DTAI
Declaratieve Talen en Artificiële Intelligentie

Big ML

Some tools & techniques for large-scale ML

Joaquin Vanschoren

Universiteit Leiden

Big$_{ML}$

Some tools & techniques for large-scale ML

Joaquin Vanschoren

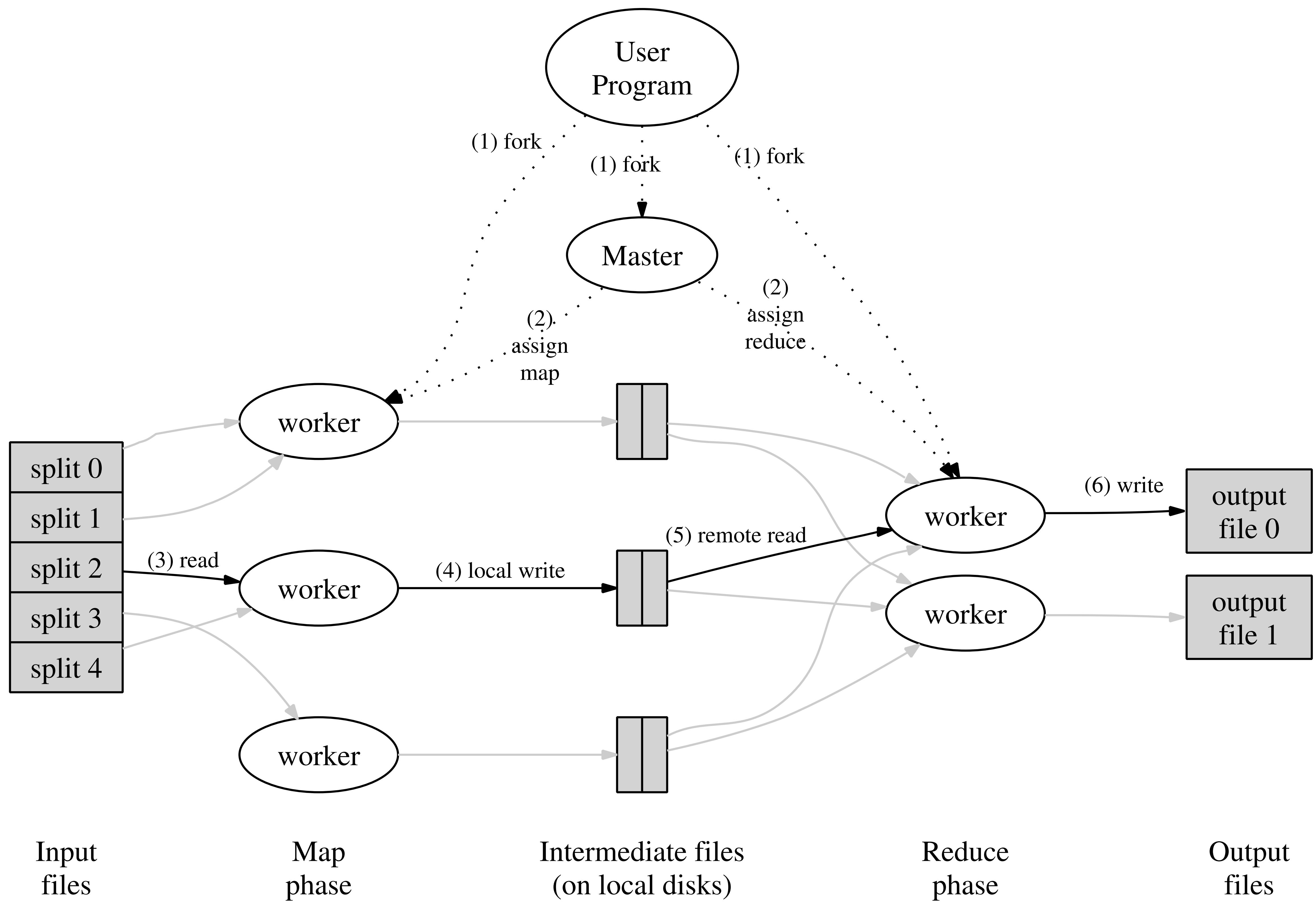
The Why

- A lot of ML problems concern really large datasets
  - Huge graphs: internet, social networks, protein interactions
  - Sensor data (cfr. Kristian)
    - InfraWatch: 5GB/day, 2TB/year, 8TB/4years   @50MB/s -> 2 days
  - Data is just getting bigger...

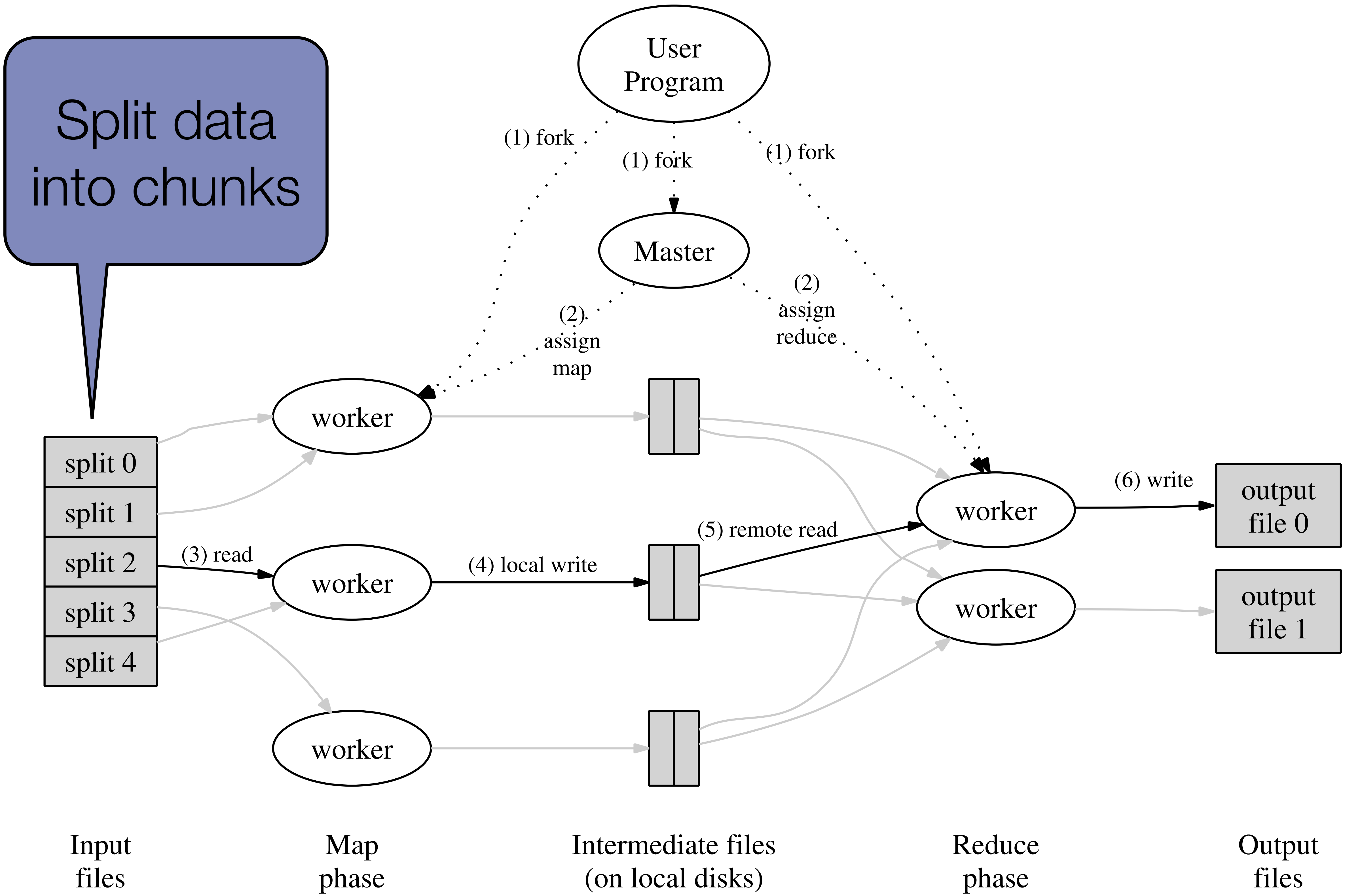
The What

- Map-Reduce (Hadoop) -> I/O bound processes
- Distributed computing -> CPU bound processes

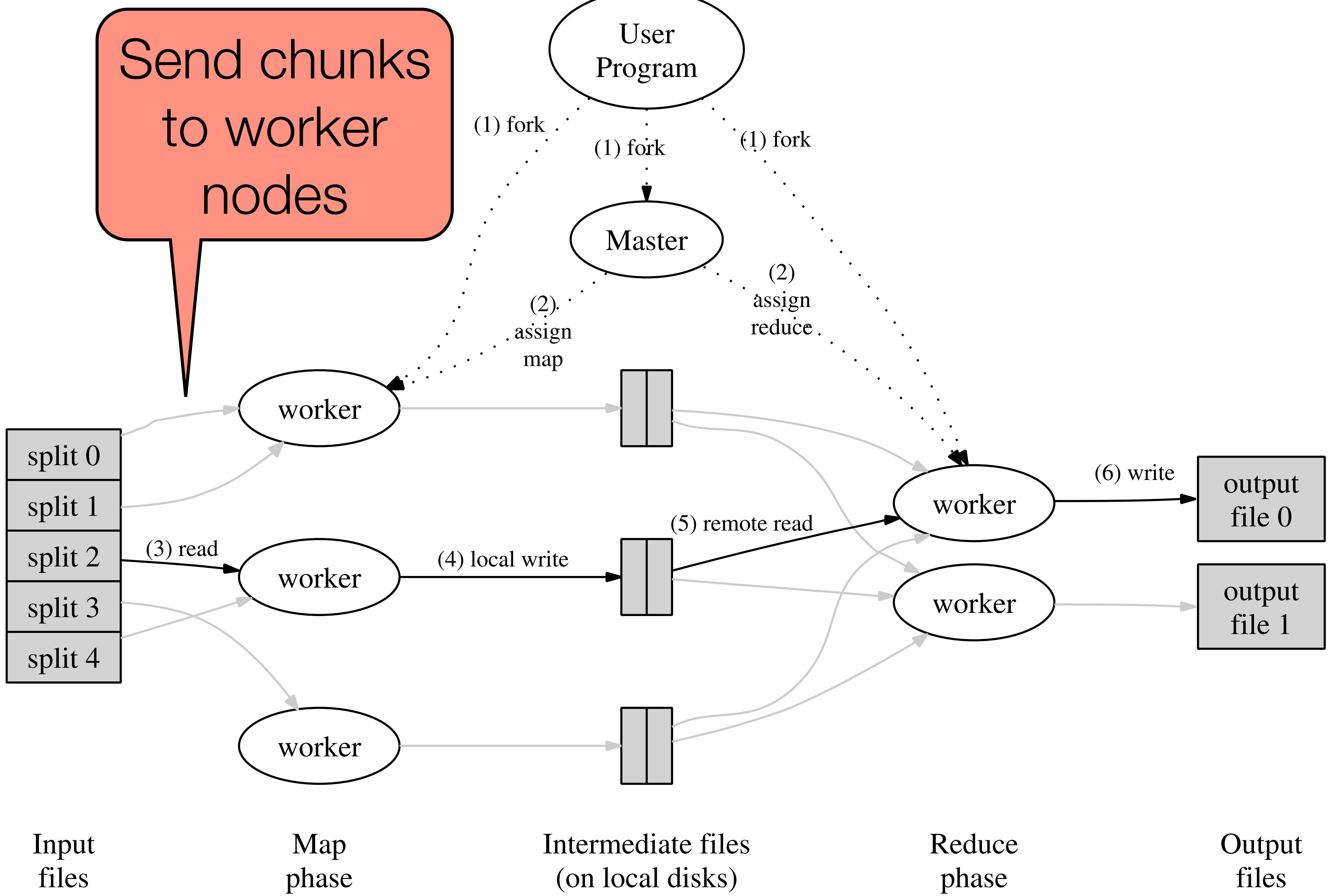
Map-reduce: distribute I/O over many nodes



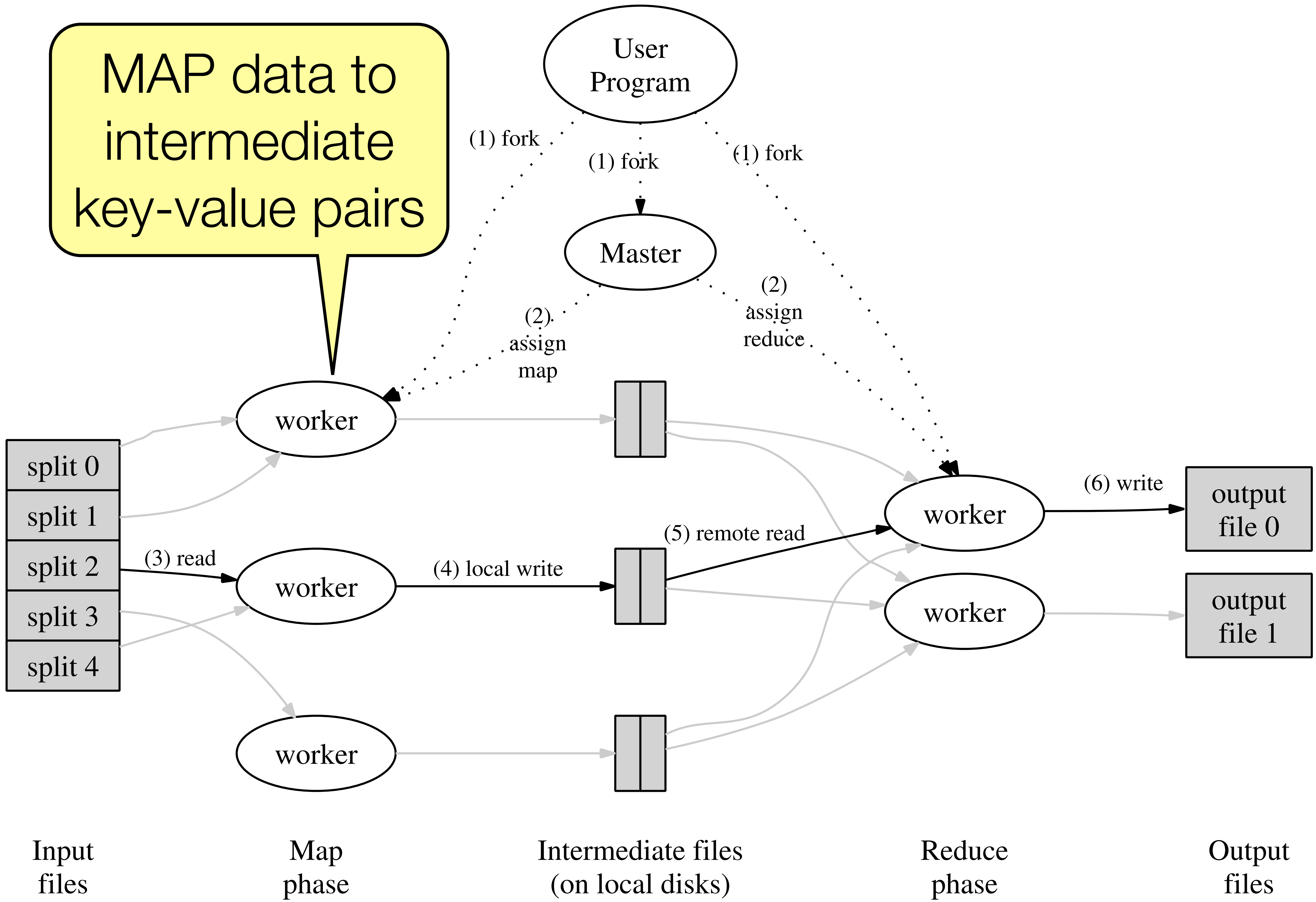
Map-reduce: distribute I/O over many nodes



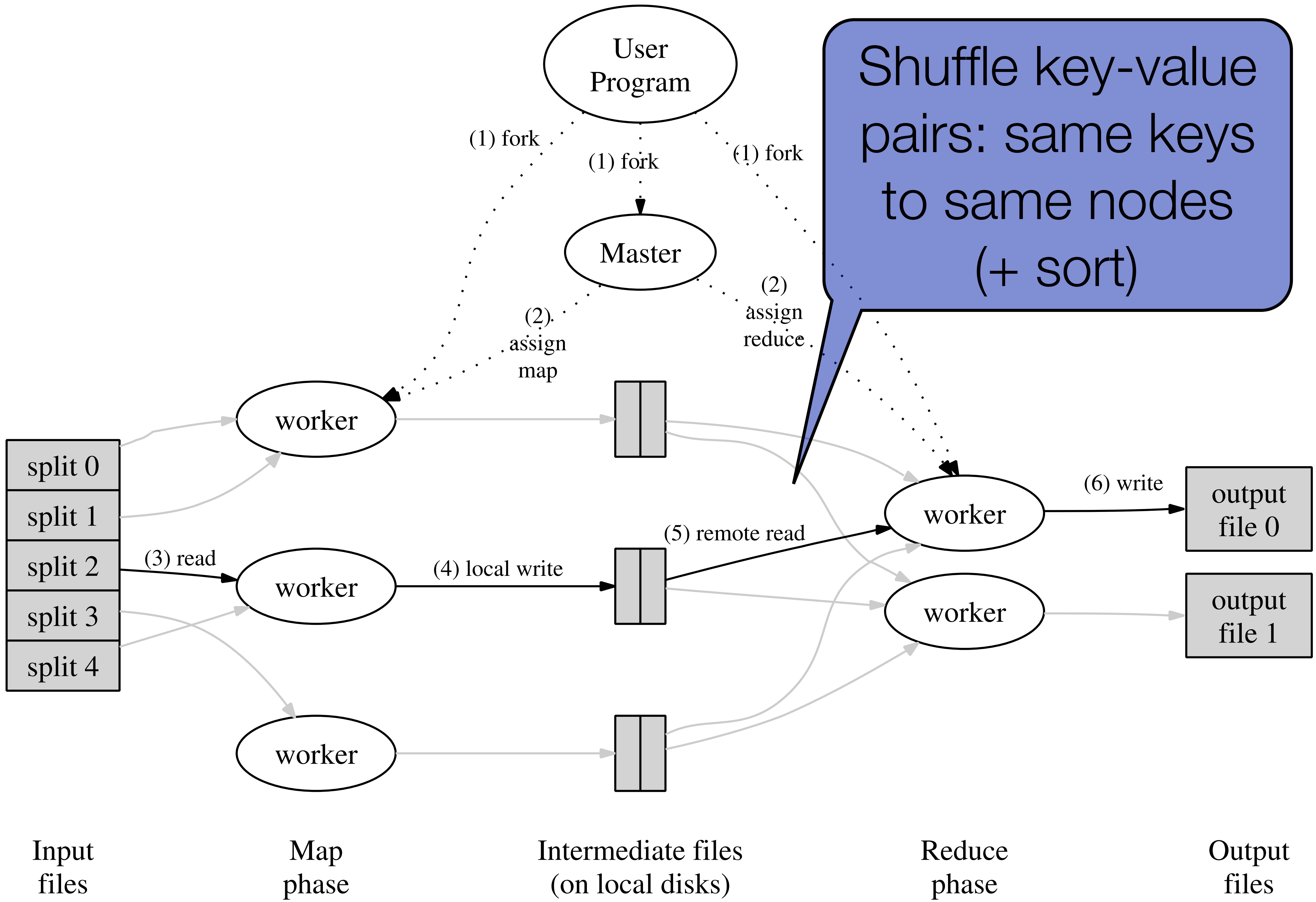
Map-reduce: distribute I/O over many nodes



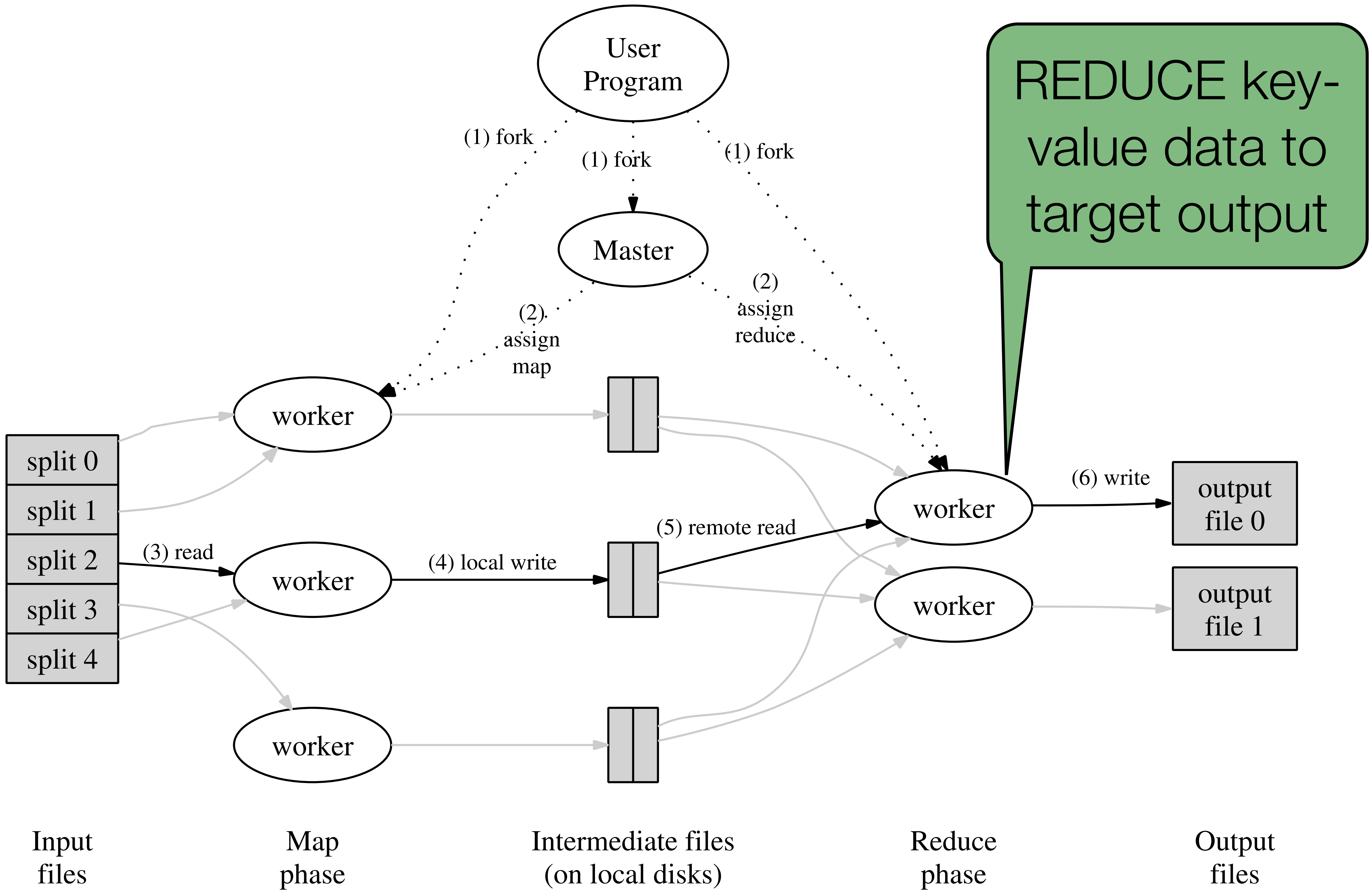
Map-reduce: distribute I/O over many nodes



Map-reduce: distribute I/O over many nodes



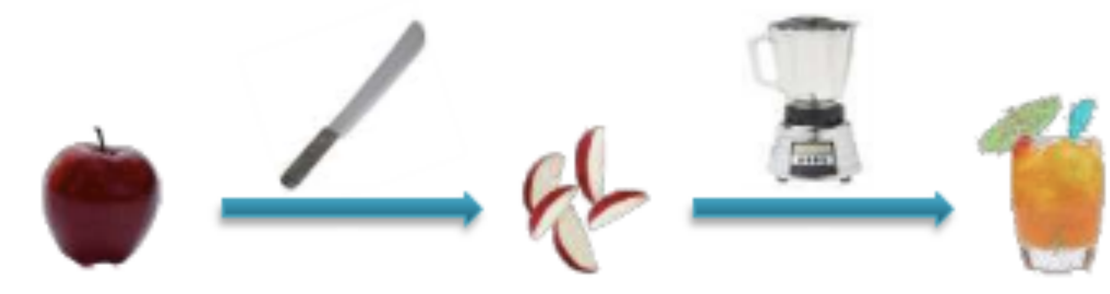
Map-reduce: distribute I/O over many nodes



Thanks to Saliya Ekanayake

Map-reduce: Intuition

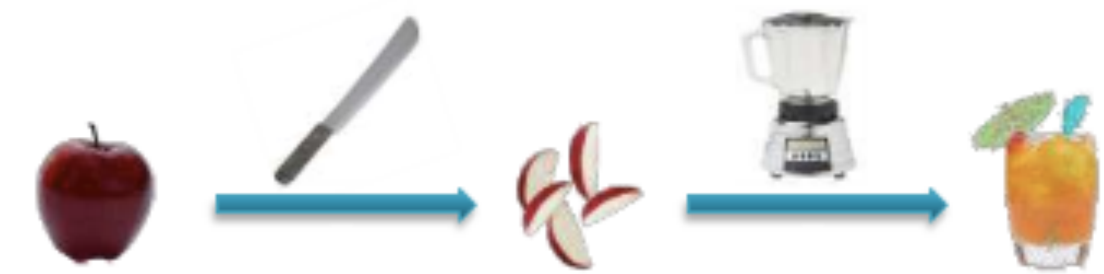
One apple



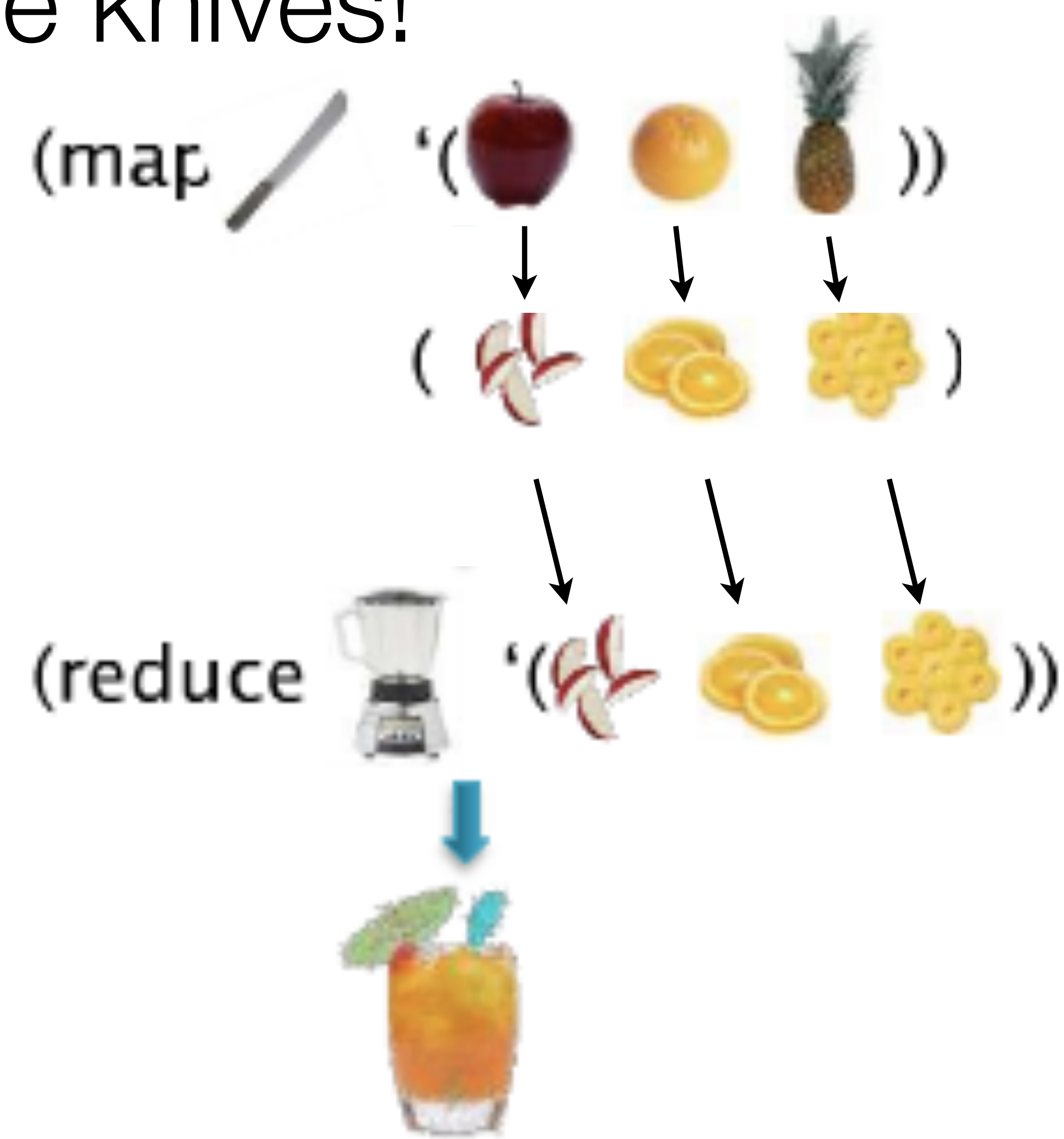
Thanks to Saliya Ekanayake

Map-reduce: Intuition

One apple



A fruit basket: more knives!

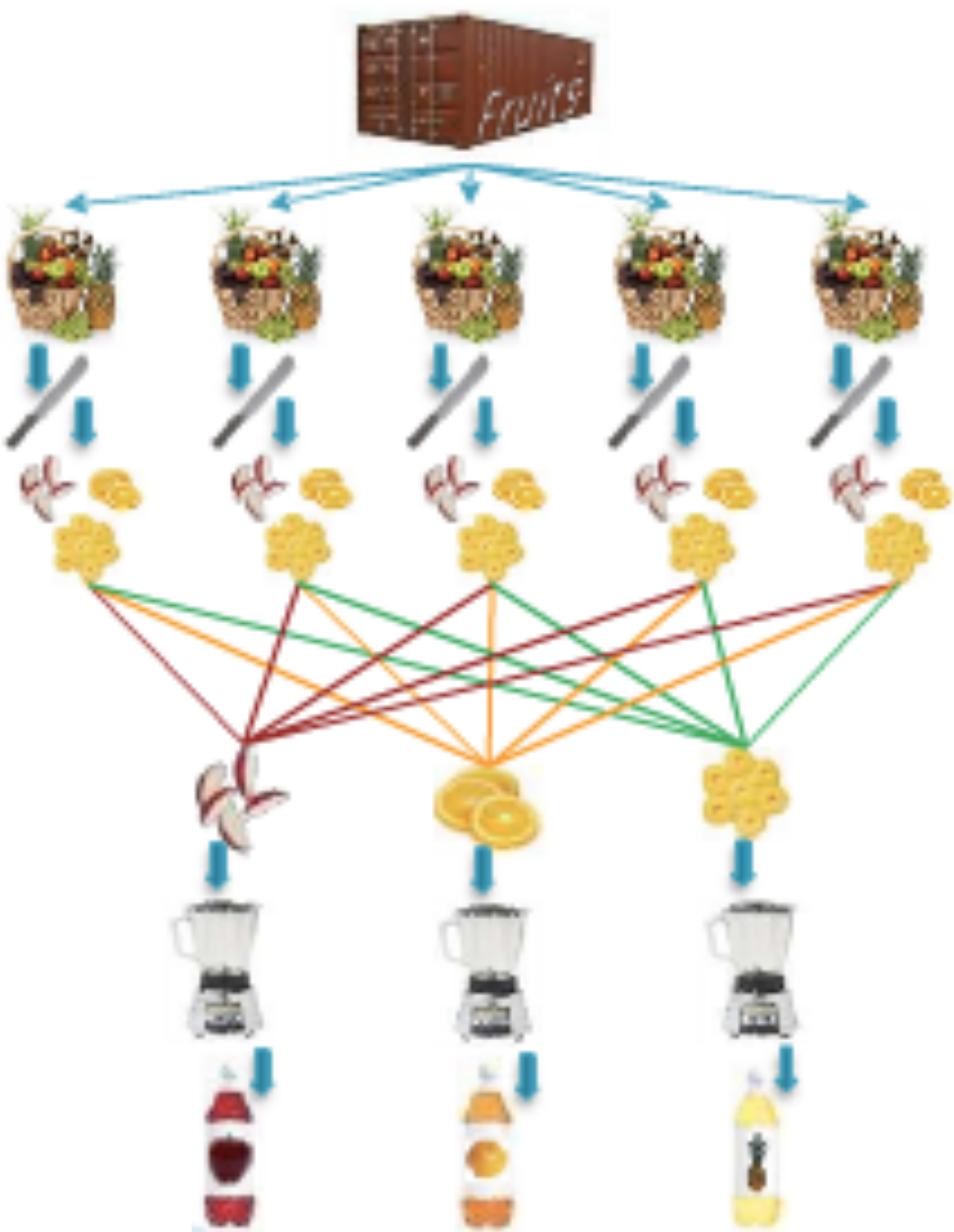


many mappers
(people with knifes)

1 reducer
(person with blender)

Map-reduce: Intuition

A fruit container: more blenders too!

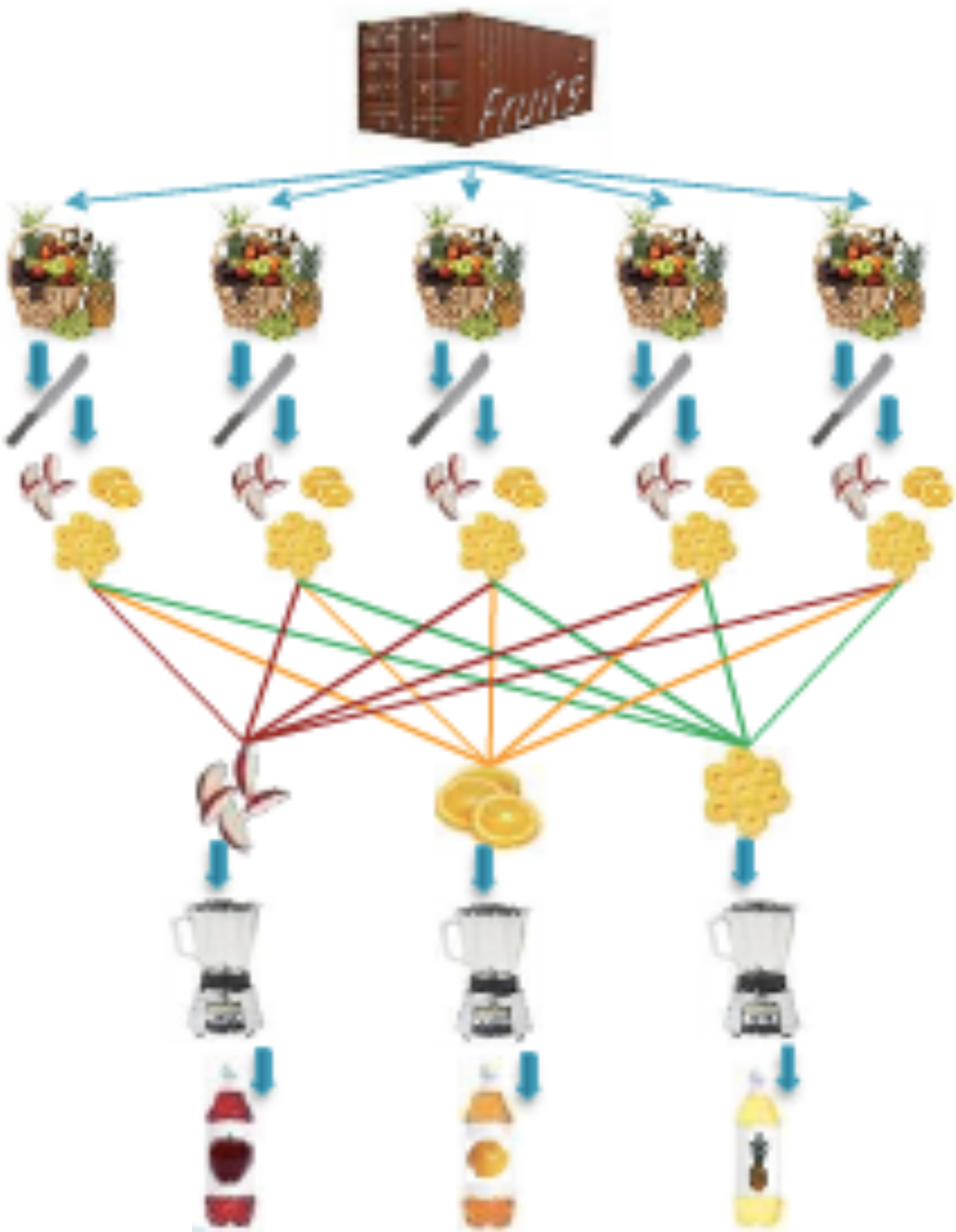


Map input:

<a, > <o, > <p,  > , ...

Map-reduce: Intuition

A fruit container: more blenders too!



Map input:

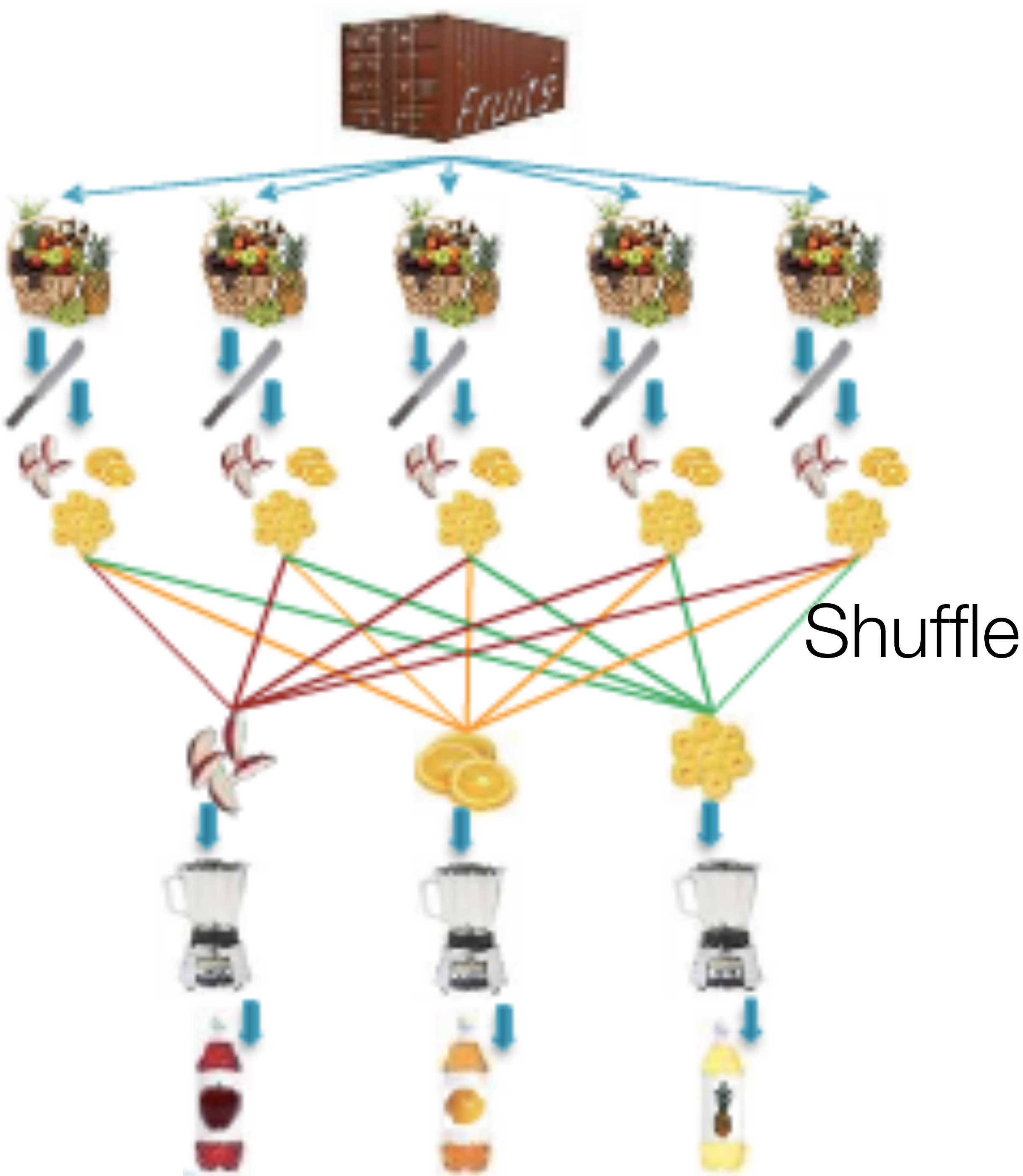
<a, > <o, > <p, > , ...

Map output:

<a’, ><o’, ><p’, >, ...

Map-reduce: Intuition

A fruit container: more blenders too!



Map input:

<a,  > <o,  > <p,  > , ...

Map output:

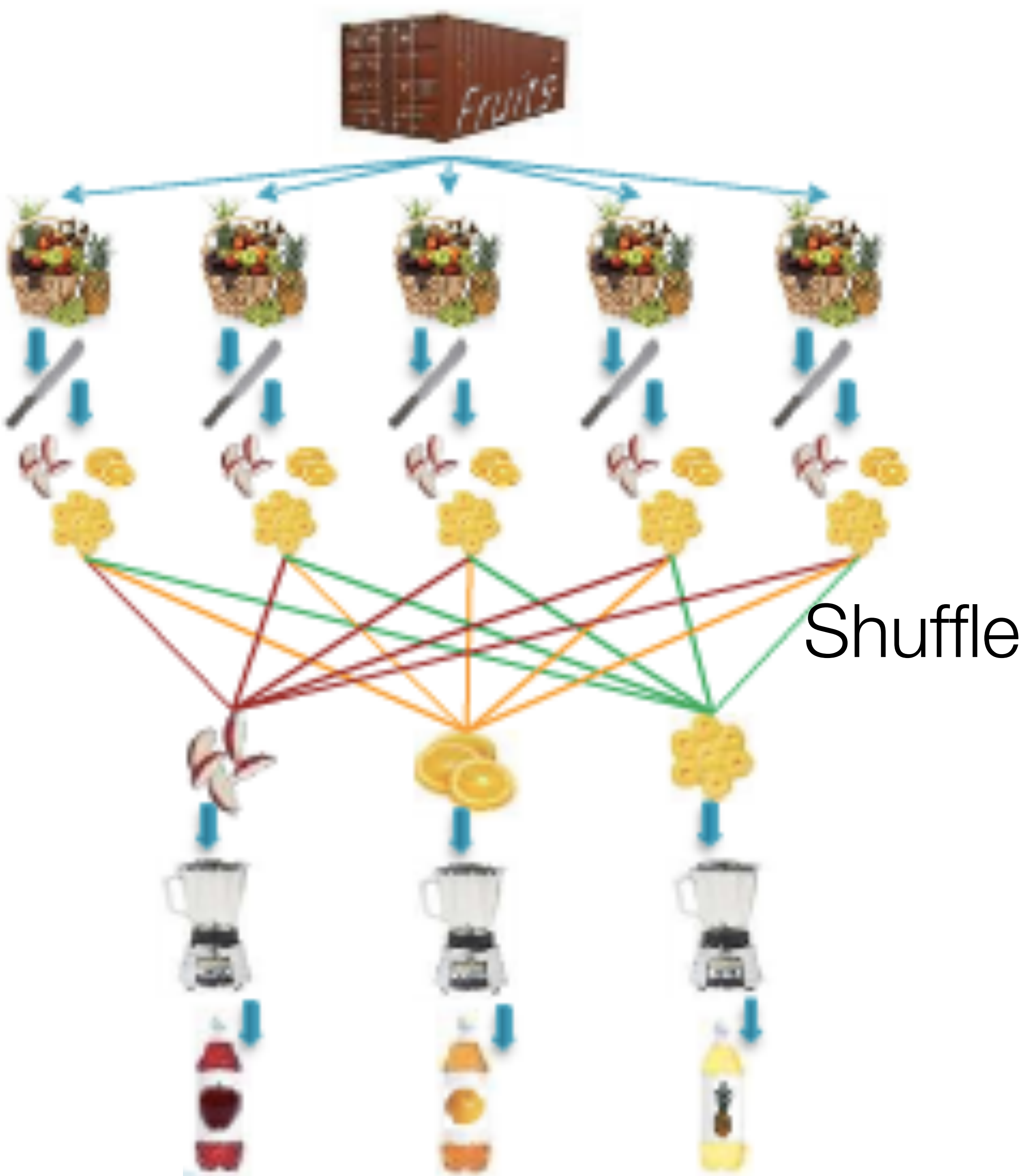
<a’,  ><o’,  ><p’,  >, ...

Reduce input:

<a’, , , , ,  >

Map-reduce: Intuition

A fruit container: more blenders too!



Map input:

<a, > <o, > <p, > , ...

Map output:

<a', ><o', ><p', >, ...

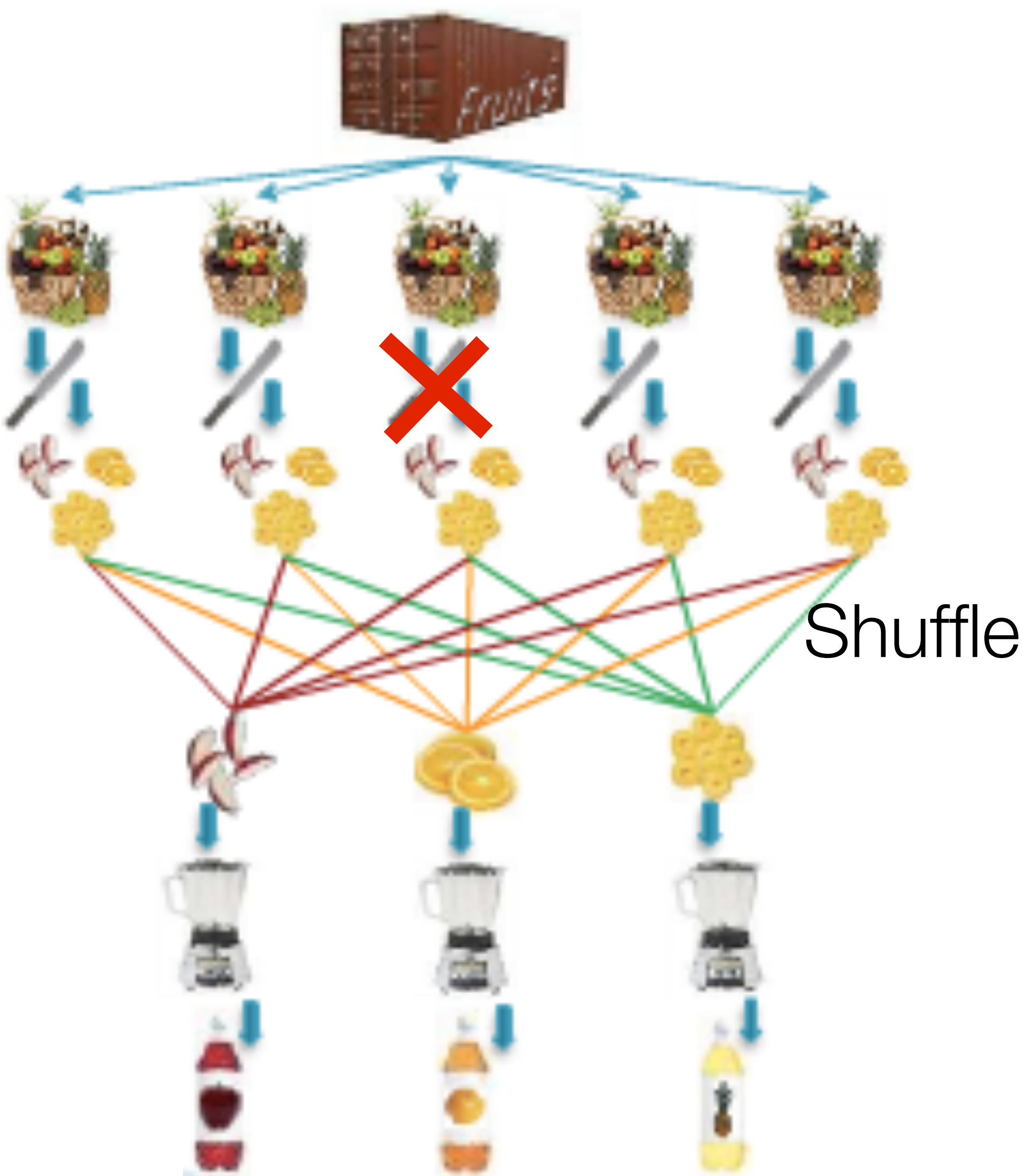
Reduce input:

<a', , , , , >

Reduce output:

Map-reduce: Intuition

A fruit container: more blenders too!



Map input:

<a, > <o, > <p, > , ...

Map output:

<a’, ><o’, ><p’, >, ...

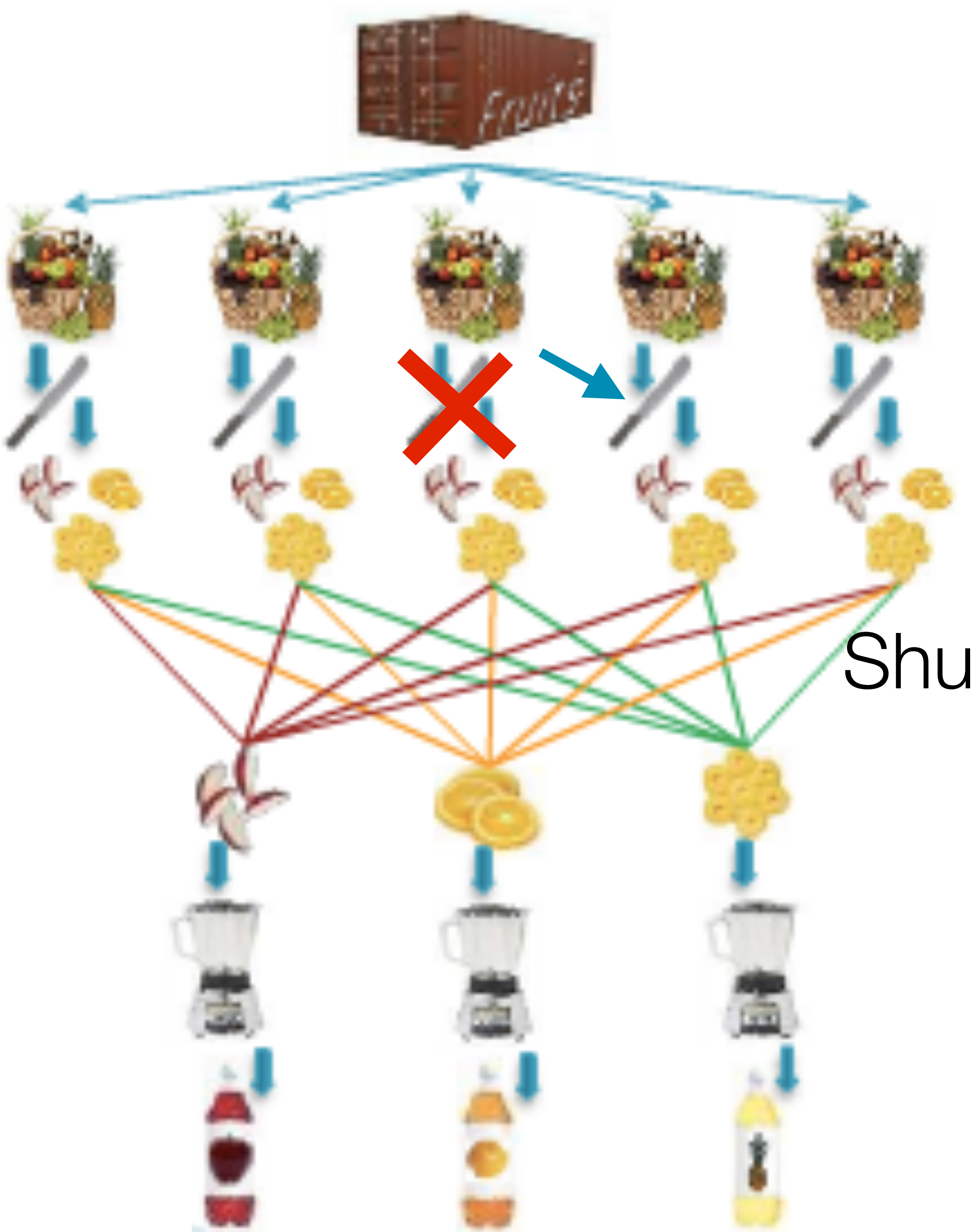
Reduce input:

<a’, , , , , >

Reduce output:

Map-reduce: Intuition

A fruit container: more blenders too!



Map input:

<a, > <o, > <p, > , ...

Map output:

<a’, ><o’, ><p’, >, ...

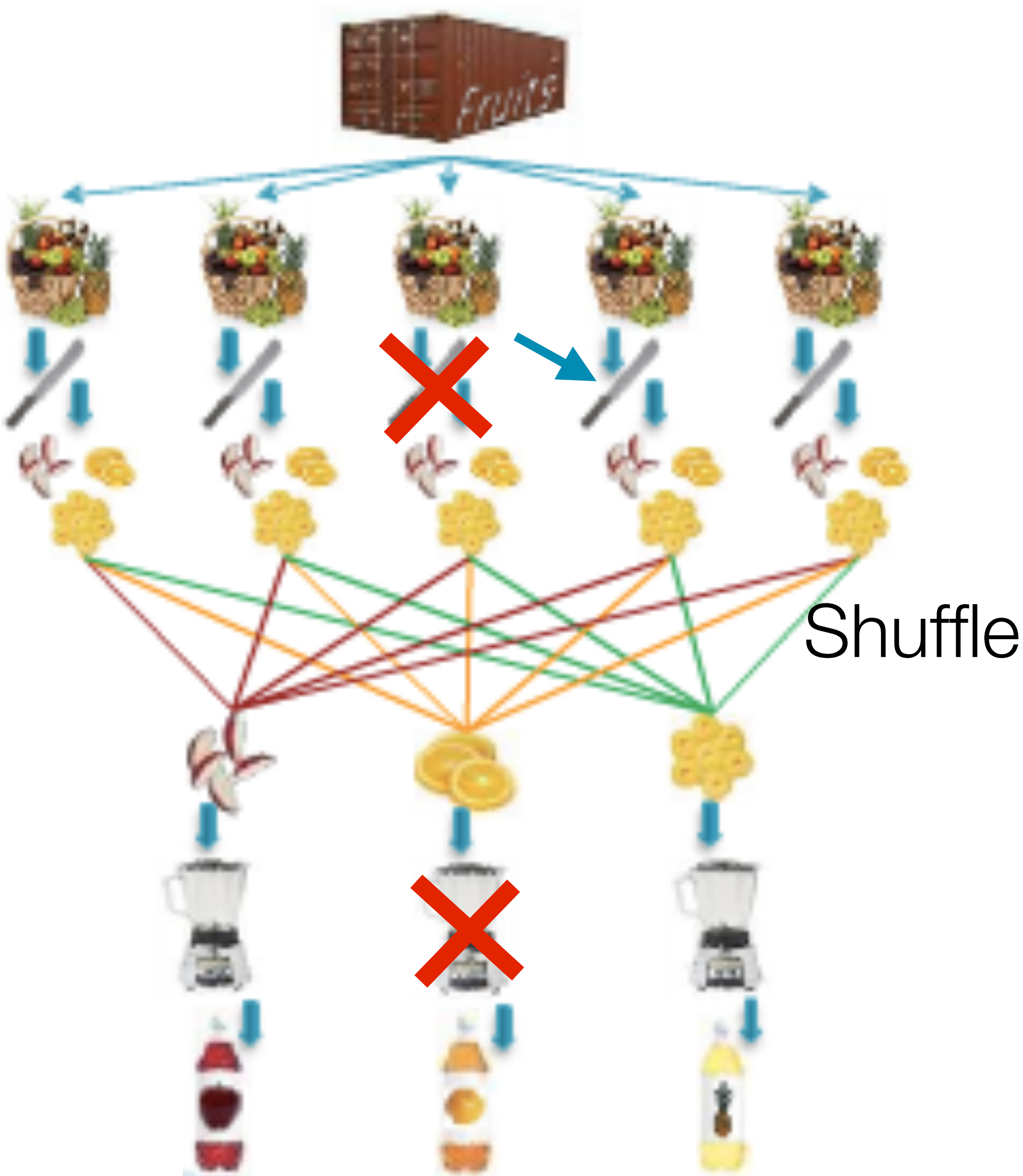
Reduce input:

<a’, , , , , >

Reduce output:

Map-reduce: Intuition

A fruit container: more blenders too!



Map input:

<a, > <o, > <p, > , ...

Map output:

<a’, ><o’, ><p’, >, ...

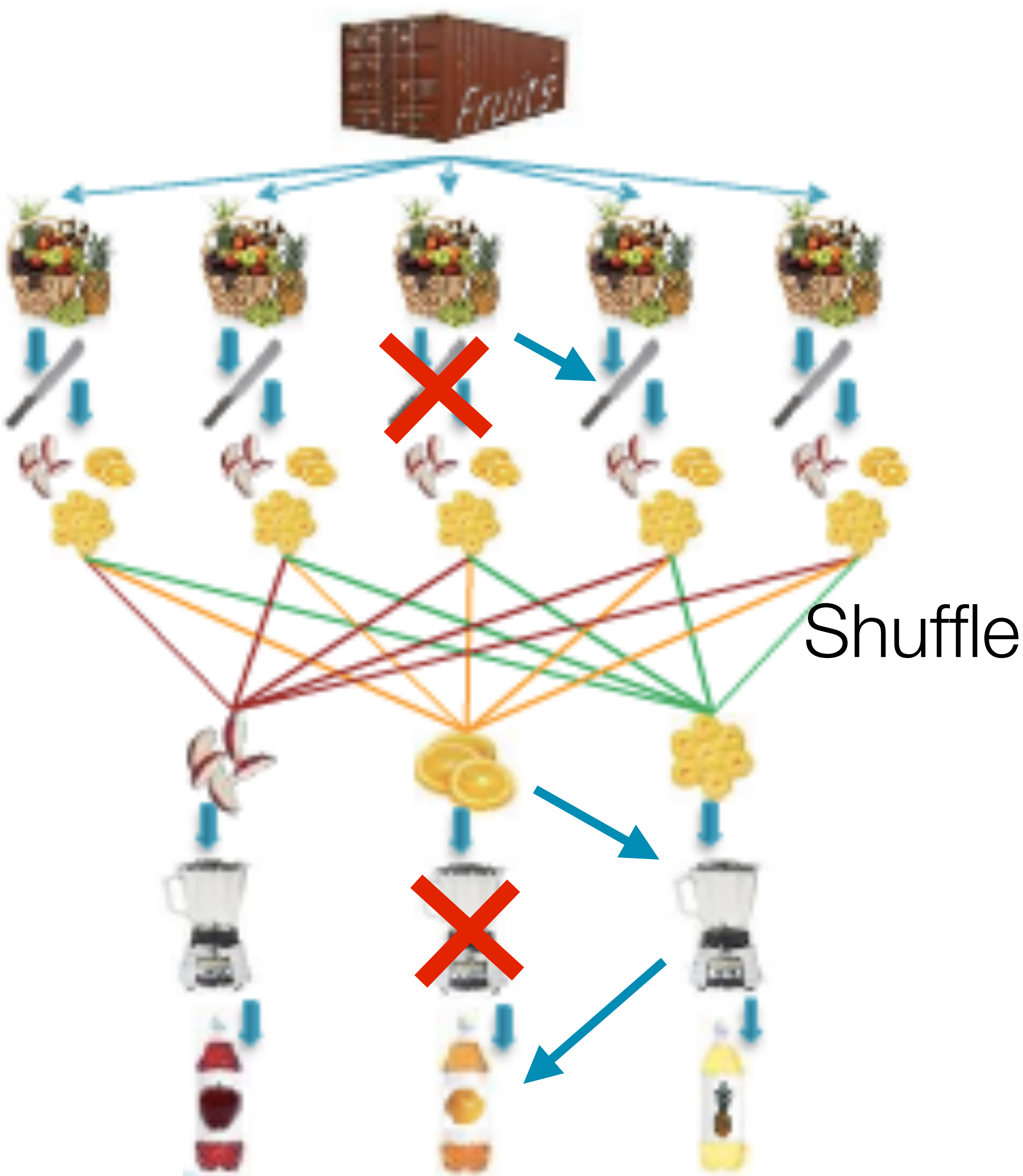
Reduce input:

<a’, , , , , >

Reduce output:

Map-reduce: Intuition

A fruit container: more blenders too!



Map input:

<a, > <o, > <p, > , ...

Map output:

<a', ><o', ><p', >, ...

Reduce input:

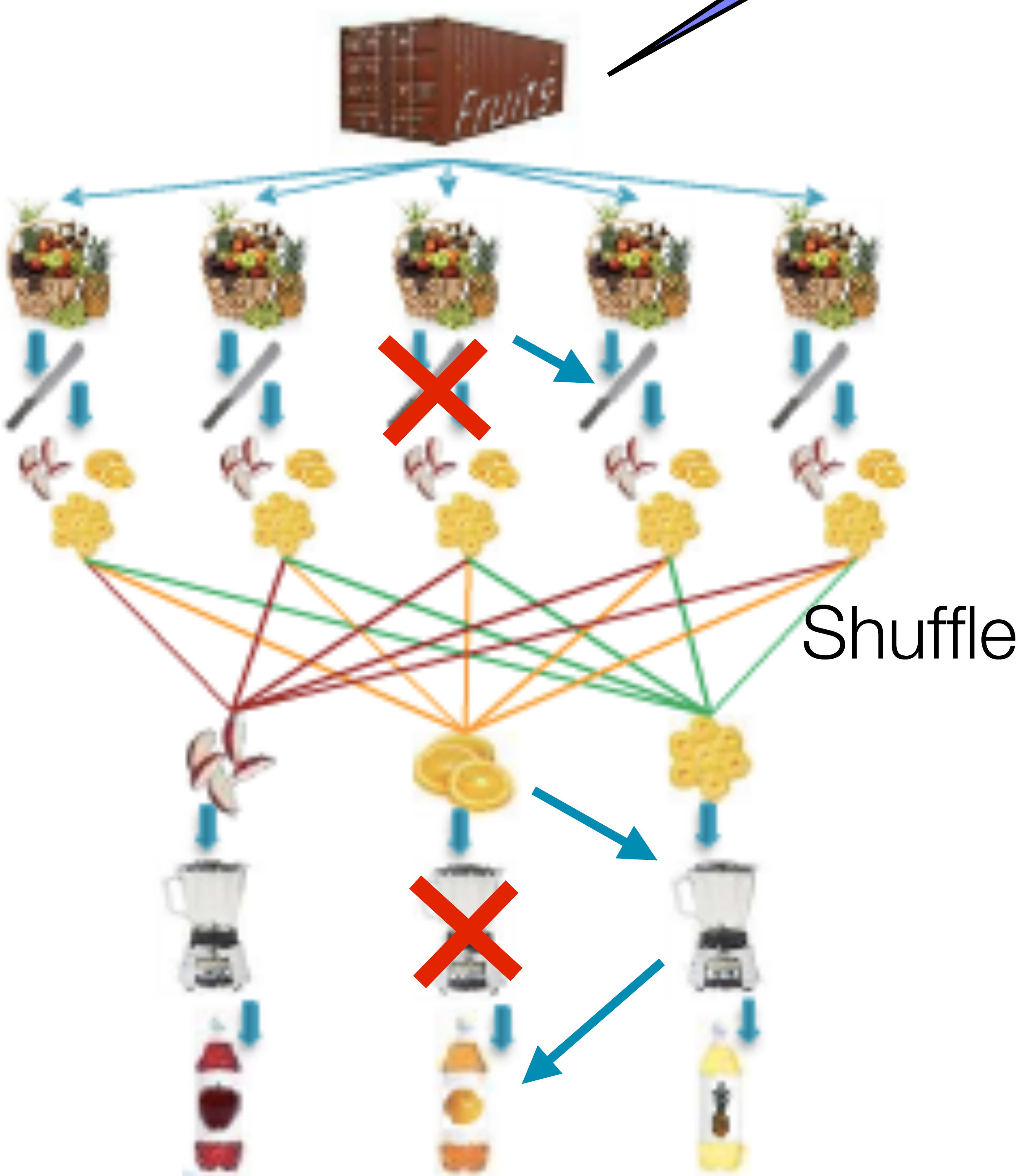
<a', , , , , >

Reduce output: 

Map-reduce: Intuition

an academic analogy:
papers, reviewers, editors

A fruit container: more blenders too!



Map input:

<a, > <o, > <p, > , ...

Map output:

<a’, ><o’, ><p’, >, ...

Reduce input:

<a’, , , , , >

Reduce output:

Example: inverted indexing (e.g. webpages)

Example: inverted indexing (e.g. webpages)

- Input: documents/webpages
  - Doc 1: “Why did the chicken cross the road?”
  - Doc 2: “The chicken and egg problem”
  - Doc 3: “Kentucky Fried Chicken”

- Output
  - Index of words:
    - The word “the” occurs twice in Doc 1 (positions 3 and 6), once in Doc 2 (position 1)

Example: inverted indexing (e.g. webpages)

Map phase (3 parallel tasks)

- map_1 => *("why",(doc_1,1)), ("did",(doc_1,2)), ("the",(doc_1,3)), ("chicken",(doc_1,4)), ("cross",(doc_1,5)), ("the",(doc_1,6)), ("road",(doc_1,7))*
- map_2 => *("the",(doc_2,1)), ("chicken",(doc_2,2)), ("and",(doc_2,3)), ("egg",(doc_2,4)), ("problem", (doc_2,5))*
- map_3 => *("kentucky",(doc_3,1)), ("fried",(doc_3,2)), ("chicken",(doc_3,3))*

Example: inverted indexing (e.g. webpages)

Map phase (3 parallel tasks)

- map_1 => *("why",(doc_1,1)), ("did",(doc_1,2)), ("the",(doc_1,3)), ("chicken",(doc_1,4)), ("cross",(doc_1,5)), ("the",(doc_1,6)), ("road",(doc_1,7))*
- map_2 => *("the",(doc_2,1)), ("chicken",(doc_2,2)), ("and",(doc_2,3)), ("egg",(doc_2,4)), ("problem", (doc_2,5))*
- map_3 => *("kentucky",(doc_3,1)), ("fried",(doc_3,2)), ("chicken",(doc_3,3))*

Intermediate shuffle & sort phase

- *("why", <(doc_1,1)>),*
- *("did", <(doc_1,2)>),*
- *("the", <(doc_1,3), (doc_1,6), (doc_2,1)>)*
- *("chicken", <(doc_1,4), (doc_2,2), (doc_3,3)>)*
- *("cross", <(doc_1,5)>)*
- *("road", <(doc_1,7)>)*
- *("and", <(doc_2,3)>)*
- *("egg", <(doc_2,4)>)*
- *("problem", <(doc_2,5)>)*

Example: inverted indexing (e.g. webpages)

Intermediate shuffle & sort phase

- *("why", <(doc_1,1)>),*
- *("did", <(doc_1,2)>),*
- *("the", <(doc_1,3), (doc_1,6), (doc_2,1)>)*
- *("chicken", <(doc_1,4), (doc_2,2), (doc_3,3)>)*
- *("cross", <(doc_1,5)>)*
- *("road", <(doc_1,7)>)*
- *("and", <(doc_2,3)>)*
- *("egg", <(doc_2,4)>)*
- *("problem", <(doc_2,5)>)*
- *("kentucky", <(doc_3,1)>)*
- *("fried", <(doc_3,2)>)*

Example: inverted indexing (e.g. webpages)

Intermediate shuffle & sort phase

- *("why", <(doc$_1$,1)>),*
- *("did", <(doc$_1$,2)>),*
- *("the", <(doc$_1$,3), (doc$_1$,6), (doc$_2$,1)>)*
- *("chicken", <(doc$_1$,4), (doc$_2$,2), (doc$_3$,3)>)*
- *("cross", <(doc$_1$,5)>)*
- *("road", <(doc$_1$,7)>)*
- *("and", <(doc$_2$,3)>)*
- *("egg", <(doc$_2$,4)>)*
- *("problem", <(doc$_2$,5)>)*
- *("kentucky", <(doc$_3$,1)>)*
- *("fried", <(doc$_3$,2)>)*

Reduce phase (11 parallel tasks)

- *("why", <(doc$_1$,<1>)>),*
- *("did", <(doc$_1$,<2>)>),*
- *("the", <(doc$_1$, <3,6>), (doc$_2$, <1>)>)*
- *("chicken", <(doc$_1$,<4>), (doc$_2$,<2>), (doc$_3$,<3>)>)*
- *("cross", <(doc$_1$,<5>)>)*
- *("road", <(doc$_1$,<7>)>)*
- *("and", <(doc$_2$,<3>)>)*
- *("egg", <(doc$_2$,<4>)>)*
- *("problem", <(doc$_2$,<5>)>)*
- *("kentucky", <(doc$_3$,<1>)>)*
- *("fried", <(doc$_3$,<2>)>)*

Example: inverted indexing (e.g. webpages)

Intermediate shuffle & sort phase

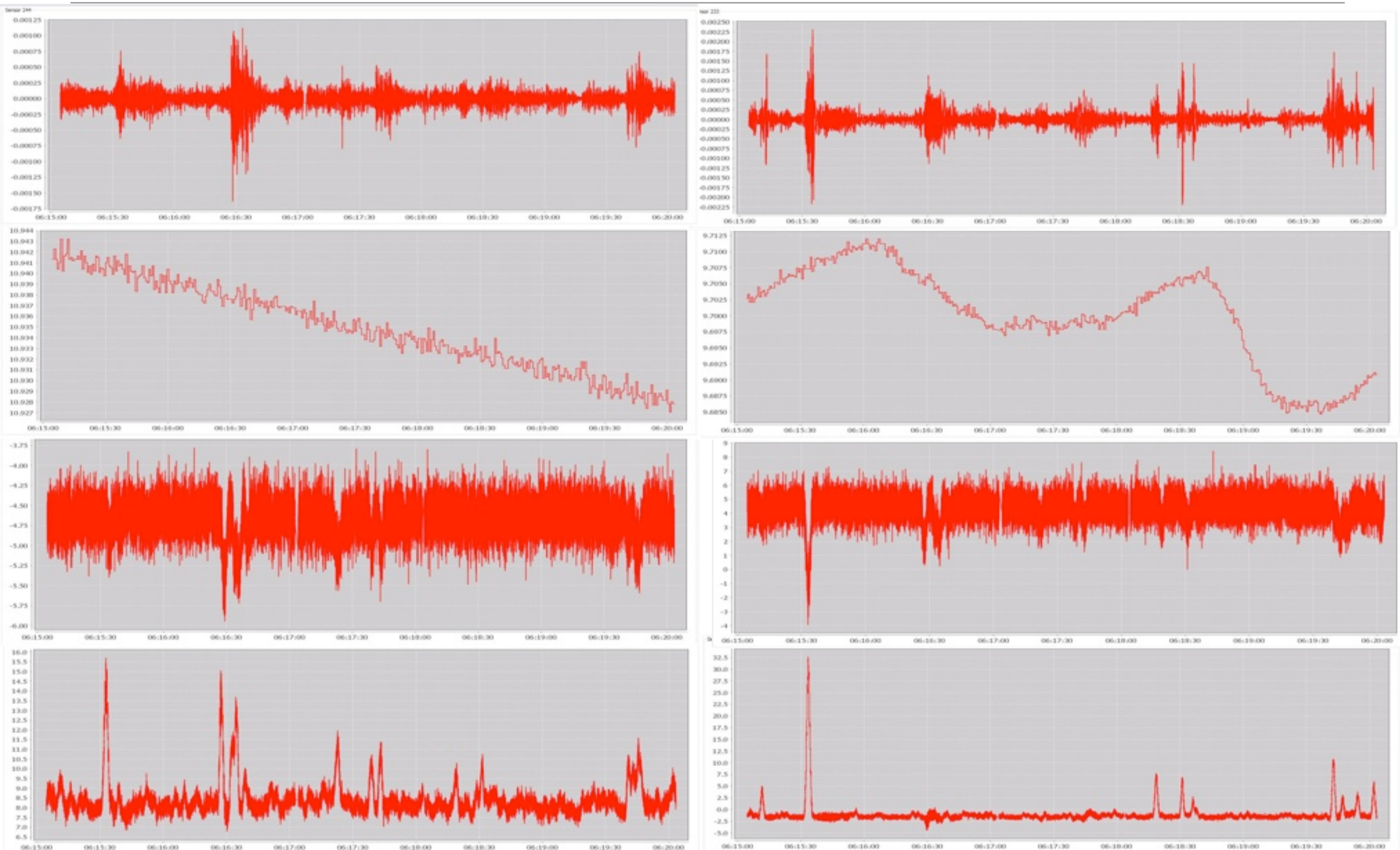
- *("why", <(doc_1,1)>),*
- *("did", <(doc_1,2)>),*
- *("the", <(doc_1,3), (doc_1,6), (doc_2,1)>)*
- *("chicken", <(doc_1,4), (doc_2,2), (doc_3,3)>)*
- *("cross", <(doc_1,5)>)*
- *("road", <(doc_1,7)>)*
- *("and", <(doc_2,3)>)*
- *("egg", <(doc_2,4)>)*
- *("problem", <(doc_2,5)>)*
- *("kentucky", <(doc_3,1)>)*
- *("fried", <(doc_3,2)>)*

Reduce phase (11 parallel tasks)

- *("why", <(doc_1,<1>)>),*
- *("did", <(doc_1,<2>)>),*
- *("the", <(doc_1, <3,6>), (doc_2, <1>)>)*
- *("chicken", <(doc_1,<4>), (doc_2,<2>), (doc_3,<3>)>)*
- *("cross", <(doc_1,<5>)>)*
- *("road", <(doc_1,<7>)>)*
- *("and", <(doc_2,<3>)>)*
- *("egg", <(doc_2,<4>)>)*
- *("problem", <(doc_2,<5>)>)*
- *("kentucky", <(doc_3,<1>)>)*
- *("fried", <(doc_3,<2>)>)*

Simply write mapper and reducer method, often few lines of code

Map-reduce on time series data



A simple operation: aggregation

2008-10-24 06:15:04.559, -6.293695, -1.1263204, 2.985364, 43449.957, 2.3577218, 38271.21
2008-10-24 06:15:04.570, -6.16952, -1.3805857, 2.6128333,43449.957, 2.4848552, 37399.26
2008-10-24 06:15:04.580, -5.711255, -0.8897944, 3.139107, 43449.957, 2.1744132, 38281.0

2008-10-24 06:15:04.559 min -524.0103
2008-10-24 06:15:04.559 max 38271.21
2008-10-24 06:15:04.559 avg 447.37795925930226
2008-10-24 06:15:04.570 min -522.7882
2008-10-24 06:15:04.570 max 37399.26
2008-10-24 06:15:04.570 avg 437.1266600847675

A simple operation: aggregation

- Input: Table with sensors in columns and timestamps in rows

```
2008-10-24 06:15:04.559,  -6.293695,  -1.1263204,  2.985364,  43449.957,  2.3577218,  38271.21
2008-10-24 06:15:04.570,  -6.16952,    -1.3805857,  2.6128333,43449.957,  2.4848552,  37399.26
2008-10-24 06:15:04.580,  -5.711255,  -0.8897944,  3.139107,  43449.957,  2.1744132,  38281.0
```

- Desired output: Aggregated measures per timestamp

```
2008-10-24 06:15:04.559 min     -524.0103
2008-10-24 06:15:04.559 max      38271.21
2008-10-24 06:15:04.559 avg     447.37795925930226
2008-10-24 06:15:04.570 min     -522.7882
2008-10-24 06:15:04.570 max      37399.26
2008-10-24 06:15:04.570 avg     437.1266600847675
```

Map-reduce on time series data

```
public void map(LongWritable key, Text value, Context context) {
        String values[] = value.toString().split("\t");
        Text time = new Text(values[0]);
        for(int i = 1; i <= nrStressSensors; i++)
            context.write(time, new Text(values[i]));
}


public void reduce(Text key, Iterable<Text> values, Context context) {
      //init; sum, min, max, count = 0
      Double d;
      for (Text v : values) {
            d = Double.valueOf(v.toString());
        sum += d;
        min = Math.min(min, d);
        max = Math.max(max, d);
        count++;
      }
      context.write(new Text(key+" min"), new Text(Double.toString((min))));
      context.write(new Text(key+" max"), new Text(Double.toString((max))));
      context.write(new Text(key+" avg"), new Text(Double.toString((sum/count))));
}
```

Map-reduce on time series data

```
public void map(LongWritable key, Text value, Context context) {
        String values[] = value.toString().split("\t");
        Text time = new Text(values[0]);
        for(int i = 1; i <= nrStressSensors; i++)
            context.write(time, new Text(values[i]));
}
```

MAP data to <timestamp,value>

```
public void reduce(Text key, Iterable<Text> values, Context context) {
        //init; sum, min, max, count = 0
        Double d;
        for (Text v : values) {
            d = Double.valueOf(v.toString());
          sum += d;
          min = Math.min(min, d);
          max = Math.max(max, d);
          count++;
        }
        context.write(new Text(key+" min"), new Text(Double.toString((min))));
        context.write(new Text(key+" max"), new Text(Double.toString((max))));
        context.write(new Text(key+" avg"), new Text(Double.toString((sum/count))));
}
```

Map-reduce on time series data

```
public void map(LongWritable key, Text value, Context context) {
        String values[] = value.toString().split("\t");
        Text time = new Text(values[0]);
        for(int i = 1; i <= nrStressSensors; i++)
            context.write(time, new Text(values[i]));
}
```

MAP data to <timestamp,value>

Shuffle/sort per timestamp

```
public void reduce(Text key, Iterable<Text> values, Context context) {
        //init; sum, min, max, count = 0
        Double d;
        for (Text v : values) {
                d = Double.valueOf(v.toString());
            sum += d;
            min = Math.min(min, d);
            max = Math.max(max, d);
            count++;
        }
        context.write(new Text(key+" min"), new Text(Double.toString((min))));
        context.write(new Text(key+" max"), new Text(Double.toString((max))));
        context.write(new Text(key+" avg"), new Text(Double.toString((sum/count))));
}
```

Map-reduce on time series data

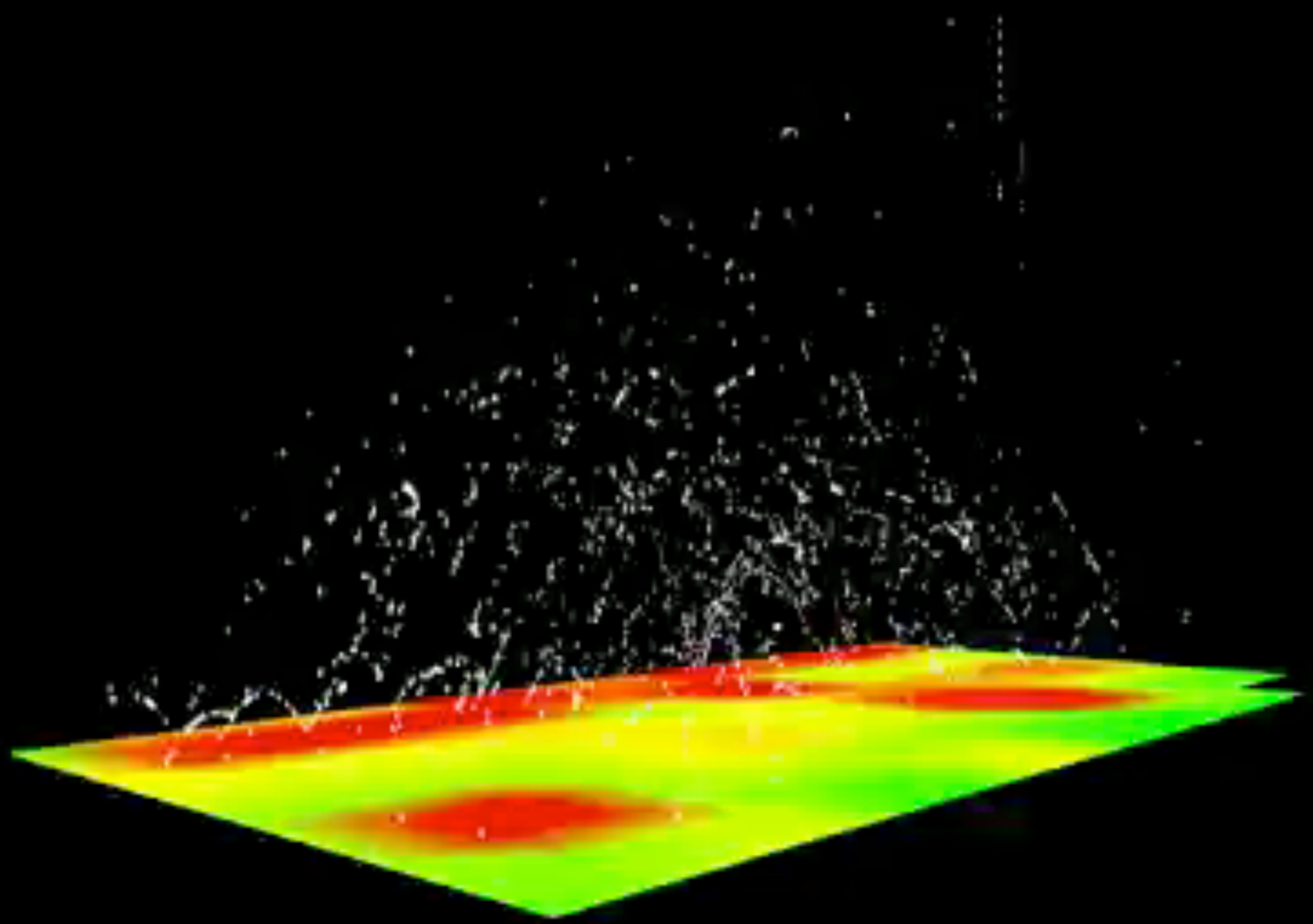
```java
public void map(LongWritable key, Text value, Context context) {
            String values[] = value.toString().split("\t");
            Text time = new Text(values[0]);
            for(int i = 1; i <= nrStressSensors; i++)
                context.write(time, new Text(values[i]));
}


public void reduce(Text key, Iterable<Text> values, Context context) {
        //init; sum, min, max, count = 0
        Double d;
        for (Text v : values) {
                d = Double.valueOf(v.toString());
            sum += d;
            min = Math.min(min, d);
            max = Math.max(max, d);
            count++;
        }
        context.write(new Text(key+" min"), new Text(Double.toString((min))));
        context.write(new Text(key+" max"), new Text(Double.toString((max))));
        context.write(new Text(key+" avg"), new Text(Double.toString((sum/count))));
}
```

MAP data to <timestamp,value>

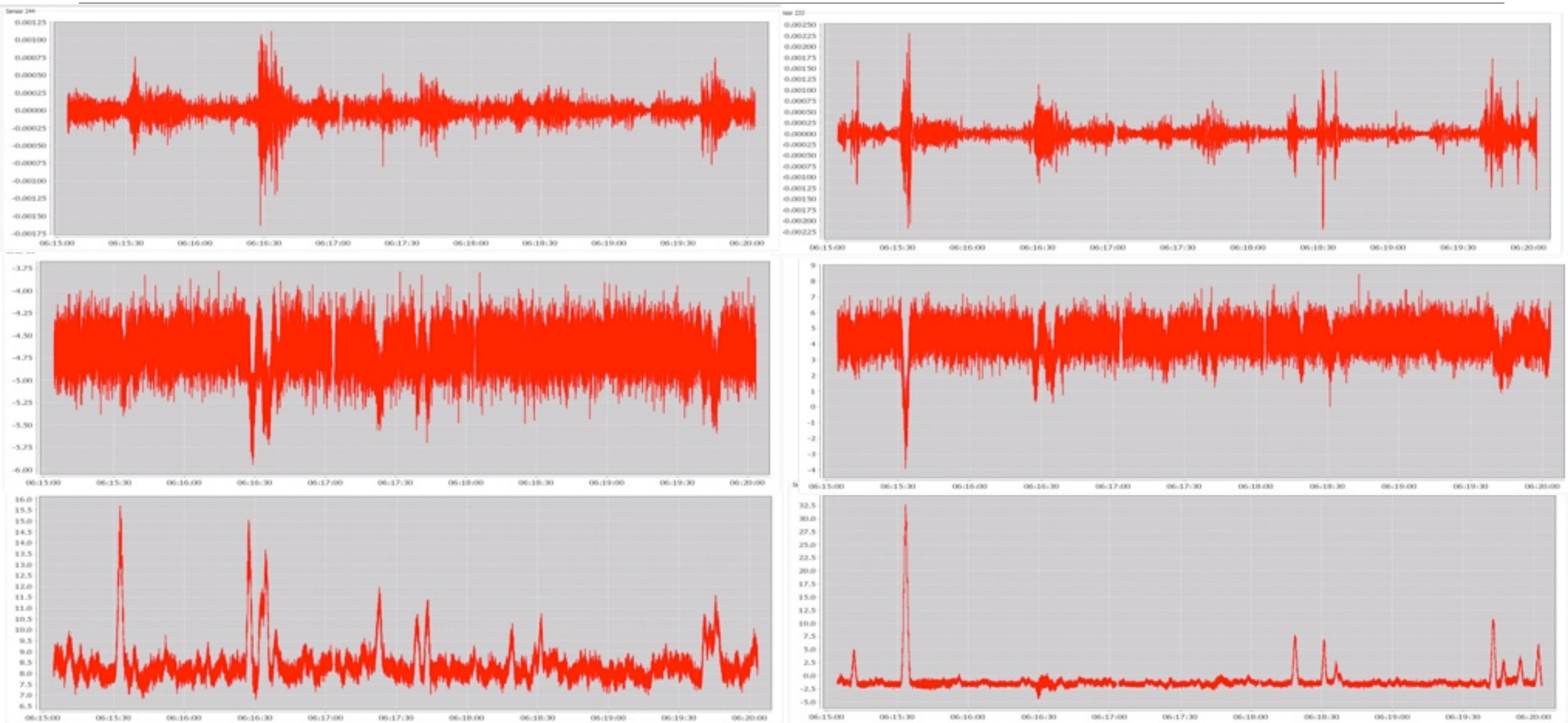
Shuffle/sort per timestamp

REDUCE data per timestamp to aggregates

Shuffling

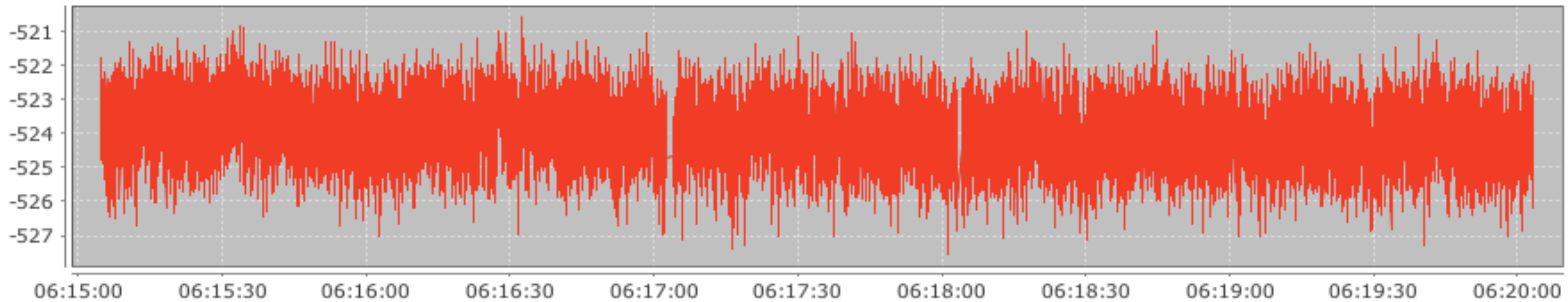


Map-reduce on time series data

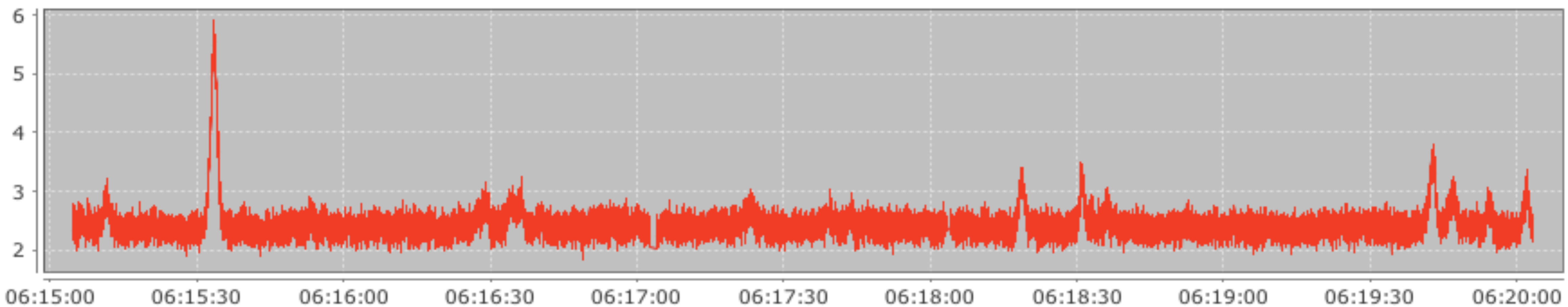


Map-reduce on time series data

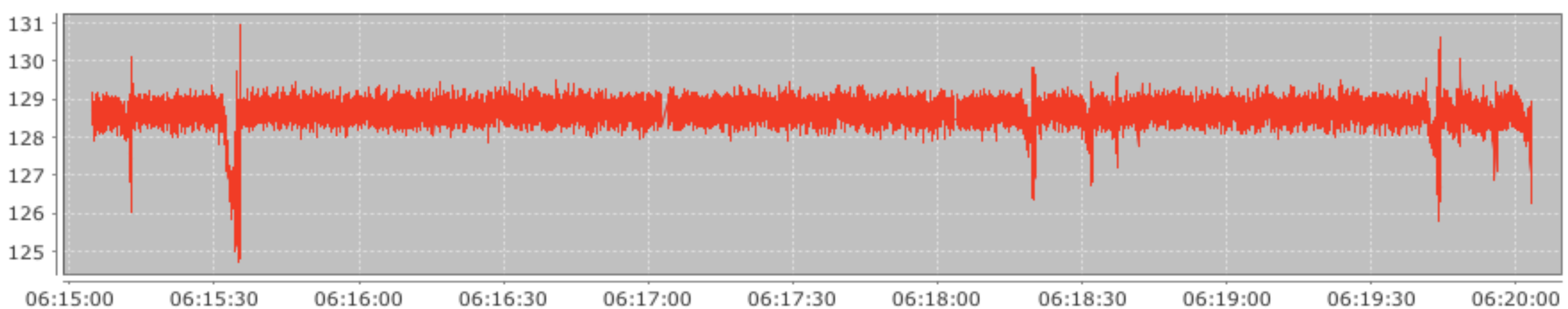
Min



Avg



Max

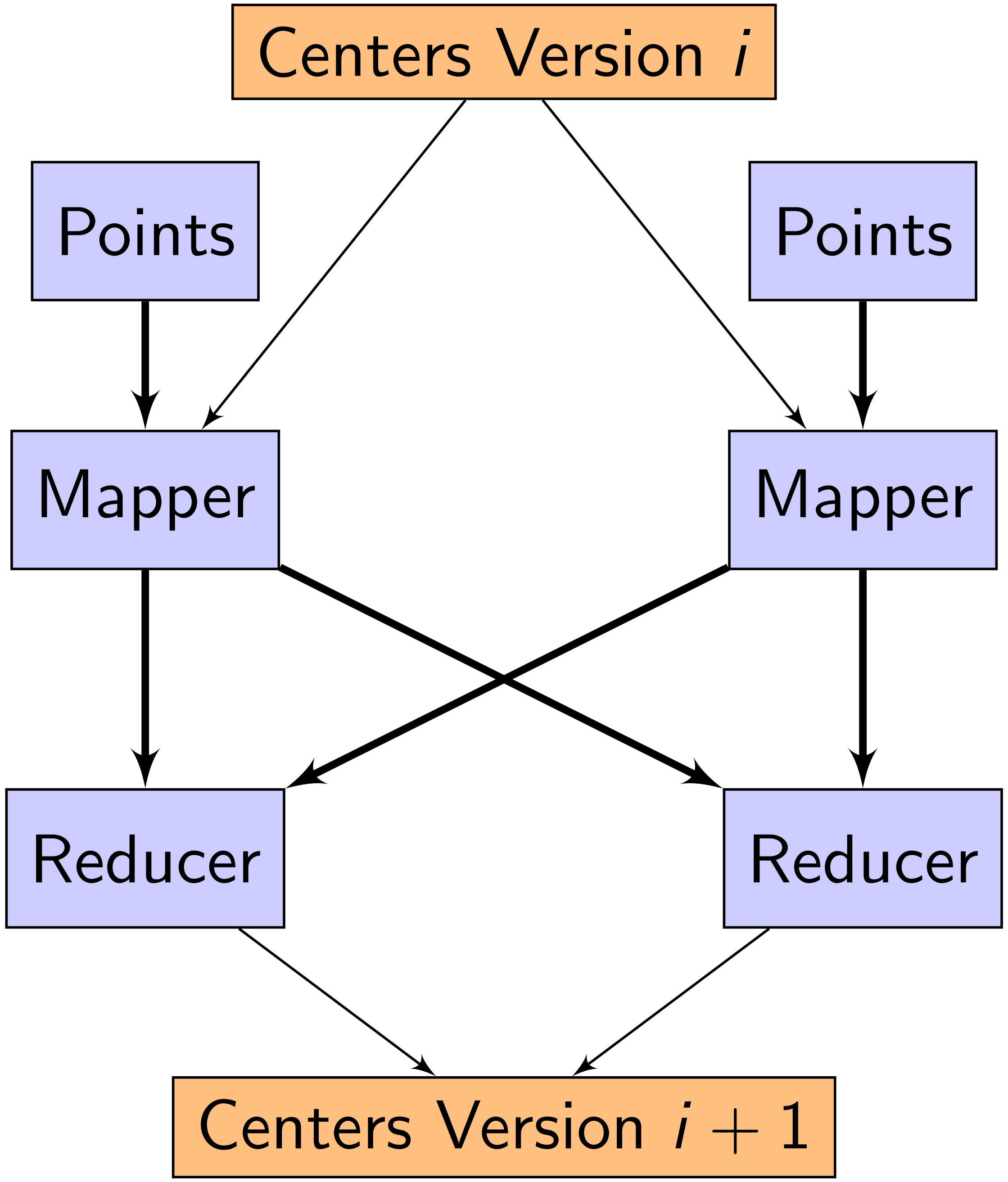


ML with map-reduce

- What part is I/O intensive (related to data points), and can be parallelized?
  - E.g. k-Means?

ML with map-reduce

- What part is I/O intensive (related to data points), and can be parallelized?
  - E.g. k-Means?
- Calculation of distance function!
  - Split data in chunks
  - Choose initial centers
  - MAP: calculate distances to all centers: <point,[distances]>
  - REDUCE: calculate new centers
  - Repeat
- Others: SVM, NB, NN, LR,...
  - *Chu et al. Map-Reduce for ML on Multicore, NIPS '06*



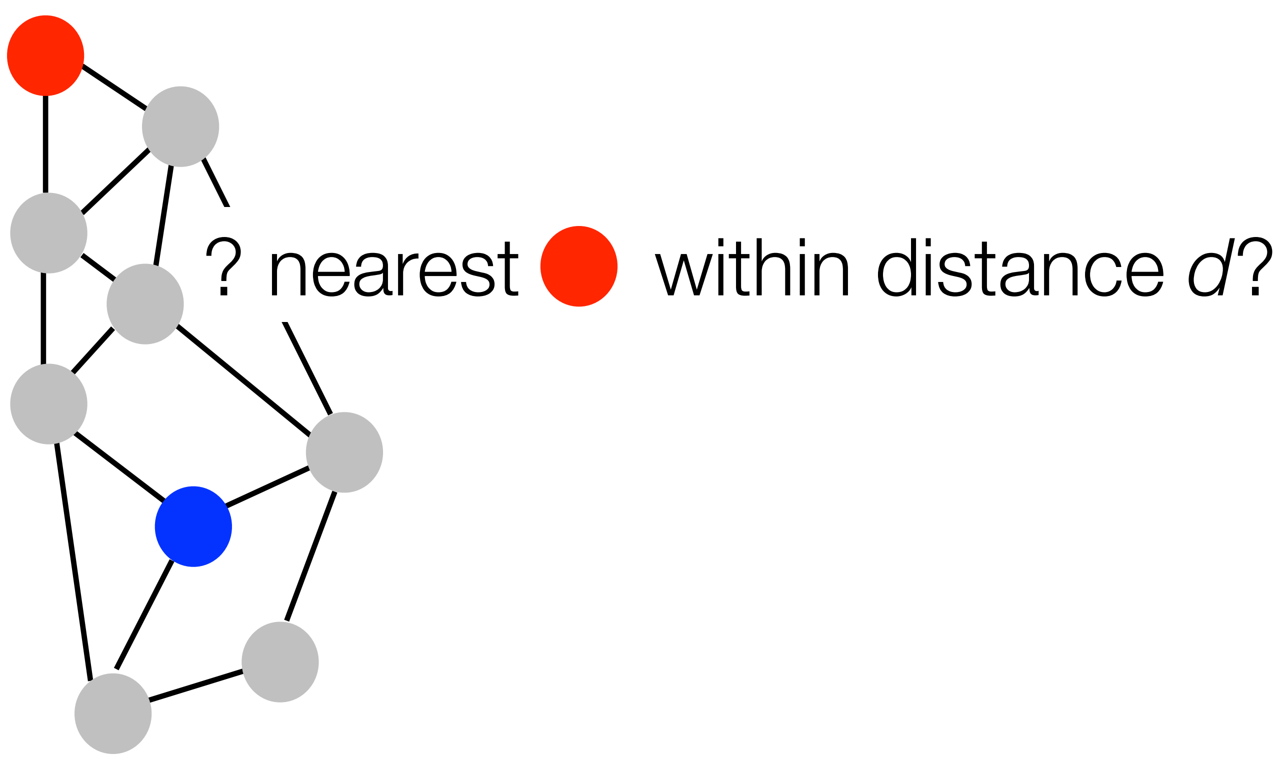
Map-reduce on graph data

- Find nearest feature on a graph

Input

graph

(node,label)



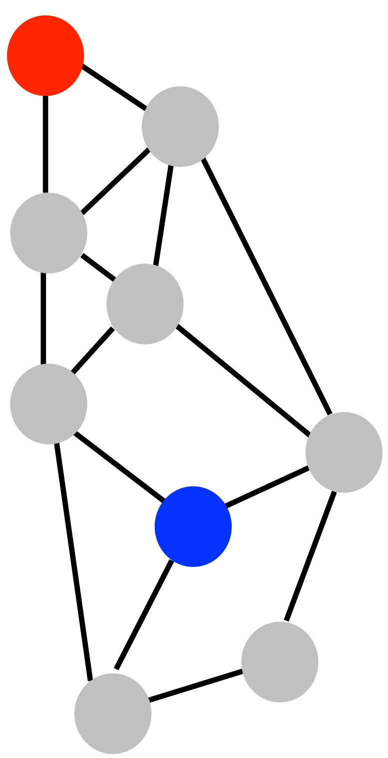
Map-reduce on graph data

- Find nearest feature on a graph

Input

graph

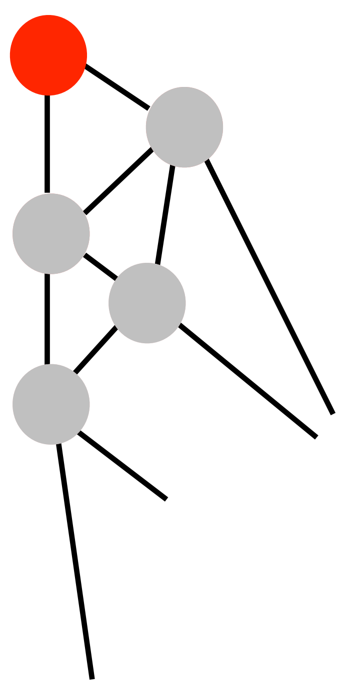
(node,label)



Map

∀●, search graph with radius *d*

< ● ,{● ,*distance*} >



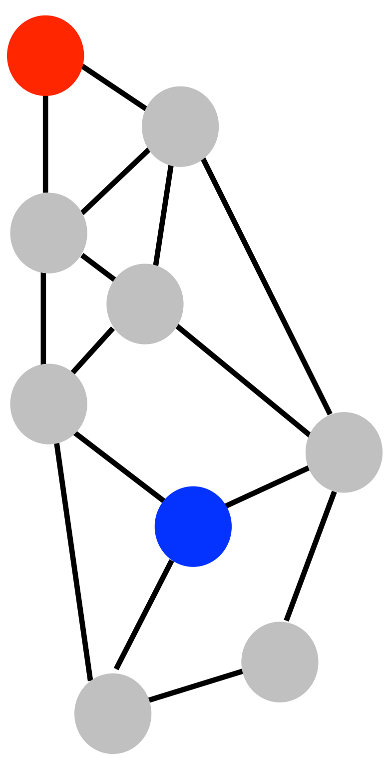
Map-reduce on graph data

- Find nearest feature on a graph

Input

graph

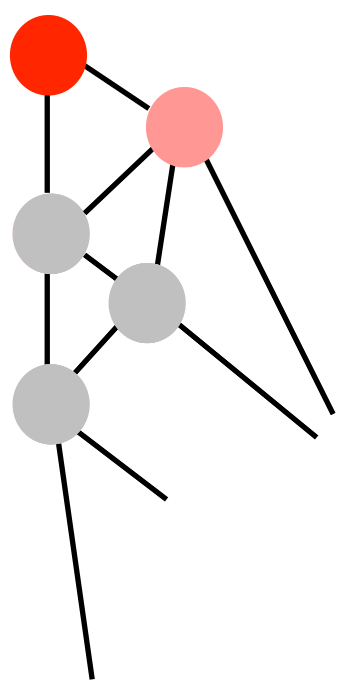
(node,label)



Map

∀, search graph with radius *d*

< ,{ ,*distance*} >

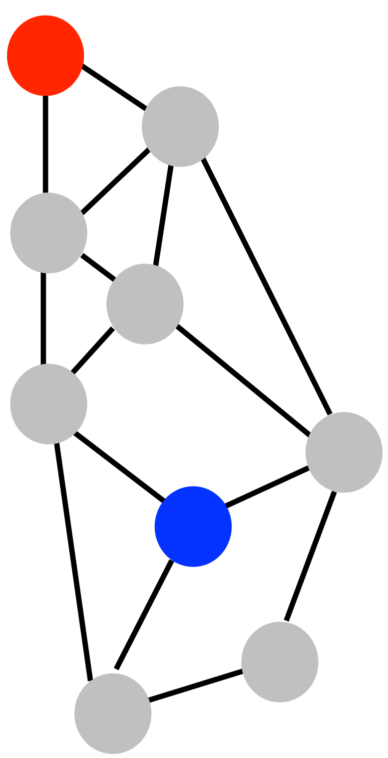


Map-reduce on graph data

- Find nearest feature on a graph

Input

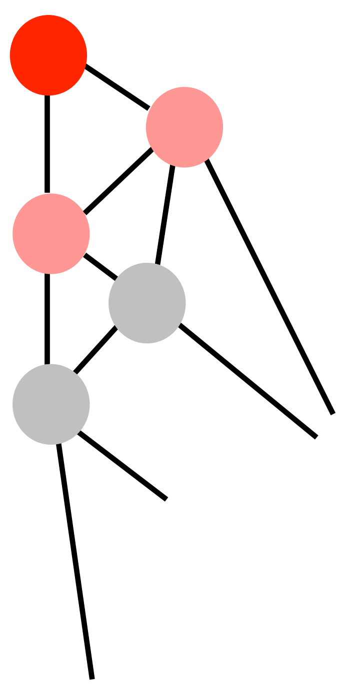
graph
(node,label)



Map

∀●, search graph with radius *d*

< ● ,{● ,*distance*} >



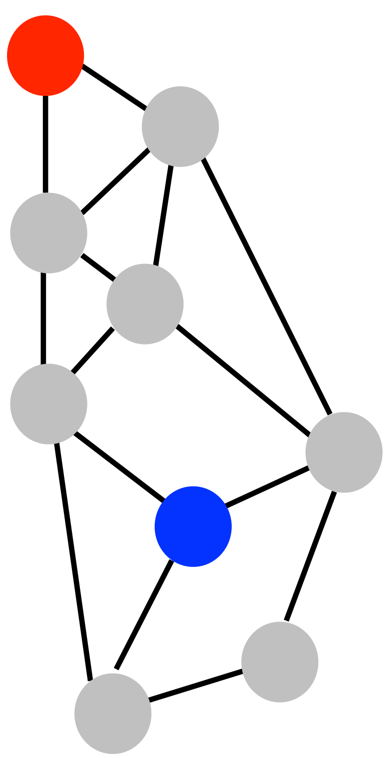
Map-reduce on graph data

- Find nearest feature on a graph

Input

graph

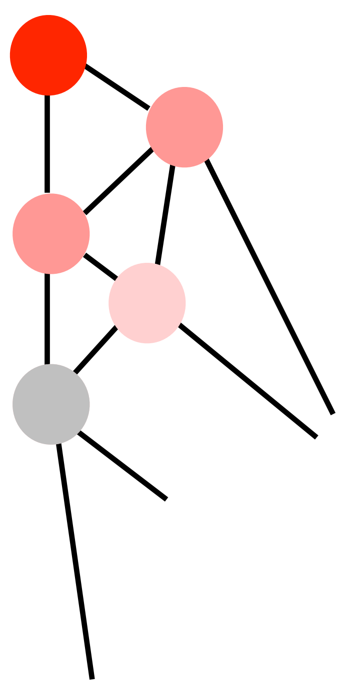
(node,label)



Map

∀●, search graph with radius *d*

< ● ,{● ,*distance*} >

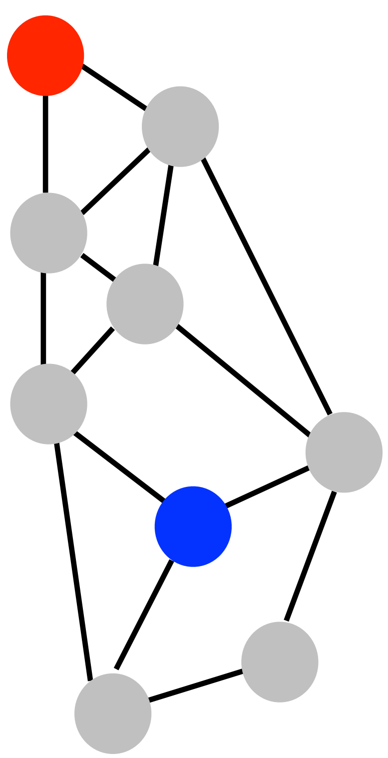


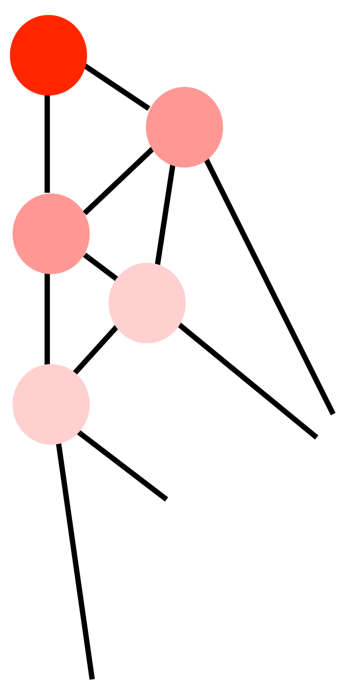
Map-reduce on graph data

- Find nearest feature on a graph

Input
graph
(node,label)

Map
∀●, search graph with radius *d*
< ● ,{● ,*distance*} >





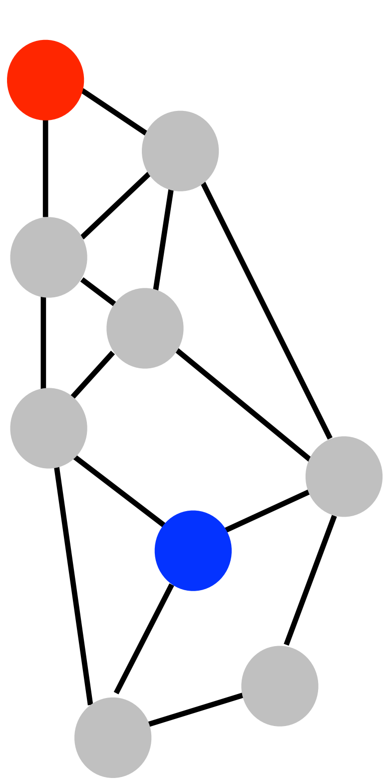
Map-reduce on graph data

- Find nearest feature on a graph

Input

graph

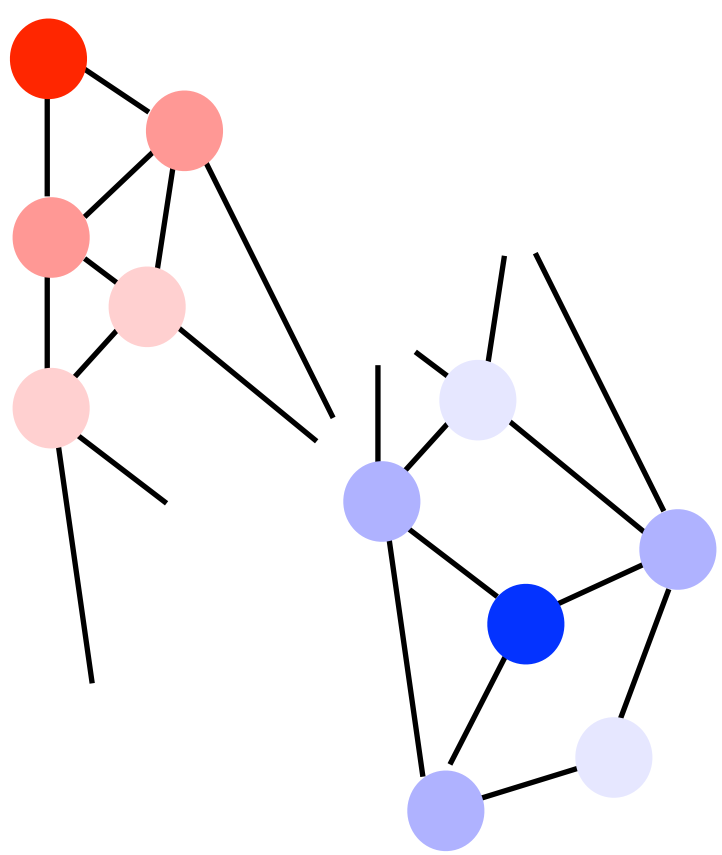
(node,label)



Map

∀●, search graph with radius *d*

< ● ,{● ,*distance*} >



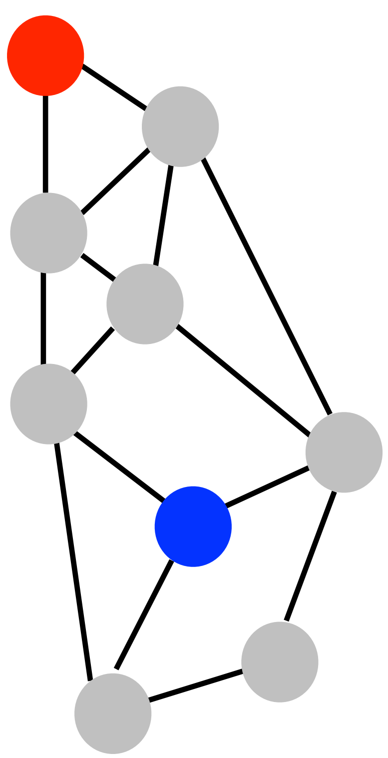
Map-reduce on graph data

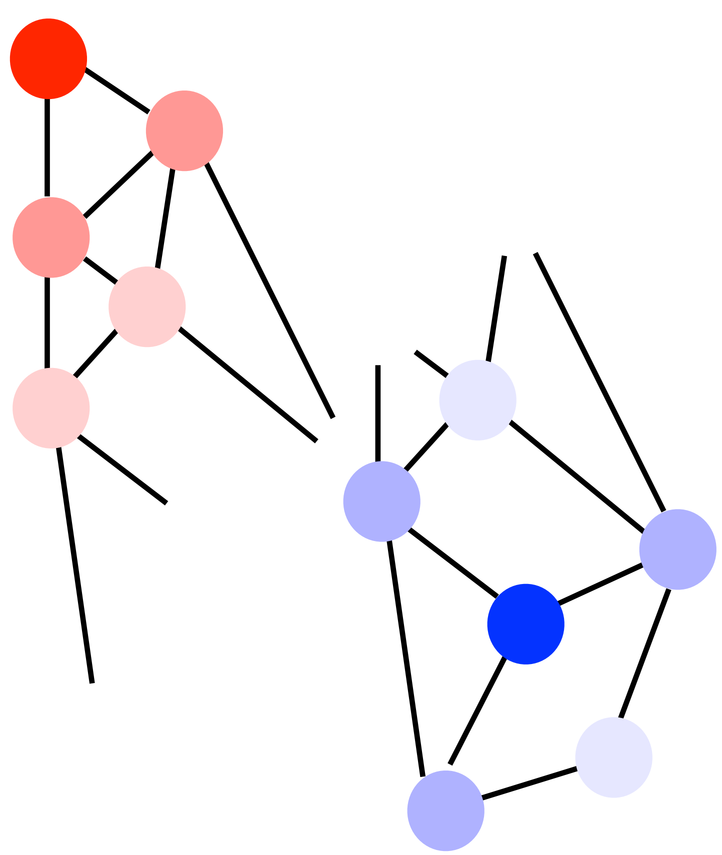
- Find nearest feature on a graph

Input
graph
(node,label)

Map
∀●, search graph
< ● ,{● ,*distance*} >

Shuffle/ Sort
by ● id





Map-reduce on graph data

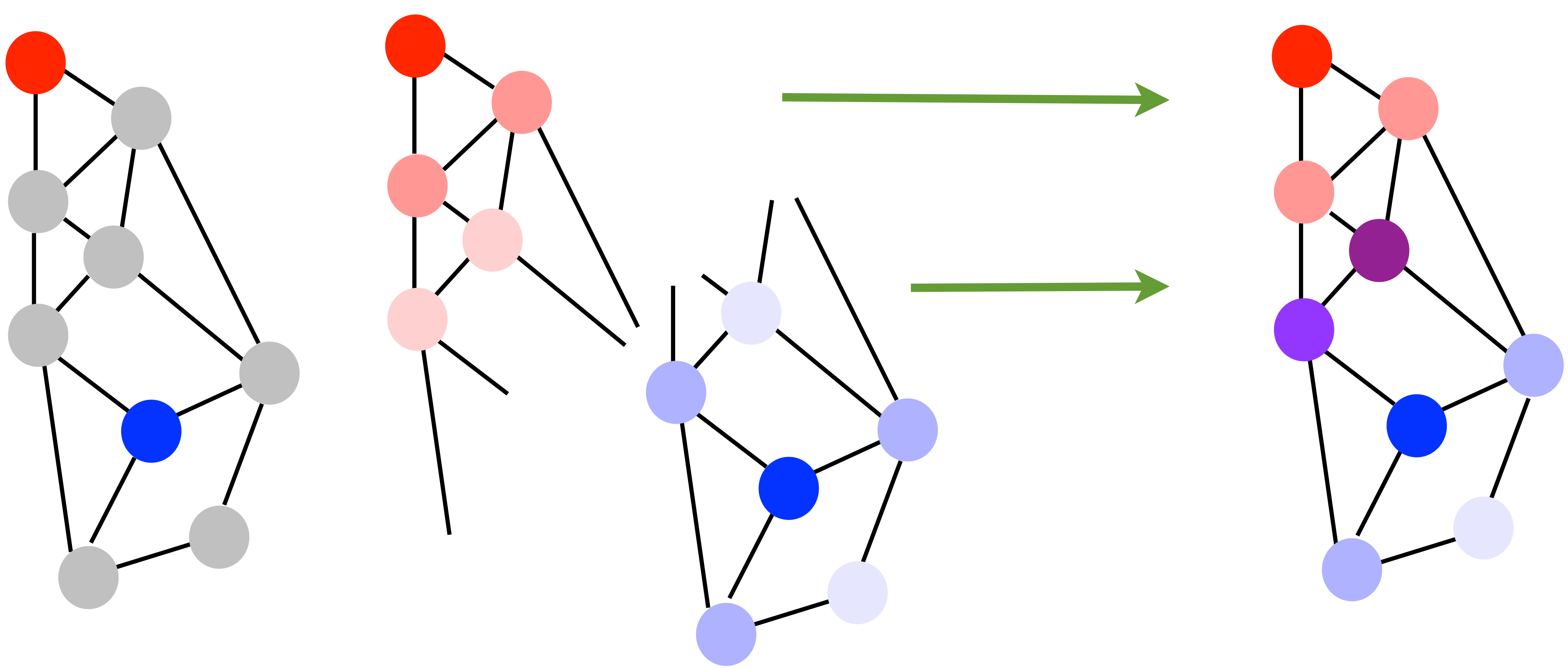
- Find nearest feature on a graph

Input
graph
(node,label)

Map
∀ , search graph
< , { , *distance*} >

Shuffle/ Sort
by id

Reduce
< , [{ , *distance*},
{ , *distance*}] >
-> min()



Map-reduce on graph data

- Find nearest feature on a graph

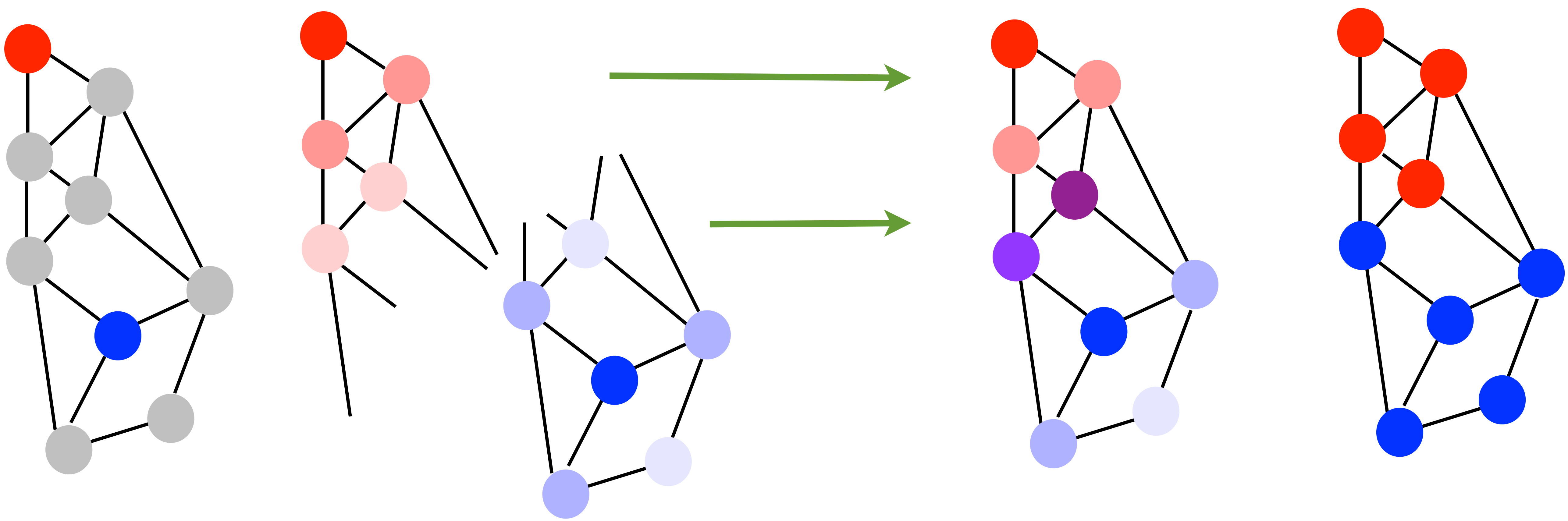
Input
graph
(node,label)

Map
∀ , search graph
< , { ,*distance*} >

Shuffle/ Sort
by id

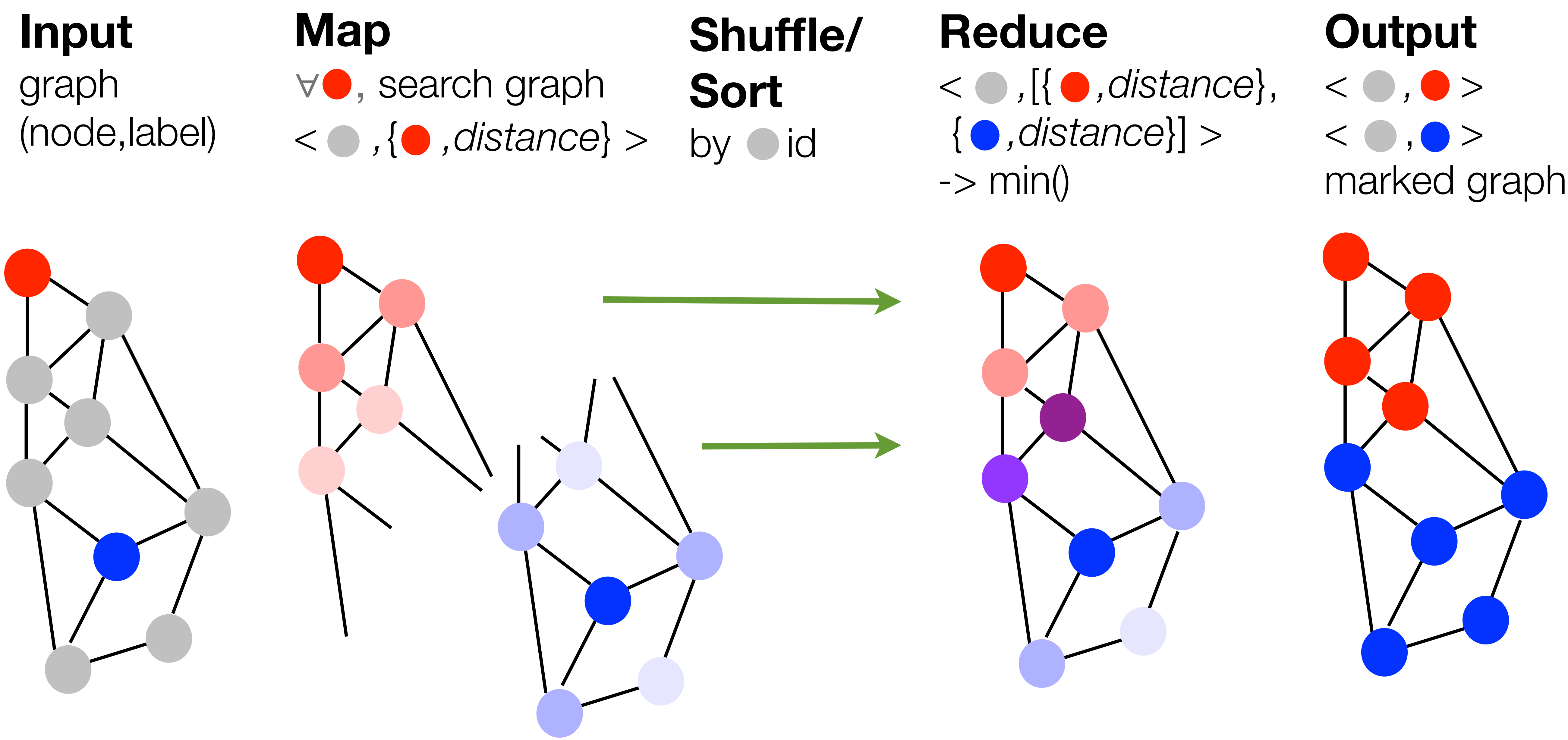
Reduce
< ,[{ ,*distance*},
{ ,*distance*}] >
-> min()

Output
< , >
< , >
marked graph



Map-reduce on graph data

- Find nearest feature on a graph



Be smart about mapping: load nearby nodes on same computing node: choose a good representation

The How?

Hadoop: The Definitive Guide
(Tom White, Yahoo!)

The How?

Hadoop: The Definitive Guide
(Tom White, Yahoo!)

Amazon Elastic Cloud (EC2)
Amazon Simple Storage Service (S3)
$0.085/CPU hour, $0.1/GBmonth

or... your university's HPC center
or... install your own cluster

The How?

Hadoop: The Definitive Guide
(Tom White, Yahoo!)

Amazon Elastic Cloud (EC2)
Amazon Simple Storage Service (S3)
$0.085/CPU hour, $0.1/GBmonth

or… your university's HPC center
or… install your own cluster

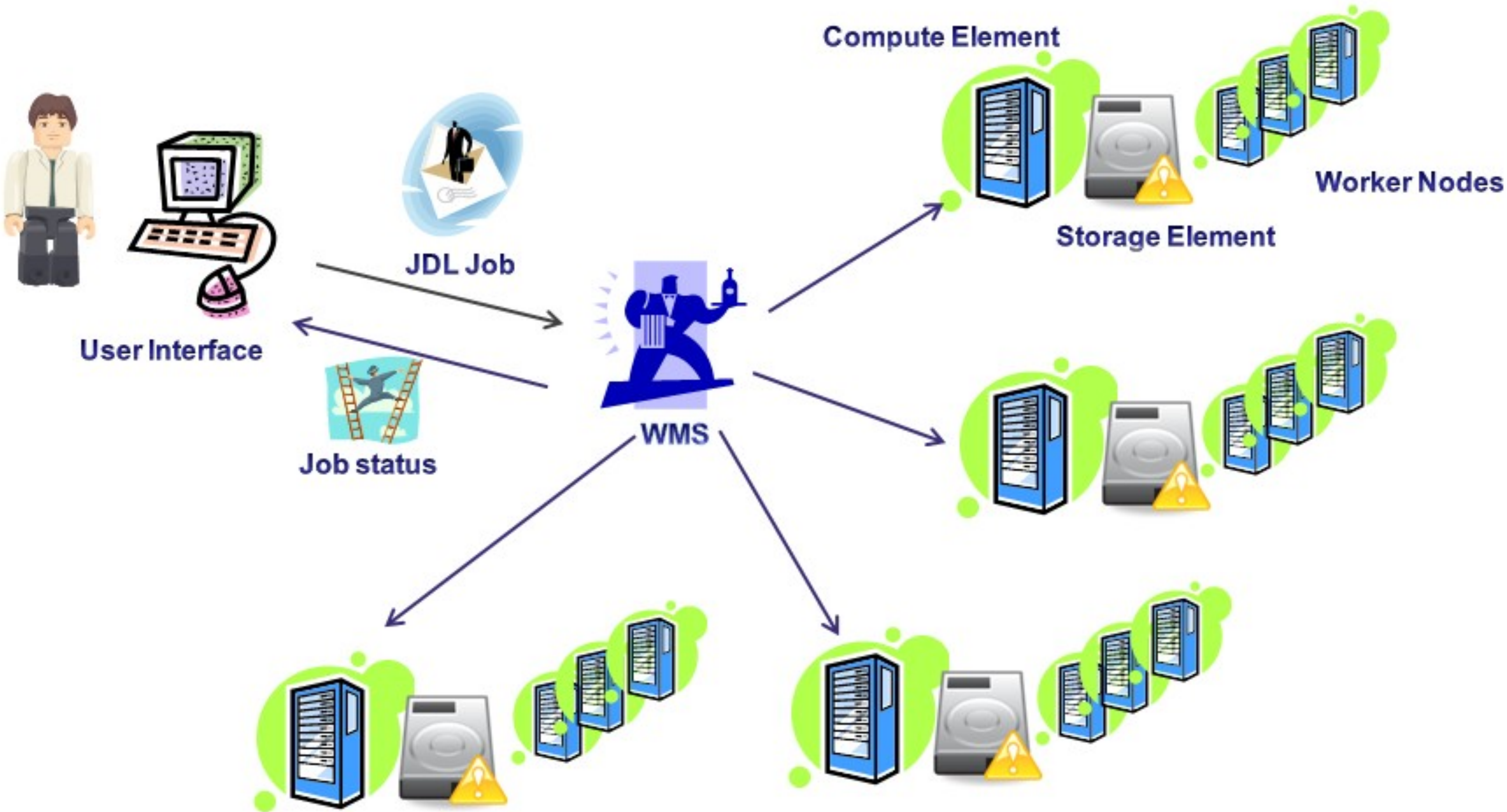
Hadoop-based ML library
Classification: LR, NB, SVM*, NN*, RF
Clustering: kMeans, canopy, EM, spectral
Regression: LWLR*
Pattern mining: Top-k FPGrowth
Dim. Reduction, Coll. Filtering
(*under development)

Part II: Grid computing (CPU bound)

- Disambiguation:
  - Supercomputer (shared memory)
    - CPU + memory bound, you’re rich
    - Identical nodes
  - Cluster computing
    - Parallellizable over many nodes (MPI), you’re rich/patient
    - Similar nodes
  - Grid computing
    - Parallellizable over few nodes or *embarrassingly parallel*
    - Very heterogenous nodes, but loads of `free’ computing power

Grid computing

Grid computing



Grid computing

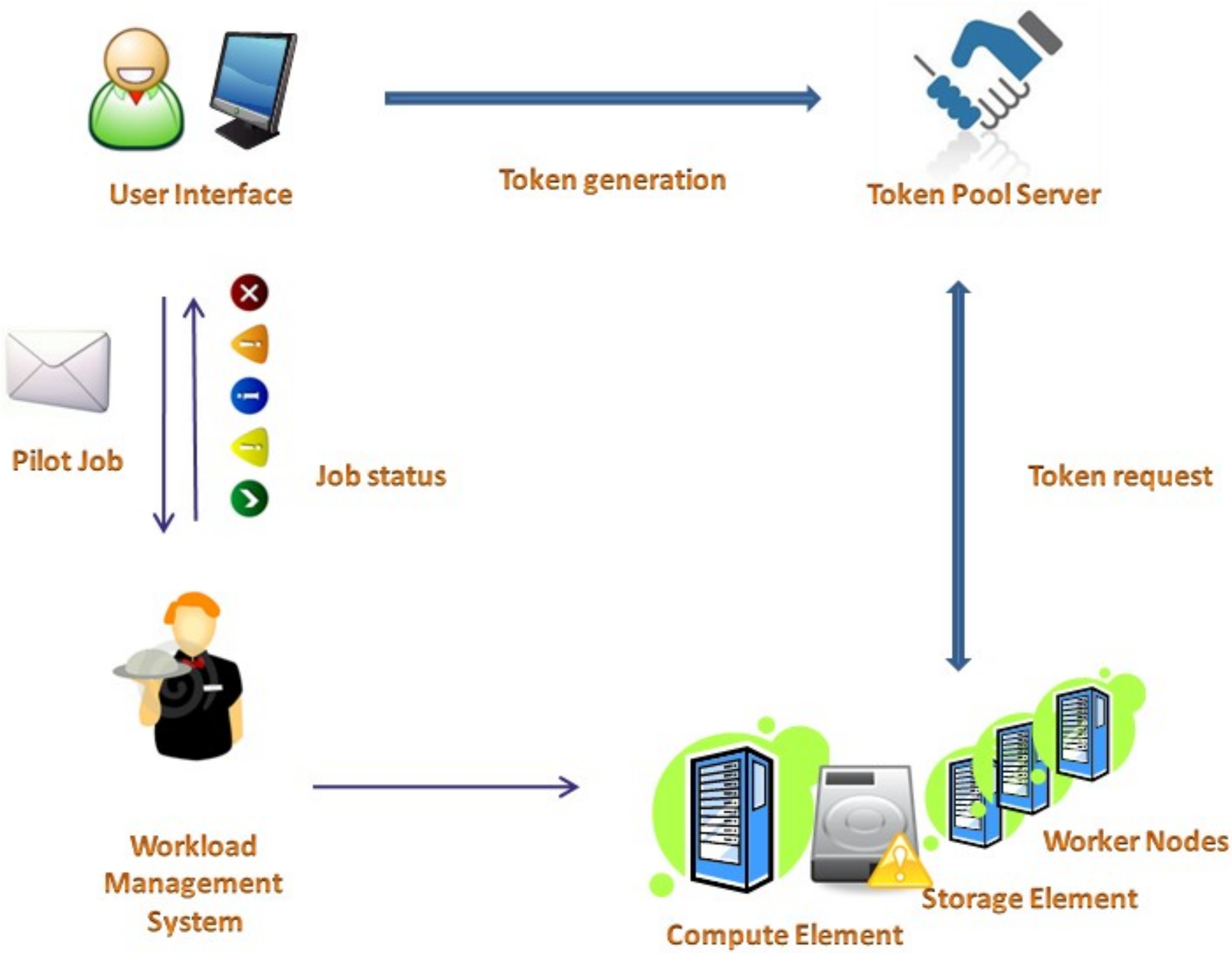
Grid computing

- Grid certificate, you’ll be part of a Virtual Organization

Grid computing

- Grid certificate, you'll be part of a Virtual Organization
- (Complex) middleware commands to move data/jobs to computing nodes:
  - Submit job: glite-wms-job-submit -d $USER -o <jobId> <jdl_file>.jdl
  - Job status: glite-wms-job-status -i <jobID>
  - Copy output to dir: glite-wms-job-output --dir <dirname> -i <jobID>
  - Show storage elements: lcg-infosites --vo ncf se
  - ls: lfc-ls -l $LFC_HOME/joaquin
  - Upload file (copy-register): lcg-cr --vo ncf -d srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/ncf/joaquin/<remotename> -l lfn:/grid/ncf/joaquin/<remotename> "file://$PWD/<localname>"
  - Retrieve file from SE: lcg-cp --vo ncf lfn:/grid/ncf/joaquin/<remotename> file://$PWD/<localname>

Token pool servers



Video

Danke

Thanks

Diolch

Merci

Spasiba

Obrigado

Köszönöm

Dank U

Gracias

Dhanyavaad

Tesekkurler

Arigato

Efharisto

Grazie

Toda

Xie Xie

Hvala