stateless

Building a better network through disaggregation

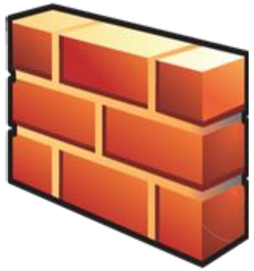
Eric Keller

Networks Need Network Functions

NAT

Intrusion Prevention

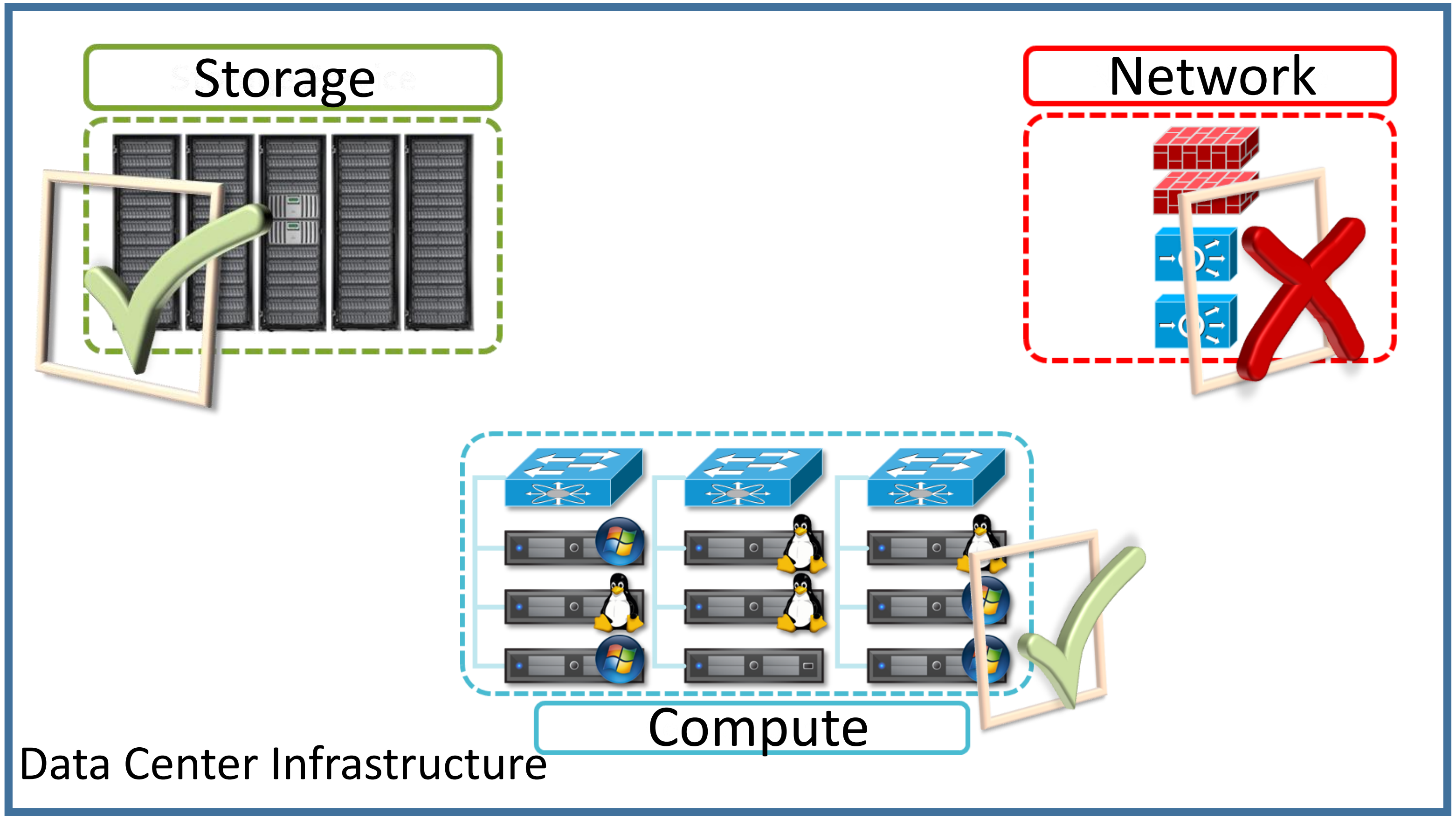
Firewall



Load balancer

To protect and manage the network traffic

Networks Need *Agile* Network Functions



To match the agility of today's (cloud) compute infrastructure

Network Agility == Ability to move quickly and easy

Seamless Scalability

Failure Resiliency

Instant Deployment

Without Sacrificing Performance

Virtual Network Functions to the Rescue ?

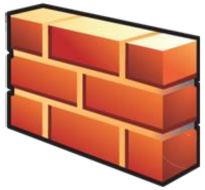
Hardware Network Functions

Software Network Functions
(Virtual Machines
or containers)

Same core architecture, same fundamental limit in agility

The Challenge is with The State

- **Firewall** : connection tracking information

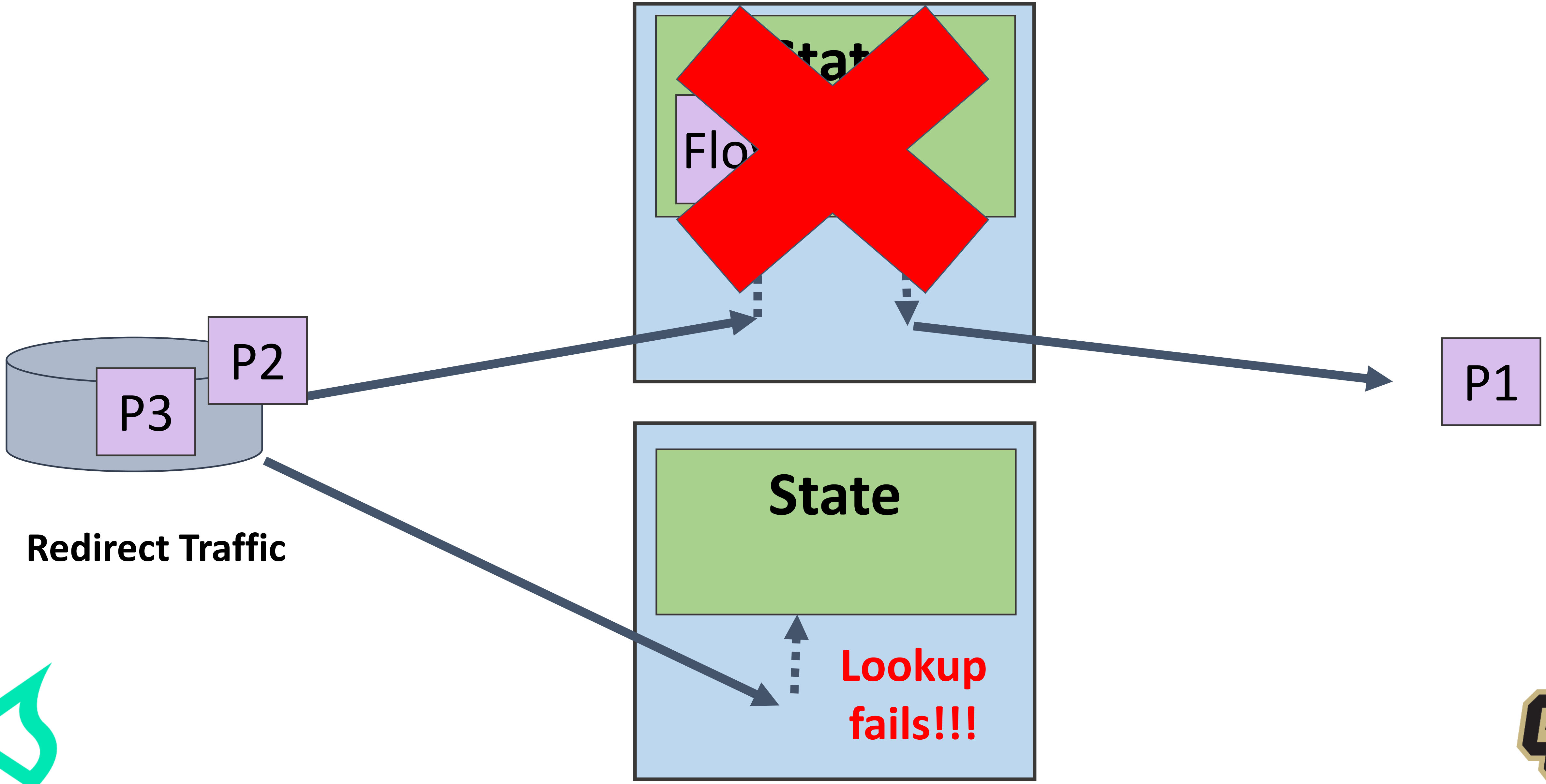


- **Load balancer**: mapping to back end server

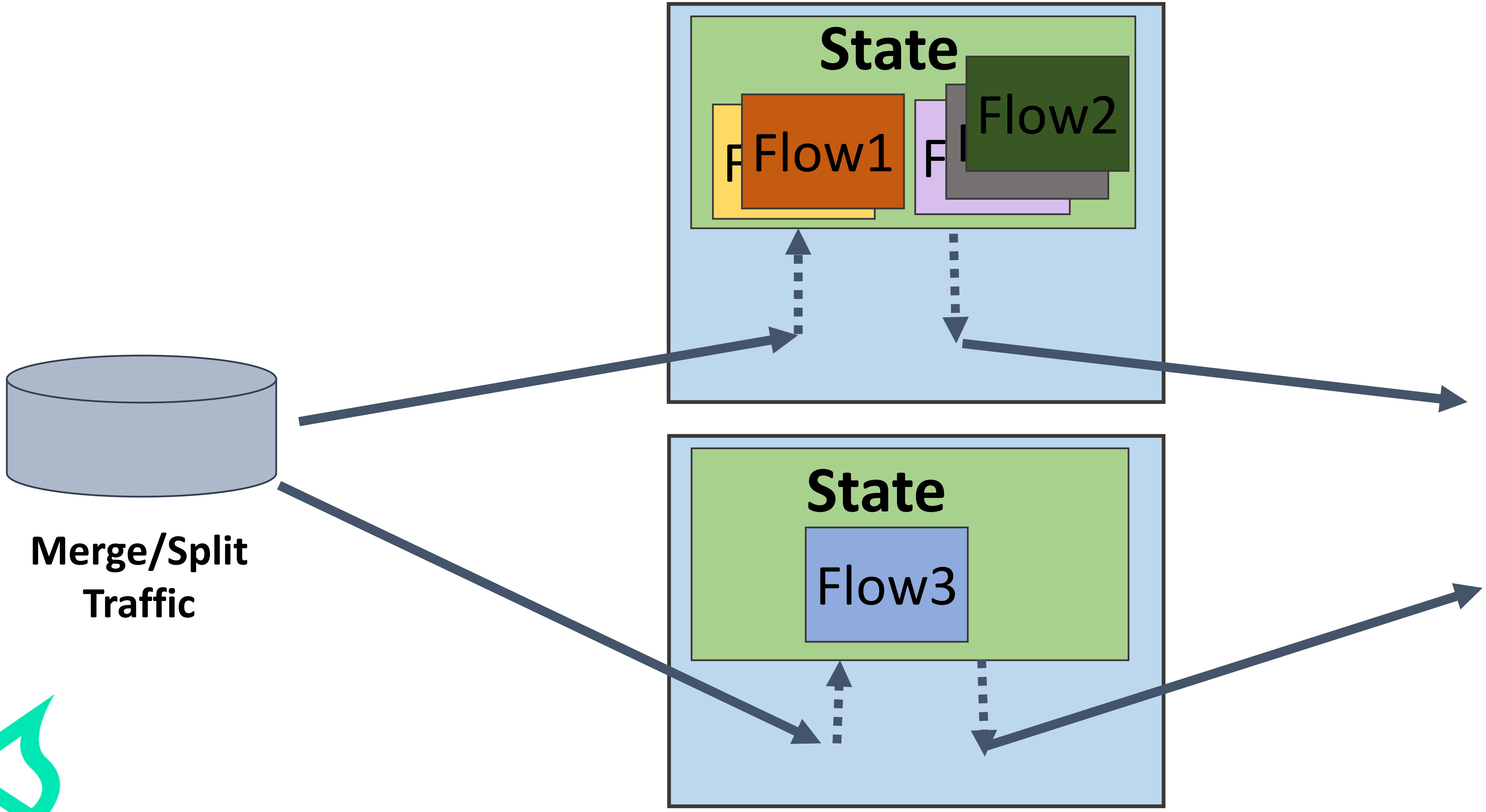
- **Intrusion Prevention**: automata state

- **NAT**: mapping of internal to external addresses

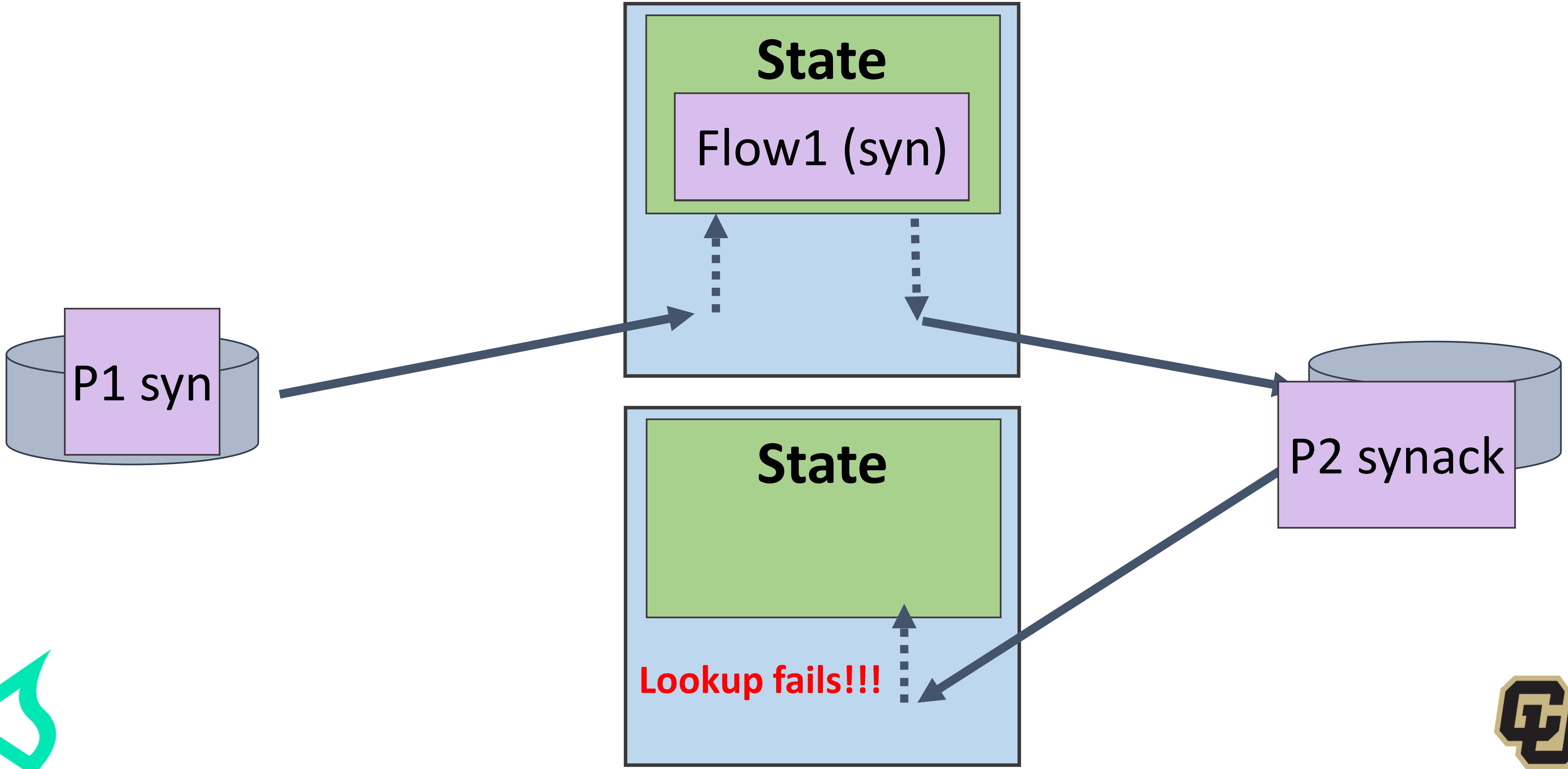
Example Problem 1: Failure



Example Problem 2: Scaling In and Out



Example Problem 3: Asymmetric / Multi-path



Other Solutions

Industry Approaches to Deal with State

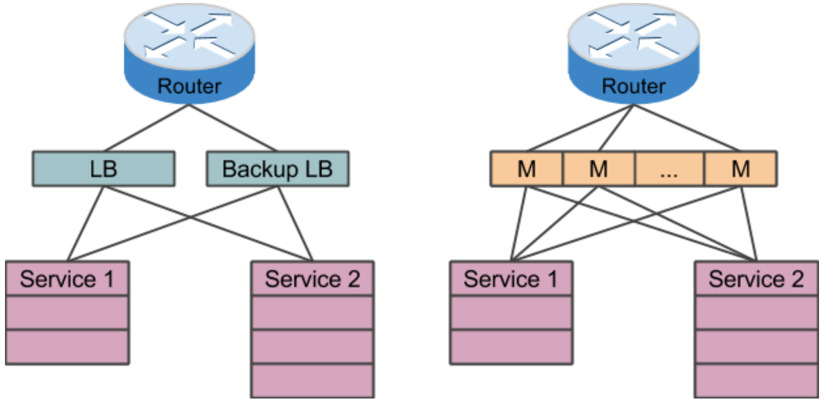
HA Pairs

- Doubles cost, limited scalability, unreliable [Jain2009]

Active　　　　Backup

Don’t use state

- e.g., Google Maglev
  - (hash 5-tuple to select backend).
  - Limited applications



Dealing with State: State Migration (for scaling)

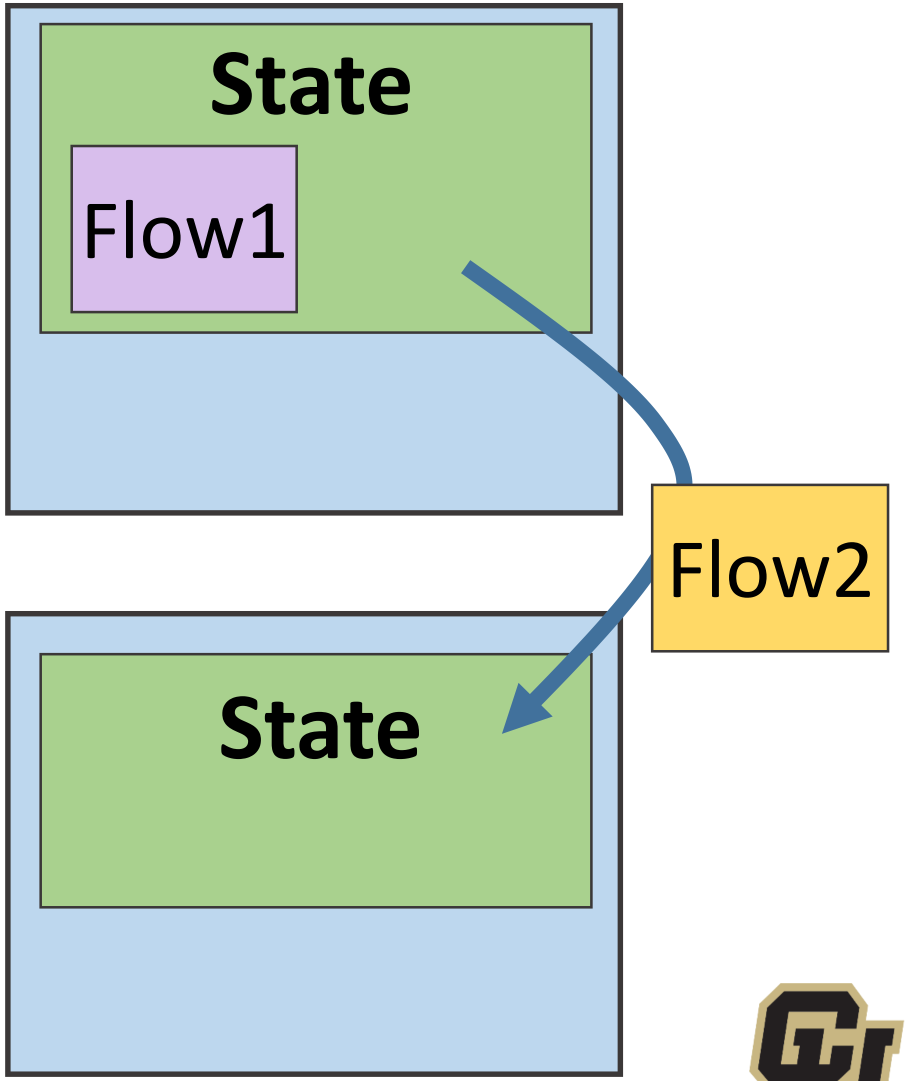
Router Grafting [NSDI 2010],

Split Merge [NSDI 2013],

OpenNF [SIGCOMM 2014]

- When needed, migrate the relevant state

- Only handles pre-planned events
- High overhead to migrate state (e.g., 100 ms)
- Relies on flow affinity



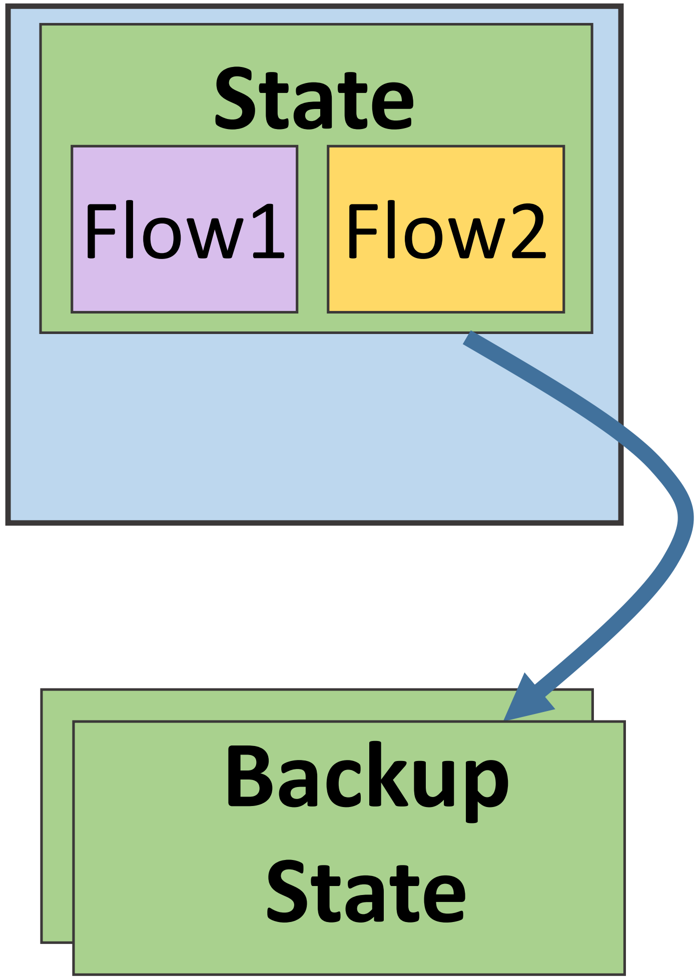
Dealing with State: Check Pointing (for failure)

Pico Replication [SoCC 2013]

- Periodically checkpoint state (only diffs, and only network state)

Limitations:

- Quick recovery from failure
- High packet latency (can't release packets until state check pointed)



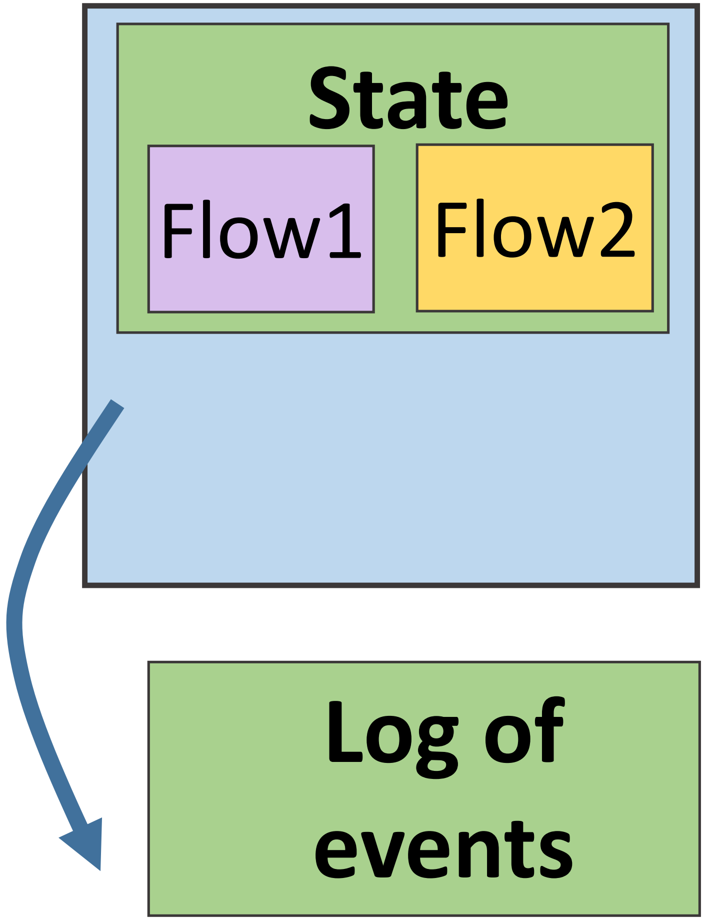
Dealing with State: Deterministic Replay (for failure)

FTMB [SIGCOMM 2015]

- Log events so that upon failure we can re-play those events to rebuild the state
- Use periodic check pointing to limit the replay time
- Improves packet latency

Limitation:

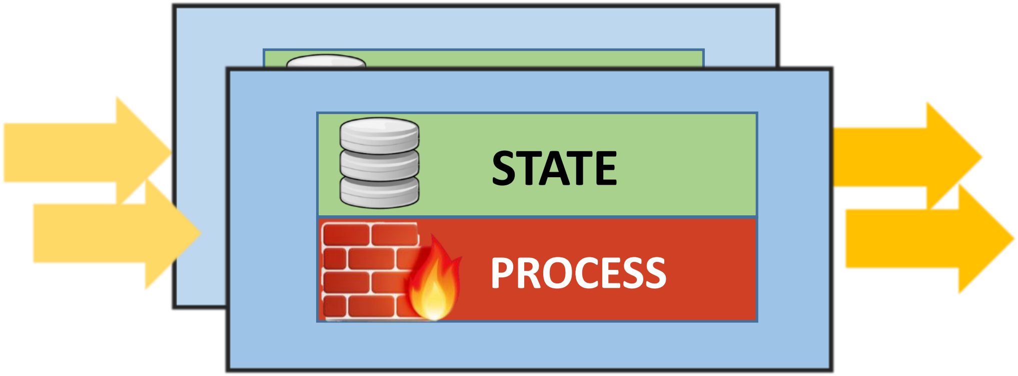
- Long recovery time (time since last check point)



What is the root of the problem?

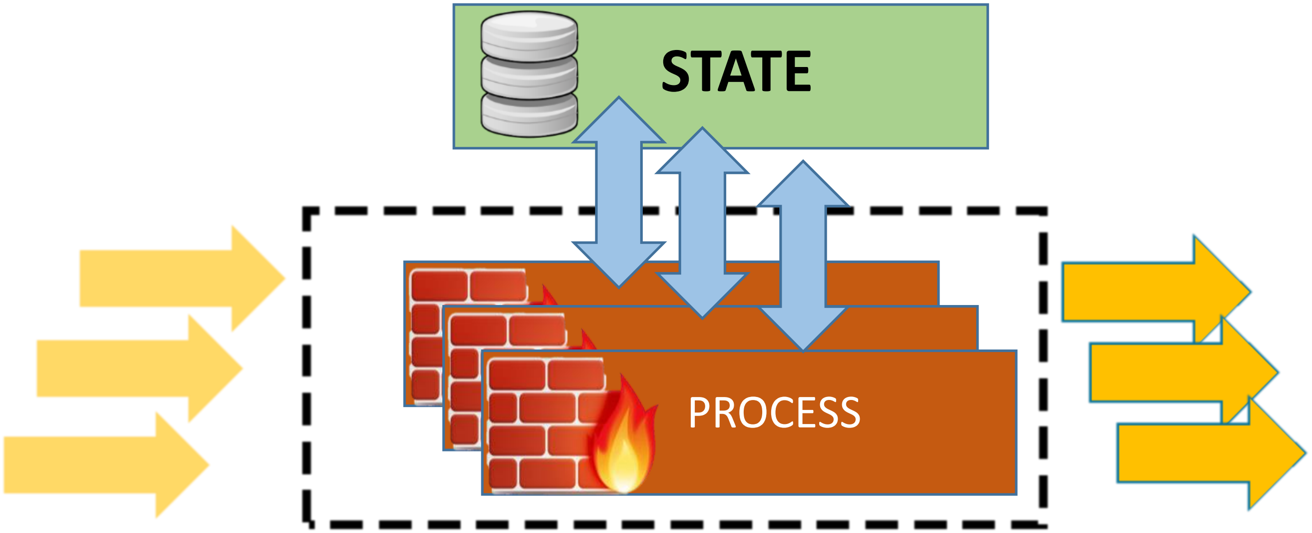
… Appliance mentality

Maintaining the Tight Coupling between State and Processing



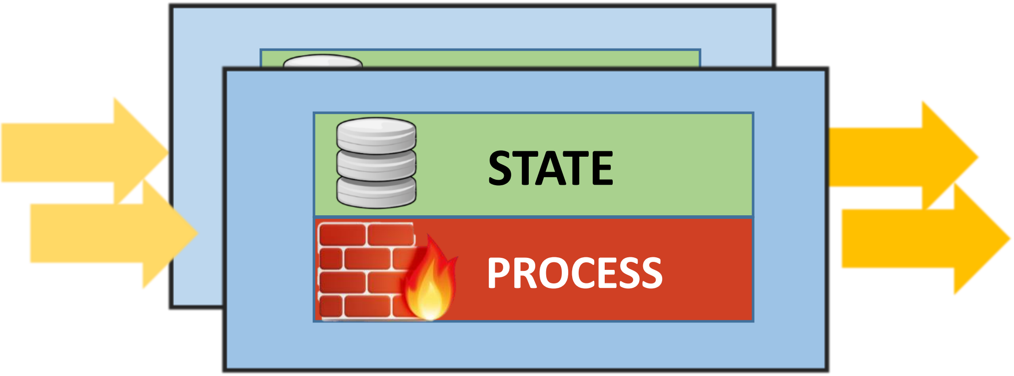
Stateless Network Functions

- Re-designed as a distributed system from the ground up.
- Decoupling the state from the processing



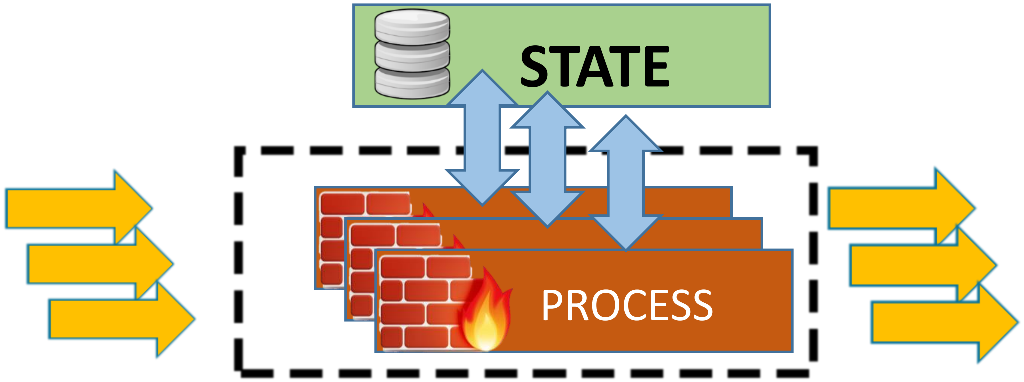
Benefits of Decoupling State from Processing

Traditional Network Function
e.g., Firewall



- High overhead to manage state
- Relies on flow affinity
- Hard to achieve both resiliency and elasticity

Stateless Network Function
e.g., Stateless Firewall



- Seamless elasticity
- No disruption in failure
- Doesn't rely on flow affinity
- Centralized state (simpler to manage)

Is this even possible?

We need to handle millions of packets per second

A Counter-Intuitive Proposal… But it is possible

Why we ***can*** do this:

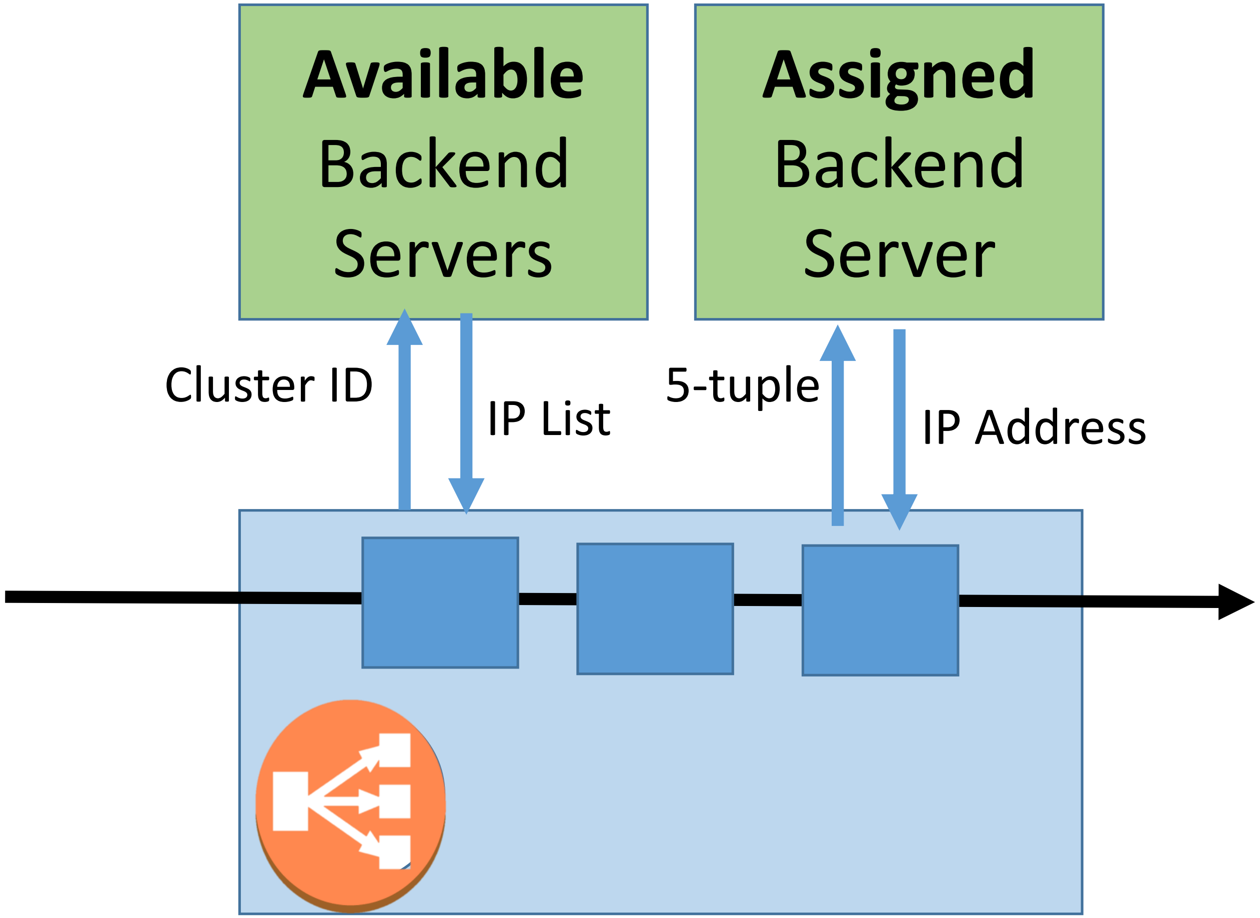
- Common packet processing pipeline has a lookup stage
  (so, per packet request to data store, but not lots of back and forth)

- Requests to data store are much smaller than packets
  (so, scaling traffic rates does not result in same scaling of data store)

- Advances in low-latency technologies
  (data stores, network I/O, etc.)

How State is Accessed

- Example for Load balancer



1st Packet of flow
(Pick an available server)

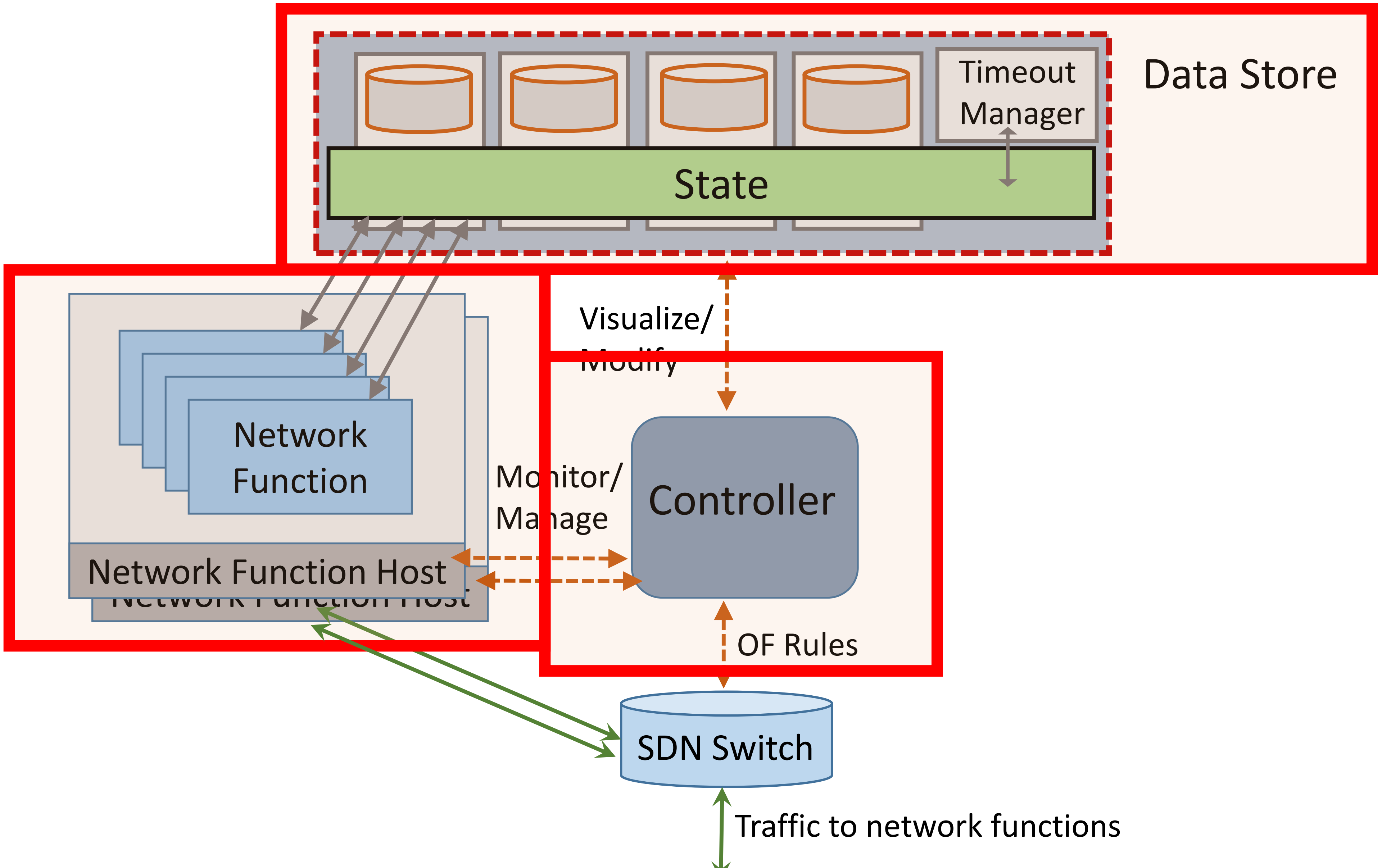
- **1 Read** from Available table,
- **1 Write** to Assigned table

Every other Packet of flow
(look up assigned server)

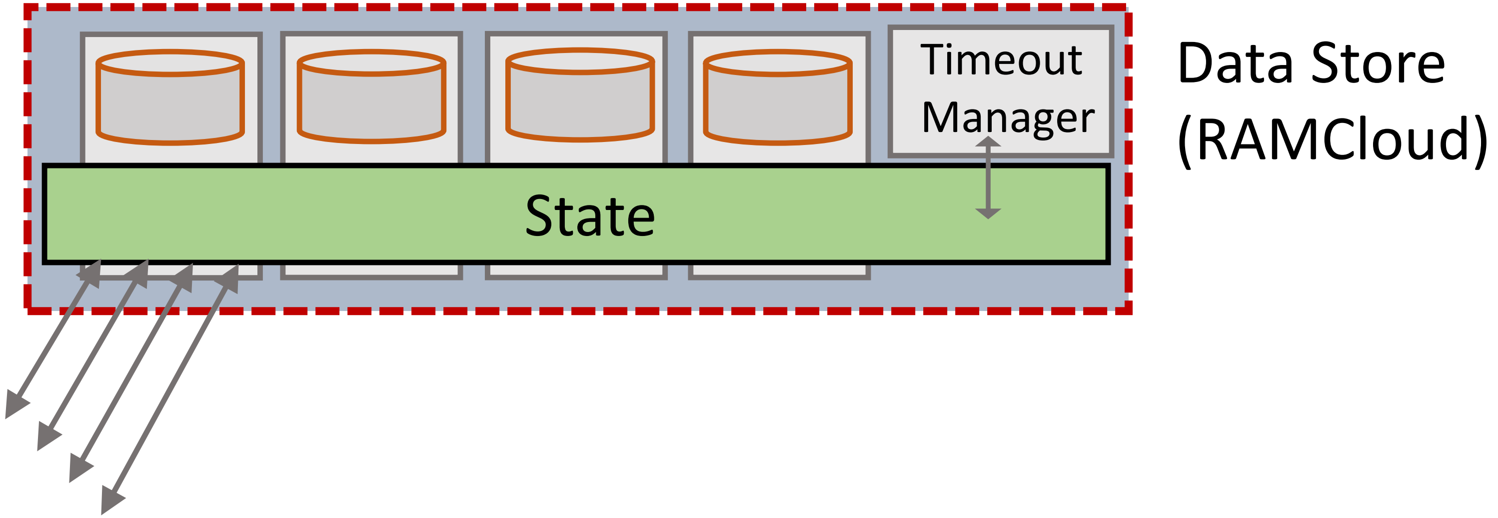
- **1 Read** from Assigned table

System Architecture
StatelessNF

StatelessNF Architecture

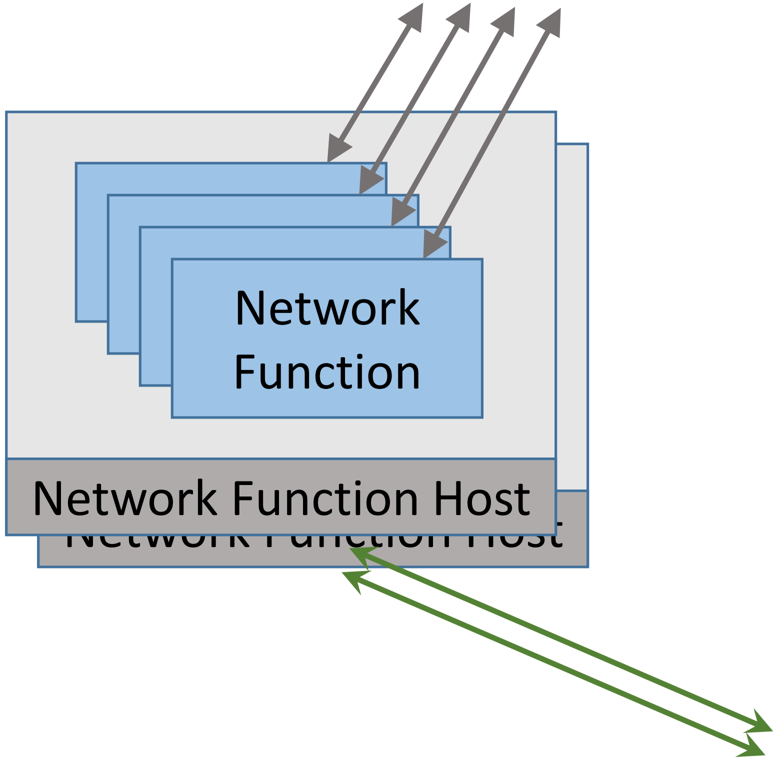


Data Store



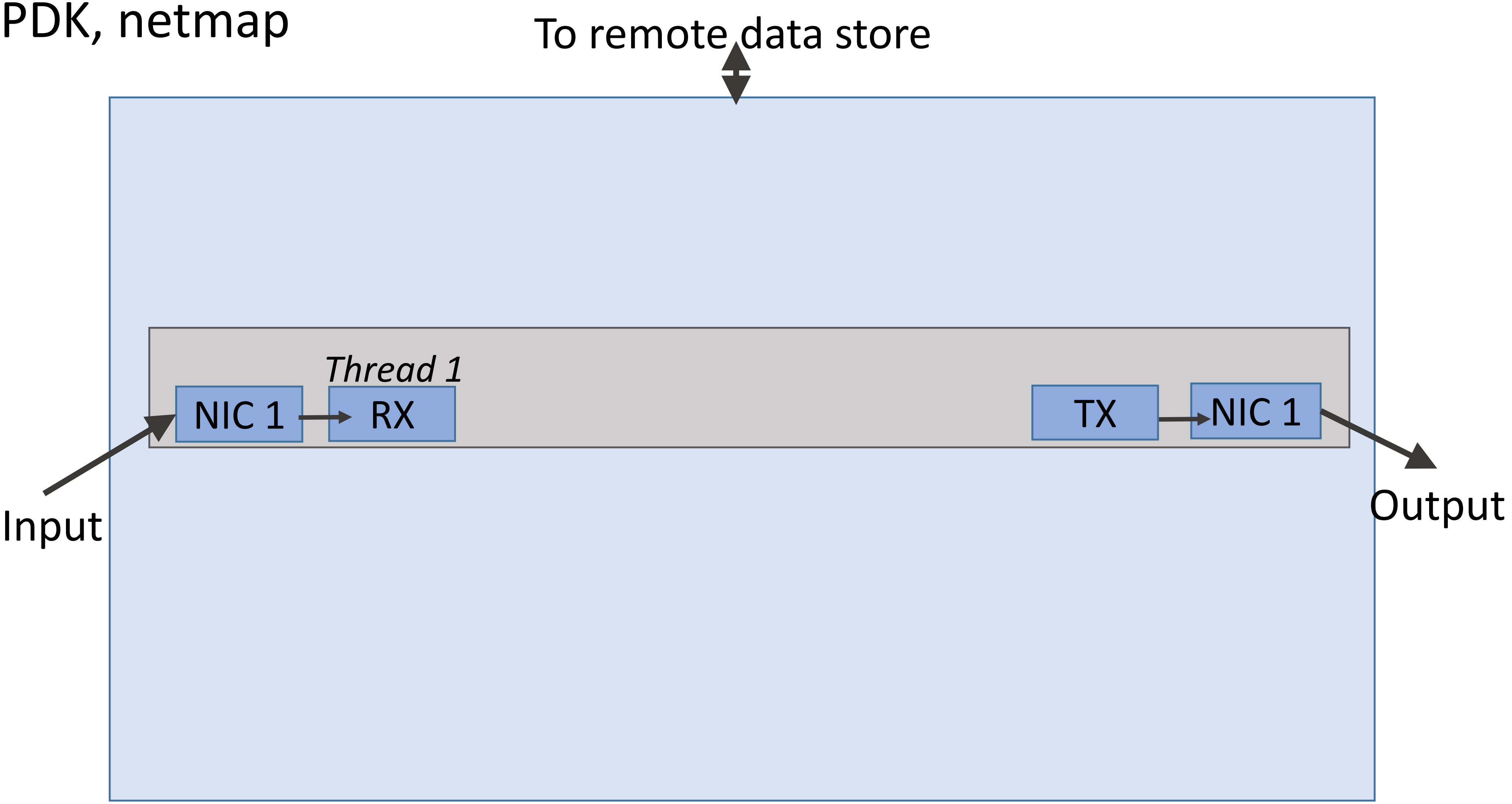
- Low latency, etc.
- Also needs (or could use) support for timers, atomic updates, queues

Network Function Instances



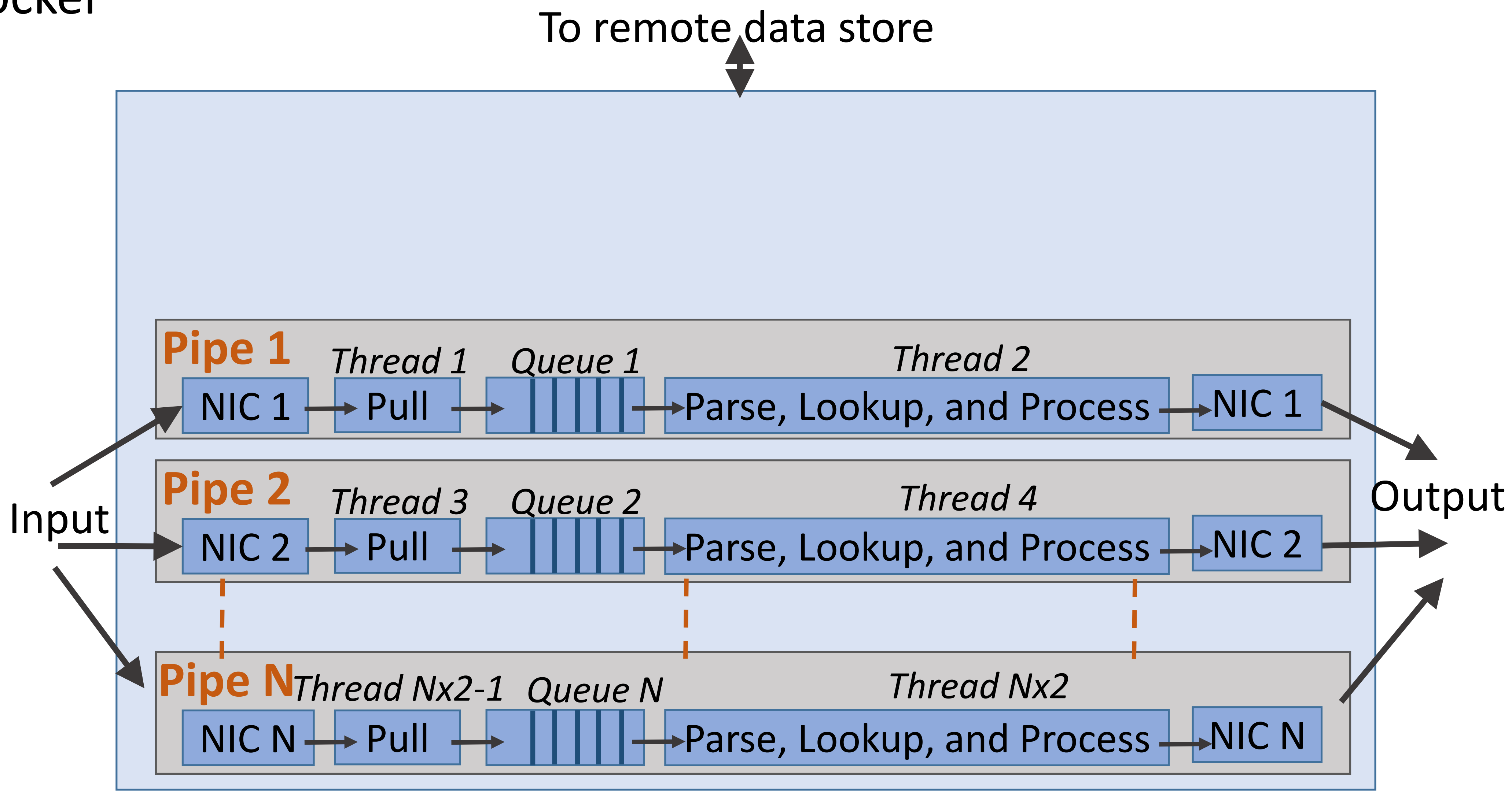
High-Performance Network I/O

e.g., DPDK, netmap



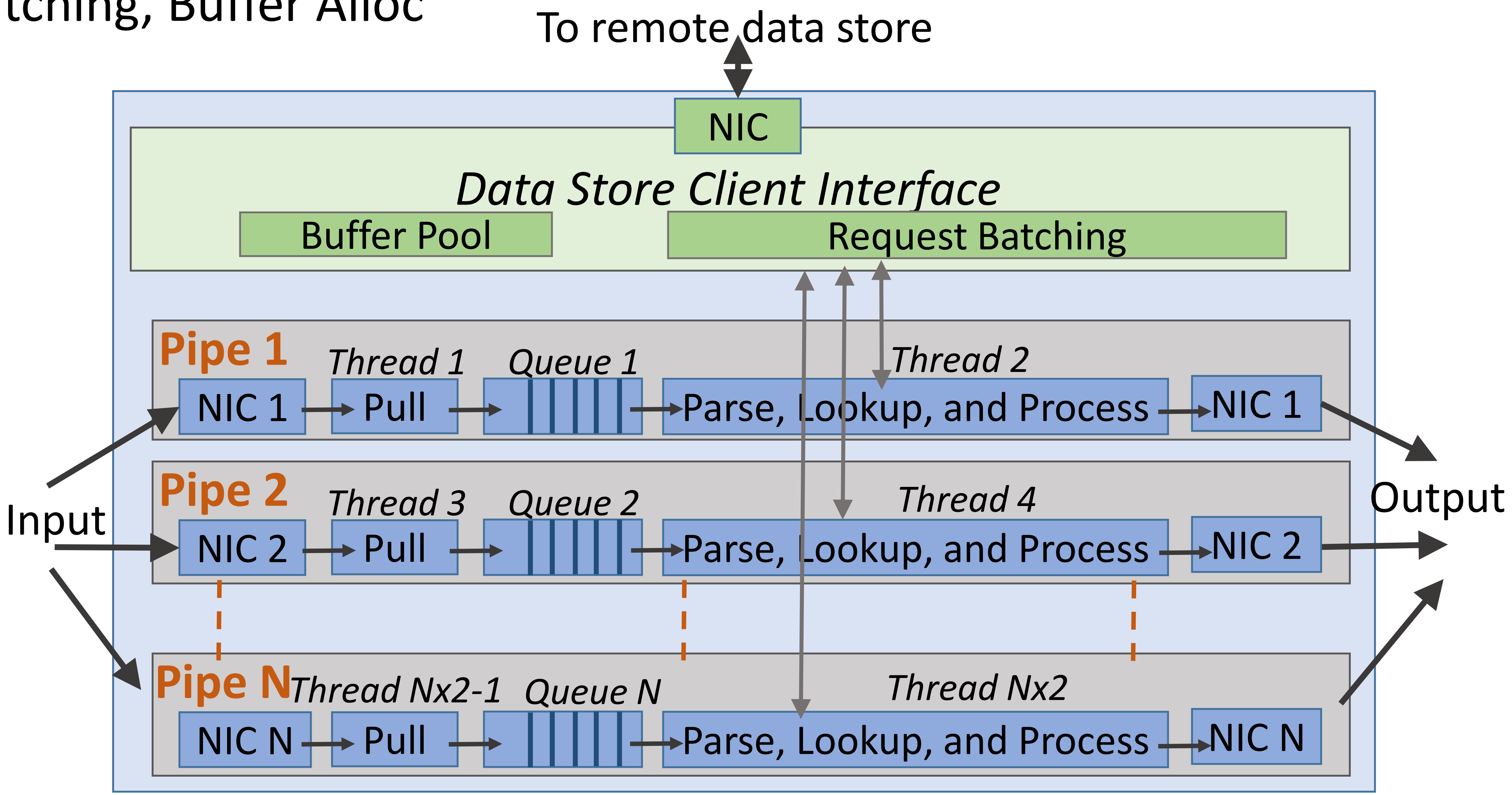
Deployable Packet Processing Container

e.g., Docker



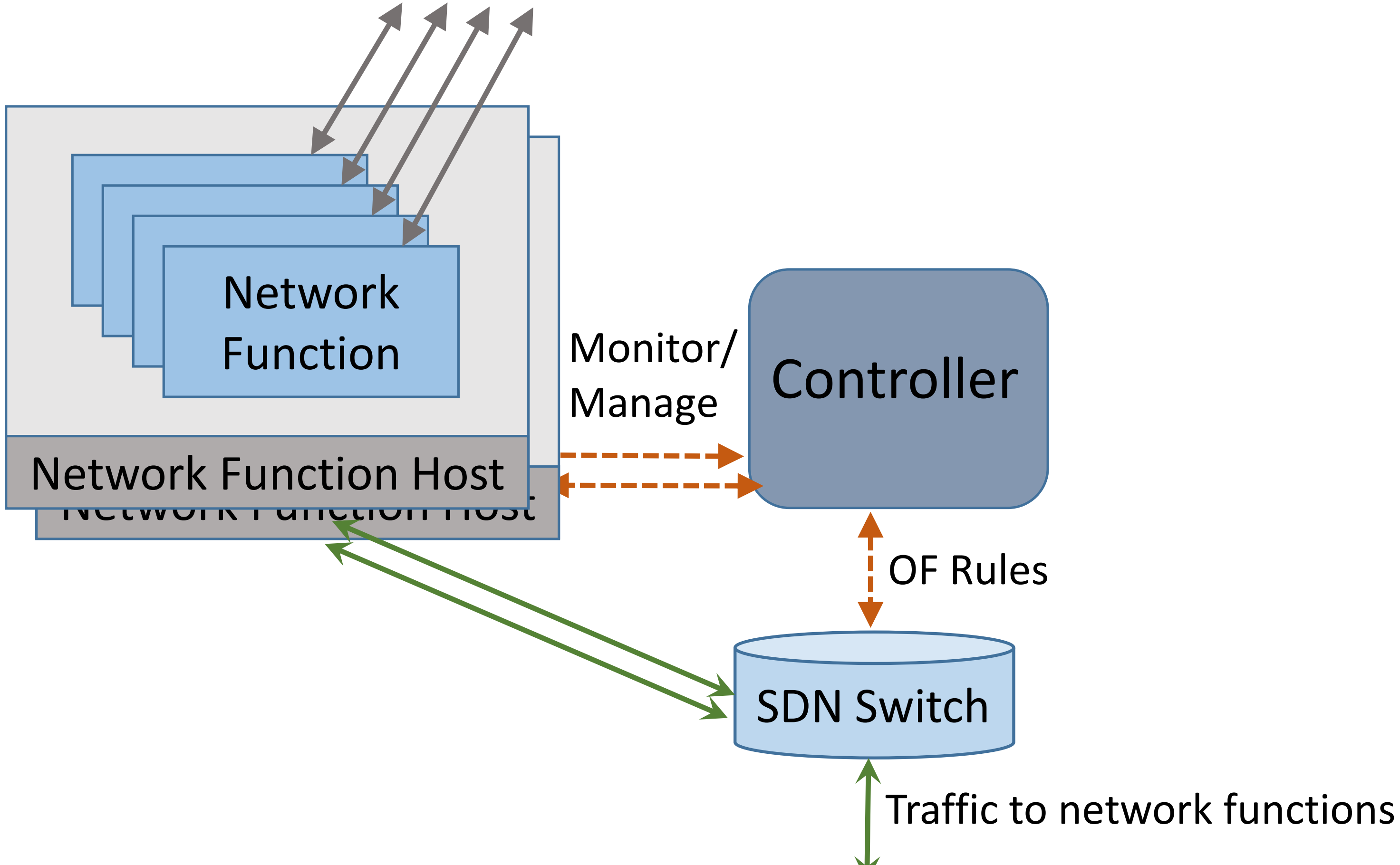
Optimized Data Store Client Interface

e.g., Batching, Buffer Alloc



Orchestration

- Failure handling – speculative failure detection (much faster reactivity)
- Scaling in and out – no need to worry about state when balancing traffic



Implementation

Network Functions (NAT, Firewall, Load balancer)

- DPDK
- SR-IOV
- Docker
- Infiniband to Data store (DPDK since paper)

Data store

- RAMCloud (Redis since paper)
- Extending

Controller

- Extended FloodLight, basic policies for handling scaling and failure. (complete re-write since paper)

StatelessNF System Evaluation

Evaluation

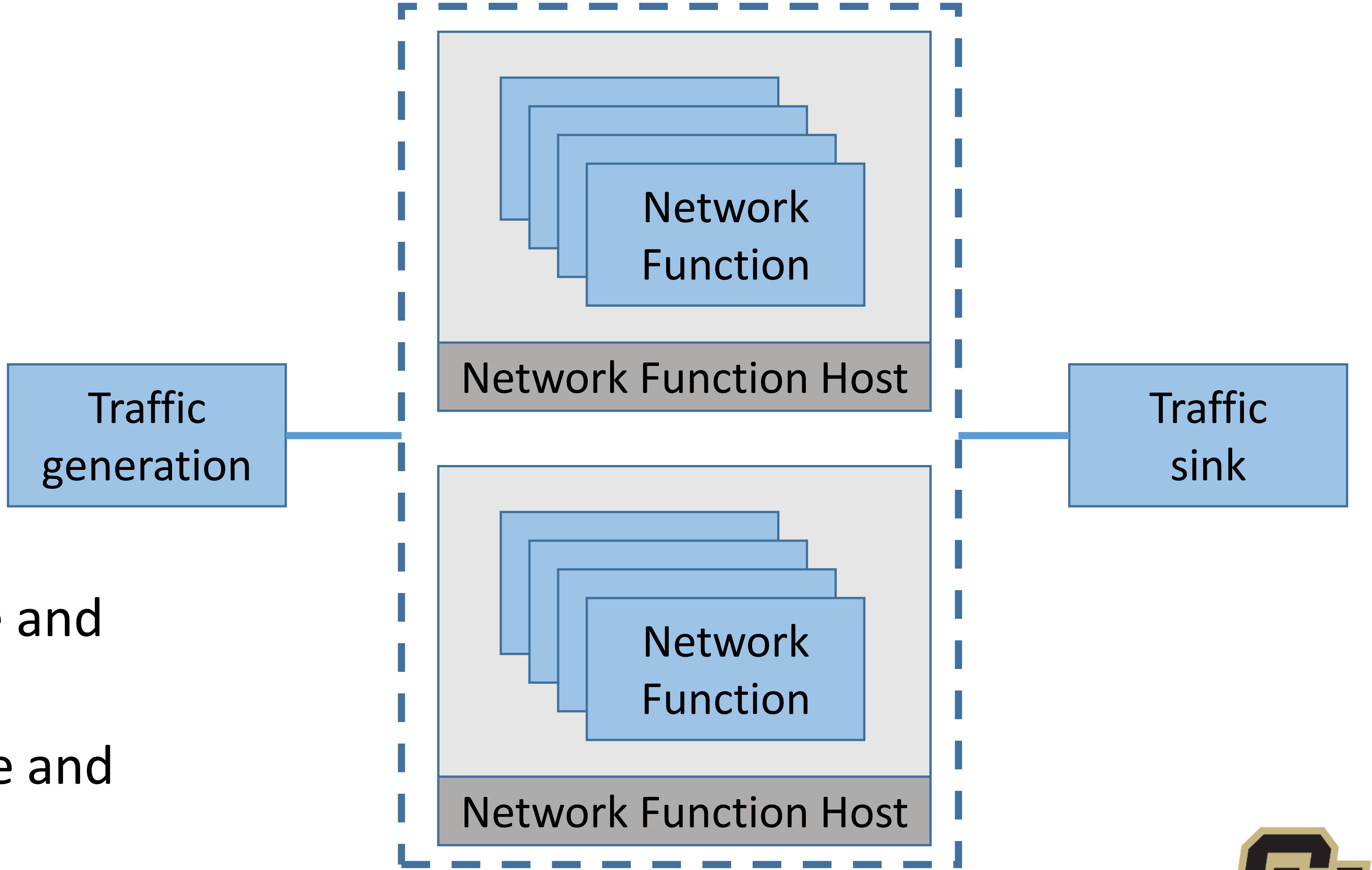
Goal: in this extreme case architecture, can we get **similar throughput and latency** as other software solutions,

but with better **handling of resilience and failure**?

Experiment Setup

Tests:

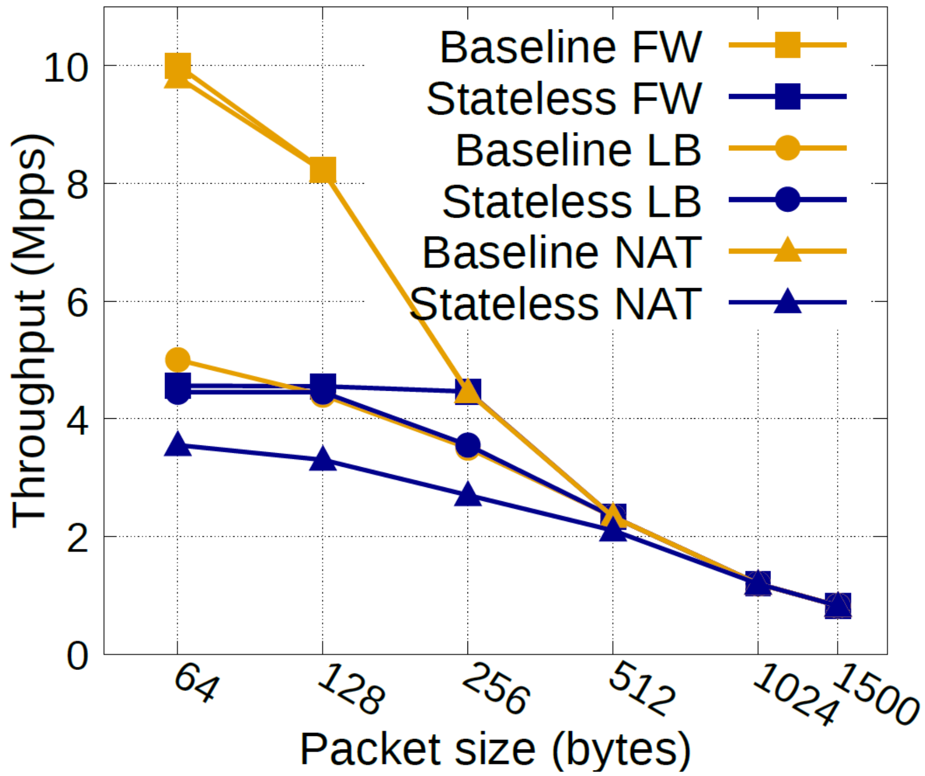
- Raw throughput, latency
- Handling failure
- Handling scaling in-out

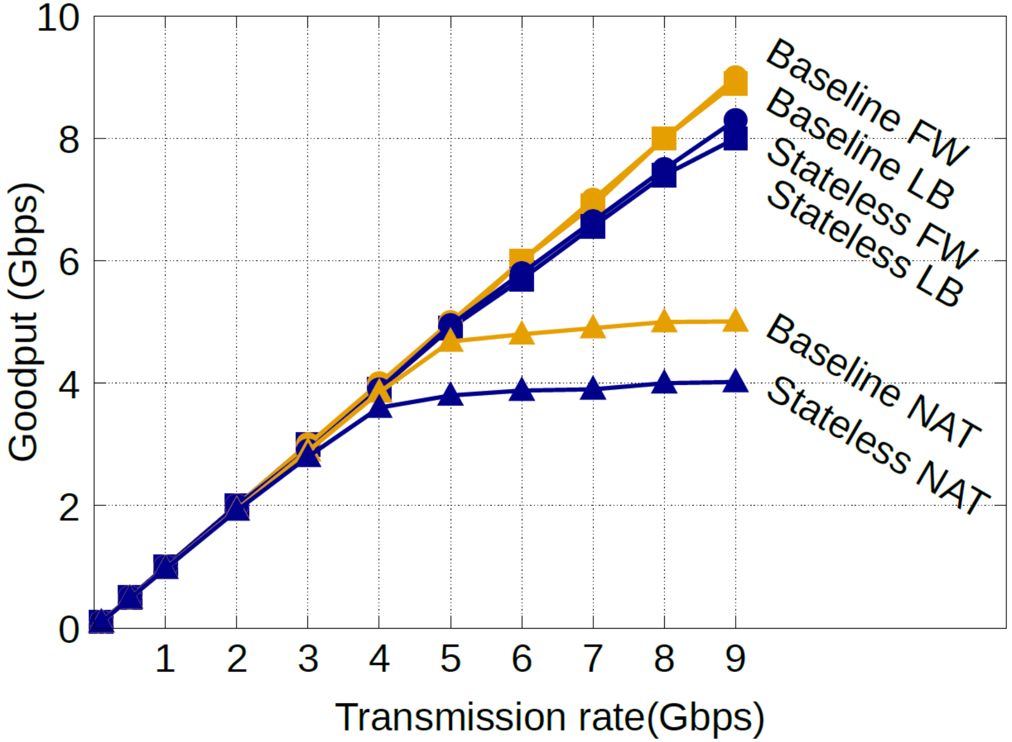


Network Functions:

- Baseline Network Functions (state and processing are coupled)
- Stateless Network Functions (state and processing are decoupled)

Throughput



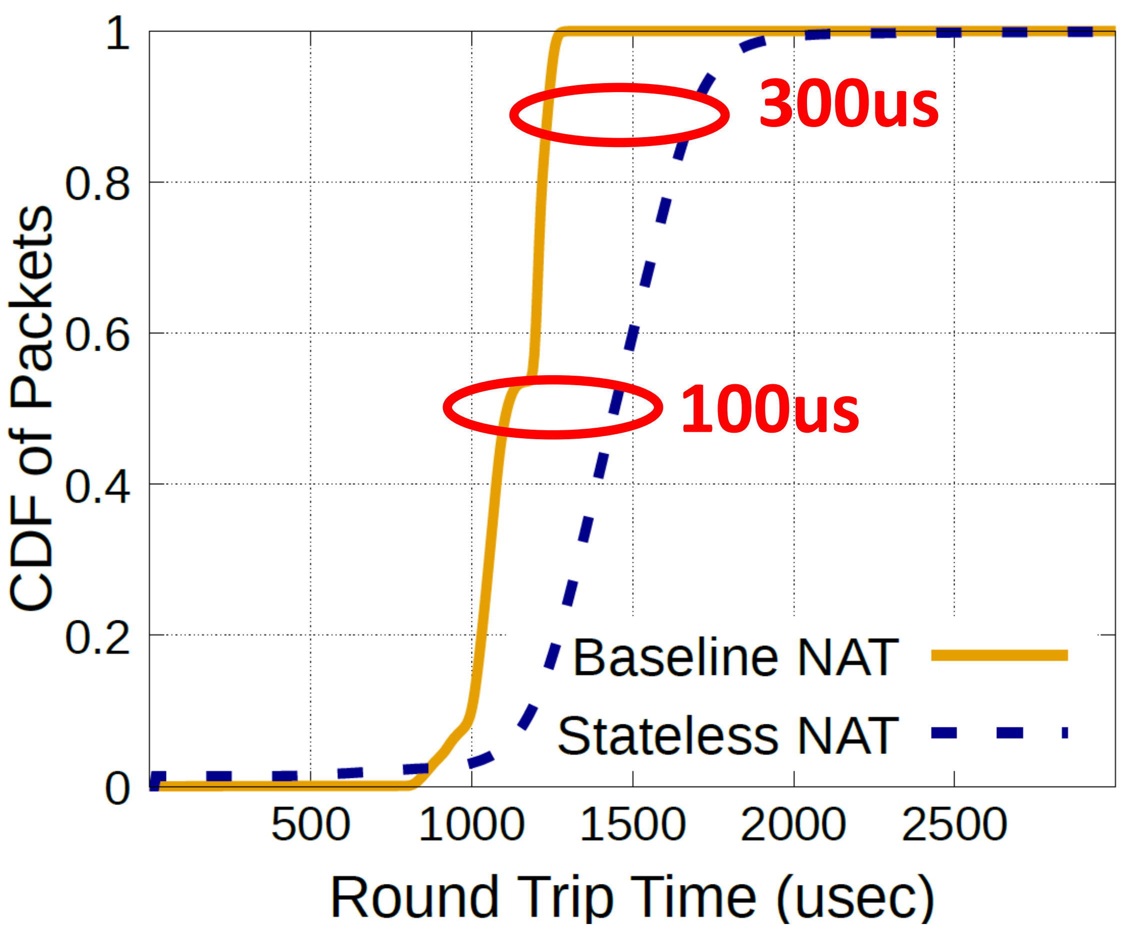


Raw packets per second – lower until about 256 byte packets

Enterprise Trace – Stateless Roughly matches Baseline

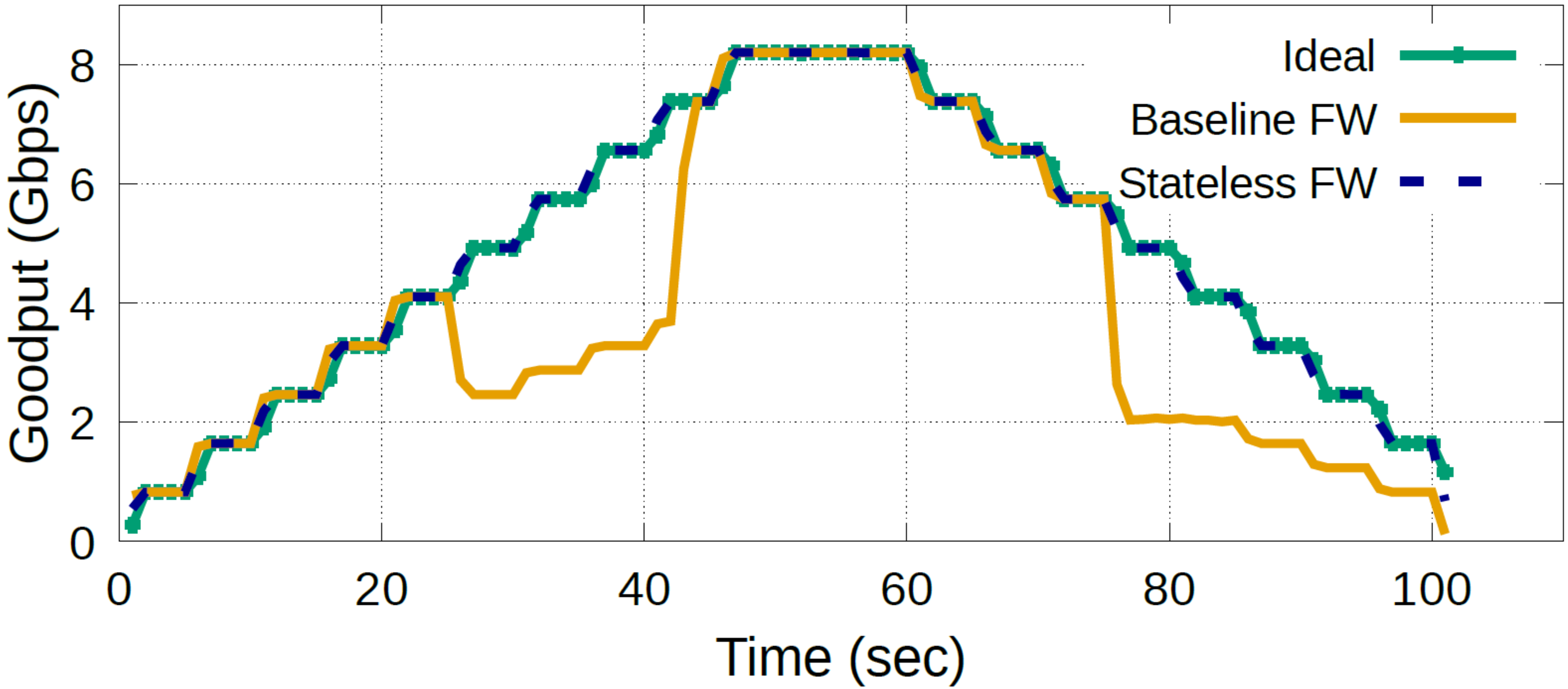
Note: similar to systems which have added support for scaling or failure

Latency

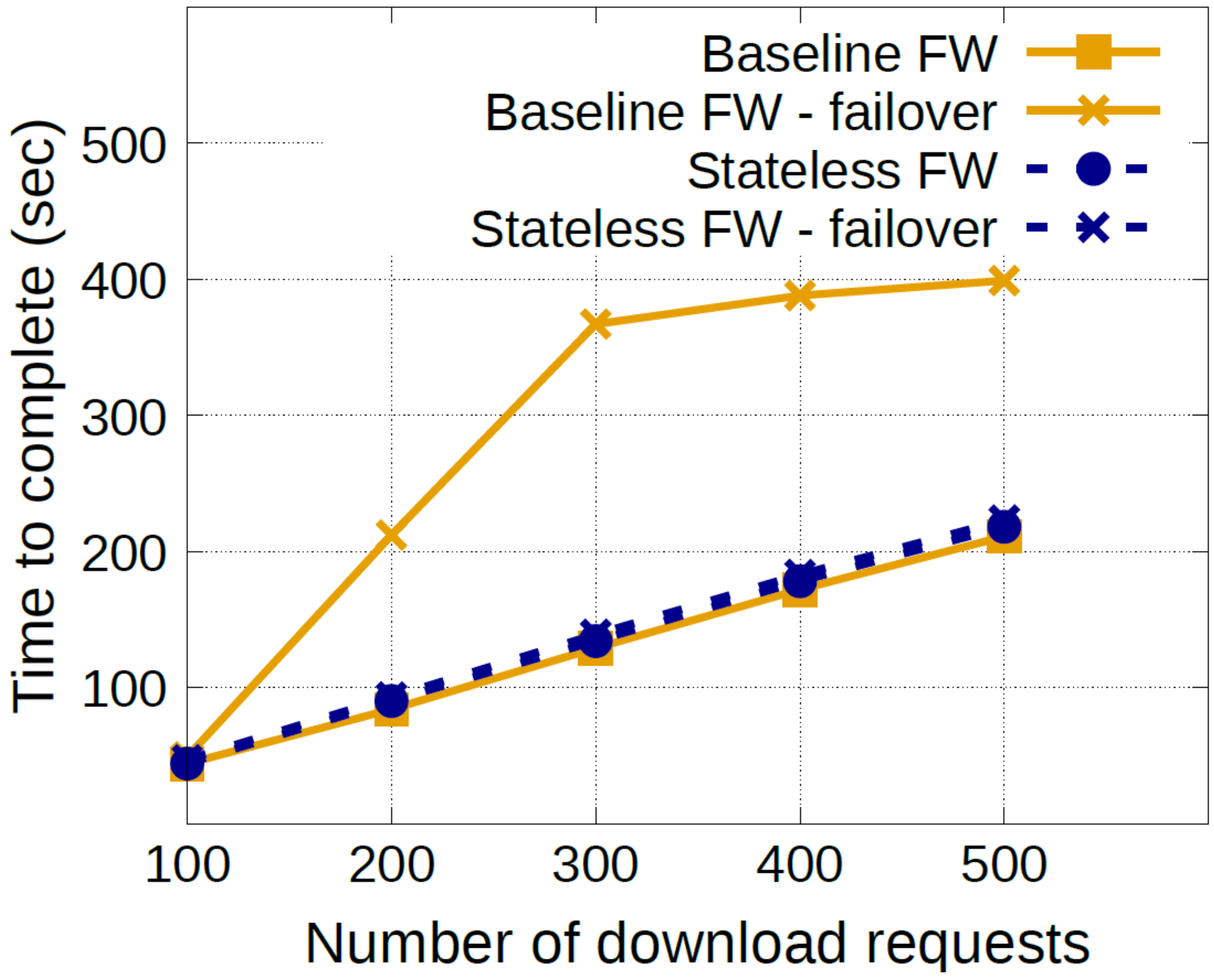


NAT (Firewall and Load balancer has slight less latencies)

Scaling In and Out



Handling Failure



Commercialization Effort

About

Murad Kablan

Eric Keller

+ 5 Engineers, 1 BizDev/Marketing, 1 intern

nsdi '17

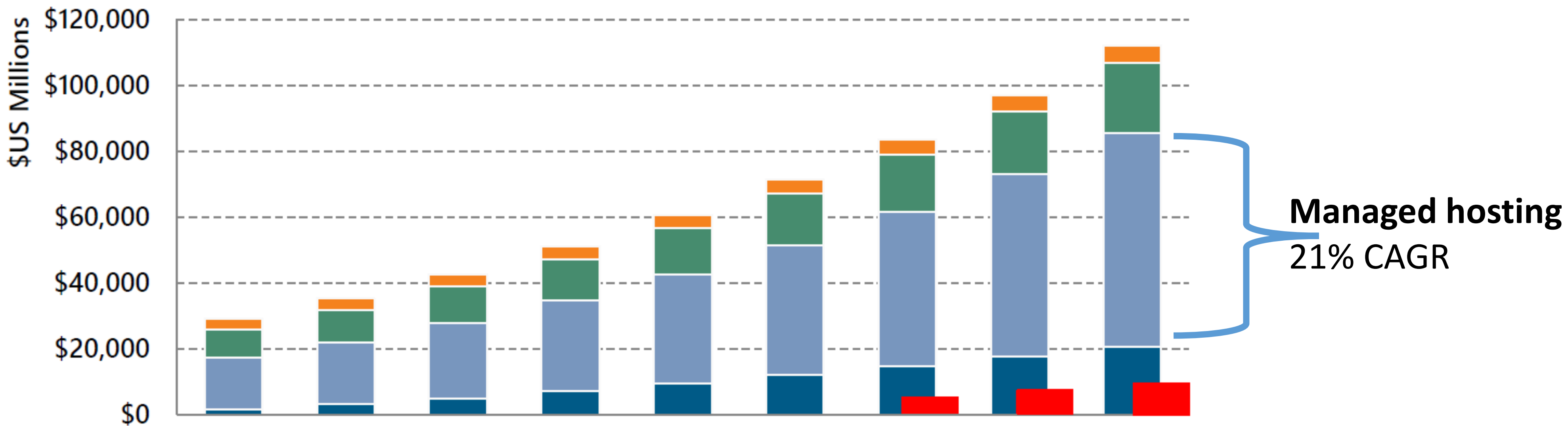
techstars

NSF CORPS
NSF Innovation Corps

NSF SBIR/STTR
SMALL BUSINESS INNOVATION RESEARCH
SMALL BUSINESS TECHNOLOGY TRANSFER

Target Customers

- Initial: Managed Service Providers, Next: Cable / Telco



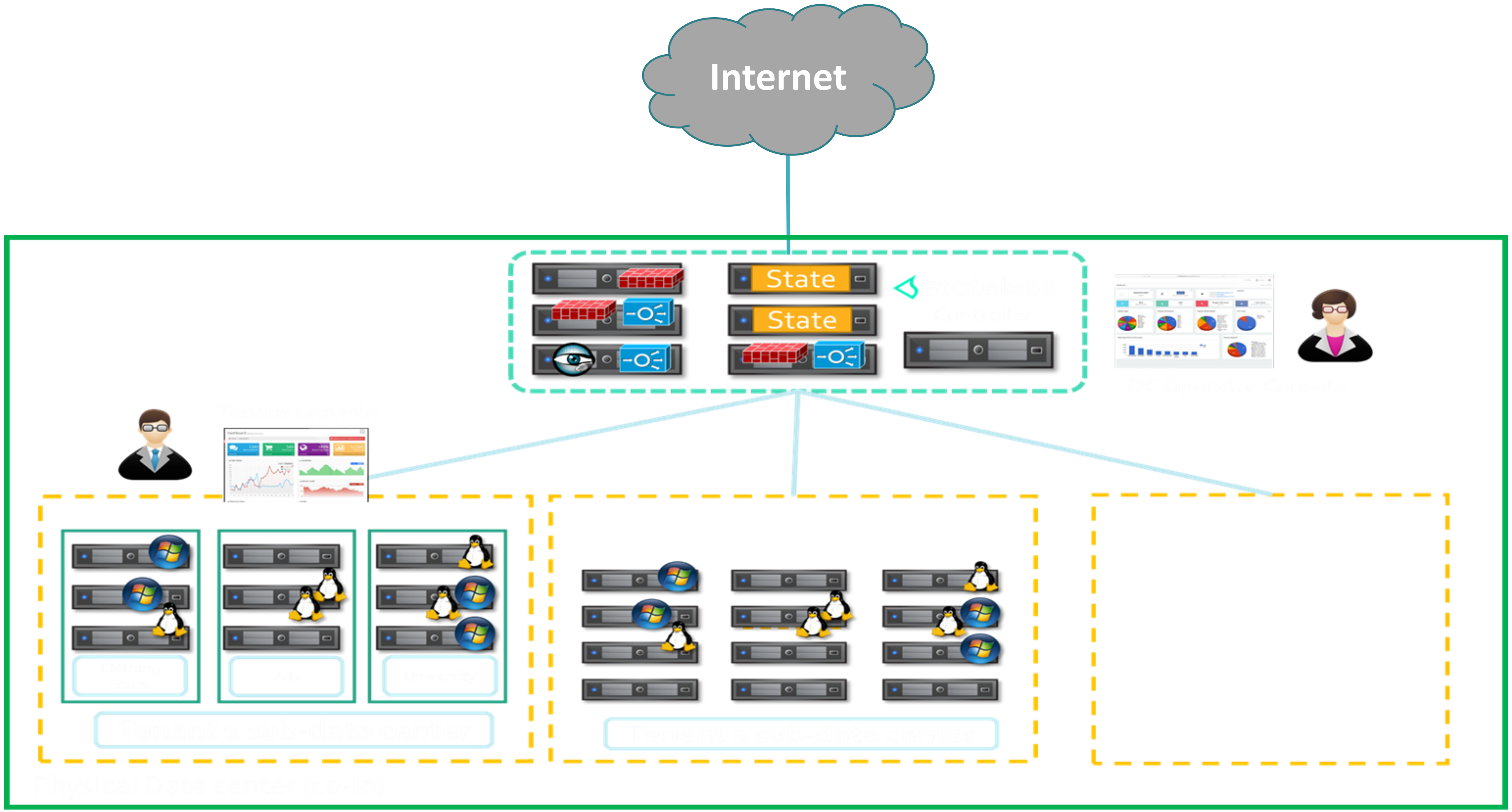
Key Business Drivers:

New revenue streams	Gain more customers	Streamline operations	Reduce risk	Improve customer satisfaction

“Building and running a network service is difficult and expensive”

	Hardware Infrastructures	Virtualized Infrastructures	Network as a Service
DESIGN	Slicing is hard	Lots of dev. effort	Ready to use
REQUEST	Support tickets handled by network operator	Support tickets handled by network operator	Automated via platform
PURCHASING	Specialized hardware with long delivery & deployment times	Commodity hardware with restrictive license pools	Plug & play commodity Hardware, pay per use
CONFIG	Extensive and time-consuming	Quick, but complex to scale	Automated and scalable via platform
FUNCTIONS	Complex to update and scale	Easy updates and scaling, but with disruption	Seamless updates without disruption
UPGRADE	Once every three to five years	Once a year for new license pools	Anytime, on-demand

Deployment



Prove Technology outside of Lab

PoC

Mechanism:
- Deploy in sandbox
- Setup for fake tenants
- Simulate traffic / events (failure)

Goal:
- Demonstrate ease of use
- Product functionality feedback

Exit Criteria:
- Pass initial tests of stability, performance, and resilience
- Positive customer experience

Pilot

Mechanism:
- Step1: Tapped real traffic.
- Step2: low-profile tenants.
- Simulate events (failure)

Goal:
- Quantify perf. and resilience
- Quantify value (cost savings)

Exit Criteria:
- Metrics meet expectations

Full Deployment

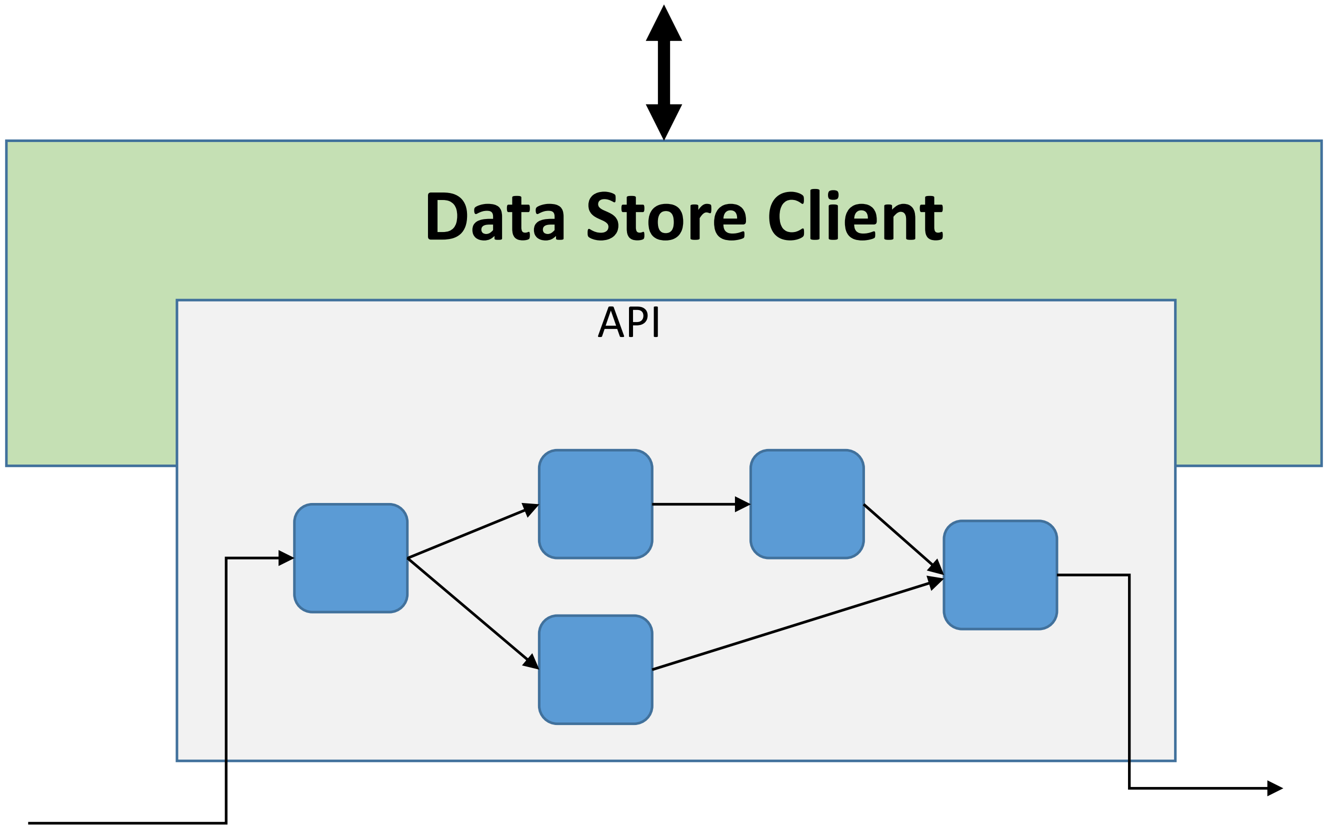
Mechanism:
- Offer out as service to tenants.

Support:
- Support to initial customers 24/7.
- Frequent product updates

1 started, 1 committed, 2 in discussion, willing to bring on 2 more over next 12-18 months

From the Academic Paper to Product

Network Function Design



Reduce interaction

Hide optimizations

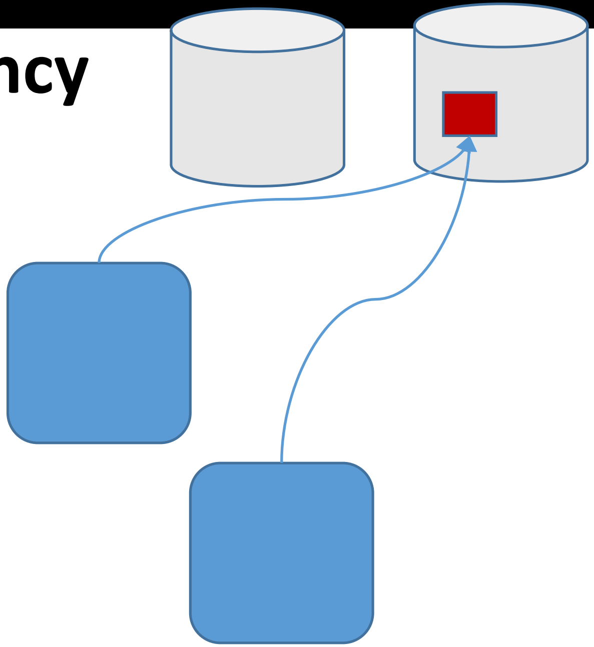
Easy to write NFs
(code is agnostic to opt.)

Processing as graph of fine or coarse functions

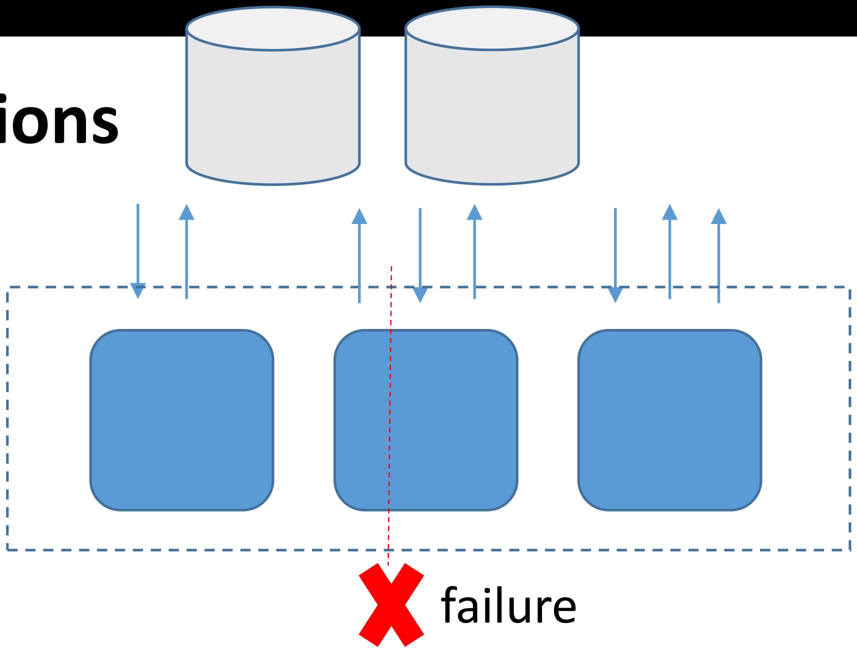
Near term: clean API, leverage ubiquity of DPDK

Standard Distributed System Issues

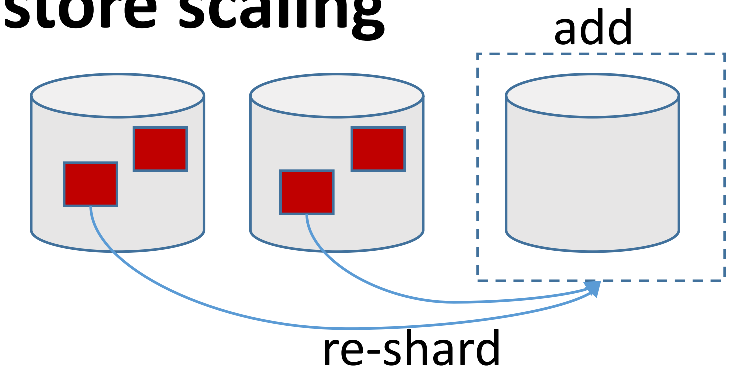
Consistency



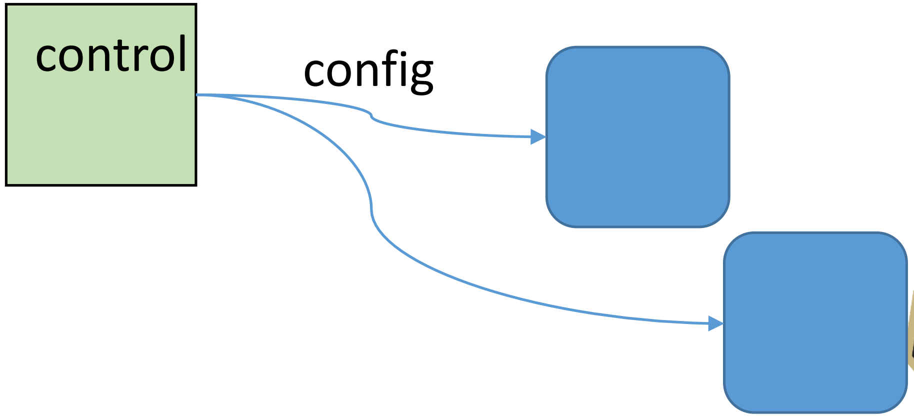
Transactions



Data store scaling



Configuration



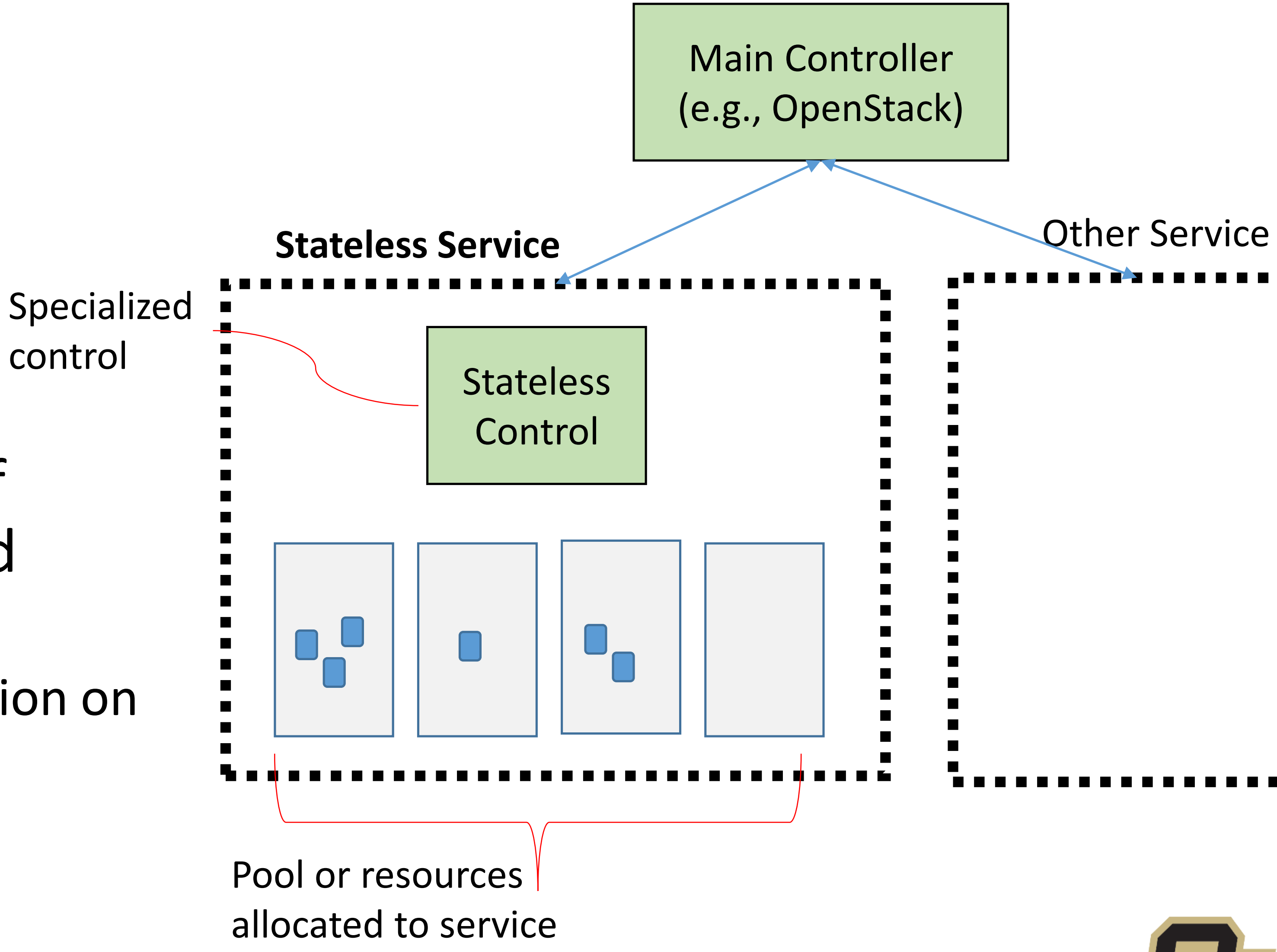
Platform

ONAP, OpenStack, …

- Doesn’t (shouldn’t) matter to us

Public Cloud?

- Current impl. hindered by lack of control in virtualization layer, and network layer
    - (e.g., lack of DPDK support, limitation on tunneling, unpredictable network)



Conclusions and Future Work

- Networks need agile network functions
  - Seamless scalability, failure resiliency, without sacrificing performance
- StatelessNF is a design from the ground up
  - Zero loss scaling, zero loss fail-over
- Main potential drawback… performance, but in this extreme point:
  - Throughput similar to other solutions
  - 100-300us added latency (similar to other solutions)
- Future work: Evolve data store design for network functions

Thanks!

eric@bestateless.com