Advanced Napkin Math

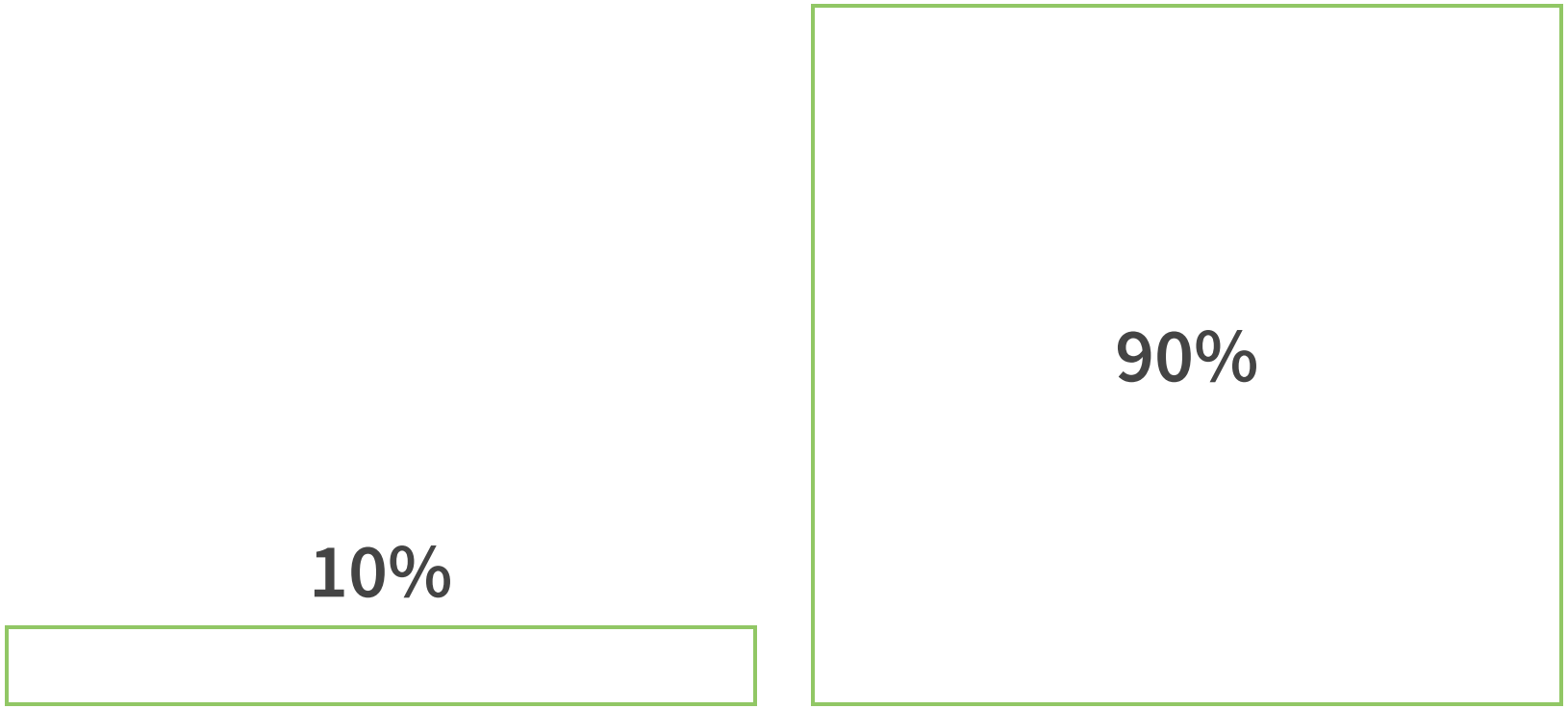
Estimating System Performance from First Principles

Simon Eskildsen @ SRECON, Dublin 2019

“[..] physics teaches you to reason from first principles rather than by analogy. So I said, okay, let’s look at the first principles. What is a rocket made of? Aerospace-grade aluminum alloys, plus some titanium, copper, and carbon fiber. Then I asked, what is the value of those materials on the commodity market? It turned out that the materials cost of a rocket was around two percent of the typical price.”

Elon Musk

Example: Why is my Cloud bill $100,000?



Base Rates: Cost

Loosely based off of https://cloud.google.com/products/calculator, similar for other Cloud providers.

- **CPU: $10 / core / month**
- **Memory: $1 / GB / month**
- **SSD: $0.1 / GB / month**
- **Disk: $0.01 / GB / month**
- **Cloud-storage (S3): $0.01 / GB / month**
- **Network: $0.01 / GB / month**

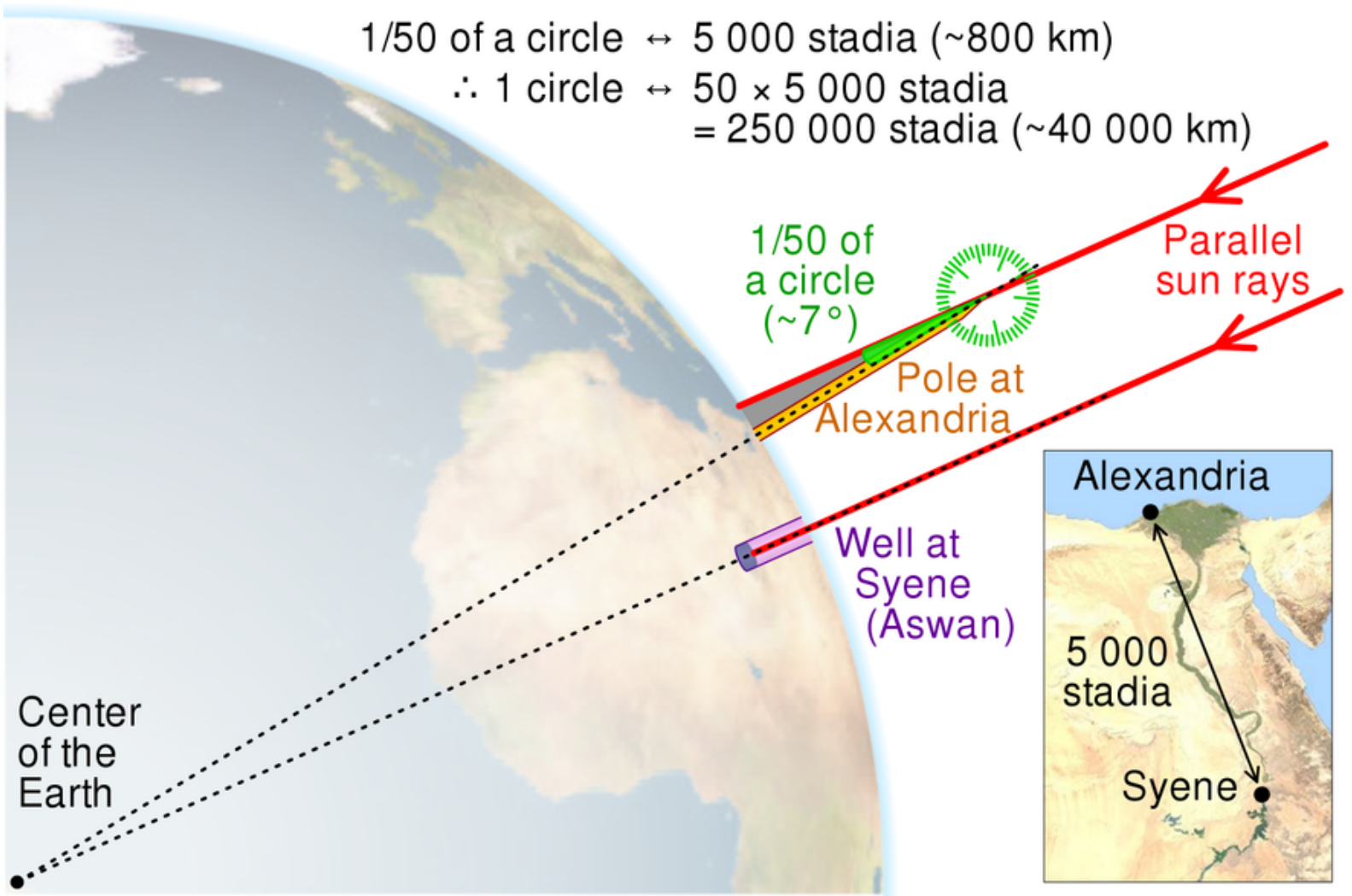
  Between zones, egress, between regions

Base Rates: Application

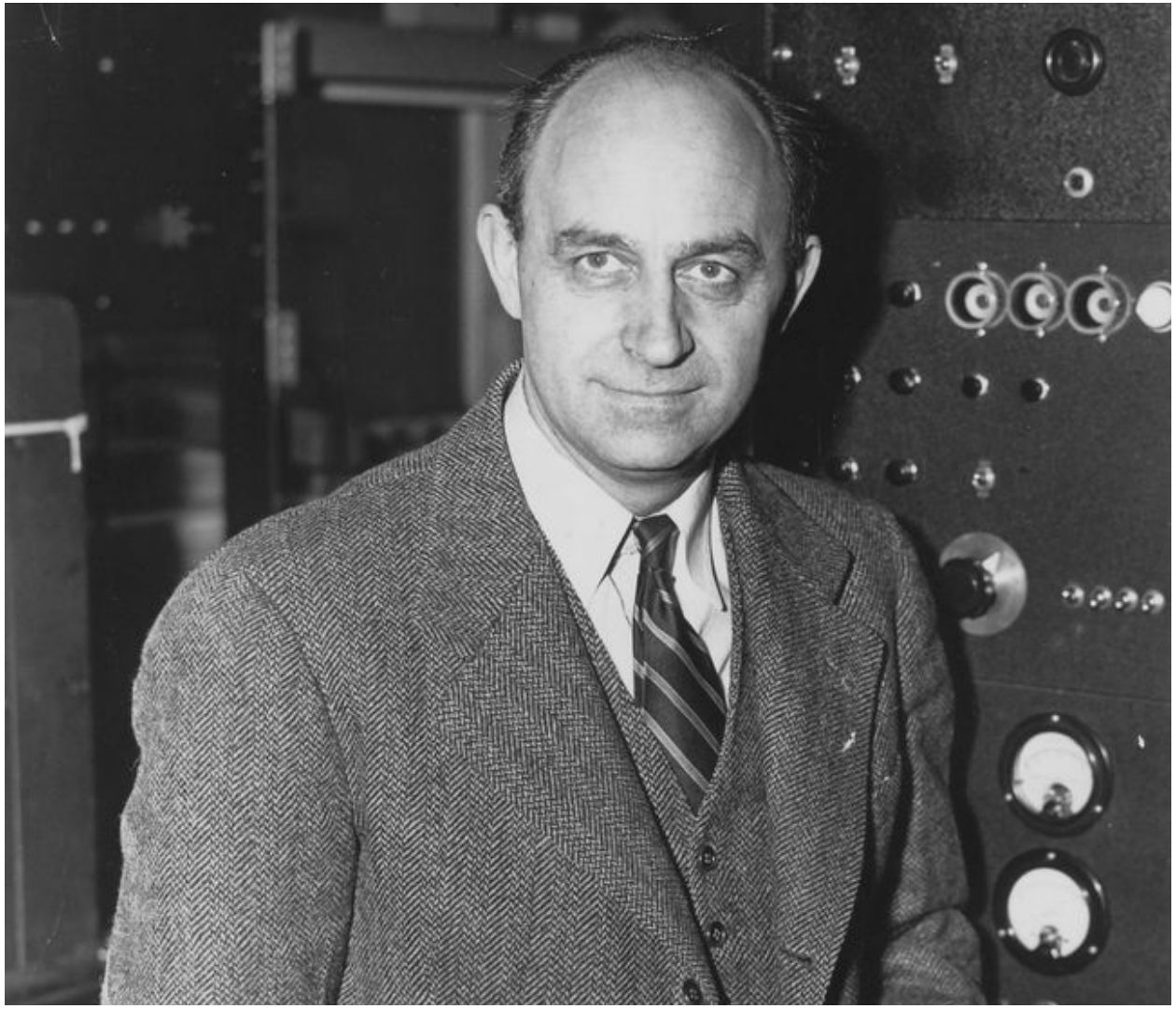
It's important for you to remember the key numbers that are relevant to your business.

- **Median and P99 response time: xx ms**
- **Throughput: x RPS**
- **Transactions: x RPS**
- **Customers: x**
- ***... and so on***

Eratosthenes measurement of Earth's circumference



Fermi Problems



“How many piano tuners are there in Chicago?”

Start by laying out the **base rates** and approximations

1. ~9M in Chicago's metro-area
2. ~2 people per household in the area
3. 1/20 households has a piano
4. Pianos are tuned ~once a year
5. Tuning a piano, including driving, is ~2 hours
6. Piano tuners work 8 hours a day, 50 weeks a year

“How many piano tuners are there in Chicago?”

Combine the base rates and approximations to arrive within an order of magnitude of the real answer.

- *(9,000,000 persons in Chicago) / (2 persons/household) × (1 piano/20 households) × (1 piano tuning per piano per year)* = 225,000 piano tunings per year in Chicago.
- *(50 weeks/year) × (5 days/week) × (8 hours/day) ÷ (2 hours to tune a piano)* = 1,000 piano tunings per year per piano tuner.
- *(225,000 piano tunings per year in Chicago) ÷ (1000 piano tunings per year per piano tuner)* = 225 piano tuners in Chicago.

You need less precision than you think!

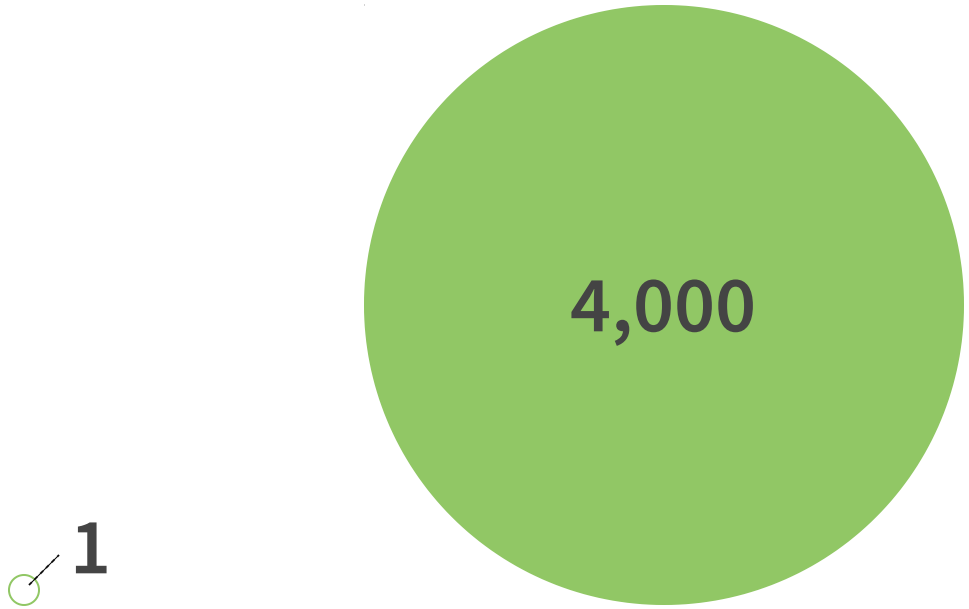
The objective of a napkin calculation is to provide a quick approximation within an order of magnitude of the real answer.

Fermi Decomposition of Snapshot-Restore Failover

Is it feasible to fail over a simple, 16 GiB in-memory database by dumping it to disk, sending it over the network, and restoring it in the target?

- Reading 1 GiB of sequential memory takes ~100ms
- Writing 1 GiB to SSD takes ~500ms
- Transferring 1 GiB from one Cloud Region (not zone) to another takes ~1 minute (150 Mbit/s)
- Reading 1 GiB from SSD takes ~250ms
- Writing 1 GiB of random memory in 64-bit increments takes 1.5 seconds
- => Not feasible to consider doing this for 16 GiB. *Need to explore other options.*

Big O Notation and Mechanical Sympathy



O(n) algorithm doing random SSD seeks.

O(n) algorithm reading from memory.

Using napkin math for systems design was popularized by Jeff Dean and Peter Norvig

Base Rates: Performance

https://github.com/sirupsen/base-rates, contribute your own

- Sequential Memory Reads <64 bit>: 1 ns @ 6 GiB/s (1MiB: 150 µs, 1 GiB: 150ms)
- Sequential Memory Writes <64 bit>: 5 ns @ 1.5 GiB/s (1MiB: 600 µs, 1 GiB: 600ms)
- Random Memory Read <64 bit>: 25 ns @ 300 MiB/s (1 MiB: 3.5ms, 1 GiB: 3.5s)
- Sequential SSD Read <8 KiB>: 1 µs @ 4 GiB/s (1 MiB: 200 µs, 1 GiB: 200 ms)
- Seqential SSD Write <16 KiB>, No Fsync: 15 µs @ 3.5 MiB/s (1 MiB: 250ms, 1 GiB: 5 min)
- TCP Echo Server, Localhost <64 bytes>: 15 µs
- Random SSD Read <64 bits>: 100 µs @ 0.5 MiB/s (1 MiB: 1.5 s, 1 GiB: 0.5 hour)
- Cloud Within-Zone Roundtrip: 250 µs
- Sequential SSD Write <16 KiB>, Fsync: 5 ms @ 10 KiB/s (1 MiB: 100s, 1GiB: 1 day)

Sign up for a monthly napkin math practice by email at sirupsen.com/napkin

Debugging an existing system

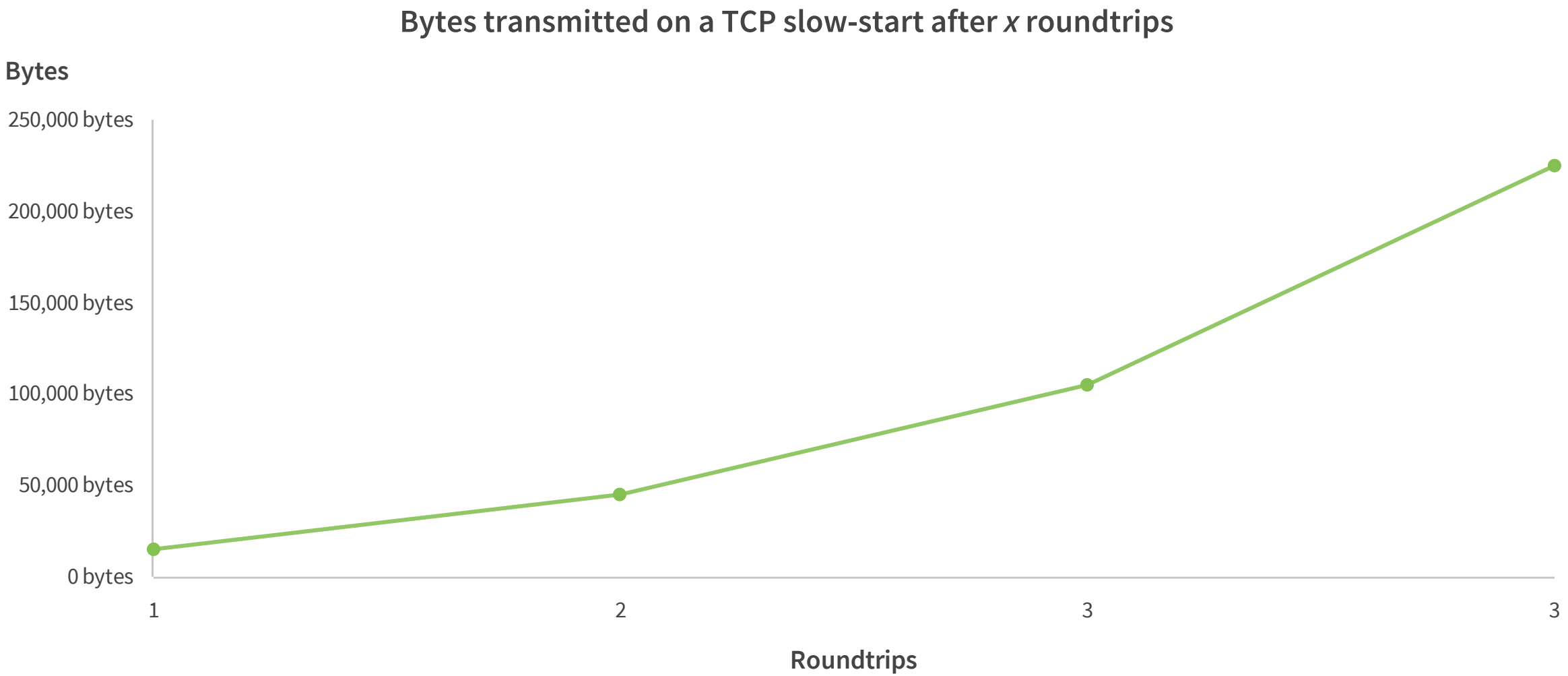
Why did it once take 2-3 seconds to serve a response for some Australian merchants?

1. **Render time: ~100ms**
2. **Round-trip time between Australia and D.C.: ~250ms**
3. **Request cycle round-trips: ~4.5 from DNS (1), TCP (1), SSL (2), HTTP (1)**
4. **=> Expected response time: *4 * 250ms* + *100ms* = 1.1 second**

   How could it possibly take 2-3 seconds on fast connections as reported?!

TCP Window-Scaling

Initial window is 10 * 1,500 bytes, and each roundtrip (if no packets are lost) will double the window size.



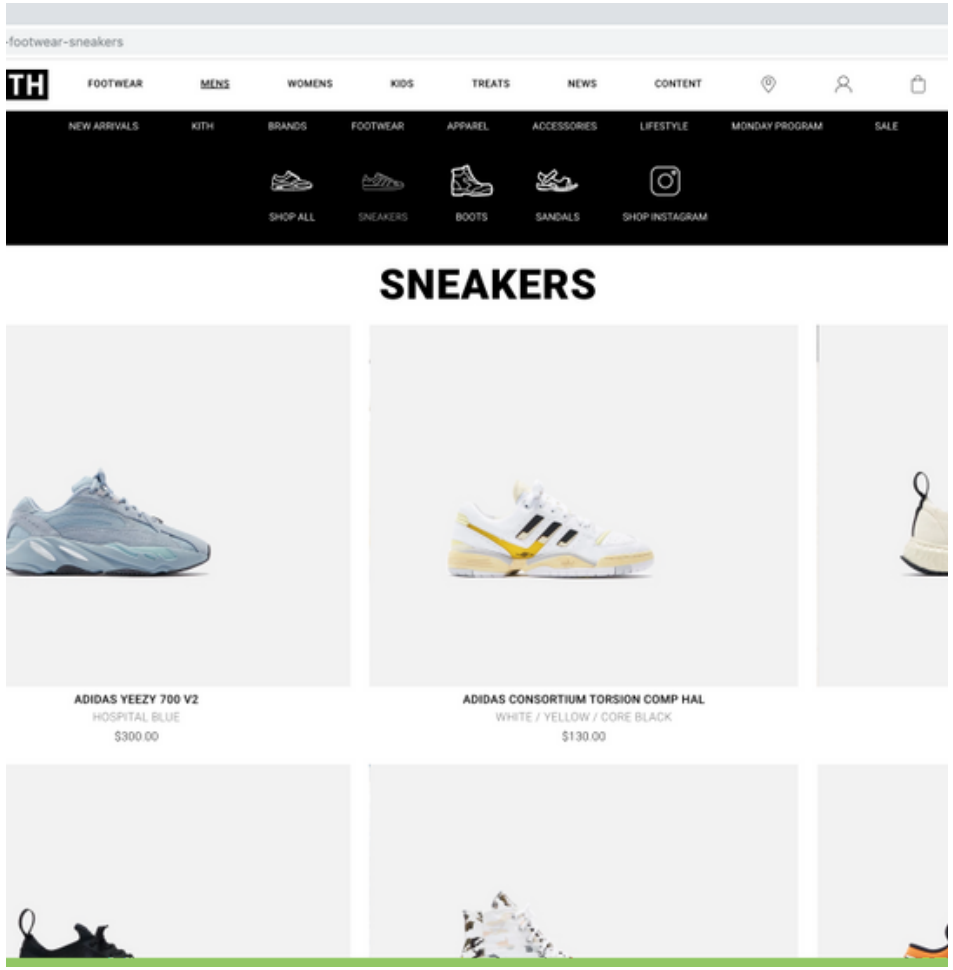
Base Rates: Networking

Rounded numbers from https://wondernetwork.com/pings to make them easier to remember.

- D.C. -> Frankfurt: 100ms
- D.C. -> {Singapore, Sydney}: 250ms
- D.C. -> Los Angeles: 60ms
- D.C. -> Tokyo: 150ms
- D.C. -> Kansas City: 40ms
- Singapore -> {Sydney, Tokyo}: 100ms
- D.C. -> Sao Paulo: 100ms
- Frankfurt -> Cape Town: 150ms

Example: Collection's Problem

Let's apply what we have learned to a real problem.



1. Merchant has P <= 2,09,152 (2^21) products
2. Collection filters products by R <= 64 rules
3. Rules: contains, not-contains, starts-with, ends-with, greater-than, less-than, equal-to, not-equal-to
4. Each product has K <= 64 fields rules can be applied to are on average 8 bytes (64 bits), and maximum 256 bytes.
5. => Serve queries in <= 10ms (don't have to be SQL, just here to illustrate): *SELECT product_id FROM products WHERE price <= 200 AND product_description LIKE "%sneaker%" AND product_rating >= 3 AND product_color IN ("BLACK", "YELLOW", "RED") ORDER BY product_color LIMIT 100 OFFSET 1000*

Example: Collection's Database

First-principle thinking is **required** to break out of iterative improvements and make order of magnitude improvements.

How do you memorize and practice base rates?

Spaced Repetition

Anki, Communis.io (Messenger-bot)

Sign up for Periodic Challenges

sirupsen.com/napkin

Develop your own base rates

Contribute to github.com/sirupsen/base-rates or start your own!

Apply them to your own problems

Reconciling the difference between your first-principle, napkin model and reality is going to either present an opportunity to improve the system or fix your mental model.

Thank You

Simon Eskildsen, sirupsen.com/napkin

@sirupsen

Resources

- Measure Anything (Book)
- What Every Programmer Should Know About Memory
- Mechanical Sympathy Blog
- Wonder Network for Ping Times
- Latency Numbers Every Programmer Should Know
- Computers are Fast Quiz
- Jeff Dean Presentation
- github.com/shopify/base-rates
- Guesstimate Spreadsheets
- Monthly Challenges