Lecture on 20 Jul

0900-1030	lecture
1030-1045	break
1045-1200	lecture
1200-1400	lunch
1400-1445	lecture
1445-1500	break
1500-1545	lecture
1545-1600	break
1600-1700	assessment 2

L04: Applications & Paradigm

L05: IBM Bluemix
Cloud Services

L06: Projects

L04: Applications and Paradigms

Outline

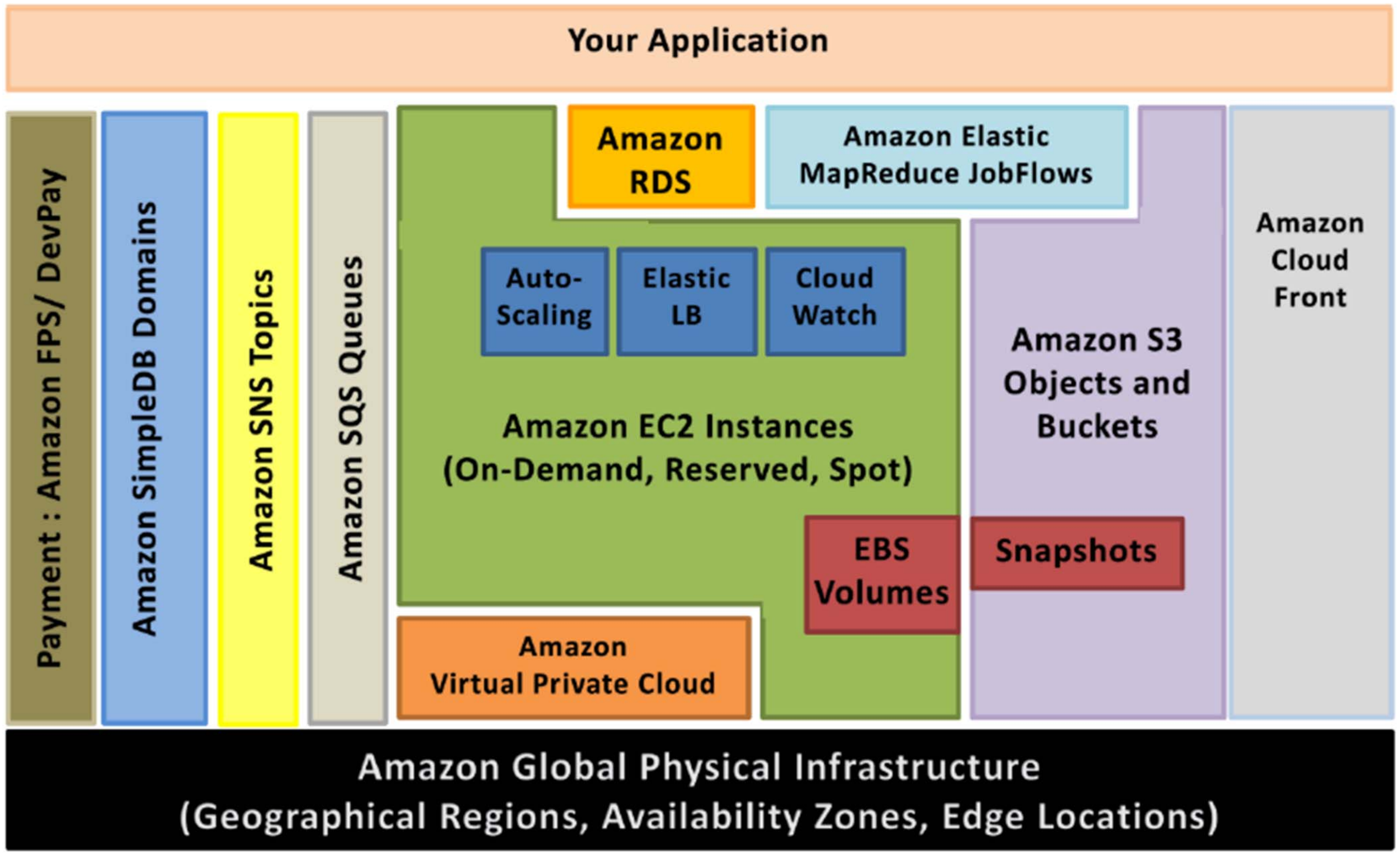
Key Terms

1. Divisible workload
2. Performance isolation
3. Web application architecture layers
4. Application development models (IaaS, PaaS, SaaS)
5. MapReduce & Hadoop

Common Features of Cloud Providers

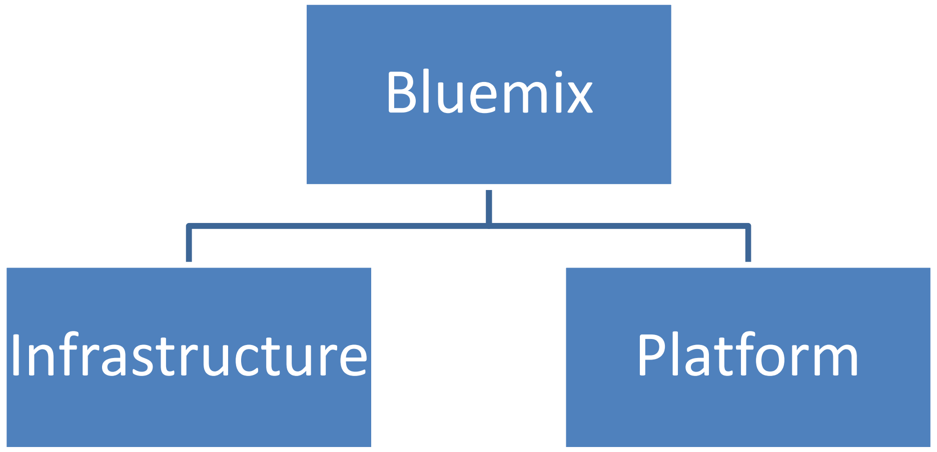
- Basic and higher-level services (IaaS -> SaaS)
- Compute and storage resources - virtual servers (Linux and Windows) and object store
- Deployment management services - load balancer, auto-scaling, message queueing, monitoring, …
- User interface - graphical user interface, command-line interface
- …

Example: Amazon Web Services (AWS) Ecosystem


Your Application
Payment : Amazon FPS/ DevPay
Amazon SimpleDB Domains
Amazon SNS Topics
Amazon SQS Queues
Amazon RDS
Amazon Elastic MapReduce JobFlows
Auto-Scaling
Elastic LB
Cloud Watch
Amazon EC2 Instances (On-Demand, Reserved, Spot)
EBS Volumes
Snapshots
Amazon S3 Objects and Buckets
Amazon Cloud Front
Amazon Virtual Private Cloud
Amazon Global Physical Infrastructure (Geographical Regions, Availability Zones, Edge Locations)

J. Varia, Architecting for the Cloud: Best Practices, Amazon Web Services, May 2010.

Example: IBM Bluemix Services



Infrastructure

1. Compute
2. Storage
3. Network
4. Security
5. Containers
6. VMware

Platform

1. Boilerplates
2. APIs
3. Application Services
4. Blockchain
5. Cloud Foundry Apps
6. Data & Analytics
7. DevOps
8. Finance
9. Functions
10. Integrate
11. Internet of Things
12. Mobile
13. Security
14. Watson

CLOUD APPLICATIONS

- focus on enterprise computing

- ideally, an application can partition its workload into ***n*** segments and spawn ***n*** instances, and the execution time reduced by a factor close to ***n***

why?

- key challenges:
  - cloud consumer – scale application up and down to accommodate a dynamic load, recover after a system failure, efficiently support checkpoint/restart
  - cloud provider – managing a large *number of systems*, provides *quality of service guarantee*

Cloud Applications

- ideal applications:
  - web services
  - database services
  - transaction-based service

- unlikely to perform well:
  - applications with a complex workflow and multiple dependencies such as high-performance computing
  - applications with intensive communication among concurrent instances
  - when the workload cannot be arbitrarily partitioned **why?**

Challenges in Developing Cloud Applications

- Performance isolation – how to ensure that customer performance is not affected by other users? nearly impossible to reach in a real system, especially when the system is heavily loaded

- Reliability - major concern, server failures expected when a large number of servers cooperate for the computations

- Cloud infrastructure exhibits latency and bandwidth fluctuations which affect the application performance

- Performance considerations limit the amount of *data logging* (frequent logging to identify unexpected results and errors)

Architectural Styles for Cloud Applications

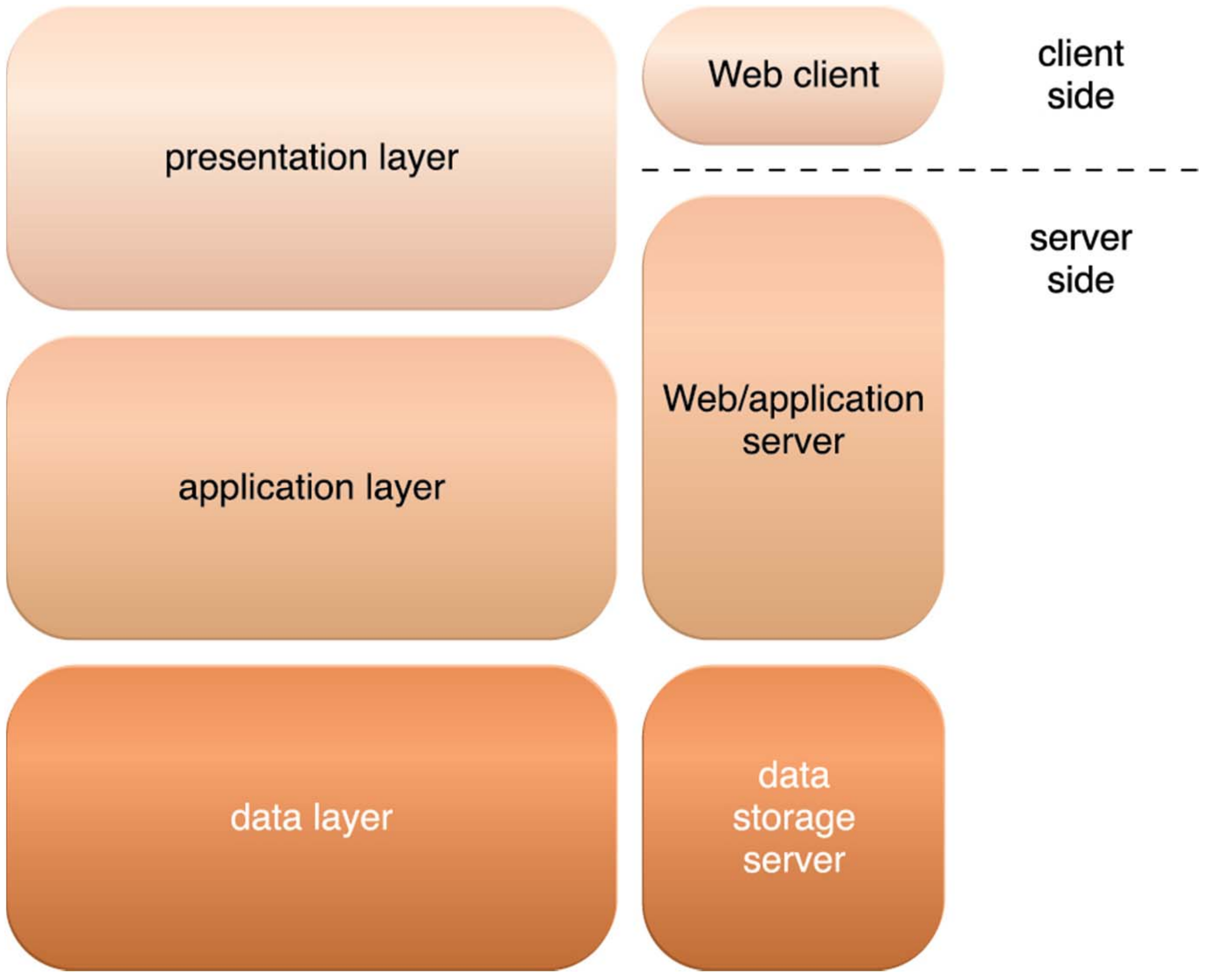
- Cloud computing is reliance on internetworking and web technology (high accessibility, web browser universality, ease of web-based service development)

- Cloud services use web technology as both the implementation medium and management interface

- 2 basic components of the web are web browser client and web server, i.e. based on client-server architecture

- Three-tier architecture abstraction for a web application:
  1. Presentation layer represents the user-interface
  2. Application layer represents the implementation logic
  3. Data layer comprises of persistent data stores

Architecture of Web Applications



Three architectural tiers of Web applications

1. Presentation layer has components on both the client and server side
2. Web server receives client requests and retrieves the requested resources or generates the web content according to the application logic
3. Web servers interact with application servers and the underlying databases

PaaS typically provides separate instances of the web server, application server and data storage server environments

Architectural Styles for Cloud Applications

- Web Services technology – provides “as-a-service” cloud delivery models:

  1. Simple Object Access Protocol (SOAP) - *application protocol for web applications*; defines a common web service messaging format for request and response message exchanges; based on the XML, uses TCP or UDP transport protocols

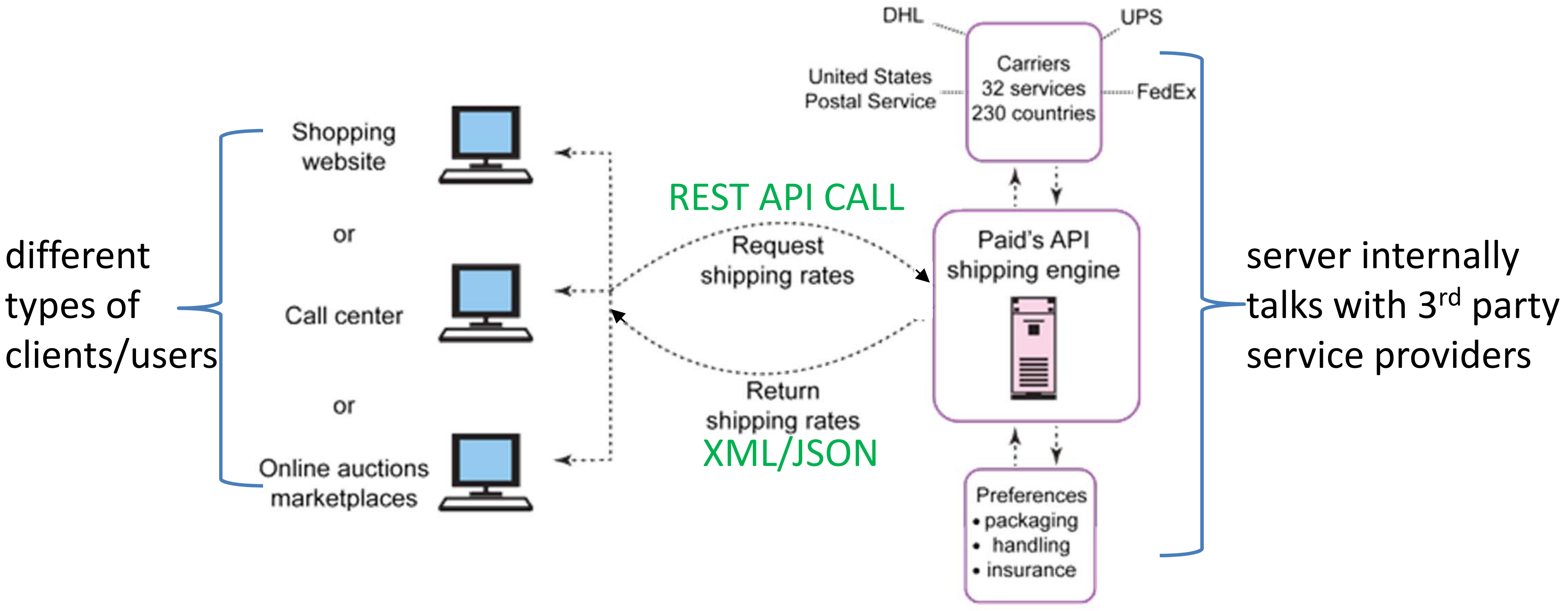
  2. Representational State Transfer (REST) - *software architecture for distributed hypermedia systems*. Supports client communication with stateless servers, platform and language independent, supports data caching, and can be used in the presence of firewalls

Web Services for Cloud Applications

- Cloud services are naturally distributed. A single service request might require message passing between multiple servers from different providers
- Cloud services are accessed by different clients using different programming languages and technologies. (eg. Java, Python, ..) Thus, requires a standard interface for communication between clients and server
- Web Service APIs (eg. REST) implement a standard communication interface for cloud services
- REST API requests/responses are transferred on HTTP in common formats such as XML and JSON

Web Services for Cloud Applications

Example: Retrieve shipping rates from shipping company server. Different clients send requests through REST API. Server returns responses in JSON format.



Source: https://www.ibm.com/developerworks/cloud/library/cl-RESTfulAPIsincloud/index.html

Web Services for Cloud Applications

REST example: REST API from AWS S3 for Domain Name System (DNS) redirects

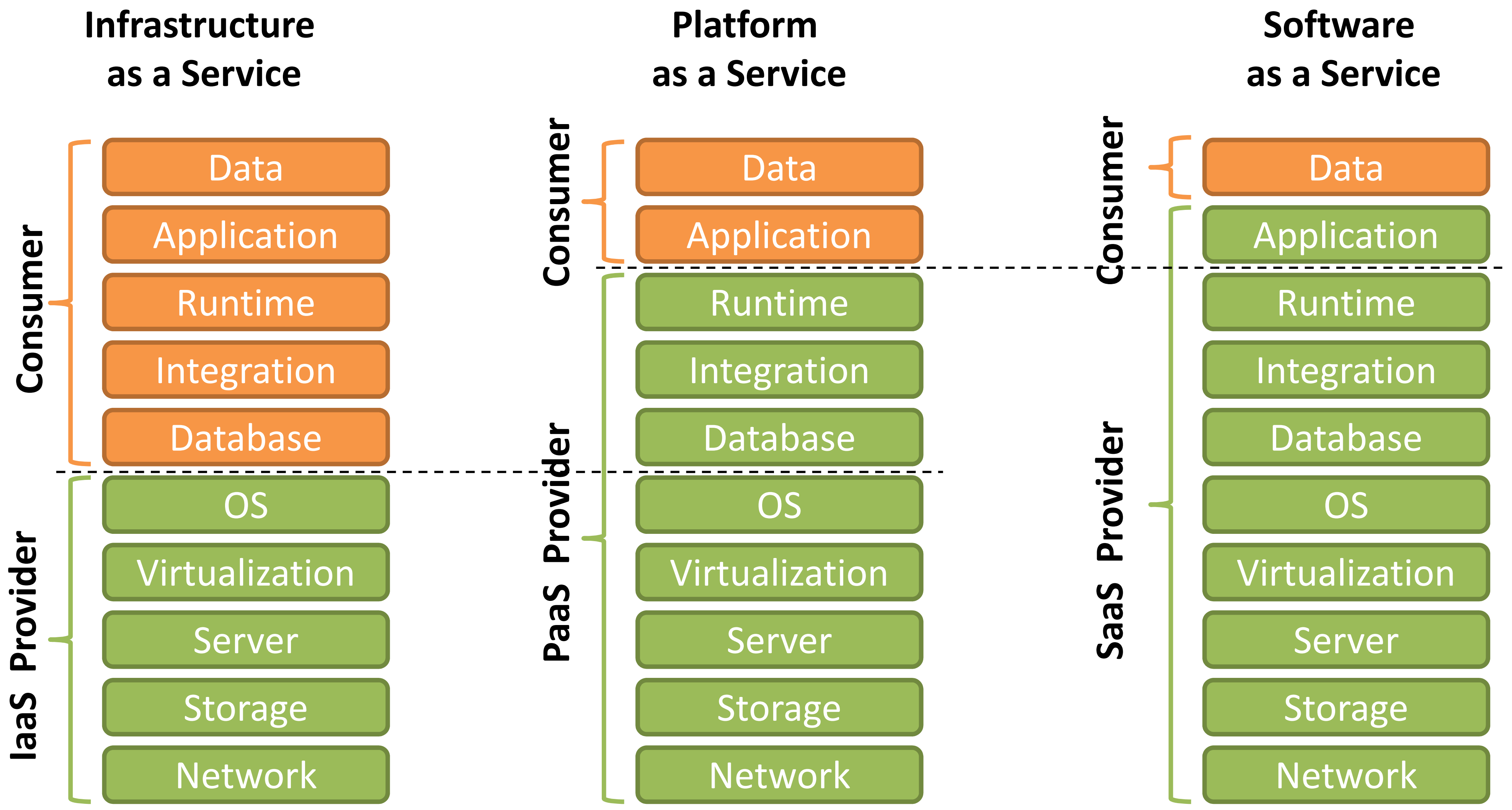
```
HTTP/1.1 307 Temporary Redirect
Location: http://johnsmith.s3-
gztb4pa9sq.amazonaws.com/photos/puppy.jpg?rk=e2c69a31
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Fri, 12 Oct 2007 01:12:56 GMT
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>TemporaryRedirect</Code>
  <Message>Please re-send this request to the specified temporary
endpoint.
  Continue to use the original request endpoint for future
requests.</Message>
  <Endpoint>johnsmith.s3-gztb4pa9sq.amazonaws.com</Endpoint>
</Error>
```

Source: https://www.ibm.com/developerworks/cloud/library/cl-RESTfulAPIsincloud/index.html

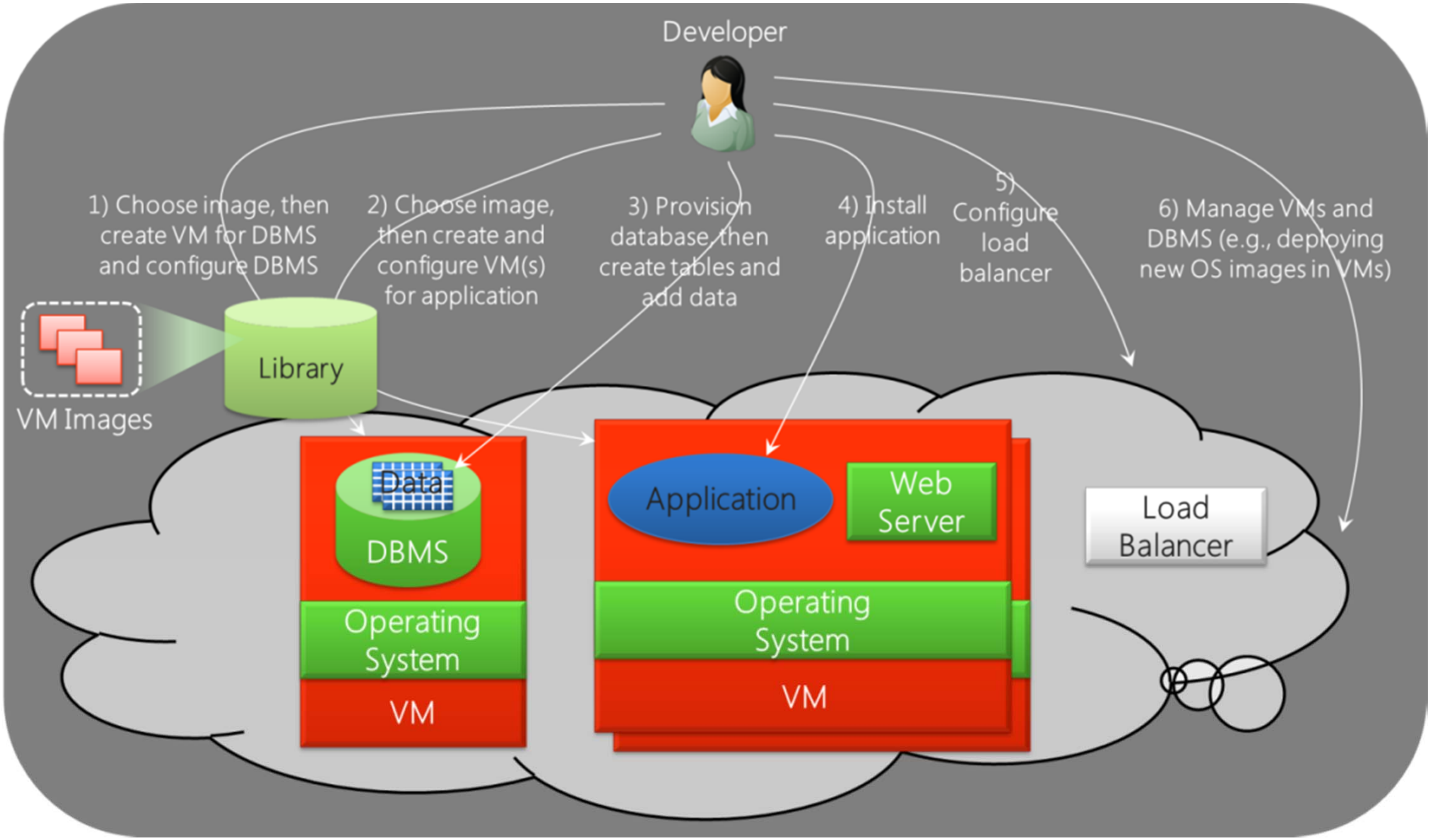
Cloud Application Development Models: Examples

- IaaS
- PaaS
- SaaS

IaaS vs PaaS vs SaaS



IaaS Development Model


Developer
1) Choose image, then create VM for DBMS and configure DBMS
2) Choose image, then create and configure VM(s) for application
3) Provision database, then create tables and add data
4) Install application
5) Configure load balancer
6) Manage VMs and DBMS (e.g., deploying new OS images in VMs)
Library
VM Images
Data
DBMS
Operating System
VM
Application
Web Server
Operating System
VM
Load Balancer

Mark Russinovich, at the BUILD conference

Example of IaaS Development

- A user wants to setup a blog – IaaS Consumer

- Steps:
  1. Creates an Amazon compute instance running Linux
  2. Installs and configures Apache, MySQL and PHP
  3. Installs and configures Wordpress blog engine
  4. Configures a new blog
  5. Blogs happily about IaaS development

♥ *IaaS*

Platform as a Service

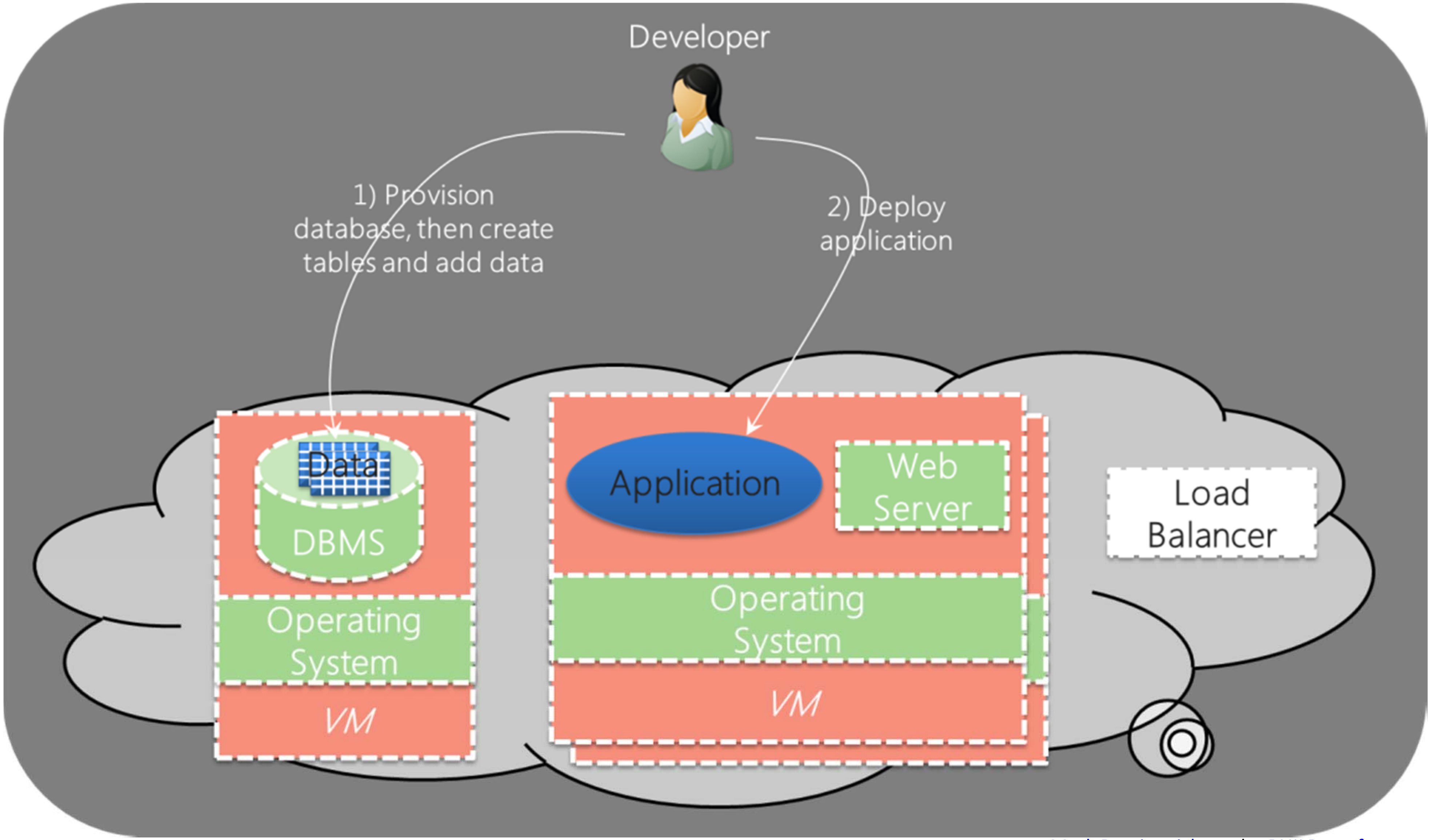
Consumer

- Creates the software using tools and libraries from the PaaS provider
- Controls software deployment and configuration settings

Provider

- Installs and maintains software, libraries and middleware
- Handles the transparent infrastructure from the user
- **Exposes an Application Programming Interface (API) for programmers**
- Provides integrated services of scalability, maintenance, and versioning
- E.g. total application hosting, development, testing, and deployment environment

PaaS Development Model



Mark Russinovich, at the BUILD conference

Example of PaaS Development

- A user wants to setup a blog – PaaS Consumer

- Steps:
  1. Accesses a system with Linux, Apache, MySQL and PHP (LAMP)
  2. Installs and configures Wordpress blog engine
  3. Setup a new blog
  4. Blogs happily about PaaS provisioning

♥ *PaaS*

Software as a Service

- Application is located on the cloud
- Clients use a thin client (web browser) to access the program
- Often built on top of multi-tiered PaaS and IaaS
  - **Clients do not deal with underlying PaaS and IaaS**
  - **Clients do not deal with APIs**
- Various pricing mechanisms
  - Free, pay-as-you-go, subscription, freemium etc.

Example of SaaS Application

- A user wants to setup a blog – SaaS Consumer

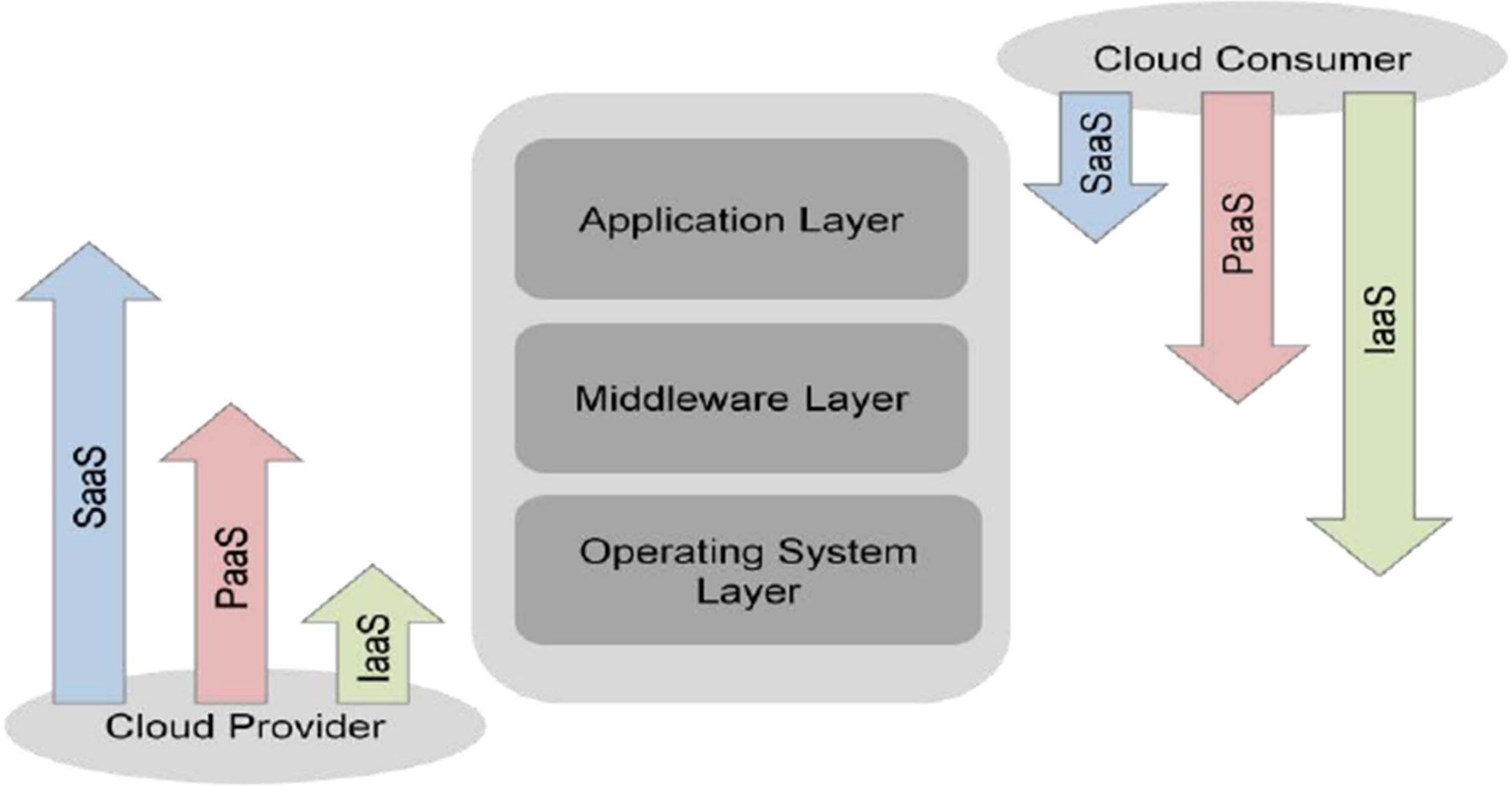
- Steps:
  1. Accesses a server with Wordpress installed and configured
  2. Setup a new blog
  3. Blogs happily about SaaS development

♥ *SaaS*

Steps in Cloud Application Development

IaaS	PaaS	SaaS
1. Creates an Amazon compute instance running Linux	---	---
2. Installs and configures Apache, MySQL and PHP	1. Accesses a system with Linux, Apache, MySQL and PHP (LAMP)	---
3. Installs and configures Wordpress blog engine	**2. Installs and configures Wordpress blog engine**	1. Accesses a server with Wordpress installed and configured
4. Configures a new blog	3. Setup a new blog	2. Setup a new blog
5. Blogs happily about IaaS development	4. Blogs happily about PaaS provisioning	3. Blogs happily about SaaS development

Cloud Service Models



MapReduce Programming Model

- supports arbitrarily divisible workload
- supports distributed computing on large data sets on multiple machines (clusters, public or private clouds)
- how large an amount of work?
  - web-scale data on the order of 100s of GBs to TBs or PBs
  - input data set will not likely fit on a single computer's hard drive
  - A distributed file system (e.g., Google File System- GFS) is typically required
- inspired by the **map** and **reduce** functions in functional programming languages

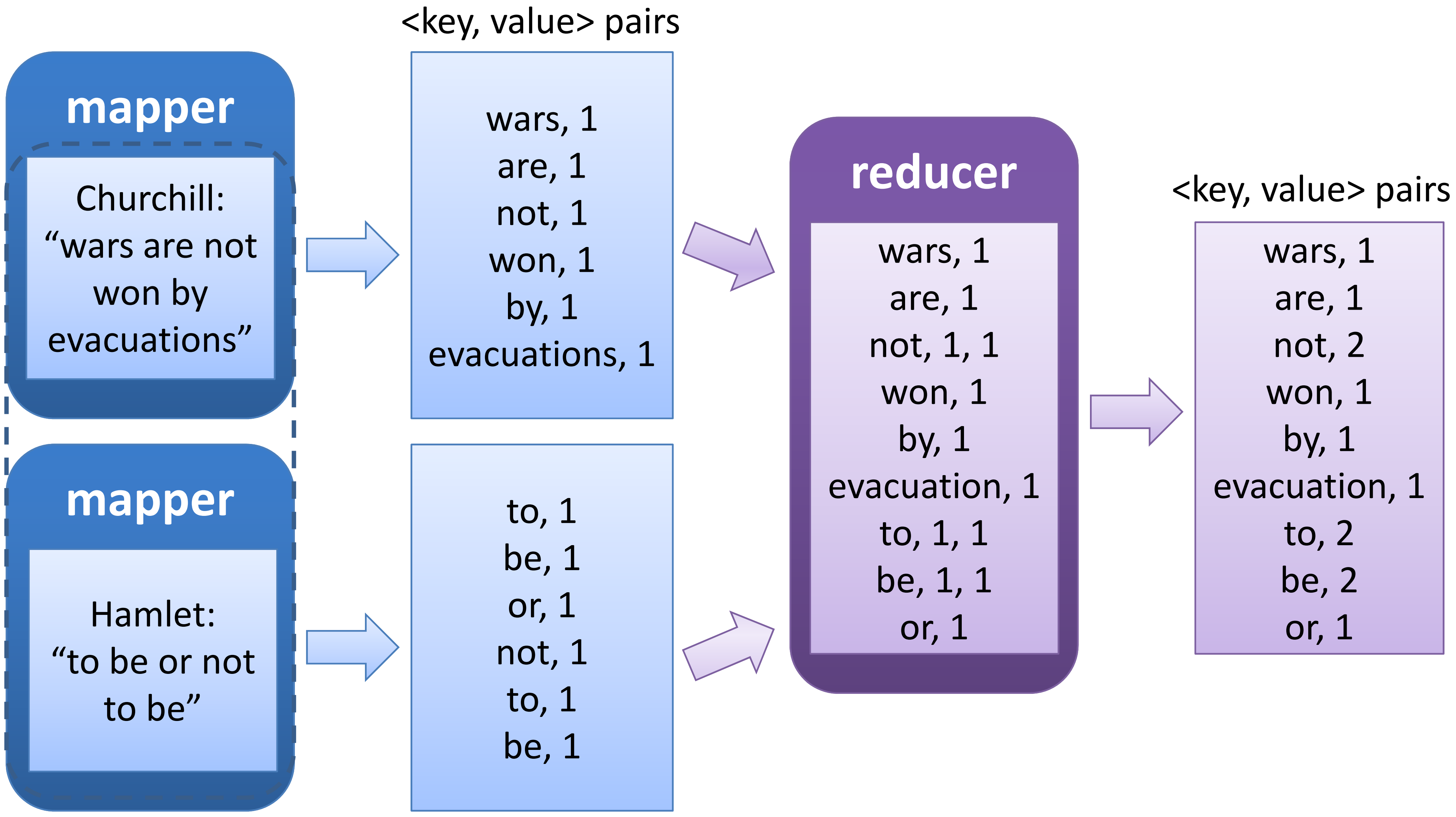
MapReduce Programming Model

- Key idea: (a) split data into blocks and assign each block to an instance/process for parallel execution (b) merge partial results produced by individual instances after all instances completed execution

- Transforms a set of input <key, value> pairs into a set of output <key, value> pairs:
  - Example: given a log of web page requests, count URL access frequency
    - **map**: output the pairs <URL, 1>
    - **reduce**: produce the pairs <URL, totalcount>

- SPMD (Same Program Multiple Data) – a master instance partitions the data and gathers the partial results

Example: Word Counting*



* for simplicity, we left out the sort phase

Example: Word Counting

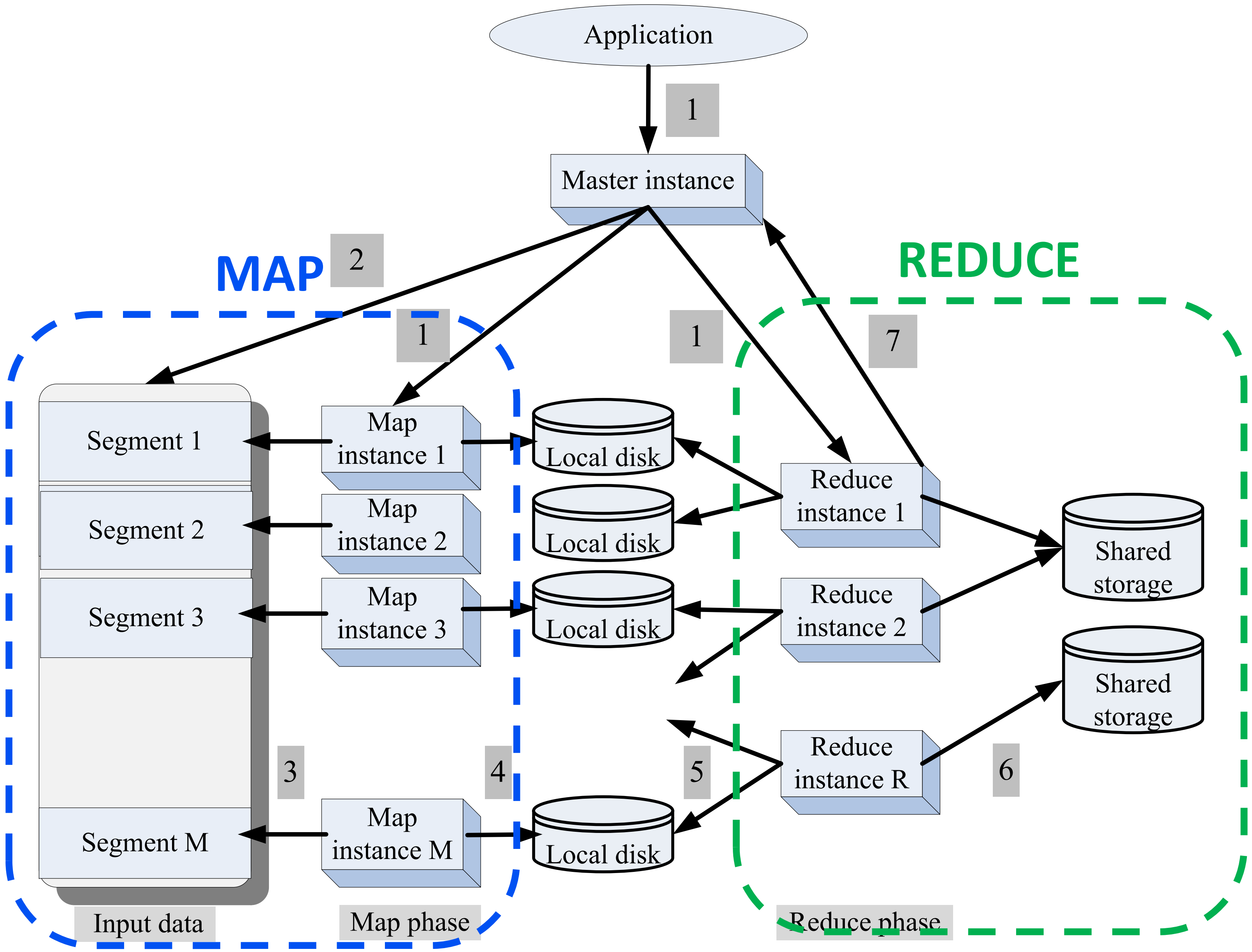
Count how many times each word appears in a set of documents

```
public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> output, Reporter reporter)
throws IOException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken
                output.collect(word, one);
        }
}
public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter)
throws IOException {
        int sum = 0;
        while (values.hasNext()) {
                sum += values.next().get()
                }
        output.collect(key, new IntWritabl
}
```

key = file name
value = file contents
write = signal word occurrence

key = word
values = map signals
write = signal **total** word occurrences

MapReduce


Application
1
Master instance
MAP
2
REDUCE
1
1
7
Segment 1
Segment 2
Segment 3
Segment M
Map instance 1
Map instance 2
Map instance 3
Map instance M
Local disk
Local disk
Local disk
Local disk
Reduce instance 1
Reduce instance 2
Reduce instance R
Shared storage
Shared storage
3
4
5
6
Input data
Map phase
Reduce phase

MapReduce

1. an application starts a **master instance**, M worker instances for the *Map phase* and later R worker instances for the *Reduce phase*
2. master instance partitions the input data in M *segments*
3. each *map instance* reads its input data segment and processes the data
4. results of the processing are stored on the local disks of the servers where the map instances run
5. when all map instances have finished processing their data, the R reduce instances read the results of the first phase and merge (collect) the partial results
6. final results are written by the reduce instances to a shared storage server
7. master instance monitors the reduce instances and when all of them report task completion the application is terminated

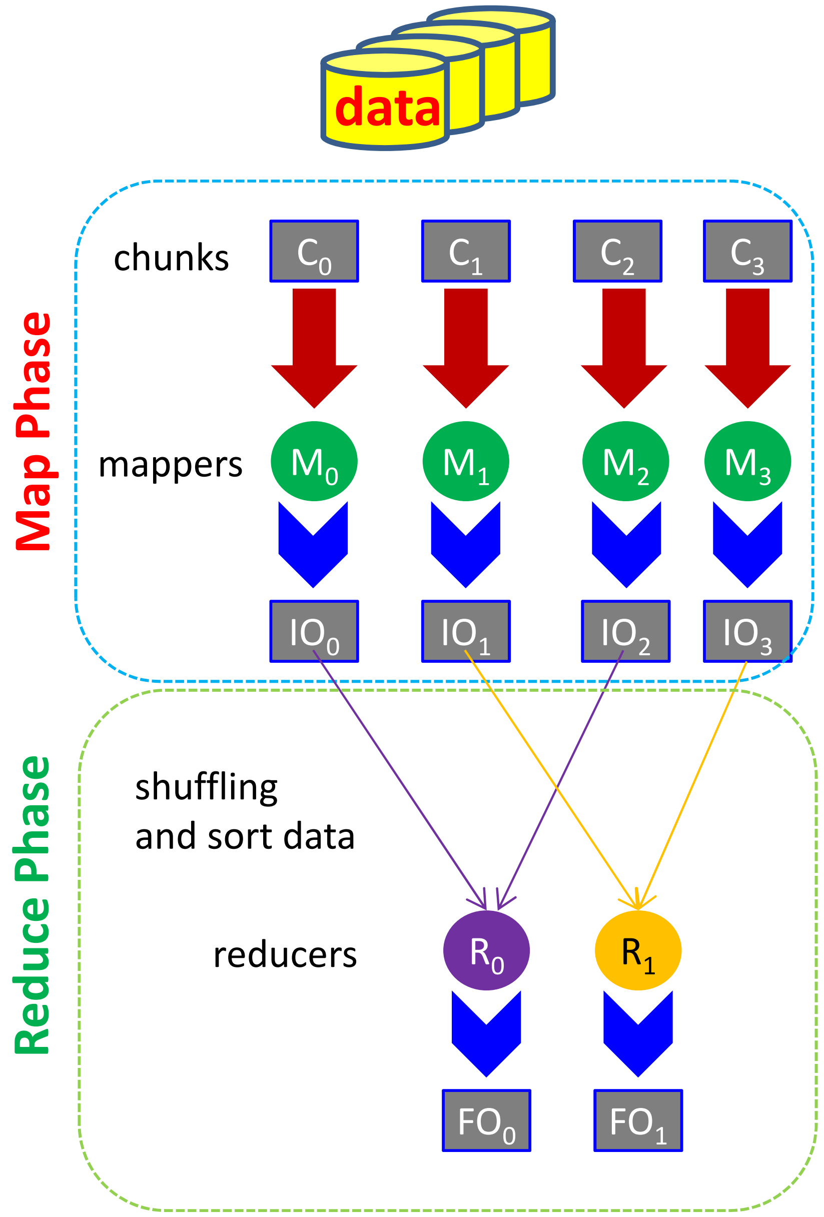
Structure of a MapReduce Program

1. Read (a lot of) data
2. MAP (extract data you need from each record)
3. Shuffle and Sort data
4. REDUCE (aggregate, summarize, filter, transform extracted data)
5. Write the results

Programmers implement the MAP and REDUCE functions to fit their problem

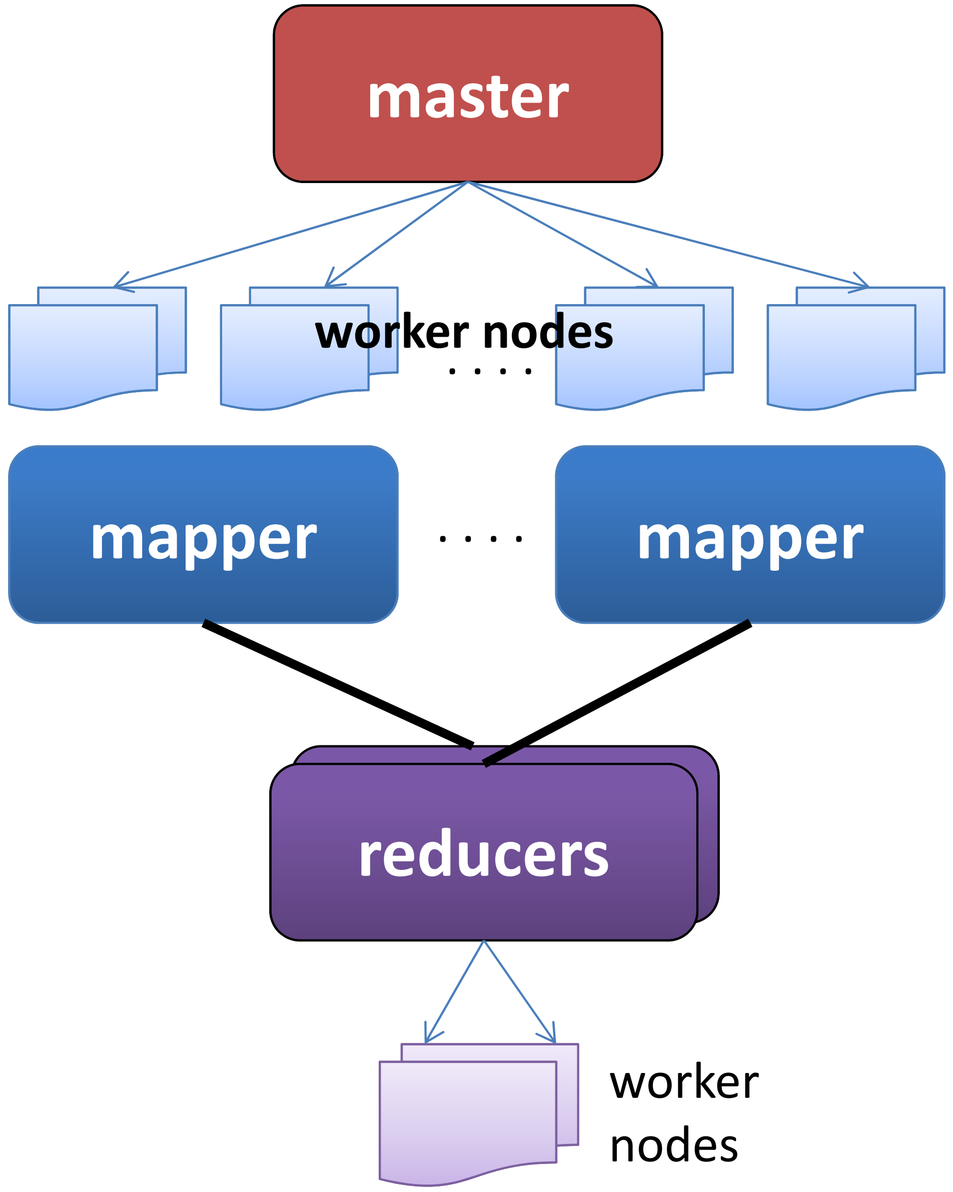
High-level View of MapReduce

- breaks the data flow into two ***map*** and ***reduce*** *phases*
- Chunks are processed in isolation by tasks called *Mappers*
- Outputs from mappers, called intermediate outputs (IOs), are brought into a second set of tasks called *Reducers*
- Process of bringing together IOs into a set of Reducers is called *shuffling*
- Reducers produce the final outputs (FOs)



Parallelism in MapReduce

- Master program creates data-localized tasks to minimize network communication
- Worker nodes switch between mapper and reducer
- **Map** workers **run in parallel**
  - each has its own data set
  - each creates intermediate local values from each input data set
- **Reduce** workers **run in parallel**
  - may reduce intermediate data
  - there is an optimal number of reducers for best performance



Applications of MapReduce

- Indexing for Google search
- Organizing articles in Google news
- Performing various user statistics

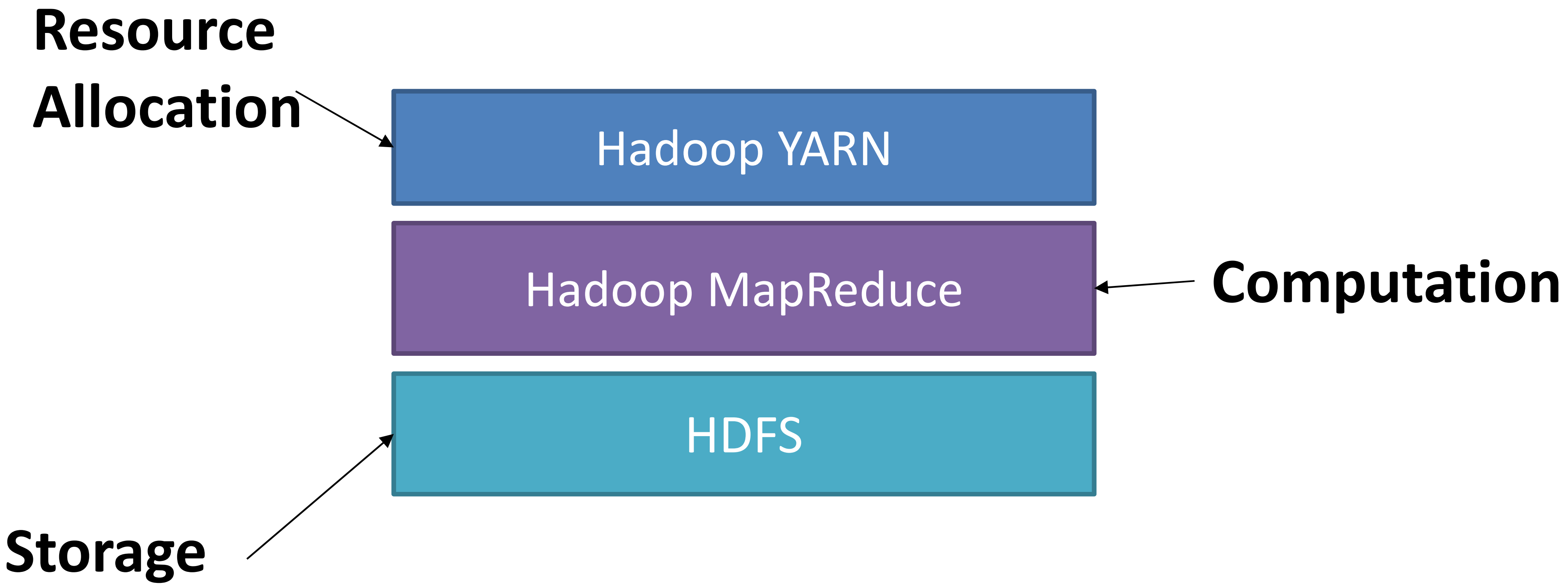
- Mapping Yahoo search
- Detecting spam in Yahoo mail

- Data Mining
- Optimizing ads for users
- Detecting spam on Facebook

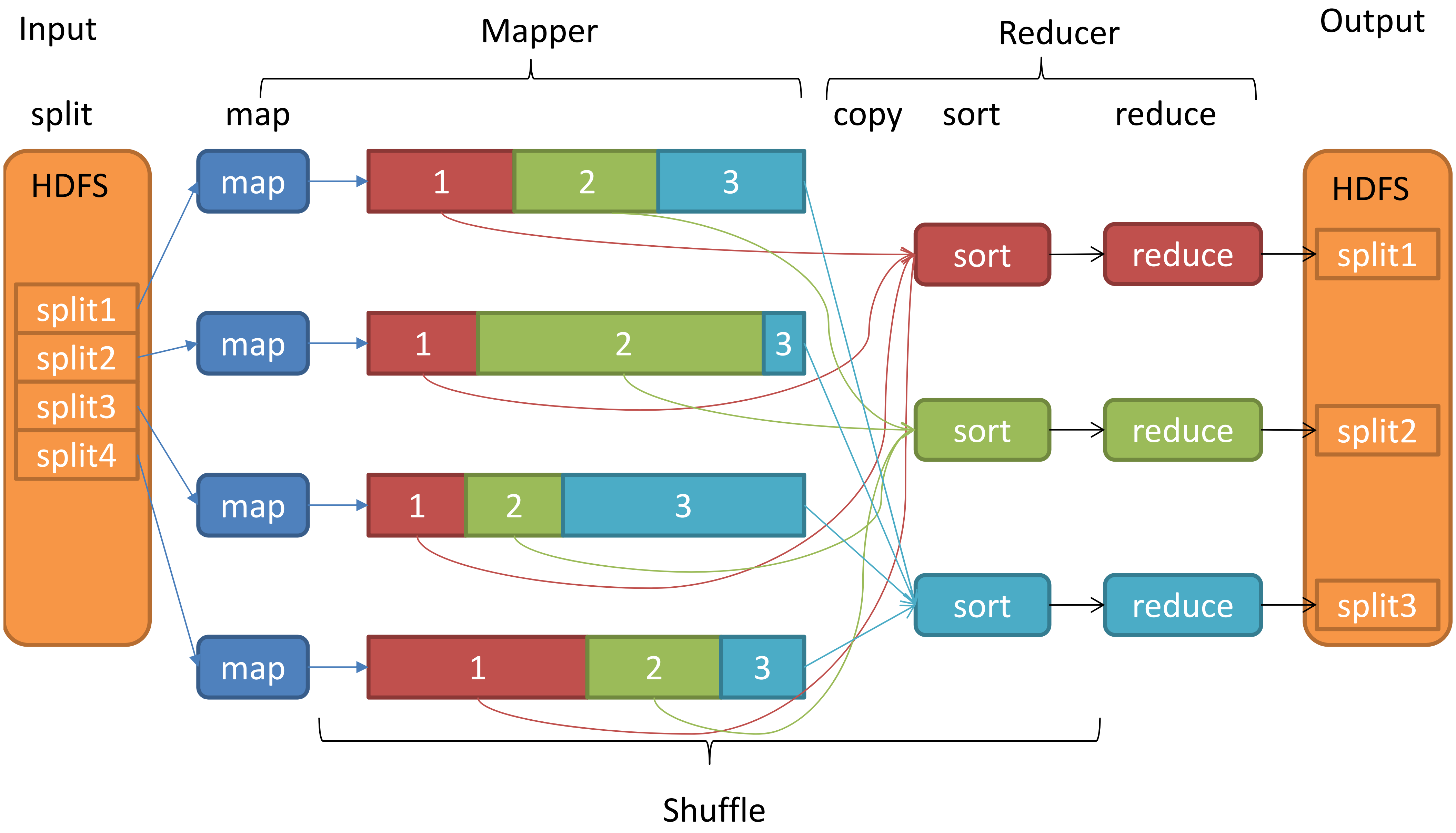
What is Hadoop?

- Google MapReduce - closed source developed by Google (2004) to process large amounts of raw data
- Hadoop MapReduce – open source developed by Apache + Yahoo (Java programming language) – a popular implementation of MapReduce
- Presents MapReduce as an analytics engine with a distributed storage layer referred to as Hadoop Distributed File System (*HDFS*); HDFS mimics Google File System (*GFS*)
- Amazon Elastic MapReduce creates a Hadoop cluster and handles data transfers between Amazon EC2 (computation) and Amazon S3 (storage)

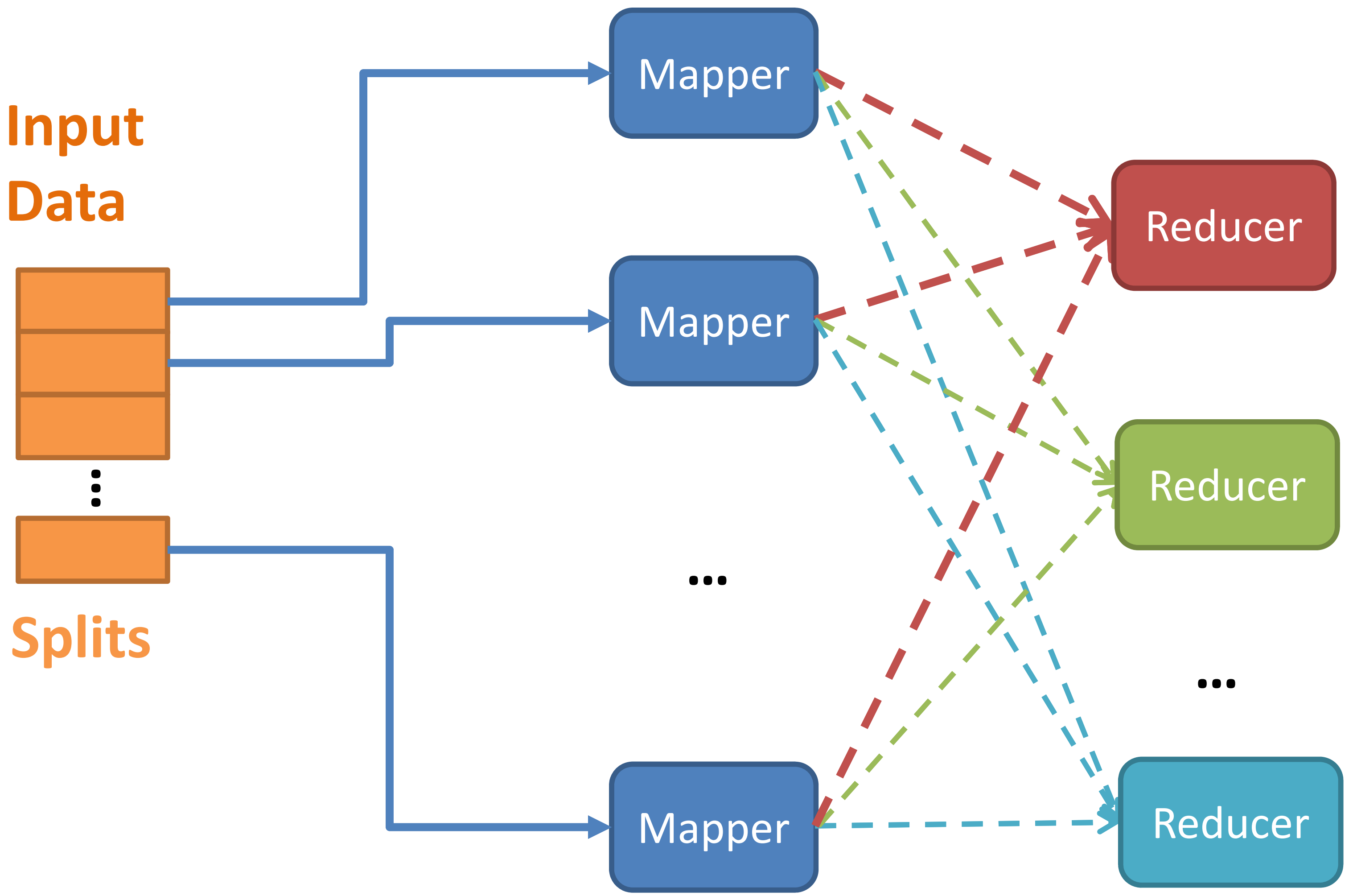
Hadoop Architecture



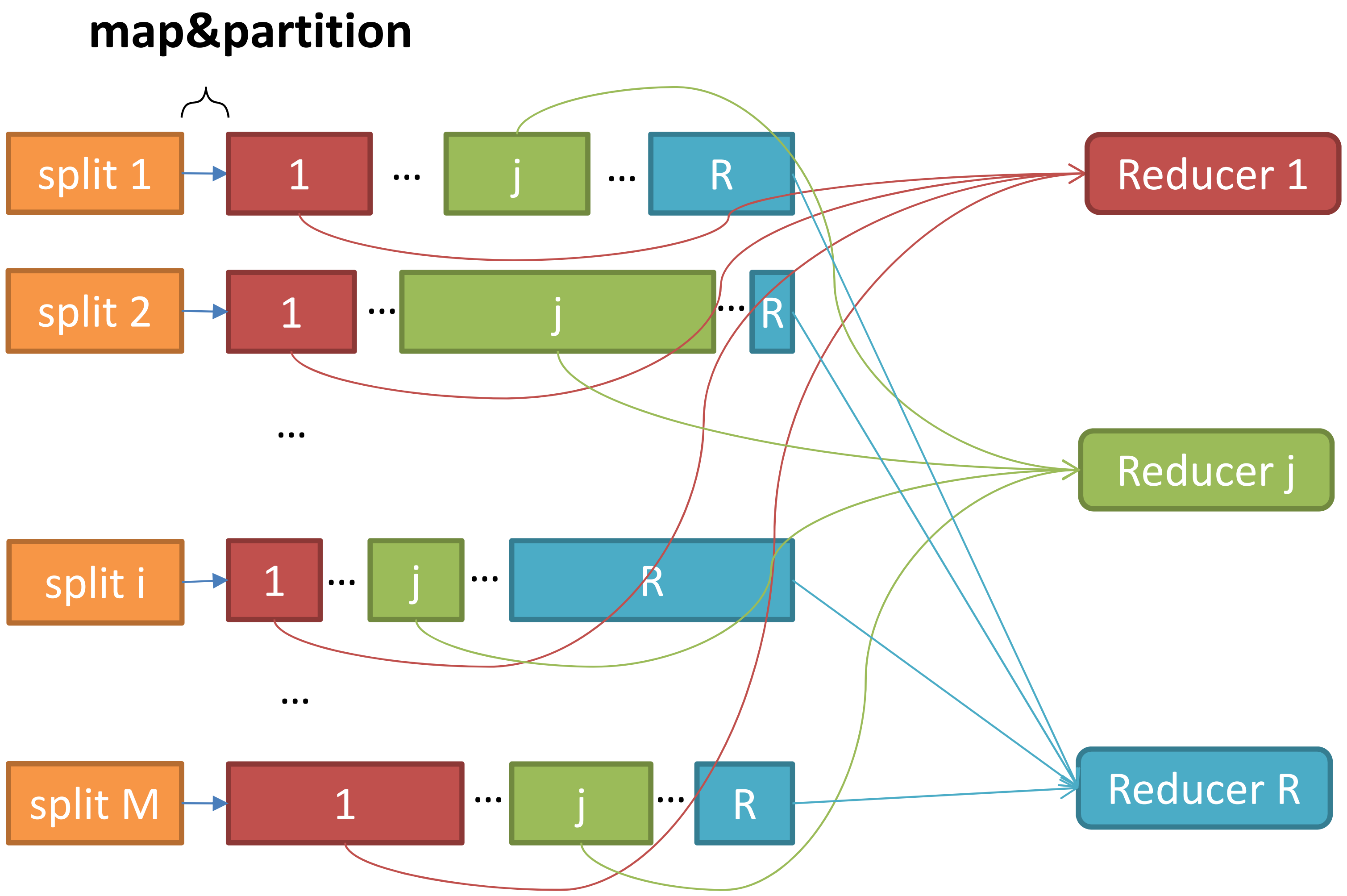
How Hadoop Works



Map


Input Data
Splits
Mapper
Mapper
...
Mapper
Reducer
Reducer
...
Reducer

Shuffle



map&partition
split 1
1
…
j
…
R
split 2
1
…
j
…
R
…
split i
1
…
j
…
R
…
split M
1
…
j
…
R
Reducer 1
Reducer j
Reducer R

Summary

- common features of cloud providers
- cloud applications: challenges in developing applications, architecture styles
- cloud application development models with examples
- MapReduce programming model for arbitrarily divisible workload
- Hadoop

References

- REST in the cloud: A primer on RESTful APIs in the cloud (https://www.ibm.com/developerworks/cloud/library/cl-RESTfulAPIsincloud/index.html)

- MapReduce: simplified data processing on large clusters (https://research.google.com/archive/mapreduce-osdi04.pdf)

- Apache Hadoop: http://hadoop.apache.org

- The Hadoop distributed file system (http://storageconference.us/2010/Papers/MSST/Shvachko.pdf)

Key Terms

1. Divisible workload
2. Performance isolation
3. Web application architecture layers
4. Application development models (IaaS, PaaS, SaaS)
5. MapReduce & Hadoop

Lecture on 20 Jul

0900-1030	lecture
1030-1045	break
1045-1200	lecture
1200-1400	lunch
1400-1445	lecture
1445-1500	break
1500-1545	lecture
1545-1600	break
1600-1700	assessment 2

L04: Applications & Paradigm

L05: IBM Bluemix
Cloud Services

L06: Projects