The Network is The Computer: Running Distributed Services on Programmable Switches

Robert Soulé
Università della Svizzera italiana
and Barefoot Networks

Conventional Wisdom

- The network is “just plumbing”
- Teach systems grad students the end-to-end principle [Saltzer, Reed, and Clark, 1981]
- Programmable networks are too expensive, too slow, or consume too much power

This Has Changed

A new breed of switch is now available:

- They are programmable
- No power or cost penalties
- They are just as fast as fixed-function devices (6.5 Tbps!)*

* *Yes, I work at Barefoot Networks.*

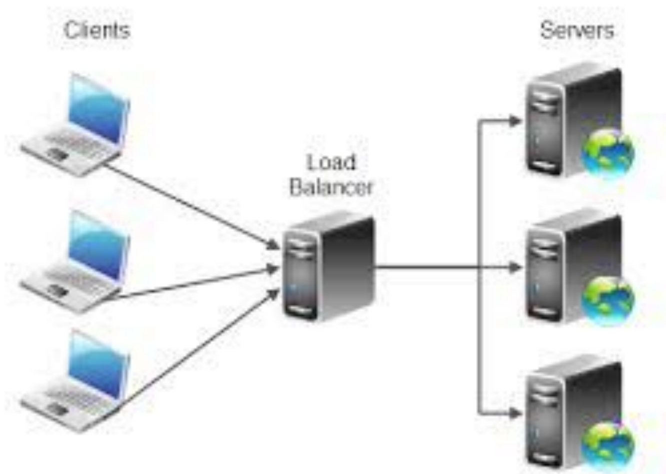
If This Trend Continues…

Java OpenCL MatLab TensorFlow ?

CPUs GPUs DSPs TPUs ASICs

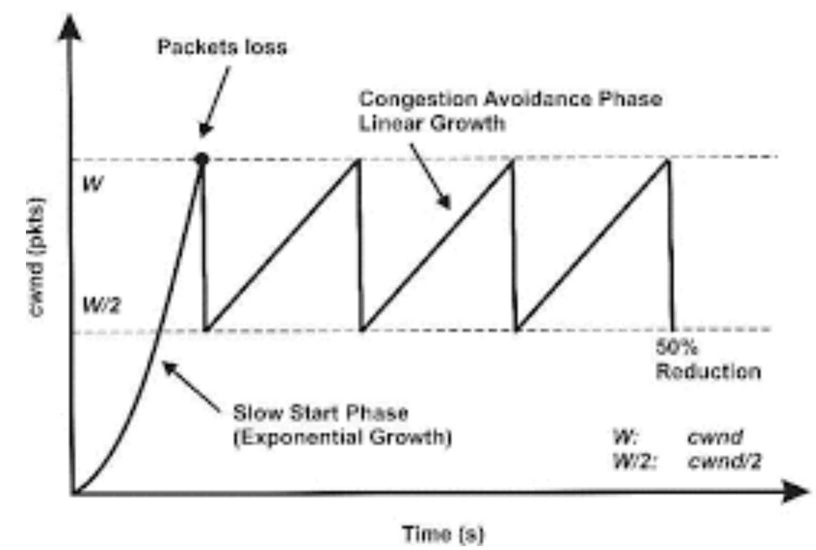
Programmable ASICs will replace fixed-function chips in data centers

What Functionality Belongs in the Network?



Load Balancing

Firewall



Congestion Control

Tremendous Opportunity

Fault-tolerance

Key-Value Store

Stream Processing

Run important, widely used distributed services in the network

Tremendous Opportunity

Fault-tolerance

A 10,000x improvement in throughput

[NetPaxos SOSR ’15, P4xos CCR ’16]

Tremendous Opportunity

2 billion

queries / second

with 50%

reduction

in latency

Key-Value Store

[NetCache, NSDI ’17]

Tremendous Opportunity

Process

4 billion

events

per second.

Stream Processing

[Linear Road, SOSR '18]

Key Questions

This sounds good on paper, but…

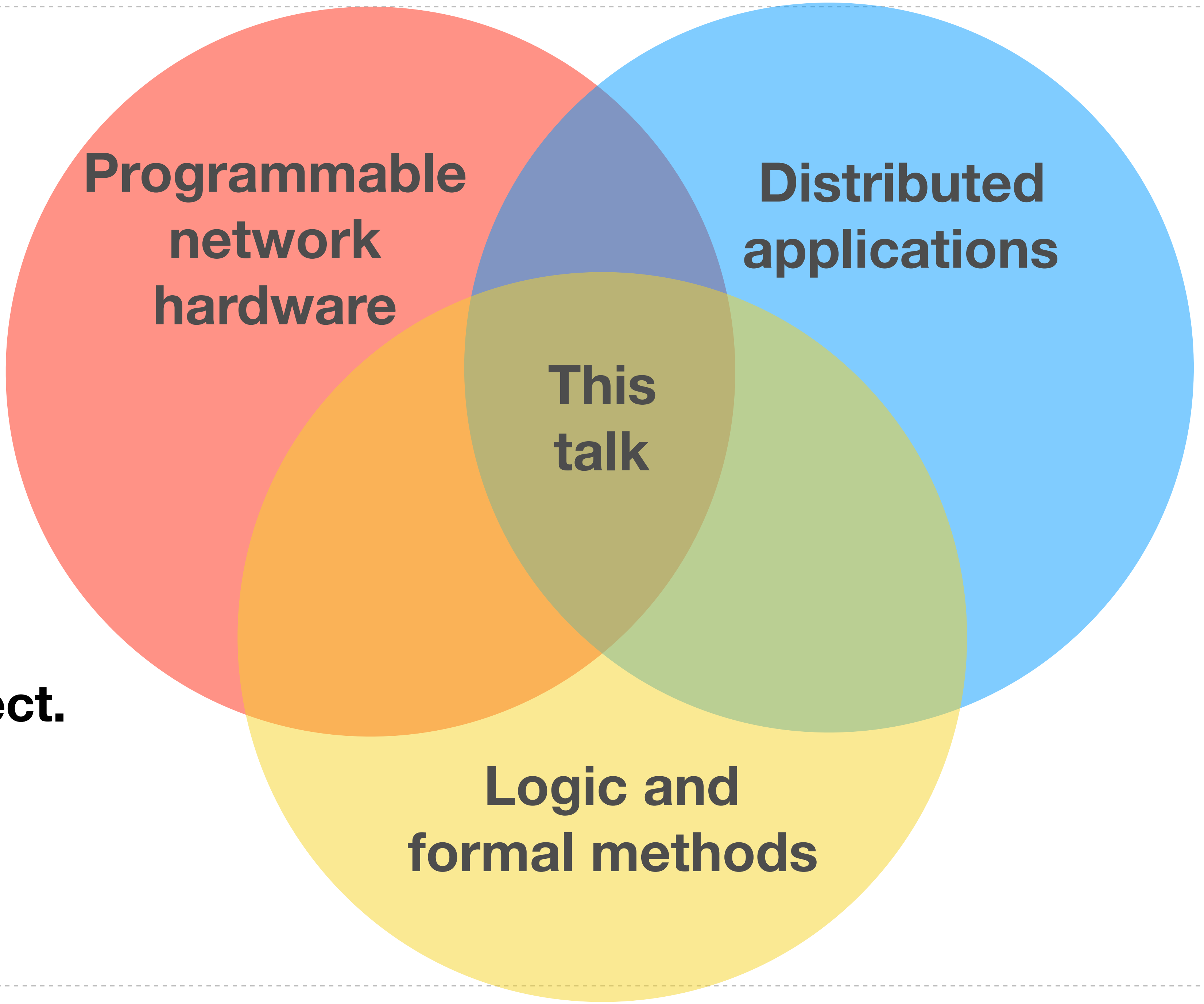
- How do we actually program network devices?
  What are the limitations? What are the abstractions?
- What (parts of) applications could or should be in the network?
  What is the right architecture?
- Given that we are asking the network to do so much more work,
  how can we be sure that it is implemented correctly?

Agenda and Tools

Leverage emerging hardware…

… to accelerate distributed services…

… and prove that the implementations are correct.

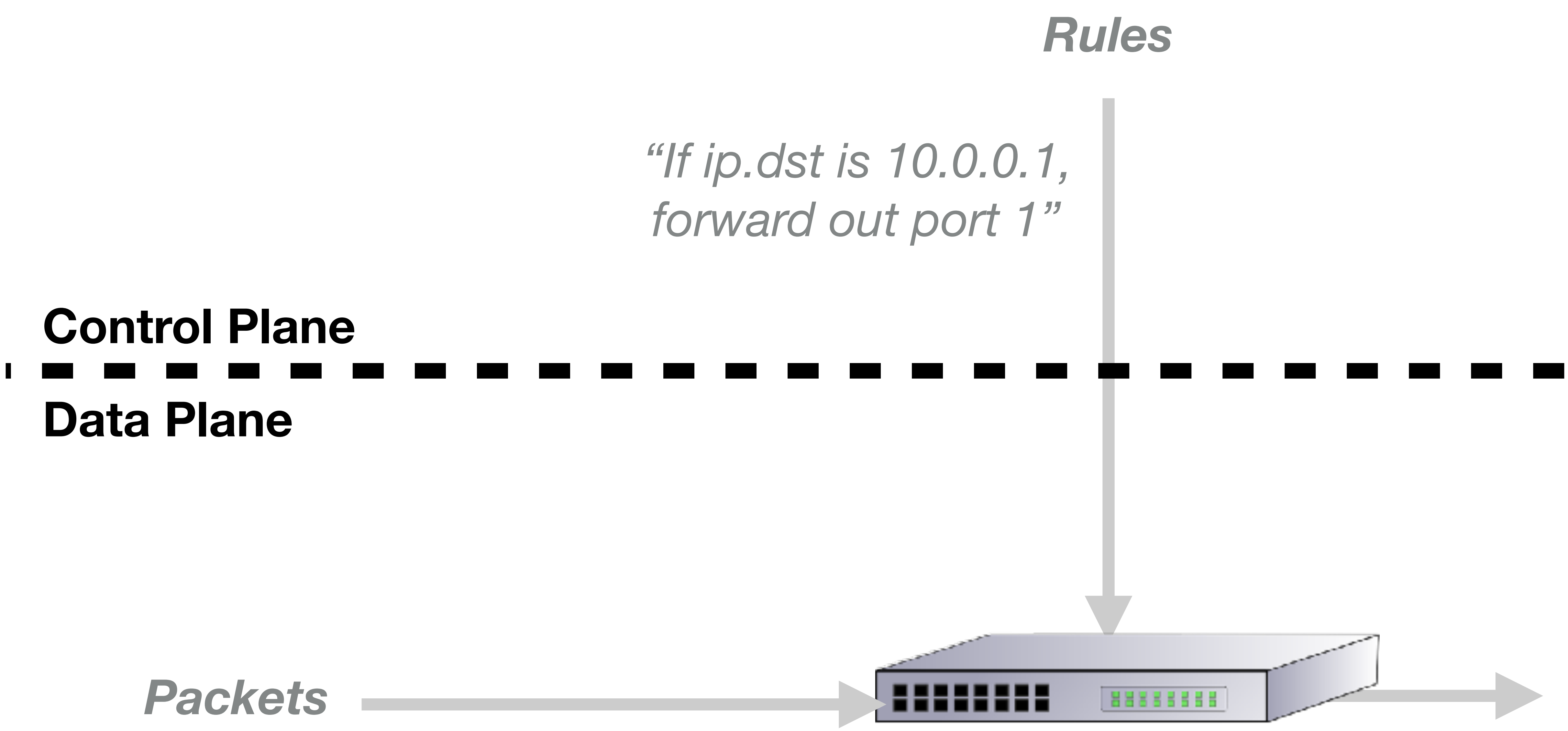


Outline of This Talk

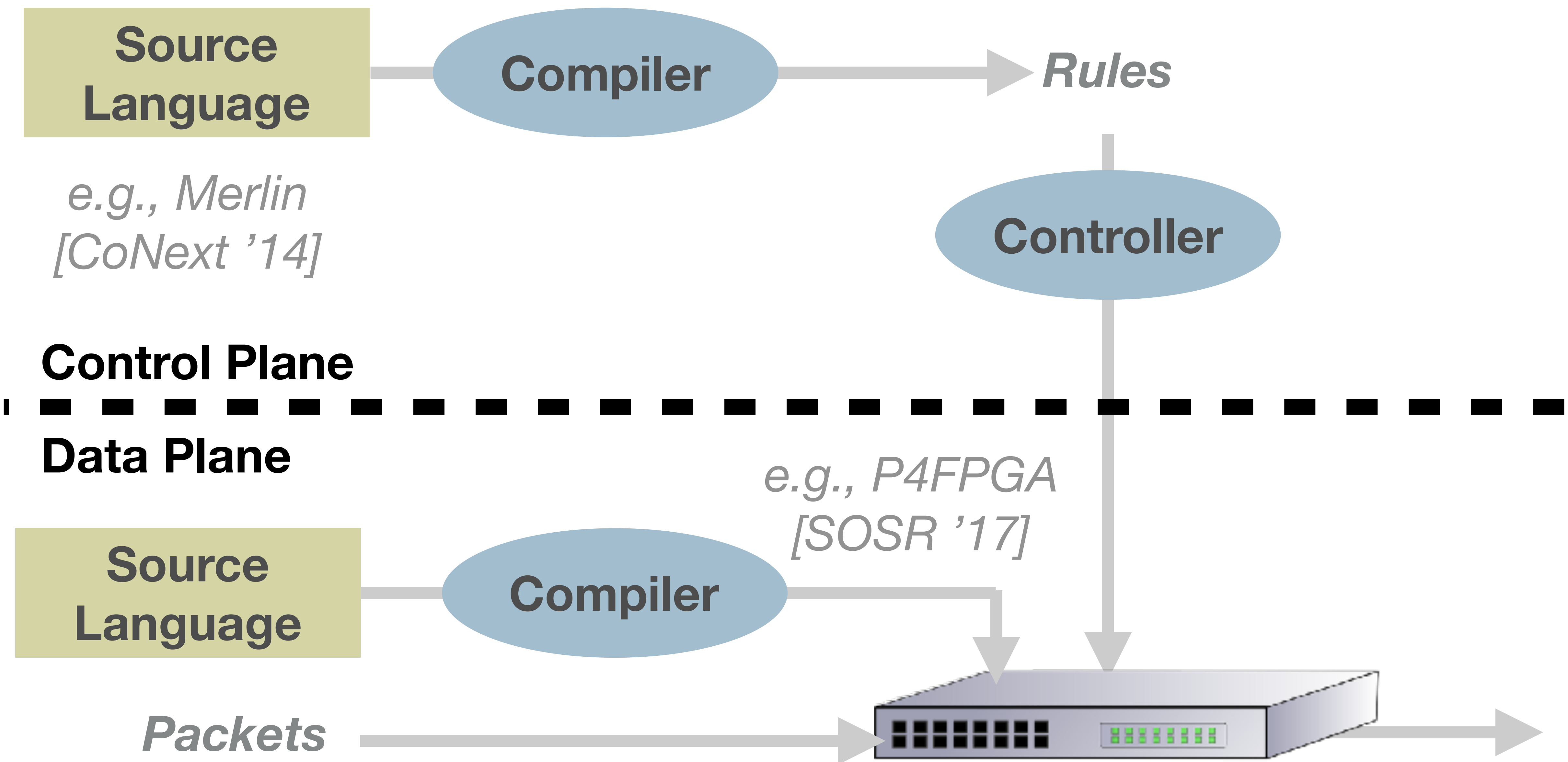
- Introduction
- **Programmable Network Hardware**
- Co-designing Networks
  and Distributed Systems
- Proving Correctness
- Outlook

Programmable Network Hardware

What is A Programmable Network?



What is A Programmable Network?



Match Action Table

Match	Action

Data plane programming specifies:
- fields to read
- possible actions
- size of table

Main abstraction for data plane programming

Match Action Table

Match	Action
10.0.0.1	Drop
10.0.0.2	Forward out 1
10.0.0.3	Forward out 2
10.0.0.4	Modify header

Control plane programming specifies the rules in the table

Match Action Unit

Match

- SRAM for exact match
- TCAM for ternary match

Match Action Unit

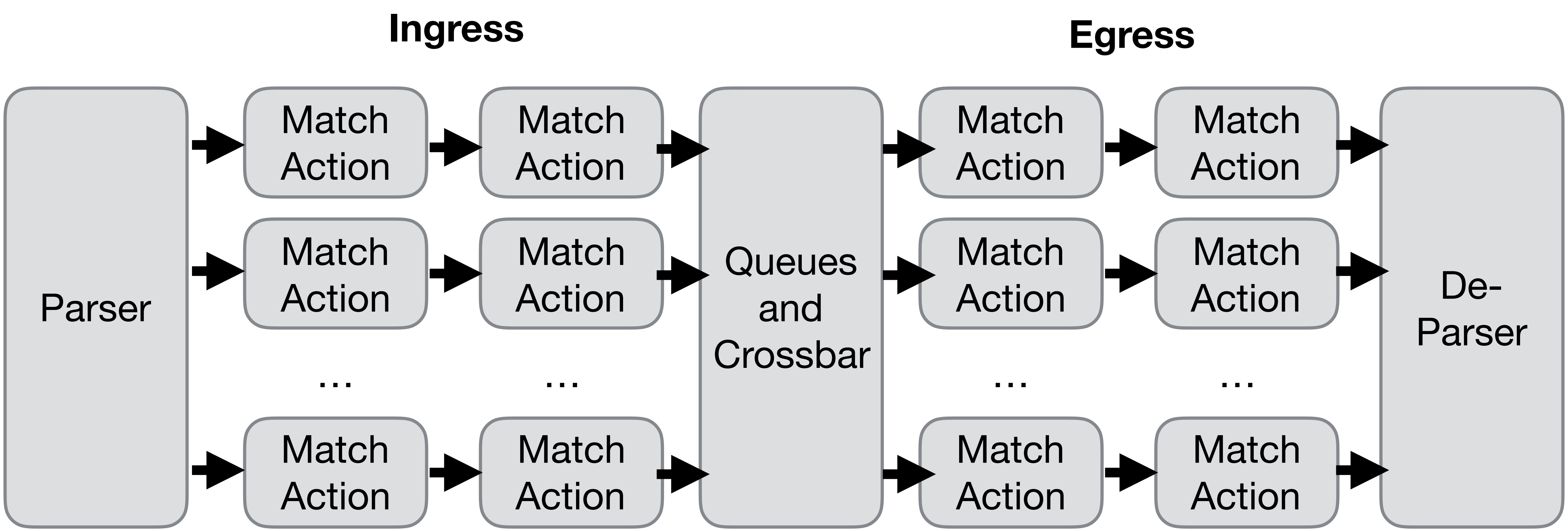
Action

- Stateless ALU
  - Limited instruction set
  - Arithmetic operations
  - Bitwise operations
- Stateful ALU
  - Counters
  - Meters

Massively Parallelized:

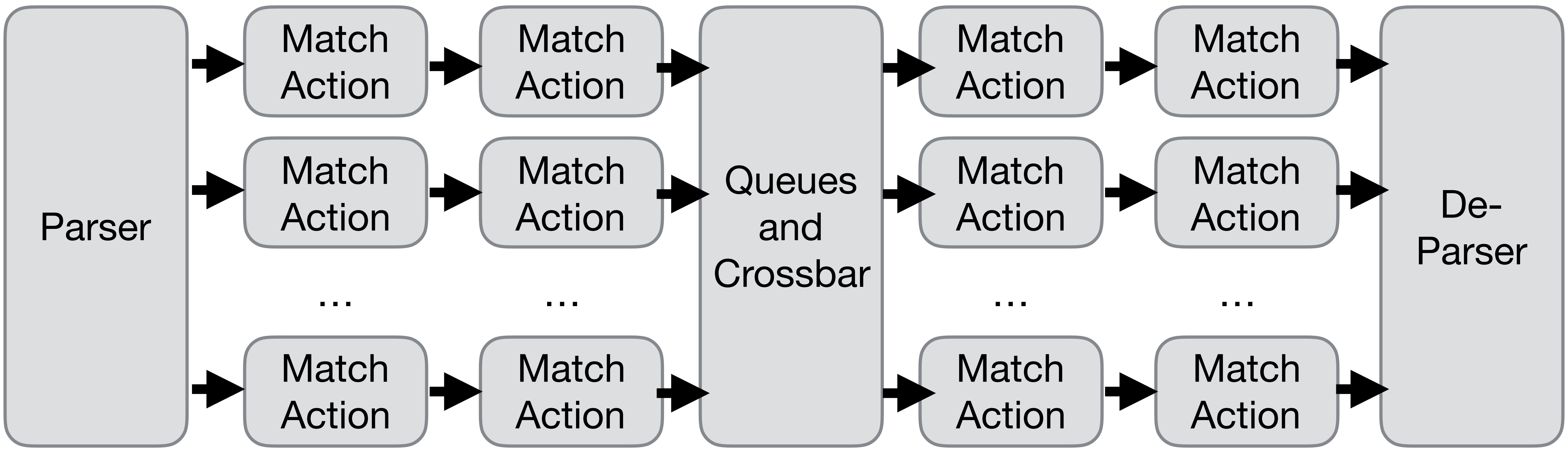
- Data Parallelism for performance
- Pipelined stages for data dependencies

Programmable Data Plane

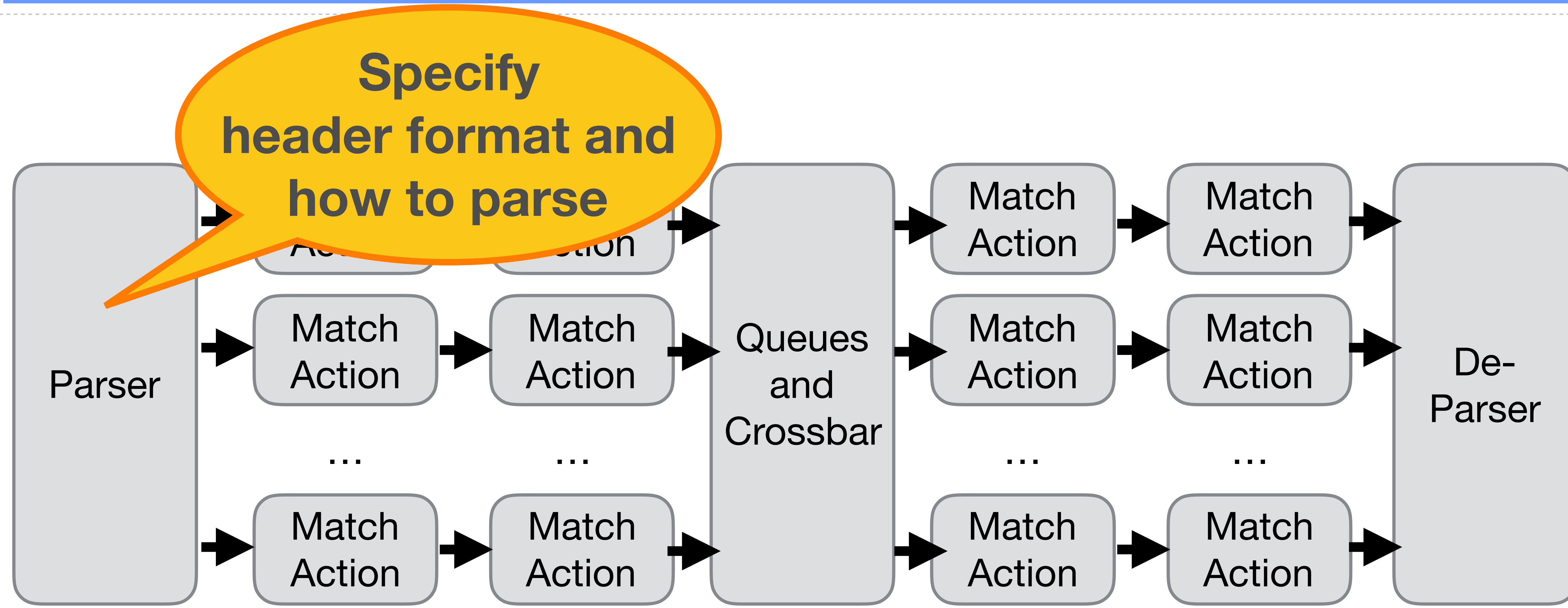


Programmable ASIC Architecture

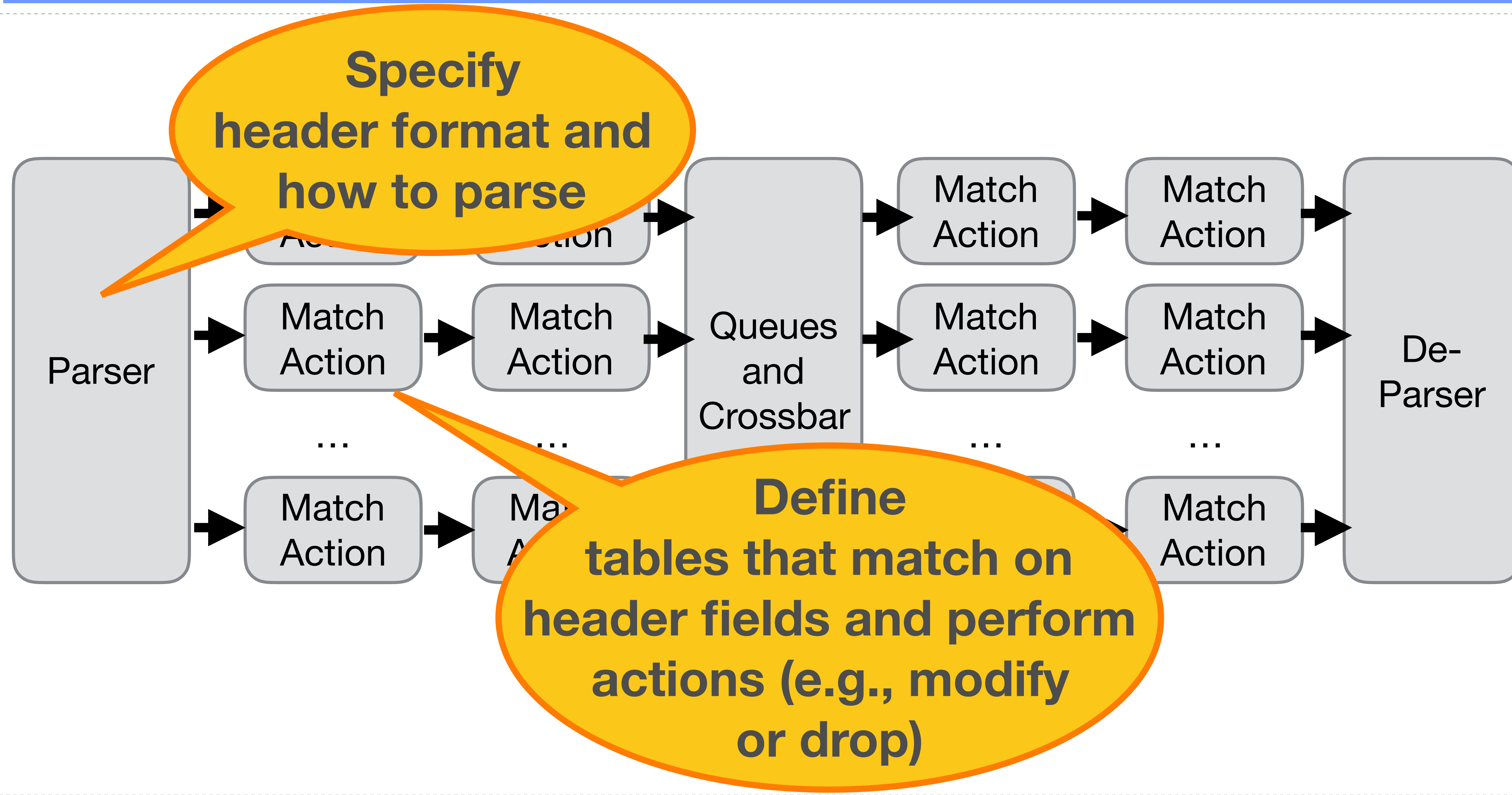
P4 Language Concepts



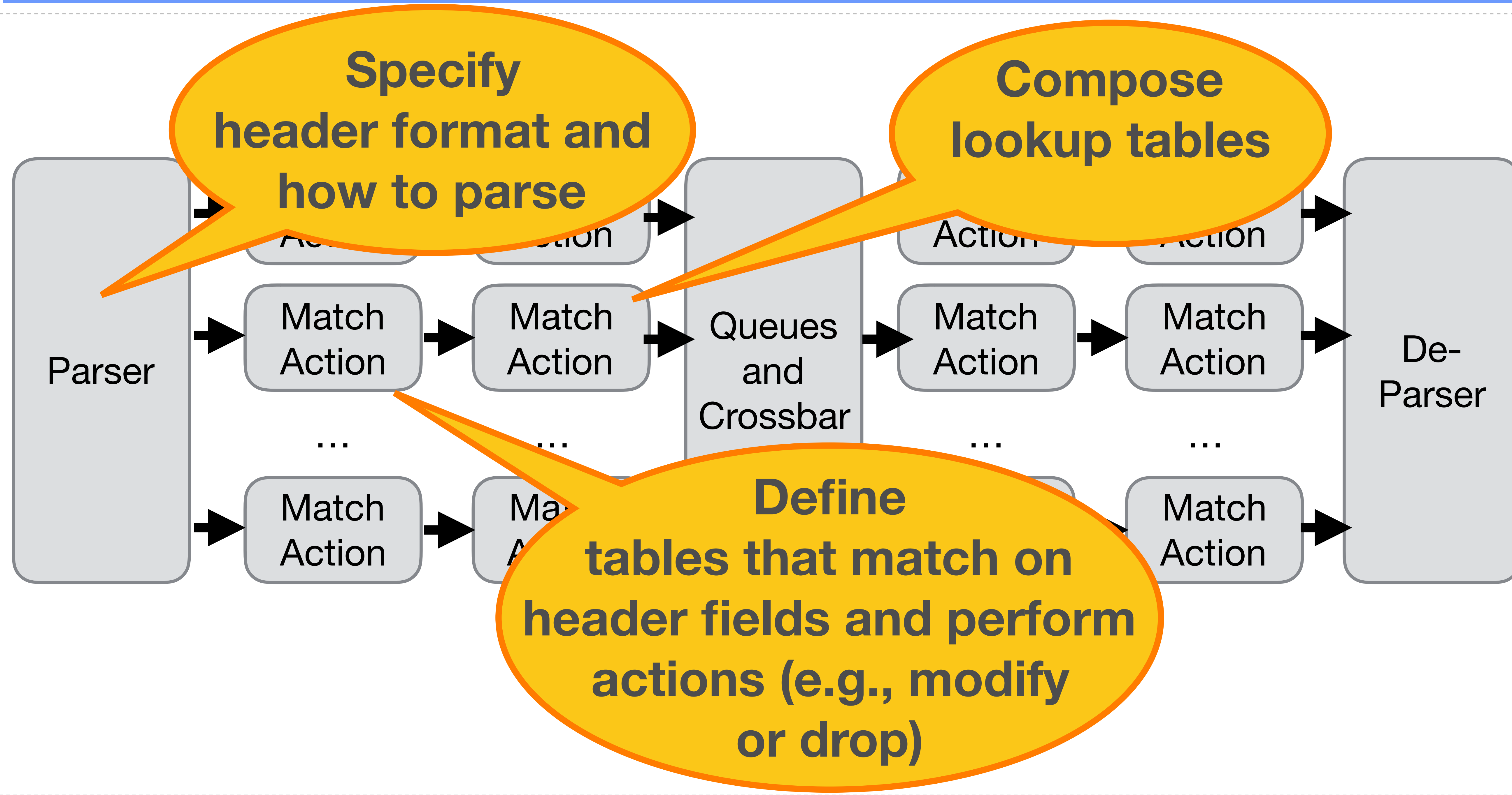
P4 Language Concepts



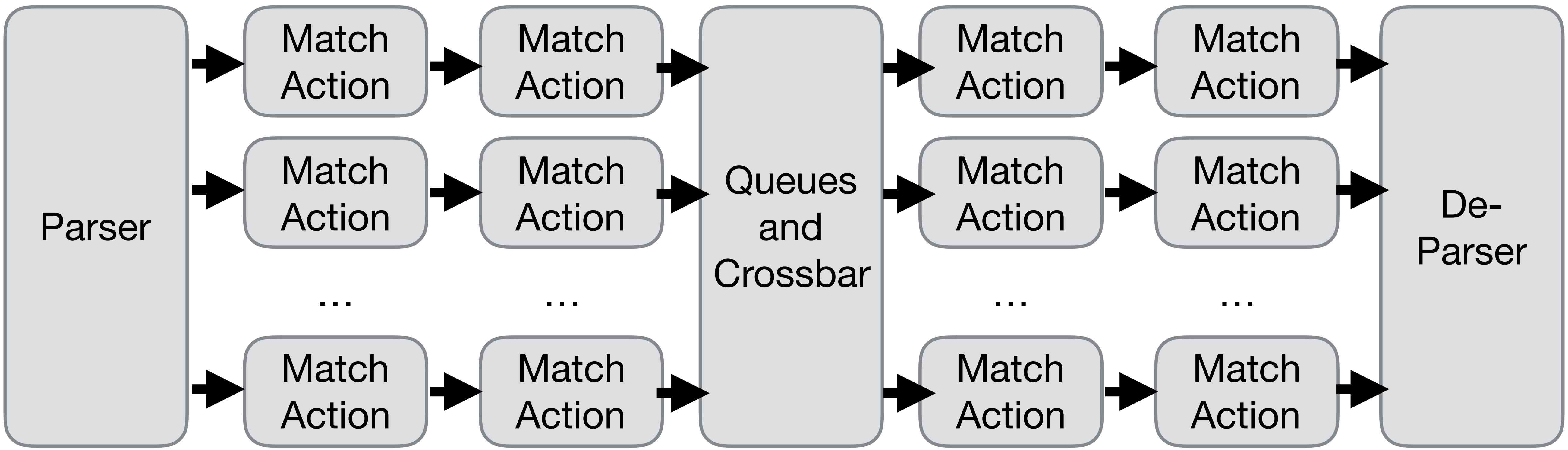
P4 Language Concepts



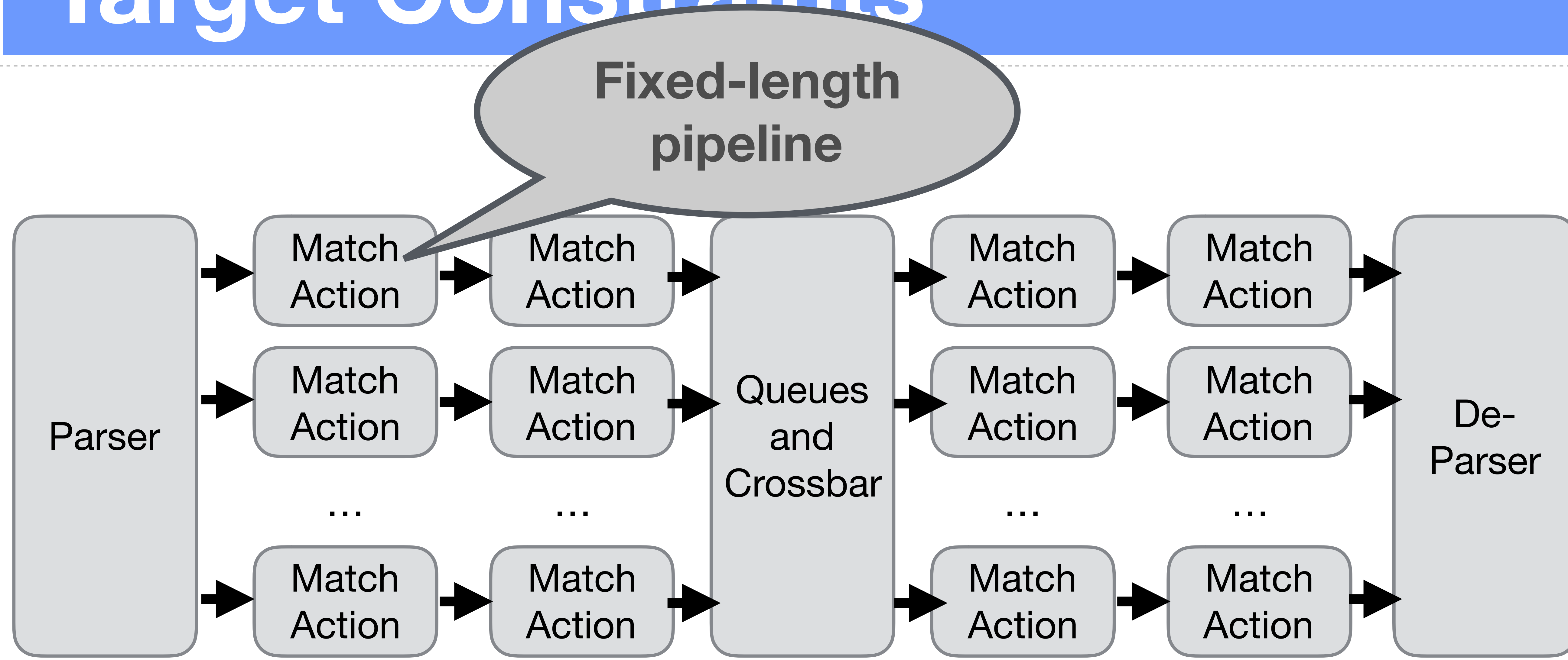
P4 Language Concepts



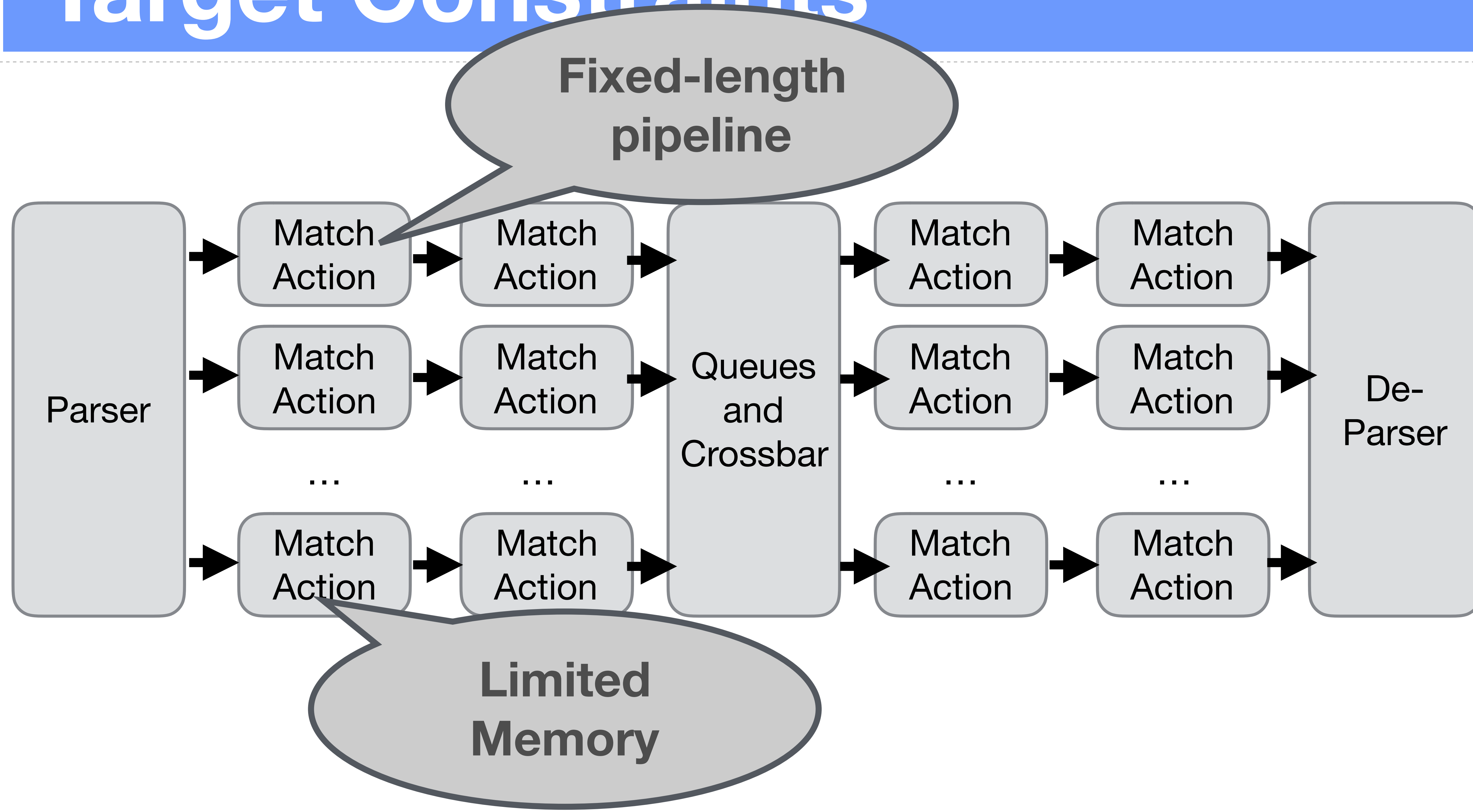
Target Constraints



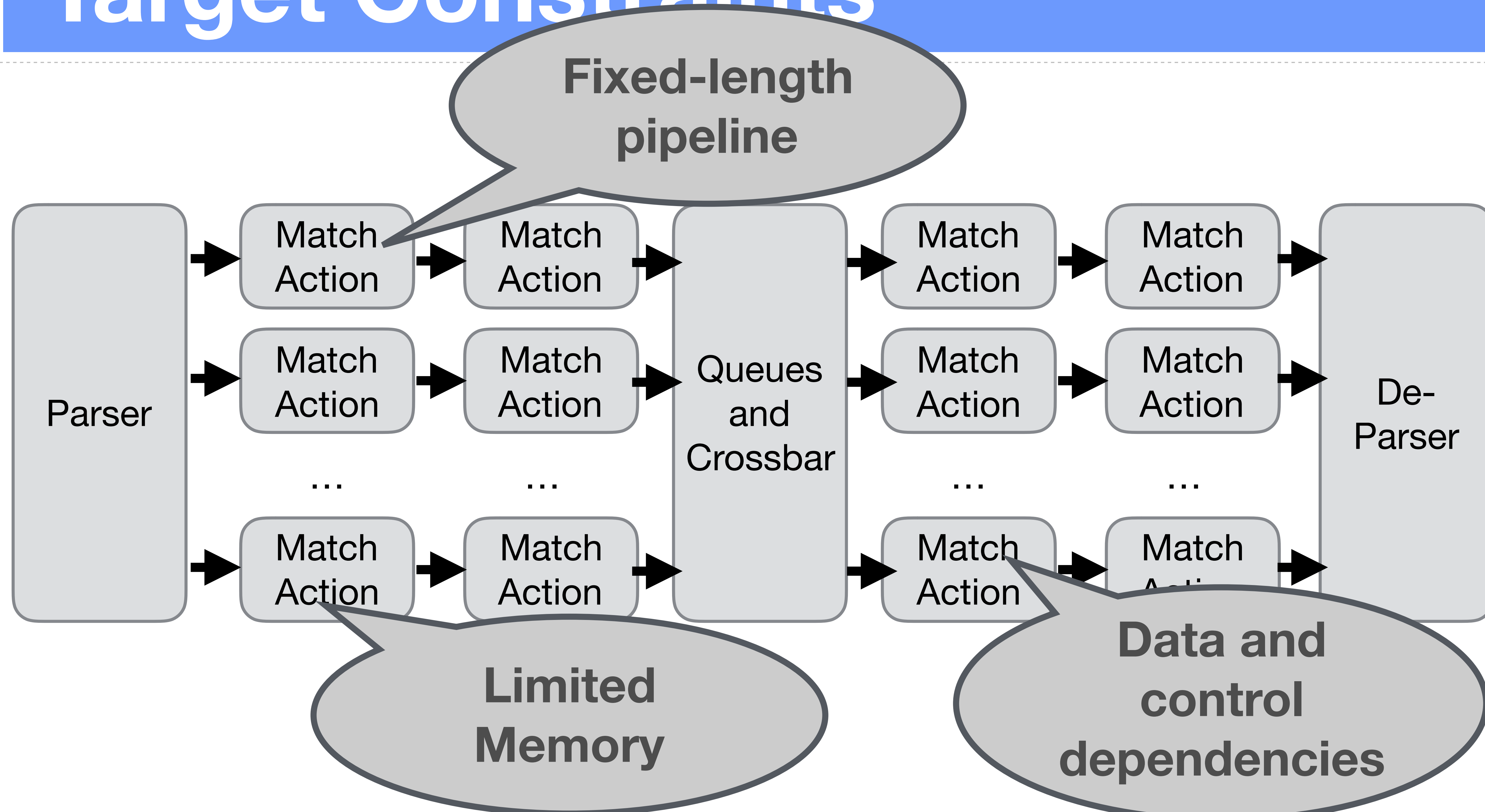
Target Constraints



Target Constraints



Target Constraints



Observations

- Architecture is designed for speed and efficiency
- Performance doesn't come for free
    - Limited degree of programmability
    - Not Turing complete by design
- Language syntax and hardware generations may change, but the basic design is fundamental

Co-Designing Networks and Distributed Systems

What Applications Should We Put in the Network?

Monte Carlo Simulation

Fundamental Building Blocks

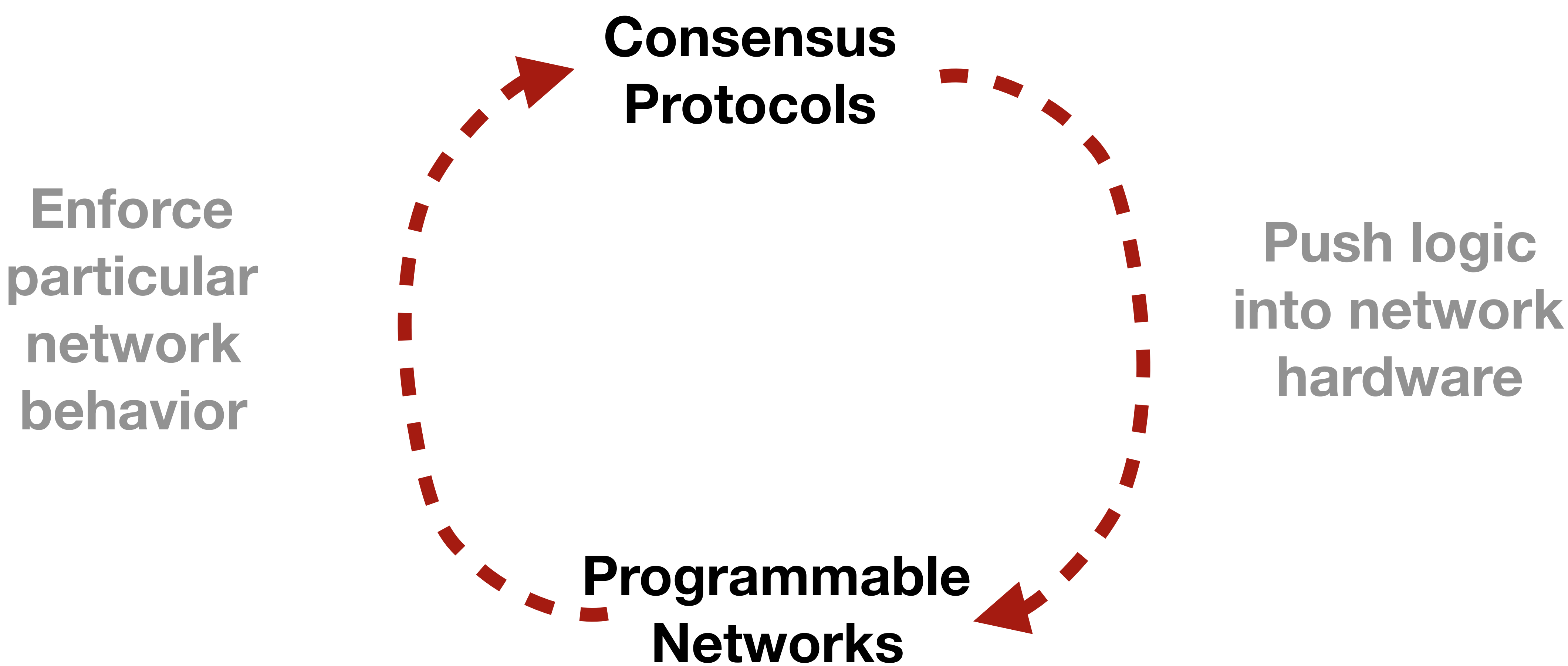
Building Blocks For Distributed Systems

Building Block	Description	System
Consensus	Essential for building fault-tolerant, replicated systems	NetPaxos SOSR ’15 P4xos, CCR ’16
Caching	Maximize utilization of available resources	NetCache, SOSP ’17 NetChain, NSDI ’18
Data Processing	In-network computation and analytics	Linear Road, SOSR ’18
Publish/Subscribe	Semantically meaningful communication	In submission

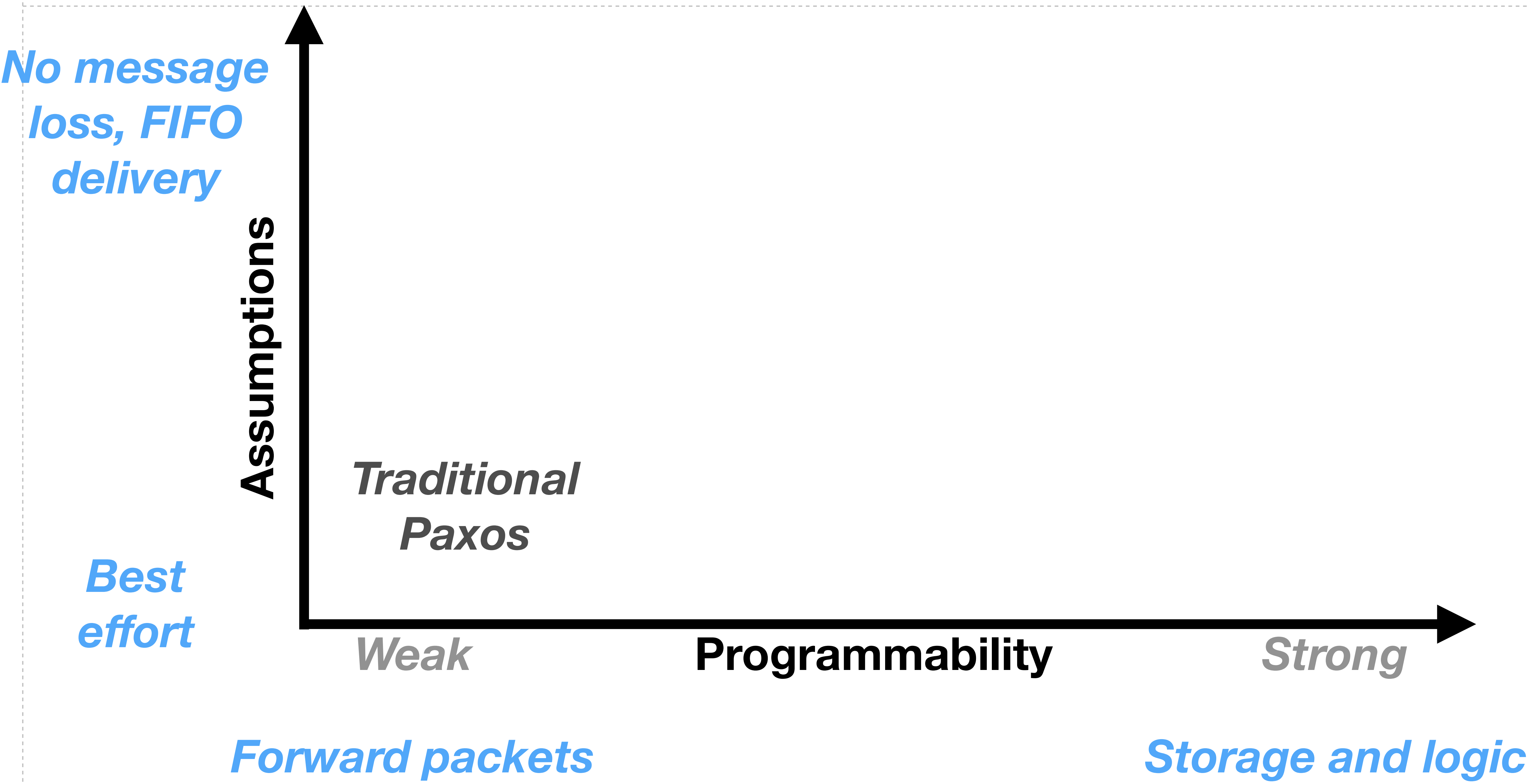
Consensus Protocols

- Get a group of replicas to agree on next application state
- Consensus protocols are the foundation for fault-tolerant systems
  - E.g., OpenReplica, Ceph, Chubby
- Many distributed systems problems can be reduced to consensus
  - E.g., Atomic broadcast, atomic commit

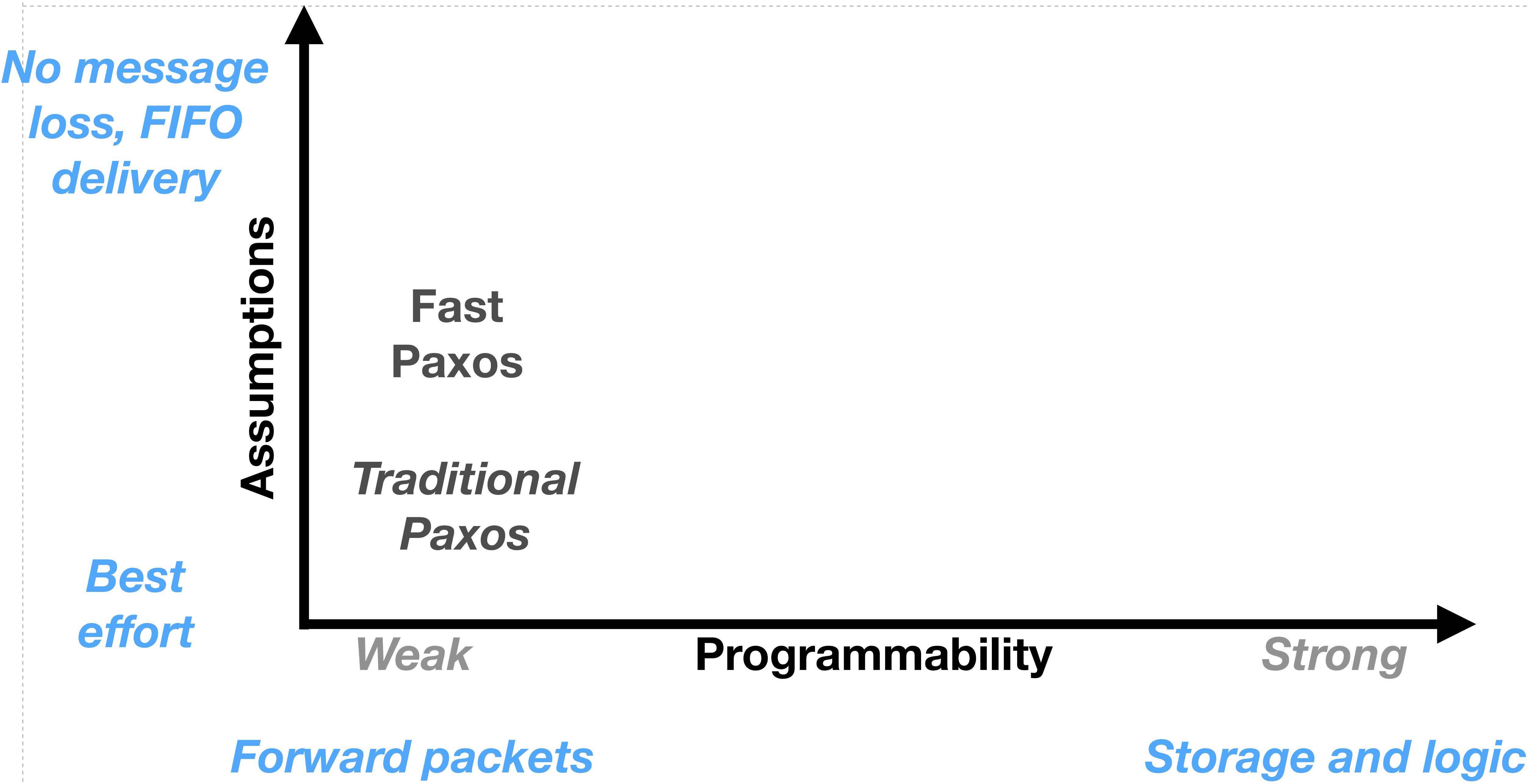
Ways to Improve Consensus Performance



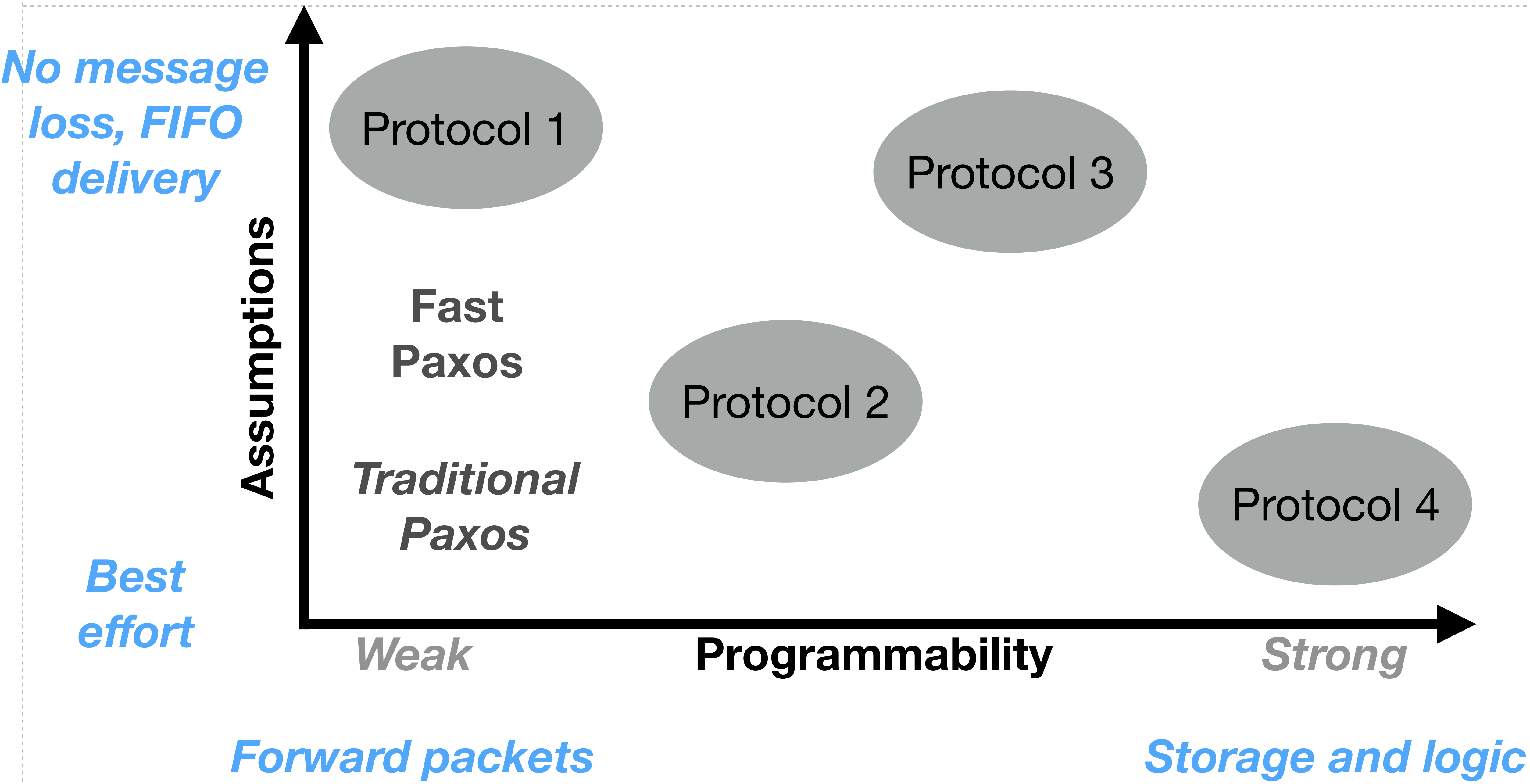
Consensus / Network Design Space



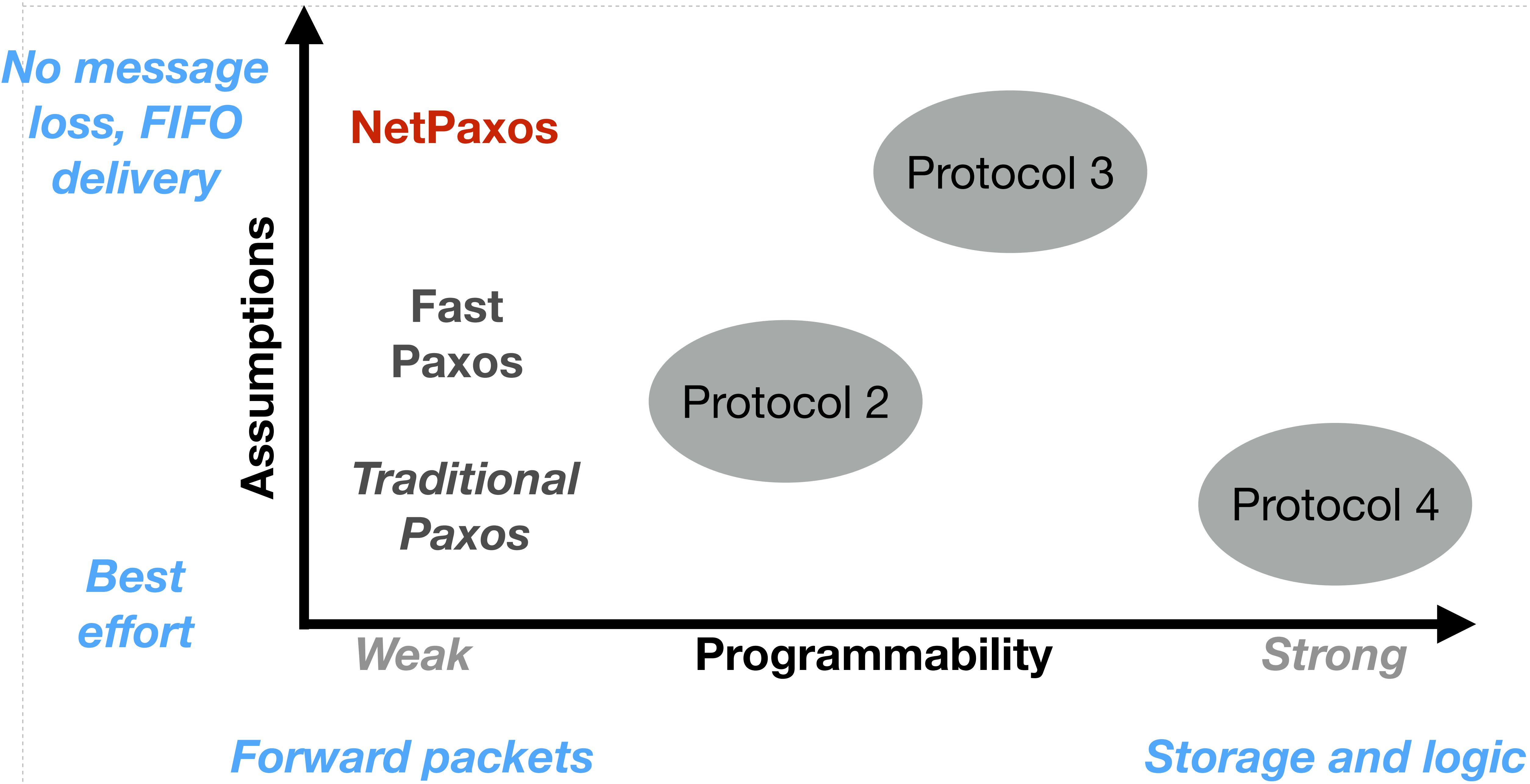
Consensus / Network Design Space



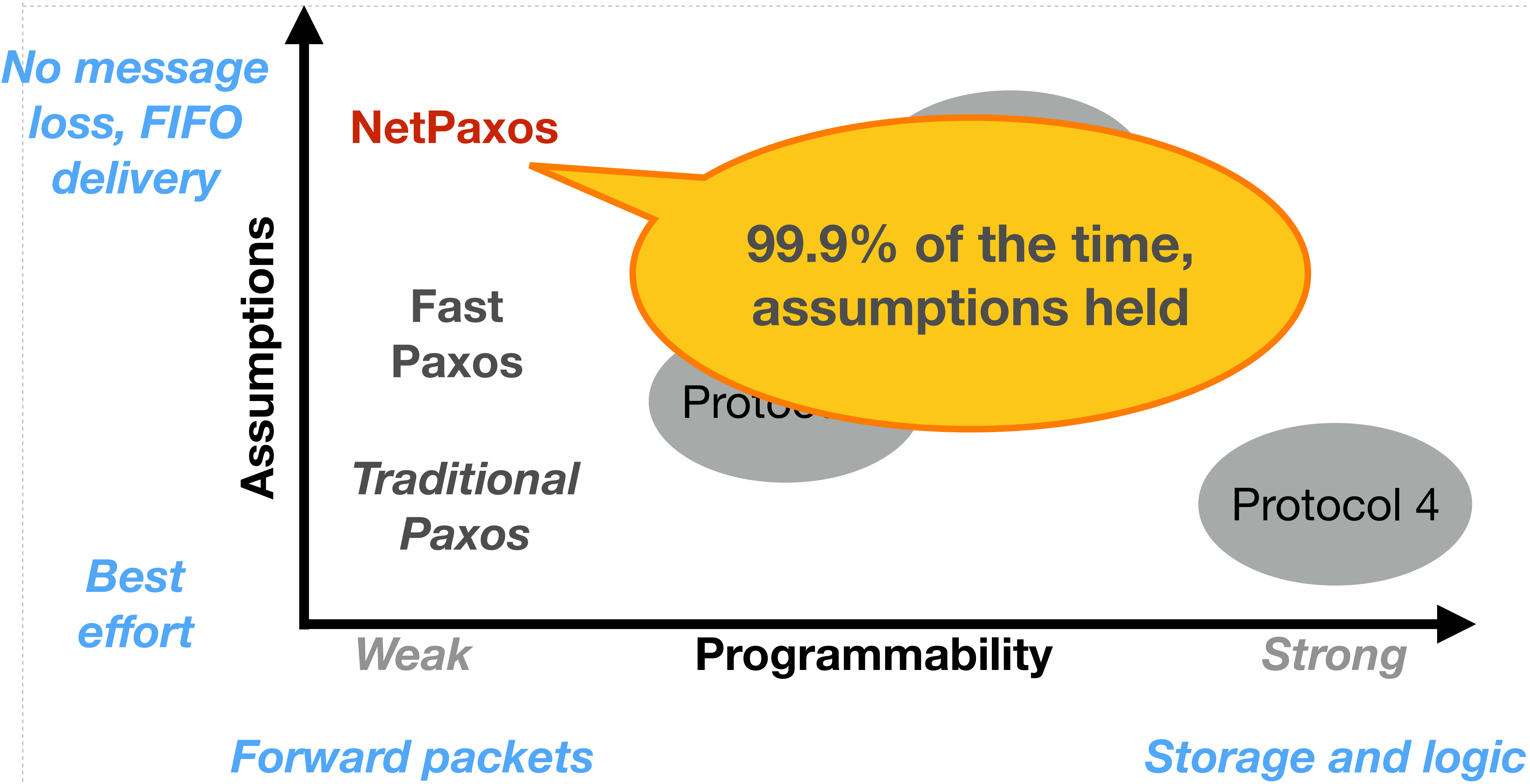
Consensus / Network Design Space



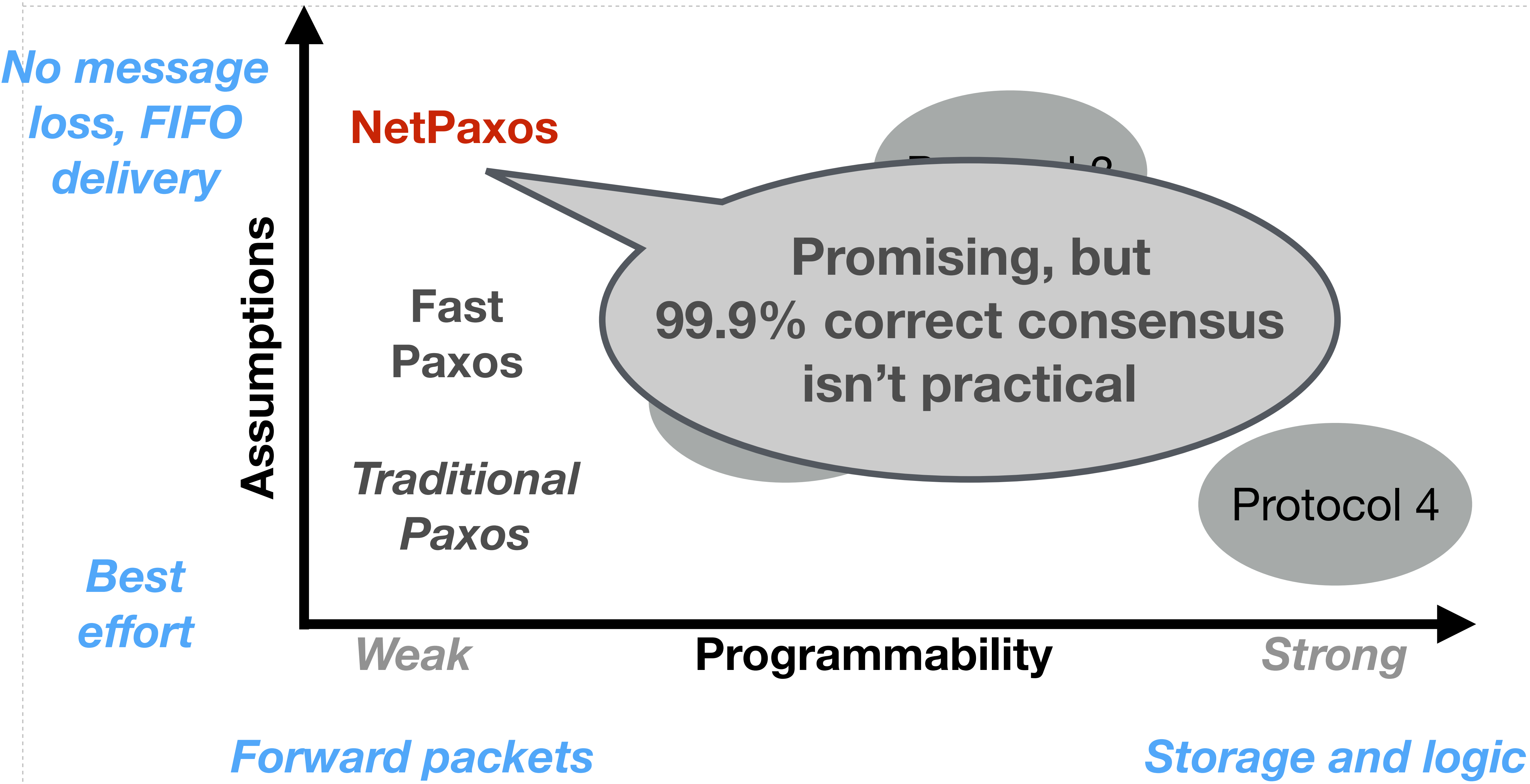
Consensus / Network Design Space



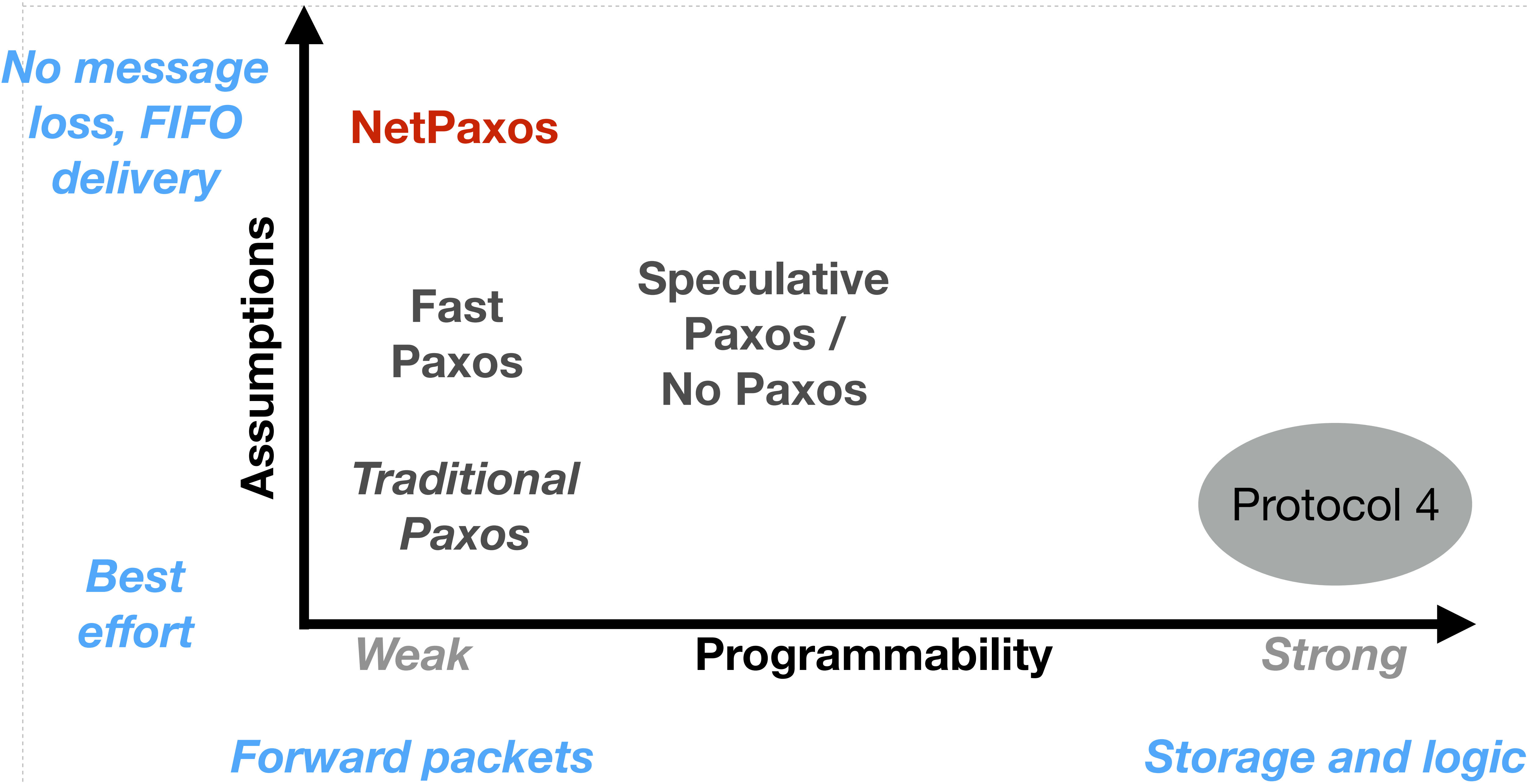
Consensus / Network Design Space



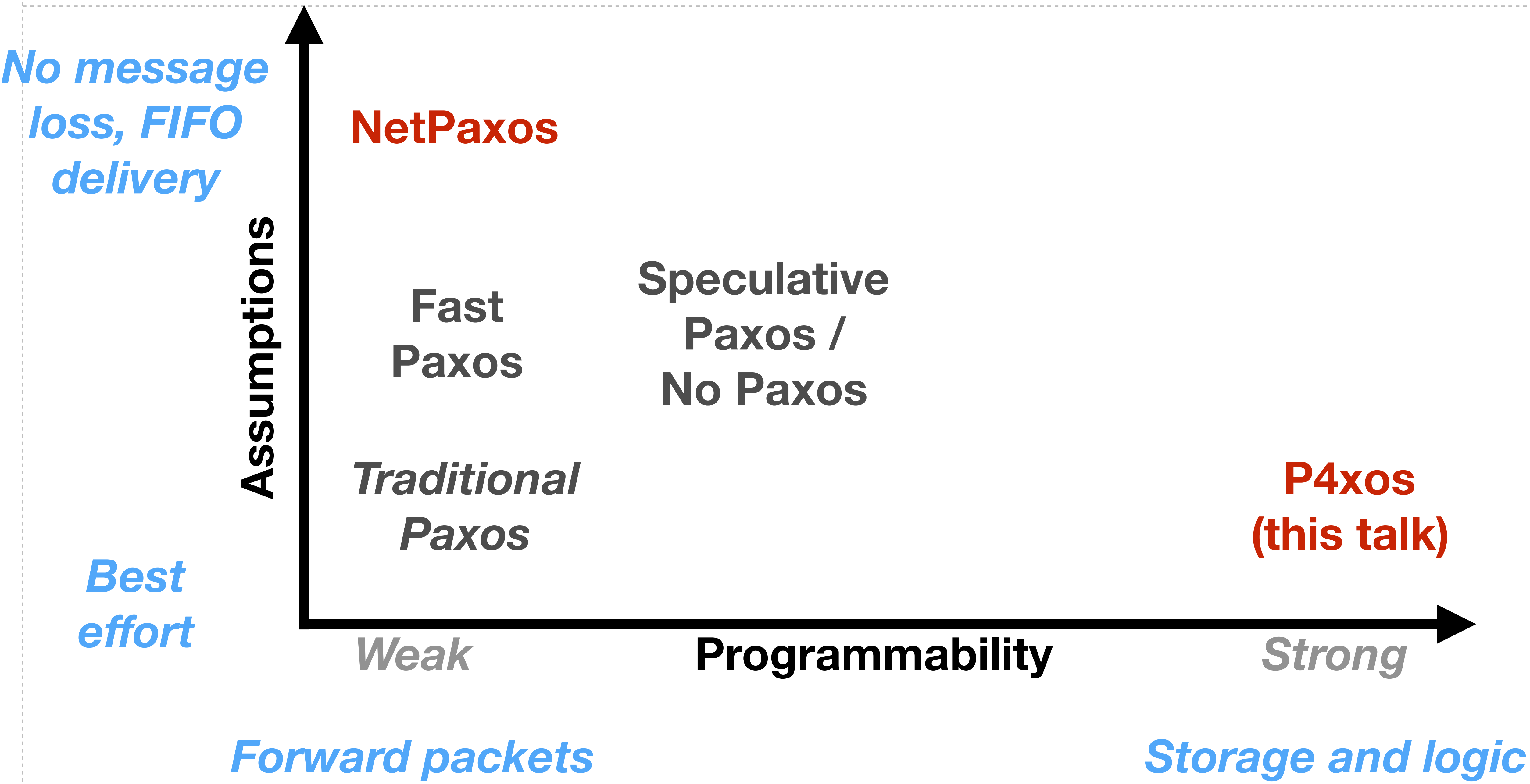
Consensus / Network Design Space



Consensus / Network Design Space



Consensus / Network Design Space



Paxos

Of the various consensus protocols, we focus on Paxos because:

- One of the most widely used
- Often considered the “gold standard”
- Proven correct

“There are two kinds of consensus protocols: those that are Paxos, and those that are incorrect”
— attributed to Butler Lampson

Paxos In the Network

Key questions:

- What parts of Paxos should be accelerated?
- How to map the algorithm to stateful forwarding decisions (i.e., Paxos logic as sequence of match/actions)?
- How do we map from complex protocol to low-level abstractions?
- What are the right interfaces? How do we deploy?

Paxos in a Nutshell

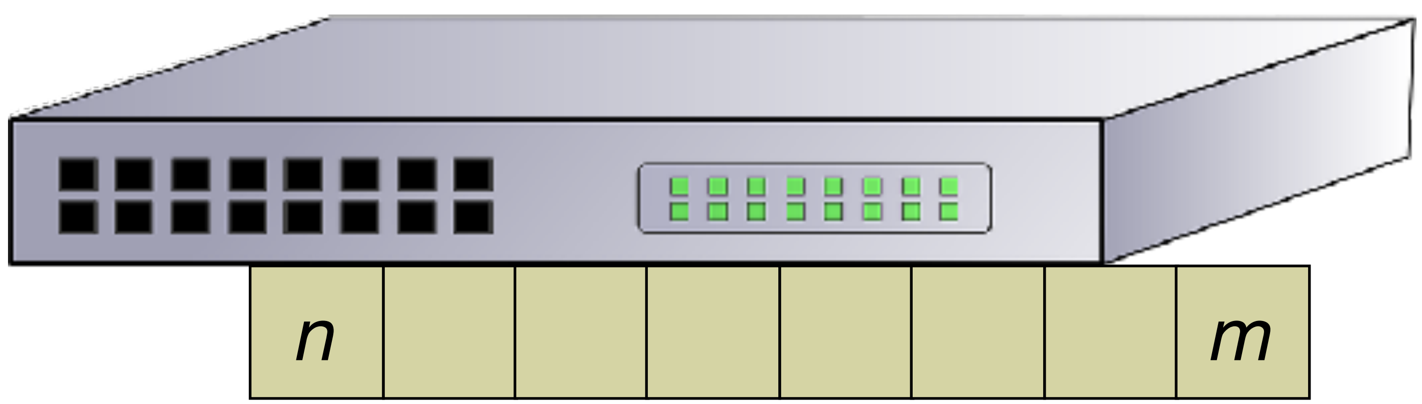
- An execution of Paxos is called an **instance**. Each instance is associated with an ID, called the **instance number**.
- The protocol has two phases. Each phase may contain multiple **rounds**. There is a **round number** to identify the round.
  - Phase 1: “What instance number are we talking about?”
  - Phase 2: “What is the value for the instance number?”
- Observation: Phase 1 does not depend on a particular value. We should accelerate Phase 2.

Paxos In The Switch

Paxos Packets

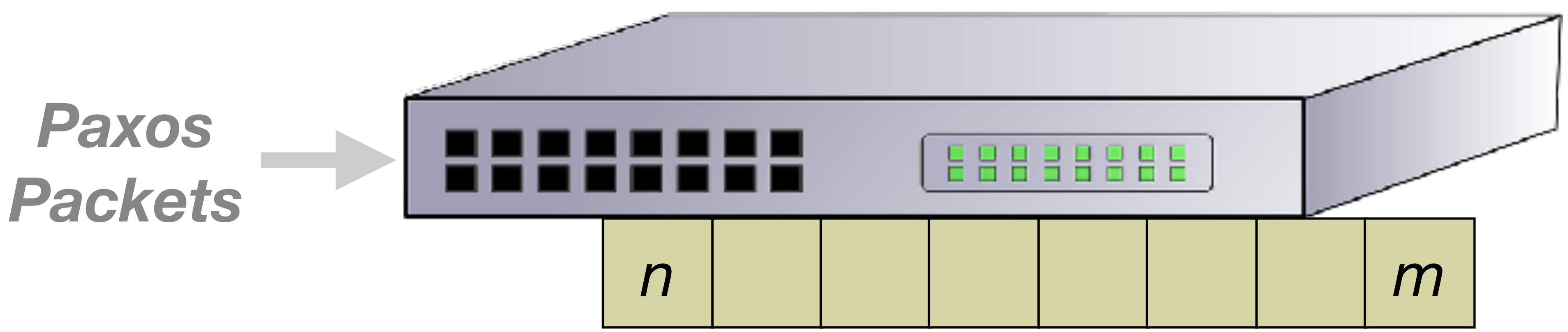
- type
- instance
- round
- vround
- value

Union of all Paxos messages

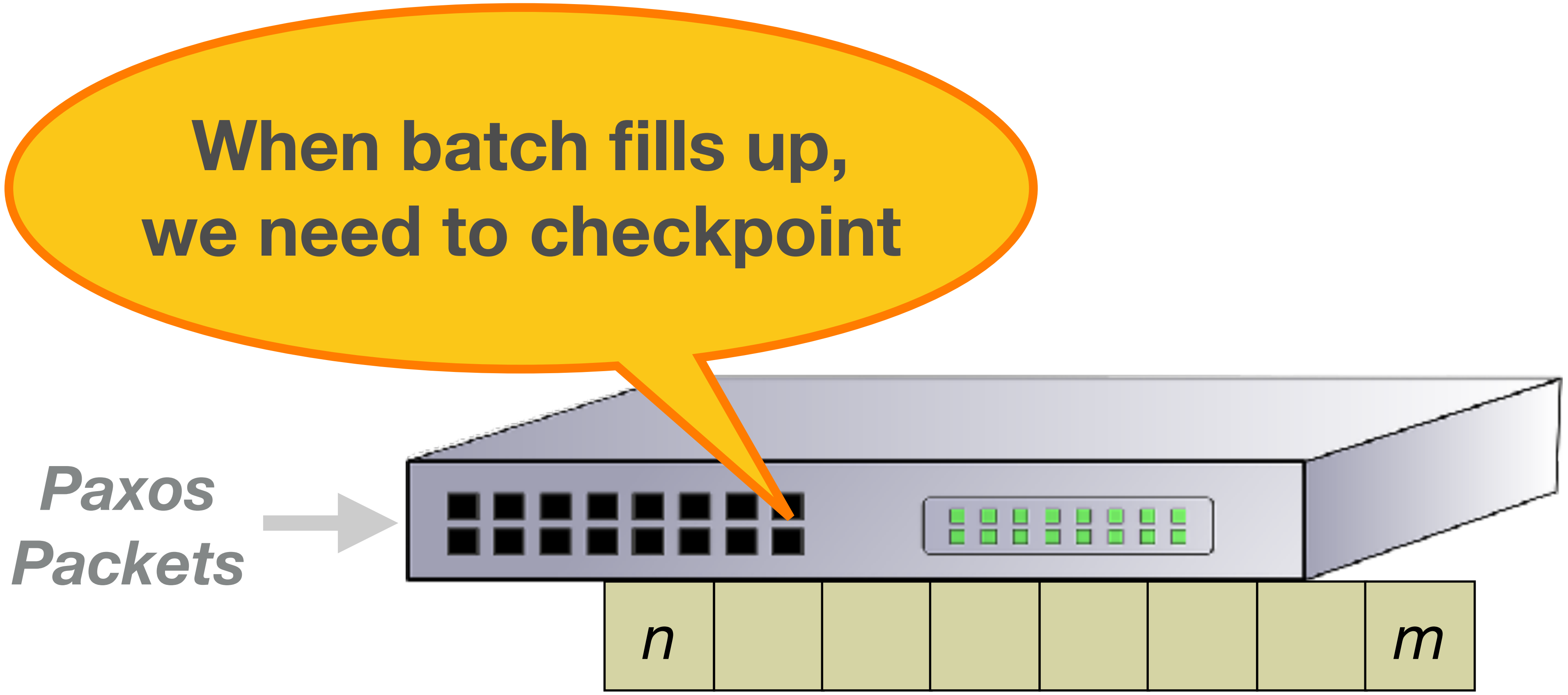


Run Phase 1 in a batch, declare the instance numbers to use

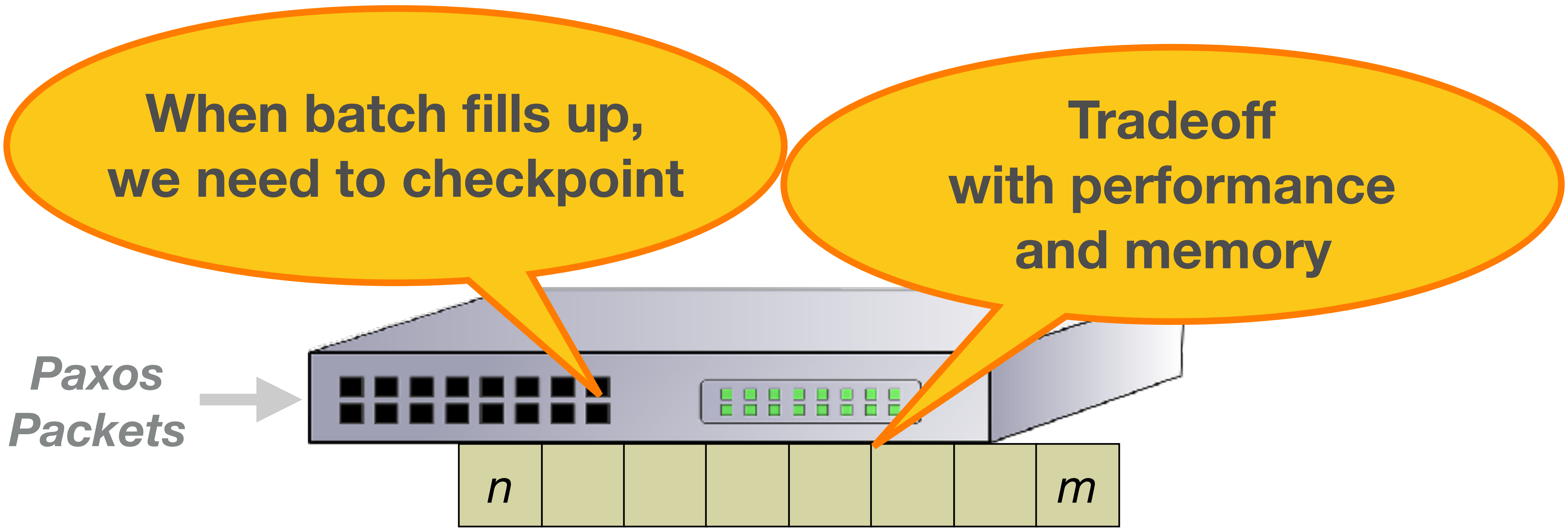
Paxos In The Switch



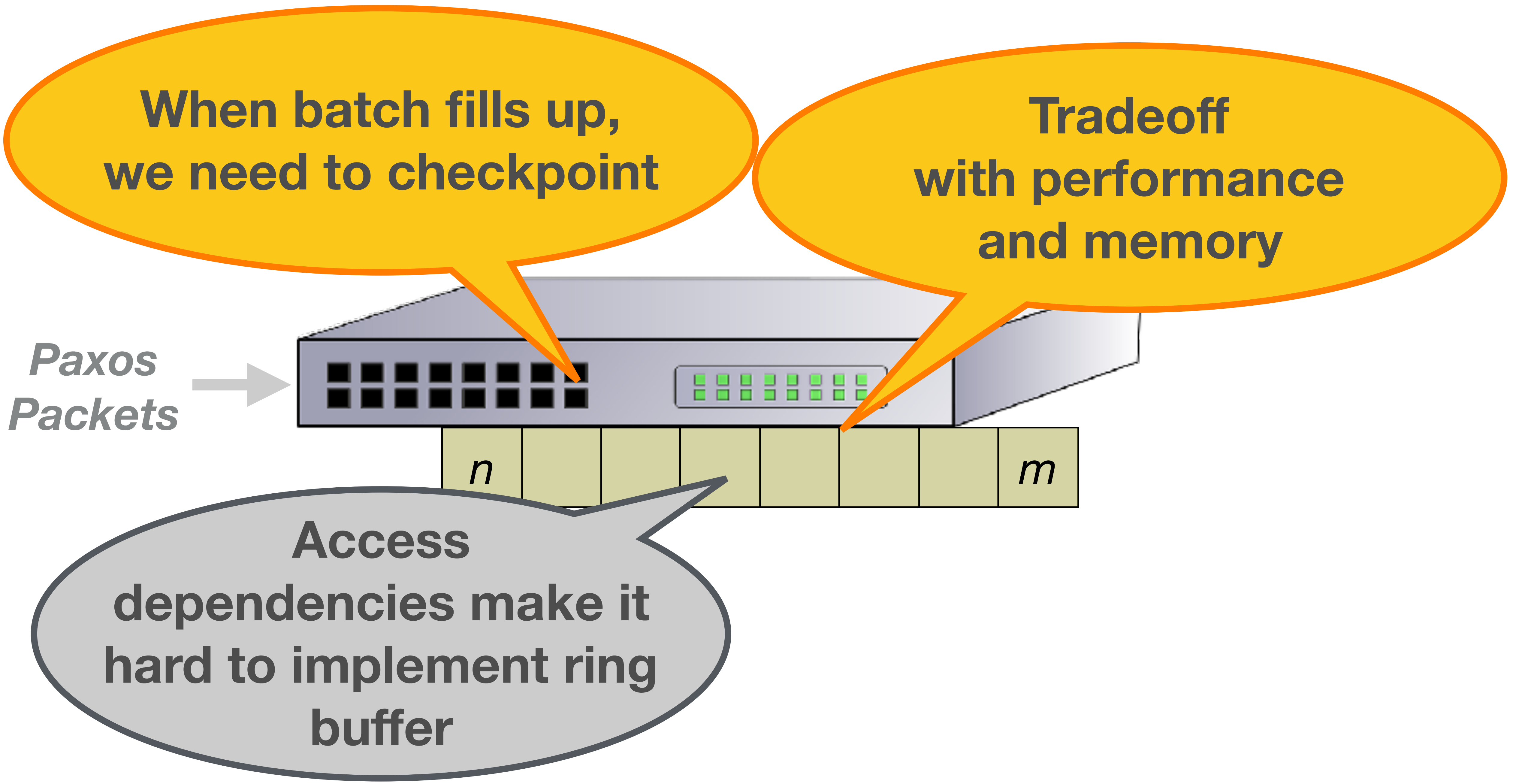
Paxos In The Switch



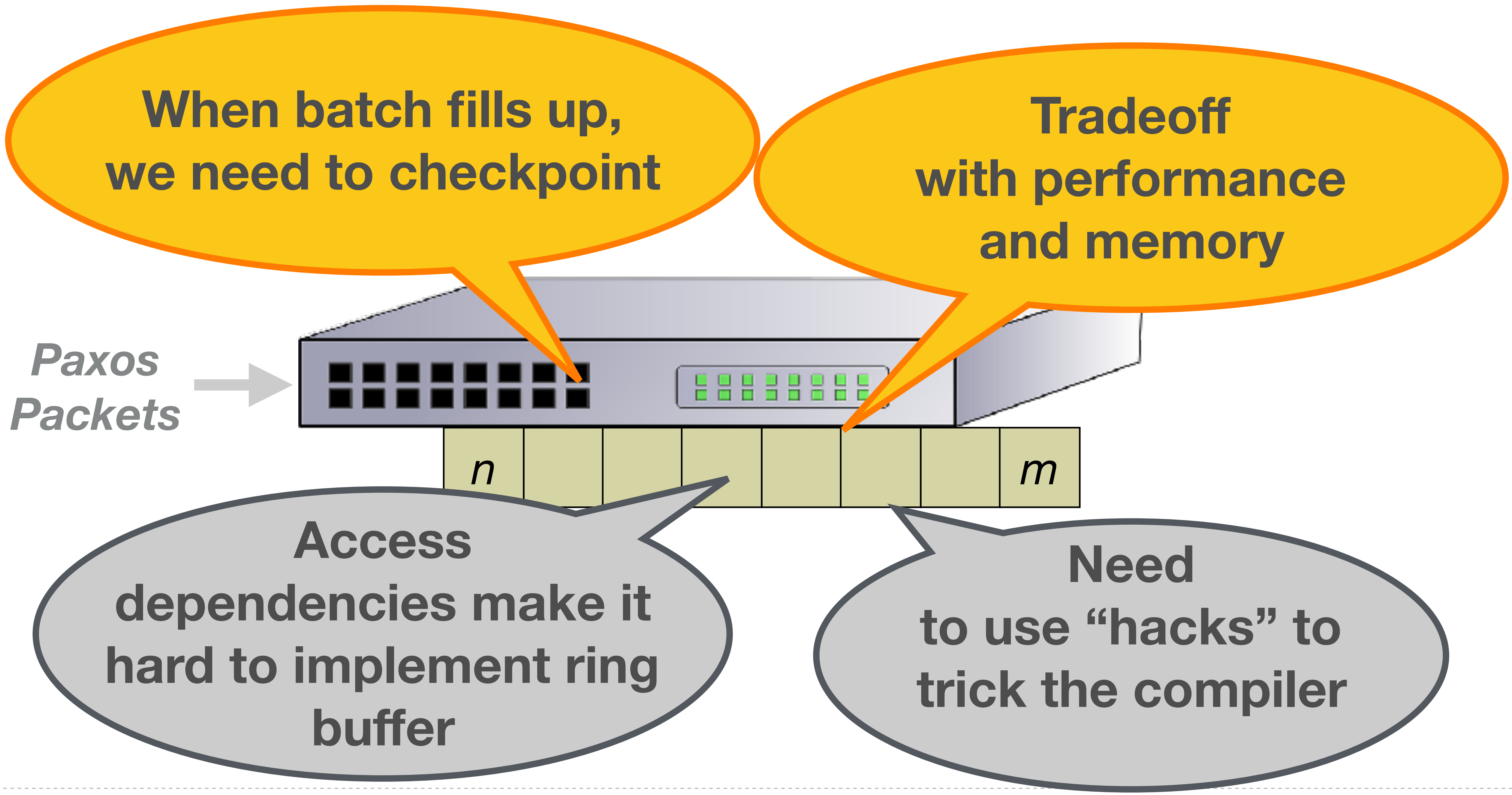
Paxos In The Switch



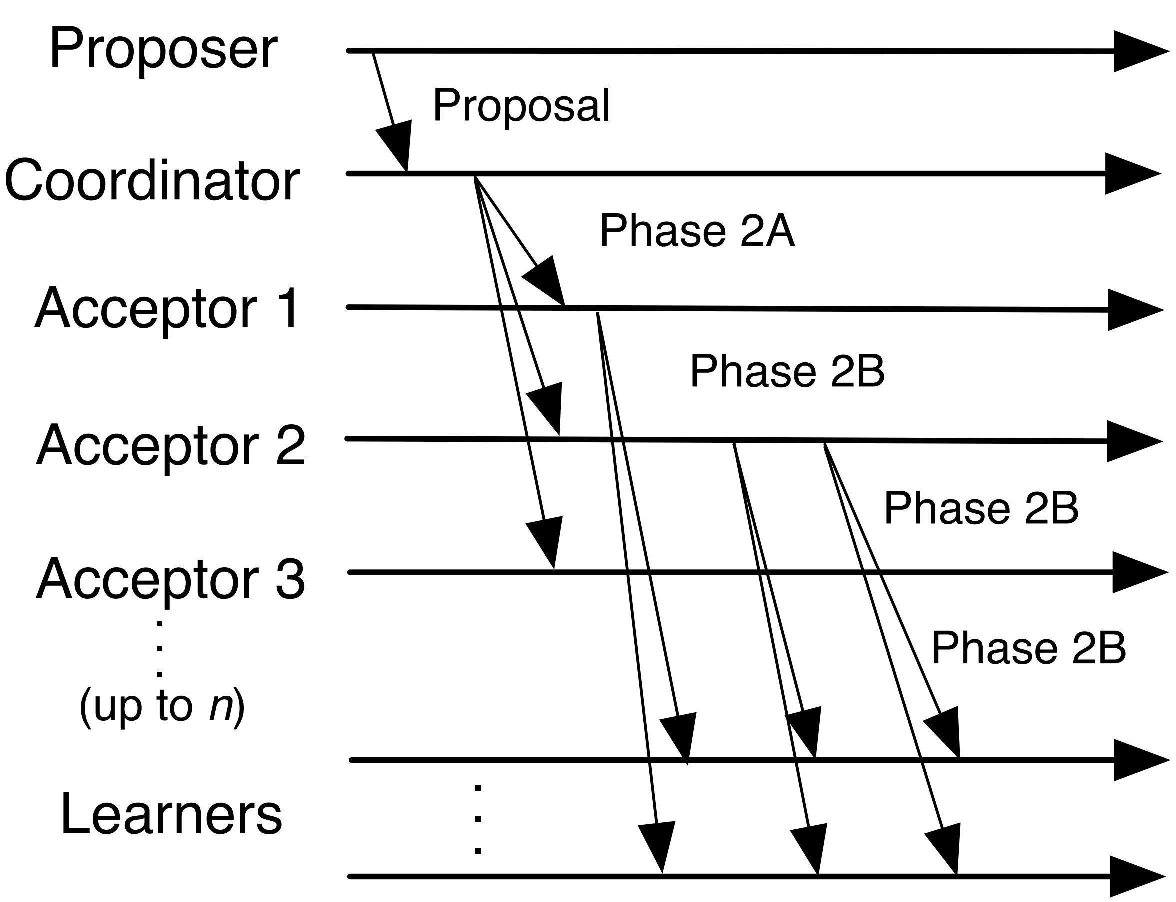
Paxos In The Switch



Paxos In The Switch

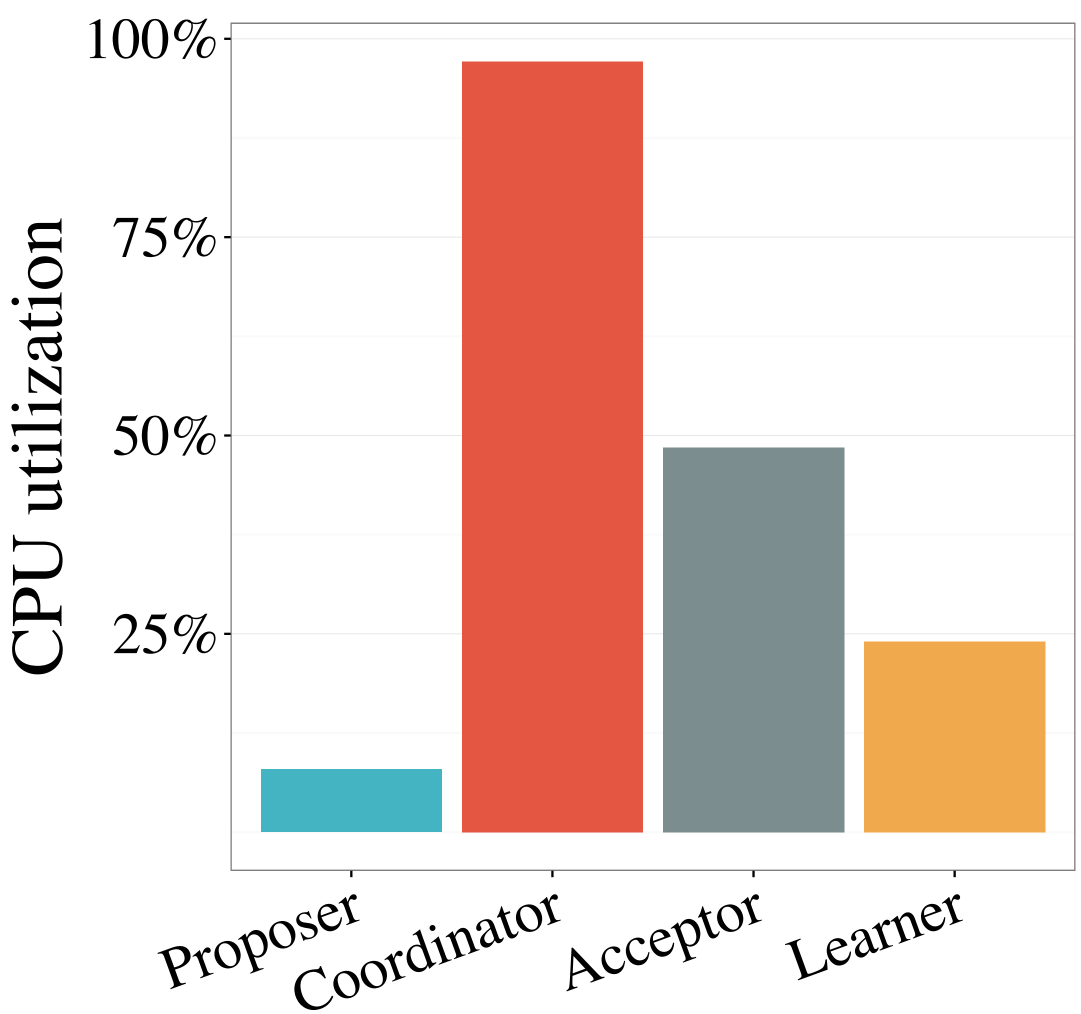


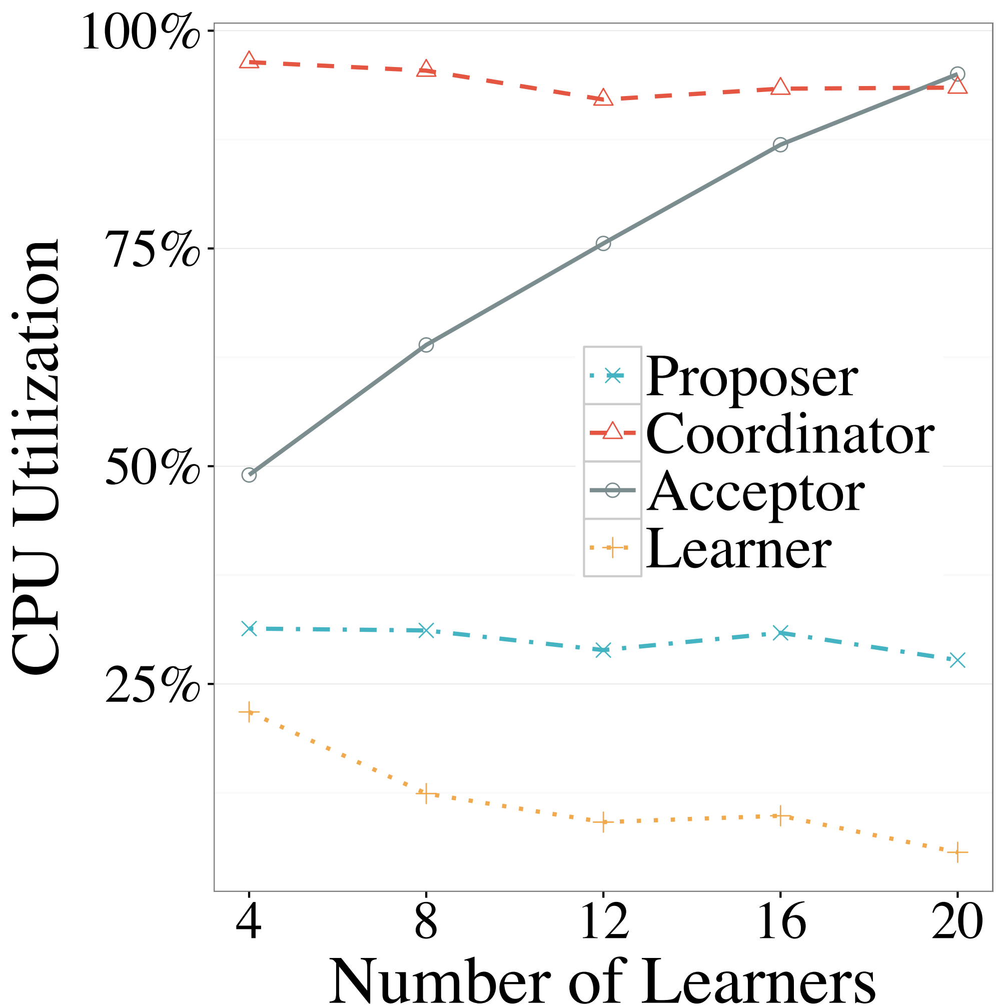
Phase 2 Roles and Communication



- Proposers propose a value via the Coordinator (Phase 2).
- Acceptors accept value, promise not to accept any more proposals for instance (Phase 2).
- Learners require a quorum of messages from Acceptors, "deliver" a value (Phase 2).

Paxos Bottlenecks





Observation: accelerate *agreement*:
Coordinator and Acceptors

Paxos as Prose

1. (a) If $crnd[c] < i$, then c starts round i by setting $crnd[c]$ to i, setting $cval[c]$ to $none$, and sending a message to each acceptor a requesting that a participate in round i.

   (b) If an acceptor a receives a request to participate in round i and $i > rnd[a]$, then a sets $rnd[a]$ to i and sends coordinator c a message containing the round number i and the current values of $vrnd[a]$ and $vval[a]$.

   If $i \leq rnd[a]$ (so a has begun round i or a higher-numbered round), then a ignores the request.

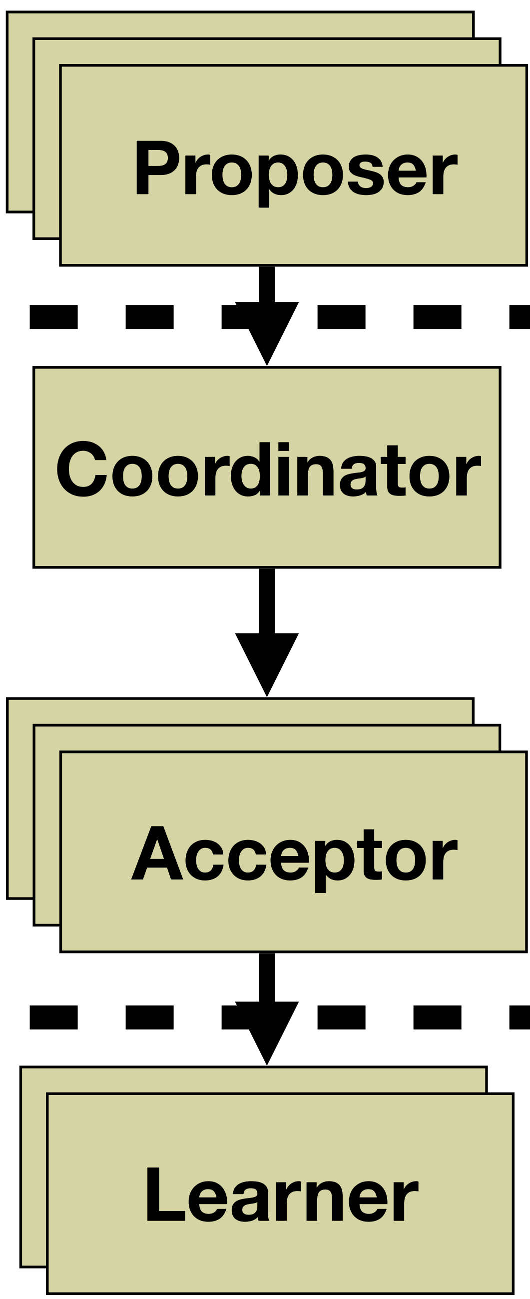
[Lamport, Distributed Computing '06]

Paxos as Match-Action

```
Initialize State:
    instance[1] := {0}
upon receiving pkt(msgtype, inst, rnd, vrnd, swid, value)
    match pkt.msgtype:
        case REQUEST:
            pkt.msgtype ← PHASE2A
            pkt.rnd ← 0
            pkt.inst ← instance[0]
            instance[0] := instance[0] + 1
            multicast pkt
        default :
            drop pkt
```

Coordinator Algorithm

Paxos as Match-Action



Encode value in a packet header.

Application

Network

If match, add sequence number, and forward

If match, compare round field in header, update state, and forward

Network

Application

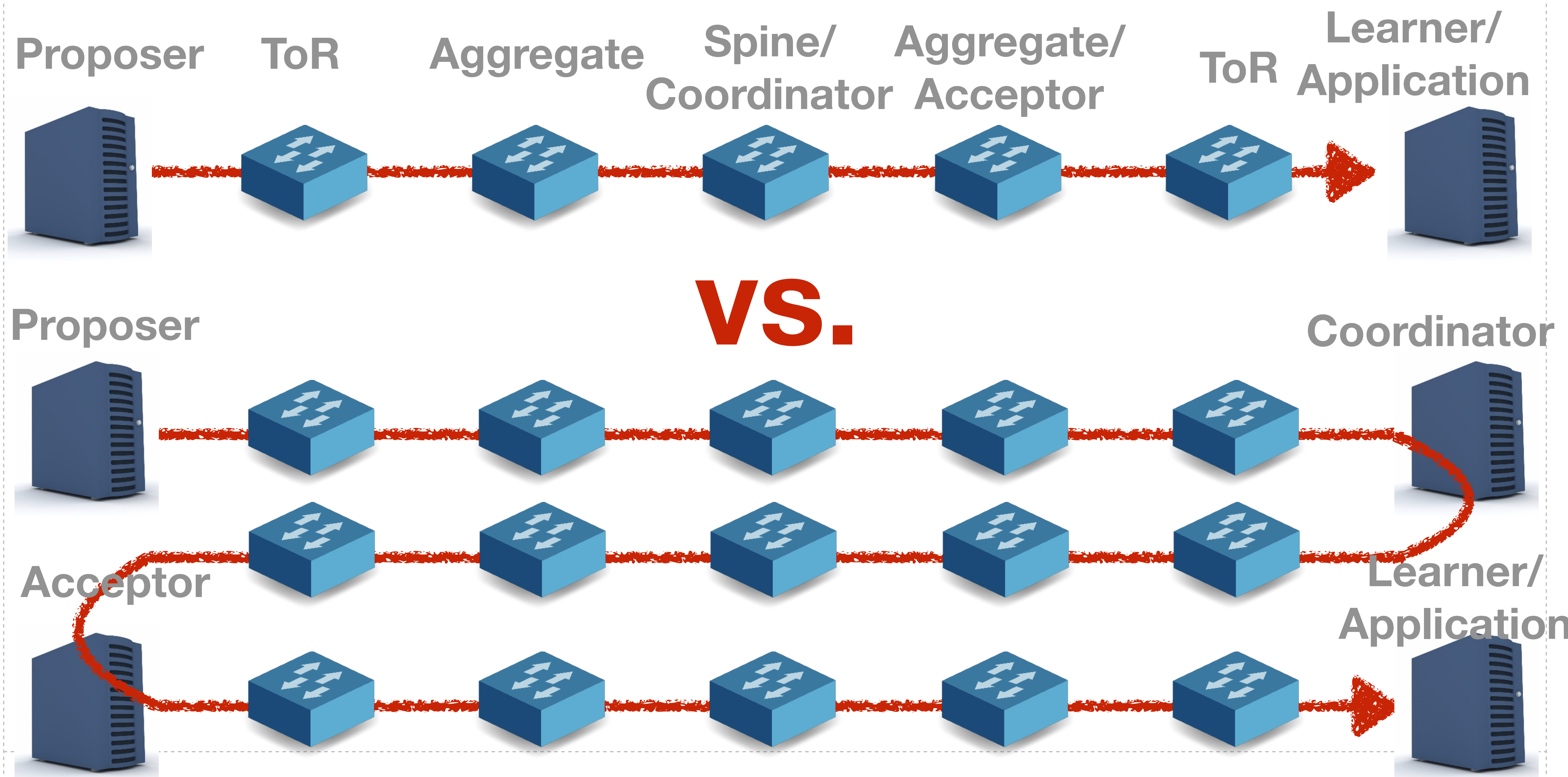
De-encode and return value to the application.

Application Interface

API Function Names	Description
submit	Application to network: Send a value
deliver	Network to application: Deliver a value
recover	Application to network: Discover a prior value

C wrapper provides a drop-in replacement for existing Paxos libraries!

P4xos Deployment



Experiments

Focus on two questions:

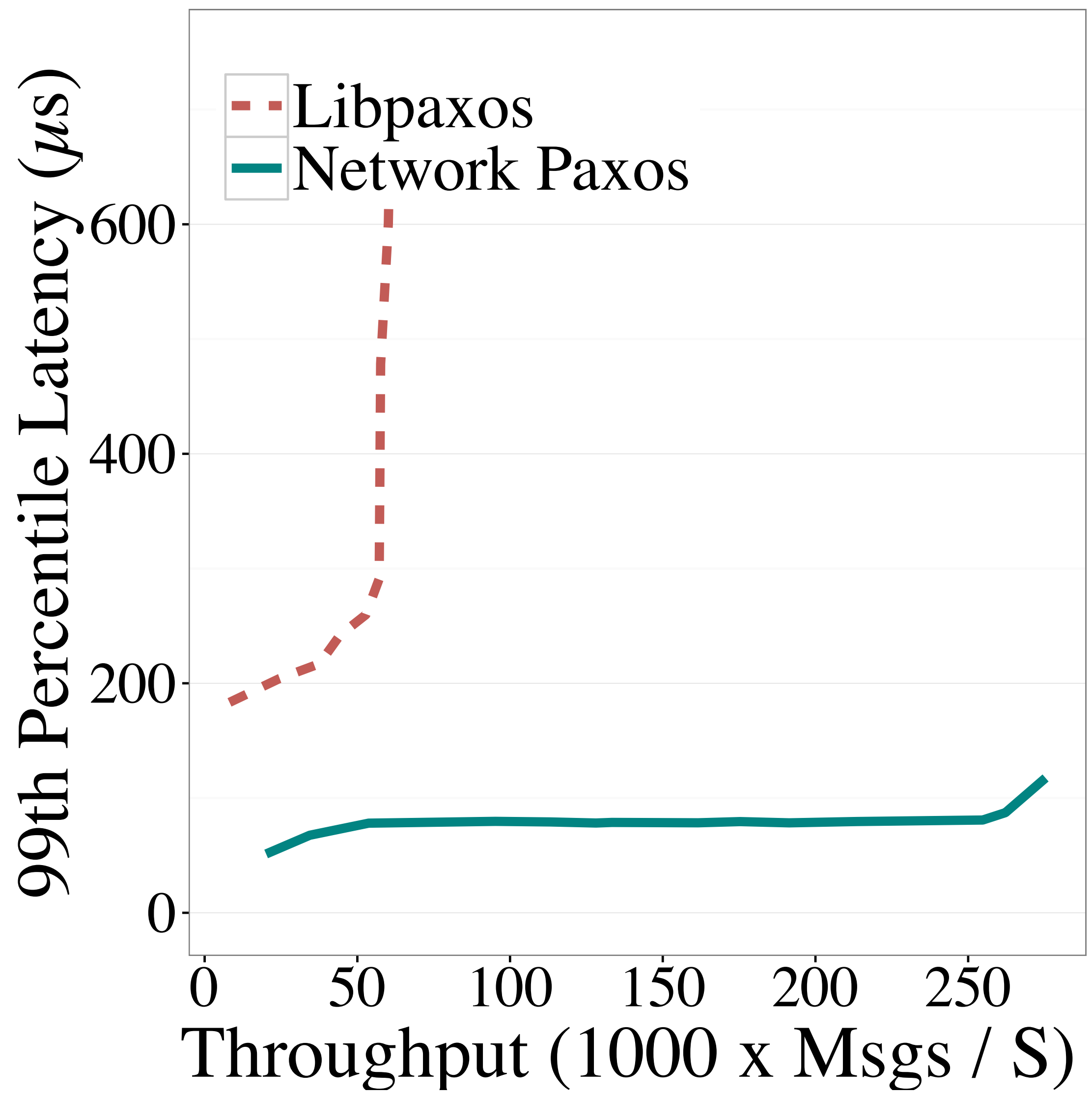
- What is the absolute performance?
- What is the end-to-end performance?

Absolute Performance

- Measured each role separately on 64x40G ToR switch (Barefoot Tofino) and IXIA XGS12-H as packet sender
- **Throughput is over 2.5 billion consensus messages / second. This is a 10,000x improvement over software.**
- Data plane latency is less than 0.1 µs (measured inside the chip)

End-to-End Performance

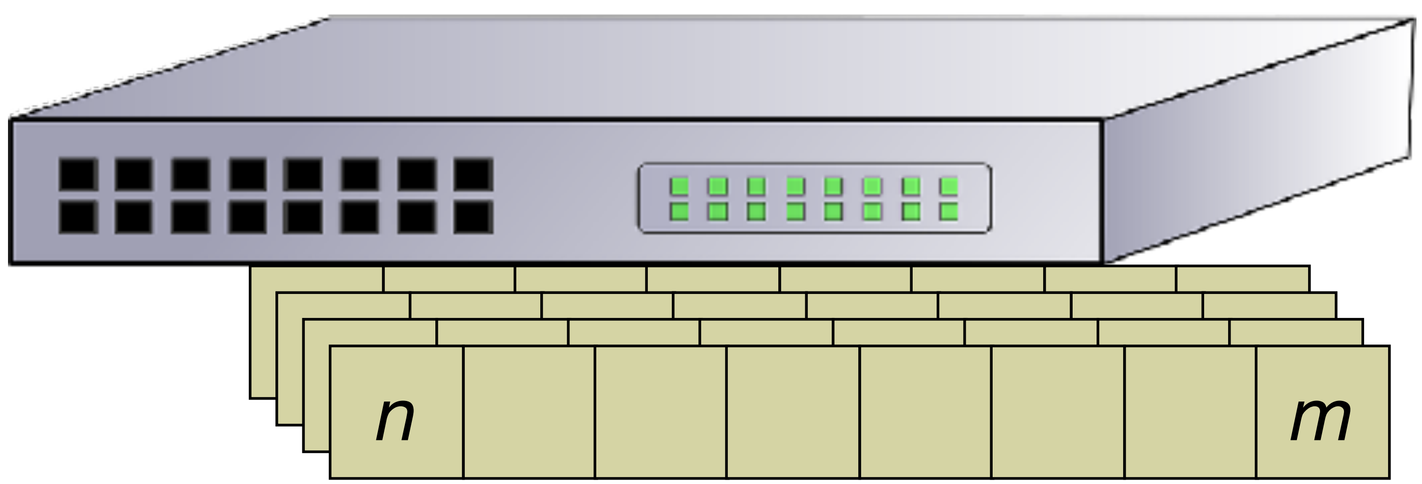
- Application delivers to RocksDB with read and write commands
- 4.3x throughput improvement over software implementation
- 73% reduction in latency



Accelerating Execution (Work-in-Progress)

Paxos Packets

- type
- instance
- round
- vround
- value
- **partition**

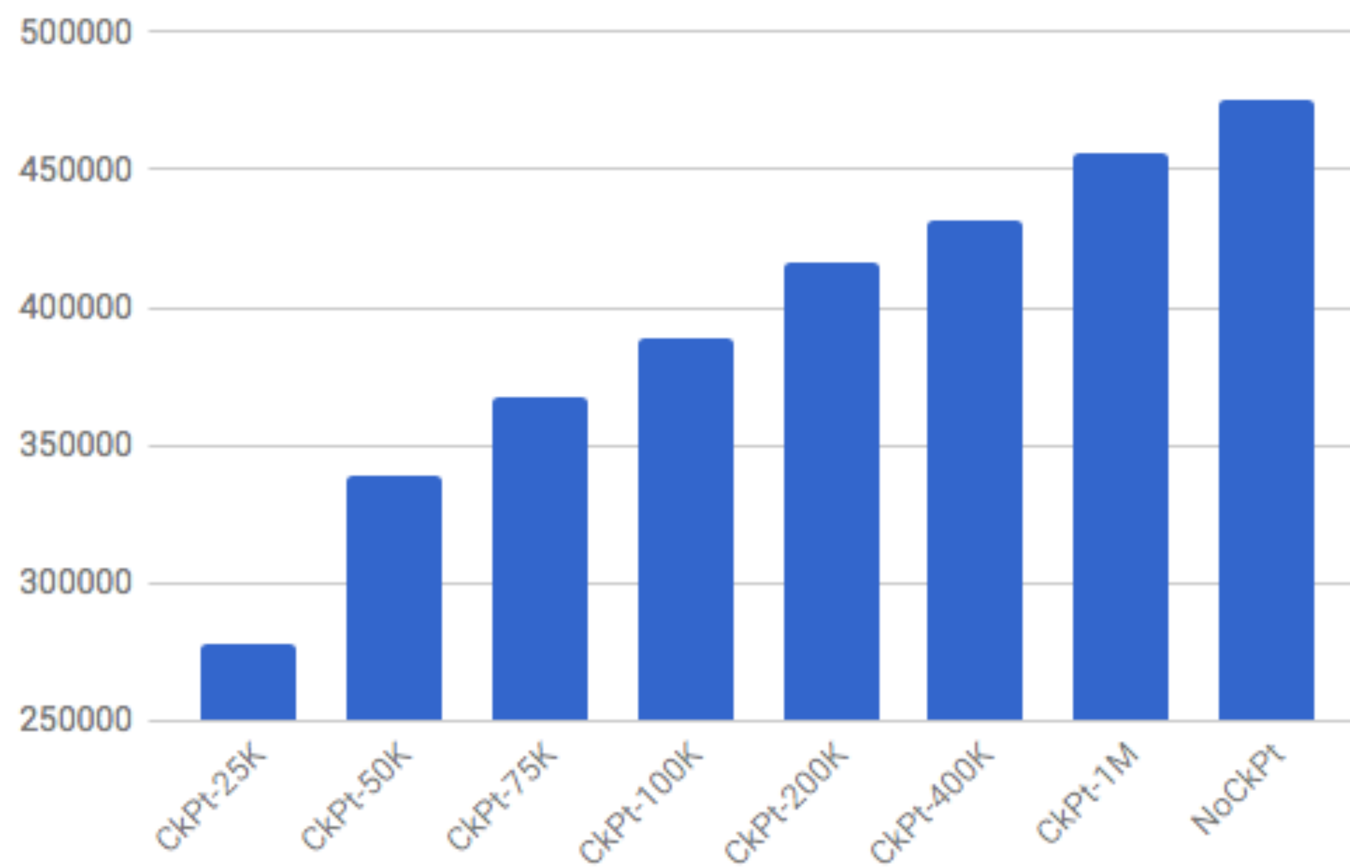


Run multiple Paxi in parallel

Partition application state

Accelerating Execution (Work-in-Progress)

- Not yet done: handling "cross partition" requests
- Must add barriers to synchronize learners
- Fully partitioned workload reaches 500K msgs/sec

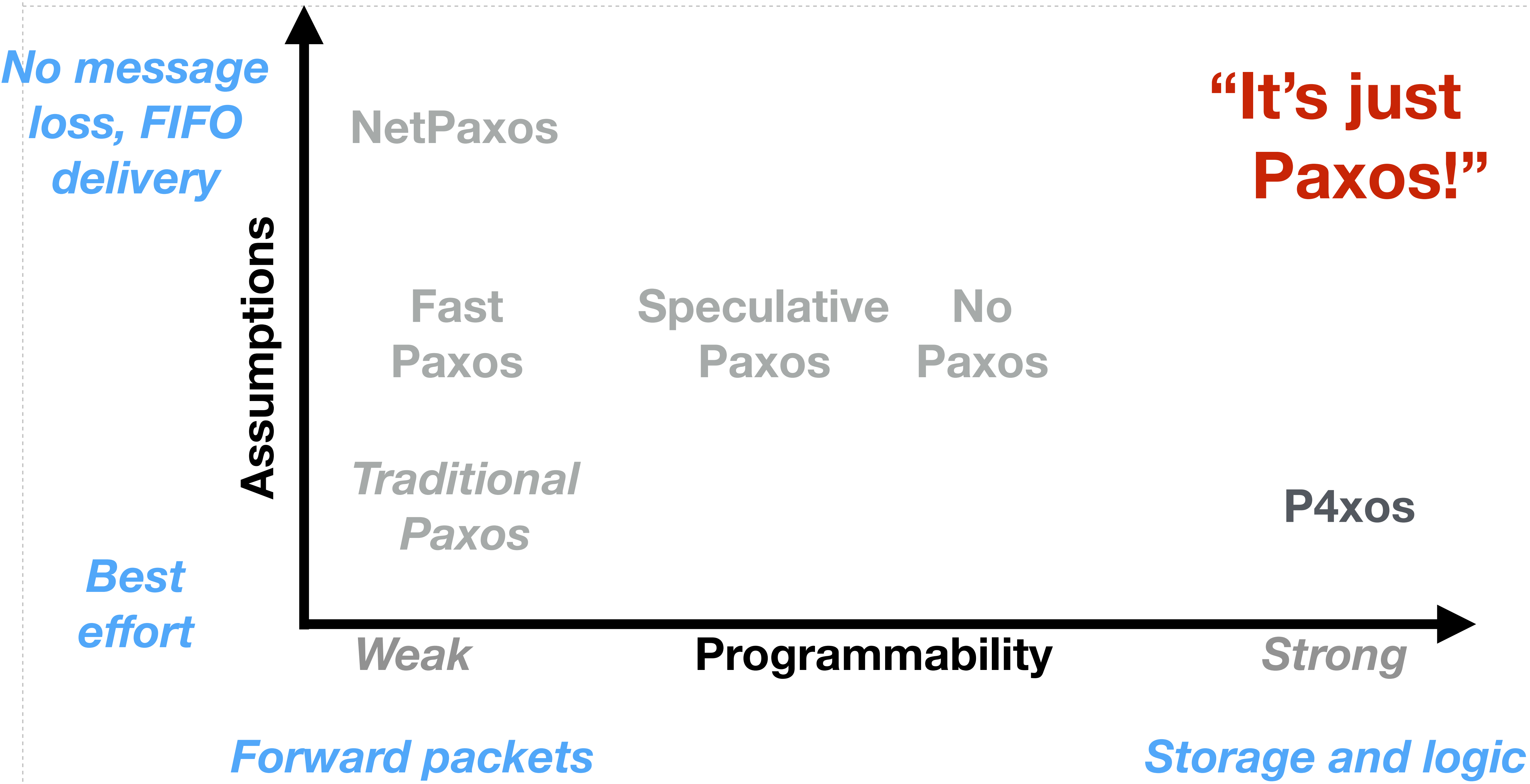


RocksDB Throughput vs. Checkpoint Interval

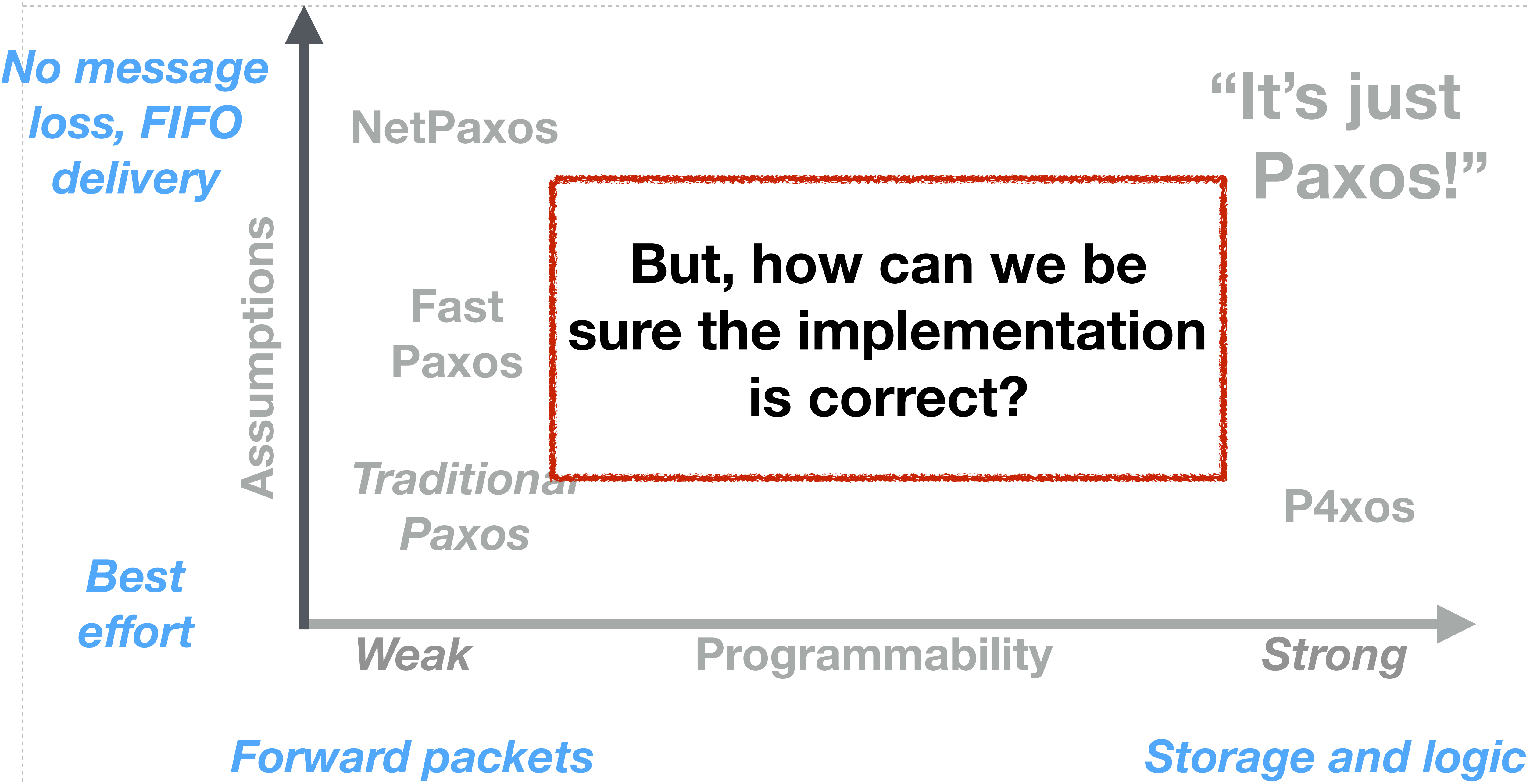
Practical Application: Storage Class Memory

- Fast network interconnect allows users to scale storage and compute separately (i.e., disaggregated storage)
- Several companies, including Western Digital, have developed new types of non-volatile memory
  - Persistent, with latency comparable to DRAM
  - But, wears out over time…
- Use in-network consensus to keep replicas consistent

To Recap


No message loss, FIFO delivery
Best effort
Assumptions
NetPaxos
"It's just Paxos!"
Fast Paxos
Speculative Paxos
No Paxos
Traditional Paxos
P4xos
Weak
Programmability
Strong
Forward packets
Storage and logic

To Recap


No message loss, FIFO delivery
Assumptions
Best effort
NetPaxos
Fast Paxos
Traditional Paxos
"It's just Paxos!"
P4xos
But, how can we be sure the implementation is correct?
Weak
Programmability
Strong
Forward packets
Storage and logic

Proving Correctness (or How Do We Know Our Implementation is Correct?)

An Old Story You've Heard Before

- We checked the Paxos algorithm with SPIN model checker. No problems!
- We wrote the Paxos code.
- We ran in the network, but didn't get consensus.

There is a bug in our implementation.

Verification is So Tempting...

- To the extent networks are verified, the focus is on forwarding (e.g., no path loops)
- If the network is going to take on more work, how can we be sure that is correct?
- P4 is so tempting to verify: no loops, no pointers, etc.

Verification Problem

The specific behavior of a P4 program depends on the control plane

Rules

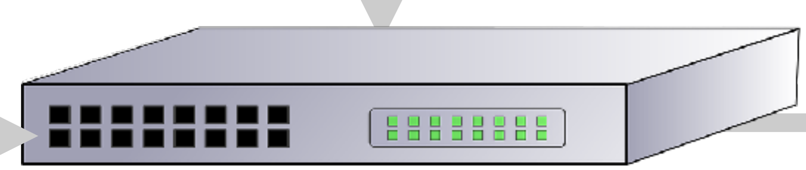
“I[illegible]1, fo[illegible]”

Control Plane

Data Plane

We only have half the program!

P4



Hoare Logic

Axioms capture relational properties: what is true before and after a command executes.

$$\vdash \{P\}\ c\ \{Q\}$$

If P holds and c executes, then Q holds.

- **Standard approach to verification**
- **Use automated theorem-prover to check if there is an initial state that leads to a violation**
- **Generate a counter example via weakest pre-condition**

P4 + Hoare Logic

Axioms capture relational properties: what is true before and after a command executes.

$$\vdash \{\, P + \text{“control plane assumptions”} \,\}\ c\ \{\, Q \,\}$$

If P plus some assumed knowledge holds and c executes, then Q holds.

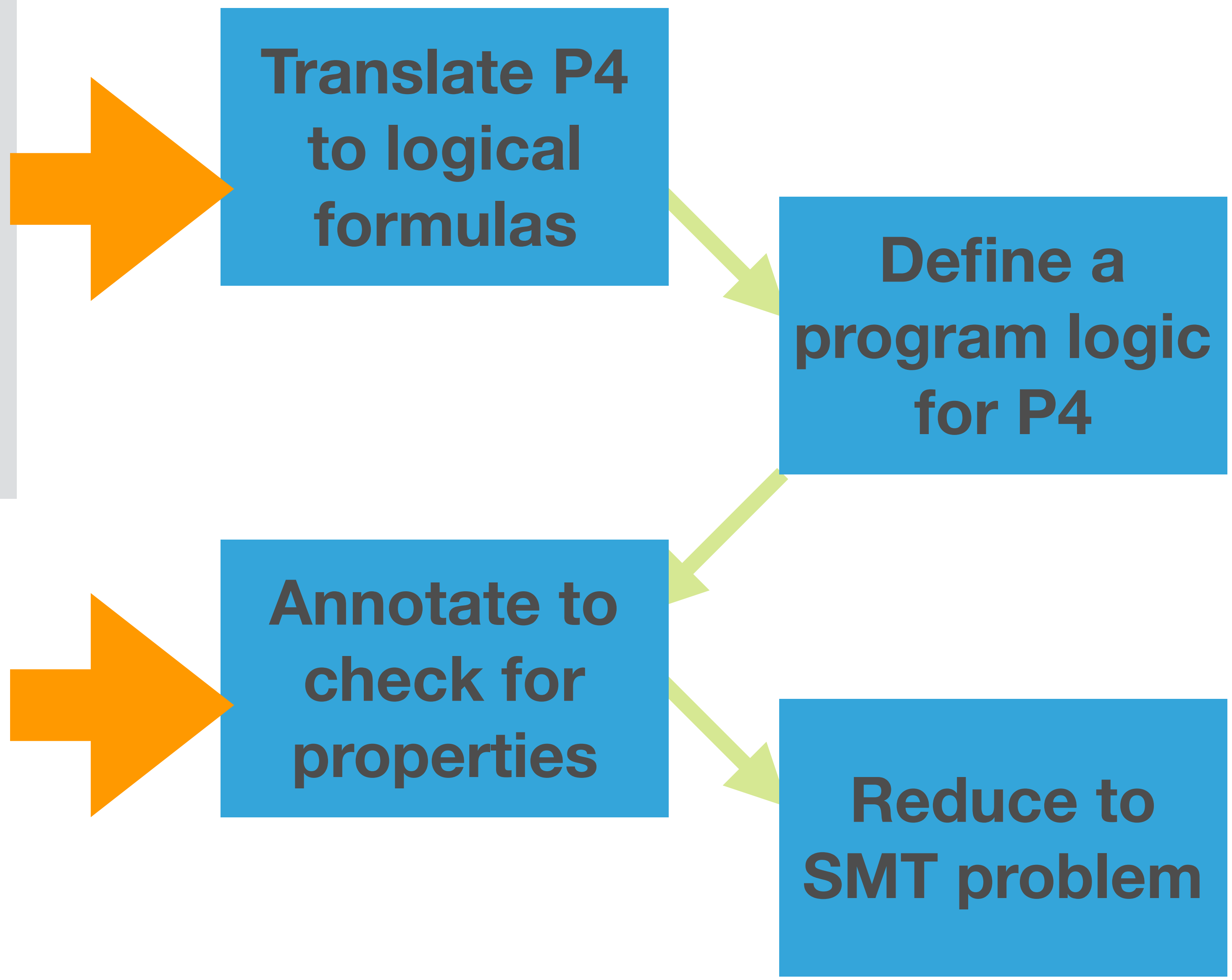
- **Allow programmers to express symbolic constraints on the control plane in terms of predicates on data plane state**
- **Combined, the control plane and data plane behave as expected**

Verification Challenges

Challenge	Solution
P4 does not have a formal semantics	We had to define one via translation
What should the annotations look like?	Leveraged our domain-specific knowledge to define language
How do we make the solver scale?	Standing on the shoulders of giants, e.g., passivization [Flanagan and Saxe, POPL 2001]

P4v : Basic Approach

```
action forward(p) { … }
table T {
  reads {
    tcp.dstPort;
    eth.type;}
  actions {
    drop;
    forward; } }
```

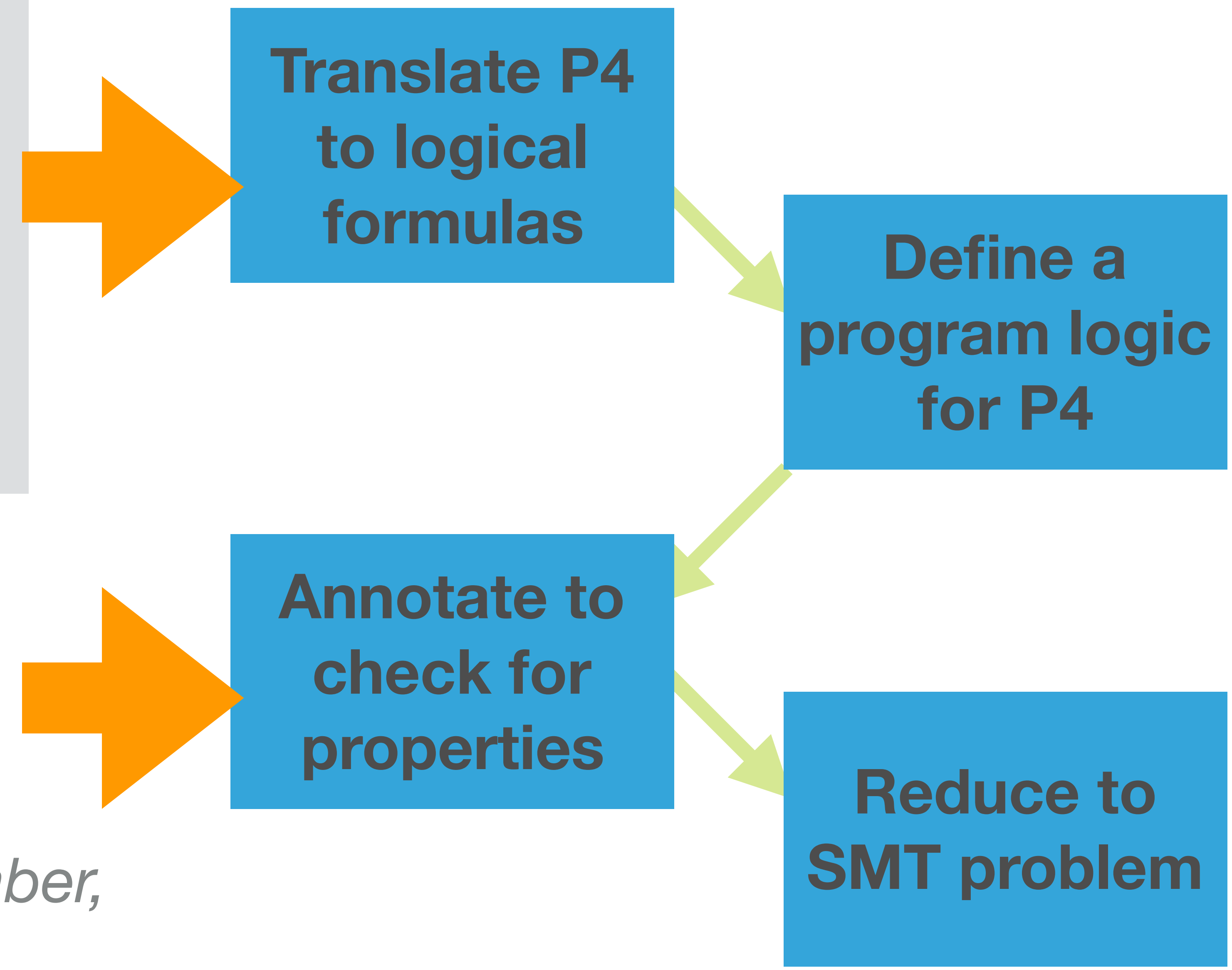


Desired Property:

"If the tcp.dstPort is 22, then drop the packet."

P4v : Basic Approach

```
action forward(p) { … }
table T {
  reads {
    tcp.dstPort;
    eth.type;}
  actions {
    drop;
    forward; } }
```



Desired Property:

"If the round number of arriving packet is greater than the stored round number, then drop the packet."

CCR Paper Bug

```
@pragma assume valid(paxos) implies local.round <= paxos.rnd
 apply(rount_table) {
    if (local.round <= paxos.rnd) { apply(acceptor_table) }
 }
@pragma assert valid(paxos)  implies local.set_drop == 0
```

Action failed to set the "drop flag" when the arriving round number is greater than the stored round number.

Evaluation

- Ran our verifier on a diverse collection of 13 P4 programs
  - Conventional forwarding: Router, NAT, Switch
  - Source routing: ToR, VPC
  - In-network processing: Paxos, LinearRoad
- Most finished in 10s of ms; switch.p4 finished in 15 seconds.

Only system to verify switch.p4

Outlook

Summarizing

- System artifact that can achieve orders-of-magnitude improvements in performance
  - Identified techniques for programming within fundamental hardware constraints
- Novel re-interpretation of the Paxos algorithm
  - Hopefully add clarity through a different perspective
- Mechanized proof of correctness of the implementation

A Few Lessons Learned

- What are good candidate applications for network acceleration?
  - "Squint a little bit, and they look like routing"
  - Applications with transient state, rather than persistent
  - Services that are I/O bound
  - Network acceleration helps latency, but throughput is the big win

What's Next?

- Very exciting time for networking and systems
- Network programmability provides an amazing opportunity to revisit the entire stack
- Redesign systems using an integrated approach, combining databases, networking, distributed systems, and PL

http://www.inf.usi.ch/faculty/soule/