Video Lectures

- Video recordings for lecture and tutorials
  - Canvas: Go to our course, and then look for "Videos/Panopto" > "Web Lectures".
- FYI. Crash course videos.
  - Memory & Storage: Crash Course Computer Science #19 https://www.youtube.com/watch?v=TQCr9RV7twk
  - Operating Systems: Crash Course Computer Science #18 https://www.youtube.com/watch?v=26QPDBe-NB8

CS4225/CS5425 Big Data Systems for Data Science

Principles of Big Data Systems

Bingsheng He
School of Computing
National University of Singapore
hebs@comp.nus.edu.sg

NUS
National University of Singapore
School of Computing

Learning Objectives

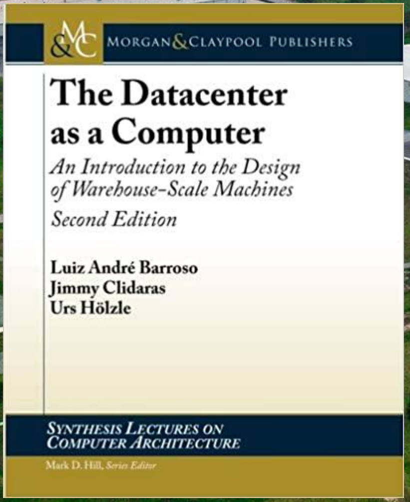
- Learn the storage and memory architectures of data centers as well as the cost of moving data in the data center.
- Understand the four “big Ideas” for building efficient big data systems
- Understand the motivations for the right abstractions of big data systems

Outline

- **Data center architecture**
- The four “Big Ideas”
- Abstractions for big data systems

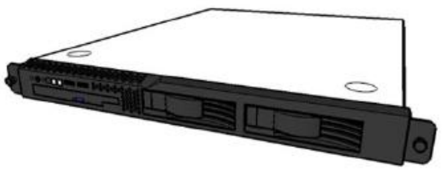
The datacenter *is* the computer!

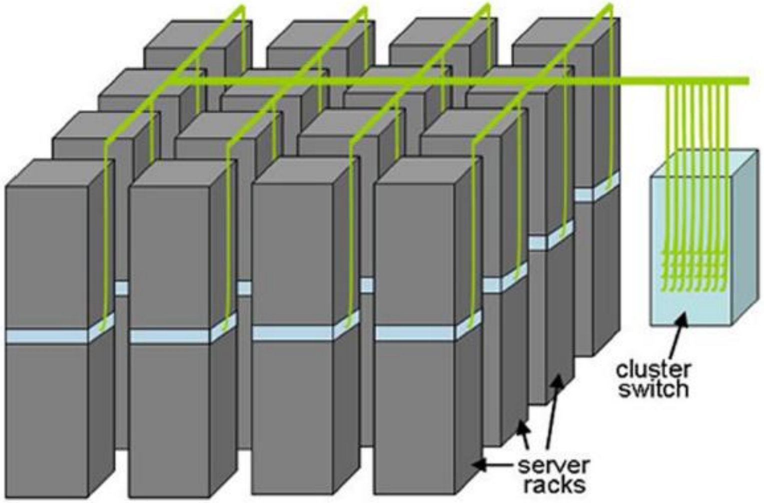
(Meaning: we think of many machines in a data center as a big “processing unit” being used to solve a problem, rather than just as independent machines)



Source: Google

Building Blocks





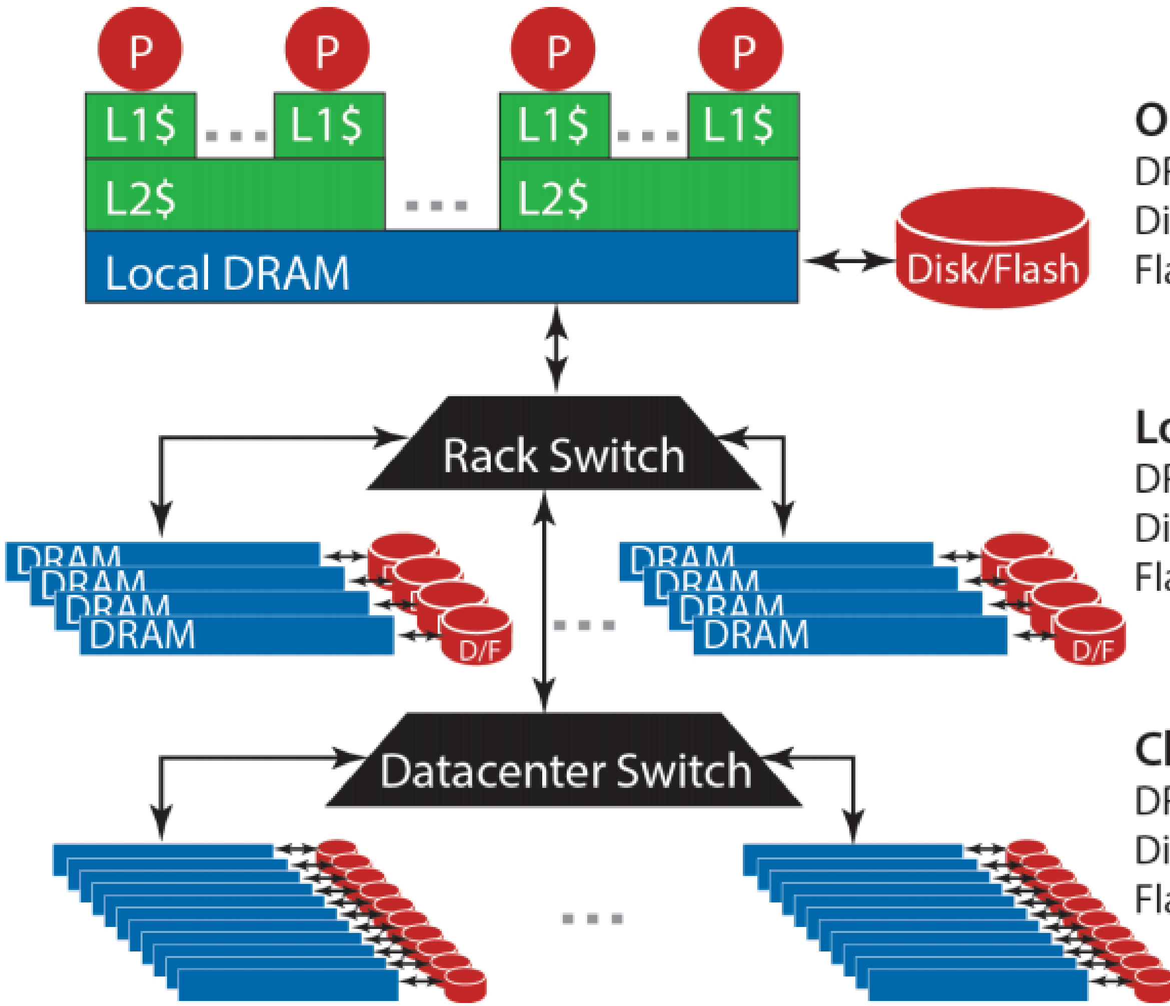
Source: Barroso and Urs Hölzle (2009)

Source: Google

Source: Google

Source: Facebook

Storage Hierarchy



One Server
DRAM: 16 GB, 100 ns, 20 GB/s
Disk: 2T B, 10 ms, 200 MB/s
Flash: 128 GB, 100 us, 1 GB/s

Local Rack (80 servers)
DRAM: 1 TB, 300 us, 100 MB/s
Disk: 160 TB, 11 ms, 100 MB/s
Flash: 20 TB, 400 us, 100 MB/s

Cluster (30 racks)
DRAM: 30 TB, 500 us, 10 MB/s
Disk: 4.80 PB, 12 ms, 10 MB/s
Flash: 600 TB, 600 us, 10 MB/s

Source: Barroso and Urs Hölzle (2013)

Bandwidth vs Latency

- **Bandwidth**: maximum amount of data that can be transmitted per unit time (e.g. in GB/s)
- **Latency**: time taken for 1 packet to go from source to destination (*one-way*) or from source to destination back to source (*round trip*), e.g. in ms
- When transmitting a large amount of data, bandwidth tells us roughly how long the transmission will take.
- When transmitting a very small amount of data, latency tells us how much delay there will be.
- Throughput is similar to bandwidth, but instead of referring to capacity, it refers to the rate at which some data was *actually transmitted* across the network during some period of time.

Low Bandwidth

High Bandwidth

Low Latency

High Latency

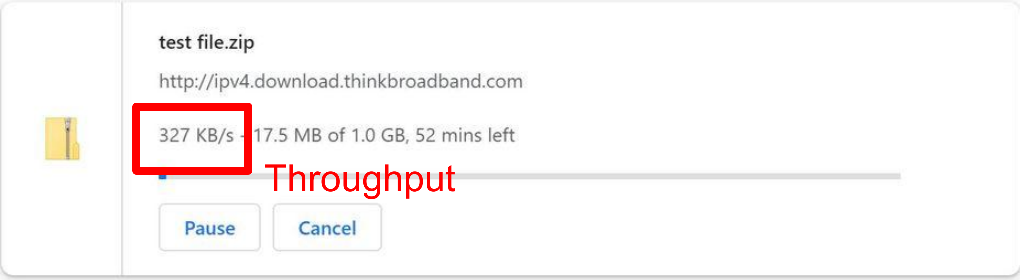
Adding many lanes to a highway: increases bandwidth, but does not decrease latency

Credit: Katie Hempenius (via Jake Archibald)

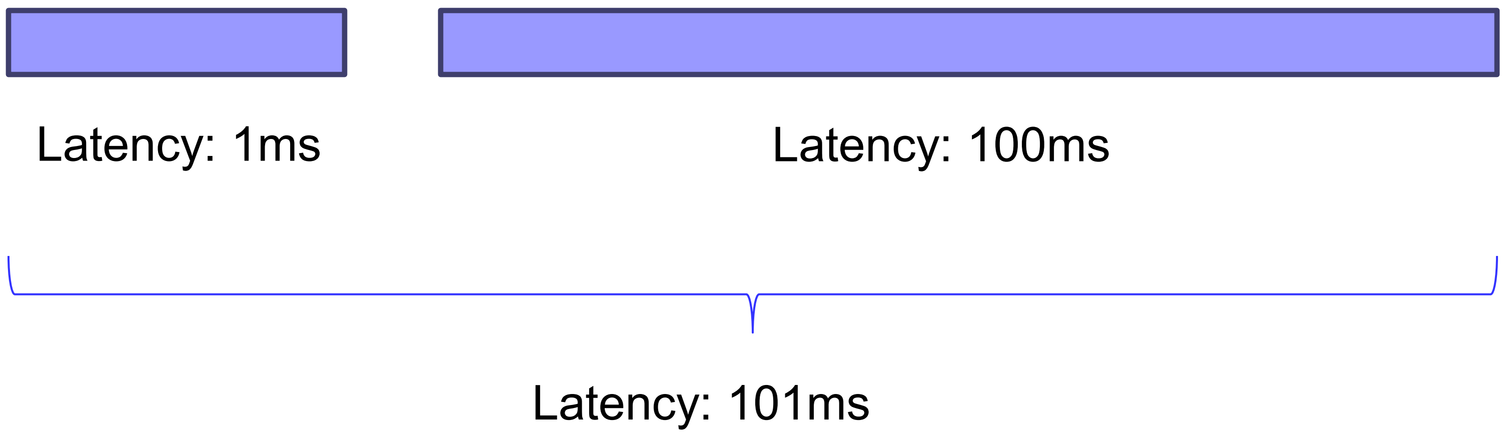
Latency and Bandwidth in Daily Life

Latency: affects delay when communicating, e.g. over a Zoom call

Bandwidth & Throughput: affect file transfer speed (esp. for large files)

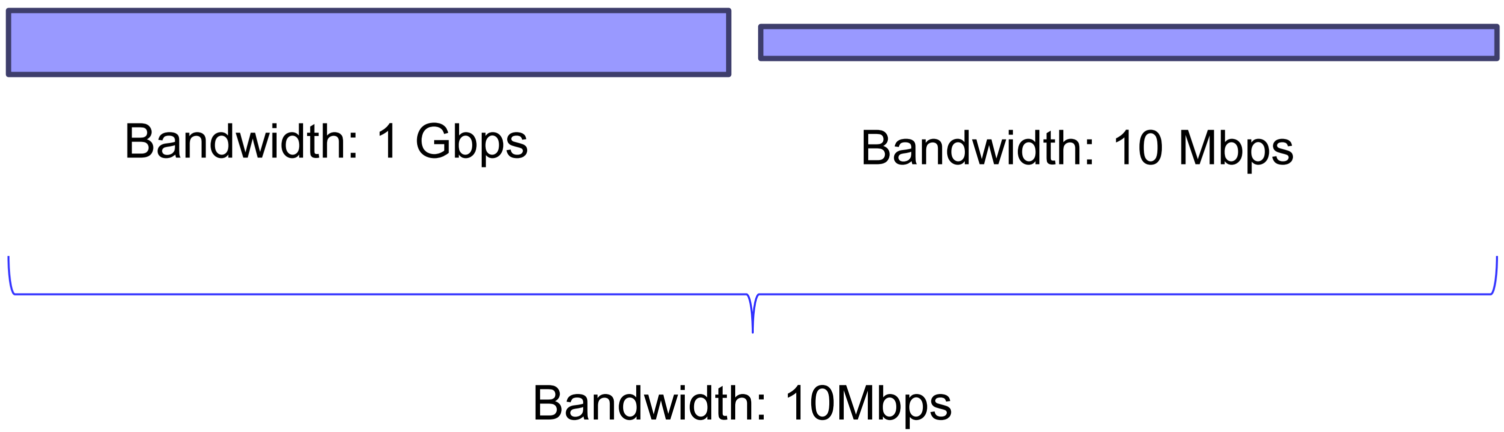


Simplified Model of Latency Along Path



- Latency combines approximately **additively**
  - Reason: in this example, the packet takes 1ms to go through the first part, and 100ms to go through the second part
- Note: The simplified model is just meant as a reasonable approximation, and is sufficient for our class' purposes
  - In practice, there are complicating factors due to transport protocols, congestion, queueing etc.

Simplified Model of Bandwidth Along Path



- Bandwidth of the whole path is approximately the **minimum** bandwidth along the path
  - Reason: the rate at which data flows through the path is “bottlenecked” by the lower bandwidth segment
- Again, this is just meant as a reasonable approximation

The Cost of Data Accesses in Data Center

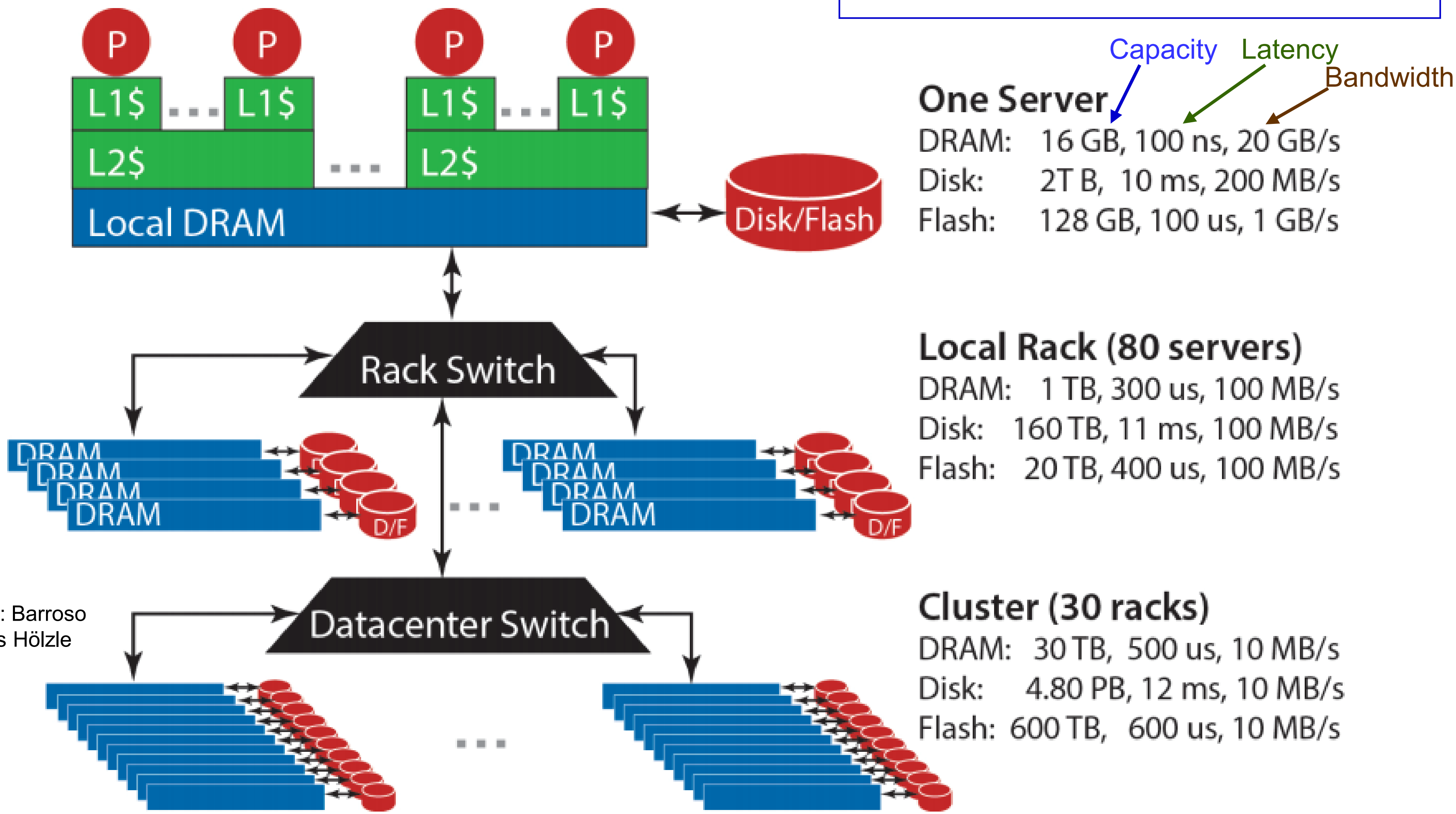
- A data access may cross several components including hard disk, DRAM, rack switch and cluster switch etc.
- We will do some back-of-envelop calculations on the cost
  - Like algorithmic complexity analysis, we approximate for the scale or the order of magnitude, rather than the exact number.
  - Many details are simplified: a) an one-way communication (data -> program), b) no failures etc.
- Latency
  - = The **sum** of the latency caused in all components in the data access.
- Bandwidth
  - = The **minimum** of the bandwidths of all components in the data access.
  - Assume the hardware peak bandwidth
  - The actual achieved bandwidth depends on the algorithm.

Storage Hierarchy

DRAM = “Dynamic Random Access Memory”: fast but limited capacity

Disk: slow but large capacity

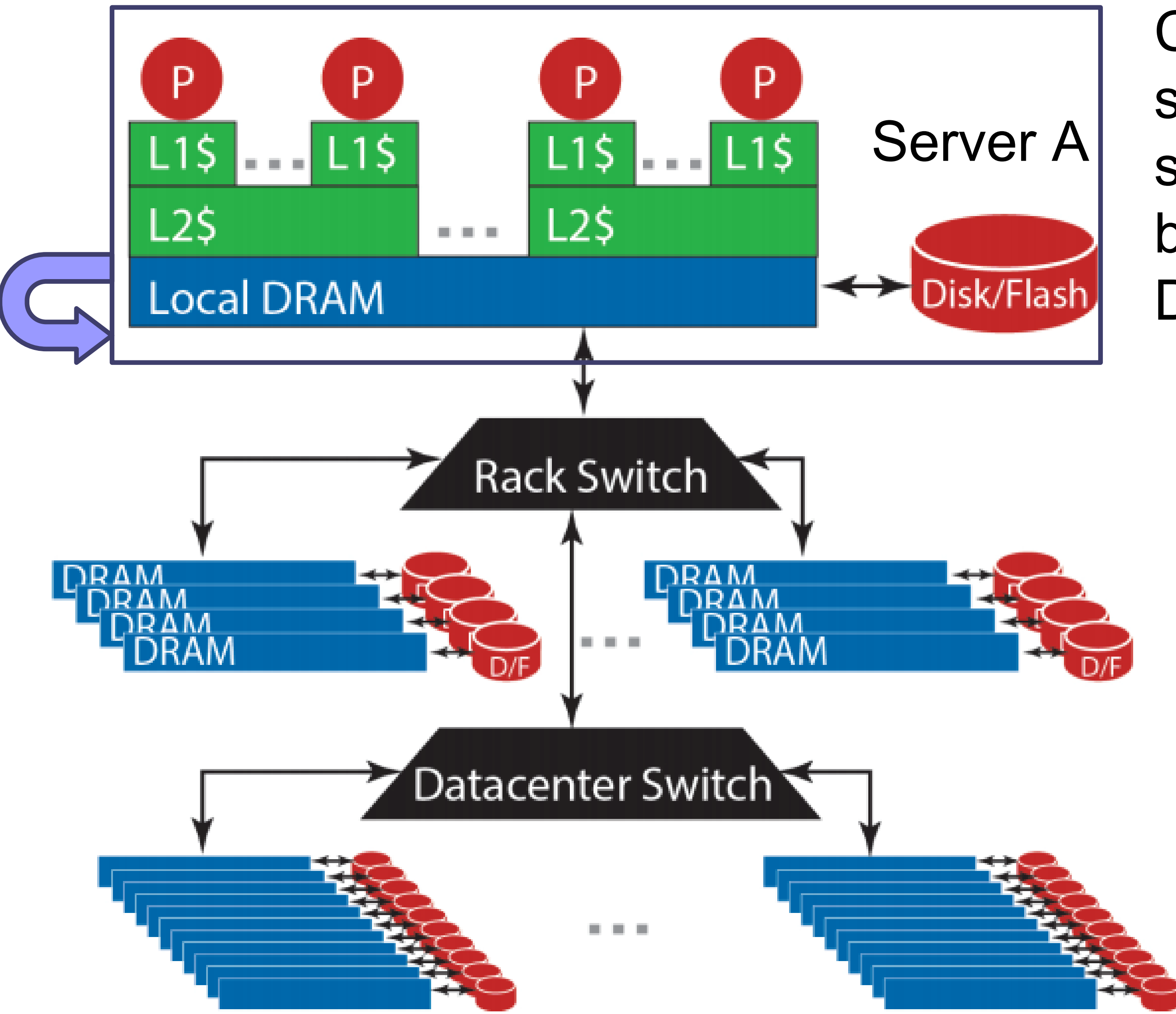
Flash: in between



Source: Barroso and Urs Hölzle (2013)

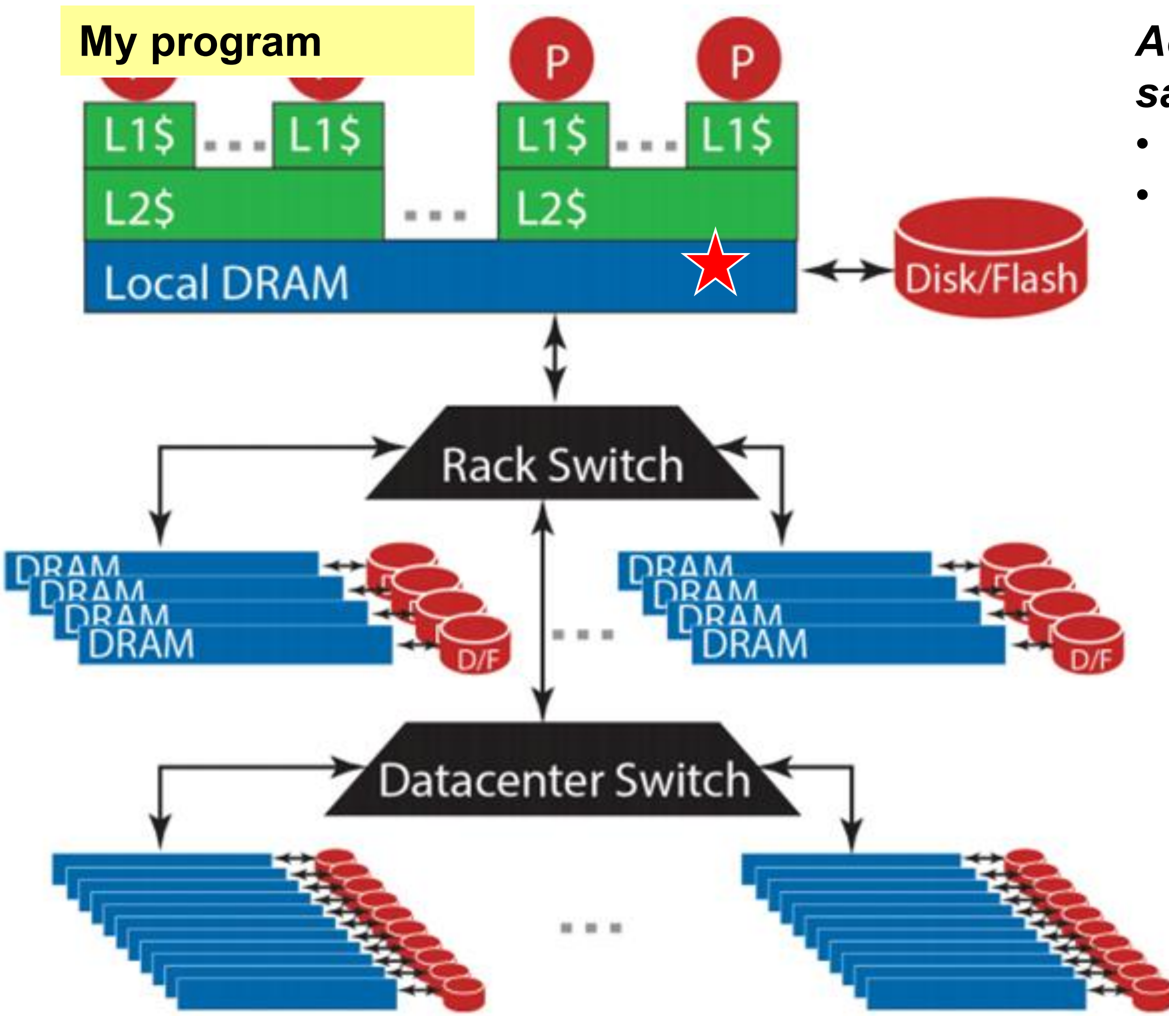
Note: no need to memorize any values. The values should be taken more as a rough “order of magnitude (OOM)” approximates. As of 2024, some of the values have improved (particularly the capacity for Flash) but not enough to completely change the discussion in terms of concepts / OOMs

Data Flow Paths: Within Local DRAM



Case 1: We are sending data from server A's DRAM, back to its own DRAM

Data Accesses in Local Machine

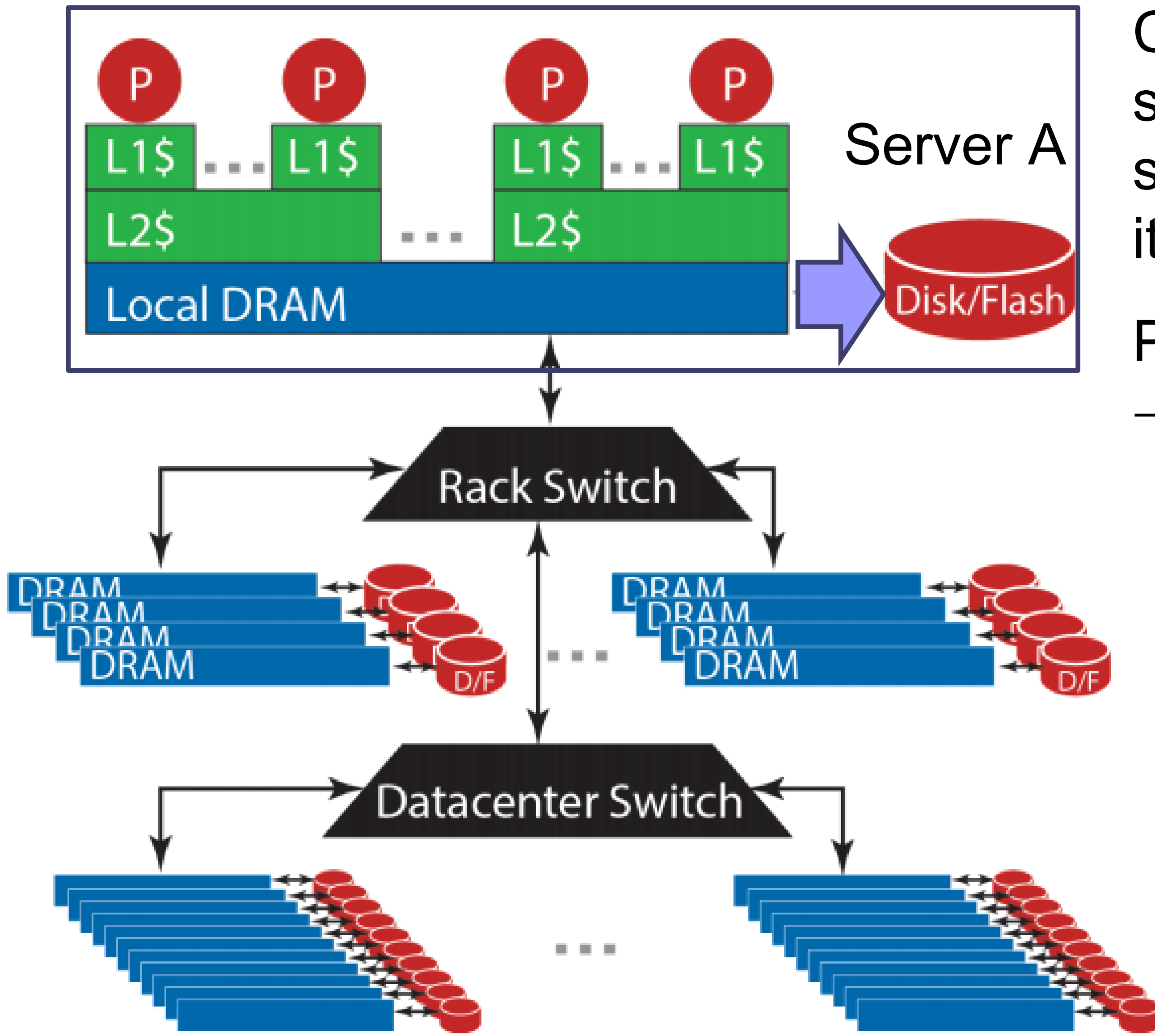


Accessing the <u>DRAM</u> on the same machine:

- **Latency=100ns**
- **Bandwidth=20GB/sec**

A nanosecond (ns or nsec) is one-billionth (10^{-9}) of a second.

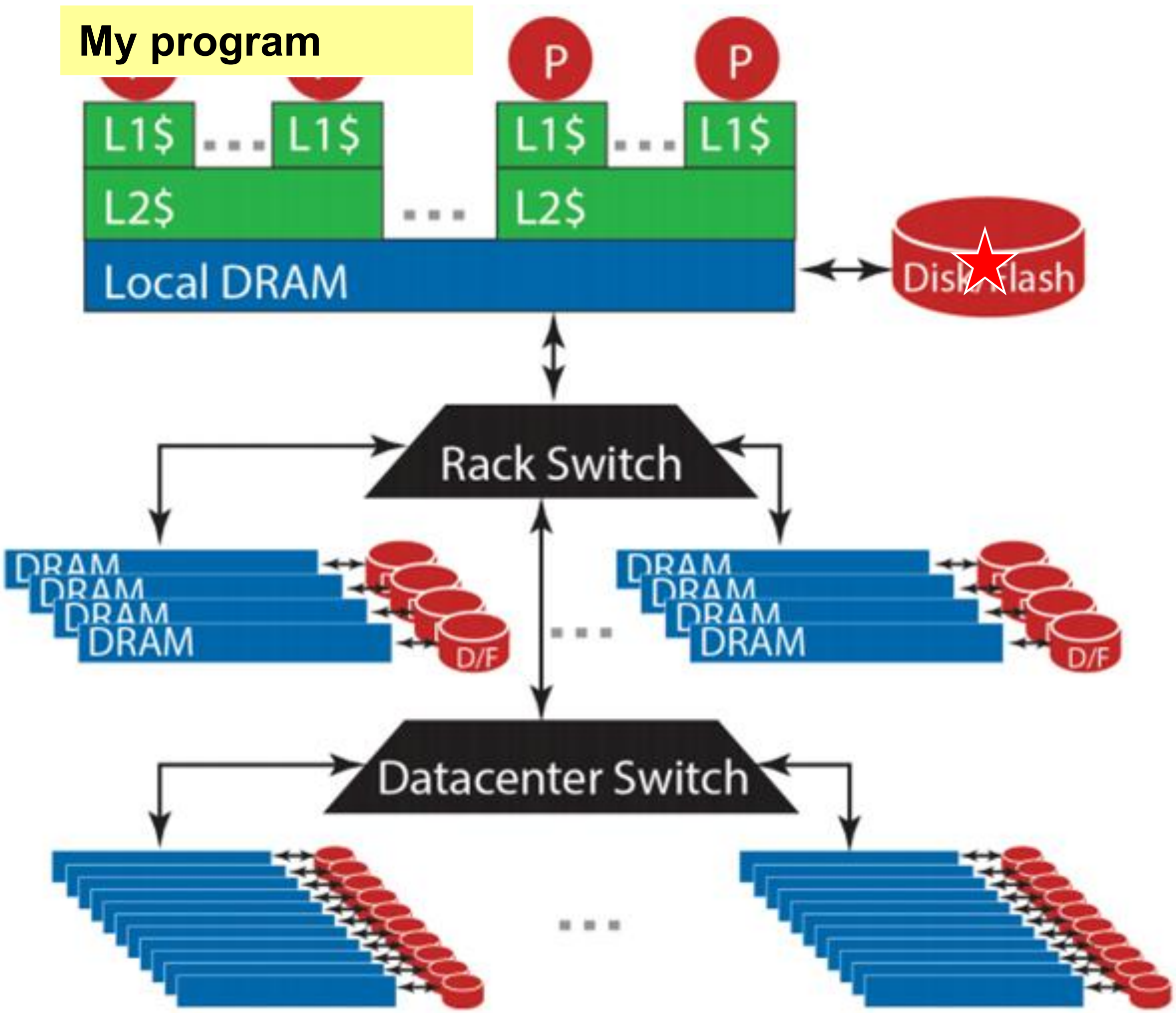
Data Flow Paths: To Local Disk



Case 2: We are sending data from server A's DRAM, to its own disk

Path: Local DRAM → Local Disk

Data Accesses in Local Machine

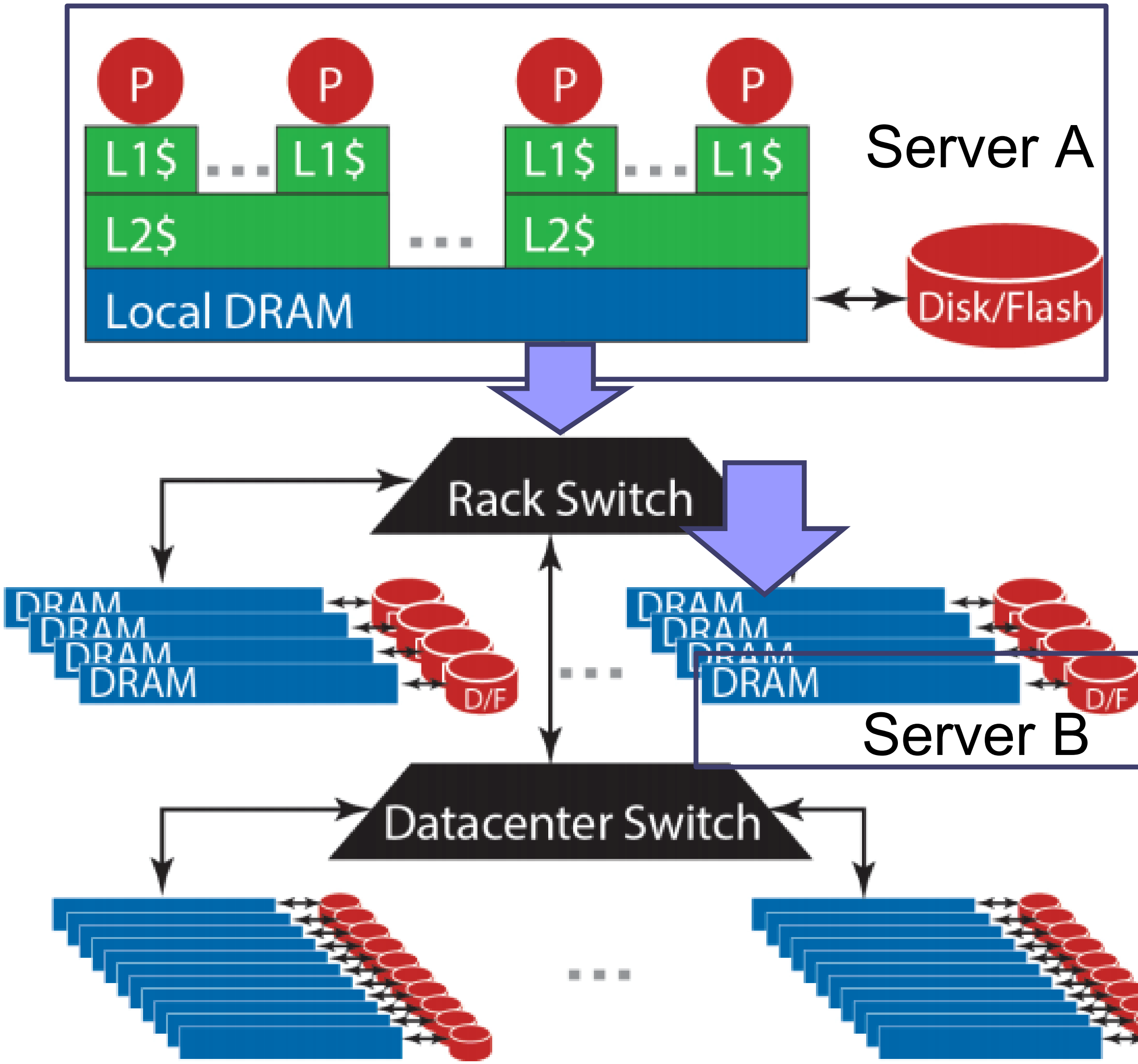


Accessing the <u>disk</u> on the same machine:

- **Latency=10ms**
- **Bandwidth= 200MB/sec**

A millisecond (ms or msec) is one-thousandth of a second (10^{-3})

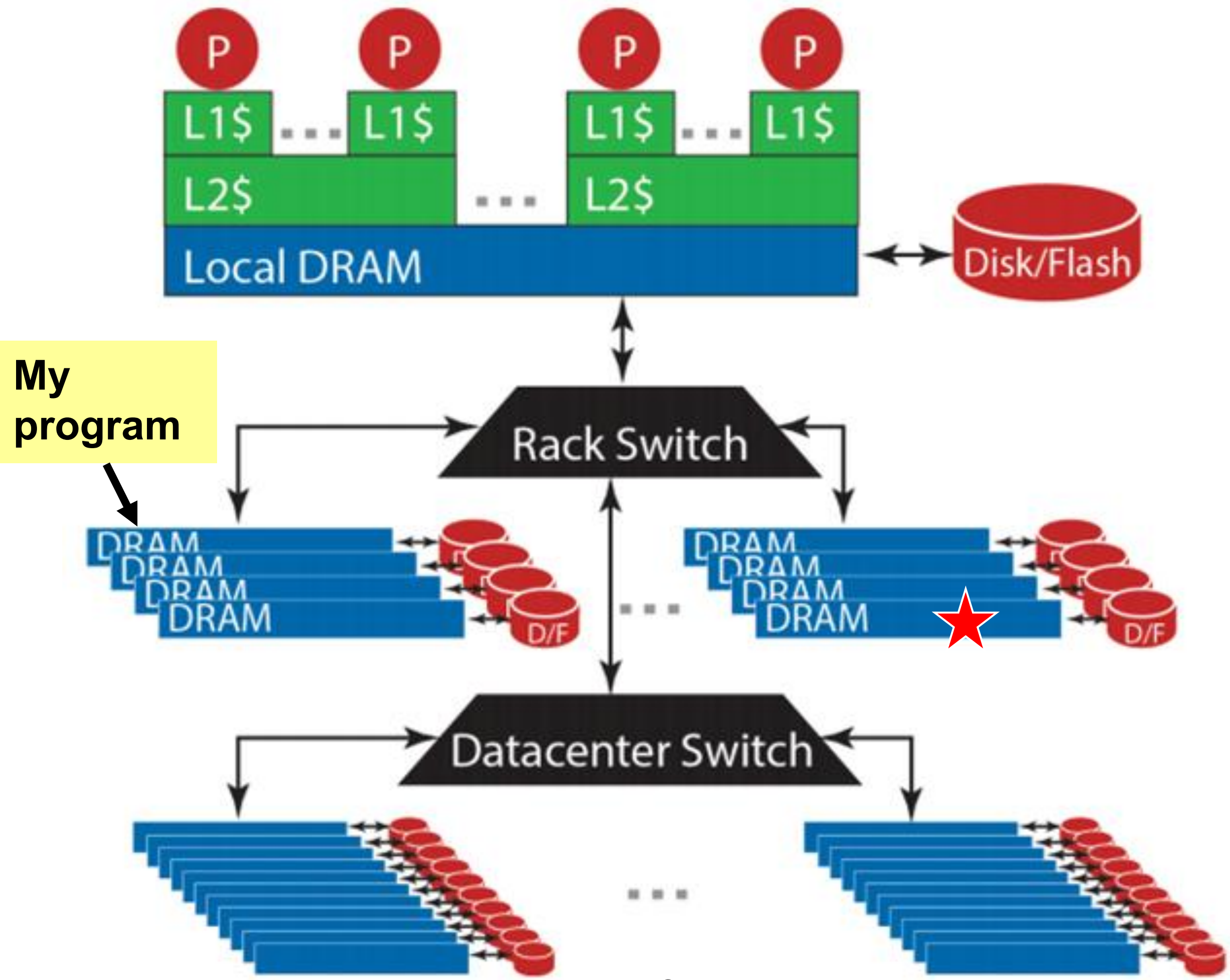
Data Flow Paths: To Rack DRAM



Case 3: We are sending data from server A's DRAM, to the DRAM of another server (B) in the same rack

Path: Local DRAM → Rack Switch → Server B's DRAM

Data Accesses within Same Rack



Rack switch (per port):
- Latency=300us
- Bandwidth=100MB/sec

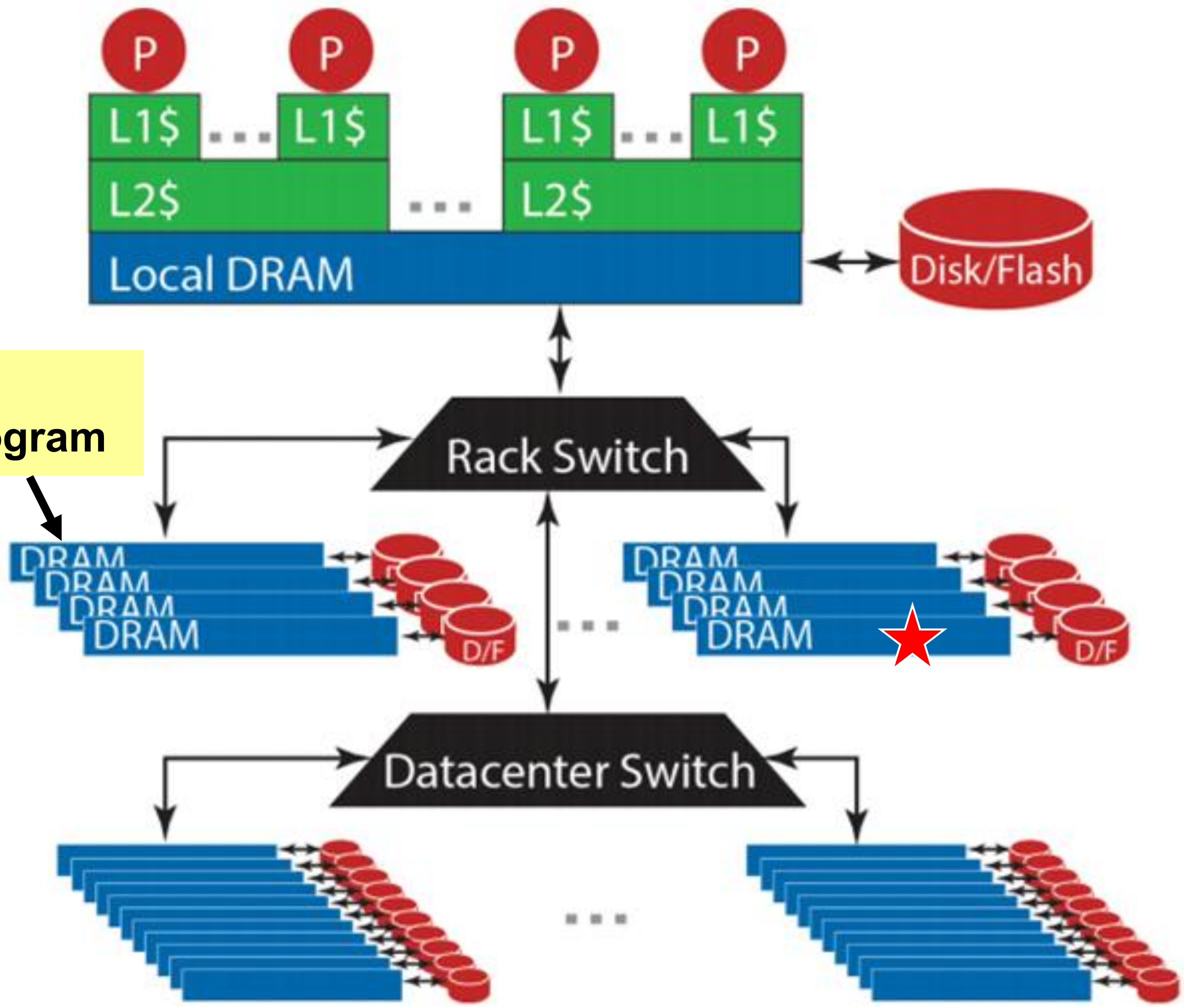
Accessing the DRAM on the another machine in the same rack:

- **Latency= ?**
- **Bandwidth= ?**

A microsecond is one-millionth (10^{-6}) of a second and is represented as µs (Greek letter mu plus s).

Data Accesses within Same Rack

My program



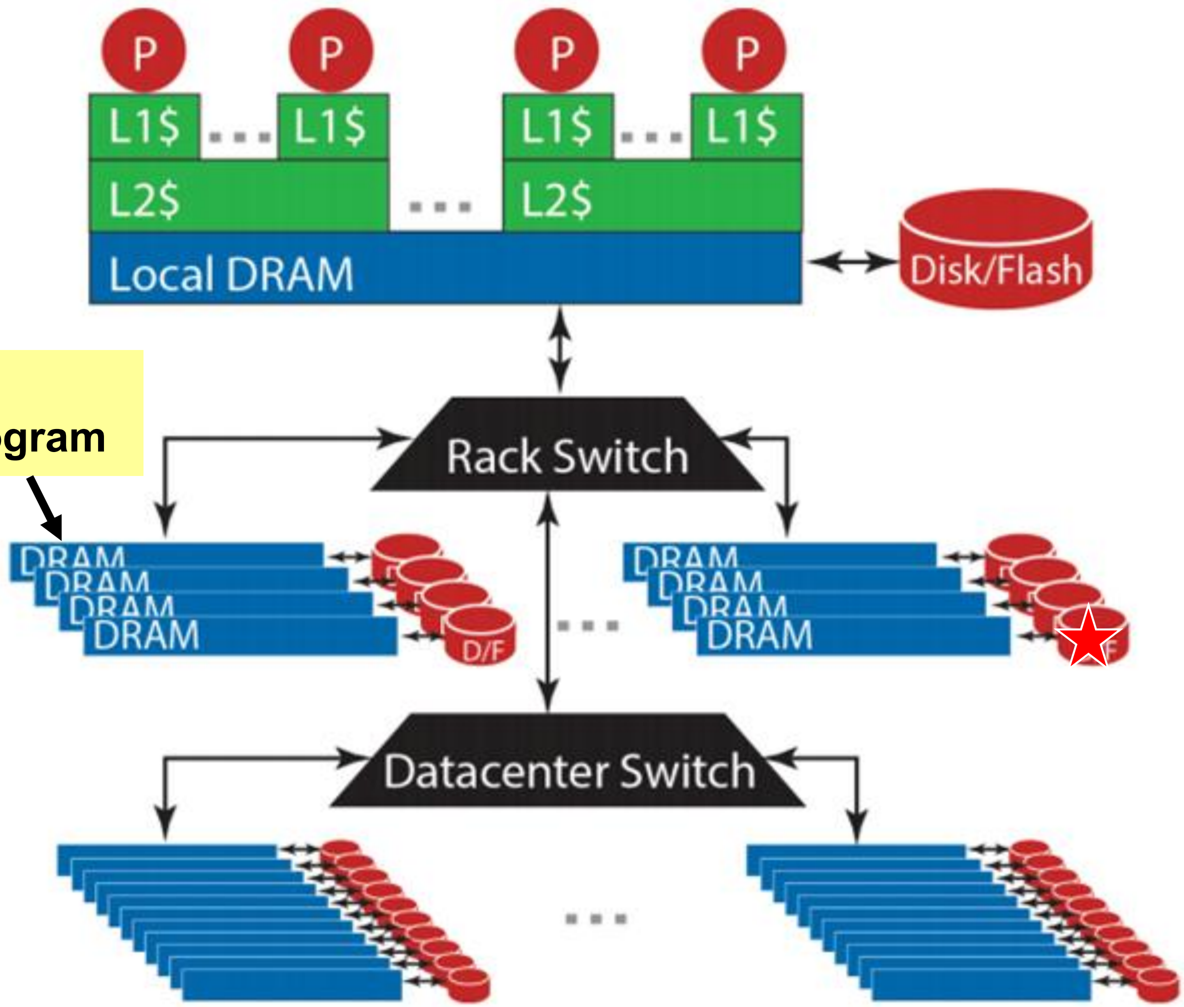
Rack switch (per port):
- Latency=300us
- Bandwidth=100MB/sec

Accessing the <u>DRAM</u> on the another machine in the same rack:

- **Latency= ~300*2 us**
- **Bandwidth= ~100MB/sec**

Data Accesses within Same Rack

My program



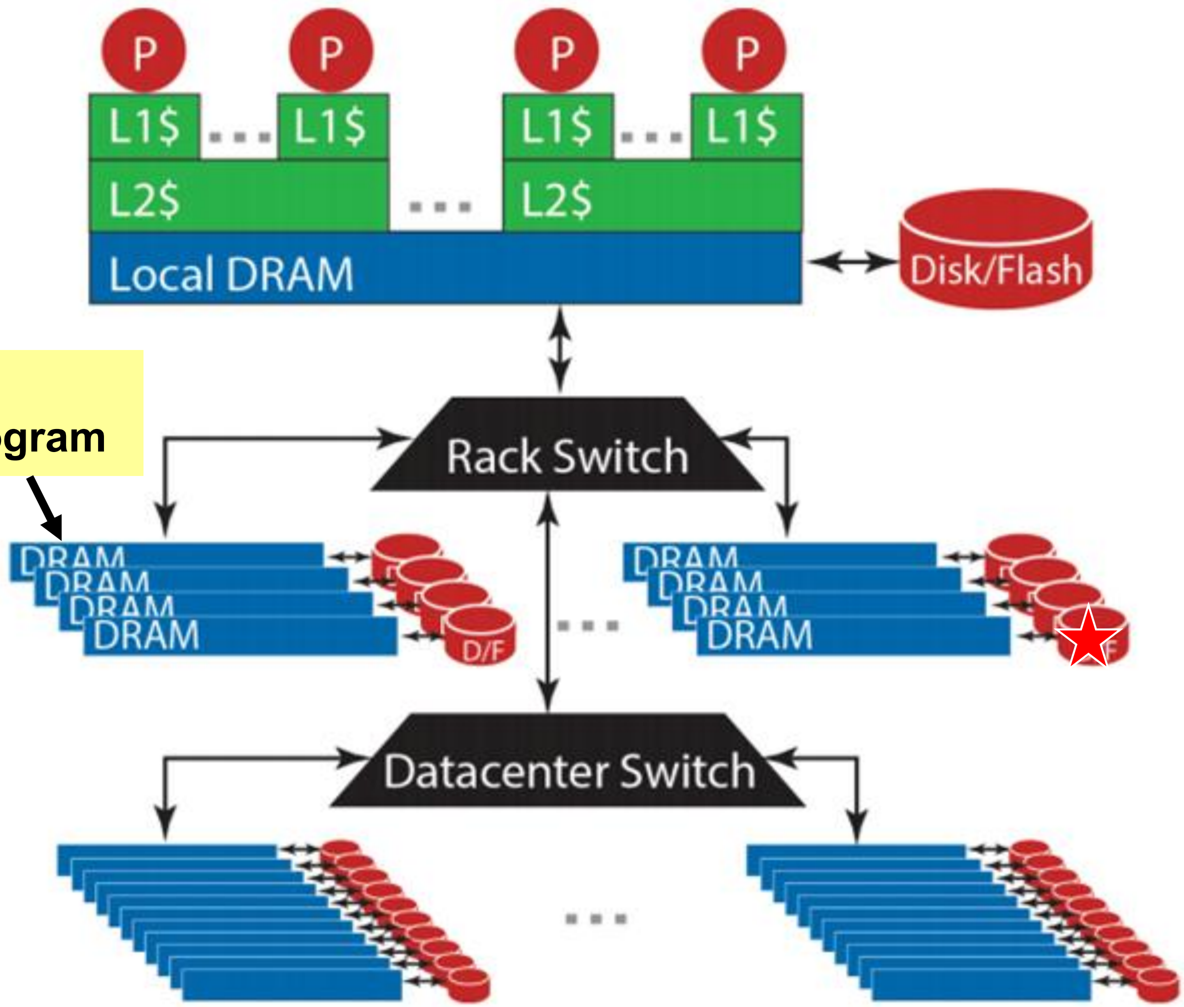
Rack switch (per port):
- Latency=300us
- Bandwidth=100MB/sec

Accessing the <u>disk</u> on the another machine in the same rack:

- **Latency= ?**
- **Bandwidth= ?**

Data Accesses within Same Rack

My program

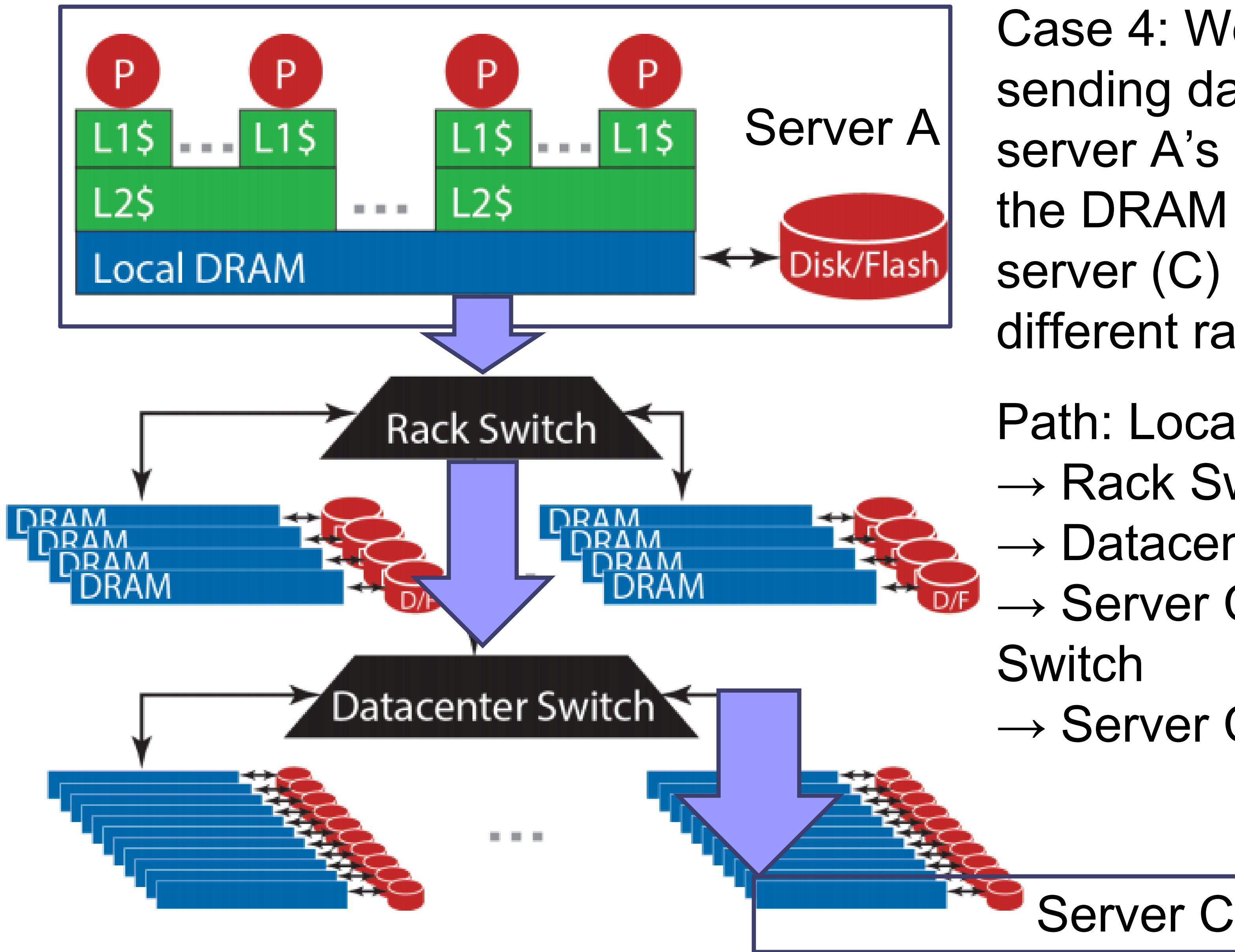


Rack switch (per port):
- Latency=300us
- Bandwidth=100MB/sec

Accessing the <u>disk</u> on the another machine in the same rack:

- **Latency= ~10ms**
- **Bandwidth= ~100MB/sec**

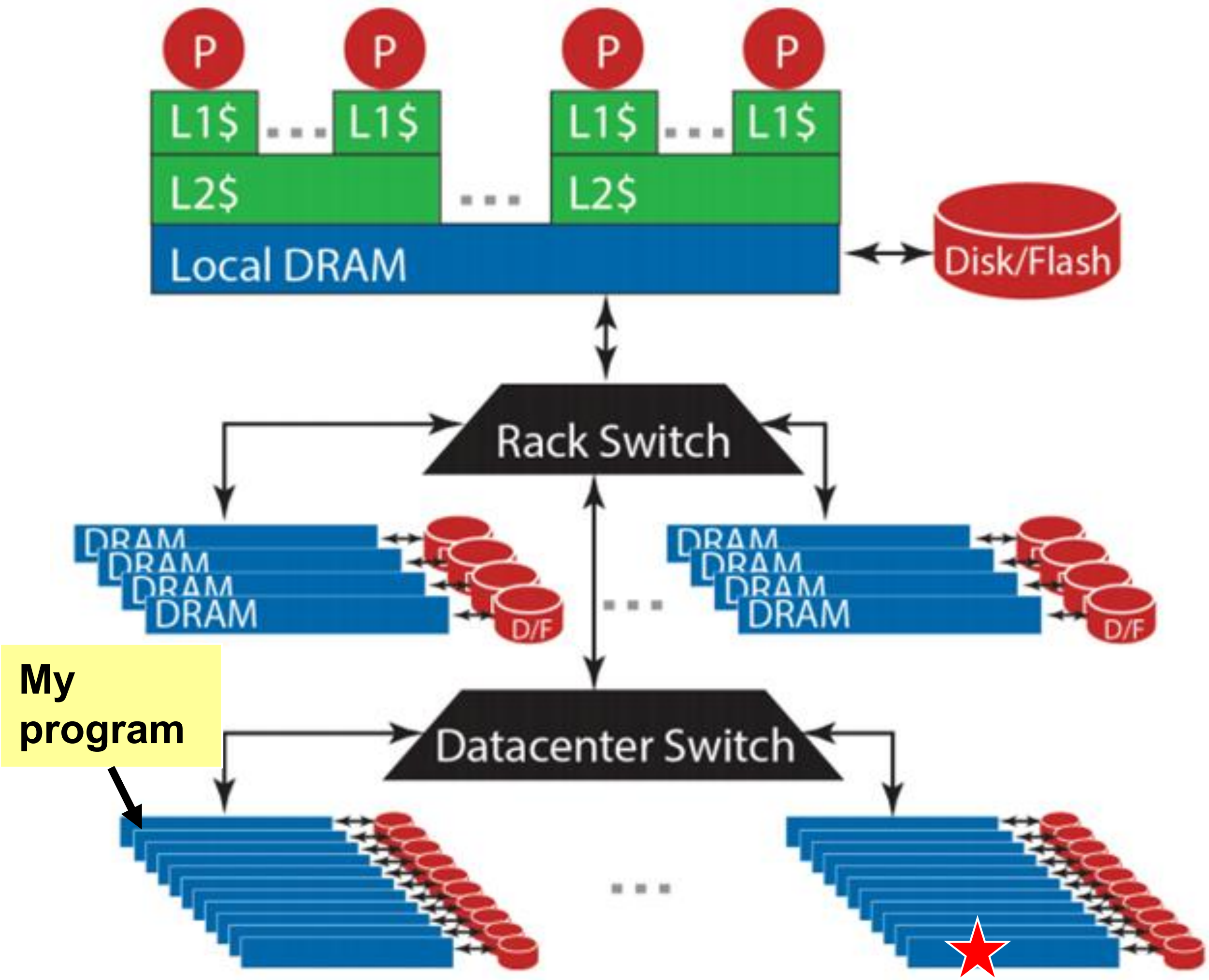
Data Flow Paths: To Datacenter DRAM



Case 4: We are sending data from server A's DRAM, to the DRAM of another server (C) in a different rack

Path: Local DRAM
→ Rack Switch
→ Datacenter Switch
→ Server C's Rack Switch
→ Server C's DRAM

Data Accesses within Data Center (But in Different Racks)

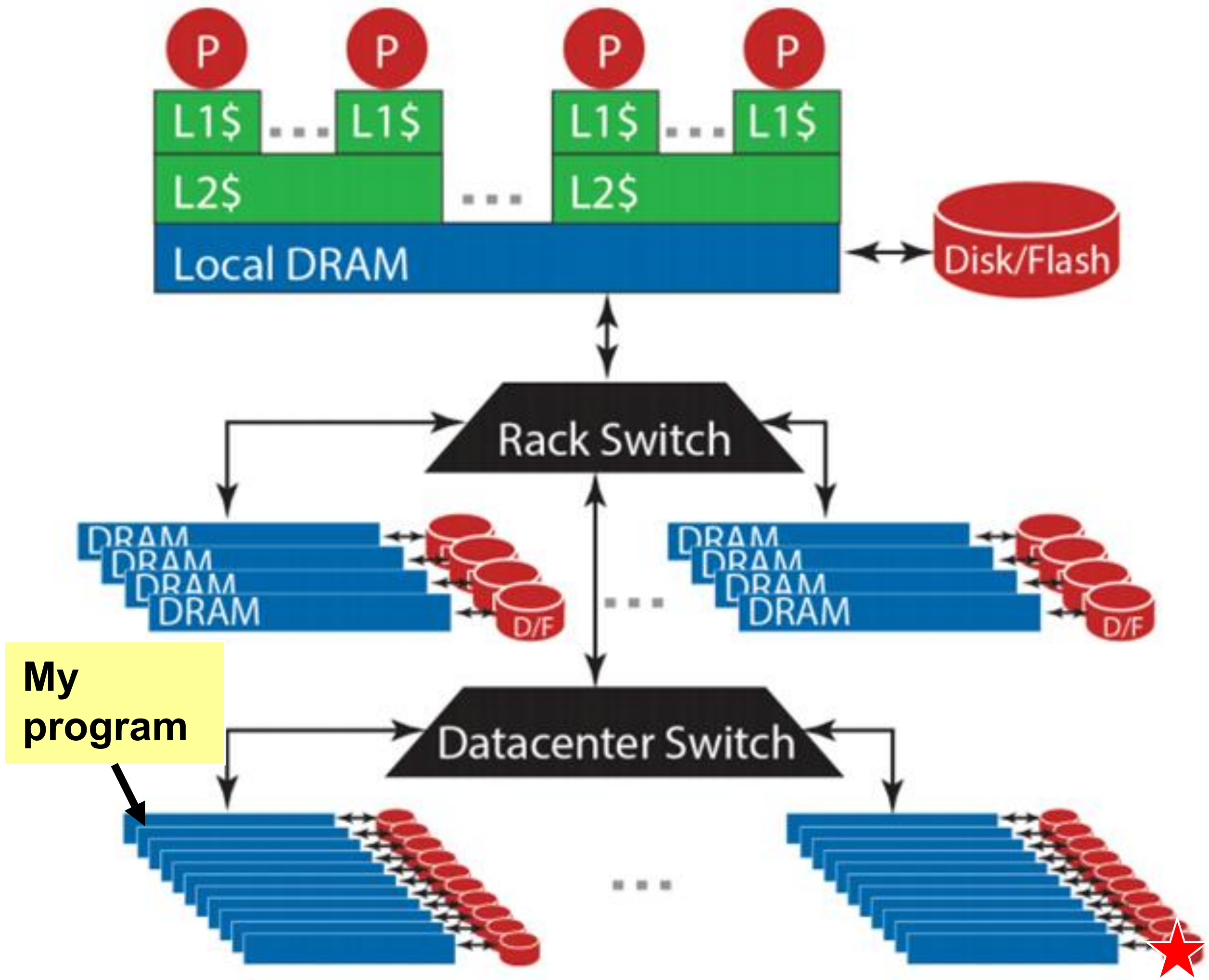


Datacenter switch (per port):
- Latency=500us
- Bandwidth=10MB/sec

Accessing the <u>DRAM</u> on the another machine in different rack:

- **Latency= ~(500*2+300*2)=1600 us**
- **Bandwidth= ~10MB/sec**

Data Accesses within Data Center (But in Different Racks)

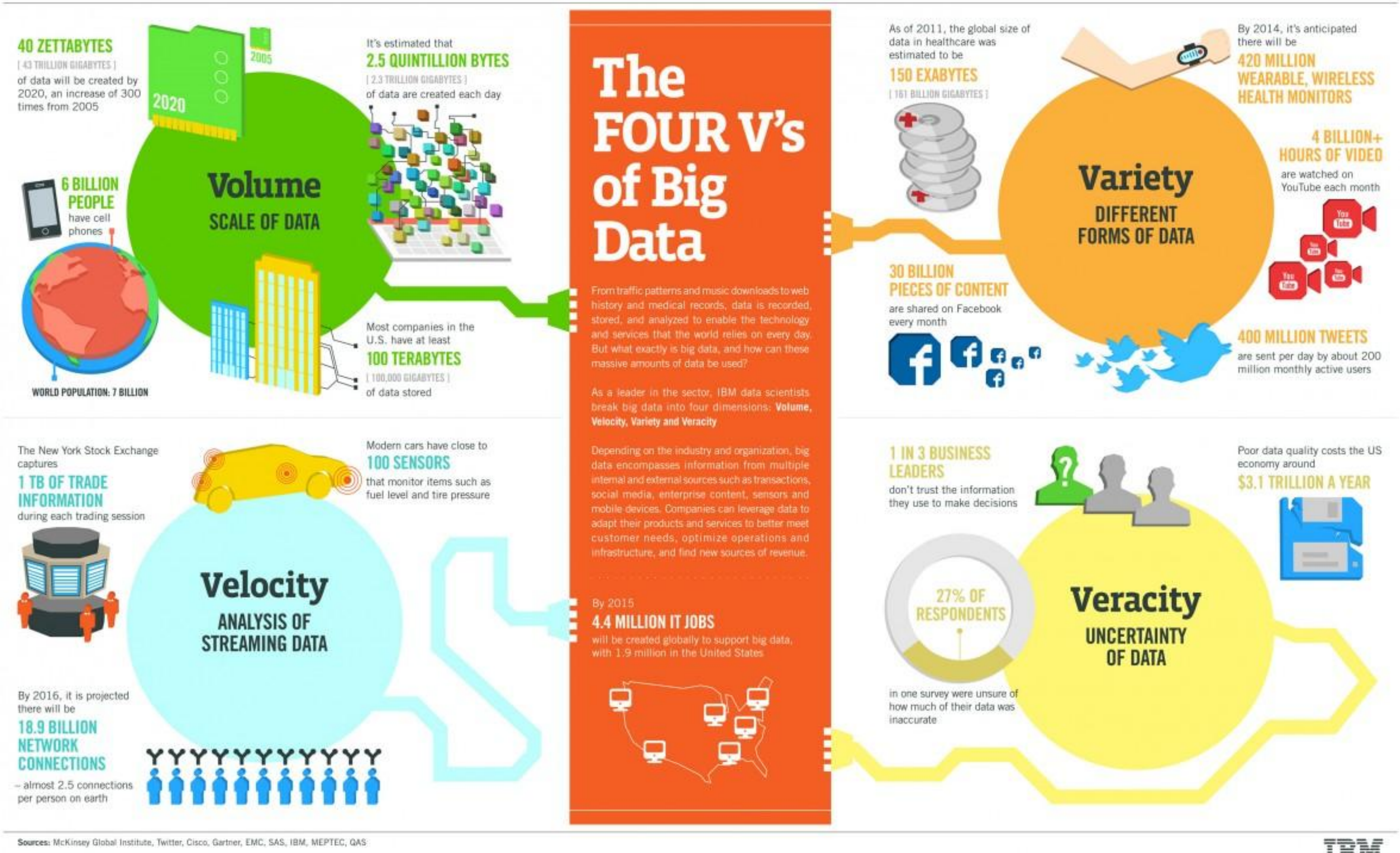


Datacenter switch (per port):
- Latency=500us
- Bandwidth=10MB/sec

Accessing the <u>hard disk</u> on the another machine in different rack:

- **Latency= ~10ms**
- **Bandwidth= ~10MB/sec**

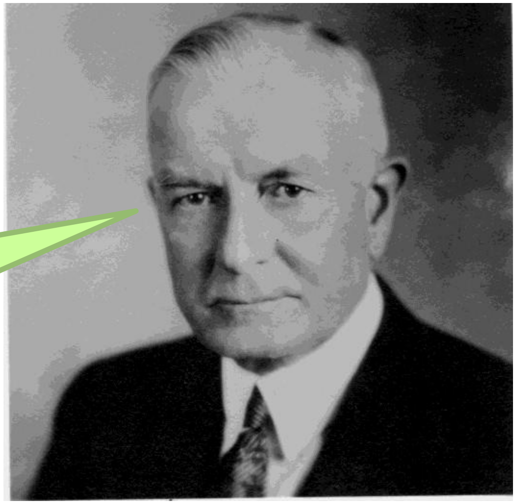
[Recap] Challenges of Big Data: the 4 'V's



[Recap] Utility Computing

- What?
  - Computing resources as a metered service (“pay as you go”)
  - Ability to dynamically provision virtual machines
- Why?
  - Scalability: “infinite” capacity
  - Elasticity: scale up or down on demand

I think there is a world market for about five computers.



Thomas J. Watson (attributed?)

[Recap] Everything as a Service

- **Infrastructure as a Service (IaaS)**: Utility Computing
  - User rents a virtual machine and makes all the decisions on what to run on it
  - Examples: Amazon’s EC2, Rackspace, Google Compute Engine
- **Platform as a Service (PaaS)**
  - Provides hosting for web applications and takes care of the hardware maintenance, upgrades, …
  - Example: Google App Engine. User provides their web application (e.g. in Python / Java) and the system takes care of all the details for hosting it.
- **Software as a Service (SaaS)**
  - User typically doesn’t write code, and is just using an existing app
  - Example: Gmail, Dropbox, Zoom

[Recap] Bandwidth vs Latenc

- **Bandwidth**: maximum amount of data that can be transmitted per unit time (e.g. in GB/s)
- **Latency**: time taken for 1 packet to go from source to destination (*one-way*) or from source to destination back to source (*round trip*), e.g. in ms
- When transmitting a large amount of data, bandwidth tells us roughly how long the transmission will take.
- When transmitting a very small amount of data, latency tells us how much delay there will be.
- Throughput is similar to bandwidth, but instead of referring to capacity, it refers to the rate at which some data was *actually transmitted* across the network during some period of time.

Low Bandwidth

High Bandwidth

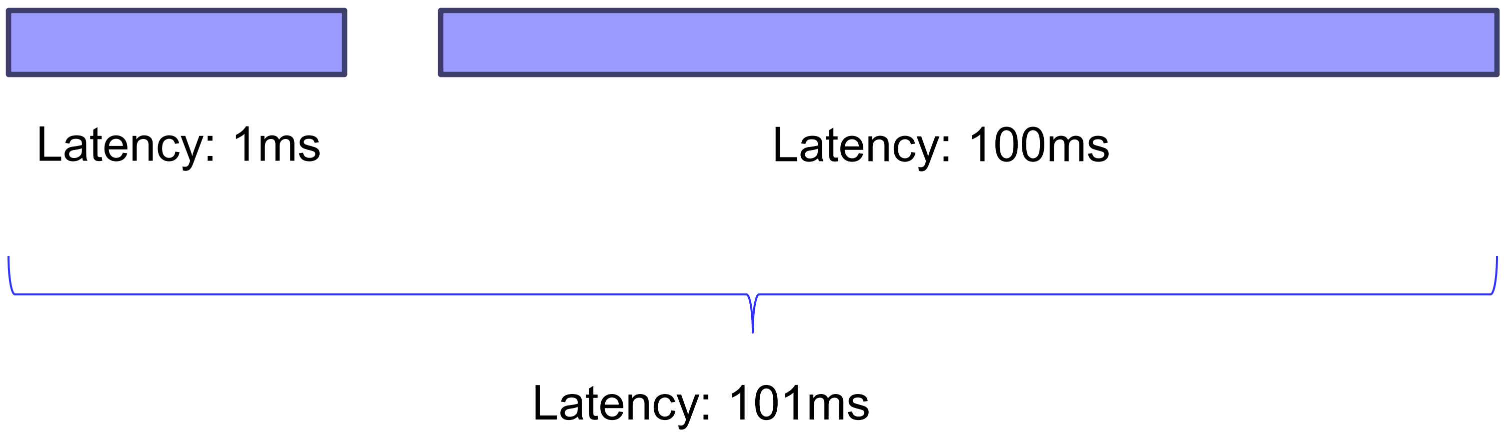
Low Latency

High Latency

Adding many lanes to a highway: increases bandwidth, but does not decrease latency

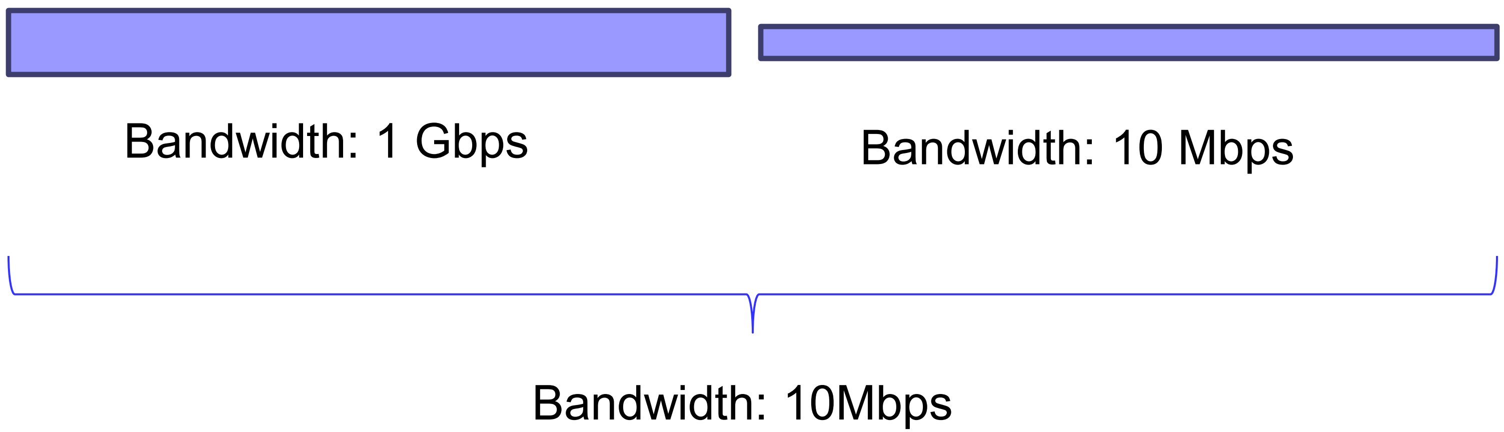
Credit: Katie Hempenius (via Jake Archibald)

[Recap] Simplified Model of Latency Along Path



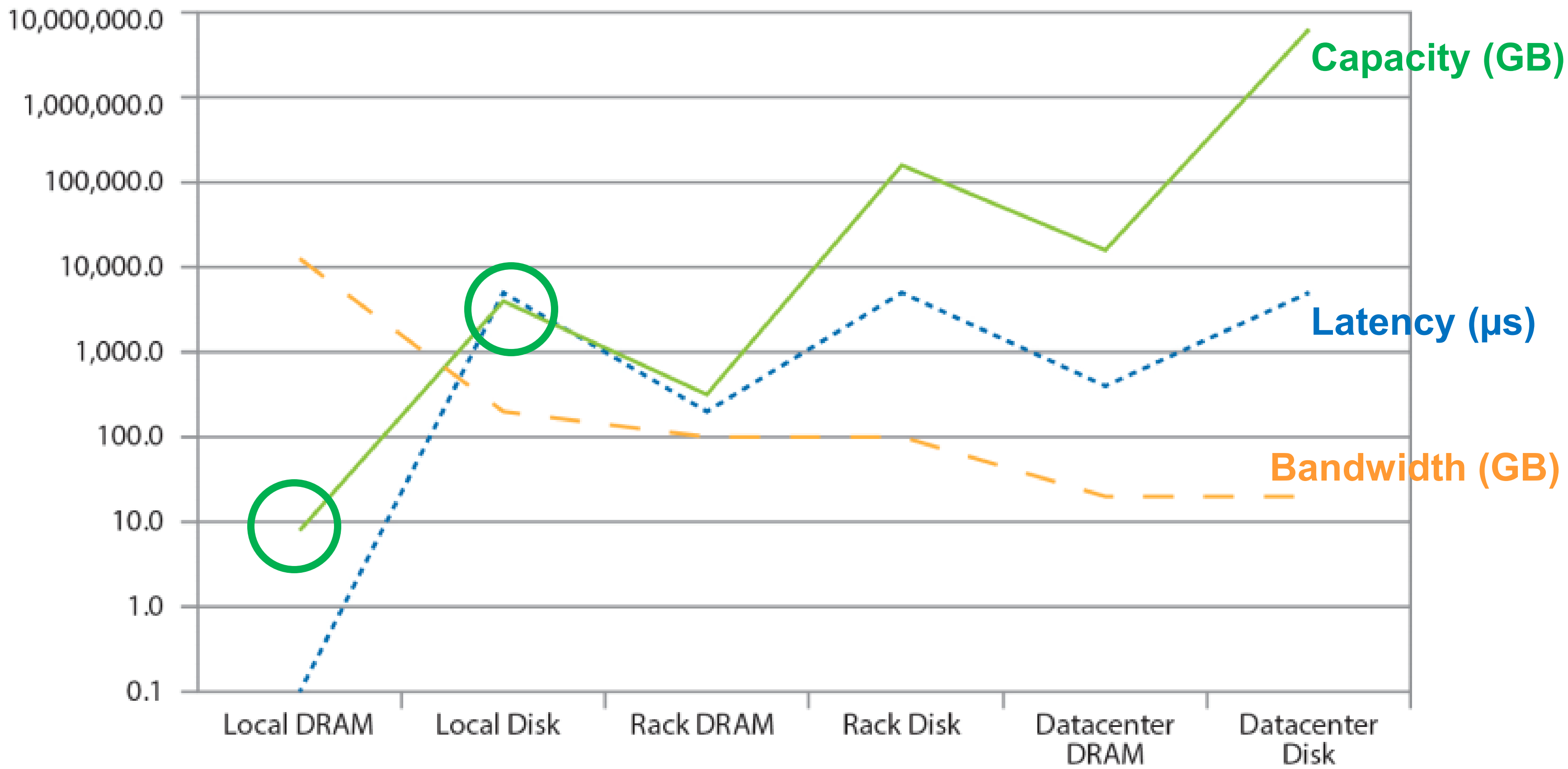
- Latency combines approximately **additively**
  - Reason: in this example, the packet takes 1ms to go through the first part, and 100ms to go through the second part
- Note: The simplified model is just meant as a reasonable approximation, and is sufficient for our class' purposes
  - In practice, there are complicating factors due to transport protocols, congestion, queueing etc.

[Recap] Simplified Model of Bandwidth Along Path



- Bandwidth of the whole path is approximately the **minimum** bandwidth along the path
  - Reason: the rate at which data flows through the path is "bottlenecked" by the lower bandwidth segment
- Again, this is just meant as a reasonable approximation

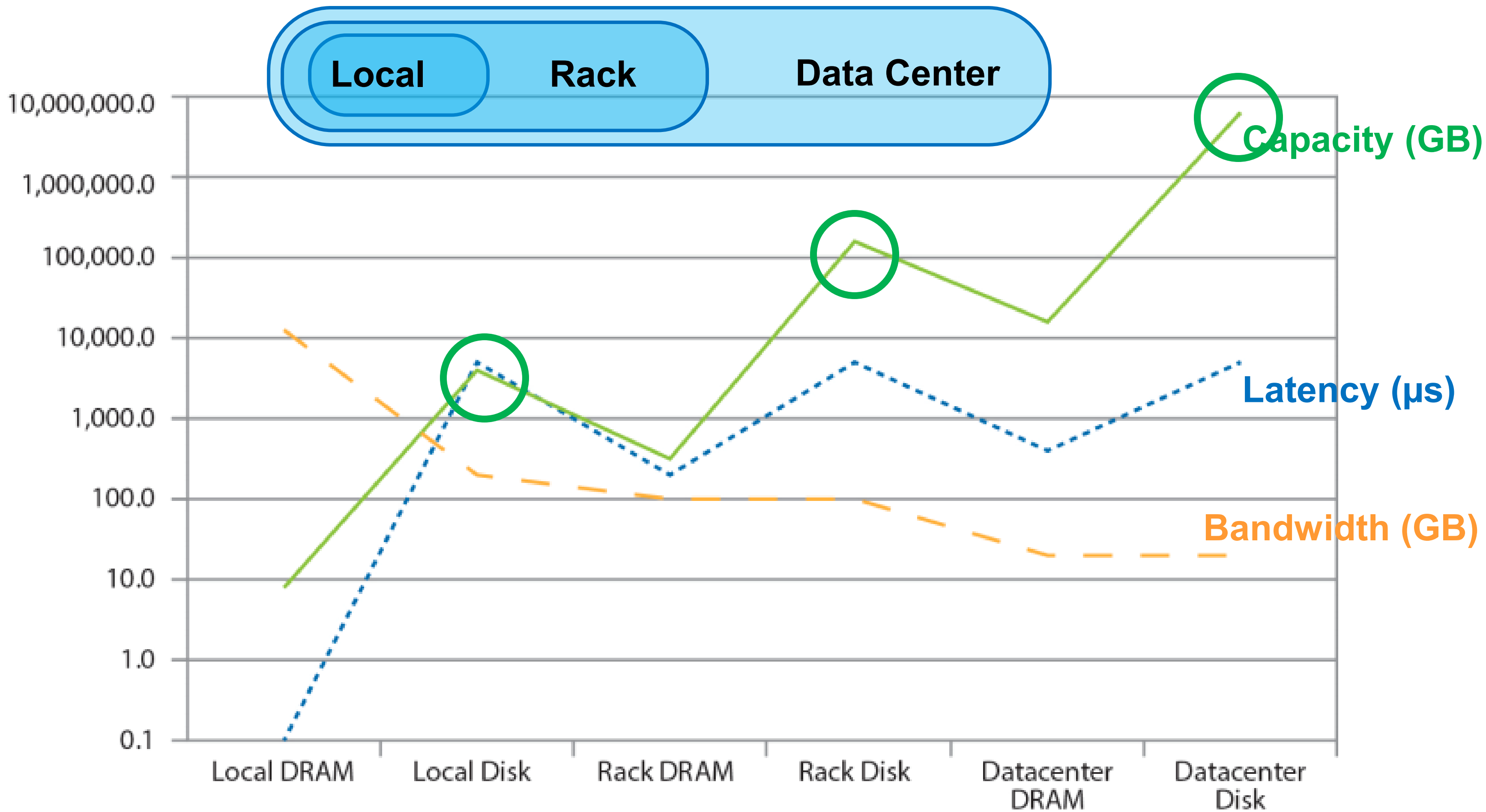
Capacity of Storage Hierarchy



1. **Disk has much higher capacity than DRAM**

Source: Barroso and Urs Hölzle (2013)

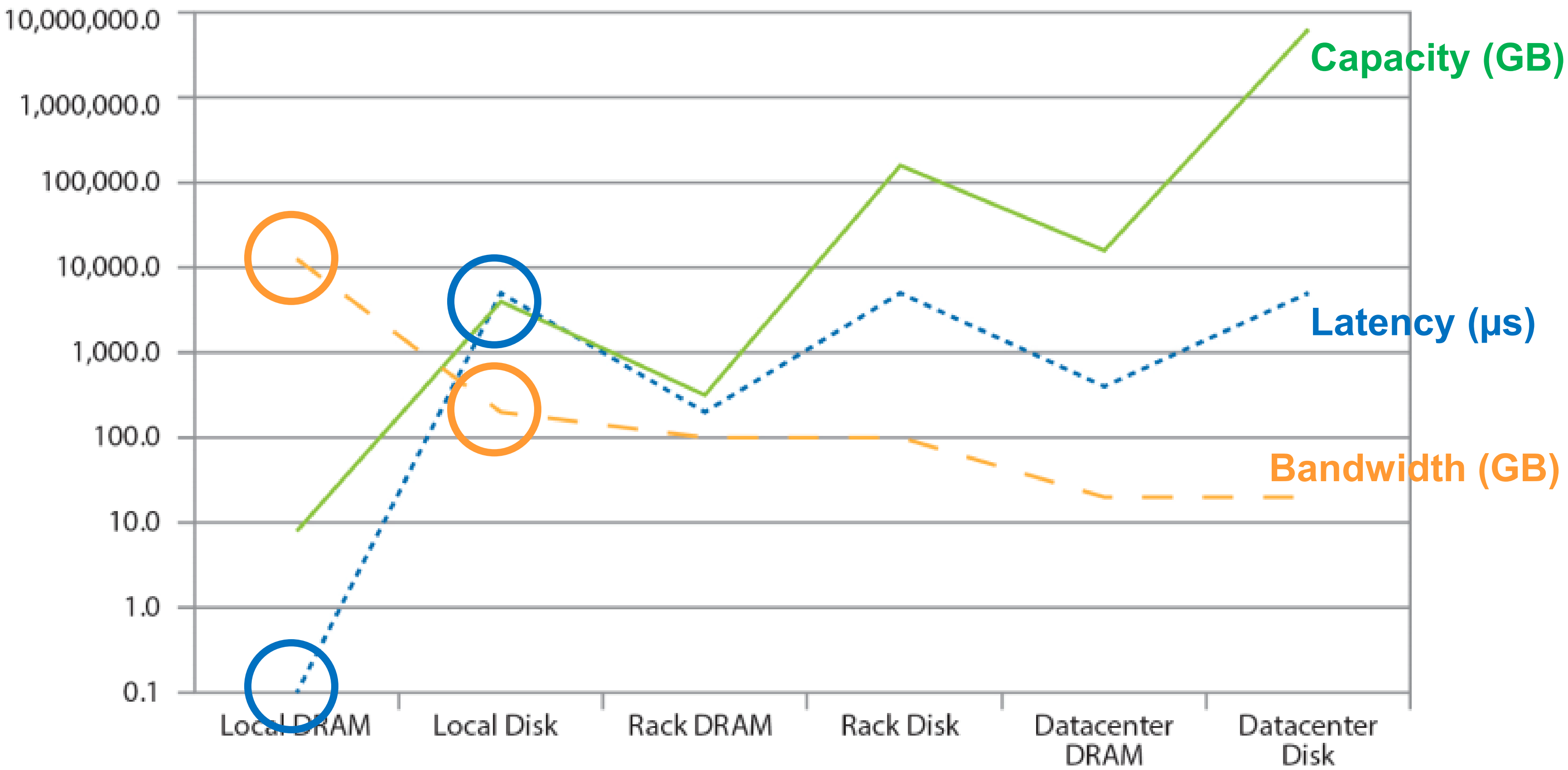
Capacity of Storage Hierarchy



2. **Storage hierarchy**: *capacity* increases as we go from Local Server, to Rack, to Datacenter.

Source: Barroso and Urs Hölzle (2013)

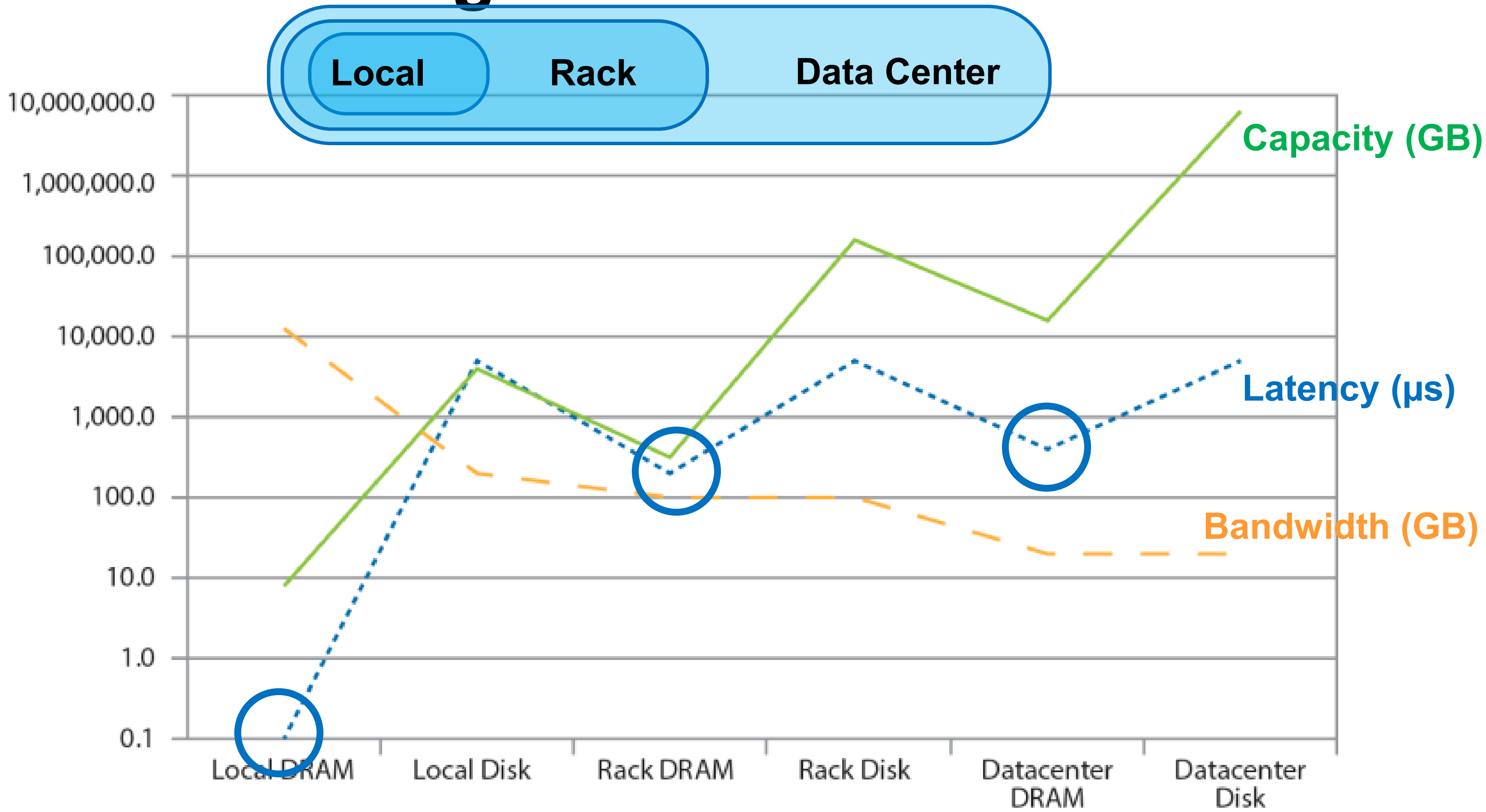
Speed of Moving Data Around Data Center



3. **Disk reads are much more expensive than DRAM**, both in terms of with **higher latency** and **lower bandwidth**.

Source: Barroso and Urs Hölzle (2013)

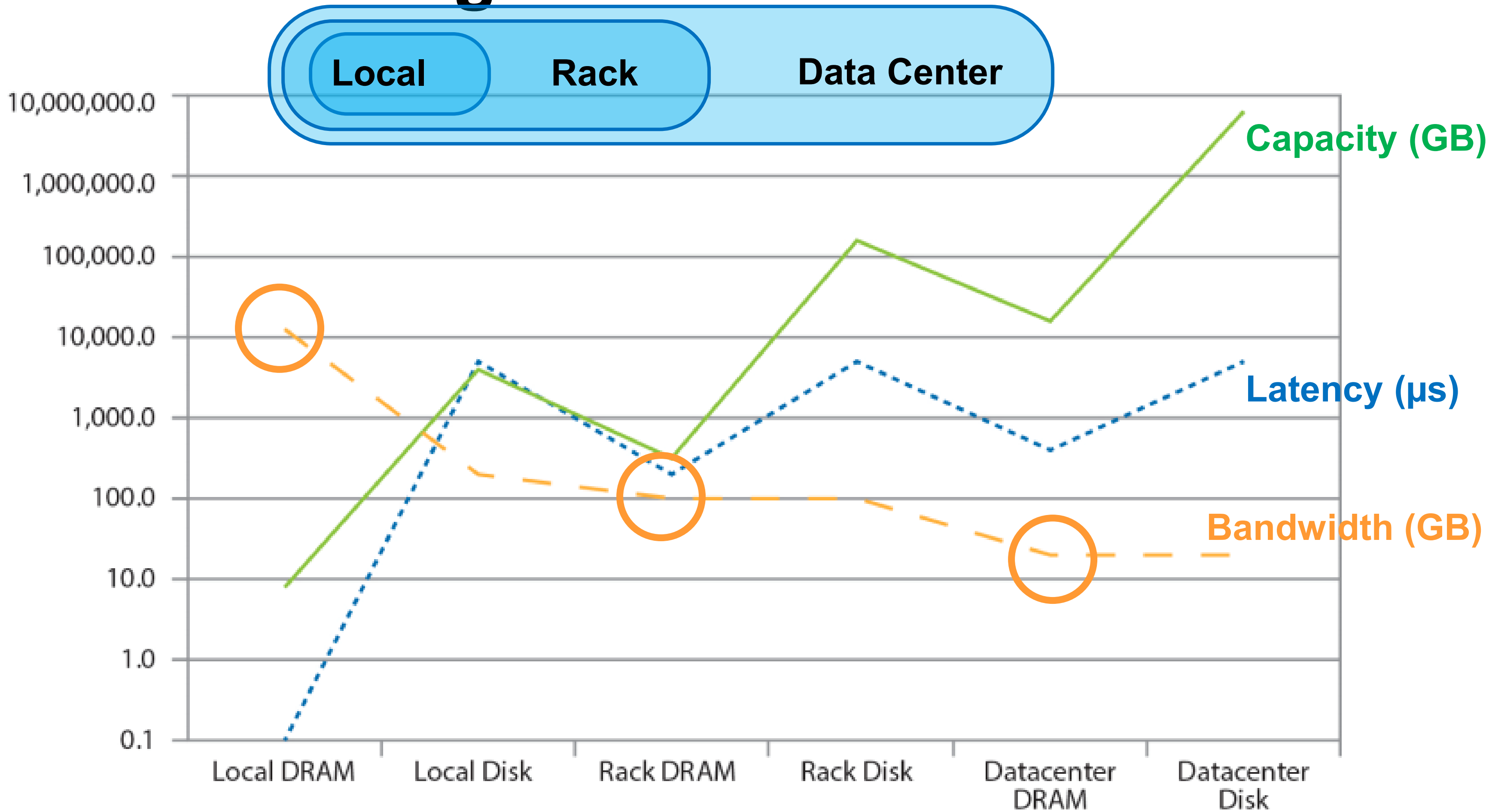
Speed of Moving Data Around Data Center



4. **Costs increase over the storage hierarchy**: **latency increases** as we go from Local to Rack to Datacenter.

Source: Barroso and Urs Hölzle (2013)

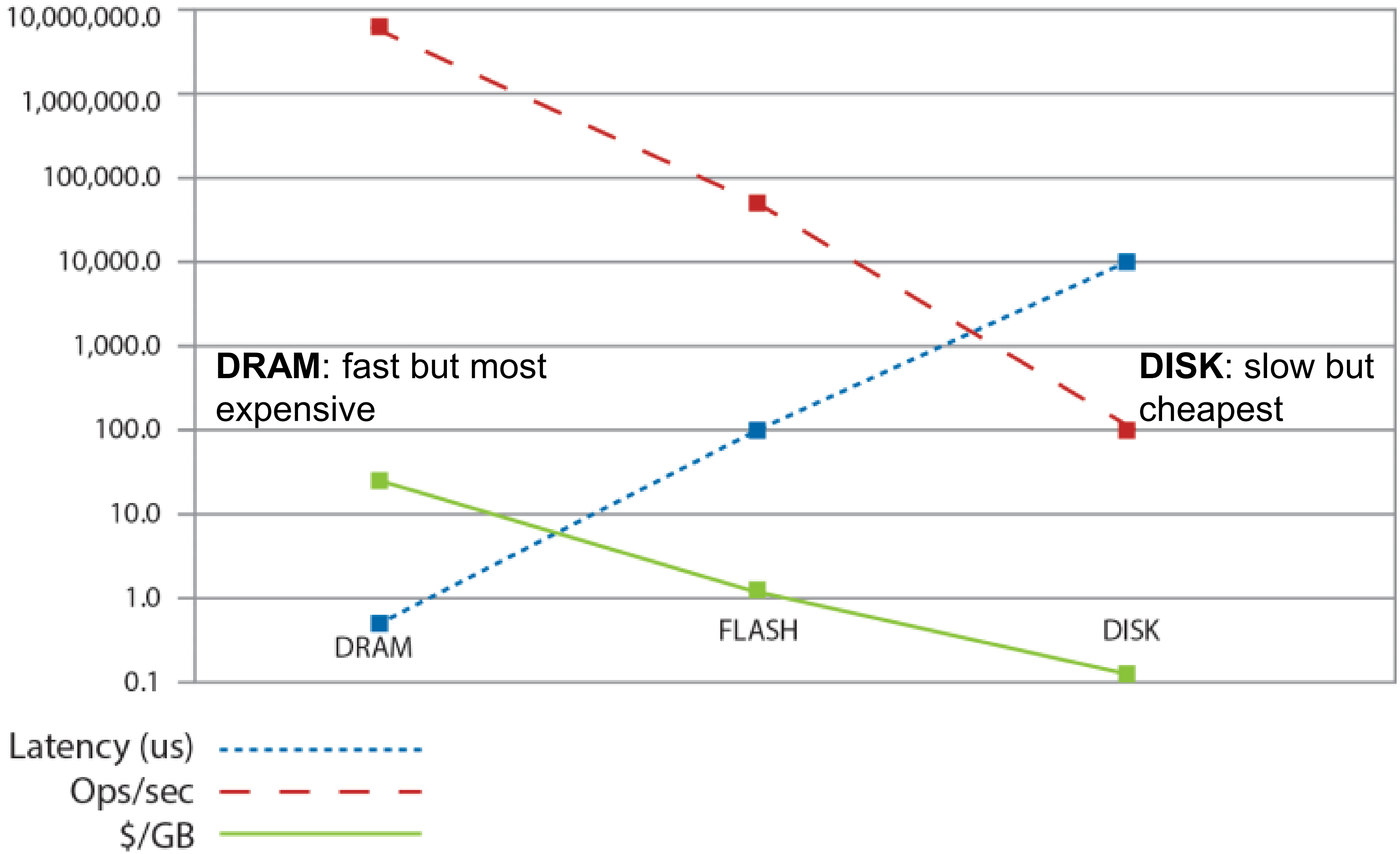
Speed of Moving Data Around Data Center



5. **Costs increase over the storage hierarchy**: **bandwidth decreases** as we go from Local to Rack to Datacenter.

Source: Barroso and Urs Hölzle (2013)

The Cost of Moving Data Within a Server



Source: Barroso and Urs Hölzle (2013)

Quiz Q1

- Server A is downloading a file from the disk of server B in the same rack. What is the pathway that data from the file travel to go from server B to server A?

(A) Server B's DRAM → Rack Switch → Server A's DRAM

(B) Server B's disk → Server B's DRAM → Rack Switch → Server A's DRAM

(C) Server B's disk → Server B's DRAM → Rack Switch → Datacenter Switch → Rack Switch → Server A's DRAM

Quiz Q1

- Server A is downloading a file from the disk of server B in the same rack. What is the pathway that data from the file travel to go from server B to server A?

(A) Server B's DRAM → Rack Switch → Server A's DRAM

(B) Server B's disk → Server B's DRAM → Rack Switch → Server A's DRAM

(C) Server B's disk → Server B's DRAM → Rack Switch → Datacenter Switch → Rack Switch → Server A's DRAM

Quiz Q2

- Server A is downloading a file from the disk of server B in the same rack. If the file is very small, which of these would most likely determine how long this process takes?

(A) Disk latency

(B) Disk bandwidth

(C) Network switch bandwidth

(D) Network switch latency

Quiz Q2

- Server A is downloading a file from the disk of server B in the same rack. If the file is very small, which of these would most likely determine how long this process takes?

$\boxed{\text{(A) Disk latency}}$

(B) Disk bandwidth

(C) Network switch bandwidth

(D) Network switch latency

Quiz Q3

- Server A is downloading a file from the disk of server B in the same rack. If the file is very large, which of these would most likely determine how long this process takes?

(A) Disk latency

(B) Disk bandwidth

(C) Network switch bandwidth

(D) Network switch latency

Quiz Q3

- Server A is downloading a file from the disk of server B in the same rack. If the file is very large, which of these would most likely determine how long this process takes?

(A) Disk latency

(B) Disk bandwidth

(C) Network switch bandwidth

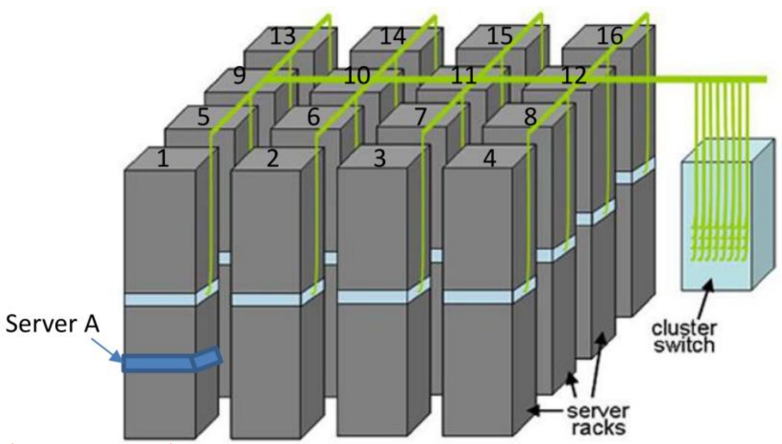
(D) Network switch latency

Quiz 4

The architecture of a commercial data center is illustrated in the below figure. The number on the top of each rack is the identifier of each rack. Users can run Hadoop or Spark jobs in the data center.

Suppose a program P is running on Server A in Rack 1. Denote the latency of P accessing a byte in the hard disk of Server A, a byte in the hard disk of the other servers in Rack 1, and a byte in the hard disk of the other server in Rack 16 is L1, L2 and L3, respectively. Which of the following statements are True?

- S1. L3 can be ten times larger than L2.
- S2. L2 can be ten times larger than L1.
- S3. L3 is roughly the same as L2.
- S4. L2 is roughly the same as L1.



Answer: S3 and S4 (hard disk is the slowest)

Outline

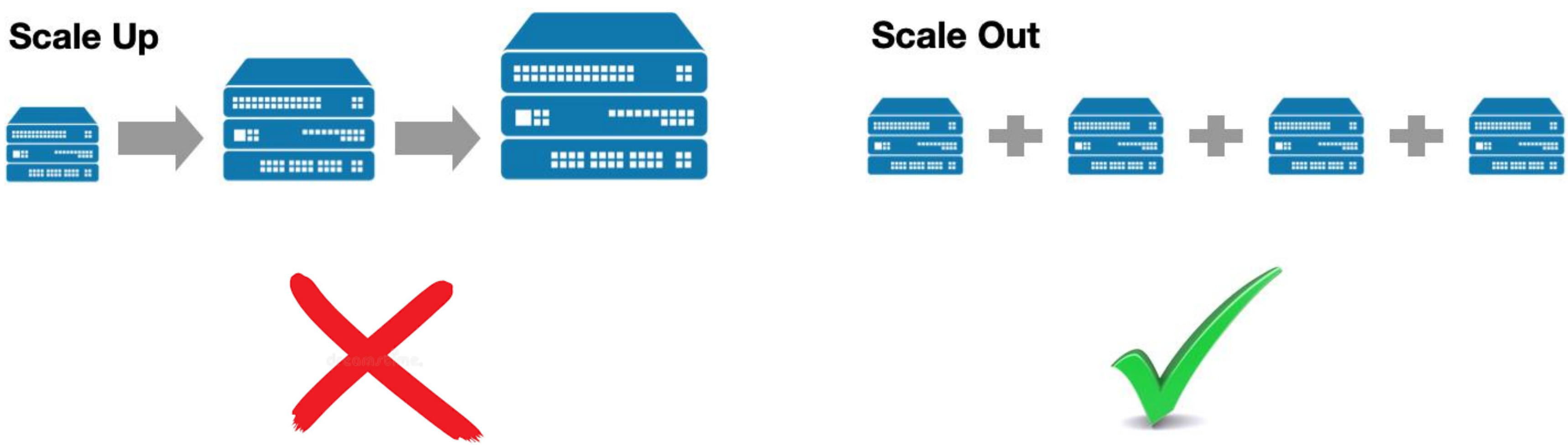
- Data center architecture
- **The four “Big Ideas”**
- Abstractions for big data systems

“Big Ideas” of Massive Data Processing in Data Centers

- Scale “out”, not “up”
  - scale ‘out’ = combining many cheaper machines; scale ‘up’= increasing the power of each individual machine
  - Also called ’horizontal’ vs ‘vertical’ scaling
- Seamless scalability
  - E.g. if processing a certain dataset takes 100 machine hours, ideal scalability is to use a cluster of 10 machines to do it in about 10 hours.
- Move processing to the data
  - Clusters have limited bandwidth: we should move the task to the machine where the data is stored
- Process data sequentially, avoid random access
  - Seeks are expensive, disk throughput is reasonable

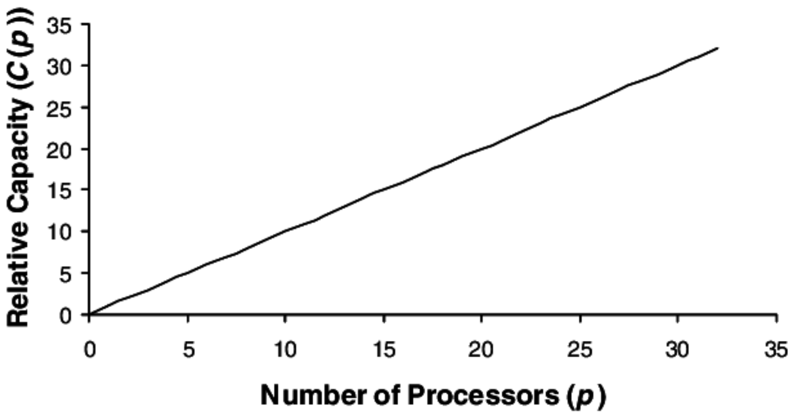
Scale “out”, not “up”

- Scaling up = adding further resources, like hard drives and memory, to increase the computing capacity of physical servers.
- Scaling out = adding more servers to your architecture to spread the workload across more machines.

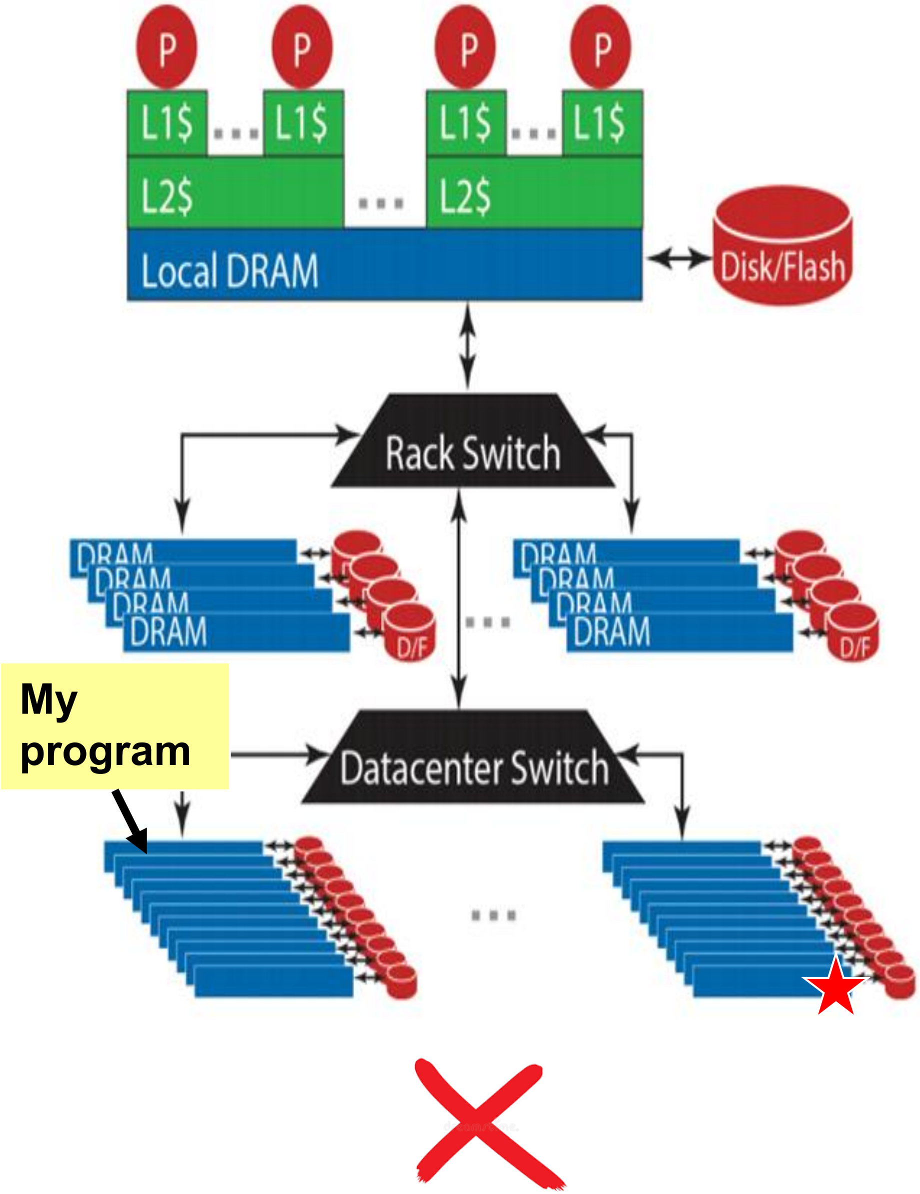


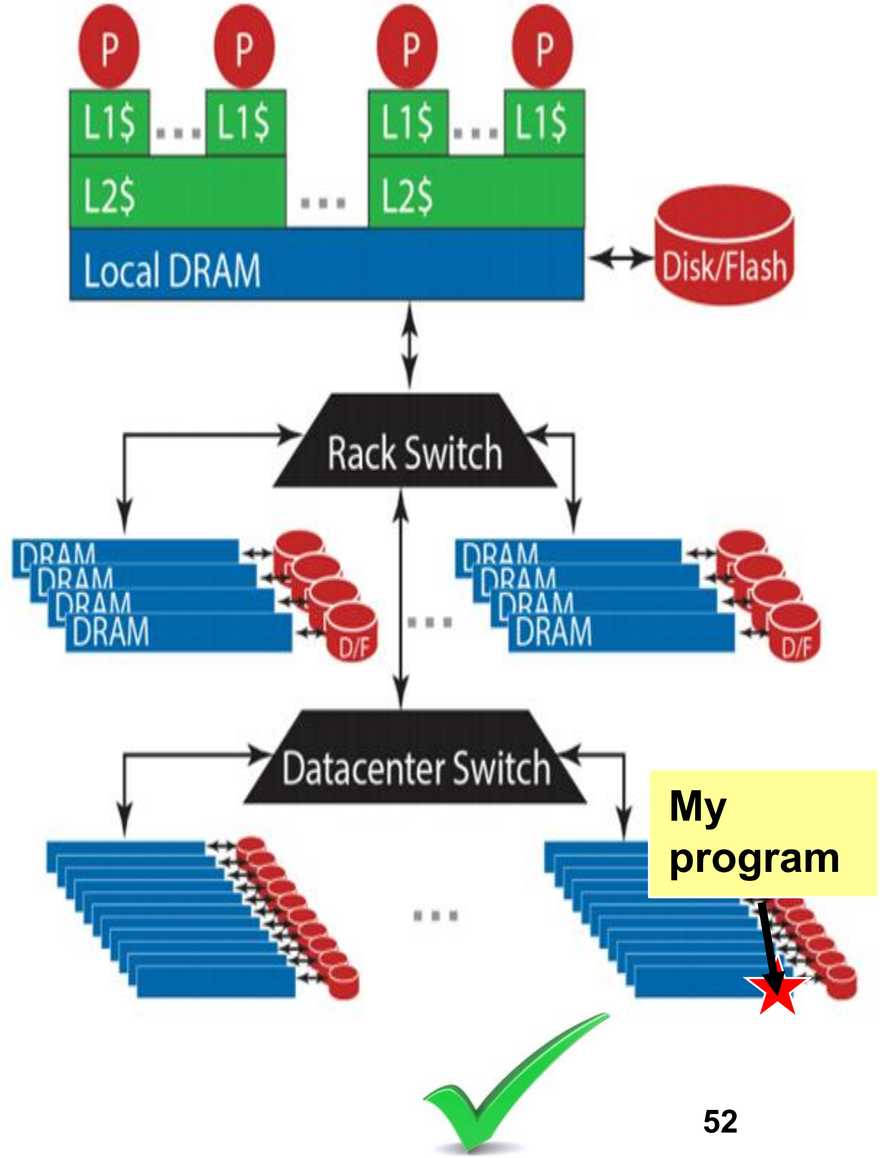
Seamless Scalability

- Think about aggregated bandwidth
  - 1 machine: 200MB/sec disk, 20GB/sec DRAM
  - 10 machine: 2 GB/sec disk, 200 GB/sec DRAM
  - 100 machine: 20 GB/sec disk, 2 TB/sec DRAM
- If our design scales linearly to #machine, we can trade more machines with better performance.



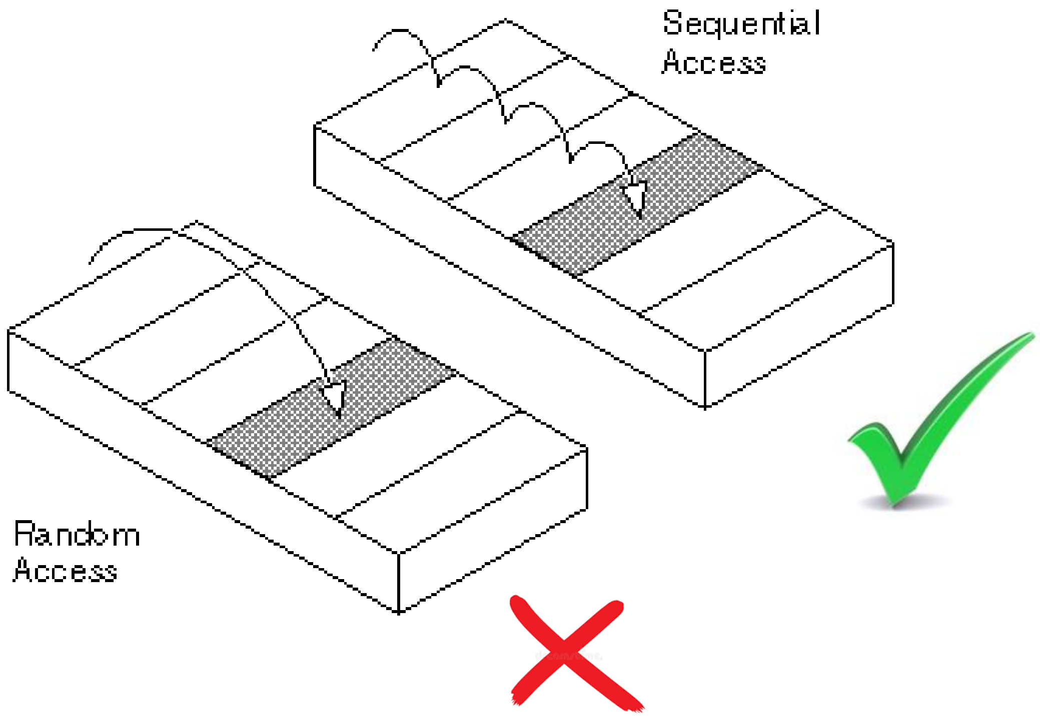
Move processing to the data





Sequential Accesses vs. Random Accesses

- Take hard disk as an example
- Random access: 10ms for 4KB = 400KB/sec
- Sequential access: 200MB/sec



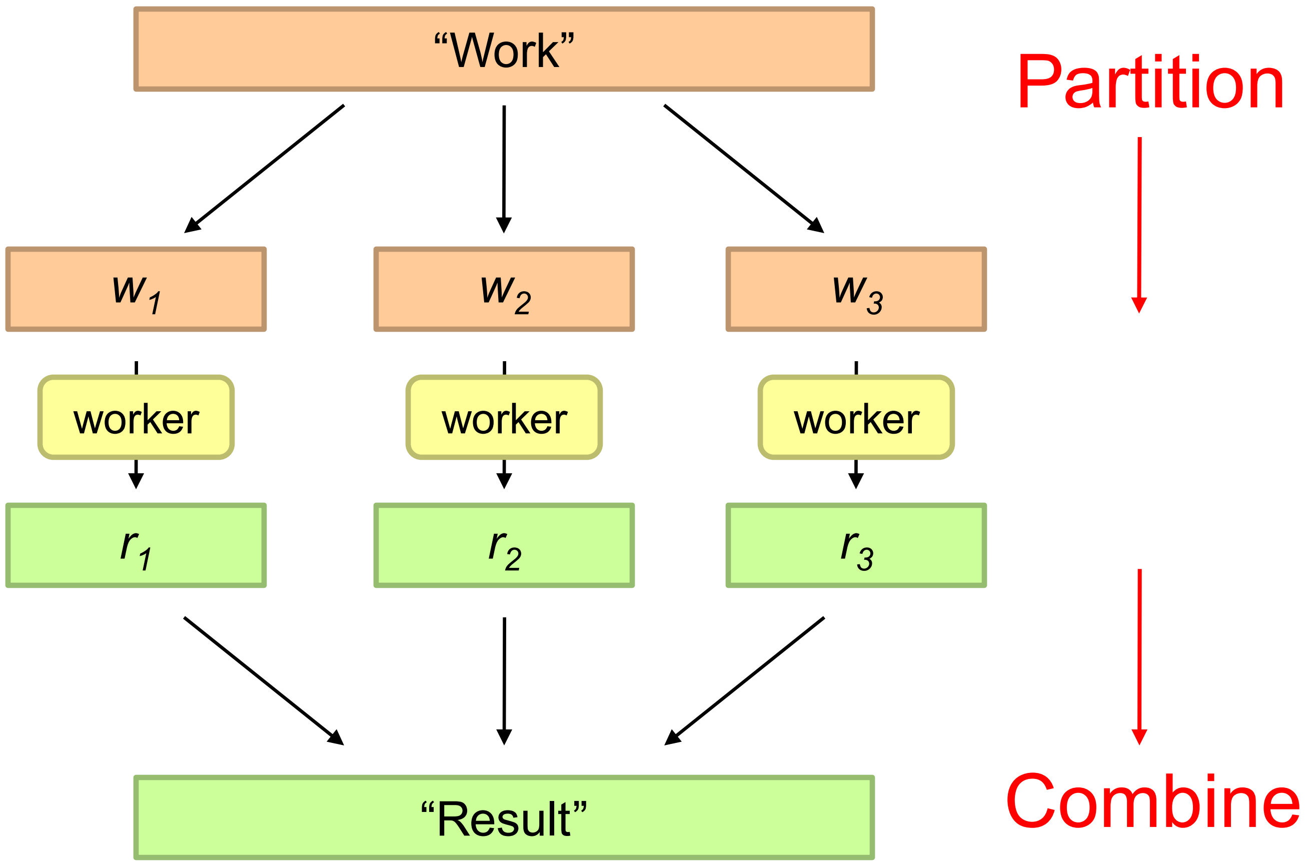
Outline

- Data center architecture
- The four “Big Ideas”
- **Abstractions for big data systems**

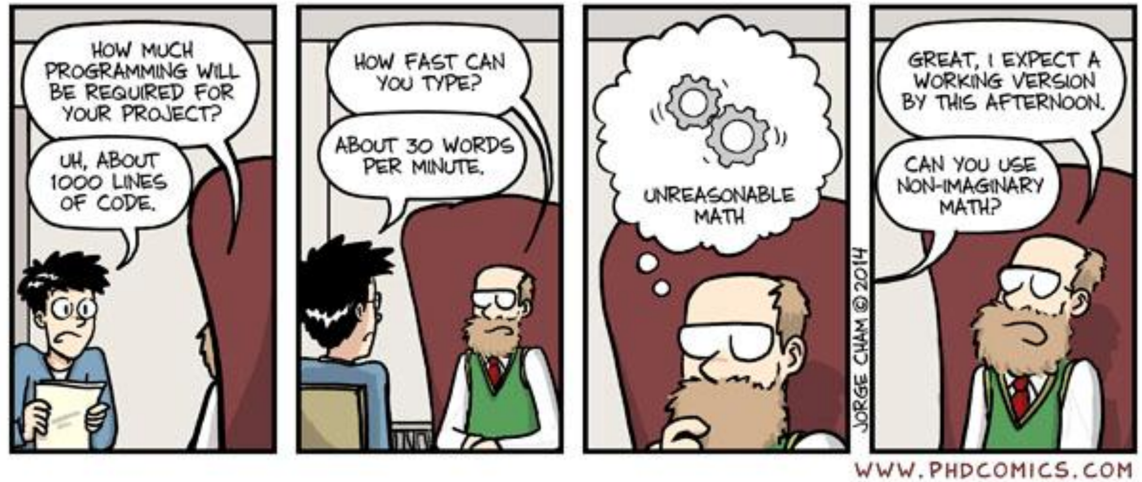
Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web
- Takes even more to **do** something useful with the data!
- Infrastructure: cloud + data centers
  - Cluster of commodity nodes
  - Commodity network (ethernet) to connect them
- Solution:
  - Parallelization + divide and conquer

Divide and Conquer



Challenges



- How do we assign work units to workers?
- What if we have more work units than workers?
- What if workers need to share partial results?
- How do we aggregate partial results?
- How do we know all the workers have finished?
- What if workers die/fail?

Challenge 1: Machine Failures

- One server may stay up 10 years (~3650 days)
- If you have 3650 servers, expect to loose 1/day
- People estimated Google had ~2.5 million machines in 2016
  - ~700 machines fail every day!

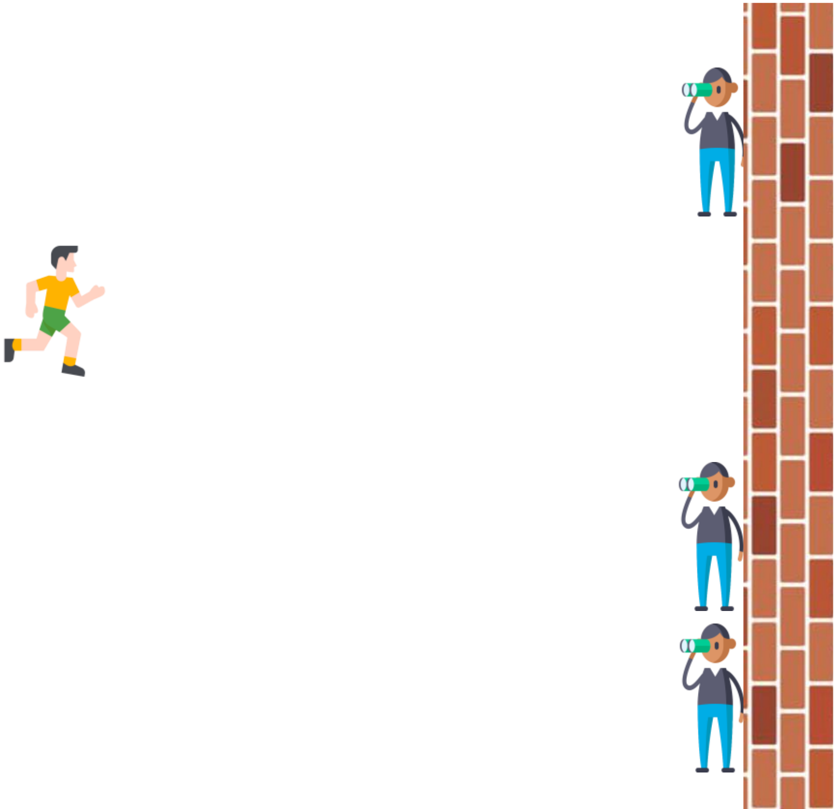
Challenge 2: Synchronization

- Difficult because
  - We don’t know the order in which workers run
  - We don’t know when workers interrupt each other
  - We don’t know when workers need to communicate partial results
  - We don’t know the order in which workers access shared data
- Thus, we need (note: not required knowledge for class)
  - Semaphores (lock, unlock)
  - Conditional variables (wait, notify, broadcast)
  - Barriers
- Still, lots of problems:
  - Deadlock, livelock, race conditions...
  - Dining philosophers, sleeping barbers...

Source: Ricardo Guimarães Herrmann

Barrier

- Simple tool used when multiple processes are running at the same time
- It ensures that every process must stop at this point until all processes have reached the barrier



Challenge 3: Programming Difficulty

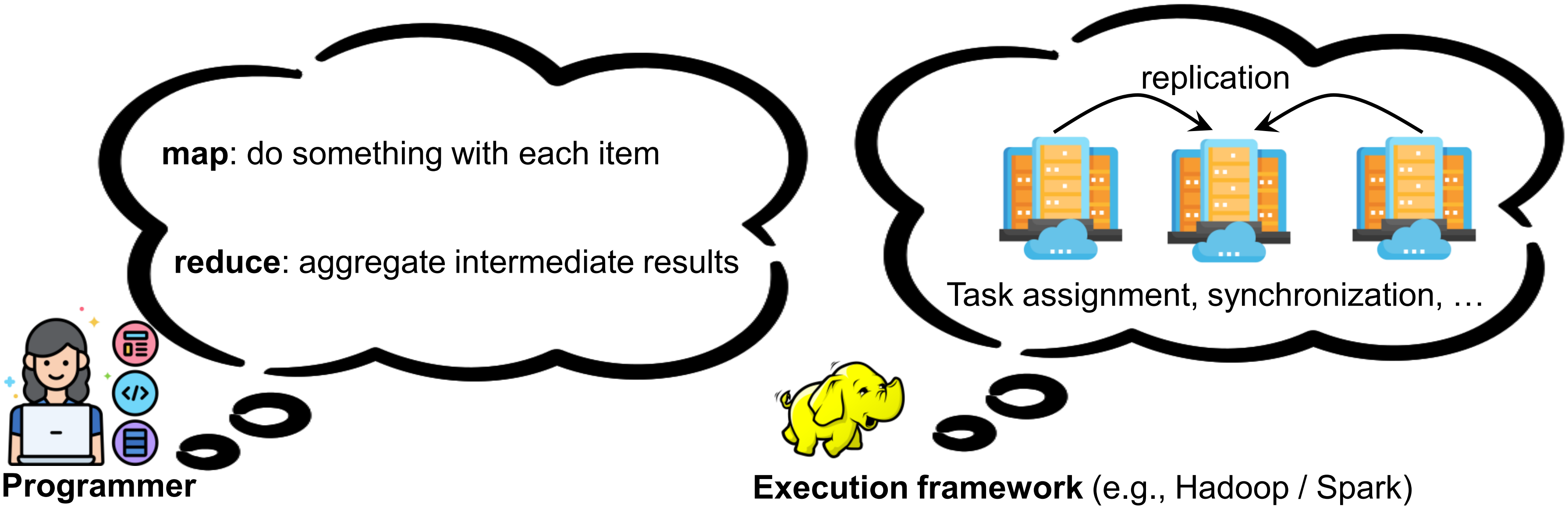
- Concurrency is difficult to reason about
  - At the scale of datacenters and across datacenters
  - In the presence of failures
  - In terms of multiple interacting services
- Not to mention debugging…
- The reality:
  - Lots of one-off solutions, custom code
  - Write you own dedicated library, then program with it
  - Burden on the programmer to explicitly manage everything

The datacenter *is* the computer

- It’s all about the right level of abstraction
  - Moving beyond the single machine architecture
  - What’s the “instruction set” (or “API”) of the datacenter computer?
- Hide system-level details from the developers
  - No more race conditions, lock contention, etc.
  - No need to explicitly worry about reliability, fault tolerance, etc.
- Separating the *what* from the *how*
  - Developer specifies the computation that needs to be performed
  - Execution framework (“runtime”) handles actual execution

API for Data Center

- This interface is the "API" of the datacenter "computer"



Take-away

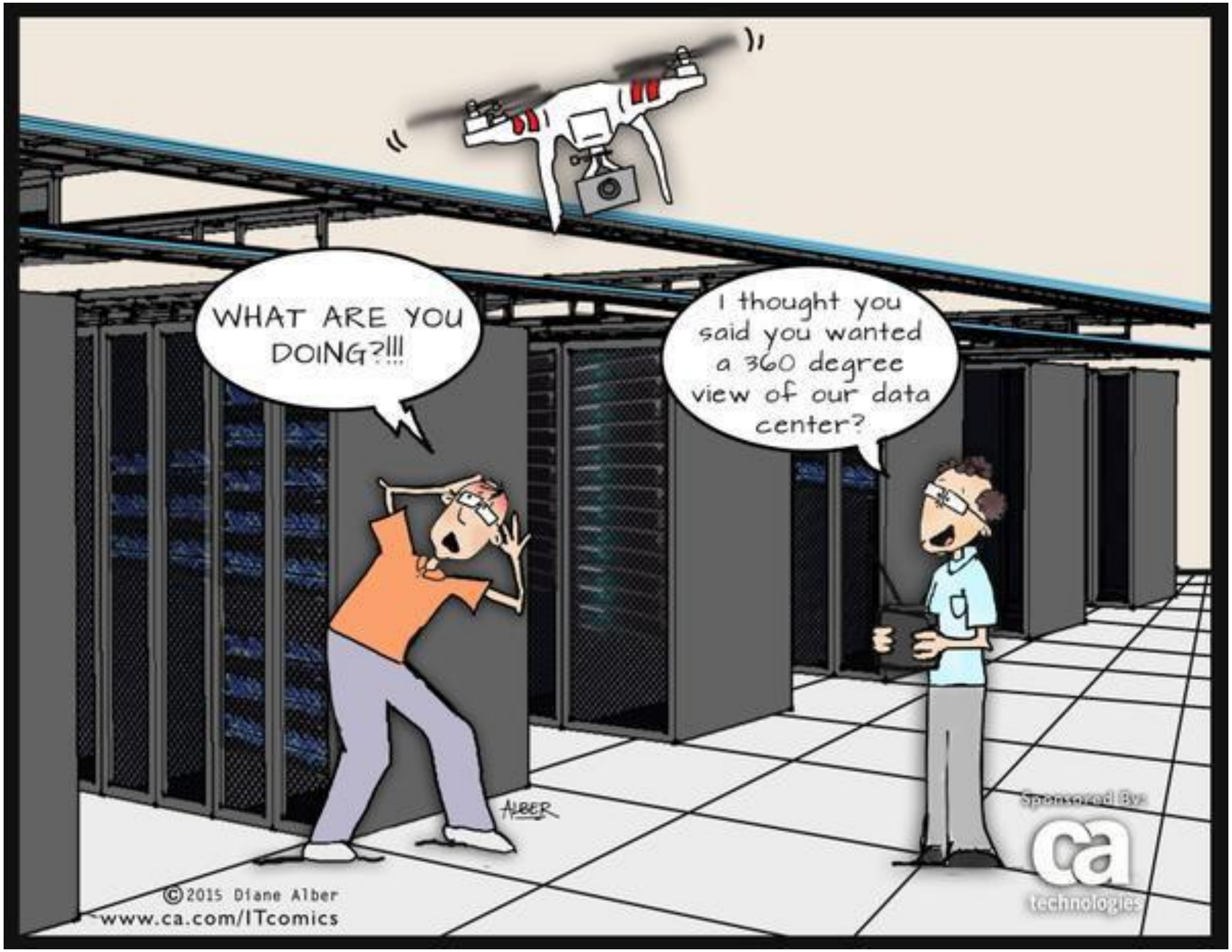
- “The data centre is the computer”.
- Four “big ideas” are the design principles of big data systems on the current hardware.
- Further readings:
  - Chapter 1. Jimmy Lin and Chris Dyer. 2020. Data-Intensive Text Processing with Mapreduce. Morgan and Claypool Publishers. https://lintool.github.io/MapReduceAlgorithms/MapReduce-book-final.pdf

Take-away in the AI Era

- Human Skills This Chapter Builds (AI Cannot Replace These)
- Reasoning from hardware reality
- Order-of-magnitude thinking
- System-level bottleneck identification
- Designing under constraints
- (Bad system design cannot be fixed by better code — even if the code is written by AI.)

AI writes code.
Humans decide systems.

Questions?


WHAT ARE YOU DOING?!!!
I thought you said you wanted a 360 degree view of our data center?
ALBER
©2015 Diane Alber
www.ca.com/ITcomics