[544] The Cloud

Meenakshi Syamkumar

Learning Objectives

- recall different things that typically show up on a cloud bill (for example, different types of network I/O)
- identify PaaS cloud offerings that are similar to the open-source systems we have been learning this semester
- describe BigQuery's relationship to other systems

Outline

Background

Resources

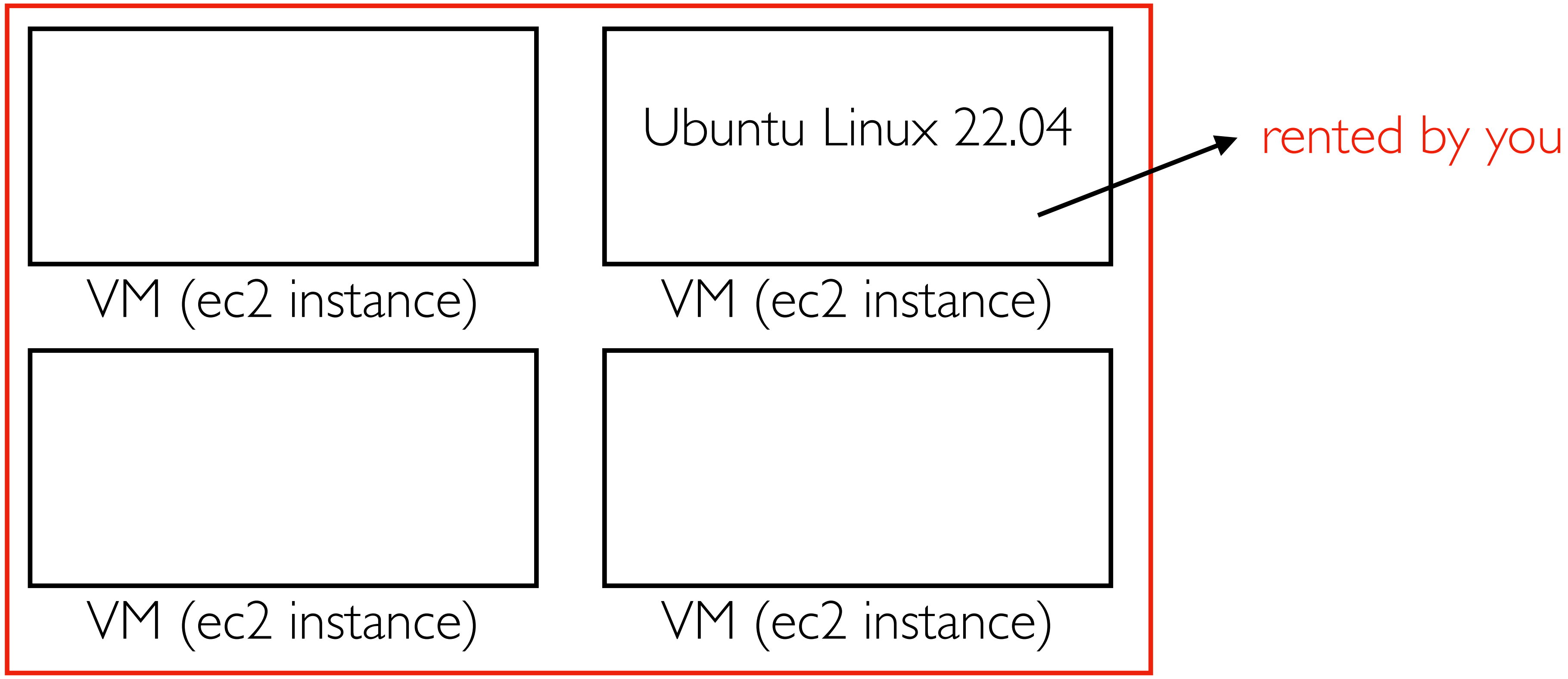
Billing Models

Platforms

The Beginning

Amazon Web Services (AWS)

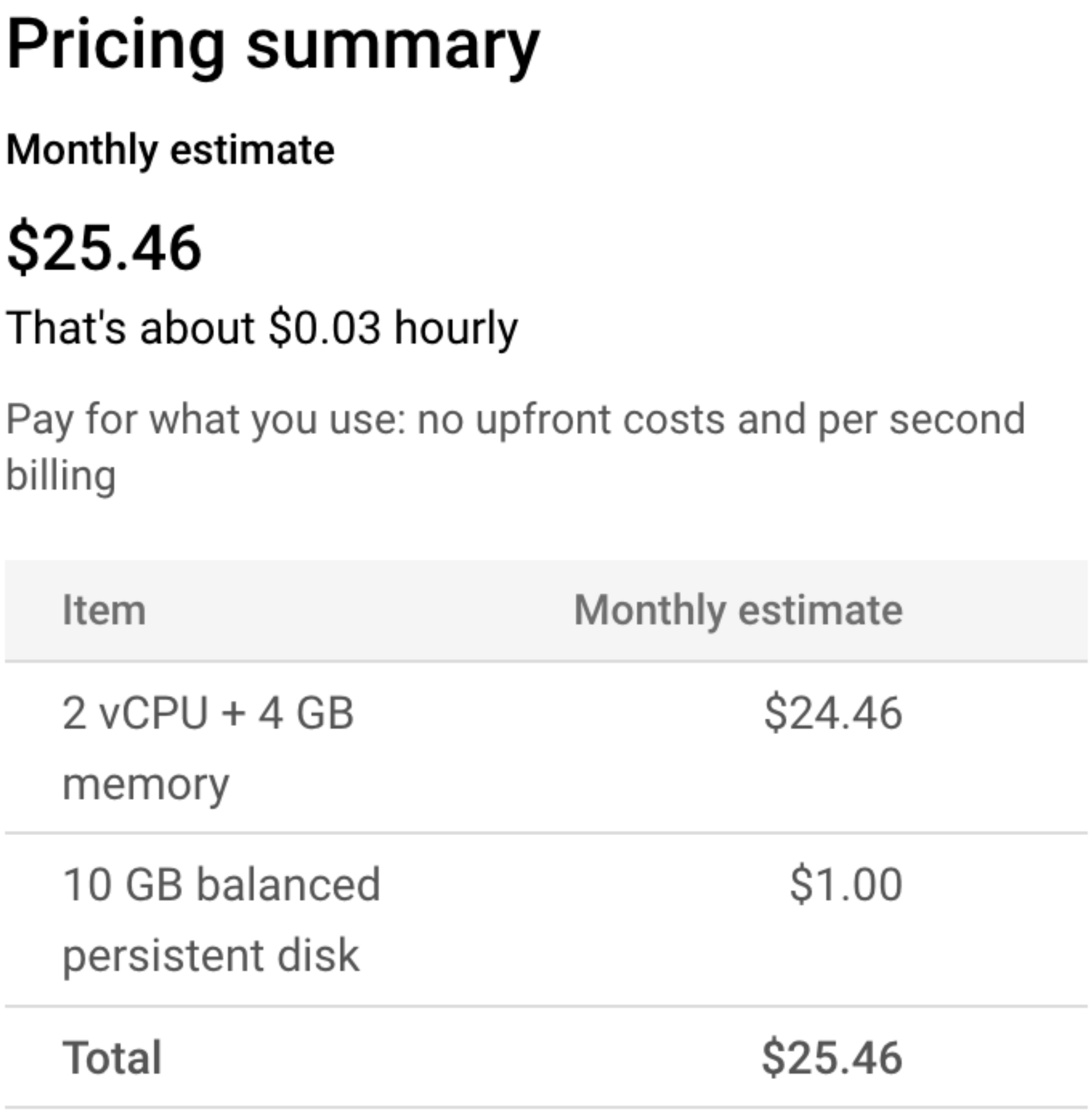
- Elastic Compute Cloud (EC2), rented VMs, launched in 2006
- "Infrastructure as as Service" (IaaS) -- rent infrastructure (network, storage, compute) instead of owning the hardware yourself.



physical machine in a Amazon data center

"Sometimes you need a lot of processing power, and sometimes you need just a little. Sometimes you need a lot, but you only need it for a limited amount of time."
~ Jeff Barr (https://aws.amazon.com/blogs/aws/amazon_ec2_beta/)

VM Hours



Pricing summary

Monthly estimate

$25.46

That's about $0.03 hourly

Pay for what you use: no upfront costs and per second billing

Item	Monthly estimate
2 vCPU + 4 GB memory	$24.46
10 GB balanced persistent disk	$1.00
Total	**$25.46**

Pricing comparison

- one VM for a month: about $25
- about 744 hours/month (31*24)
- 744 VMs for an hour: about $25
- same computation resources
- very different wait time

Be careful!

- programmers previously optimized when things were too slow
- now we need to optimize when it is too expensive
- cost is not always obvious at the moment you're running a job (need to do "back of the envelope" estimates until you get a bill)

Other Cloud Services

AWS now has >200 services beyond EC2 (and growing).

IaaS (Infrastructure as a Service)

- EC2, other services that feel closer to raw hardware
- virtual disks, virtual network, some storage systems, etc.
- cheap+flexible -- you can deploy anything on it (Cassandra, Kafka, etc).

PaaS (Platform as as Service)

- Cloud provider has deployed systems on the infrastructure; you pay to use the deployed system
- databases, application framework/platforms, ML training/deployment systems
- less flexible, easier to use
- often more expensive (though not necessarily more than doing it yourself due to efficiencies available to cloud provider but not you)

Line between IaaS vs. PaaS distinction is a bit subjective.

Lock In

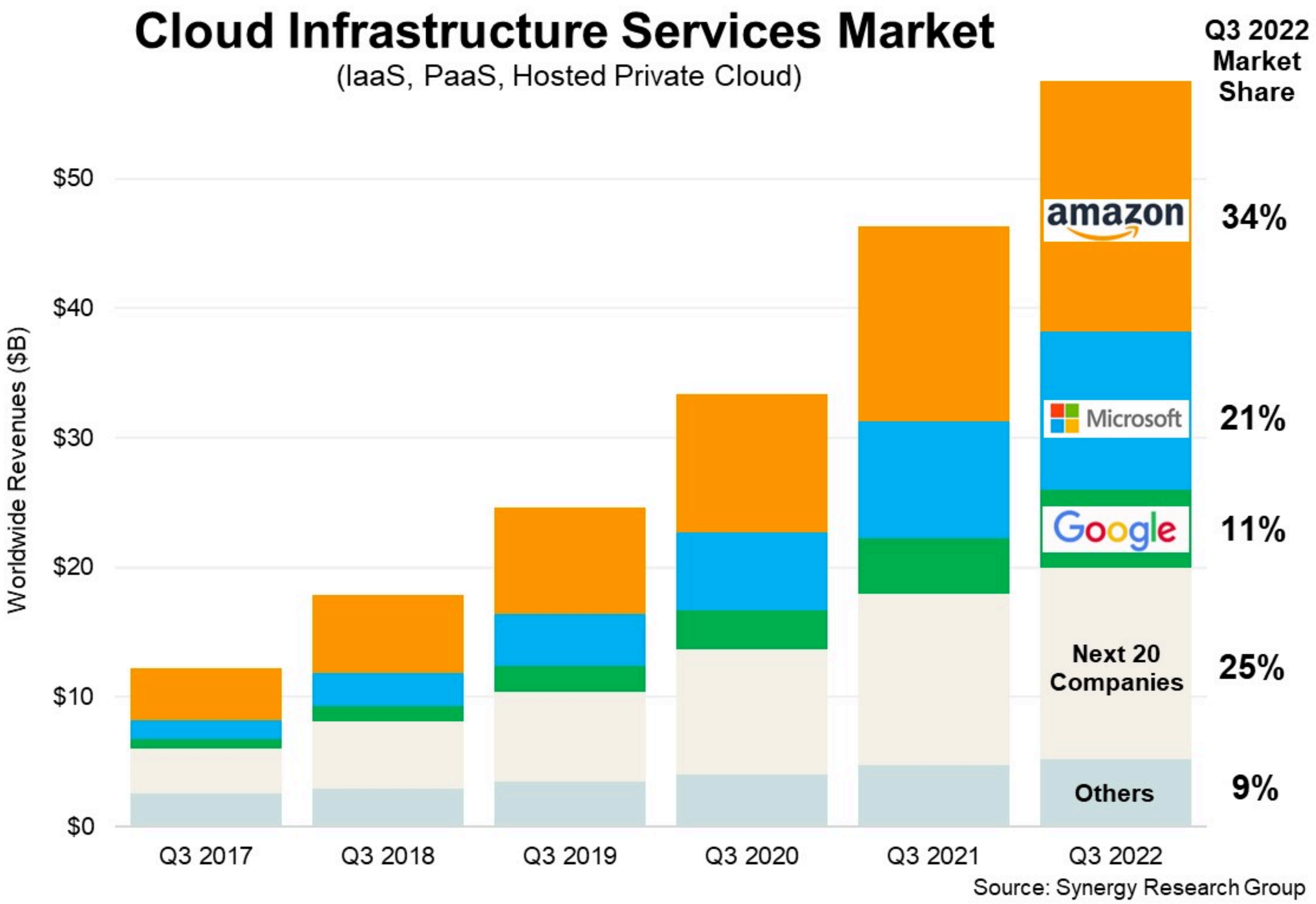
Customers worry: what if the cloud provider increases the price? If it's hard to move to a competing cloud, you're "locked in".

PaaS: services are often unique, and it would be hard to move to a different cloud providers.

IaaS: services like VMs are more uniform -- it would be easier to switch to a different cloud to find the cheapest place to rent VMs.

Data: cloud providers often make it free to bring data into the cloud (ingress) but expensive to take it out (egress).

Major Cloud Providers Today



Source: Synergy Research Group

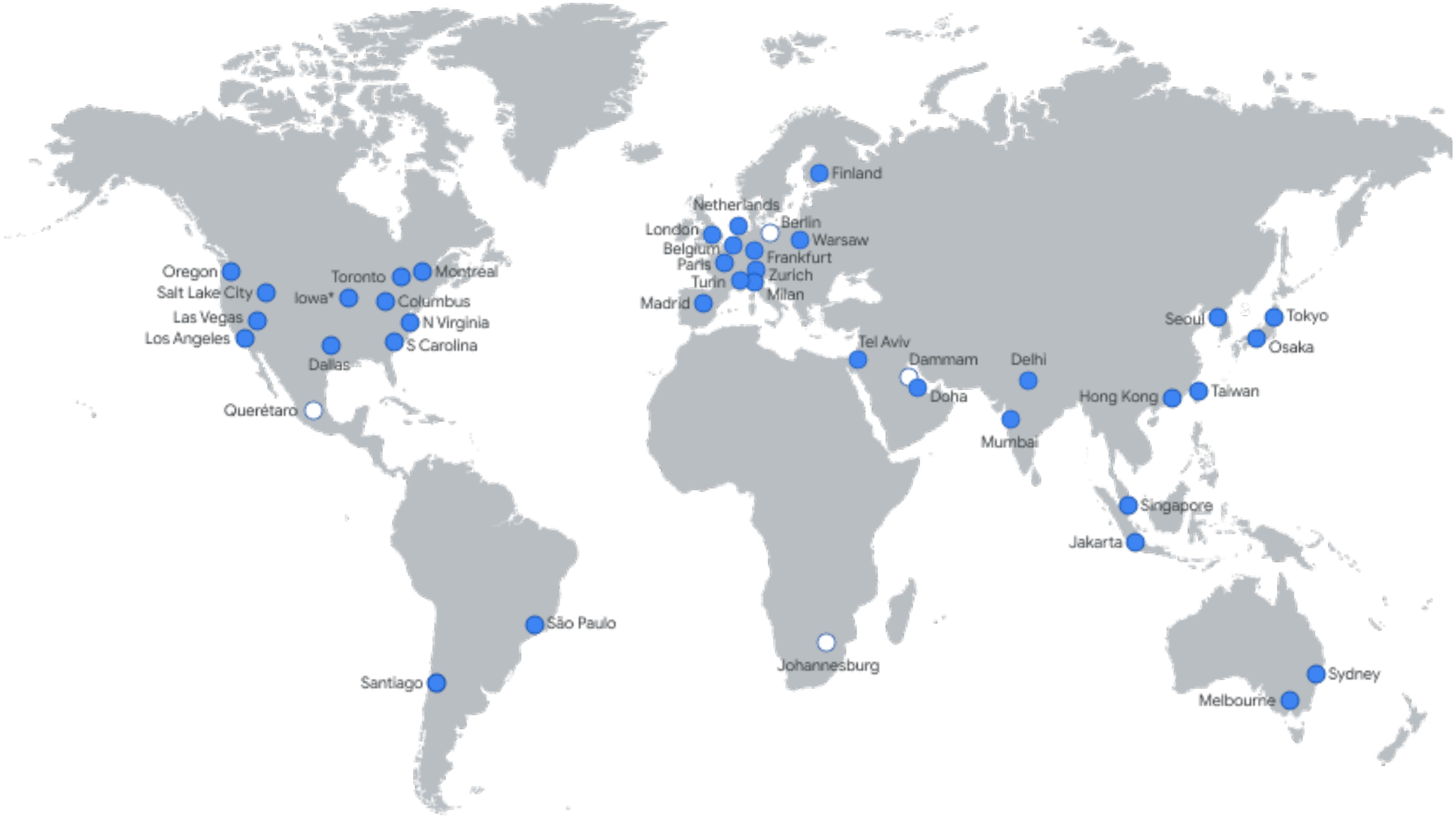
AWS

Azure

GCP

https://www.srgresearch.com/articles/q3-cloud-spending-up-over-11-billion-from-2021-despite-major-headwinds-google-increases-its-market-share

Numerous Regions Globally (GCP in 2023)



https://cloud.google.com/about/locations#regions

Outline

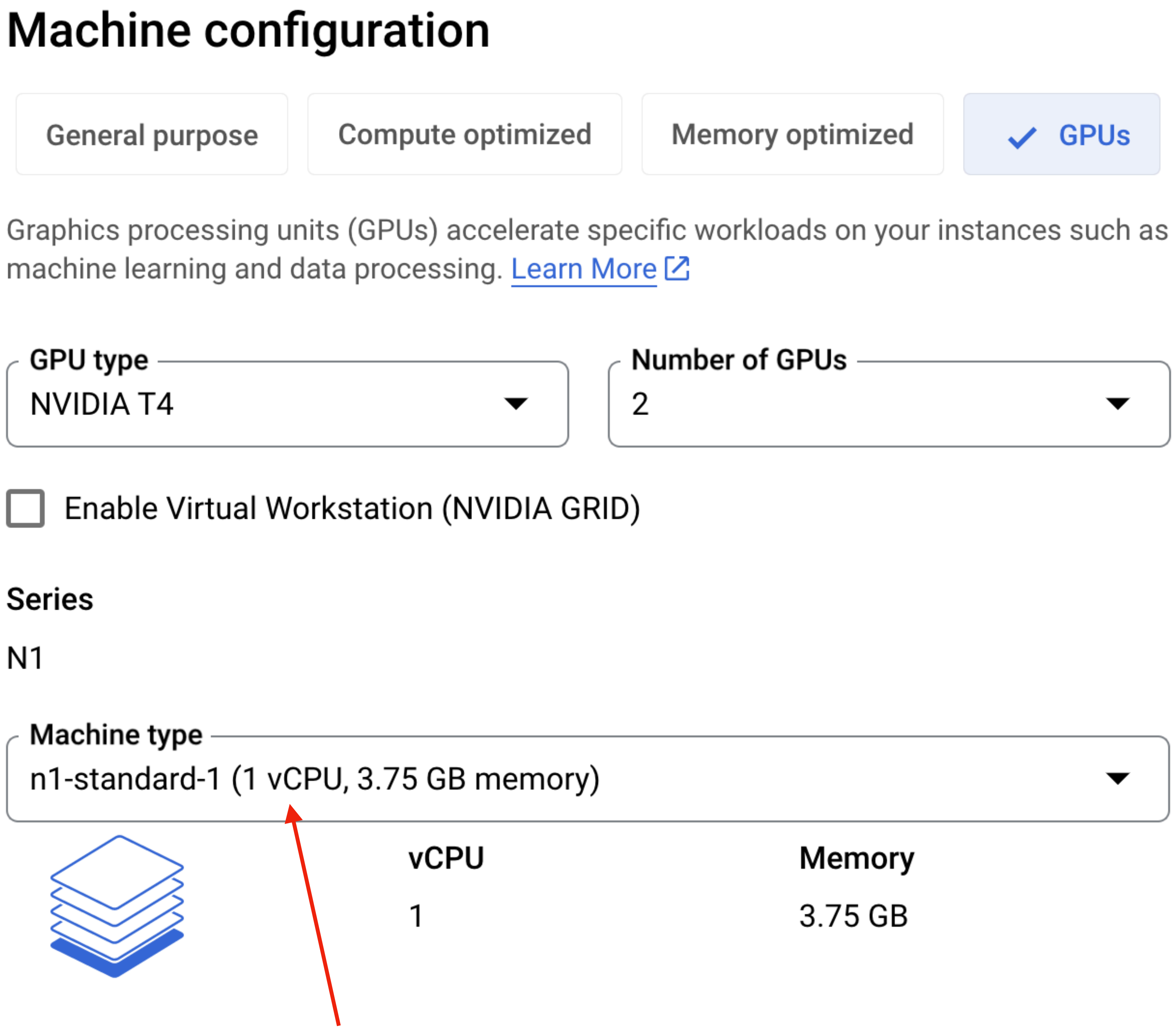
Background

Resources

Billing Models

Platforms

Compute - Memory - Storage - Network



Machine configuration

Graphics processing units (GPUs) accelerate specific workloads on your instances such as machine learning and data processing. Learn More

GPU type: NVIDIA T4

Number of GPUs: 2

□ Enable Virtual Workstation (NVIDIA GRID)

Series

N1

Machine type: n1-standard-1 (1 vCPU, 3.75 GB memory)

vCPU	Memory
1	3.75 GB

can choose number and type of GPUs

Google offers TPUs (tensor processing units) -- custom hardware for ML. Works with PyTorch and TensorFlow

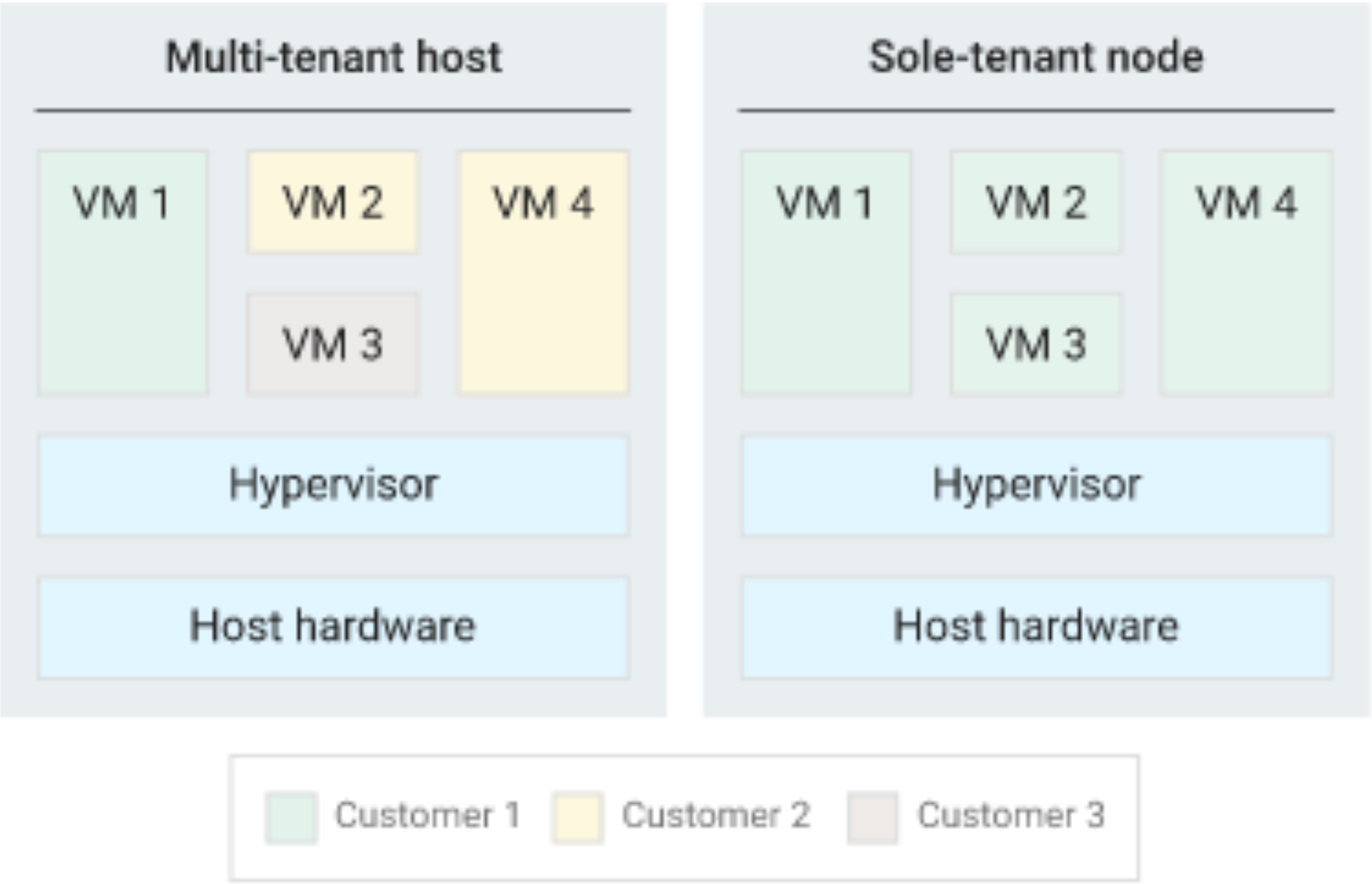
can choose number of vCPUs

this VM is ~$400/month (or $0.50/hour)

Compute - Memory - Storage - Network

Forms in which to buy compute

- VMs on multi-tenant hosts (typical case)
- VMs on sole-tenant hosts (better isolation/security, $1000s/month)
- Containers (Kubernetes Engine)
- Serverless Functions (functions run when events happen; pay by the millisecond)



https://cloud.google.com/compute/docs/nodes/sole-tenant-nodes

Compute - **Memory** - Storage - Network

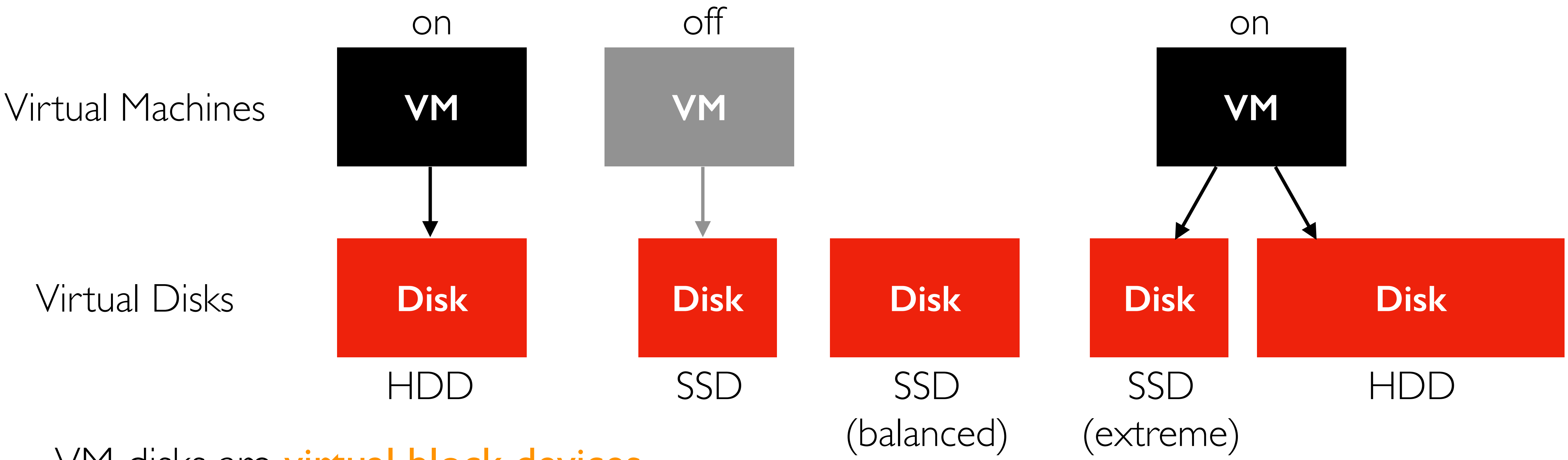
IaaS

- memory is often roughly proportional to CPU resources
- "memory optimized" VMs skew heavy on RAM
(very expensive! at high end >10 TB)

PaaS: often open-sources platforms provided as a service. Examples:

- memcached (cache)
- redis (in-memory DB)

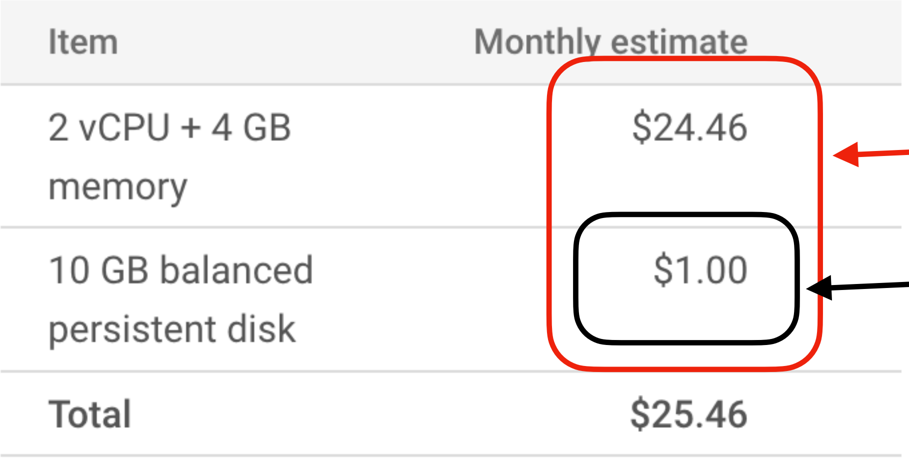
Compute - Memory - **Storage** - Network



VM disks are **virtual block devices**

- can be attached, detached, re-attached to VMs
- different disk types offer different performance/price tradeoffs
- HDD (standard); SSD (balanced, SSD, extreme)
- price depends on size and type

VM creation:

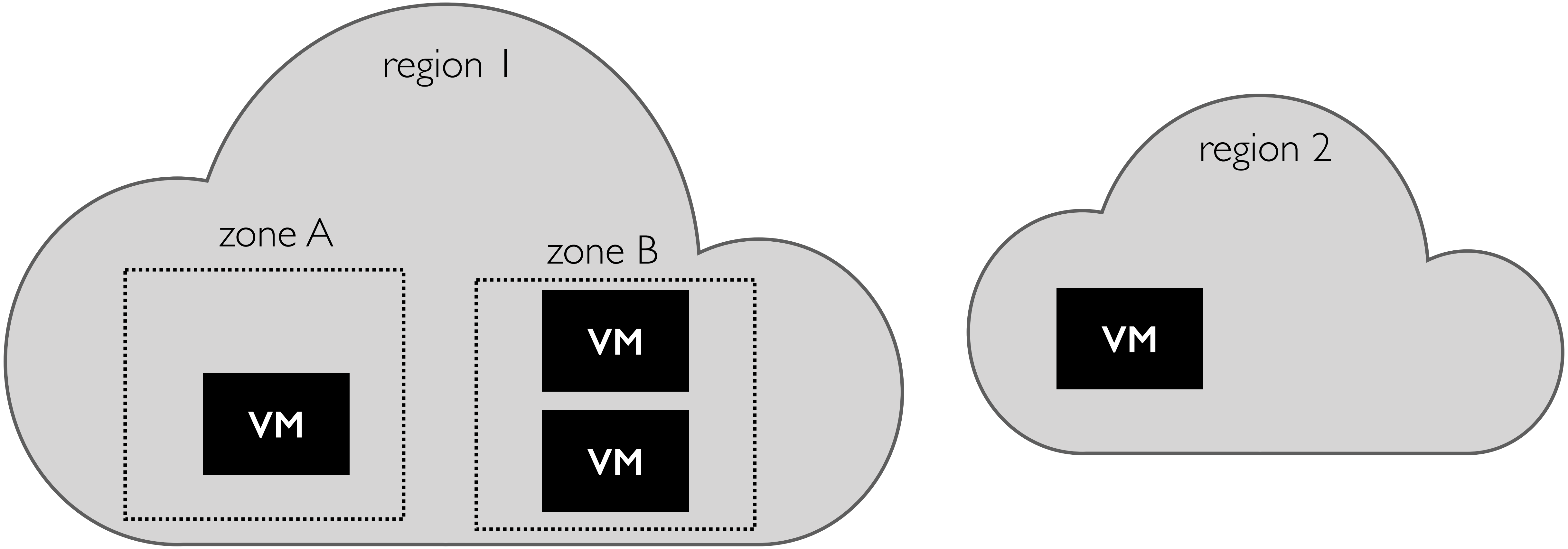


Item	Monthly estimate
2 vCPU + 4 GB memory	$24.46
10 GB balanced persistent disk	$1.00
Total	**$25.46**

cost when running

cost when off (or VM only deleted)

Compute - Memory - Storage - Network



Cloud hierarchy

- continents (approximate)
- regions (data center consisting of 1 or more nearby buildings)
- zone (area of region with fast interconnect but usually common points of failure, like power, routers, etc)

Compute - Memory - Storage - Network

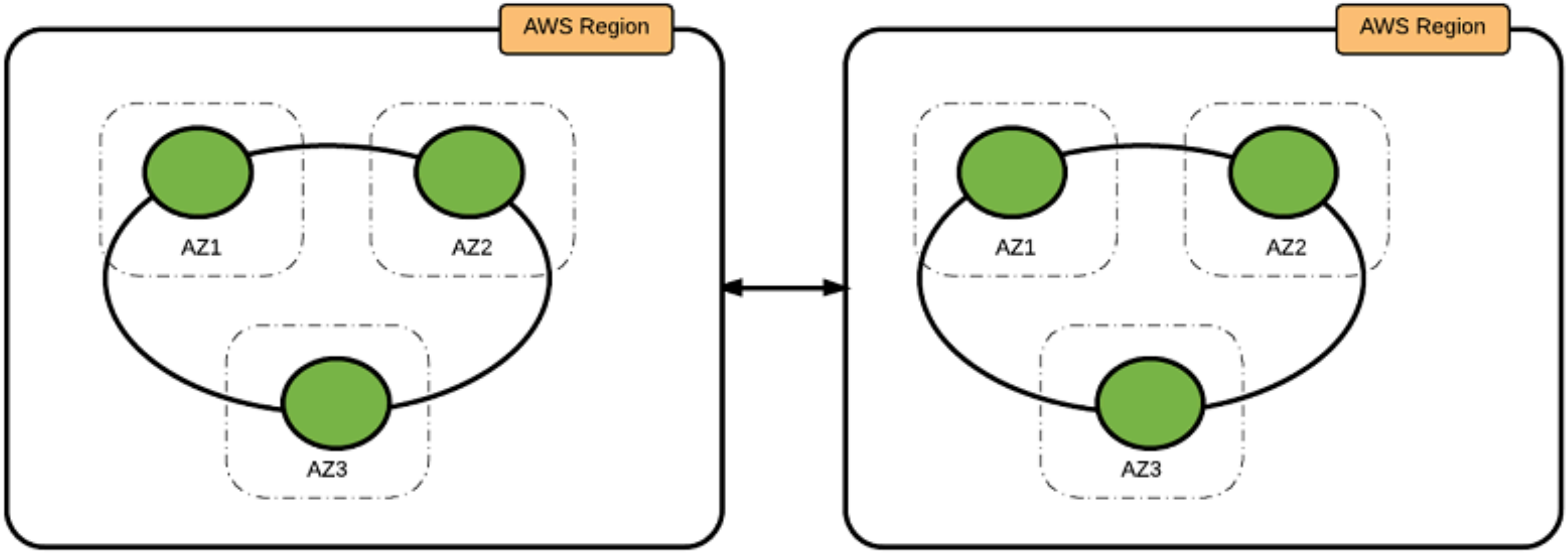
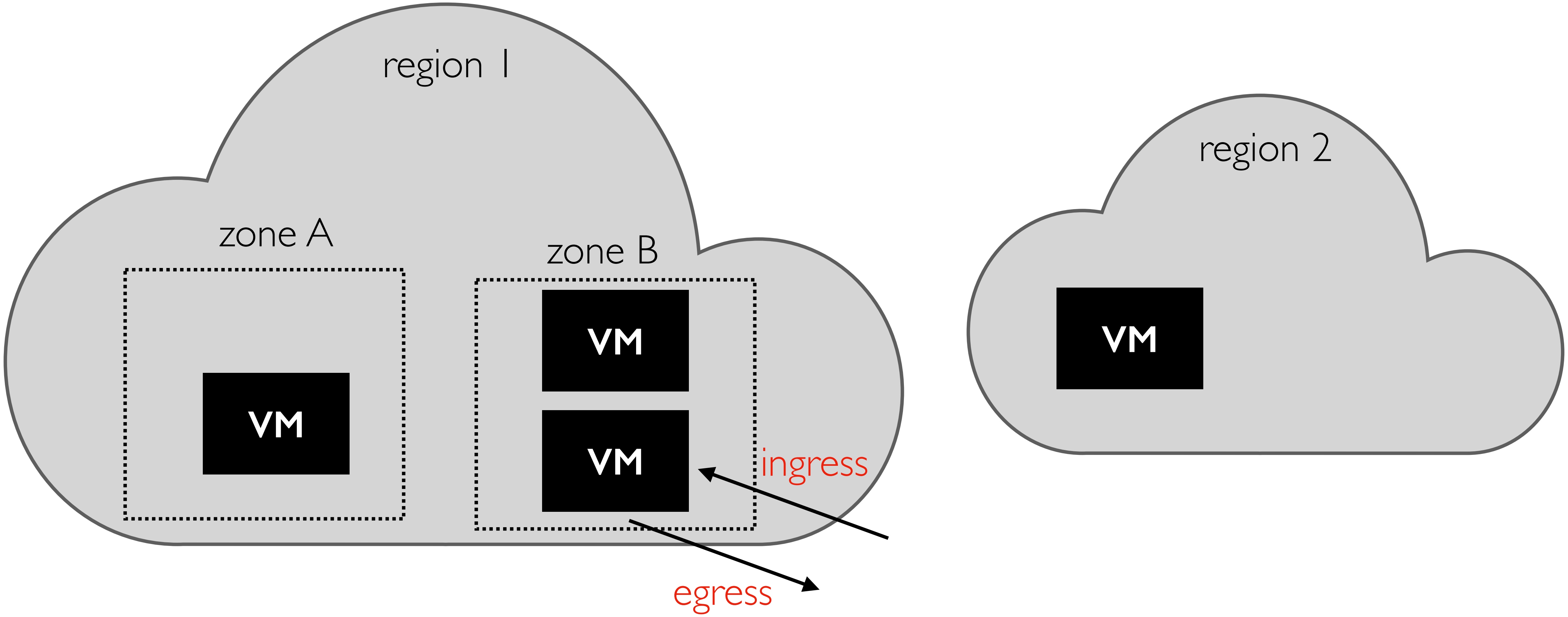


Image from *Best Practices for Running Apache Cassandra on Amazon EC2*
(https://aws.amazon.com/blogs/big-data/best-practices-for-running-apache-cassandra-on-amazon-ec2/)

Fault tolerance

- deploy under the assumption that nodes in the same zone may reasonably all go down together (e.g., due to power loss)
- being extra careful: assume a region can go down (e.g., tornado destroys couple buildings)

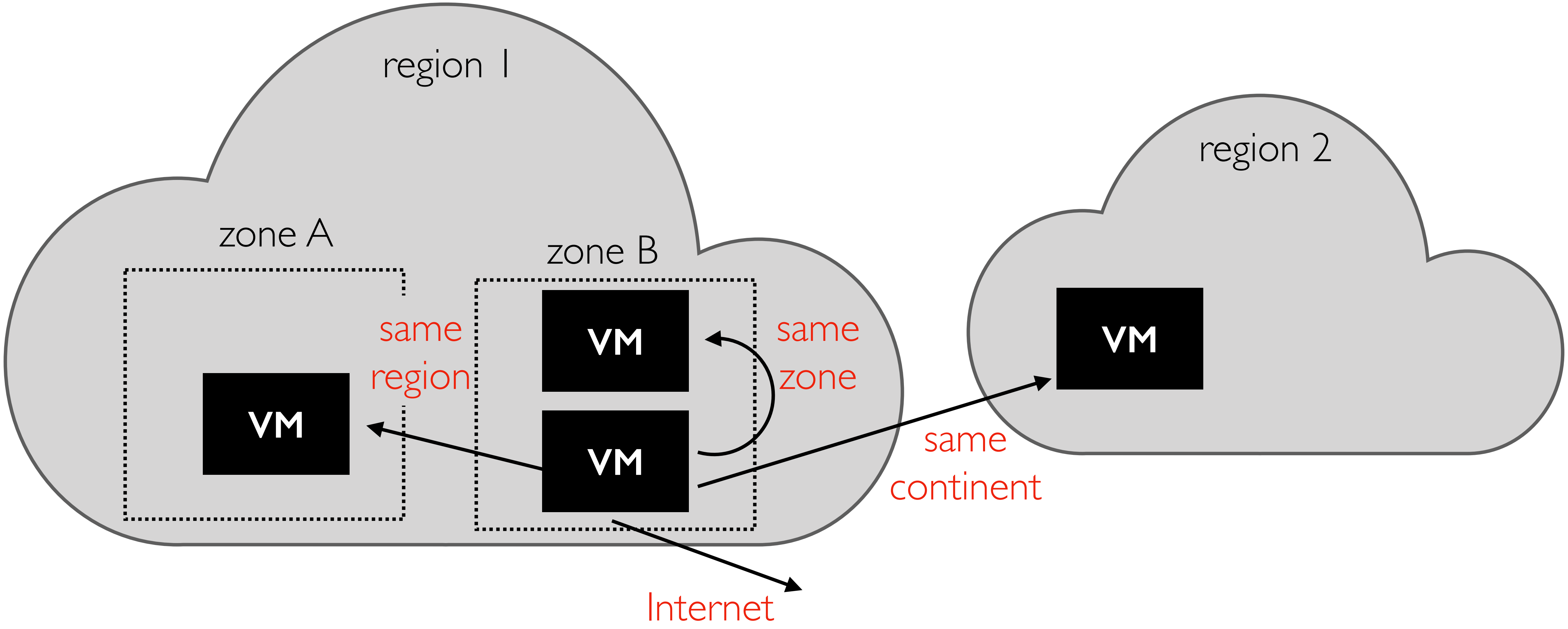
Compute - Memory - Storage - Network



Clouds generally bill per GB of network I/O

- ingress is usually free (incentivize you to start using the service, charge to move your data elsewhere
- egress rate is complicated (depends on many factors)

Compute - Memory - Storage - **Network**



Egress examples (ballpark for GCP in 2023, but very simplified):

- Internet: $0.085/GB
- Same continent: $0.05/GB (Asia)
- Same region: $0.01/GB
- Same zone: free

TopHat

Outline

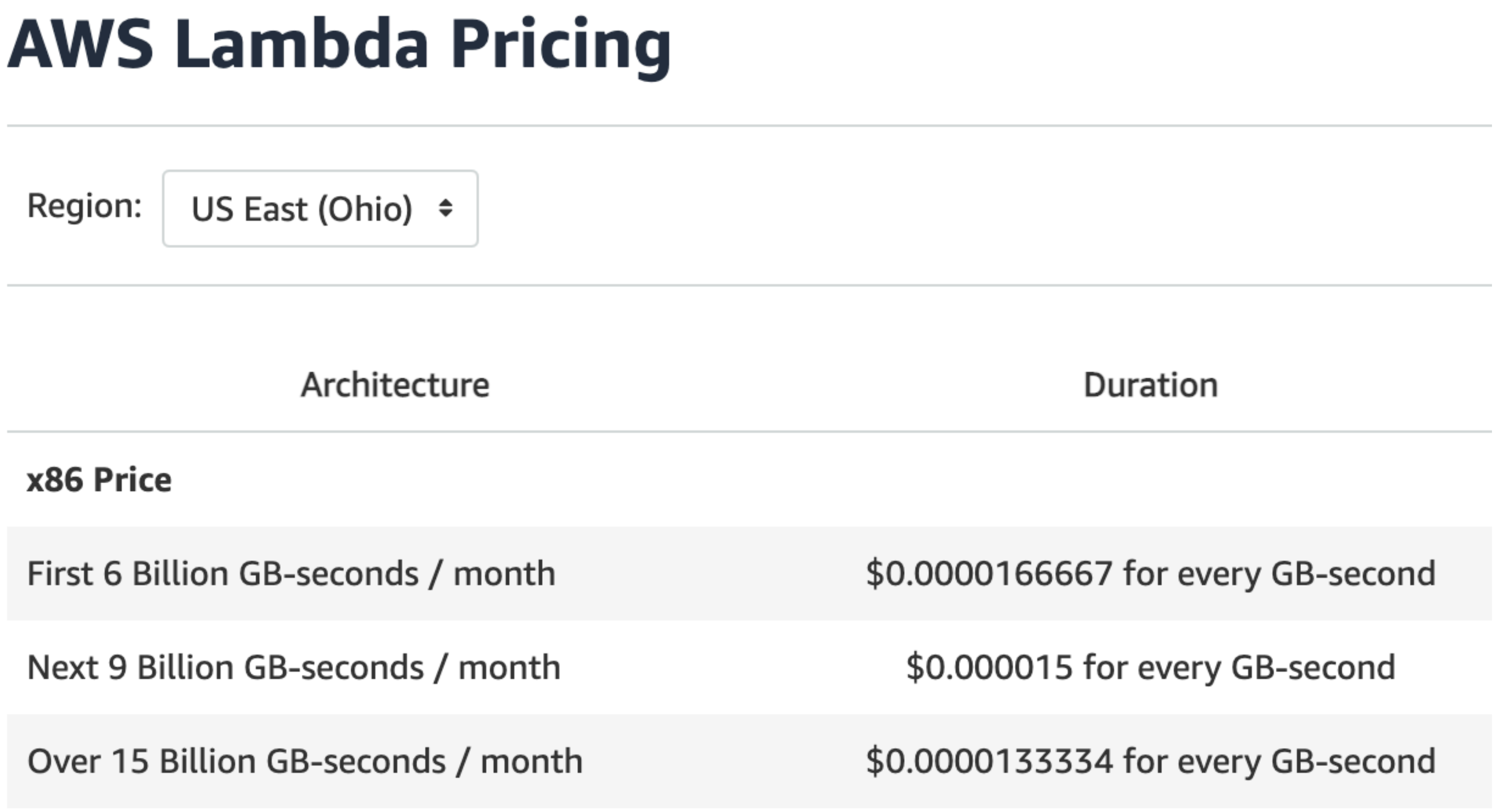
Background

Resources

Billing Models

Platforms

Free Tier, Discounts at Scale (AWS Lambda Example)



AWS Lambda Pricing

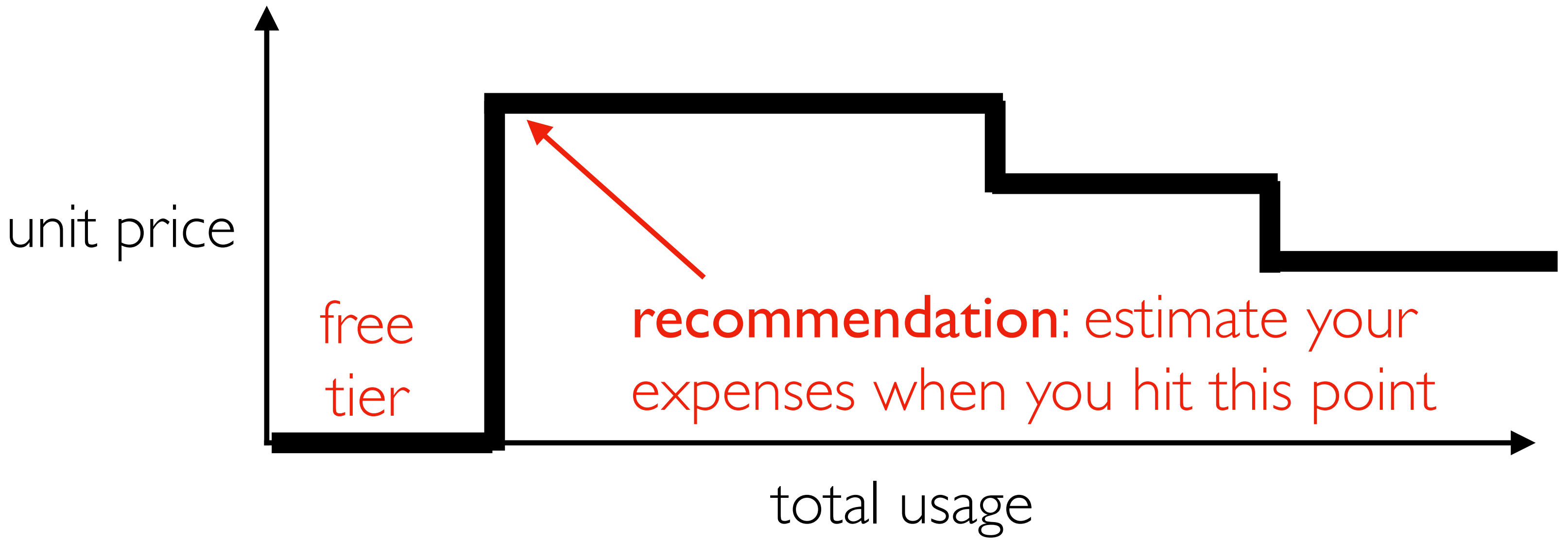
Region: US East (Ohio) ⬍

Architecture	Duration
x86 Price	
First 6 Billion GB-seconds / month	$0.0000166667 for every GB-second
Next 9 Billion GB-seconds / month	$0.000015 for every GB-second
Over 15 Billion GB-seconds / month	$0.0000133334 for every GB-second

"The AWS Lambda free tier includes one million free requests per month and 400,000 GB-seconds of compute time per month"

https://aws.amazon.com/lambda/pricing/

common billing model pattern

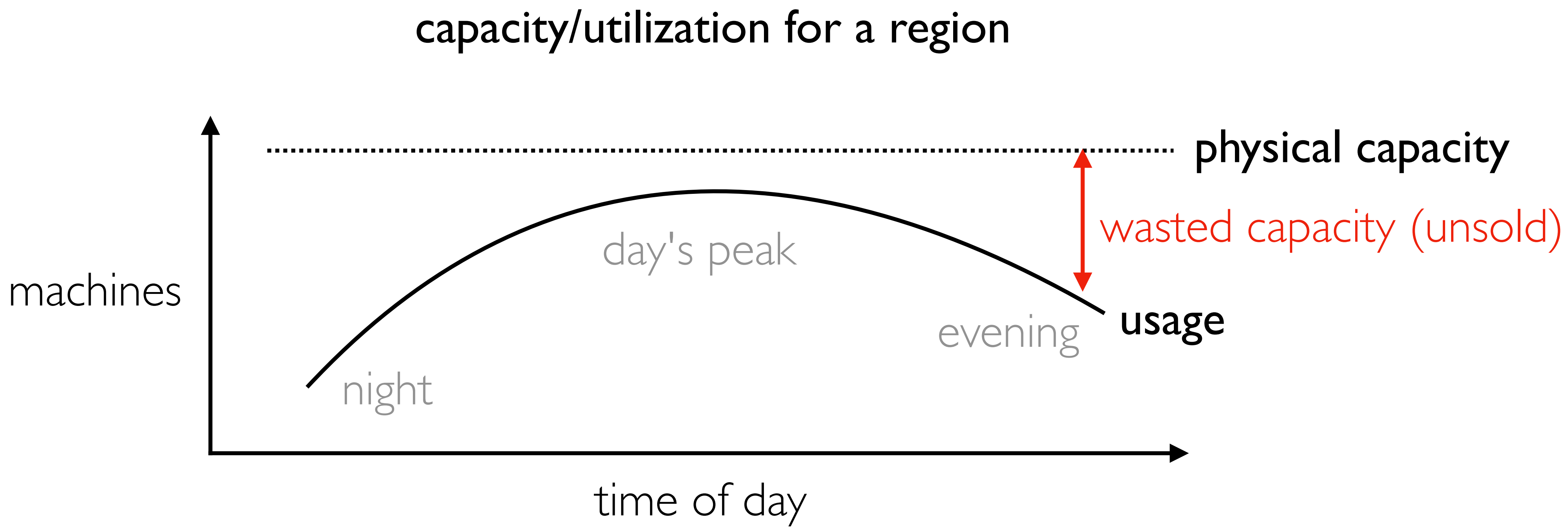


recommendation: estimate your expenses when you hit this point

"Duration is calculated from the time your code begins executing until it returns or otherwise terminates, rounded up to the nearest 1 ms*"

recommendation: check if you have a large number of small ops getting rounded up

On-Demand vs. Spot Instances



How to create incentives for customers?

- use **less** at peak time
- use **more** at low times

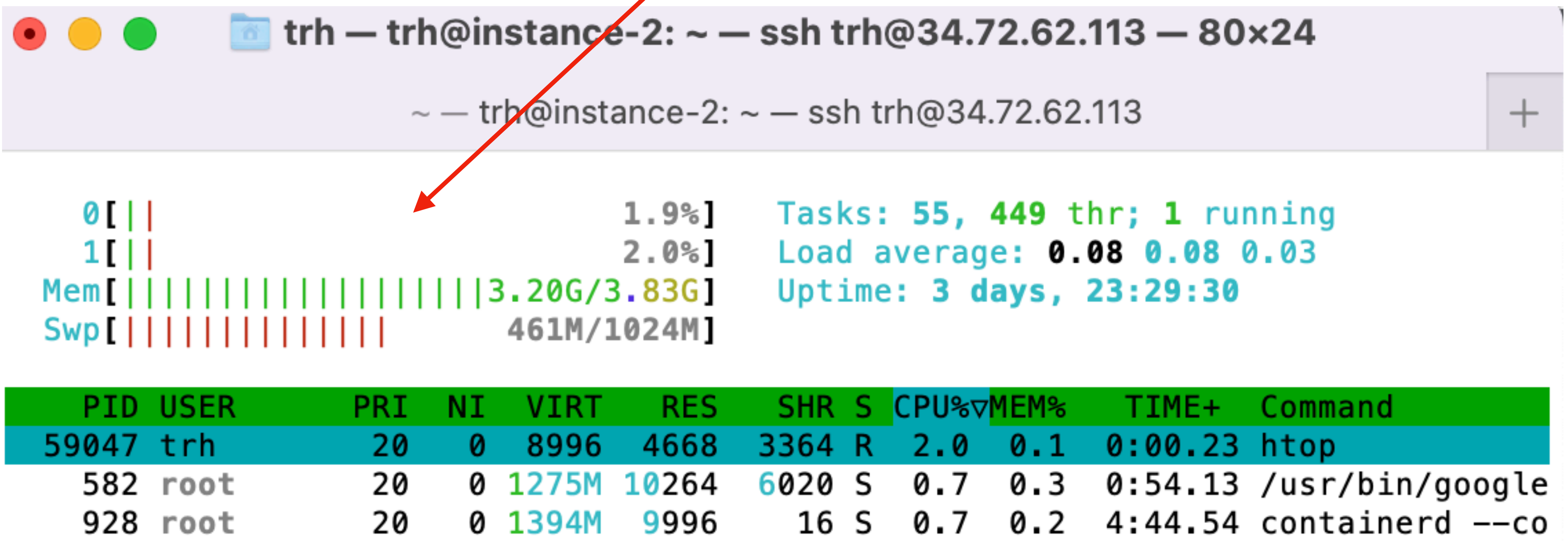
Two VM deployment options

- on-demand instances: constant (high) price. Can generally get a VM. Won't be taken away from your arbitrarily. Used when capacity is needed at specific times.
- spot instances: price varies throughout day. If you're not willing to pay enough, your computation waits for a cheaper price. VM might be interrupted ("preempted") once started. Excellent for once-a-day batch jobs.

Scaling and Billing

many (most?) VMs are mostly idle

but you pay the same rate for your VM based on the configuration, regardless of how you actually you the VM


trh — trh@instance-2: ~ — ssh trh@34.72.62.113 — 80×24
~ — trh@instance-2: ~ — ssh trh@34.72.62.113
+

```
    0[||                         1.9%]   Tasks: 55, 449 thr; 1 running
    1[||                         2.0%]   Load average: 0.08 0.08 0.03
  Mem[|||||||||||||||||||3.20G/3.83G]   Uptime: 3 days, 23:29:30
  Swp[||||||||||||||        461M/1024M]

    PID USER      PRI  NI  VIRT   RES   SHR S CPU%▽MEM%   TIME+  Command
  59047 trh        20   0  8996  4668  3364 R  2.0  0.1  0:00.23 htop
    582 root       20   0 1275M 10264  6020 S  0.7  0.3  0:54.13 /usr/bin/google
    928 root       20   0 1394M  9996    16 S  0.7  0.2  4:44.54 containerd --co
```

Models

- **fixed**: you configure what you want, then pay a constant amount. Low risk, often wasteful, doesn't handle unexpected bursts. Example: VM instances.
- **auto scaling**: the cloud service detects high/low load and automatically increases/decreases your reservation. Often cannot scale to zero. Example: Elastic Beanstalk
- **pay as you go**: pay for actual resources consumed with fine granularity. Example: AWS Lambda.

Outline

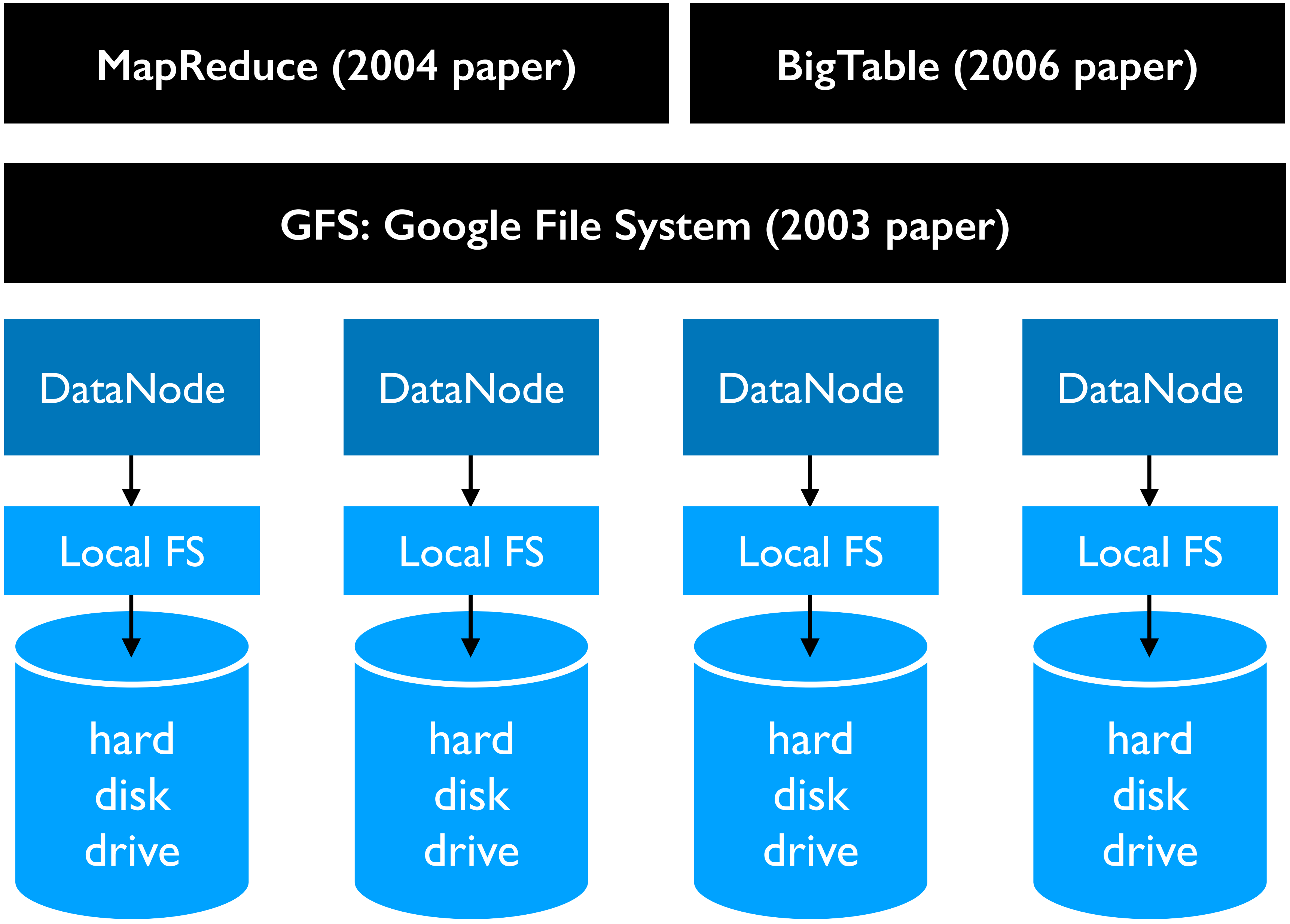
Background

Resources

Billing Models

Platforms

Review: Google Architecture (early systems)



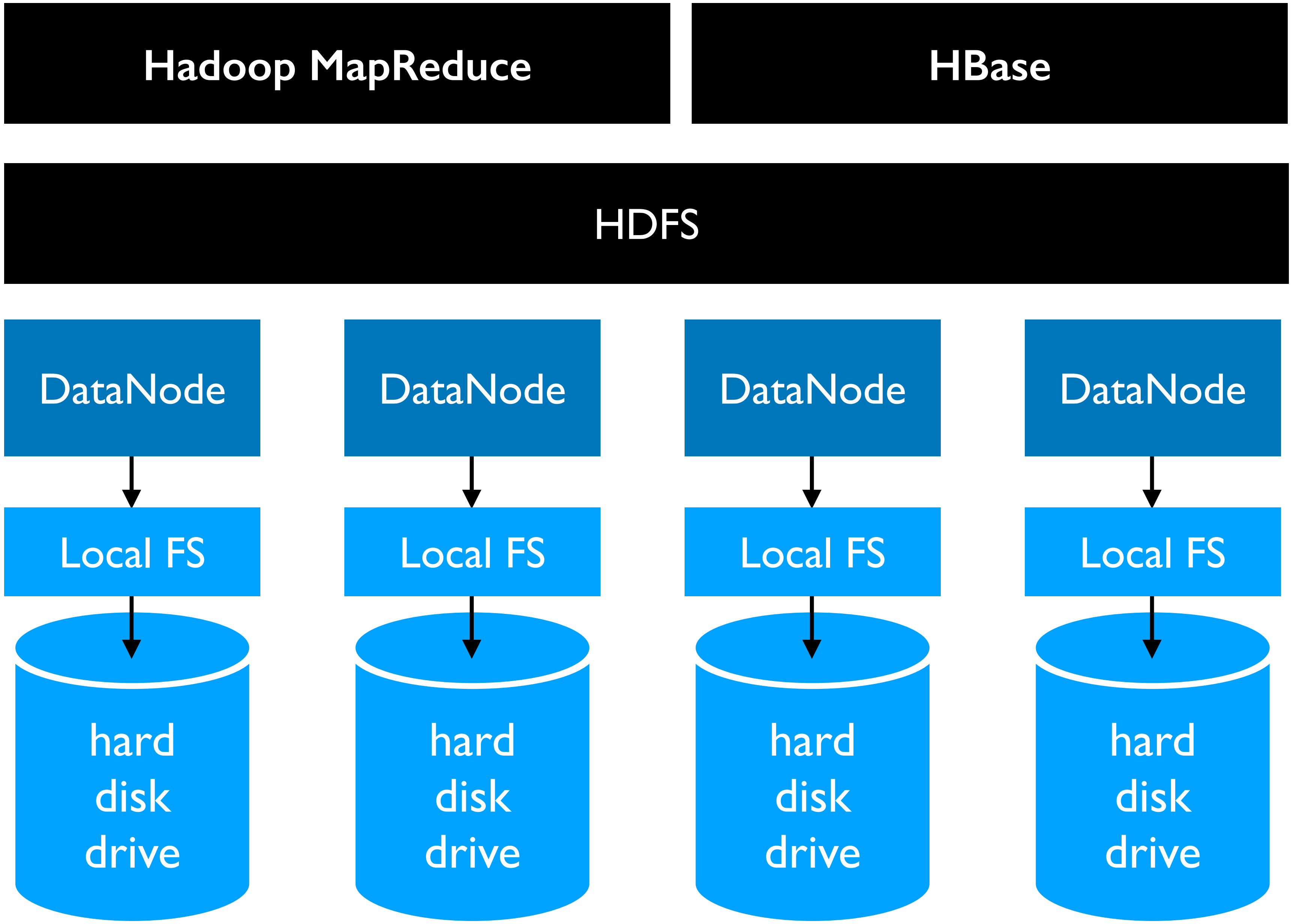
MapReduce (2004 paper)

BigTable (2006 paper)

GFS: Google File System (2003 paper)

radical idea: base everything on lots of cheap, commodity hardware

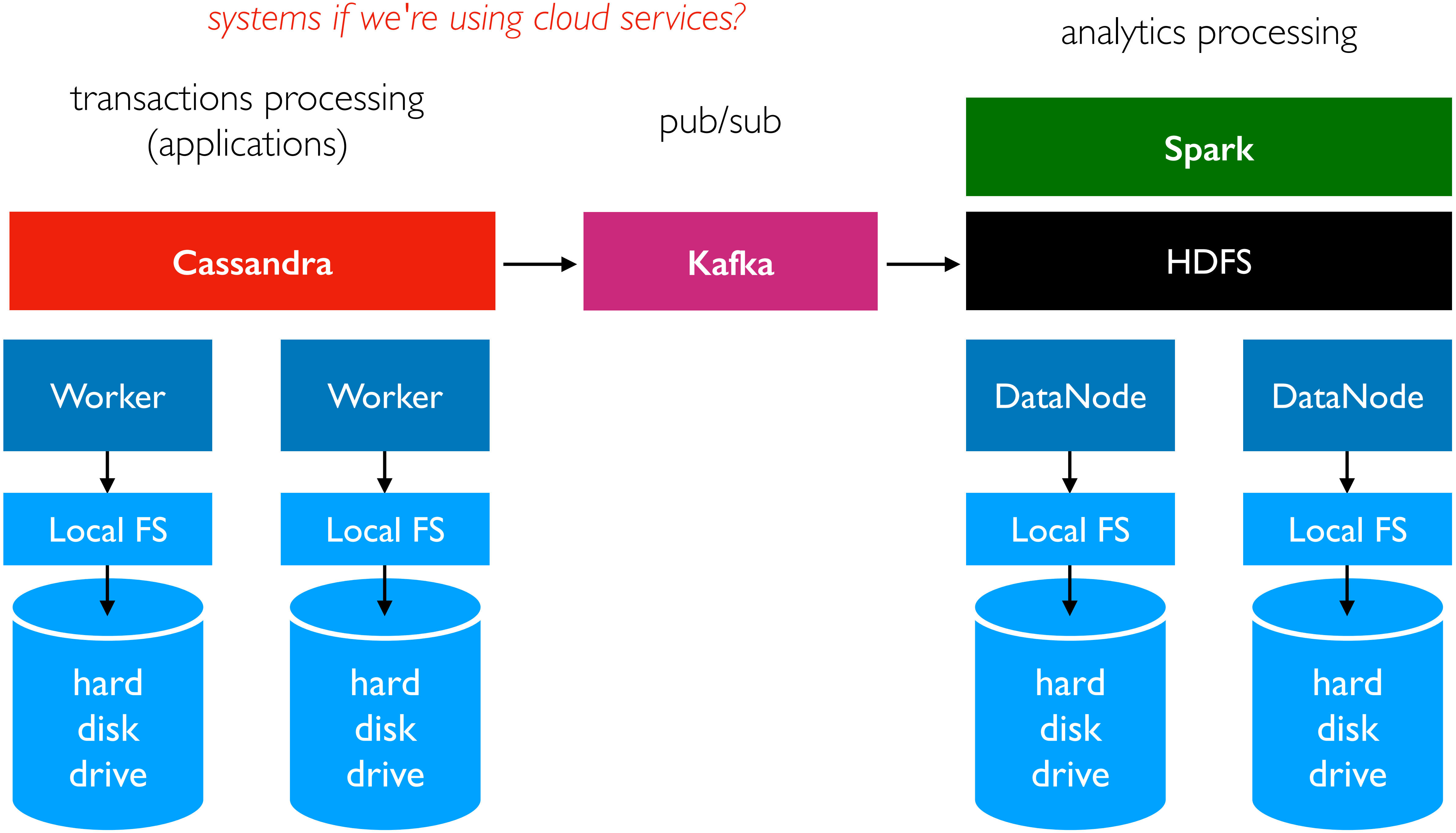
Google (Papers) => Hadoop (open-source software)



Systems both within the Google ecosystem and Hadoop ecosystem have been evolving a LOT.

HDFS - Spark - Cassandra - Kafka

what might we use instead of these Hadoop systems if we're using cloud services?

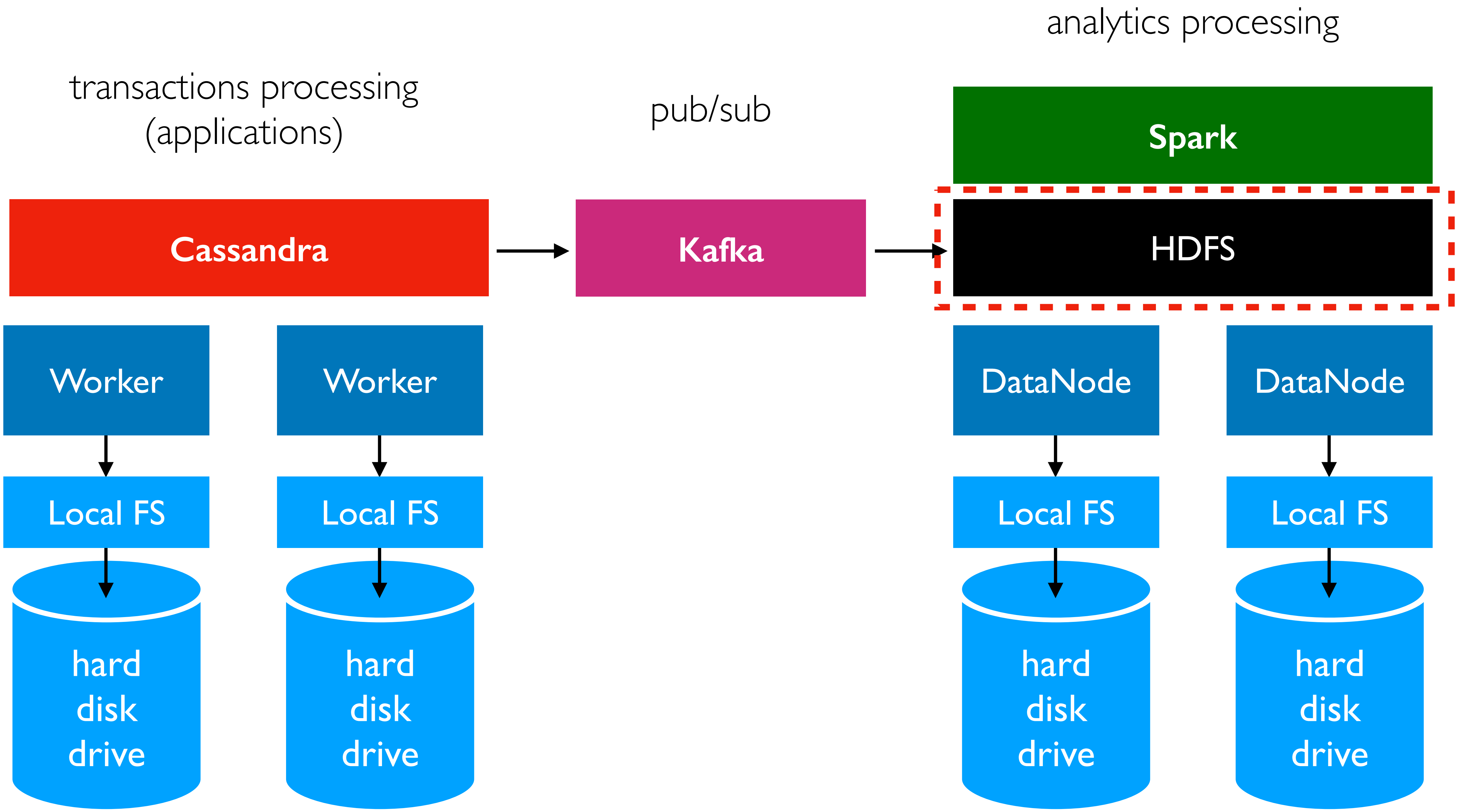


major systems we used this semester

(this shows one possible way they could relate to each other)

HDFS - Spark - Cassandra - Kafka

Cloud: Colossus

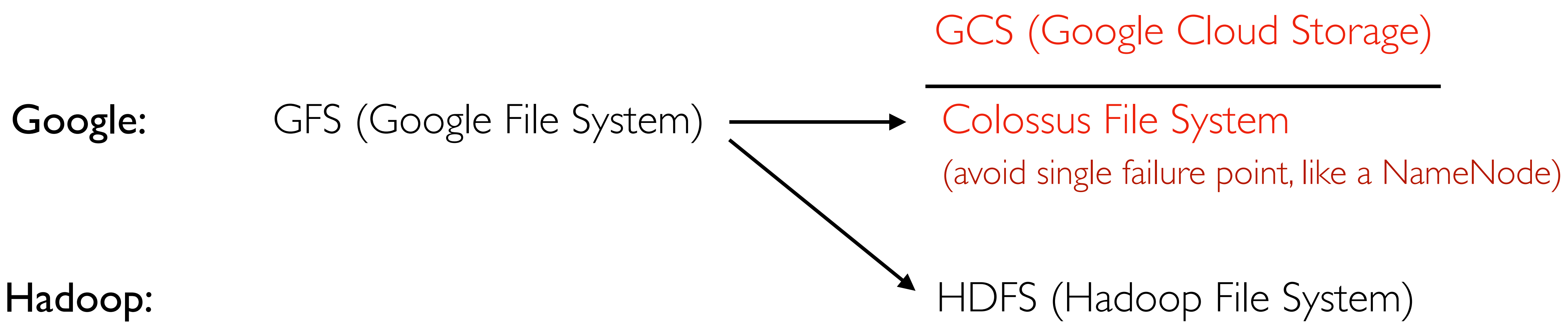


major systems we used this semester

(this shows one possible way they could relate to each other)

HDFS - Spark - Cassandra - Kafka

Cloud: Colossus



GCS (Google Cloud Storage)

Colossus File System

(avoid single failure point, like a NameNode)

Google: GFS (Google File System)

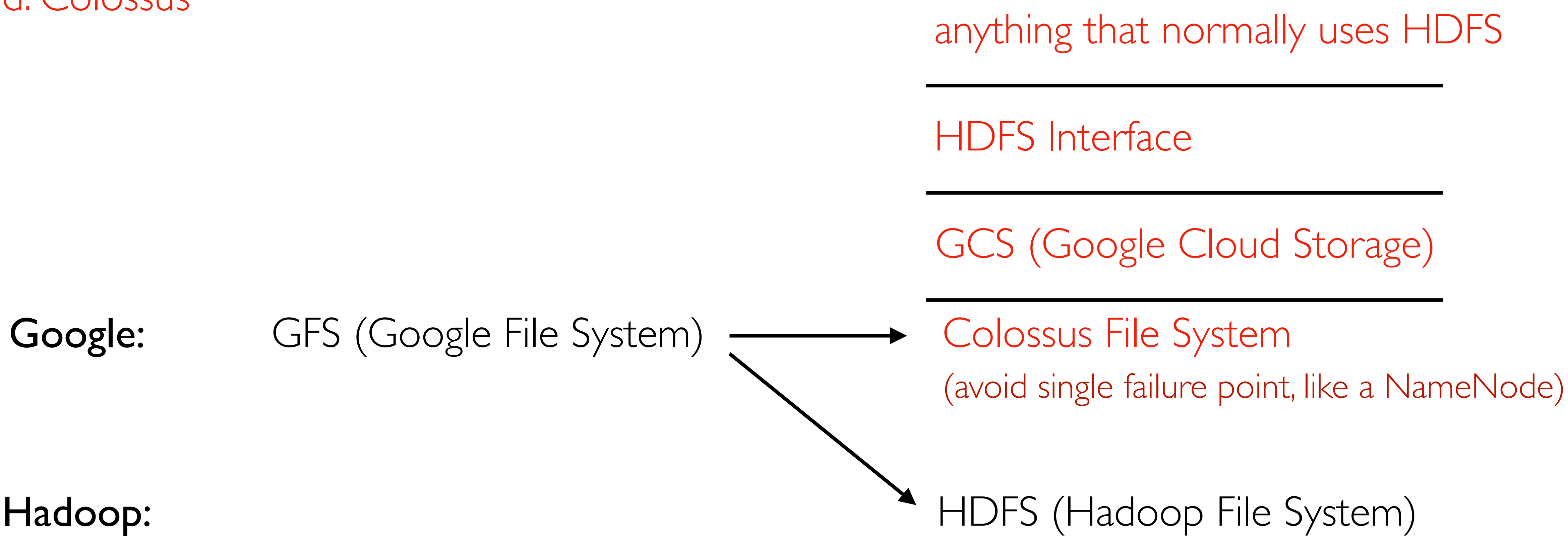
Hadoop: HDFS (Hadoop File System)

Colossus is indirectly available to customers via GCS and other services

- users can create GCS "buckets" containing "objects" (corresponding to files in Colossus)
- buckets can be public or private

HDFS - Spark - Cassandra - Kafka

Cloud: Colossus



anything that normally uses HDFS

HDFS Interface

GCS (Google Cloud Storage)

Google: GFS (Google File System) → Colossus File System (avoid single failure point, like a NameNode)

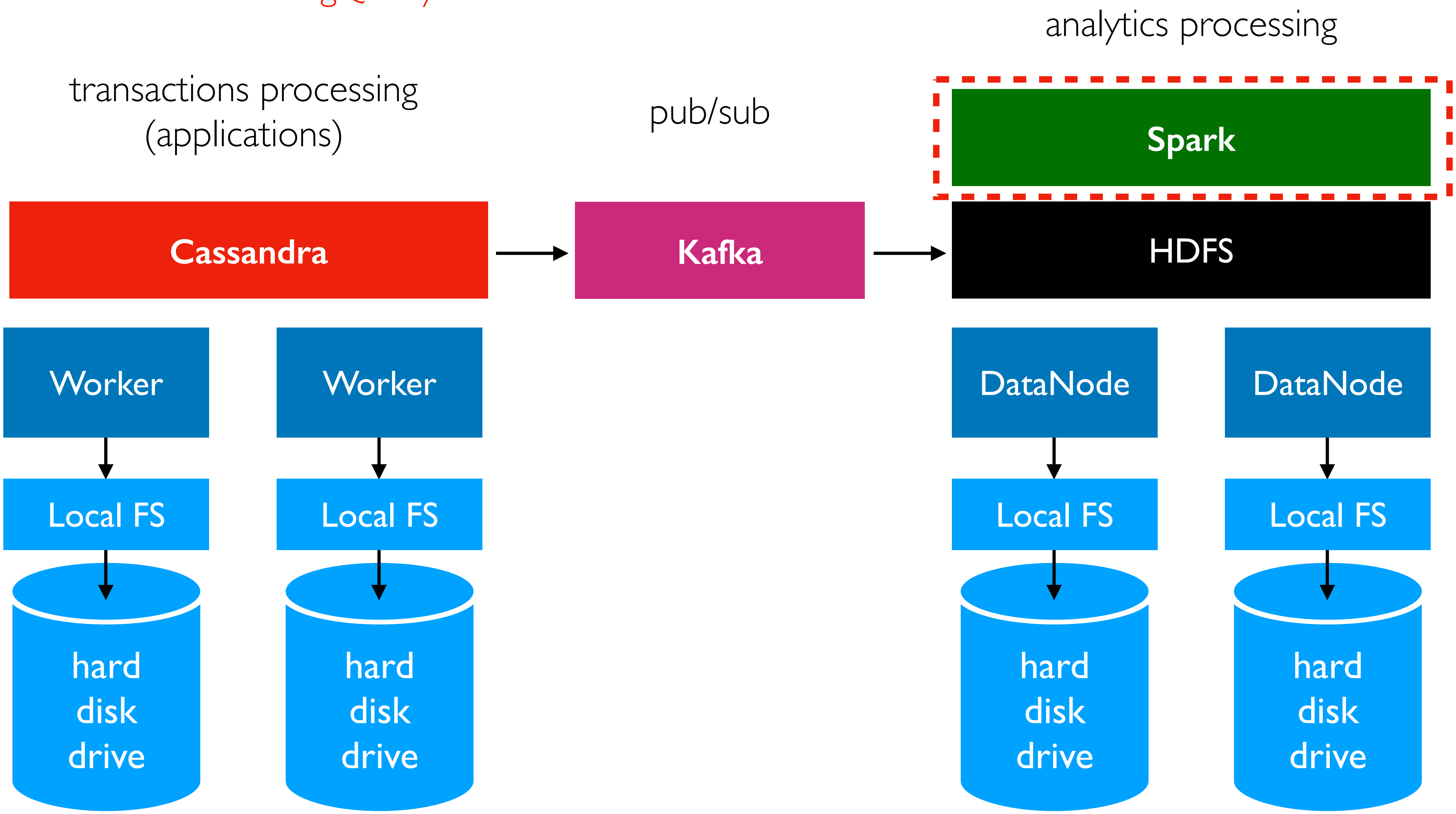
Hadoop: HDFS (Hadoop File System)

Colossus is indirectly available to customers via GCS and other services

- users can create GCS "buckets" containing "objects" (corresponding to files in Colossus)
- buckets can be public or private
- GCS connector for Hadoop implements HDFS interface over GCS (https://github.com/GoogleCloudDataproc/hadoop-connectors/tree/master/gcs)
- Applications (like Spark) can switch out HDFS for GCS

HDFS - Spark - Cassandra - Kafka

Cloud: BigQuery

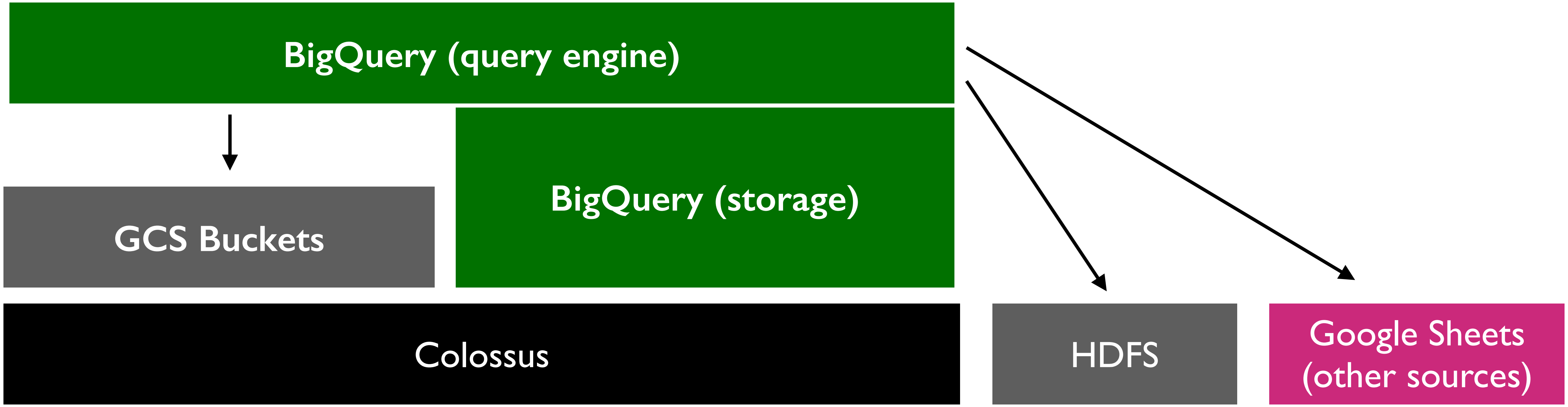


major systems we used this semester

(this shows one possible way they could relate to each other)

HDFS - Spark - Cassandra - Kafka

Cloud: BigQuery

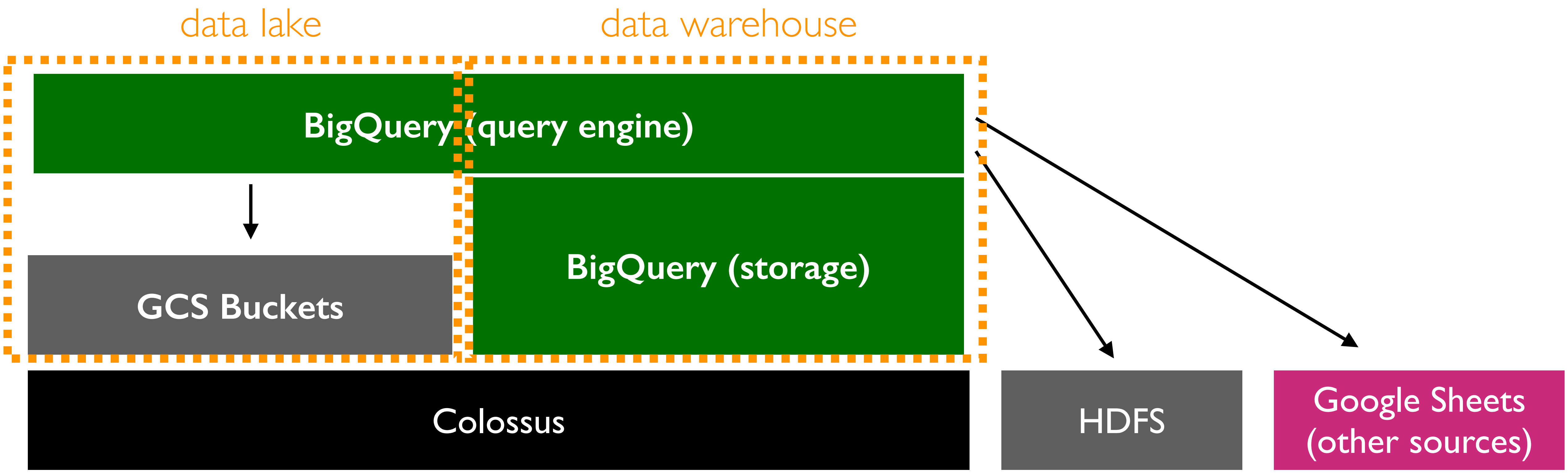


BigQuery

- similar to Spark SQL
- query engine based on Dremel (2010 system in Google that replaced a lot of MapReduce work)
- tightly integrated with BigQuery storage engine (that uses Colossus)
- can also run queries on other data sources

HDFS - Spark - Cassandra - Kafka

Cloud: BigQuery

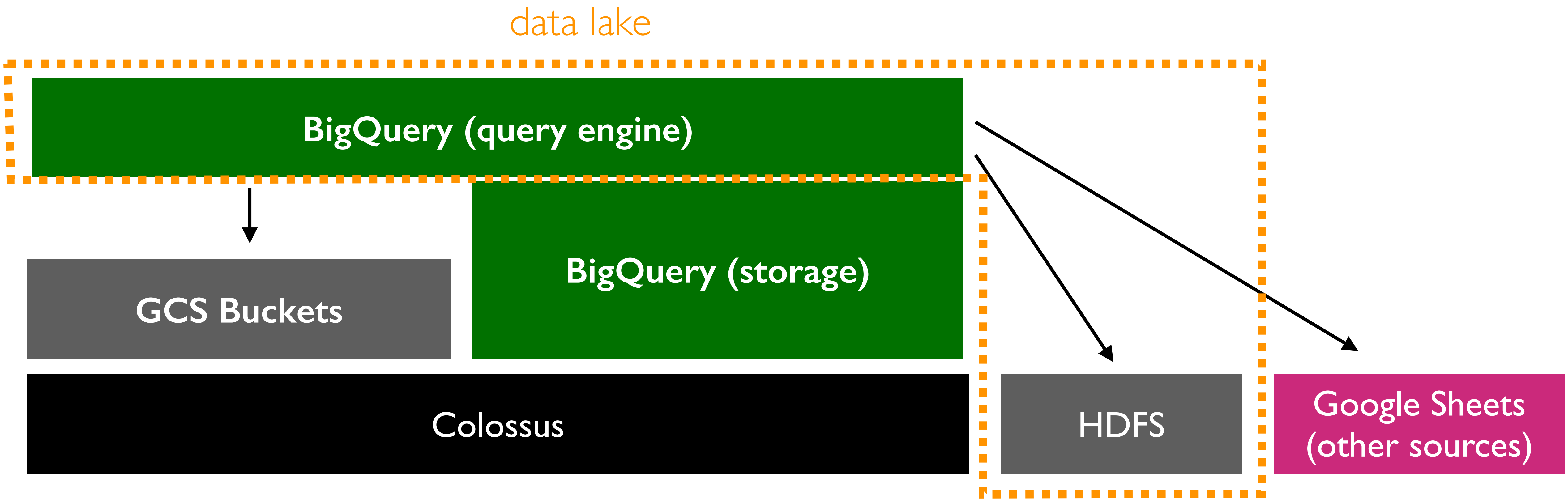


Blurred analytics architecture

- **data warehouse:** BigQuery query engine with BigQuery storage
- **data lake:** part of BigQuery used with another system

HDFS - Spark - Cassandra - Kafka

Cloud: BigQuery

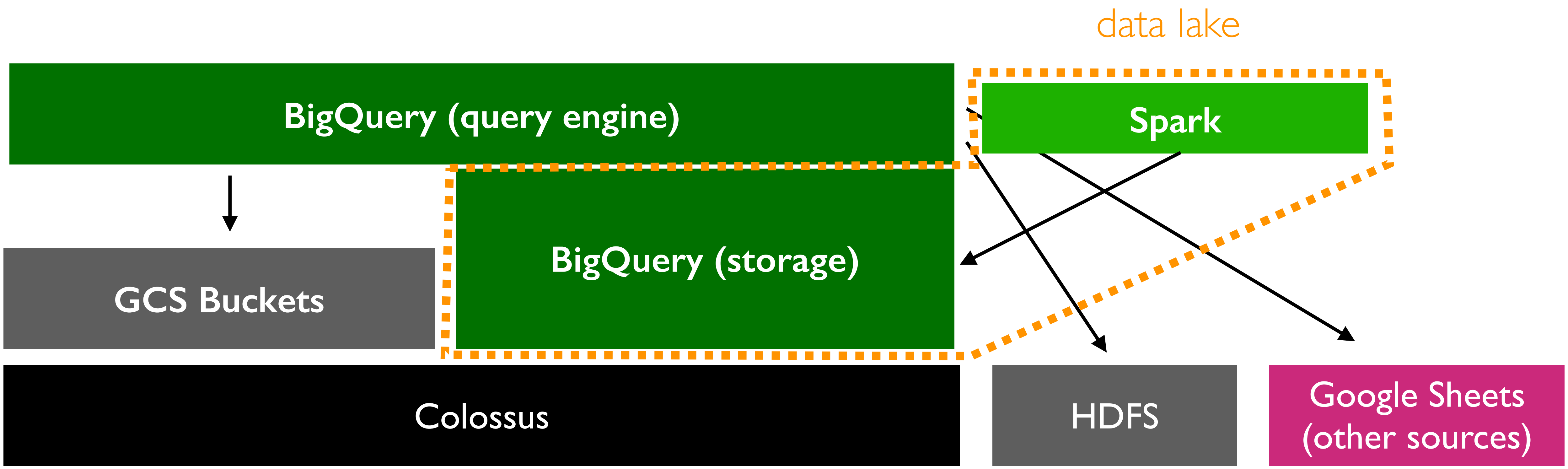


Blurred analytics architecture

- **data warehouse:** BigQuery query engine with BigQuery storage
- **data lake:** part of BigQuery used with another system

HDFS - Spark - Cassandra - Kafka

Cloud: BigQuery

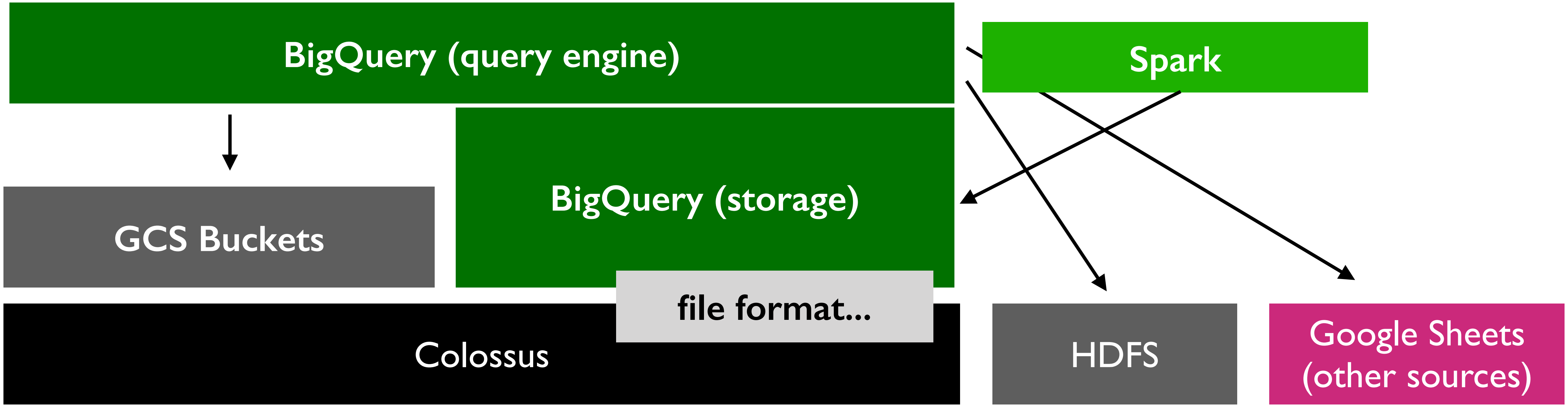


Blurred analytics architecture

- **data warehouse:** BigQuery query engine with BigQuery storage
- **data lake:** part of BigQuery used with another system

HDFS - **Spark** - Cassandra - Kafka

Cloud: BigQuery

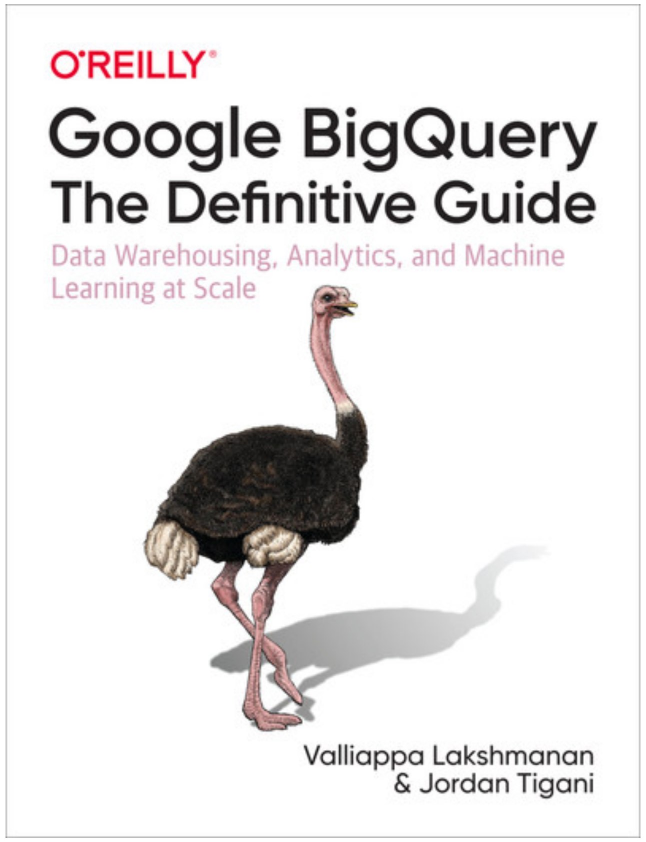


for analytics, we'll want a column-oriented format...

- Parquet
- ColumnIO
- Capacitor

HDFS - Spark - Cassandra - Kafka

Cloud: BigQuery


O'REILLY

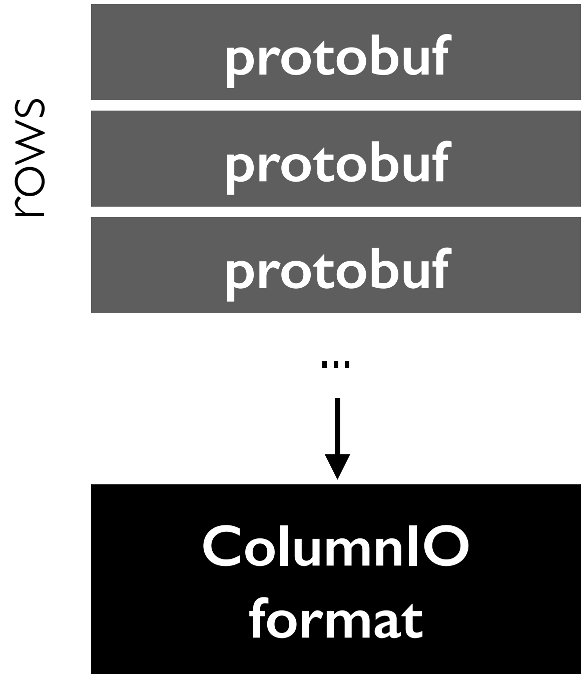
Google BigQuery
The Definitive Guide

Data Warehousing, Analytics, and Machine Learning at Scale

Valliappa Lakshmanan
& Jordan Tigani

protocol buffers (protobufs)

- some protobufs at Google had grown to have 100s of thousands of columns
- OK for applications/logging, horrible for analysis


rows

protobuf

protobuf

protobuf

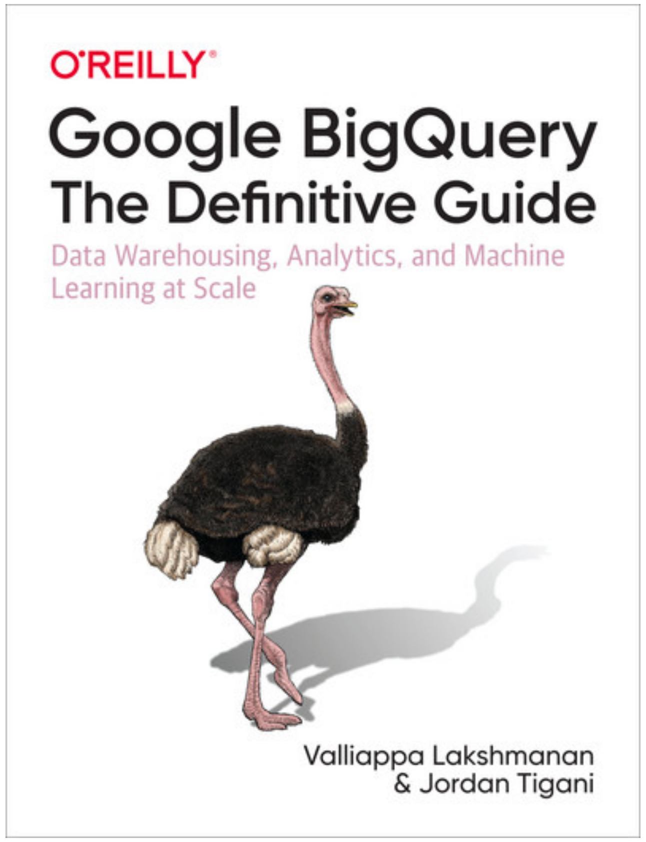
...

ColumnIO format

col-oriented format

HDFS - Spark - Cassandra - Kafka

Cloud: BigQuery



O'REILLY

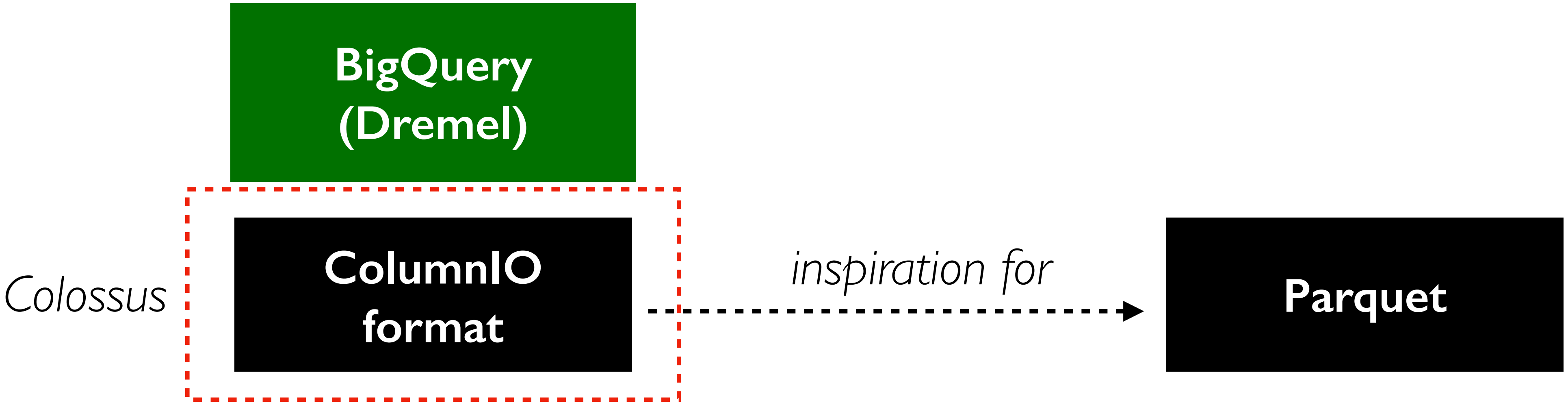
Google BigQuery
The Definitive Guide

Data Warehousing, Analytics, and Machine Learning at Scale

Valliappa Lakshmanan
& Jordan Tigani

protocol buffers (protobufs)

- some protobufs at Google had grown to have 100s of thousands of columns
- OK for applications/logging, horrible for analysis



BigQuery (Dremel)

Colossus

ColumnIO format

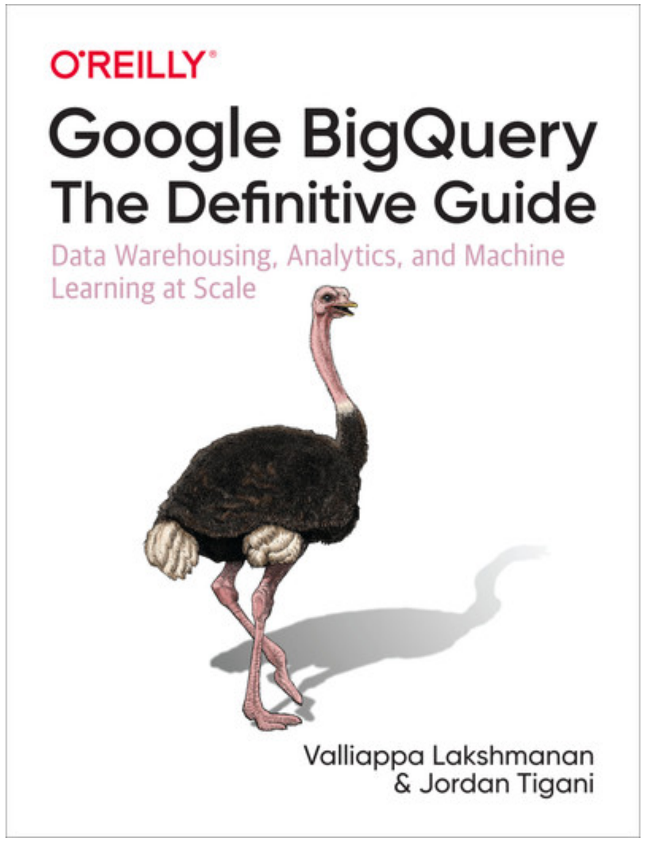
inspiration for

Parquet

- Dremel (used in BigQuery) originally used ColumnIO files
- ColumnIO inspired Parquet files (introduced by Twitter+Cloudera) https://blog.twitter.com/engineering/en_us/a/2013/dremel-made-simple-with-parquet

HDFS - **Spark** - Cassandra - Kafka

Cloud: BigQuery



O'REILLY

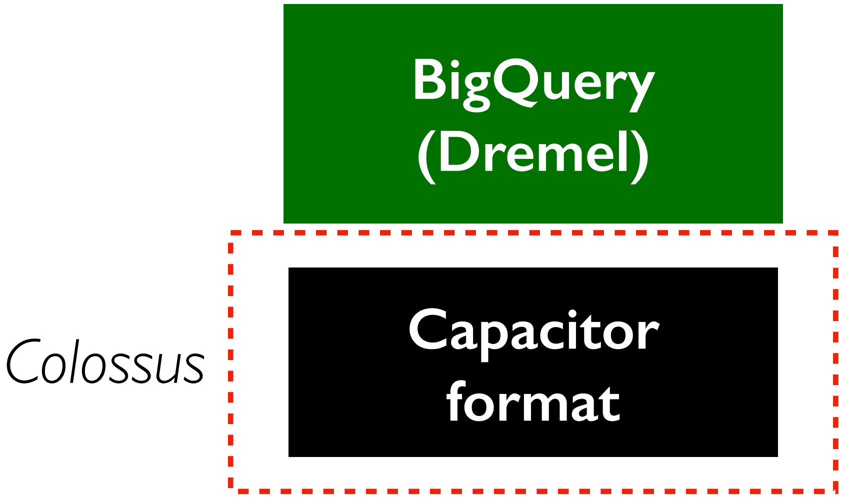
Google BigQuery
The Definitive Guide

Data Warehousing, Analytics, and Machine Learning at Scale

Valliappa Lakshmanan
& Jordan Tigani

Capacitor Format

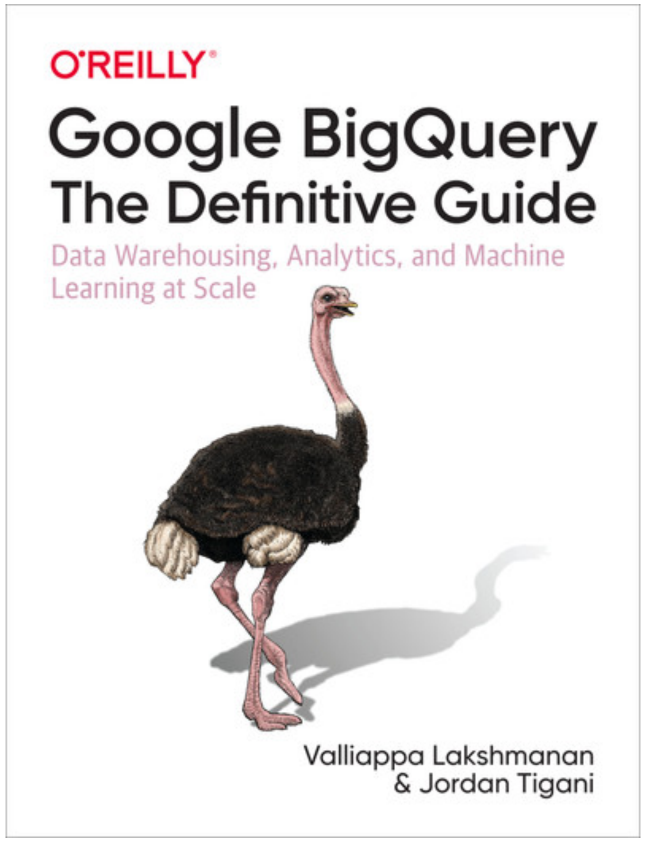
- sucessor to ColumnIO in Google
- optimized for repeated values



column:
apple
apple
apple
banana
banana
apple
apple
apple
apple

HDFS - **Spark** - Cassandra - Kafka

Cloud: BigQuery



O'REILLY

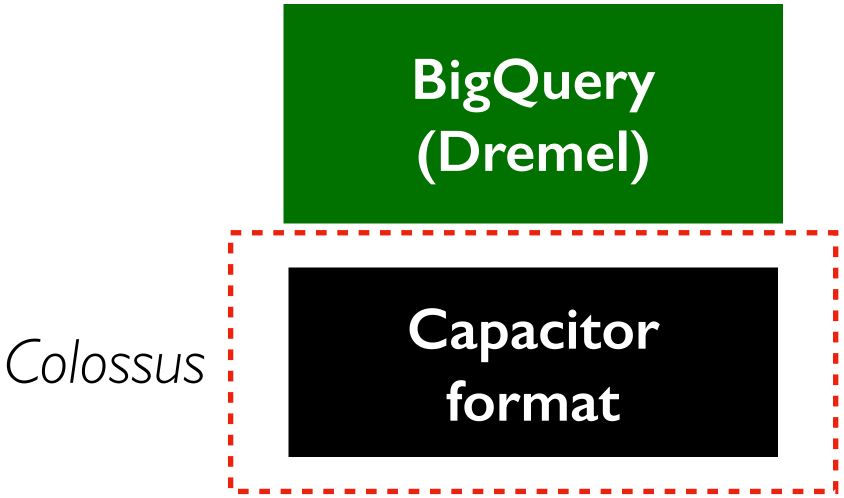
Google BigQuery
The Definitive Guide

Data Warehousing, Analytics, and Machine Learning at Scale

Valliappa Lakshmanan
& Jordan Tigani

Capacitor Format

- sucessor to ColumnIO in Google
- optimized for repeated values

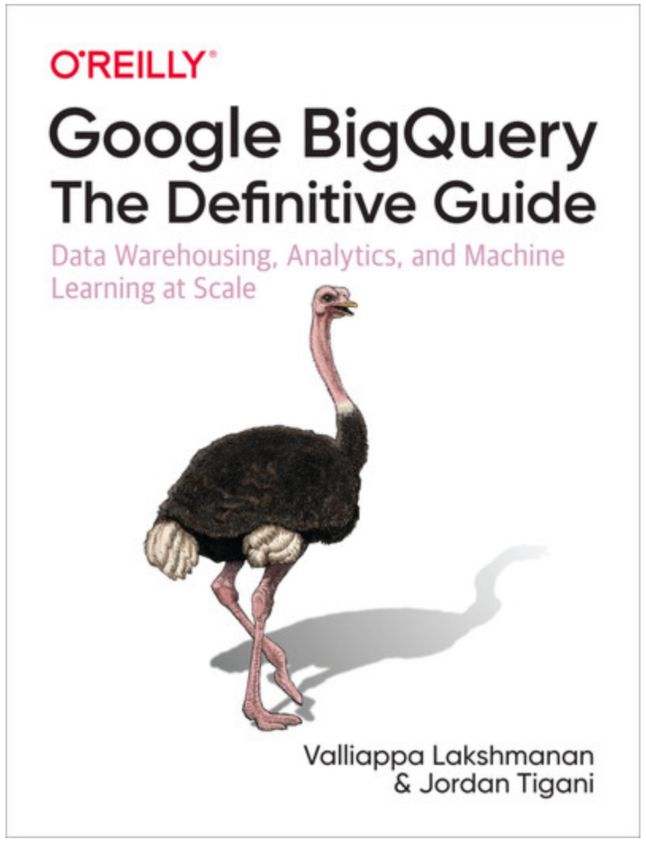


column:
3: apple
2: banana
4: apple

optimization: run-length encoding

HDFS - **Spark** - Cassandra - Kafka

Cloud: BigQuery



O'REILLY

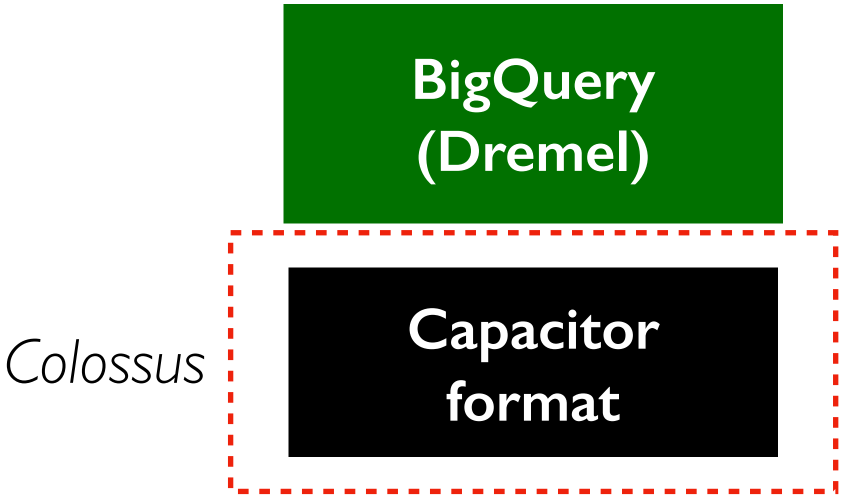
Google BigQuery
The Definitive Guide

Data Warehousing, Analytics, and Machine Learning at Scale

Valliappa Lakshmanan
& Jordan Tigani

Capacitor Format

- sucessor to ColumnIO in Google
- optimized for repeated values



column:
3: 1
2: 2
4: 1

{"apple":1 "banana": 2}

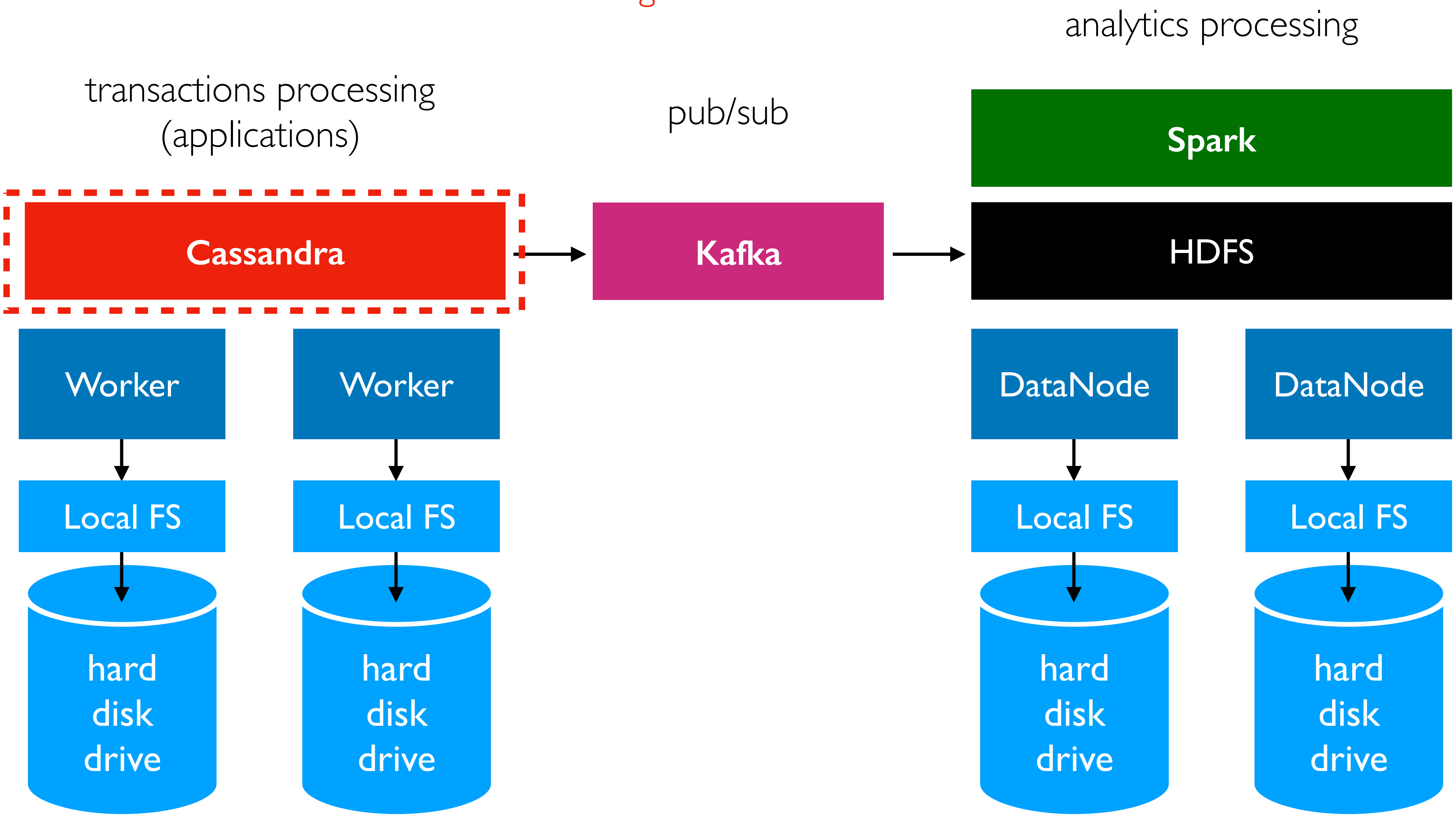
optimization: run-length encoding

optimization: dictionary encoding

TopHat

HDFS - Spark - Cassandra - Kafka

Cloud: BigTable

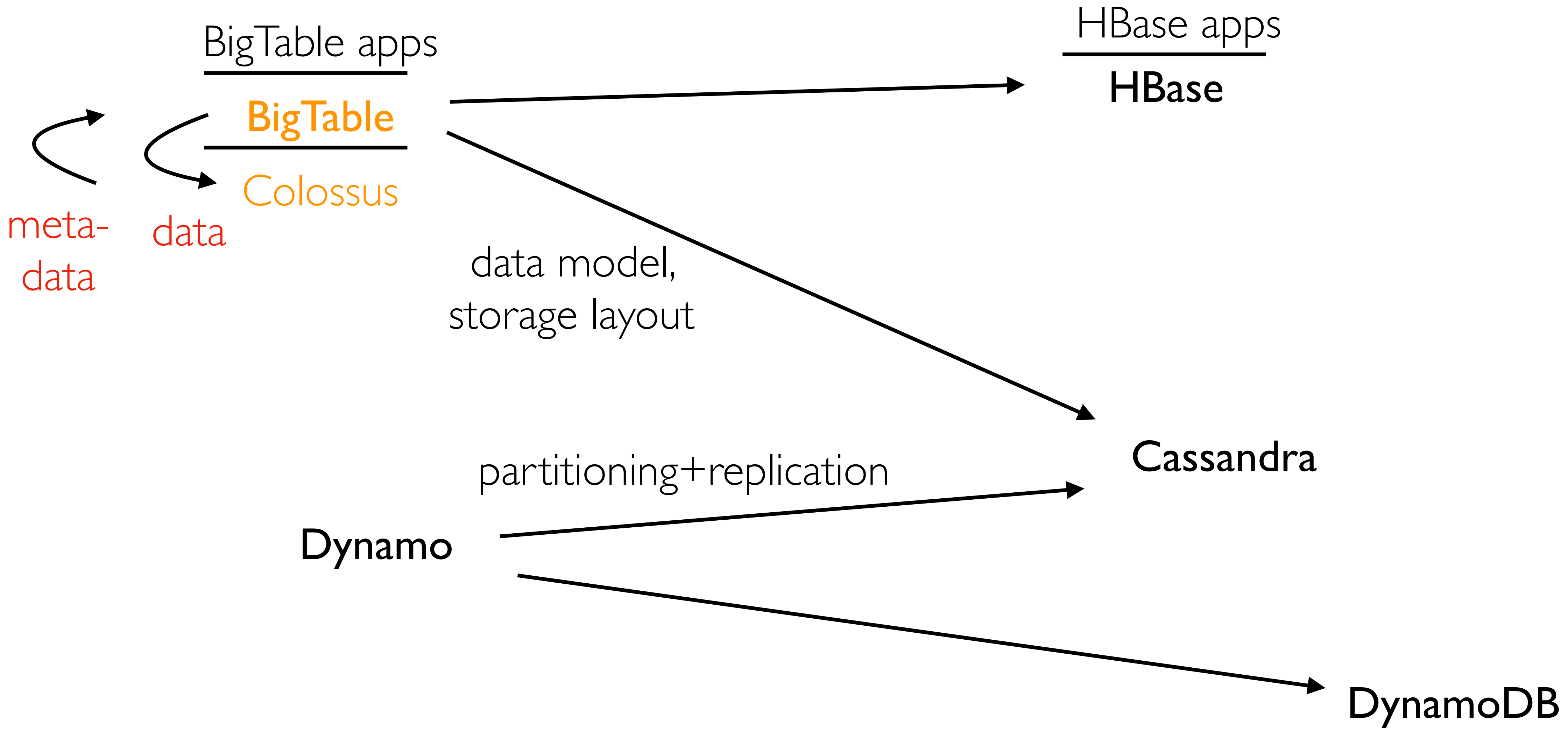


major systems we used this semester

(this shows one possible way they could relate to each other)

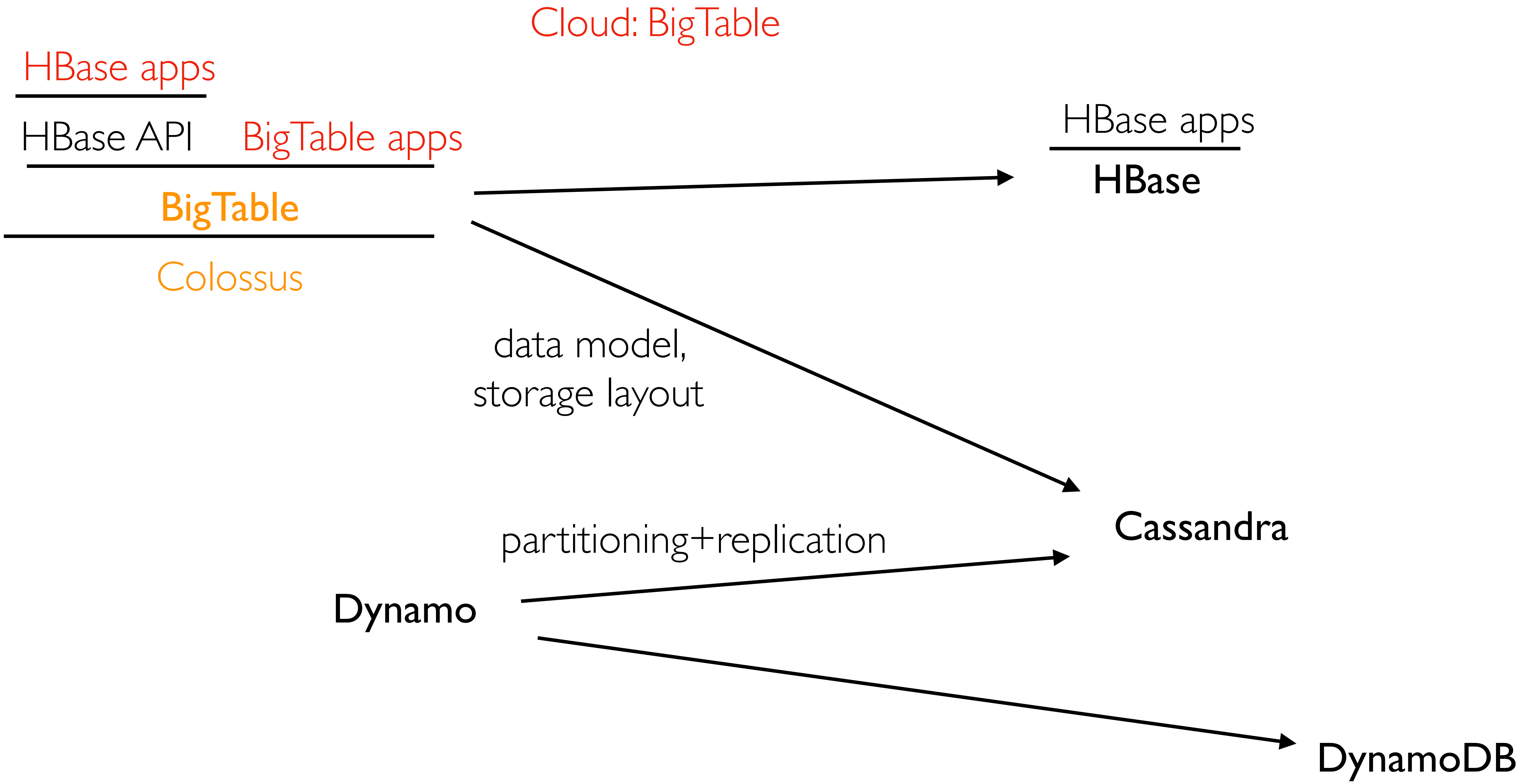
HDFS - Spark - Cassandra - Kafka

Cloud: BigTable



- BigTable is directly available to customers as a GCP service
- It's now built on Colossus. "The original motivation for building Colossus was to solve scaling limits we experienced with Google File System (GFS) when trying to accommodate metadata related to Search. Storing file metadata in BigTable allowed Colossus to scale up by over 100x over the largest GFS clusters." (https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system)

HDFS - Spark - Cassandra - Kafka



Cloud: BigTable

HBase apps

HBase API   BigTable apps

BigTable

Colossus

HBase apps

HBase

data model,
storage layout

Cassandra

partitioning+replication

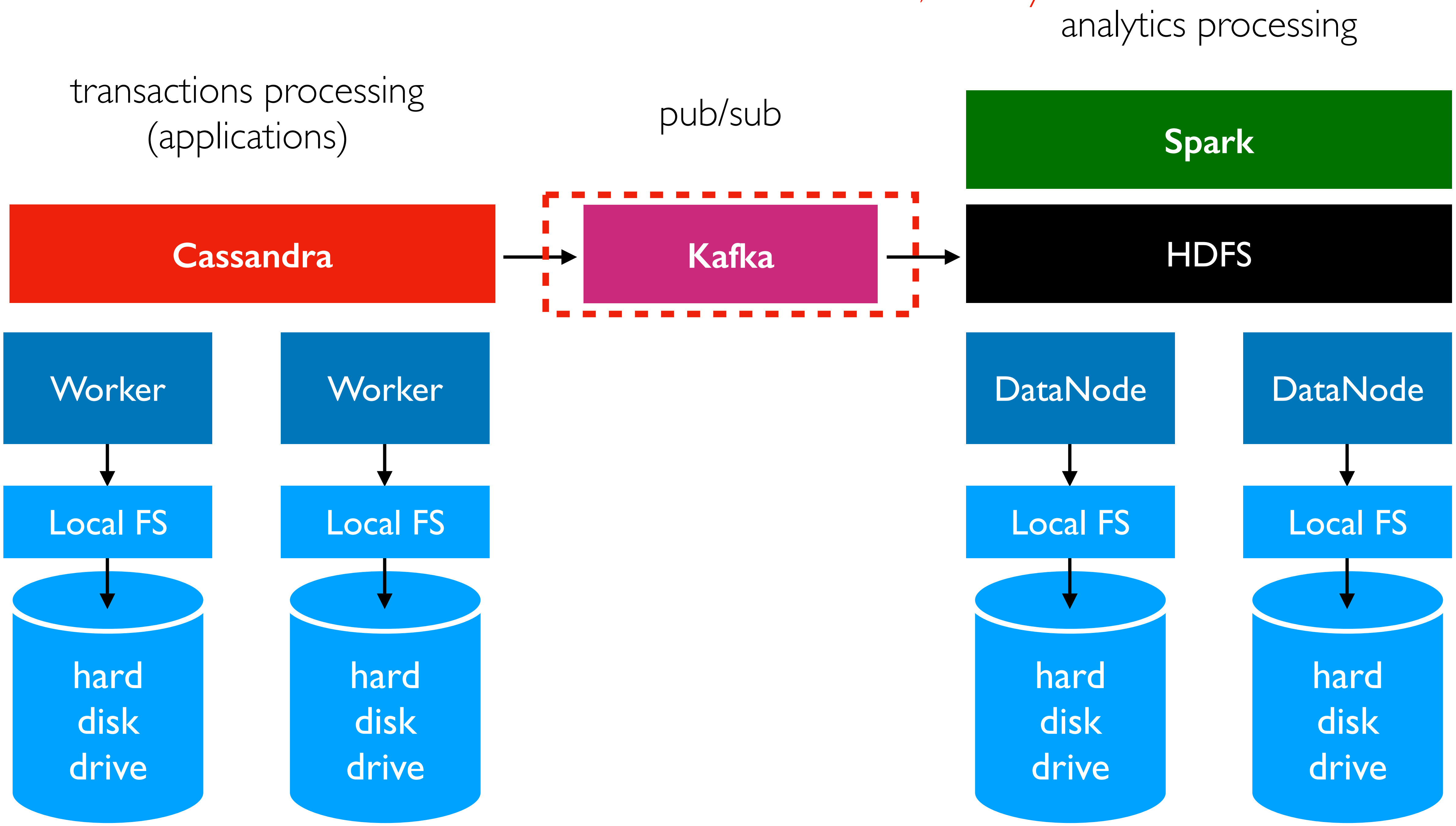
Dynamo

DynamoDB

- Some apps can directly use the BigTable API
- BigTable also supports the similar HBase API now
  (presumably to bring back HBase users who don't want the hassle of deployment, or of re-writing their code to use a managed cloud service)

HDFS - Spark - Cassandra - Kafka

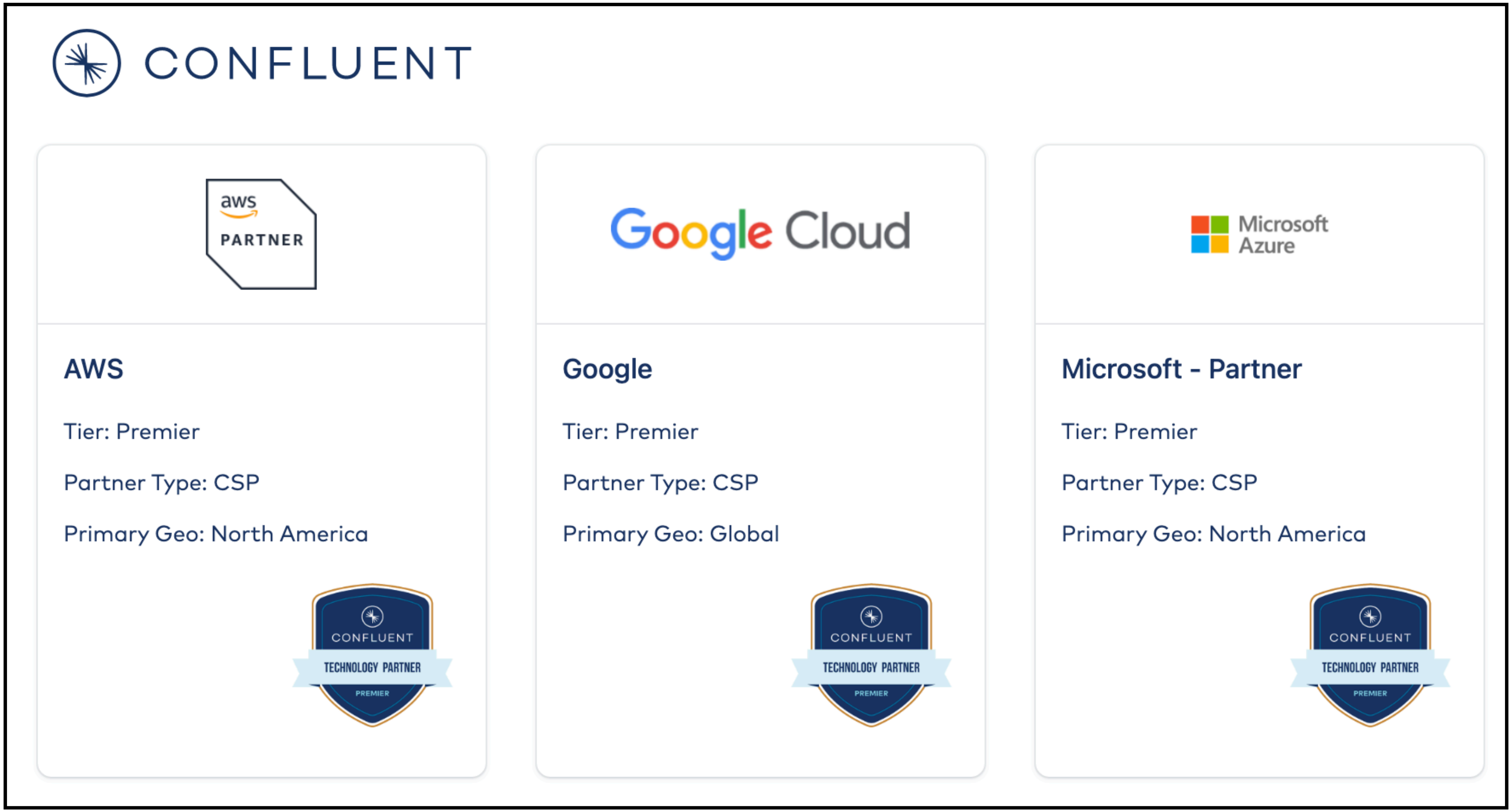
Cloud: Kafka, actually



major systems we used this semester

(this shows one possible way they could relate to each other)

HDFS - Spark - Cassandra - Kafka



CONFLUENT

AWS

Tier: Premier

Partner Type: CSP

Primary Geo: North America

Google

Tier: Premier

Partner Type: CSP

Primary Geo: Global

Microsoft - Partner

Tier: Premier

Partner Type: CSP

Primary Geo: North America

https://partners.confluent.io/English/directory/search?f0=Partner+Type&f0v0=CSP

- Apache Kafka - open source
- Confluent Kafka - closed source, more features, available as service is the major cloud providers

Aside: Open Source Software and Business Models

Open-Source Licenses (very rough overview -- I'm not a lawyer!)

- GPL: if you make improvements and sell/distribute the software, your code needs to be made open source too
- MIT+BSD: fine to take open source code, make closed-source improvements, and sell a product based on it. Minimal requirements (e.g., related to attribution, liability)
- Apache: similar to MIT and BSD, but relates to patents (not just copyright).

All the major systems we have learned this semester (HDFS, Spark, Cassandra, Kafka) are distributed under the Apache license. Thus, it is possible to build companies around closed-source variants of these systems. Examples:

- Databricks (Spark)
- Datastax (Cassandra)
- Confluent (Kafka)

Conclusions

Cloud keeps increasing in importance

- total global revenue
- cloud providers
- number of services for each provider

Compute, memory, storage, and network resources are all rentable.

Even though you pay a markup to the cloud provider, it's often cheaper than owning your own hardware if your usage fluctuates a lot and most resources are idle during low times.

[544] BigQuery: Complex Types

Meenakshi Syamkumar

Learning Objectives

- authenticate on a GCP VM to gain access to BiqQuery datasets
- execute BiqQuery queries in a variety of settings (console, Jupyter with extension, Python call)
- query complex data types (arrays and structs) using correlated cross joins

Demos...

Outline

Types: Simple and Arrays/Structs

Cross Joining

Unnesting, Correlated Cross Join

Geographic Data

Types

Basics

- BOOL, INT64, FLOAT64
- STRING, BYTES
- DATE, DATETIME
- etc.

Nesting

- **ARRAY** (repeated):
  `myarray[OFFSET(5)]`
- **STRUCT** (record)
  `mystruct.some_attribute`

example from https://cloud.google.com/bigquery/docs/nested-repeated

```
+-------------------+--------------------------------+------------+
| title             | authors                        | num_pages  |
+-------------------+--------------------------------+------------+
| Example Book One  | [{123, Alex, 01-01-1960},      | 487        |
|                   |  {789, Kim, 01-01-1980}]       |            |
| Example Book Two  | [{456, Rosario, 01-01-1970}]   | 89         |
+-------------------+--------------------------------+------------+
```

array of structs

struct

Outline

Types: Simple and Arrays/Structs

Cross Joining

Unnesting, Correlated Cross Join

Geographic Data

Cross Joins

Previously covered JOIN types (they each specify "ON" to filter row pairings):

- INNER, LEFT, RIGHT

CROSS JOIN: every row in table 1 with every row in table 2

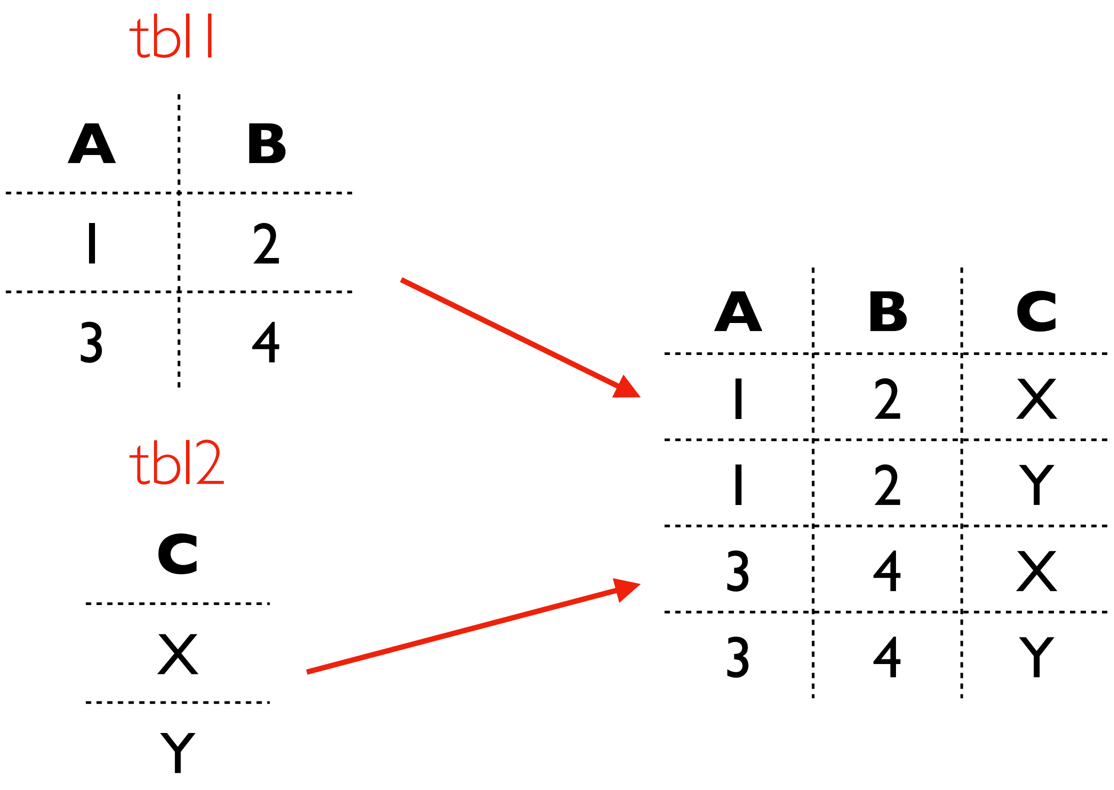
format 1

```
SELECT *
FROM tbl1
CROSS JOIN tbl2
```

format 2

```
SELECT *
FROM tbl1, tbl2
```

same meaning as format 1
(comma means "cross join")



tbl1

A	B
1	2
3	4

tbl2

C
X
Y

A	B	C
1	2	X
1	2	Y
3	4	X
3	4	Y

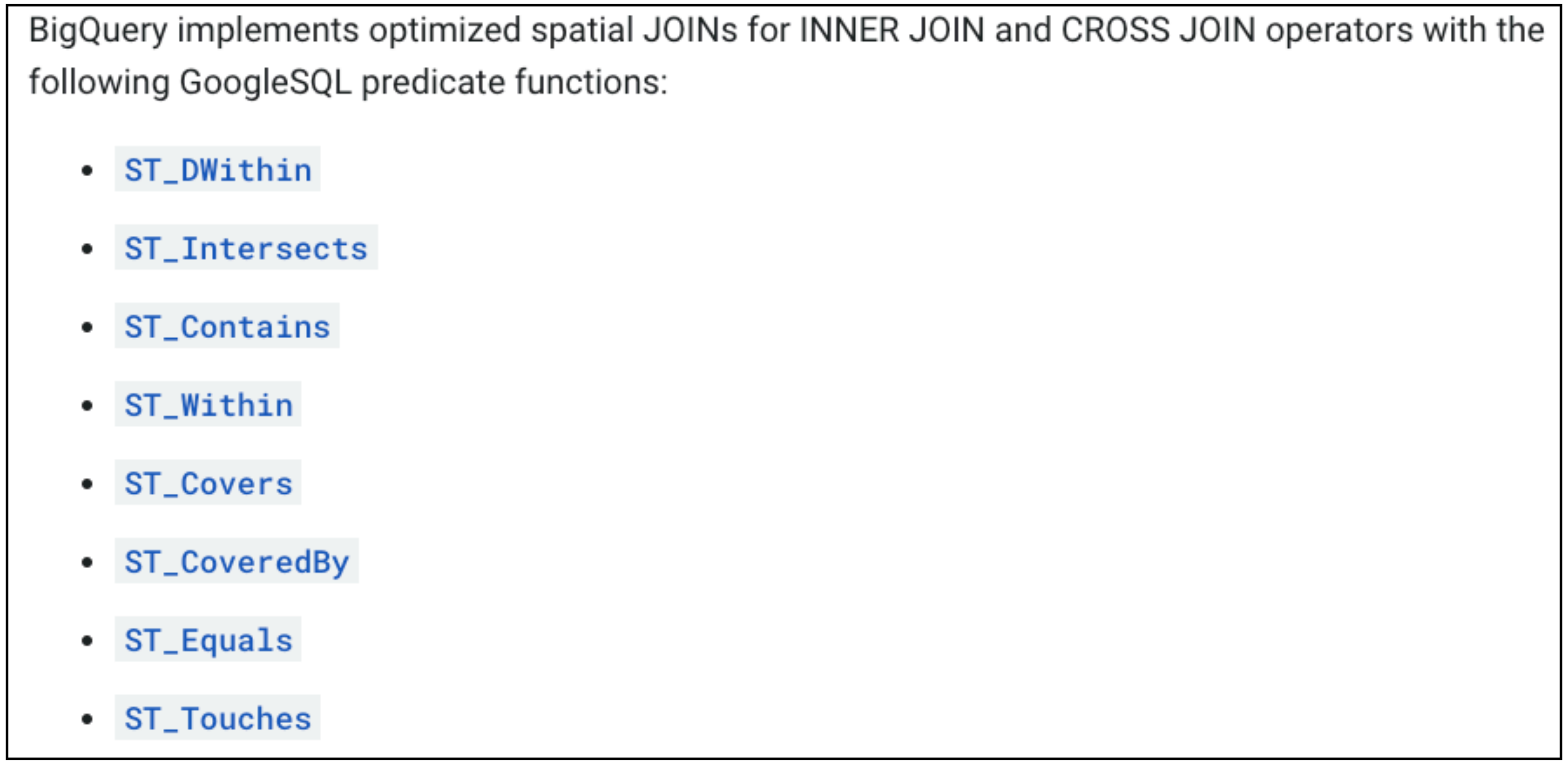
Cross Joins: Filtering

Predicates

- don't use "ON" as in other JOINs
- can optionally use "WHERE"

Naive version: get every combination of pairs, then filter down after.
Can we do better?

Sometimes query engines can optimize certain WHERE filters with CROSS JOIN.



BigQuery implements optimized spatial JOINs for INNER JOIN and CROSS JOIN operators with the following GoogleSQL predicate functions:

- `ST_DWithin`
- `ST_Intersects`
- `ST_Contains`
- `ST_Within`
- `ST_Covers`
- `ST_CoveredBy`
- `ST_Equals`
- `ST_Touches`

BigQuery Documentation

Outline

Types: Simple and Arrays/Structs

Cross Joining

Unnesting, Correlated Cross Join

Geographic Data

Unnesting and Correlated Cross Join

```
SELECT x,y,z
FROM tbl
CROSS JOIN UNNEST(tbl.coord)
```

x	coord
1	[{2,3}, {4,5}]
6	[{7,8}, {9,10}]

unnest

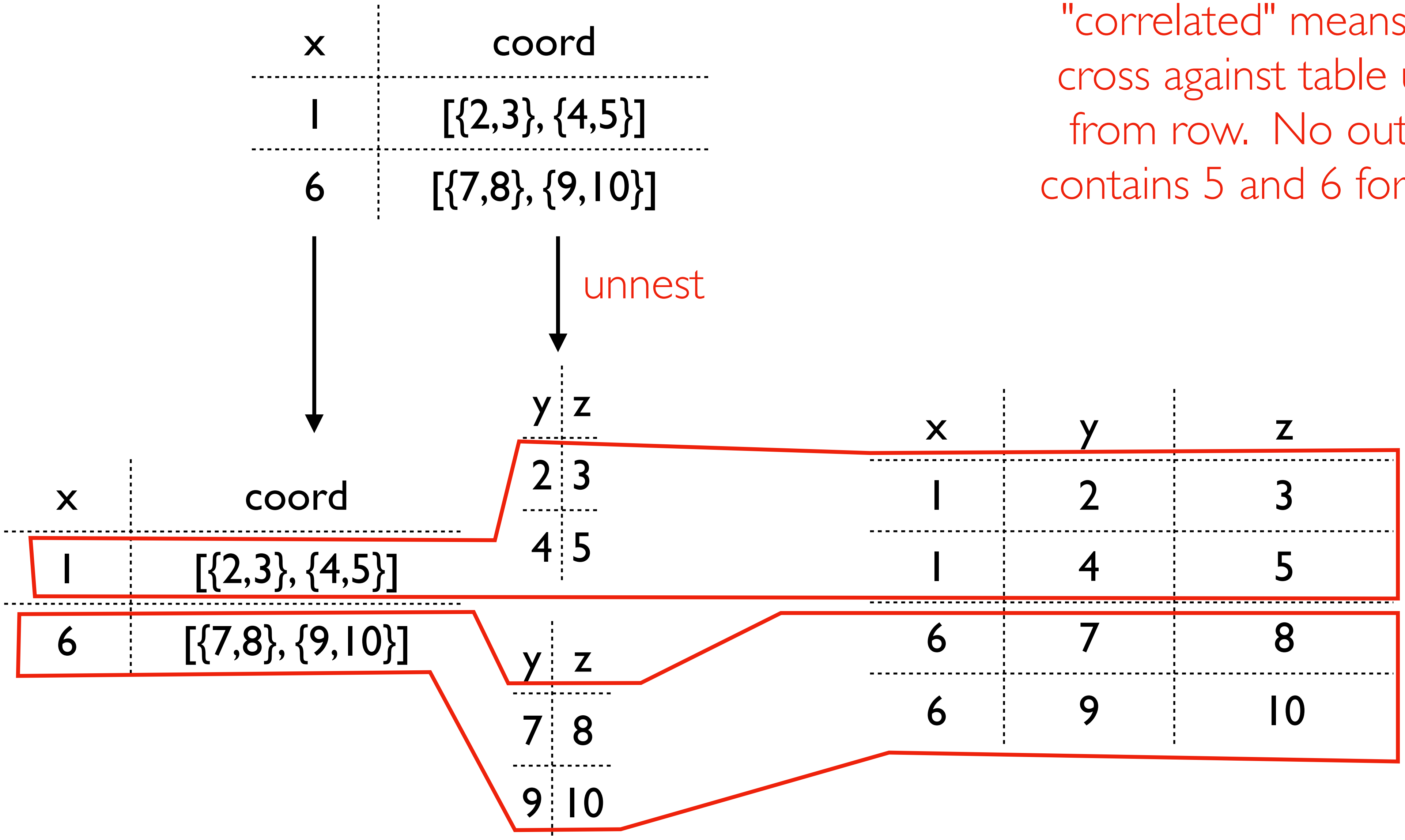
y	z
2	3
4	5

y	z
7	8
9	10

different logical table for each row

Unnesting and Correlated Cross Join

```
SELECT x,y,z
FROM tbl
CROSS JOIN UNNEST(tbl.coord)
```



x	coord
1	[{2,3}, {4,5}]
6	[{7,8}, {9,10}]

x	coord
1	[{2,3}, {4,5}]
6	[{7,8}, {9,10}]

y	z
2	3
4	5

y	z
7	8
9	10

x	y	z
1	2	3
1	4	5
6	7	8
6	9	10

"correlated" means we only cross against table unnested from row. No output row contains 5 and 6 for example.

TopHat

Outline

Types: Simple and Arrays/Structs

Cross Joining

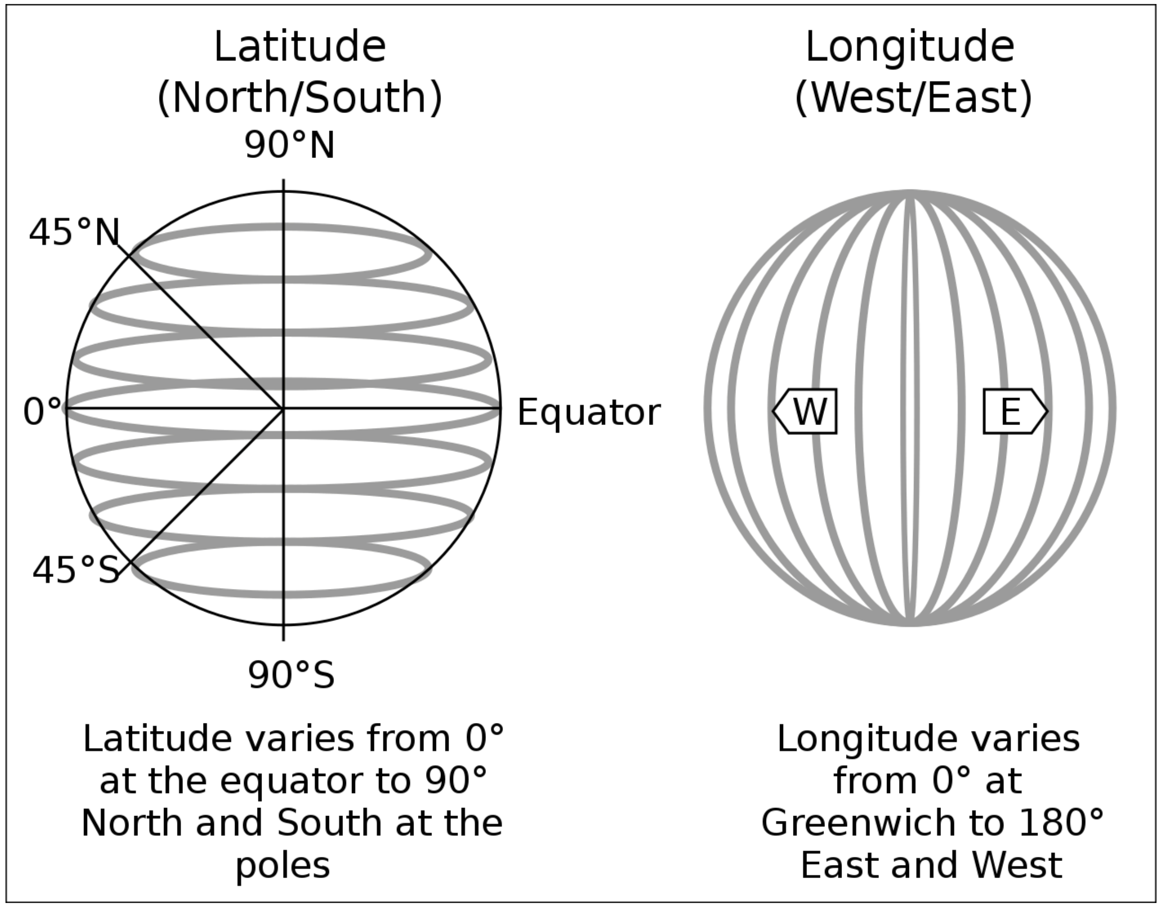
Unnesting, Correlated Cross Join

Geographic Data

Geographic Data

Coordinate reference systems

- way to associate coordinates with a point on earth
- latitude/longitude (used by GPS) is most famous
- some systems incorporate altitude too (3D coordinate system)



Latitude varies from 0° at the equator to 90° North and South at the poles

Longitude varies from 0° at Greenwich to 180° East and West

https://en.wikipedia.org/wiki/Geographic_coordinate_system#/media/File:FedStats_Lat_long.svg

BigQuery support

- common geo operations (e.g., geographic joins)
- uses lat/lon by default (no altitude)

Shape constructors

- ST_GEOGPOINT ●
- ST_MAKELINE
- ST_MAKEPOLYGON △

other shape (e.g., multi-polygons) are possible with operations on these

Demos...