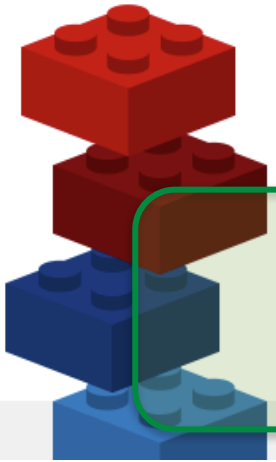
# The Forwarding Plane: An Old New Frontier of Networking Research


CS244, Spring 2019

Changhoon Kim
chang@barefootnetworks.com

ABOUT PROJECTS SOLUTIONS SOFTWARE DEFINED STANDARDS GET INVOLVED

SDN OVERVIEW PRODUCT CERTIFICATION SKILLS CERTIFICATION CORD LEARNING LABS NEW



# Software-Defined Networking (SDN) Definition

**What is SDN?** The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices.

# What is SDN in plain English?

- Ideally at the level for college freshmen
  - Because, if you can't, you are not really understanding it! [*Feynman's guiding principle*]

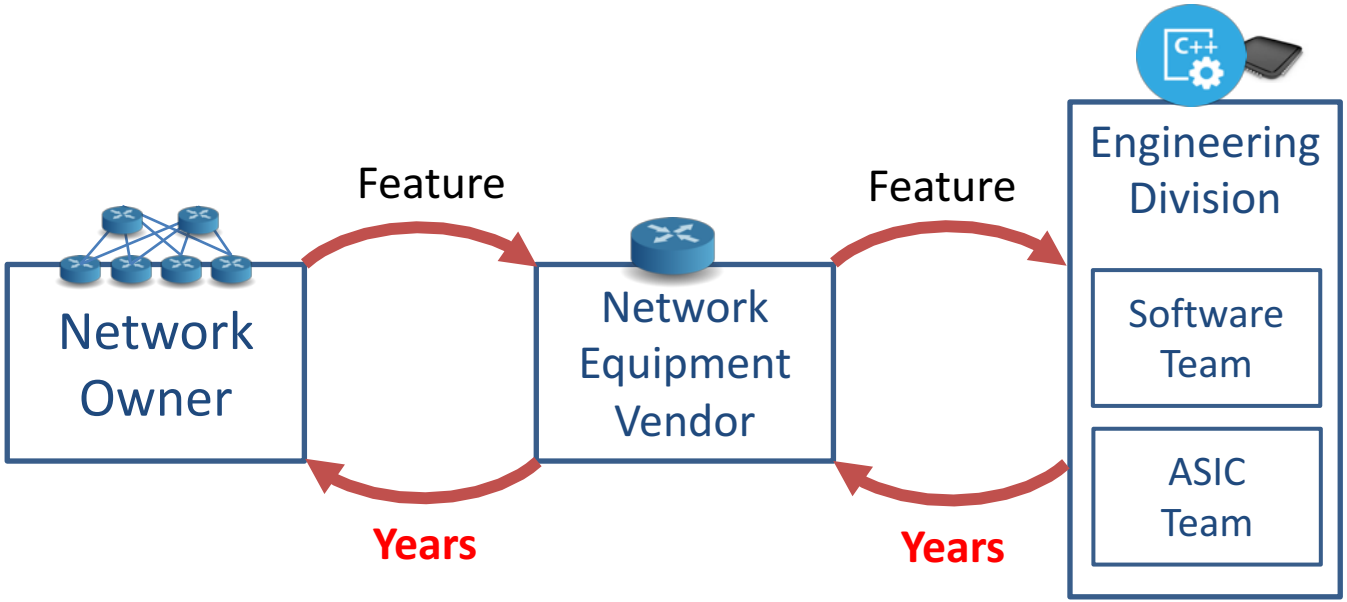
**"Making programming networks as easy as programming computers."**

# Natural questions that follow

**"Making programming networks as easy as programming computers."**

- Why should we program a network?
  - To realize some "beautiful ideas" easily, preferably on our own
- What are those "beautiful ideas"?
  - Any impactful or intriguing apps in particular?
- Why couldn't we do this easily in the pre-SDN era?
  - Any fundamental shifts happening?

# Pre-SDN state of the network industry

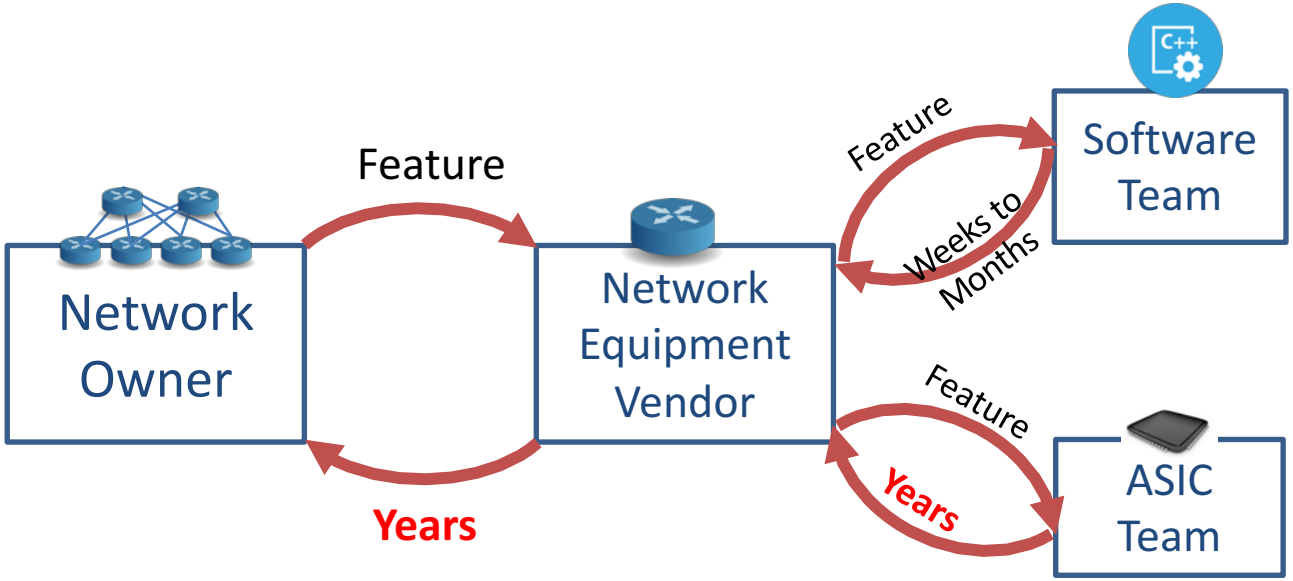


# Compared to other industries, this is very unnatural

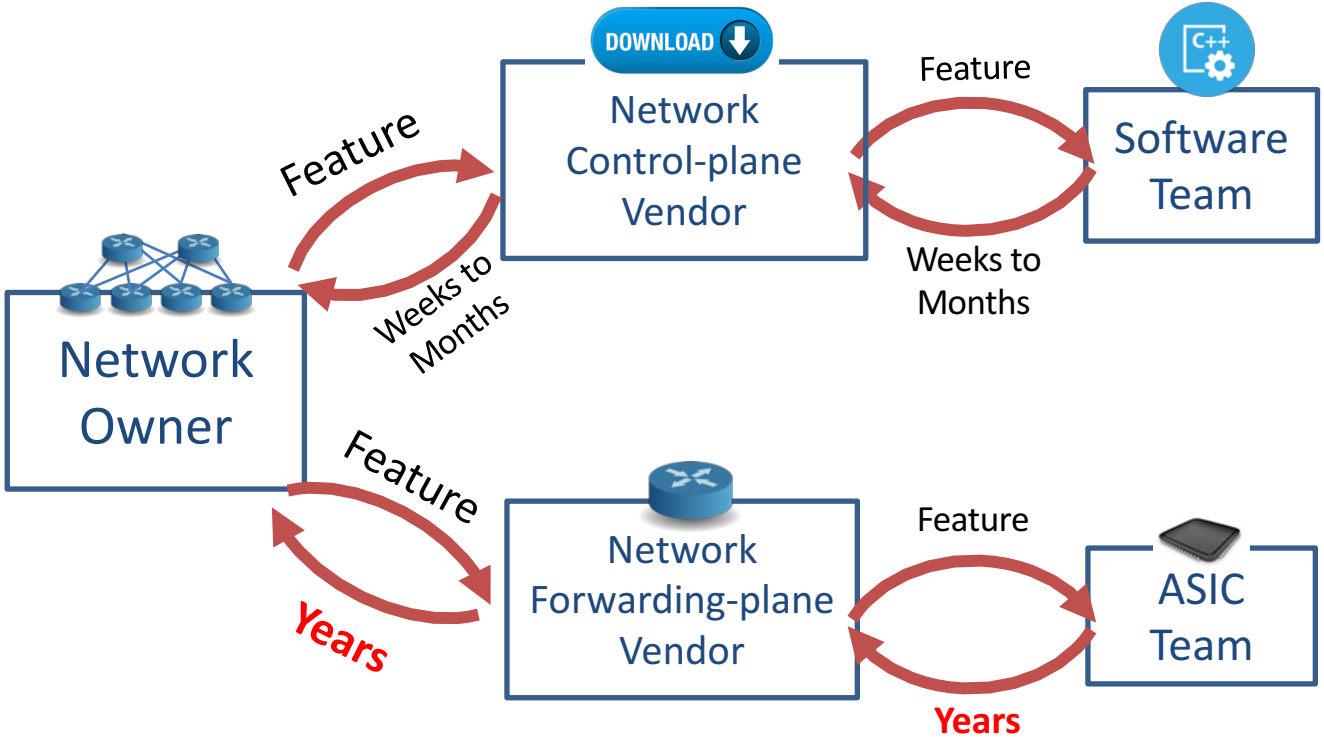
- Because we all know how to realize our own ideas by programming CPUs, GPUs, TPUs, etc.
  - Programs used in every phase (implement, verify, test, deploy, and maintain)
  - Extremely fast iteration and differentiation
  - We own our own ideas
  - A sustainable ecosystem where all participants benefit

**Can we replicate this healthy, sustainable ecosystem for networking?**

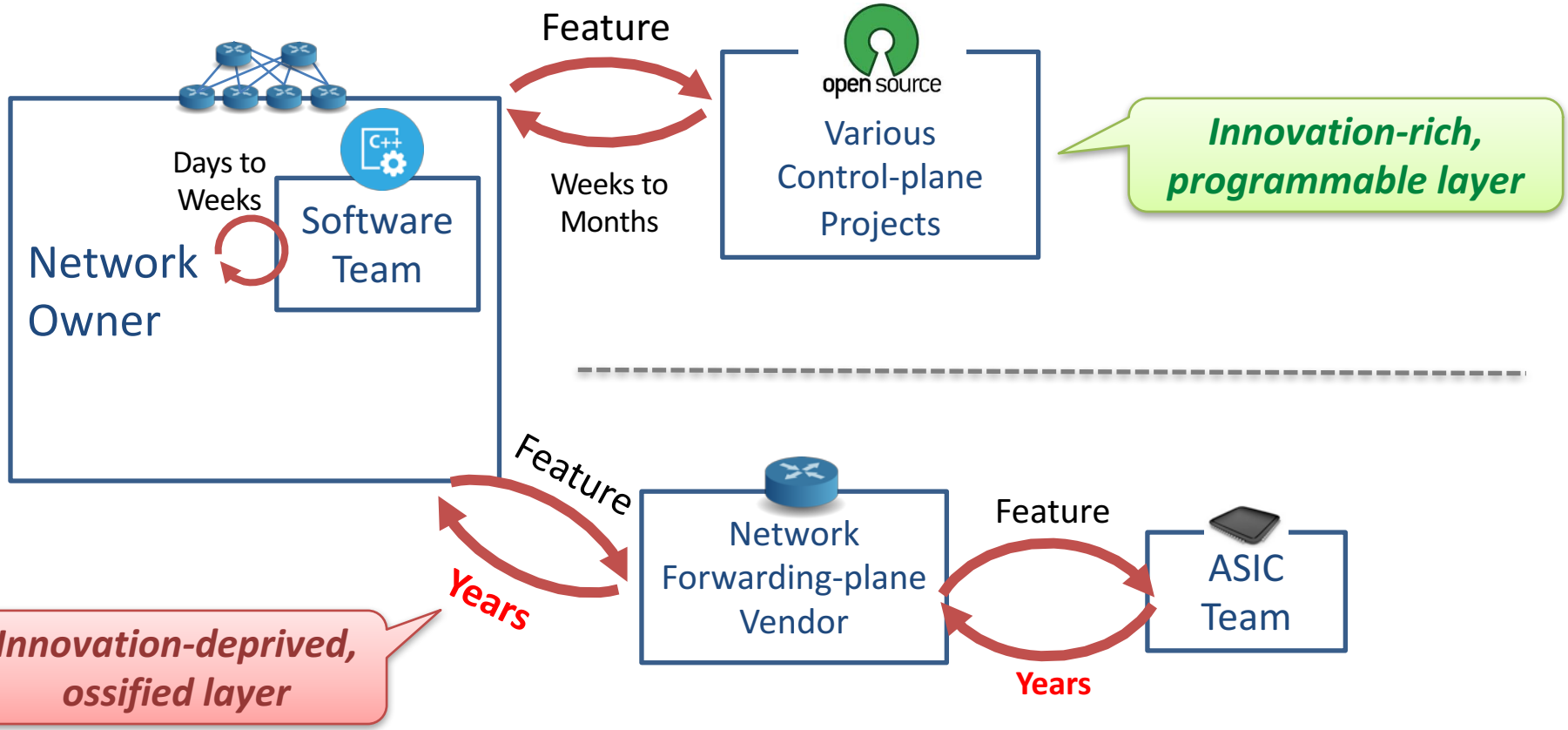
# What SDN pioneers had realized …



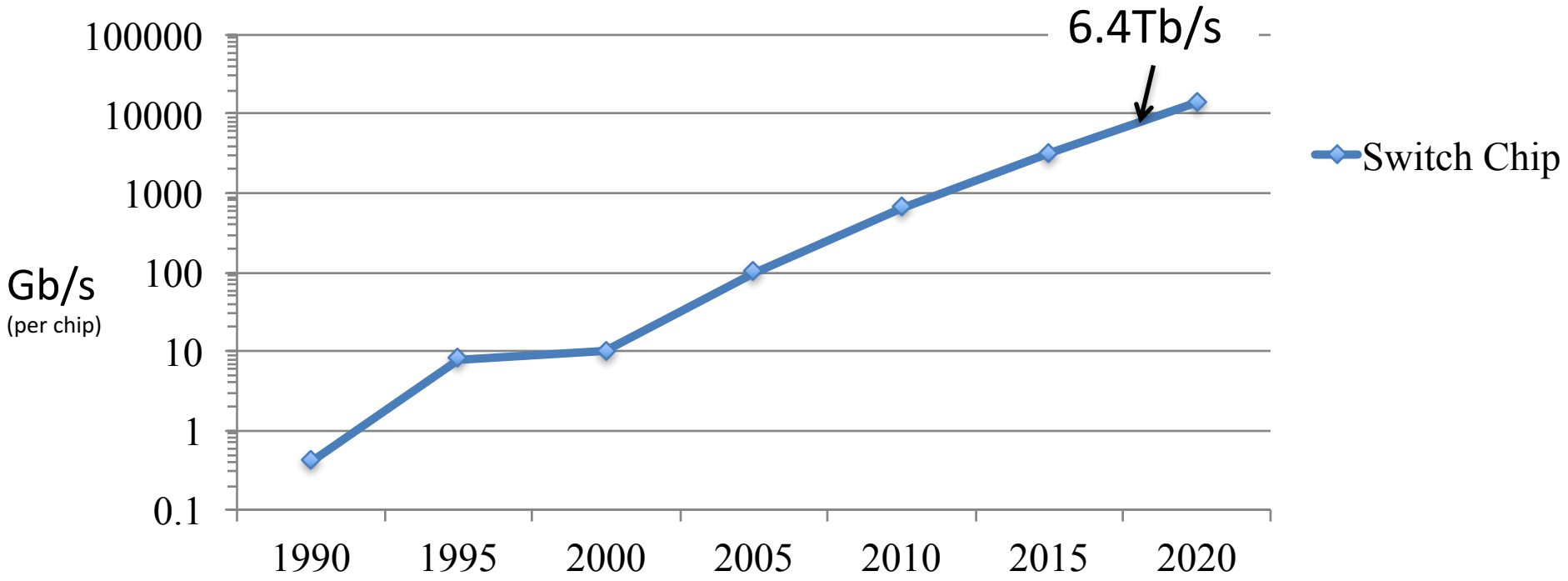
# And, SDN started to unfold …



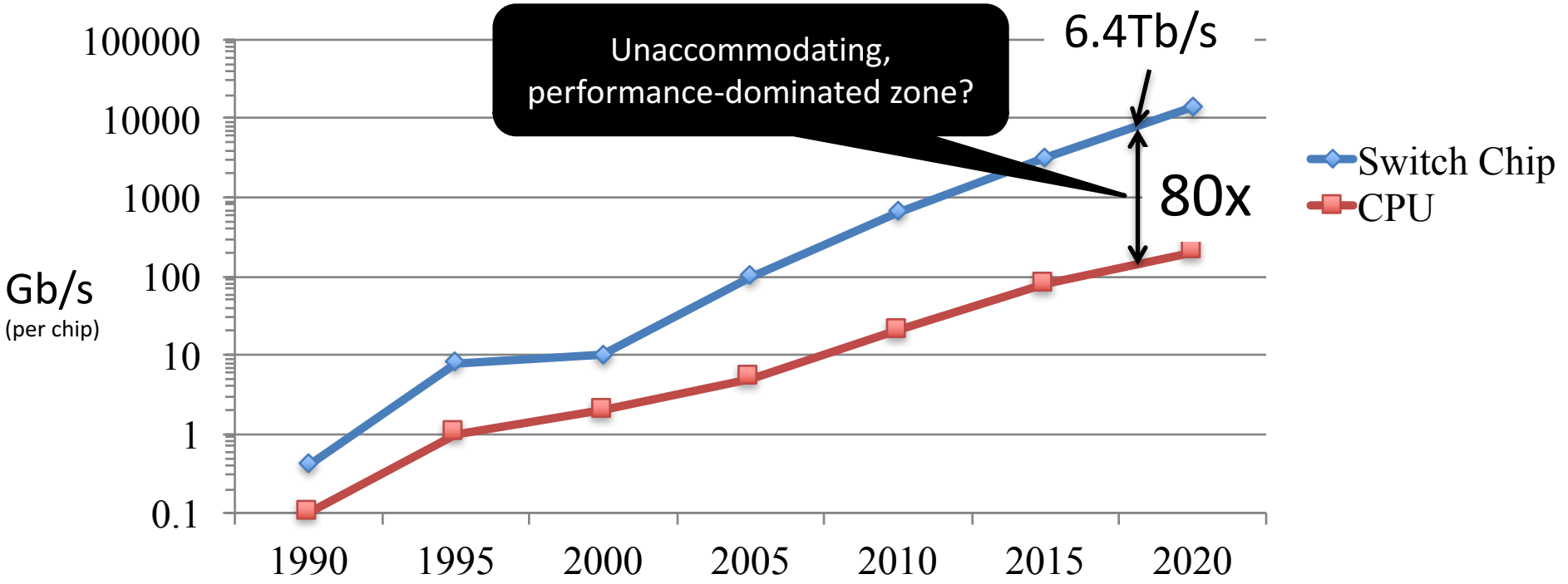
# And, SDN started to unfold …



# Reality: Packet forwarding speeds



# Reality: Packet forwarding speeds



*"Programmabl* *-100x slower than fixed-* *ey cost more and* *wer."*

*tional wisdom in networking*

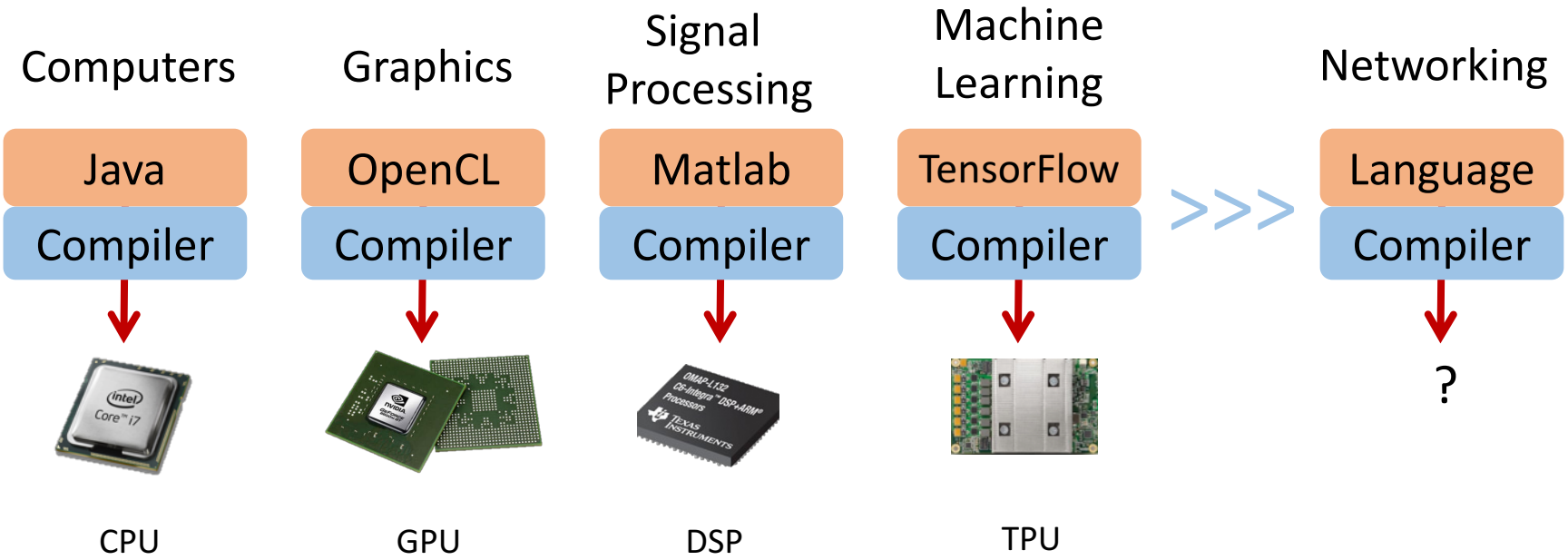
# **Evidence: Tofino 6.5Tb/s switch** (arrived Dec 2016)


The world's fastest and most programmable switch.

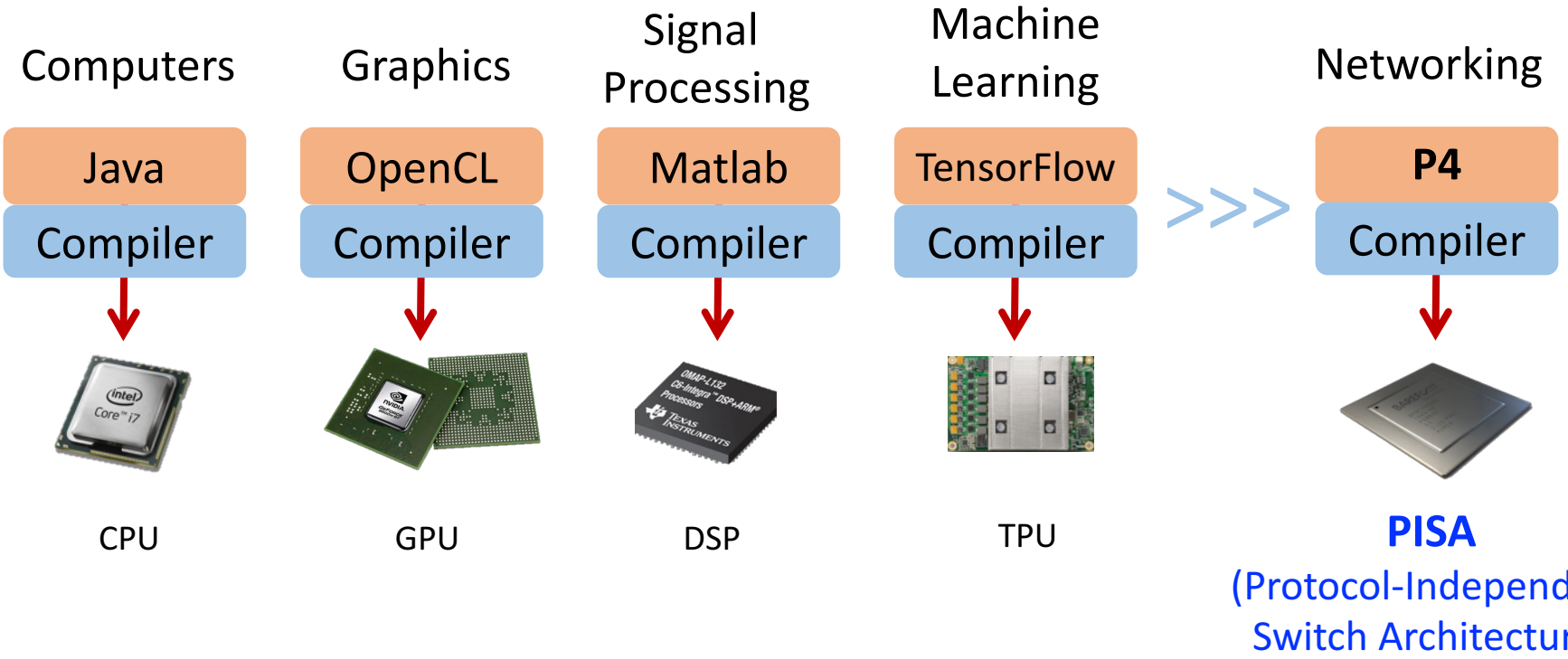
No power or cost penalty compared to fixed-function switches.

An incarnation of **PISA (Protocol Independent Switch Architecture)**

# Domain-specific processors

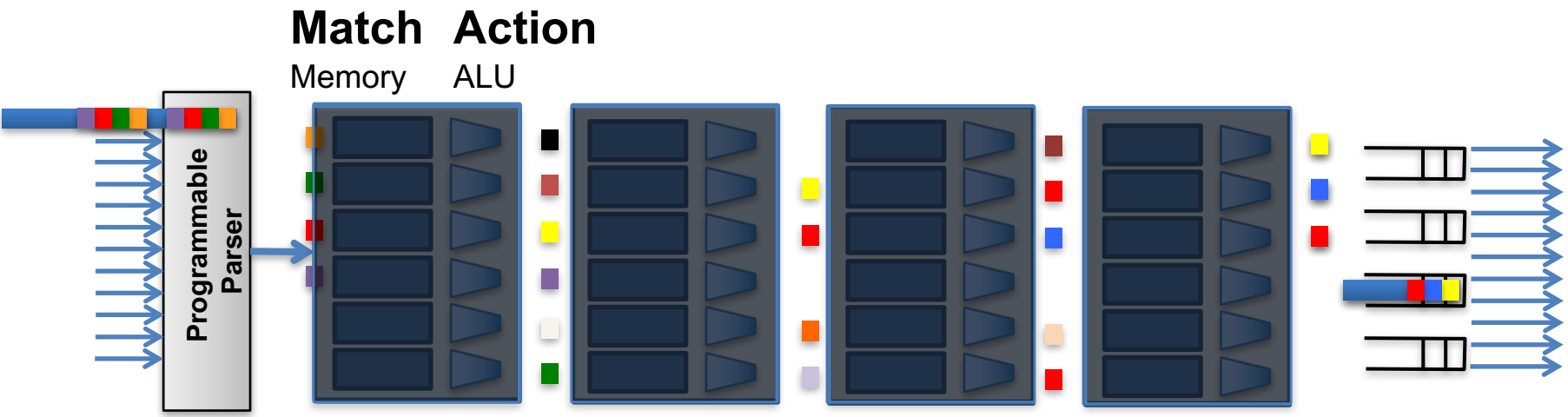


# Domain-specific processors

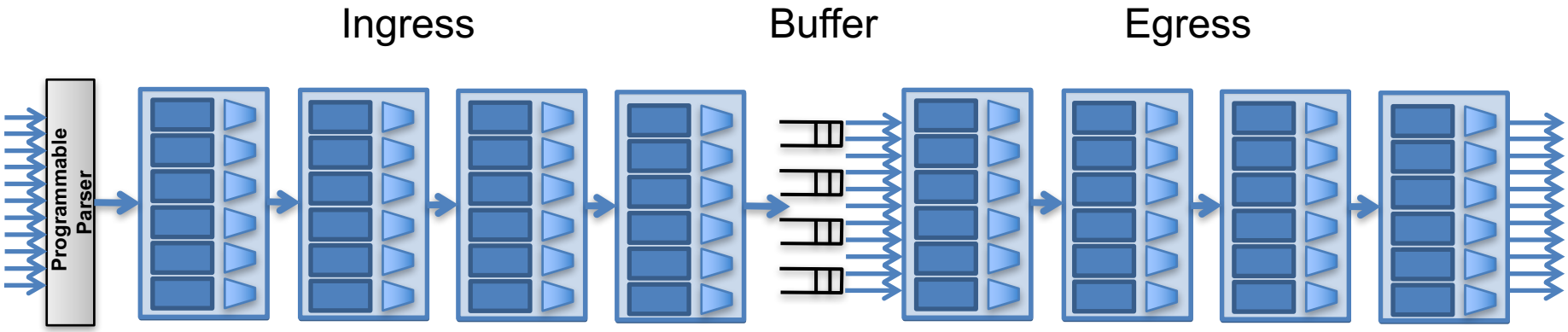


# PISA: An architecture for high-speed programmable packet forwarding

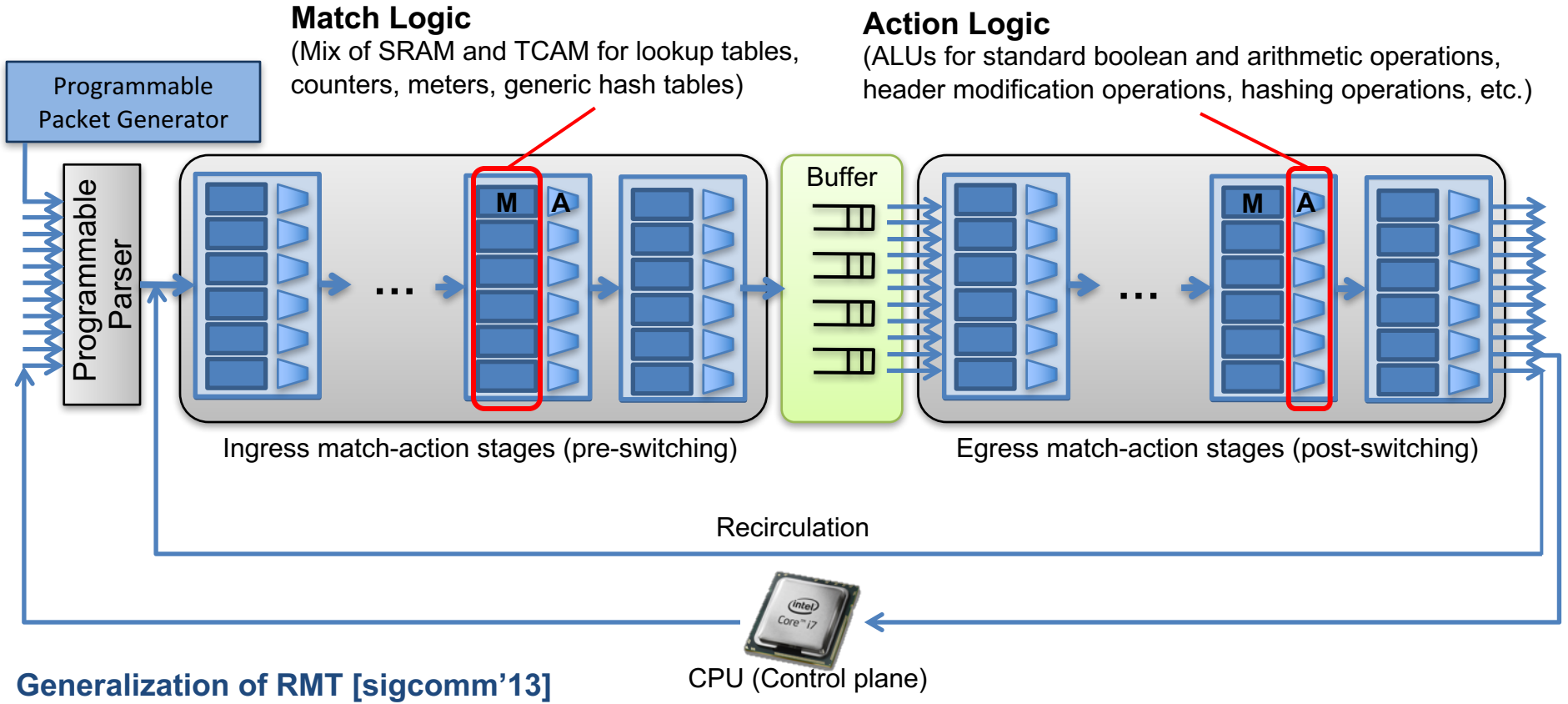
# PISA: Protocol Independent Switch Architecture



# PISA: Protocol Independent Switch Architecture



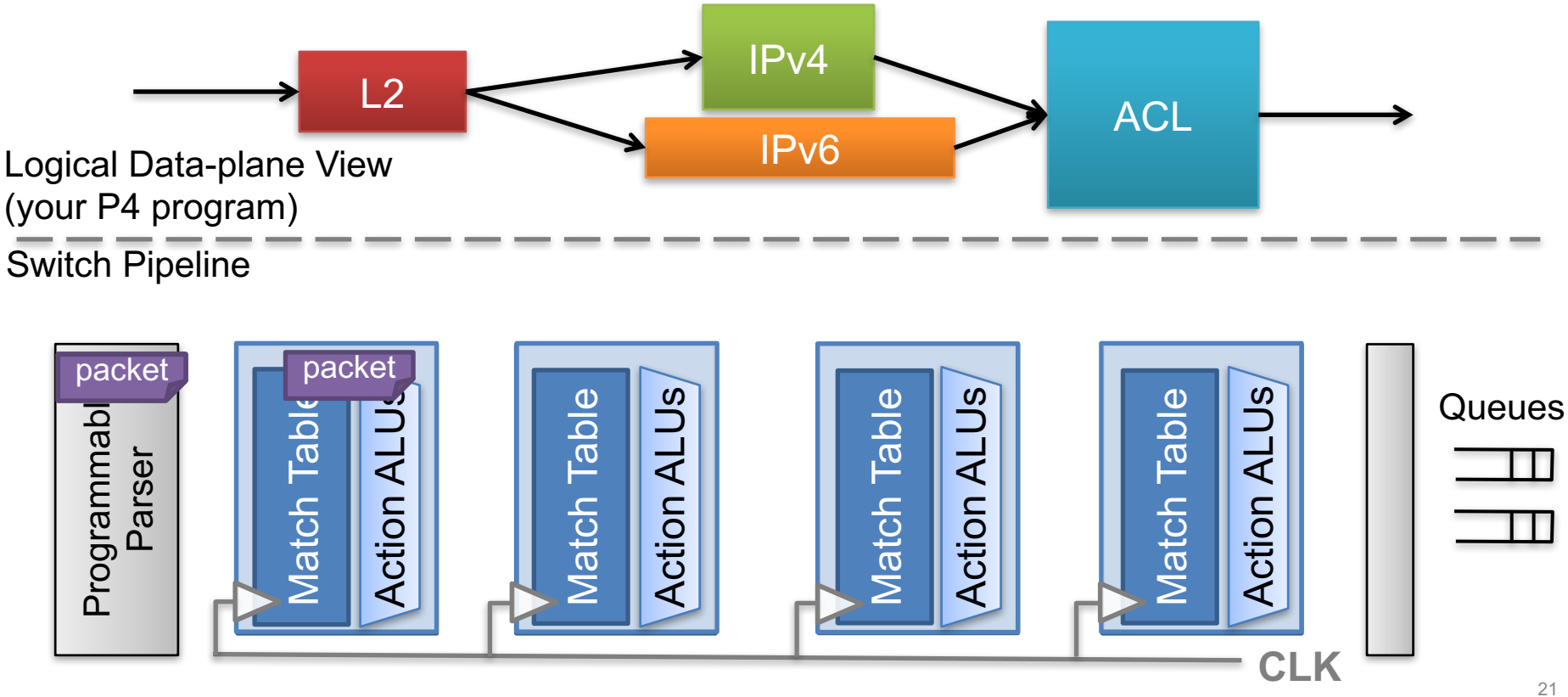
# PISA: Protocol Independent Switch Architecture



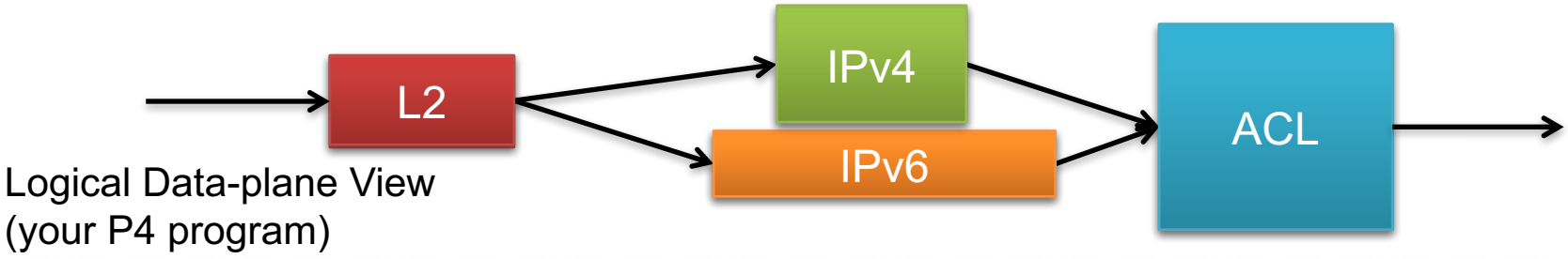

**Generalization of RMT [sigcomm'13]**

# Why we call it protocol-independent packet processing

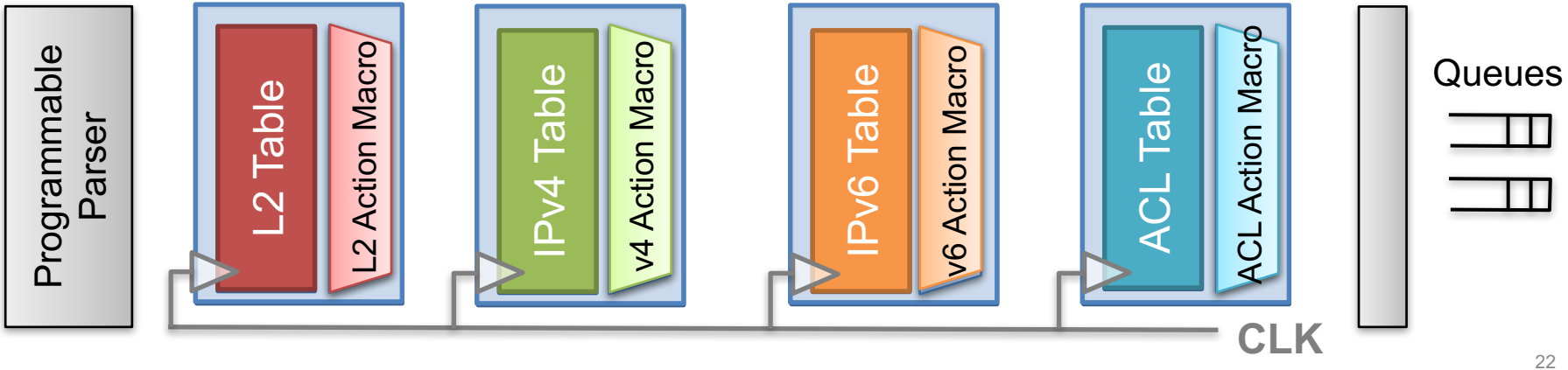
# Device does not understand any protocols until it gets programmed



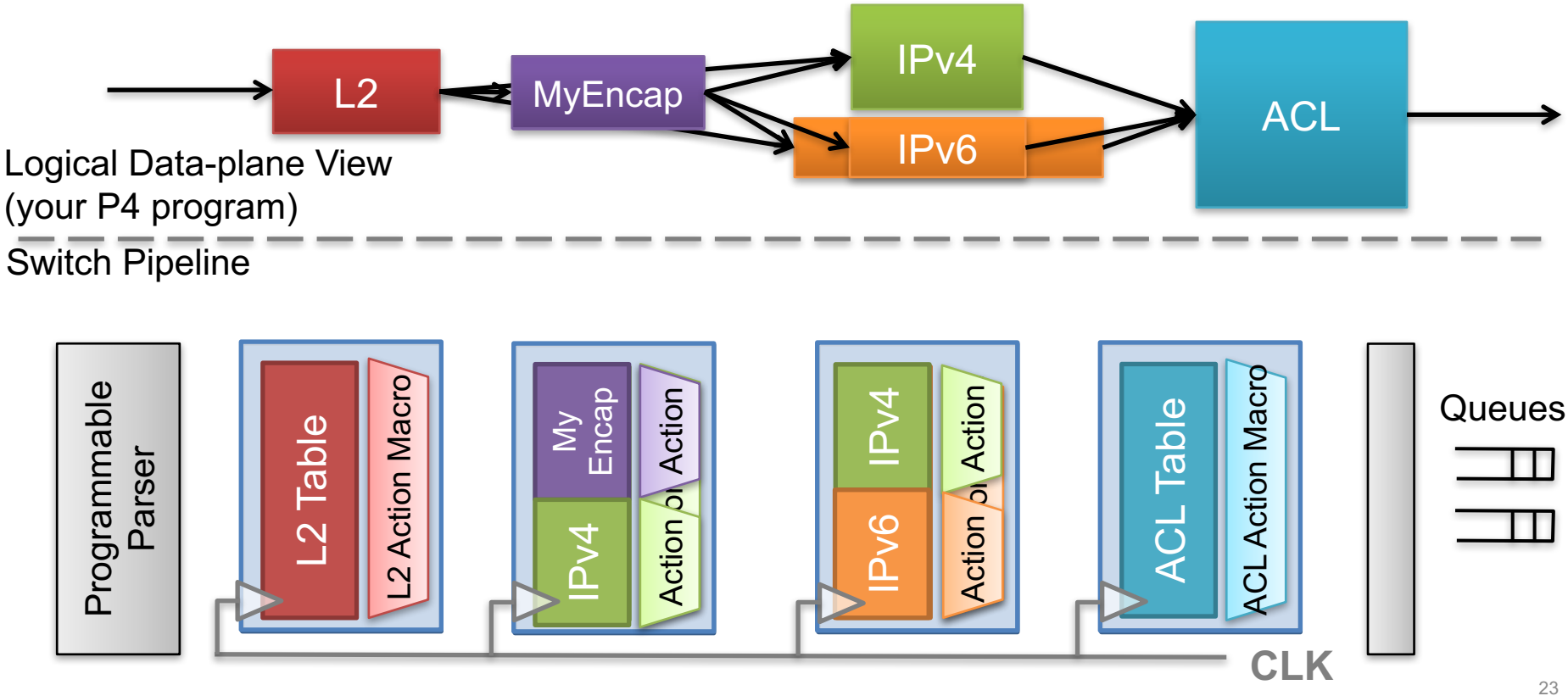
# Mapping logical data-plane design to physical resources




Switch Pipeline



# Re-program in the field



# P4 language components


State-machine;
Field extraction

Table lookup and update;
Field manipulation;
Control flow

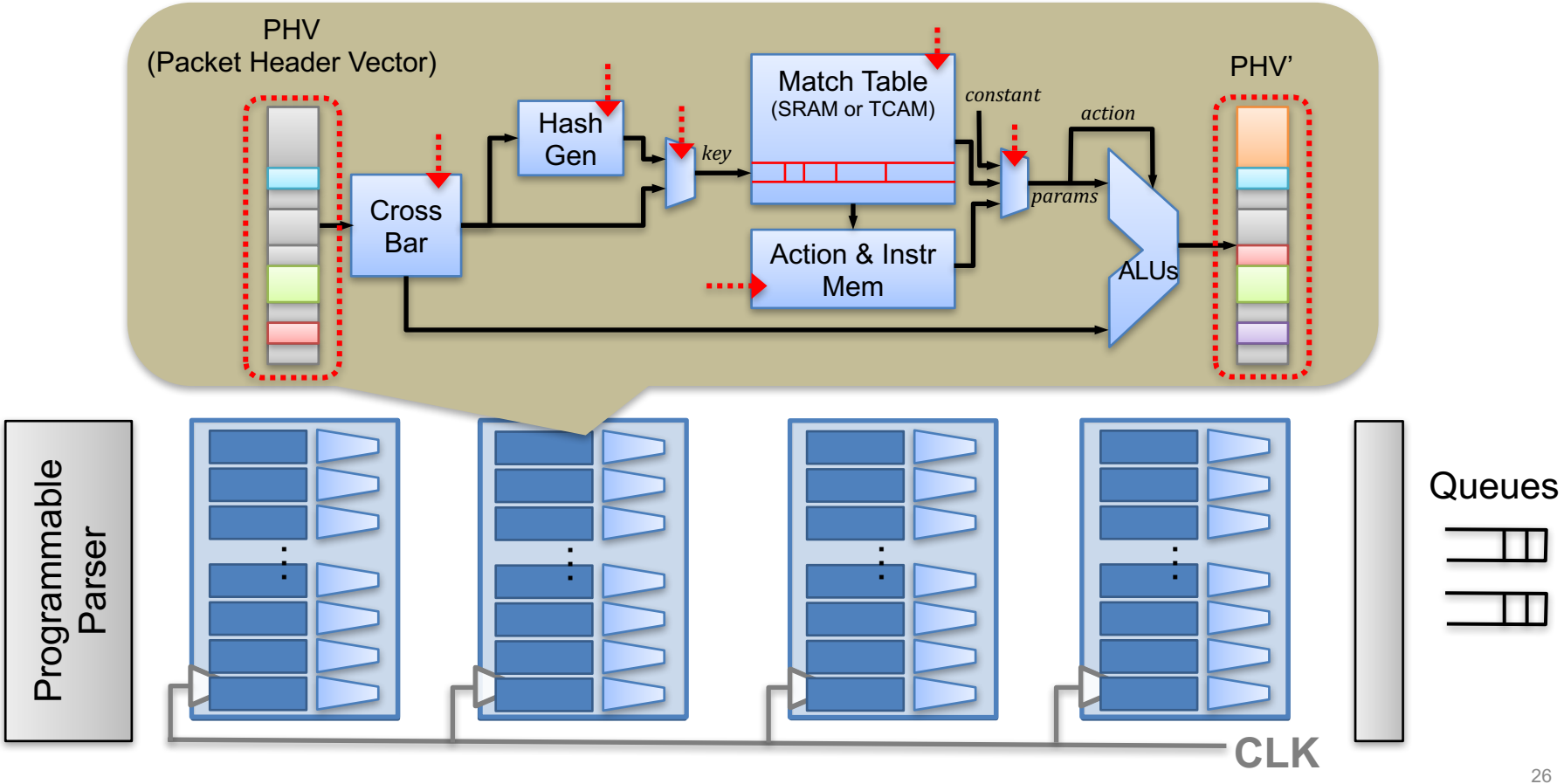
Field assembly

No: memory (pointers), loops, recursion, floating point

# Questions and critiques …?

- What does a compiler do?
- What's the latest on P4? Have you heard of $P4_{16}$?
- How do you update tables at runtime?
- Why is it important to derive a runtime API from a P4 program?
- What about queueing, scheduling, and congestion control?

# What exactly does a compiler do?



# $P4_{16}$: Why and how?

- **Embrace target heterogeneity without language churns**
  - Architectural heterogeneity via architecture-language separation
  - Functional heterogeneity via extern types
- **Help reuse code more easily: portability and composability**
  - Standard architecture and standard library
  - Local name space, local variables, lexical scoping, parameterization, and sub-procedure-like constructs
- **Make P4 programs more intuitive and explicit**
  - Expressions, sequential execution semantics for actions, strong type, and explicit de-parsing

# To recap: Why data-plane programming?

1. **New features**: Realize your beautiful ideas very quickly
2. **Reduce complexity**: Remove unnecessary features and tables
3. **Efficient use of H/W resources**: Achieve biggest bang for buck
4. **Greater visibility**: New diagnostics, telemetry, OAM, etc.
5. **Modularity**: Compose forwarding behavior from libraries
6. **Portability**: Specify forwarding behavior once; compile to many devices
7. **Own your own ideas**: No need to share your ideas with others

***"Protocols are being lifted off chips and into software"***
– Ben Horowitz

# What kind of “*stunt*” can you do by programming data planes?

- **Advanced network measurement, analysis, and diagnostics**
  - In-band Network Telemetry [SIGCOMM'15], Packet History [NSDI'14], FlowRadar [NSDI'16], Marple [SIGCOMM'17]
- **Advanced congestion control**
  - RCP, XCP, TeXCP, DCQCN++, Timely++
- **Novel DC network fabric**
  - Flowlet switching, CONGA [SIGCOMM'15], HULA [SOSR'16], NDP [SIGCOMM'17]
- **World's fastest middleboxes**
  - L4 connection load balancing [SIGCOMM'17], TCP SYN authentication, etc.
- **Offloading parts of the distributed apps**
  - NetCache [SOSP'17], NetChain [NSDI'18], SwitchPaxos [SOSR'15, ACM CCR'16]
- **Jointly optimizing network and the apps running on it**
  - Mostly-ordered Multicast [NSDI'15, SOSP'15]
- ***And many more …*** -- we're just starting to scratch the surface!

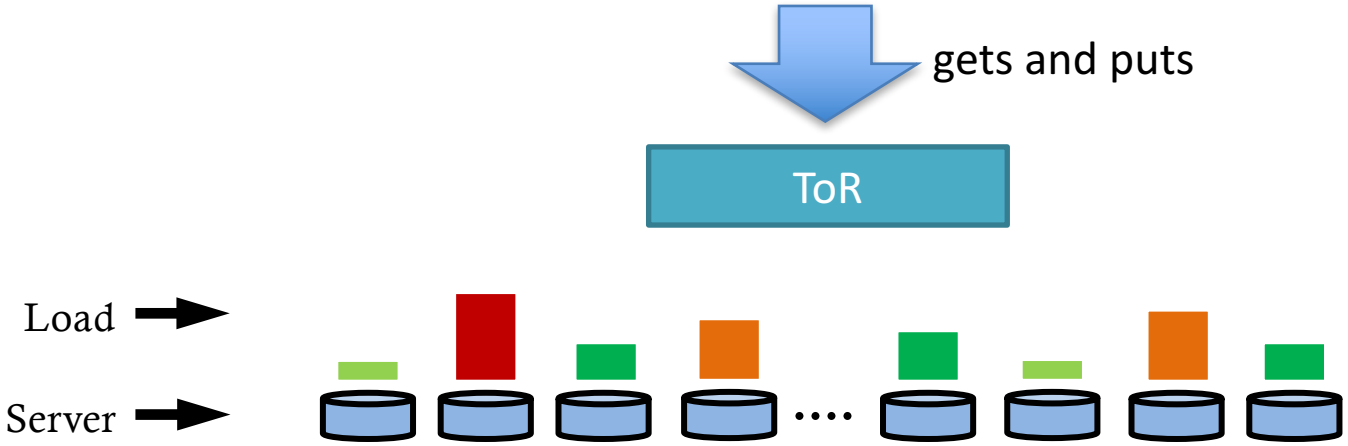
# PISA: An architecture for high-speed programmable ~~packet forwarding~~ I/O event processing

# What we have seen so far: Accelerating part of computing with PISA

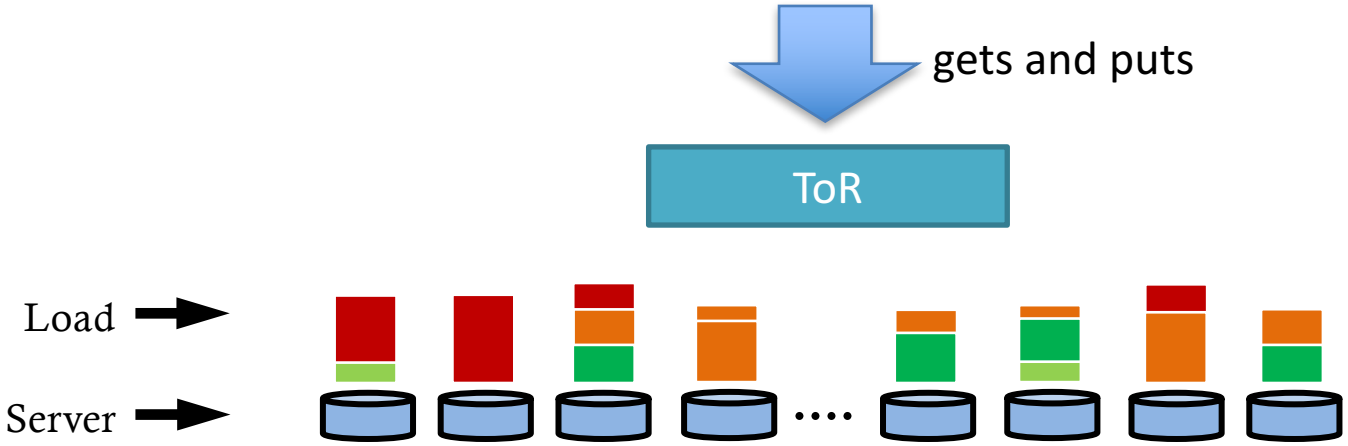
1. DNS cache
2. Key-value cache [NetCache - SOSP'17]
3. Key-value replication [NetChain - NSDI'18]
4. Consensus acceleration [P4xos - CCR'16, Eris - SOSP'17]
5. Parameter service for distributed deep learning
6. Pub-sub service
7. String searching [PPS – SOSR'19]
8. Pre-processing DB queries and streams

# *NetCache*: Accelerating KV caching

# Suppose a KV cluster coping with a highly-skewed & rapidly-changing workload

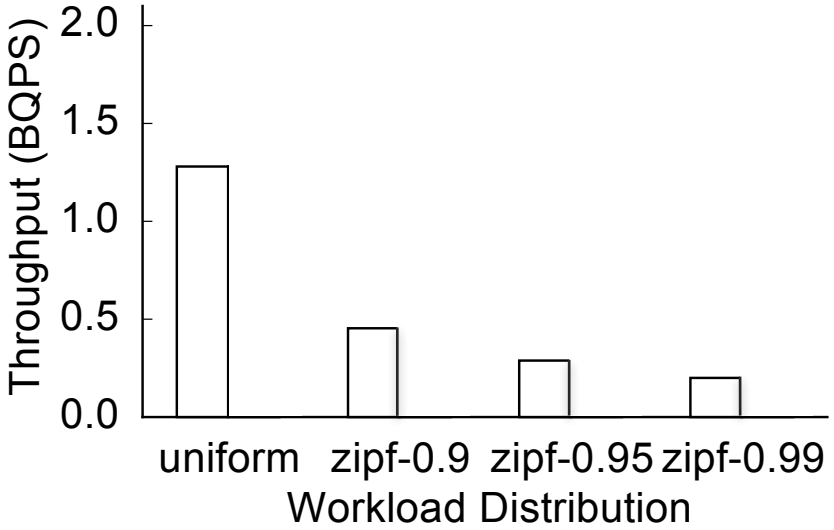


# Suppose a KV cluster coping with a highly-skewed & rapidly-changing workload

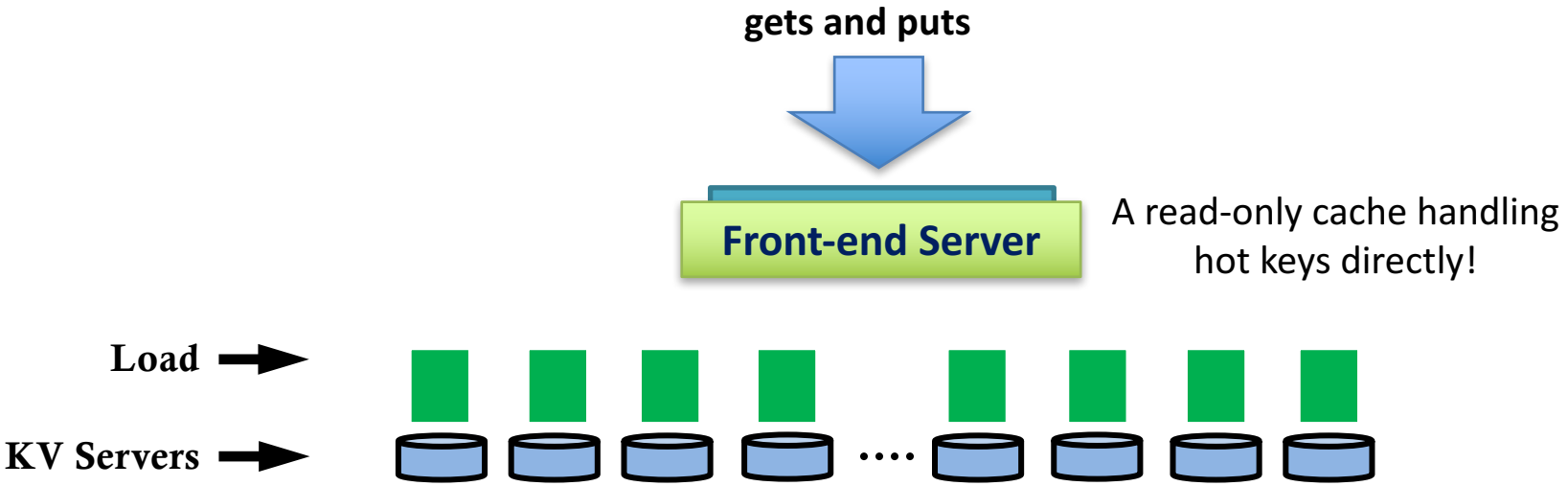



**Q: How can you ensure a high throughput and bound tail latency?**

# Here comes the problem



# What if we had a very fast front-end server?




*Q: How **big** and **fast** the front-end cache should be?*

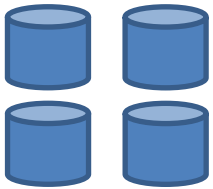
# For a front-end cache to be effective

- **How big should it be?**
  - Keep $O(N*logN)$ hot keys where $N$ is the number of KV servers
  - Theory proves that such a front-end cache bounds the variance of KV server utilization *irrespective* of the total number of keys

- **How fast should it be?**
  - At least as large as the aggregated throughput of all KV servers ($N*C$)

# Why is this relevant now?

Cache needs to provide the **aggregate** throughput of the storage layer




flash/disk

each: O(100) KQPS

**total: O(10) MQPS**

storage layer

cache

in-memory

**O(10) MQPS**

cache layer

in-memory

each: O(10) MQPS

**total: O(1) BQPS**

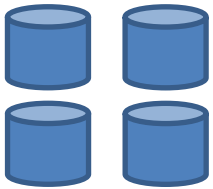
cache

?

**O(1) BQPS**

# Why is this relevant now?

Cache needs to provide the **aggregate** throughput of the storage layer




flash/disk

each: O(100) KQPS

**total: O(10) MQPS**

storage layer

cache

in-memory

**O(10) MQPS**

cache layer

in-memory

each: O(10) MQPS

**total: O(1) BQPS**

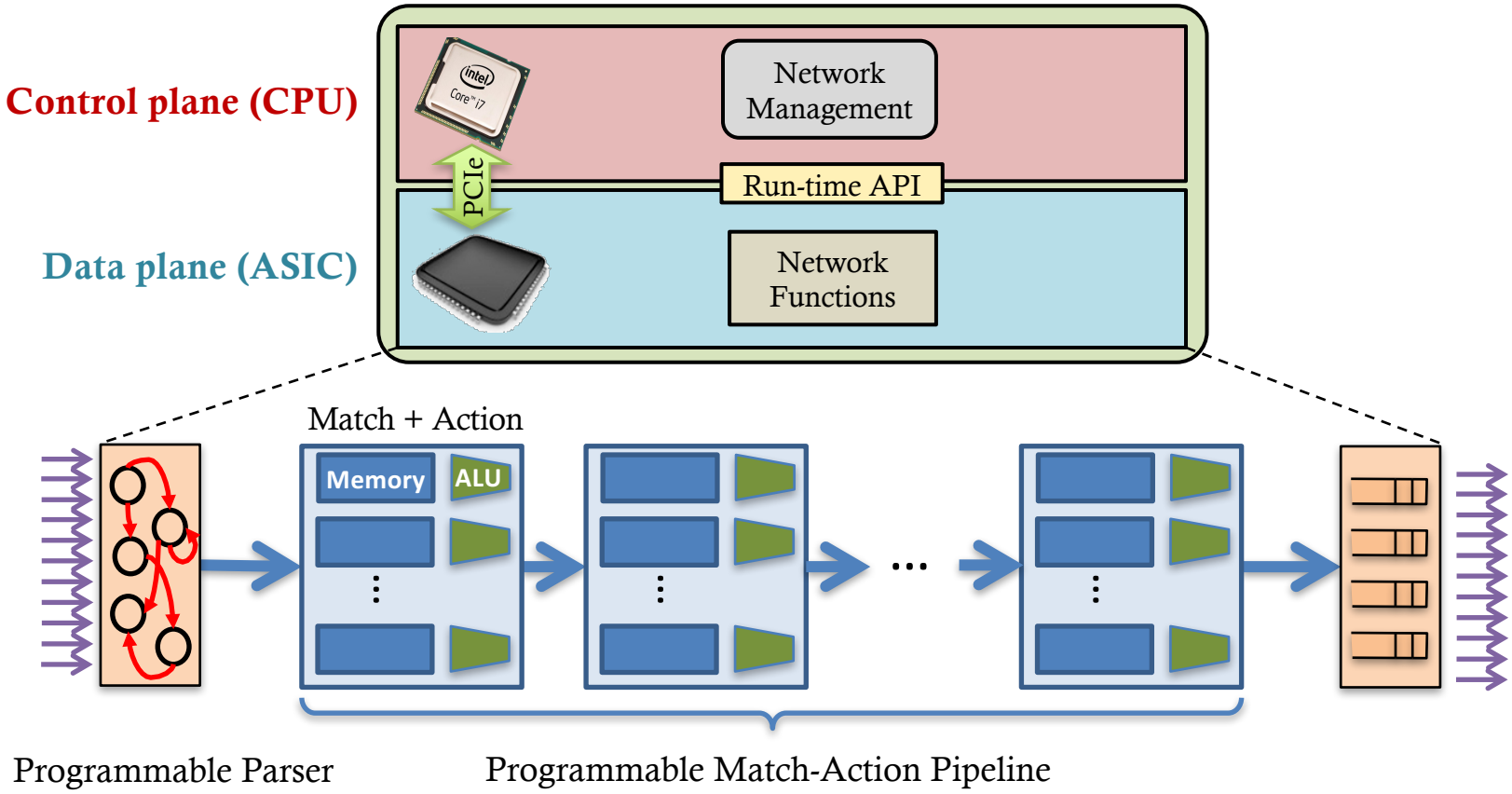
cache

**PISA (real-time I/O machine)**

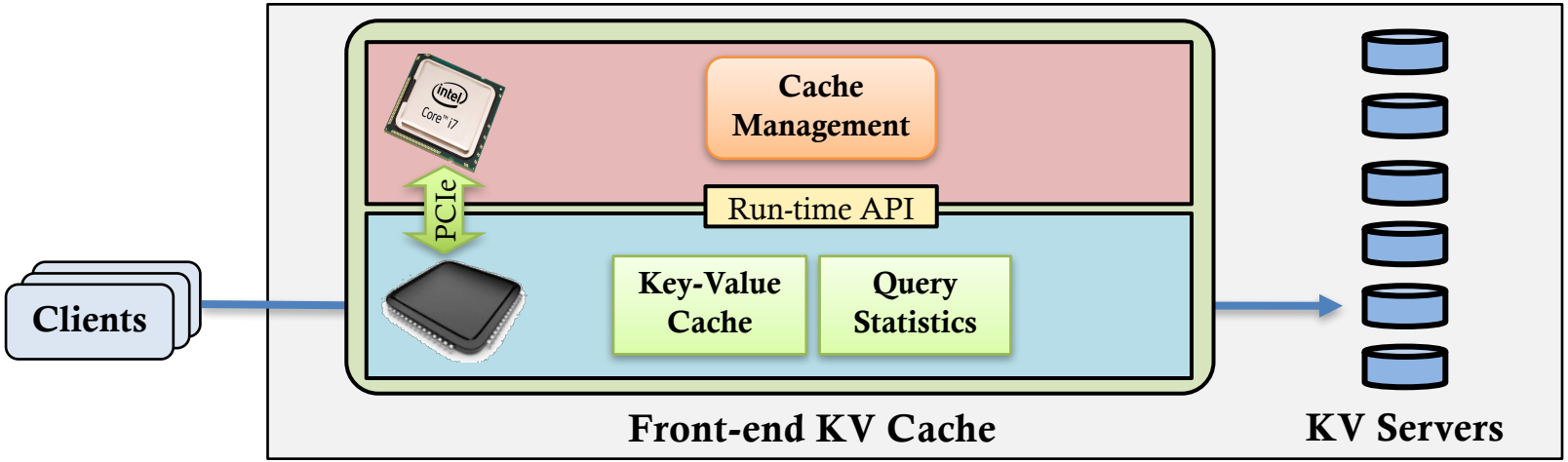
**O(1) BQPS**

Small on-chip memory?
Only cache **O(*N* log *N*) small** items

# A conventional switch built with PISA

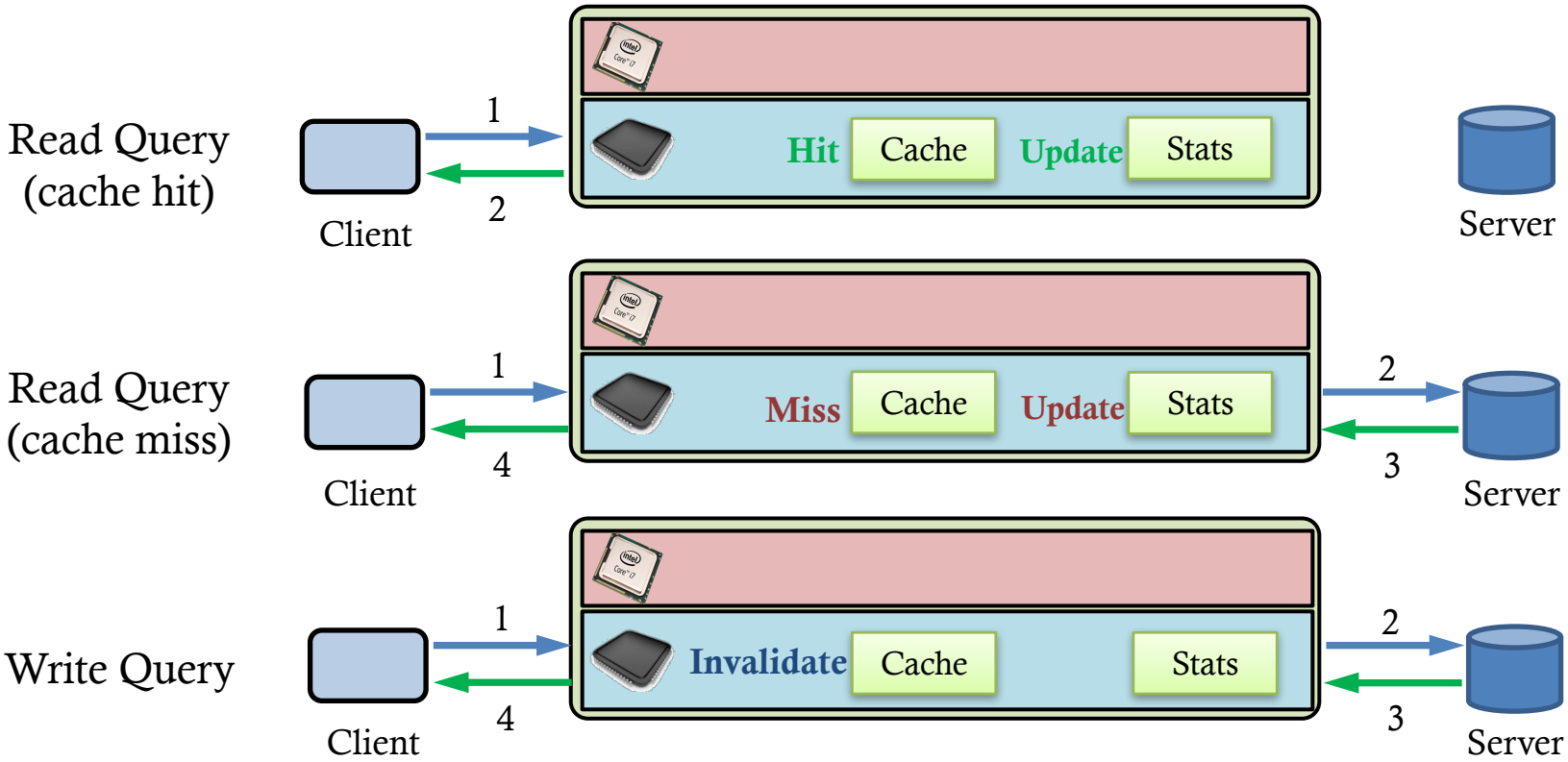


# A front-end KV cache built with PISA

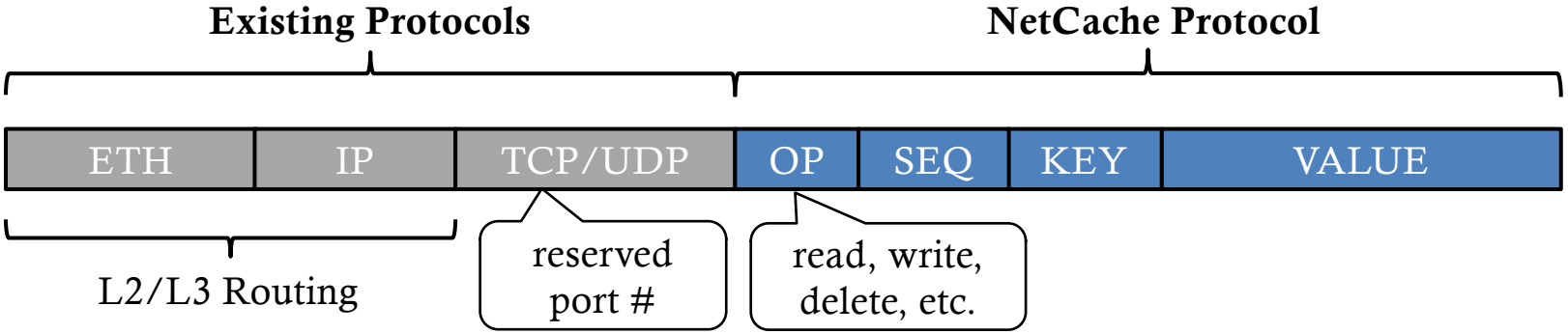



- **Data plane**
  - Key-value store to serve queries for cached keys
  - Query statistics to enable efficient cache updates
- **Control plane**
  - Insert hot items into the cache and evict less popular items
  - Manage memory allocation for on-chip key-value store

# Line-rate query handling in the data plane



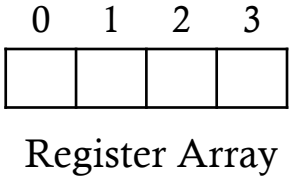
# Packet format




- Application-layer protocol; compatible with existing L2-L4 layers
- Only the front-end cache needs to parse NetCache fields

# Key-value store using register array in network ASIC

```
action process_array(idx):
  if pkt.op == read:
    pkt.value ← array[idx]
  elif pkt.op == cache_update:
    array[idx] ← pkt.value
```

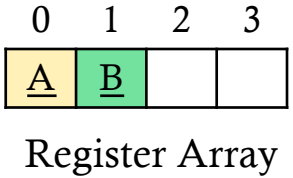


# Key-value store using register array in network ASIC

| Match | pkt.key == A | pkt.key == B |
|---|---|---|
| Action | **process_array**(0) | **process_array**(1) |

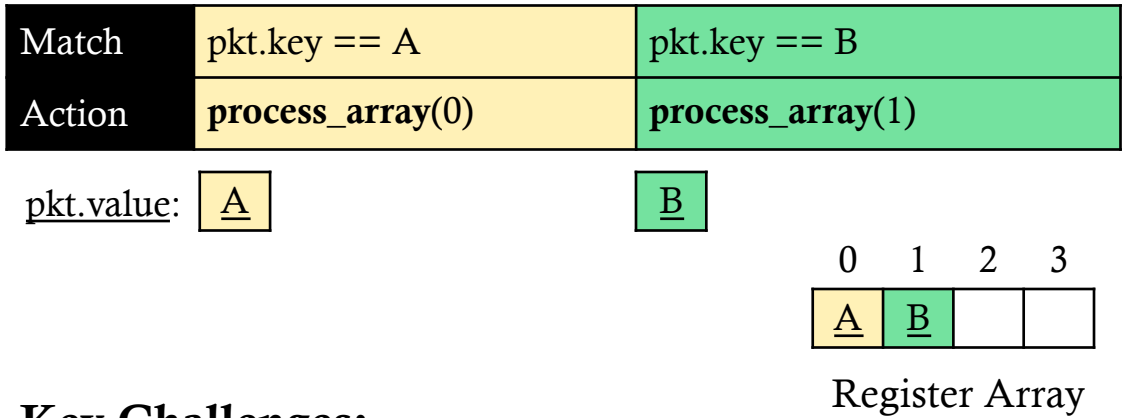
pkt.value: A B

```
action process_array(idx):
  if pkt.op == read:
    pkt.value ← array[idx]
  elif pkt.op == cache_update:
    array[idx] ← pkt.value
```




Register Array

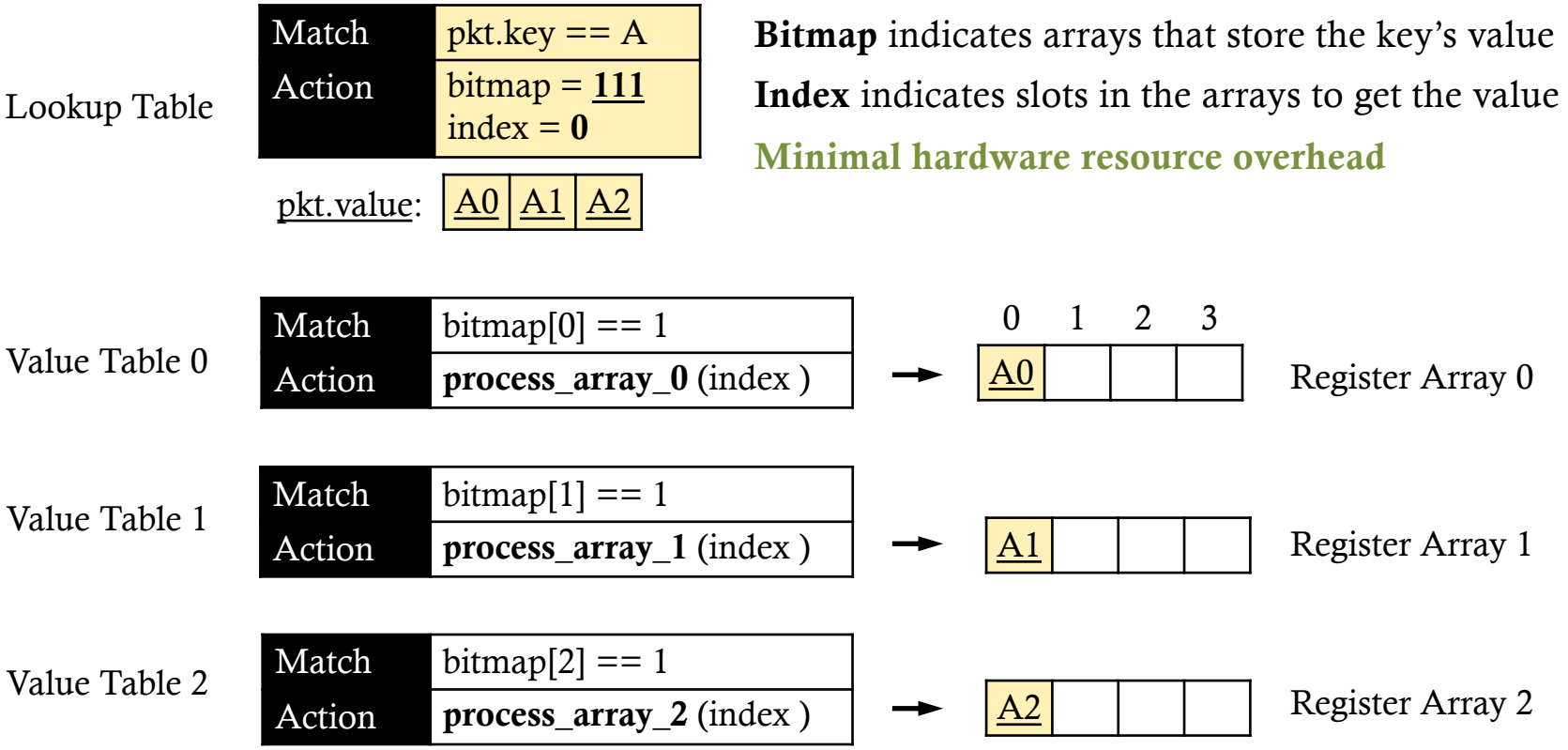
# Variable-length key-value store in network ASIC?



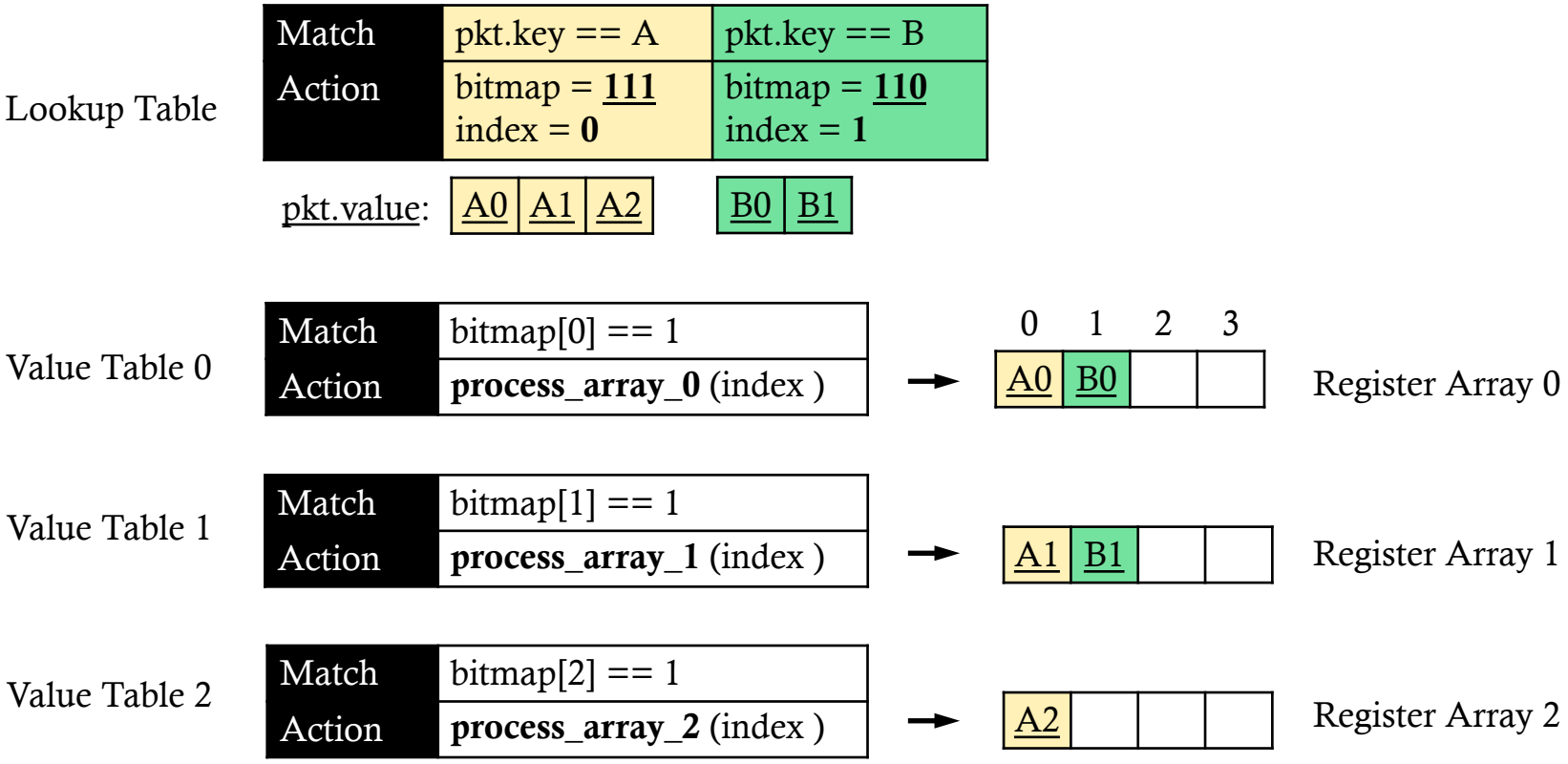

**Key Challenges:**

- No loop or string due to strict timing requirements
- Need to minimize hardware resources consumption
  - Number of table entries
  - Size of action data for table each entry
  - Size of intermediate metadata across tables

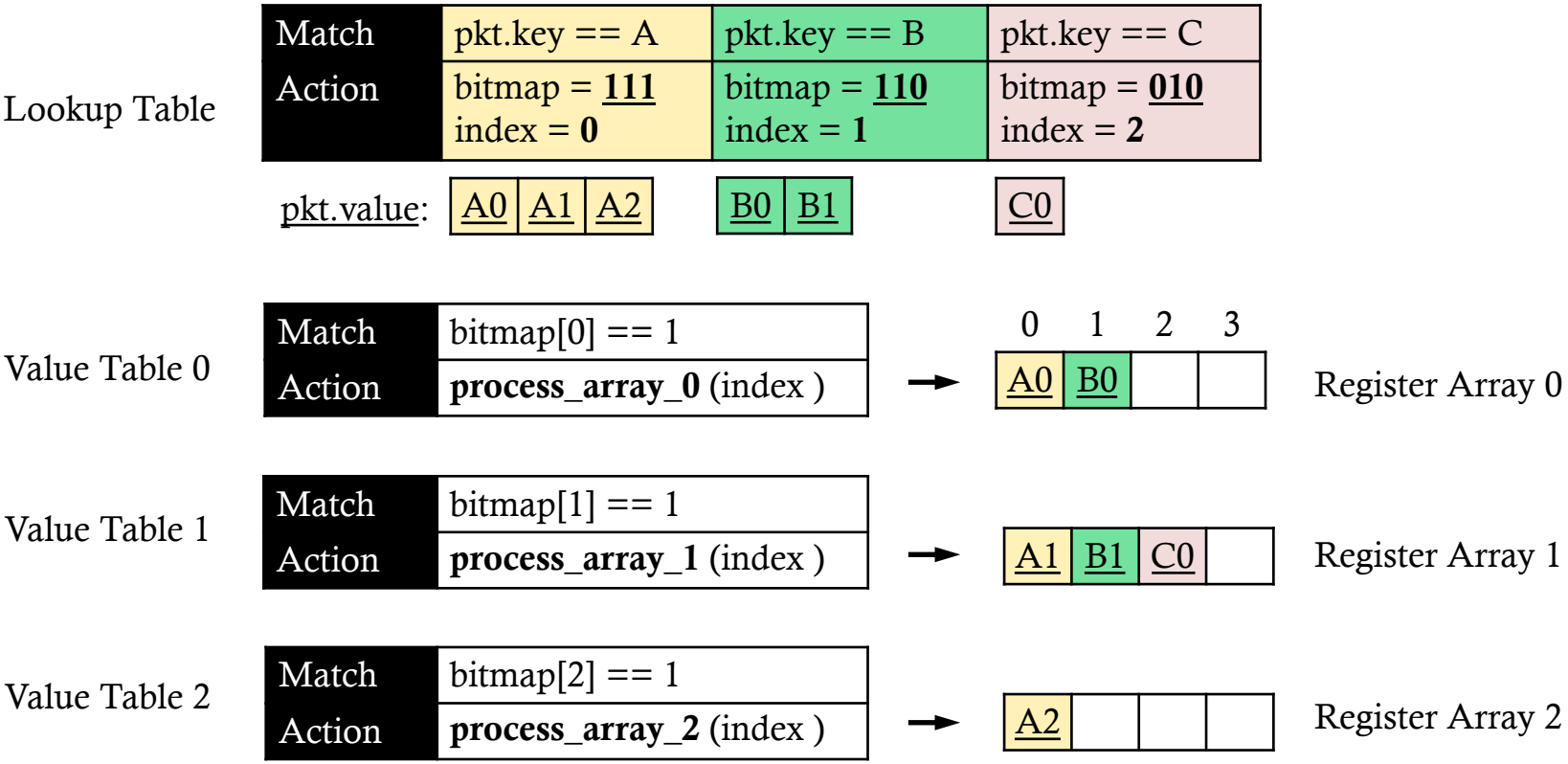
# Combine outputs from multiple arrays



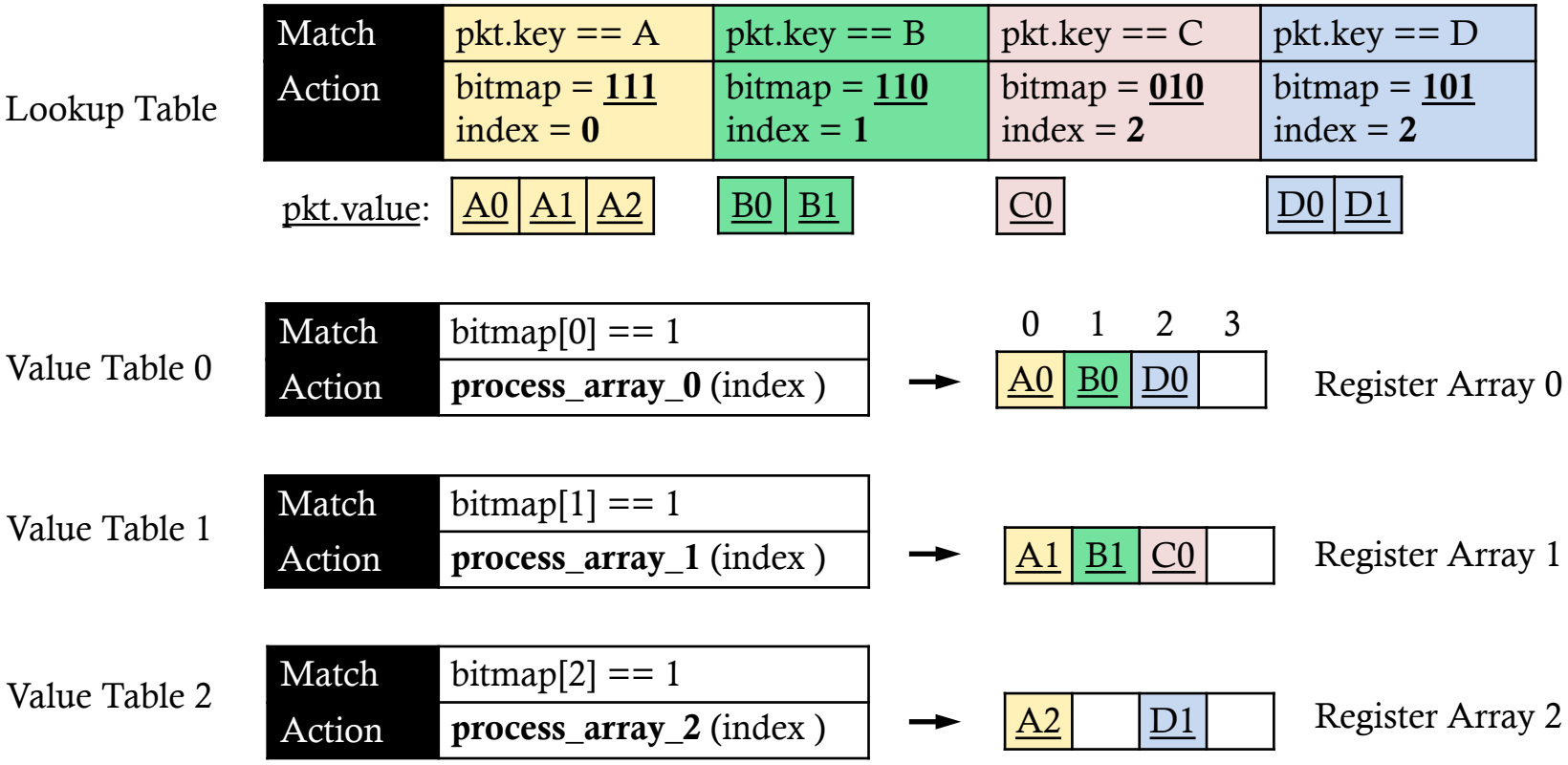
# Combine outputs from multiple arrays



# Combine outputs from multiple arrays

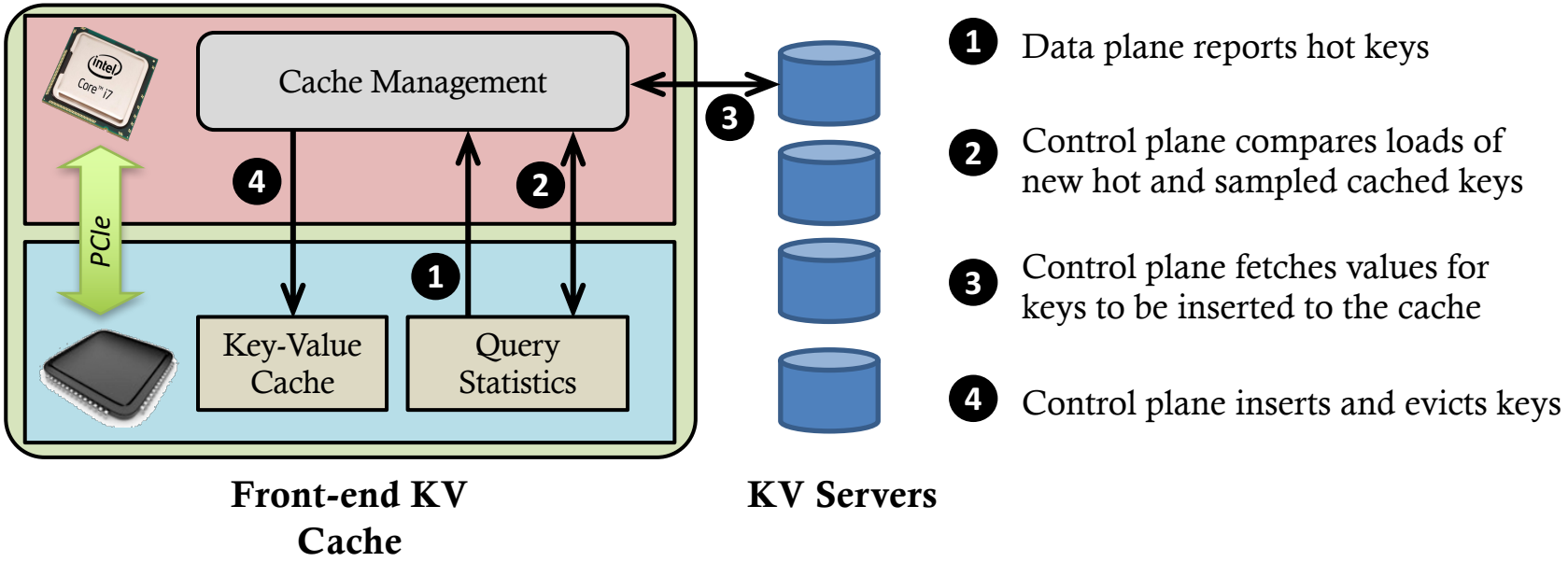


# Combine outputs from multiple arrays

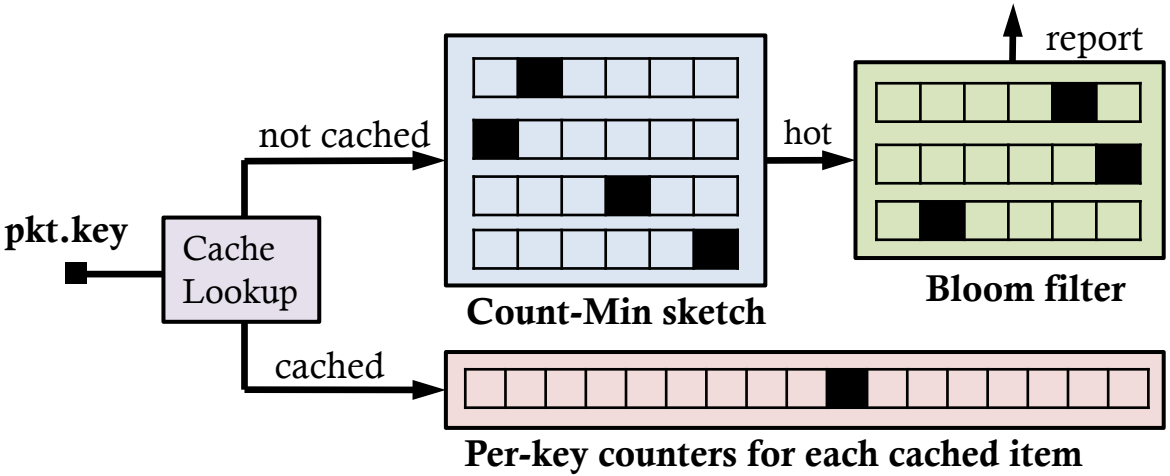


# Cache insertion and eviction

- Challenge: Keeping the **hottest** O($N$ log$N$) items in the cache
- Goal: React quickly and effectively to workload changes with **minimal updates**

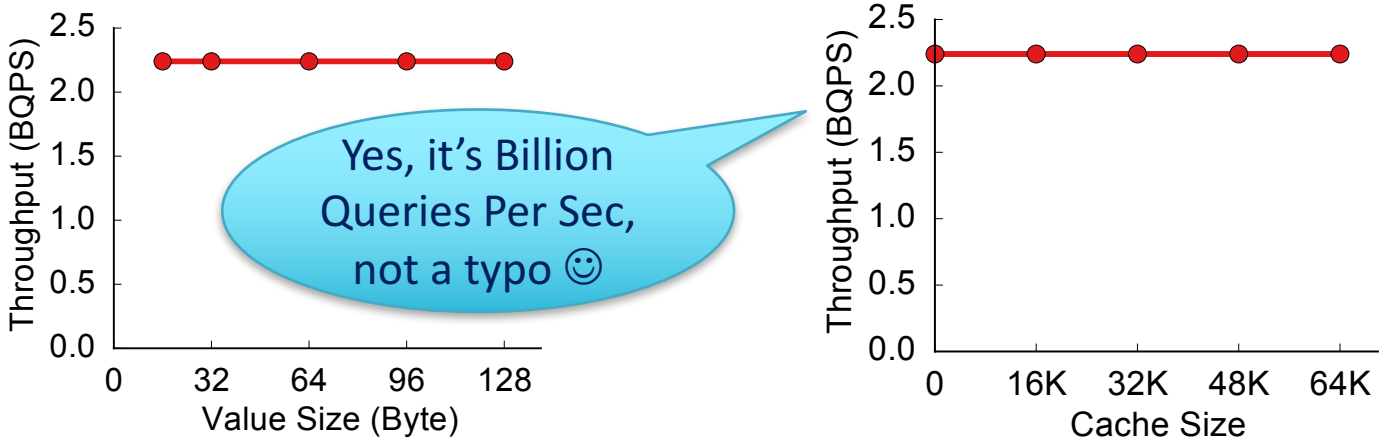


# Query statistics in the data plane




- Cached key: per-key counter array
- Uncached key
  - Count-Min sketch: report new hot keys
  - Bloom filter: remove duplicated hot key reports

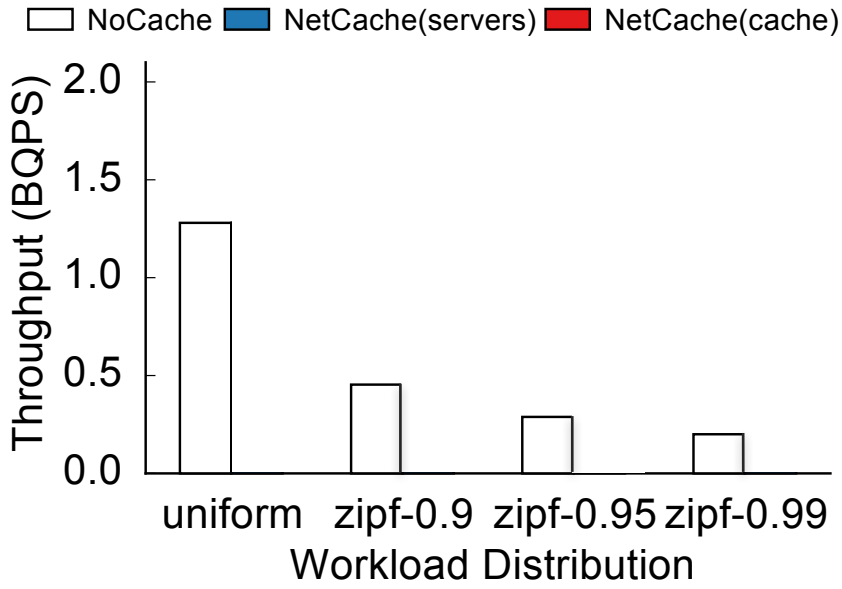
# The “boring life” of a NetCache switch




One can further increase the value sizes with more stages, recirculation, or mirroring.

# And its “not so boring” benefits

Throughput of a key-value storage rack with one Tofino switch and 128 storage servers.




NetCache provides **3-10x throughput improvements**.

# Questions and critiques …?

- Conflating “in-network computing” with “accelerating apps using PISA”
- What are the common and unique strengths of PISA for those offload-able apps?
- Will the “stunt” become a main stream approach? What would be the triggers for that?
- Is P4 the right way of implementing PISA-server apps?

# Some observations

- PISA and P4: The first attempt to define a machine architecture and programming models for networking in a ***disciplined*** way
- Inherently multi-disciplinary; we need more expertise across various fields in computer science
- It's super fun to figure out the best workloads for this new machine architecture

# Want to find more resources or follow up?

- Visit http://p4.org and http://github.com/p4lang
  - P4 language spec
  - P4 dev tools and sample programs
  - P4 tutorials
  - List of papers regarding PISA, PISA Apps, and P4
- Join P4 workshops and P4 developers' days
- Participate in P4 working group activities
  - Language, target architecture, runtime API, applications
- Need more expertise across various fields in computer science
  - To enhance PISA, P4, dev tools (e.g., for formal verification, equivalence check, automated test generation, and many more …)