Distributed Systems

COS 418: *Distributed Systems*
Lecture 1

Mike Freedman

Case Study: MapReduce

(Data-parallel programming at scale)

Application: Word Count

```
SELECT count(word) FROM data
    GROUP BY word
```

```
cat data.txt
    |  tr -s '[[:punct:][:space:]]' '\n'
    |  sort  |  uniq -c
```

Using partial aggregation

1. Compute word counts from individual files
2. Then merge intermediate output
3. Compute word count on merged outputs

Using partial aggregation

1. In parallel, send to worker:
   - Compute word counts from individual files
   - Collect result, wait until all finished
2. Then merge intermediate output
3. Compute word count on merged intermediates

MapReduce: Programming Interface

`map(key, value) -> list(<k', v'>)`

- Apply function to (key, value) pair and produces set of intermediate pairs

`reduce(key, list<value>) -> <k', v'>`

- Applies aggregation function to values
- Outputs result

MapReduce: Programming Interface

```
map(key, value):
   for each word w in value:
      EmitIntermediate(w, "1");


reduce(key, list(values):
   int result = 0;
   for each v in values:
      result += ParseInt(v);
   Emit(AsString(result));
```

MapReduce: Optimizations

`combine(list<key, value>) -> list<k,v>`

- Perform partial aggregation on mapper node:

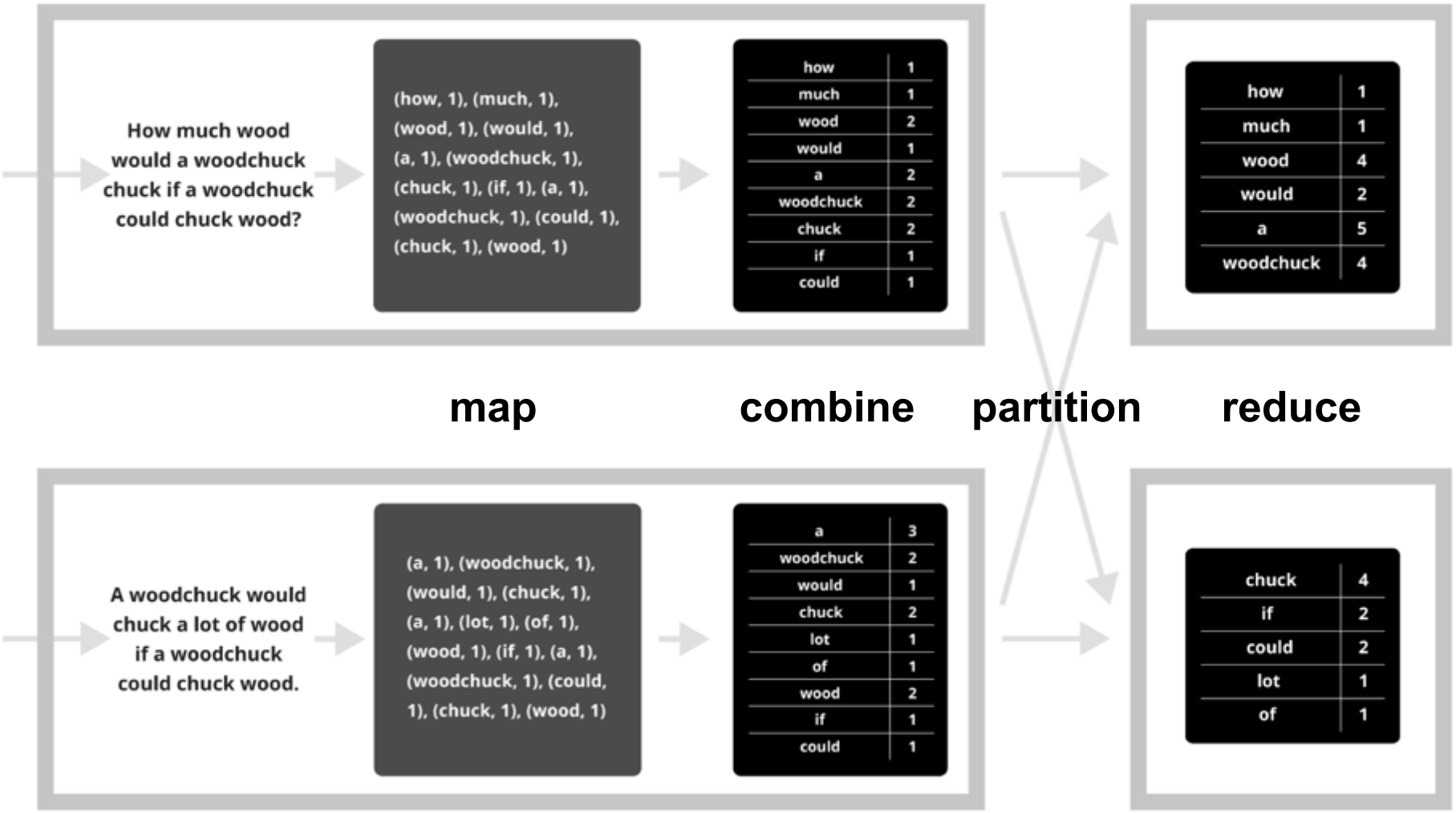
  <the, 1>, <the, 1>, <the, 1> → <the, 3>

- reduce() should be commutative and associative

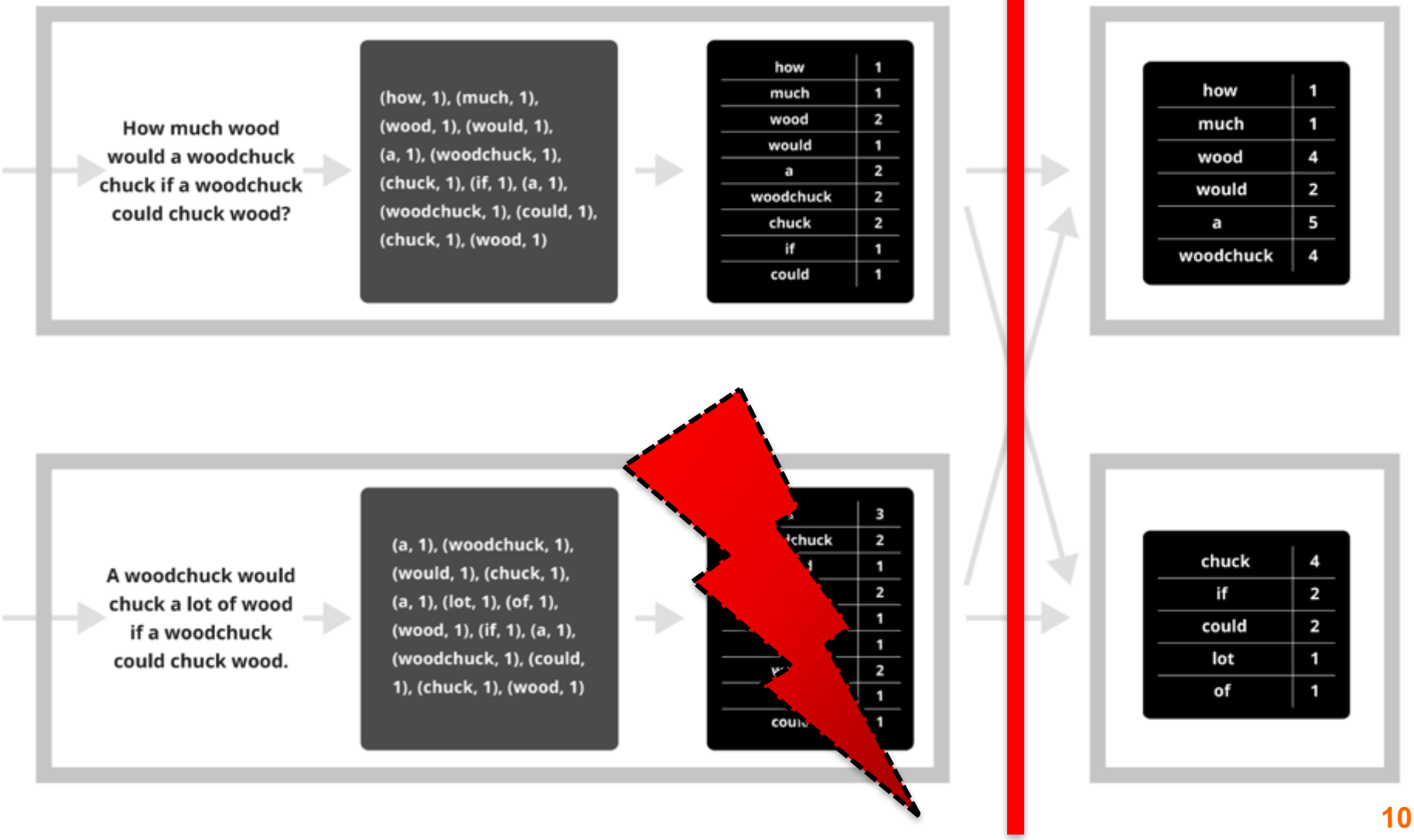
`partition(key, int) -> int`

- Need to aggregate intermediate vals with same key
- Given n partitions, map key to partition $0 \leq i < n$
- Typically via hash(key) mod n

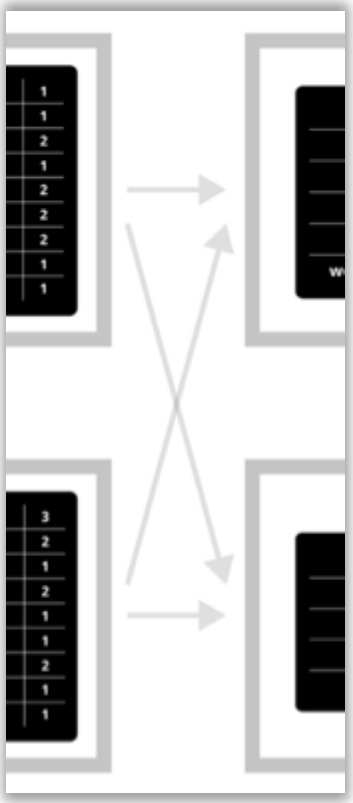
Putting it together…



Synchronization Barrier



Fault Tolerance in MapReduce

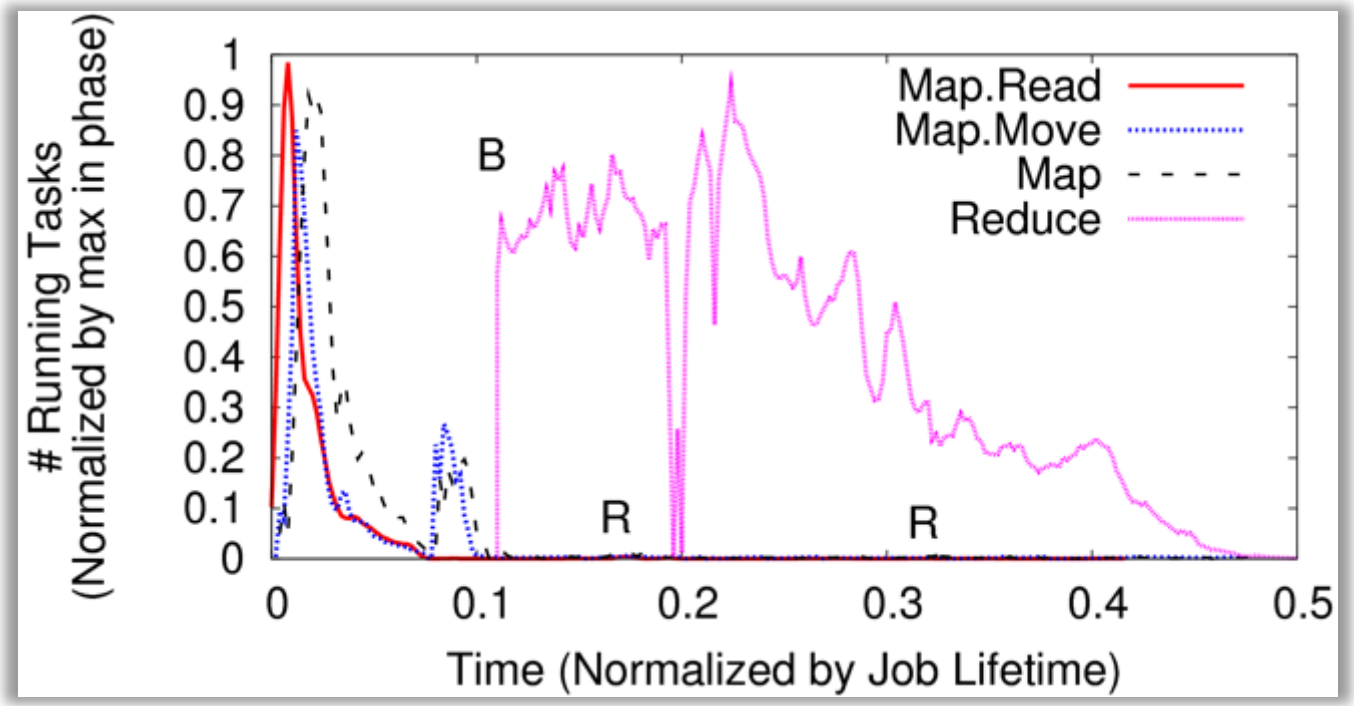


- Map worker writes intermediate output to local disk, separated by partitioning. Once completed, tells master node.
- Reduce worker told of location of map task outputs, pulls their partition's data from each mapper, execute function across data
- Note:
  - "All-to-all" shuffle b/w mappers and reducers
  - Written to disk ("materialized") b/w *each* stage

Fault Tolerance in MapReduce

- Master node monitors state of system
  - If master failures, job aborts and client notified
- Map worker failure
  - Both in-progress/completed tasks marked as idle
  - Reduce workers notified when map task is re-executed on another map worker
- Reducer worker failure
  - In-progress tasks are reset to idle (and re-executed)
  - Completed tasks had been written to global file system

Straggler Mitigation in MapReduce



- Tail latency means some workers finish late
- For slow map tasks, execute in parallel on second map worker as “backup”, race to complete task