MapReduce

What do we use distributed systems for?

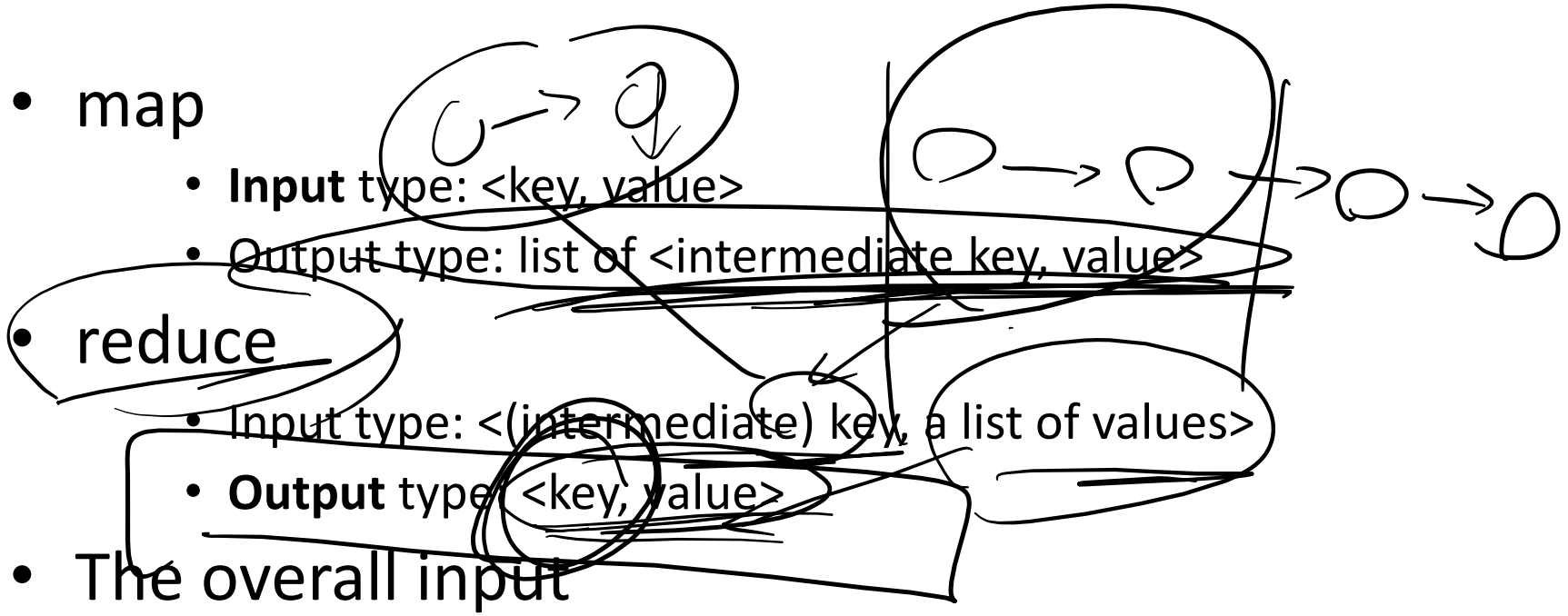
- Distributed storage systems
  - NFS, AFS, GFS
- Distributed computation framework
  - Mapreduce, DryadLINQ

- Key techniques
  - Partition & Replication
- Key concerns
  - Fault tolerance

Overview

- Goal: ease the processing and generating of large data sets
- A programming model
  - Map (key-value → intermediateKey-values)
  - Reduce (group and aggregate)
- A run-time system
  - Data partition
  - Task scheduling
  - Failure handling
  - Communication

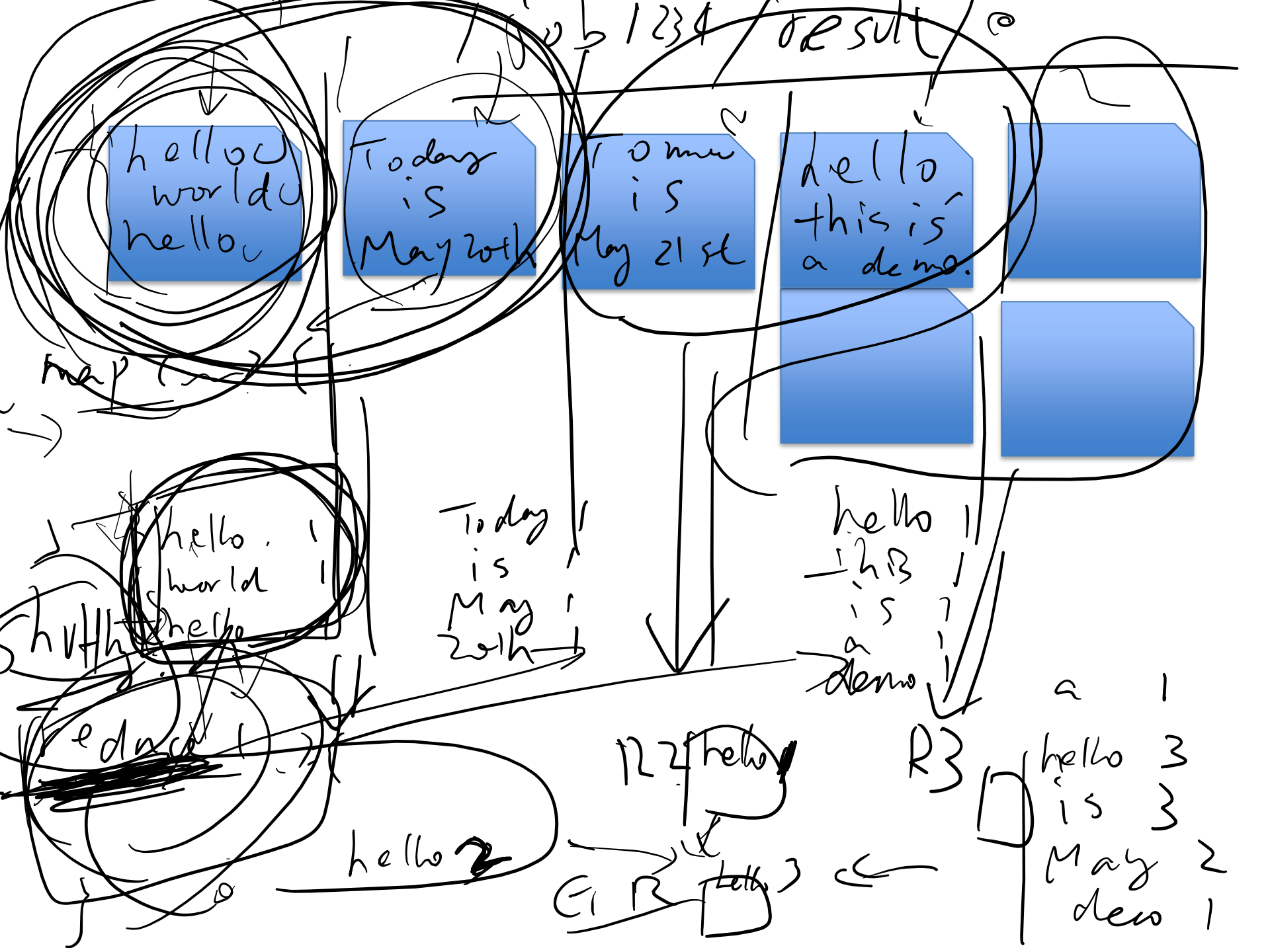
Programming model



- map
  - **Input** type: <key, value>
  - Output type: list of <intermediate key, value>
- reduce
  - Input type: <(intermediate) key, a list of values>
  - **Output** type: <key, value>
- The overall input
  - A set of <key, value>
- The overall output
  - A set of <(intermediate) key, value>

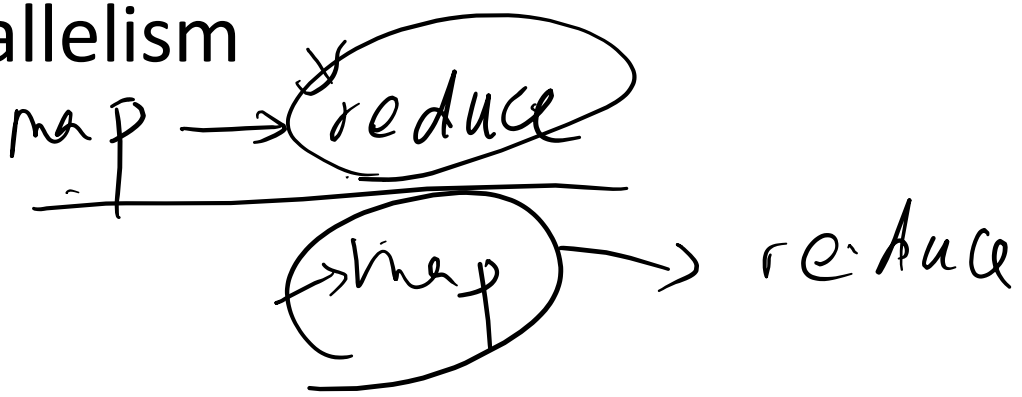
Example: word count

- map
  - **Input** type: <key, value>
  - Output type: list of <intermediate key, value>
- reduce
  - Input type: <(intermediate) key, a list of values>
  - **Output** type: <key, value>


hello world hello
Today is May 20th
is May 21st
hello this is a demo.
Today is May 20th
hello this is a demo
hello world hello
hello 2
hello 3
hello 3 is 3 May 2 demo 1

Parallelization

- Data parallelism
- Task parallelism
- Pipeline parallelism



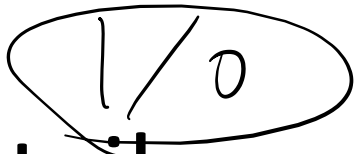
Example 1

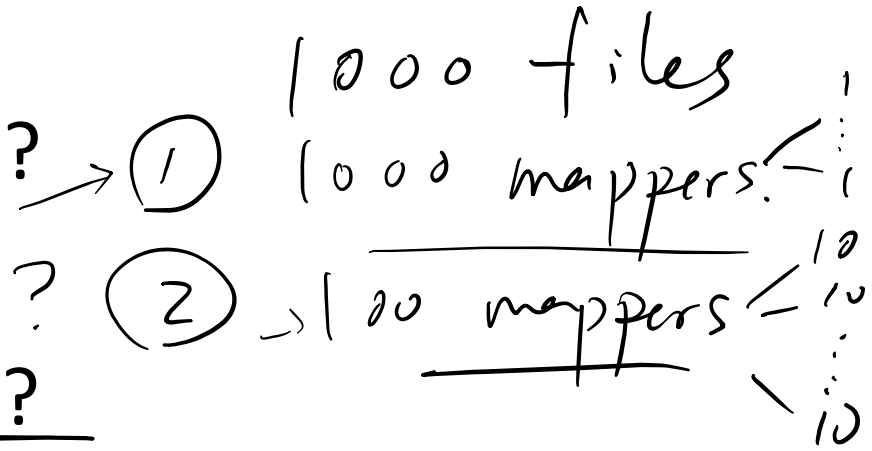
- Word counting
  - Input (a set of document)
  - Output (a list of <word, #cnt>)
- map (string doc-name, string doc)

- reduce (,)

How to partition/parallelize the work?

Partition details

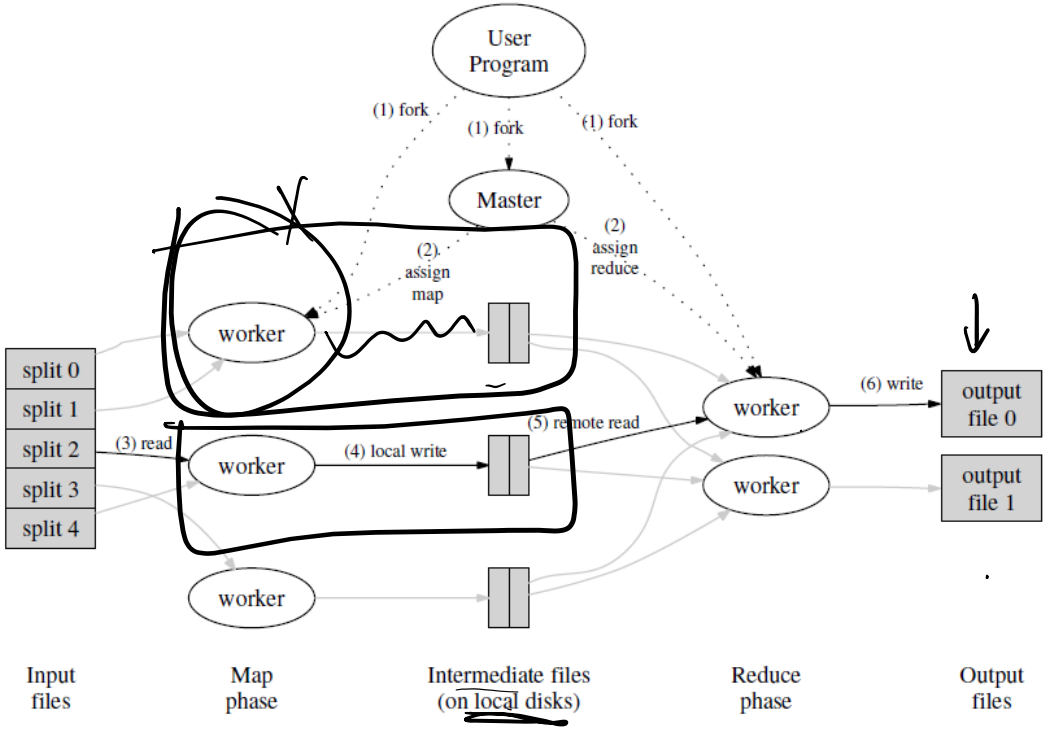




- How many mappers?
- How many reducers?
- When to start a reducer?
- Who ~~manage all these~~?

What does the master do?

- Task → idle/in-progress/completed
- Task →worker machine
- Intermediate results locations (pushed to R)



Run-time flow (1)

- Step 1: split data to M (16MB—64MB) pieces
- Step 2: start R+M+1 Processes
- Step 3: map worker works
  - Where are the results of a map worker?
  - Should the result be partitioned? How?

Run-time flow (2)

- Step 4: reduce worker works
  - How to get the input?
  - Preprocessing
  - Reduce
  - Where is the output?
- Step 5: master contacts the user
  - Where is the output?

Performance details

- How to schedule for good data locality?
- How to partition to achieve good load balance?
- Could the master become a bottleneck?
- What if some nodes are extremely slow?
  - Straggler, Backup execution

- *M, R, worker: 200,000; 5,000; 2,000*

Fault tolerance details

- Worker failure
  - How to detect it?
  - What about finished map/reduce tasks on that worker?
  - What about on-going map/reduce tasks on that worker?
- Master failure

Other examples

- Distributed grep (I: one or multiple docs)
- Count of URL access frequency (I: URL log)
- Reverse web-link graph
  - Webpages → <target, list(source)>
- Distributed sort
  - Records →sorted records

What is special about this programming model?

- What is the relationship between it as parallel computation?
  - Easy
  - Functional programming model (no side effect)
  - Easy for recovery (reexecution)
  - Only for data parallelism

- Questions
  - How to handle iterative job?
  - How to handle non-commutative, non-associative job?
  - What if map/reduce is non-deterministic?

Summary

- The programming model
- The run-time system
- How load balance, scheduling, failure tolerance … are transparently achieved