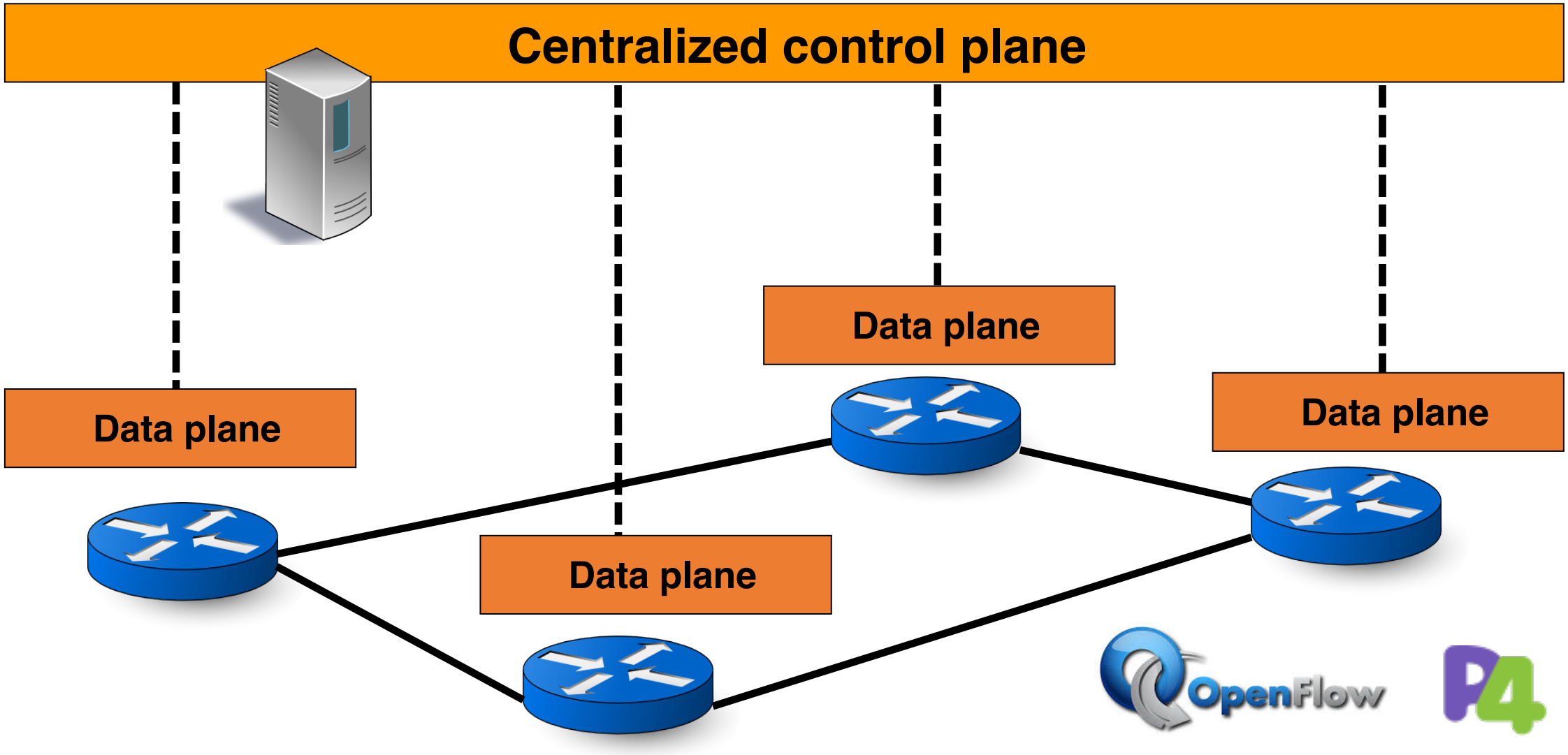
Programmable Software Switches

Lecture 11, Computer Networks (198:552)

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Software-Defined Network (SDN)



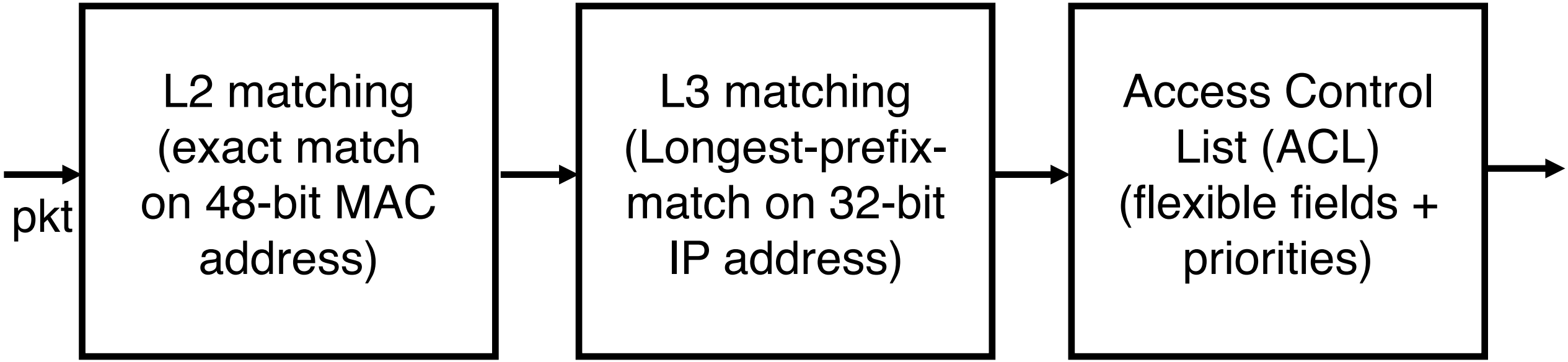
Why software switching?

- Early roots in networking: first switches were fully in software
  - … until high link speeds forced everyone to make ASICs
- A tool for experimentation
- Virtualization: need flexible switching policies inside endpoint!
  - Why switching?
  - Where is all the policy coming from?

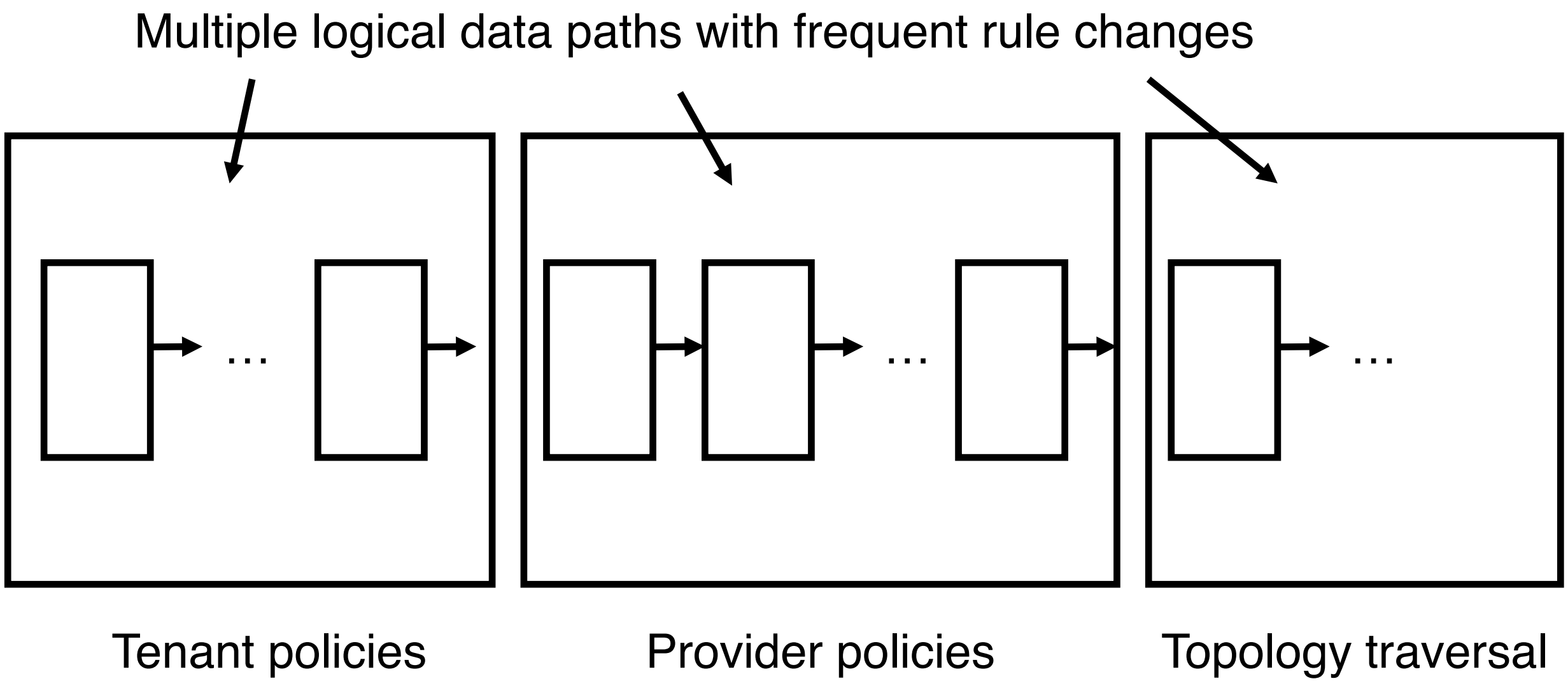
What do software switches need to do?

- Forward packets based on specified rules
- Allow changing the forwarding rules frequently
- Provide high performance
  - High throughput and low latency
- Be robust
  - Different traffic patterns (e.g., port scans)
  - Can we make performance deterministic?

A conventional packet data path



A packet data path in a virtualized cluster



High performance is challenging

- Lots of lookups
  - Compute hashes and maintain other data structures
  - Large tables: numerous highly-specific keys
- User-kernel crossings
  - Programmable toolchain much easier to build in user space
- Rules and policies change a lot
  - New tenants, new endpoints, VM migrations, …
- Need to understand computer systems deeply
  - Costs of memory accesses and hash computations
  - Core/thread-level parallelism, NIC-queue-to-core mapping, etc.

A few other quirks to accommodate

- Metadata carried between table stages
  - Ex: Late-bind forwarding decisions
  - Ex: Accumulate partial field modifications before one-shot write
- Statistics per flow
  - Counters
  - Timeouts
- Stateful processing
  - Connection tracking for firewalls: e.g., TCP establishment state
  - NAT, DNS tunnel detection, load balancing, …
  - <insert your idea here!>

Papers for today: Click and OpenVSwitch

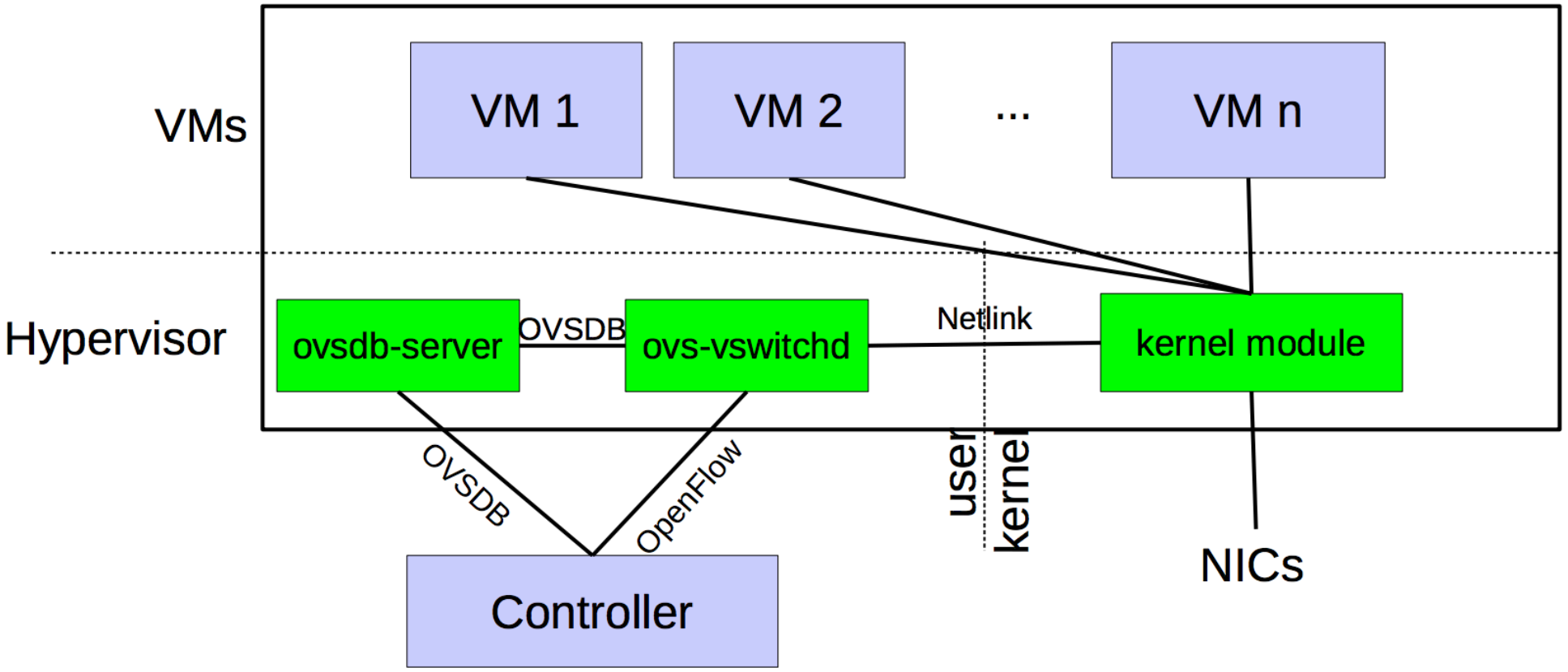
- Each highly influential in its own right
  - Click spurred a lot of work in flexible router design
  - OVS spurred work on programmable networks
- Useful software tools: Course project and beyond!
- Useful tools layered on top
  - Mininet, network functions, ...
- Active research area: packet processing performance + flexibility
  - Kernel bypass
  - Hardware offloads
  - Kernel network stack optimizations

The Design and Implementation of OpenVSwitch

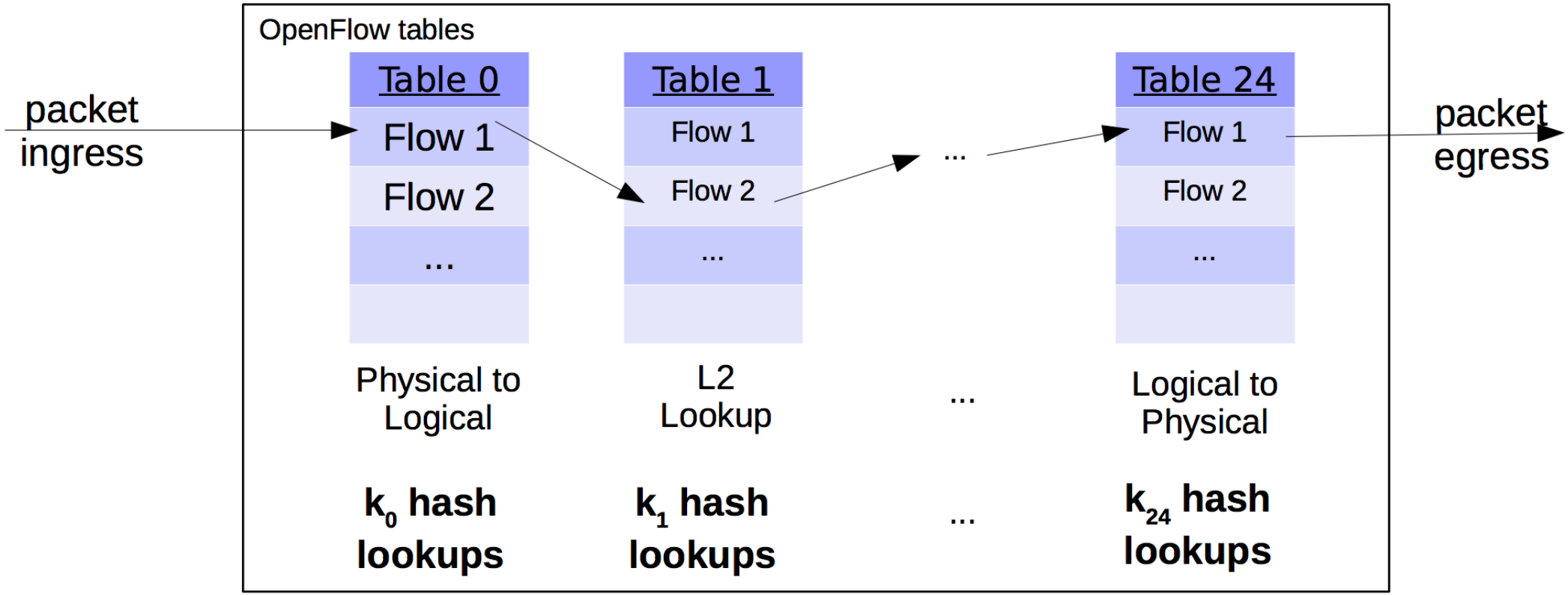
Usenix NSDI ‘15

Ben Pfaff et al.

Deployment setting: Virtualized clusters



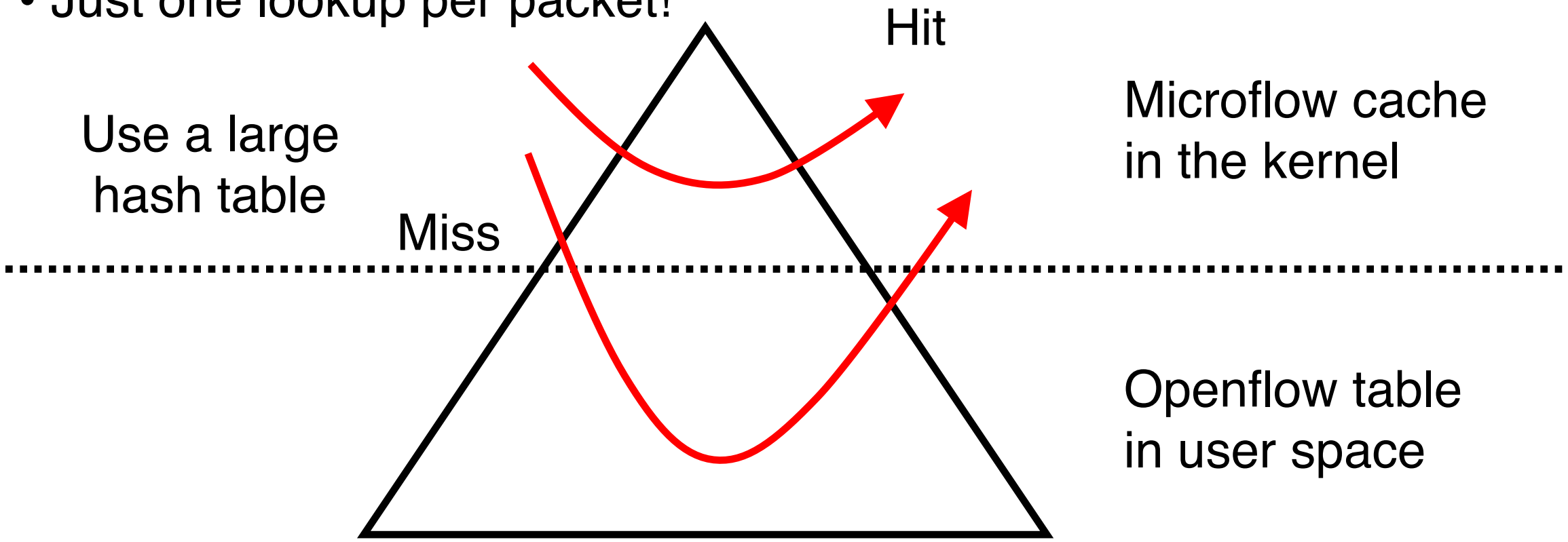
Implications for forwarding performance



Complex policies: Can’t do 100+ lookups per packet!

Idea 1: Microflow cache

- Microflow: complete set of packet headers and metadata
  - Example: srcIP, dstIP, IP TTL, srcMAC, dstMAC
- Just one lookup per packet!



Problems with micro-flows

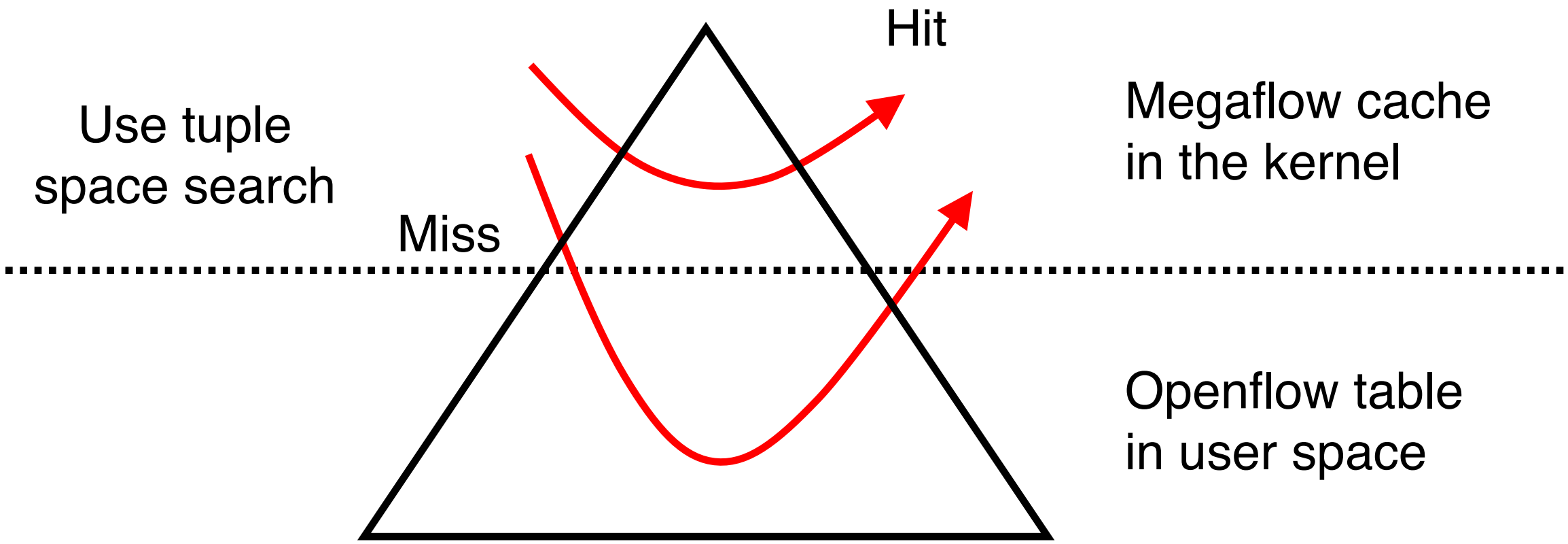
- Too many micro-flows: e.g., each TCP port
- Many micro-flows may be short lived!
  - Poor cache-hit rate

- Can we cache the outcome of rule lookup directly?

- Naive approach: Cross-product explosion!
  - Example: Table 1 on source IP, table 2 on destination IP

Idea 2: Mega-flow cache

- Build the cache of rules *lazily* using fields accessed in OF table
  - Ex: contain just src/dst IP combinations that appeared in packets



Improvements to mega-flow caching

- You have an OF table. What happens if you populate the mega-flow blindly by concatenating the fields accessed on lookup?
  - Hint 1: consider flow tables that match on highly variable fields
  - Hint 2: consider priorities of rules with overlapping matches
  - Hint 3: consider the number of lookups vs. microflow cache

- OVS introduced a series of *algorithmic* improvements
  - Priority sorting of mega-flow tables
  - Staged lookups starting with more static sets of fields
  - Prefix trie to detect non-overlapping longest-prefix matches
  - Combine with micro-flow cache!

Click Modular Router

Symposium on Operating Systems Principles (SOSP) ‘99

Robert Morris, Eddie Kohler, John Jannotti, and Frans Kaashoek

Motivation for Click

- Flexibility: Add new features and enable experimentation
- Openness: Allow users/researchers to build and extend
  - … In contrast to most commercial routers
- Modularity
  - Simplify the composition of existing features
  - Simplify the addition of new features
- Speed/efficiency
  - Operation (optionally) in the operating system
  - ... without the user needing to grapple with OS internals

Router as a graph of elements

- Large number of small elements
  - Each performing a simple packet function
  - E.g., IP look-up, TTL decrement, buffering
- Connected together in a graph
  - Elements inputs/outputs snapped together
  - Beyond elements in series to a graph
  - E.g., packet duplication or classification
- Packet flow as main organizational primitive
  - Consistent with data-plane operations on a router
  - (Larger elements needed for, say, control planes)

Click elements: Push vs. pull

- Packet hand-off between elements
  - Directly inspired by properties of routers
  - Annotations on packets to carry temporary state
- Push processing
  - Initiated by the source end
  - E.g., when an unsolicited packet arrives (e.g., from a device)
- Pull processing
  - Initiated by the destination end
  - E.g., to control timing of packet processing (e.g., based on a timer or packet scheduler)

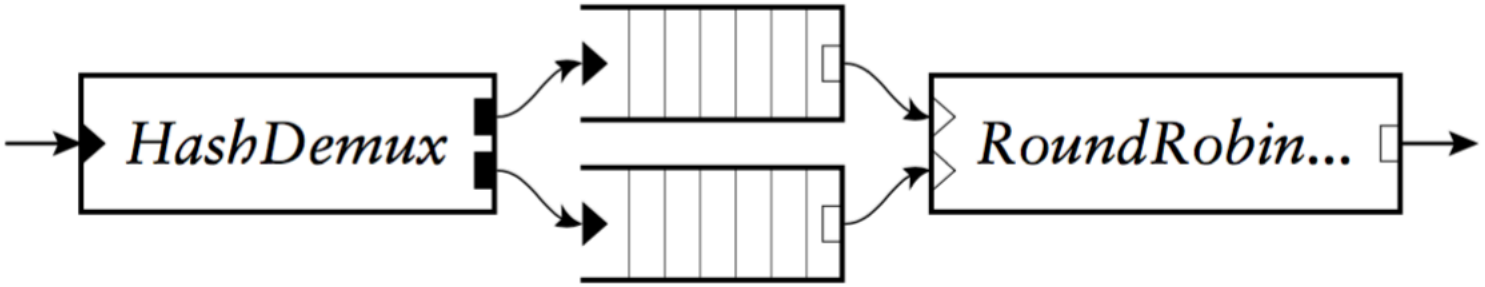
Click language

- Declarations
  - Create elements
- Connections
  - Connect elements
- Compound elements
  - Combine multiple smaller elements, and treat as single, new element to use as a primitive class
- Language extensions through element classes
  - Use configuration strings, rather than syntactic extensions

```
src :: FromDevice(eth0);
ctr :: Counter;
sink :: Discard;
src -> ctr;
ctr -> sink;

# the same, with anonymous elements
FromDevice(eth0) -> Counter -> Discard;
```

Example: Stochastic Fair Queueing



Backup Slides

Handlers and Control Socket

- Access points for user interaction
  - Appear like files in a file system
  - Can have both read and write handlers
- Examples
  - Installing/removing forwarding-table entries
  - Reporting measurement statistics
  - Changing a maximum queue length
- Control socket
  - Allows other programs to call read/write handlers
  - Command sent as single line of text to the server

Example: EtherSwitch Element

- Ethernet switch
  - Expects and produces Ethernet frames
  - Each input/output pair of ports is a LAN
  - Learning and forwarding switch among these LANs
- Element properties
  - Ports: any # of inputs, and same # of outputs
  - Processing: push
- Element handlers
  - Table (read-only): returns port association table
  - Timeout (read/write): returns/sets TIMEOUT