How the Metrics Backend Works at Datadog

Adam / Tahia / SREcon22

dtdg.co/srecon22

DATADOG

Adam Mckaig 👋

- I often seek validation from my dad by mentioning that I have worked at:
  - **Datadog**
  - Google
  - NYTimes
  - Bloomberg
  - UNICEF
- Ask me about:
  - Robots
  - CNC machines
  - Computer games
- Don't bother visiting:
  - adammck.com

DATADOG

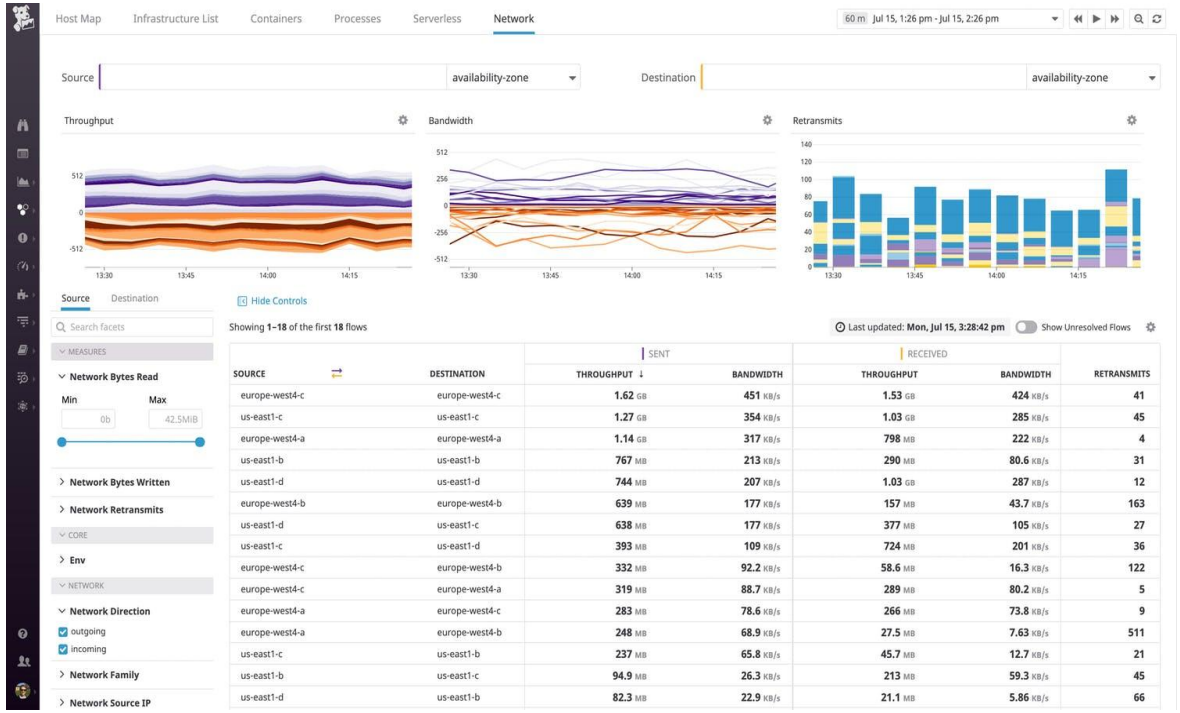
Tahia Khan 🙋🏽‍♀️

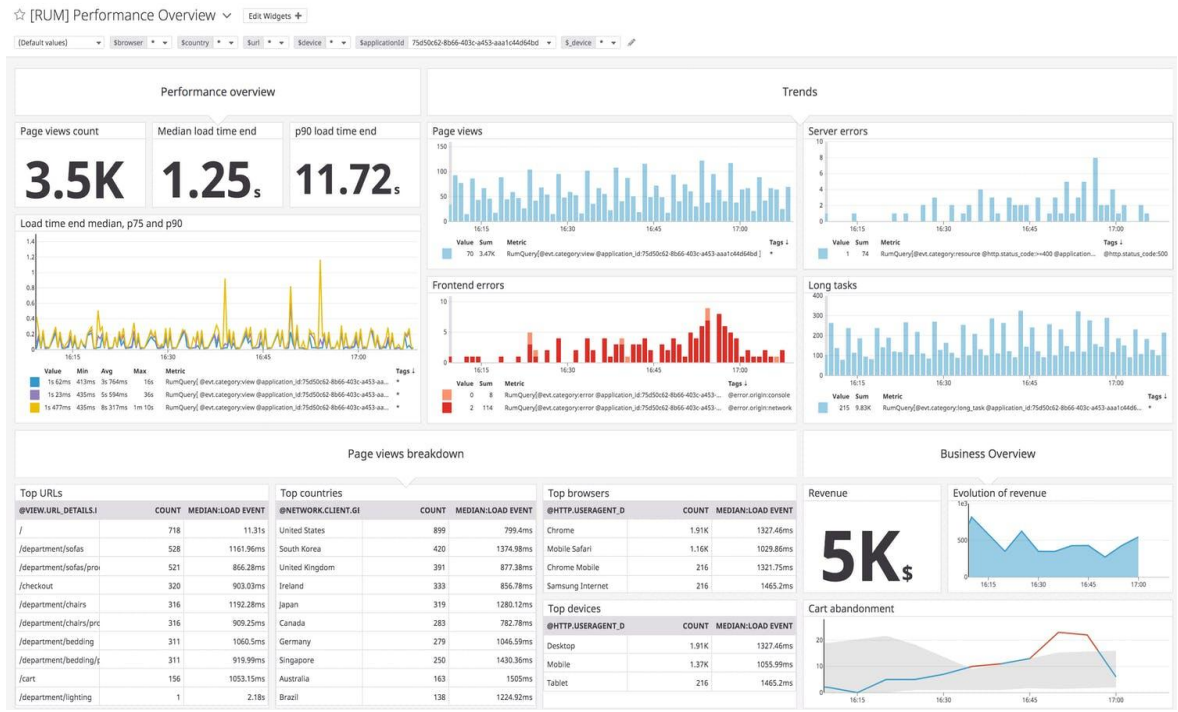
- I too am credible:
  - **Datadog**
  - bunch of start ups
  - Mozilla
  - Amazon
- Ask me about:
  - linux experiments, nixos
  - drawing stuff 👩🏽‍🎨
  - my cat (not pictured), my dog -->
- Don't bother visiting:
  - @pls_tnx
  - i have 8 followers

DATADOG

What is a "Data Dog" ?

- a suite of cloud observability products
- “cloud monitoring as a service”



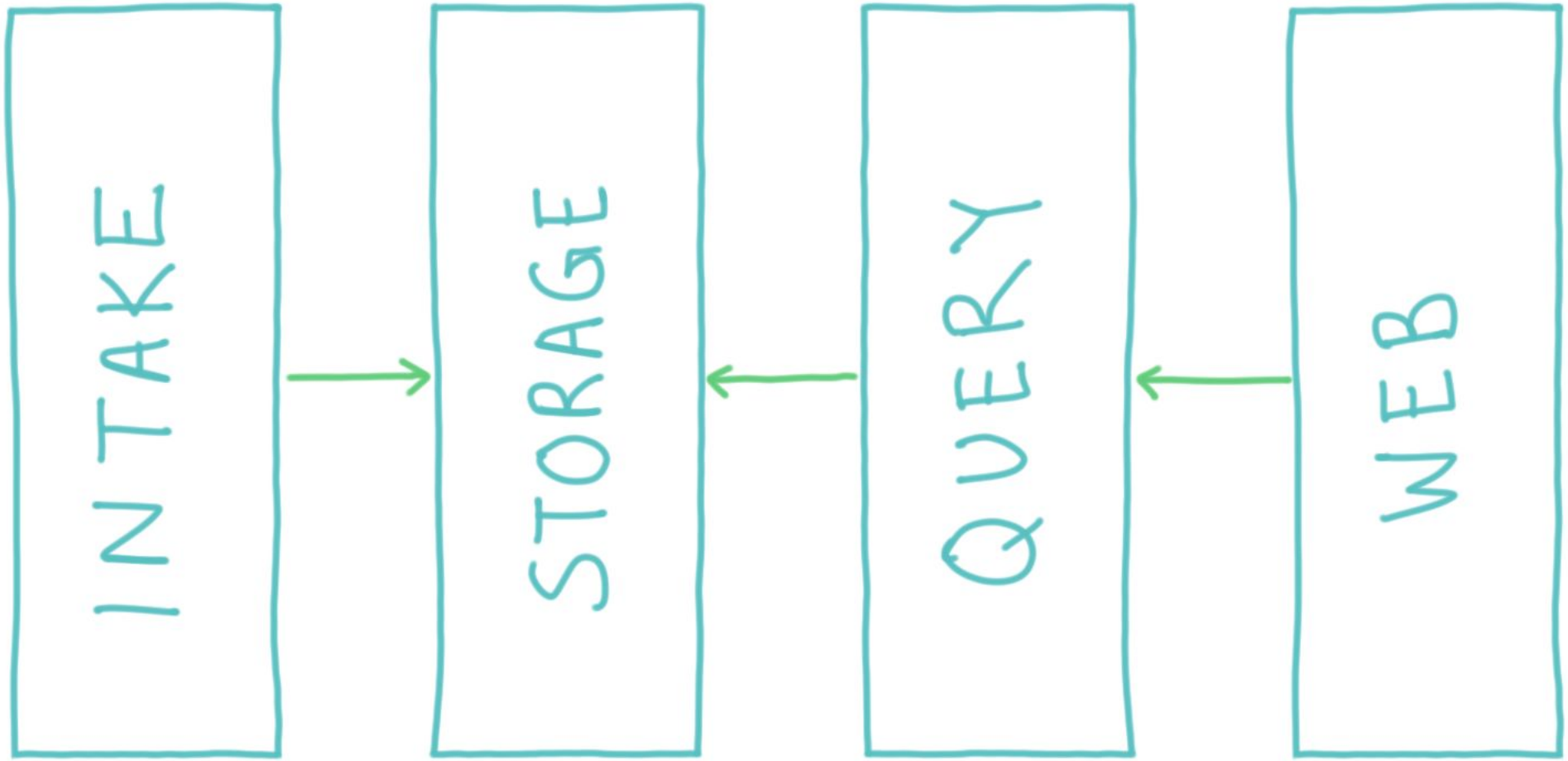


What is this talk about?

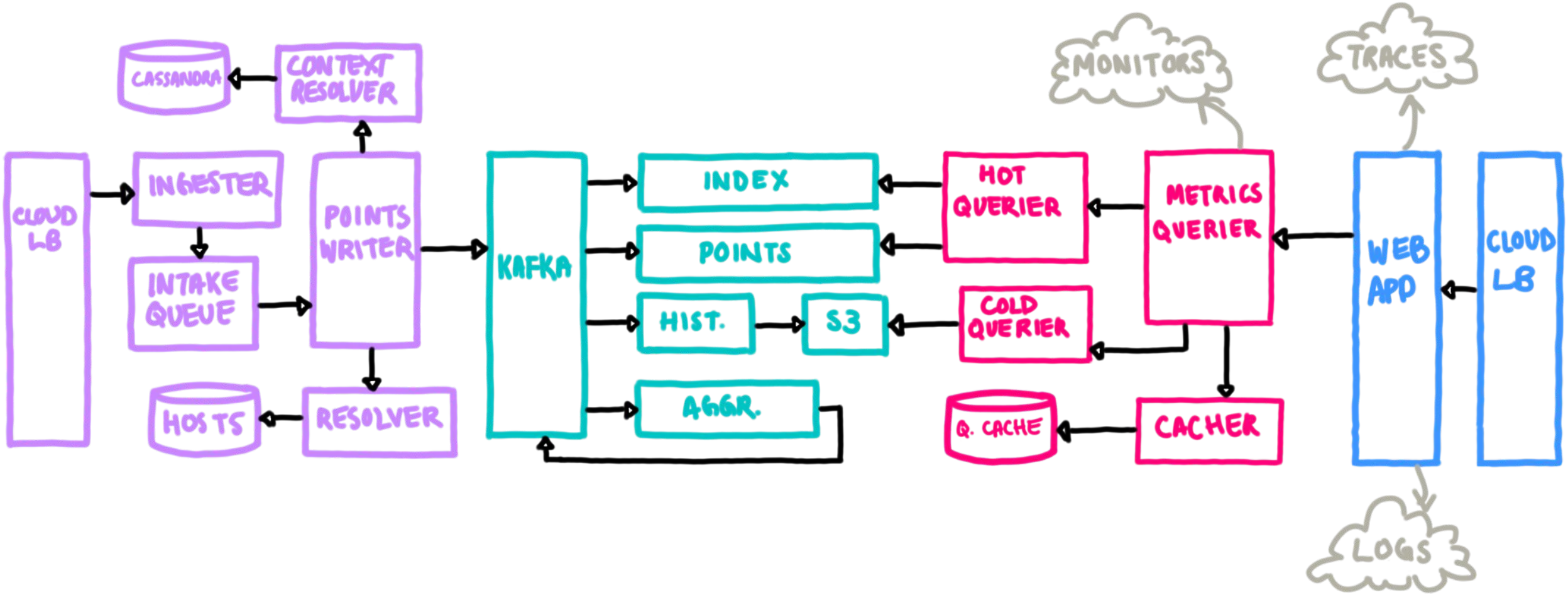
- Scaling is hard
- Reliability is hard
- Our problems are not all unique
- We have learned many things
- It's nice to share
- We are nice
- Q.E.D.

Architecture

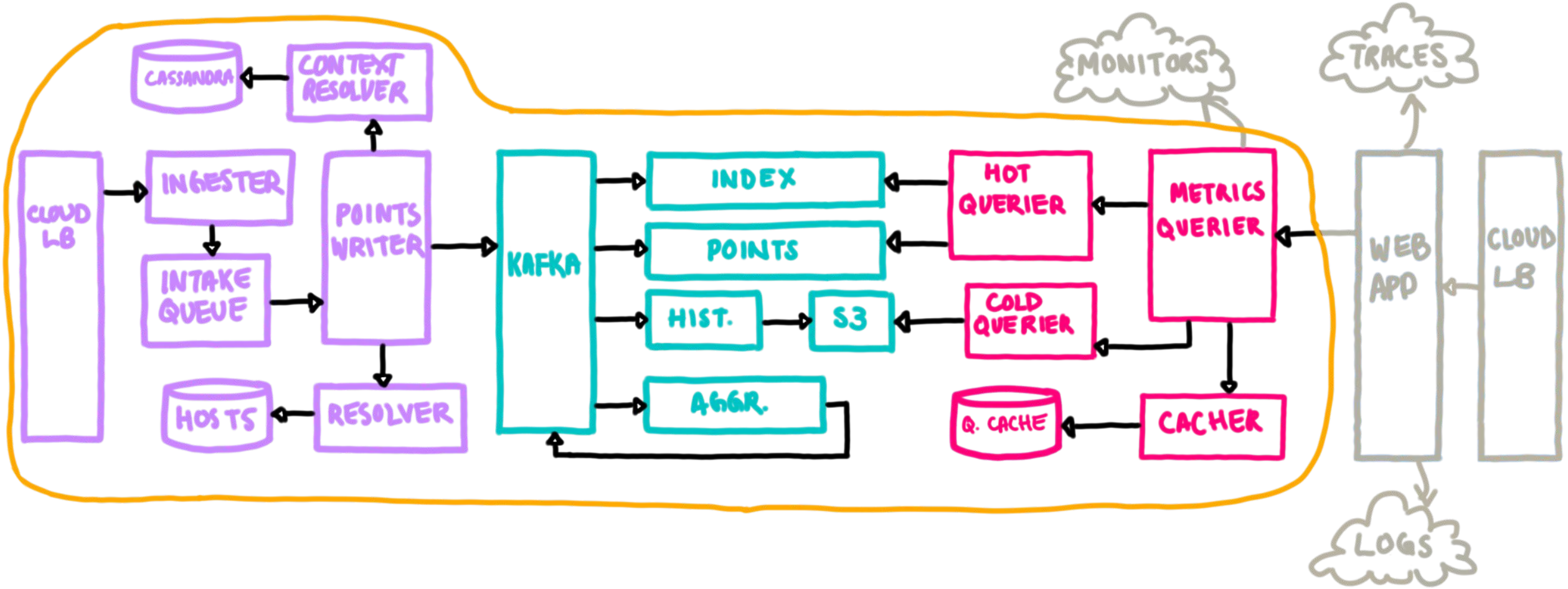
Overview



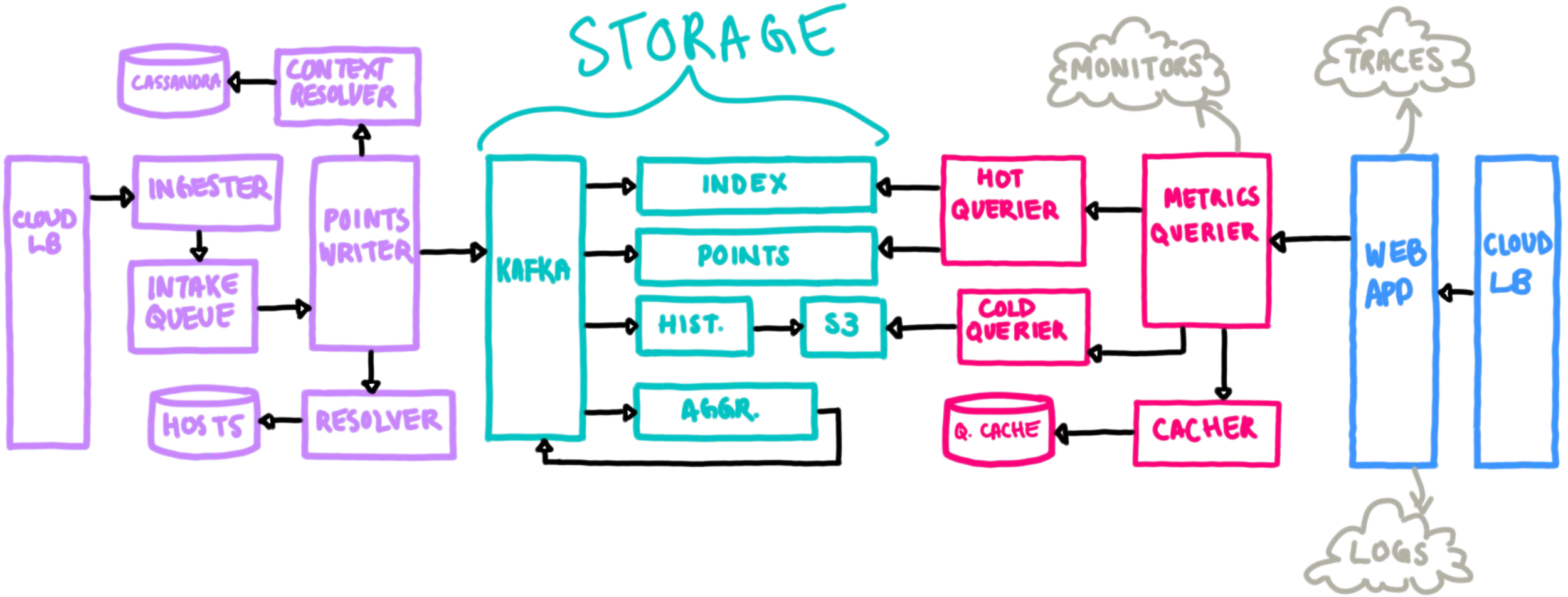
Overview


CASSANDRA
CONTEXT RESOLVER
MONITORS
TRACES
CLOUD LB
INGESTER
POINTS WRITER
KAFKA
INDEX
HOT QUERIER
METRICS QUERIER
WEB APP
CLOUD LB
INTAKE QUEUE
POINTS
HIST.
S3
COLD QUERIER
HOSTS
RESOLVER
AGGR.
Q. CACHE
CACHER
LOGS

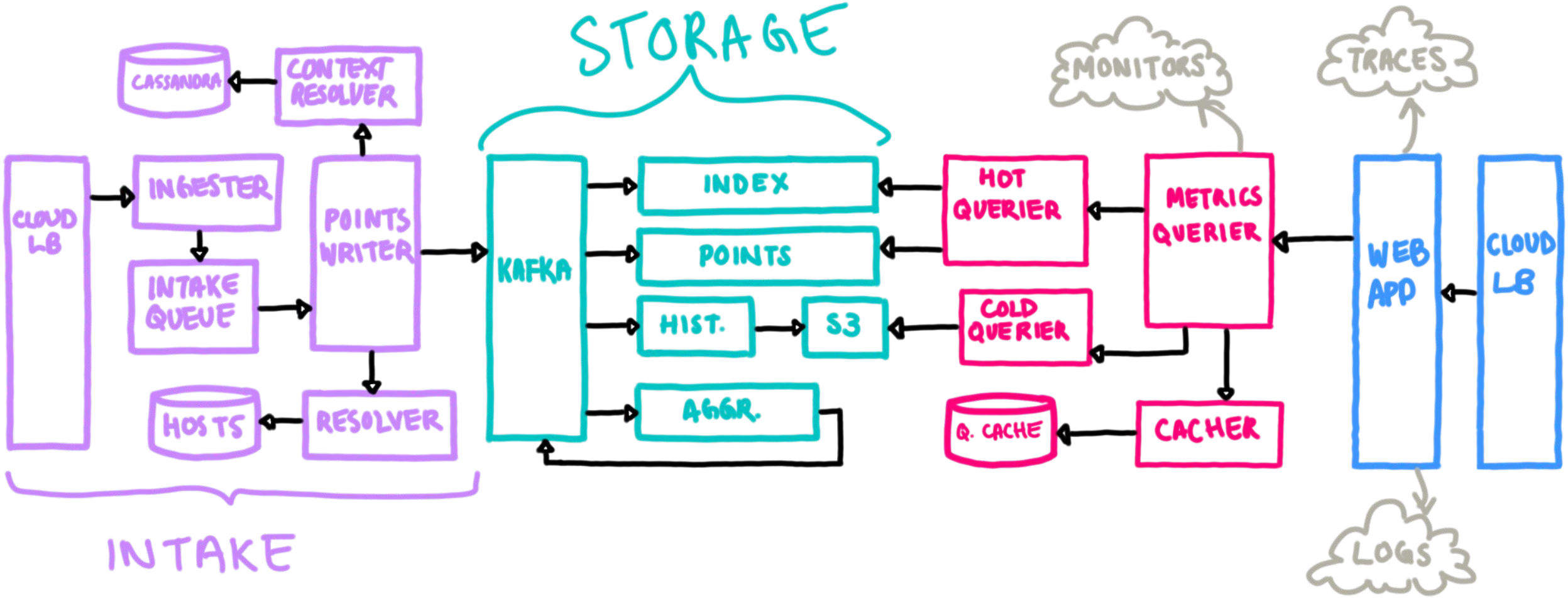
Overview


CASSANDRA
CONTEXT RESOLVER
MONITORS
TRACES
CLOUD LB
INGESTER
POINTS WRITER
KAFKA
INDEX
HOT QUERIER
METRICS QUERIER
WEB APP
CLOUD LB
INTAKE QUEUE
POINTS
HIST.
S3
COLD QUERIER
HOSTS
RESOLVER
AGGR.
Q. CACHE
CACHER
LOGS

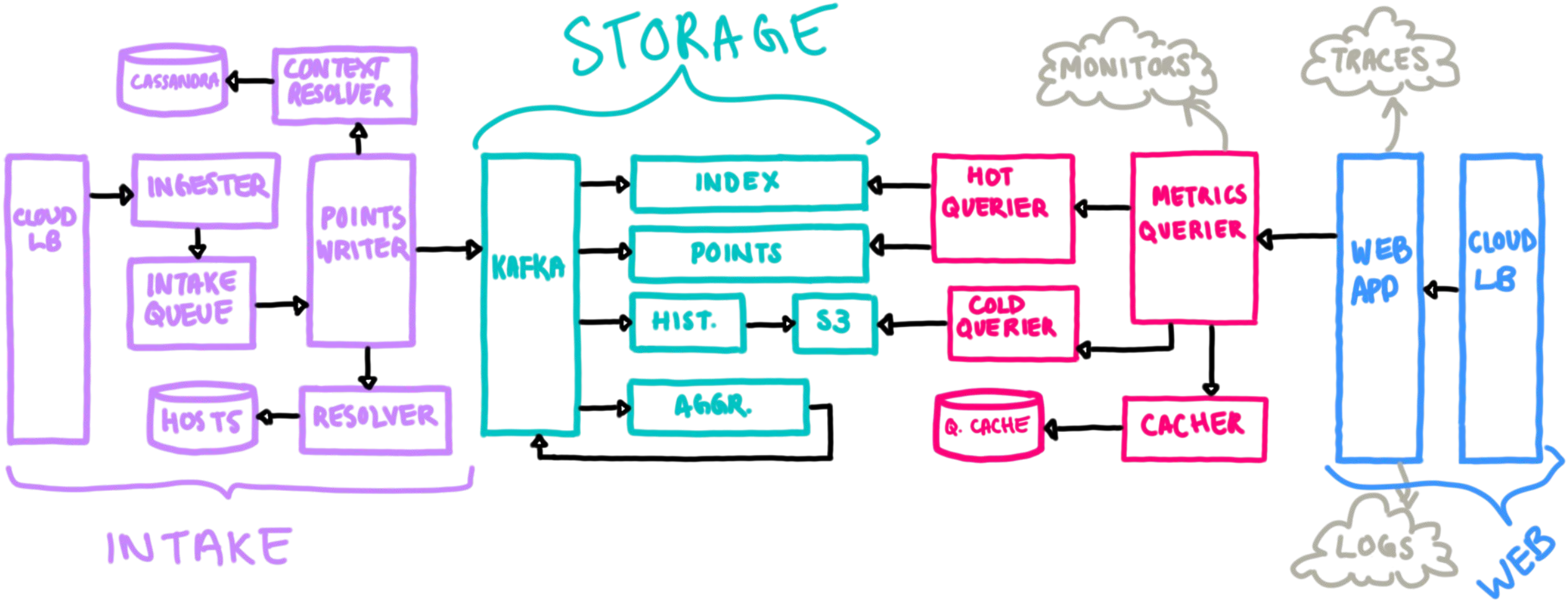
Overview


STORAGE
CASSANDRA
CONTEXT RESOLVER
CLOUD LB
INGESTER
INTAKE QUEUE
POINTS WRITER
HOSTS
RESOLVER
KAFKA
INDEX
POINTS
HIST.
S3
AGGR.
HOT QUERIER
COLD QUERIER
Q. CACHE
METRICS QUERIER
CACHER
MONITORS
TRACES
WEB APP
CLOUD LB
LOGS

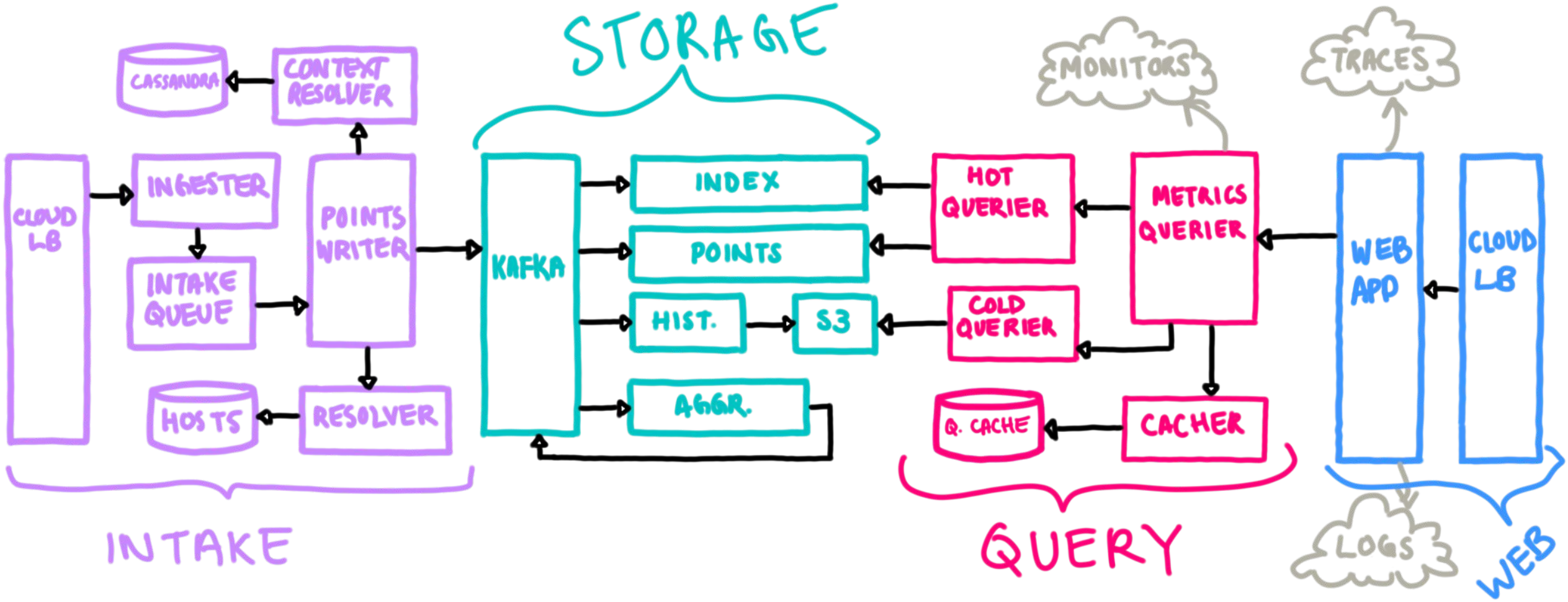
Overview


STORAGE
CASSANDRA
CONTEXT RESOLVER
MONITORS
TRACES
CLOUD LB
INGESTER
POINTS WRITER
KAFKA
INDEX
HOT QUERIER
METRICS QUERIER
WEB APP
CLOUD LB
INTAKE QUEUE
POINTS
HIST.
S3
COLD QUERIER
HOSTS
RESOLVER
AGGR.
Q. CACHE
CACHER
LOGS
INTAKE

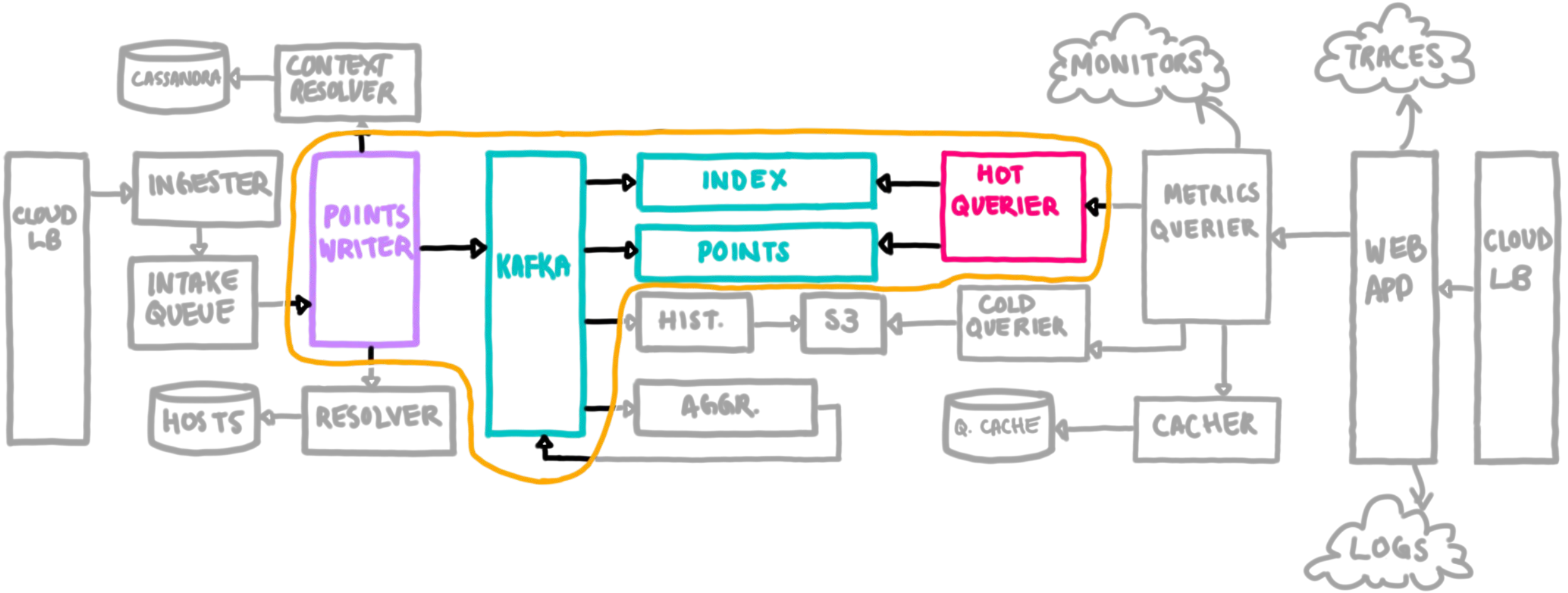
Overview


STORAGE
CASSANDRA
CONTEXT RESOLVER
CLOUD LB
INGESTER
POINTS WRITER
INTAKE QUEUE
HOSTS
RESOLVER
KAFKA
INDEX
POINTS
HIST.
S3
AGGR.
HOT QUERIER
COLD QUERIER
Q. CACHE
METRICS QUERIER
CACHER
MONITORS
TRACES
WEB APP
CLOUD LB
LOGS
WEB
INTAKE

Overview

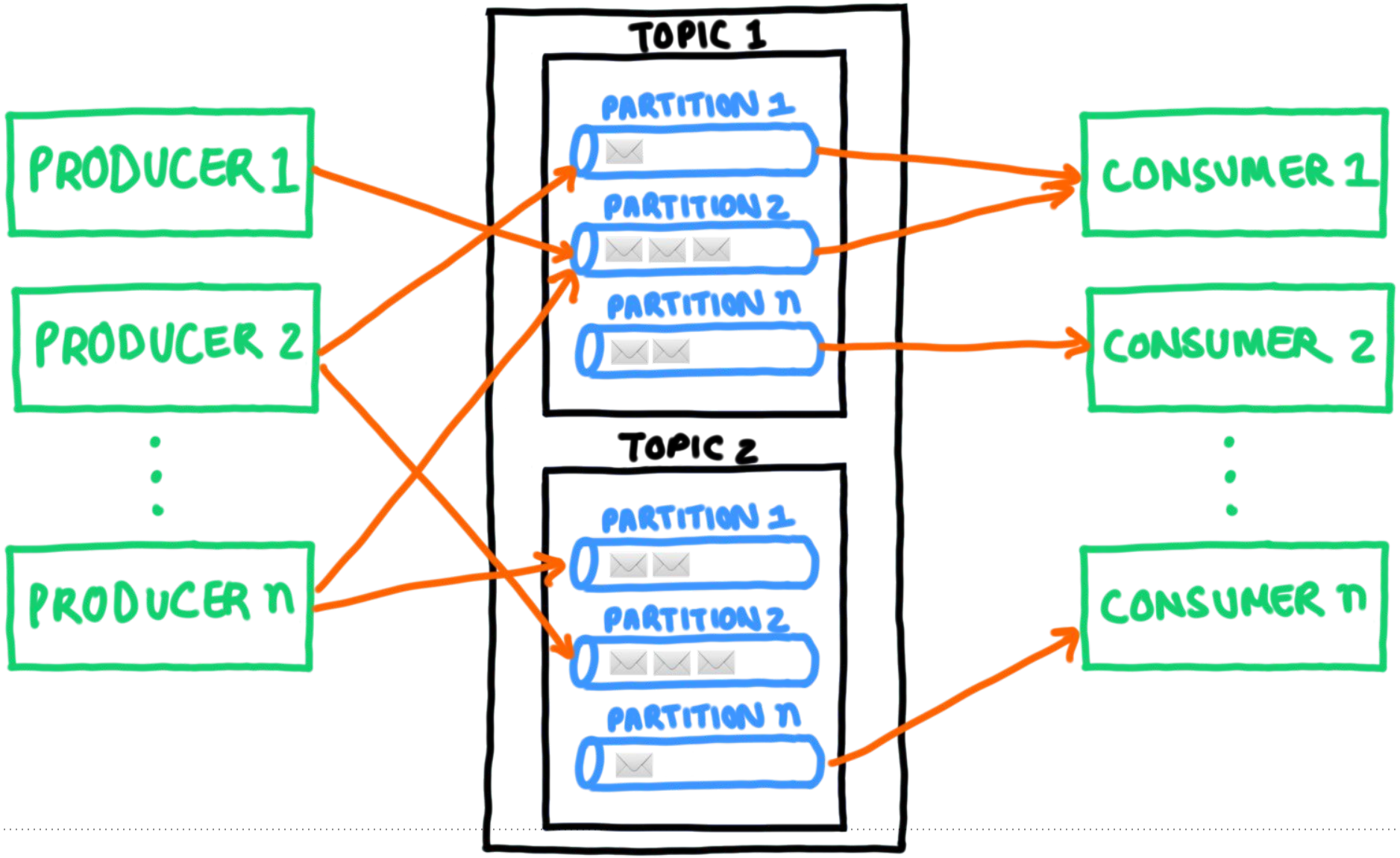

STORAGE
CASSANDRA
CONTEXT RESOLVER
CLOUD LB
INGESTER
POINTS WRITER
INTAKE QUEUE
HOSTS
RESOLVER
KAFKA
INDEX
POINTS
HIST.
S3
AGGR.
HOT QUERIER
METRICS QUERIER
COLD QUERIER
Q. CACHE
CACHER
MONITORS
TRACES
WEB APP
CLOUD LB
LOGS
INTAKE
QUERY
WEB

Overview

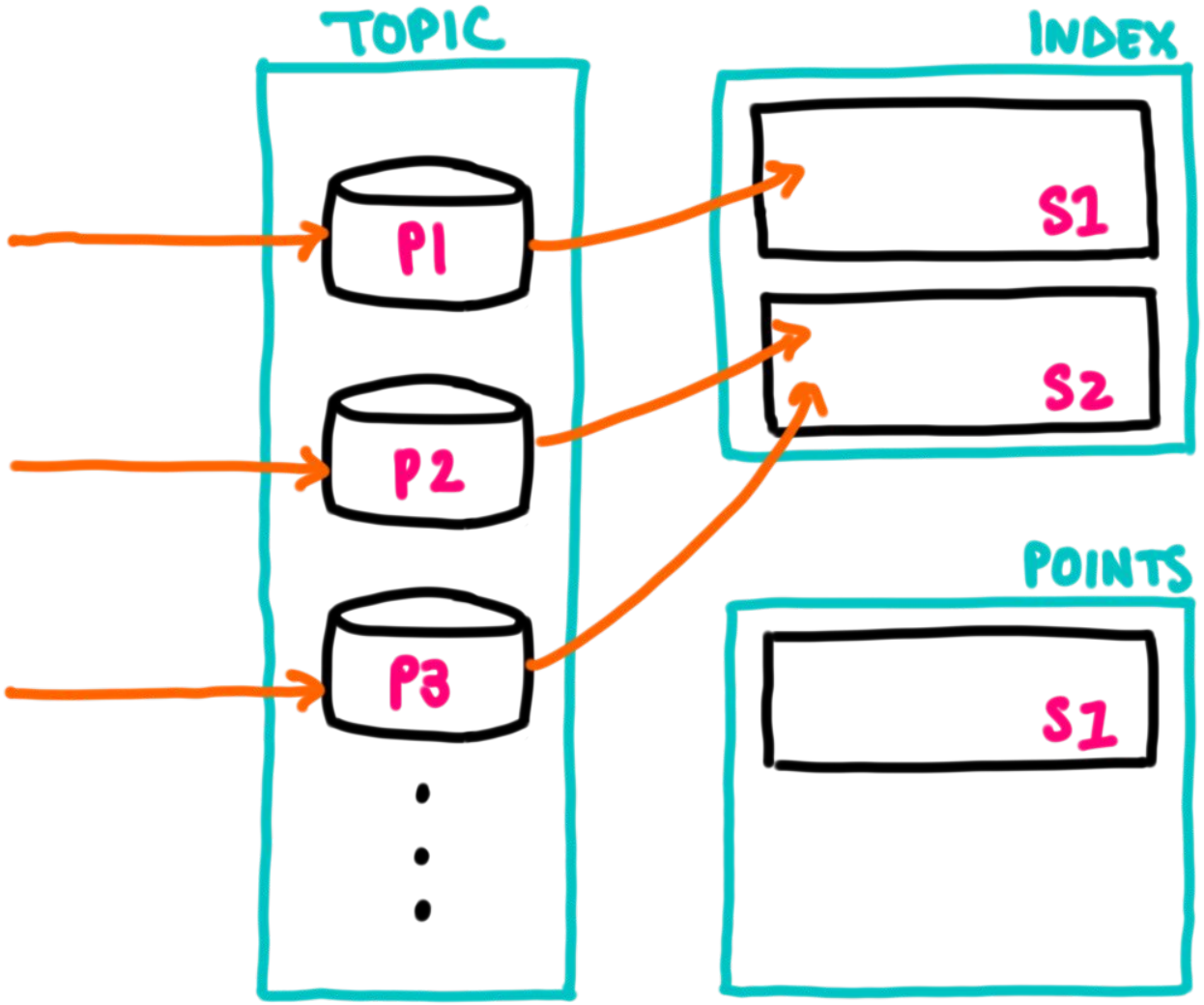

CASSANDRA
CONTEXT RESOLVER
MONITORS
TRACES
CLOUD LB
INGESTER
POINTS WRITER
KAFKA
INDEX
POINTS
HOT QUERIER
METRICS QUERIER
WEB APP
CLOUD LB
INTAKE QUEUE
HIST.
S3
COLD QUERIER
HOSTS
RESOLVER
AGGR.
Q. CACHE
CACHER
LOGS

a Brief Segue into Kafka (at Datadog)

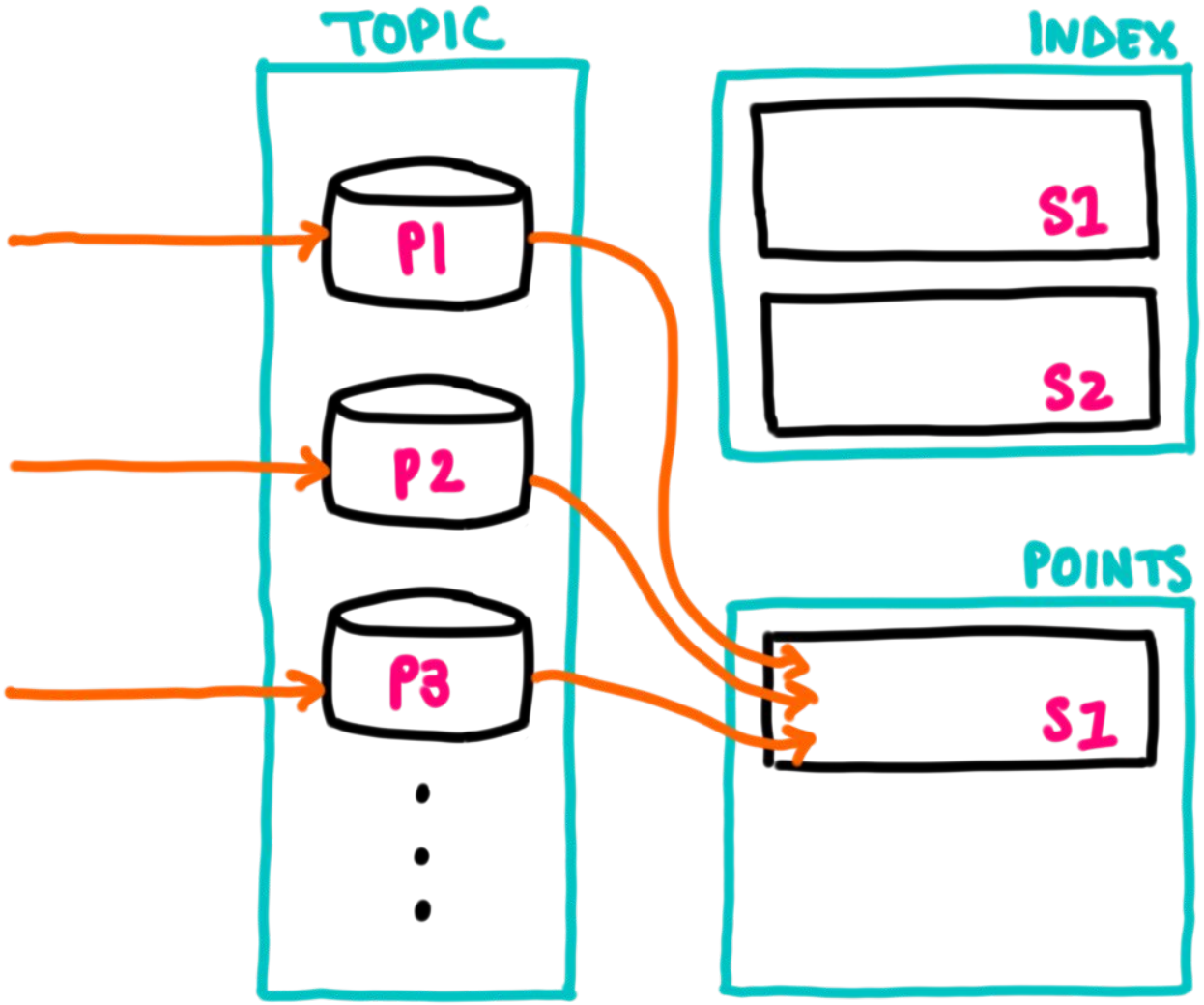
Kafka 101


TOPIC 1
PARTITION 1
PARTITION 2
PARTITION n
TOPIC 2
PARTITION 1
PARTITION 2
PARTITION n
PRODUCER 1
PRODUCER 2
PRODUCER n
CONSUMER 1
CONSUMER 2
CONSUMER n

Kafka at Datadog



Kafka at Datadog



Baby's First Keyfunc 👶

$$p = mod(n, hash(customerID))$$

Kafka at Datadog

$$p = mod(n, hash(customerID))$$

e.g.

$p = 420$

Kafka at Datadog

$$p = mod(n, hash(customerID))$$

e.g.
customerID = 1000
hash(1000) = 2147483556

Kafka at Datadog

$$p = mod(n, hash(customerID))$$

e.g.

$n = 512$

Kafka at Datadog

$$p = mod(n, hash(customerID))$$

e.g.

$$2147483556 \% 512 = 420$$

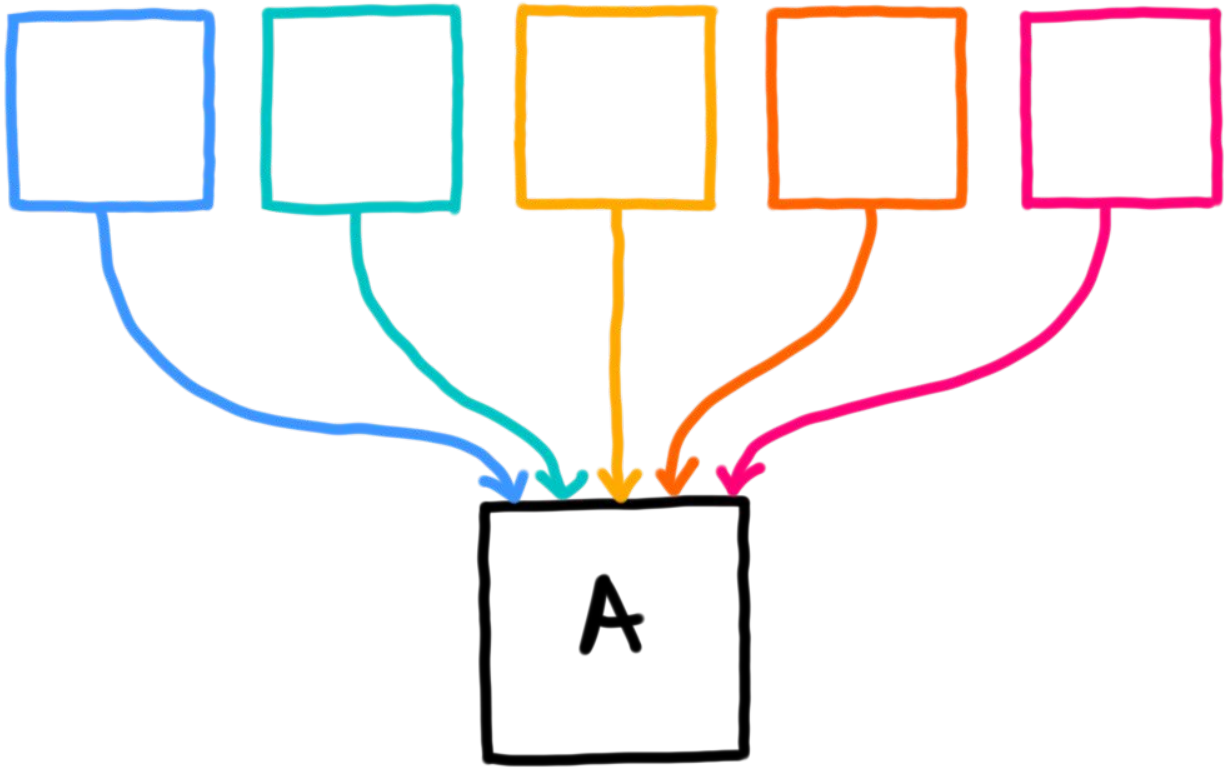
Reliability Challenges

Problem:

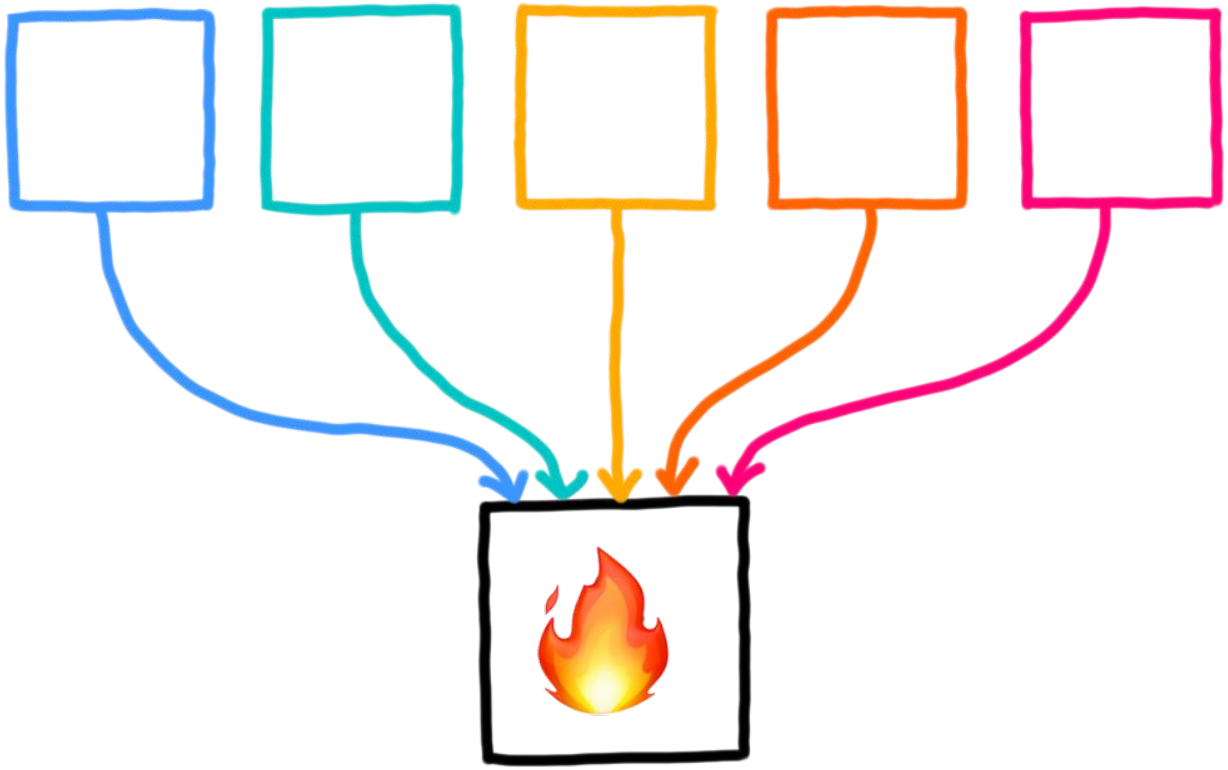
Failures Happen (at Every Level)

DATADOG

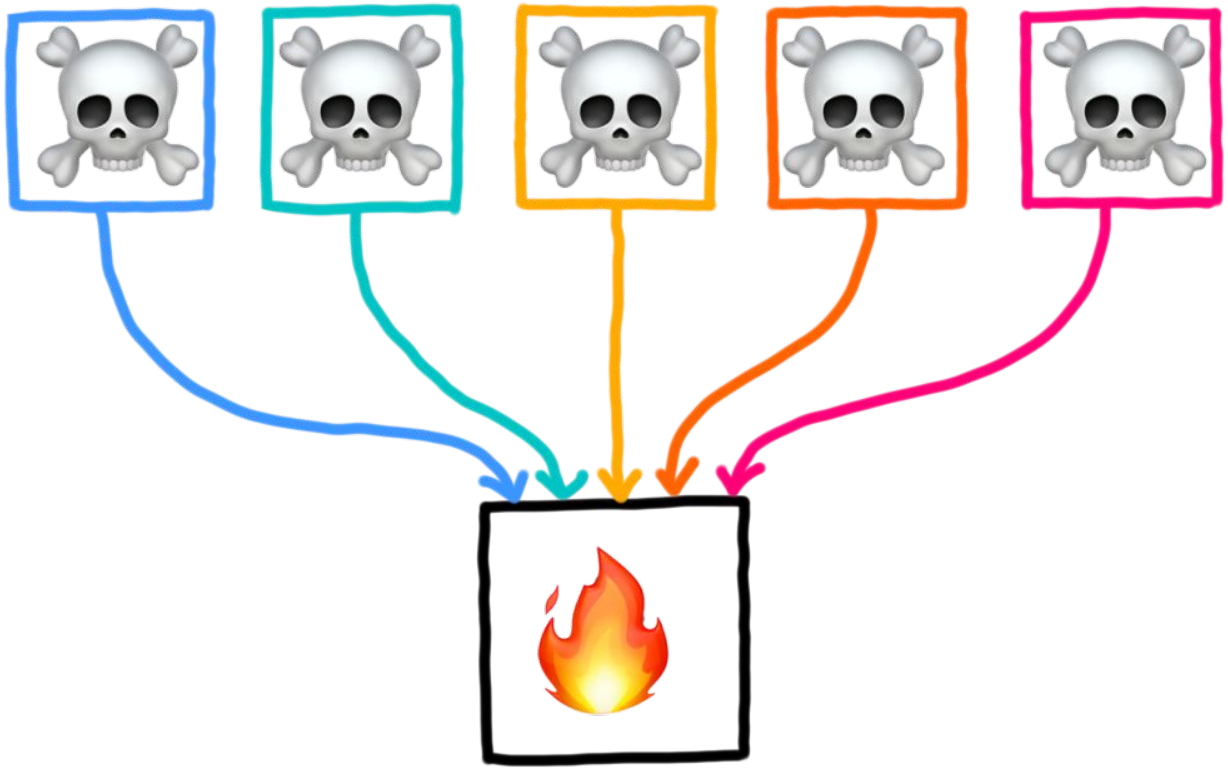
Node Failure



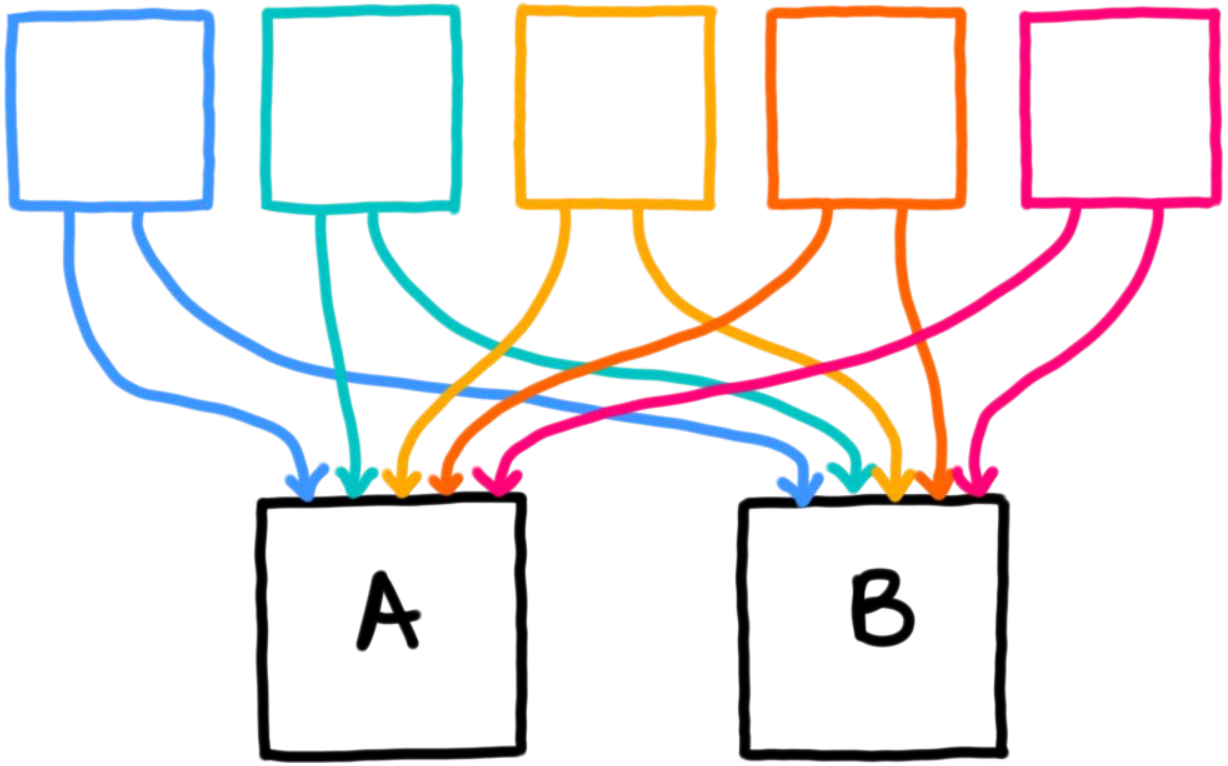
Node Failure



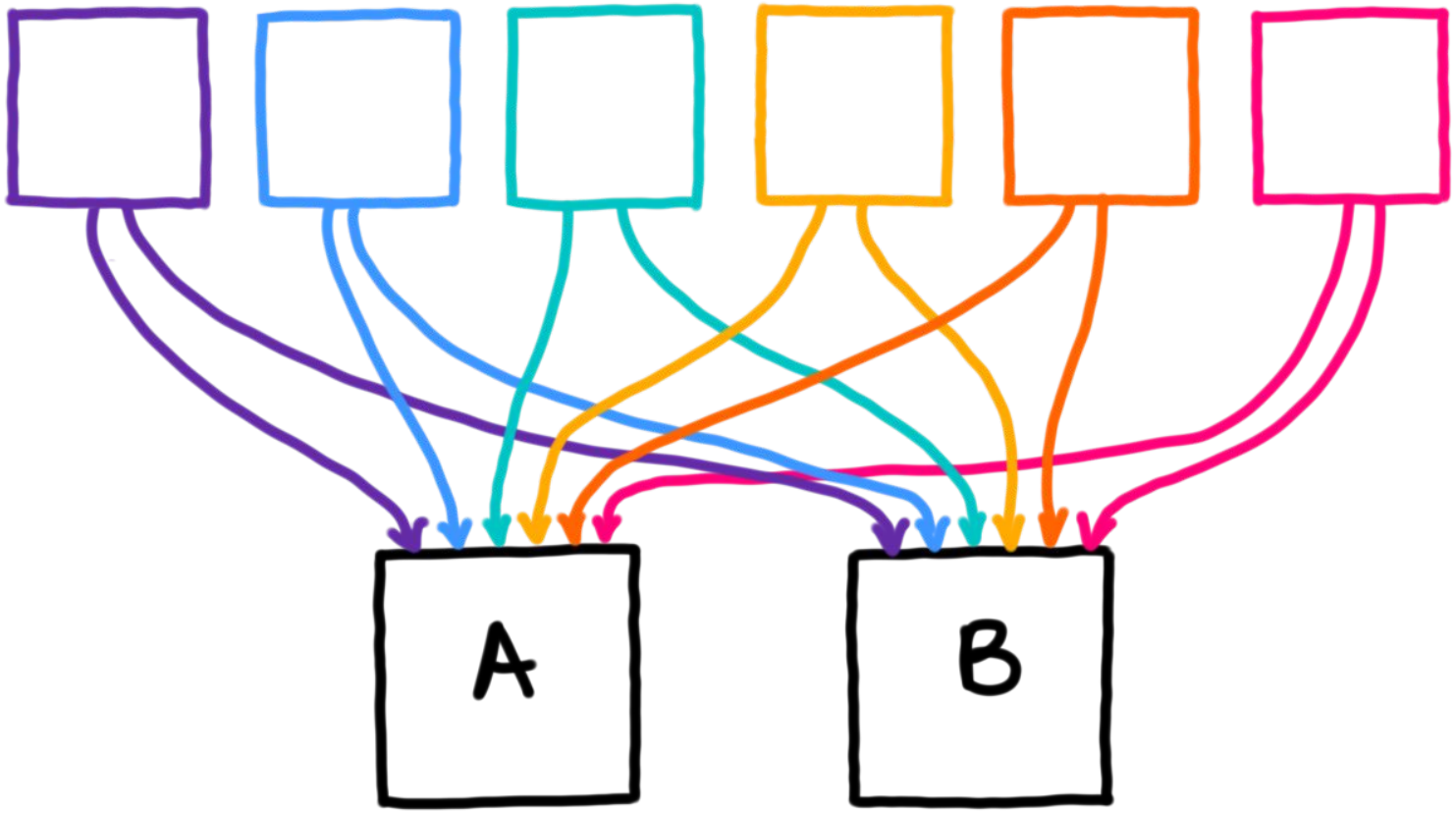
Node Failure



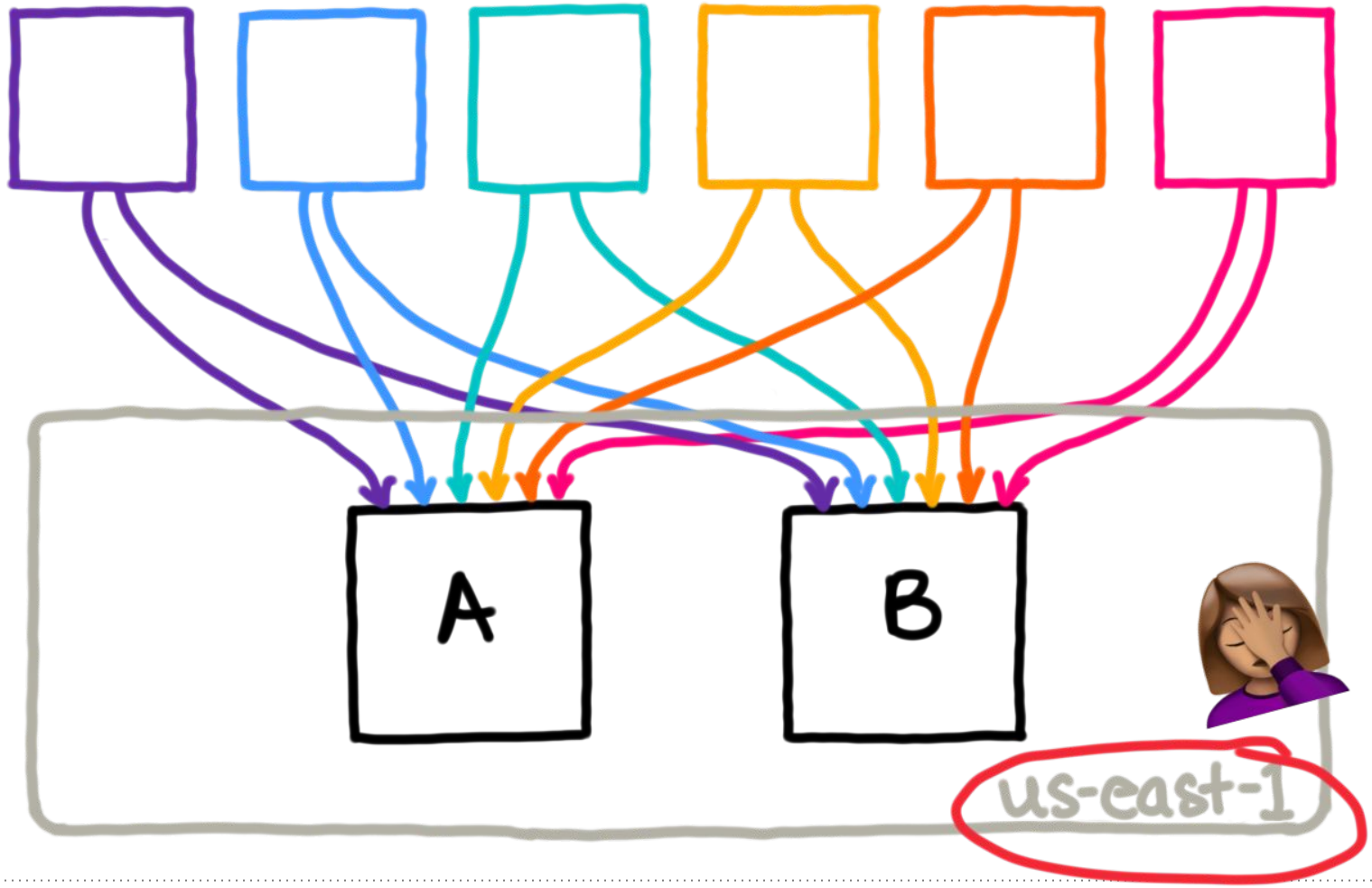
Node Failure



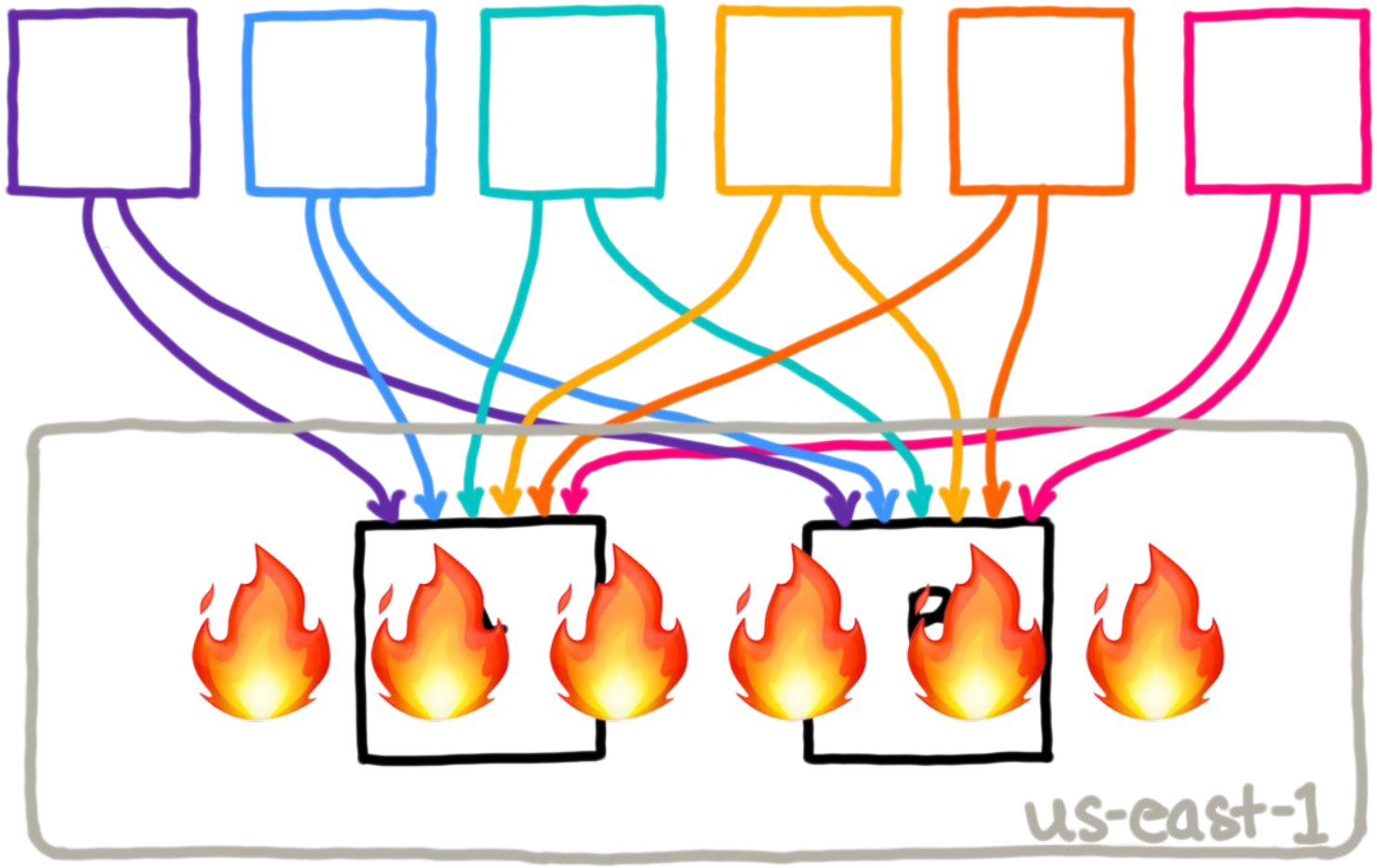
AZ Failure



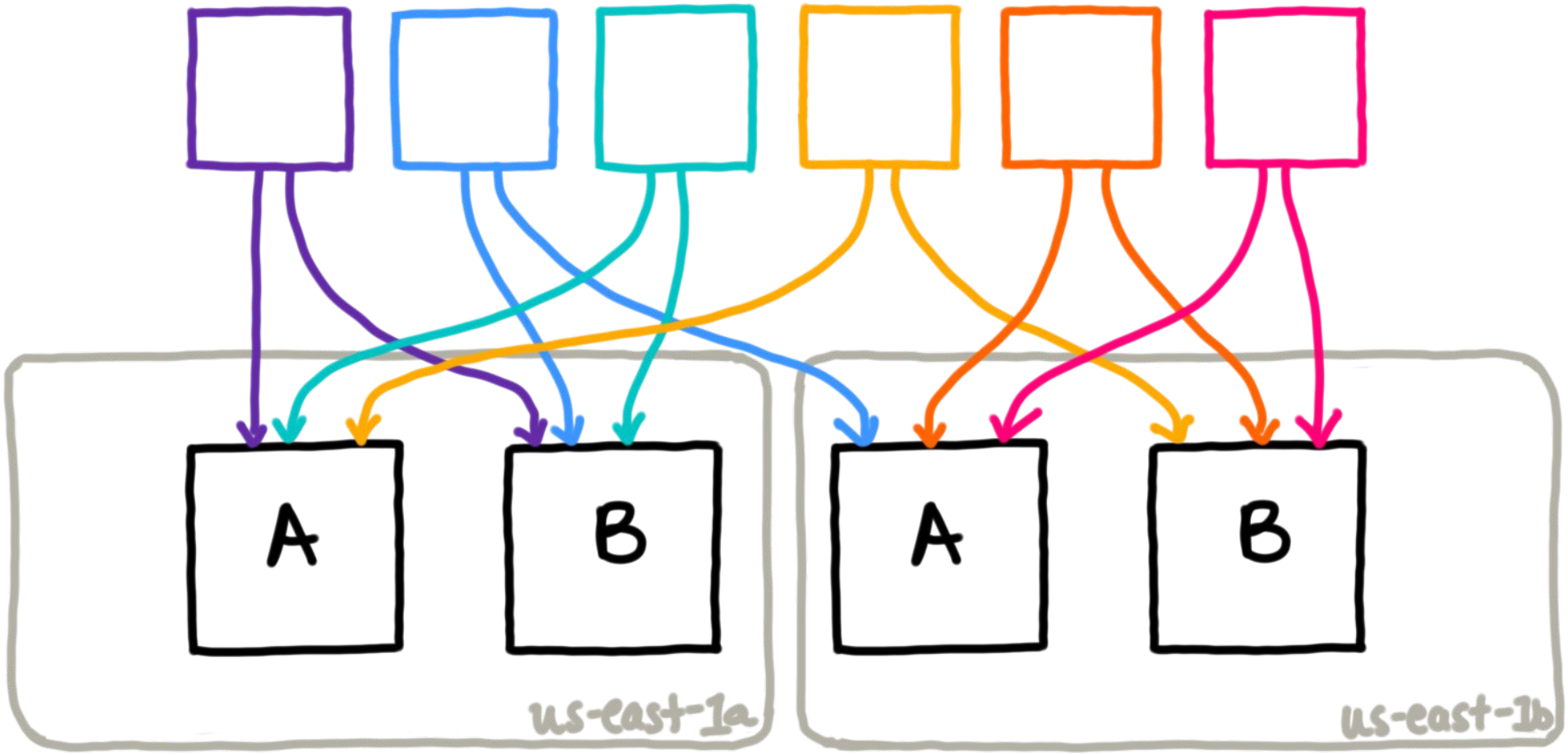
AZ Failure



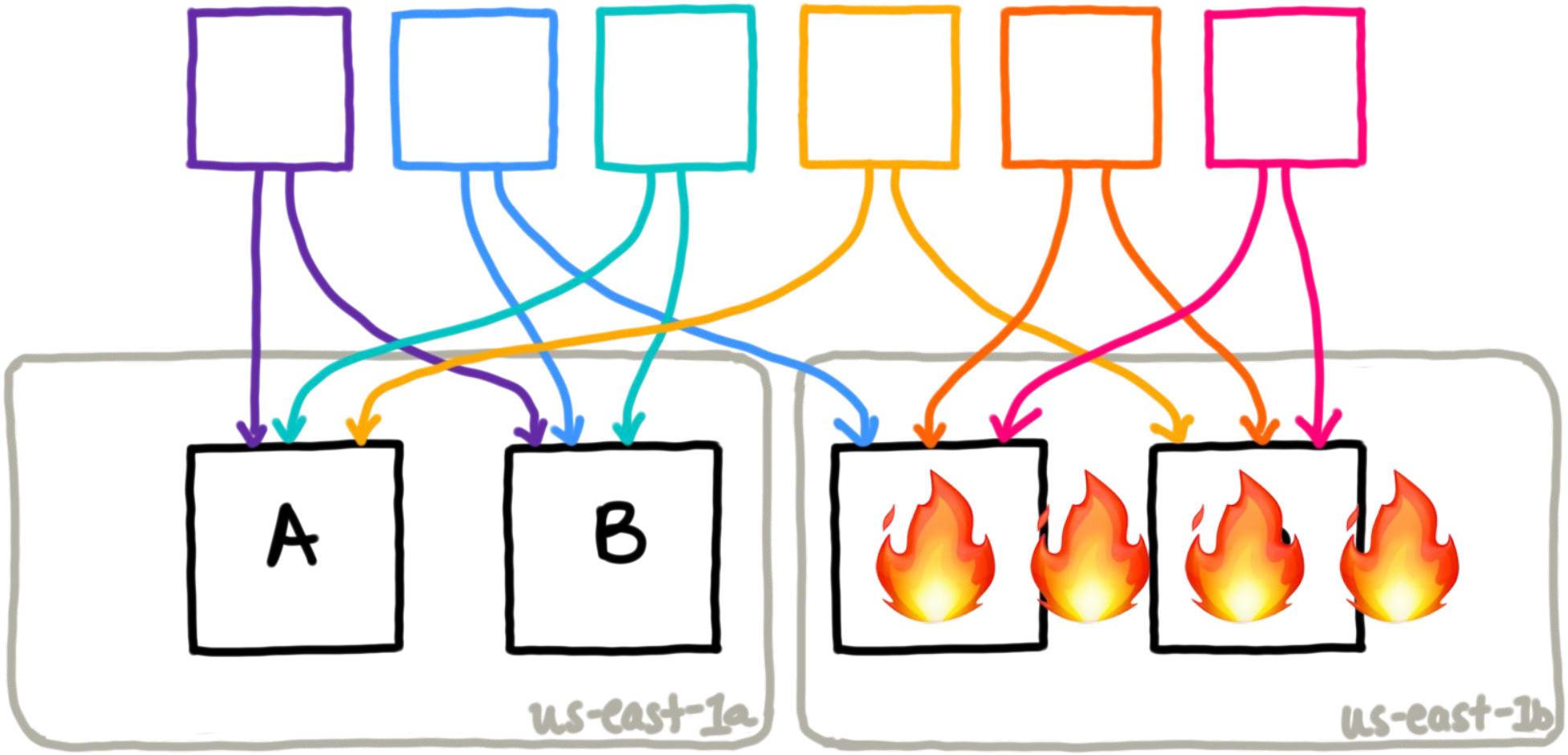
AZ Failure



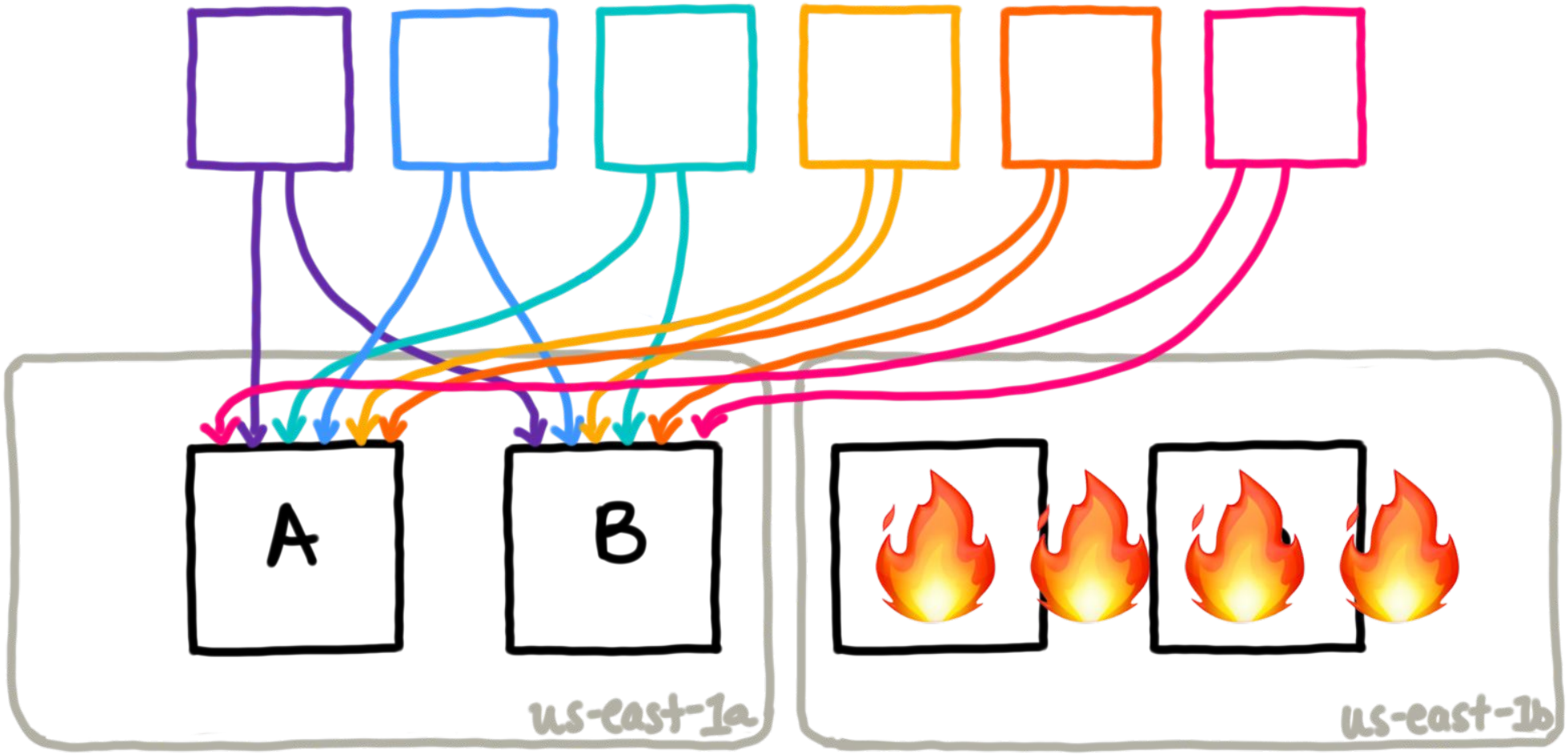
AZ Failure



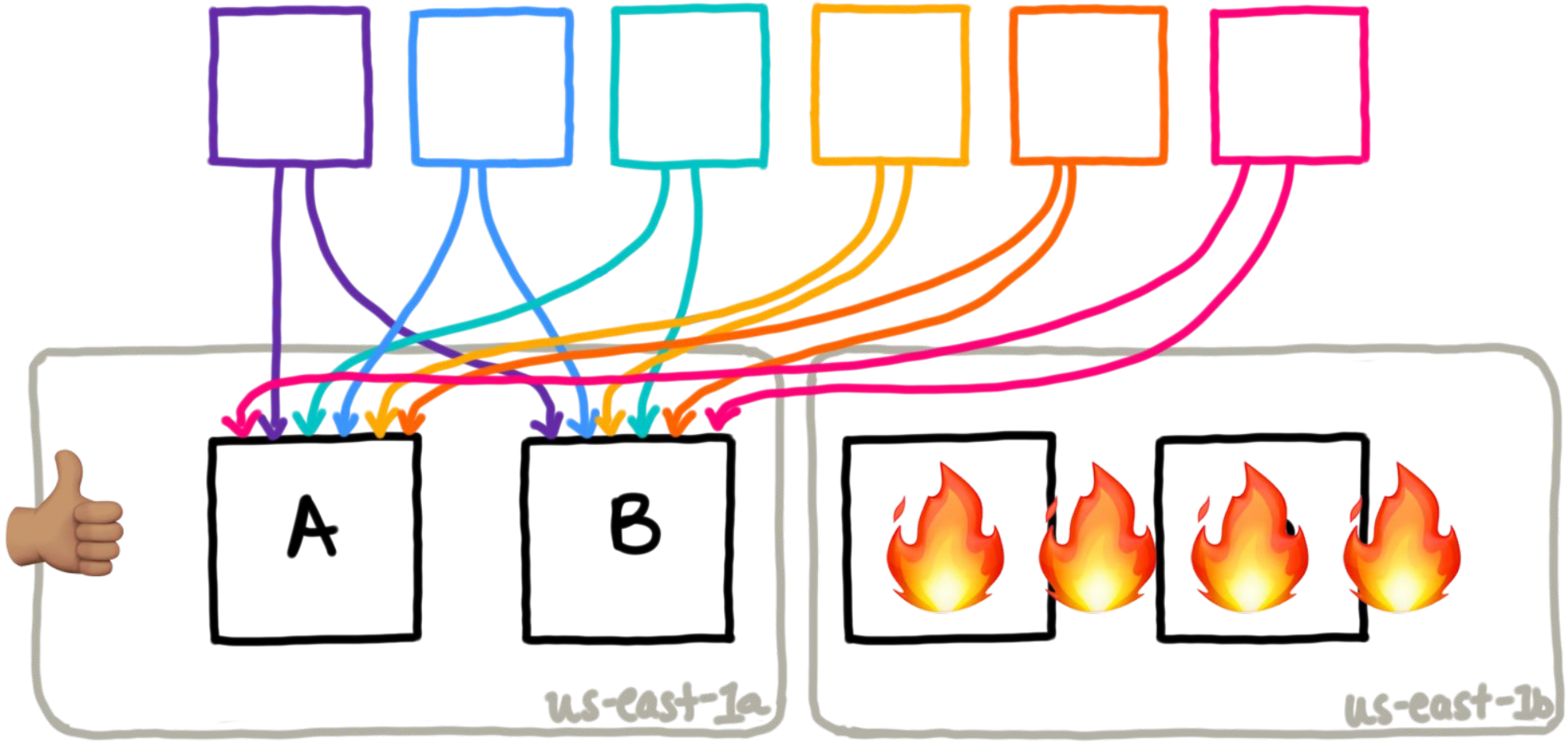
AZ Failure



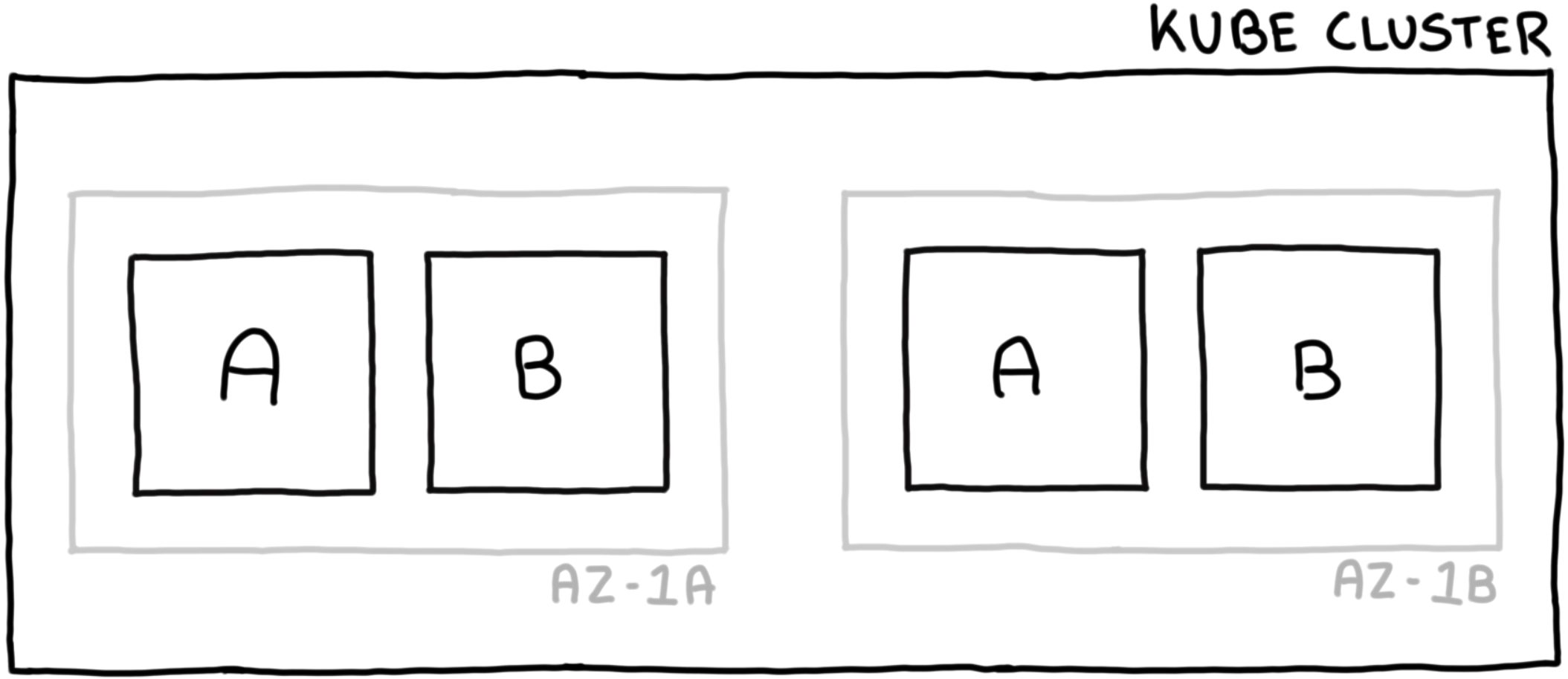
AZ Failure



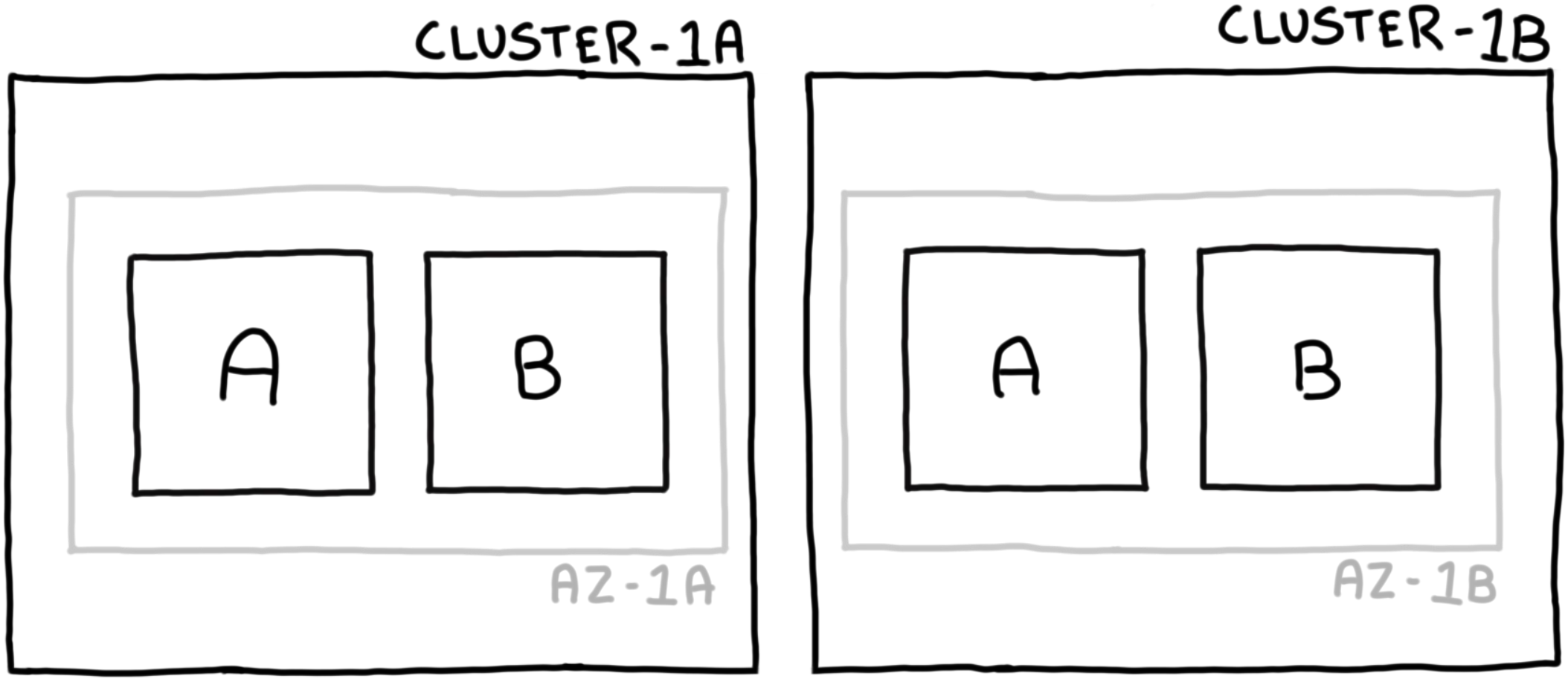
AZ Failure



Kube Cluster Failure



Kube Cluster Failure



Cloud Vendor Failure

https://status.cloud.google.com/incident/zall/20005

Lesson Learned:

Failure domains are many; choose which to accept

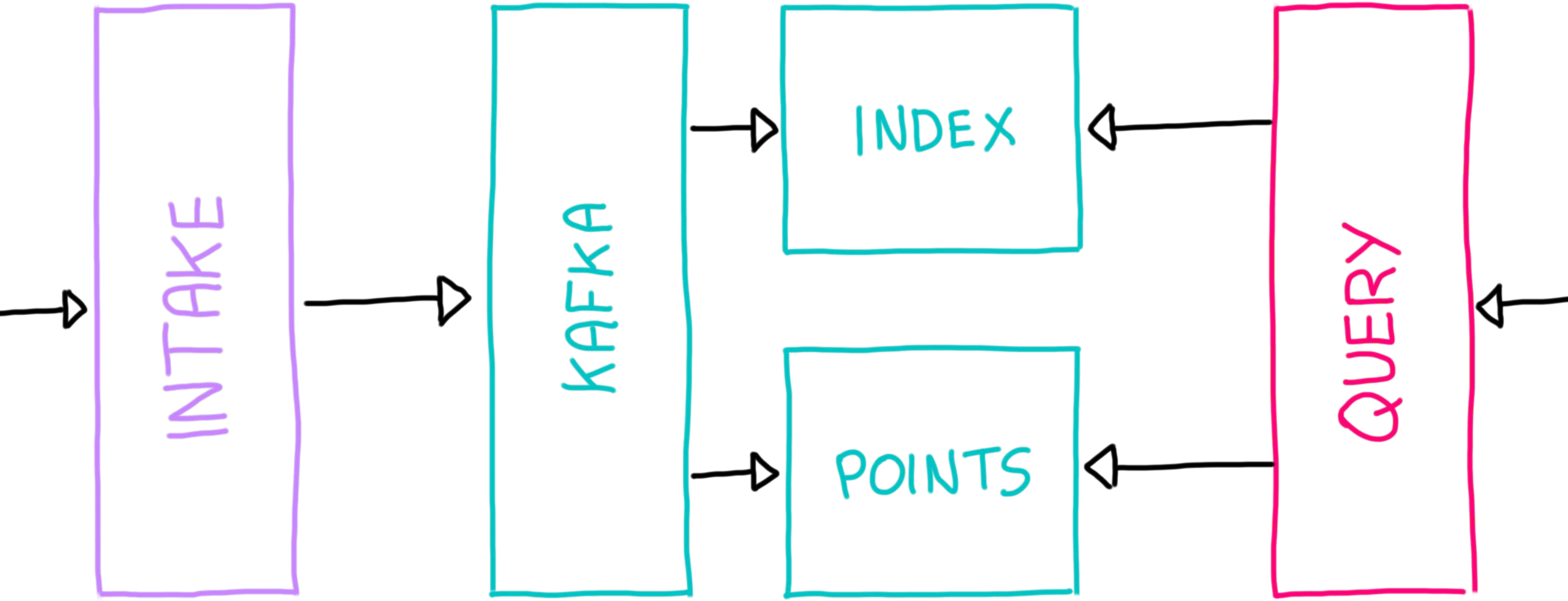
DATADOG

Problem:

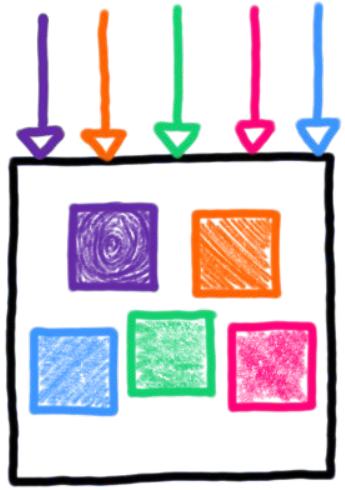
Too Many Customers 🥲

DATADOG

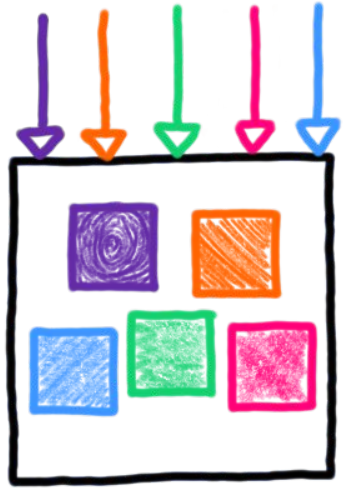
Remember This



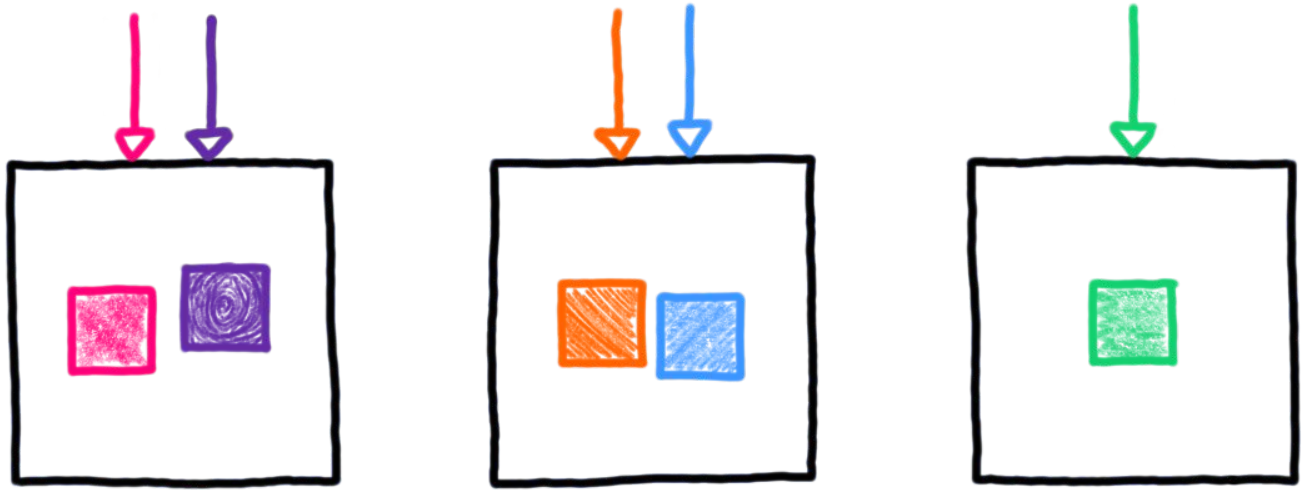
Partitioning (not the Kafka kind)



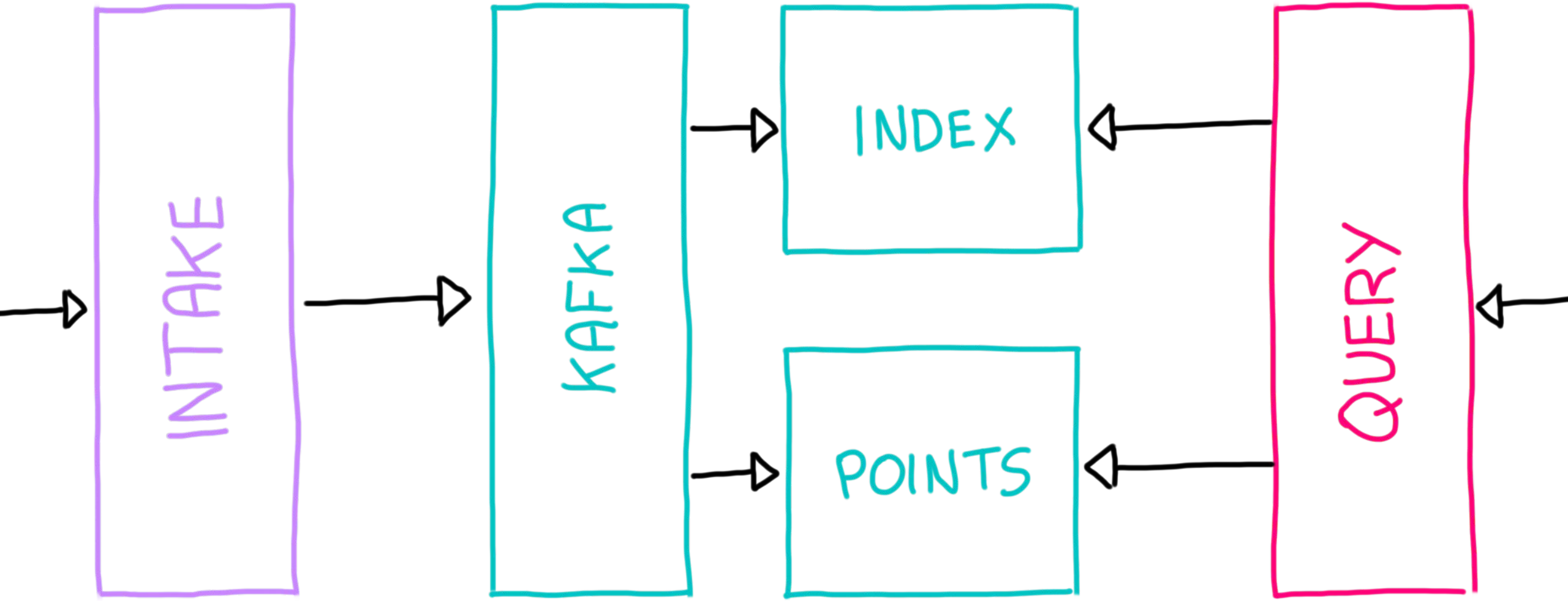
Partitioning (not the Kafka kind)



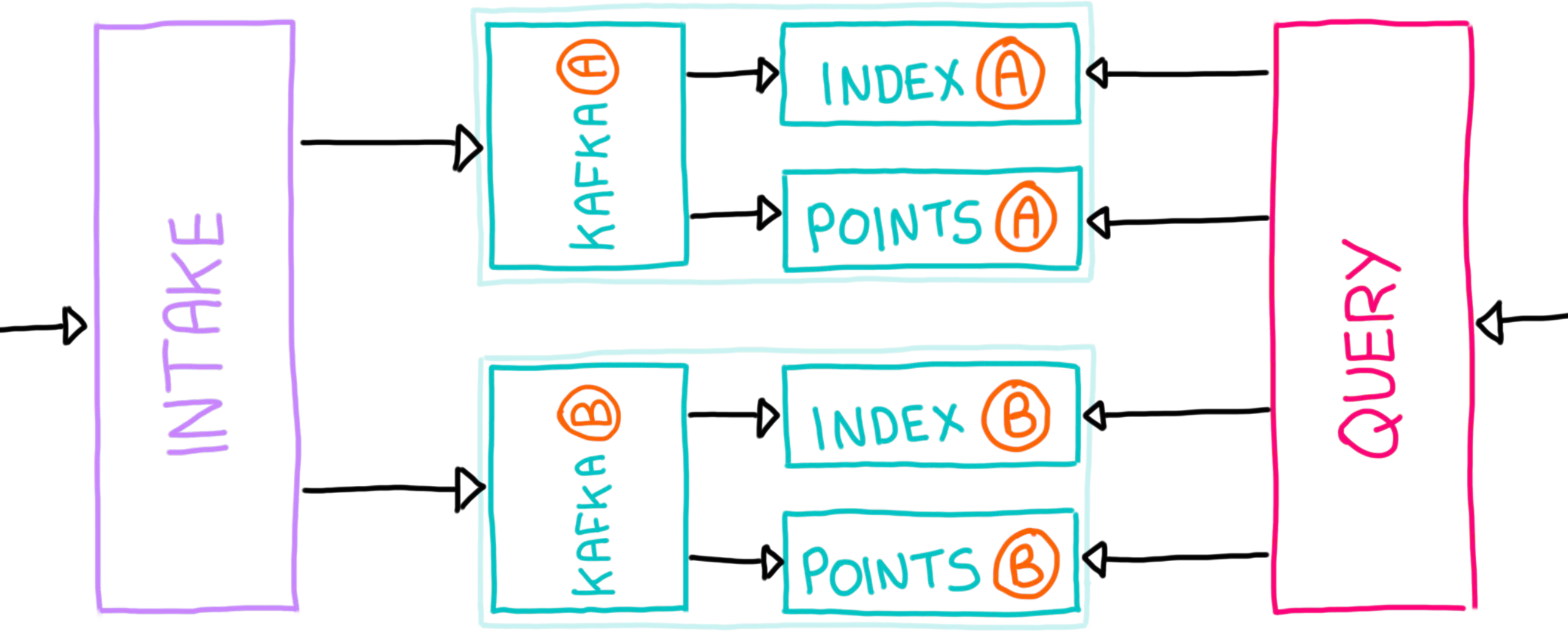
Partitioning (not the Kafka kind)



Partitioning: Before



Partitioning: After



Partitioning

$$p = mod(n, hash(customerID))$$

Partitioning

$$t = ddog\text{-}mega\text{-}topic$$

$$p = mod(n, hash(customerID))$$

Partitioning

$$t = mod(nt, hash(customerID))$$

$$p = mod(np, hash(customerID))$$

Lesson Learned:

You might not need fancy sharding; maybe just multiple levels of same

DATADOG

Lesson Learned:

One → Two is the hardest;
Start with two if you can

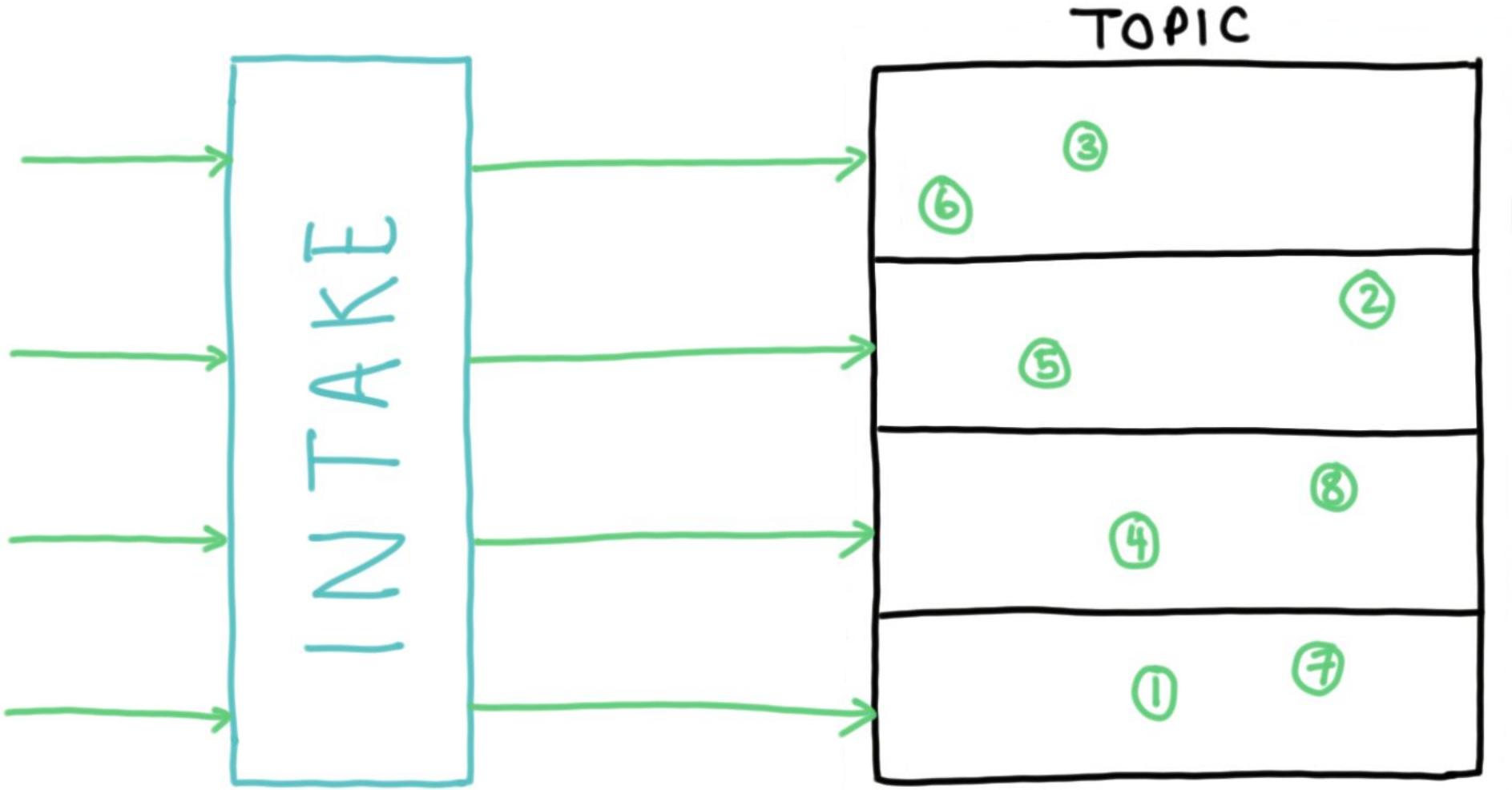
DATADOG

Problem:

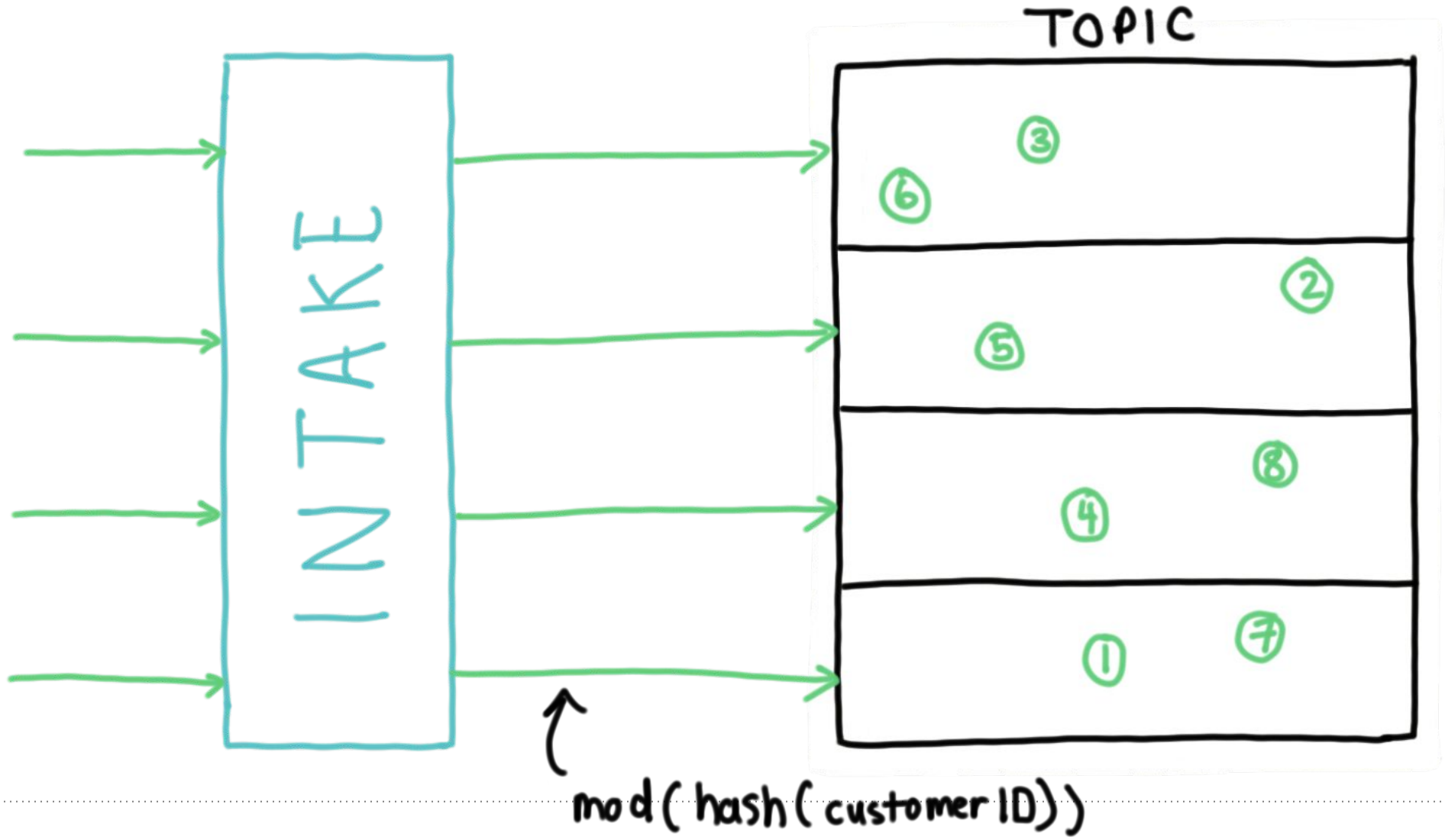
Big Customers

DATADOG

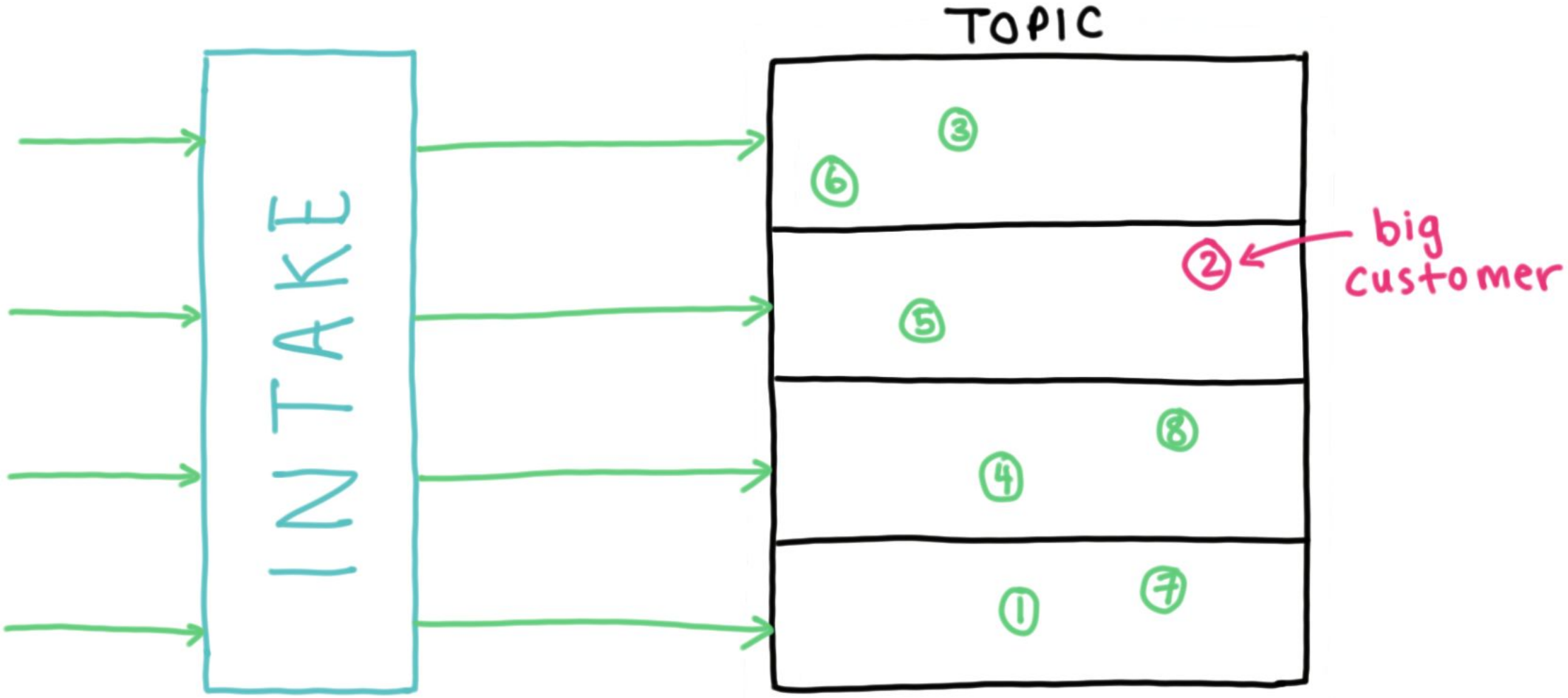
Balanced Topic



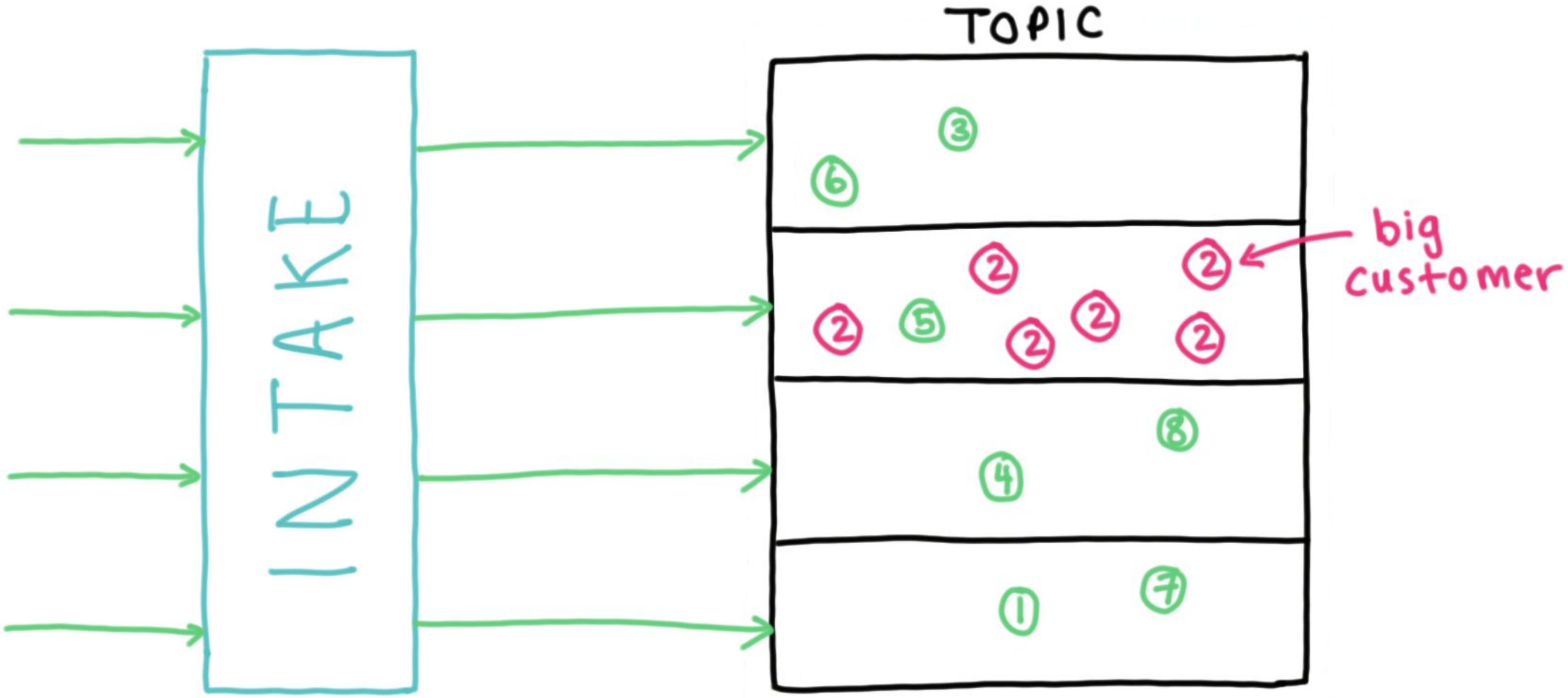
Balanced Topic



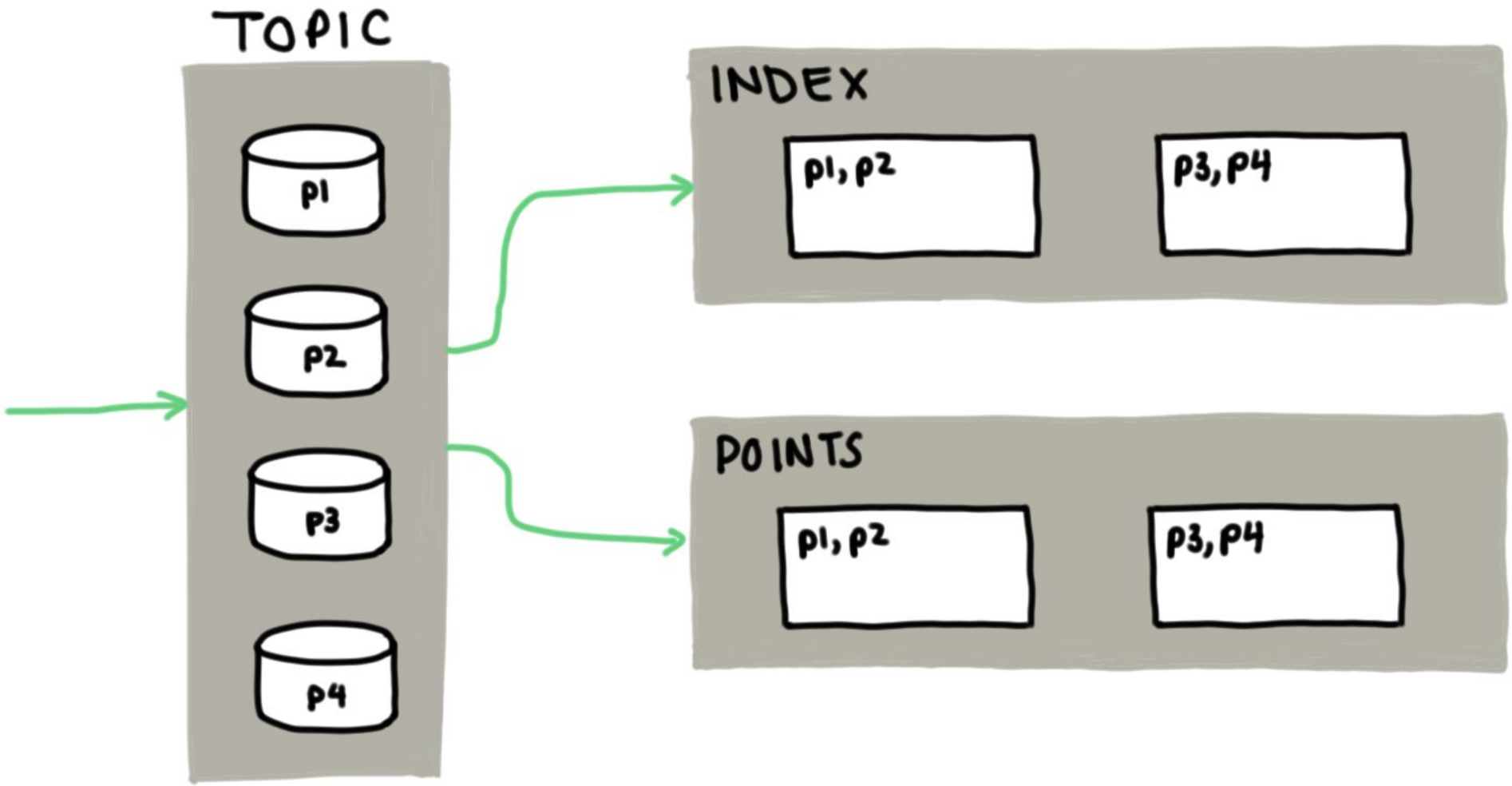
Balanced Topic



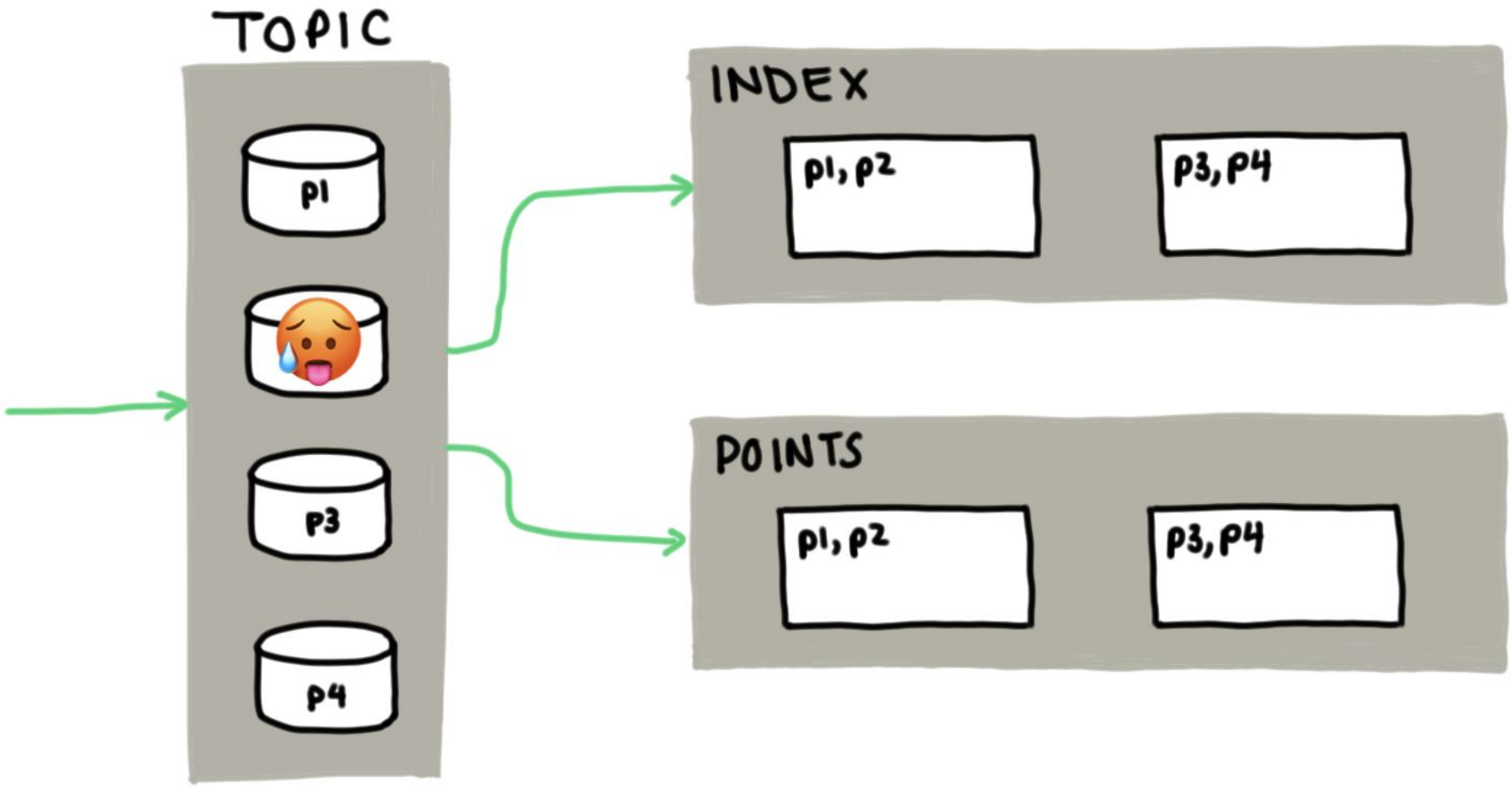
Hot Topic 🧛



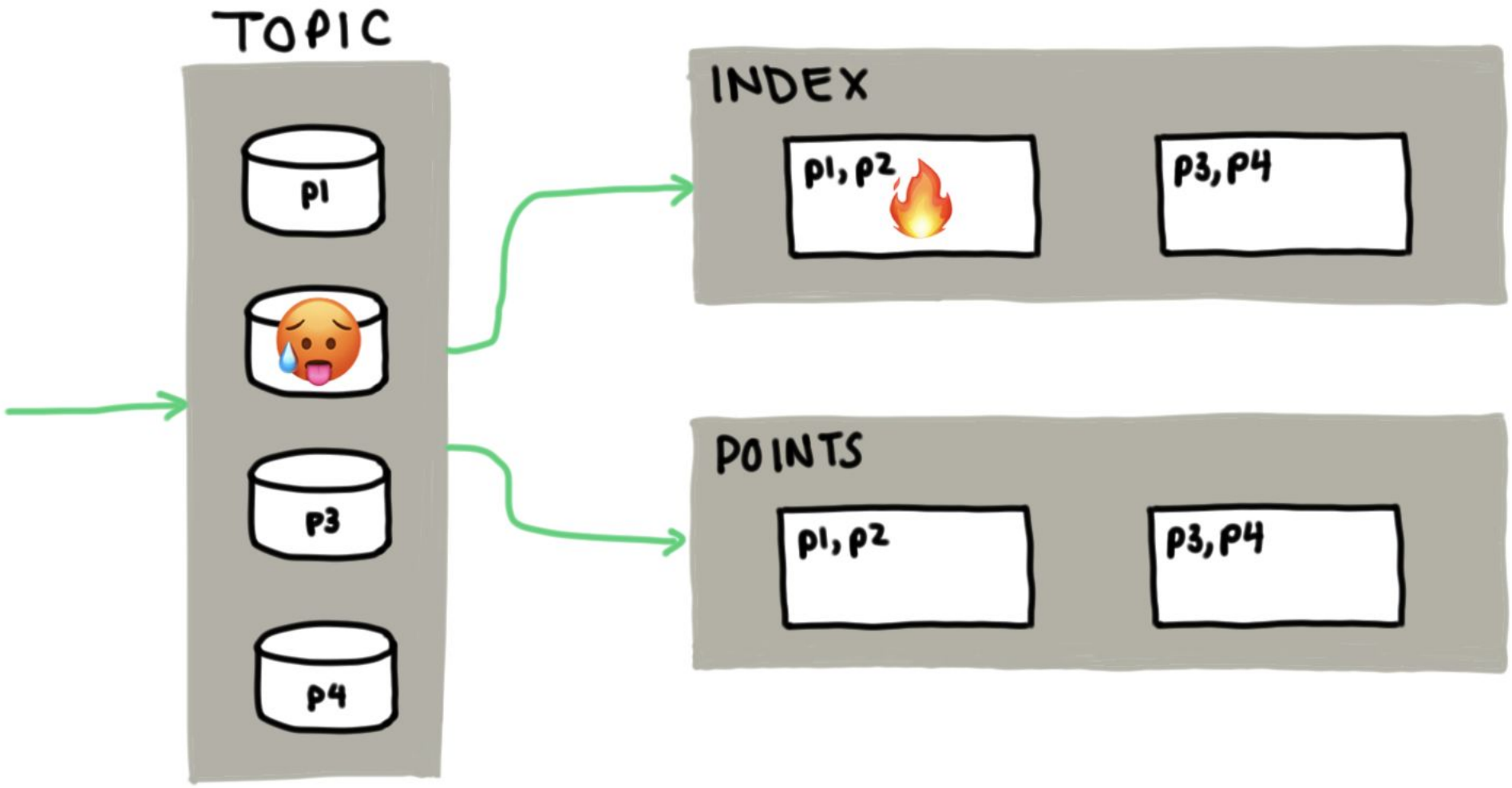
Consumer Shards



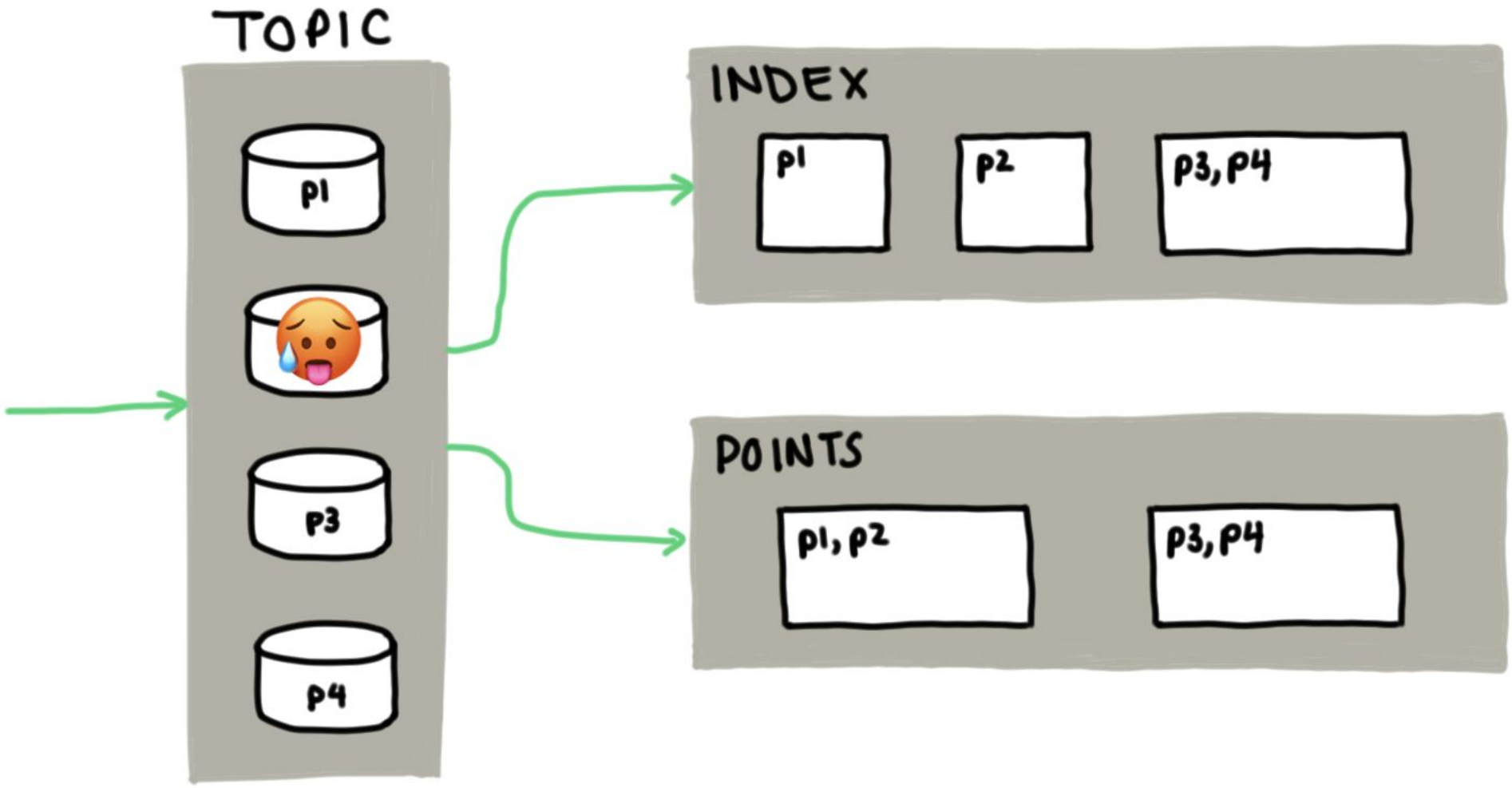
Consumer Shards



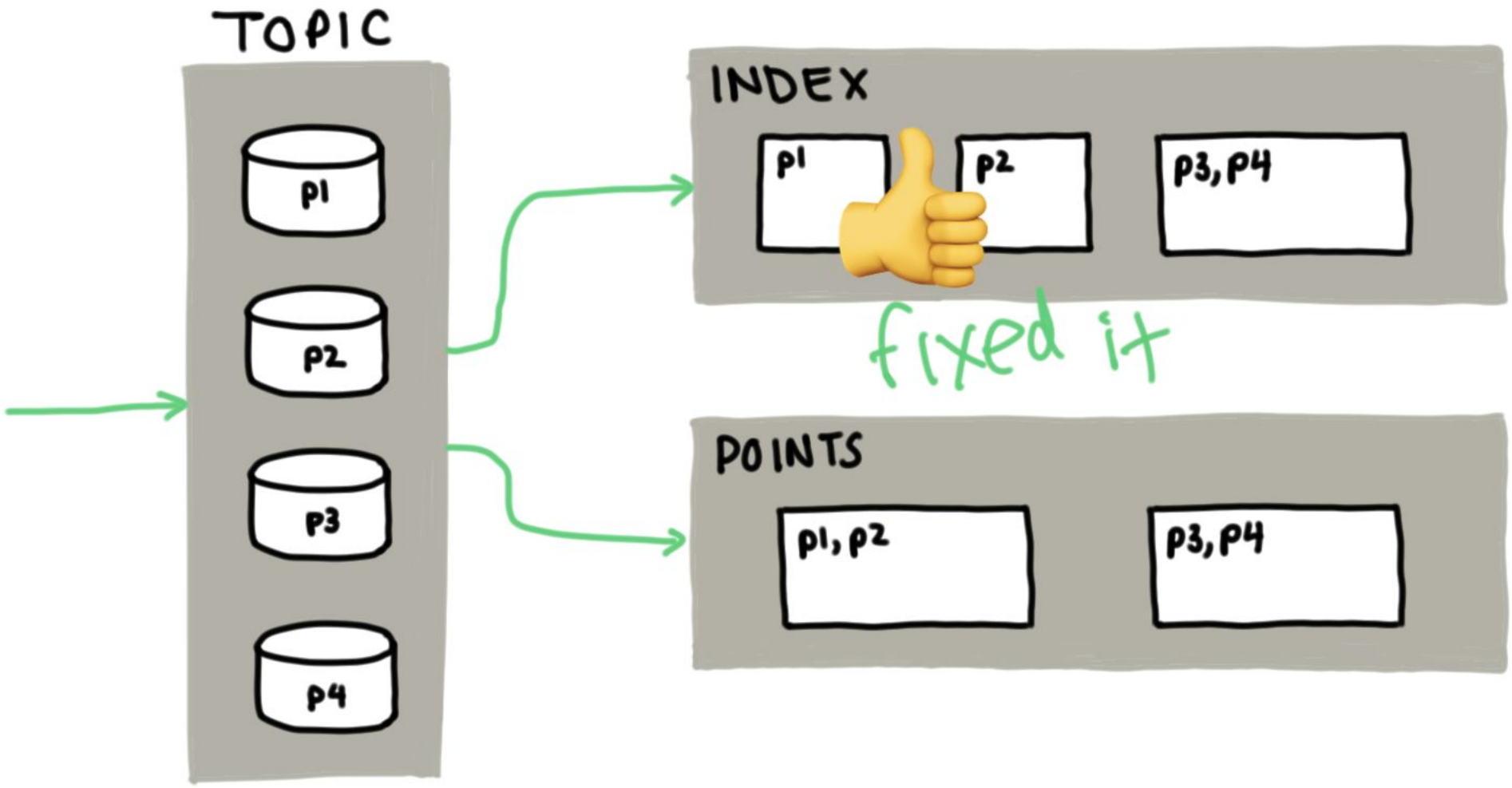
Consumer Shards



Consumer Shards



Consumer Shards

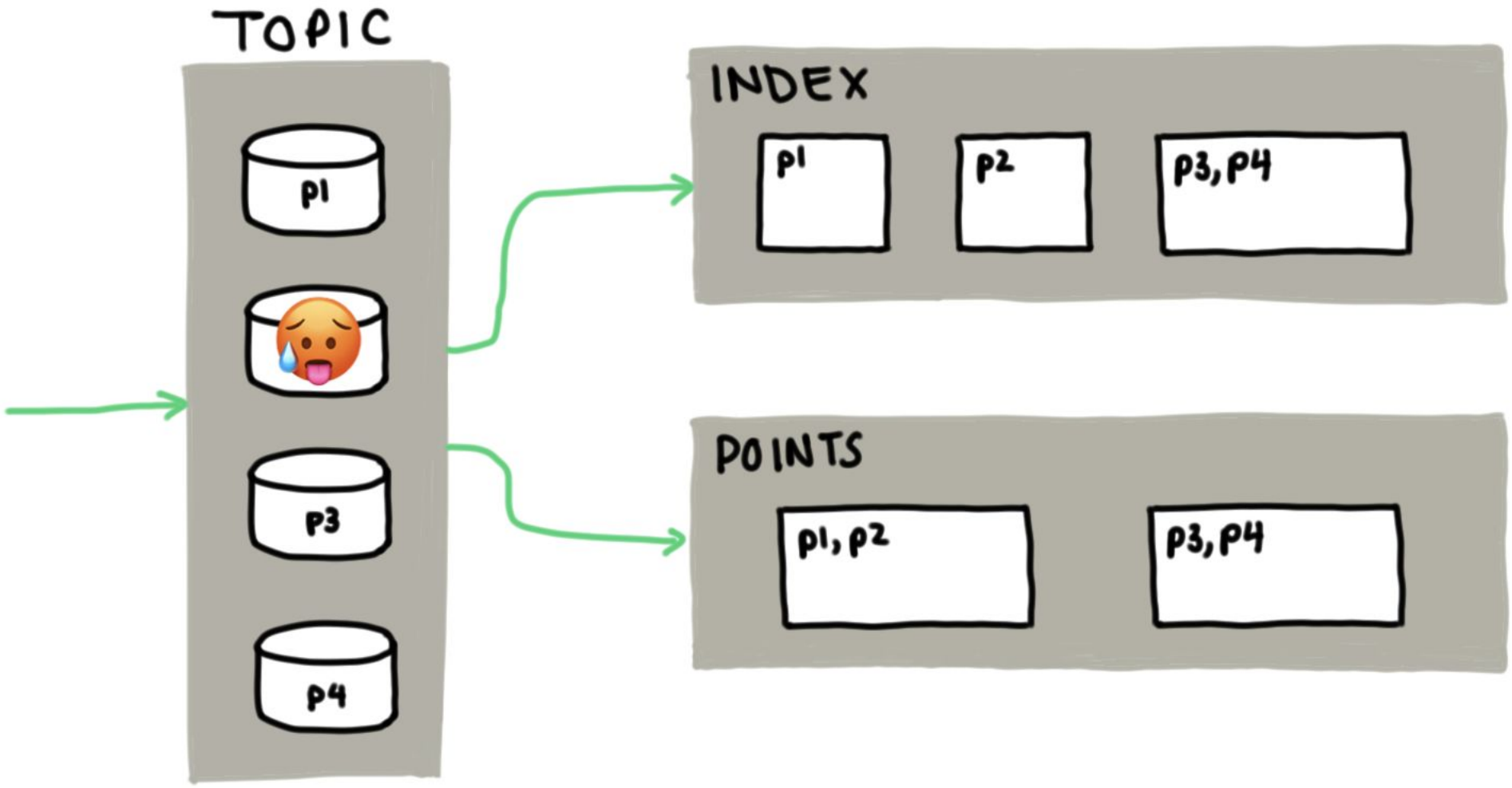


PROTIP:

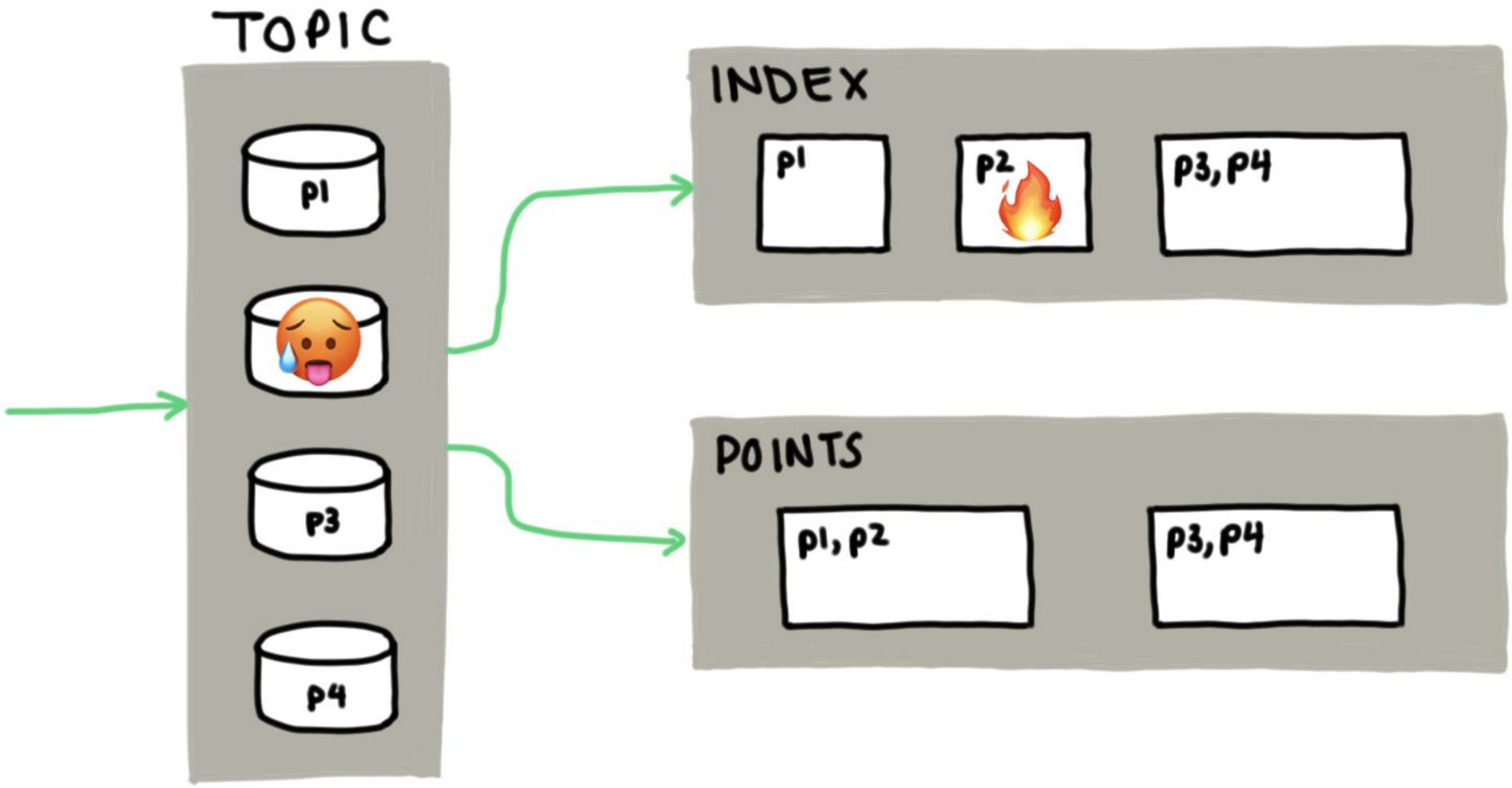
Many partitions, one shard

DATADOG

Consumer Shards



Consumer Shards



Big Customers

$$p = mod(n, hash(customerID))$$

Big Customers

$$p = mod(n, hash(metricID))$$

Big Customers

Big Customers

Lesson Learned:

Coarse sharding schemes don't work for outliers; and that's okay.

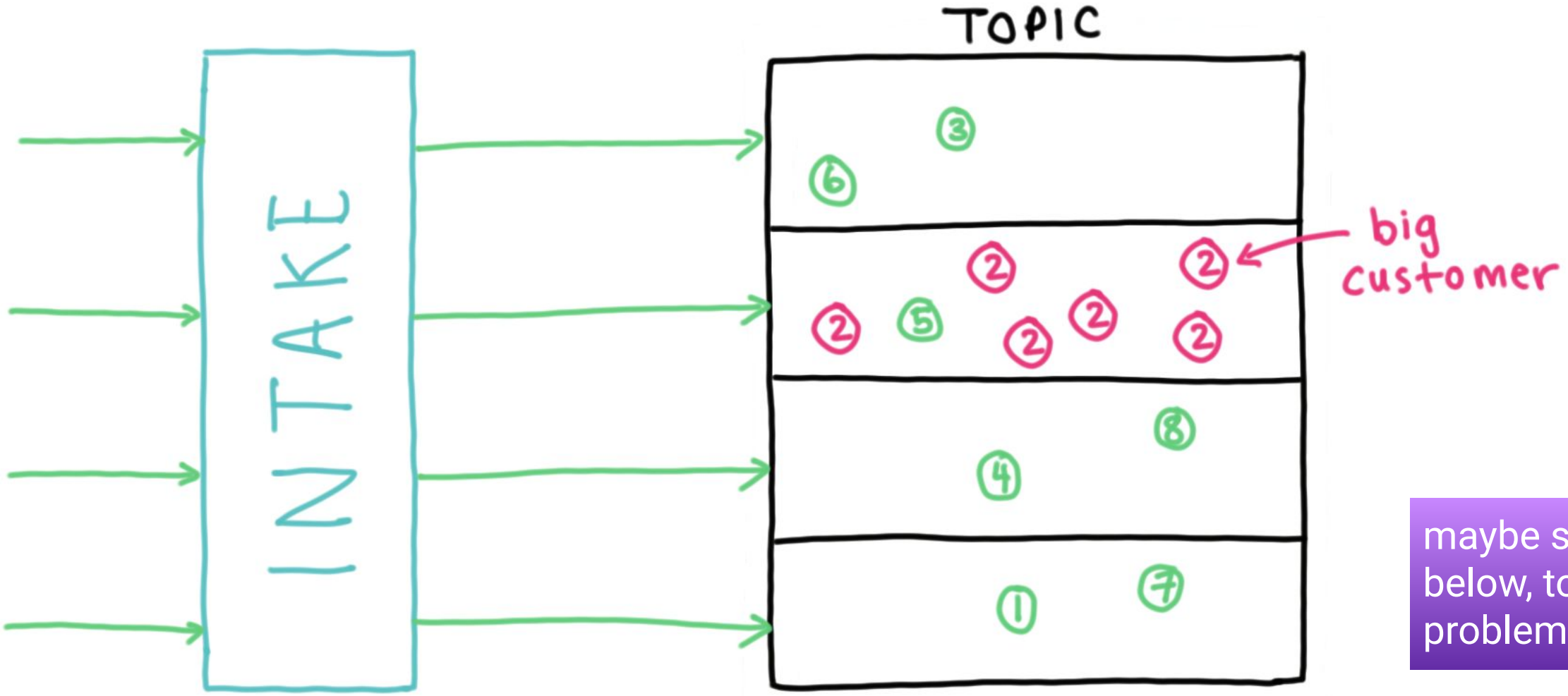
DATADOG

Problem:

Big Metrics (in Big Customers)

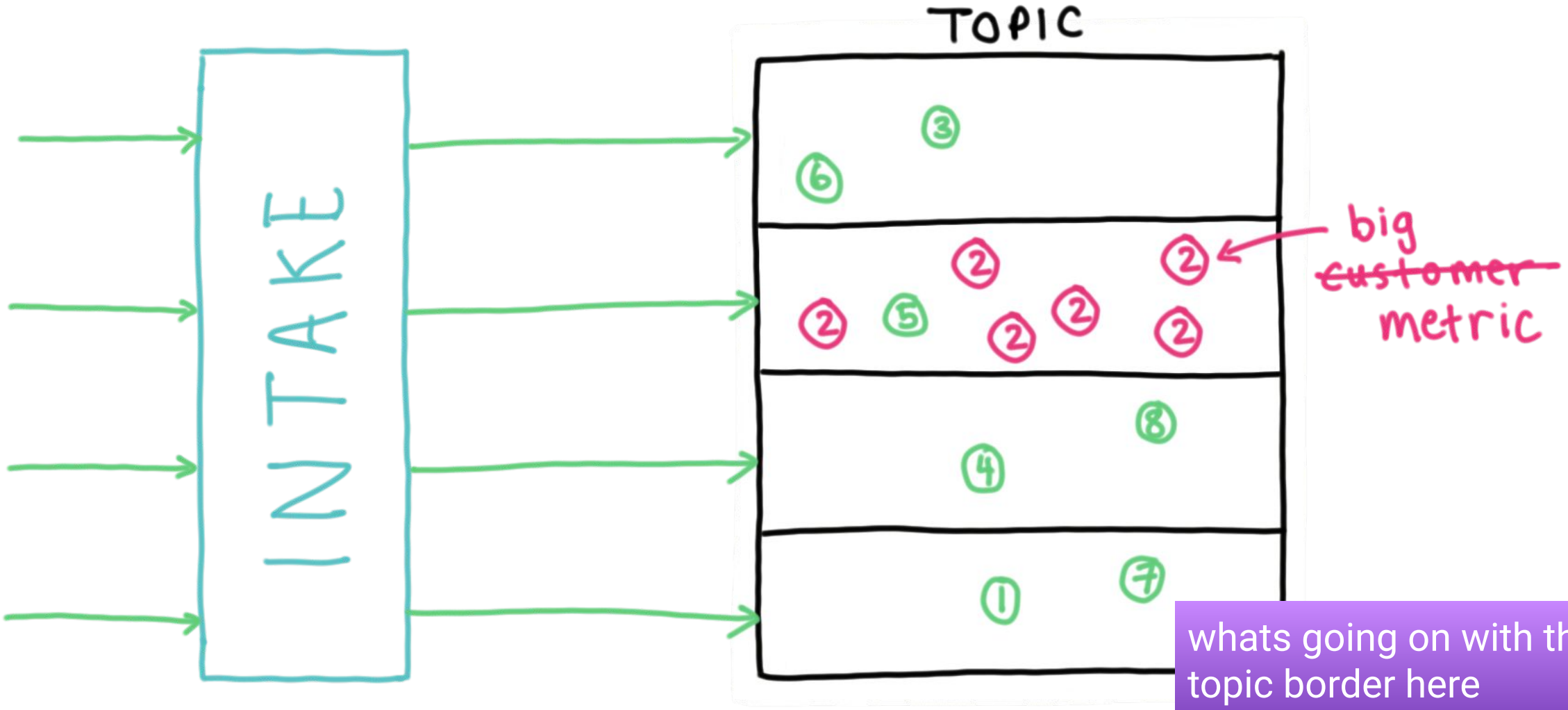
DATADOG

Big Metrics



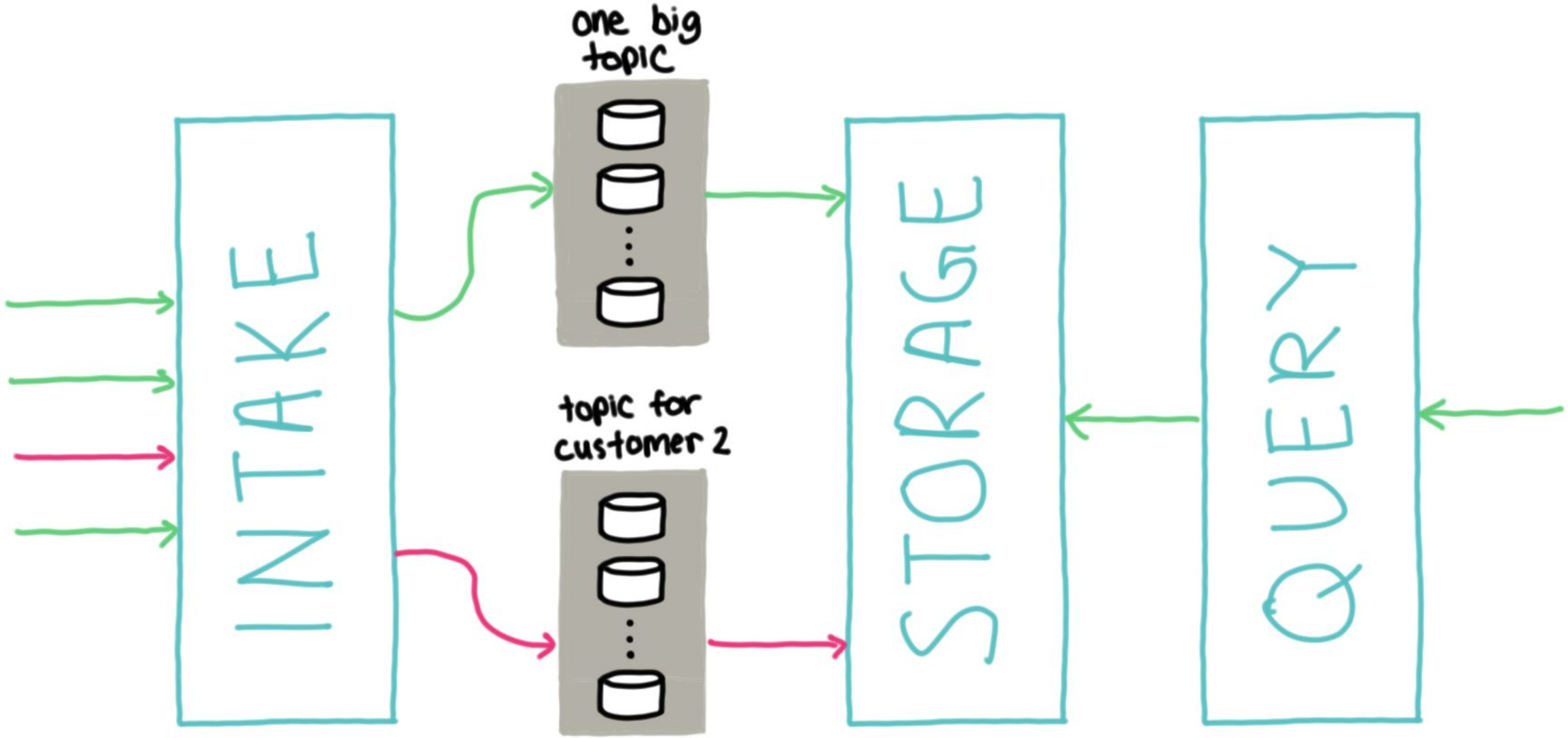
maybe strike o
below, to reinf
problem

Big Metrics



whats going on with the topic border here

Big Metrics

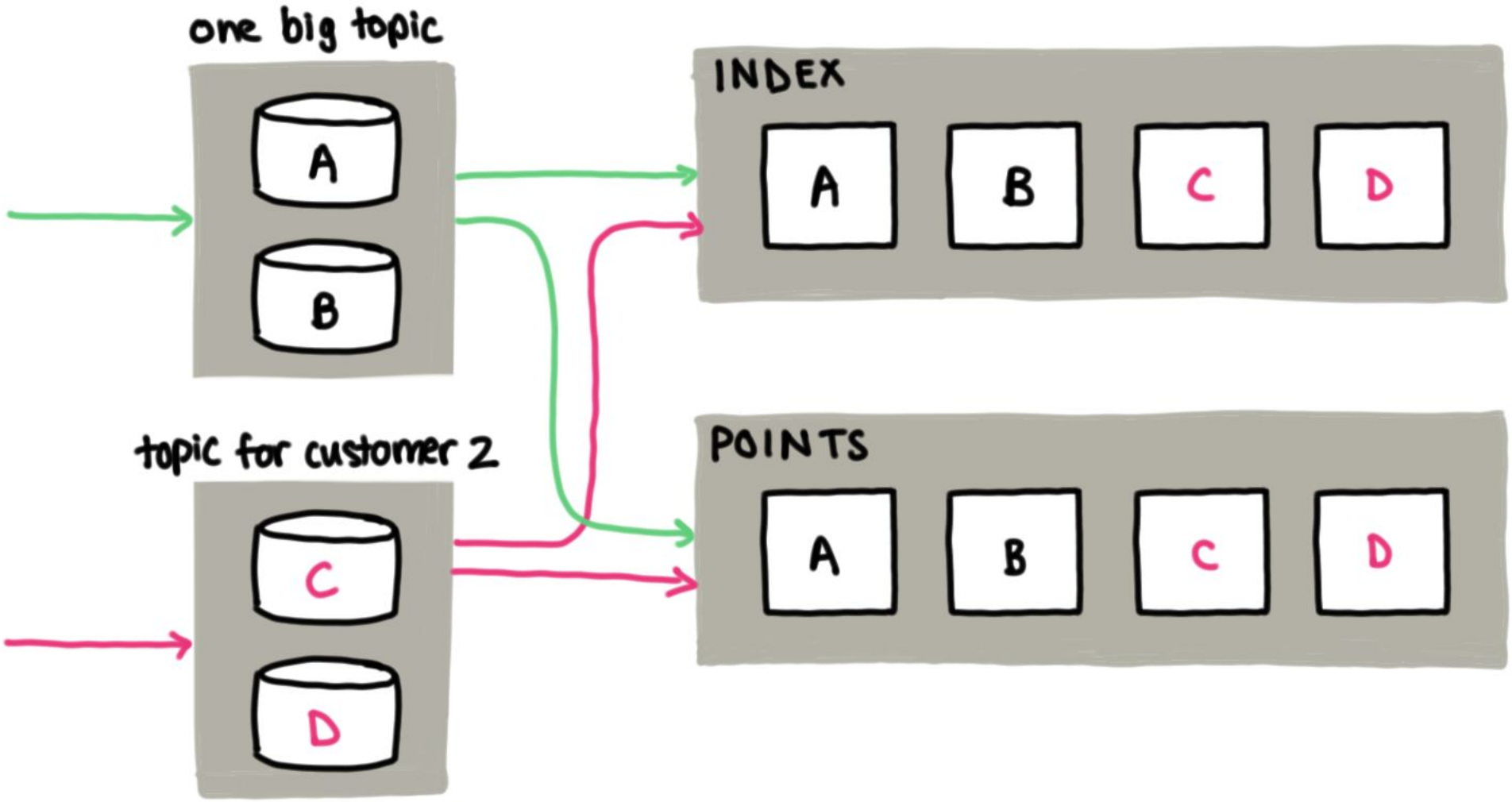


Big Metrics

$$p = mod(n, hash(customerID))$$

$$p = mod(n, hash(metricID, contextID))$$

Big Metrics


one big topic
A
B
topic for customer 2
C
D
INDEX
A
B
C
D
POINTS
A
B
C
D

Lesson Learned:

Simple sharding assumptions only work for so long

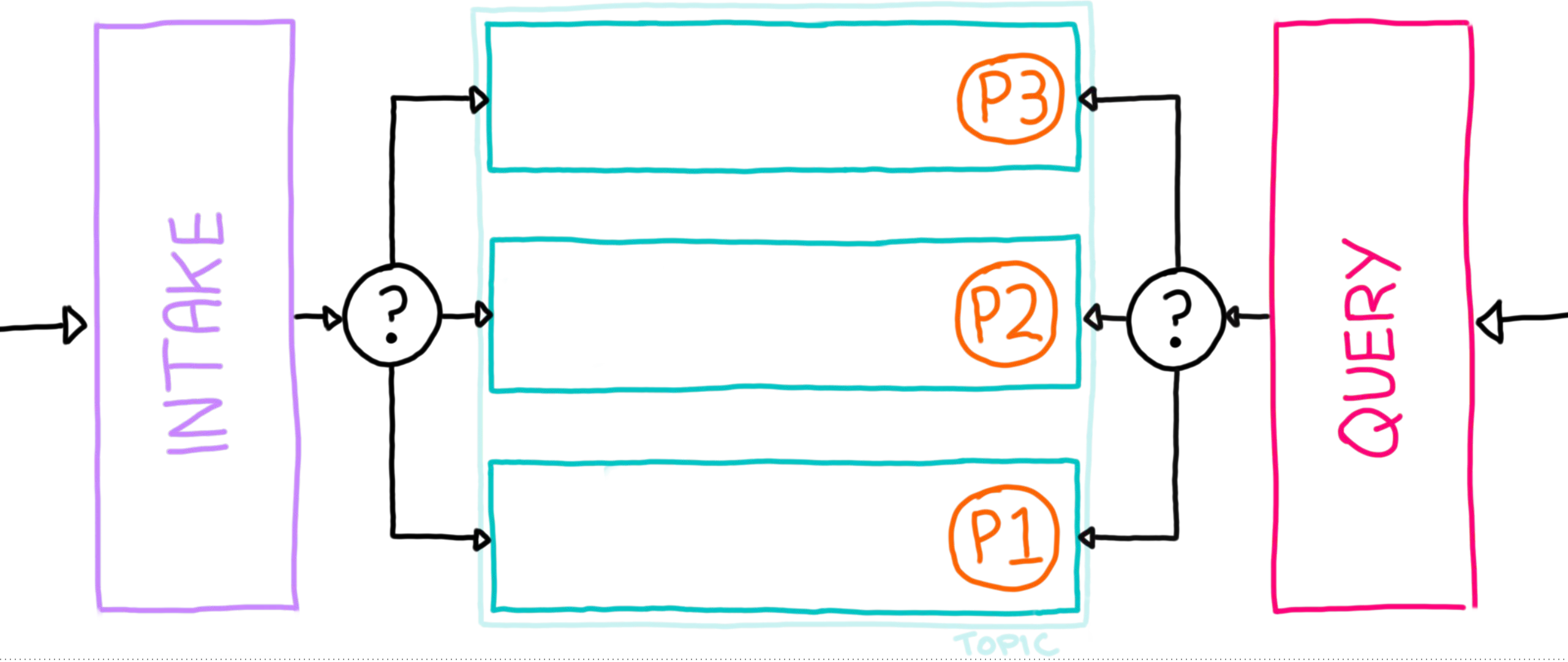
DATADOG

Problem:

Partition Imbalance

DATADOG

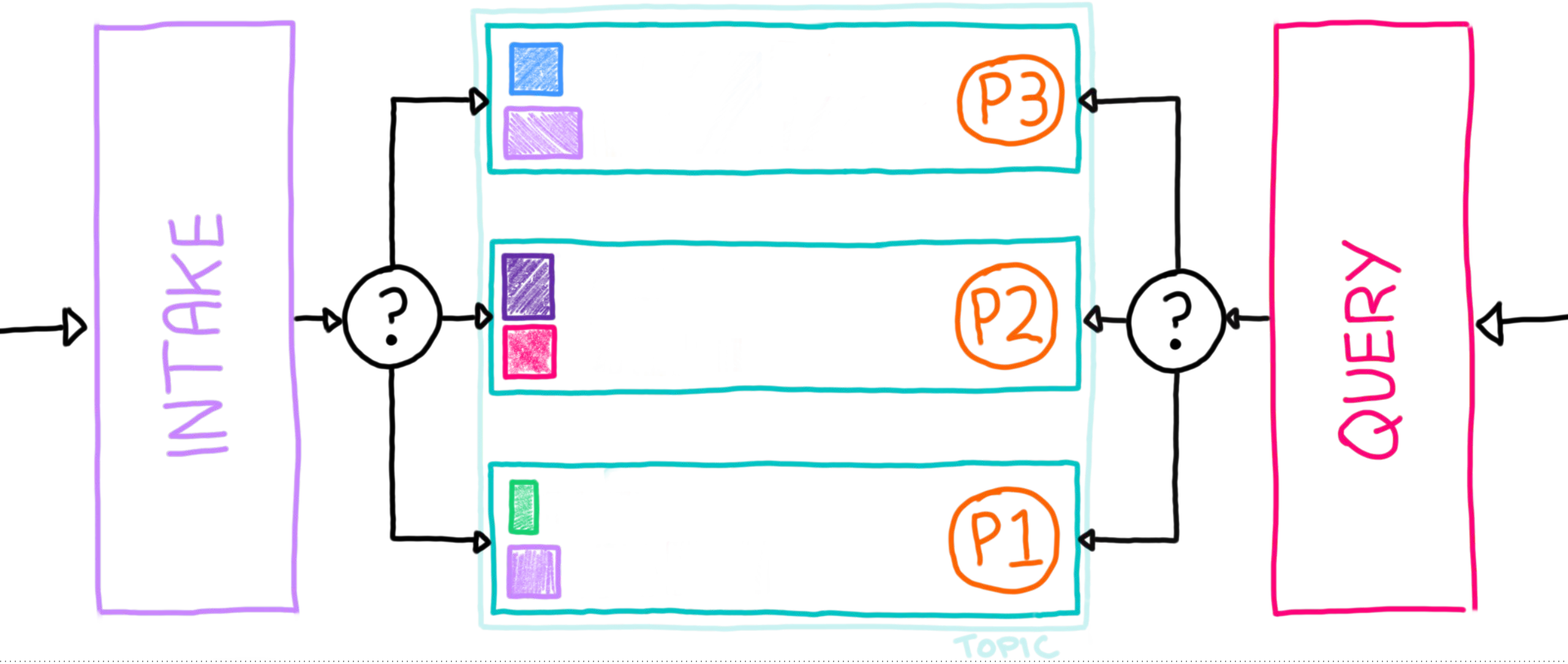
Partition Imbalance



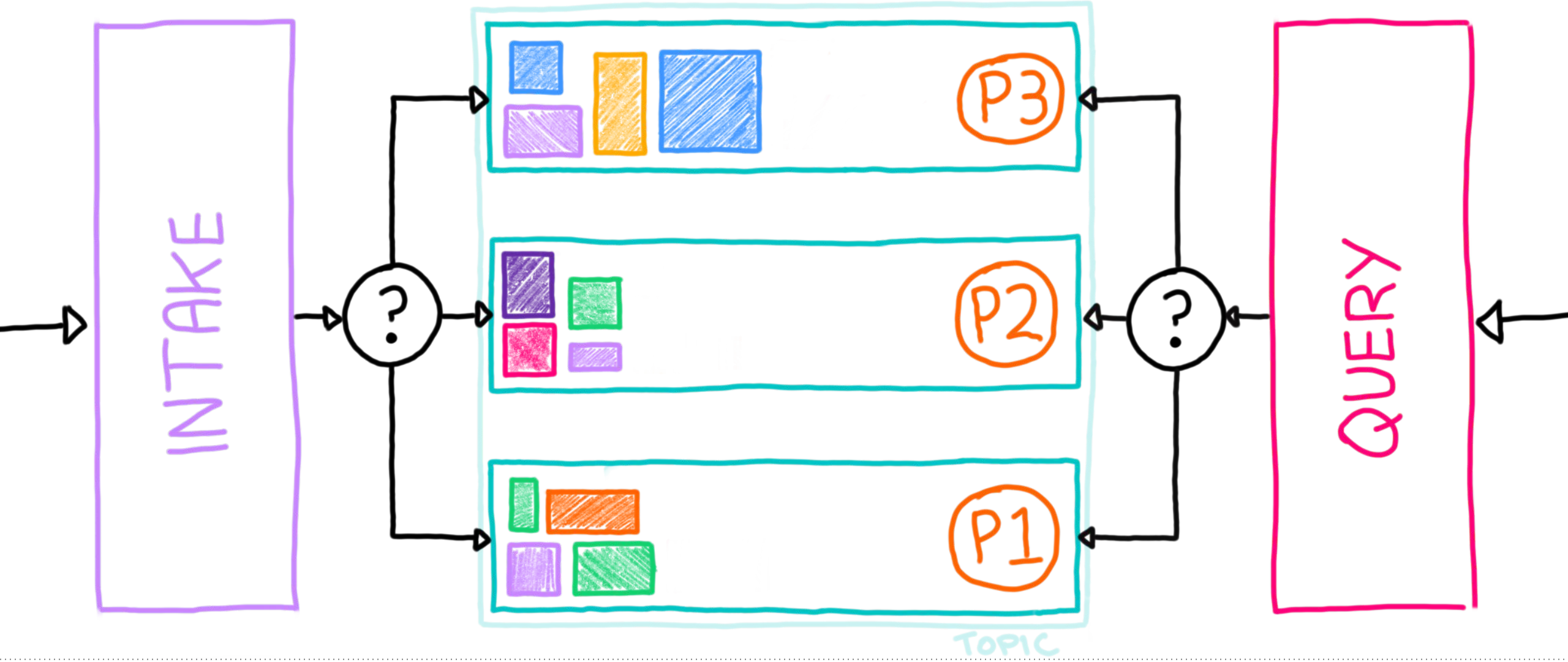
Partition Imbalance

$$p = mod(n, hash(customerID))$$

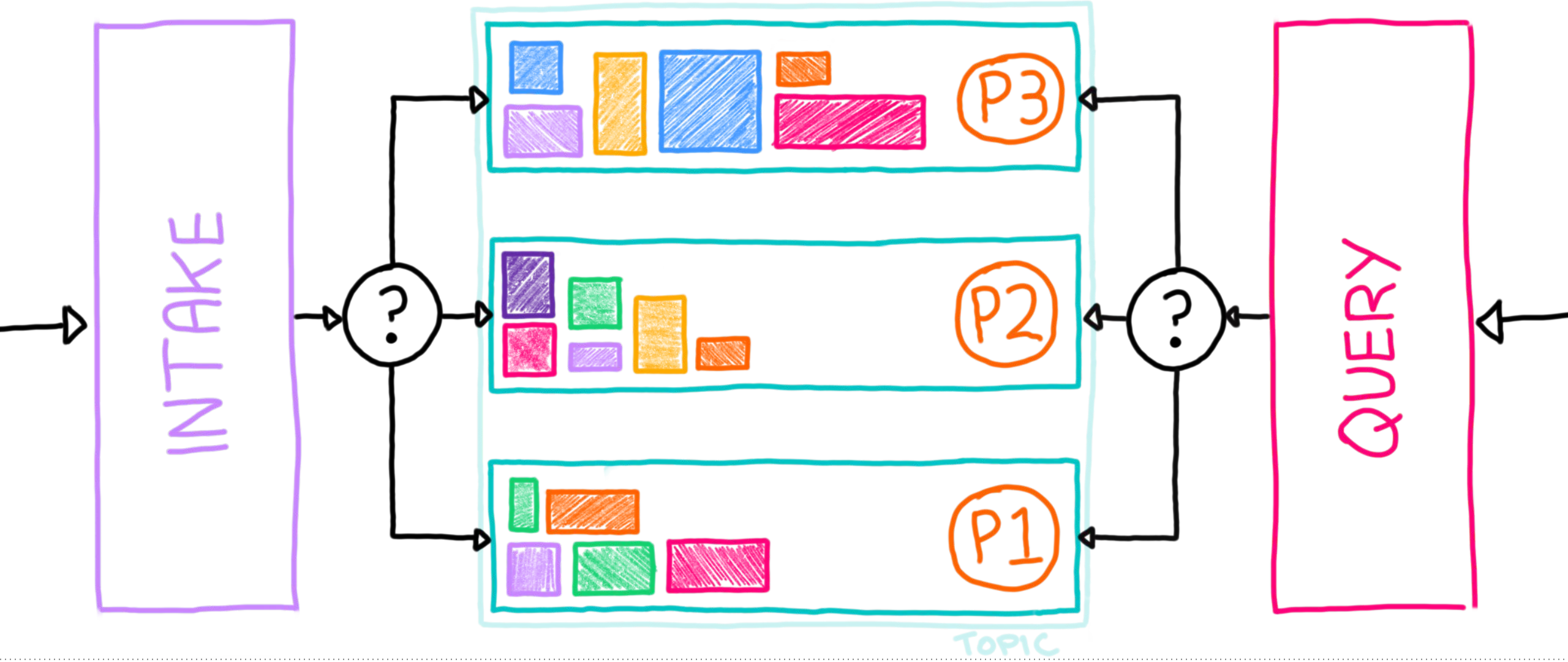
Partition Imbalance



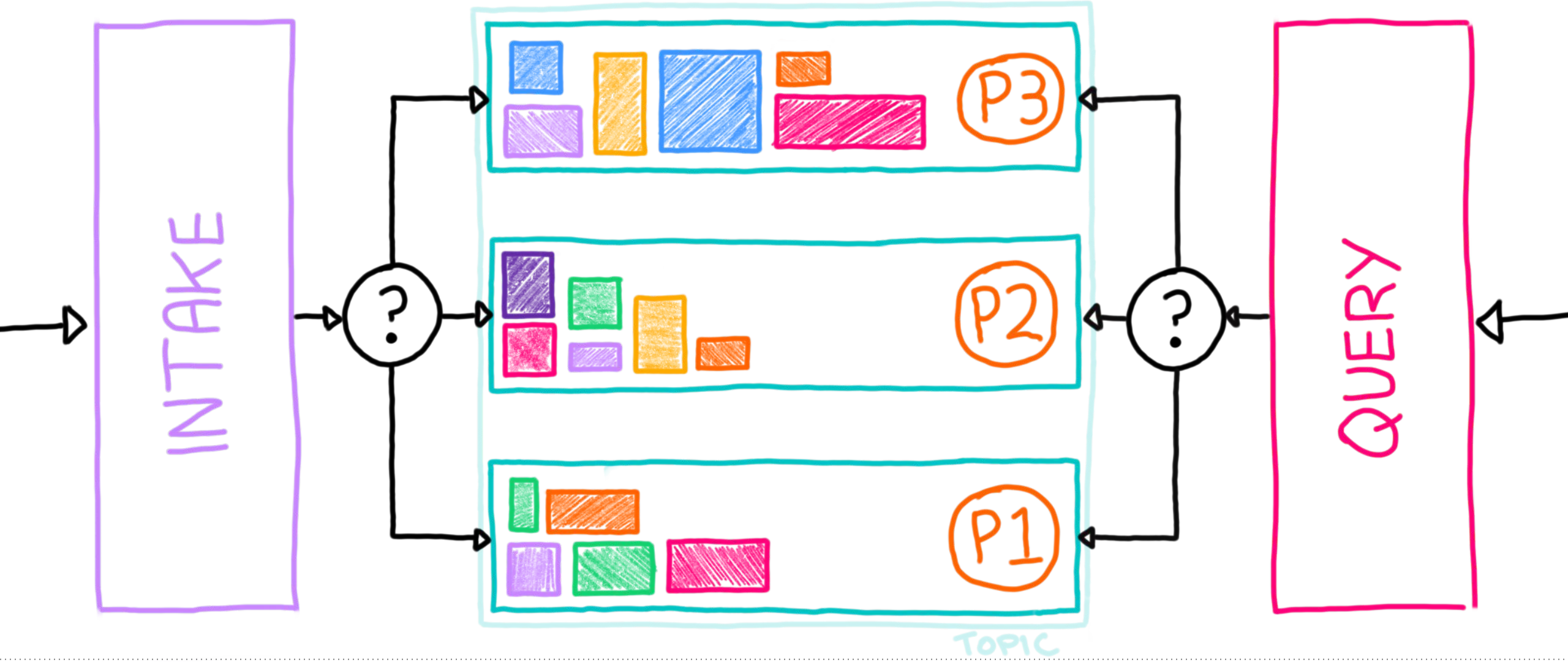
Partition Imbalance



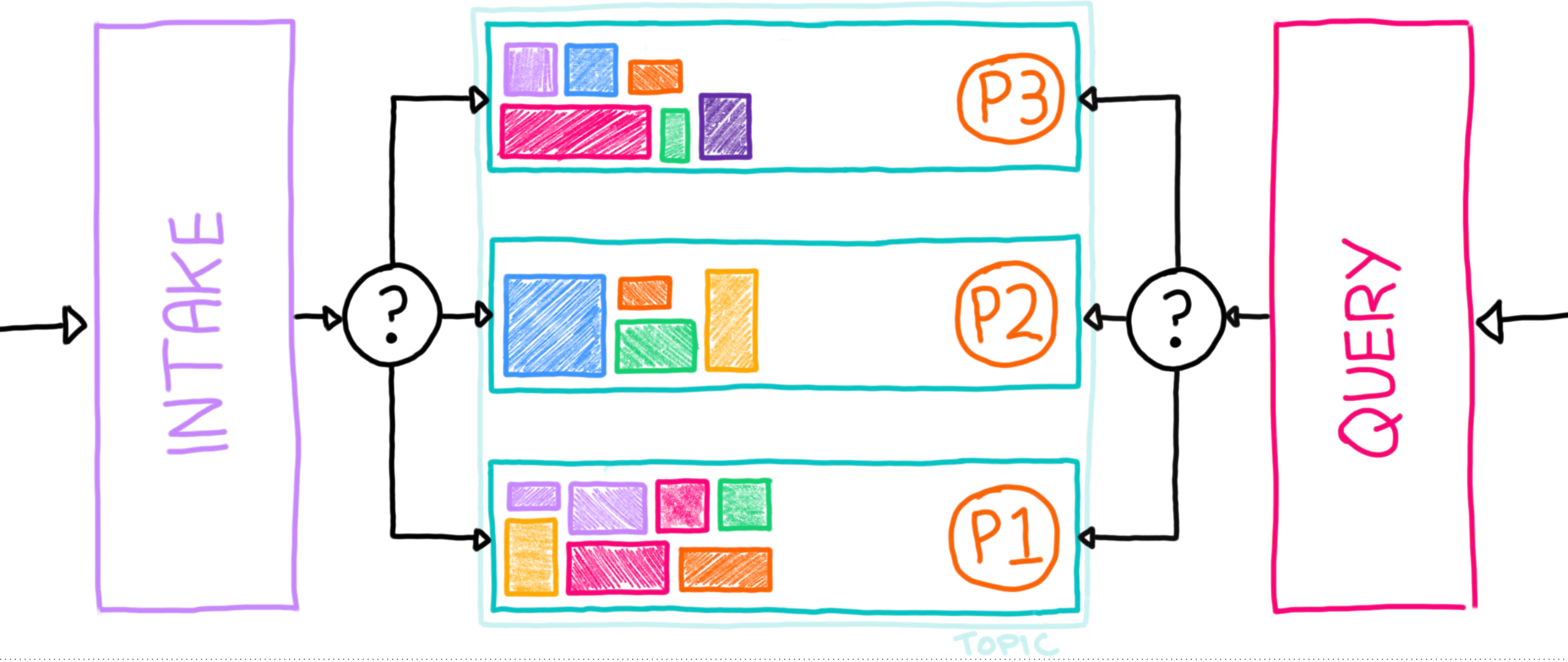
Partition Imbalance



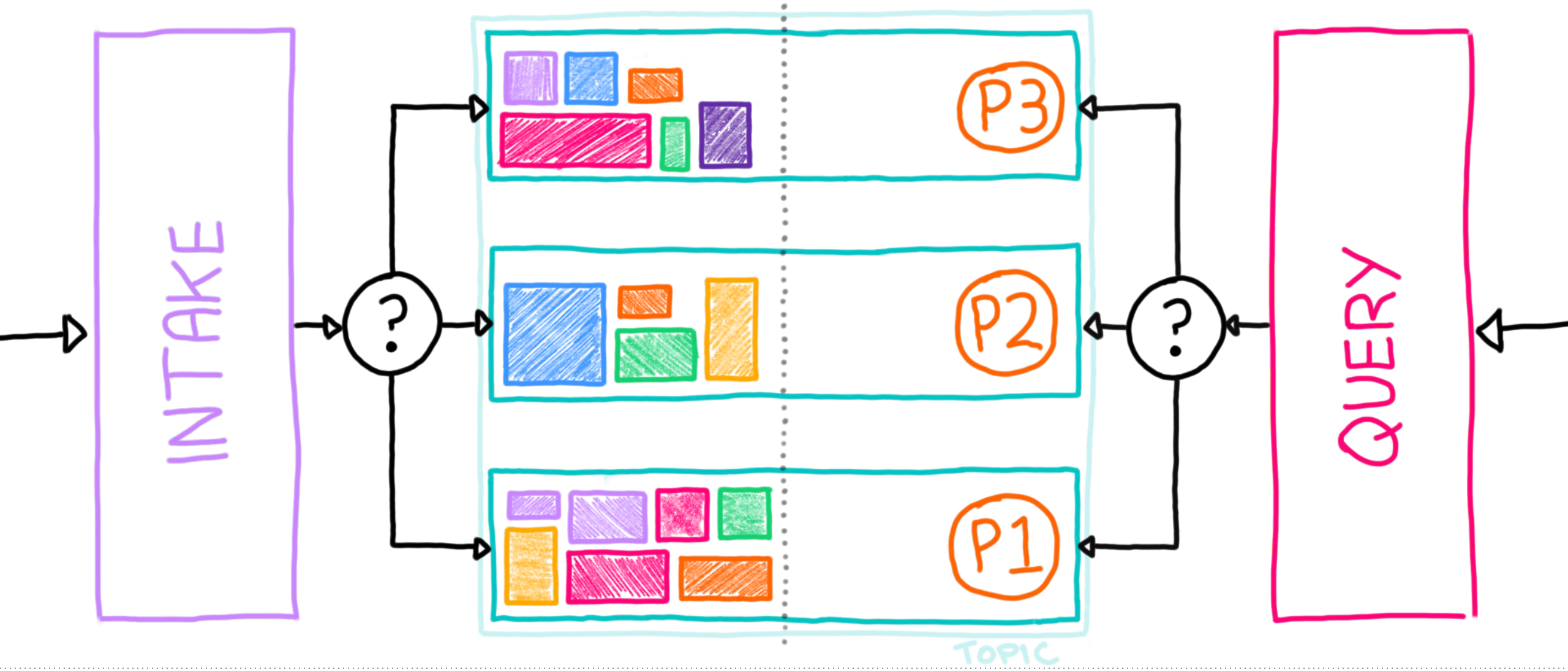
Partition Imbalance



Partition Imbalance



Partition Imbalance



Slicer: Auto-Sharding for Datacenter Applications

Atul Adya, Daniel Myers, Jon Howell, Jeremy Elson, Colin Meek, Vishesh Khemani,
Stefan Fulger, Pan Gu, Lakshminath Bhuvanagiri, Jason Hunter, Roberto Peon, Larry Kai,
Alexander Shraer, Arif Merchant, and Kfir Lev-Ari[†]

Google [†]*Technion - Israel*

Abstract

Sharding is a fundamental building block of large-scale applications, but most have their own custom, ad-hoc implementations. Our goal is to make sharding as easily reusable as a filesystem or lock manager. Slicer is Google's general purpose sharding service. It monitors signals such as load hotspots and server health to dynamically shard work over a set of servers. Its goals are to maintain high availability and reduce load imbalance while minimizing churn from moved work.

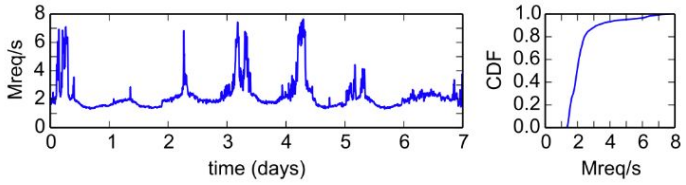
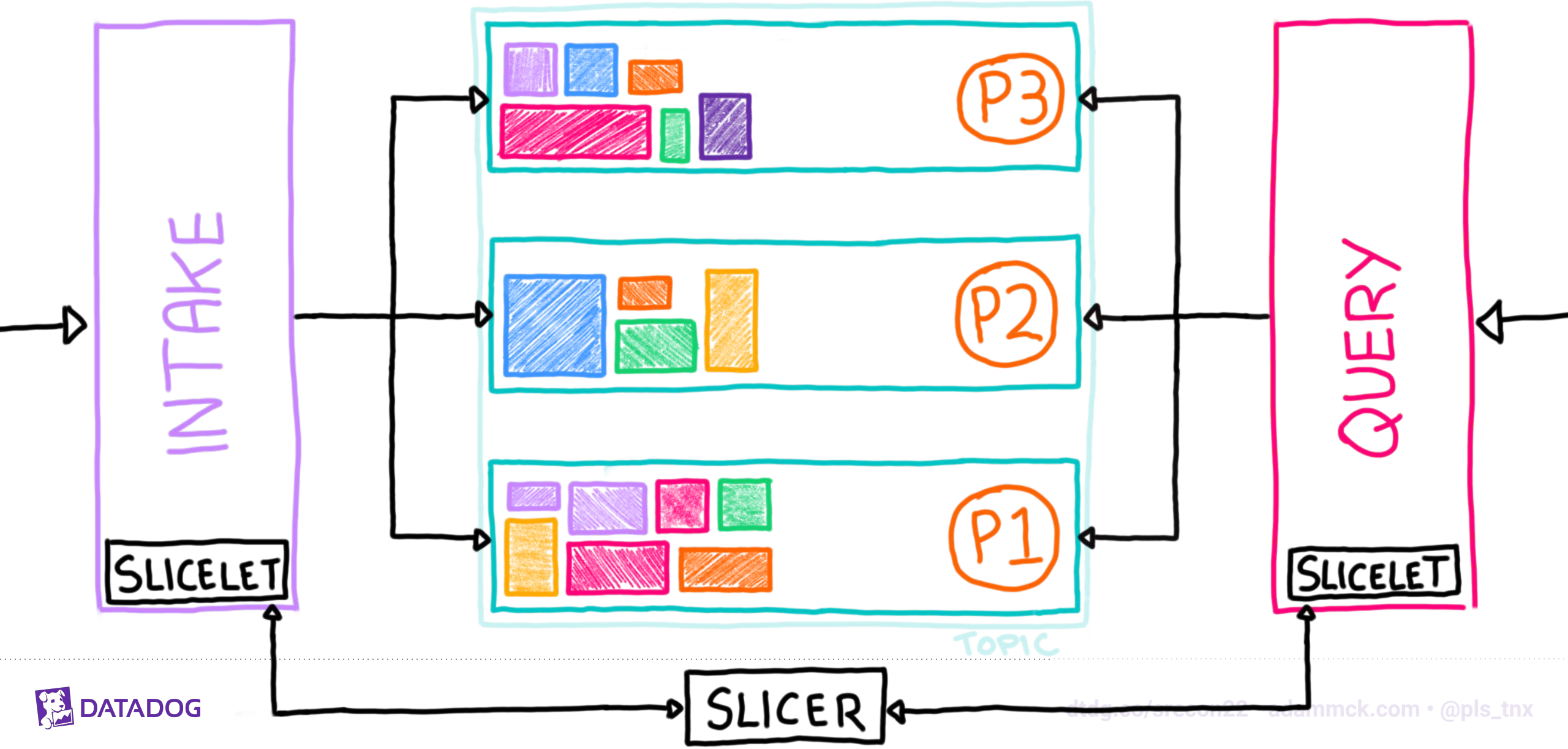


Figure 1: Over five-minute intervals in a recent week, Slicer directed a median of 2 Mreq/s of production traffic with peaks exceeding 7 Mreq/s.

Slicer 🤩



Lesson Learned:

You're gonna need a smart sharding strategy

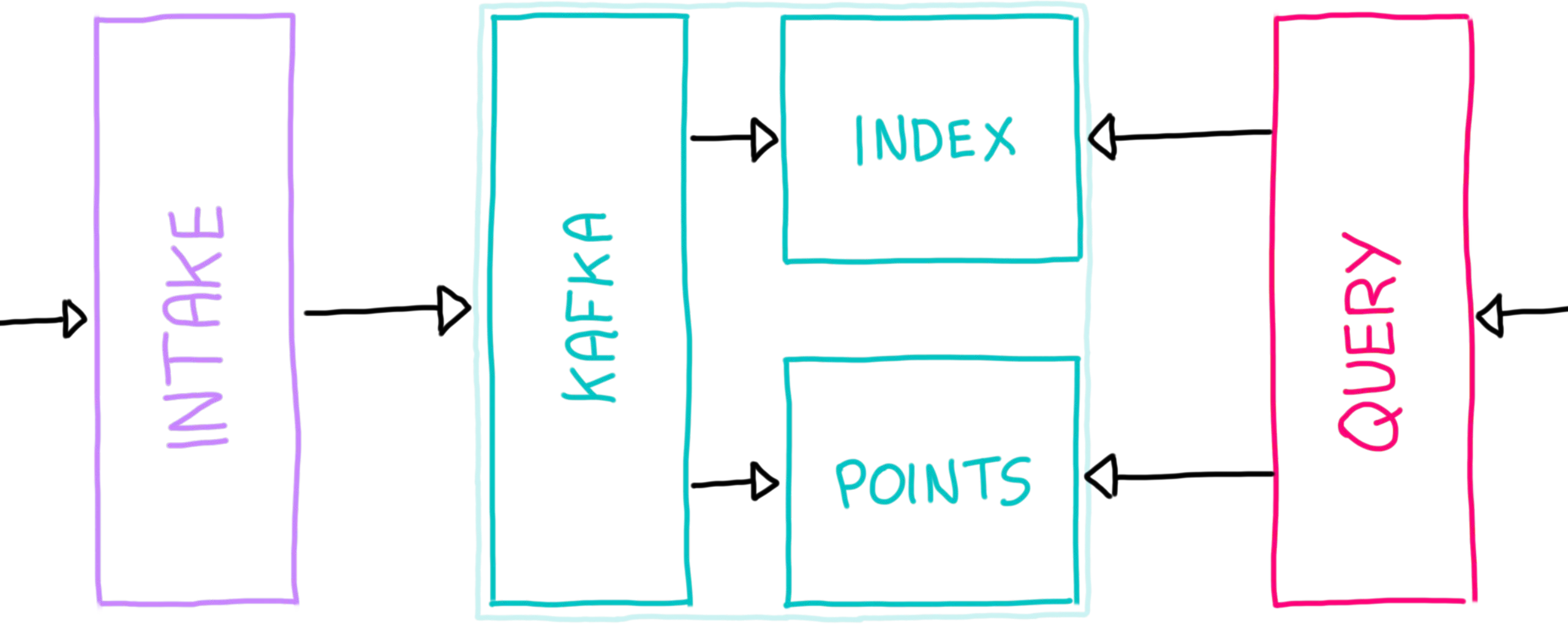
DATADOG

Problem:

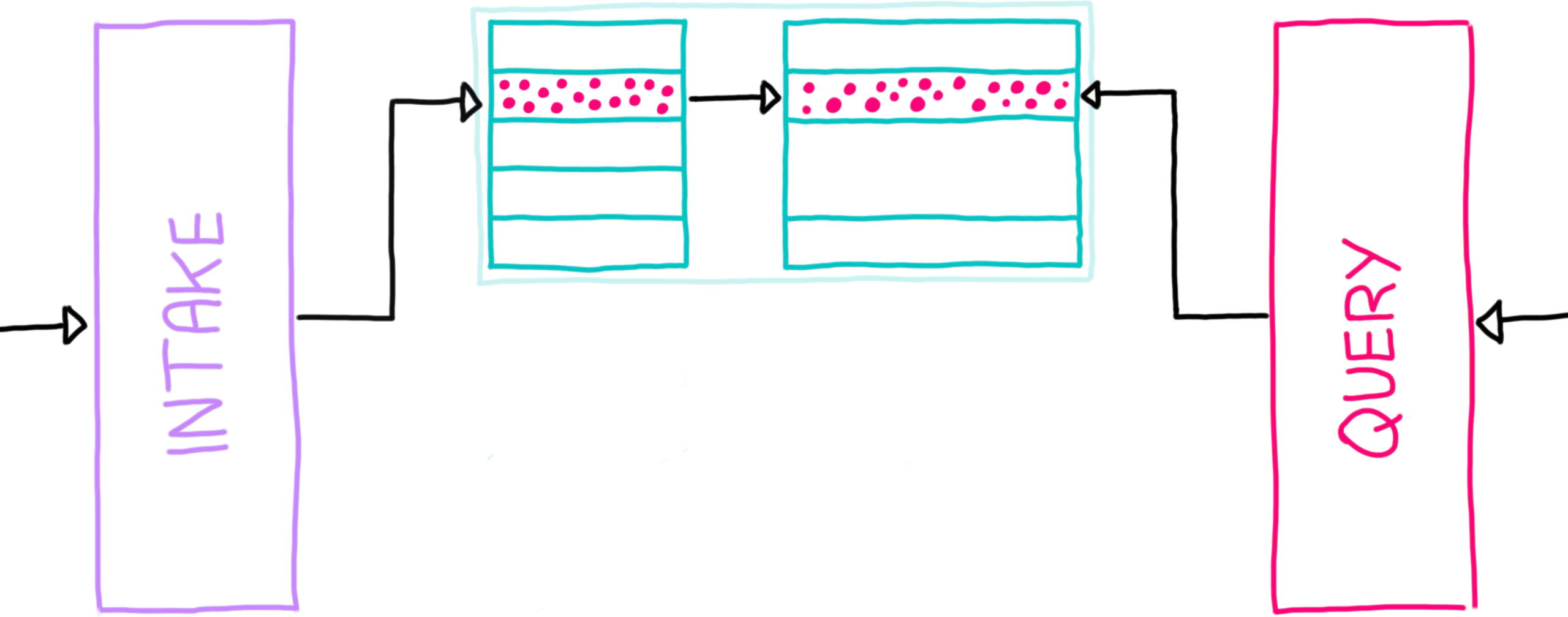
Applying Rebalancing

DATADOG

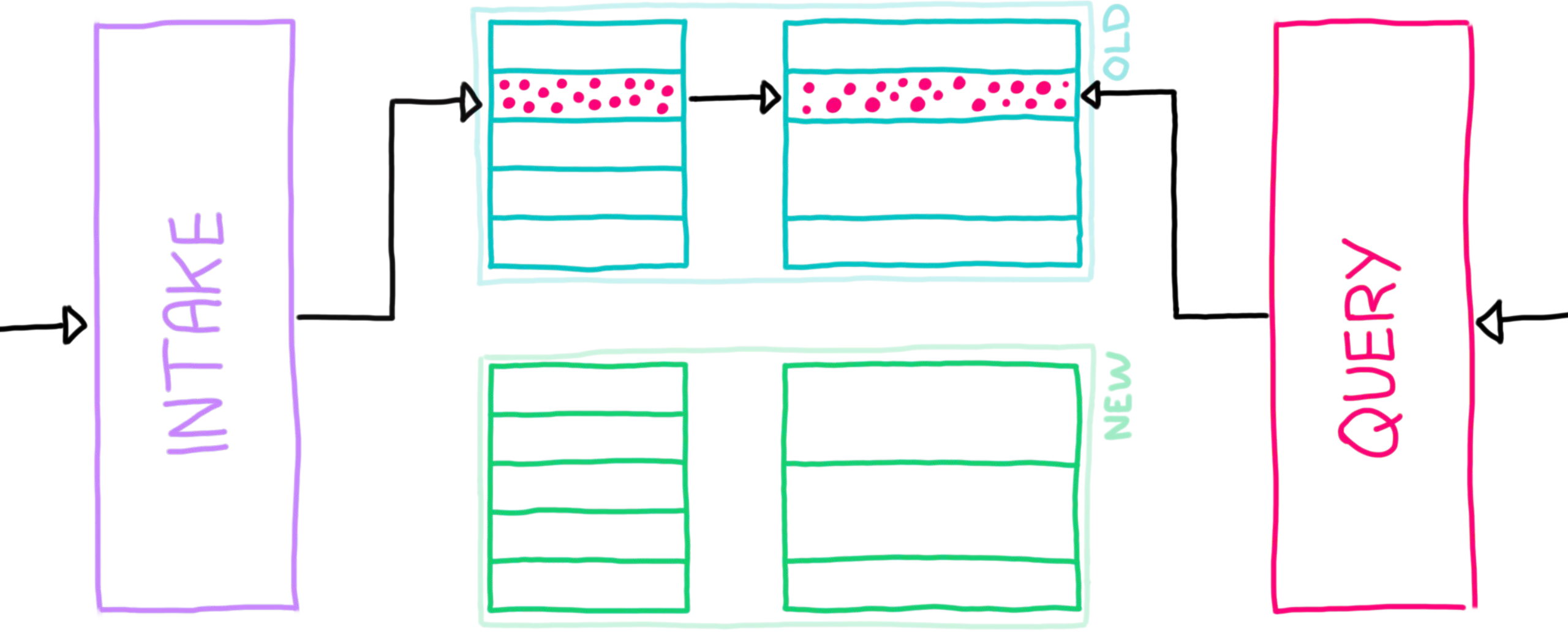
Rebalancing



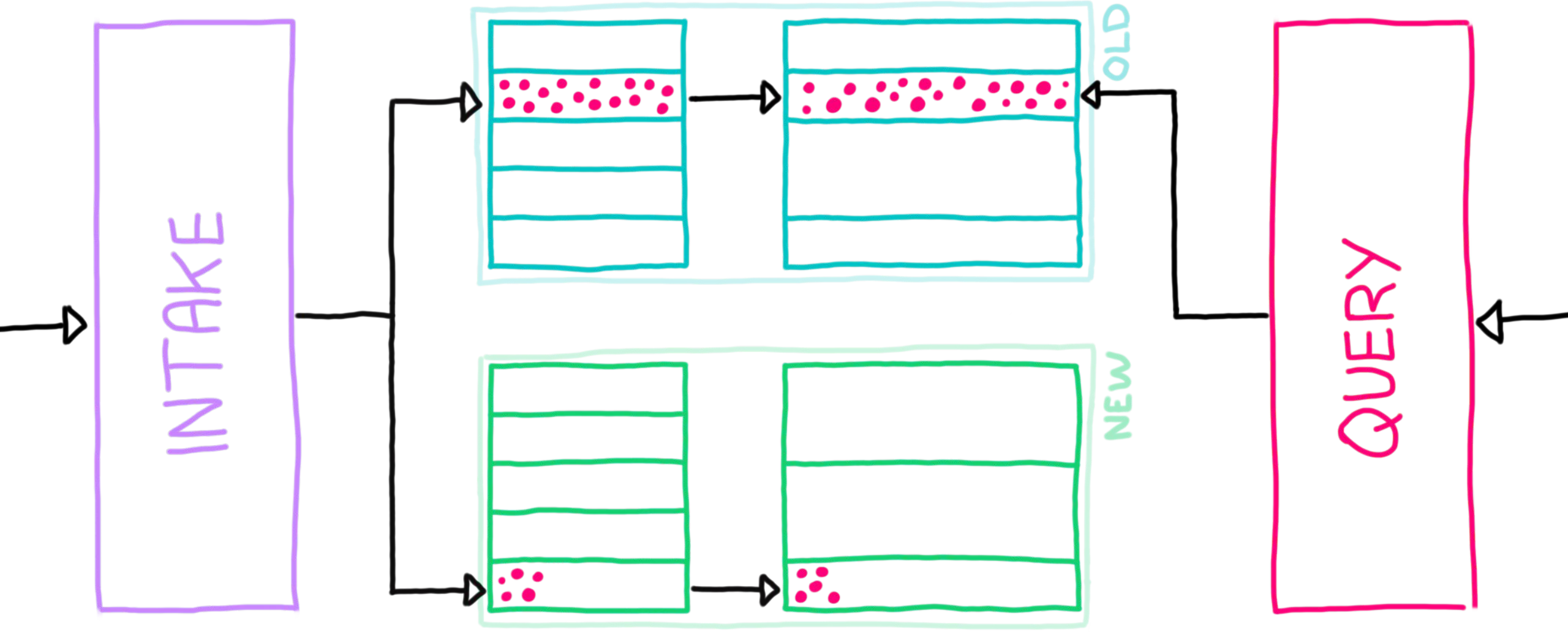
Rebalancing



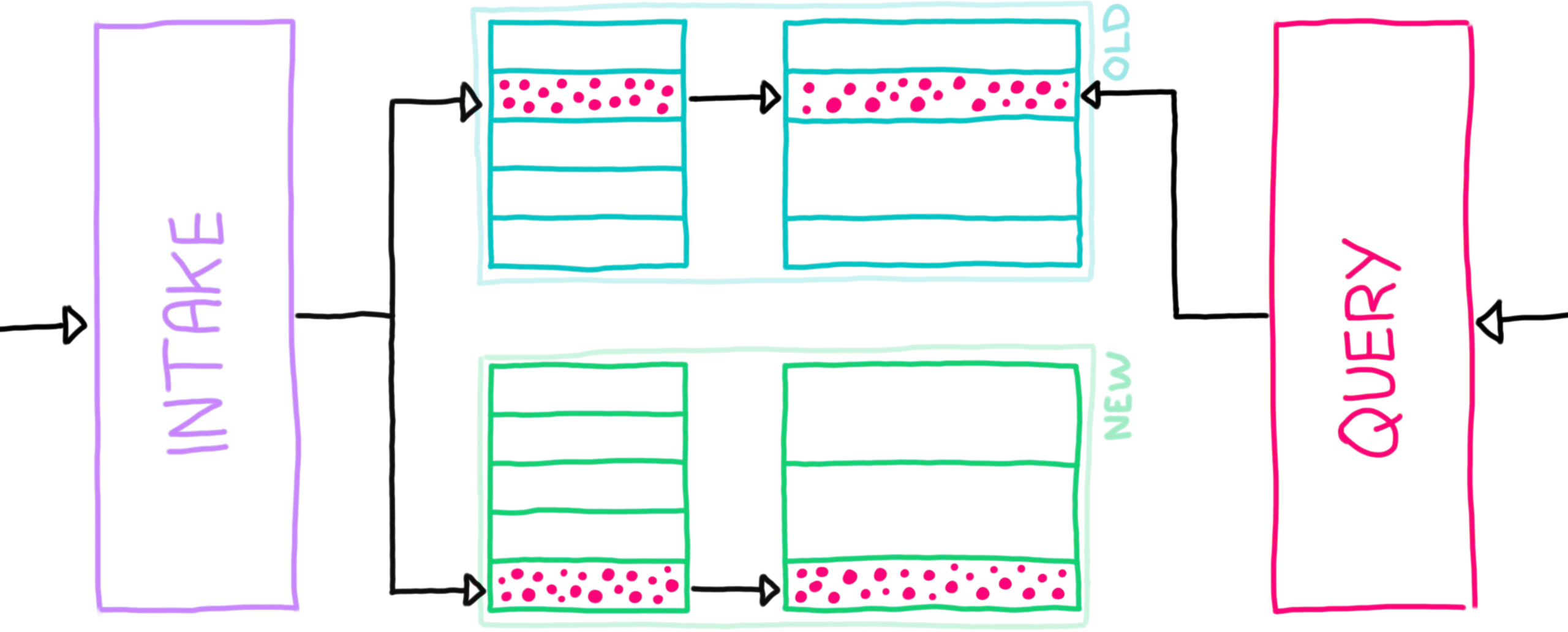
Rebalancing



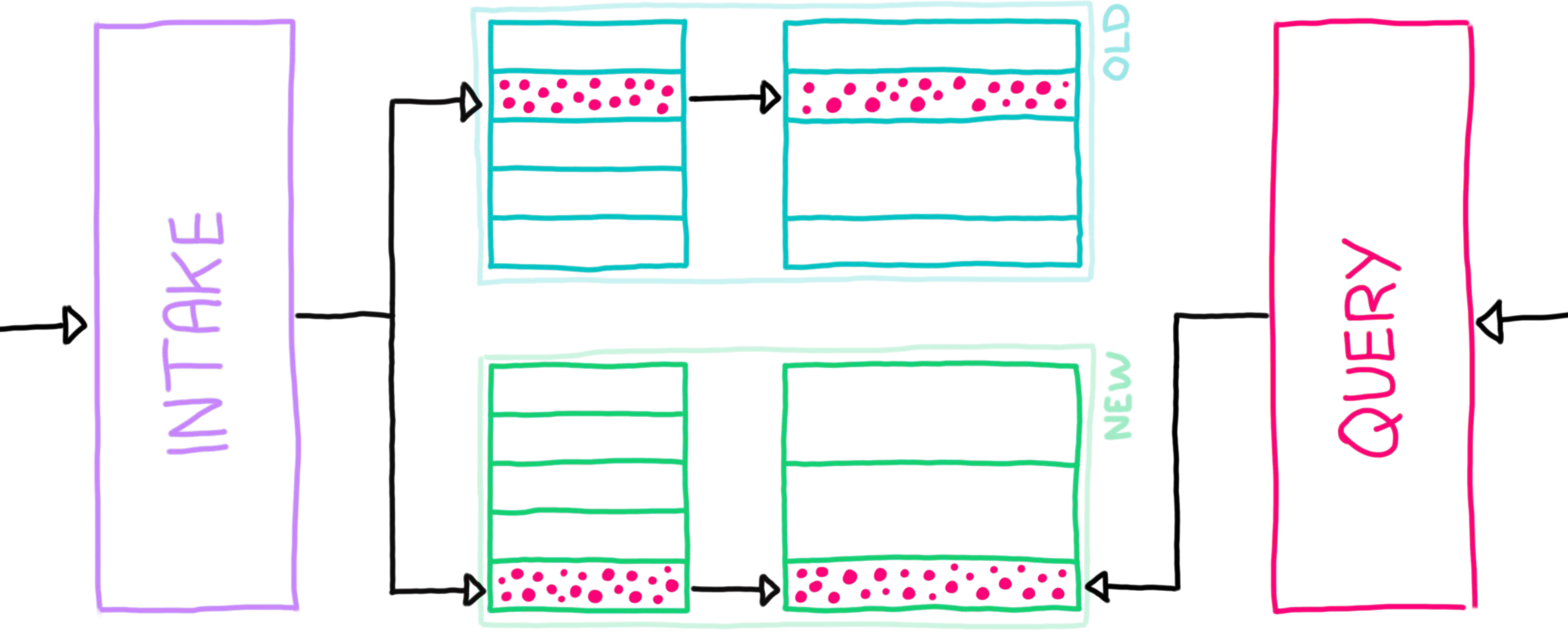
Rebalancing



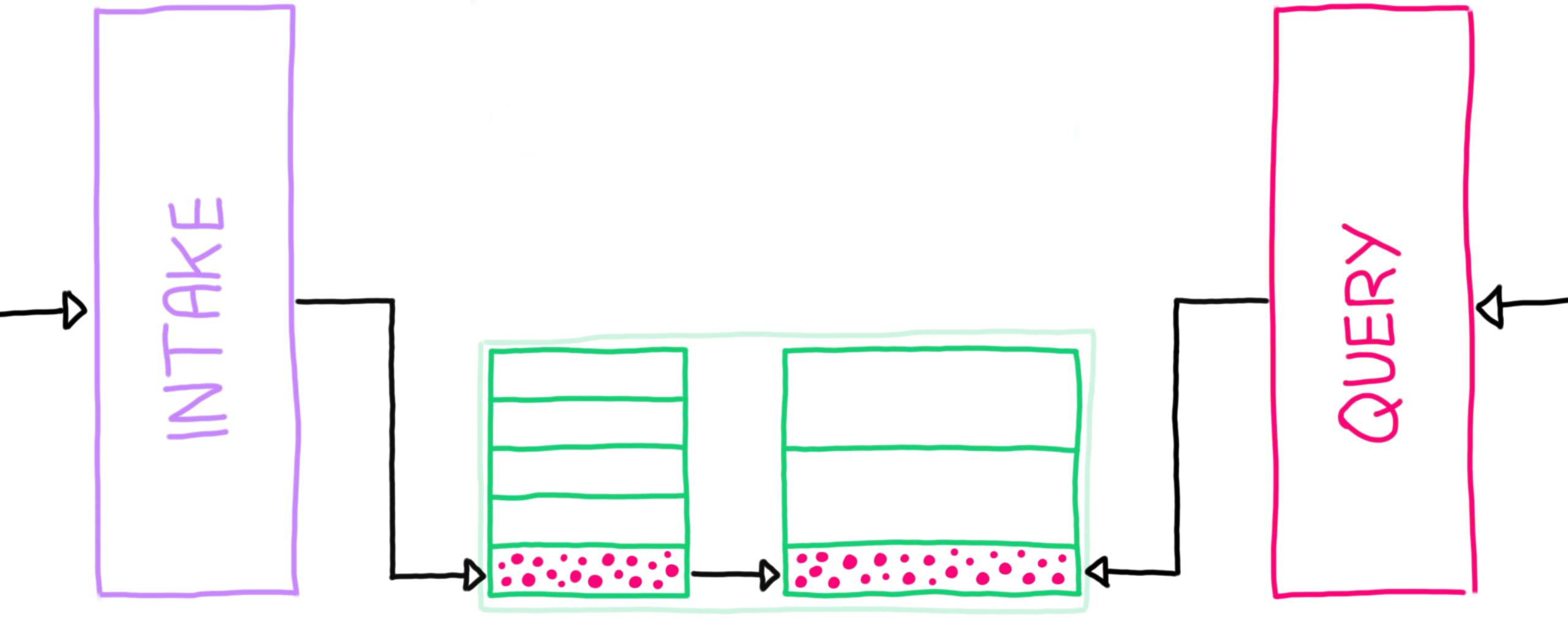
Rebalancing



Rebalancing



Rebalancing



Lesson Learned:

???

DATADOG

Key Takeaways

- **Failure domains are many; choose which to accept**

- Failure domains are many;
  choose which to accept

- You might not need fancy sharding;
  maybe just multiple levels of same

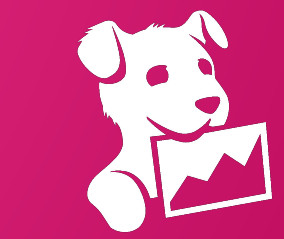
- Failure domains are many;
  choose which to accept

- You might not need fancy sharding;
  maybe just multiple levels of same

- One → Two is the hardest;
  Start with two if you can

DATADOG

- Failure domains are many;
  choose which to accept
- You might not need fancy sharding;
  maybe just multiple levels of same
- One → Two is the hardest;
  Start with two if you can
- Coarse sharding schemes don't
  work for outliers; and that's okay



DATADOG

- Failure domains are many;
  choose which to accept
- You might not need fancy sharding;
  maybe just multiple levels of same
- One → Two is the hardest;
  Start with two if you can
- Coarse sharding schemes don't
  work for outliers; and that's okay
- You're gonna need a smart sharding strategy

Thank You!

dtdg.co/srecon22

btw we're hiring 👀

DATADOG