Google Cloud

Running Reliable systems

Part 2: Service Level Objective (SLO) Math

HPE Meetups 2022

Meet our speaker

Leonid Yankulin
Developer Relations Engineer, DEE,
Observability lead

at in/minherz, /minherz and /@minherz

Let's begin with...

What I hope you already know: What an SLO **is**.

What I hope you'll learn: How to **use** SLOs. How **not** to use SLOs.

Heads up: Math Ahead. It is just probability.

A challenge of defining SLO

The Front Door SLO

User happiness

- ★ **Available** enough
- ★ **Fast** enough
- ★ **Complete** enough

Meet Expectations –
Don't Expect Perfection

Photo by Evelyn Paris on Unsplash

"Nested" SLOs

"But it's more complicated than that"
"My service depends on other teams"

~~Bad~~ Naive Math

IF user expects	99.0%
THEN webserver should be	99.**9**%
THEN hypervisor should be	99.**99**%
THEN infrastructure should be	99.**999**%

… but what there are *more* layers? 🤯

Photo by Nathan Dumlao on Unsplash

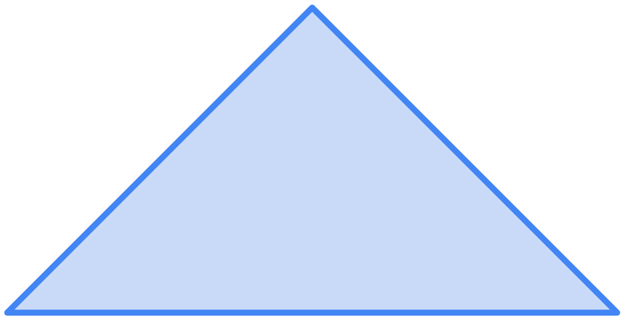
DevOps/IT Strategies: The Pyramids

Component-level reliability

Scalable reliability

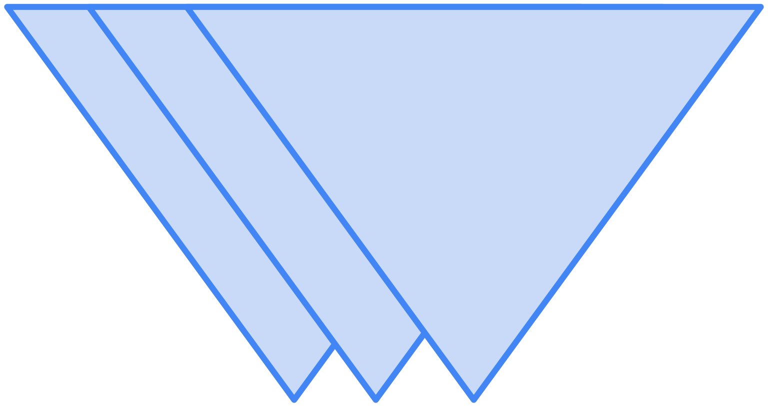
Component-Level Reliability

- Solid base (big cold building, heavy iron, redundant disks/net/power)
- **Each** component up as much as possible
- **Total availability** as goal
- Scales up

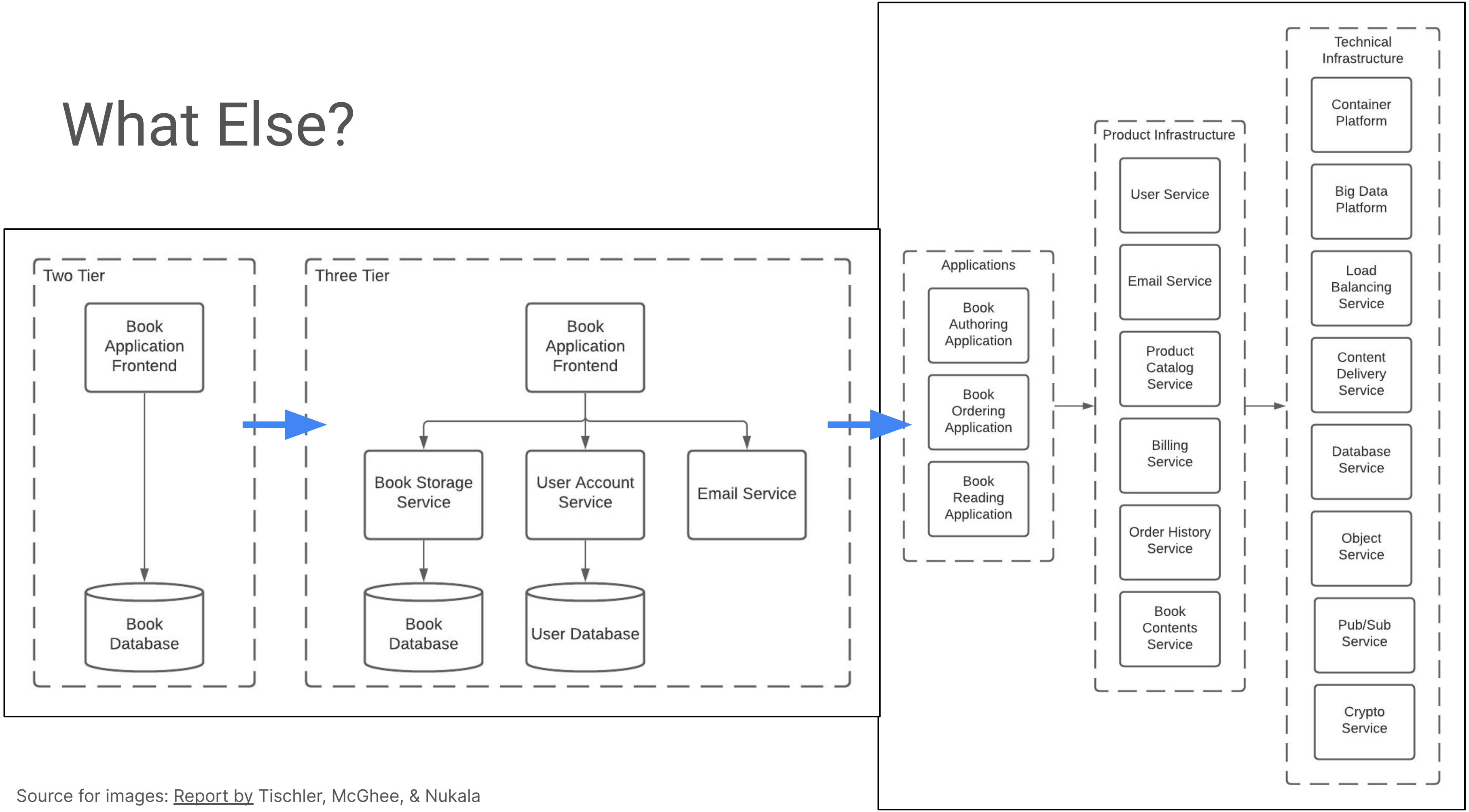


Scalable Reliability

- Less-reliable, but Cost-effective base
- Warehouse scale (many machines)
- Software improves availability
- Aggregate availability as goal
- Scales out



What Else?



Source for images: Report by Tischler, McGhee, & Nukala

Good math to calculate SLO

Probability, Really Quick

One 6-sided dice:

"bad" roll $\frac{1}{6}$ = 0.167...

vs.

"good" roll $\frac{5}{6}$ = 0.833...

For **Four** 6-sided dice:

$\frac{5}{6} \cdot \frac{5}{6} \cdot \frac{5}{6} \cdot \frac{5}{6}$ = 0.482

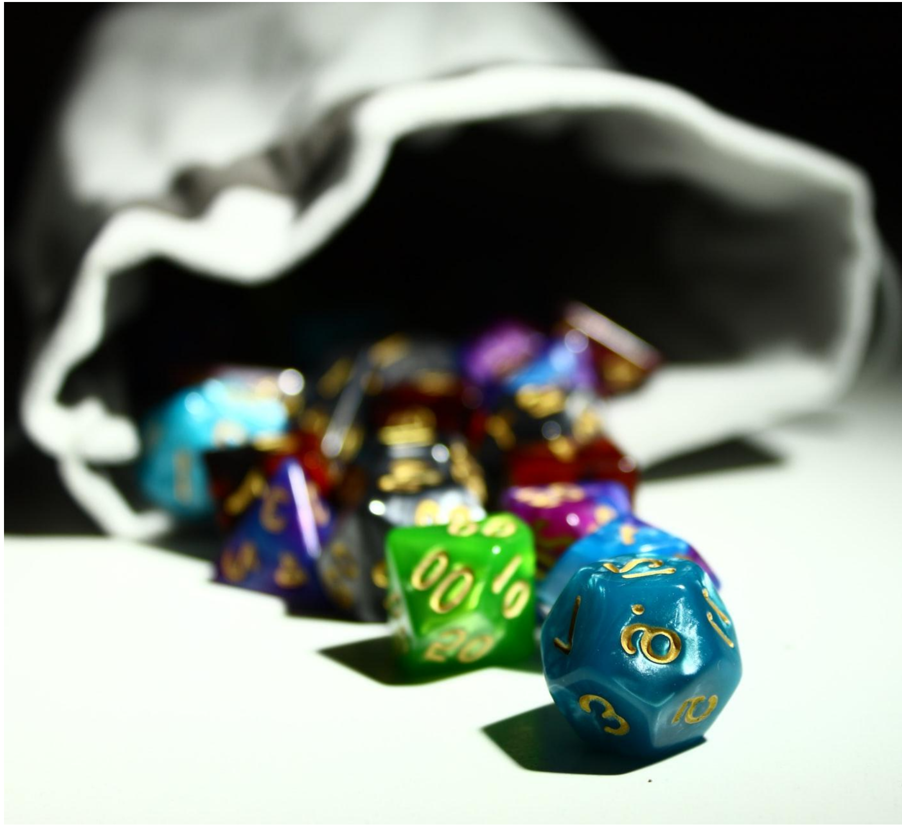


Photo by Alperen Yazgı on Unsplash

"you'll never do as well as the **worst case single throw**"

Probability, Really Quick

Four N-sided dices:

$((N-1)/N)^4$

eg: 10-sided: $(0.9)^4 = 0.6561$

M arbitrary-sided dices:

$(N_1 - 1/N_1) * (N_2 - 1/N_2) * (N_M - 1/N_M)$

eg: two 6-sided, one 10 sided, one 20 sided dice:
5/6•5/6•9/10•19/20 = 0.83•0.83•0.90•0.95 = **0.59**

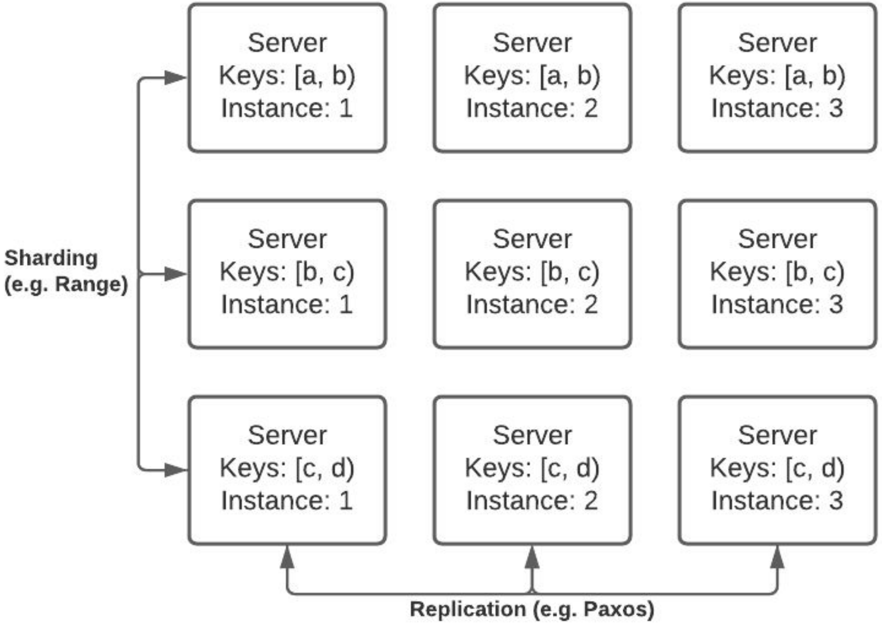
"you'll never do as well as the **worst case single throw**"

Photo by Alperen Yazgı on Unsplash

Good Math Needs a Model

Distributed system model to allow

- Scalability (Horizontal, Vertical)
- Sharding, Partitioning
- Replication, Load Balancing



Source for images: Report by Tischler, McGhee, & Nukala

Serial Services

Sequential dependencies:

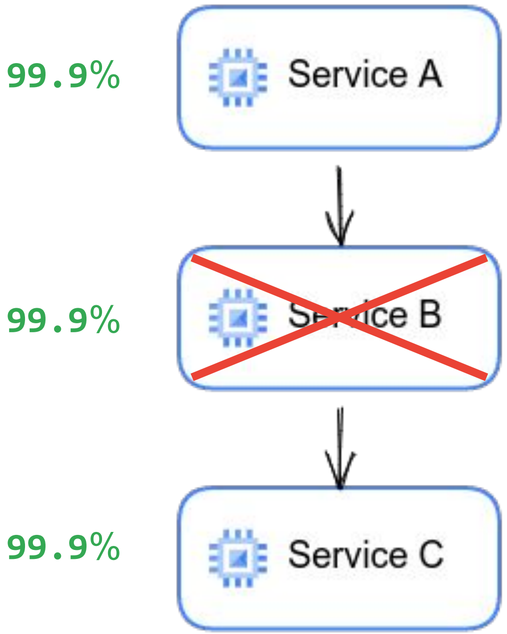
3 nines @ depth 3 gets us "2.7" nines

$$0.999 \cdot 0.999 \cdot 0.999 = \mathbf{0.997}$$

Or **99.7%** a.k.a **SLO^depth**

By the way...

$$0.999 \cdot 0.9999 \cdot 0.99999 = \mathbf{0.9988901}$$



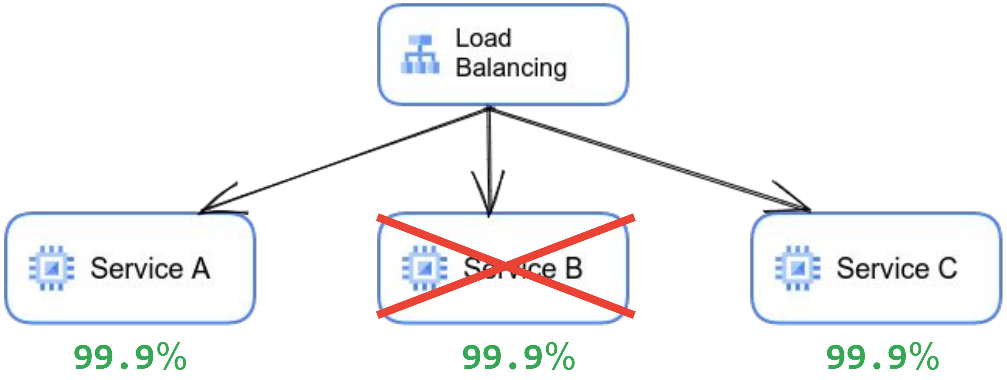
"your **architecture choices** can have more of an impact than SLOs of your dependencies"

Parallel Services

Parallel service composition when they all are needed:

0.999•0.999•0.999 = 0.997

Or **99.7%** a.k.a **SLO^depth**

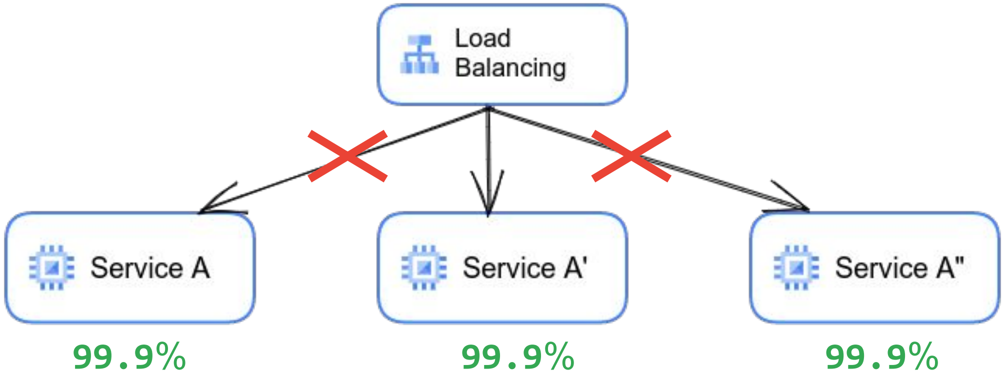


Parallel Services With Redundancy

Computed SLO:

1 - failure_ratio^redundancy

where **failure_ratio** = 1 - service SLO



For 3 copies of the *same* service

$1-(0.001)^3 = .999999999$

Or **99.99...**% (9 nines!!)

Set Theory of SLO

	Intersection availability / **all** dependencies must be available /			**Union availability** / at **least one** dependency is available /	
Component Availability	Number of Components			Number of Components	
	3	10	100	2	3
99%	97%	90%	37%	**4** Nines	**6** Nines
99.9%	99.7%	99%	90%	**6** Nines	**9** Nines
99.99%	99.97%	99.9%	99%	**8** Nines	**11** Nines
99.999%	99.997%	99.99%	99.9%	**10** Nines	**15** Nines

$$SLO_1 \bullet SLO_2 \bullet \ldots \bullet SLO_N$$

$$1-(FR_1 \bullet FR_2 \bullet \ldots \bullet FR_N)$$

Is it for real?
Why aren't we swimming in nines?

Reality Behind the Theory

- Technical bottlenecks
  - Networking and Load Balancing are nines-lynchpin
  - End-to-end SDLC ownership
- Human bottlenecks
  - Mistakes, Churn, Toil
  - Shallow Understanding / Striving for Over-Simplified Learnings

What to do?

- ✓ DO NOT worry about downward implications of your SLOs
- ✓ CONSIDER your customer's happiness first and foremost
- ✓ HELP infrastructure teams understand the new world

Two Models

- ★ Stacks:
  - ○ Multiple copies of real stack
  - ○ No problem to lose one
  - ○ Advantage Load Balancing among copies
- ★ Service Mesh:
  - ○ Hard to accomplish
  - ○ Cognitive & operational cost
  - ○ Better resilience and flexibility

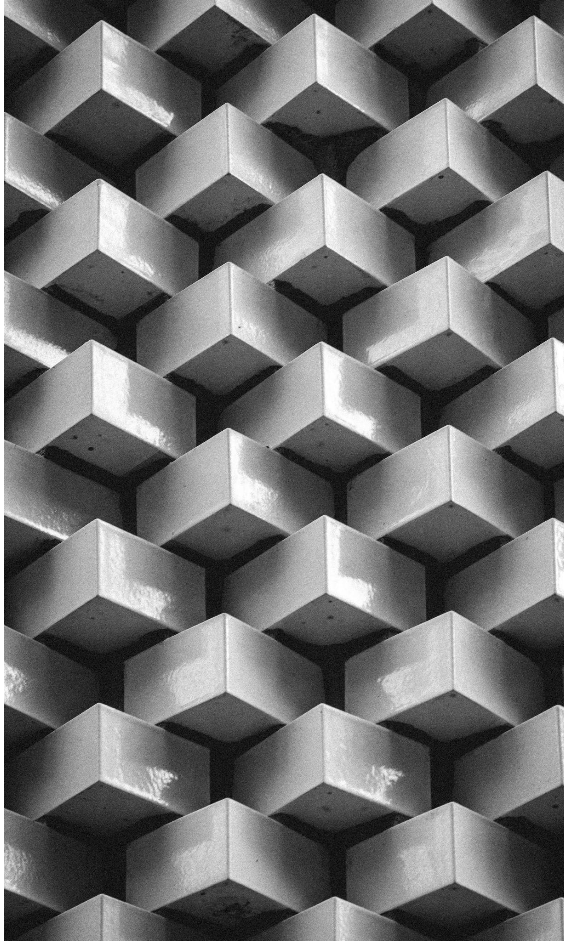
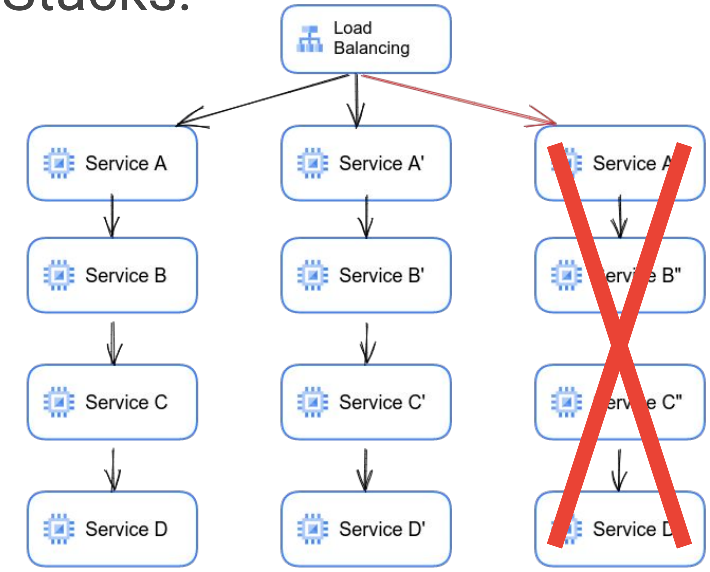


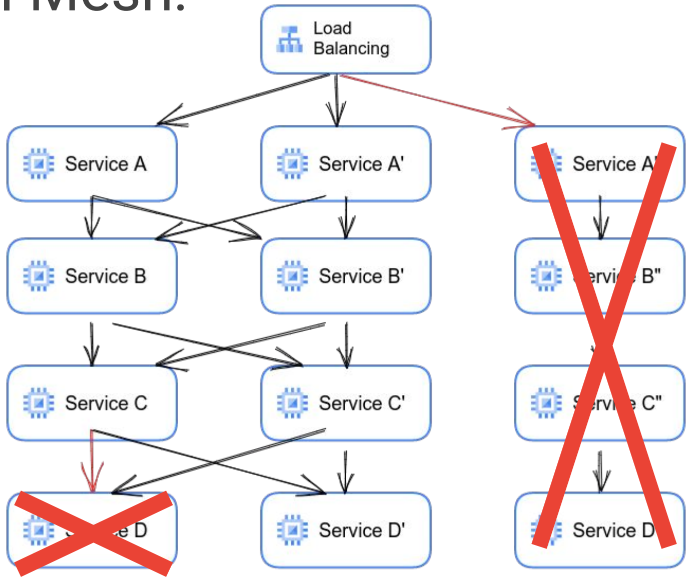
Photo by Crawford Jolly on Unsplash

Two Models

Stacks:



Full Mesh:



99.9%

99.9%

99.9%

99.9%

Gnarly Details

- ✓ Other models
  - ➔ **Y**ou **O**nly **L**ook **O**nce (YOLO)
  - ➔ Megalith
- ✓ Costs
  - ➔ Compute and storage
  - ➔ Operational complexity
  - ➔ Consistency, sharding, replication
- ✓ Further reading
  - ➔ Failure domains and modes
  - ➔ Graceful degradation

Photo by Natalya Letunova on Unsplash

This seems too easy! You're totally right...

Fallacies (so far):

(1) SLOs must get tighter with depth

(2) I need to control the entire stack

Solutions:

(1) Resilience via Engineering!

(2) Do I own load balancer, mobile tower, power grid?

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

Watch "SRE I aspire to be" by @aknin

Closing...

You should design a **system** at "the front door". 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.

Photo by Adam Miller on Unsplash

Thank you

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