Cloud Computing
Review & Sample Questions

Minchen Yu
SDS@CUHK-SZ
Fall 2024

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF
DATA SCIENCE
數據科學學院

What is cloud computing?

A definition

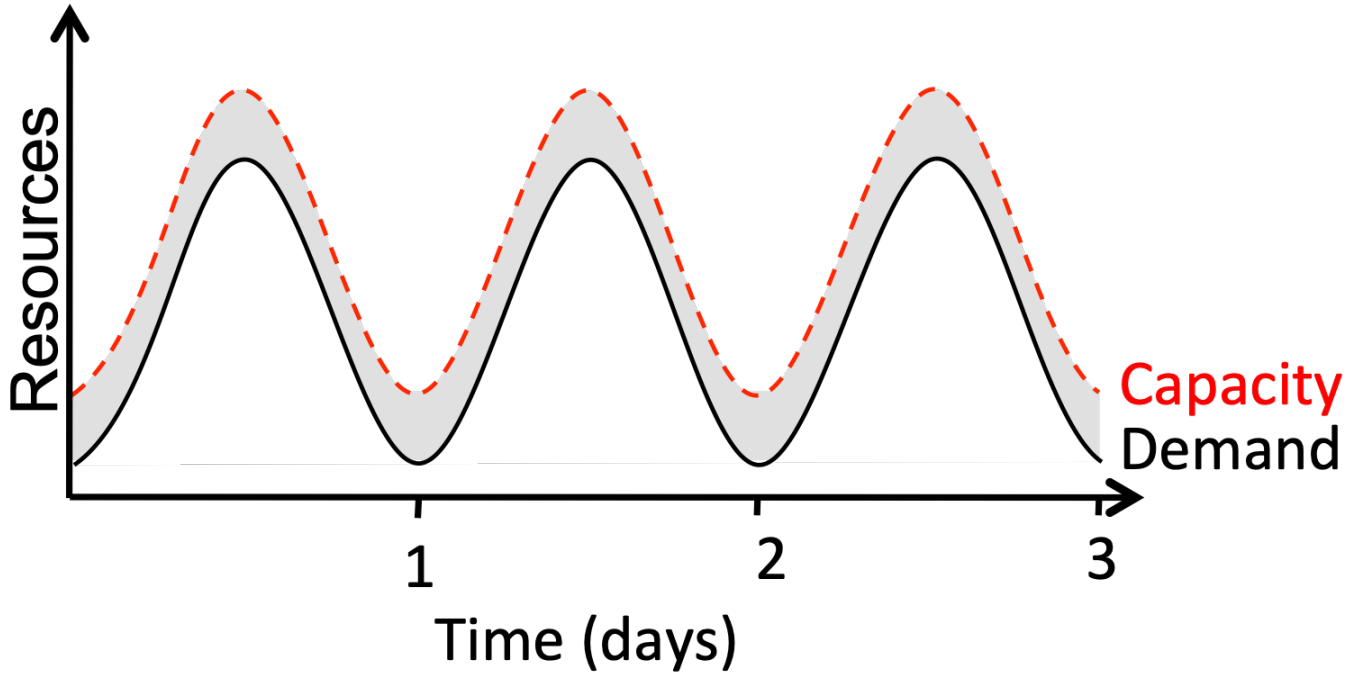
Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

National Institute of Standards and Technology (NIST), U.S. Department of Commerce

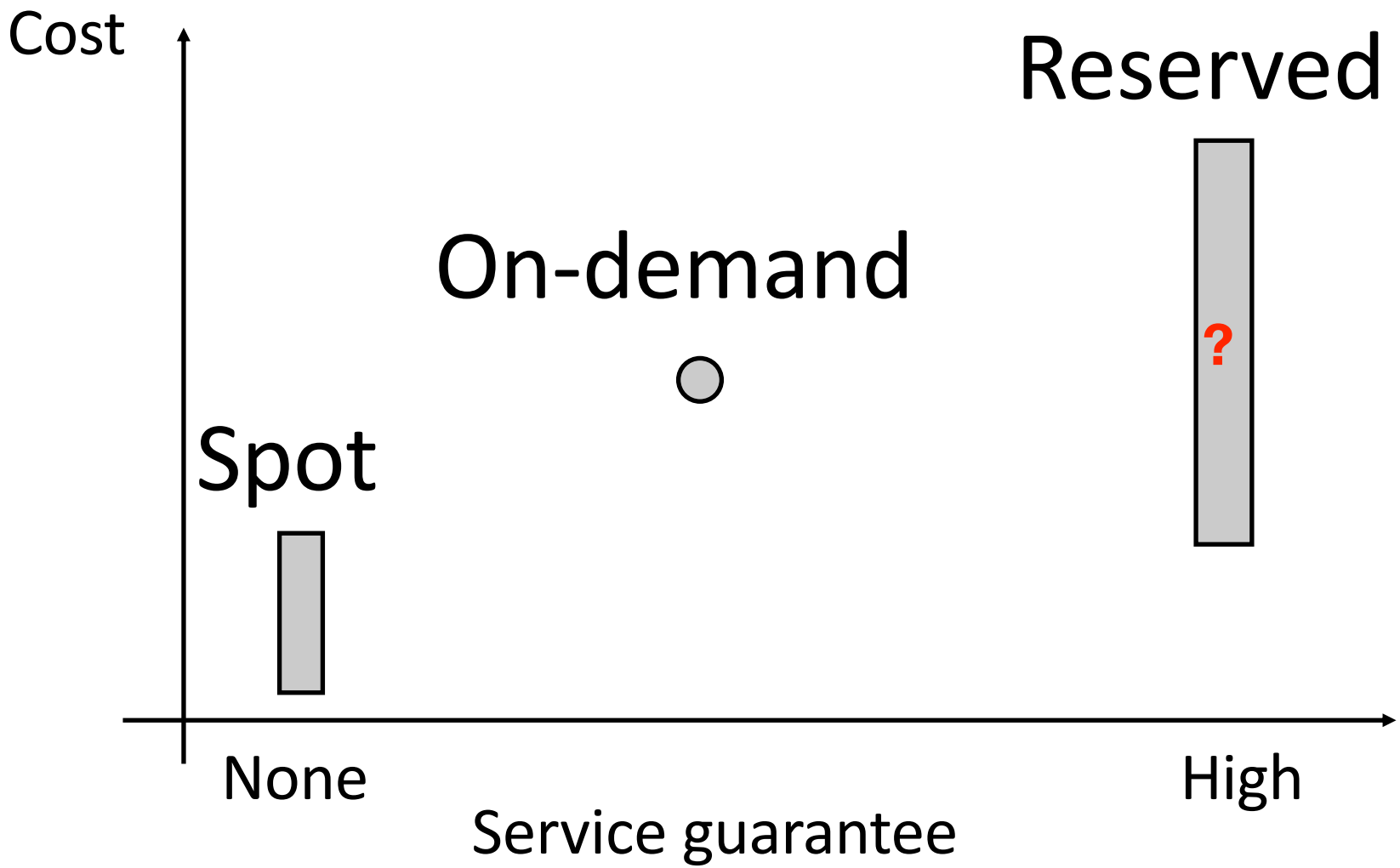
Computing as a utility: On-demand
Computing delivered over the Internet

1 instance runs 1000 h = 1000 instances run 1 h

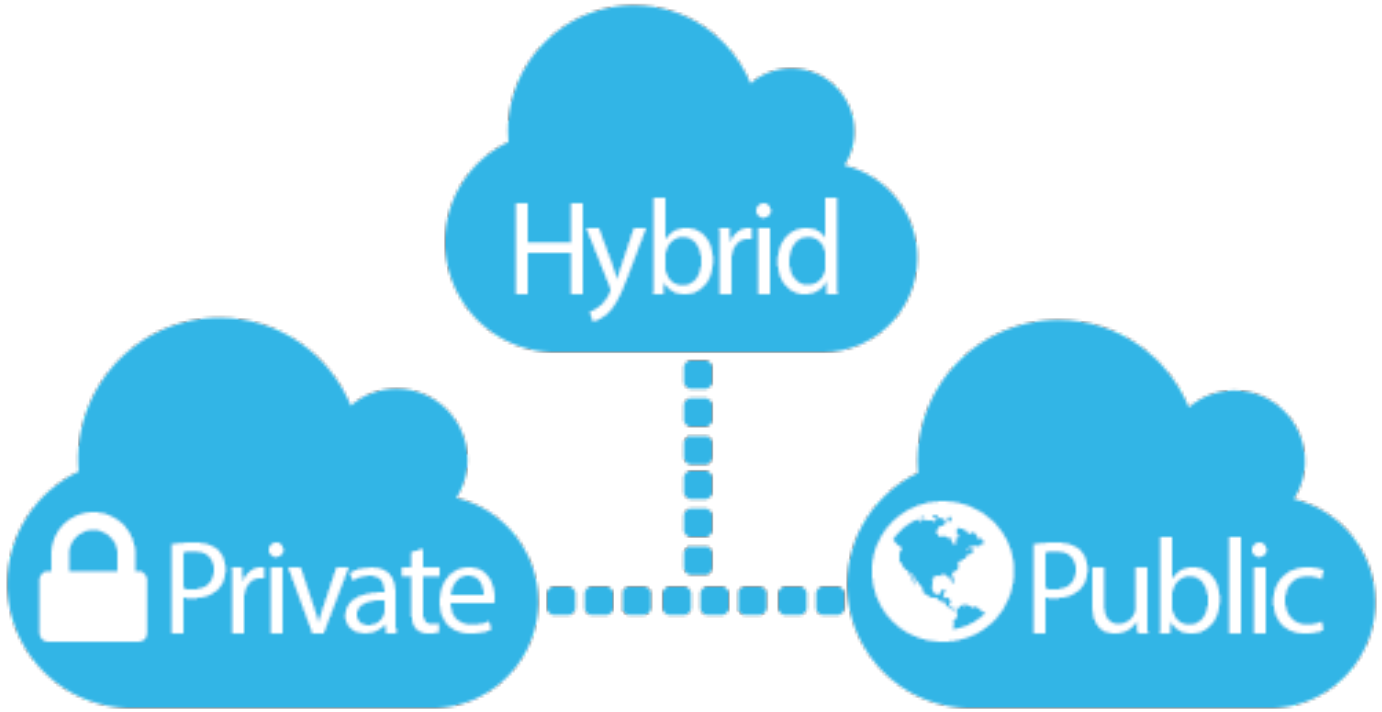
Cloud provisioning on demand



Cloud pricing

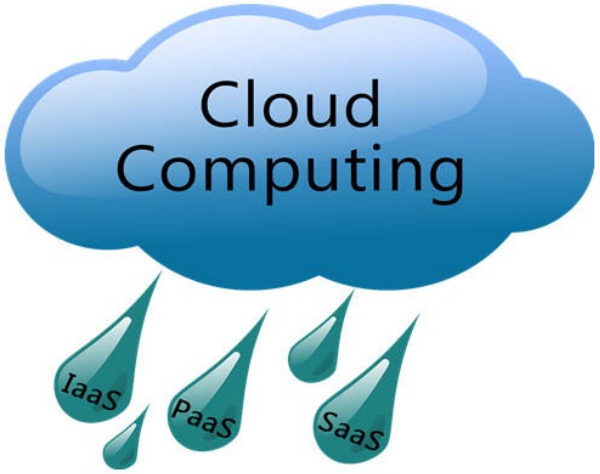


Cloud deployment models

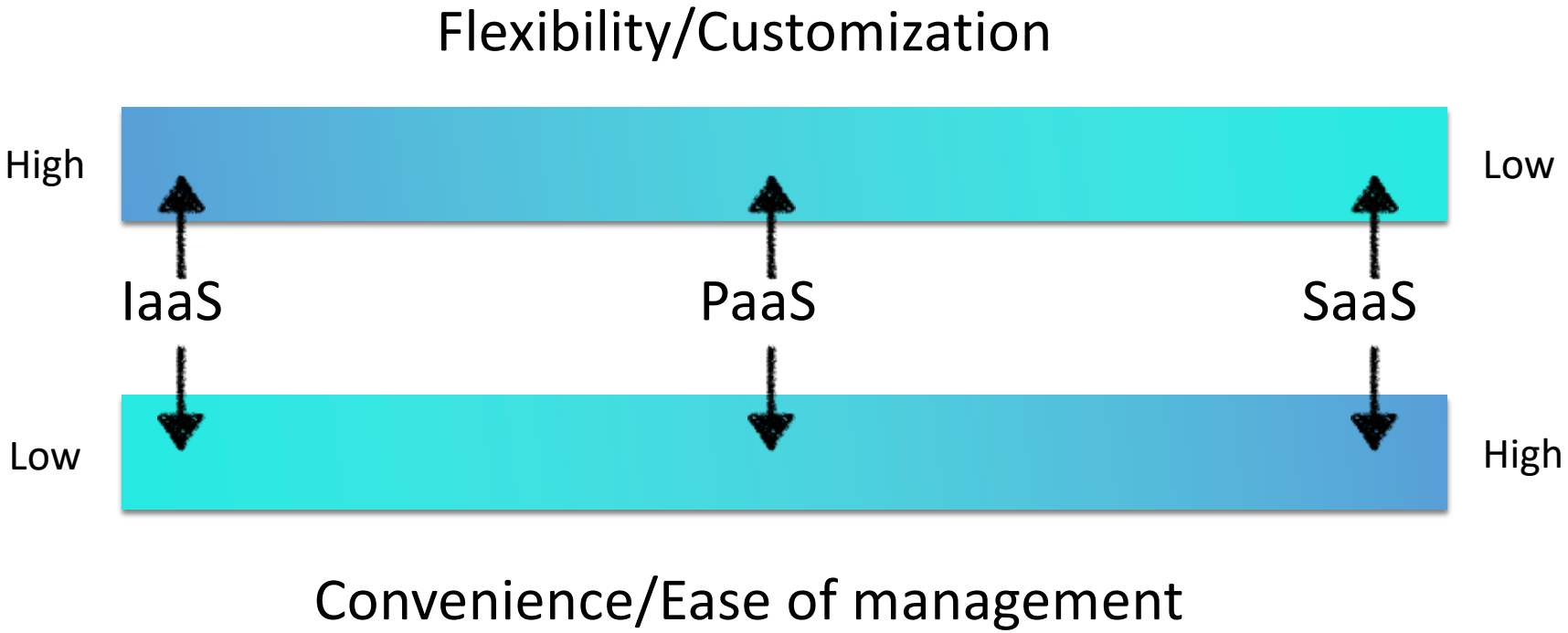


Cloud service models

- Infrastructure-as-a-Service (IaaS)
- Platform-as-a-Service (PaaS)
- Software-as-a-Service (SaaS)
- Other X-as-a-Service
  - Function-as-a-Service (FaaS)
  - Machine-Learning-as-a-Service (MLaaS)



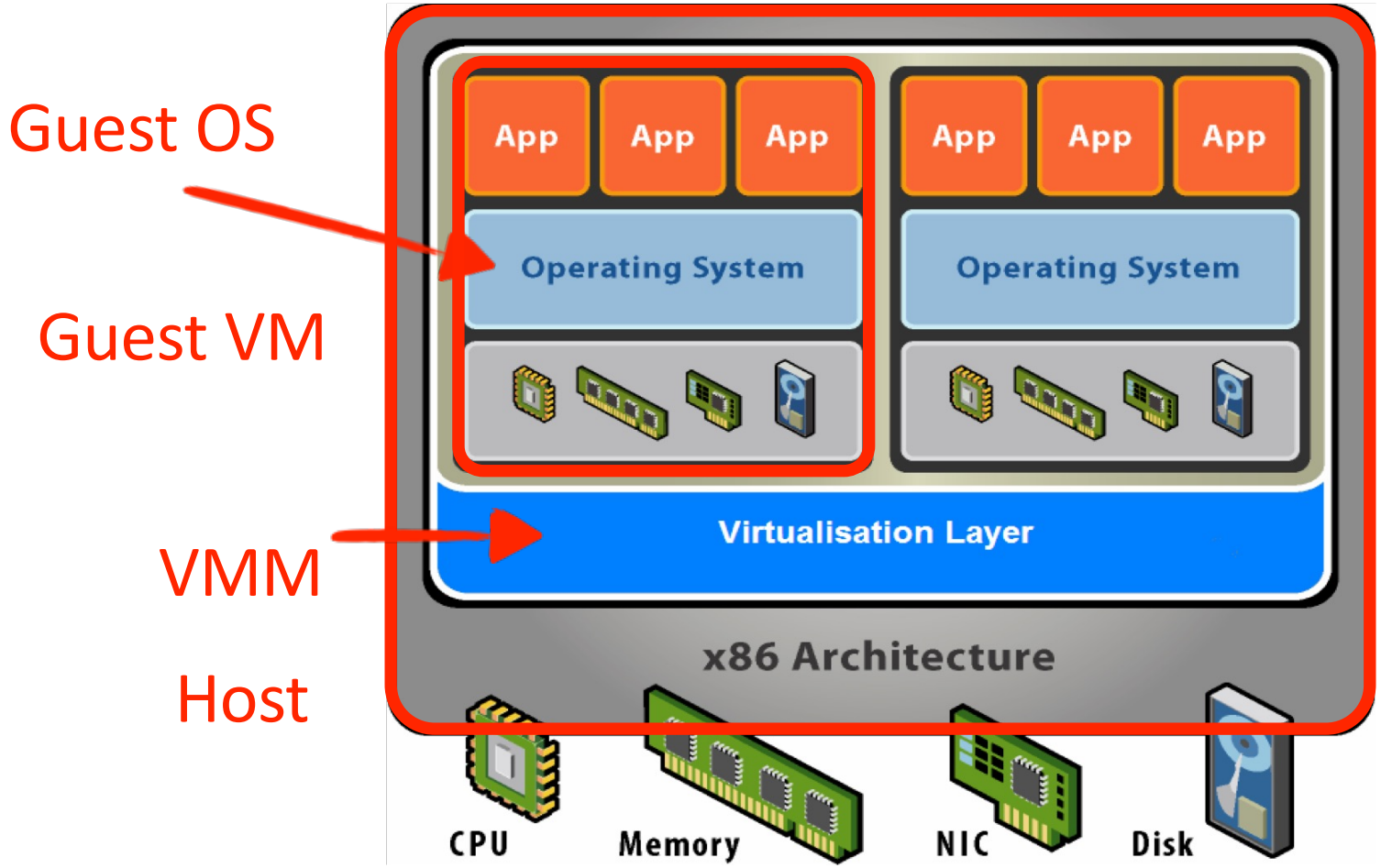
A comparison



Tradeoff between flexibility and “built-in” functionality

Virtualization

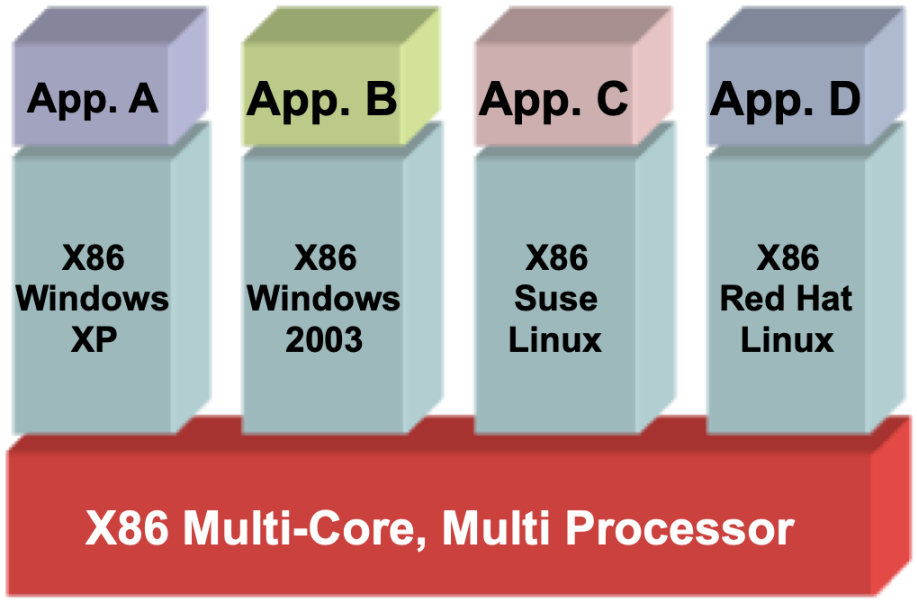
Virtualization



Why virtualization?

- Big advantage: consolidation improves utilization

Individual Util. 10% 12% 18% 20%



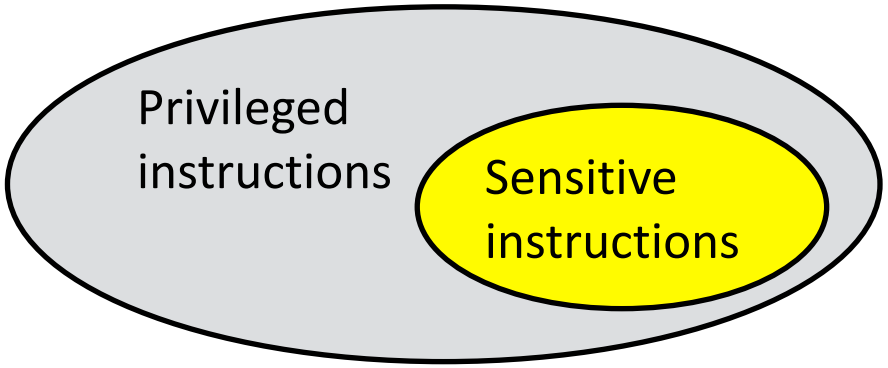
Overall Util. 60%

How? Trap and Emulate

- Trap: guest attempting a privileged instruction in user mode causes an error -> host traps to kernel mode
- Emulate: VMM gains control, analyzes the error, emulates the effect of instruction attempted by guest
  - VMM provides a virtual HW/SW interface to guest
- Return: VMM returns control to guest in user mode

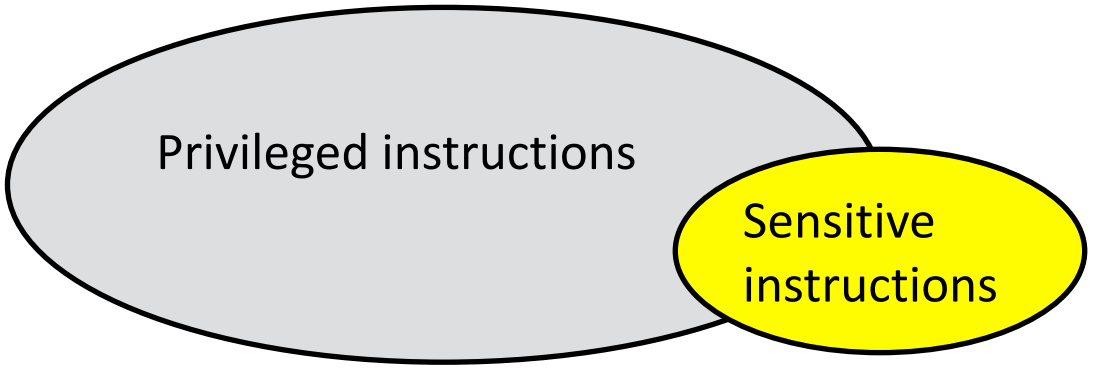
Trap and Emulate

- VMM emulates the effect of sensitive instruction on the virtual HW that it provides to guest OSs
- Popek & Goldberg (1974)
  - Privileged instructions trap when CPU is in user-mode
  - Sensitive instructions modify HW configuration or resources



Problem

- The Intel architecture did not meet Popek & Goldberg's requirement
  - some instructions are not virtualizable



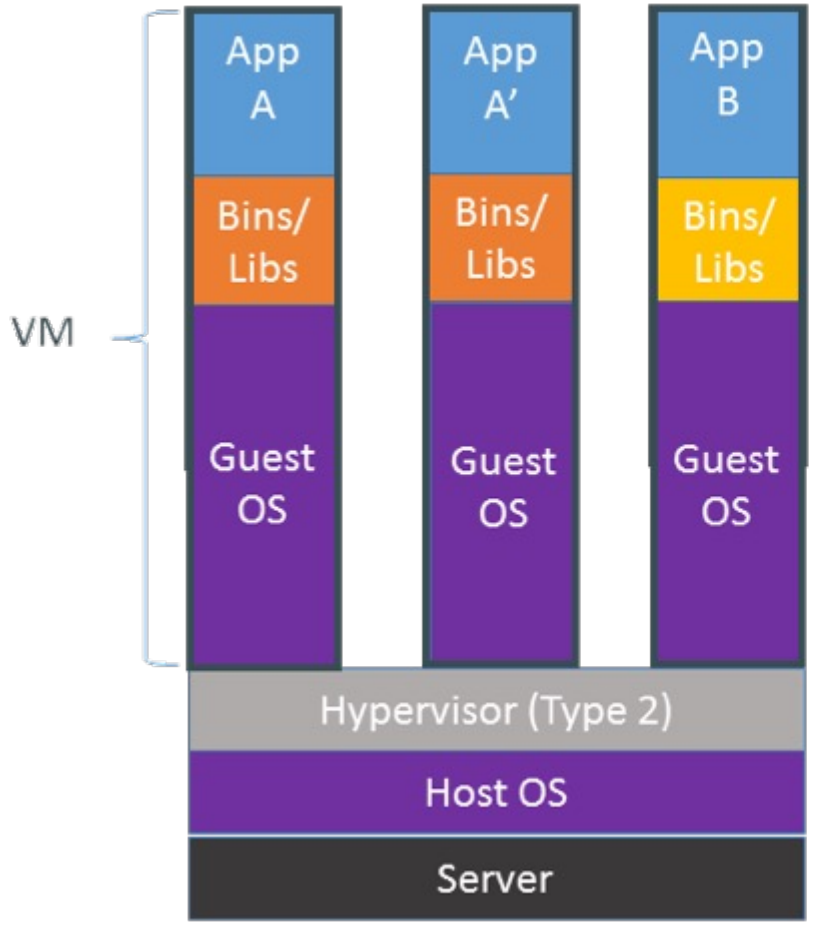
Three solutions

- Full virtualization
  - Emulate + binary translation: this is rocket science and what VMware did!
- Para-virtualization
  - modify the guest OS to avoid non-virtualizable instructions
- Hardware-assisted virtualization
  - fix the CPUs

Comparisons

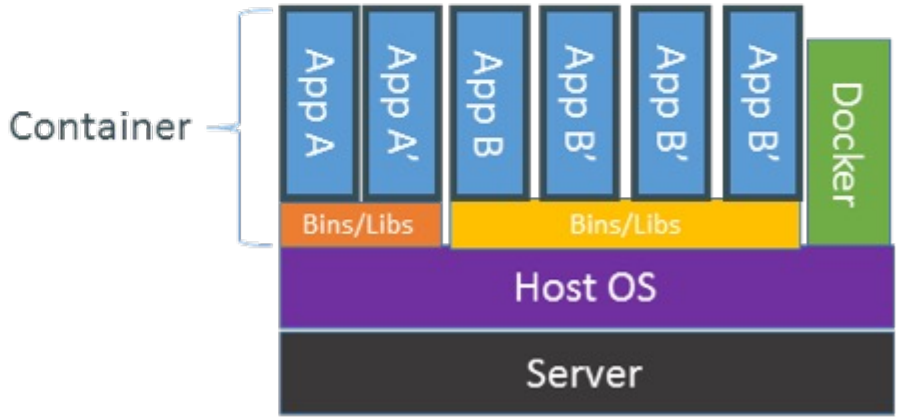
	Full	Para-	Hardware-assisted
Handling privileged insturctions	binary translation	hypercalls	non-root / root mode
Guest OS modifications	No	Yes	No
Performance	Good	Best	Better
Examples	VMware, VirtualBox	Xen	Xen, VMware, VirtualBox, KVM

VM vs. Containers



Containers are isolated, but share OS and, where appropriate, bins/libraries

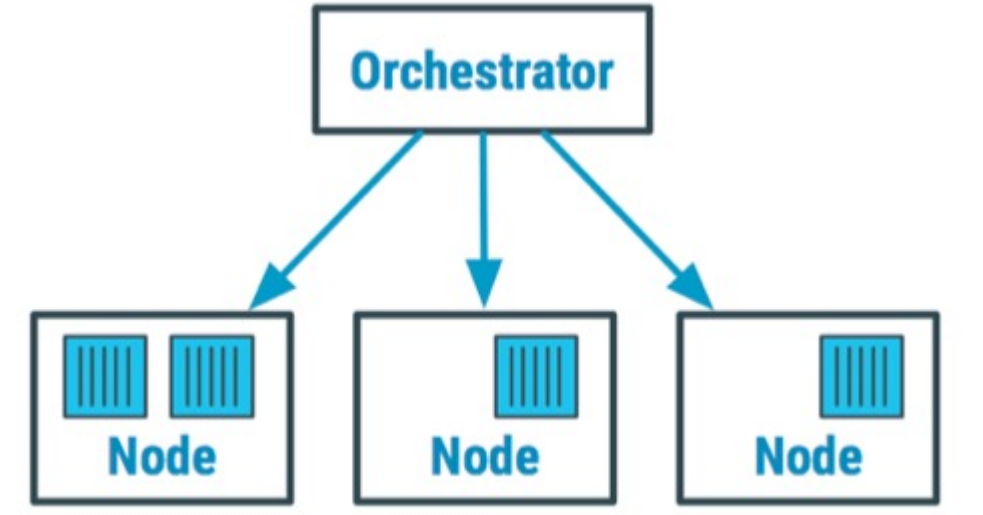
...result is significantly faster deployment, much less overhead, easier migration, faster restart



Container orchestration

Manage and organize both hosts and docker containers running on a cluster

Main Issue - **resource allocation** - where can a container be scheduled, to fulfill its requirements (CPU/RAM/disk) + how to keep track of nodes and scale



Function-as-a-Service (FaaS)

- Using cloud functions
  - Users upload function code and get an endpoint from the serverless provider
  - Functions get executed when “triggered” (e.g., HTTP request, timer, or other event sources)

> “cloud user simply writes the code and leaves all the server provisioning and administration tasks to the cloud provider”[1]

[1] Eric Jonas et al. “Cloud Programming Simplified: A Berkeley View on Serverless Computing”

Serverless vs. Serverful

Characteristic	**Serverful (IaaS)**	**Serverless (FaaS)**
Server Instance	Cloud users select	Cloud providers select
Operating System & Libraries	Cloud users select	Cloud providers select
Deployment	Cloud users handle	Cloud providers handle
Fault Tolerant	Cloud users handle	Cloud providers handle
Monitoring	Cloud users handle	Cloud providers handle
Scaling	Cloud users handle	Cloud providers handle
Billing	Per allocation	Per use
Charging Basis	Per second to per hour	Per millisecond

Distributed Storage Systems

Scale “out”, not “up”!

Lots of cheap, commodity PCs, each with disk and CPU

- 100s to 1000s of PCs in cluster in early days

High aggregate storage capacity

- No costly “big disk”

Spread search processing across many machines

- High I/O bandwidth, proportional to the # of machines
- Parallelize data processing

GFS/HDFS

A highly reliable storage system built atop highly unreliable hardwares

Target environment

Thousands of computers

Failures are the norm

Files are huge, but not many

Write-once, read-many

I/O bandwidth is more important than latency

Design decisions

File stores as/divided into chunks

- Large chunk size: 64/128 MB (Why?)

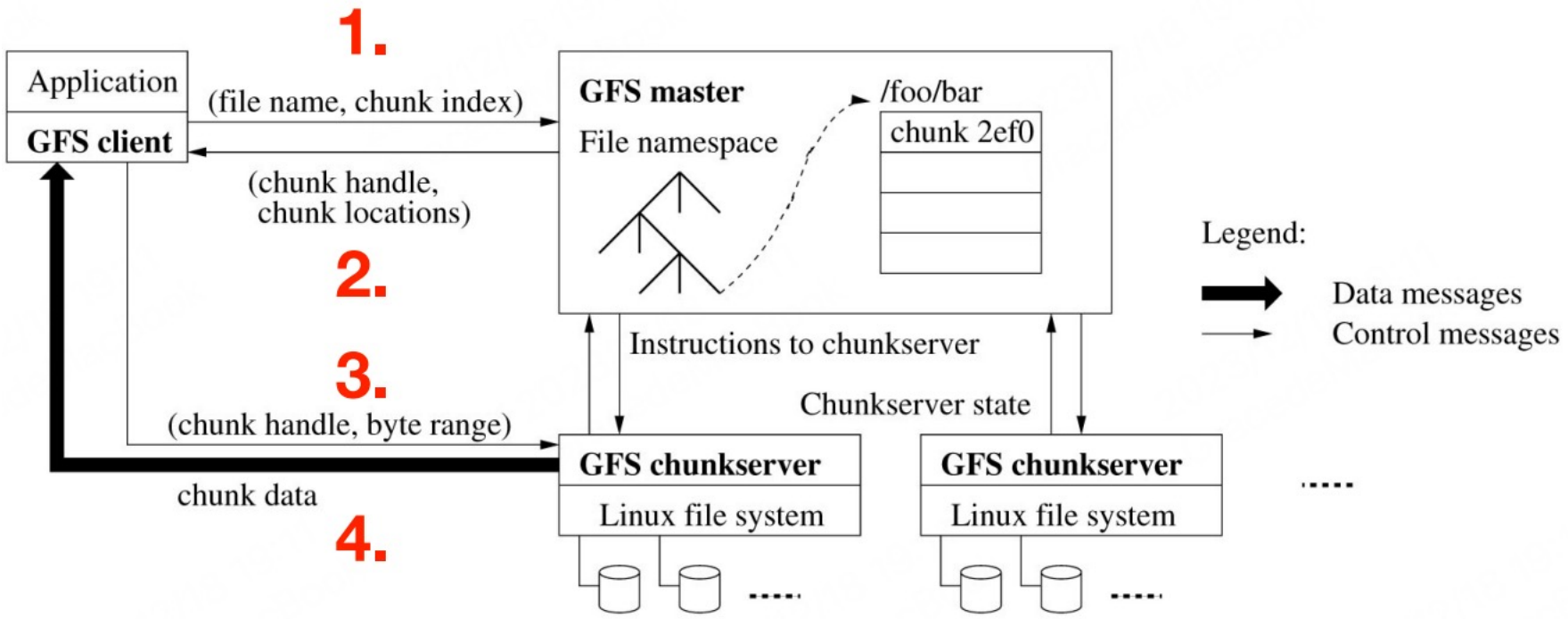
Reliability through replications

- 1 replica in the same rack
- 2+ in other, different racks

Single master to coordinate access

- keep metadata only, file data stored in chunkservers (Why?)

Architecture



Limitations

Assumes write-once, read-many workloads

Assumes a modest number of large files

Emphasizes throughput over latency

MapReduce

MapReduce: Recap

Programmers specify two functions

- **map** (k, v) → [<k2, v2>]
- **reduce** (k2, [v2]) → [<k3, v3>]
- all values with the same key are reduced together

Optionally also:

- **combine** (k, [v]) → <k, v>
- **partition** (k, # of partitions)

The execution framework handles everything else…

Importance of local aggregation

Ideal scaling characteristics

- twice the data, twice the running time
- twice the resources, half the running time

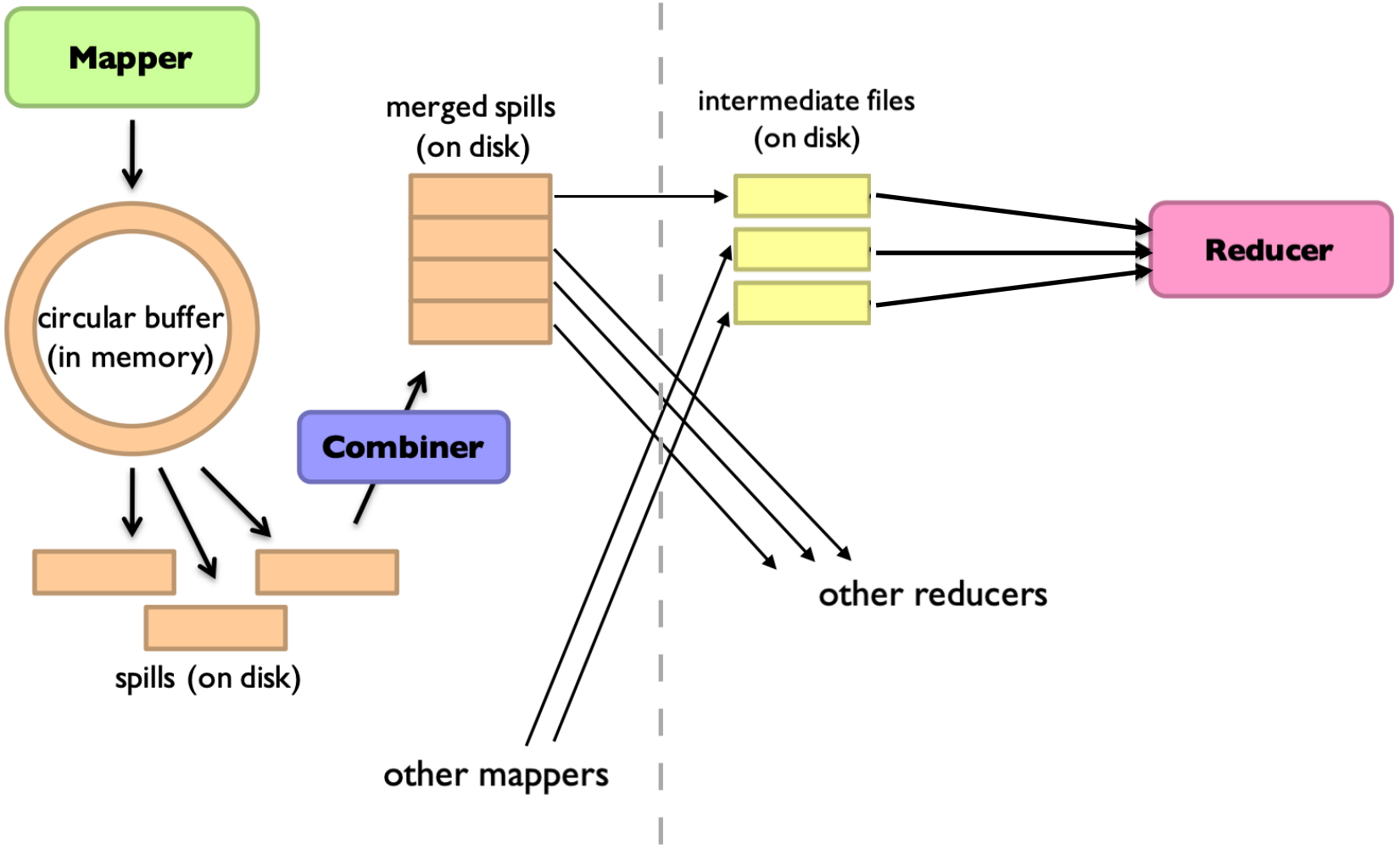
Why can't we achieve this?

- synchronization requires communication
- communication kills performance

Thus… avoid communication, as much as possible!

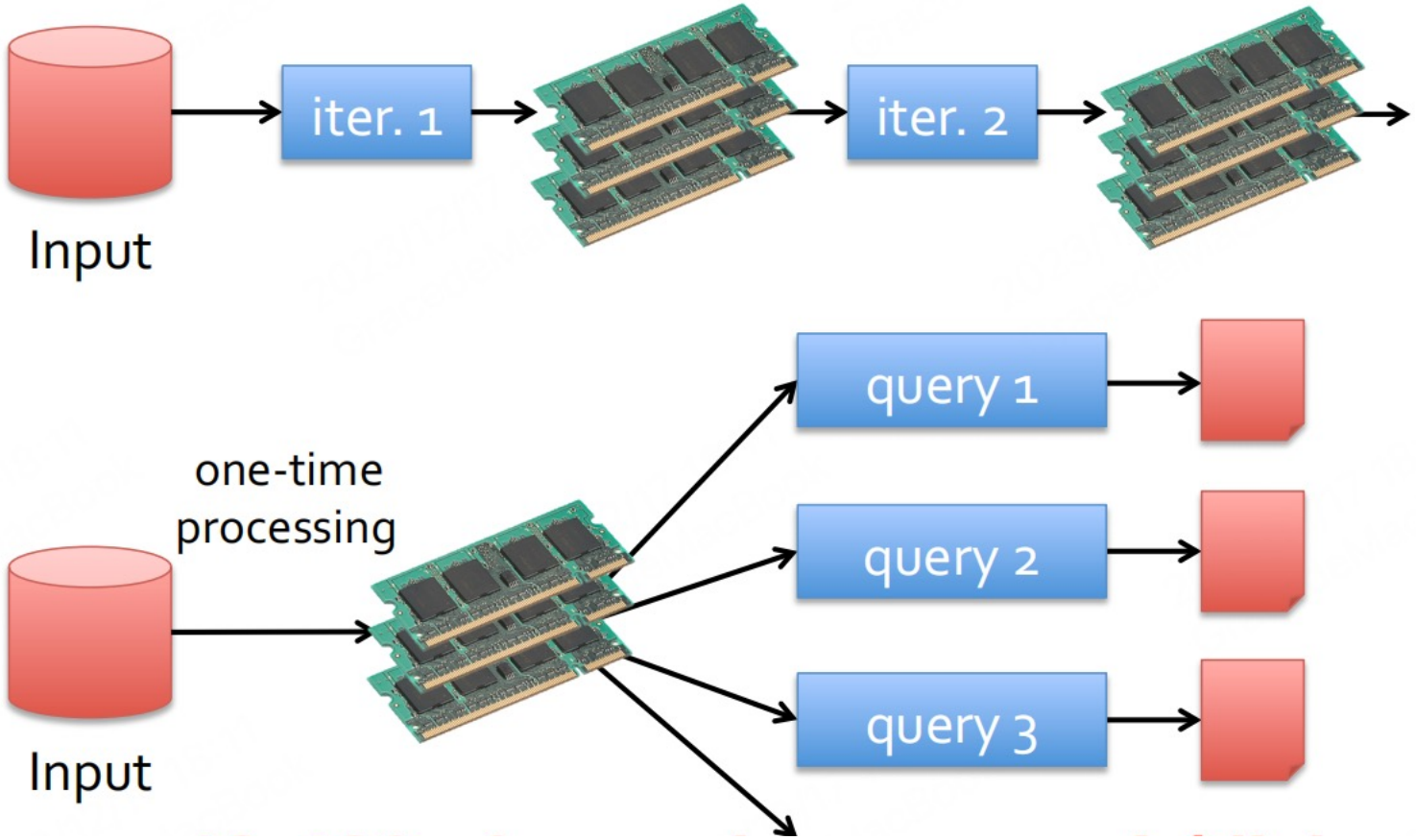
- reduce intermediate data via local aggregation
- combiners can help

Network


Mapper
circular buffer
(in memory)
spills (on disk)
Combiner
merged spills
(on disk)
intermediate files
(on disk)
Reducer
other reducers
other mappers

From MapReduce to Spark

Spark: In-memory data sharing



10-100x faster than network/disk

Zaharia, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," Talk in NSDI 2012.

RDDs

Restricted form of distributed shared memory

- Immutable, partitioned collections of records
- Built through coarse-grained, deterministic transformations (map, filter, join, …)

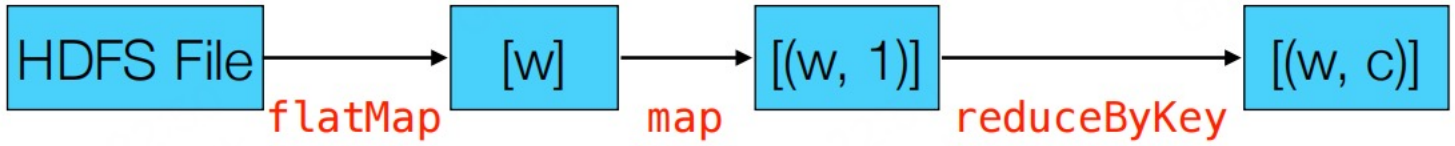
Efficient fault recovery using lineage

- Log one operation to apply to many elements
- Recompute lost partitions on failure
- No cost if nothing fails

Spark WordCount

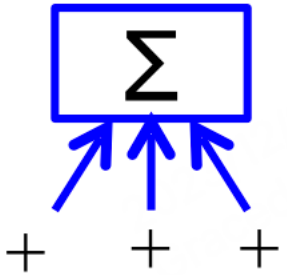
```
# create an RDD from HDFS
text_file = sc.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split(" ")) \
         .map(lambda word: (word, 1)) \
         .reduceByKey(lambda a, b: a + b) \
         .saveAsTextFile("hdfs://...")
```



PySpark Shared Variables

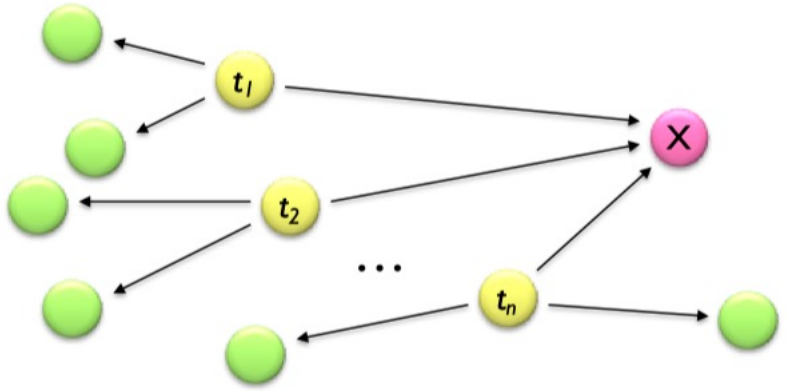
Broadcast variables
Accumulators



PageRank

Evaluate the importance of web pages, rank them in search engines

- Start each page with a rank
- On each iteration, update each page's rank to $\Sigma_{i \in neighbors} rank_i / |neighbors_i|$



Recap: What's an RDD?

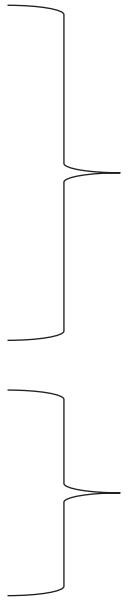
Set of partitions

List of dependencies on parent RDDs

Function to compute a partition given its parents

(Optional) partitioner (hash, range)

(Optional) preferred locations for each partition



Spark vs. MapReduce

	Hadoop MapReduce	Spark
Storage	Disk only	In memory or on disk
Operations	Map and Reduce	Map, Reduce, Join, Sample, etc.
Execution model	Batch	Batch, interactive, streaming
Programming environments	Java	Scala, Java, R and Python

From RDDs to DataFrames

RDD with schema: distributed collection of data grouped into named columns

immutable once constructed

track linage info to efficiently recompute lost data

enable operations on collection of elements in parallel

```
# RDD Version:
data = sc.textFile("data.txt").map(lambda line: line.split(" "))
data.map(lambda x: (x[0], [int(x[1]), 1])) \
  .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
  .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
  .collect()
```

```
# DataFrame Version:
df.groupBy("name").avg("age").collect()
```

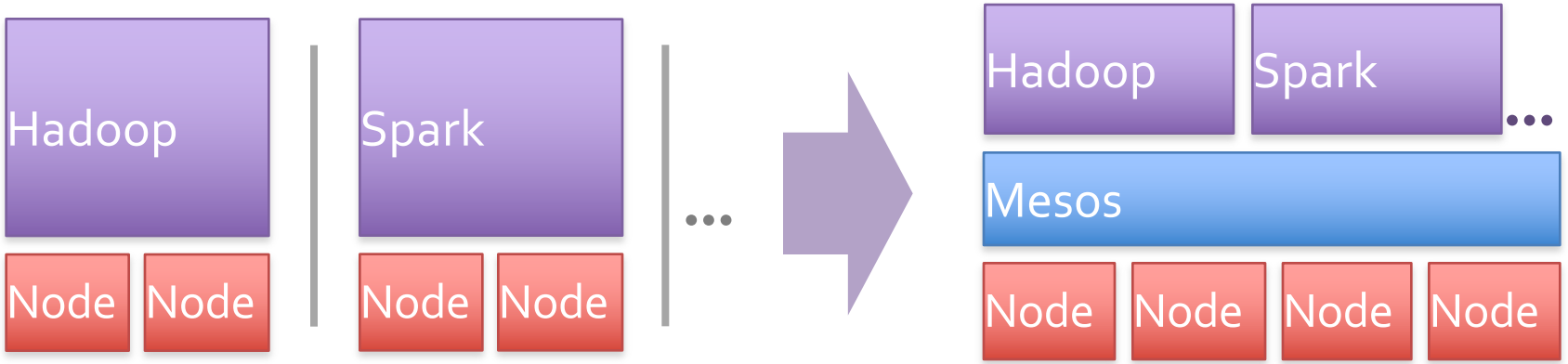
Cluster Management

Why cluster management?

- It is very difficult and challenging for a single framework (application-specific) to efficiently manage the resources of clusters
- The aim of the cluster management is to be able to run multiple frameworks in a single cluster to:
  - maximize utilization
  - share data between frameworks

Mesos

A thin resource sharing layer that enables fine-grained sharing across diverse cluster computing frameworks, by giving frameworks a common interface for accessing cluster resources.



Sample questions

Cloud concepts & models

(a) SmartHealth Inc. has many departments that run different types of analytics applications on large patient data sets. These data sets contain sensitive personal information that is highly private to patients. On the other hand, technical consultations suggest significant cost savings if the company were to adopt the cloud model. Which deployment model would you recommend for SmartHealth to adopt, and why?

Private cloud

The private cloud deployment model would enable the company to run analytics on its regulated patient data internally and still lead to savings.

(b) What are the possible reasons that could dissuade a company from the adoption of the cloud model? Please list two reasons and explain.

1. Vendor lock-in: different cloud platforms have different features, APIs, and ecosystems that could lock in the customer.
2. Privacy/security: customers cannot be assured that their data are securely stored without a privacy leakage.
3. Cost of migration to the cloud.
4. Software licensing issues.

Virtualization

(a) Trap-and-emulate is a basic approach used to implement virtualization. (1) How does it work? (2) How about its performance?

(1) Instead of OS providing high-level abstractions to process via system calls, trap-and-emulate relies on VMM to provide a virtual HW/SW interface to guest OS by trapping and emulating sensitive instructions.

(2) Almost no overhead for non-sensitive instructions; significant performance hit for sensitive instructions

(b) While para-virtualization generally outperforms full virtualization, the latter is still widely used in many systems. What is the advantage of full virtualization over paravirtualization? Explain your answer.

Ability to run unmodified OSs within guest VMs. With para-virtualization, the guest OSs must be modified to interact properly with the hypervisor, through hypercalls. With full hardware virtualization, the guest OSs need not be aware that they are not running on real hardware.

Distributed storage systems

(a) Justify if HDFS is suitable for the following cases:

A. A data-intensive application that batch processes the dynamically generated log messages in a large web service cluster.
B. An interactive data analytics job that responds to user queries in sub-seconds.

A. Suitable. Log data are write-once, mostly appended to. Batch analytics jobs are less latency-sensitive and perform streaming reads.
B. Ill-suited. HDFS incurs a long I/O latency in exchange of high throughput.

(b) Your company uses a distributed data processing system based on Hadoop, deployed on Amazon EC2 instances with all data stored in an HDFS file system running on those same instances. Your colleague argues that, because HDFS maintains three replicas of each block, it is safe to use Spot instances instead of on-demand instances. Do you agree? Explain your answer.

No. All of the spot instances could be taken back by Amazon at the same time, voiding the benefits of the replication. Replication in HDFS and other distributed storage assumes some degree of independence of failures, so as to provide enough time for repair before all machines holding replicas of a block fail.

MapReduce

(a) Assume you run a word count MapReduce program with M mappers and R reducers. There are K different words each occurring one or multiple times in the input file, where $K >> R$. How many output files will you get at the end of the job? And how many key-value pairs will there be in each file?

There will be R output files, each having K/R key-value pairs.

MapReduce (Cont.)

Consider the Hadoop pseudo code shown below. It computes, for each key, the log average across all values associated with that key. The formula for the log average is:

$$logAvg = \exp\left(\frac{1}{n}\sum_{i=1}^{n}\log a_i\right)$$

```
map(string t, int r):
       emit(string t, int r)

reduce(string t, ints [r1, r2, r3, ...]):
       float sum = 0.0;
       int count = 0;

       foreach int r in ints:
          sum += log(r);
          count++;

       float logAvg = exp( 1/(float)count * sum );
       emit(string t, float logAvg);
```

Rewrite the pseudo-code above and include a combiner to speed up the job. Make sure that both versions of the code, with and without the combiner, give the same final results.

```
map(string t, int r):
       Emit(string t, pair((float) log(r), 1))

combiner(string t, pairs [p1, p2, p3, ...]          ):
       float sum = 0.0
       int count = 0;
       foreach pair p in pairs:
              sum += p._1
              count++;
       Emit(string t, pair(sum, count))

reduce(string t, pairs [p1, p2, p3, ..]             ):
       float sum = 0.0;
       int count = 0;

       foreach pair p in pairs:
          sum += p._1;
          count += p._2;

       float logAvg = exp( 1/(float)count * sum );
       Emit(string t, float logAvg);
```

Spark

(a) Spark has many benefits over MapReduce. List and explain three of them.

1. "in-memory computing": Spark allows the application program to cache frequently used RDDs in application memory so future computation may reuse data from memory; while Hadoop can only share data through stable storage, leading to a long I/O overhead.
2. Spark provides a rich set of operations beyond "map" and "reduce", making it easier to program than Hadoop.
3. Spark supports multiple languages, including Java, Python, Scala, and R; Hadoop only supports Java, which is more verbose to program than Python, Scala, and R.
4. Spark supports diverse computing models beyond batch data processing, such as stream analytics machine learning, graph analytics.

Any three of above should be fine.

Spark (Cont.)

(b) Write down the output of the following code snippets, and justify your answer:

```
odd = sc.accumulator(0)
even = 0

def count(element):
  global even
  if element % 2 == 0:
    even += 1
  else:
    odd.add(1)

sc.parallelize([1, 6, 7, 8, 3, 4, 4, 2]).foreach(count)
print odd,even
```

3 0