

Advanced Computer Networks

Transport in Data Center Networks (I)

<https://pages.cs.wisc.edu/~mgliu/CS740/F26/index.html>

Ming Liu
mgliu@cs.wisc.edu

Outline

- Last lecture
 - SmartNICs
- Today
 - Transport in Data Center Networks (I)
- Announcements
 - Lab2 due 11/06/2026 11:59 PM
 - Midterm report due 11/10/2026 11:59 PM

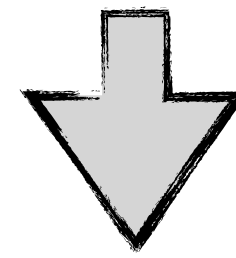
A Few Words on Project

A Few Words on Project

You'll have ~5 weeks to complete the course project.

A Few Words on Project

You'll have ~5 weeks to complete the course project.

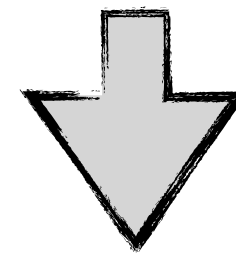


Please reshape it into a small-sized problem:

- **It's okay to answer just one question**

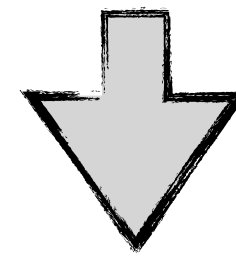
A Few Words on Project

You'll have ~5 weeks to complete the course project.



Please reshape it into a small-sized problem:

- It's okay to answer just one question



Course project = Testbed + Implementation + Evaluation

- Testbed: what are the hardware platforms, software frameworks, and workloads?
- Implementation: where are your codes running?
- Evaluation: what questions do you plan to answer and how?

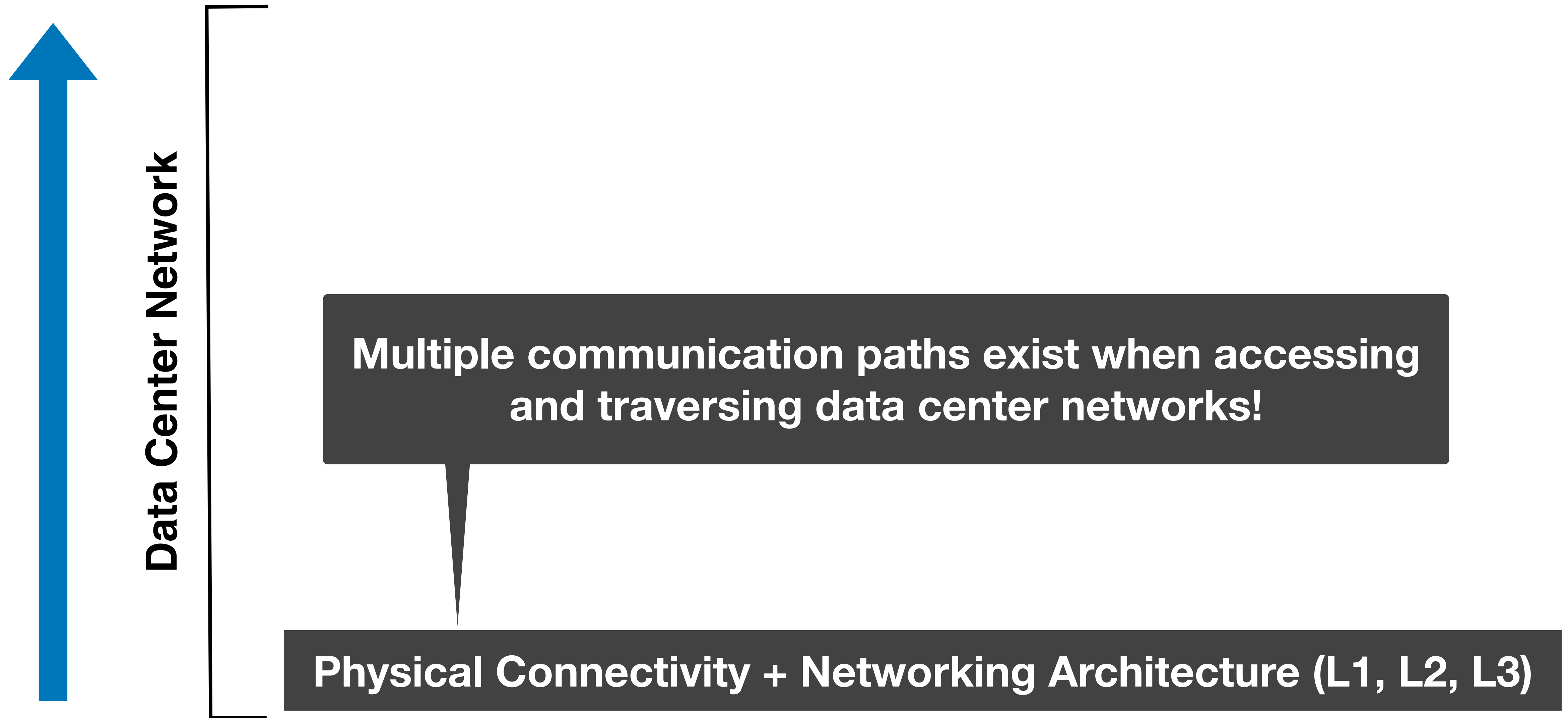
A Few Words on Project

You'll have ~5 weeks to complete the course project.

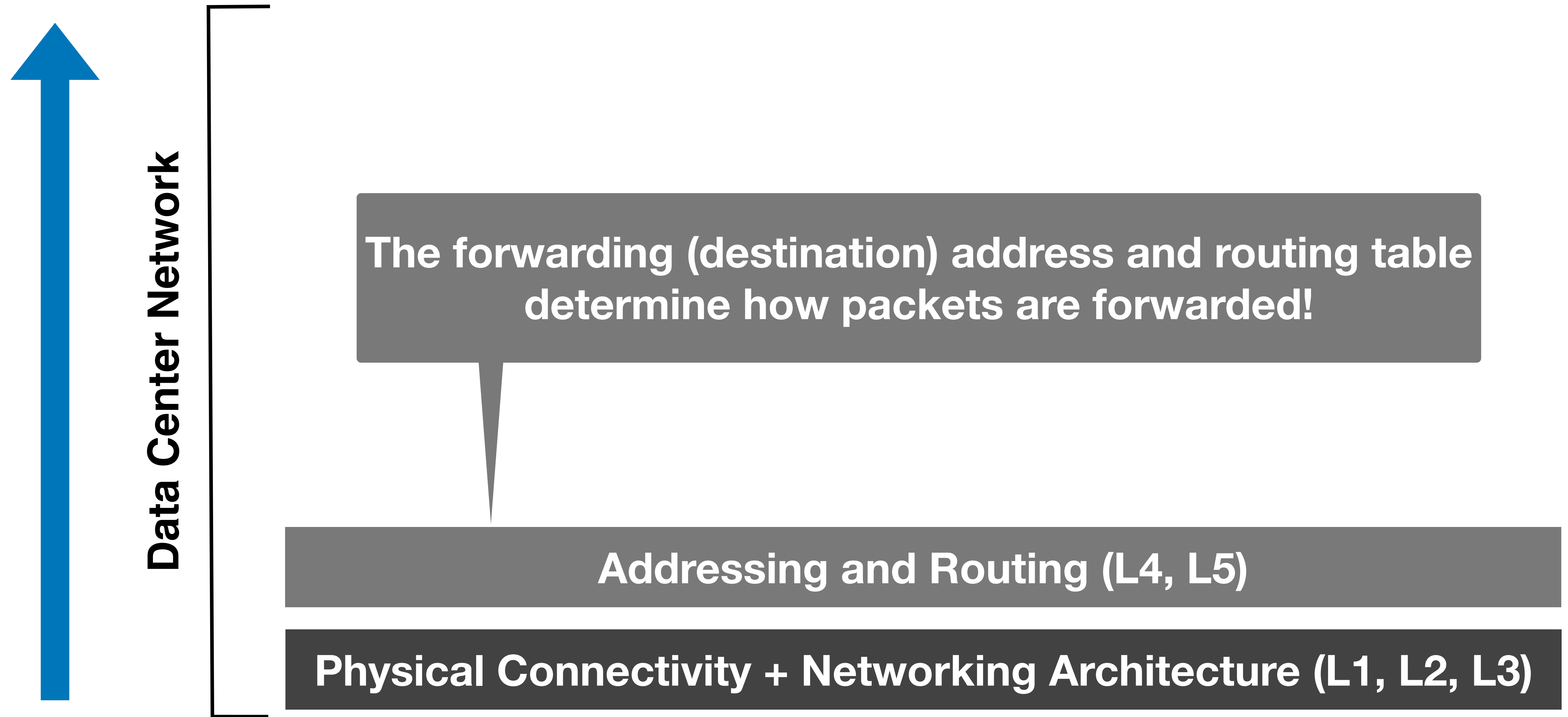
Midterm report:

- Refine your problem statement concretely;
 - Finalize your testbed;
 - Specify what you will implement;
 - Specify your outcome;
-
- Implementation: where are your codes running?
 - Evaluation: what questions do you plan to answer and how?

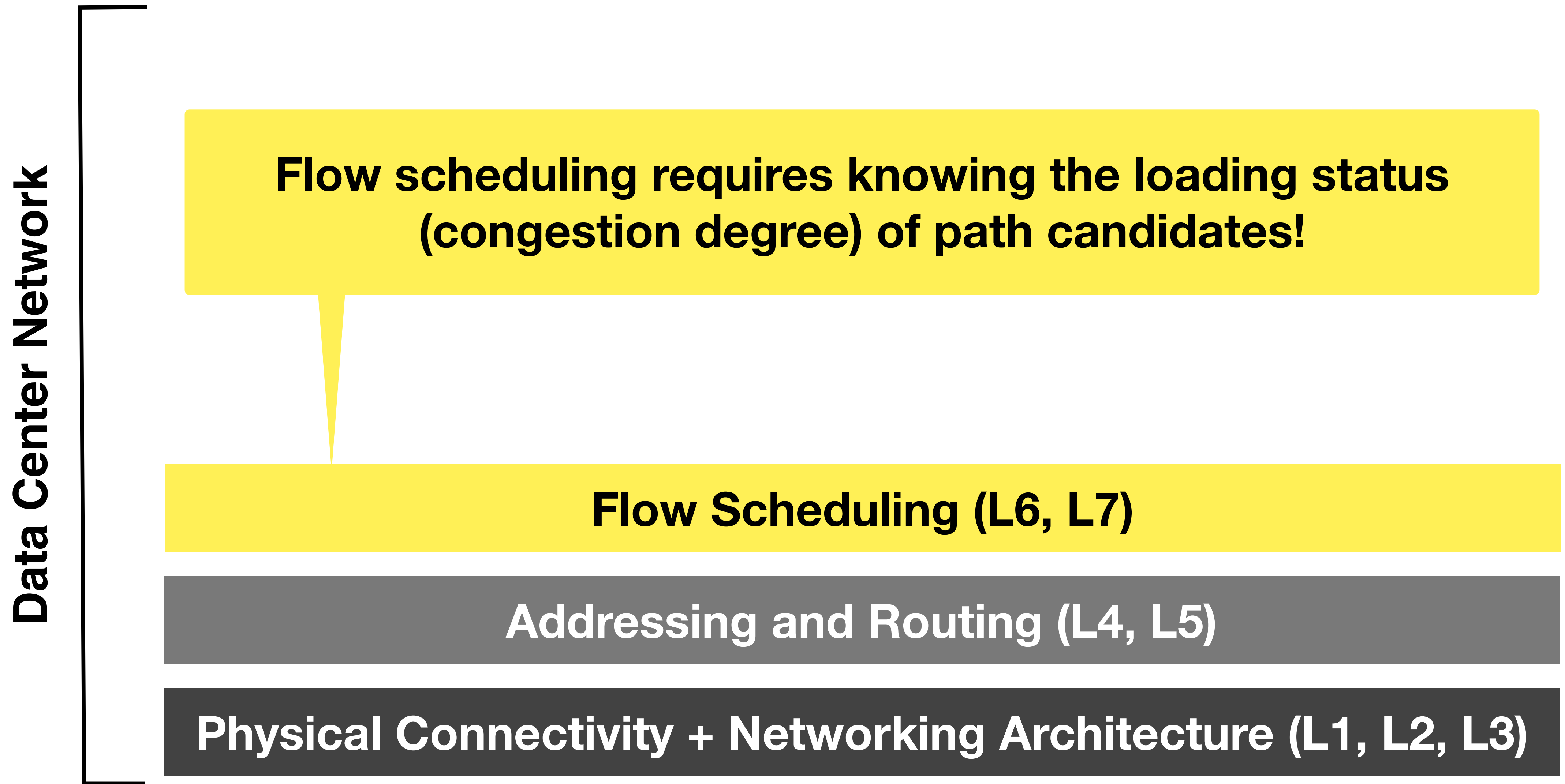
Where we are?



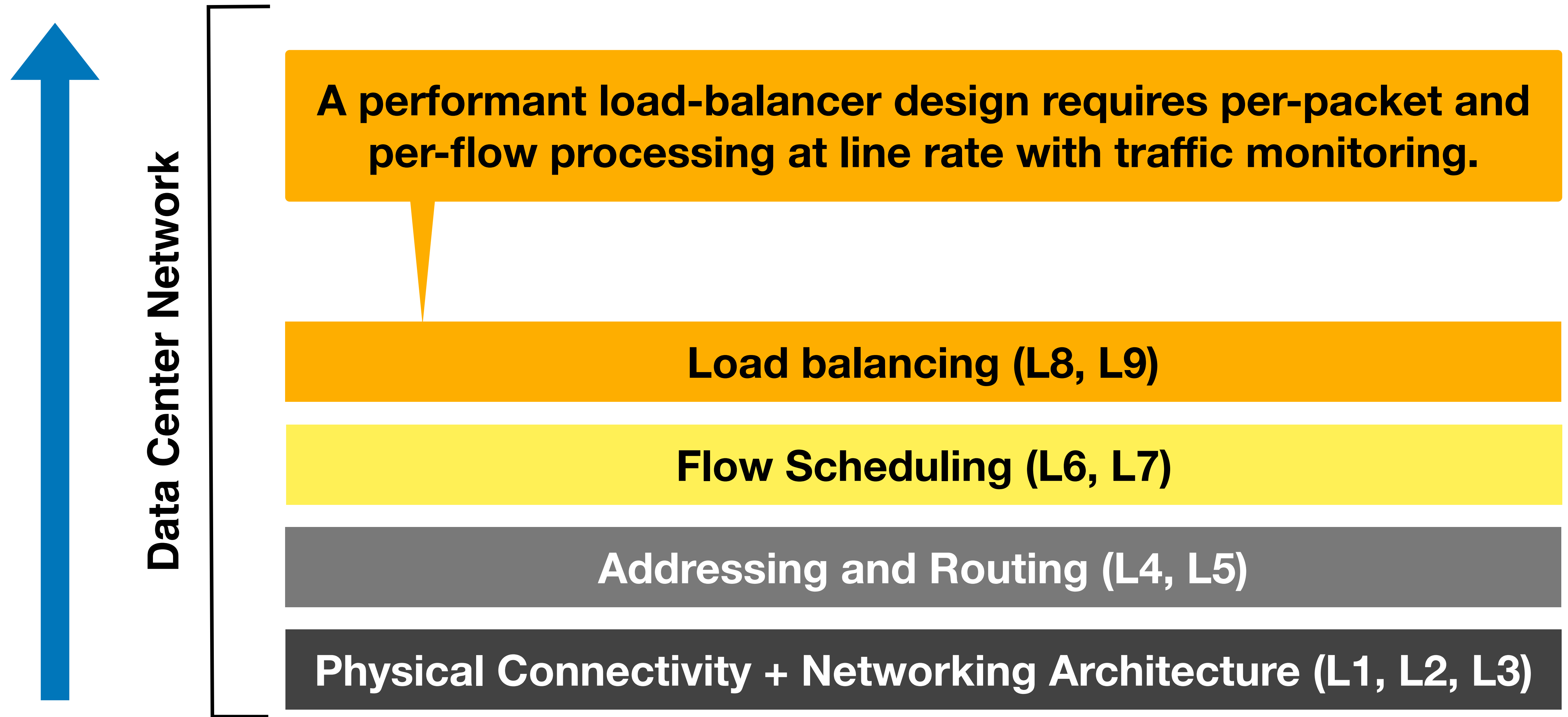
Where we are?



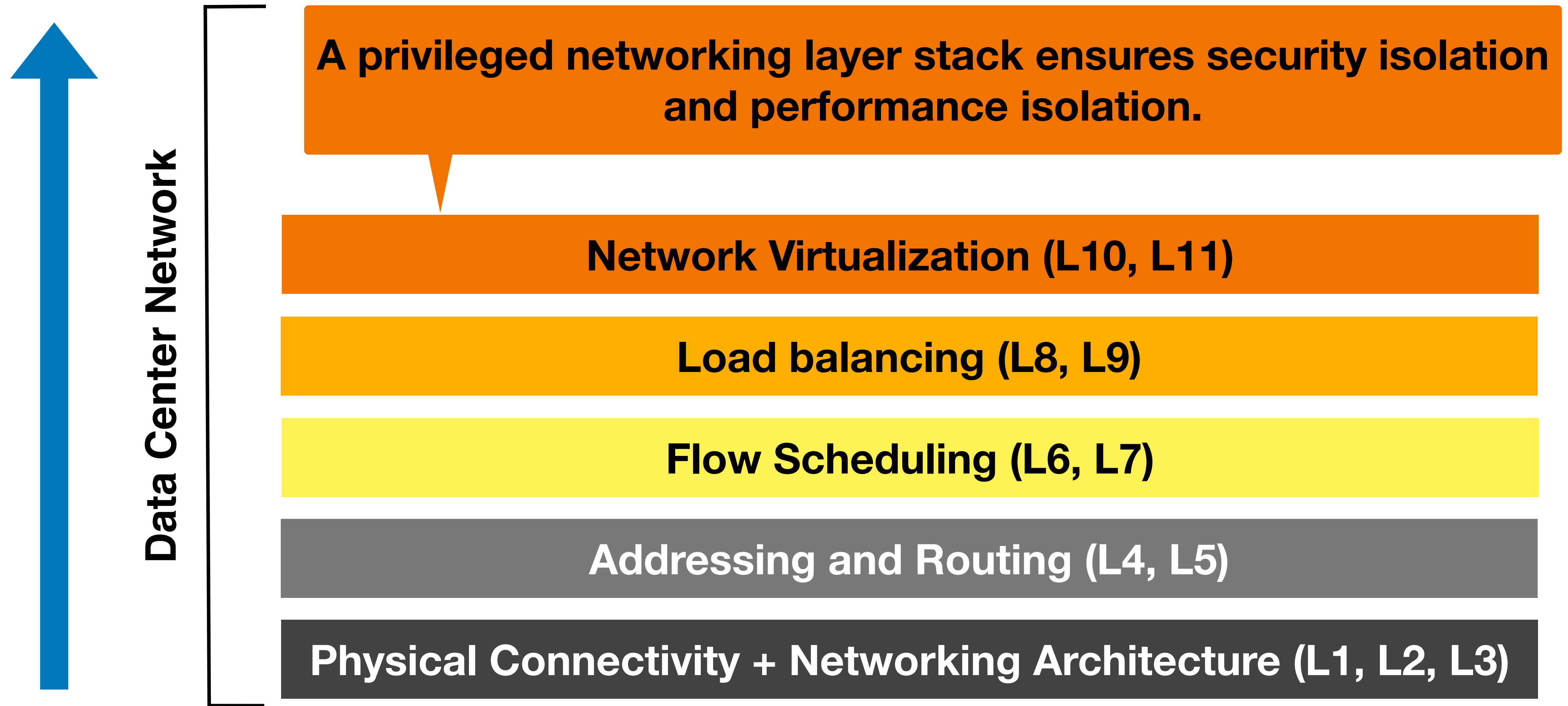
Where we are?



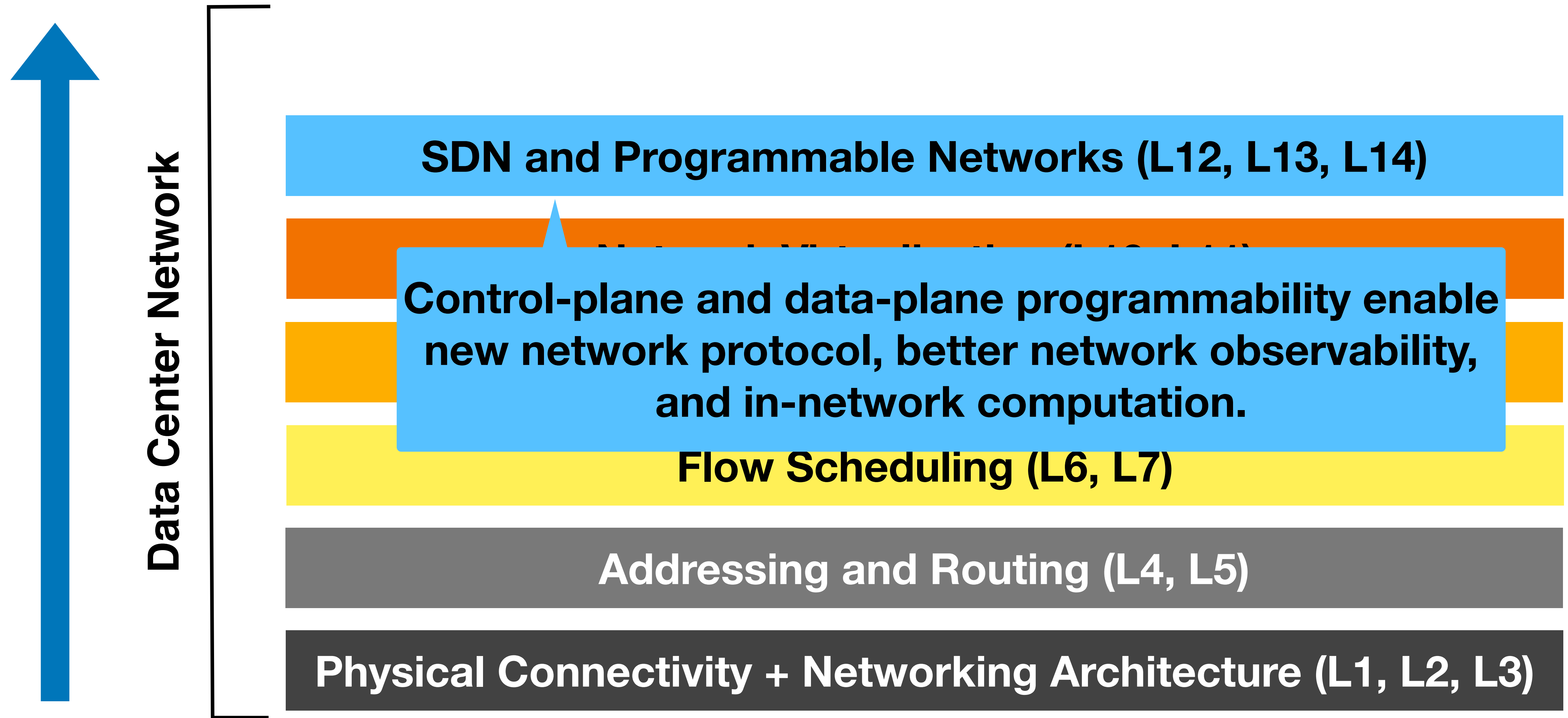
Where we are?



Where we are?



Where we are?



TCP is the dominant transport protocol in data centers.

**TCP is the dominant transport protocol in
data centers, but**

The Problem

- TCP cannot satisfy data center application demands
 - Throughput drops under burst
 - High tail latency
 - Inferior fairness guarantees
 - ...

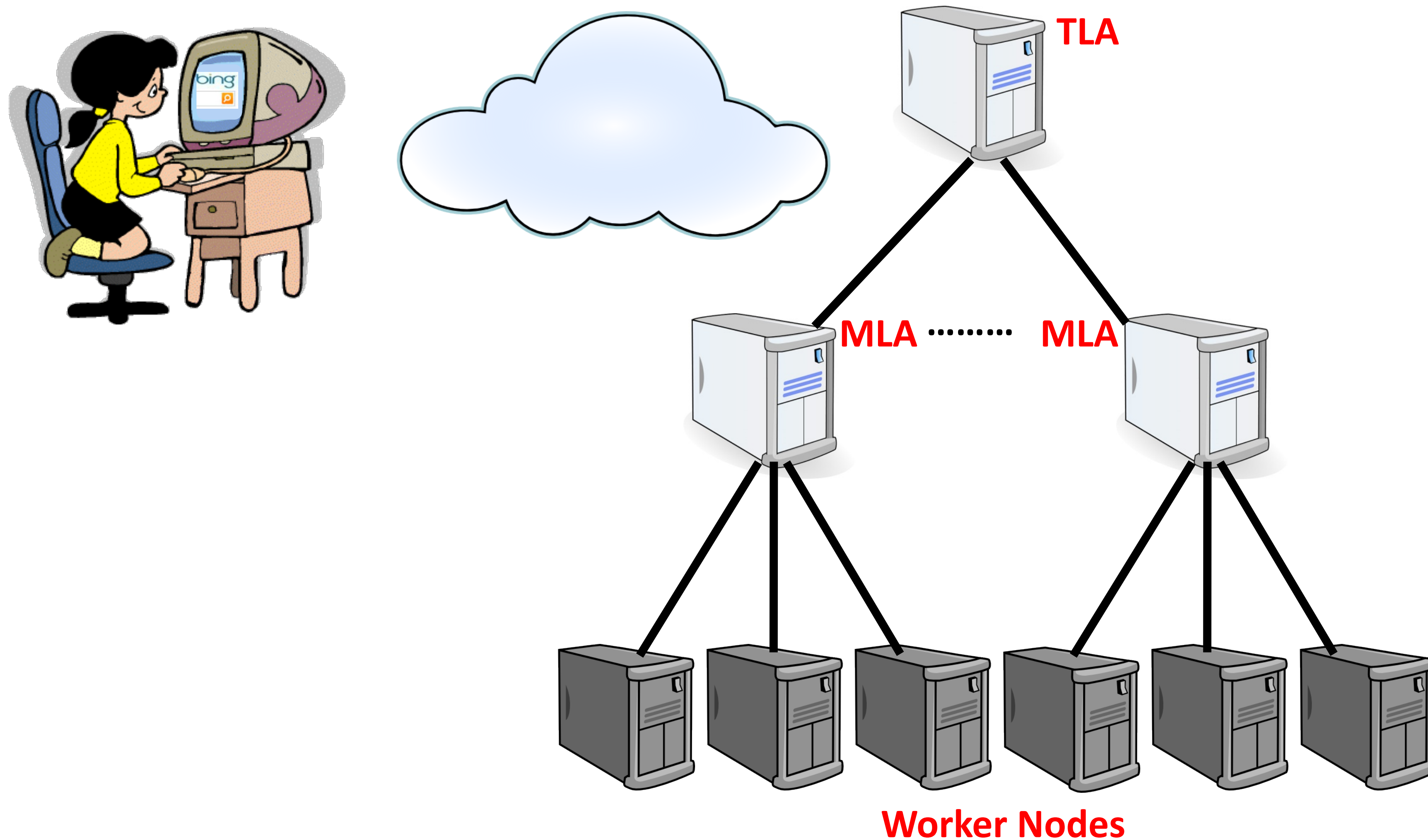
Limitations of Prior Solutions

- #1: Ad-hoc, inefficient, and expensive solutions
 - E.g., Work for app A, not app B
- #2: No solid understanding of consequences and tradeoffs
 - Not capture data center application characteristics
 - Not capture data center network characteristics

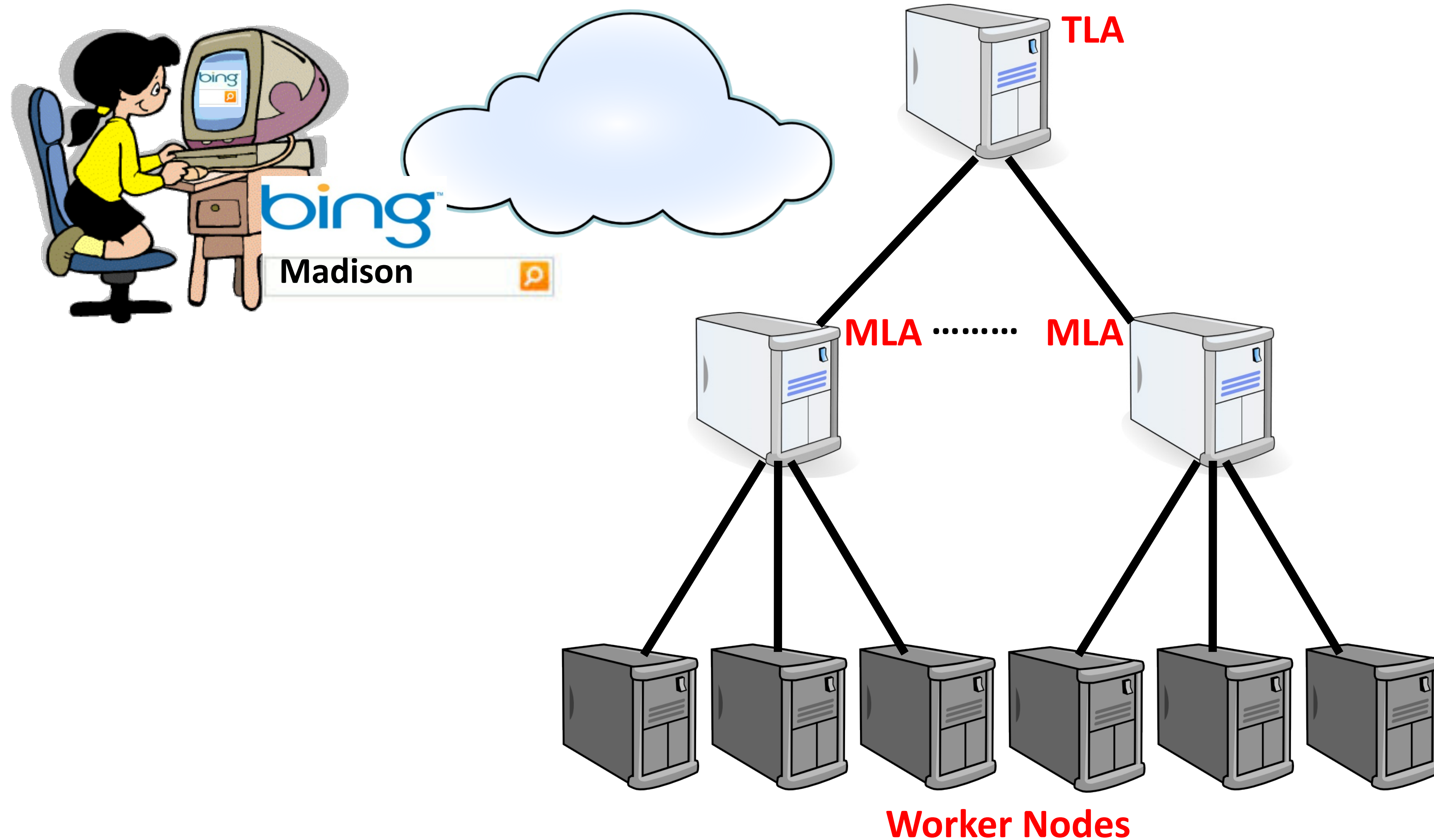
Microsoft Bing Search

- Large scale
 - Measurements from 6000 server production cluster
 - Collect more than 150TB of compressed data over a month
- Extensive instrumentation
 - Application-level
 - Socket-level
 - Selected packet-level

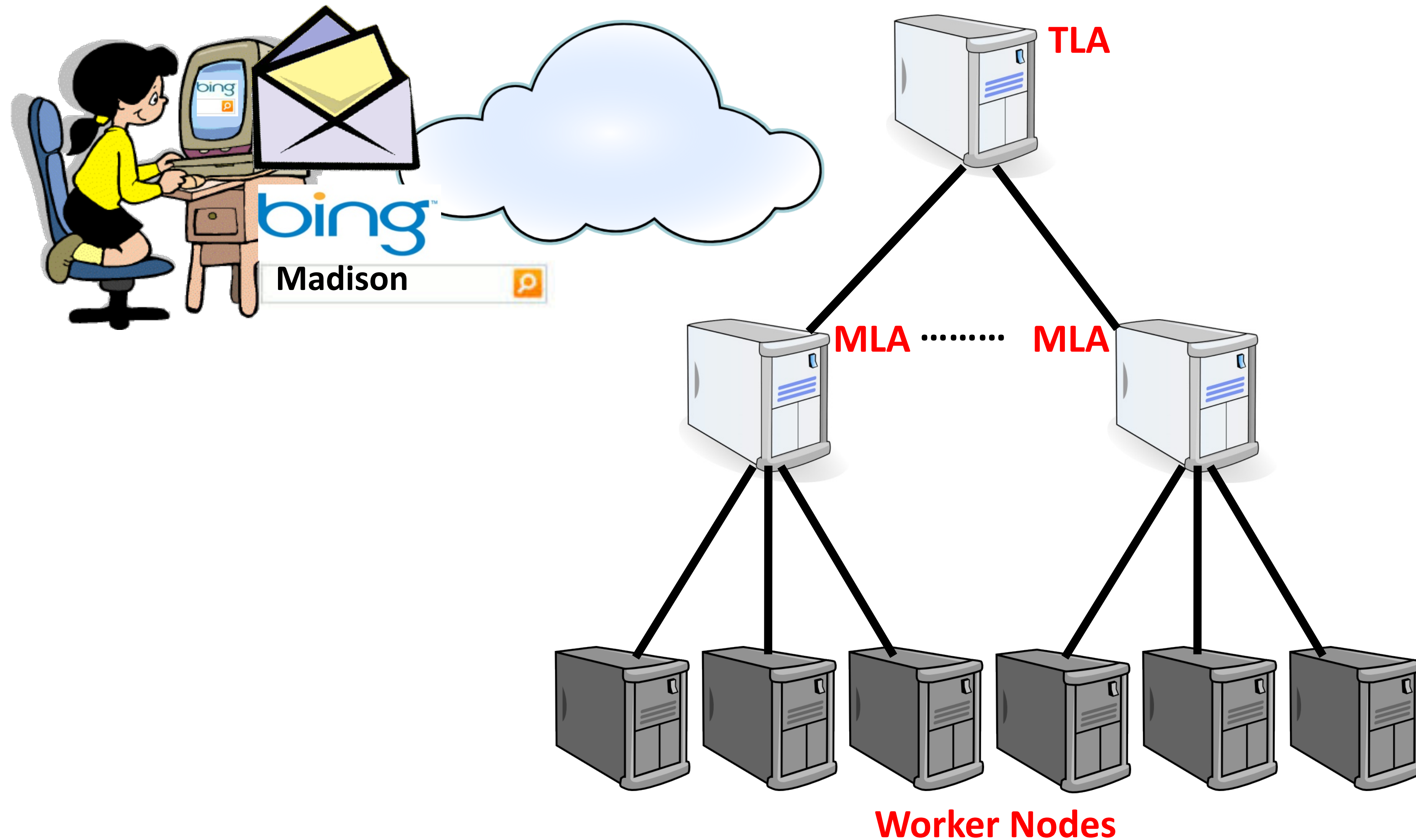
Partition/Aggregation Application Structure



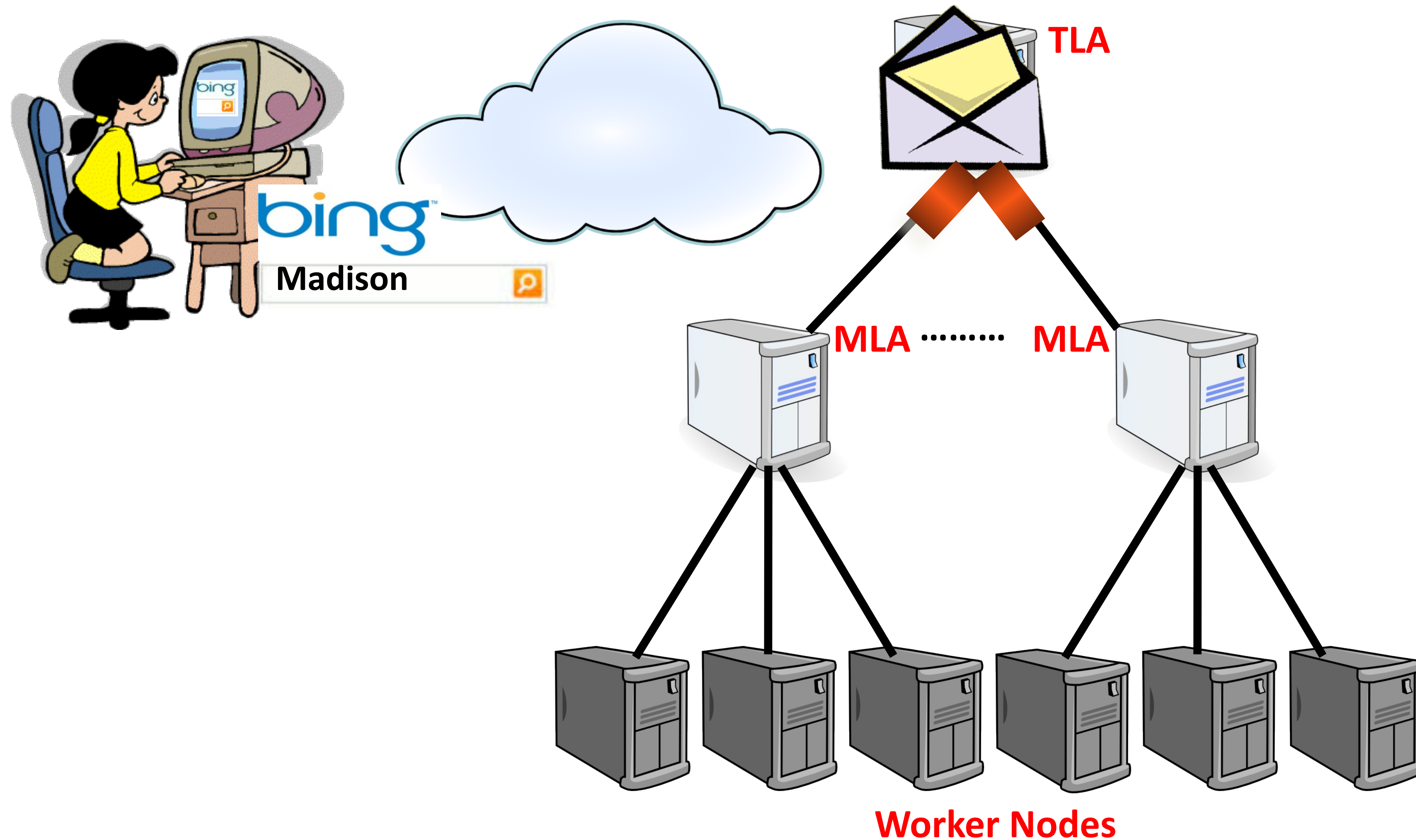
Partition/Aggregation Application Structure



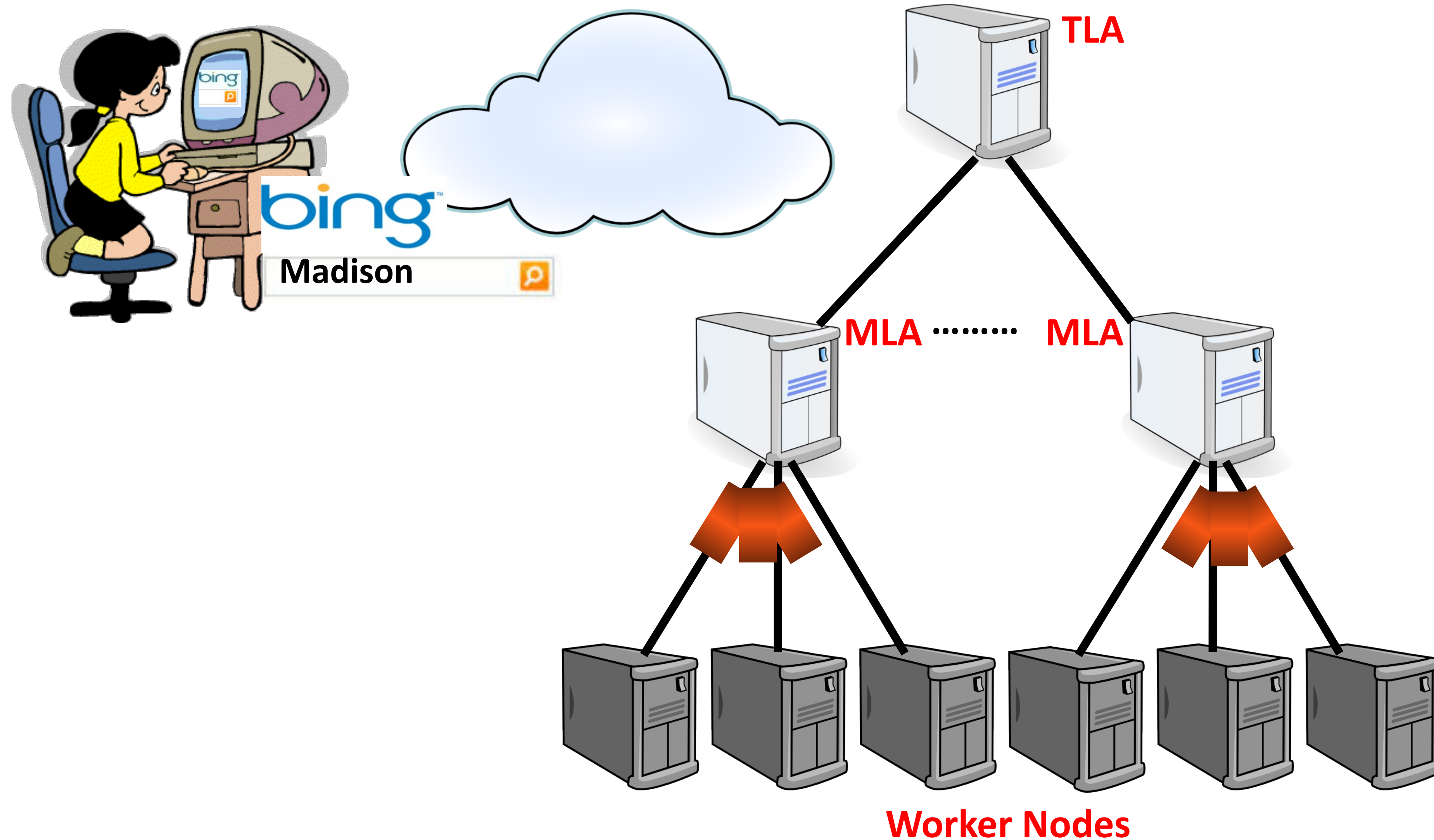
Partition/Aggregation Application Structure



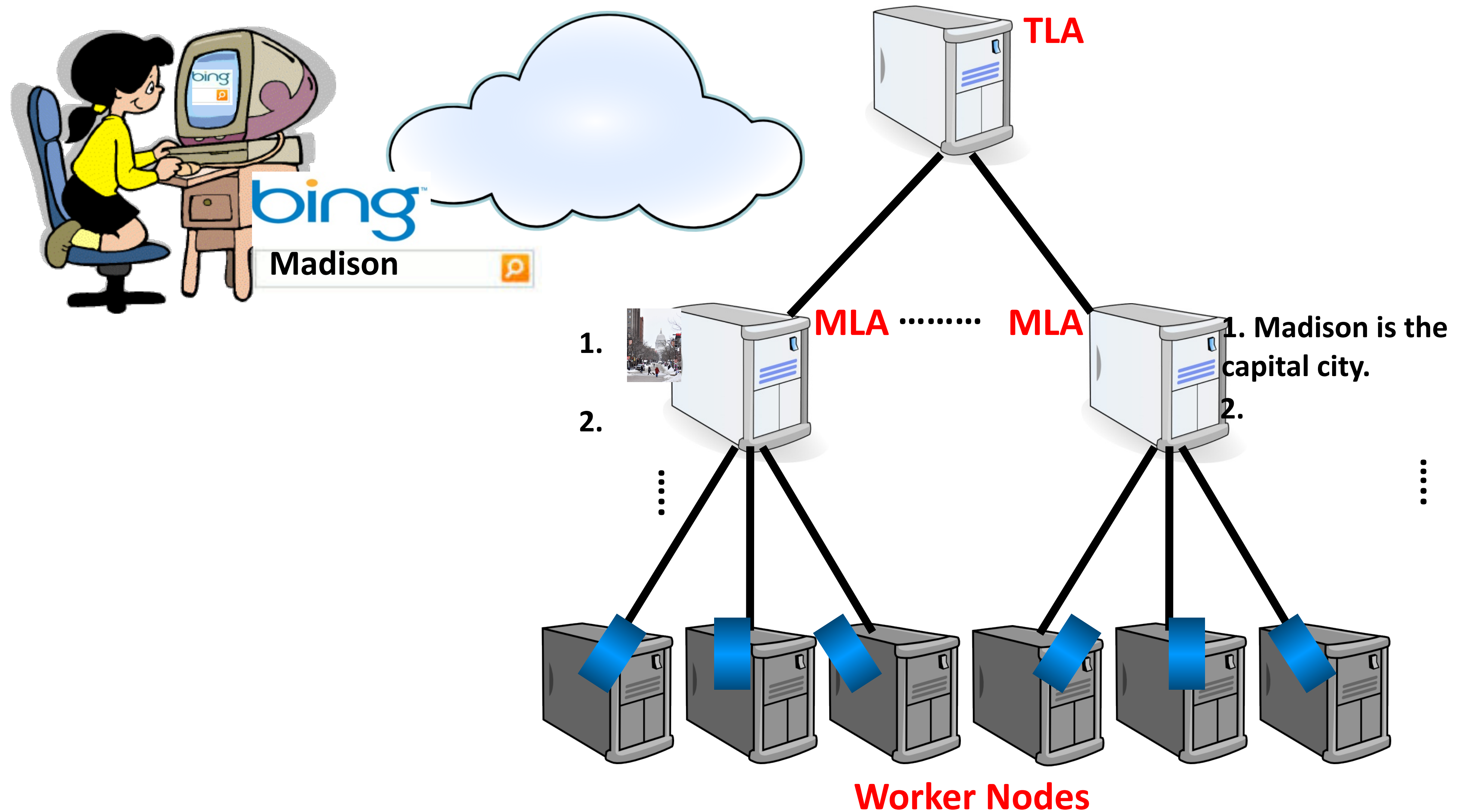
Partition/Aggregation Application Structure



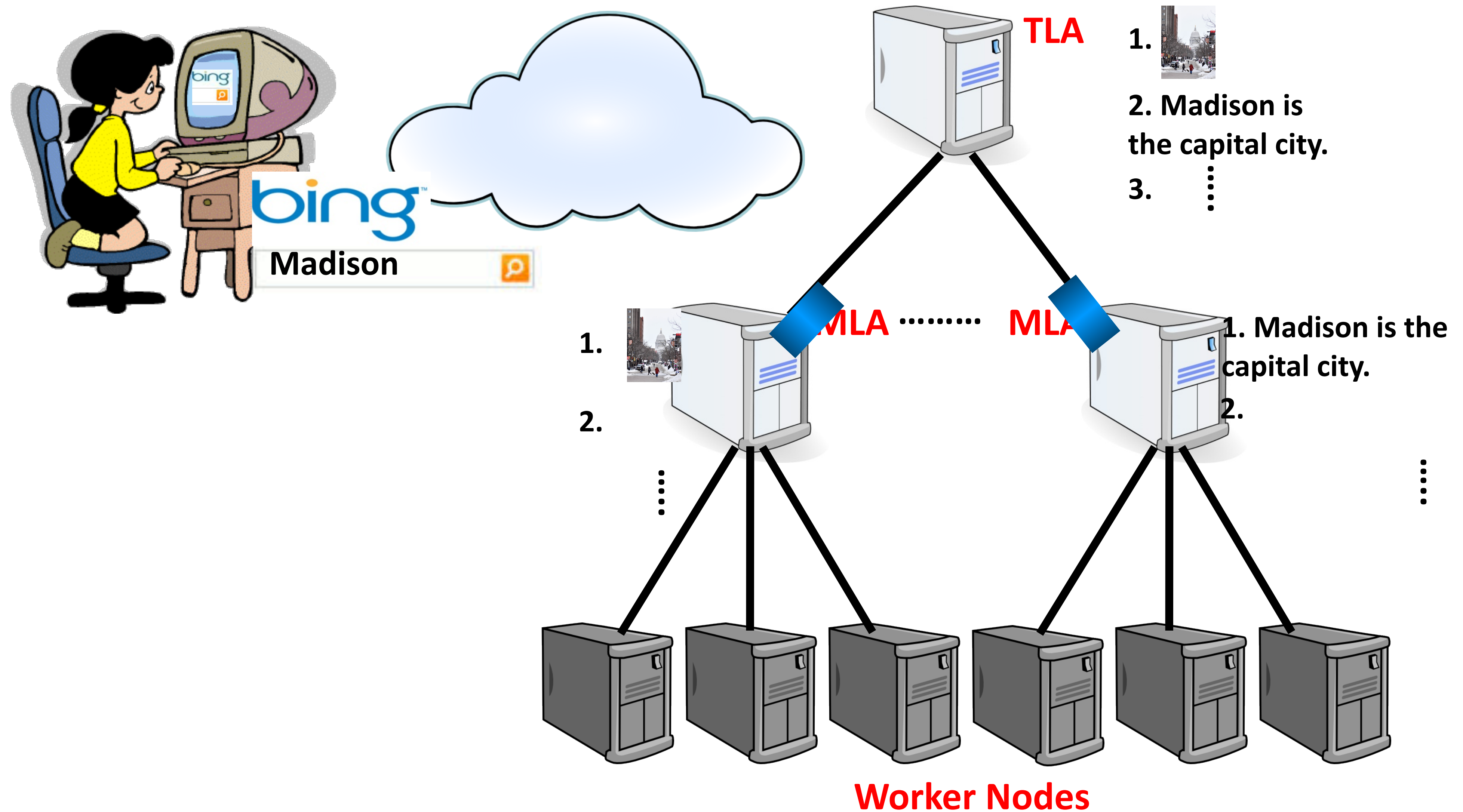
Partition/Aggregation Application Structure



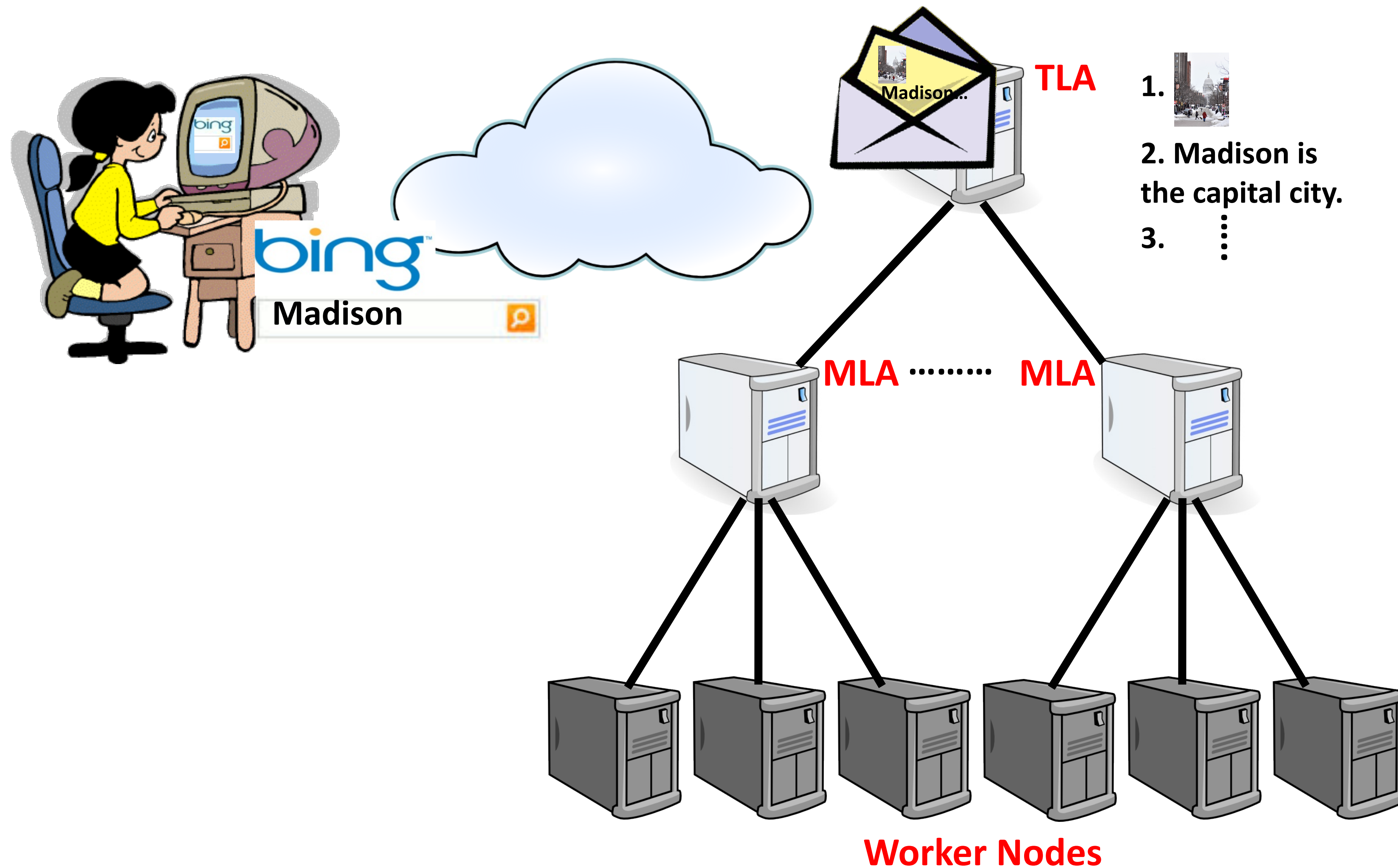
Partition/Aggregation Application Structure



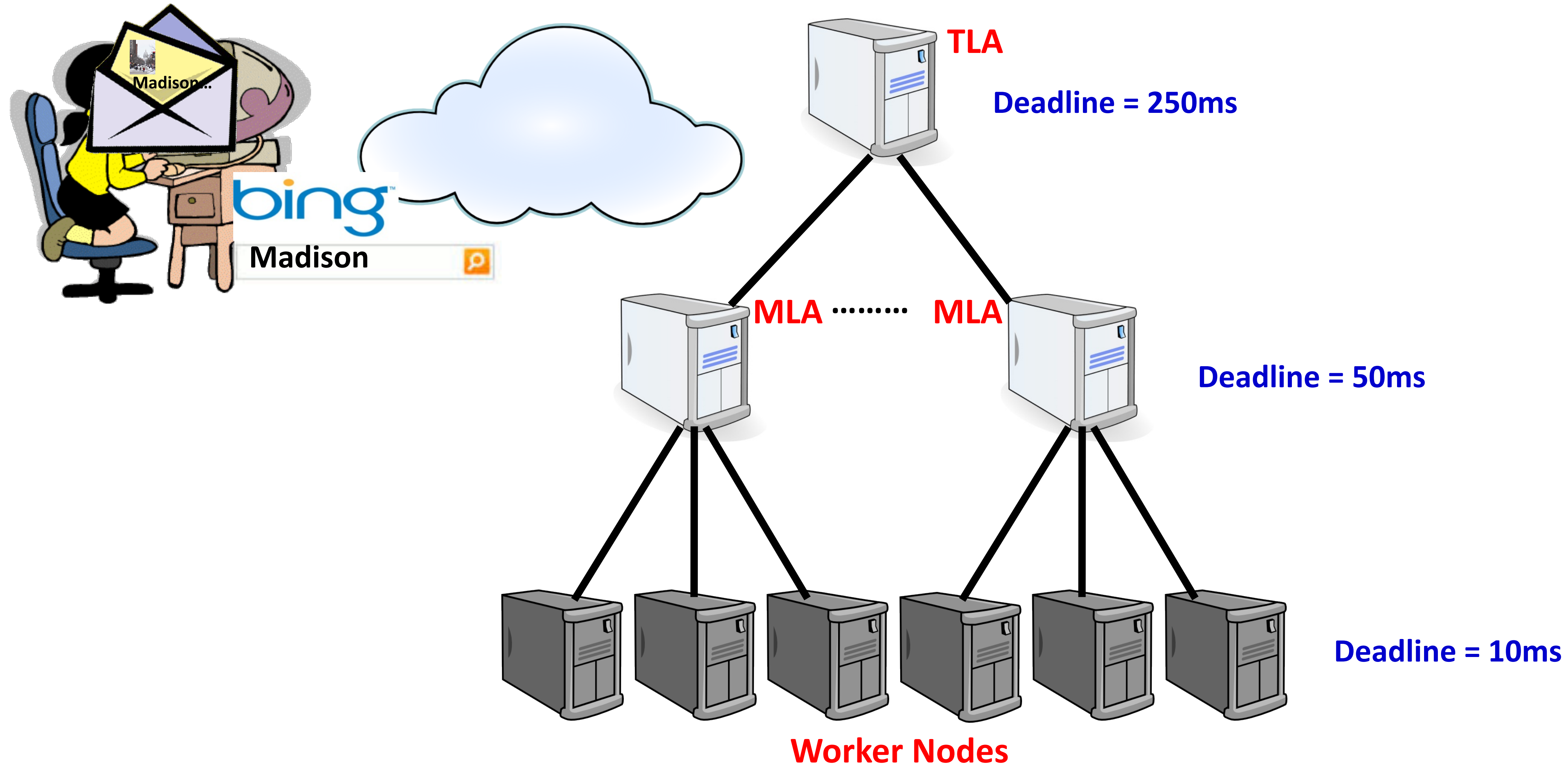
Partition/Aggregation Application Structure



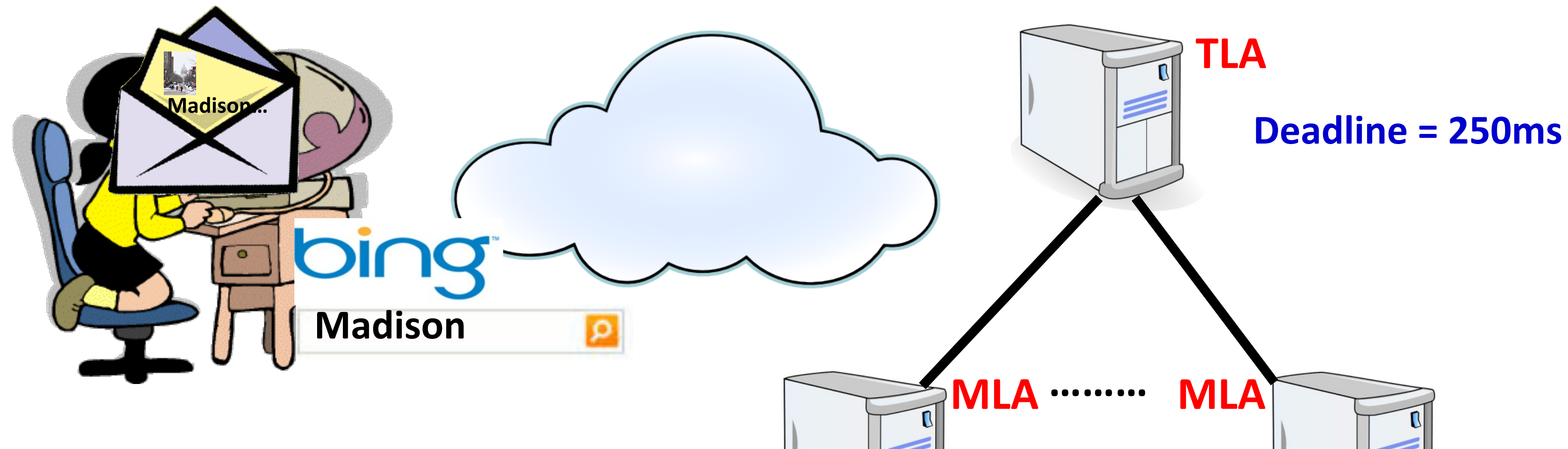
Partition/Aggregation Application Structure



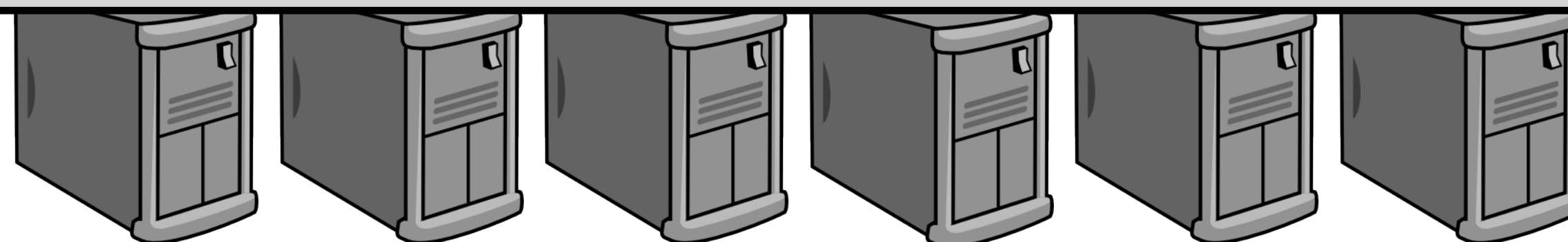
Partition/Aggregation Application Structure



Partition/Aggregation Application Structure



- Time is money => Strict deadline (SLAs)
- Missed deadline => Lower quality results



Deadline = 10ms

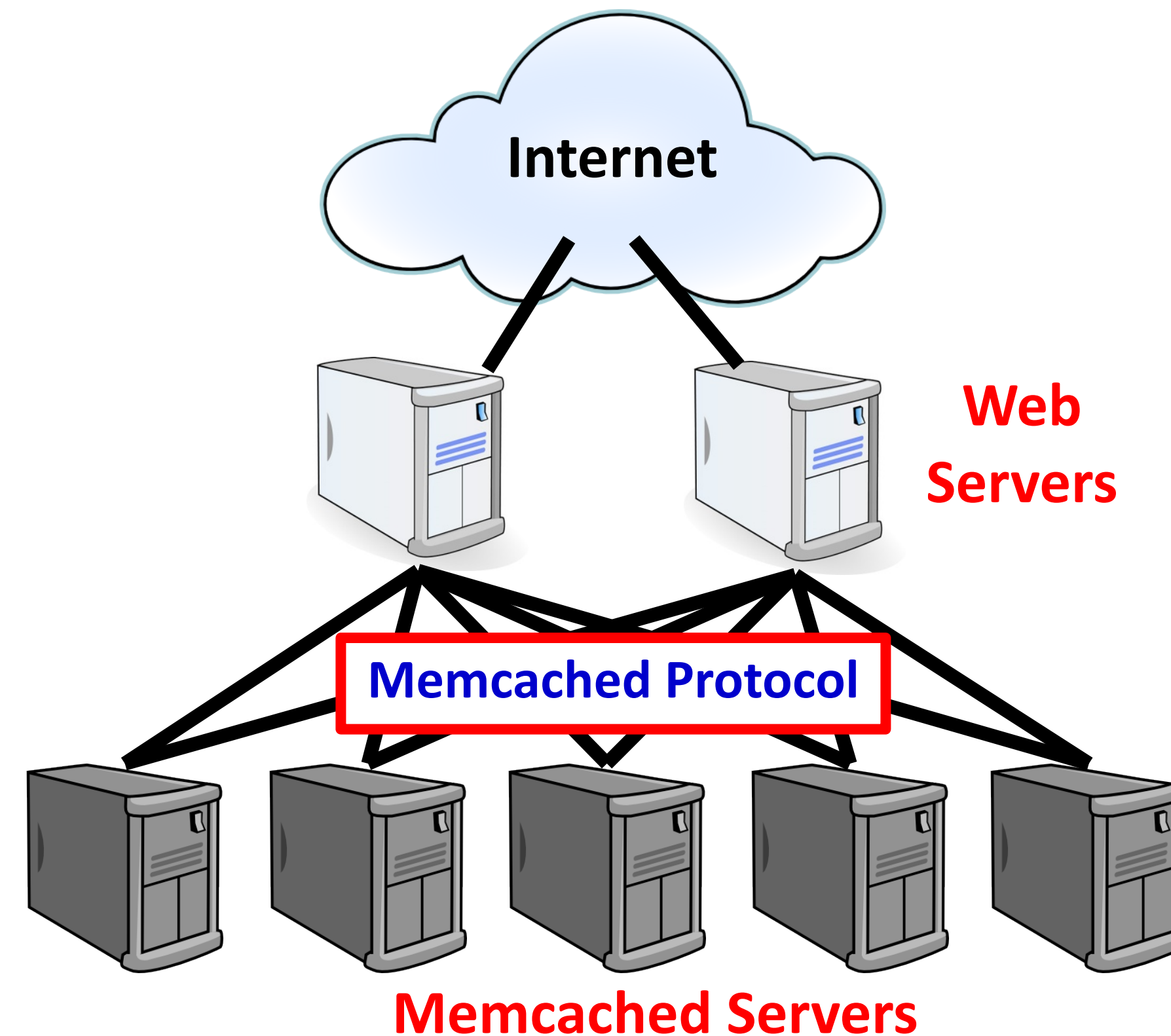
Worker Nodes

Partition/Aggregation application structure is everywhere!

Generality of Partition/Aggregation

- The foundation for many large-scale web applications.
 - Social network, ad selection/recommendation, AI inference,...

- Example: Facebook
 - Aggregators: Web servers
 - Workers: Memcached servers



Workload Characteristics

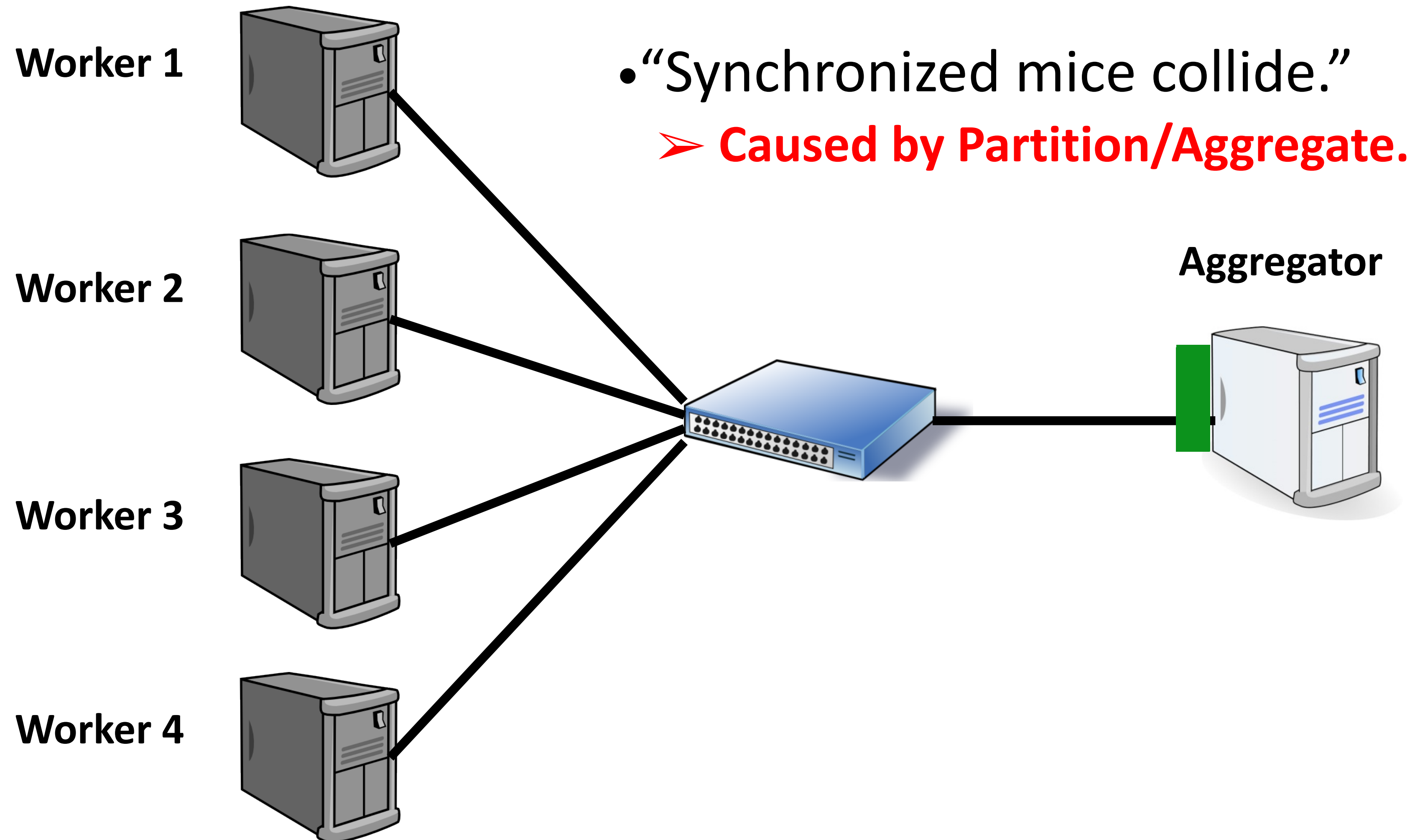
- #1: Split/Merge
 - Query-like execution model
- #2: Short message [50K-1MB]
 - Coordination, control state,...
- #3: Large flows [1MB-50MB]
 - Data retrieval and update

Workload Characteristics

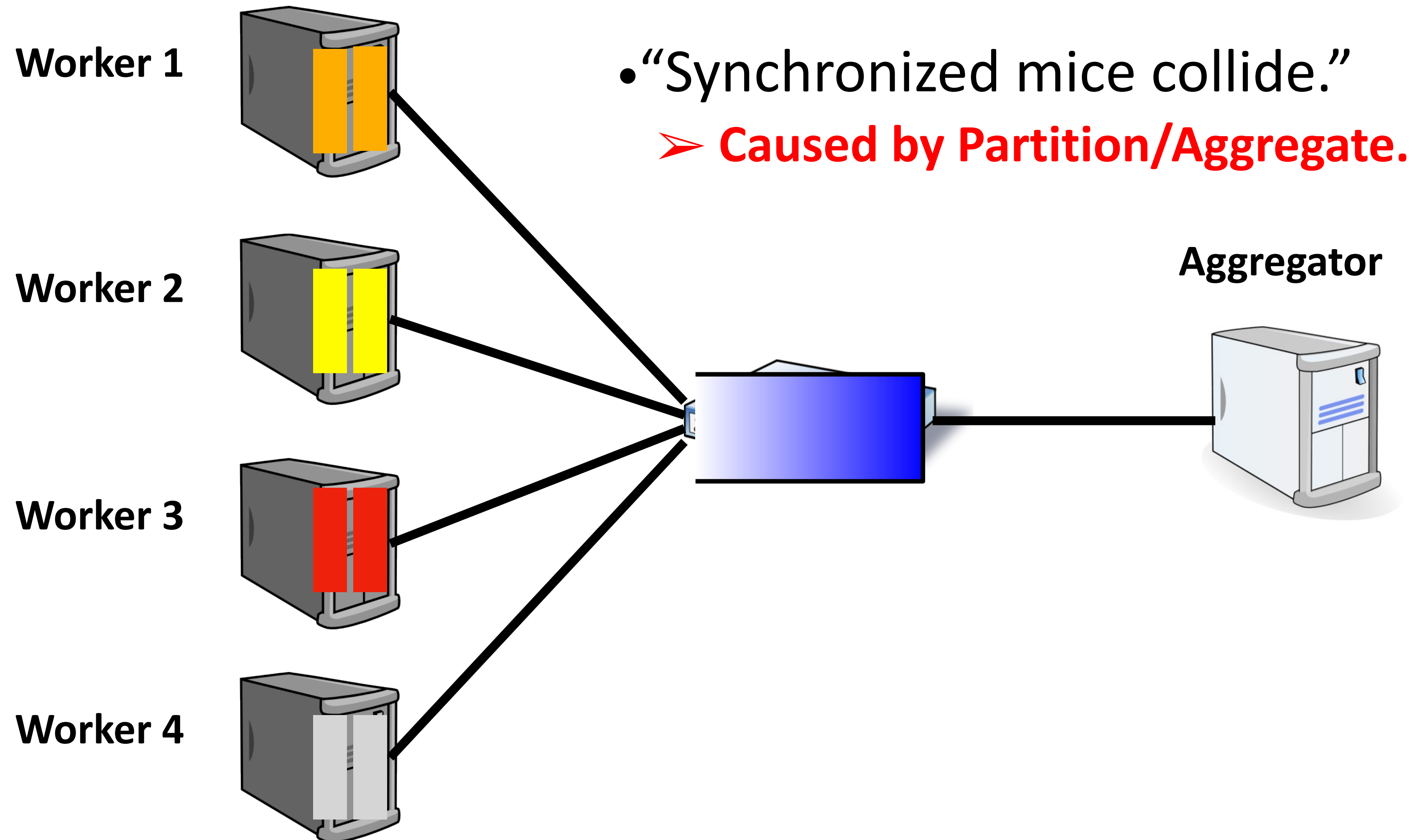
- #1: Split/Merge
 - Query-like execution model→ Delay-sensitive
- #2: Short message [50K-1MB]
 - Coordination, control state,...→ Delay-sensitive
- #3: Large flows [1MB-50MB]
 - Data retrieval and update→ Throughput-sensitive

Given these, what are the performance abnormalities?

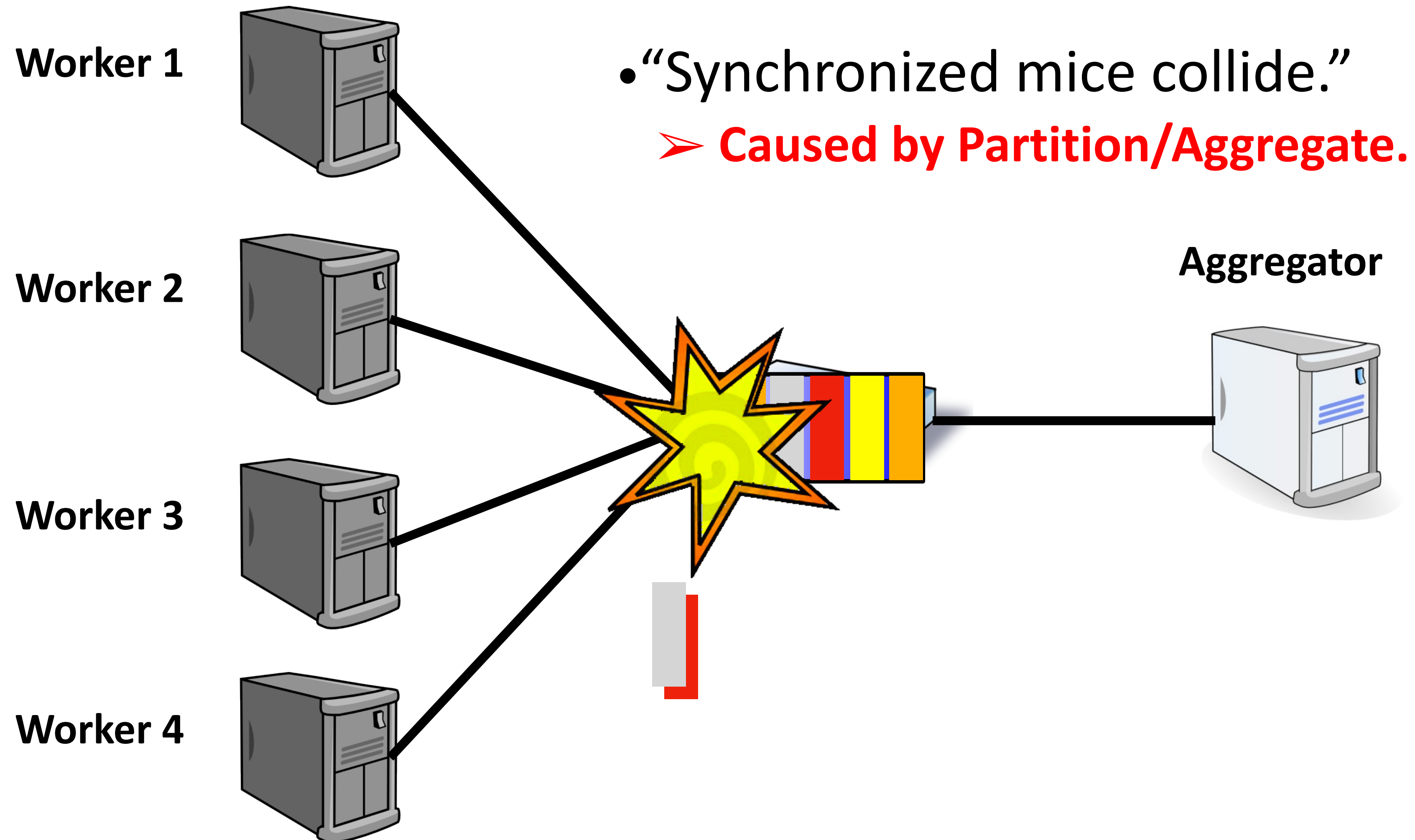
Issue #1: Incast



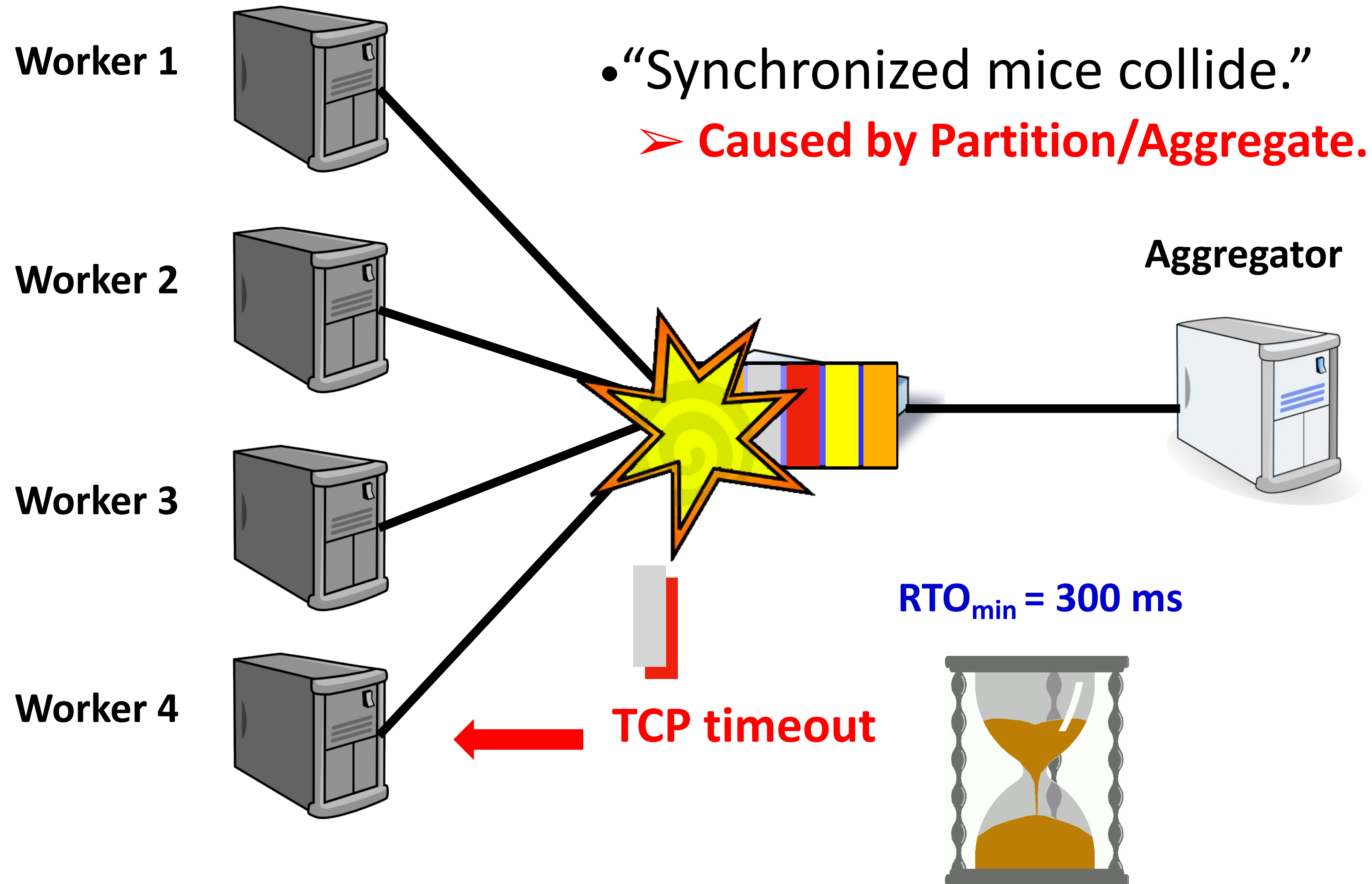
Issue #1: Incast



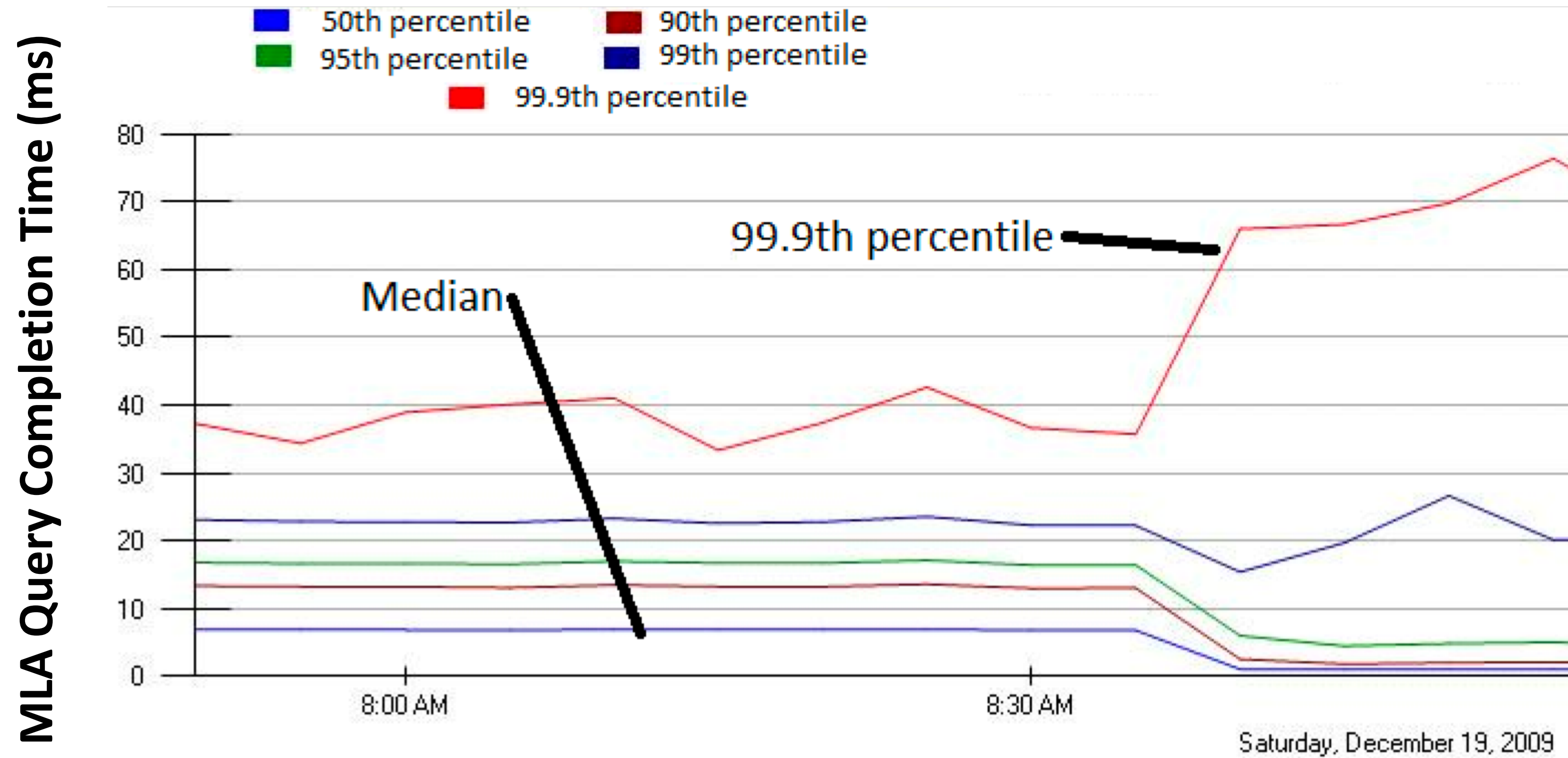
Issue #1: Incast



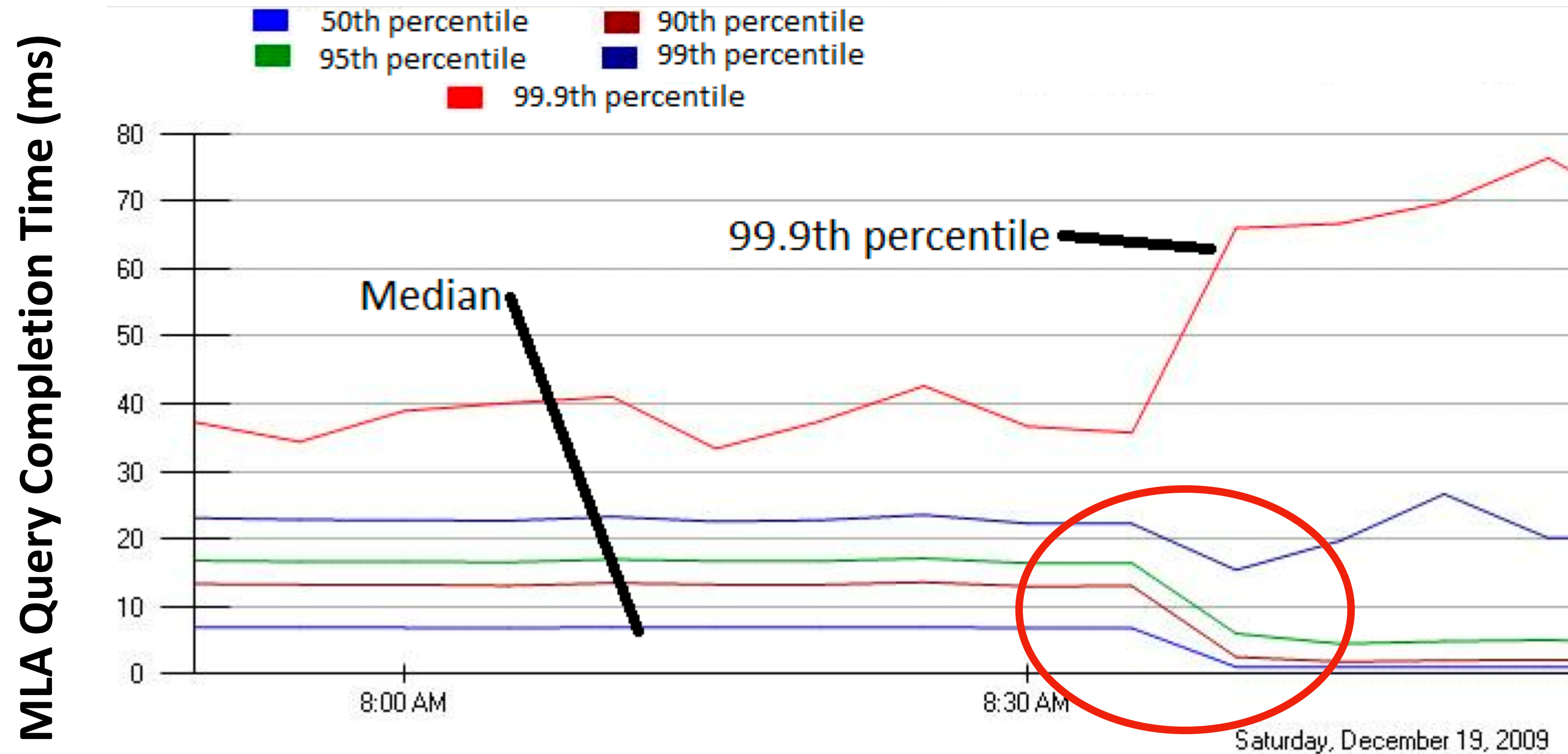
Issue #1: Incast



Performance under Incast

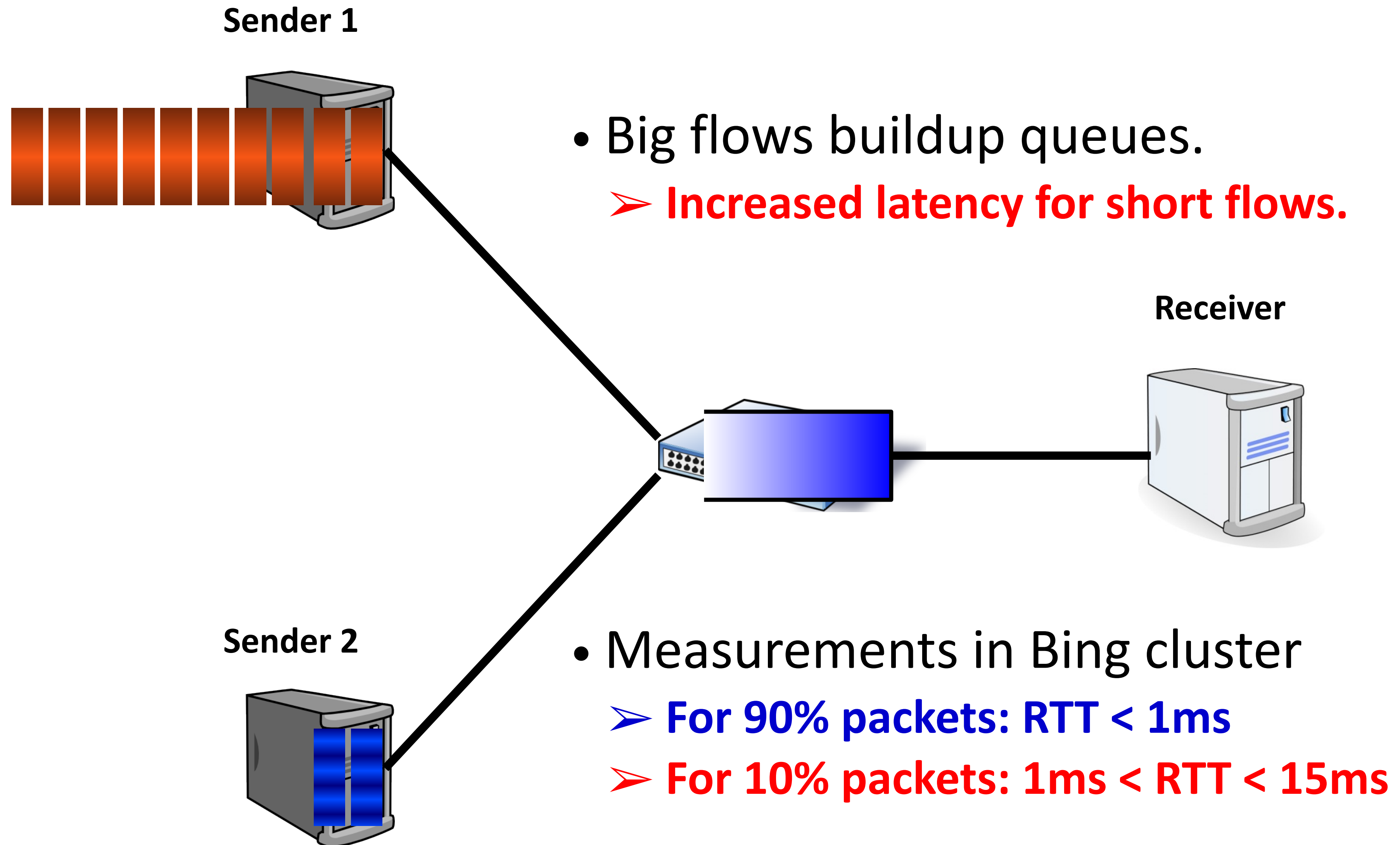


Performance under Incast



- Requests are jittered over 10ms windows
- Jittering switched off around 8:30am

Issue #2: Queue Buildup

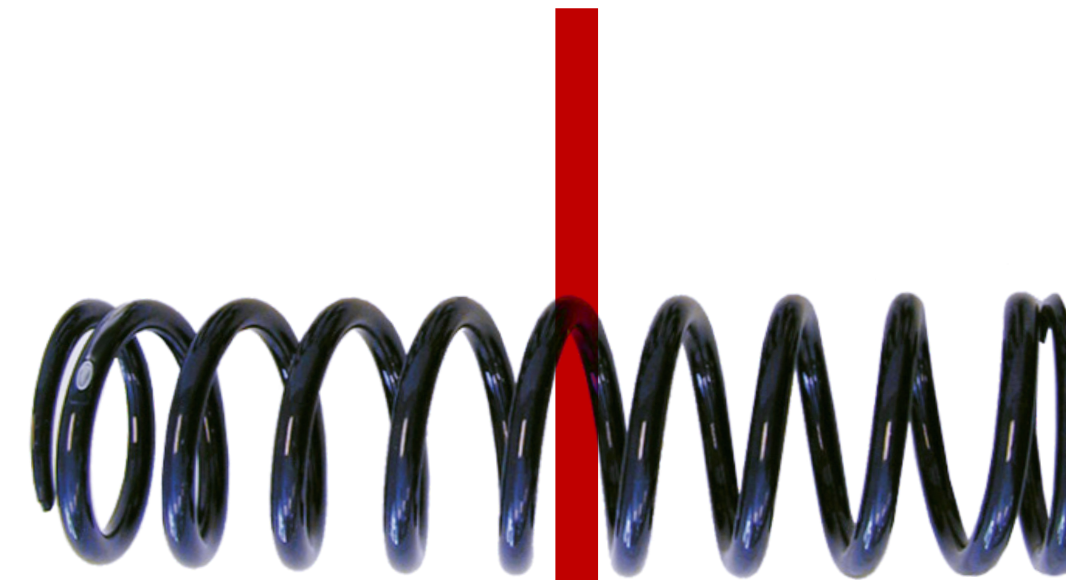


Design Goals of the “Enhanced” TCP

- #1: High burst performance
 - Tolerate the incast due to partition/aggregation
- #2: Low average/tail latency
 - Short flows
- #3: High throughput
 - Continuous data updates and large file transfers

Tension Between Requirements

**High Throughput
High Burst Tolerance**



Low Latency

Deep Buffers:

- Queuing Delays & Increase Latency

**Reduced RTO_{min}
(SIGCOMM '09)**

- Doesn't Help Latency

Shallow Buffers:

- Bad for Bursts & Throughput

AQM – RED:

- Avg Queue Not Fast Enough for Incast

Objective:

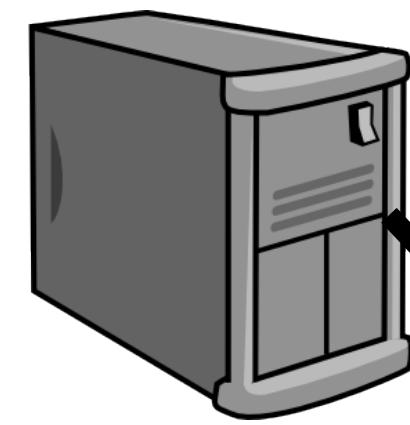
Low Queue Occupancy & High Throughput

How does DCTCP solve the problem?

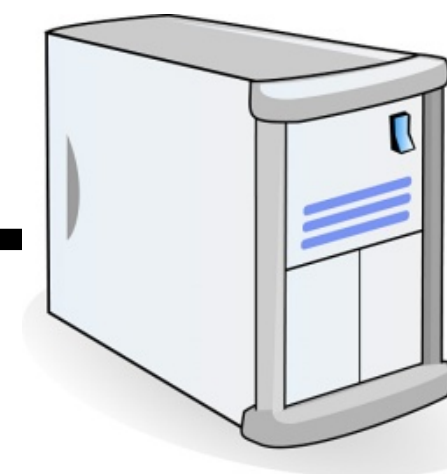
TCP/ECN Control Loop

Sender 1

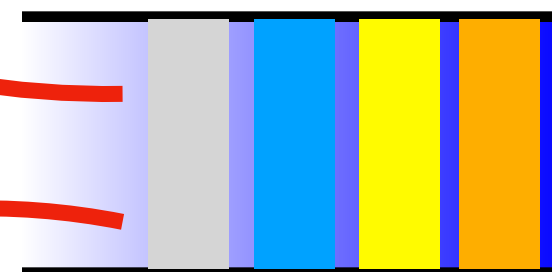
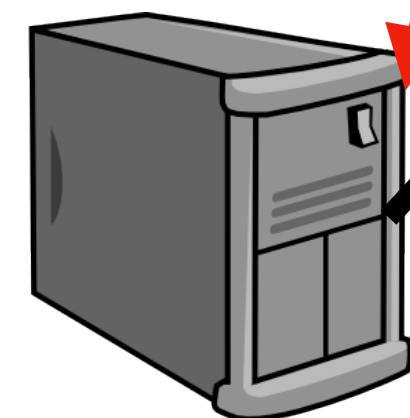
ECN = Explicit Congestion Notification



Receiver

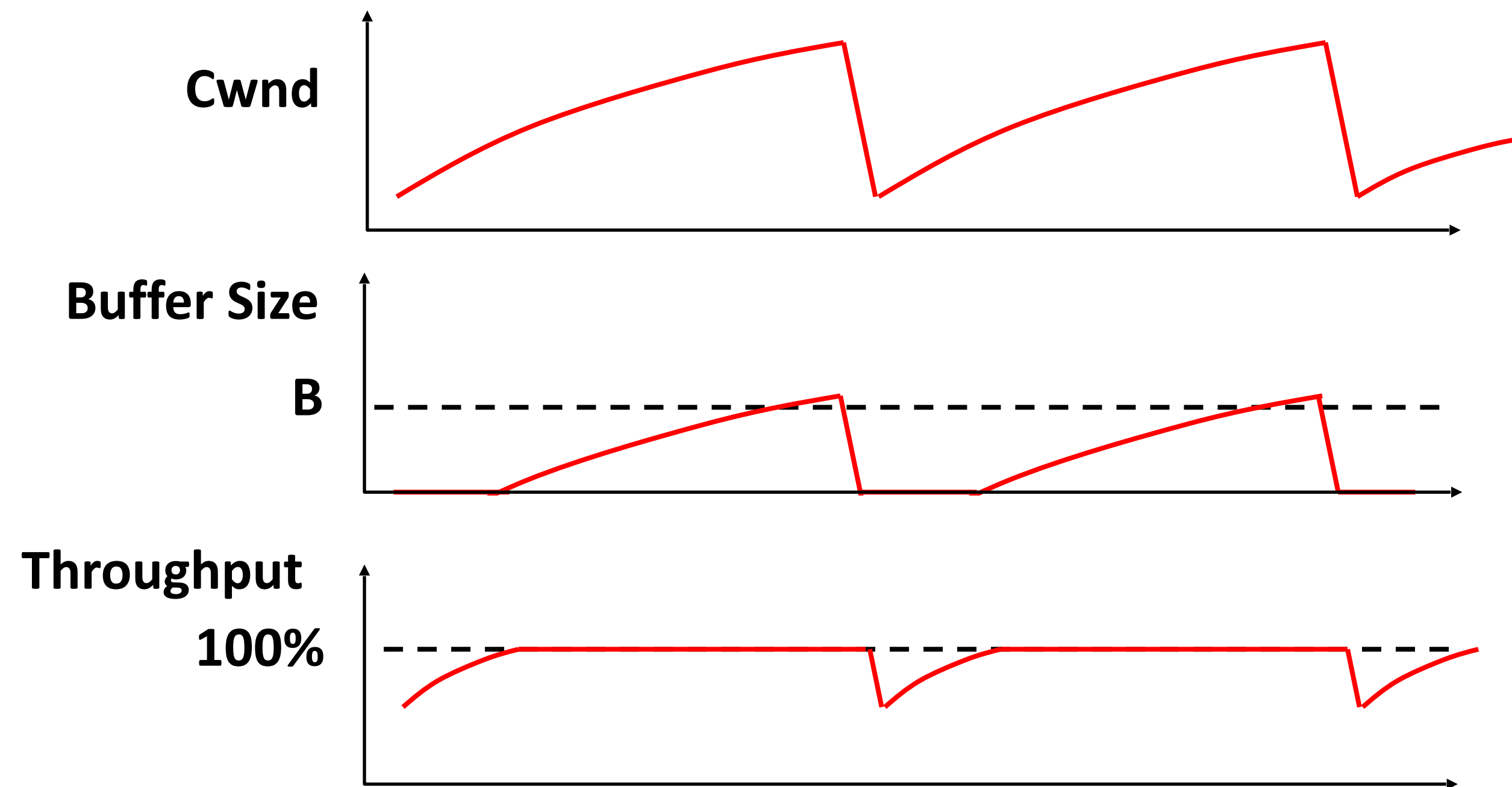


Sender 2



Buffer Sizing: Small Queue

- Bandwidth-delay product rule of thumb:
 - A single flow needs $C \times RTT$ buffers for 100% throughput



Buffer Sizing: Small Queue

- Bandwidth-delay product rule of thumb:
 - A single flow needs $C \times RTT$ buffers for 100% throughput
- Appenzeller rule of thumb (SIGCOMM'04):
 - Large # of flows: $C \times \frac{RTT}{\sqrt{N}}$ is enough

Buffer Sizing: Small Queue

- Bandwidth-delay product rule of thumb:
 - A single flow needs $C \times RTT$ buffers for 100% throughput
- Appenzeller rule of thumb (SIGCOMM'04):
 - Large # of flows: $C \times \frac{RTT}{\sqrt{N}}$ is enough
- Can't rely on stat-mux benefit in the data center
 - Measurements show typically 1-2 big flows at each server, at most 4

Buffer Sizing: Small Queue

- Bandwidth-delay product rule of thumb:
 - A single flow needs $C \times RTT$ buffers for 100% throughput
- Appenzeller rule of thumb (SIGCOMM'04):
 - Large # of flows: $C \times \frac{RTT}{\sqrt{N}}$ is enough
- Can't rely on stat-mux benefit in the data center
 - Measurements show typically 1-2 big flows at each server, at most 4

Real Rule of Thumb:
Low Variance in Sending Rate —> Small Buffers Suffice

Key Ideas of DCTCP

- #1: React proportional the extent of congestion, not its presence
 - Reduce variances in sending rates, lowering queue requirement

ECN Marks	TCP	DCTCP
1 0 1 1 1 1 0 1 1 1	Cut window by 50%	Cut window by 40%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%

Key Ideas of DCTCP

- #1: React proportional the extent of congestion, not its presence
 - Reduce variances in sending rates, lowering queue requirement

ECN Marks	TCP	DCTCP
1 0 1 1 1 1 0 1 1 1	Cut window by 50%	Cut window by 40%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%

- #2: Mark based on instantaneous queue length
 - Fast feedback to better deal with bursts

DCTCP Algorithm

Sender

Switch

Receiver

DCTCP Algorithm

Sender

Switch

Receiver

Sender side:

- Maintain the running average of the ***fraction*** of packets marked (α).

$$F = \frac{\# \text{ of marked ACKs}}{\text{Total \# of ACKs}}$$

$$\alpha \leftarrow (1 - g)\alpha + gF$$

$$Cwnd \leftarrow (1 - \frac{\alpha}{2})Cwnd$$

DCTCP Algorithm

Sender

Switch

Receiver

Sender side:

- Maintain the running average of the ***fraction*** of packets marked (α).

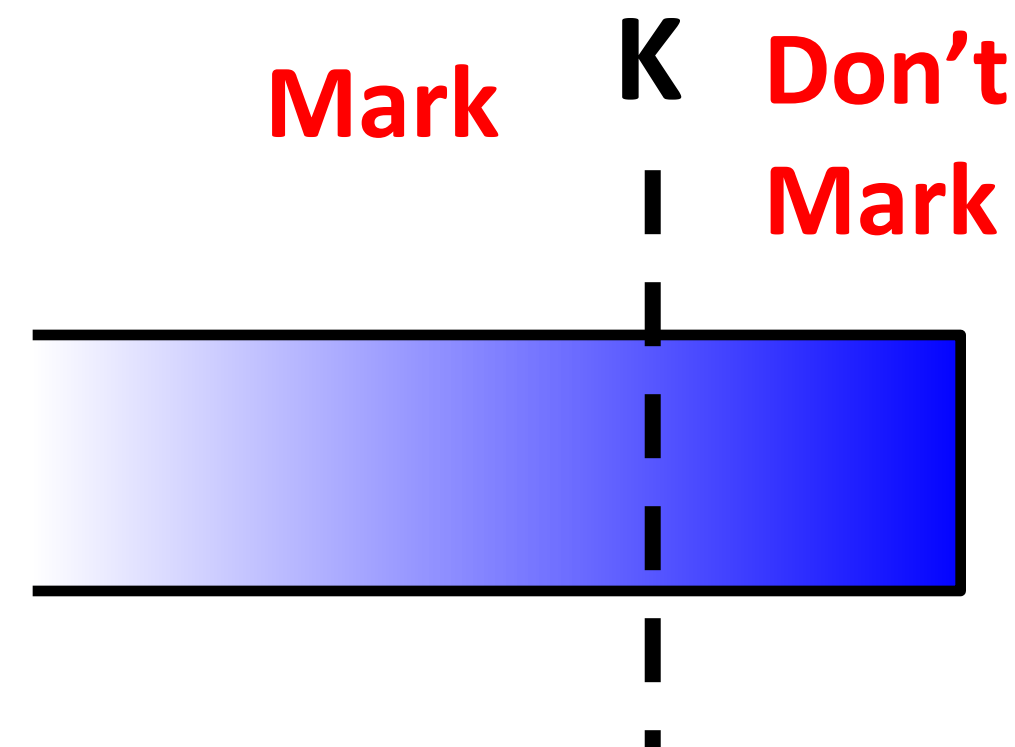
$$F = \frac{\# \text{ of marked ACKs}}{\text{Total \# of ACKs}}$$

$$\alpha \leftarrow (1 - g)\alpha + gF$$

$$Cwnd \leftarrow (1 - \frac{\alpha}{2})Cwnd$$

Switch side:

- Mark packets when **Queue Length > K**.



DCTCP Algorithm

Sender

Switch

Receiver

Sender side:

- Maintain the running average of the ***fraction*** of packets marked (α).

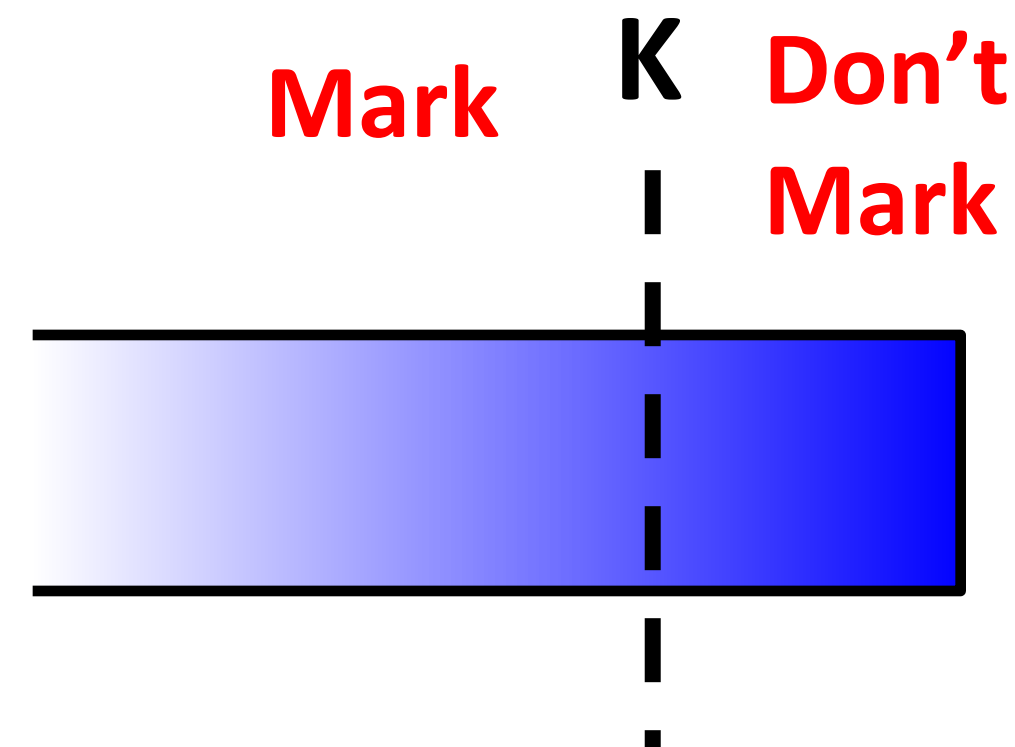
$$F = \frac{\# \text{ of marked ACKs}}{\text{Total \# of ACKs}}$$

$$\alpha \leftarrow (1 - g)\alpha + gF$$

$$Cwnd \leftarrow (1 - \frac{\alpha}{2})Cwnd$$

Switch side:

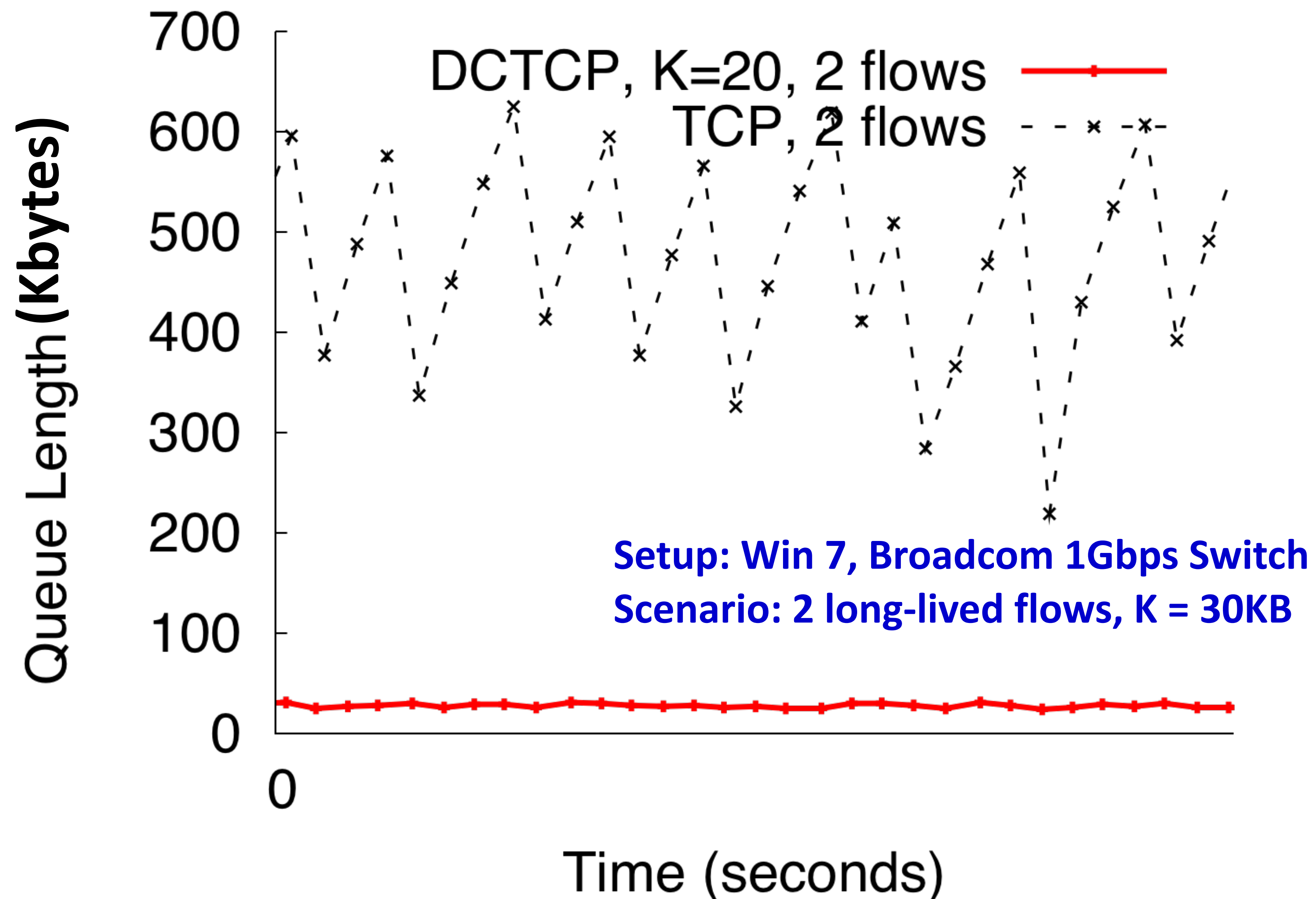
- Mark packets when **Queue Length > K**.



Receiver side:

- Piggyback

DCTCP Effect



Why does DCTCP work?

Efficiency of DCTCP

- #1: High burst performance

Efficiency of DCTCP

- #1: High burst performance

Aggressive marking —> Sources react before packets are dropped

Efficiency of DCTCP

- #1: High burst performance

Aggressive marking —> Sources react before packets are dropped

- #2: Low average/tail latency

Efficiency of DCTCP

- #1: High burst performance

Aggressive marking —> Sources react before packets are dropped

- #2: Low average/tail latency

Small buffer occupancy —> Low queueing delay

Efficiency of DCTCP

- #1: High burst performance

Aggressive marking —> Sources react before packets are dropped

- #2: Low average/tail latency

Small buffer occupancy —> Low queueing delay

- #3: High throughput

Efficiency of DCTCP

- #1: High burst performance

Aggressive marking —> Sources react before packets are dropped

- #2: Low average/tail latency

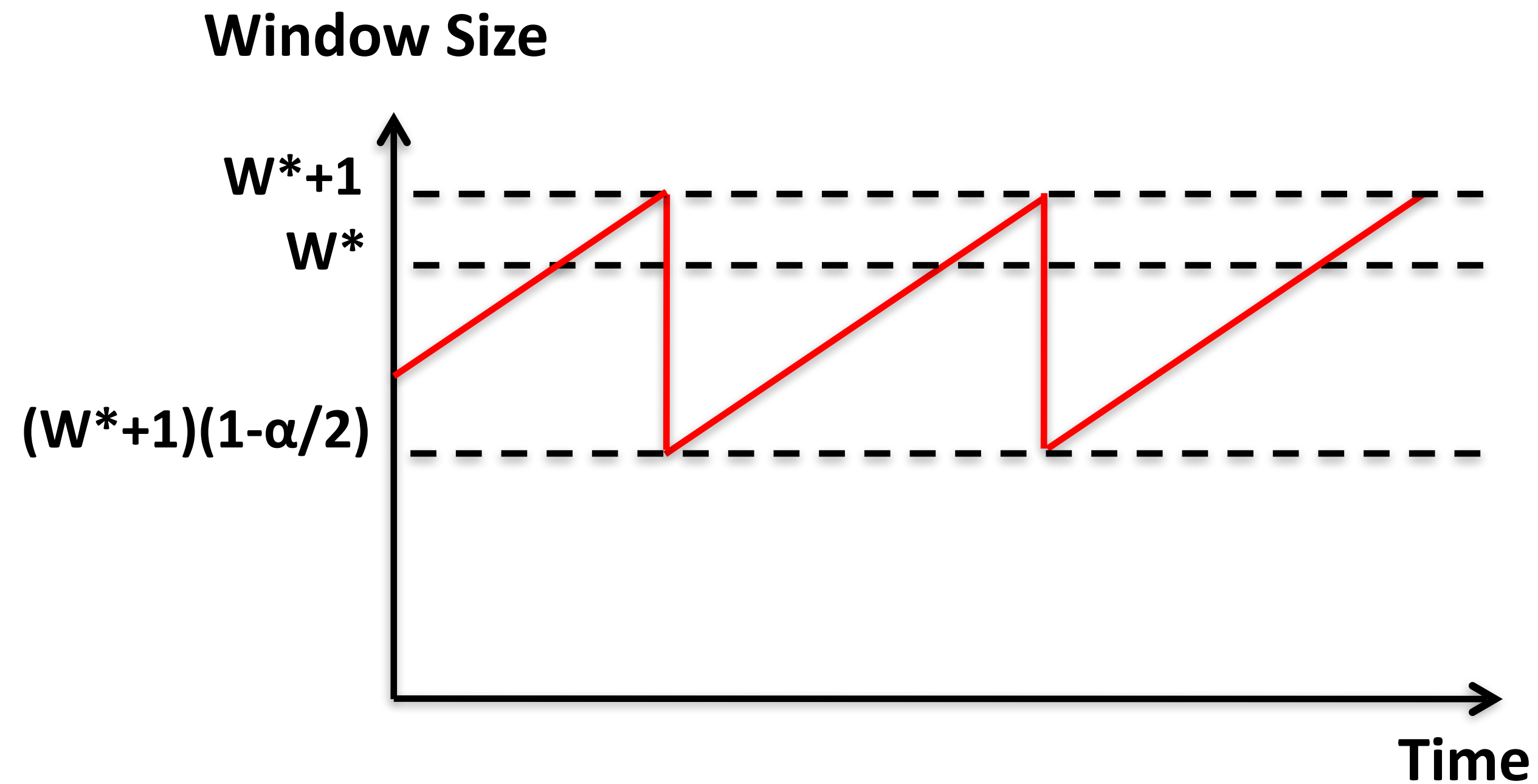
Small buffer occupancy —> Low queueing delay

- #3: High throughput

ECN averaging —> smooth rate adjustment, low variance

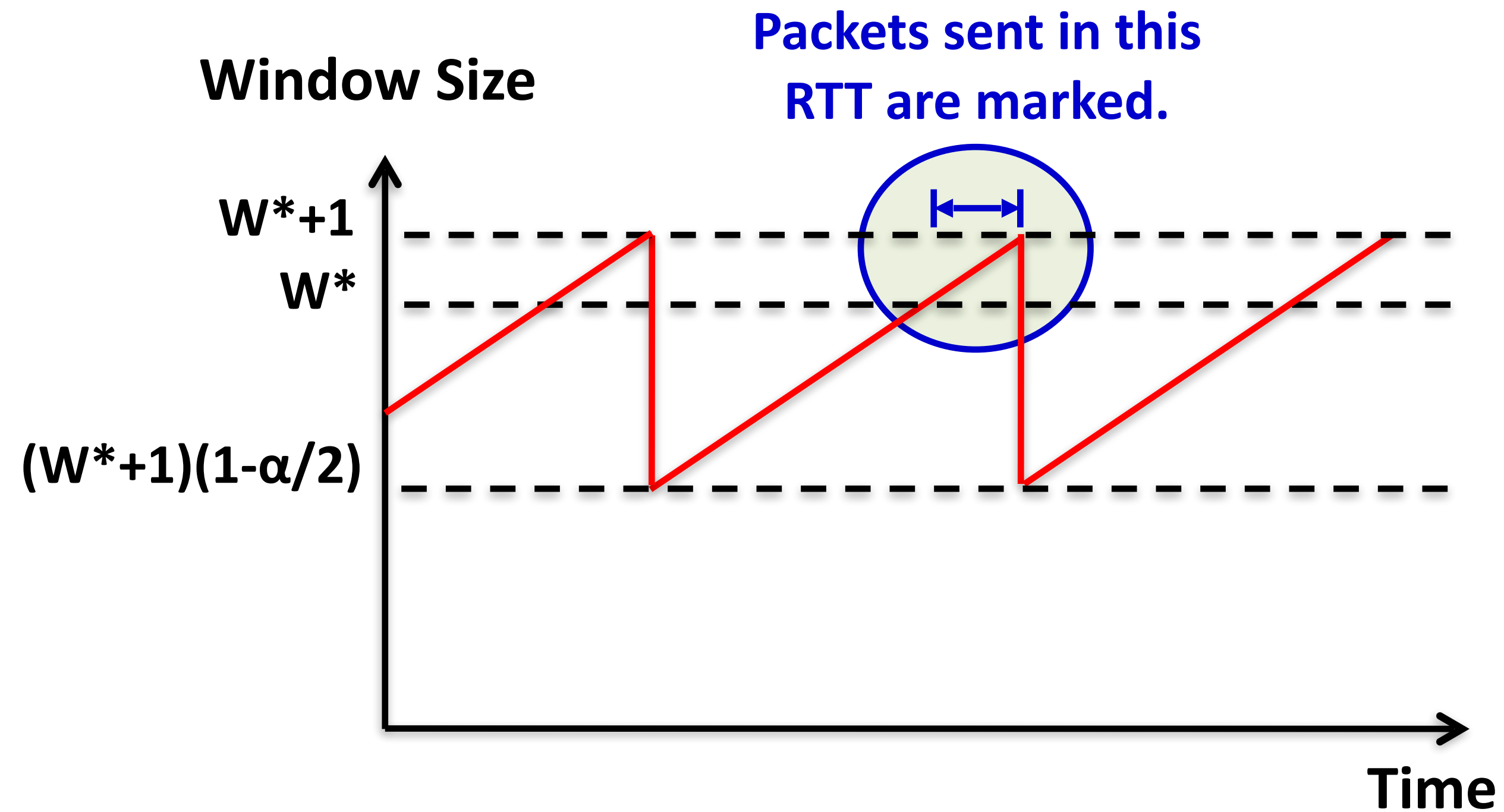
DCTCP Analysis

- How long can DCTCP maintain queues without throughput drop?



DCTCP Analysis

- How long can DCTCP maintain queues without throughput drop?



DCTCP Analysis

- How long can DCTCP maintain queues without throughput drop?
- How do we set the DCTCP parameters?

DCTCP Analysis

- How long can DCTCP maintain queues without throughput drop?
- How do we set the DCTCP parameters?

Need to quantify queue size oscillations (stability)

DCTCP Analysis

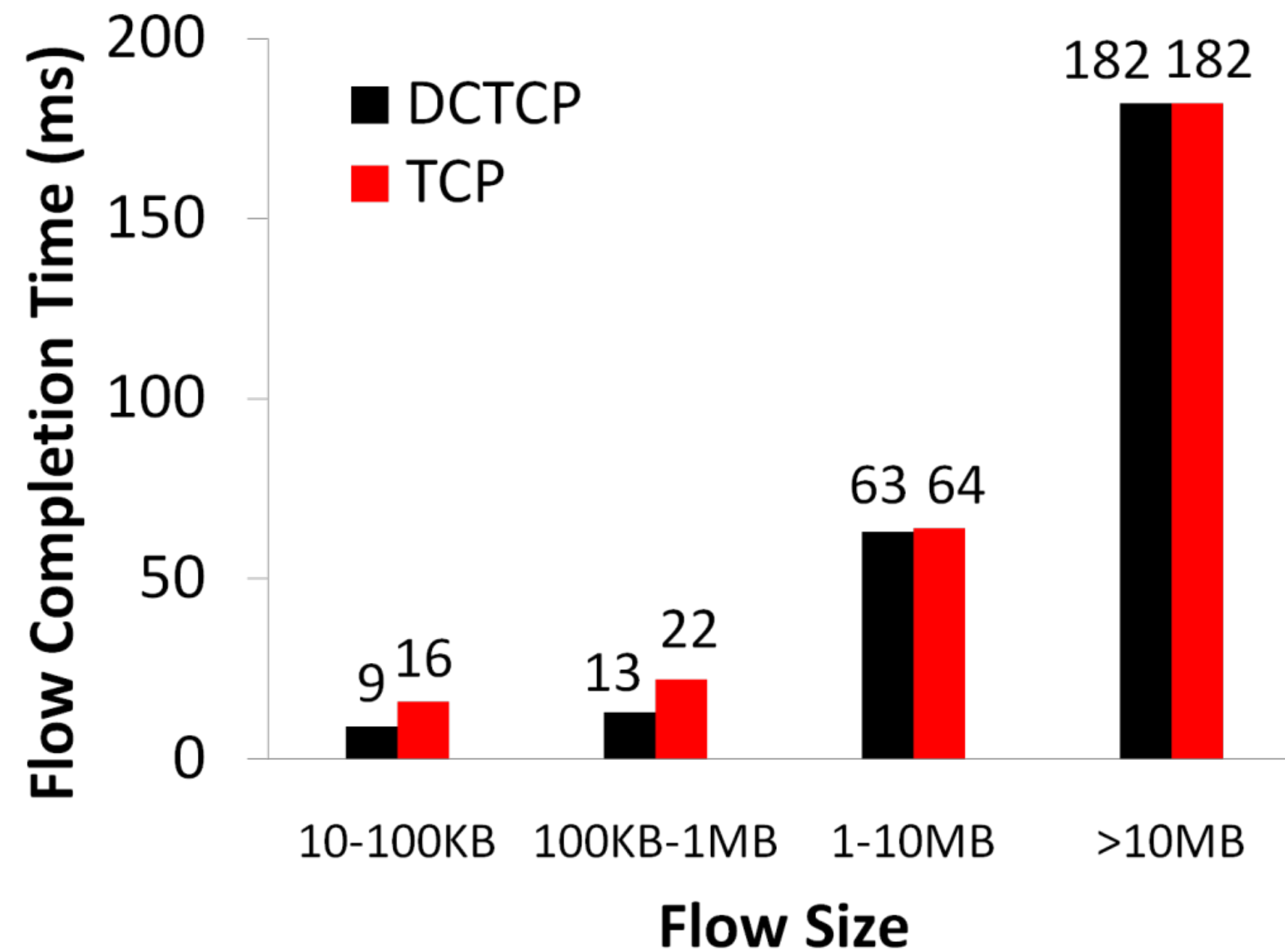
- How long can DCTCP maintain queues without throughput drop?
- How do we set the DCTCP parameters?

$$K > \frac{1}{7} C \times RTT$$

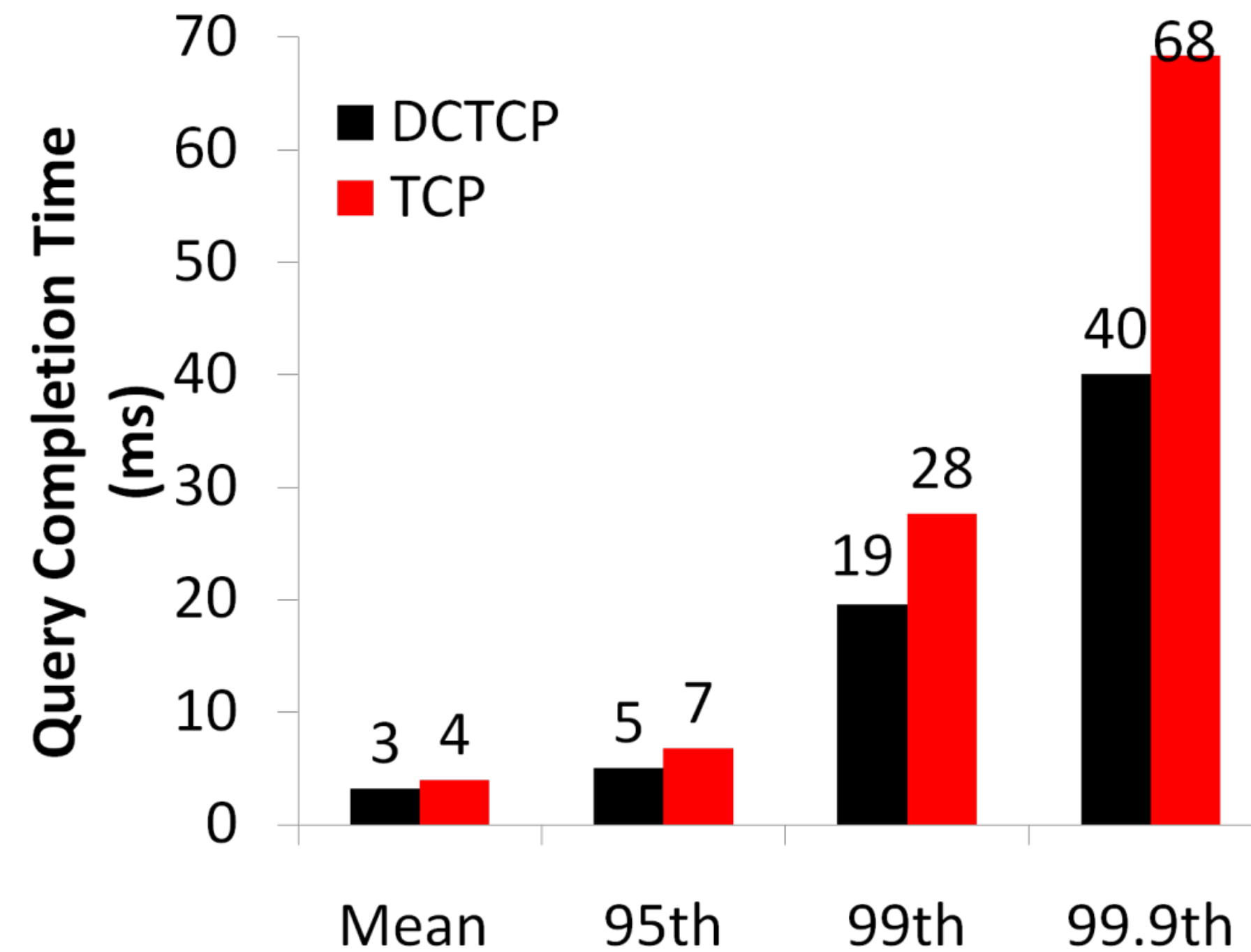
85% Less Buffer than TCP

DCTCP Performance

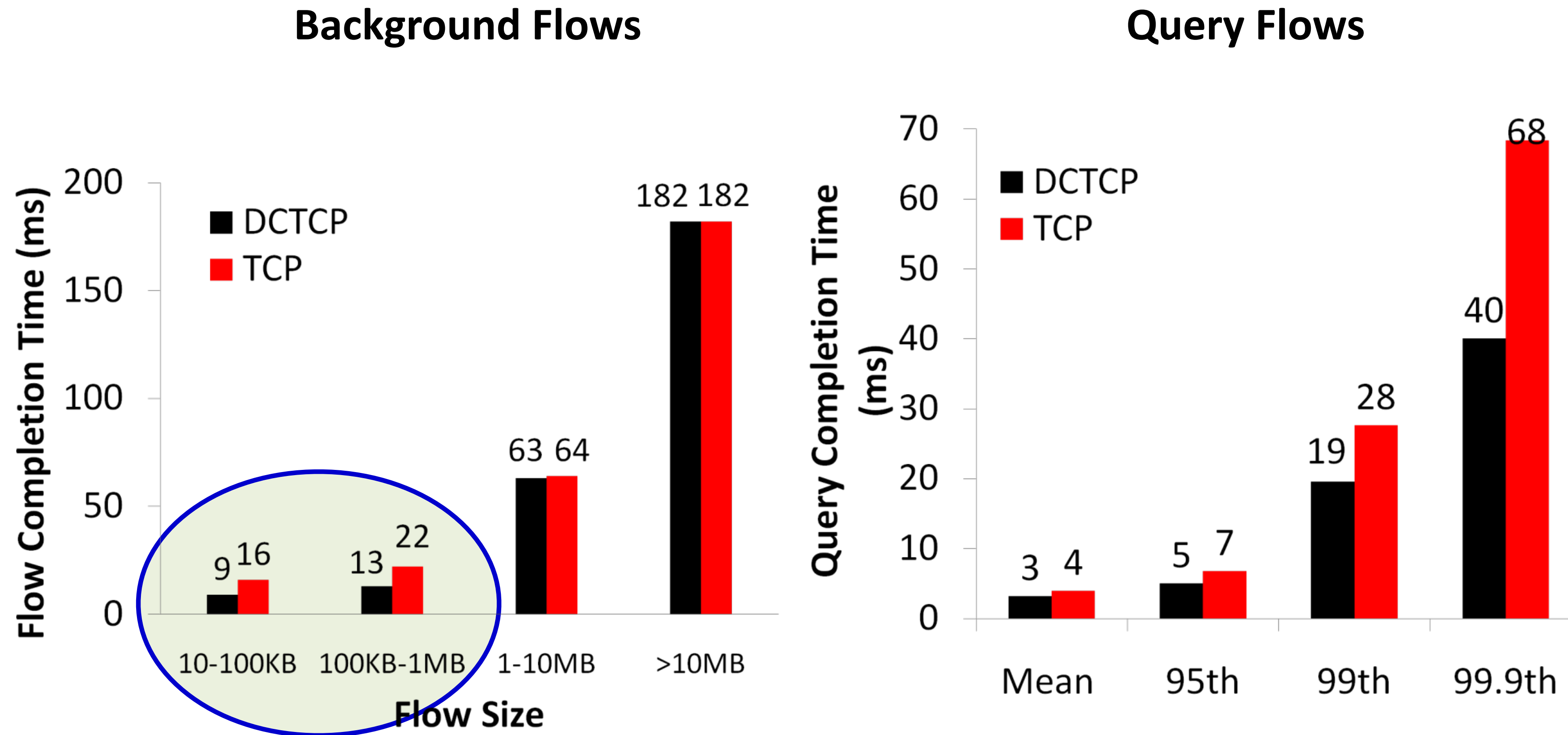
Background Flows



Query Flows

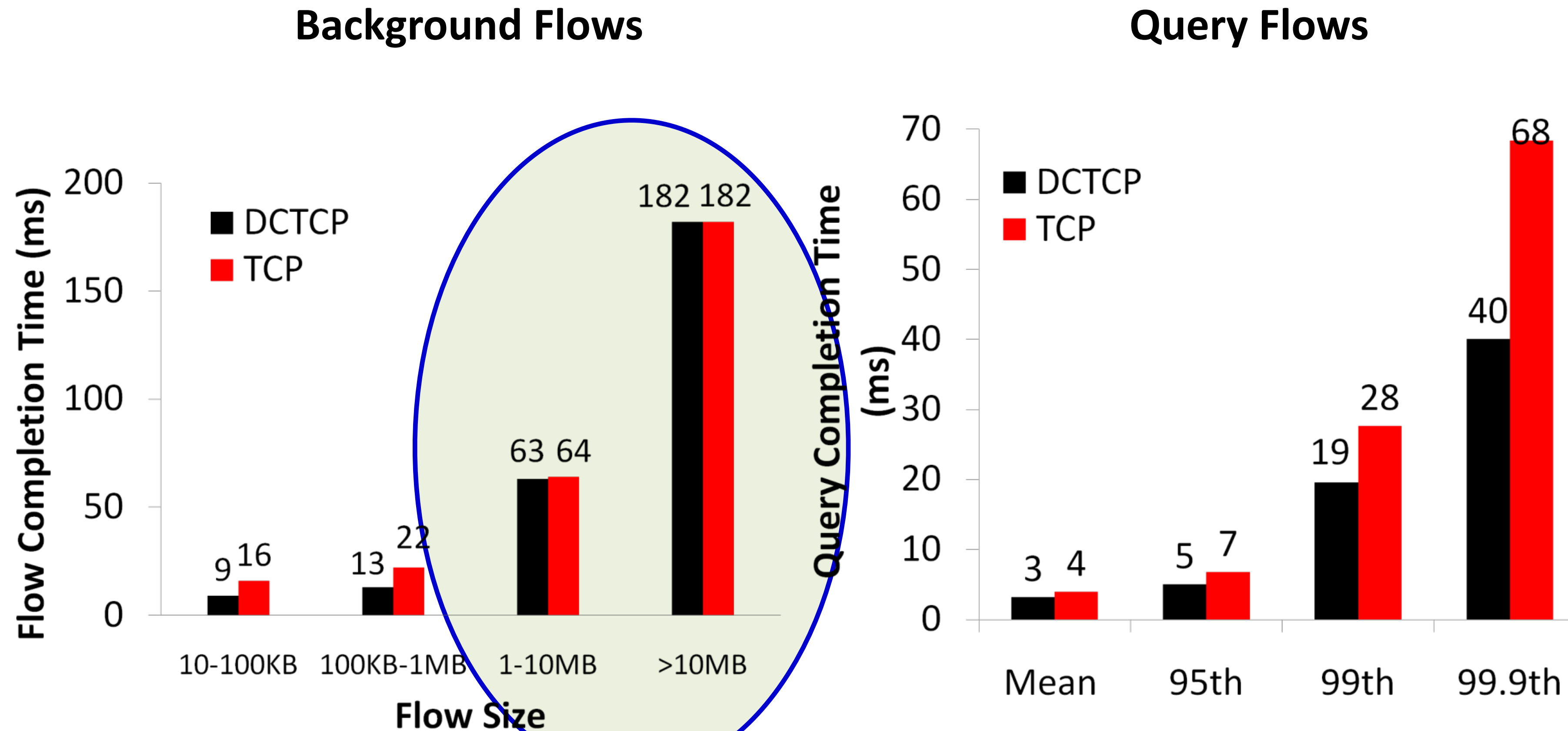


DCTCP Performance



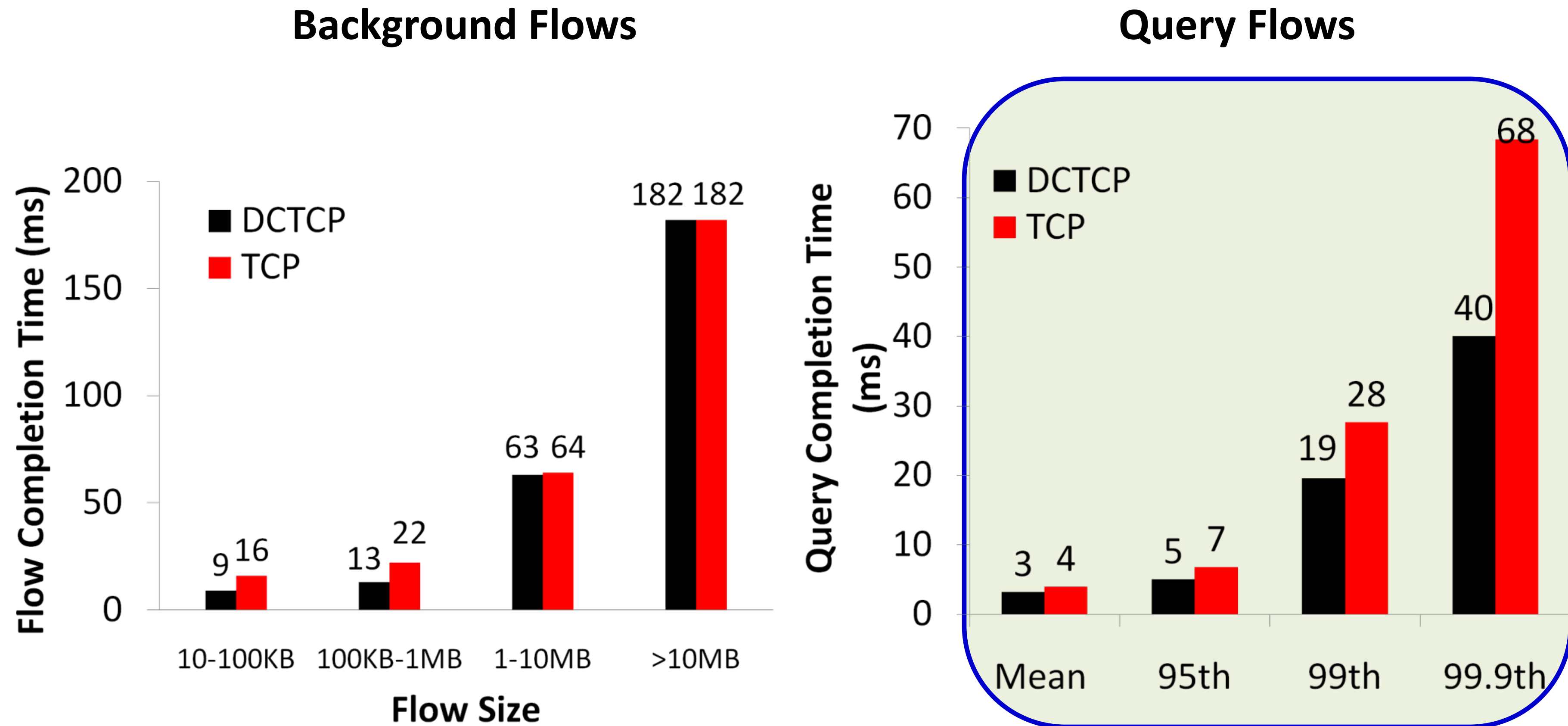
✓ Low latency for short flows

DCTCP Performance



- ✓ Low latency for short flows
- ✓ High throughput for long flows

DCTCP Performance



- ✓ Low latency for short flows
- ✓ High throughput for long flows
- ✓ High burst tolerance for query flows

DCTCP Deployment at Meta

- Coordinate switch ECN behavior, host TCP, and existing traffic

A large-scale deployment of DCTCP

Operational Systems Track

Abhishek Dhamija¹, Balasubramanian Madhavan¹, Hechao Li², Jie Meng¹, Shrikrishna Khare¹, Madhavi Rao¹, Lawrence Brakmo, Neil Spring¹, Prashanth Kannan¹, Srikanth Sundaresan¹, and Soudeh Ghorbani^{1,3}

¹Meta, ²Netflix, ³Johns Hopkins University

Abstract

This paper describes the process and operational experiences of deploying the Data Center TCP (DCTCP) protocol in a large-scale data center network. In contrast to legacy congestion control protocols that rely on loss as the primary signal of congestion, DCTCP signals in-network congestion (based on queue occupancy) to senders and adjusts the sending rate proportional to the level of congestion. At the time of our deployment, this protocol was well-studied and fairly established with proven efficiency gains in other networks. As expected, we also observed improved performance, and notably decreased packet losses, compared to legacy protocols in our data centers. Perhaps unexpectedly, however, we faced numerous hurdles in rolling out DCTCP; we chronicle these unexpected challenges, ranging from its unfairness (to other classes of traffic) to implementation bugs. We close by discussing some of the open research questions and challenges.

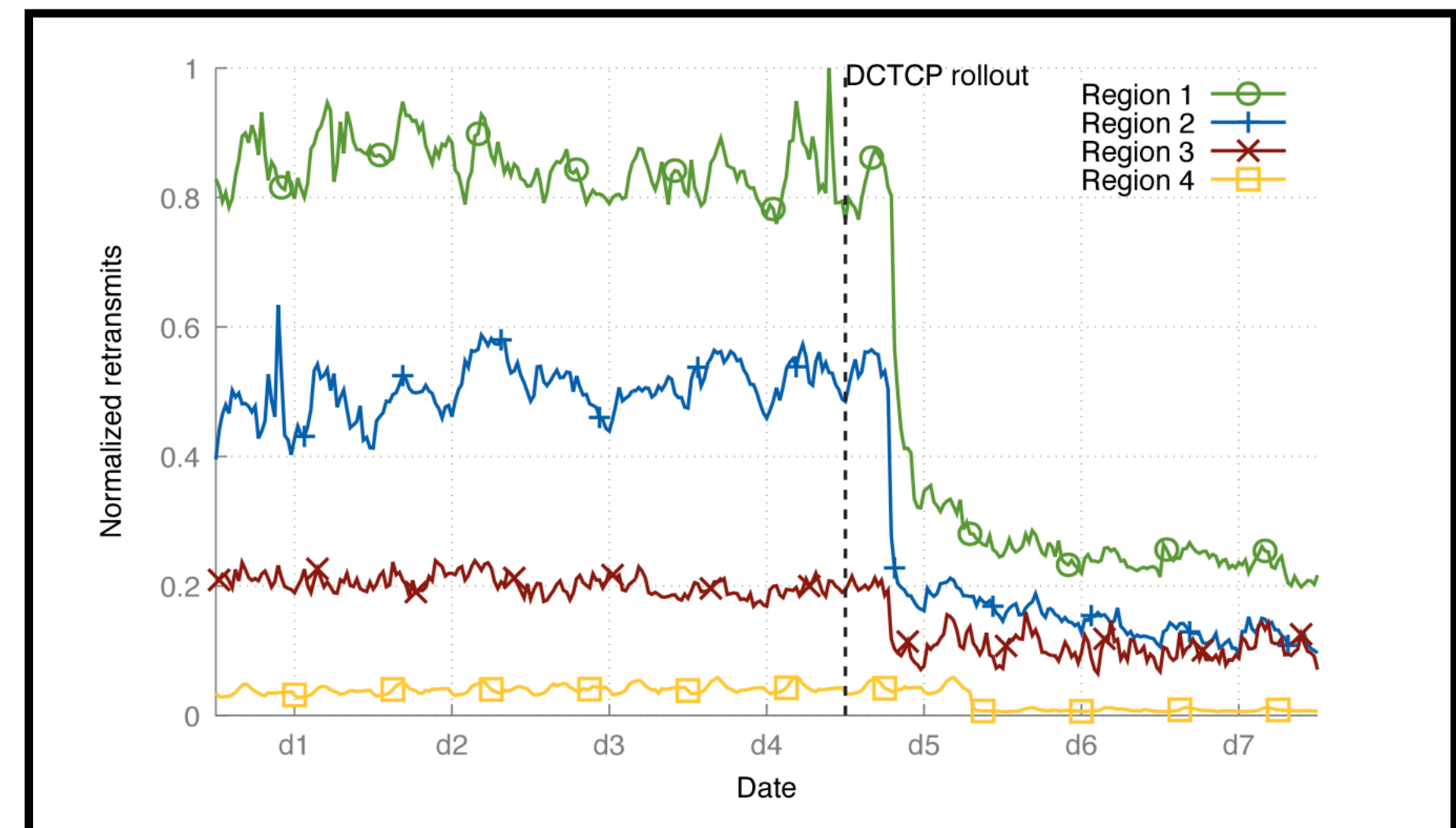
1 Introduction

Congestion control algorithms (CCAs) modulate traffic

a diverse set of congestion signals (notifications from the network, measured delay at endpoints, etc.) to detect and react to imminent congestion faster. As one of the earliest and most mature protocols in this class, Data Center Congestion Control (DCTCP) [4] uses Explicit Congestion Notifications (ECNs) from the switches to adjust the sending rate proportional to the level of congestion. ECN remedies large queue buildups and drops by providing a congestion signal before queues overflow, and DCTCP interprets the fraction of ECN-marked packets to scale its response, avoiding persistent queueing and overflows that lead to loss.

A few features of our network made DCTCP's potential worth considering. First, our top of rack switches had “shallow buffers,” providing limited space for queueing. Second, buffer sharing and contention over buffers led to variable and unpredictable queue capacities [13]. Finally, our approach to distributing jobs across datacenters created a set of racks with a mixture of large throughput-heavy flows and small latency-sensitive flows that had to compete for the limited and variable buffer space in the network. DCTCP could moderate large flows' use of switch buffers, providing more isolation between jobs sharing shallow-buffered switches.

Using DCTCP required resolving a challenge: designed



Summary

- Today
 - DCTCP
- Next
 - NDP (Sigcomm'17)