EE359 Data Mining

MapReduce

Jiaxin Ding
John Hopcroft Center

上海交通大学
约翰·霍普克罗夫特
计算机科学中心
John Hopcroft Center for Computer Science

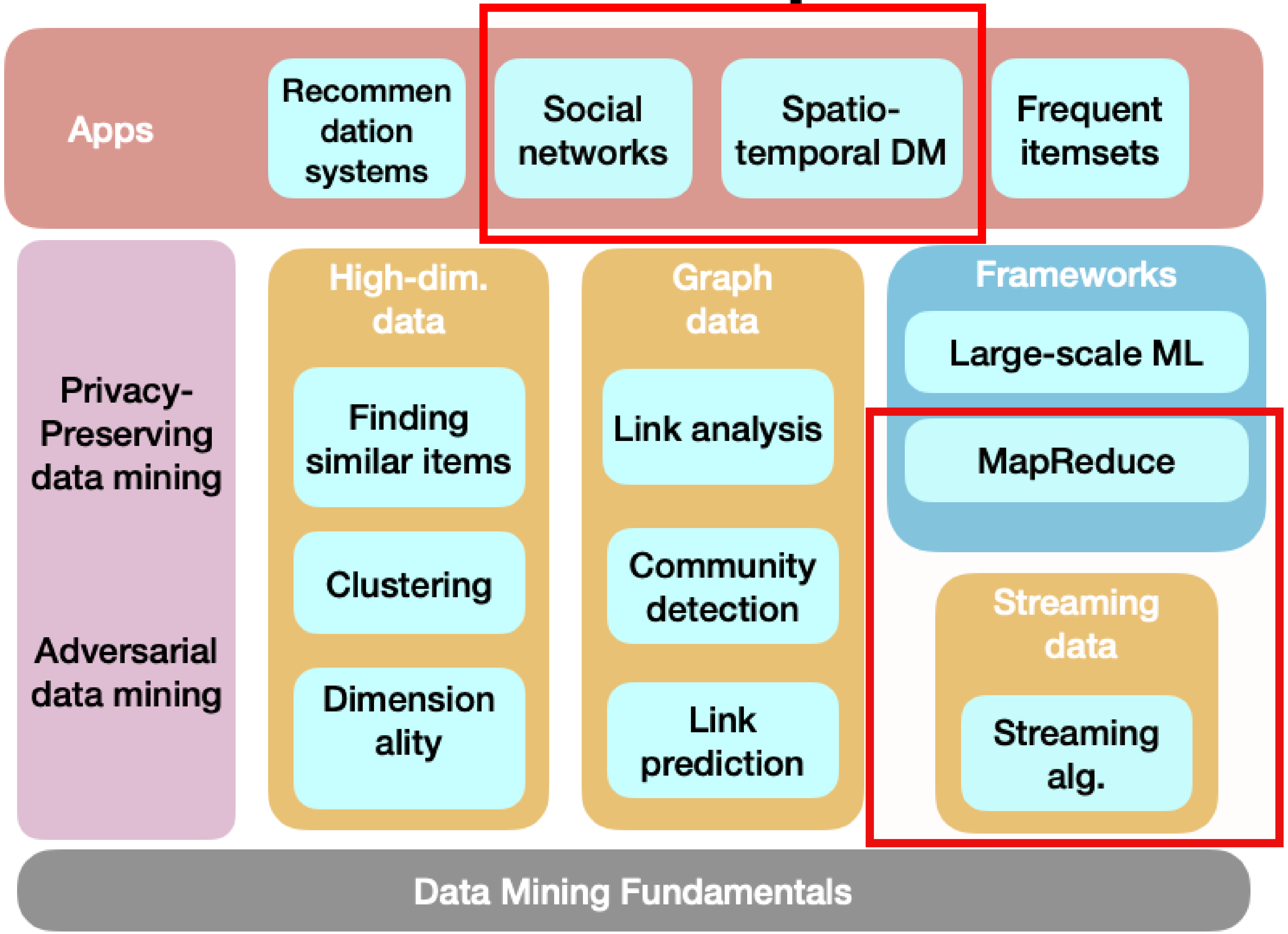
SHANGHAI JIAO TONG UNIVERSITY
1896

简介——丁家昕

- 上海交通大学John Hopcroft Center，助理教授，IIOT智能物联网中心成员
- 研究方向：时空数据挖掘，表征学习，强化学习, 物联网
- 主页：http://jhc.sjtu.edu.cn/~jiaxinding

- 教育背景
  - 2019年University of California, Davis博士后
  - 2018年State University of New York at Stony Brook计算机博士学位
  - 2012年北京大学信息科学技术学院学士学位

Course Landscape



Schedule

Week	Tues.	Thur.
13	MapReduce	Streaming 1
14	Streaming 2	Streaming Experiment
15	Social Networks	Streaming Experiment
16	Spatio-Temporal Data	Poster

A DAY IN DATA

The exponential growth of data is undisputed, but the numbers behind this explosion – fuelled by internet of things and the use of connected devcies – are hard to comprehend, particularly when looked at in the context of one day

DEMYSTIFIYING DATA UNITS

From the more familiar 'bit' or 'megabyte', larger units of measurement are more frequently being used to explain the masses of data

Unit		Value	Size
b	bit	0 or 1	1/8 of a byte
B	byte	8 bits	1 byte
KB	kilobyte	1,000 bytes	1,000 bytes
MB	megabyte	$1{,}000^2$ bytes	1,000,000 bytes
GB	gigabyte	$1{,}000^3$ bytes	1,000,000,000 bytes
TB	terabyte	$1{,}000^4$ bytes	1,000,000,000,000 bytes
PB	petabyte	$1{,}000^5$ bytes	1,000,000,000,000,000 bytes
EB	exabyte	$1{,}000^6$ bytes	1,000,000,000,000,000,000 bytes
ZB	zettabyte	$1{,}000^7$ bytes	1,000,000,000,000,000,000,000 bytes
YB	yottabyte	$1{,}000^8$ bytes	1,000,000,000,000,000,000,000,000 bytes

*A lowercase "b" is used as an abbreviation for bits, while an uppercase "B" represents bytes.

463EB

of data will be created every day by 2025

IDC

500m

tweets are sent every day

Twitter

4PB

of data created by Facebook, including

350m photos

100m hours of video watch time

Facebook Research

95m

photos and videos are shared on Instagram

Instagram Business

65bn

messages sent over WhatsApp and two billion minutes of voice and video calls made

Facebook

320bn

emails to be sent each day by 2021

294bn

billion emails are sent

Radicati Group

306bn

emails to be sent each day by 2020

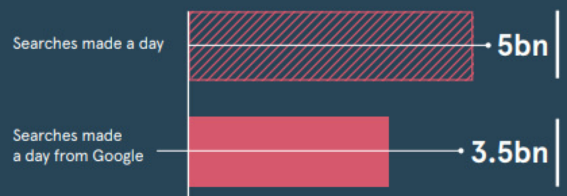
3.9bn

people use emails

4TB

of data produced by a connected car

Intel



Smart Insights

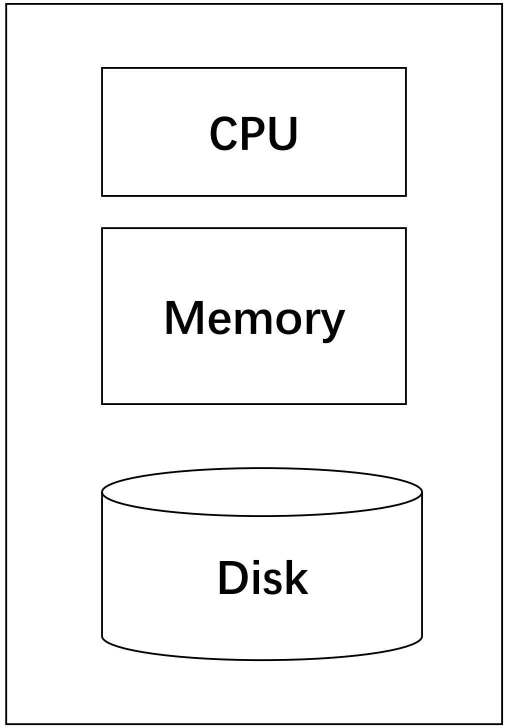
28PB

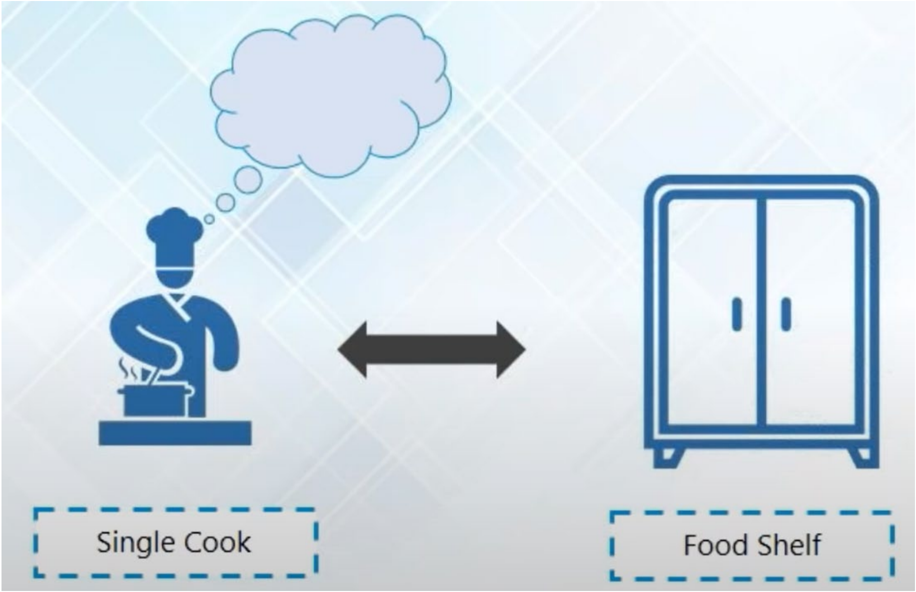
to be generated from wearable devices by 2020

Statista

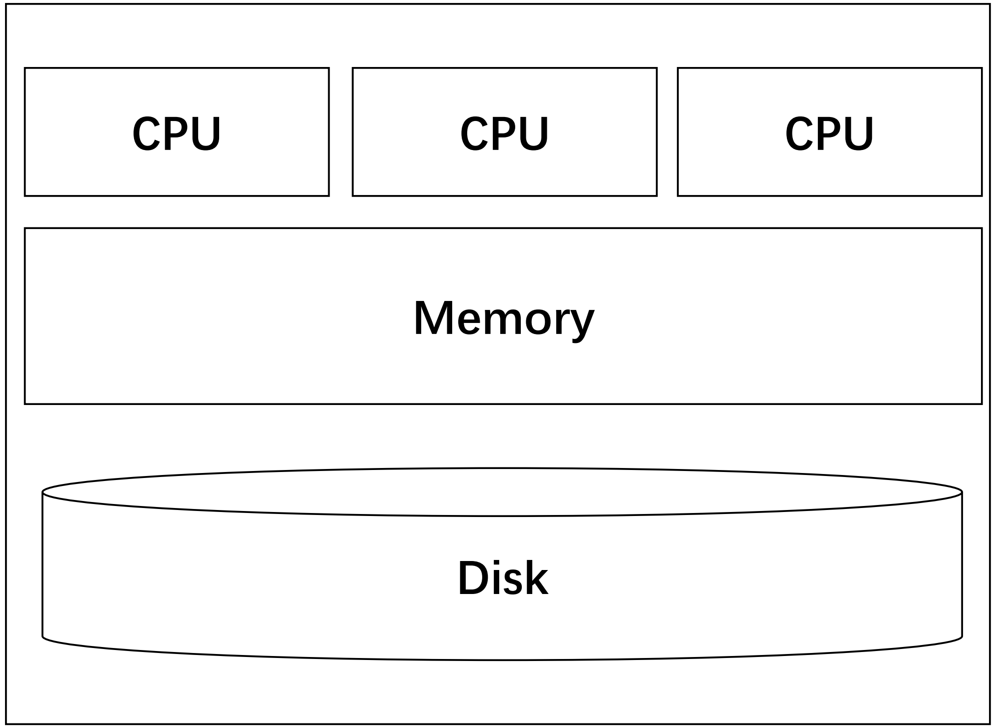
ACCUMULATED DIGITAL UNIVERSE OF DATA

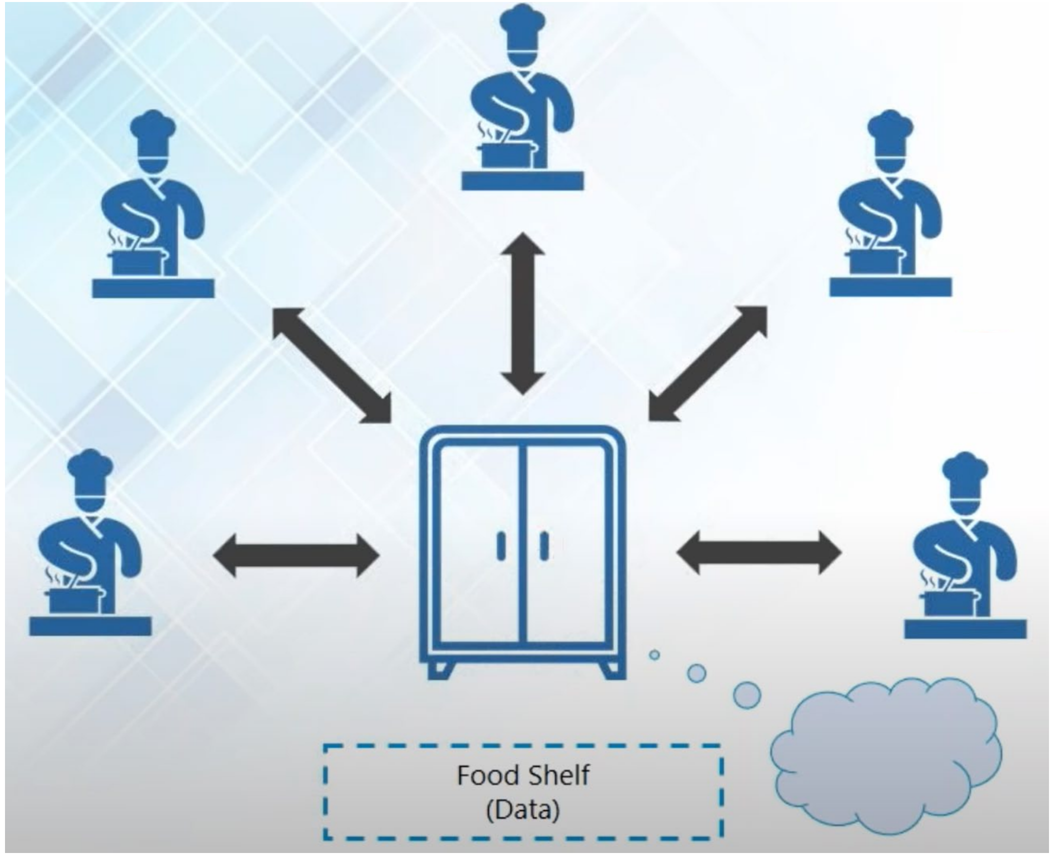
Single Node Architecture



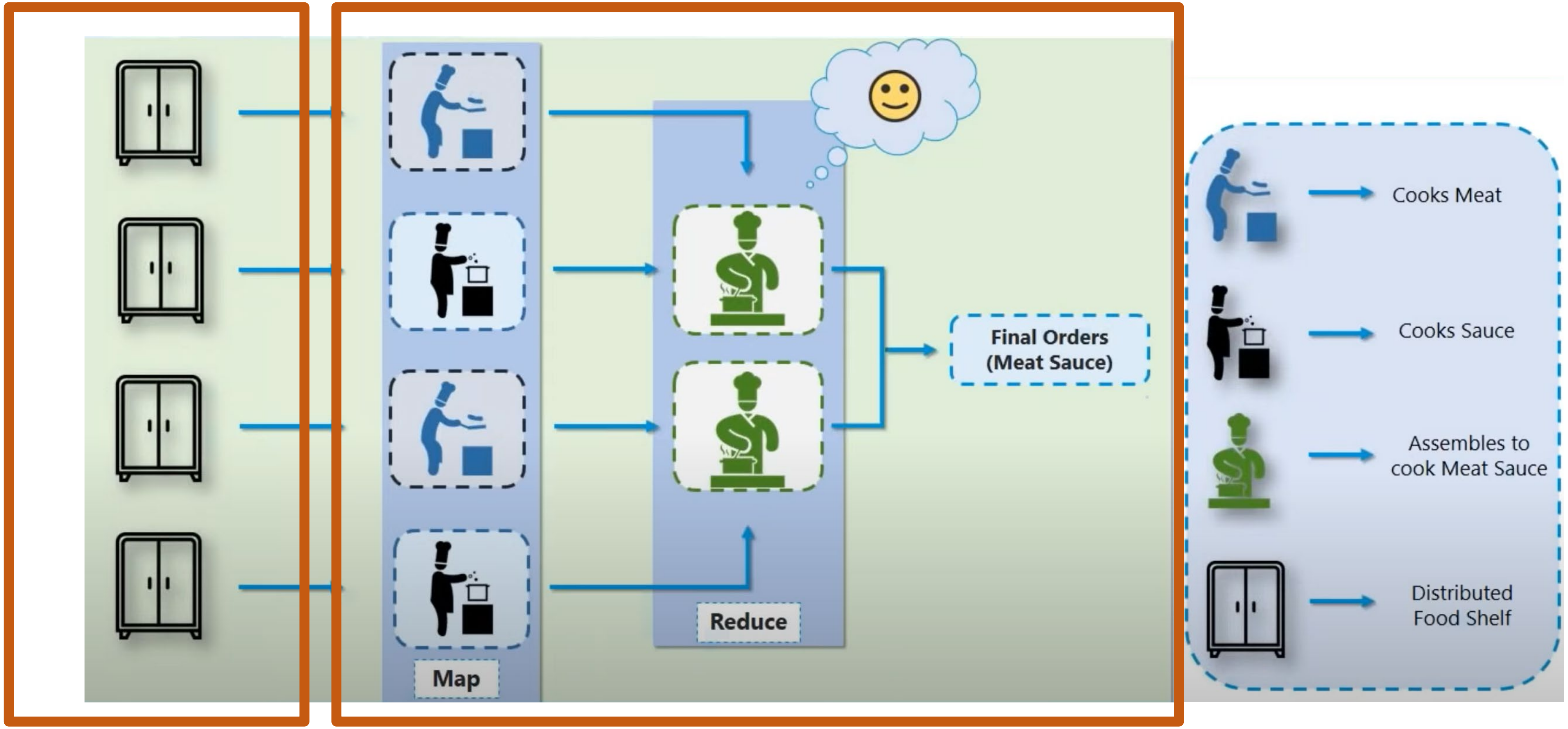


Distributed Computing





MapReduce



Distributed File System **MapReduce**

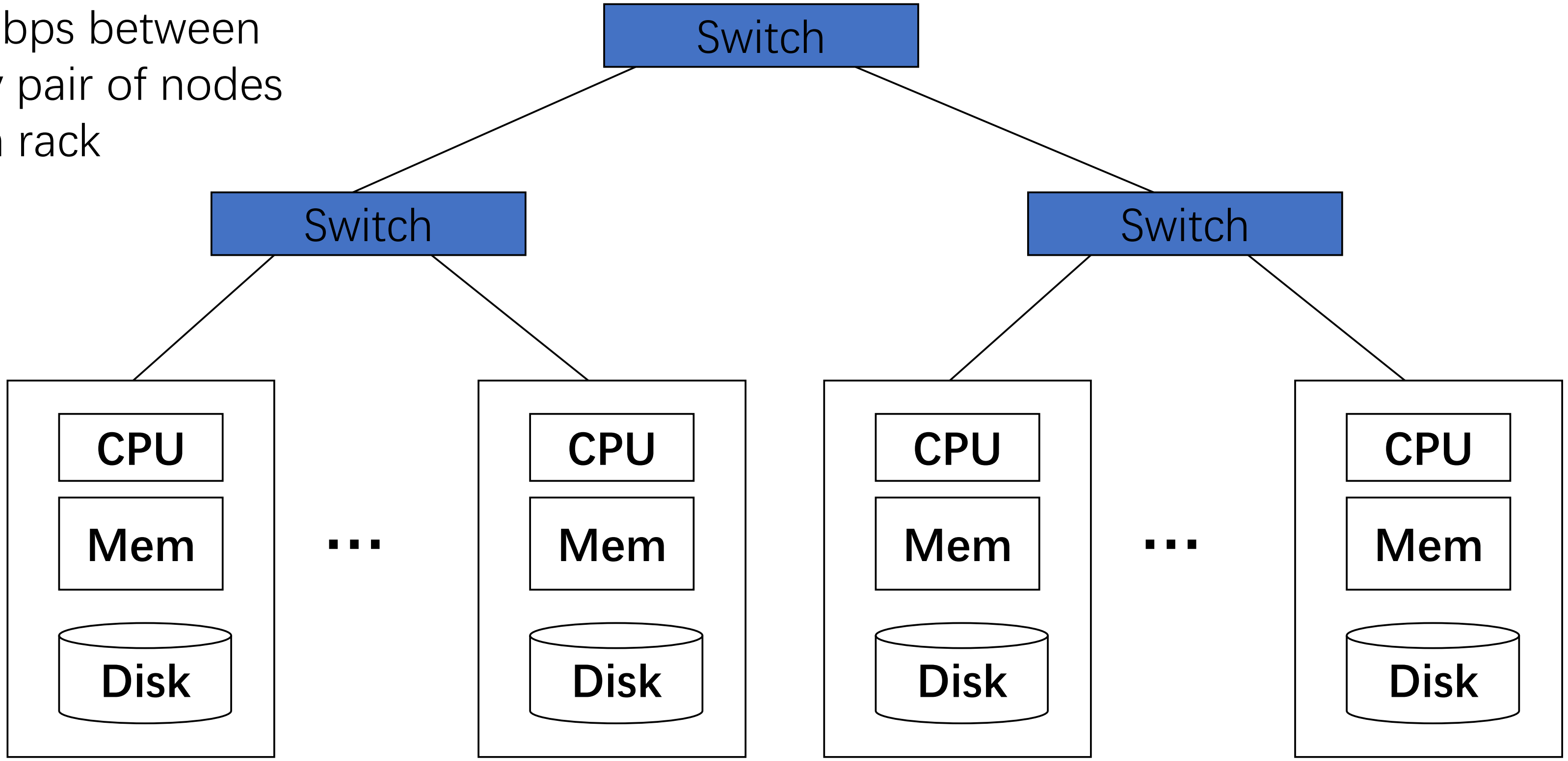
Google Example

- 50+ billion web pages x 20KB = 1000+ TB
- 1 computer reads 300 MB/sec from disk
  - ~1 months to read the web
- ~1,000 hard drives to store the web
- **Today, a standard architecture for such problems is emerging:**
  - **Cluster** of commodity Linux nodes
  - **Commodity network** (ethernet) to connect them

Cluster Architecture

2-10 Gbps backbone between racks

1 Gbps between
any pair of nodes
in a rack



Each rack contains 16-64 nodes

In 2019 it was guestimated that Google had 2.5M machines

Large-scale Computing Challenges

- **Large-scale computing** on **commodity hardware**
- **Challenges:**
  - **Latency issues:**
    - Copying data over a network takes time
  - **How do you distribute computation?**
  - **How can we make it easy to write distributed programs?**
  - **Machines fail:**
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to loose 1/day
    - People estimated Google had ~2.5 M machines in 2019
      - 2,500 machines fail every day!

Adapted from J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org
Part of pictures are from Edureka.co

Solutions

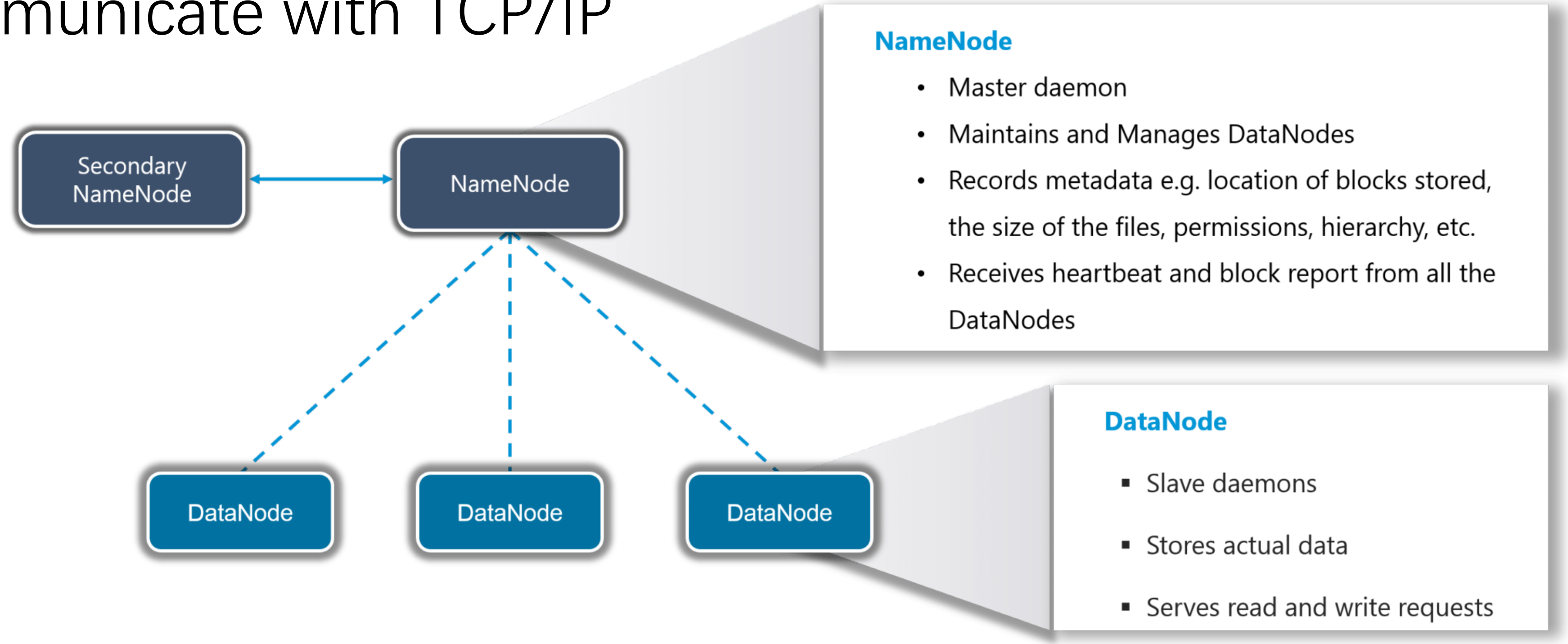
- **Idea:**
  - Bring computation close to the data
  - Store files multiple times for reliability
- **Solutions**
  - **Storage: File system**
    - Google: GFS. Hadoop: HDFS
  - **Computing: Programming model**
    - MapReduce: Google and Hadoop

Storage

- **Distributed File System:**
  - Provides global file namespace
  - Google GFS; Hadoop Distributed File System (HDFS);
- **Typical usage pattern**
  - Huge files (100s of GB to TB)
  - **Write Once – Read Many Philosophy**
    - Data is rarely updated in place
    - Reads and appends are common

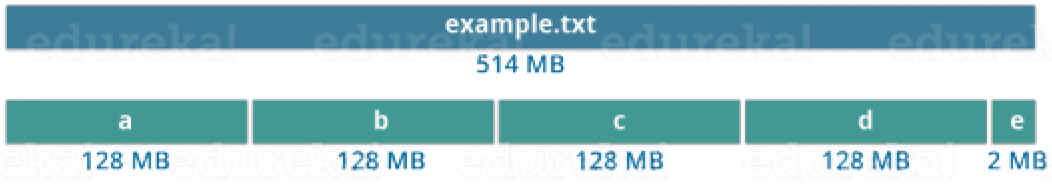
Hadoop Distributed File System

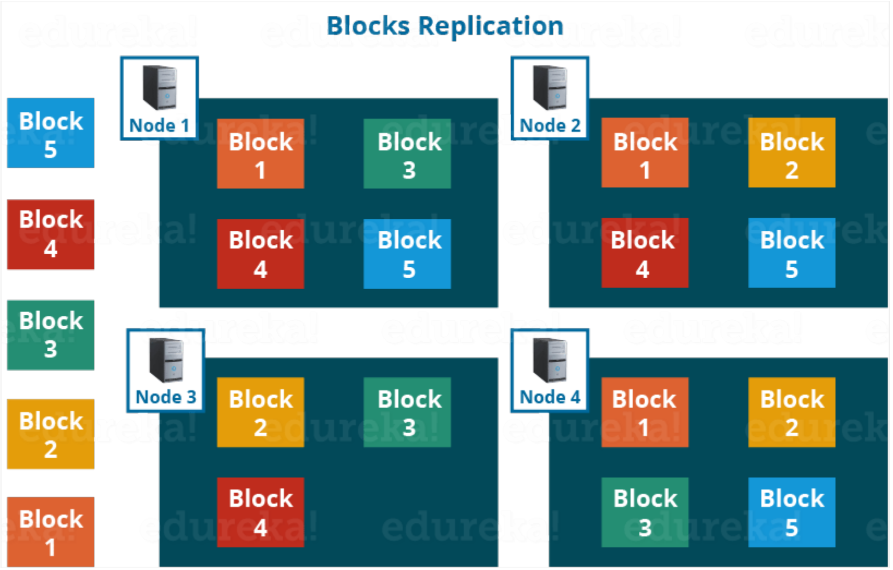
- Name Node
- Data Node
- Communicate with TCP/IP



Hadoop Distributed File System

- **Blocks**
  - HDFS stores each file as blocks which are scattered throughout the Apache Hadoop cluster. The default size of each block is 128 MB (Compared to Linux 4KB).
- **Replication** management to recovery failures.
  - How many replicas are needed?
  - How to store replicas?
- **Bring computation to data.**





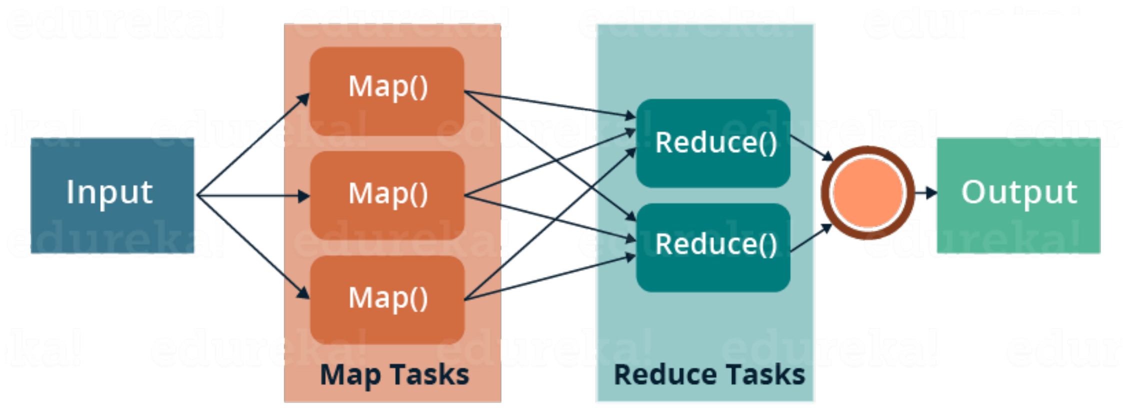
Programming Model: MapReduce

- **MapReduce** is a style of programming design for
  - **Easy** parallel **programming**
  - **Invisible** management of **hardware and software failures**
  - **Easy** management of **large-scale data**

- Implementations
  - Google MapReduce
  - Hadoop
  - Spark (improved)

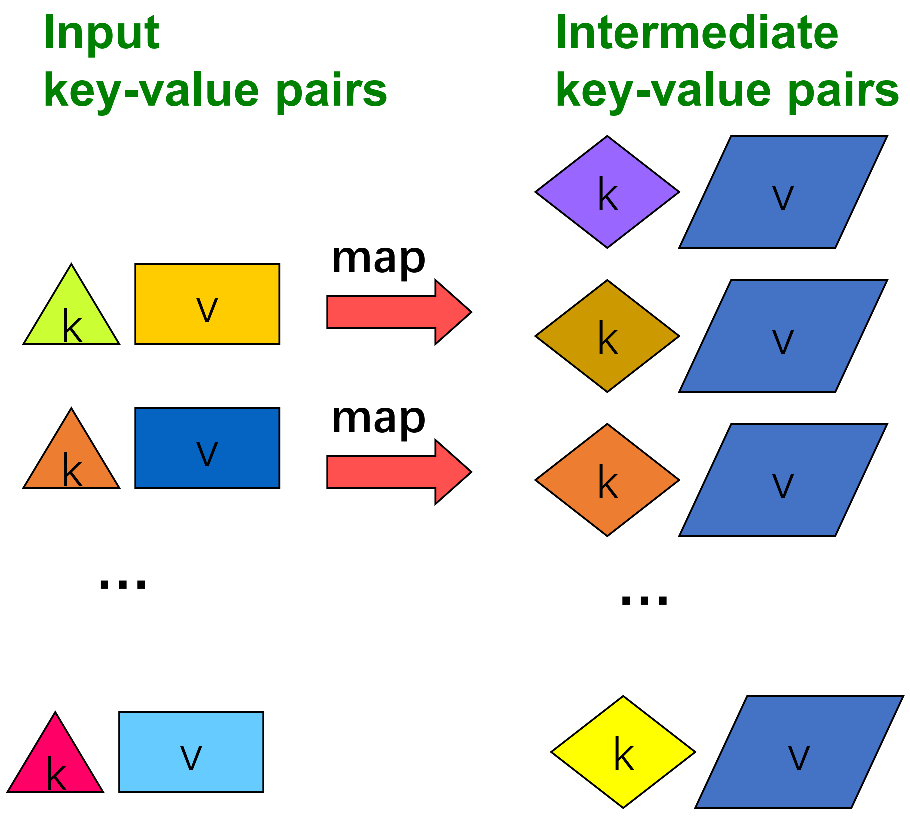
MapReduce: Overview

- Sequentially read a lot of data
- **Map:**
  - Extract something you care about
- Group by key:
  - Sort and shuffle
- **Reduce:**
  - Aggregate, summarize, filter or transform
- Write the result to disks

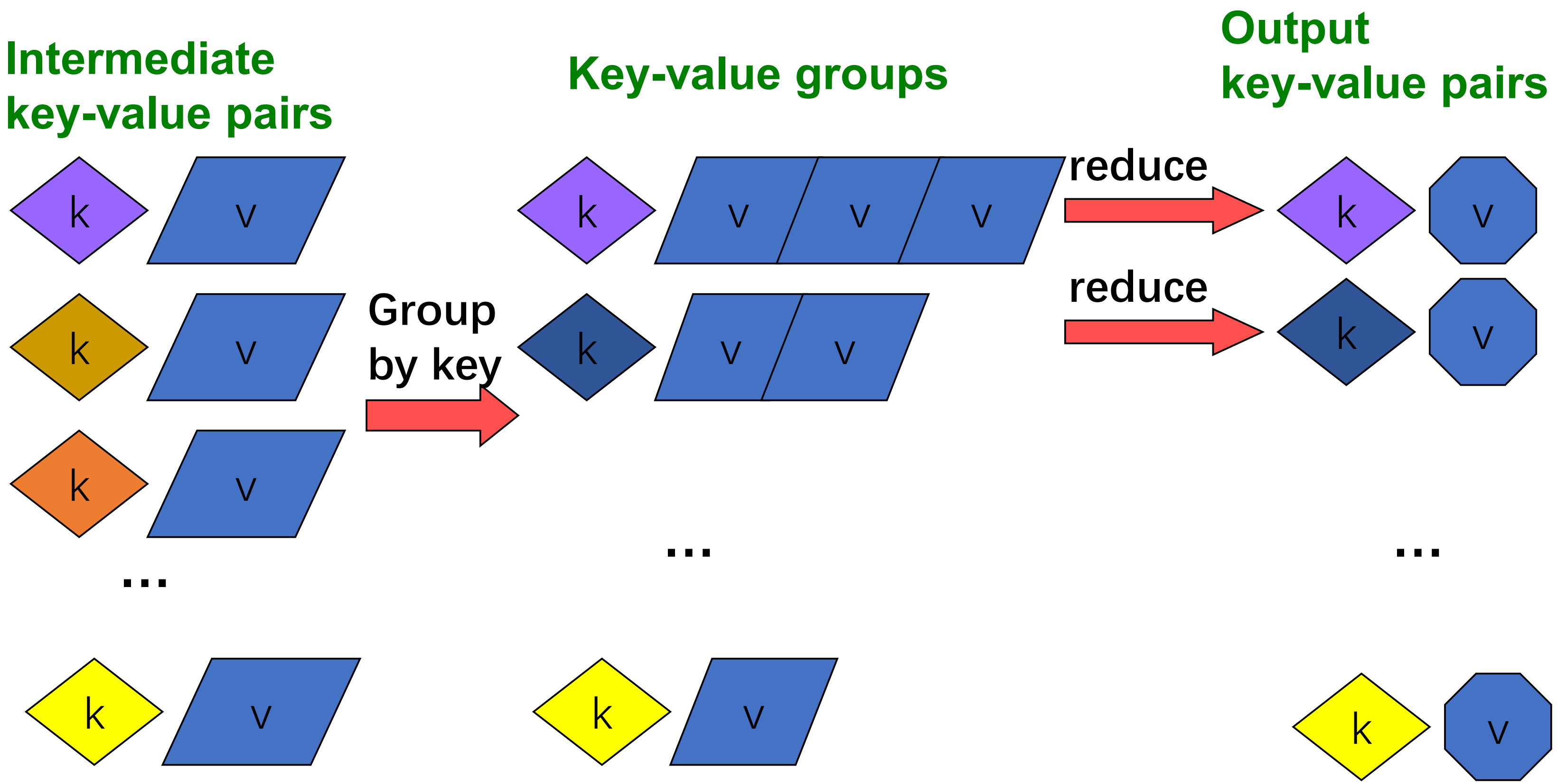


Outline stays the same, **Map** and **Reduce** change to fit the problem

MapReduce: The Map Step



MapReduce: The Reduce Step



More Specifically

- **Input:** a set of key-value pairs
- Programmer specifies two methods:
  - **Map(k, v)** $\rightarrow$ <k’, v’>*
    - **Takes a key-value pair and outputs a set of key-value pairs**
      - E.g., key is the filename, value is a single line in the file
    - There is one Map call for every (k,v) pair
  - **Reduce(k’, <v’>*)** $\rightarrow$ <k’, v’’>*
    - **All values v’ with same key k’ are reduced together and processed in v’ order**
    - There is one Reduce function call per unique key k’

Example: Word Counting

- **Word counting** task:
  - We have huge text document
  - Count the number of times each distinct word appears in a file

- Motivations:
  - Analyze web server logs to find popular websites
  - Find the most popular key words

MapReduce: Word Counting

Big document	Provided by the programmer		Provided by the programmer
	MAP: Read input and produces a set of key-value pairs	**Group by key:** Collect all pairs with same key	**Reduce:** Collect all values belonging to the key and output
The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need …………………….	(The, 1) (crew, 1) (of, 1) (the, 1) (space, 1) (shuttle, 1) (Endeavor, 1) (recently, 1) ….	(crew, 1) (crew, 1) (space, 1) (the, 1) (the, 1) (the, 1) (shuttle, 1) (recently, 1) …	(crew, 2) (space, 1) (the, 3) (shuttle, 1) (recently, 1) …
Big document	**(key, value)**	**(key, value)**	**(key, value)**

Sequentially collect to get the result

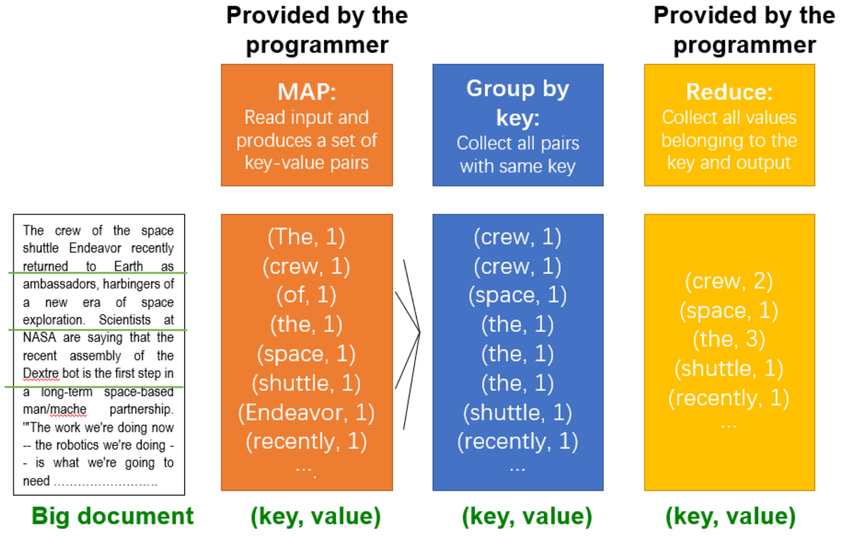
Word Count Using MapReduce

Your programs:

```
map(key, value):
// key: document name; value: text of the document
  for each word w in value:
      emit(w, 1)

reduce(key, values):
// key: a word; value: an iterator over counts
      result = 0
      for each count v in values:
            result += v
      emit(key, result)
```

Map-Reduce: Environment



Map-Reduce environment takes care of:

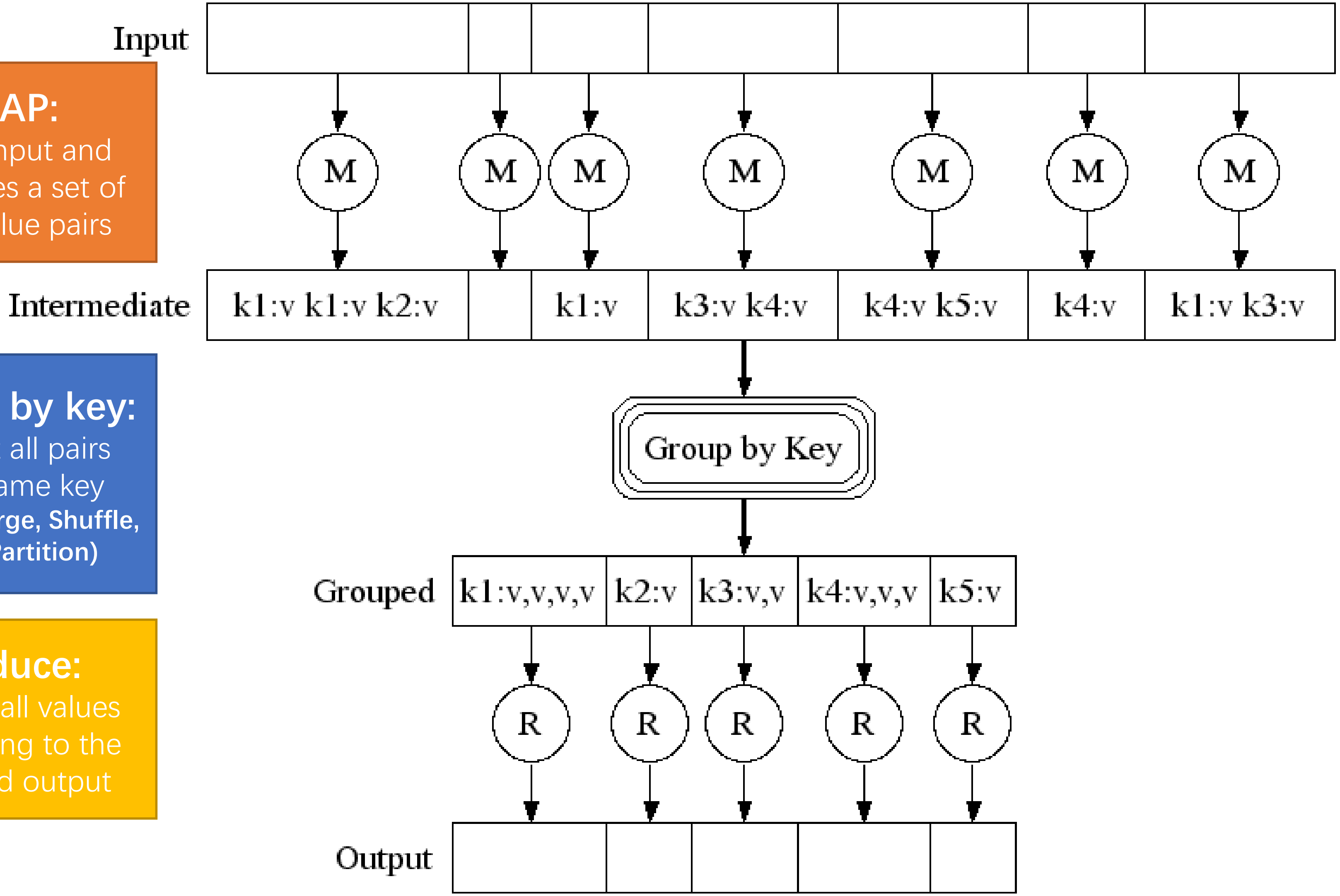
- **Partitioning** the input data
- **Scheduling** the program's execution across a set of machines
- Performing the **group by key** step
- **Refine** tasks by intermediate combiners
- Handling machine **failures**
- Managing required inter-machine **communication**

Map-Reduce: A diagram

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



Example: Join By Map-Reduce

- **Compute the natural join** $R(A,B) \bowtie S(B,C)$
- R and S are each stored in files
- Tuples are pairs (a,b) or (b,c)

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

$\bowtie$

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

$=$

A	C
a_3	c_1
a_3	c_2
a_4	c_3

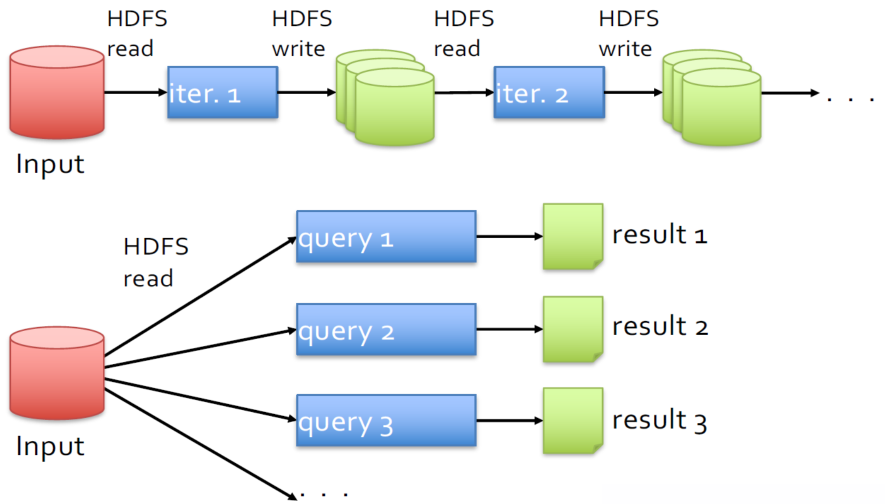
Join by MapReduce

- **A Map process turns:**
  - Each input tuple *R(a,b)* into key-value pair *(b,(a,R))*
  - Each input tuple *S(b,c)* into *(b,(c,S))*

- Group by keys:
  - Use a hash function *h* from B-values to *1...k*, Map processes send each key-value pair with key *b* to Reduce process *h(b)*
- Each **Reduce process** matches all the pairs *(b,(a,R))* with all *(b,(c,S))* and outputs *(a,c)*.

Problems with MapReduce

- Hadoop MapReduce is **inefficient** for applications that repeatedly reuse a working set of data:
  - **Iterative** algorithms (machine learning, graphs): incurs substantial **overheads** due to data replication, disk I/O
  - **Interactive** data mining tools： all Java codes; R, Python not supported

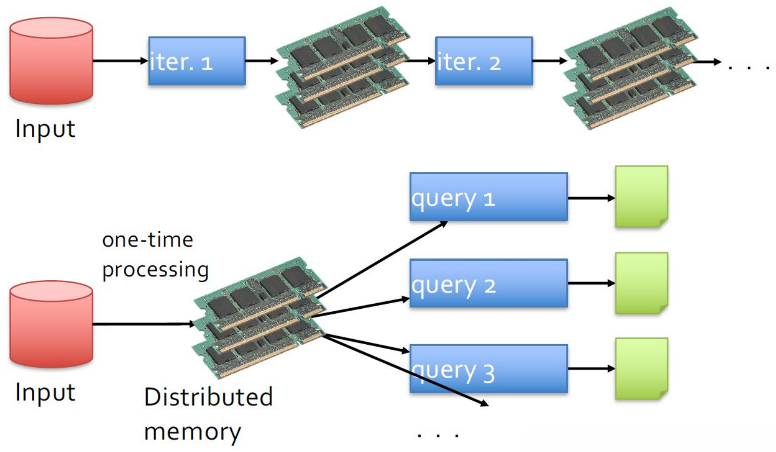


Problems with MapReduce

- **Data flow** is not flexible enough
  - MapReduce uses only two types of tasks: Map and Reduce;
  data flows are always from Map to Reduce.

Solution: Spark

- Allow apps to keep working sets in **memory** for efficient reuse
- Retain the attractive properties of MapReduce
  - Fault tolerance, data locality, scalability
- Additions to MapReduce model:
  - Richer functions than just map and reduce
  - Better data flow scheduler



Spark Overview

- Spark is a **unified analytics** engine for large-scale **data processing**.
- **100x Faster**
  - **RDD**: resilient distributed datasets(弹性分布式数据集), core building block.
  - **DAG**: directed acyclic graph(有向无环图), general execution graph scheduler.
- **Ease of use**
  - Spark provides **data focused API** which makes writing large-scale programs easy, such as DataFrames & DataSets
  - Compatible with Scala, Java, R, Python

Core Concept: RDD

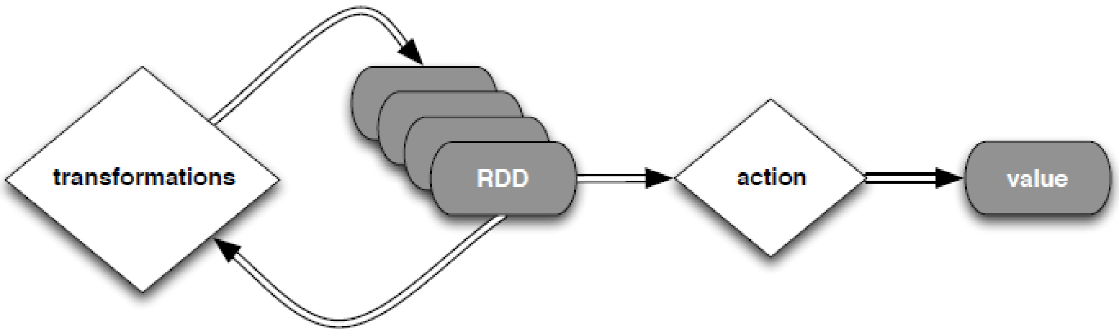
Resilient distributed datasets (RDDs): Primary abstraction

- **Immutable**, **partitioned** collections of objects
  - Generalized key-value pairs
- Caching in **memory**

- There are currently two types:
  - *parallelized collections* – take an existing collection and run functions on it in parallel
  - *Hadoop datasets* – run functions on each record of a file in Hadoop distributed file system or any other storage system supported by Hadoop

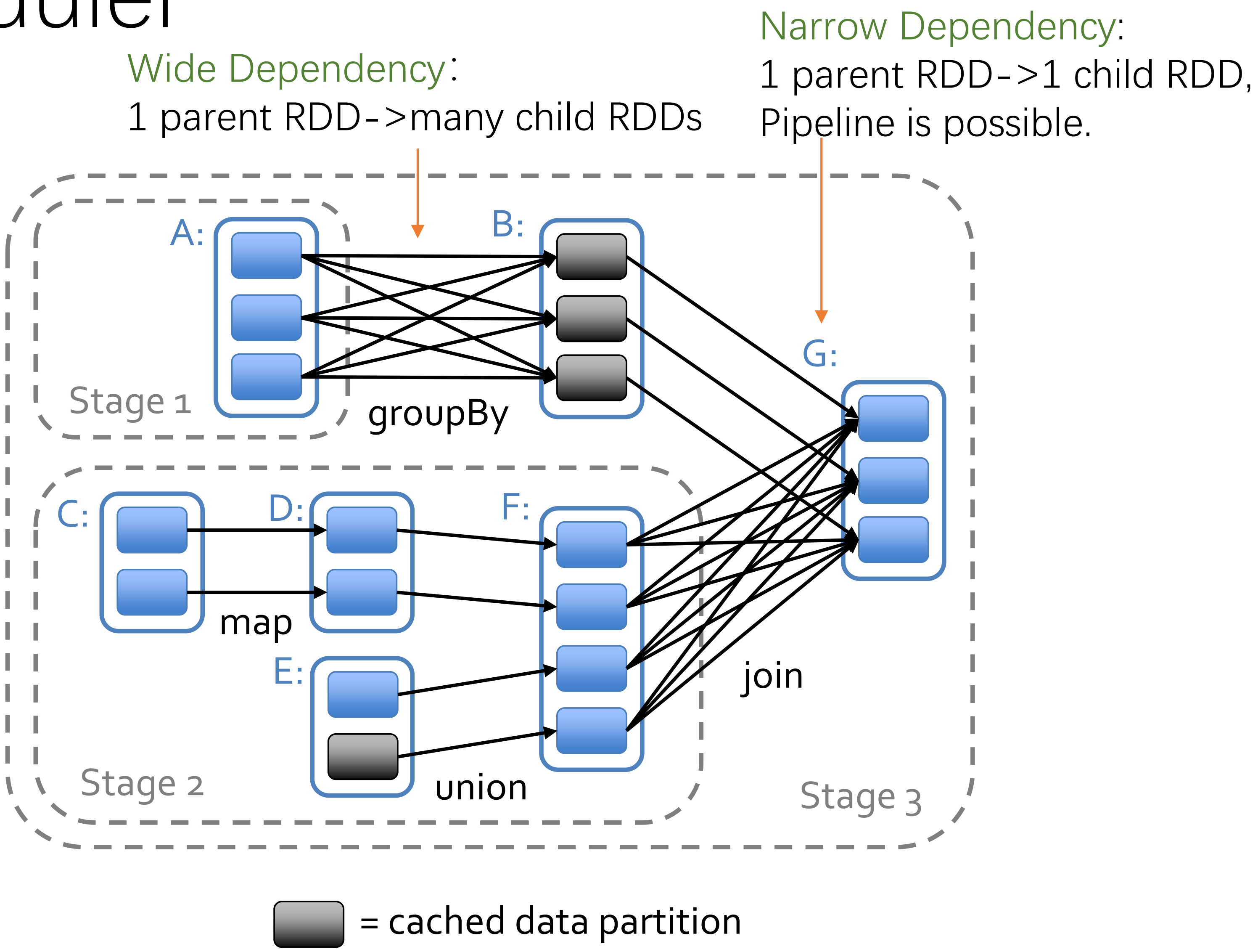
Spark RDD Operations

- **Operations** on RDDs:
  - **Transformations:** build RDDs from other RDDs
    - Transformations create a new RDD from an existing one
    - Transformations are **lazy**: nothing computed until an action requires it.
    - map, filter, groupBy, join, union, intersection, …
  - **Actions:** get results
    - A transformed RDD gets recomputed when an action is run on it
    - reduce, count, collect, save, …



Spark DAG Scheduler

- Supports general task **graph** scheduling
- Pipelines functions within a **stage**
  - Narrow vs Wide dependency
  - Divide into stages where there is a wide dependency (can not use pipeline)
- **Cache-aware** work reuse & locality

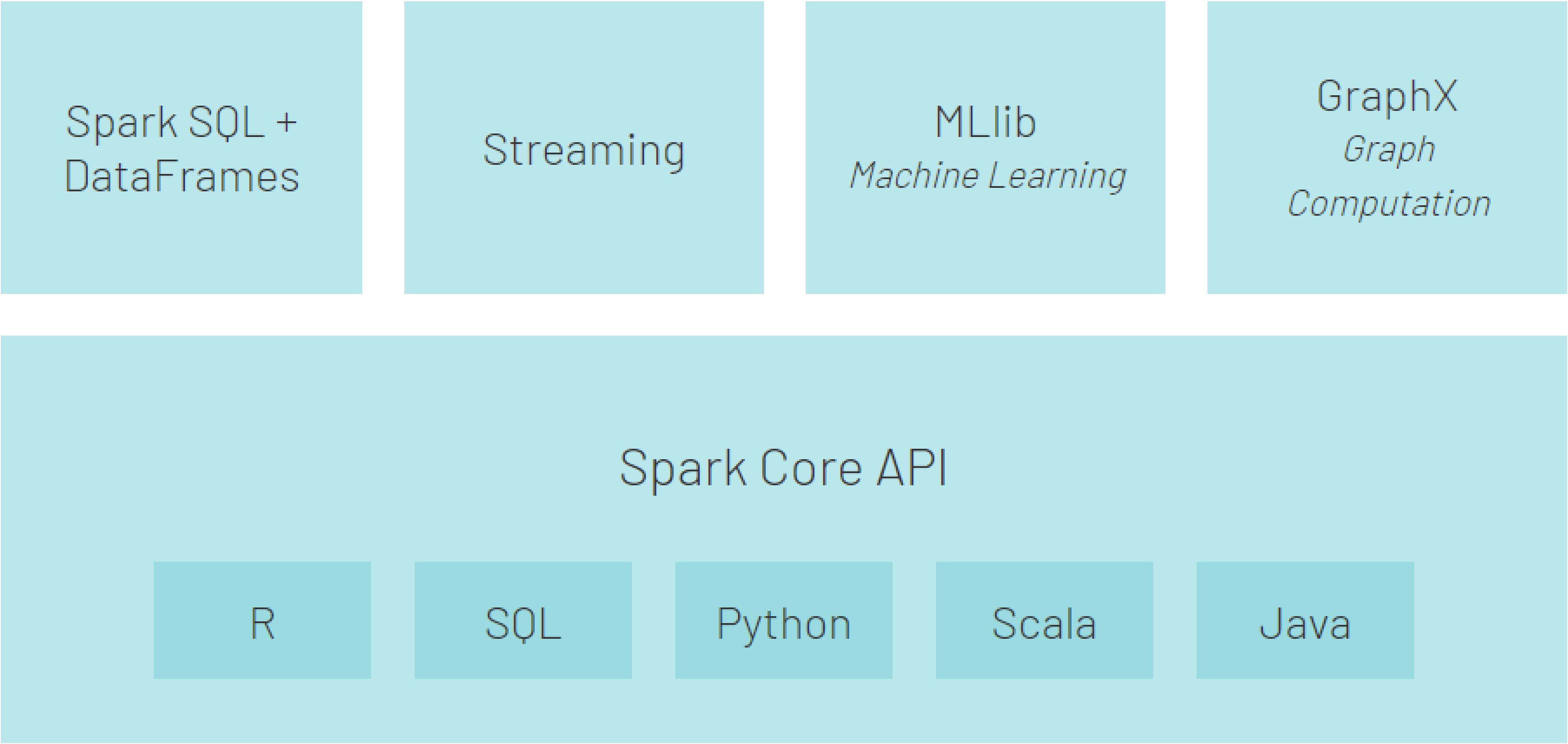


Spark DataFrame: High Level Abstraction

- A **DataFrame** is a dataset organized into **named columns**.
  - For structured and semi-structured data.
  - Conceptually equivalent to a table in a relational database or a data frame in Python.
- Common characteristics with RDD:
  - Immutable in nature: You will be able to create a DataFrame but you will not be able to change it.
  - Lazy Evaluations: a task is not executed until an action is performed.
  - Distributed: DataFrames just like RDDs are both distributed in nature.
- DataFrame allows higher-level abstraction and optimization
  - Support SQL queries
  - Better optimization engines

Spark ecosystem

- Useful libraries



Summary

Big data processing:

- MapReduce: distributed programming/computing framework
  - HDFS
  - Map and Reduce
  - System handles all other processes
  - Save results to file systems
- Spark: improved over MapReduce
  - RDD: distributed in memory
  - DAG scheduling
  - Programming friendly