

# CS 744: MAPREDUCE

Shivaram Venkataraman

Spring 2025

# ANNOUNCEMENTS

## Assignment I deliverables

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

# Applications

Machine Learning

SQL

Streaming

Graph

Computational Engines

Scalable Storage Systems

Resource Management



Datacenter Architecture



# 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);
}
```

# BACKGROUND: MPI

```
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
```

# MOTIVATION

Build Google Web Search

- Crawl documents, build inverted indexes etc.

Need for

- automatic parallelization
- network, disk optimization
- handling of machine failures

# OUTLINE

- Programming Model
- Execution Overview
- Fault Tolerance
- Optimizations

# PROGRAMMING MODEL

Data type: Each record is (key, value)

**Map** function:

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

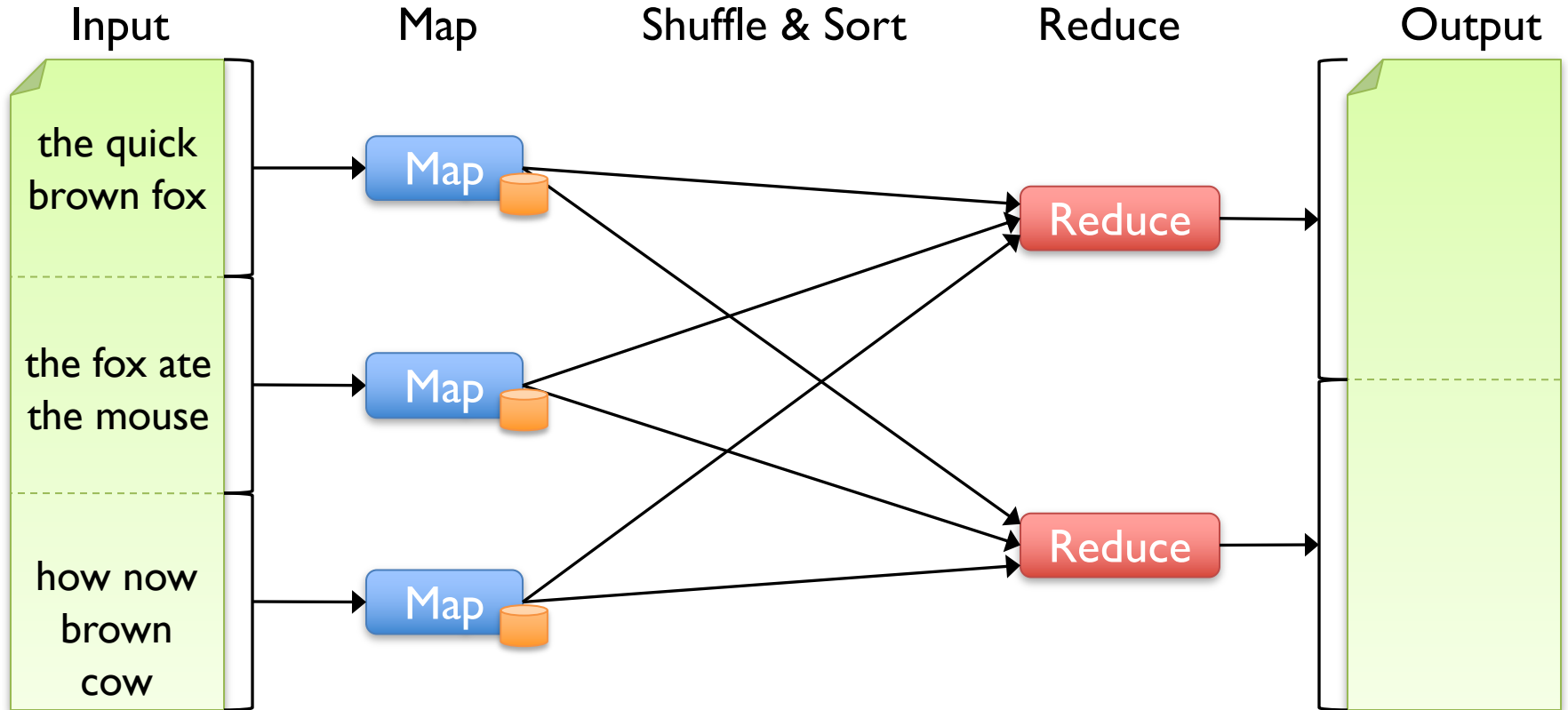
**Reduce** function:

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

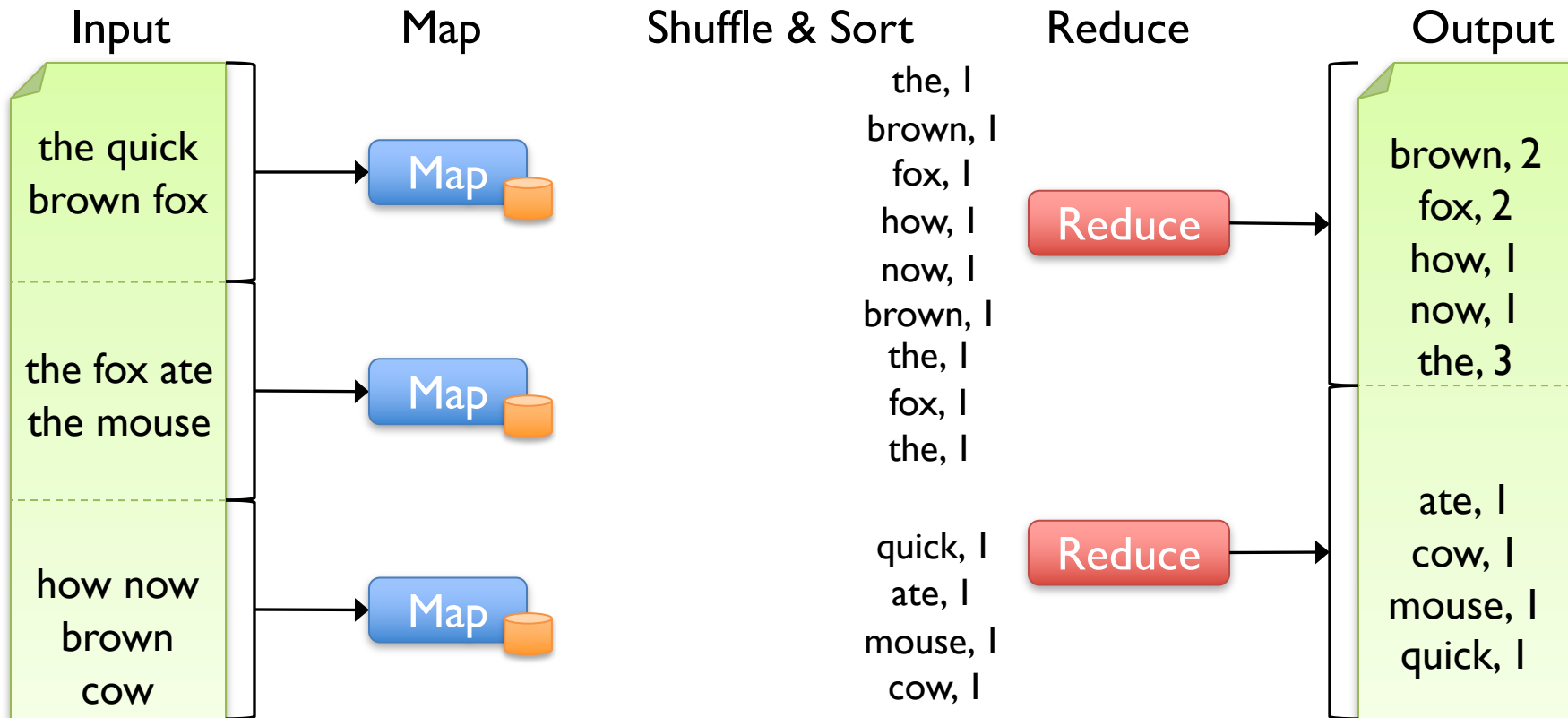
# EXAMPLE: WORD COUNT

```
def mapper(line):  
    for word in line.split():  
        output(word, 1)  
  
def reducer(key, values):  
    output(key, sum(values))
```

# WORD COUNT EXECUTION: PART 1



# WORD COUNT EXECUTION: PART 2

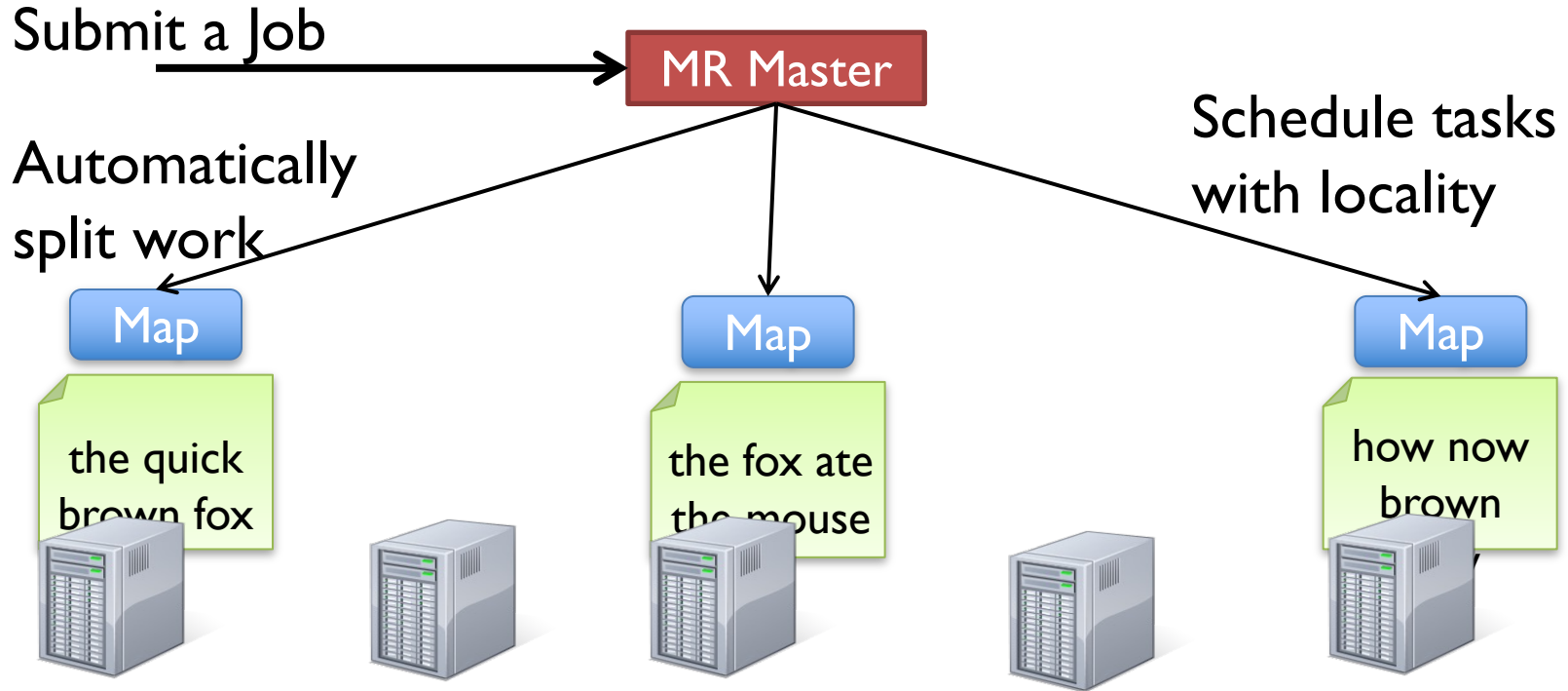


# ASSUMPTIONS

# ASSUMPTIONS

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

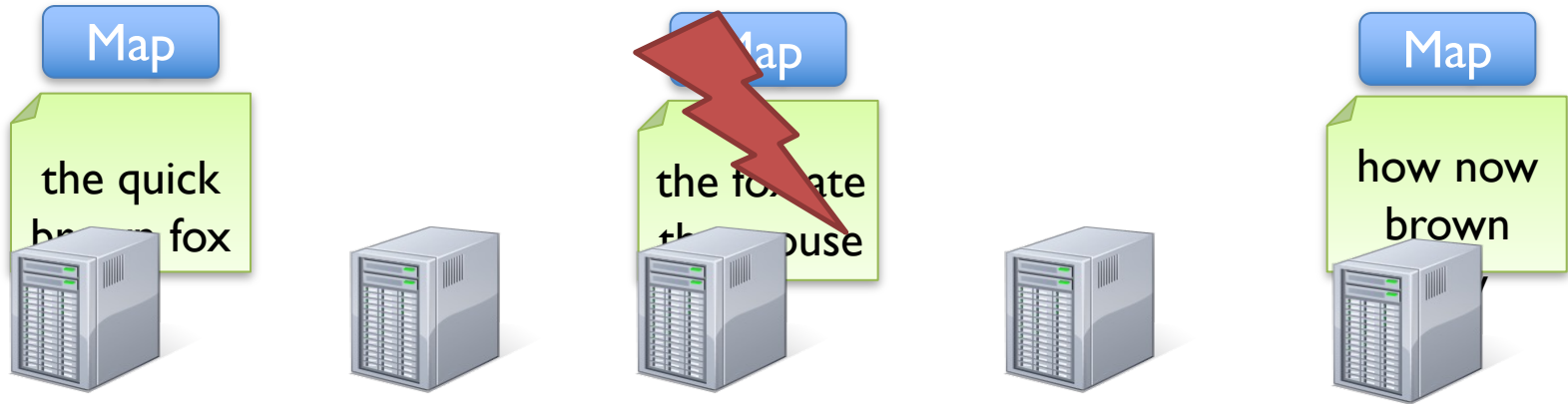
# WORD COUNT EXECUTION



# FAULT RECOVERY

If a task crashes:

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

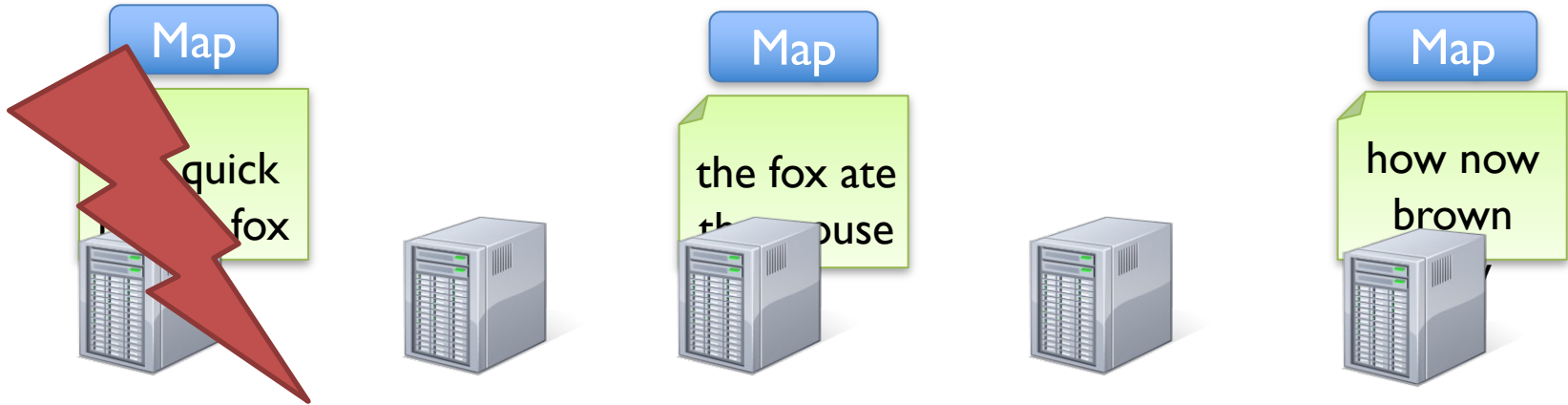


# FAULT RECOVERY

If a node crashes:

- Relaunch its current tasks on other nodes

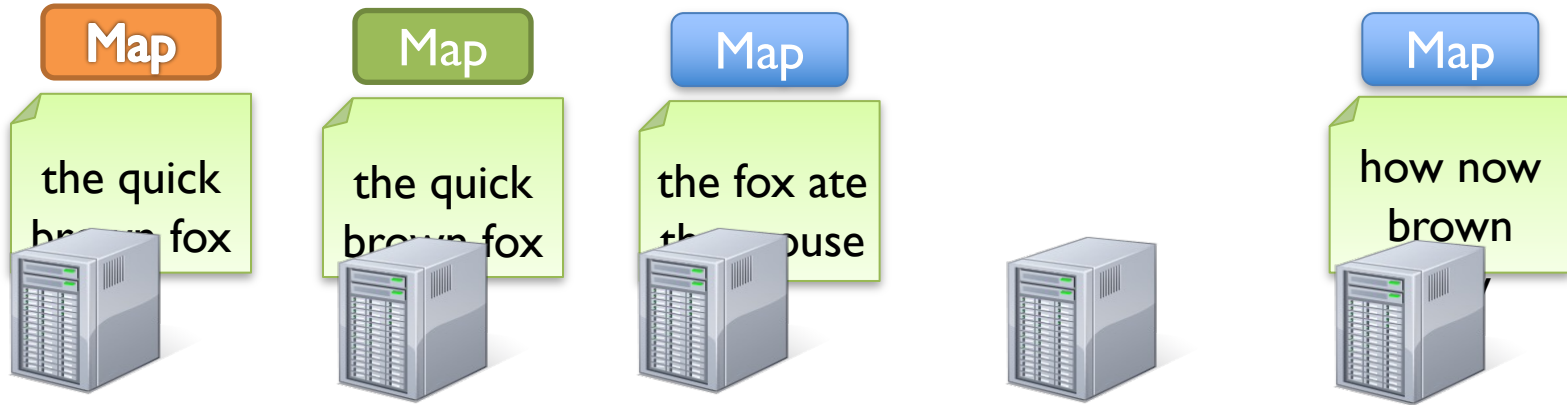
What about task inputs ? File system replication



# FAULT RECOVERY

If a task is going slowly (straggler):

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



# MORE DESIGN

Master failure

Locality

# 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



# DISCUSSION

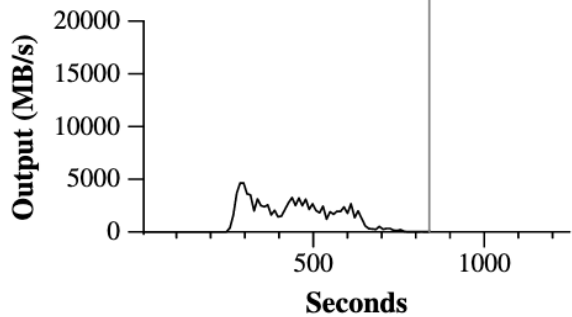
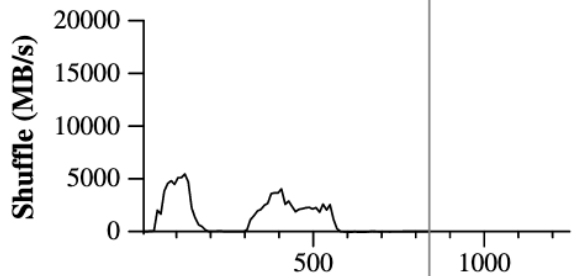
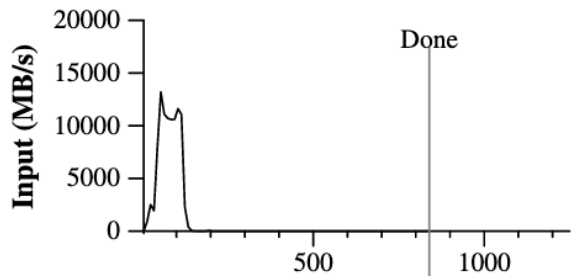
<https://forms.gle/VuwU7pdTWE5q92Su8>

# DISCUSSION

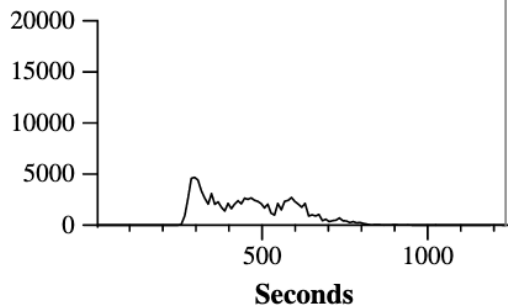
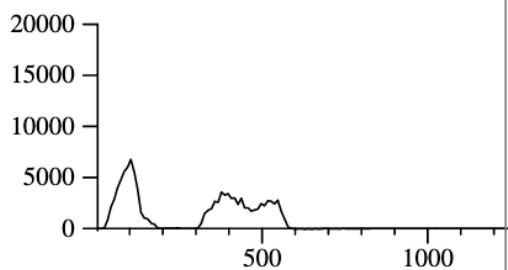
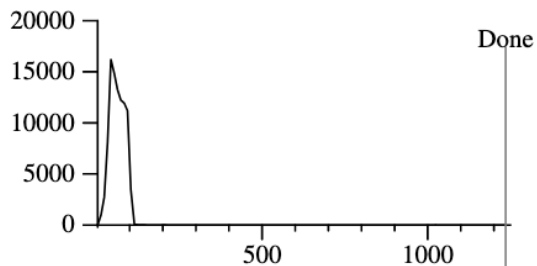
Indexing pipeline where you start with HTML documents. You want to index the documents after removing the most commonly occurring words.

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

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



(a) Normal execution



(b) No backup tasks

**WHAT HAPPENED NEXT**

# 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

# DRYAD

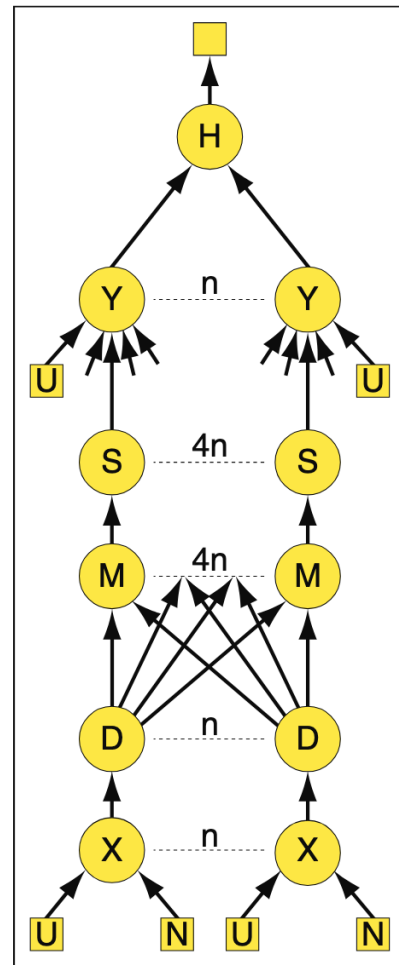
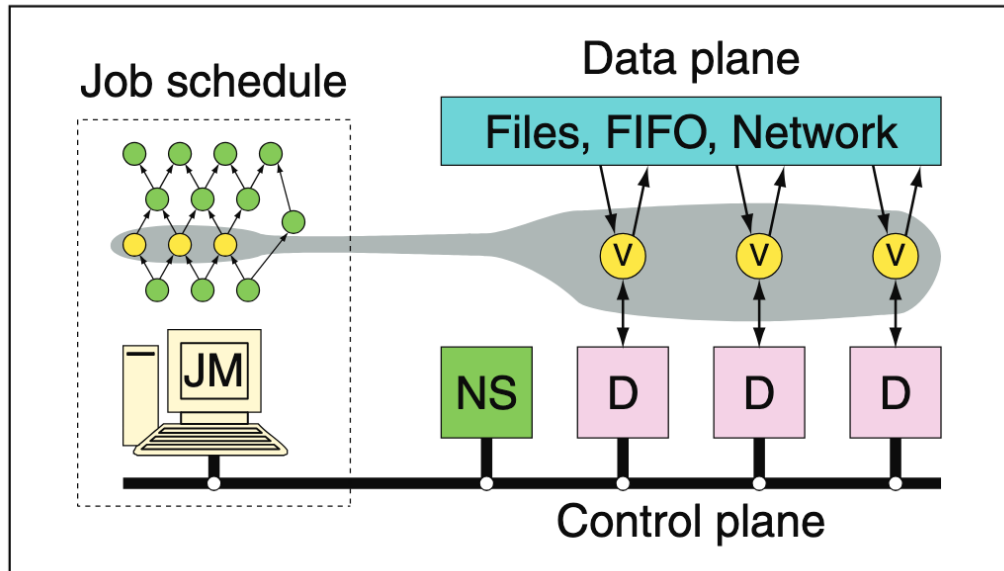


Figure 2: The communication graph for an SQL query.

# NEXT STEPS

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