Teads

Moving from Pull to Push

and dividing by 1000 the network traffic

Sunny Tech 2023

Agenda

Context

Architecture

Implementation

Feedback

Context

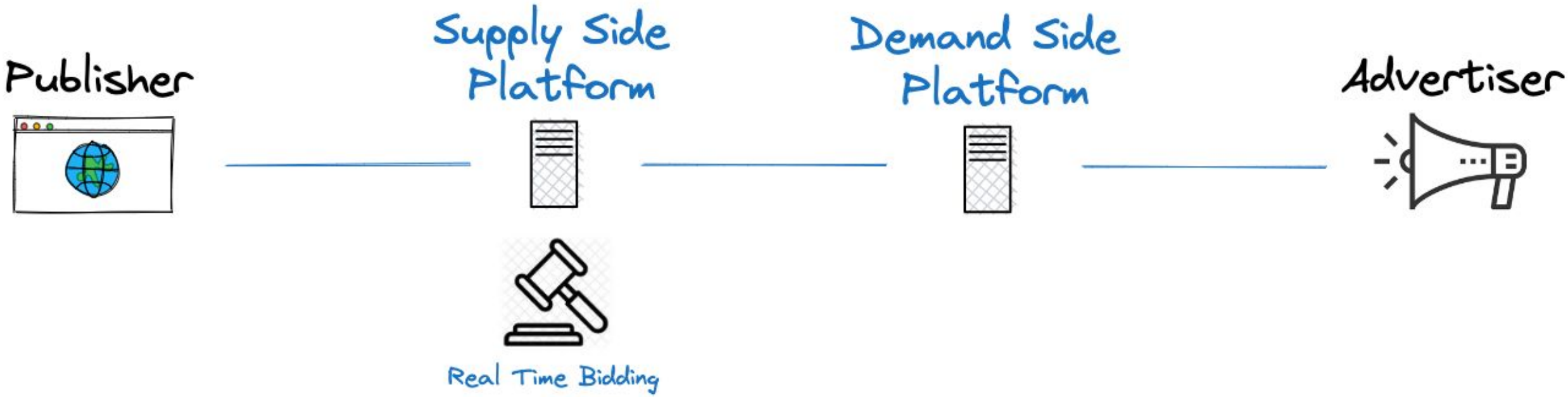
Business

Advertising

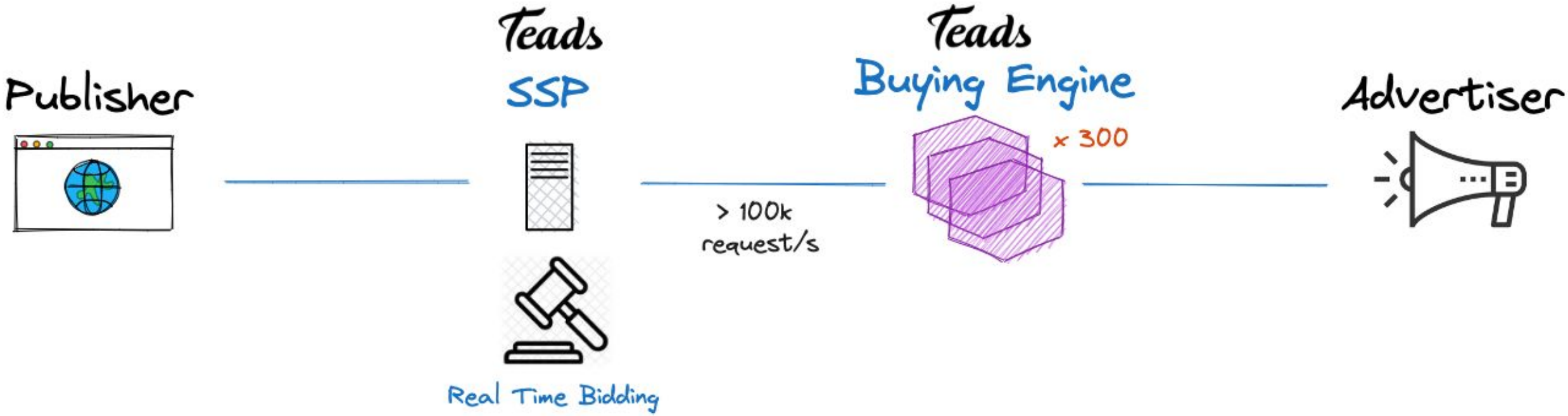
Digital Advertising at the beginnings

Digital Advertising in the 2000s

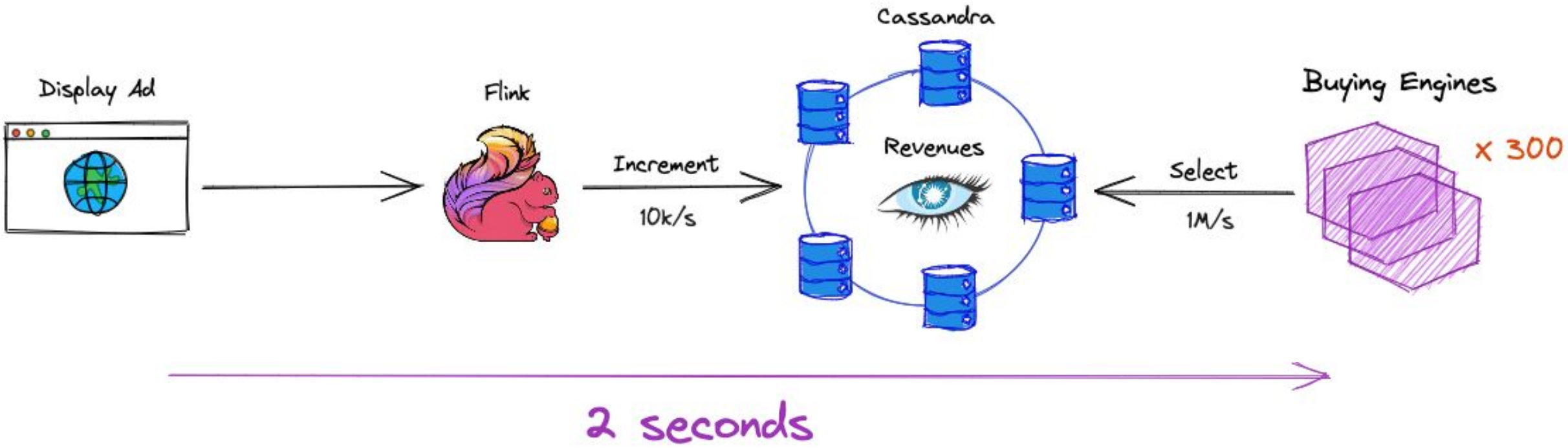
Digital Advertising in the 2010s



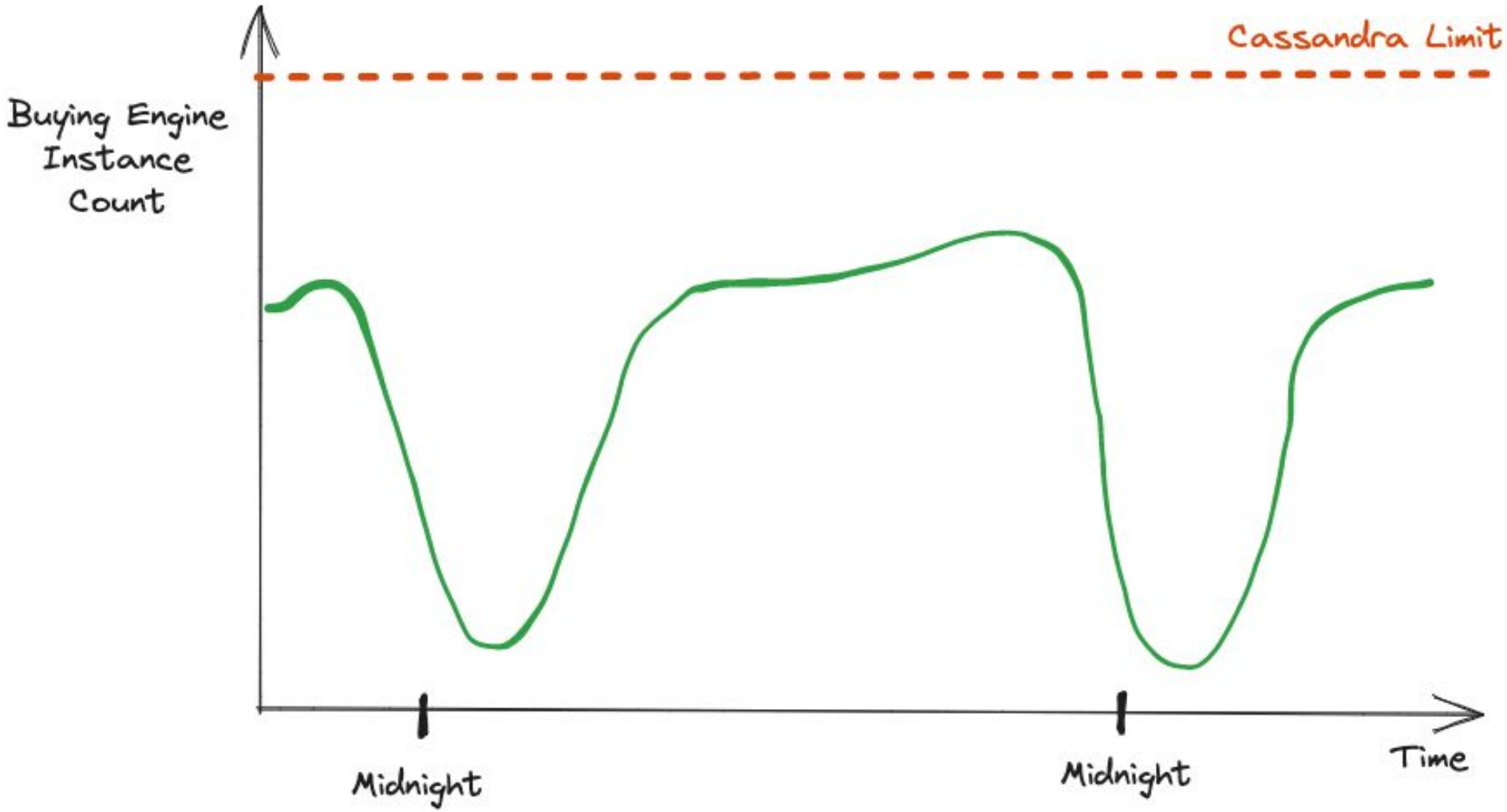
The Buying Engine



Ad Budget Control

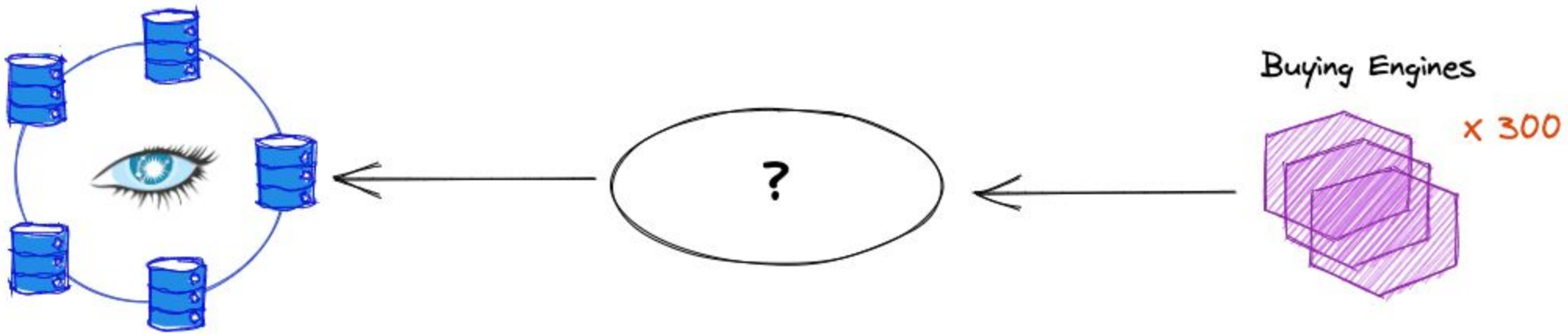


The Scaling Limitation

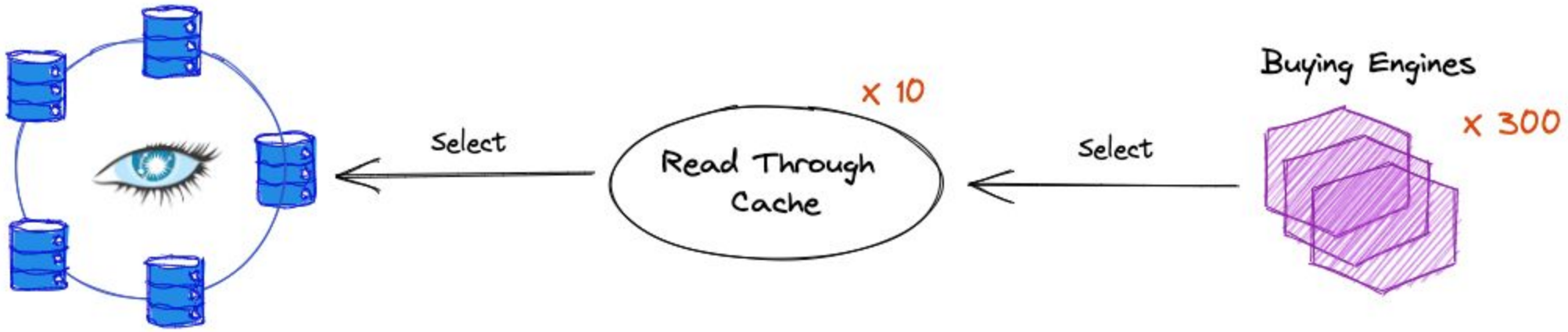


Architecture
Design

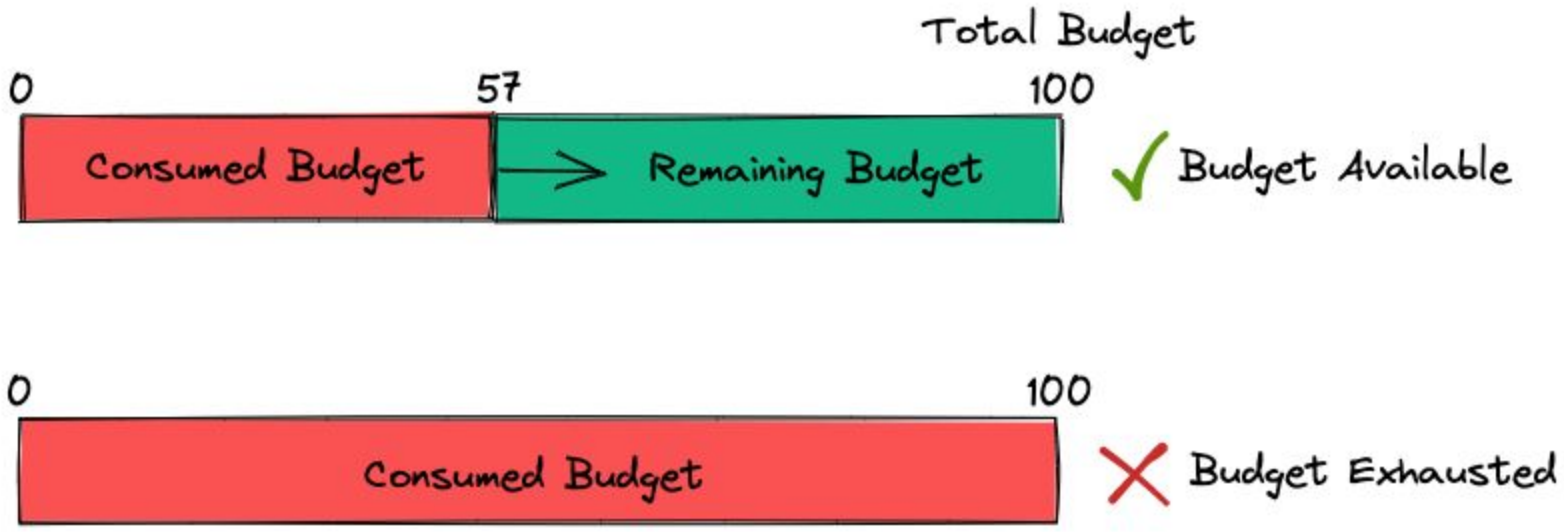
Decorrelate Cassandra & Buying Engines



A Read Through Cache ?



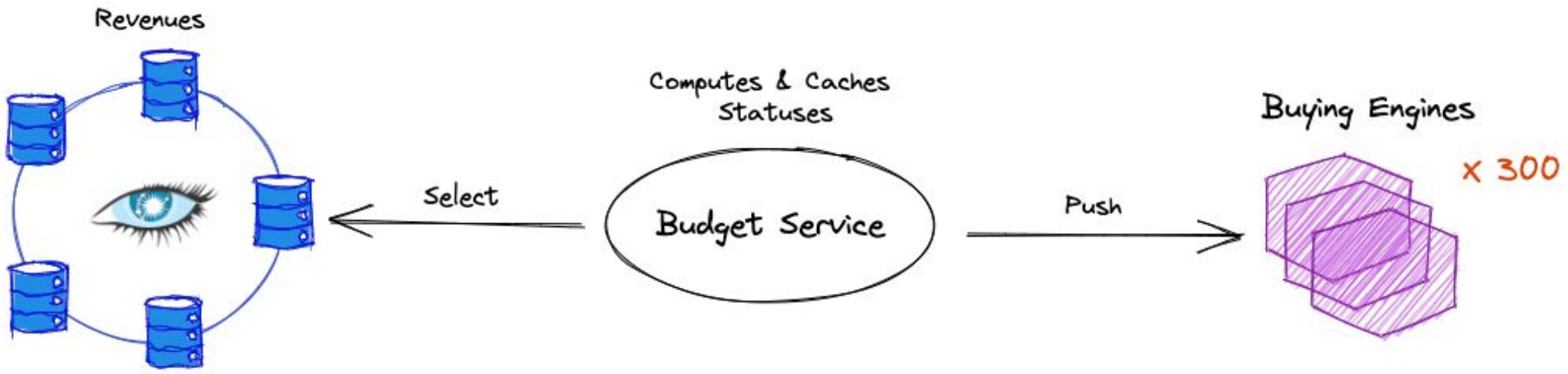
Why do we need revenue counters ?



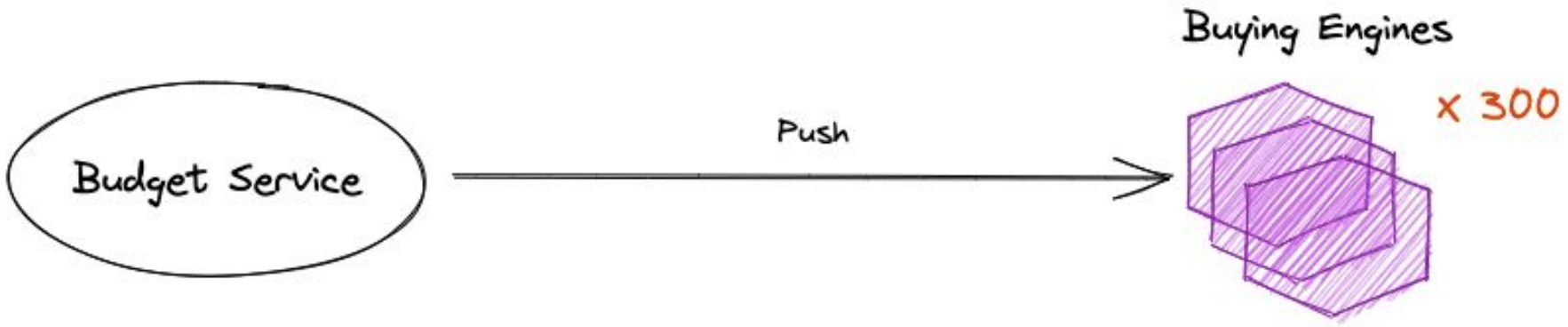
Why do we need revenue counters ?

	Data	Period	Hourly size /Ad
Revenue counters	10 doubles	2 seconds	>10000 bytes
Budget Status	1 boolean	30 minutes	~10 bytes

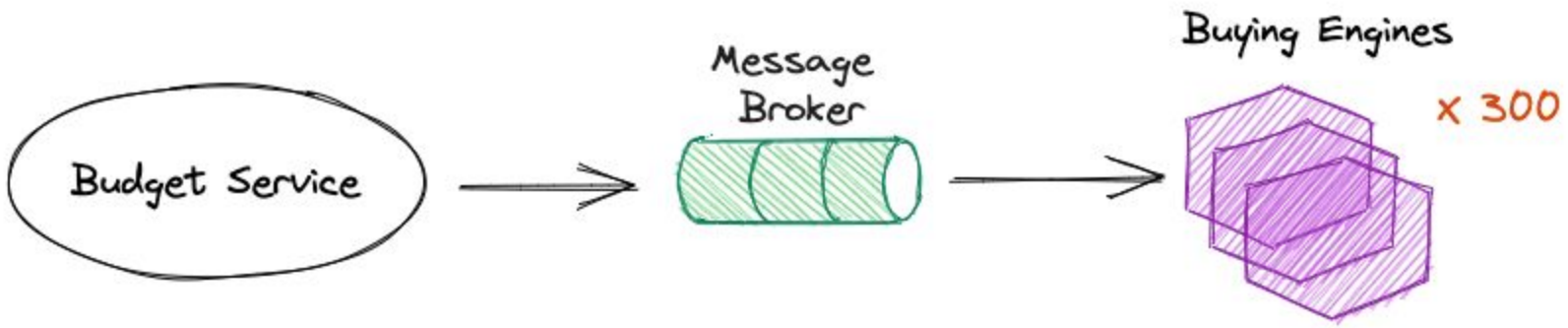
The Budget Service



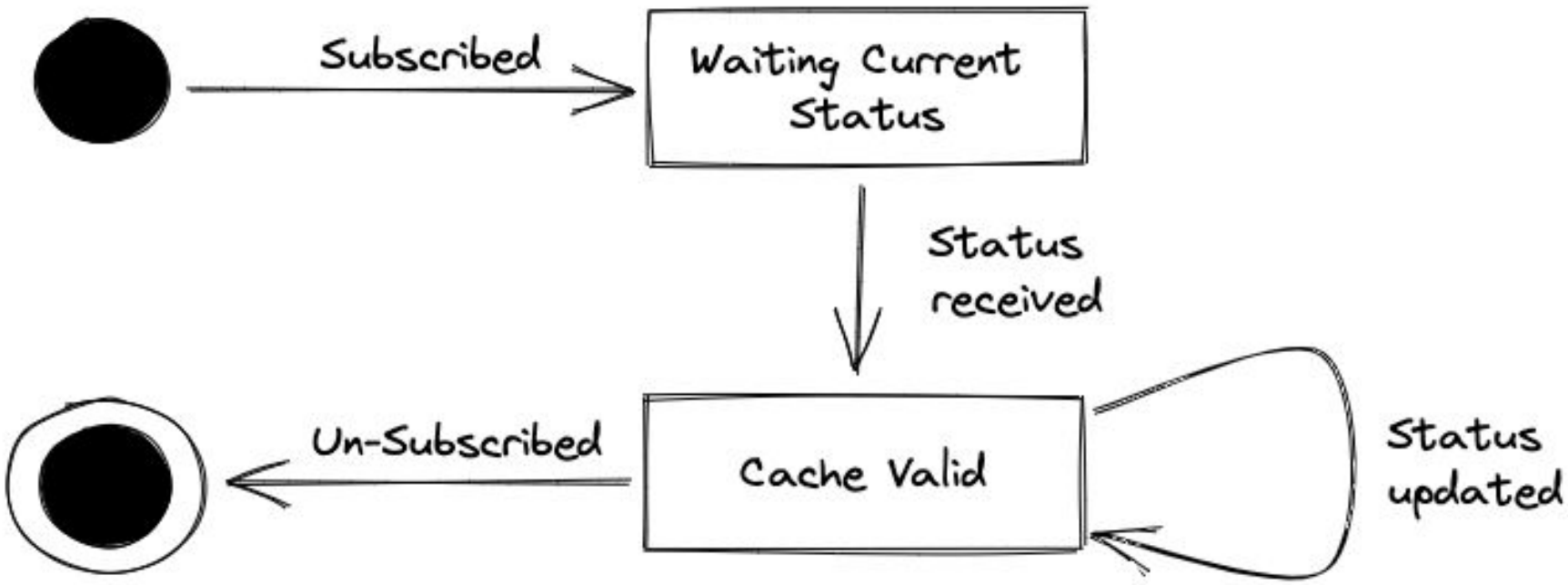
How to Push ?



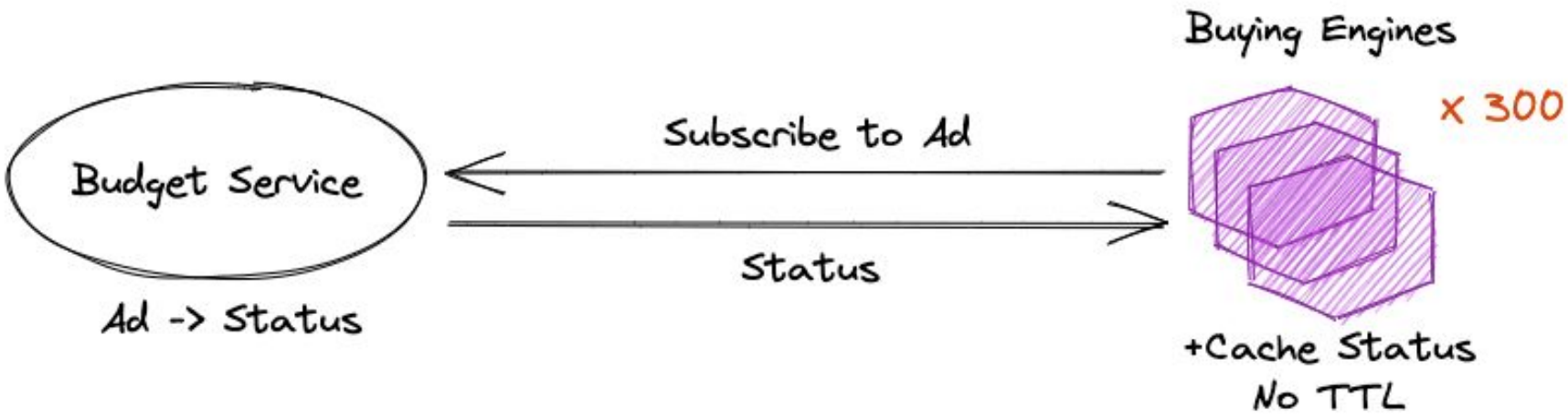
Use a Message Broker?



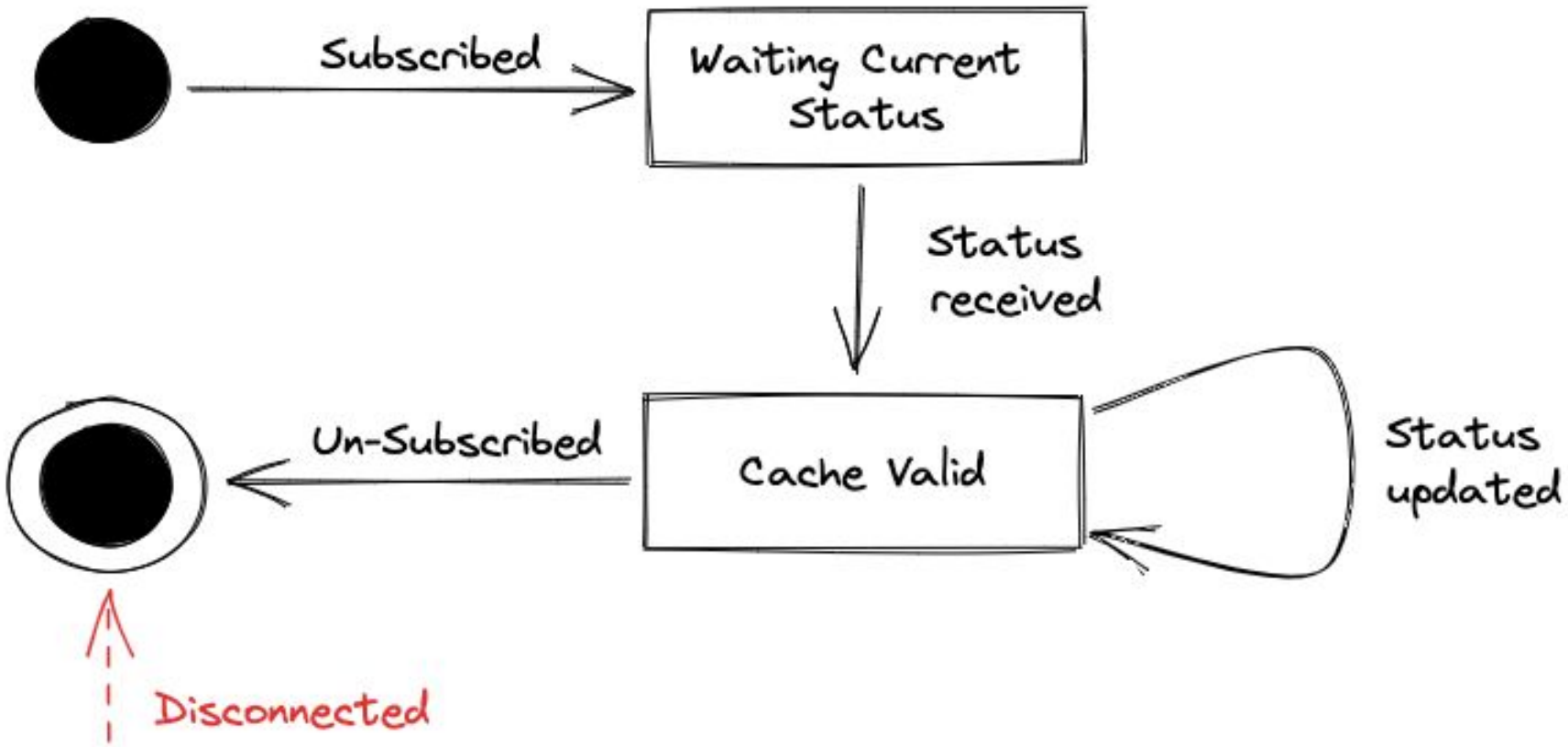
The Observer Pattern



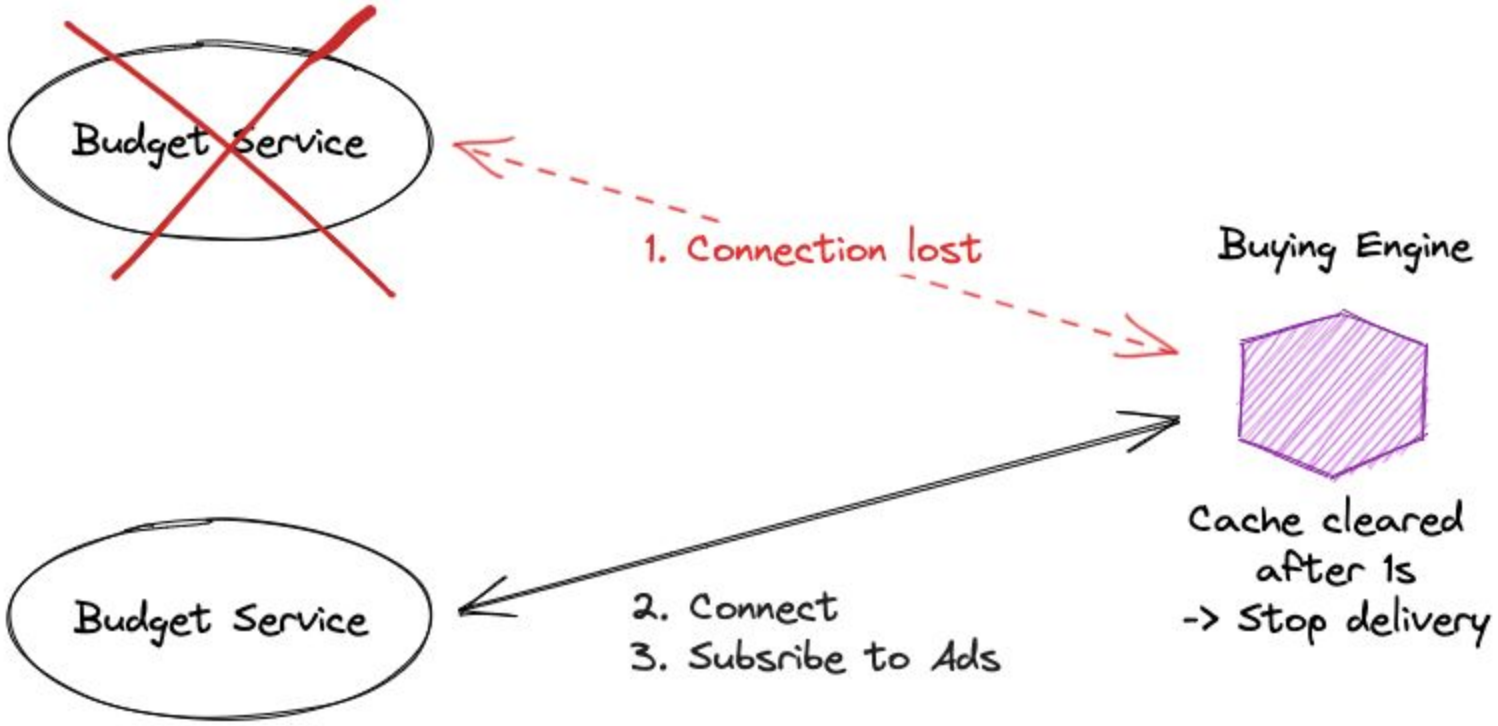
The Observer Pattern



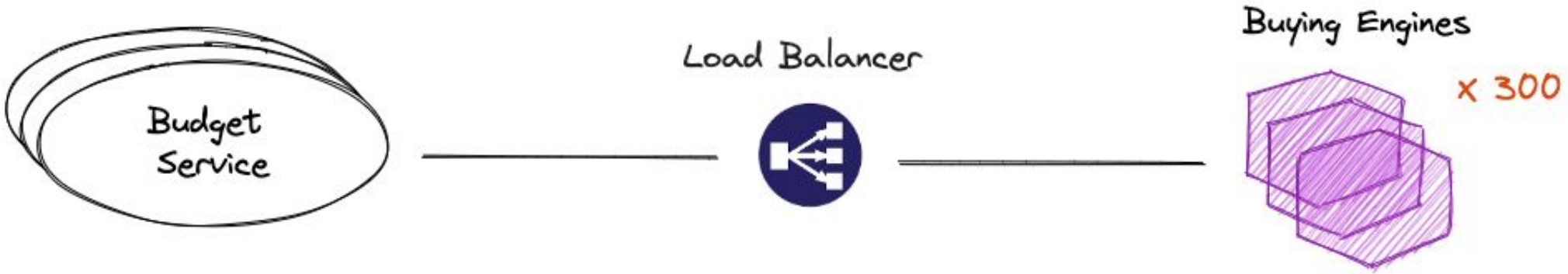
Disconnections



Disconnections



Load balancing



Implementation

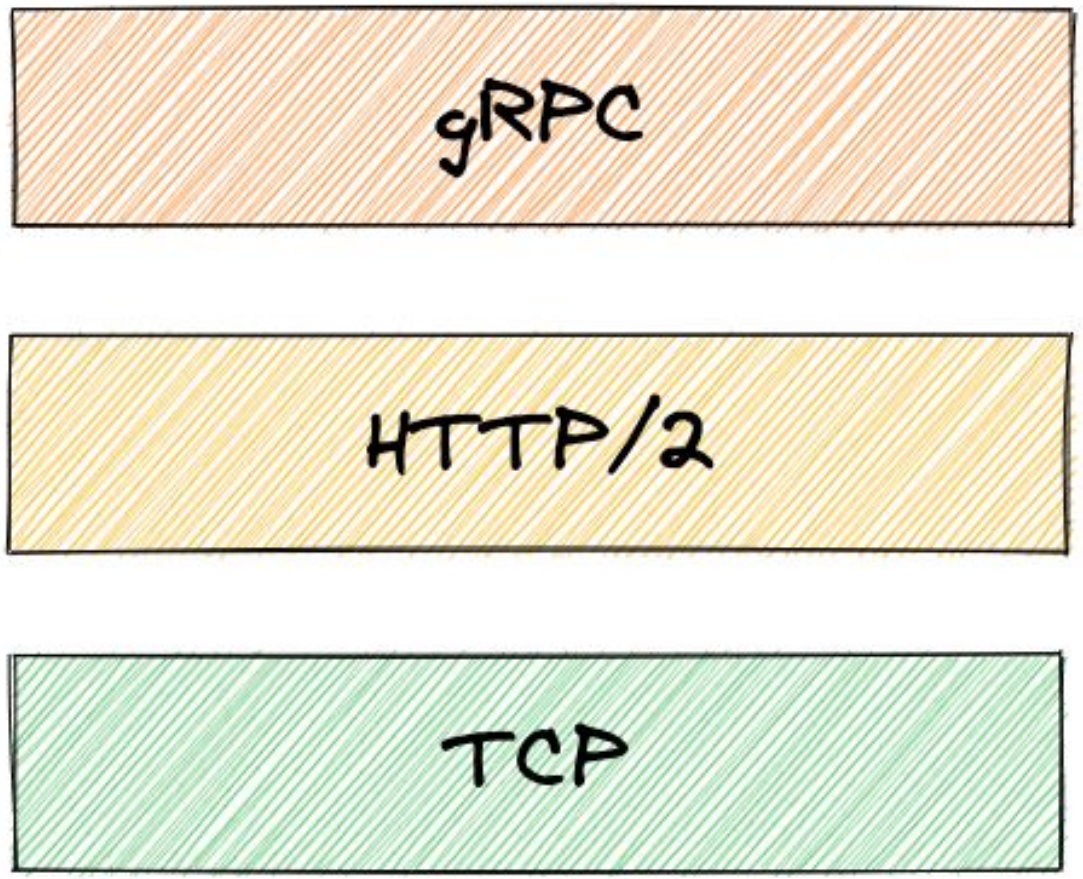
Choices

gRPC

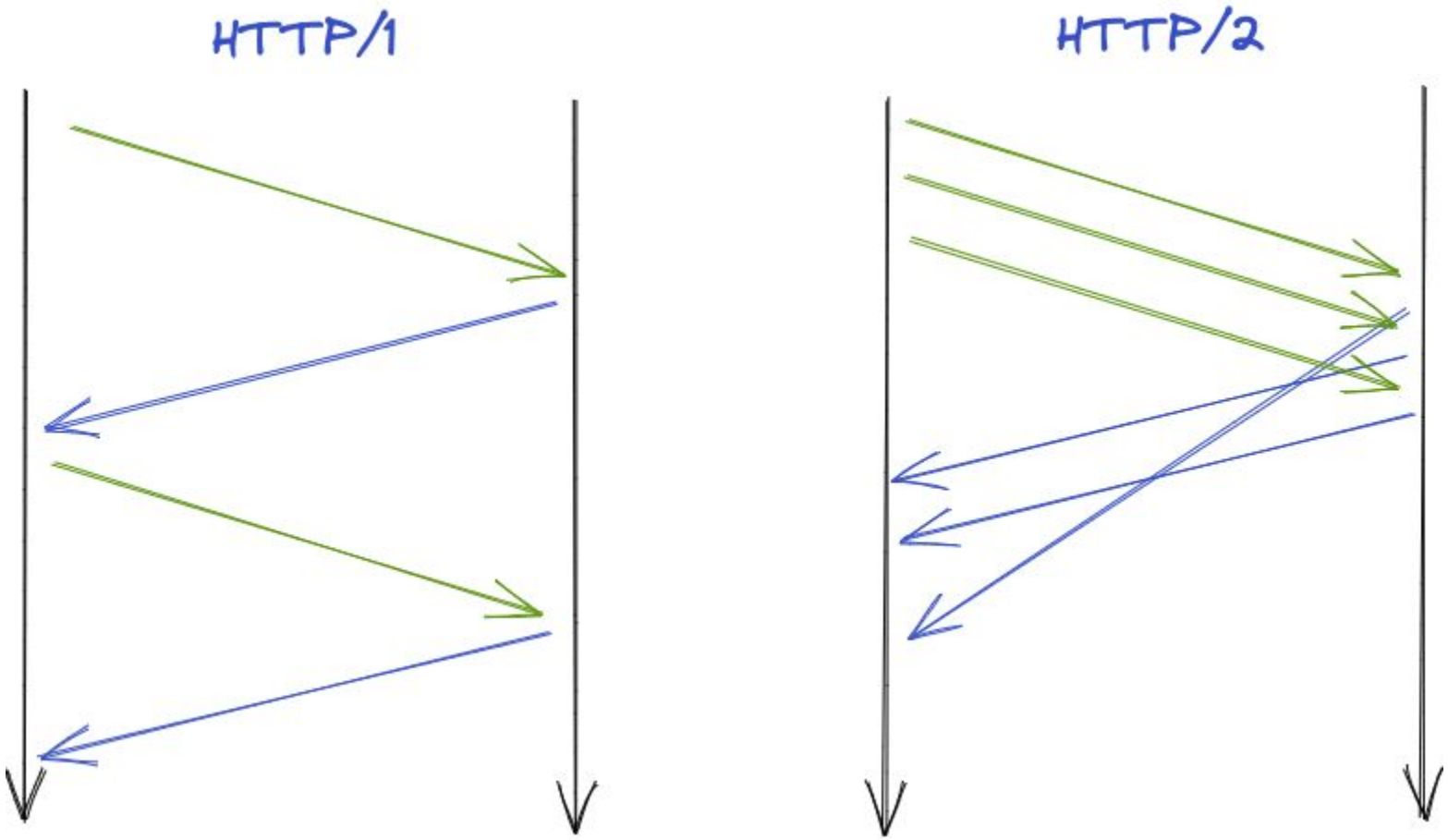
- Standard
- Well supported clients
- Supports bi-directionnel streaming natively

```
service MyService {
   rpc MyMethod (stream Request) returns (stream Response);
}
```

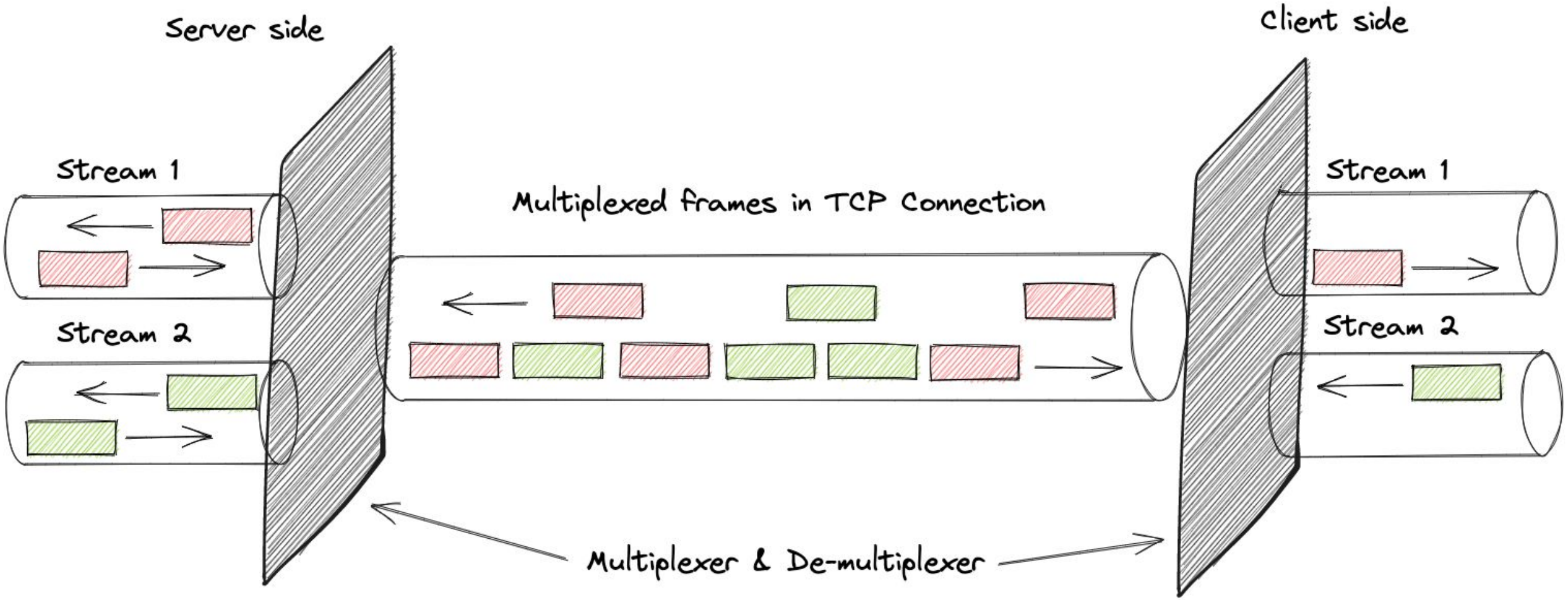
gRPC



Multiplexing



Multiplexing



gRPC - HTTP/2 Mapping

gRPC	HTTP/2
Channel	Connection
Call	Stream

/!\ Only one parameter

gRPC Features

- Streaming
- Performance
  - HTTP/2
  - Protobuf
- Eco-system (TLS, HTTP/2 servers, clients, proxies)

Service Definition

```
service BudgetService {
   rpc SubscribeToStatus (stream SubscriptionRequest)
       returns (stream AdStatus);
}
```

Service Definition

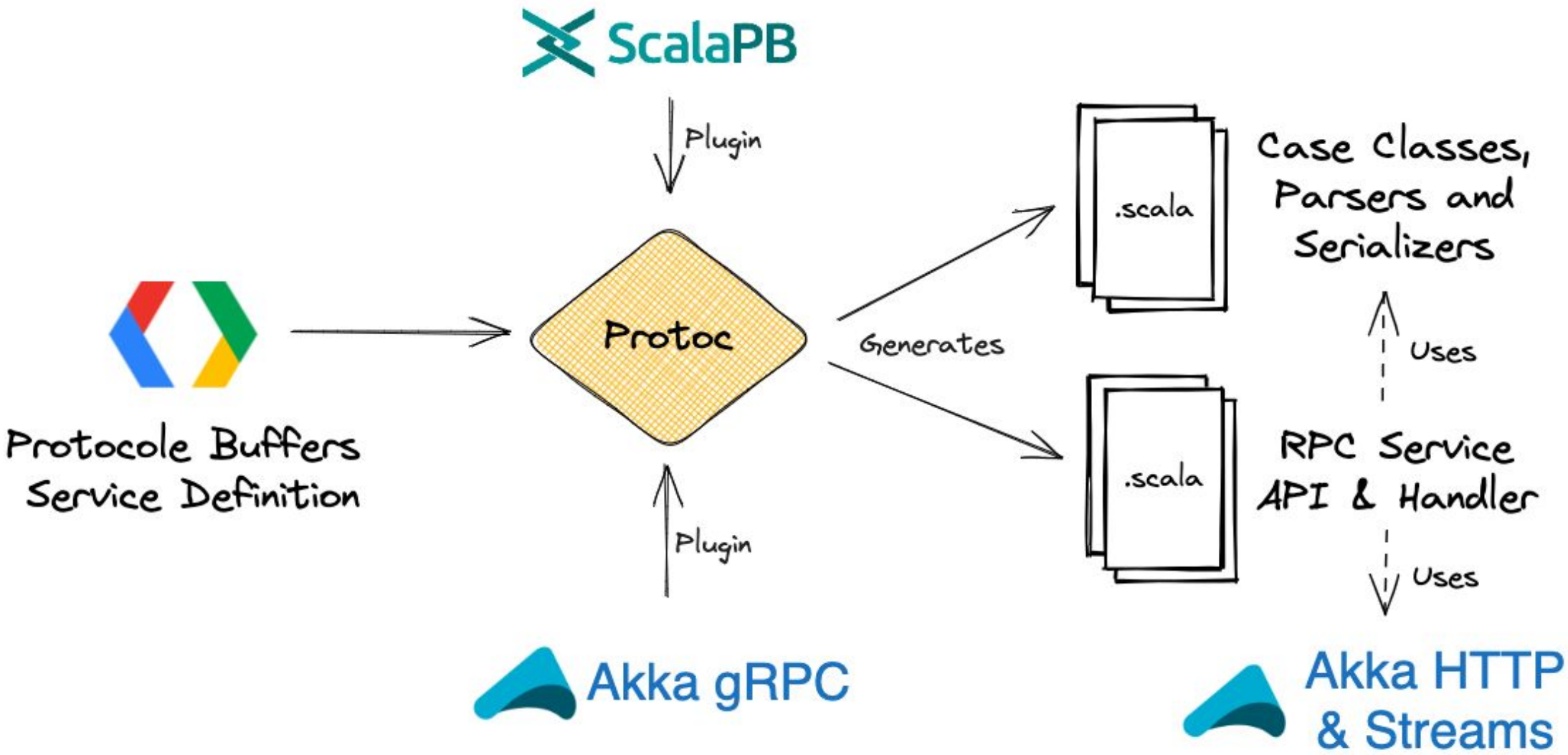
```
service BudgetService {
   rpc SubscribeToStatus (stream SubscriptionRequest)
       returns (stream AdStatus);
}

message SubscriptionRequest {
   int64 ad_id = 1;
   bool subscription_toggle = 2;
}
```

Service Definition

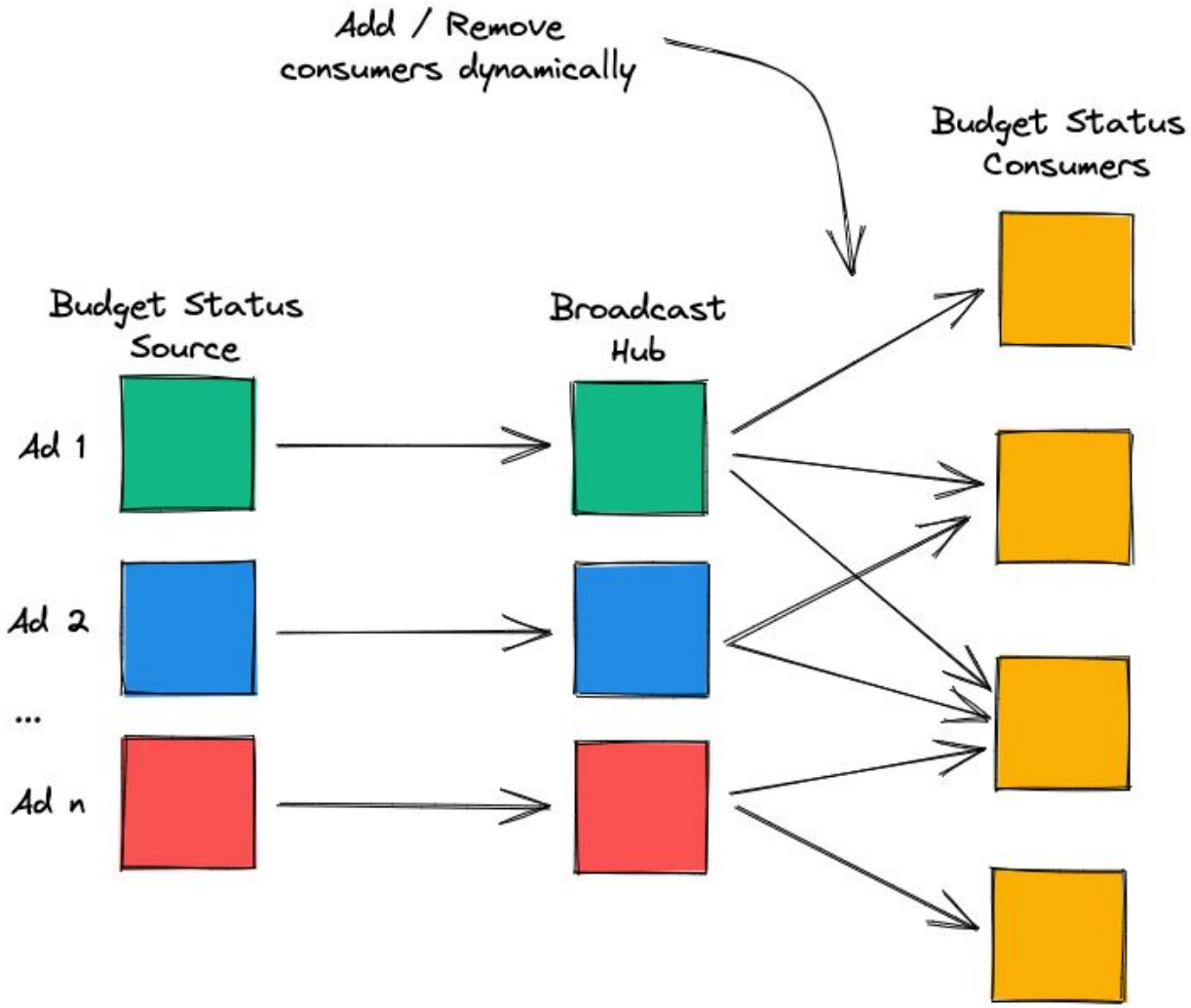
```
service BudgetService {
   rpc SubscribeToStatus (stream SubscriptionRequest)
       returns (stream AdStatus);
}

message SubscriptionRequest {
   int64 ad_id = 1;
   bool subscription_toggle = 2;
}

enum Status {
   BudgetAvailable = 0;
   BudgetExhausted = 1;
}

message AdStatus {
   int64 ad_id = 1;
   Status status = 2;
}
```

Scala gRPC Implementation

gRPC Impl	Stream Library	Runtime
Akka-gRPC	Akka Stream	Akka
FS2-gRPC	FS2	Cats Effects
ZIO-gRPC	ZStream	ZIO


ScalaPB
Plugin
Protocole Buffers
Service Definition
Protoc
Generates
.scala
.scala
Case Classes,
Parsers and
Serializers
Uses
RPC Service
API & Handler
Uses
Plugin
Akka gRPC
Akka HTTP
& Streams

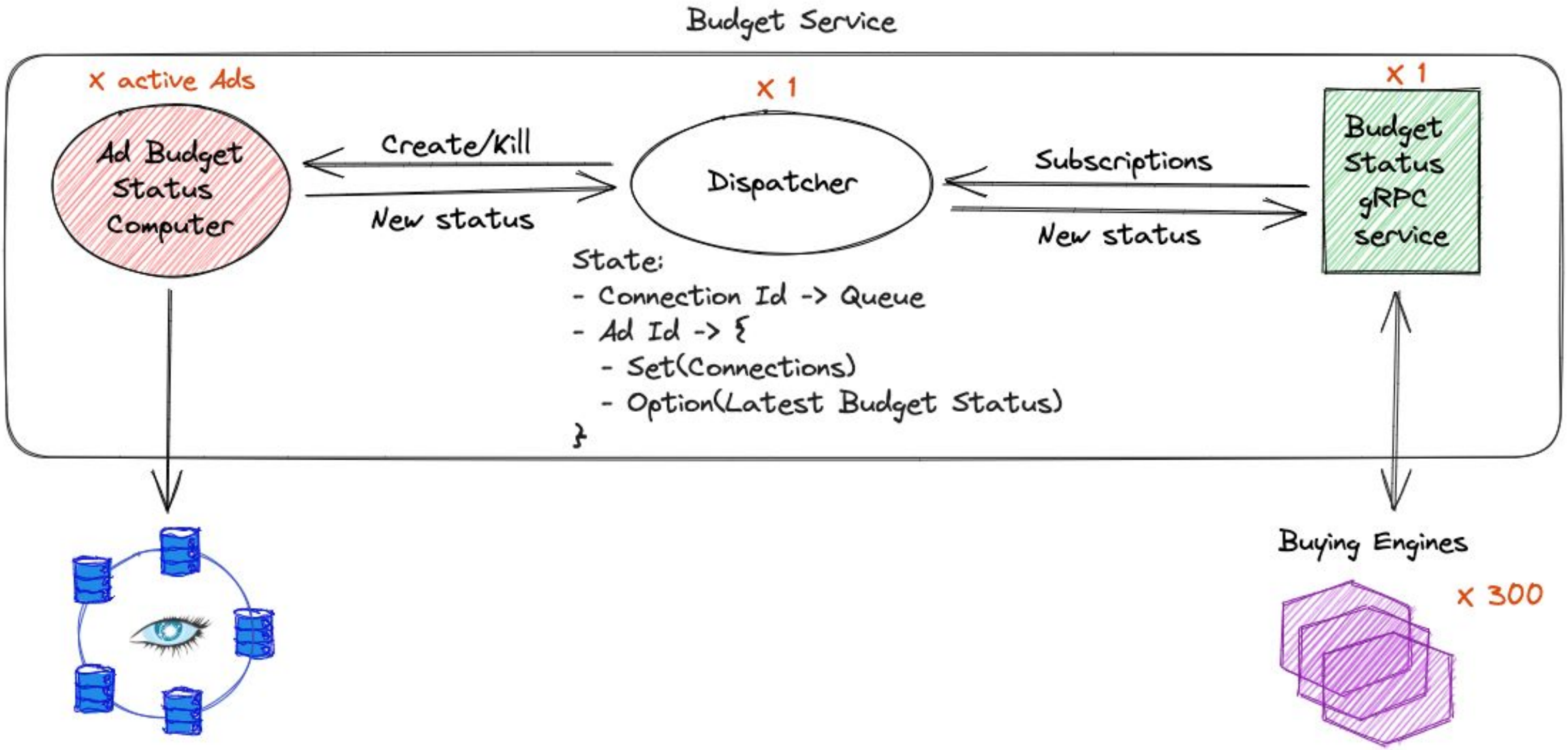
Use Akka Stream & Broadcast Hub ?



Thread Races

Ad 1 Status Computation thread	**gRPC thread**
Recompute Status	Subscribe connection A to Ad 1
	- Read current status Available
Write New Status: Exhausted -	
Read subscriptions -	
	- Publish latest status Available
	- Add subscription
Publish status Exhausted to each -	

Use Actors


Budget Service
x active Ads
Ad Budget Status Computer
Create/Kill
New status
x 1
Dispatcher
Subscriptions
New status
x 1
Budget Status gRPC service
State:
- Connection Id -> Queue
- Ad Id -> {
- Set(Connections)
- Option(Latest Budget Status)
}
Buying Engines
x 300

gRPC Service Implementation

```
class BudgetServiceImpl(dispatcherRef: ActorRef[DispatcherActor.Command])
                        (implicit mat: Materializer) extends BudgetService {

  private implicit val ec: ExecutionContextExecutor = mat.executionContext

  override def subscribeToStatus(
    inSrc: Source[SubscriptionRequest, NotUsed]
  ): Source[AdStatus, NotUsed] = ???
}
```

gRPC Service Implementation

```
val (outBoundQueue, outSrc) = Source.queue[AdStatus](1000).preMaterialize()
```

gRPC Service Implementation

```
val (outBoundQueue, outSrc) = Source.queue[AdStatus](1000).preMaterialize()

dispatcherRef ! DispatcherActor.Connect(id, outBoundQueue)
```

gRPC Service Implementation

```
val (outBoundQueue, outSrc) = Source.queue[AdStatus](1000).preMaterialize()

dispatcherRef ! DispatcherActor.Connect(id, outBoundQueue)

inSrc
  .runForeach { request =>
    dispatcherRef ! DispatcherActor.Subscribe(id, AdId(request.adId))
  }
  .onComplete(_ => dispatcherRef ! DispatcherActor.Disconnect(id))

outSrc
```

Feedback
& conclusion

Statistics

- 2 developers
- Allocated time: ~75%
- Total duration: ~5 months
  - Remote observer impl: 2 weeks

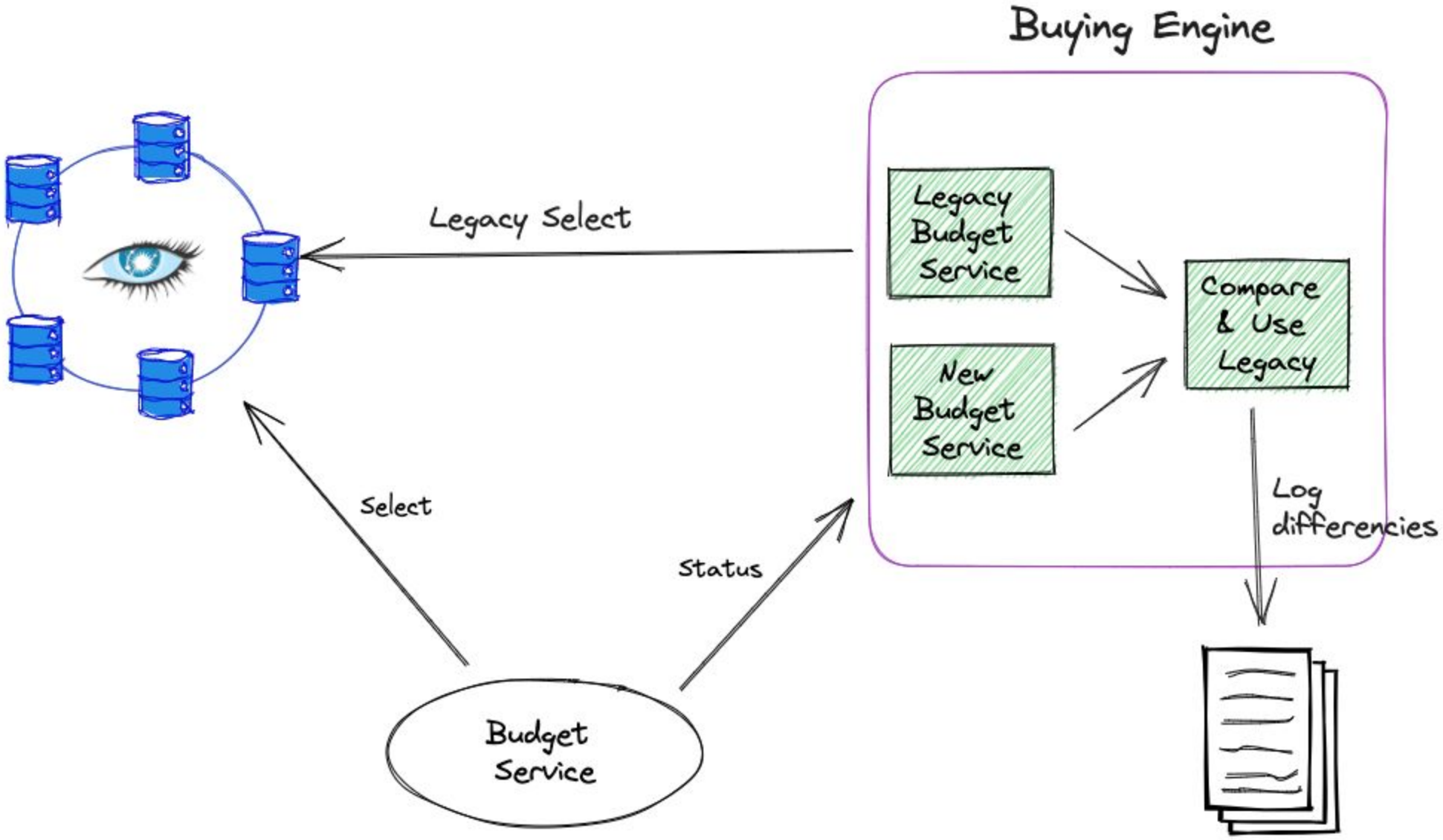
Test Strategy

- Unitary +++
  - BDD
- Integration & Load: ---

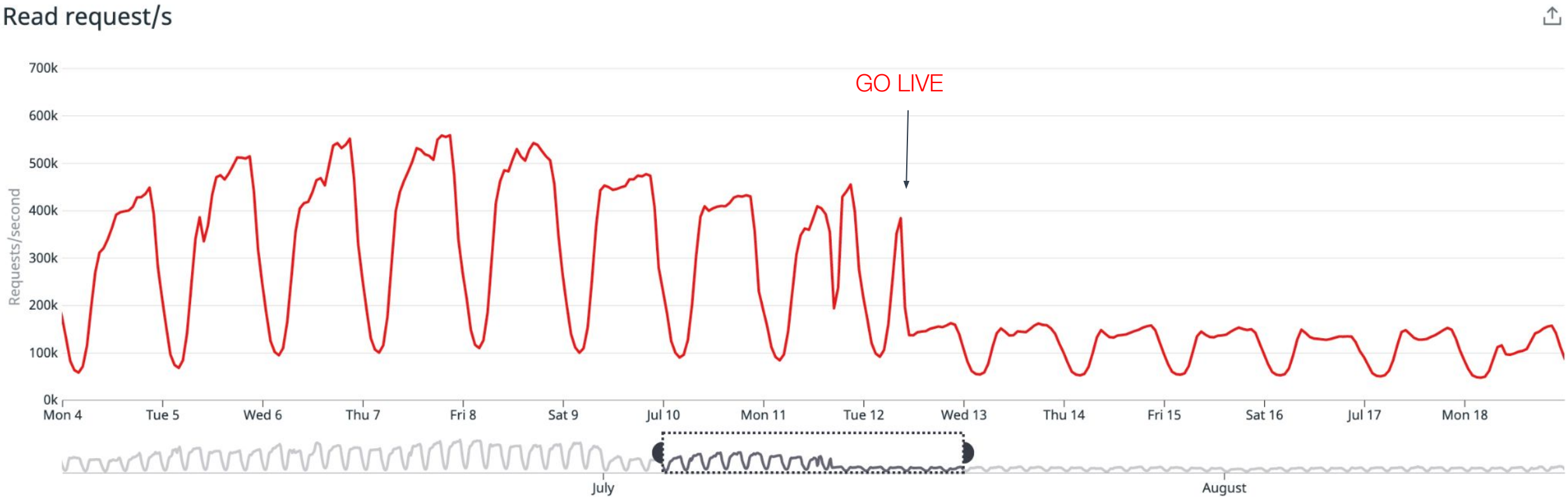
Debug tools

- 60 metrics
  - Budget status distribution
  - 3 consecutive errors
  - Count events: subscriptions / status updates
- Alerts
  - Dispatcher lag
- /debug route

Double Run


Buying Engine
Legacy Select
Legacy Budget Service
New Budget Service
Compare & Use Legacy
Log differencies
Select
Status
Budget Service

Outcomes


Read request/s
GO LIVE
Requests/second
700k
600k
500k
400k
300k
200k
100k
0k
Mon 4
Tue 5
Wed 6
Thu 7
Fri 8
Sat 9
Jul 10
Mon 11
Tue 12
Wed 13
Thu 14
Fri 15
Sat 16
Jul 17
Mon 18
July
August

What’s next ?

- Migration vers Pekko

Takeaways

- gRPC
- Remote Observer Pattern
- Reliable & Sober & Simple

Useful links

- Dynamic Cache Replication Using gRPC Streaming
- The Ad-Tech Book
- Apache Flink
- Apache Cassandra
- Google Protocol Buffers
- Scala PB
- Akka gRPC
- Apache Pekko
- Illustrations were made with excalidraw.com

Questions
& Responses

Feedback - - >

< - - Code sample

Illustrations made with excalidraw.com