% whoami

- has worked here **17 years, 9 months**
- last joined **5 years, 2 months ago**
- first joined **19 years, 5 months ago**
- has joined **twice**
- has had **7 roles**
- has worked from **8 offices**
- has worked from **2 countries**

Steve McGhee
Reliability Advocate

@stevemcghee

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

Production Excellence

let's do this backwards.

What the heck?!? ← Incident status

What was that?? ← Postmortem

How’s it going? ← Periodic reviews

How's it going? Periodic reliability reviews

- Where decisions are made!
  - Learn from the past
  - Plan for the future
- Scheduled and templated
  - Twice a year
    (or thereabouts; adjust as needed)
- Attendees:
  - Decision makers (management; executives)
  - Technical leads from the team
    - Not just SRE; all "owners," including Devs, Product, etc.
  - SRE experts (from beyond the team)

How's it going? Periodic reliability reviews

Look back

Look ahead

Look all around

How's it going? Periodic reliability reviews

Look back: Incident report

- Major incidents
  - Where did they happen?
    - Are there trends? Clusters?
    - Specific teams that struggle?
  - What was the impact?
    - Is it concentrated on particular regions? Customers? User types?
  - Action Items
    - Are they getting done?

Beware of recency bias

How's it going? Periodic reliability reviews

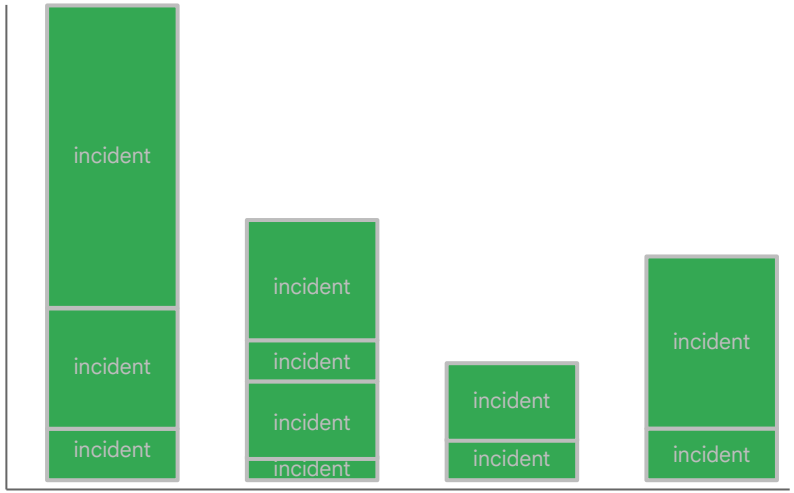
Look back: SLO rollup

- Restate SLO targets
  - Assess SLO compliance: targets hit/missed
- **Analyze upstream and downstream dependencies**
  - This service's impact to others; other services' impact to this
- Consider revising targets

How's it going? Periodic reliability reviews

Look back: Incident report

- Don't focus on incident counts
  - Each incident is unique
- Do focus on the aggregated impact of incidents
  - Revenue lost
  - Error budget burned
  - Blast radius
  - Teams/services affected
  - *Look for patterns*



How's it going? Periodic reliability reviews

Look back: Team health

- People Does the team have the right mix of skills? What training can we offer?
- Cognitive Load Is the team at capacity, or can they onboard additional services?
- Toil Are we doing the right amount? Are we learning from it?
- Interrupts Consider a paging "budget" (e.g. max 2 per on-call shift)
- Morale A happy team is a reliable team

How's it going? Periodic reliability reviews

Look ahead

- Forecast: demand
  - What capacity will be needed?
  - What other factors will become relevant?
- Forecast: work to be done
  - What's in the reliability backlog? What might prevent it from getting prioritized?
- Plans
  - Are stakeholders expecting improved reliability? Can we achieve it?
  - What launches are coming, for this team or related teams?
  - What known future bottlenecks/deprecations/etc. can we prepare for?
  - How does the team's work align to (ongoing or emerging) organizational strategy?

How's it going? Periodic reliability reviews

Look all around

- Dependencies
  - Who is dependent on this team? Are we supporting them well?
  - Who does this team depend on? Are they supporting us well?
- Platforms and tools
  - Does this team use common standards?
  - Does this team contribute back to the ecosystem?
    - PRs to upstream projects; creating/maintaining tools; tech talks or consultation
- Learning
  - Who can learn from this team? Who can this team learn from?

How's it going? Periodic reliability reviews

Requests and proposals → decisions!

- **Do we need to change our plans?**
  - Product Roadmap
  - Communications
  - Protocols
  - Resourcing
  - Team structure
  - Engagement model
- **Meeting outcome: updated plans**
  - e.g. OKRs or other planning artifacts; staffing; budgets

Reliability is hard.

When we talk about reliability, it helps to...

- Respect the inevitable
  - Incidents will happen
- Practice blamelessness
  - Assume good intent
- Share insights broadly
  - Nurture a learning community
- Reflect and iterate

Emergency Incident Response

Planet-Scale Distributed Systems

Service Level Objectives (SLOs)

Systems Engineering

Global Storage

Load Balancing

Google

Monitoring

Availability

Embracing Risk

Blameless Failures

Site Reliability Engineering

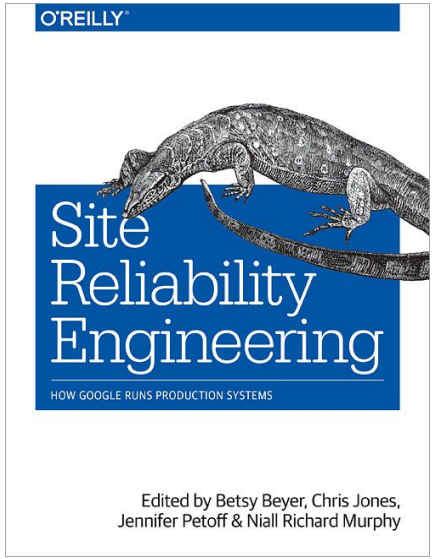
Software Engineering

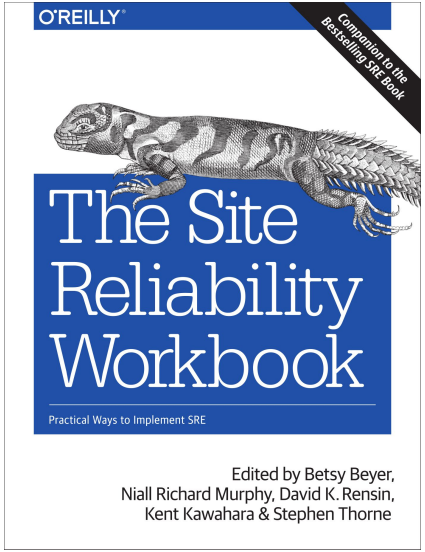
Automation

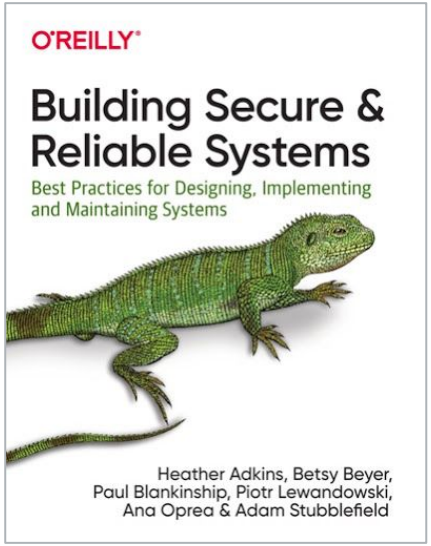
"Hope Is Not A Strategy"

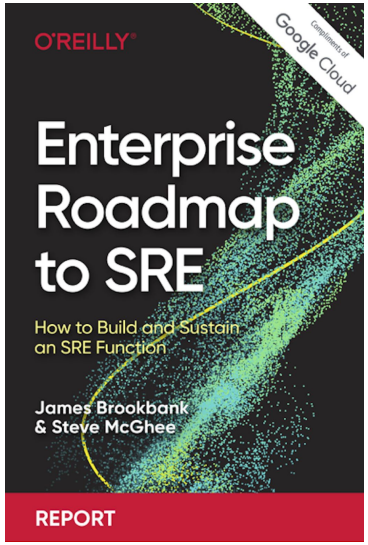
sre.google

Find Google SRE publications—including the SRE Books, articles, trainings, and more—for free at sre.google/resources.









SLO Calculus

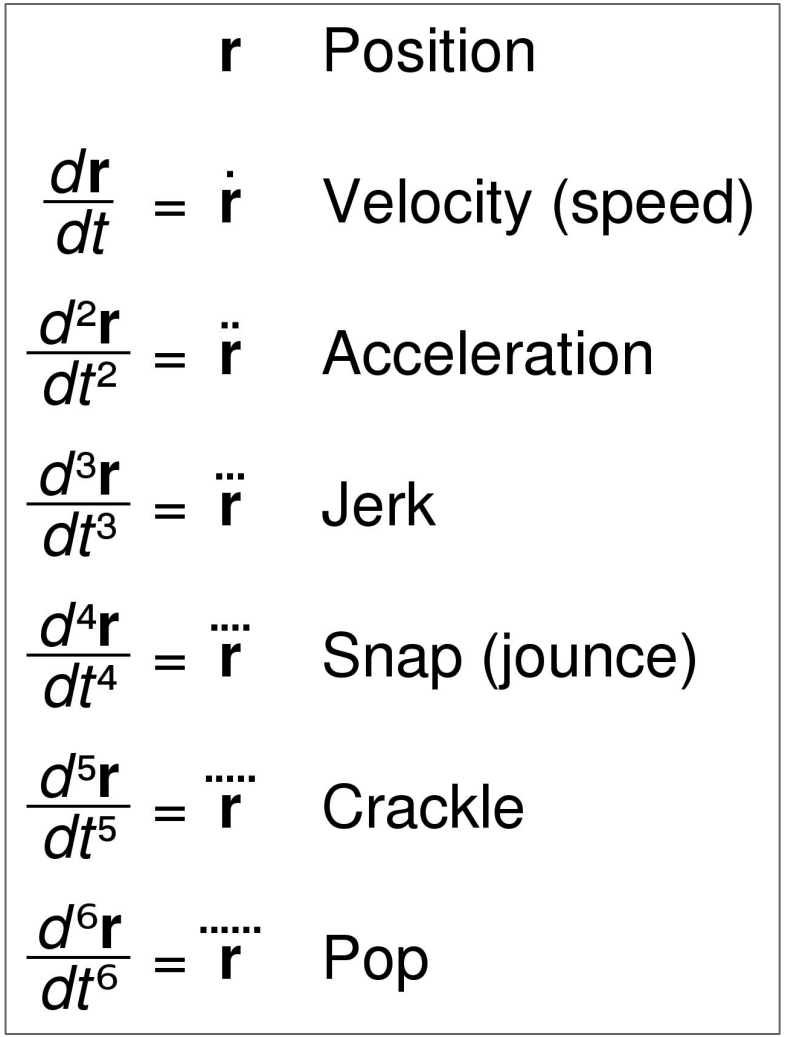
count ~ "num-errors.service"

rate() = SLI

ratio-rate() = SLO

ratio-rate() over time = budget

budget "burn" over time = alert



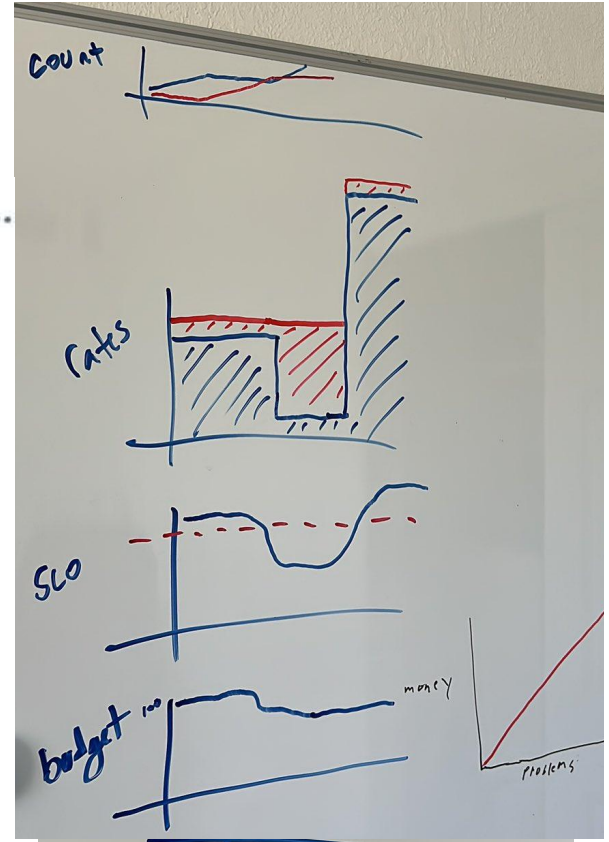
Steve McGhee
@stevemcghee

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 1 12

What is “Reliability”?

Users are happy because we are meeting their implicit and explicit expectations

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:

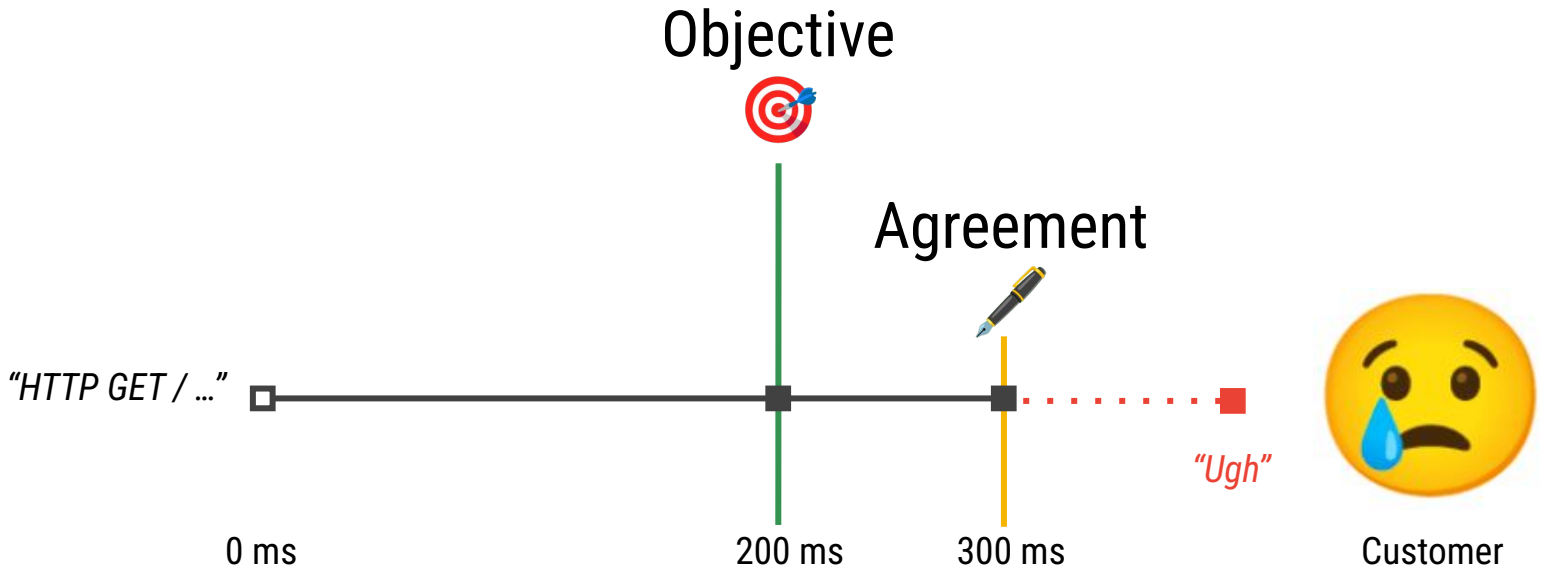
$$\text{SLI} : \left(\frac{\text{good events}}{\text{valid events}} \right) \times 100\%$$

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/QKzvT8_hBZM

~~Bad~~ Naive Math

my users expect 99.0%

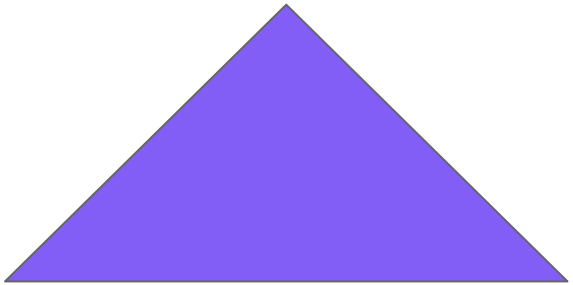
so my webserver should be 99.**9**%

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

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

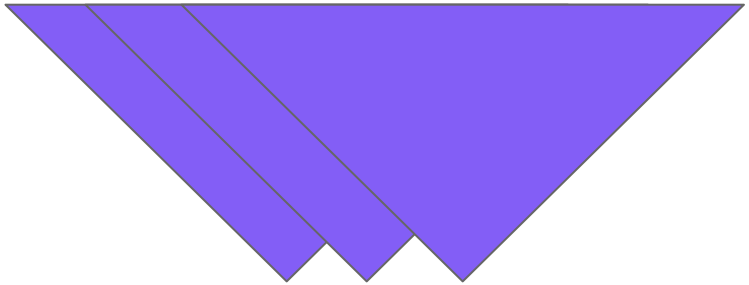
… but what if i have *more* layers? 🤯

Context: The Pyramids



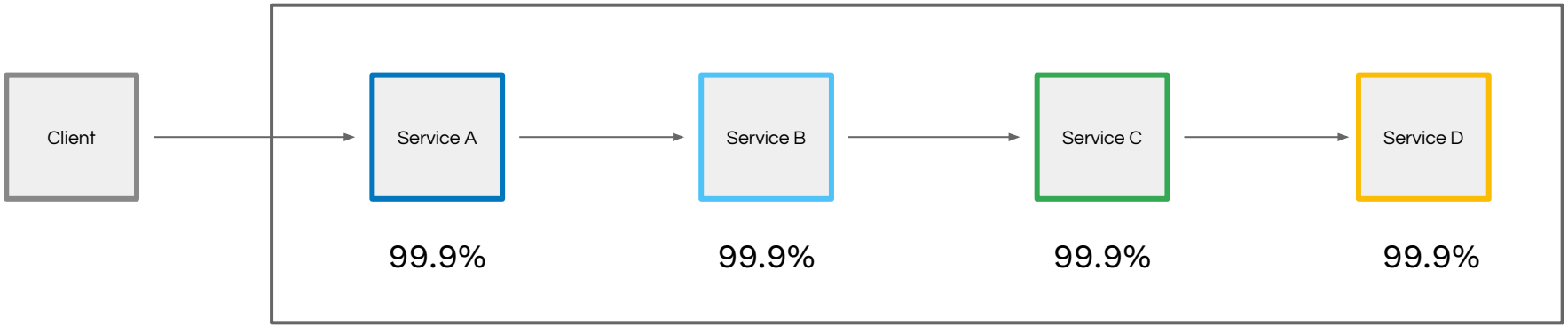
Component-level reliability:

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

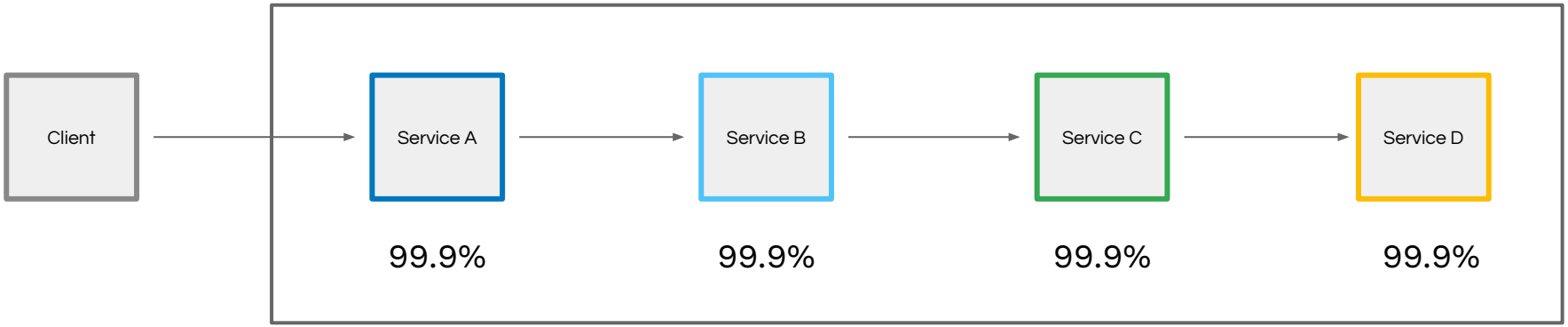


Scalable reliability:

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

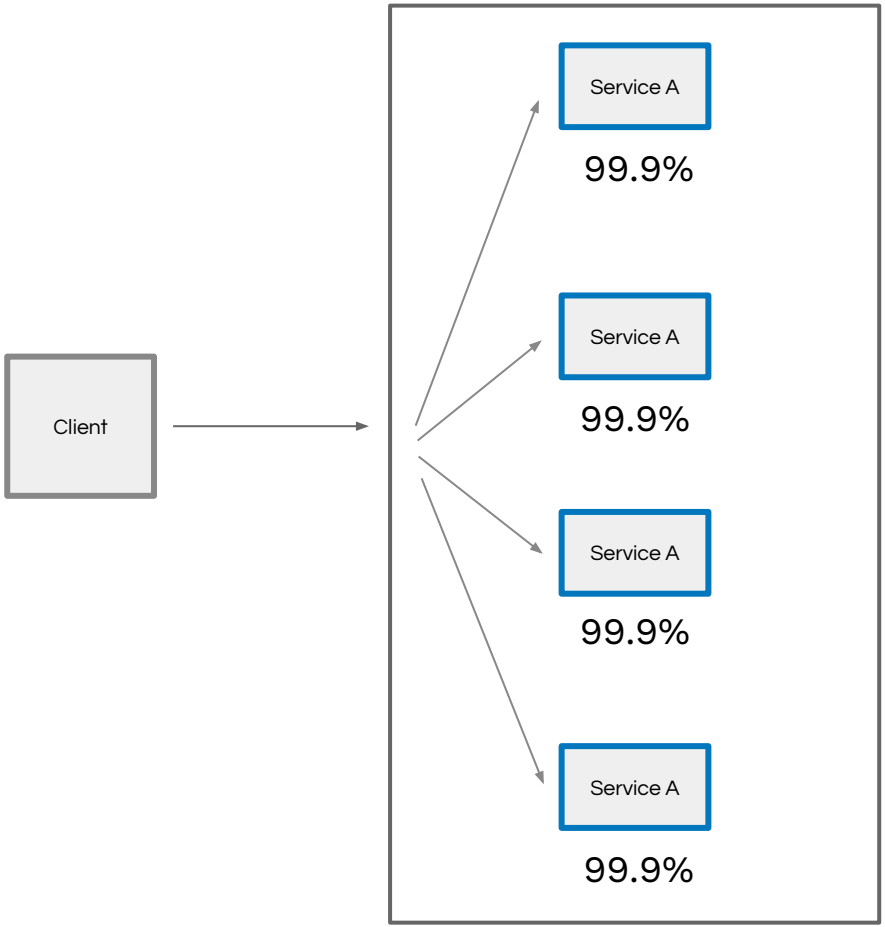


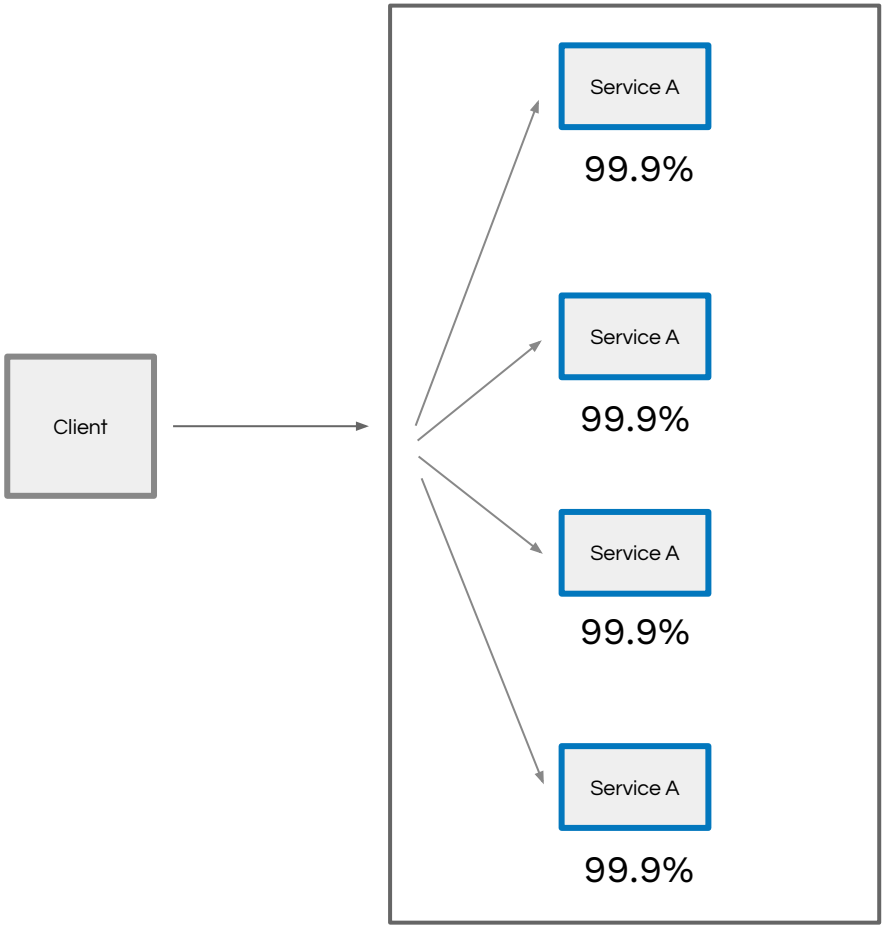
Intersection (or serial)



0.999 x 0.999 x 0.999 x 0.999

99.6% SLO





Union (aka parallel)

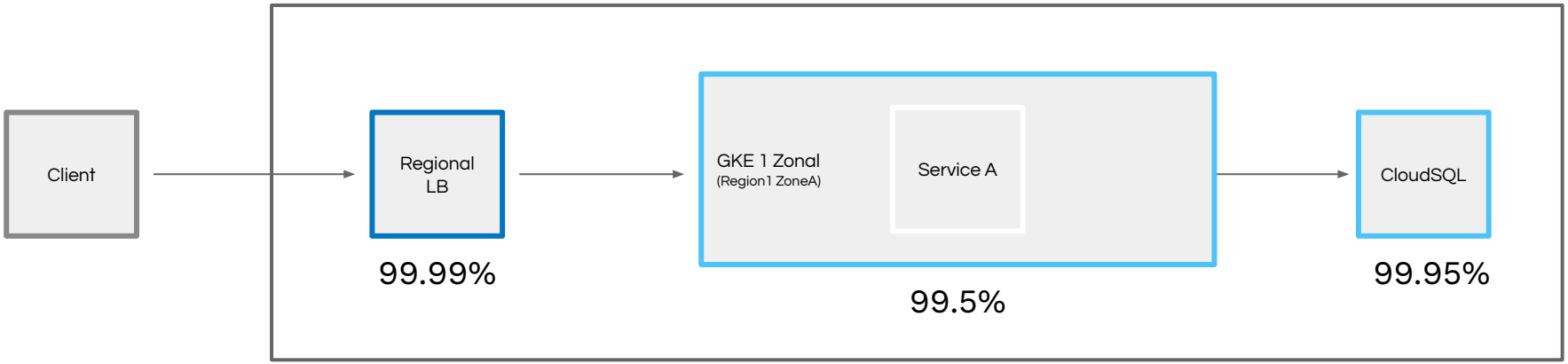
$1 - (0.001)^{4}$

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

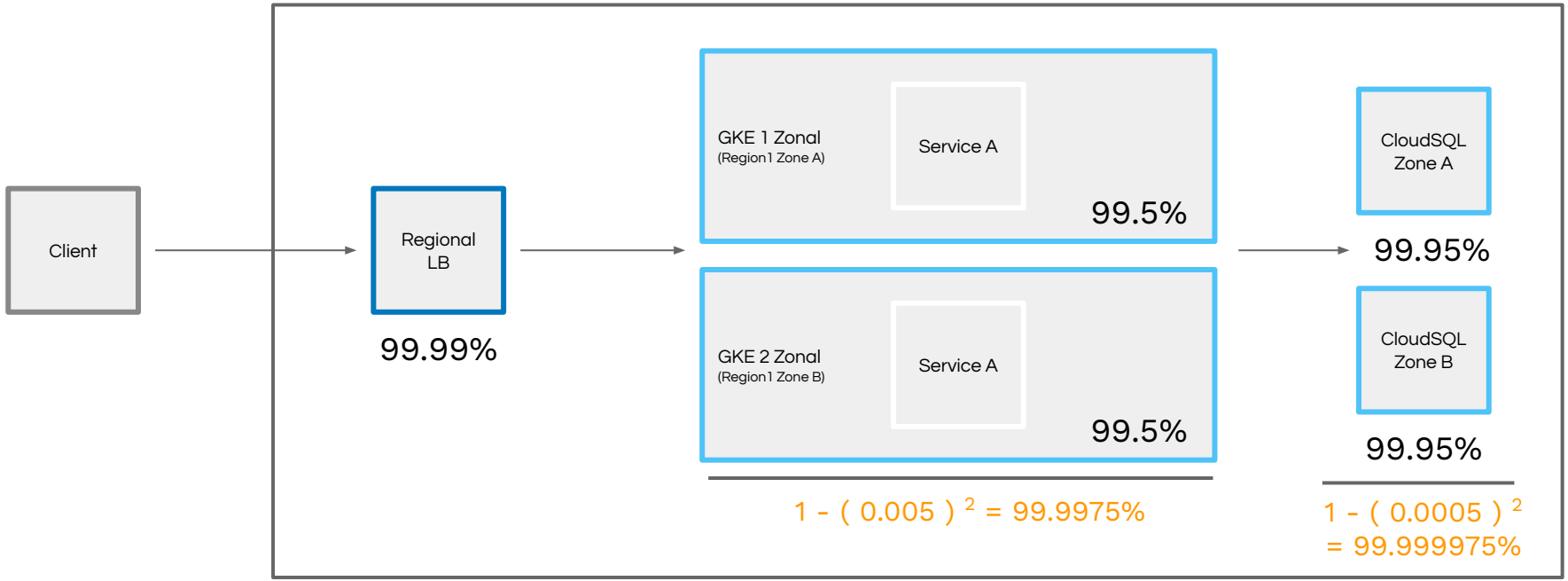
Archetype 2.1



0.9999 x 0.995 x 0.9995

99.44% LIMIT

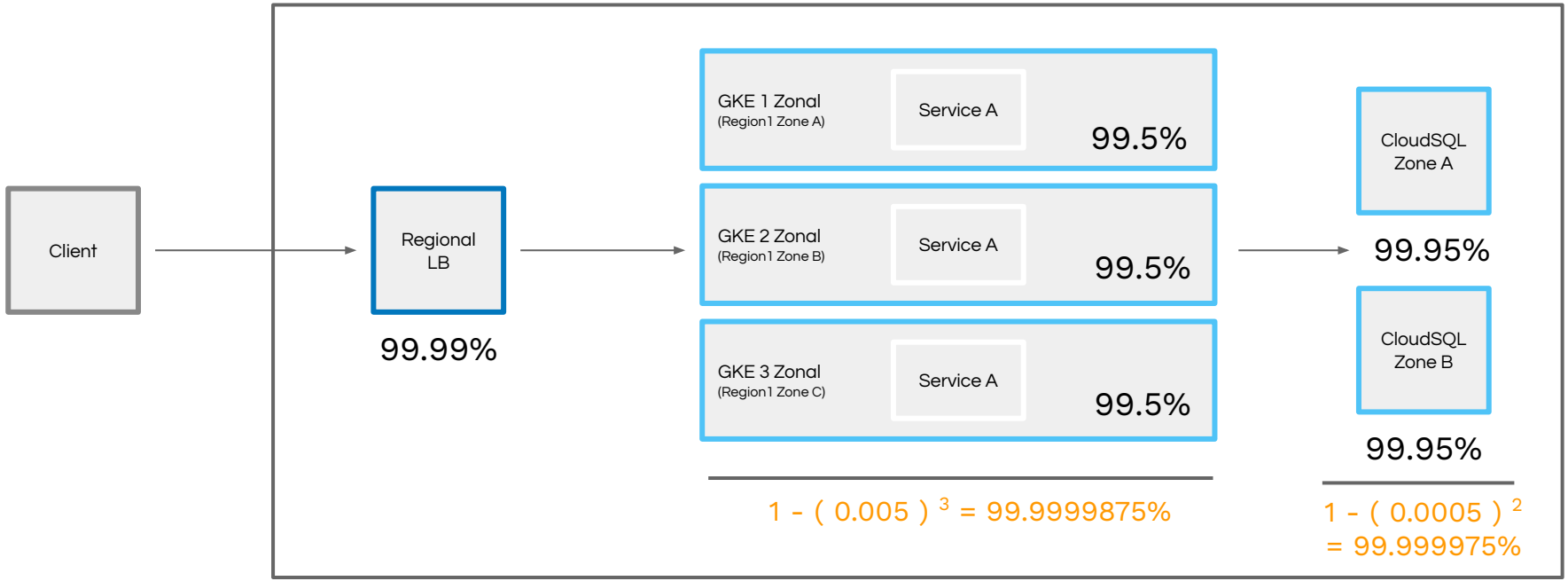
Archetype 2.2



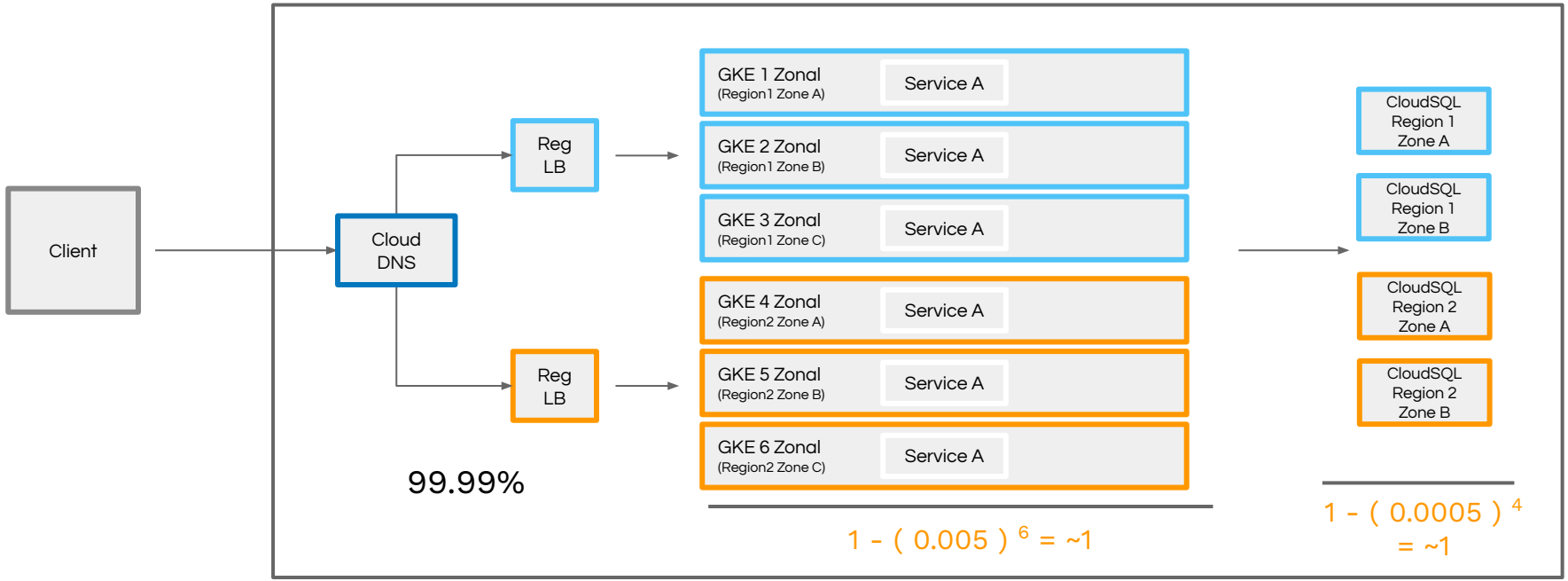
0.9999 x 0.999975 x 0.99999975

99.98% LIMIT

Archetype 3.1



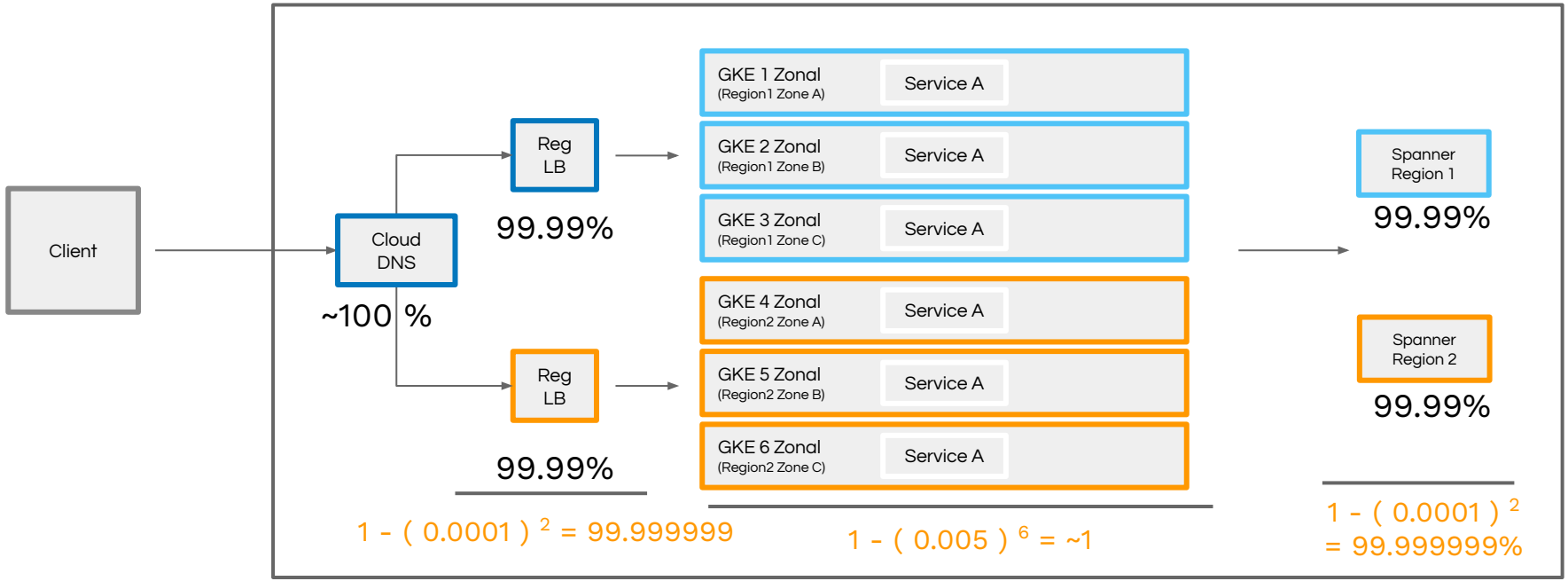
Archetype 3.2



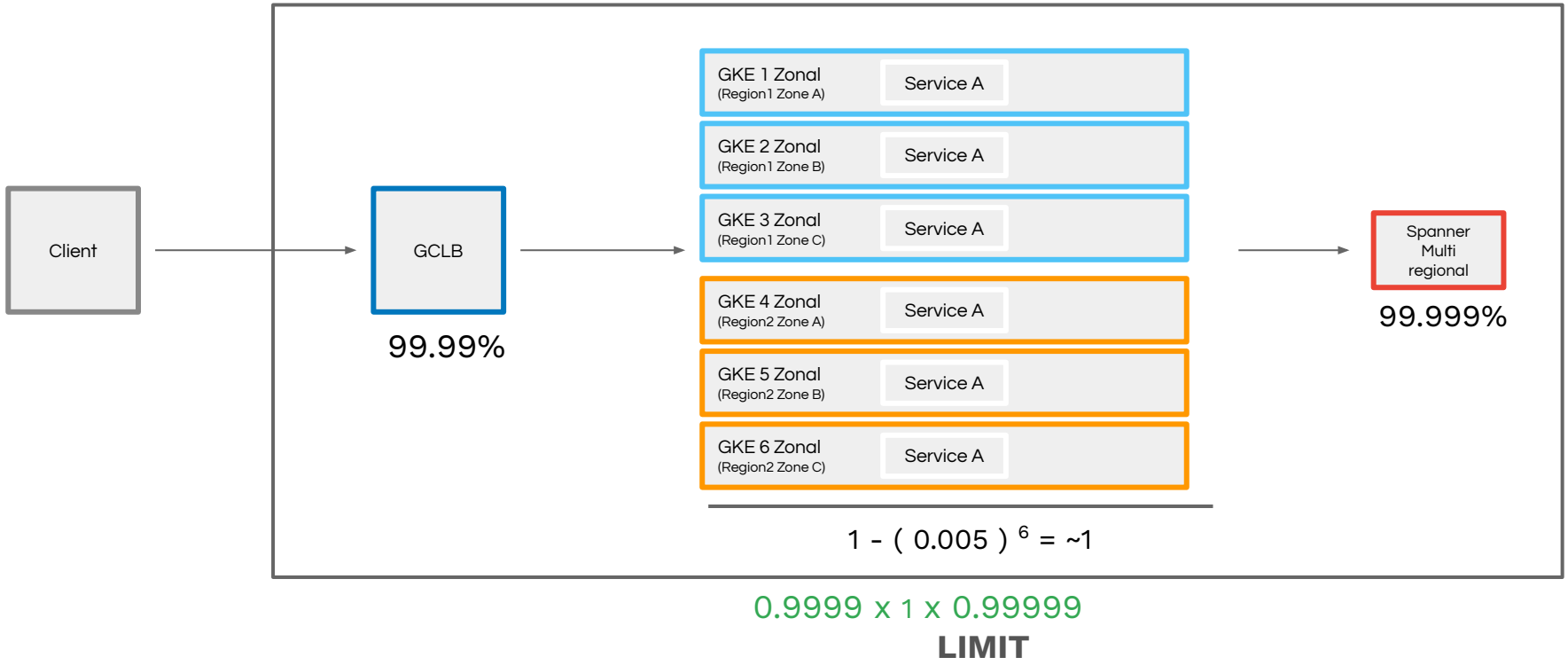
0.9999 x ~1 x ~1

99.99% LIMIT

Archetype 4.3 Isolated Regions with Cloud Spanner



Archetype 5.2 Global with Cloud Spanner (Multi regional)



A colleague asked me:

"When should you define an SLO for a system vs it's components?"

I hope what you take away from this talks is:

You should design a **system** at "the front door" but it's a common mistake to follow Conway's Law and define it at team boundaries, then get frustrated by the "bad math" that ensues.

Build a platform that lets you focus on customer happiness. All else will follow.

https://unsplash.com/photos/dBaz0xhCkPY

Those Other SLOs

@stevemcghee
Reliability Advocate, SRE
Google Cloud

SLOs in one slide

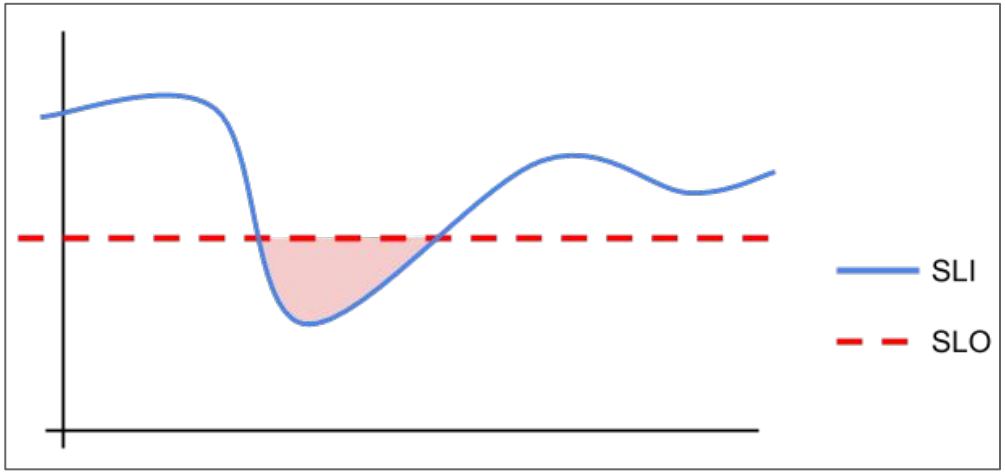
A **ratio-rate** of `good/total`, measured over a time duration.

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**



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.

https://cloud.google.com/blog/products/storage-data-transfer/understanding-cloud-storage-11-9s-durability-target

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 :)

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.

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

Bonus / R&D - LLM for SLIs

Bizarrely, the hardest part of SLOs is not the technology,

it's deciding **what** to measure.

Lil' help, bot?

check_sli > cuj01.txt

```
1    As a shopper I want to see items for sale in the virtual store.
```

check_sli > cuj01_non_func.txt

```
1    The product listing page must load quickly
2    The product listing page must be available most of the time
3    The system must support up to 1,000 simultaneous users browsing the product listing page without degradation
4    All communication between the client and server must be encrypted using SSL/TLS
```

```go
func checkSLI(w http.ResponseWriter, r *http.Request) {
    ctx := context.Background()
    client, err := genai.NewClient(ctx, projectID, region)
    if err != nil {
        fmt.Println("Error to load the project: %v", err)
        fmt.Fprintf(w, "Error to load the project: %s", err)
        return
    } else {
        gemini := client.GenerativeModel(modelName)
        prompt := ReadFile(promptReq)
        business_request := ReadFile(requirement)
        non_functional_req := ReadFile(nonfuncreq)
        code := ReadRowData(product_catalog)

        replacer := strings.NewReplacer(
            "[critical_user_jorney]",
            business_request,
            "[list_of_non_functional_req]",
            non_functional_req,
            "[implementation_code]",
            code)

        prompt_text := replacer.Replace(prompt)

        resp, _ := gemini.GenerateContent(ctx, genai.Text(prompt_text))
        content_part := resp.Candidates[0].Content.Parts[0]
```

Critical User Journey: As a shopper I want to see items for sale in the virtual store

The selection of SLIs was based on the following trade-offs:

â€¢ Fidelity: We prioritized SLIs that closely represent the user experience, such as latency and availability.
â€¢ Coverage: We aimed for SLIs that provide broad coverage of user interactions, considering both individual page loads and overall system performance.
â€¢ Cost: We considered the cost of implementation and maintenance, opting for metrics that can be collected and analyzed cost-effectively using existing Google Cloud resources.

Group	SLI	Description	Implementation	Collection	Metric Resource	Metric Category	Metric	Aspirational SLO
Performance	Product Listing Page Load Time (95th percentile)	Measures the time it takes for the product listing page to fully load for 95% of users.	Measure the time it takes for the product listing page to fully load, using a user-agent simulation tool that emulates real user behavior. This can be achieved using browser automation frameworks like Selenium or Puppeteer.	Use a monitoring agent or custom instrumentation to collect the page load times. Send this data to Google Cloud Monitoring for analysis.	Kubernetes Cluster	Load	Kubernetes Pod CPU Usage	95th percentile load time < 2 seconds
Performance	API Latency (99th percentile)	Measures the average latency of API calls for retrieving product data, considering the 99th percentile to account for outliers.	Use distributed tracing tools like Jaeger or OpenTelemetry to track the time taken for each API call in the product listing journey.	Use a monitoring agent or custom instrumentation to collect API call latencies. Send this data to Google Cloud Monitoring for analysis.	Kubernetes Cluster	Latency	Kubernetes Pod Network Egress Rate	99th percentile API latency < 500 milliseconds
Availability	Product Catalog Service Availability	Measures the percentage of time the Product Catalog Service is available and responding to requests.	Implement health checks to periodically probe the Product Catalog Service and determine its availability. Integrate these checks with Google Cloud Monitoring for automated monitoring and alerting.	Use Google Cloud Monitoring to monitor the health checks and collect availability data.	Global Service Vm Client	Availability	Global Service Vm Client Request Latency Distribution	Service availability > 99.9%

1. Observations:
    1. The product listing page load time metric is crucial for user experience, and the 95th percentile captures the experience of most users.
    2. API latency is another important factor affecting user perception.
    3. The availability of the Product Catalog Service is essential to ensure uninterrupted service.
2. Step-by-Step Implementation:
    1. Create a new SLO within Google Cloud Monitoring.
        1. Select your project from the list.
        2. Select Monitoring from the navigation menu.
        3. Select SLO from the left navigation panel.
        4. Click Create SLO.
    2. Configure the SLO:
        1. Choose a relevant metric from the Metric dropdown menu (e.g., Kubernetes Pod CPU Usage).
        2. Configure the Time Series Aligment and Time Window according to your needs.

requirements.txt (**THE PROMPT**)

You are an expert developer working as SRE and bilingual in Portuguese and English.

You received a critical user journey to evaluate and help determine the Service Level Indicators (SLI) with the architecture and development teams.

The critical user journey is [critical_user_journey]

The product area has determined that the non-functional requirements:
[list_of_non_functional_req]

The first line of the return must necessarily be a header with the data of the critical user journey evaluated, using the font "Roboto, sans-serif", font color "Black" in bold, and size "30px".

The choice of implementation method for each SLI should consider trade-offs between the following:
- Fidelity: the accuracy with which it captures the user experience.
- Coverage: the proportion of user interactions measured.
- Cost: the financial value and engineering time required to create and maintain the solution.

In the evaluation for selecting SLIs, the following possibilities must be considered:
- Availability
- Latency
- Quality
- Update
- Coverage
- Correction
- Throughput
- Latency
- Performance

You should create a unique HTML table with the SLIs with the most relevant reliability dimensions for the critical user journey.

The metrics for the assigned indicator can be collected and exported from Google Cloud Console through the resources:
- Vm Instance
- Autoscaler

FIN ACK

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

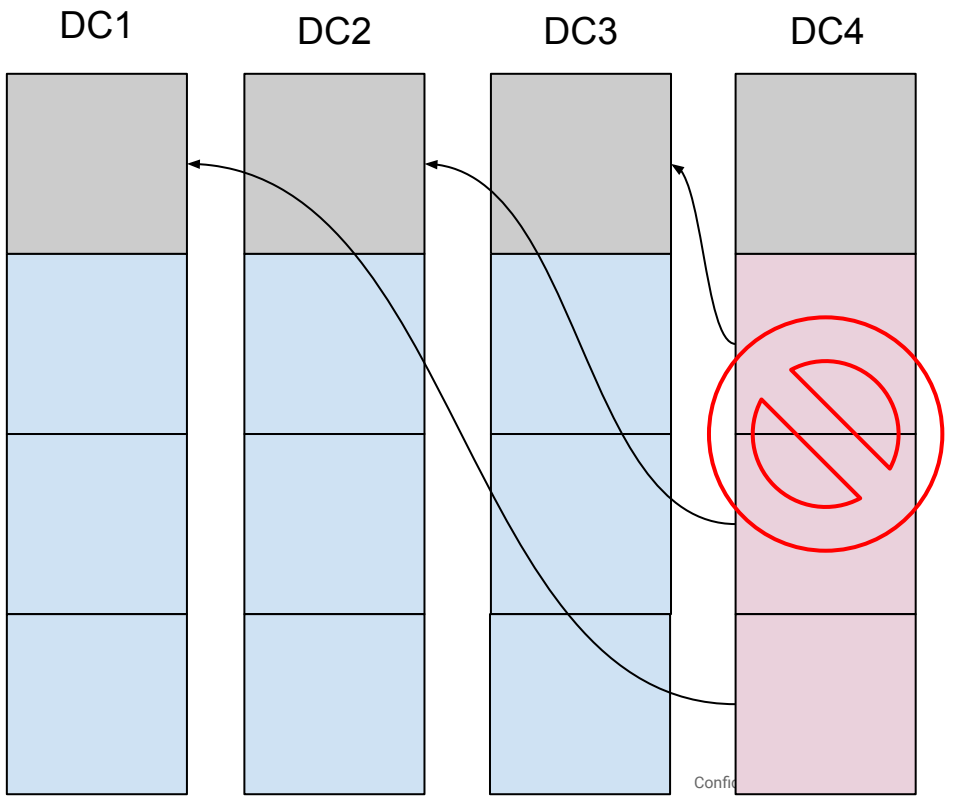
"Holdback" = 1/N

As you increase N:

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

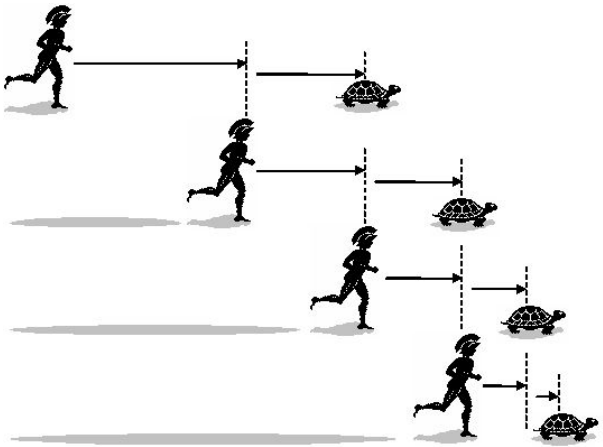
Better global spread should also result in better latency (faster experience) to global users

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"



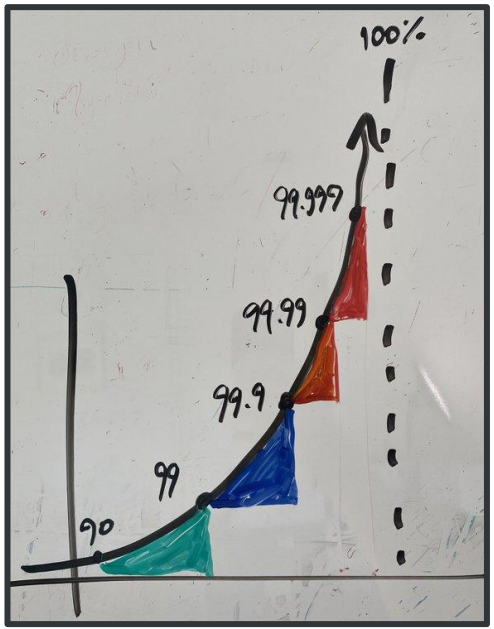
ibmathsresources.com/2018/11/30/zenos-paradox-achilles-and-the-tortoise-2/

- 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

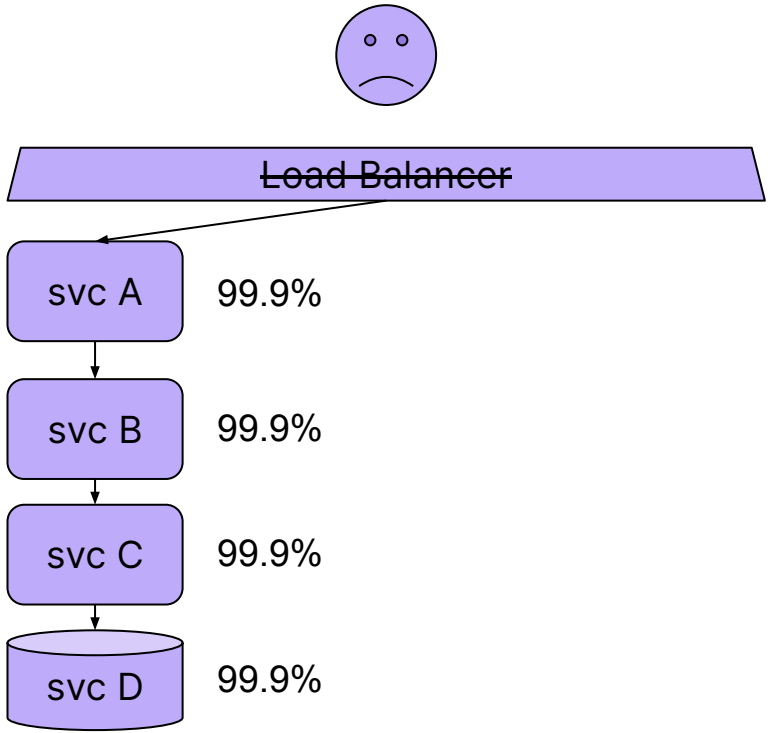


Serial Services

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.

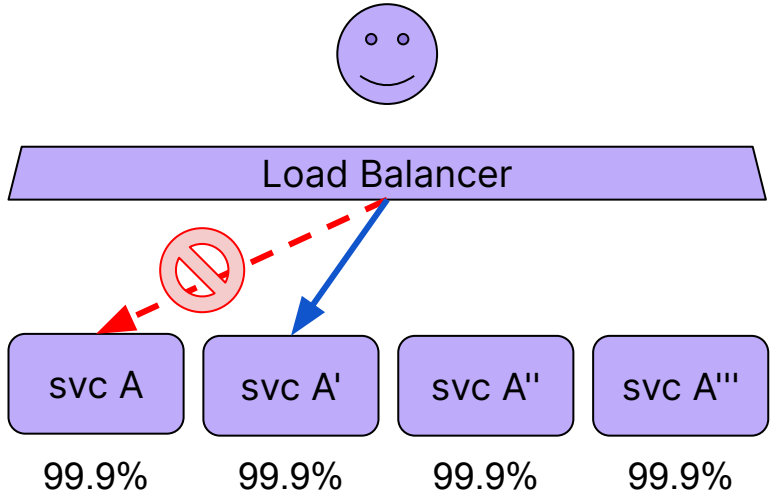
Redundant Services!

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!)
```