Processamento de Dados em Larga Escala

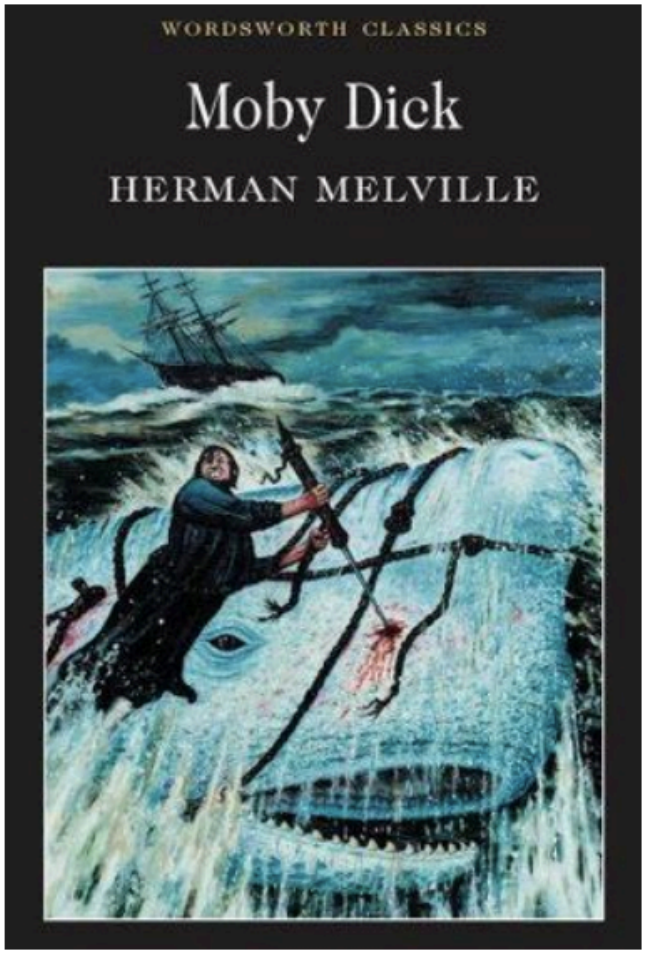
Luciano Barbosa

Princípio da Localidade

- Def: acesso ao mesmo conjunto da memória em um curto período de tempo
- Temporal
  - Acesso ao mesmo conjunto de dados num curto período
- Espacial
  - Acesso a dados presentes em locais de armazenamento relativamente próximos

Exemplo: Contar Palavras em um Livro

- 218 mil palavras
- 17 mil palavras distintas
- Qual a frequência de cada palavra



Exemplo: Contar Palavras em um Livro

- **Entrada**: Call me Ishmael. Some years ago – never mind how long precisely — having little or no money in my purse, and nothing particular to interest me onshore, I thought I would sail …
- Arquivo de 1.3 MB
- **Saída**:
  - Call: 354
  - Me: 53423
  - Ismael: 1322

Solução Simples

- Iterar sobre as palavras e contar a frequência

In [5]:

```
%%time
def simple_count(list):
    D={}
    for w in list:
        if w in D:
            D[w]+=1
        else:
            D[w]=1
    return D
D=simple_count(all)
```

```
CPU times: user 49.5 ms, sys: 5.95 ms, total: 55.5 ms
Wall time: 53.1 ms
```

Lista Ordenada

```
=== unsorted list:
the,vernacular,but,as,for,you,ye,carrion,rogues,turning,to,
the,three,men,in,the,rigging,for,you,i,mean,to,mince,ye,up,
for

=== sorted list:
lines,lingered,lingered,lingered,lingered,lingered,lingerin
g,lingering,lingering,lingering,lingering,lingering,lingeri
ng,lingering,lingers,lingo,lingo,lining,link,link,linked,li
nked,linked,linked,links,links
```

5

8

Solução Usando Lista Ordenada

- Ordenação melhora localidade que reduz tempo para a contagem da frequência

```
def sort_count(list):
    t0=time()
    S=sorted(list)
    t1=time()
    D={}
    current=''
    count=0
    for w in S:
        if current==w:
            count+=1
        else:
            if current!='':
                D[current]=count
            count=1
            current=w
    t2=time()
    return D,t1-t0,t2-t1
D,sort_time,count_time=sort_count(all)
print 'sort time= %5.1f ms, count time=%5.1f ms'%(1000*
```

Sort words

Iterate over sorted list

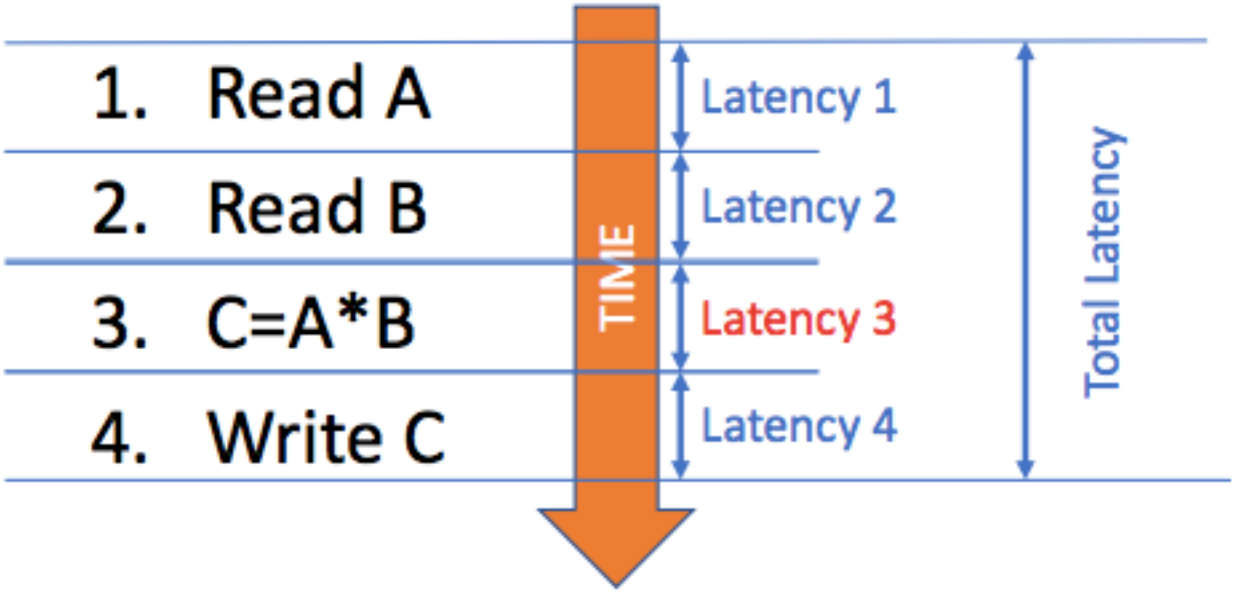
Count occurrences of same word

Switch on word boundry

```
sort time= 103.0 ms, count time= 37.6 ms
CPU times: user 138 ms, sys: 5.33 ms, total: 143 ms
Wall time: 143 ms
```

Latência de Armazenamento

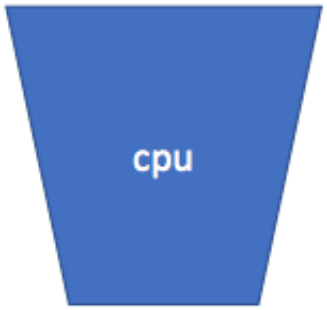
- Boa parte do tempo é gasto nos passos 1,2,4 (leitura e escrita) e não na computação do passo 3



Latência

- Maior fonte de latência em análise de dados: leitura e escrita dos dados
- Diferentes tipos de armazenamento oferece diferente latência, capacidade e preço
- Big data analytics: organizar armazenamento e computação para maximizar velocidade e minimizar custo

Localidade Temporal



Fast & Small

Cache

12	67
50	51
52	53
32	33

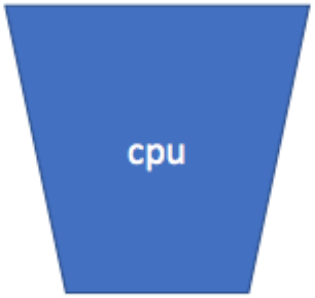
Slow & Large

Memory

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79

Localidade Espacial

- Vários acessos a endereços de memória próximos em um curto período de tempo
- Memória particionada em blocos



Cache

50	51
52	53
32	33
34	35

Memory

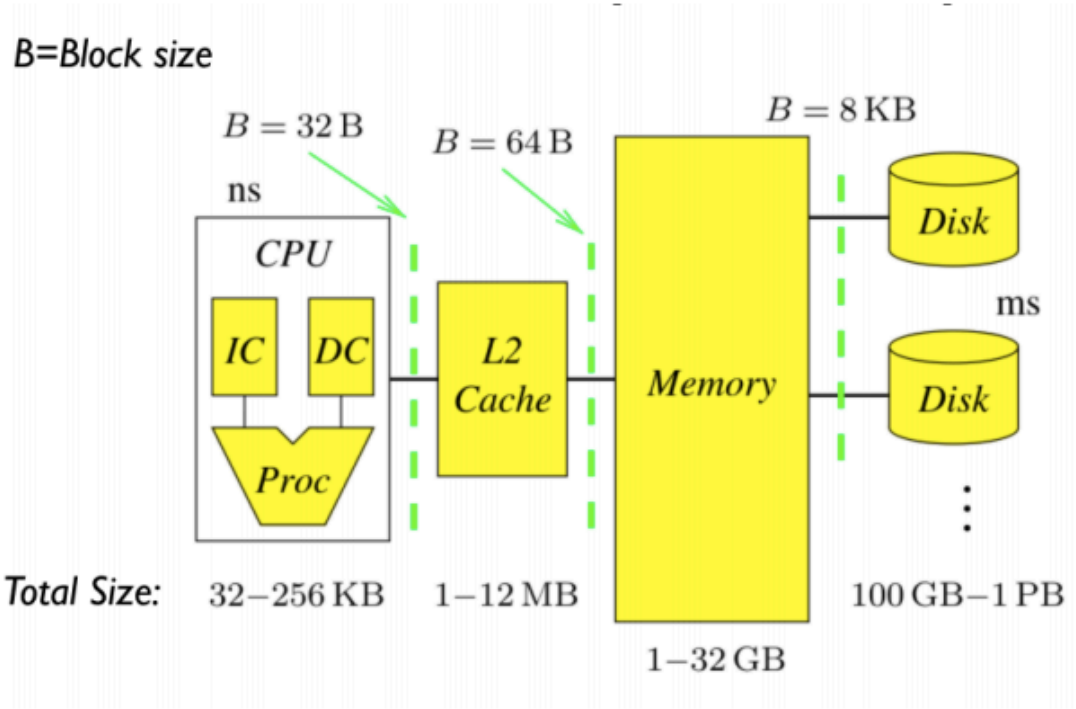
0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79

Em Resumo

- Cache reduz latência por trazer dados relevantes ao processamento próximos à CPU
- Para isso:
  - Localidade temporal: acesso ao mesmo dado várias vezes
  - Localidade espacial: acesso a dados próximos

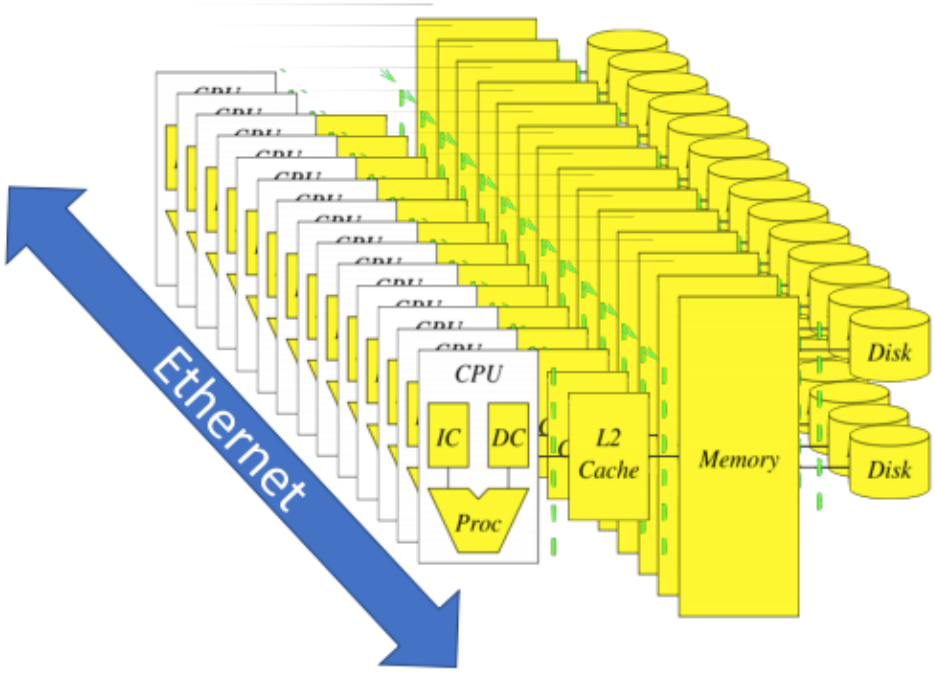
Hierarquia da Memória

- Topo: pequena, rápida e perto da CPU
- Cache usada para transferir dados entre diferentes níveis



Clusters de Computadores

- Extende a hierarquia de memória
- Ligados numa ethernet
- Armazenamento compartilhado



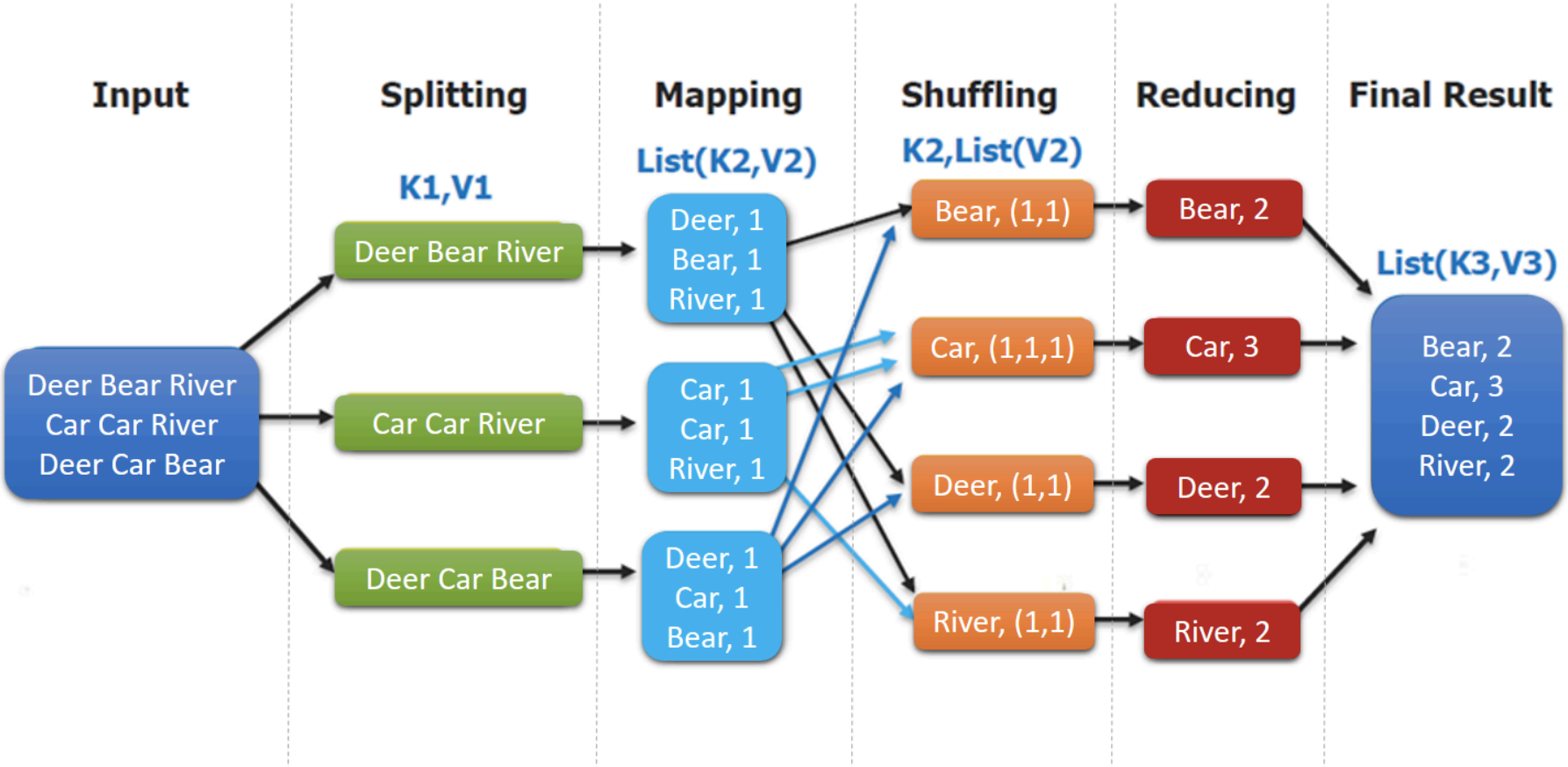
Exemplo de Tamanho e Latência em uma Hierarquia de Memória

	CPU (Registers)	L1 Cache	L2 Cache	L3 Cache	Main Memory	Disk Storage	Local Area Network
Size (bytes)	1KB	64KB	256KB	4MB	4-16GB	4-16TB	16TB - 10PB
Latency	300ps	1ns	5ns	20ns	100ns	2-10ms	2-10m
Block size	64B	64B	64B	64B	32KB	64KB	1.5-64KB

12 orders of magnitude

6 orders of magnitude

Map Reduce



MapReduce Word Count Process

Map

- Aplica uma função a todos elementos de uma lista
- Ex: Computar a soma dos quadrados de uma lista
  - Lista L = [0,1,2,3]
  - Computar o quadrado de cada item
  - Saída: L = [0,1,4,9]

```
## For Loop
O=[]
for i in L:
    O.append(i*i)

## List Comprehension
[i*i for i in L]
```

Tradicional

```
map(lambda x:x*x, L)
```

Map-Reduce

Reduce

- Realiza a computação em uma lista e retorna um resultado
- Ex: Computar a soma dos quadrados de uma lista
  - Lista L = [0,1,2,3]
  - Calcular a soma
  - Saída: 16

```
## Use Builtin
sum(L)

## for loop
s=0
for i in L:
    s+=i
```

Tradicional

```
reduce(lambda (x,y): x+y, L)
```

Map-Reduce

Exemplo em Python

```
## For Loop
s=0
for i in L:
    s+= i*i
## List comprehension
sum([i*i for i in L])
```

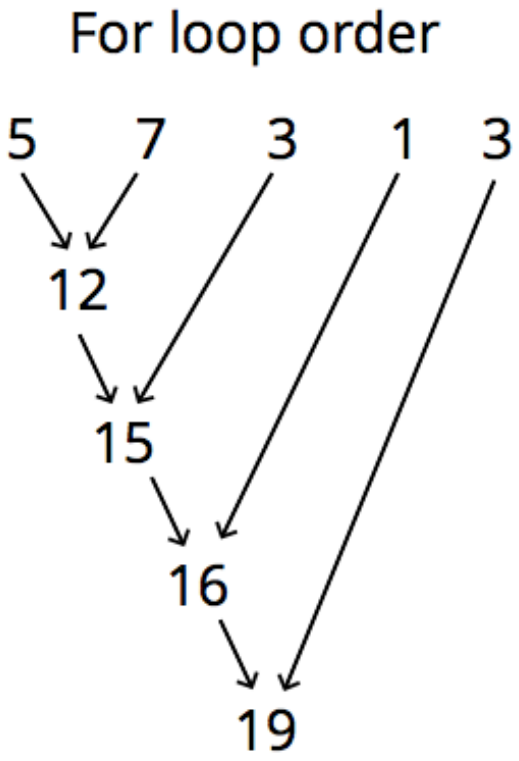
Tradicional

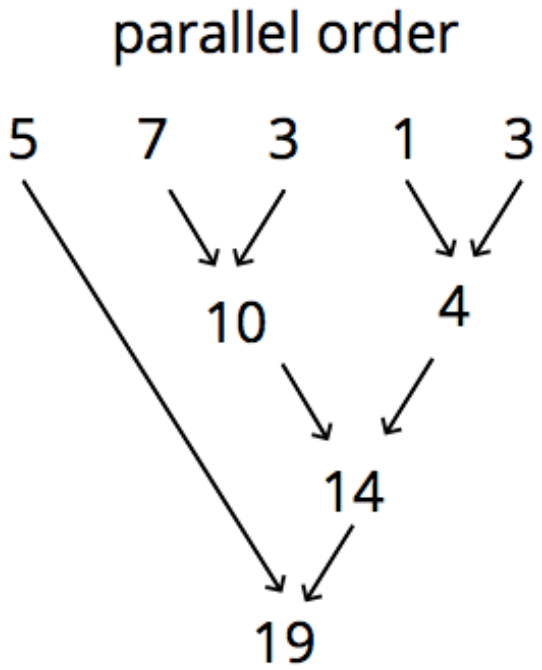
```
reduce(lambda x,y:x+y, \\
        map(lambda i:i*i,L))
```

Map-Reduce

Independência de Ordem

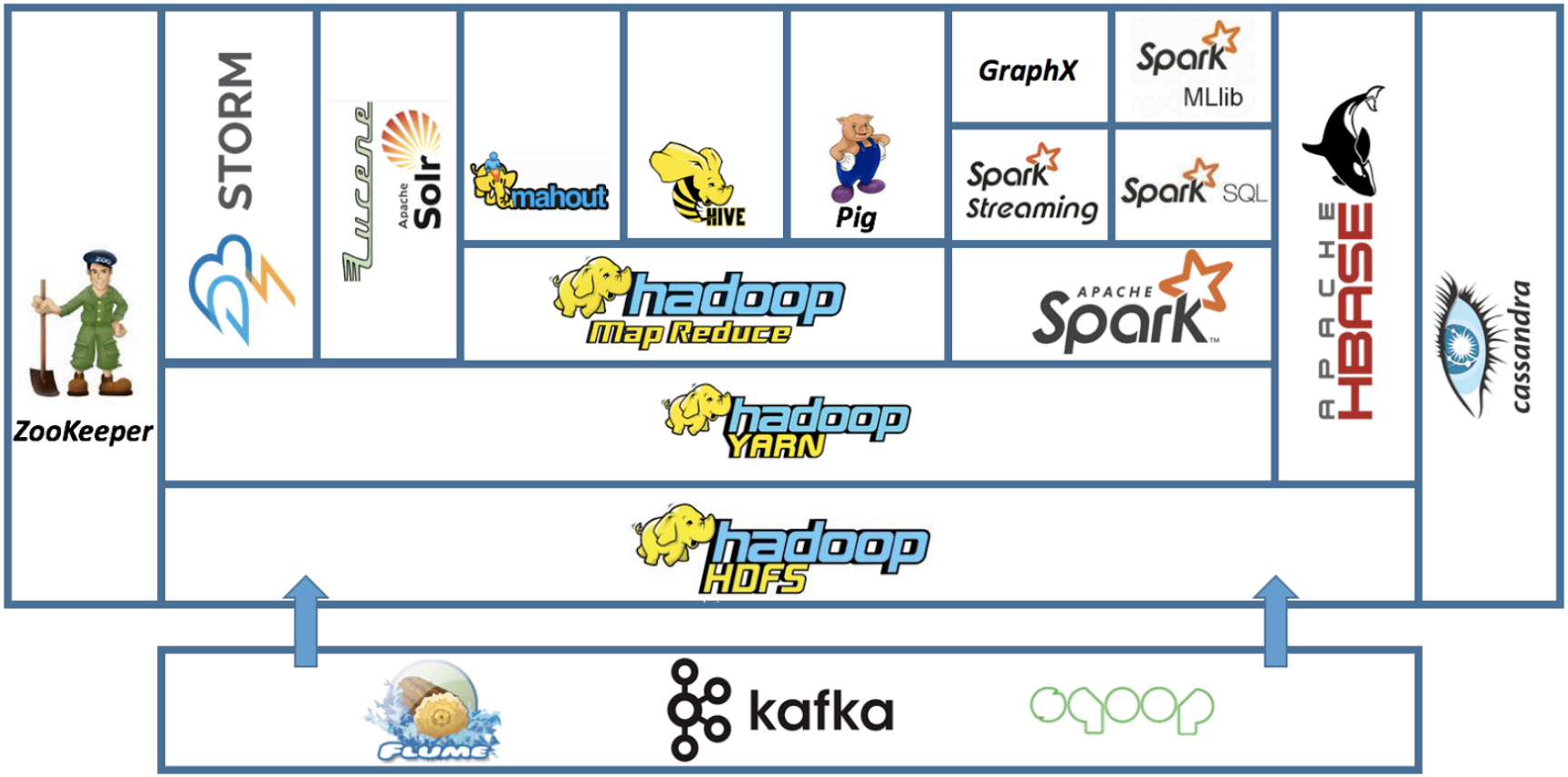
- O resultado não pode depender da ordem





Apache Hadoop

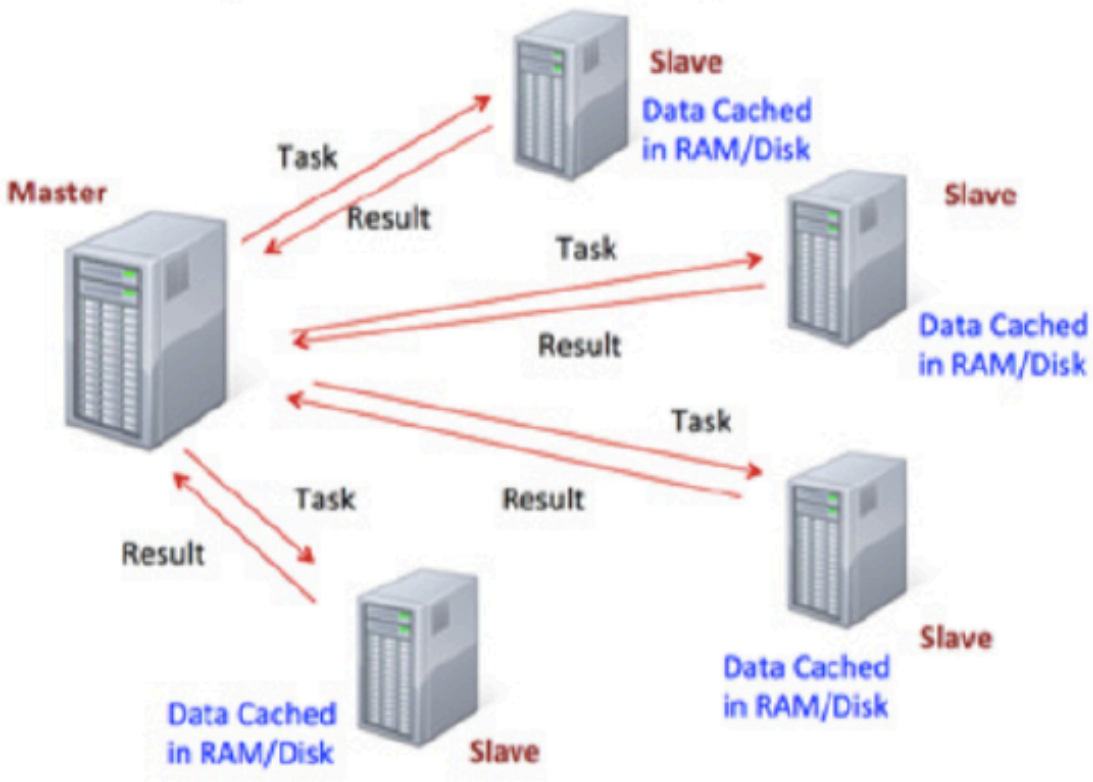
- Implementação open-source do map-reduce



Apache Spark

- Diferença do Hadoop: memória distribuída ao invés de arquivos
- Java é a linguagem nativa do Hadoop
- Scala: linguagem principal para Spark
  - Problema: poucas pessoas usam
- PySpark
  - Extensão de Python
  - Nem sempre possui a mesma eficiência de scala
  - Mais fácil de aprender

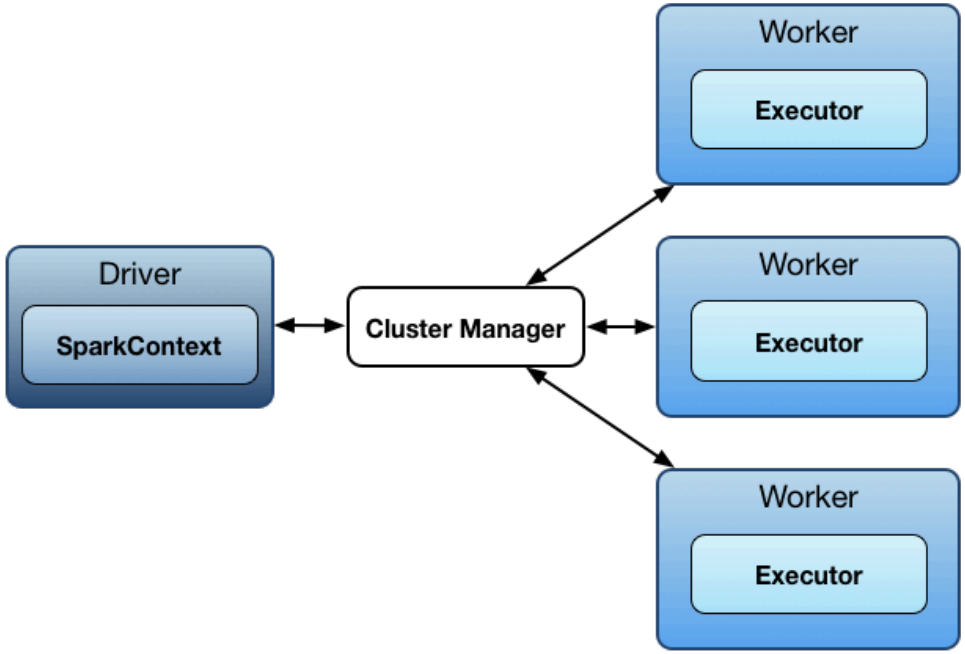
Ambiente de Execução



- Em máquina única: cores da cpu servem como master e slaves

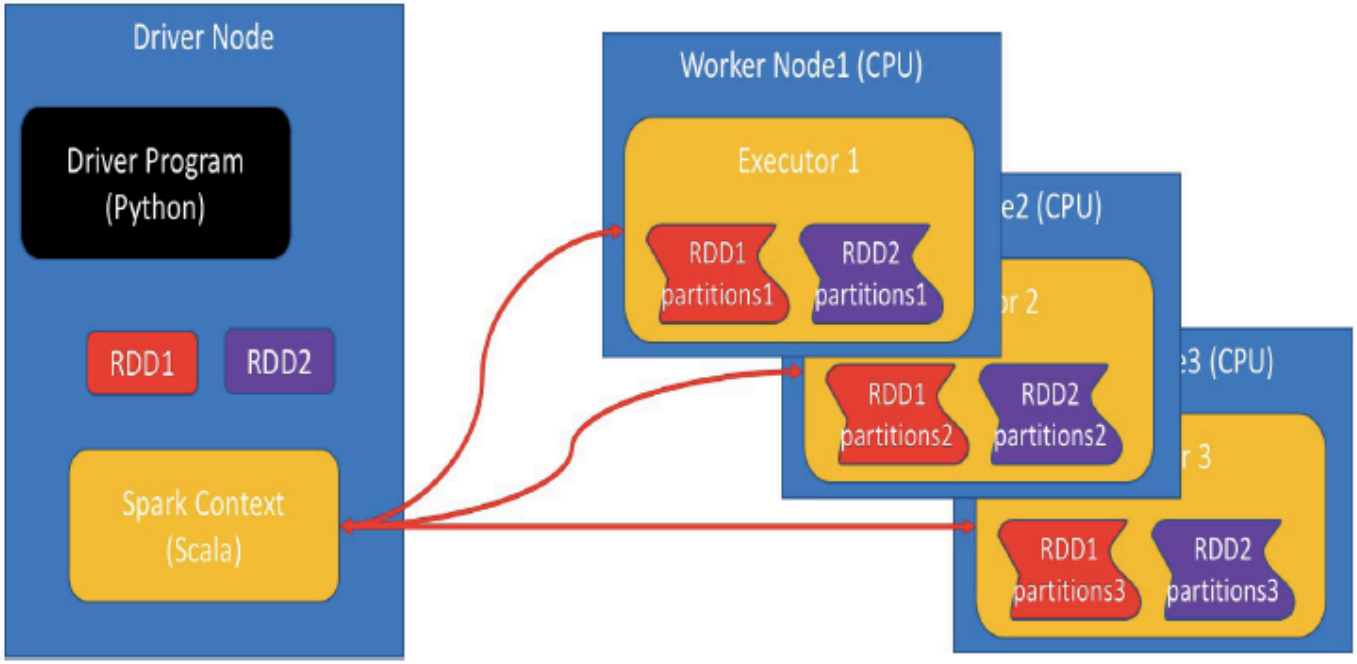
PySpark: Spark Context

- Executado no nó central
- Controla os outros nós
- Necessário somente 1



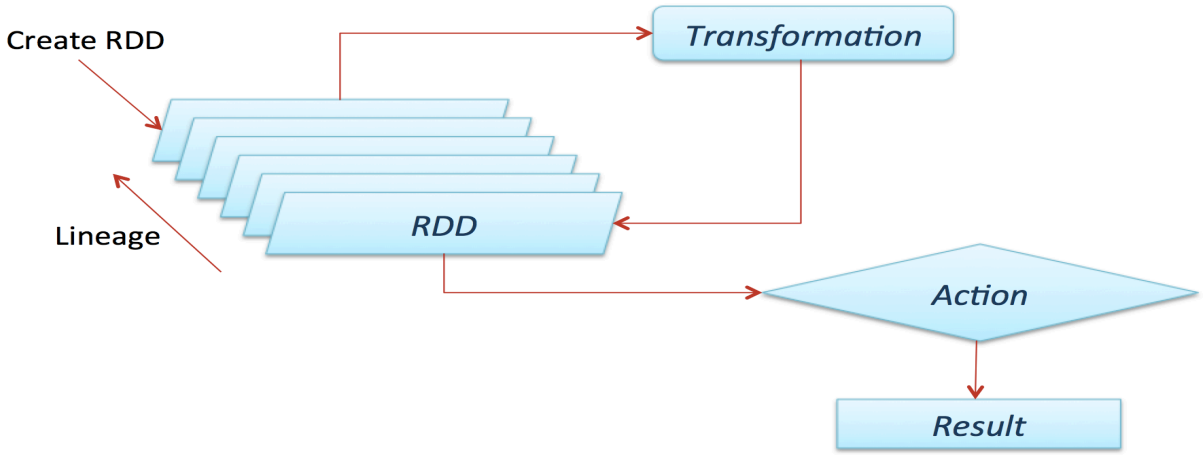
PySpark: RDD

- RDD: Resilient Distributed Dataset
- Paraleliza o processamento: clusters/cores
- Tolerante à falha



PySpark: RDD

- Permite três tipos de operações
  - Creation: lê dados em arquivos ou banco de dados
  - Transformation: aplicadas a um RDD para gerar outro (filtro, groupby etc)
  - Action: escreve dados para arquivos ou banco de dados



Comandos Básicos

- Parallelize
  - Forma mais simples de criar um RDD
  - Distribue o RDD entre executores
  - Ex: A=sc.parallelize(L)
- Collect
  - Coleta dados distribuídos
  - Retorna lista
  - Ex: A.collect()

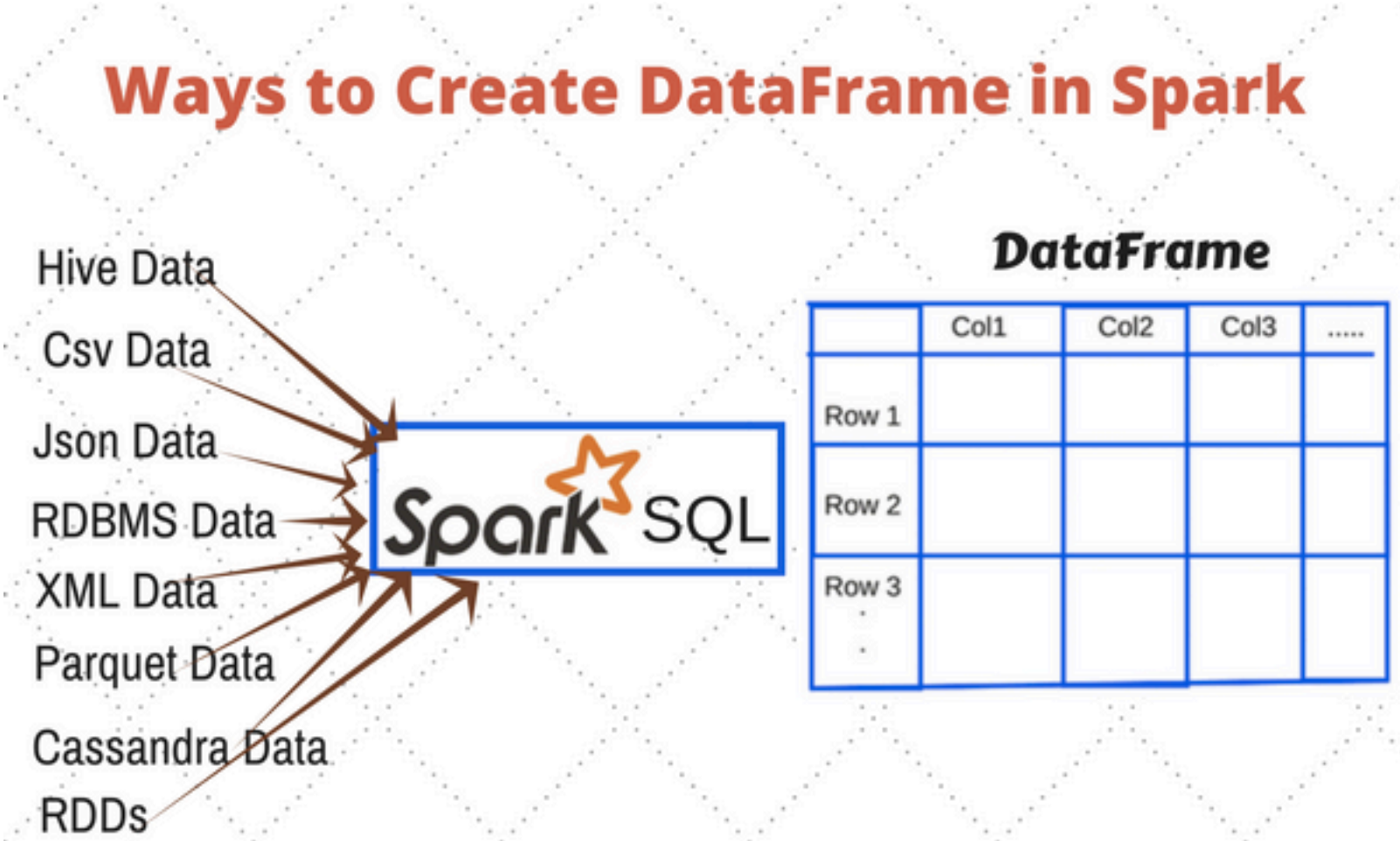
Comandos Básicos: Map

- Aplica uma operação a cada elemento do RDD
- Operações executadas em paralelo em todos executores
- Retorna um RDD
- Cada executor opera em dados locais dele
- Ex: B = A.map(lambda x: x-2)

Comandos Básicos: Reduce

- Entrada: RDD
- Saída: único valor
- Os resultados de todos executores são combinados
- Ex: A.reduce(lambda x,y: x+y)

DataFrame


Ways to Create DataFrame in Spark
Hive Data
Csv Data
Json Data
RDBMS Data
XML Data
Parquet Data
Cassandra Data
RDDs
Spark SQL
DataFrame
Col1
Col2
Col3
.....
Row 1
Row 2
Row 3

Diferenças entre DataFrame em Pandas e PySpark

- Pandas
  - Não roda em paralelo
  - Resultado das operações disponíveis qdo ela termina
  - Suporta mais operações
  - Operações complexas mais fáceis
- PySpark
  - Roda em paralelo em nós do cluster
  - Operações are lazy