Zero-waste Designs for Terabit Network Systems

Liangcheng (LC) Yu

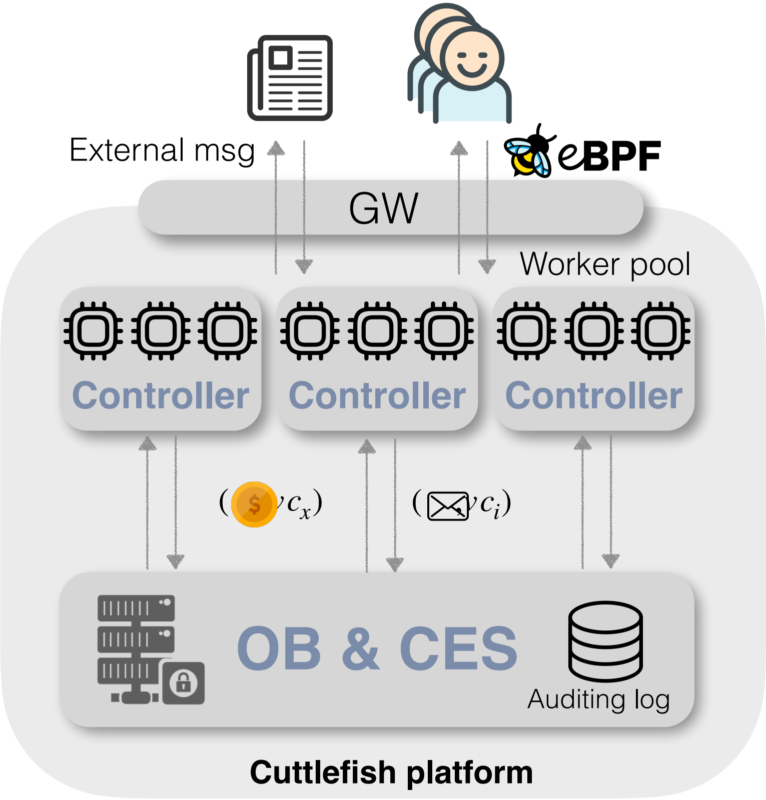
Microsoft Research

Cuttlefish: a fair, predictable cloud-hosted exchange platform

Liangcheng (LC) Yu, Prateesh Goyal, Ilias Marinos, and Vincent Liu

Advances in Financial Technologies (AFT) 2025

Microsoft Research NVIDIA Penn University of Pennsylvania



Abstraction
- Equal cloud networks
- Equal execution hardware
- …

- Abstracting out variances in cloud infrastructure
- An efficient implementation runnable on commercial cloud

Zero-waste Designs for Terabit Network Systems

Liangcheng (LC) Yu

Microsoft Research

Ever-increasing user applications

Online Conferencing

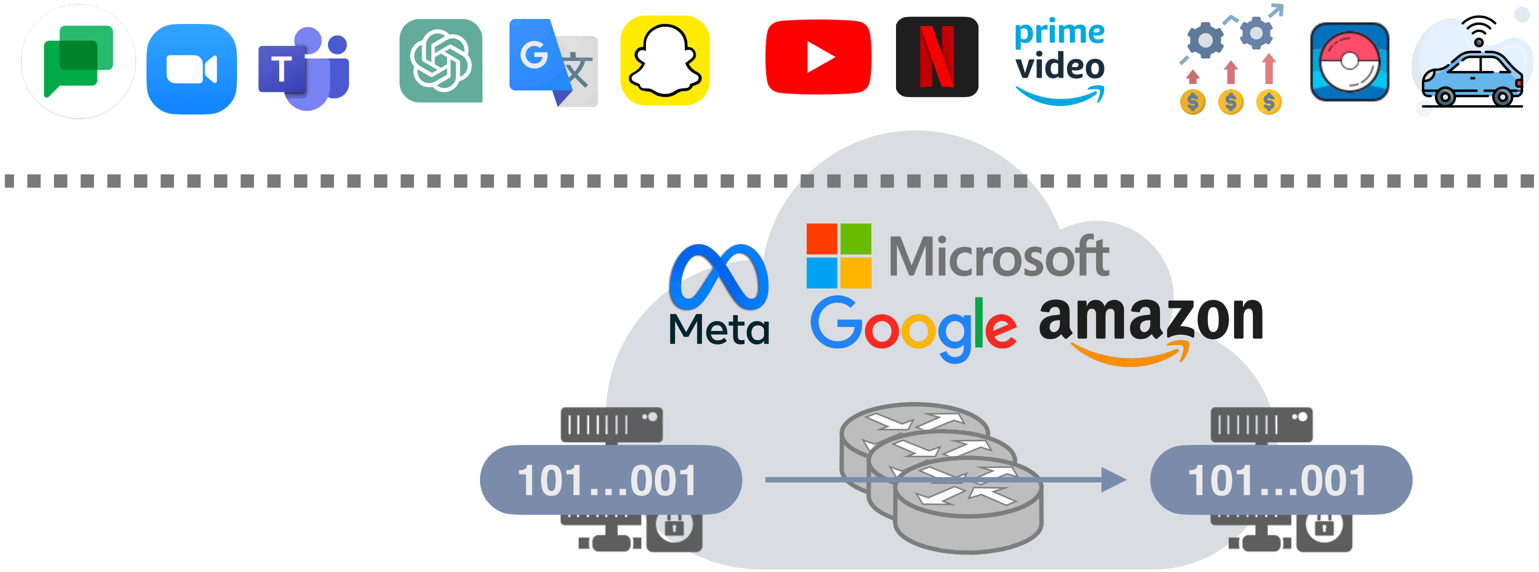
Machine Learning

Video Streaming

Latency-critical Applications

Application

Network systems, a packet forwarding engine



Application

Network system

Networks serve to **forward user data**

Network systems, a packet forwarding engine

Networks serve to **forward user data**

*Today, networks are **far more complex**!*

Network systems: an operator's view

Application

Network system

Time drift

Node / link failure

DoS attack

Congestion collapse

TCP incast

Network hotspot

Bandwidth starvation

Networks serve to **forward user data**

*Today, networks are **far more complex**!*

*…must handle **out-of-control events!***

Network systems: an operator's view

Application

Clock synchronization

Failure mitigation

Defense policy

Load balancing

Desynchronization

Fairness control

Congestion control

Network system

Networks serve to **forward user data**

*Today, networks are **far more complex**!*

…a vast array of control tasks

Network systems: an operator's view

Application

Network system

Networks serve to **forward user data**

*Today, networks are **far more complex**!*

…a vast array of control tasks

…in-network computation w/ emerging HW accelerators

…and more!

Network systems: an operator's view

Application

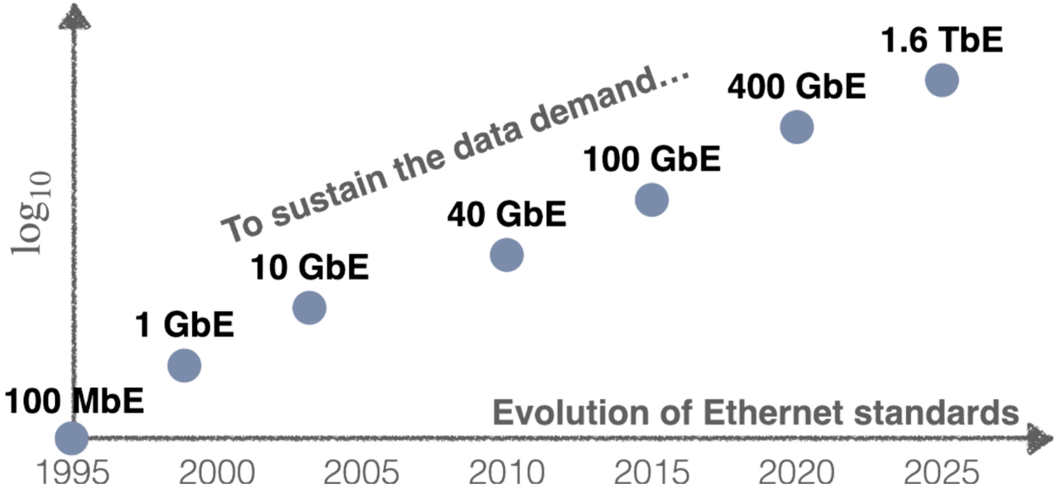
Today, network systems are
more than just about **data forwarding**!

Networks serve to **forward user data**

*Today, networks are **far more complex**!*

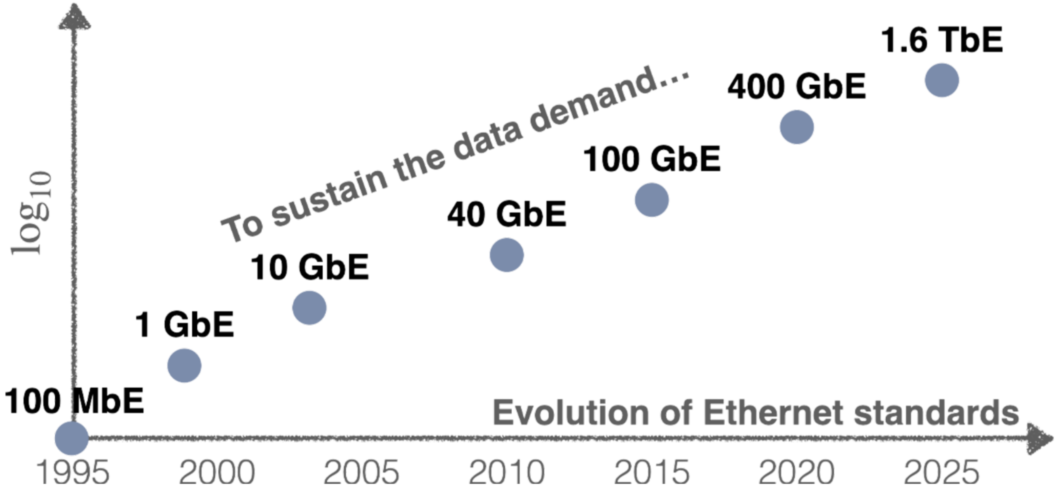
…a vast array of control tasks
…in-network computation w/ emerging HW accelerators
…and more!

Trend toward terabit speed…



The speed of networking is ***outpacing*** many others

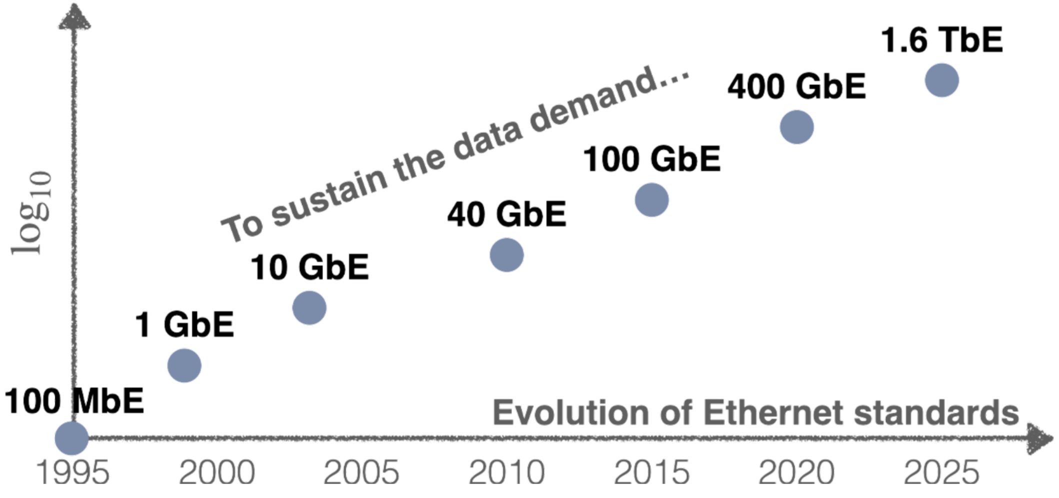
Trend toward terabit speed…



The speed of networking is ***outpacing*** many others

Great for ***application data transfer*** ☺

Trend toward terabit speed...

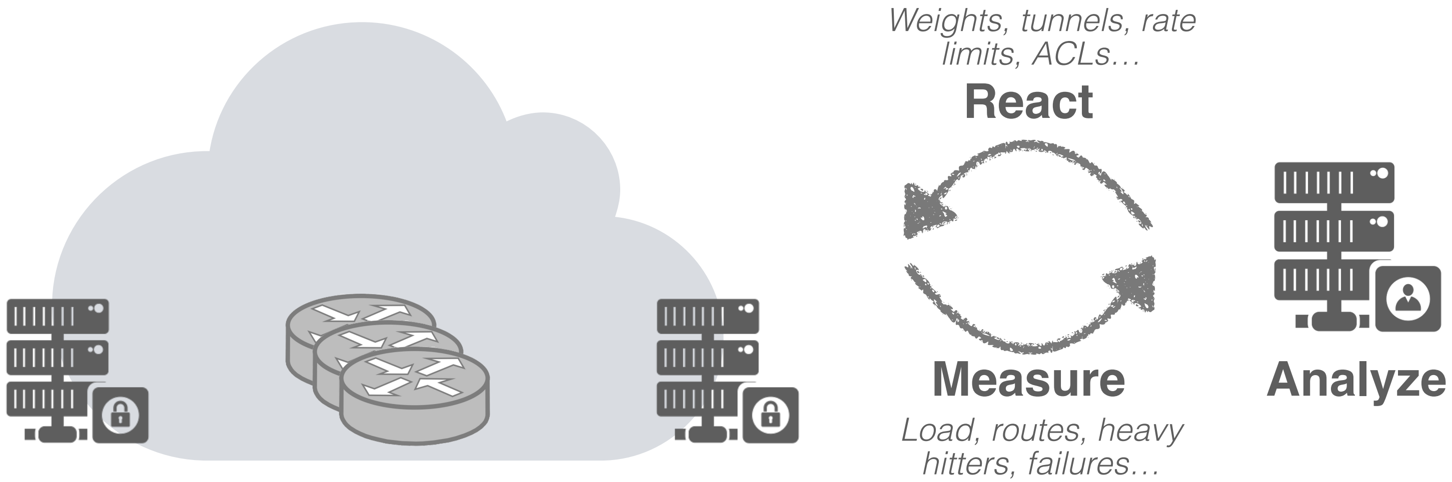


The speed of networking is ***outpacing*** many others

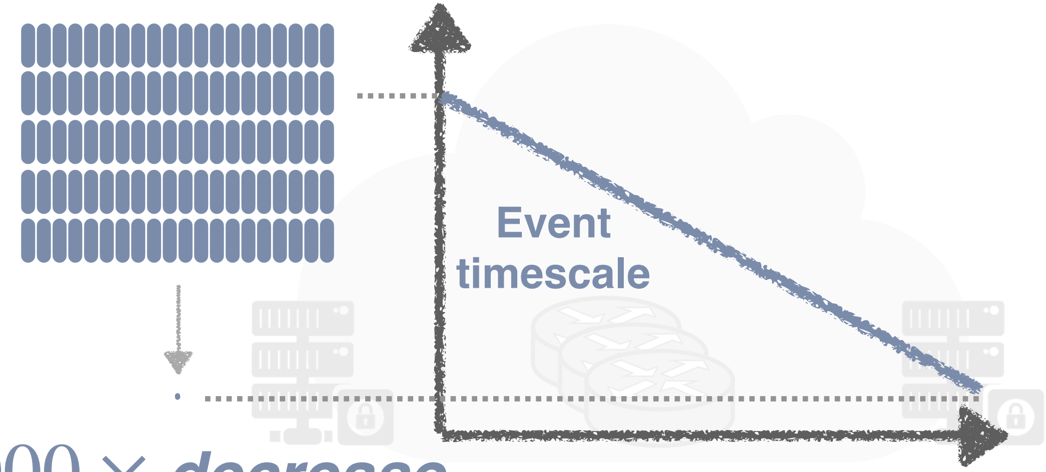
Great for ***application data transfer*** ☺

... problematic for <u>auxiliary</u> tasks! ☹

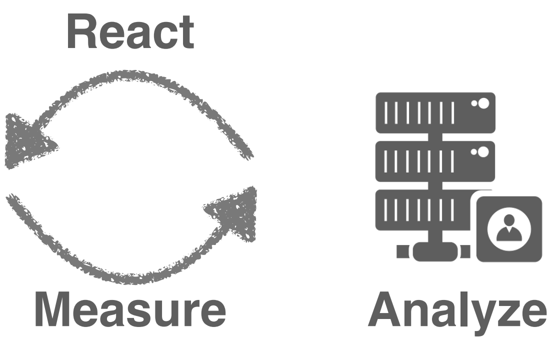
Network control function as an example



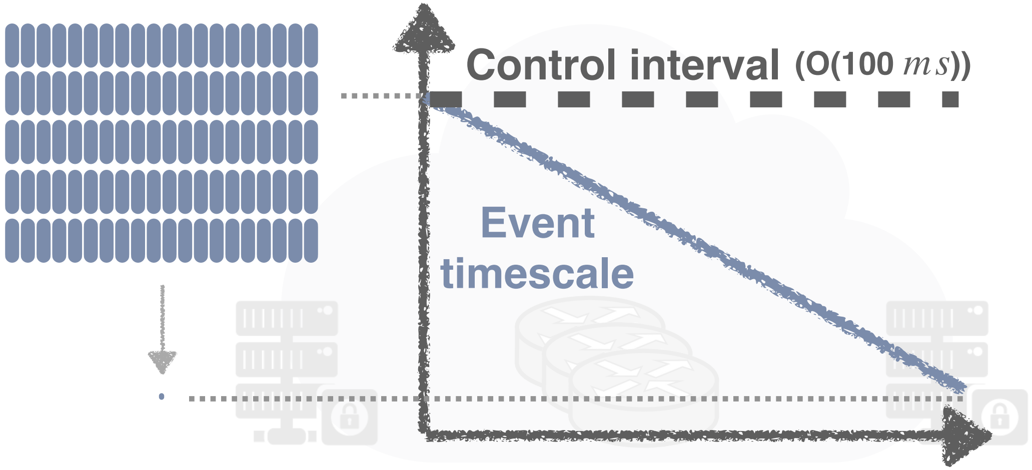
Network control function as an example

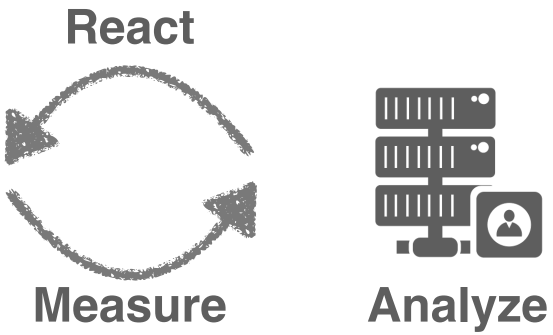


> 1000 × ***decrease***
> in duration!



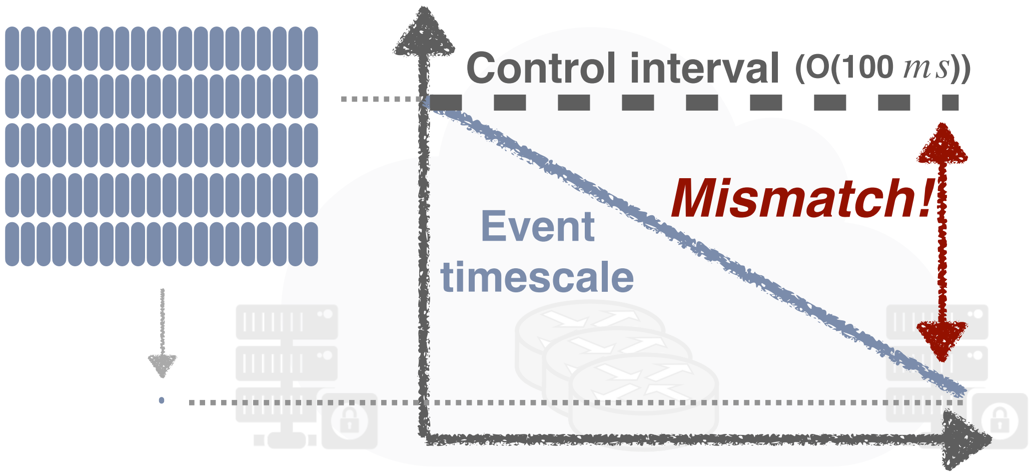
Control, fast and slow

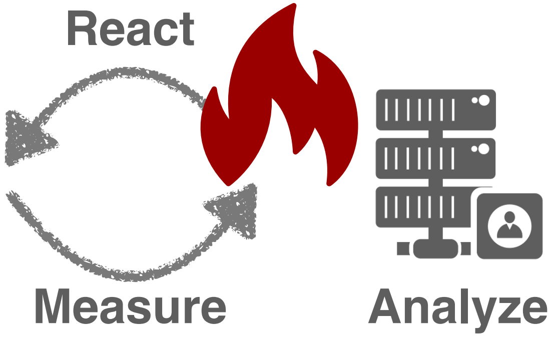




If the control interval remains coarse-grained…

Control, fast and slow

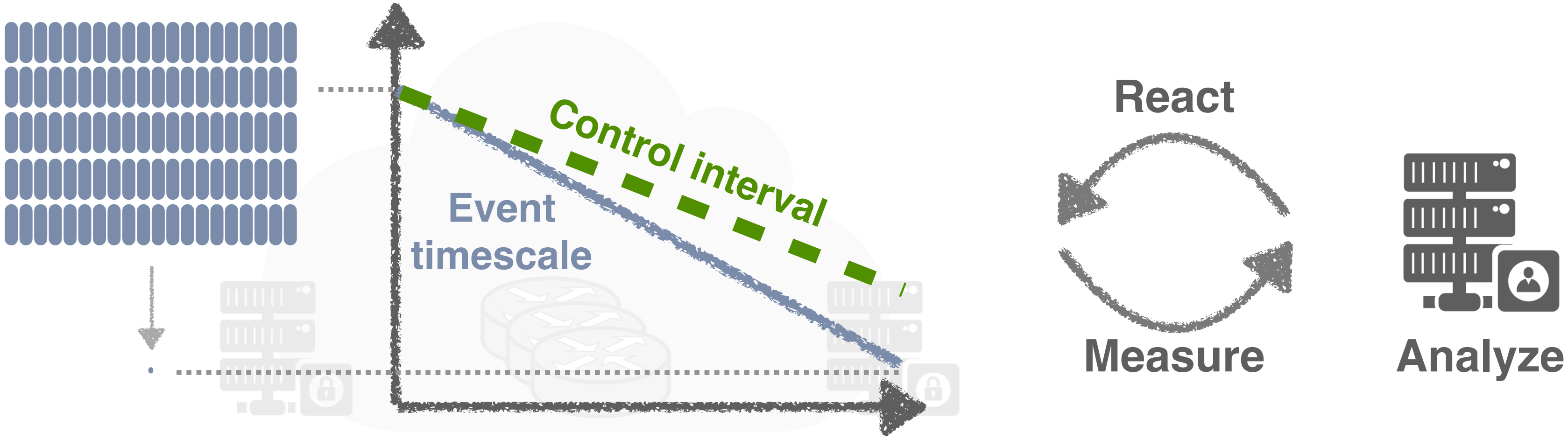




If the control interval remains coarse-grained…

☹ **Hard to react to microscopic events**

Control, fast and slow

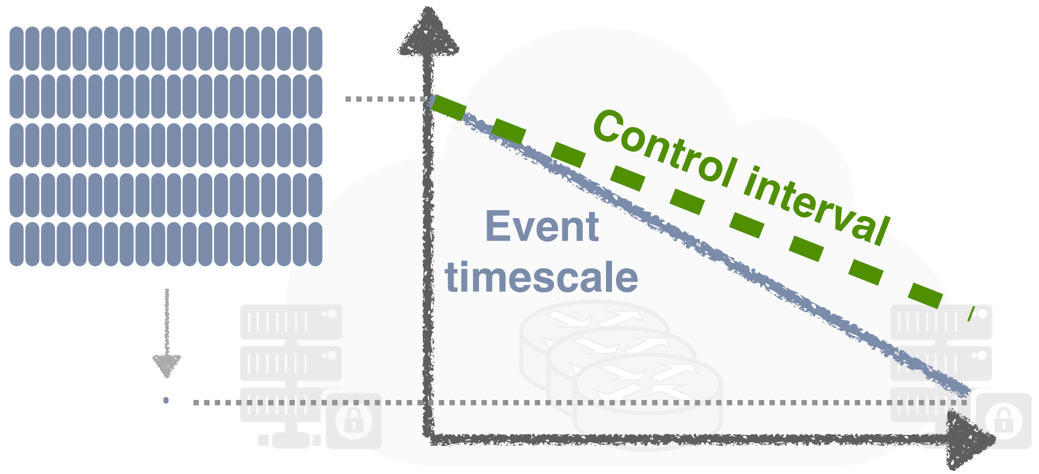


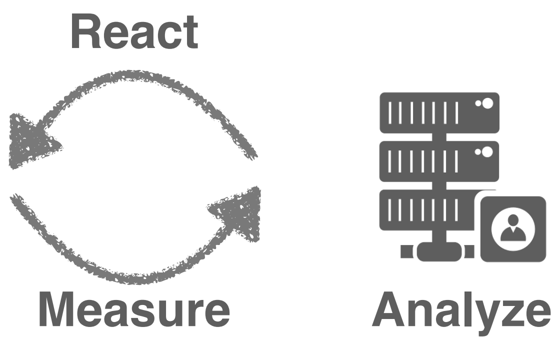
If the control interval remains coarse-grained…

☹ **Hard to react to microscopic events**

If were to catch up with the link speeds…

Control, fast and slow





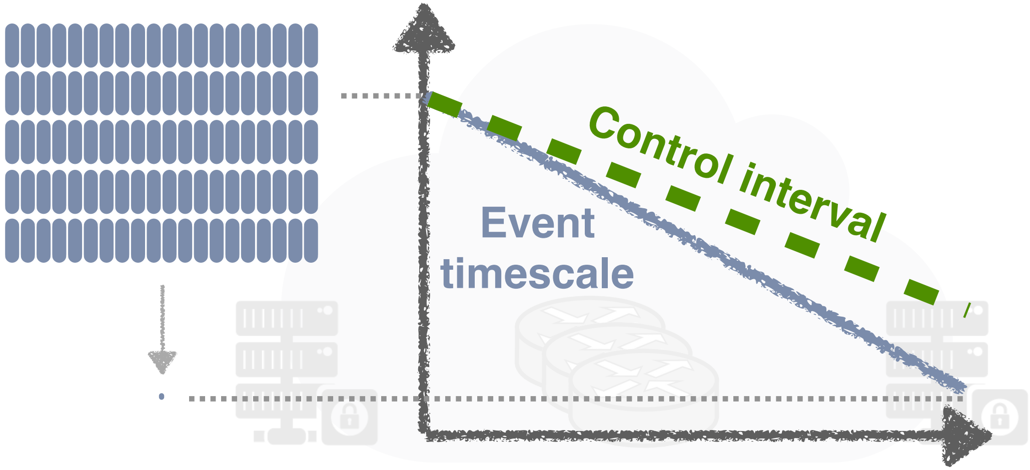
If the control interval remains coarse-grained…

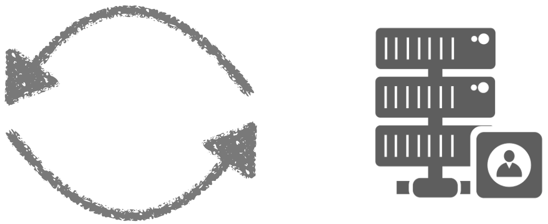
☹ **Hard to react to microscopic events**

If were to catch up with the link speeds…

Allocate more cables, CPUs…?

Control, fast and slow





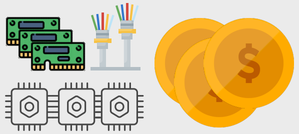
If the control interval remains coarse-grained…

☹ **Hard to react to microscopic events**

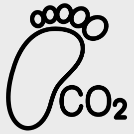
If were to catch up with the link speeds…

Allocate more cables, CPUs…?

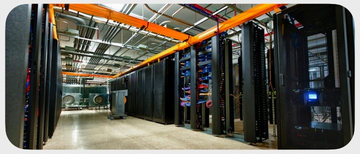
☹ **Costs!**



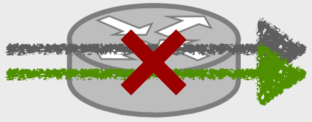
Device purchasing



Embodied & operational carbon



Power & cooling



Impact to existing traffic

Control, fast and slow

Hard to react to microscopic events

If were to catch up with the link speeds…

Allocate more resources (cables, CPUs…)?

Costs!

Device purchasing

Embodied & operational carbon

Power & cooling

Impact to existing traffic

Control, fast and slow

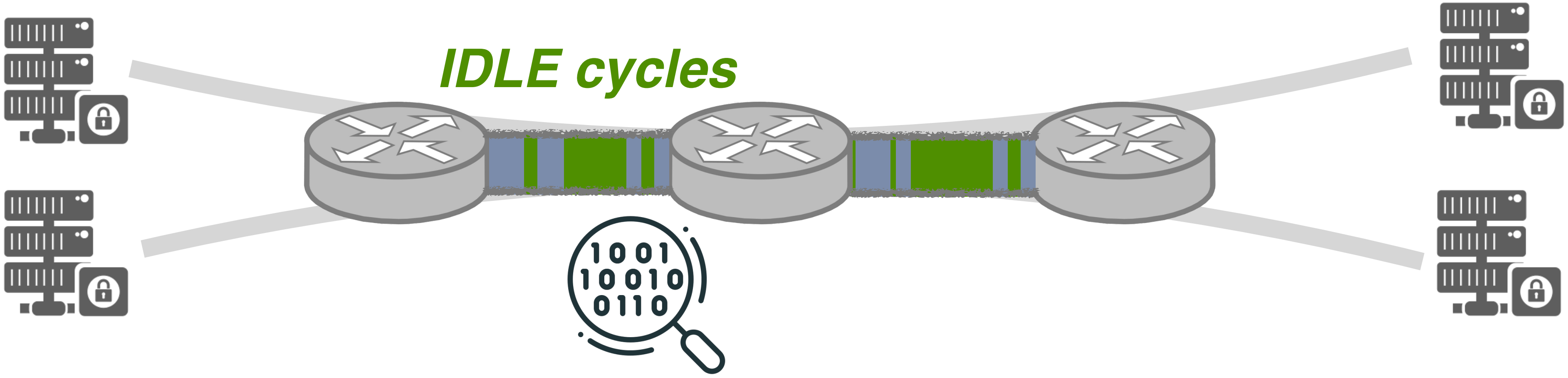
Can we break this tension?

If were to catch up with the link speeds…

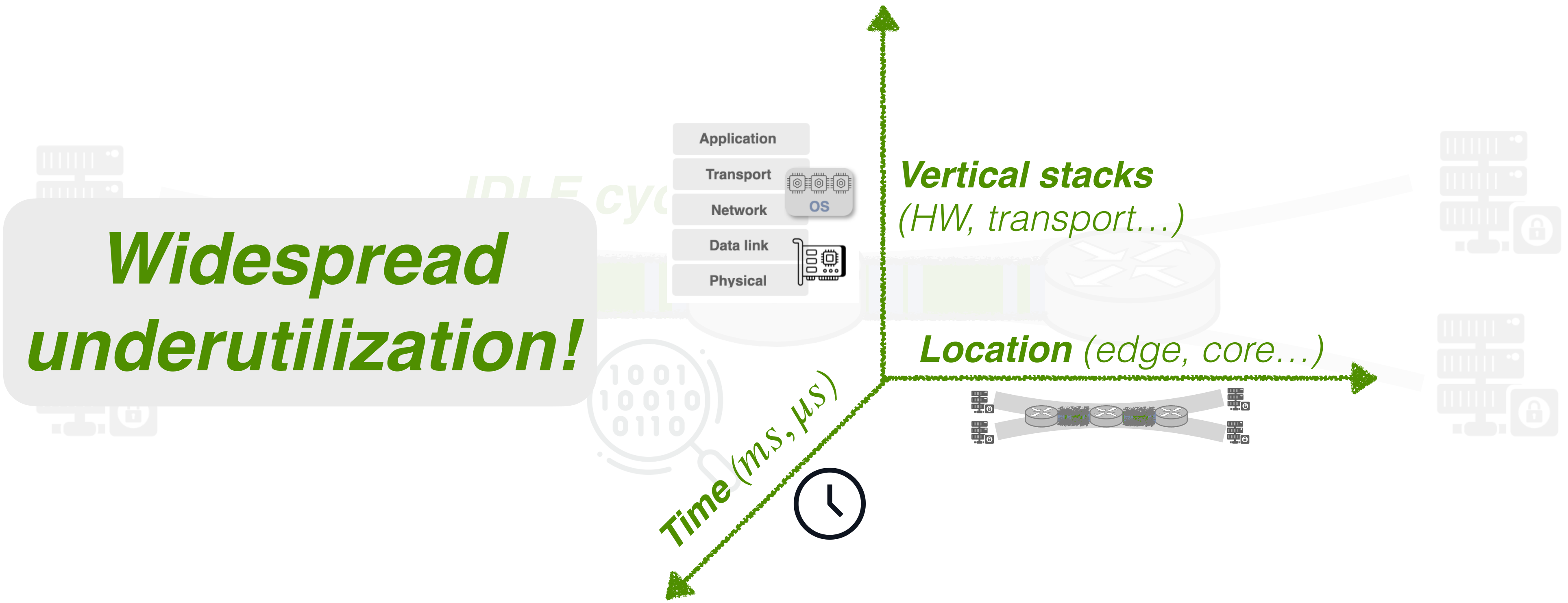
Allocate more resources (cables, CPUs…)?

Costs!

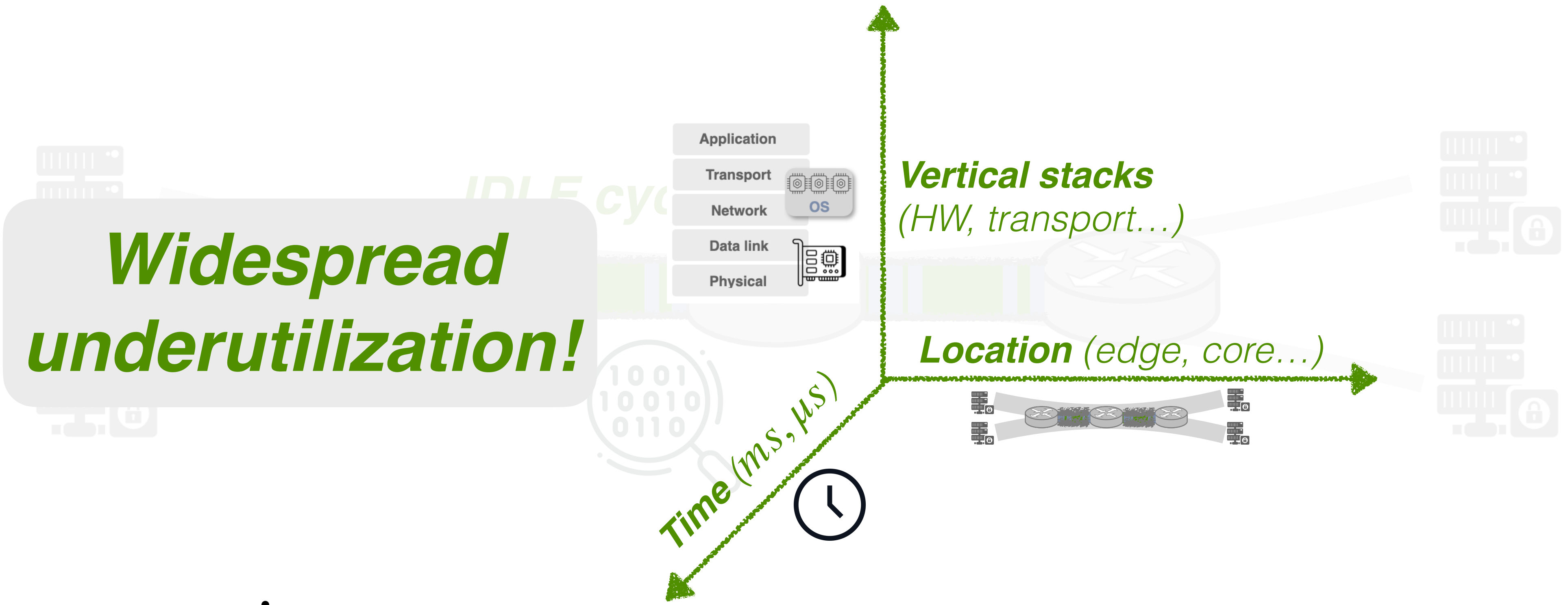
Observation: in-network waste



Observation: in-network waste



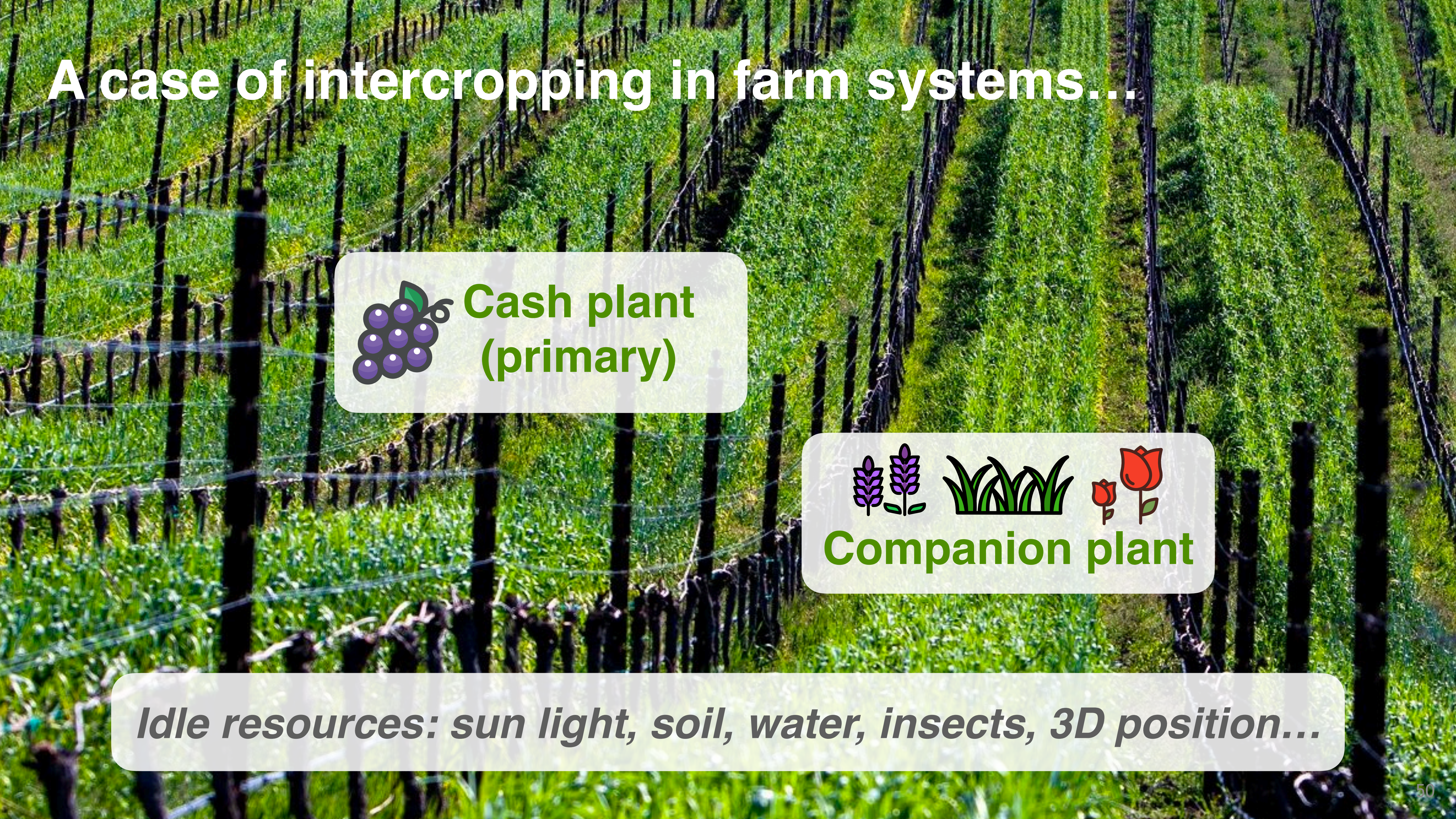
Observation: in-network waste



Why not harness them to support auxiliary functions?

A case of intercropping in farm systems…

A case of intercropping in farm systems…



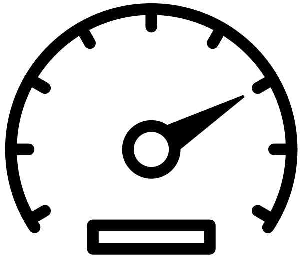
Cash plant (primary)

Companion plant

Idle resources: sun light, soil, water, insects, 3D position…

A zero-waste design approach

High-efficiency designs



Input: user workload

Goal: output a network to optimize performances with minimal resource usage

Zero-waste designs

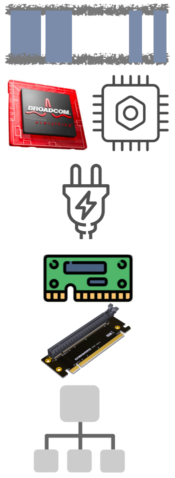
Input: the workload ***and the network***

Goal: maximize the utility of ***that network***

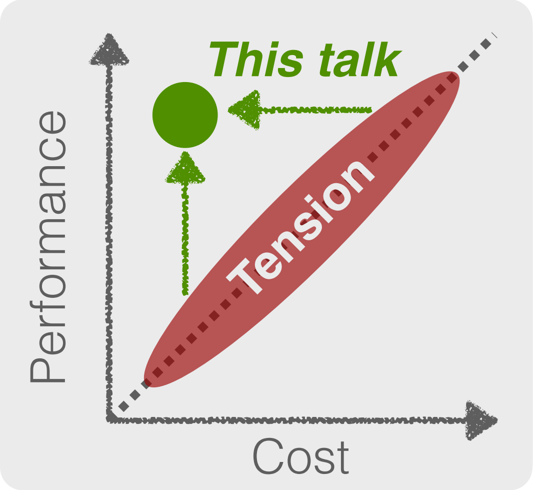
This talk: takeaway

1. In-network waste is ***widespread***, and in ***numerous forms***

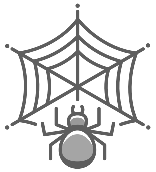
- Ethernet link IDLE cycles
- Switch CPUs
- Wasted power
- Memory
- Spare PCIe payload
- Middleboxes
- …



2. By exploiting domain-specific underutilization, it is ***possible*** to integrate performant functions with ***near-zero costs***

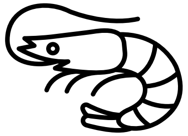


Instantiations of zero-waste designs



OrbWeaver *(NSDI 2022)*
Reusing IDLE link cycles for in-band control communication

Reuse



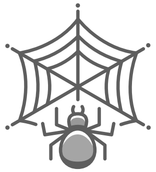
Mantis *(SIGCOMM 2020)*
Recycling switch resources for flexible, sub-RTT reactions

Recycle

Beaver *(OSDI 2024)*
Reducing 'tax' of partial snapshots for distributed cloud services

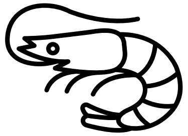
Reduce

Outline



OrbWeaver *(NSDI 2022)*
Reusing IDLE link cycles for in-band control communication

Reuse



Mantis (*SIGCOMM 2020)*
Recycling switch resources for flexible, sub-RTT reactions

Recycle

Beaver *(OSDI 2024)*
Reducing 'tax' of partial snapshots for distributed cloud services

Reduce

Networks are woven from packets

- A primary goal of computer networks: ***delivery packets***

Networks are woven from packets

- A primary goal of computer networks: ***delivery packets***
    - ***User application***: video streaming, web browsing, file transfer…

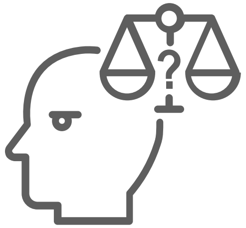
Networks are woven from packets

- A primary goal of computer networks: ***delivery packets***
  - ***User application***: video streaming, web browsing, file transfer…
  - ***Non-user application***: control messages, probes about network state, keep alive heartbeats…

Networks are woven from packets

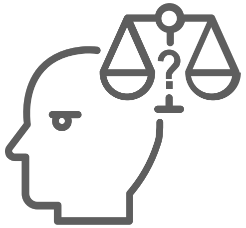
- A primary goal of computer networks: ***delivery packets***
  - ***User application***: video streaming, web browsing, file transfer…
  - ***Non-user application***: control messages, probes about network state, keep alive heartbeats…
- Often, two classes of traffic ***multiplex*** the same network

When introducing an in-band control function…



To cost ***extra bandwidth*** for ***efficacy***, or not?

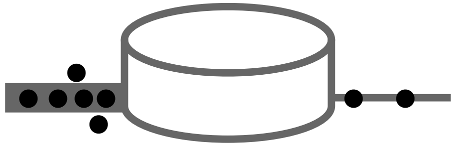
When introducing an in-band control function…



To cost ***extra bandwidth*** for ***efficacy***, or not?

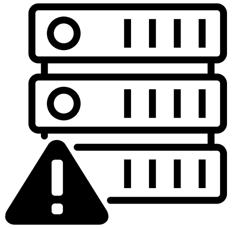
Time synchronization

Clock-sync rate ↔ **clock precision**



Congestion notification

Probe data/rate ↔ **measurement accuracy**



Failure detection

Heartbeat frequency ↔ **detection speed**

In-band telemetry

INT postcard volume ↔ **post-mortem analysis**

When introducing an in-band control function…

To cost ***extra bandwidth*** for ***efficacy***, or not?

Can we coordinate at ***high-fidelity*** with a ***near-zero cost*** (to usable bandwidth, latency…)?

clock precision

measurement accuracy

Failure detection

Heartbeat frequency ↔ **detection speed**

In-band telemetry

INT postcard volume ↔ **post-mortem analysis**

When introducing an in-band control function…

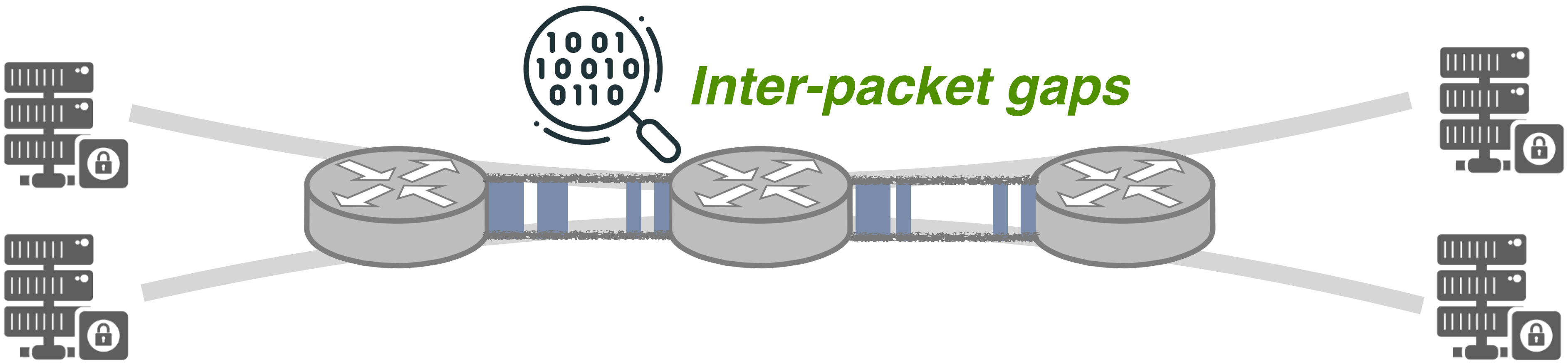
To cost ***extra bandwidth*** for ***efficacy***, or not?

Can we coordinate at ***high-fidelity*** with a ***near-zero cost*** (to usable bandwidth, latency…)?

<u>Idea: Weaved Stream</u>

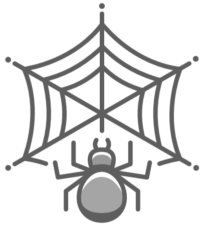
- Exploit ***every gap*** ($O(100$ns$)$) between user packets opportunistically
- Inject customizable ***IDLE packets*** carrying information across devices

Opportunity: $< \mu s$ gaps are prevalent

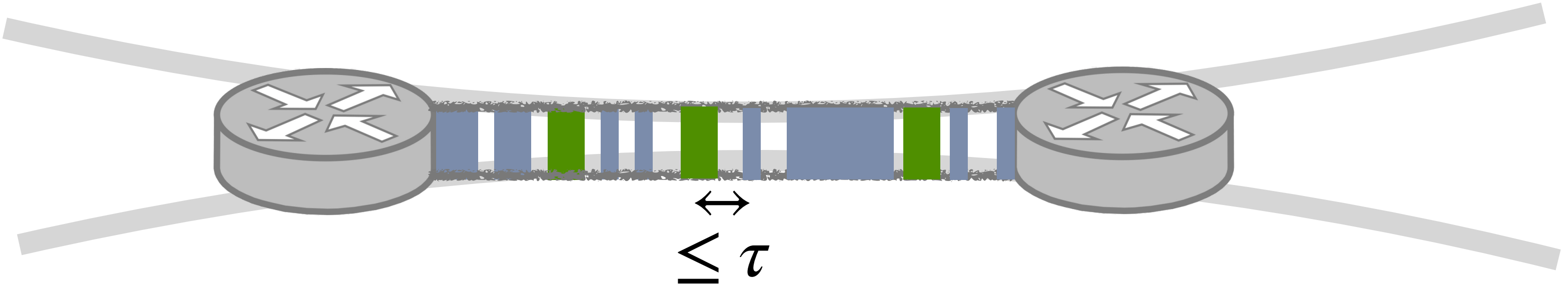


Root causes?

- Uncertainties in application load patterns (e.g., burstiness)
- Conservative resource provisioning for peak usages
- Bottlenecks at CPU processing vs network BW
- TCP effects
- Structural asymmetry
- …

Abstraction: weaved stream 

Union of ***user*** *AND* ***IDLE*** *(injected) packets*



[R1 Predictability] Interval between ***any*** two consecutive packets $\leq \tau$

$$\tau = B_{100Gbps}/MTU_{1500B} = 120ns$$

[R2 Little-to-zero overhead] Near-zero impact to user packets or power draw

Abstraction: weaved stream

Union of ***user*** *and* ***IDLE*** *(injected) packets*

Implement many ***in-network functions***

(failure detection, clock sync, congestion notification…)

for free!

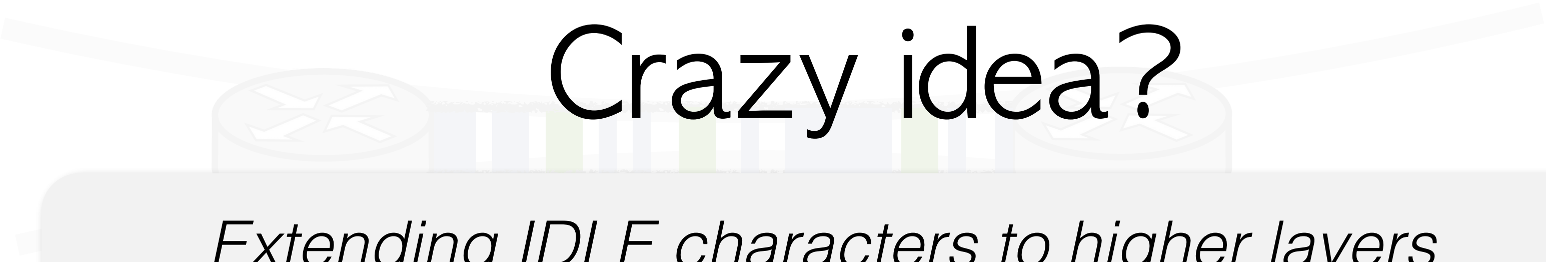
[R1

$$\tau = B_{100Gbps}/MTU_{1500B} = 120ns$$

[R2 Little-to-zero overhead] Not impact user packets or power draw

Abstraction: weaved stream

*Union of **user** and **IDLE** (injected) packets*



Crazy idea?

Extending IDLE characters to higher layers

- Data plane packet generator
- Replication engine
- Data plane programmability
- Flexible switch configuration (priorities, buffers…)

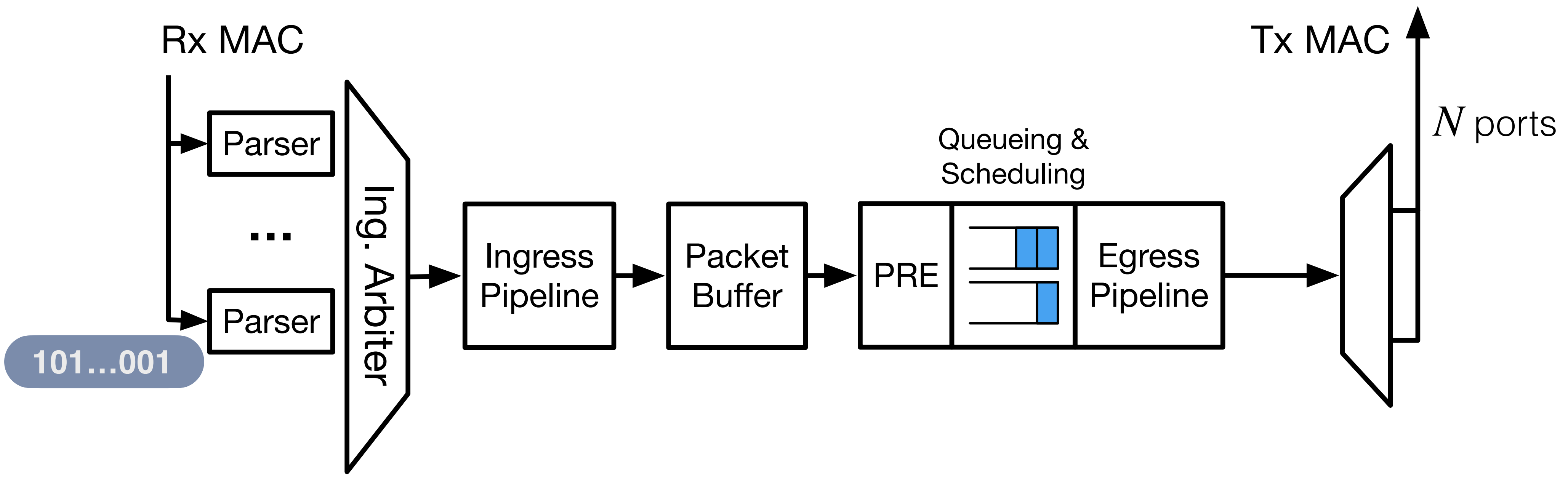
[R1 Pre

[R2 Little-to-zero overhead] Not impact user packets or power draw

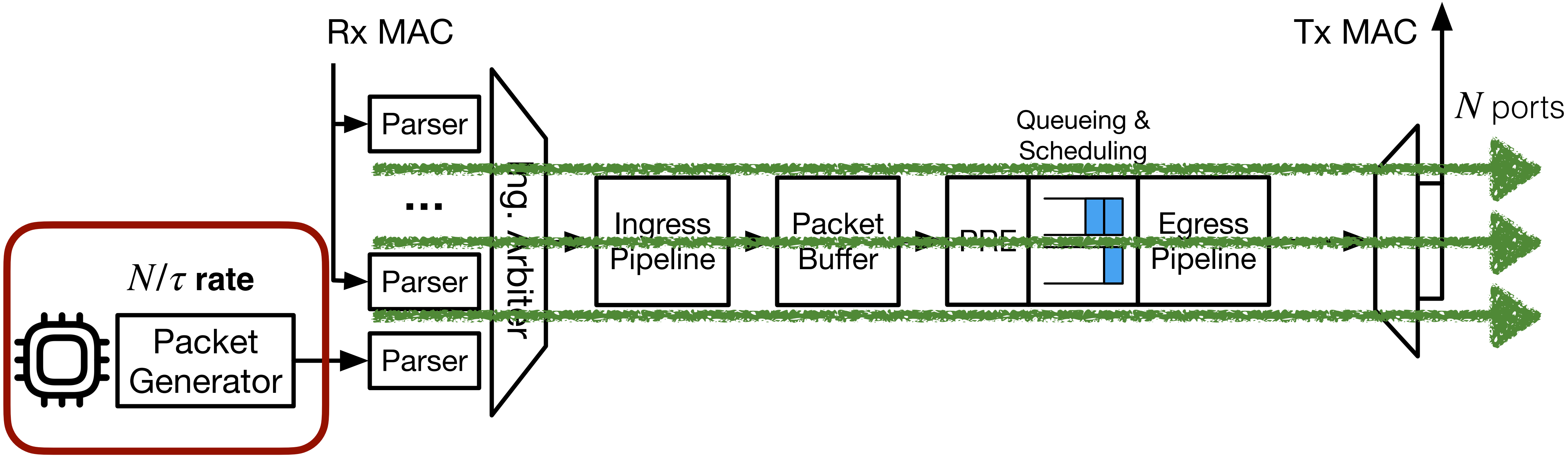
OrbWeaver: outline

1. RMT switch data plane architecture
2. Implementing weaved stream abstraction
3. OrbWeaver applications

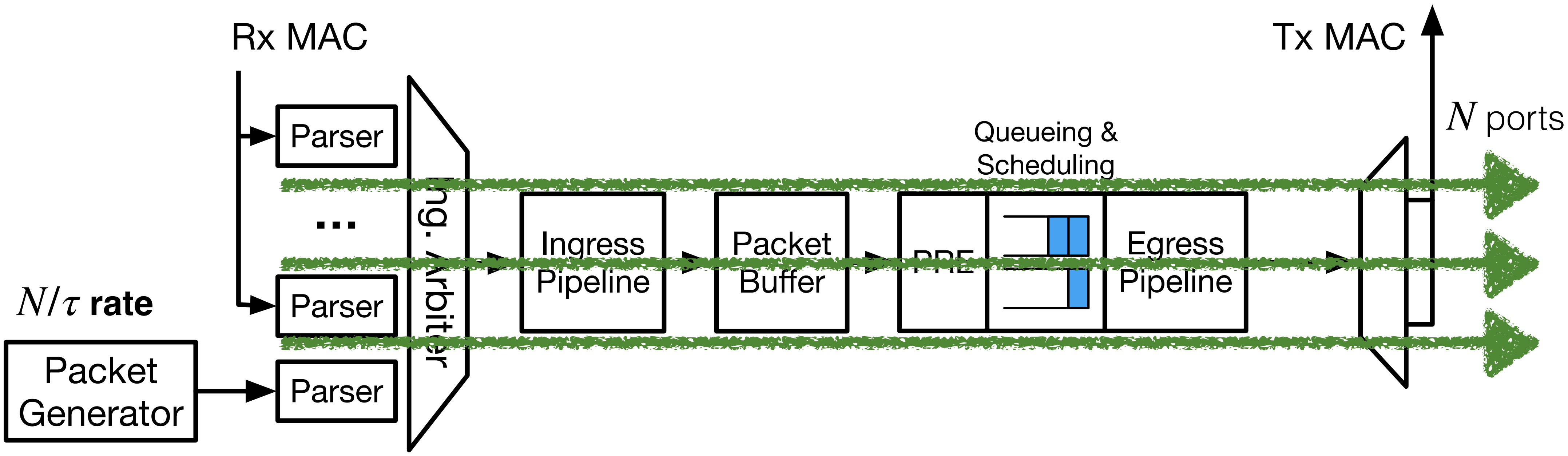
RMT switch architecture



Strawman: blind packet generation

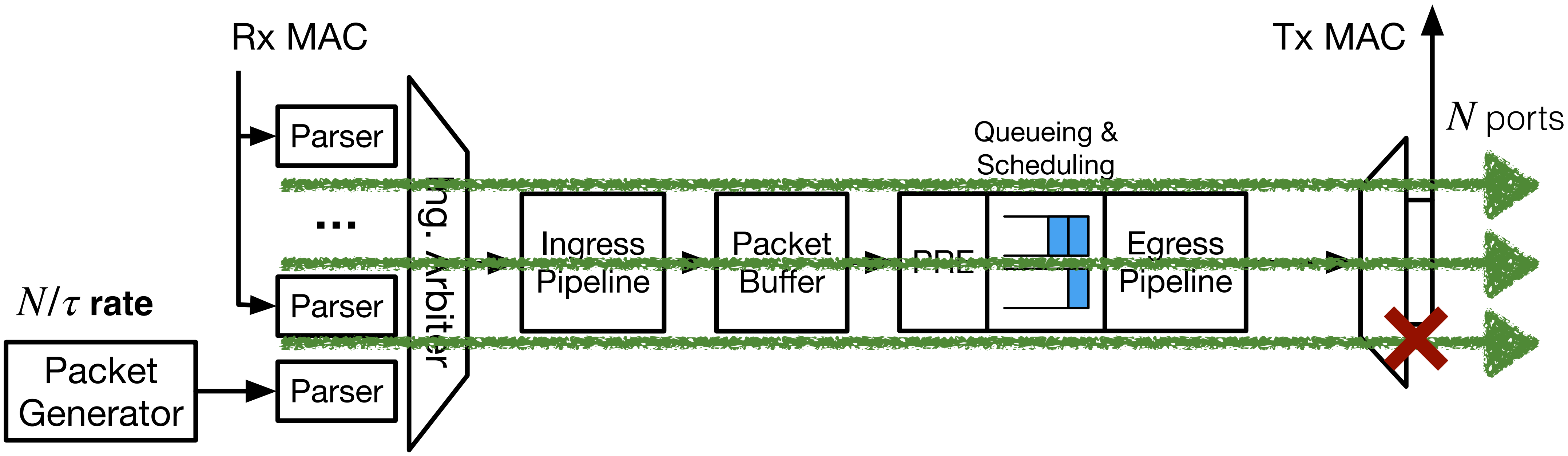


Strawman: blind packet generation



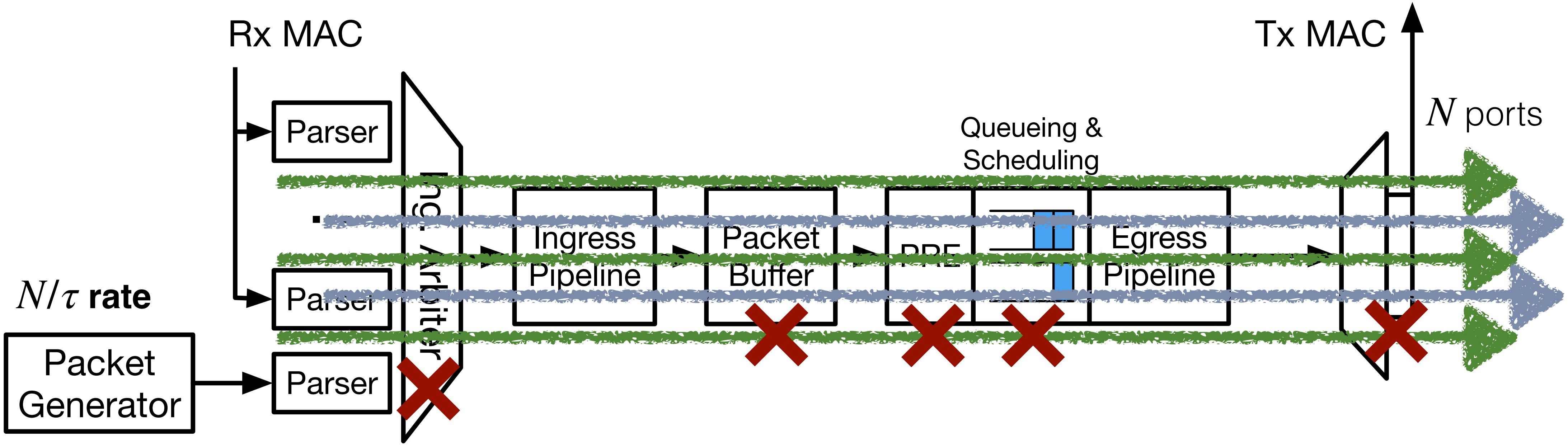
Predictability even there is no user traffic ✓

Problems with blind packet generation



#1 Scalability: overwhelm generator capacity to satisfy target rate for all ports

Problems with blind packet generation

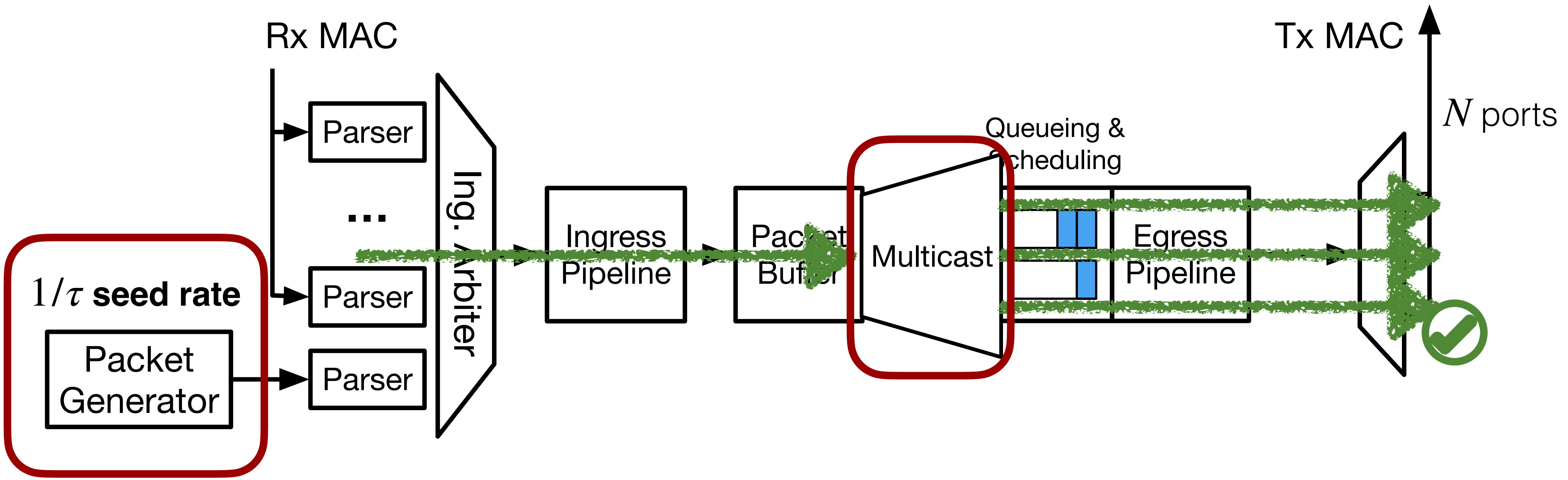


#1 Scalability: overwhelm generator capacity to satisfy target rate for all ports

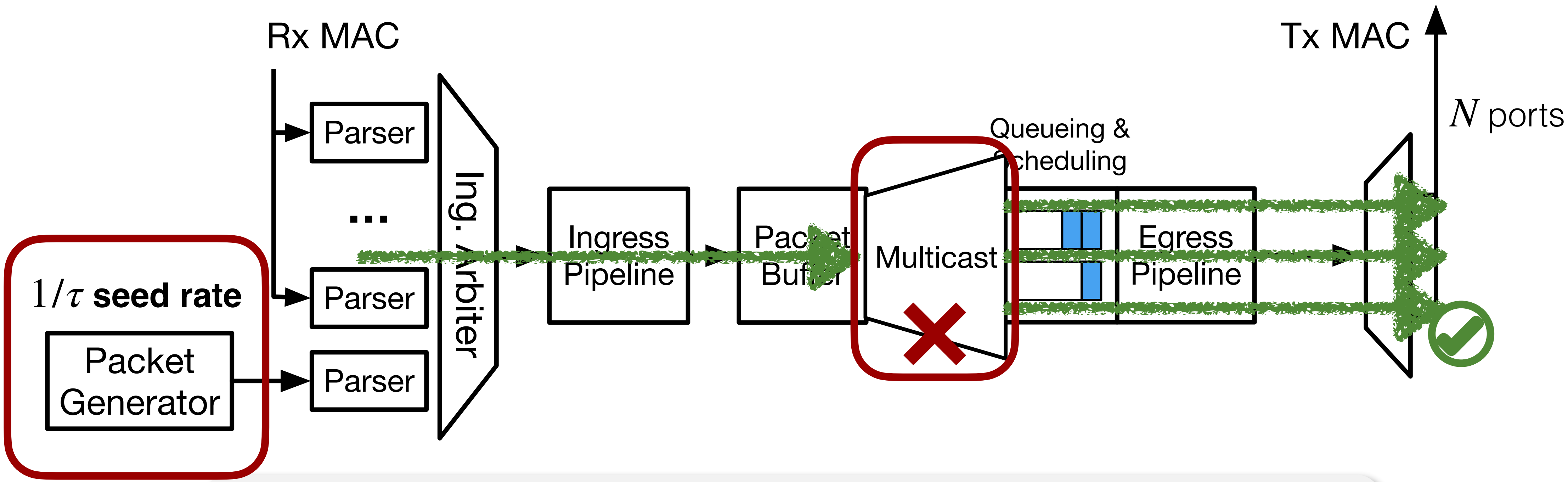
#2 Cross-traffic contention: affect throughput, latency, or loss of **user traffic**!

Problem #1 : scalability

Solution: *seed stream amplification*



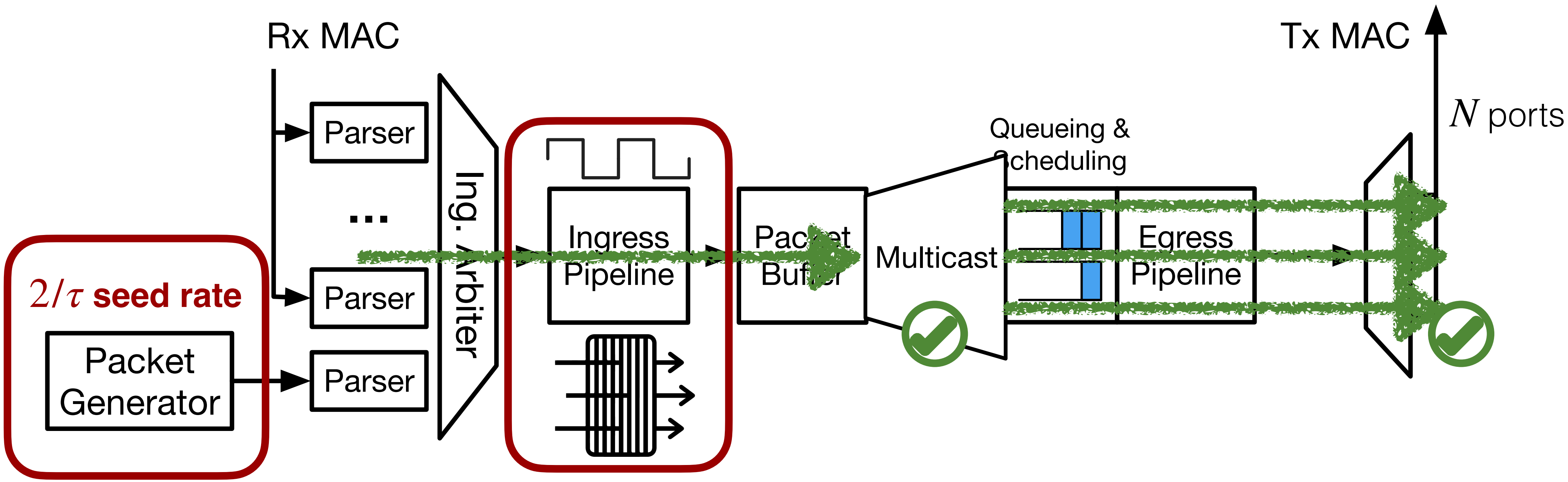
Problem #2: cross-traffic contention at PRE



Monopolize usage and waste PRE packet-level BW!

Problem #2: cross-traffic contention at PRE

Solution**: amplify seed stream **on-demand

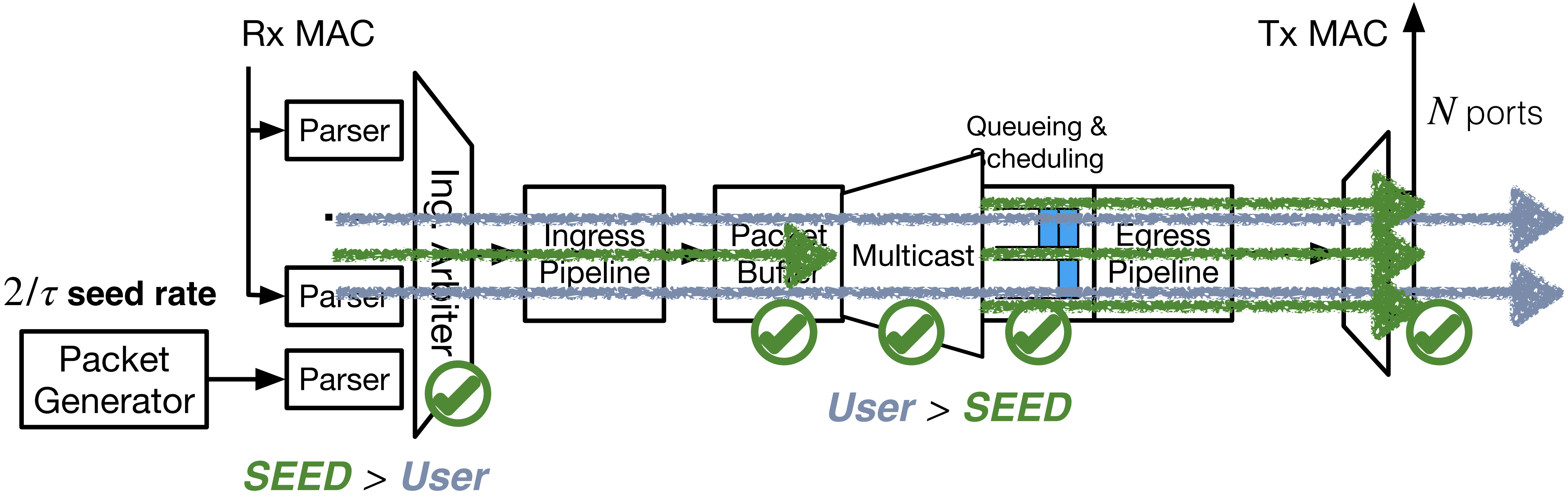


Selective filtering

- Per-egress port bitmap indicating packet presence in the last $\tau/2$ cycle
- If not, replicate an IDLE to the port

Problem: other contention points

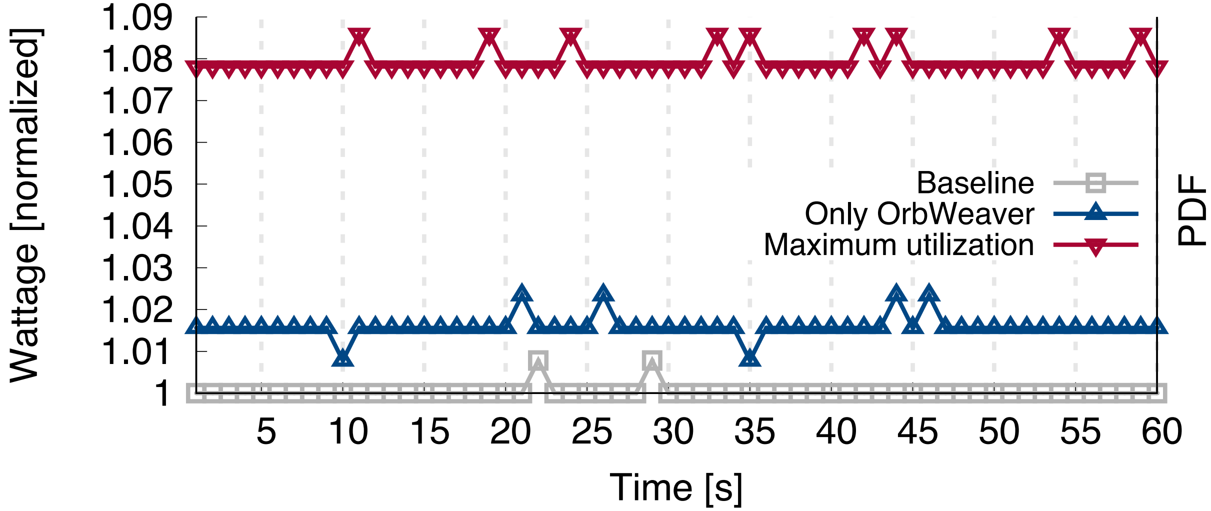
***Solution**: leverage rich configuration options for priorities and buffer management*

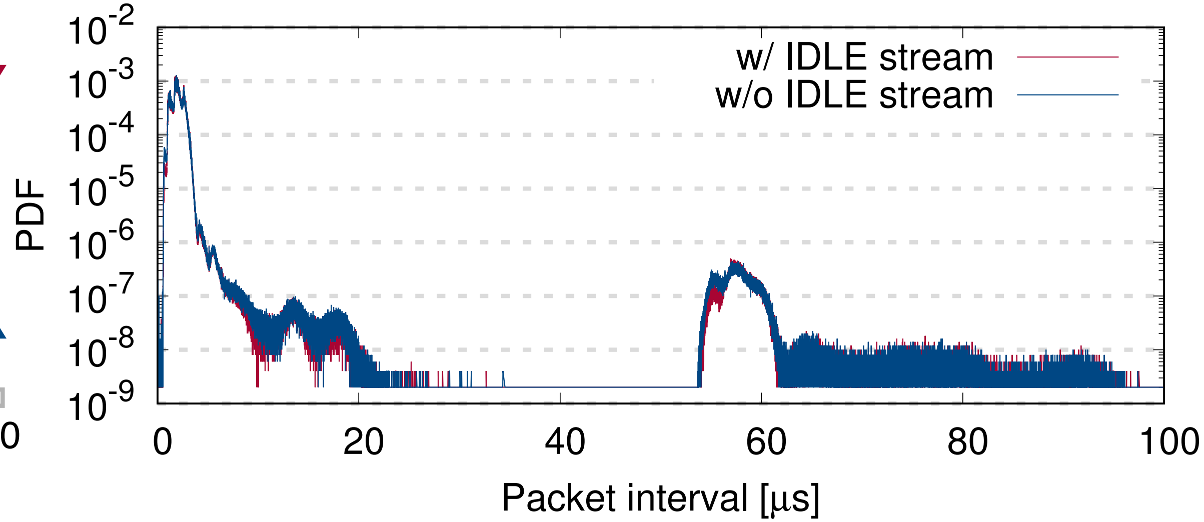


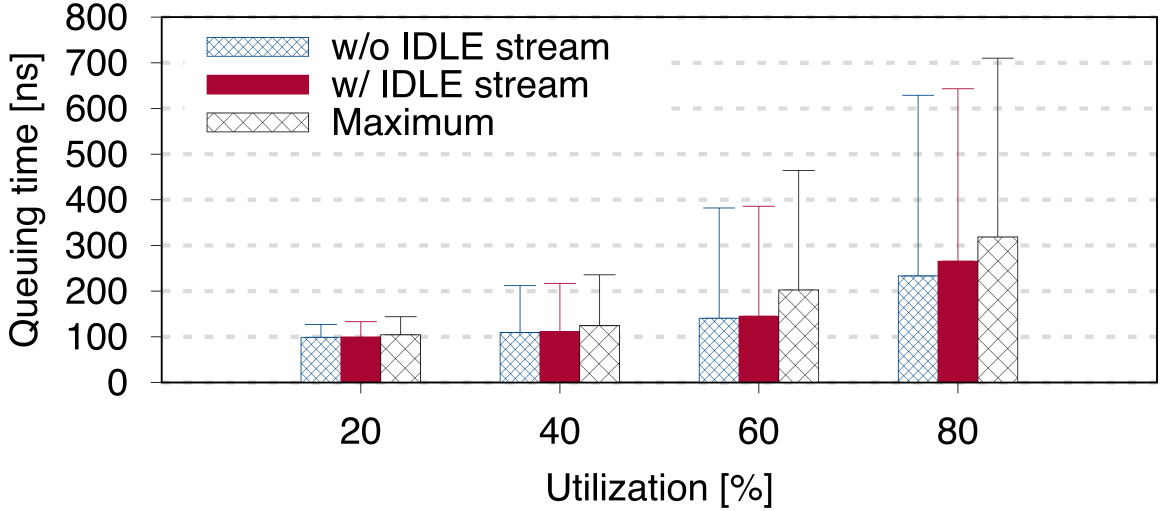
- *Zero impact of weaved stream predictability*
- *Zero impact of **user traffic** throughput or buffer usage*
- *Negligible impact of latency of **user packets***

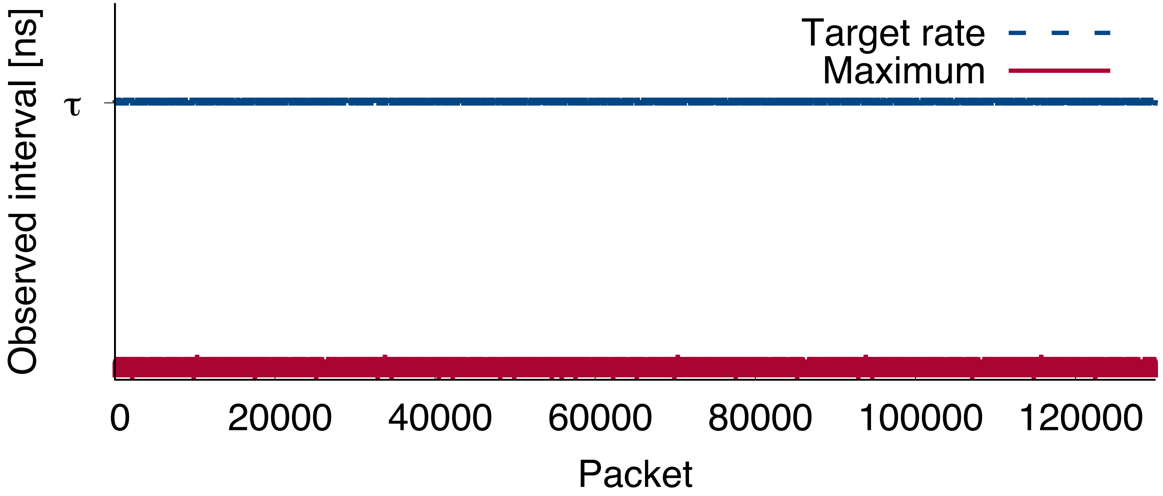
Implementation and evaluation

Hardware prototype on a pair of Wedge100BF-32X Tofino switches

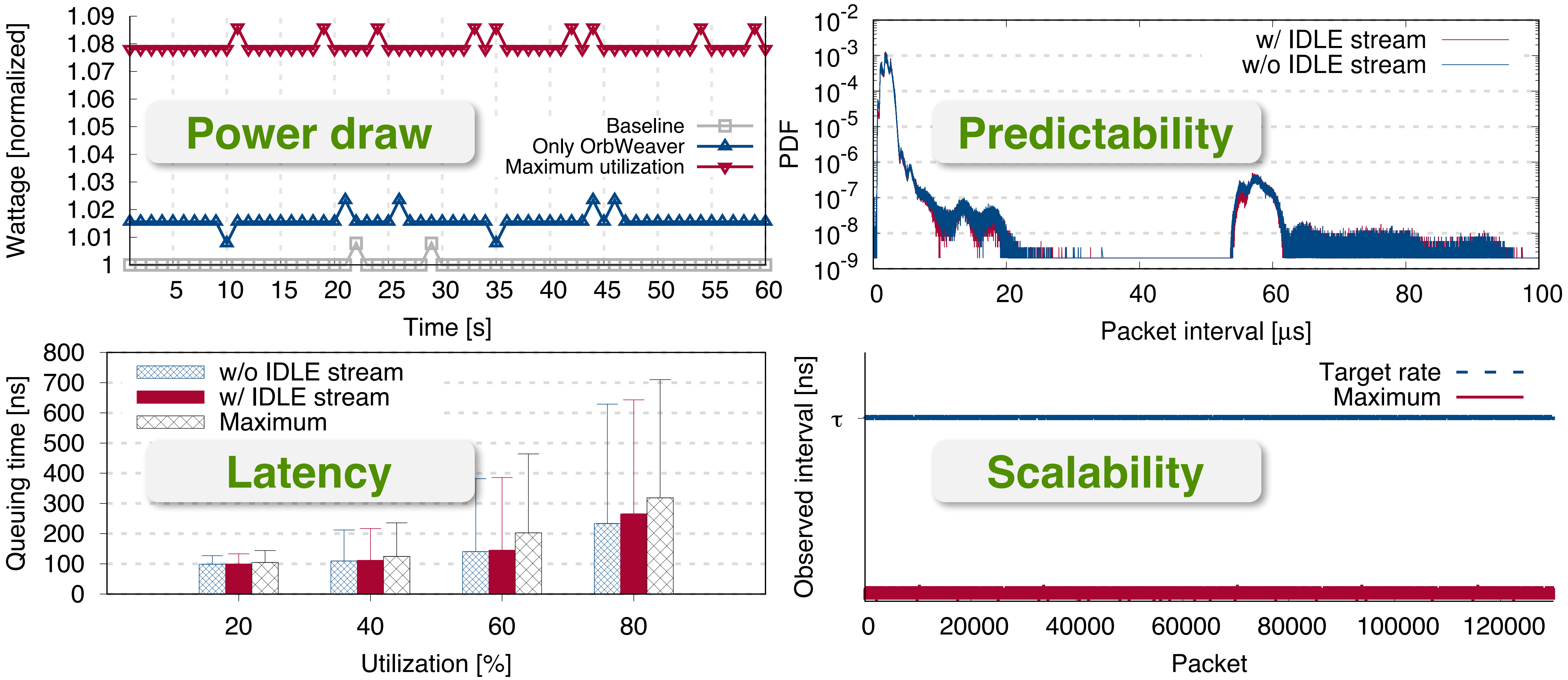




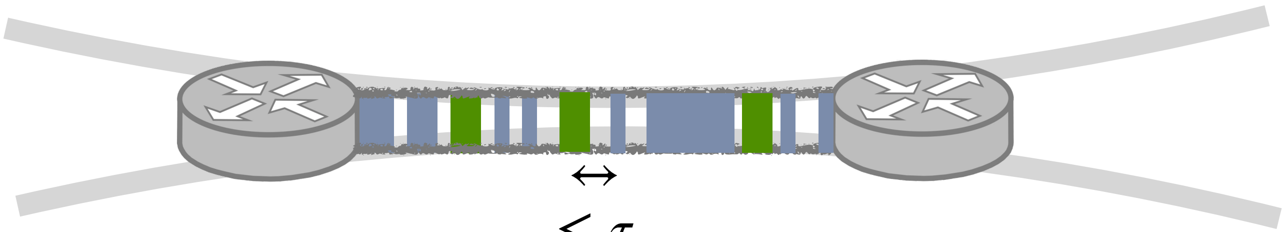




Takeaway: **Little-to-no impact** of power draw, latency, or throughput while guaranteeing **predictability** of the weaved stream!



OrbWeaver use cases



$\leftrightarrow$

$\leq \tau$

[R1 Predictability] → Infer network state at fine-granularity!

[R2 Little-to-zero overhead] → Inject information using IDLE cycles!

OrbWeaver use cases

Self-healing failure detection

Flowlet load imbalance

Performance aware routing

Latency localization

In-band telemetry

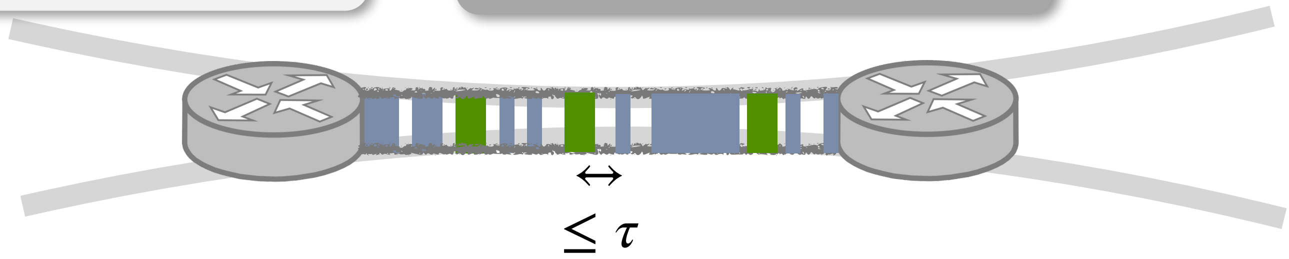
Header compression

Microburst detection

Consistent replicas

Packet forensics

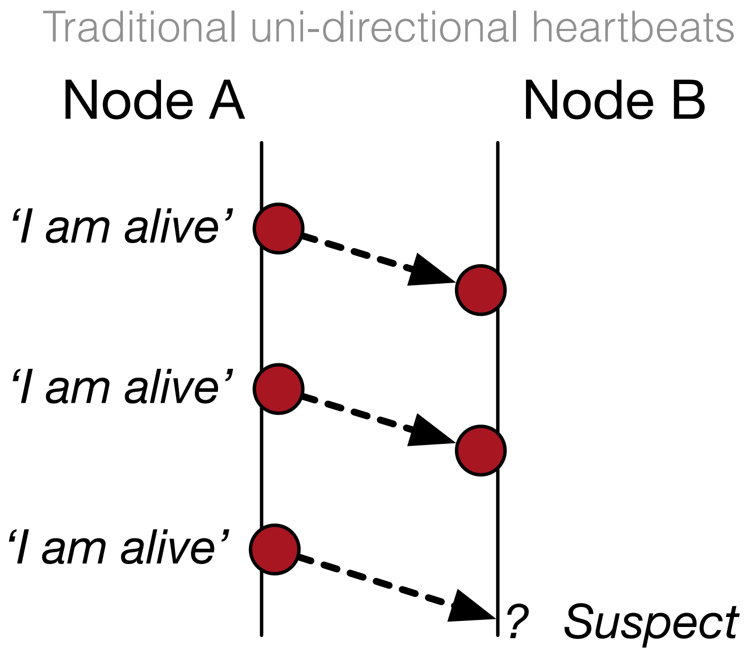
Clock synchronization



[R1 Predictability] → Infer network state at fine-granularity!

[R2 Little-to-zero overhead] → Inject information using IDLE cycles!

Example: failure detection



Common approach:

Periodic, high priority heartbeats

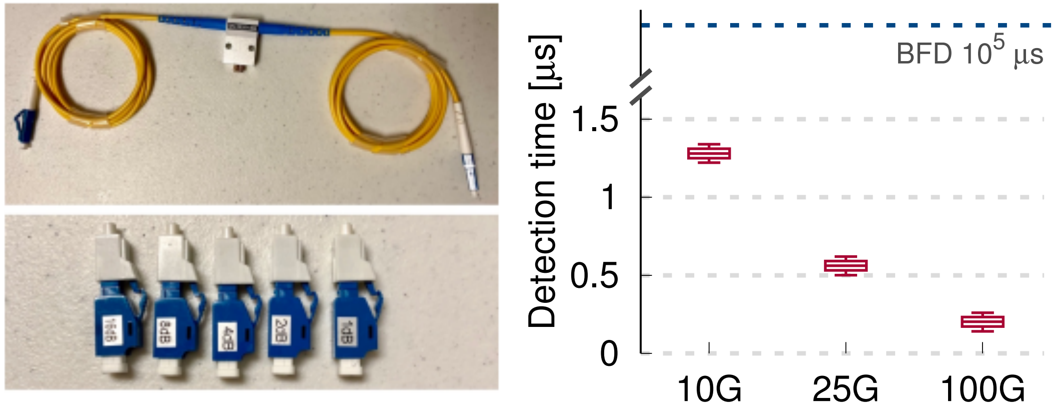
***Fundamentally indistinguishable**: message drop or actual failure?*

Empirically, use conservative detection thresholds

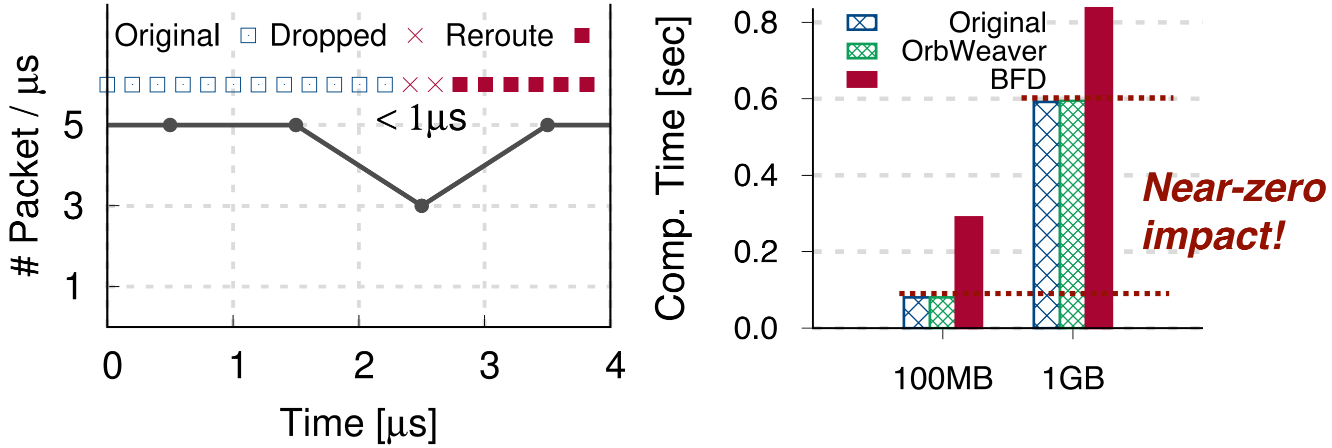
Failure detection with OrbWeaver

Before: Weak guarantee of the messaging channel

After: Guaranteed maximum inter-packet gap (120ns for 100 GbE)



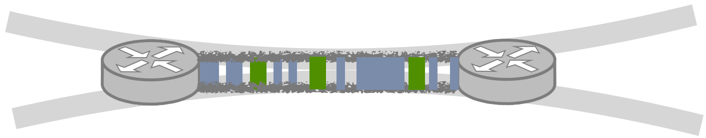
Detection time of emulated failures using optical attenuators under varying link speeds



Instantaneous self-healing failure mitigation when combined with data-plane reroute

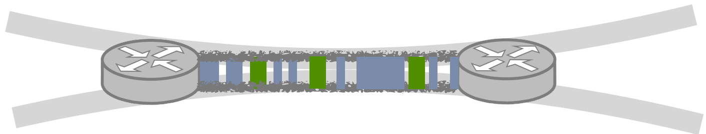
OrbWeaver pushes the detection speed to its ***limits***

OrbWeaver: summary



- **Weaved stream abstraction** to harvest IDLE cycles
  - Sufficient for many in-band control functions
  - *Don't* need to choose between coordination fidelity and bandwidth overhead

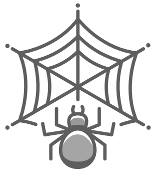
OrbWeaver: summary



- **Weaved stream abstraction** to harvest IDLE cycles
  - Sufficient for many in-band control functions
  - *Don't* need to choose between coordination fidelity and bandwidth overhead

- Implementable on today's RMT switches
  - Push the utilization of IDLE cycles to its limits
  - Guarantee predictability with little-to-zero overhead

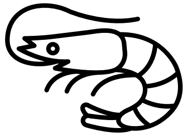
Outline



OrbWeaver *(NSDI 2022)*
Reusing IDLE link cycles for in-band control communication

Reuse

Zero-waste designs



Mantis (*SIGCOMM 2020)*
Recycling switch resources for flexible, sub-RTT reactions

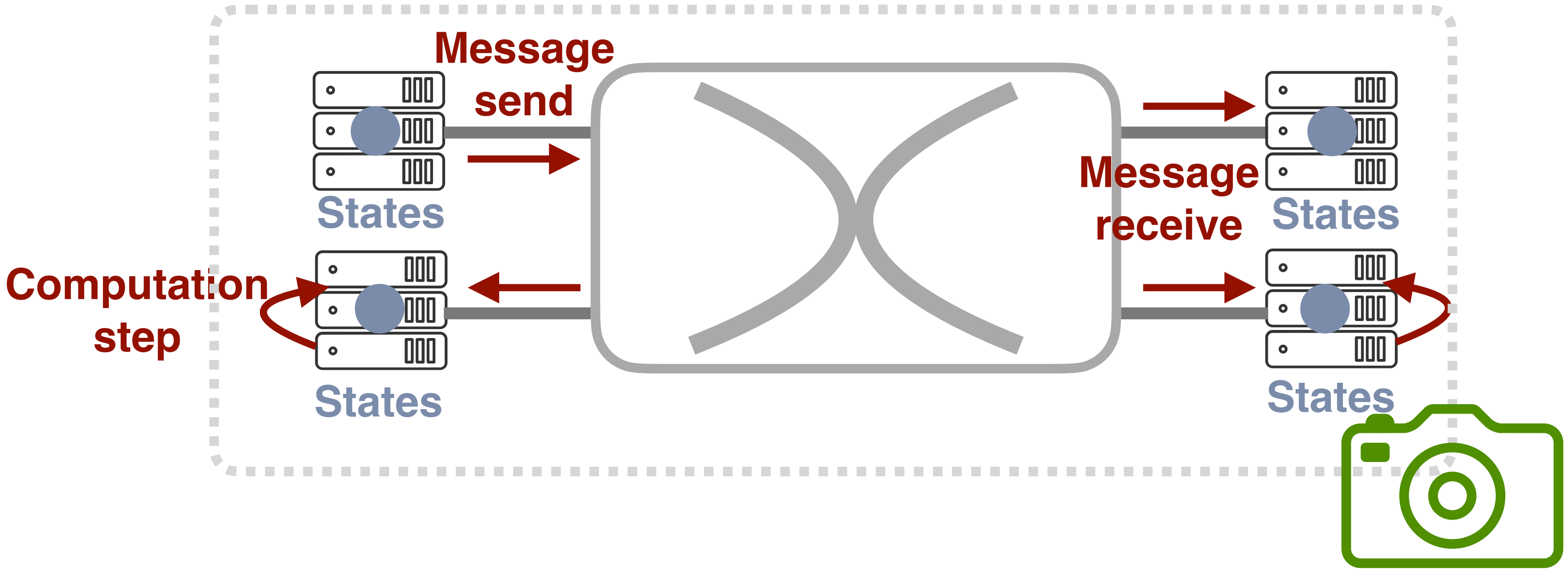
Recycle

Beaver *(OSDI 2024)*
Reducing ‘tax’ of partial snapshots for distributed cloud services

Reduce

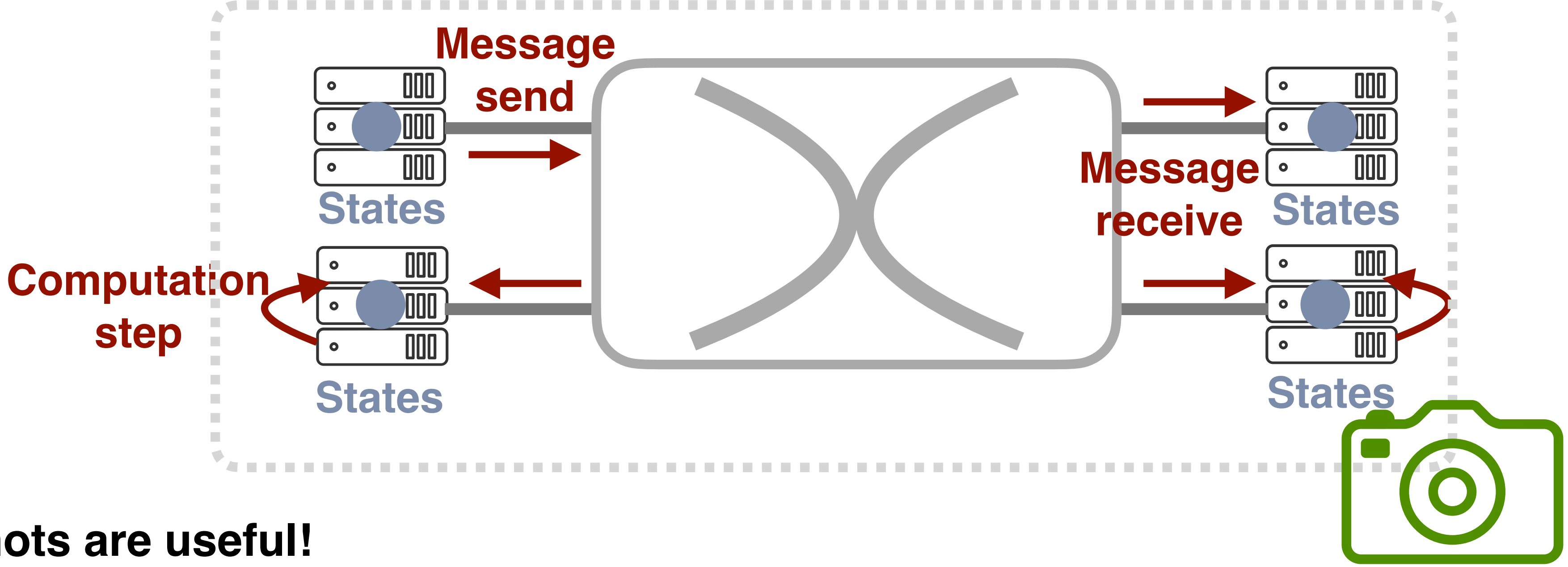
Let's talk about snapshots

Distributed snapshots: a class of distributed algorithms to capture **consistent, global view** of **states**



Let's talk about snapshots

Distributed snapshots: a class of distributed algorithms to capture **consistent, global view** of **states**



Snapshots are useful!

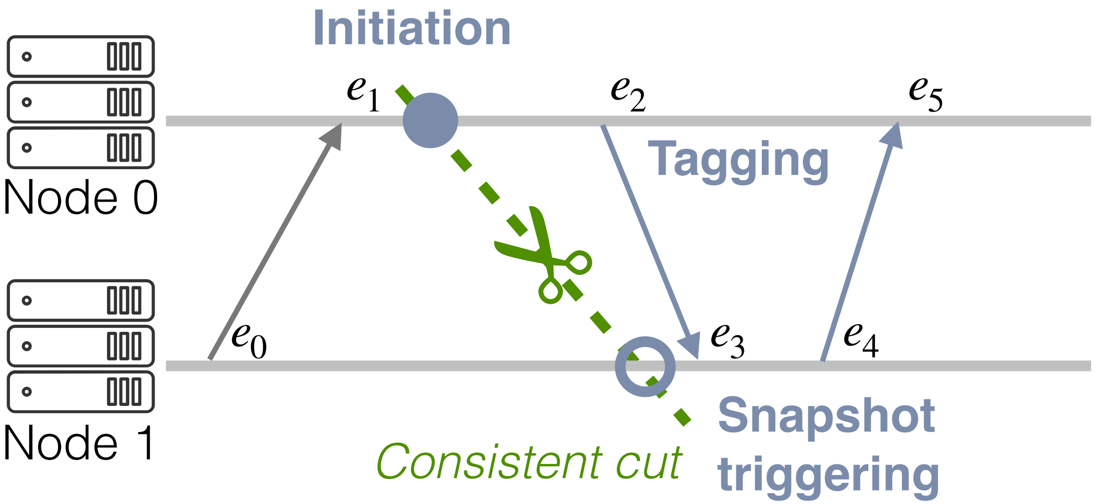
Network telemetry *Distributed software debugging* *Deadlock detection* *Checkpointing and failure recovery* ……

Classic distributed snapshots

e.g., Chandy-Lamport (TOCS 1985)

Classic distributed snapshots

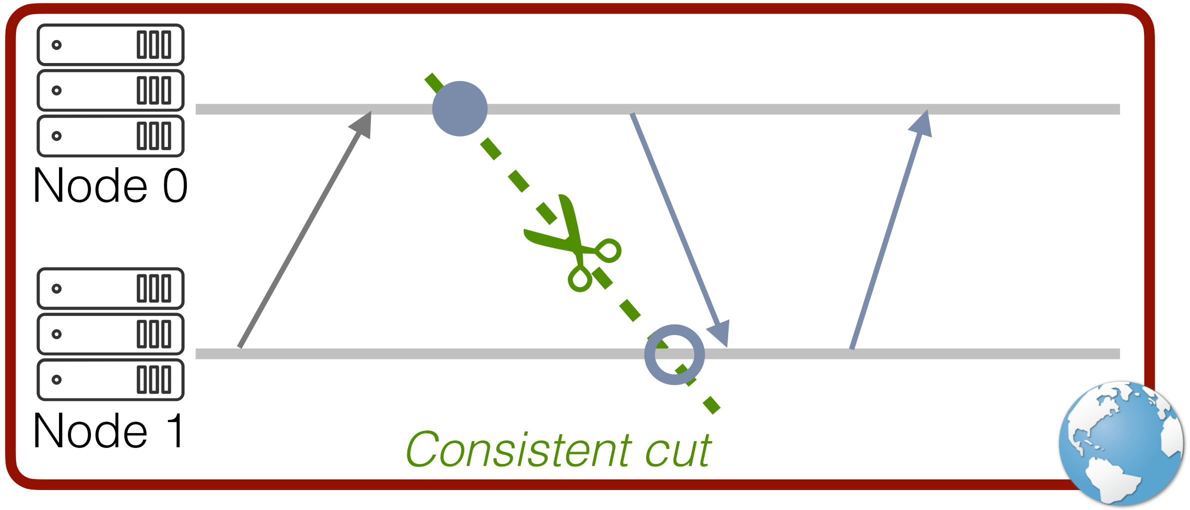
e.g., Chandy-Lamport (TOCS 1985)



Guarantee of causal consistency ✅

*For **any** event e in the cut, if $e' \rightarrow e$ (Lamport's 'happened before'), e' is in the cut.*

Classic snapshots operate in an isolated universe

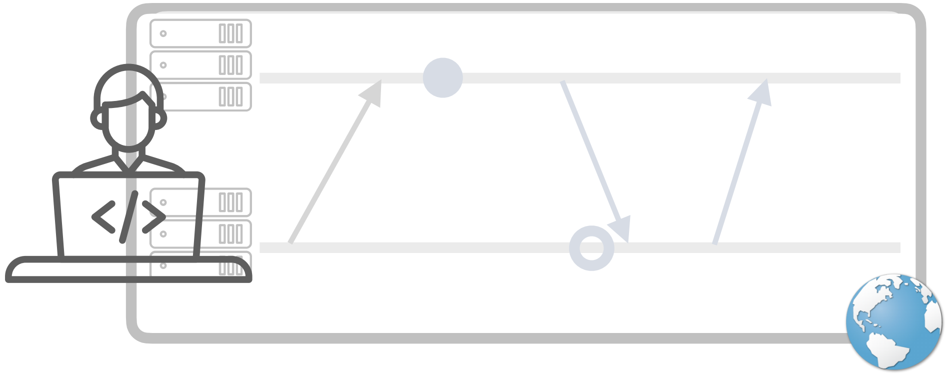


Utopian: isolated 'universe' of nodes

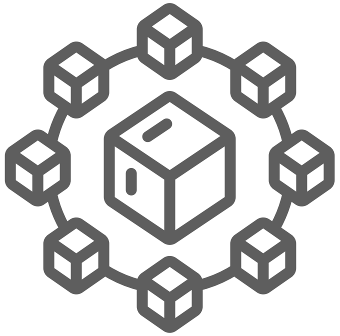
Fundamental assumption:
The set of participants are ***closed*** under causal propagation.

☹ *Unfortunately, the assumption mismatches the real-world scenarios!*

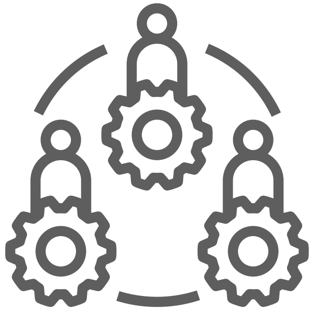
The assumption rarely matches reality!



Utopian: isolated 'universe' of nodes

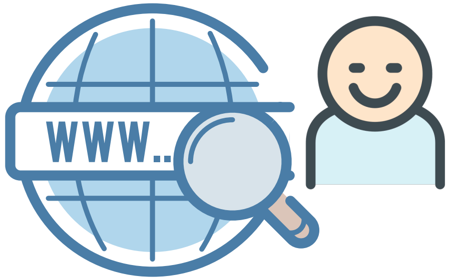


Modular services



Instrumentation constraints

Costs and overheads



Hidden causality due to human

The assumption mismatches the reality!

Unrealistic to assume *zero* external interaction

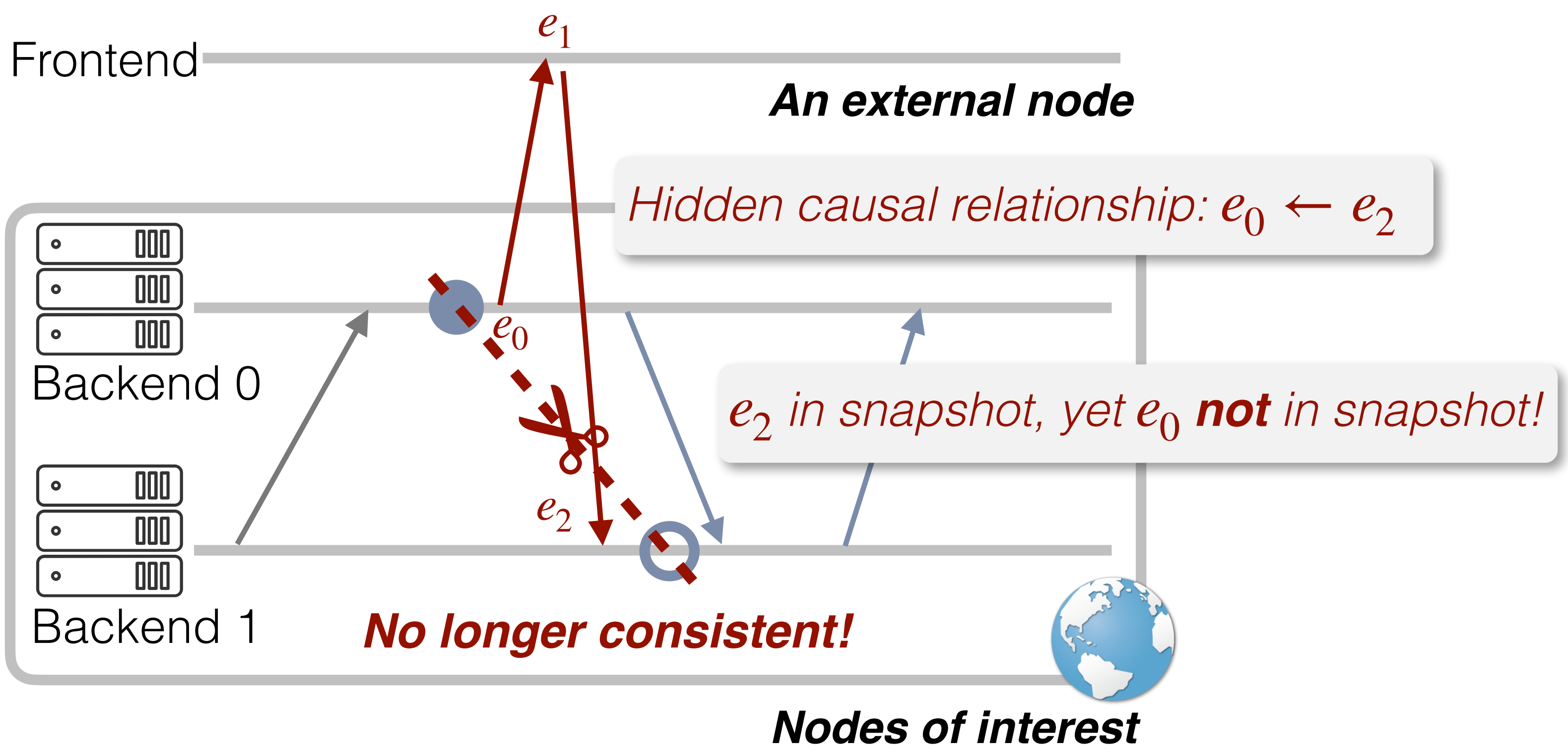
Impractical to instrument *all* processes

Utopian: isolated 'universe' of nodes

Costs and overheads

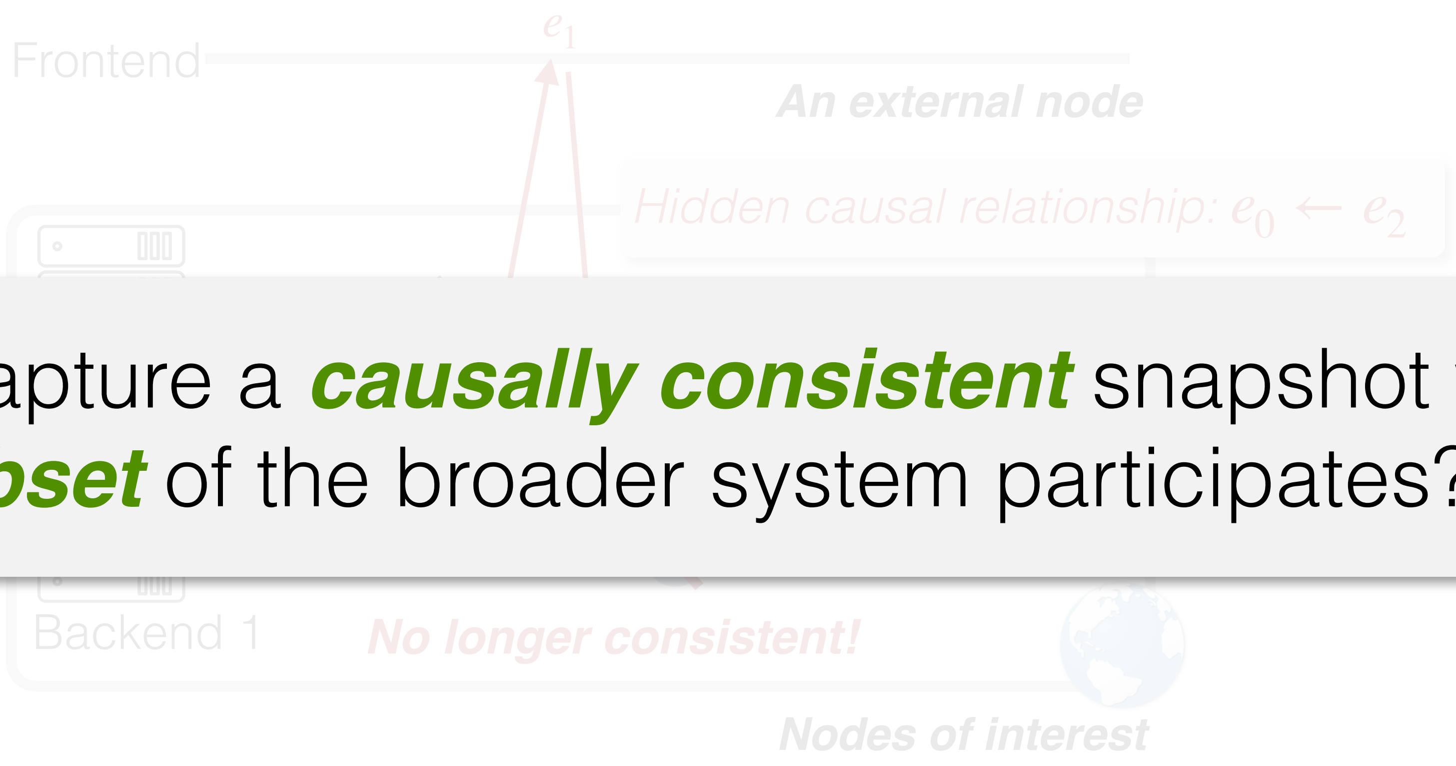
Hidden causality due to human

Consequences?



A single external node can break the guarantee!

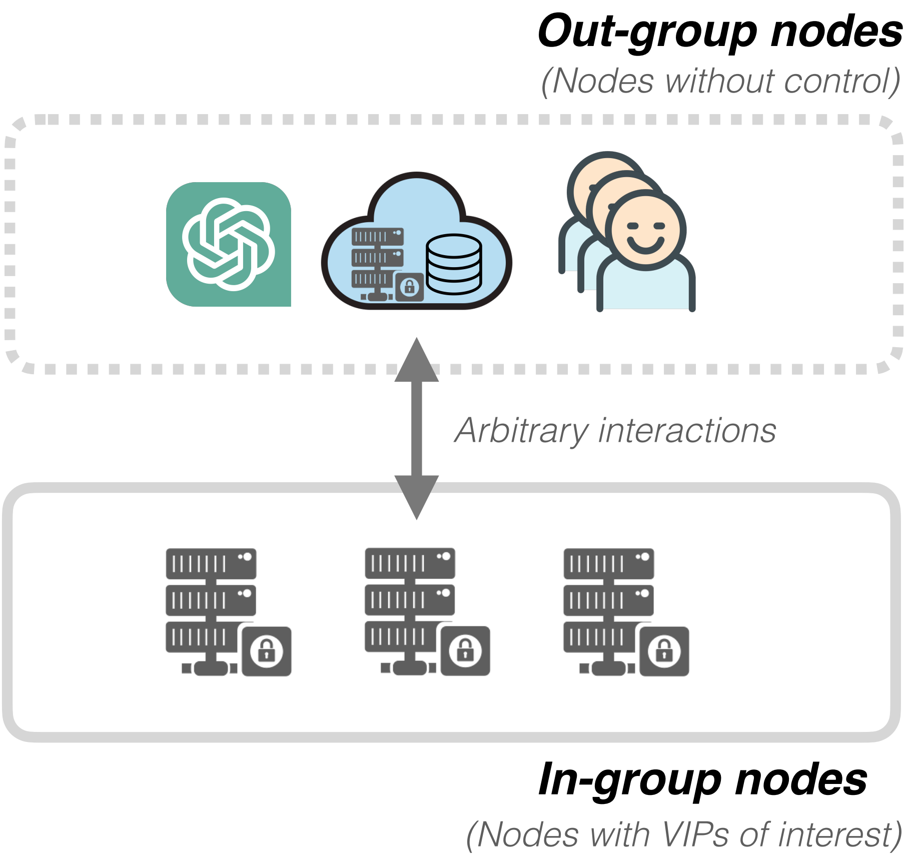
Consequences?



Can we capture a ***causally consistent*** snapshot when a ***subset*** of the broader system participates?

A single external node can break the guarantee!

Beaver: practical partial snapshots



The same causal consistency abstraction

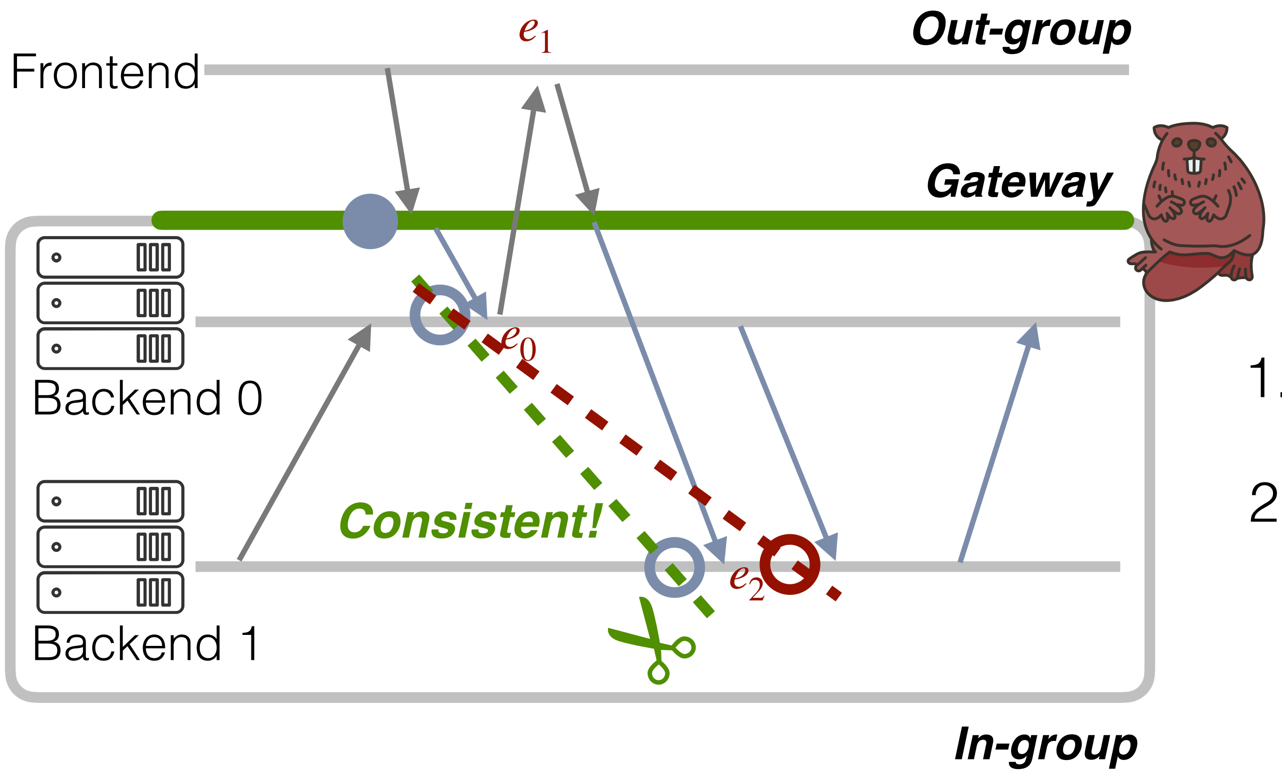
*Even when the target service interact with **external, black box services** (arbitrary number, scale, placement, or semantics) via **arbitrary pattern** (including multi-hop propagation of causal dependencies)*

Zero impact over existing service traffic

*That is, **absence of blocking or any form of delaying operations** during distributed coordination*


How is it even possible ***without*** coordinating machines external to those of interest?
Build a dam like a Beaver!
96

Idea 1: Gateway (GW) indirection



Beaver's gateway (GW) indirection:

1. Initiate GW to enter snapshot out-of-band

2. Mark ***inbound*** packets correspondingly

Before: **inconsistent** cut at ○ (after e_2)

With GW: **consistent** cut at ○ (before e_2)

Formalizing idea 1: Monolithic Gateway Marking

***Theorem* 1.** *With MGM, a partial snapshot C_{part} for $P^{in} \subseteq P$ is causally consistent, that is, $\forall e \in C_{part}$, if $e'.p \in P^{in} \wedge e' \rightarrow e$, then $e' \in C_{part}$.*

Proof. Let $e.p = p_i^{in}$ and $e'.p = p_j^{in}$. There are 3 cases:

1. Both events occur in the same process, i.e., $i = j$.
2. $i \neq j$ and the causality relationship $e' \rightarrow e$ is imposed purely by in-group messages.
3. Otherwise, the causality relationship $e' \rightarrow e$ involves *at least* one $p \in P^{out}$.

In cases (1) and (2), the theorem is trivially true using identical logic to proofs of traditional distributed snapshot protocols. We prove (3) by contradiction.

Assume $(e \in C_{part}) \wedge (\exists e' \rightarrow e)$ but $(e' \notin C_{part})$. With (3), $e' \rightarrow e$ means that there must exist some e^{out} (at an out-group process) satisfying $e' \rightarrow e^{out} \rightarrow e$. Now, because $e' \notin C_{part}$, we know $e^{ss}_{p_j^{in}} \rightarrow e'$ or $e^{ss}_{p_j^{in}} = e'$, that is, p_j^{in}'s local snapshot happened before or during e'. Combined with the fact that the gateway is the original initiator of the snapshot protocol, we know that $e^{ss}_g \rightarrow e' \rightarrow e^{out} \rightarrow e$.

We can focus on a subset of the above causality chain: $e^{ss}_g \rightarrow e$. From the properties of the in-group snapshot protocol, $e^{ss}_g \rightarrow e$ implies that $e \notin C_{part}$.

This contradicts our original assumption that $e \in C_{part}$! □

Formal proof in paper

Holds even if treating the out-group nodes as black boxes

Sufficient to ***only*** observe the inbound messages

Key ideas in Beaver 

How to ensure consistency without coordinating external machines?

Idea 1: Indirection through Monolithic Gateway Marking (MGM)

How to enforce MGM practically in today's network?

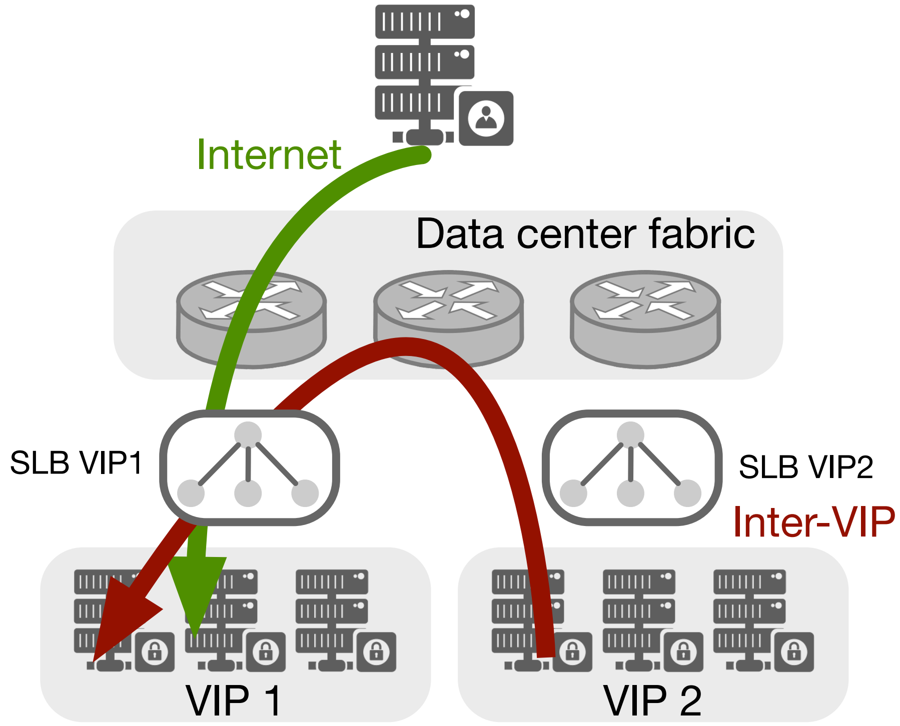
Challenge 1 How to instantiate GW cost-effectively?

Challenge 2 How to handle asynchronous GWs?

Challenge 1 : instantiating GWs

☹ Rerouting all inbound traffic through the GW is ***costly***

💡 Cloud data centers already place layer-4 load balancers (SLBs)



☺ Repurpose SLBs for in-situ marking

Key ideas in Beaver 

How to ensure consistency without coordinating external machines?

Idea 1: Indirection through Monolithic Gateway Marking (MGM)

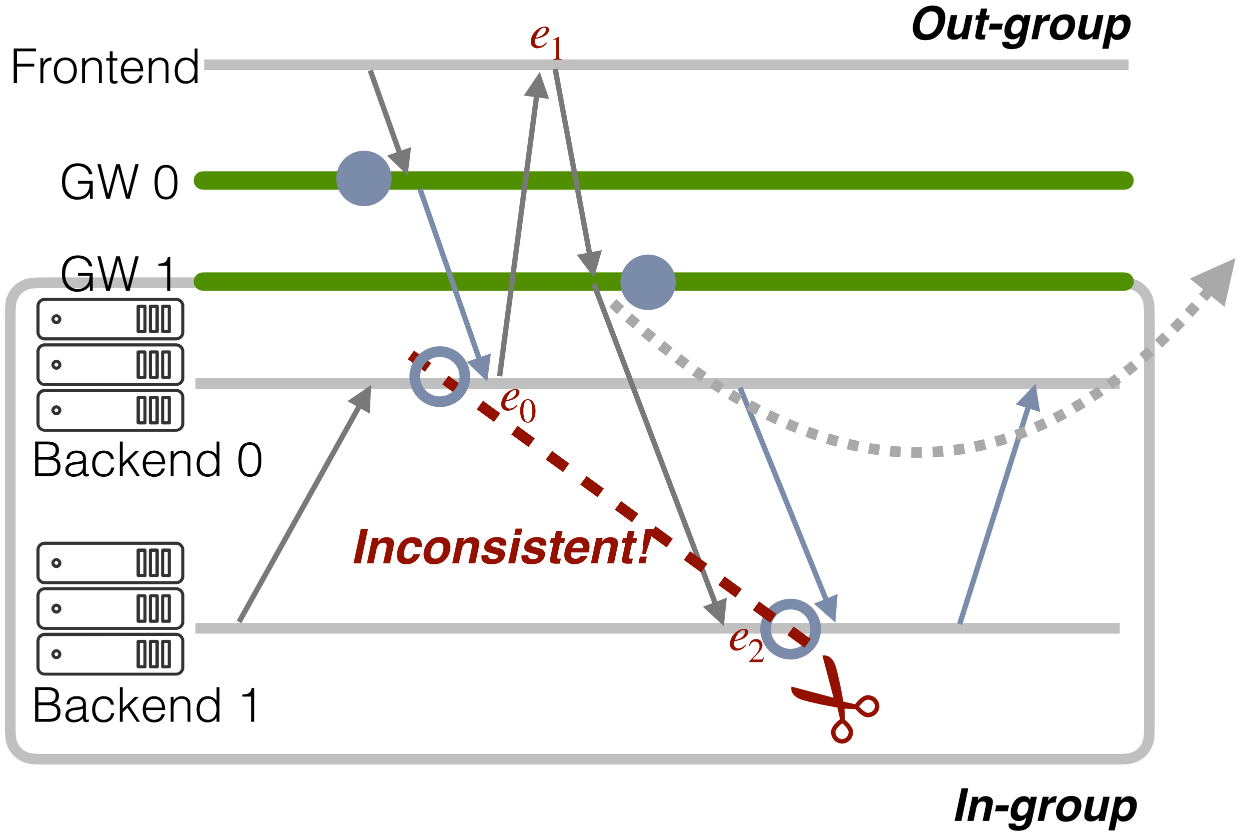
How to enforce MGM practically in today's network?

Challenge 1 How to instantiate GW?

Idea 2: Reuse existing SLBs with unique locations

Challenge 2 How to perform atomic snapshot initiation for asynchronous GWs?

Implications of multiple SLBs

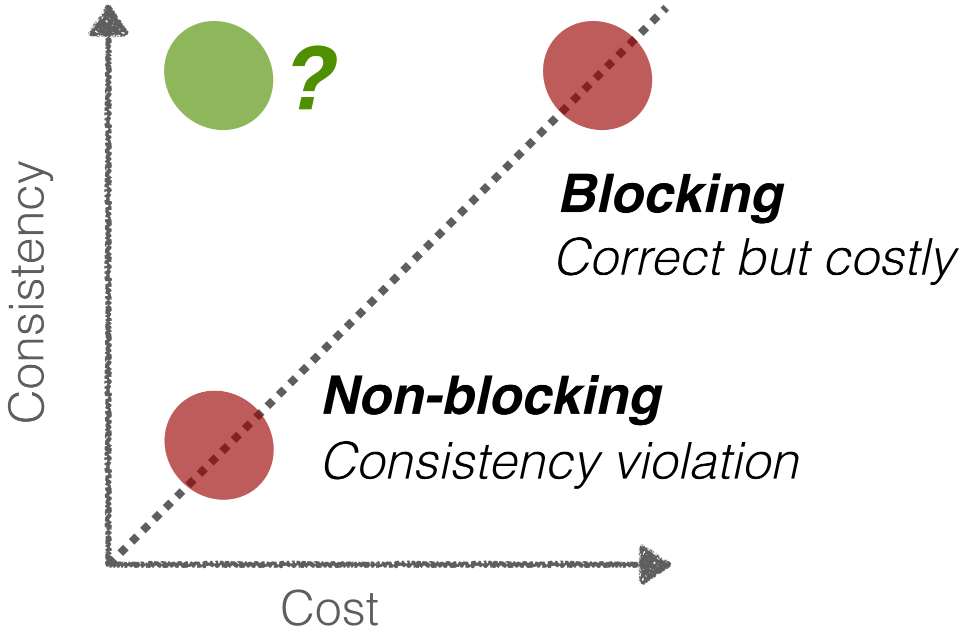


GW 1 hasn't initiated the new snapshot mode to mark it, triggering the ***violation***

e_2 in snapshot, yet e_0 that leads to it is not, inconsistent!

Handling multiple GWs: design space

How about blocking messages to 'atomically' trigger all SLBs?



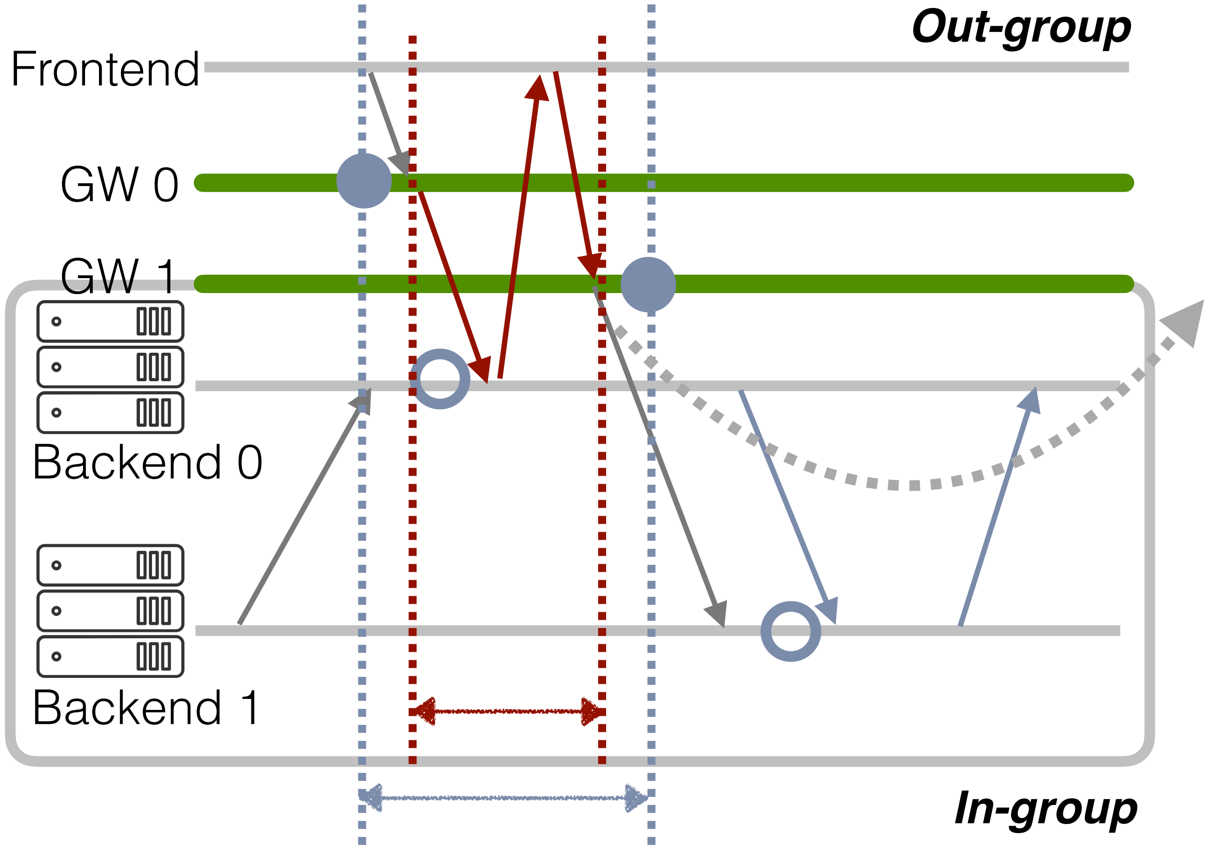
*Can we get both **consistency** and **zero cost**?*

Optimistic Gateway Marking (OGM)

- *Intuition & formalism*
- *Mechanism*

Challenge 2: handling multiple SLBs

***Reflection**: Beyond worst cases, when and how often does the violation occur?*



Time gap between SLB initiation points

Observation:

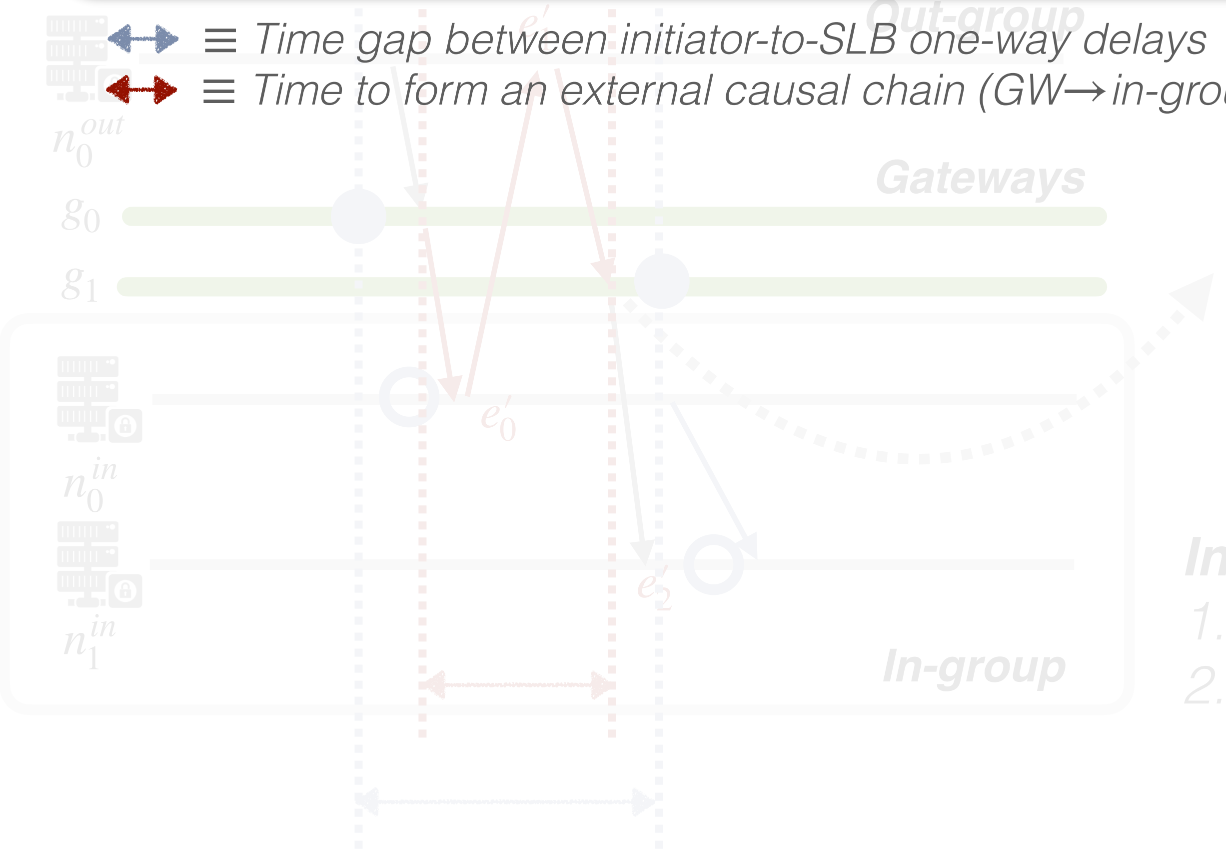
Causally relevant messages are rare!
GW→in-group→out-group→GW (external causal chain)

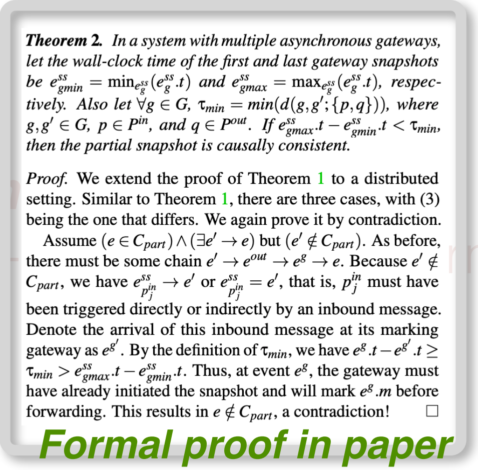
***Intuition**: the resulting snapshot is consistent*
*1. if ↔ is **large enough***
*2. or if ↔ is **'close' enough***

Theorem: if ↔ (blue) < ↔ (red), the partial snapshot is consistent!

↔ (blue) ≡ *Time gap between initiator-to-SLB one-way delays*

↔ (red) ≡ *Time to form an external causal chain (GW→in-group→out-group→GW)*





Theorem 2. *In a system with multiple asynchronous gateways, let the wall-clock time of the first and last gateway snapshots be* $e^{ss}_{gmin} = \min_{e^{ss}_g}(e^{ss}_g.t)$ *and* $e^{ss}_{gmax} = \max_{e^{ss}_g}(e^{ss}_g.t)$, *respectively. Also let* $\forall g \in G$, $\tau_{min} = min(d(g,g';\{p,q\}))$, *where* $g, g' \in G$, $p \in P^{in}$, *and* $q \in P^{out}$. *If* $e^{ss}_{gmax}.t - e^{ss}_{gmin}.t < \tau_{min}$, *then the partial snapshot is causally consistent.*

Proof. We extend the proof of Theorem 1 to a distributed setting. Similar to Theorem 1, there are three cases, with (3) being the one that differs. We again prove it by contradiction.

Assume $(e \in C_{part}) \wedge (\exists e' \rightarrow e)$ but $(e' \notin C_{part})$. As before, there must be some chain $e' \rightarrow e^{out} \rightarrow e^{g} \rightarrow e$. Because $e' \notin C_{part}$, we have $e^{ss}_{p^{in}_j} \rightarrow e'$ or $e^{ss}_{p^{in}_j} = e'$, that is, p^{in}_j must have been triggered directly or indirectly by an inbound message. Denote the arrival of this inbound message at its marking gateway as $e^{g'}$. By the definition of τ_{min}, we have $e^{g}.t - e^{g'}.t \geq \tau_{min} > e^{ss}_{gmax}.t - e^{ss}_{gmin}.t$. Thus, at event e^{g}, the gateway must have already initiated the snapshot and will mark $e^{g}.m$ before forwarding. This results in $e \notin C_{part}$, a contradiction! □

Formal proof in paper

Theorem: if ↔ < ↔, the partial snapshot is consistent!

↔ ≡ *Time gap between initiator-to-SLB one-way delays*

↔ ≡ *Time to form an external causal chain (GW→in-group→out-group→GW)*

Observation: condition holds in most cases anyway!

↔ *can **approximate zero***

- *SLBs share the same region*
- *Proper placement of controller*

↔ *is relatively high*

- *≥ 3 trips through the fabric*
- *Higher when the out-group is in another DC or Internet*

Theorem 2. *In a system with multiple asynchronous gateways, let the wall-clock time of the first and last gateway snapshots be $e^{ss}_{gmin} = \min_{e^{ss}_g}(e^{ss}_g.t)$ and $e^{ss}_{gmax} = \max_{e^{ss}_g}(e^{ss}_g.t)$, respectively. Also let $\forall g \in G$, $\tau_{min} = min(d(g,g';\{p,q\}))$, where $g,g' \in G$, $p \in P^{in}$, and $q \in P^{out}$. If $e^{ss}_{gmax}.t - e^{ss}_{gmin}.t < \tau_{min}$, then the partial snapshot is causally consistent.*

Proof. We extend the proof of Theorem 1 to a distributed setting. Similar to Theorem 1, there are three cases, with (3) being the one that differs. We again prove it by contradiction.

Assume $(e \in C_{part}) \wedge (\exists e' \rightarrow e)$ but $(e' \notin C_{part})$. As before, there must be some chain $e' \rightarrow e^{out} \rightarrow e^g \rightarrow e$. Because $e' \notin C_{part}$, we have $e^{ss}_{p^{in}_j} \rightarrow e'$ or $e^{ss}_{p^{in}_j} = e'$, that is, p^{in}_j must have been triggered directly or indirectly by an inbound message. Denote the arrival of this inbound message at its marking gateway as $e^{g'}$. By the definition of τ_{min}, we have $e^g.t - e^{g'}.t \geq \tau_{min} > e^{ss}_{gmax}.t - e^{ss}_{gmin}.t$. Thus, at event e^g, the gateway must have already initiated the snapshot and will mark $e^g.m$ before forwarding. This results in $e \notin C_{part}$, a contradiction! □

Formal proof in paper

Optimistic Gateway Marking (OGM)

→ *Optimistic execution in common cases*

→ *Verification/rejection of snapshots under worst cases*

How does Beaver detect a snapshot violation?

Theorem: if ↔ < ↔, the partial snapshot is consistent

↔ ≡ *Time gap between initiator-to-SLB one-way delays*
↔ ≡ *Time to form an external causal chain (GW→in-group→out-group→GW)*

1. Determine the lower bound of ↔ statically

2. Measure a safe upper bound for ↔ online using a single clock

False positives is fine as one can always retry!

Key ideas in Beaver 

How to ensure consistency without coordinating external machines?

Idea 1: Indirection through Monolithic Gateway Marking (MGM)

How to enforce MGM practically in today's network?

Challenge 1 How to instantiate GW cost-effectively?

Idea 2: Reuse existing SLBs with unique locations

Challenge 2 How to perform atomic snapshot initiation for asynchronous GWs?

Idea 3: Optimistic Gateway Marking (OGM)

- Optimistic execution *in common cases*
- Verification/rejection of snapshot *under worst cases*

Key ideas in Beaver

How to ensure consistency without coordinating external machines?

Ide

Ch

Ide

More details about Beaver's protocol…

- Synchronization-free snapshot verification
- Supporting parallel snapshots
- Handling failures
- Handling packet loss, delay, and reordering
- …

Challenge 2 How to handle asynchronous GWs?

Idea 3: Optimistic Gateway Marking (OGM)

- Optimistic execution *in common cases*
- Verification/rejection of snapshot *under worst cases*

Implementation and evaluation

SLB-associated workflow

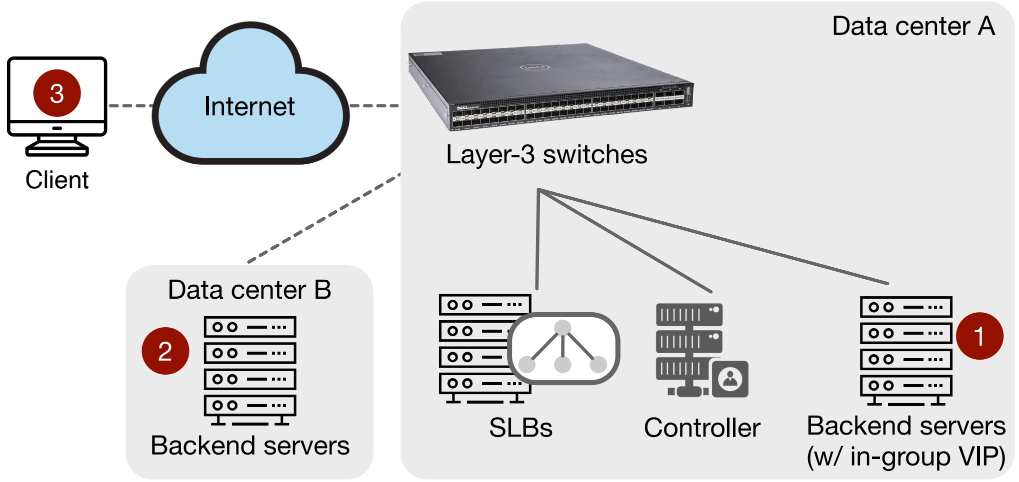
- Layer-3 ECMP forwarding per service VIPs: DELL EMC PowerSwitch S4048-ON
- 1860 LoC for core SLB functions in DPDK
- 1040 LoC for backend server functions in XDP and tc

Beaver protocol integration (minimal logic)

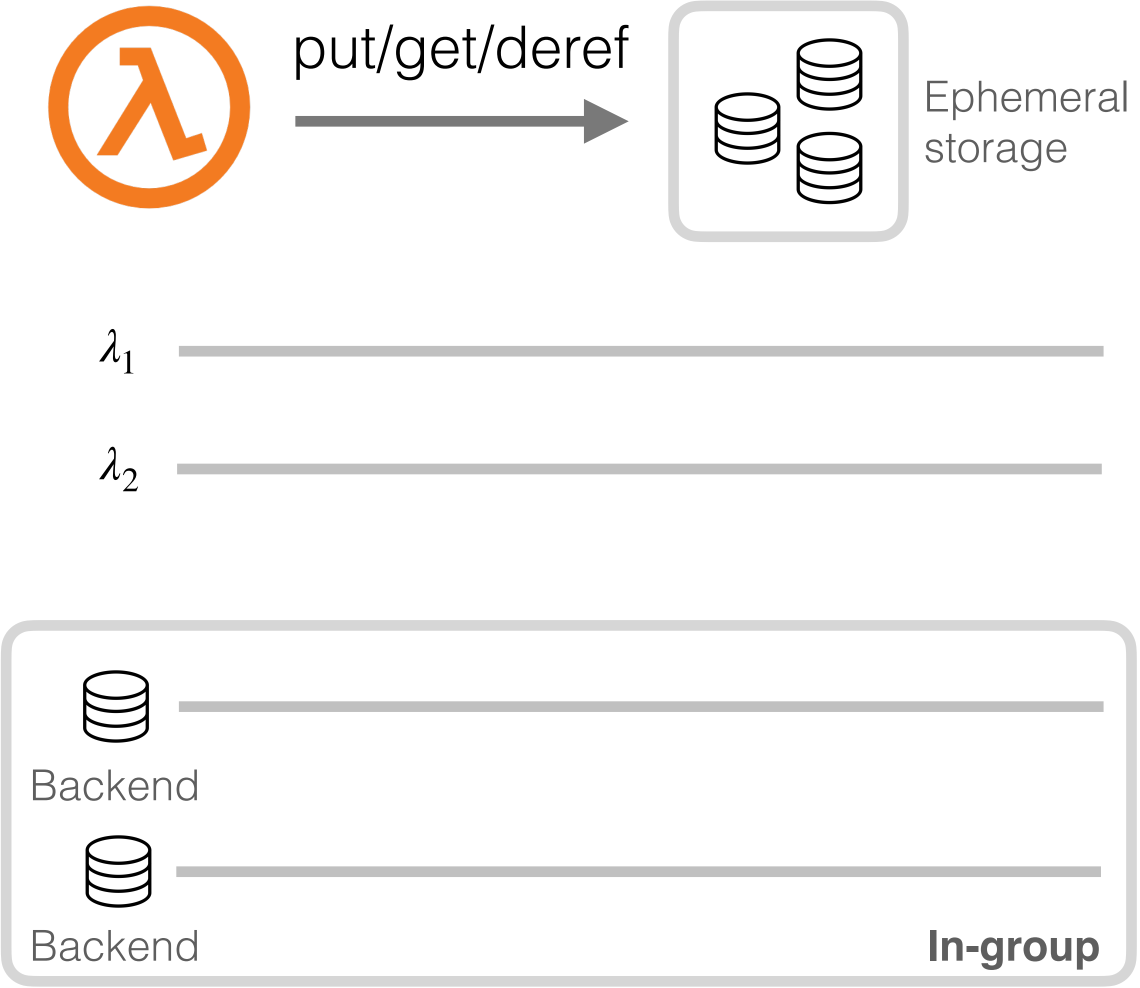
- 68 LoC for SLB DPDK data path logic
- 102 LoC for eBPF at in-group VMs

Topology

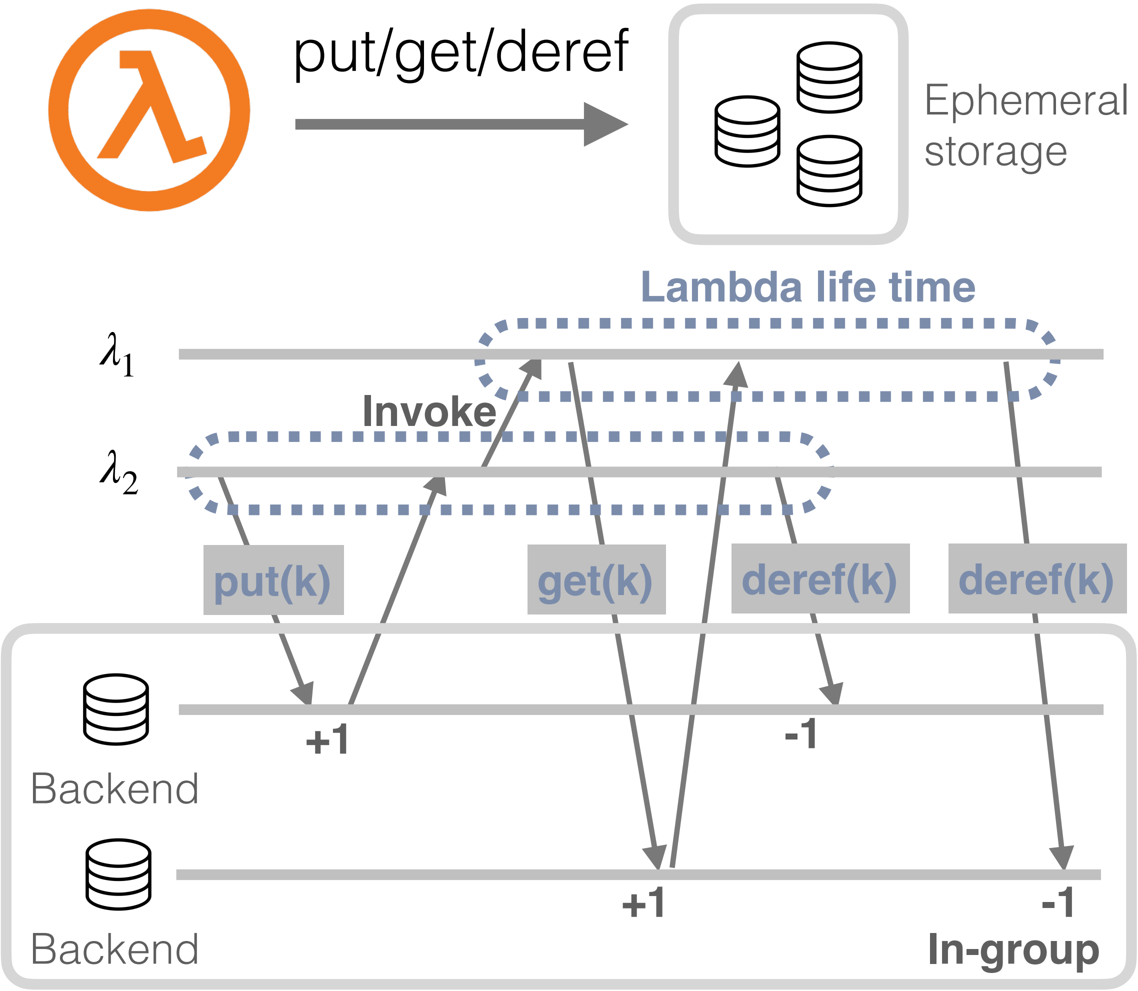
- Out-group locations: within the same DC, DC at a different region, or on the Internet
- Scale up to 16 SLB servers and 1024 backend applications



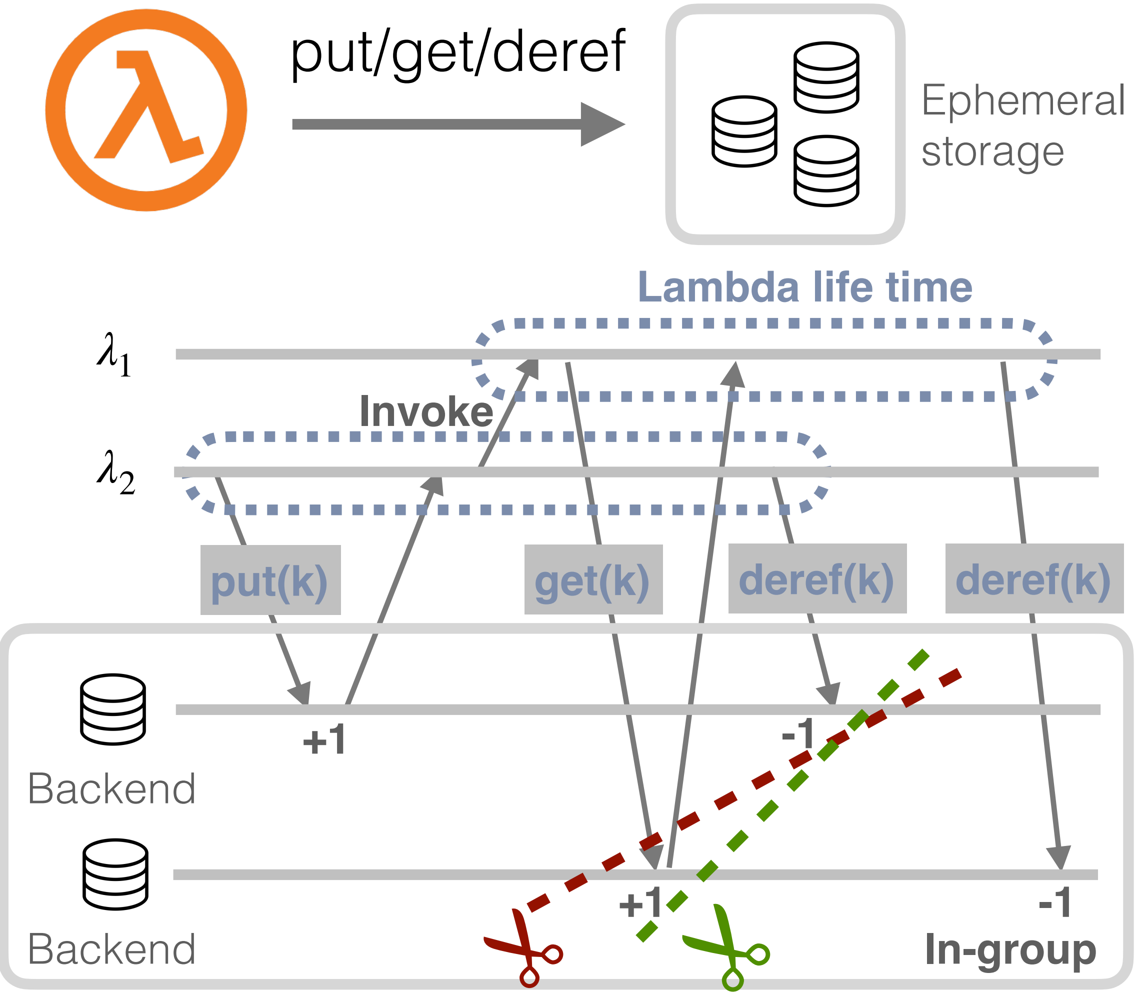
Example: garbage collection for ephemeral storage



Example: garbage collection for ephemeral storage



Example: garbage collection for ephemeral storage



Strawman

Reference count = 0, unsafe recycle decision of k!

Reference count = 1, safe decision recognizing open reference to k

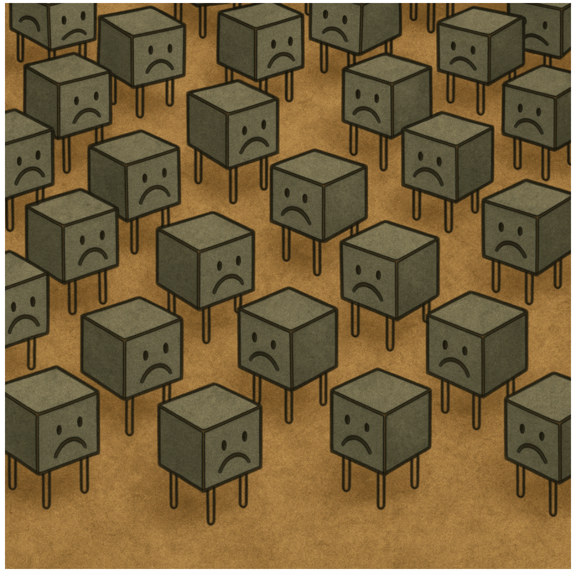
Beaver: summary

The first partial snapshot protocol that extends classic distributed snapshots in **practical cloud settings**

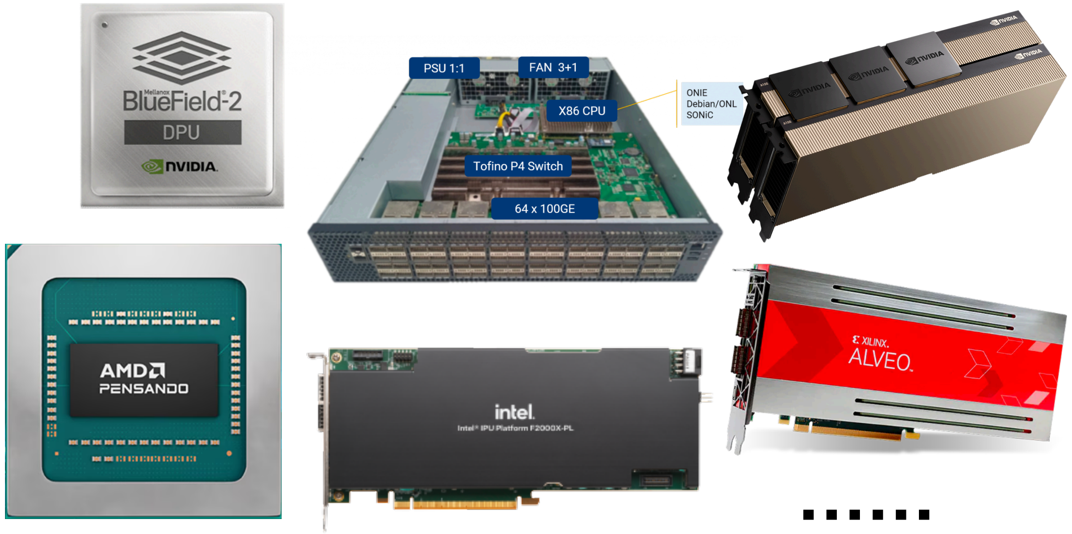
Guarantees **causal consistency** while incurring **minimal changes and overheads**

Key idea: Exploit data center characteristics (e.g., unique topologies)

…something I am excited about

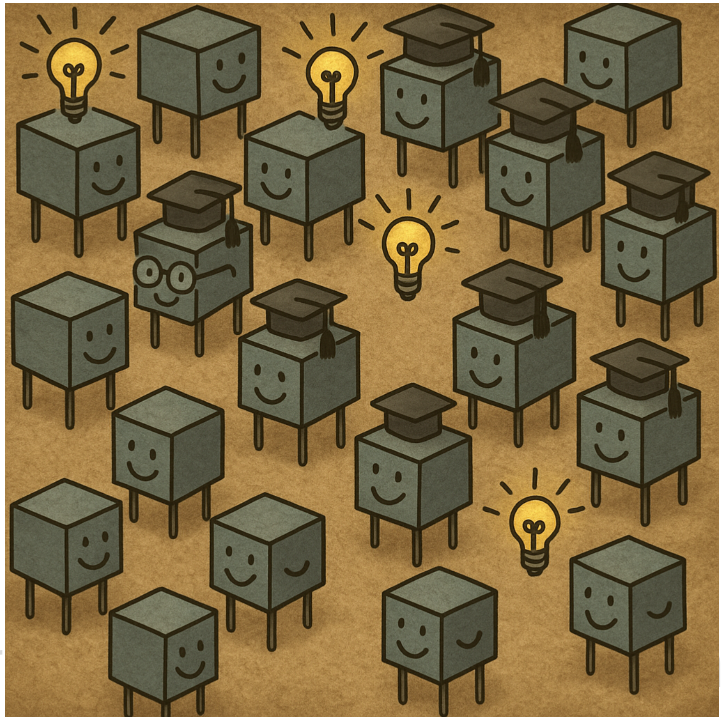


Transistor scaling is hitting walls

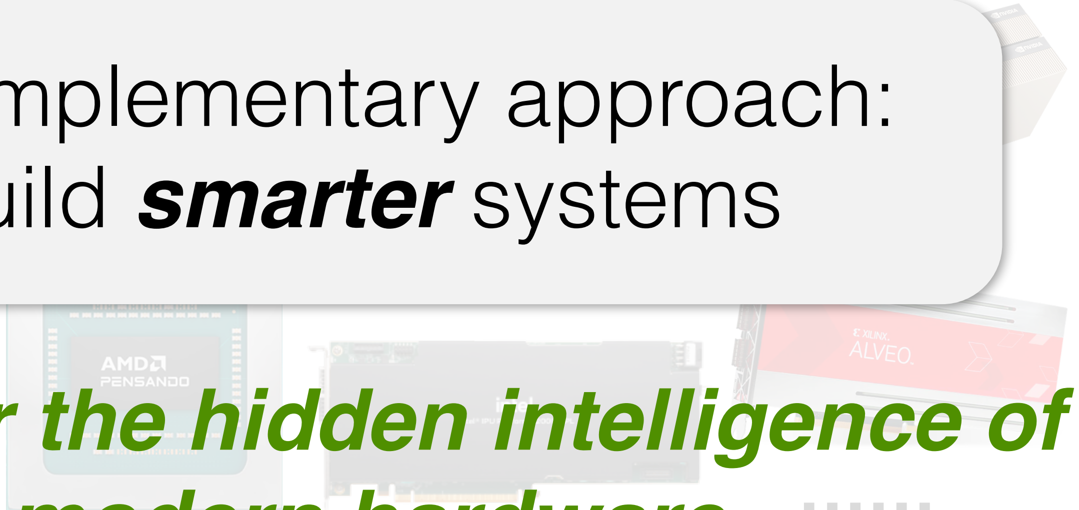


Rise of domain-specific accelerators

…something I am excited about



A complementary approach:
build ***smarter*** systems



Uncover the hidden intelligence of modern hardware

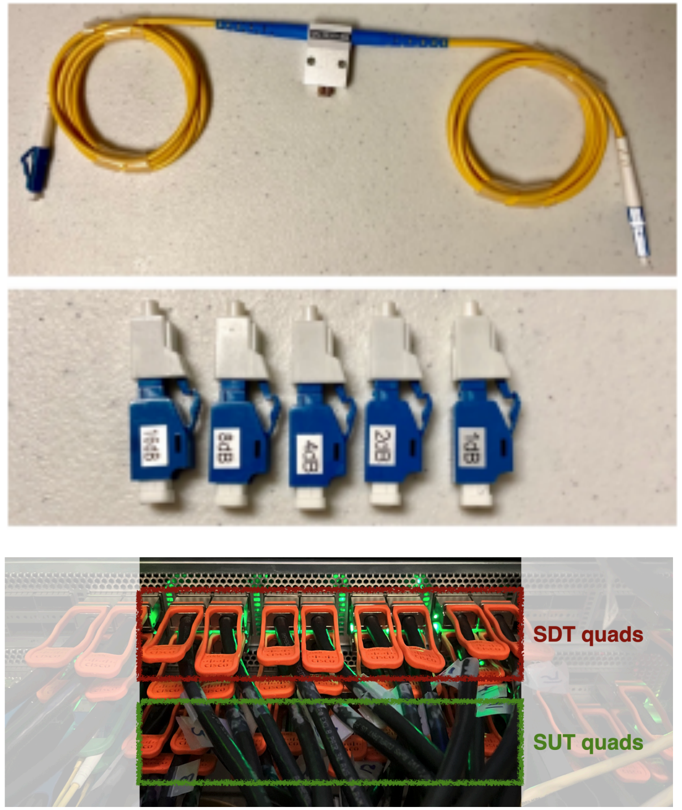
……

Tr g walls

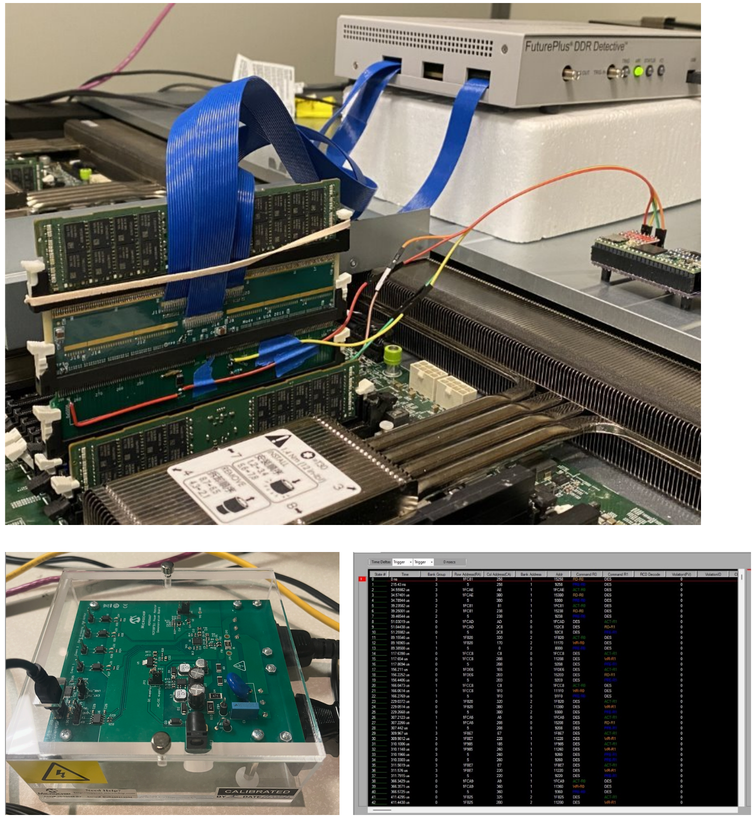
Rise of dom ecific accelerators

…today!

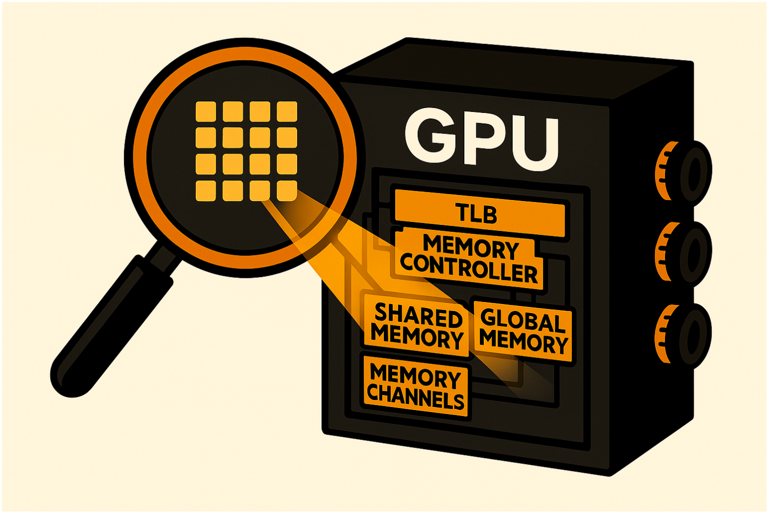
Uncover the hidden intelligence of modern hardware



Networking ASICs (programmable switches and SmartNICs)



Memory in general compute servers (memory controller, DRAMs…)



GPU memory subsystem (GDDR, HBM…)

Q & A