# Reindexing In Record Time

How Shopify Reindexes Search for 800,000+ Merchants

Conor Landry

@ConorLandry_

shopify

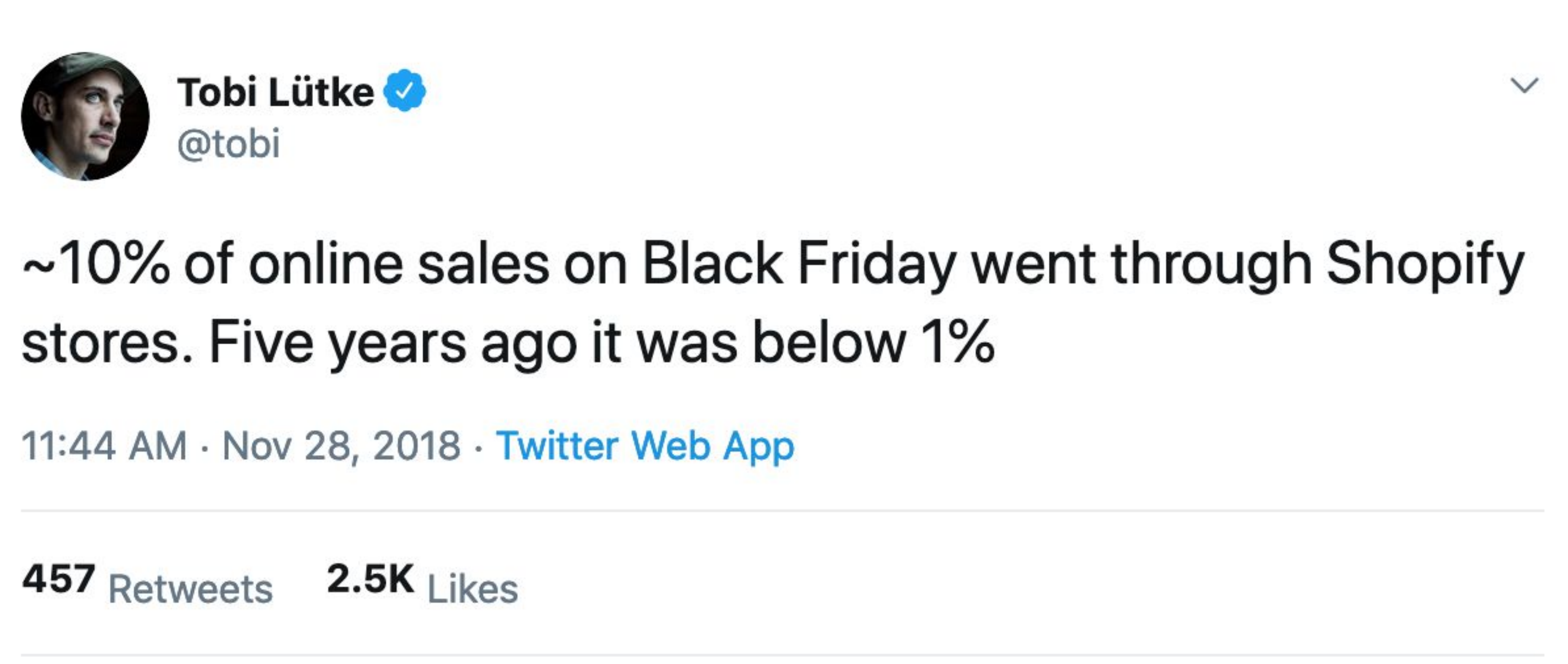
shopify

$100B+

Gross Merchant Volume

4000

Employees



Tobi Lütke ✓
@tobi

~10% of online sales on Black Friday went through Shopify stores. Five years ago it was below 1%

11:44 AM · Nov 28, 2018 · Twitter Web App

457 Retweets 2.5K Likes

# Agenda

1) Elasticsearch @ Shopify

2) Shopify Core Topology

3) What is Reindexing?

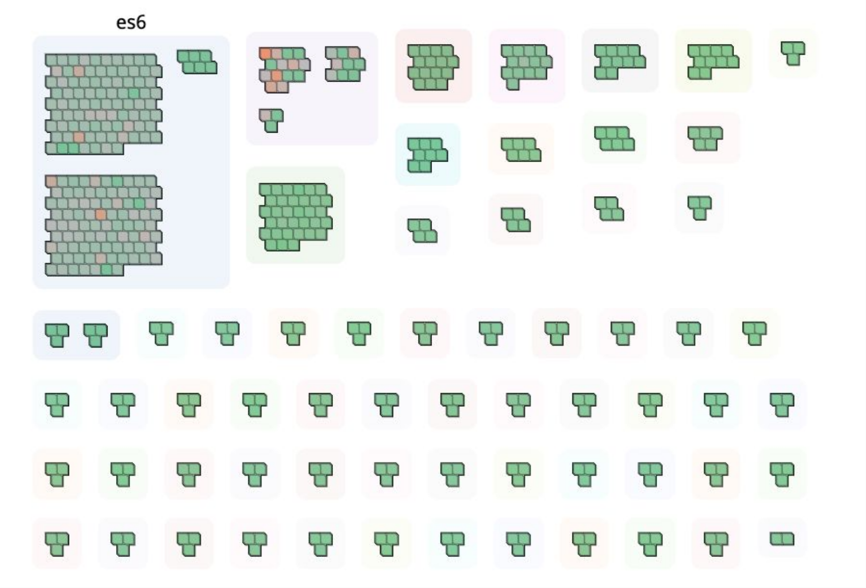
4) How We Reindex Elasticsearch

5) Future Plans & Improvements

@ConorLandry_

# Elasticsearch @ Shopify

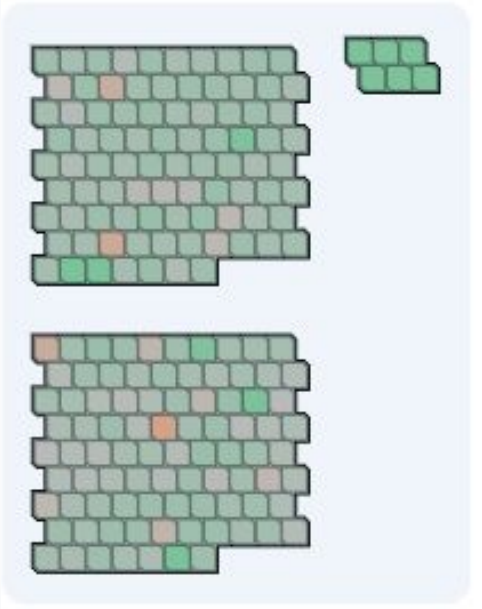
- 60+ managed Elasticsearch clusters
- Built a Kubernetes Controller for search
- Our largest cluster has ~50 billion documents



@ConorLandry_

# Elasticsearch @ Shopify

- Active-active
- Replicated across 2 regions
- ~90 nodes per region
- ~100,000 queries per minute
- Routed by shop ID
- 1 replica per shard



@ConorLandry

# Shopify Core Stack

- Hundreds of services at Shopify, largest is “Shopify Core”
- Core is a big Rails monolith, sharded by shop ID
- 99% in Kubernetes, data centers for load balancing
- Running in GCP regions in us-central1, us-east1, and northamerica-northeast1 🌿

RAILS

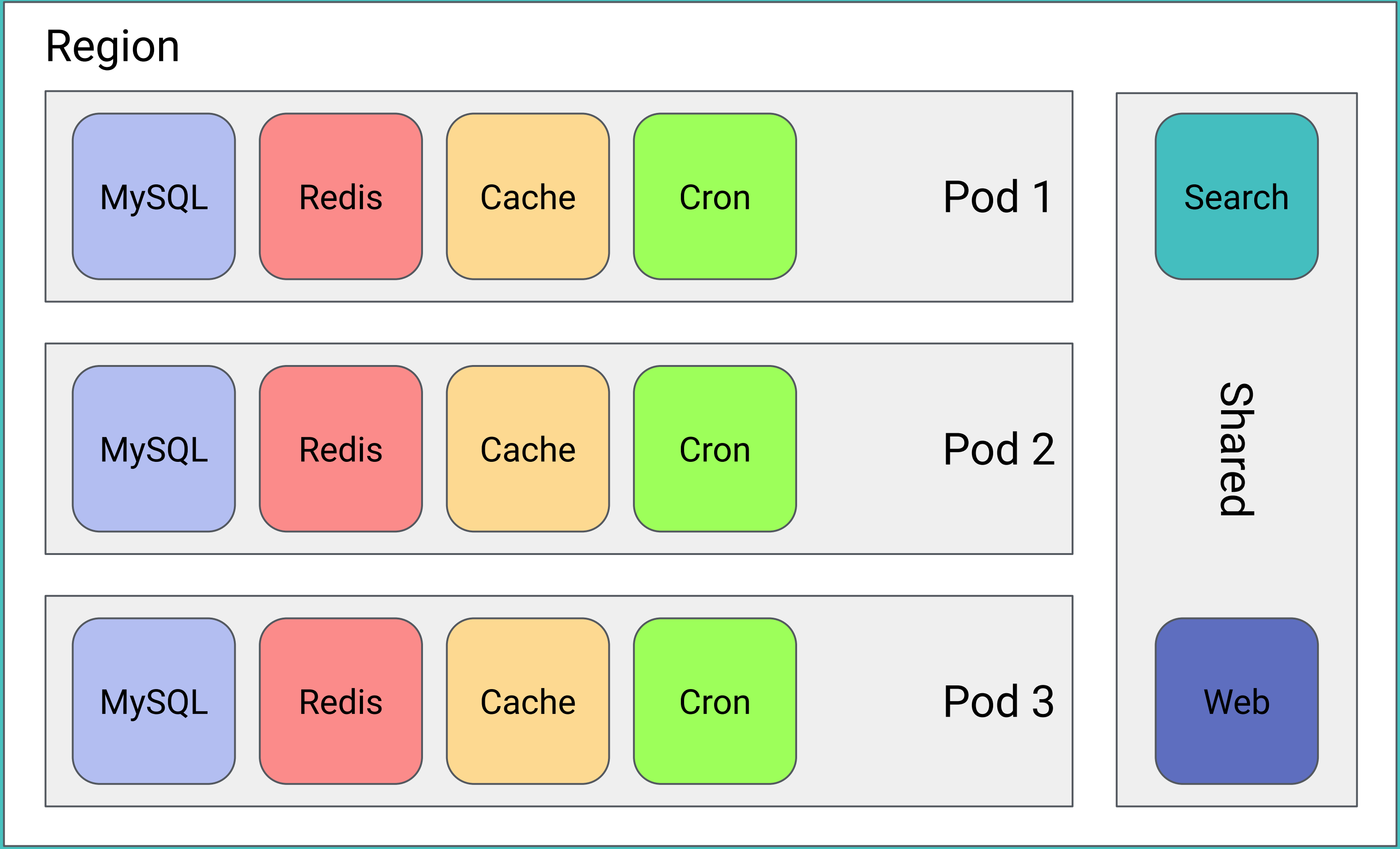
redis

kafka

elasticsearch

MySQL®

kubernetes



Region

MySQL Redis Cache Cron Pod 1

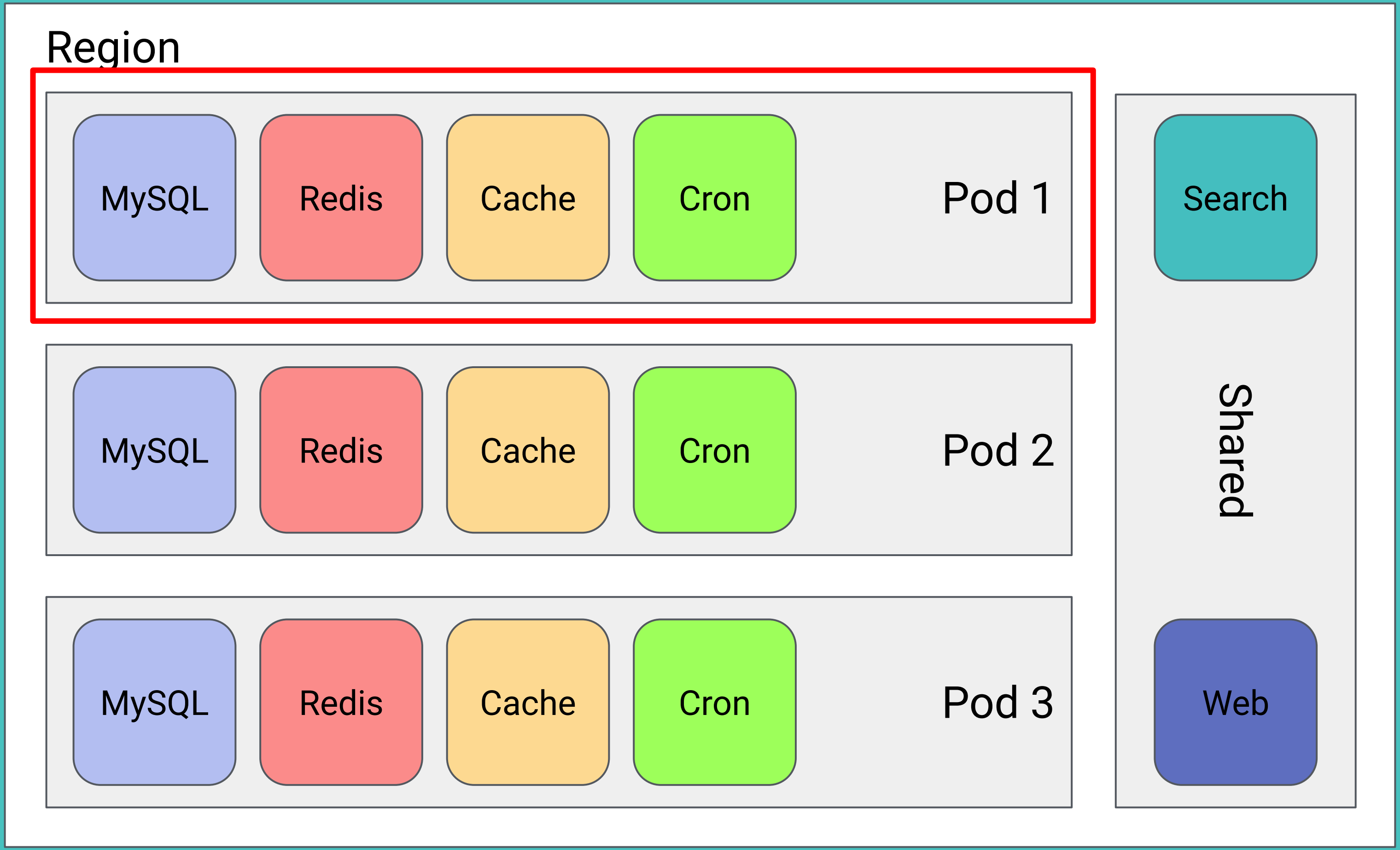
MySQL Redis Cache Cron Pod 2

MySQL Redis Cache Cron Pod 3

Search

Shared

Web



Region

MySQL Redis Cache Cron Pod 1

MySQL Redis Cache Cron Pod 2

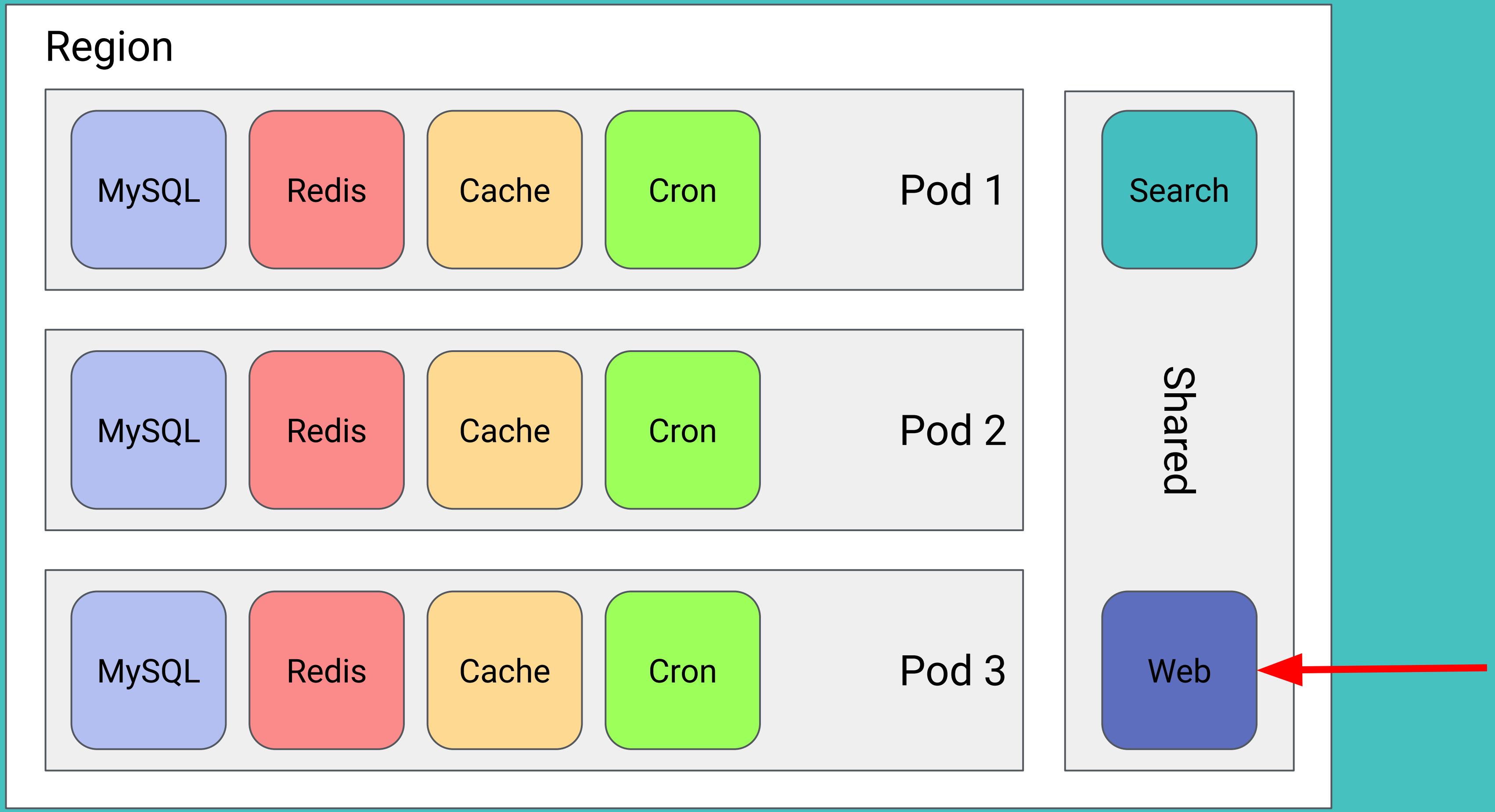
MySQL Redis Cache Cron Pod 3

Search

Shared

Web

@ConorLandry_



Region

MySQL Redis Cache Cron Pod 1

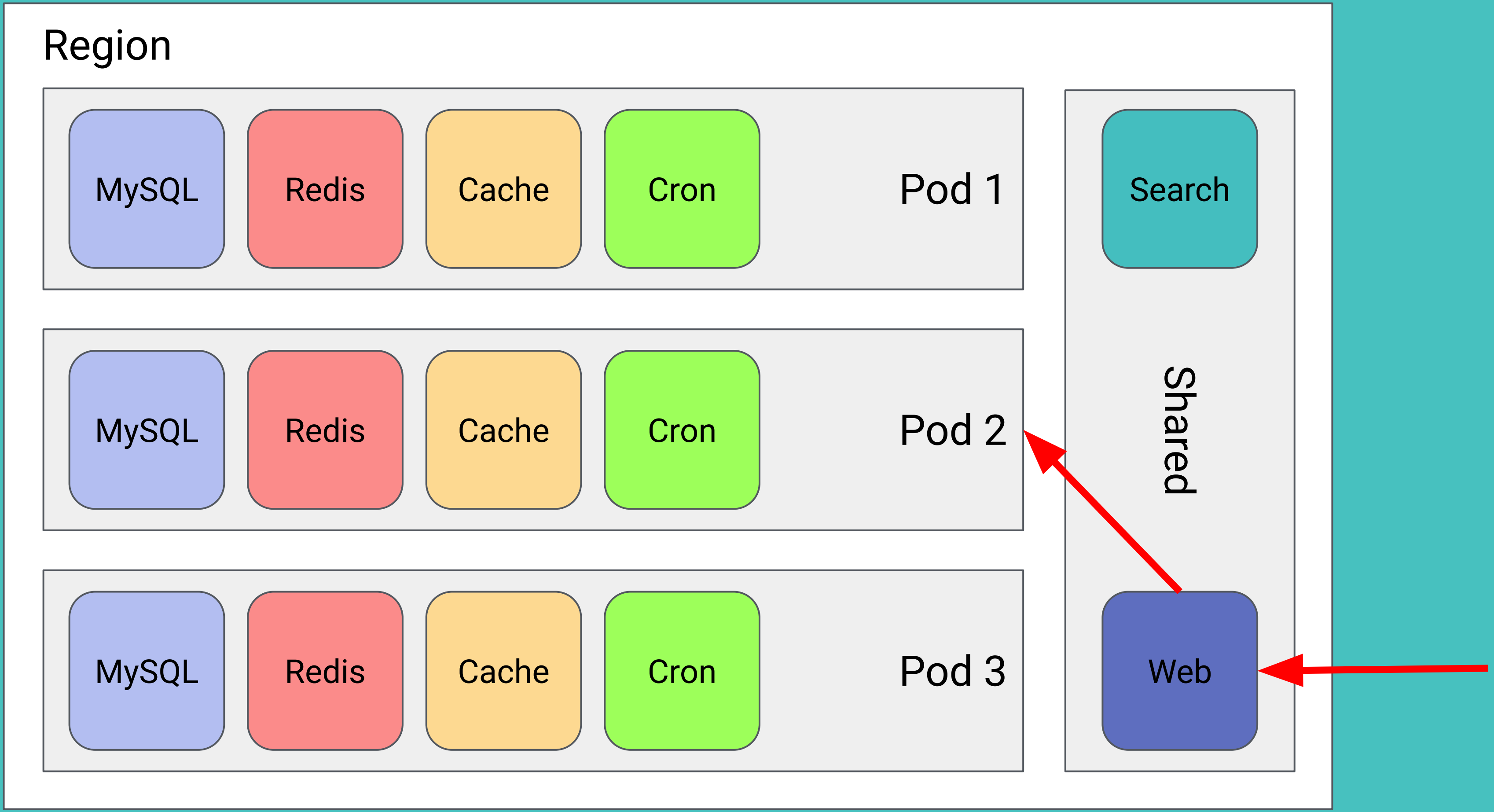
MySQL Redis Cache Cron Pod 2

MySQL Redis Cache Cron Pod 3

Shared

Search

Web



Region

MySQL Redis Cache Cron Pod 1

MySQL Redis Cache Cron Pod 2

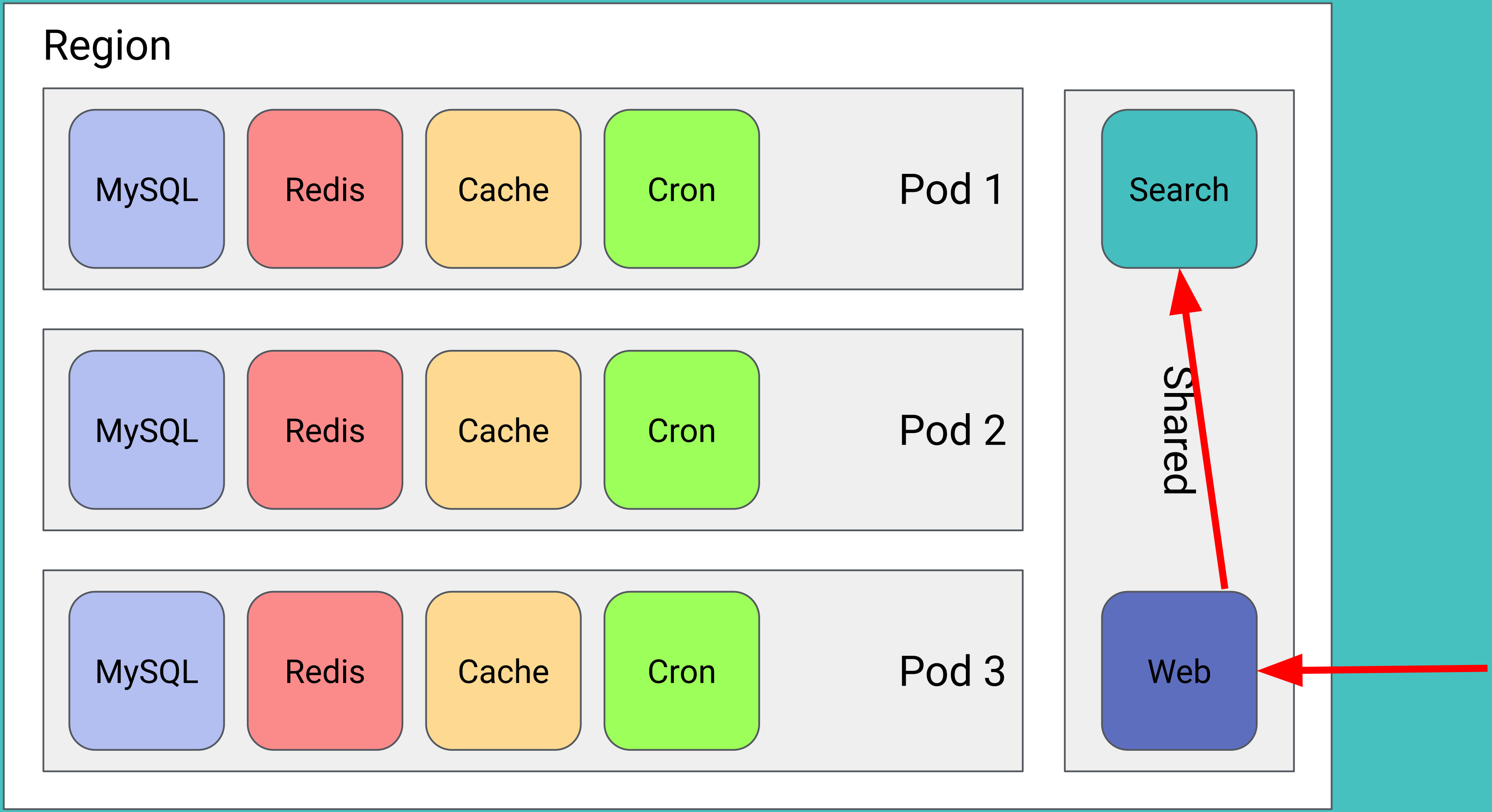
MySQL Redis Cache Cron Pod 3

Search

Shared

Web

@ConorLandry_



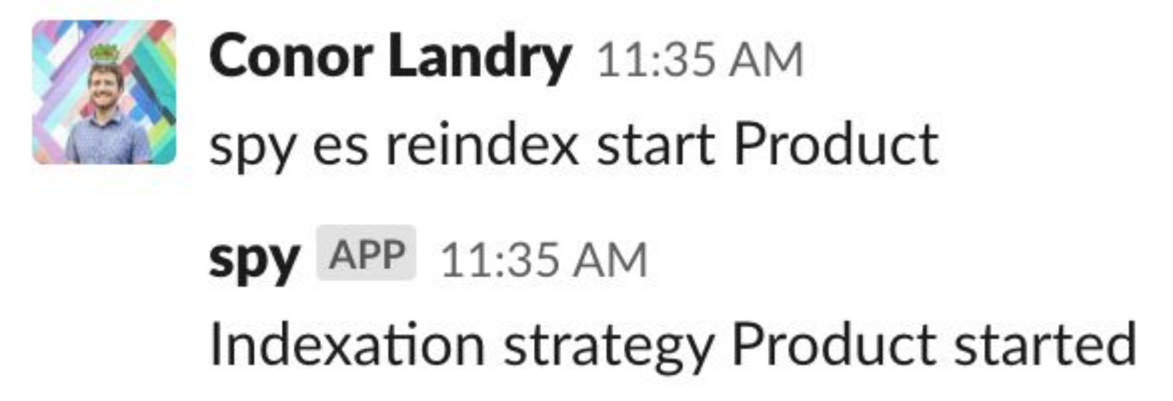
@ConorLandry_

# What Is Reindexing?

- Often developers need to add new fields, modify analyzers, or change an existing field
- Reindexing (for us) is the process of migrating data from one index version to another after making mapping changes
- Different from “real-time” indexing

# So how do we start reindexing?

## It starts with a slack message



**Conor Landry** 11:35 AM

spy es reindex start Product

**spy** APP 11:35 AM

Indexation strategy Product started

@ConorLandry_

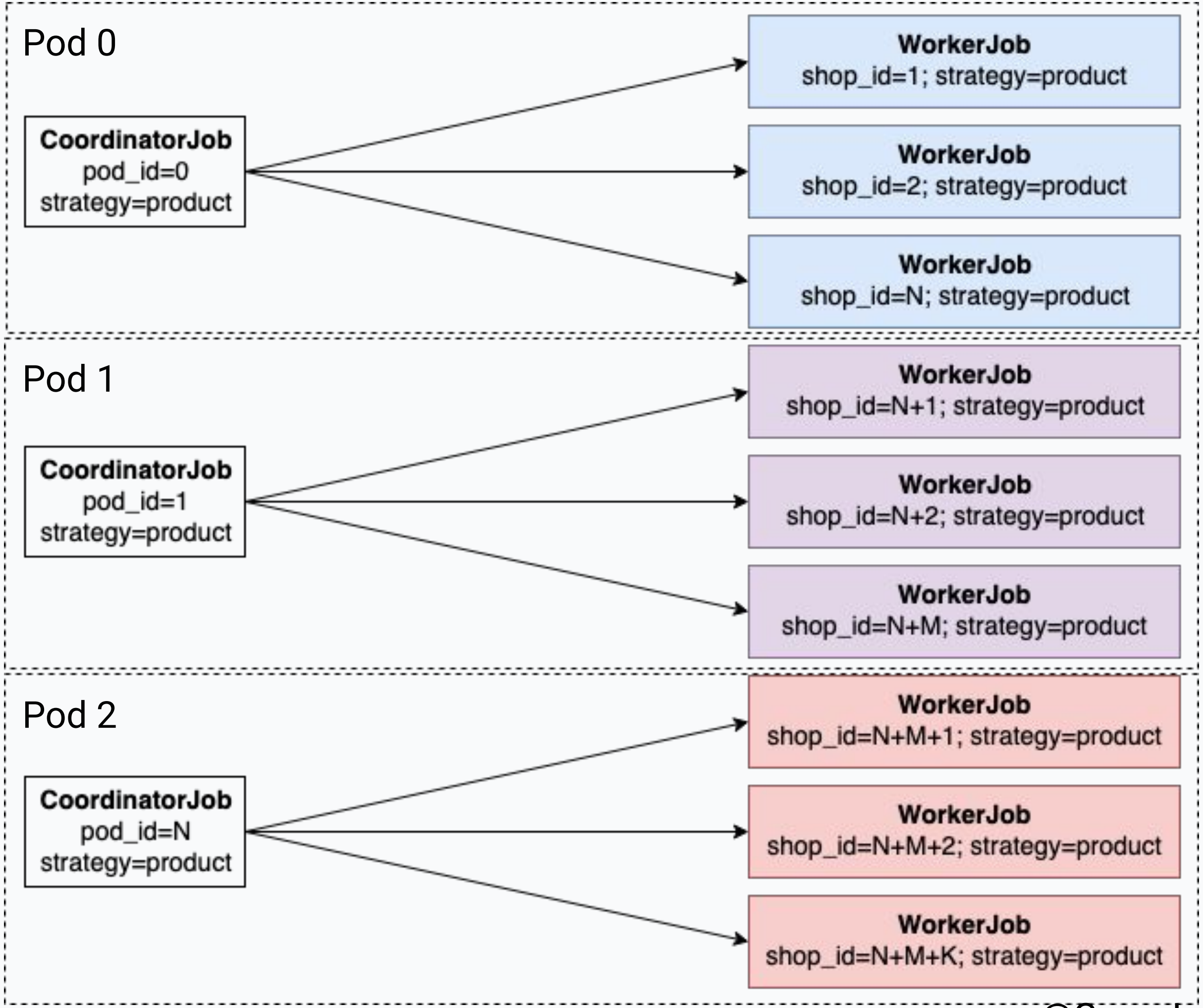
| Alias | Index |
|---|---|
| products | products.0 |
| products.new | |

| Alias | Index |
|---|---|
| products | products.0 |
| products.new | products.1 |

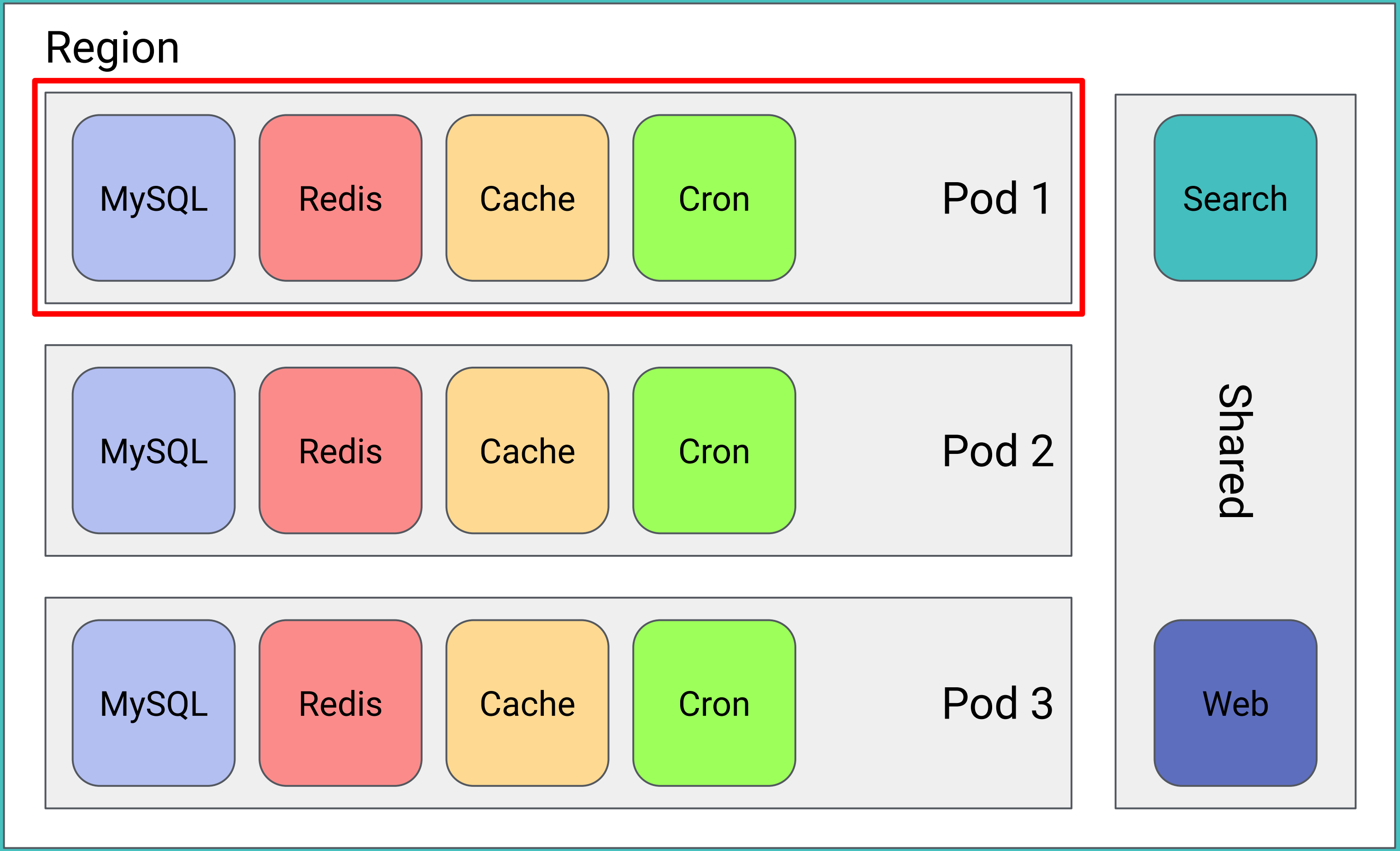
# Moving documents from MySQL to ES

@ConorLandry_

- All reindexing work starts in the monolith
- Parent-child set of ActiveJob / Resque jobs
- One worker job per active shop
- Throttled based on number of worker jobs running per pod


Pod 0
CoordinatorJob
pod_id=0
strategy=product
WorkerJob
shop_id=1; strategy=product
WorkerJob
shop_id=2; strategy=product
WorkerJob
shop_id=N; strategy=product
Pod 1
CoordinatorJob
pod_id=1
strategy=product
WorkerJob
shop_id=N+1; strategy=product
WorkerJob
shop_id=N+2; strategy=product
WorkerJob
shop_id=N+M; strategy=product
Pod 2
CoordinatorJob
pod_id=N
strategy=product
WorkerJob
shop_id=N+M+1; strategy=product
WorkerJob
shop_id=N+M+2; strategy=product
WorkerJob
shop_id=N+M+K; strategy=product

@ConorLandry_



Region

MySQL Redis Cache Cron Pod 1

MySQL Redis Cache Cron Pod 2

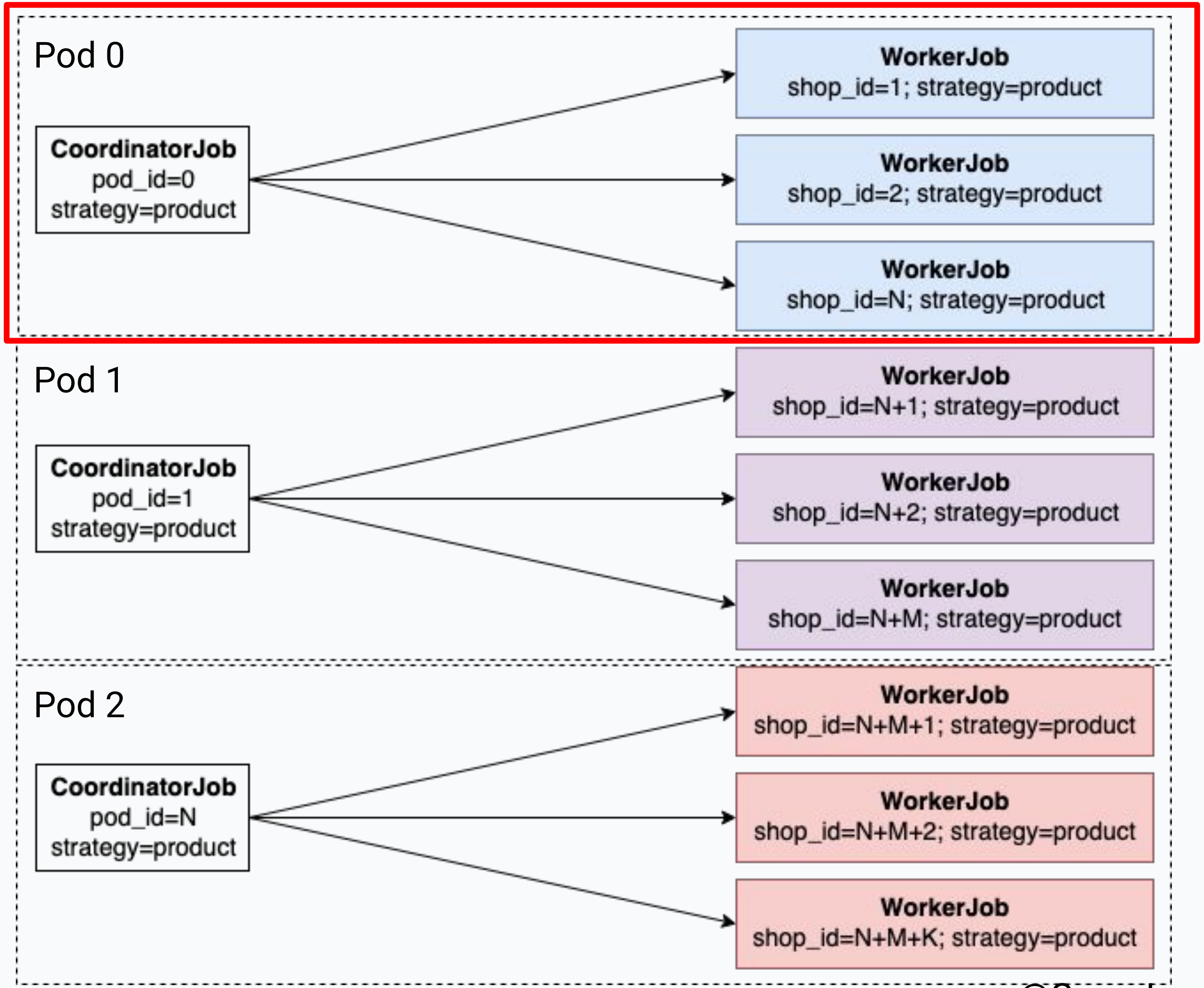
MySQL Redis Cache Cron Pod 3

Shared

Search

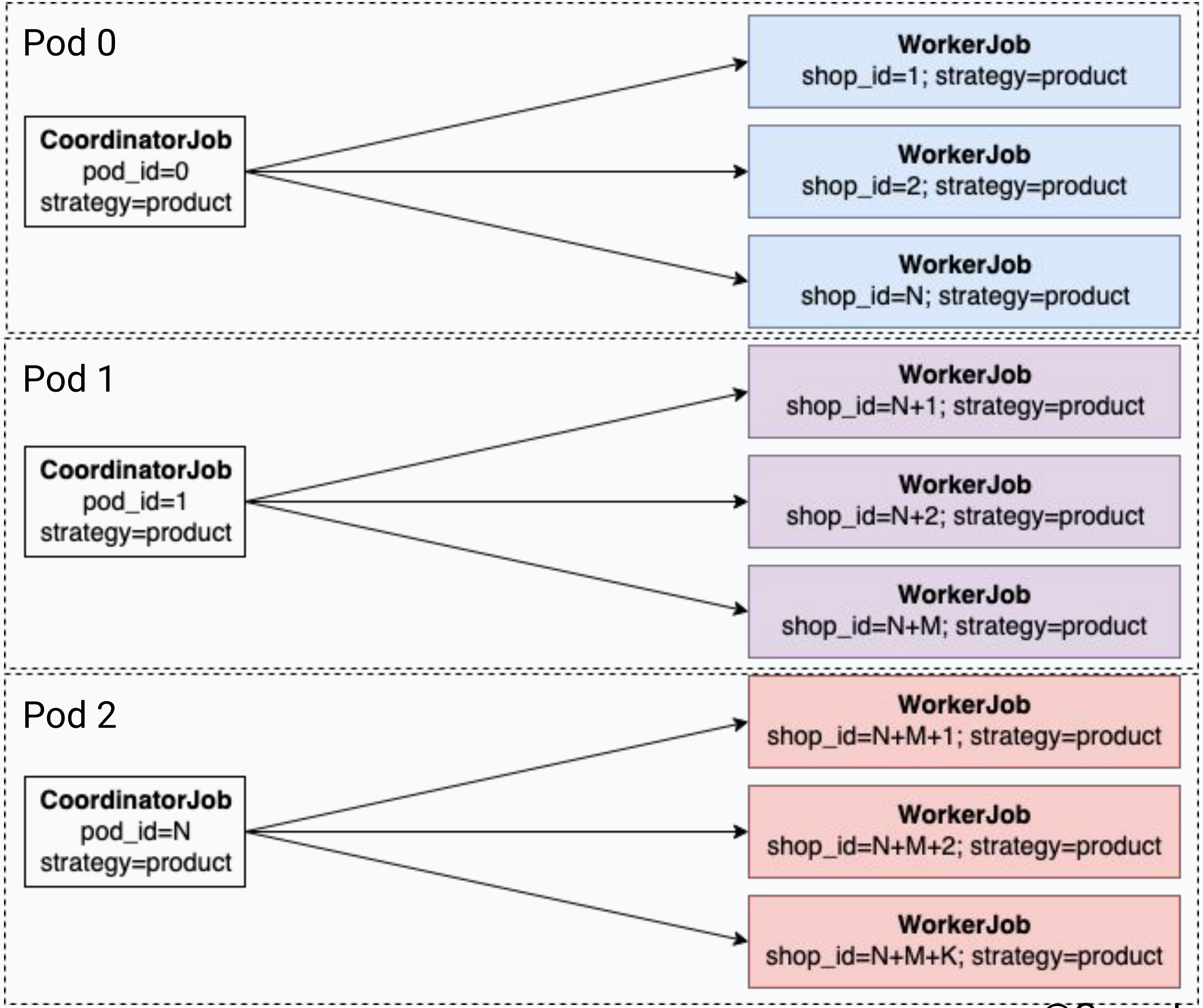
Web

- All reindexing work starts in the monolith
- Parent-child set of ActiveJob / Resque jobs
- One worker job per active shop
- Throttled based on number of worker jobs running per pod


Pod 0
**CoordinatorJob**
pod_id=0
strategy=product
**WorkerJob**
shop_id=1; strategy=product
**WorkerJob**
shop_id=2; strategy=product
**WorkerJob**
shop_id=N; strategy=product
Pod 1
**CoordinatorJob**
pod_id=1
strategy=product
**WorkerJob**
shop_id=N+1; strategy=product
**WorkerJob**
shop_id=N+2; strategy=product
**WorkerJob**
shop_id=N+M; strategy=product
Pod 2
**CoordinatorJob**
pod_id=N
strategy=product
**WorkerJob**
shop_id=N+M+1; strategy=product
**WorkerJob**
shop_id=N+M+2; strategy=product
**WorkerJob**
shop_id=N+M+K; strategy=product

@ConorLandry_

- All reindexing work starts in the monolith
- Parent-child set of ActiveJob / Resque jobs
- One worker job per active shop
- Throttled based on number of worker jobs running per pod



Pod 0
**CoordinatorJob** pod_id=0 strategy=product
**WorkerJob** shop_id=1; strategy=product
**WorkerJob** shop_id=2; strategy=product
**WorkerJob** shop_id=N; strategy=product
Pod 1
**CoordinatorJob** pod_id=1 strategy=product
**WorkerJob** shop_id=N+1; strategy=product
**WorkerJob** shop_id=N+2; strategy=product
**WorkerJob** shop_id=N+M; strategy=product
Pod 2
**CoordinatorJob** pod_id=N strategy=product
**WorkerJob** shop_id=N+M+1; strategy=product
**WorkerJob** shop_id=N+M+2; strategy=product
**WorkerJob** shop_id=N+M+K; strategy=product

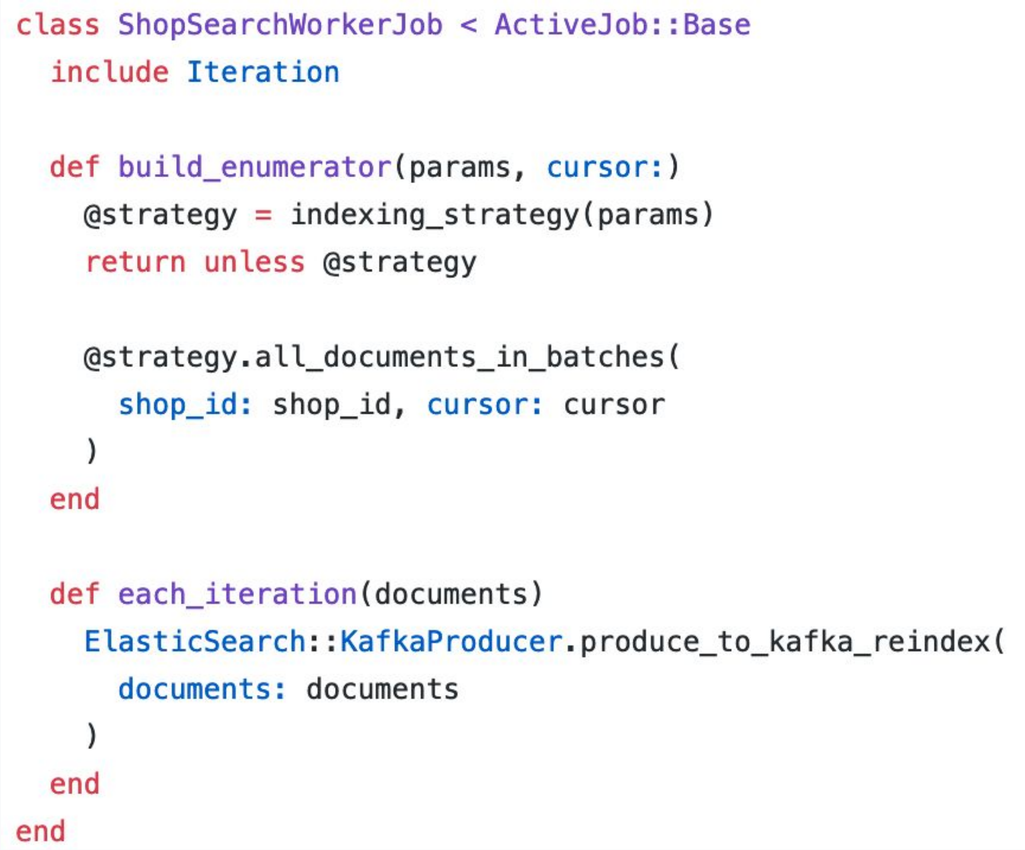
@ConorLandry_

# Okay, but won’t this use up all of Shopify’s Resque worker capacity?

# Iteration API

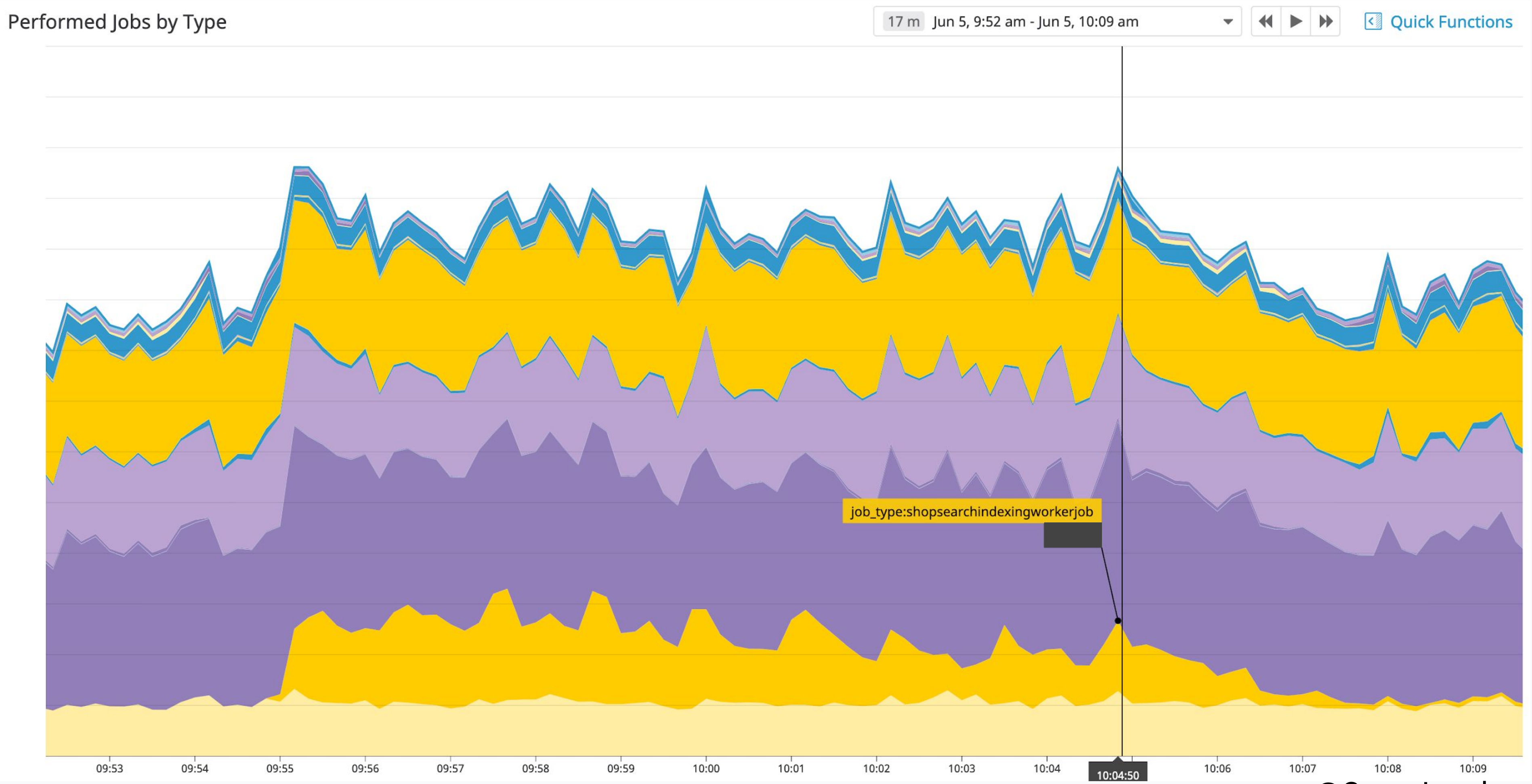
- Split the job definition
    - Collection to process
    - Action to apply

https://github.com/Shopify/job-iteration



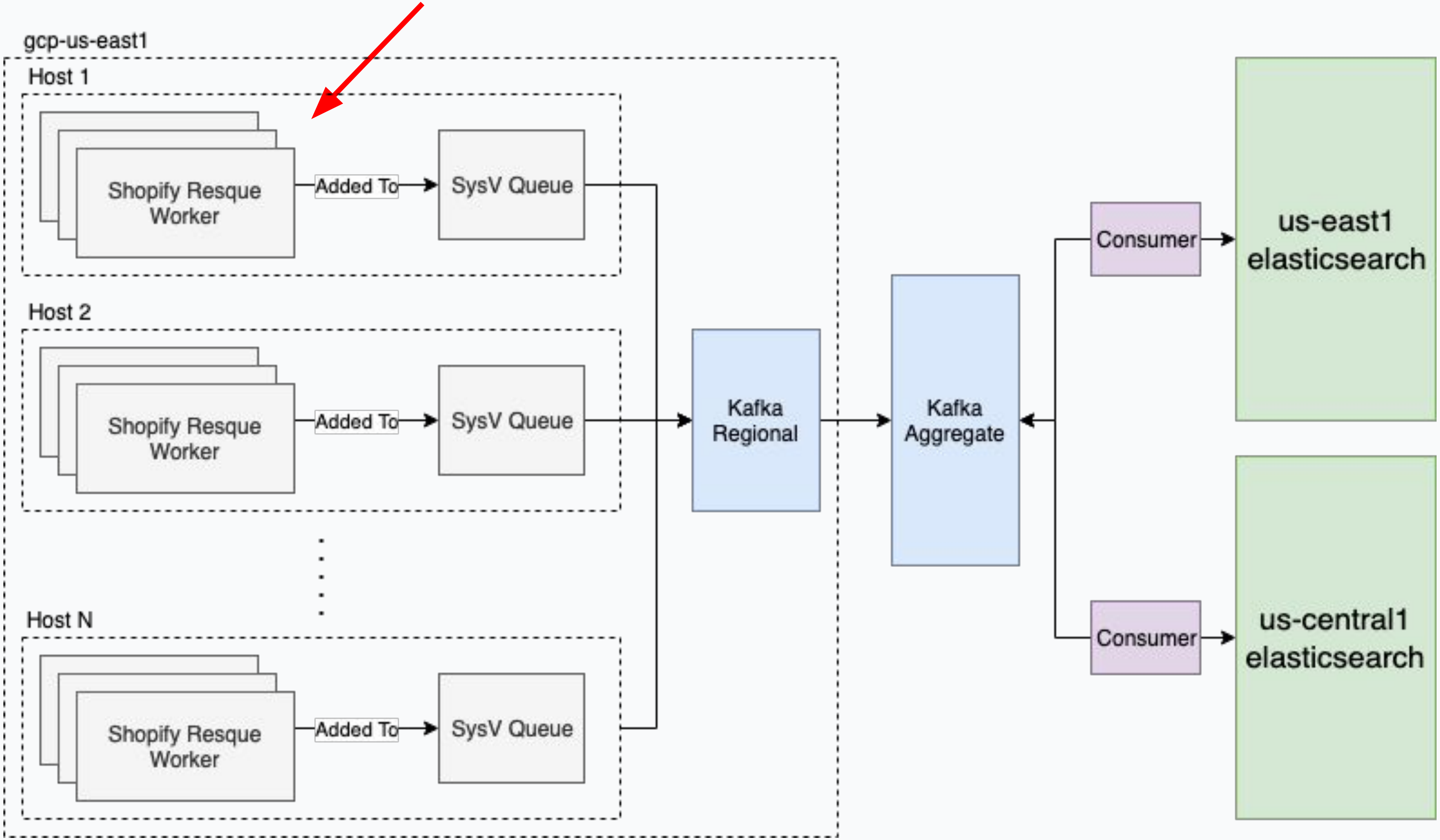
```ruby
class ShopSearchWorkerJob < ActiveJob::Base
  include Iteration

  def build_enumerator(params, cursor:)
    @strategy = indexing_strategy(params)
    return unless @strategy

    @strategy.all_documents_in_batches(
      shop_id: shop_id, cursor: cursor
    )
  end

  def each_iteration(documents)
    ElasticSearch::KafkaProducer.produce_to_kafka_reindex(
      documents: documents
    )
  end
end
```

@ConorLandry_

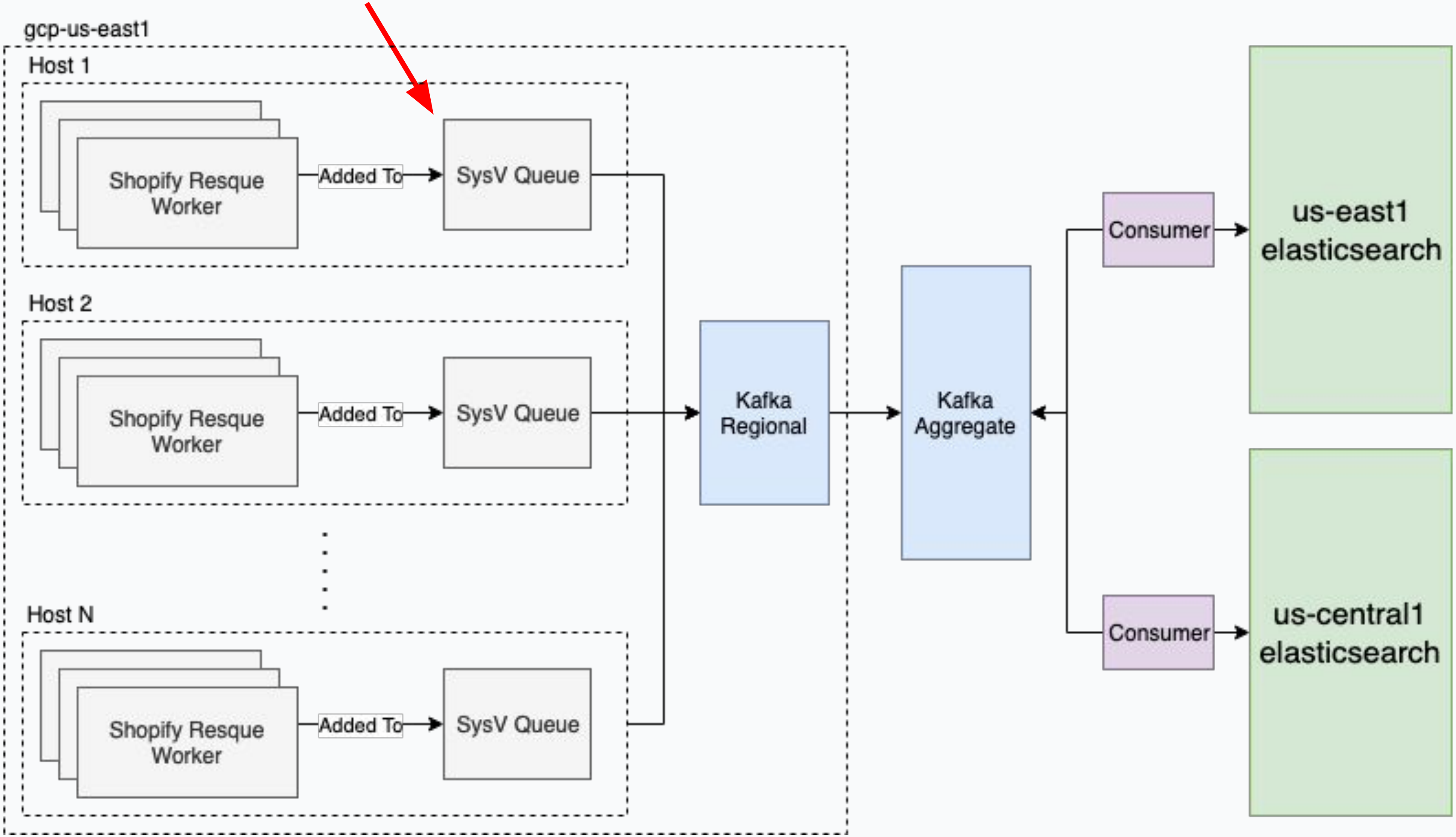


@ConorLandry_

# Indexing Into Elasticsearch



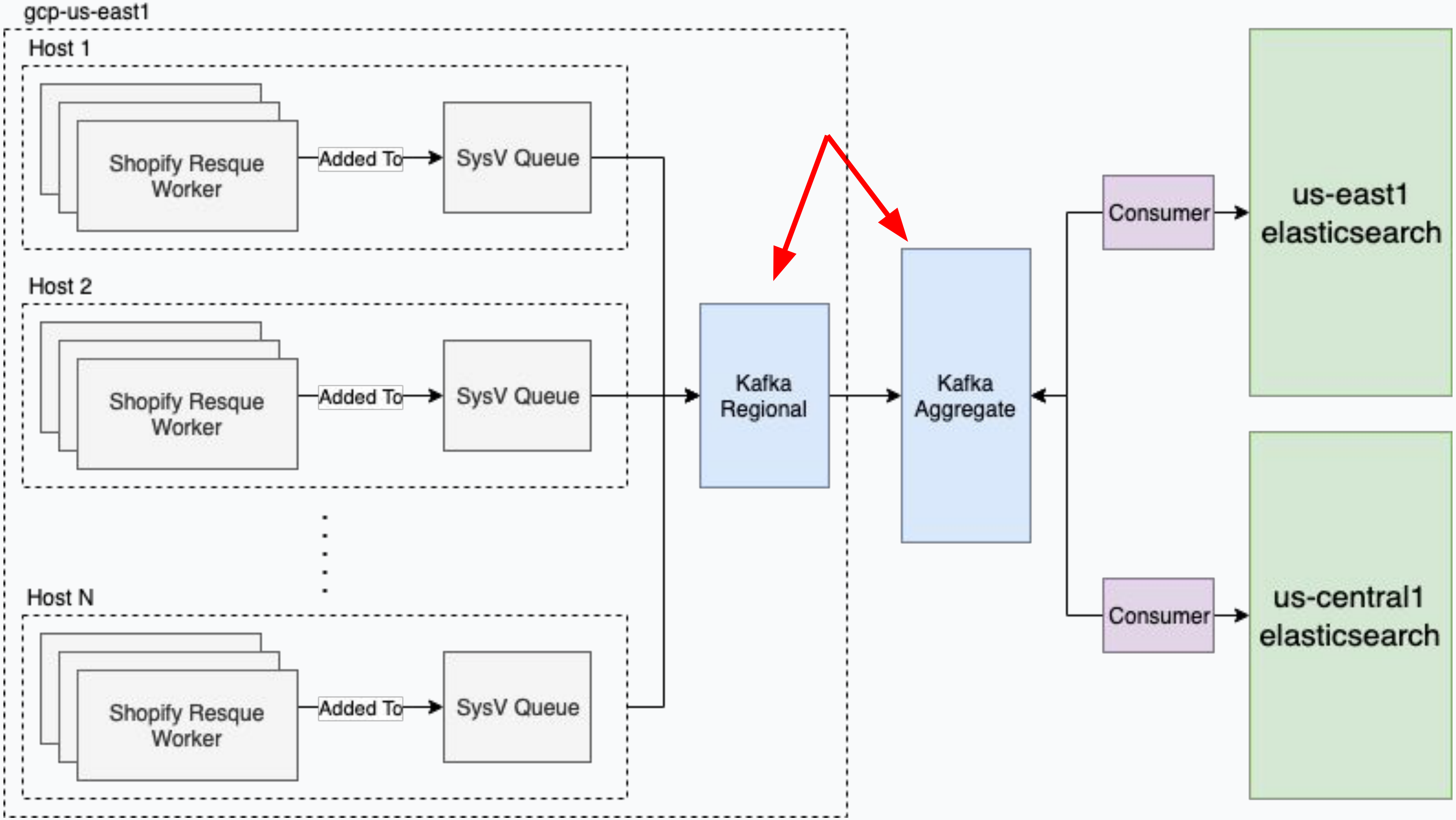
@ConorLandry_



@ConorLandry_

# Why System V Message Queues?

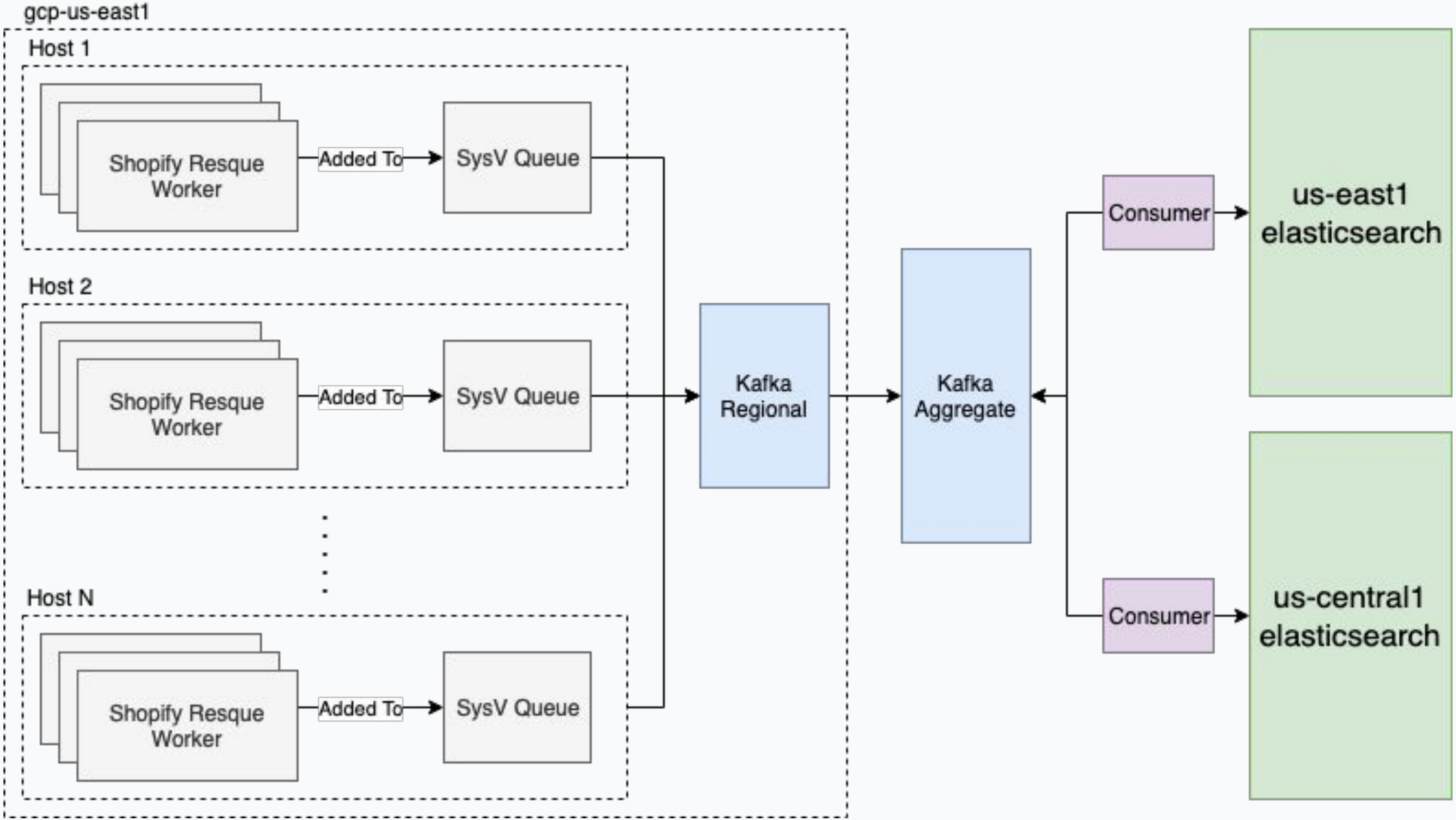
- Jobs can be simpler with respect to Kafka status
- Persistent across container restarts



@ConorLandry_

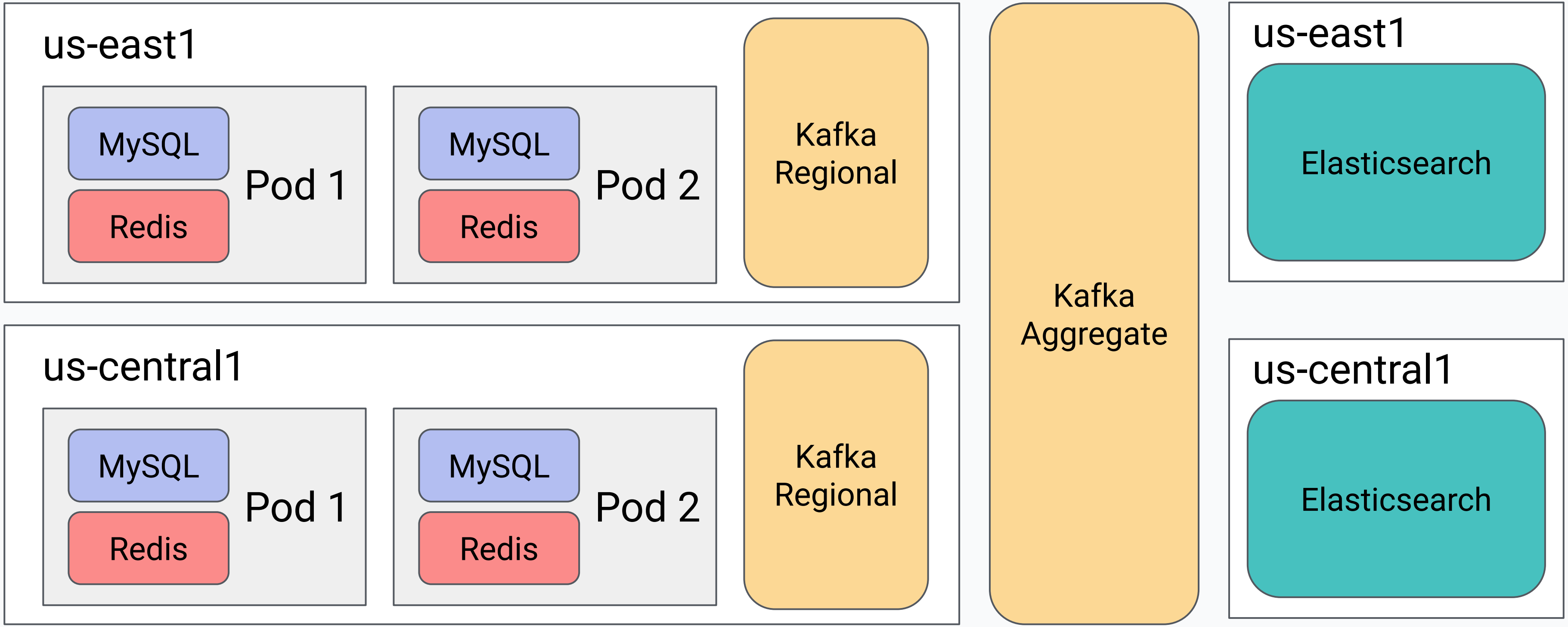
# Why Kafka?

- Resiliency & Persistence
- Implicit Replication
- Offset Committing
- Order guarantees



@ConorLandry_

# Bringing It Together



us-east1

MySQL

Redis

Pod 1

MySQL

Redis

Pod 2

Kafka Regional

us-central1

MySQL

Redis

Pod 1

MySQL

Redis

Pod 2

Kafka Regional

Kafka Aggregate

us-east1

Elasticsearch

us-central1

Elasticsearch

@ConorLandry_

# After Everything Is Reindexed

| Alias | Index |
|---|---|
| products | products.0 |
| products.new | products.1 |

| Alias | Index |
|---|---|
| products | products.1 |
| products.new | |

# What are our future plans?

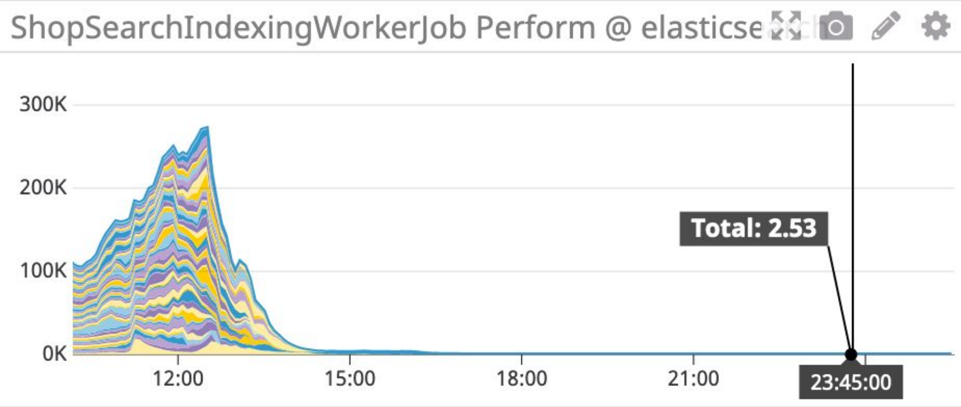
## MySQL Query Optimization

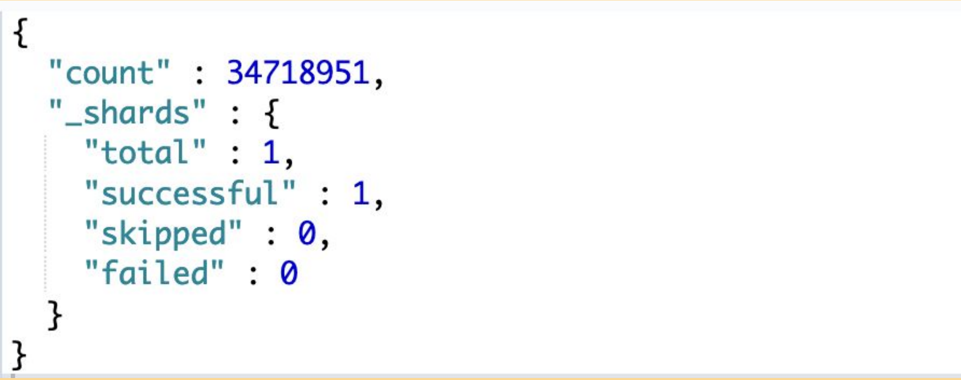
- Normalized -> Denormalized is hard

- Push product teams to think about the impact of complex queries

@ConorLandry_

# Shop Level Reindex Concurrency





```
{
  "count" : 34718951,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  }
}
```

# What We Covered

- Elasticsearch @ Shopify
- Shopify Core Topology
- What is Reindexing?
- How we Reindex at Shopify
- Future Plans & Improvements

# Q & A

conor.landry@shopify.com

shopify.ca/careers