QCon by InfoQ

NETFLIX AI PLATFORM · QCON San Francisco· NOV 2026

Failing in Layers: Lessons from (re)building Netflix 's ML Serving Platform

Rajat Shah
AI Platform, Netflix

Wenbing Bai
AI Platform, Netflix

In 2023, one bad instance exposed what we needed to rebuild

THE CAUSE

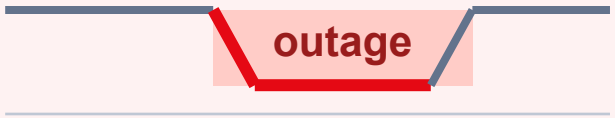
1 Bad Node

A degraded EC2 instance

THE IMPACT

Starts per second

Dropped in one region



THE DURATION

2 Hours

Impact duration

How can one instance do that?

And what did we change?

NETFLIX AI PLATFORM

Talk Outline

OVERVIEW

Three failures, three layers

Each failure hit a different layer: organization, request path, data contract.

01 **ML Surfaces in Netflix Product**

02 **What is a Model at Netflix?**

03 **Failure #1: Decentralized Serving (an Organizational Failure)**

04 **Failure #2: Keep the Hot Path Thin**
First centralized design: a member-impacting incident

05 **Failure #3: Corrupted GPU Scores**
Deep-dive investigation and the multi-day recovery effort

06 **Async Model Serving**

07 **Failure #4: Cold start becomes the bottleneck**

08 **Failure #5: Workflow Overhead is unacceptable for large scale**

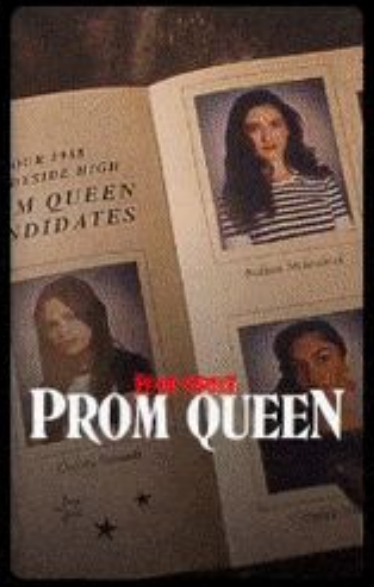
Home Shows Movies Games My Netflix

ML

Today's Top Picks For You



ML







ML

Drama · 2025 · 2h 21m · PG-13

Benoit Blanc returns for his most dangerous case yet. Sequel to 'Glass Onion (2022)'.

Only on Netflix

Model is a workflow

At Netflix, a model is a workflow, not just weights

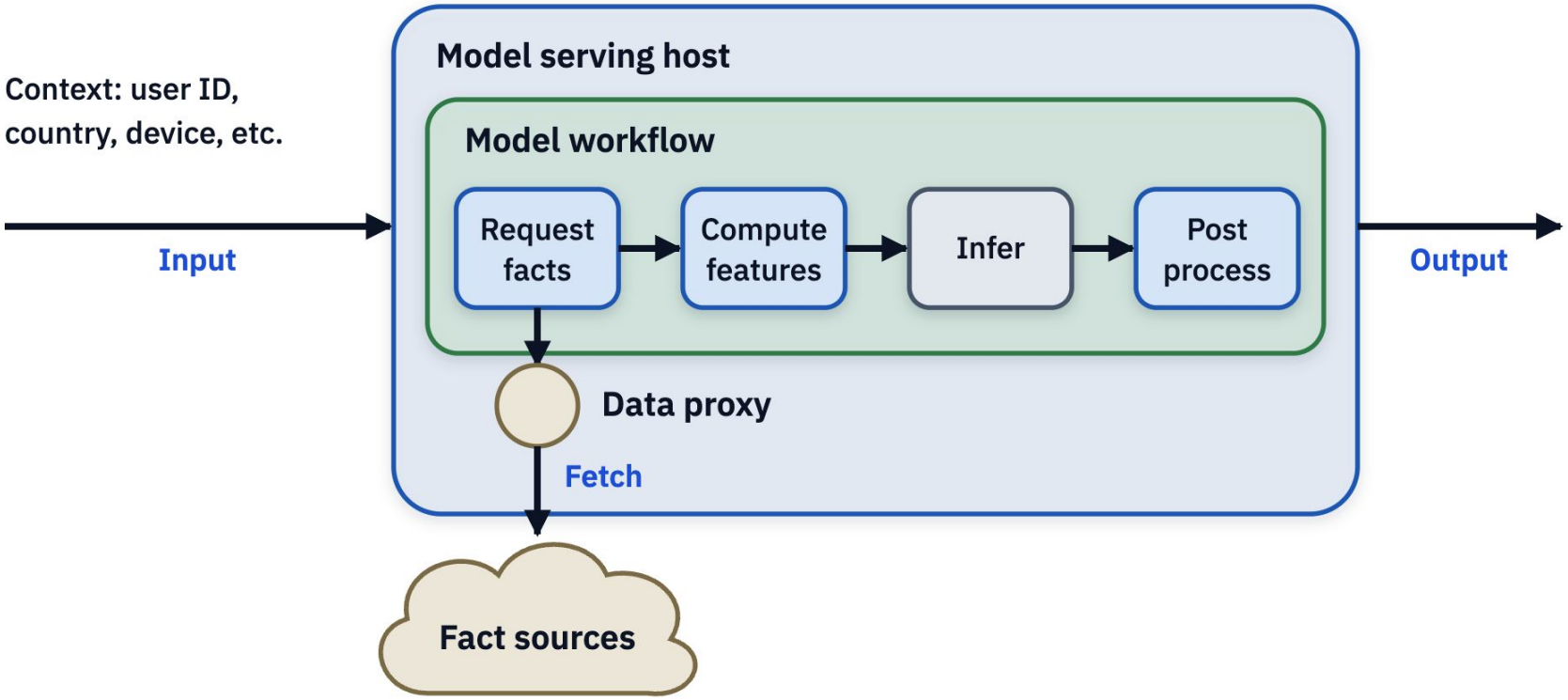


Image from State of Routing in Model Serving | by Netflix Technology Blog

Architecture

How the model shapes serving

1. Surrounding Business Logic

User code and data hints ship with the model

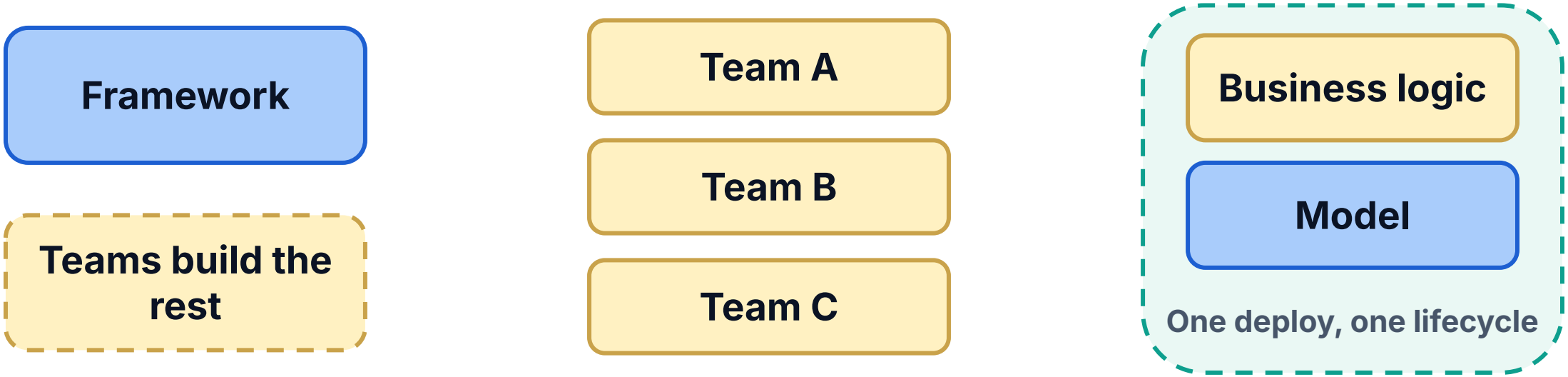
2. Diverse Business Logic & I/O

Recommendations and fraud look very different

3. Universal & Scalable Platform

One platform serves them all, at scale

Failure #1: every team built its own serving stack



1 Framework, not a platform

2 Inconsistent to run

3 Logic and model coupled

***Conway's Law**: systems copy how their builders communicate.*

Defining Principles

Overcoming Failure #1

1. One Central ML Serving Platform

2. One API

3. One Routing Layer

4. One Operational Model

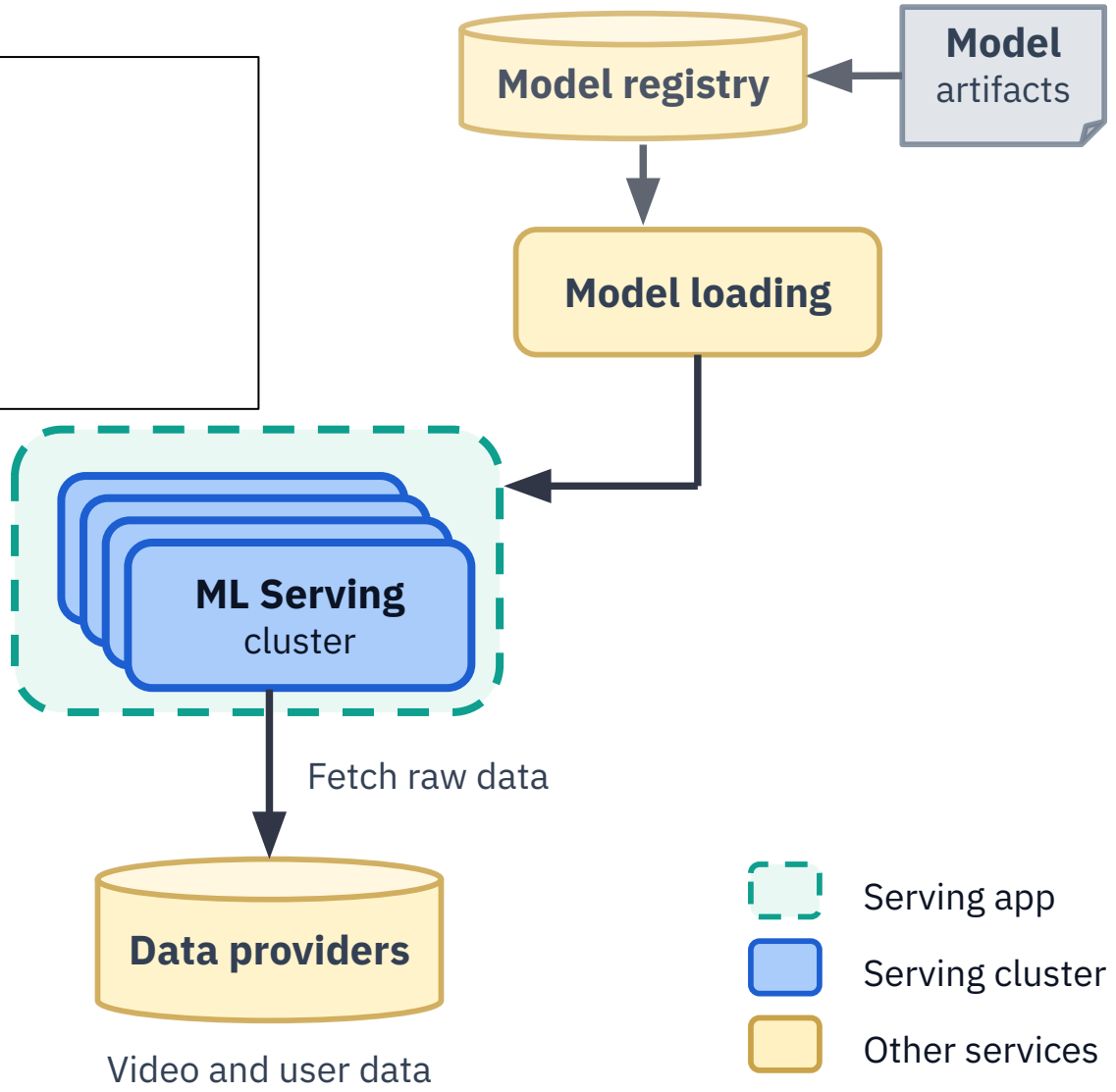
Key Components on the Sync ML Serving System

Built for low latency.

Model Serving App

ML Serving cluster

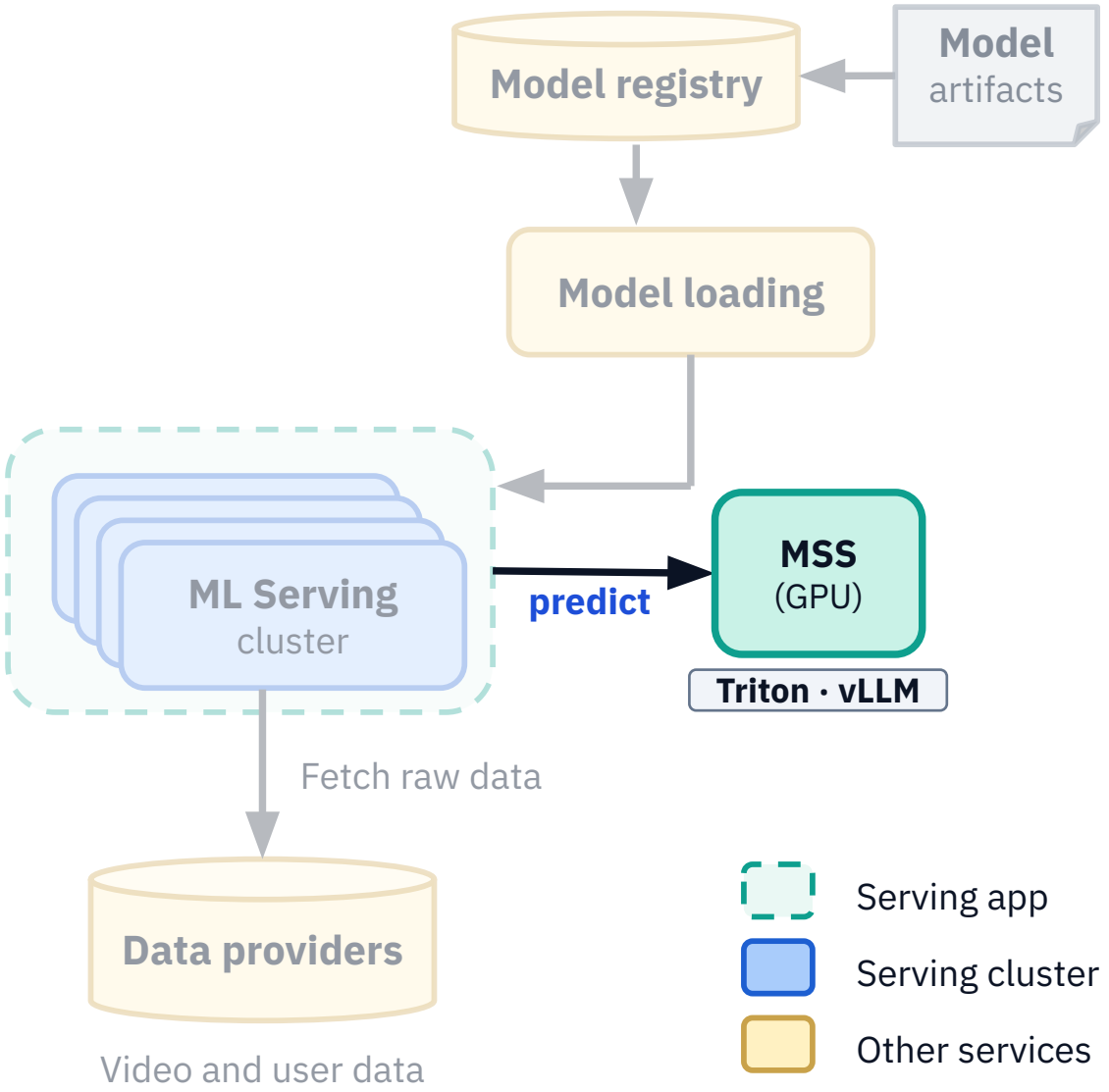
- A JVM Spring Boot gRPC service
- Loads the models a request needs
- Fetches raw data, turns it into features
- Runs on-box inference via JNI bridge
- Logs for training and observability



Model Serving App

Model Scoring Service (MSS)

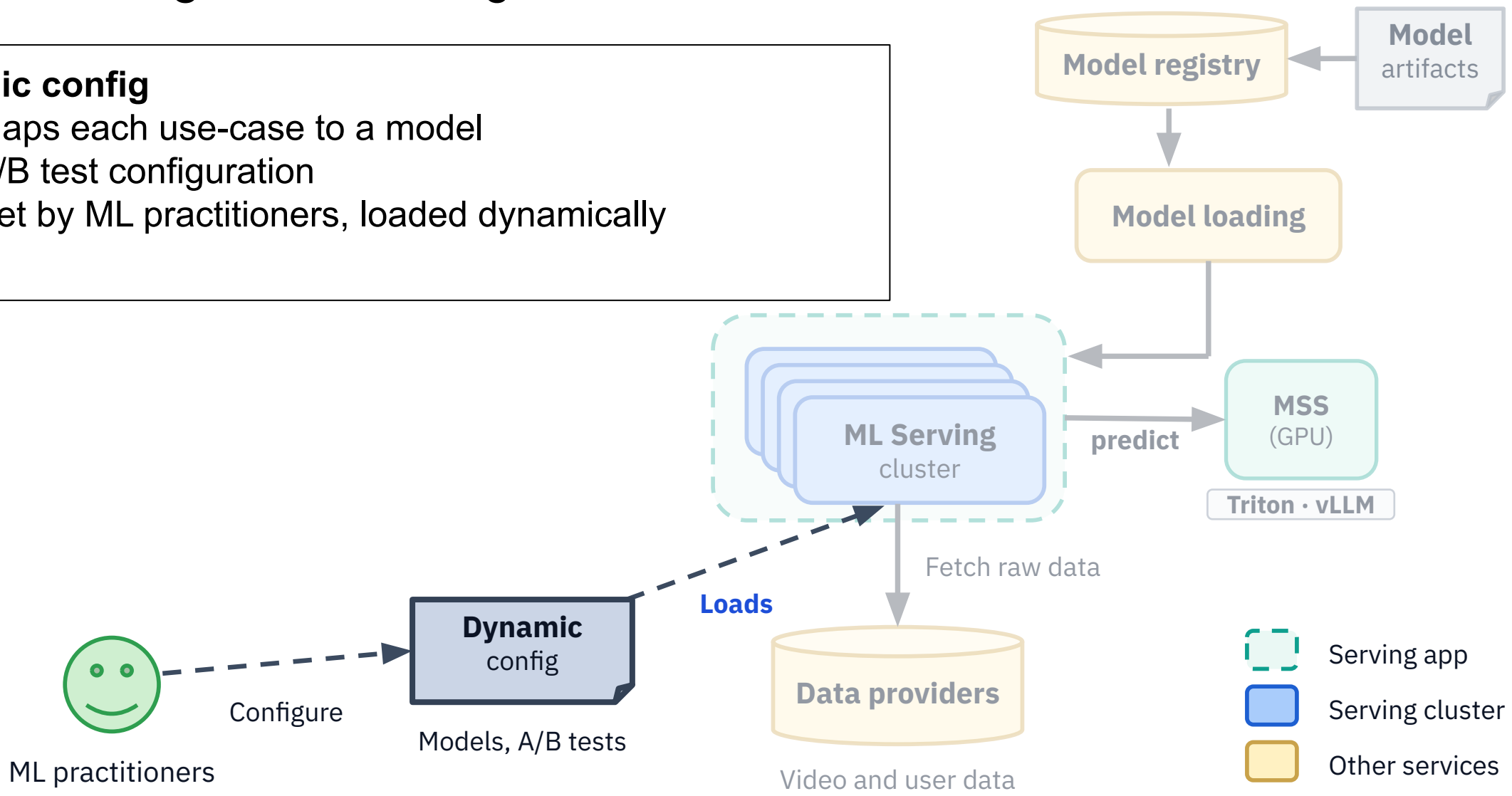
- For efficient off-box GPU inference
- Also used for LLM serving



Dynamic config: how models get chosen

Dynamic config

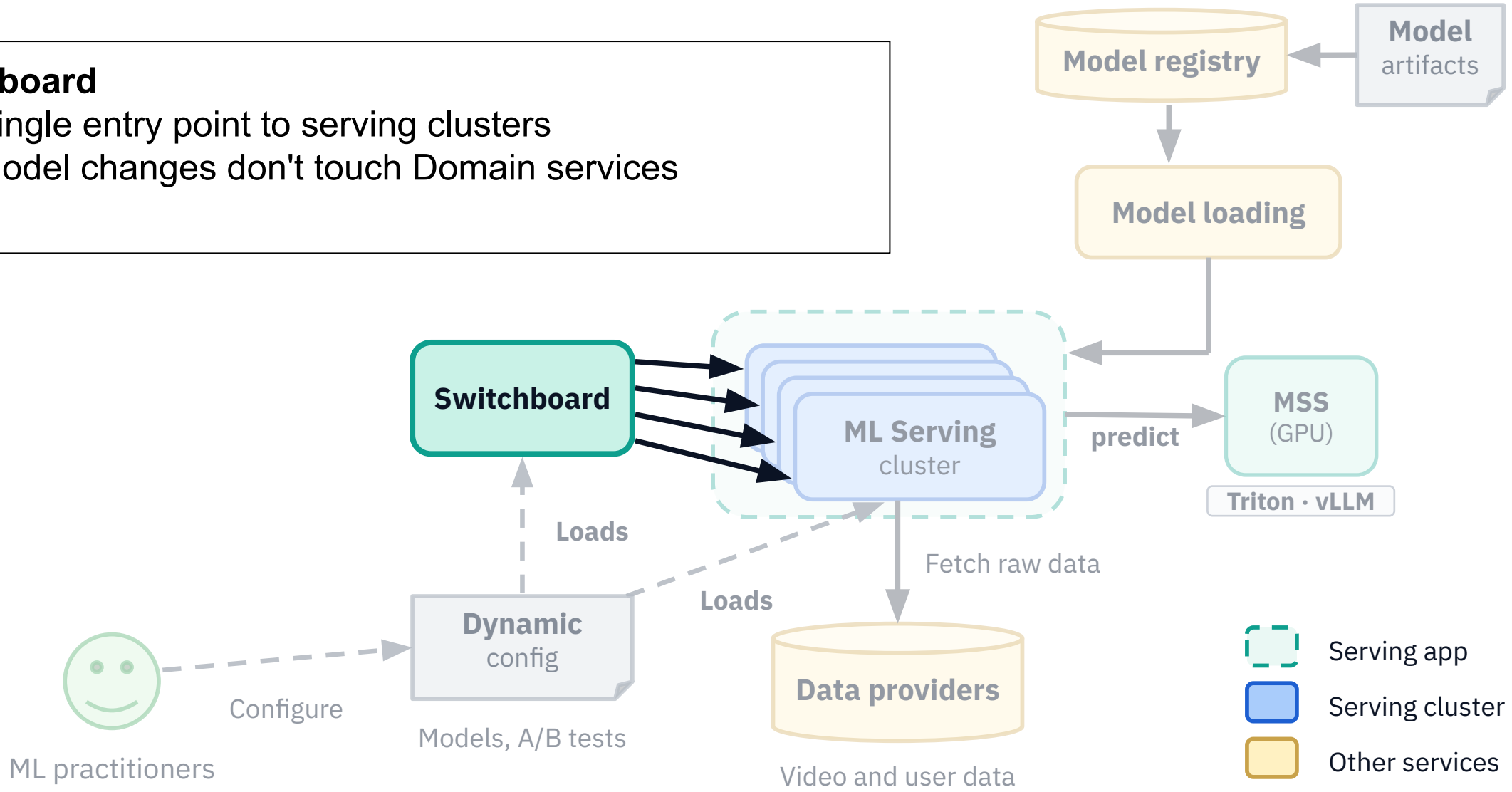
- Maps each use-case to a model
- A/B test configuration
- Set by ML practitioners, loaded dynamically



Switchboard: Routing and service abstraction

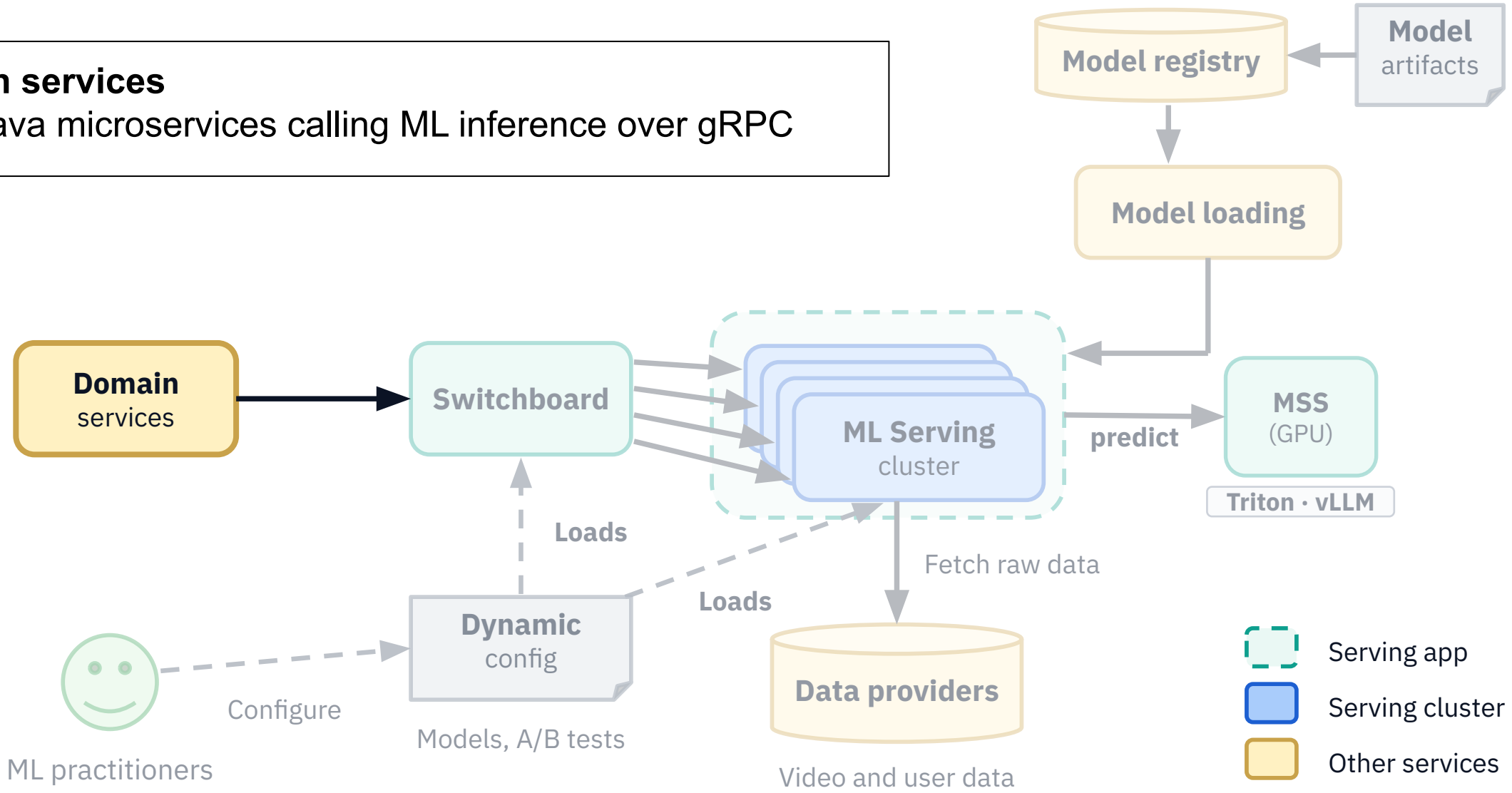
Switchboard

- Single entry point to serving clusters
- Model changes don't touch Domain services

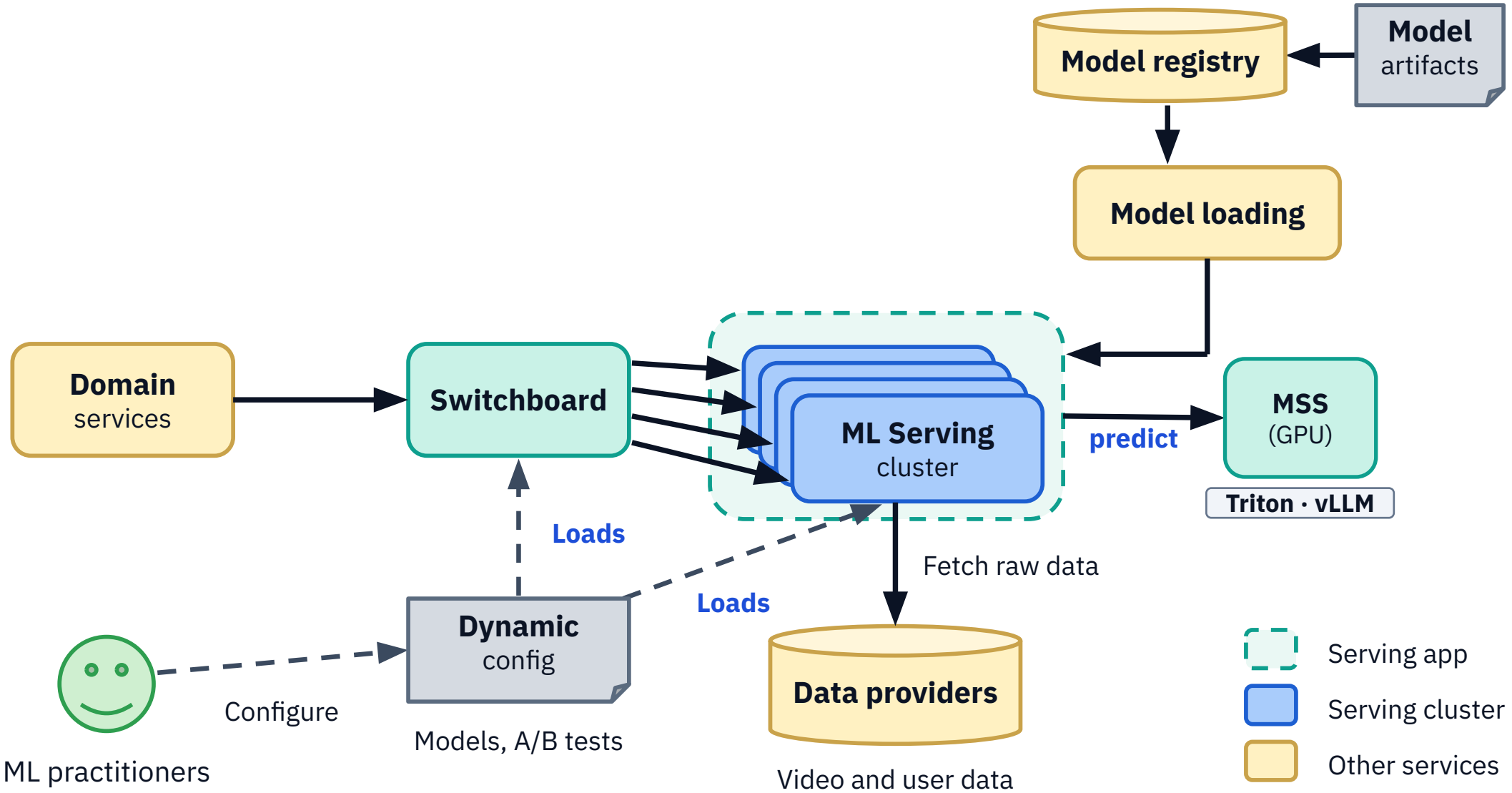


Caller: Domain services

Domain services
- Java microservices calling ML inference over gRPC



ML Serving Platform: High-level architecture



System Complexity

1000+

Prod Clusters

Tier 0 clusters for ML critical workload

~1M+

Requests Per Second

High-throughput serving ~1 Million+ requests/sec

99.99%

Availability

Platform availability SLA for high resilience

300+

Models Served

Types supported: TensorFlow, XGBoost, PyTorch, and more

10ms – 1.5s

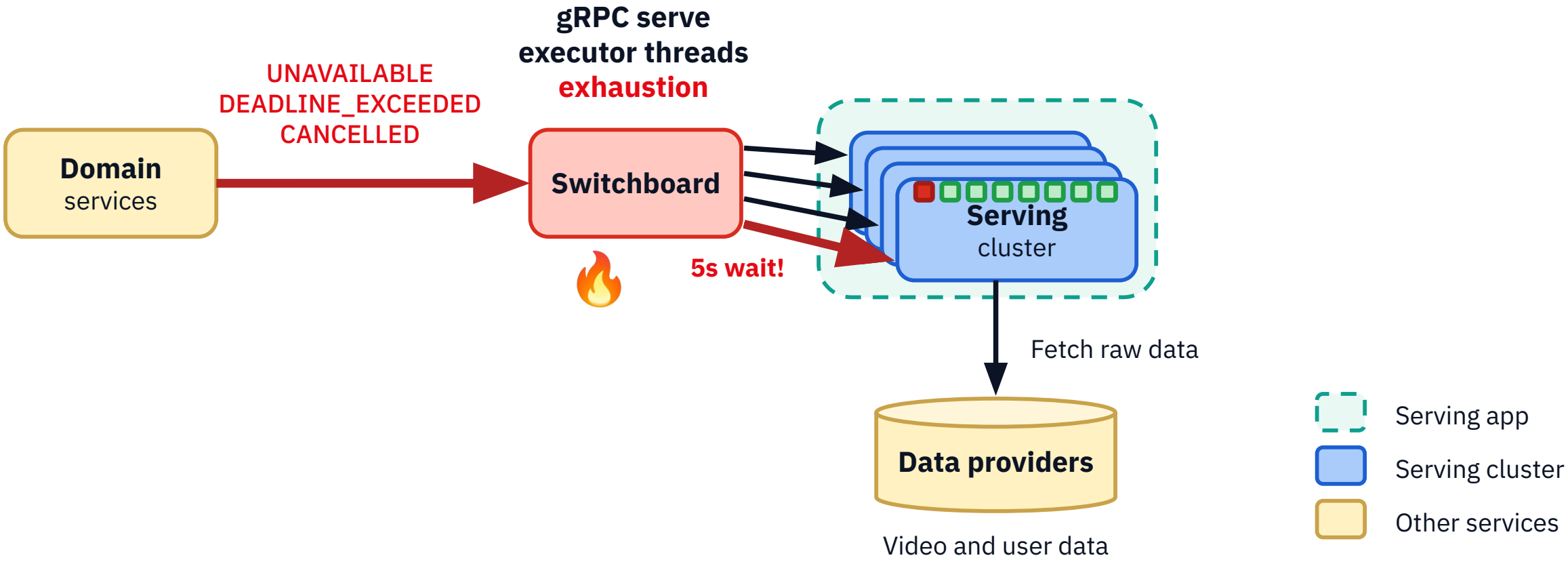
Supported Latencies

Range of latencies supported: ~[10ms–1.5s] @ p95

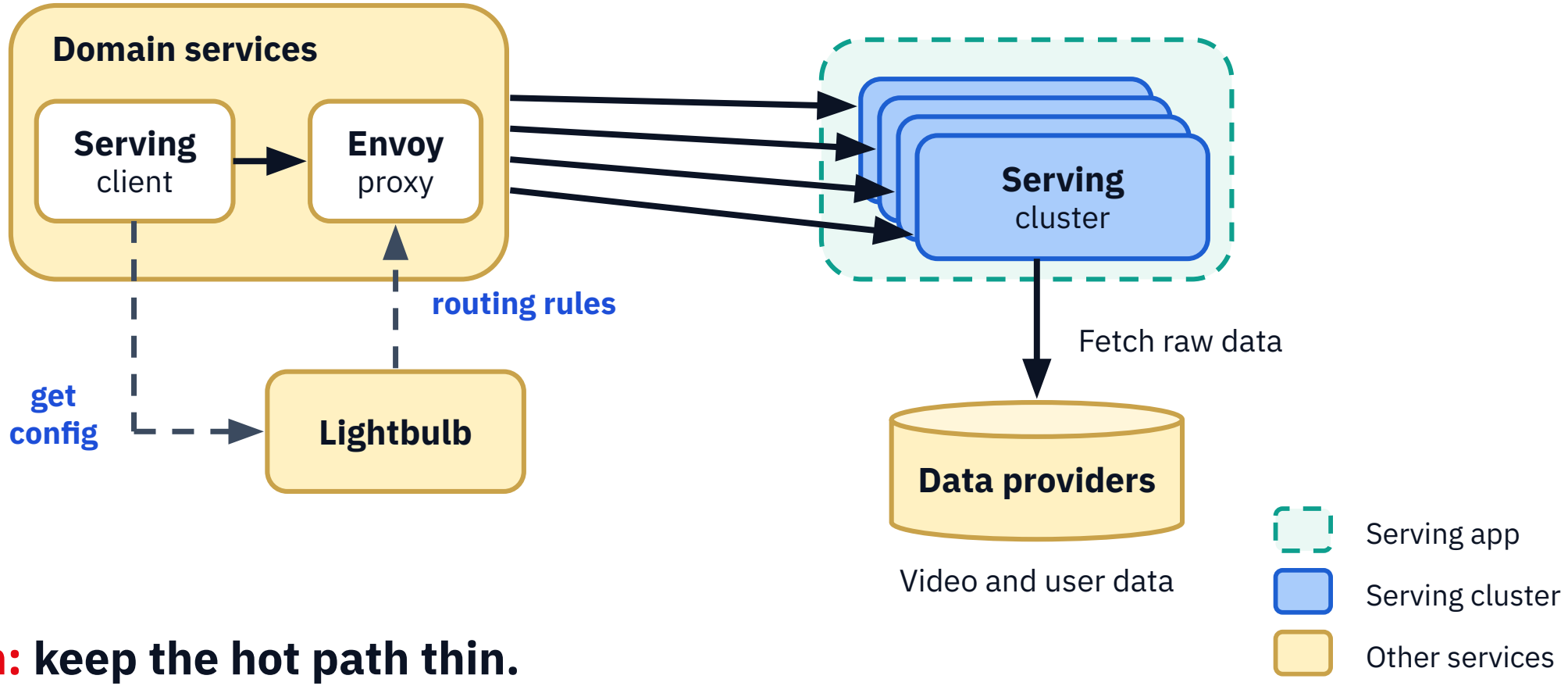
It worked for two years.

Then one Saturday afternoon,
I got paged.

Failure #2: Everything flowed through one service, and it failed



We took routing off the request path: Envoy routes, Lightbulb supplies the metadata



Lesson: keep the hot path thin.

Image from [State of Routing in Model Serving | by Netflix Technology Blog]

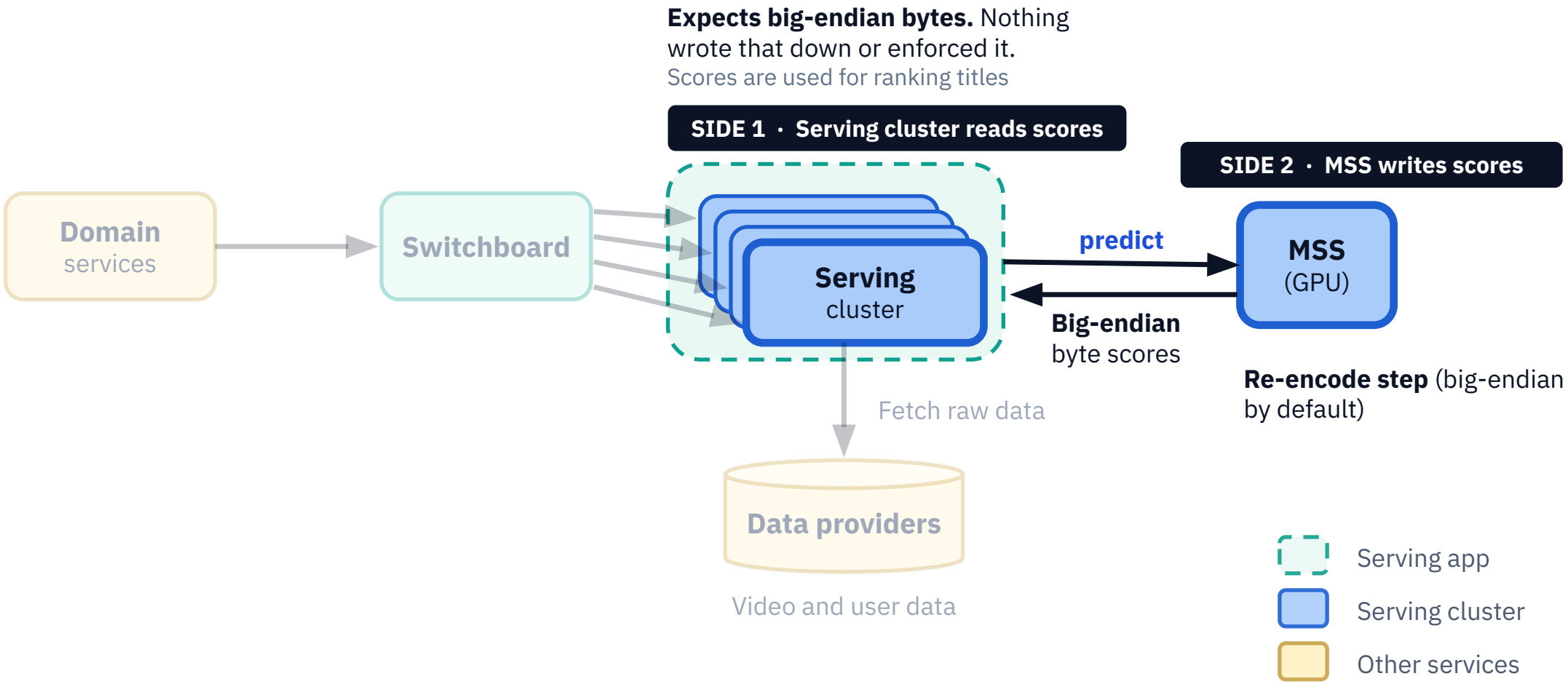
The silver lining

It failed loudly!

Not every failure is loud. **The silent ones are worse.**

For years, a hidden step made MSS and the serving cluster agree

Byte order: how a 4-byte number is laid out in memory.

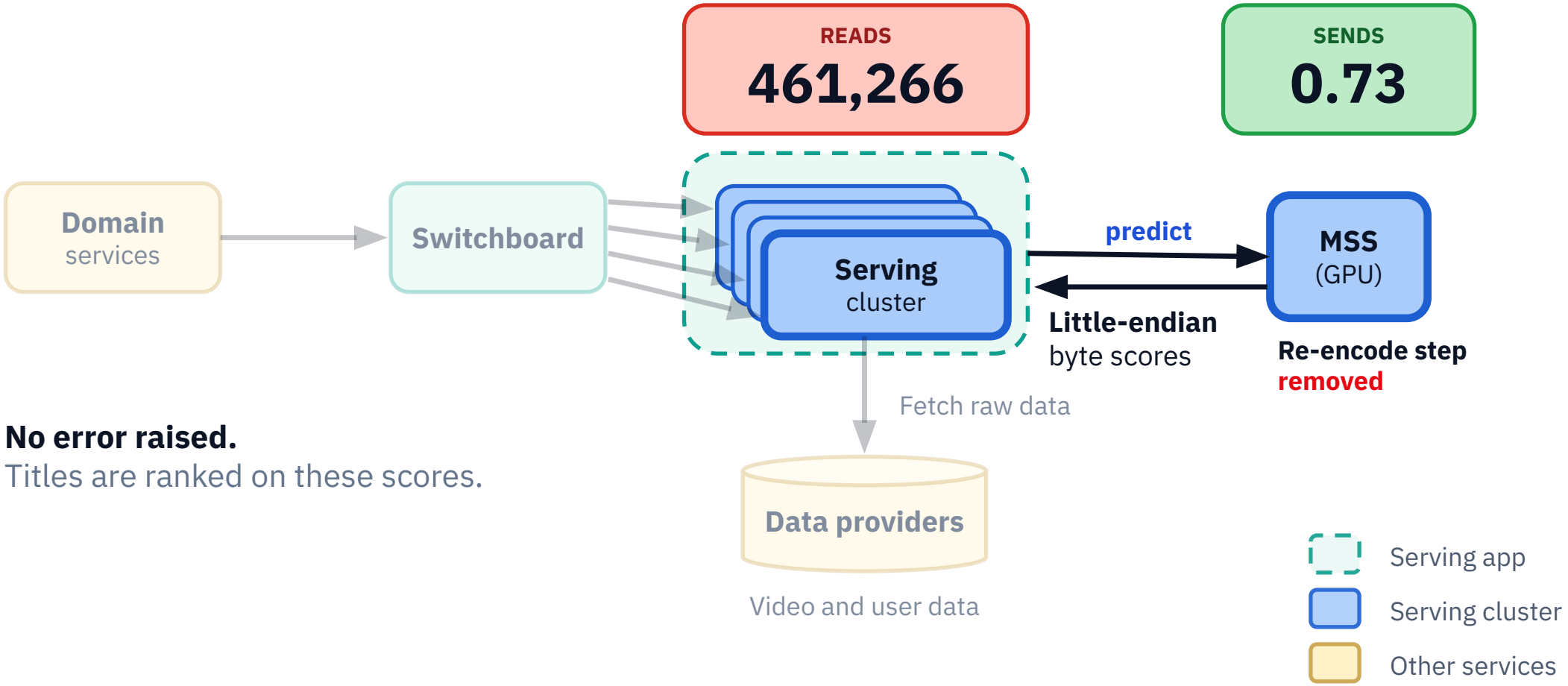


Then a routine performance optimization shipped

It removed the re-encode step.

Intended Goal: less copying, better tail latency.

Failure #3: GPU score corruption



No error raised.
Titles are ranked on these scores.

Failure #3: Takeaways

01 / CONTRACTS

Enforce Unwritten Agreements

Find and enforce assumptions between services.

02 / MONITORING

Validate Model Output Ranges

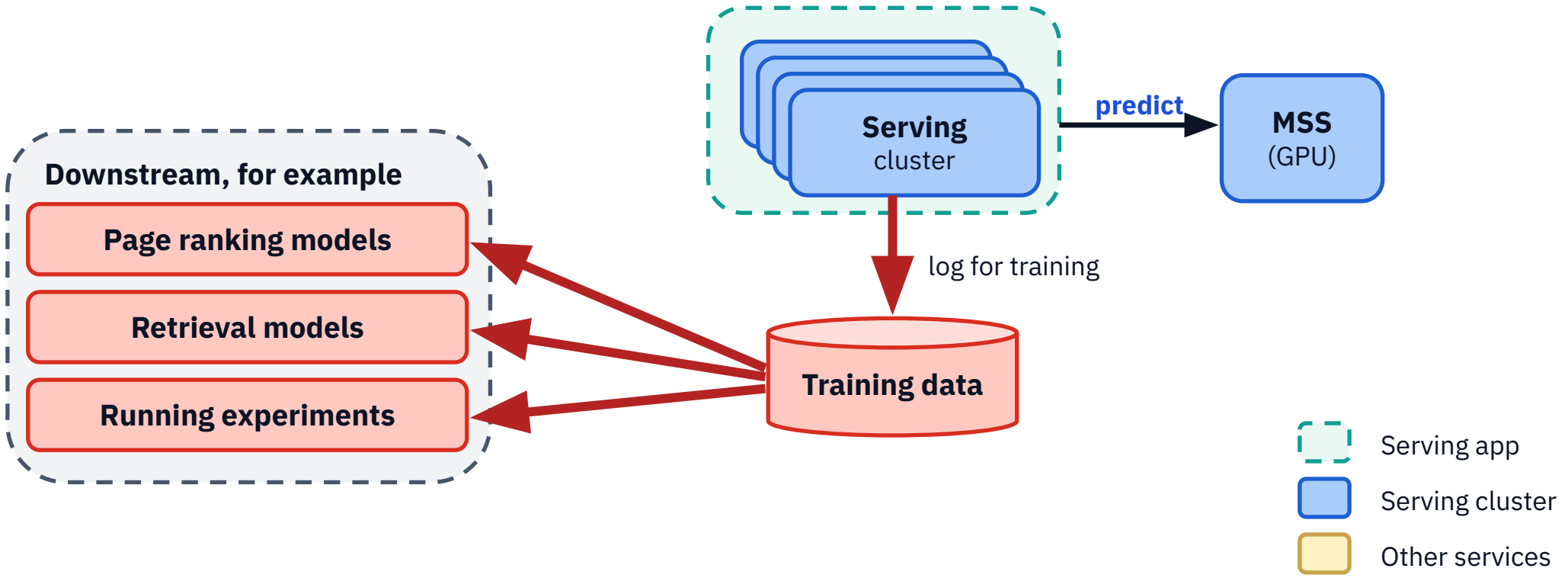
Alert when scores leave their expected range.

03 / VERIFICATION

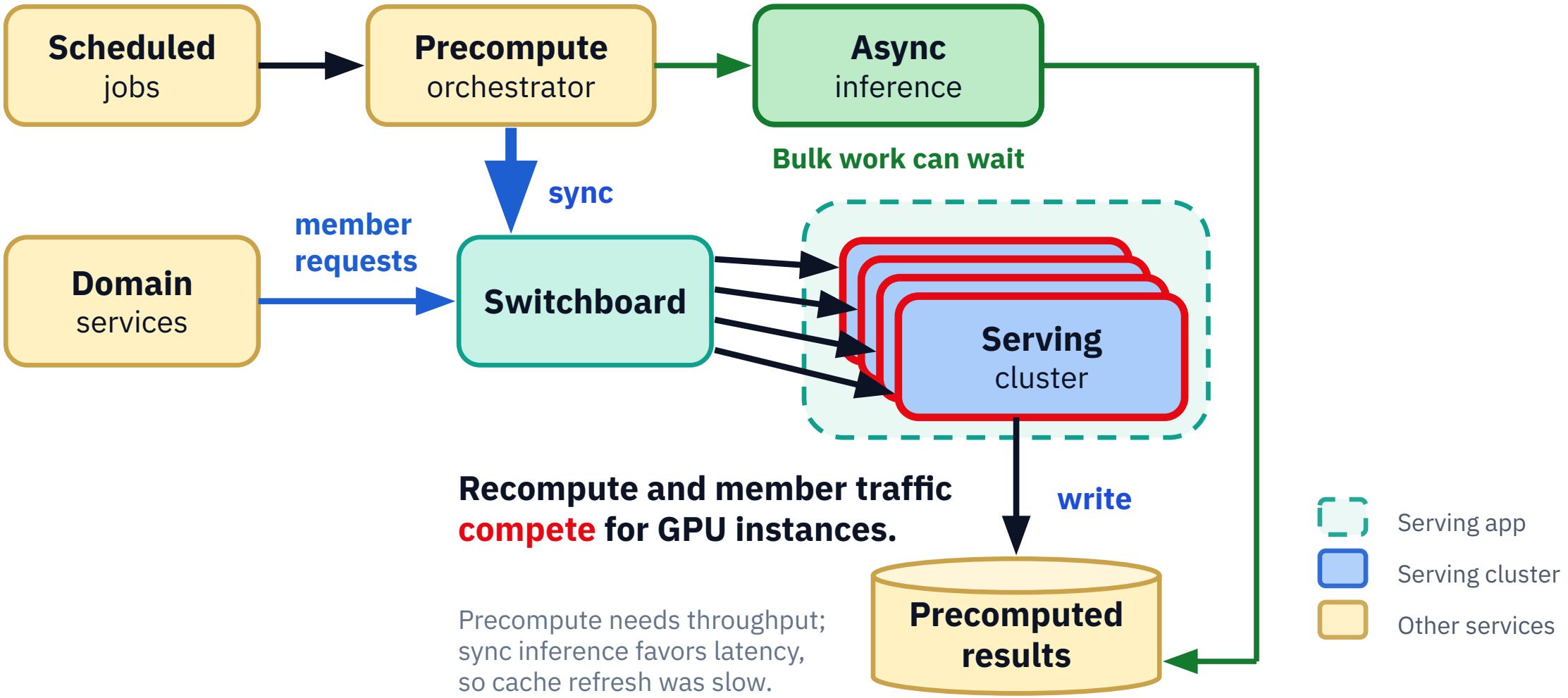
Healthy ≠ Correct

HTTP 200 doesn’t mean correct scores.

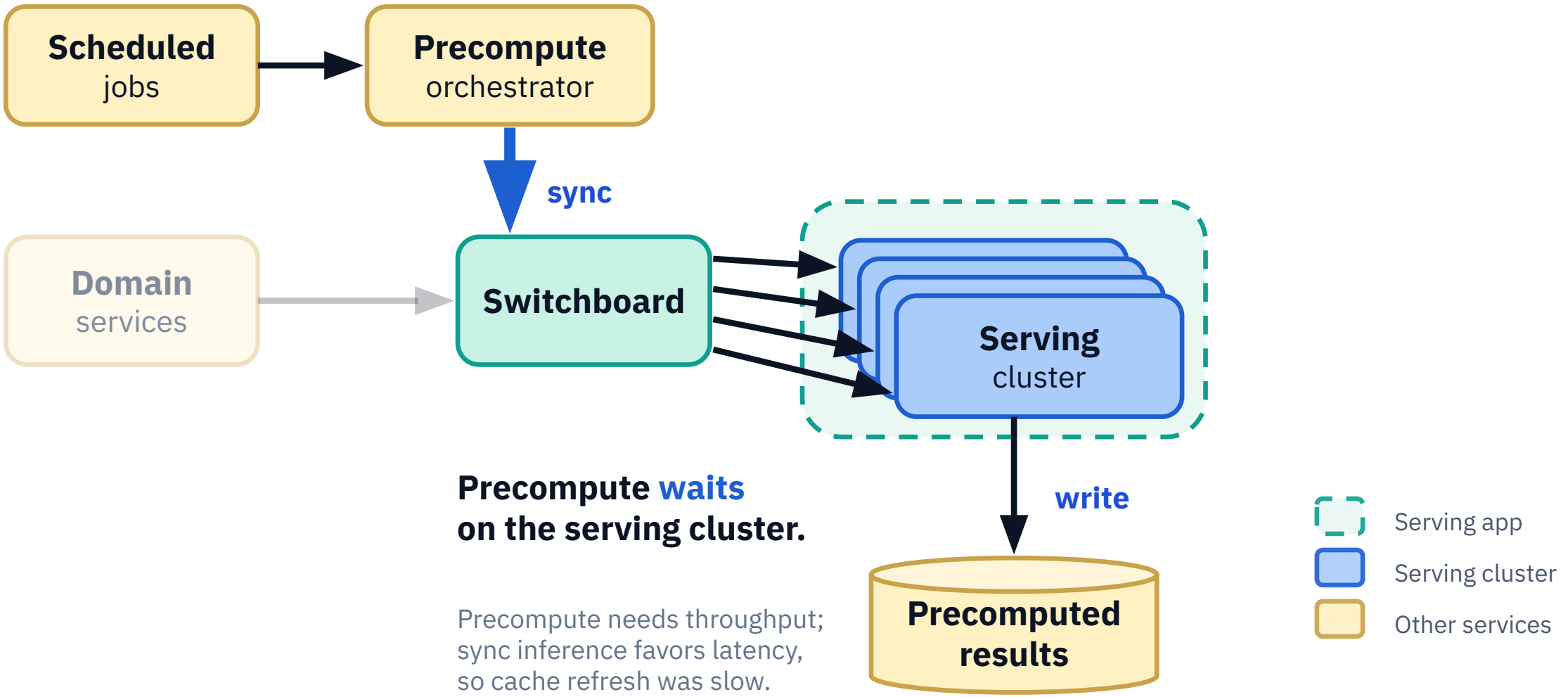
Failure Aftermath #1: Other models and experiments consumed the bad scores



Failure Aftermath #2: Recompute competed for GPU capacity



Failure Aftermath #2: Precompute workload design



Async Model Serving

Built for throughput, not latency.

Sync inference buys low latency with idle capacity; async buys throughput

	Sync (Realtime) Inference	Async Inference
Call Pattern	Request → response Blocking: Caller waits for result to proceed	Event-driven, cron-based, Bulk-batch Non-blocking
Workload Duration	Typically milliseconds.	Typically seconds to minutes to hours.
Latency	< 100ms P99 Tight end-user-facing SLOs	Operational workflow based SLOs, per model
Throughput	Low-medium One prediction per request typically	High; Batched & parallelized
Compute	Always-on dedicated compute Pay for idle capacity	Scale-to-zero; Shared compute, priorities
On Failure	Client retry No completion guarantees	Durable; step-level retry Guaranteed completion

Async inference runs ranking, speech, artwork, and embeddings

PERSONALIZATION RANKING



Member personalization

Ranking scored offline in large batches for Recommendations

SPEECH & LOCALIZATION



Localization

Automatic Speech recognition and machine translation

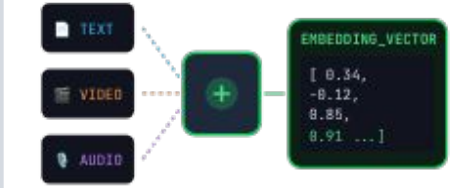
ARTWORK CANDIDATES



Artwork Candidates

Rank & Score artwork imagery for each title.

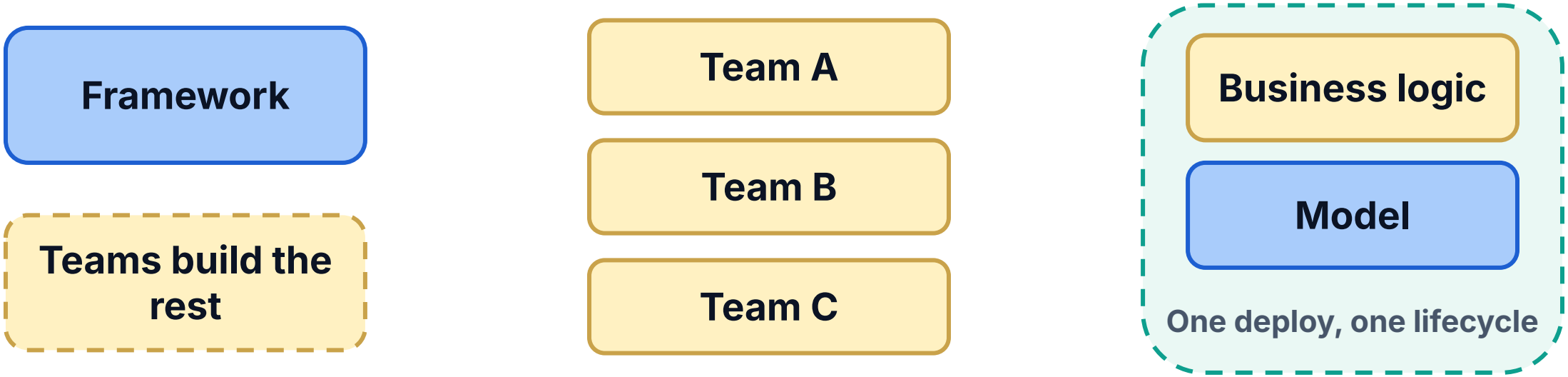
CONTENT EMBEDDINGS



Content Understanding

Multimodal models that turn content into embeddings

Failure #1 (same as sync serving): every team built its own serving stack



1 Framework, not a platform

2 Inconsistent to run

3 Logic and model coupled

***Conway's Law**: systems copy how their builders communicate.*

Defining Principles

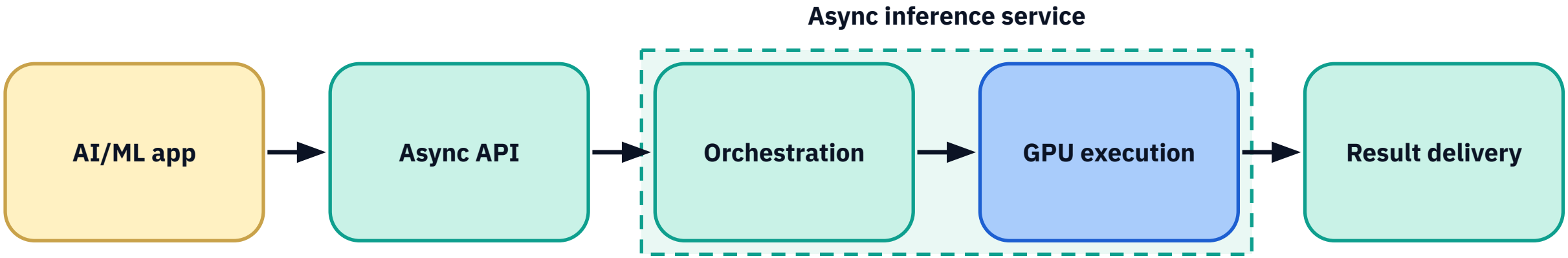
Overcoming Failure #1

1. One platform for all async workloads

2. Decoupled request lifecycle

3. Distributed GPU workers

Async inference runs as submit, orchestrate, execute, deliver



Failure Mode

Failure #4

Cold Start

1. Setup on every job

Each job starts a container, pulls model weights, and only then runs inference.

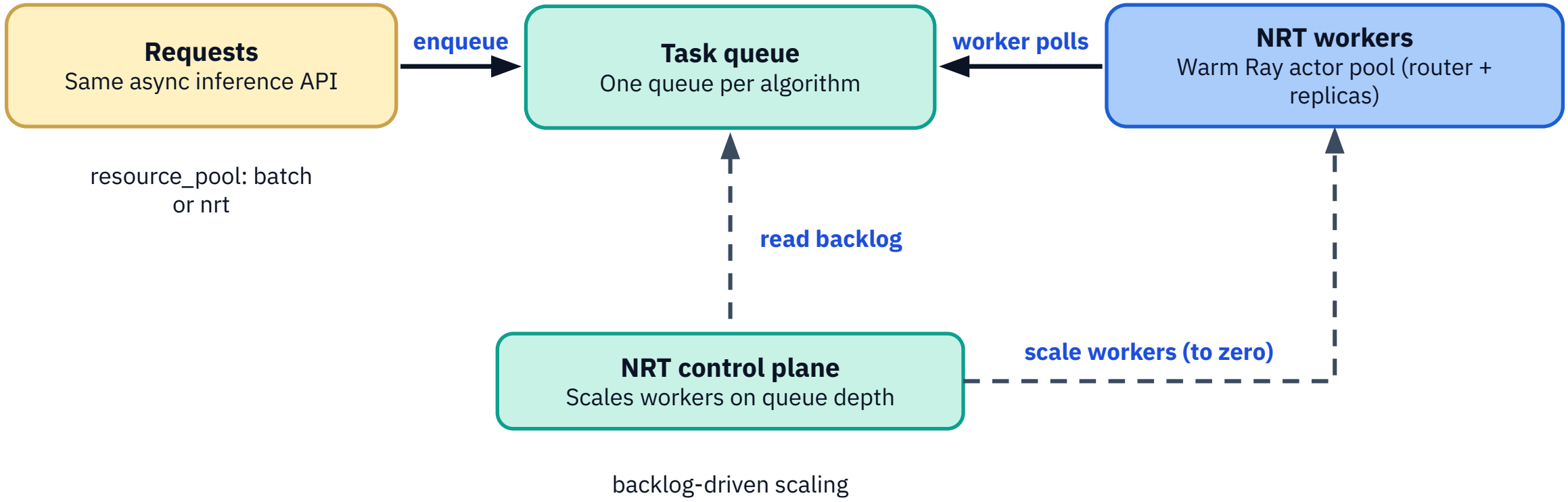
2. Caching only shrinks it

Model weight caching, image caching and a planned virtual-environment cache cut the wait, but cold start is still not zero.

3. Interactive GenAI can’t wait

A person generating images or video needs zero cold start.

Near-Real-Time keeps the async API and swaps in a warm worker pool



Inference Runtime (Diffusion Transformer models): vLLM-Omni serves every modality through one API

THE SCOPE

Omni-modal

Text, image, audio, and video in and out

THE API

One API

Know vLLM, know vLLM-Omni

THE SPEED

Diffusion

Caching, parallelism, and quantization built in

vLLM-Omni also runs outside NRT

Inference Runtime (LLMs): First-class vLLM support cuts serving code and raises GPU use

THE CODE

Less code

Add generation or embeddings with one call

THE GPUS

Better use

Concurrent prompts let vLLM batch continuously

THE SCALING

Simpler

Replicas and multi-GPU without placement logic

Inference Runtime (LLMs) cont'd: First-class vLLM support adds flexibility and visibility

THE LOADING

Flexible

Hub IDs, local weights, registry URIs, or object-store streaming

THE OPTIONS

Custom

Engine and sampling options, templates, structured output, LoRA

THE METRICS

Visible

Latency, token counts, finish reasons, and engine failures

Failure Mode

Failure #5

Workflow Overhead at Member Scale

1. Cost grows with every request

Each request pays for a workflow and its queues, which is too costly at member scale.

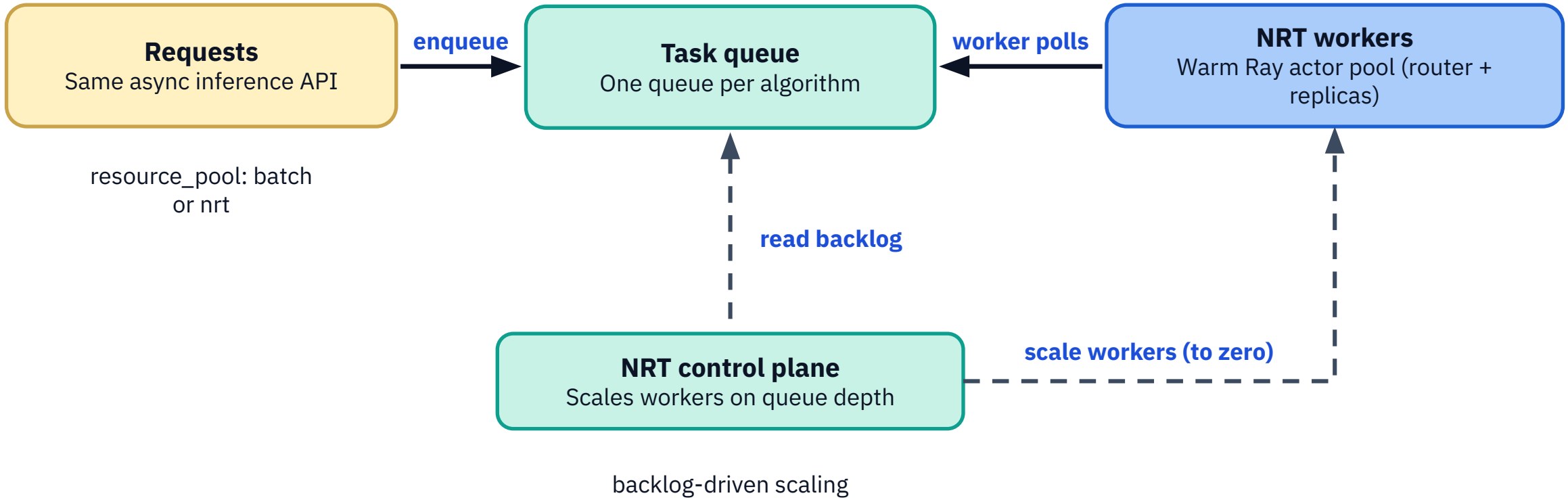
2. Scheduling outweighs inference

The wait from client to worker start is far longer than the model inference itself.

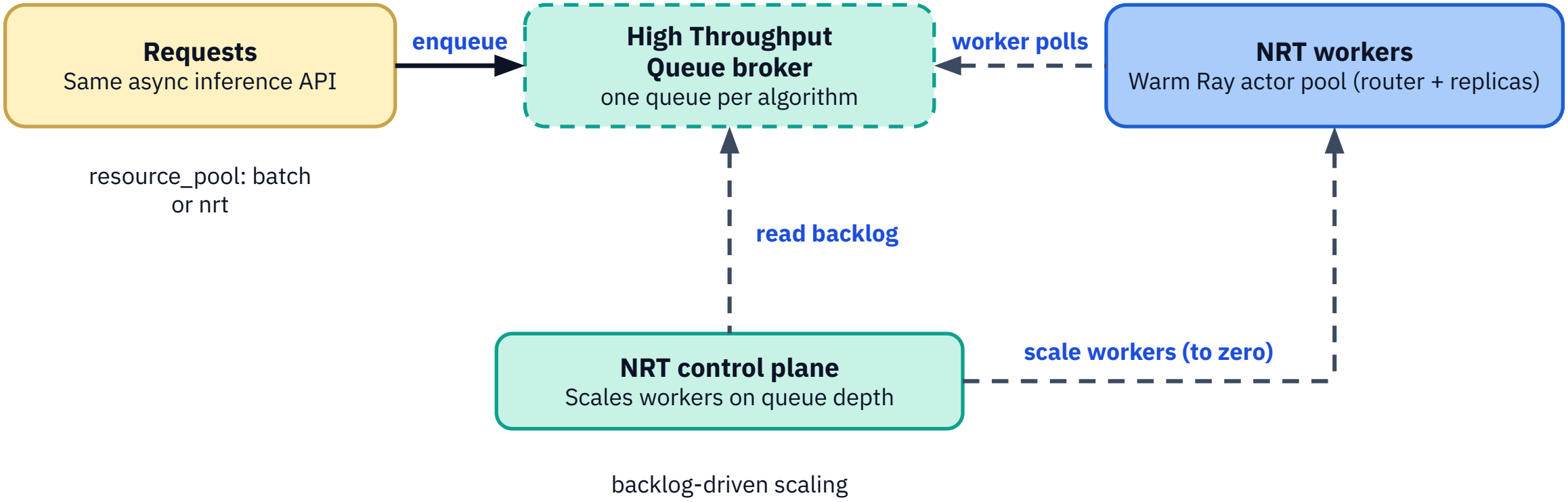
3. Throughput falls short

The workflow engine sustains far fewer requests per second than member traffic needs.

Near-Real-Time keeps the async API and swaps in a warm worker pool



A queue broker feeds warm NRT workers



System Complexity

100+

Prod Clusters

ML workflows on Media artifacts and member-scale offline inference

~10k

Requests Per Second

Peak throughput of the async path

99%

Availability

Availability of the async platform

60~70

Models Served

LLMs, diffusion models, PyTorch-based and more

Seconds to Hours

Supported Latencies

Range of latencies the async path supports

Takeaways

What failure taught us

1. One platform, not a framework
2. Keep the hot path thin
3. Enforce the contracts
4. Match serving to the workload
5. Loud beats silent

Thank you

Questions?

Speakers



Rajat Shah

Staff Software Engineer

AI Platform, Netflix



Wenbing Bai

Senior Software Engineer

AI Platform, Netflix