Designing Efficient Distributed Systems Primitives by Exploiting Data Center Network Characteristics

Liangcheng (LC) Yu

Researcher@Microsoft Research

Guest lecture, CS 134: Distributed Systems
May 22, 2025

Microsoft Research

About me

I am a researcher at Microsoft Research Redmond

I work on computer systems and networking

…with a focus on improving the efficiency of modern cloud networks

This course is about distributed systems…

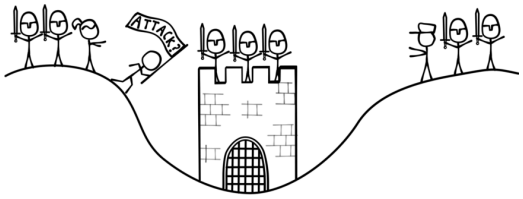
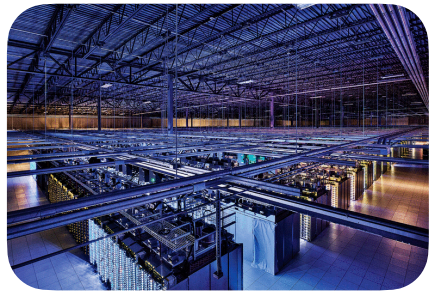


Image source: https://haydenjames.io/the-two-generals-problem/

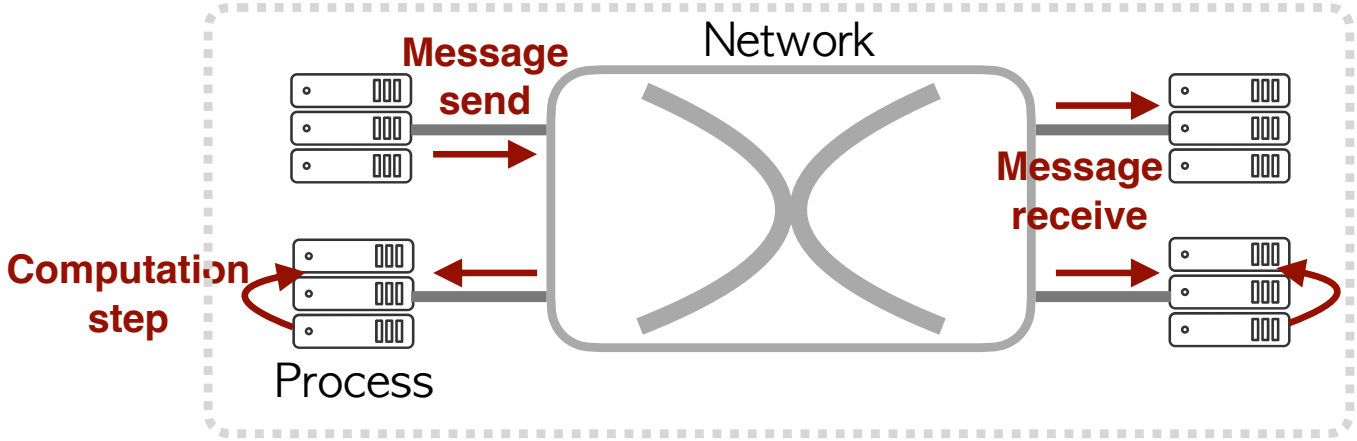
Classic distributed systems



Cloud data centers

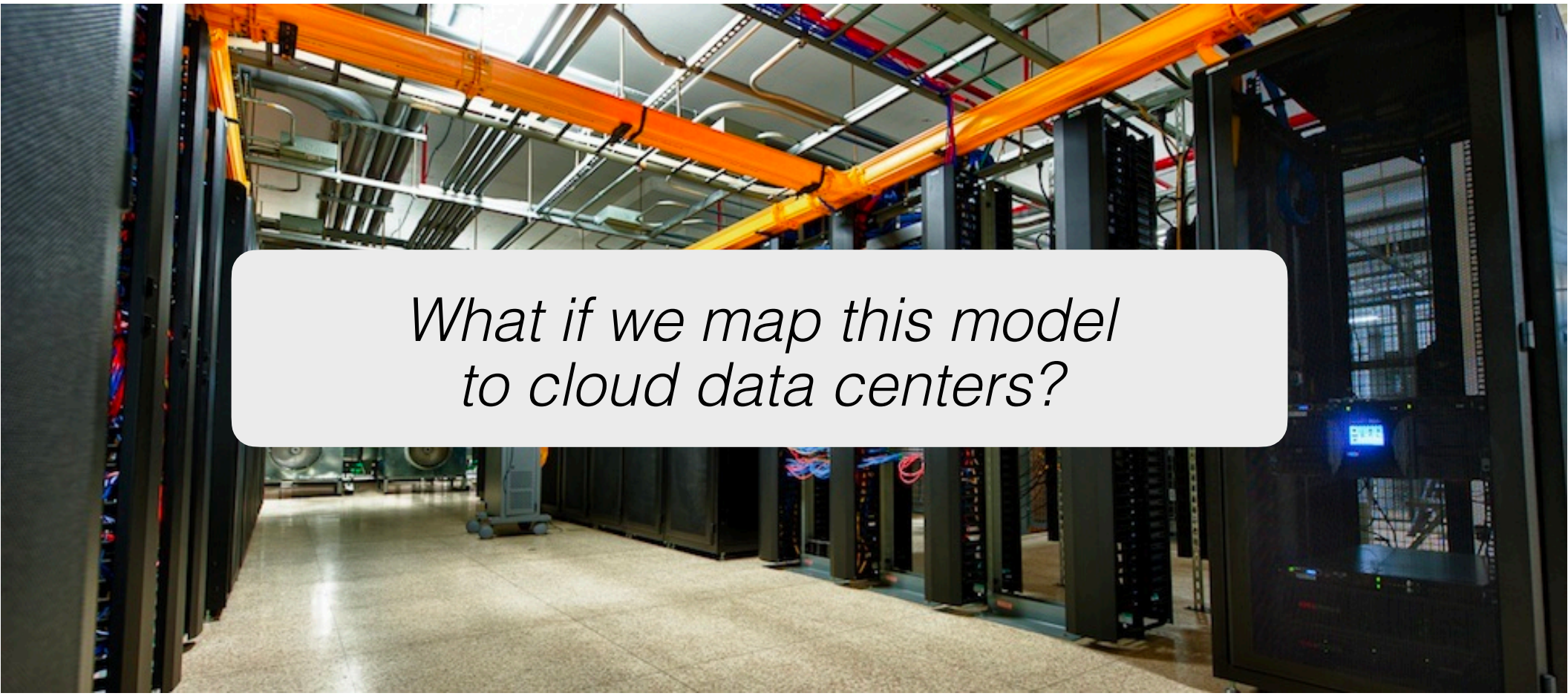
What concepts come to mind when you think about distributed systems?

A conceptual model of distributed systems



*A distributed system is a collection of autonomous computing elements that appears to its users as **a single coherent system**.*

—Maarten van Steen and Andrew S. Tanenbaum


What if we map this model
to cloud data centers?

What about cloud data centers?

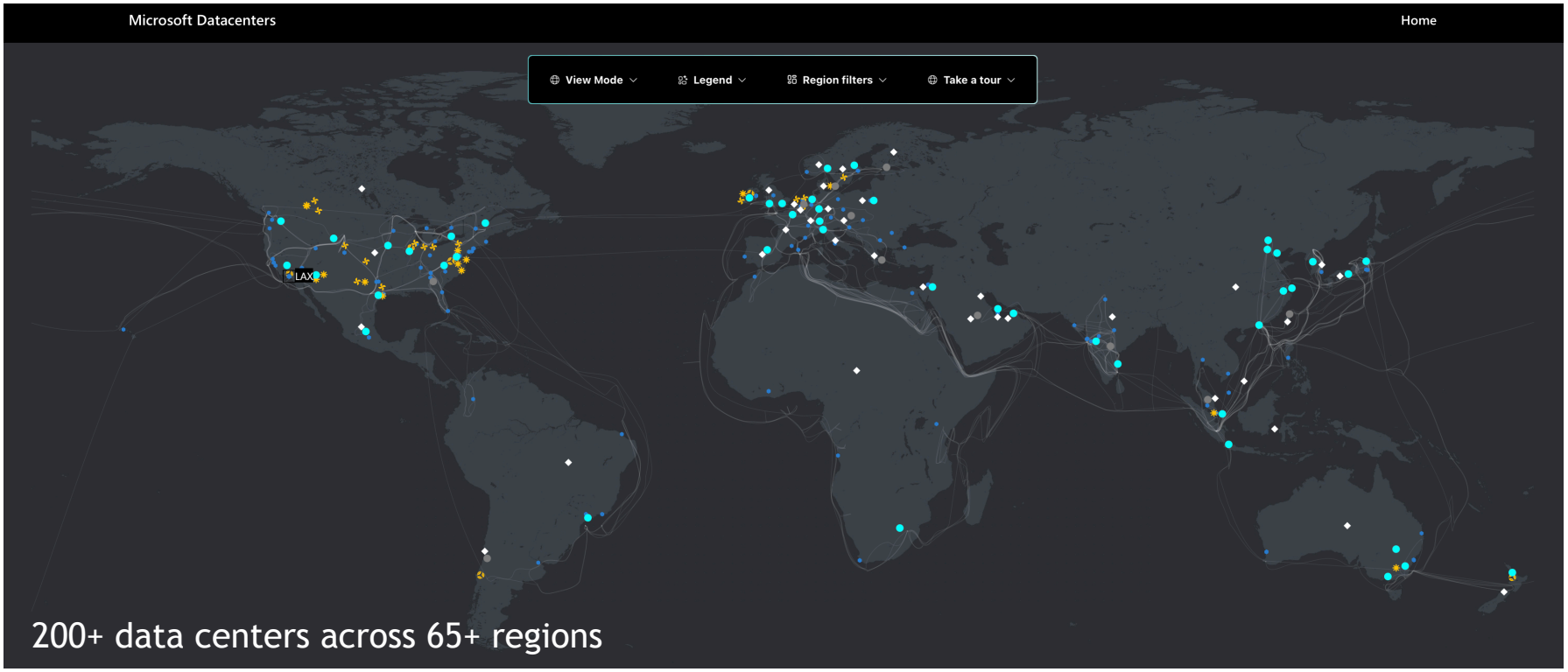


Image source: https://datacenters.microsoft.com/globe/explore

What about cloud data centers?

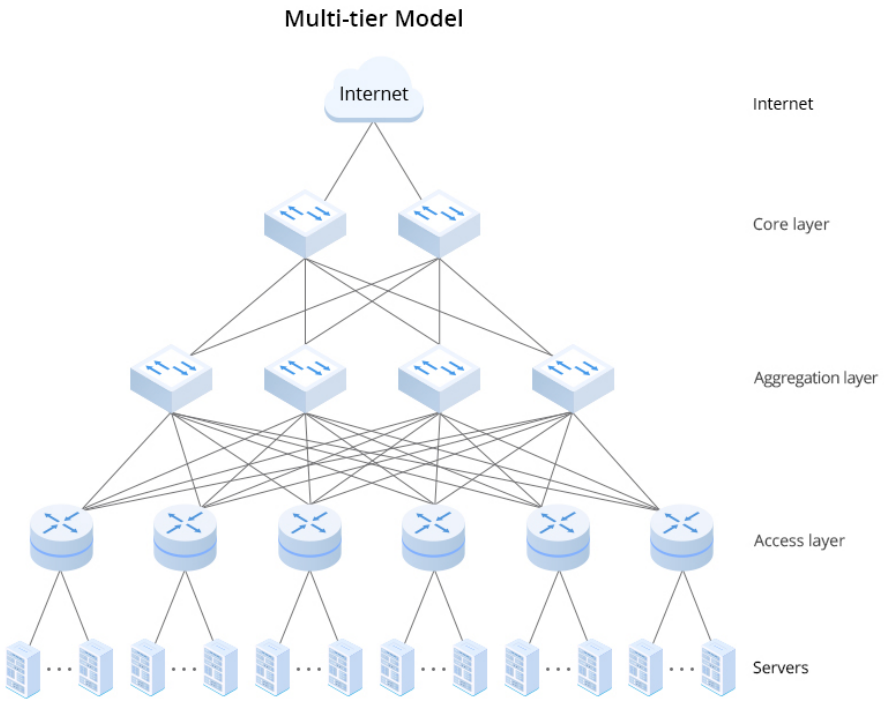
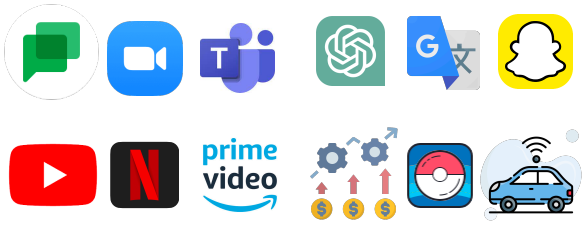


Image source: https://www.fs.com/blog/what-is-data-center-architecture-2929.html

What about cloud data centers?

Emerging application requirements



Massive-scale data centers

How to design distributed systems primitives, efficiently?

Data centers are not arbitrary systems!

- Regularities in network topology
- Emerging hardware capabilities
- Specific application requirements
- …

…exploit data center characteristics and rethink the classic design principles!

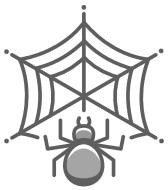
Case studies

Beaver *(OSDI 2024)*
Practical Partial Snapshots for Distributed Cloud Services

Distributed snapshots

Cuttlefish *(WIP)*
Cuttlefish: A Fair, Predictable Execution Environment for Cloud-hosted Financial Exchange

Synchronous coordination



OrbWeaver *(NSDI 2022)*
Using IDLE Cycles in Programmable Networks for Opportunistic Coordination

Failure detection

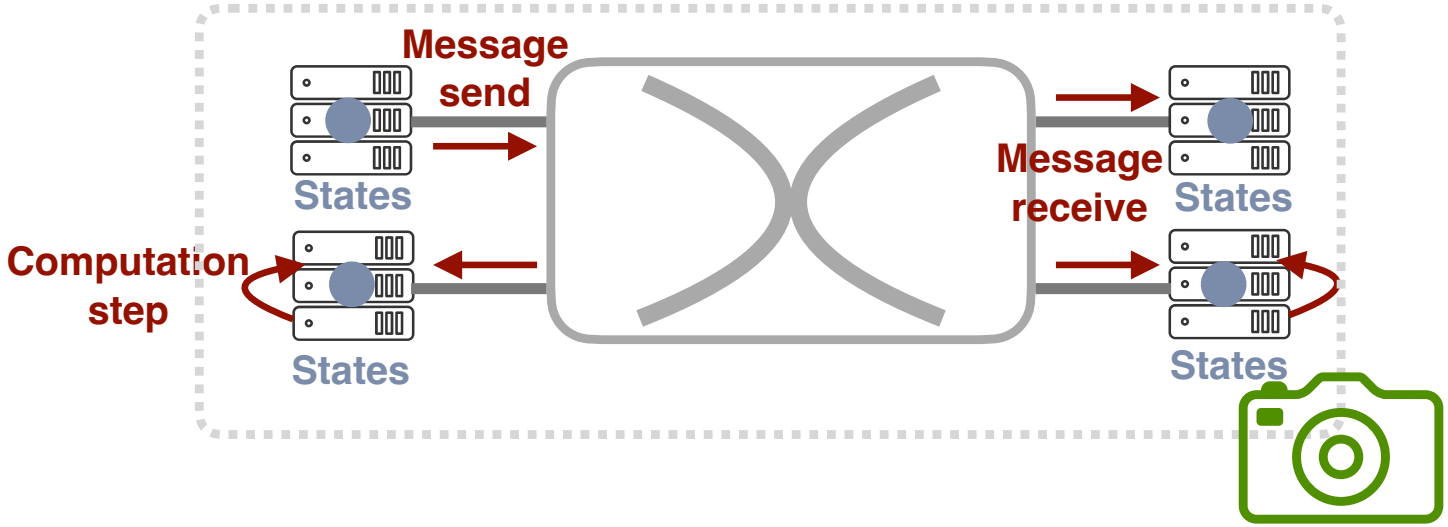
Beaver:

Practical Partial Snapshots for Distributed Cloud Services

Liangcheng (LC) Yu, Xiao Zhang, Haoran Zhang, John Sonchack, Dan R. K. Ports, and Vincent Liu

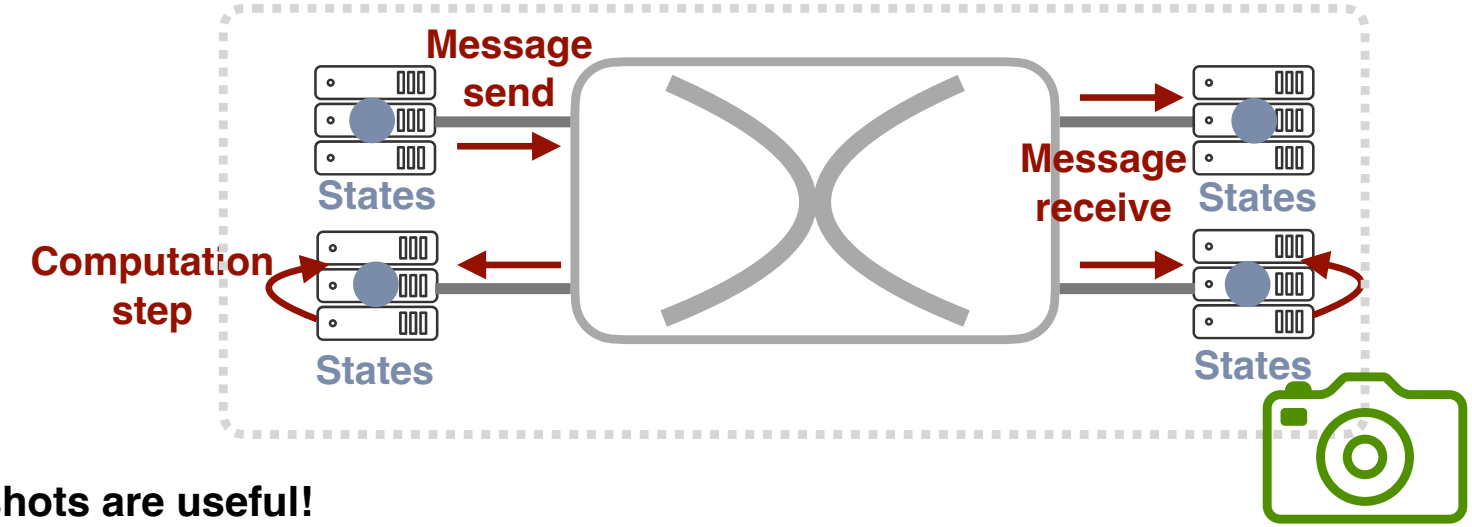
Let's talk about snapshots

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

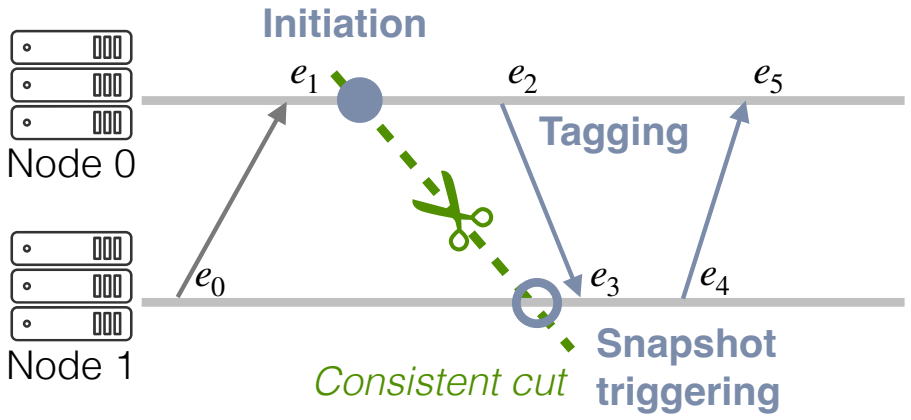
.....

Classic distributed snapshots

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

Classic distributed snapshots

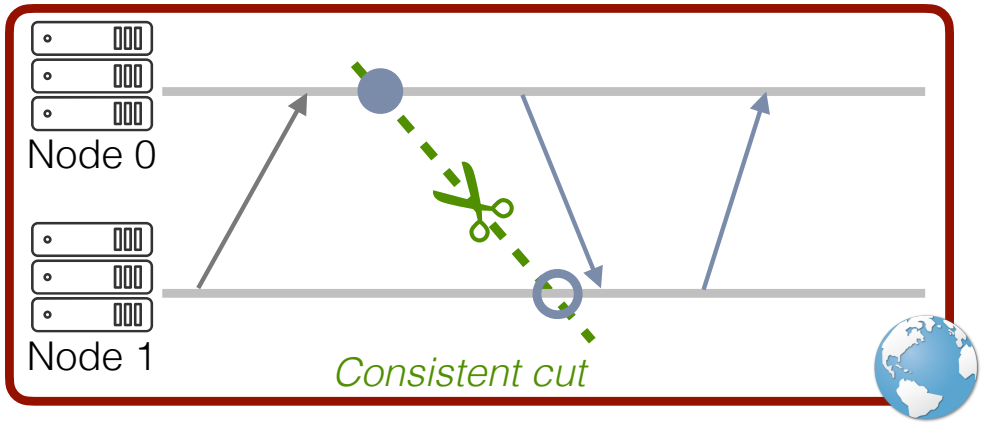
e.g., Chandy-Lamport (TOCS 1985)



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

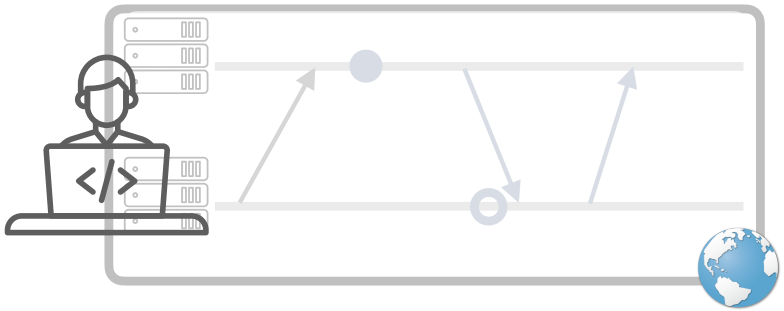


Utopian: isolated 'universe' of nodes

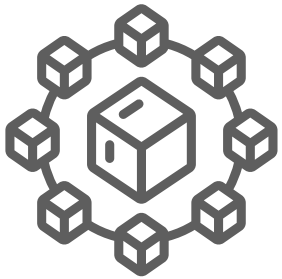
Fundamental assumption:
The set of participants are ***closed*** under causal propagation.

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

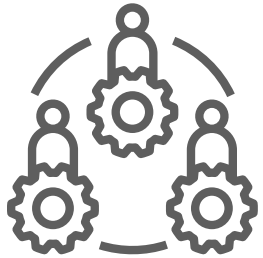
The assumption rarely matches reality!



Utopian: isolated 'universe' of nodes

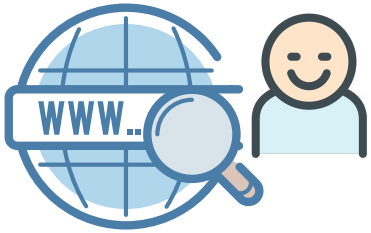


Modular services



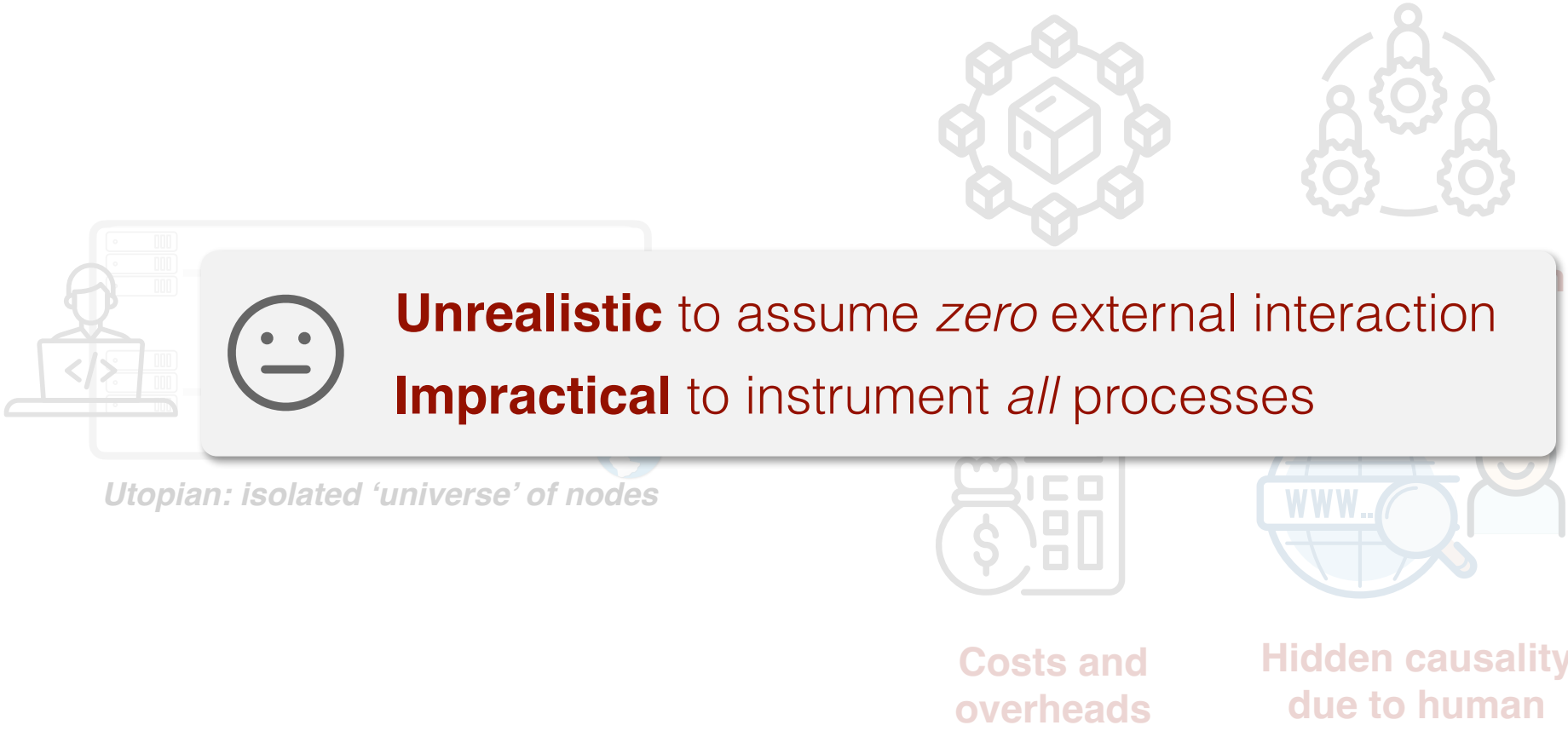
Instrumentation constraints

Costs and overheads



Hidden causality due to human

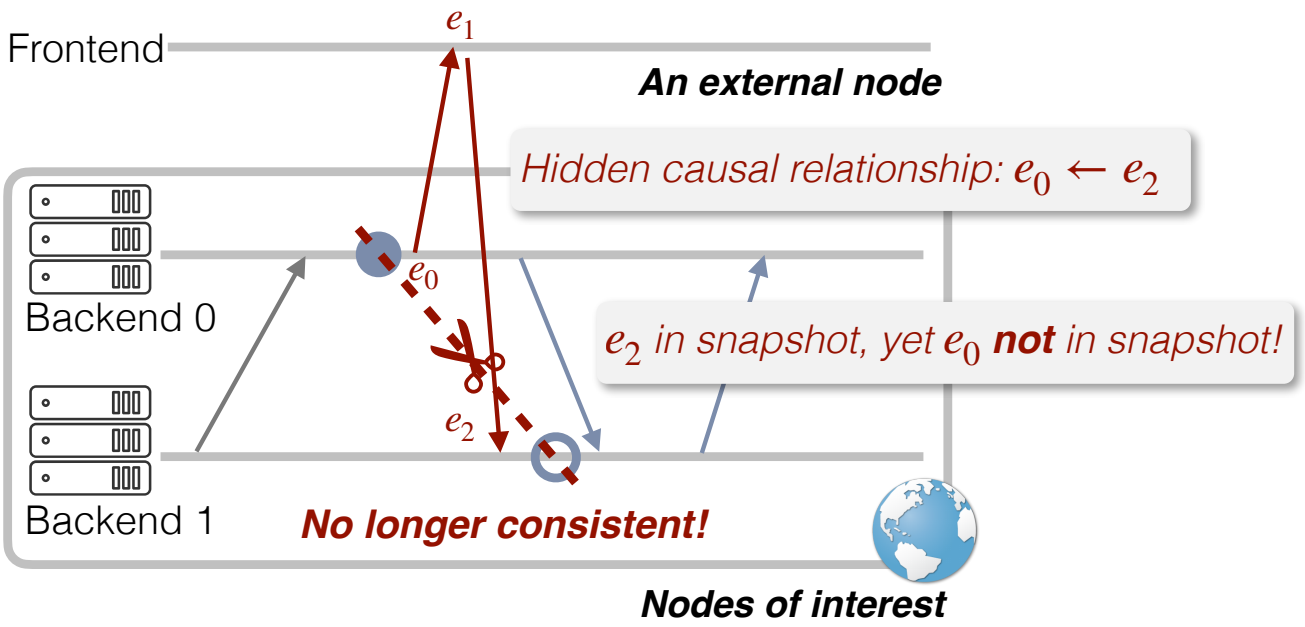
The assumption mismatches the reality!



Unrealistic to assume *zero* external interaction

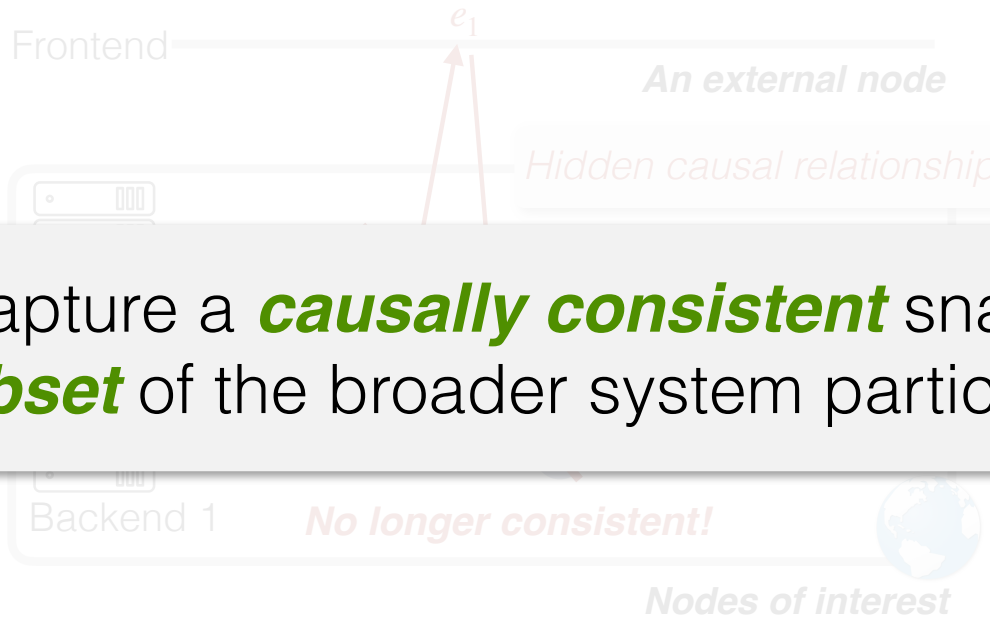
Impractical to instrument *all* processes

Consequences?



☹ *A single external node can break the guarantee!*

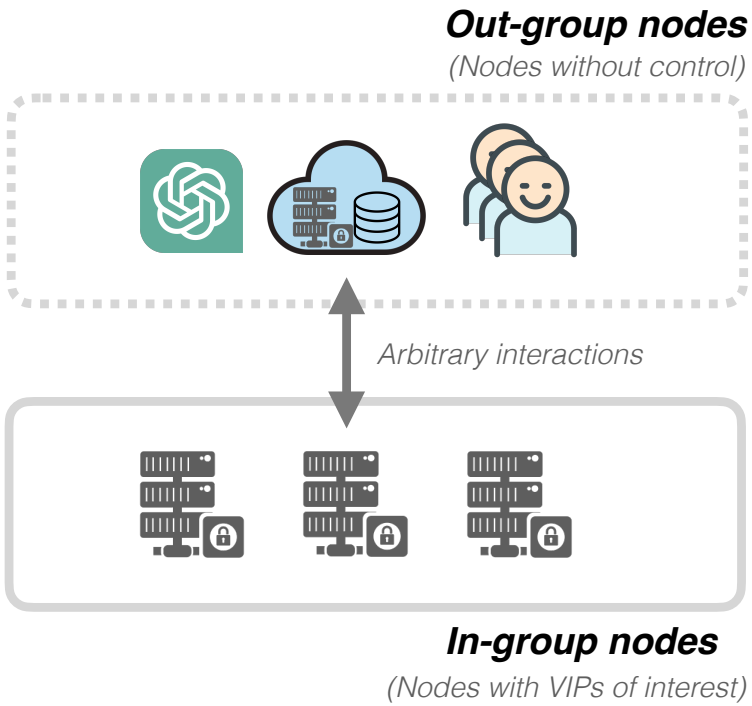
Consequences?



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)*

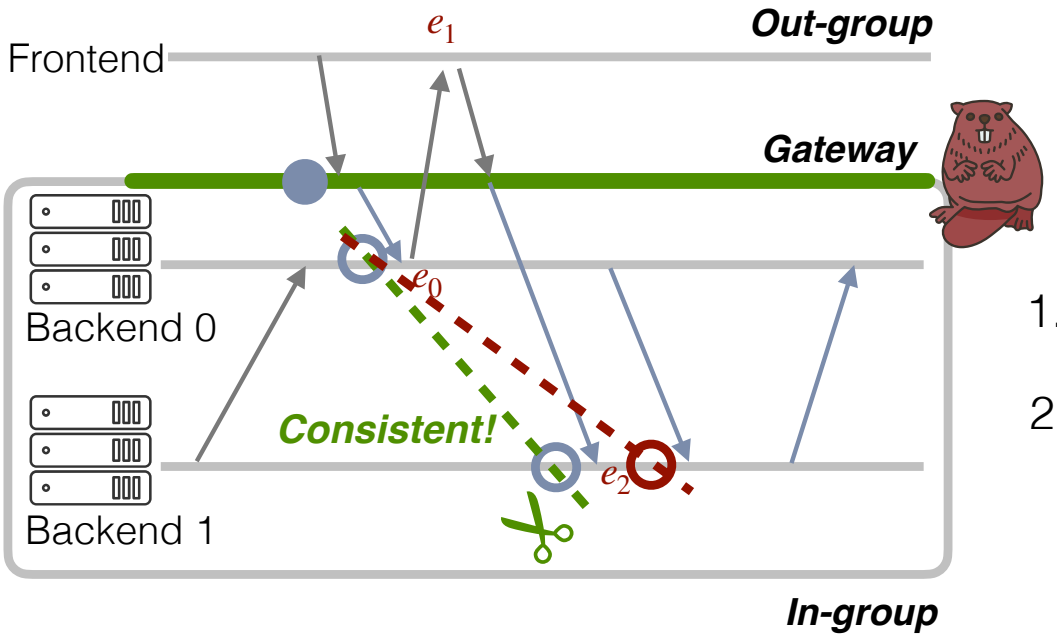
Zero impact over existing service traffic

*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!

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 $e_g^{ss} \rightarrow e' \rightarrow e^{out} \rightarrow e$.

We can focus on a subset of the above causality chain: $e_g^{ss} \rightarrow e$. From the properties of the in-group snapshot protocol, $e_g^{ss} \rightarrow e$ 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?

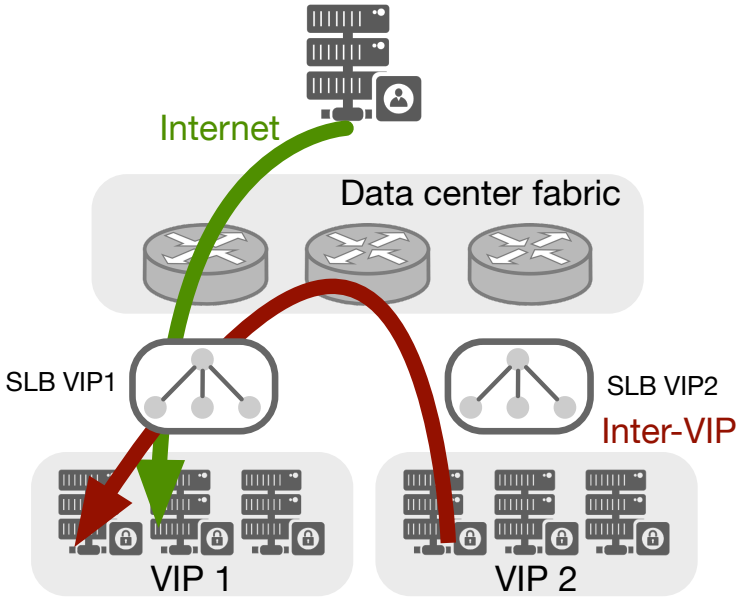
Challenge 1 How to instantiate GW?

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)

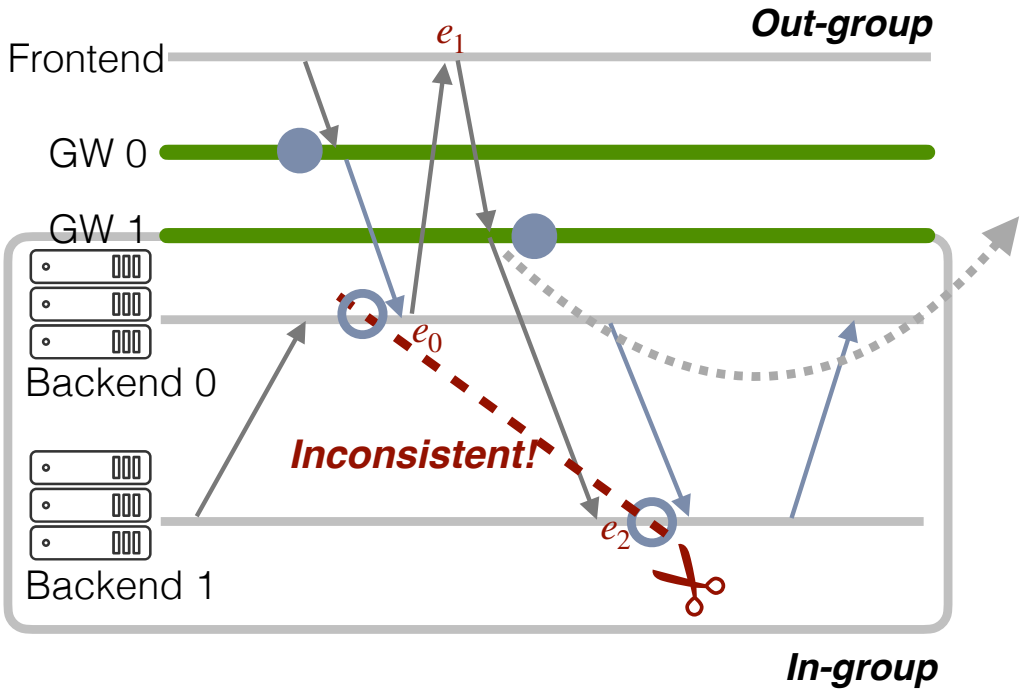
How to enforce MGM practically in today's network?

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

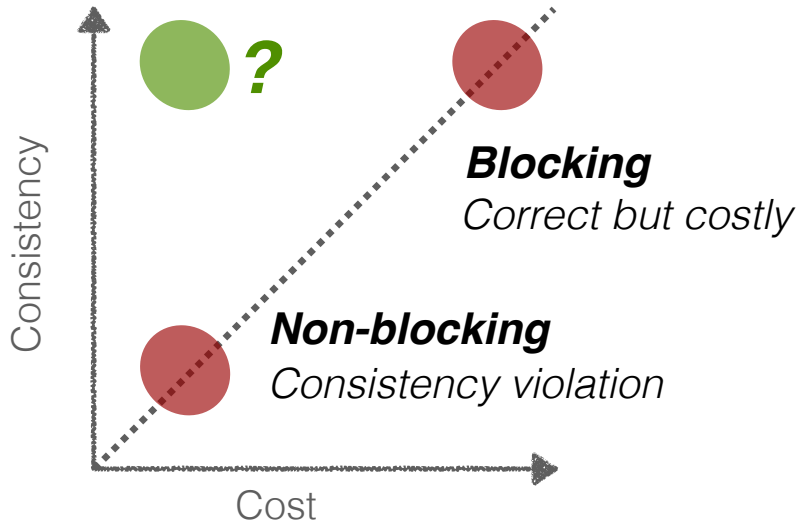


GW 1 hasn't initiated the new snapshot mode to mark it, triggering the ***violation***

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?



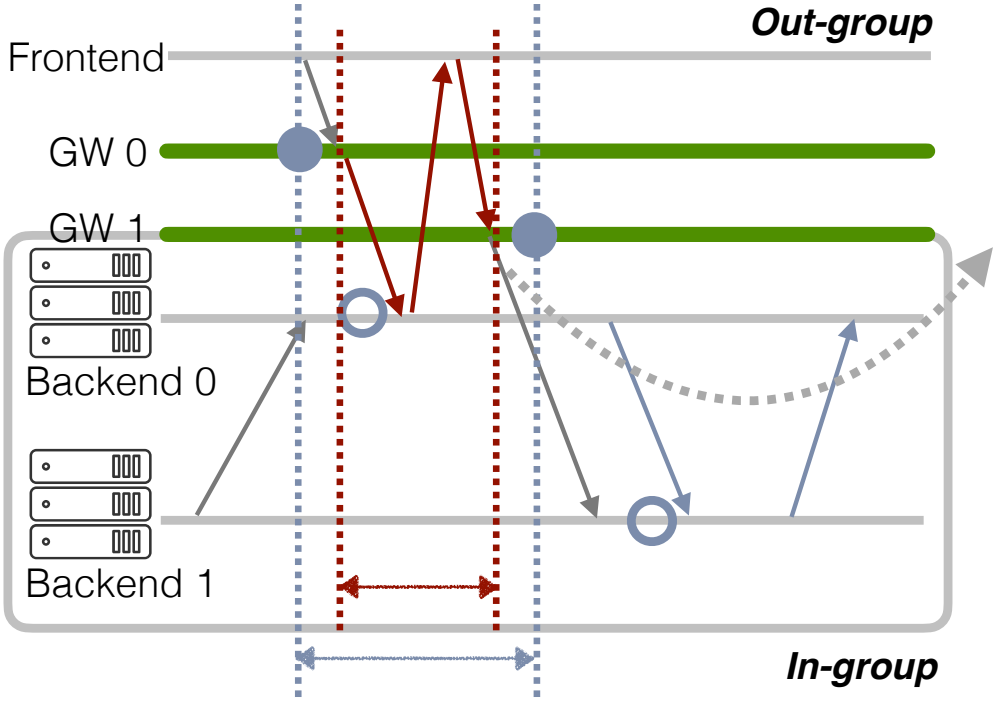
*Can we get both **consistency** and **zero cost**?*

Optimistic Gateway Marking (OGM)

- *Intuition & formalism*
- *Mechanism*

Challenge 2: handling multiple SLBs

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



Time gap between SLB initiation points

Observation:

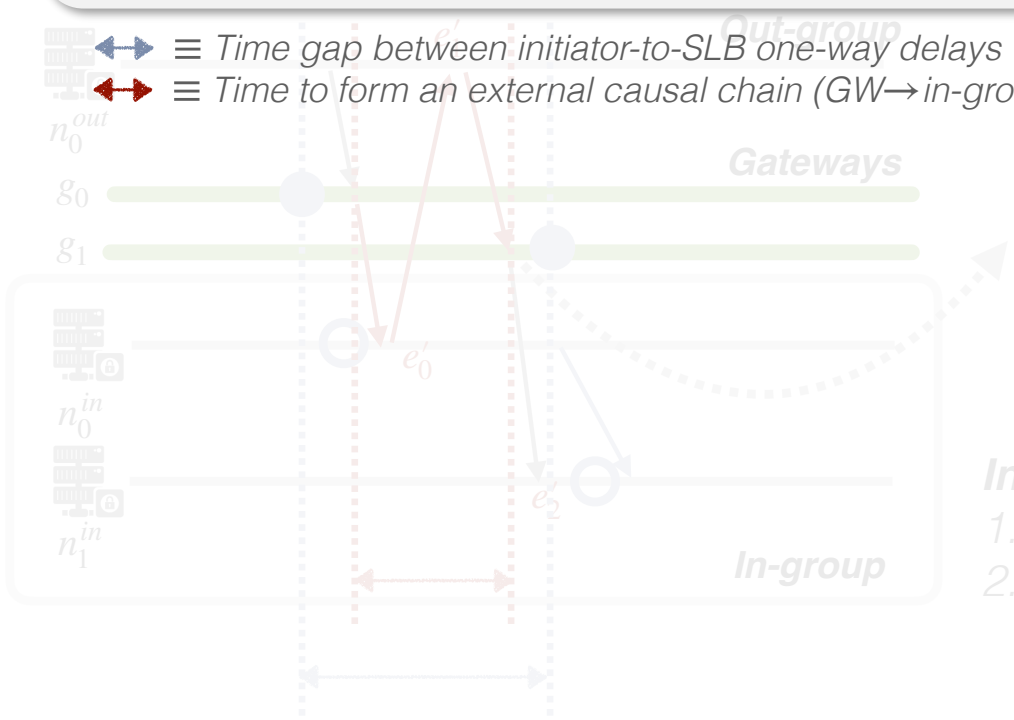
Causally relevant messages are rare!
GW→in-group→out-group→GW (external causal chain)

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

Theorem: if ↔ < ↔, 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)*



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

Observation:
Causally relevant n
GW→in-group→out
causal chain)

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

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*

Optimistic Gateway Marking (OGM)

→ *Optimistic execution in common cases*

→ *Verification/rejection of snapshots under worst cases*

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

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?

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

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*

More details about Beaver's protocol…

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

Implementation and evaluation

SLB-associated workflow

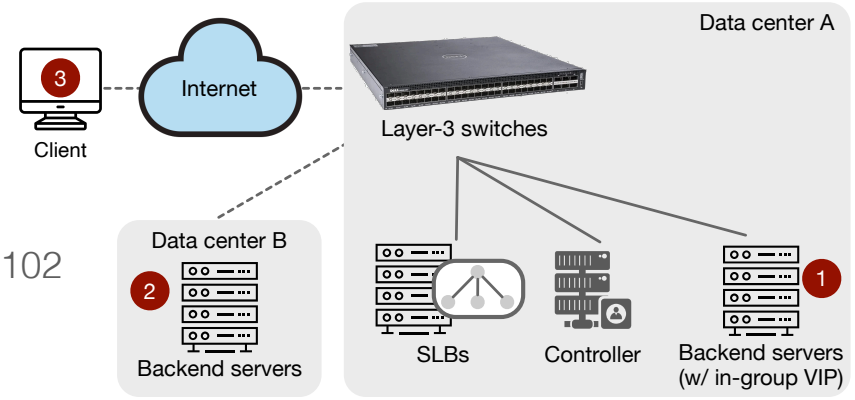
- Layer-3 ECMP forwarding per service VIPs: DELL EMC PowerSwitch S4048-ON
- Core SLB functions in DPDK: ~1860 LoC
- Backend server functions in XDP and tc: ~1040 LoC

Beaver protocol integration

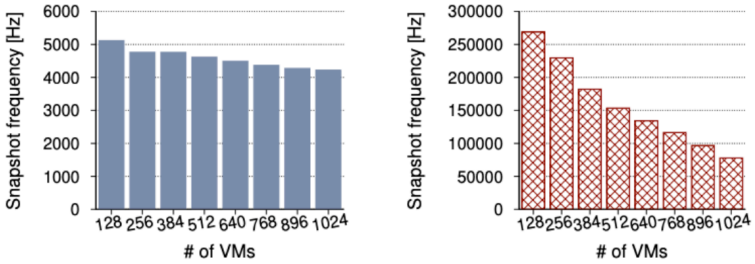
- Minimal logic: (1) 68 LoC for SLB DPDK data path logic (2) 102 LoC for eBPF at in-group VMs

Topology

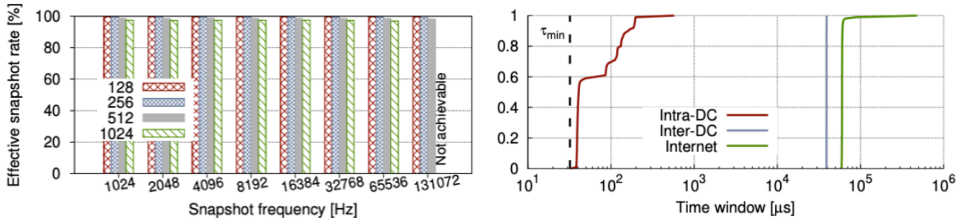
- Support typical communication patterns
- Possible out-group locations: within the same DC, DC at a different region, or on the Internet
- Scale up to 16 SLB servers and 1024 backend applications



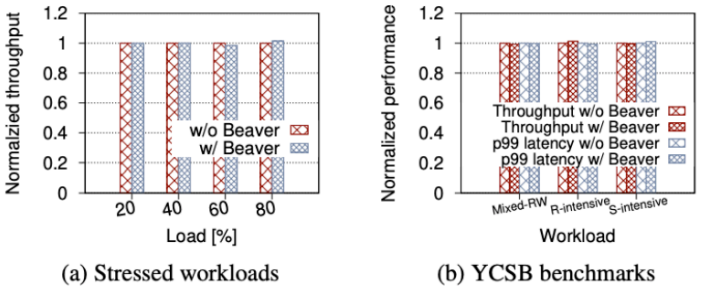
Details in the paper…



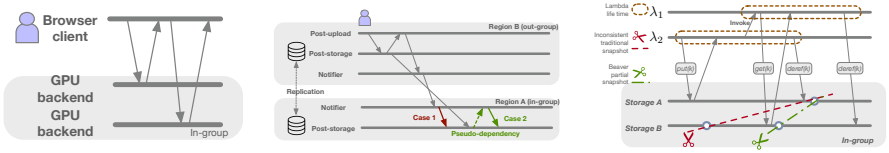
Beaver supports fast snapshot rates



Beaver rejects snapshots infrequently

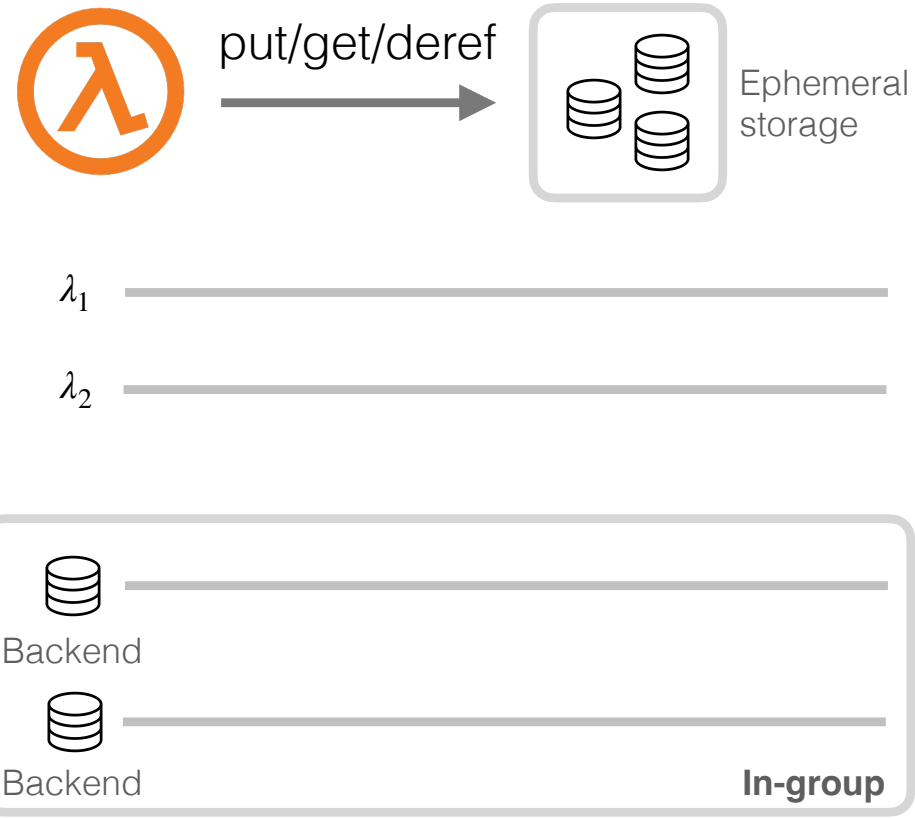


Beaver incurs zero impact

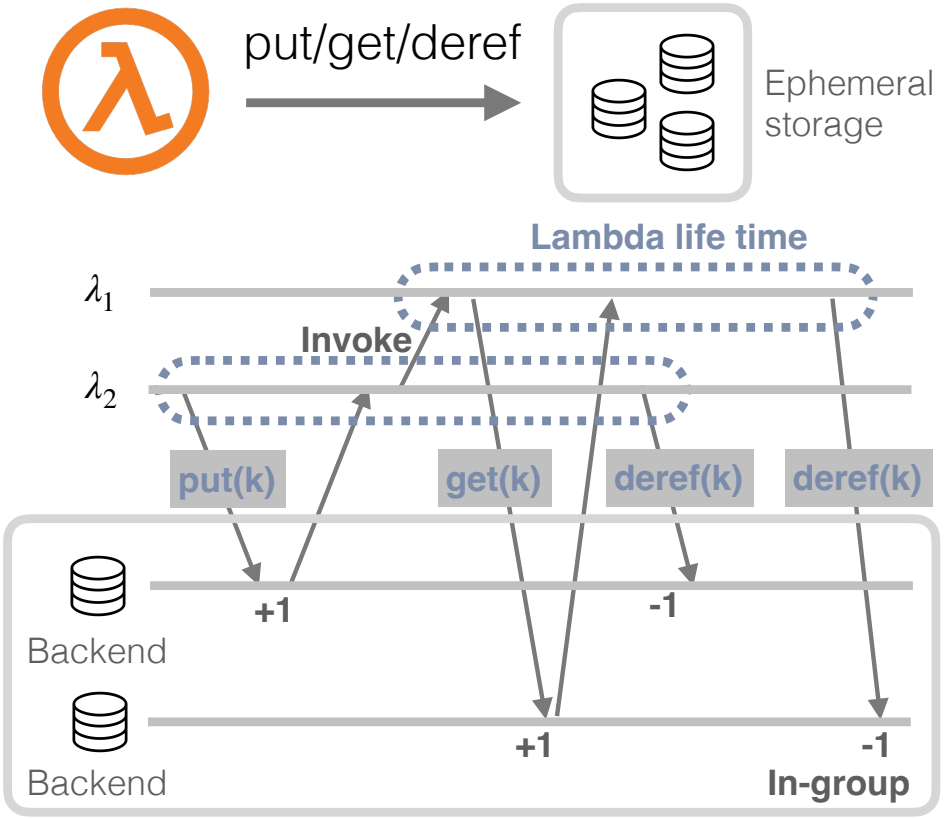


Use cases: integration testing, service analytics, deadlock detection, garbage collection…

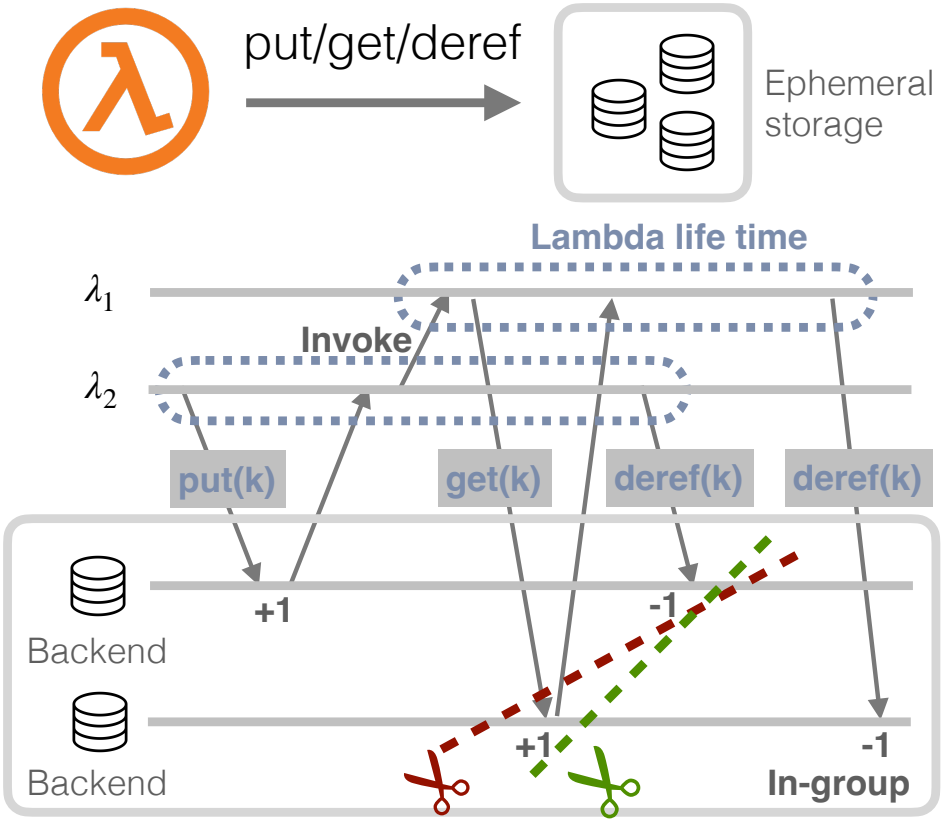
Example: garbage collection for ephemeral storage



Example: garbage collection for ephemeral storage



Example: garbage collection for ephemeral storage

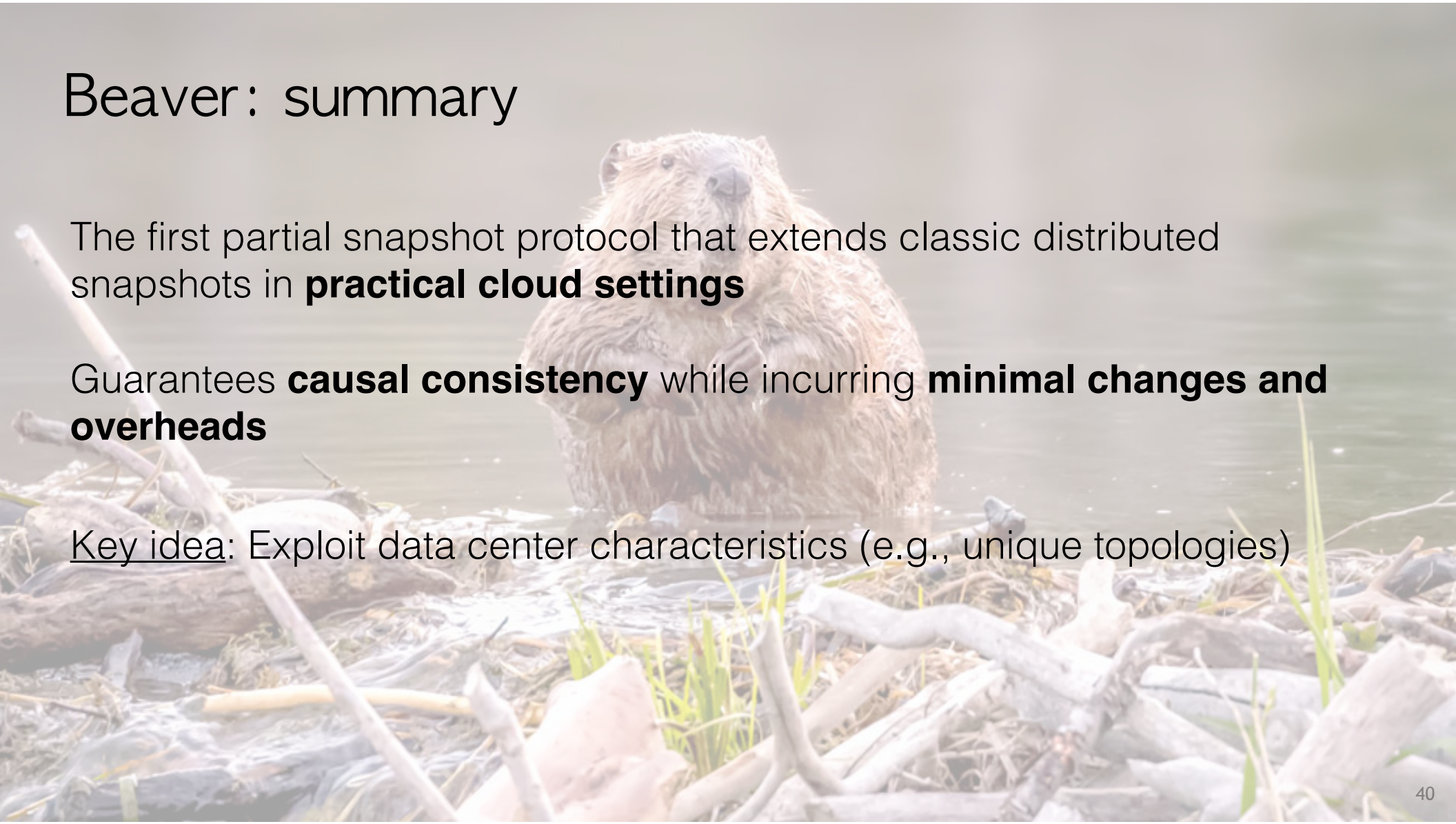


Strawman

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)

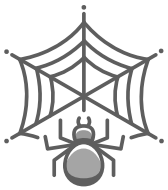
Case studies

Beaver *(OSDI 2024)*
Practical Partial Snapshots for Distributed Cloud Services

Distributed snapshots

Cuttlefish *(WIP)*
Cuttlefish: A Fair, Predictable Execution Environment for Cloud-hosted Financial Exchange

Synchronous coordination



OrbWeaver *(NSDI 2022)*
Using IDLE Cycles in Programmable Networks for Opportunistic Coordination

Failure detection

Cuttlefish: A Fair, Predictable Execution Environment for Cloud-hosted Financial Exchange

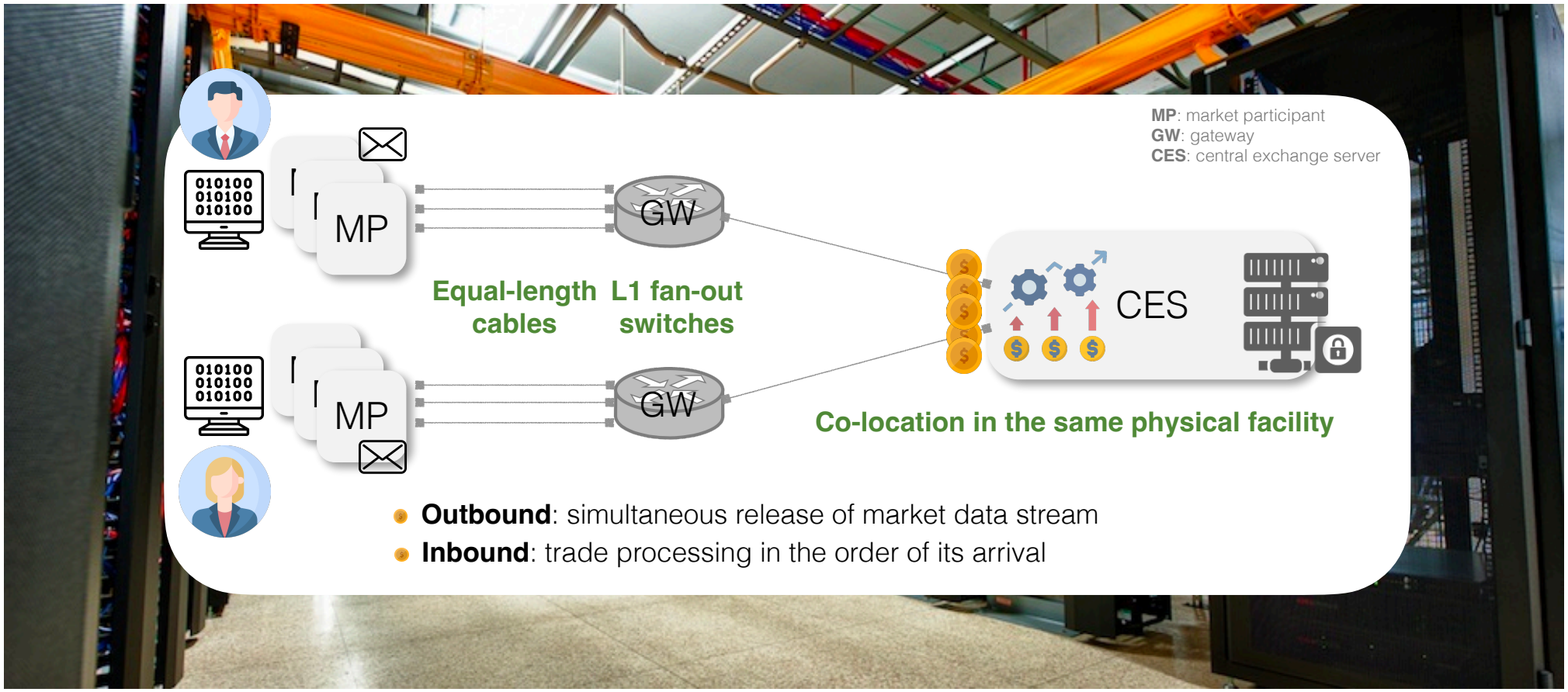
Liangcheng (LC) Yu, Pradesh Goyal, Ilias Marinos, and Vincent Liu

Microsoft Research

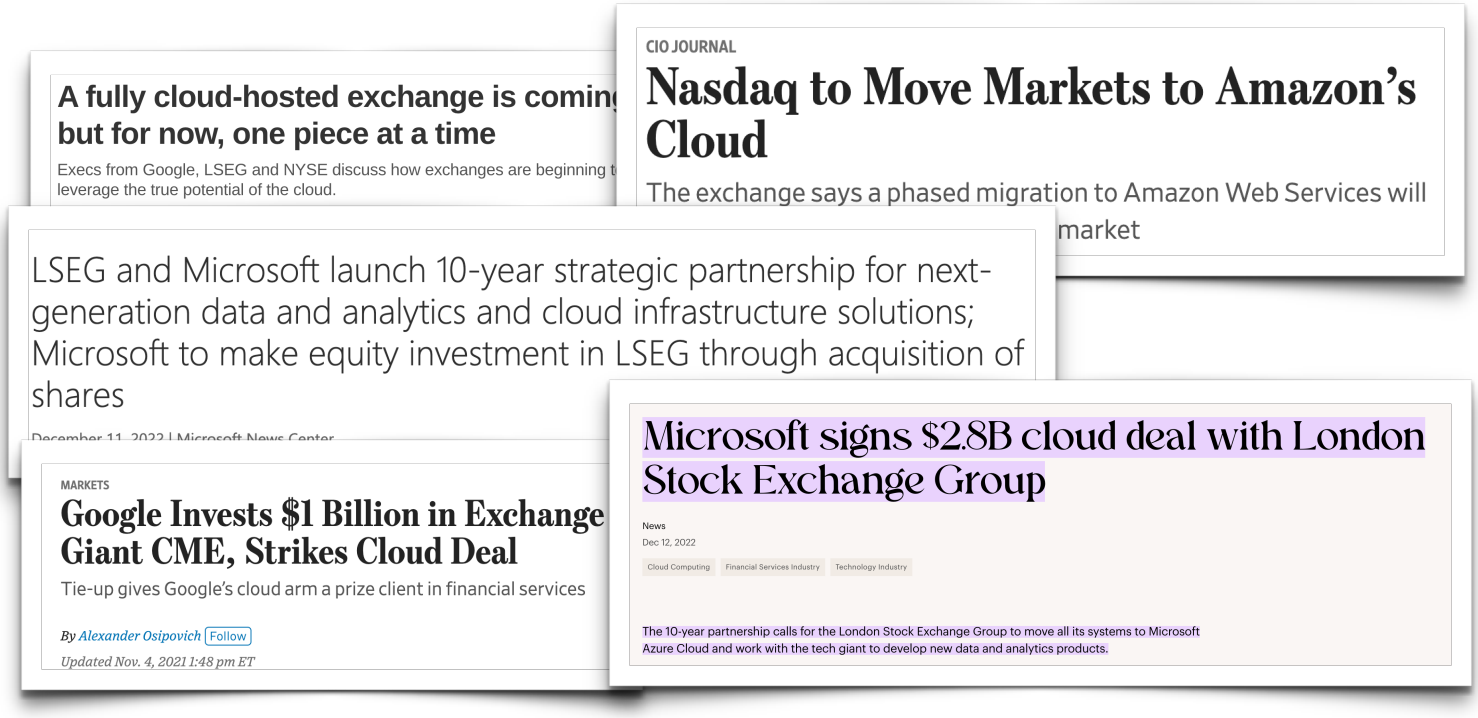
NVIDIA

Penn
University of Pennsylvania

Fairness, in on-premise infrastructure

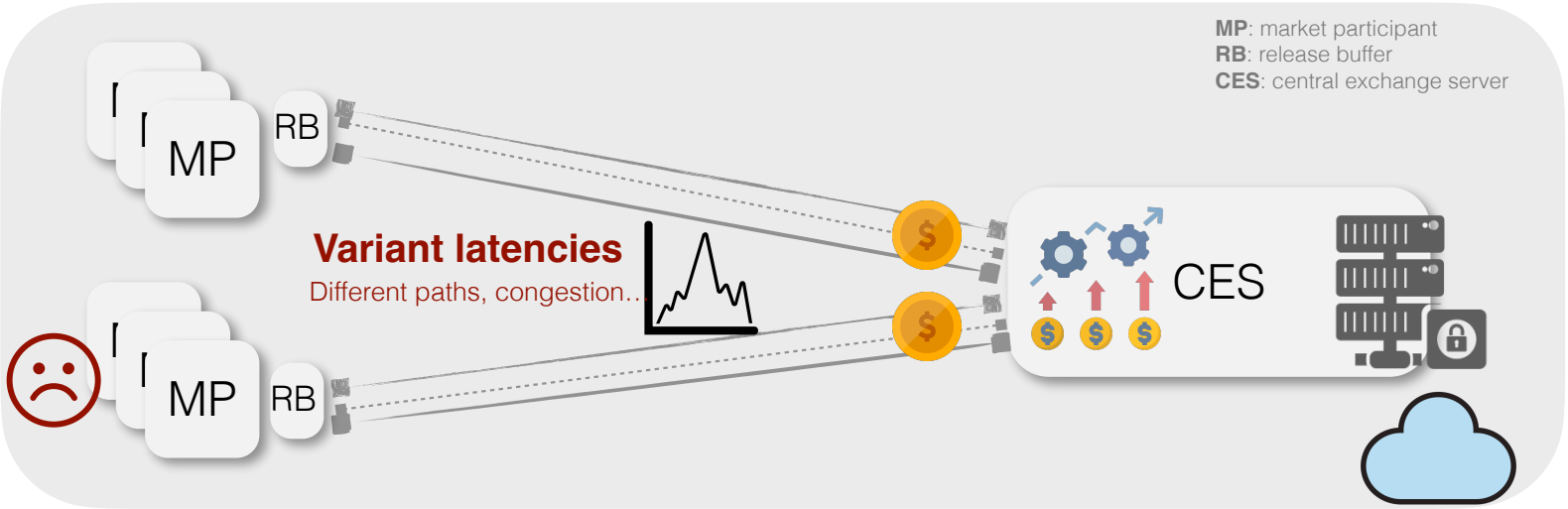


Rising interest in cloud-hosted exchange services



- System scalability and resource elasticity
- Cost reduction and ease of management
- Rise of remote work
- …

Fairness, in the cloud

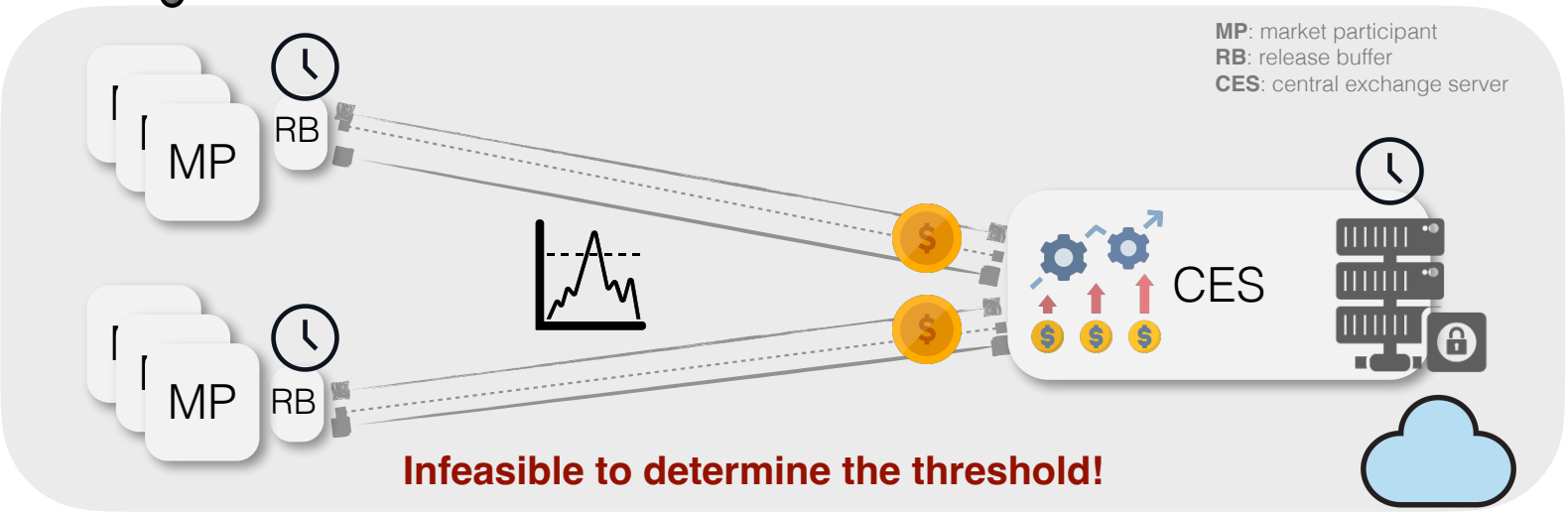


- **Outbound**: simultaneous release of market data streams
- **Inbound**: trade processing in the order of its arrival

Unfairness!

Fairness, in CloudEx〔HotOS '21〕

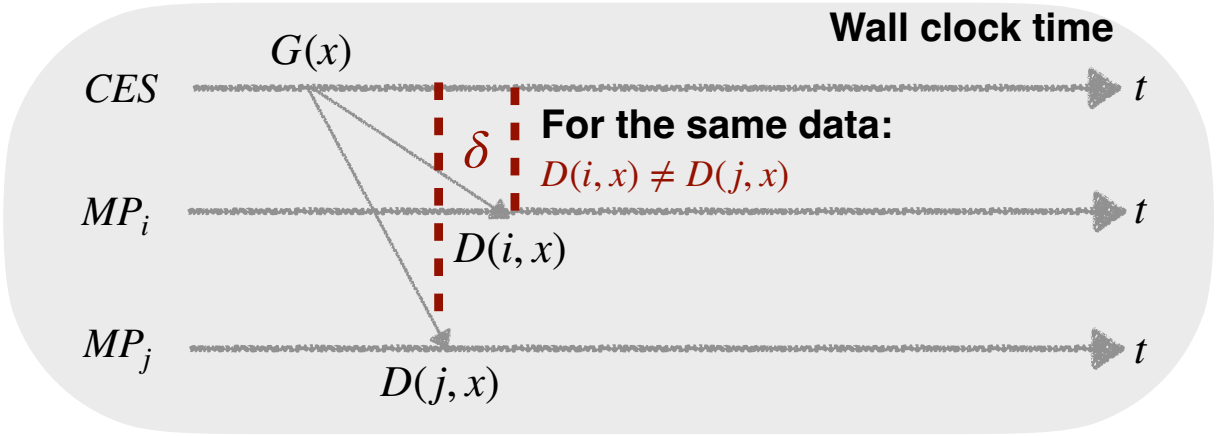
Idea: clock synchronization + message inhibition

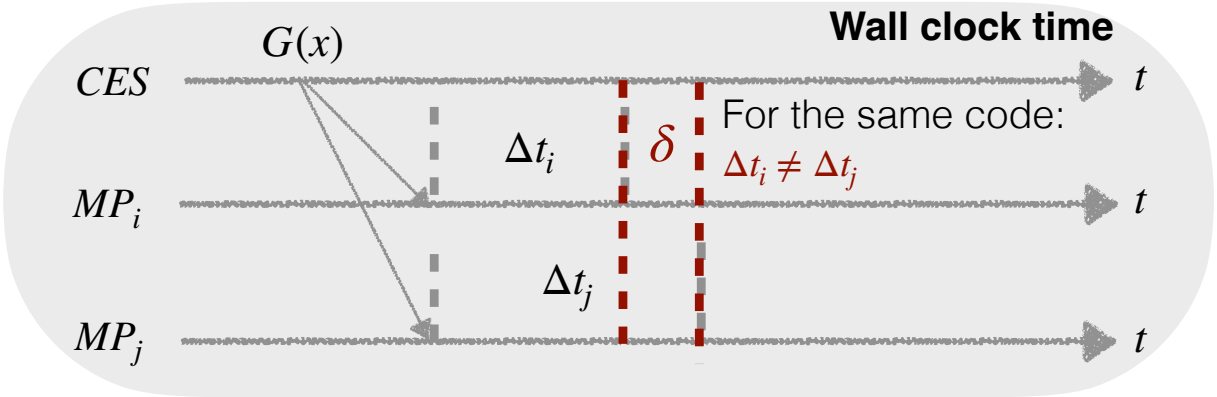


😐 Perfect clock synchronization is **hard**

☹ Latencies are **unpredictable** and **unbounded**

Let's reflect on underlying model today…





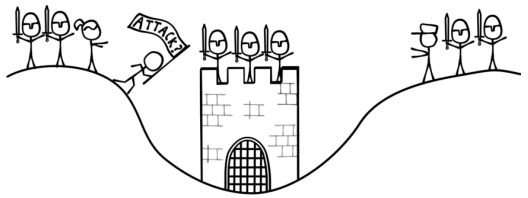
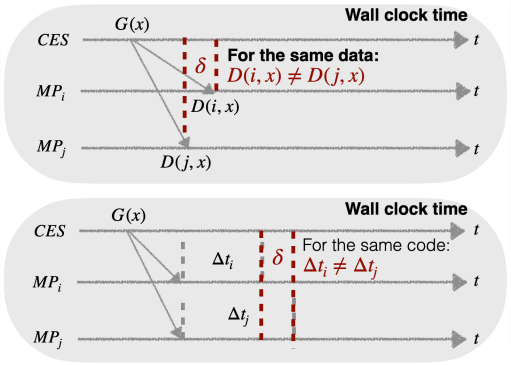


Image source: https://haydenjames.io/the-two-generals-problem/

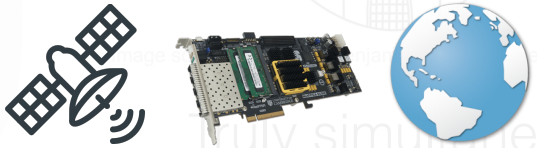
Truly simultaneous delivery is ***impossible***!

Computation can be ***non-deterministic*** at $O(\mu s)$
(thermal condition, resource utilization…)

Let's reflect on underlying model today…



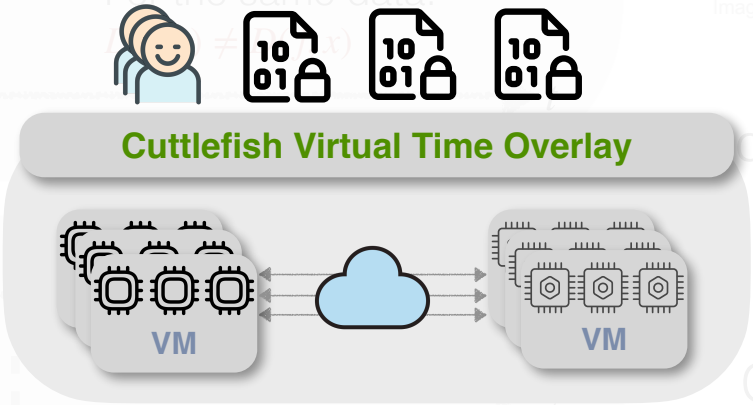
Communication and computation synchrony are ***challenging*** in real-time



Symptoms: trading ***arms race*** to gain δ advantage and increasing market ***consolidation***

Can we **guarantee fairness** via achieving **communication** and **computation synchrony**?

Cuttlefish: A Predictable Execution Environment



- Determinism w.r.t. underlying communication & computation
- Generality to trading patterns
- Democratized competition for special hardware

Cuttlefish outline

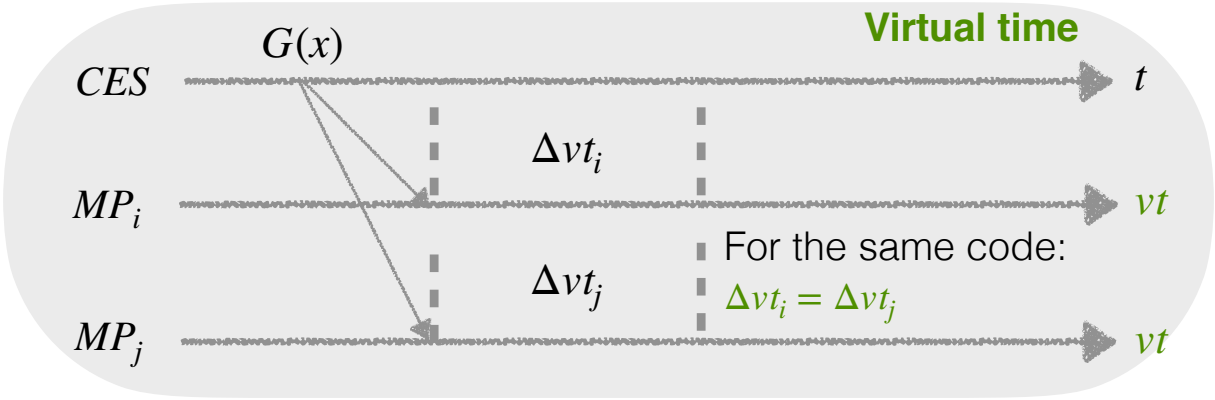
- Conceptual foundation
- User abstraction

} **This lecture**

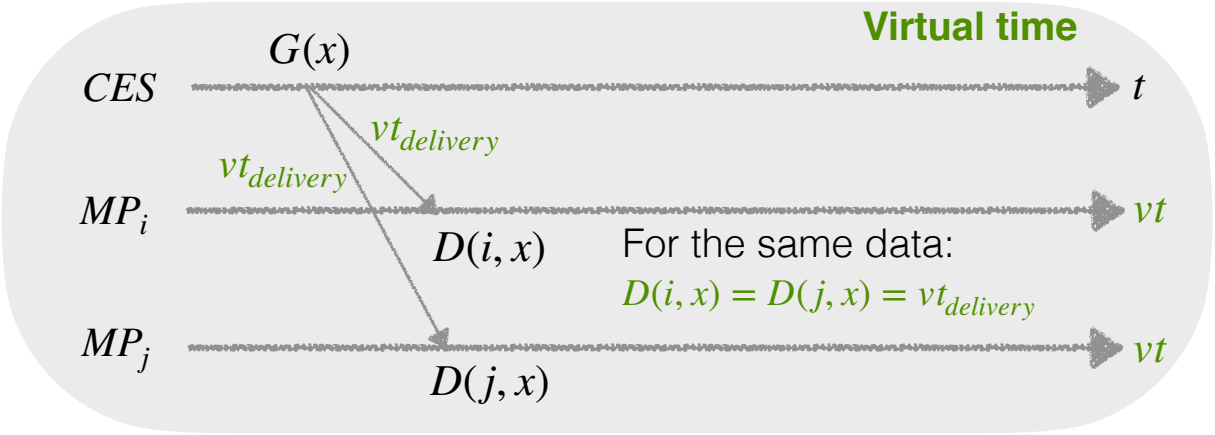
- Demo of the real system
- Implementation and benchmarks

Impossible? Imagine in virtual time domain …

Virtual time unit ≡ *some equal amount of work*



Quantizing vt per ***'actual amount of work'*** for computation synchrony

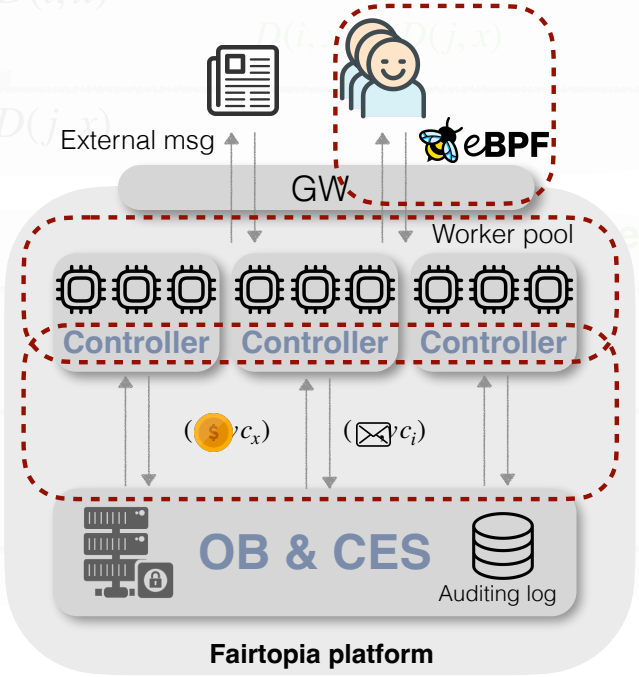


Freezing and ***advancing*** vt for communication synchrony

How to implement a real system?

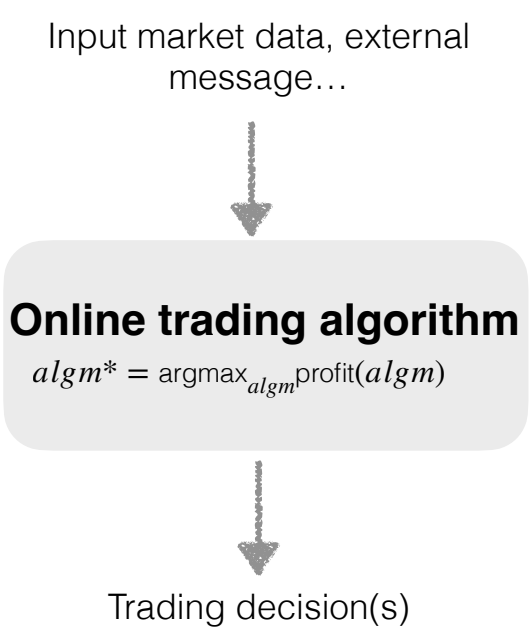
Instantiate *vt* as virtual cycles of a platform-agnostic IR/VM

Account and **control** the advancement of virtual cycles



1. **Programming interface**
2. **Virtual cycle tracking**
3. **Runtime execution**

User programming abstraction




```
#include <fairtopia_user.h>

int mu_handler(subscribed_context_t* data) {
    if ((*data) > 100) {
        // Sell
        trade_t trade = 1;
        submit_trade(&trade);
    } else if ((*data) < 10) {
        // Buy
        trade_t trade = 2;
        submit_trade(&trade);
    }
    map_update(0, &trade);
    return 0;
}
```

White-list set of extensible service APIs

Just-in-time trade submission

Narrow KV store API (e.g., lookup, update) for stateful invocations

The interface is expressive enough

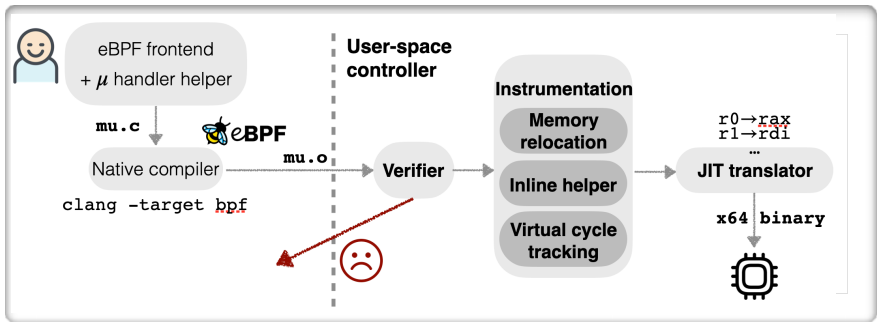
- Fibonacci, Bubble Sort…
- SMA Mean Reversion
- EMA Mean Reversion
- Relative Strength Index
- Moving Average Crossover Strategy
- Bollinger Bands Strategy

- Moving Average Convergence Divergence
- Multiple Moving Average Crossover Strategy
- Parabolic SAR
- On Balance Volume (OBV) + EMA
- Stochastic Oscillator
- Basic Market Making
- …

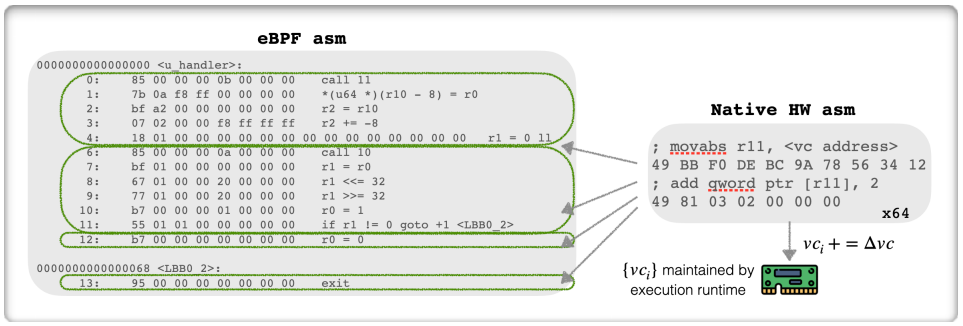
GPT-4

Running out-of-the-box

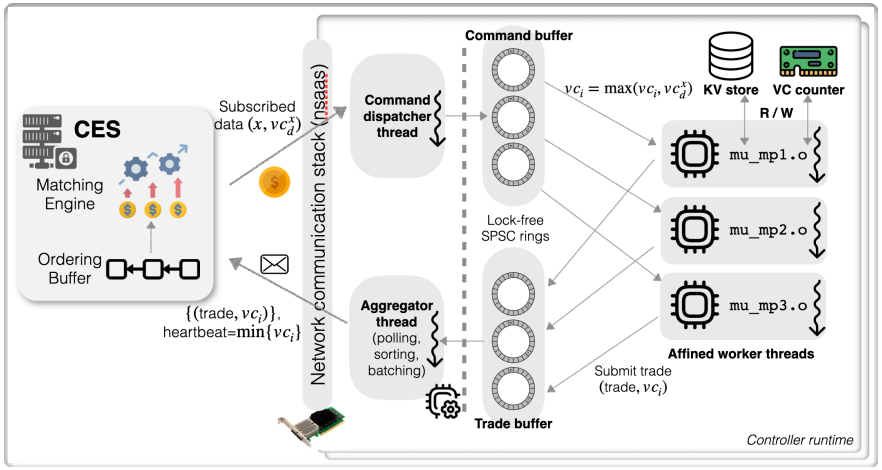
Implementation



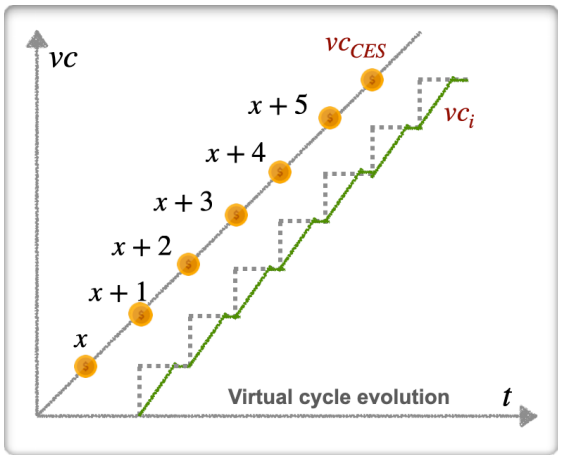
Program life time



Virtual cycle tracking instrumentation

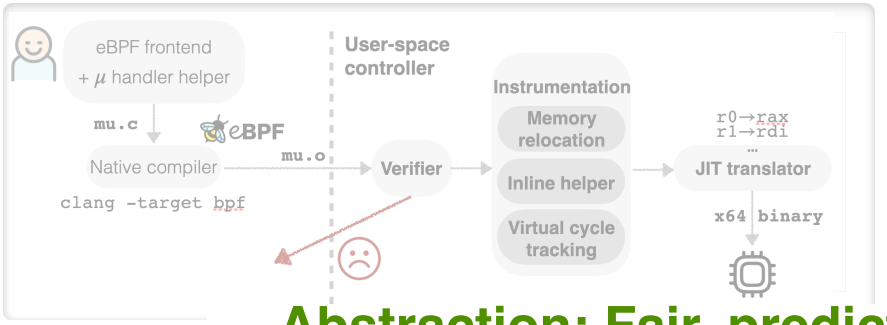


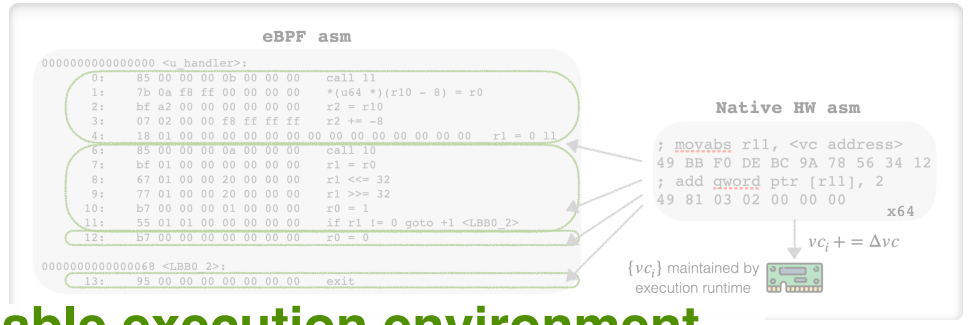
Runtime execution engine



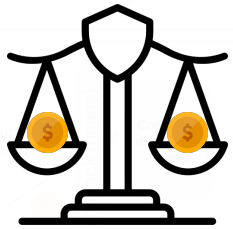
Virtual cycle assignment

Implementation



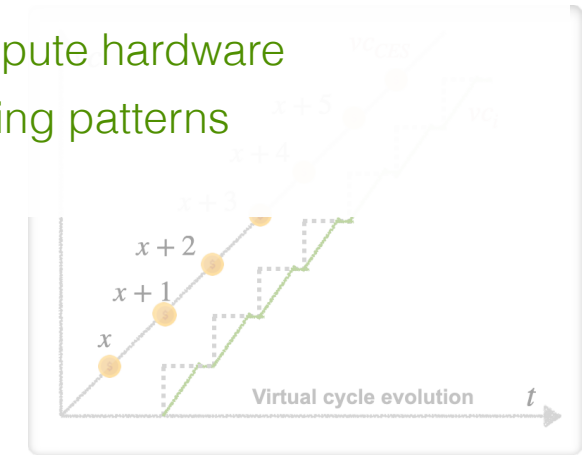


Abstraction: Fair, predictable execution environment



Guaranteed fairness w.r.t. heterogeneity in:

- Cloud network communication
- Compute hardware
- Trading patterns
- …



Runtime execution engine

Virtual cycle assignment

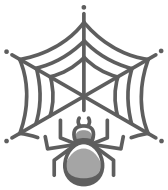
Case studies

Beaver *(OSDI 2024)*
Practical Partial Snapshots for Distributed Cloud Services

Distributed snapshots

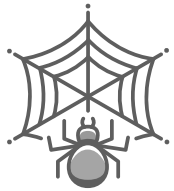
Cuttlefish *(WIP)*
Cuttlefish: A Fair, Predictable Execution Environment for Cloud-hosted Financial Exchange

Synchronous coordination



OrbWeaver *(NSDI 2022)*
Using IDLE Cycles in Programmable Networks for Opportunistic Coordination

Failure detection



OrbWeaver:

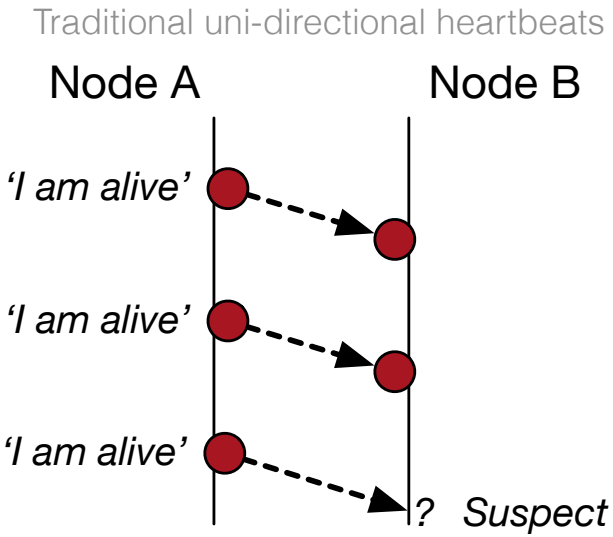
Using IDLE Cycles in Programmable Networks for Opportunistic Coordination

Liangcheng (LC) Yu, John Sonchack, and Vincent Liu

Penn
University of Pennsylvania

PRINCETON
UNIVERSITY

Example: failure detection



Common approach:

Periodic, high priority heartbeats

Fundamentally indistinguishable:
message drop or actual failure?

Empirically, use conservative detection thresholds

When introducing a distributed coordination function…

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

Time synchronization

Failure detector

Congestion notification

In-band telemetry

…

When introducing a distributed coordination function…

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

Time synchronization

clock-sync rate ↔ **clock precision**

Failure detector

Congestion notification

In-band telemetry

…

When introducing a distributed coordination function…

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

Time synchronization	**clock-sync rate ↔ clock precision**
Failure detector	**keep alive message frequency ↔ detection speed**
Congestion notification	**probe data/rate ↔ measurement accuracy**
In-band telemetry	**INT postcard volume ↔ post-mortem analysis**
…	

When introducing a distributed coordination function…

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

Time synchronization	**clock-sync rate ↔ clock precision**
Failure detector	**keep alive message frequency ↔ detection speed**
Congestion notification	**probe data/rate ↔ measurement accuracy**
In-band telemetry	**INT postcard volume ↔ post-mortem analysis**
…	

Is this trade-off between overhead and fidelity necessary?

When introducing an in-band control function…

To consume ***extra bandwidth*** for ***efficacy***, or not to?

Time synchronization **clock-sync rate ↔ clock precision**

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

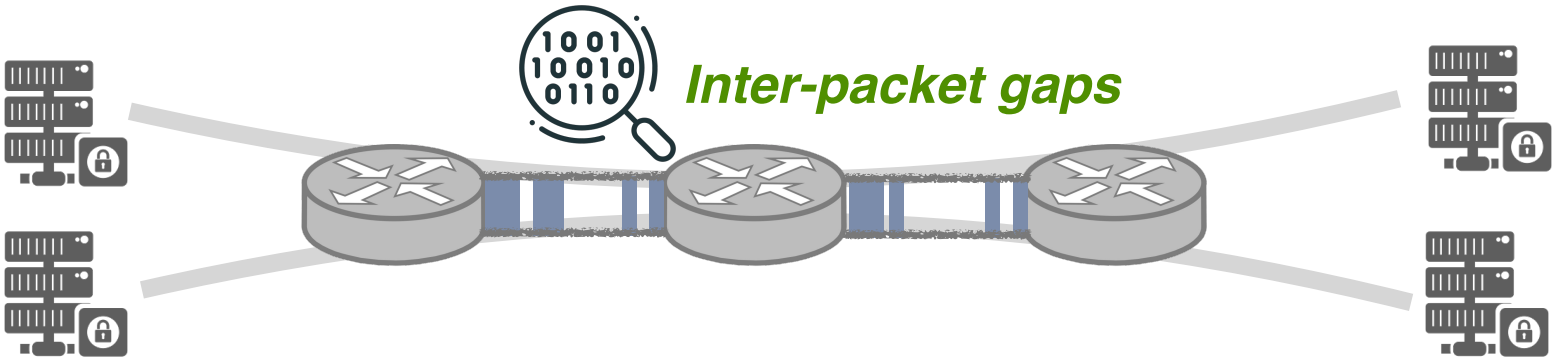
Is this trade-off between fidelity and overhead necessary?

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

Idea: Weaved Stream

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

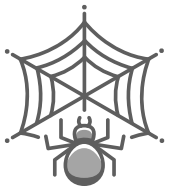
Opportunity: $< \mu s$ gaps are prevalent



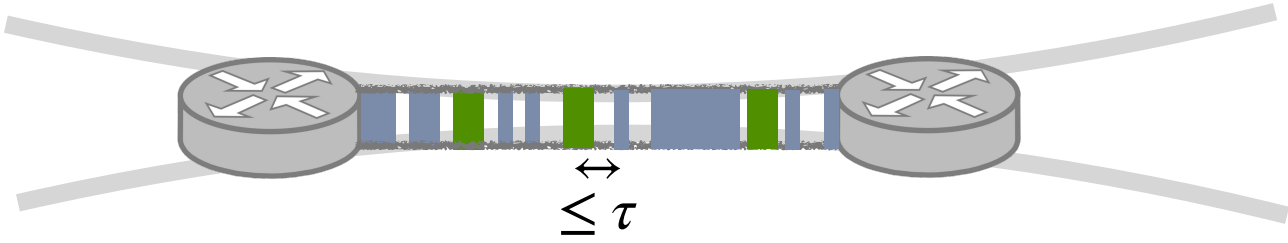
Root causes?

- 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] Not impact user packets or power draw

Abstraction: weaved stream

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

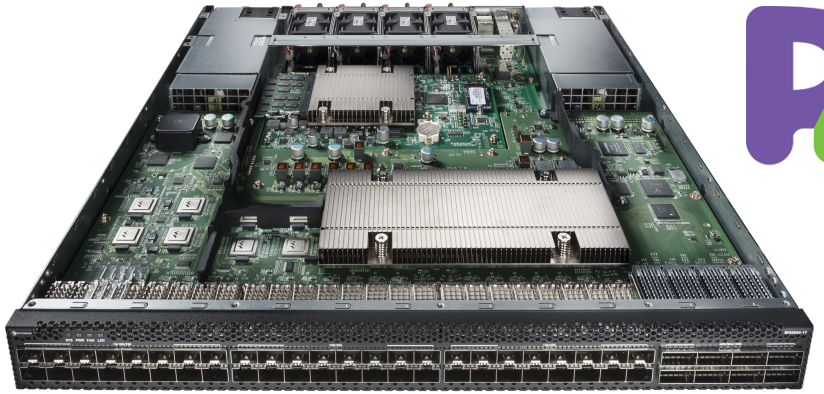
Implement many ***in-network applications***
(failure detection, clock sync, congestion notification…)
for free!

[R1

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

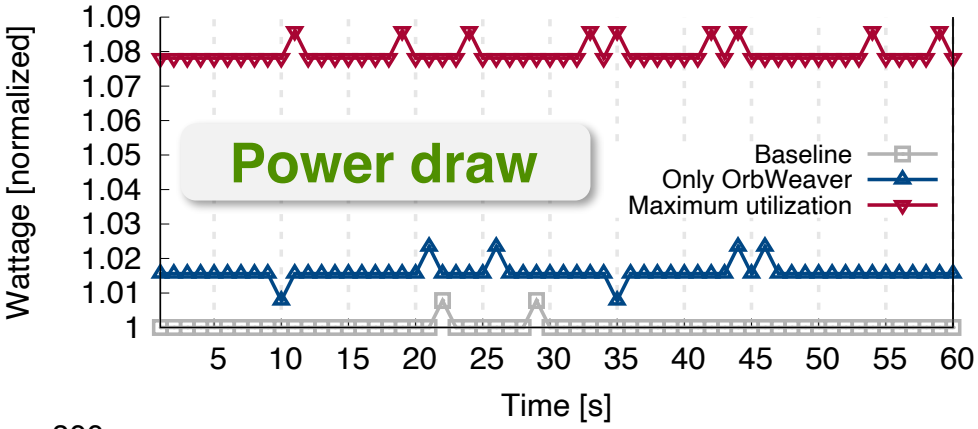
[R2 Little-to-zero overhead] Not impact user packets or power draw

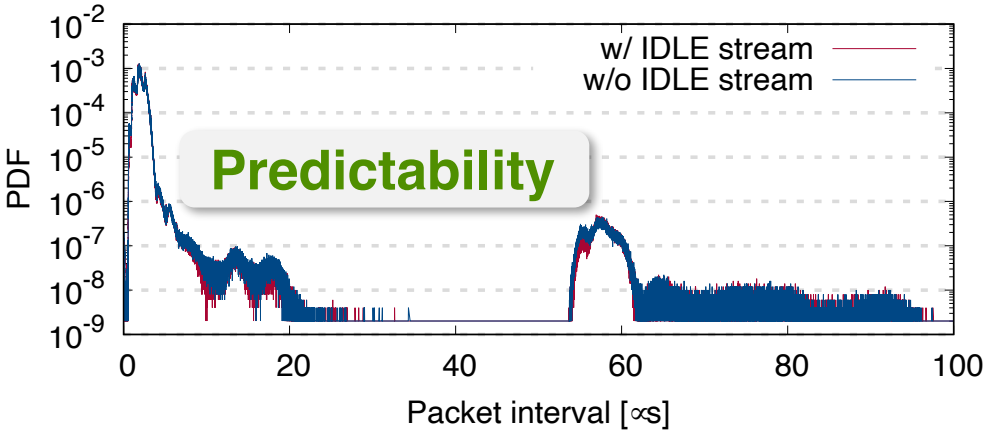
Crazy idea?

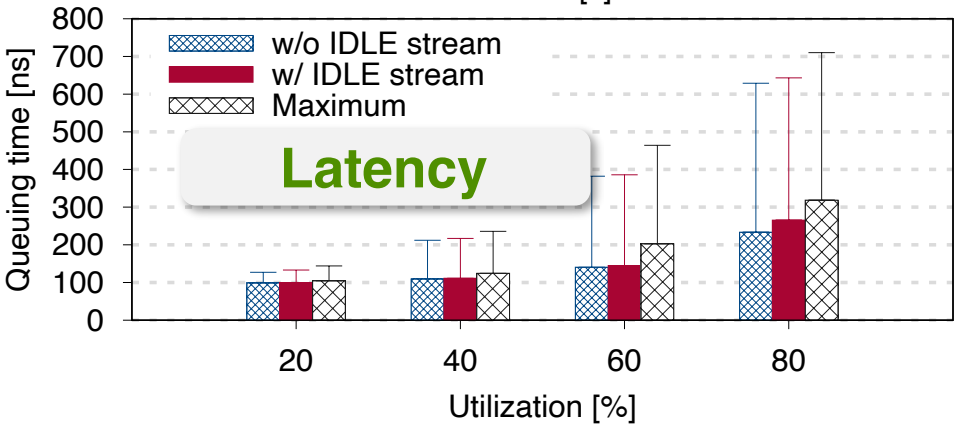


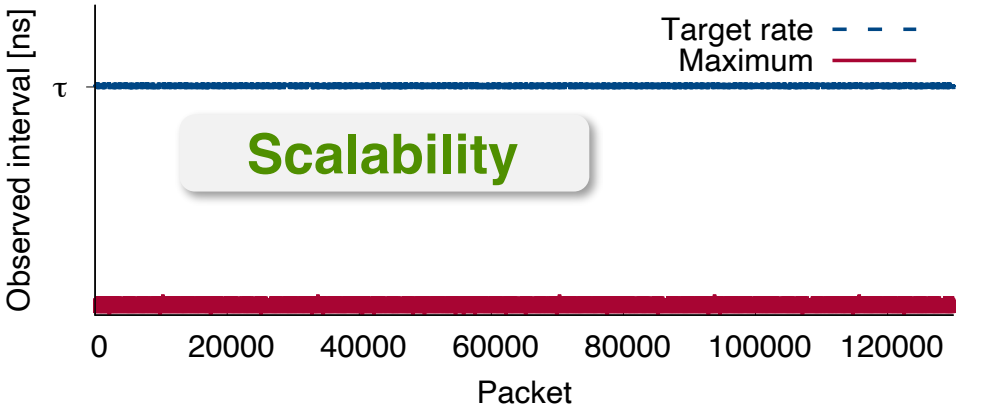
Programmable in-network devices (switches, NICs)

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

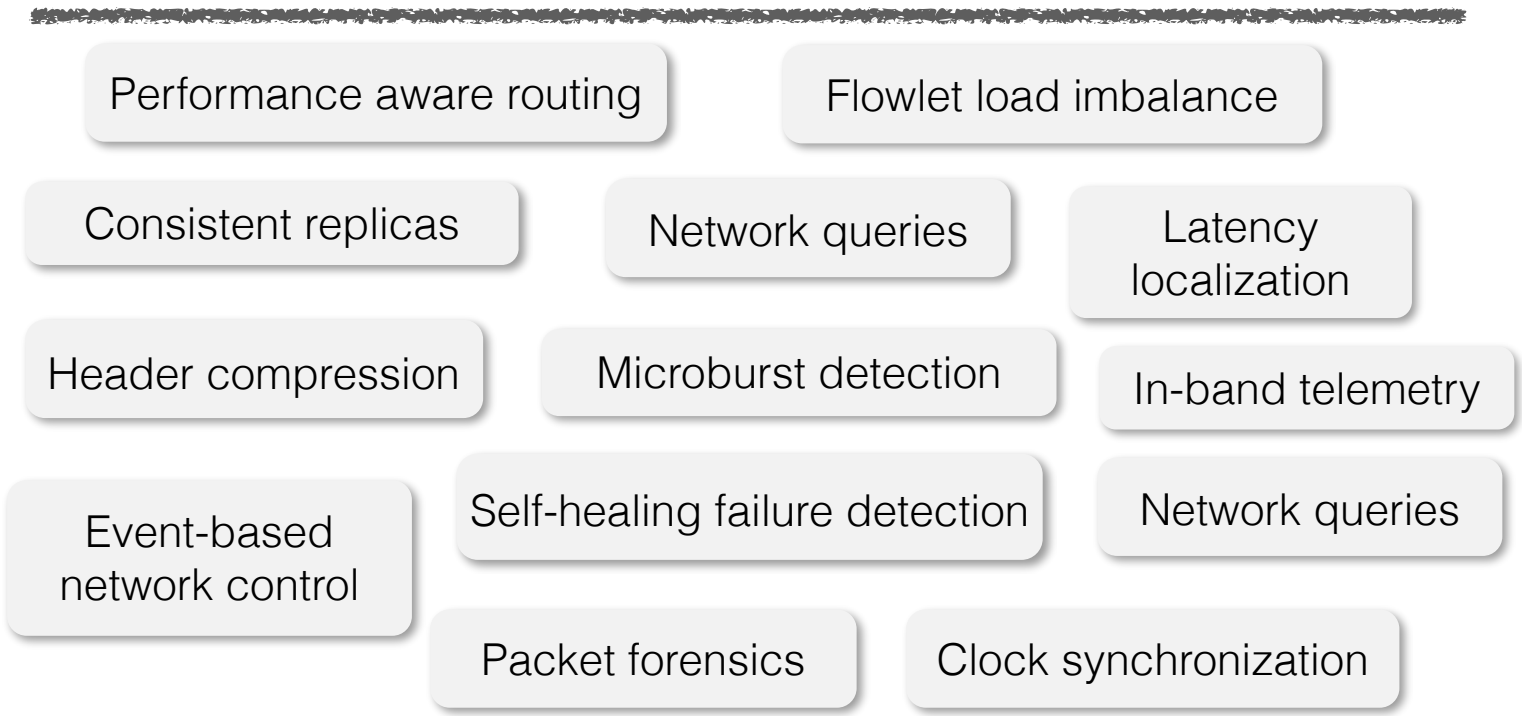




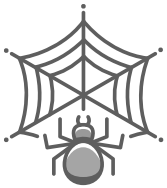




OrbWeaver use cases

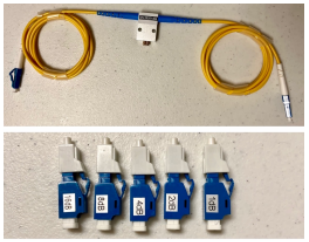


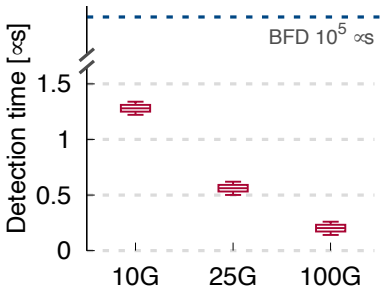
Failure detection with OrbWeaver



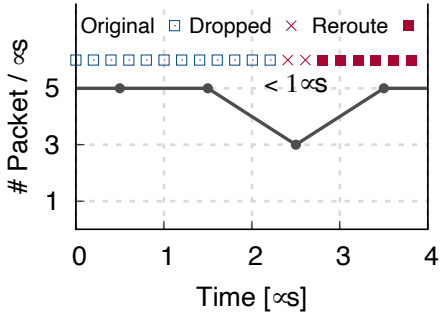
Before: Weak guarantee of the messaging channel

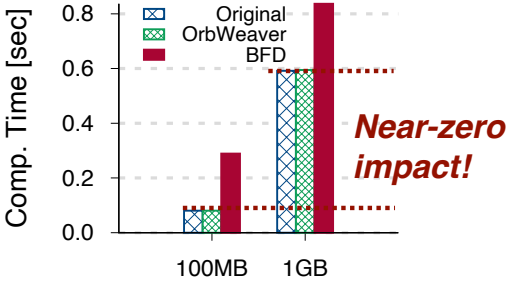
After: OrbWeaver's weaved stream abstraction guarantees maximum inter-packet gap (120ns for 100 GbE)





Emulated failures with optical attenuators tested under varying link speeds





Combining it with data-plane reroute

Push the detection speed to its ***limits*** toward instantaneous, self-healing failure mitigation

Summary

Designing efficient distributed systems primitives by exploiting the characteristics of modern data centers:

Beaver *(OSDI 2024)* *Distributed snapshots*

Cuttlefish *(WIP)* *Synchronous coordination*

OrbWeaver *(NSDI 2022)* *Failure detection*

More opportunities for innovations with emerging data center applications (e.g., LLM agents) and hardware (e.g., time appliance, programmable accelerators)!