RouteBricks: A Fast, Software-Based, *Distributed* IP Router

Brad Karp

UCL Computer Science

(with thanks to Katerina Argyraki of EPFL for slides)

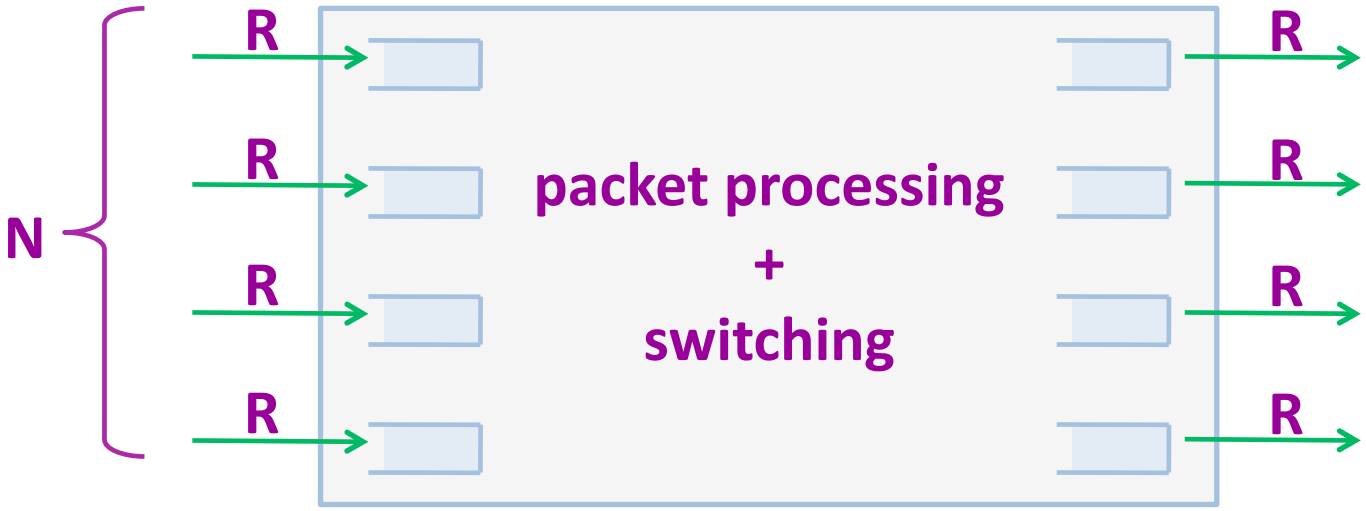
CS GZ03 / M030

15th November, 2010

Distributed Systems Context

- Many models of consistency:
  - sequential consistency (Ivy)
  - close-to-open consistency (NFS)
  - eventual consistency/conflict resolution despite disconnection (Bayou)
  - atomic appends for multiple writers (GFS)
- Cluster as platform: Ivy, GFS
- Today: distributing an IP router over a cluster of PCs, for capacity and programmability
  - highly parallel workload: packets forwarded independently (apart from flow ordering constraint)
  - distributed within a single PC: multiple multi-core CPUs

Router =



- Fast = high R, N
- Programmable = any processing

Why programmable routers?

- **New ISP services**
  - » intrusion detection, application acceleration
- **Monitoring**
  - » measure link latency, track down traffic
- **New protocols**
  - » IP traceback, Trajectory Sampling, …

Today: fast OR programmable

- **Fast "hardware" routers**
  - » processing by specialized hardware
  - » not programmable
  - » aggregate throughput: Tbps

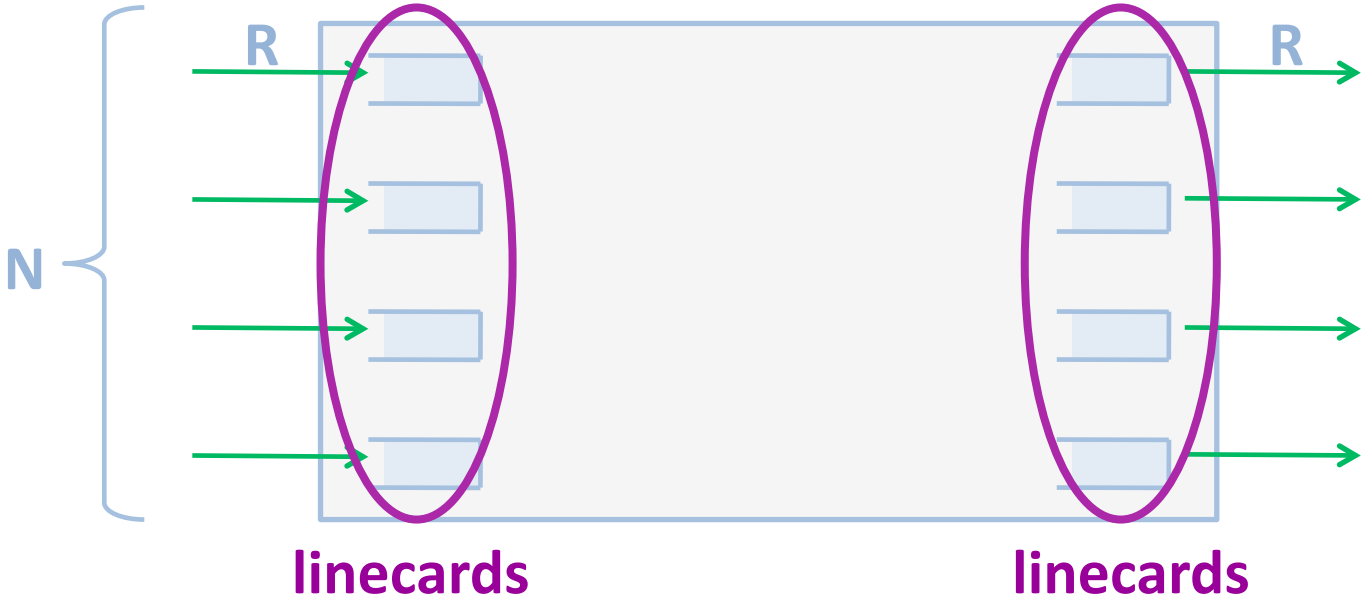
- **Programmable "software" routers**
  - » processing by general-purpose CPUs
  - » aggregate throughput < 10Gbps

RouteBricks

- **A router out of off-the-shelf PCs**
  - » familiar programming environment
  - » large-volume manufacturing (cheap, widely available, growing in CPU and I/O with PCs)

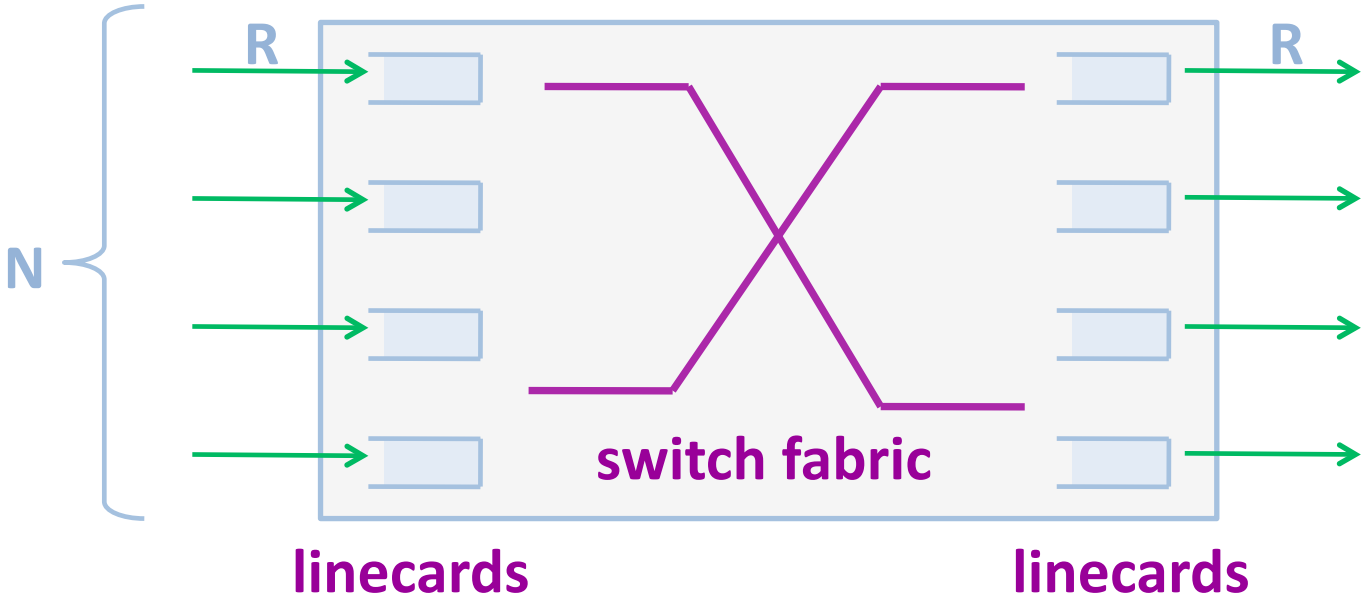
- ***Can we build a Tbps router out of PCs?***

A hardware router



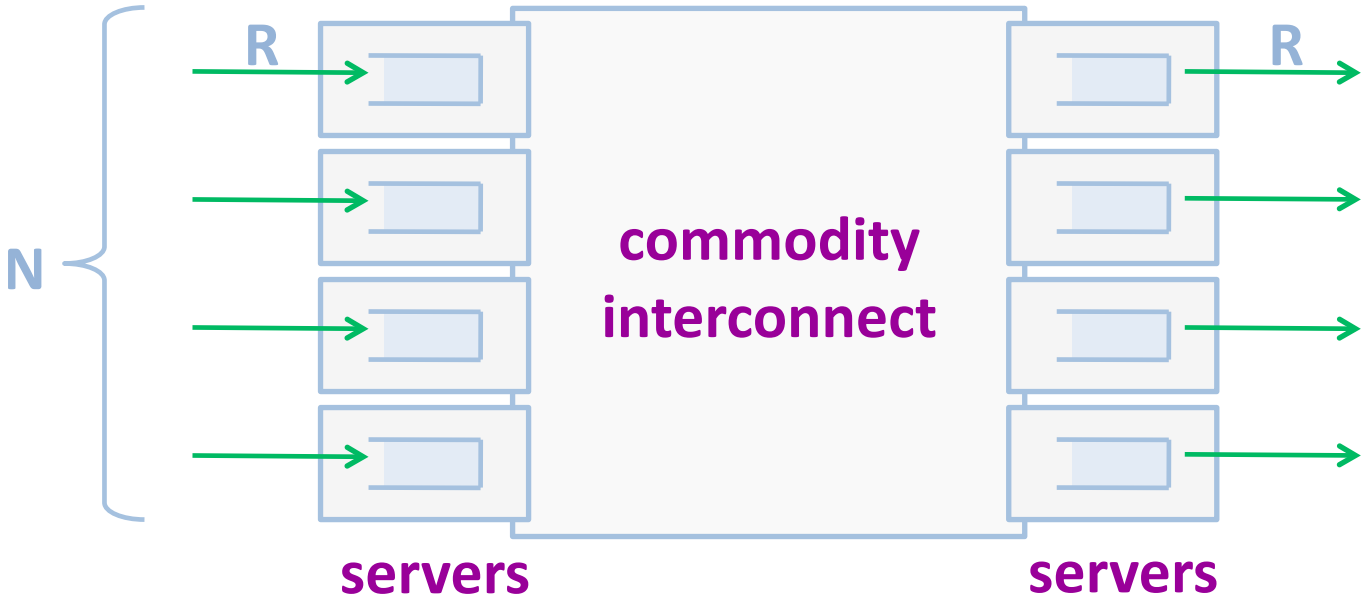
- Processing at rate ~*R* per linecard

A hardware router



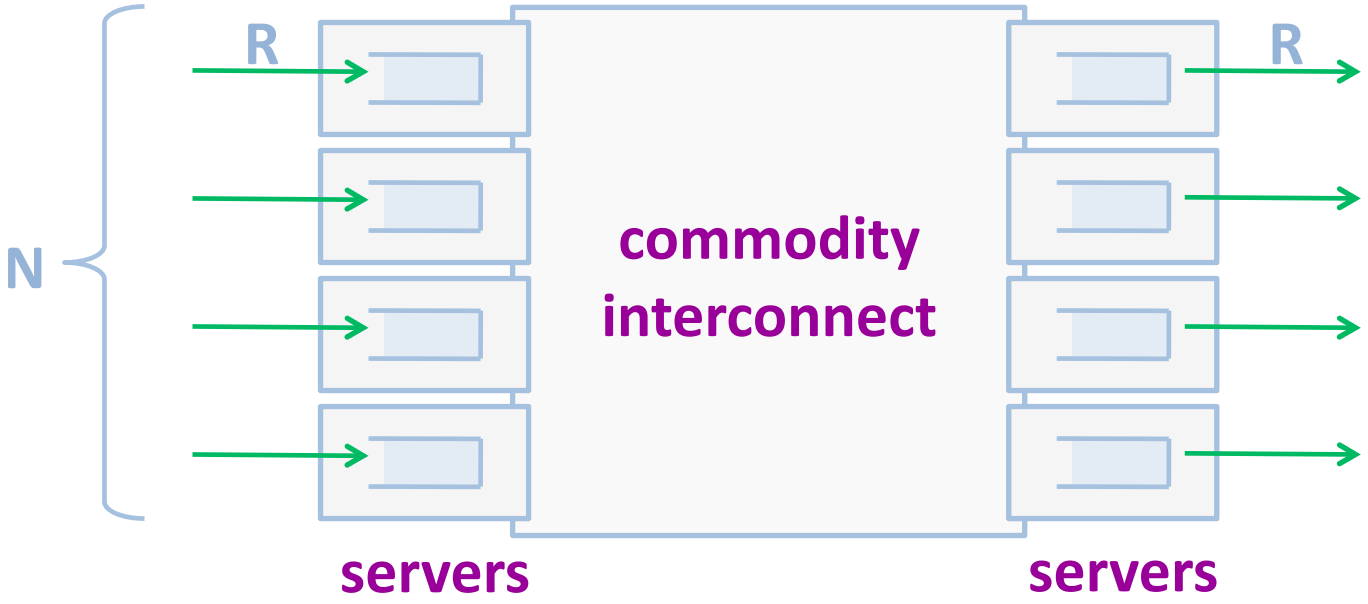
- Processing at rate ~*R* per linecard
- Switching at rate *NR* by switch fabric

RouteBricks



- Processing at rate ~*R* per server
- Switching at rate ~*R* per server

RouteBricks

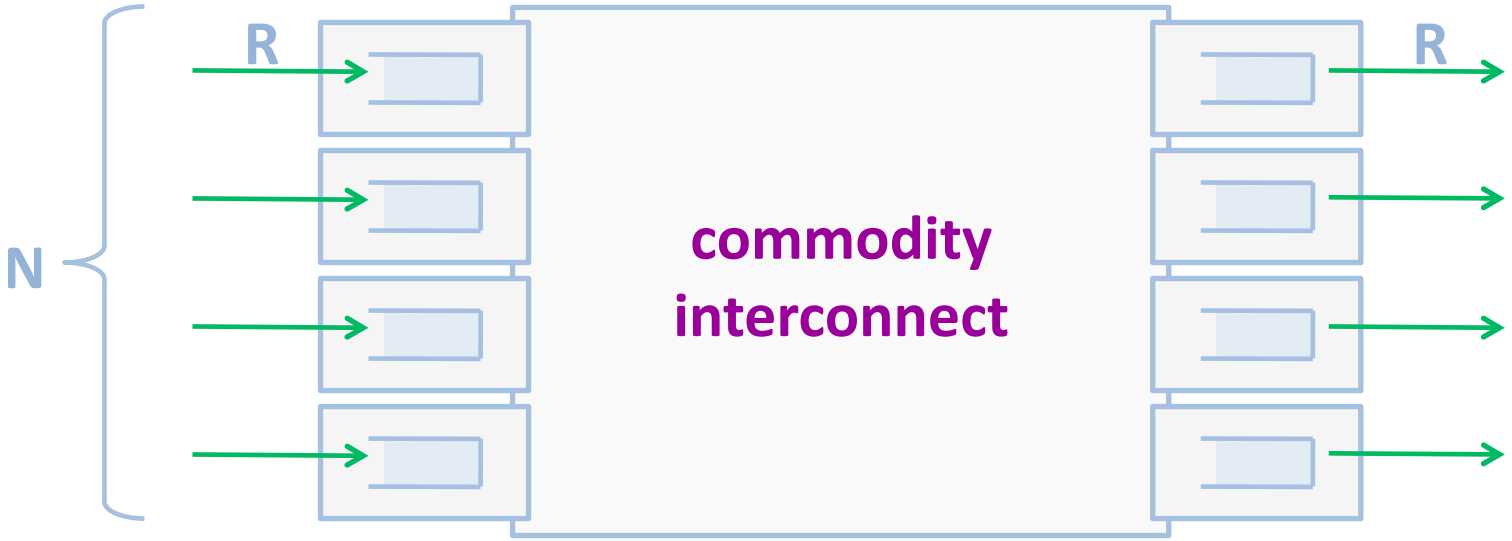


Per-server processing rate: *cR*

Outline

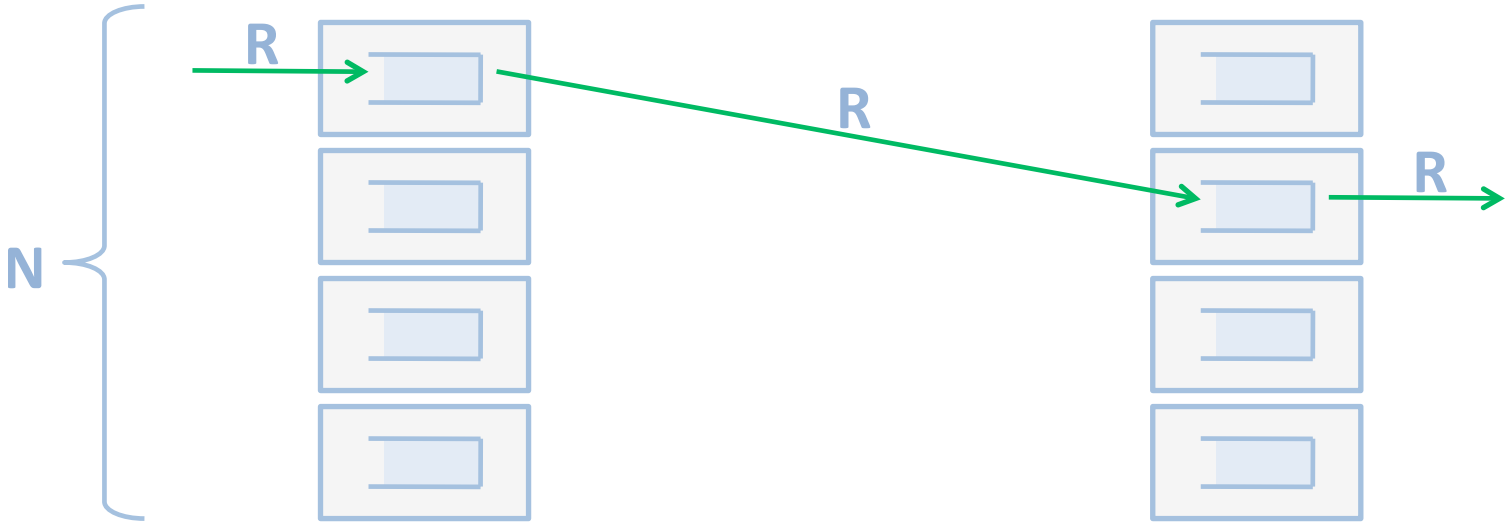
- Interconnect
- Server optimizations
- Performance
- An application
- Conclusions

Requirements

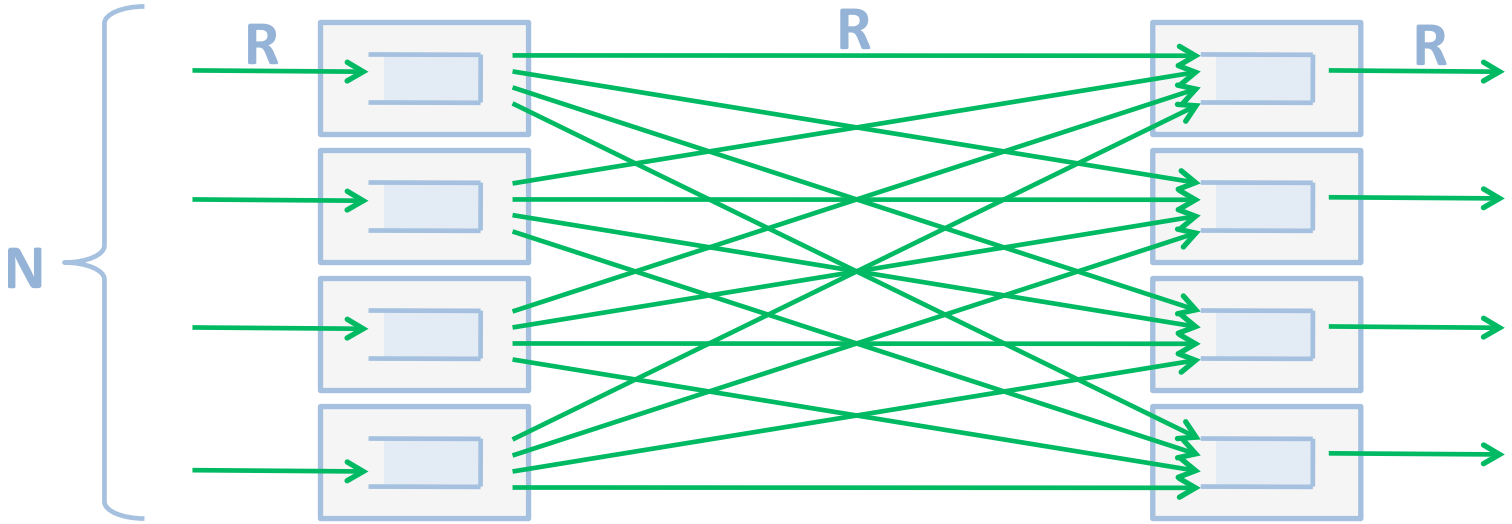


- Internal link rates < *R*
- Per-server processing rate: *cR*
- Per-server fanout: constant

Straw Man: Direct Full Mesh

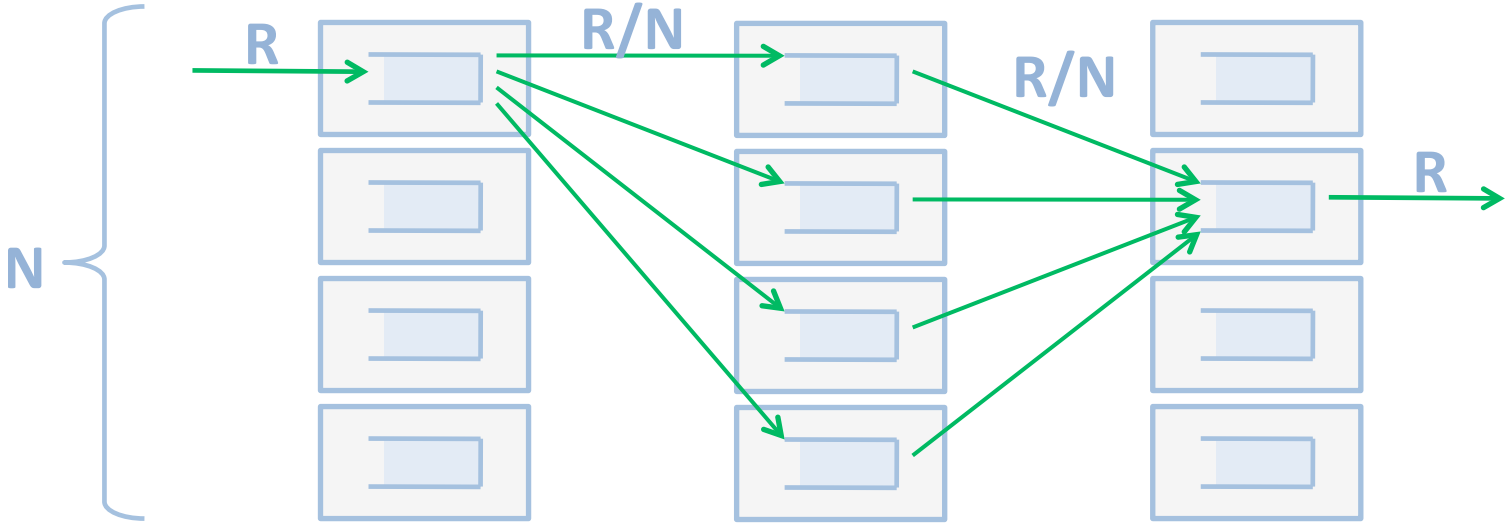


Straw Man: Direct Full Mesh

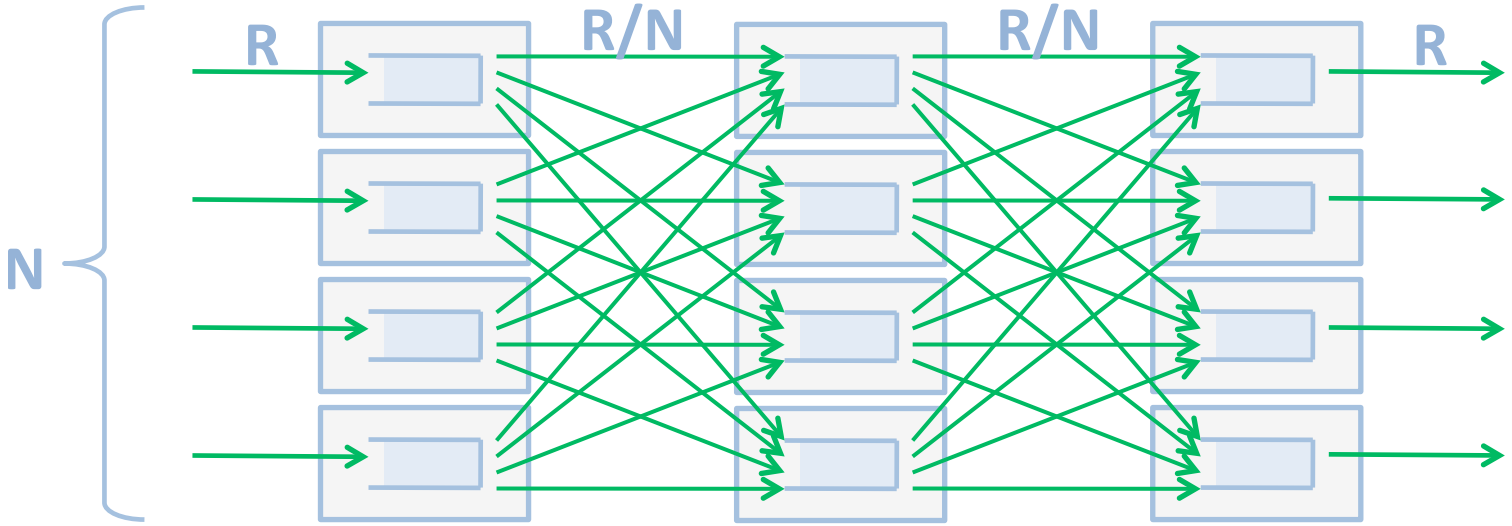


- *N* external links of capacity *R*
- N^2 internal links of capacity *R*

Better: Valiant Load Balancing [Valiant, Brebner, 1981]

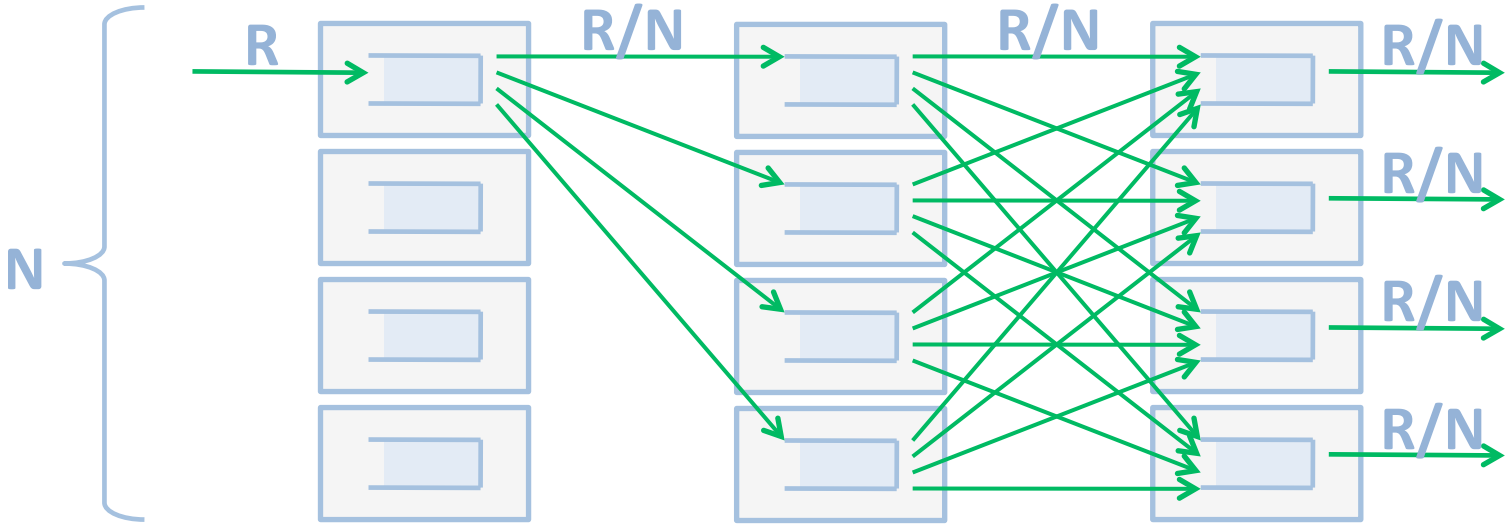


Valiant Load Balancing

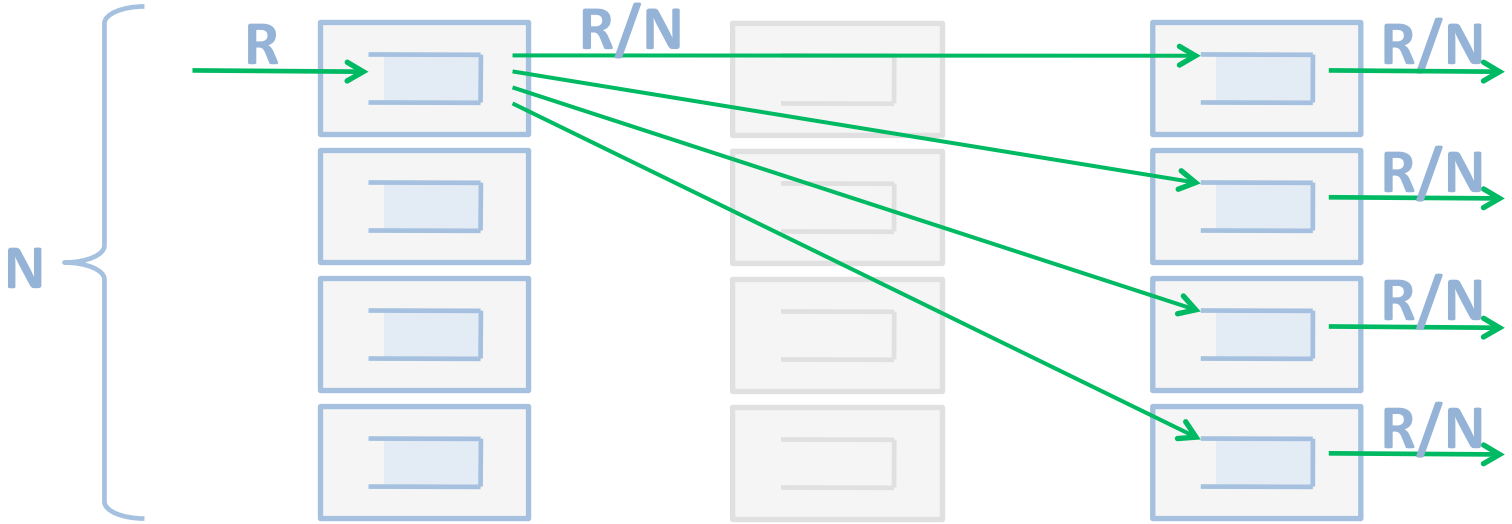


- Per-server processing rate: 3*R*
  - » processing overhead: 50% (2*R* without VLB)
- N^2 links of capacity 2*R*/*N*

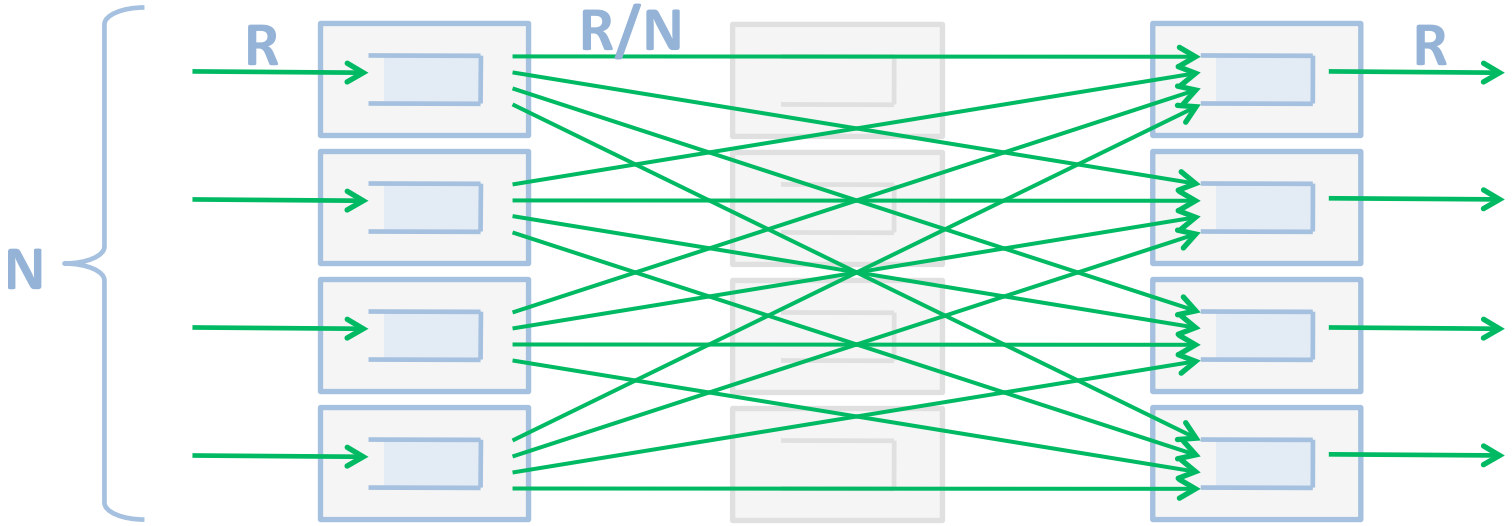
VLB with Uniform Traffic



Direct Valiant LB: Optimize for Uniform Traffic Load

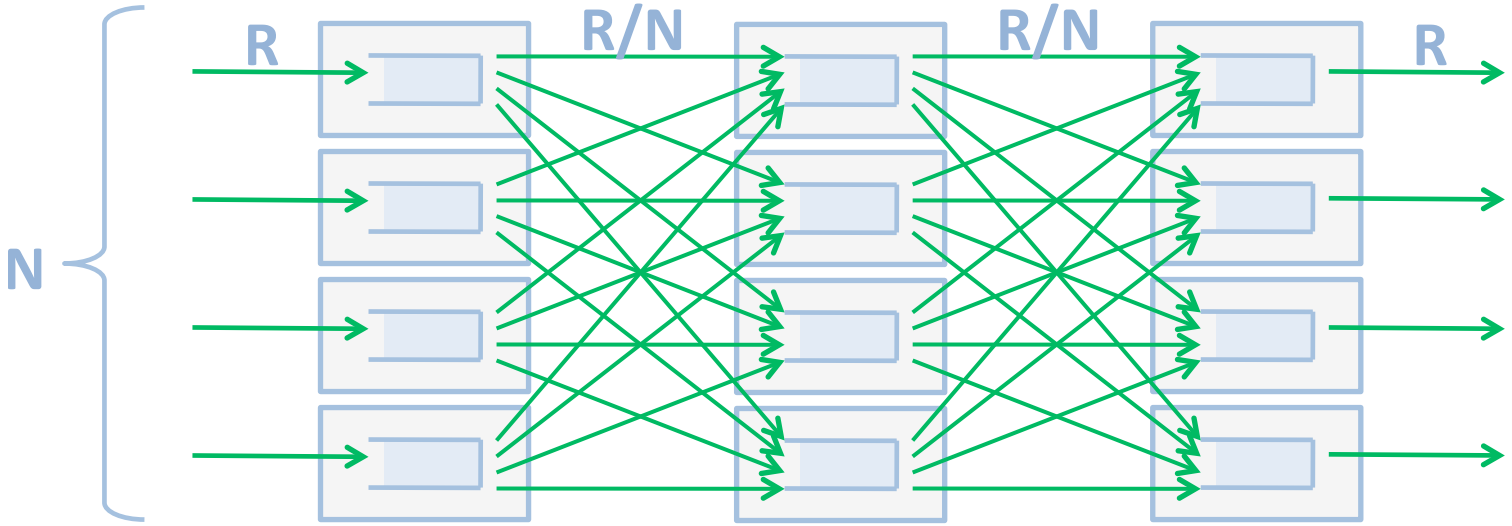


Direct Valiant LB



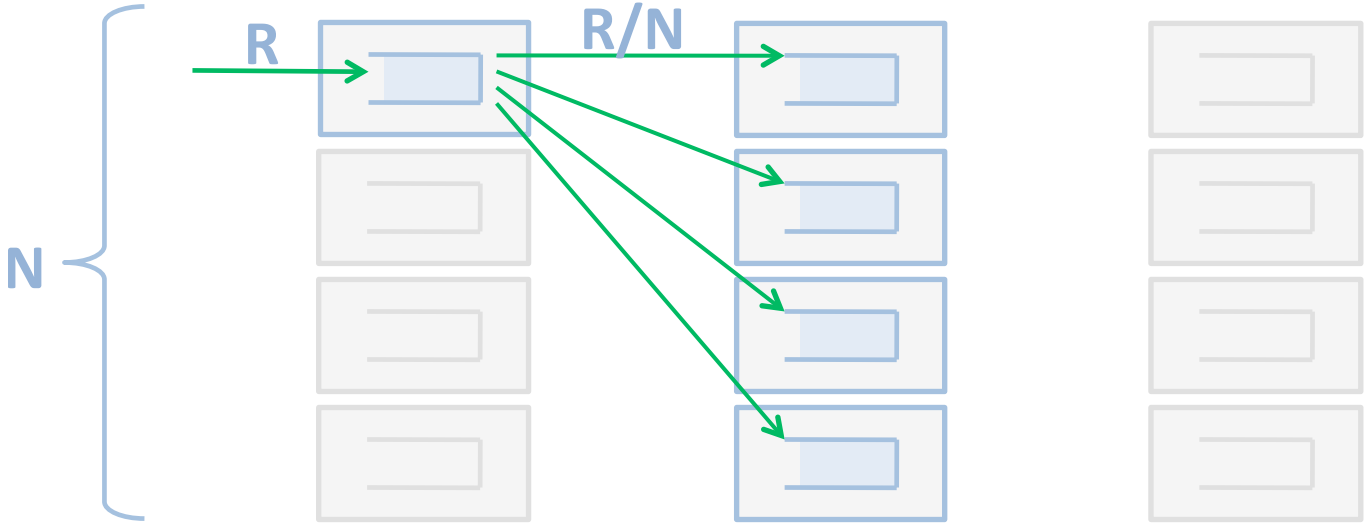
- If uniform traffic matrix: 2*R*

VLB Summary



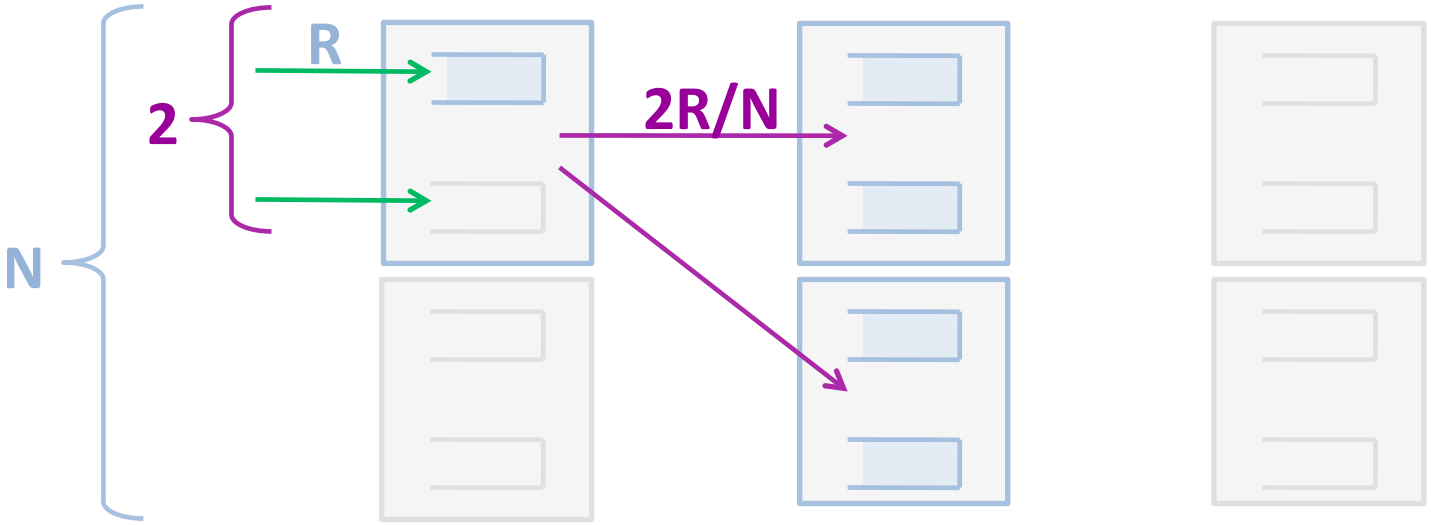
- Worst-case per-server processing rate: 3*R*
- If uniform traffic matrix: 2*R*

Per-Server Fanout?



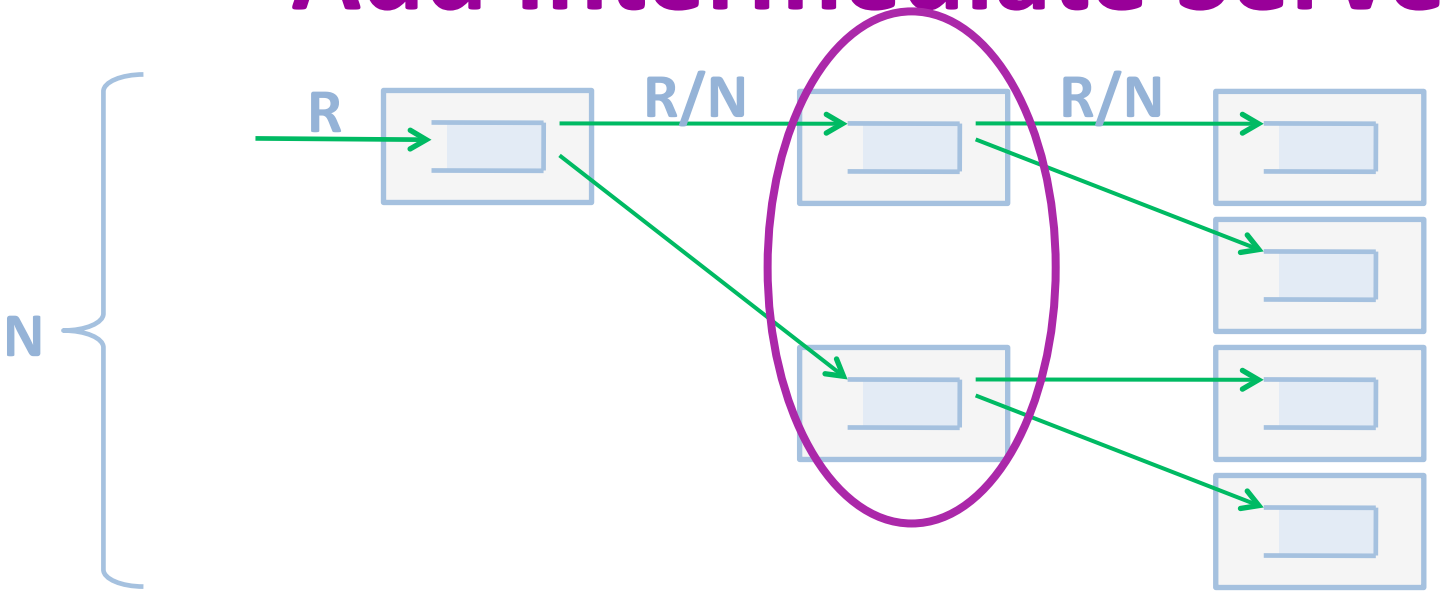
- Connectivity degree: *N* per server
- What if not enough network ports?

Solution #1: Increase Server Capacity



- e.g., double # external ports per server
- Doubles data rate on internal links, processing rate per server
- Cuts fanout by half

Solution #2: Add Intermediate Servers



- *k*-degree, *n*-stage butterfly ("*k*-ary *n*-fly")
- Per-server fanout: *k*
- Stages: $n = \log_k N$

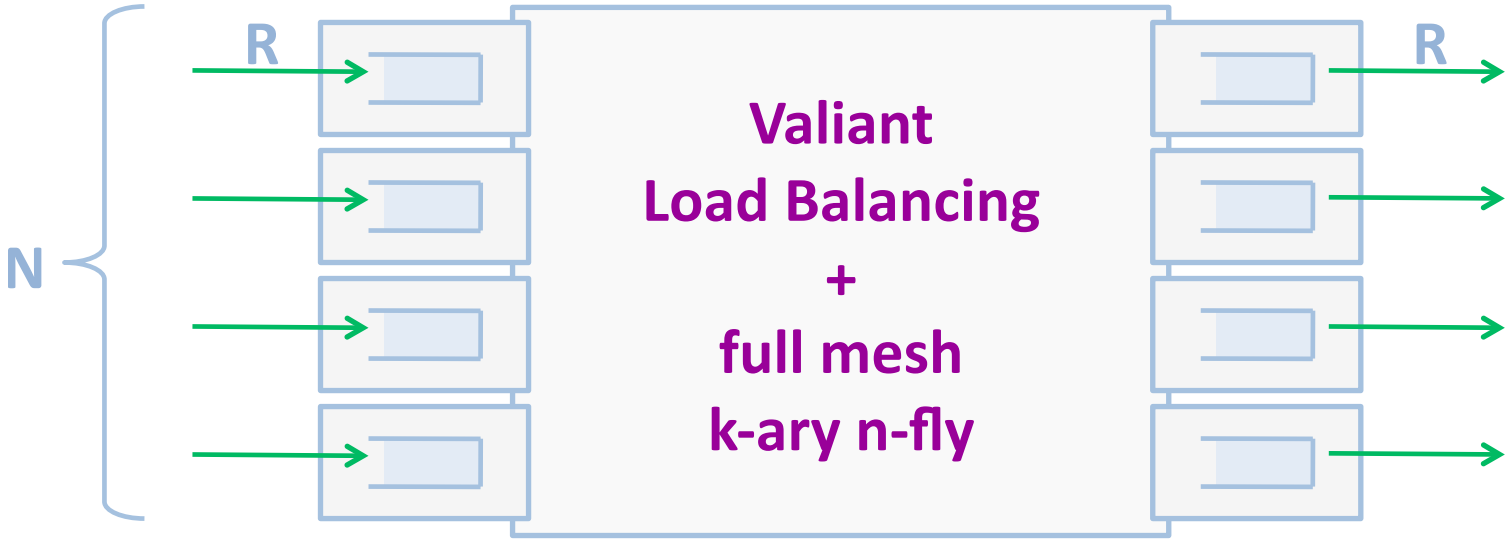
The RouteBricks Interconnect: Combination

- Assign max external ports per server
- Full mesh if fanout allows
- Extra servers otherwise

Example

- Assuming current servers
  - » 5 NICs, 2 x 10G ports or 8 x 1G ports
  - » 1 external port per server
- N = 32 ports: full mesh
  - » 32 servers
- N = 1024 ext. ports: 16-ary 4-fly
  - » 3072 servers (2 extra servers per port)

Recap

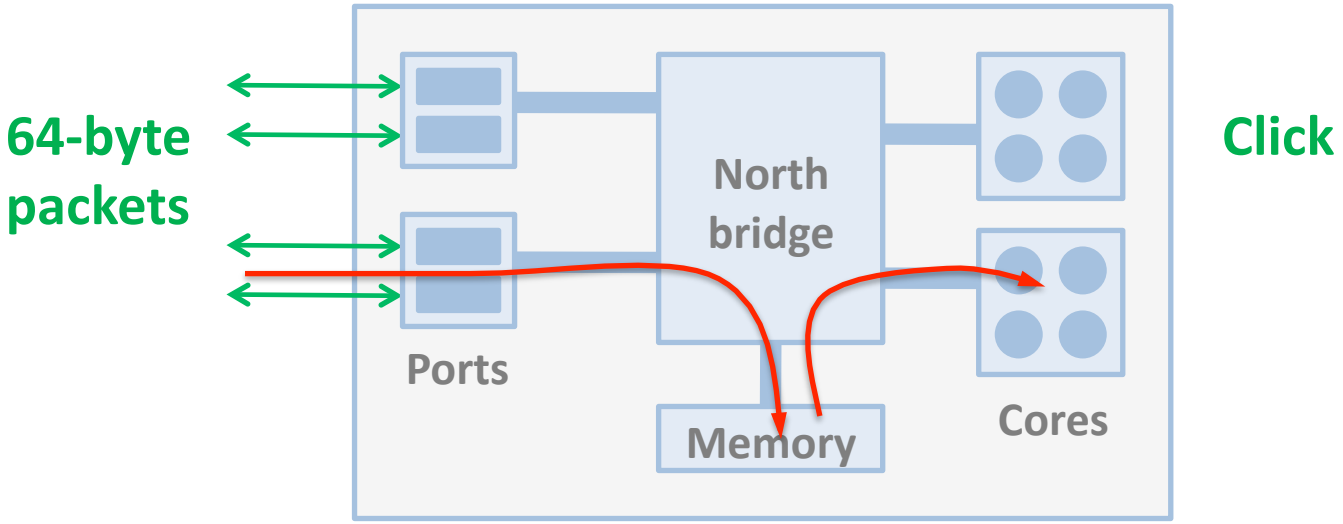


Per-server processing rate: 2*R* – 3*R*

Outline

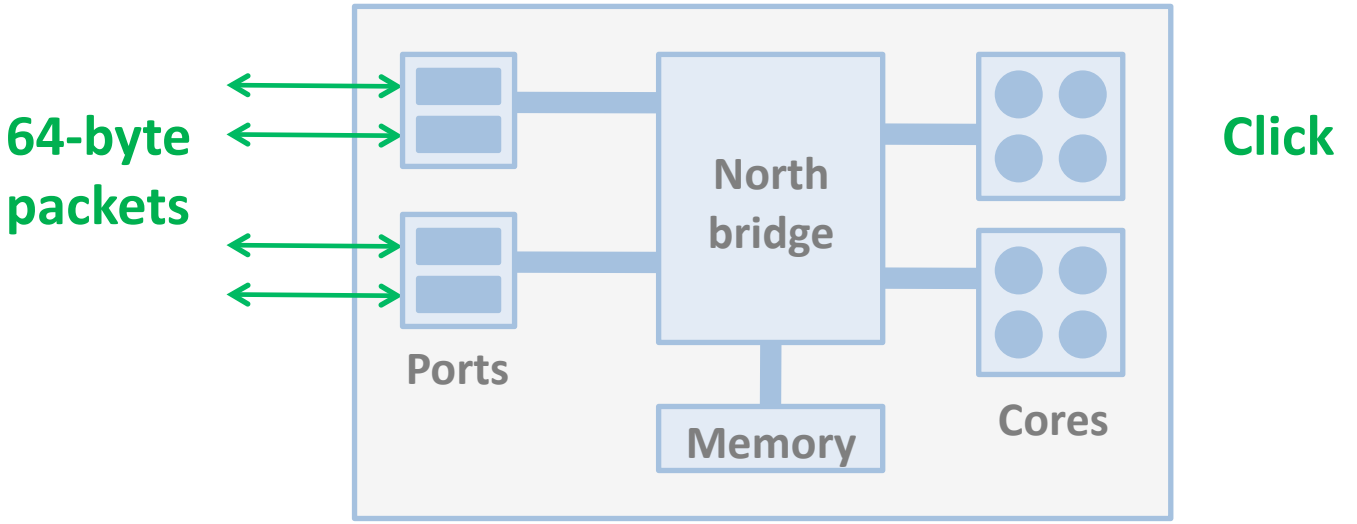
- Interconnect
- **Server optimizations**
- Performance
- An application
- Conclusions

First Try: a Shared-Bus Server

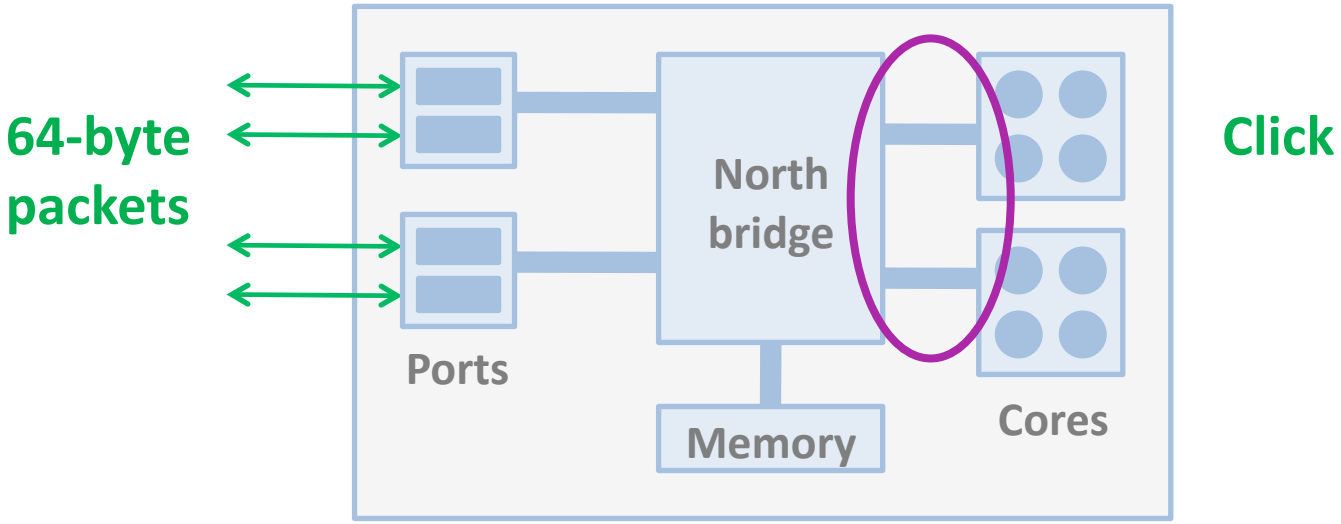


- Cloverton architecture
- FSB: 2 x 1.33 GHz
- CPUs: 2 x Xeon 2.4GHz 4-core
- NICs: 2 x Intel XFSR 2x10Gbps

First Try: a Shared-Bus Server



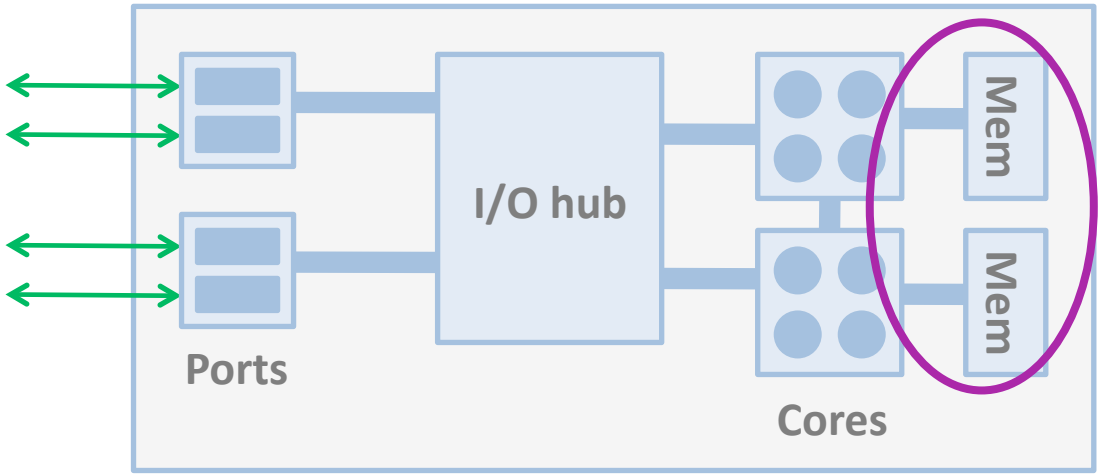
Problem #1: the Shared Bus



FSB address bus saturated

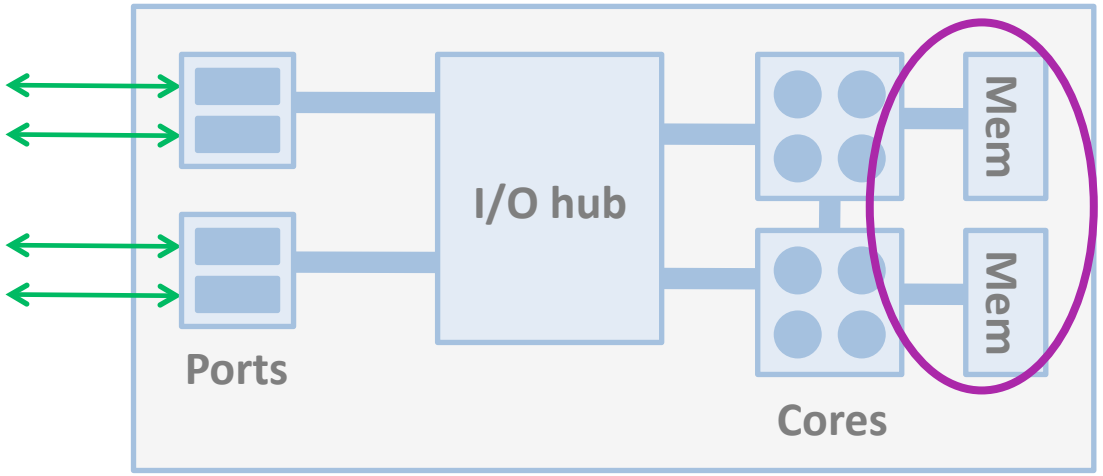
Multi-core alone is not enough

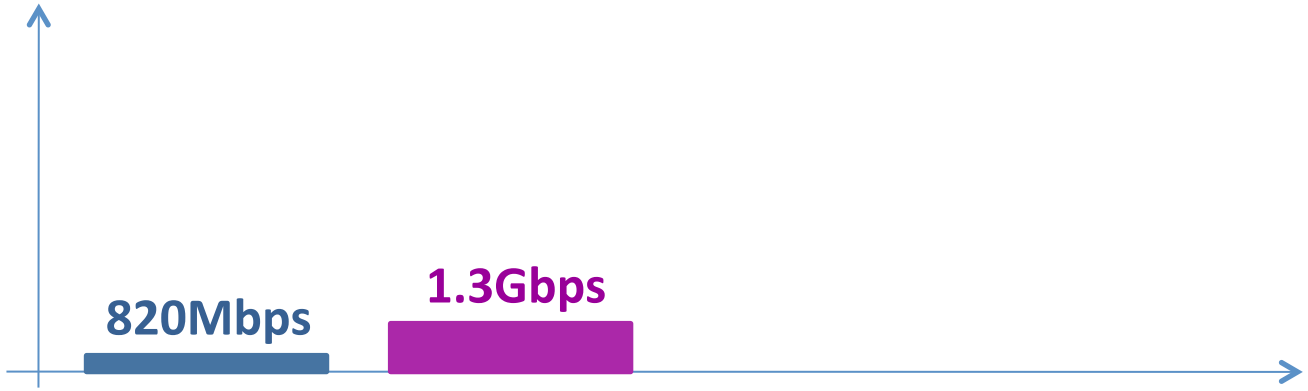
Solution: NUMA architecture



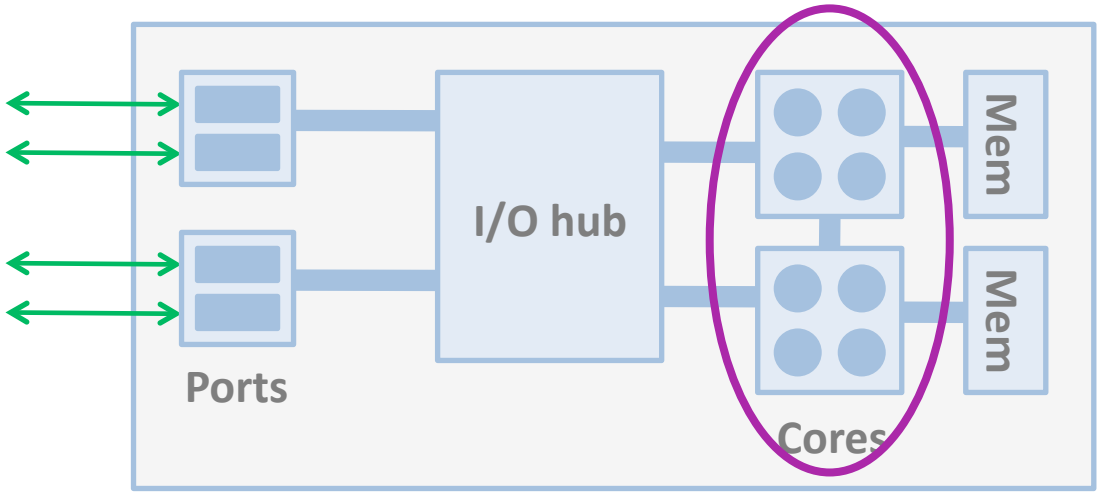
- Nehalem architecture
- QuickPath interconnect
- CPUs: 2 x Xeon 2.4GHz 4-core
- NICs: 2 x Intel XFSR 2x10Gbps

Solution: NUMA architecture





Problem #2: Per-Packet Overhead

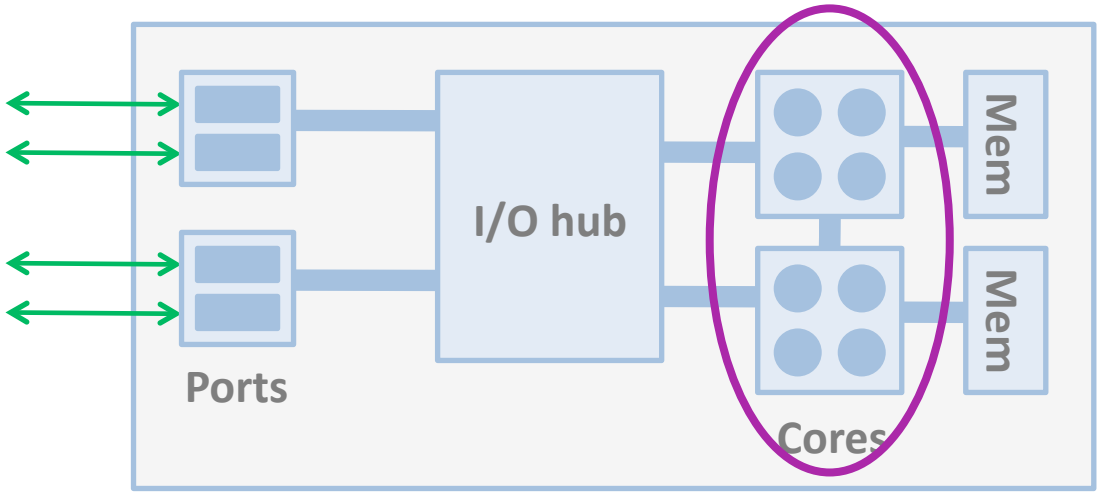


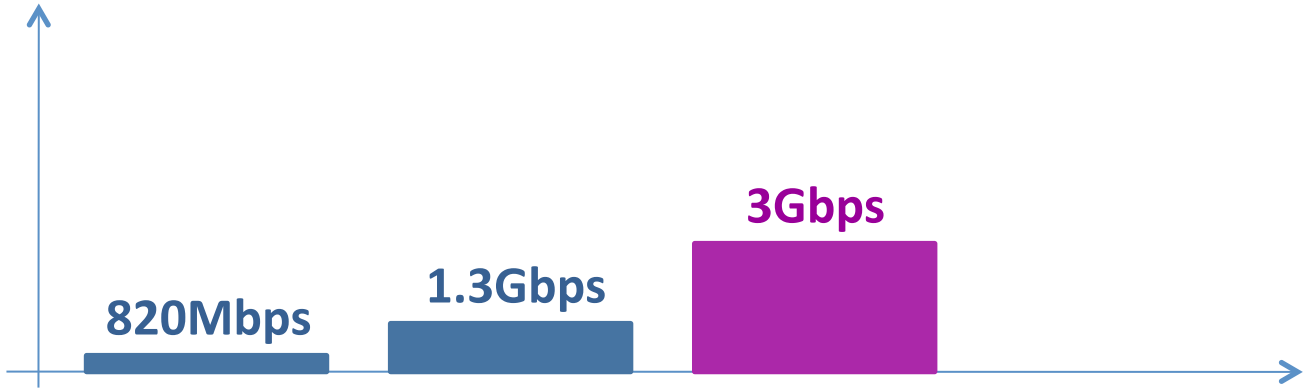
- **Bookkeeping operations**
  - » moving packet descriptors between NIC and memory
  - » updating descriptor rings

Solution: Batching

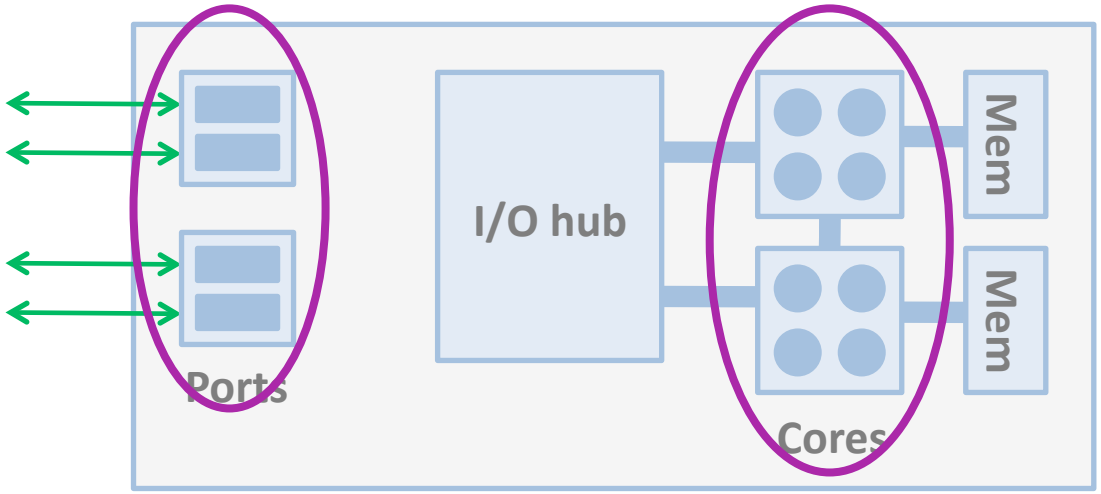
- **Poll-driven batching**
  - » poll multiple packets at a time
  - » reduces updates on descriptor rings
  - » Click already supported it
- **NIC-driven batching**
  - » relay multiple packet descriptors at a time
  - » reduces transactions on PCIe and I/O buses
  - » **changed NIC driver**

Solution: Batching



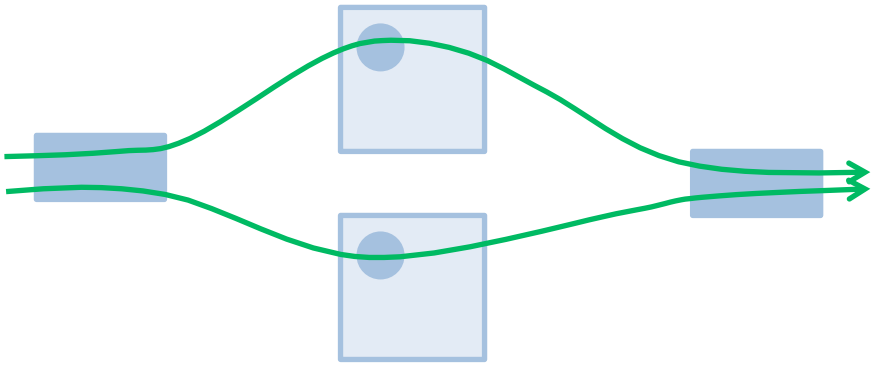


Problem #3: Queue Access



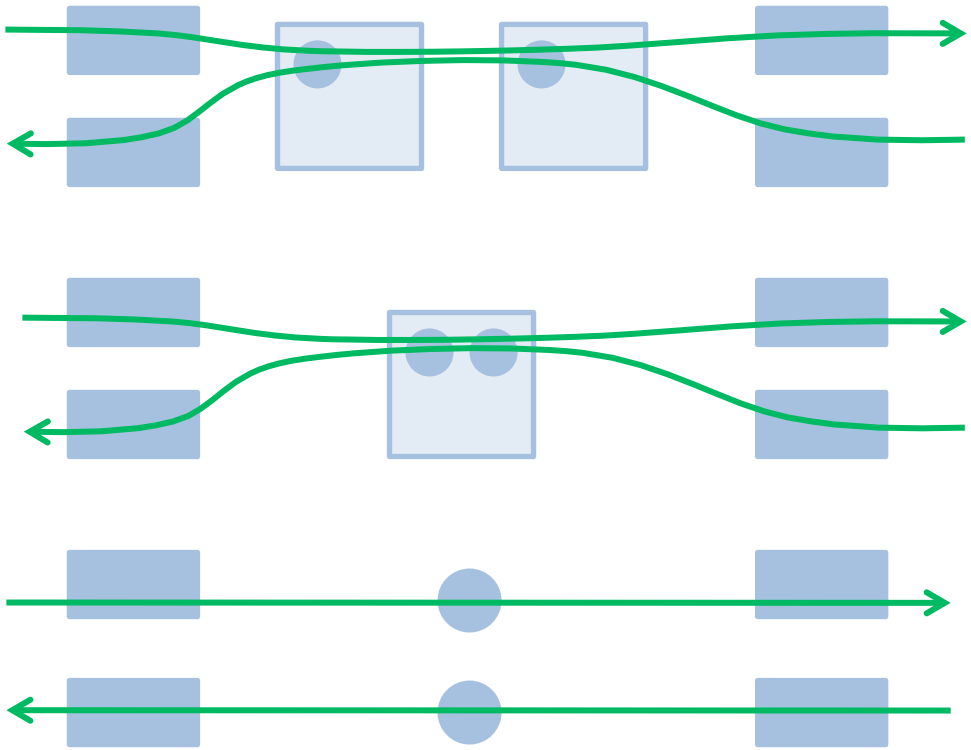
Problem #3: Queue Access

Problem #3: Queue Access



Rule 1: one core per port

Problem #3: Queue Access



0.6 Gbps

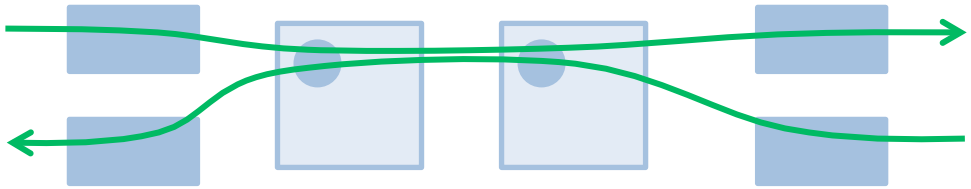
1.2 Gbps

1.7 Gbps

Rule 1: one core per port

Rule 2: one core per packet

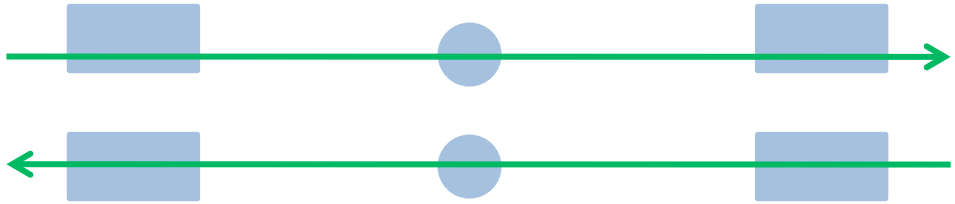
Problem #3: Queue Access



0.6 Gbps

Can we always enforce both rules?

What about when receiving on one 10 Gbps port?

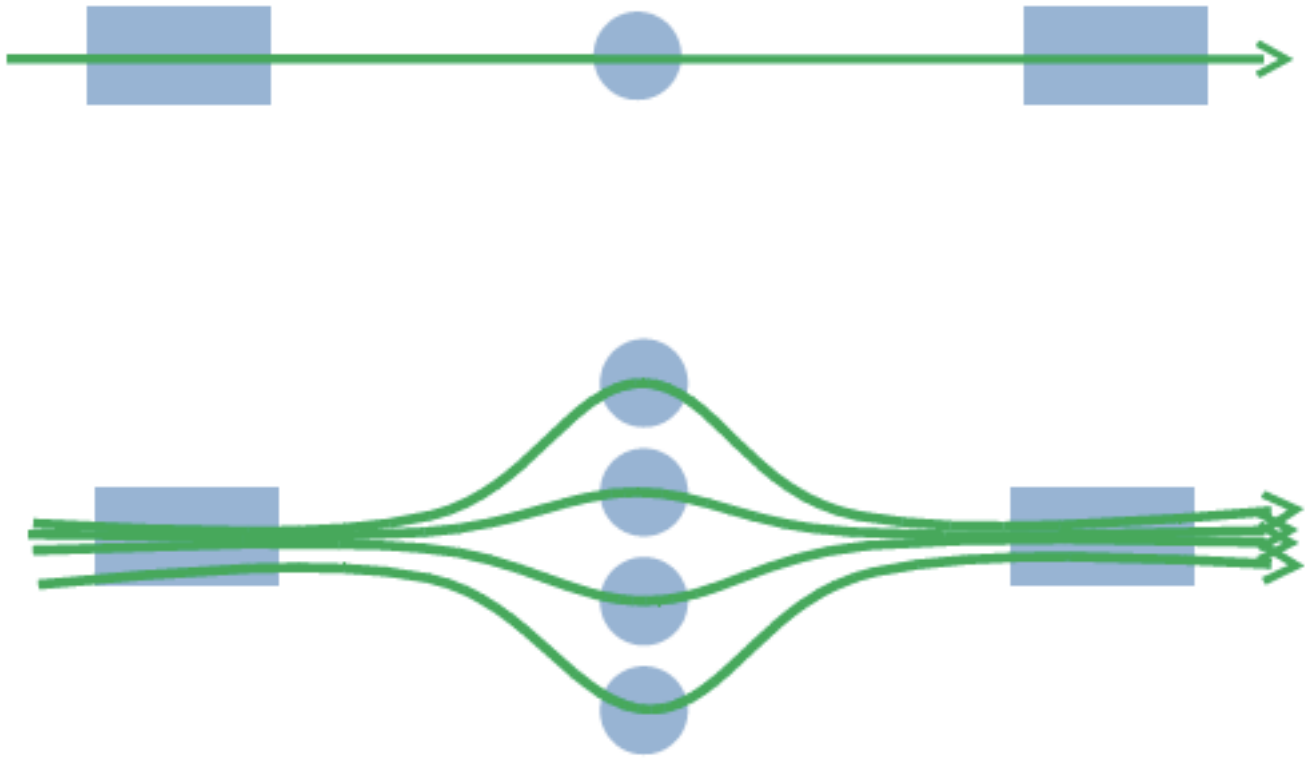


1.7 Gbps

Rule 1: one core per port

Rule 2: one core per packet

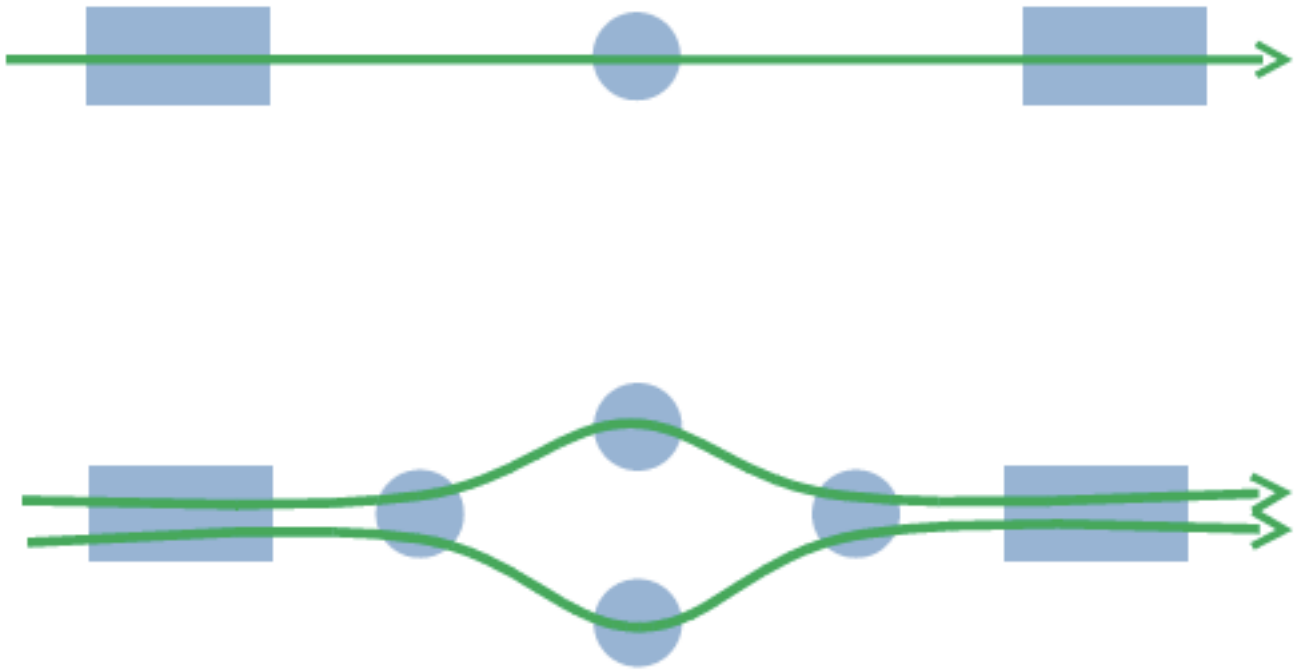
Problem #3: Queue Access



Rule 1: one core per port

Rule 2: one core per packet

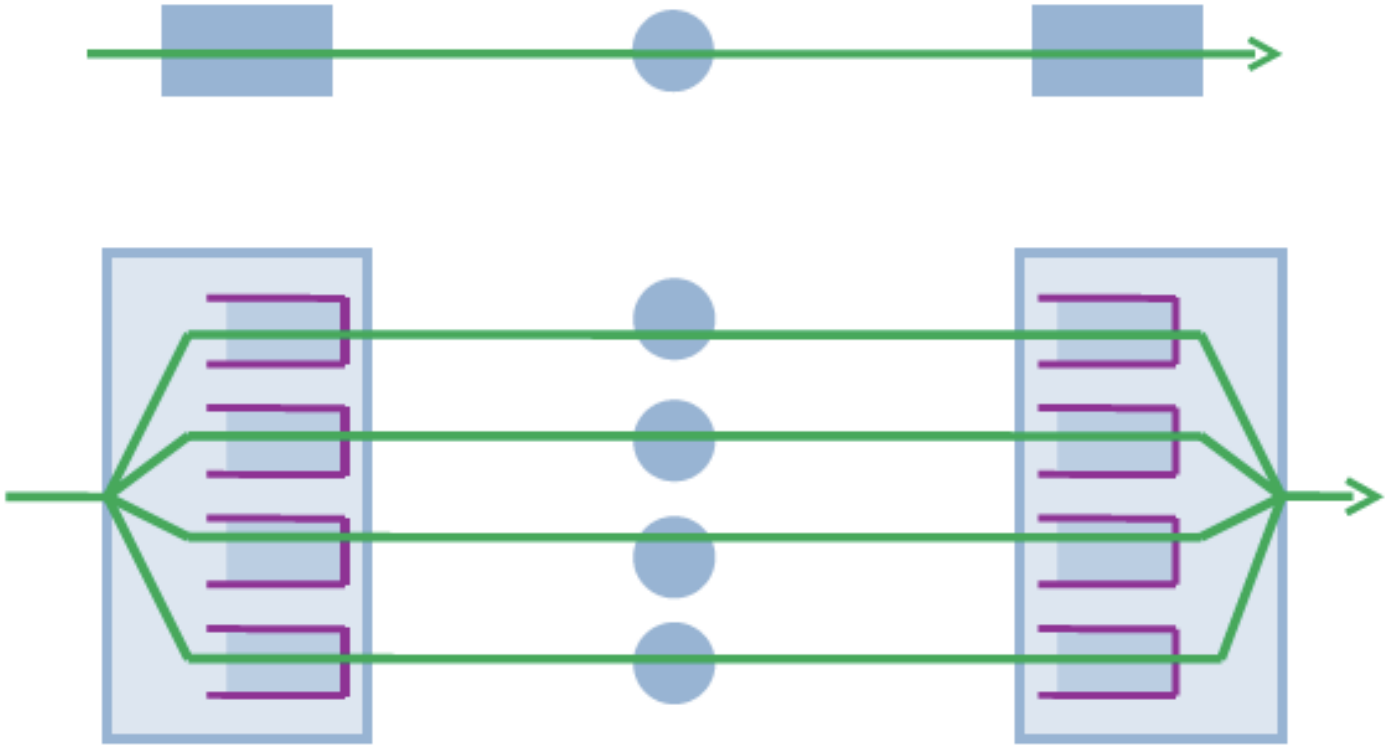
Problem #3: Queue Access



Rule 1: one core per port

Rule 2: one core per packet

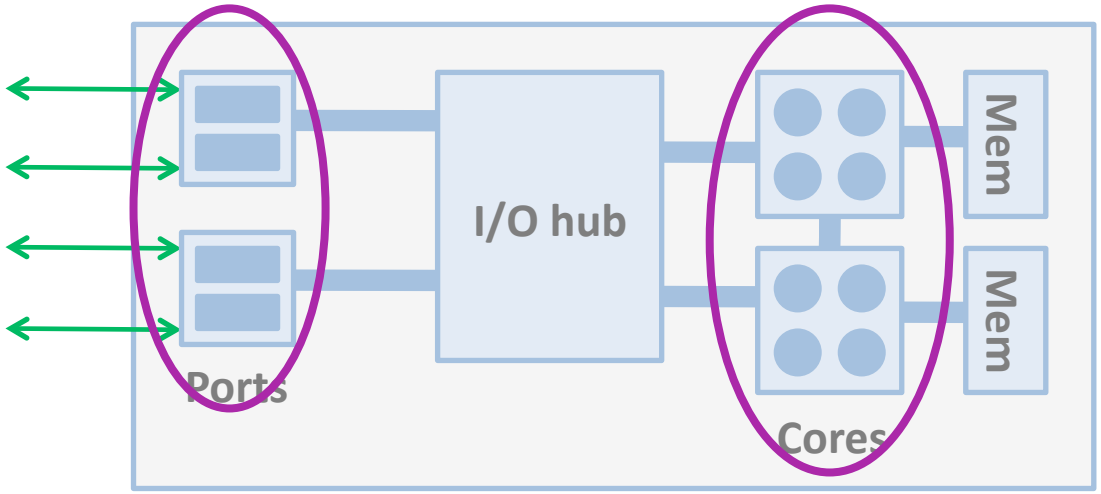
Solution: Multi-Queue NICs

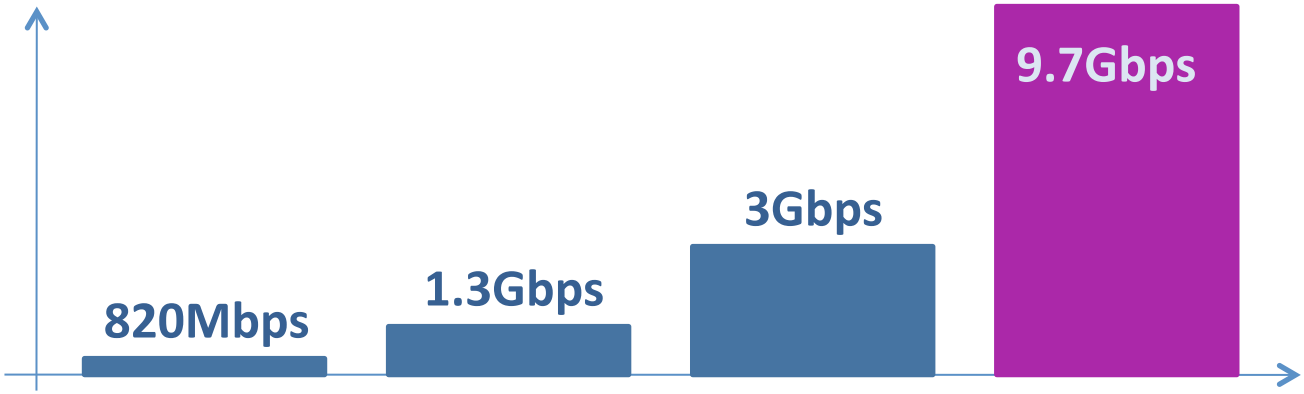


Rule 1: one core per port

Rule 2: one core per packet

Single-Server Performance





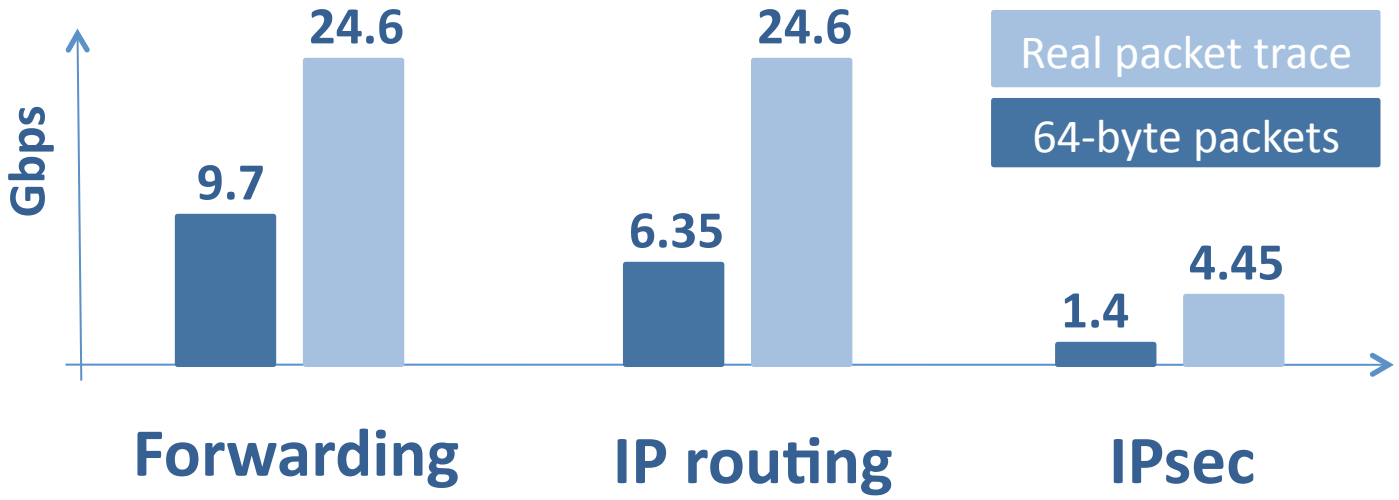
Recap

- **State-of-the art hardware**
  - » NUMA architecture
  - » multi-queue NICs
- **Wrote NIC driver**
  - » batching
  - » lock-free queue access
- **Careful queue-to-core allocation**
  - » one core per queue
  - » one core per packet

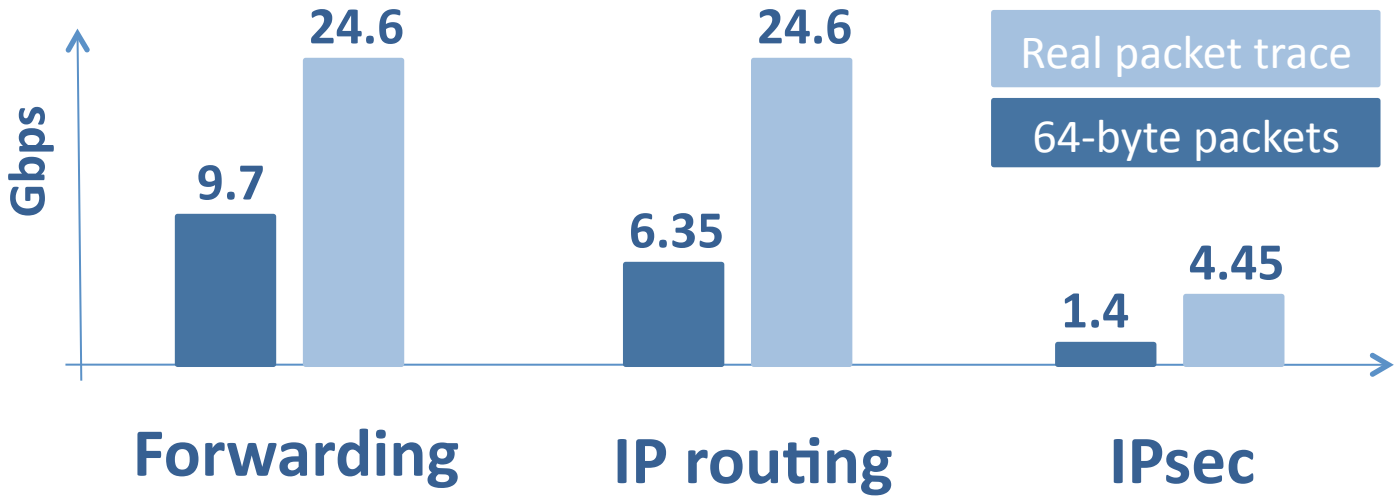
Outline

- Interconnect
- Server optimizations
- **Performance**
- An application
- Conclusions

Single-Server Performance



Feasible Router Line Rate

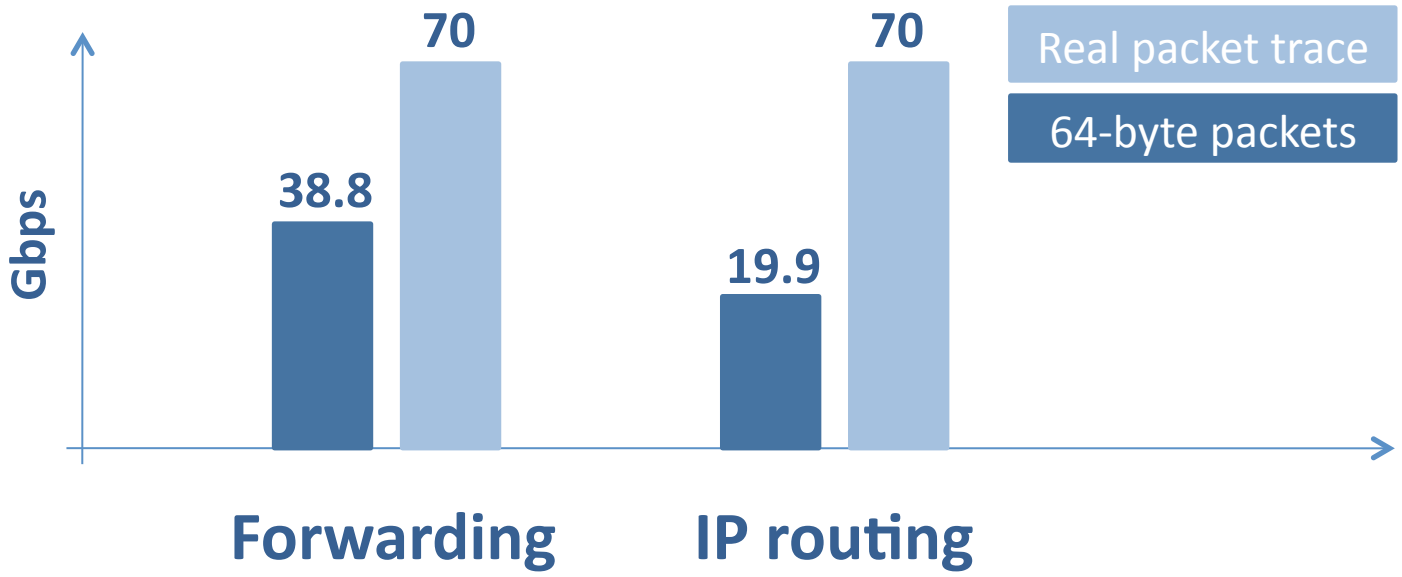


- Per-server processing rate: 2*R* – 3*R*
- Real packet mix: *R* = 8 – 12 Gbps
- Small packets: *R* = 2 – 3 Gbps

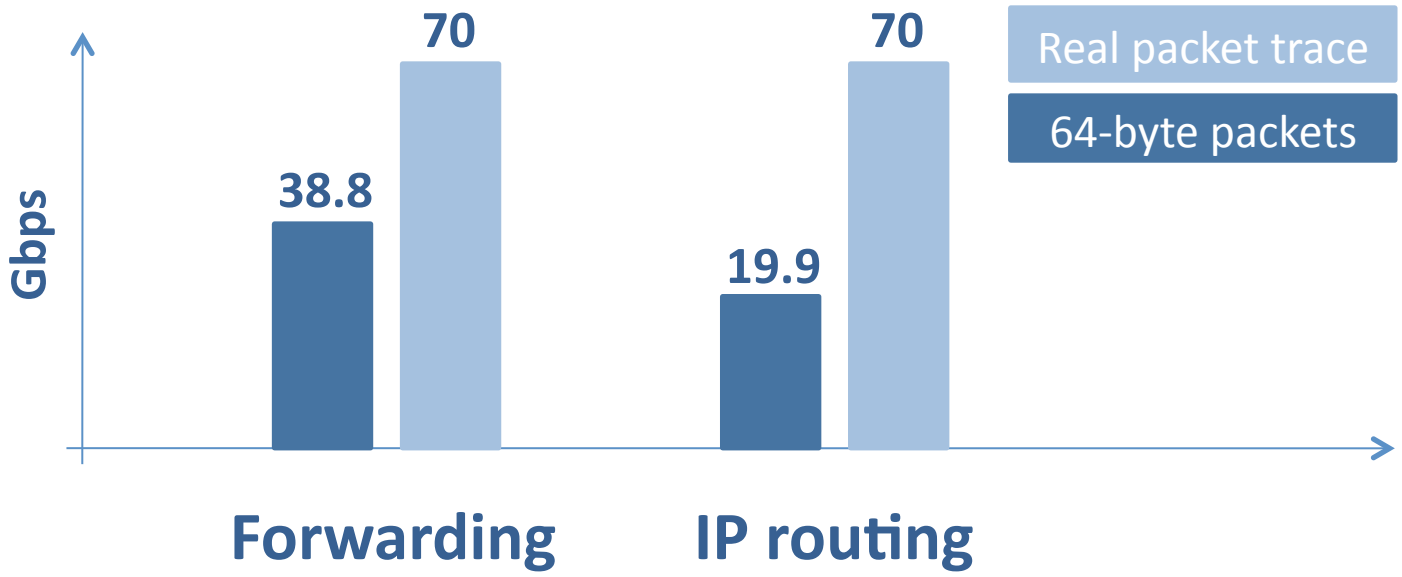
Bottlenecks

- **Small packets: CPU**
  - » buses far from saturation
- **Real packet-size mix: none**
  - » limited PCIe lanes
  - » only for the prototype
- **Expected evolution?**
  - » next Nehalem: 4 x 8 = 32 cores
  - » 4 – 8 PCIe2.0 slots

Projected Single-Server Performance



Projected Router Line Rate



- Real packet mix: *R* = 23 – 35 Gbps
- Small packets: *R* = 6.5 – 10 Gbps

RB4 Prototype

- **N = 4 external ports**
  - » 1 server per port
  - » full mesh
- **Real packet mix: 4 x 8.75 = 35 Gbps**
  - » expected *R* = 8 – 12 Gbps
- **Small packets: 4 x 3 = 12 Gbps**
  - » expected *R* = 2 – 3 Gbps

What About Packet Order?

- TCP cuts sending rate by ½ if packets are ever reordered by more than 3 positions
- But VLB sprays packets randomly across intermediate nodes!
- RouteBricks' partial solution:
  - » Assign packets on same flow to same receive queue
  - » During any 100 ms interval, VLB forwards packets from same flow to same next hop
- 0.15% of packets reordered with this mechanism; 5.5% without it

Latency

- One server: 24 microseconds
- Three servers: 66.4 microseconds

RouteBricks Summary

- **RouteBricks: fast software router**
  - » Valiant Load-Balanced cluster of commodity servers
- **Programmable with Click**
- **Performance:**
  - » Easily R = 1Gbps, N = 100s
  - » R = 10Gbps with next-generation servers
- **Programming model for more complex functionality?**