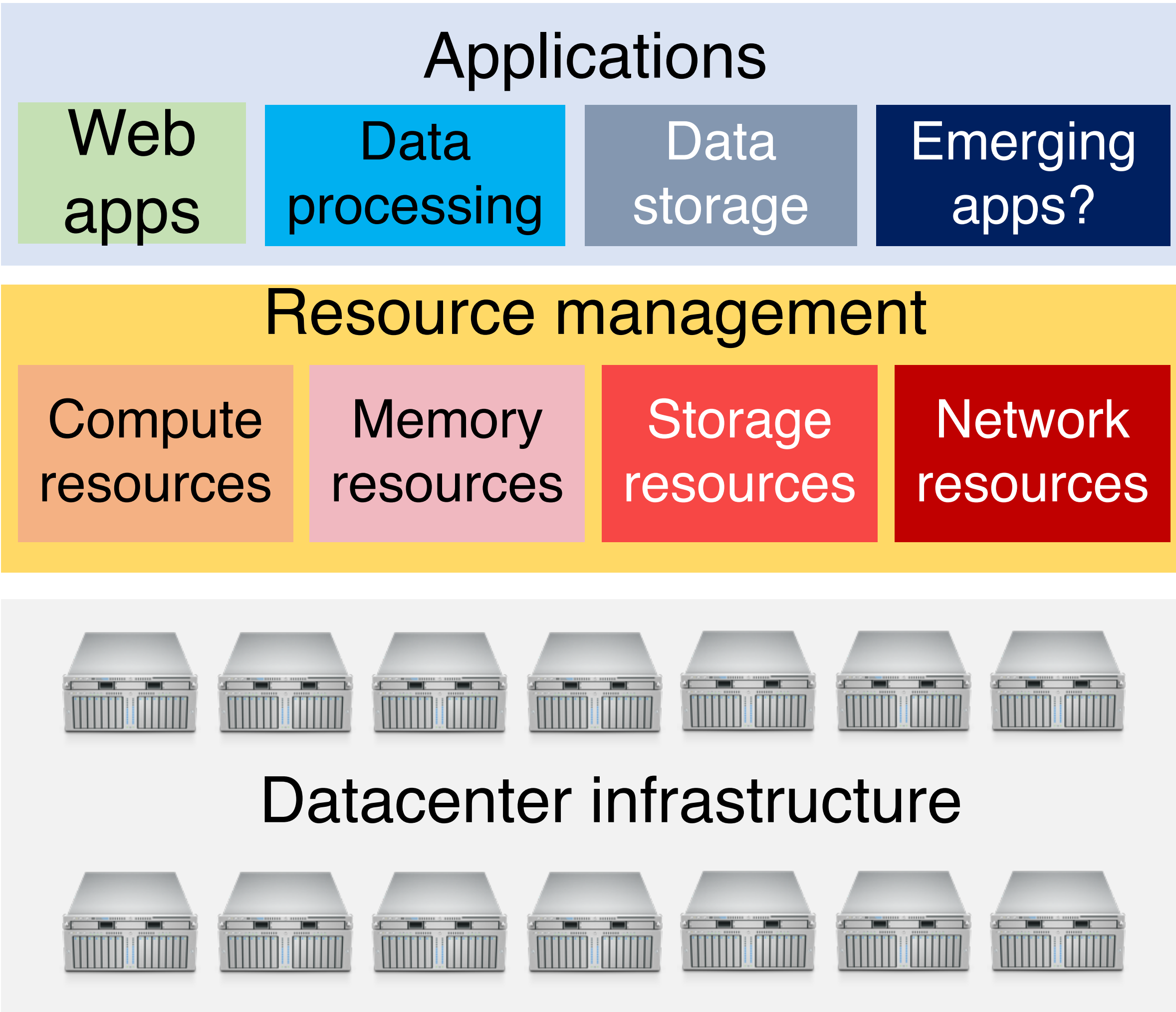
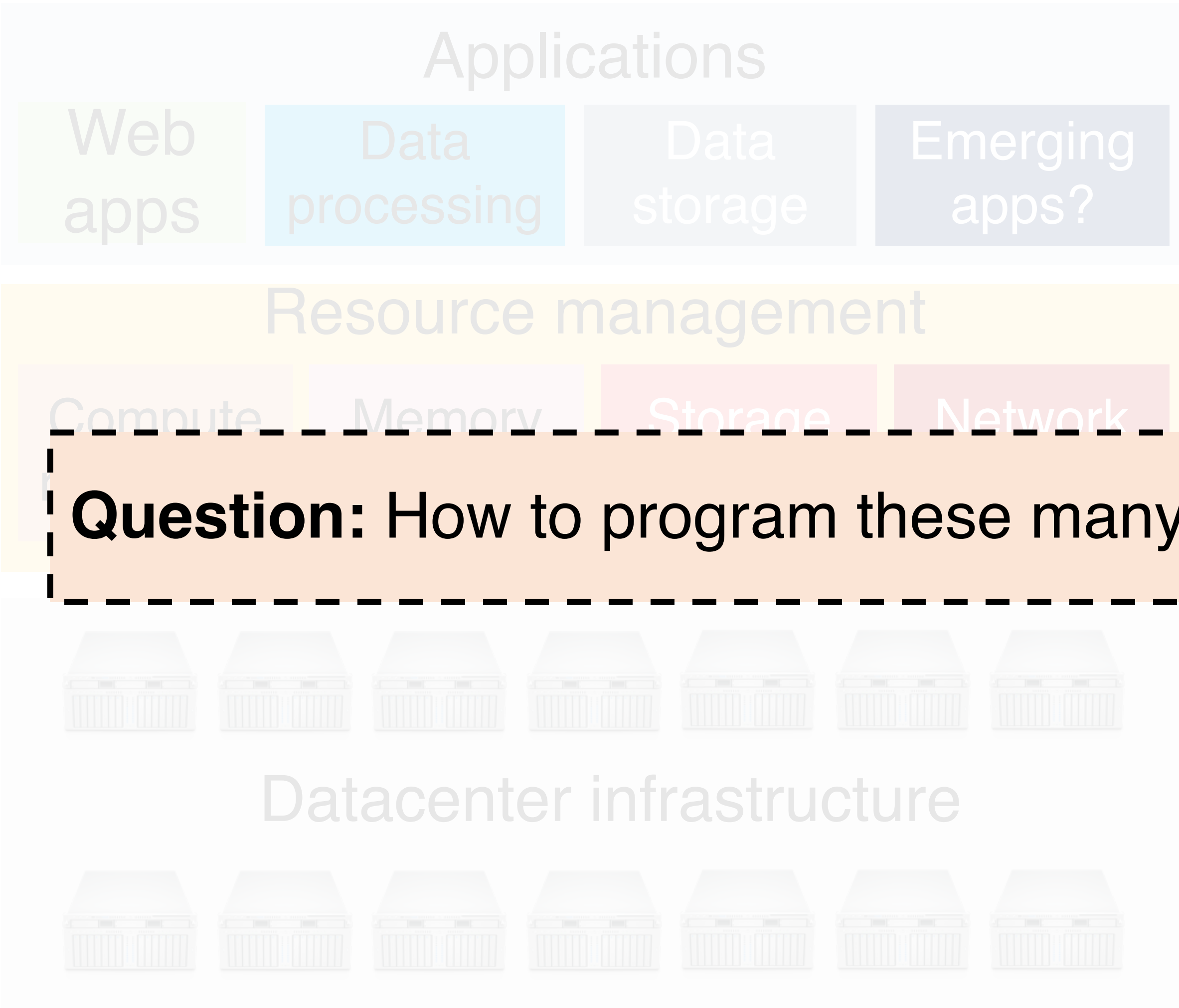
MapReduce

CS 475: Concurrent & Distributed Systems (Fall 2021)

Lecture 4

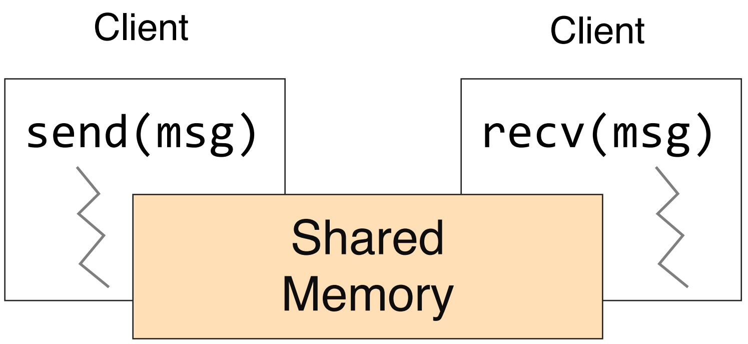
Yue Cheng


Applications
Web apps
Data processing
Data storage
Emerging apps?
Resource management
Compute resources
Memory resources
Storage resources
Network resources
Datacenter infrastructure



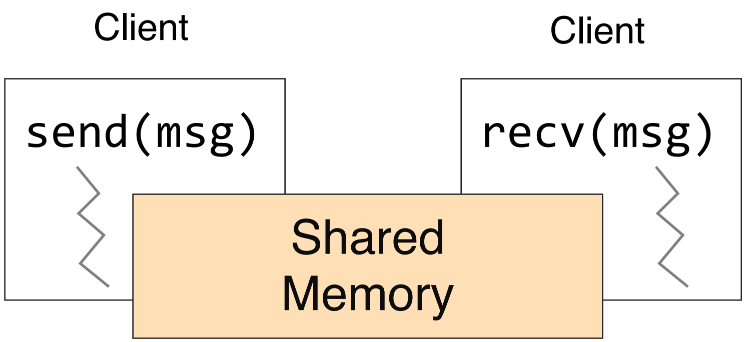
Question: How to program these many computers?

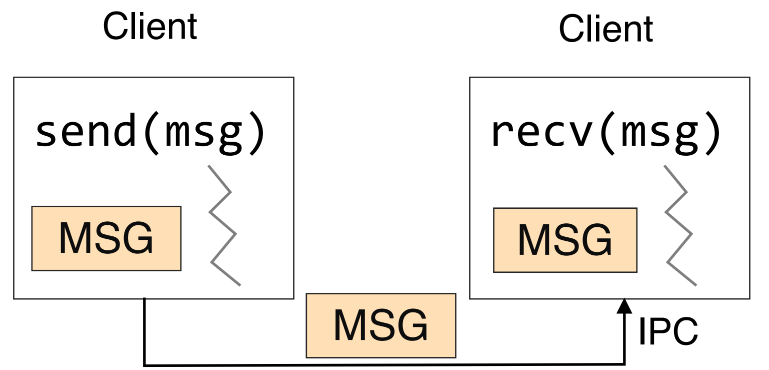
Review: Shared memory



- Shared memory: multiple processes to share data via memory
- Applications must locate and and map shared memory regions to exchange data

Review:
Shared memory vs. Message passing





- Shared memory: multiple processes to share data via memory
- Applications must locate and and map shared memory regions to exchange data

- Message passing: exchange data explicitly via IPC
- Application developers define protocol and exchanging format, number of participants, and each exchange

Review: Shared memory vs. Message passing

- Easy to program; just like a single multi-threaded machines
- Hard to write high perf. apps:
  - Cannot control which data is local or remote (remote mem. access much slower)
- Hard to mask failures

- Message passing: can write very high perf. apps
- Hard to write apps:
  - Need to manually decompose the app, and move data
- Need to manually handle failures

Shared memory: Pthread

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization

- API specifies behavior of the thread library, implementation is up to development of the library

- Common in UNIX (e.g., Linux) OSes

Shared memory: Pthread

```
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);
}
```

Message passing: MPI

- MPI – Message Passing Interface
  - Library standard defined by a committee of vendors, implementers, and parallel programmers
  - Used to create parallel programs based on message passing

- Portable: one standard, many implementations
  - Available on almost all parallel machines in C and Fortran
  - De facto standard platform for the HPC community

Message passing: 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, workld_size);

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

Message passing: MPI

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

```
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, workld_size);

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

MapReduce

The big picture (motivation)

- Datasets are too big to process using a single computer

The big picture (motivation)

- Datasets are too big to process using a single computer
- Good parallel processing engines are rare (back then in the late 90s)

The big picture (motivation)

- Datasets are too big to process using a single computer
- Good parallel processing engines are rare (back then in the late 90s)
- Want a parallel processing framework that:
  - is **general** (works for many problems)
  - is **easy to use** (no locks, no need to explicitly handle communication, no race conditions)
  - can **automatically parallelize** tasks
  - can **automatically handle machine failures**

Context (Google circa 2000)

- Starting to deal with massive datasets
- But also addicted to cheap, unreliable hardware
  - Young company, expensive hardware not practical
- Only a few expert programmers can write distributed programs to process them
  - Scale so large jobs can complete before failures

Context (Google circa 2000)

- Starting to deal with massive datasets
- But also addicted to cheap, unreliable hardware
  - Young company, expensive hardware not practical
- Only a few expert programmers can write distributed programs to process them
  - Scale so large jobs can complete before failures
- **Key question:** how can every Google engineer be imbued with the ability to write parallel, scalable, distributed, fault-tolerant code?
- **Solution:** abstract out the redundant parts
- Restriction: relies on job semantics, so restricts which problems it works for

Application: Word Count

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

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

Deal with multiple files?

1. Compute word counts from individual files

Deal with multiple files?

1. Compute word counts from individual files

2. Then merge intermediate output

Deal with multiple files?

1. Compute word counts from individual files

2. Then merge intermediate output

3. Compute word count on merged outputs

What if the data is too big to fit in one computer?

1. In parallel, send to worker:
   - Compute word counts from individual files
   - Collect results, wait until all finished

What if the data is too big to fit in one computer?

1. In parallel, send to worker:
   - Compute word counts from individual files
   - Collect results, wait until all finished

2. Then merge intermediate output

What if the data is too big to fit in one computer?

1. In parallel, send to worker:
   - Compute word counts from individual files
   - Collect results, wait until all finished

2. Then merge intermediate output

3. Compute word count on merged intermediates

MapReduce: Programming interface

- `map(k1, v1) → list(k2, v2)`
  - Apply function to (`k1, v1`) pair and produce set of intermediate pairs (`k2, v2`)

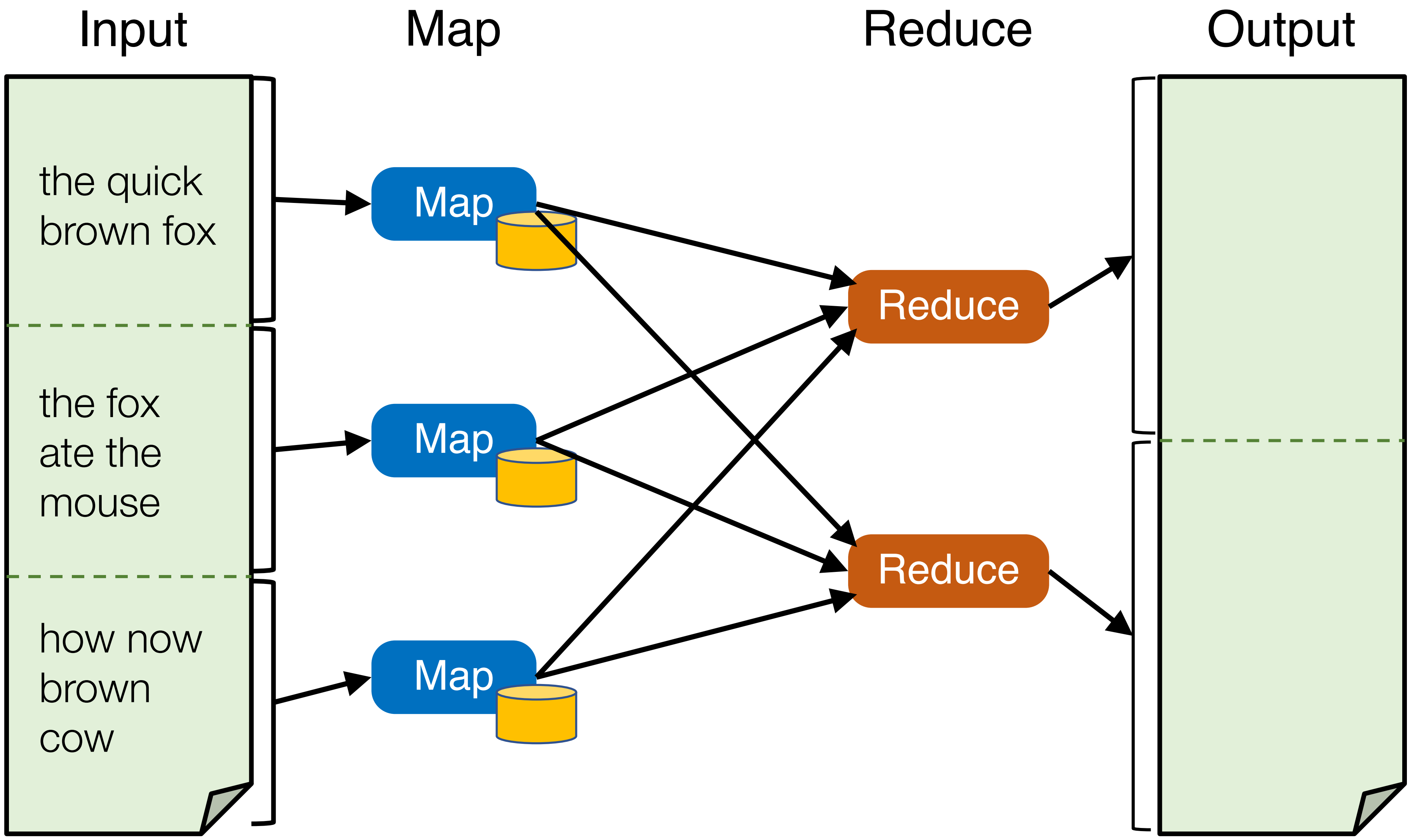
- `reduce(k2, list(v2)) → list(k3, v3)`
  - Apply aggregation (reduce) function to values
  - Output results

MapReduce: Word Count

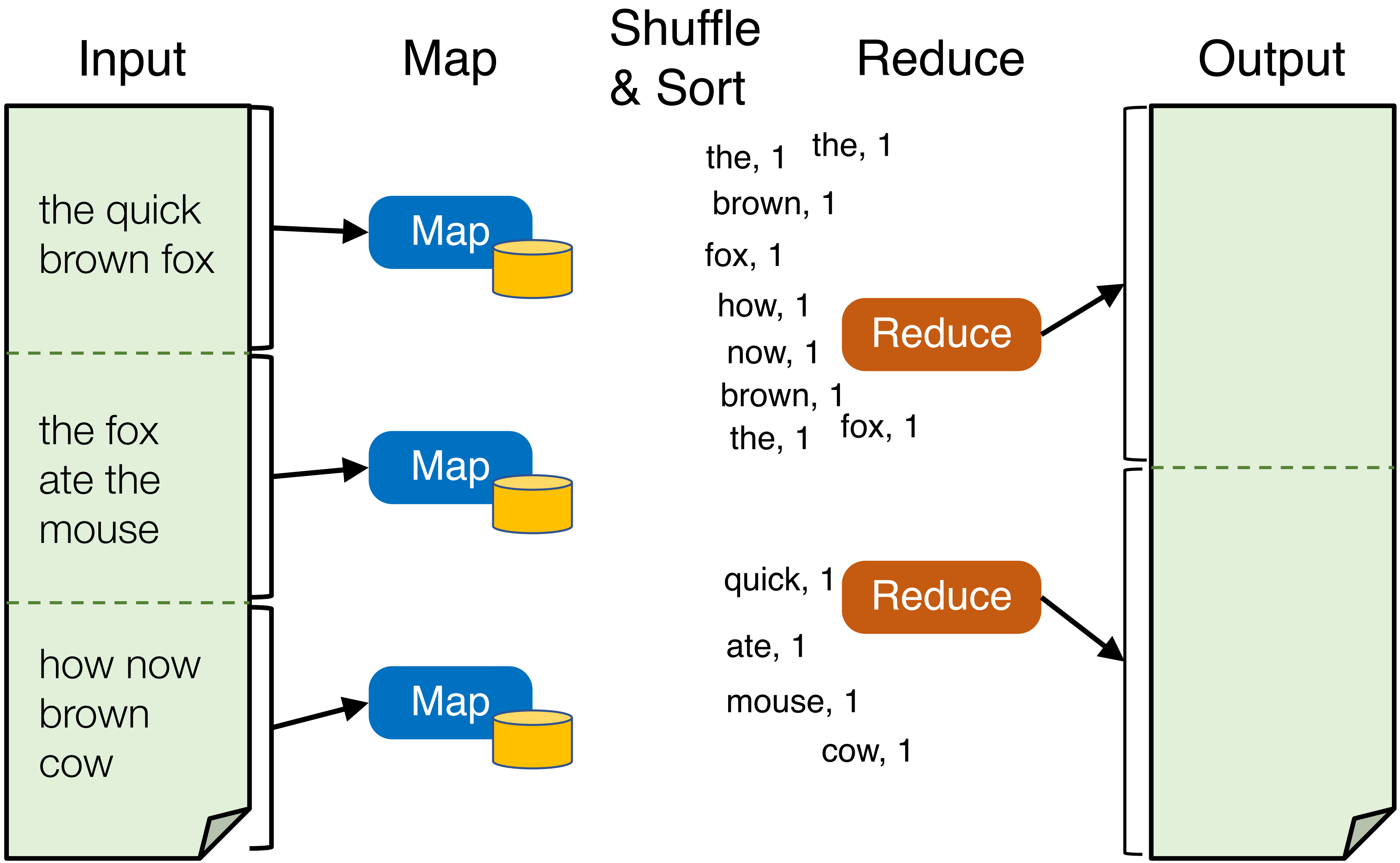
```
map(key, value):
    for each word w in value:
        EmitIntermediate(w, "1");

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

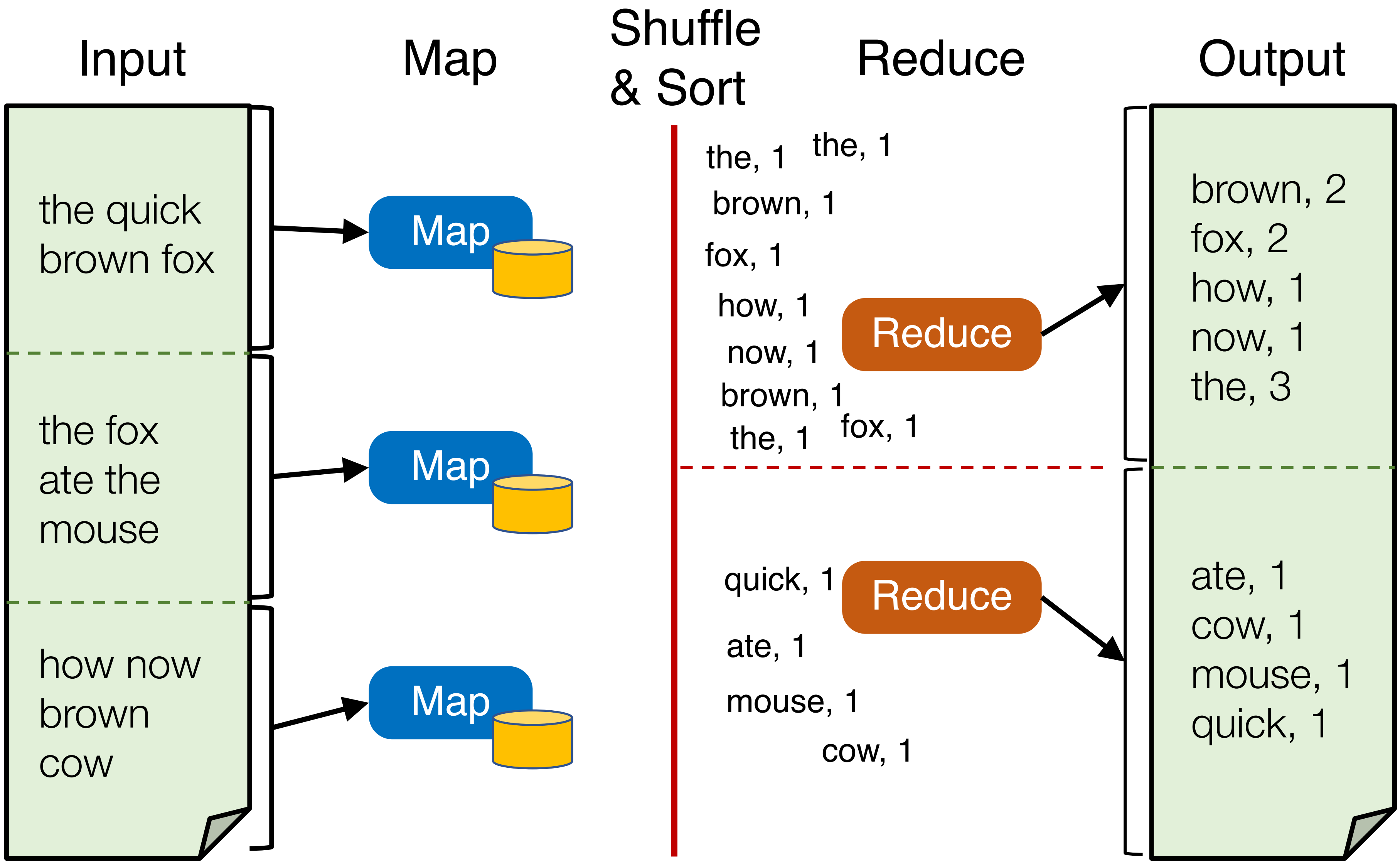
Word Count execution



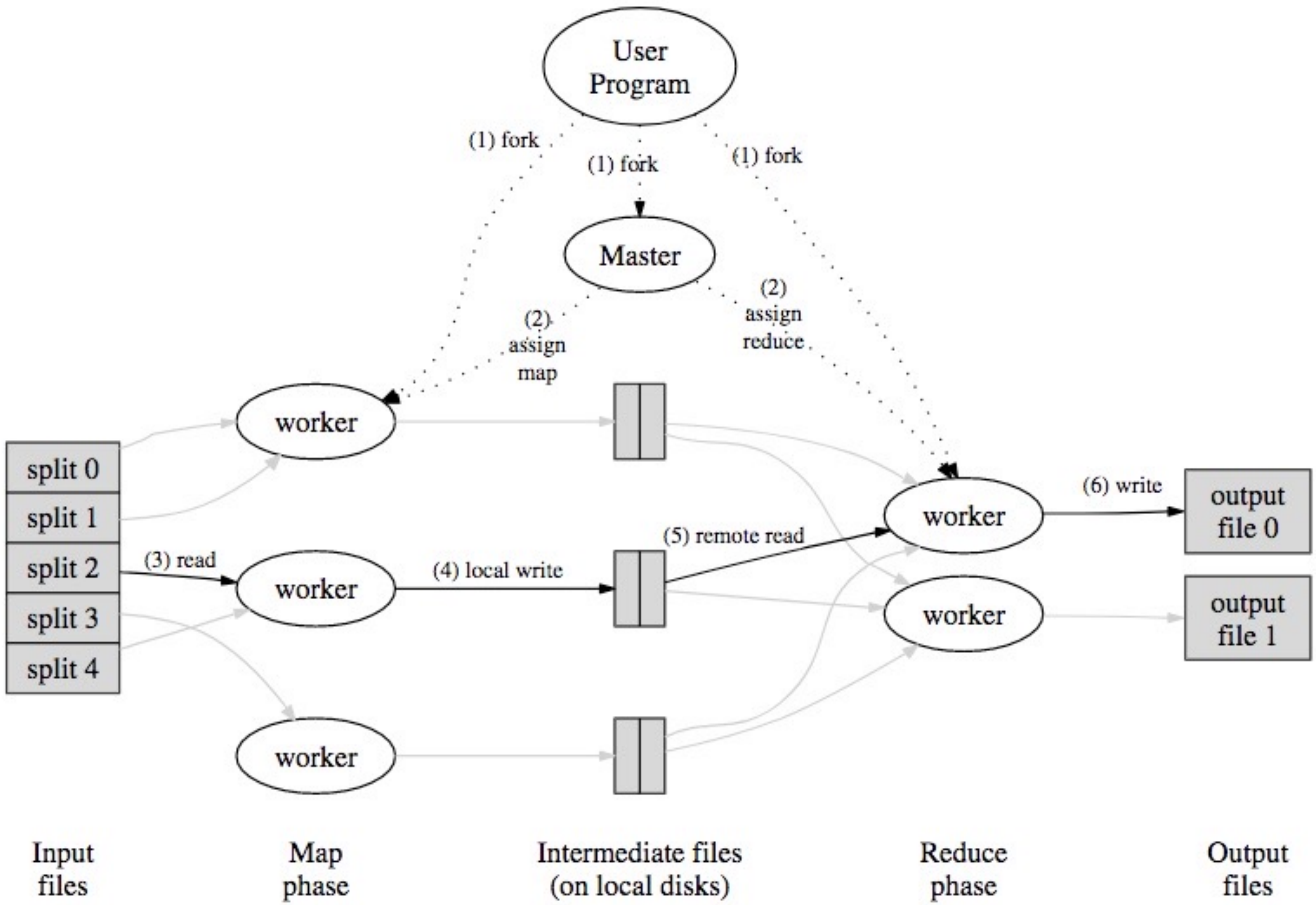
Word Count execution



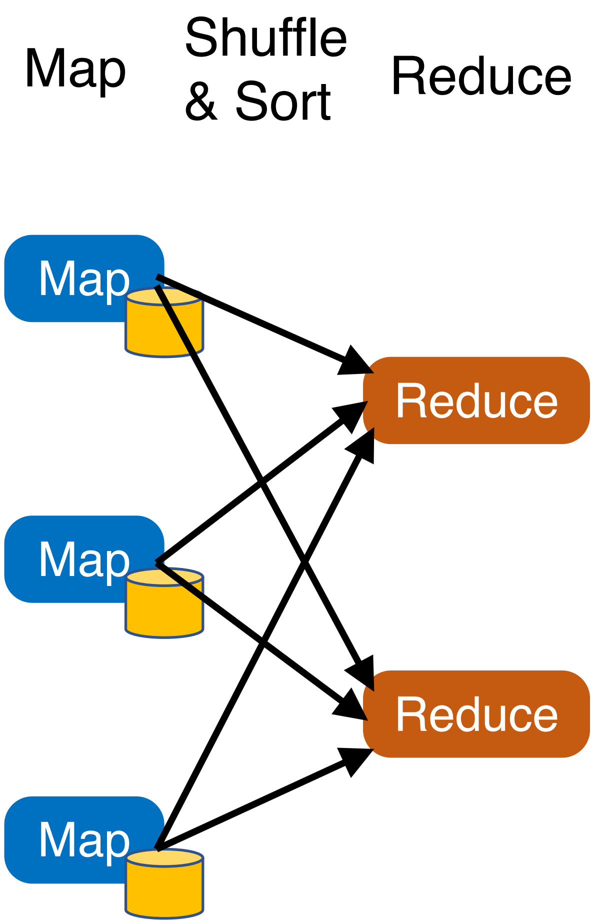
Word Count execution



MapReduce data flows



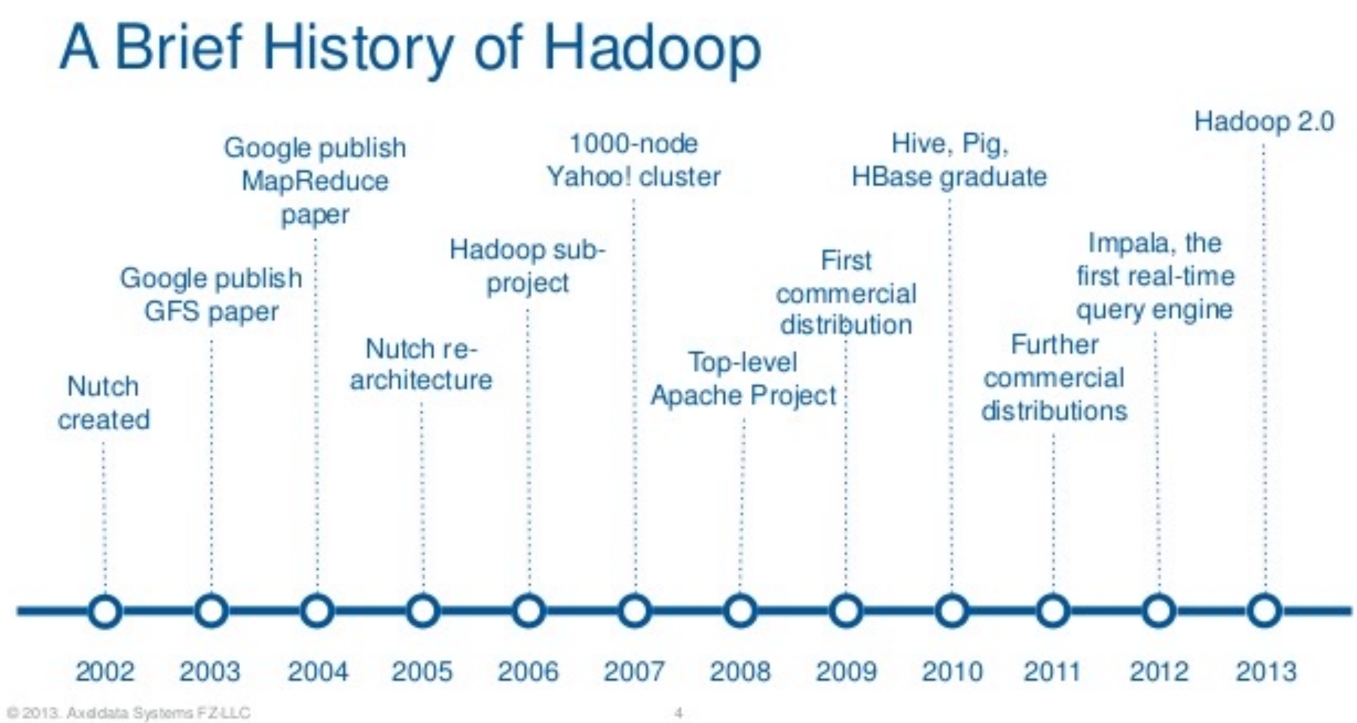
MapReduce processes

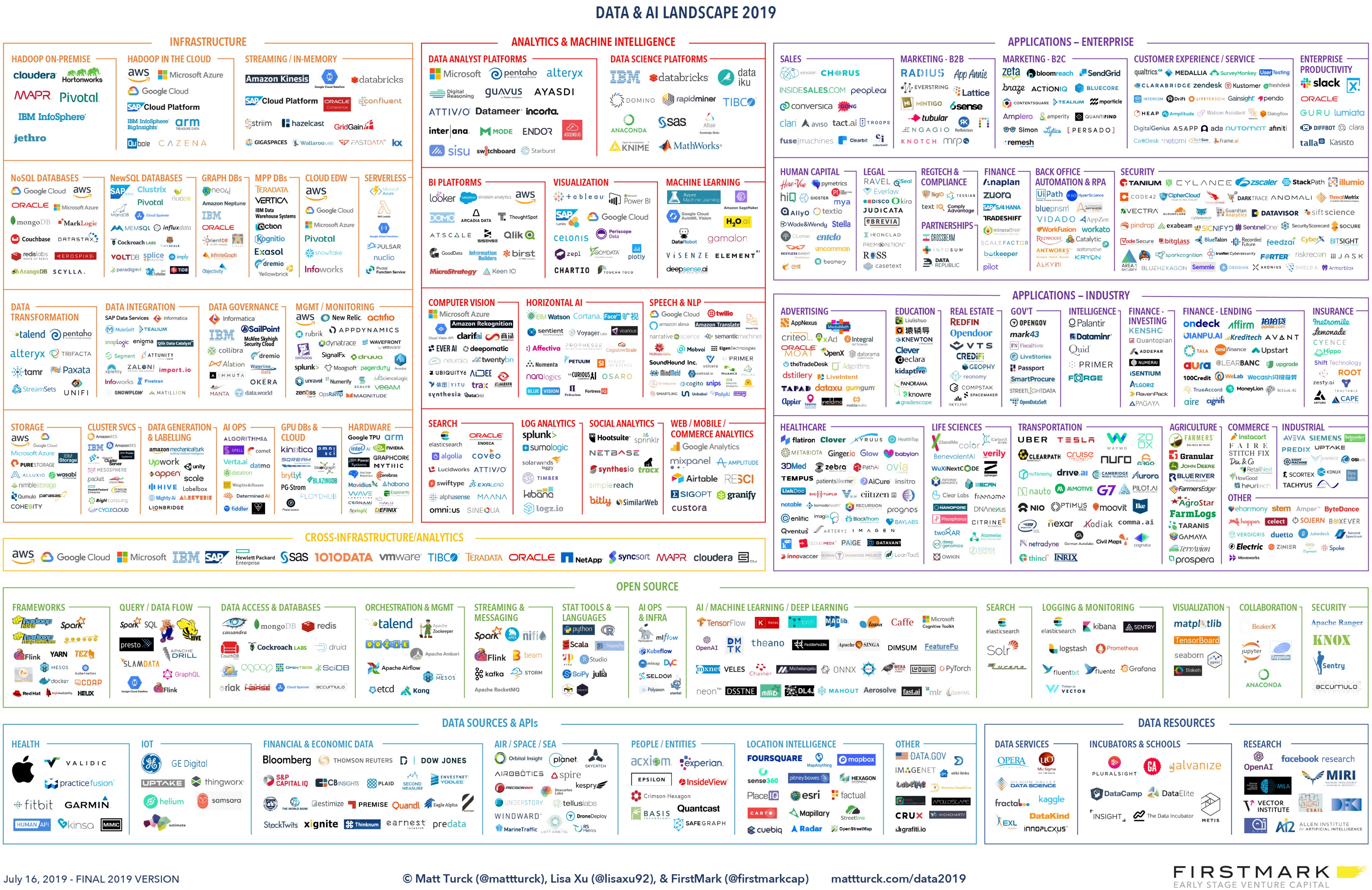


- Map workers write intermediate output to local disk, separated by partitioning. Once completed, tell master node
- Reduce worker told of location of map task outputs, pulls their partition's data from each mapper, execute function across data
- Note:
  - "All-to-all" shuffle b/w mappers and reducers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

Apache Hadoop

- An open-source implementation of Google's MapReduce framework
  - Hadoop MapReduce atop Hadoop Distributed File System (HDFS)




DATA & AI LANDSCAPE 2019
INFRASTRUCTURE
HADOOP ON-PREMISE
HADOOP IN THE CLOUD
STREAMING / IN-MEMORY
NoSQL DATABASES
NewSQL DATABASES
GRAPH DBs
MPP DBs
CLOUD EDW
SERVERLESS
DATA TRANSFORMATION
DATA INTEGRATION
DATA GOVERNANCE
MGMT / MONITORING
STORAGE
CLUSTER SVCS
DATA GENERATION & LABELLING
AI OPS
GPU DBs & CLOUD
HARDWARE
ANALYTICS & MACHINE INTELLIGENCE
DATA ANALYST PLATFORMS
DATA SCIENCE PLATFORMS
BI PLATFORMS
VISUALIZATION
MACHINE LEARNING
COMPUTER VISION
HORIZONTAL AI
SPEECH & NLP
SEARCH
LOG ANALYTICS
SOCIAL ANALYTICS
WEB / MOBILE / COMMERCE ANALYTICS
APPLICATIONS – ENTERPRISE
SALES
MARKETING - B2B
MARKETING - B2C
CUSTOMER EXPERIENCE / SERVICE
ENTERPRISE PRODUCTIVITY
HUMAN CAPITAL
LEGAL
REGTECH & COMPLIANCE
PARTNERSHIPS
FINANCE
BACK OFFICE AUTOMATION & RPA
SECURITY
APPLICATIONS – INDUSTRY
ADVERTISING
EDUCATION
REAL ESTATE
GOV'T
INTELLIGENCE
FINANCE - INVESTING
FINANCE - LENDING
INSURANCE
HEALTHCARE
LIFE SCIENCES
TRANSPORTATION
AGRICULTURE
COMMERCE
INDUSTRIAL
OTHER
CROSS-INFRASTRUCTURE/ANALYTICS
OPEN SOURCE
FRAMEWORKS
QUERY / DATA FLOW
DATA ACCESS & DATABASES
ORCHESTRATION & MGMT
STREAMING & MESSAGING
STAT TOOLS & LANGUAGES
AI OPS & INFRA
AI / MACHINE LEARNING / DEEP LEARNING
SEARCH
LOGGING & MONITORING
VISUALIZATION
COLLABORATION
SECURITY
DATA SOURCES & APIs
HEALTH
IOT
FINANCIAL & ECONOMIC DATA
AIR / SPACE / SEA
PEOPLE / ENTITIES
LOCATION INTELLIGENCE
OTHER
DATA RESOURCES
DATA SERVICES
INCUBATORS & SCHOOLS
RESEARCH
July 16, 2019 - FINAL 2019 VERSION
© Matt Turck (@mattturck), Lisa Xu (@lisaxu92), & FirstMark (@firstmarkcap)
mattturck.com/data2019
FIRSTMARK
EARLY STAGE VENTURE CAPITAL