Hiding within those mounds of data is knowledge that could change the life of a patient, or change the world

—Atul Buttle, Buttle Labs – Solving Problems in Genomic Medicine

Computação em Larga Escala

Aula 04

Spark RDD (Transformações e Ações)

Partições, Otimizações

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

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

Insper

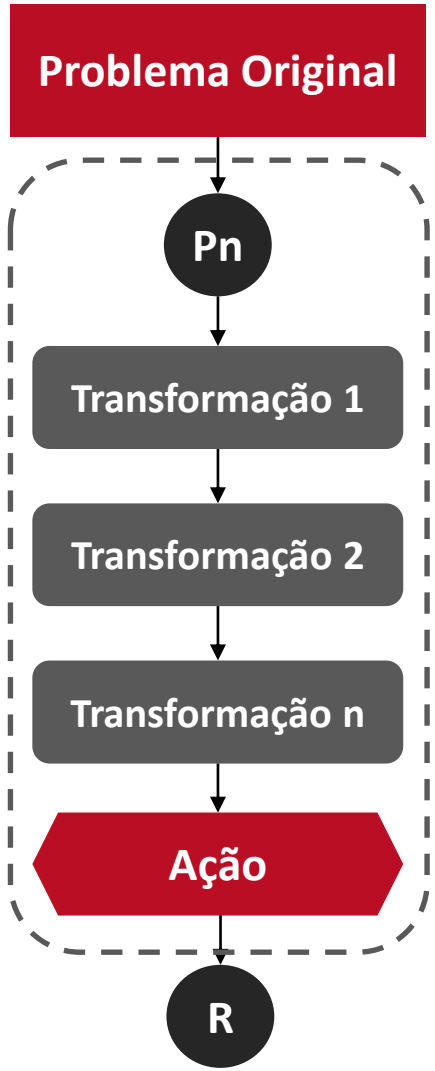
Spark RDD

Resilient Distributed Datasets

RDD:
Conjunto de objetos

Transformações:
Aplicadas a cada elemento da RDD

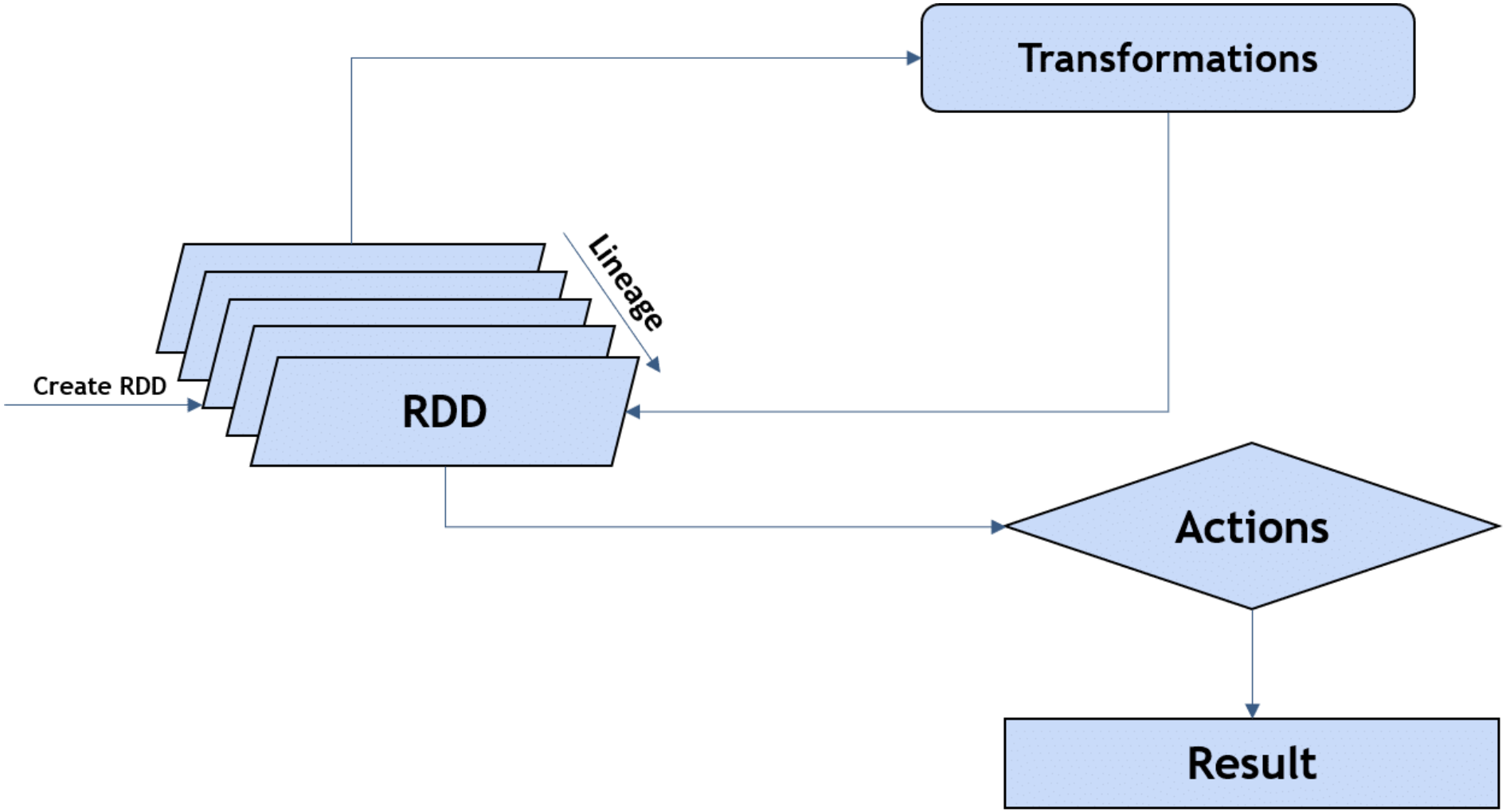
Ações:
Aplicadas a todos os elementos das RDDs.



- **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.

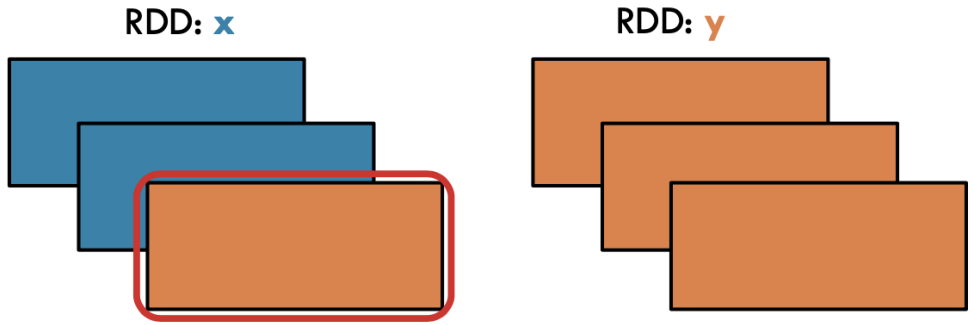
Fluxo de trabalho Spark: Transformação e Ações



Insper

Transformação: Map

MAP



`map(f, preservesPartitioning=False)`

Return a new RDD by applying a function to each element of this RDD

```python
x = sc.parallelize(["b", "a", "c"])
y = x.map(lambda z: (z, 1))
print(x.collect())
print(y.collect())
```

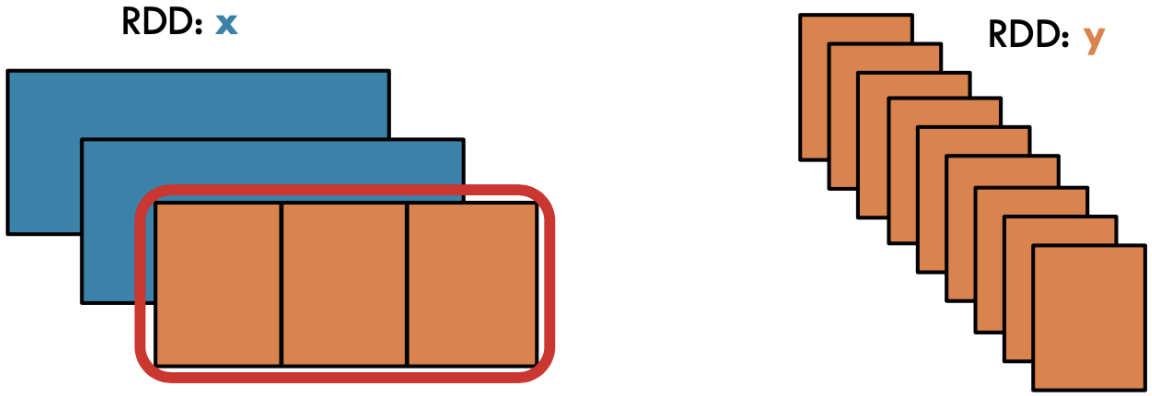
```scala
val x = sc.parallelize(Array("b", "a", "c"))
val y = x.map(z => (z,1))
println(x.collect().mkString(", "))
println(y.collect().mkString(", "))
```

```
x: ['b', 'a', 'c']

y: [('b', 1), ('a', 1), ('c', 1)]
```

Insper

Transformação: FlatMap

FLATMAP



`flatMap(f, preservesPartitioning=False)`

Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results

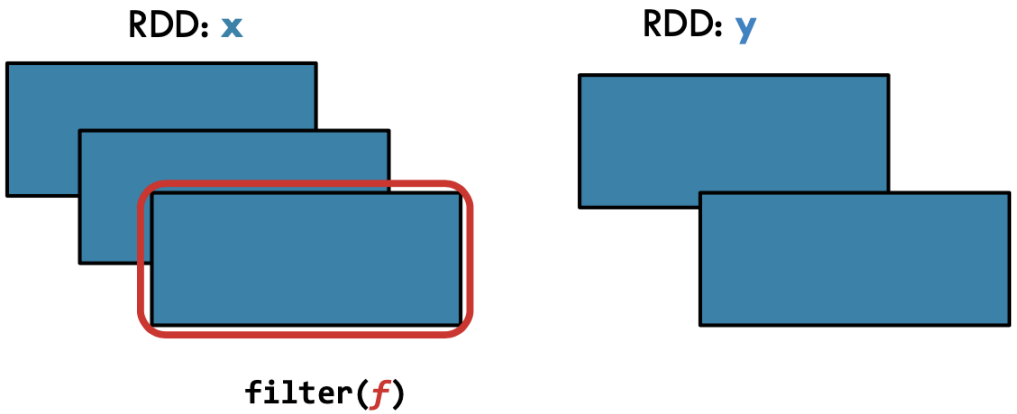
```python
x = sc.parallelize([1,2,3])
y = x.flatMap(lambda x: (x, x*100, 42))
print(x.collect())
print(y.collect())
```

```scala
val x = sc.parallelize(Array(1,2,3))
val y = x.flatMap(n => Array(n, n*100, 42))
println(x.collect().mkString(", "))
println(y.collect().mkString(", "))
```

```
x: [1, 2, 3]

y: [1, 100, 42, 2, 200, 42, 3, 300, 42]
```

Transformação: Filter

FILTER



Return a new RDD containing only the elements that satisfy a predicate

```python
x = sc.parallelize([1,2,3])
y = x.filter(lambda x: x%2 == 1) #keep odd values
print(x.collect())
print(y.collect())
```

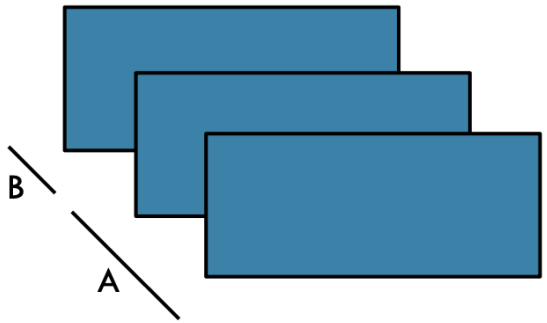
```
x: [1, 2, 3]

y: [1, 3]
```

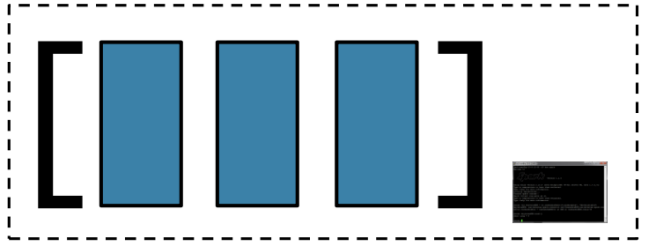
```scala
val x = sc.parallelize(Array(1,2,3))
val y = x.filter(n => n%2 == 1)
println(x.collect().mkString(", "))
println(y.collect().mkString(", "))
```

Insper

Ação: Collect

COLLECT





collect()

Return all items in the RDD to the driver in a single list

```python
x = sc.parallelize([1,2,3], 2)
y = x.collect()

print(x.glom().collect())
print(y)
```

```scala
val x = sc.parallelize(Array(1,2,3), 2)
val y = x.collect()

val xOut = x.glom().collect()
println(y)
```

```
x: [[1], [2, 3]]

y: [1, 2, 3]
```


Insper

Transformações e Ações

= easy = medium

Essential Core & Intermediate Spark Operations

General

- map
- filter
- flatMap
- mapPartitions
- mapPartitionsWithIndex
- groupBy
- sortBy

Math / Statistical

- sample
- randomSplit

Set Theory / Relational

- union
- intersection
- subtract
- distinct
- cartesian
- zip

Data Structure / I/O

- keyBy
- zipWithIndex
- zipWithUniqueID
- zipPartitions
- coalesce
- repartition
- repartitionAndSortWithinPartitions
- pipe

ACTIONS

- reduce
- collect
- aggregate
- fold
- first
- take
- forEach
- top
- treeAggregate
- treeReduce
- forEachPartition
- collectAsMap

- count
- takeSample
- max
- min
- sum
- histogram
- mean
- variance
- stdev
- sampleVariance
- countApprox
- countApproxDistinct

- takeOrdered

- saveAsTextFile
- saveAsSequenceFile
- saveAsObjectFile
- saveAsHadoopDataset
- saveAsHadoopFile
- saveAsNewAPIHadoopDataset
- saveAsNewAPIHadoopFile

Insper

Transformações e Ações

 = easy  = medium

Essential Core & Intermediate PairRDD Operations

General

- flatMapValues
- groupByKey
- reduceByKey
- reduceByKeyLocally
- foldByKey
- aggregateByKey
- sortByKey
- combineByKey

Math / Statistical

- sampleByKey

Set Theory / Relational

- cogroup (=groupWith)
- join
- subtractByKey
- fullOuterJoin
- leftOuterJoin
- rightOuterJoin

Data Structure

- partitionBy

- keys
- values

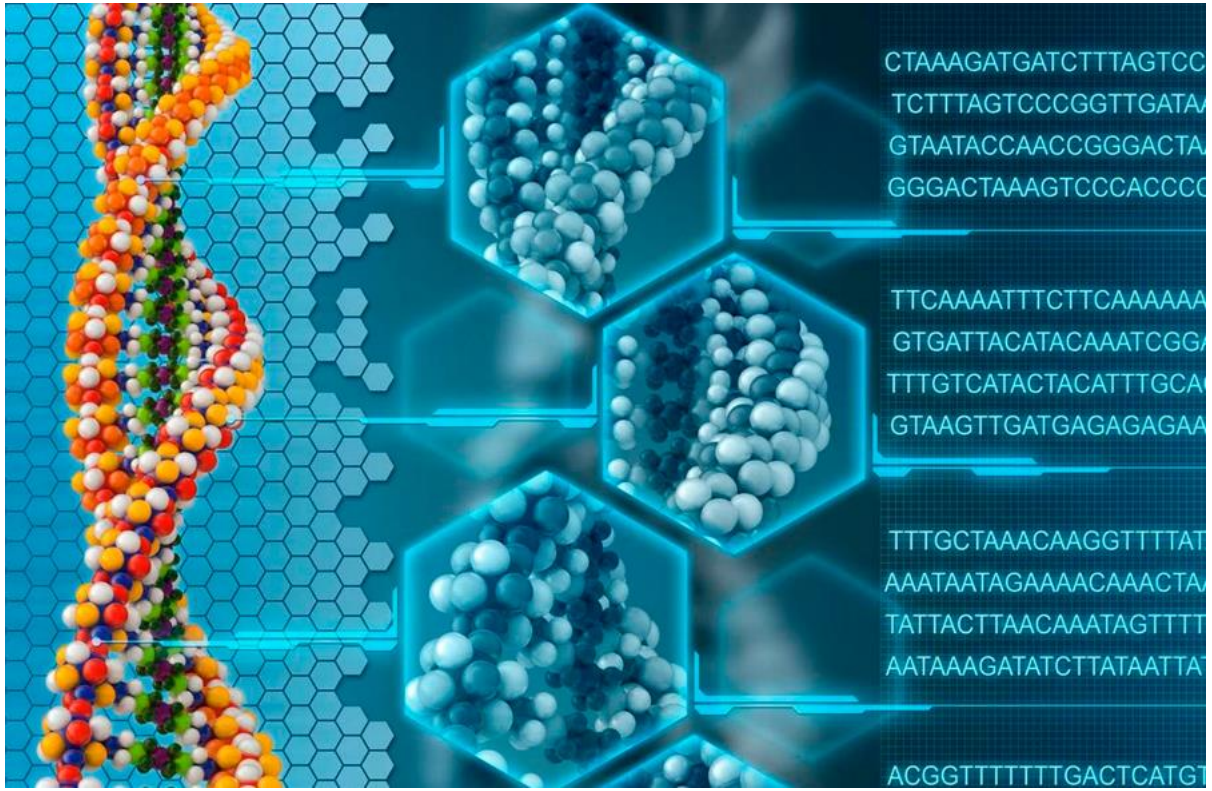
- countByKey
- countByValue
- countByValueApprox
- countApproxDistinctByKey
- countApproxDistinctByKey
- countByKeyApprox
- sampleByKeyExact

Insper

Prática

Map/Reduce | Otimização

- Processamento de Sequências de DNA
  - Uma sequência de DNA é uma série de letras representando a estrutura primária de uma molécula de DNA com a capacidade de carregar informações
  - As possíveis letras são A, C, G e T, representando os quatro nucleotídeos de uma cadeia de DNA



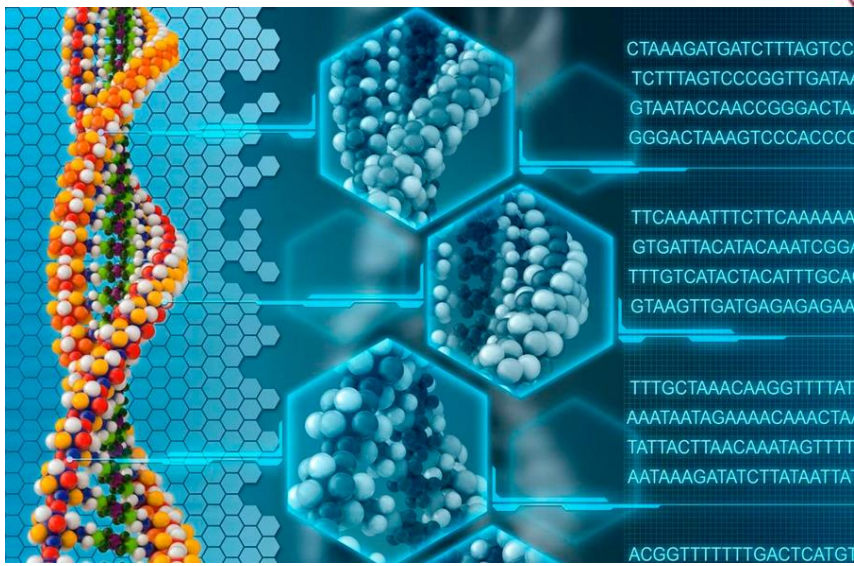
Insper

Prática

Map/Reduce | Otimização

- Processamento de Sequências de DNA

- Em alguns casos especiais, outras letras além de A, T, C e G estão presentes numa sequência
- International Union of Pure and Applied Chemistry (IUPAC):
  - A = adenina
  - C = citosina
  - G = guanina
  - T = timina
  - R = G A (purina)
  - Y = T C (pirimidina)
  - K = G T (ceto)
  - M = A C (amino)
  - S = G C (ligações de hidrogênio fortes)
  - W = A T (ligações de hidrogênio fracas)
  - B = G T C (todos, menos A)
  - D = G A T (todos, menos C)
  - H = A C T (todos, menos G)
  - V = G C A (todos, menos T)
  - N = A G C T (qualquer nucleotídeo)

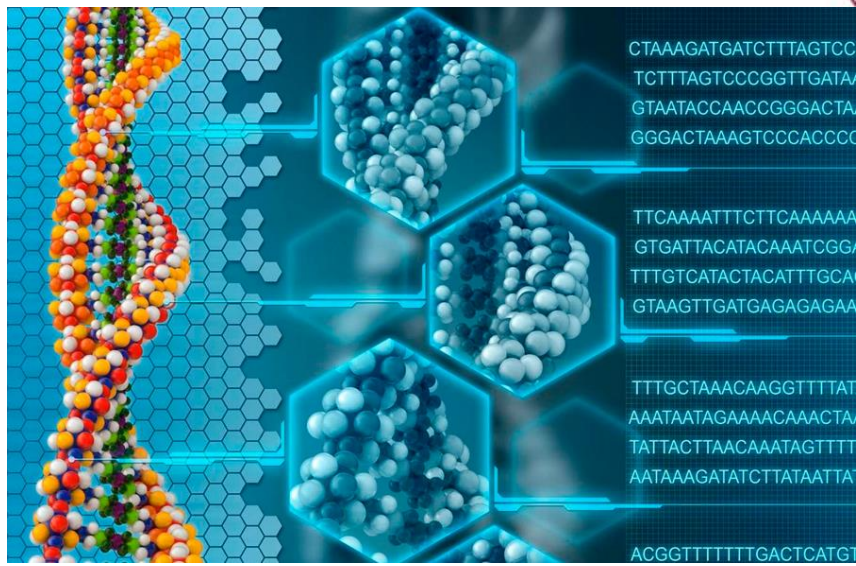


Insper

Prática

Map/Reduce | Otimização

- Formato FASTA
  - Formato baseado em texto para representar uma sequência de DNA



Header
>VIT_201s0011g03530.1
Sequence
AATTAAGCATAAATACTCACTCTTACCCCCTTATTTTCTTATCTCTCATCACTTTTGGTGCGAAG
GACCATGAGAACAAGCTGCAATGGGTGTAGGGTTCTTCGCAAGGCATGCAGCCAAGACTGCATCA
Header
>VIT_201s0011g03540.1
Sequence
CAGGTAGCGTGAAGTTAAACCCTAGCGCTTTAGACAAACAGCTGTAGTCACCGCCCACAAACACC
AGCCTCTGAGACACCACCTCAAACCTTTCCACTTAAATACACATCCCTCACACCCTTTTCAATTC
Header
>VIT_201s0011g03550.1
Sequence
CATGCAAAGCTGAACGCGATGCTGTGATTGGTGGTAAGTGGTAGTTGAGTAAATTTGACAGTGAA
GCCGAAATGGTAAAAGACTAAGGCTAGAAGTAGAATACCACTGTTCTTCTCATCACGTGGGCCCA

Insper

Prática

Map/Reduce | Otimização

- Nossa tarefa:
  - Utilizando o arquivo SARS-CoV-2-Wuhan-NC_045512.2.fasta
  - Contar cada aminoácido de DNA das sequências mapeadas neste arquivo
    - Você deve filtrar a linha que se inicia com >
  - Calcule a frequência relativa de cada nucleotídeo

- Faremos uso da estrutura Map / Reduce

```
[('g', 19.60672842189747),
 ('c', 18.366050229074006),
 ('a', 29.943483931378122),
 ('t', 32.083737417650404)]
```

- Na sequência, faça o mesmo para o arquivo SARS-CoV-2-Washington_MT293201.1.fasta, obtendo a frequência relativa de cada nucleotídeo para essa nova sequência

Podemos melhorar o desempenho?

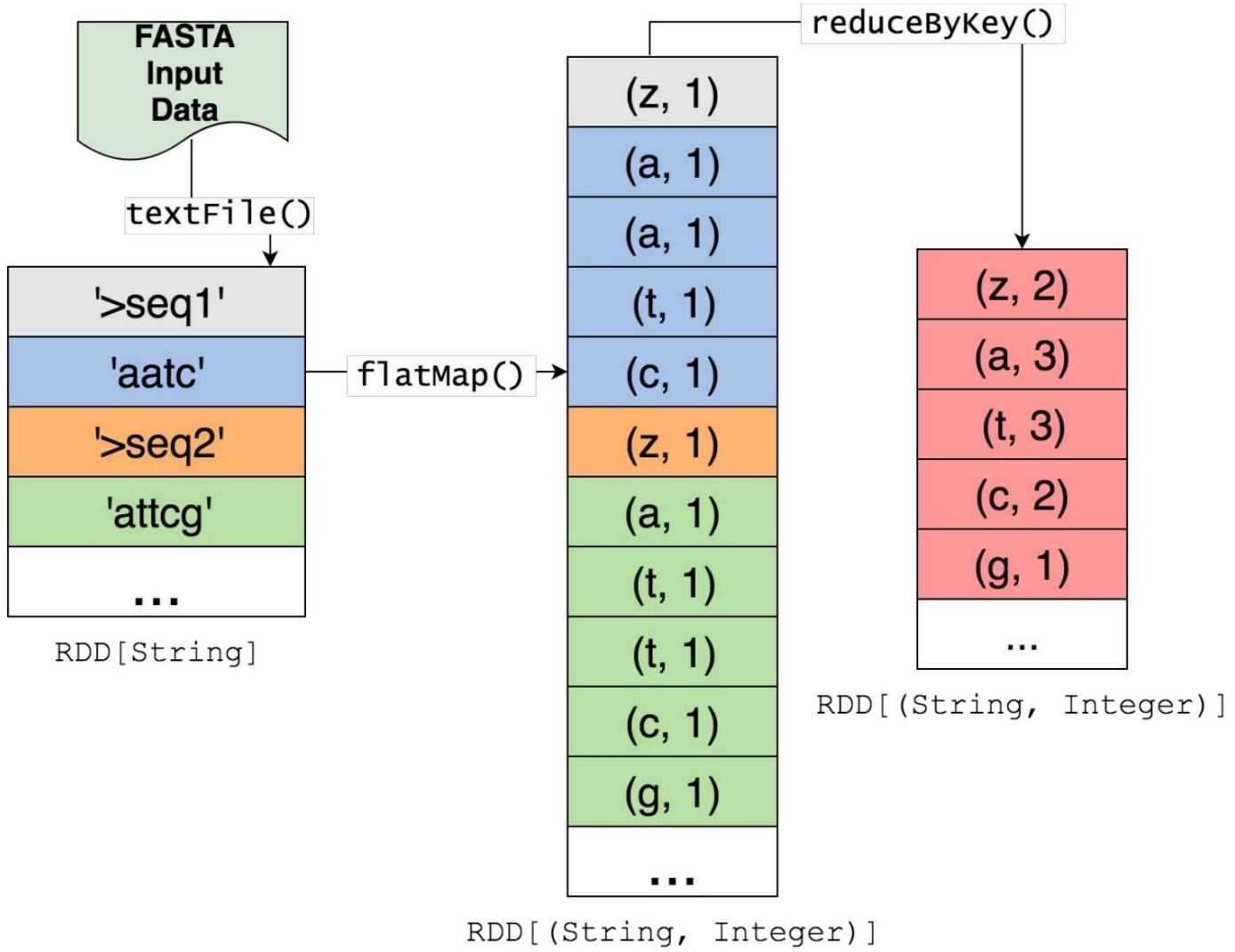
Genoma com RDD

```
'TCTTCGTAAGAACGGTAATAAAGGAGCTGGTGGCCATAGTTACGGCGCCGATCTAAAGTCATTTGACTTA'
```

```
def process_FASTA_record(record):
    key_value_list = []
    chars = record.lower()
    for c in chars:
        key_value_list.append((c,1))
    return key_value_list
```

Estratégia I

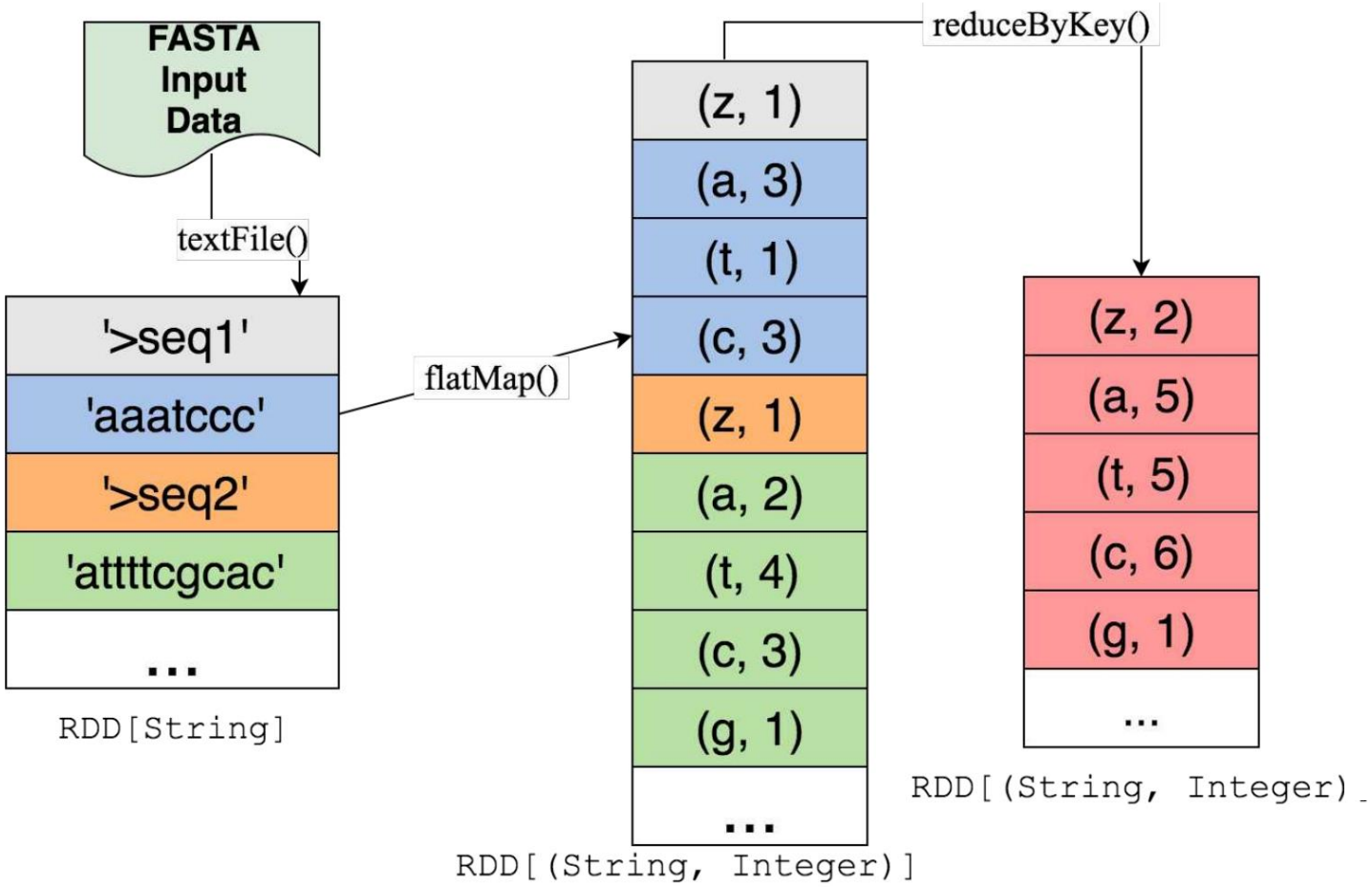
Utilizando Map / Reduce Clássico



Insper

Estratégia II

Otimizando o FlatMap para contabilizar as sequências

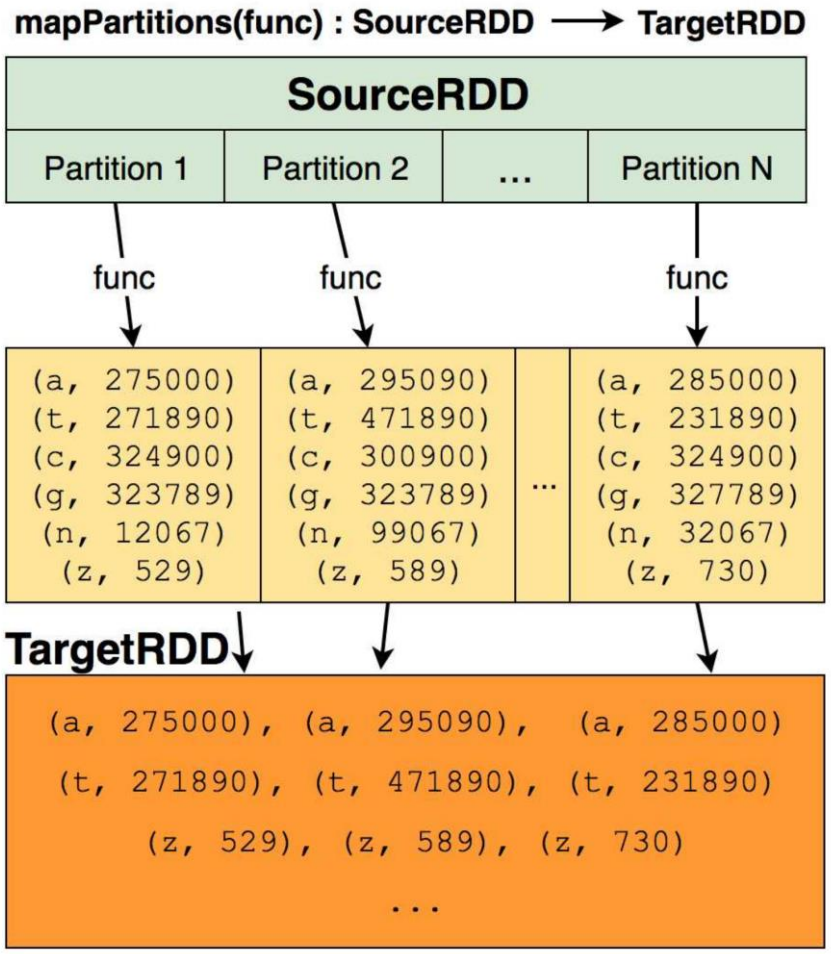


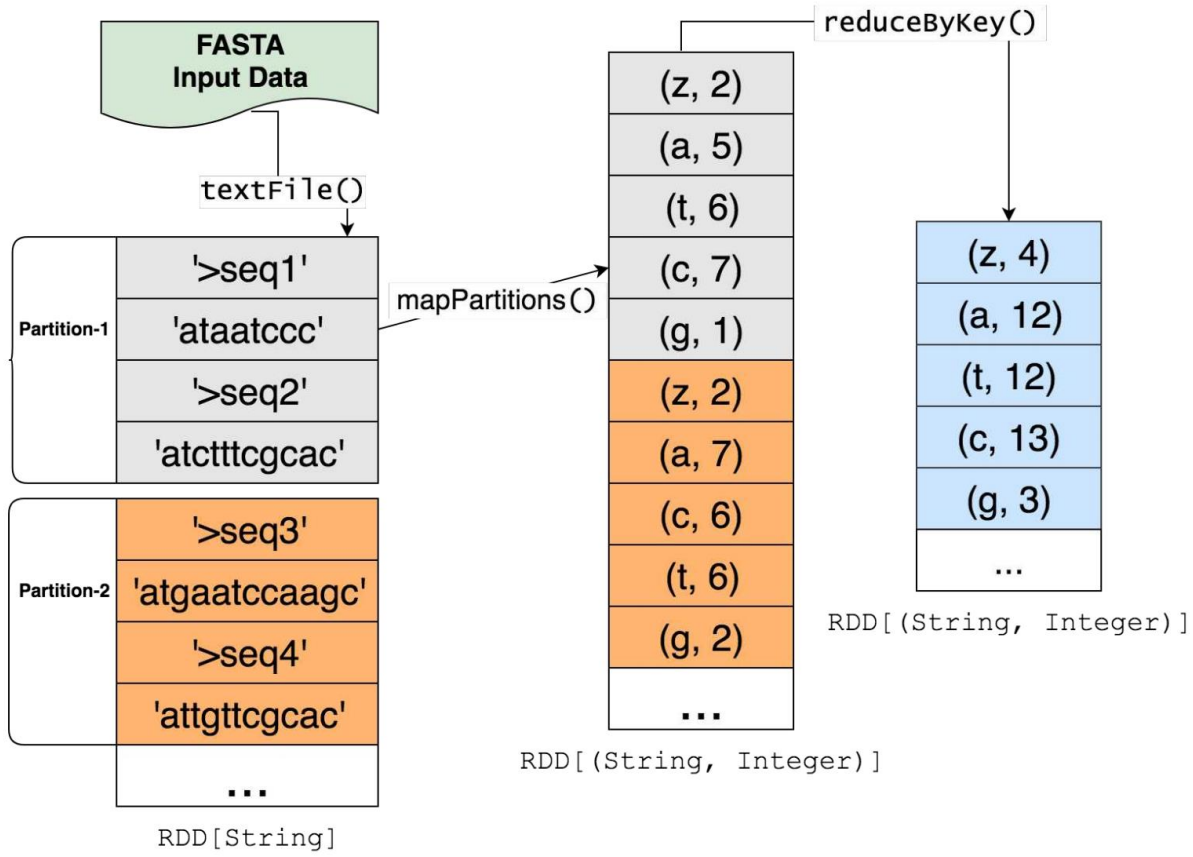
Version 1	Version 2
(a, 1)	(a, 5)
(a, 1)	(t, 3)
(a, 1)	(c, 1)
(t, 1)	(g, 4)
(t, 1)	
(t, 1)	
(c, 1)	
(g, 1)	
(g, 1)	
(g, 1)	
(g, 1)	
(a, 1)	
(a, 1)	

Insper

Estratégia III

Utilizando MapPartitions





Insper

Insper