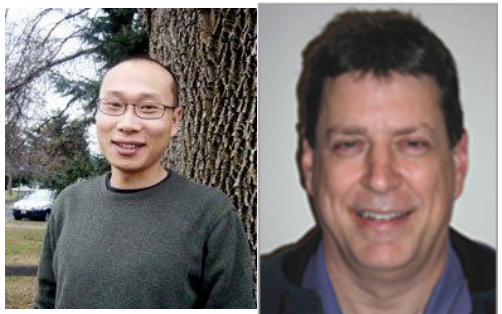
EyeQ
Protecting Network Performance in the Cloud

Vimal

Mohammad Alizadeh
Balaji Prabhakar
David Mazières

Changhoon Kim
Albert Greenberg



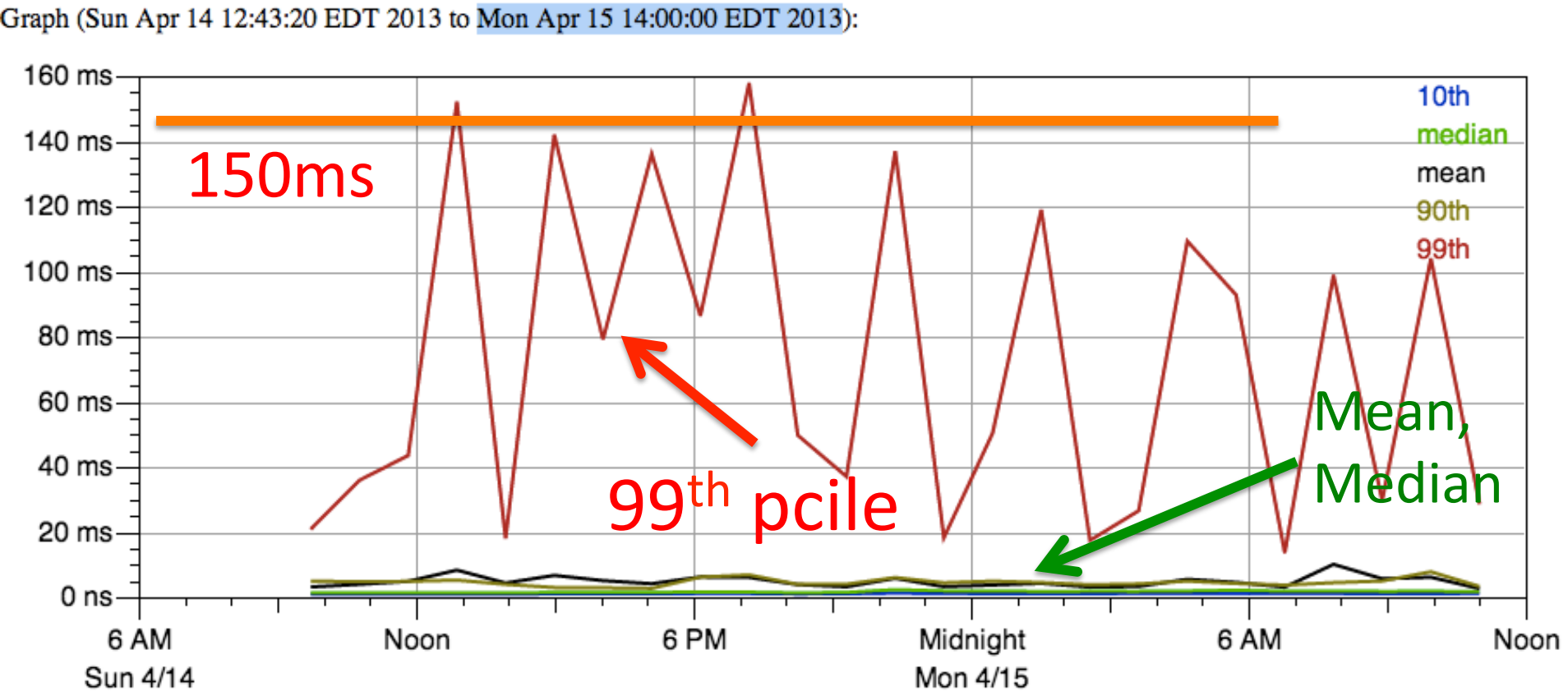
Once upon a time…

Once upon a time…

Once upon a time…

Onc time…

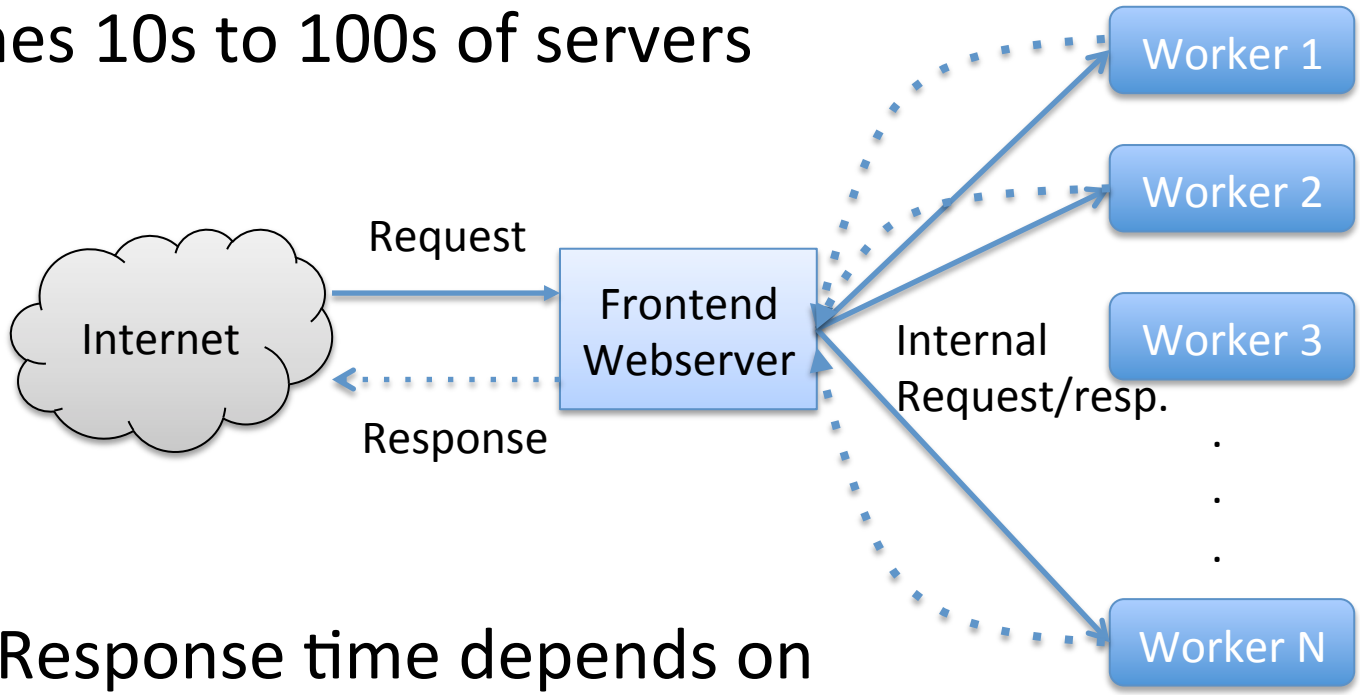
Performance Unpredictability



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

99th percentile latency: Who cares?

Web services: each request touches 10s to 100s of servers



Web Response time depends on the slowest worker.

As N increases, 99th percentile latency really matters

Network Congestion Kills Predictability

Is this how we deal with variability?

Is this how we deal with variability?

Why We Moved Off The Cloud

The cloud's intractable problem

… is variable — no, highly variable — performance.

`http://code.mixpanel.com/2011/10/27/why-we-moved-off-the-cloud/`

Give up on cloud, move to dedicated

Is this how we deal with variability?

Why We Moved Off The Cloud

The cloud's intractable problem

… is variable — no, highly variable — performance.

`http://code.mixpanel.com/2011/10/27/why-we-moved-off-the-cloud/`

Give up on cloud, move to dedicated

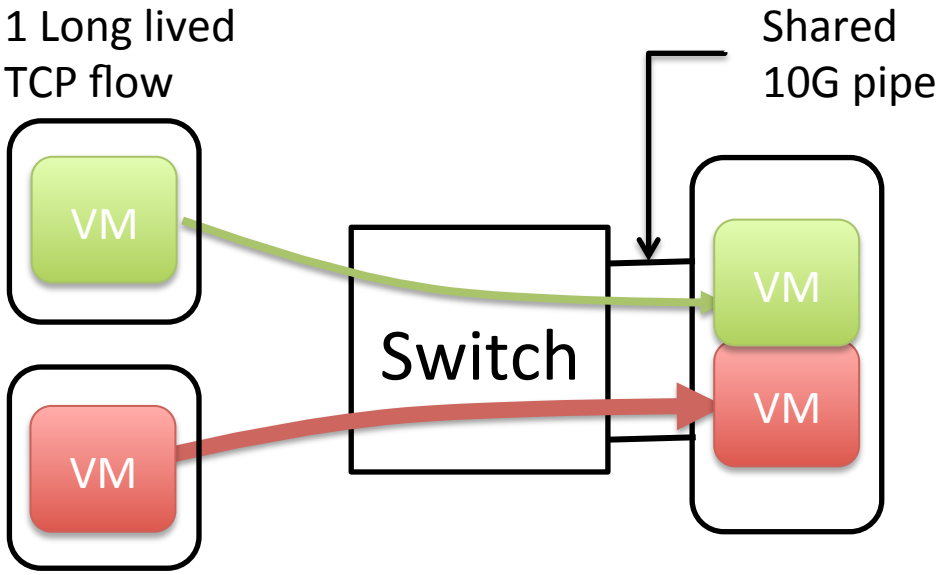
5 Lessons We've Learned Using AWS

… in the Netflix data centers, we have a high capacity, super fast, highly reliable network. This has afforded us the luxury of designing around chatty APIs to remote systems. AWS networking has more variable latency.

`http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html`

Overhaul apps to deal with *variability*

Congestion is notorious you when you can't "see" it

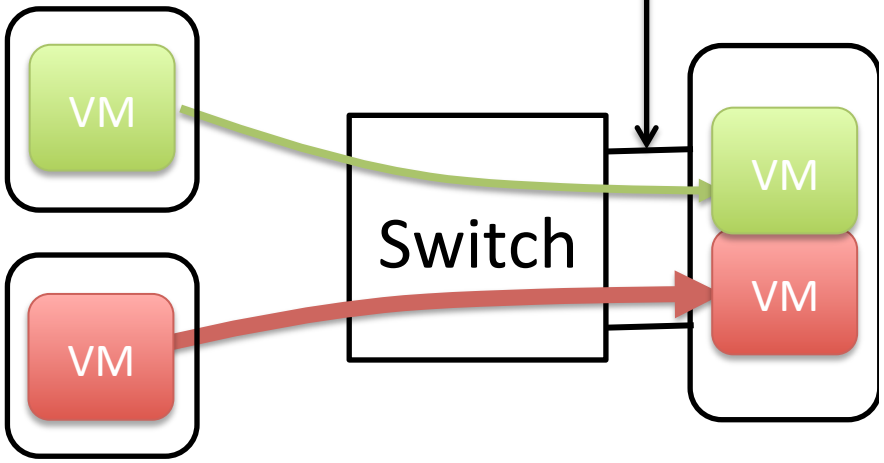


Bursty UDP session:
2.5Gb/s
ON: 5ms
OFF: 15ms

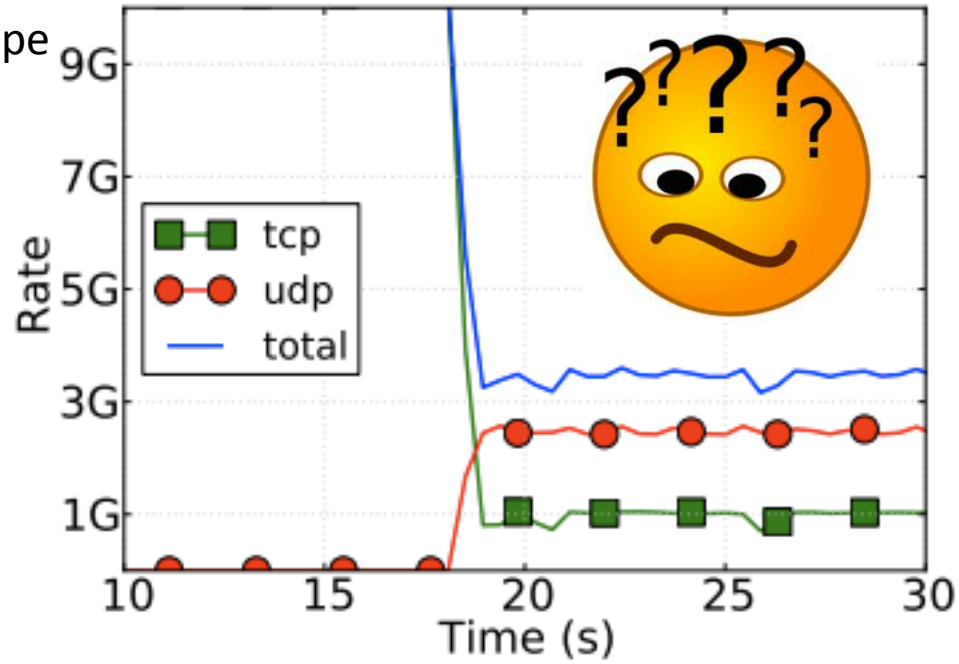
Congestion is notorious you when you can't "see" it

1 Long lived TCP flow

Shared 10G pipe



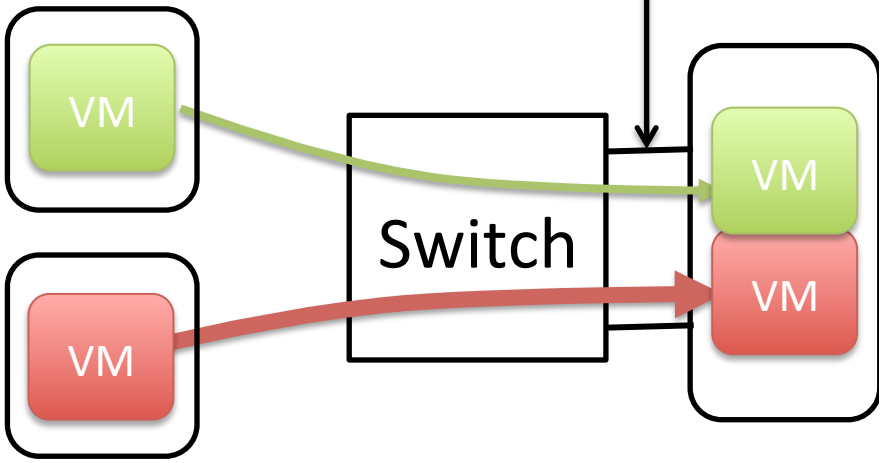
Bursty UDP session:
2.5Gb/s
ON: 5ms
OFF: 15ms



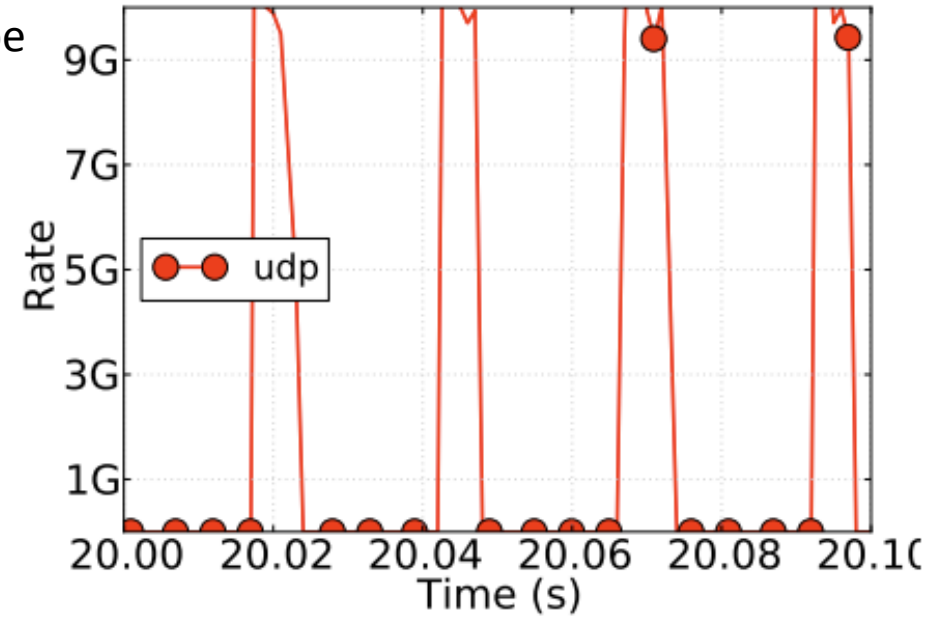
Congestion is notorious you when you can't "see" it

1 Long lived TCP flow

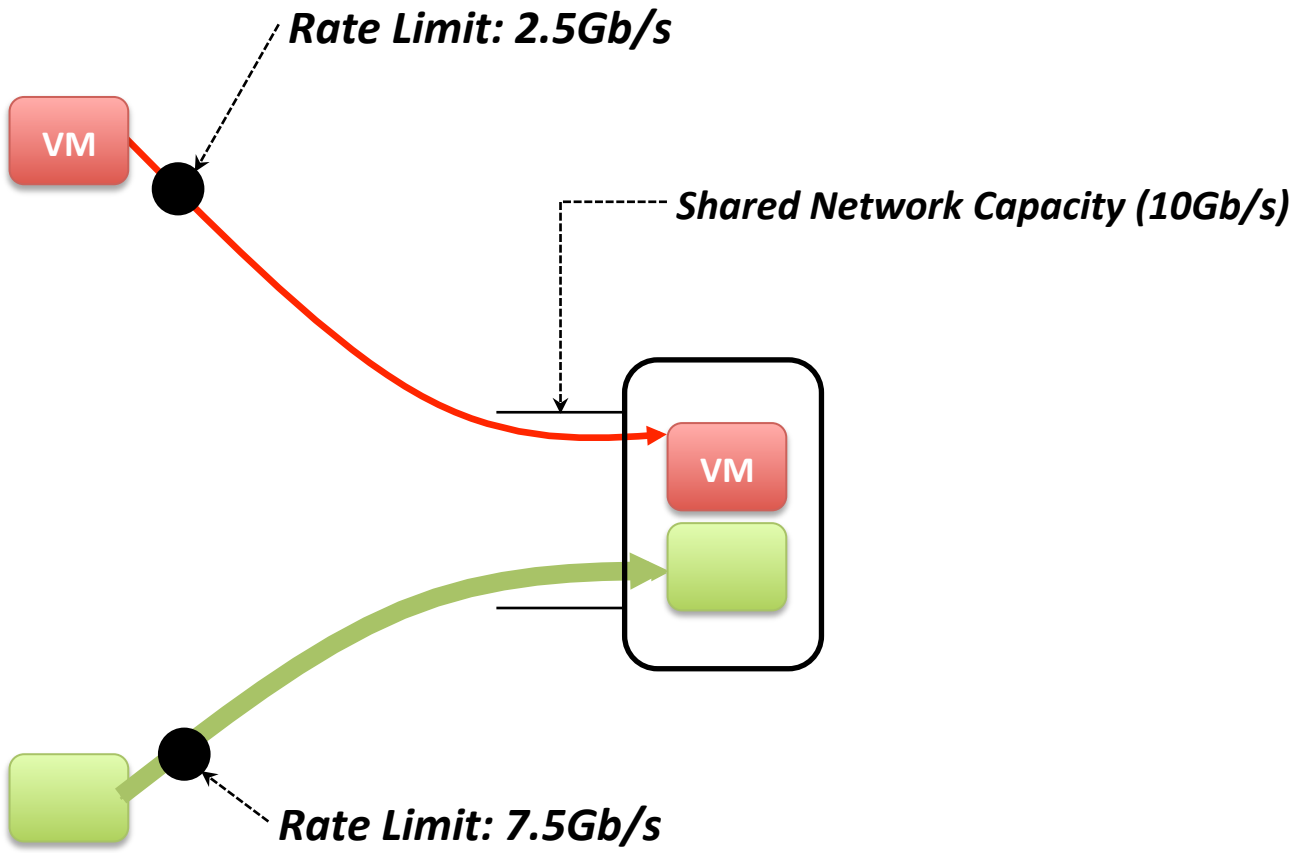
Shared 10G pipe



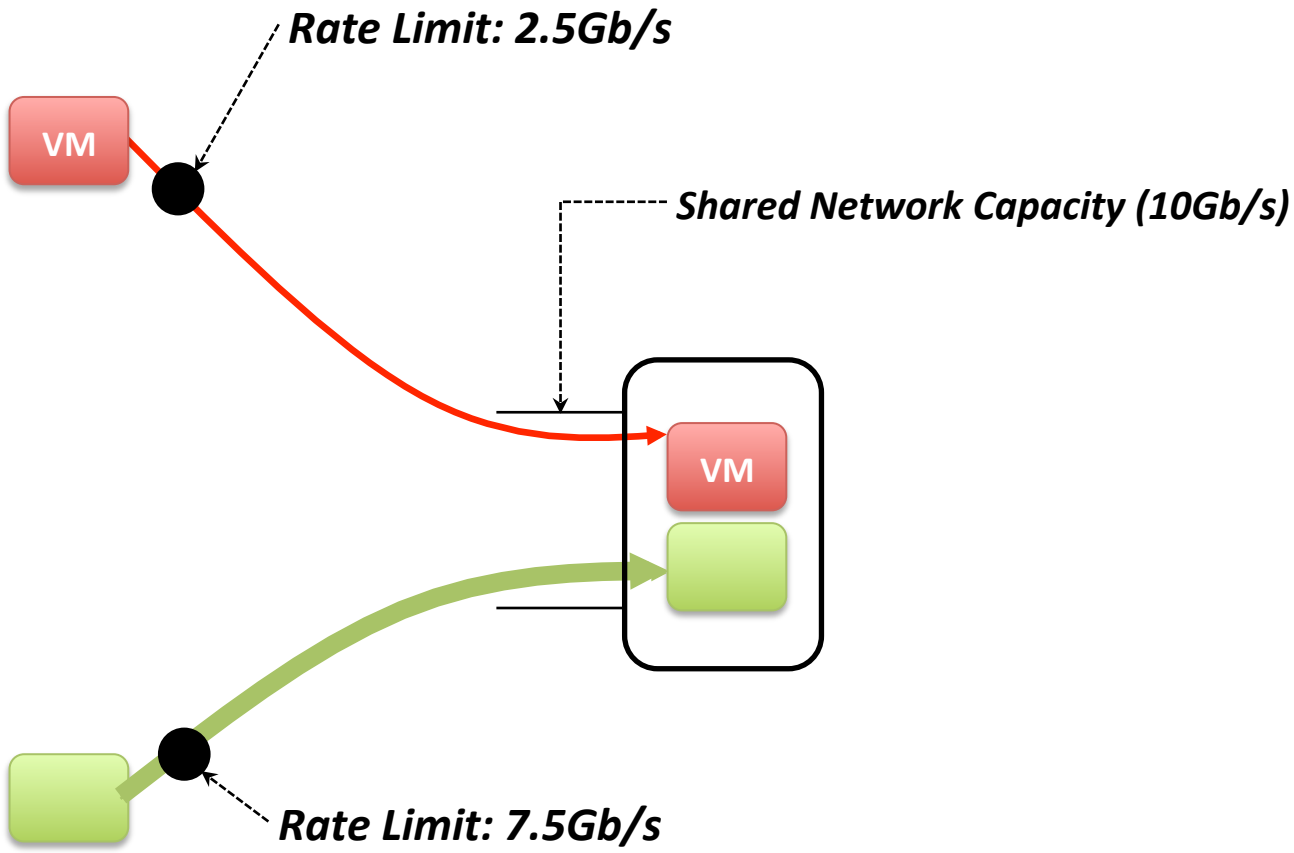
Bursty UDP session:
2.5Gb/s
ON: 5ms
OFF: 15ms



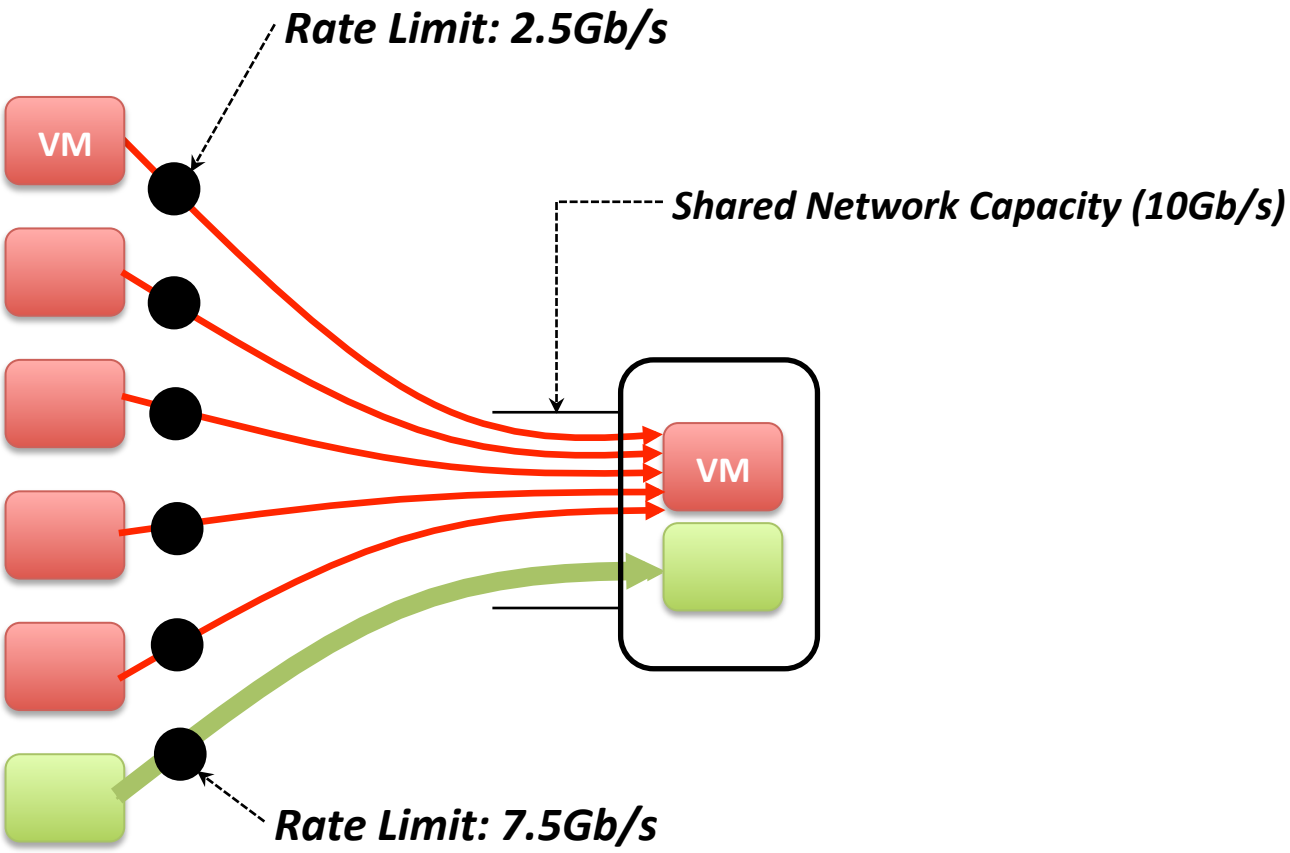
Strawman: Rate Limiting



Strawman: Rate Limiting



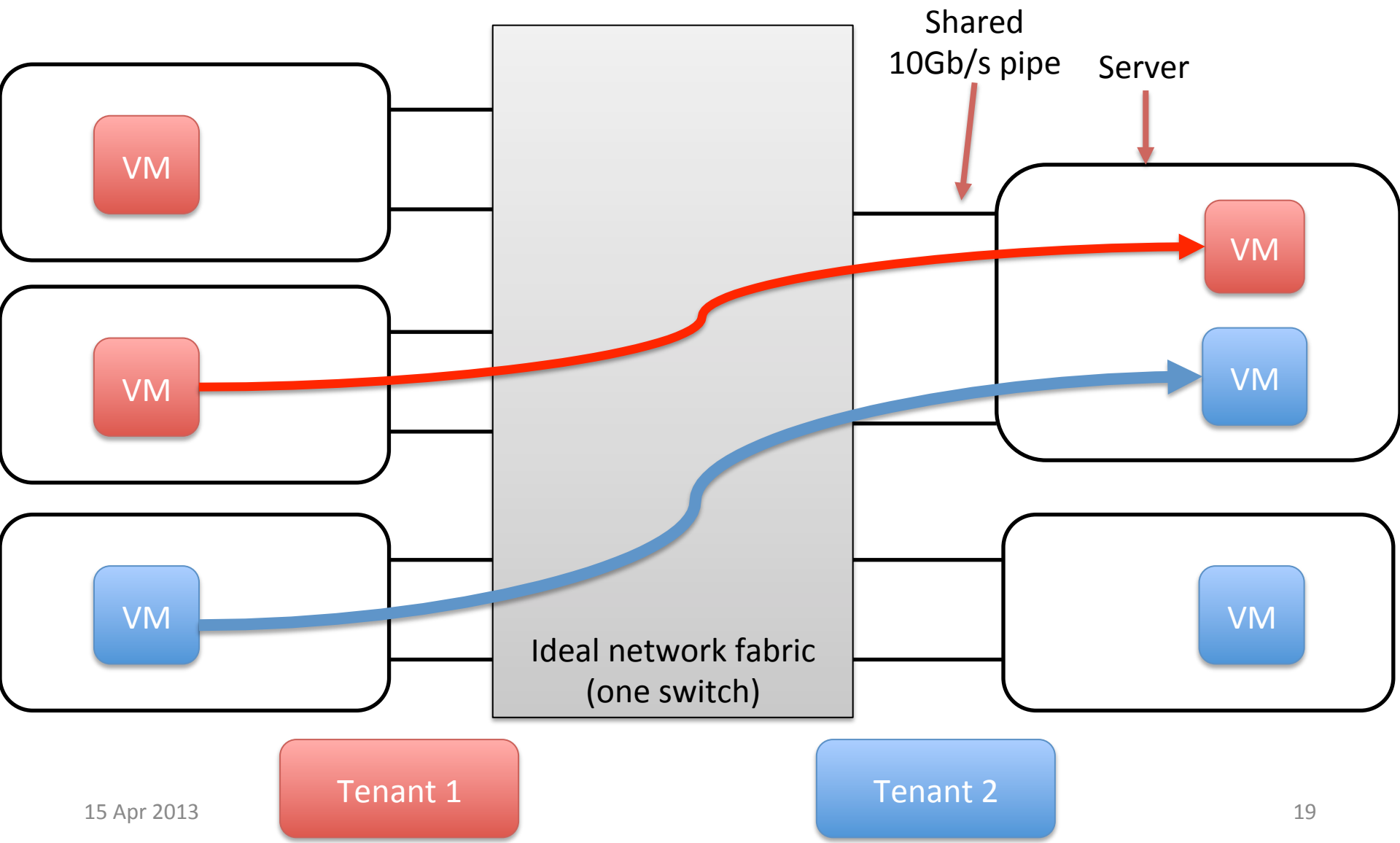
Strawman: Rate Limiting



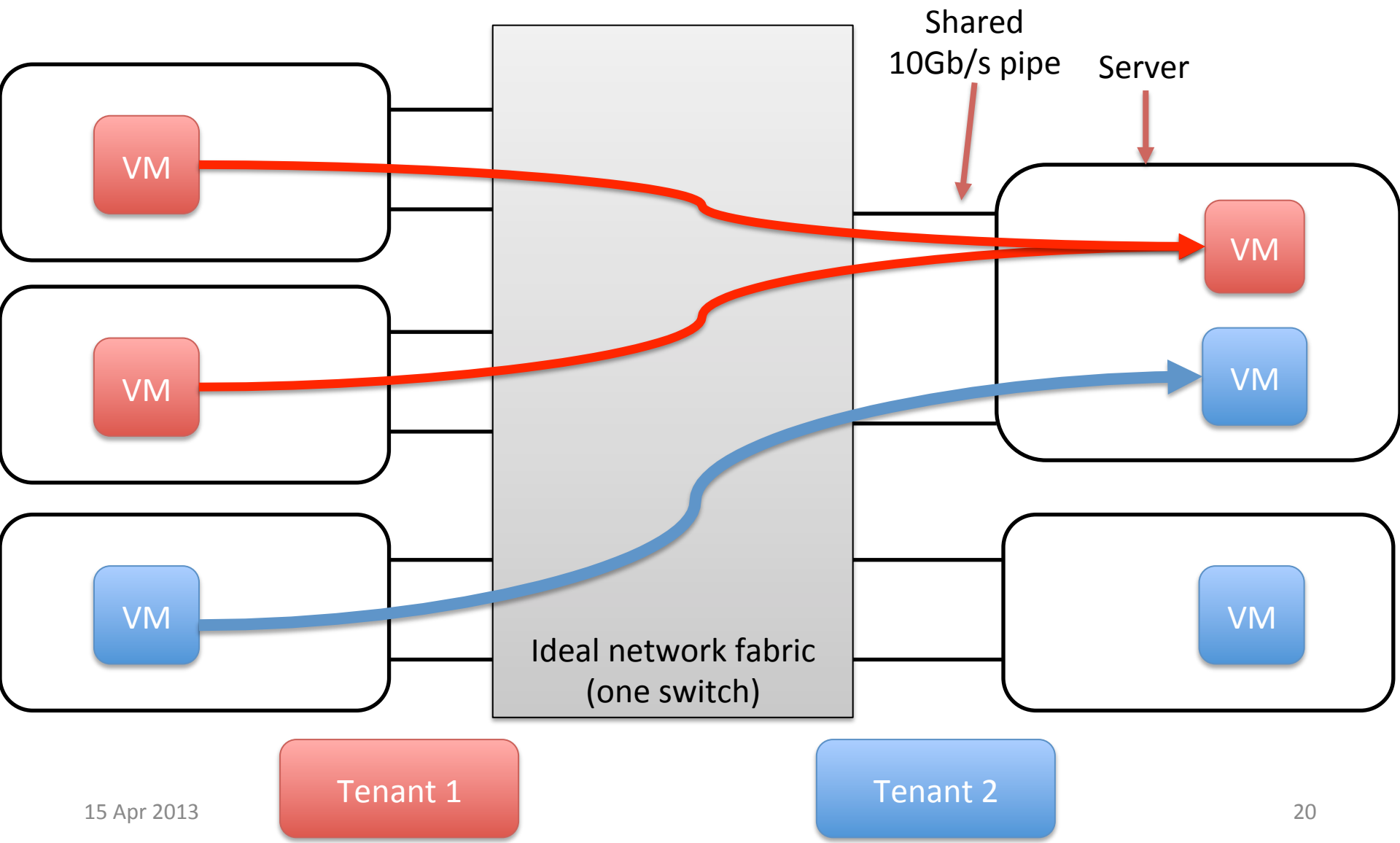
Recap…

- Bandwidth contention
  - Can occur in a few milliseconds!
  - Happens even if tenants don't talk to each other
  - Invisible on 5 minute logs!
- Cannot trust tenants at all
  - Easy to grab more bandwidth simply by blasting traffic (UDP)
- Naïve rate limiting not enough
  - VMs can gang up and blast traffic

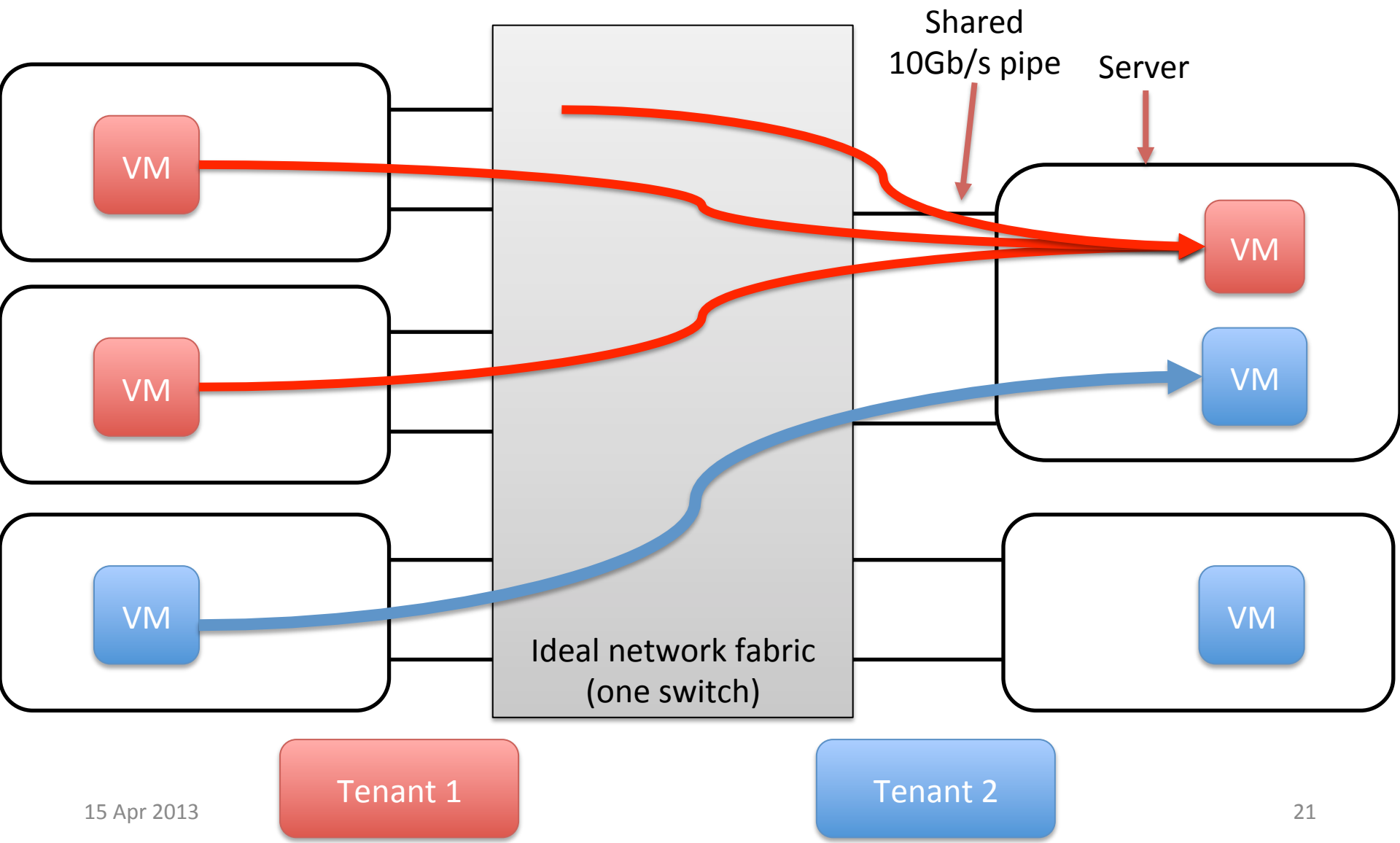
Where does Congestion Happen?



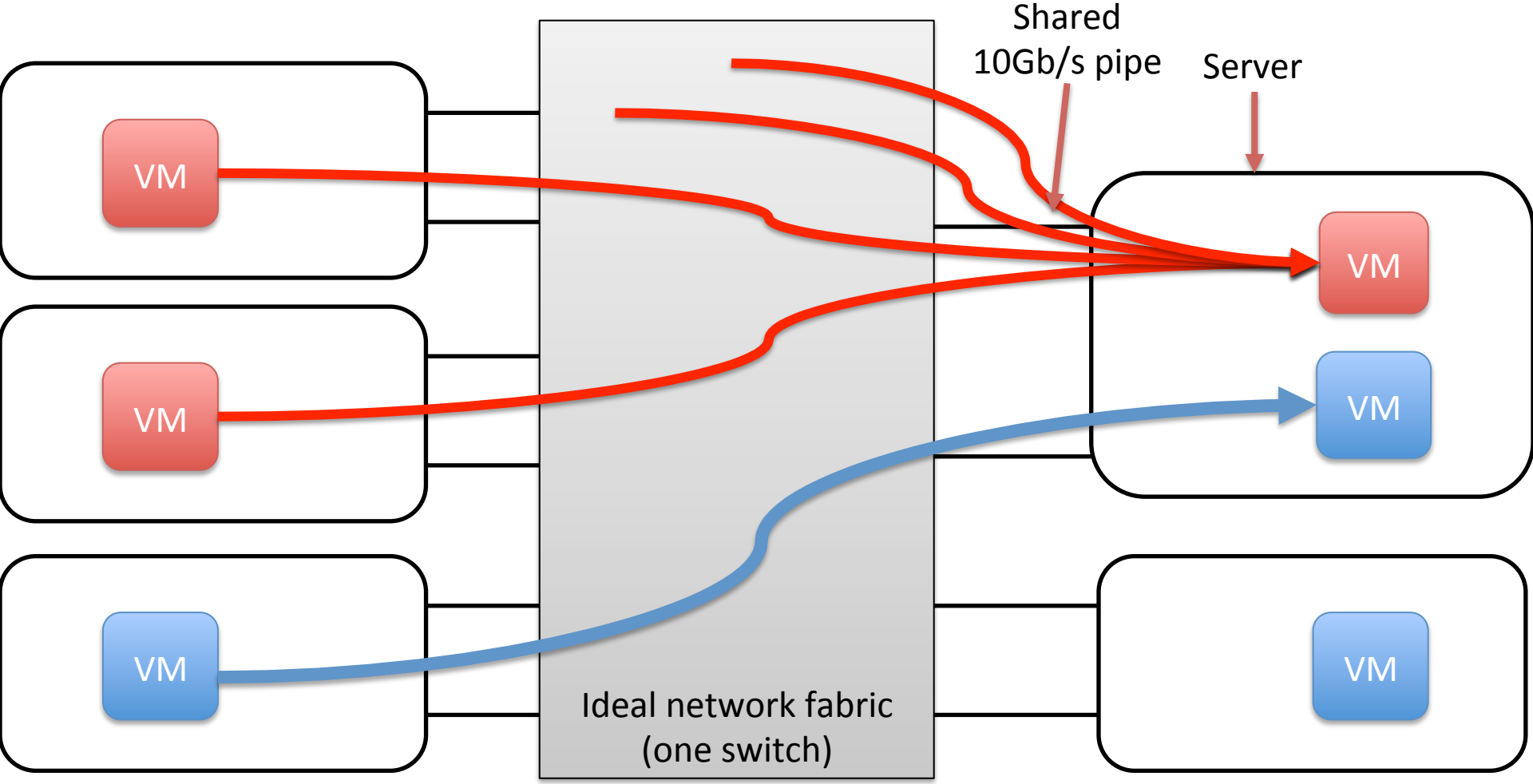
Where does Congestion Happen?



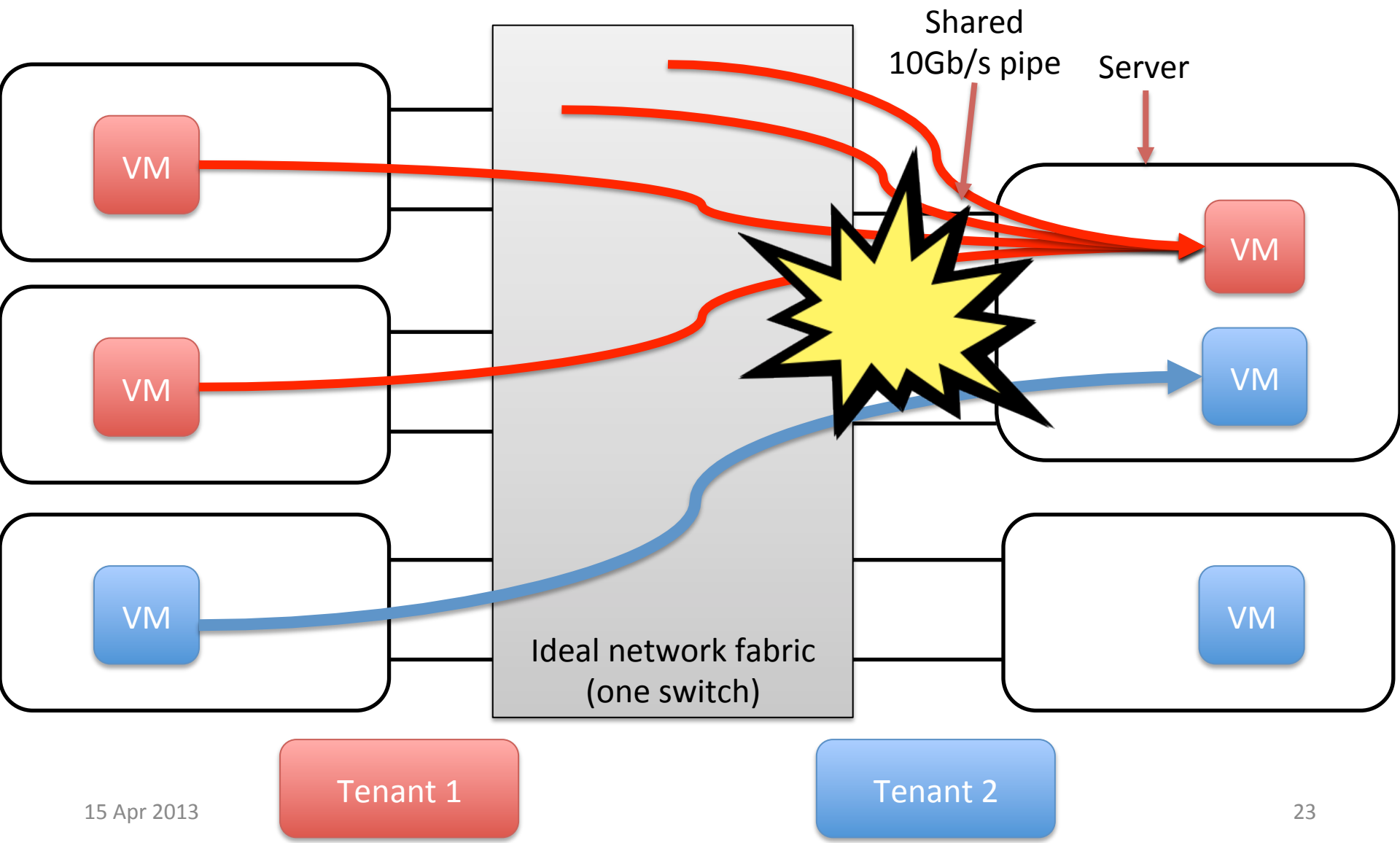
Where does Congestion Happen?



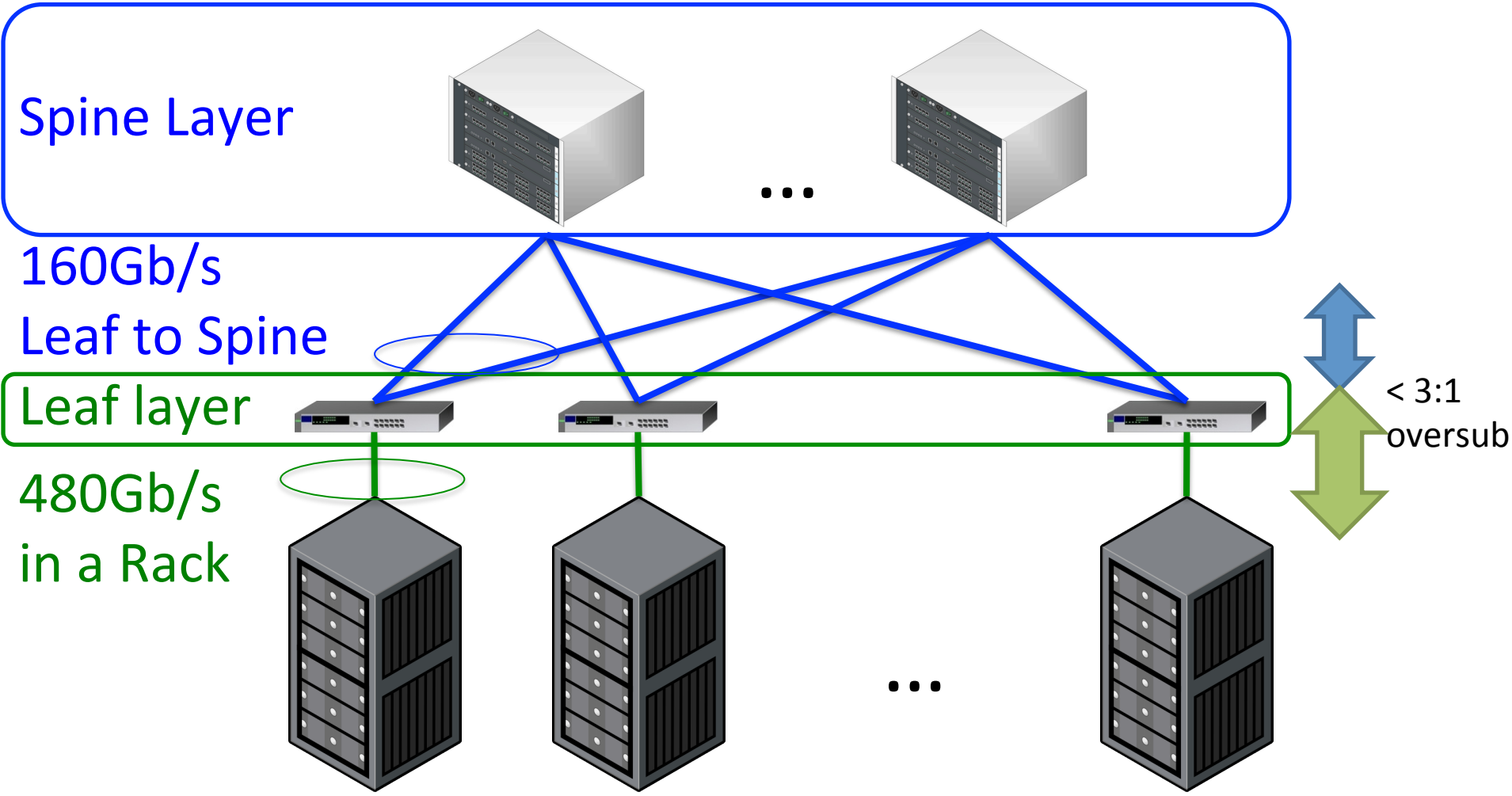
Where does Congestion Happen?

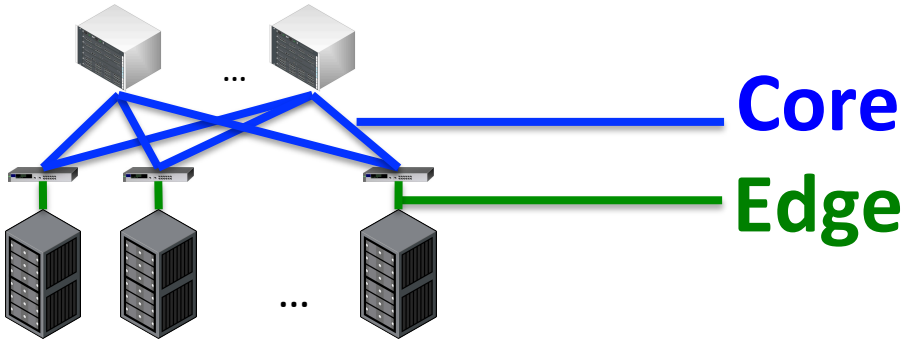


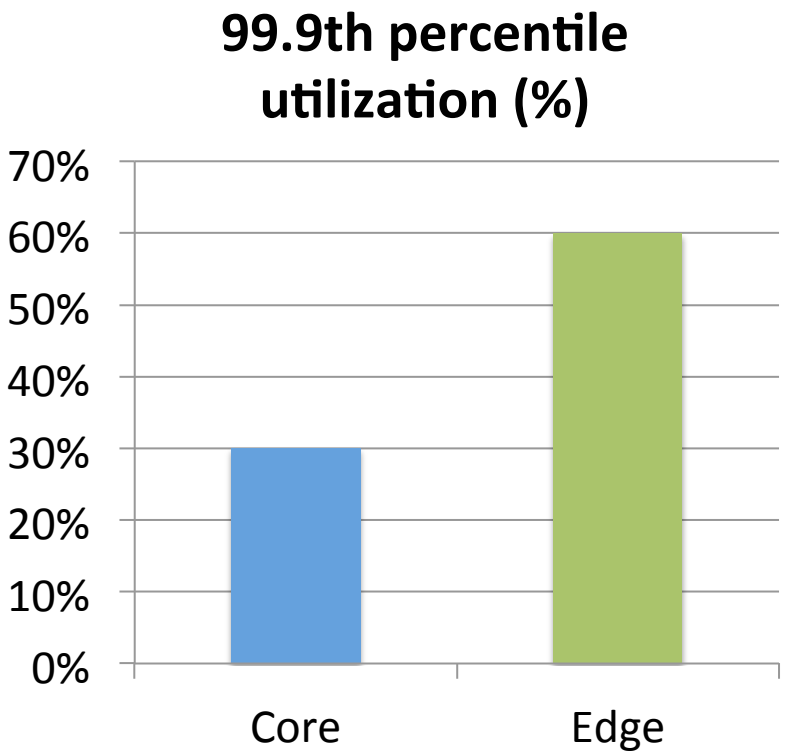
Where does Congestion Happen?



Congestion Study on Windows Azure







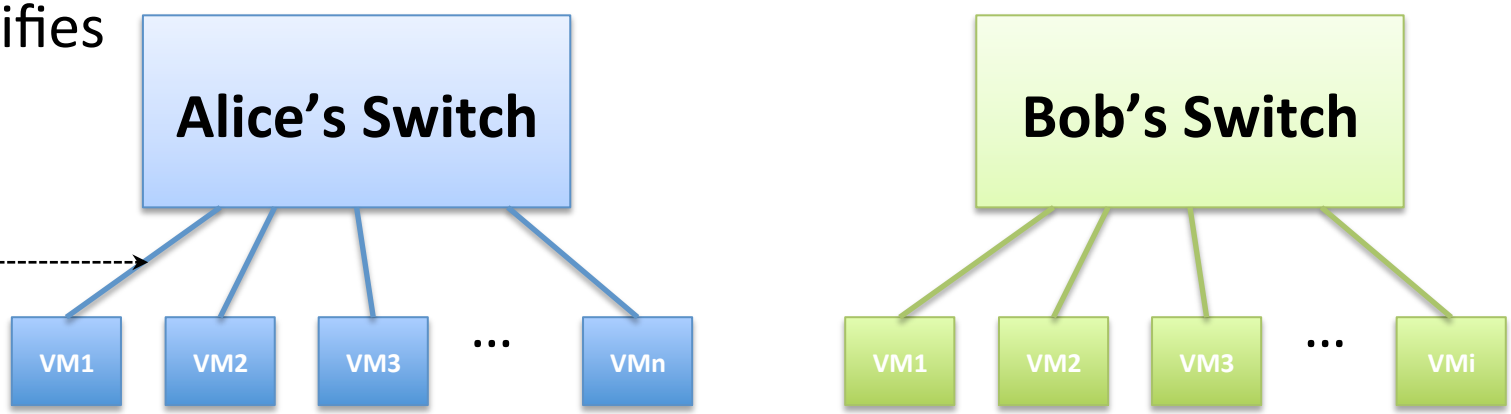
Timescales: over 2 weeks, 99.9th pcile = several minutes

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

16 of 17 clusters: 0 drops in the Core.

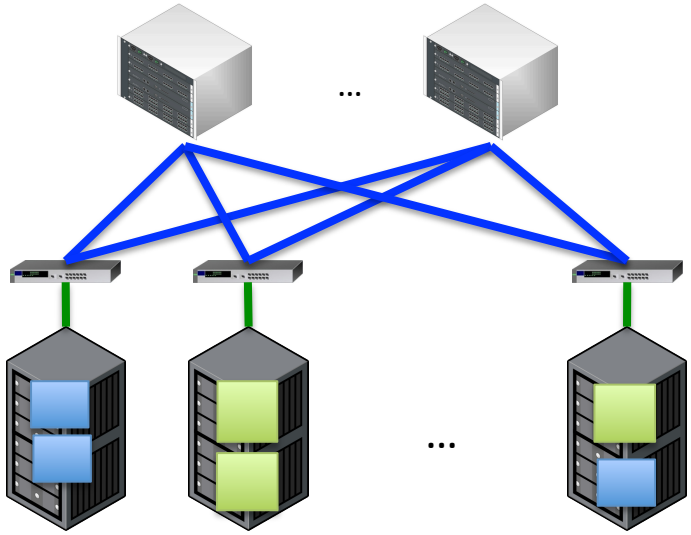
EyeQ's Goal: Rate Guarantees for VMs in the Cloud

Customer specifies capacity of the virtual NIC.

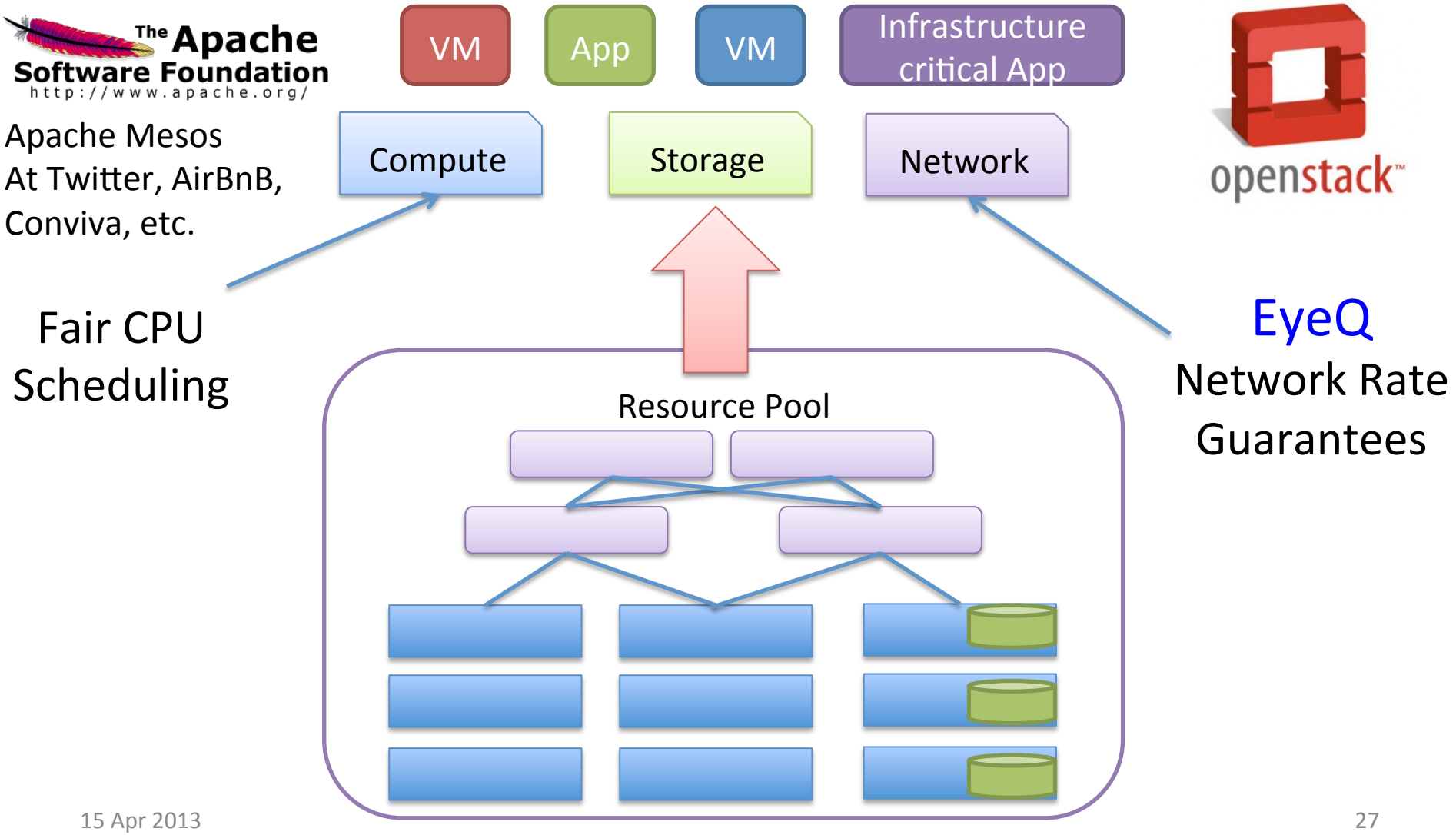


Provider: assures near dedicated performance.

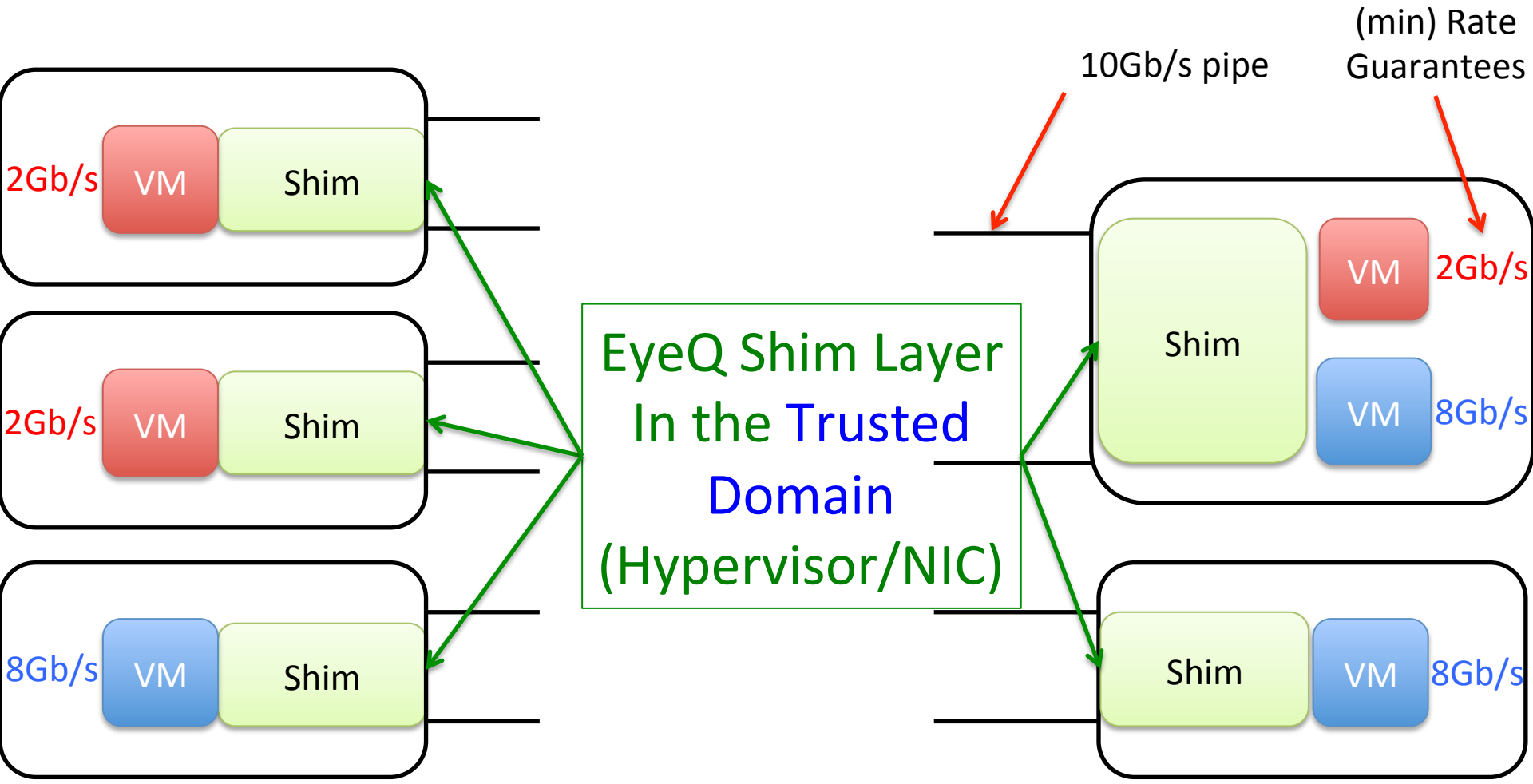
Rate guarantees => Performance isolation



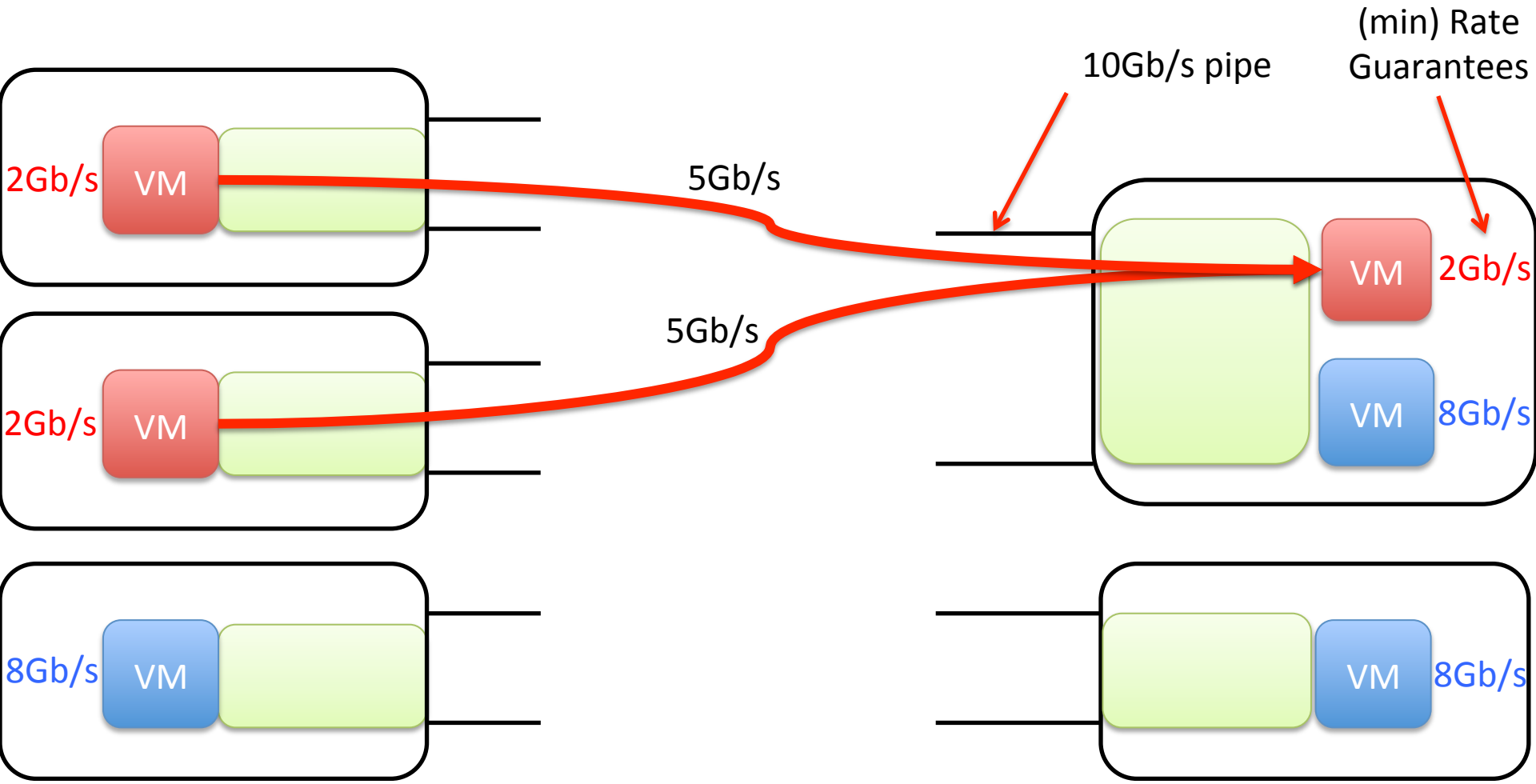
The Big Picture: Resource Management



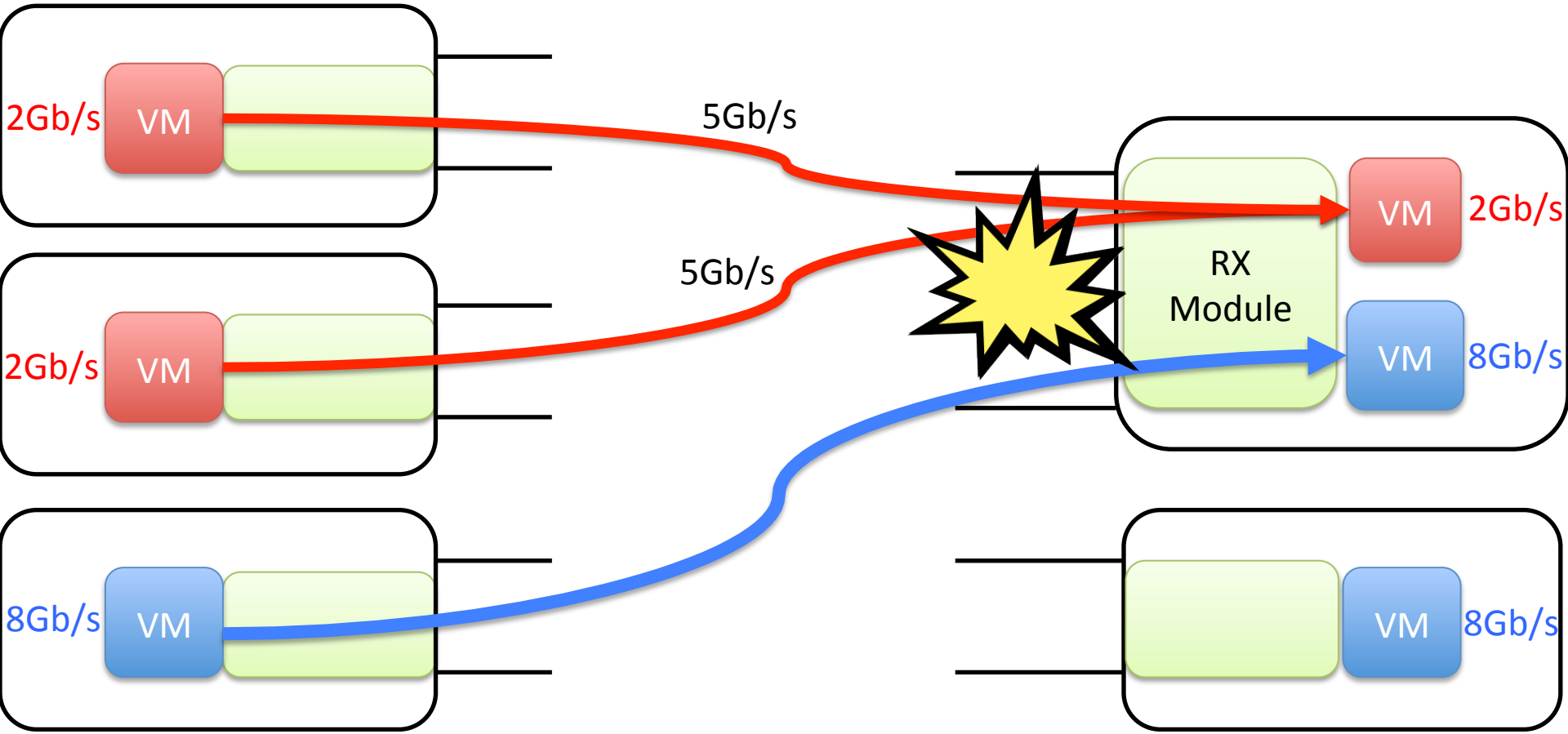
Decentralized Scheduling



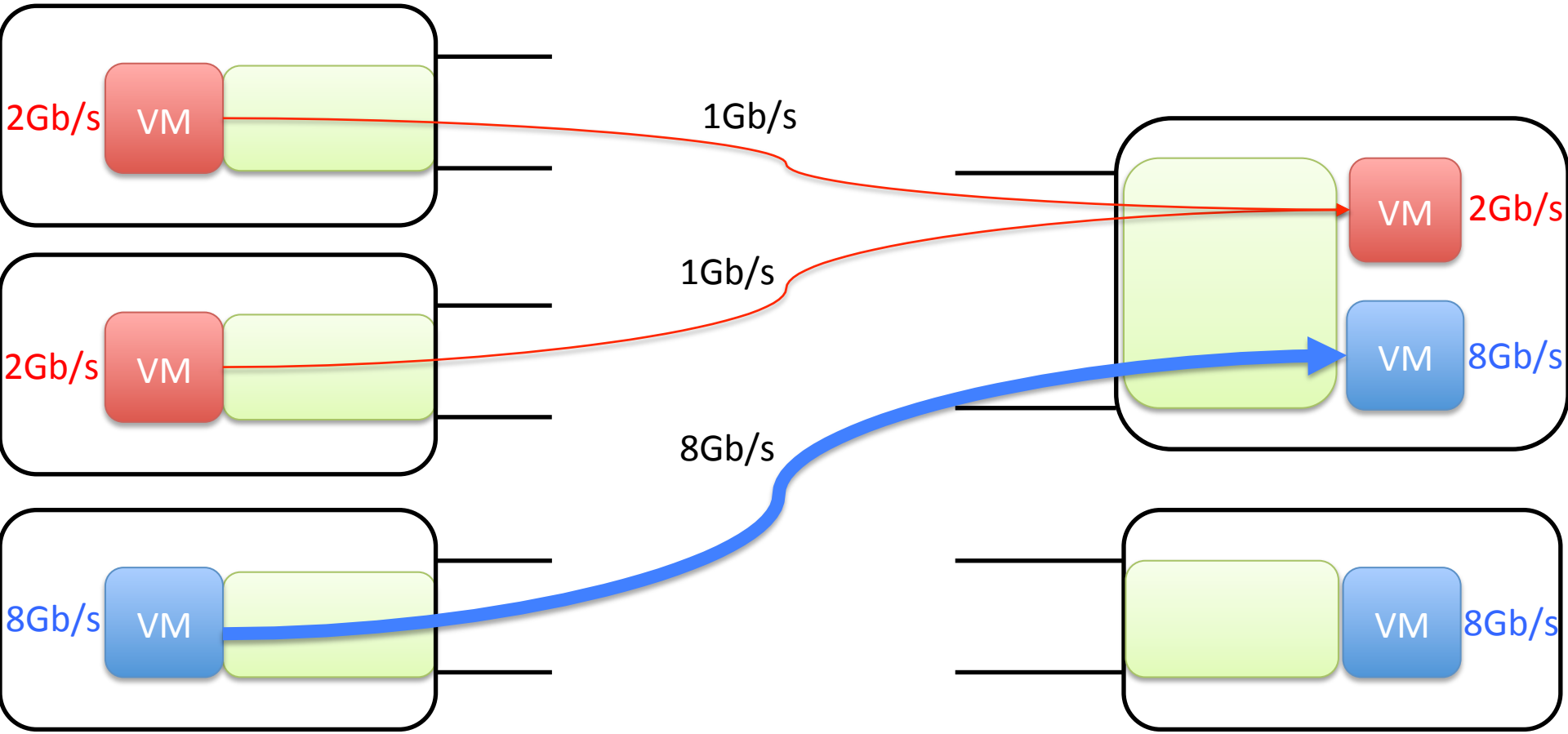
Decentralized Scheduling



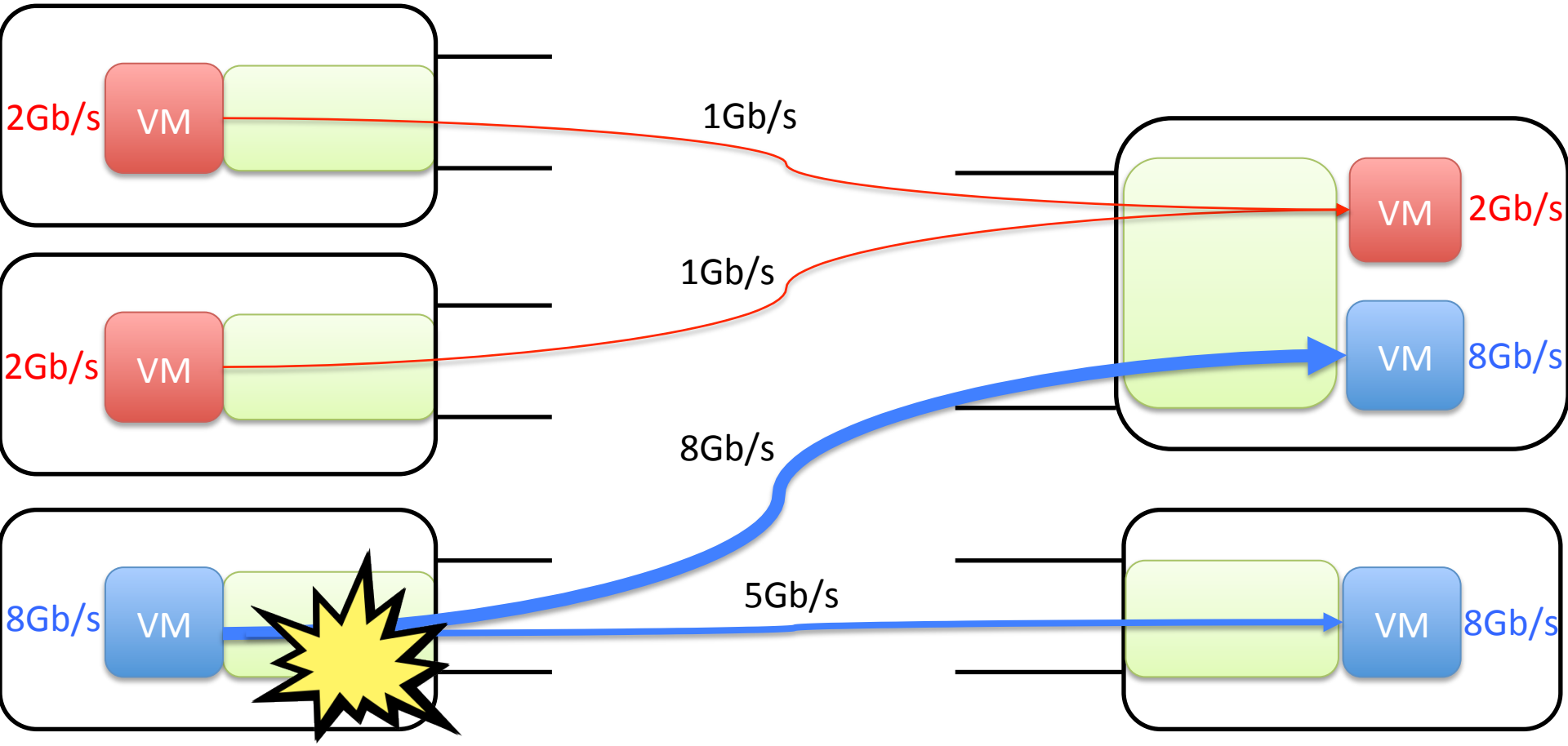
Decentralized Scheduling



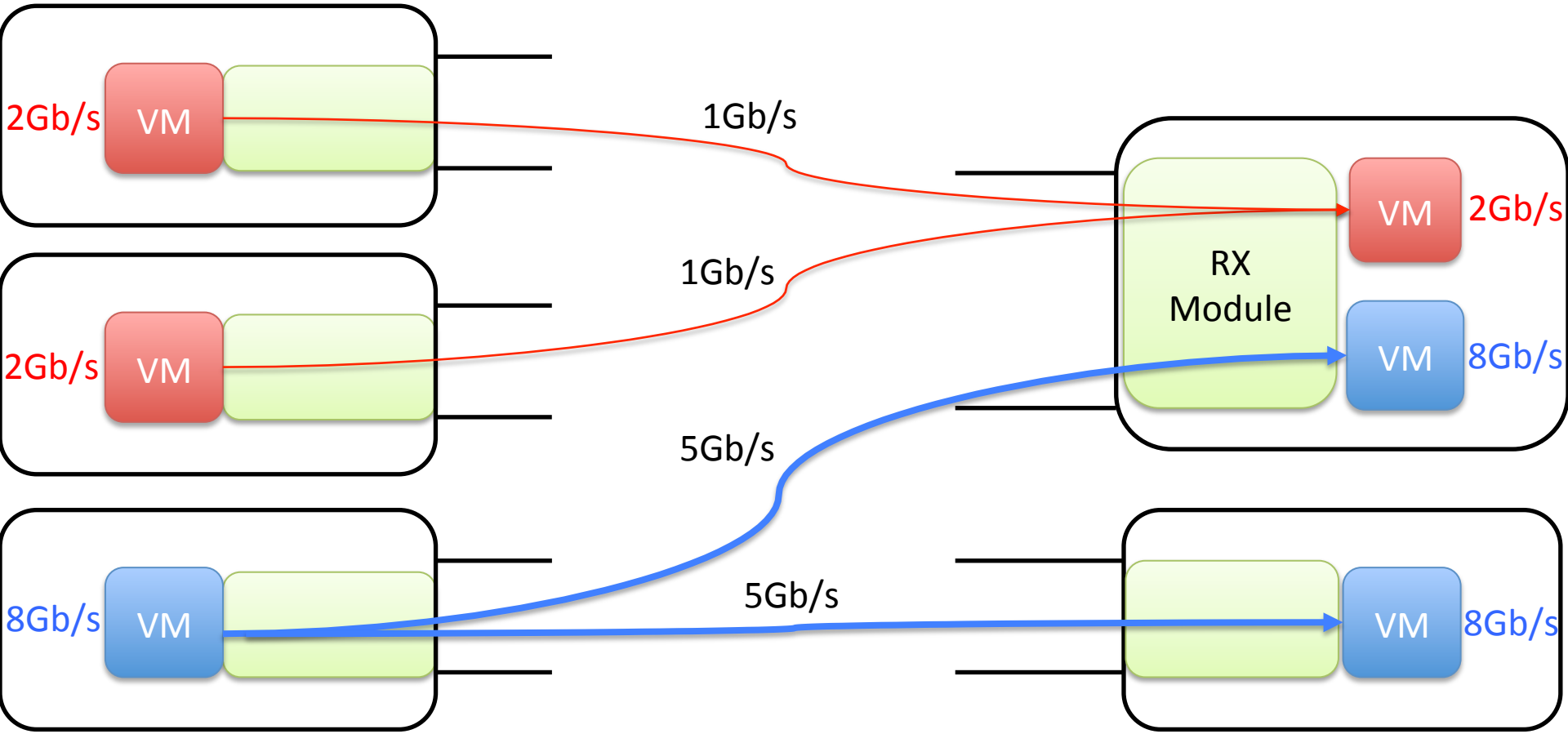
Decentralized Scheduling



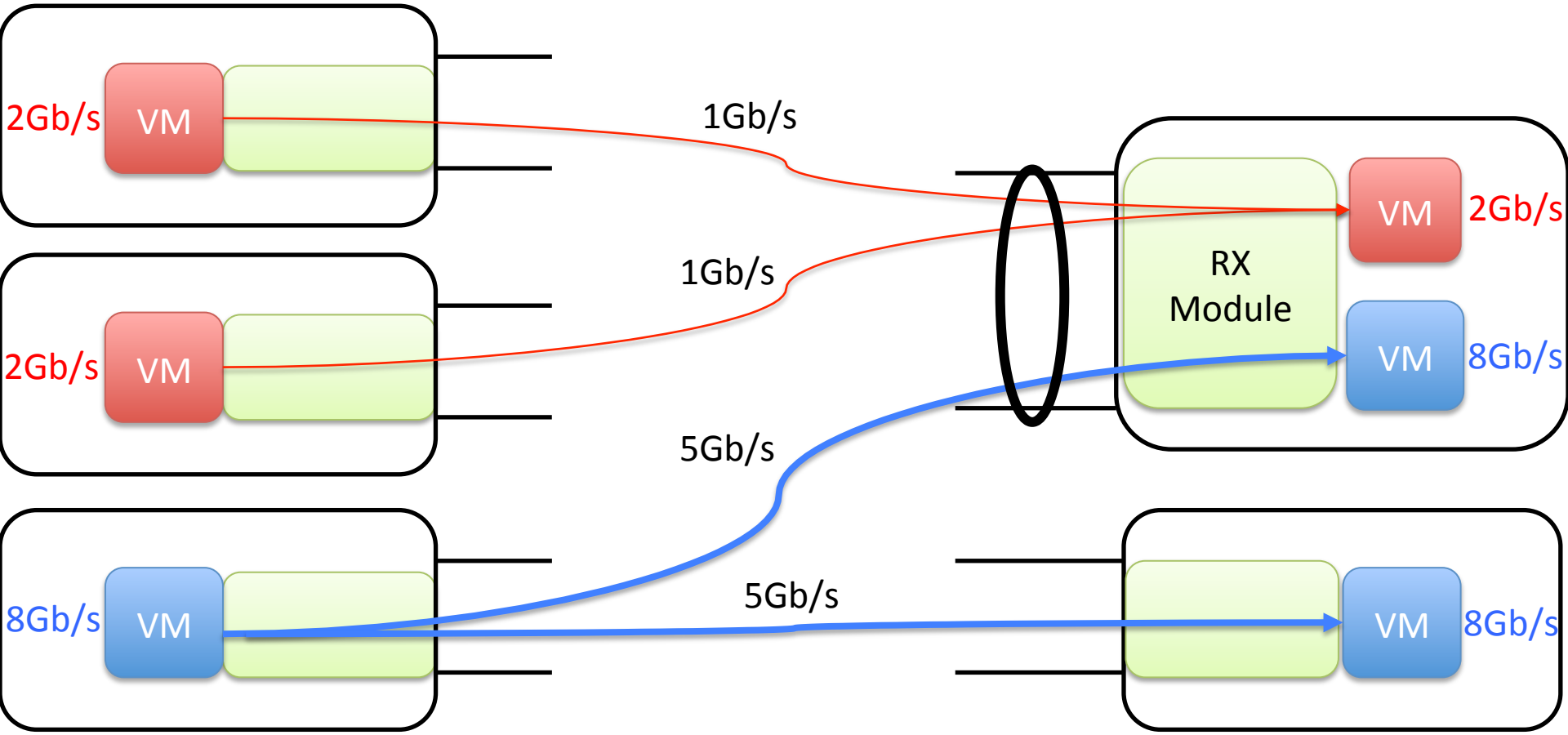
Decentralized Scheduling



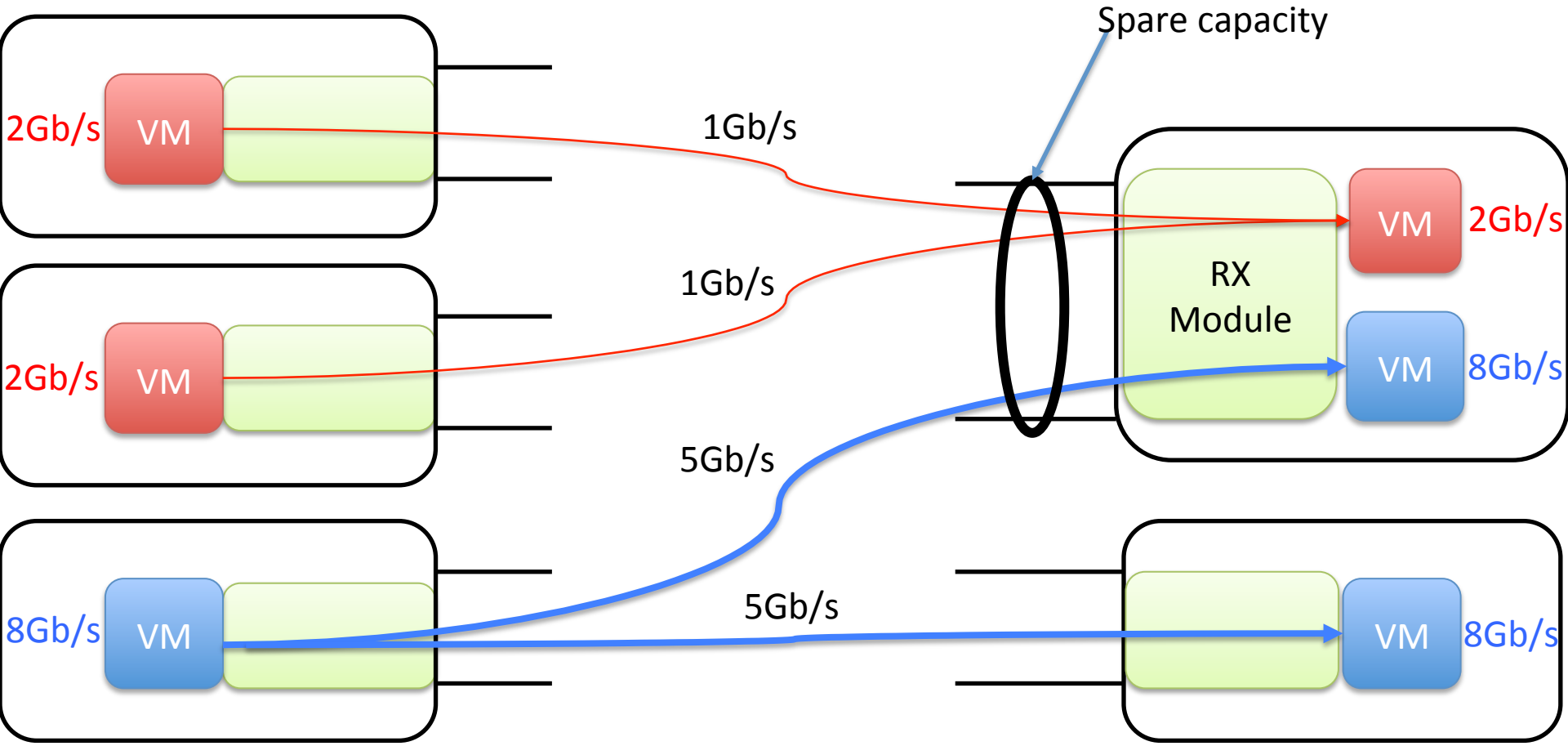
Work Conserving Allocations



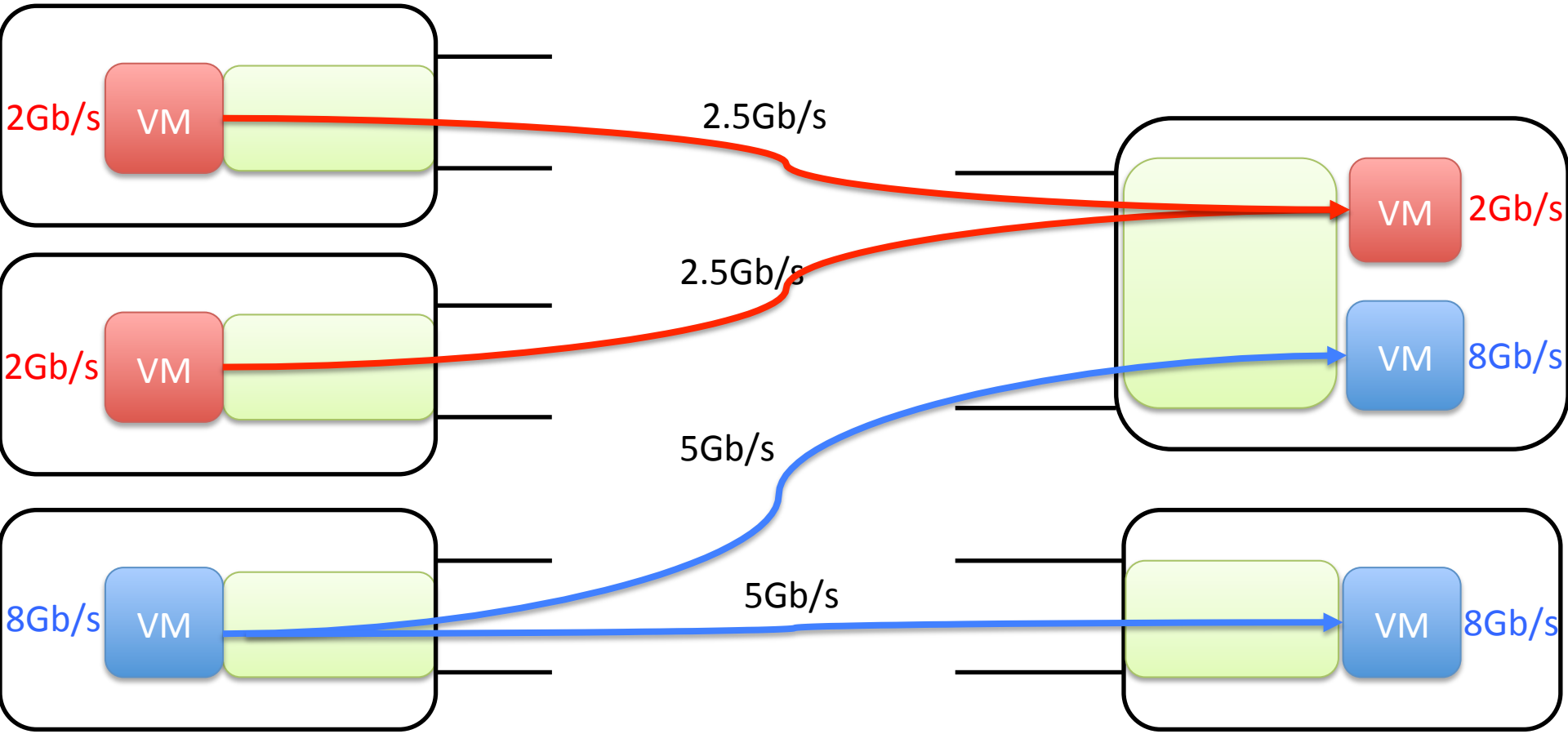
Work Conserving Allocations



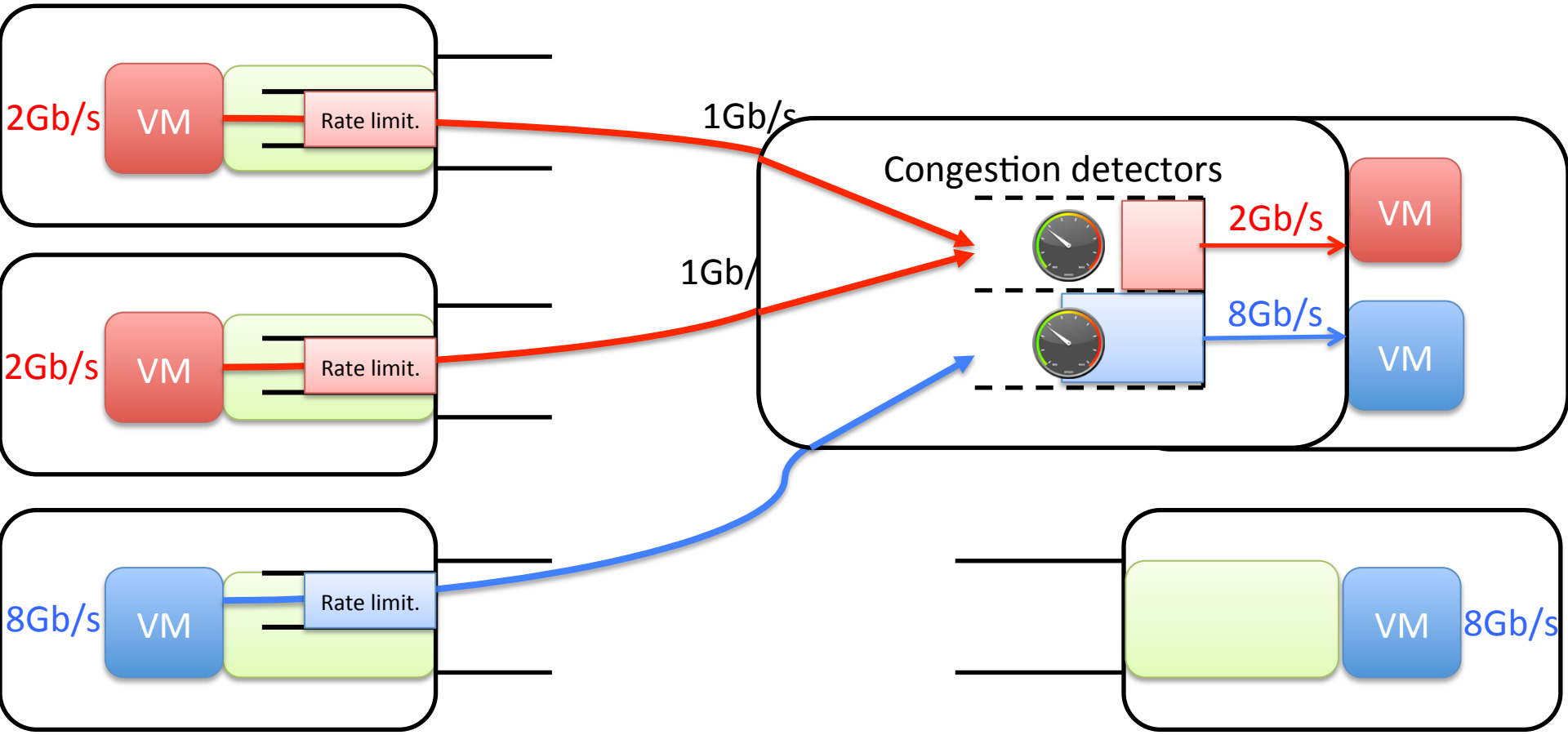
Work Conserving Allocations



Work Conserving Allocations

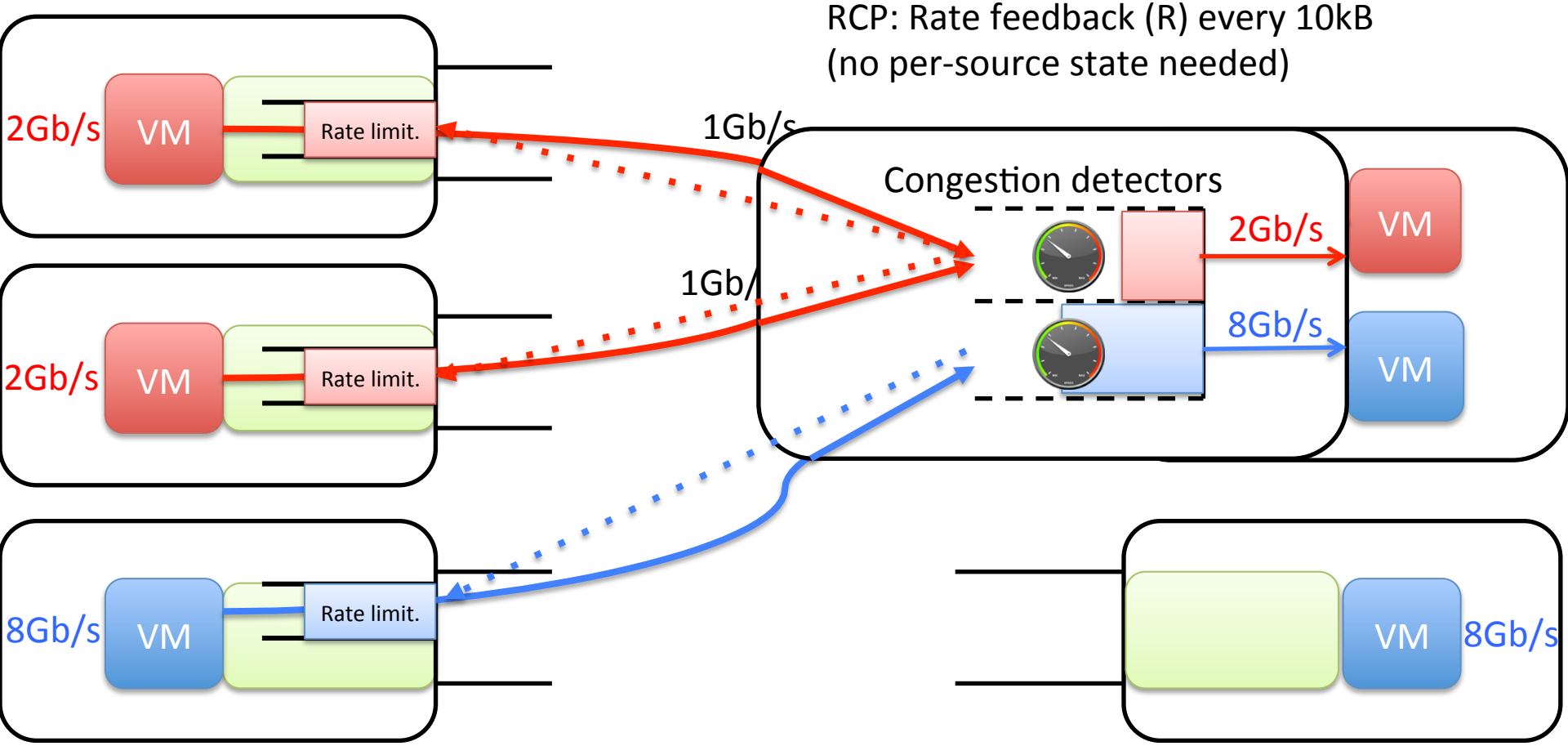


Transmit/Receive Modules



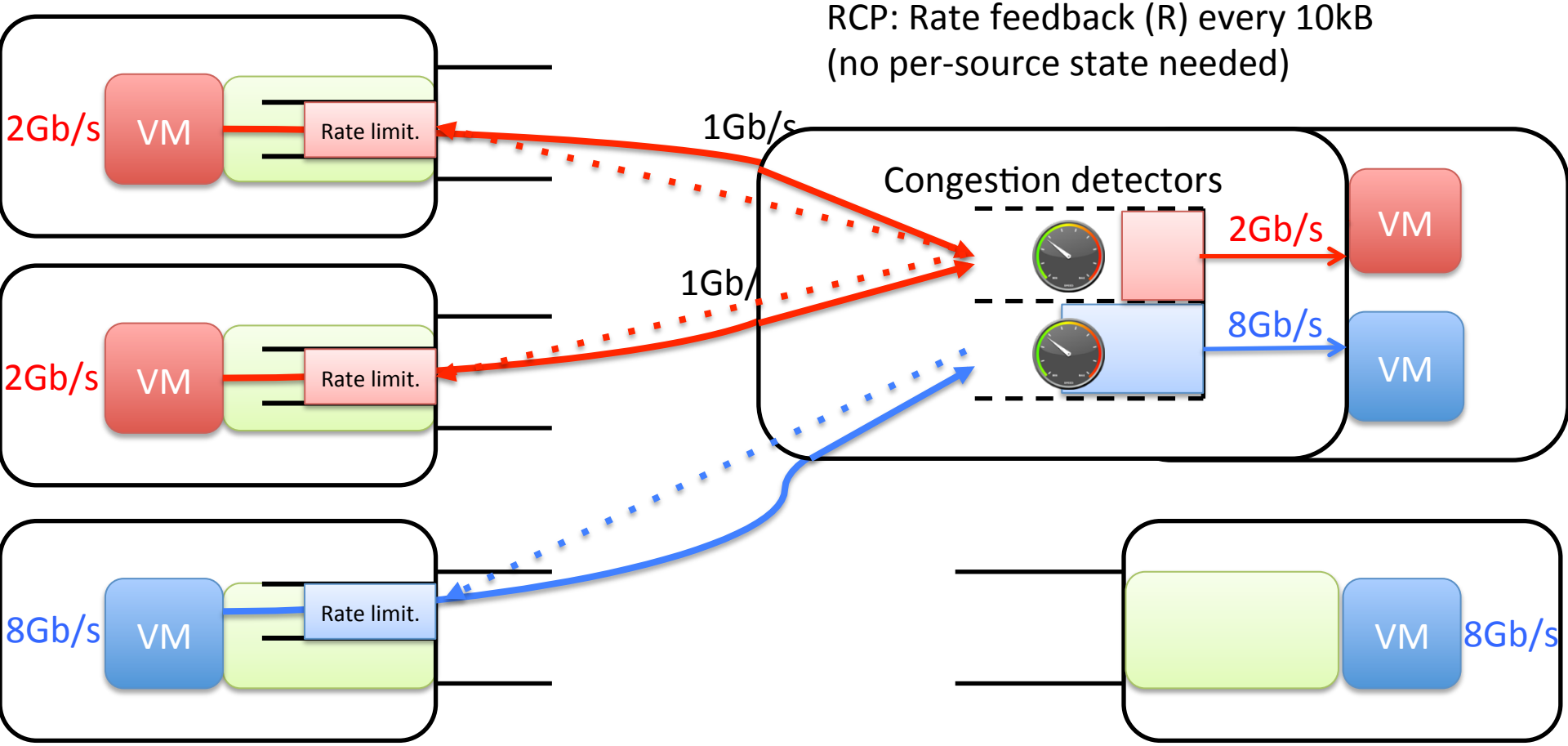
Per-destination rate limiters:
only if dest. is congested… bypass otherwise

Transmit/Receive Modules



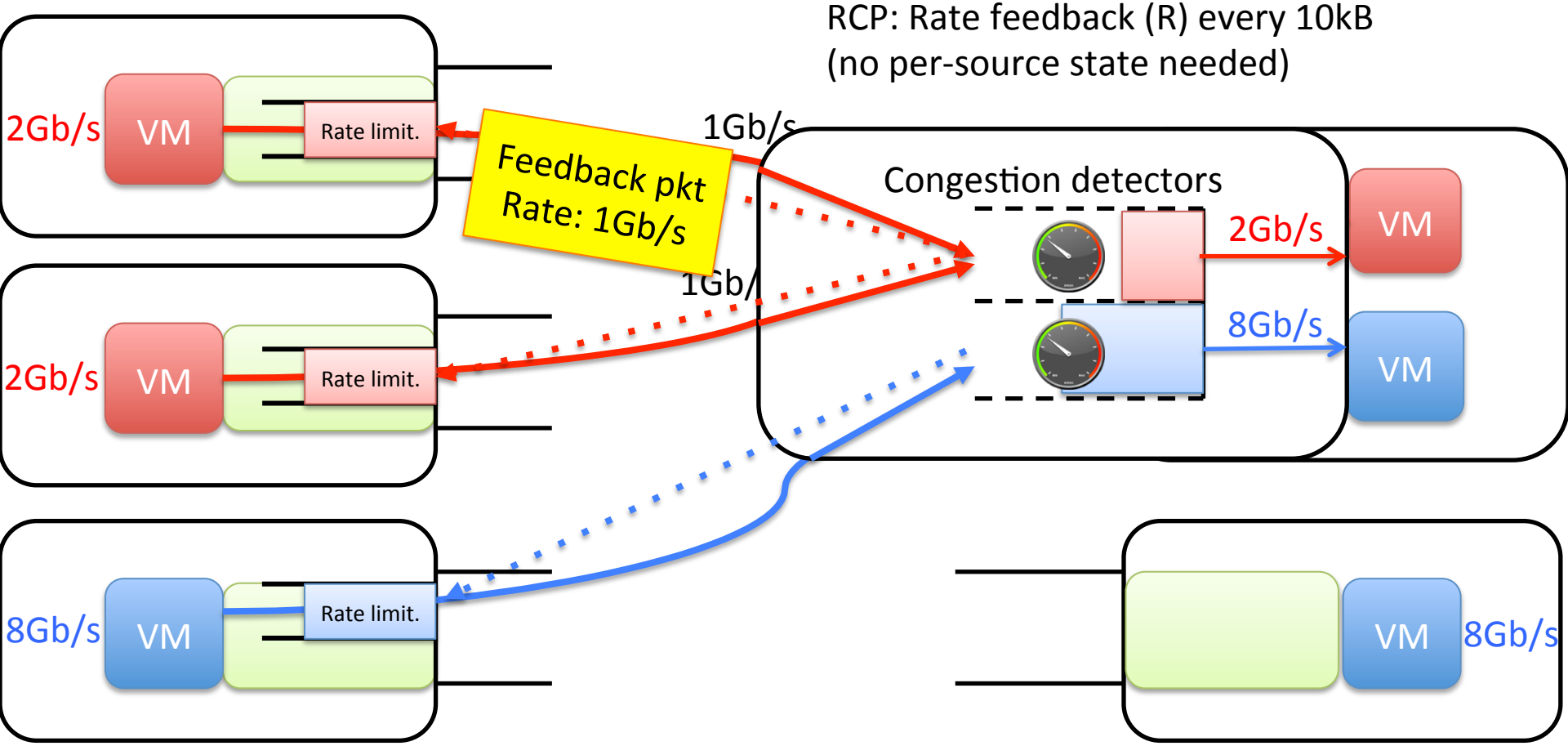
Per-destination rate limiters:
only if dest. is congested… bypass otherwise

Transmit/Receive Modules



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

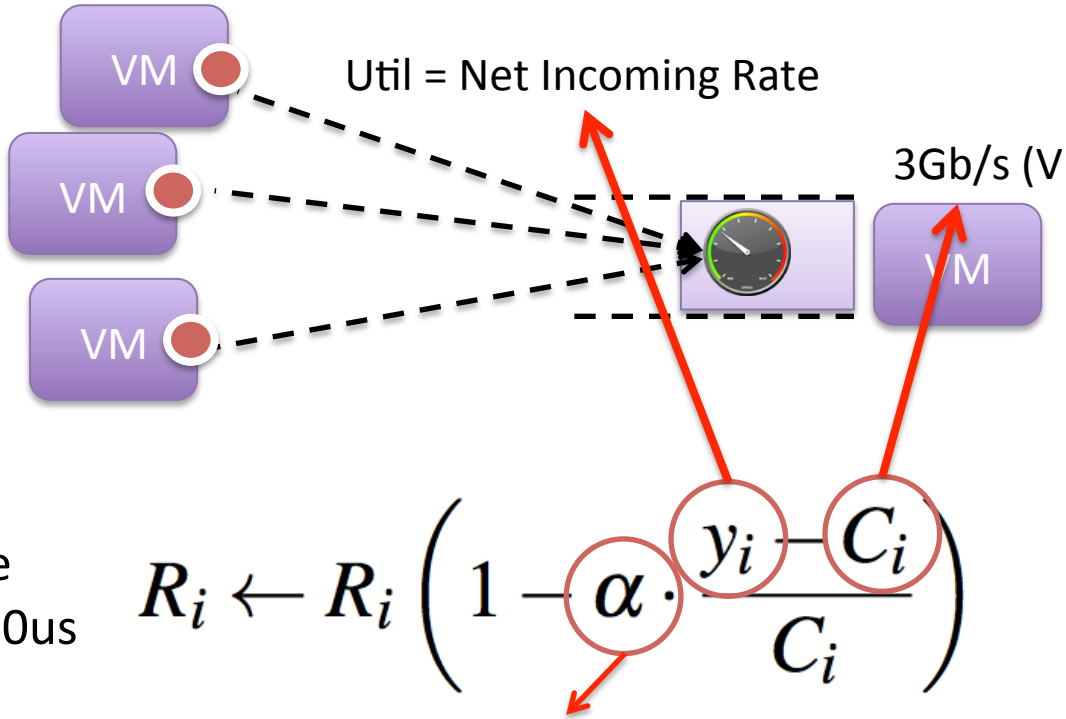
Transmit/Receive Modules



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

Determining Rate

Determine **one rate Ri** so that utilisation matches allowed limits

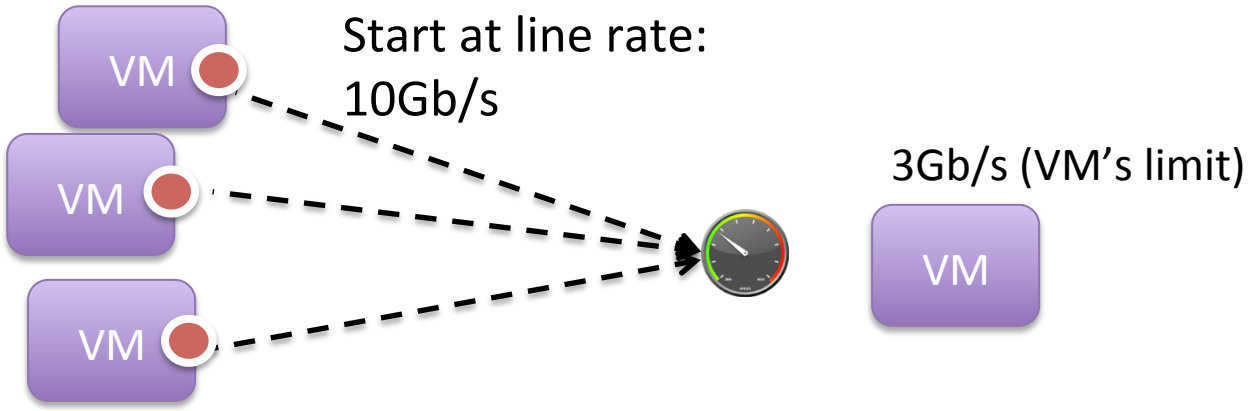


Recompute
Ri every 200us

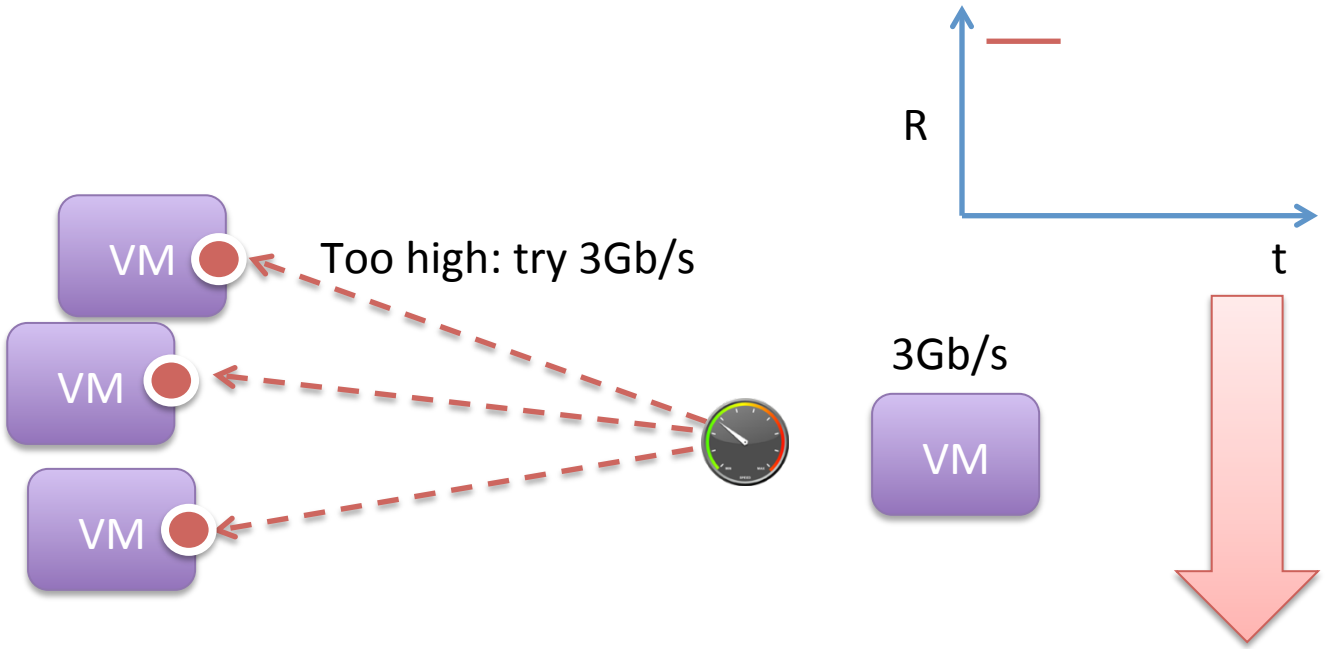
$$R_i \leftarrow R_i \left(1 - \alpha \cdot \frac{y_i - C_i}{C_i}\right)$$

Aggressiveness parameter. Set to 0.5

Determining Rate

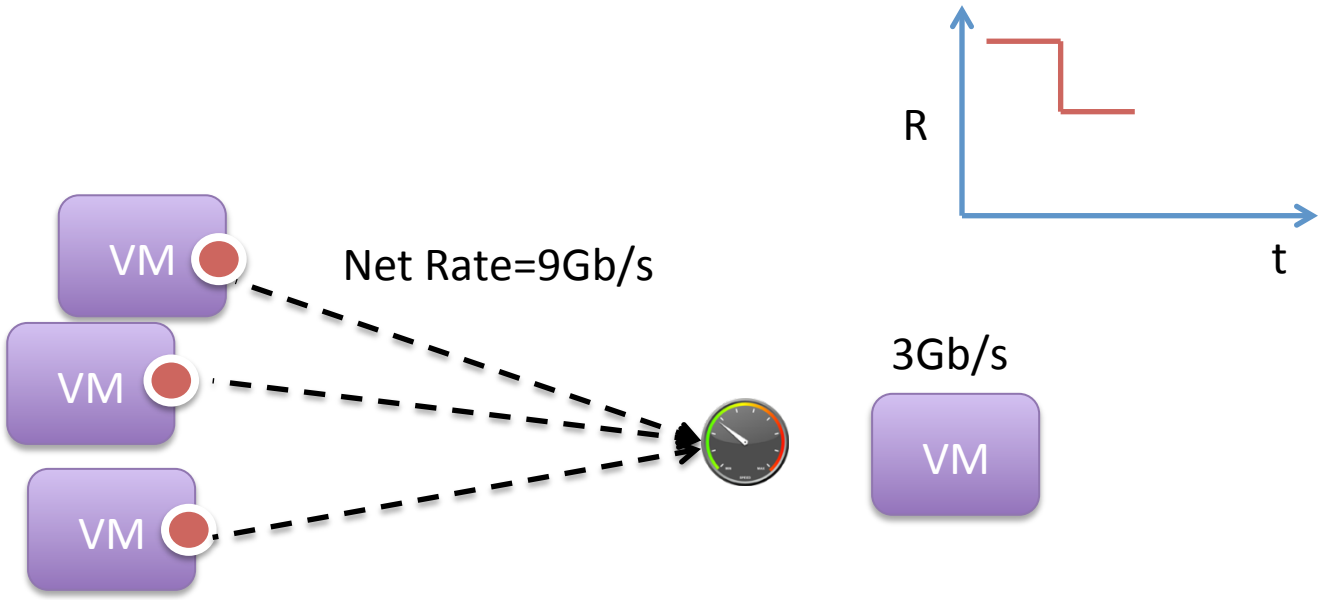


Determining Rate (distributed guessing)

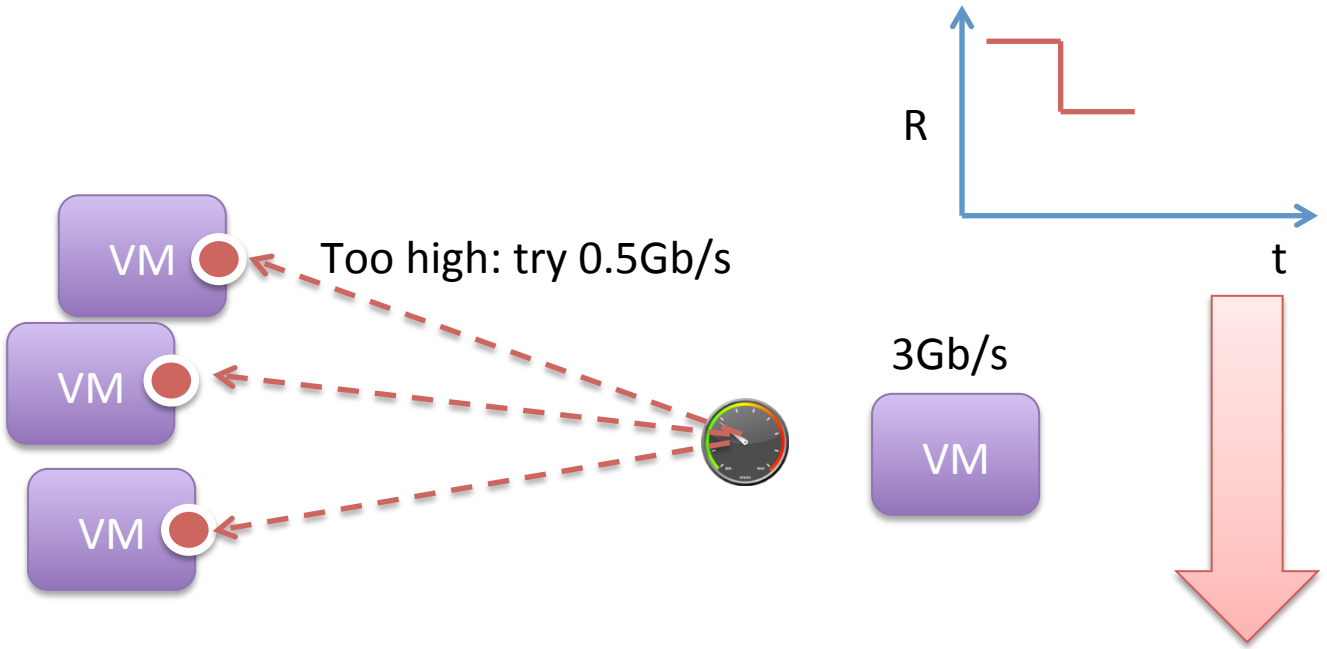


Tiny feedback packets sent to traffic sources

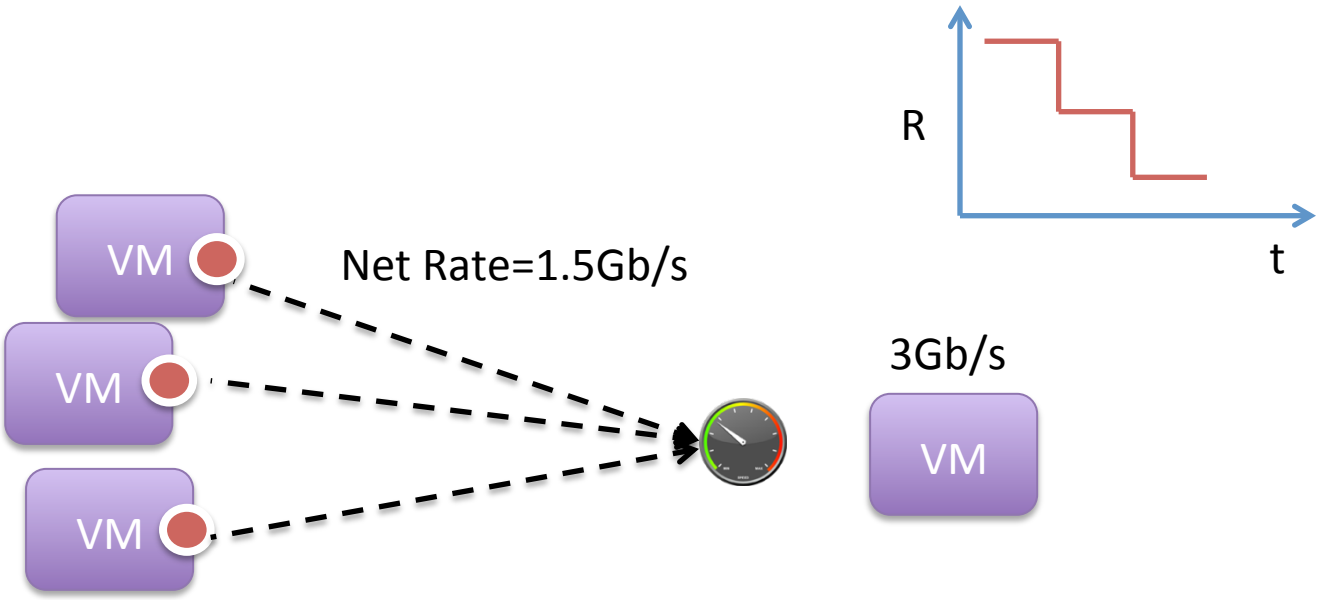
More guessing…



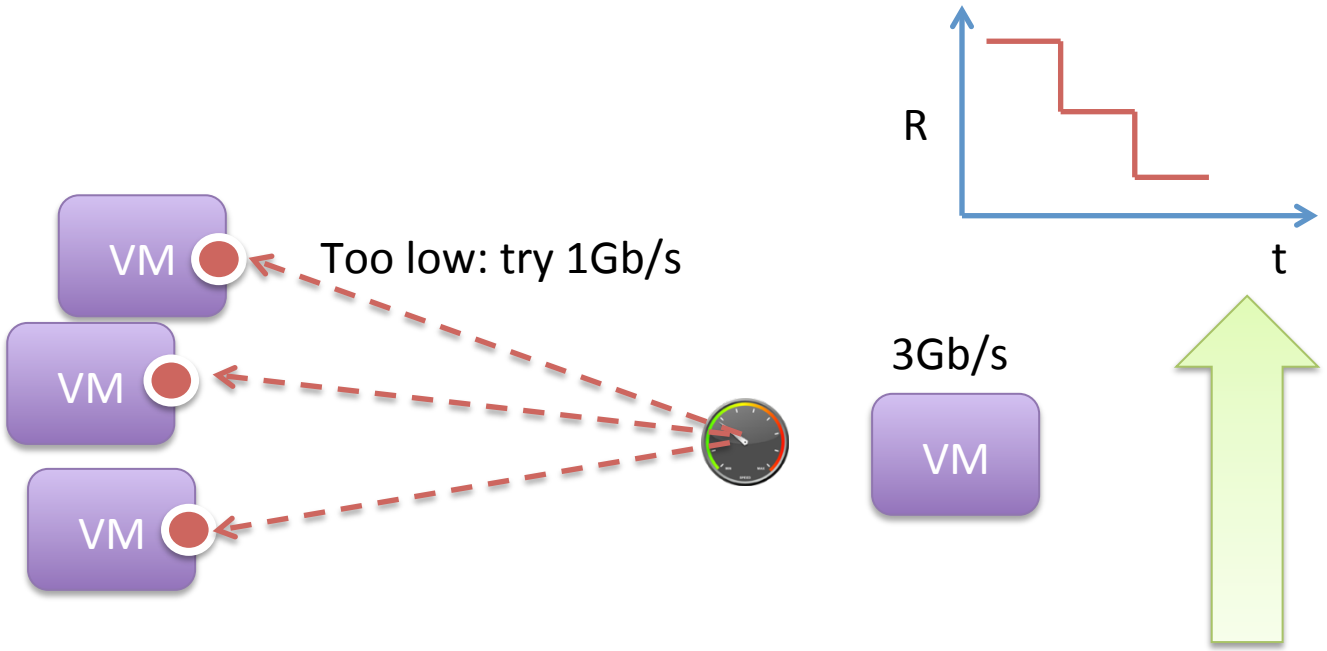
Continue guessing…



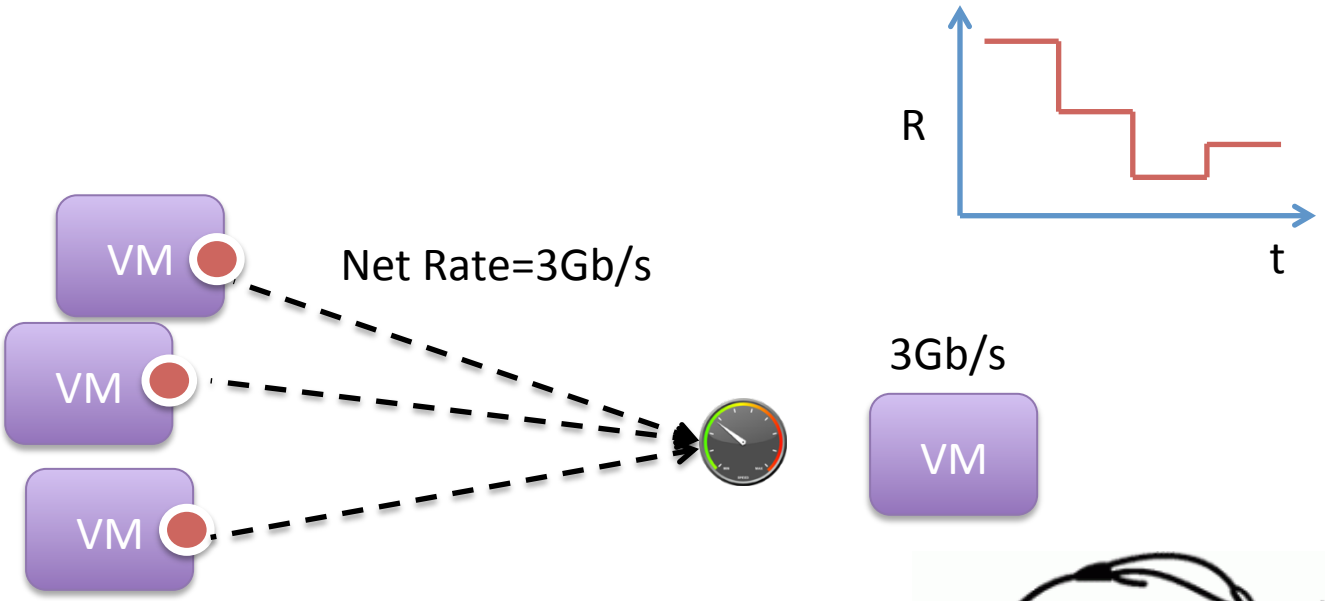
Oops…



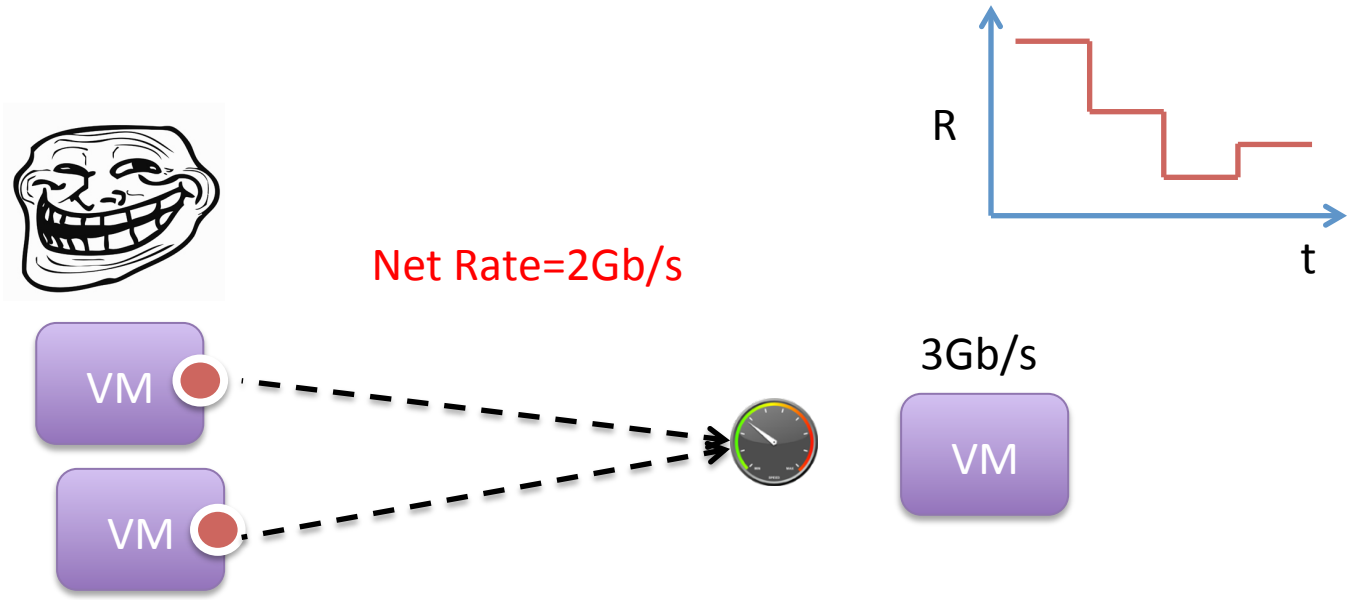
Almost there



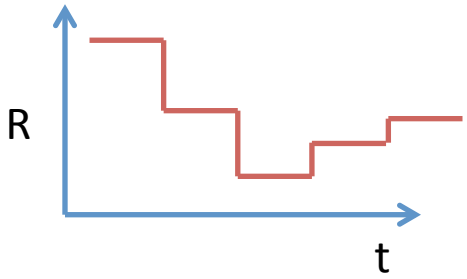
Fixed point

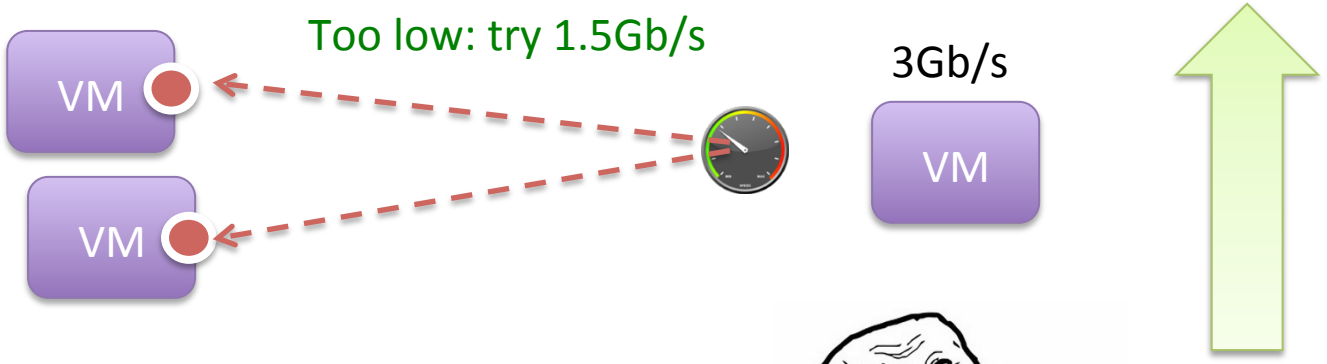


Continuous Scheduling



Continuous Scheduling





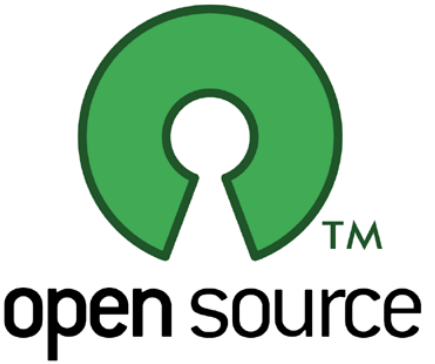
This happens every 200us.

Software Prototype

Linux Kernel Module (qdisc)

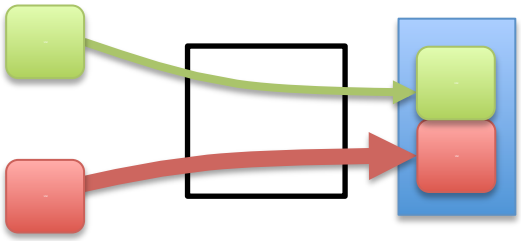
Windows Filter Driver (in VMSwitch)

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



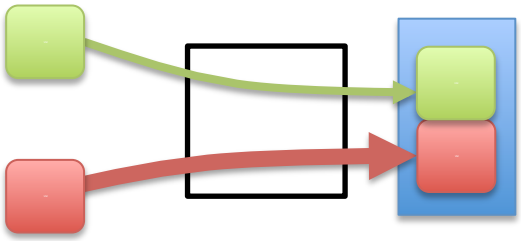
Linux Kernel Module is Open-Source

- Full system and documentation at http://jvimal.github.com/eyeq
- EyeQ's rate limiters more efficient than today's rate limiters in Linux/Windows



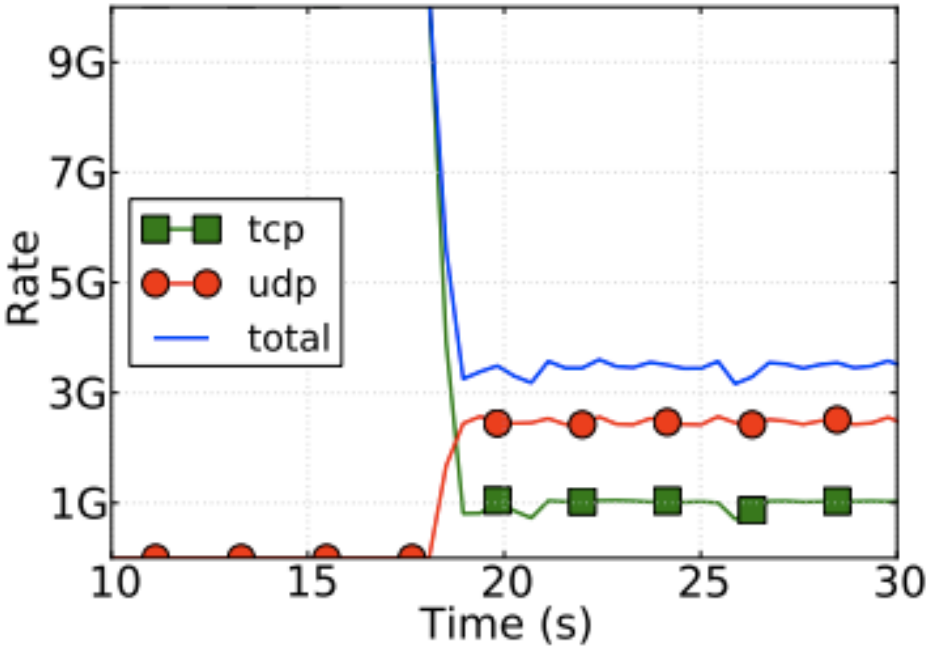
Does it work?

TCP: 6Gb/s
UDP: 3Gb/s

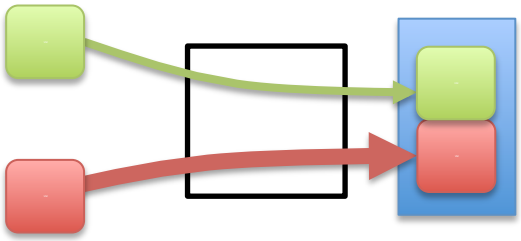


Does it work?

TCP: 6Gb/s
UDP: 3Gb/s

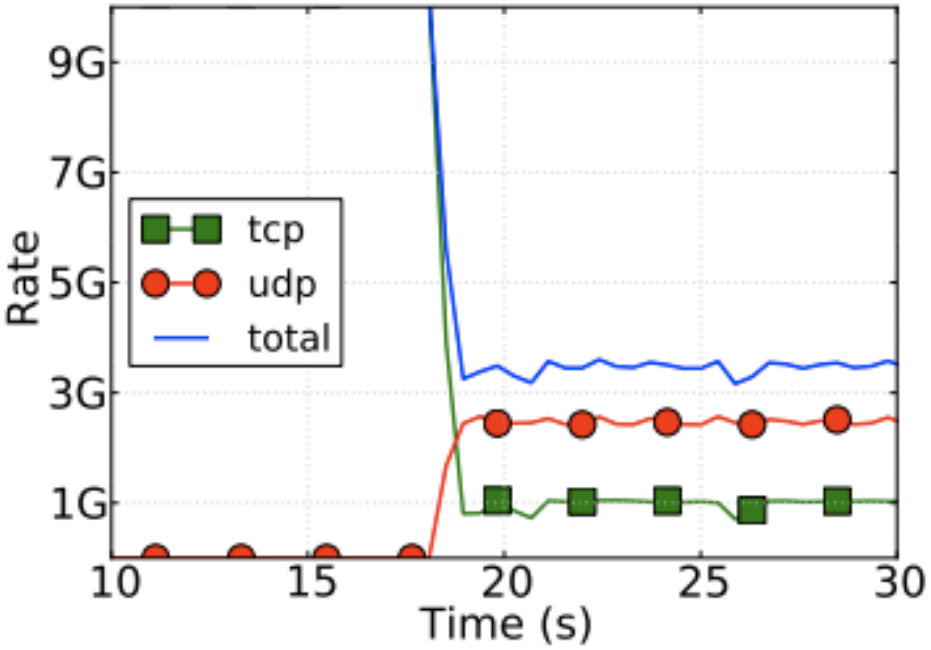


Without EyeQ

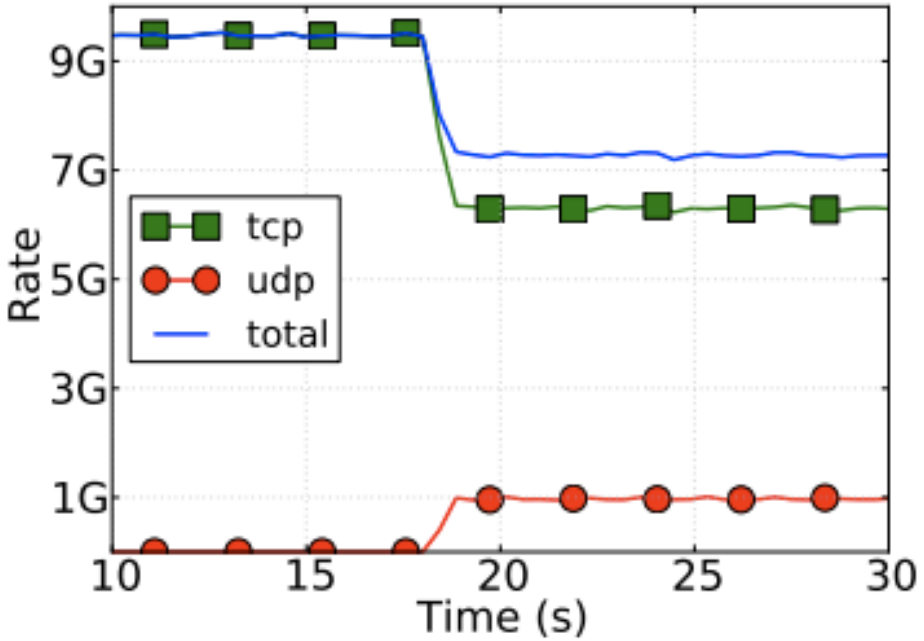


Does it work?

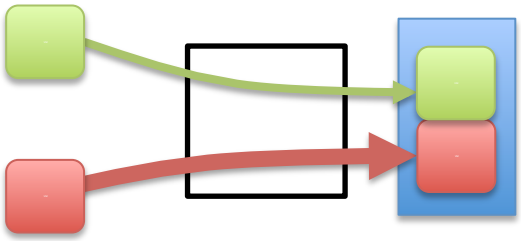
TCP: 6Gb/s
UDP: 3Gb/s



Without EyeQ

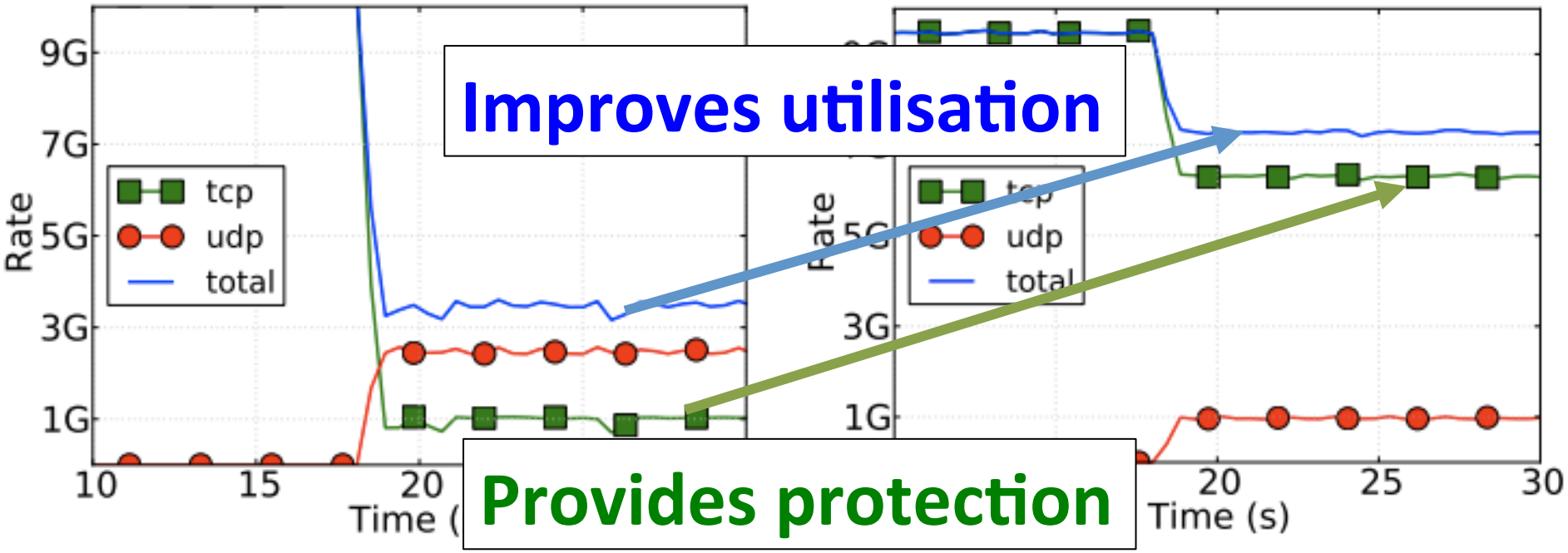


With EyeQ



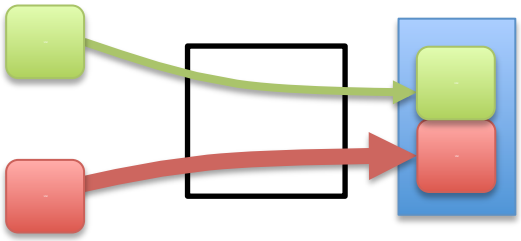
Does it work?

TCP: 6Gb/s
UDP: 3Gb/s



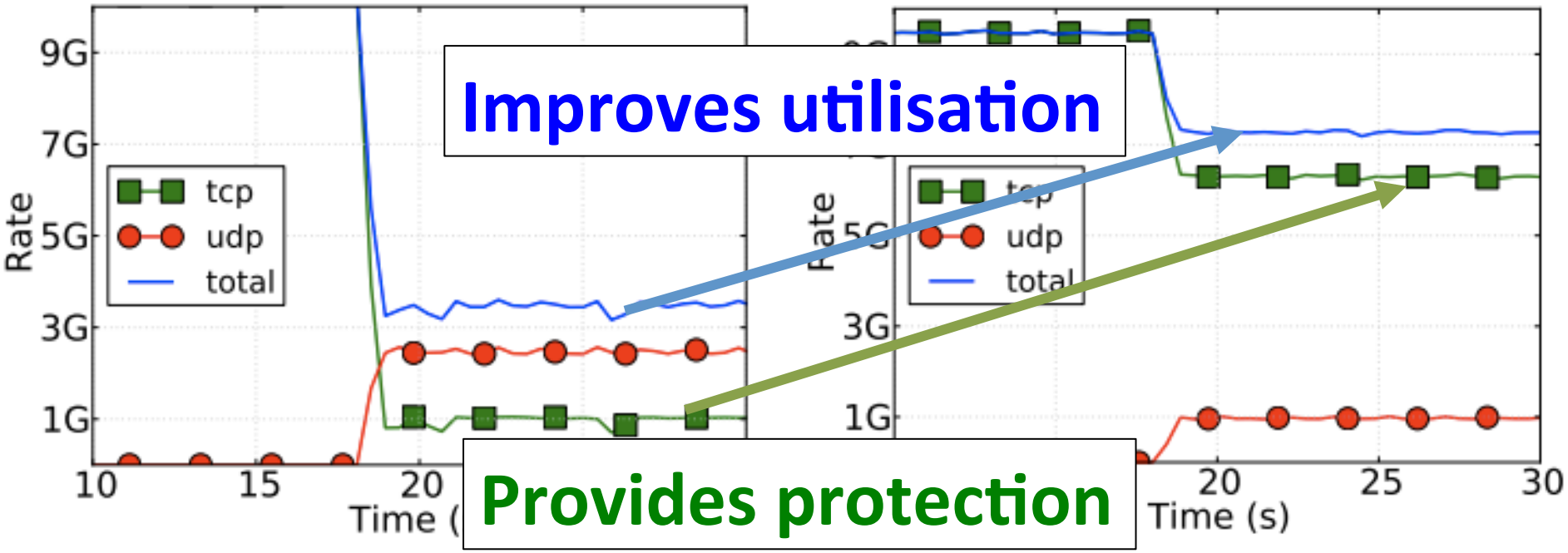
Without EyeQ

With EyeQ



Does it work?

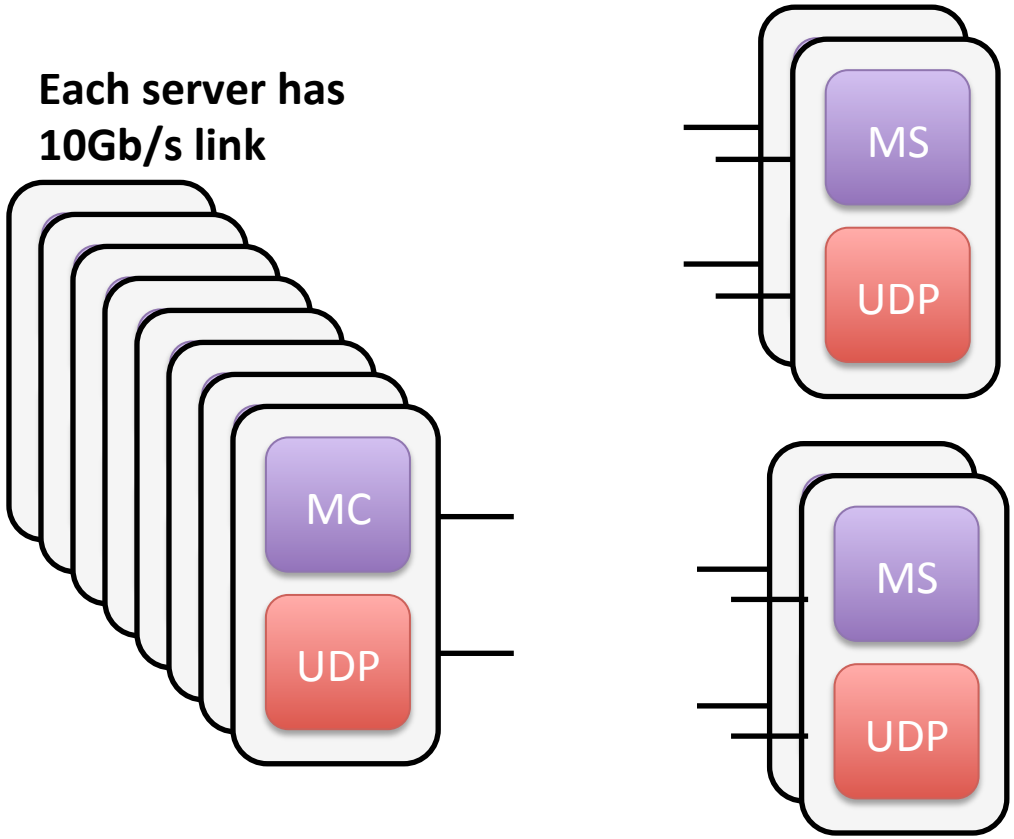
TCP: 6Gb/s
UDP: 3Gb/s



Without EyeQ

With EyeQ

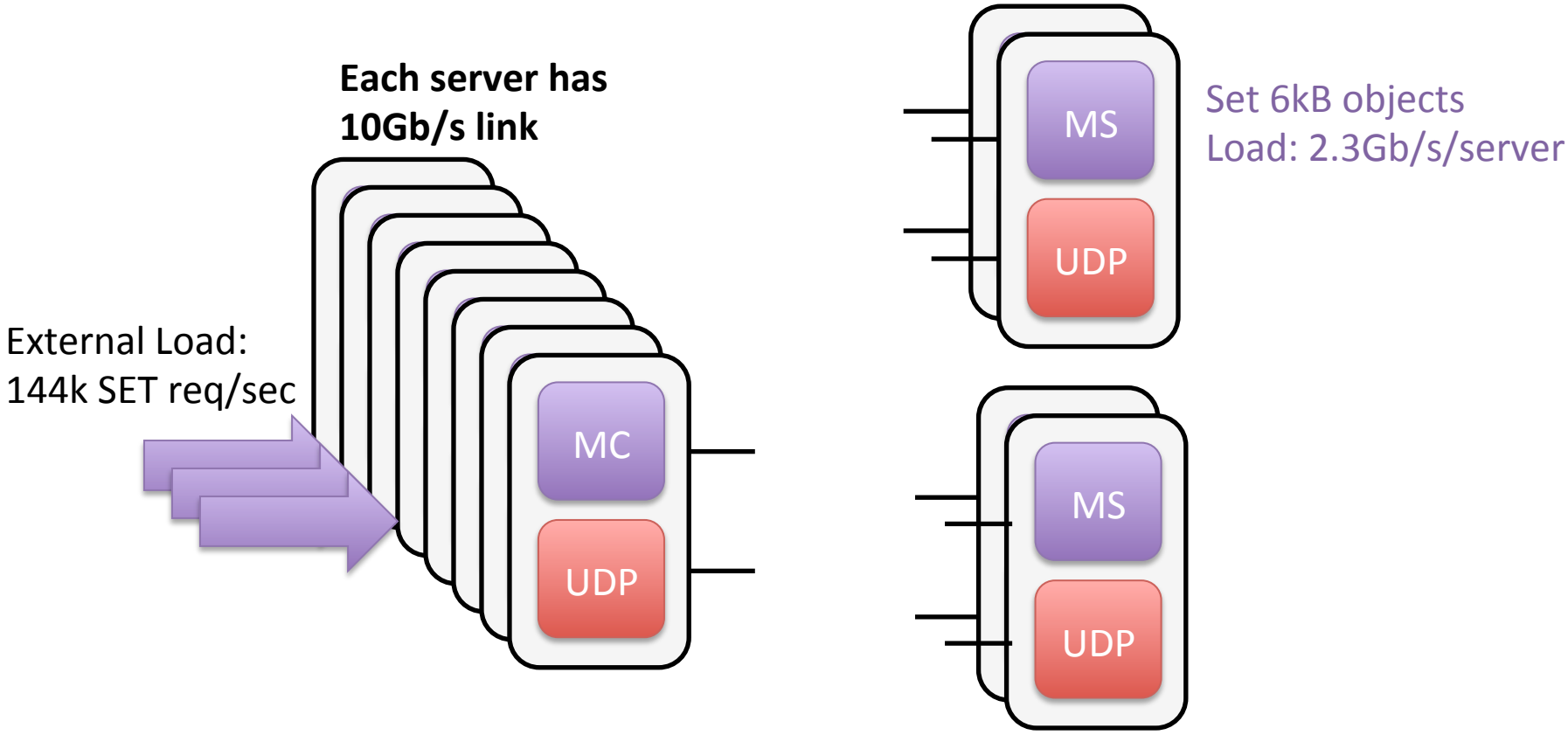
Close to Bare-Metal Latency?



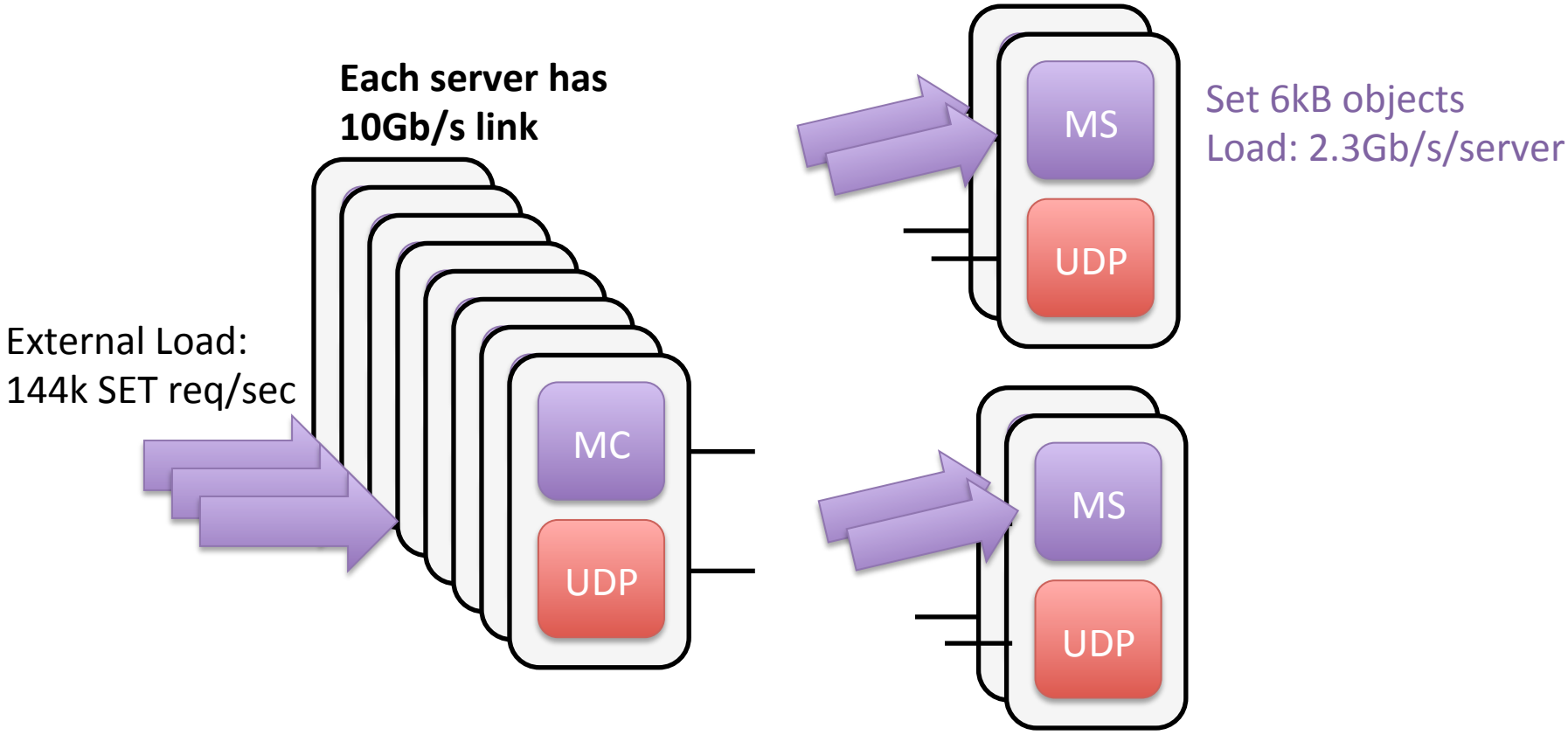
12 Client Pool

4 Server Pool

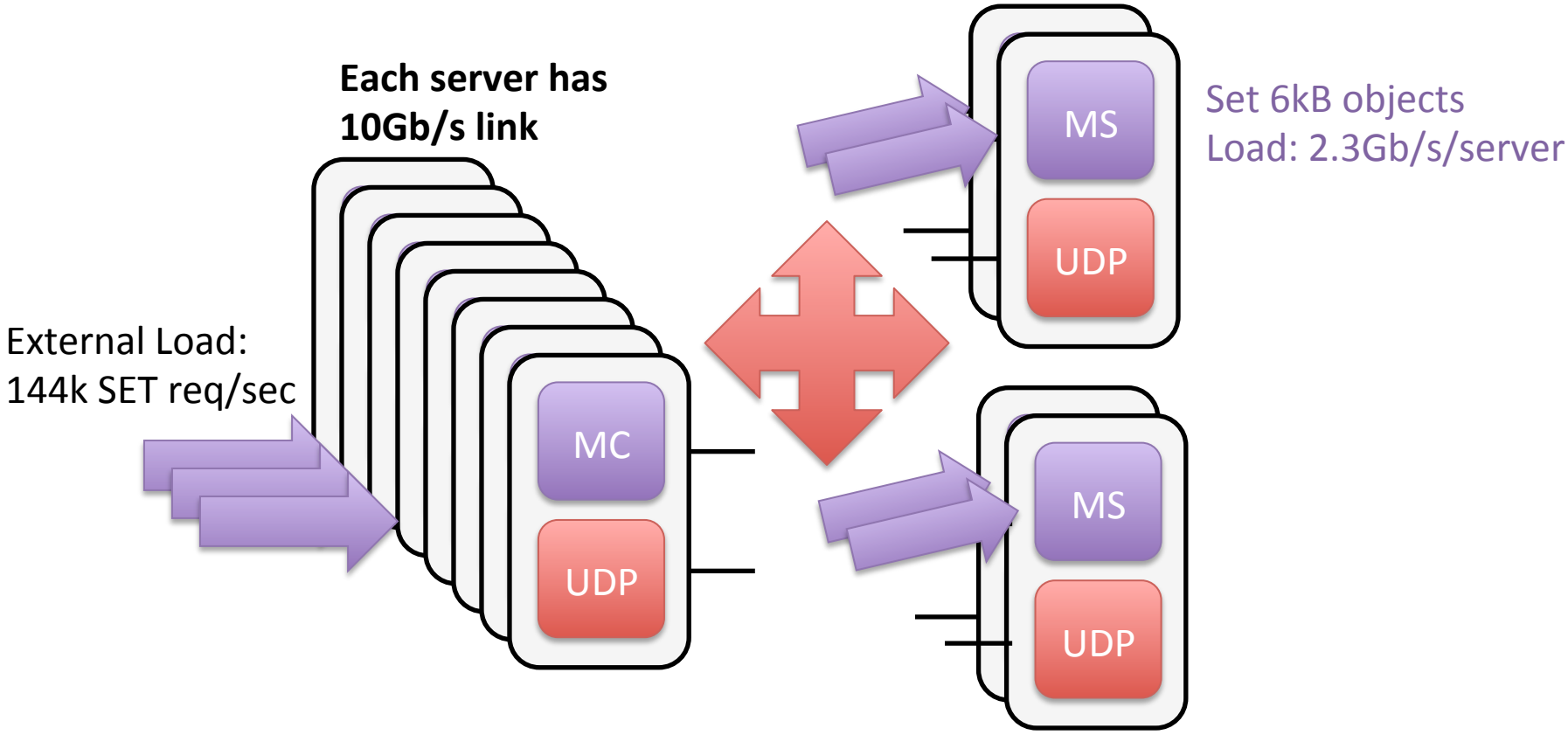
Close to Bare-Metal Latency?



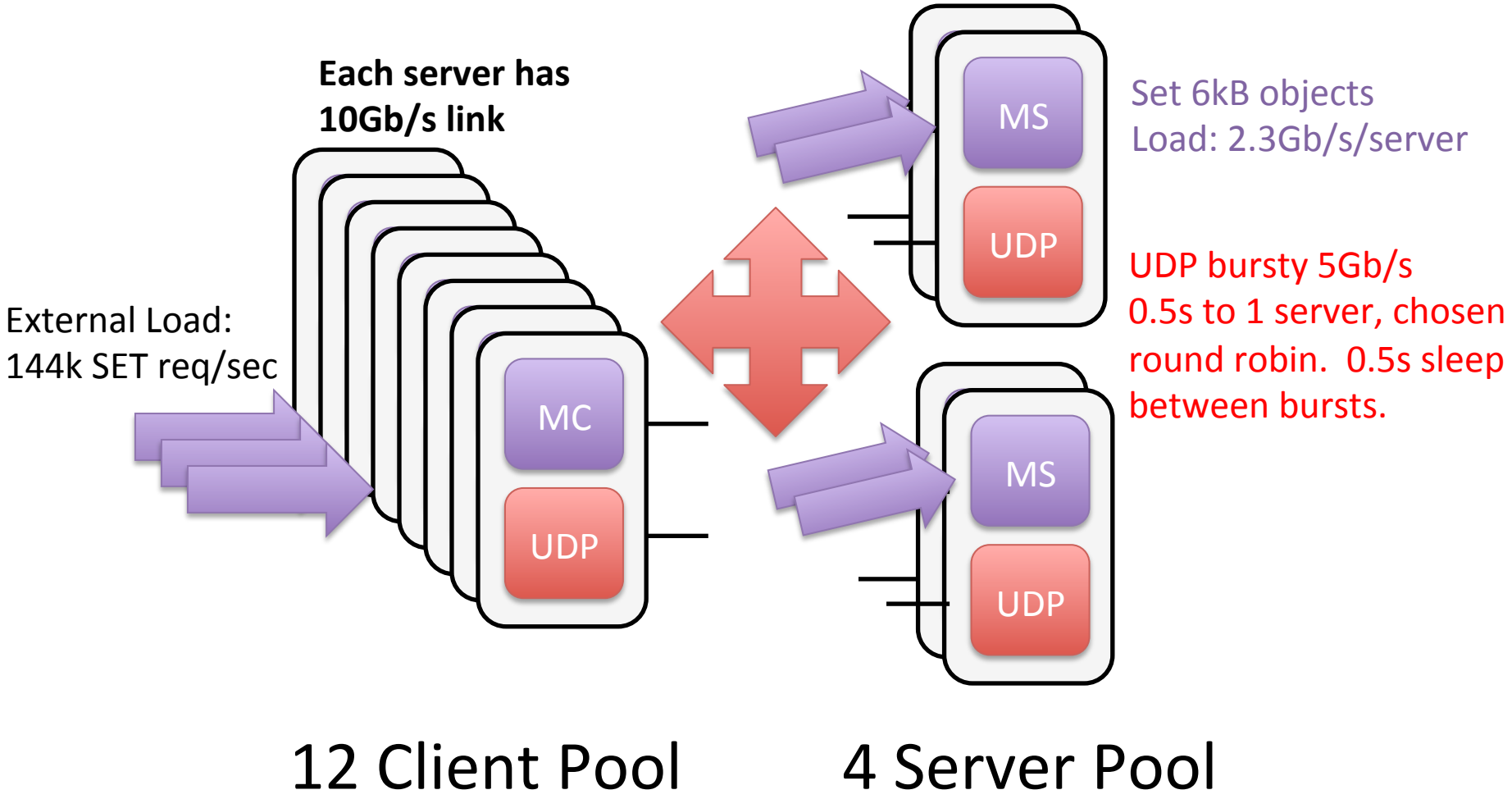
Close to Bare-Metal Latency?



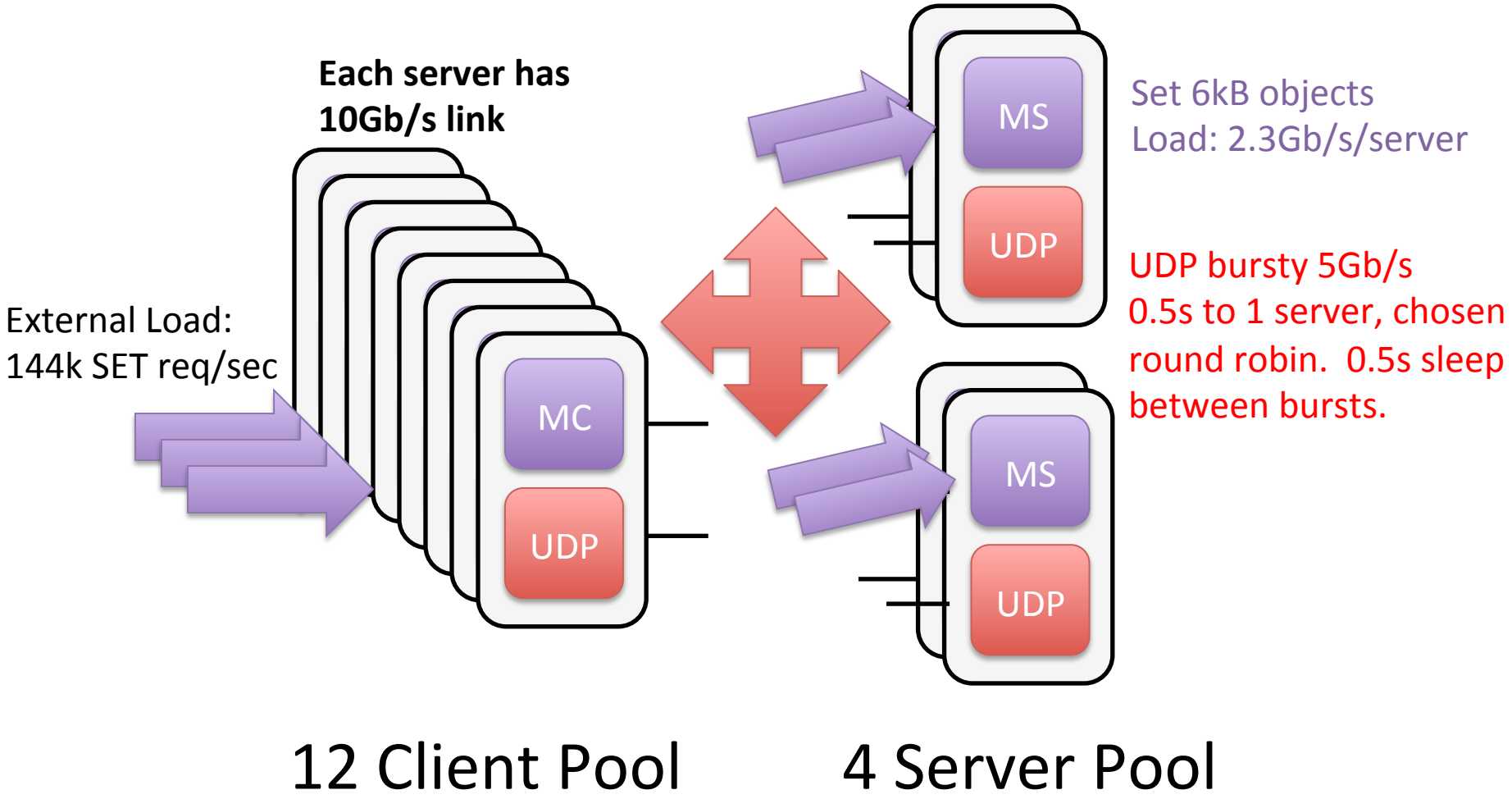
Close to Bare-Metal Latency?



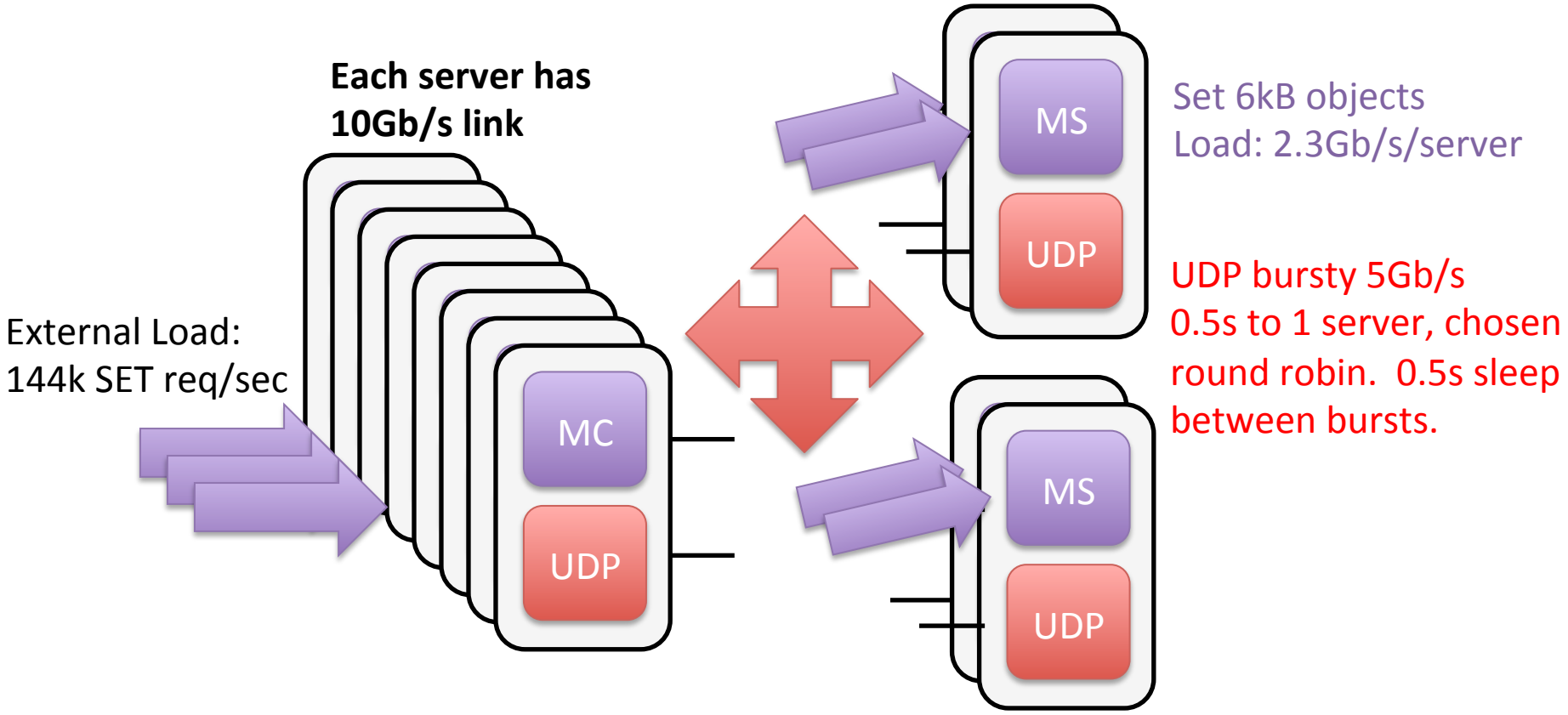
Close to Bare-Metal Latency?



Close to Bare-Metal Latency?



Close to Bare-Metal Latency?



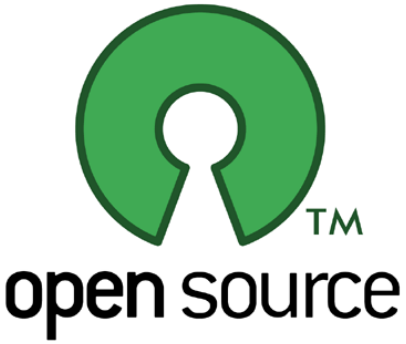
Close to Bare-Metal Latency?

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	**>10^6**us	144kreq/s
With Interference + EyeQ	102us	**750**us	144kreq/s

12 Client Pool 4 Server Pool

Thank you!

EyeQ: a system to partition bandwidth within a data center in a simple and predictable way



http://jvimal.github.com/eyeq

jvimal@stanford.edu

Rate Limiter Memory Overhead

112B + NCUPS * 104B

- 8 CPUs: ~0.9kB
- 16 CPUs: ~1.8kB
- Scales linearly with number of IP destinations, not connections (struct sock: 648B)