Data Intensive Computing

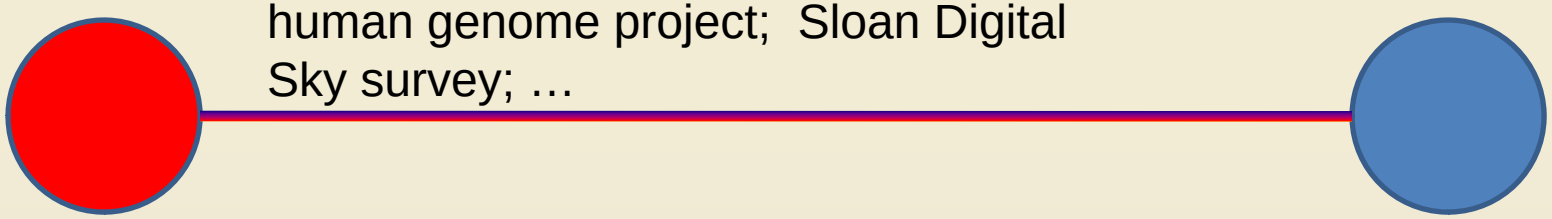
B. Ramamurthy

This work is Partially Supported by
NSF DUE Grant#: 0737243, 0920335

Topics for discussion

- **Problem Space: explosion of data**
- **Solution space: emergence of multi-core, virtualization, cloud computing**
- **Inability of traditional file system to handle data deluge**
- **Emerging systems: Google File System (GFS)**
- **Salient features of GFS**
- **The Big-data Computing Model**
  - **MapReduce Programming Model (Algorithm)**
  - **Hadoop Distributed File System (Data Structure)**
- **Cloud Computing and its Relevance to Big-data and Data-intensive computing (next class)**

Data-Computation Continuum

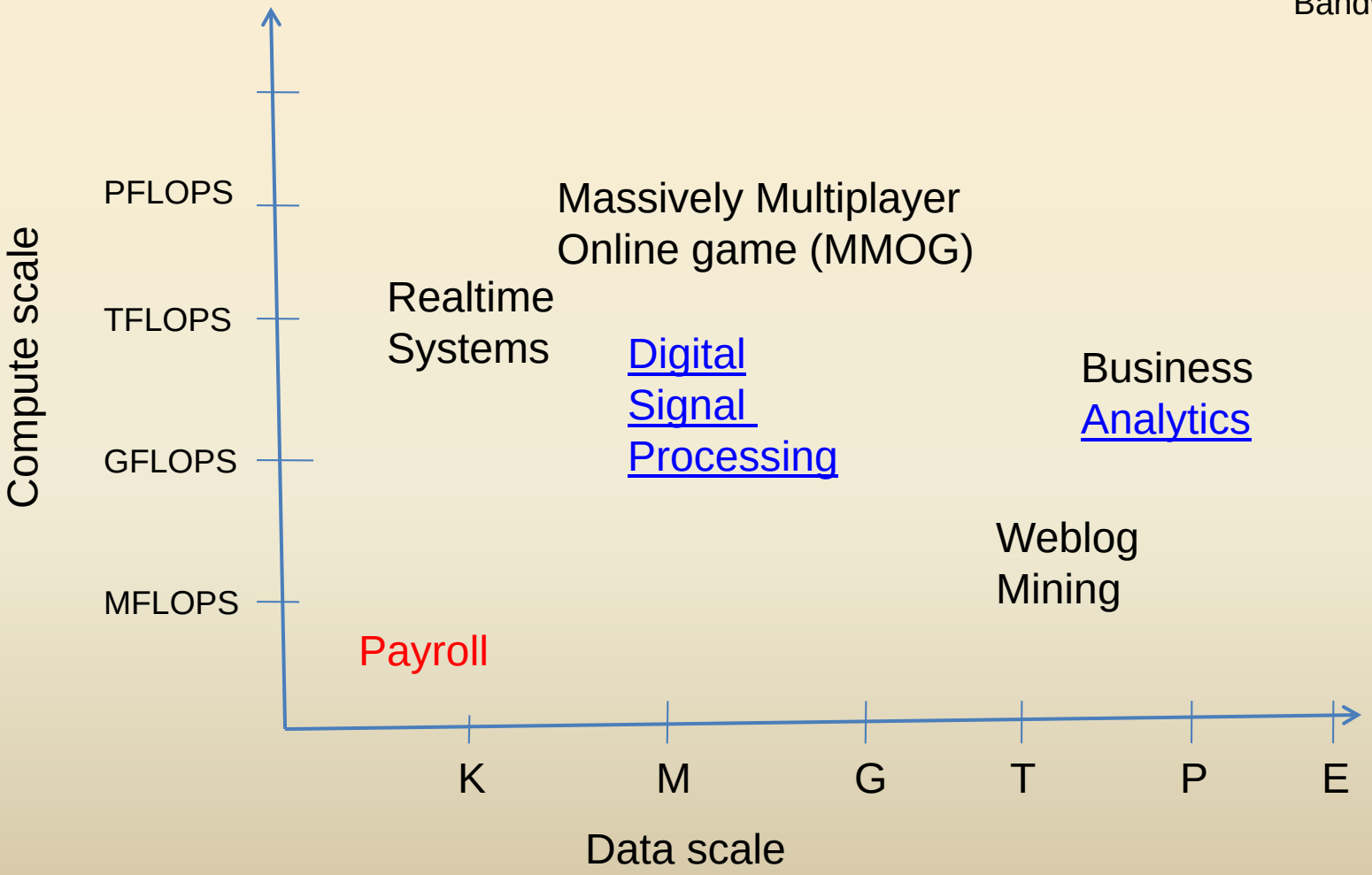


Compute intensive
Ex: computation of digits of PI
http://bellard.org/pi/pi2700e9/pipcrecord.pdf

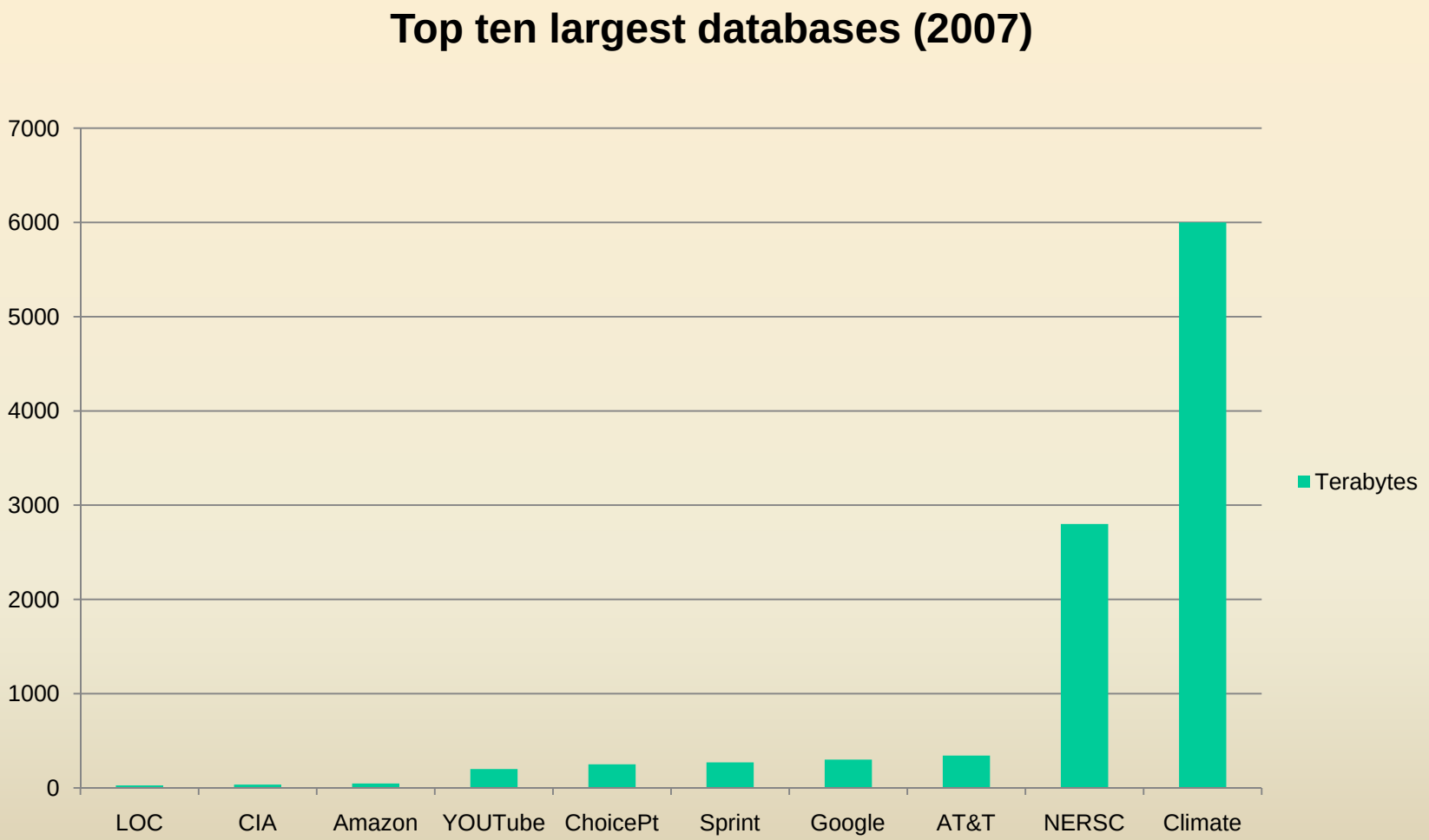
Data intensive
Ex: analyzing web pages
& index them

More dimensions

Other variables:
Communication
Bandwidth, ?



Top Ten Largest Databases



Ref: http://www.businessintelligencelowdown.com/2007/02/top_10_largest_.html

Solution Processing Granularity

Data size: small

Pipelined Instruction level

Concurrent Thread level

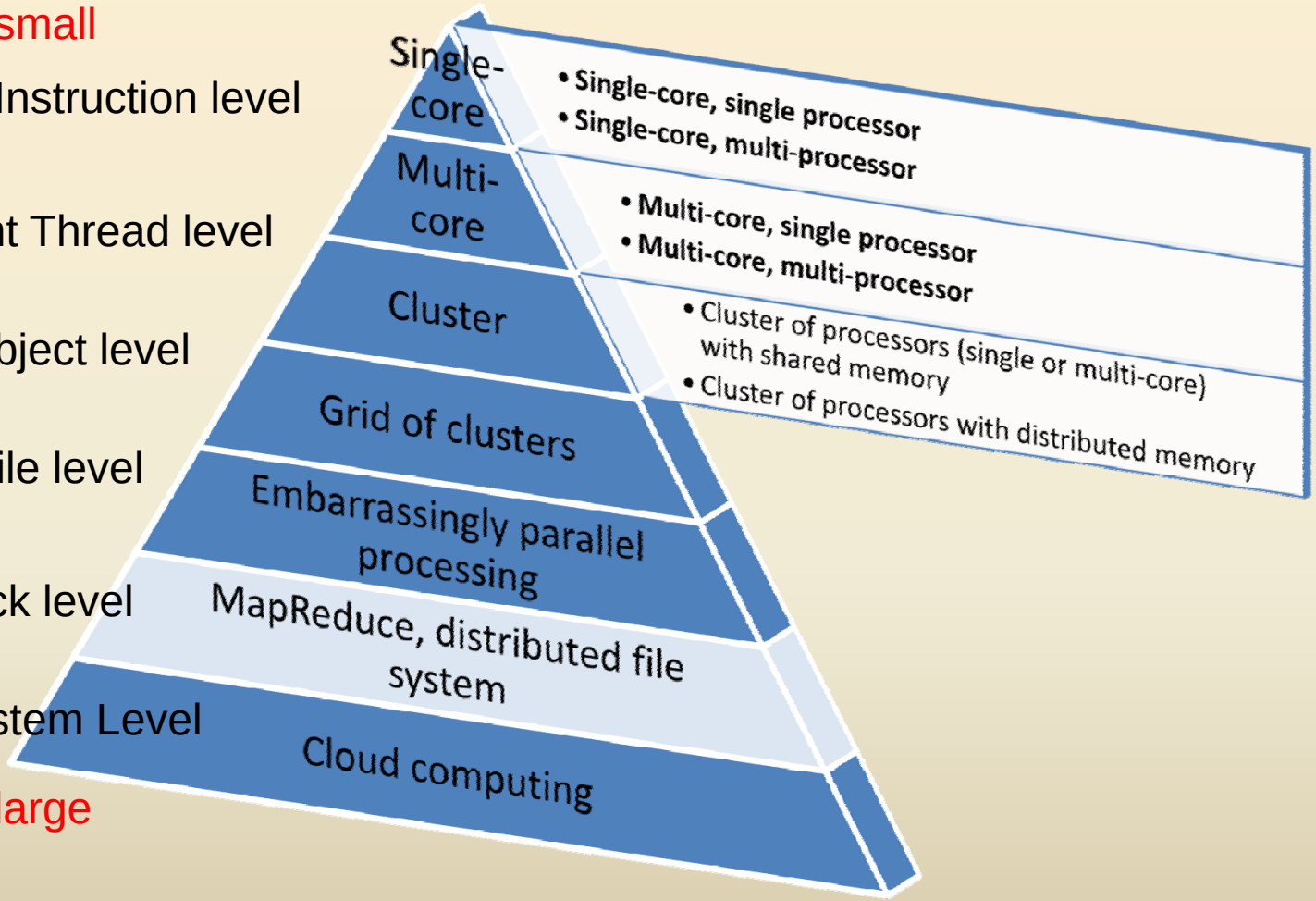
Service Object level

Indexed File level

Mega Block level

Virtual System Level

Data size: large



Traditional Storage Solutions

Off system/online storage/ secondary memory

File system abstraction

Offline/ tertiary memory

RAID: Redundant Array of Inexpensive Disks

NAS: Network Accessible Storage

SAN: Storage area networks

Database and Database Management System

- Data source
- Transactional
- Data base server
- Relational db or similar foundation
- Tables, rows, result set, SQL
- ODBC: open data base connectivity
- Very successful business model: Oracle, DB2, MySQL, and others
- Persistence models: EJB, DAO, ADO (look up the abbreviation in any of enterprise model documentation you are working with)

Distributed file system(DFS)

- A dedicated server manages the files for an compute environment
- For example, nickelback,cse.buffalo.edu is your file server and that is why we did not want you to run your user applications on this machine.
- DFS addresses various transparencies: location transparency, sharing, performance etc.
- Single largest file is approximately few terabytes with typical page size of 4-8K.
- What is the page table size for the largest file? Ex: 16T/8K = ~8G

Emerging Systems

On to Google File

- Internet introduced a new challenge in the form web logs, web crawler's data: large scale "peta scale"
- But observe that this type of data has an uniquely different characteristic than your transactional or the "order" data on amazon.com: "write once" ;
  - HIPPA protected healthcare and patient information;
  - Historical financial data;
  - Any historical data
- Google exploited this characteristics in its Google file system: S. Ghemavat

Data Characteristics

- Streaming data access
- Applications need streaming access to data
- Batch processing rather than interactive user access.
- Large data sets and files: gigabytes to terabytes size
- High aggregate data bandwidth
- Scale to hundreds of nodes in a cluster
- Tens of millions of files in a single instance
- Write-once-read-many: a file once created, written and closed need not be changed – this assumption simplifies coherency
- WORM inspired a new programming model called the MapReduce programming model

The Big-data Computing 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
  - Complex computational models
  - No single environment is good enough: need elastic, on-demand capacities
- We are looking at newer
  - programming models, and
  - Supporting algorithms and data structures.
- NSF refers to this area as “data-intensive computing” and industry calls it “big-data” and “cloud computing”

Goals Of this Discussion

- 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 students to explore ways to infuse relevant concepts of this emerging area into their application development

The Outline

- Introduction to MapReduce
- From CS Foundation to MapReduce
- MapReduce programming model
- Demo of MapReduce on Virtualized hardware
- Hadoop Distributed File System
- 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,…}

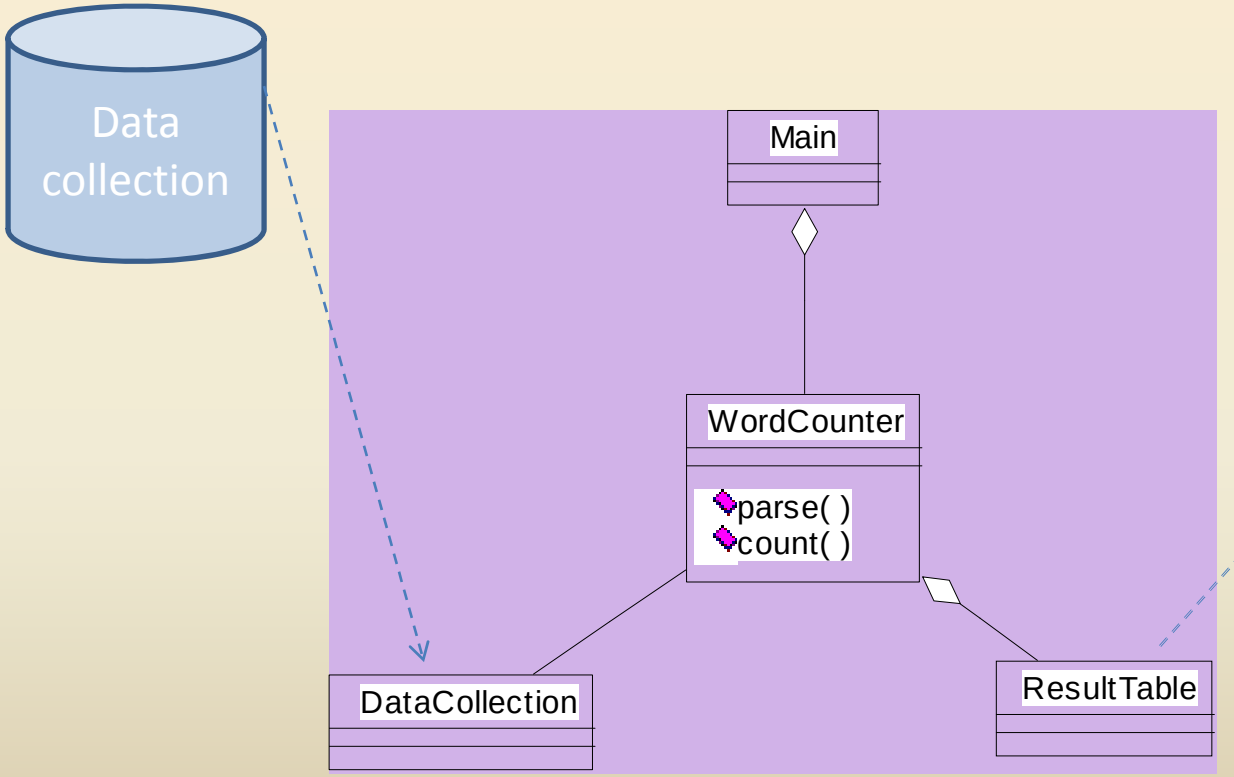
Problem: Count the occurrences of the different words in the collection.

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

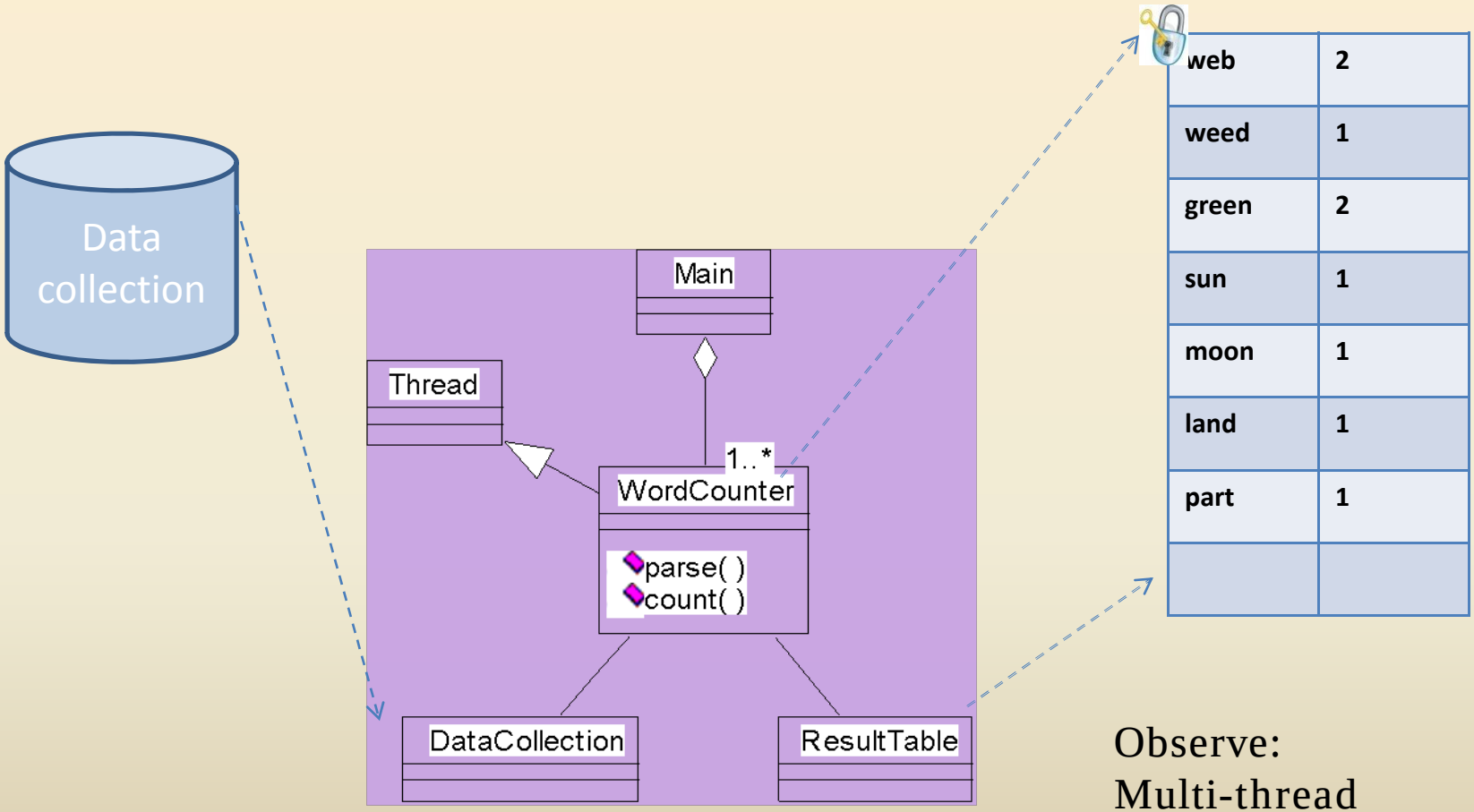
Word Counter and Result Table

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



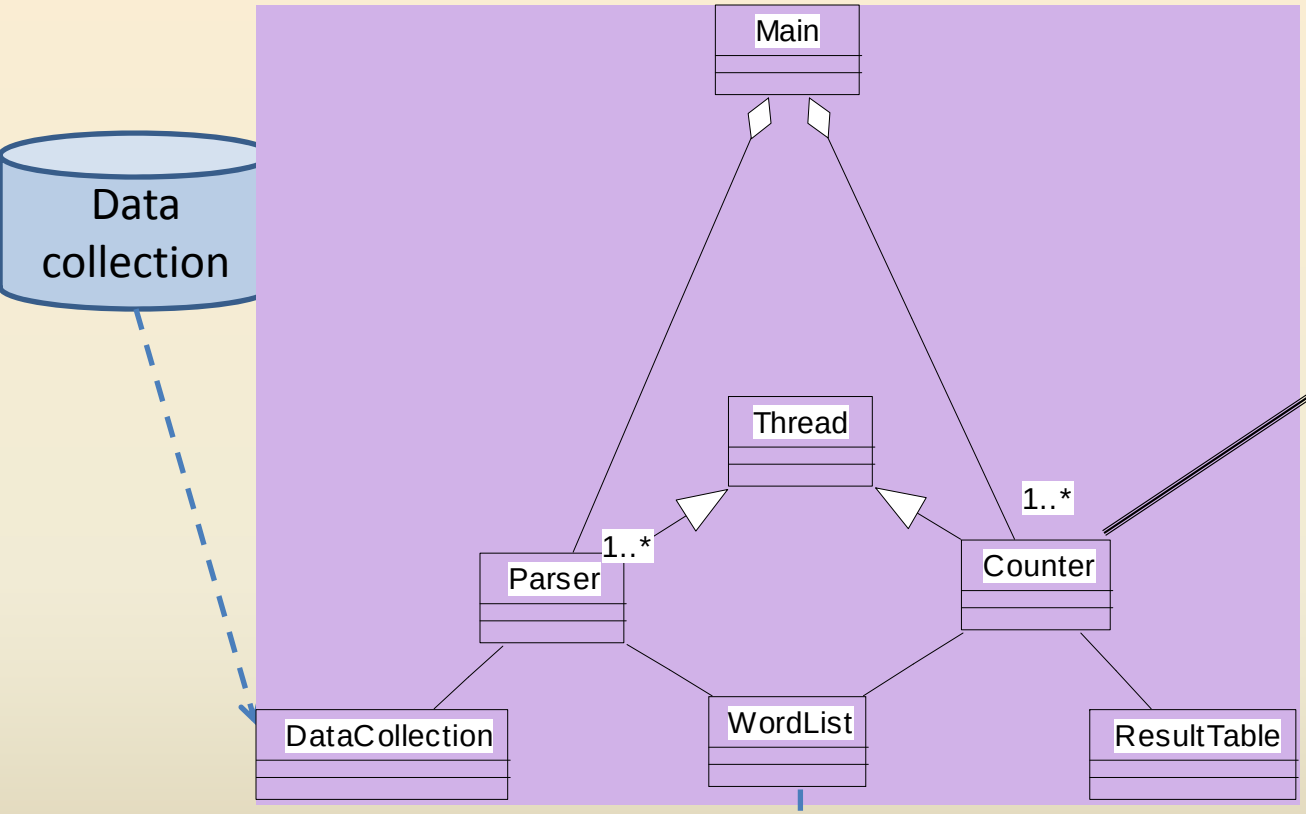
web	**2**
weed	**1**
green	**2**
sun	**1**
moon	**1**
land	**1**
part	**1**

Multiple Instances of Word Counter



Observe:
Multi-thread
Lock on shared data

Improve Word Counter for Performance



No need for lock

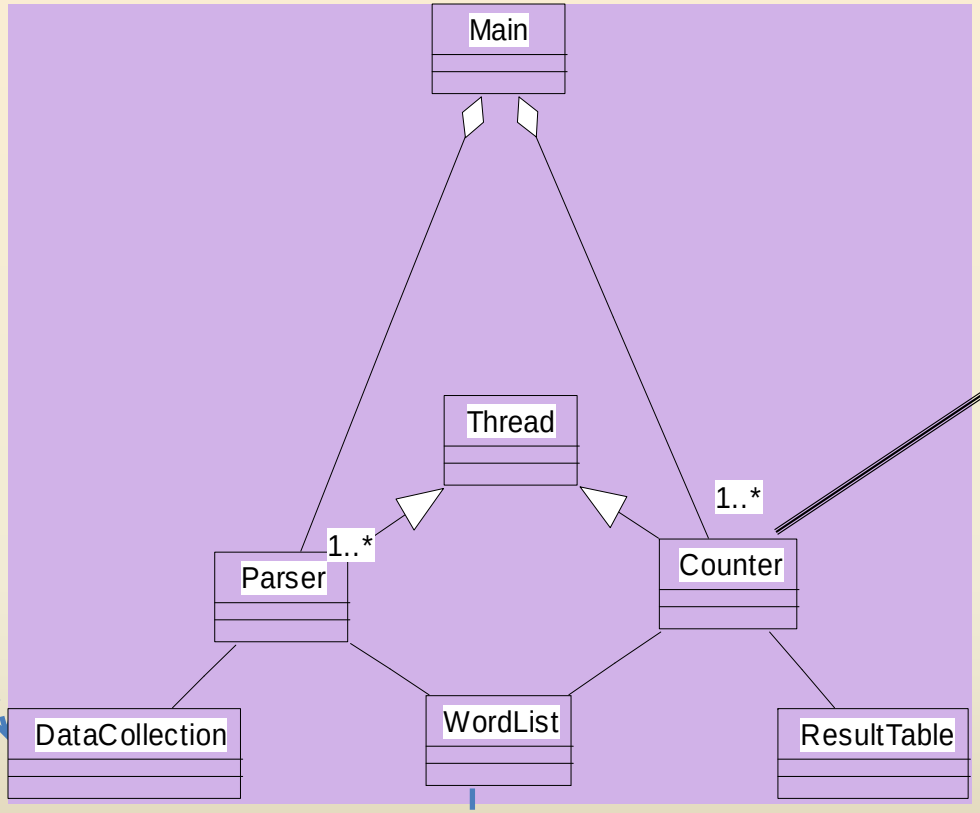
web	2
weed	1
green	2
sun	1
moon	1
land	1
part	1

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										

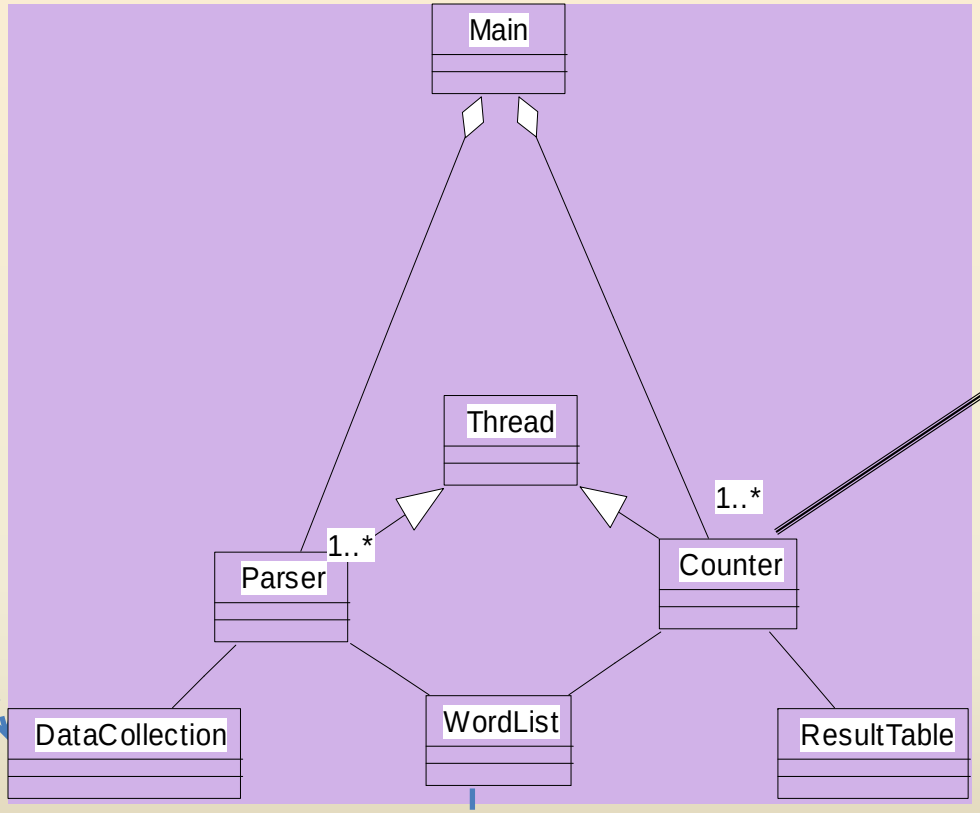
Addressing the Scale Issue

- 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

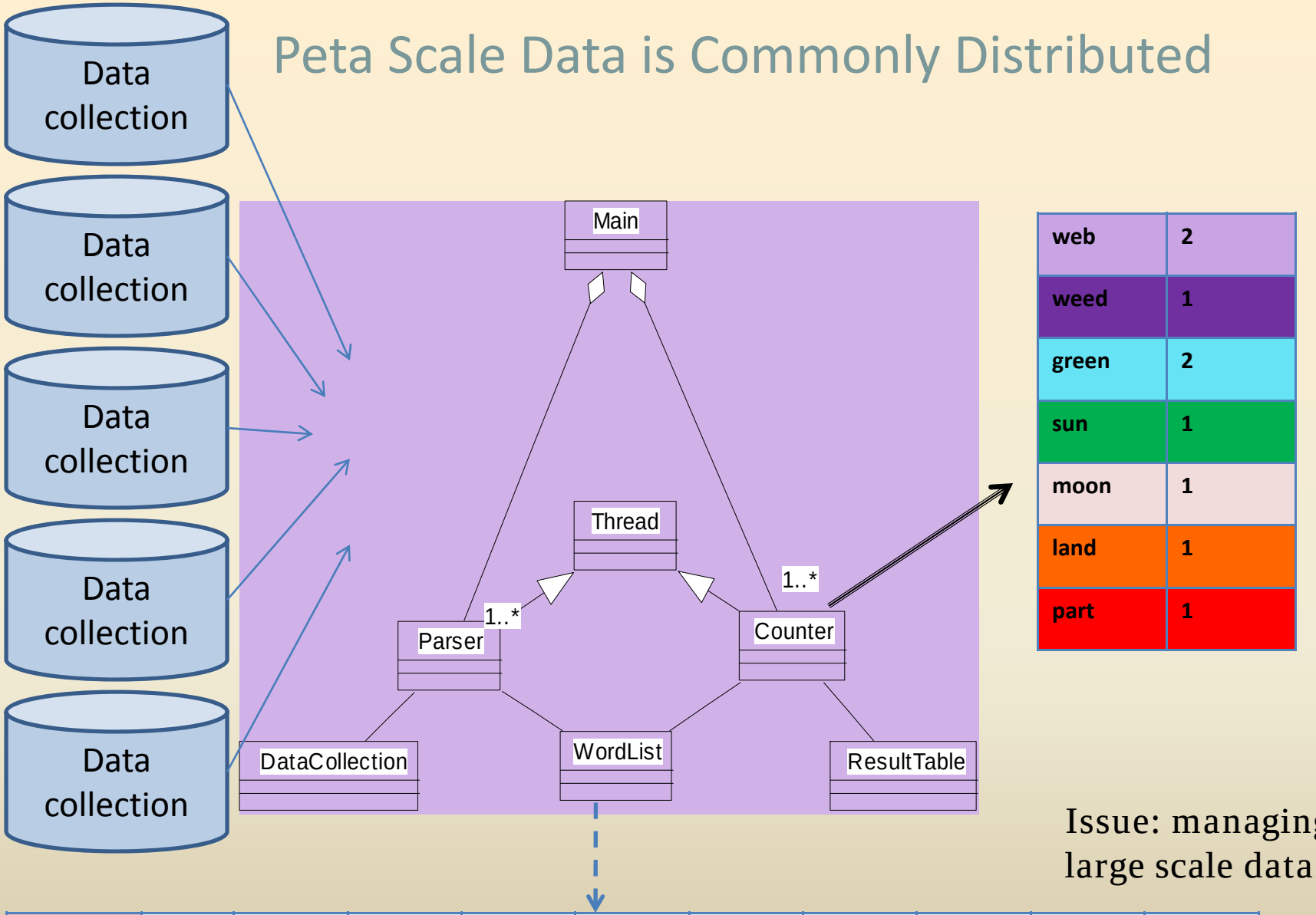
Data collection



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

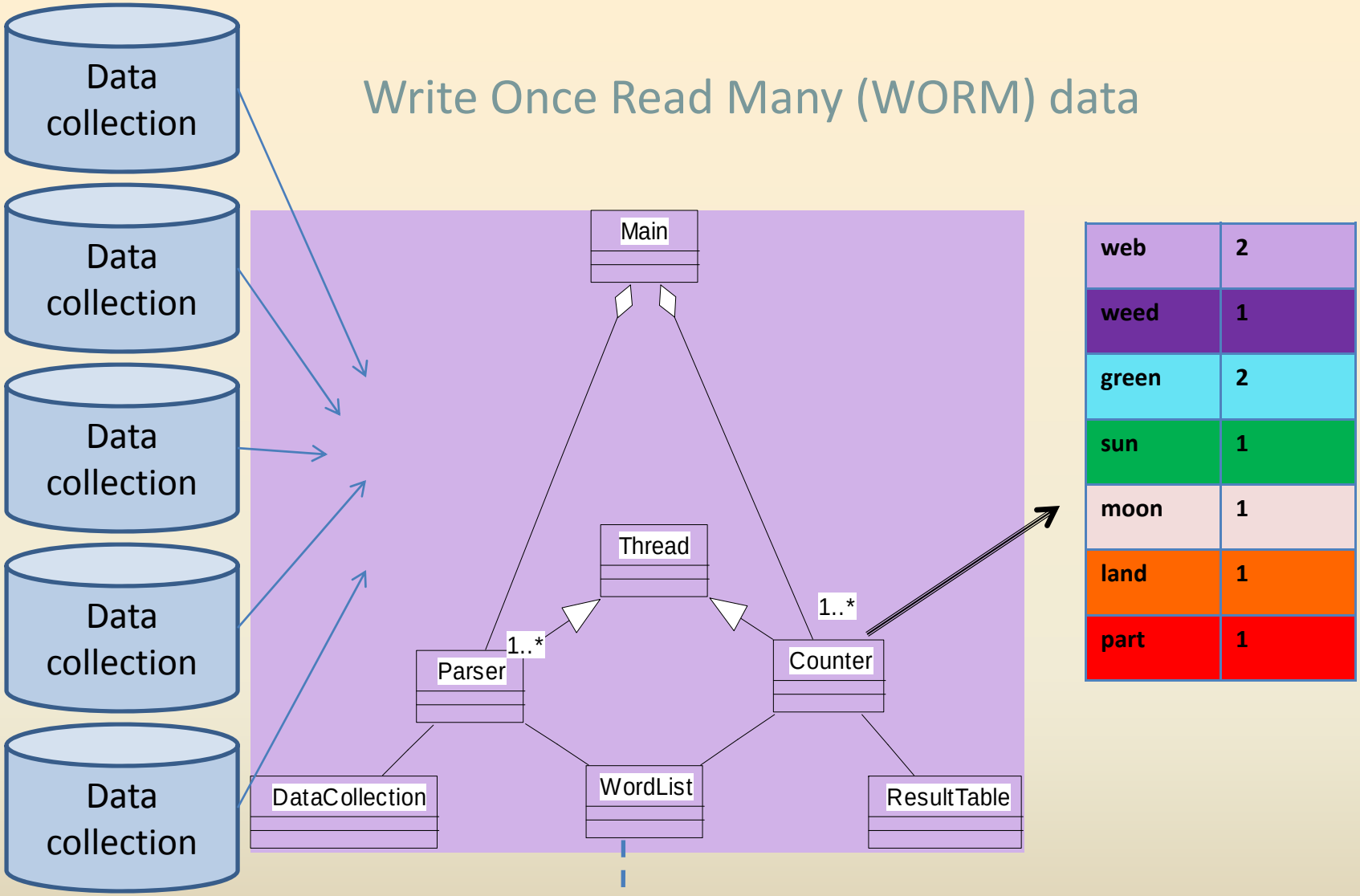


web	2
weed	1
green	2
sun	1
moon	1
land	1
part	1

Issue: managing the large scale data

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

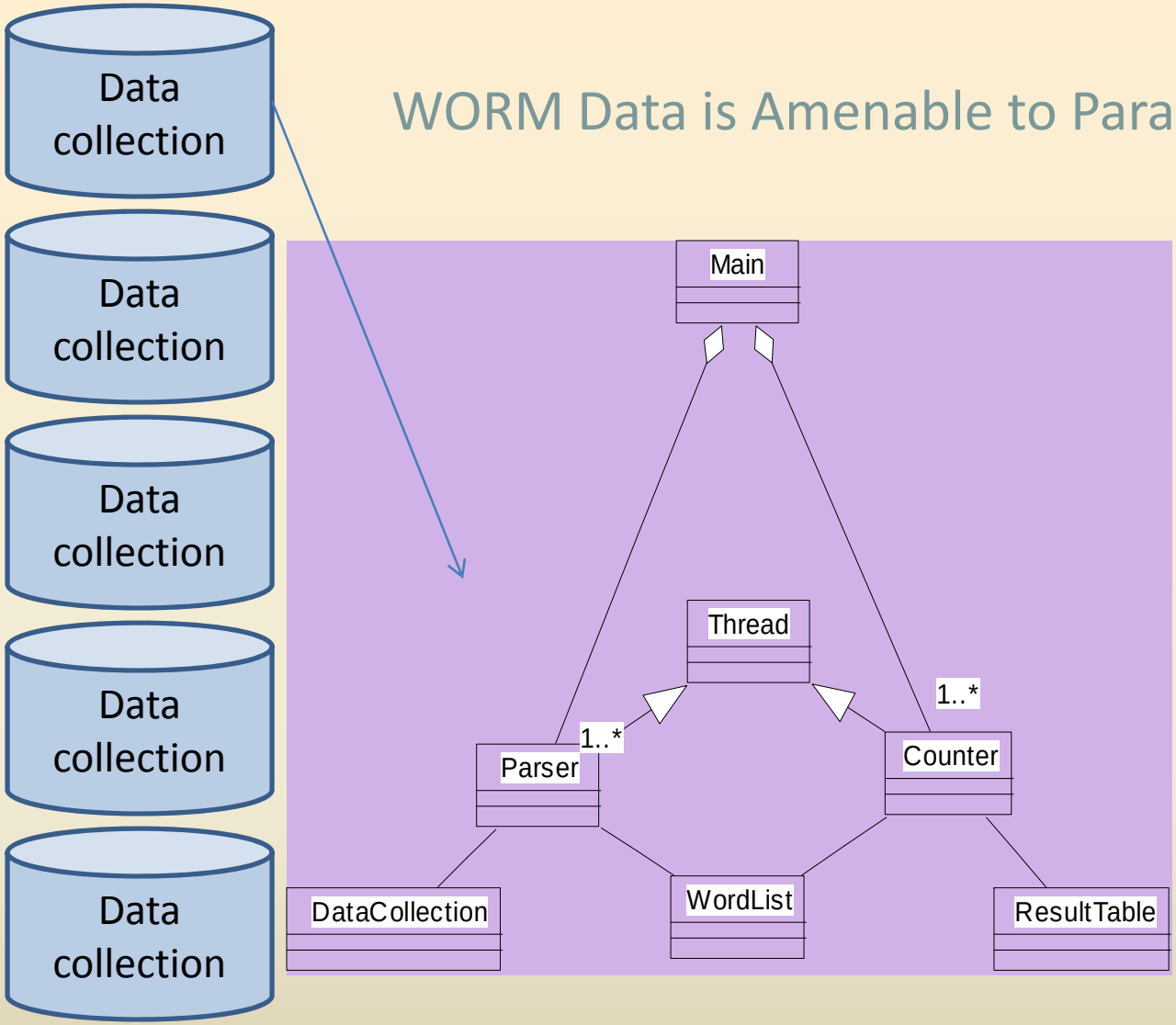
Write Once Read Many (WORM) data



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

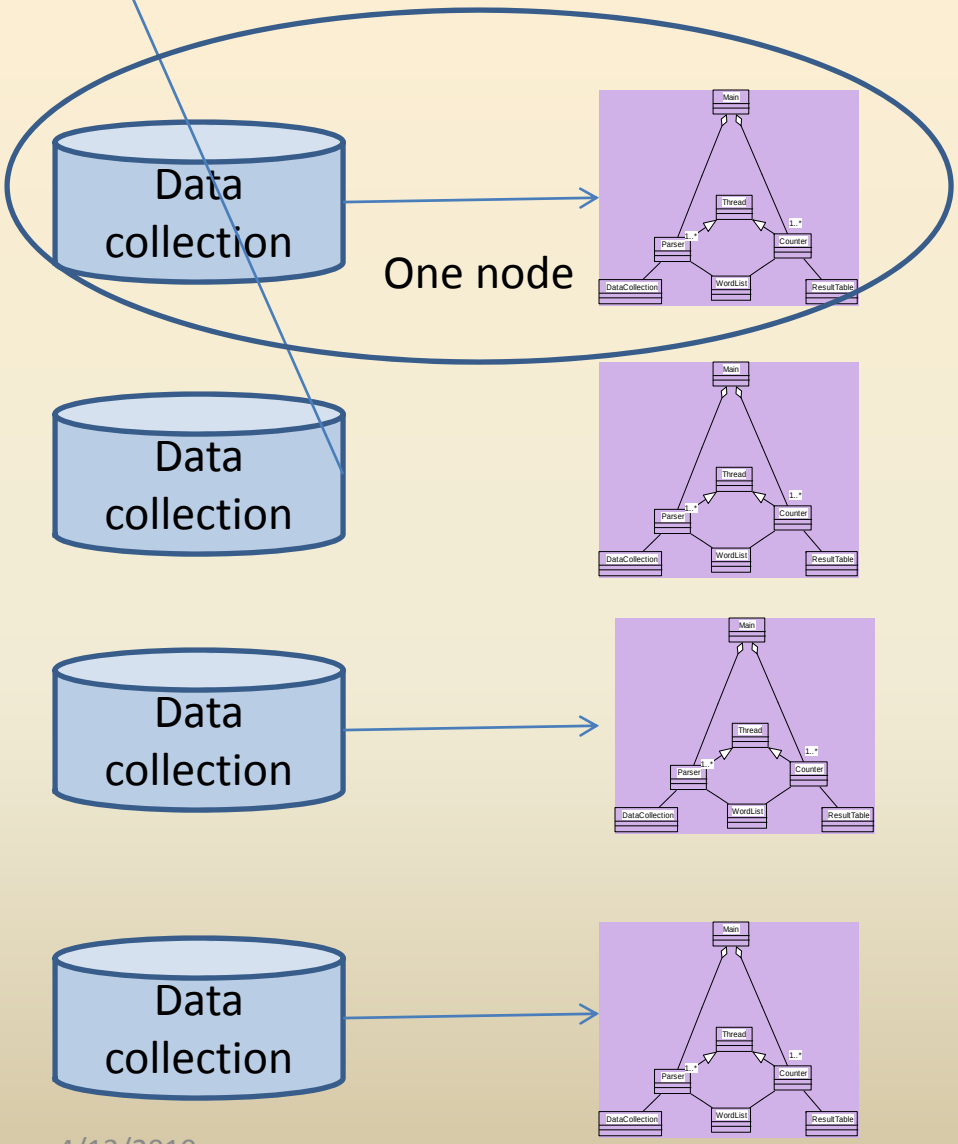
KEY	web	weed	green	sun	moon	land	part	web	green	…….
VALUE										

WORM Data is Amenable to Parallelism



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:

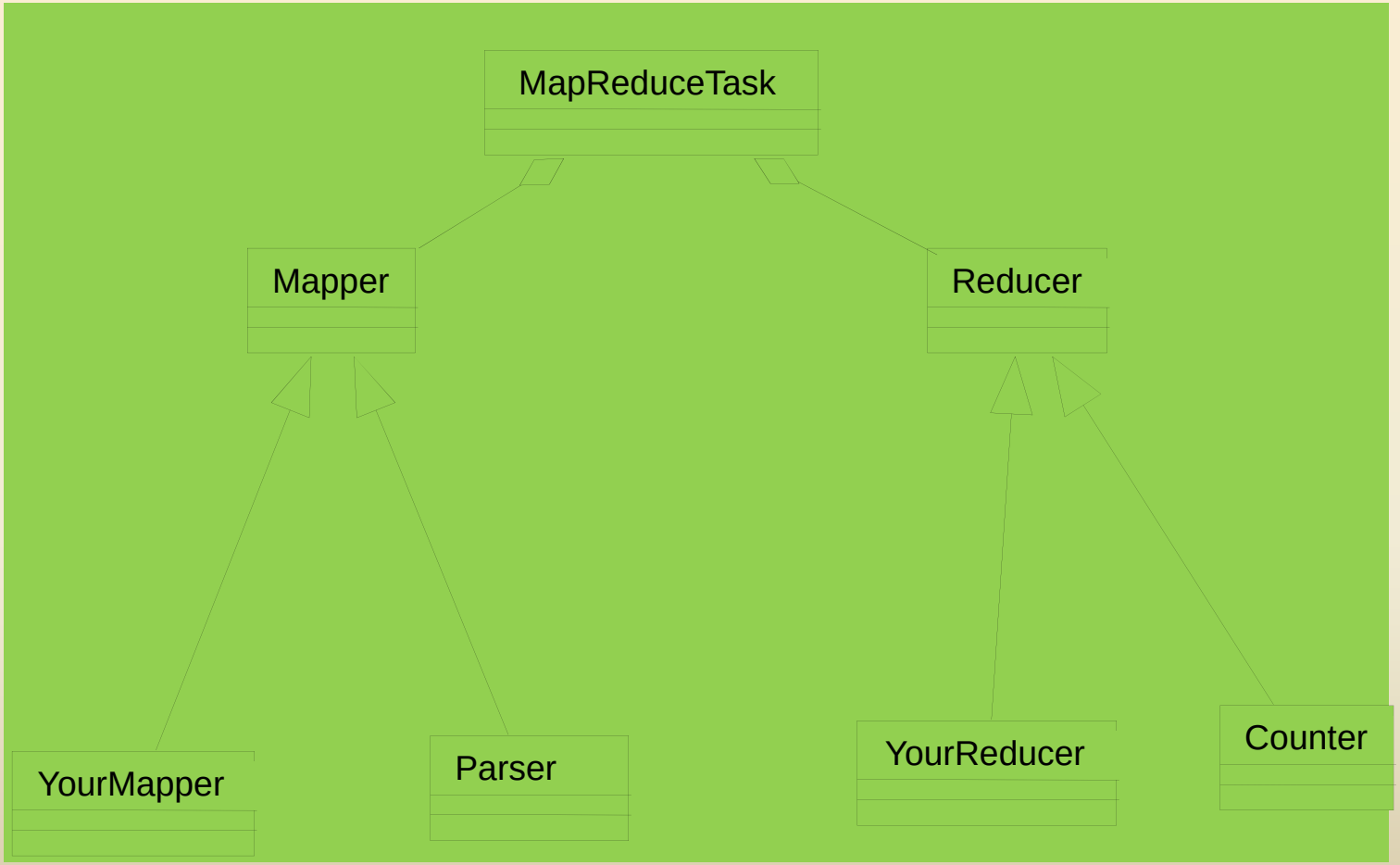
Our parse is a mapping operation:
MAP: input → <key, value> pairs

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!

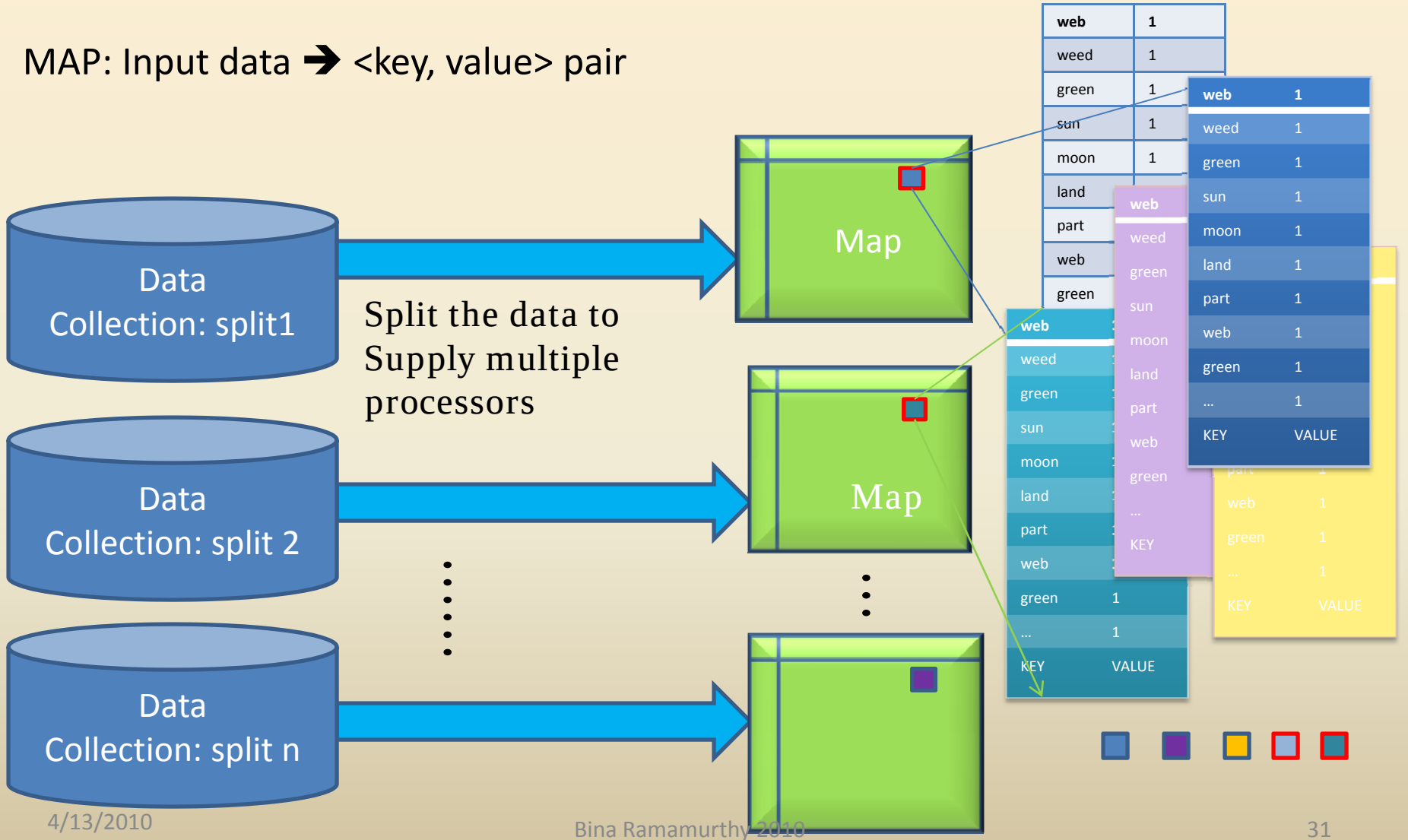
Mapper and Reducer



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

Map Operation

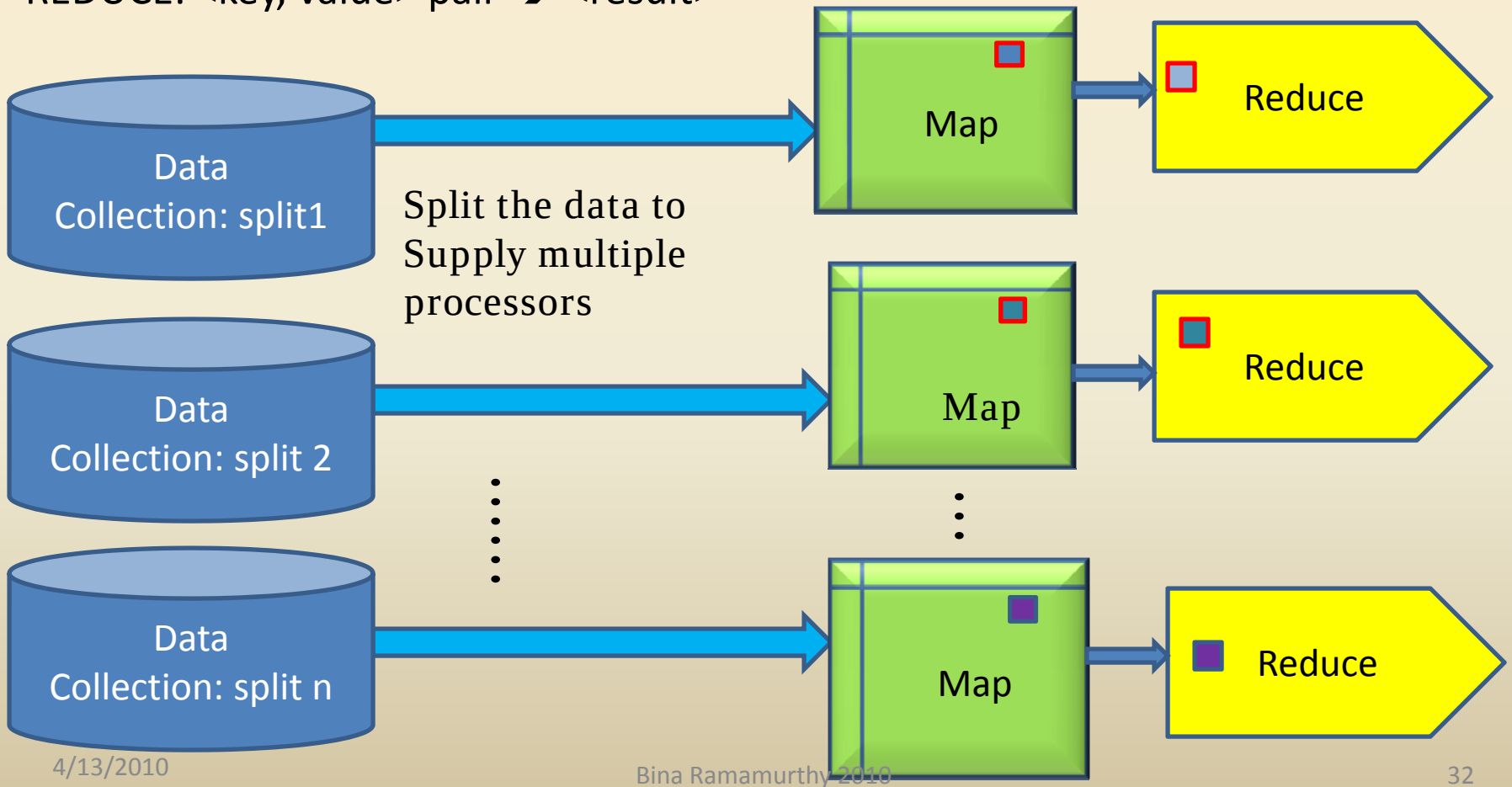
MAP: Input data ➔ <key, value> pair

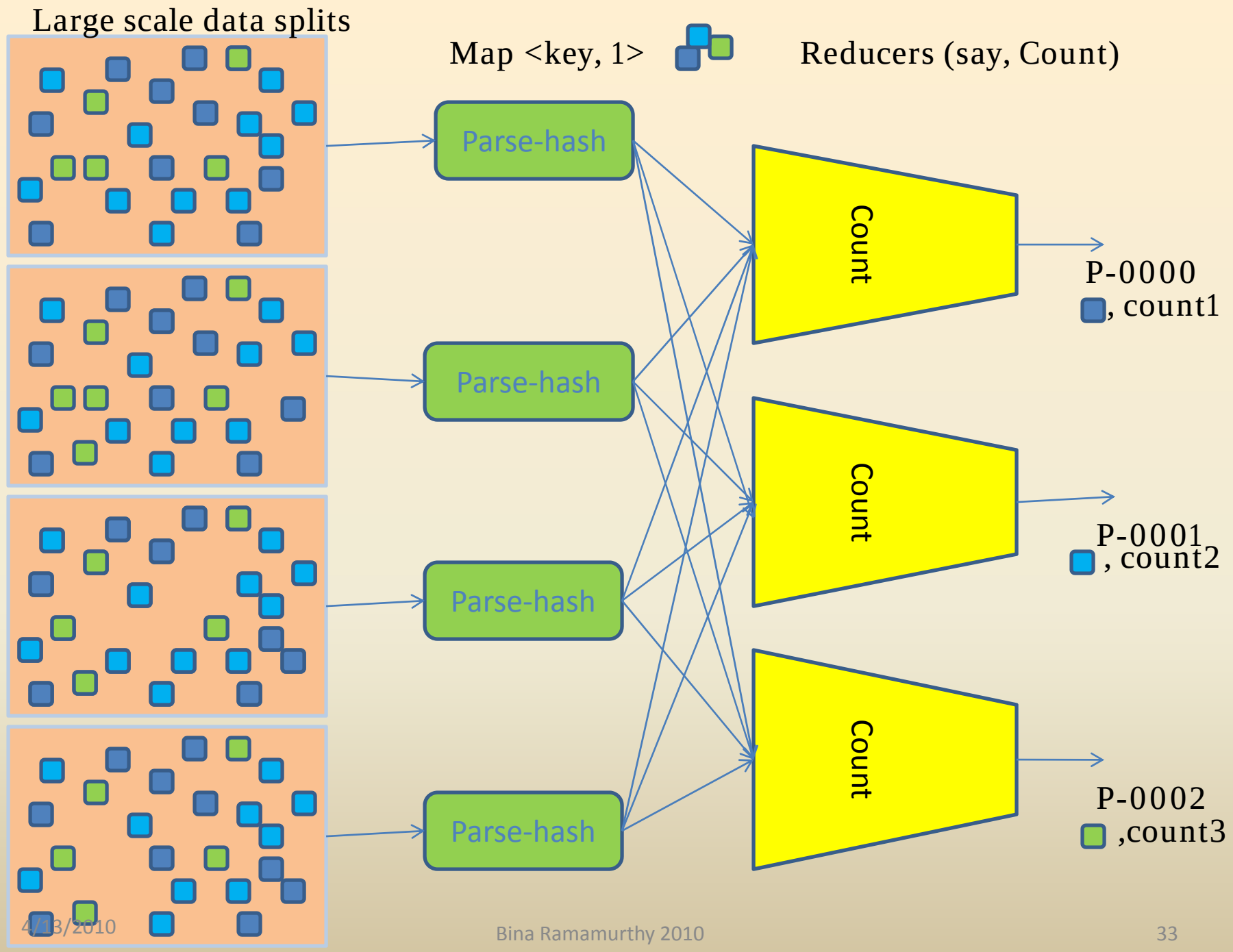


Reduce Operation

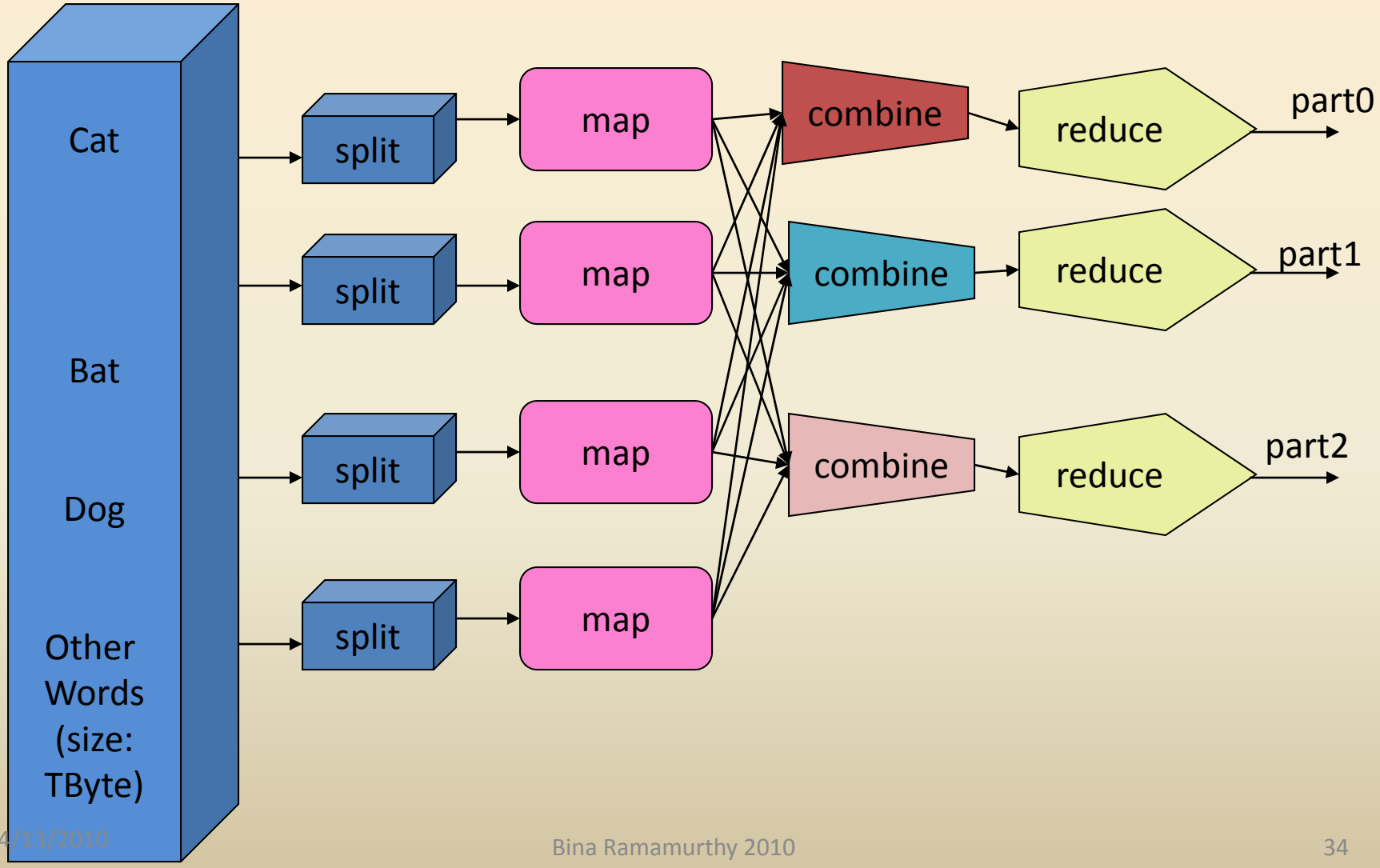
MAP: Input data ➔ <key, value> pair

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

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.

Reference

- The Hadoop Distributed File System: Architecture and Design by Apache Foundation Inc.

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

Fault tolerance

- Failure is the norm rather than exception
- A HDFS instance may consist of thousands of server machines, each storing part of the file system's data.
- Since we have huge number of components and that each component has non-trivial probability of failure means that there is always some component that is non-functional.
- Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

Data Characteristics

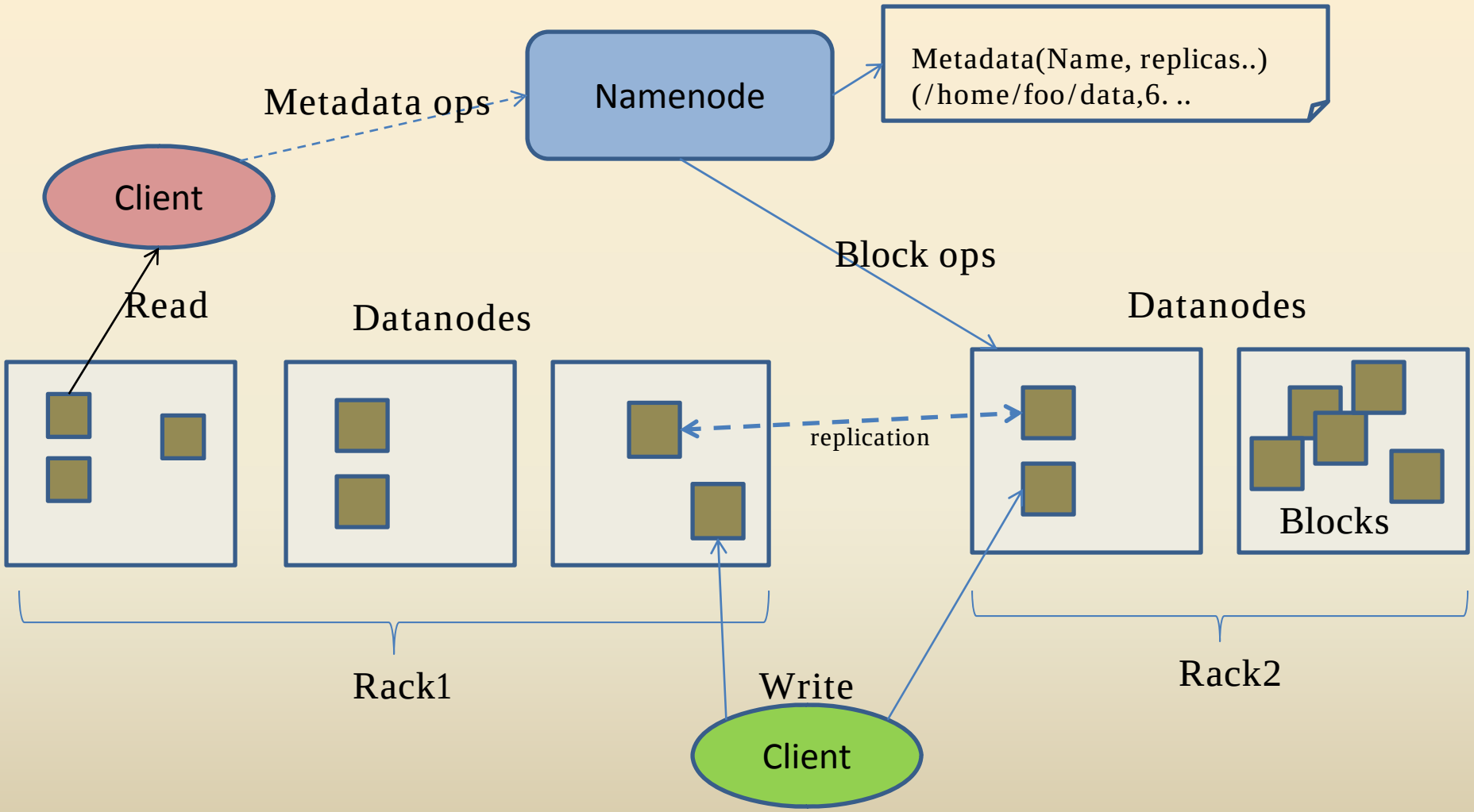
- Streaming data access
- Applications need streaming access to data
- Batch processing rather than interactive user access.
- Large data sets and files: gigabytes to terabytes size
- High aggregate data bandwidth
- Scale to hundreds of nodes in a cluster
- Tens of millions of files in a single instance
- Write-once-read-many: a file once created, written and closed need not be changed – this assumption simplifies coherency
- A map-reduce application or web-crawler application fits perfectly with this model.

ARCHITECTURE

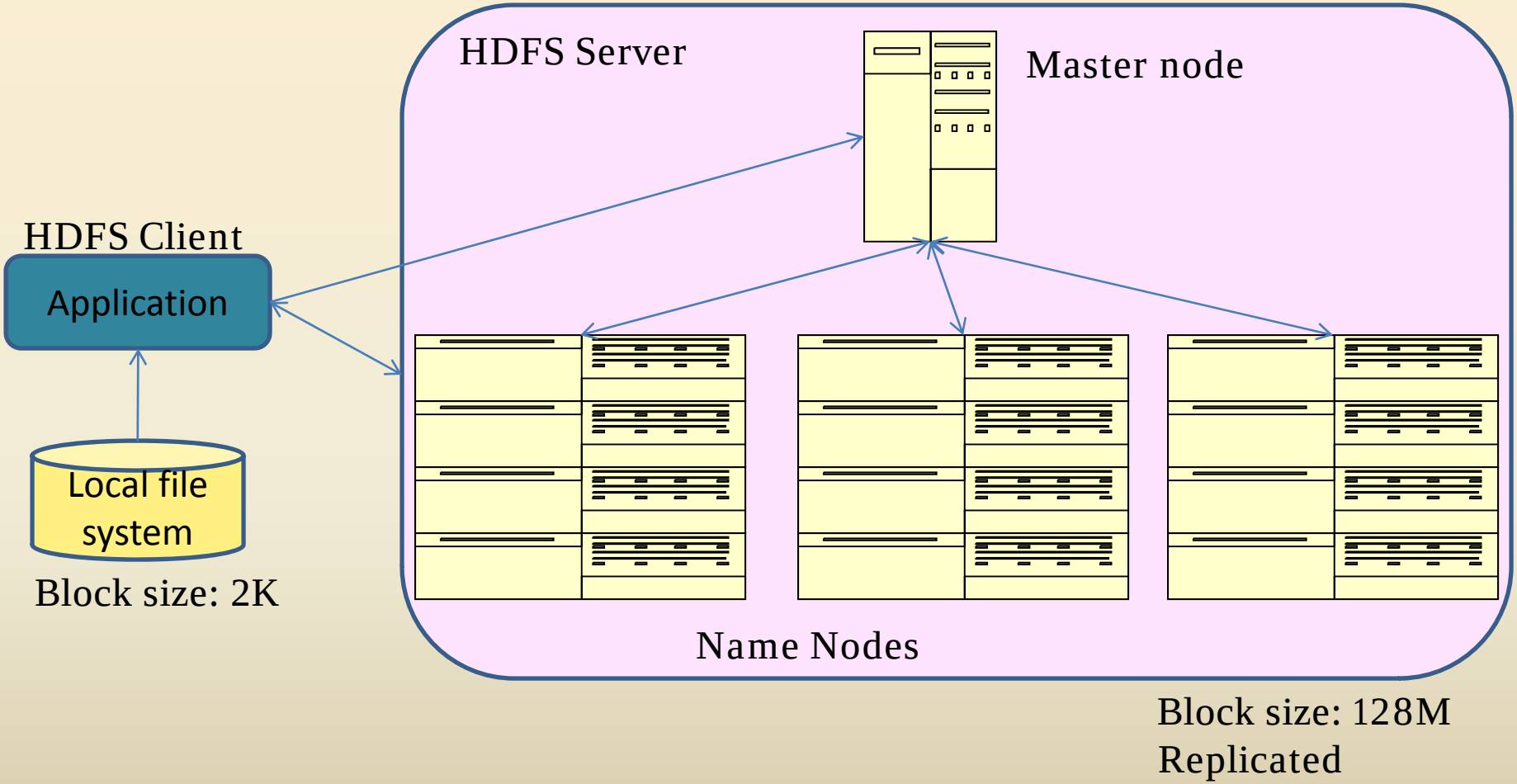
Namenode and Datanodes

- Master/slave architecture
- HDFS cluster consists of a single **Namenode**, a master server that manages the file system namespace and regulates access to files by clients.
- There are a number of **DataNodes** usually one per node in a cluster.
- The DataNodes manage storage attached to the nodes that they run on.
- HDFS exposes a file system namespace and allows user data to be stored in files.
- A file is split into one or more blocks and set of blocks are stored in DataNodes.
- DataNodes: serves read, write requests, performs block creation, deletion, and replication upon instruction from Namenode.

HDFS Architecture



Hadoop Distributed File System



File system Namespace

- Hierarchical file system with directories and files
- Create, remove, move, rename etc.
- Namenode maintains the file system
- Any meta information changes to the file system recorded by the Namenode.
- An application can specify the number of replicas of the file needed: replication factor of the file. This information is stored in the Namenode.

Data Replication

- HDFS is designed to store very large files across machines in a large cluster.
- Each file is a sequence of blocks.
- All blocks in the file except the last are of the same size.
- Blocks are replicated for fault tolerance.
- Block size and replicas are configurable per file.
- The Namenode receives a Heartbeat and a BlockReport from each DataNode in the cluster.
- BlockReport contains all the blocks on a Datanode.

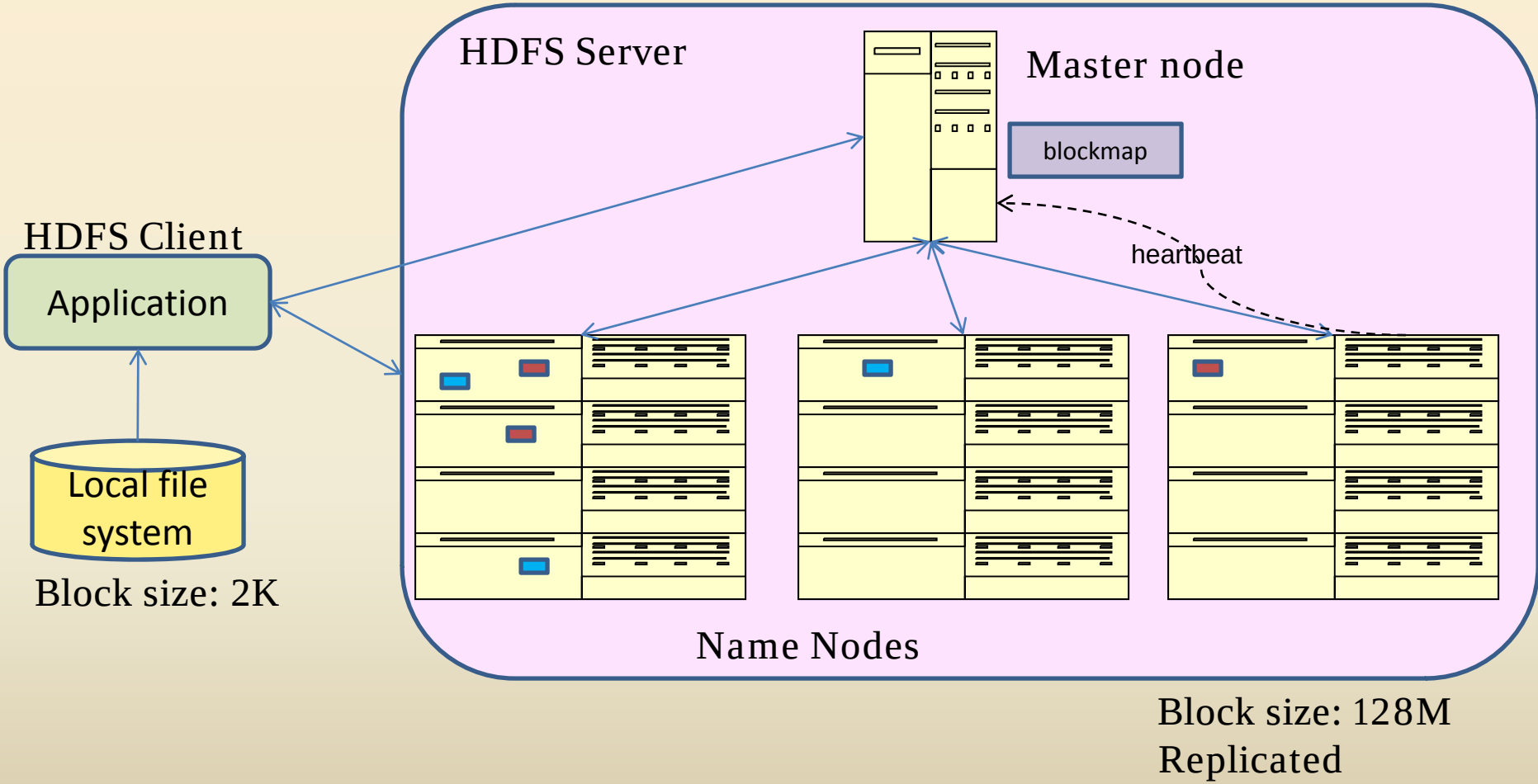
Replica Placement

- The placement of the replicas is critical to HDFS reliability and performance.
- Optimizing replica placement distinguishes HDFS from other distributed file systems.
- Rack-aware replica placement:
  - Goal: improve reliability, availability and network bandwidth utilization
  - Research topic
- Many racks, communication between racks are through switches.
- Network bandwidth between machines on the same rack is greater than those in different racks.
- Namenode determines the rack id for each DataNode.
- Replicas are typically placed on unique racks
  - Simple but non-optimal
  - Writes are expensive
  - Replication factor is 3
  - Another research topic?
- Replicas are placed: one on a node in a local rack, one on a different node in the local rack and one on a node in a different rack.
- 1/3 of the replica on a node, 2/3 on a rack and 1/3 distributed evenly across remaining racks.

Replica Selection

- Replica selection for READ operation: HDFS tries to minimize the bandwidth consumption and latency.
- If there is a replica on the Reader node then that is preferred.
- HDFS cluster may span multiple data centers: replica in the local data center is preferred over the remote one.

Hadoop Distributed File System



Safemode Startup

- On startup Namenode enters Safemode.
- Replication of data blocks do not occur in Safemode.
- Each DataNode checks in with Heartbeat and BlockReport.
- Namenode verifies that each block has acceptable number of replicas
- After a configurable percentage of safely replicated blocks check in with the Namenode, Namenode exits Safemode.
- It then makes the list of blocks that need to be replicated.
- Namenode then proceeds to replicate these blocks to other Datanodes.

Filesystem Metadata

- The HDFS namespace is stored by Namenode.
- Namenode uses a transaction log called the EditLog to record every change that occurs to the filesystem meta data.
  - For example, creating a new file.
  - Change replication factor of a file
  - EditLog is stored in the Namenode's local filesystem
- Entire filesystem namespace including mapping of blocks to files and file system properties is stored in a file FsImage. Stored in Namenode's local filesystem.

Namenode

- Keeps image of entire file system namespace and file Blockmap in memory.
- 4GB of local RAM is sufficient to support the above data structures that represent the huge number of files and directories.
- When the Namenode starts up it gets the FsImage and Editlog from its local file system, update FsImage with EditLog information and then stores a copy of the FsImage on the filesytstem as a checkpoint.
- Periodic checkpointing is done. So that the system can recover back to the last checkpointed state in case of a crash.

Datanode

- A Datanode stores data in files in its local file system.
- Datanode has no knowledge about HDFS filesystem
- It stores each block of HDFS data in a separate file.
- Datanode does not create all files in the same directory.
- It uses heuristics to determine optimal number of files per directory and creates directories appropriately:
  - Research issue?
- When the filesystem starts up it generates a list of all HDFS blocks and send this report to Namenode: Blockreport.

PROTOCOL

The Communication Protocol

- All HDFS communication protocols are layered on top of the TCP/IP protocol
- A client establishes a connection to a configurable TCP port on the Namenode machine. It talks ClientProtocol with the Namenode.
- The Datanodes talk to the Namenode using Datanode protocol.
- RPC abstraction wraps both ClientProtocol and Datanode protocol.
- Namenode is simply a server and never initiates a request; it only responds to RPC requests issued by DataNodes or clients.

ROBUSTNESS

Objectives

- Primary objective of HDFS is to store data reliably in the presence of failures.
- Three common failures are: Namenode failure, Datanode failure and network partition.

DataNode failure and heartbeat

- A network partition can cause a subset of Datanodes to lose connectivity with the Namenode.
- Namenode detects this condition by the absence of a Heartbeat message.
- Namenode marks Datanodes without Hearbeat and does not send any IO requests to them.
- Any data registered to the failed Datanode is not available to the HDFS.
- Also the death of a Datanode may cause replication factor of some of the blocks to fall below their specified value.

Re-replication

- The necessity for re-replication may arise due to:
  - A Datanode may become unavailable,
  - A replica may become corrupted,
  - A hard disk on a Datanode may fail, or
  - The replication factor on the block may be increased.

Cluster Rebalancing

- HDFS architecture is compatible with data rebalancing schemes.
- A scheme might move data from one Datanode to another if the free space on a Datanode falls below a certain threshold.
- In the event of a sudden high demand for a particular file, a scheme might dynamically create additional replicas and rebalance other data in the cluster.
- These types of data rebalancing are not yet implemented.

Data Integrity

- Consider a situation: a block of data fetched from Datanode arrives corrupted.
- This corruption may occur because of faults in a storage device, network faults, or buggy software.
- A HDFS client creates the checksum of every block of its file and stores it in hidden files in the HDFS namespace.
- When a clients retrieves the contents of file, it verifies that the corresponding checksums match.
- If does not match, the client can retrieve the block from a replica.

Metadata Disk Failure

- FsImage and EditLog are central data structures of HDFS.
- A corruption of these files can cause a HDFS instance to be non-functional.
- For this reason, a Namenode can be configured to maintain multiple copies of the FsImage and EditLog.
- Multiple copies of the FsImage and EditLog files are updated synchronously.
- Meta-data is not data-intensive.
- The Namenode could be single point failure: automatic failover is NOT supported! Another research topic.

DATA ORGANIZATION

Data Blocks

- HDFS support write-once-read-many with reads at streaming speeds.
- A typical block size is 64MB (or even 128 MB).
- A file is chopped into 64MB chunks and stored.

Staging

- A client request to create a file does not reach Namenode immediately.
- HDFS client caches the data into a temporary file. When the data reached a HDFS block size the client contacts the Namenode.
- Namenode inserts the filename into its hierarchy and allocates a data block for it.
- The Namenode responds to the client with the identity of the Datanode and the destination of the replicas (Datanodes) for the block.
- Then the client flushes it from its local memory.

Staging (contd.)

- The client sends a message that the file is closed.
- Namenode proceeds to commit the file for creation operation into the persistent store.
- If the Namenode dies before file is closed, the file is lost.
- This client side caching is required to avoid network congestion; also it has precedence is AFS (Andrew file system).

Replication Pipelining

- When the client receives response from Namenode, it flushes its block in small pieces (4K) to the first replica, that in turn copies it to the next replica and so on.
- Thus data is pipelined from Datanode to the next.

API (ACCESSIBILITY)

Application Programming Interface

- HDFS provides Java API for application to use.
- Python access is also used in many applications.
- A C language wrapper for Java API is also available.
- A HTTP browser can be used to browse the files of a HDFS instance.

FS Shell, Admin and Browser Interface

- HDFS organizes its data in files and directories.
- It provides a command line interface called the FS shell that lets the user interact with data in the HDFS.
- The syntax of the commands is similar to bash and csh.
- Example: to create a directory /foodir

/bin/hadoop dfs –mkdir /foodir

- There is also DFSAdmin interface available
- Browser interface is also available to view the namespace.

Space Reclamation

- When a file is deleted by a client, HDFS renames file to a file in be the /trash directory for a configurable amount of time.
- A client can request for an undelete in this allowed time.
- After the specified time the file is deleted and the space is reclaimed.
- When the replication factor is reduced, the Namenode selects excess replicas that can be deleted.
- Next heartbeat(?) transfers this information to the Datanode that clears the blocks for use.

Cloud Computing

- Complex computational models
- Large scale data
- Extraordinary resource needs
- Software as a service – SaaS (Web services, Turbi Tax online tax filer)
- Platform as a service – PaaS (Storage, bandwidth, compute cycles, machine as service: amazon EC2)
- Infrastructure as a service –IaaS (Storage, CPUs, etc.: MS Azure)
- Enabling technology: Virtualization

Summary (1)

- We discussed the features of the Hadoop File System, a peta-scale file system to handle big-data sets.
- What discussed: Architecture, Protocol, API, etc.
- Missing element: Implementation
  - The Hadoop file system (internals)
  - An implementation of an instance of the HDFS (for use by applications such as web crawlers).
- KOSMIX file system (KFS) called cloudstore is a C++ implementation of GFS-workalike.

Summary (2)

- 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
- Relationship to Cloud Computing

Demo

- VMware simulated Hadoop and MapReduce demo
- Remote access to NEXOS system at my Buffalo office (broke?)
  - 5-node HDFS running HDFS on Ubuntu 8.04
  - 1 –name node and 5 data-nodes
  - Each is an old commodity PC with 512 MB RAM, 120GB – 160GB external memory
  - Zeus (namenode), datanodes: hermes, dionysus, aphrodite, athena, theos
- CSE department's infrastructure (Dr. Chaudhary's)
- Amazon Elastic Cloud Computing (EC2)

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