Advanced Cloud Computing

MapReduce

Wei Wang
CSE@HKUST
Spring 2025

THE DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

Common theme?

Parallelization problem arise from

- communication between workers (e.g., state exchange)
- access to shared resources (e.g., data)

Thus, we need a synchronization mechanism

Managing multiple workers

Thus, we need

- semaphores (lock, unlock)
- conditional variables (wait, notify, broadcast)
- barriers (a job cannot start until its prerequisites have completed)

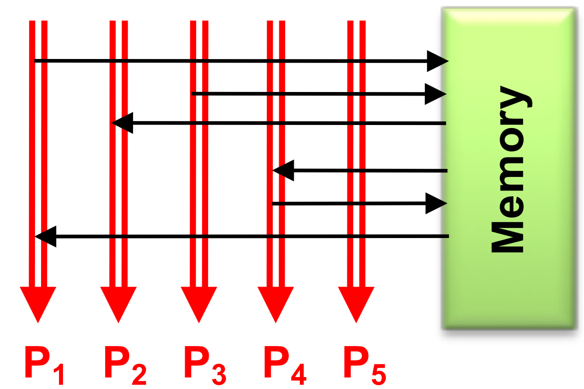
But still…

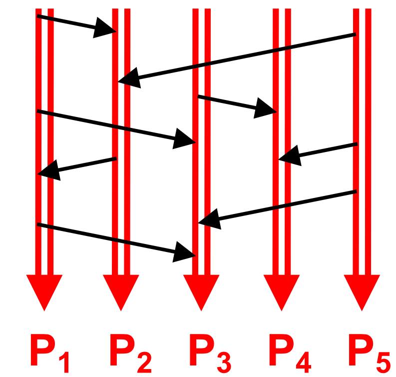
- deadlock, race conditions…
- dinning philosophers, sleeping barbers…

Current tools

Programming models

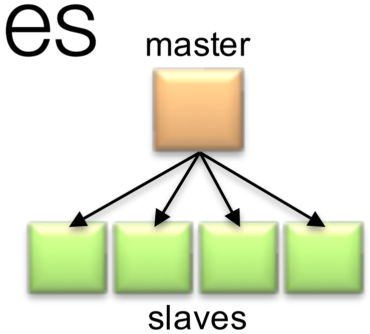
- shared memory (pthreads)
- message passing (MPI)

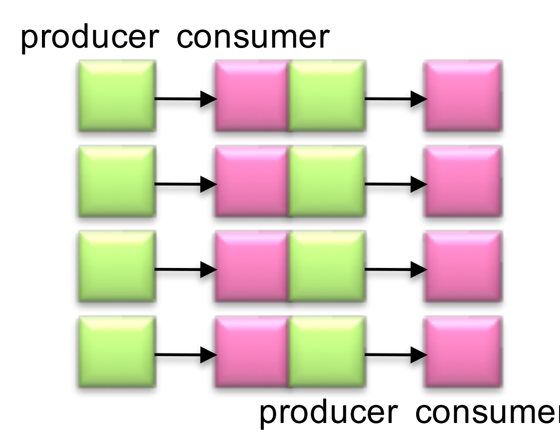


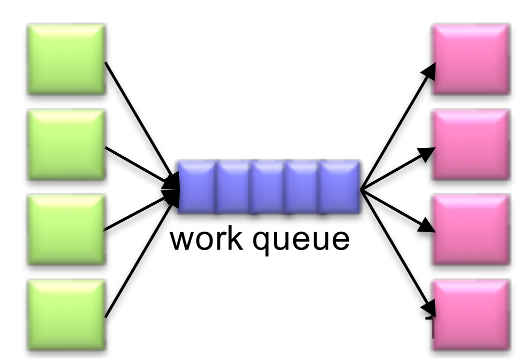


Design Patterns

- master-slaves
- producer-consumer flows
- shared work queues







Where the rubber meets the road

Concurrency is difficult to reason about

And it is even more so

- at the scale of datacenters
- in the presence of failures
- in terms of multiple interacting services

Not to mention debugging…

Typical big data problems

Iterate over a large number of records

Extract something of interest from each

Map

Shuffle and sort intermediate results

Aggregate intermediate results

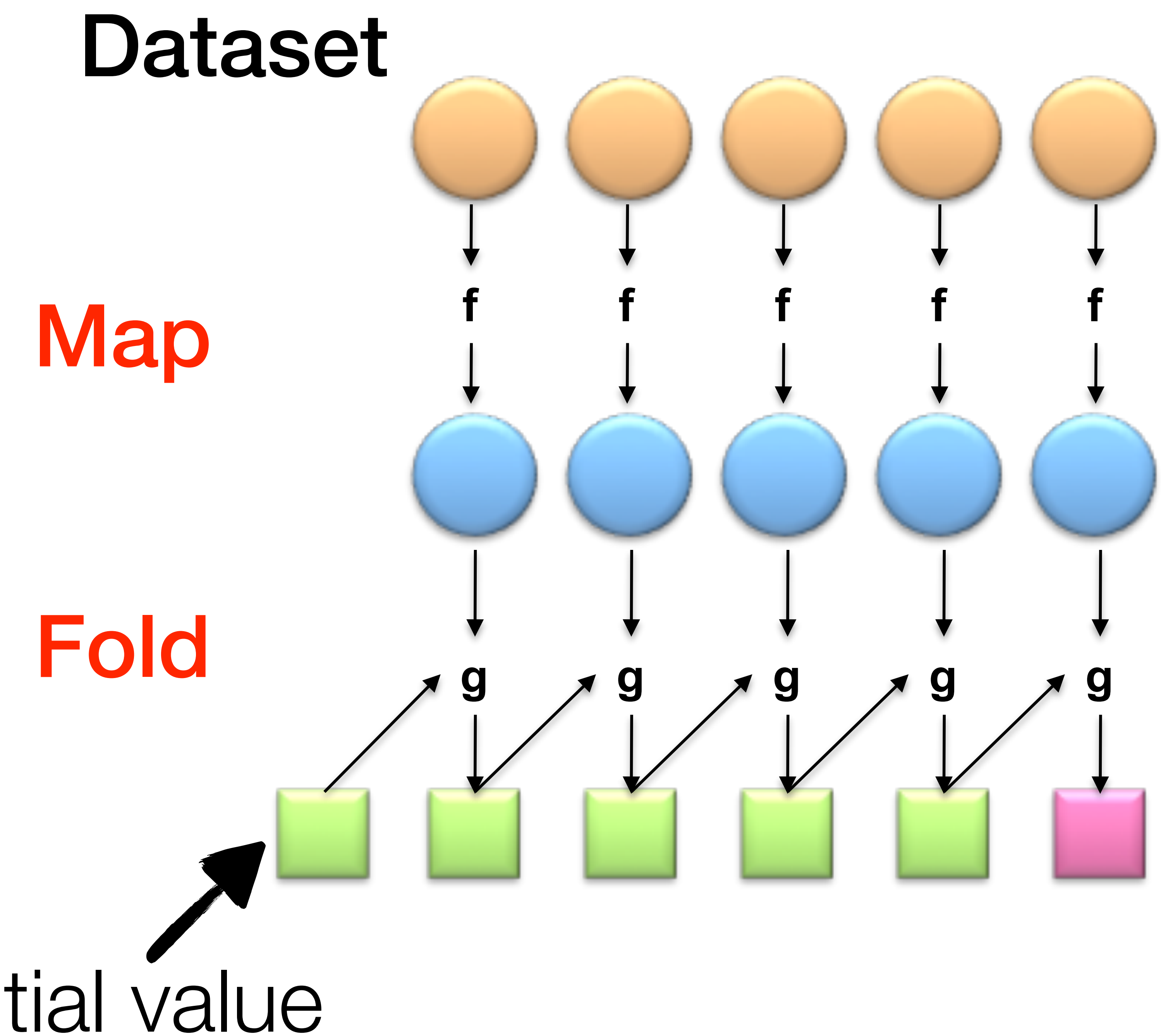
Generate final output

Reduce

Key idea: provide a *functional* abstraction for these two operations

(Dean and Ghemawat, OSDI 2004)

Roots in functional programming



Functional programming

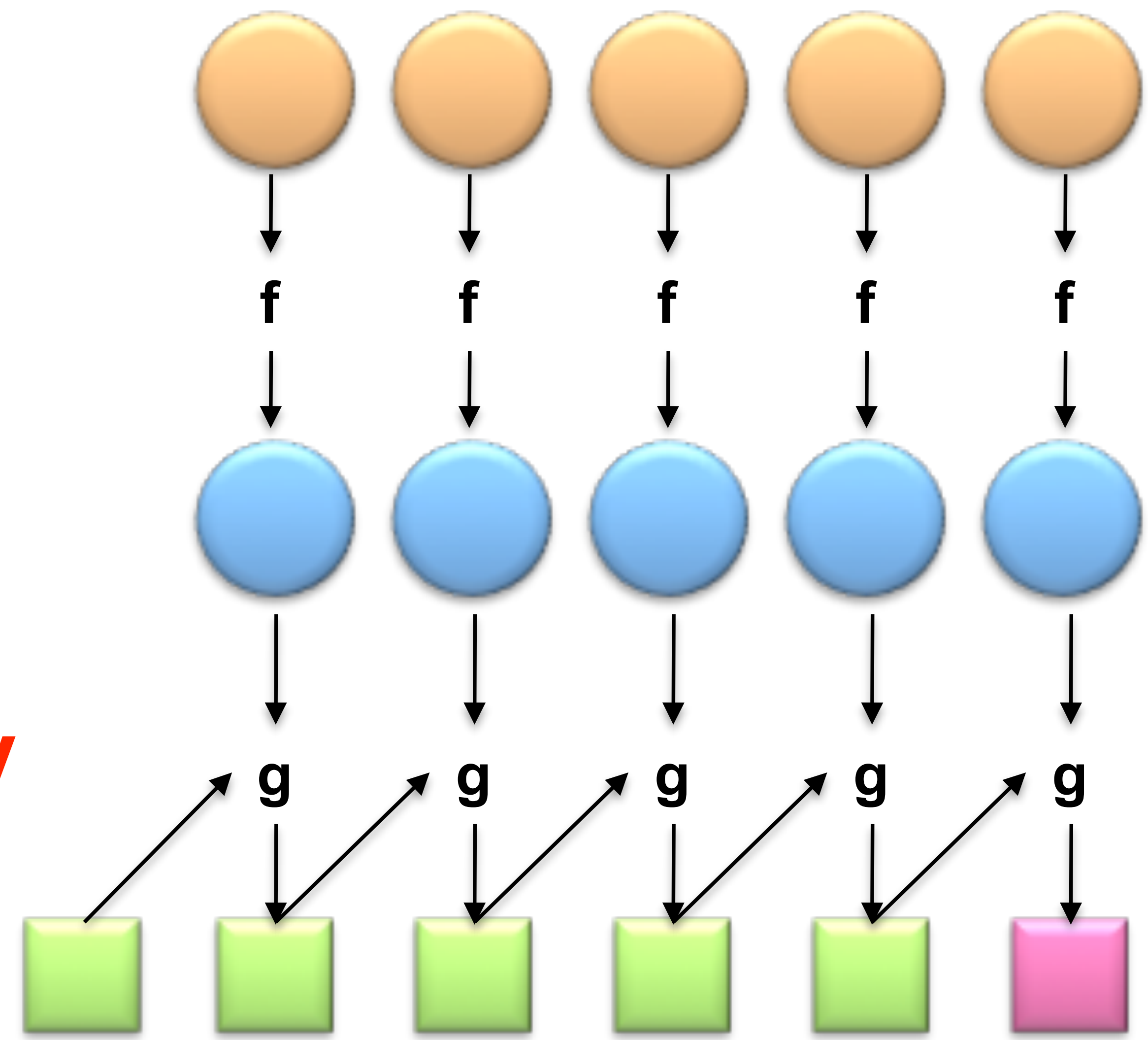
Given a dataset $X = [x_1, \ldots, x_n]$, compute the square sum

$$\sum_i x_i^2$$

f(x) = x²

g(x, y) = x + y

0=

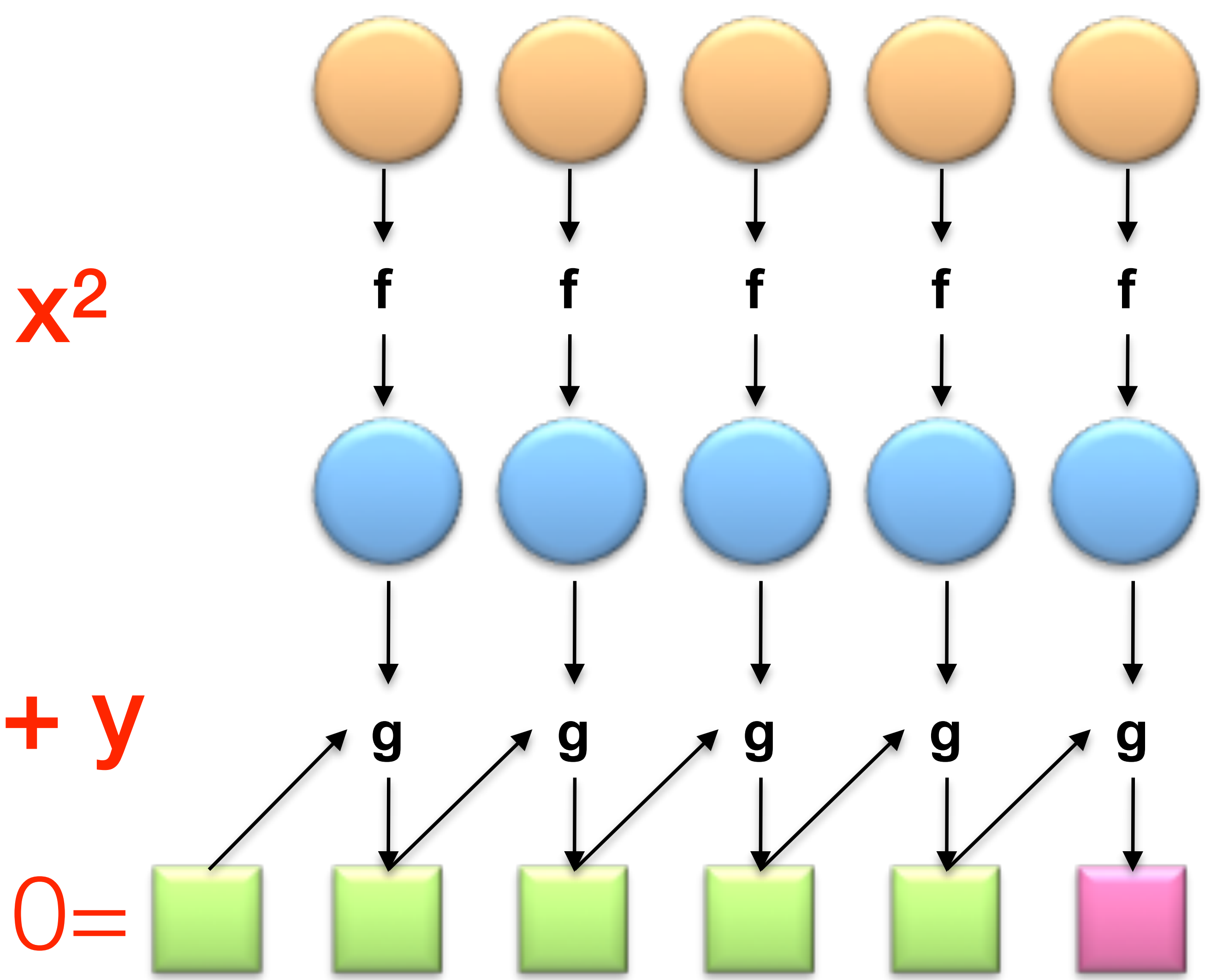


Functional programming

Functional operations never modify existing datasets, but they **create new ones**

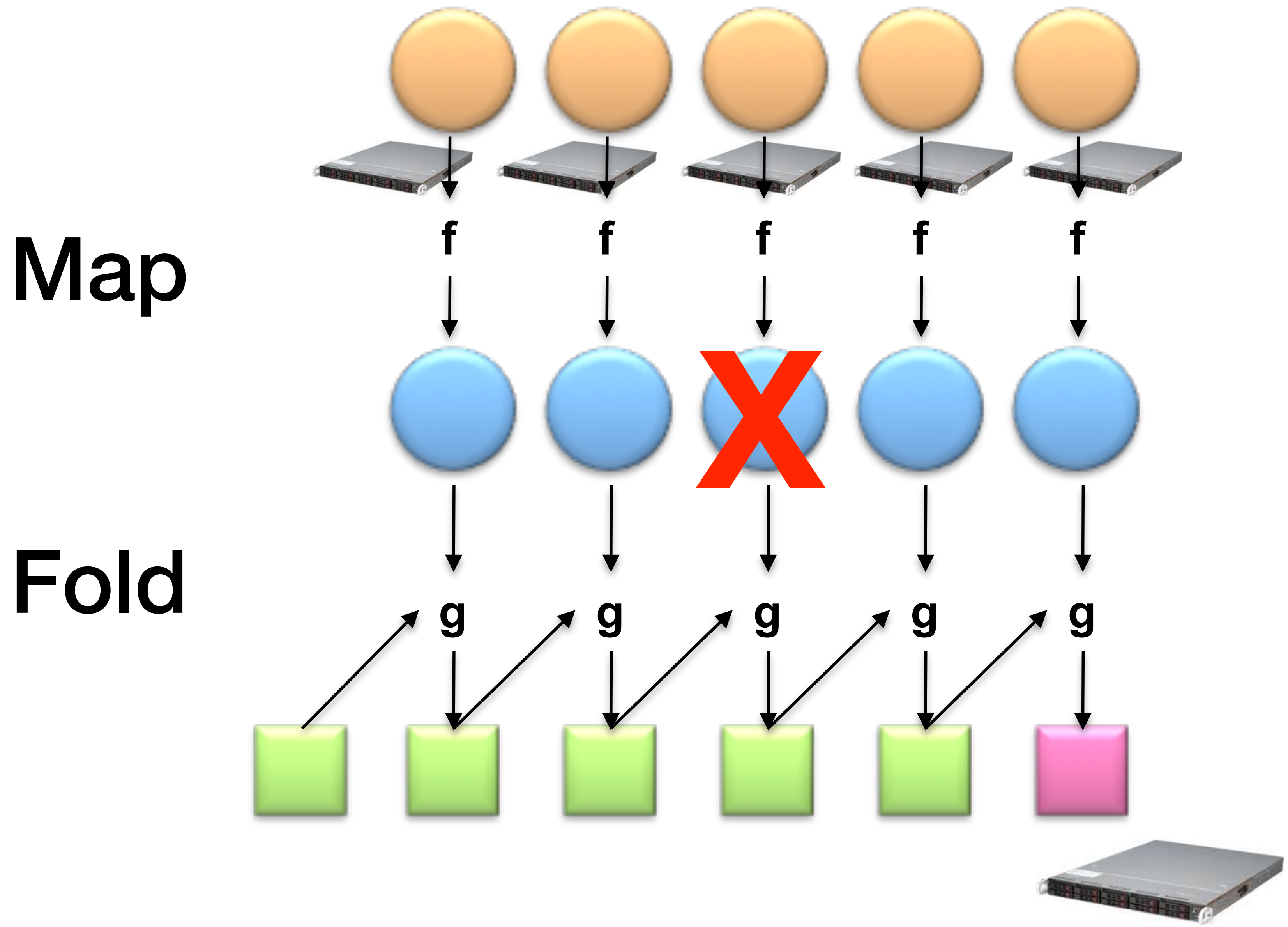
$f(x) = x^2$

$g(x, y) = x + y$



Ideal for parallelization

What if a worker fails?

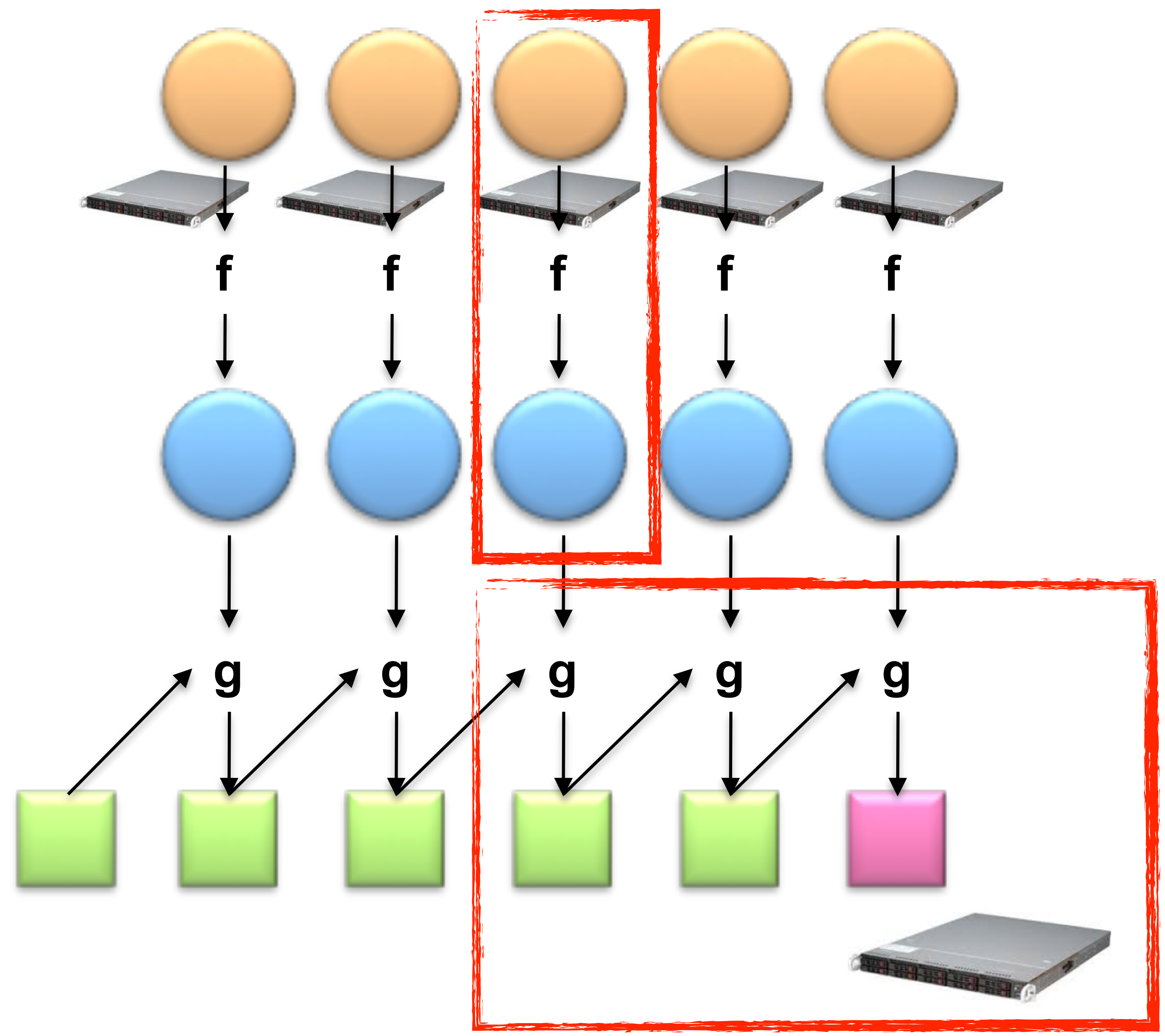


Ideal for parallelization

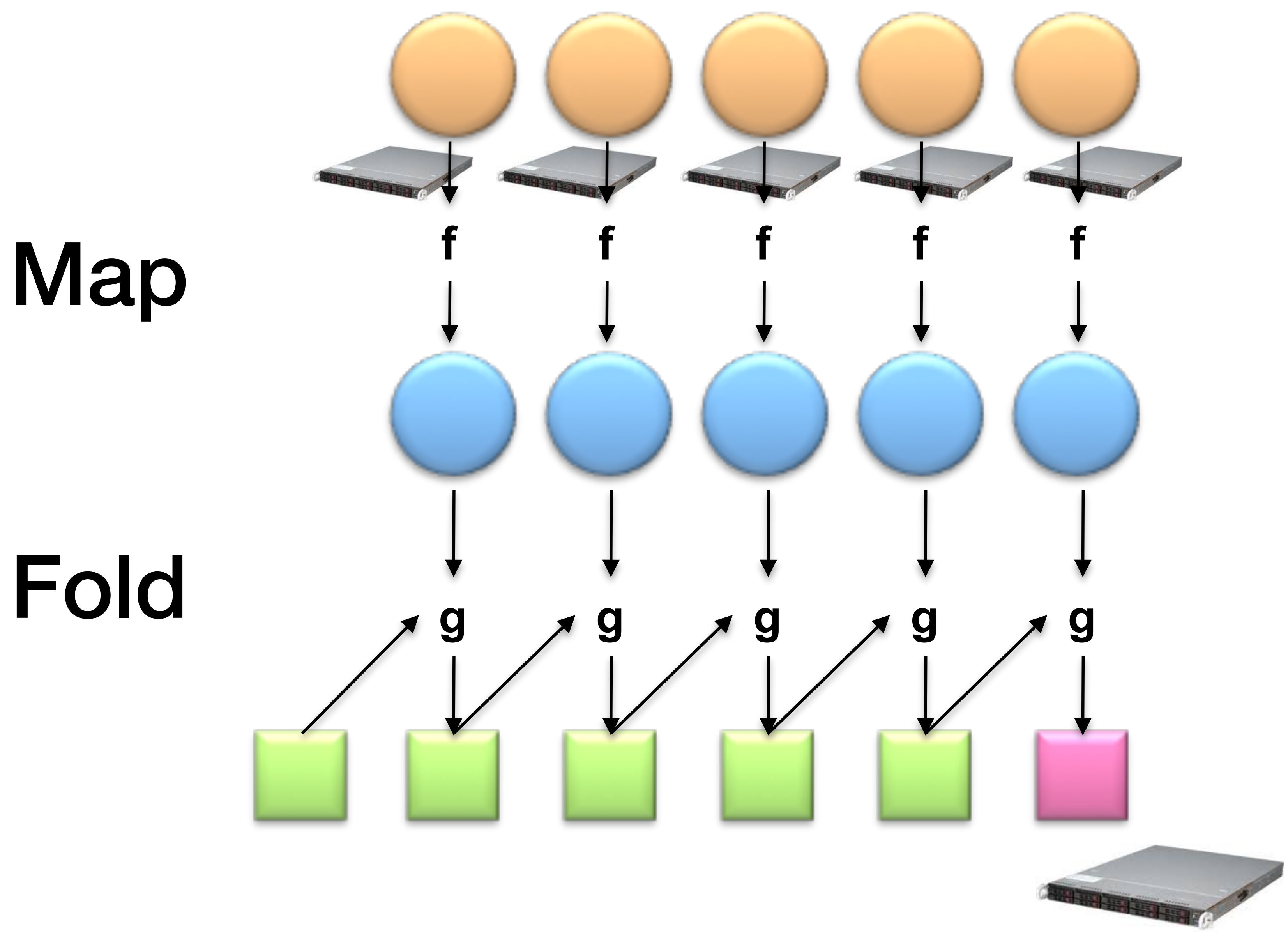
Do the work again, on some other machines

Map

Fold



Can we apply any function?

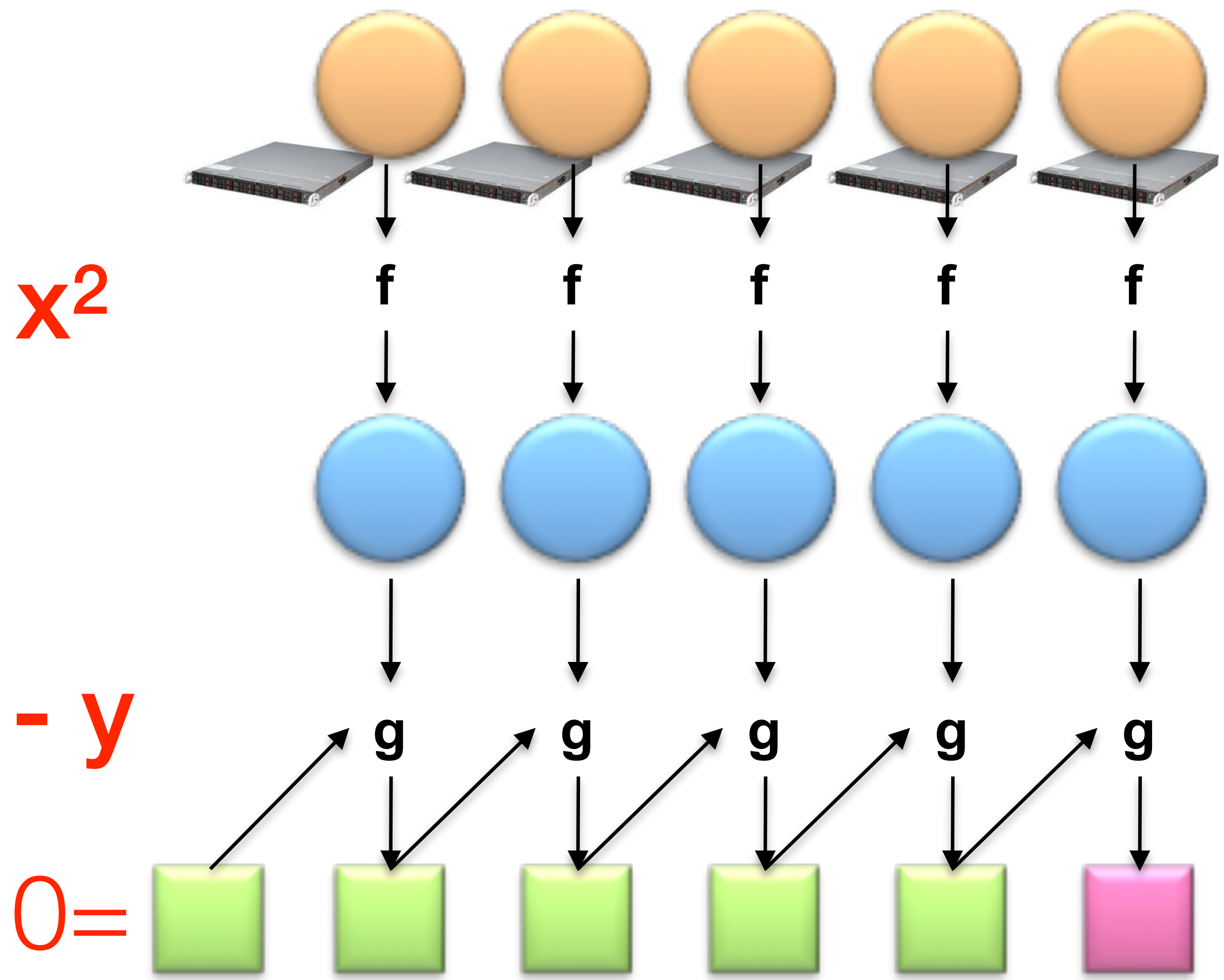


Can we apply any function?

What if $g(x, y) = x - y$?

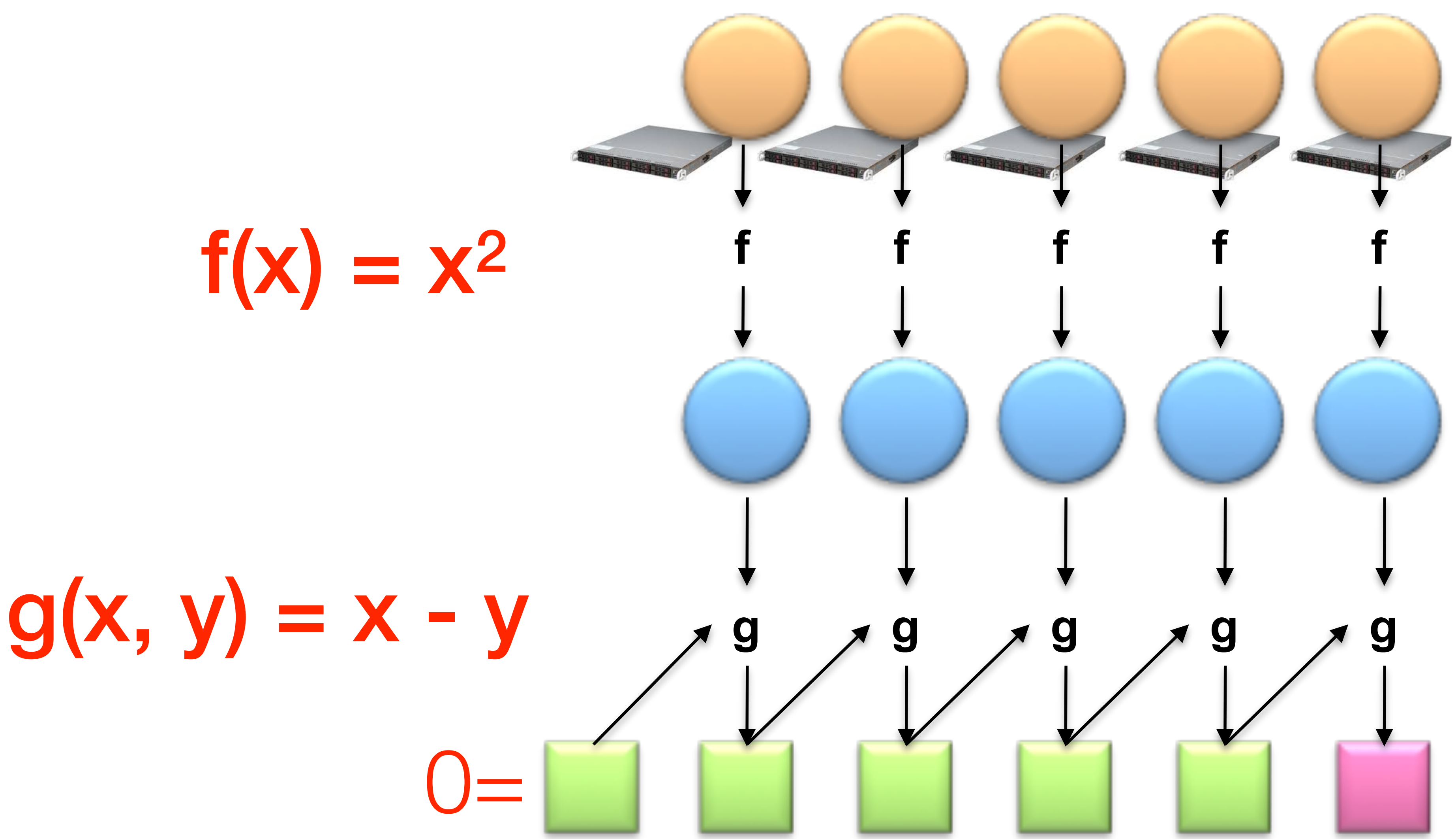
$f(x) = x^2$

$g(x, y) = x - y$



Nope…

The order matters, making the results indefinite and hard to reason about!



Thus, we require…

Commutativity

- $g(x, y) = g(y, x)$
- e.g., $x + y = y + x$

Associativity

- $g(g(x, y), z) = g(x, g(y, z))$
- e.g., $(x + y) + z = x + (y + z)$

The programming model of MapReduce borrows from functional programming

Records from the data source are fed as **key-value pairs**, e.g., [<filename, line>]

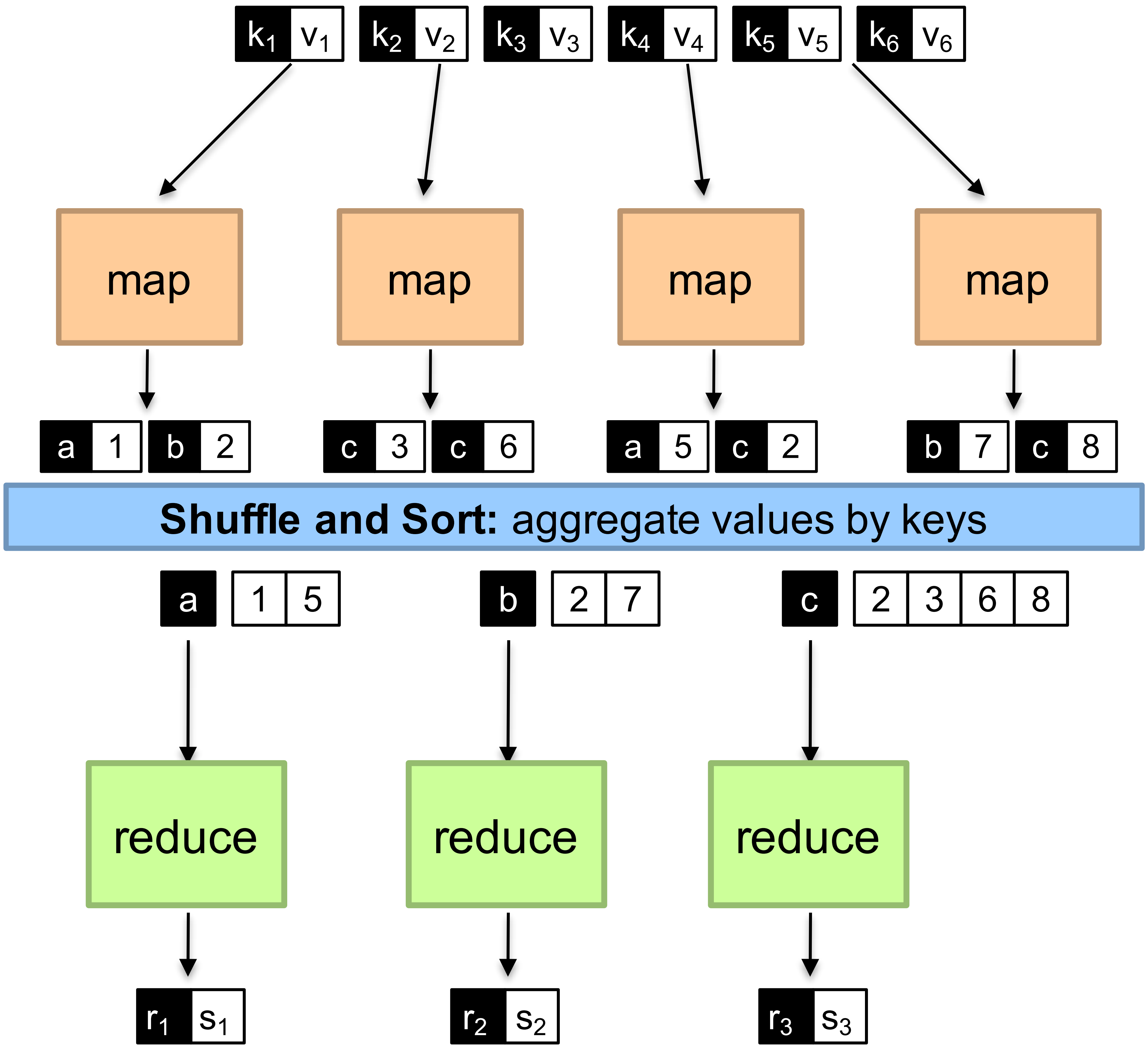
MapReduce

Programmers specify two functions

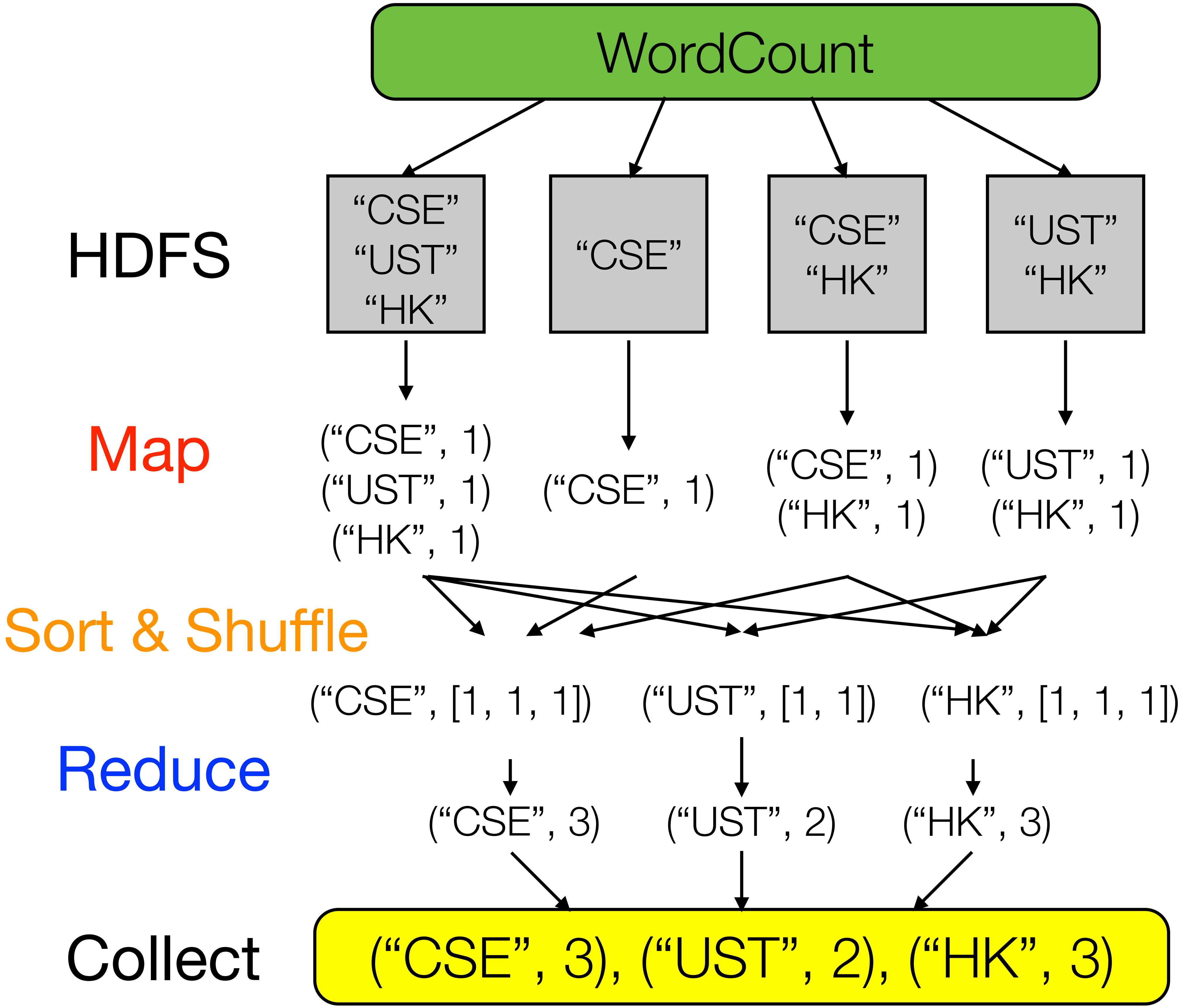
- **map** (k, v) → [<k_2, v_2>]
- **reduce** (k_2, [v_2]) → [<k_3, v_3>]

All values with the **same key** are sent to the **same reducer**

The execution framework handles ***everything else***…


k1 v1 k2 v2 k3 v3 k4 v4 k5 v5 k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 3 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

WordCount: a “Hello World” from MapReduce


WordCount
HDFS
"CSE" "UST" "HK"
"CSE"
"CSE" "HK"
"UST" "HK"
Map
("CSE", 1) ("UST", 1) ("HK", 1)
("CSE", 1)
("CSE", 1) ("HK", 1)
("UST", 1) ("HK", 1)
Sort & Shuffle
("CSE", [1, 1, 1])
("UST", [1, 1])
("HK", [1, 1, 1])
Reduce
("CSE", 3)
("UST", 2)
("HK", 3)
Collect
("CSE", 3), ("UST", 2), ("HK", 3)

A tale of two functions…

What to emit?

```
Map(String docid, String text):
    for each word w in text:
        Emit(w, 1);

Reduce(String term, Iterator<Int> values):
    int sum = 0;
    for each v in values:
        sum += v;
    Emit(term, sum)
```

How to reduce?

MapReduce

Programmers specify two functions

- **map** $(k_1, v_1) \rightarrow [<k_2, v_2>]$
- **reduce** $(k_2, [v_2]) \rightarrow [<k_3, v_3>]$

All values with the **same key** are sent to the **same reducer**

The execution framework handles ***everything else***…

What's "everything else"?

Everything else…

Handles scheduling

- assigns workers to map and reduce tasks
- load balancing

Handles “data distribution”

- move processes to data
- automatic parallelization

Everything else…

Handles synchronization

- gathers, sorts, and shuffles intermediate data
- network and disk transfer optimization

Handles errors and faults

- detects worker failures and restarts

Everything happens on top of a distributed filesystem

MapReduce refinement

Programmers specify two functions

- **map** $(k_1, v_1) \rightarrow [\langle k_2, v_2 \rangle]$
- **reduce** $(k_2, [v_2]) \rightarrow [\langle k_3, v_3 \rangle]$
- All values with the **same key** are reduced together

Not quite… usually, programmers also specify combiner and partitioner

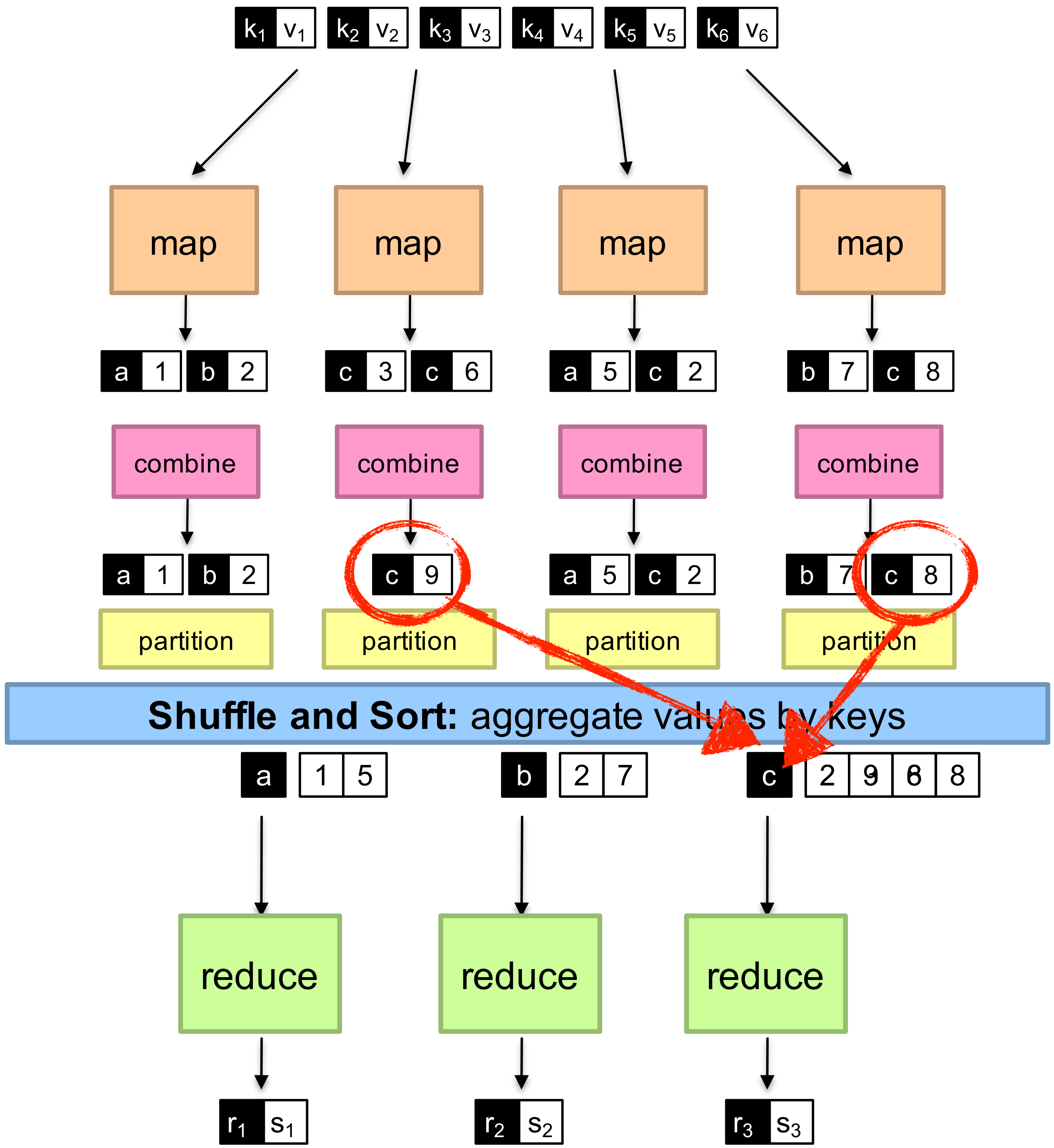
Combiner and partitioner

combine (k, [v]) → <k, v>

- mini-reducers that run in memory after the map phase
- used as an optimization to reduce network traffic

partition (k, # of partitions) → partition for k

- divides up key spaces for parallel reduce operations
- often a simple hash of the key, e.g., hash(k) mod n


k1 v1 k2 v2 k3 v3 k4 v4 k5 v5 k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 8 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

MapReduce

Programmers specify:

- **map** $(k_1, v_1) \rightarrow [<k_2, v_2>]$
- **combine** $(k_2, [v_2]) \rightarrow <k_2, v_2>$
- **partition** $(k_2$, # of partitions$) \rightarrow$ partition for k_2
- **reduce** $(k_2, [v_2]) \rightarrow [<k_3, v_3>]$

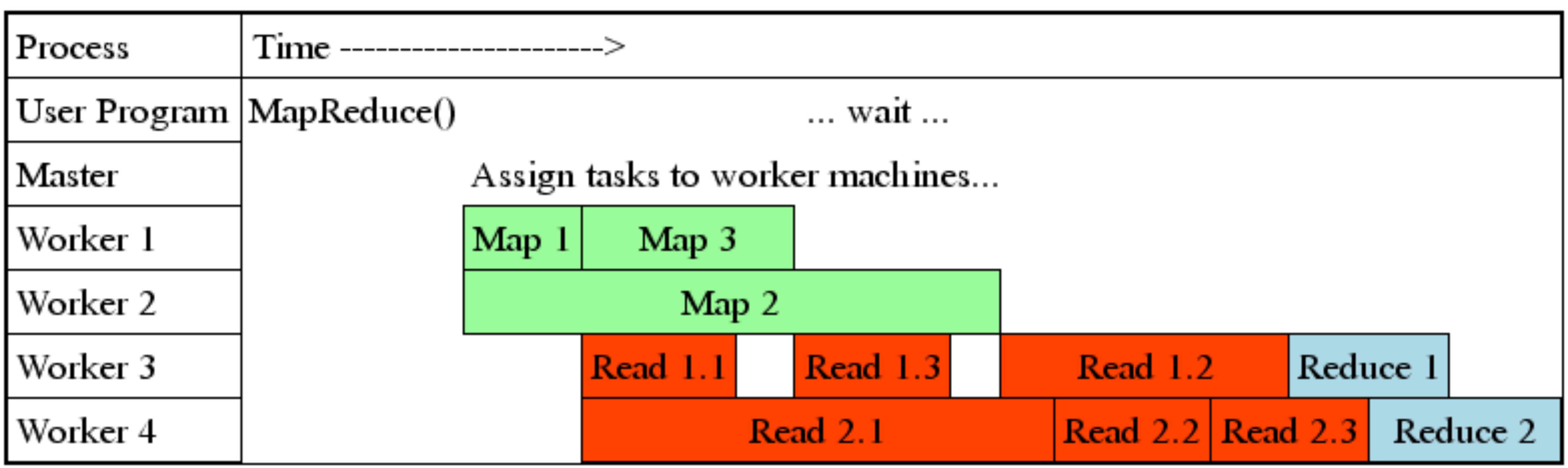
All values with the **same key** are reduced together

The execution framework handles **everything else**…

Two more details…

Barrier between map and reduce phases

- no reduce can start until map is complete
- but we can begin transferring intermediate data earlier to **pipeline** shuffling with map execution



Two more details…

Barrier between map and reduce phases

- no reduce can start until map is complete
- but we can begin transferring intermediate data earlier to **pipeline** shuffling with map execution

Keys arrive at each reducer in **sorted order**

- no enforced ordering *across* reducers

MapReduce can refer to…

The programming model

The execution framework (aka “runtime”)

The specific implementation

Usage is usually clear from context!

MapReduce Implementations

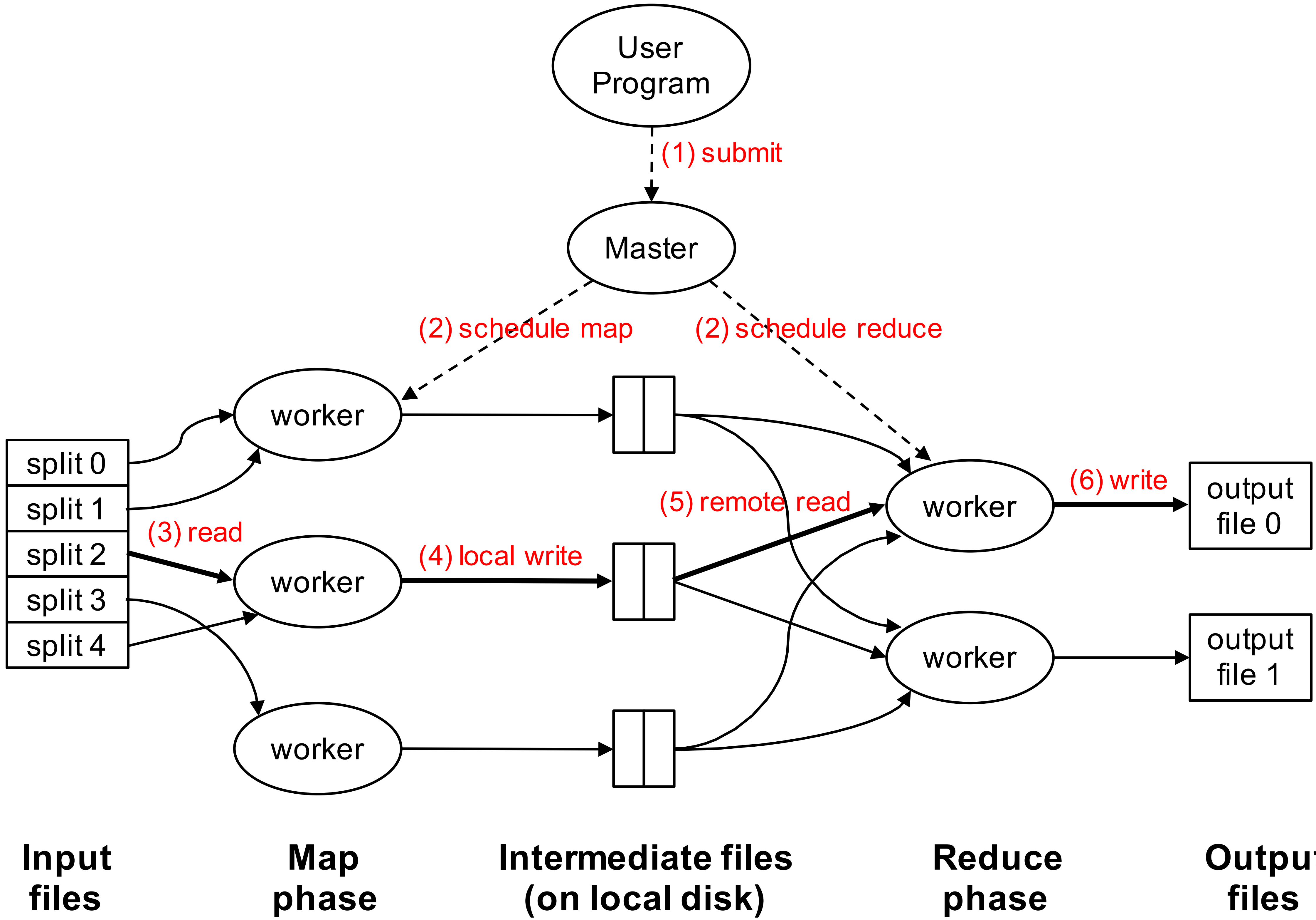
MapReduce implementations

Google has a proprietary implementation in C++

- bindings in Java, Python
- master-slave architecture

One of the most influential work in computer systems:

- J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In USENIX OSDI, 2004.



local filesystem

Adapted from (Dean and Ghemawat, OSDI 2004)

Credits

Some slides are adapted from Prof. Jimmy Lin’s slides at the University of Waterloo