MapReduce and Hadoop Distributed File System

The Context: Big-data

- Man on the moon with 32KB (1969); my laptop had 2GB RAM (2009)
- Google collects 270PB data in a month (2007), 20000PB a day (2008)
- 2010 census data is expected to be a huge gold mine of information
- Data mining huge amounts of data collected in a wide range of domains from astronomy to healthcare has become essential for planning and performance.
- We are in a knowledge economy.
  - Data is an important asset to any organization
  - Discovery of knowledge; Enabling discovery; annotation of data
- We are looking at newer
  - programming models, and
  - Supporting algorithms and data structures.
- NSF refers to it as “data-intensive computing” and industry calls it “big-data” and “cloud computing”

Purpose of this talk

- To provide a simple introduction to:
  - "The big-data computing" : An important advancement that has a potential to impact significantly the CS and undergraduate curriculum.
  - A programming model called MapReduce for processing "big-data"
  - A supporting file system called Hadoop Distributed File System (HDFS)
- To encourage educators to explore ways to infuse relevant concepts of this emerging area into their curriculum.

The Outline

- Introduction to MapReduce
- From CS Foundation to MapReduce
- MapReduce programming model
- Hadoop Distributed File System
- Relevance to Undergraduate Curriculum
- Demo (Internet access needed)
- Our experience with the framework
- Summary
- References

MapReduce

What is MapReduce?

- MapReduce is a programming model Google has used successfully is processing its "big-data" sets (~ 20000 peta bytes per day)
  - Users specify the computation in terms of a *map* and a *reduce* function,
  - Underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, and
  - Underlying system also handles machine failures, efficient communications, and performance issues.

-- Reference: Dean, J. and Ghemawat, S. 2008. **MapReduce: simplified data processing on large clusters.** *Communication of ACM* 51, 1 (Jan. 2008), 107-113.

From CS Foundations to MapReduce

Consider a large data collection:

{web, weed, green, sun, moon, land, part, web, green,…}

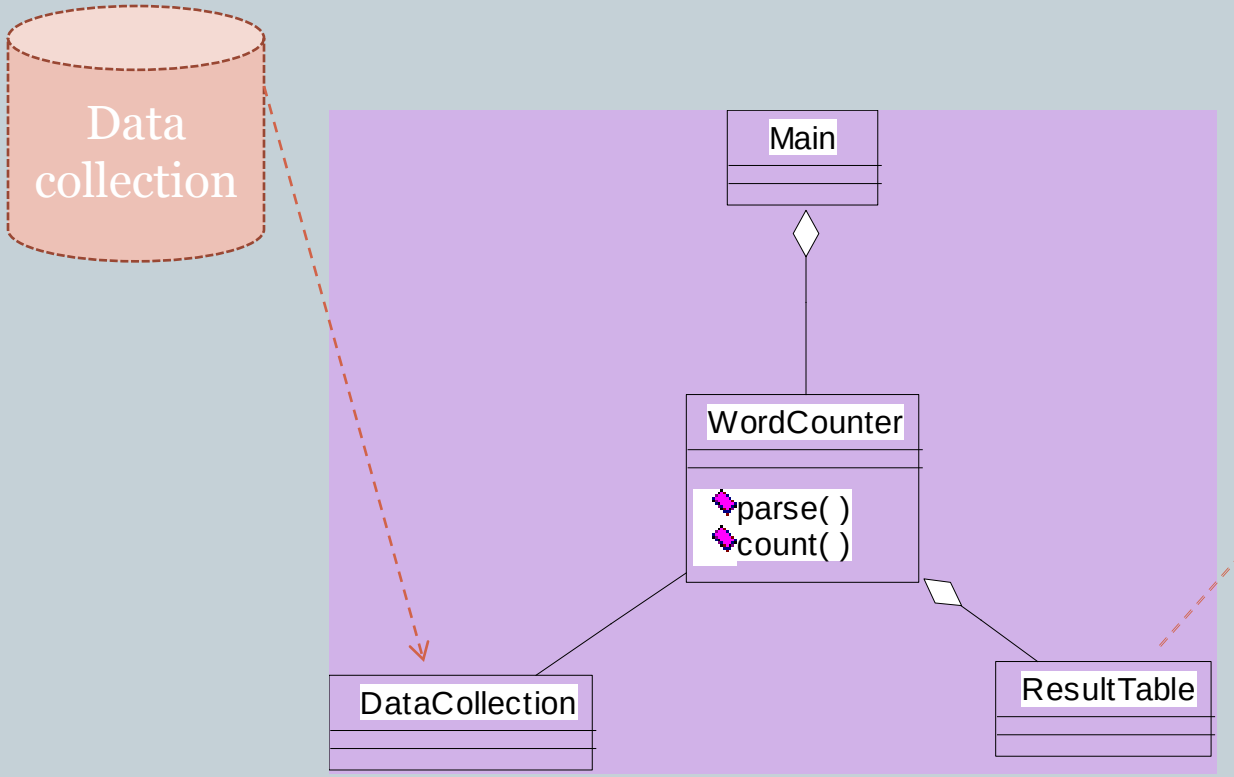
Problem: Count the occurrences of the different words in the collection.

Lets design a solution for this problem;

- We will start from scratch
- We will add and relax constraints
- We will do incremental design, improving the solution for performance and scalability

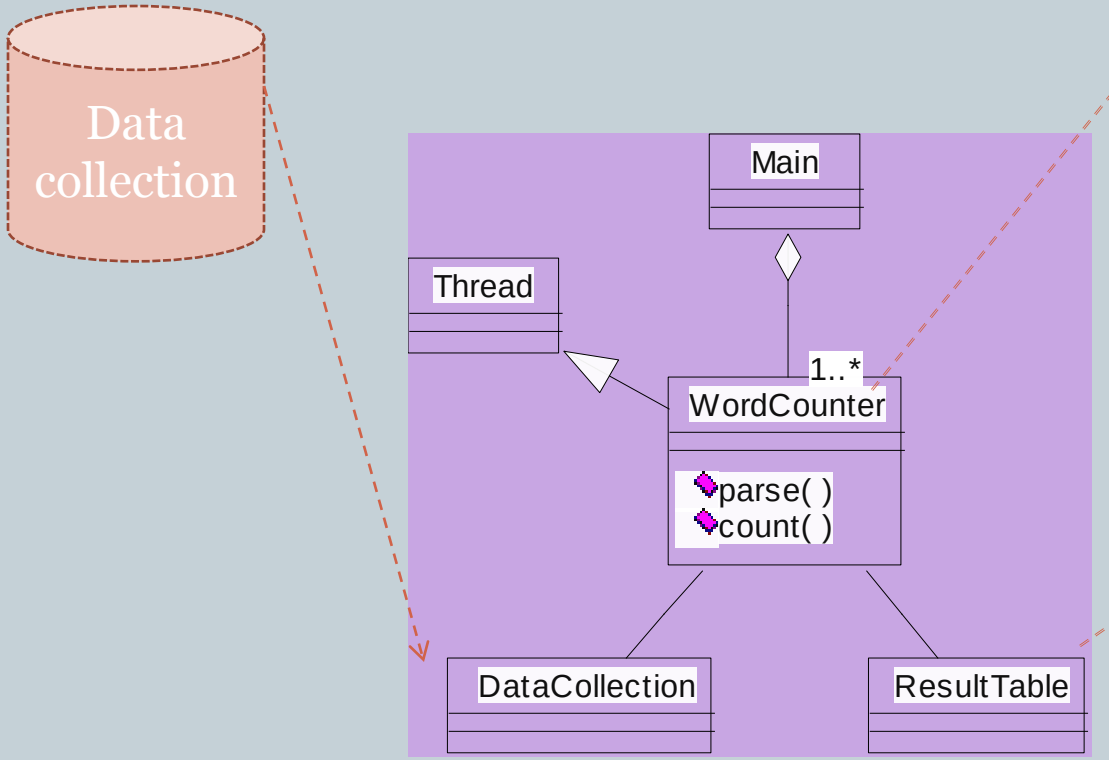
Word Counter and Result Table

{web, weed, green, sun, moon, land, part, web, green,...}



web	**2**
weed	**1**
green	**2**
sun	**1**
moon	**1**
land	**1**
part	**1**

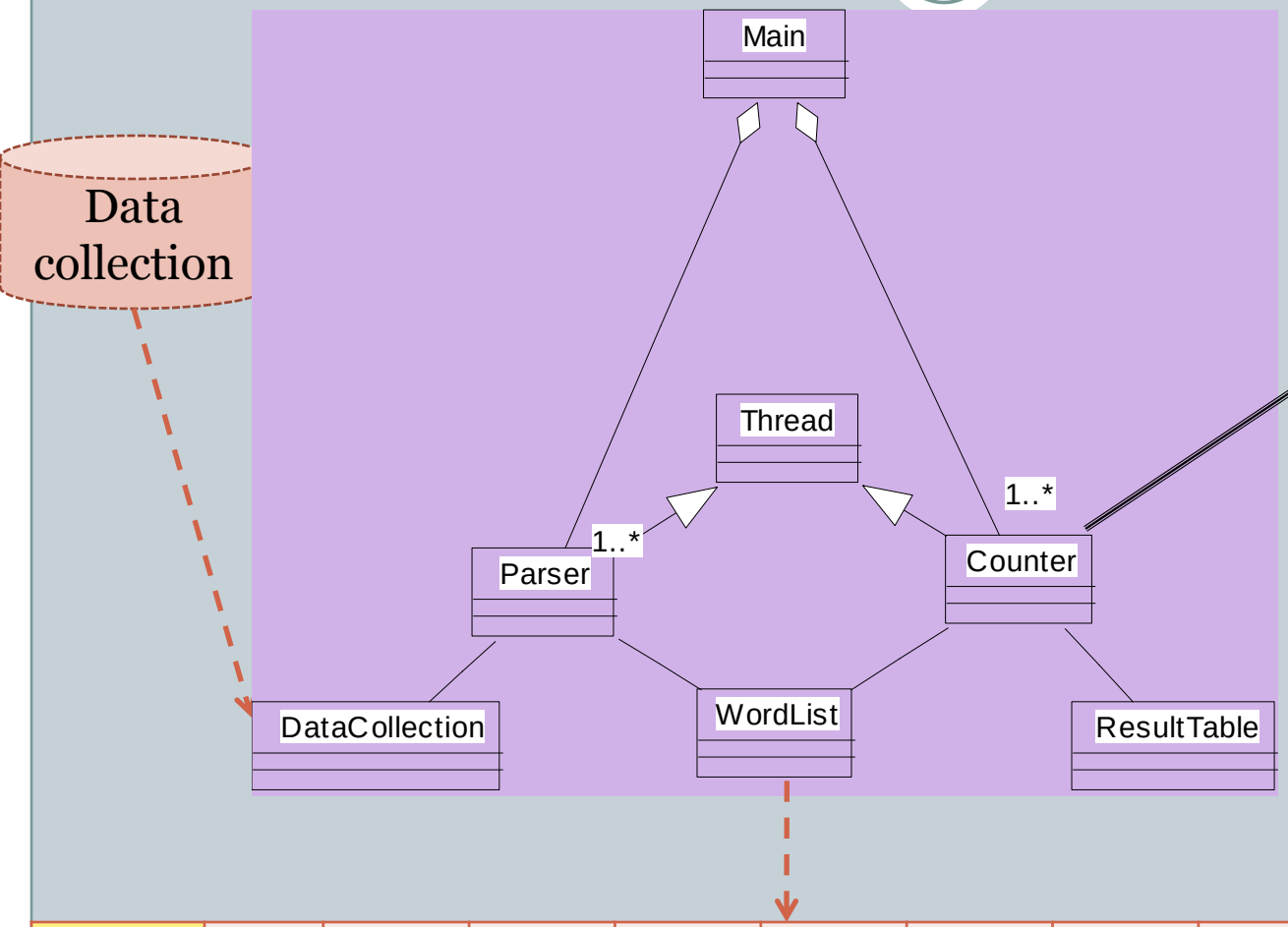
Multiple Instances of Word Counter



web	2
weed	1
green	2
sun	1
moon	1
land	1
part	1

Observe:
Multi-thread
Lock on shared data

Improve Word Counter for Performance



No need for lock

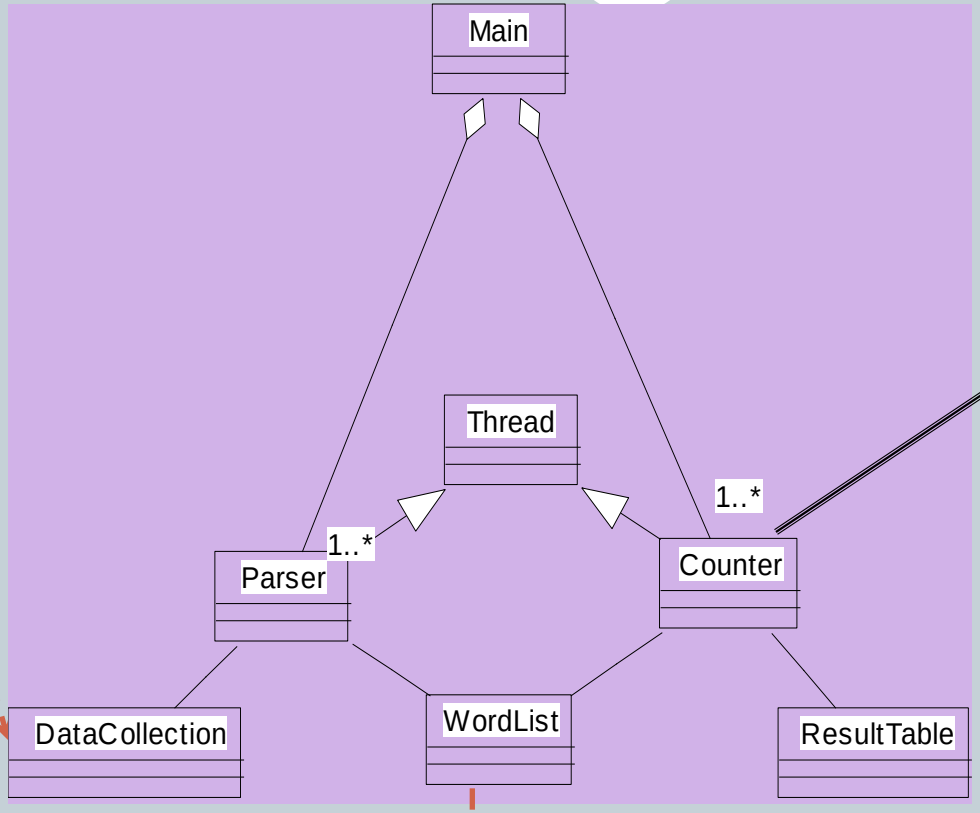
web	2
weed	1
green	2
sun	1
moon	1
land	1
part	1

Separate counters

KEY	web	weed	green	sun	moon	land	part	web	green	.....
VALUE										

Peta-scale Data

Data collection



web	2
weed	1
green	2
sun	1
moon	1
land	1
part	1

KEY	web	weed	green	sun	moon	land	part	web	green	.....
VALUE										

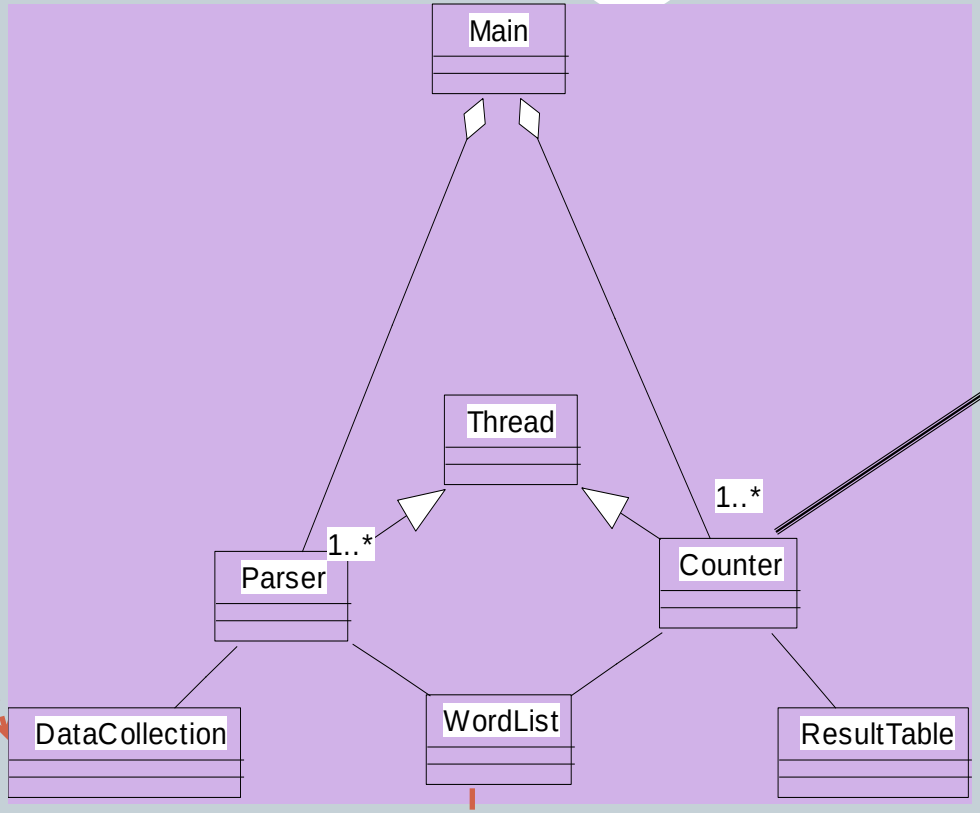
Addressing the Scale Issue

- Single machine cannot serve all the data: you need a distributed special (file) system
- Large number of commodity hardware disks: say, 1000 disks 1TB each
  - Issue: With Mean time between failures (MTBF) or failure rate of 1/1000, then at least 1 of the above 1000 disks would be down at a given time.
  - Thus failure is norm and not an exception.
  - File system has to be fault-tolerant: replication, checksum
  - Data transfer bandwidth is critical (location of data)

- Critical aspects: fault tolerance + replication + load balancing, monitoring
- Exploit parallelism afforded by splitting parsing and counting
- Provision and locate computing at data locations

Peta-scale Data

Data collection



web	2
weed	1
green	2
sun	1
moon	1
land	1
part	1

KEY	web	weed	green	sun	moon	land	part	web	green	.....
VALUE										

Peta Scale Data is Commonly Distributed

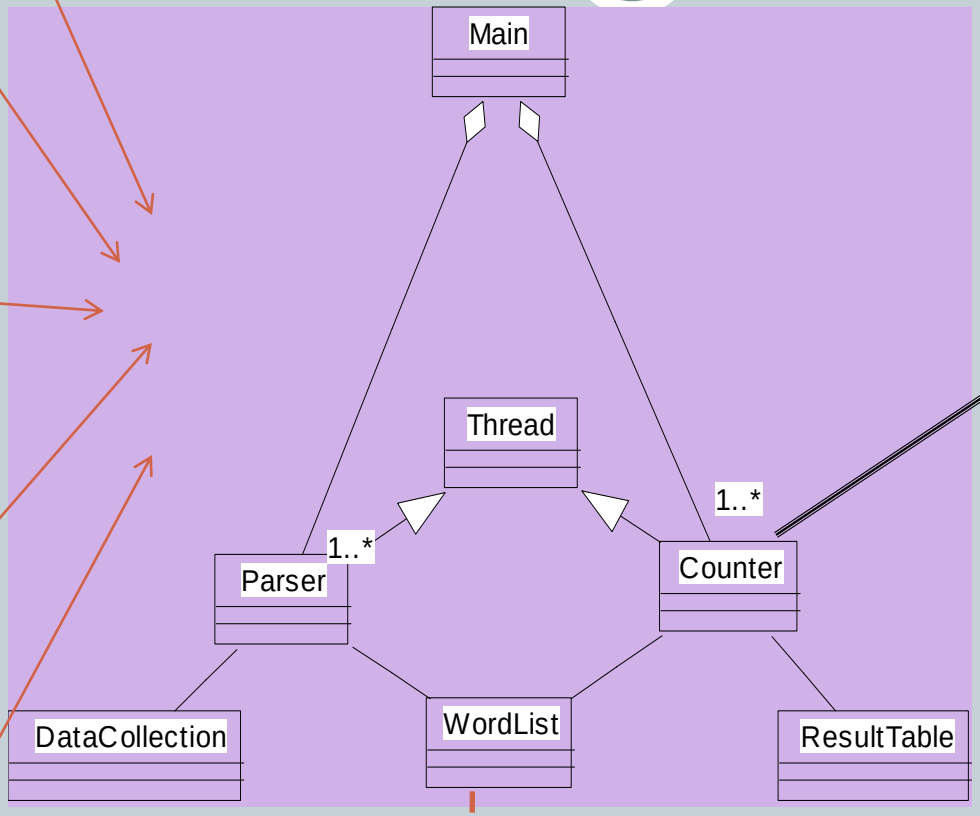
Data collection

Data collection

Data collection

Data collection

Data collection



web	2
weed	1
green	2
sun	1
moon	1
land	1
part	1

Issue: managing the large scale data

KEY	web	weed	green	sun	moon	land	part	web	green	.....
VALUE										

Write Once Read Many (WORM) data

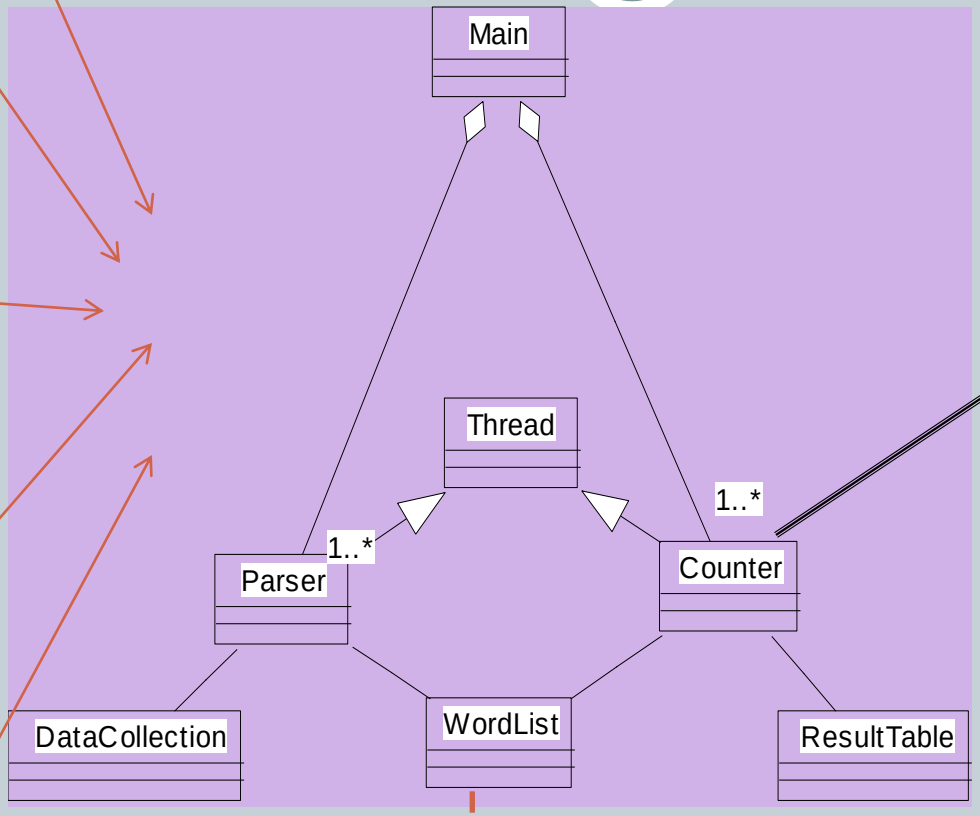
Data collection

Data collection

Data collection

Data collection

Data collection



web	2
weed	1
green	2
sun	1
moon	1
land	1
part	1

KEY	web	weed	green	sun	moon	land	part	web	green	.....
VALUE										

WORM Data is Amenable to Parallelism

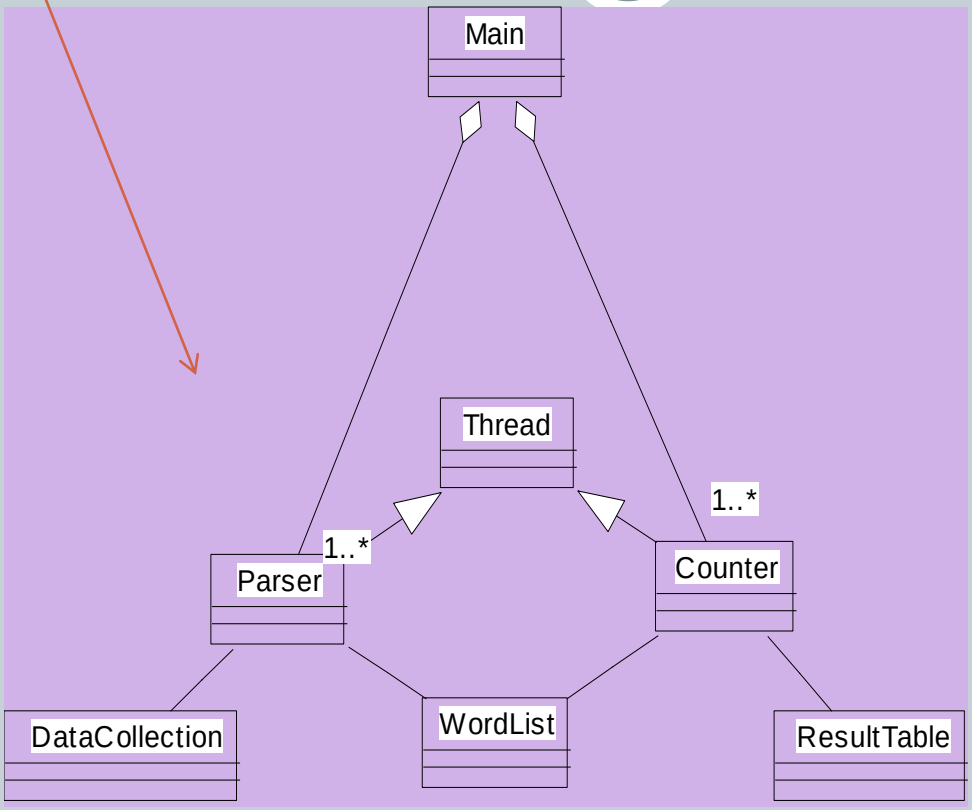
Data collection

Data collection

Data collection

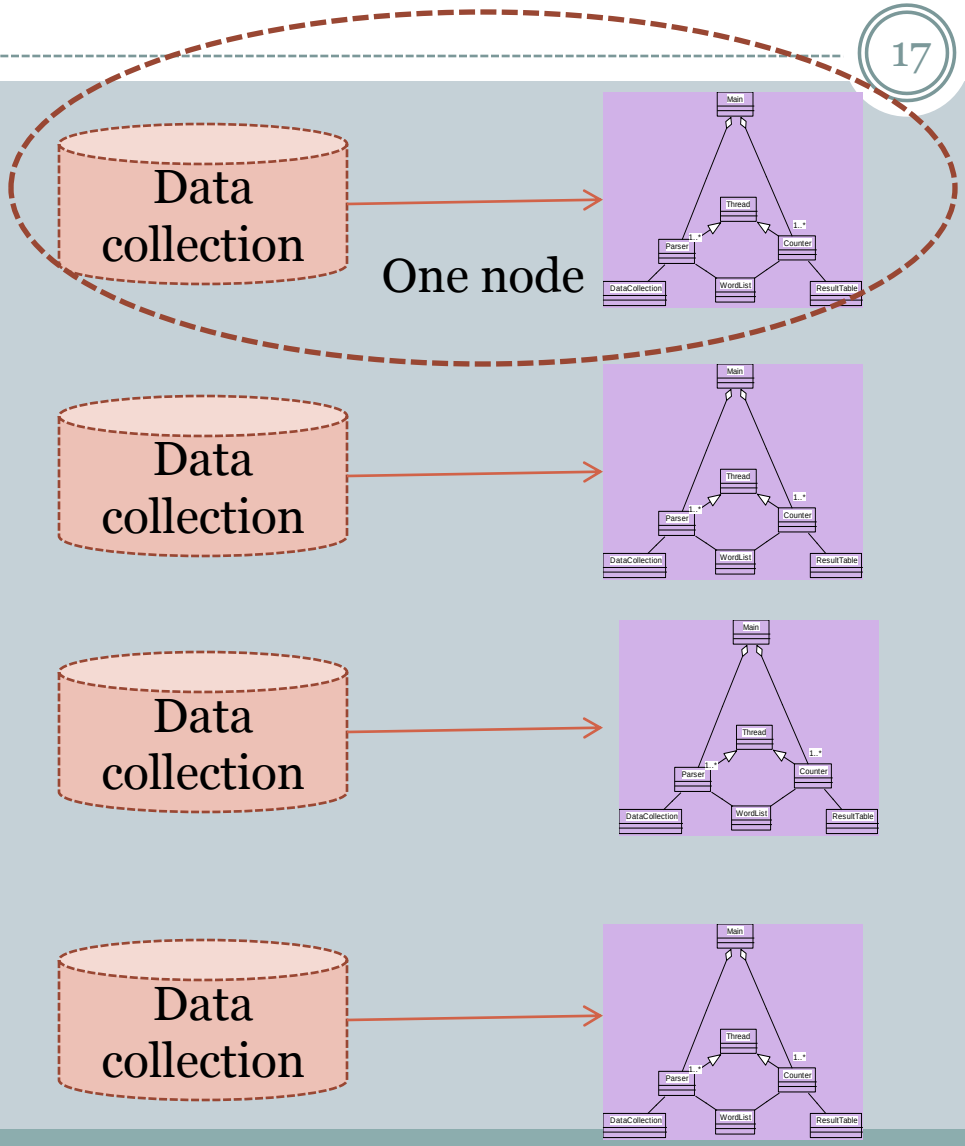
Data collection

Data collection



1. Data with WORM characteristics : yields to parallel processing;
2. Data without dependencies: yields to out of order processing

Divide and Conquer: Provision Computing at Data Location



For our example,
#1: Schedule parallel parse tasks
#2: Schedule parallel count tasks

This is a particular solution;
Lets generalize it:

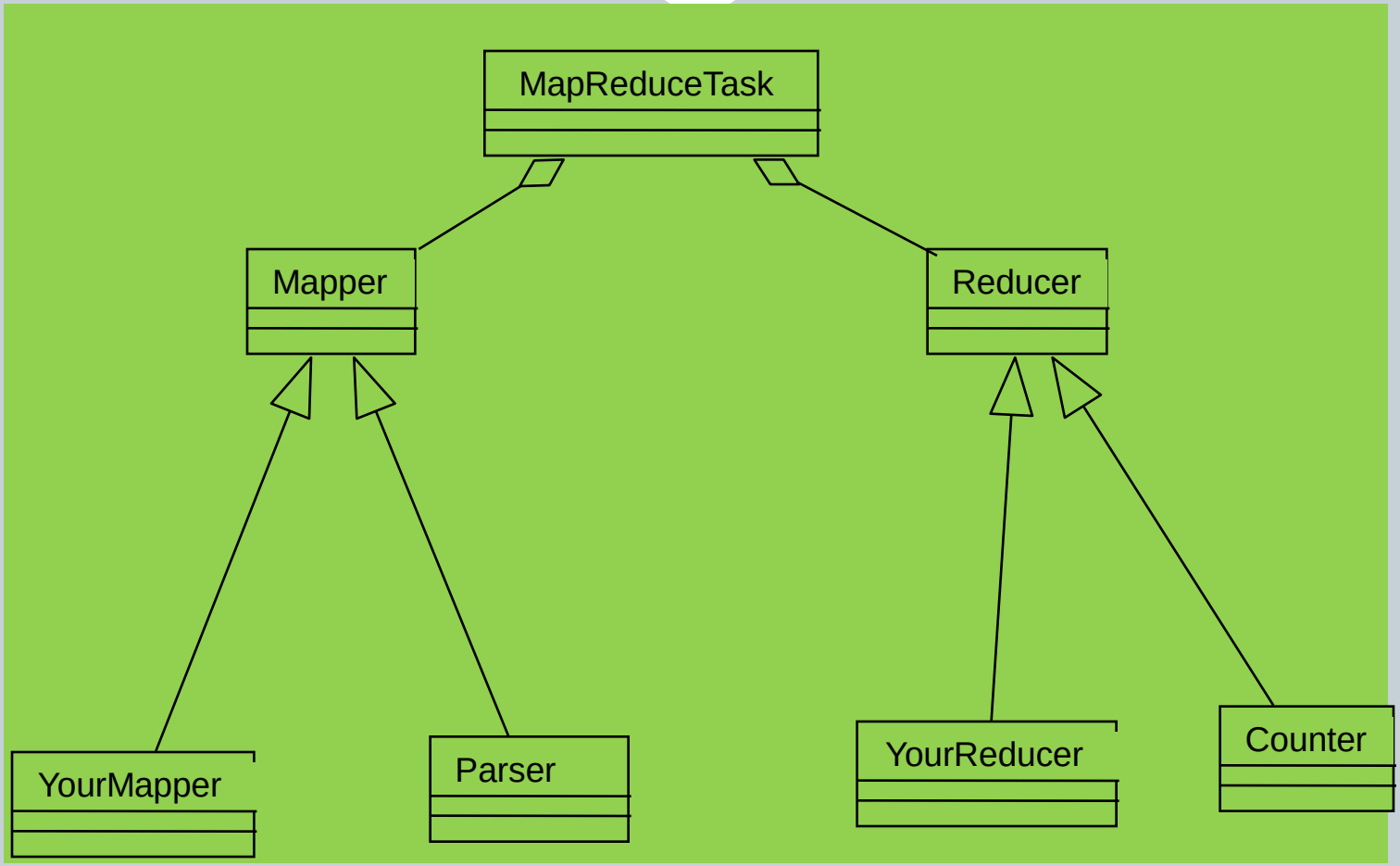
Our parse is a mapping operation:
MAP: input → <key, value> pairs

Our count is a reduce operation:
REDUCE: <key, value> pairs reduced

Map/Reduce originated from Lisp
But have different meaning here

Runtime adds distribution + fault tolerance + replication + monitoring + load balancing to your base application!

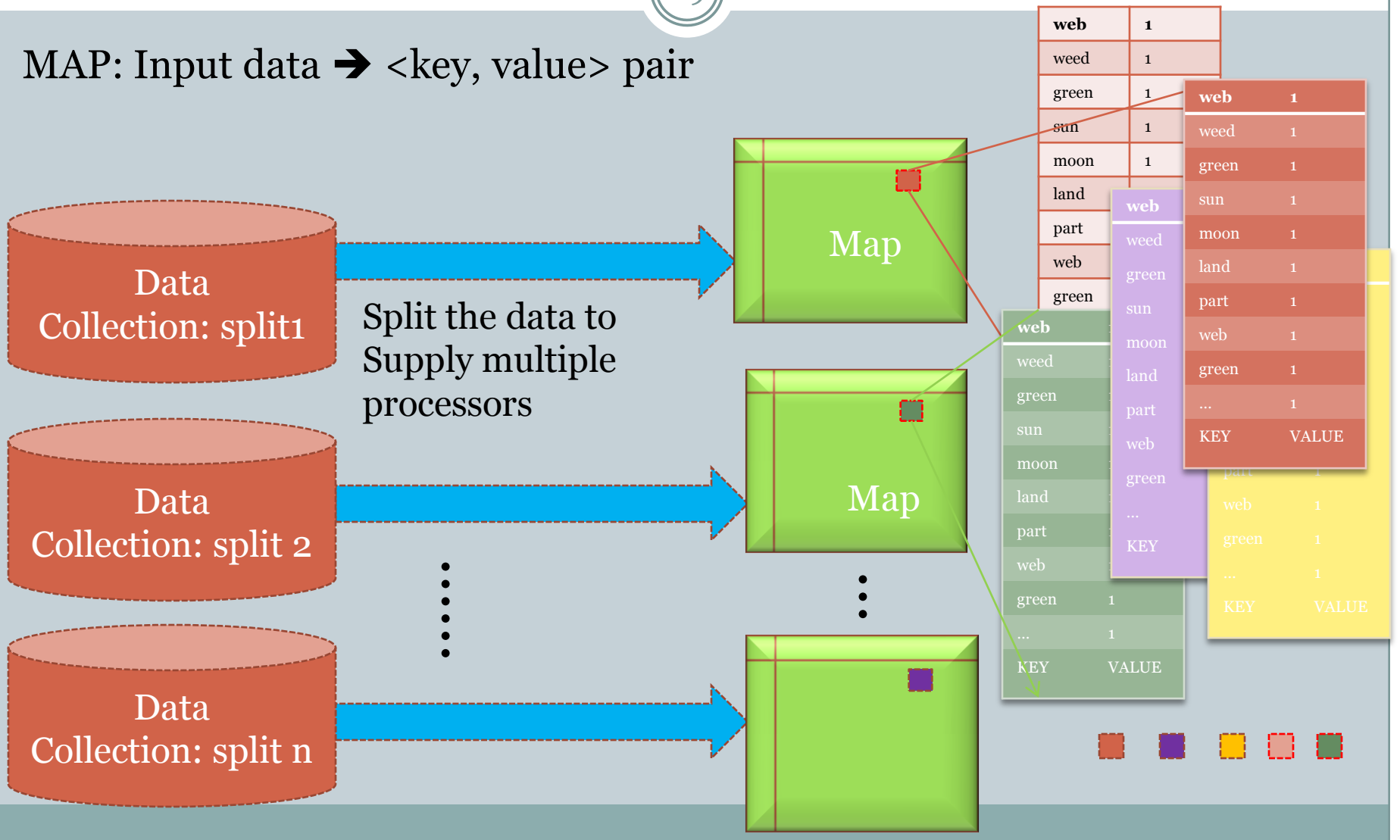
Mapper and Reducer



Remember: MapReduce is simplified processing for larger data sets:
MapReduce Version of [WordCount Source code]

Map Operation

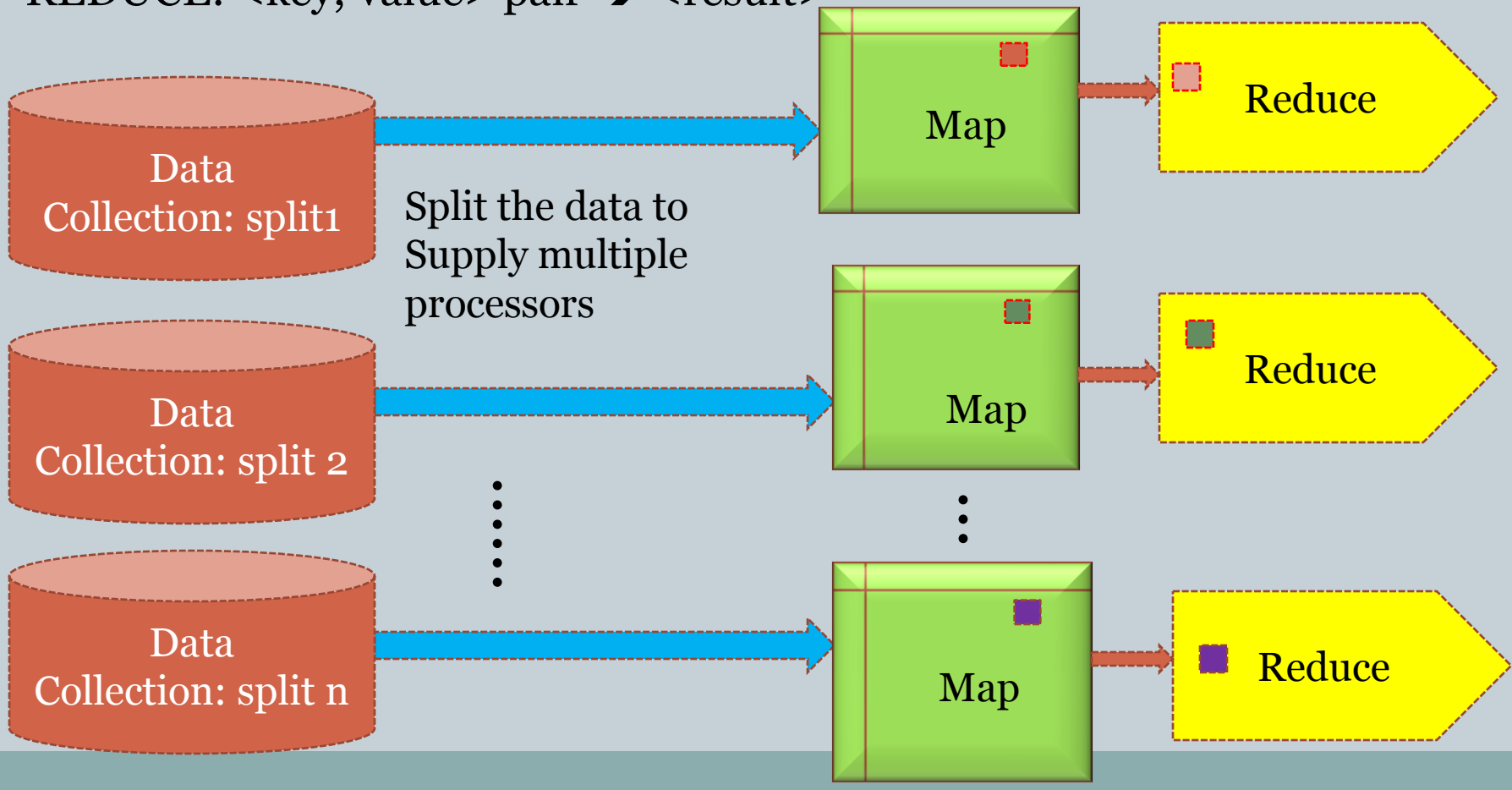
MAP: Input data ➔ <key, value> pair

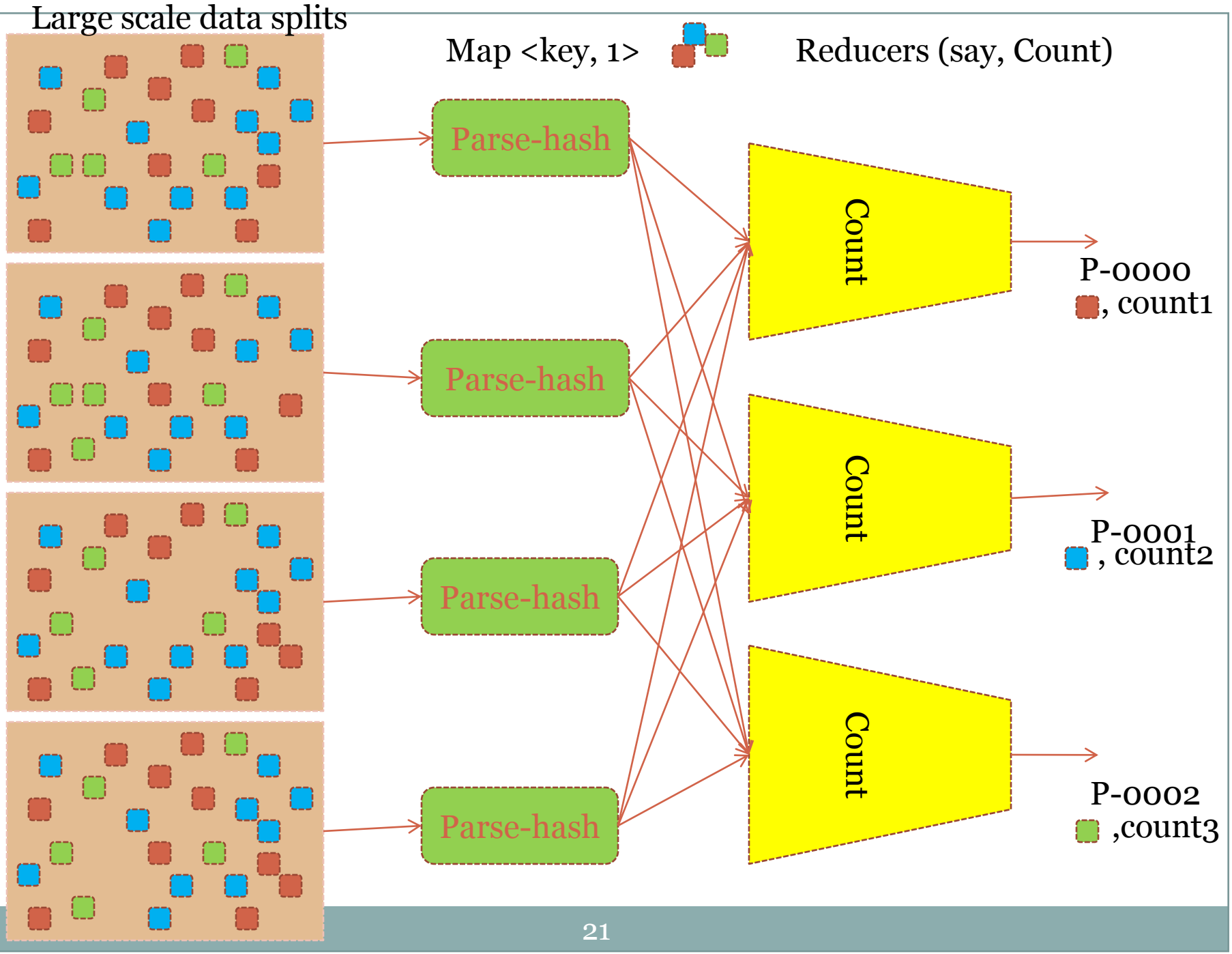


Reduce Operation

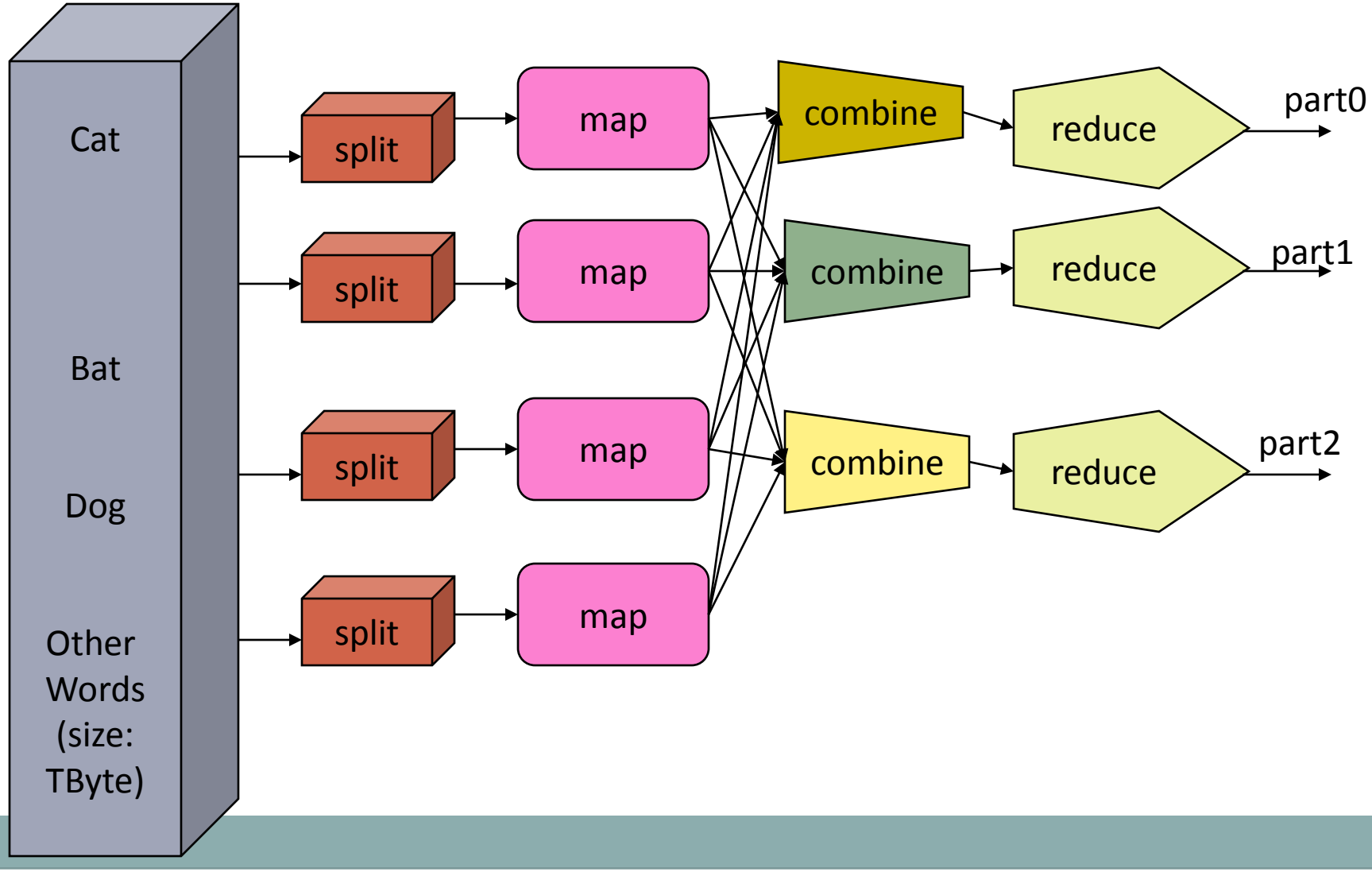
MAP: Input data ➔ <key, value> pair

REDUCE: <key, value> pair ➔ <result>




Large scale data splits
Map <key, 1>
Reducers (say, Count)
Parse-hash
Parse-hash
Parse-hash
Parse-hash
Count
Count
Count
P-0000
, count1
P-0001
, count2
P-0002
,count3

MapReduce Example in my operating systems class



MapReduce Programming Model

MapReduce programming model

- Determine if the problem is parallelizable and solvable using MapReduce (ex: Is the data WORM?, large data set).
- Design and implement solution as Mapper classes and Reducer class.
- Compile the source code with hadoop core.
- Package the code as jar executable.
- Configure the application (job) as to the number of mappers and reducers (tasks), input and output streams
- Load the data (or use it on previously available data)
- Launch the job and monitor.
- Study the result.
- Detailed steps.

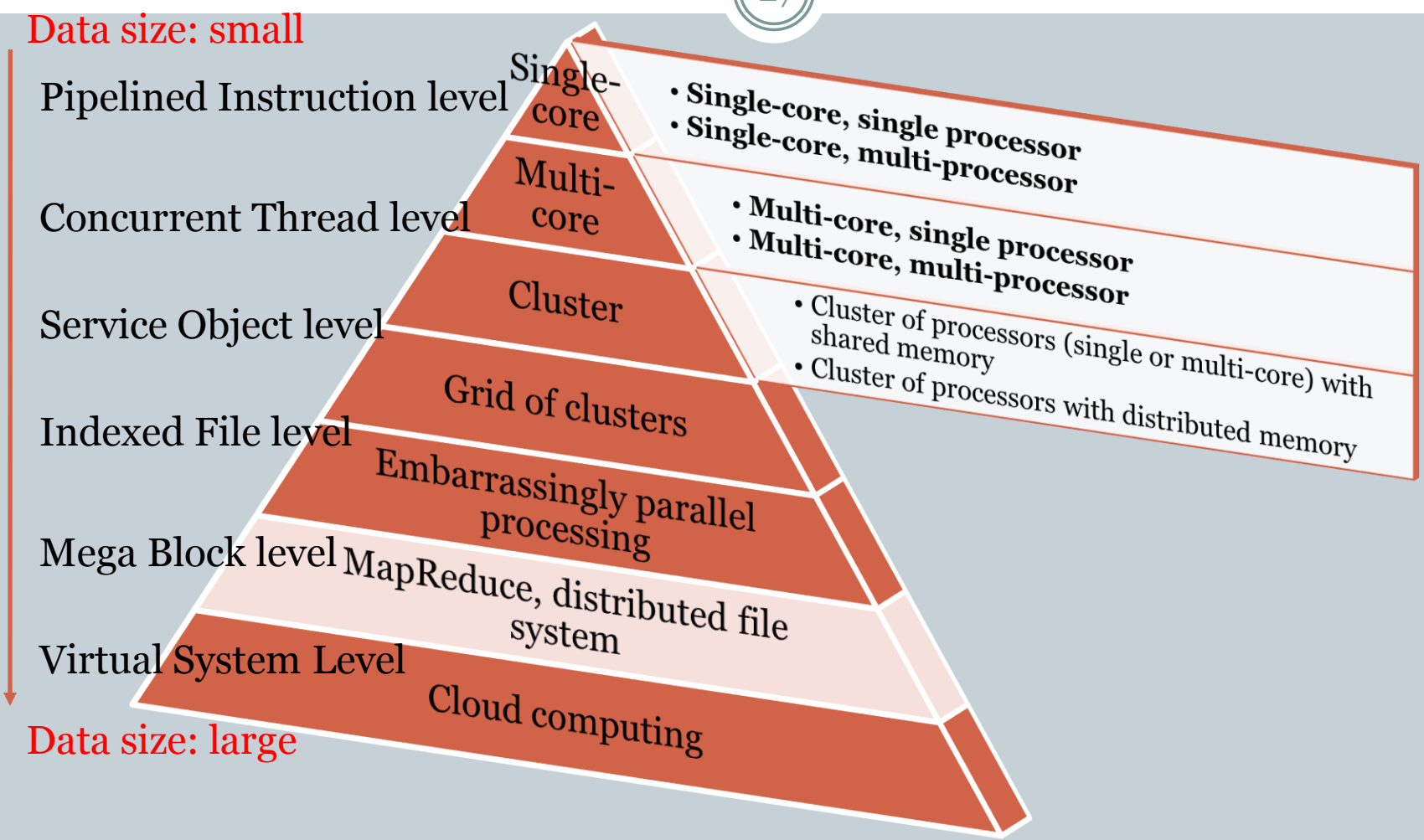
MapReduce Characteristics

- Very large scale data: peta, exa bytes
- Write once and read many data: allows for parallelism without mutexes
- Map and Reduce are the main operations: simple code
- There are other supporting operations such as combine and partition (out of the scope of this talk).
- All the map should be completed before reduce operation starts.
- Map and reduce operations are typically performed by the same physical processor.
- Number of map tasks and reduce tasks are configurable.
- Operations are provisioned near the data.
- Commodity hardware and storage.
- Runtime takes care of splitting and moving data for operations.
- Special distributed file system. Example: Hadoop Distributed File System and Hadoop Runtime.

Classes of problems “mapreducable”

- Benchmark for comparing: Jim Gray’s challenge on data-intensive computing. Ex: “Sort”
- Google uses it (we think) for wordcount, adwords, pagerank, indexing data.
- Simple algorithms such as grep, text-indexing, reverse indexing
- Bayesian classification: data mining domain
- Facebook uses it for various operations: demographics
- Financial services use it for analytics
- Astronomy: Gaussian analysis for locating extra-terrestrial objects.
- Expected to play a critical role in semantic web and web3.0

Scope of MapReduce



Hadoop

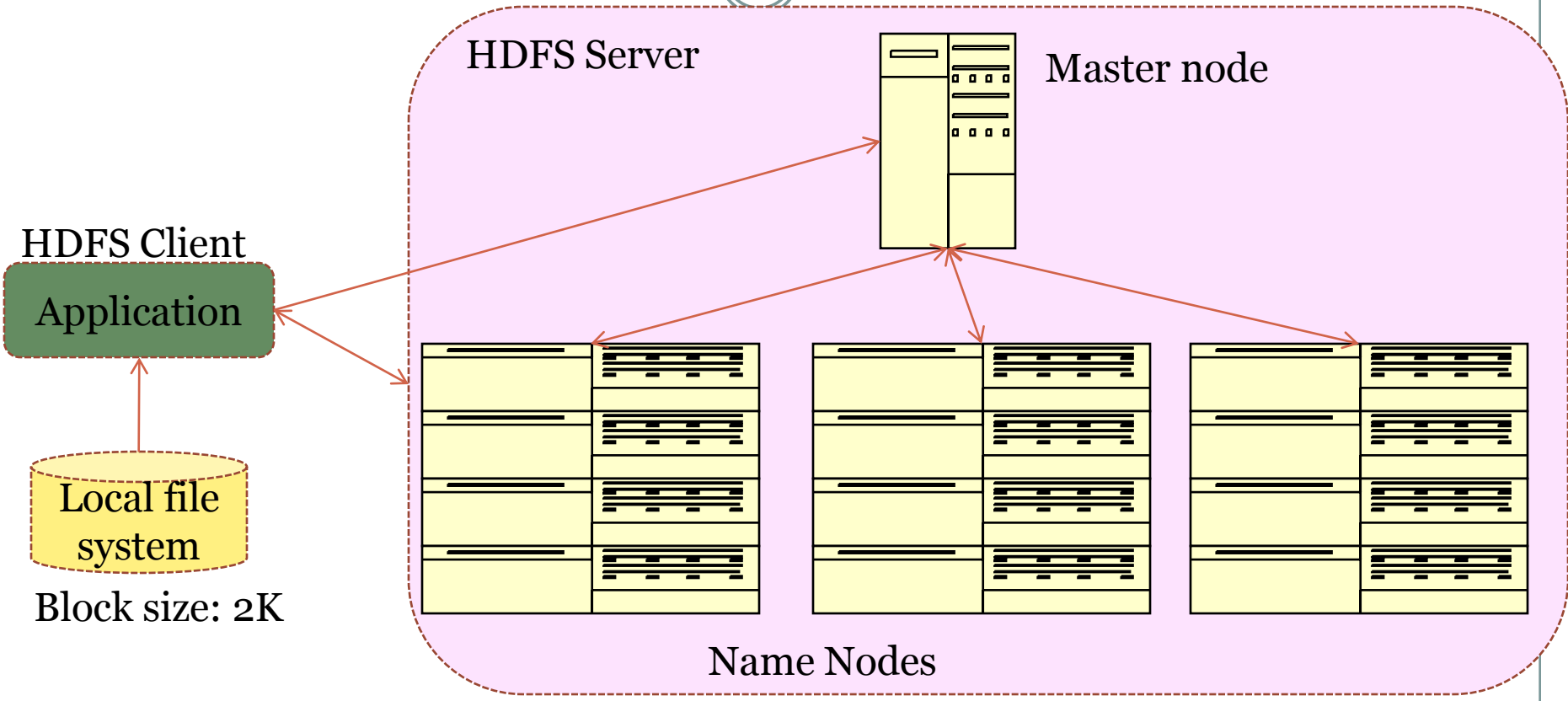
What is Hadoop?

- At Google MapReduce operation are run on a special file system called Google File System (GFS) that is highly optimized for this purpose.
- GFS is not open source.
- Doug Cutting and Yahoo! reverse engineered the GFS and called it Hadoop Distributed File System (HDFS).
- The software framework that supports HDFS, MapReduce and other related entities is called the project Hadoop or simply Hadoop.
- This is open source and distributed by Apache.

Basic Features: HDFS

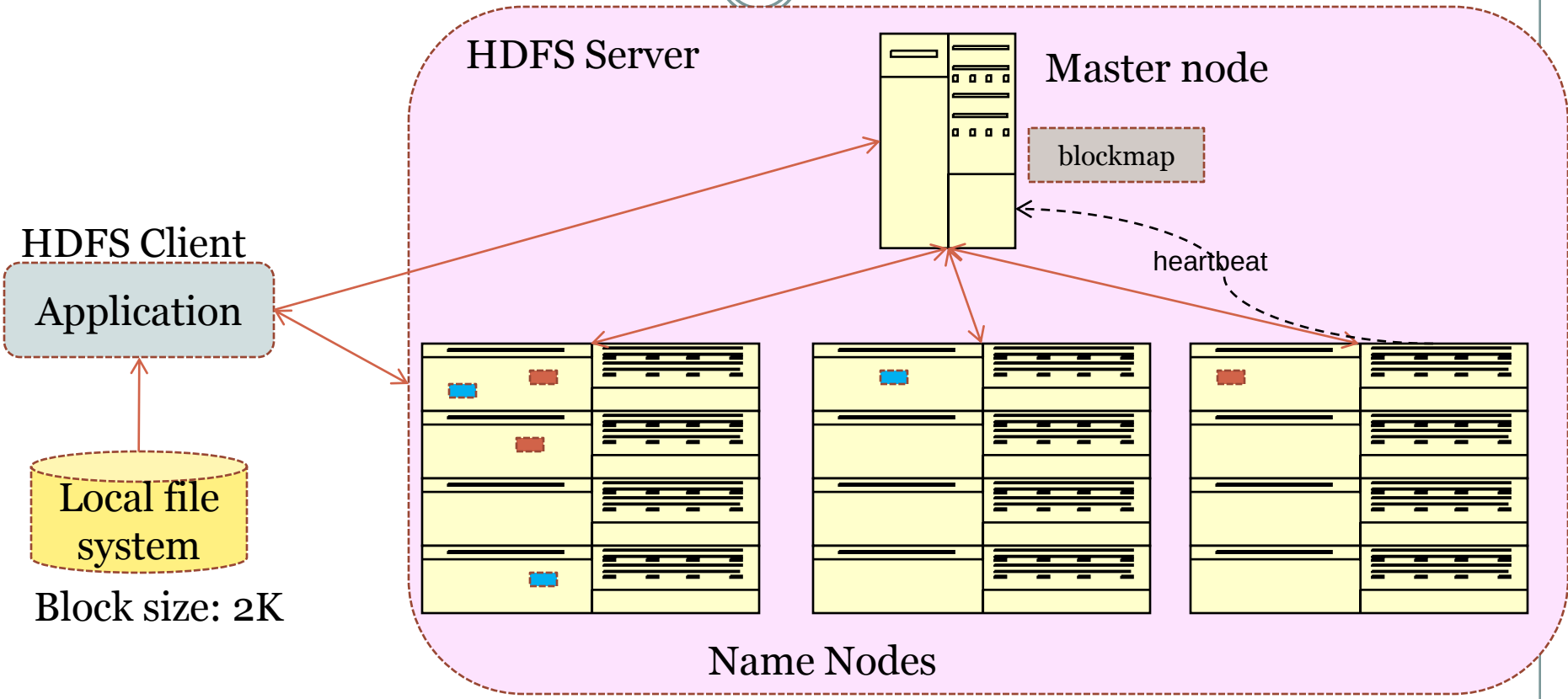
- Highly fault-tolerant
- High throughput
- Suitable for applications with large data sets
- Streaming access to file system data
- Can be built out of commodity hardware

Hadoop Distributed File System



More details: We discuss this in great detail in my Operating Systems course

Hadoop Distributed File System



More details: We discuss this in great detail in my Operating Systems course

Relevance and Impact on Undergraduate courses

- Data structures and algorithms: a new look at traditional algorithms such as sort: Quicksort may not be your choice! It is not easily parallelizable. Merge sort is better.
- You can identify mappers and reducers among your algorithms. Mappers and reducers are simply place holders for algorithms relevant for your applications.
- Large scale data and analytics are indeed concepts to reckon with similar to how we addressed “programming in the large” by OO concepts.
- While a full course on MR/HDFS may not be warranted, the concepts perhaps can be woven into most courses in our CS curriculum.

Demo

- VMware simulated Hadoop and MapReduce demo
- Remote access to NEXOS system at my Buffalo office
- 5-node HDFS running HDFS on Ubuntu 8.04
- 1 –name node and 4 data-nodes
- Each is an old commodity PC with 512 MB RAM, 120GB – 160GB external memory
- Zeus (namenode), datanodes: hermes, dionysus, aphrodite, athena

Summary

- We introduced MapReduce programming model for processing large scale data
- We discussed the supporting Hadoop Distributed File System
- The concepts were illustrated using a simple example
- We reviewed some important parts of the source code for the example.
- Relationship to Cloud Computing

References

1. Apache Hadoop Tutorial: http://hadoop.apache.org http://hadoop.apache.org/core/docs/current/mapred_tutorial.html
2. Dean, J. and Ghemawat, S. 2008. **MapReduce: simplified data processing on large clusters.** *Communication of ACM* 51, 1 (Jan. 2008), 107-113.
3. Cloudera Videos by Aaron Kimball:

   http://www.cloudera.com/hadoop-training-basic
4. http://www.cse.buffalo.edu/faculty/bina/mapreduce.html