processamento massivo em memória sempre que possível
- MapReduce não é um algoritmo para Stream processing. Consegue imaginar o porquê?
  - Spark possui funcionalidades que permitem o processamento de dados em real time!

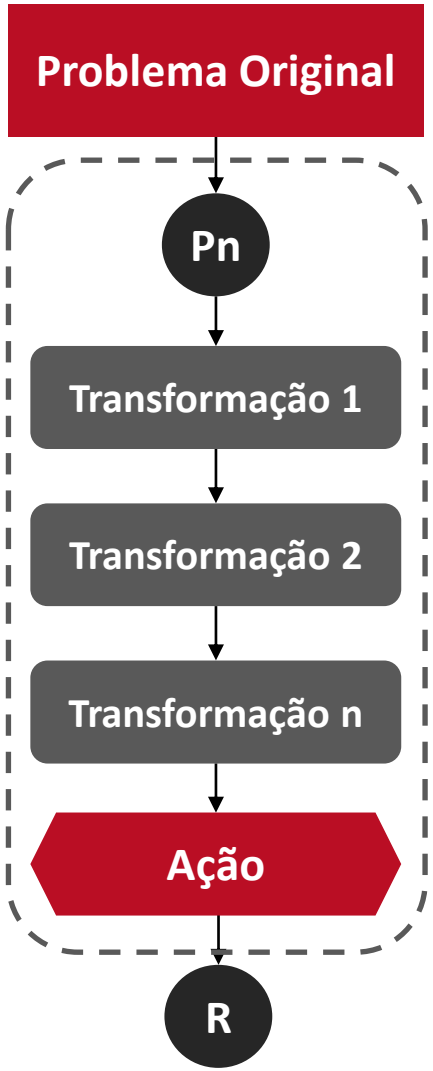
A estrutura básica de dados do Spark são os conjuntos de dados resilientes distribuídos (RDDs).

Resilient Distributed Datasets

RDD:
Conjunto de objetos

Transformações:
Aplicadas a cada elemento da RDD

Ações:
Aplicadas a todos os elementos das RDDs.



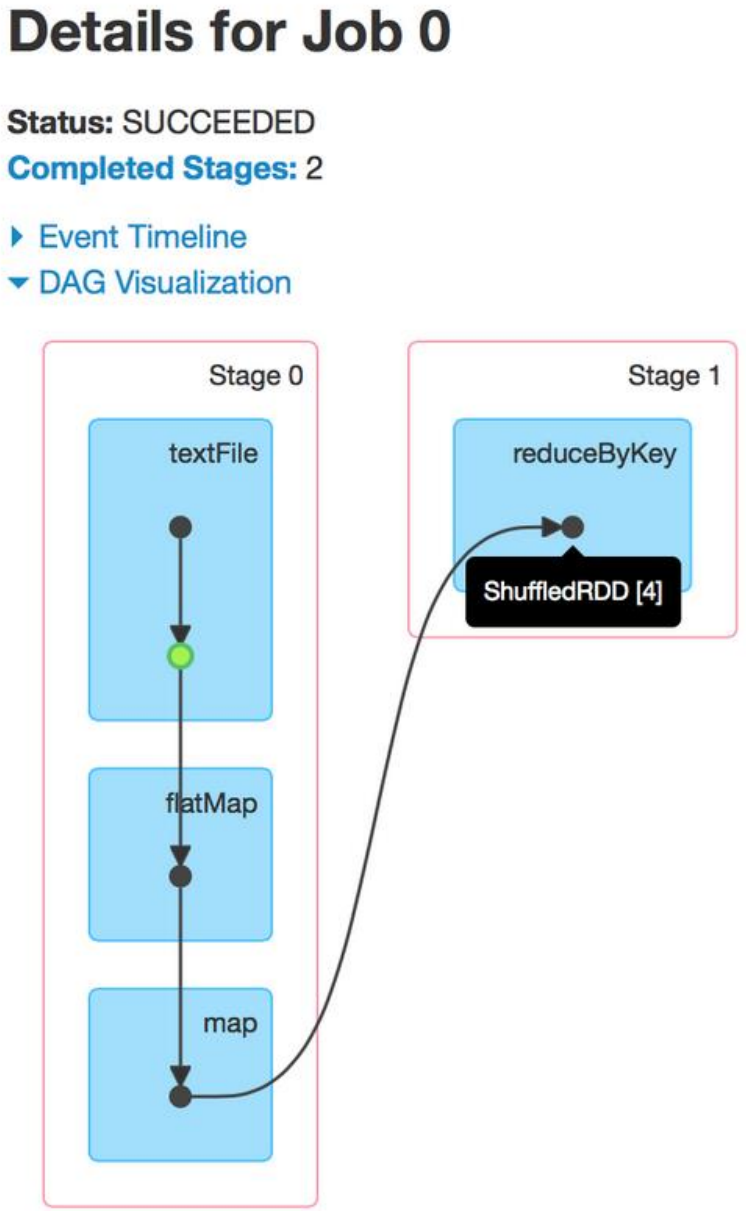
- **RDDs:**
  - *Resilient*: resilientes, isso é, caso um executor caia, o processo é reconstruído sem prejudicar o resultado;
  - *Distributed*: cada elemento da RDD pode ser processado em um nó diferente;
  - *Dataset*: conjunto de elementos tratados em conjunto, por exemplo, dados sobre clientes.

- **RDD Lineage:**
  - O DAG (*directed acyclic graph*) mapeia as transformações aplicadas às RDDs, mantendo a resiliência da execução.

Insper

Spark - DAG

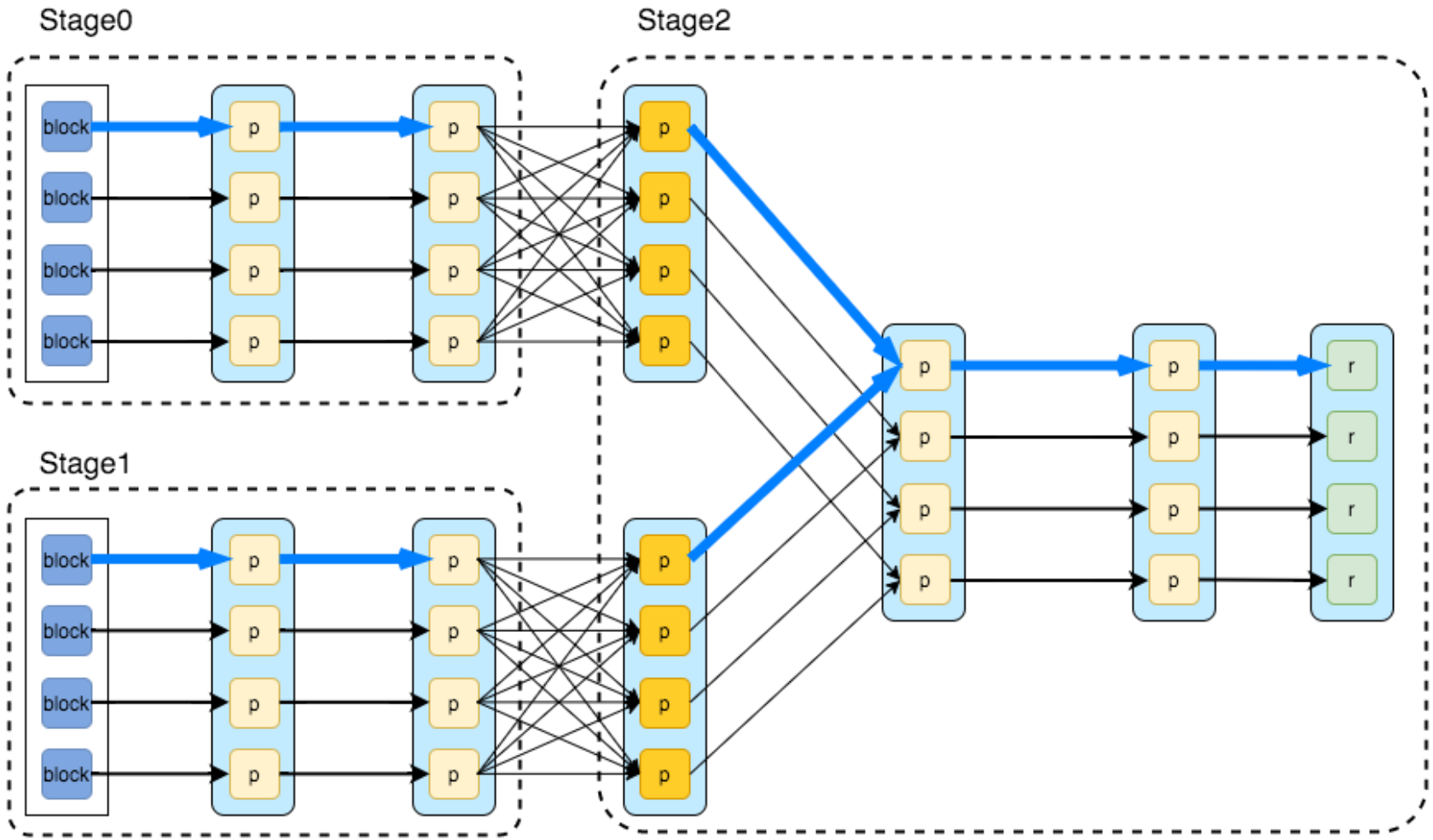
- Directed Acyclic Graph



Insper

Spark - DAG

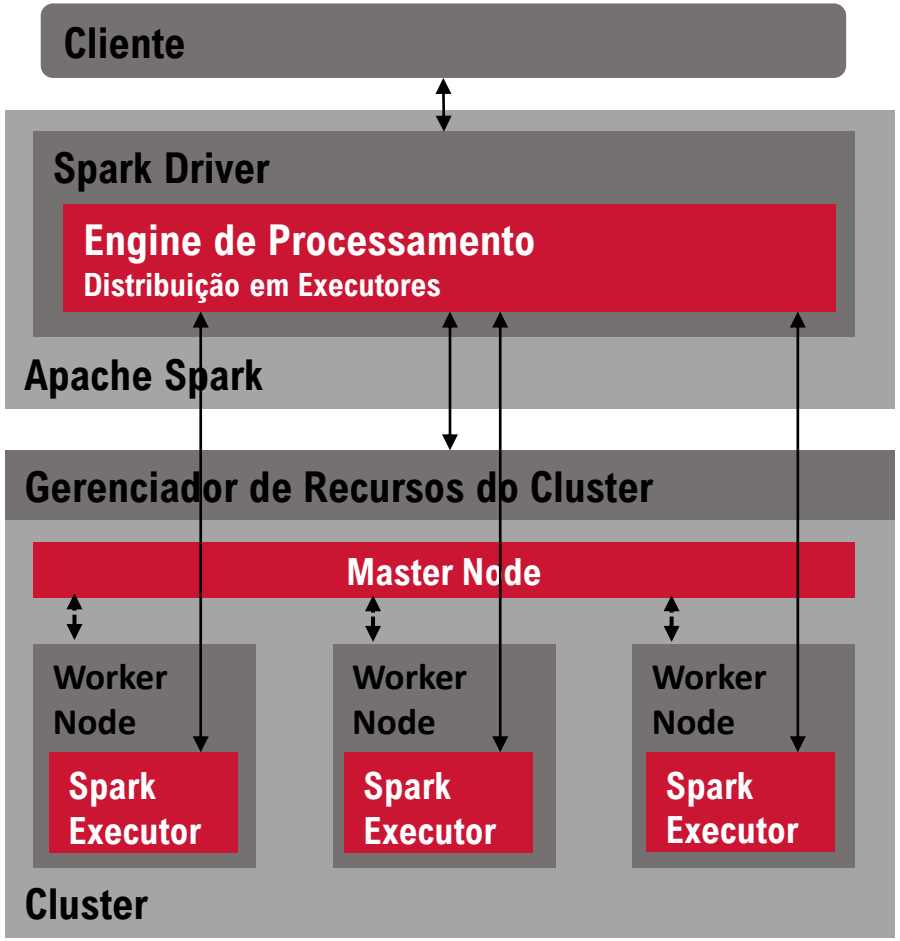
- **Stages** – shuffle boundaries



Insper

O Spark é composto por dois tipos de processos, o *Driver* e seus *Executors*, ambos criados dentro de *Java Virtual Machines*.

Anatomia do Spark

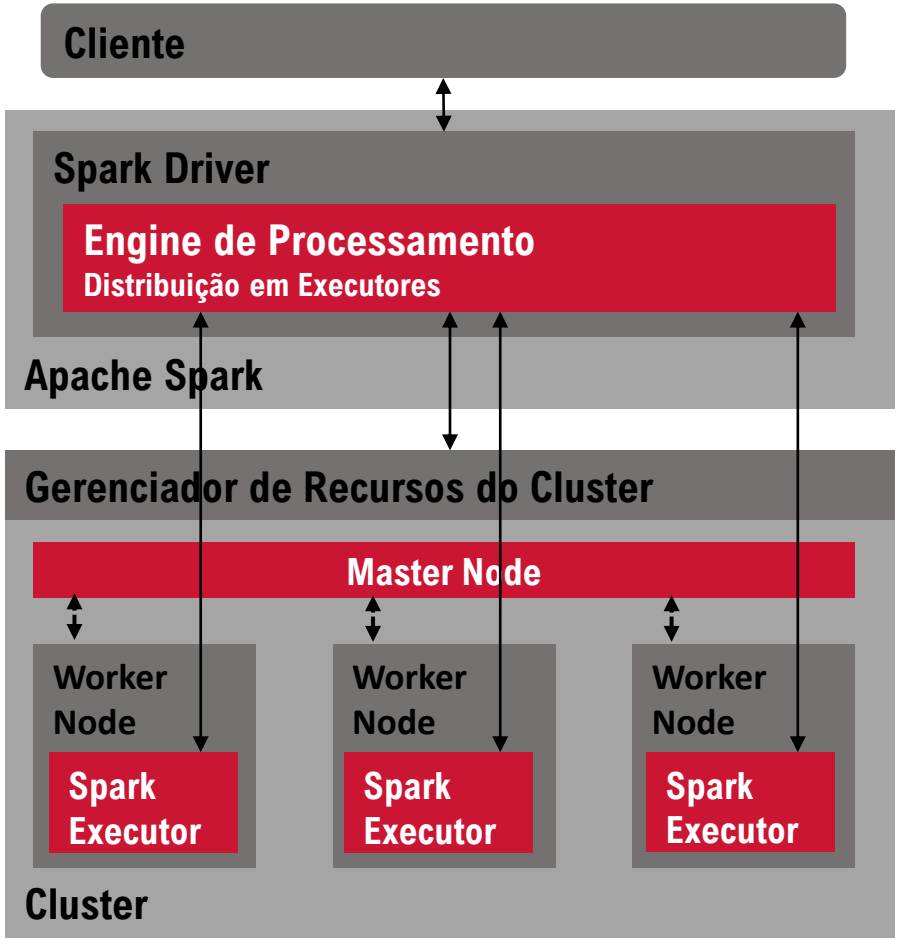


Fluxo de execução:

1. Cliente submete aplicação ao driver;
2. Driver planeja a execução:
   - Cria o DAG;
   - Requisita recursos para o cluster: *Executors* são disponibilizados;
   - Aloca tarefas aos *Executors*;
3. *Executors* executam suas tarefas;
4. Driver coordena execuções;
5. Driver retorna resultados para o cliente.

O Driver é a aplicação de interface com o cliente, responsável por planejar a execução distribuída.

Anatomia do Spark

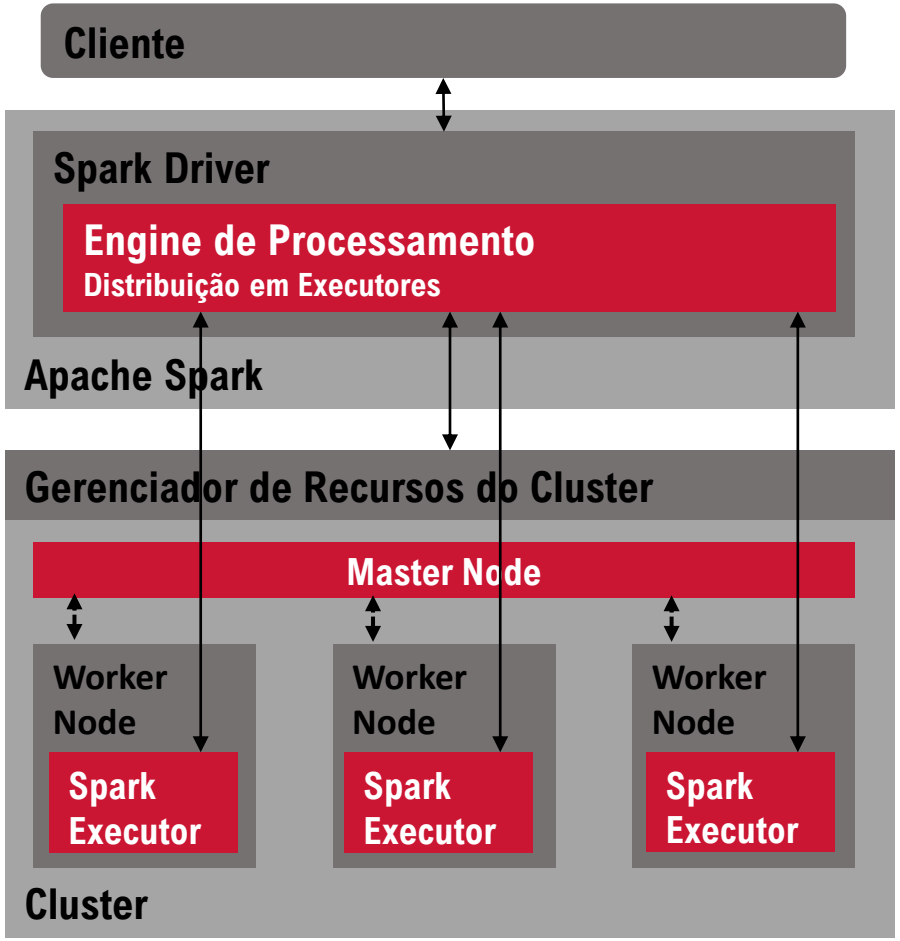


Driver:

- Planejamento da aplicação para execução:
  - Criação do DAG;
  - Quebra em *stages* e *tasks:*
    - *Tasks:* menor unidade de execução;
    - *Stage*: conjunto de *tasks* que deve ser executado como unidade;
  - Negociação de recursos com *cluster manager;*
- Orquestração das aplicações em execução:
  - Gestão dos recursos disponíveis;
  - Agendamento da execução de *tasks* próximas ao dado;

Os Executores são os processos de execução das tarefas. Estes processos são executados nos nós Worker do cluster.

Anatomia do Spark



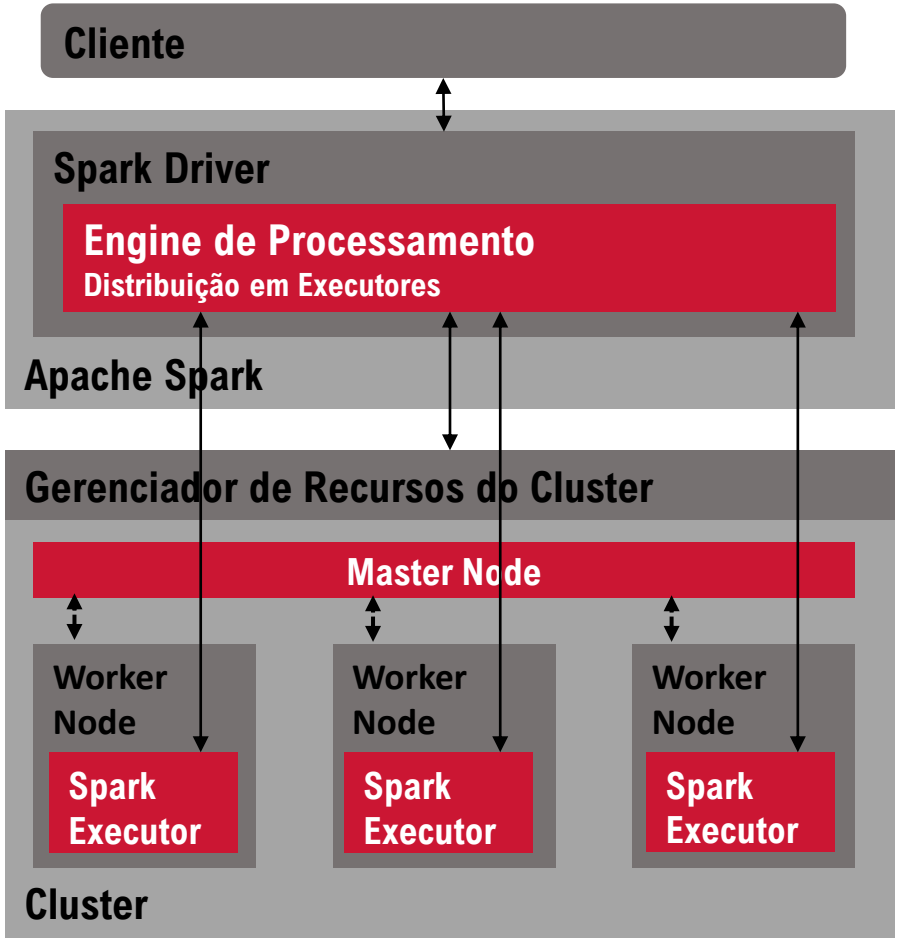
Executors:

- Aplicações JAVA com:
  - Memória e CPU reservados;
- Executam tarefas:
  - *Tasks*, partes de um DAG planejado pelo *Master;*

Insper

O Spark pode ser executado de diferentes modos. No modo local ele paraleliza a execução em processadores da máquina.

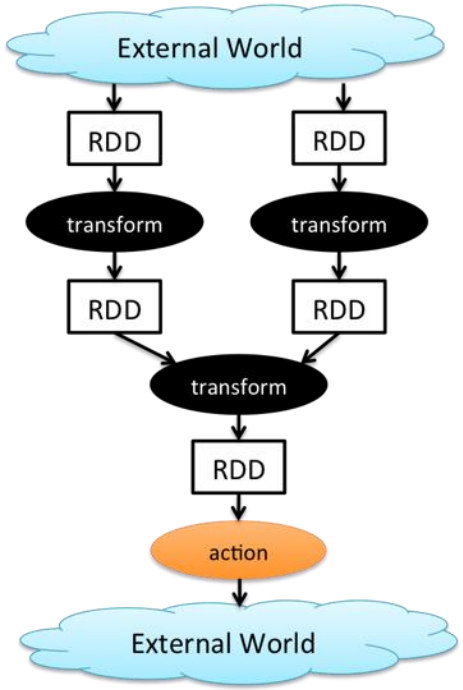
Modos de Execução do Spark

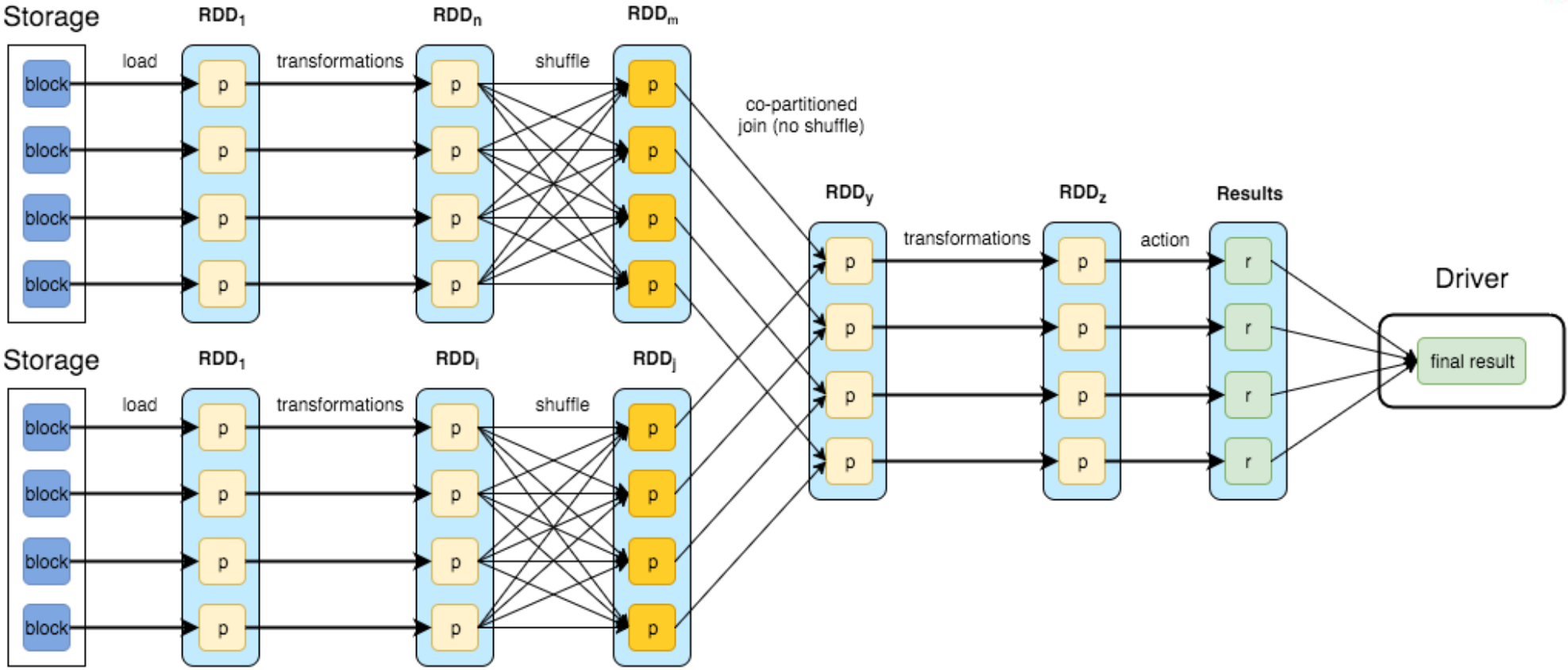


- **Local Mode:**
  - Spark opera localmente. Ideal para aprender e testar aplicações com poucos dados.
- **Standalone:**
  - Spark gerencia também os recursos do cluster, sem a necessidade de um gerenciador de recursos.
- **Cluster mode:** em conjunto com Hadoop
  - Utiliza o gerenciador de recursos do próprio cluster, podendo ele ser o YARN ou o MESOS;

Insper

Fluxo de trabalho Spark: Transformação e Ações

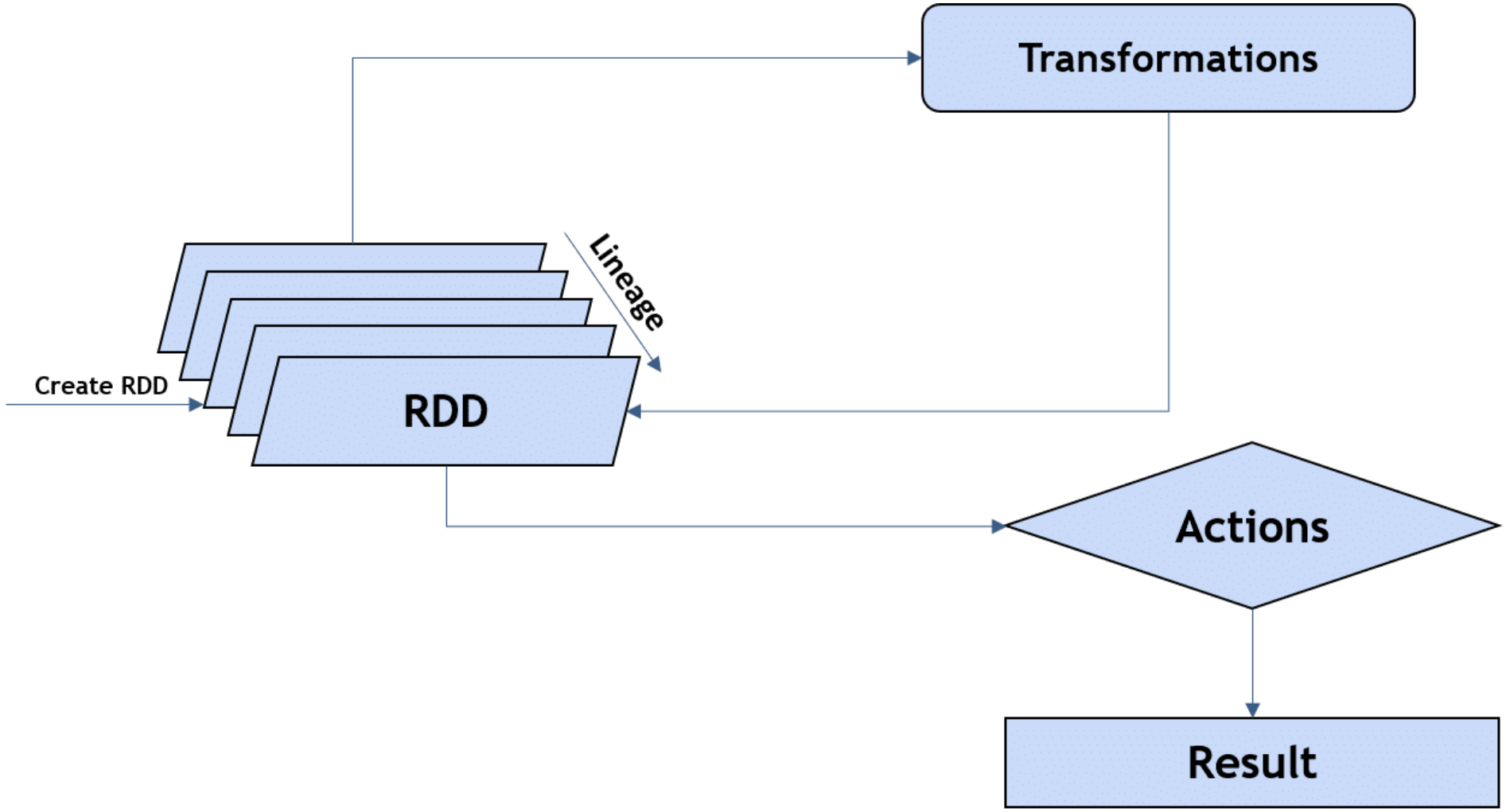




http://datastrophic.io/core-concepts-architecture-and-internals-of-apache-spark/

Insper

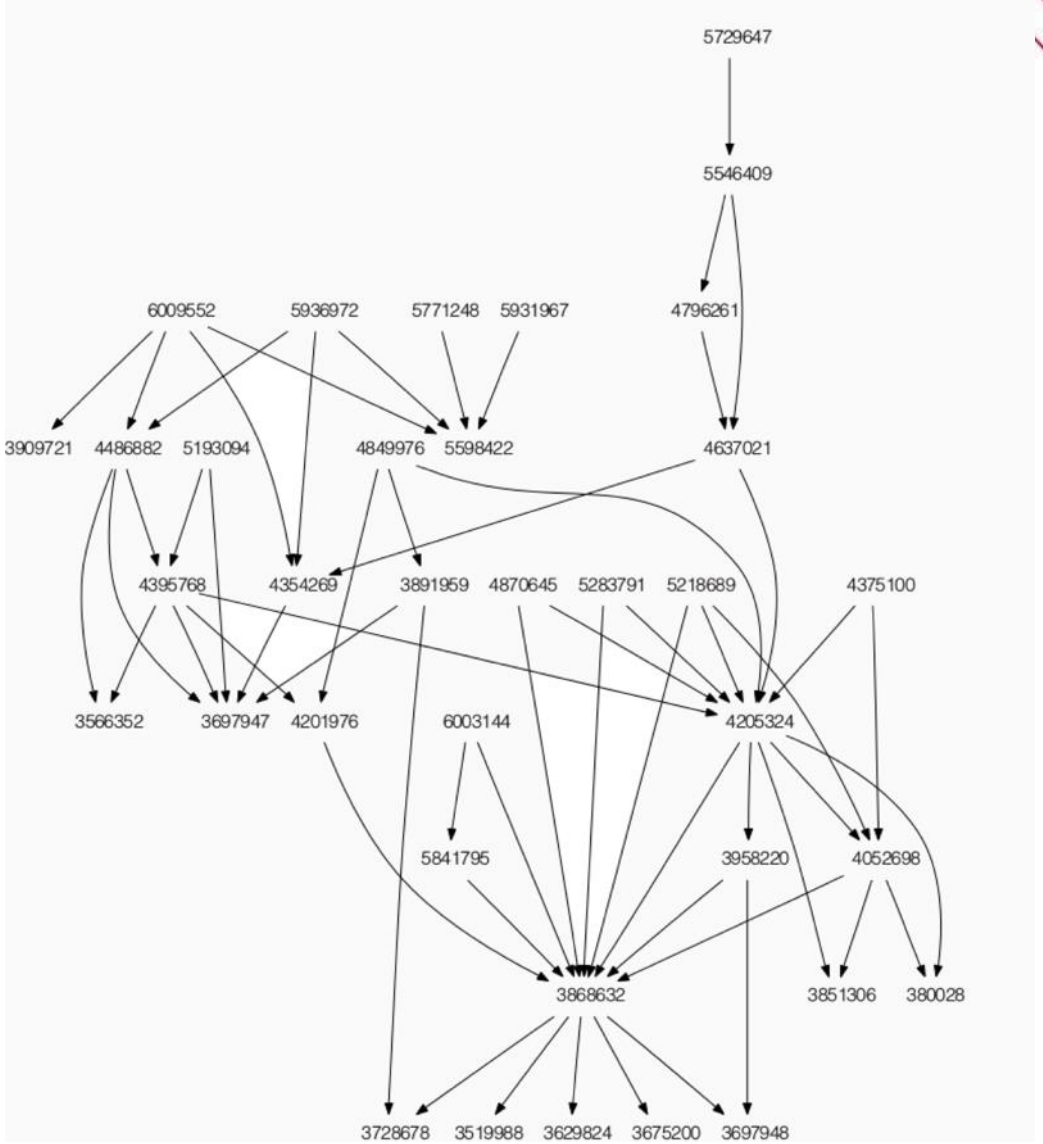
Fluxo de trabalho Spark: Transformação e Ações



Insper

Exemplo Prático

Contagem das Patentes com Spark RDD



Insper