Monolith-ifying perfectly good microservices

Monolith-ifying perfectly good microservices

happens

"Why don't we just..."

Intercom is kind of

BIG

(for a Ruby on Rails app)

Commits

7c242879c2

Commits on Jan 9, 2011

initial commit

 **wal** committed on Jan 9, 2011

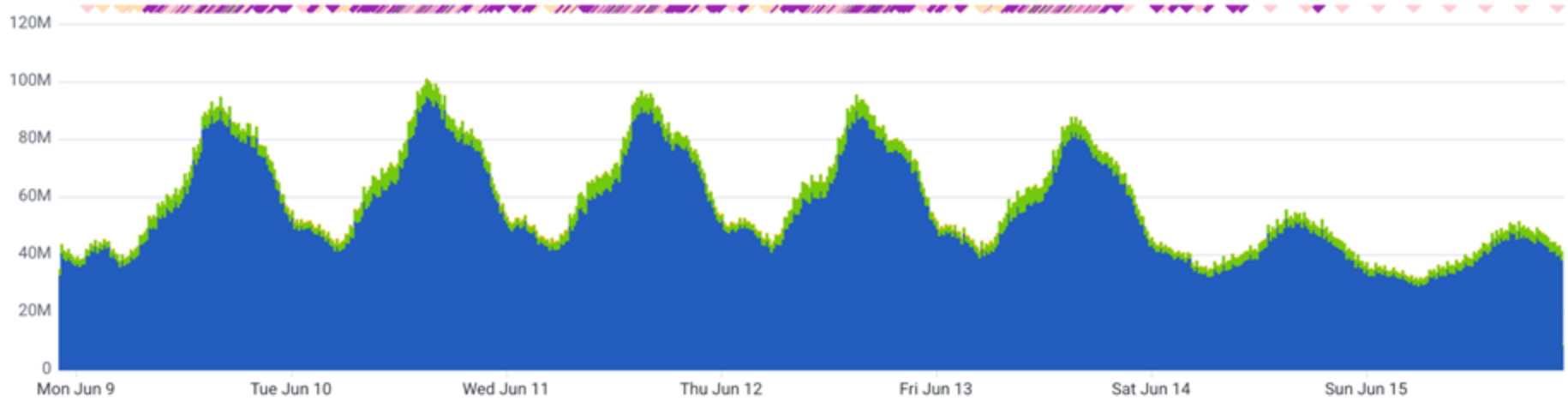
Query Results

Last 7 days (run just now)

Jun 8 2025 22:03:14 – Jun 15 2025 22:03:14 UTC+01:00 (Granularity: 15 min)

COUNT

Modify chart New



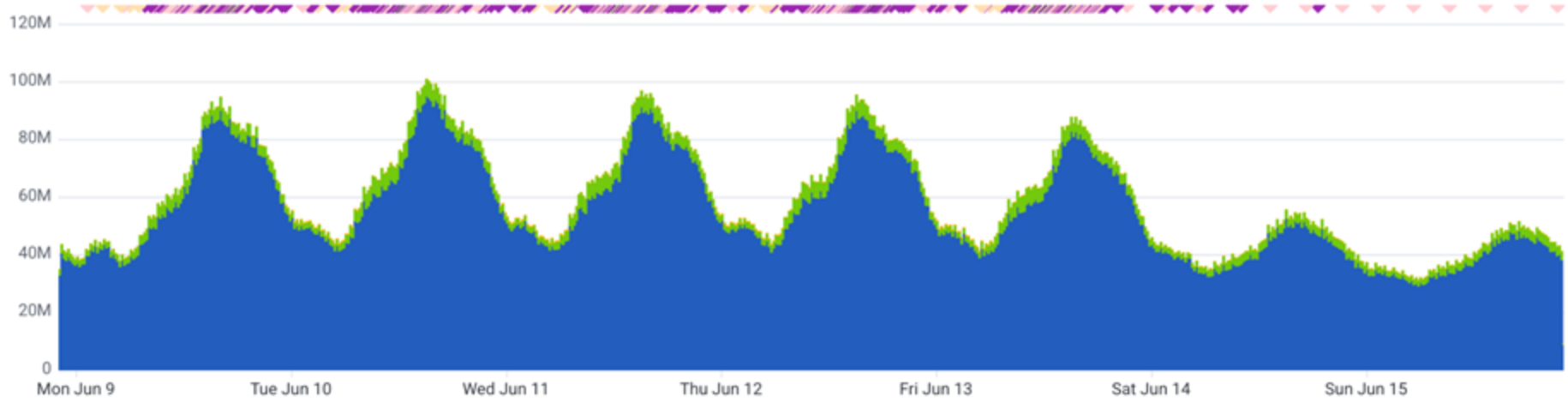
Query Results

Last 7 days (run just now)

Jun 8 2025 22:03:14 – Jun 15 2025 22:03:14 UTC+01:00 (Granularity: 15 min)

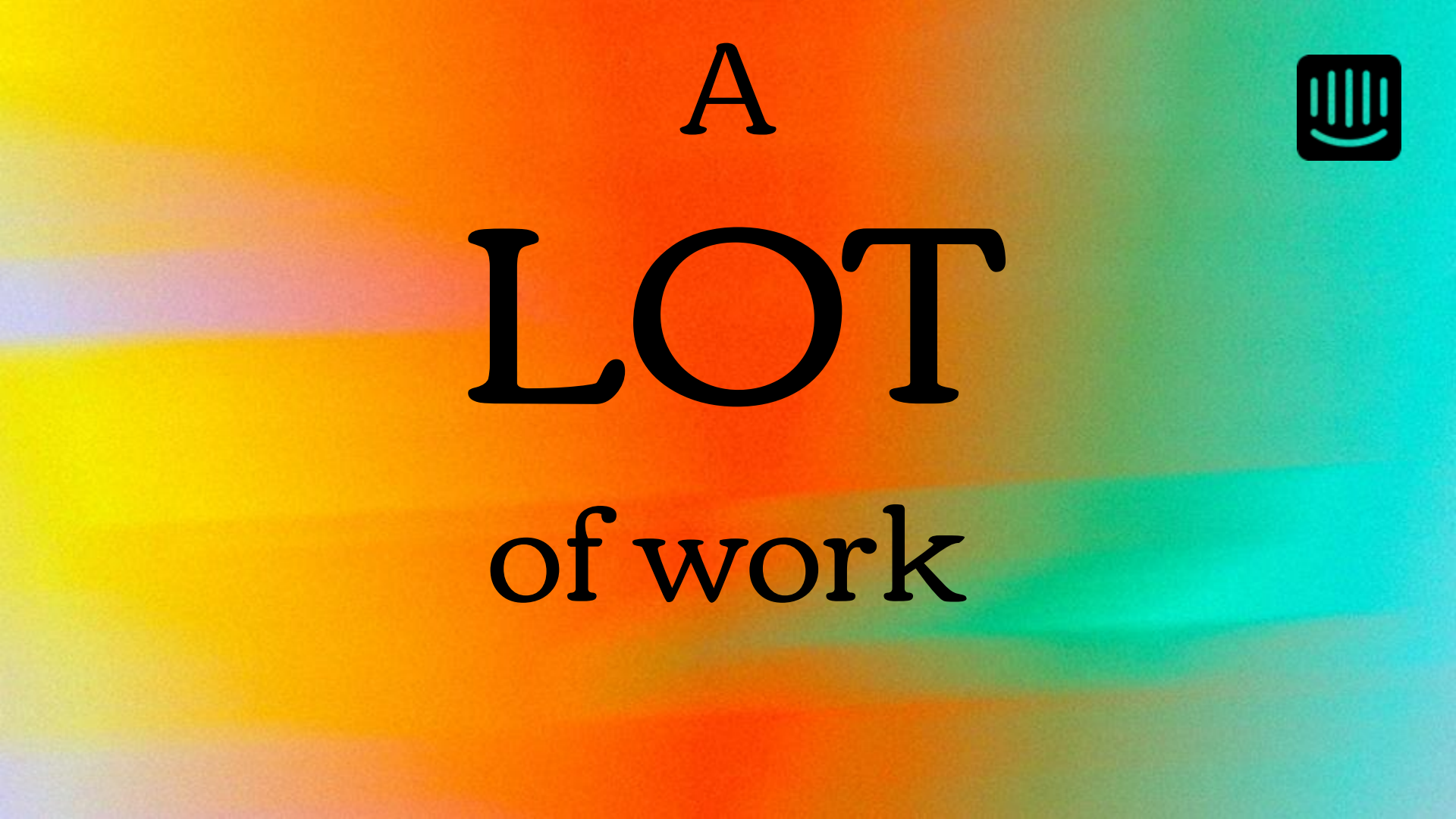
COUNT

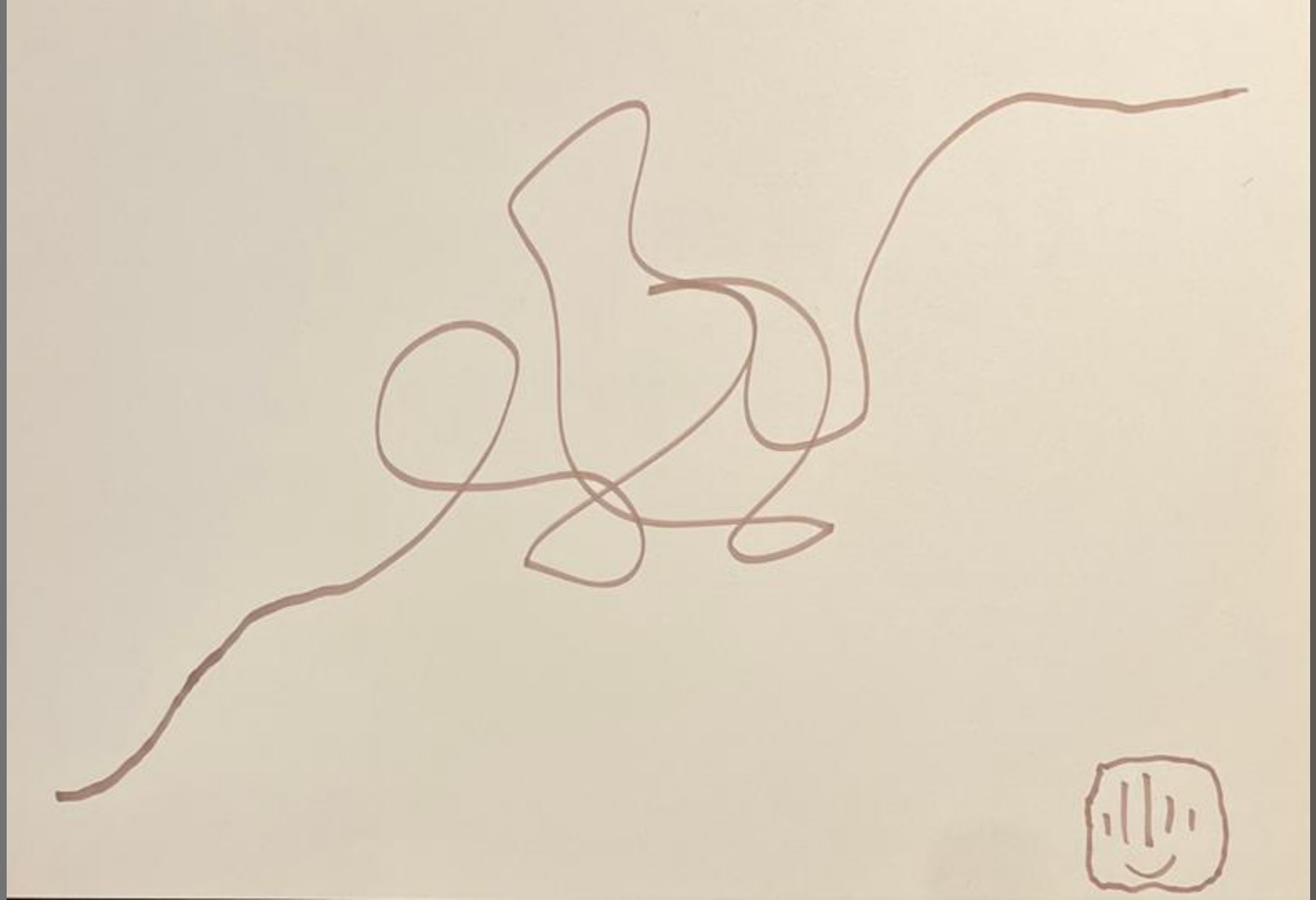
Modify chart New



Thank you for coming to my talk!

 scanlanb


A
LOT
of work



The Three Acts of Intercom's architecture

Act 1.

The Setup.

Act 1.

The Setup.

(Insert rocket-ship emoji here)

Act 2.

The Confrontation.

We started building new features in standalone applications

We extracted billing into its own Rails app

- ★ Splitting Intercom and Billing allows for easier development of both independently
- ★ Billing is a large independent process that doesn't need to reside in the Intercom monorepo
- ★ Now an internal-only microservice deployed by Muster
- ★ Speed up Intercom CI times by removing ~1,800 unit tests
- ★ Billing system uptime no longer directly linked to Intercom's uptime
- ★ Separation of concerns: app's knowledge of billing is now just a single, simple 'customer' objec
- ★ Shipped about a month ago, working great so far

Serverless looks kinda cool?!

Keep the monolith running

Act 3.

The Resolution.


logging
deployment
package managers
health check
unit testing
alerting
OMG
init scripts
switching cost
backups
training
operational metrics
security upgrades
restores
scaling


RUNLESS
SOFTWARE

We got good at

Linting
Upgrades
Test Reliability
Test speed
Code Ownership
Deployments
Scaling MySQL
Caching
...

We got good at

Linting
Upgrades
Test Reliability
Test speed
Code Ownership
Deployments
Scaling MySQL
Caching
...

We rewrote some core stuff

Ok Brian, are we ever going to the part where we monolith-ify perfectly good microservices?!

🙄


Webhooks


Webhooks

Webhooks

Reimplemented in the Rails Monolith

Help Centre

Copy and pasted into the Rails Monolith

Challenges with current architecture

Building new or changing existing features requires changes to two repositories. It requires additional thought about bundling changes in each repository to prevent bugs arising from different versions of applications running in production. Also adds a layer of complexity to reading the code for features as you need to grasp code across two repositories.

Challenges with current architecture

The standalone Rails application needs to be kept up to date by the team. This is coming at a significant cost. We haven't stayed on top of Rails upgrades. We are now reactively performing upgrades in response to security vulnerabilities getting discovered in dependencies. The last time we did this, we also caused a P1 by accidentally breaking the help centre for mobile SDKs

Copying and pasting

instantly fixed...

Upgrades!
Observability!
Developer ergonomics!
Availability!

Surprising benefits

Team fluidity

Operations


Oncall is wonderful.
Lead Dev London

AI


A
LOT
more to do.


A
LOT
more to do.

Lessons.

Lessons.

Thank you for coming to my talk!

 scanlanb