EyeQ: Practical Network Performance Isolation at the Edge

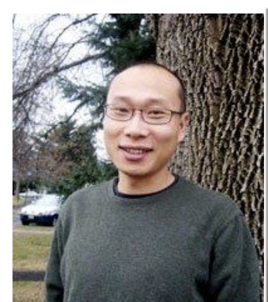
Vimalkumar Jeyakumar

Mohammad Alizadeh
Balaji Prabhakar
David Mazières

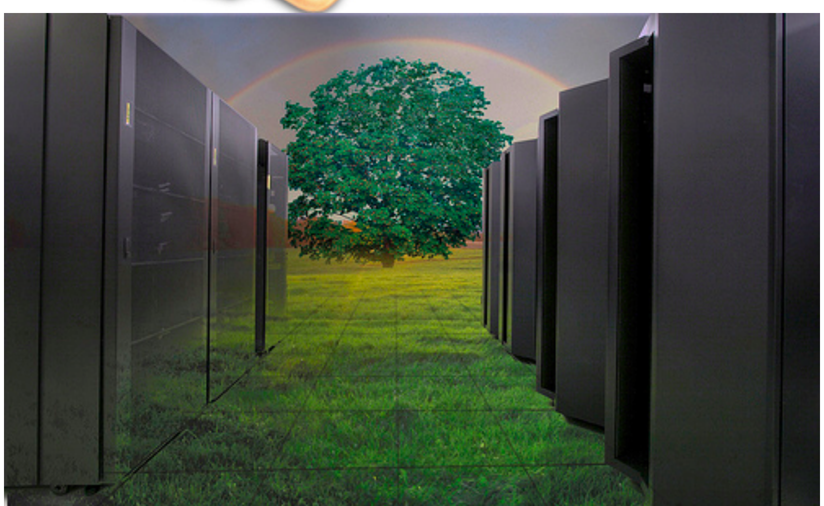
Changhoon Kim
Albert Greenberg

Windows Azure



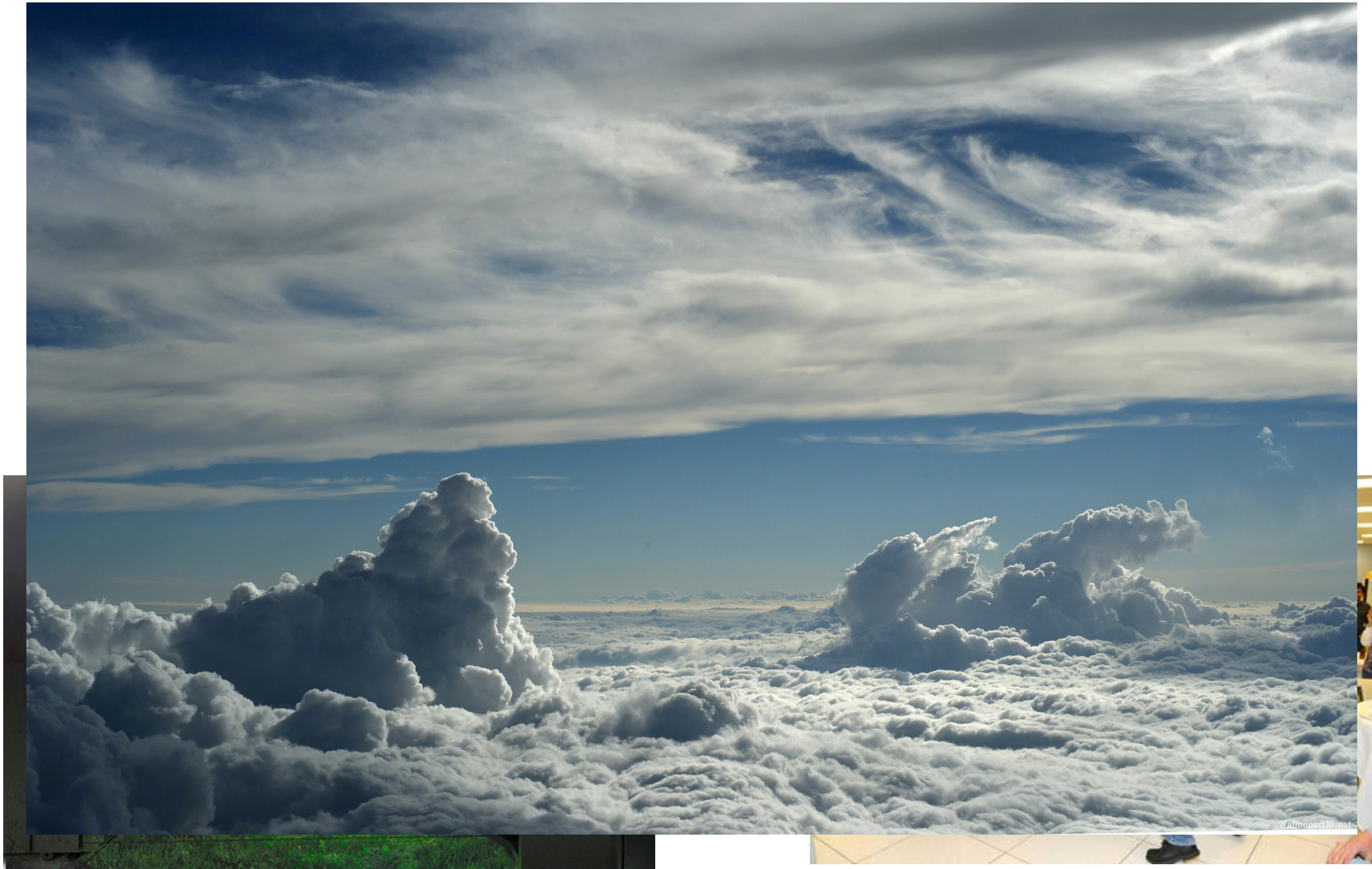




Once upon a time…



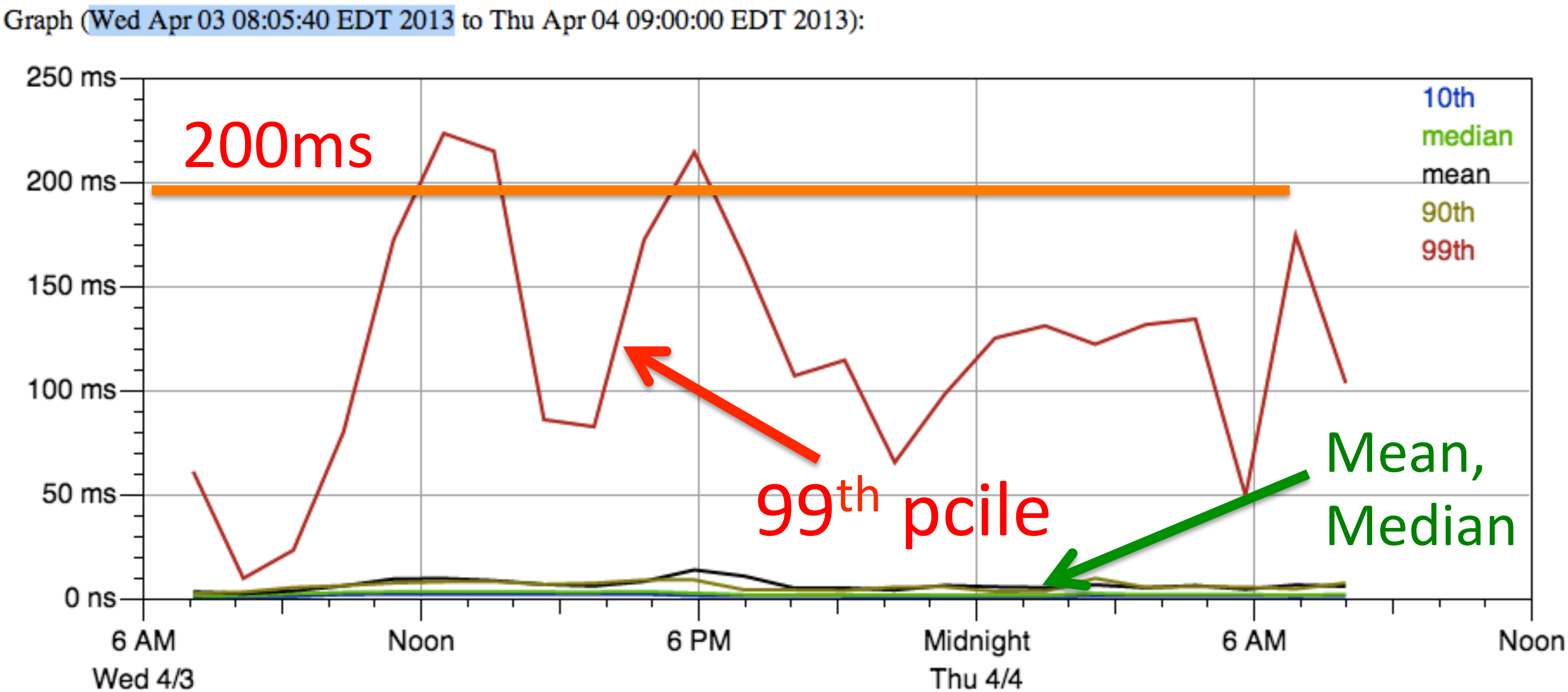
Once upon a time…

Once upon a time…



Onc… time…

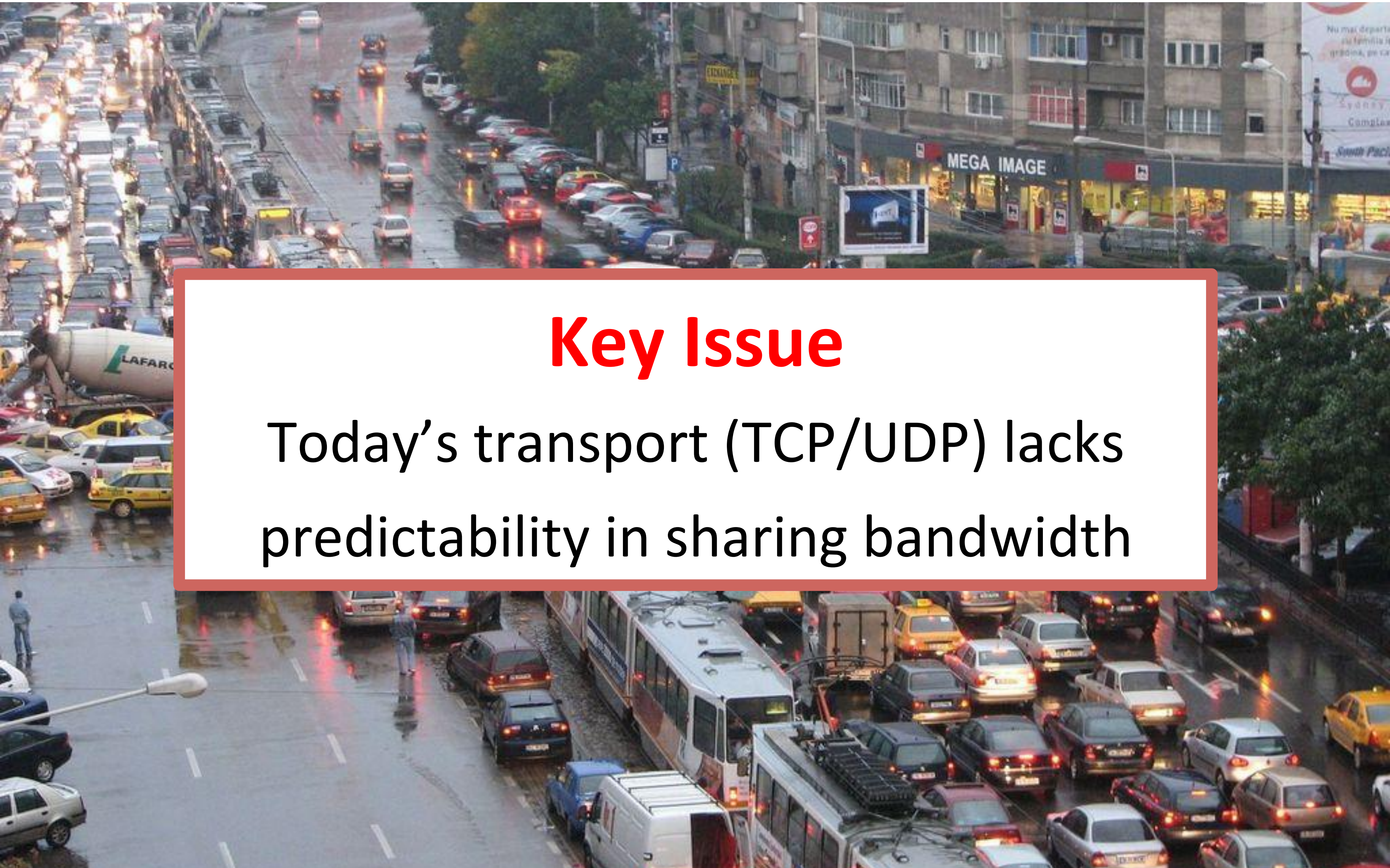
Performance Unpredictability



http://amistrongeryet.com/op_detail.jsp?op=gae_db_readCachedHandles_1&hoursAgo=24

Congestion Kills Predictability

Congestion Kills Predictability



Key Issue

Today's transport (TCP/UDP) lacks predictability in sharing bandwidth

Status Quo is Insufficient

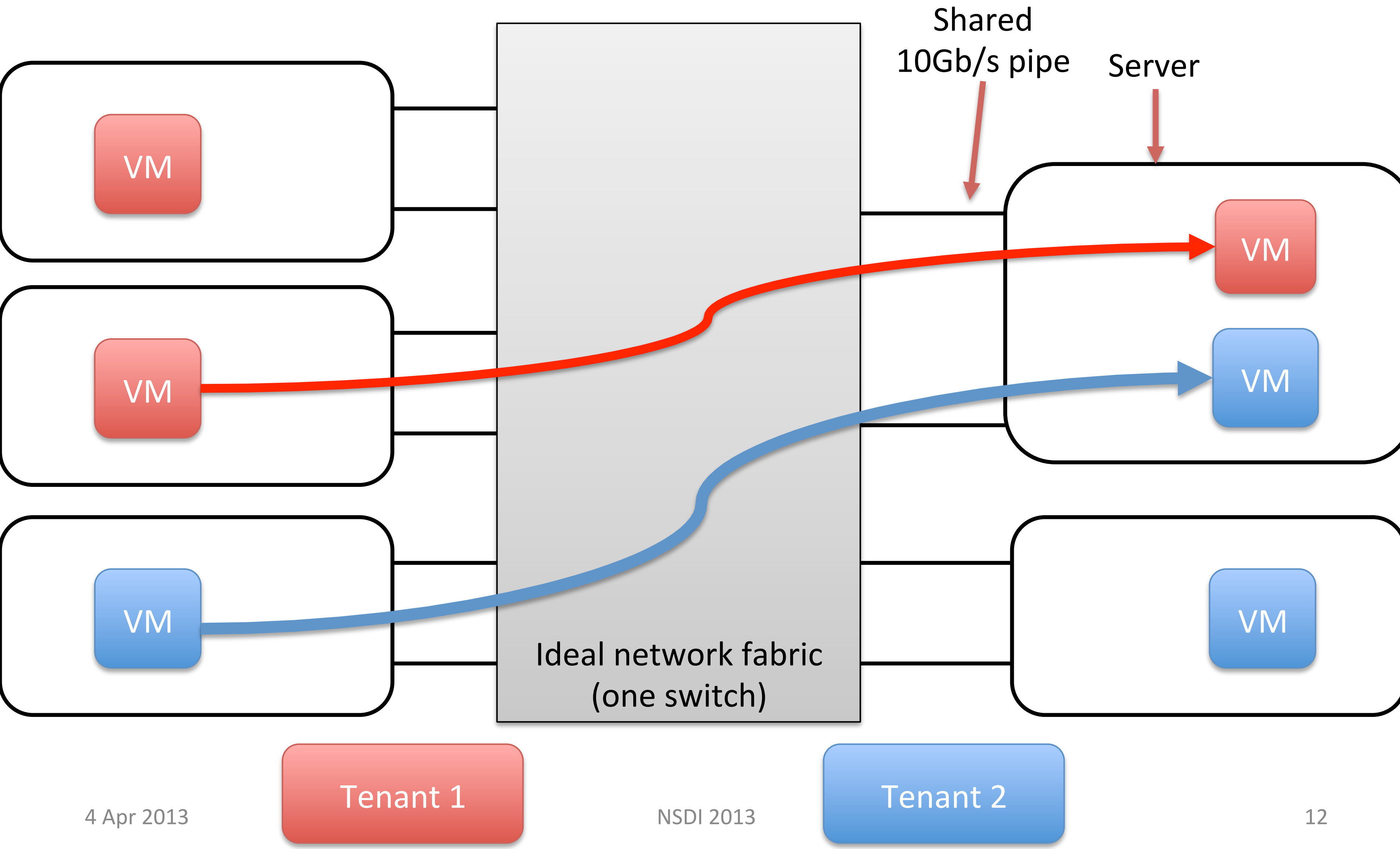
Status Quo is Insufficient

- TCP
  - Cannot force all to use TCP or agree on one TCP version!
  - Sharing is per-flow: not built for predictability
- Performance Isolation with Per-tenant Queues
  - State management complexity: >10k tenants, configuring queues on all links is an operational nightmare
  - WFQ/DRR does not ensure admissibility

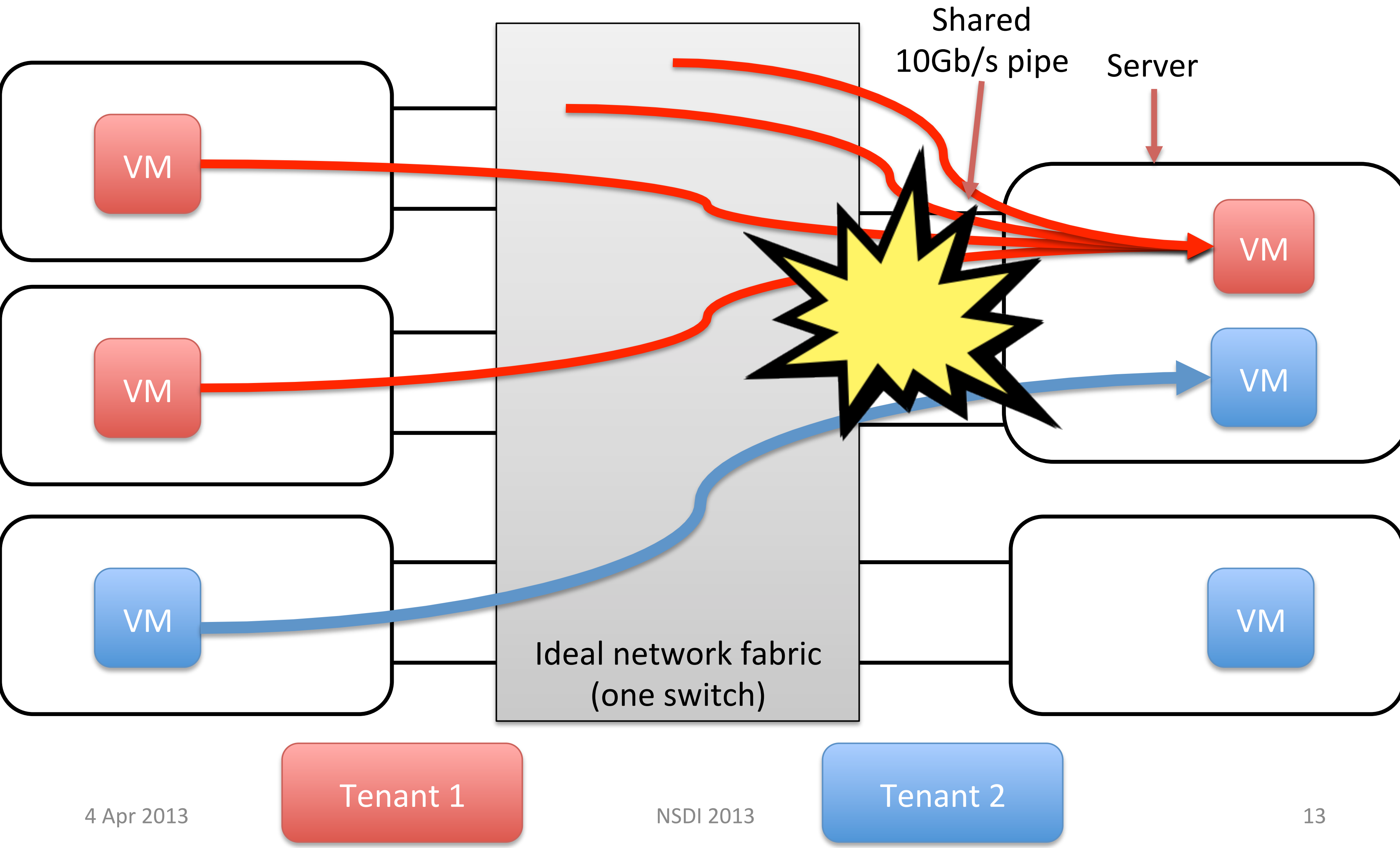
Status Quo is Insufficient

- TCP
  - Cannot force all to use TCP or agree on one TCP version!
  - Sharing is per-flow: not built for predictability
- Performance Isolation with Per-tenant Queues
  - State management complexity: >10k tenants, configuring queues on all links is an operational nightmare
  - WFQ/DRR does not ensure admissibility

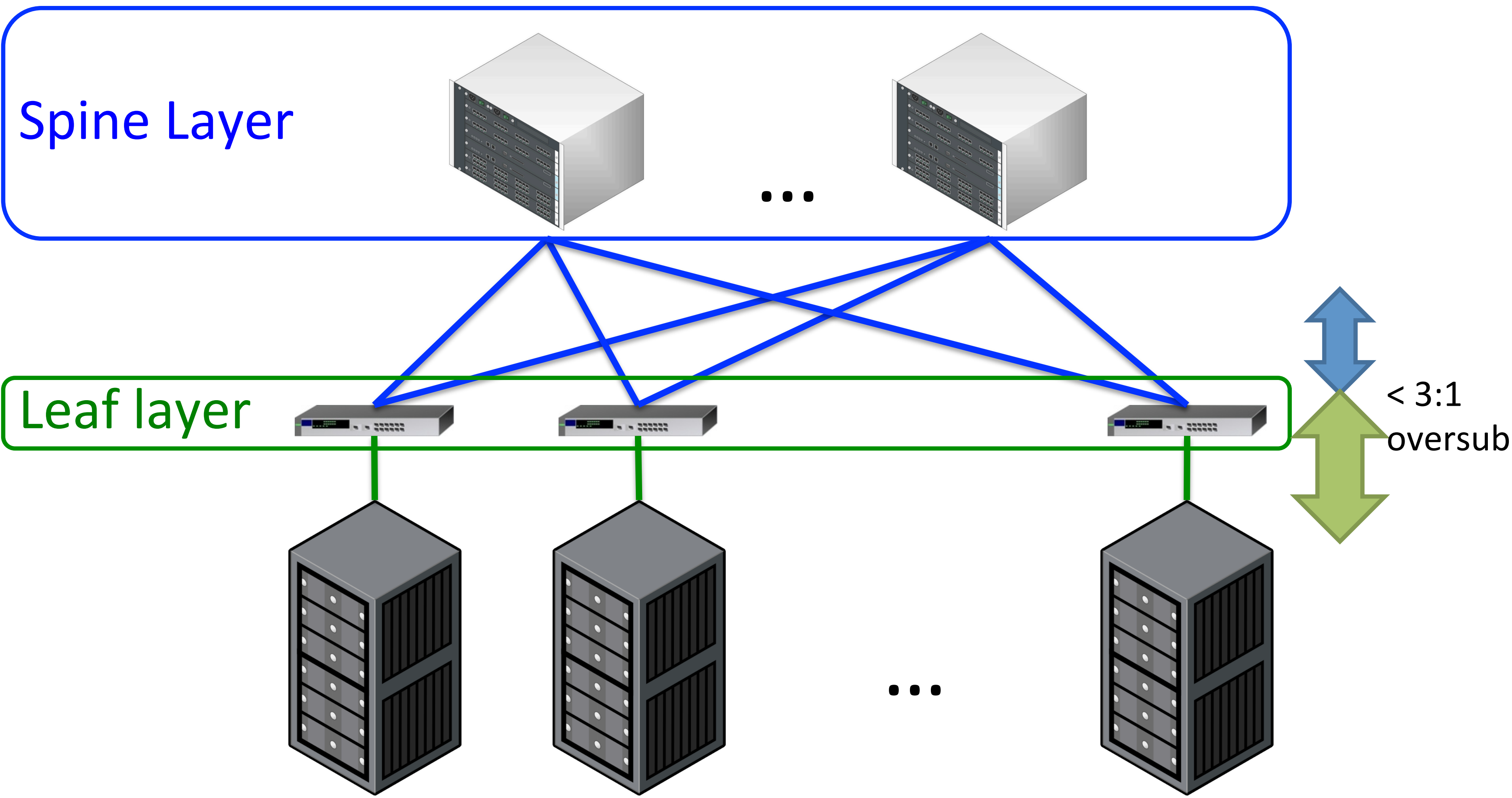
Where does Congestion Happen?

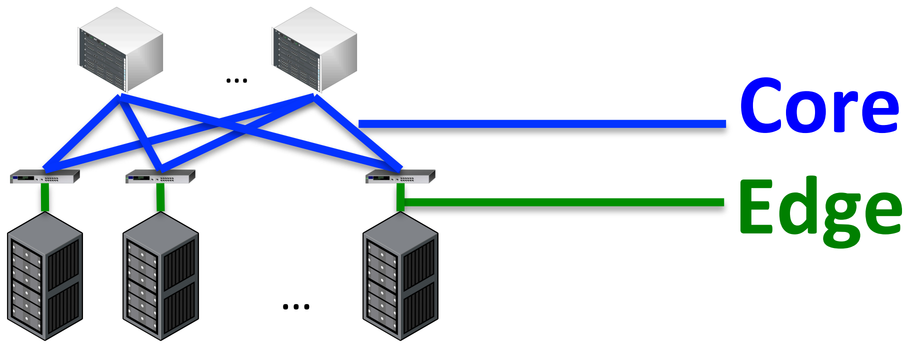


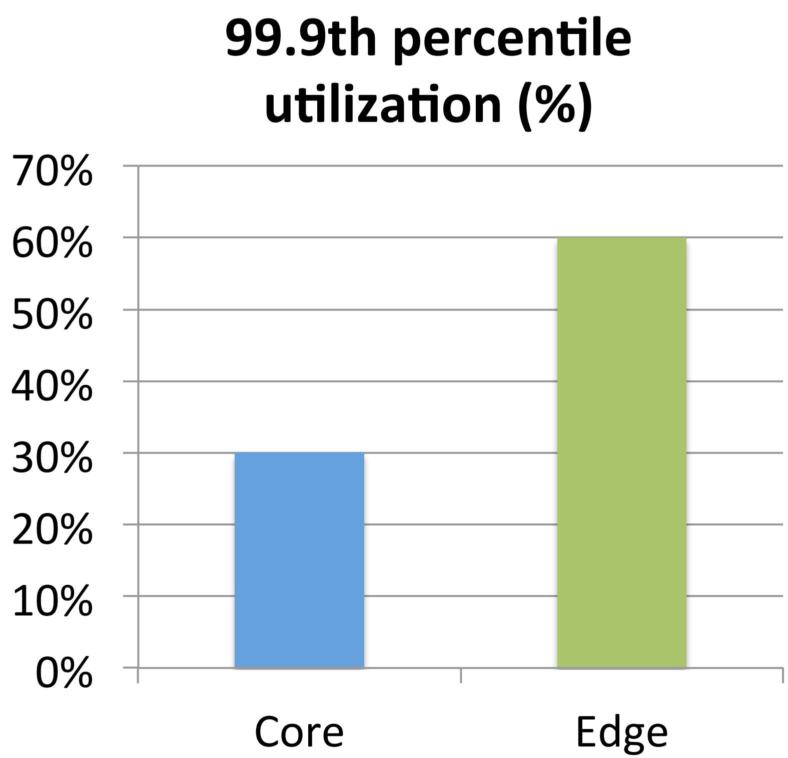
Where does Congestion Happen?



Congestion Study on Windows Azure







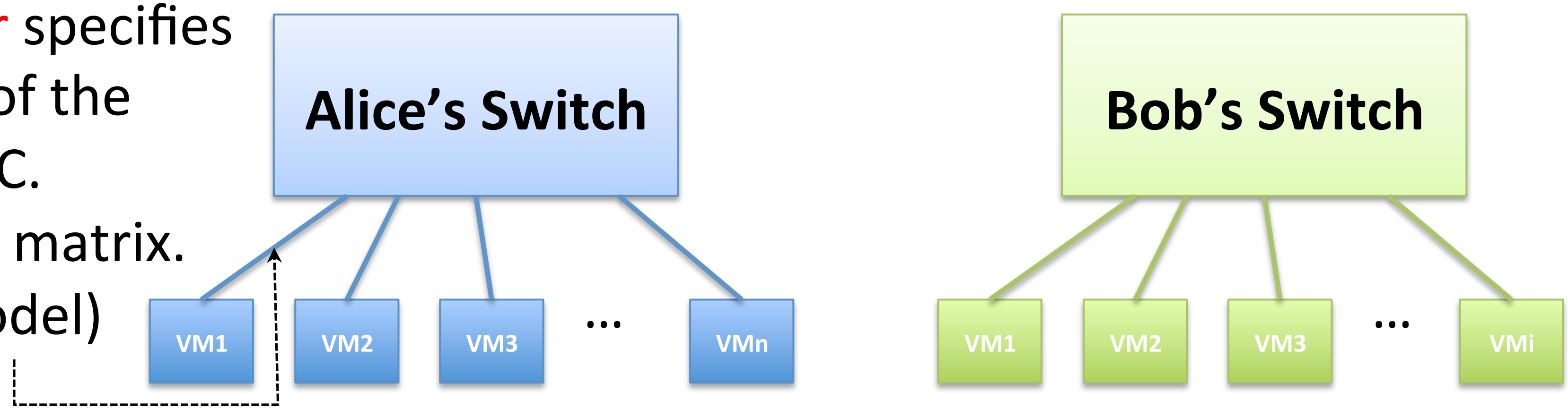
Timescales: over 2 weeks,
99.9th pcile = several minutes

Hottest storage cluster:
1000x more drops at the Edge, than Core.

16 of 17 clusters:
0 drops in the Core.

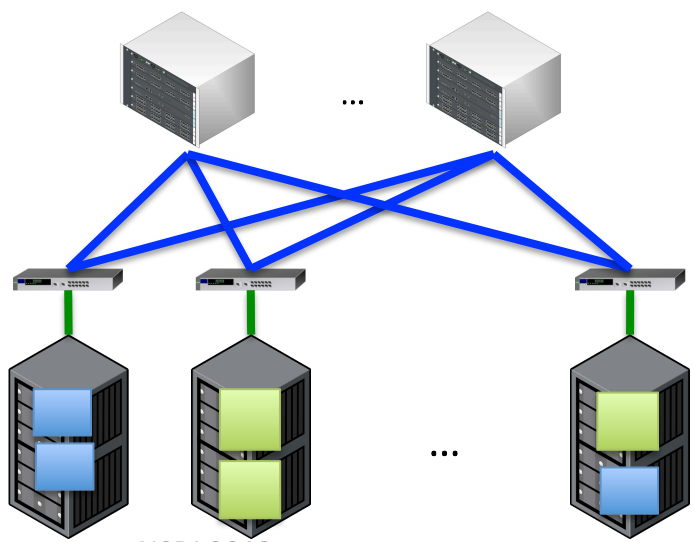
EyeQ: Predictable Bandwidth Partitioning at the Edge

Customer specifies capacity of the virtual NIC.
No traffic matrix.
(Hose Model)



Provider: assures near dedicated performance.

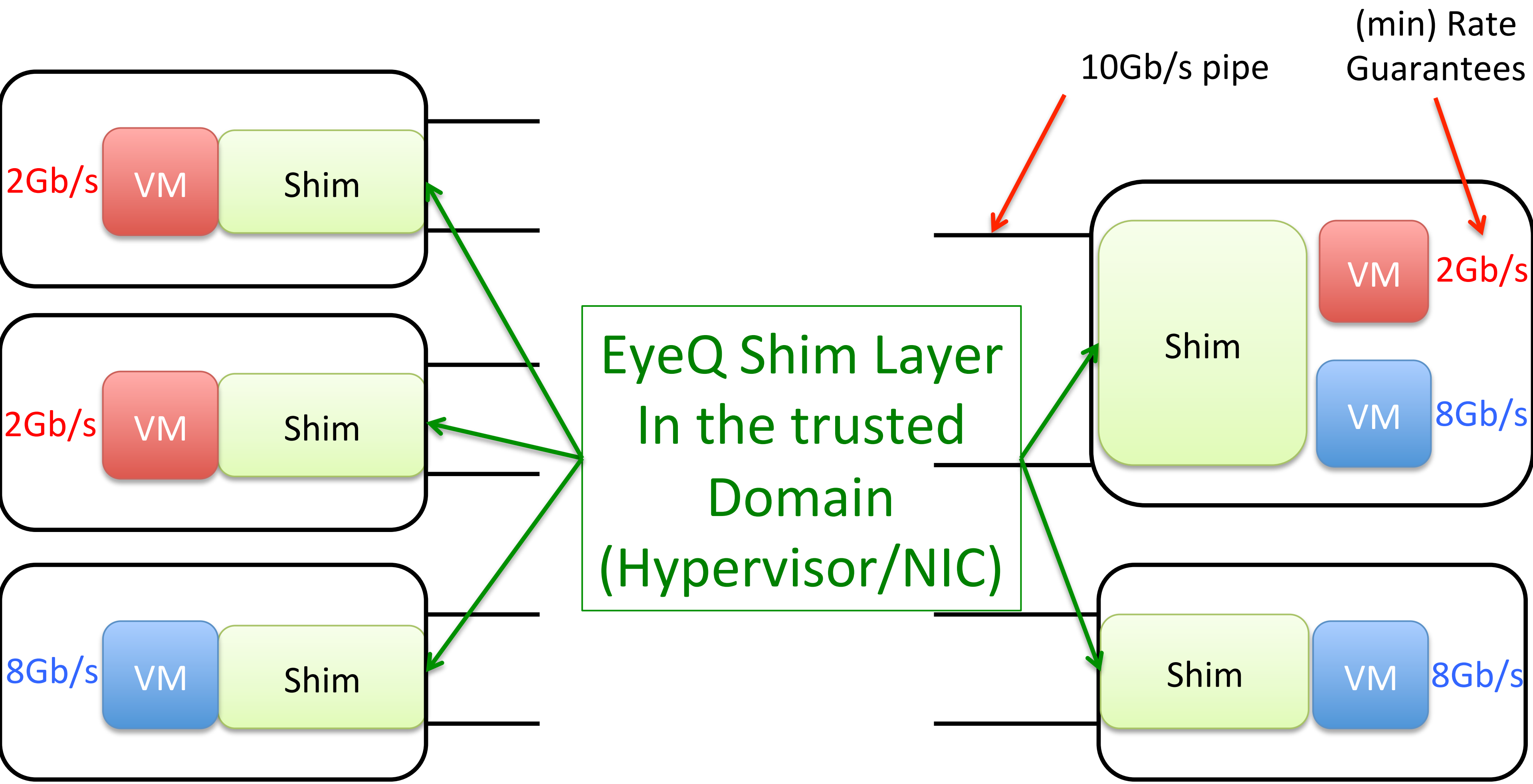
EyeQ is deployable ***today*** at the Edge.



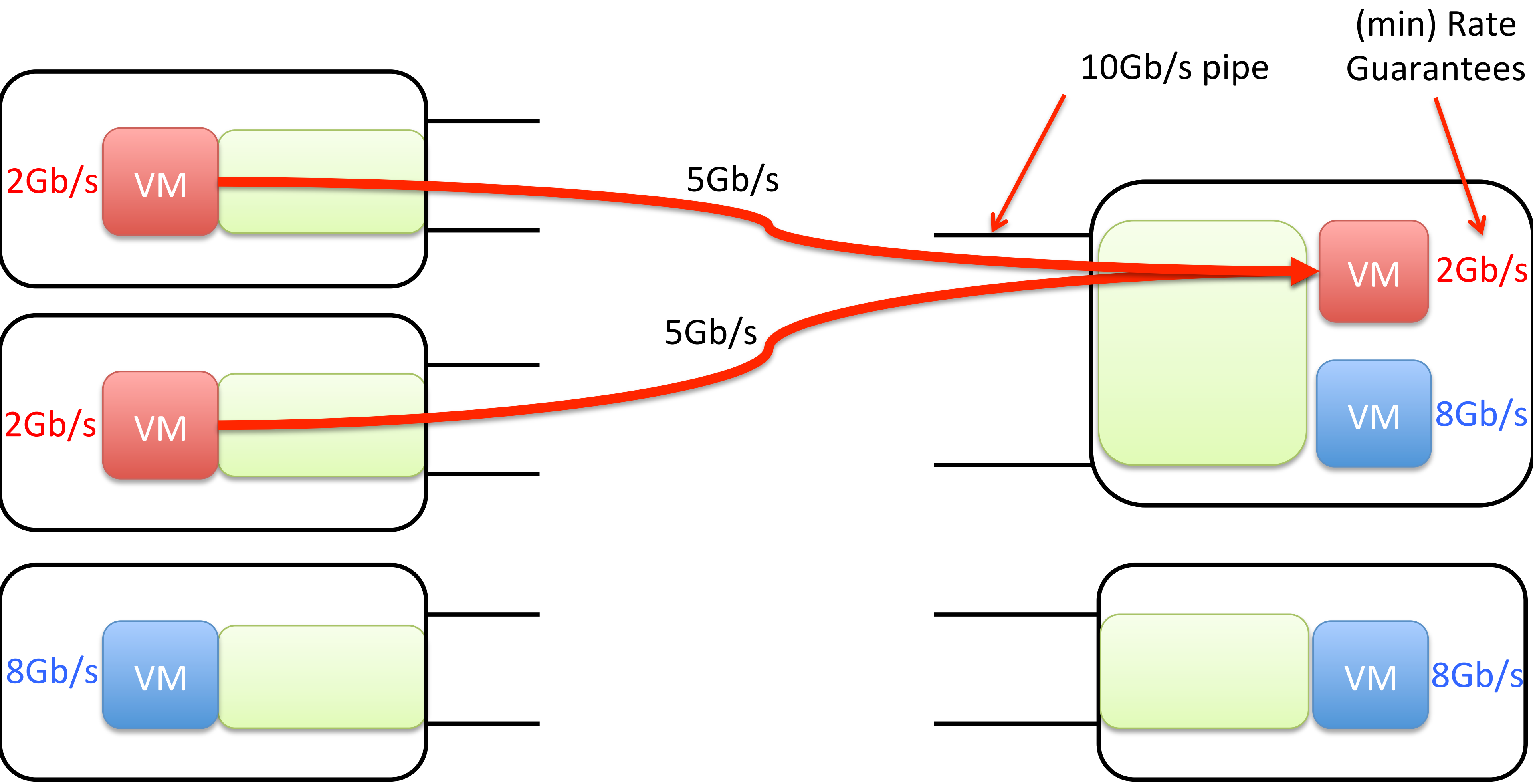
EyeQ's Key Contribution: Simplicity

- Observation
  - Network Congestion predominantly occurs at the Edge (Hypervisor / Top of Rack)
- Consequences: Simplicity
  - Distributed, end-to-end bandwidth allocation
    - Amenable to NIC-based implementation
  - Network need not be tenant aware
- Implementation
  - High speed in software at 10Gb/s

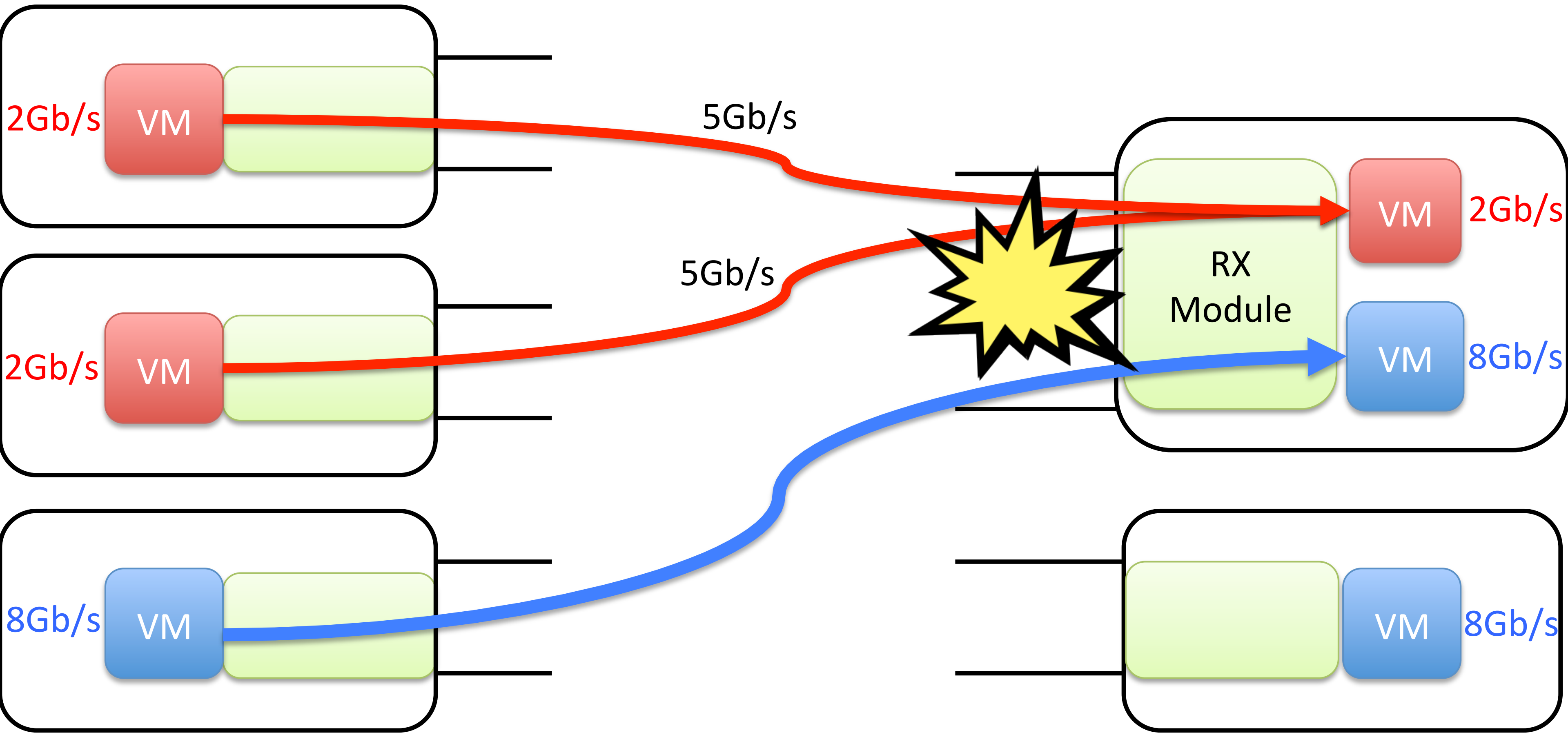
Decentralized Scheduling



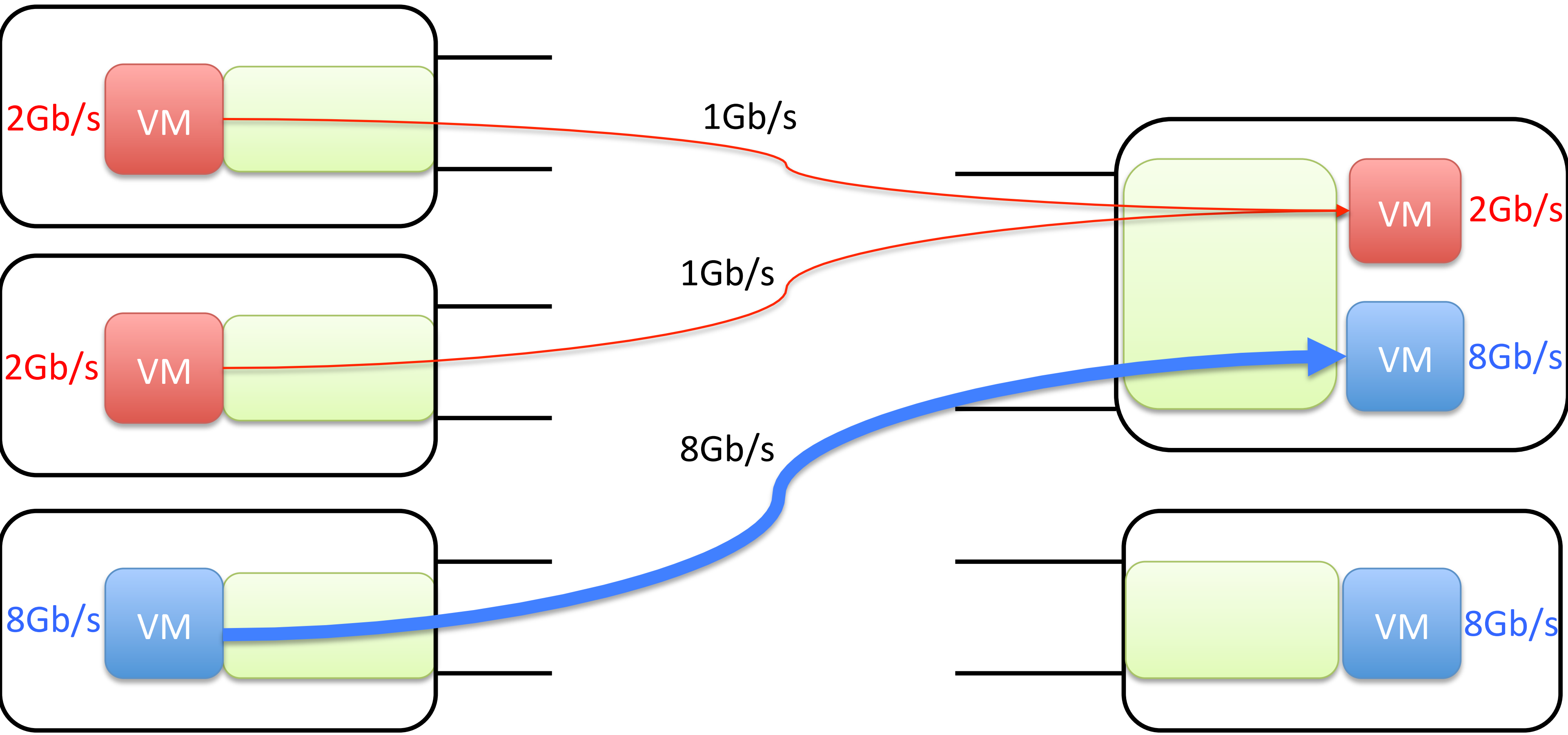
Decentralized Scheduling



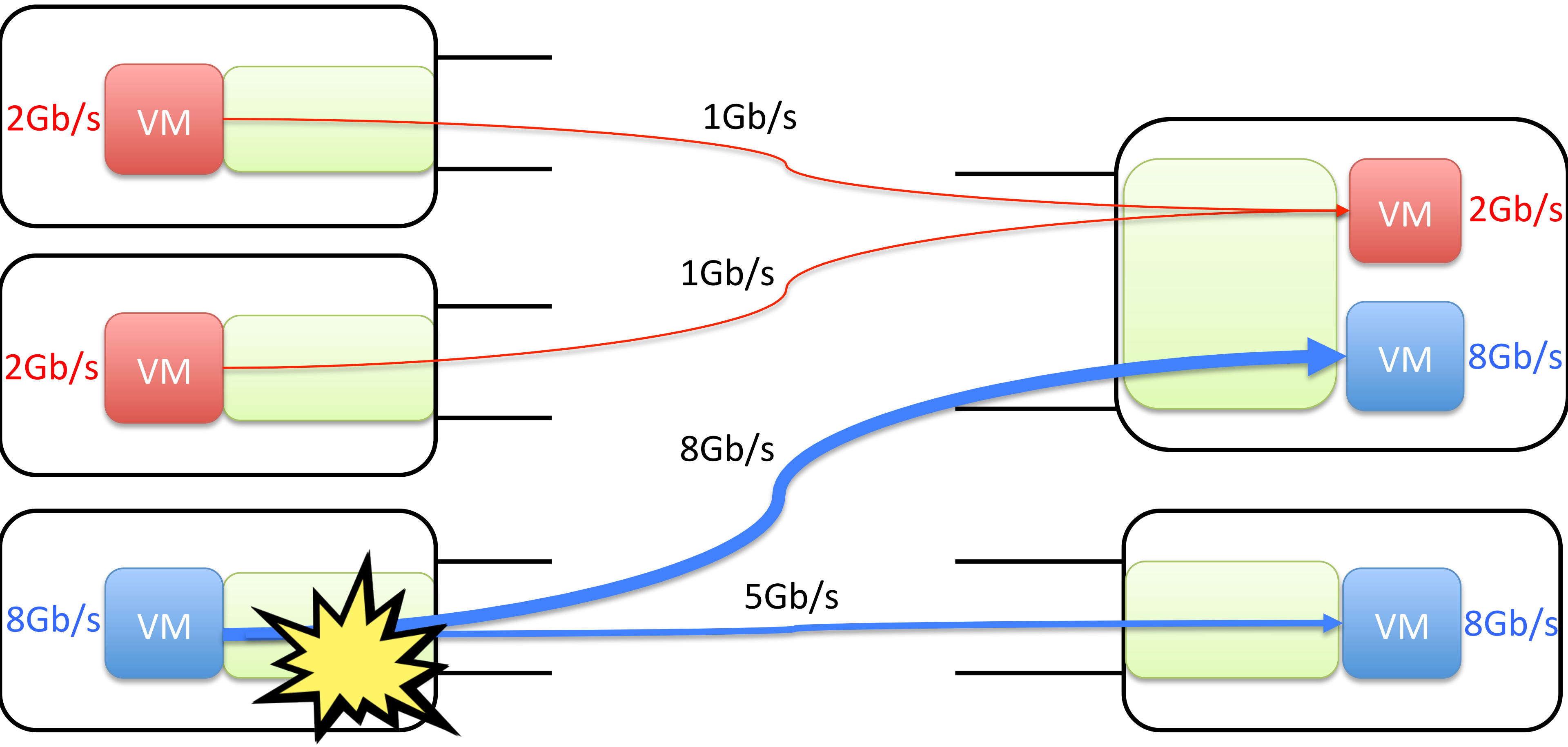
Decentralized Scheduling



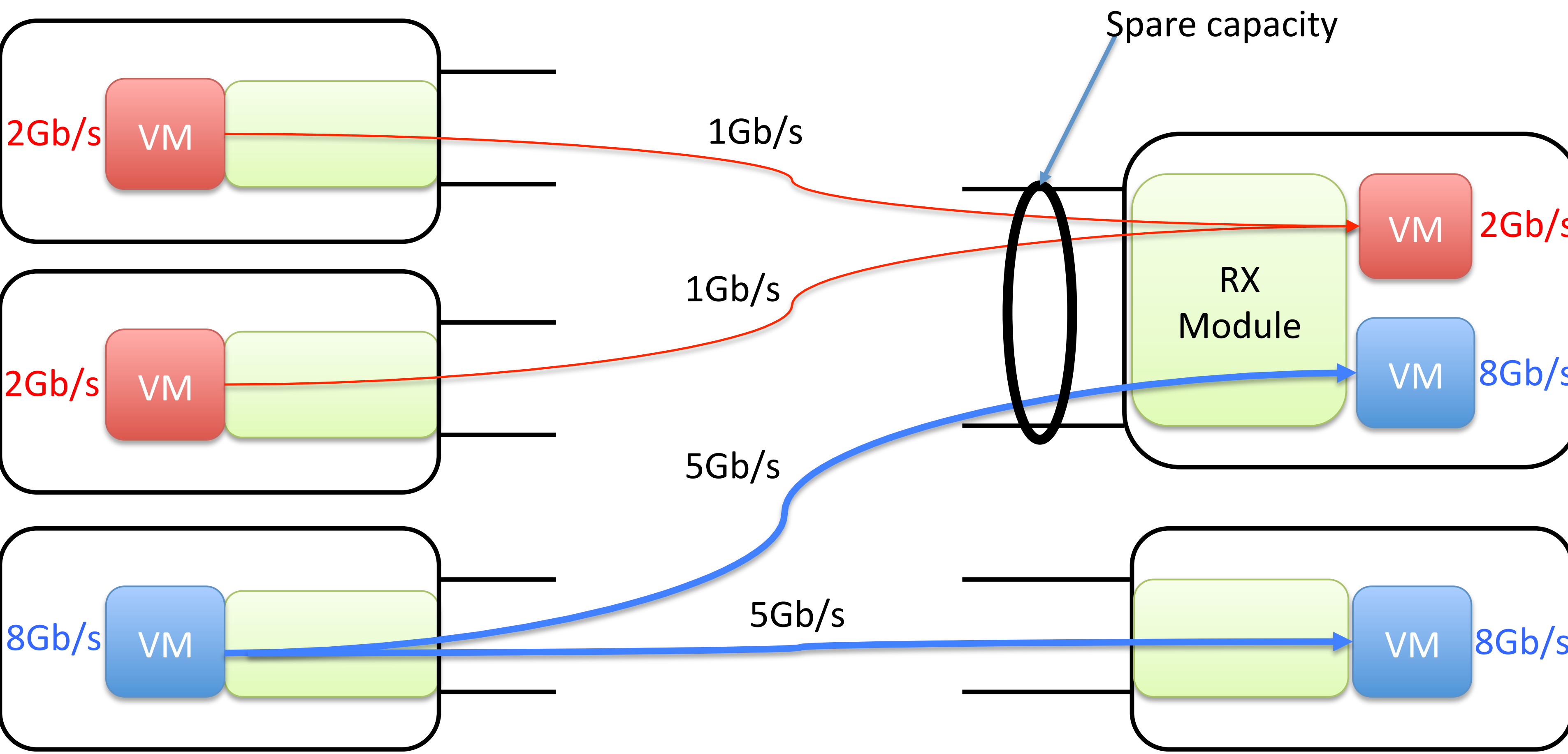
Decentralized Scheduling



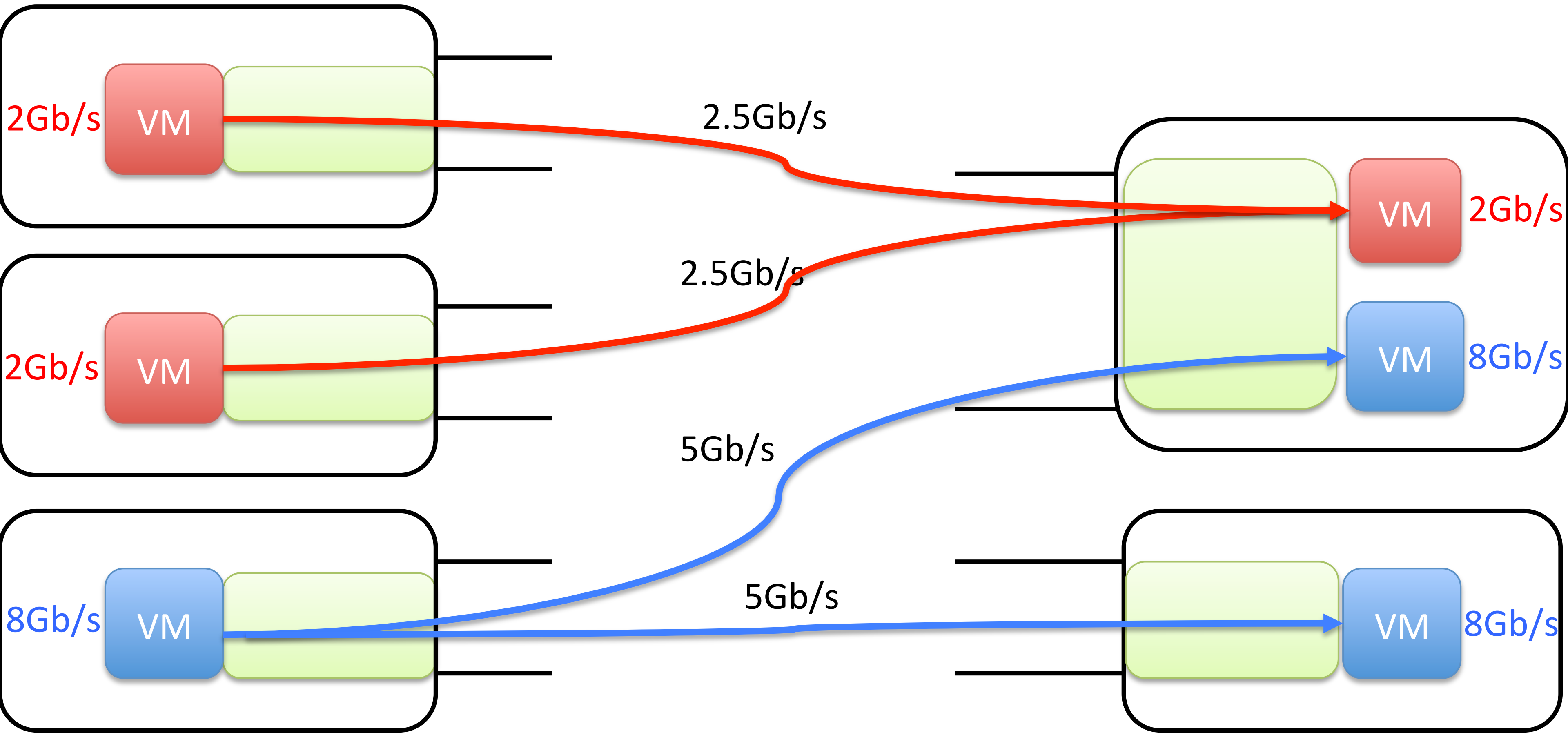
Decentralized Scheduling



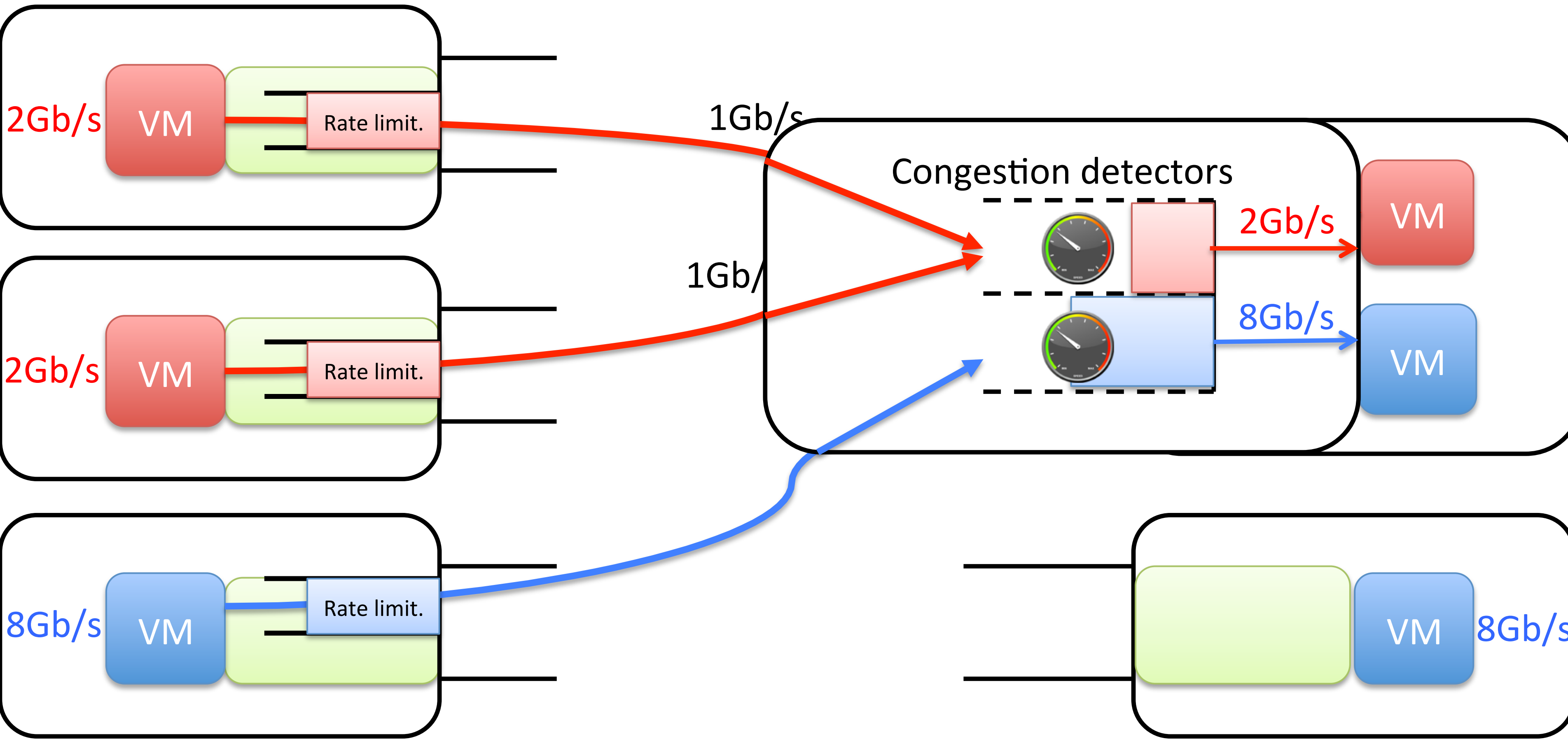
Work Conserving Allocations



Work Conserving Allocations



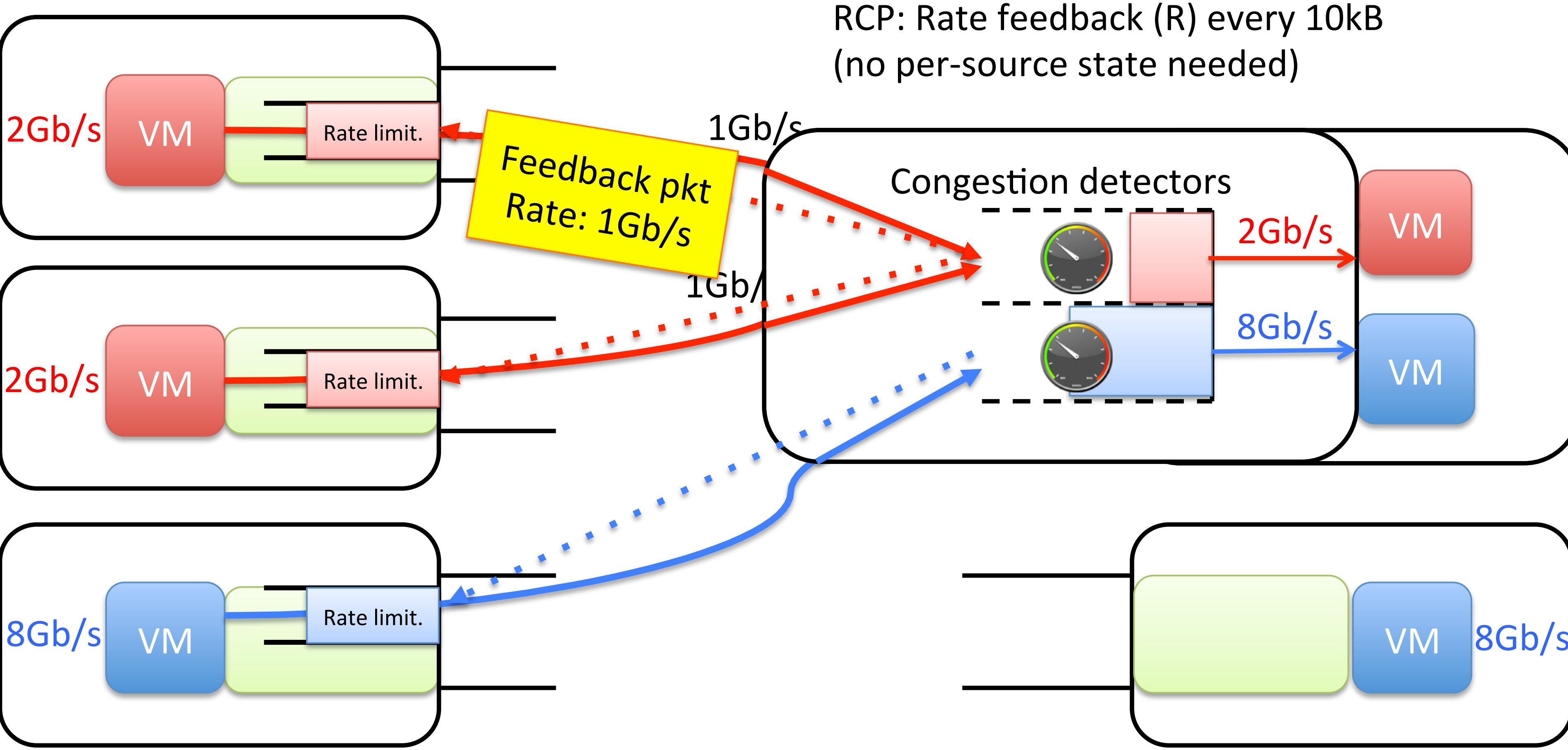
Transmit/Receive Modules



Per-destination rate limiters:
only if dest. is congested… bypass otherwise

Transmit/Receive Modules

RCP: Rate feedback (R) every 10kB
(no per-source state needed)



Per-destination rate limiters:
only if dest. is congested… bypass otherwise

Timescales Matter

- Fast convergence important
  - Switches only have few MB (milliseconds) worth of buffering before they drop packets
- RCP's worst-case convergence time
  - N long lived flows competing for a single bottleneck: few milliseconds.
  - Usually few 100 microseconds.

But what if the Core gets congested?

How? → Transient failures or ECMP collisions

Case 1: Mild network congestion

- Use ECN for graceful fallback
  - Per receiver-VM max-min sharing
  - Congestion detector: multiplicative decrease on advertised rate on receiving ECN

Case 2: Severe network congestion (unlikely!)

- Multiplicative decrease (rate limiter timeout)

Software Prototype

Linux Kernel Module (qdisc)

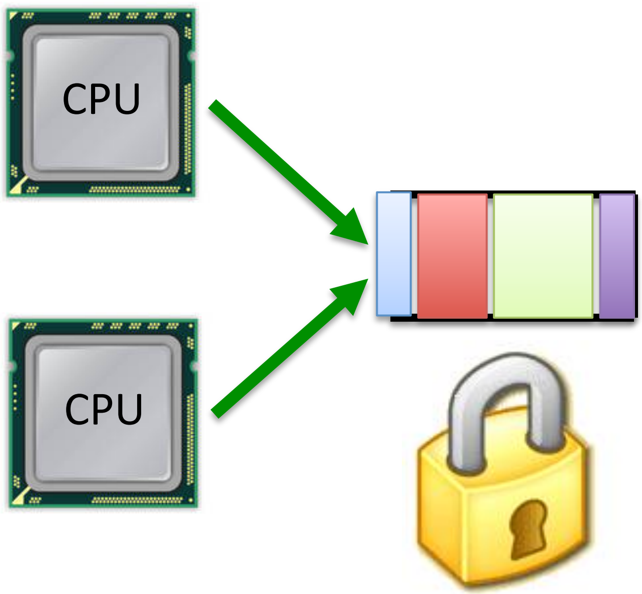
Windows Filter Driver (in VMSwitch)

- Non-intrusive: no changes to applications or existing network stack. Works even with UDP.
- ~1700 lines of code

Linux Kernel Module is Open-Source

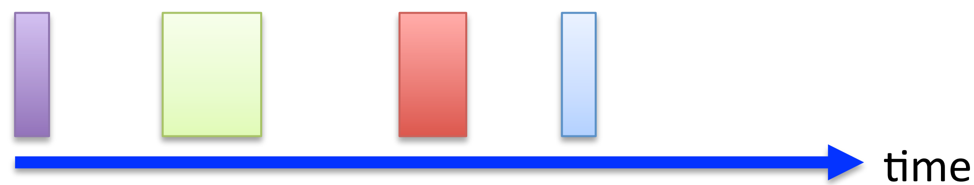
- Full system and documentation at http://jvimal.github.com/eyeq
- Fully functional version in Mininet to play with ☺

High speed software rate limiters



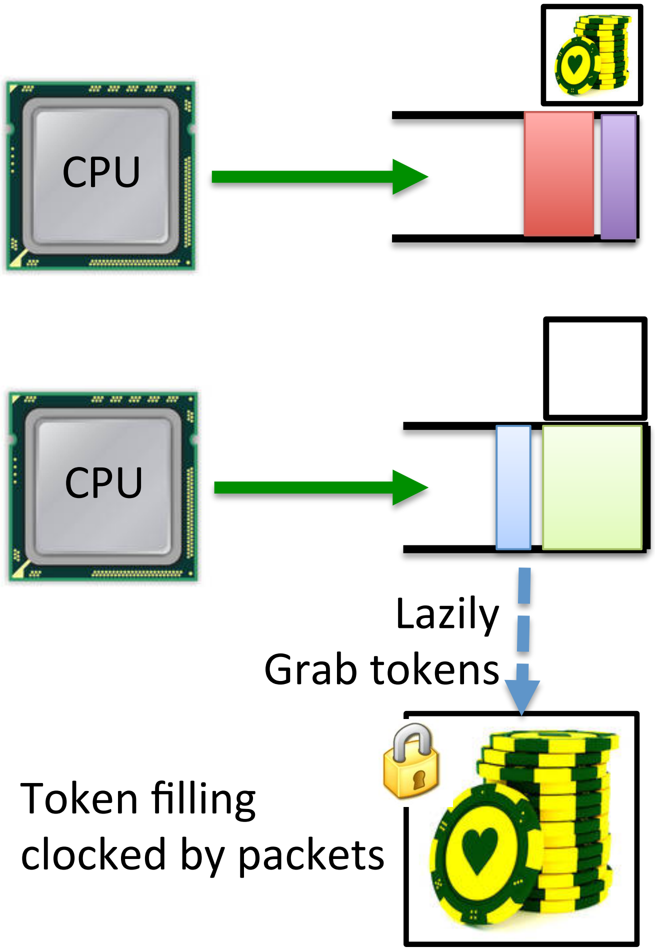
Single shared queue increases lock contention

- High CPU overhead
- High packet latency
- Controlled burst



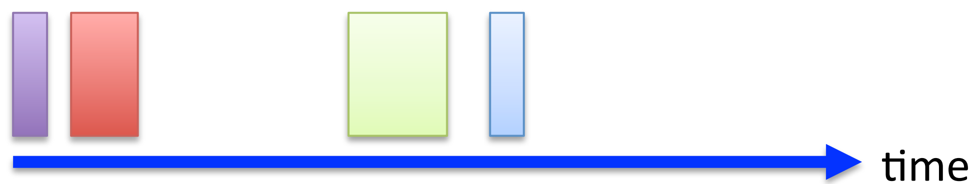
Packets on the wire

Parallel transmit path



Split queue to per-cpu queues

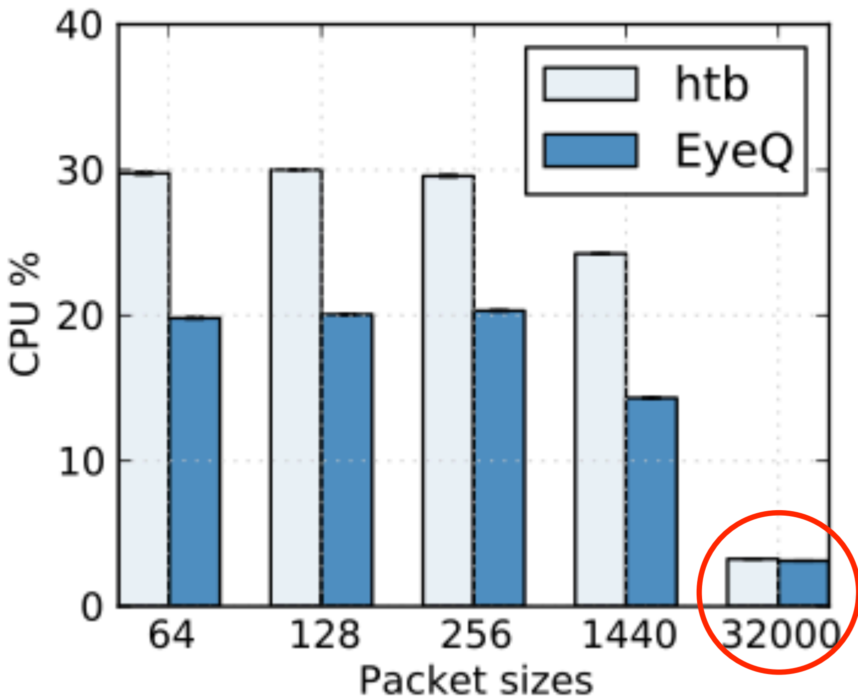
- Lower CPU overhead
- Lower packet latency
- Fairness across CPU queues
- Higher, but bounded burst

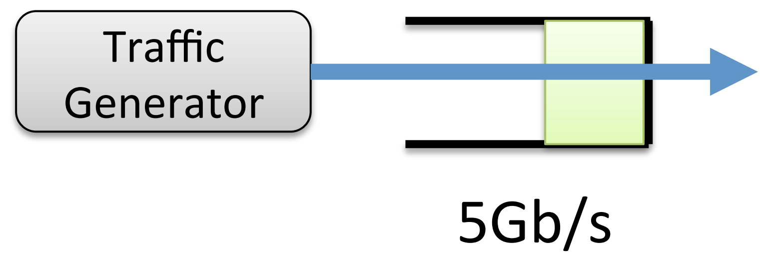


Packets on the wire

Rate Limiter Efficiency

Throughput

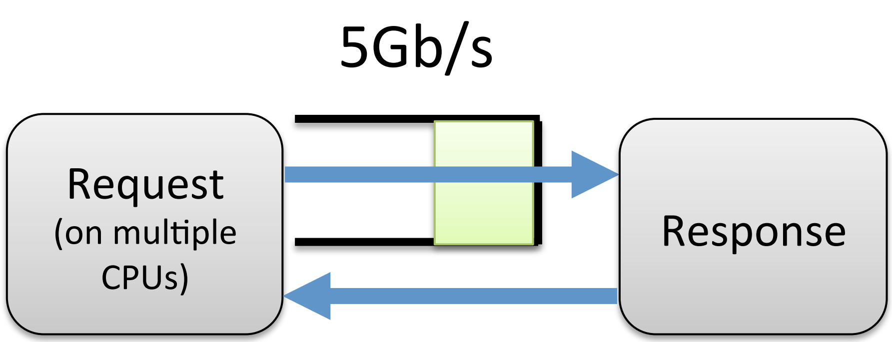




Low lock contention due to
fewer packets/sec and timer interrupts/sec.

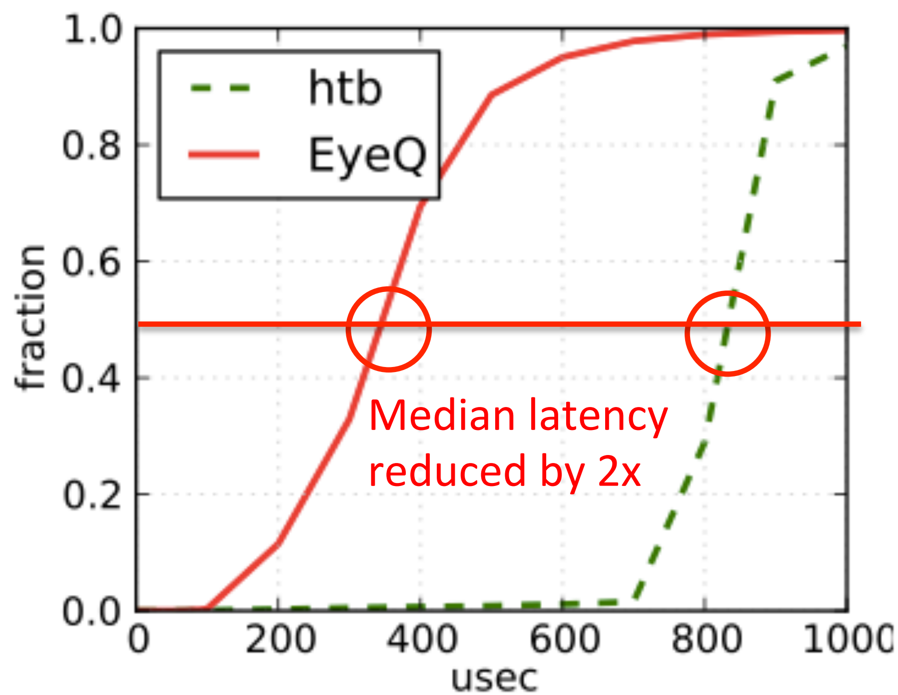
Single rate limiter at 5Gb/s.
HTB succumbs at 9Gb/s.

Rate Limiter Efficiency



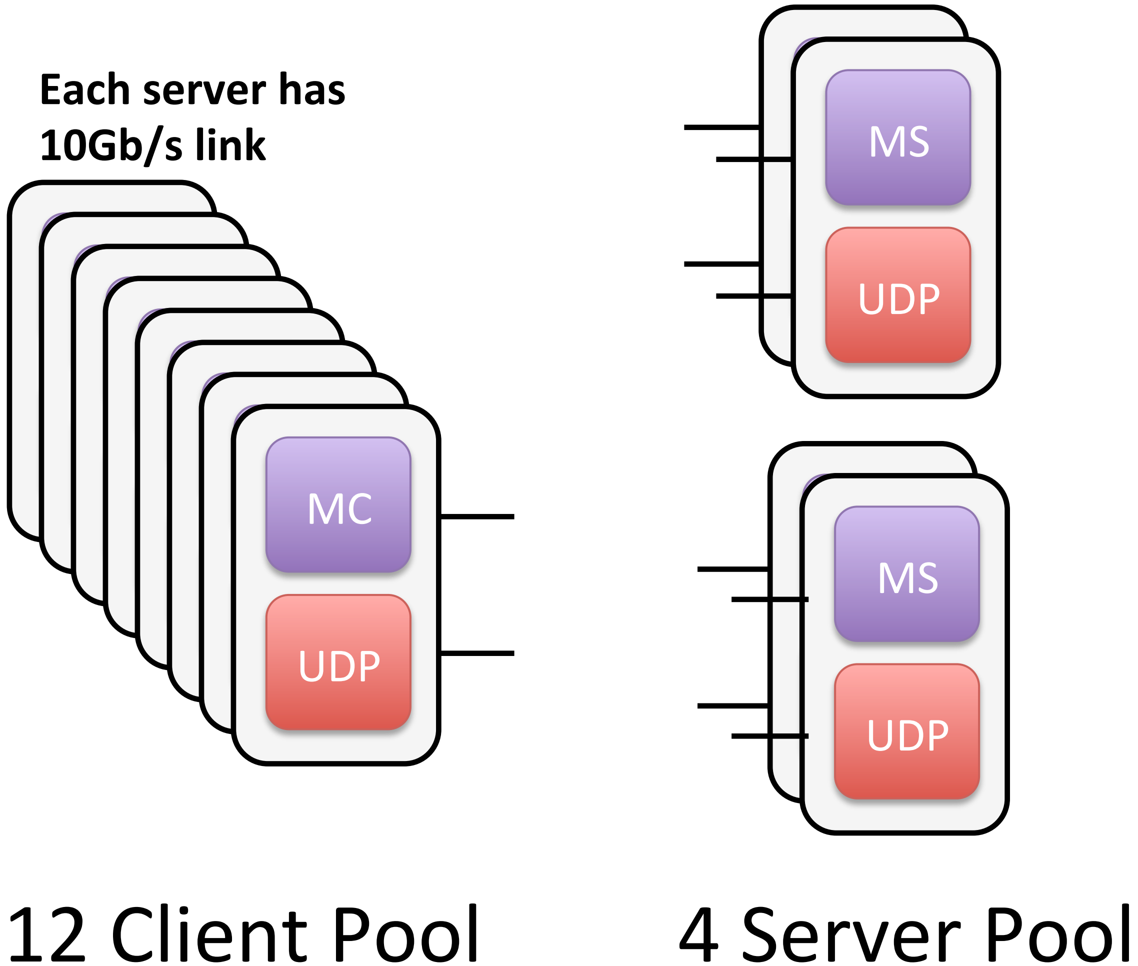
Input rate to rate limiter limited by end-to-end latency.

Latency

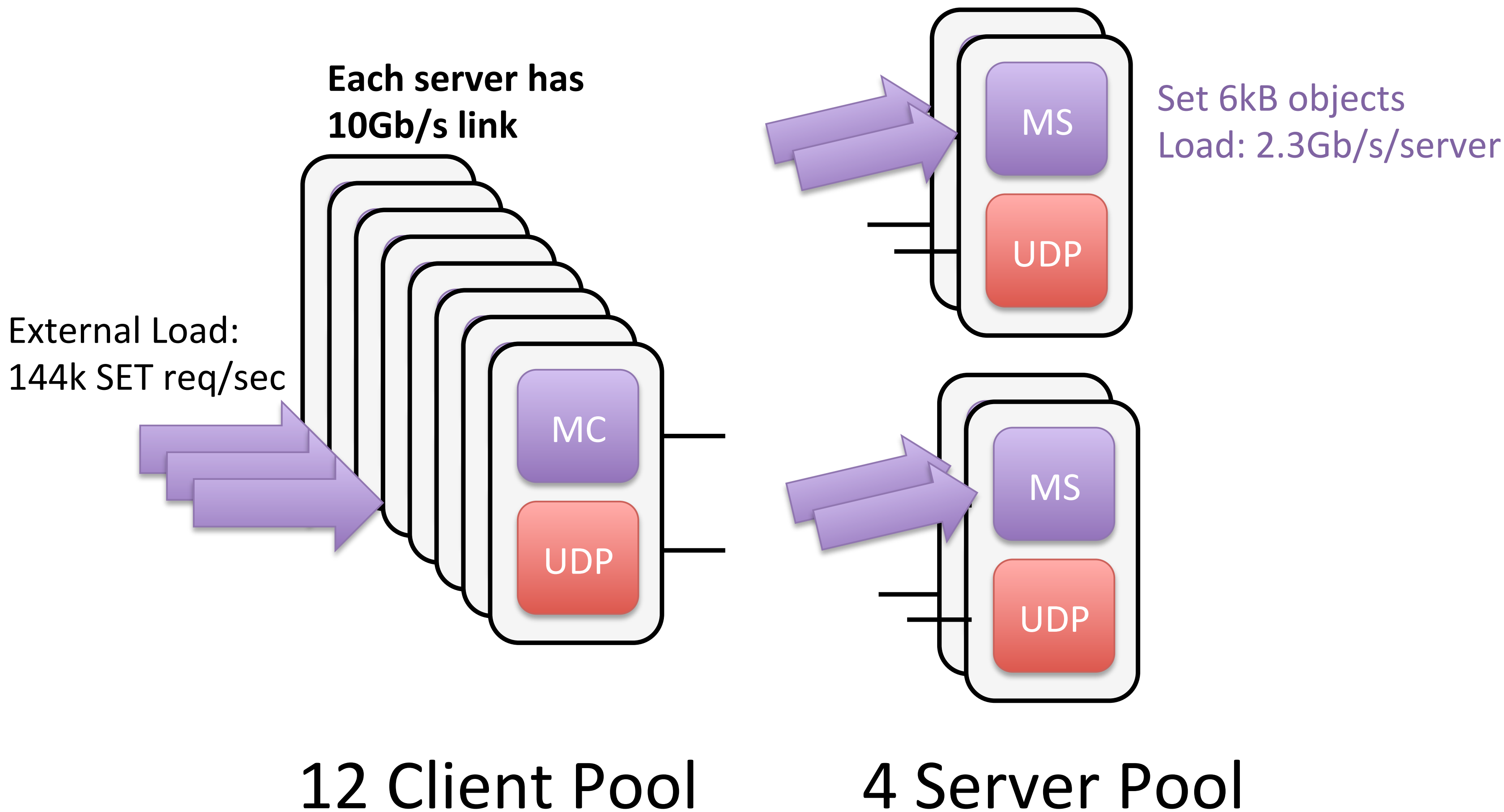


1B synchronous request response loop.

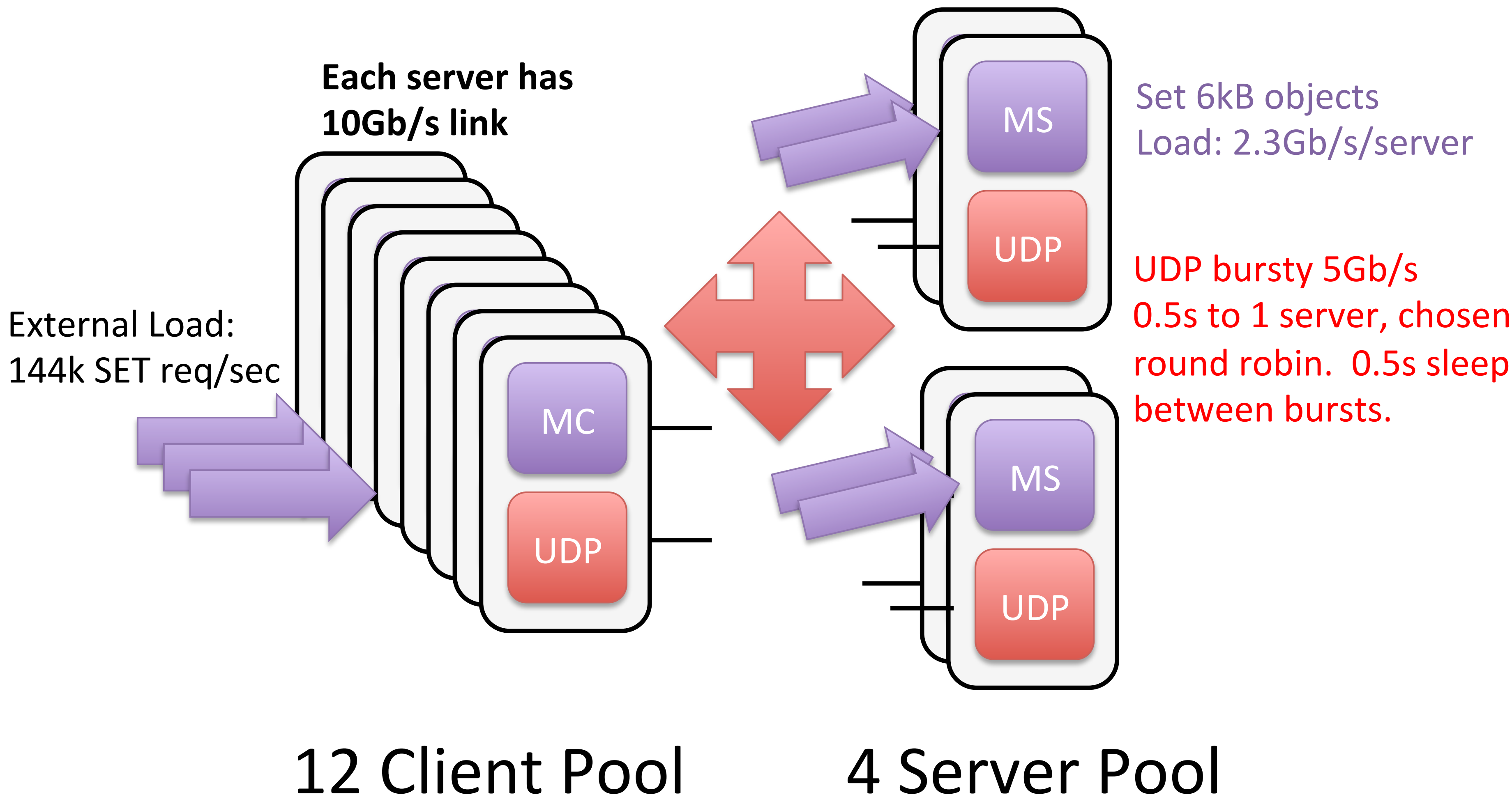
Macro Evaluation: Memcached Latency



Macro Evaluation: Memcached Latency

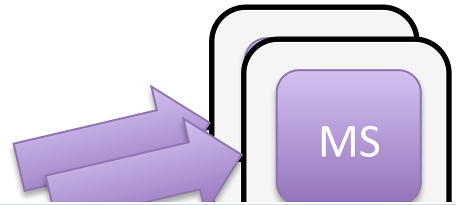


Macro Evaluation: Memcached Latency



Macro Evaluation: Memcached Latency

Each server has 10Gb/s link



Set 6kB objects
Load: 2.3Gb/s/server

E
1

Scenario	50th	99.9th	Throughput
Baseline (Linux 3.4)	98us	666us	144kreq/s
Without Interference + EyeQ	100us	**630**us	144kreq/s
With Interference	**4127**us	**>10⁶**us	144kreq/s
With Interference + EyeQ	102us	**750**us	144kreq/s

12 Client Pool 4 Server Pool

Thank you!

EyeQ: An edge-based flow scheduler for the data center…

to partition bandwidth in a simple and predictable way.

open source™

http://jvimal.github.com/eyeq

jvimal@stanford.edu