ΕΠΛ 602:Foundations of Internet Technologies

Cloud Computing

Above the Clouds Presentation

*Based on "**Above the Clouds: A Berkeley View of Cloud Computing**" by Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Technical Report EECS-2009-28, EECS Department, University of California, Berkeley.*

Above the Clouds
A Berkeley View of Cloud Computing

UC Berkeley RAD Lab

Outline

- What is it?
- Why now?
- Cloud killer apps
- Economics for users
- Economics for providers
- Challenges and opportunities
- Implications

WEB is replacing the Desktop

What is Cloud Computing?

What is the "cloud"?

- Many answers. Easier to explain with examples:
  - Gmail is in the cloud
  - Amazon (AWS) EC2 and S3 are the cloud
  - Google AppEngine is the cloud
  - Windows Azure is the cloud
  - SimpleDB is in the cloud
  - The "network" (cloud) is the computer

What is Cloud Computing?

A Cloud is

- an infrastructure, transparent to the end-user,
- which is used by a company or organization to provide services to its customers
- via ***network***
- where the infrastructure resources are used ***elastically*** and the
- customer is ***charged according to usage***.

under the hood: large clusters of commodity hardware

terms: virtualization, elasticity, utility computing, pay-as-you-go

What is Cloud Computing?

What about Wikipedia?

“**Cloud computing** is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a utility (like the electricity grid) over a network (typically the Internet).“

What is Cloud Computing?

- Delivering applications and services over the Internet:
  - Software as a service

- Extended to:
  - Infrastructure as a service: Amazon EC2
  - Platform as a service: Google AppEngine, Microsoft Azure

  Poorly defined so we avoid *all* "**X** as a service"

- Utility Computing: pay-as-you-go computing
  - Illusion of infinite resources (no need to plan ahead)
  - No up-front cost (start small)
  - Fine-grained billing (e.g. hourly)

What is Cloud Computing?

Cloud Computing:

- the applications delivered as services over the Internet (SaaS) and
- the hardware and systems software in the datacenters that provide these services (a Cloud)

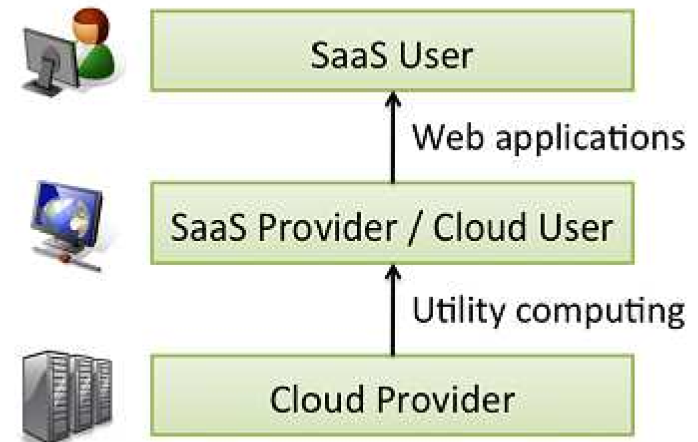
Public Cloud: a Cloud made available in a pay-as-you-go manner to the public

Utility Computing: the service being sold

Private Cloud: the internal datacenters of a business or other organization that are not made available to the public.

Cloud Computing: SaaS and Utility Computing

roles of the people as *users* or *providers* of these layers of Cloud Computing



What is Cloud Computing?

Cloud Services

- Data Storage

  Examples:

  AWS Simple Storage System (S3)

- Infrastructure as a Sevice (IaaS)

  Provide computing instances (e.g., servers running Linux) as a service.

  Examples:

  AWS Elastic Computing Cloud (EC2).

- Platform as a Service (PaaS)

  The delivery of a computing platform and solution stack as a service via network.

  Examples:

  Google AppEngine

- Software as a Service (SaaS)

  Software that is deployed over network as a service.

  Examples:

  Google Documents

  Google Calendar

  Google Reader

What is Cloud Computing?

Some Cloud Providers

- Amazon Web Services (AWS)
  http://aws.amazon.com/
  The most complete set of Cloud services.
- Google App Engine
  http://code.google.com/appengine/
- IBM Cloud
  http://www.ibm.com/ibm/cloud/
- Microsoft Windows Azure
  http://www.microsoft.com/windowsazure/

Cloud Computing: Why Now?

> If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry.

—John McCarthy, speaking at the MIT Centennial in 1961[2]

Cloud Computing: Why Now?

- Experience with very large datacenters
  - Unprecedented economies of scale
  - Transfer of risk

  Examples, PayPal, Google AdSense, Amazon CloudFront
- Technology factors
  - Pervasive broadband Internet
  - Maturity in Virtualization Technology
- Business factors
  - Minimal capital expenditure
  - Pay-as-you-go billing model
  - Web 2.0: shift fir “high-touch, high-margin high-commitment”- “low-touch, low-margin low-commitment” (pay-as-you-go computing with no contract)

  Examples, PayPal, Google AdSense, Amazon CloudFront

Cloud Killer Apps

Factor in the cost of moving data

- Mobile and web applications
  - Availability (connectivity), large data sets, mash-ups (combine more data sources or services)
- Extensions of desktop software
  - Matlab, Mathematica
- Batch processing / MapReduce
  - Oracle at Harvard, Hadoop at NY Times
  - Business analytics (understanding customers, supply chains, buying habits, etc), decision support

Spectrum of Clouds

An application needs

- a model of computation,
- a model of storage and,
- a model of communication.

The *statistical multiplexing (multi tenancy)* necessary to achieve elasticity and the illusion of infinite capacity ==>*resources are virtualized*

So that the implementation of how they are multiplexed and shared can be hidden from the programmer.

Different utility computing offerings distinguished based on the level of abstraction presented to the programmer and the level of management of the resources.

Spectrum of Clouds

At one end of the spectrum: hardware virtual machines -- Amazon EC2

- An EC2 instance looks much like *physical hardware*
- Users can control nearly *the entire software stack, from the kernel upwards*.
- Low level of virtualization—raw CPU cycles, block-device storage, IP-level connectivity

Thin API: a few dozen API calls to request and configure the virtualized hardware. *No a priori limit on the kinds of applications* that can be hosted; developers code whatever they want.

Inherently difficult to offer automatic scalability and failover, because the semantics associated with replication and other state management issues highly application-dependent.

AWS does offer a number of higher-level managed services, including several different managed storage services for use in conjunction with EC2, such as SimpleDB.
Oracle databases hosted on AWS

Spectrum of Clouds

At the other end of the spectrum: application domain-specific platforms --
Google AppEngine and Force.com

Google AppEngine targeted exclusively at traditional web applications

AppEngine application
- Structured with clean separation between a stateless computation tier and a stateful storage tier (proprietary MegaStore -- based on BigTable)
- Expected to be request-reply based (severely rationed in how much CPU time they can use in servicing a particular request)

Impressive automatic scaling and high-availability mechanisms

Not suitable for general-purpose computing.

Force.com (SalesForce business software development platform)
Designed to support only business applications that run against the salesforce.com database

Spectrum of Clouds

An intermediate point *on the spectrum of flexibility vs. programmer convenience:* Microsoft's Azure

Azure applications are

- written using the .NET libraries, and
- compiled to the Common Language Runtime, a language-independent managed environment.

The system supports general-purpose computing

Users get a choice of language, but cannot control the underlying operating system or runtime.

The libraries provide a degree of automatic network confguration and failover/scalability, but require the developer to declaratively specify some application properties in order to do so.

Spectrum of Clouds

Which one is the best model?

An analogy with programming languages and frameworks.

- Low-level languages such as C and assembly language allow fine control and close communication with the bare metal,
but if writing a Web application, the mechanics of managing sockets, dispatching requests, and so on are cumbersome and tedious to code, even with good libraries.

- High-level frameworks such as Ruby on Rails make these mechanics invisible to the programmer, but are only useful if the application readily fits the request/reply structure and the abstractions provided by Rails

Different tasks will result in demand for different classes of utility computing.

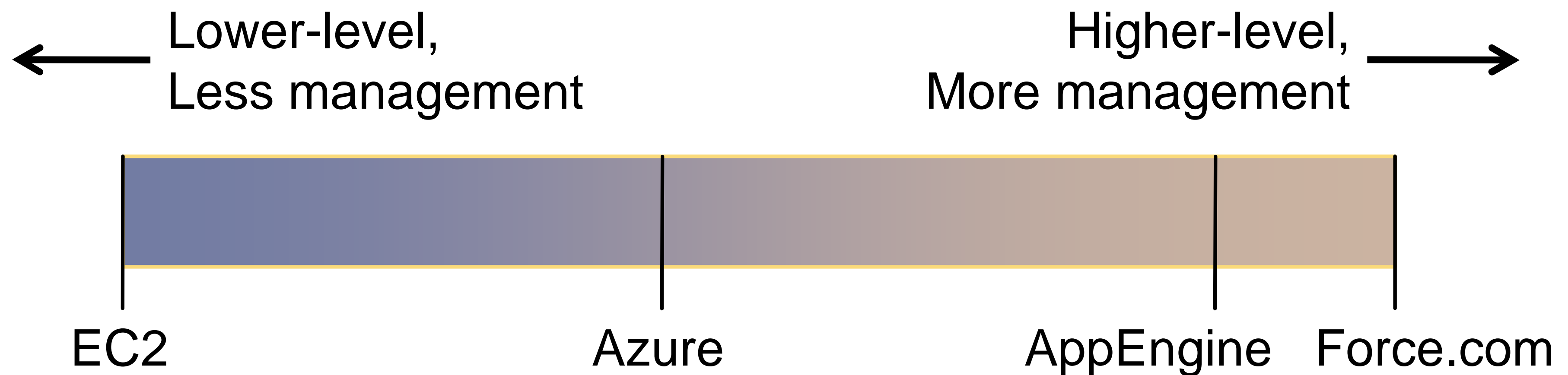
Just as high-level languages can be implemented in lower-level ones, highly-managed cloud platforms can be **hosted on top** of less-managed ones (?proprietary)

Table 4: Examples of Cloud Computing vendors and how each provides virtualized resources (computation, storage, networking) and ensures scalability and high availability of the resources.

	Amazon Web Services	Microsoft Azure	Google AppEngine
Computation model (VM)	• x86 Instruction Set Architecture (ISA) via Xen VM • Computation elasticity allows scalability, but developer must build the machinery, or third party VAR such as RightScale must provide it	• Microsoft Common Language Runtime (CLR) VM; common intermediate form executed in managed environment • Machines are provisioned based on declarative descriptions (e.g. which "roles" can be replicated); automatic load balancing	• Predefined application structure and framework; programmer-provided "handlers" written in Python, all persistent state stored in MegaStore (outside Python code) • Automatic scaling up and down of computation and storage; network and server failover; all consistent with 3-tier Web app structure
Storage model	• Range of models from block store (EBS) to augmented key/blob store (SimpleDB) • Automatic scaling varies from no scaling or sharing (EBS) to fully automatic (SimpleDB, S3), depending on which model used • Consistency guarantees vary widely depending on which model used • APIs vary from standardized (EBS) to proprietary	• SQL Data Services (restricted view of SQL Server) • Azure storage service	•MegaStore/BigTable
Networking model	• Declarative specification of IP-level topology; internal placement details concealed • Security Groups enable restricting which nodes may communicate • Availability zones provide abstraction of independent network failure • Elastic IP addresses provide persistently routable network name	• Automatic based on programmer's declarative descriptions of app components (roles)	• Fixed topology to accommodate 3-tier Web app structure • Scaling up and down is automatic and programmer-invisible

Spectrum of Clouds

- Instruction Set VM (Amazon EC2, 3Tera)
- Bytecode VM (Microsoft Azure)
- Framework VM
  - Google AppEngine, Force.com

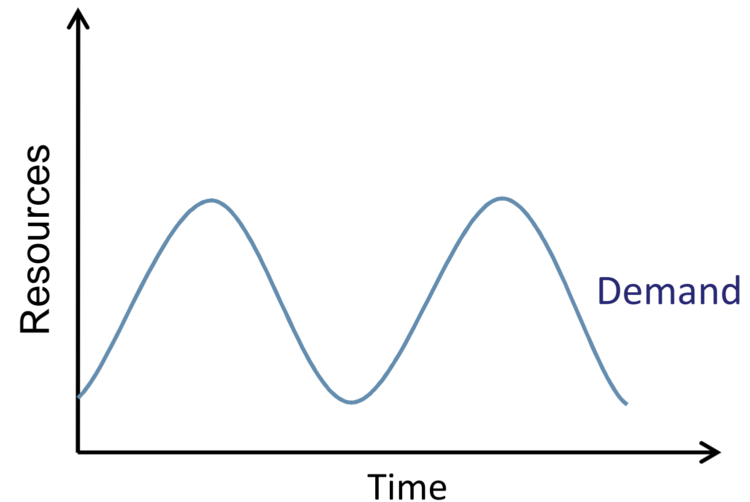


Outline

- What is it?
- Why now?
- Cloud killer apps
- **Economics for users**
- **Economics for providers**
- Challenges and opportunities
- Implications

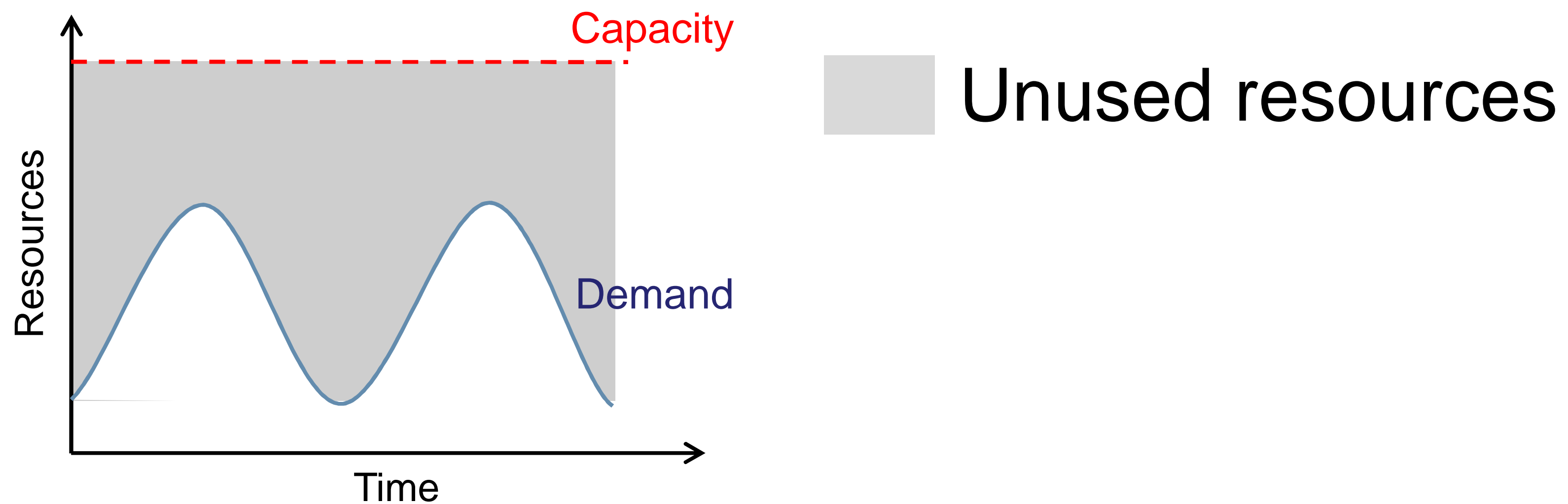
Economics of Cloud Users

- Many cloud applications have cyclical demand curves
  - Daily, weekly, monthly, …
- Workload spikes more frequent and significant
  - Death of Michael Jackson:
    - 22% of tweets, 20% of Wikipedia traffic, Google thought they are under attack
  - Obama inauguration day: 5x increase in tweets



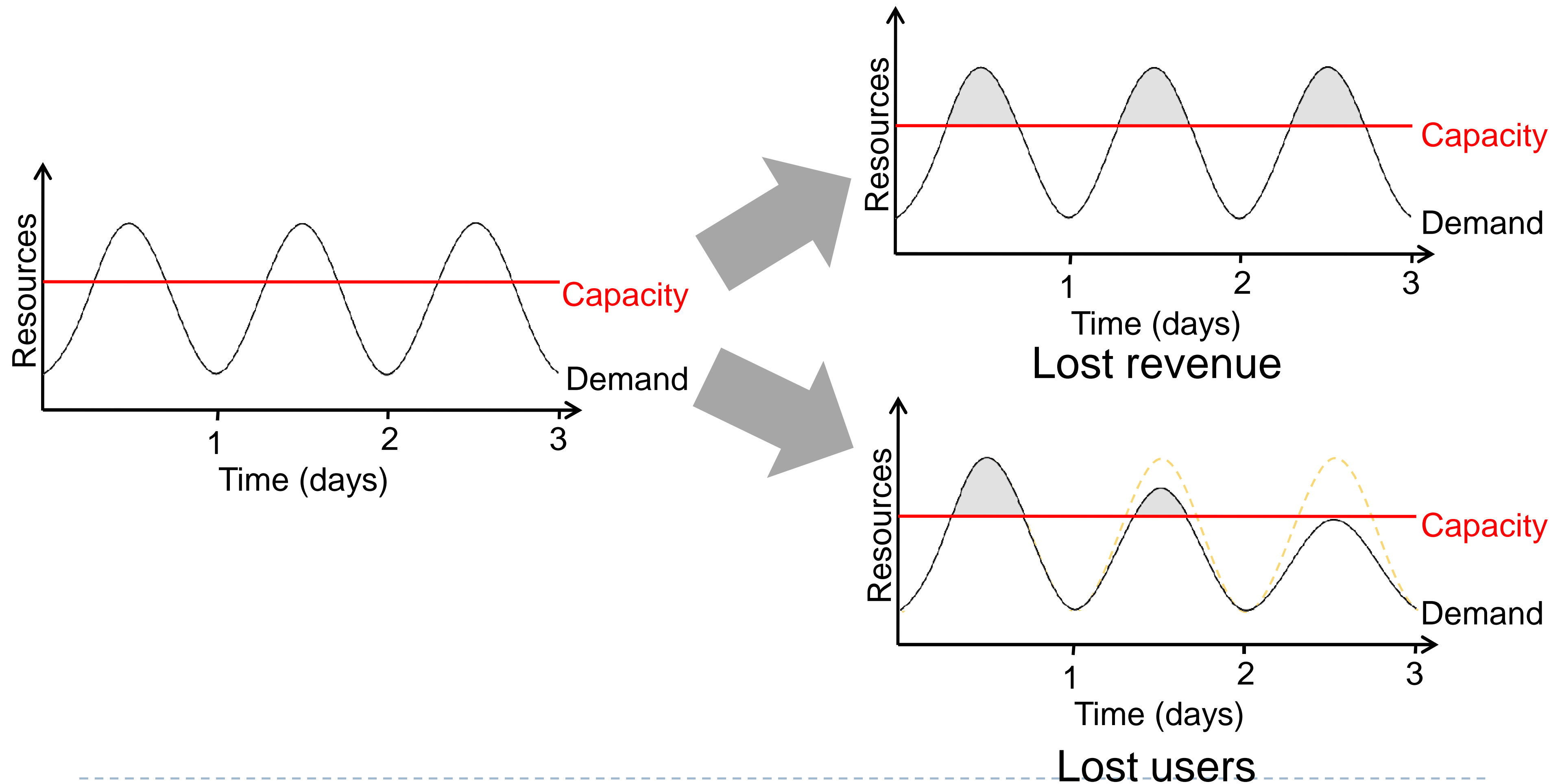
Economics of Cloud Users

- Risk of over-provisioning: underutilization



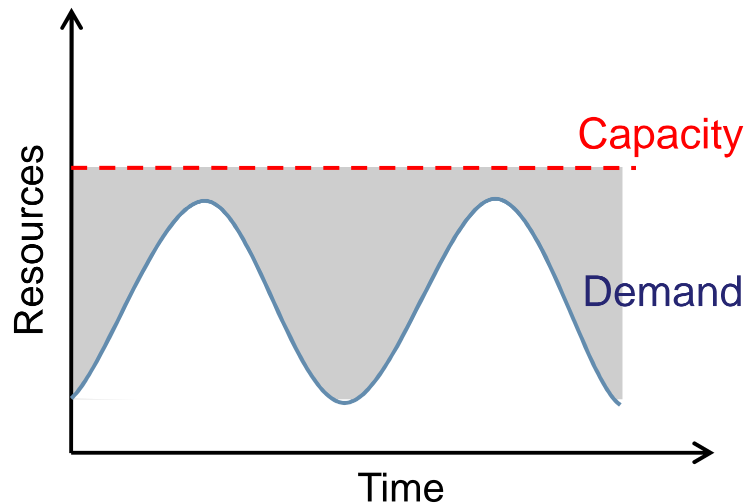
Economics of Cloud Users

- Heavy penalty for under-provisioning

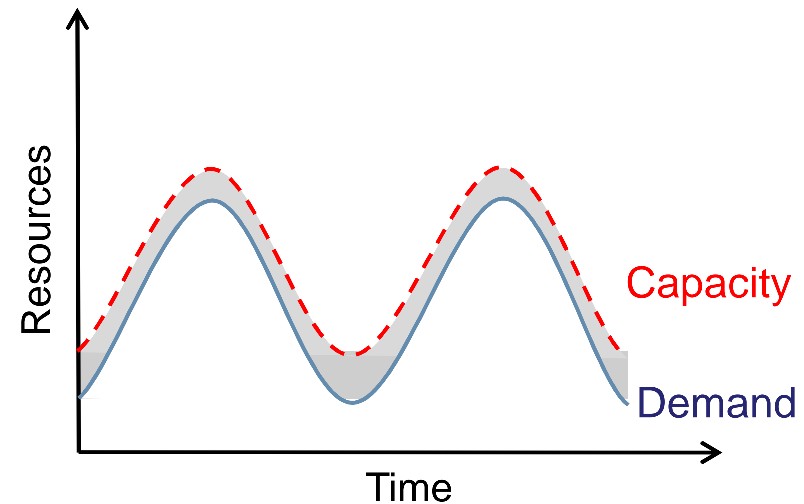


Economics of Cloud Users

- Pay by use instead of provisioning for peak



Static data center (provisioning for peak load)



Data center in the cloud

Unused resources

Economics of Cloud Users

$$\text{UserHours}_{\text{cloud}} \times (\text{revenue} - \text{Cost}_{\text{cloud}}) \geq \text{UserHours}_{\text{datacenter}} \times (\text{revenue} - \frac{\text{Cost}_{\text{datacenter}}}{\text{Utilization}})$$

Left: expected profit for
(revenue realized per user-hour minus cost of paying Cloud Computing per user-hour) X the number of user-hours
Right-hand side performs the same calculation for a fixed-capacity datacenter
by factoring in the average utilization, including nonpeak workloads. Whichever side is greater represents the opportunity for higher profit.
Apparently, if Utilization = 1:0 (the datacenter equipment is 100% utilized), the two sides of the equation look the same. However, basic queueing theory tells us that as utilization approaches 1.0, system response time approaches infinity. In practice, the usable capacity of a datacenter (without compromising service) is typically 0.6 to 0.8. Whereas a datacenter must necessarily overprovision to account for this “overhead,” the cloud vendor can simply factor it into Cost
cloud. (This overhead explains why we use the phrase “pay-as-you-go” rather than rent or lease for
utility computing. The latter phrases include this unusable overhead, while the former doesn’t. Hence, even if you lease a 100 Mbits/second Internet link, you can likely use only 60 to 80 Mbits/second in practice.)
The equation makes clear that the common element in all of our examples is the ability to control the cost per user-hour of operating the service. In Example 1, the cost per user-hour without elasticity was high because of resources sitting idle—higher costs but same number of user-hours. The same thing happens when over-estimation of demand results in provisioning for workload that doesn’t materialize. In Example 2, the cost per user-hour increased as a result
of underestimating a spike and having to turn users away: Since some fraction of those users never return, the fixed costs stay the same but are now amortized over fewer user-hours. This illustrates fundamental limitations of the “buy”
model in the face of any nontrivial burstiness in the workload.

Economics of Cloud Users

control the cost per user-hour of operating the service.

- (over-provisioning/over-estimation of demands) the cost per user-hour without elasticity high because of resources sitting idle—higher costs but same number of user-hours.
- (under-provisioning) the cost per user-hour increases as a result of underestimating a spike and having to turn users away: some fraction of those users never return, the fixed costs stay the same but are now amortized over fewer user-hours.

Economics of Cloud Users

Pay separately per resource
Power, cooling and physical plan costs
Operations cost

Economics of Cloud Users

Example: Moving to cloud. Suppose a biology lab creates 500 GB of new data for every wet lab experiment. A computer the speed of one EC2 instance takes 2 hours per GB to process the new data. The lab has the equivalent 20 instances locally, so the time to evaluate the experiment is $500 \times 2/20$ or 50 hours. They could process it in a single hour on 1000 instances at AWS. The cost to process one experiment would be just $1000 \times \$0.10$ or $100 in computation and another $500 \times \$0.10$ or $50 in network transfer fees. So far, so good. They measure the transfer rate from the lab to AWS at 20 Mbits/second. [19] The transfer time is $(500GB \times 1000MB/GB \times 8bits/Byte)/20Mbits/sec = 4,000,000/20 = 200,000$ seconds or more than 55 hours. Thus, it takes 50 hours locally vs. 55 + 1 or 56 hours on AWS, so they don't move to the cloud. (The next section offers an opportunity on how to overcome the transfer delay obstacle.)

Utility Computing Arrives

- Amazon Elastic Compute Cloud (EC2)
- "Compute unit" rental: $~~0.10-0.80~~ 0.085-0.68/hour
  - 1 CU ≈ 1.0-1.2 GHz 2007 AMD Opteron/Intel Xeon core

	Platform	Units	Memory	Disk
Small - ~~$0.10~~ $.085/hour	32-bit	1	1.7GB	160GB
Large - ~~$0.40~~ $0.35/hour	64-bit	4	7.5GB	850GB – 2 spindles
X Large - ~~$0.80~~ $0.68/hour	64-bit	8	15GB	1690GB – 4 spindles
High CPU Med - ~~$0.20~~ $0.17	64-bit	5	1.7GB	350GB
High CPU Large - ~~$0.80~~ $0.68	64-bit	20	7GB	1690GB
High Mem X Large - $0.50	64-bit	6.5	17.1GB	1690GB
High Mem XXL - $1.20	64-bit	13	34.2GB	1690GB
High Mem XXXL - $2.40	64-bit	26	68.4GB	1690GB

Northern VA cluster

- No up-front cost, no contract, no minimum
- Billing rounded to nearest hour (also regional, spot pricing)
- New paradigm(!) for deploying services?, HPC?

Cloud properties

- Cloud offers:
  - Scalability : scale out vs scale up (also scale back) means that you (can) have infinite resources, can handle unlimited number of users
  - Reliability (hopefully!)
  - Availability (24x7)
  - *Elasticity* : pay-as-you go depending on your demand you can add or remove computer nodes and the end user will not be affected/see the improvement quickly.
  - Utility computing (similar to electrical grid)
  - Multi-tenancy enables sharing of resources and costs across a large pool of users. Lower cost, higher utilization … but other issues: e.g. security.

Economics of Cloud Providers

Who could become a Cloud Computing Provider? (large data centers, large scale software infrastructure, operational expertise)

Economics of Cloud Providers

- 5-7x economies of scale [Hamilton 2008]

Resource	Cost in Medium DC	Cost in Very Large DC	Ratio
Network	$95 / Mbps / month	$13 / Mbps / month	7.1x
Storage	$2.20 / GB / month	$0.40 / GB / month	5.7x
Administration	≈140 servers/admin	>1000 servers/admin	7.1x

- Extra benefits
  - Amazon: utilize off-peak capacity
  - Microsoft: sell .NET tools
  - Google: reuse existing infrastructure

Outline

- What is it?
- Why now?
- Cloud killer apps
- Economics for users
- Economics for providers
- **Challenges and opportunities**
- **Implications**

Adoption Challenges

Challenge	Opportunity
Availability	Multiple providers & DCs
Data lock-in	Standardization

Think about Google search, or gmail?
Multiple providers (a single one, common software infrastructure, accounting system or may even go out of business)

Threaten SaaS providers by making their service unavailable, use large "botnets" that rent bots on the black market -> utility computing offer SaaS the opportunity to defend by scale-up

APIs for clouds are still proprietary, not easy extract own data and programs from one site to run on another

Adoption Challenges

Challenge	Opportunity
Data Confidentiality and Auditability	Encryption, VLANs, Firewalls; Geographical Data Storage

Auditability (third party)
Keep data and copyrighted material within national boundaries, or do not like for a country to get access to their data via the court system, etc

Growth Challenges

Challenge	Opportunity
Data transfer bottlenecks	FedEx-ing disks, Data Backup/Archival
Performance unpredictability	Improved VM support, flash memory, scheduling VMs

CPU and main memory sharing predictable but not I/O
Scheduling of VMs for some classes of batch processing, especially for HPC (ensure that all threads of a program are running simultaneously)

Growth Challenges

Challenge	Opportunity
Scalable storage	Invent scalable store (e.g., schemaless blobs, column-oriented storage, etc)
Bugs in large distributed systems	Invent Debugger that relies on Distributed VMs
Scaling quickly	Invent Auto-Scaler that relies on ML; Snapshots

Bugs cannot be reproduced
Scale quickly up and down in response to load in order to save money but without violating service level agreements

Policy and Business Challenges

Challenge	Opportunity
Reputation Fate Sharing	Offer reputation-guarding services like those for email
Software Licensing	Pay-for-use licenses; Bulk use sales

One customer's bad behavior may affect the cloud as a whole
Transfer of legal liability

Eg an EC2 instance running Microsoft windows costs $0.15 per hour instead if the traditional 0.10 per hour for the open source version

Short Term Implications

- Startups and prototyping
- One-off tasks
  - Washington post, NY Times
- Cost associativity for scientific applications
- Research at scale

Long Term Implications

- Application software:
  - Cloud & client parts, disconnection tolerance
    - Cloud part needs to scale down rapidly as well as scale up
    - Client piece work disconnected
    - Pay-for-use licensing
- Infrastructure software:
  - Resource accounting, VM awareness
- Hardware systems:
  - Containers, energy proportionality

Is it New?

Cloud vs. Grid

- Grid comes from academia.
- Cloud comes from enterprise.

Similarities:

- Distributed computing.
- Large scale clusters.
- Commodity hardware
- Heterogeneous cluster.

Differences:

- Cloud: Elasticity and pay-as-you-go (if not, it is not Cloud).
- Grid: can be more loosely coupled and geographically dispersed than a Cloud
- Grid: may use the user computer as a part of it (volunteer computing)
- Grid: may have been built for a particular purpose and then disappear

Cloud Infrastructure

Some of the slides from
http://www.cs.bu.edu/faculty/gkollios/ada11/
http://www.dblab.ece.ntua.gr/~vergoulis/documents/cloud%20and%20mr_v2.pdf

Cloud Computing Infrastructure

- Computation model: MapReduce*
- Storage model: HDFS*
- Other computation models: HPC/Grid Computing
- Network structure

*Some material adapted from slides by Jimmy Lin, Christophe Bisciglia, Aaron Kimball, & Sierra Michels-Slettvet, Google Distributed Computing Seminar, 2007 (licensed under Creation Commons Attribution 3.0 License)

Cloud Computing Computation Models

- Finding the right level of abstraction
  - von Neumann architecture vs cloud environment
- Hide system-level details from the developers
  - No more race conditions, lock contention, etc.
- Separating the *what* from *how*
  - Developer specifies the computation that needs to be performed
  - Execution framework ("runtime") handles actual execution

Cloud Computing Infrastructure

- Computation model: MapReduce*
- Storage model: HDFS*
- Other computation models: HPC/Grid Computing
- Network structure

*Some material adapted from slides by Jimmy Lin, Christophe Bisciglia, Aaron Kimball, & Sierra Michels-Slettvet, Google Distributed Computing Seminar, 2007 (licensed under Creation Commons Attribution 3.0 License)

MapReduce

Jeffrey Dean, Sanjay Ghemawat: MapReduce: simplified data processing on large clusters, OSDI 2004

"Big Ideas"

- Scale "out", not "up"
  - Limits of SMP and large shared-memory machines
- Idempotent operations
  - Simplifies redo in the presence of failures
- Move processing to the data
  - Cluster has limited bandwidth
- Process data sequentially, avoid random access
  - Seeks are expensive, disk throughput is reasonable
- Seamless scalability for ordinary programmers
  - From the mythical man-month ("adding manpower to a late software project makes it later") to the tradable machine-hour

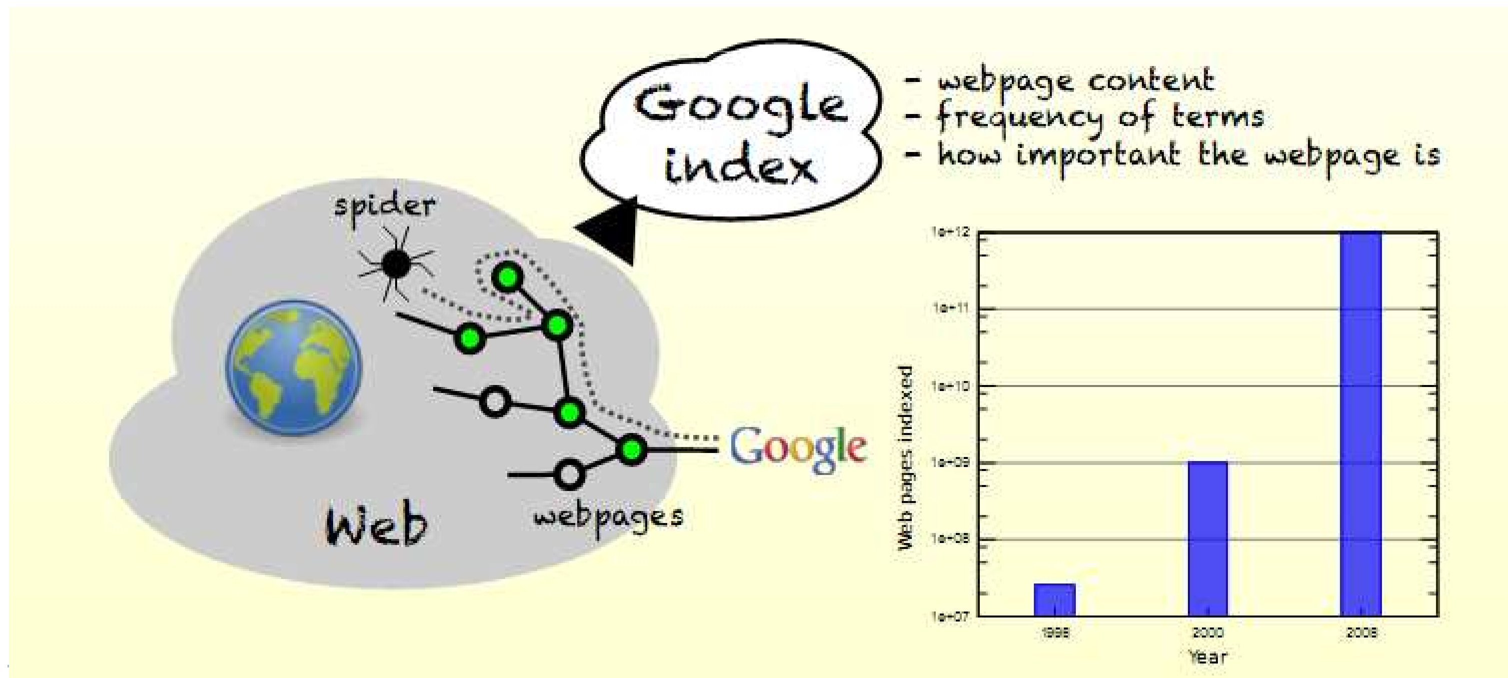
Map Reduce

A programming paradigm that comes with a framework to provide to the programmers an easy way for parallel and distributed computing.

- Designed by Google (published in 2004)
- Designed to scale well on large clusters --> Perfect for Cloud Computing.
- Input & output data stored in a distributed file system.
- Fault tolerance, status & monitoring tools
- It is attractive because it provides a simple model.
- More than 10,000 distinct MapReduce programs have been implemented in Google.
    - Graph processing, text processing, machine learning, statistical machine translation etc
- Open source implementation (Hadoop)

Why Google introduced MapReduce?

Information Retrieval (IR) problem: Indexing the Web
Because of the explosion of data Google decided:
- To use large clusters of commodity hardware (i.e., Cloud)
- To introduce a framework that makes easy programming on Cloud (i.e., MapReduce)



Typical Large-Data Problem

- Iterate over a large number of records
- Map tract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate re Reduce
- Generate final output

Key idea: provide a functional abstraction for these two operations – MapReduce

MapReduce

- Programmers specify two functions:

  map (k, v) → <k’, v’>*

  Takes an input pair and produces an intermediate (key, value) pair

  reduce (k’, v’) → <k’, v’>*

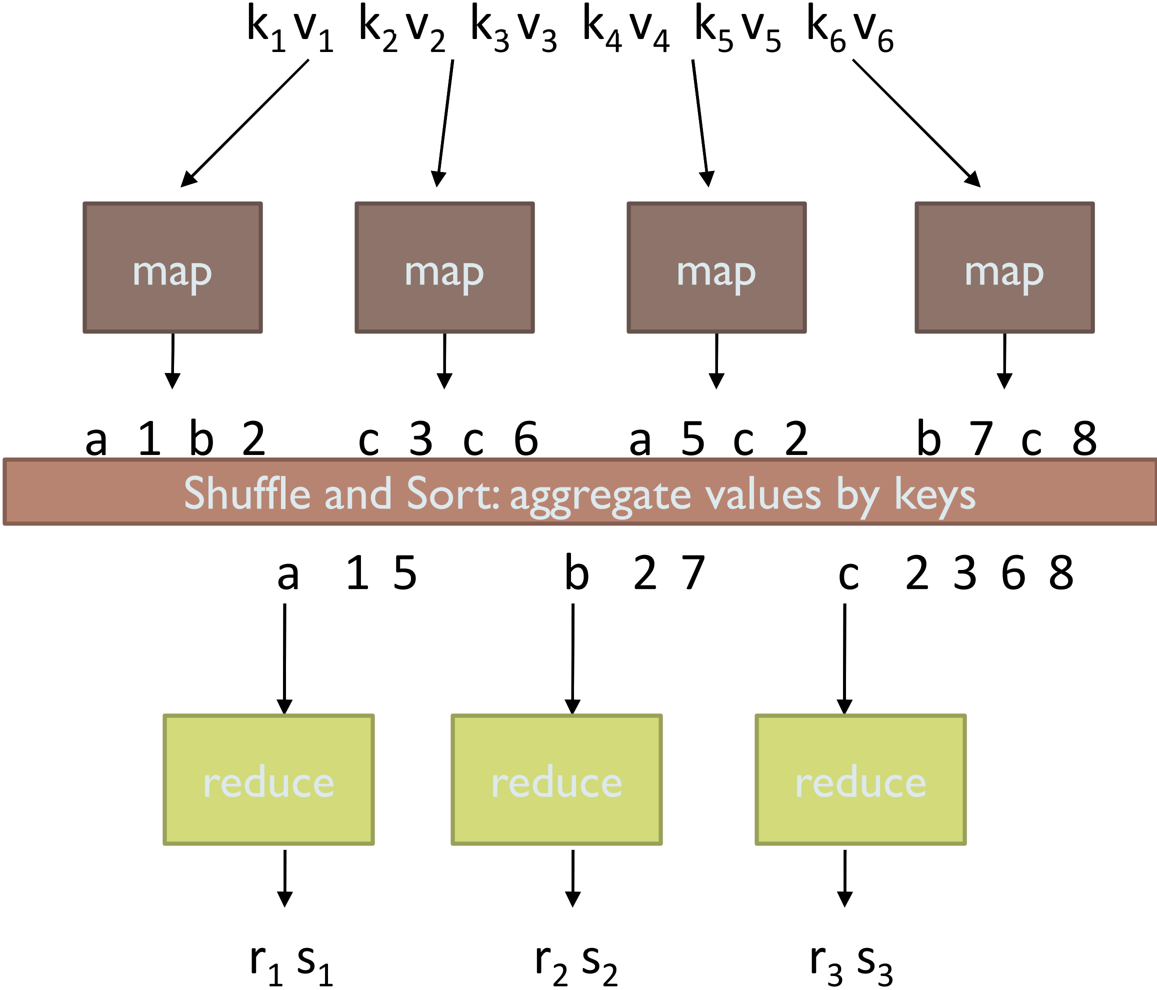
  - *All values with the same key are sent to the same reducer*

  Each reducer accepts as input all values associated with the same intermediate key and merges them together to produce a possibly smaller set

  The intermediate values are supplied to reduce function via an iterator

- The execution framework handles everything else…

MapReduce



The WordCount MapReduce example

```
map(String input_key, String input_value):
for each word w in input_value:
EmitIntermediate(w, "1");
```

- input_key: document name
- input_value: document contents

```
reduce(String output_key, Iterator intermediate_values):
int result = 0;
for each v in intermediate_values:
result += ParseInt(v);
Emit(AsString(result));
```

- output_key: a word
- output_values: a list of counts

MapReduce Implementation

The *MapReduce library* in the user program
- partitions the input files into M *splits*
- starts up copies of the program on a cluster of machines.

1 master + workers assigned work by the master (M map and R reduce tasks)

A **mapper worker**
- o reads the contents of the corresponding input split,
- o parses key/value pairs out of the input data and
- o passes each pair to the user-defined Map function.

The intermediate key/value pairs produced by the Map function

o Buffered in memory.

o Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function.

o *Locations of buffered pairs on the local disk passed back to the master*, responsible for forwarding them to the reduce workers.

o A reduce worker *uses remote procedure calls to read the buffered data* from the local disks of the map workers.

MapReduce Implementation

When a **reduce worker** has read all intermediate data,

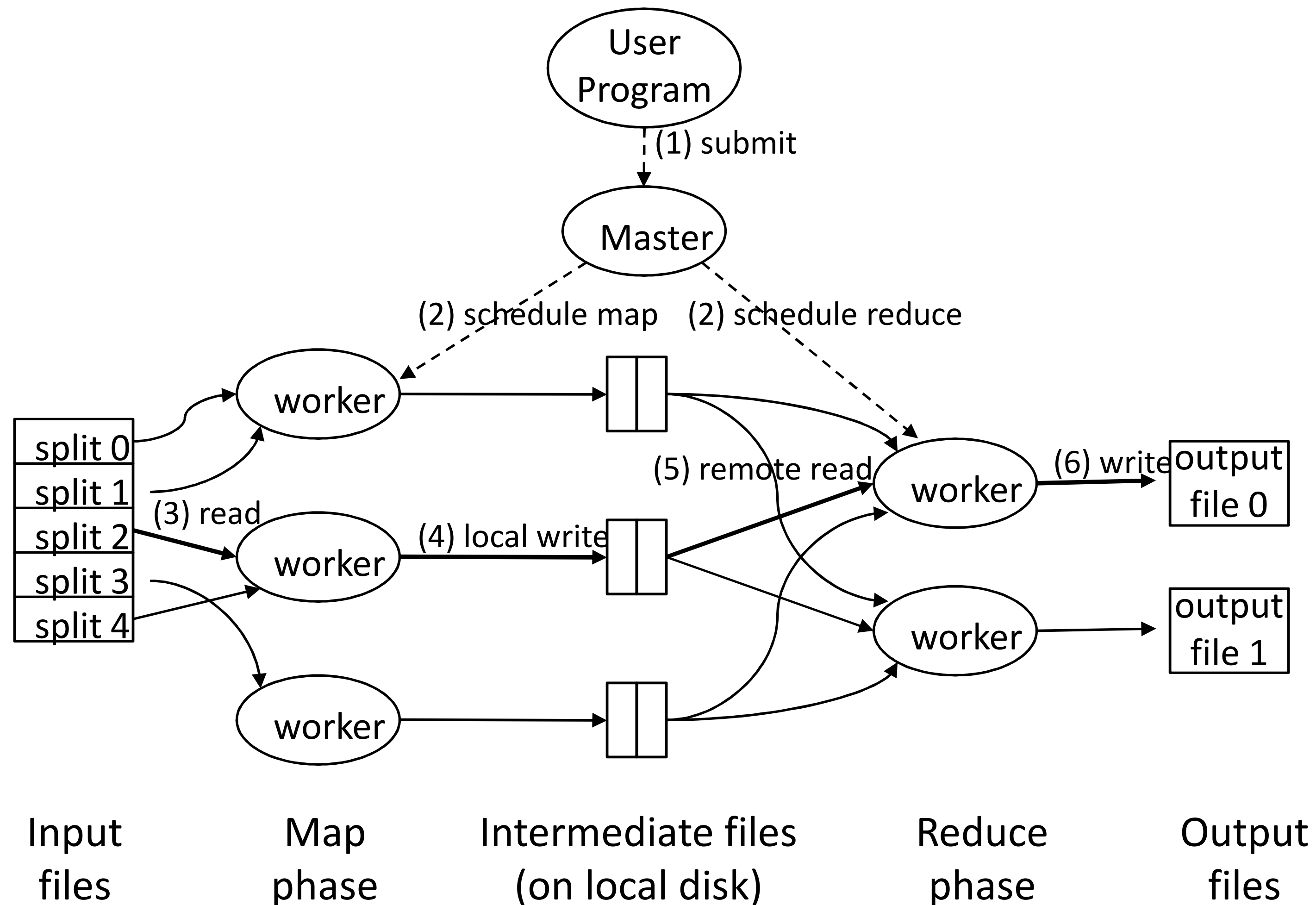
- sorts it by the intermediate keys (sorting needed because typically many different keys map to the same reduce task) -- external sort may be needed
- iterates over the sorted intermediate data
    - for each unique intermediate key encountered,
      passes the key and the corresponding set of intermediate values to the user's Reduce function.

Output of the Reduce function is appended to a final output file for this reduce partition.

When all map and reduce tasks complete, the master wakes up the user program.

Output available in the R output files
User may
- pass these files as input to another MapReduce call, or
- use them from another distributed application.

MapReduce Overall Architecture



Adapted from (Dean and Ghemawat, OSDI 2004)

The MapReduce paradigm

- MapReduce program input: A set of files stored in the DFS.
- MapReduce program output: A set of files stored in the DFS.
- Each program is divided into two subprograms:
  - Map subprogram.
  - Reduce subprogram.
- MapReduce framework is responsible for executing a large number of instances of each subprogram on a computer cluster.
- Map subprogram instances:
  - Process a part of the input.
  - Produce (*key*, *value*) pairs.
- Reduce subprogram instances:
  - Process (*key*, *value*) pairs having particular keys.
  - Produce output.
- The execution of multiple map instances reminds of the map function of functional programming.
- The execution of multiple reduce instances reminds of the fold function of functional programming.

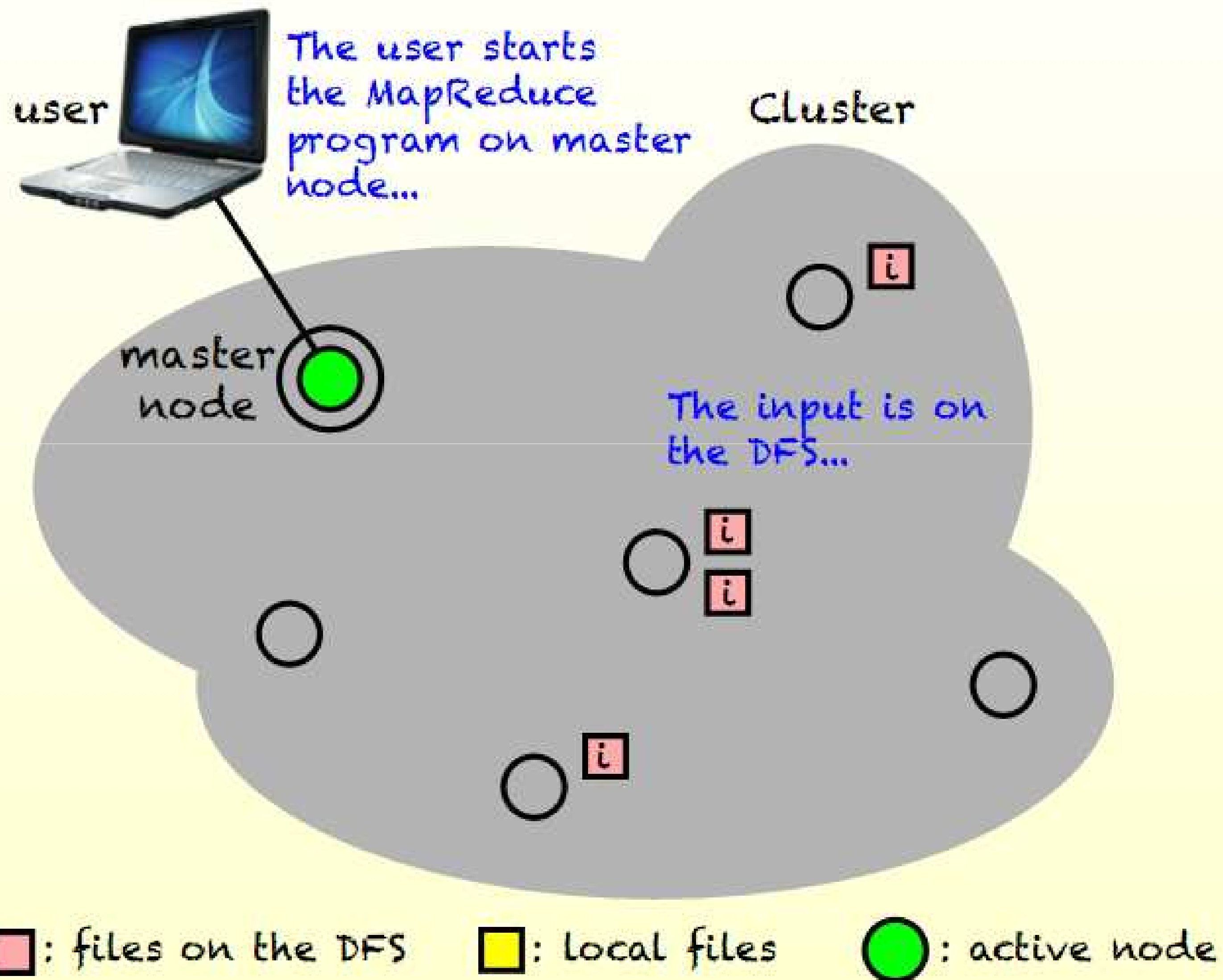
The MapReduce paradigm: The map subprogram

- Multiple map instances run on different nodes of a cluster.
- Each map instance...
  - ... gets disjoint portion of the input from the DFS (files, records of files, etc.)
  - ... processes its input to produce (*key*, *value*) pairs
  - ... uses a split function to partition these pairs into R disjoint buckets according to their key
  - ... writes any completed bucket to the disk (locally)
- The input is given to the instances by the MapReduce Scheduler.
- The split function and R are given by the user.
- The output of each map instance consists of R files on the disk.
- The map subprogram is an arbitrary computation in a general-purpose language.

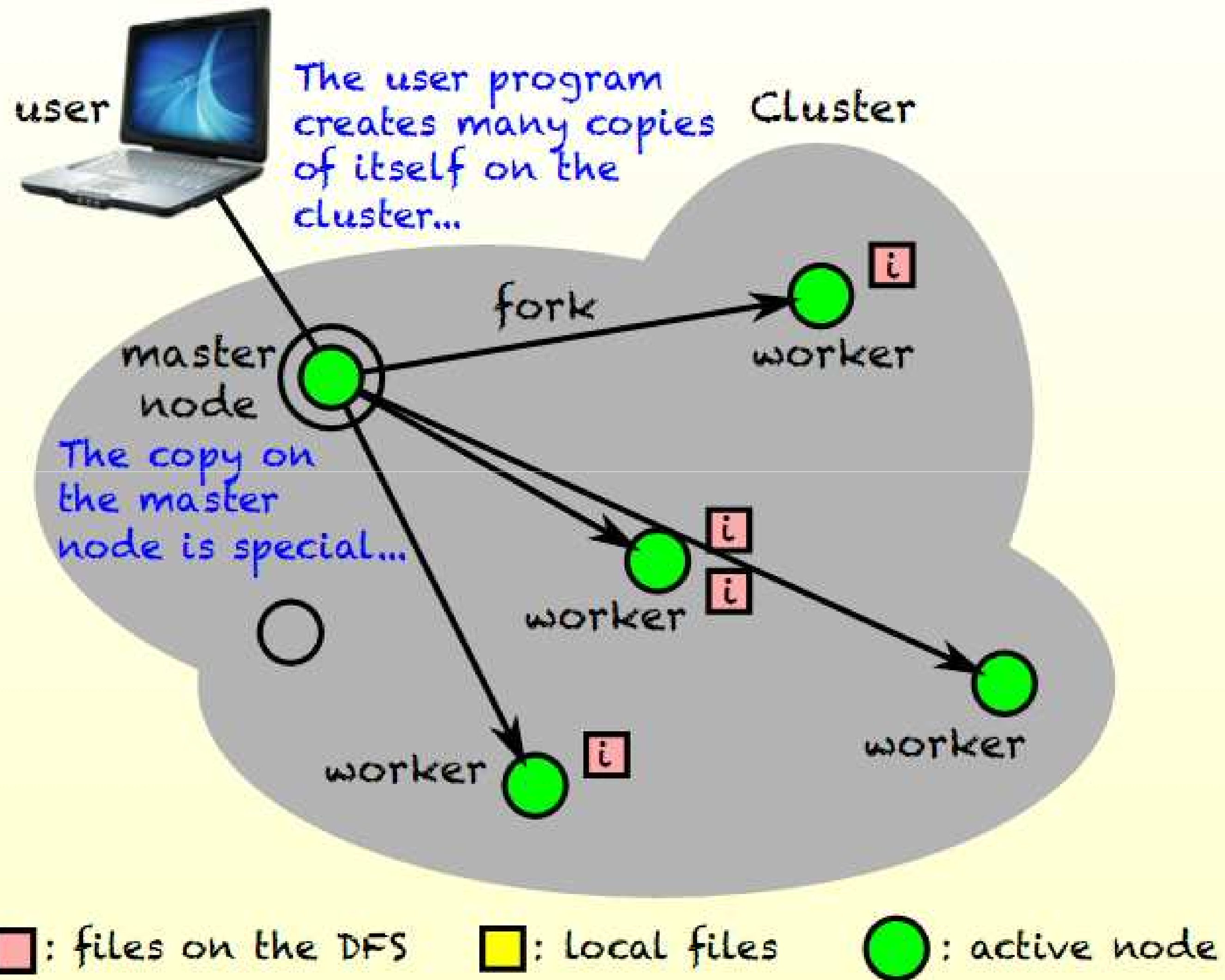
The MapReduce paradigm: The reduce subprogram

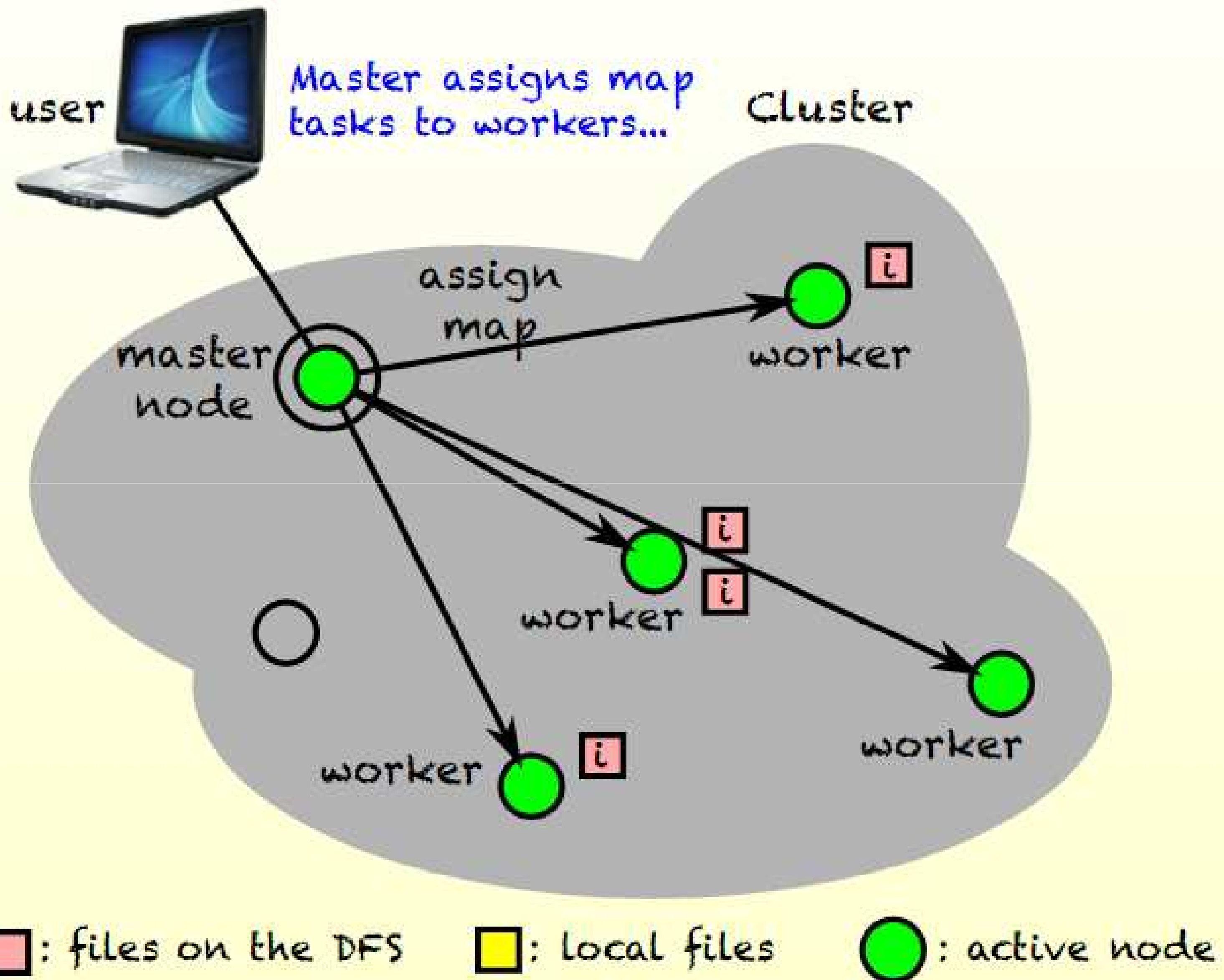
- Multiple reduce instances run on different nodes of a cluster.
- Each reduce instance...
  - ... gets those (*key*, *value*) pairs which have particular keys
  - ... processes its input to produce the output
  - ... writes the output to the GFS (not locally!)
- The fetching of (*key*, *value*) pairs is done by remote reads (use of pull instead of push).
- The output of each reduce instance is part of the output (i.e., the output is distributed in many files).
- The map subprogram is an arbitrary computation in a general-purpose language.

The Overall Flow of a MapReduce Program

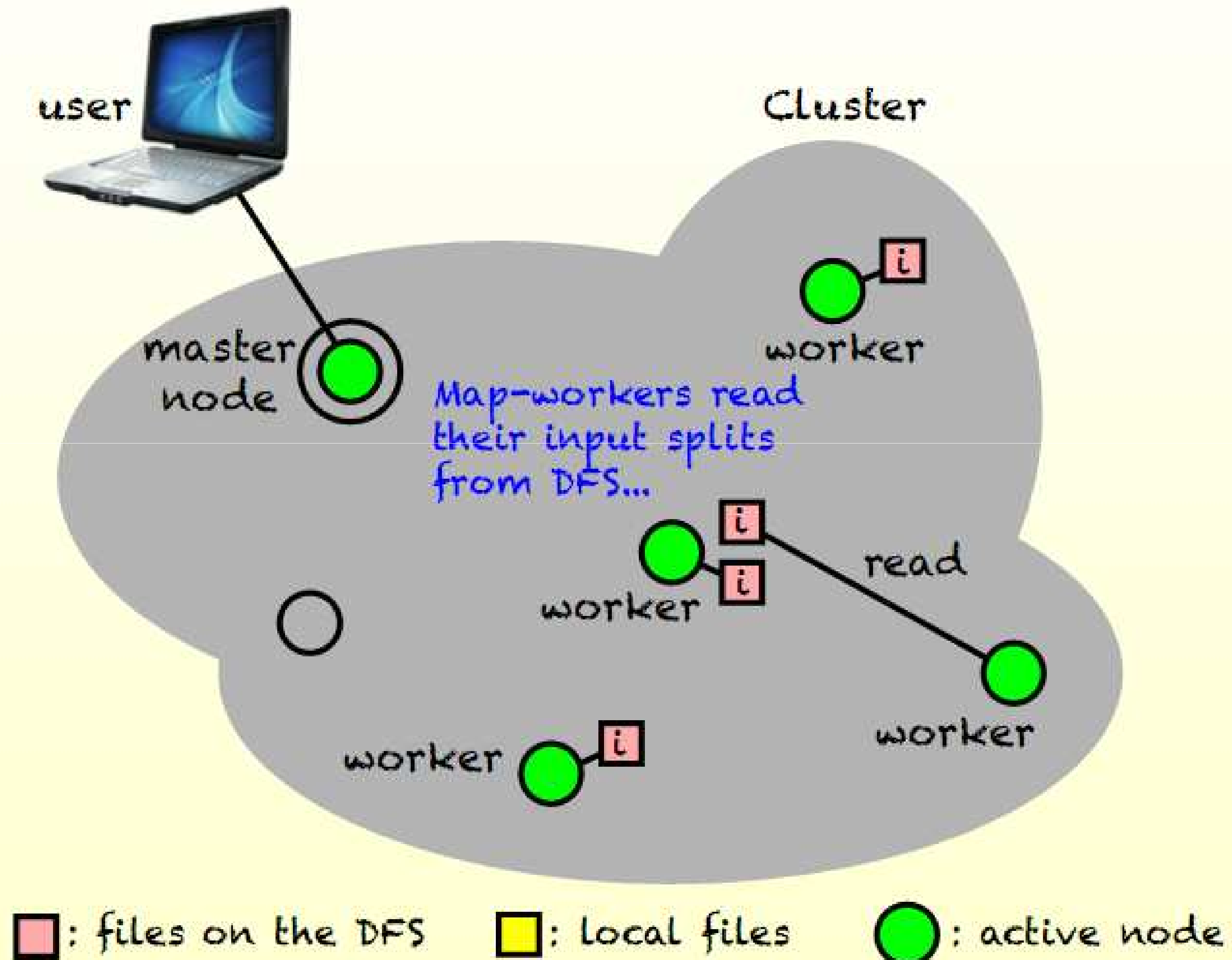


The Overall Flow of a MapReduce Program

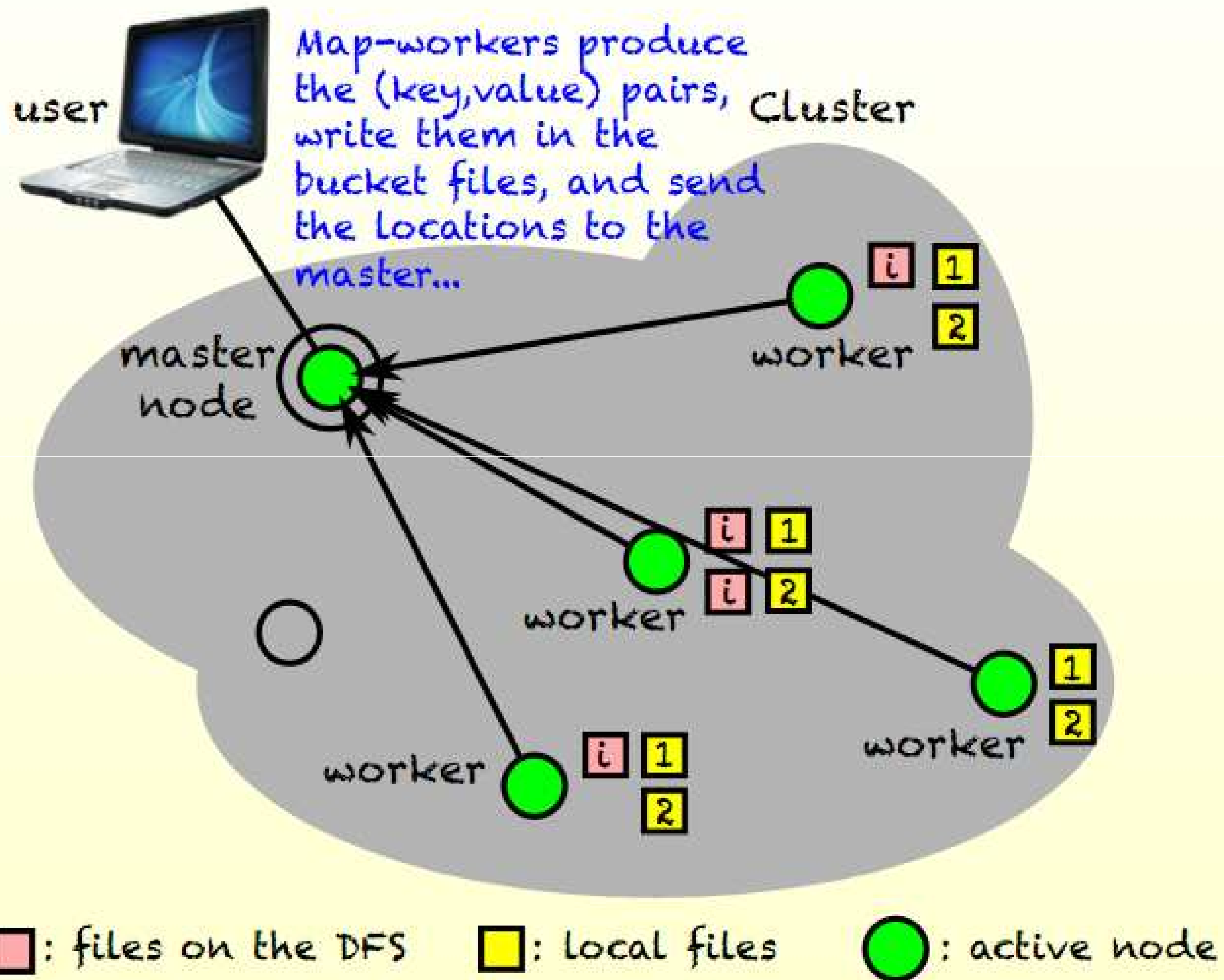



user
Master assigns map
tasks to workers...
Cluster
assign
map
master
node
worker
i
worker
i
i
worker
worker
i
: files on the DFS
: local files
: active node

The Overall Flow of a MapReduce Program



The Overall Flow of a MapReduce Program



The Overall Flow of a MapReduce Program

user

Master assigns reduce tasks to workers and inform them about the locations of the bucket files...

Cluster

master node

assign reduce

worker

worker

worker

worker

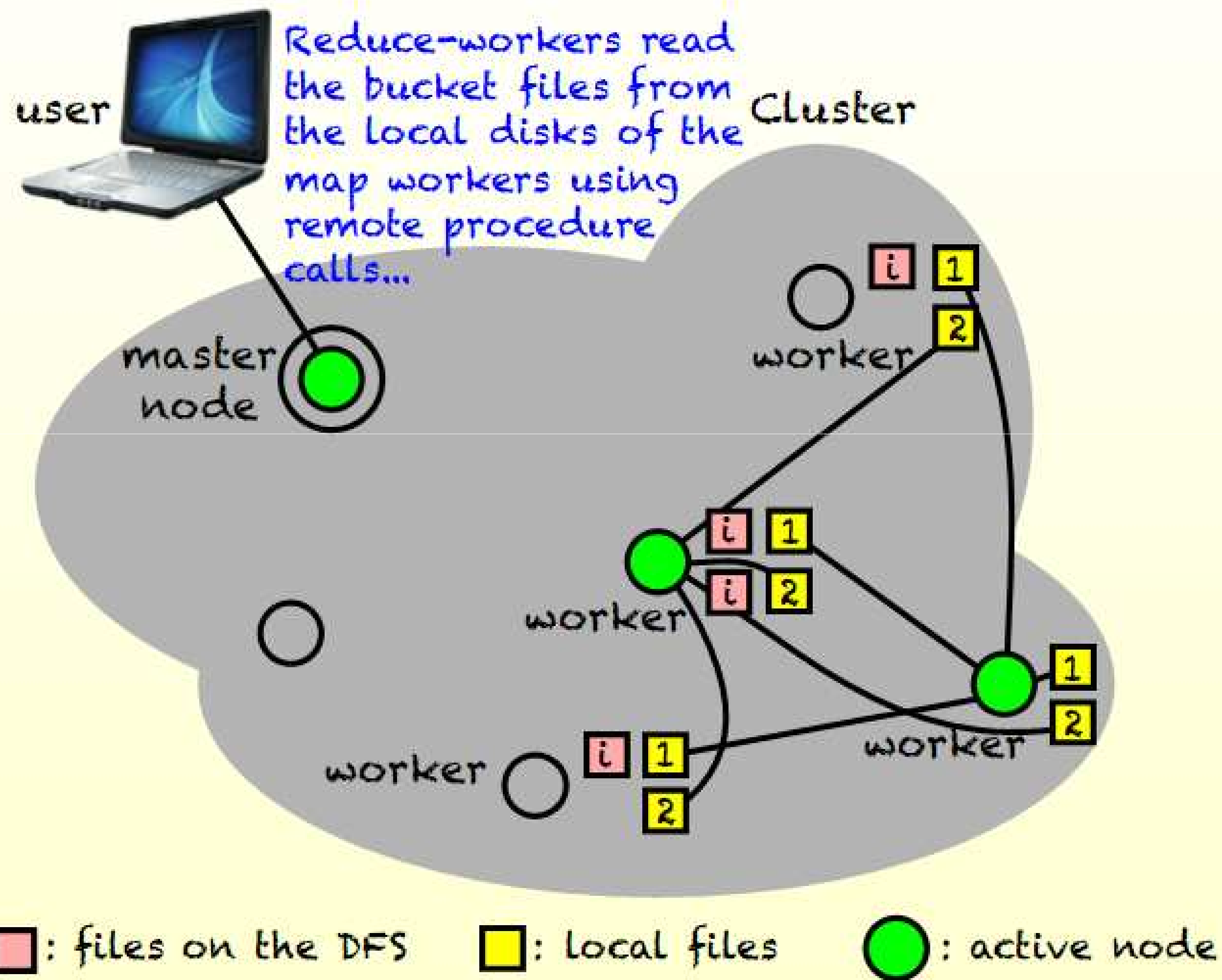
i 1 2

: files on the DFS

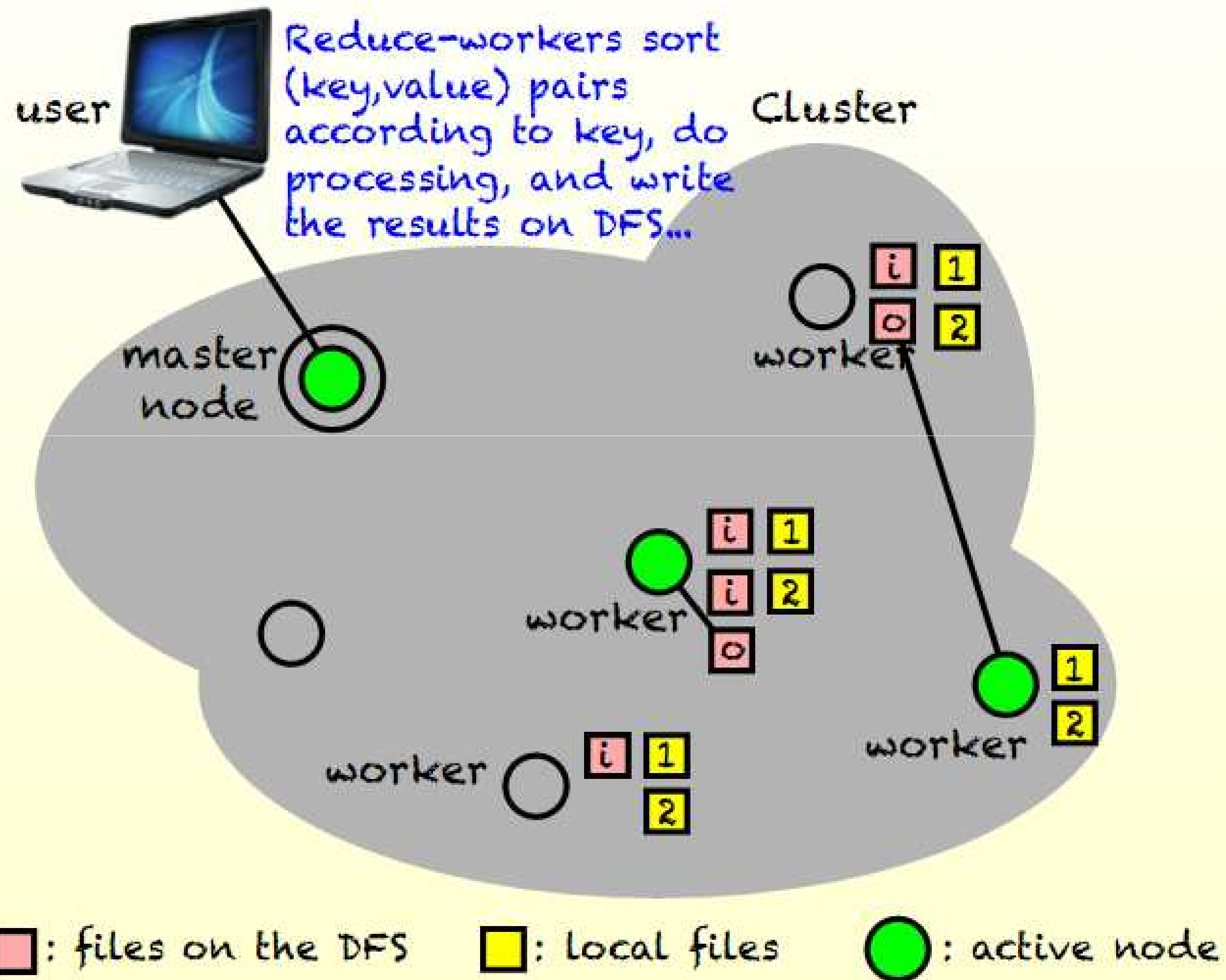
: local files

: active node

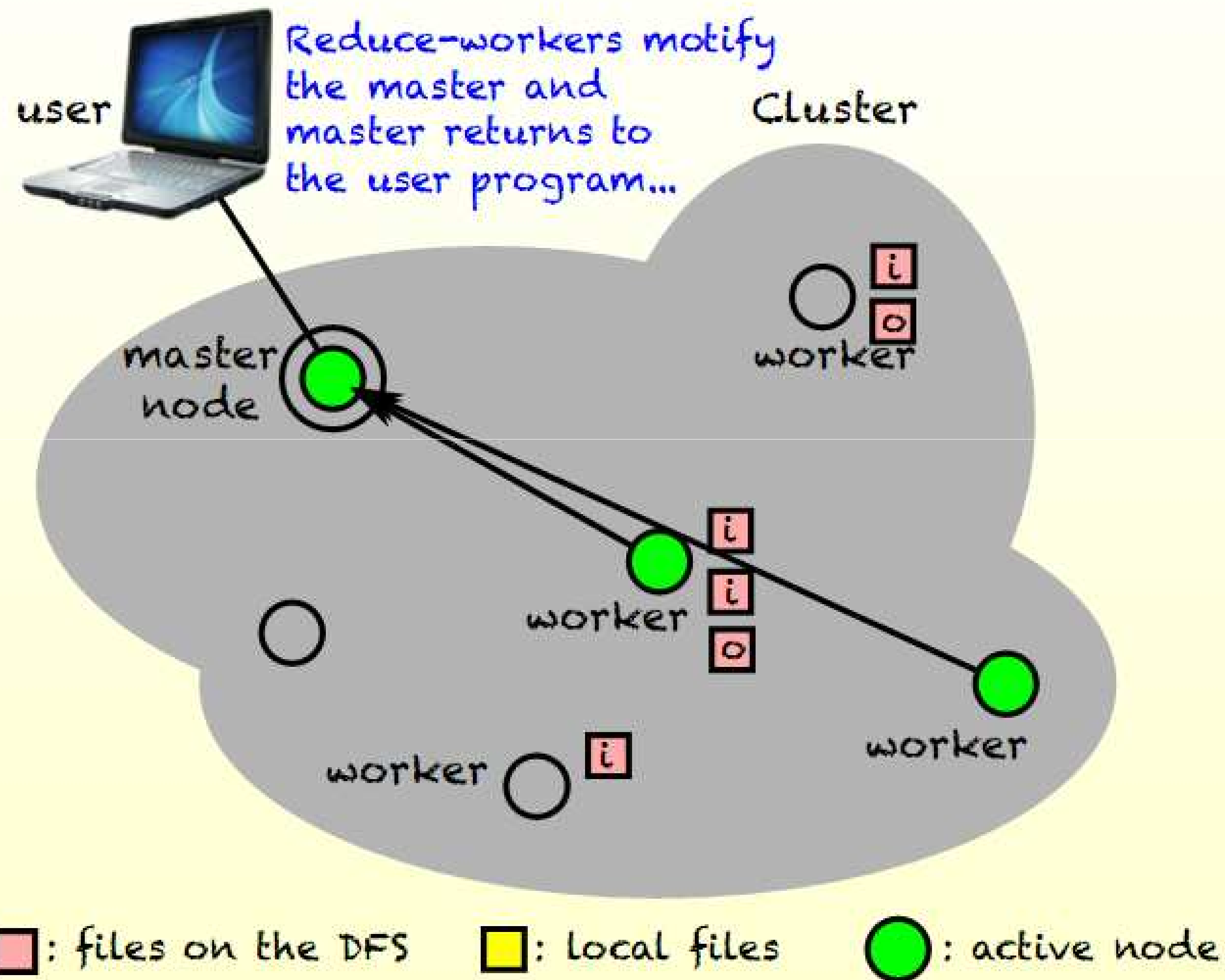
The Overall Flow of a MapReduce Program



The Overall Flow of a MapReduce Program



The Overall Flow of a MapReduce Program



MapReduce "Runtime"

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles "data distribution"
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and automatically restarts
- Handles speculative execution
  - Detects "slow" workers and re-executes work
- Everything happens on top of a distributed FS (later)

Sounds simple, but many challenges!

Some Details

Master data structure

- Idle, in-progress, completed
- Location of the intermediate file regions
- For each completed map task, the locations and sizes of the R intermediate file regions
- Pushed incrementally to workers with in-progress reduce tasks

Some Details

Backup tasks

- "Stragglers"
- when a MapReduce operation is close to completion, the master schedules backup executions for the remaining in-progress tasks

Local execution for debugging

Status information

The master runs an internal HTTP server and exports a set of status pages for human consumption

A **counter facility** to count occurrences of various events

Ordering

In a given partition, the intermediate key/value pairs are processed in increasing key order -- No enforced ordering across reducers

Barrier between map and reduce phases

But we can begin copying intermediate data earlier

Some Details

- Locality is good.
  - Master assigns to Map-workers data chunks that are stored on their disk or on disks of machines which are near to them (e.g., on the same rack).
- Fault Tolerance
  - Master detects worker failures and then orders another worker to re-execute.
  - If particular (*key*, *value*) pairs create problems, skips them during re-execution.

Completed map tasks re-executed (local memory of the failed machine) – reducers not (result in the global file system)

Failure semantics

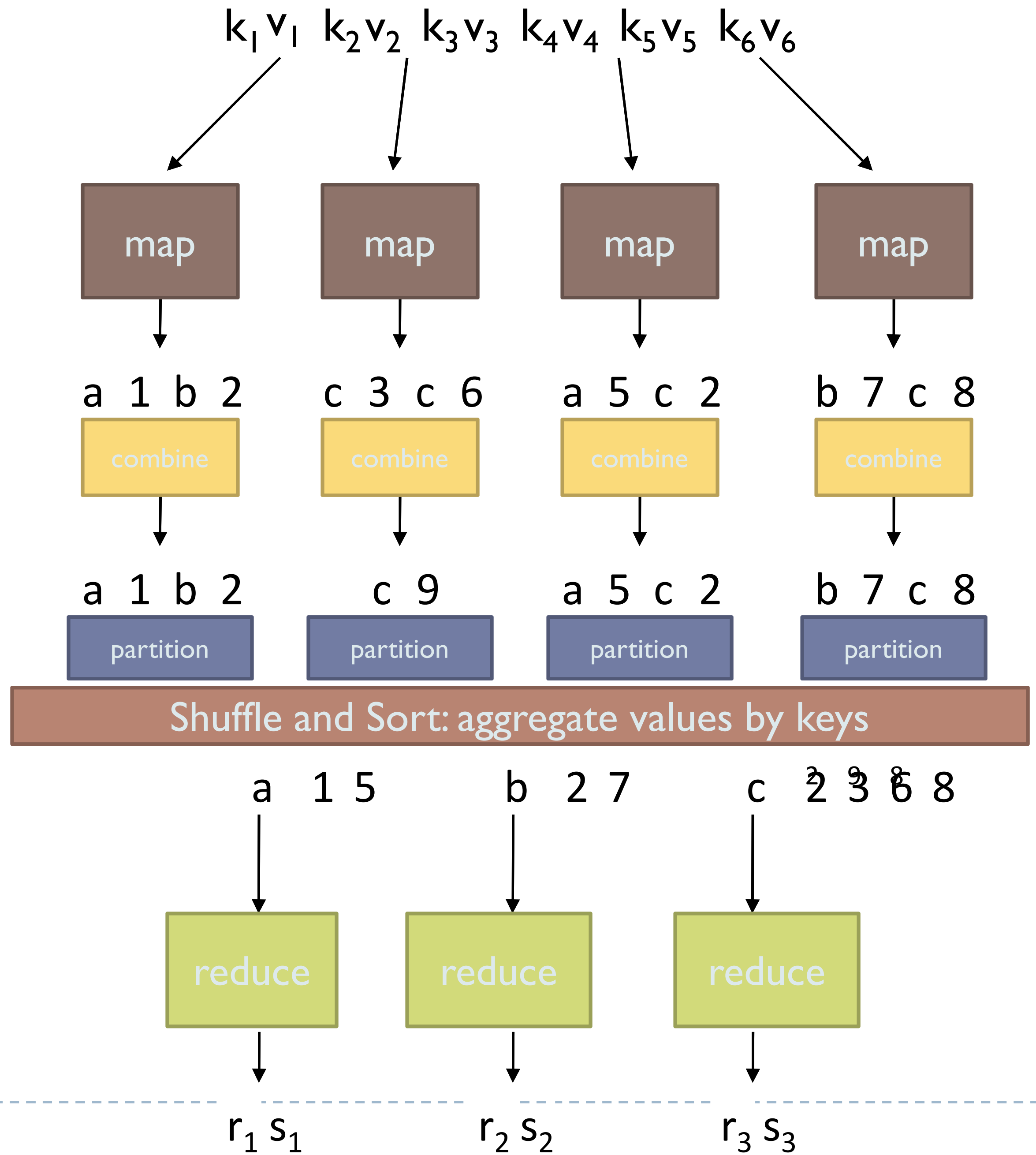
A single output file (when reducer completes atomically renames the temporary to the final)

If execution is deterministic, then the output the same as if no failure (idempotent operations)

MapReduce

- Programmers specify two functions:
  map (k, v) → <k', v'>*
  reduce (k', v') → <k', v'>*
  - All values with the same key are reduced together
- The execution framework handles everything else…
- Not quite…usually, programmers also specify:
  partition (k', number of partitions) → partition for k'
  - Often a simple hash of the key, e.g., hash(k') mod R
  - Divides up key space for parallel reduce operations

  combine (k', v') → <k', v'>*
  - Mini-reducers that run in memory after the map phase
  - Used as an optimization to reduce network traffic



$k_1 v_1$ $k_2 v_2$ $k_3 v_3$ $k_4 v_4$ $k_5 v_5$ $k_6 v_6$
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 3 6 8
reduce
reduce
reduce
$r_1 s_1$
$r_2 s_2$
$r_3 s_3$

MapReduce can refer to…

- The programming model
- The execution framework (aka "runtime")
- The specific implementation

MapReduce Implementations

- Google has a proprietary implementation in C++
  - Bindings in Java, Python
- Hadoop is an open-source implementation in Java
  - Development led by Yahoo, used in production
  - Now an Apache project
  - Rapidly expanding software ecosystem, but still lots of room for improvement
- Lots of custom research implementations
  - For GPUs, cell processors, etc.

Questions?