Consensus as a Network Service

Huynh Tu Dang, Pietro Bressana,
Han Wang, Ki Suh Lee, Hakim Weatherspoon,
Marco Canini, Fernando Pedone, and Robert Soulé
Università della Svizzera italiana (USI),
Cornell University, and Université catholique de Louvain

Consensus is a Fundamental Problem

- Consensus protocols are the foundation for fault-tolerant systems
  - E.g., OpenReplica, Ceph, Chubby
- Many distributed problems can be reduced to consensus
  - E.g., Atomic broadcast, atomic commit

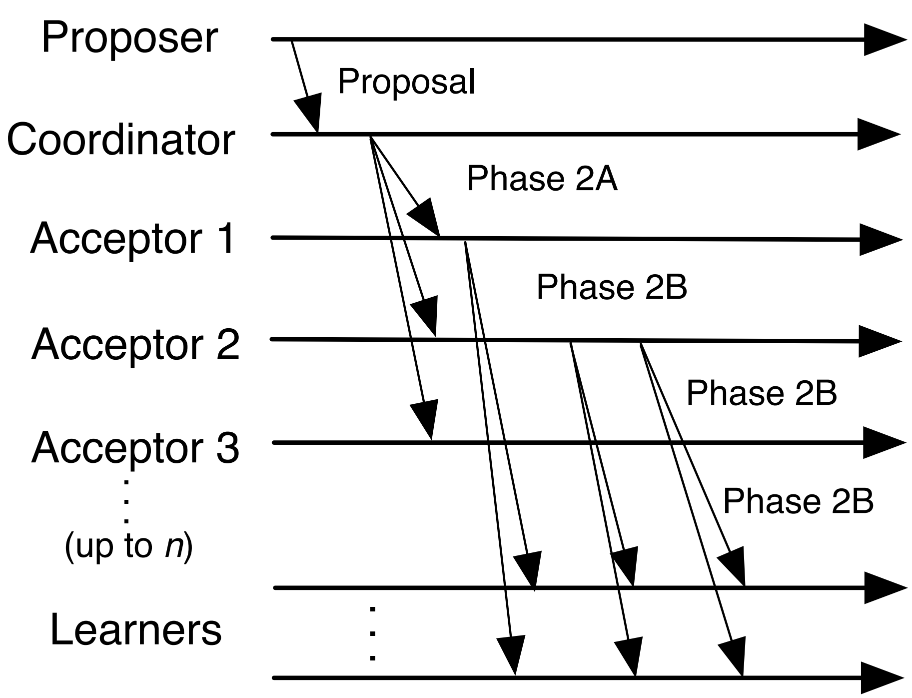
Key Idea: Move Consensus Into Network Hardware

- This work focuses on Paxos
    - One of the most widely used consensus protocol
    - “There are two kinds of consensus protocols: those that are Paxos, and those that are incorrect”,
      attributed to Butler Lampson
- Enabling technology trends:
    - Hardware is becoming more *flexible*: e.g. PISA, FlexPipe, NFP-6xxx
    - Hardware is becoming more *programmable*: e.g., POF, PX, and P4

Outline of This Talk

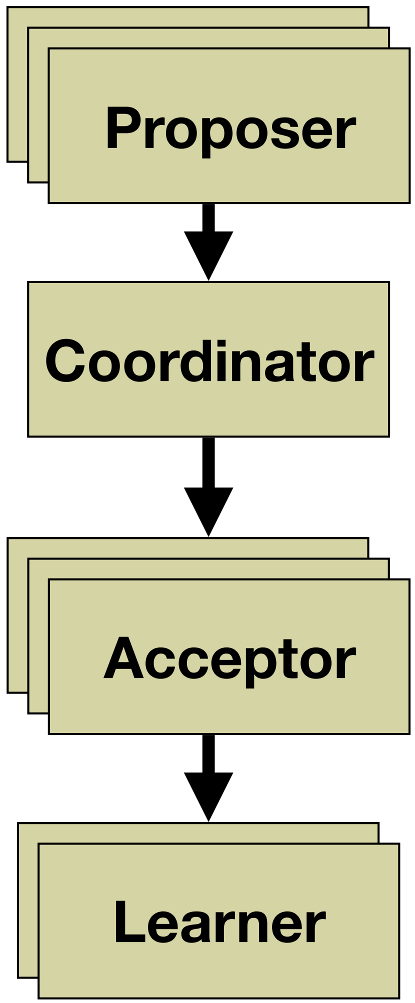
- Introduction
- **Background and Motivation**
- Design
- Implementation
- Evaluation
- Conclusions

Paxos 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 Functionality and Requirements



Issue requests. Craft a message with value.

Advocate requests. Add a sequence number to the message.

Choose a value and provide memory. Execute logic, keep persistent state.

Provide replication. Return chosen value to the application via a callback.

Design

Design Goals 1: Be a Drop-In Replacement

- Istvav et al. [NSDI ’16] implement ZAB in an FPGA, but require that the application also be implemented in the FPGA
- High-level languages make hardware development easier
- Implementing LevelDB in P4 might still be tricky….

Standard Paxos API

```
void submit(struct paxos_ctx * ctx,
            char * value,
            int size);

void (*deliver)(struct paxos_ctx* ctx,
                int instance,
                char * value,
                int size);

void recover(struct paxos_ctx * ctx,
             int instance,
             char * value,
             int size);
```

Standard Paxos API

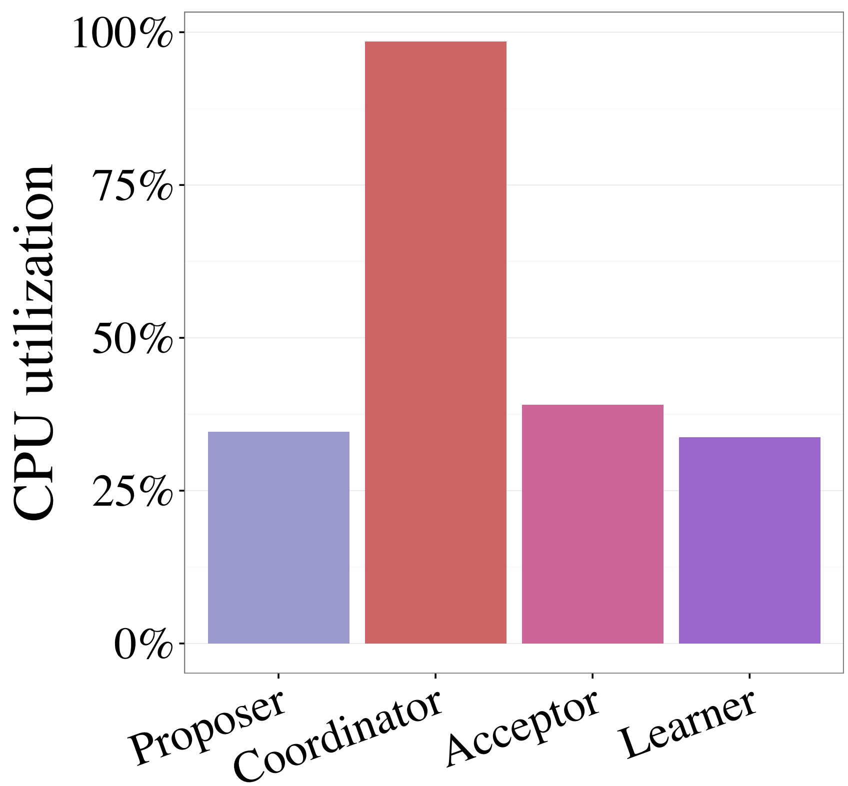
```
void submit(struct paxos_ctx * ctx,
            char * value,
            int size);

void (*deliver)(struct paxos_ctx* ctx,
                int instance,
                char * value,
                int size);

void recover(struct paxos_ctx * ctx,
             int instance,
             char * value,
             int size);
```

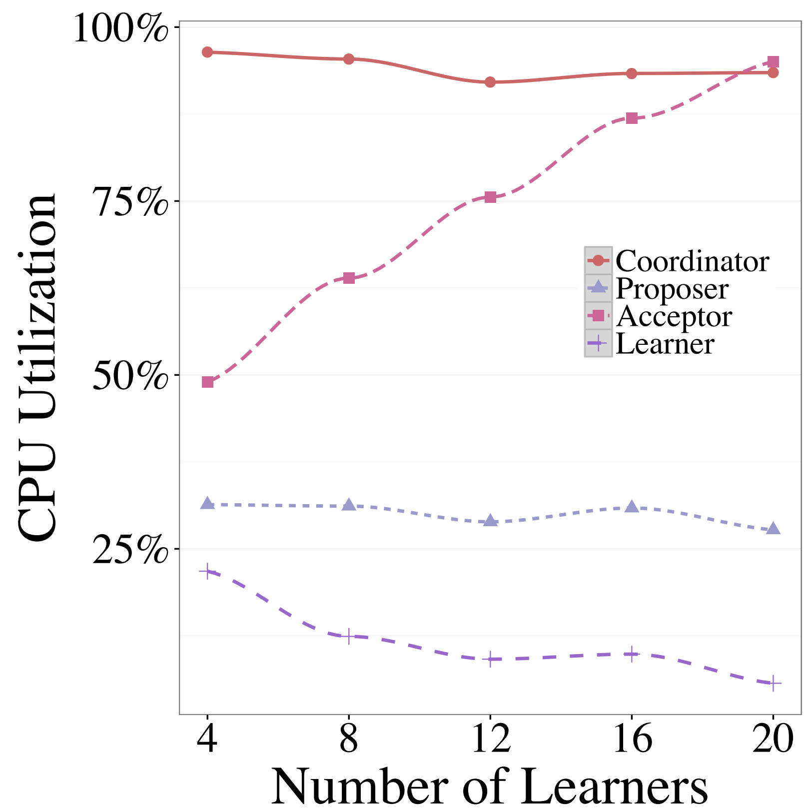
Send a value

Deliver a value

Discover prior value

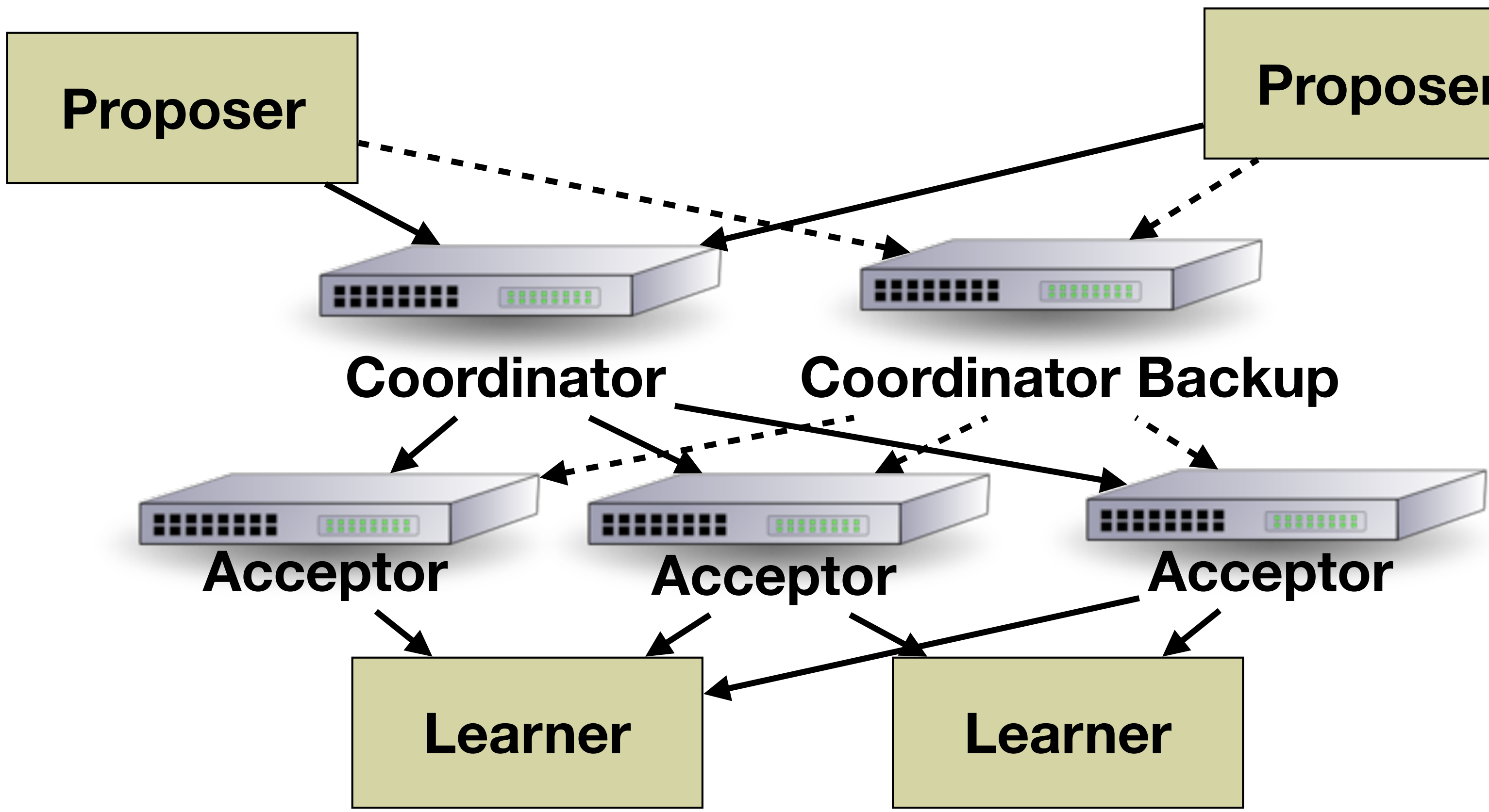
Design Goals 2: Alleviate Bottlenecks



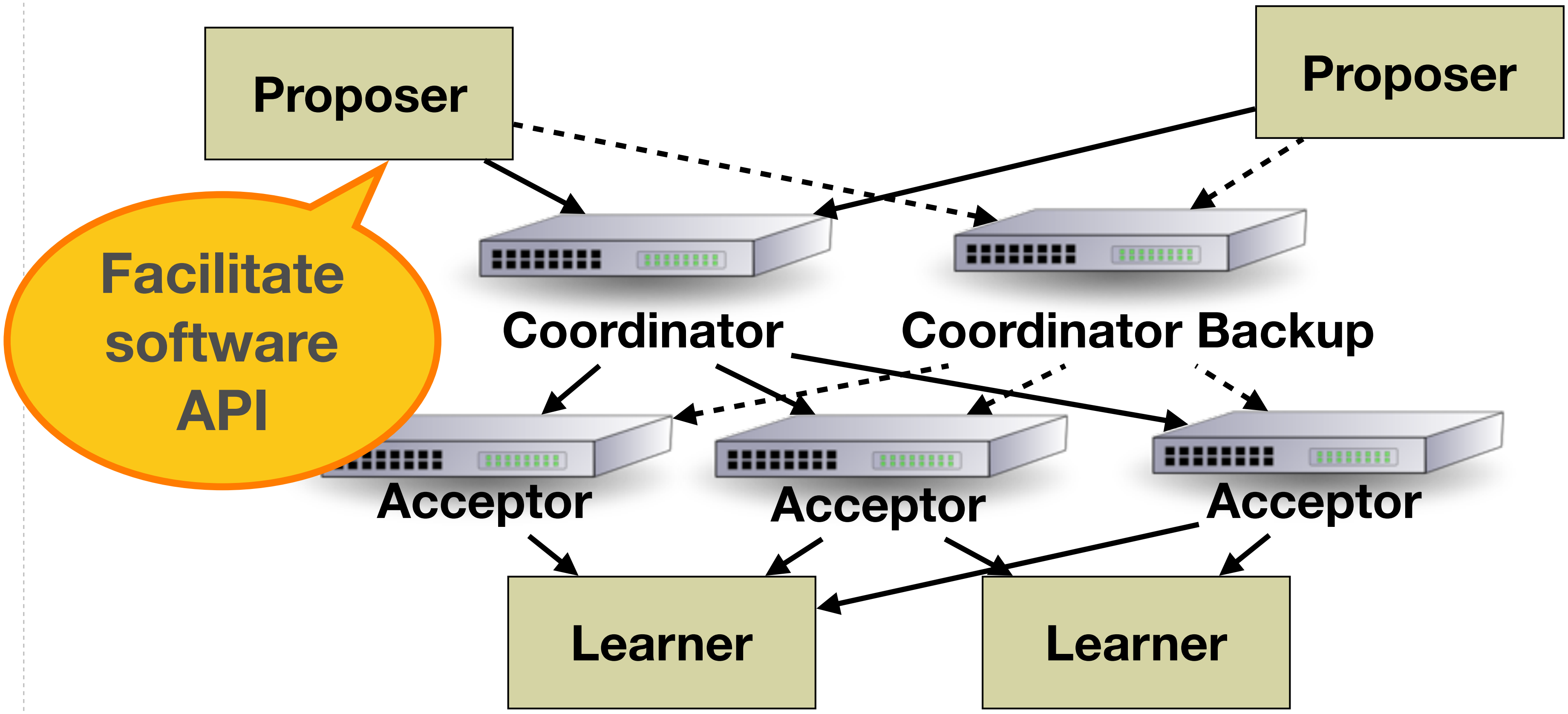


Coordinator and acceptors are to blame!

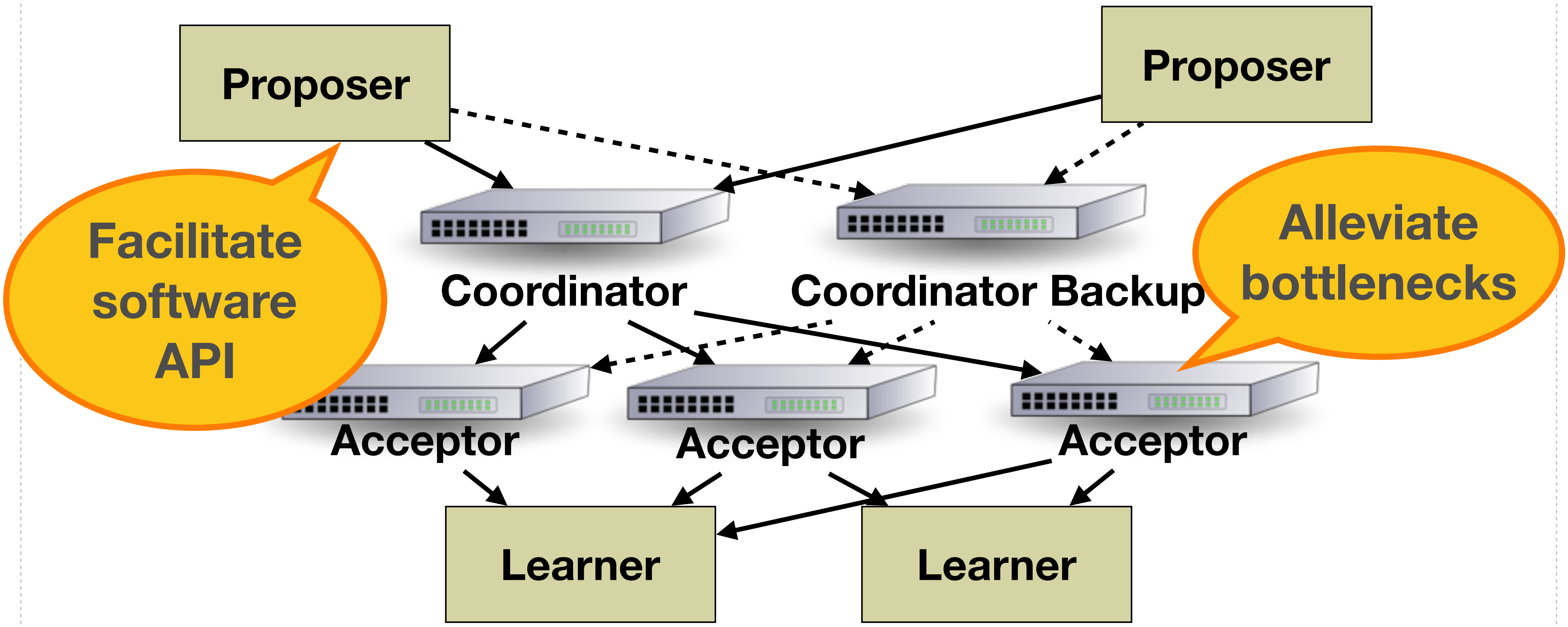
Hardware/Software



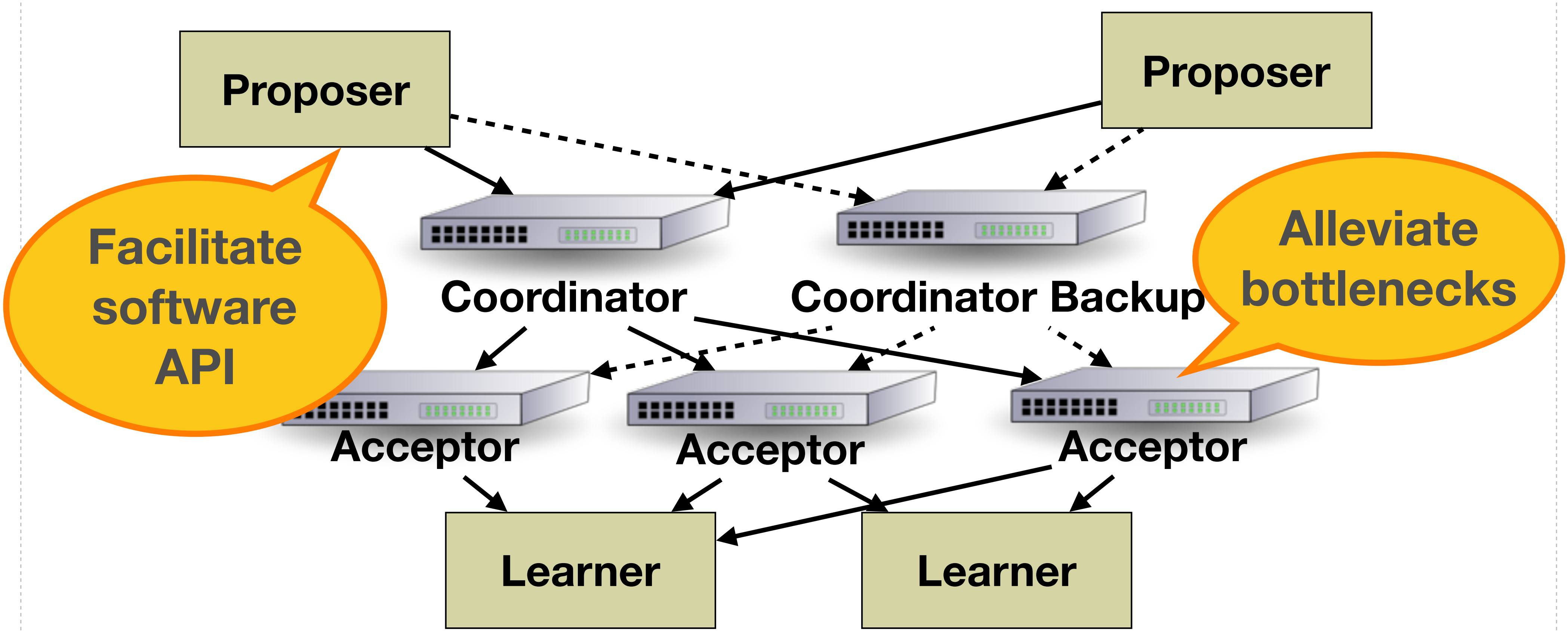
Hardware/Software



Hardware/Software



Hardware/Software



Challenge: map Paxos logic into stateful forwarding decisions

Paxos Header Format

- Network devices don't create messages, only forward them
- Header is union of all Paxos message fields
- Tradeoff: larger instance number allows for pre-initialization; requires more space

```
header_type paxos_t {
    fields {
        msgtype : 8;
        inst : INST_SIZE;
        rnd : 8;
        vrnd : 8;
        swid : 64;
        value : VALUE_SIZE;
    }
}
```

Implementation

Implementation

- Source code
  - Proposer and learner written in C
  - Coordinator and acceptor written in P4
- 3 Compilers
  - P4FPGA
  - Netronome Open-NFP
  - Xilinx SDNet

- 4 Hardware target platforms
  - NetFPGA SUME (4x10G)
  - Netronome AgilIO-CX (1x40G)
  - Alpha Data ADM-PCIE-KU3 (2x40G)
  - Xilinx VCU109 (4x100G)

P4 Tools

- **P4FPGA**: *P4 to Bluespec to FPGA. We implemented some code by hand in Bluespec, and optimized by hand (e.g., naive translation produced 6 tables, only needed 3).*
- **Xilinx SDNet:** *P4 to PX to FPGA. We wrote a Verilog wrapper around SDNet IP block, matching SDNet interface to SUME interface*
- **Netronome OpenNFP**: *Does not support register operations, uses a custom P4 syntax to call actions written in MicroC.*

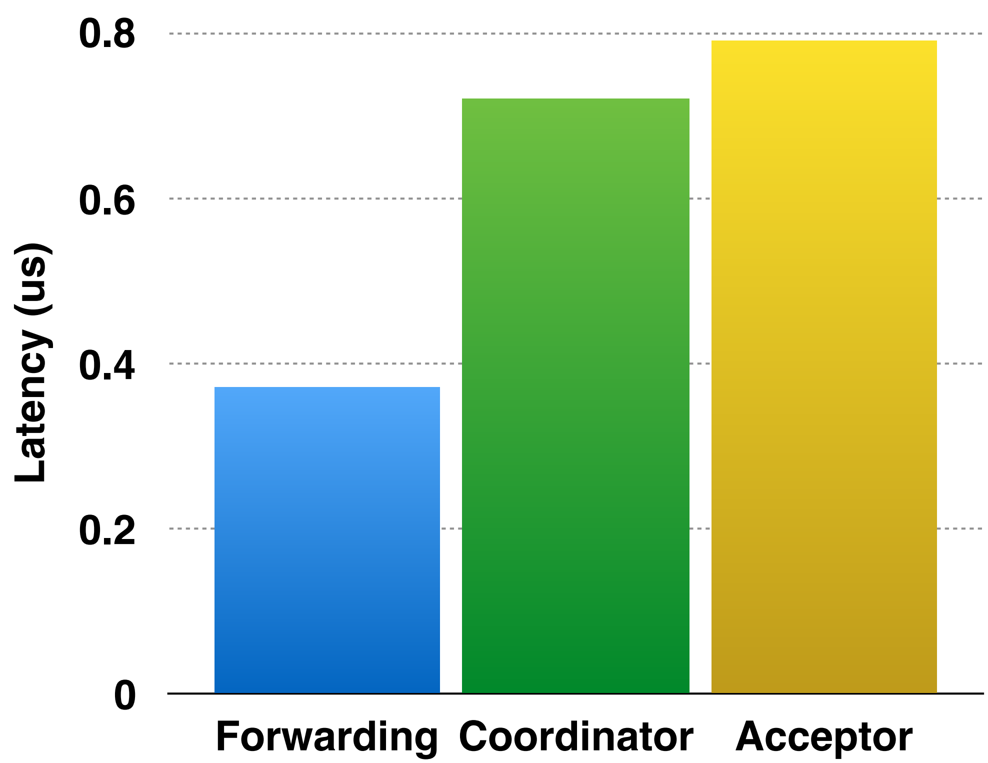
Evaluation

Experiments

- Focus on three questions:
  - What is the absolute performance?
  - What is the end-to-end performance?
  - What is the performance after failure?
- Testbed:
  - Four NetFPGA SUME boards in SuperMicro Servers
  - One Pica8 Pronto 3922 switch, 10Gbps links

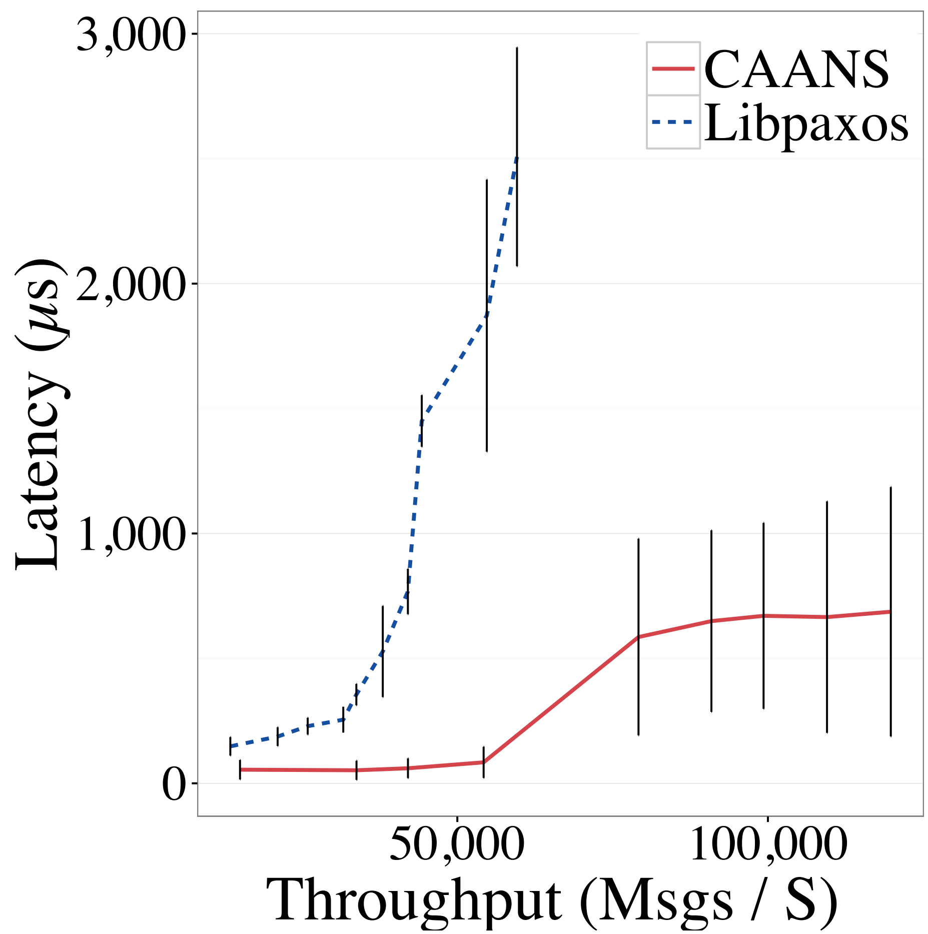
Absolute Performance

- Measured on NetFPGA SUME using P4FPGA
- Throughput is over **9 million consensus messages / second** (close to line rate)
- Little overhead latency compared to simply forwarding packets

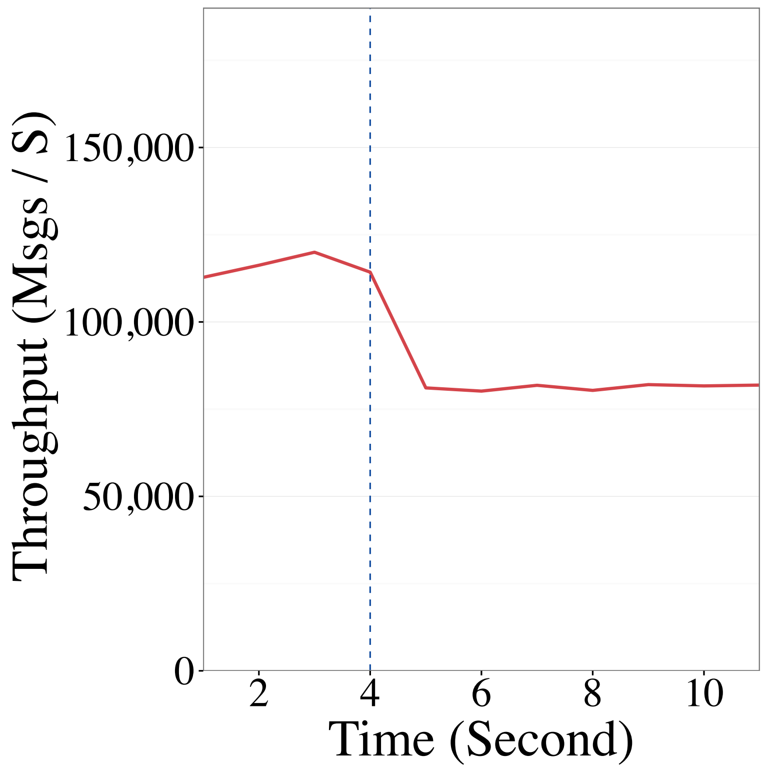


End-to-End Performance

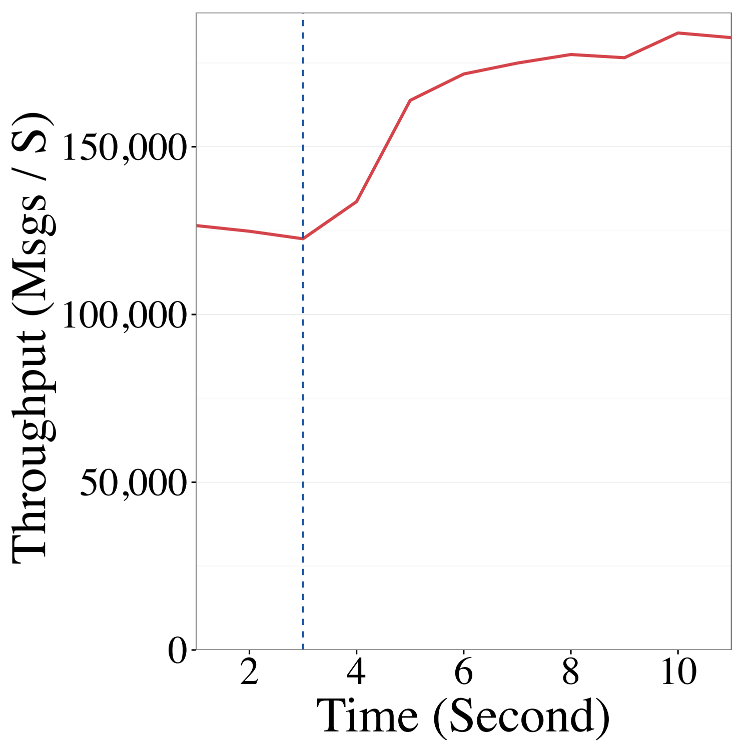
- Application discards result from “deliver” callback
- 2.24x throughput improvement over software implementation
- 75% reduction in latency
- Similar results when replicating LevelDB as application



Performance After Failure



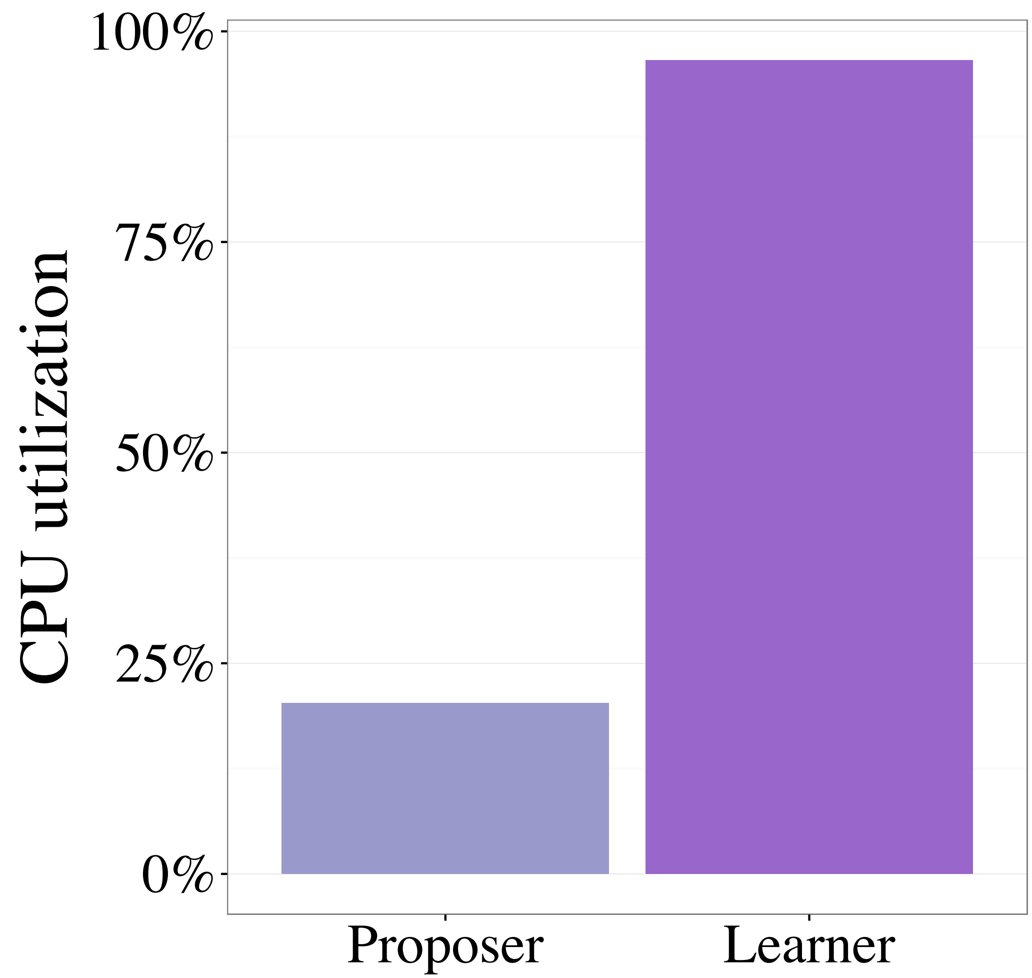
Coordinator failure with software backup



Acceptor failure

Conclusion

- We make consensus great again!
- The ball is now in the application developer's court
- Suggests direction for future work



Outlook

- The performance of consensus protocols has a dramatic impact on the performance of data center applications
- Moving consensus logic into network hardware results in significant performance improvements
- Suggests new line of research: don’t optimize the protocol, investigate how to handle lots of consensus messages

Acknowledgements

- Thank you to Gordon Brebner and Xilinx for donating two SUME boards, providing access to SDNet, performing measurements
- Thank you to Jici Gao, Mary Pham, Bapi Vinnakotam, and Netronome for providing us with a hardware testbed, and support with using their toolchain

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