MapReduce

Doug Woos

Logistics notes

Deadlines, etc. up on website

Slip day policy

Piazza!!!

https://piazza.com/washington/spring2017/cse452

Outline

- Why MapReduce?
- Programming model
- Implementation
- Technical details (performance, failure, limitations)
- Lab 1
- Piazza discussion

Why MapReduce?

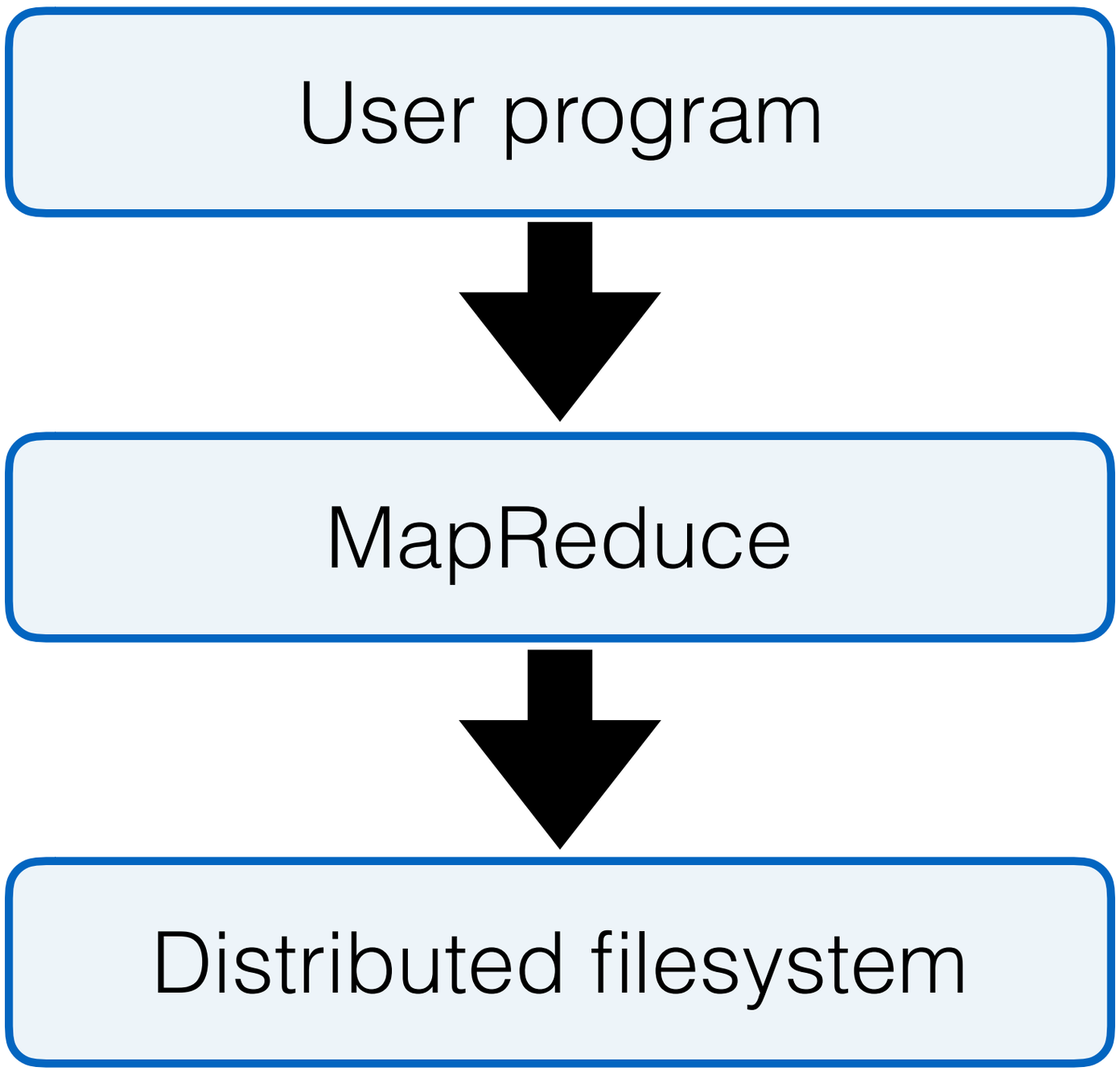
Distributed systems are **hard**

- Failure
- Consistency
- Performance
- Testing
- etc. etc

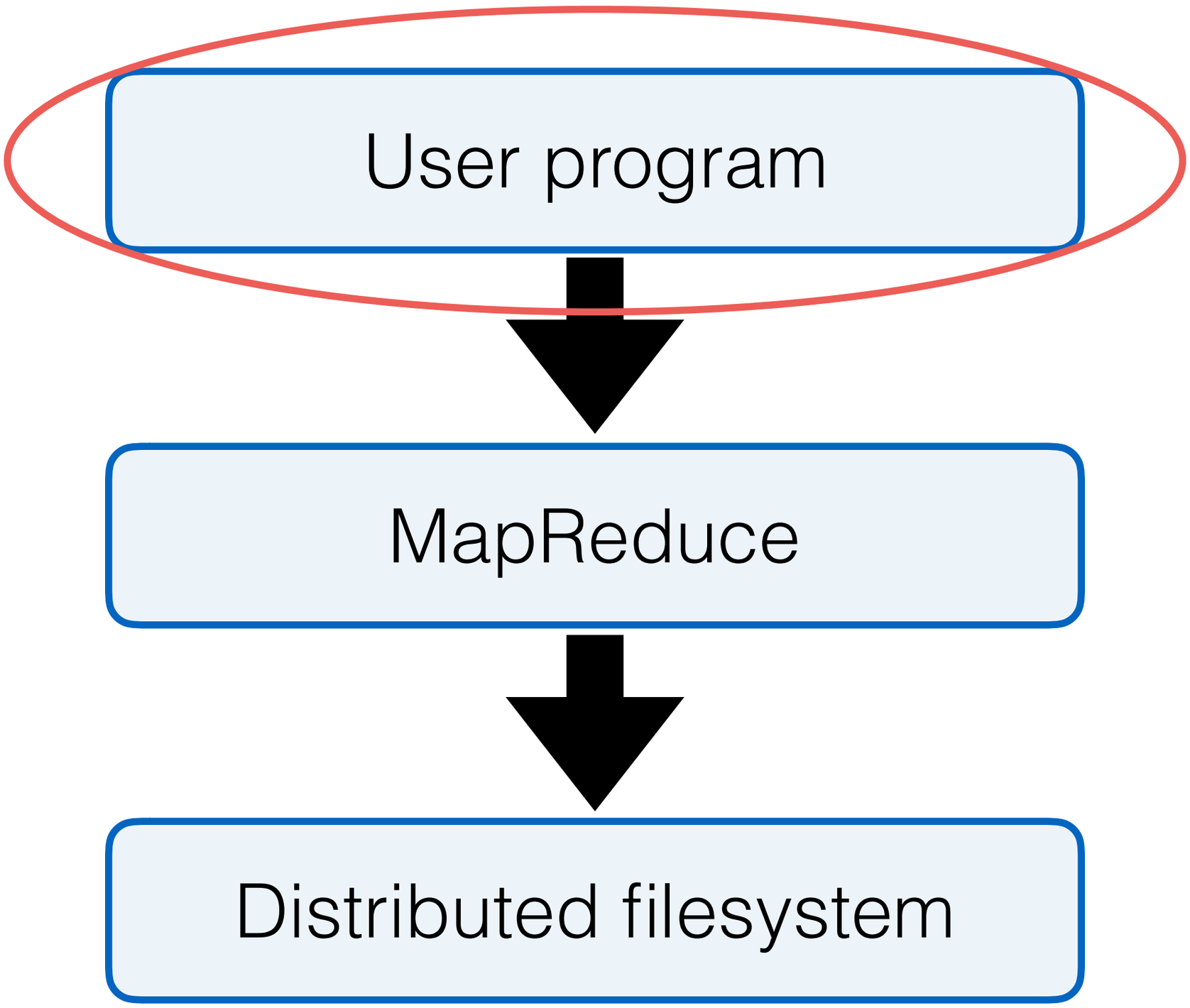
Shouldn’t have to write one for every task

- separation of concerns

Separation of concerns



Separation of concerns



Programming model

Input: list of key/value pairs

```
[(“1”, “in the town where I was born”),
 (“2”, “lived a man who sailed to sea”),
 (“3”, “and he told us of his life”),
 (“4”, “in the land of submarines”),
 …]
```

Output: list of key/value pairs

```
[(“13”, “yellow”),
 (“9”, “submarine”),
 (“7”, “in”),
 (“7”, “we”),
 …]
```

Programming model

Map: (k1, v1) -> [(k2, v2)]

```
for word in value:
  emit (word, “1”)
```

Reduce: (k2, [v2]) -> [v3]

```
emit len(values)
```

Programming model

Map runs on every key/value pair, produces new pairs

```
[(“In”, “1”), (“the”, “1”), (“town”, “1”), (“where”, “1”), …]
```

Resulting pairs sorted by key

```
[[(“a”, “1”), (“a”, “1”), (“a”, “1”), …],
 [(“and”, “1”), (“and”, “1”), (“and”, “1”), …],
 …]
```

Reduce runs on every key and all associated values

```
[(“13”, “yellow”),
 (“9”, “submarine”),
 (“7”, “in”),
 (“7”, “we”),
 …]
```

Other example programs

Surprising anagram finder

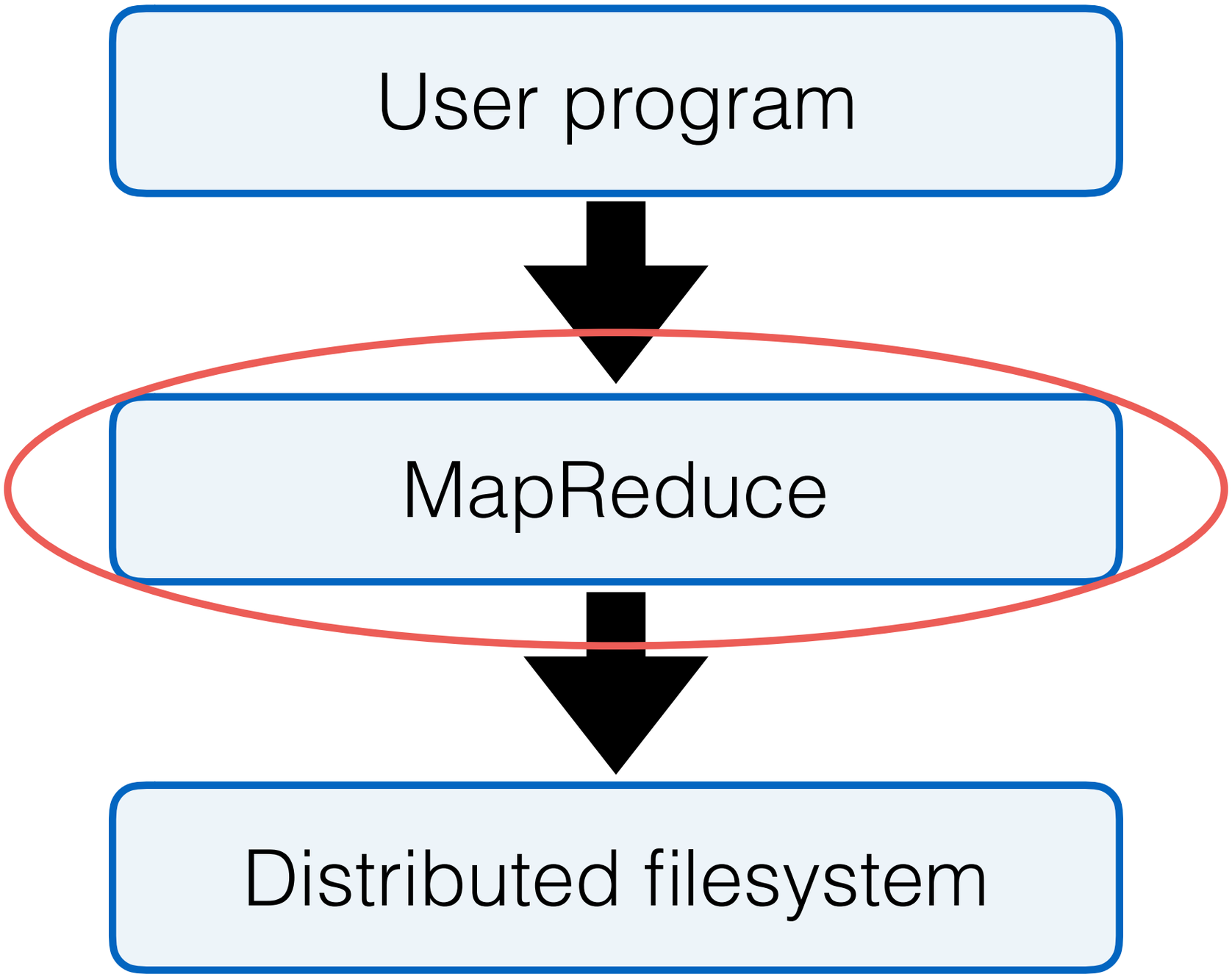
```
- emit (sorted(value), value)
- emit highest scoring anagram in values
```

PageRank

```
- for outbound link in page.links:
    emit (url, page.rank)
- page.rank = sum(page.rank for page in links) / len(page.links)
```

Others?

Separation of concerns



MapReduce Implementation

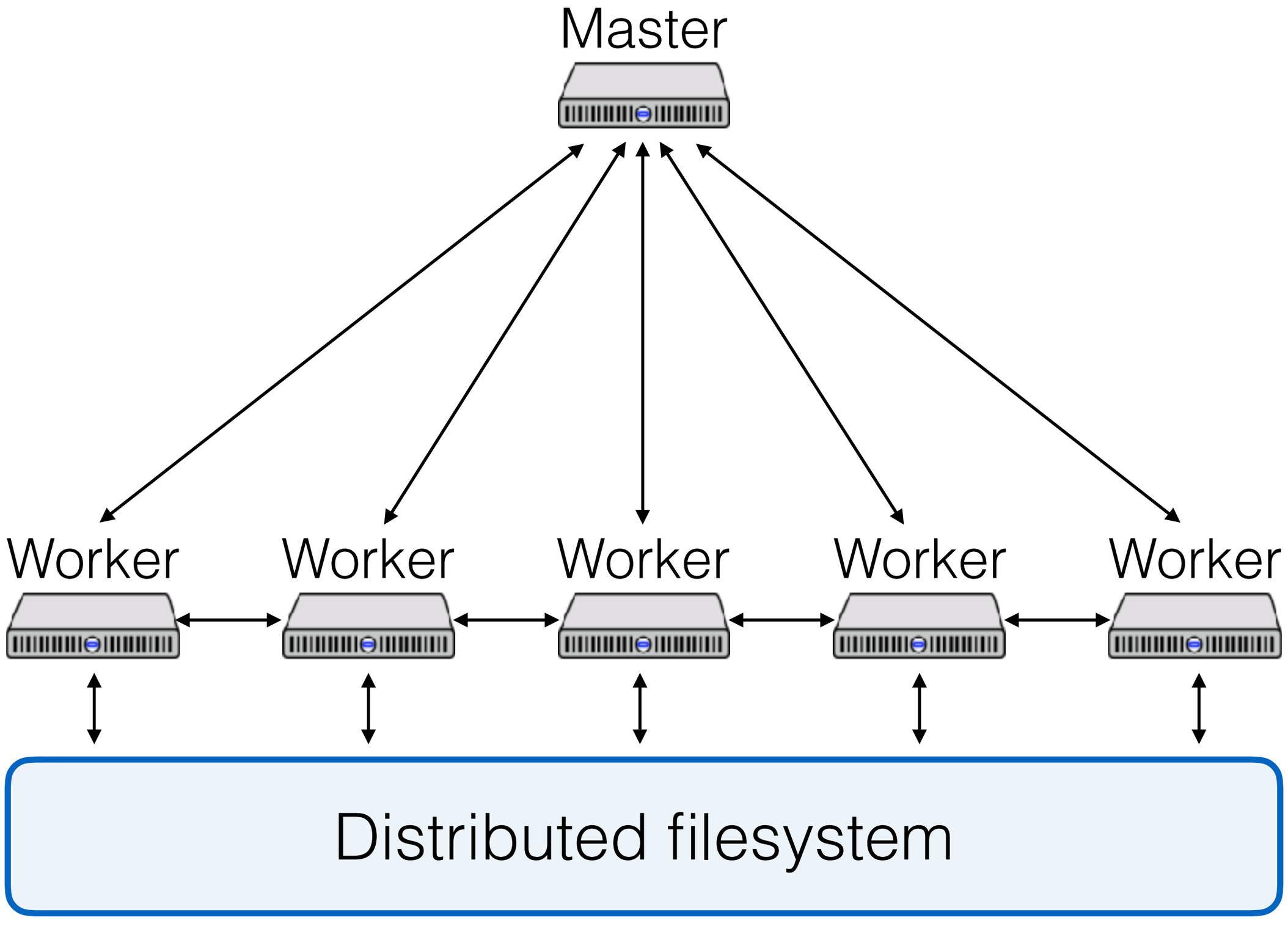
Goals:

- Run on large amount of data
- Run in parallel
- Tolerate failures/slowness at worker nodes

Assume:

- Distributed filesystem
- No master failures

MapReduce Architecture



MapReduce steps

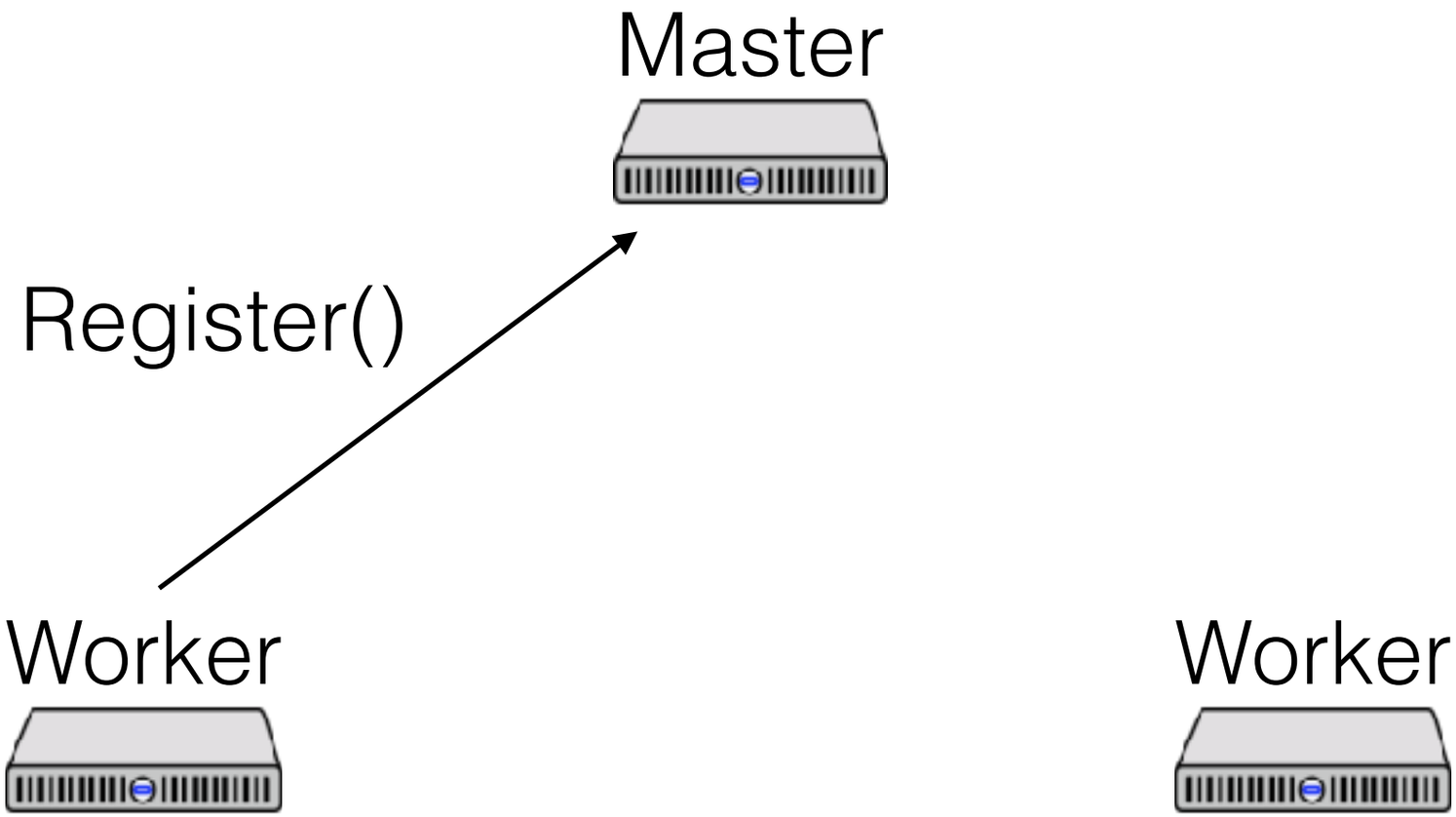
Master

Worker

Worker

Distributed filesystem

MapReduce steps



Distributed filesystem

MapReduce steps

Master

M map tasks, R reduce tasks

Worker

Worker

Distributed filesystem

MapReduce steps

Master

Split input into M
~ fixed-size splits

Worker

Worker

Distributed filesystem

MapReduce steps

Master

Write splits

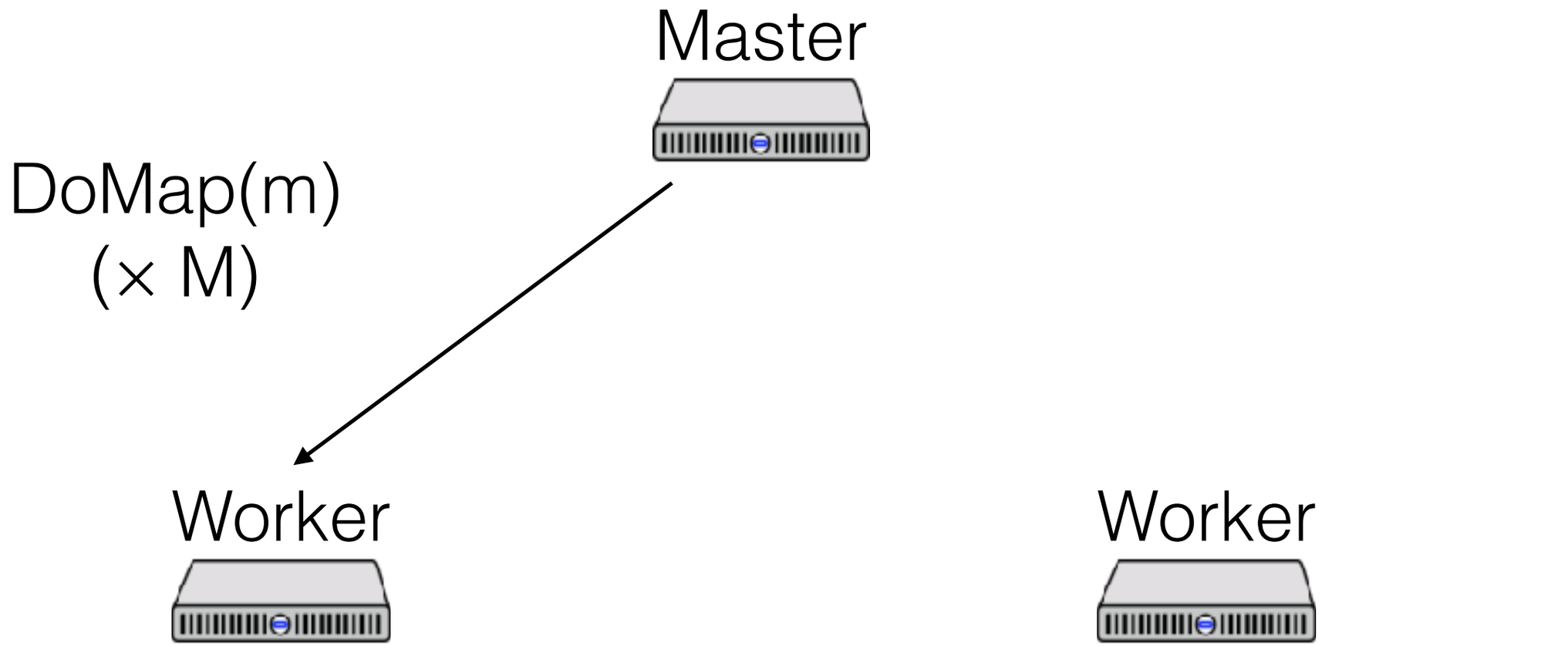
mrtmp.<name>-<m>

Worker

Worker

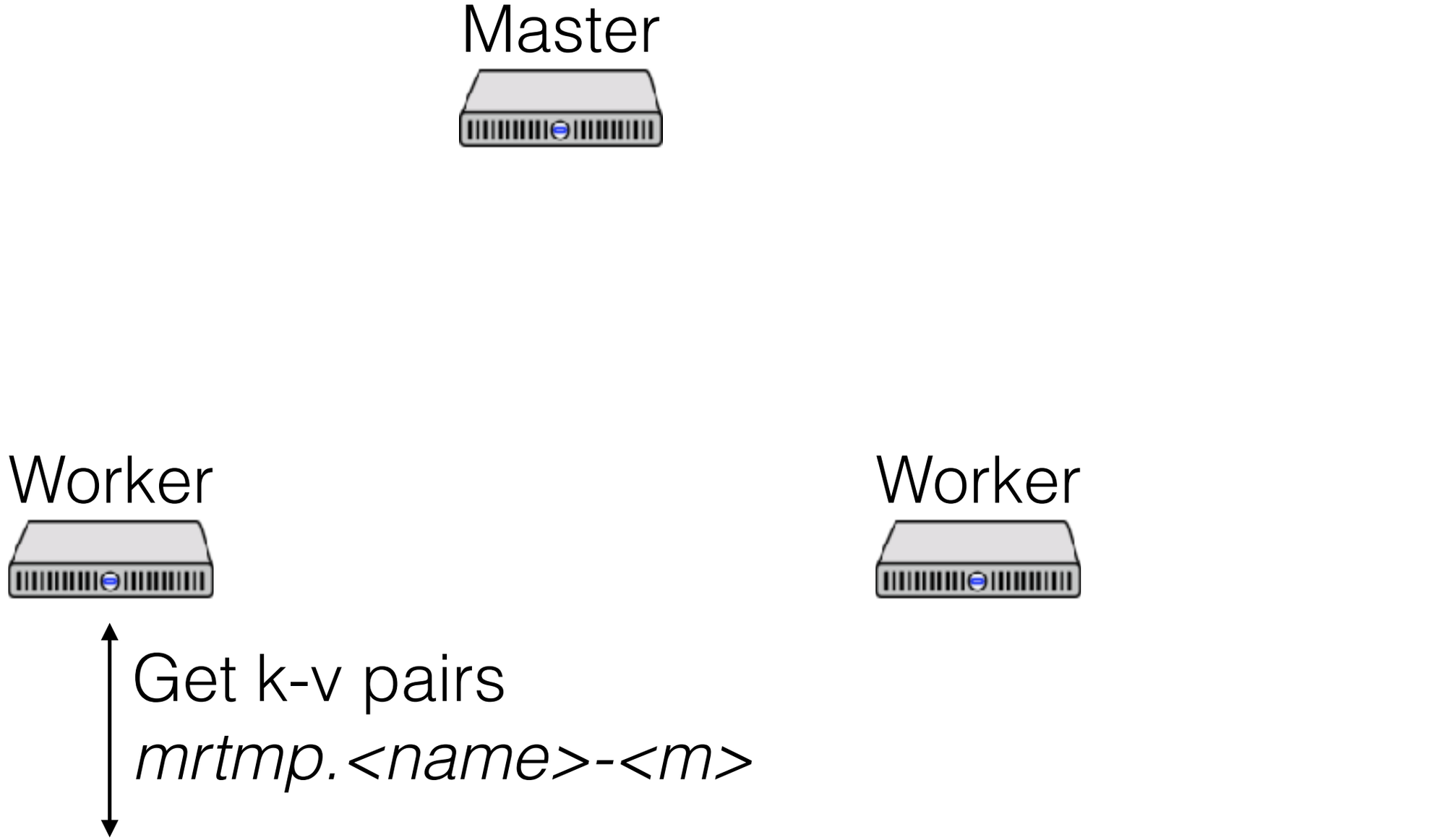
Distributed filesystem

MapReduce steps



Distributed filesystem

MapReduce steps



MapReduce steps

Master

Worker

Call Map() on k-v pairs
Partition results into R “regions”

Worker

Distributed filesystem

MapReduce steps

Master

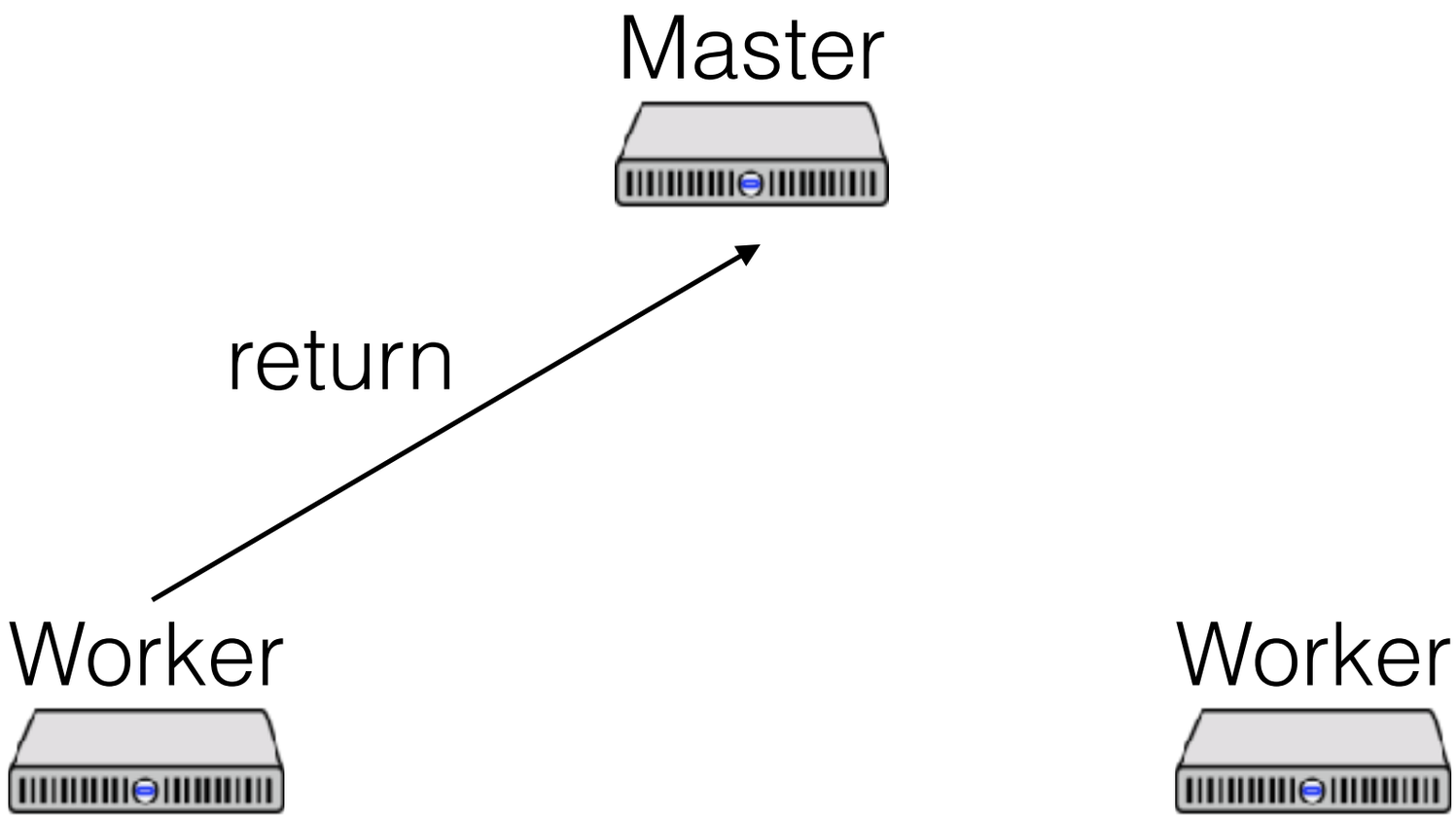
Worker

Worker

Write regions
mrtmp.<name>-<m>-<r>

Distributed filesystem

MapReduce steps



Distributed filesystem

MapReduce steps

Master

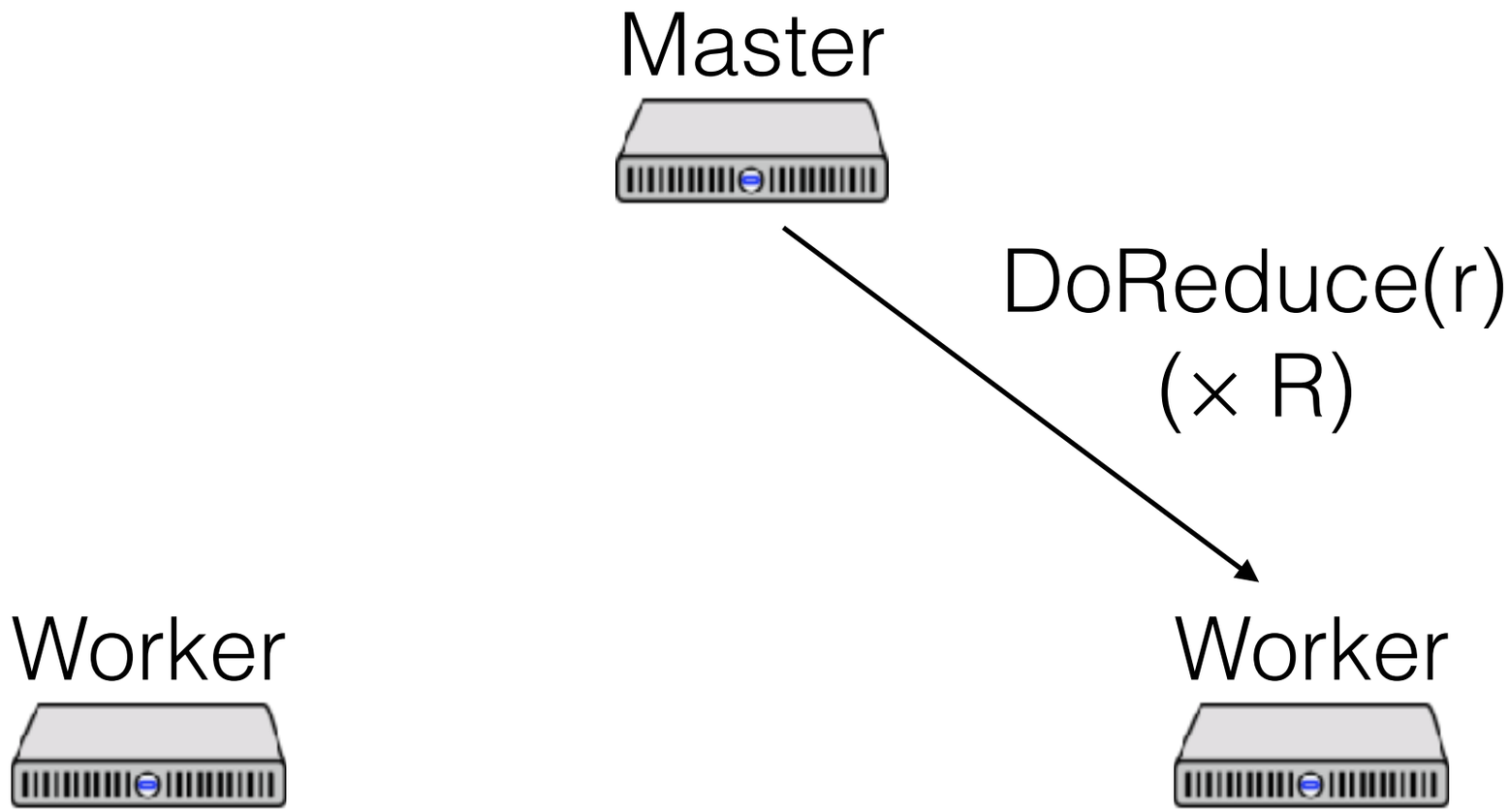
Wait for M Map
tasks to finish

Worker

Worker

Distributed filesystem

MapReduce steps



Distributed filesystem

MapReduce steps

Master

Worker

Worker

Get k-v pairs for r

mrtmp.<name>-<m>-<r>

Distributed filesystem

MapReduce steps

Master

Worker

Worker

Sort pairs
Run Reduce() per key

Distributed filesystem

MapReduce steps

Master

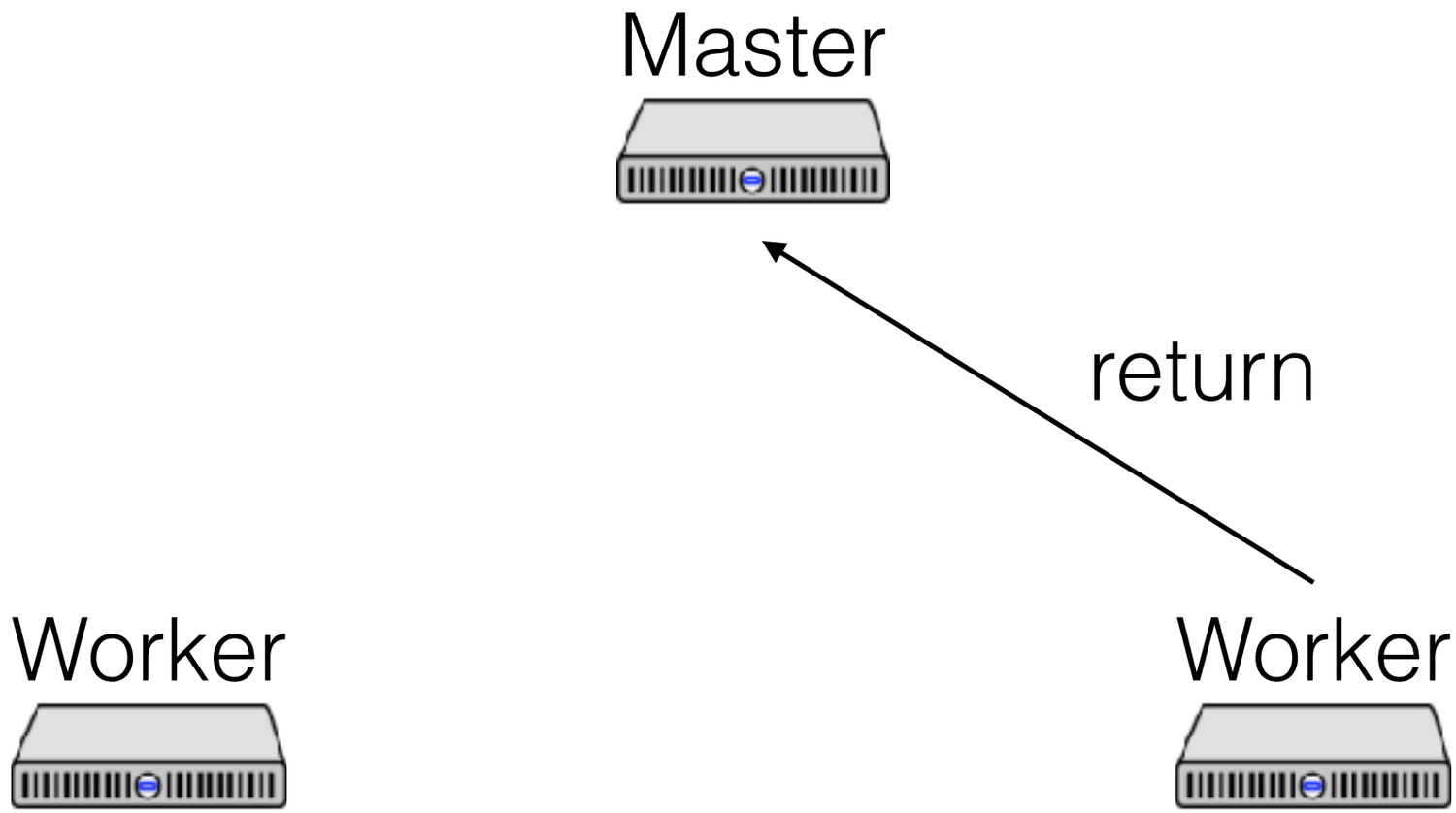
Worker

Worker

Write results

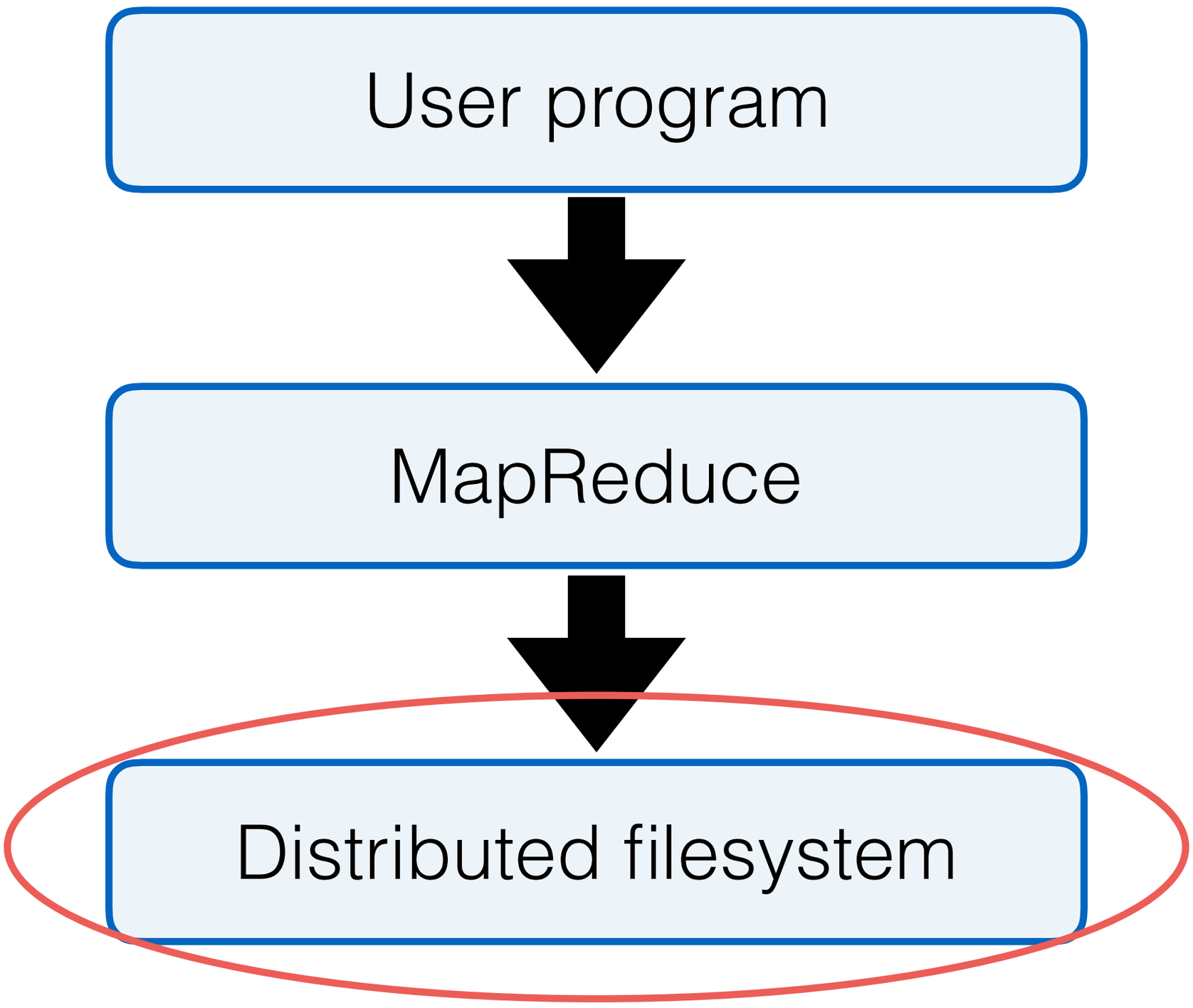
Distributed filesystem

MapReduce steps



Distributed filesystem

Separation of concerns



Distributed filesystem

Will cover later in the quarter!

In the lab, just use the local FS

For now, it's *sort of* a black box

But: why the 64MB default split size?

What if we didn't have a distributed filesystem?

Technical details

- Failures
- Performance
- Optimizations
- Limitations

Handling failure

Basically: just re-run the job

- Handle stragglers, failures in the same way
- If the master fails, have to start over
- How would we handle a master failure?

Why is this easy in MapReduce?

Why wouldn't this be easy in other systems?

- Can I re-run "charge user's credit card?"

Fault-tolerance model

Master never fails

Workers are fail-stop

- Don’t send garbled packets
- Don’t otherwise misbehave
- Can reboot

Packets can be dropped

Performance

How much speedup do we *want* on N servers?

How much speedup do we *expect* on N servers?

What are the bottlenecks?

Optimizations

Data locality is key

- Run Map jobs near data
- Can we run Reduce jobs near data?

Run Reduce function on each Map node's results

- "Combiner" function in the paper
- When can we do this?

Limitations

What problems doesn’t MR solve?

DeWitt/Stonebraker critique

1. A giant step backward in the programming paradigm for large-scale data intensive applications

2. A sub-optimal implementation, in that it uses brute force instead of indexing

3. Not novel at all: represents a specific implementation of well known techniques developed nearly 25 years ago

4. Missing most of the features that are routinely included in current DBMS

5. Incompatible with all of the tools DBMS users have come to depend on

Lab 1

Linked from the course website now!

Due next Friday (April 7), 9:00pm

Turn-in procedure:

- Dropbox on course site
- One partner turns in code
- Both partners turn in **brief** writeup
- Writeup: ~ how long it took, ~ which parts you did

Lab 1

Three parts:

- Implement word count
- Implement naive MapReduce master
- Handle worker failures

Some simplifications w.r.t the paper:

- Map takes strings, not k/v pairs
- Runs locally, so no separation btw local/global FS
- No partial failures (no file-write issues)

Lab 1

Partly a warm-up exercise: learn Go, etc.

Go tutorial section tomorrow

Some general hints next lecture

Have fun!

Discussion

What's the deal with master failure?

Why is atomic rename important?

Why not store intermediate results in RAM?

- Apache Spark

Aren't some Reduce jobs much larger?

What about infinite loops?

Why does novelty matter?