Sign O' The Times

Iterative system evolution at hyperscale

@Randommood

Play in the Sunshine

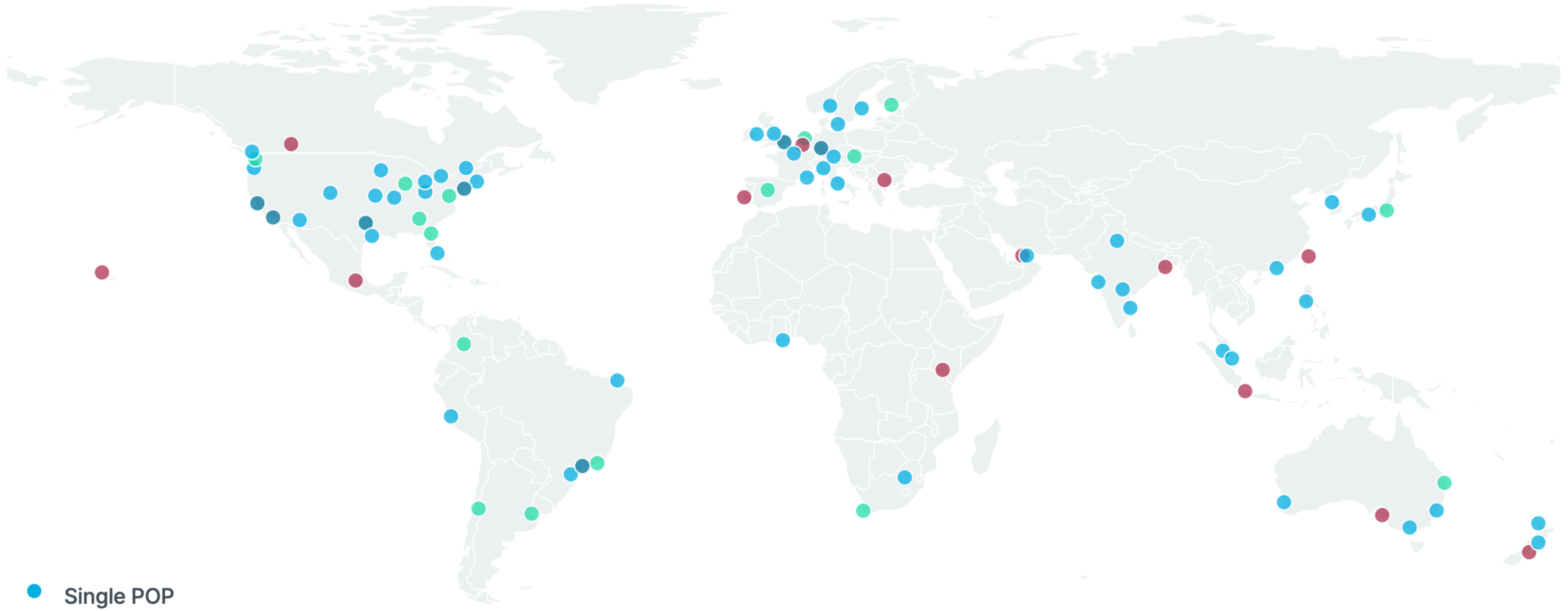
Dearly Beloved

Hello!

I'm Ines

@randommood

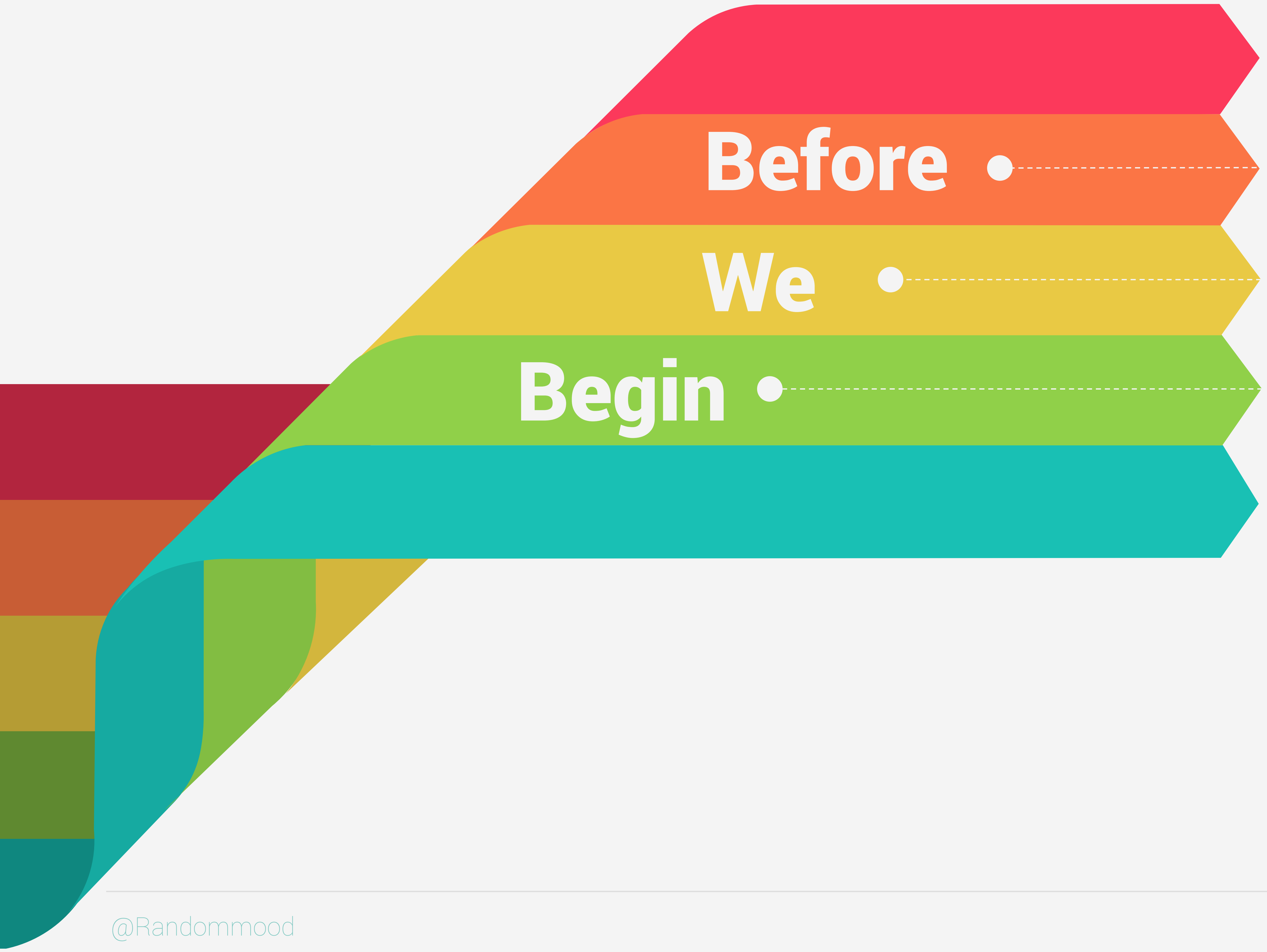
What is Fastly?



● Single POP
● Multiple POPs
● Planned new POP
● Planned upgrade

@Randommood

It’s Gonna Be A Beautiful Night



General Disclaimer

Most advice is **contextual**: mileage will vary

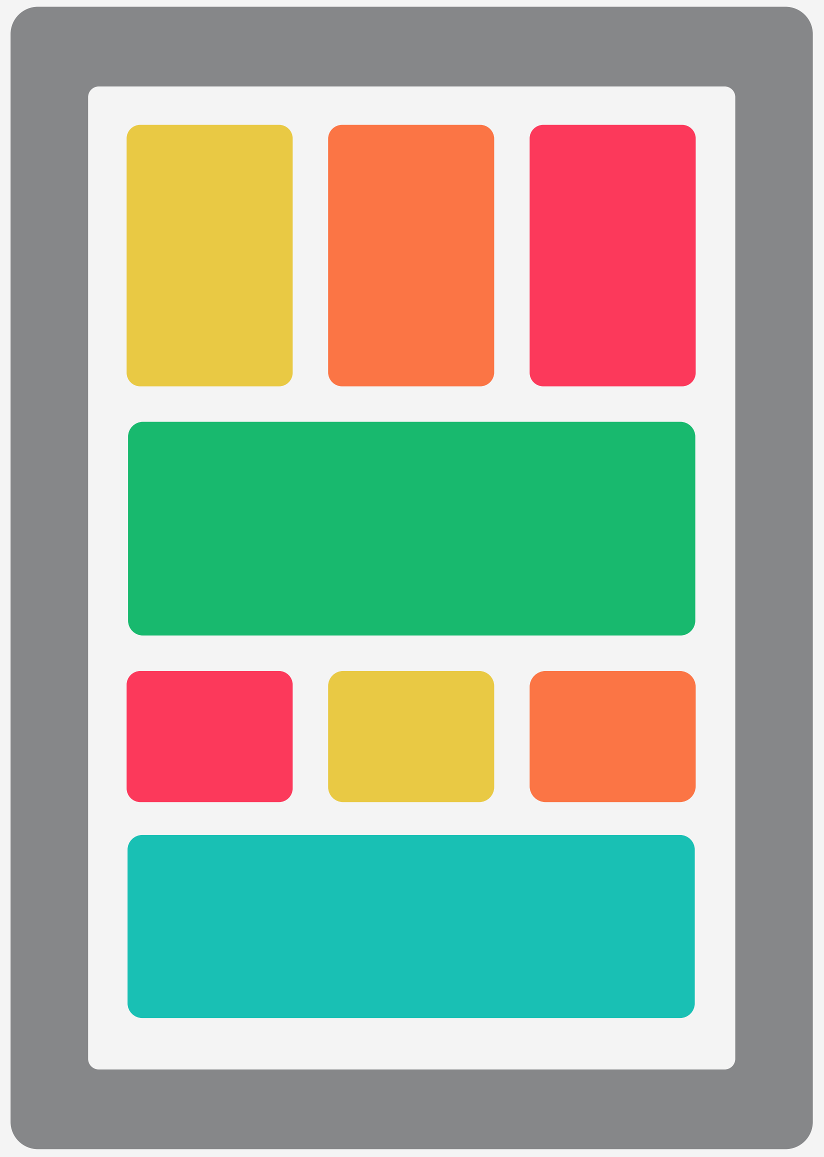
Approaches & patterns observed and tried throughout the years

Reflective of where I am in my own career & evolution

Sign O' The Times

On System Architecture

System Architecture



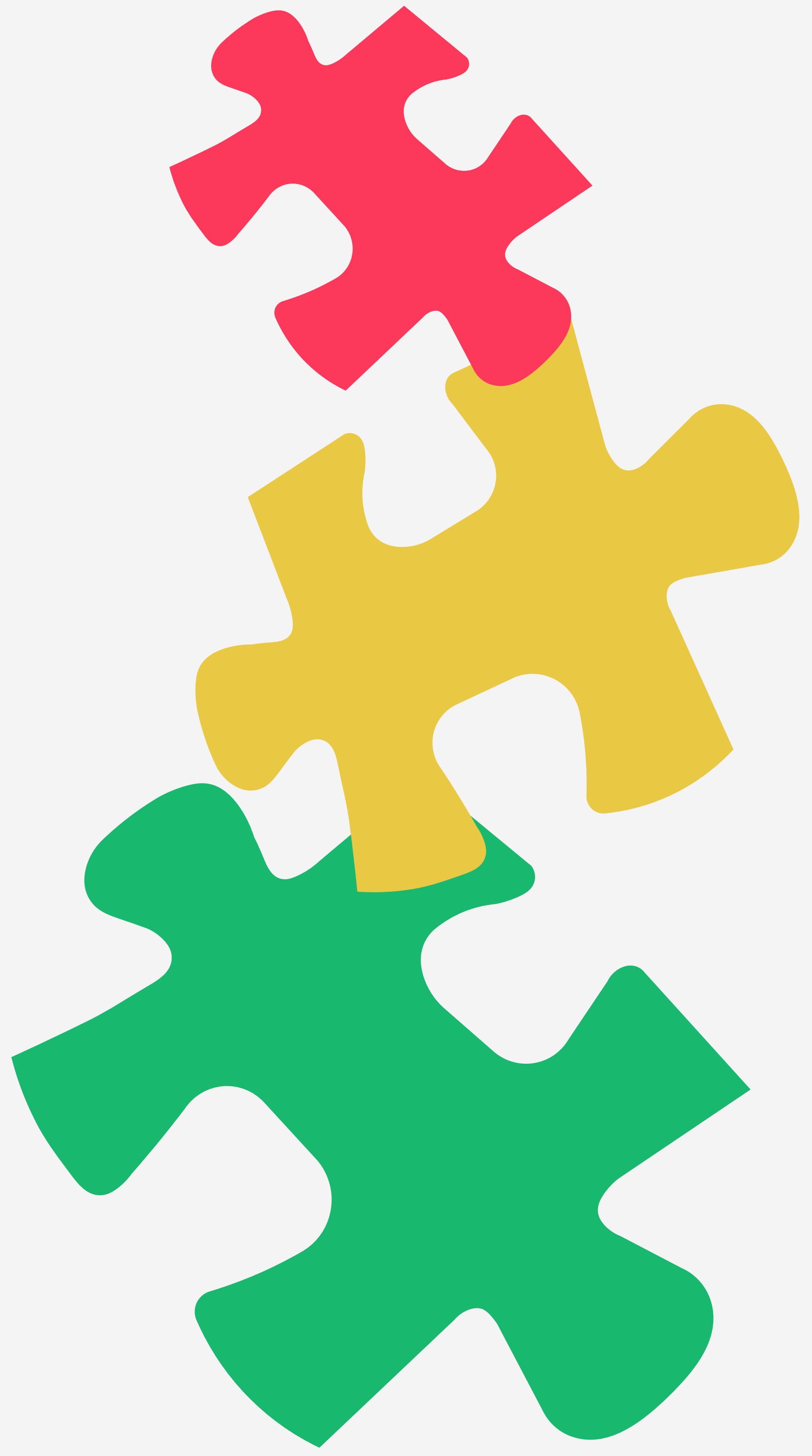
A System's shape or structure

- Language
- Abstractions
- Interfaces
- Operability & limits
- Singleton vs multi-node
- Stateful vs stateless
- Monolith vs service-oriented
- Relationship to its dependencies

A sign and a point in time

System architecture reflects the **history** and **evolution** of its

- **People** - designers, engineers, managers
- **Organizations** - leaders, team structure, team focus (full stack, specialized, product-focused)
- And their needs & constraints



System architecture is **contextual**

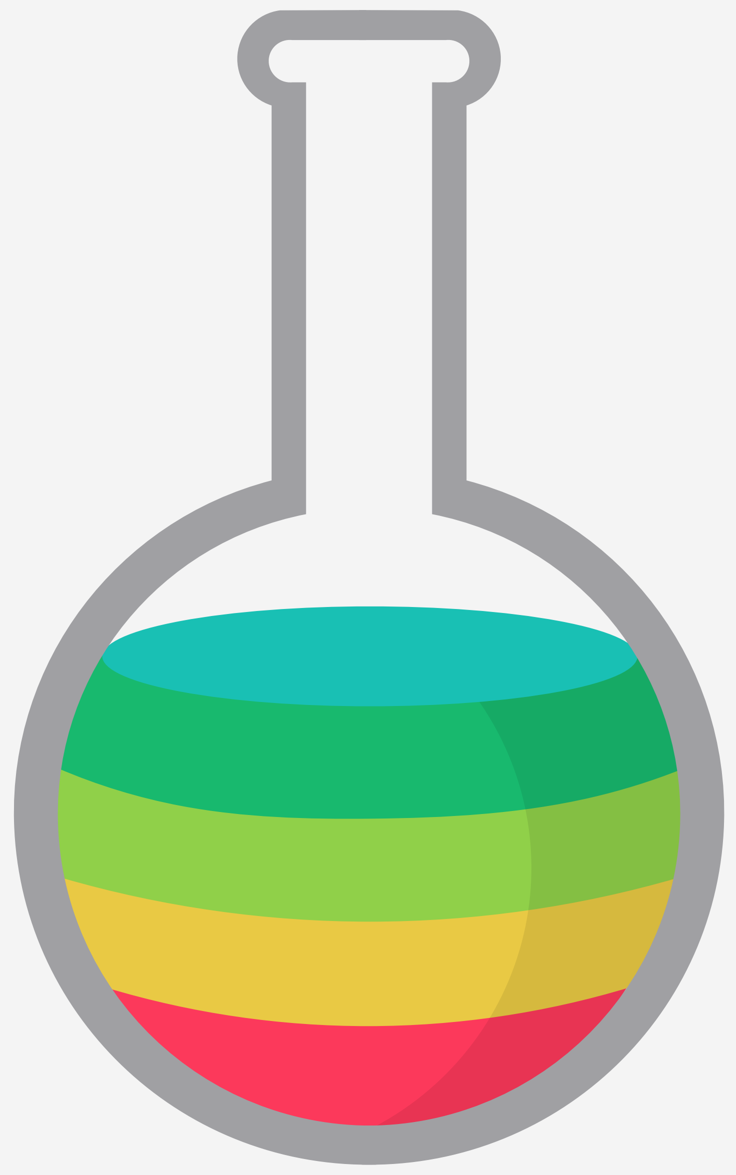
Balancing Needs & Limitations

REQUIREMENTS	CONSTRAINTS
Availability	Time
Performance	Cost
Scale	Team staffing
Cost	Organizational structure
	Compliance requirements

We make choices

They encode & frame a system's adaptability

- If we are right we can support 1-2 orders of magnitude growth without changes
- If growth is sustained it will **stress** our systems
- How this stress will manifest is **architecture dependent** and may be hard to predict



Strange Relationship

Sustained Growth & Systems

Growth & Scale

An arguably good problem to have

- Your architecture will determine where and how your system behaves under load *(resource starvation, degraded performance, plain process death)*
- This can be **difficult to reason about** or predict

Growth & Scale

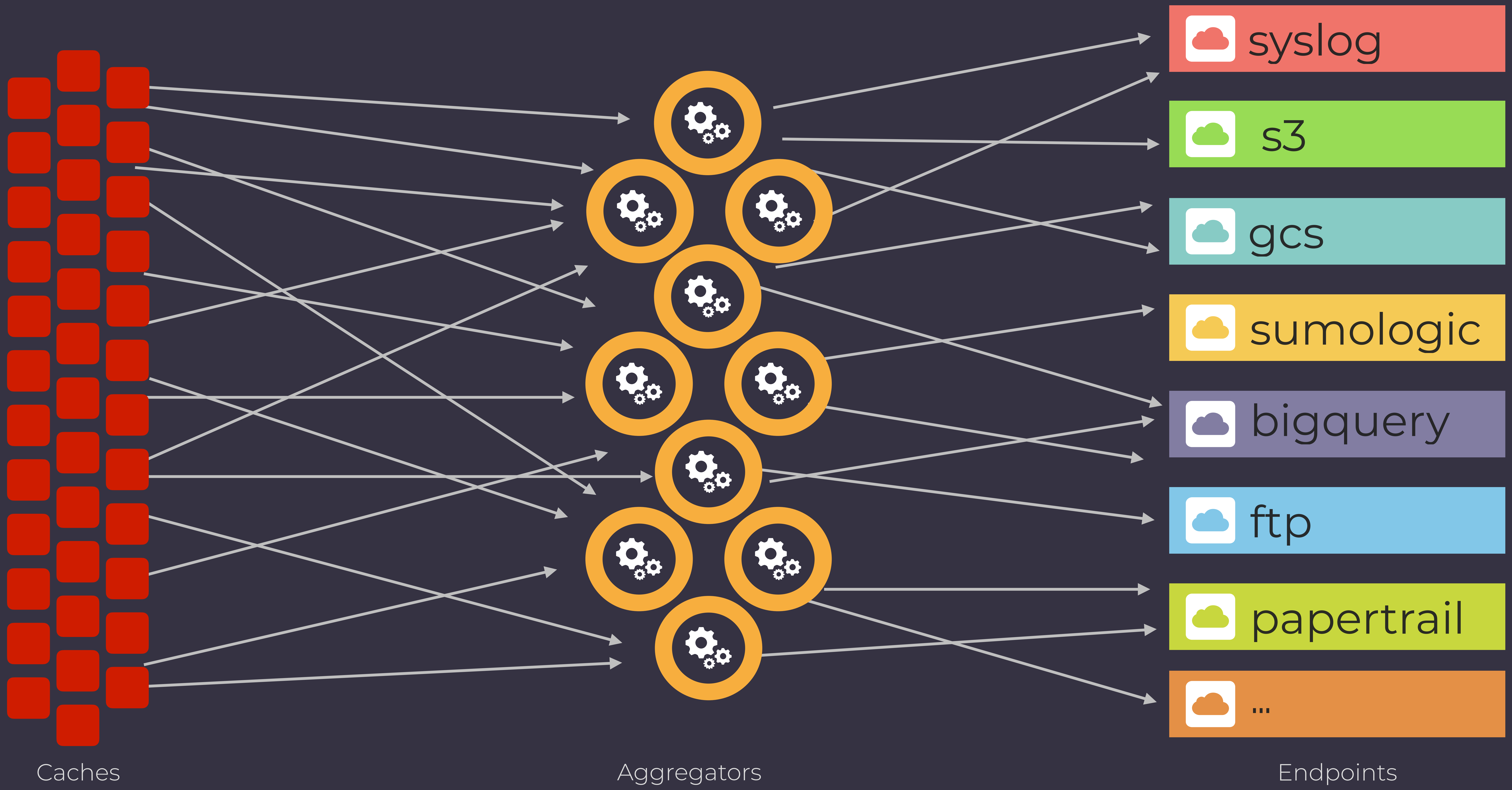
- Sustained load also impacts a system's environment, needs, and constraints
- Architectural changes are even more burdensome for hyperscale systems - lower tolerances for errors & performance regressions
- Your edge cases change too!

Two Examples

Maintenance and evolution of two critical data pipelines

- **Logging** pipeline
- **Metrics** pipeline
- Different architectures, teams, needs, constraints, and compliance scopes
- Both have large **sustained YoY growth**

Logging Architecture

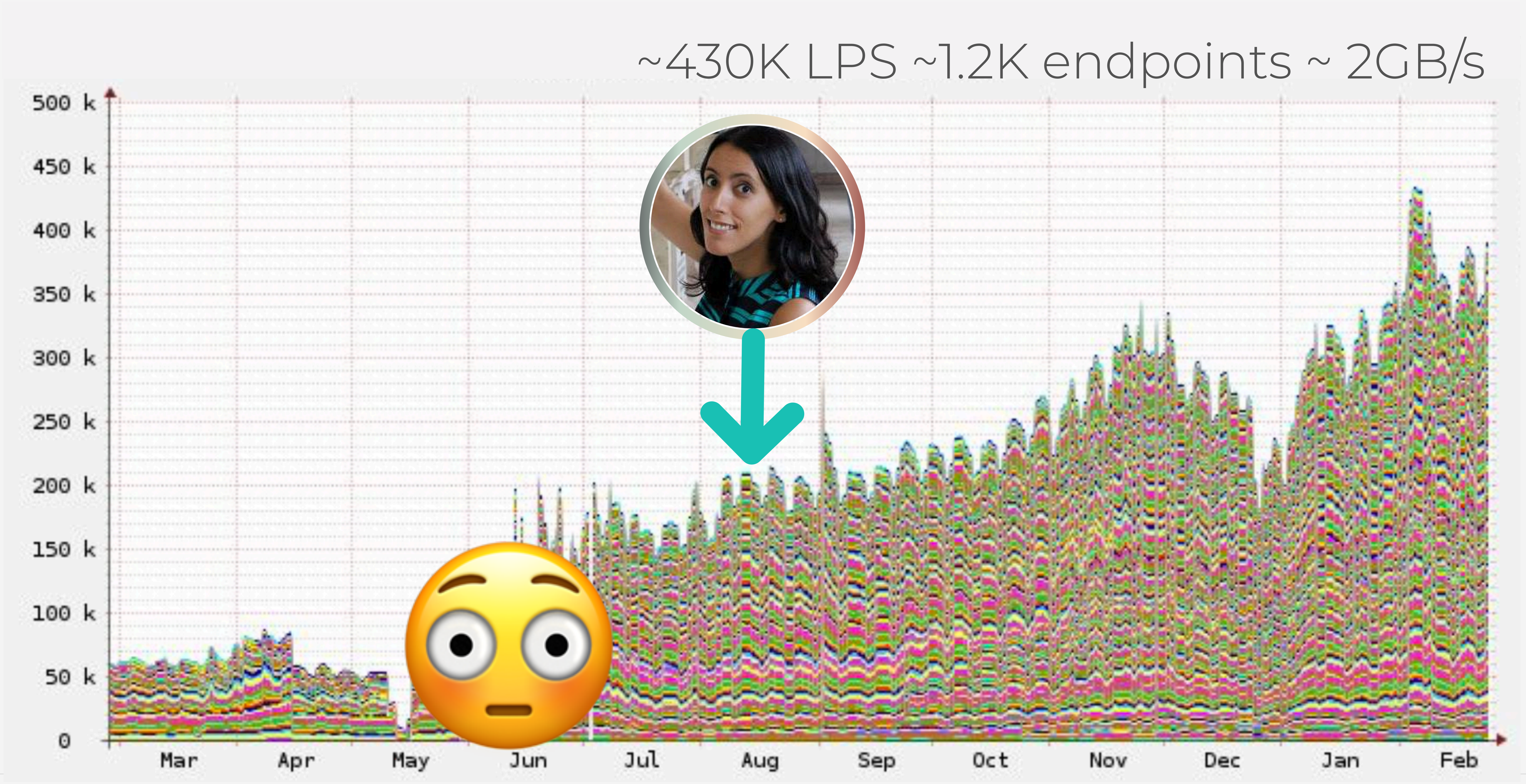


Architecture: Logging

- **Best Effort** - we try our best to send logs to your defined endpoint
- **Horizontally scalable** - more aggregators, more work
- **Stateless** - we have minimal buffering
- **Distributed** - send to whomever is available
- Pipeline optimized for log streaming speed

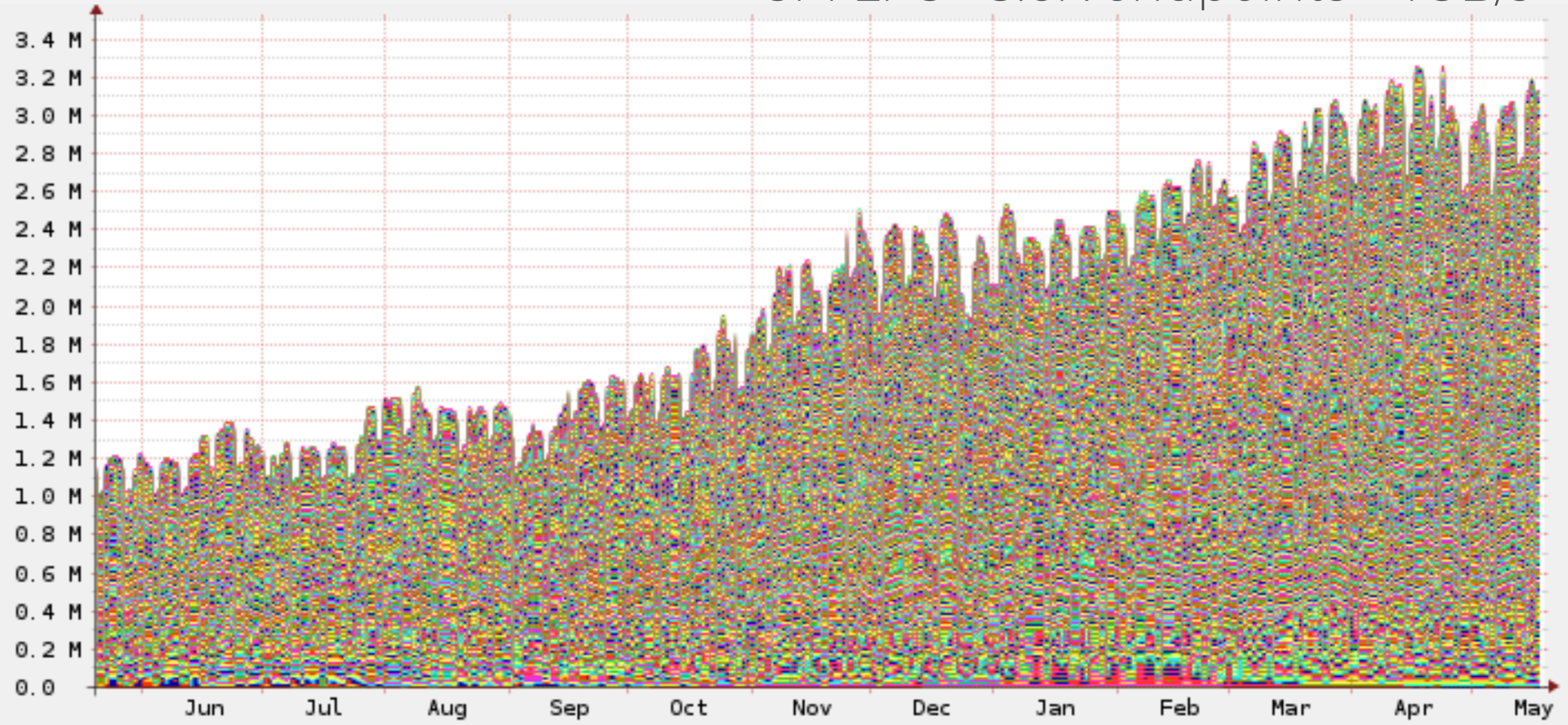
Growth: Logging (2014-2015)

~430K LPS ~1.2K endpoints ~ 2GB/s



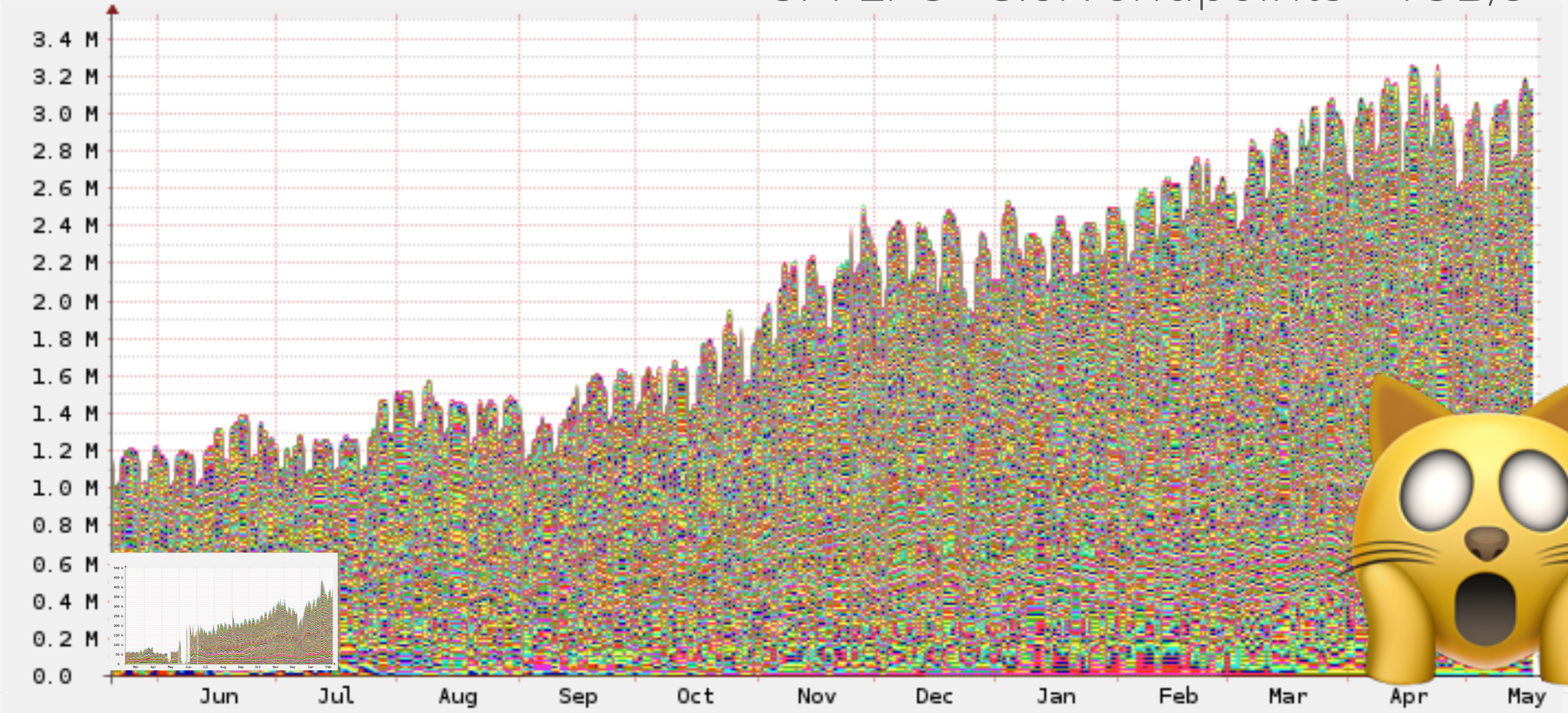
Growth: Logging (2017-2018)

~3M LPS ~8.6K endpoints ~4GB/s



Growth: Logging (2017-2018)

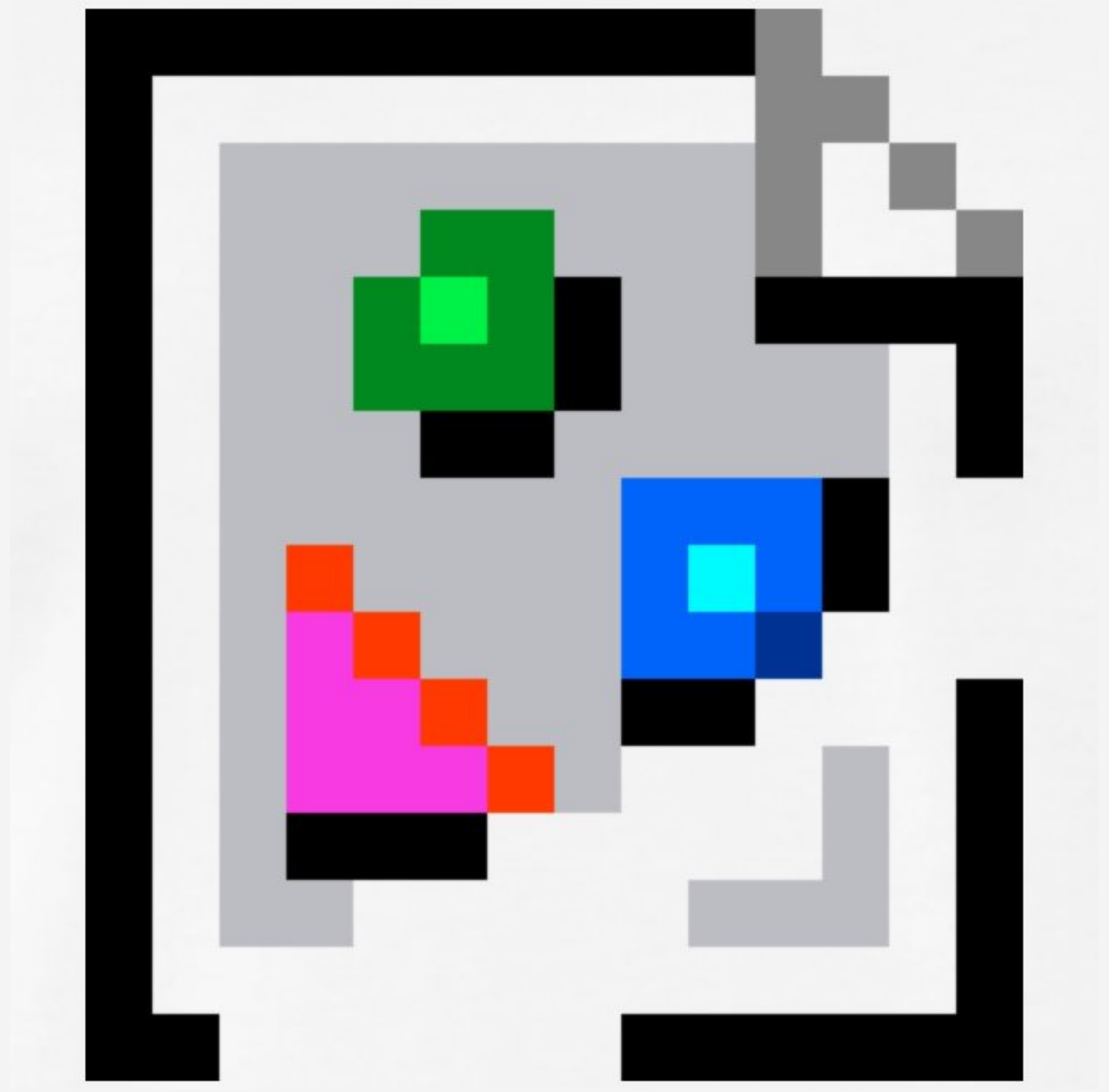
~3M LPS ~8.6K endpoints ~4GB/s



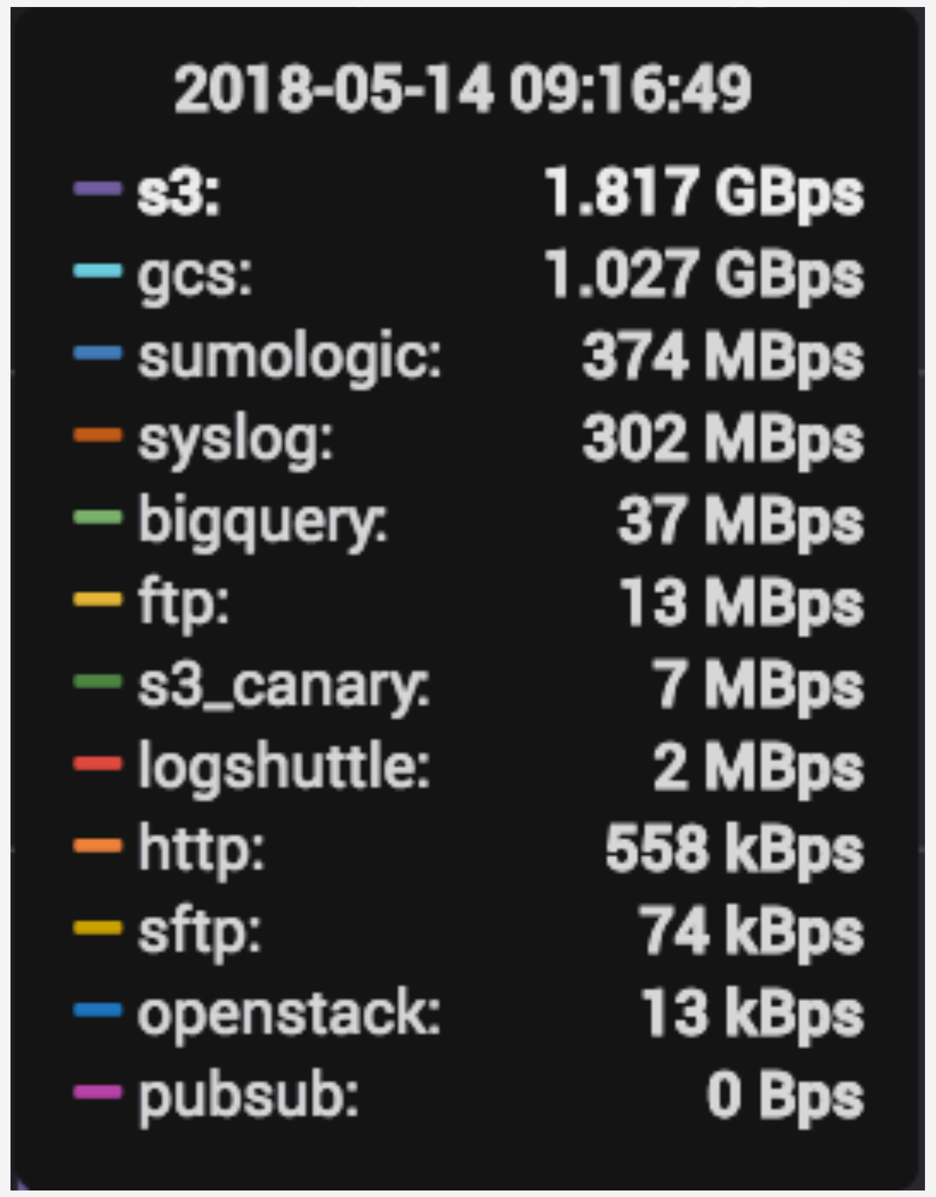
@Randommood

Growth: Logging (2021)

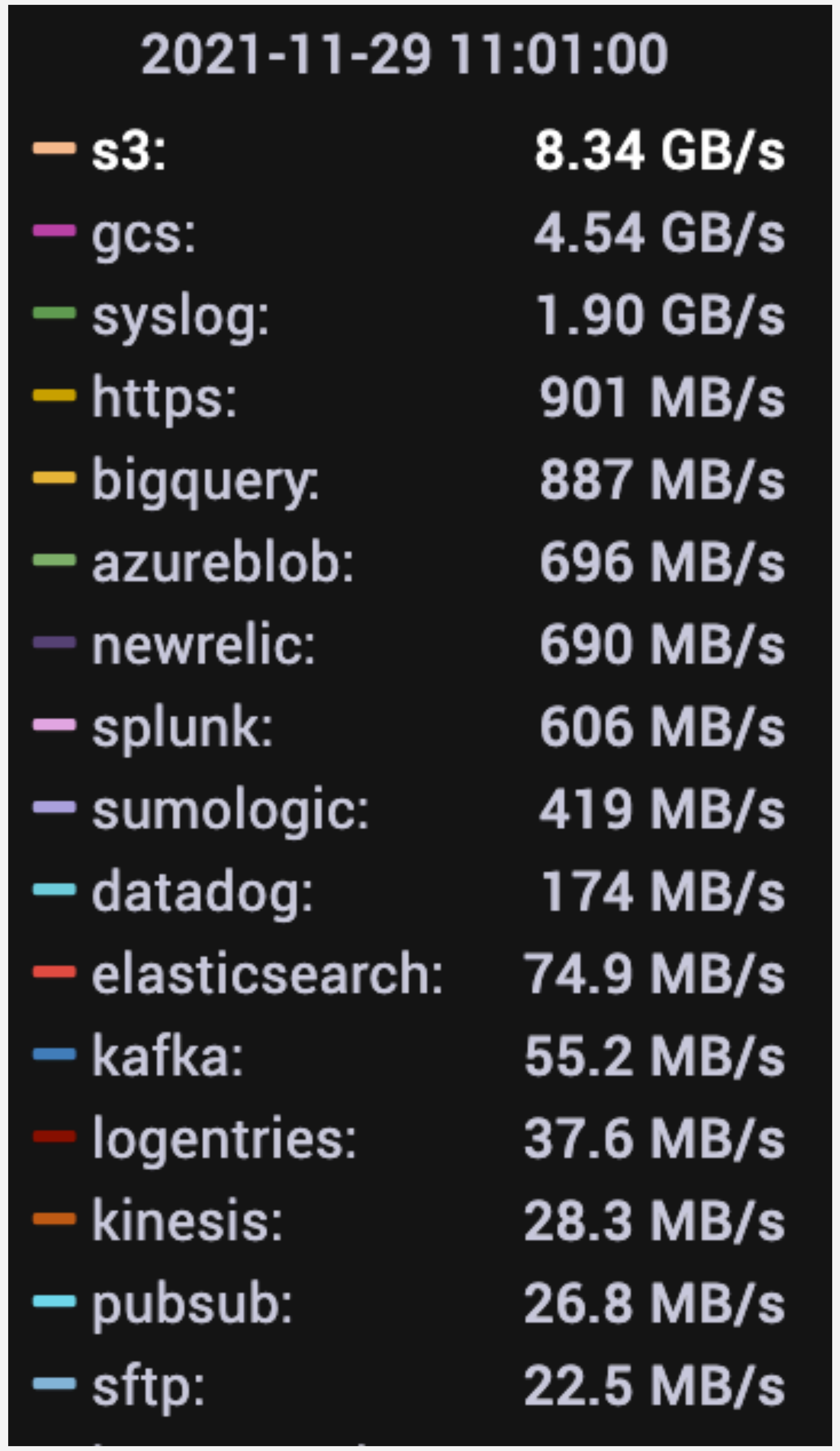
~18M LPS ~60.K endpoints ~19.5 GB/s



Growth: Endpoints (2018-2021)



```
      2018-05-14 09:16:49
— s3:                1.817 GBps
— gcs:               1.027 GBps
— sumologic:           374 MBps
— syslog:              302 MBps
— bigquery:             37 MBps
— ftp:                  13 MBps
— s3_canary:             7 MBps
— logshuttle:            2 MBps
— http:                558 kBps
— sftp:                 74 kBps
— openstack:            13 kBps
— pubsub:                 0 Bps
```



```
        2021-11-29 11:01:00
— s3:                   8.34 GB/s
— gcs:                  4.54 GB/s
— syslog:               1.90 GB/s
— https:                901 MB/s
— bigquery:             887 MB/s
— azureblob:            696 MB/s
— newrelic:             690 MB/s
— splunk:               606 MB/s
— sumologic:            419 MB/s
— datadog:              174 MB/s
— elasticsearch:       74.9 MB/s
— kafka:               55.2 MB/s
— logentries:          37.6 MB/s
— kinesis:             28.3 MB/s
— pubsub:              26.8 MB/s
— sftp:                22.5 MB/s
```

Challenges of Logging at Scale

- **No hard limits** to what you can log - this can be challenging
- System is **multi-tenant** - noisy neighbors can affect delivery
- Language introduction in the ecosystem complicates user experience (Varnish vs Rust)
- **Connectivity is critical** - can’t ship where we can’t reach

Challenges of Logging at Scale

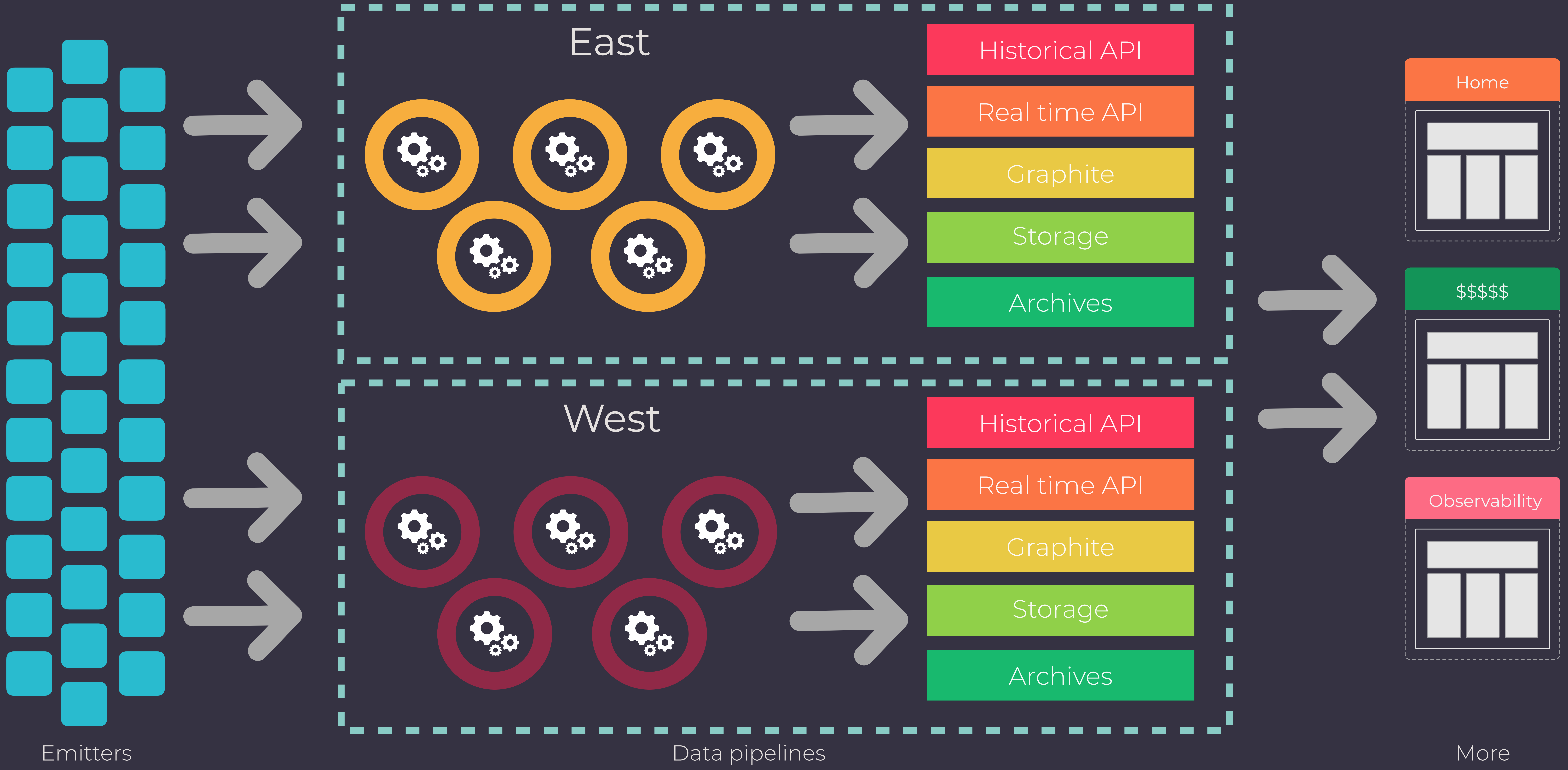
- Difficult to infer if an endpoint is working or not (Hard to test setup too)
- Classic **integrations challenges** - each endpoint is a murky downstream dependency
- Evolving our clients is challenging & takes time too - *example: new region support, 2018 ~20K LP/s BQ limits*

Evolution of Logging

What we are currently working on

- Per endpoint metric emission to facilitate debugging
- Decoupling **intent** from **log line structure** for better user experience & language support
- Dynamic cache-side configurations for ease of operability
- Aggregator affinity & regionality efficiencies

Metrics Architecture



Architecture: Metrics

Another critical pipeline

- **Guarantees needed** - metrics == observability + $$$
- **Mixed-scaling** - multiple independent regions pattern
- **Stateful** - storage, aggregation, archival, presentation
- **Replicated & verified** - needs continuous data verification processes
- Pipeline optimized for metric aggregation & availability

Growth: Metrics

What drives this pipeline's growth?

- **New emitter types** - 1 msg/sec each * number nodes
- **New streams types** - Varnish, C@E, more!
- **Network growth** - New POPs & caches
- **New metrics** - 200 and counting (we started w/ around 40)

Growth: Metrics (2020-2021)

Metrics pipeline has grown ~6x YoY

- Message **count** - 3k msg/sec > 15k msgs/sec
- Message **size** - 150 MiB/sec > 900 MiB/sec
- **Emitter** growth - ~4 new emitter types
- **Storage** growth - *way more, just trust me**

Challenges of Metrics at Scale

- Coupling of billing & observability metrics - pipeline likely needs folding into two
- **Connectivity is critical** - can't observe or monetize what we don't have
- Legacy serialization formats take time to replace
- Retention & presentation - APIs needed evolving to adjust for larger data volumes

Evolution of Metrics

What we are currently working on

- We evolved our retention policies to reduce cost
- Reworked & lowered emitter cost to bolster growth
- Gave up on pipeline unification - unify emitters & APIs instead
- Moving aggregation closer to emission to reduce bandwidth
- **ABS** (Always be Scaling)

Infinity is **folly**

Know your limits

Important things to be aware of

- **Scaling limits** - *Are we aware of what they are?*
- **Operational ergonomics** - *Do they set us up for errors?*
- **Teams & Organization** - *How much work can we do? Do we have organizational alignment?*
- **Cost & budget** - *Can you trade money for speed?*
- **Dependency relationships** - *who needs priority? Can you degrade or prioritize specific workloads?*

Hyperscale changes things

Your perspective will evolve, get comfortable with

- **A shortened time to rewrite** - yesterday's peak is today's baseline
- **New operational needs** - load shedding, failover, unified ergonomics, more consistency
- **Thinking about your emitters** - paved roads, organizational processes for workload onboarding
- **Knowing you'll never be done**

U Got The Look

When re-architecture is unavoidable

Photo: Jeff Katz

When?

Evolution is unavoidable when you start facing

- **Availability** issues
- **Performance** regressions
- **Scalability** limitations - of your system or its dependencies
- **Slower growth** / evolution - of your system components, dependencies, team, or company

Constraints frame your scaling approach

How?

Things that help you when re-architecting

- ❖ Foundational components, pipelines, systems
- ❖ Fewer dependencies
- ❖ Flexibility over scope & error tolerance
- ❖ Degraded modes of operation
- ❖ Flexibility over client behavior

How?

Helpful to rethink your approach to your system, people, team interactions, needs, & constraints

- Means setting up and supporting decoupling mechanisms between orgs
- Encouraging iterative improvements
- Paying for performance and cost increases with other improvements

Where?

You have choices here too!

- Emitter tier
- Transport tier
- Processing tier
- Storage tier
- Presentation tier
- Team / Org processes

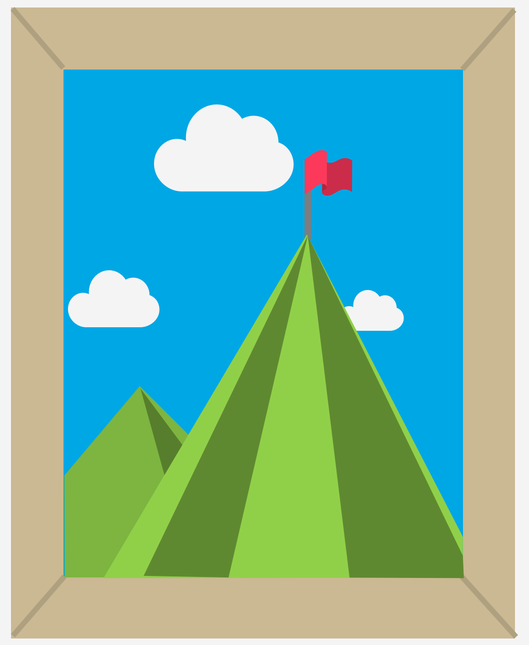
Forever In My Life

Reflections & Takeaways

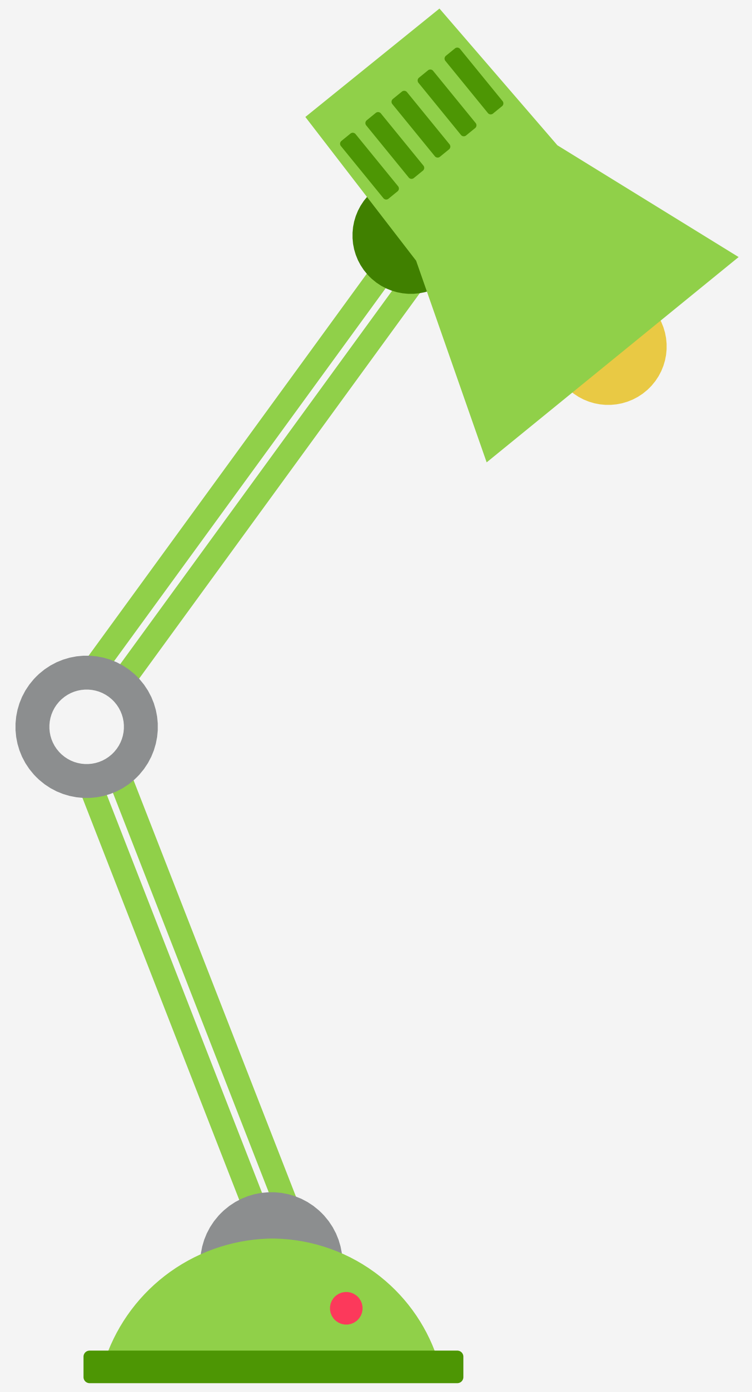
Design your evolutionary feature-set

Best approaches for a re-architecture are **iterative**

- Stalling during rapid growth is very hard and expensive
- Do you *really* have to do the whole thing at once?
- Can new features be used to validate abstractions?
- What architectural levels are in your control?



Ergonomics of evolution



- Always worth trying to **evolve things in place**
- Get comfortable with diversity - homogeneity makes evolution harder
- **Mixed mode as common** - new things should be backwards compatible
- Flexibility & control over your configuration helps

Always pick the shortest path to Customer Value

Mind unknown constraints

You're charting new territories

- Compliance & Governance
- Staffing & Cost
- Organizational changes
- Unknown dependency limits
- Regulatory changes

tl;dr

ARCHITECTURE

All architecture is **contextual**

It reflects history & evolution of people, organizations, needs, and constraints

SCALE

Scale / hyperscale means **growth**!

Nothing is infinitely scalable

The only control you have is how you deal with it w/o making too many mistakes

EVOLUTION

Best approach is **iterative**

Shortest path to customer value is ideal

Constraints frame your scaling approach

Thank you

github.com/Randommood/SignOtheTimes

Photo: Jeff Katz