Good morning!

CS 744: MAPREDUCE

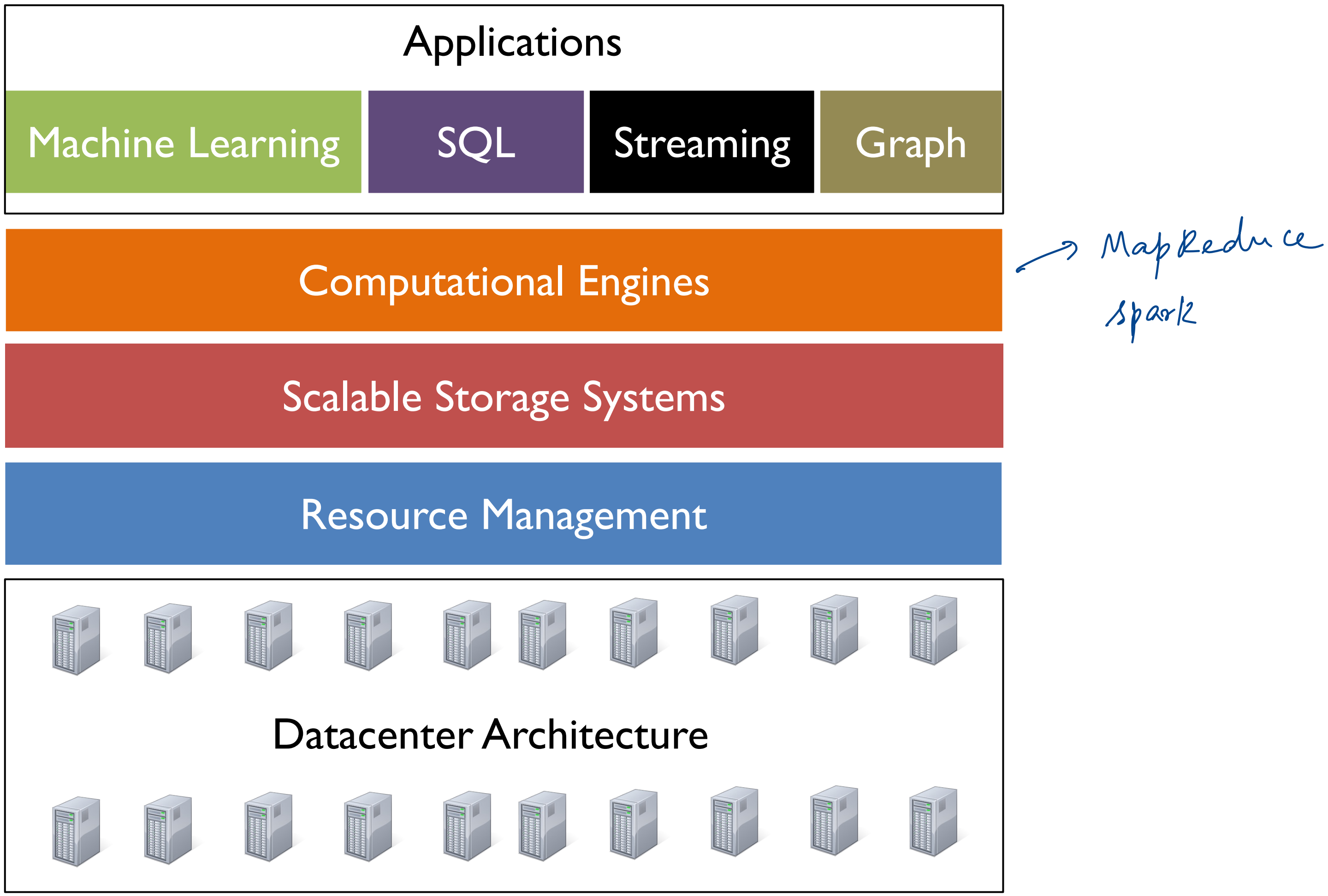
Shivaram Venkataraman

Spring 2024

ANNOUNCEMENTS

- Assignment 1 deliverables
  - Code (comments, formatting)
  - Report
    - Partitioning analysis (graphs, tables, figures etc.)
    - Persistence analysis (graphs, tables, figures etc.)
    - Fault-tolerance analysis (graphs, tables, figures etc.)

INSTALLTION, SPARK UI


Applications
Machine Learning
SQL
Streaming
Graph
Computational Engines
Scalable Storage Systems
Resource Management
Datacenter Architecture
MapReduce
Spark

BACKGROUND: PTHREADS

```
void *myThreadFun(void *vargp)
{
    sleep(1);
    printf("Hello World\n");
    return NULL;
}

int main()
{
    pthread_t thread_id_1, thread_id_2;
    pthread_create(&thread_id_1, NULL, myThreadFun, NULL);
    pthread_create(&thread_id_2, NULL, myThreadFun, NULL);
    pthread_join(thread_id_1, NULL);
    pthread_join(thread_id_2, NULL);
    exit(0);
}
```

Communicate between threads
→ shared variables (memory)
→ Synchronize
→ locks, CVs

Run in parallel
→ multi core machine

BACKGROUND: MPI

User-defined parallelism through rank

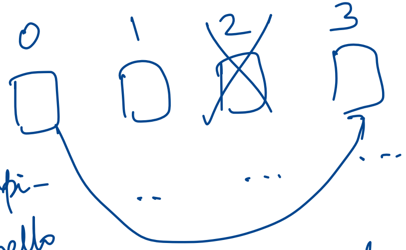
library

```
int main(int argc, char** argv) {
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Print off a hello world message
    printf("Hello world from rank %d out of %d processors\n",
           world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

```
mpirun -n 4 -f host_file ./mpi_hello_world
```

node0 –
node1 –
node2 –
node3 –

Simultaneously



mpi-hello world

MPI_Send
MPI_Recv

Same program on all the machines

MOTIVATION

Build Google Web Search

- Crawl documents, build inverted indexes etc. → I/O intensive

Need for

- automatic parallelization → you dont need to reason about how many tasks in parallel
- network, disk optimization → Commodity machines
- handling of machine failures → Automatically

OUTLINE

- Programming Model
- Execution Overview
- Fault Tolerance
- Optimizations

PROGRAMMING MODEL

Data type: Each record is (key, value) → Structured data

struct {
int ts
IP addr.
....
}

Map function:

$(K_{in}, V_{in}) \rightarrow list(K_{inter}, V_{inter})$

Reduce function:

$(K_{inter}, list(V_{inter})) \rightarrow list(K_{out}, V_{out})$

↓ grouped together

↳ Zero or more outputs

EXAMPLE: WORD COUNT

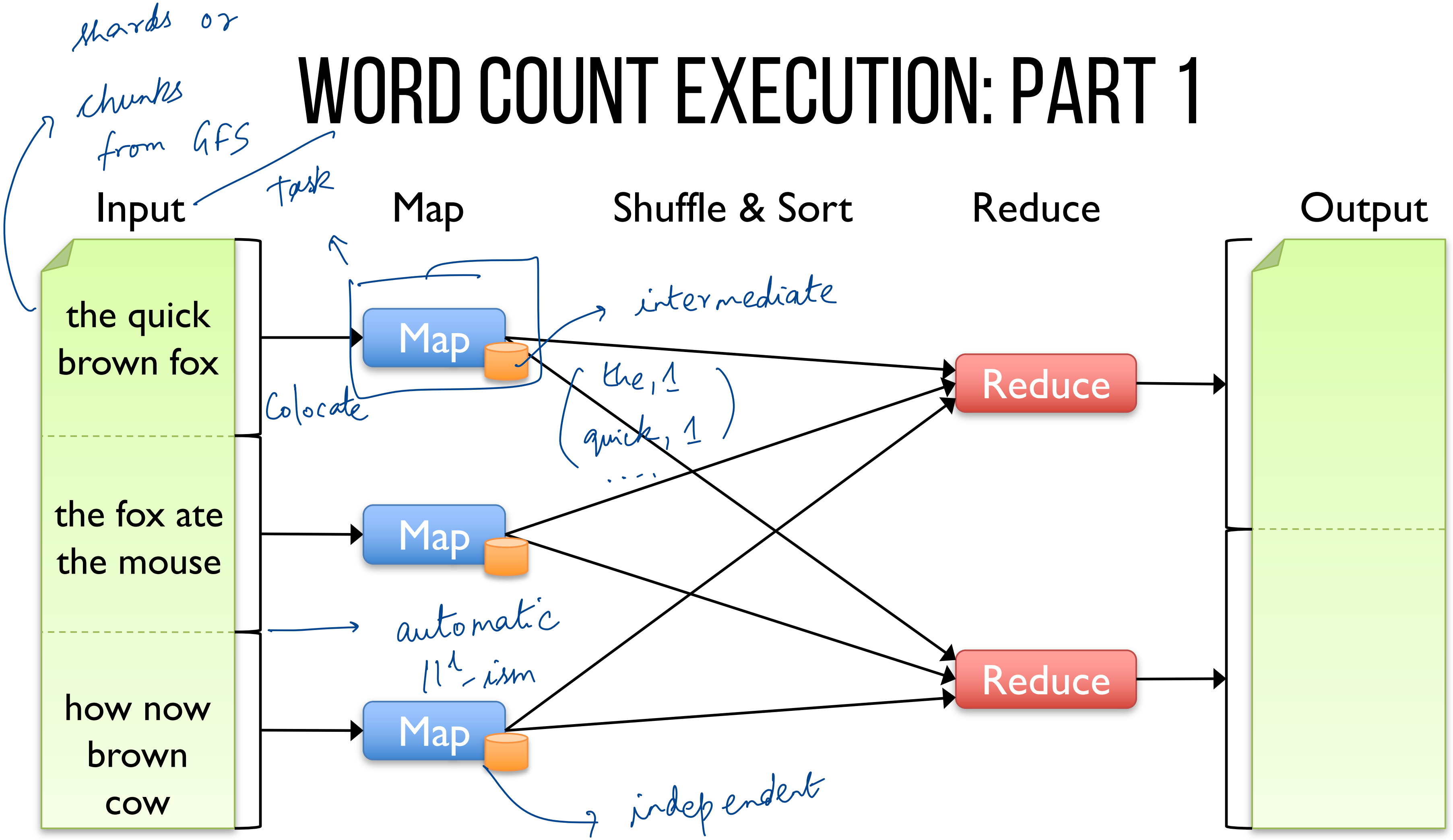
```
def mapper(line):
    for word in line.split():
        output(word, 1)


def reducer(key, values):
    output(key, sum(values))
```

intermediate data

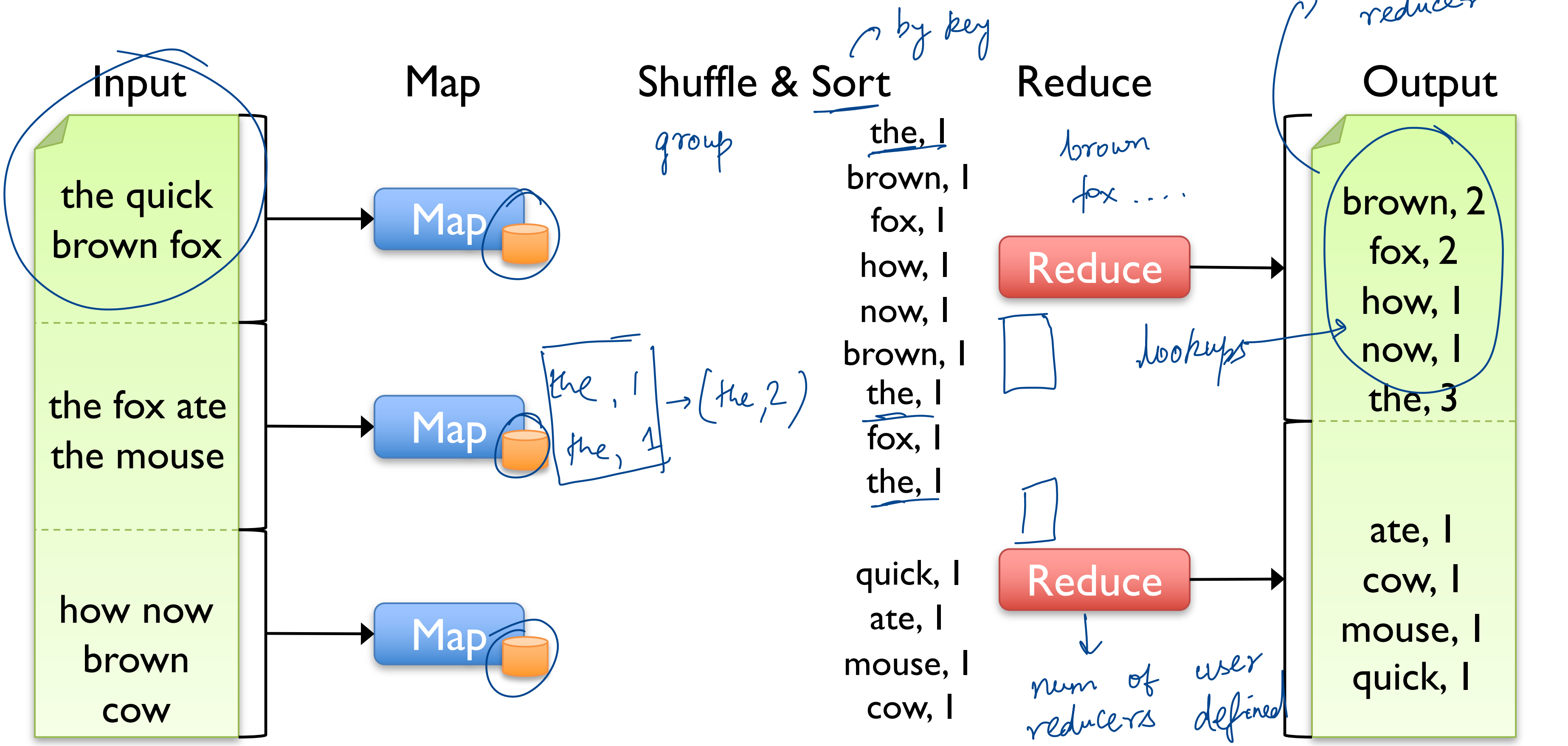
Wisconsin

list (1,1)

WORD COUNT EXECUTION: PART 1



WORD COUNT EXECUTION: PART2



ASSUMPTIONS

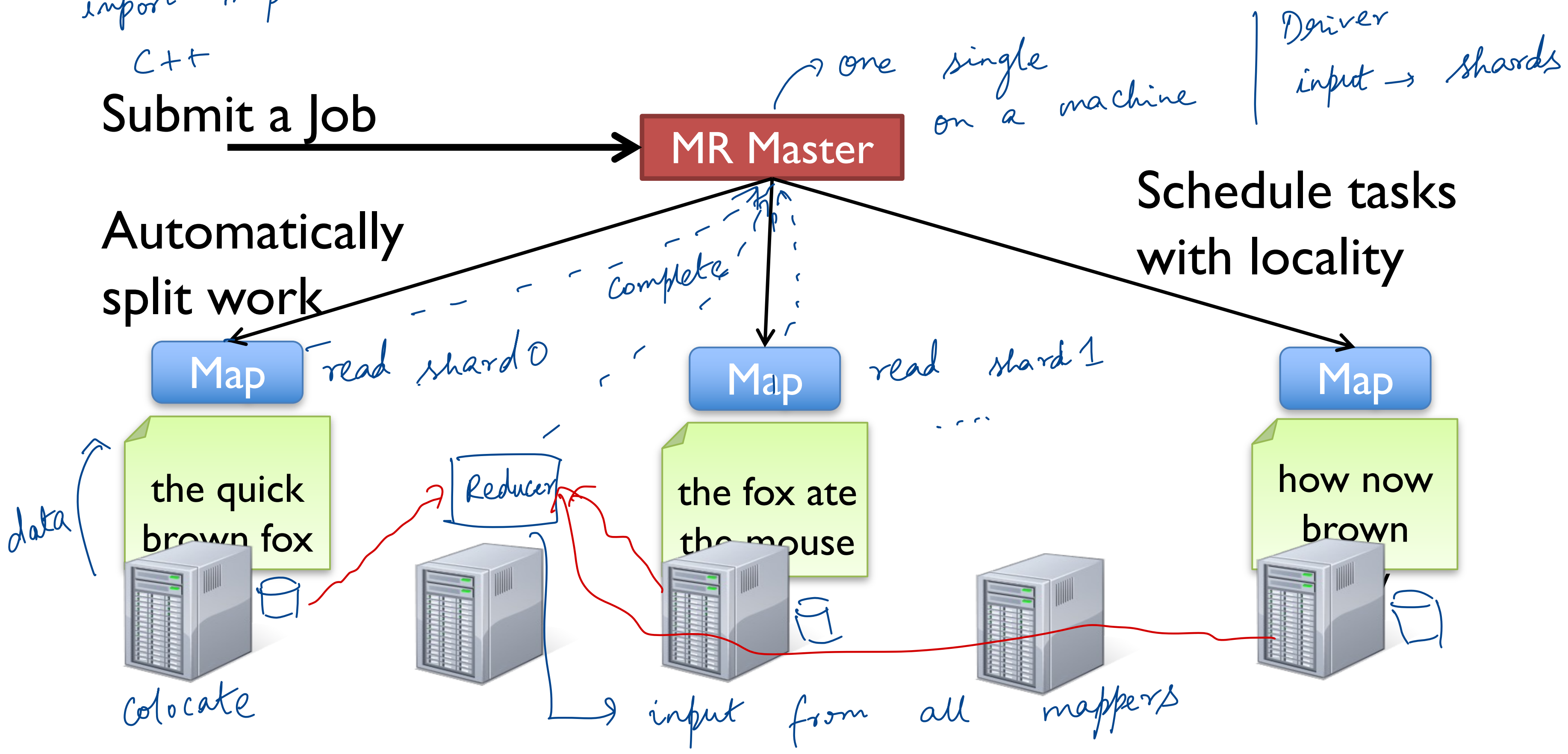
- Assumes data can be processed independently
- Split the data
- load is related to data size
- Reliable storage
  - Input, Output in DFS
  - local disk space

ASSUMPTIONS

1. Commodity networking, less bisection bandwidth
2. Failures are common
3. Local storage is cheap
4. Replicated FS
5. Input is splittable

WORD COUNT EXECUTION

library
import "mapreduce.h"
C++



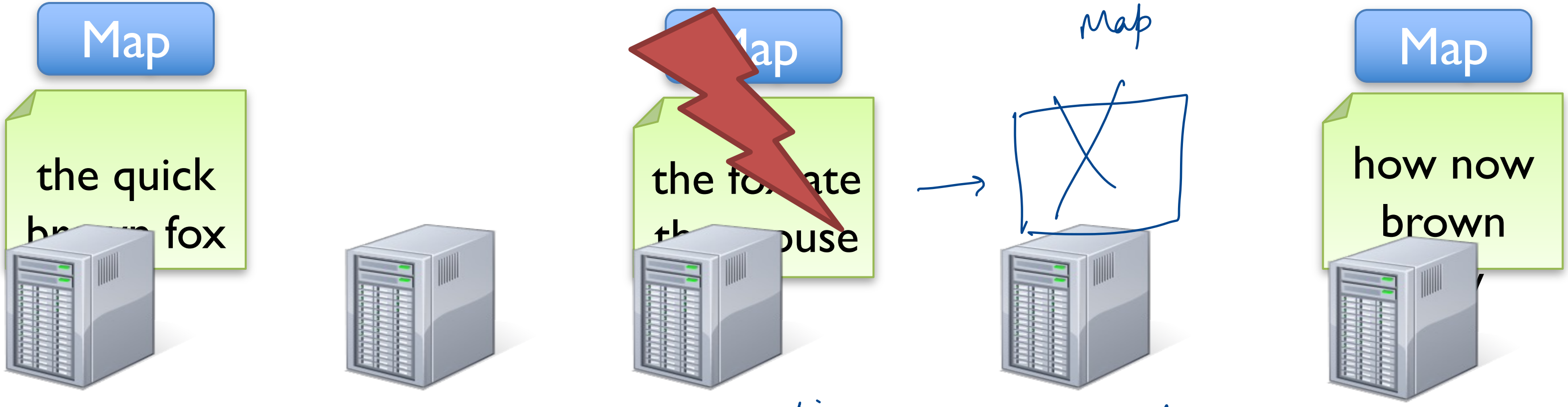
FAULT RECOVERY

If a task crashes:

- Retry on another node
- If the same task repeatedly fails, end the job

buggy code

independent of each other



→ deterministic

→ idempotent

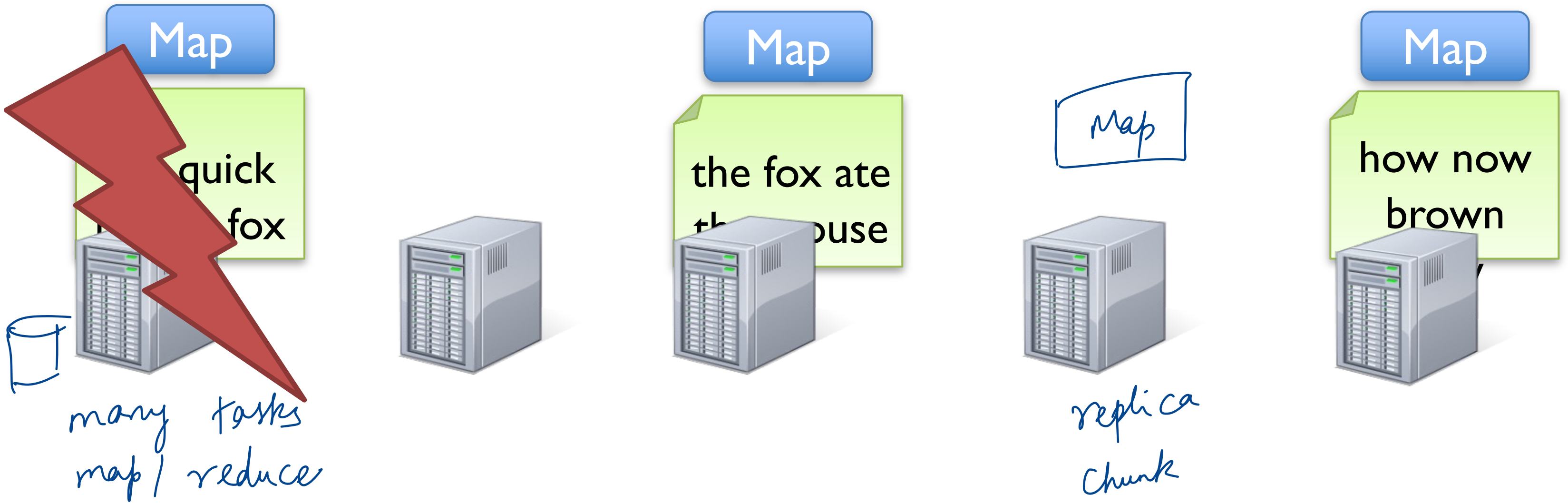
restart

FAULT RECOVERY

If a node crashes:

- Relaunch its current tasks on other nodes

  What about task inputs ? File system replication



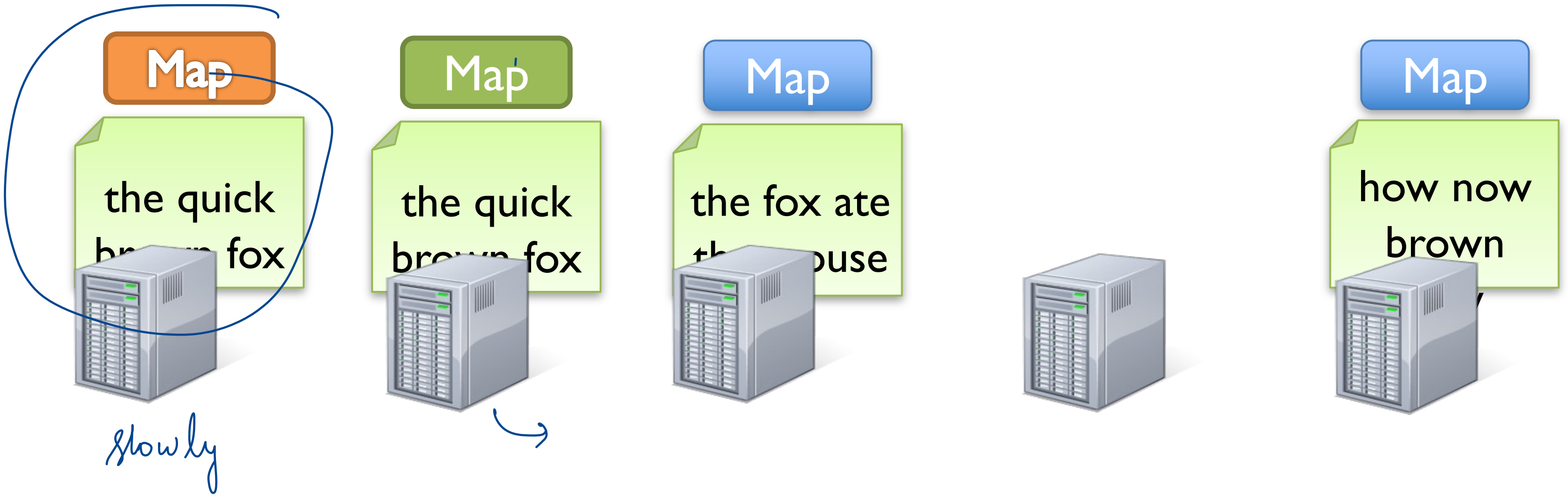
FAULT RECOVERY

bad disk, other processes etc.

If a task is going slowly (straggler):

- Launch second copy of task on another node
- Take the output of whichever finishes first

speculative execution



MORE DESIGN

Master failure

↳ single machine → lower chance

→ fail the job, restart the job!

→ very long running

→ deadline / continuous data

Locality

↳ Map tasks scheduled where input is

Combiner → run reduction on the map side

→ user-defined partitioning function

MAPREDUCE: SUMMARY

- Simplify programming on large clusters with frequent failures
- Limited but general functional API
  - Map, Reduce, Sort
  - No other synchronization / communication
- Fault recovery, straggler mitigation through retries

Sort benchmark

→ Terasort

DISCUSSION

https://forms.gle/zLqtVUEYsZXWoYcL6

DISCUSSION

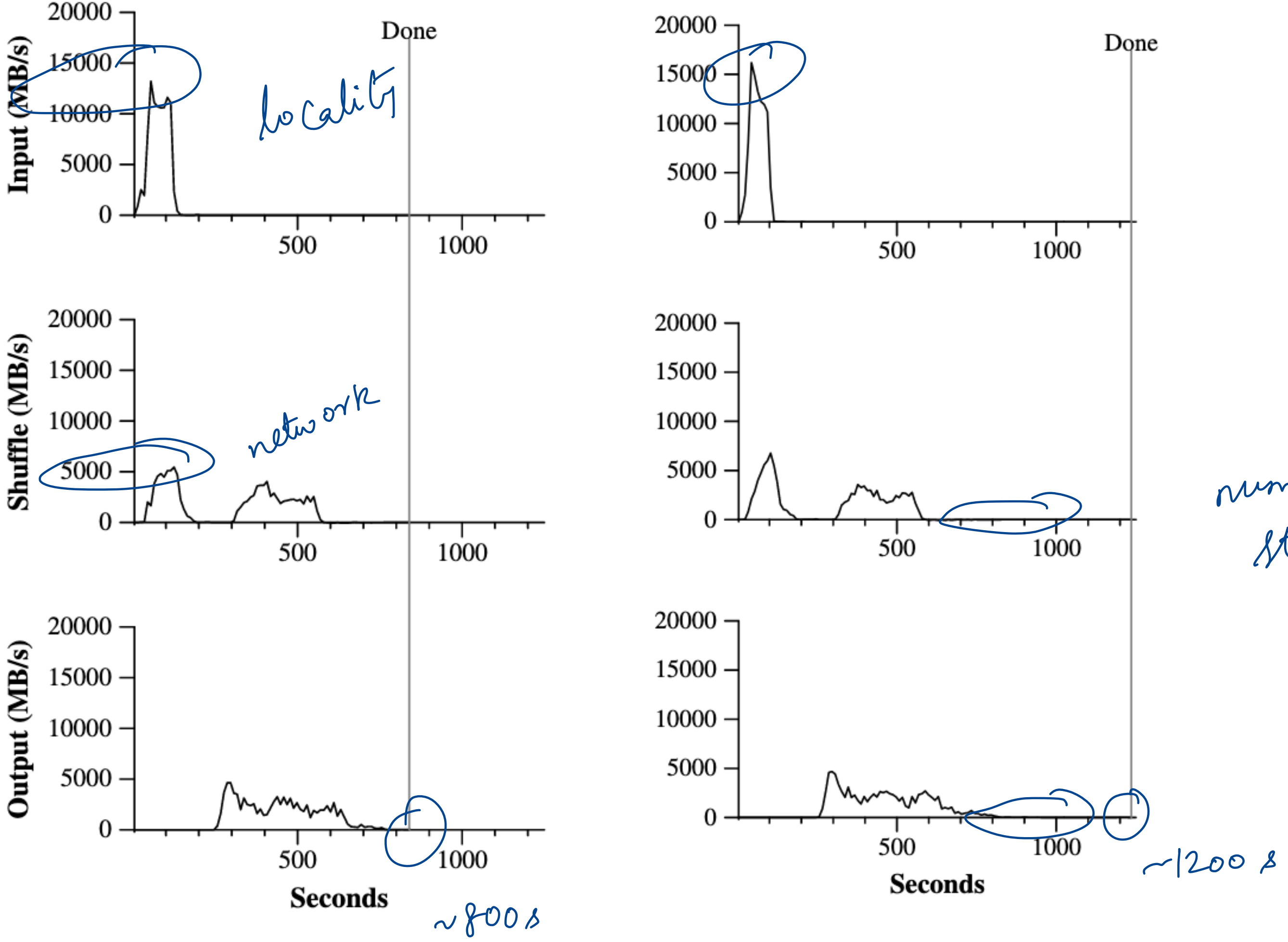
Indexing pipeline where you start with HTML documents. You want to index the documents after removing the most commonly occurring words.

1. Compute most common words.
2. Remove them and build the index.

What are the main shortcomings of using MapReduce to do this?

Two MR jobs → in one pass

more flexible framework?



(a) Normal execution

(b) No backup tasks

number of stragglers is small

→ only a few backup tasks

MapReduce Usage Statistics Over Time

	Aug, '04	Mar, '06	Sep, '07	Sep, '09
Number of jobs	29K	171K	2,217K	3,467K
Average completion time (secs)	634	874	395	475
Machine years used	217	2,002	11,081	25,562
Input data read (TB)	3,288	52,254	403,152	544,130
Intermediate data (TB)	758	6,743	34,774	90,120
Output data written (TB)	193	2,970	14,018	57,520
Average worker machines	157	268	394	488

Jeff Dean, LADIS 2009

NEXT STEPS

- Next lecture: Spark
- Assignment 1: Use Piazza!