Stabilizer: Geo-Replication with User-defined Consistency

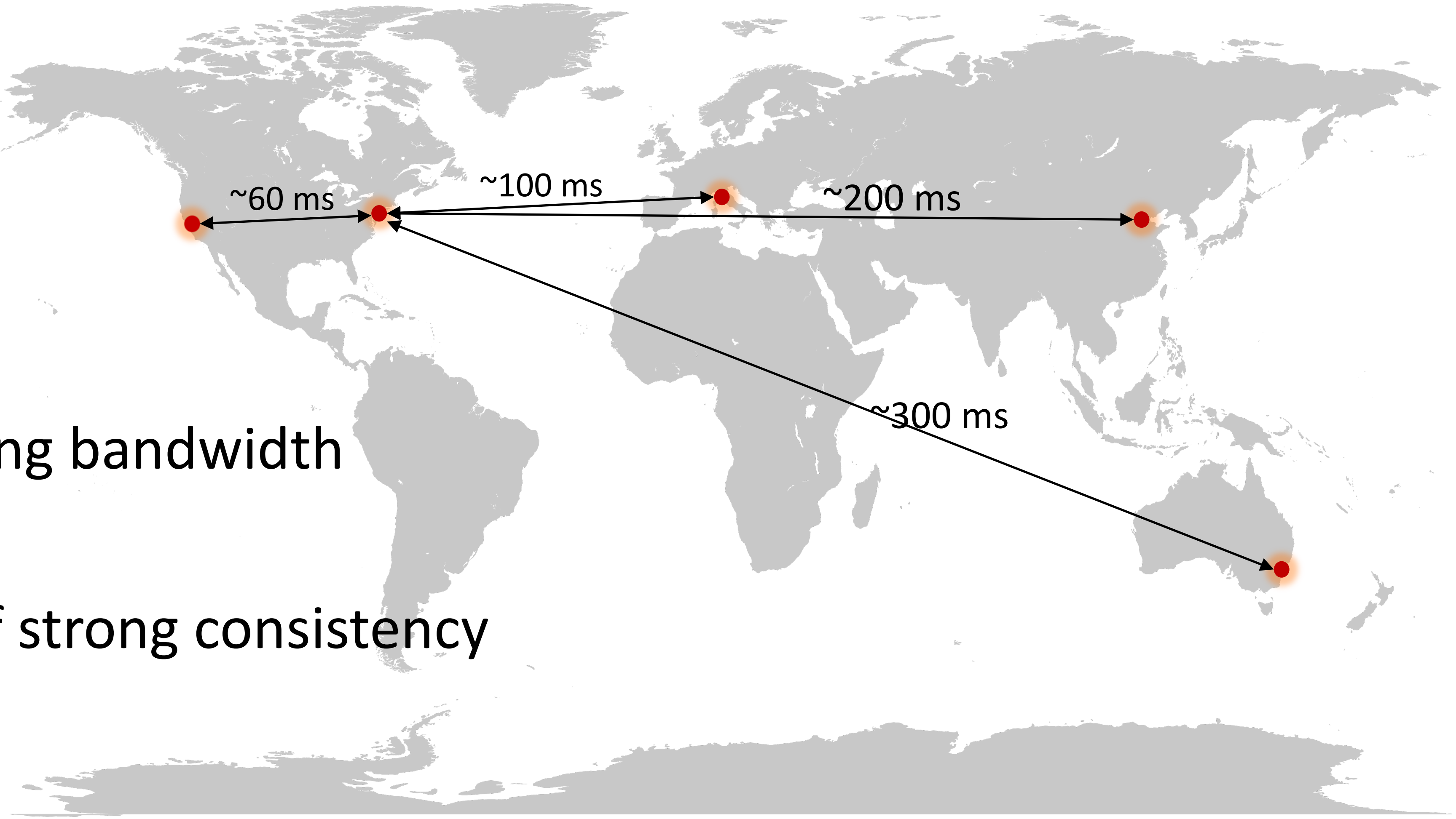
Pengze Li†, Lichen Pan†, Xinzhe Yang‡, **Weijia Song**⋆, Zhen Xiao†, Ken Birman⋆

†Department of Computer Science, Peking University

‡Pure Storage

⋆Department of Computer Science, Cornell University

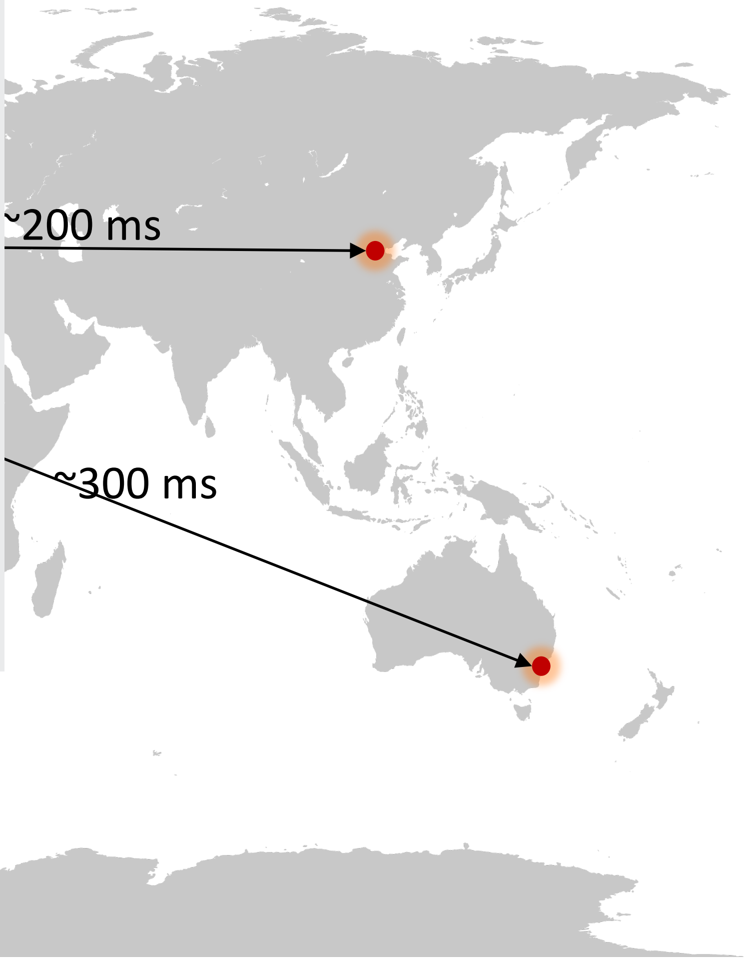
The Challenges of Geo-Replication



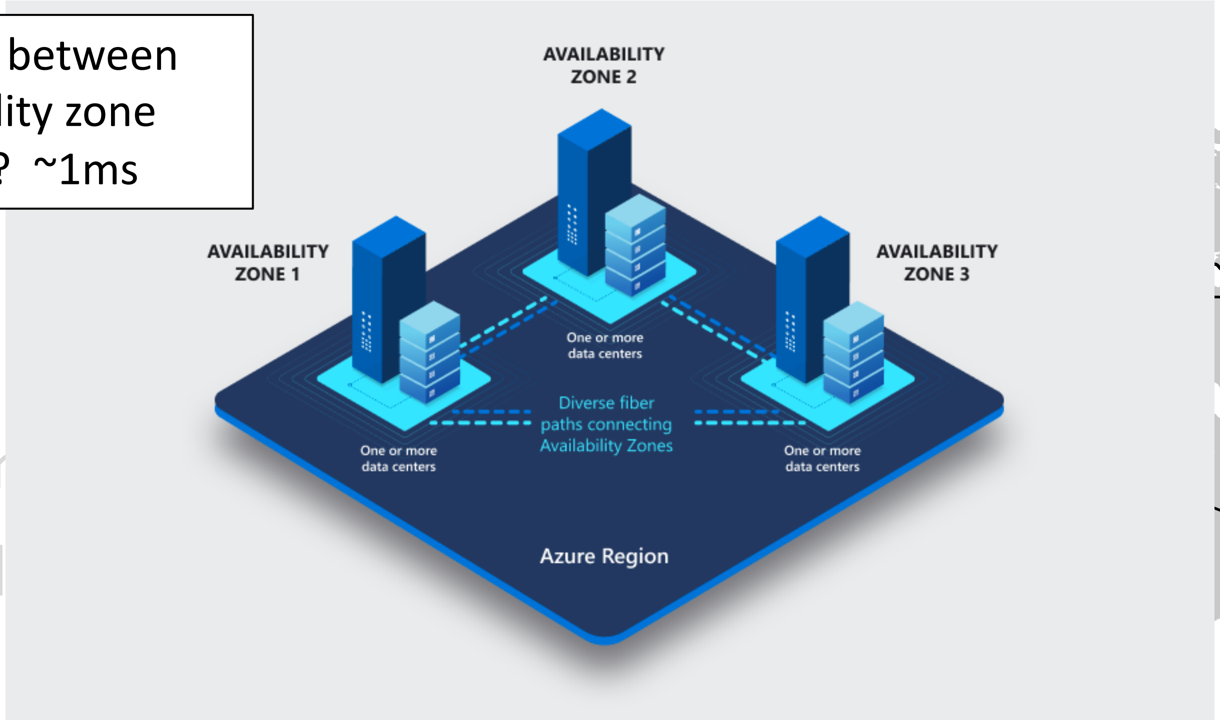
- Long latency
- Limited/fluctuating bandwidth

- High overhead of strong consistency

The Challenges of Geo-Replication



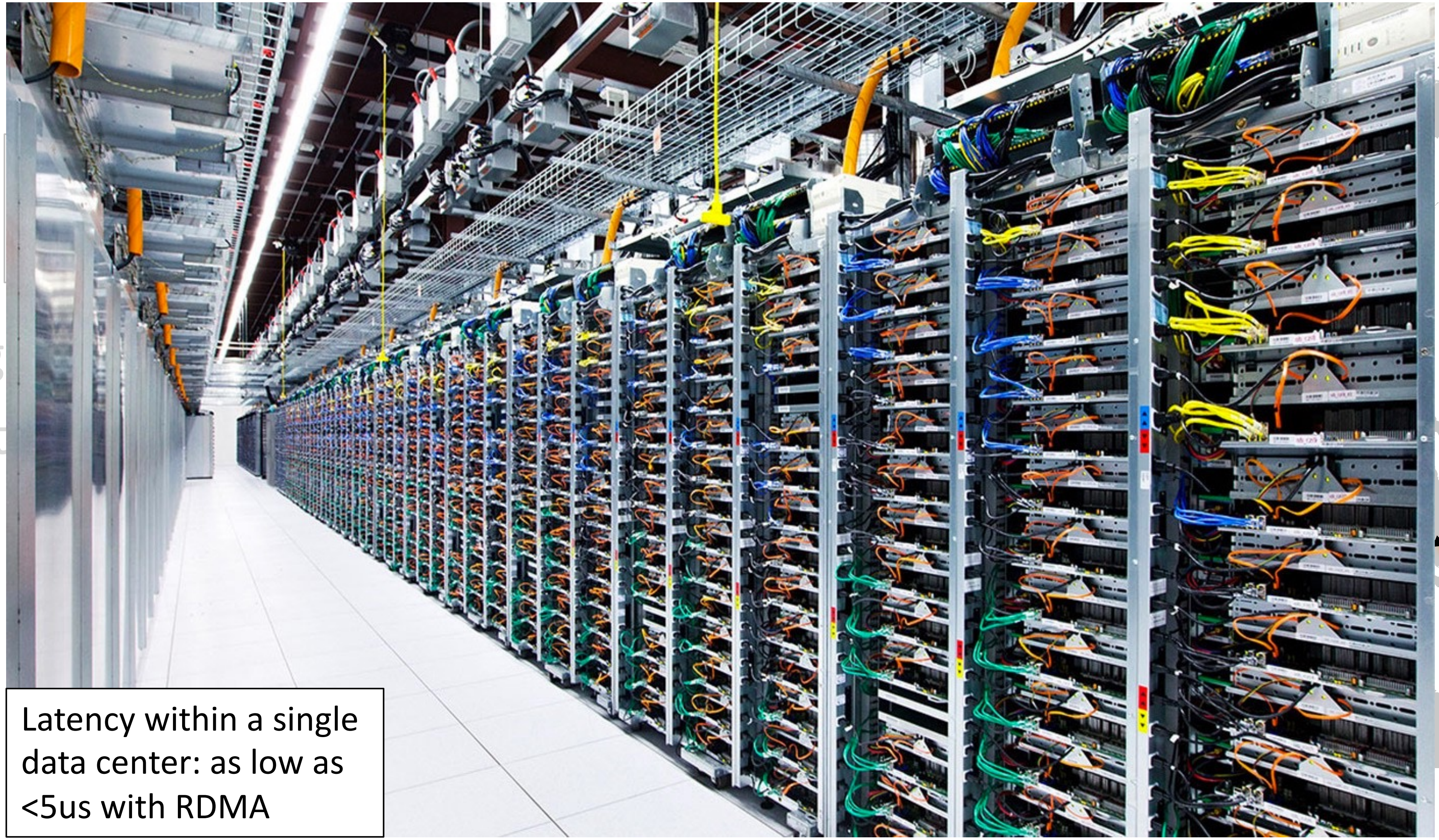
- Long later
- Limited/fl
- High overhead of strong consistency



Latency between availability zone regions? ~1ms

The Challenges of Geo-Replication

- Long
- Limit
- High



Latency within a single data center: as low as <5us with RDMA

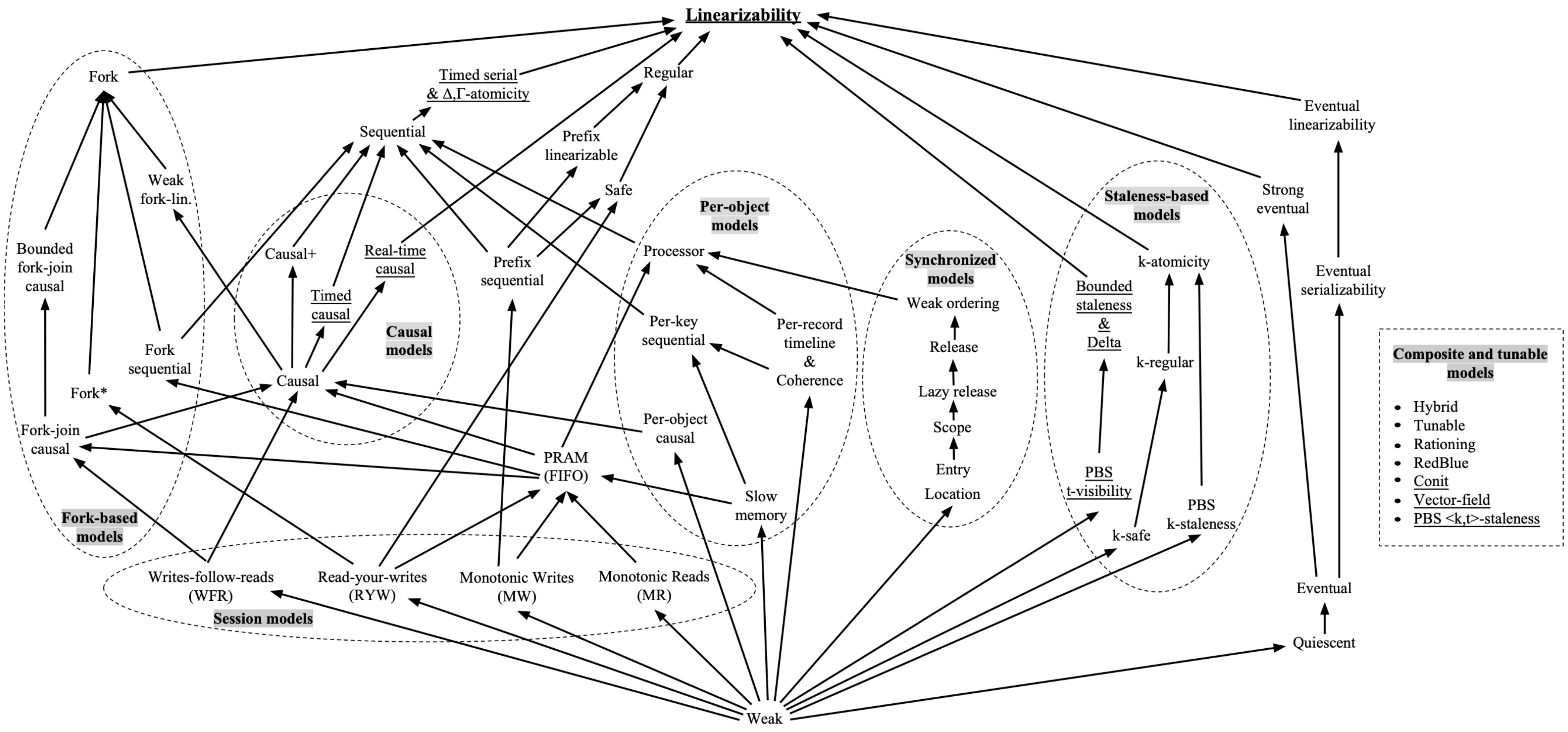
A hierarchy of latencies!

- Applications do need data replication, at scale
- But any “one size fits all” story would impose those geo-WAN latencies and replication delay is (obviously) bounded by latency
- Challenge: Can one solution be customizable across multiple uses?

Different applications?

- Data Mirroring
- Social media
- Banking System
- …

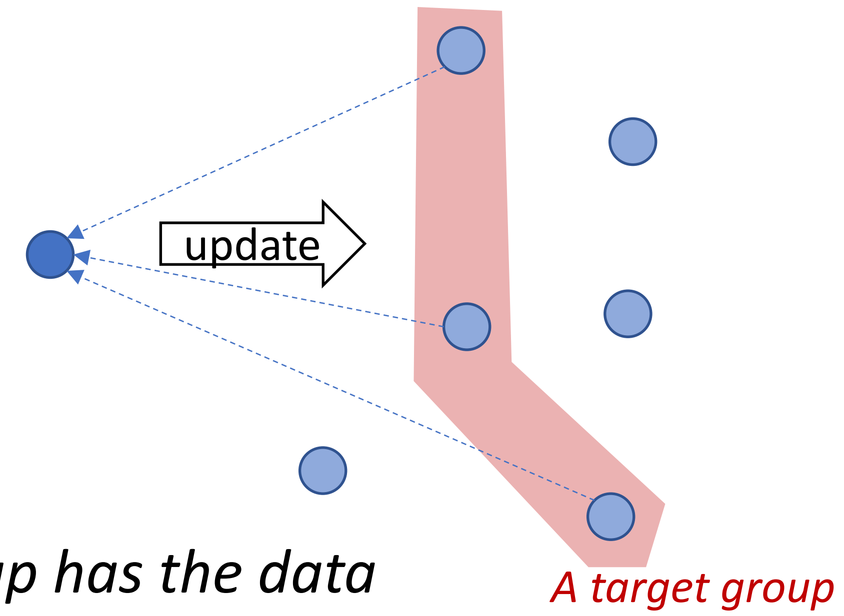
… different consistency models!



Source: Consistency in Non-Transactional Distributed Storage Systems survey, P Viotti and M Vukolić 2016

User-defined stability

- Stabilizer builds on the idea of *stability within user-defined target groups*
  - User offered flexible ways to define the group
  - For example,
    - "majority in my data center",
    - "all regions in some availability zone"
    - "At least 2 geo-distributed regions"
    - "A quorum from this set of targets…"
- What should "stability" mean?
  - For us, *confirmation that the desired target group has the data*
  - Proof of stability? Flexible: The application itself confirms it

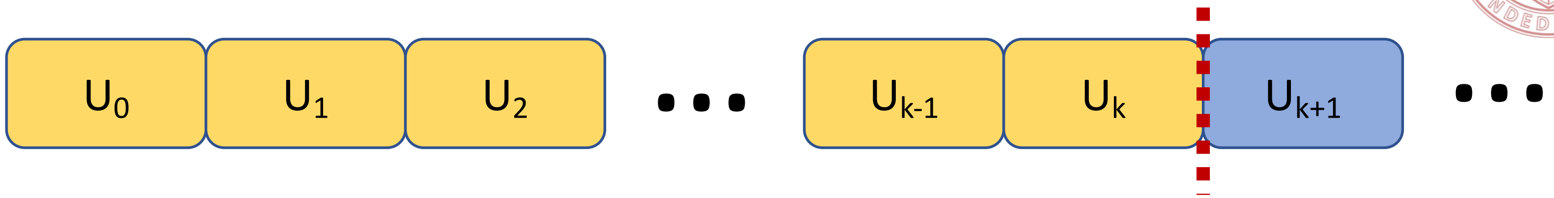


User-defined <u>stability</u>

- Examples of application-defined forms of stability:
  - For trusting applications, the user code in some region might simply report that “all data from source **S** up through update **K** has been **received**”
  - An application focused on archival safety might change that to:
    “all data from source **S** up through update **K** has been **persisted**”
  - A less trusting application might, for example, check the integrity of a blockchain, then report that “region **R** has persisted chain **S** to update **K**”

- Application trusts its members? The update number, **K**, suffices.
- Less trusting? Application can use cryptographic “witness” signatures

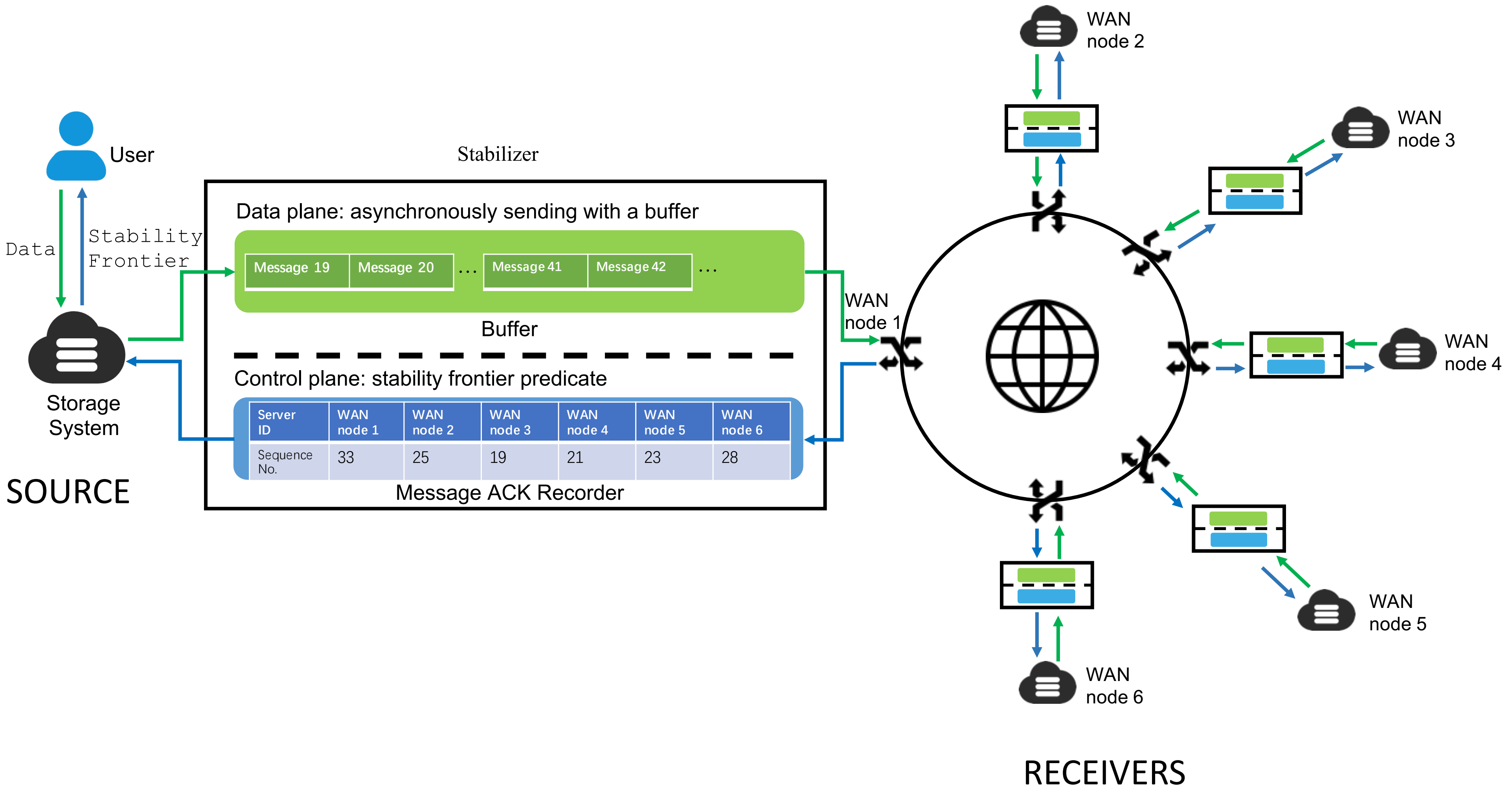
Stabilizer



Stability Frontier: K

- A geo-replication library for cloud applications
- Abstraction of target regions.
  - Data model: Each region is "owner"(primary) of some **stream of updates**
  - Streams can be replicated to any desired target(s) in **lossless FIFO** channels
  - A receiver (backup) is a component of the application that ingests the stream, announce status via a sequence of **stability reports** ("certificates", which can be signed)
  - Signature certificates can include extra application-specific content
- **Stability frontier**: for a specific target group, all updates from source **S** have reached the desired stability level up through update **K**.
  - Each stability frontier advances monotonically
  - Certificate through **K** implies stability for updates [0…K]

System Architecture



Stability Frontier Predicate

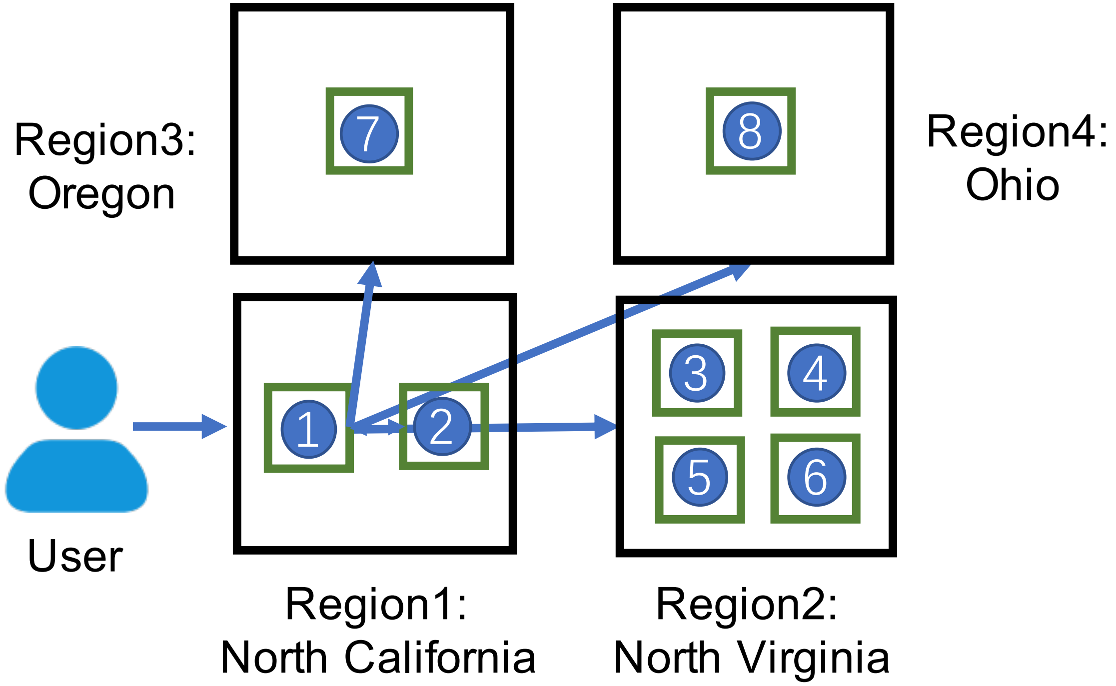
Stability Frontier predicate function

- Domain: the maximum sequence numbers acknowledged by every receivers
- Output: the maximum sequence number of the “stable” update

We introduced a set of building-block tools to describe such a predicate

- K^{th}-MAX() and K^{th}-MIN() operator
- $ALLWNODES, $MYAZNODES
- SIZEOF(), -
- Suffixes: .received, .persisted, .signed

Examples of Stability Frontier Predicate



An update is considered stable only after it is confirmed by **all receivers.**
MIN($ALLWNODES)

An update is considered stable only after it is confirmed by **a majority of receivers.**
Kth-MIN(SIZEOF($ALLWNODES)/2+1,$ALLWNODES)

An update is considered stable only after it is confirmed by **a majority of remote regions.**
Kth-MIN(3, MAX($AZ North Virginia),MAX($AZ Oregon),MAX($AZ Ohio))

Performance and Flexibility Design

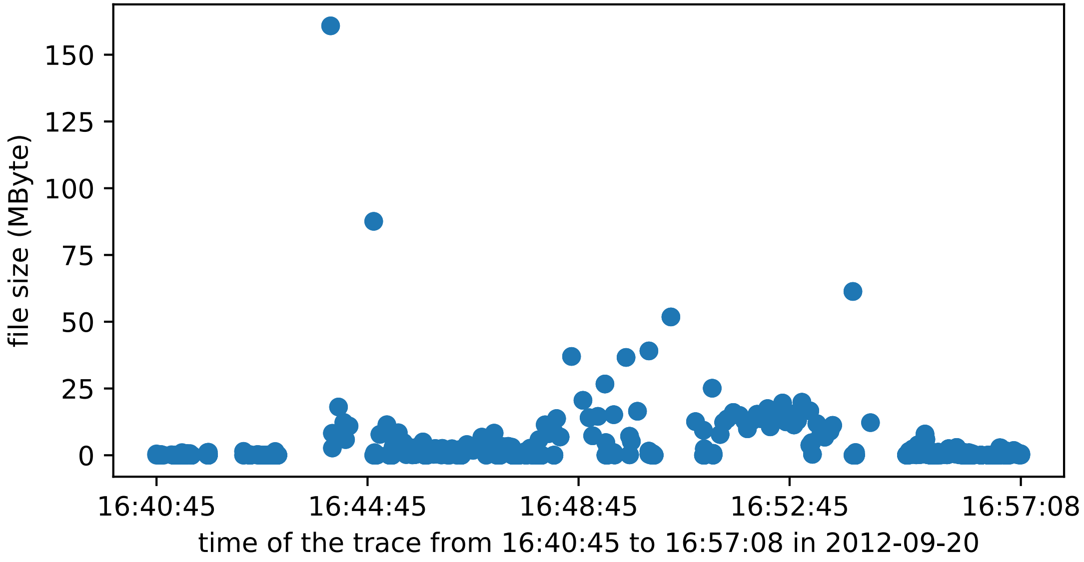
- The Stability Frontier predicate is compiled to dynamic linked library and loaded at runtime using gcc-jit.
  - Native performance (**5x faster** than an interpreter approach)
  - **same flexibility** of the interpreter approach

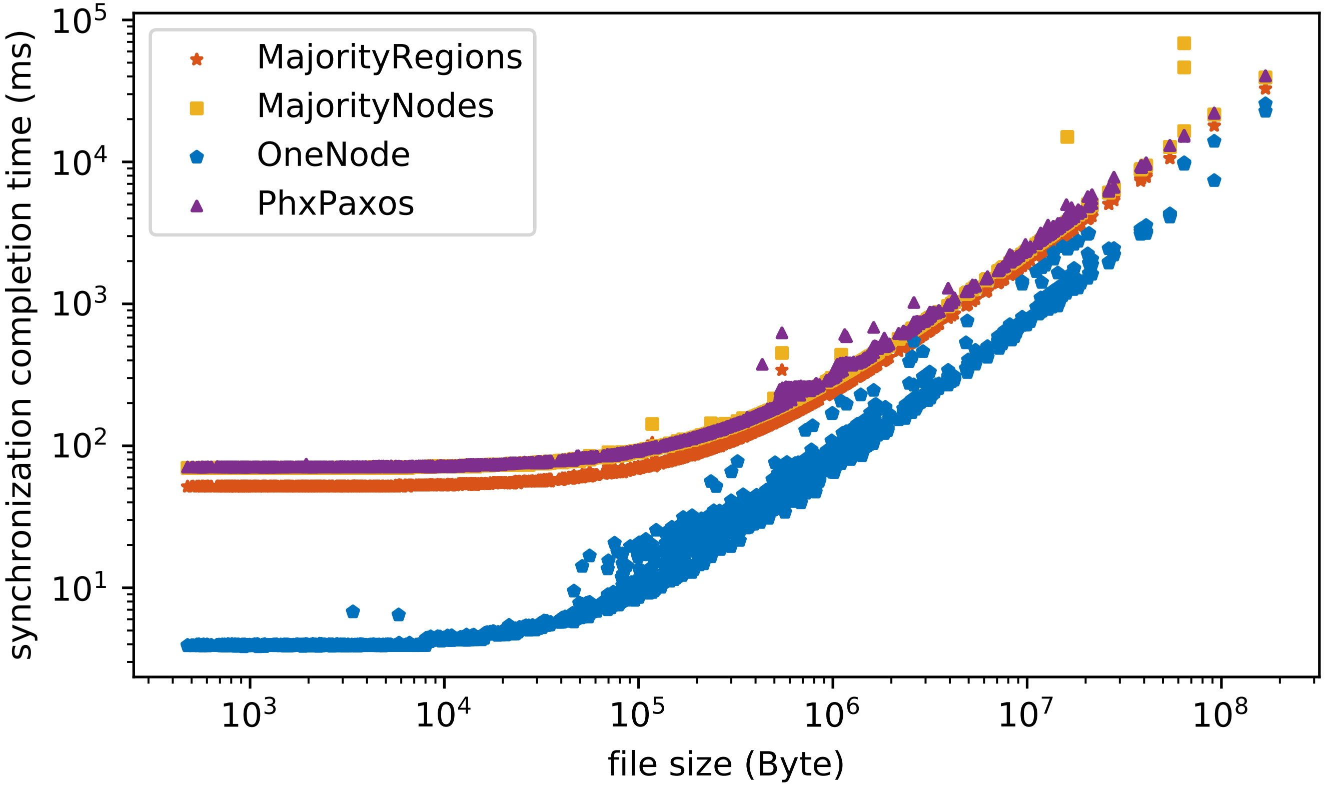
Dropbox Latency

NETWORK STATUS BETWEEN NORTH CALIFORNIA AND OTHER REGIONS

	Lat (ms)	Thp (Mbit/s)	Half Thp (Mbit/s)
North California*	3.7	667	333.5
Ohio	53.87	89	44.5
Oregon	23.29	113	56.5
North Virginia	64.12	74	37

*The network status between availability zones in North California region

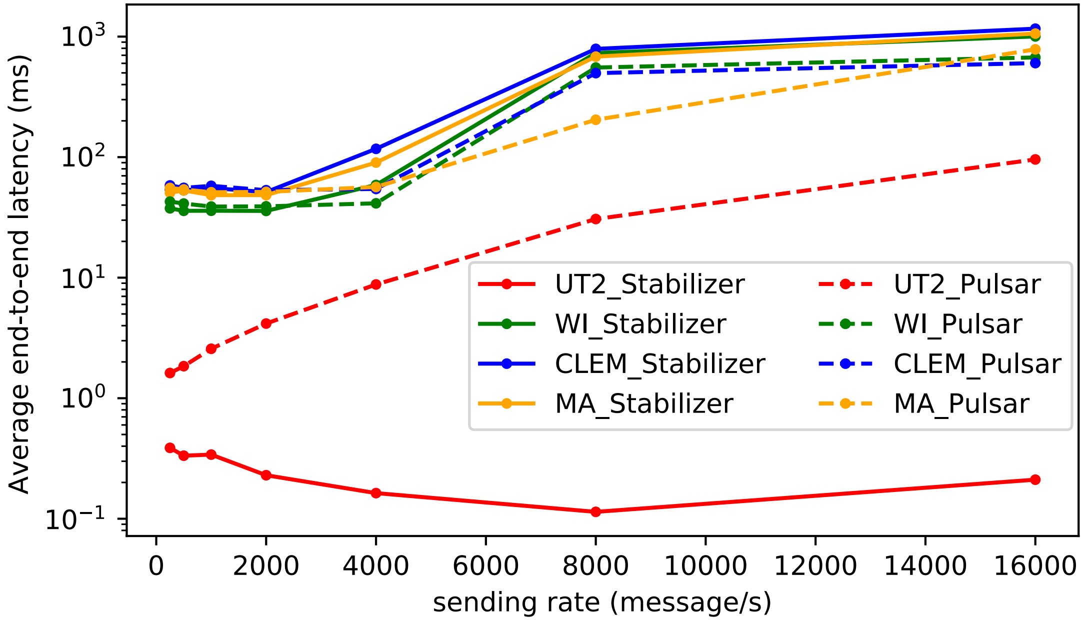


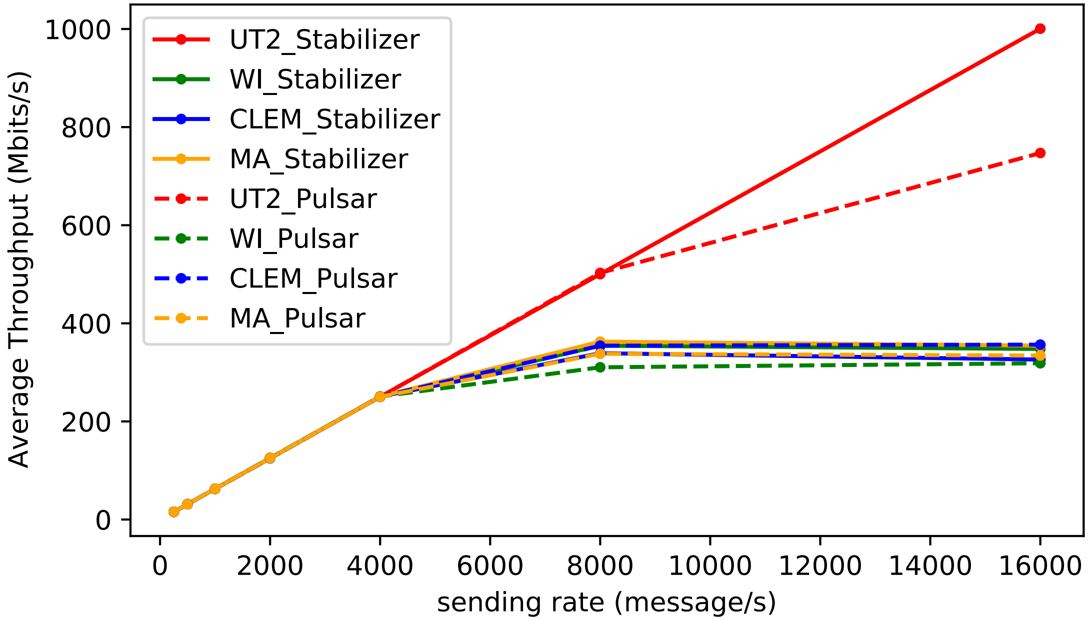


Pub/Sub application: Stabilizer vs Pulsar

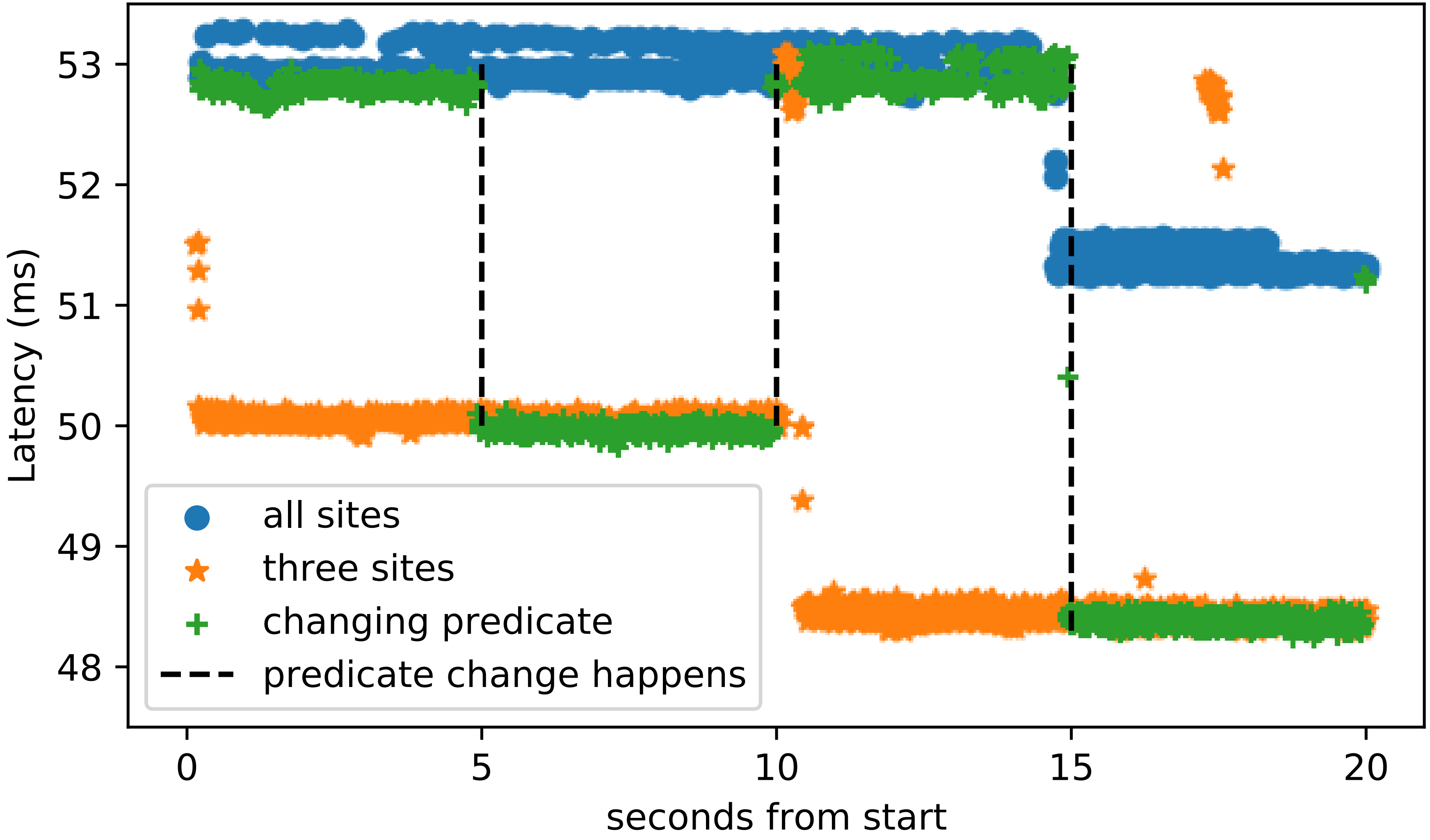
Network performance between Utah1 and other servers

	Utah2	Wisconsin	Clemson	Massachusetts
Thp(Mbit/s)	9246.99	361.82	416.27	437.11
Lat(ms)	0.124	35.612	50.918	48.083





Dynamic Stability Frontier Reconfiguration



Conclusion

- Design and Implement a geo-replication library that allows the user-defined stability
- Introduced the stability frontier concepts along with a Domain Specific Language (DSL) to describe it.
- Build several applications (K/V store, cloud file storage, and pub/sub system) to demonstrate/evaluate Stabilizer.

Thank you!

- Q&A