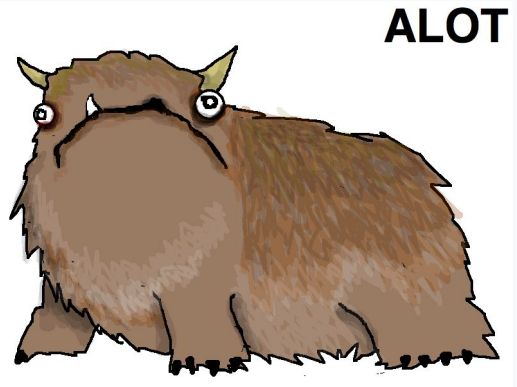
Google



SLOs are a Lot

go/slos-are-a-lot

Reliability
Advocacy

smcghee@
go/smcghee

Jan 2025

Agenda

01 Super Abstract

02 Slightly Less Abstract

03 Abstract Examples

04 Philosophy Again

05 Some Actual Examples

06 Wrap-up, Links

07 Q&A

% whoami

- has worked here **18 years, 4 months**
- last joined **5 years, 9 months ago**
- first joined **20 years ago**
- has joined **twice**
- has had **7 roles**
- has worked from **8 offices**
- has worked from **2 countries**
- has had **19 different managers**
- has had **62 unique direct reports**

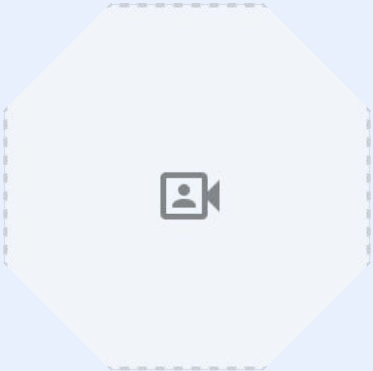
Steve McGhee
Reliability Advocate

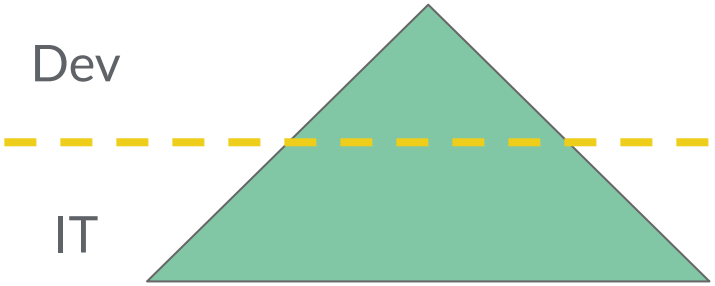
@stevemcghee

SRE on: Ads, Search, Gmail, Android, Fiber, YouTube, Cloud

01

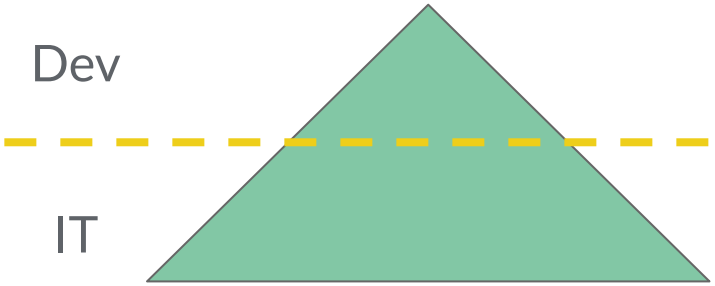
Super Abstract





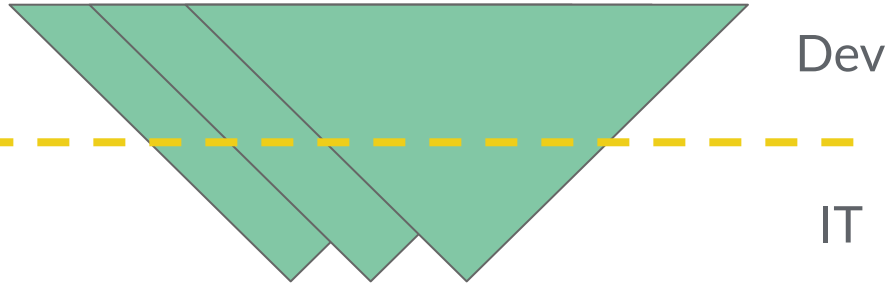
Worked great, for a long time

Common mental model



Worked great, for a long time

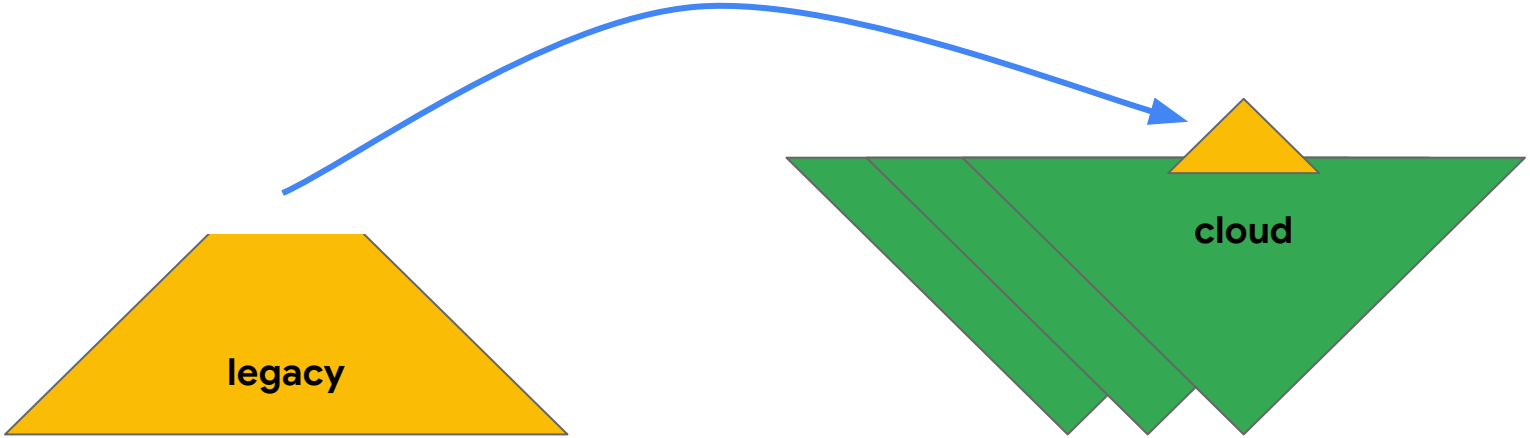
Common mental model



Cloud is here, though.

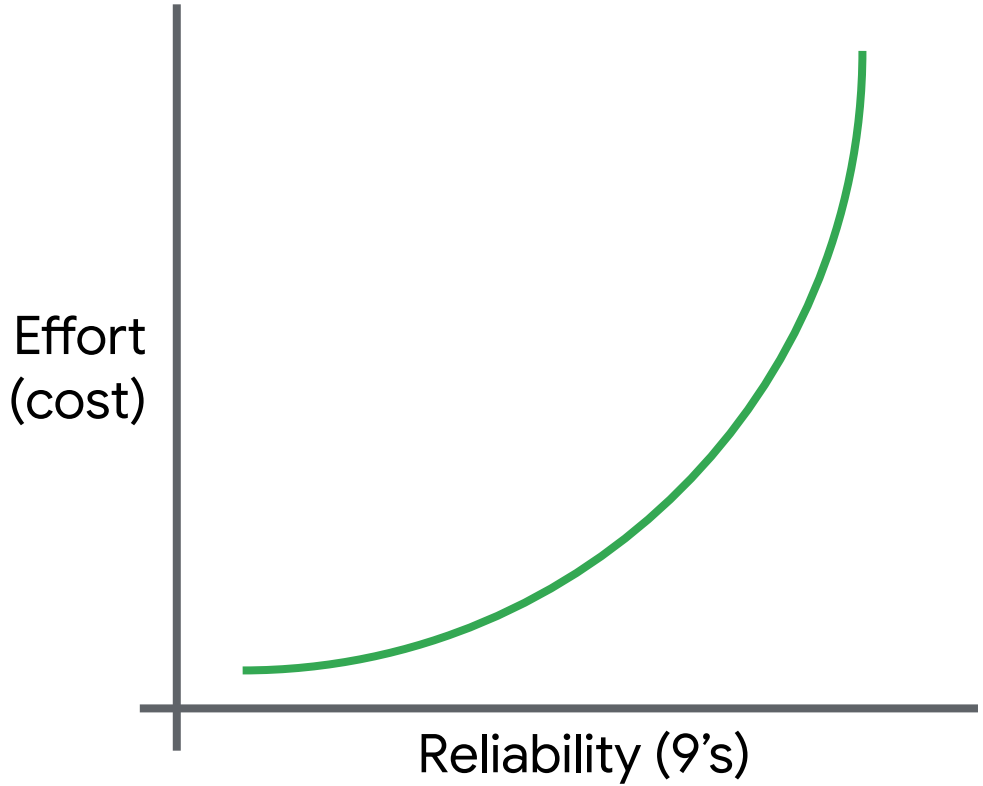
(because scale, mostly)

((You can't buy more nines for your VM in Cloud))



Infrastructure changes **can't fix** the app.

Continuum of platforms and complexity (reliability example)



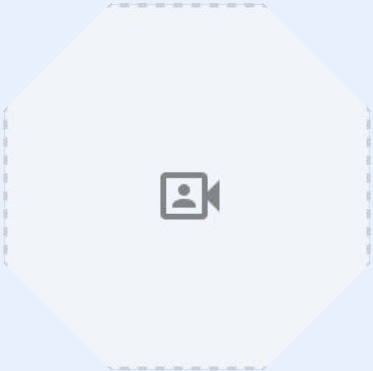
This Bears Repeating

You can build
more reliable things
on top of
less reliable things

a simple example: RAID. see: *The SRE I Aspire to Be*, @aknin SREconEMEA 2019

02

Slightly Less Abstract



SLOs in one slide

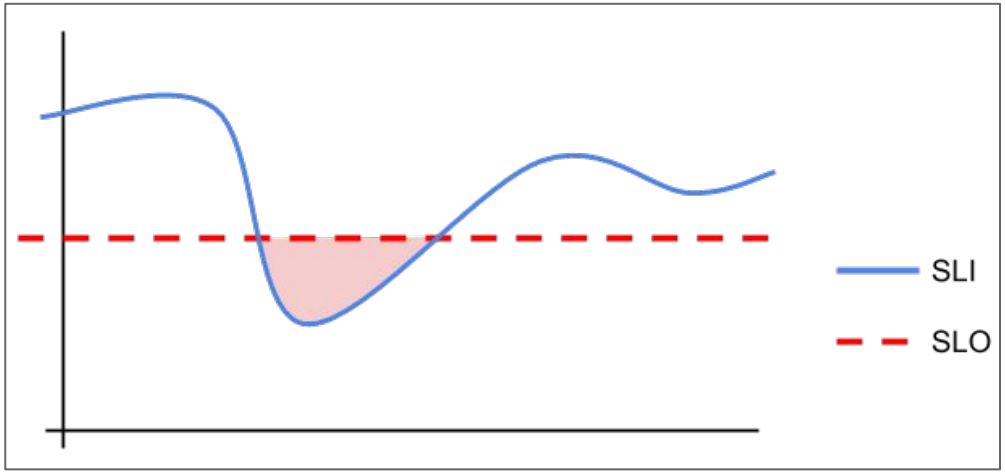
A **ratio-rate** of `good/total`, measured over a time duration.

If too much non-good, for too long, tell a human.

SLI is the squiggly line

SLO is the straight one

Area is consumed **Error Budget**



Glossary of Terms

CUJ

critical **user journey**: specific steps that a user takes to accomplish a goal

SLI

service level **indicator**: a well-defined measure of success

SLO

service level **objective**: a top-line target for fraction of successful interactions

Error Budget

proportion of **“affordable” unreliability**; one minus the SLO

SLA

service level **agreement**: business consequences

Critical User Journey

User interacts with Service to achieve Goal

Service Level Indicators (SLI)

Quantitative and carefully-defined as seen in the following equation:

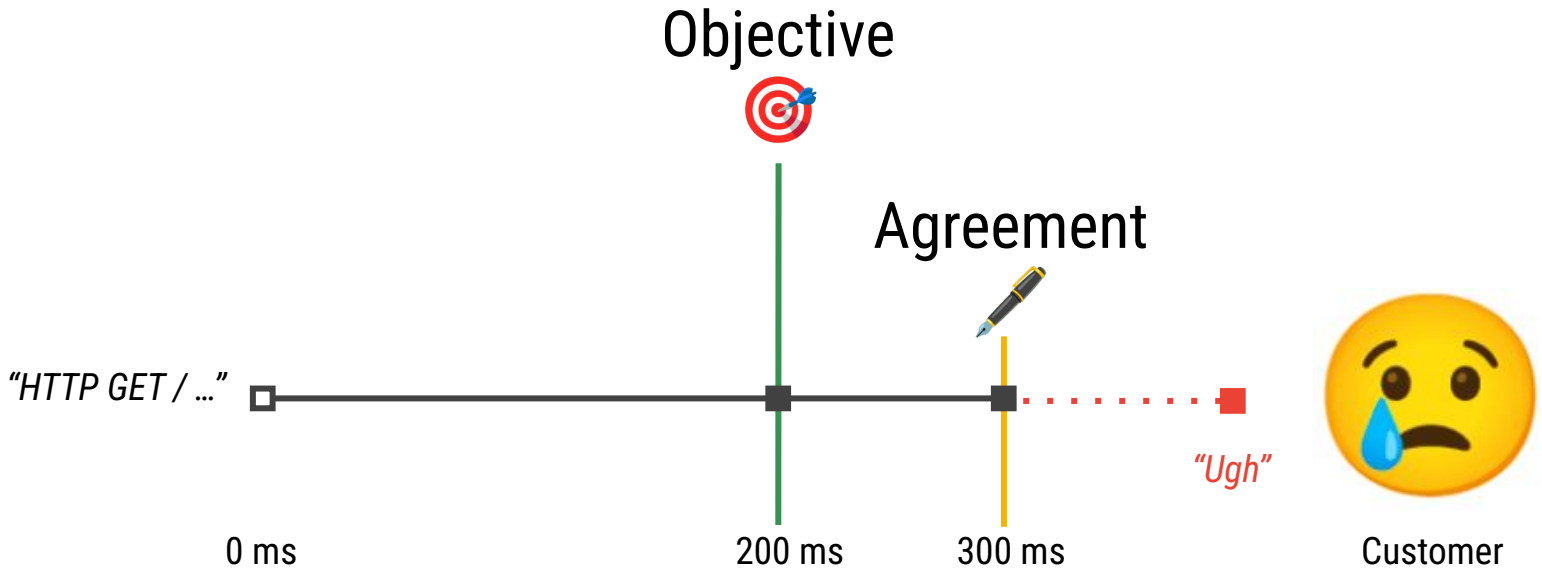
$$\text{SLI} : \left(\frac{\text{good events}}{\text{valid events}} \right) \times 100\%$$

Monitoring systems may (and should) capture a large number of potential SLIs, but most are not immediately useful to back SLOs

SLI Menu

Request / Response	Availability Latency Quality
Data Processing	Coverage Correctness Freshness Throughput
Storage	Throughput Latency

SLO vs SLA



"The Front Door SLO"

Focus on the customer's happiness.

- ➔ Available (enough)
- ➔ Fast (enough)
- ➔ Complete (enough)

Don't think about the serving system (yet).

Meet Expectations
Don't Expect Perfection

https://unsplash.com/photos/QR_vT8_hBZM

~~Bad~~ Naive Math

my users expect 99.0%

so my webserver should be 99.**9**%

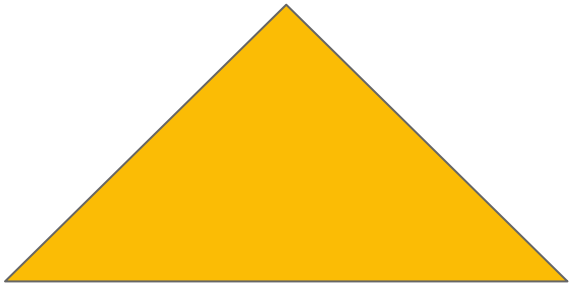
so my database should be 99.**99**%

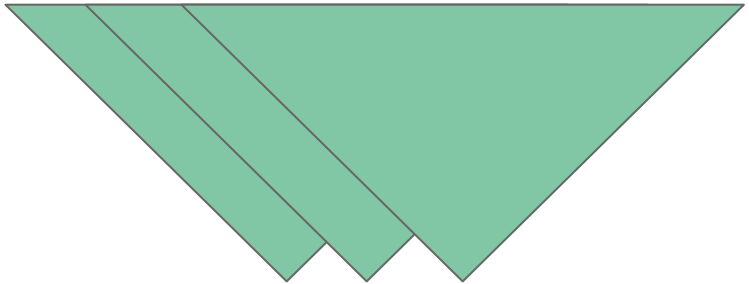
so my infrastructure should be 99.**999**%

… but what if i have *more* layers? 🤯

https://unsplash.com/photos/3kZpELkaxHc

Recall





Component-level reliability:

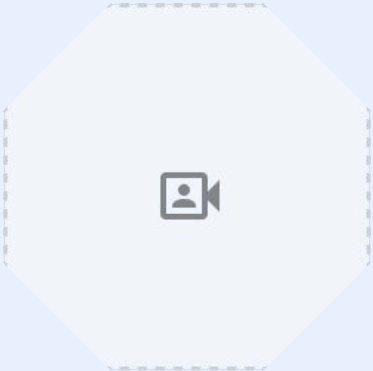
- solid base (big cold building, heavy iron, redundant disks/net/power)
- **each** component up as much as possible
- **total availability** as goal
- "scale up"

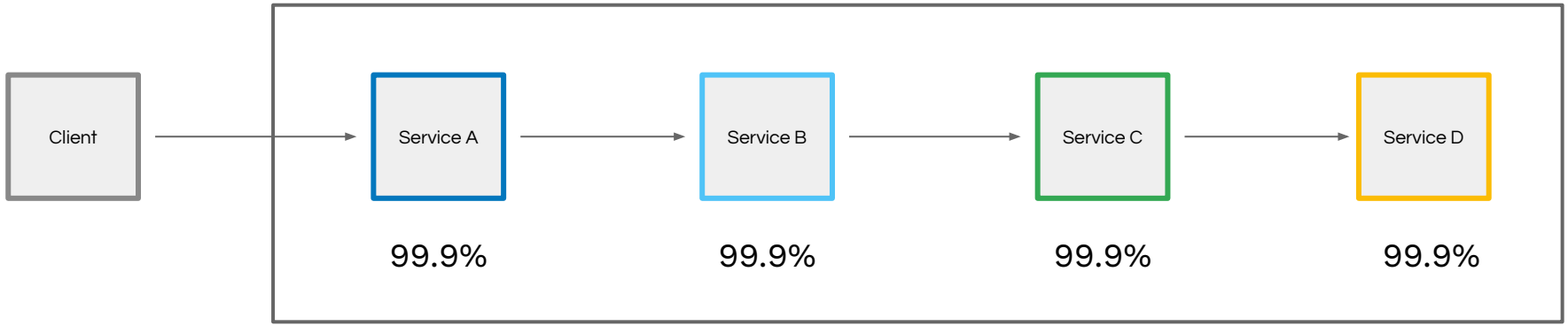
Scalable reliability:

- less-reliable, cost-effective base
- "warehouse scale" (many machines)
- software *improves* availability
- **aggregate availability** as goal
- "scale out"

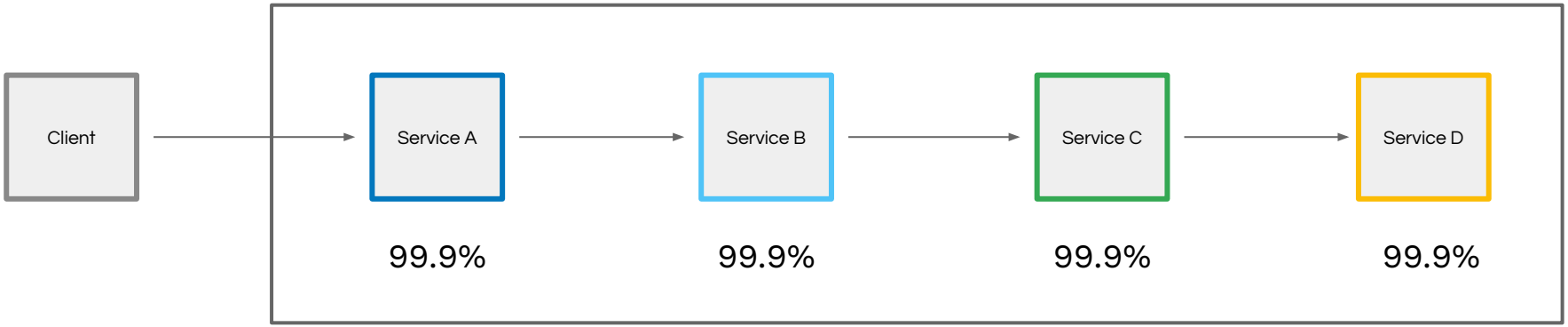
03

Abstract Examples



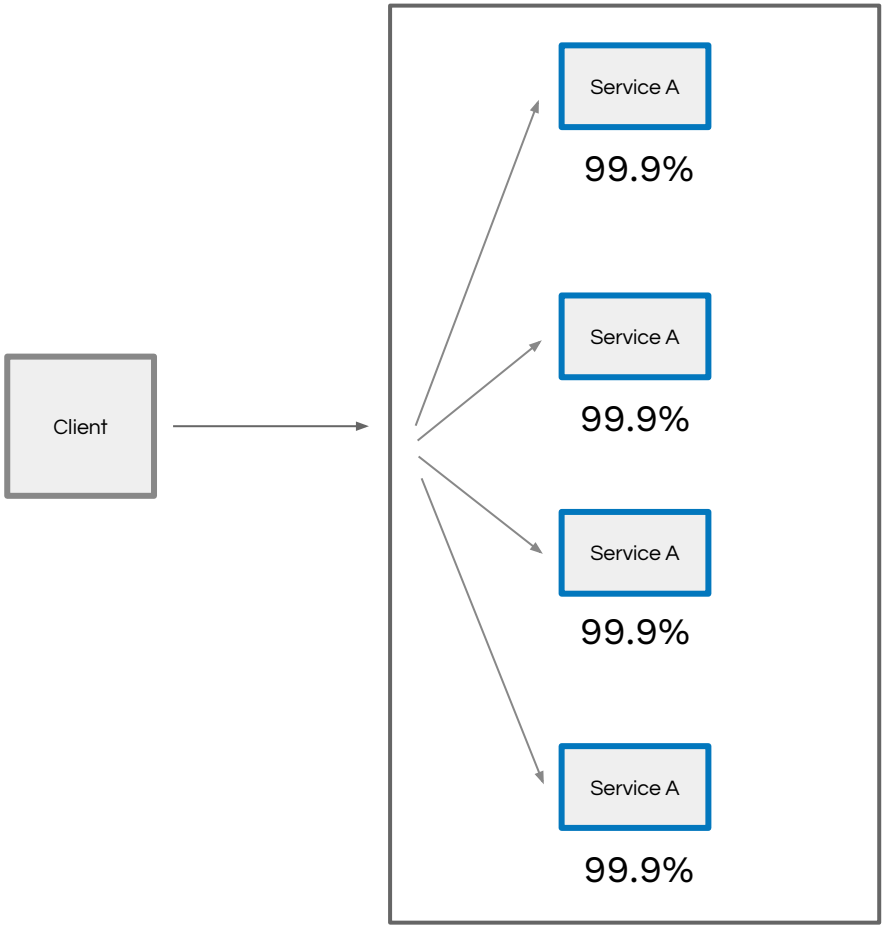


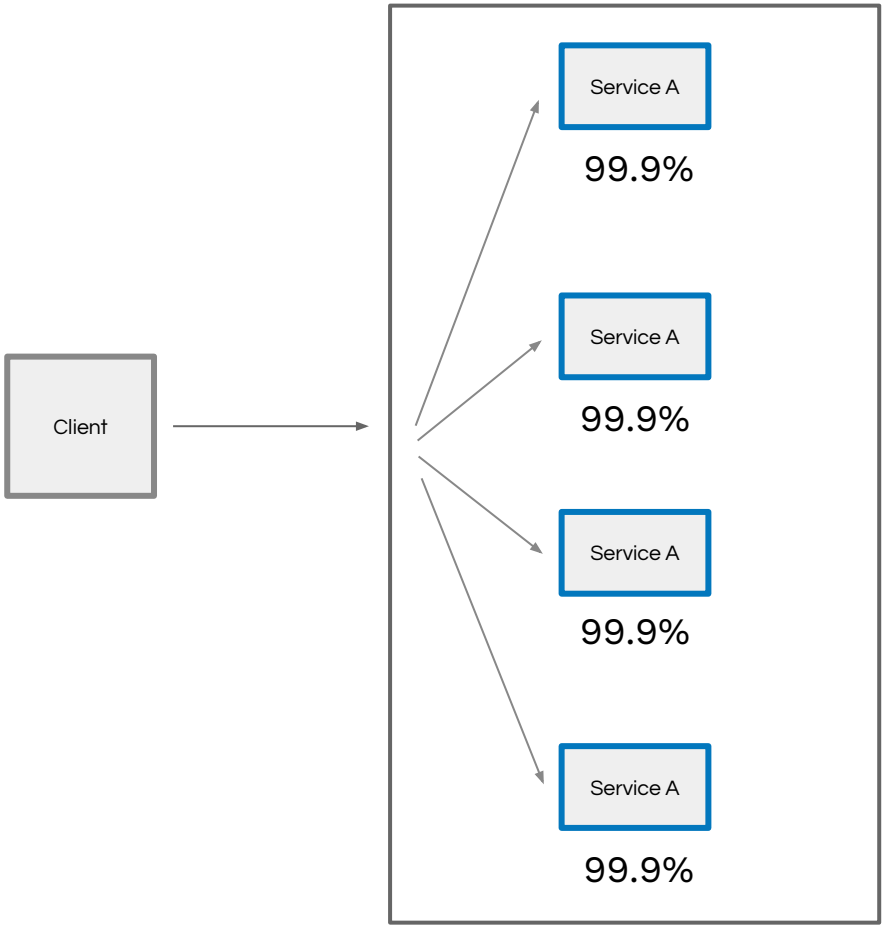
Intersection (or serial)



0.999 x 0.999 x 0.999 x 0.999

99.6% SLO





Union (aka parallel)

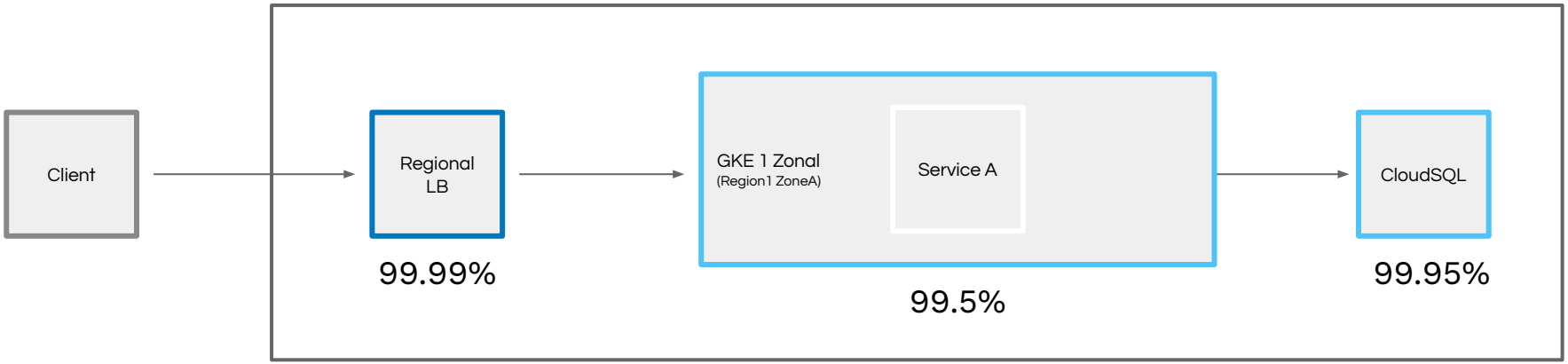
$1 - (0.001)^{4}$

99.999999999% SLO

or 11 nines

This is strictly mathematical and does not include any dependent variables like network, LBs, capacity planning, connectivity, and other dependent services

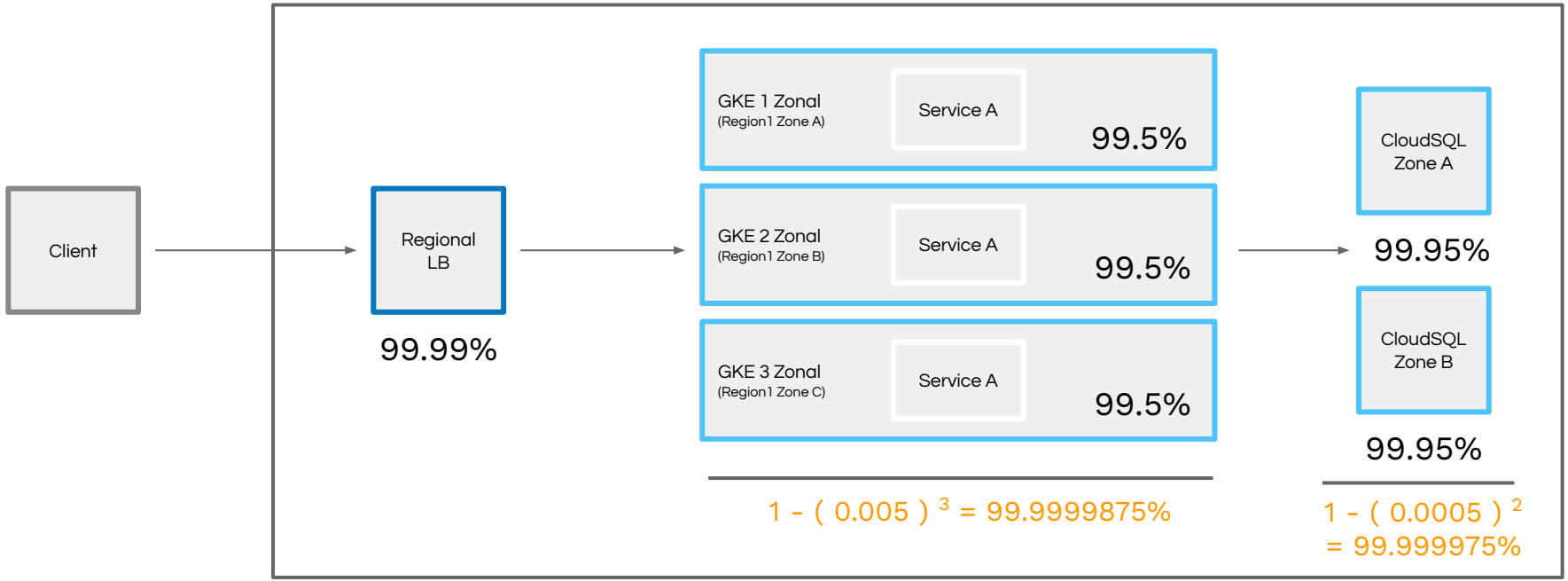
Archetype 2.1



0.9999 x 0.995 x 0.9995

99.44% LIMIT

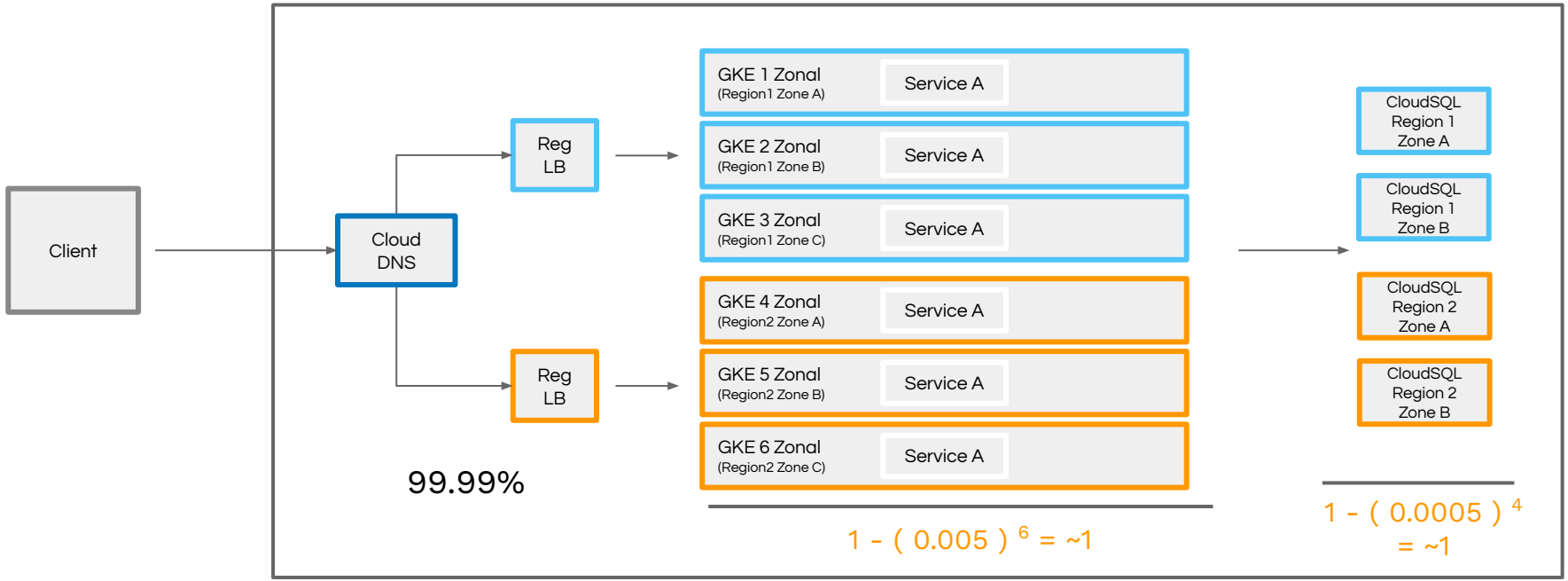
Archetype 3.1



0.9999 x 0.999999875 x 0.99999975

99.99% LIMIT

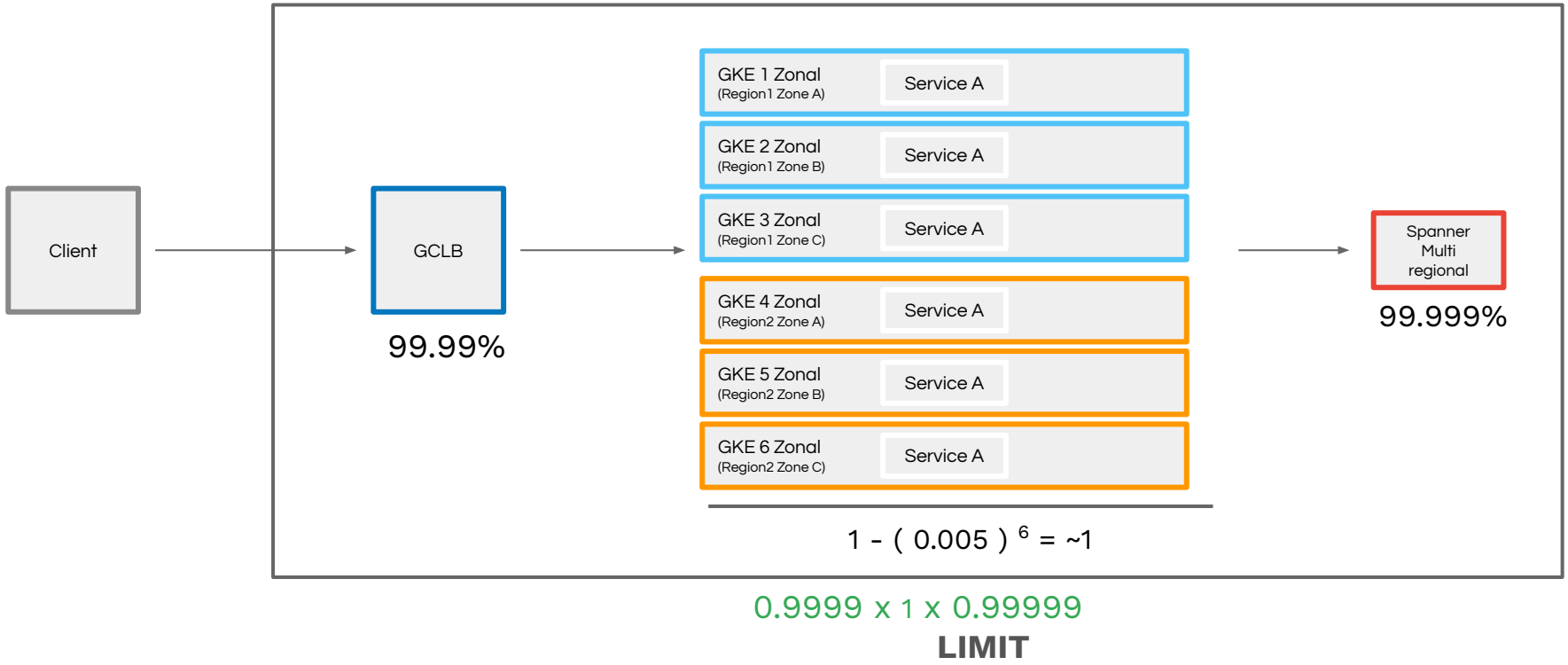
Archetype 3.2



0.9999 x ~1 x ~1

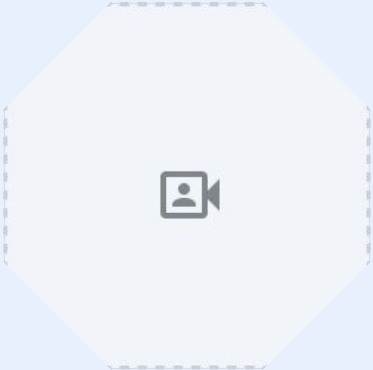
99.99% LIMIT

Archetype 5.2 Global with Cloud Spanner (Multi regional)



04

Philosophy Again



SLO Calculus

count ~ "num-errors.service"

rate() = SLI

ratio-rate() = SLO

ratio-rate() over time = budget

budget "burn" over time = alert

		$\mathbf{r}$	Position
$\frac{d\mathbf{r}}{dt}$	=	$\dot{\mathbf{r}}$	Velocity (speed)
$\frac{d^2\mathbf{r}}{dt^2}$	=	$\ddot{\mathbf{r}}$	Acceleration
$\frac{d^3\mathbf{r}}{dt^3}$	=	$\dddot{\mathbf{r}}$	Jerk
$\frac{d^4\mathbf{r}}{dt^4}$	=	$\ddddot{\mathbf{r}}$	Snap (jounce)
$\frac{d^5\mathbf{r}}{dt^5}$	=	$\overset{\cdot\cdot\cdot\cdot\cdot}{\mathbf{r}}$	Crackle
$\frac{d^6\mathbf{r}}{dt^6}$	=	$\overset{\cdot\cdot\cdot\cdot\cdot\cdot}{\mathbf{r}}$	Pop

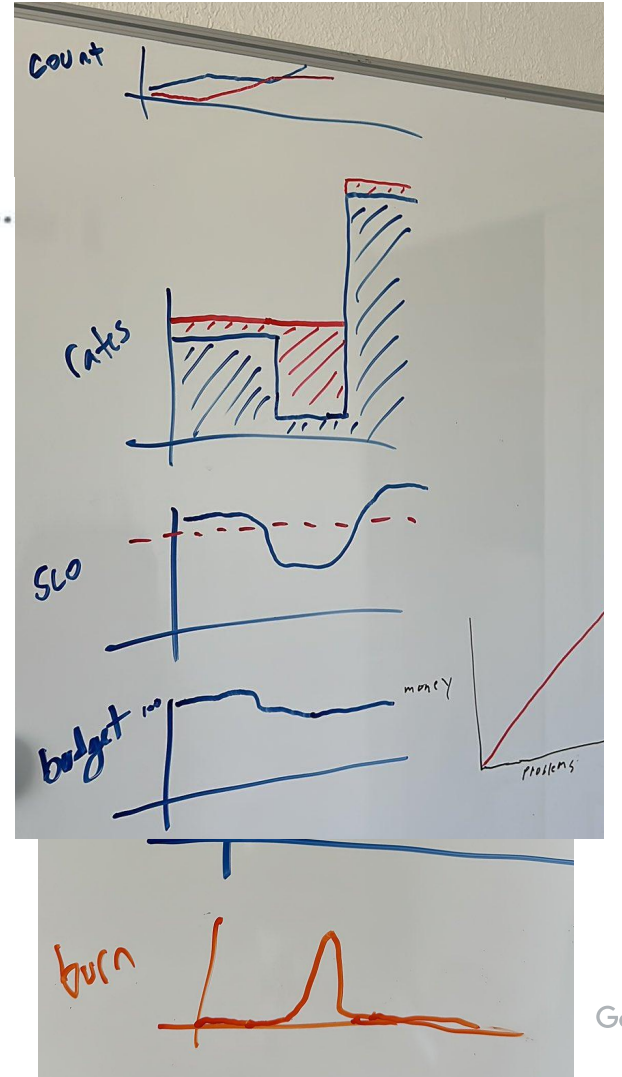
Steve McGhee
@stevemcghee

SLO Calculus

count ~ "num_errors"
rate() = indicator
ratio-rate() = SLI [ideal kSLO (a constant)]
100 - ratio-rate(), over time = budget
budget "burn", over time = alert

that last one is the trickiest. it's the derivative of an integral measured against a constant. i think.

2:41 PM · Jul 12, 2023 · **778** Views



Simon Frankau @simon_frankau · 16h

I think the SLO burn is an integral of a derivative. So you *could* collapse that away, bypass the rate step and treat it as the number of errors in the monitoring time window.

In the end, you're still looking for an increased average error rate over a time window.

1 30

Steve McGhee @stevemcghee · 10h

Yup there's value in both views.

1 1 15

Simon Frankau @simon_frankau · 10h

As an implementer I think it's invaluable to have multiple ways to look at it, as long as by the time you're done the numbers you plug in to it are meaningful to humans.

1 1 9

Steve McGhee @stevemcghee · 10h

My goal in the tweet was to show that the intuition isn't always there. It's hard to sort it out quickly in your head, esp when starting out. A sub sub subtweet with ahidalgo includes my attempts to graph. Not perfect despite a decade of this stuff.

Simon Frankau @simon_frankau · 9h

It's an area surprisingly replete with gotchas (and hence also curse-of-knowledge effects). One of my favourites is averaging errors-as-a-fraction-of-requests vs. absolute error rates in a system with spiky load.

1 1 15

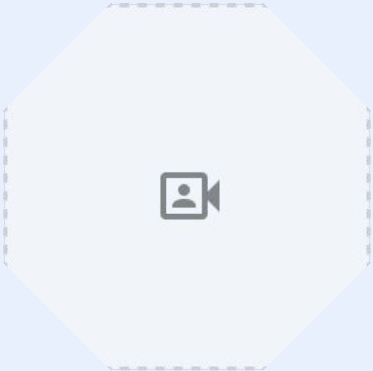
Steve McGhee @stevemcghee · 9h

Coordinated error spikes vs aggregate windows is so so dangerous. In android we faced 100kqps+ spikes that looked like ~1k in our "normal" graphs. Oops. Ask kits ;)

1 12

05

Some Actual Examples



Tired: Availability & Latency

Everyone talks about **Availability**: is it up?

Sometimes we talk about **Latency**: is it fast?

Sometimes we even combine them. Ooh, Fancy!

Wired: Freshness, Coverage, Skew, Duration

Those were: Request Driven Services

What about:

- Data Processing
- Scheduled Execution
- Using ML models ?!

How can SLOs help here?

Need to define `good` and `total`

Data Processing: **Freshness**

The proportion of valid data updated more recently than a threshold.

Examples:

- Generating map tiles in a game
- "X items in stock" in an ecommerce store
  - "The percentage of views that used stock information that was refreshed within the last minute."

Generally:

```
good = datetime_served - datetime_built < threshold
```

Data Processing: **Coverage**

The proportion of valid data processed successfully.

If your system processes many inputs, but drops some for various reasons (malformed, empty, unpaid, resource constrained), **Coverage** can help you assess the state of the whole system.

```
good = valid_records - num_processed < threshold
```

Measure this per "bucket" time period to see how coverage changes over time.

Scheduled Execution: **Skew**

The time difference between when the job should have started, and when it did

Skew tells you if a job (like cron) runs early, late, or on-time. It can have a negative or positive value.

```
good = time_started - time_scheduled < max_threshold
       and
       time_started - time_scheduled > min_threshold
```

Setting expected upper/lower boundaries provides a method for knowing what is considered "good" and making decisions from that.

Scheduled Execution: **Duration**

The time difference between when the job should have completed and when it was expected to complete by.

Duration helps us understand if a large system is "fast enough" as a whole. This can even catch never-ending jobs, if done correctly.

```
good = (time_ended or NOW) - time_started < max_expected
       and
       time_ended - time_started > min_expected
```

What about **Durability**?

Durability is confusing, as it tends to have SO MANY NINES (11!).

This is because it measures a *predicted distribution* of potential physical failure modes over time. By adding physical replicas and encoding schemes for storage, you can model, understand, and improve the durability.

This is **very different**. Try not to compare this directly.

See: pg 203-208 of Implementing Service Level Objectives by Alex Hidalgo for more math.

More SLO Terminology

***latency < 150ms, >95% per quarter*:**

- The `Service` is usually some sort of RPC request,
- The `Goal` is 95%,
- The `Criteria` is "latency < 150ms",
- The `Period` is "quarter".
- The `Performance` over the `Period` of the last quarter would be the "number of requests in the last quarter with latency < 150ms" divided by "the number of requests in the last quarter".
- The `Error Budget` is 100% - 95% = 5%.
- This `SLO` will be `Met` for the `Period` of 2016-Q1 if the `Performance` for that `Period` is greater than the `Goal` of 95%.

event-based vs time-based ?

Alex Hidalgo
@ahidalgosre

There are few things I'm confused about. First, how does this approach account for situations where your total burndown might improve due to you now having more good/total than before? For example: Huge spike in good events. Or does this assume a time-slice based approach?

3:26 PM · Jul 12, 2023 · **59** Views

Steve McGhee @stevemcghee · Jul 12
good points!
wrt burn rate, yep i think i had that in my head but it didn't fit into the tweet.

1 2 43

Alex Hidalgo @ahidalgosre · Jul 12
Pages 76-83 cover both "events-based" and "time-based" error budget maths and very smart math-types read it and let me publish it, so I think it's probably okay unless they were pulling a trick on me.

1 1 48

O'REILLY®

Implementing Service Level Objectives

A Practical Guide to SLIs, SLOs & Error Budgets

Alex Hidalgo

event-based

- N events ~= traffic
- spikey
- precise

time-based

- 86400 sec/day
- ~> 2880 30s windows
- smoother

SAME DATA, DIFFERENT VIEWS

rolling-window

- no big resets
- budget "heals" (?) 30d after errors
-

calendar-window

- monthly reset
- hides last-day badness (ok?)

Consider consequences, sprints, planning cycles
(14d? 30d? 31d?)

slow-burn

- if this keeps up …
- "file a bug/ticket"
- O(days)

fast-burn

- ~10% of 30d budget burned in 1h
- "do something now"
- O(hours)

tactics, consequences

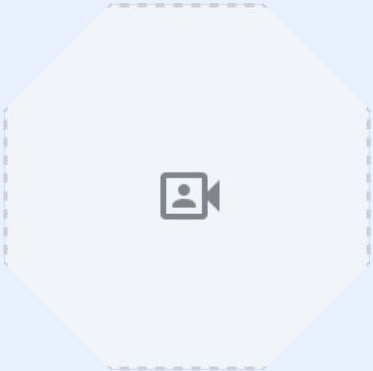
- ~~release freeze~~
- only reliability changes (?)
- …
- "turn the knob" towards reliability
- next sprint, next release

strategy, planning

- size of team ~= amount of work (beware toil tax)
- invest in reliability to slow interrupts
- it never ends :)

06

Wrap up, Links



Just write a plan:

- SLI X indicates user …
- SLO Y was chosen because …
- when it burns by …
- we will …
- because we believe …

make it public

revisit the plan quarterly/annually

or when it doesn't make sense anymore.

Google

Odysseus and the Sirens

- Odysseus and crew have a **plan** on how to handle disaster
- During the disaster, they stick to the plan, even though The Boss told them not to.
- This is known as a ***Ulysses Pact***

➔ When defining SLOs, you're deciding what is a disaster and what isn't, and what to do about it.

➔ If it **isn't** a disaster, don't treat it like one.

➔ If it is a disaster, **stick to the plan.** (note: *have a plan*) Focus on bringing the service "back into SLO"

➔ practice your plan. run drills, develop tools

https://www.theoi.com/Pontios/Seirenes.html

Parting Shots

SLOs are a measure of a system, not components. SLOs are **an abstraction.**

Not a **replacement** for the deep understanding needed for **diagnosis**.

Abstractions provide **consistent understanding** of behavior **through change**.

This is good.

Implementation can change, side-effects can come and go.

SLOs persist.

Resources

https://sre.google/resources/practices-and-processes/art-of-slos/
https://www.alex-hidalgo.com/the-slo-book
https://sre.google/sre-book/service-level-objectives/

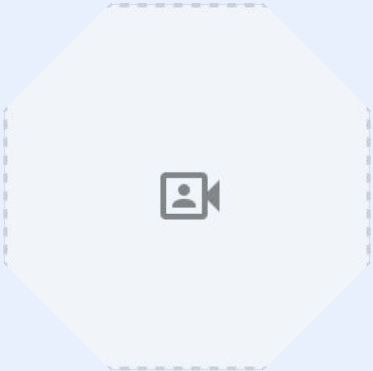
https://www.nobl9.com/
https://docs.datadoghq.com/service_management/service_level_objectives/
https://docs.newrelic.com/docs/service-level-management/intro-slm/

https://github.com/google/slo-generator
https://cloud.google.com/monitoring/slo-monitoring

https://www.youtube.com/watch?v=OdLnC8sjPCI

07

Q&A



Appendix

Error budget burn-down chart

		Percent of Service Unavailable			
		0.1%	**1%**	**10%**	**100%**
Availability SLO	**90%**	Forever	Forever	Forever	9d
	99%	Forever	Forever	9d	21h
	99.9%	Forever	9d	21h	2h
	99.99%	9d	21h	2h	12m
	99.999%	21h	2h	12m	1m

DR Math

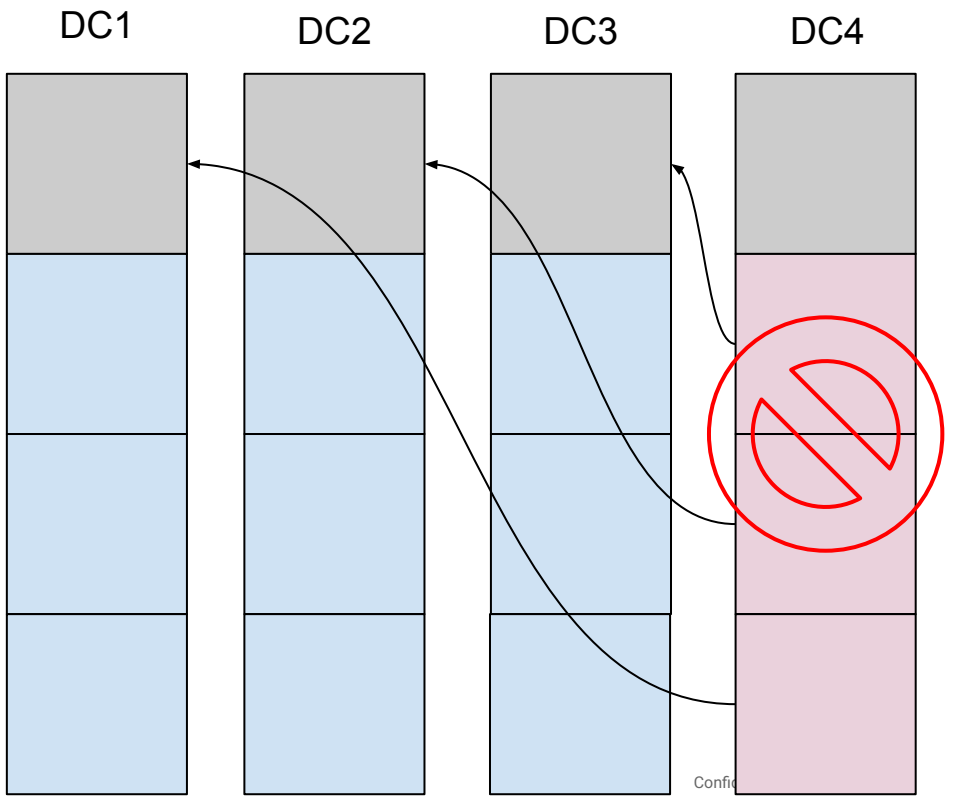
"Holdback" = 1/N

As you increase N:

DCs can each run "hotter" now (better utilization)

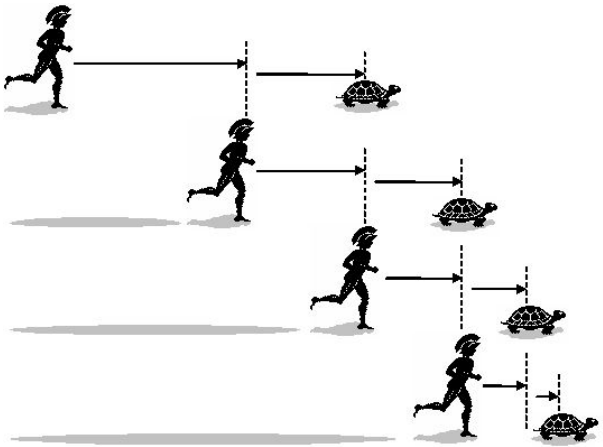
Better global spread should also result in better latency (faster experience) to global users

If each DC is totally independent, your availability "nines" improve dramatically (and are actually capped by the loadbalancer/network),



Zeno's 2nd Paradox

"Achilles and the Tortoise"



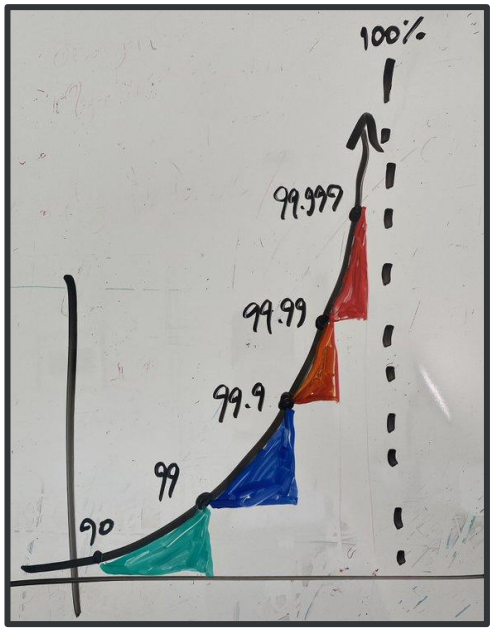
ibmathsresources.com/2018/11/30/zenos-paradox-achilles-and-the-tortoise-2/

- A story to describe **asymptotes**
- Seemingly obvious setup (demigod vs animal)
- Subtle questions arise (how close can we measure?)

➔ Helps us understand the subtlety of "nines"

➔ i.e. 99.99% is **very close to** 100%, unless you look *closely*

➔ Each leg of the race gives us **diminishing returns** – just like more 9s

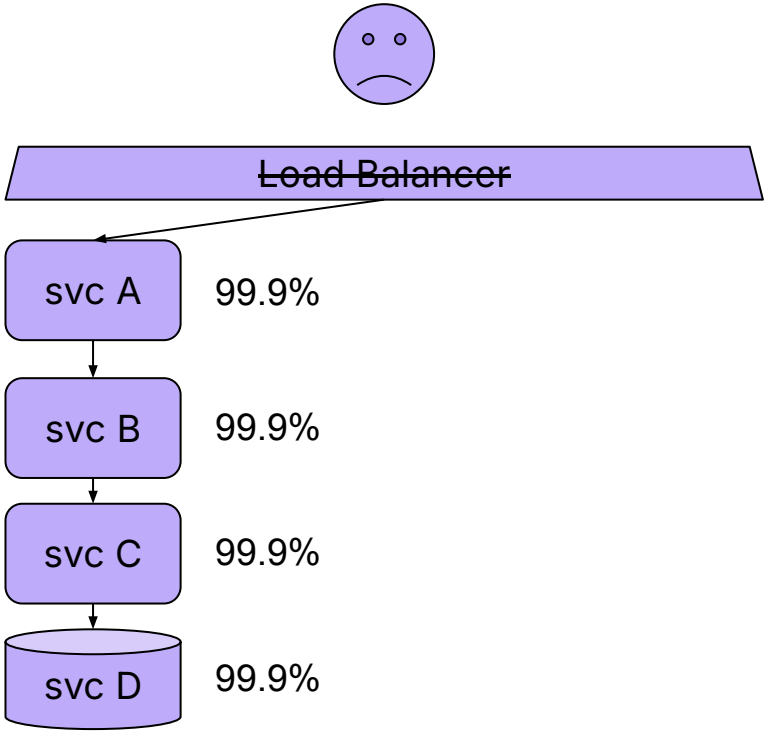


Serial Services

What if you have services that depend on each other, in a "straight line"?

3 nines @ depth 4 gets us "2.6" nines:
`(0.999, 0.999, 0.999, 0.999) = 0.999^4 =` **`99.6%`**

SLO^depth



So what? Your **architecture choices** can have *more* of an impact than the SLOs of your dependencies.

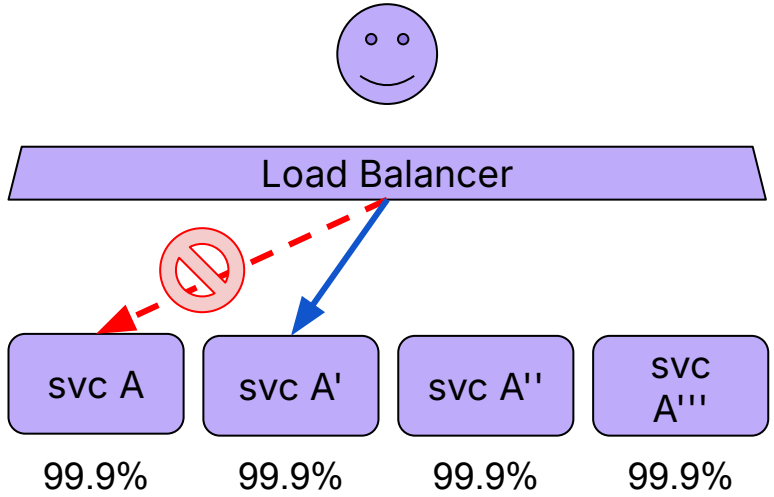
Redundant Services!

What if you have **independent copies of the same service**? As long as **one** is up, you're happy! Now we're talking!

Now your outage only has the probability of all N services failing **at the same time.** (*ahem: presuming automatic retries*)
Our failure_ratio is just: 1 - SLO

Given 4 dice, you have to **roll four ones** in order to fail.
The odds of this are: (1/6 * 1/6 * 1/6 * 1/6) = 0.00077

1 - failure_ratio ^ redundancy




```
1 - (0.1% * 0.1% * 0.1% * 0.1%) =
1 - .001^4 = 99.99999999...%  (12.6 nines!)
```