ServiceRouter

HYPERSCALE AND MINIMAL COST SERVICE MESH AT META

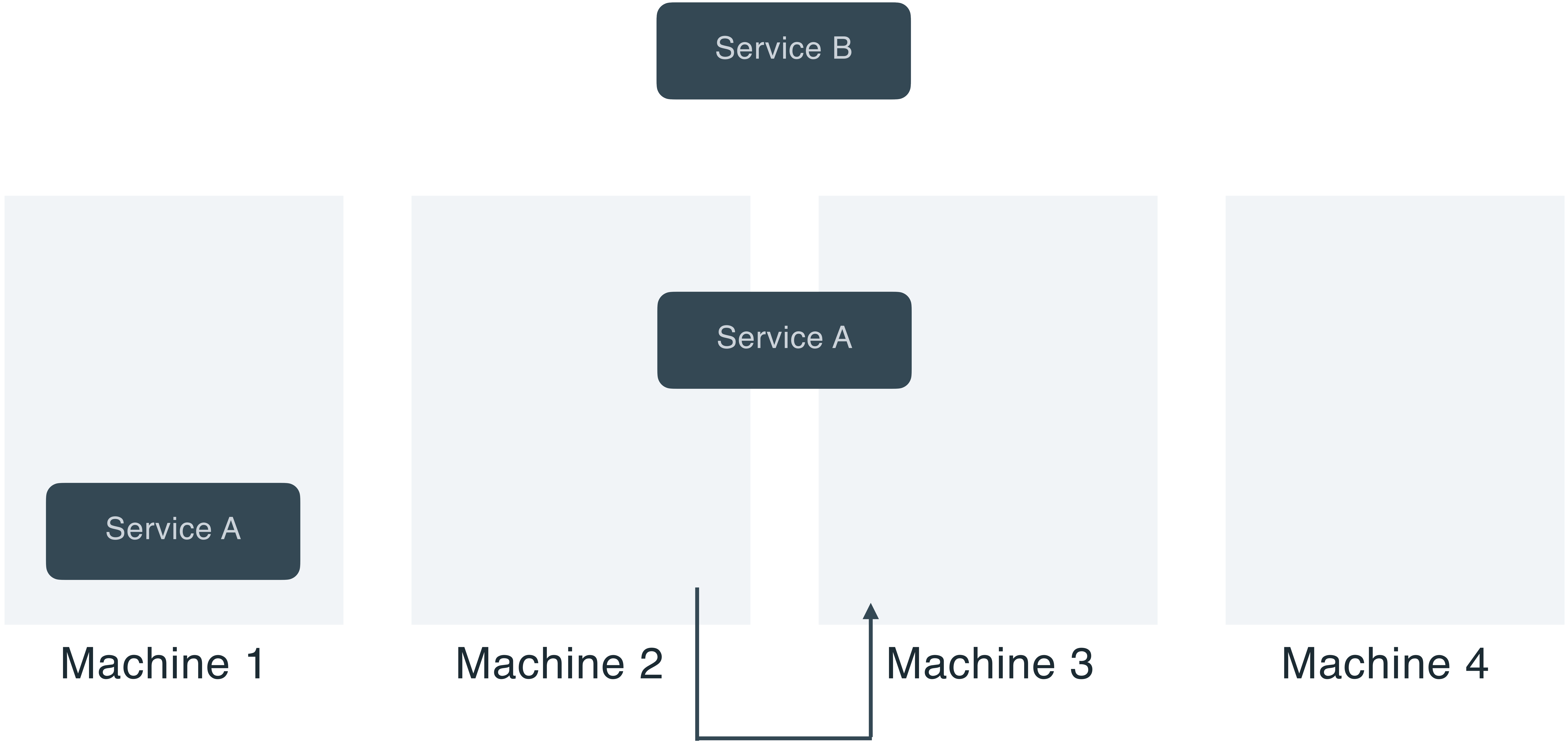
Harshit Saokar[1] | Soteris Demetriou[1,2] | Nick Magerko[1] | Max Kontorovich[1] | Josh Kirstein[1] | Margot Leibold[1] | Dimitrios Skarlatos[1,3] | Hitesh Khandelwal[1] | Chunqiang Tang[1]

[1] Meta

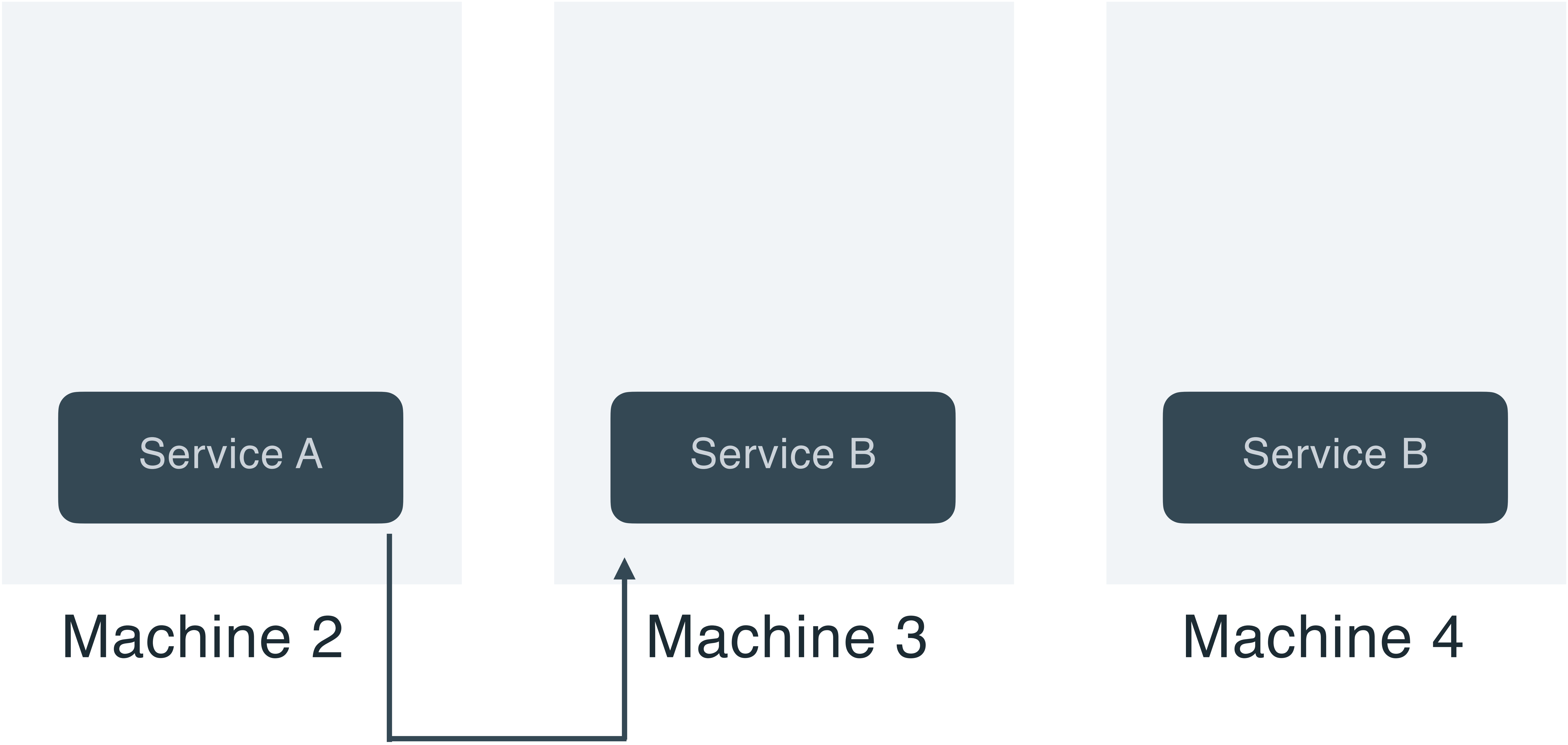
[2] Imperial College London

[3] Carnegie Mellon University Computer Science Department

01 Background & Motivation



Huye et al. Lifting the veil on Meta's microservice architecture: Analyses of topology and request workflows. USENIX ATC ‘23



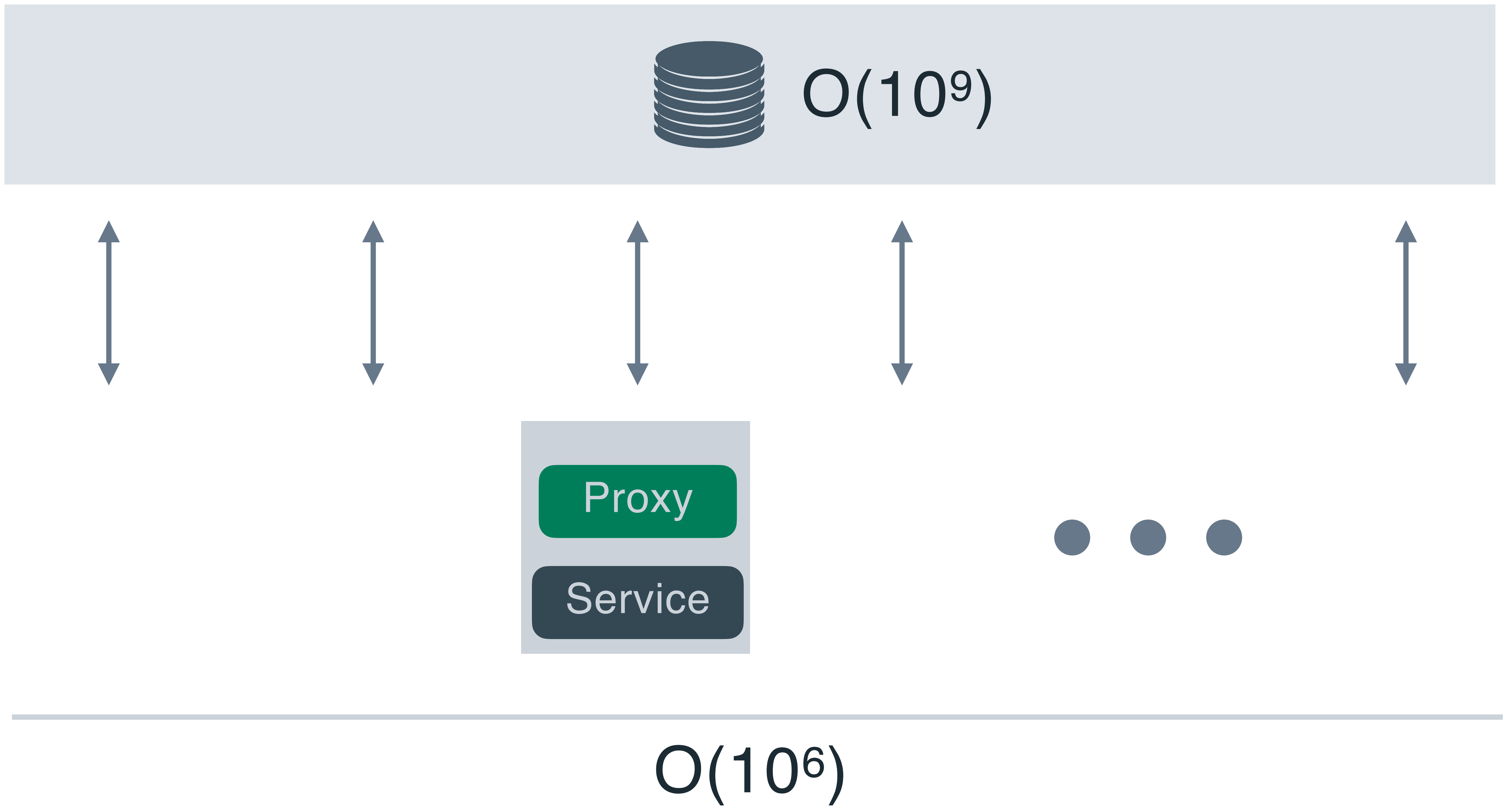
RPC Frameworks

- No Advanced Load Balancing
- Need external support for service discovery
- Examples: gRPC, Thrift

Background & Motivation: Service Mesh

Service Mesh Challenges

- **[SCALABILITY]** How can we scale service discovery to $O(10^6)$ clients and proxies?

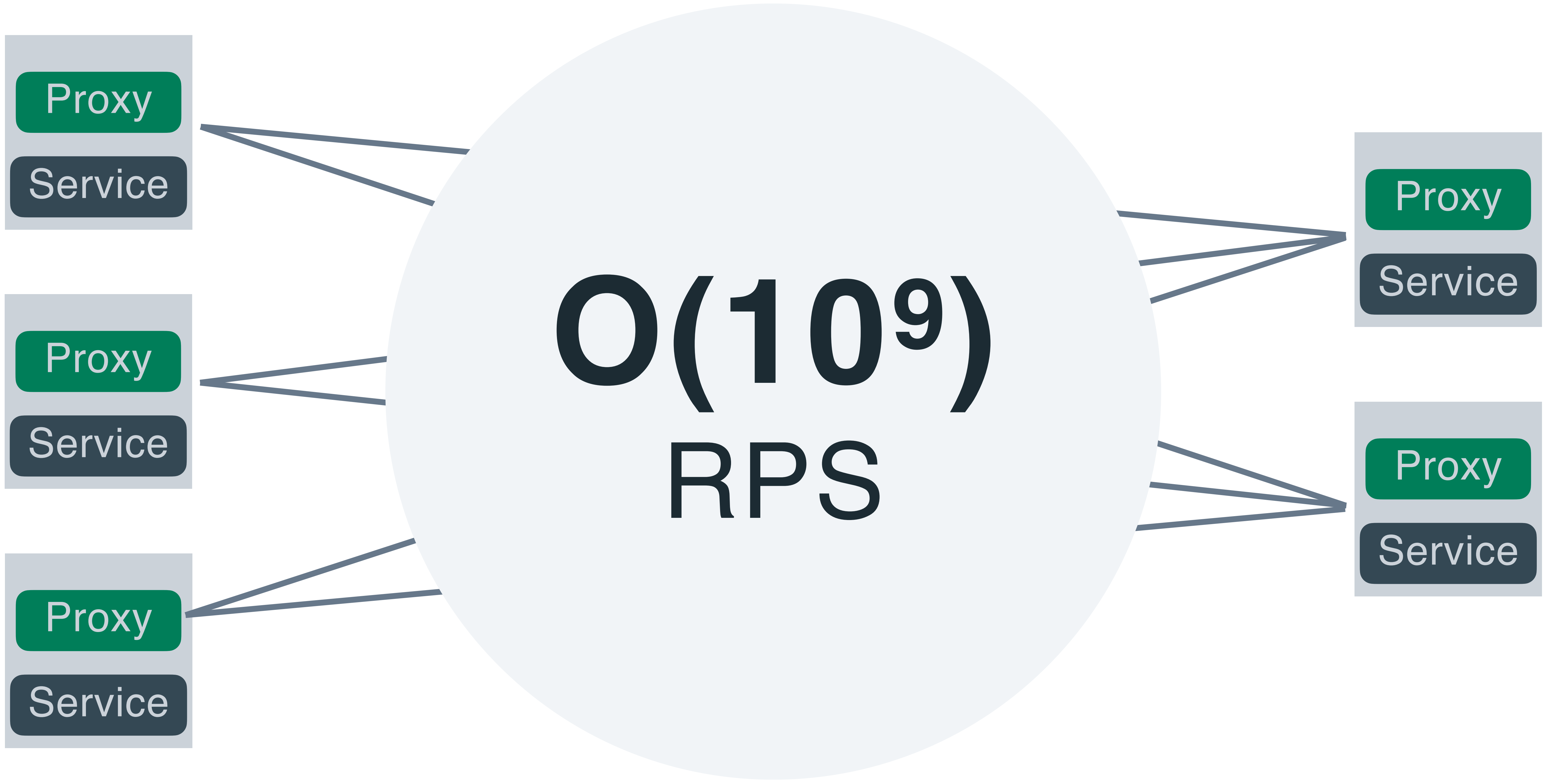


Service Mesh Challenges

- **[SCALABILITY]** How can we scale service discovery to $O(10^6)$ clients and proxies?
- **[HW COST]** How to minimize HW cost?

Istio: 0.35vCPU for $O(10^3)$ rps

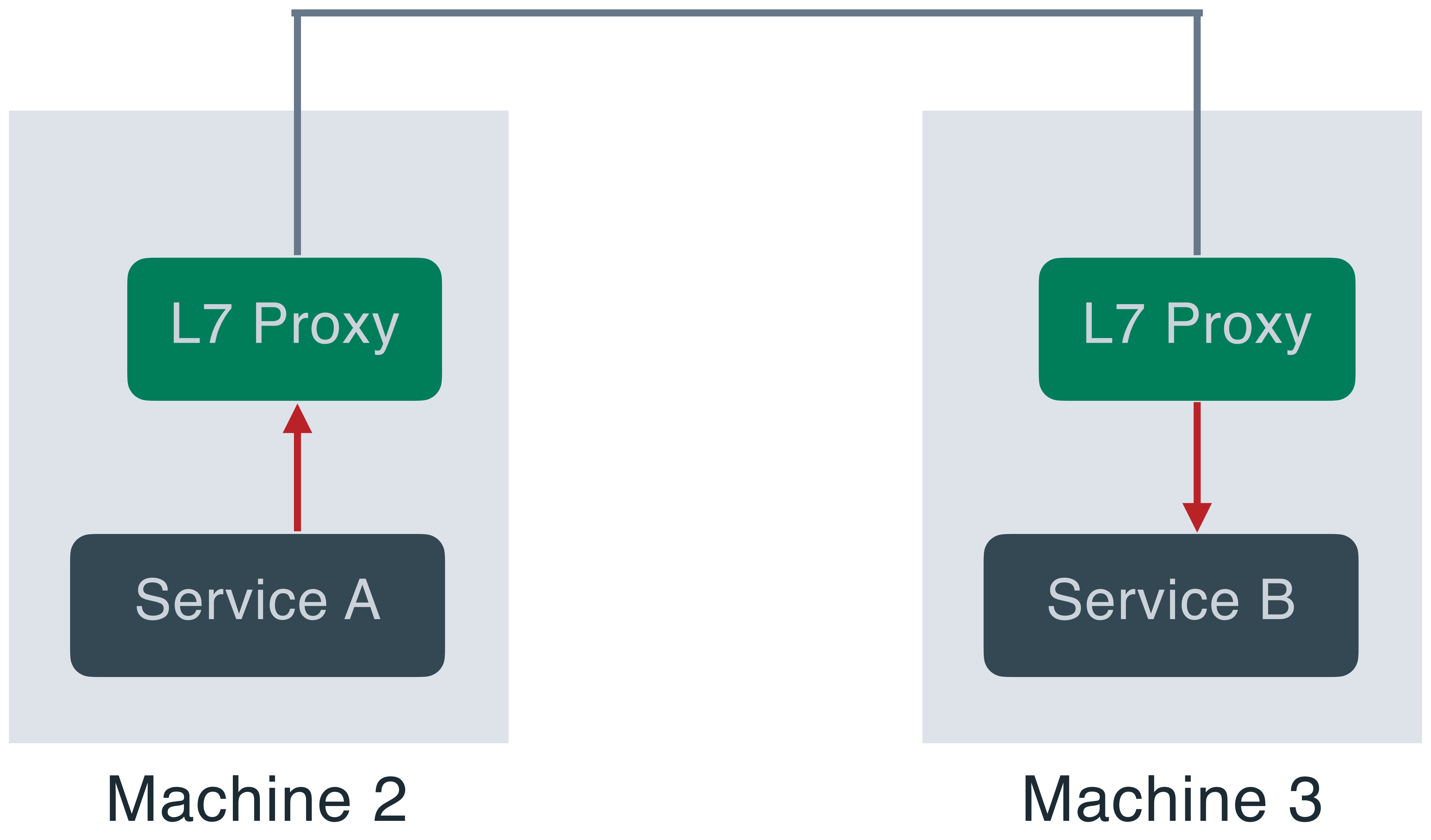
1,750,000 AWS t4g.small VMs for 10B rps



Service Mesh Challenges

- [SCALABILITY] How can we scale service discovery to $O(10^6)$ clients and proxies?
- [HW COST] How to minimize HW cost?

- **[RPC LATENCY & LB]** How to simultaneously minimize RPC latency and load balance across geo-distributed hosts?
  - Sidecars add extra latency



Zhu et al show that Istio

- increases the latency by **185%**

Zhu et al. Dissecting Service Mesh Overheads. In *arXiv preprint arXiv:2207.00592*, 2022.

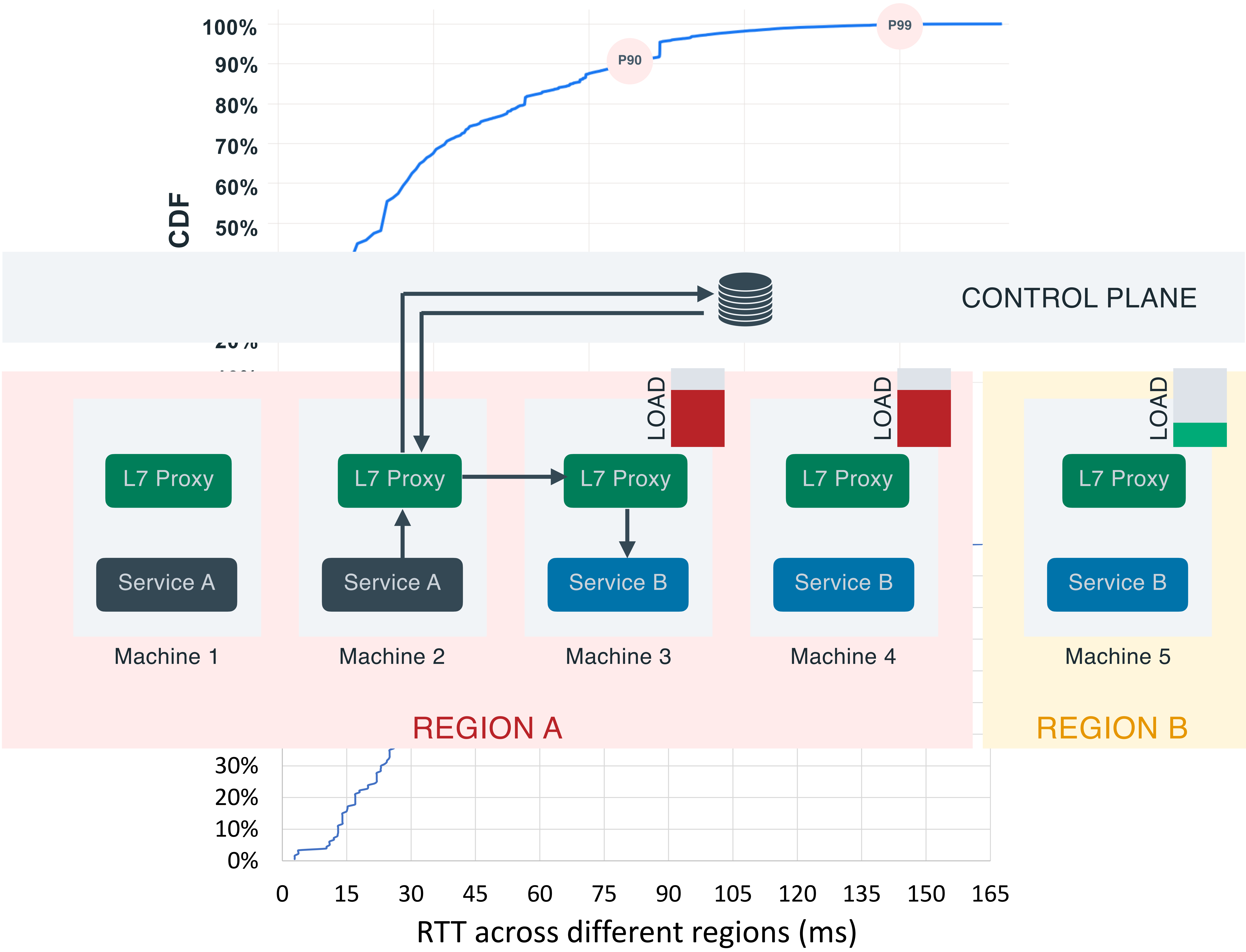
mRPC shows that a sidecar approach:

- increases P99 RPC latency by **180%**

Chen, et al. Remote procedure call as a managed system service. *NSDI '23*

Service Mesh Challenges

- [SCALABILITY] How can we scale service discovery to $O(10^6)$ clients and proxies?
- [HW COST] How to minimize HW cost?
- **[RPC LATENCY & LB]** How to simultaneously minimize RPC latency and load balance across geo-distributed hosts?
    - Sidecars add extra latency
    - $O(10\text{-}10^4)$ hosts per service
    - P90 cross-region latency: 106ms



03 ServiceRouter

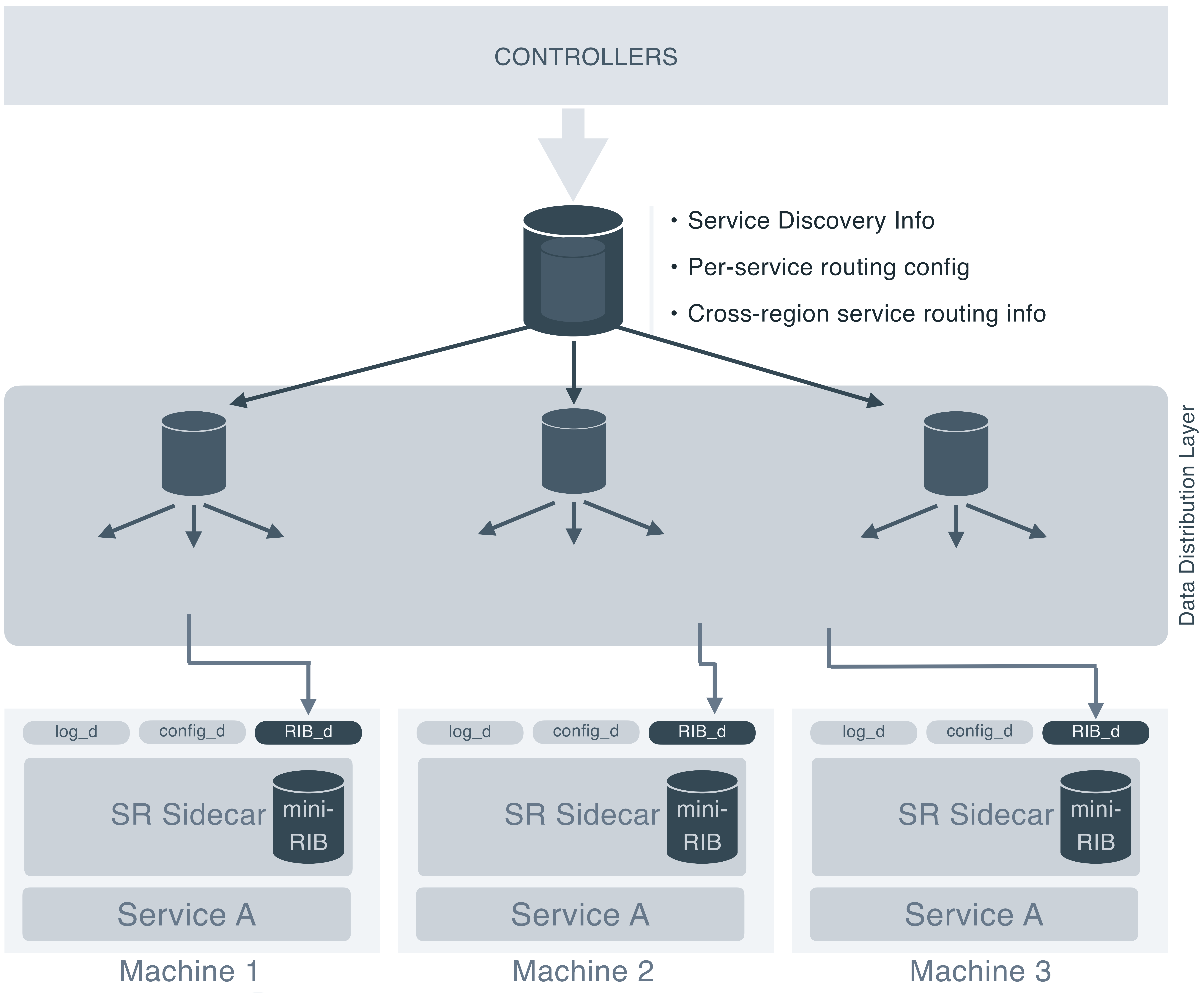
KEY DESIGN CONCEPTS

RIB

Routing Information Base

Decentralize the unscalable part of the control plane in order to scale out.

- Independent controllers execute different functions such as registering services and generating a per-service cross-region routing table.
- The data distribution layer massively replicates the RIB so that there are sufficient RIB replicas to handle read traffic from millions of proxies.
- Each proxy self-configures and self-manages without the control plane's direct involvement.



Versatility

Controllers are agnostic to the L7 architecture.

99% RPC traffic routed through SRLib.

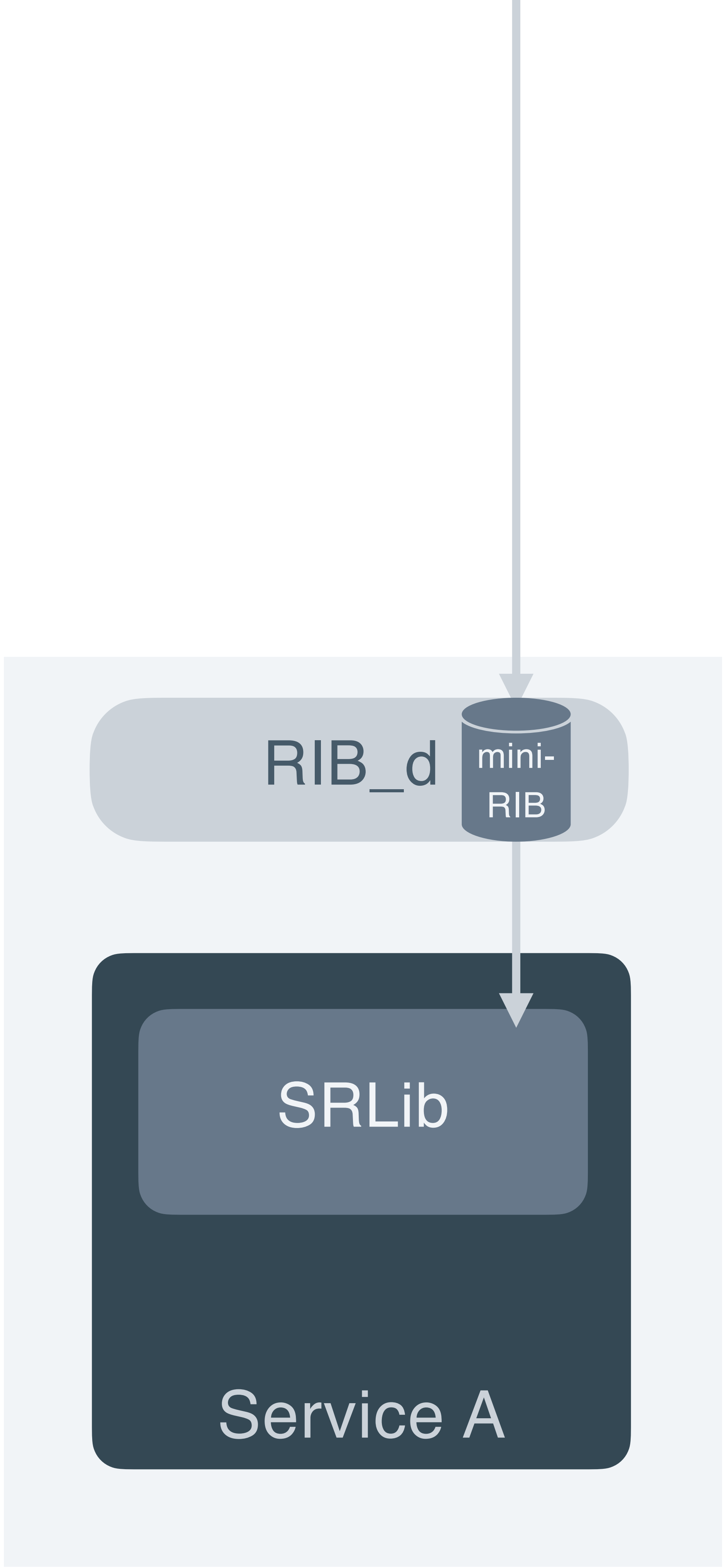
SRLib

Provide the service-mesh functions out of a library that is directly linked into the RPC client's executable

- Eliminates side car latency overhead

Run a separate RIBDaemon on the client machine to cache miniRIB.

- Performance isolation between service discovery and routing.



LATENCY RINGS AND CROSS-REGION ROUTING

SR strives to simultaneously minimize RPC latency and balance load across global regions.

- SR introduces the concept of latency rings to minimize latency.
- SR collects per-service global traffic and load information, computes a per-service cross-region routing table, and disseminate it to L7 routers to guide their local routing decisions.

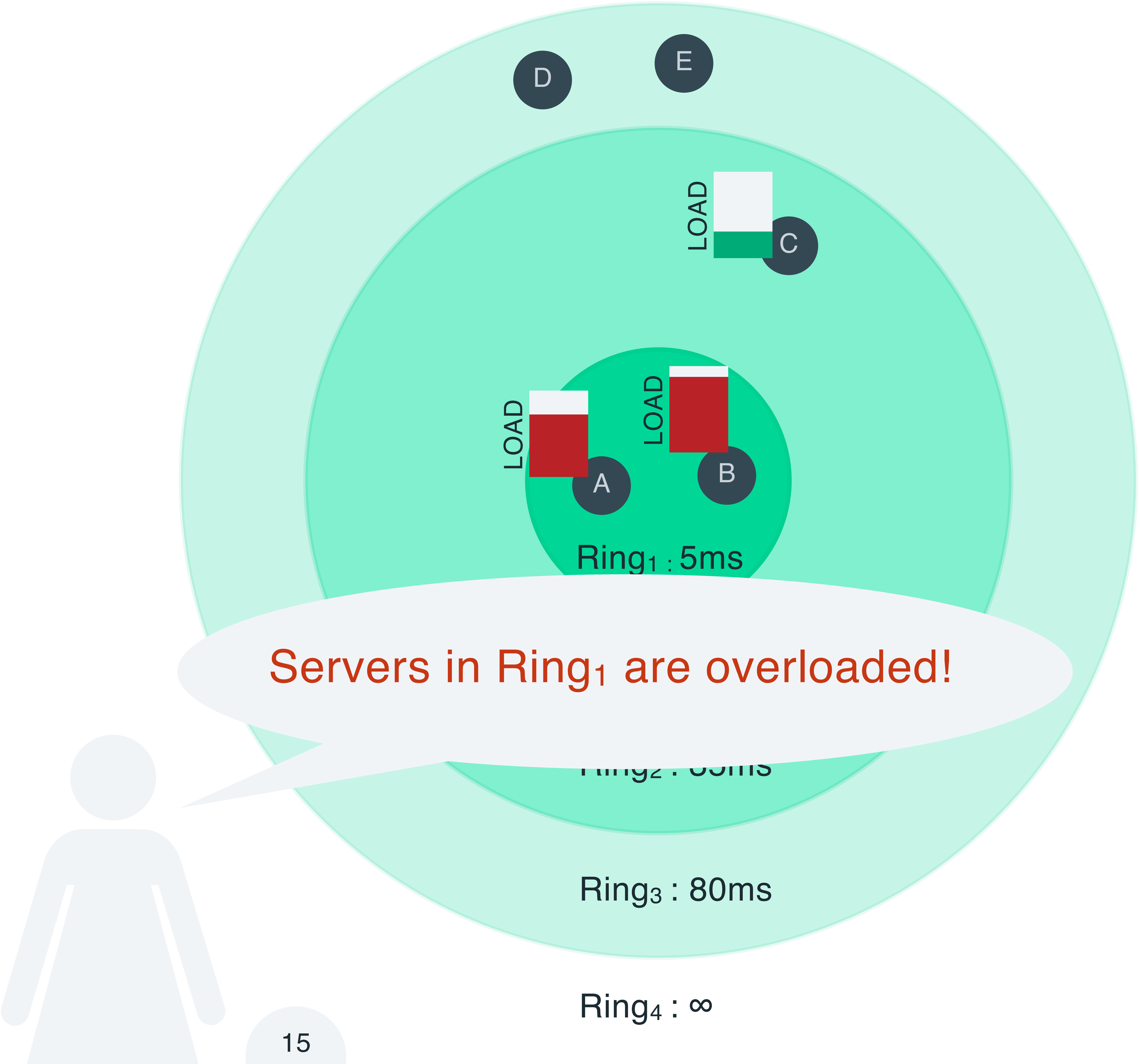
$Ring_1$: 5ms | $Ring_2$: 35ms | $Ring_3$: 80ms | $Ring_4$: ∞

LATENCY RINGS AND CROSS-REGION ROUTING

SR strives to simultaneously minimize RPC latency and balance load across global regions.

- SR introduces the concept of latency rings to minimize latency.
- SR collects per-service global traffic and load information, computes a per-service cross-region routing table, and disseminate it to L7 routers to guide their local routing decisions.

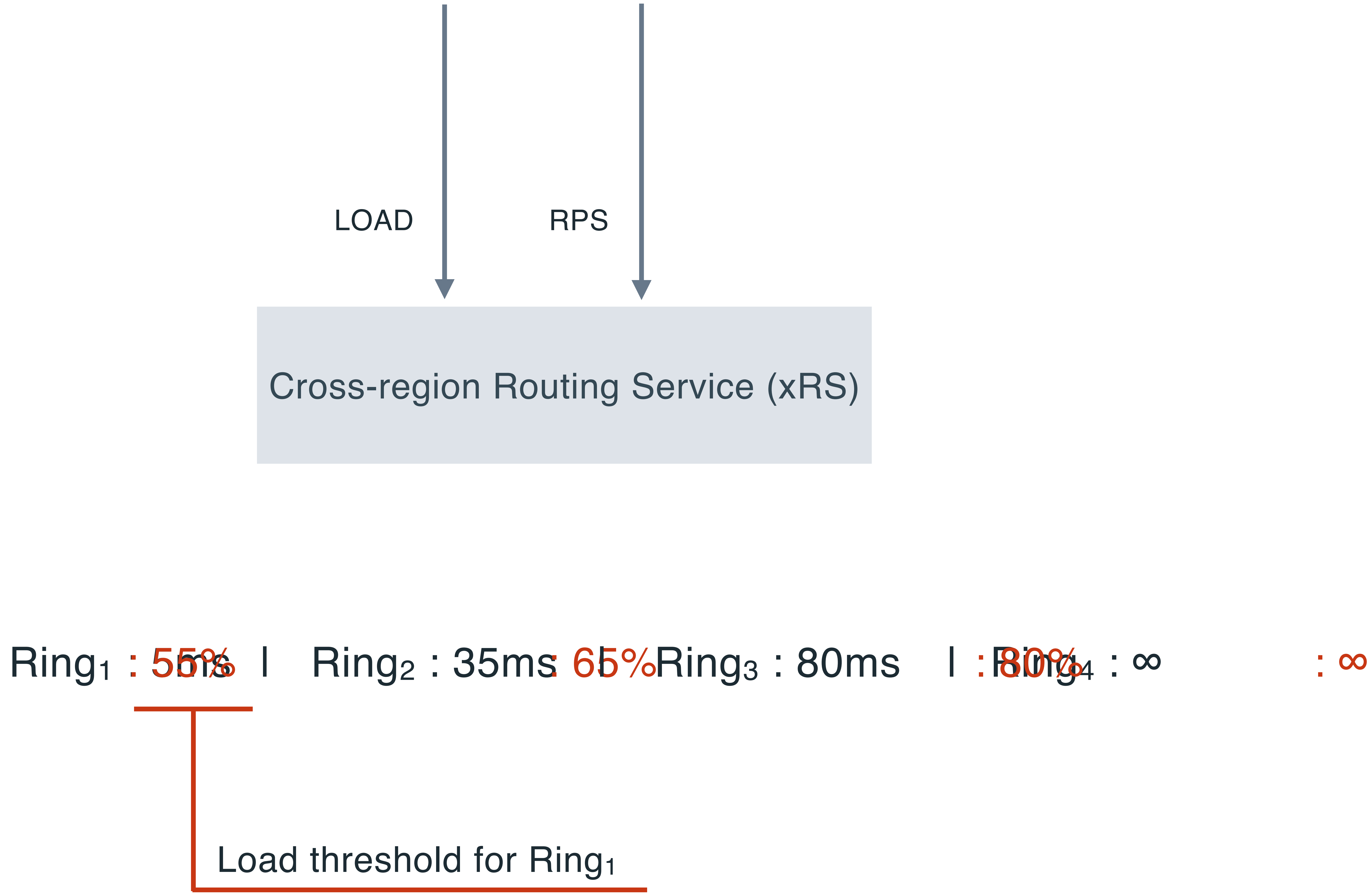
Ring$_1$: 5ms I Ring$_2$: 35ms I Ring$_3$: 80ms I Ring$_4$: ∞



LATENCY RINGS AND CROSS-REGION ROUTING

SR strives to simultaneously minimize RPC latency and balance load across global regions.

- SR introduces the concept of latency rings to minimize latency.
- SR collects per-service global traffic and load information, computes a per-service cross-region routing table, and disseminate it to L7 routers to guide their local routing decisions.



04 ServiceRouter

OVERALL ARCHITECTURE

04 ServiceRouter Architecture

04 ServiceRouter Architecture

04 ServiceRouter Architecture

05 ServiceRouter

EVALUATION

Scalability

Overall scale

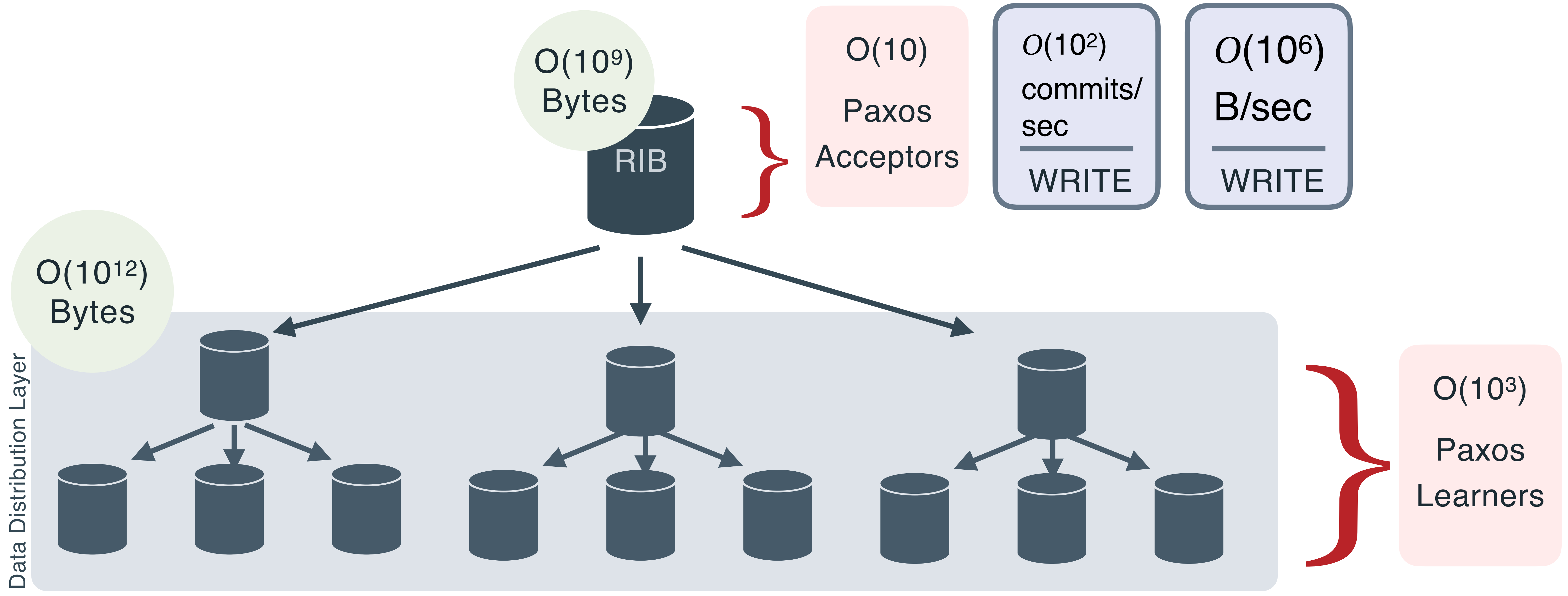
- Regions
- Routers/Clients/Servers
- Throughput

$O(10)$	$O(10^6)$	$O(10^9)$	$O(10^{14})$
Regions	L7 Routers Clients	RPS	B/sec

Scalability

RIB - Routing Information Base

- RIB Replicas
- RIB Write bandwidth
- RIB Write throughput

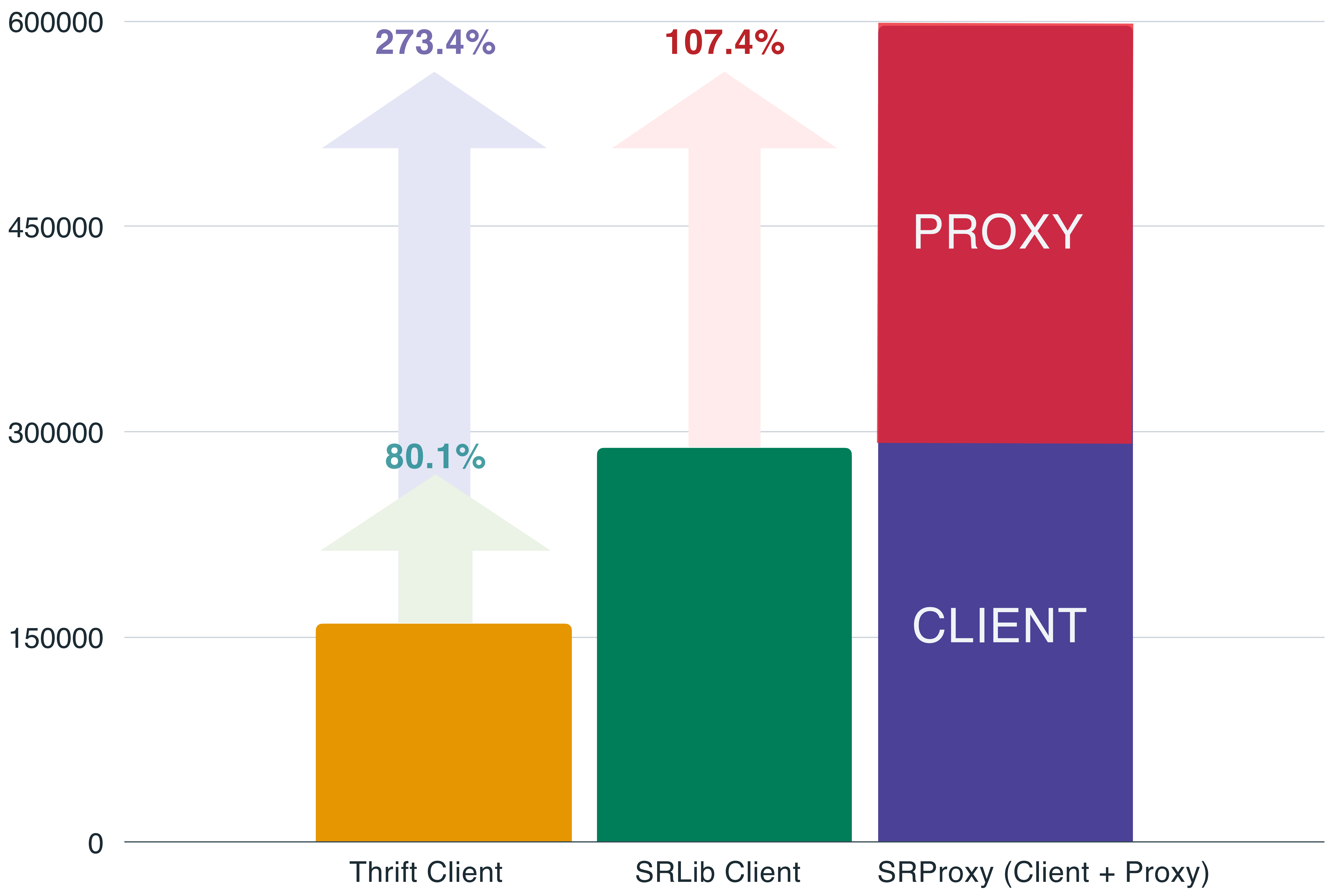


Cost

METHODOLOGY

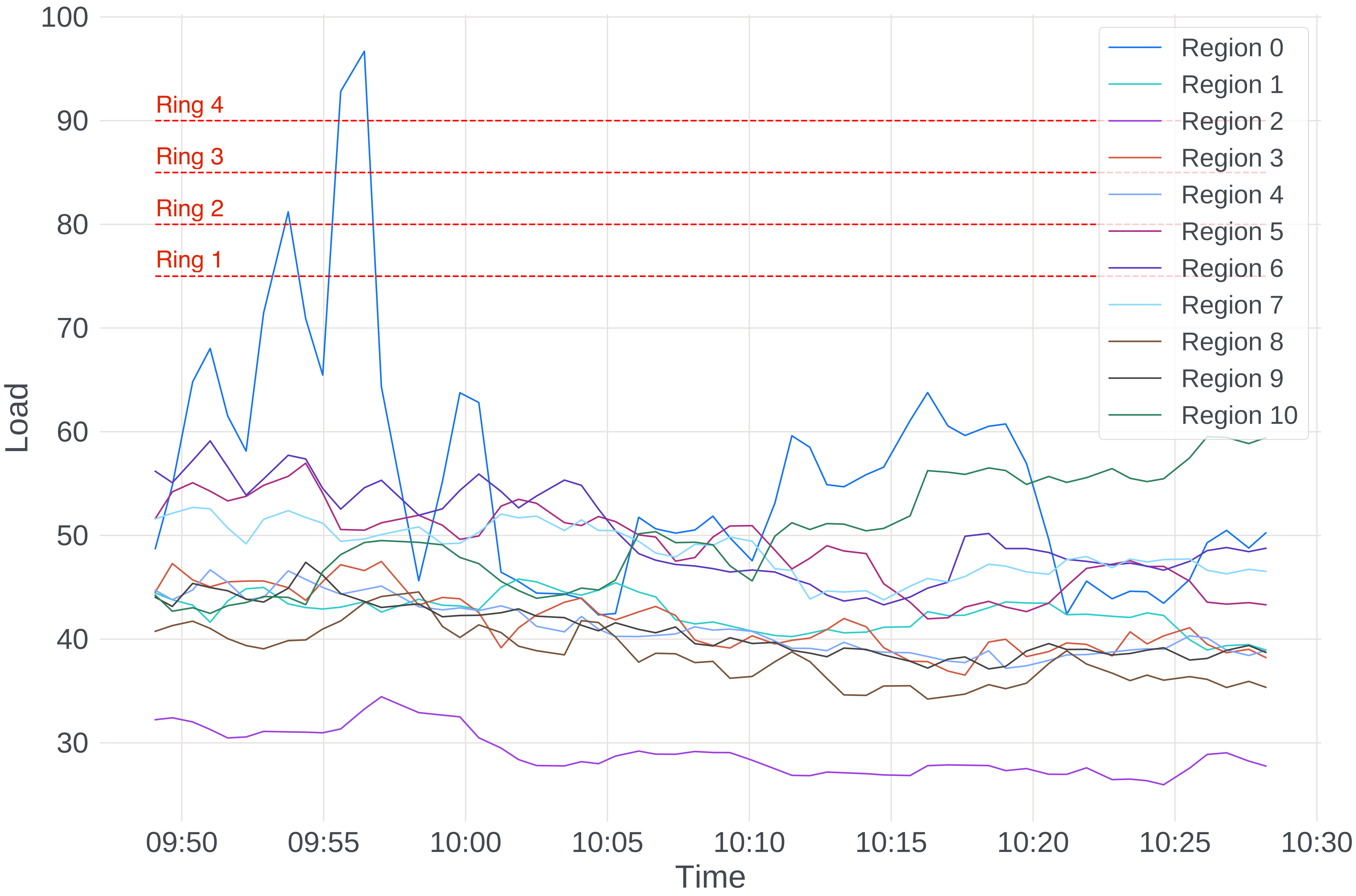
- Metrics: P50 avg request latency; CPU Instructions per request
- Designs
  - Baseline: Thrift RPC
  - SRLib
  - Remote SRProxy
- Simulated Payload:
  - Production avg request and avg response size
    - O(10^3) B
- 100K requests
- 3 trials per design

CPU Instructions/Request



Cross-Region Load Shift

- Real-world Example
- TODO - recreate plot with animated components (maybe just show R0 and R2) and show incident step-by-step. Refine verbal explanation to be time-efficient.



ServiceRouter

HYPERSCALE AND MINIMAL COST SERVICE MESH AT META

06 Summary

ServiceRouter's massive RIB replication allows decentralizing L7 router management and to scale to millions of routers and proxies.

ServiceRouter routes 99% of the traffic with an optimized embedded library approach with astounding HW savings.

ServiceRouter's source-based locality rings and xRS strike a balance between latency wins and load balancing.

Built-in support for sharded services which account for 68% of our RPCs **[not covered in this talk]**.

Soteris Demetriou I s.demetriou@imperial.ac.uk

Meta

Soteris Demetriou | s.demetriou@imperial.ac.uk

0X Design Comparison

Service Mesh Alternatives	A1: HW cost	A2: fast RPC	A3: RPC avail	A4: fast RIB	A5: RIB avail	A6: save mem	A7: adv. LB	A8: mini RIB	A9: unchg code	A10: share conn
Istio [14]	✗	✗	✓	✓	✓	✗	✓	✗	✓	✗
SRLib	✓	✓	✓	✓	✓	≈	✓	✓	✗	✗
SR sidecar proxy	✗	✗	✓	✓	✓	≈	✓	✓	✓	✗
SR remote proxy	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
SR lookaside	≈	≈	✓	✗	✗	✓	✓	✓	✗	✗
eBPF mesh [17]	≈	≈	✓	N/A	N/A	N/A	✗	N/A	✓	✗

Attributes	Description
A1: HW cost	No extra hardware cost for proxy or lookaside service.
A2: fast RPC	No overhead on the critical path of application RPC traffic, thanks to direct RPCs from client to server without the overhead of going through an intermediate proxy.
A3: RPC avail	Higher availability for application RPCs as RPCs do not go through a remote proxy outside the client machine.
A4: fast RIB	No overhead to access Routing Information Base (RIB) outside the client machine thanks to local RIB caching.
A5: RIB avail	Higher availability for application RPCs thanks to access to locally cached RIB without dependency on remote RIB.
A6: save mem	No extra memory usage on the client machine thanks to the elimination of the local RIB cache.
A7: adv. LB	Support complex load-balancing algorithms.
A8: miniRIB	Low overhead in replicating and caching RIB thanks to only fetching the actively used parts of RIB.
A9: unchg code	No need for application source code modification.
A10: share conn	Benefits of multiple clients sharing a proxy, e.g., better load balancing or connection reuse (Figure 6).

0X Measured limitations of sidecar

Zhu et al show that Istio

- adds 92% extra CPU usage
- increases the latency by 185%

Xiangfeng Zhu, Guozhen She, Bowen Xue, Yu Zhang, Yongsu Zhang, Xuan Kelvin Zou, Xiongchun Duan, Peng He, Arvind Krishnamurthy, Matthew Lentz, Danyang Zhuo, and Ratul Mahajan. Dissecting Service Mesh Overheads. In *arXiv preprint arXiv:2207.00592*, 2022.

mRPC shows that a sidecar approach:

- increases P99 RPC latency by 180%
- decreases throughput by 44%

JingrongChen,YongjiWu,ShihanLin,YechenXu,Xin- hao Kong, Thomas Anderson, Matthew Lentz, Xiaowei Yang, and Danyang Zhuo. Remote procedure call as a managed system service. In *20th USENIX Sympo- sium on Networked Systems Design and Implementation (NSDI 23)*, pages 141–159, Boston, MA, April 2023. USENIX Association.

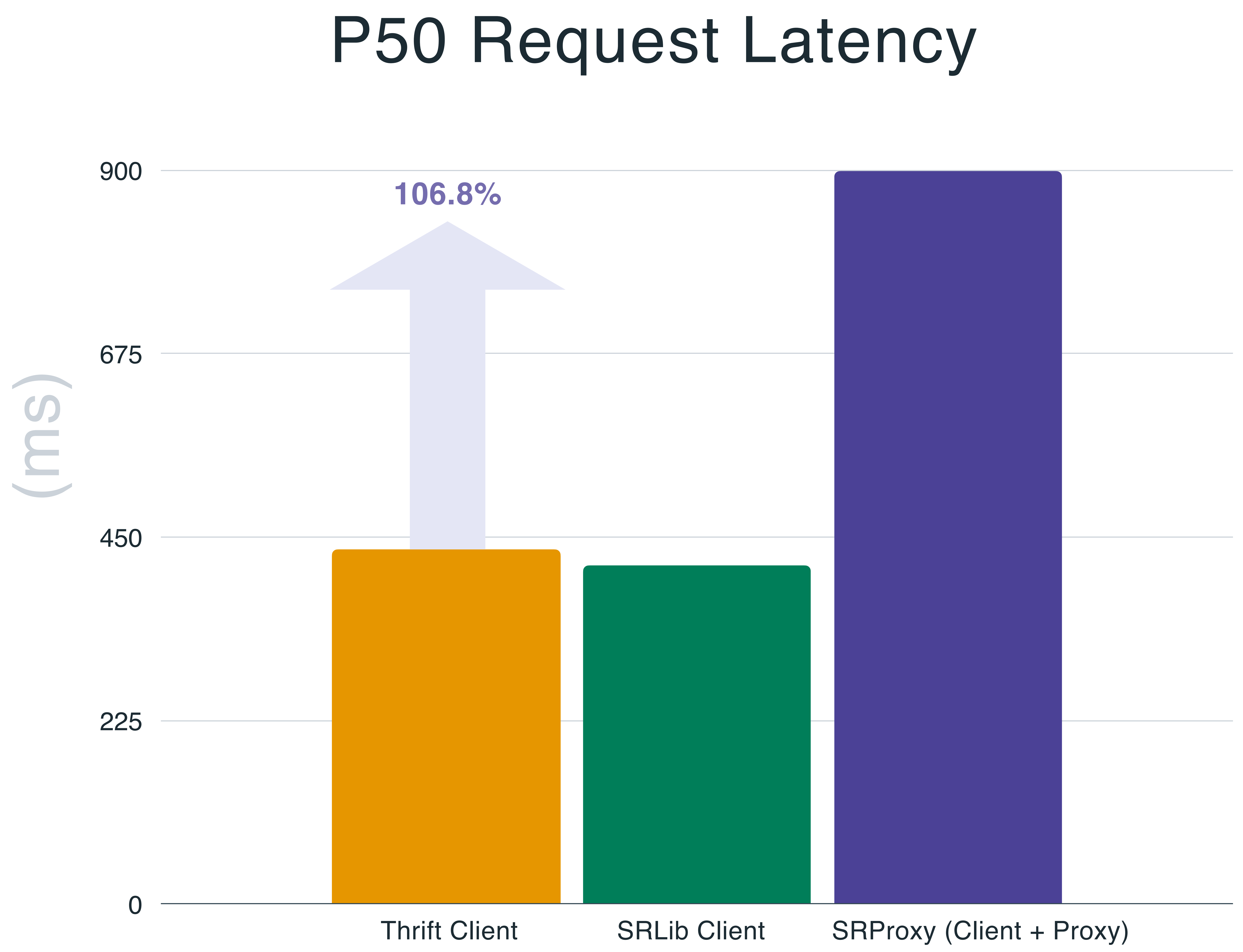
Evaluation - Scalability

Components	Scale and Comment
Datacenter regions	$O(10)$
L7 routers	$O(10^6)$
RPC clients	$O(10^6)$
RPC servers	$O(10^6)$
RPCs per second	$O(10^9)$
GRS_d	$O(10^6)$. The aggregate peak bandwidth consumption of all GRS_d is $O(10)$ TB/sec. This demonstrates the importance of decentralizing part of the control plane and making L7 routers self-managing in order to scale.
SRProxy machines	$O(10^3)$. Currently, 99% of our RPC traffic is routed by SRLib while the rest 1% is mostly routed by $O(10^3)$ SRProxies. If 100% of our traffic were to be routed by SRProxy without using SRLib, it would require $O(10^5)$ SRProxy machines, a hefty hardware cost.
Cluster managers	$O(10^2)$
Shard managers	$O(10^2)$
xRS machines	$O(10^2)$
LMS machines	$O(1)$
CMS machines	$O(10)$
RIB size	$O(10^9)$ Bytes for Paxos acceptors; $O(10^{12})$ Bytes for distribution layer; $O(10^{15})$ Bytes for GRS_d
Write bandwidth to the RIB master	$O(10^6)$ Bytes/sec
Write throughput to the RIB master	$O(10^2)$ commits/second
RIB replicas	$O(10^3)$

Cost

METHODOLOGY

- Metrics: P50 avg request latency; CPU Instructions per request
- Designs
  - Baseline: Thrift RPC
  - SRLib
  - Remote SRProxy
- Simulated Payload:
  - Production avg request and avg response size
    - O(10^3) B
- 100K requests
- 3 trials per design

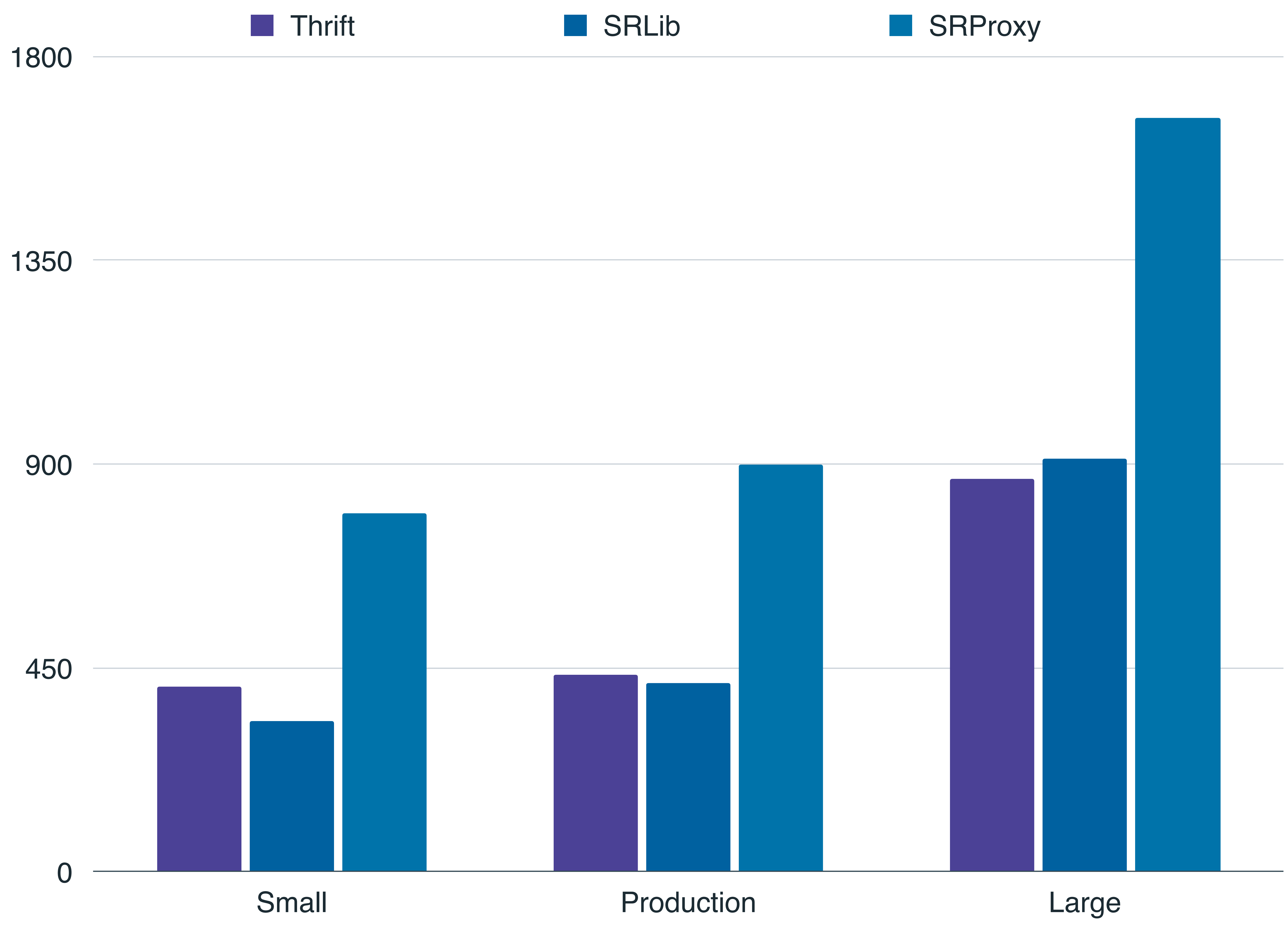


Cost

METHODOLOGY

- Metrics: P50 avg request latency; CPU Instructions per request
- Period: 1 day
- Designs
  - Baseline: Thrift RPC
  - SRLib
  - Remote SRProxy
- Simulated Payload:
  - Small: 10^{-1}x of production
  - Production: Avg request and response size
  - Large: 10x of Production
- 100K requests
- 3 trials per design

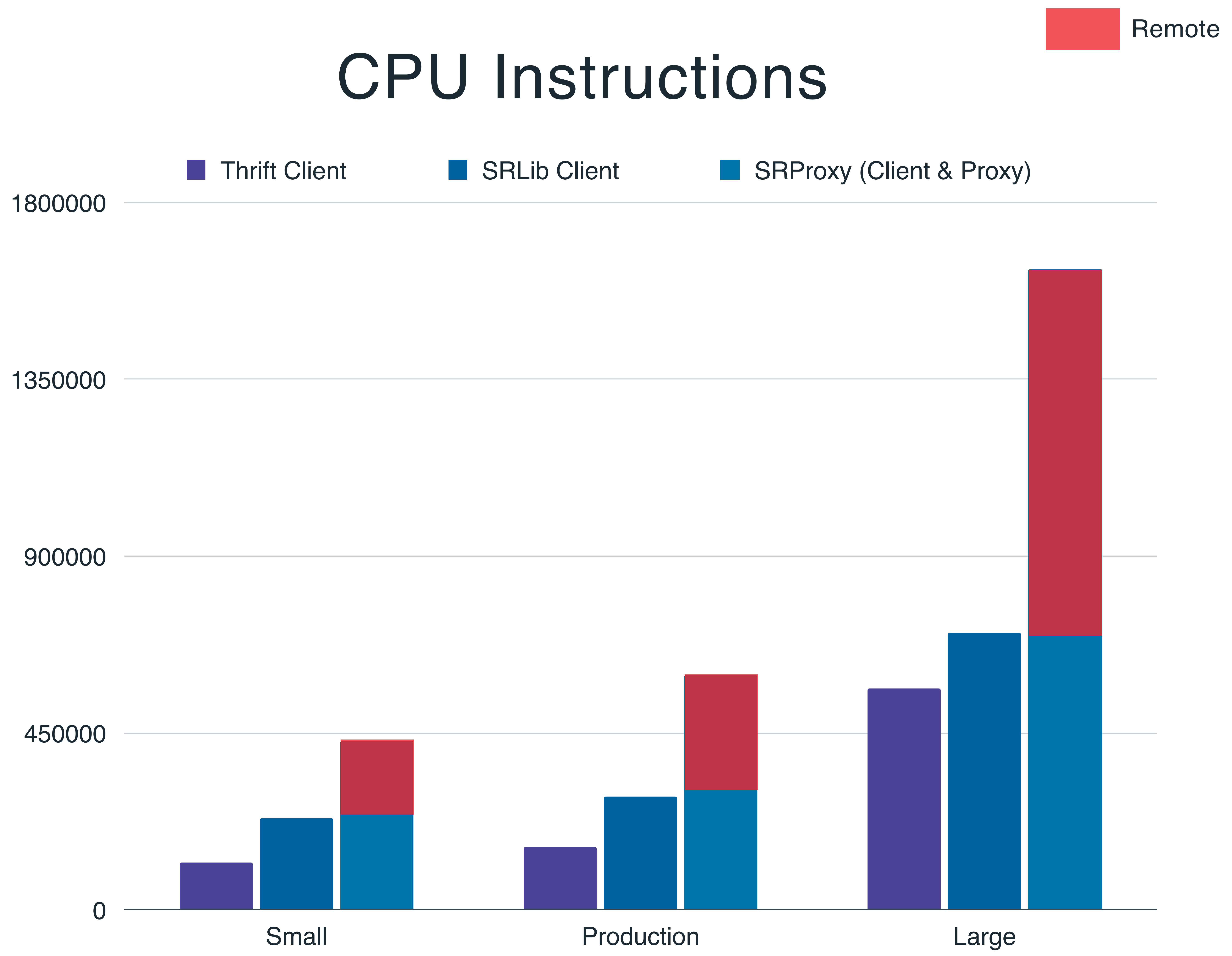
P50 Request Latency



Cost

METHODOLOGY

- Metrics: P50 avg request latency; CPU Instructions per request
- Period: 1 day
- Designs
  - Baseline: Thrift RPC
  - SRLib
  - Remote SRProxy
- Simulated Payload:
  - Small: 10^{-1}x of production
  - Production: Avg request and response size
  - Large: 10x of Production
- 100K requests
- 3 trials per design



0X Support for sharding services

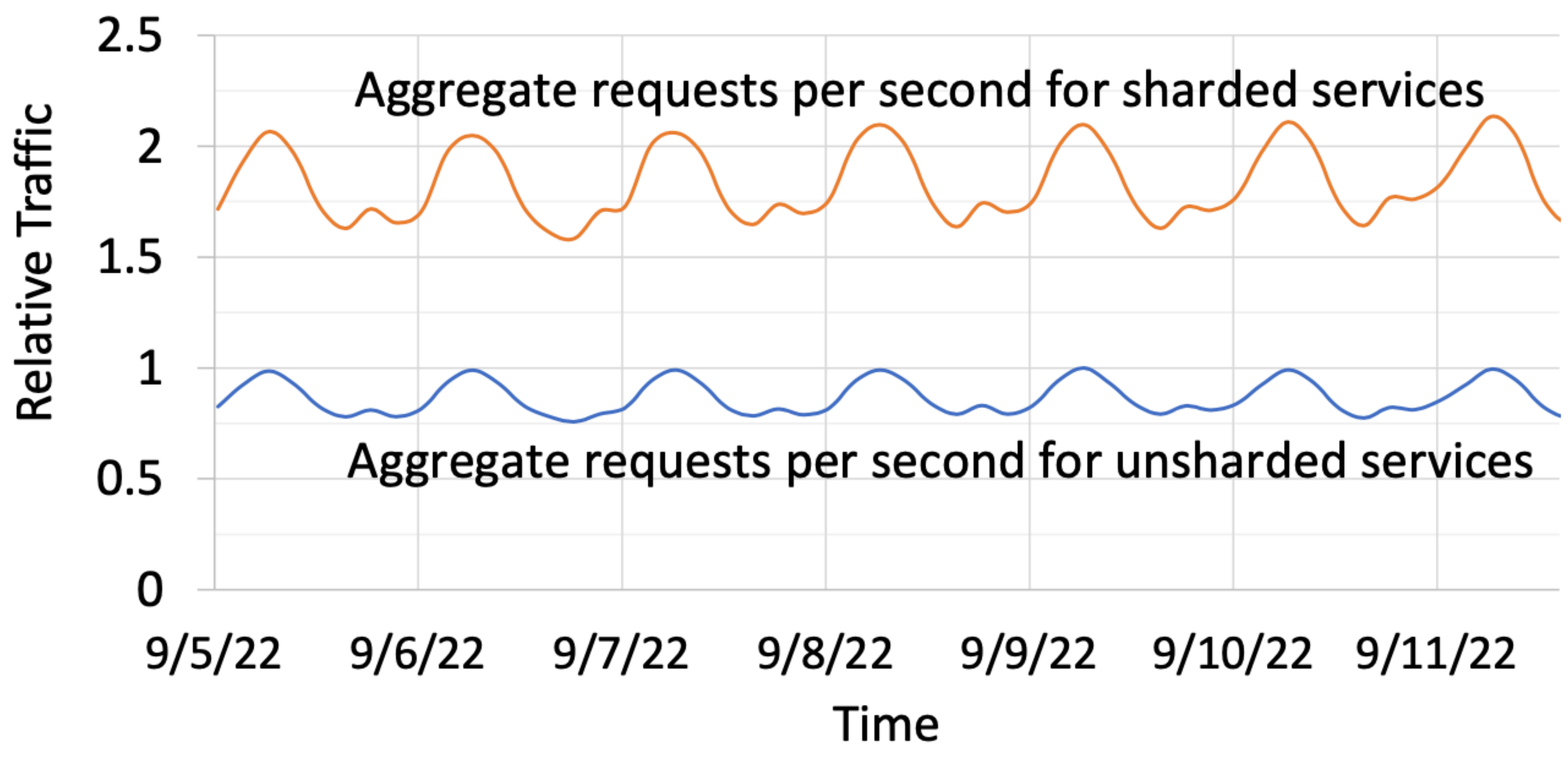


Figure 13: Total traffic for sharded vs. unsharded services.

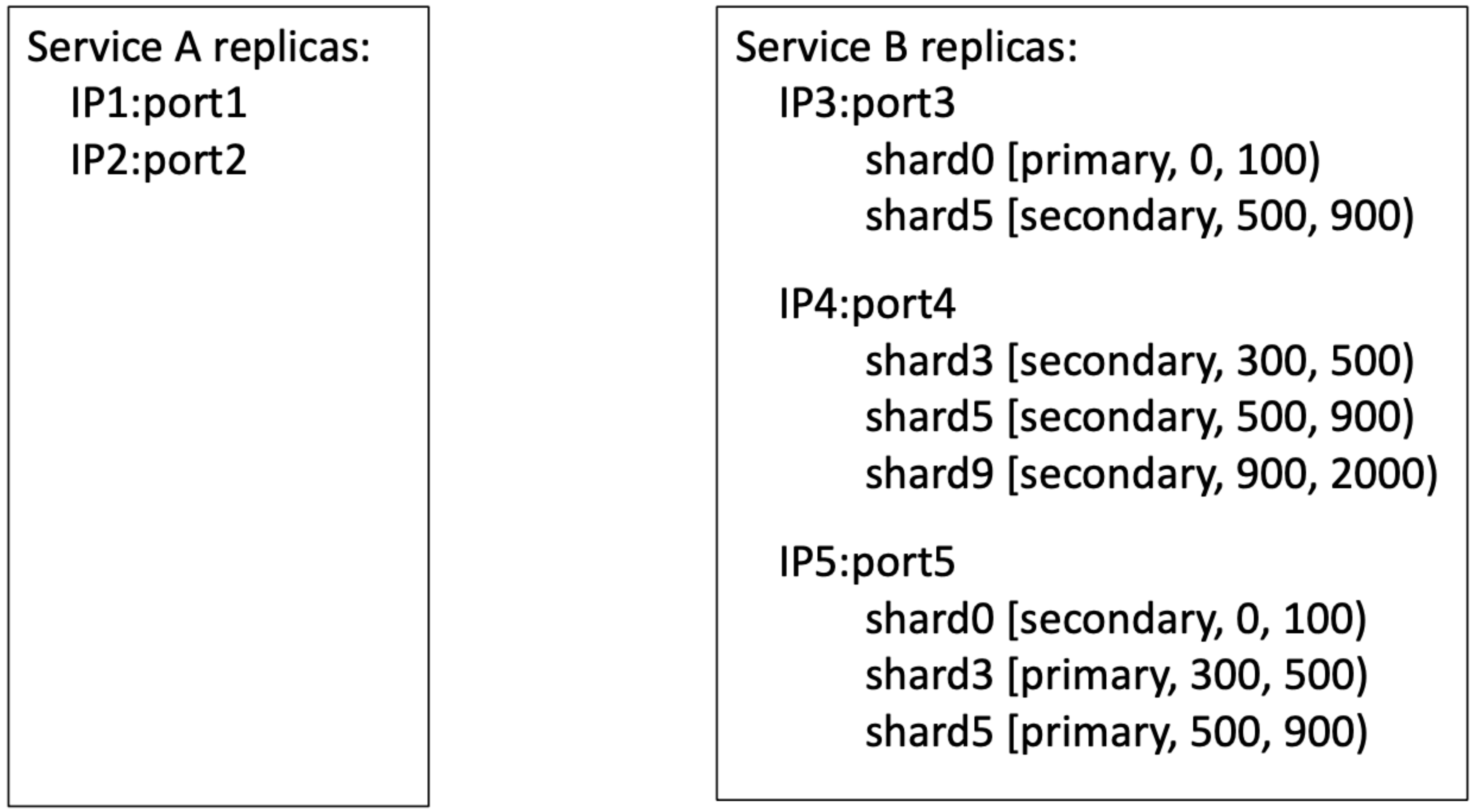
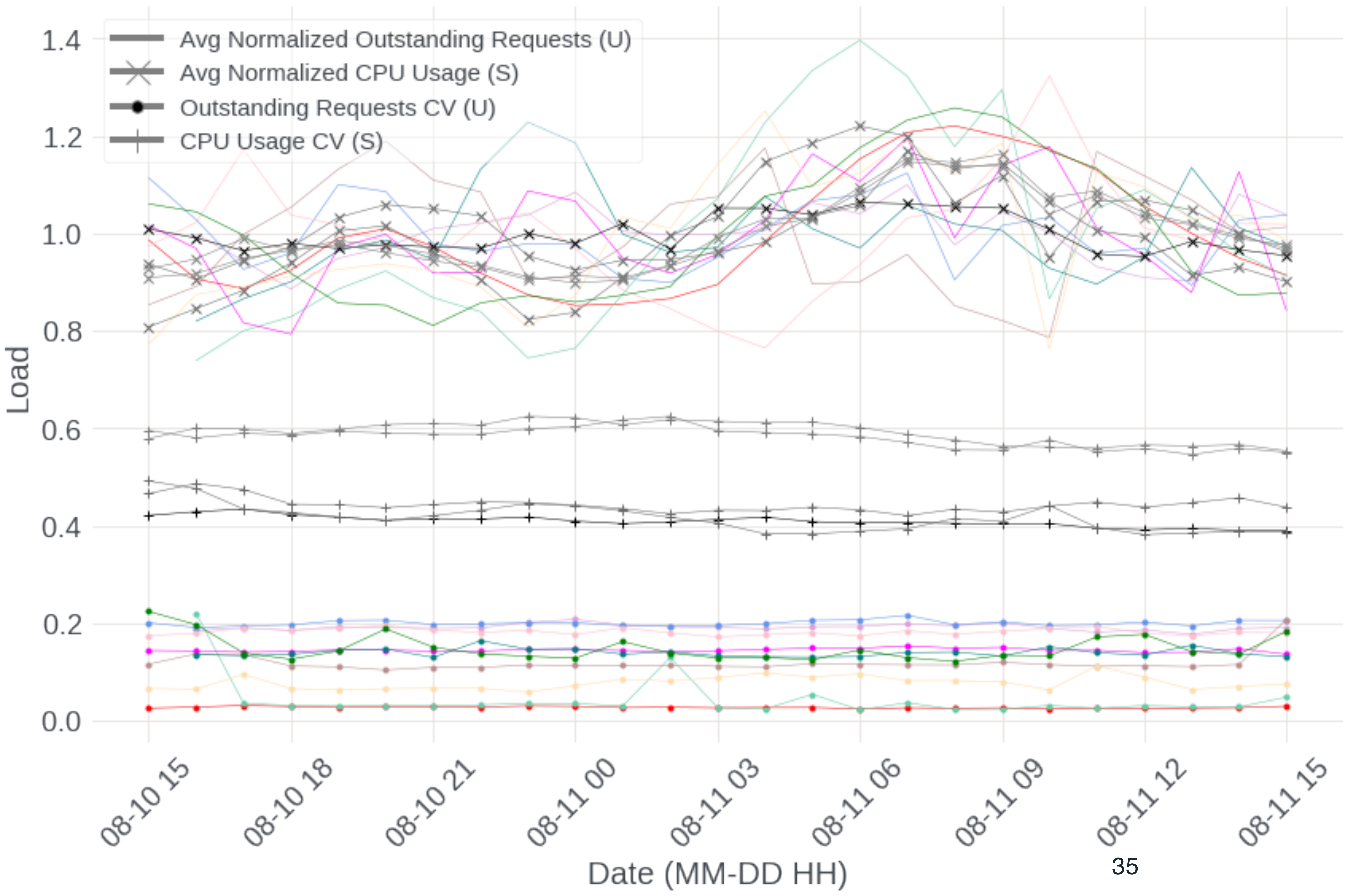


Figure 8: Examples of GRS's service registry records.

```
SRClient *cln = SR_get_client("ServiceB", 618/*key*/, SECONDARY); cln->foo(); // Call RPC foo().
```

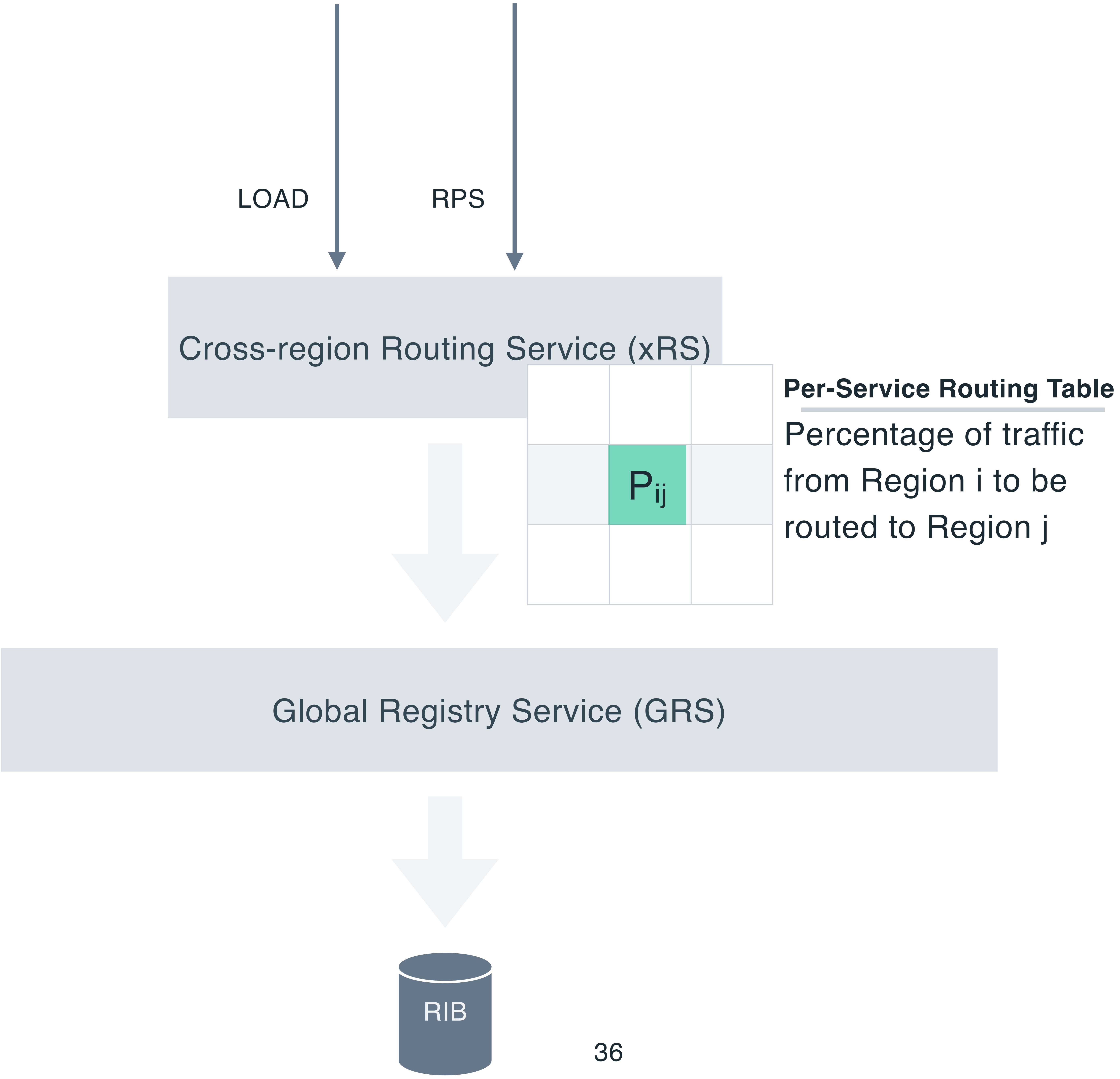


Avg Normalized Outstanding Requests (U)
Avg Normalized CPU Usage (S)
Outstanding Requests CV (U)
CPU Usage CV (S)
Load
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
08-10 15
08-10 18
08-10 21
08-11 00
08-11 03
08-11 06
08-11 09
08-11 12
08-11 15
Date (MM-DD HH)

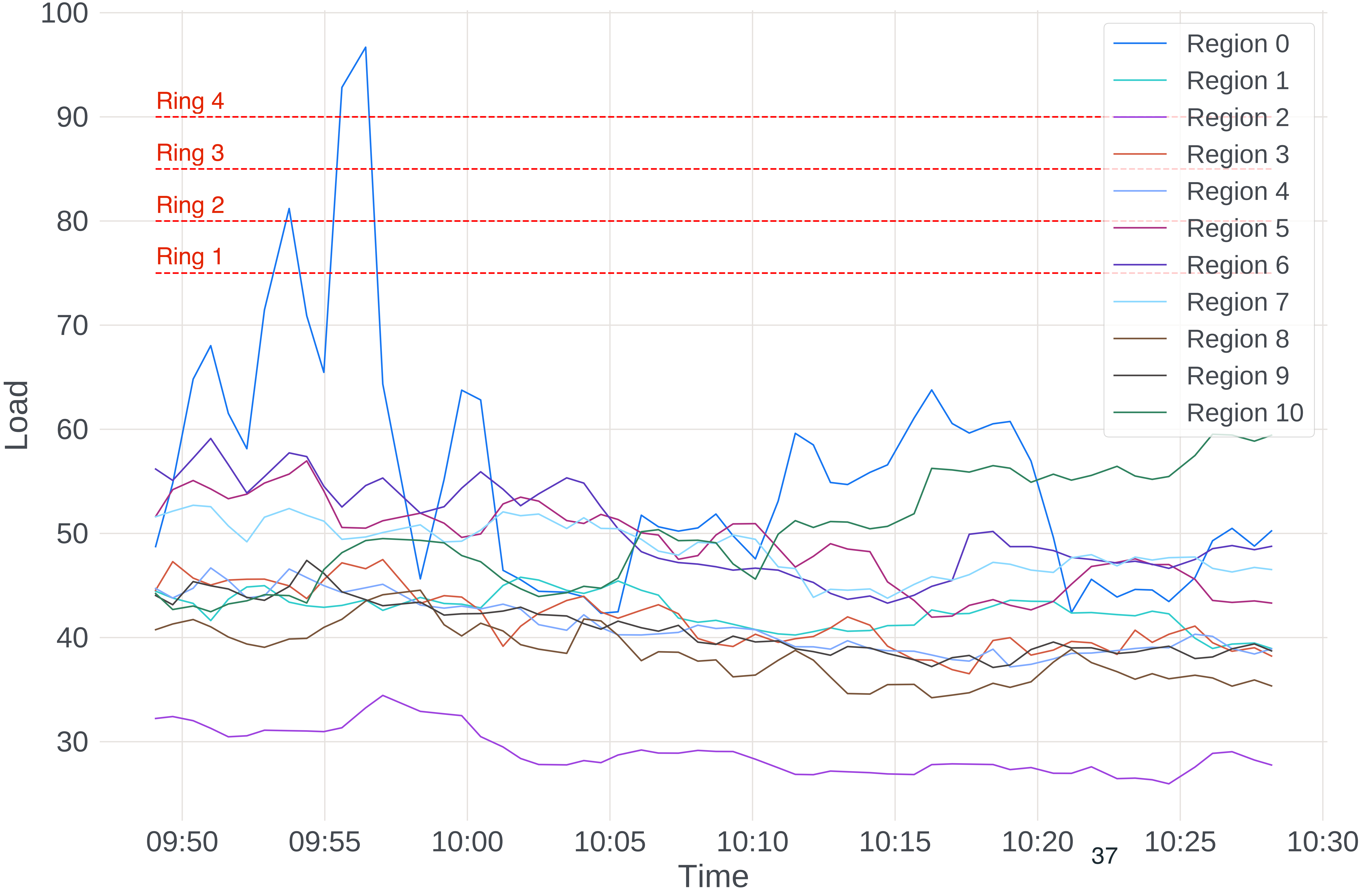
LATENCY RINGS AND CROSS-REGION ROUTING

SR strives to simultaneously minimize RPC latency and balance load across global regions.

- SR introduces the concept of latency rings for services to express their preferred tradeoff between latency and load.
- SR collects per-service global traffic and load information, computes a per-service cross-region routing table, and disseminate it to L7 routers to guide their local routing decisions.



xRS: Cross-routing Service Example



RPC Connection Reuse