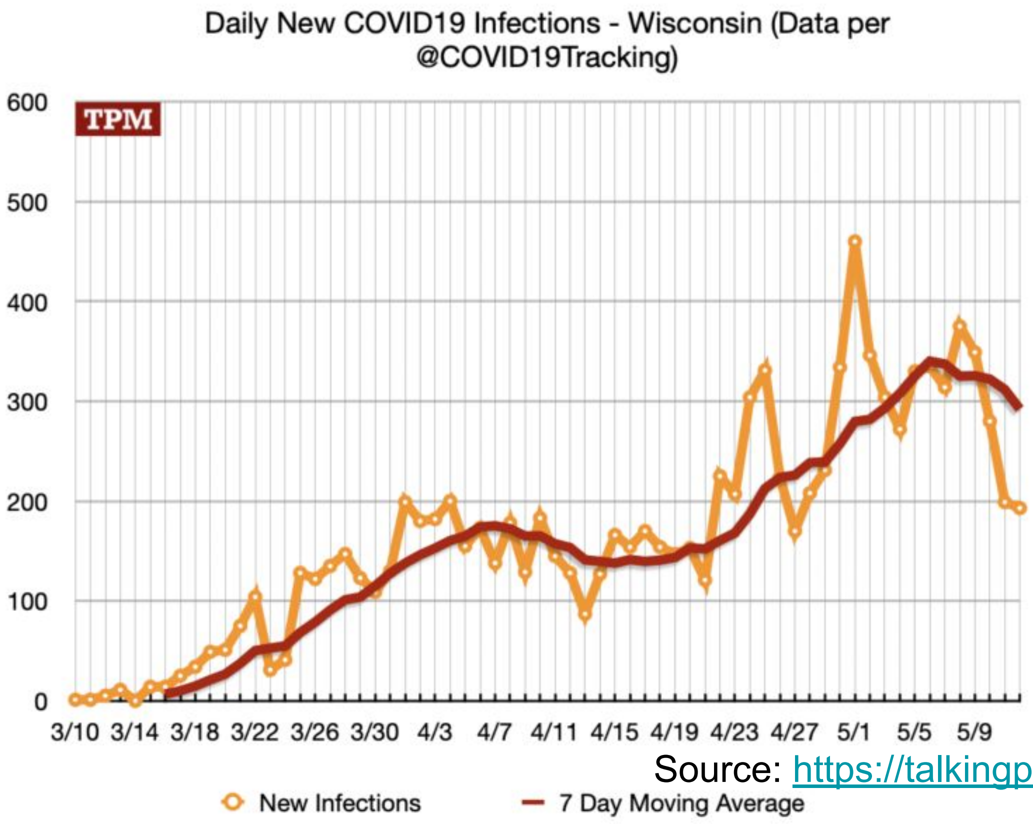
CSC 369

Introduction to Distributed Computing

Computing Moving Average



Smooth, captures trends

Precise, obscures trends

Source: https://talkingpointsmemo.com/edblog/looking-at-the

Moving Average

Problem: given COVID-19 data, report k-day moving averages for new infections for each state.

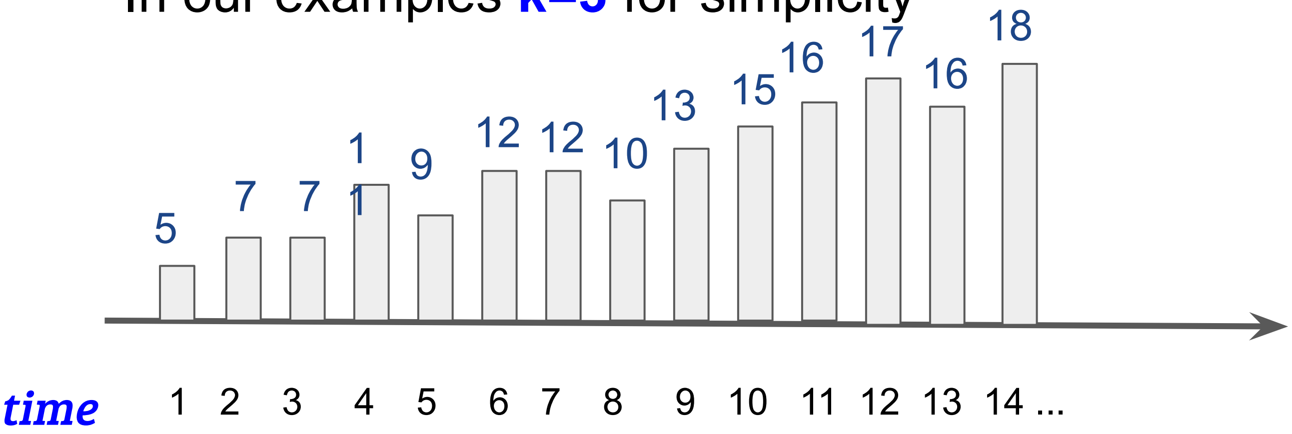
In our examples **k=3** for simplicity

Data Source:
https://api.covidtracking.com/v1/states/wi/daily.csv
The "positiveIncrease" field represents new cases.

Moving Average

Problem: given COVID-19 data, report k-day moving averages for new infections for each state.

In our examples **k=3** for simplicity

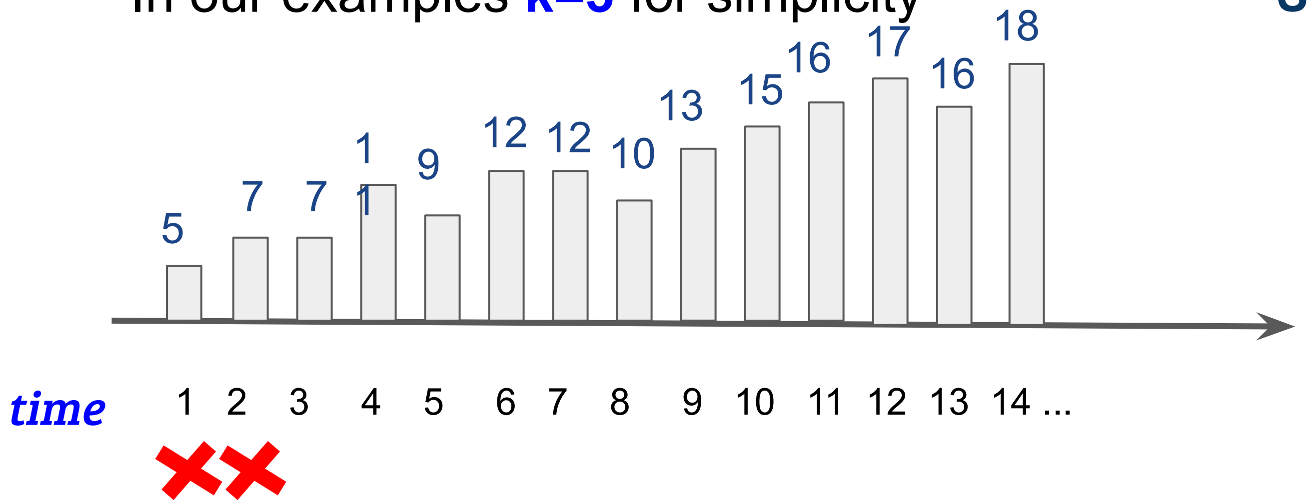


Moving Average

Problem: given COVID-19 data, report k-day moving averages for new infections for each state.

In our examples **k=3** for simplicity

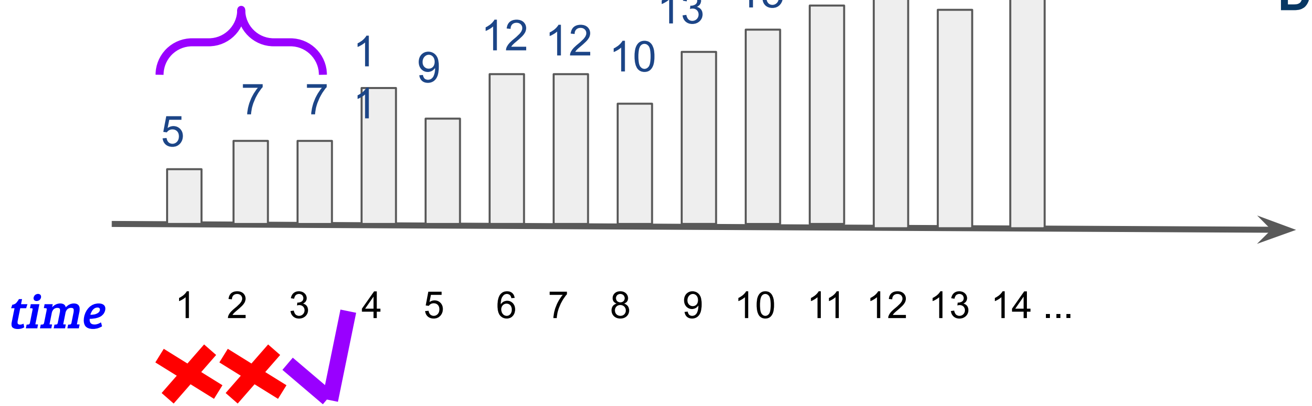
Skip days 1, and 2



Moving Average

Problem: given COVID-19 data, report k-day moving averages for new infections for each state.

In our examples **k=3** for simplicity



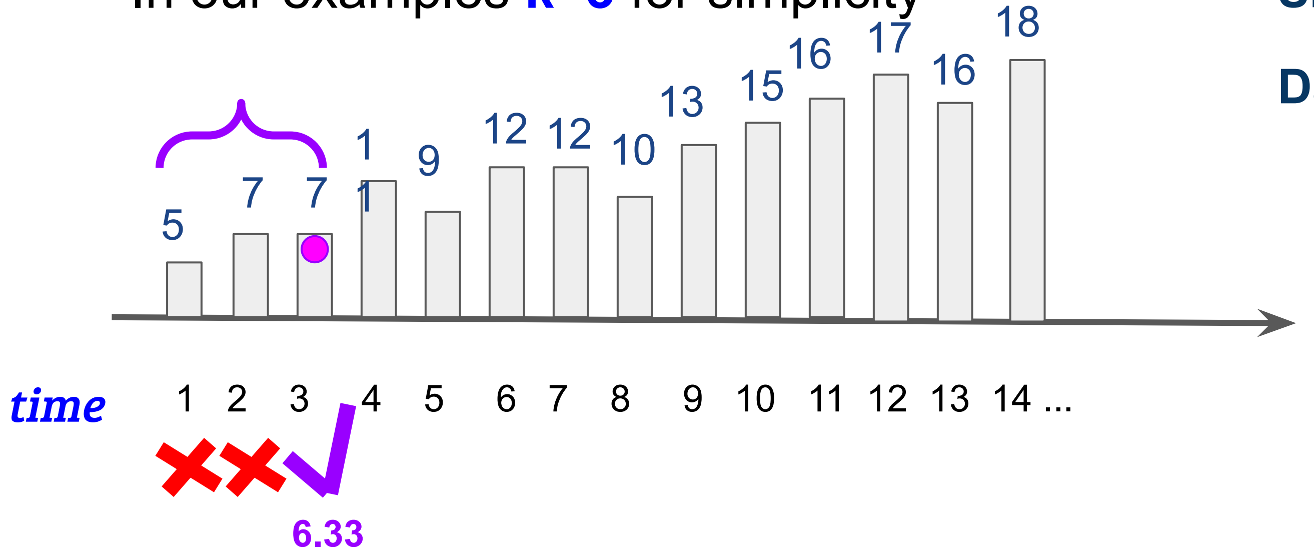
Skip days 1, and 2

Day 3: include 1,2,3

Moving Average

Problem: given COVID-19 data, report k-day moving averages for new infections for each state.

In our examples **k=3** for simplicity



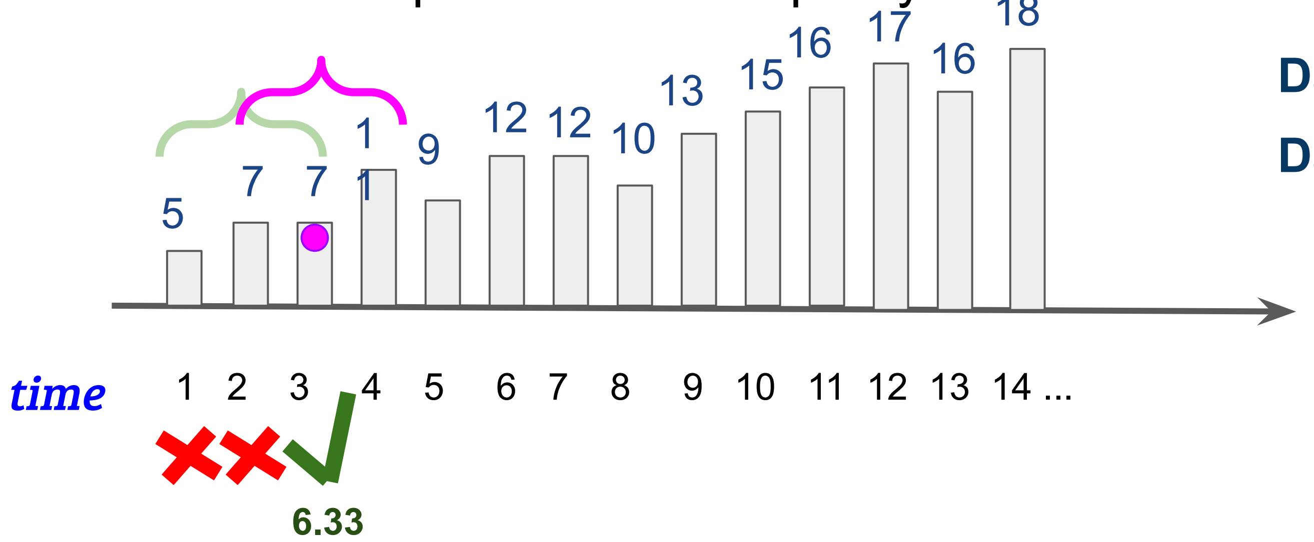
Skip days 1, and 2

Day 3: include 1,2,3

Moving Average

Problem: given COVID-19 data, report k-day moving averages for new infections for each state.

In our examples **k=3** for simplicity



Skip days 1, and 2

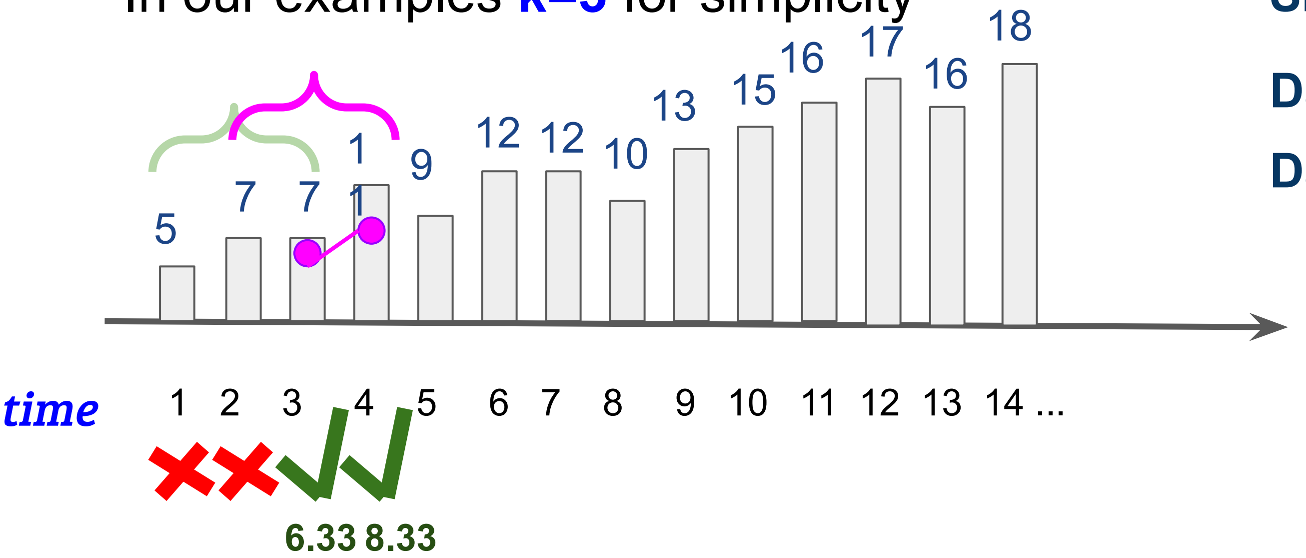
Day 3: include 1,2,3

Day 4: include 2,3, 4

Moving Average

Problem: given COVID-19 data, report k-day moving averages for new infections for each state.

In our examples **k=3** for simplicity



Skip days 1, and 2

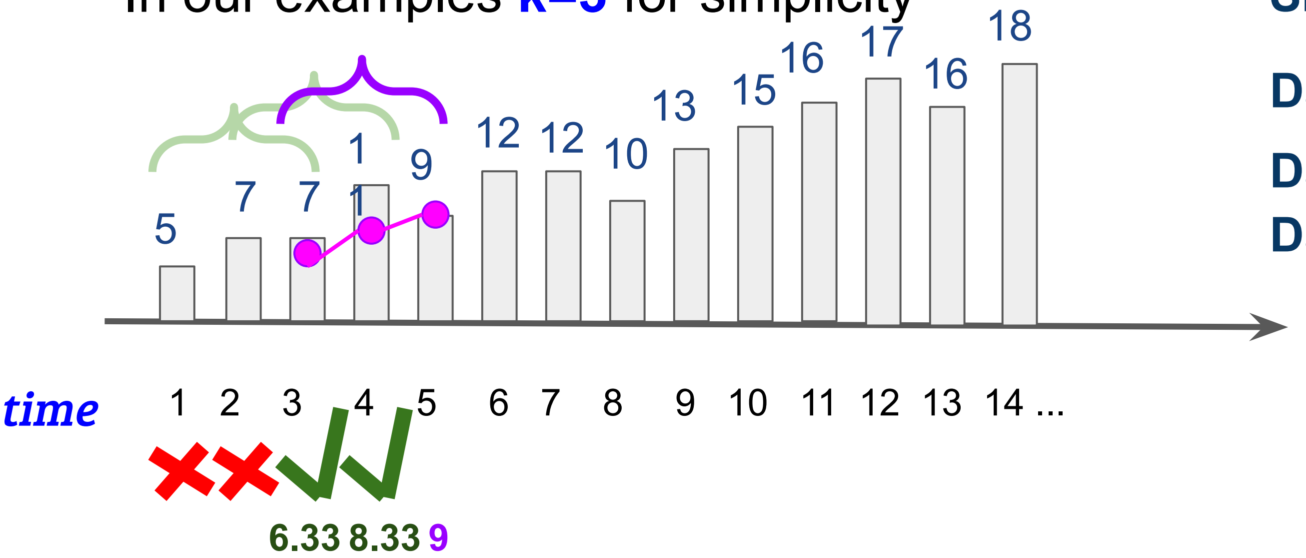
Day 3: include 1,2,3

Day 4: include 2,3, 4

Moving Average

Problem: given COVID-19 data, report k-day moving averages for new infections for each state.

In our examples **k=3** for simplicity



Skip days 1, and 2

Day 3: include 1,2,3

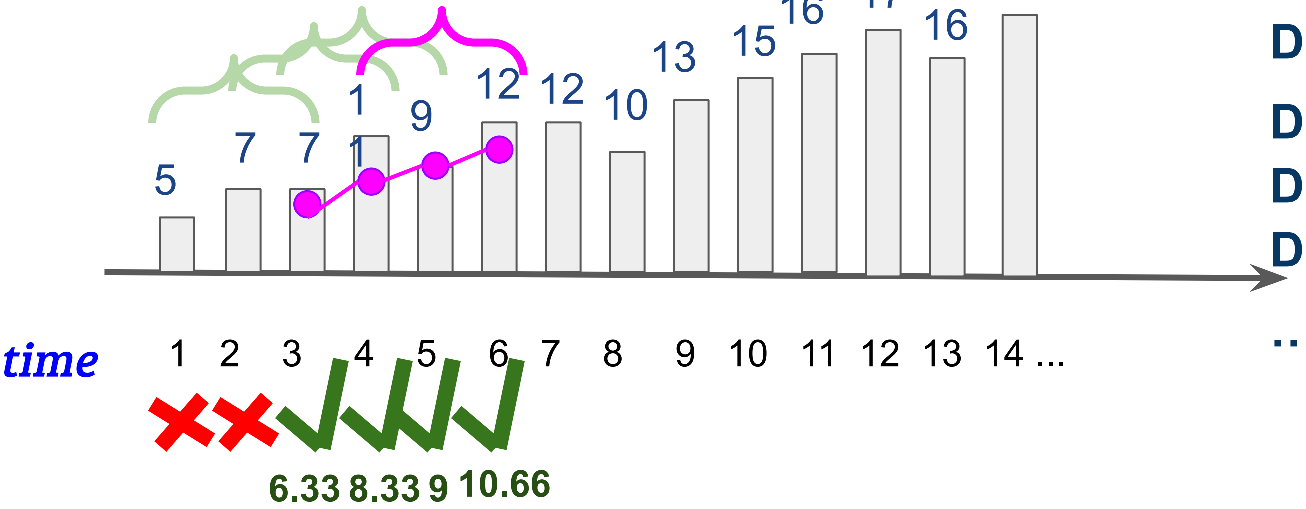
Day 4: include 2,3, 4

Day 5: include 3, 4, 5

Moving Average

Problem: given COVID-19 data, report k-day moving averages for new infections for each state.

In our examples **k=3** for simplicity



Skip days 1, and 2

Day 3: include 1,2,3

Day 4: include 2,3, 4

Day 5: include 3, 4, 5

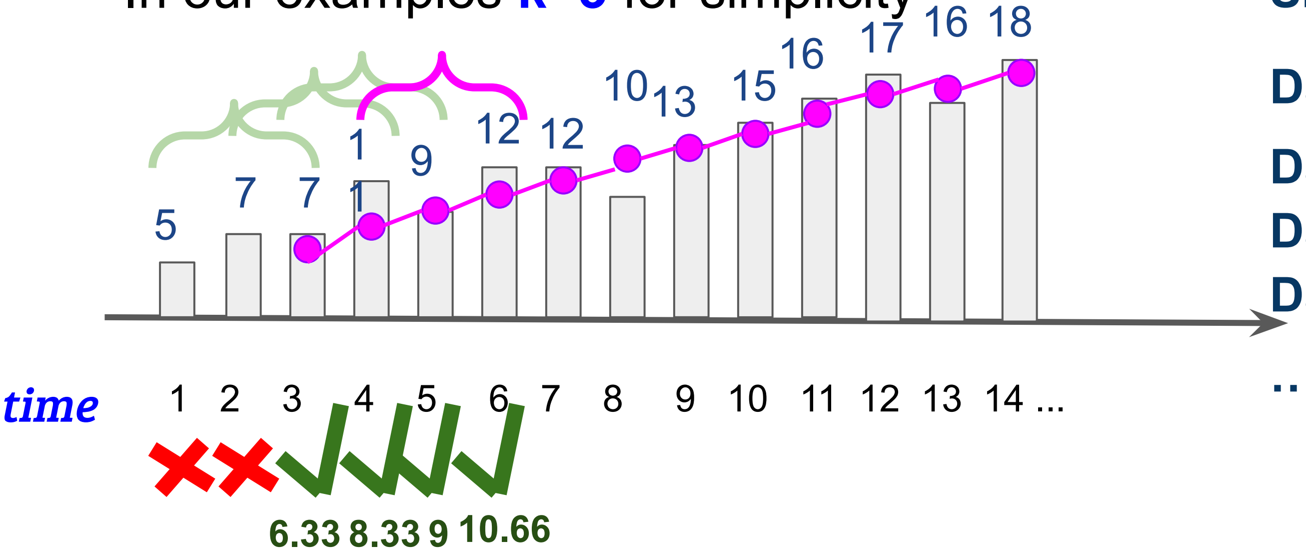
Day 6: include 4, 5, 6

… and so on

Moving Average

Problem: given COVID-19 data, report k-day moving averages for new infections for each state.

In our examples **k=3** for simplicity



Skip days 1, and 2

Day 3: include 1,2,3

Day 4: include 2,3, 4

Day 5: include 3, 4, 5

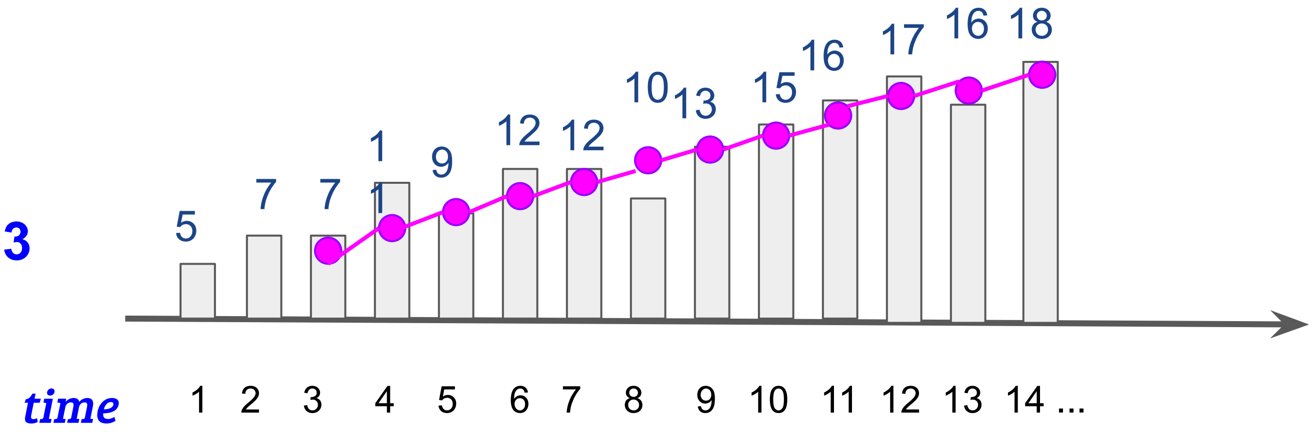
Day 6: include 4, 5, 6

… and so on

Moving Average via MapReduce

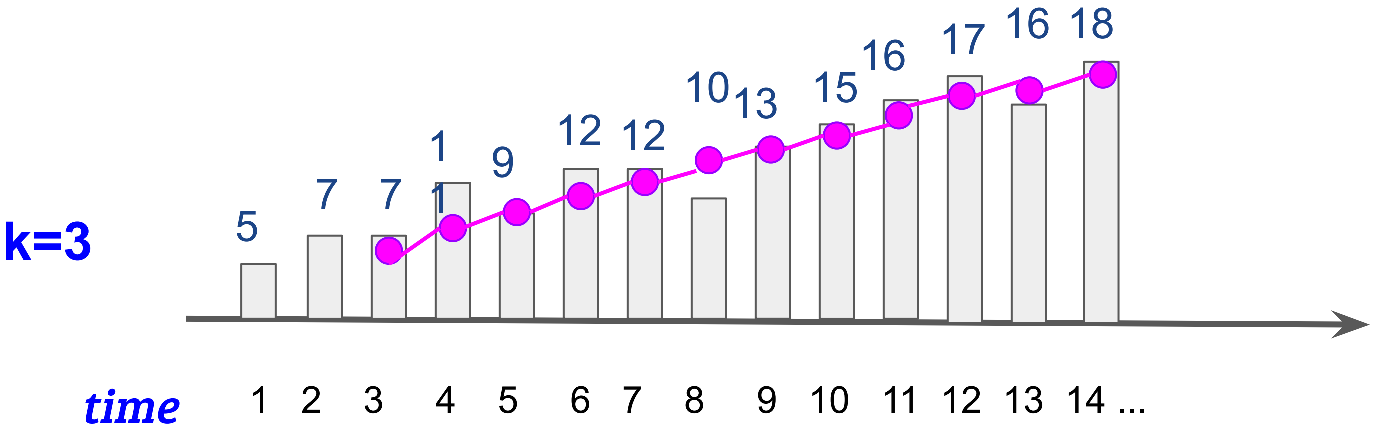
Input: <Day, Total>

k=3



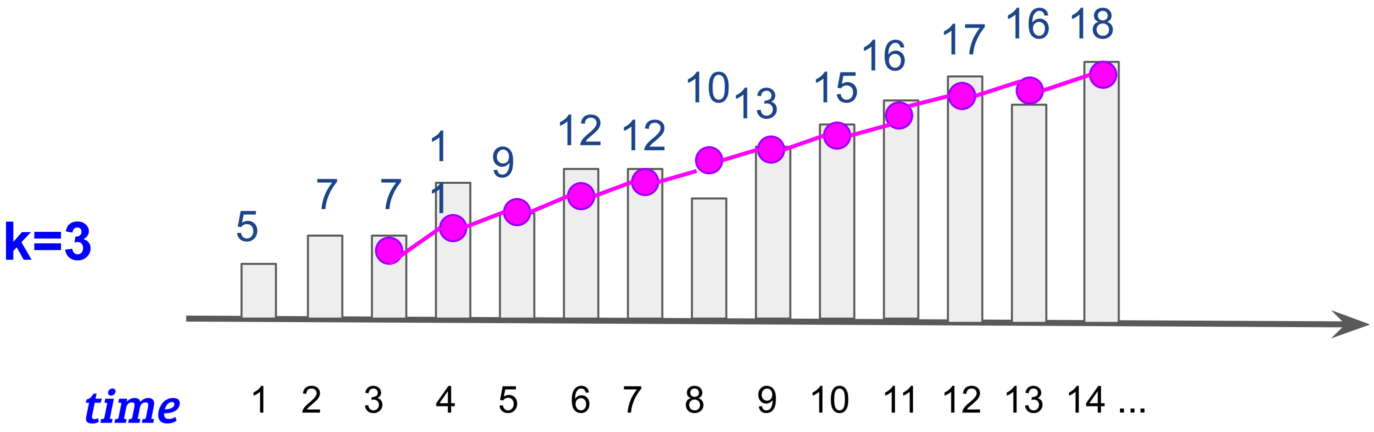
Moving Average via MapReduce

Input: <Day, Total> → map() → ? → reduce() →



Moving Average via MapReduce

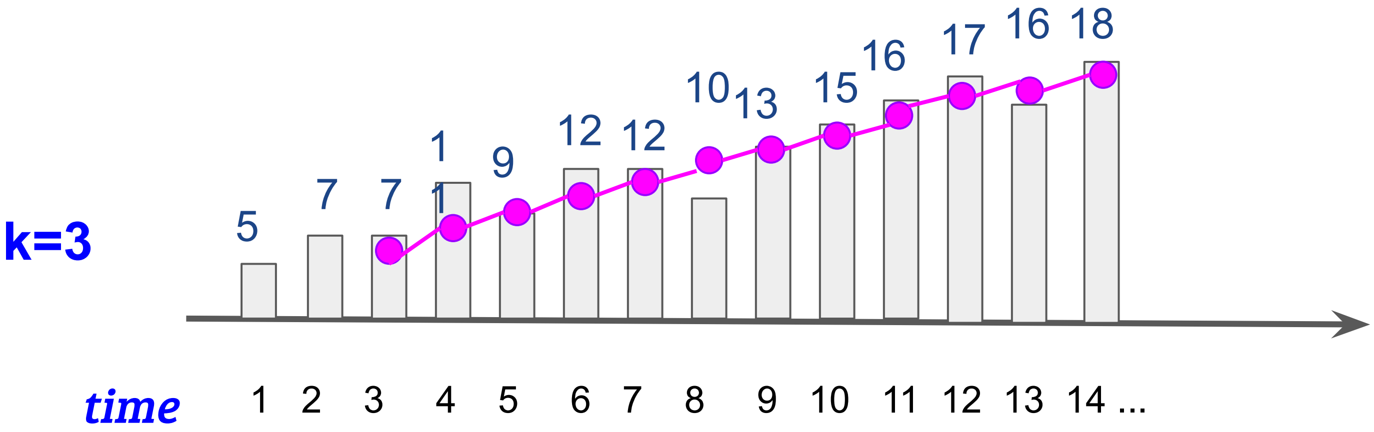
Input: <Day, Total> → map() → **?** → reduce() → <Day, Moving Average>



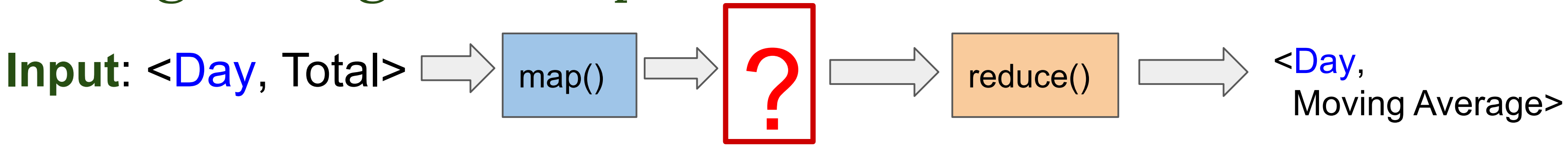
Moving Average via MapReduce

Input: <Day, Total> → map() → ? → reduce() → <Day, Moving Average>

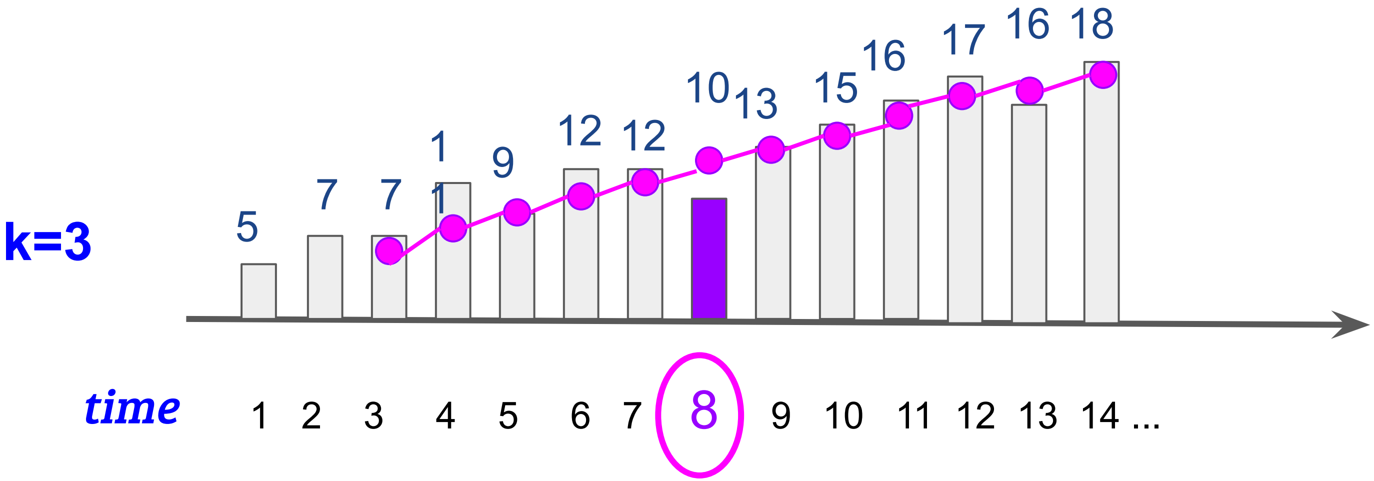
What should **reduce()** see?



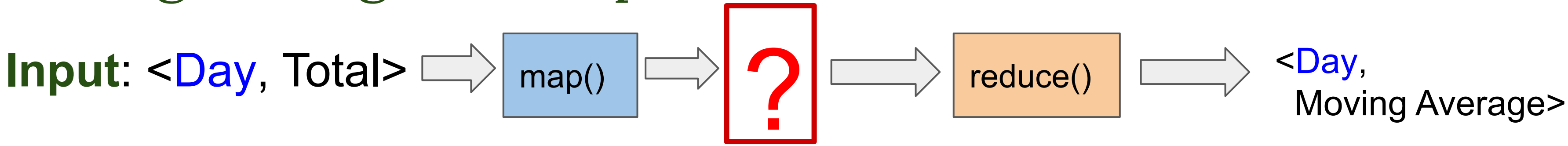
Moving Average via MapReduce



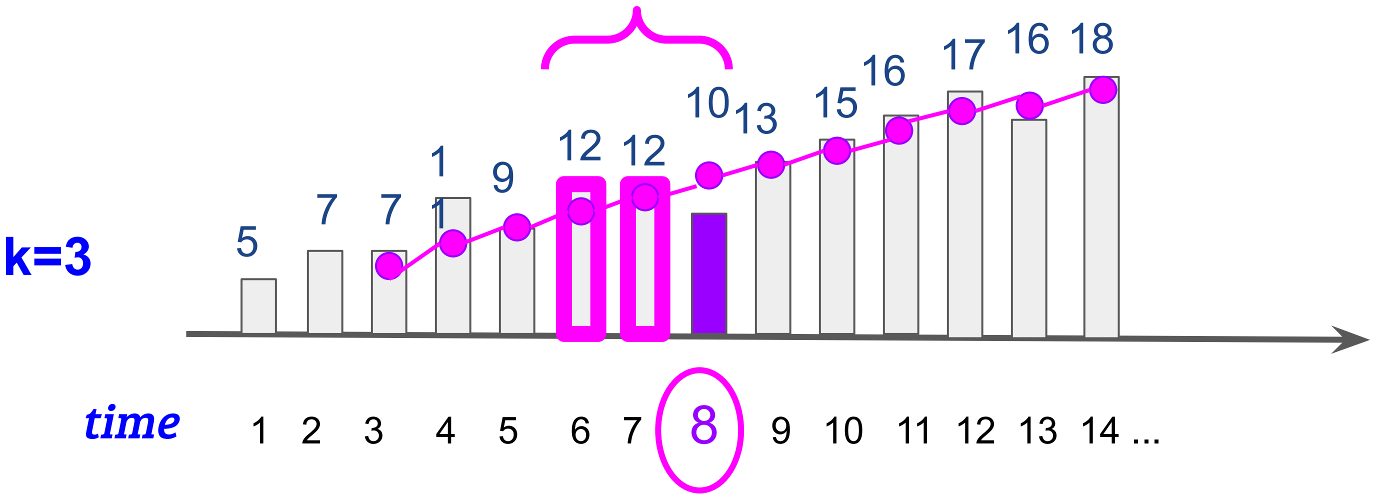
<Day: 8, Values: [???]>



Moving Average via MapReduce



<Day: 8, Values: [???]>

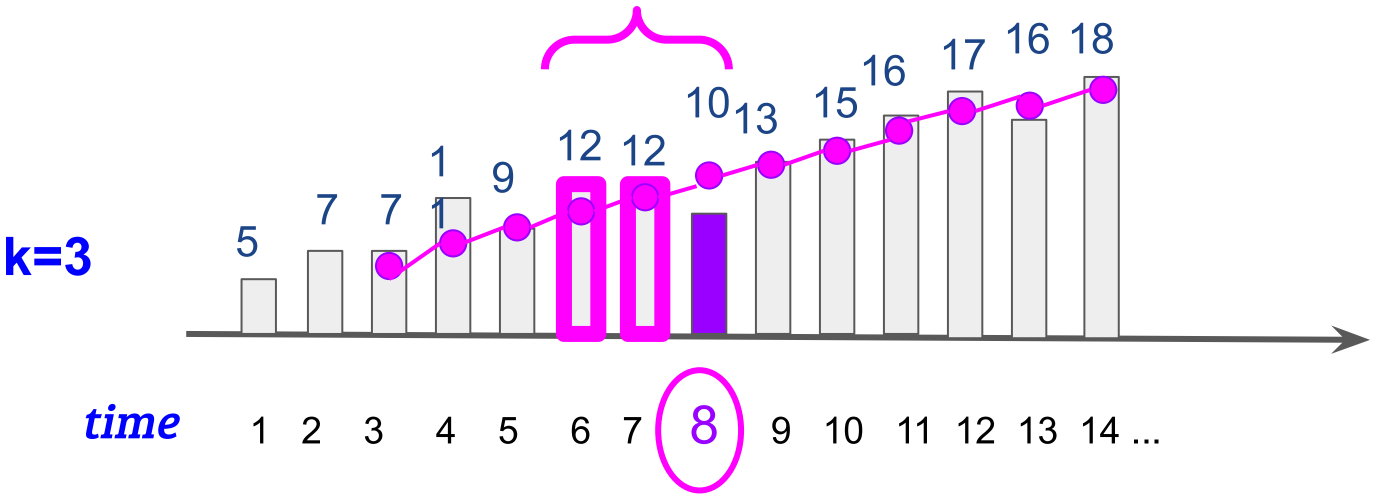


Moving Average via MapReduce

Input: <Day, Total> → map() → ? → reduce() → <Day, Moving Average>

<Day: 8, Values: [12, 12, 10]>

d-2 d-1 d

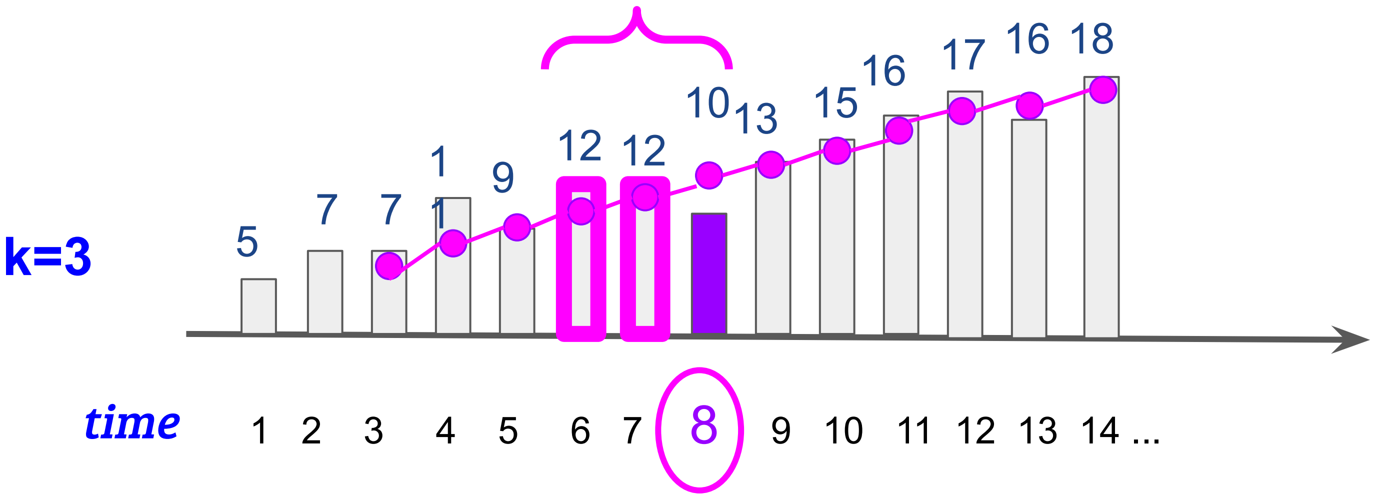


Moving Average via MapReduce

Input: <Day, Total> → map() → ? → reduce() → <Day, Moving Average>

<Day: 8, Values: [12, 12, 10]>

d-2 d-1 d



reduce() writes itself

```
reduce( key, Iterable values):
    sum = 0
    count = 0
    for v in values do
        sum = sum + v
        count = count + 1
    end for
    movingAverage = sum/count
    emit (key, movingAverage)
```

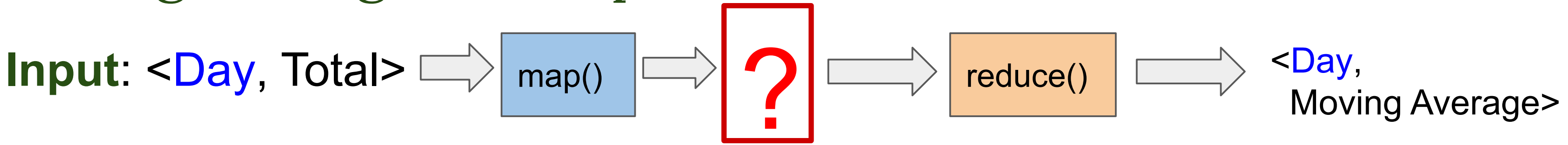
What about map()?

```
map(key, value):    // key: date;  value: total
```

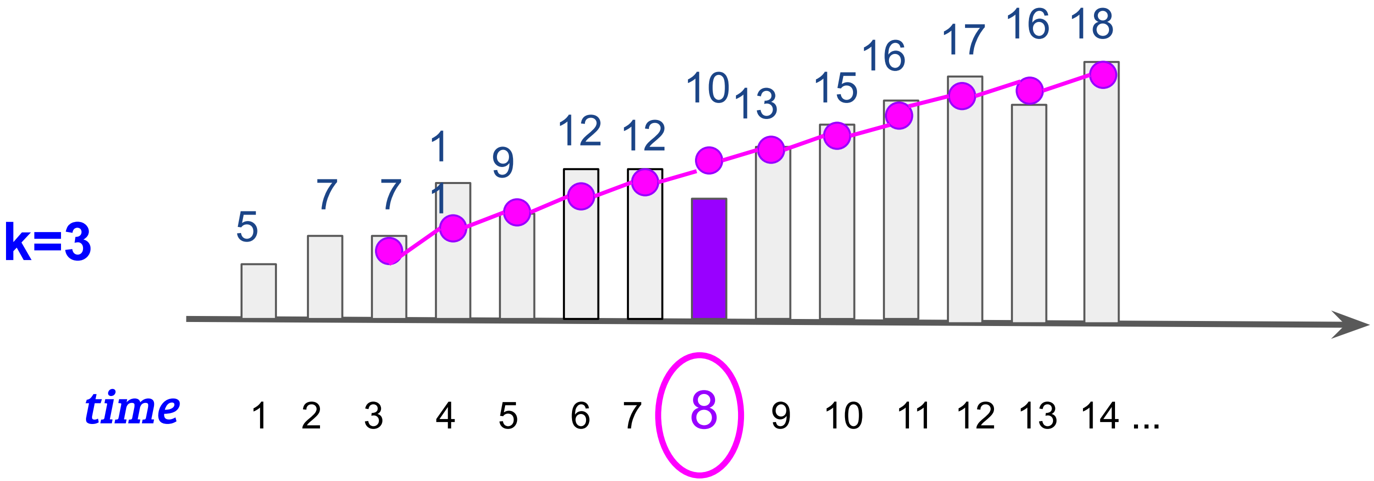
What about map()?

```
map(key, value):    // key: date;  value: total

// what computations will need our value???
```

Moving Average via MapReduce



<Day:8, Total:10>

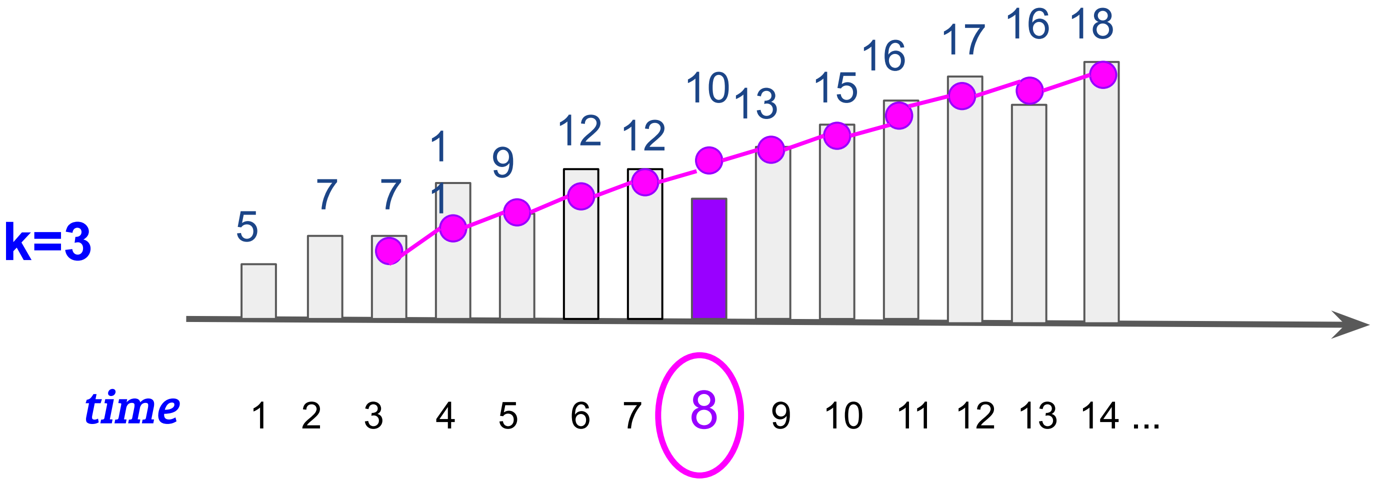


Moving Average via MapReduce

Input: <Day, Total> → map() → **?** → reduce() → <Day, Moving Average>

<Day:8, Total:10>

Which Reduce() computations will Day 8 participate in?

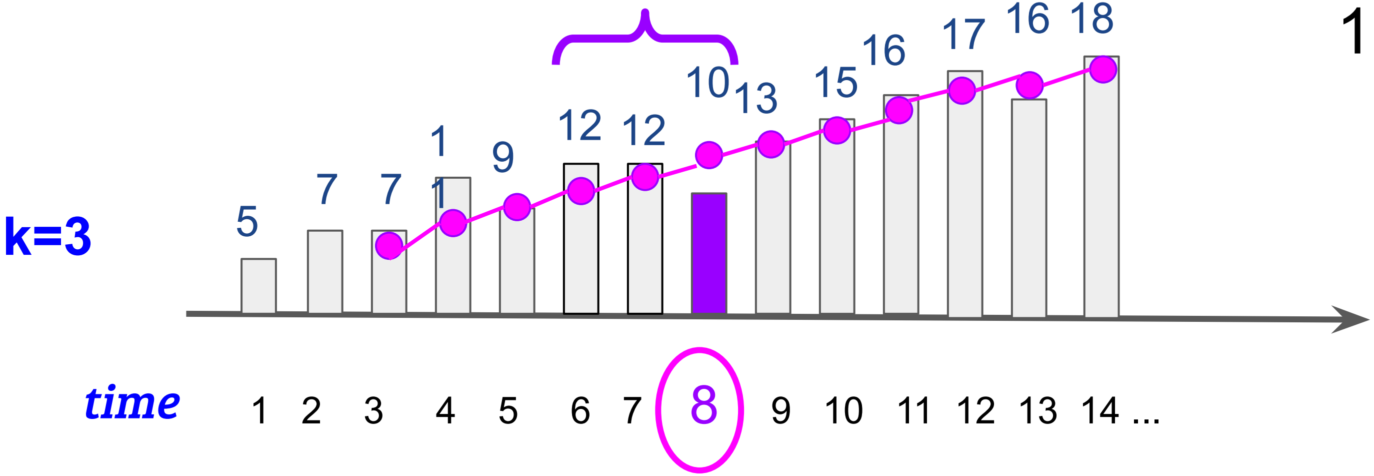


Moving Average via MapReduce

Input: <Day, Total> → map() → ? → reduce() → <Day, Moving Average>

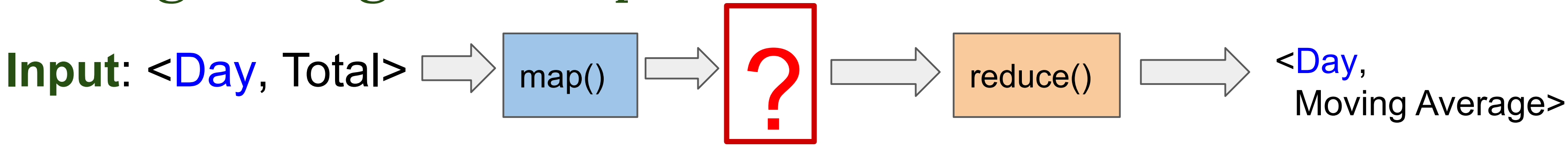
<Day:8, Total:10>

Which Reduce() computations will Day 8 participate in?



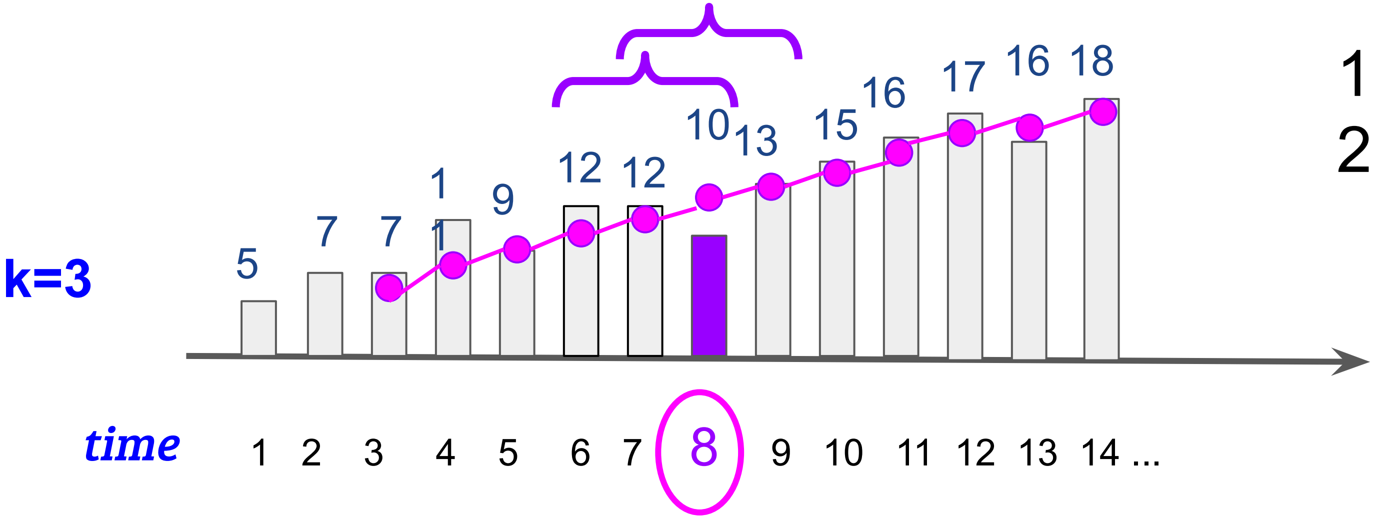
1. Day 8

Moving Average via MapReduce



<Day:8, Total:10>

Which Reduce() computations will Day 8 participate in?



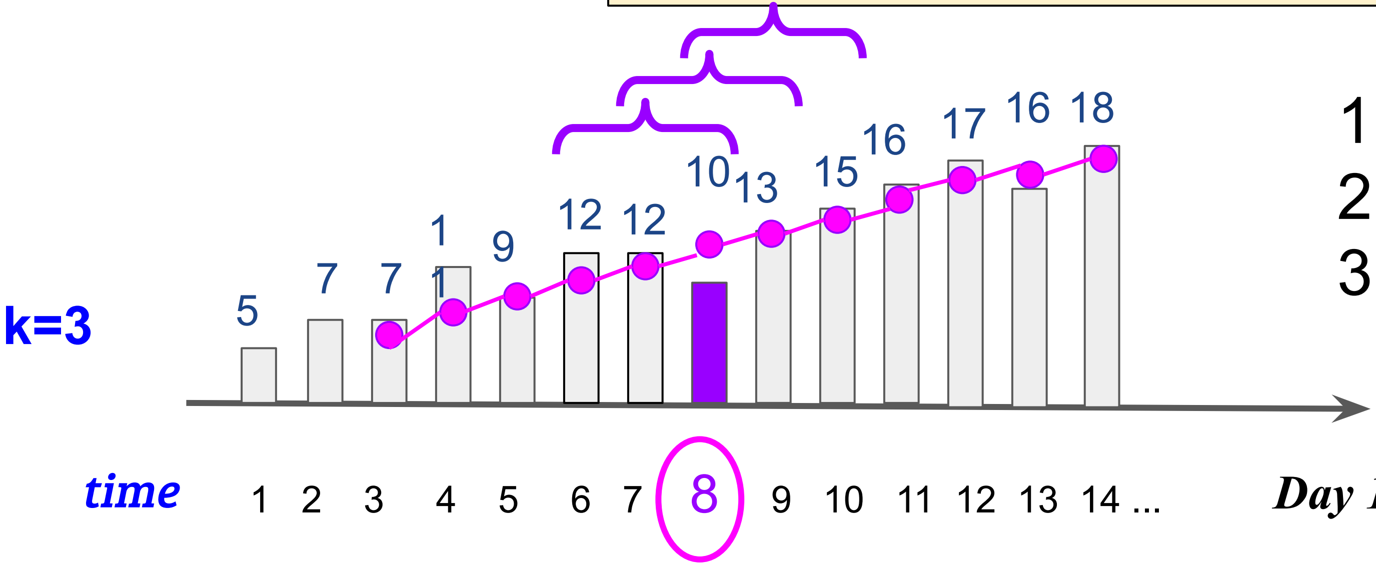
1. Day 8
2. Day 9

Moving Average via MapReduce

Input: <Day, Total> → map() → ? → reduce() → <Day, Moving Average>

<Day:8, Total:10>

Which Reduce() computations will Day 8 participate in?



1. Day 8
2. Day 9
3. Day 10

Day 11 needs Days 9,10,11

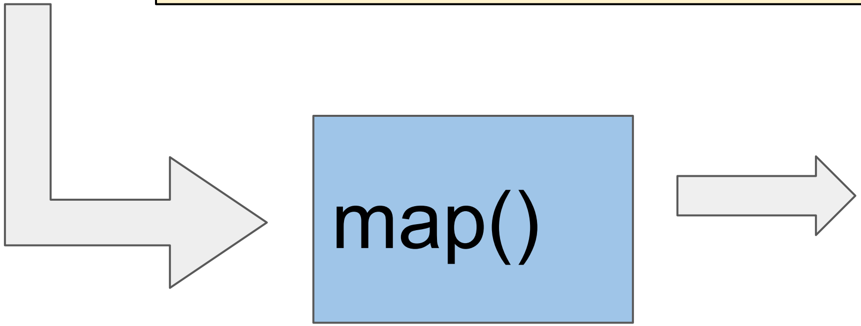
<Day:8, Total:10>

Which Reduce() computations will Day 8 participate in?

1. Day 8
2. Day 9
3. Day 10

<Day:8, Total:10>

Which Reduce() computations will Day 8 participate in?



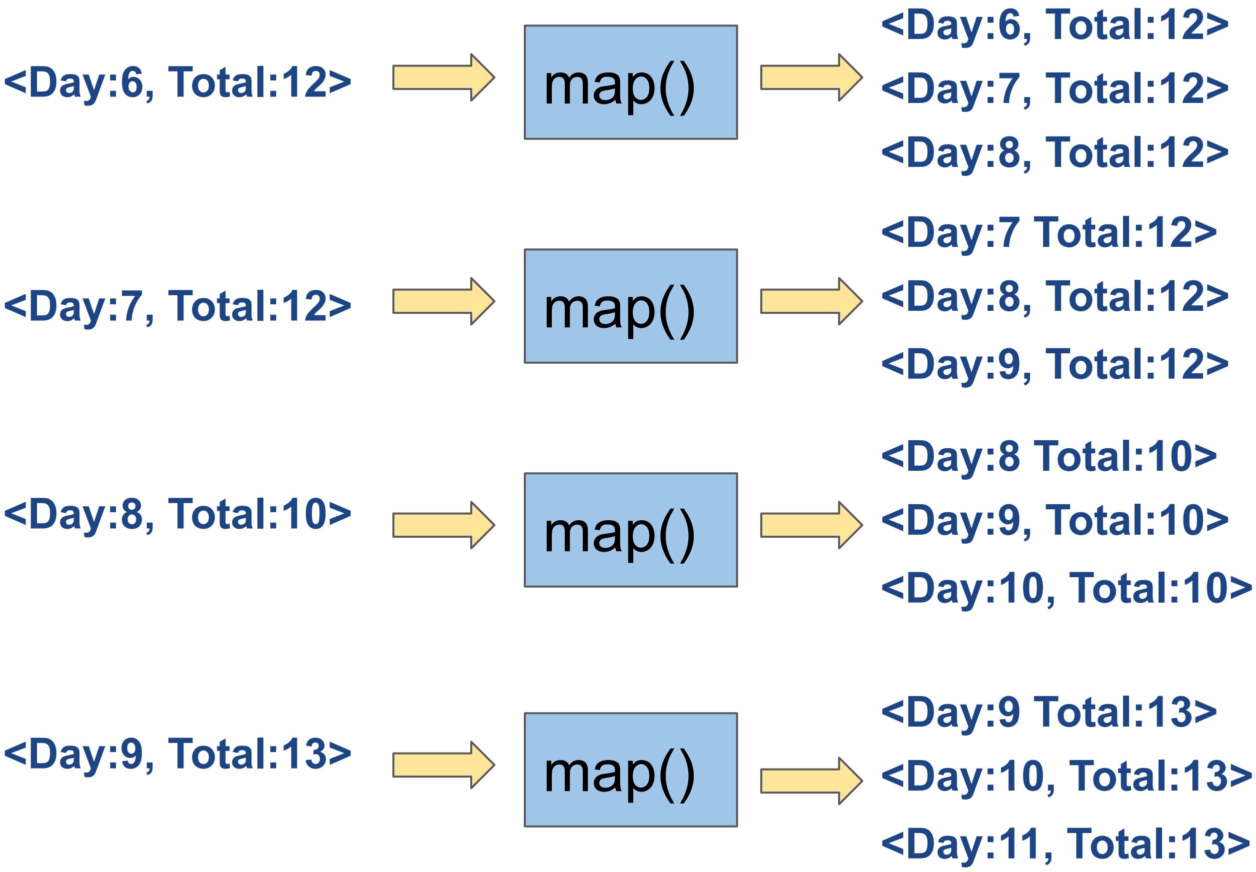
1. Day 8
2. Day 9
3. Day 10

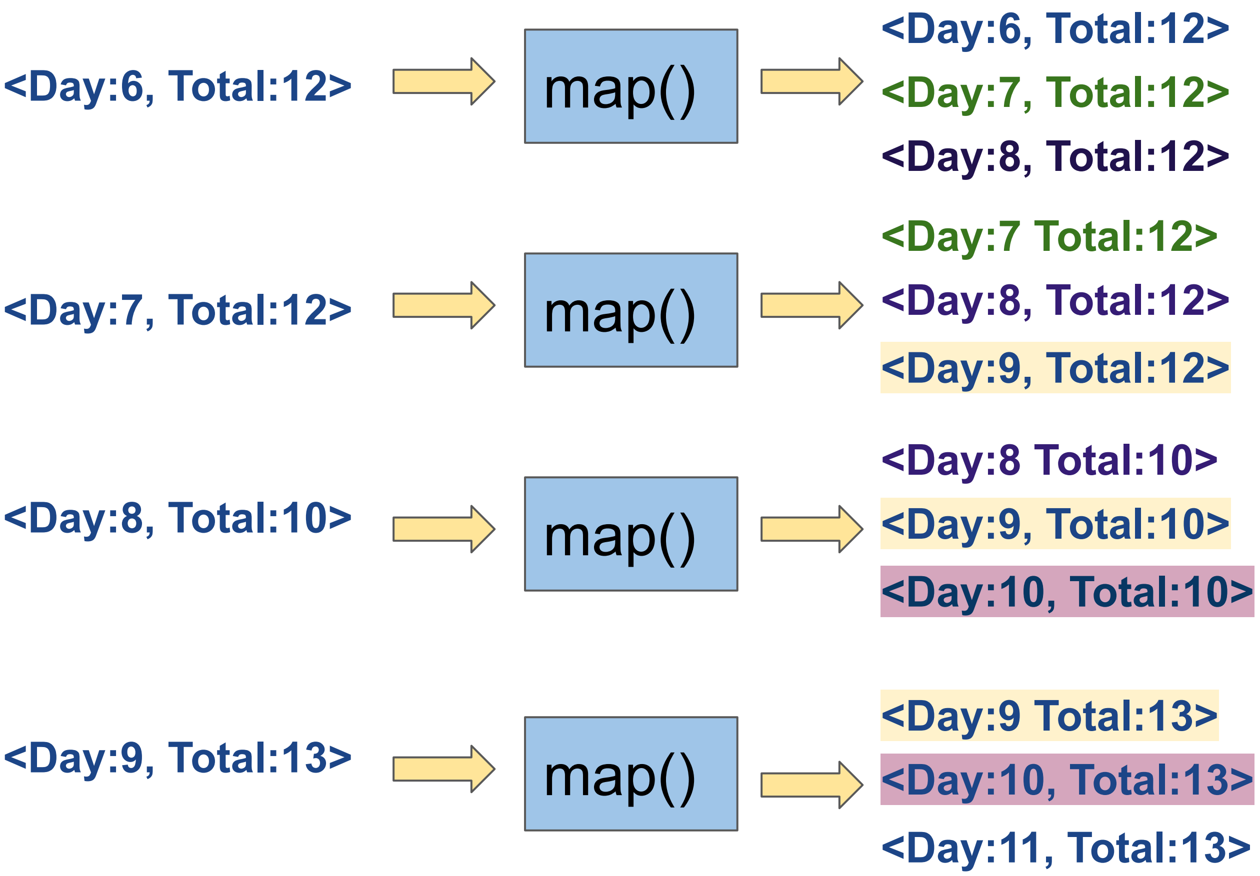
And now, map() writes itself too

<Day:8, Total:10>

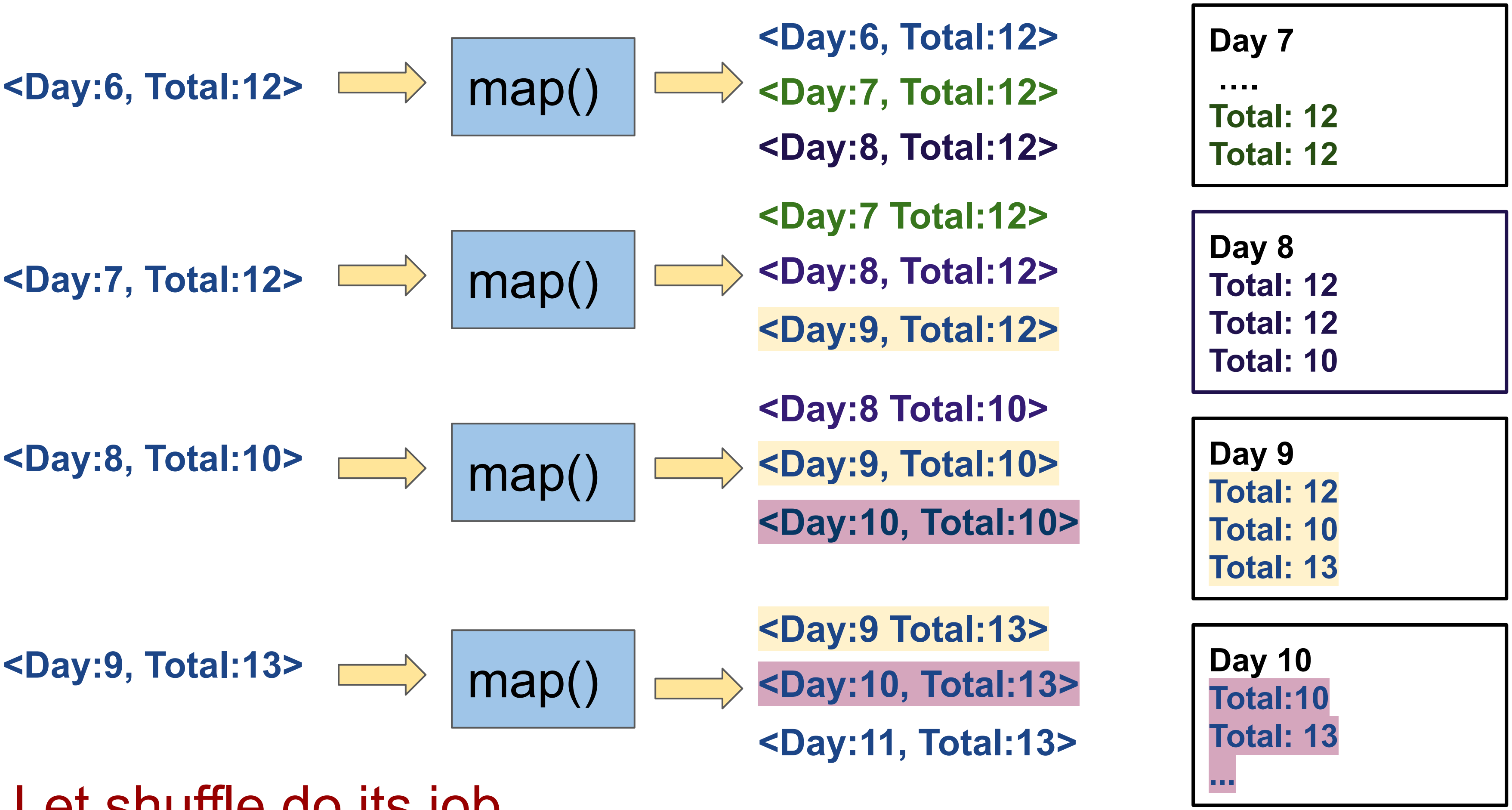
1. Day 8
2. Day 9
3. Day 10

```
map(key, value):    // key: date;  value: total

  movingWindow = 3 // set moving window

  for i = 0 to movingWindow-1 do
    newKey = key+i
    emit(key,value)
  end for
```


<Day:6, Total:12>
map()
<Day:6, Total:12>
<Day:7, Total:12>
<Day:8, Total:12>
<Day:7, Total:12>
map()
<Day:7 Total:12>
<Day:8, Total:12>
<Day:9, Total:12>
<Day:8, Total:10>
map()
<Day:8 Total:10>
<Day:9, Total:10>
<Day:10, Total:10>
<Day:9, Total:13>
map()
<Day:9 Total:13>
<Day:10, Total:13>
<Day:11, Total:13>



Let shuffle do its job



Let shuffle do its job

Moving Average

Solution idea for map(): we know what keys will need **"our"** value

Moving Average

Solution idea for map(): we know what keys will need **"our"** value

Transfers over to other problems

Matrix Multiplication

Why Matrix Multiplication?

Lies at the heart of many Machine Learning methods

- ➢ Linear regression
- ➢ Support Vector Machines
- ➢ Neural Networks

Used in many graph algorithms

- ➢ Multiplying adjacency matrices by themselves

Multiplying matrices

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} \times \begin{pmatrix} b11 & b12 \\ b21 & b22 \\ b31 & b32 \end{pmatrix}$$

3 x 3 3 x 2

Multiplying matrices

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} \quad X \quad \begin{pmatrix} b11 & b12 \\ b21 & b22 \\ b31 & b32 \end{pmatrix}$$

3×3 $\qquad$ 3×2

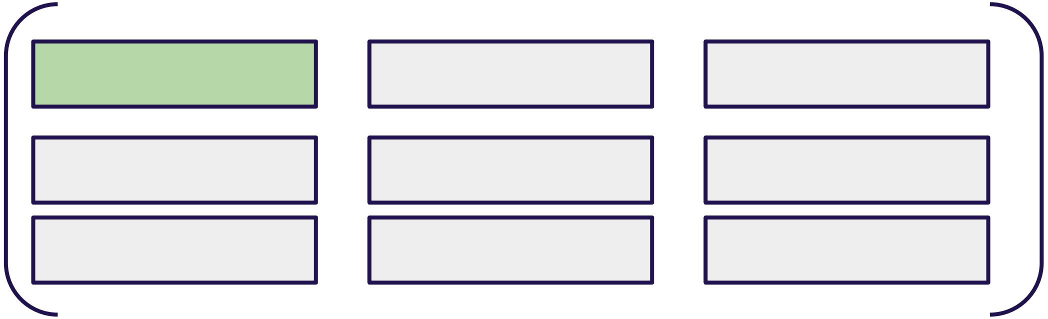
$$\begin{pmatrix} \boxed{} & \boxed{} \\ \boxed{} & \boxed{} \\ \boxed{} & \boxed{} \end{pmatrix}$$

3×2

Multiplying matrices

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} \quad X \quad \begin{pmatrix} b11 & b12 \\ b21 & b22 \\ b31 & b32 \end{pmatrix}$$

3 x 3   3 x 2



3 x 2

Multiplying matrices

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} \quad \times \quad \begin{pmatrix} b11 & b12 \\ b21 & b22 \\ b31 & b32 \end{pmatrix}$$

3 x 3   3 x 2

$$\begin{pmatrix} \boxed{a11*b11+a12*b21+a13*b31} & \boxed{} \\ \boxed{} & \boxed{} \\ \boxed{} & \boxed{} \end{pmatrix}$$

3 x 2

Multiplying matrices

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} \quad X \quad \begin{pmatrix} b11 & b12 \\ b21 & b22 \\ b31 & b32 \end{pmatrix}$$

3 x 3   3 x 2

$$\begin{pmatrix} \boxed{\textbf{a11*b11+a12*b21+a13*b31}} & \boxed{} \\ \boxed{\textbf{a21*b11+a22*b21+a23*b31}} & \boxed{} \\ \boxed{} & \boxed{} \end{pmatrix}$$

3 x 2

Multiplying matrices

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} \quad X \quad \begin{pmatrix} b11 & b12 \\ b21 & b22 \\ b31 & b32 \end{pmatrix}$$

3 x 3   3 x 2

$$\begin{pmatrix} \text{a11*b11+a12*b21+a13*b31} & \\ \text{a21*b11+a22*b21+a23*b31} & \\ \text{a31*b11+a32*b21+a33*b31} & \end{pmatrix}$$

3 x 2

Multiplying matrices

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} \quad X \quad \begin{pmatrix} b11 & b12 \\ b21 & b22 \\ b31 & b32 \end{pmatrix}$$

3 x 3   3 x 2

$$\begin{pmatrix} \boxed{\text{a11*b11+a12*b21+a13*b31}} & \boxed{\text{a11*b21+a12*b22+a13*b23}} \\ \boxed{\text{a21*b11+a22*b21+a23*b31}} & \boxed{\text{a21*b21+a22*b22+a23*b23}} \\ \boxed{\text{a31*b11+a32*b21+a33*b31}} & \boxed{\text{a31*b21+a32*b22+a33*b32}} \end{pmatrix}$$

3 x 2

Matrix Multiplication via MapReduce

Input

Two files

Row-wise format

Includes row number

Input Version 1

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a23 & a33 \end{pmatrix} \times \begin{pmatrix} b11 & b12 \\ b21 & b22 \\ b31 & b32 \end{pmatrix}$$

A.csv

```
1, a11, a12, a13
2, a21, a22, a23
3, a31, a32, a33
```

B.csv

```
1, b11, b12
2, b21, b22
3, b31, b32
```

Input Version 1

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a23 & a33 \end{pmatrix} \times \begin{pmatrix} b11 & b12 \\ b21 & b22 \\ b31 & b32 \end{pmatrix}$$

A.csv

```
1, a11, a12, a13
2, a21, a22, a23
3, a31, a32, a33
```

B.csv

```
1, b11, b12
2, b21, b22
3, b31, b32
```

Matrix Multiplication via MapReduce: Mappers

A.csv

```
1, a11, a12, a13
2, a21, a22, a23
3, a31, a32, a33
```

```
map(key, value)
```

Matrix Multiplication via MapReduce: Mappers

A.csv

```
1, a11, a12, a13
2, a21, a22, a23
3, a31, a32, a33
```

Let’s use THE SAME idea as for moving average

```
map(key, value)
```

Matrix Multiplication via MapReduce: Reduce!

A.csv

```
1, a11, a12, a13
2, a21, a22, a23
3, a31, a32, a33
```

Let's use THE SAME idea as for moving average

```
reduce(key, Iterable values)
```

Matrix Multiplication via MapReduce: Reduce!

A.csv

```
1, a11, a12, a13
2, a21, a22, a23
3, a31, a32, a33
```

B.csv

```
1, b11, b12
2, b21, b22
3, b31, b32
```

```
reduce(key, Iterable values)
```

`key` **(rowID, columnID)**

`values`

[(1,A, a11), (2,A, a12), (3,A,a13), (1,B, b11), (2,B, b21), (3,B,b31)]

Position, Matrix, Value

Matrix Multiplication via MapReduce: Reduce!

A.csv

```
1, a11, a12, a13
2, a21, a22, a23
3, a31, a32, a33
```

B.csv

```
1, b11, b12
2, b21, b22
3, b31, b32
```

```
reduce(key, Iterable values)
```

a11	a12	a13
x	x	x
b11	b21	b31

Matrix Multiplication via MapReduce: Reduce!

A.csv

```
1, a11, a12, a13
2, a21, a22, a23
3, a31, a32, a33
```

B.csv

```
1, b11, b12
2, b21, b22
3, b31, b32
```

```
reduce(key, Iterable values)
```

+	a11	a12	a13
	x	x	x
	b11	b21	b31

Matrix Multiplication via MapReduce: Reduce!

A.csv

```
1, a11, a12, a13
2, a21, a22, a23
3, a31, a32, a33
```

B.csv

```
1, b11, b12
2, b21, b22
3, b31, b32
```

```
reduce(key, Iterable values)
```

[(1,A, a11), (2,A, a12), (3,A,a13), (1,B, b11), (2,B, b21), (3,B,b31)]

Identify Source Matrix
Identify Position for dot-product computation
Identify Value

Matrix Multiplication via MapReduce: Reduce!

A.csv

```
1, a11, a12, a13
2, a21, a22, a23
3, a31, a32, a33
```

B.csv

```
1, b11, b12
2, b21, b22
3, b31, b32
```

```
reduce(key, Iterable values)
```

[(1,1,A, a11), (1,2,A, a12), (1,3,A,a13), (1,1,B, b11), (1,2,B, b21), (1,3,B,b31)]

Identify Source Matrix
Identify Position for dot-product computation
Identify Value

map()

Matrix Multiplication via MapReduce: Mappers

`A.csv`

```
1, a11, a12, a13
2, a21, a22, a23
3, a31, a32, a33
```

a11 is needed for computing cells (1,1), (1,2)

```
mapA(key, value)
// output the values for each computation they are in
// key = rowId
// reduce key is <rowId, colId>
for each pos = 1 to rowSize do
   cell = value[pos]
   for column = 1 to maxColumn do
      emit((key,column), ("A",pos, cell))
   end for
end for
```

Matrix Multiplication via MapReduce: Mappers

B.csv

```
1, b11, b12
2, b21, b22
3, b31, b32
```

```
mapB(key, value)
// output the values for each computation they are in
// key = rowId
// reduce key is <rowId, colId>
for each pos = 1 to rowSize do
  cell = value[pos]
  for column = 1 to maxColumn do
    emit((pos,column), ("B",row, cell))
  end for
end for
```

Matrix Multiplication via MapReduce: Mappers

B.csv

```
1, b11, b12
2, b21, b22
3, b31, b32
```

b11 needs to go to reduce keys (1,1), (2,1), (3,1)
b12 needs to go to reduce keys (1,2), (2,2), (3,2)

```
mapB(key, value)
// output the values for each computation they are in
// key = rowId
// reduce key is <rowId, colId>
for each pos = 1 to rowSize do  //rowSize is for B
  cell = value[pos]
  for row = 1 to maxRow do      //maxRow is for A
    emit((row,pos), ("B",key, cell))
  end for
end for
```

Distributed Batch Processing - MapReduce

- Partitioning
  - Splitting data to make high-volume storage possible and to allow efficient distributed processing: (a) input data partitioning, and (b) repartitioning for processing by multiple reducers

- Fault Tolerance
  - MapReduce writes to disk ("materializes" data) frequently. This simplifies recovery from task failure (hardware, software, network, human) at the cost of slower overall processing