Characterizing Retry Policies for Microservice Applications

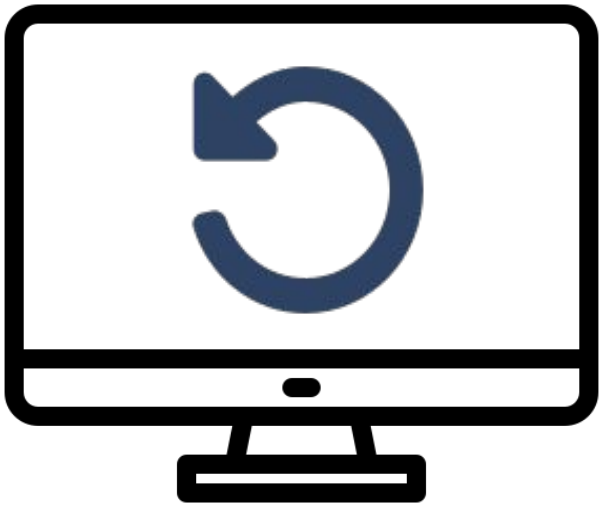
Alan Song (PRIMES CS)

Mentor: Lisa (Yueying) Li

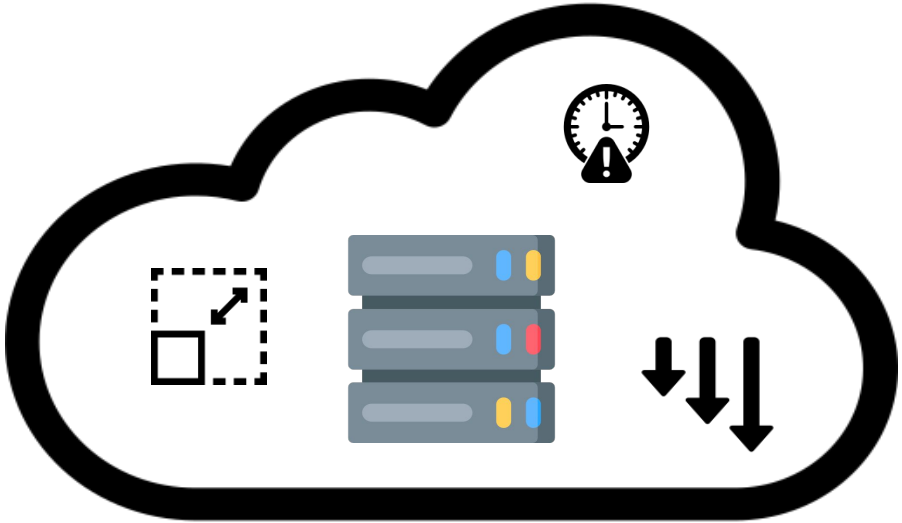
What are Microservices?

Microservice Graphs

Fault Tolerance Measures

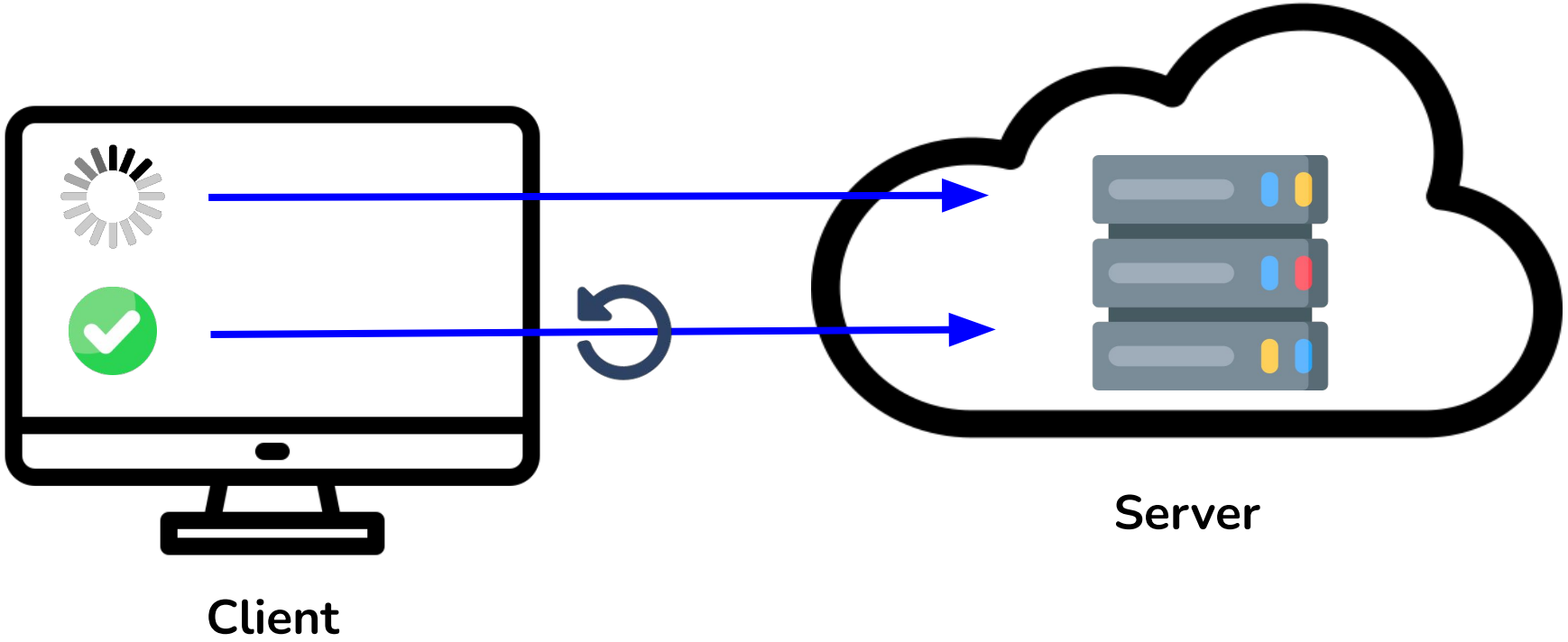


Client



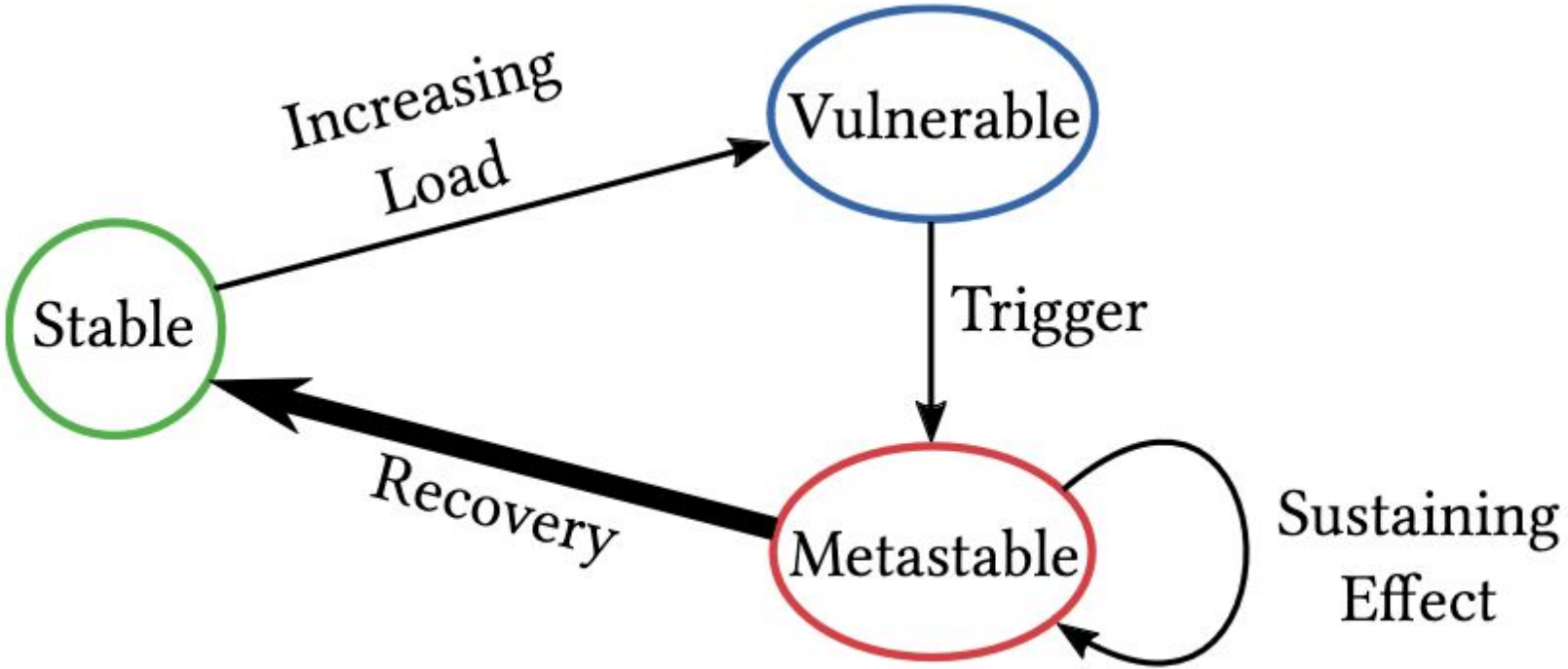
Server

How do retry mechanisms work?



Retry Storms

Metastable Failures



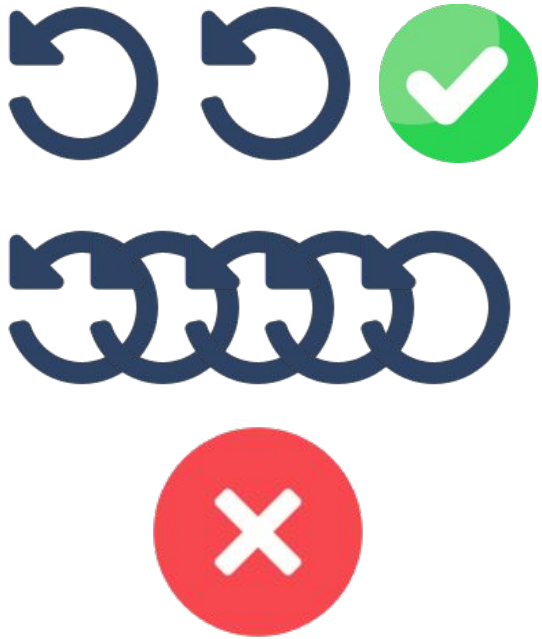
Metastable Failures in the Wild (Huang, et. al)

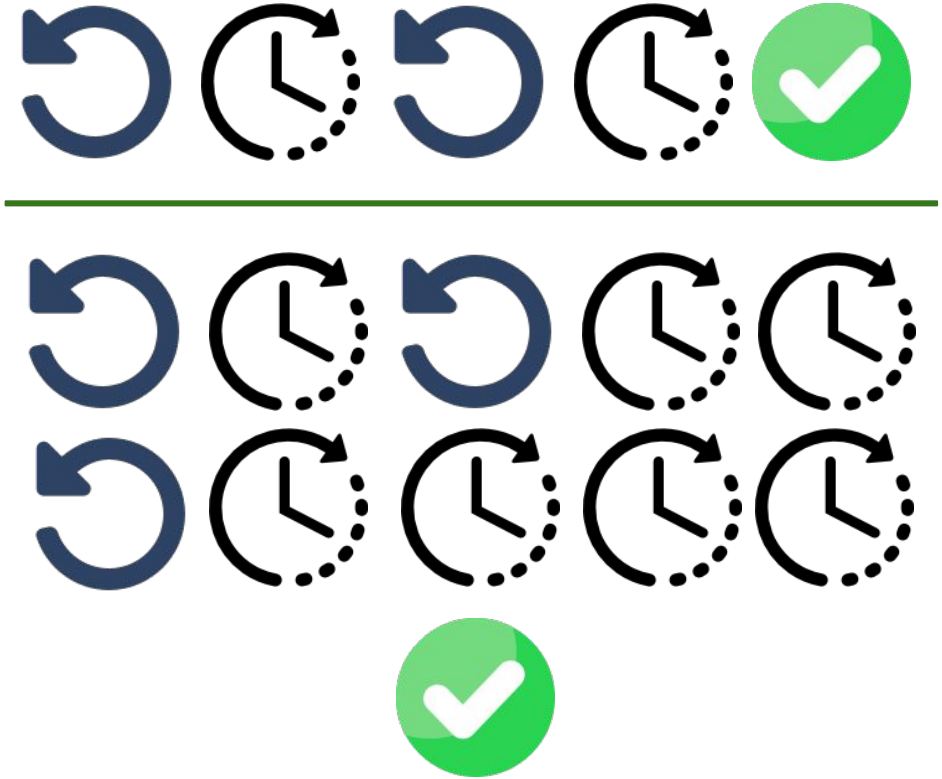
	ID	Date	Duration (hours)	Services Impacted	Triggers	Sustaining Effect	Mitigation
Google	GGL1 [22]	03/12/19	4.17	Gmail, Photos, Drive, Cloud Storage, various other GCP services	• load spike • config change	• cascading overload	• load shedding • stop config deploy
	GGL2 [23]	10/31/19	21.5	multiple components of GCE	• software bug	• retry	• load shedding • reboot • capacity increase
	GGL3 [24]	04/08/20	3.2	Google BigQuery, Cloud IAM 3% of Cloud SQL HA	• config change • software bug	• retry	• config rollback • policy change
	GGL4 [21]	04/30/13	1.5	Google API infrastructure	• config change • latent software bug	• traffic queue growth • reboots	• config rollabck • server reboot
AWS	AWS1 [47]	04/21/11	66.7	Amazon EC2, Amazon RDS	• network config change	• retry	• config rollback • policy change • load shedding • capacity increase
	AWS2 [48]	06/13/14	4.23	Amazon SimpleDB	• power loss	• retry	• load shedding • server restart
	AWS3 [49]	09/20/15	4.55	AWS SQS, EC2 Autoscaling, CloudWatch, AWS Console	• load spike • network disruption	• retry • cascading server demotion	• load shedding – pause metadata ops • capacity increase
	AWS4 [51]	12/07/21	9.3	AWS DynamoDB, EC2, Fargate, RDS, EMR, Workspaces, AWS Console, Authorization services, internal DNS	• latent software bug triggered by scale-up led to load spike	• retry	• load rebalancing • load shedding
Azure	AZR1 [4]	07/01/20	2.65	Azure SQL DB & SQL Data Warehouse, Azure Database for MySQL/PostgreSQL/MariaDB	• unspecified load imbalance trigger • latent config bug	• cascading overload	• service restart
	AZR2 [4]	04/01/21	1.15	Azure DNS	• software bug leading to cache degradation	• retry	• unknown automation • capacity increase
	AZR3 [4]	06/14/21	13.25	Management operations of many Azure Services	• latent software bug • load spike	• unspecified queue growth due to overload and timeouts	• load shedding • remove buggy software • capacity increase
	AZR4 [4]	07/12/21	7.92	Windows Virtual Desktop, Azure Front Door, Azure CDN Standard	• deployment of software bug • load spike	• retry • other unspecified	• load rebalancing • trigger hot fix • policy change
Other	IBM1 [11]	06/11/21	73.53	Private DNS, HS Crypto Service, Cloudant DNS Services, Osaka, Cloudshell services	• software bug	• retry	• load shedding • policy change • trigger hot fix
	SPF1 [19]	04/13	NA	core app/service UI	• load spike • policy failure	• retry	• load shedding
	SPF2 [19]	06/04/13	8.33	core app/service UI	• load spike due to unexpected service dependency	• retry • excessive logging in failure case	• trigger hot fix • load shedding
	ELC1 [39]	04/02/19	6.67	Elasticsearch Service	• unspecified maintenance • unspecified error	• load caused ZK churn causing more load	• restart • load shedding
	WIK1 [58]	03/30/21	2.25	media upload, misc queued jobs	• load spike	• unspecified causing queue growth	• load shedding • policy change
	CCI1 [10]	07/07/15	18.33	Core product	• load spike	• load increase due to contention	• load shedding
	CAS1 [1]	07/27/17	NA	Partial database outage	• rolling restart	• self-sustaining and increasing overload	• policy change
	CAS2 [43]	2020	0.16	ably services	• load spike of certain costly operations	• retry	• trigger removal – operated in stable state
	FB1 [18]	NA	NA	Facebook core services	• load spike	• software bug	• hot fix

Table 1: Metastable failures from public sources. Azure and IBM do not provide a direct incident link. Gray highlight indicates a plausible metastable failure, although the incident description lacked some necessary details.

Retries are by far the most common sustaining effect!

Retry Policies



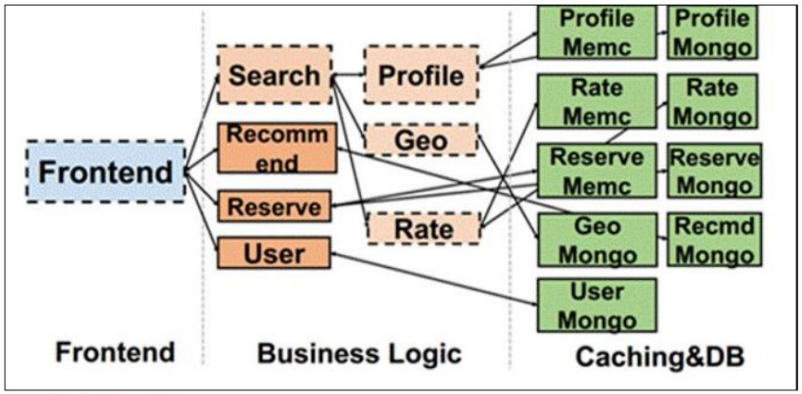


gRPC retry options

The following table describes options for configuring gRPC retry policies:

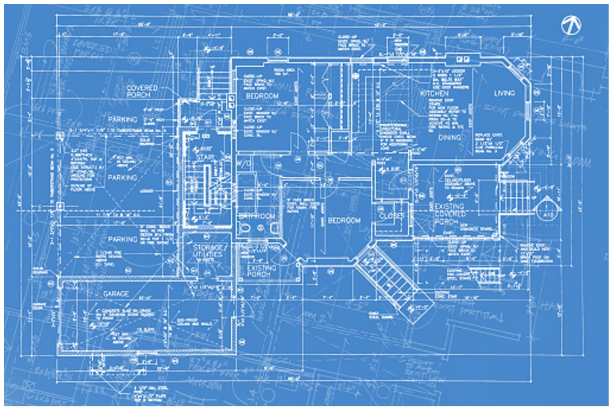
Option	Description
`MaxAttempts`	The maximum number of call attempts, including the original attempt. This value is limited by `GrpcChannelOptions.MaxRetryAttempts` which defaults to 5. A value is required and must be greater than 1.
`InitialBackoff`	The initial backoff delay between retry attempts. A randomized delay between 0 and the current backoff determines when the next retry attempt is made. After each attempt, the current backoff is multiplied by `BackoffMultiplier`. A value is required and must be greater than zero.
`MaxBackoff`	The maximum backoff places an upper limit on exponential backoff growth. A value is required and must be greater than zero.
`BackoffMultiplier`	The backoff will be multiplied by this value after each retry attempt and will increase exponentially when the multiplier is greater than 1. A value is required and must be greater than zero.
`RetryableStatusCodes`	A collection of status codes. A gRPC call that fails with a matching status will be automatically retried. For more information about status codes, see Status codes and their use in gRPC. At least one retryable status code is required.

Testbed Setup



DSB Hotel Reservation

DeathStarBench [hotel-reservation]



[Blueprint] (Anand et. al)

[wrk2] workload generator

Retry Mechanism Implementation

Fixed (Power-of-d) Retry:
- Launch a constant of d copies of request without delay

Fixed-interval Retry with Max Attempts
- Launch a retry with constant delay and max attempts

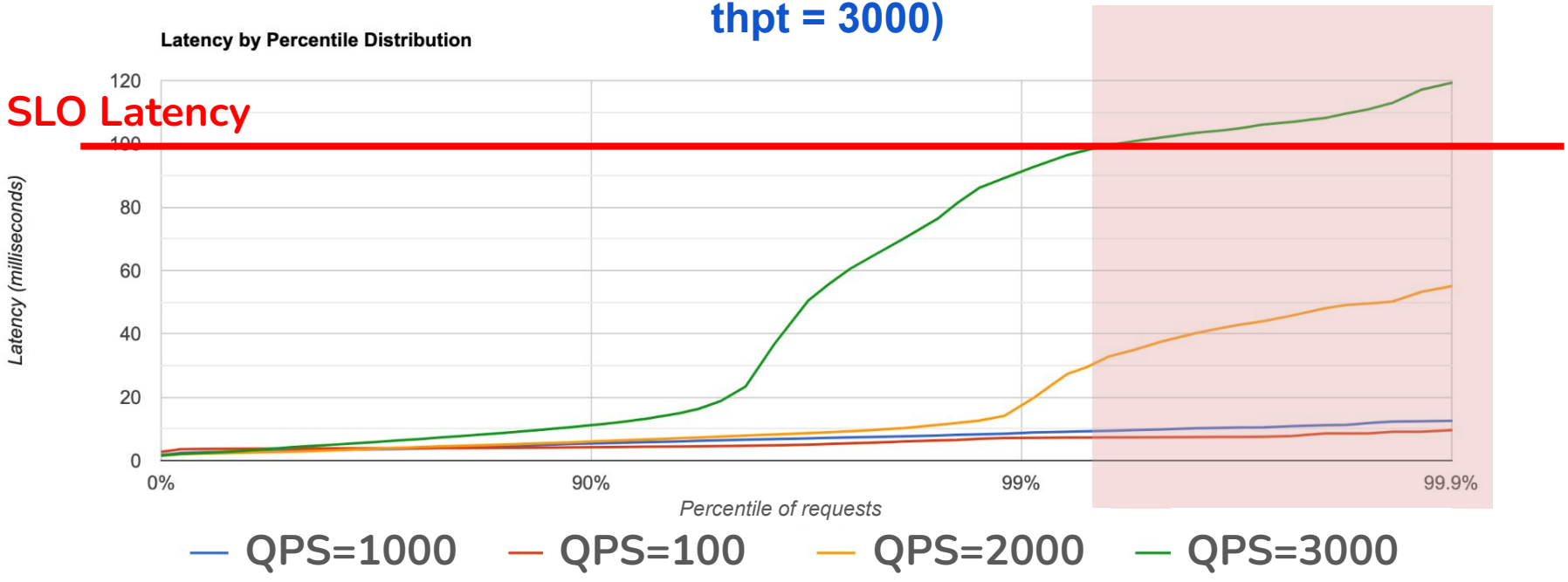
Exponential Backoff Policy (with random jitter)
- Launch retries with exponentially increased time interval

Implemented by us

Results: Limitations of Fixed Retry

Max Retries = 10

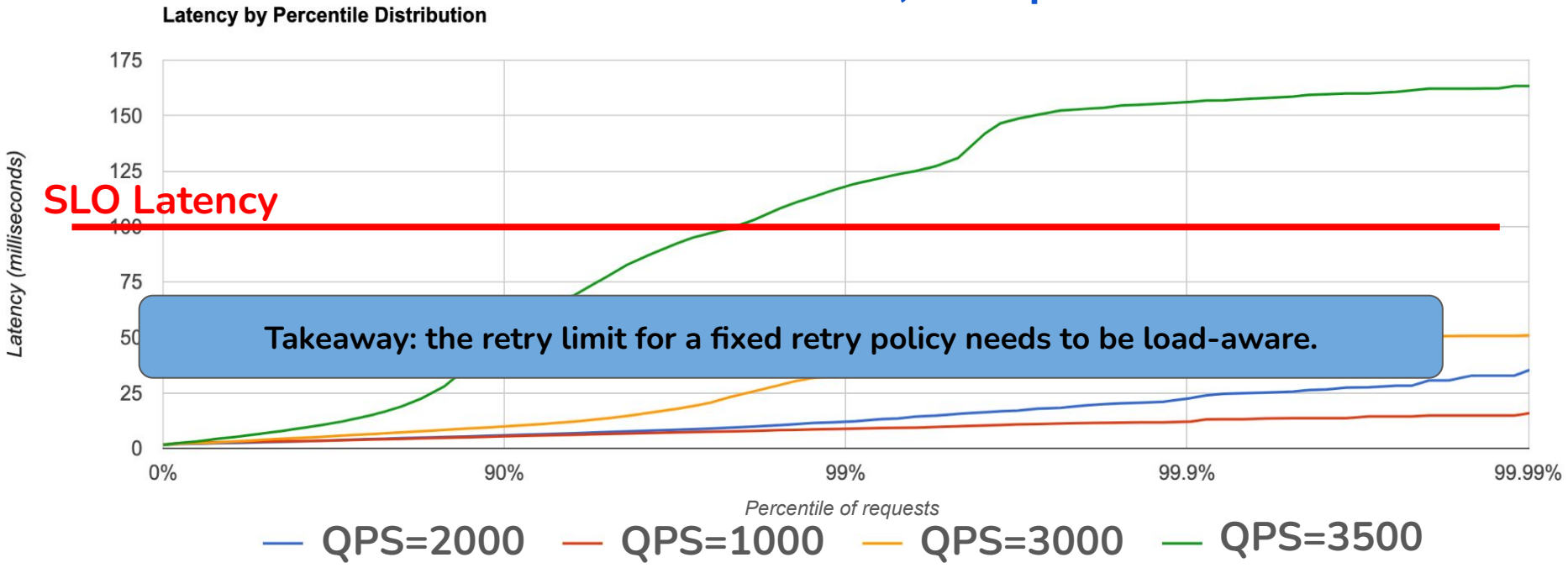
Amplification with bad retry policy (max thpt = 3000)



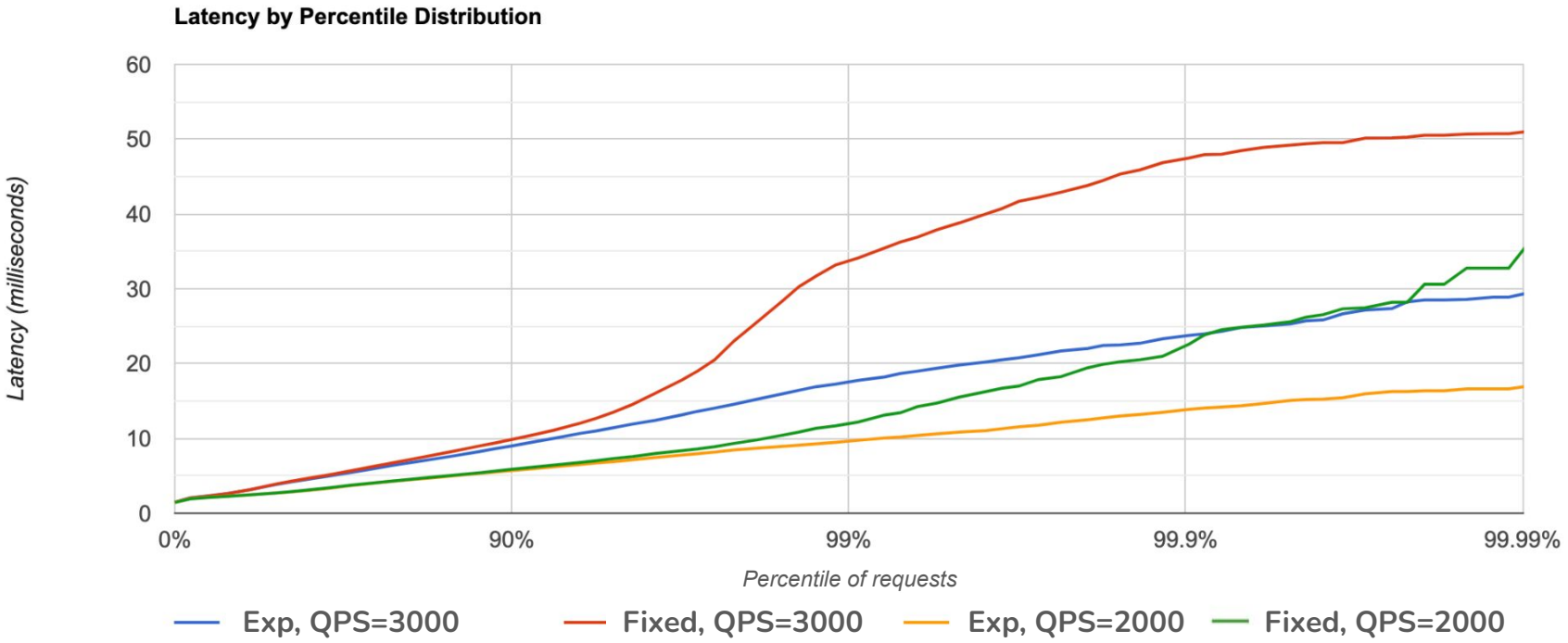
Results: Limitations of Fixed Retry

Max Retries = 2

Tail less amplified - higher throughput (3500)
For 3000 RPS, all requests are under SLO lat



Results: Exponential Retry



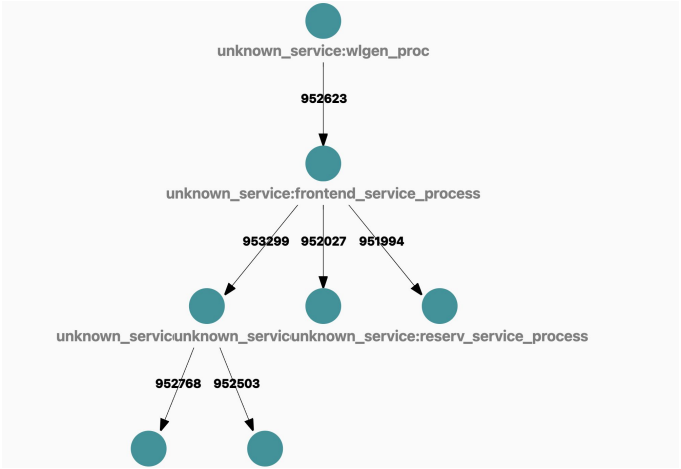
Takeaway: exponential retry is more resilient than fixed retry under load-increase triggers

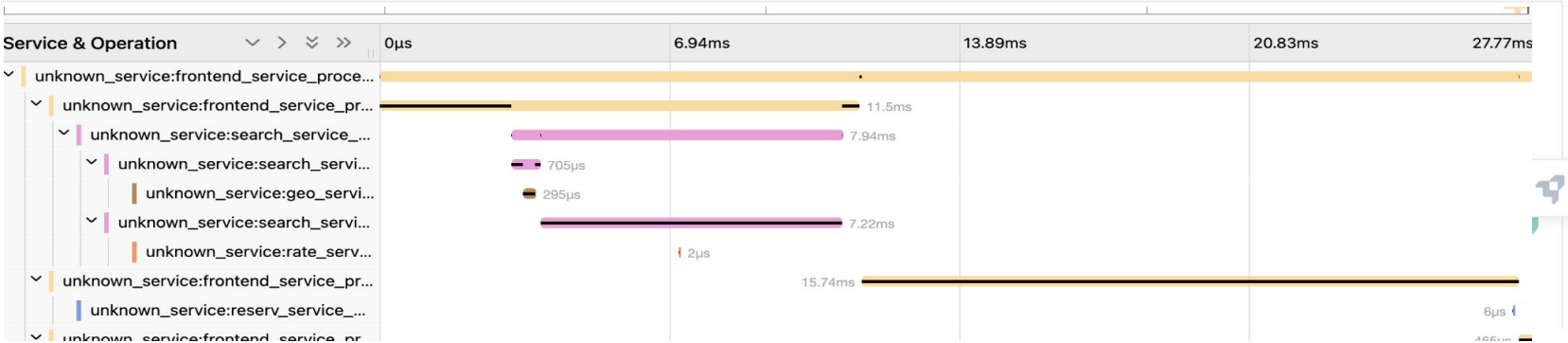
Telemetry

Dependency graph to identify percentage of retries

Latency graph to color retried requests

Call span of different tiers of services to identify triggers





Future Work

1) Study the relationship between retry policies and rate limiting policies

2) Study retries caused by different triggers like capacity degradation

3) Expand beyond simple retry policies by exploring learning-based retry mechanisms

Key Takeaways

1) Microservices is an important software architecture that demands high fault tolerance.

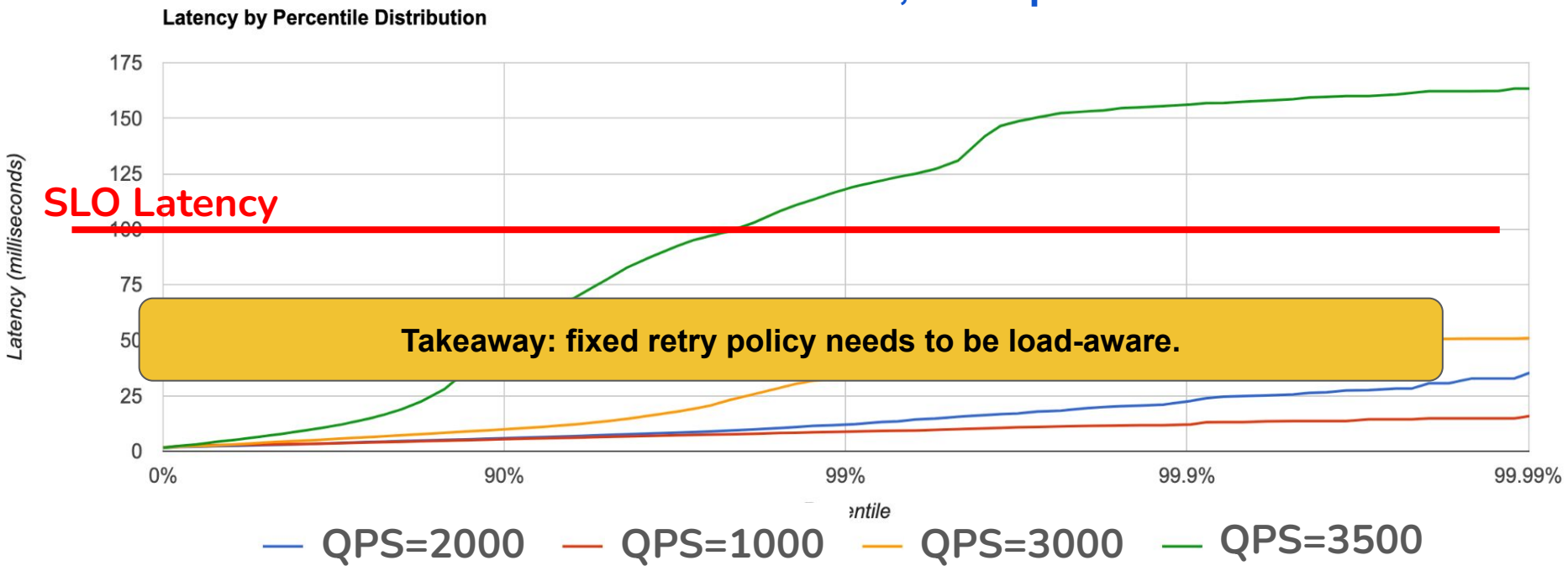
2) Retry mechanisms, meant to improve fault tolerance, inadvertently sustain metastable failures—failures that persist even when a trigger is removed.

3) We study the relationship between retry policies and performance of a microservice application operating under duress.

Results: Limitations of Exponential Retry

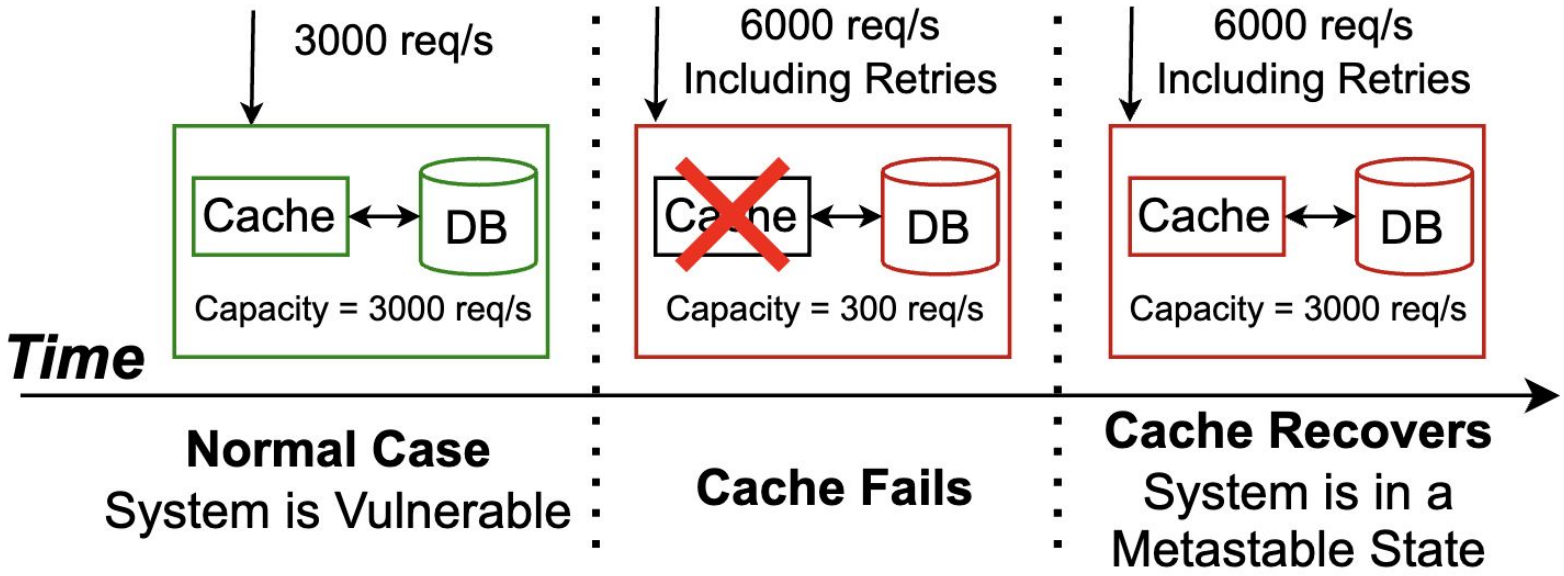
Exp. Retries ()

Tail less amplified - higher throughput (3500)
For 3000 RPS, all requests are under SLO lat



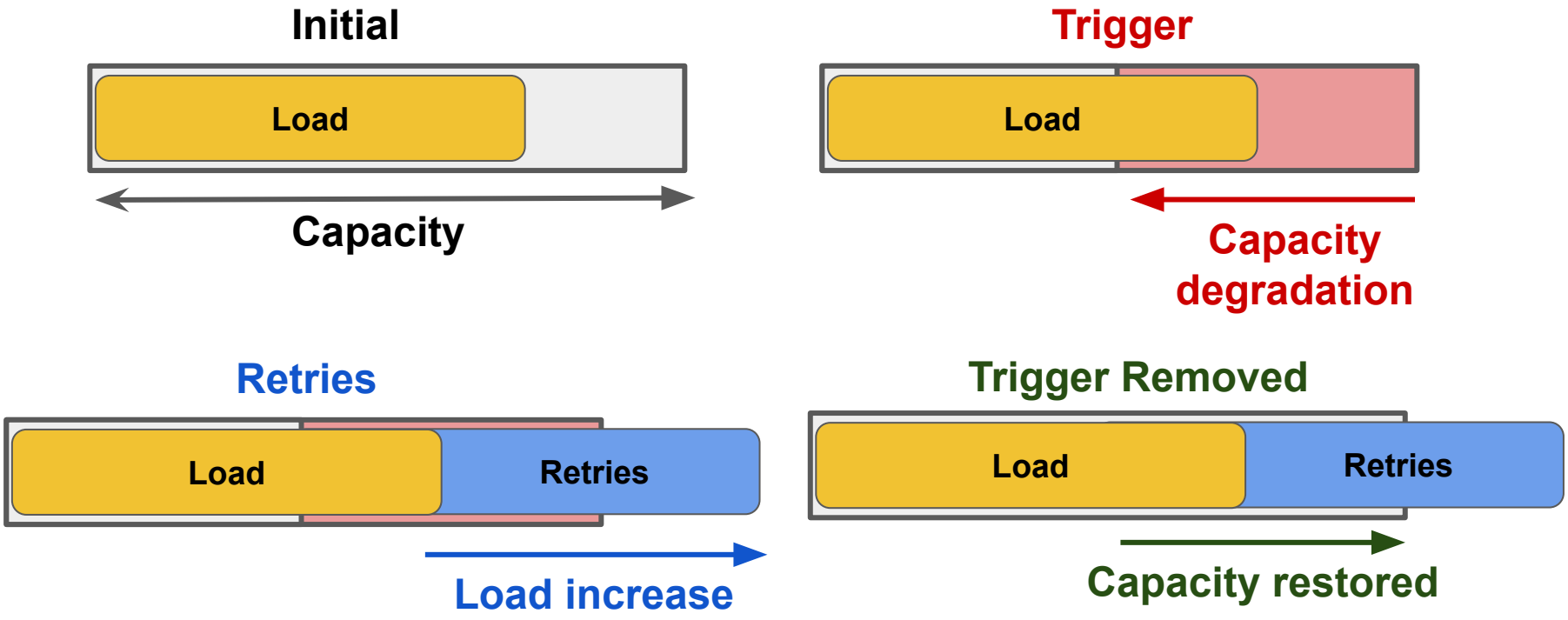
Takeaway: fixed retry policy needs to be load-aware.

Example of Metastable Failure



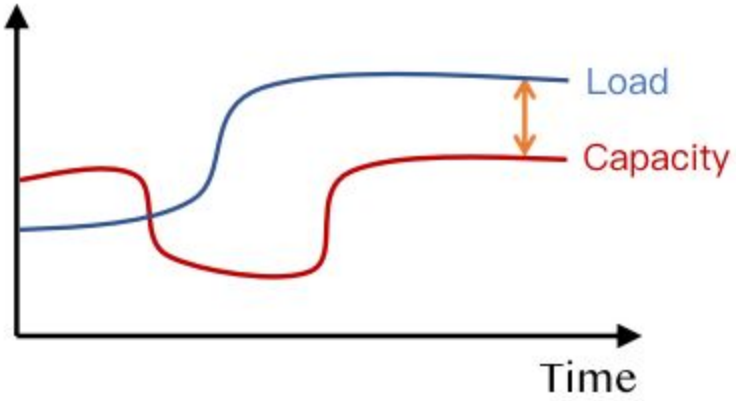
Retry needs to be system-context aware

How do Retries Cause Metastable Failures?



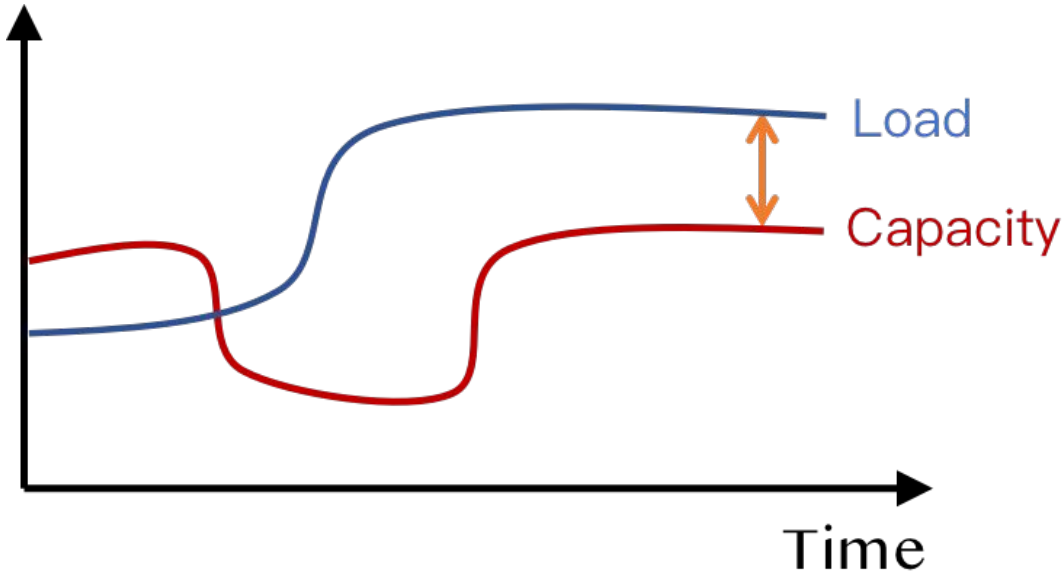
Metastable Failures in Serverless

- **Case #2: Load Increase (Retries) Leading to Sustaining SLO Violations**
  1. Given a function running at the normal load
  2. When there's resource contention leading to capacity degradation, due to cold starts or threshold cap, SLO violations happen
  3. The common strategy for function end-users trigger retries which in turn result in increased load and more cold-started containers
  4. Even after the resource contention is gone, SLO violations still exist



However, resource contention can be transient so instead of creating new containers (cold starts), a better move is to do load shedding.

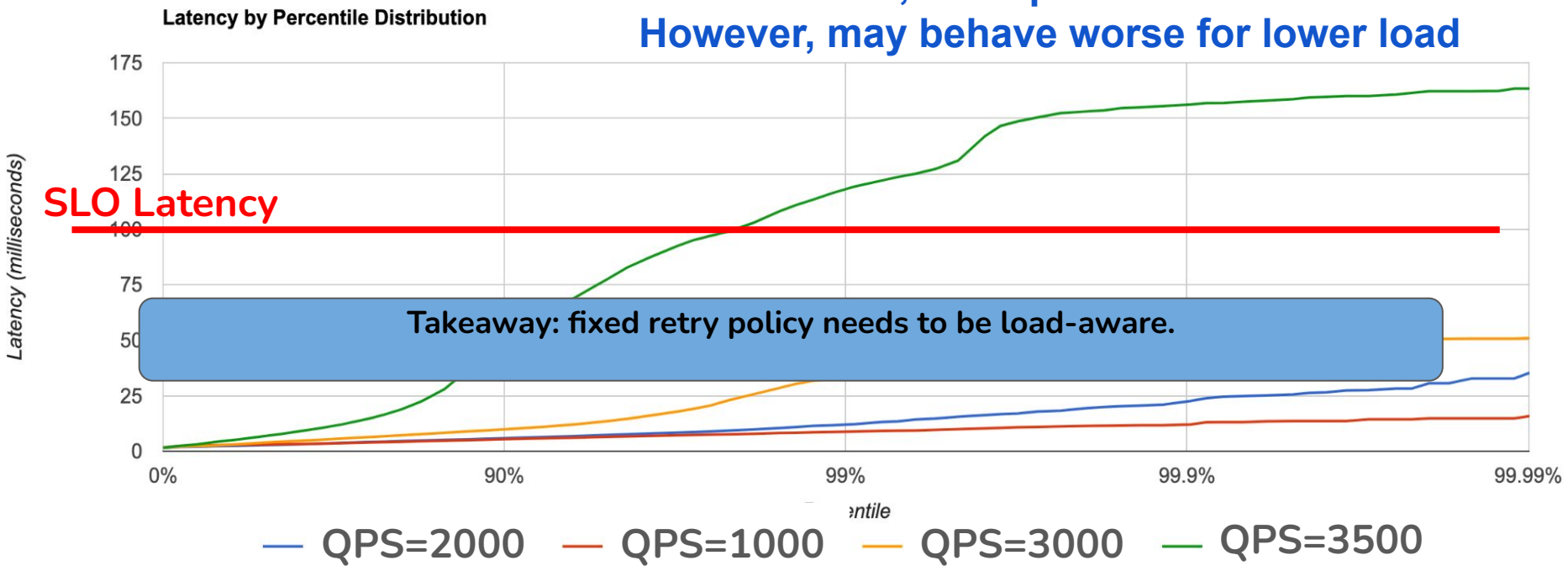
Sol: A better controller that can differentiate transient or sustaining contention to avoid metastable failures.


Load
Capacity
Time

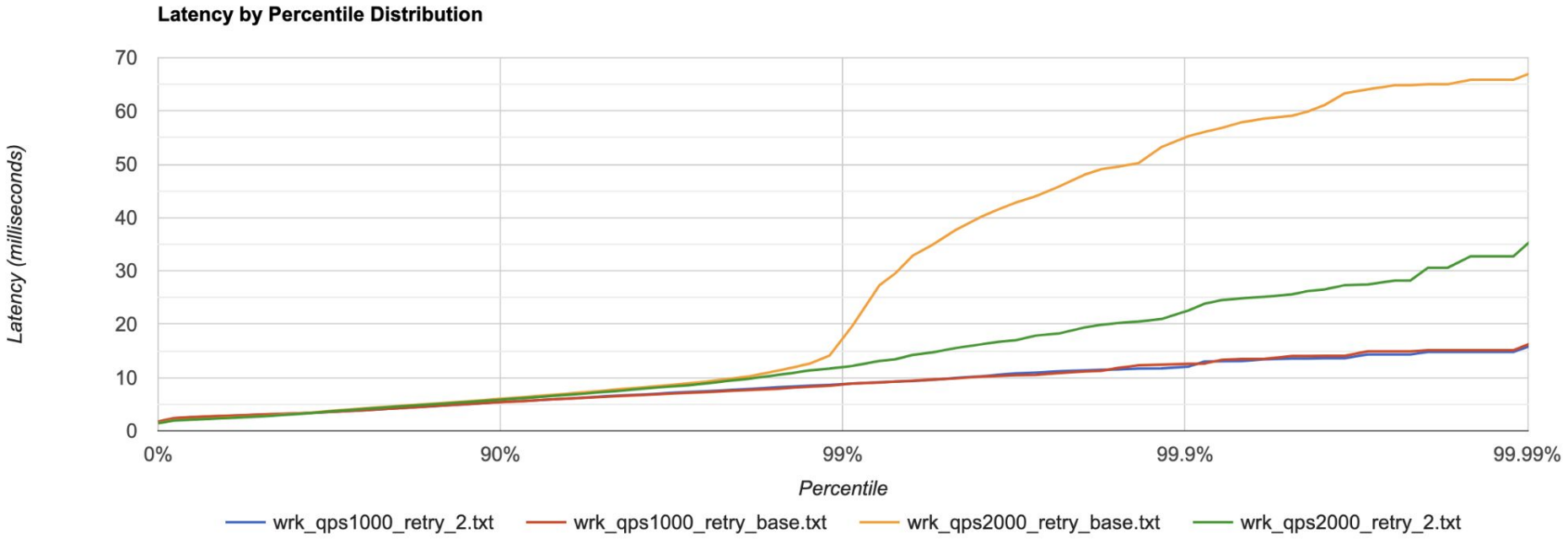
Results: Limitations of Fixed Retry

Max Retries = 2

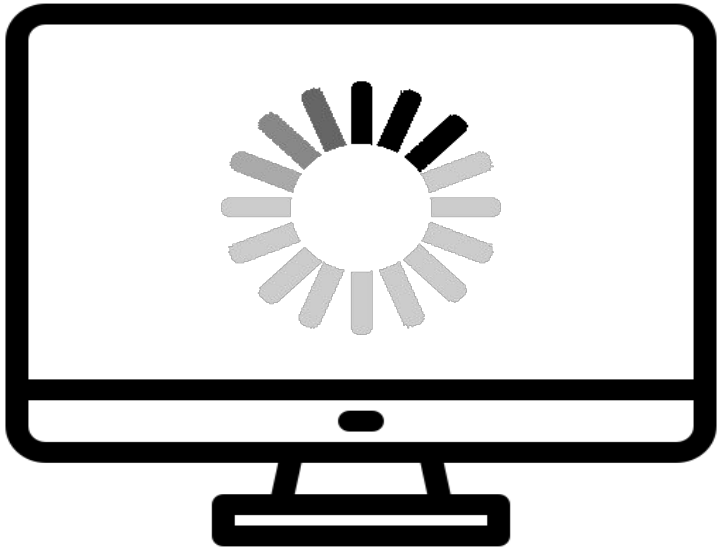
Tail less amplified - higher throughput (3500)
For 3000 RPS, all requests are under SLO lat
However, may behave worse for lower load



Limitations of Retry



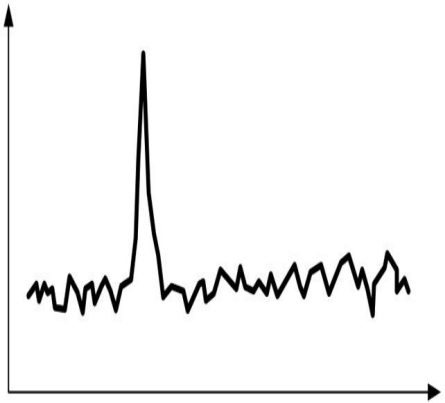
A Familiar Experience



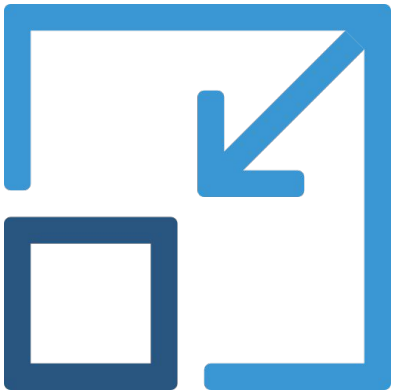
What causes metastable failures?

Vulnerable State

Triggers



Load Spike



Capacity Decrease