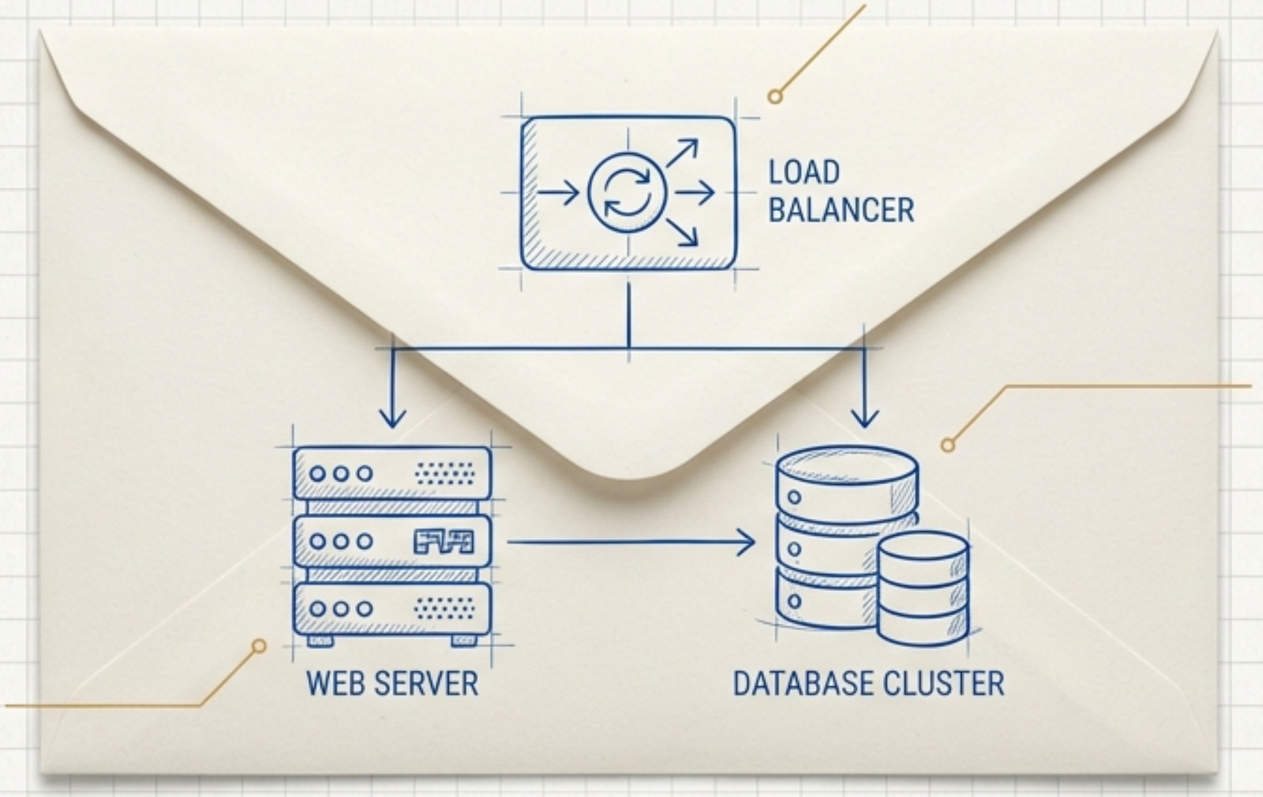
The Engineer's Field Guide to System Estimation



Mastering Back-of-the-Envelope Calculations for Architecture and Interviews

It always starts with a simple question.

The DevOps team comes to you before a deployment. They ask:

“How many resources do you need? How many servers? What CPU and memory configuration? How much storage?”

Guess too high?

You’re wasting company money on idle resources.

Guess too low?

Your system crashes under load.

Your Answer Lives on the Back of an Envelope

Core Concept

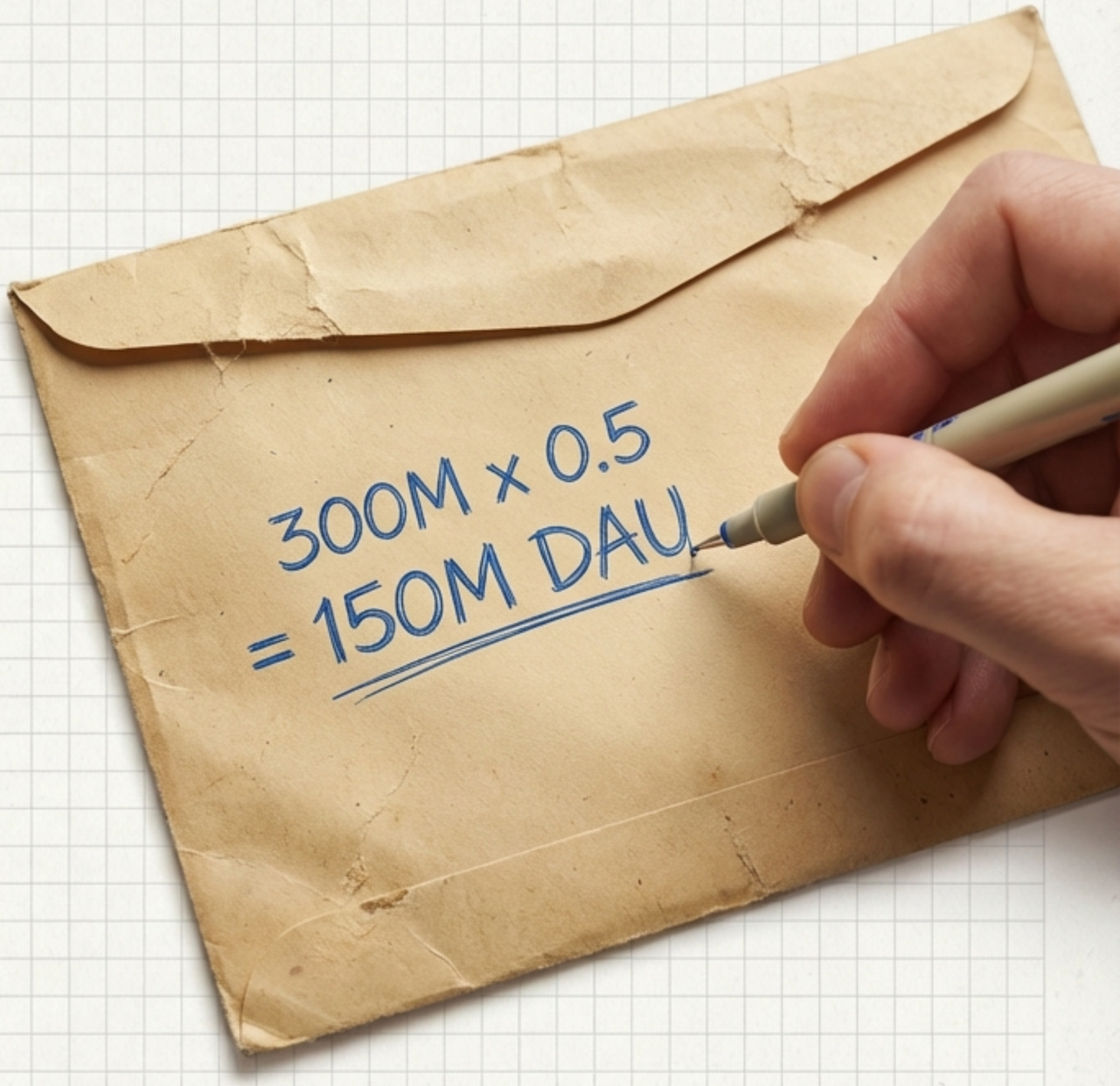
Back-of-the-Envelope Calculation is a technique for making rough, approximate calculations using simple arithmetic and sound assumptions.

Why the name?

The name comes from the practice of doing informal, unstructured calculations on any unstructured calculations on any available surface, like the back of an envelope. It's about being scrappy and fast, not about rigid precision.

Main Idea

This is how senior engineers and architects make informed decisions quickly, validating ideas long before a single line of code is deployed.



This Isn’t Guesswork; It’s a Strategic Skill

Quick **estimation** is a cornerstone of effective system design. It allows you to:

Validate Solutions: Quickly confirm if a proposed architecture is even feasible for the expected load.

Identify Bottlenecks: Find potential performance issues in your design *before* they become real problems.

Communicate Effectively: Justify your design choices with data, showing the ‘why’ behind your architecture.

Communicate Effectively: Justify your design choices with data, showing the ‘why’ behind your architecture.

Demonstrate Seniority: Showcase a thought process that balances trade-offs, constraints, and scale.

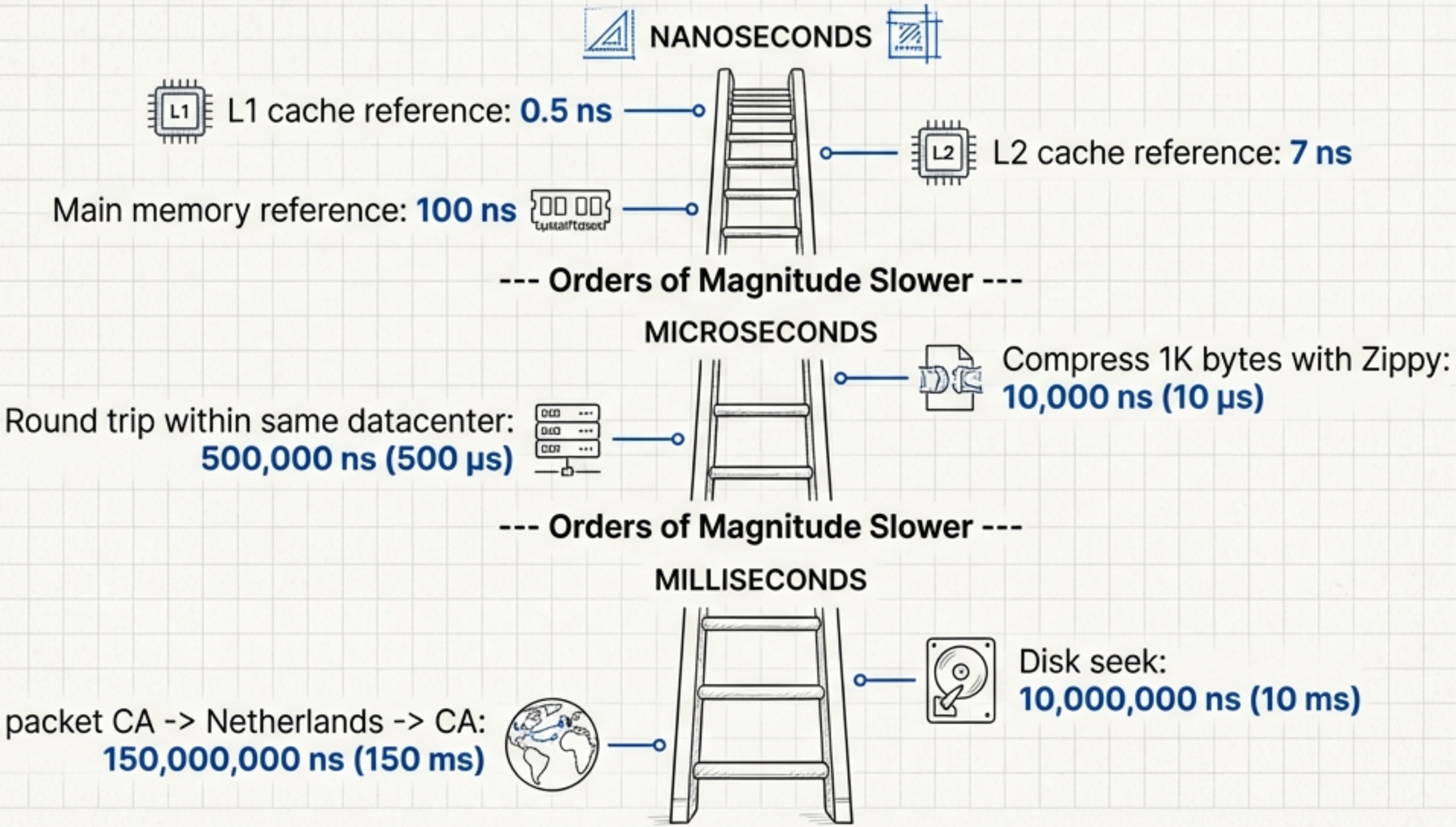
The Field Guide: Know Your Scale (Data)

All storage and bandwidth estimations start here. The key is to think in powers of 10. Each level is roughly 1,000 times the previous one.

POWER	APPROXIMATE VALUE	FULL NAME	SHORT NAME	VISUAL
2^10	1 Thousand	1 Kilobyte	1 KB	
2^20	1 Million	1 Megabyte	1 MB	
2^30	1 Billion	1 Gigabyte	1 GB	
2^40	1 Trillion	1 Terabyte	1 TB	
2^50	1 Quadrillion	1 Petabyte	1 PB	

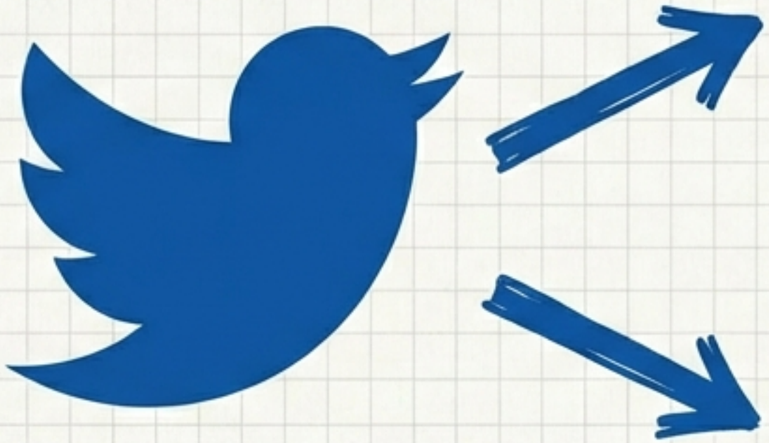
Remember this one rule: **Memory is fast. Disk is slow.**
This trade-off governs countless architectural decisions.

The Field Guide: Know Your Speed (Latency)



The Workshop: Estimating Twitter's Core Services

We've been tasked with designing a system like Twitter. Before we architect anything, let's perform a back-of-the-envelope calculation.



1. QPS (Queries Per Second)

2. Storage (5-Year Requirement)

Step 1: Write Down Your Assumptions

Every calculation is built on a set of assumptions. Stating them clearly is the most important step.

Our Assumptions for Twitter

- **Users:** 300 Million Monthly Active Users (MAU)
- **Activity:** 50% of users are active daily (DAU)
- **Content Rate:** Users post an average of 2 tweets per day
- **Media:** 10% of tweets contain media (photos/videos)
- **Media Size:** Average media upload is 1 MB
- **Retention:** Data must be stored for 5 years

Step 2: Calculate Write QPS (Queries Per Second)

1. Calculate Daily Active Users (DAU)

300M MAU * 50% = **150M DAU**

2. Calculate Total Tweets per Day

150M DAU * 2 tweets/day = **300M tweets/day**

3. Calculate Tweets per Second (QPS)

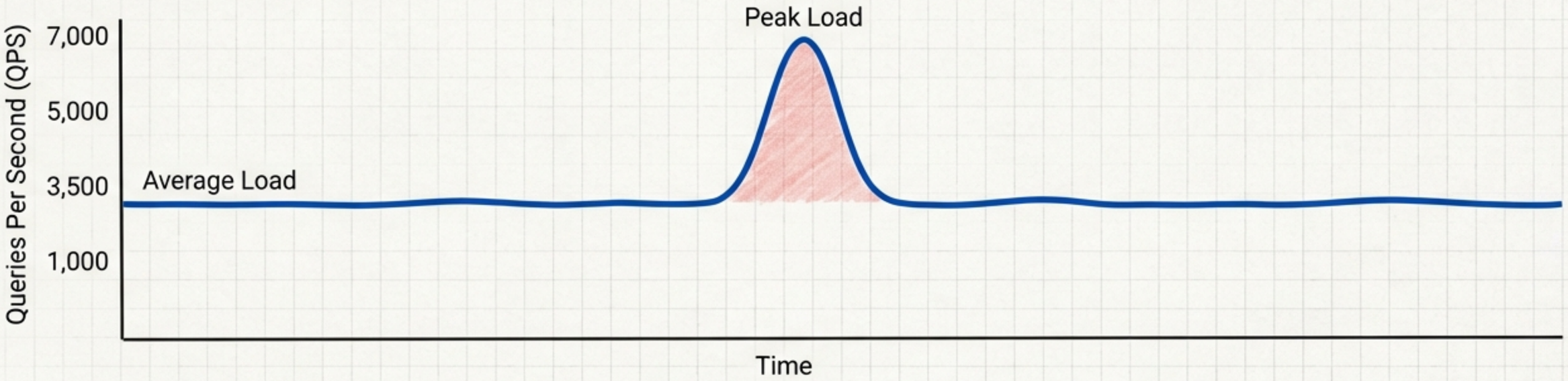
300,000,000 tweets / 86,400 seconds ≈ **3,500 tweets/second**

Note: Rounding 86,400 to 100,000 gives ~3,000, which is in the same order of magnitude. Both are valid for this exercise.

Our database must handle approximately **3,500 insert operations every second**, on average. average.

Step 3: Plan for the Peak

Average QPS is useful, but systems must survive peak traffic. A global event, a viral moment—traffic is never evenly distributed.



Rule of Thumb: A common starting point is to plan for a peak load of **2x the average**.

Peak QPS Calculation: 3,500 QPS (Average) * 2 = **7,000 QPS (Peak)**

Your system must be provisioned to handle a surge of **7,000 writes per second** without failing. This informs decisions about load balancing, auto-scaling, and database capacity.

Step 4: Estimate 5-Year Storage Needs

We'll focus on **media storage**, as it will dwarf text storage.

1. Calculate Media Tweets per Day:

300M tweets/day * 10% contain media = **30M media tweets/day**

2. Calculate Storage per Day:

30M media tweets/day * 1 MB/media = **30 Terabytes per day**

3. Calculate 5-Year Storage:

30 TB/day * 365 days/year * 5 years = **54,750 TB**

~55 PETABYTES

Over the next 5 years, we will need to procure and manage approximately **55 Petabytes** of object storage for media alone.

Guiding Principles for Estimation

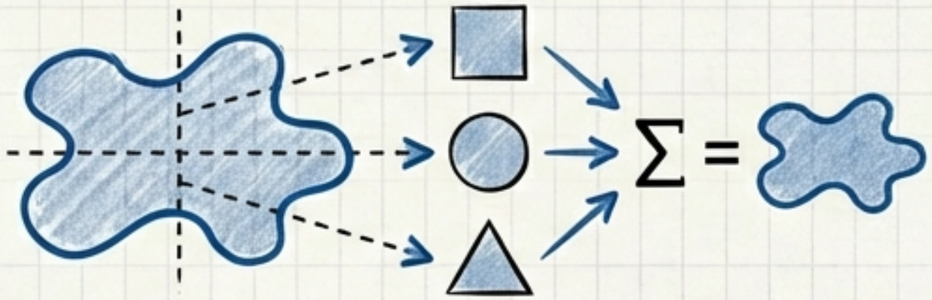
1 Rounding and Approximation Are Your Friends.

Don't get bogged down in precision. The goal is the right order of magnitude.

~~86,400 seconds~~ = **100,000 seconds**

2 Break Down Complex Problems.

Estimate requirements for smaller components individually and then aggregate them.

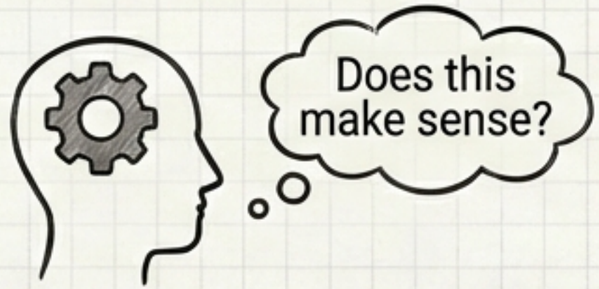


3 Always Label Your Units.

Is it 100 MB or 100 GB? Per second or per day? Ambiguity leads to massive errors.

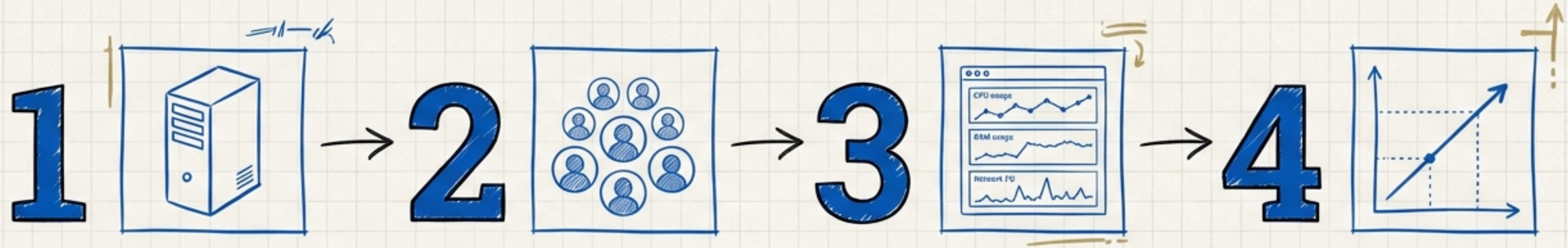
4 Perform a Sanity Check.

Does the final number feel plausible? Compare your estimate with what you know about similar existing services.



From Theory to Practice: A Real-World Workflow

When you have no existing data, you can’t be perfect. The goal is to get a reasonable starting point and iterate.



Start with a Bare Minimum

Provision a small instance for a dev environment (e.g., 2 CPU, 4GB RAM).

Deploy and Test

Get 8-10 internal users on the system to generate a small, known load.

Monitor Everything

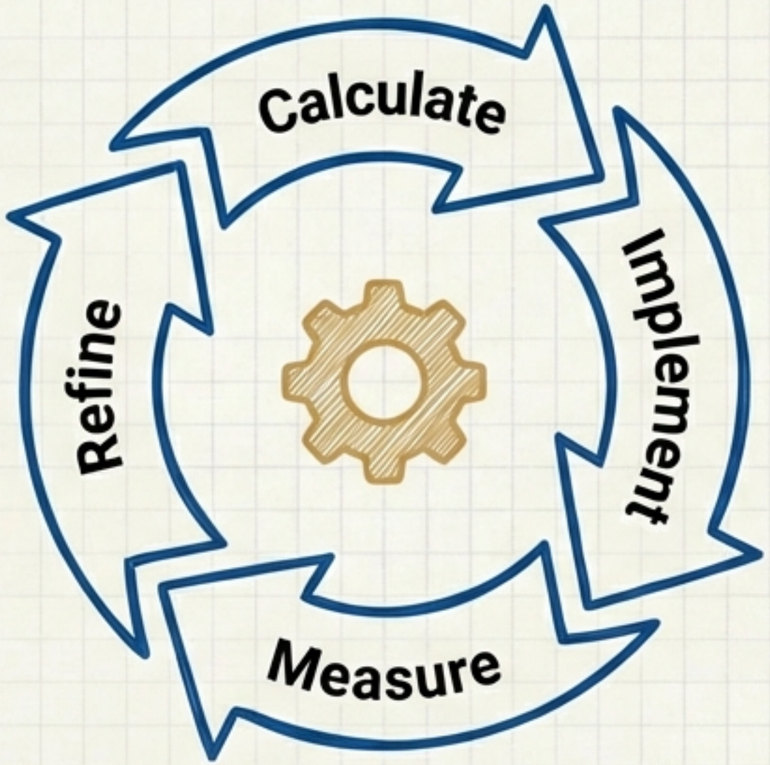
Watch CPU utilization, memory usage, and I/O under this known load.

Extrapolate for Production

Use the baseline data to make an educated guess for your target user load.

Estimation is a Craft, Honed by Experience

You won’t get it right the first time. No one does. The calculations provide the hypothesis; the real world provides the data.



This skill, famously advocated by engineers like Google’s Jeff Dean, is a journey of refinement. The goal isn’t to be perfect, but to get better with every system you build.