Scaling to 1 Million RPS and Beyond with Kubernetes, Istio and gRPC

HighLoad++

Профессиональная конференция
для разработчиков высоконагруженных
систем

Robin Percy

opsguru

- Founding Partner in OpsGuru
- Partner Consultant at Google Cloud
- Kubernetes Contributor

What will be covered?

Istio	gRPC Streams	Observability
Common Bottlenecks and their Solutions	Load Balancing Flow Control	Monitoring with Prometheus Setting up Grafana and Tracing

Istio

- Common Bottlenecks
- Recommended Tunings

Why Istio?

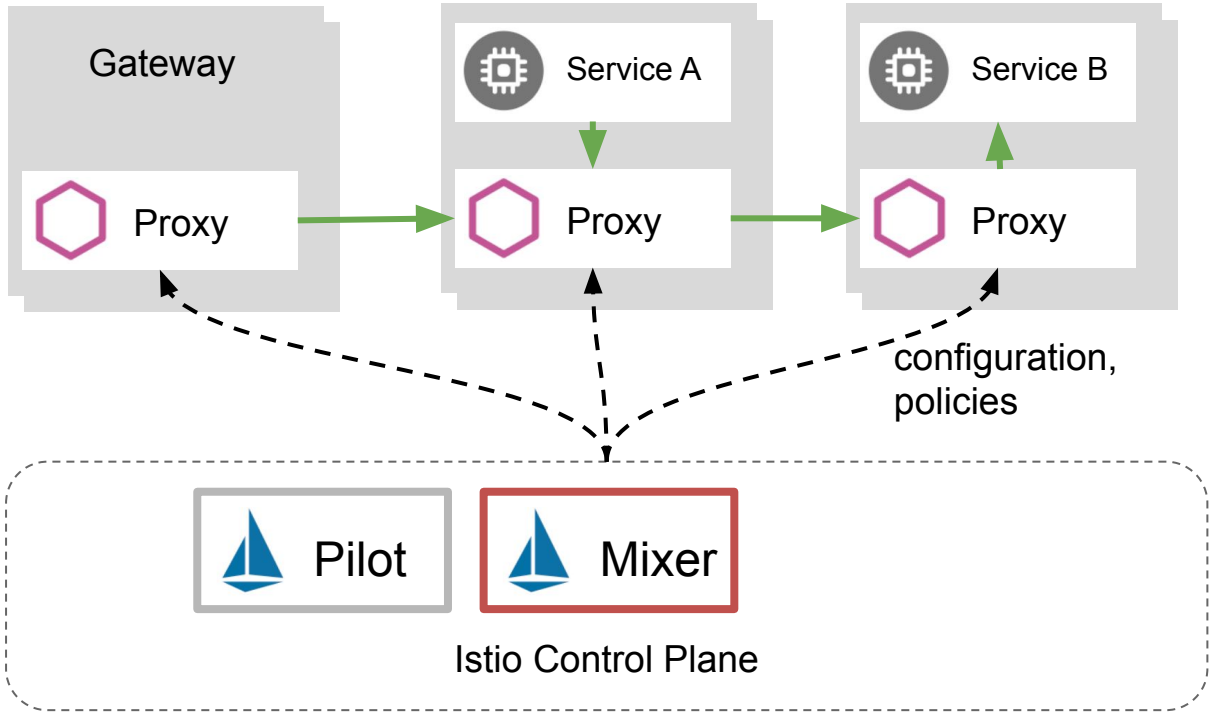
Microservices	traffic shifting, routing, separation of concerns
Security & Compliance	mTLS, JWT, RBAC, Identity Management
Observability	proxy stats, tracing

Istio Goal: Linear Scalability

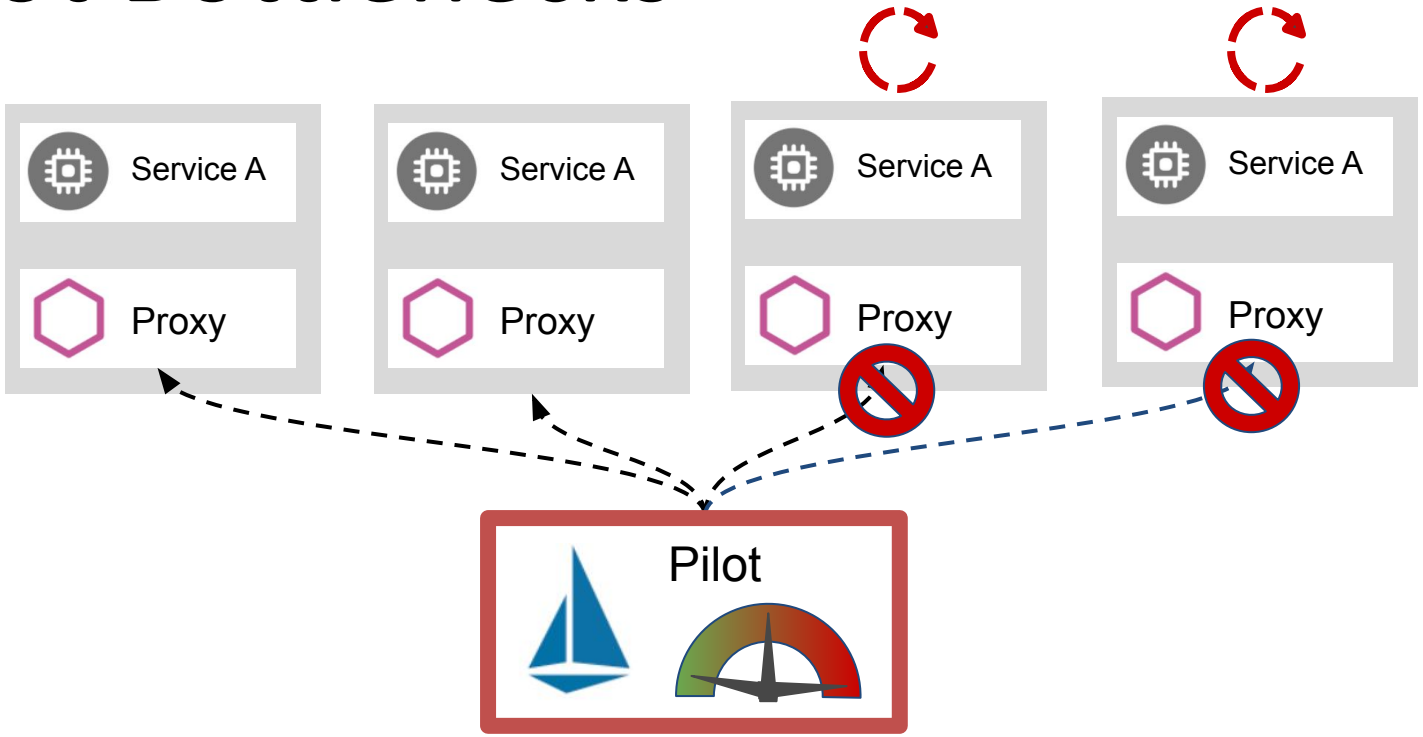
- 1K RPS per client pod
- Expected:
  - 200 clients == 200K RPS
- Actual:
  - 50 clients == 50K RPS
  - 200 clients == ~50K RPS + Errors

Common Bottlenecks

1. Pilot
2. **Mixer**
3. HPA

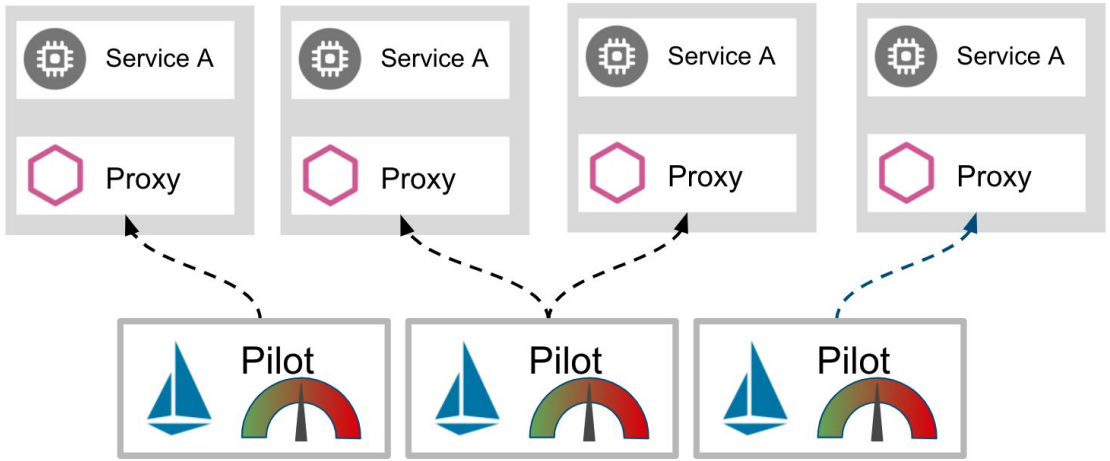


Pilot Bottlenecks



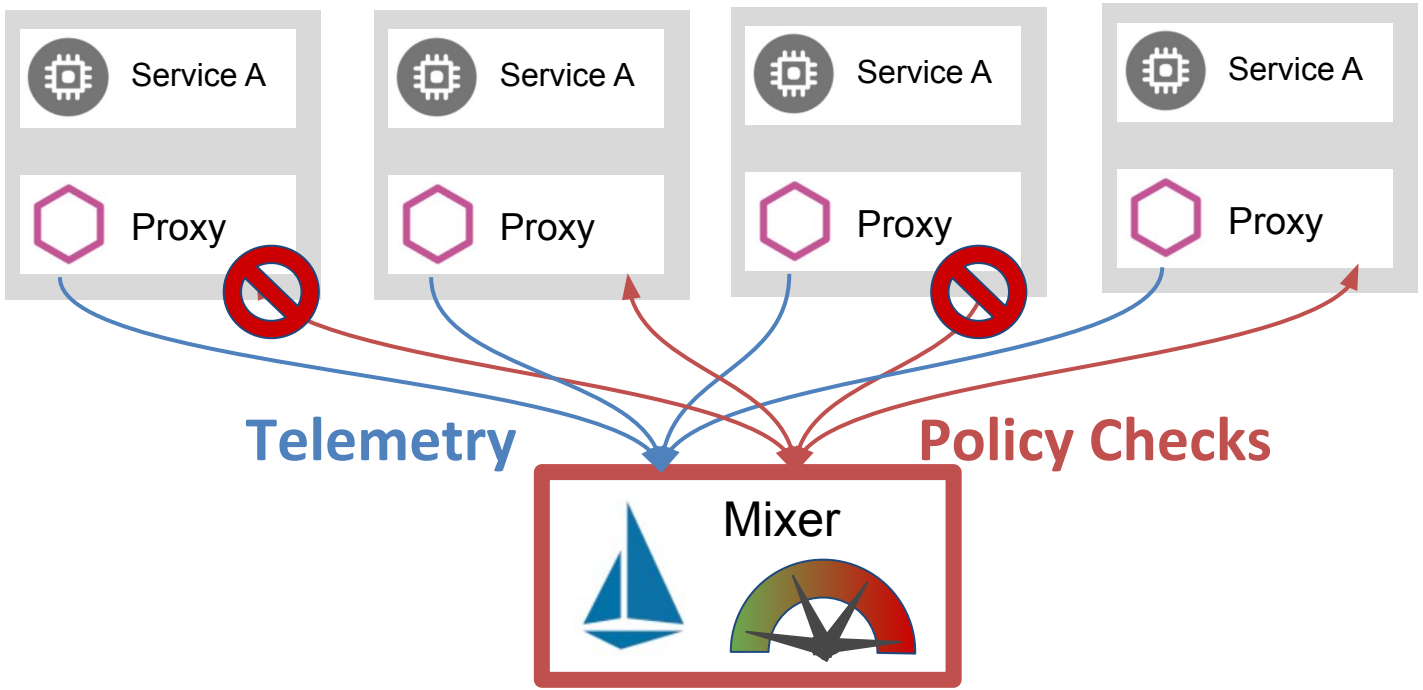
Pilot Solution

- HPA min = 3
- Pre-warm
- 1:50 ratio



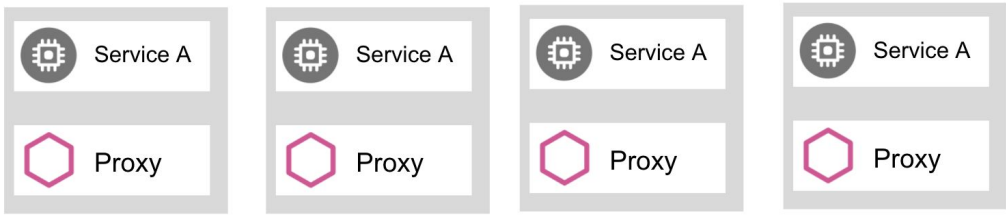
Gain: 50k -> 80k RPS

Mixer Bottlenecks

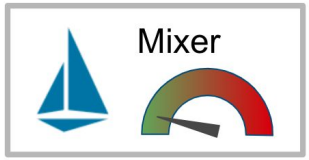


Mixer Solutions

- Inherently a SPOF
- Disable Mixer




```
mixer:
  policy.enabled=false
  telemetry.enabled=false
```



Gain: 80k -> 200k+ RPS

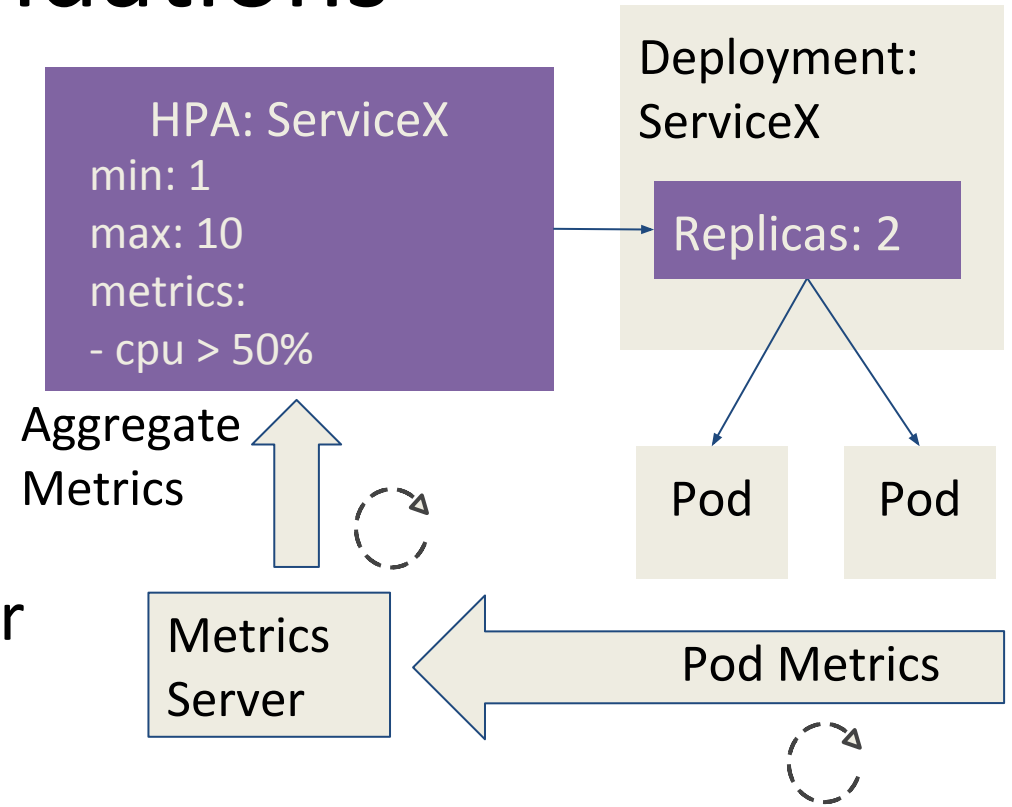
Mixer Workarounds

- Policies
  - eg. Rate limiting, whitelists, blacklists, header rewrites and redirects
  - Use RBAC and Traffic Management

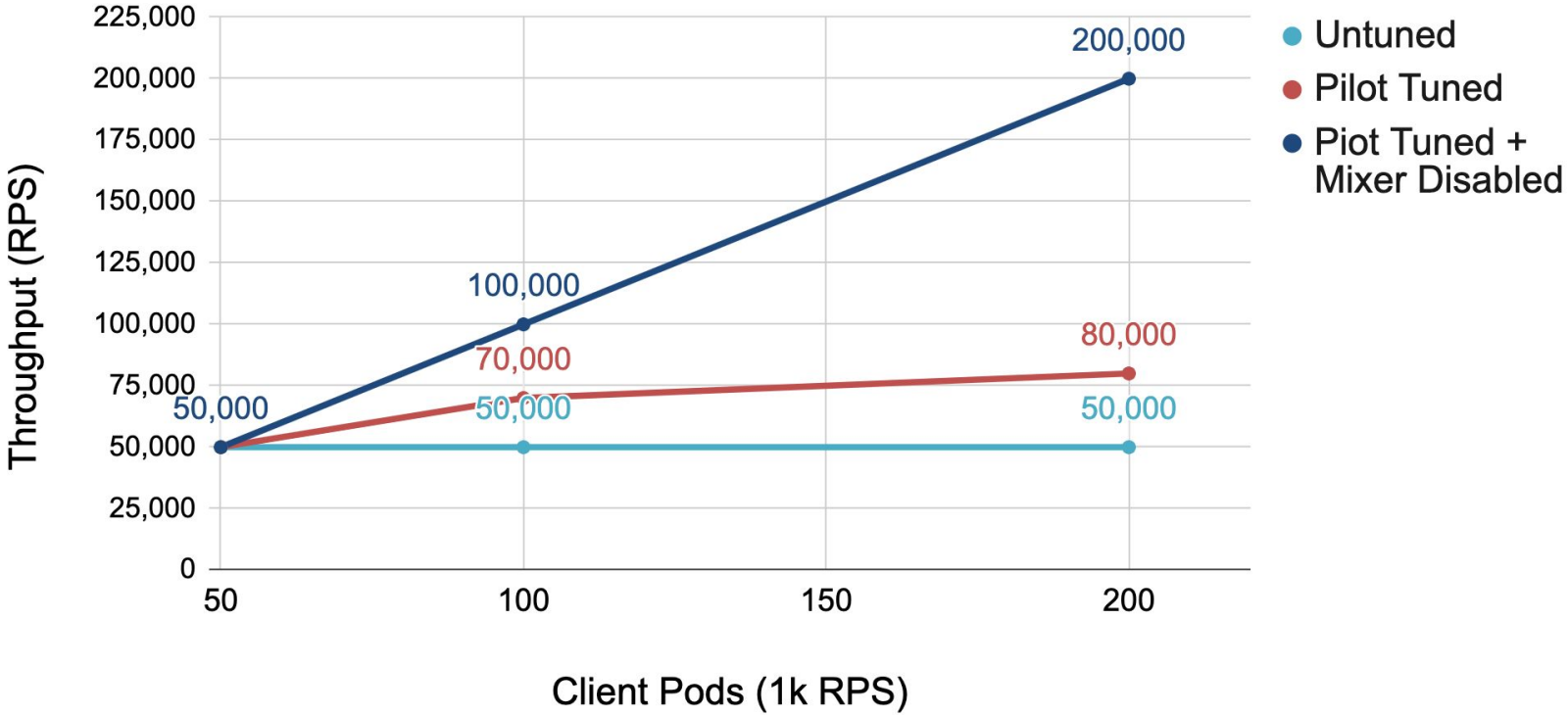
- Alternate Telemetry:
  - Envoy Native Telemetry
  - Mixerless HTTP Telemetry (Experimental)

HPA Recommendations

- Drop CPU to 50%
- Custom metrics
- Reduce interval
  - Custom adapter
  - HPA Sync Period
  - Controller Manager flags



Istio Scalability after Tunings



- Streaming Overview
- Load Balancing
- Flow Control

Why gRPC Streams?

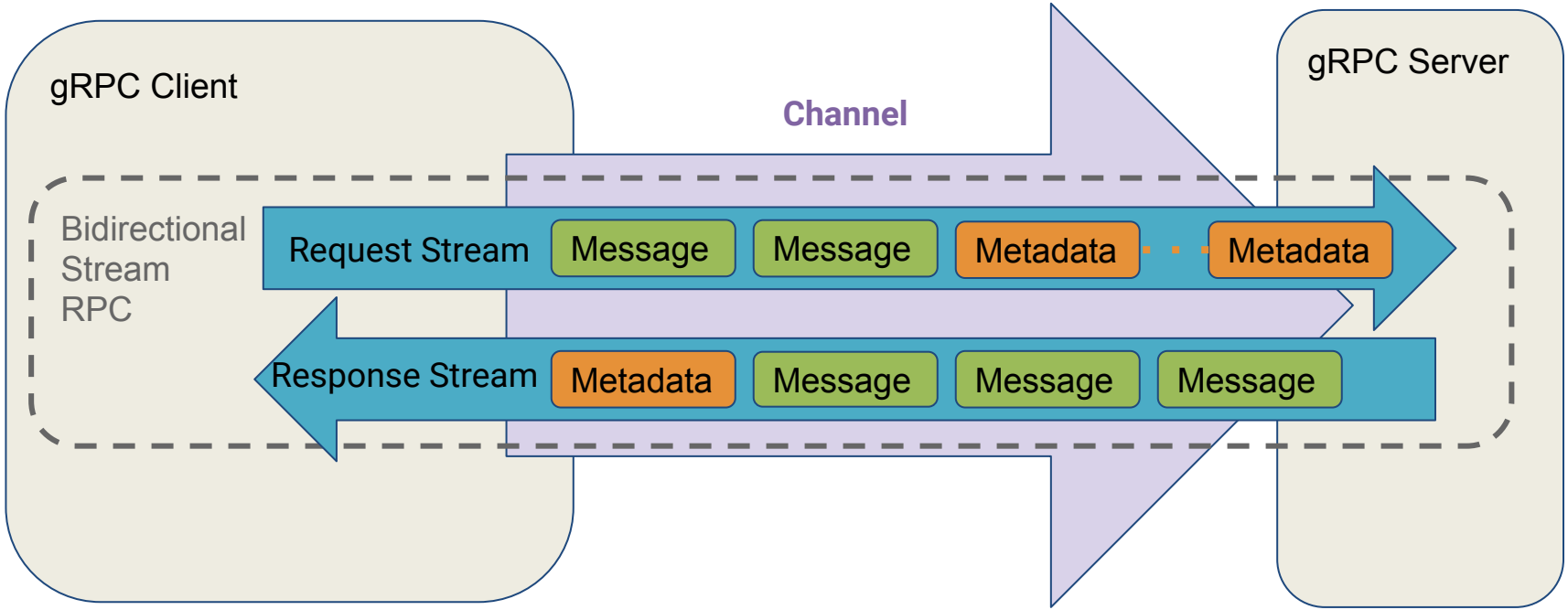
gRPC

- Protocol Buffers
- Eliminates explicit transformations
- Code generation baked into tooling

Streams

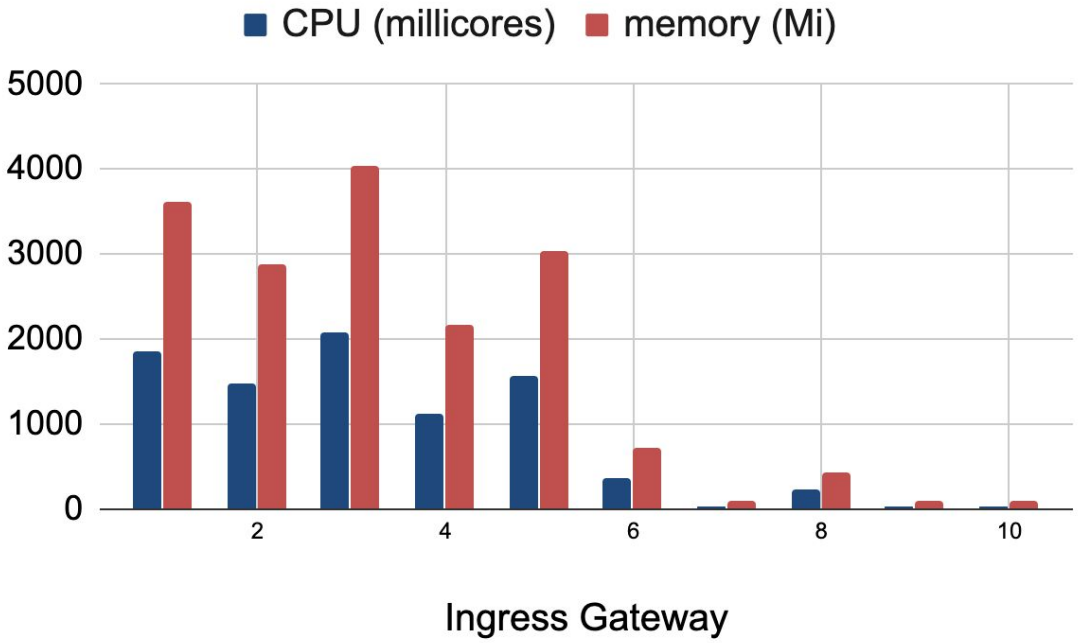
- HTTP/2 multiplexing
- Sharing of metadata
- Server-Push

Streams Overview

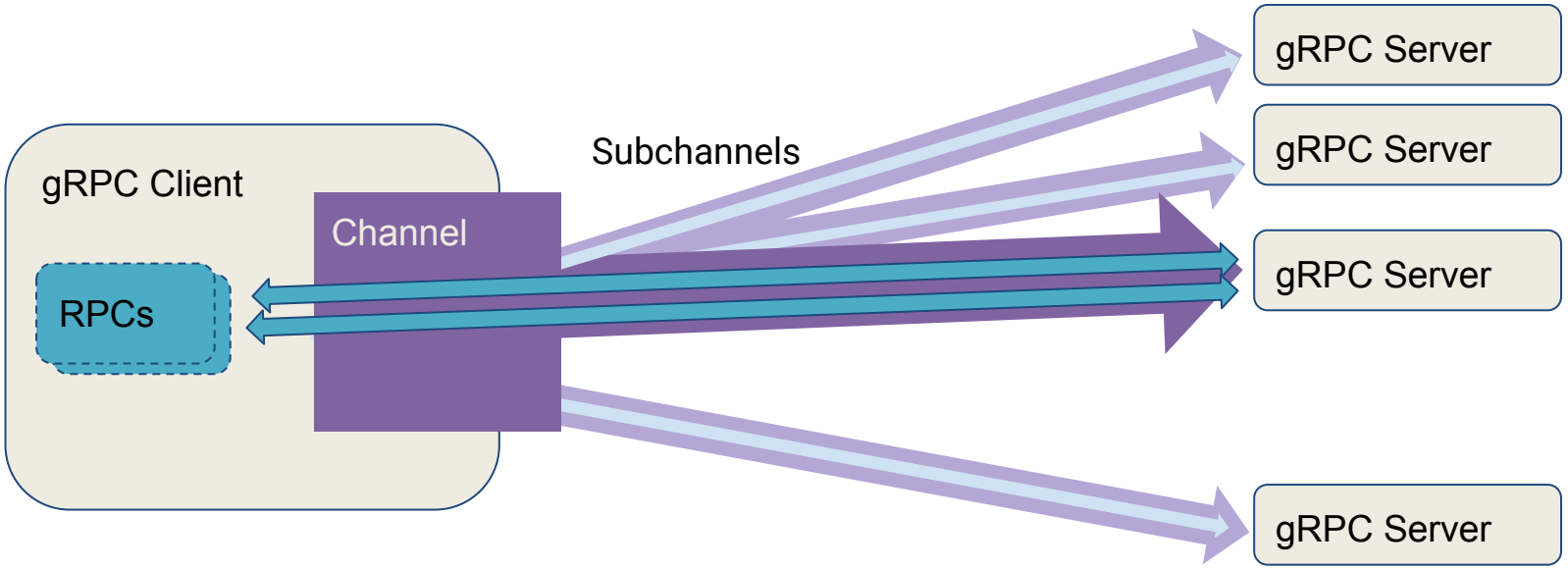


Challenge #1: Load Balancing

- 100 Clients
- 30 min after scaling

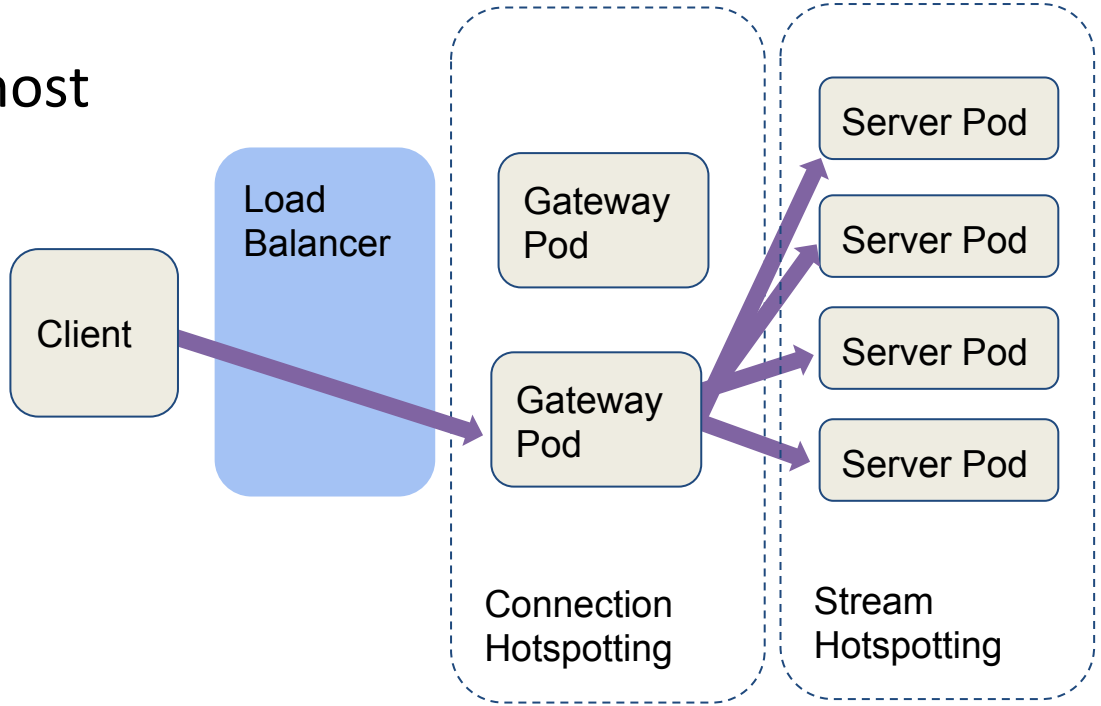


Streams: Client Load Balancing



Ingress Distribution

- Client sees a single host
- Single Gateway Pod
- Ingress vs Mesh Balancing

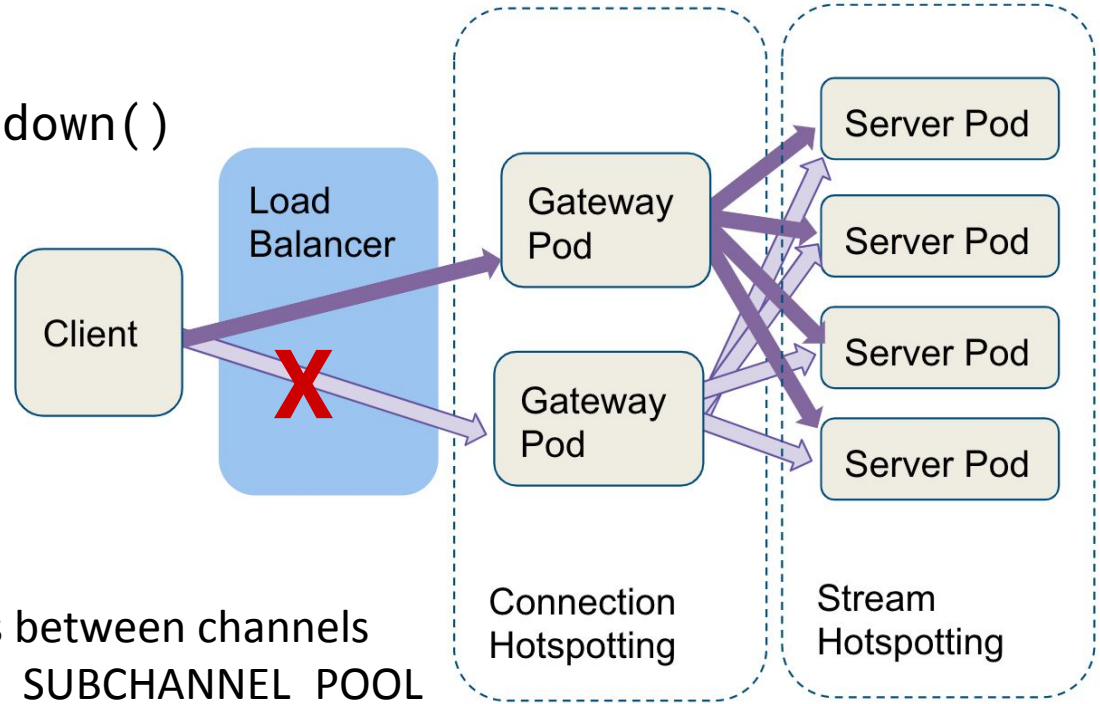


*** Assuming single IP is advertised by LB**

Solution

Java:

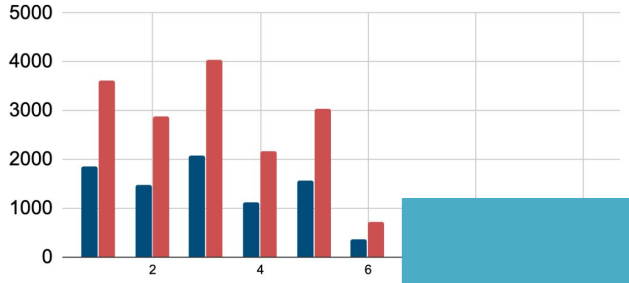
- `ManagedChannel.shutdown()`

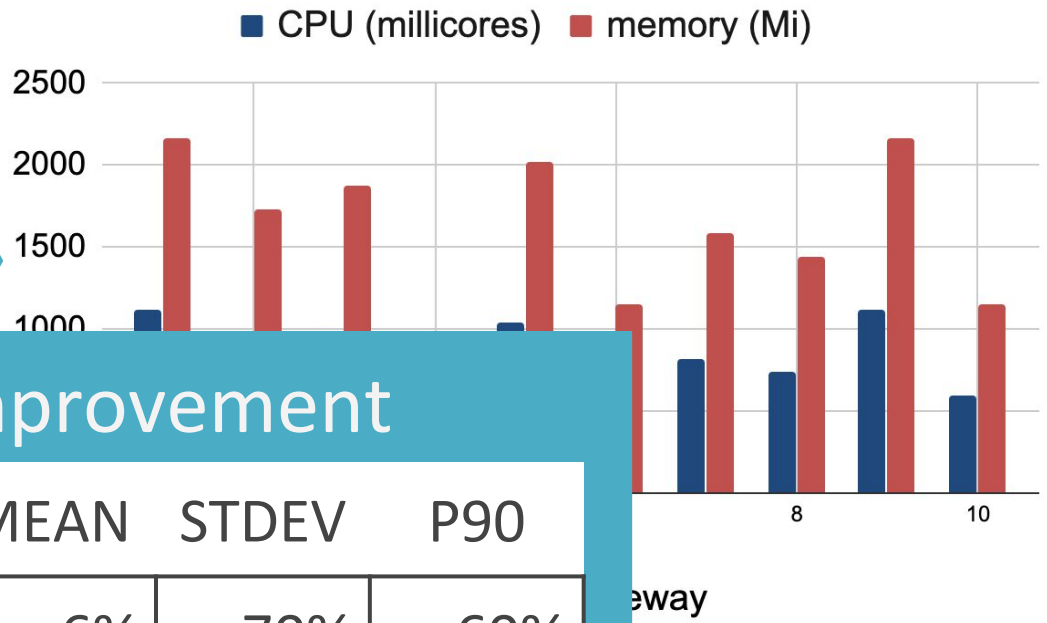


Other gRPC implementations

- May share connection pools between channels
- See GRPC_ARG_USE_LOCAL_SUBCHANNEL_POOL

Results

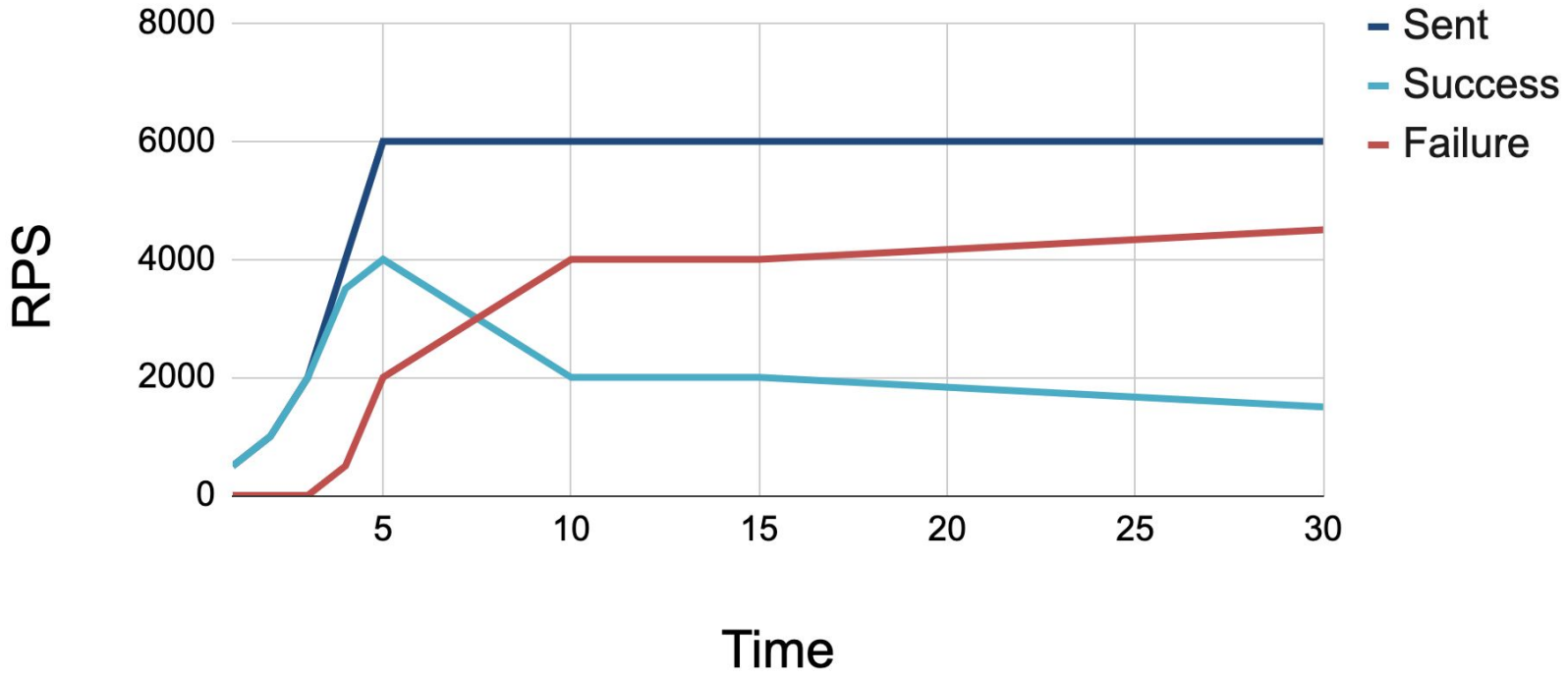




Improvement

	MEAN	STDEV	P90
CPU	6%	70%	60%
MEM	7%	65%	58%

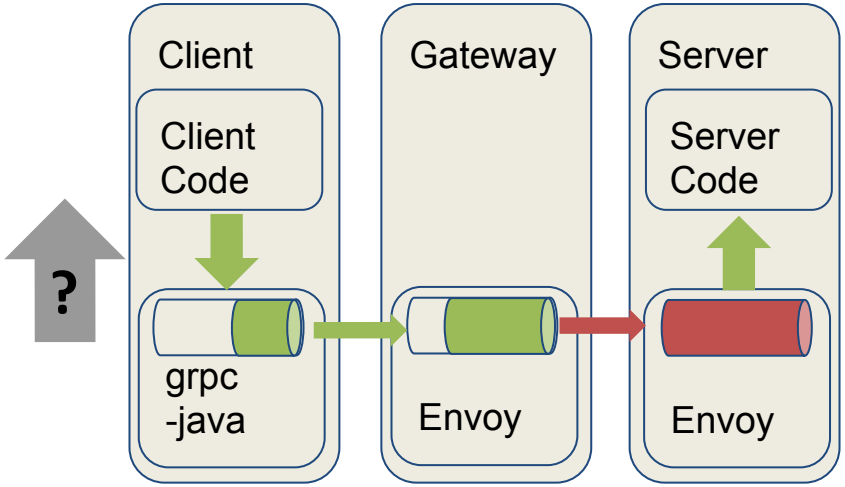
Challenge #2: Flow Control



Flow Control

- Two Primary Concerns

Backpressure Detection

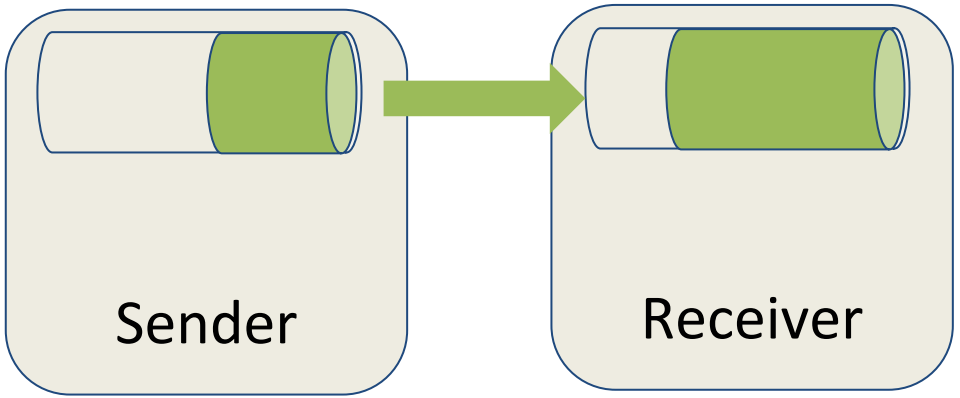


Aggregate Buffer Size

Buffer + Buffer + Buffer

Managing Envoy HTTP/2 Buffers

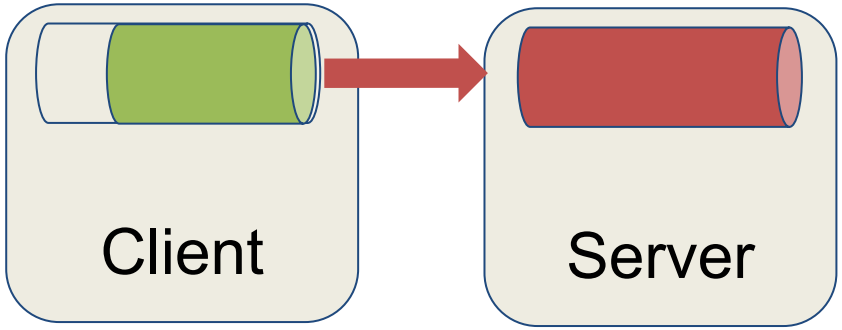
- HTTP/2 Window Size
- Start at 64 KB
- Pilot ENV vars




```
PILOT_INITIAL_CONNECTION_WINDOW_SIZE=65535
PILOT_INITIAL_STREAM_WINDOW_SIZE=65535
```

Detecting Backpressure (grpc-java)

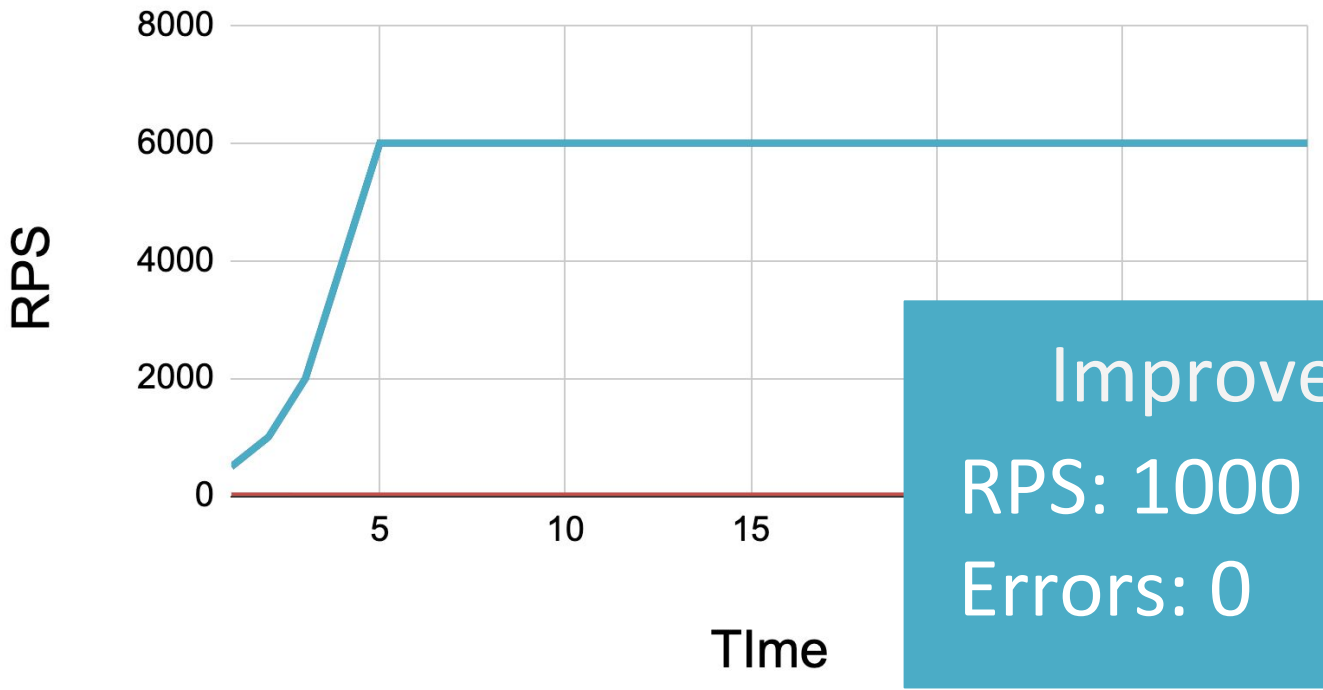
- Client must explicitly check
- Use ClientCall.isReady()



Example (grpc-java)

```
call = channel.newCall(bidiStreamingMethod, callOptions);
listener = new ClientCall.Listener<FooResponse>() {
  public void onReady() {
    while (call.isReady()) {
      call.sendMessage(nextRequest);
      ...
    }
  }
}
```

Results



Improvement

RPS: 1000 -> 6000

Errors: 0

Sample Cluster Stats

Pilot Pods (Steady/Burst)	3/5
Gateway Pods	40
Server Pods	125
Total Clients	170
Throughput per Client	6000 RPS
Total Throughput	~ 1 Million RPS

Observability

- Monitoring with Prometheus
- Custom Metrics
- Grafana and Tracing

Monitoring With Prometheus

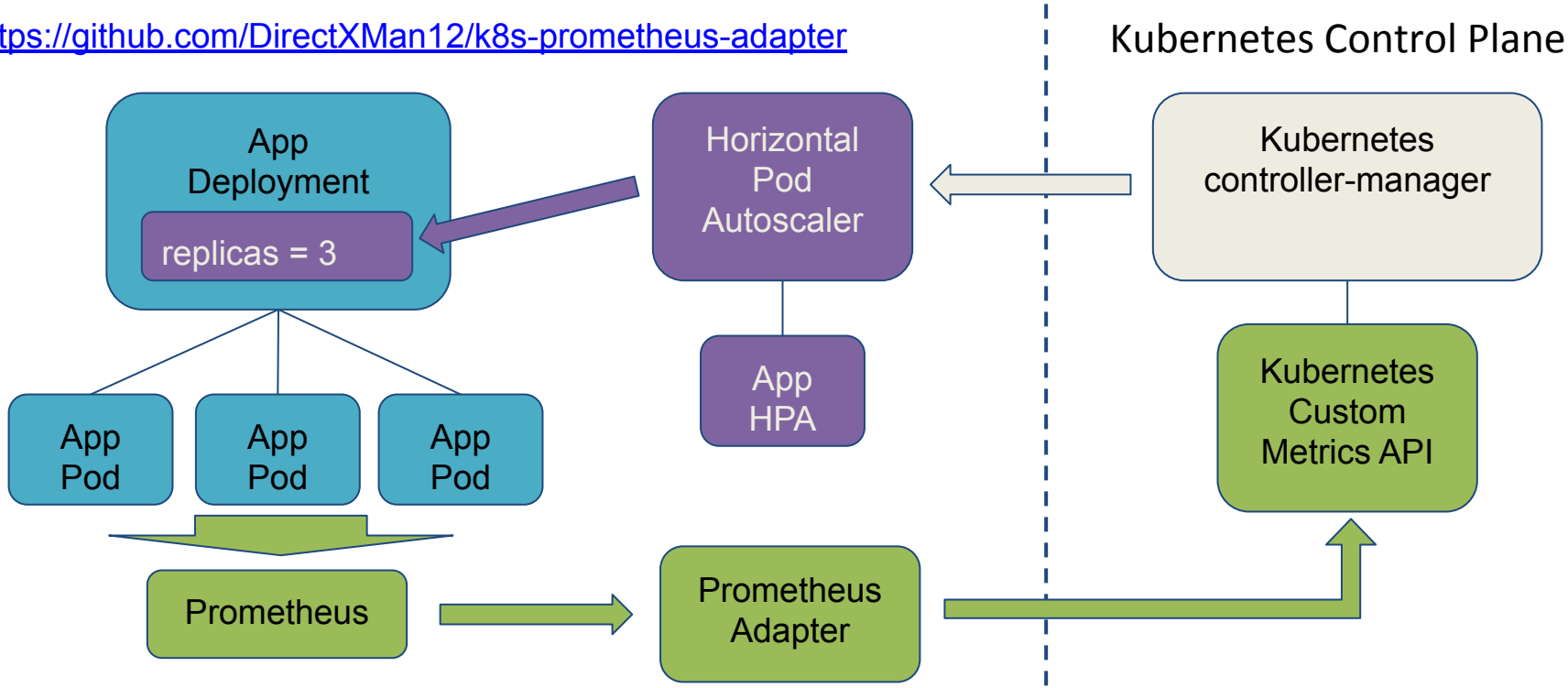
- Installing Prometheus
  - The default is not sufficient
- Enabling custom metrics
- Scaling Prometheus

Replacing Istio's Default Prometheus

- Default Prometheus is not production-grade
  - `prometheus.enabled = false`
- Option 1: Use the Prometheus Operator
  - Production-grade
  - Helpful abstractions
  - https://github.com/istio/installer/tree/master/istio-telemetry/prometheus-operator
- Option 2: Manage Prometheus manually
  - More control, but also more error-prone

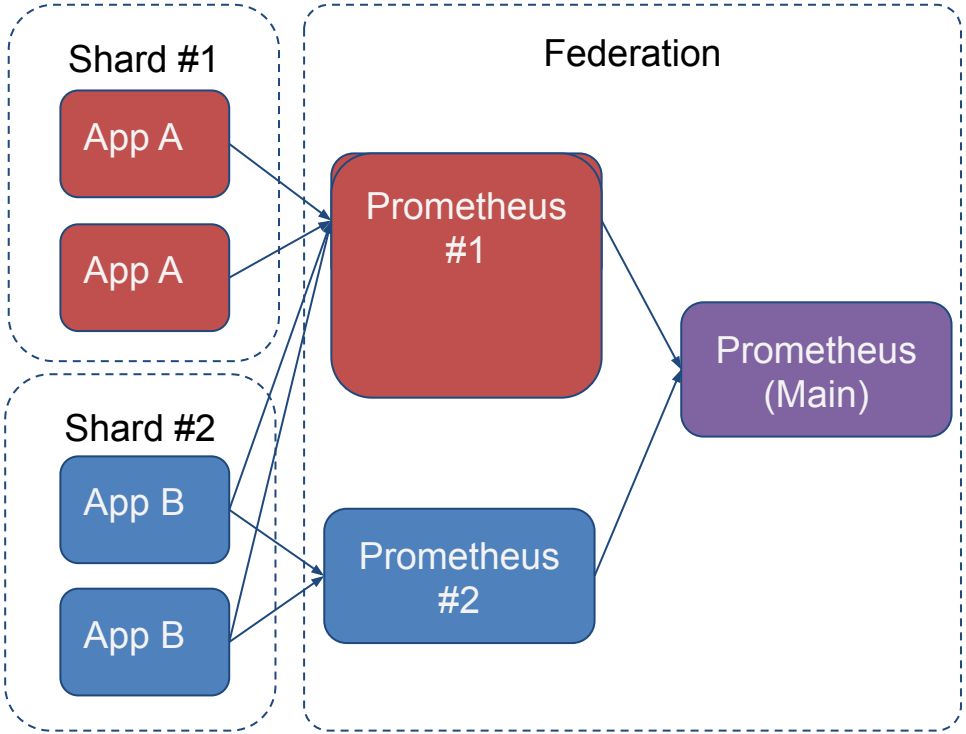
Using the Custom Metrics Adapter

https://github.com/DirectXMan12/k8s-prometheus-adapter



Scaling Prometheus

- Vertically
- Sharding
- Federation



Grafana and Tracing Components

Jaeger/Zipkin:

- Use respective operator to install
- `tracing.enabled = false`
- `global.tracer.zipkin.address = tracer_address`
- Test your `pilot.traceSampling` rate

Grafana:

- `grafana.enabled = false`
- Use grafana installed by Prometheus Operator
- Copy Dashboards from default addon:
  - https://github.com/istio/istio/tree/master/install/kubernetes/helm/istio/charts/grafana/dashboards

Summary

Istio	- Customize HPA - Beware of thundering herd and Pilot - Disable mixer
gRPC	- Force regular disconnects - Configure smaller buffers - Test for back pressure in client code
Observability	- Replace default observability components - Enable custom metrics collection - Scale via Sharding and Federation, if necessary

Thank You!

Robin Percy
Founding Partner
robin@opsguru.ru