Advanced Computer Networks

Network Function Virtualization

Lin Wang

Period 2, Fall 2021

VU VRIJE UNIVERSITEIT AMSTERDAM

Course outline

Warm-up

- Introduction (history, principles)
- Networking basics
- Networking data structures and algorithms
- Network transport

Data centers

- Data center networking
- Data center transport

Programmability

- Software defined networking
- Network automation
- **Network function virtualization**
- Programmable data plane

Application

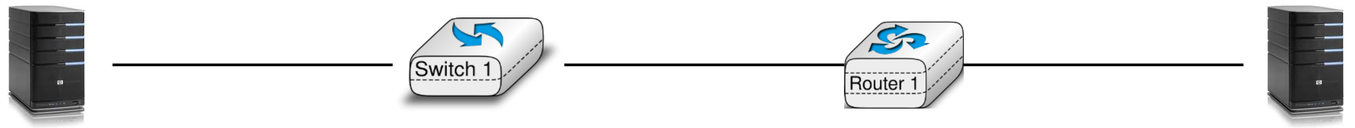
- Network monitoring
- In-network computing
- Machine learning for networking

Learning objectives

How to **simplify the programming** of virtual network functions?

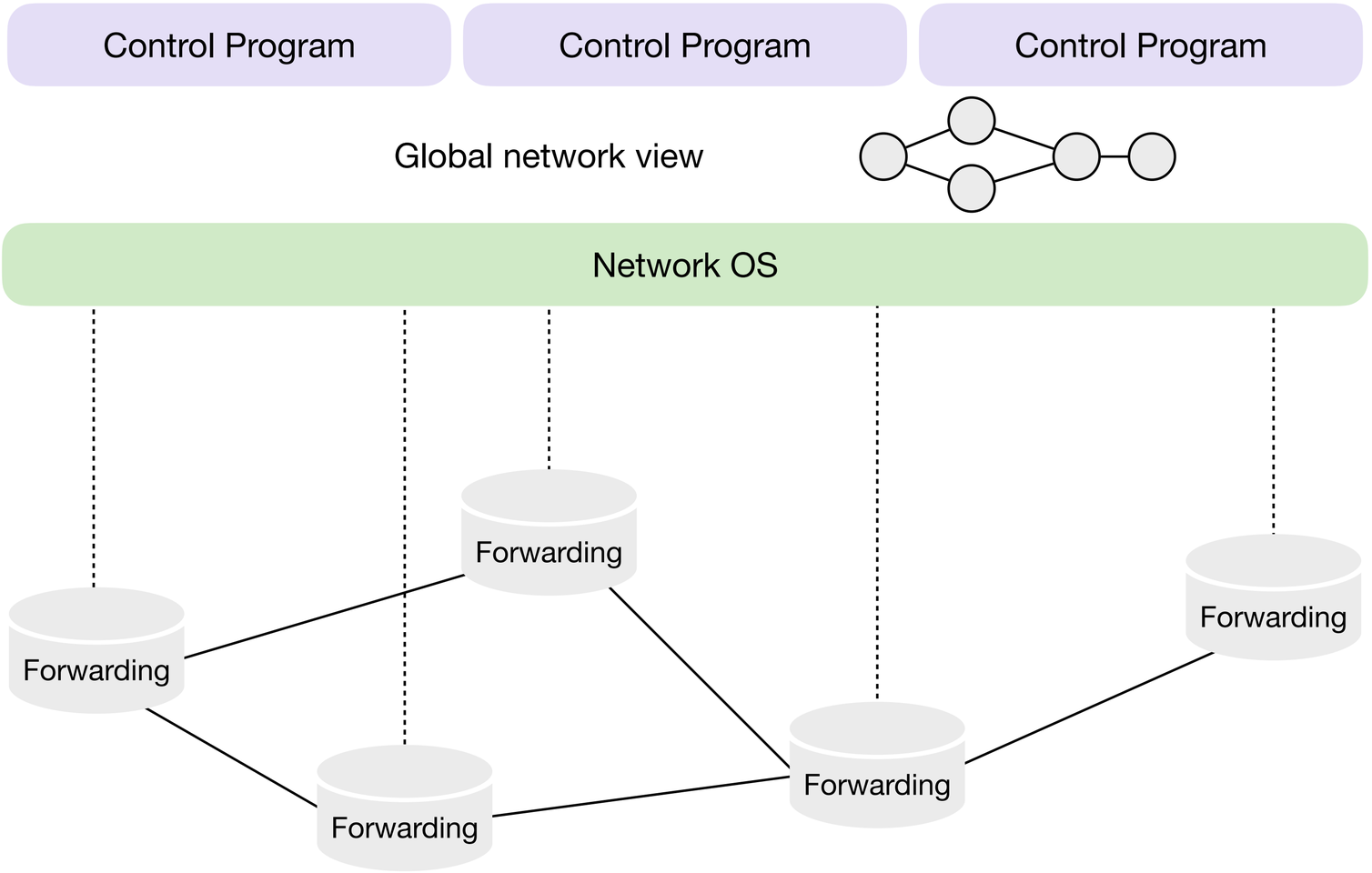
How to achieve **high performance** for virtualized network functions?

Network data plane



Application			Application
Transport			Transport
Network		Network	Network
Datalink	Datalink	Datalink	Datalink
Physical	Physical	Physical	Physical

Network control plane



Middleboxes

Essential to network operators

Security

- Firewalls, IDSes, traffic scrubbers, NATs

Traffic shaping

- Rate limiters, load balancers

Performance

- Traffic accelerators, caches, proxies


carrier-grade NAT


ad insertion


IDS


load balancer


session border controller


firewall


transcoder

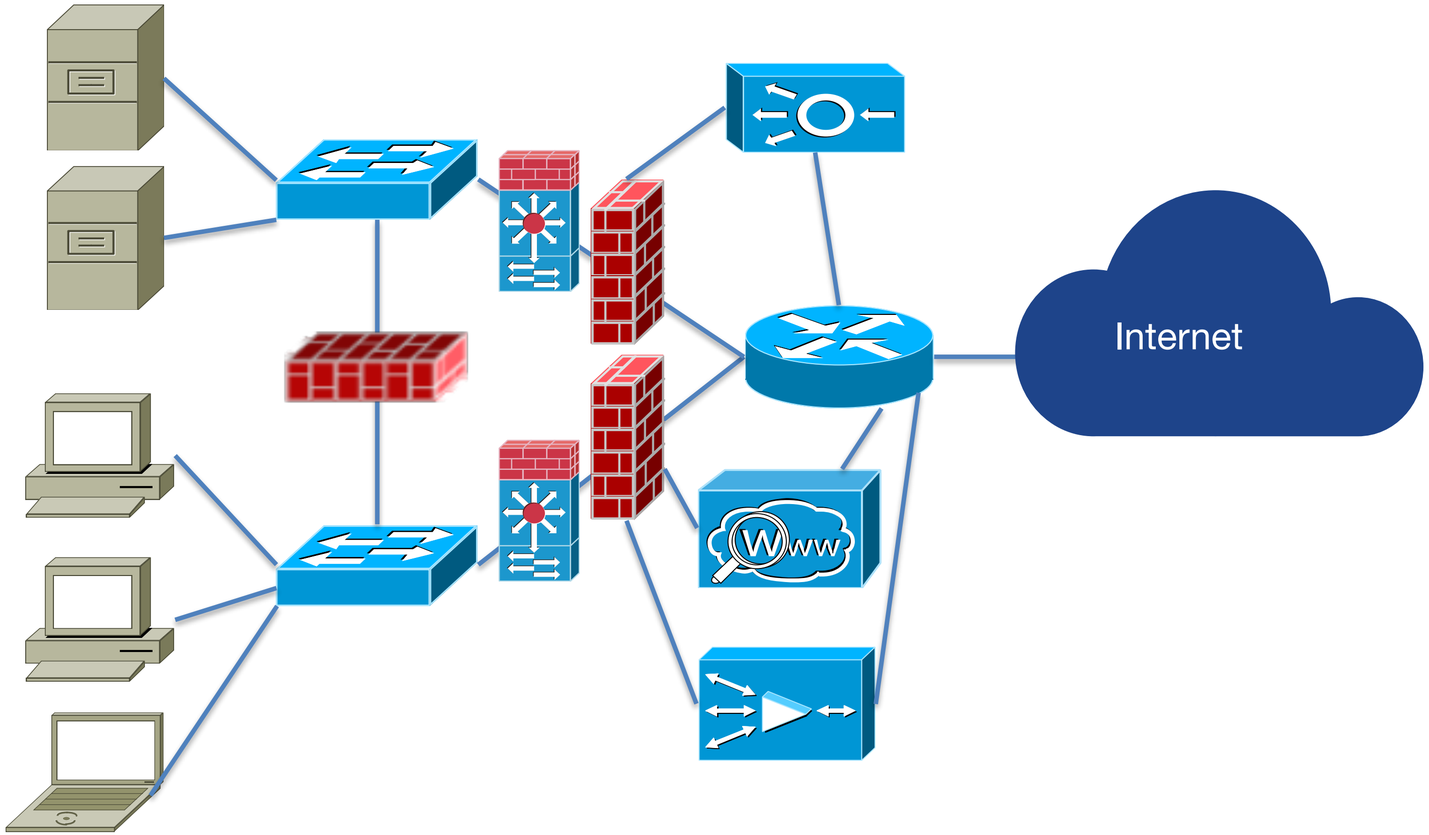

DDoS protection


DPI

QoE monitor


WAN accelerator

The real network is more complex than you think



Challenges with middleboxes

Buying and managing middlexboxes are **expensive** since these are proprietary

Deploying and managing middleboxes are **complex** in terms of wiring the chaining

Introducing new features is **slow**, requiring the purchase of new hardware with software

Not scalable upon demands shifts: devices need to be physically placed or removed

Vendor lock-in: devices are proprietary, raising innovation barriers

Alternative solutions and tradeoffs

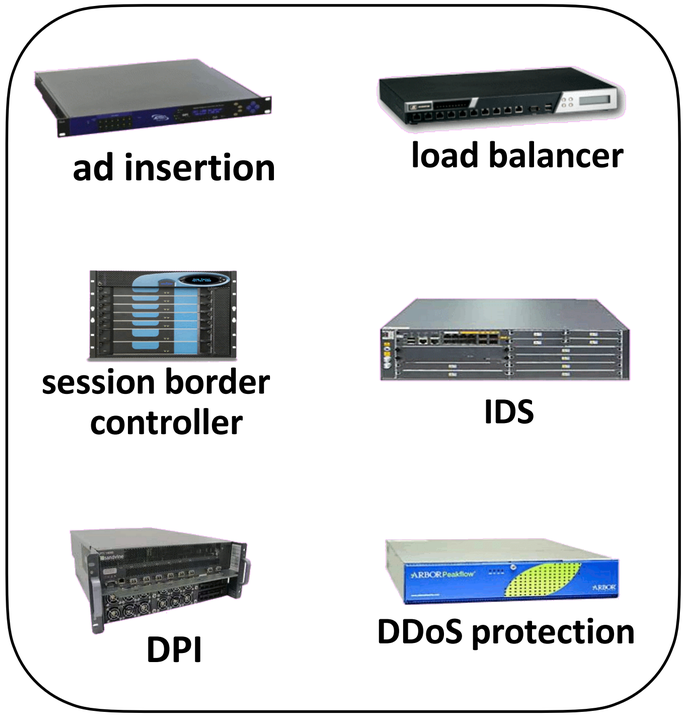
Commodity server

Specialized boxes

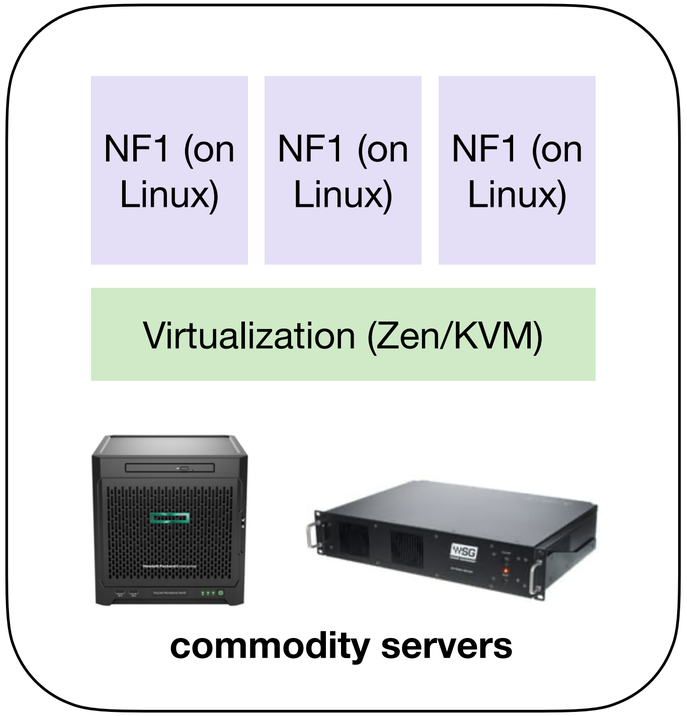
High extensibility
Easy management
Low cost
Low performance

Low extensibility
Complex management
High cost
High performance

Network function virtualization (NFV)



Current middlebox approach



Network function virtualization

Benefits with NFV

Hardware **sharing** across multiple tenants/network functions

Cost saving via reduced hardware/ power costs through consolidation

More **flexible management** using software

Safe to try **new features** on an operational network/platform

Question: How can we implement a network function conveniently and achieve high performance?

How to simplify the programming of virtual network functions?

Network functions are complex to implement

An IP router involves at least the following components

Packet classification	ARP handling	IP header operations
IP fragmentation	IP address lookup	ICMP processing

Do you still remember the fundamental principle for managing complexity?

Managing complexity with modularity

Machine languages: no abstractions

- Hard to deal with low-level details
- Mastering complexity is crucial

High-level languages: operating systems and other abstractions

- File systems, virtual memory, abstract data types...

Modern languages: even more abstractions

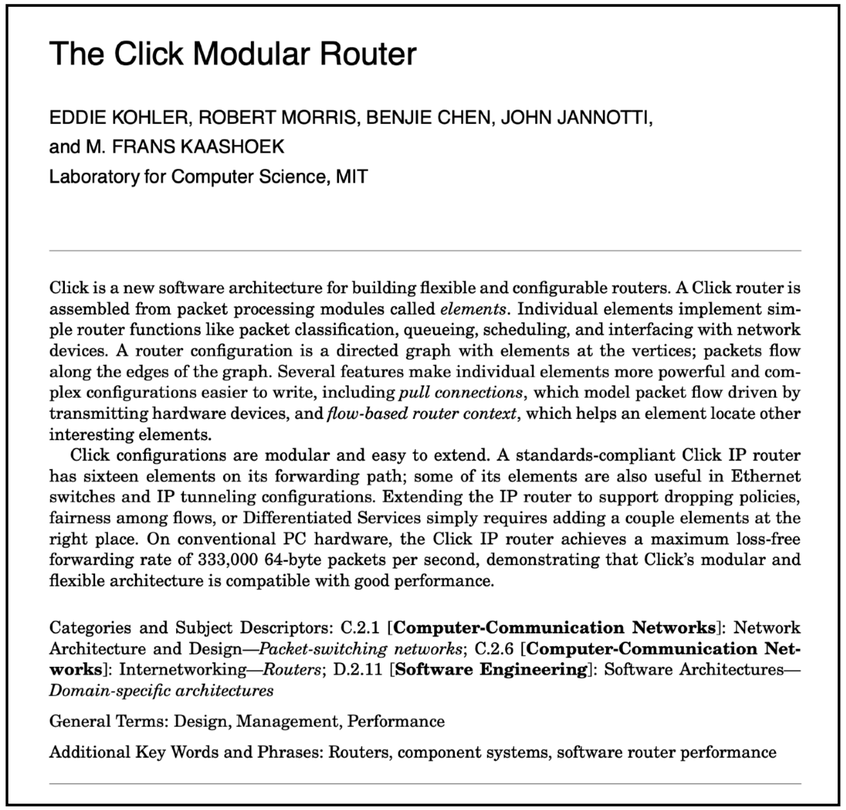
- Object oriented, garbage collection...

"Modularity based on abstractions is the way things get done!"

Barbara Liskov
(MIT, ACM Turing Award 2008, pioneer in programming languages, operating systems, distributed computing)

Click

A modular architecture for programming network functions



The Click Modular Router

EDDIE KOHLER, ROBERT MORRIS, BENJIE CHEN, JOHN JANNOTTI,
and M. FRANS KAASHOEK
Laboratory for Computer Science, MIT

Click is a new software architecture for building flexible and configurable routers. A Click router is assembled from packet processing modules called *elements*. Individual elements implement simple router functions like packet classification, queueing, scheduling, and interfacing with network devices. A router configuration is a directed graph with elements at the vertices; packets flow along the edges of the graph. Several features make individual elements more powerful and complex configurations easier to write, including *pull connections*, which model packet flow driven by transmitting hardware devices, and *flow-based router context*, which helps an element locate other interesting elements.

Click configurations are modular and easy to extend. A standards-compliant Click IP router has sixteen elements on its forwarding path; some of its elements are also useful in Ethernet switches and IP tunneling configurations. Extending the IP router to support dropping policies, fairness among flows, or Differentiated Services simply requires adding a couple elements at the right place. On conventional PC hardware, the Click IP router achieves a maximum loss-free forwarding rate of 333,000 64-byte packets per second, demonstrating that Click's modular and flexible architecture is compatible with good performance.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Packet-switching networks*; C.2.6 [**Computer-Communication Networks**]: Internetworking—*Routers*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

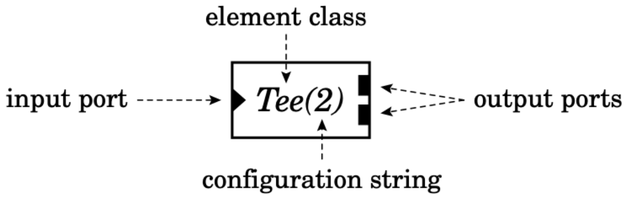
General Terms: Design, Management, Performance

Additional Key Words and Phrases: Routers, component systems, software router performance

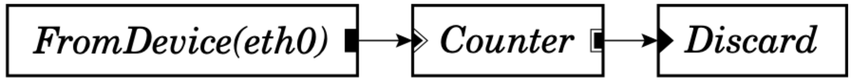
ACM SOSP 1999

Basic concepts

Element

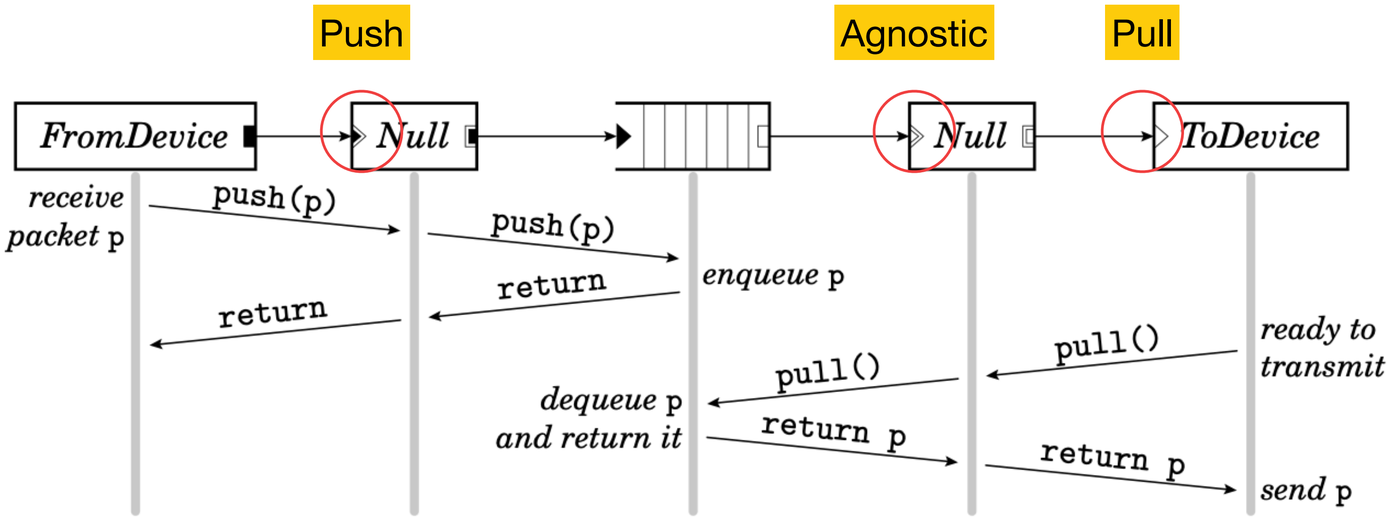


Connection and **configuration**



Connection

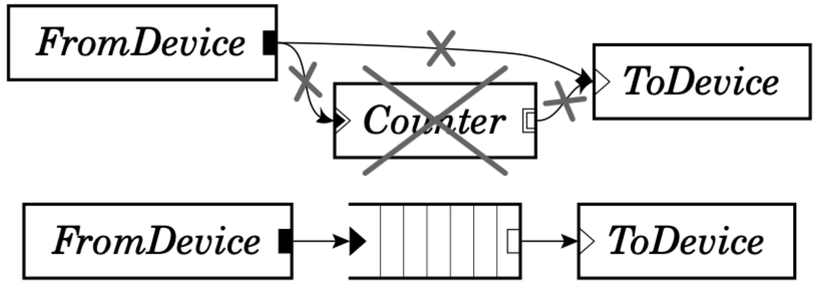
Three types of ports: push, pull, and agnostic



Push connections push packets to the downstream element, while pull connections pull packets from upstream elements. Agnostic ports are decided when they are connected to the other side.

Connection

Push/pull connection constraints



Rule 1: A connection must have the same type on the two ends

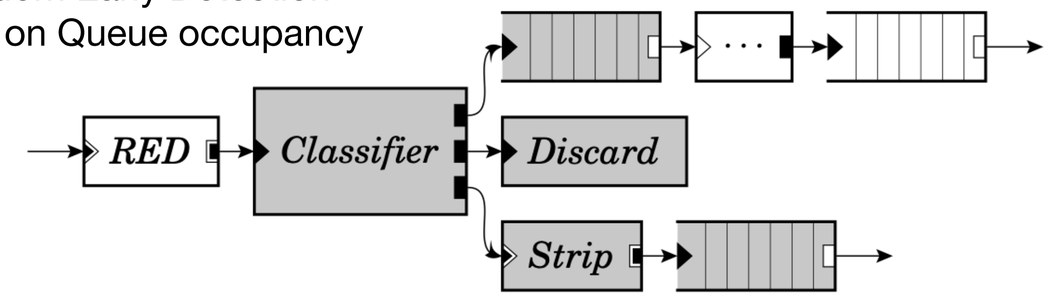
Rule 2: A port must have only one connection

Rule 3: An element must not have different types on its input/output ports if the element processes packets immediately (counterexample: Queue element)

Method sharing across elements

Use **flow-based router context** to discover an element B for using B's method interface

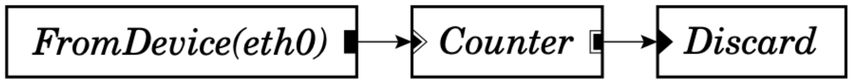
Random Early Detection
based on Queue occupancy



During configuration initialization, the system issues questions like:
"If I were to emit a packet on my second output, and it stopped at the first Queue it encountered, where might it stop?"

Click language

The interface for programmers to specify elements, connections, and configurations




```
// Declare three elements ...
src :: FromDevice(eth0);
ctr :: Counter;
sink :: Discard;
// ... and connect them together
src -> ctr;
ctr -> sink;

// Alternate definition using syntactic sugar
FromDevice(eth0) -> Counter -> Discard;
```

The language is simple and wholly declarative: it specifies what elements to create and how they should be connected, not how to process packets procedurally.

Click language

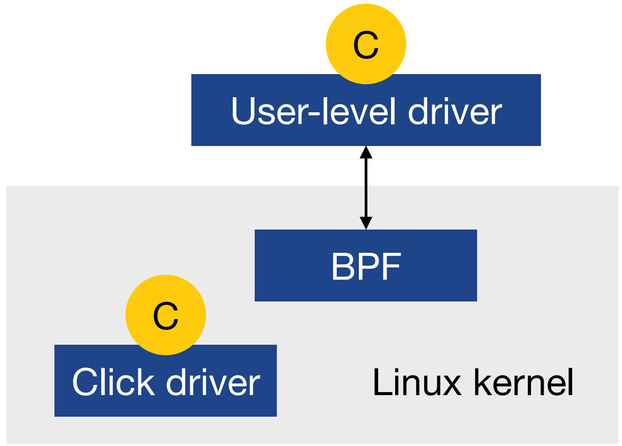
Implementing a Click element

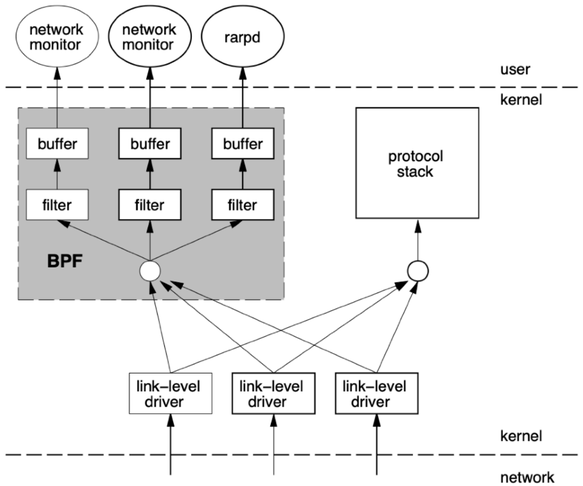
```
class NullElement: public Element { public:
  NullElement()                          { add_input(); add_output(); }
  const char *class_name() const         { return "Null"; }
  NullElement *clone() const             { return new NullElement; }
  const char *processing() const         { return AGNOSTIC; }
  void push(int port, Packet *p)         { output(0).push(p); }
  Packet *pull(int port)                 { return input(0).pull(); }
};
```

The complete implementation of a do-nothing element: Null passes packets from its single input to its single output unchanged.

Click installation

How does a Click router get installed on Linux?





Each element can install a number of **handlers** which are access points for user interaction. They appear to users a files in Linux's **`/proc`** file system.

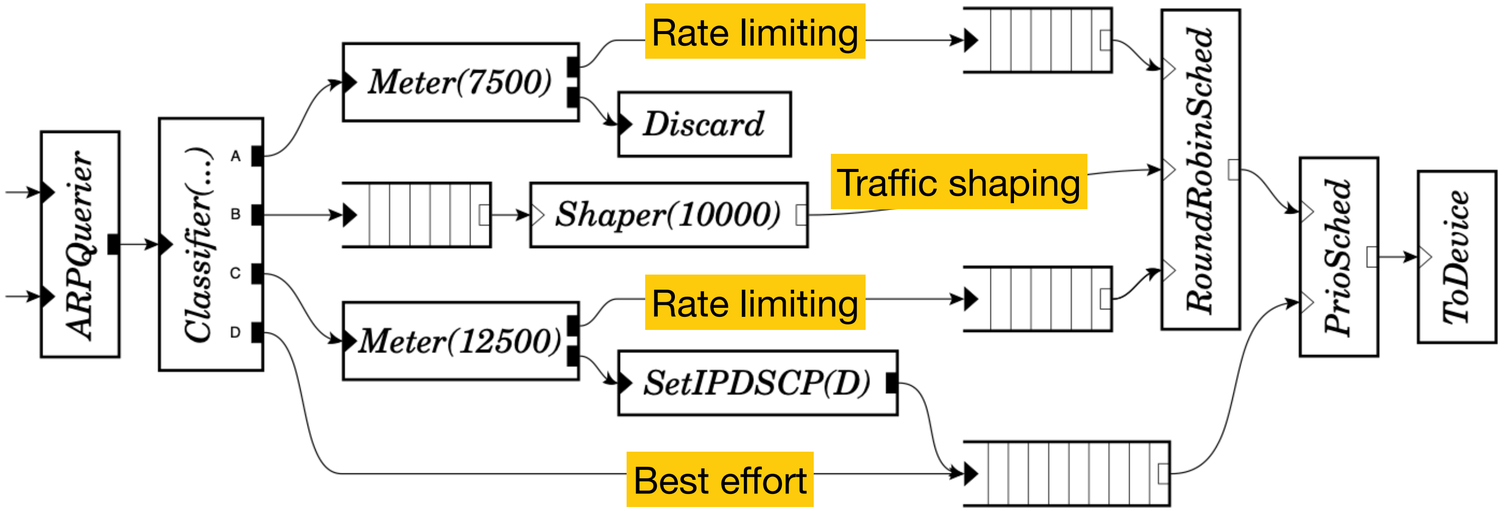
A user can write a new configuration file and install it with a **hot-swapping** option that allows the old configuration to run before the new configuration is in place.

https://github.com/kohler/click

https://ebpf.io

Click example

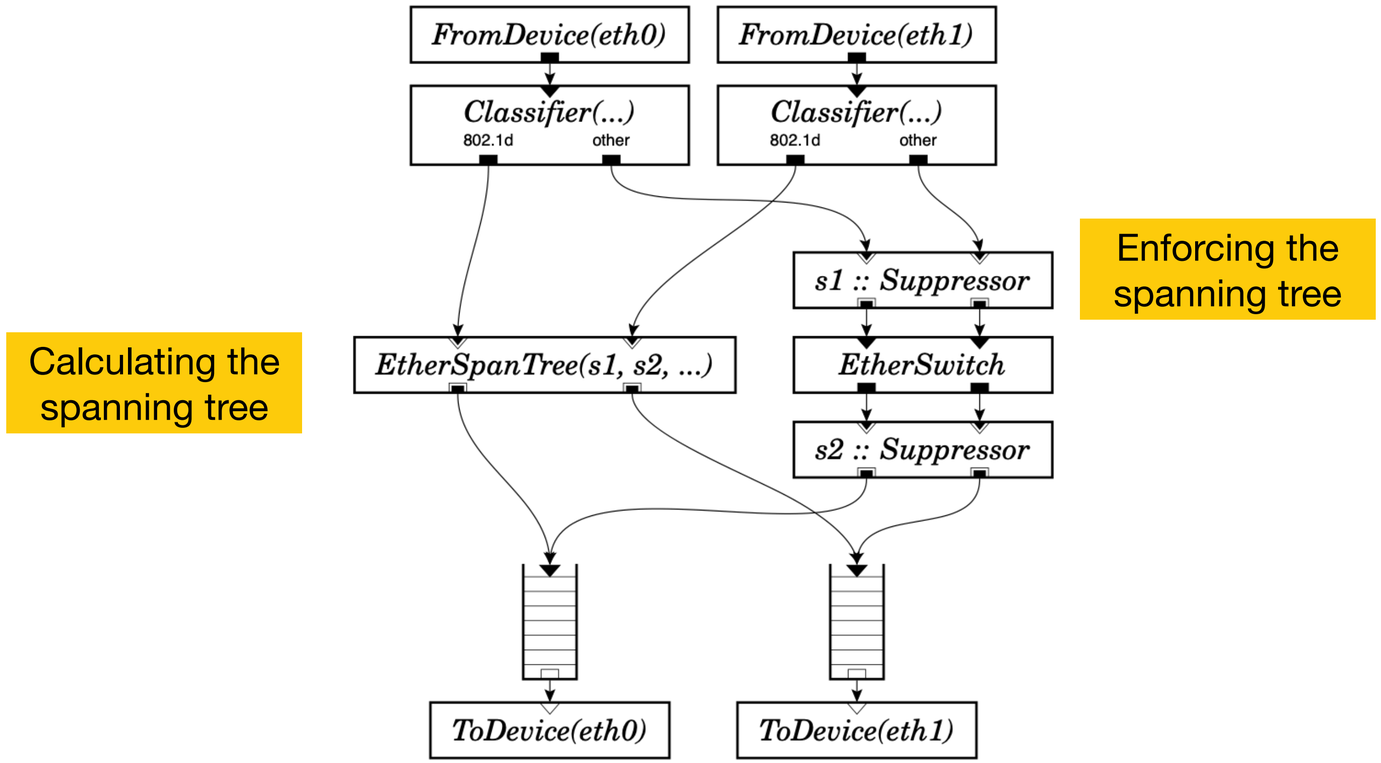
Traffic conditioning



Differentiated Services Code Point (DSCP)

Click example

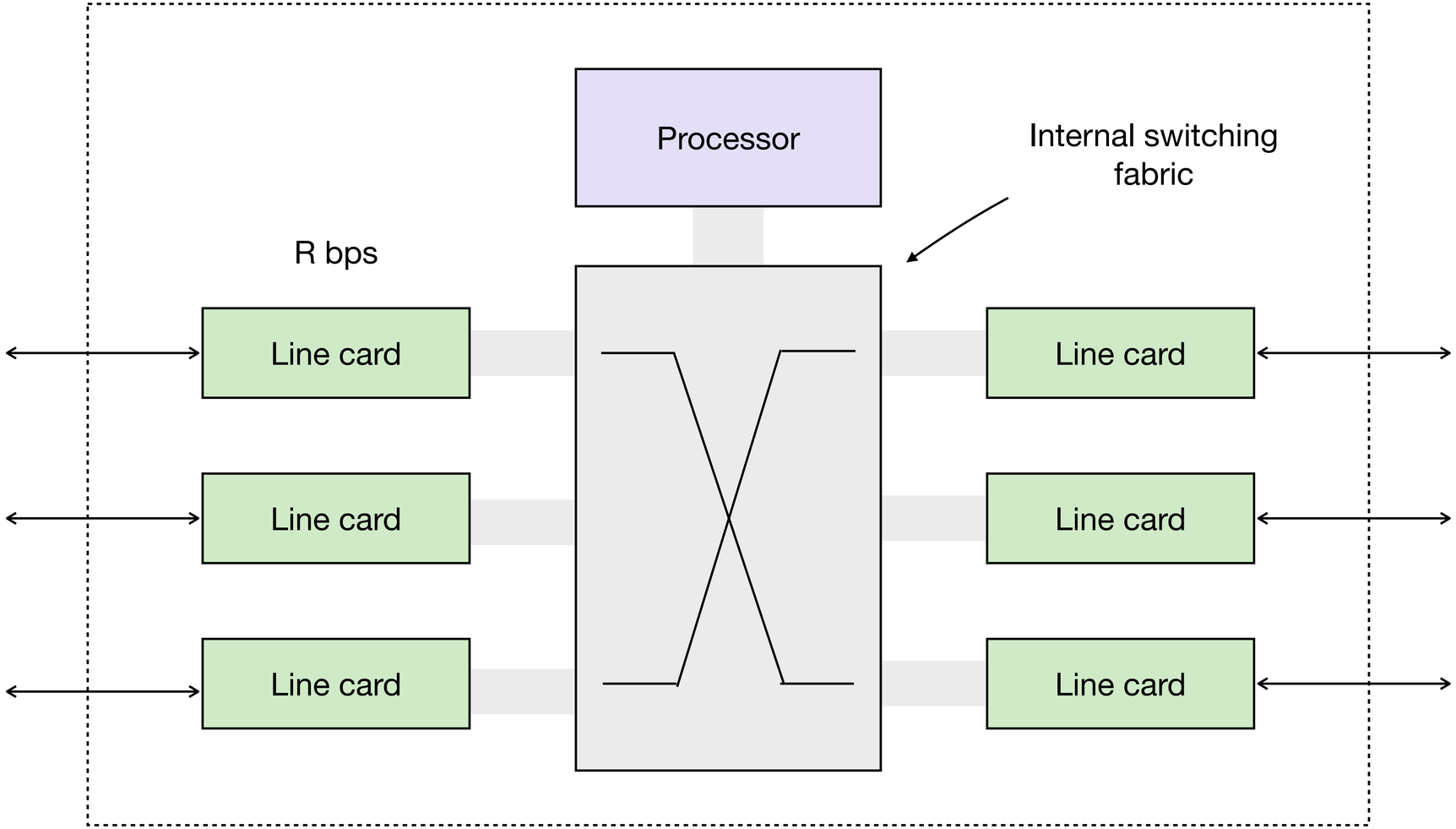
Ethernet switch



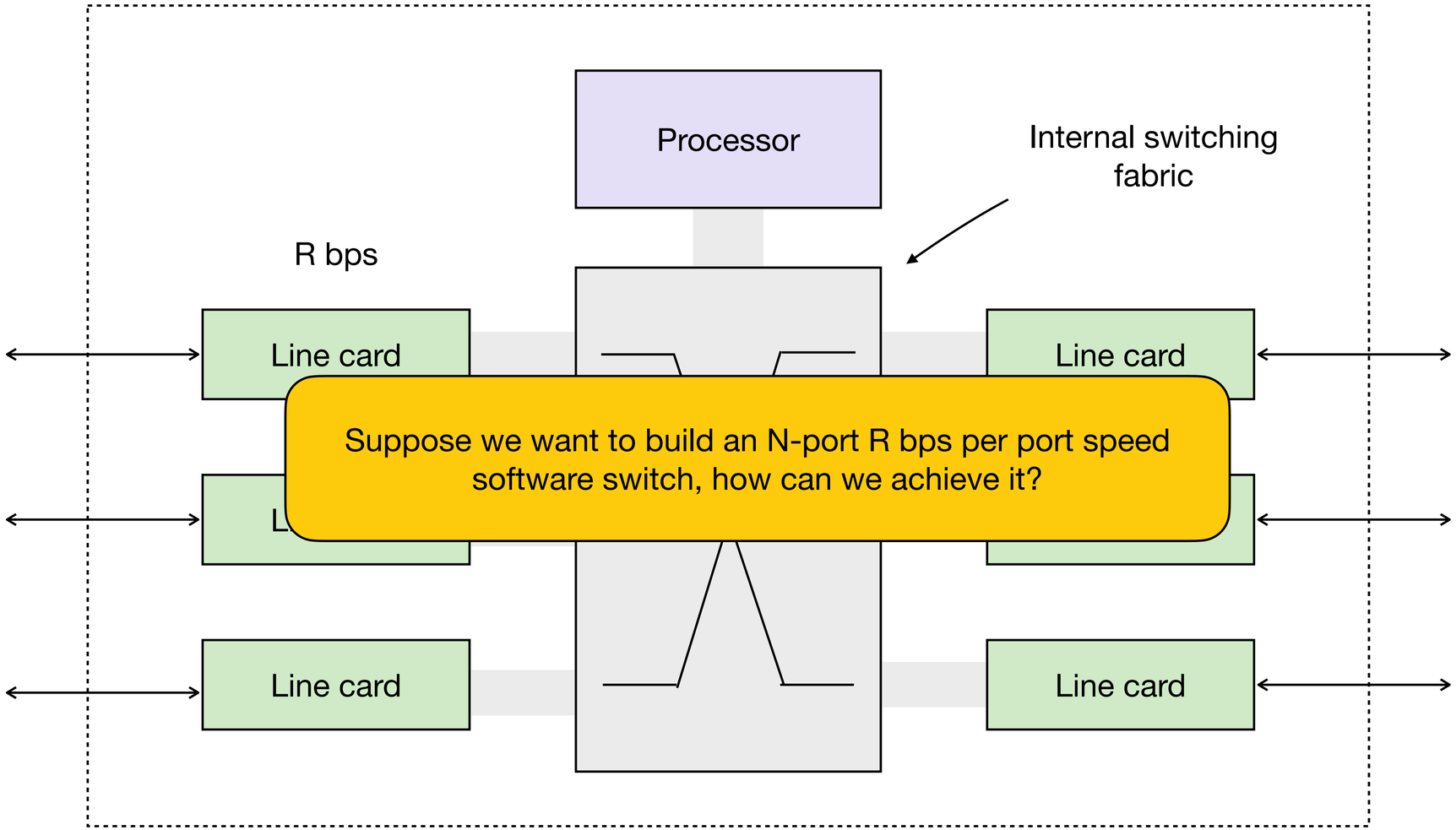
The suppressors cannot be found by the EtherSpanTree element with flow-based router context, so they must be manually specified by users.

How to achieve high performance for virtualized network functions?

Traditional router architecture



Traditional router architecture



The case with a single server

R = 1, 1.25, 10 Gbps

N=10s - 1000s

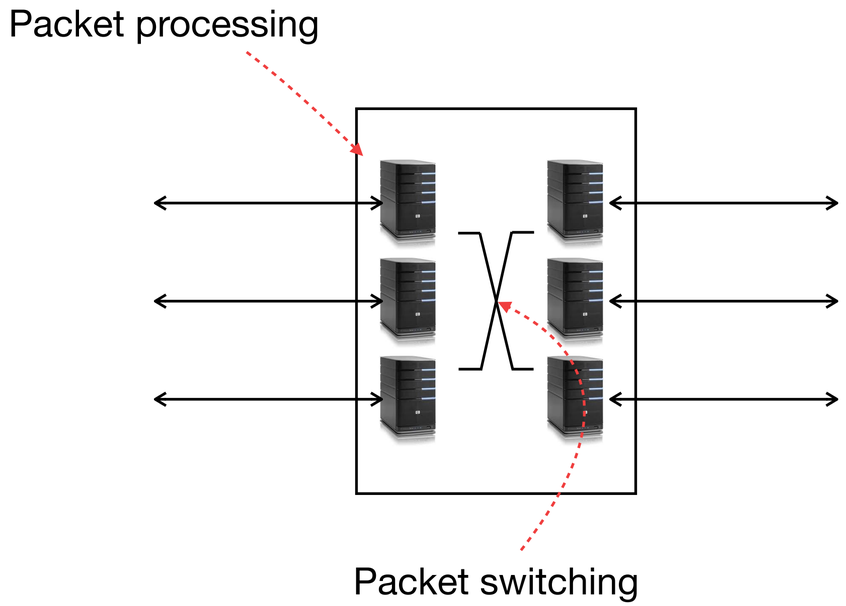
N*R bps

(required: up to few Tbps, achievable: 1-5Gbps)

What are possible approaches for scaling up?

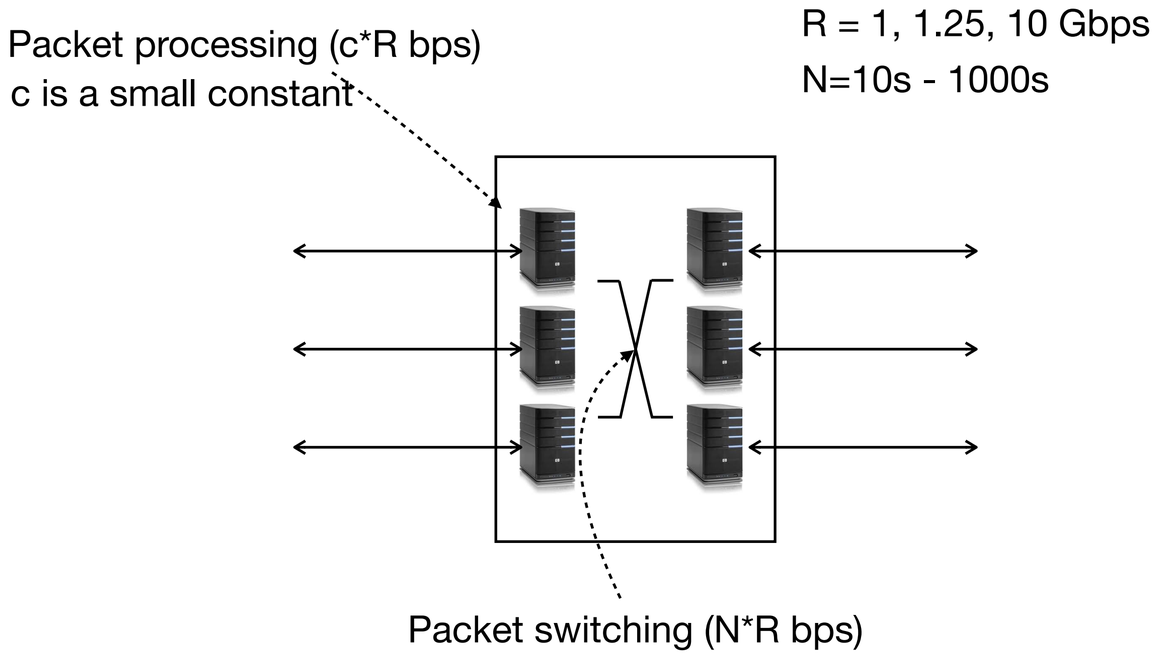
Scaling up with parallelism

Two basic functionalities: packet processing (classification, lookup), and packet switching



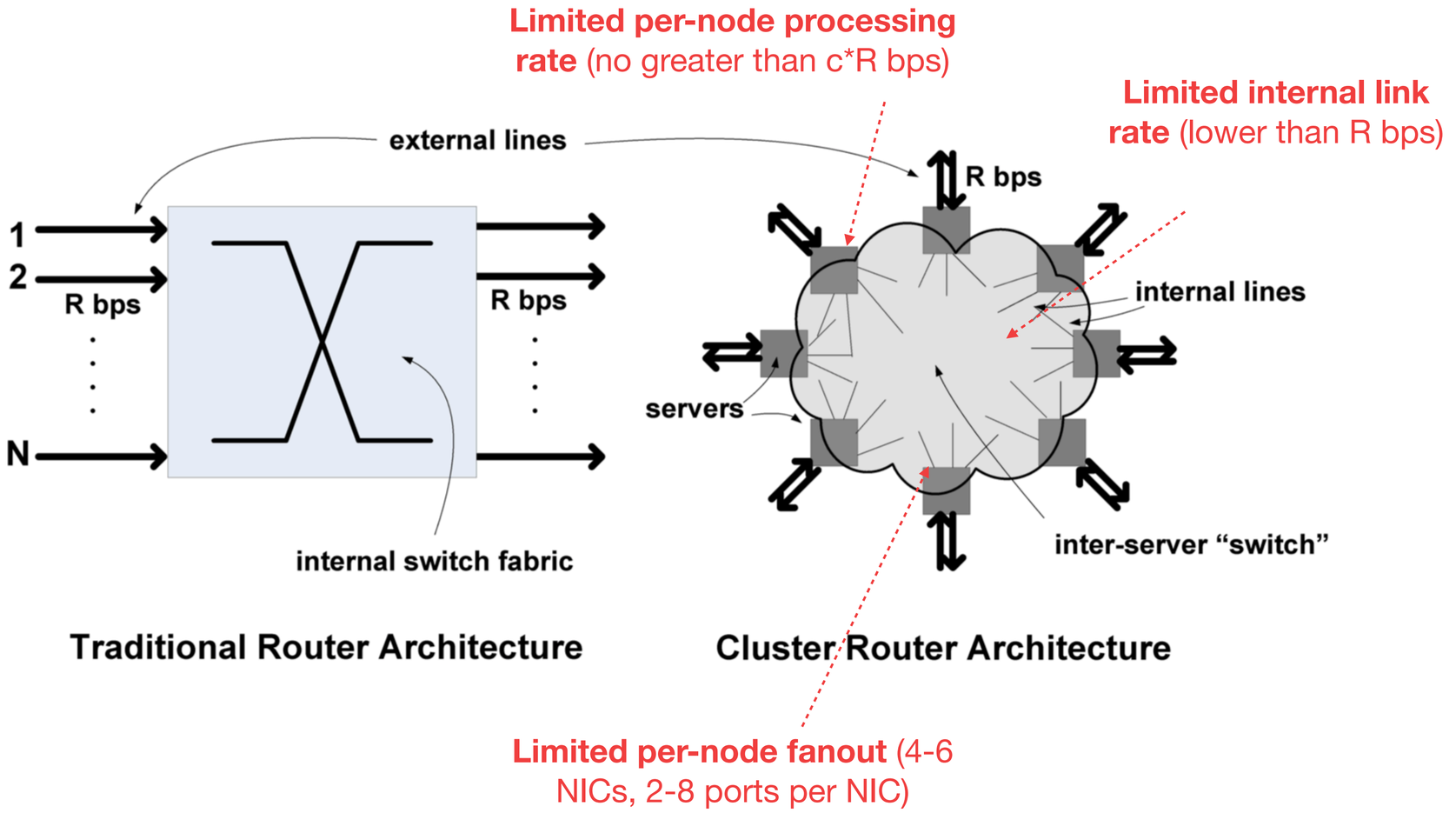
Scaling up with parallelism

Two basic functionalities: packet processing (classification, lookup), and packet switching



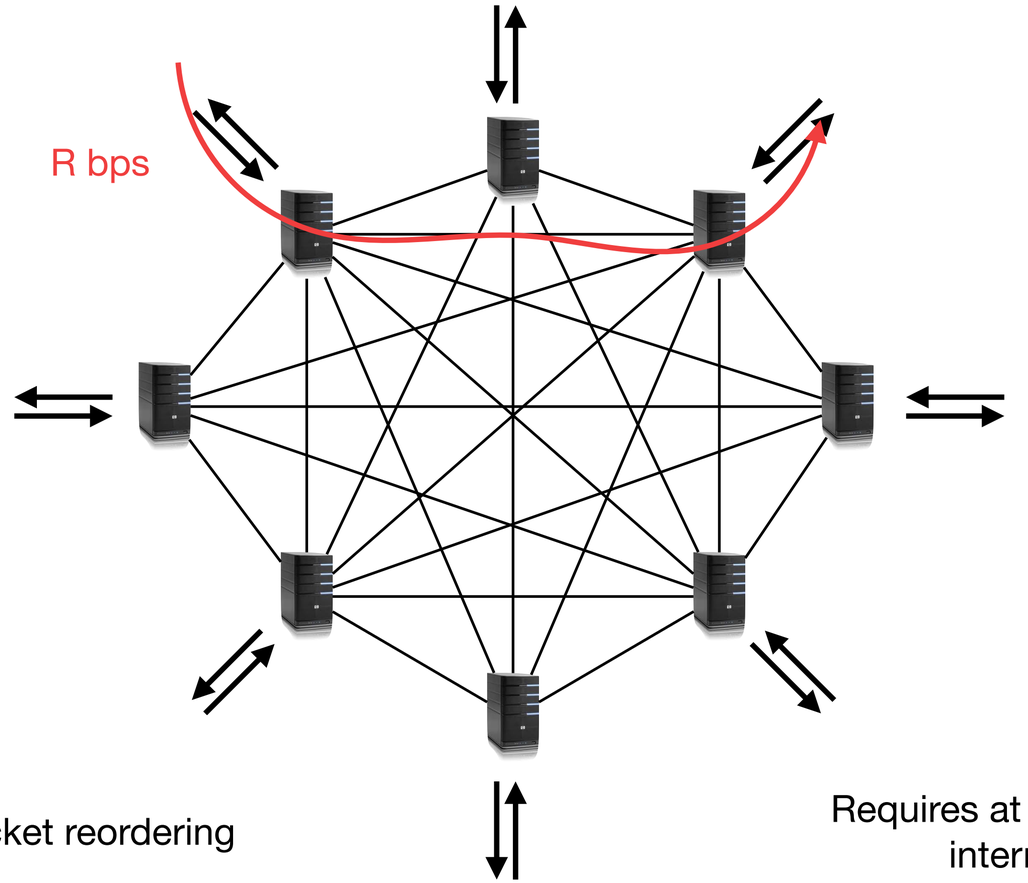
Require distributed solutions to scale

A software router built from a cluster of servers



Routing with a single path

Forward the packet directly from input server to output server

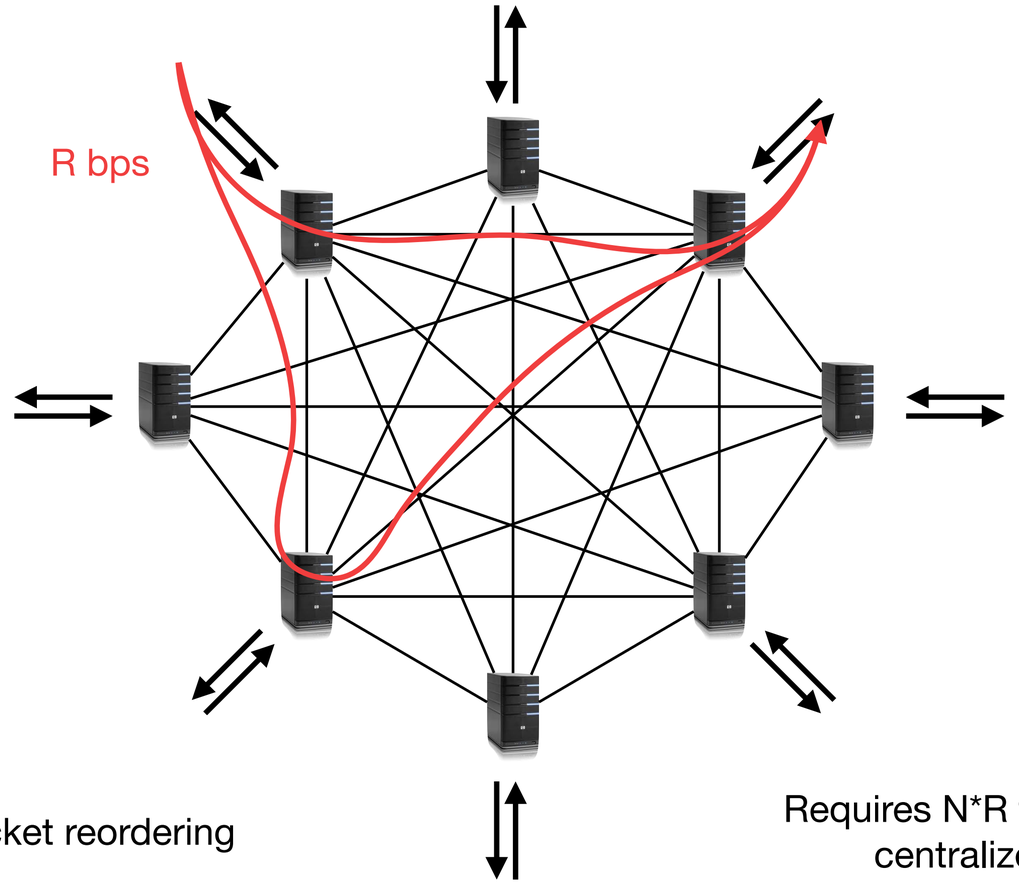


Simple, no packet reordering

Requires at least R bps for internal links

Routing with dynamic single paths

Dynamically allocate a path for a source-destination pair

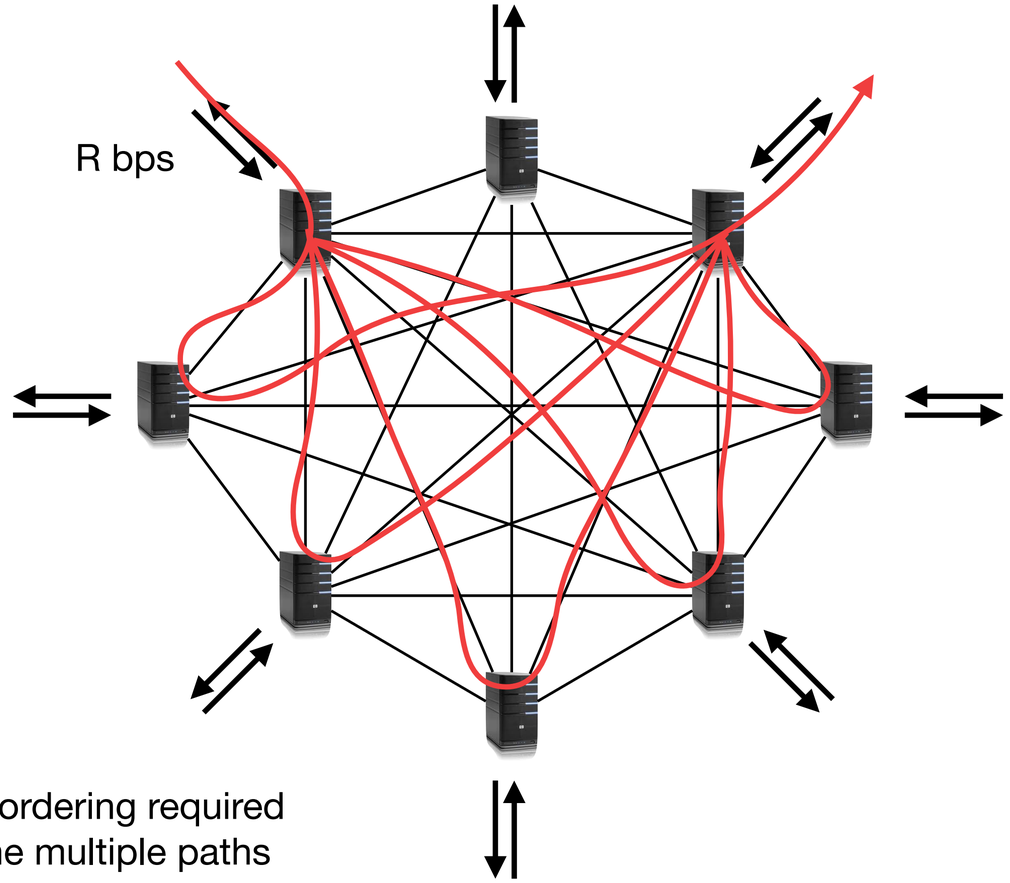


Simple, no packet reordering

Requires N*R throughput for the centralized scheduler

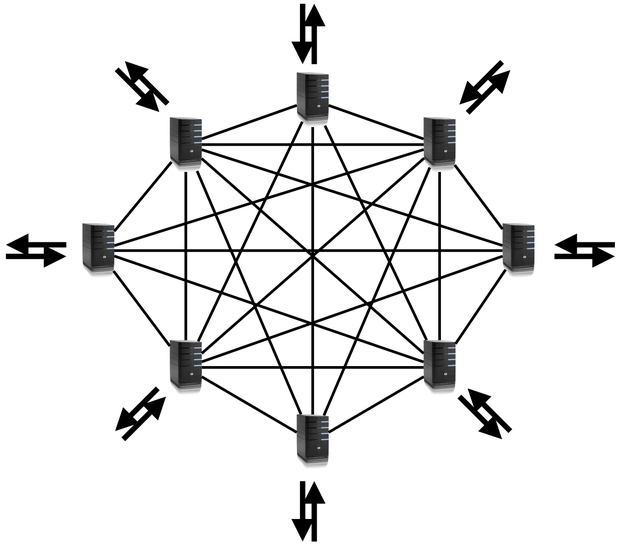
Routing with perfect load balancing

Traffic between a given input and output port is spread across multiple paths

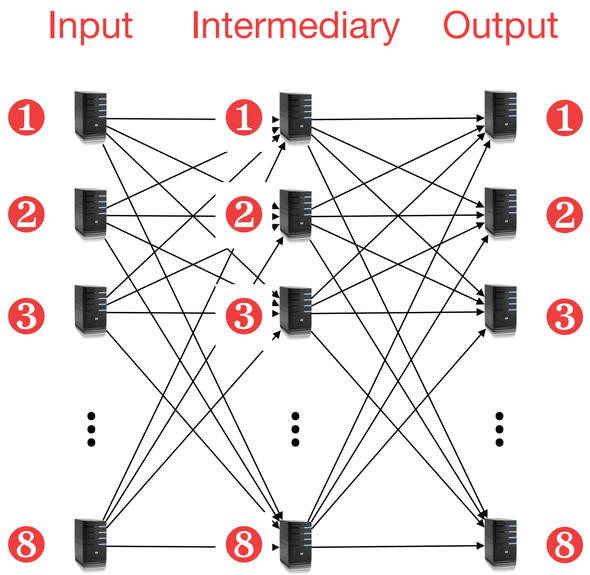


Packet reordering required due to the multiple paths

Valiant load balancing (VLB)



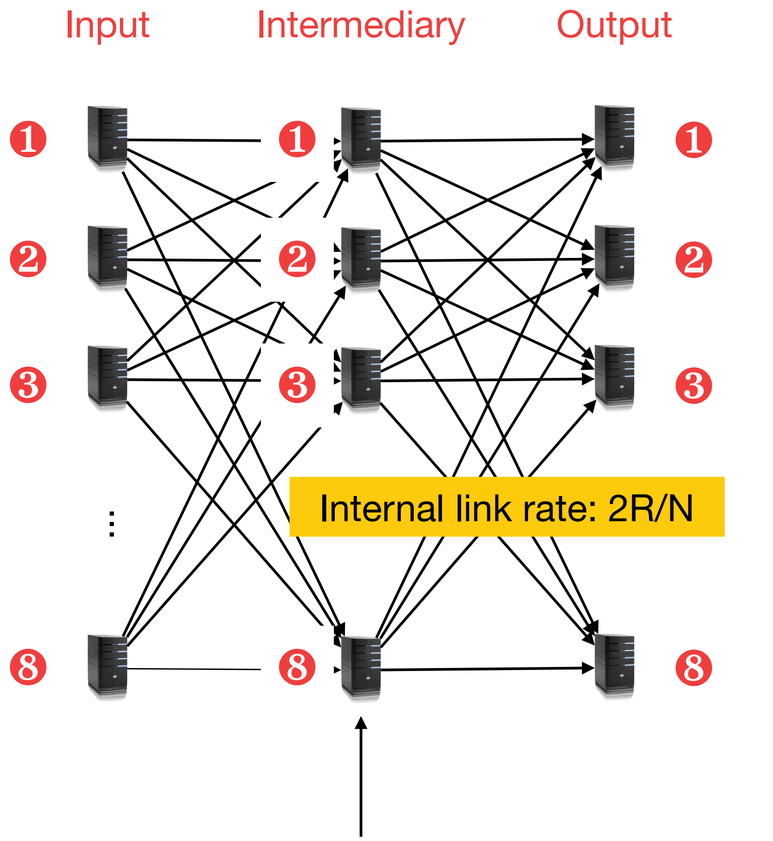
Full-mesh network



2-stage routing

Leslie Valiant, Gordon J. Brebner, Universal schemes for parallel communication, STOC 1981.

Distributed routing in VLB

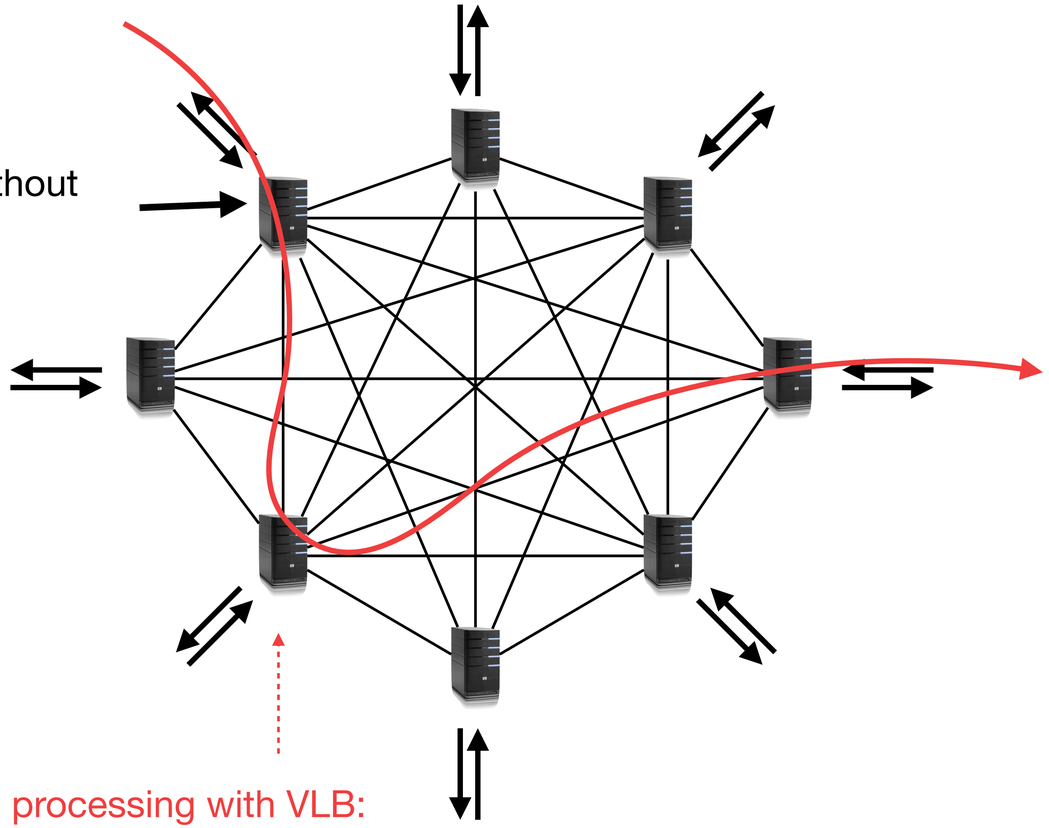


Uniform traffic sampling → 100% throughput

Server performance requirement

What is the required packet processing rate of each server?

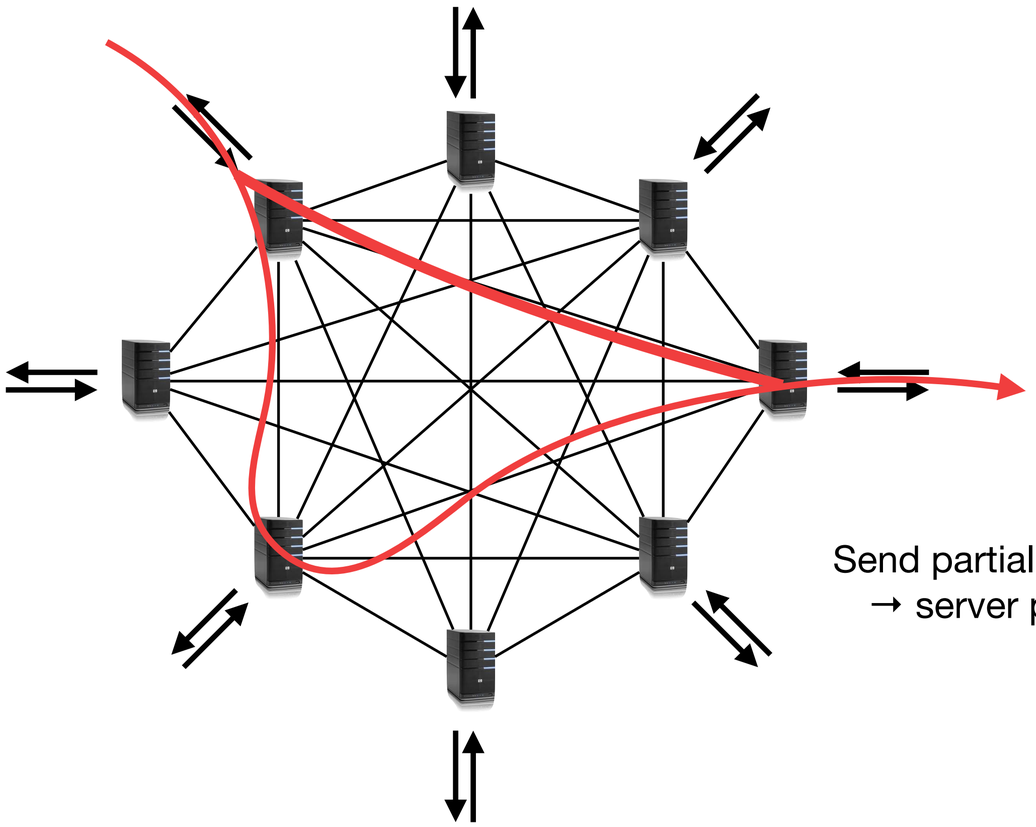
Packet processing without VLB: 2R bps



Packet processing with VLB: 3R bps

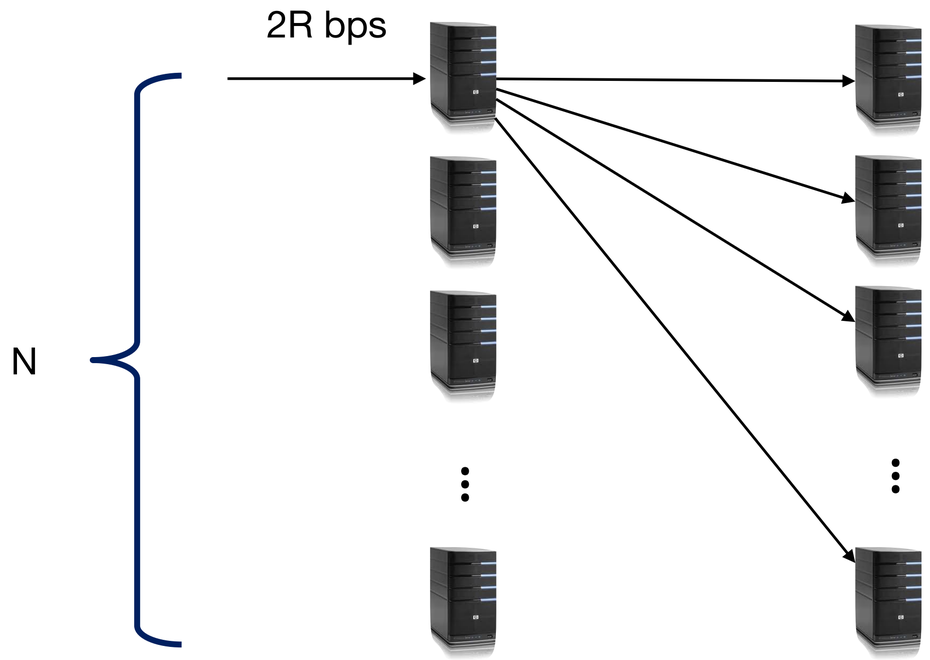
Reducing the server performance requirement with Direct VLB

If the traffic at the input is already uniformly distributed, no need to spread the traffic



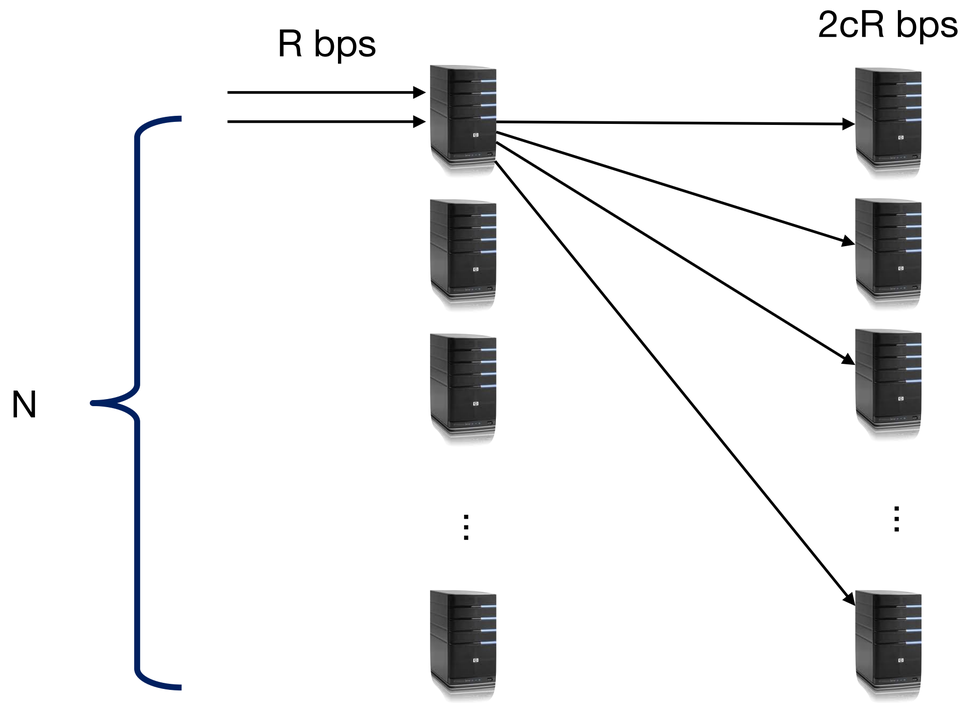
Send partial traffic directly to the destination
→ server processing rate: c*R (c in [2,3])

Scaling up with more ports or higher throughput



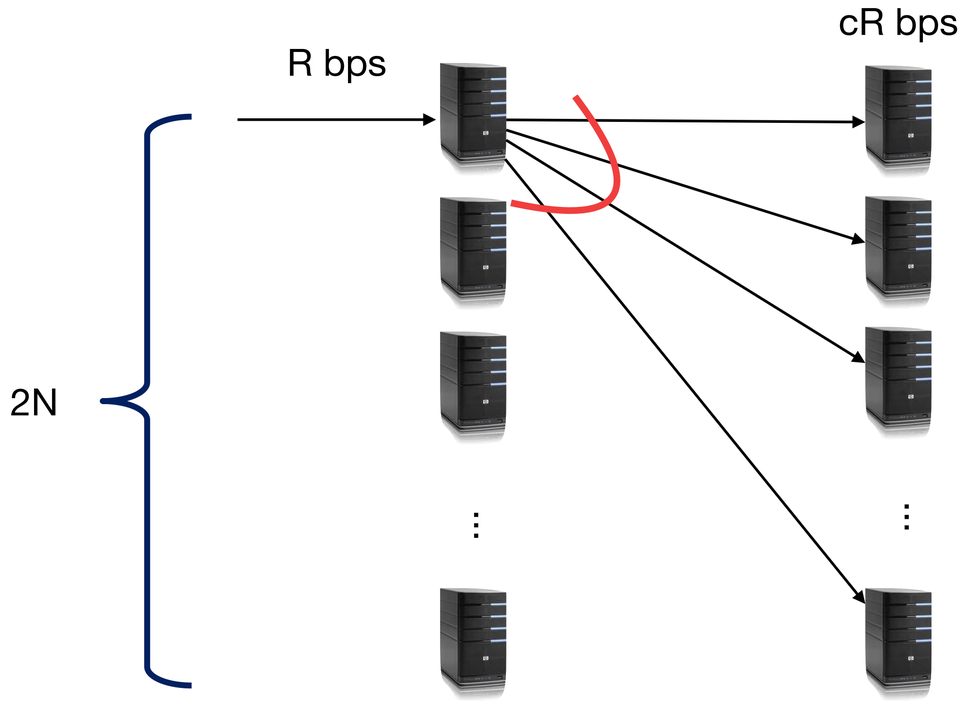
Approach #1: Increase server capacity

Scaling up with more ports or higher throughput



Approach #2: Doubling the input/output port number on a server requires the server to be able to handle traffic at 2cR bps rate.

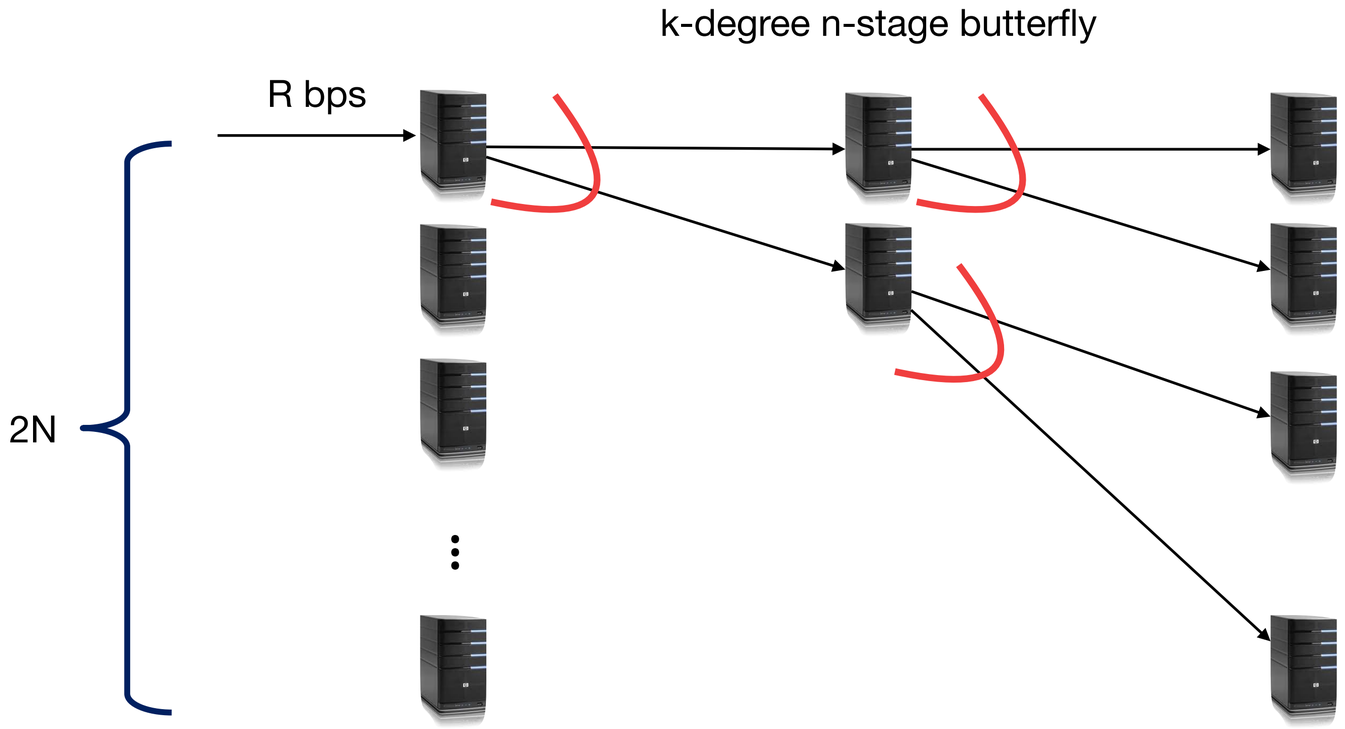
Scaling up with more ports or higher throughput



Approach #3: Doubling the number of servers would double the number of ports, but it requires higher server fan-outs which are also limited.

Introducing intermediate nodes

Use butterfly network topologies to overcome the fan-out limit issue

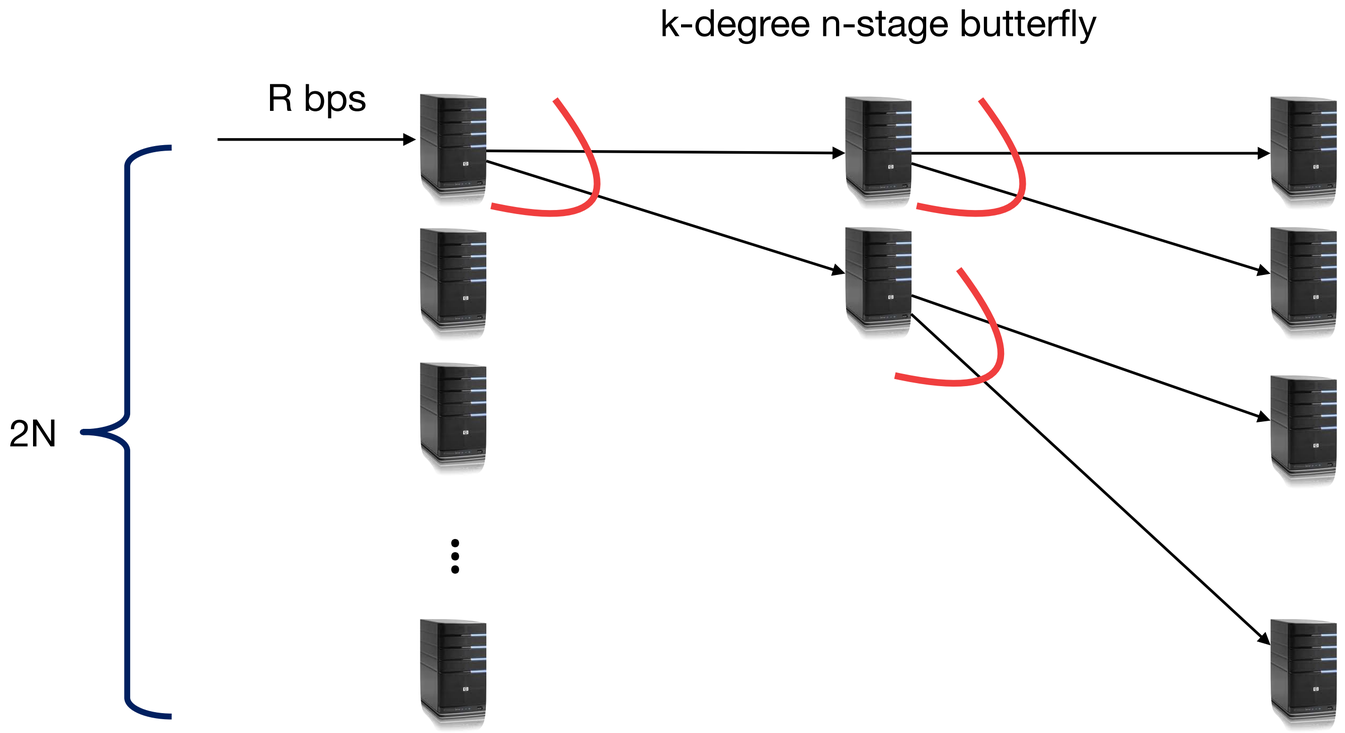


What problems can this approach bring?

Butterfly network: https://en.wikipedia.org/wiki/Butterfly_network

Introducing intermediate nodes

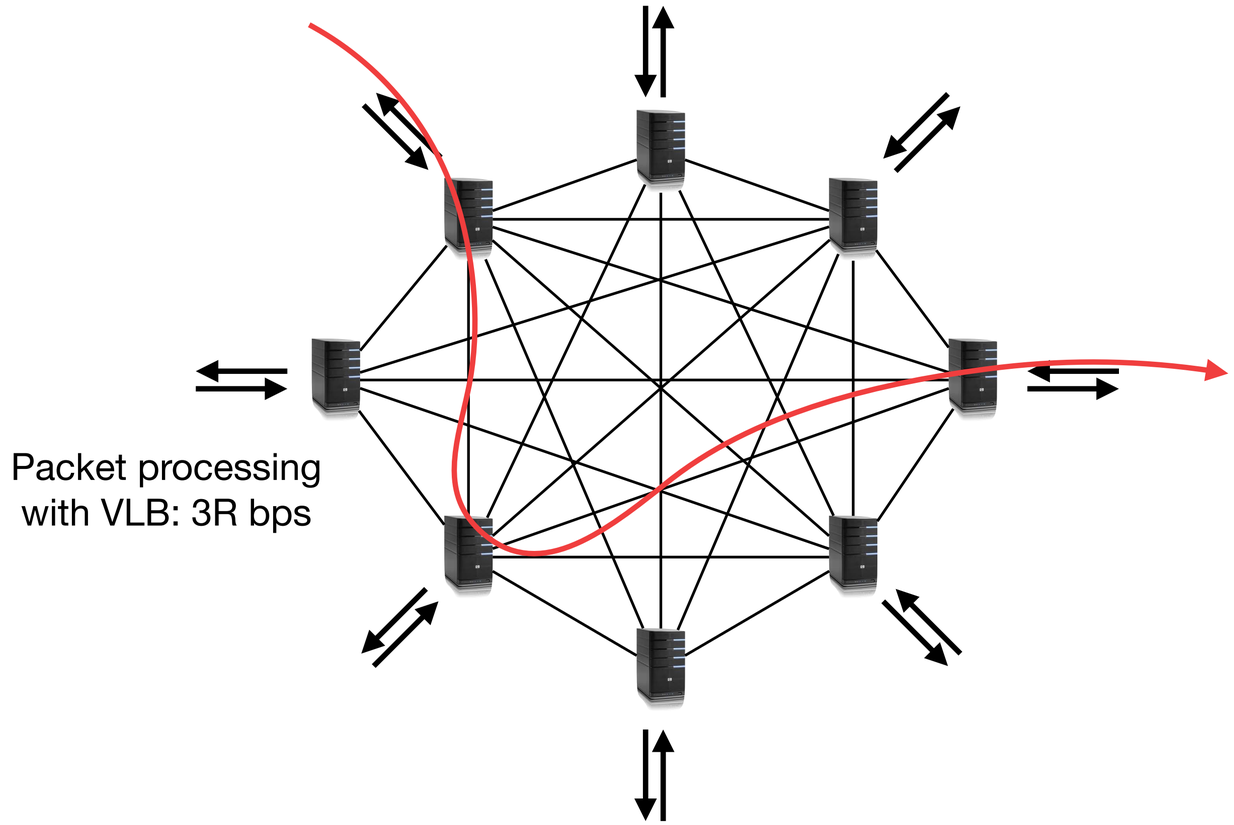
Use butterfly network topologies to overcome the fan-out limit issue



Higher per-packet latency since packets need to travel more servers.

Butterfly network: https://en.wikipedia.org/wiki/Butterfly_network

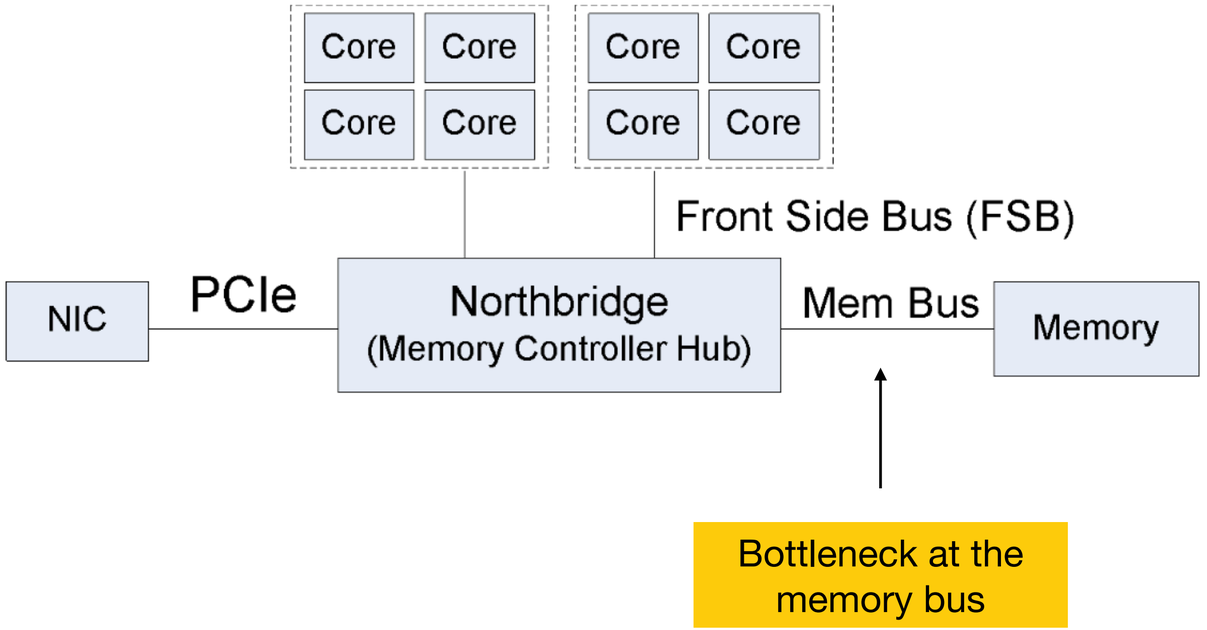
Recall server processing rate



Achieving such a rate on a server is non-trivial!

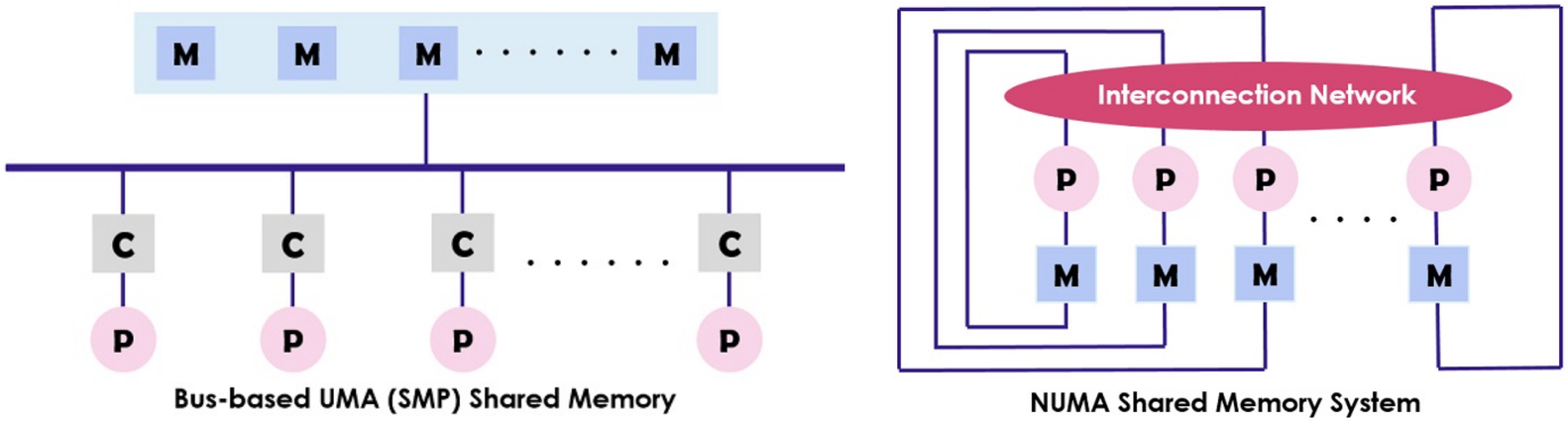
Shared-bus Xeon architecture

The expected performance cannot be achieved on traditional Xeon servers as the memory bandwidth is the bottleneck



NUMA architecture

Non-uniform memory access (NUMA)

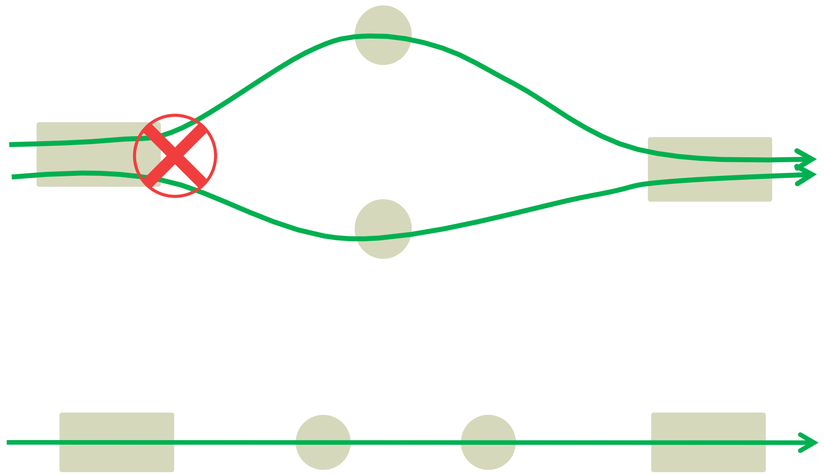


Under NUMA, a processor can access its own local memory faster than non-local memory.

Basic rules

Rule 1: Each network port be accessed by a single core

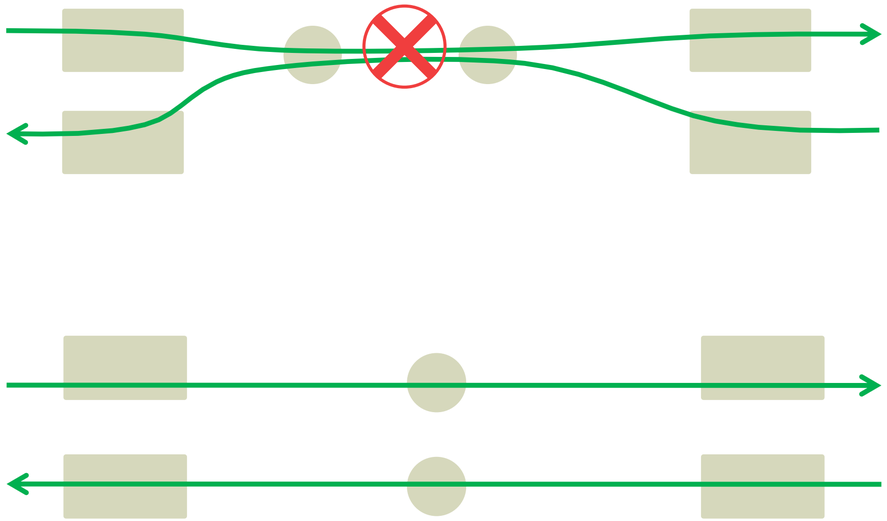
- Locking/unlocking queues is expensive



Basic rules

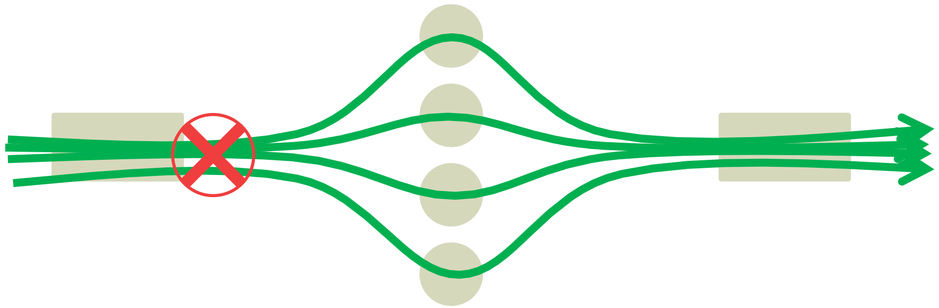
Rule 2: Each packet be handled by a single core

- Data synchronization between different cores is expensive



Basic rules

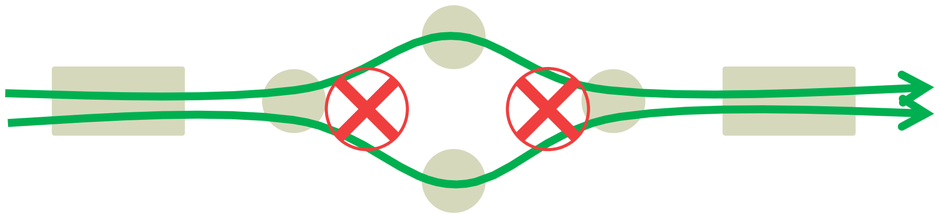
One core per port



There are more cores than ports. On the one hand, cores will be wasted if not used.
On the other hand, using multiple cores for a port would require locking on the port.

Basic rules

One core per packet

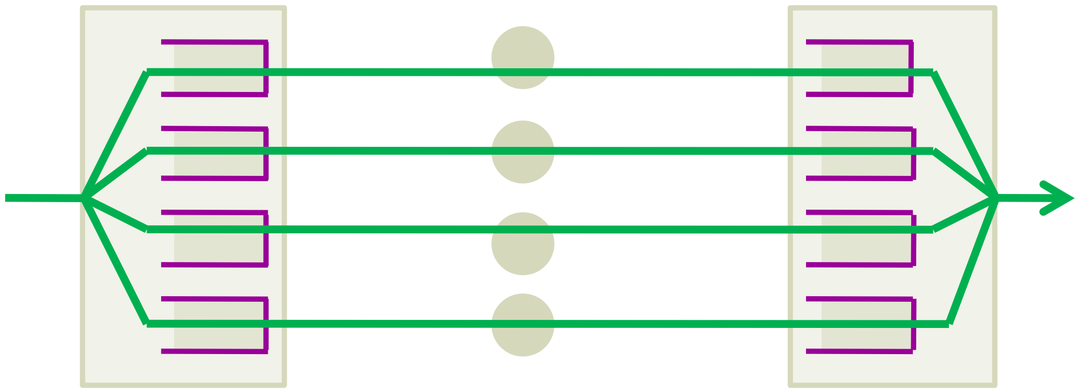


If we use a core tied to the port to poll packets and distribute/aggregate the packets to other cores, a packet will be touched by multiple cores inevitably.

Leverage multi-queue NICs

Multi-queue technique is available in most modern NICs

Allocate one core per queue, avoiding all the above issues.



A customized driver is developed to bind polling and sending elements to a particular queue (as opposed to a particular port).

Server-side optimization summary

State of the art and emerging hardware

- NUMA architecture, multi-queue NICs

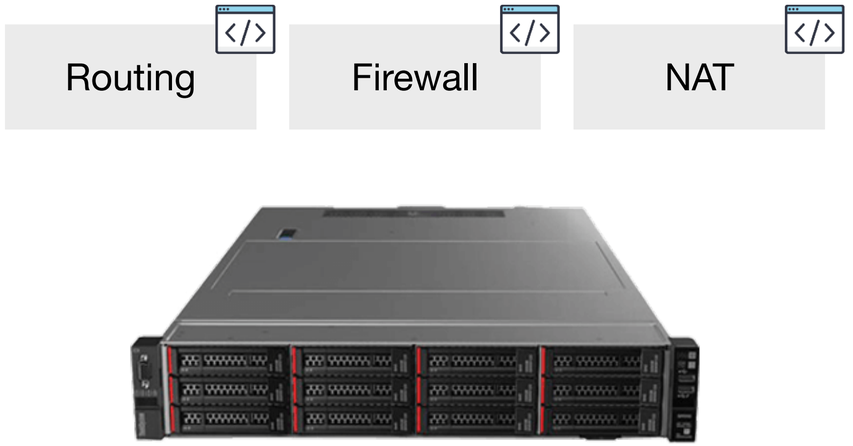
Modified NIC driver

- Batching (poll multiple packets from the NIC instead of one at a time)

Careful queue-to-core allocation

- One core per queue, per packet

Summary



Click provides a modular architecture for programming network functions.
RouteBricks provides a high-performance parallel solutions for achieving hardware-comparable speeds for network functions.

Next time: programmable data plane