Aalto University
School of Electrical
Engineering

Microservices: Architecture

31.1.2018

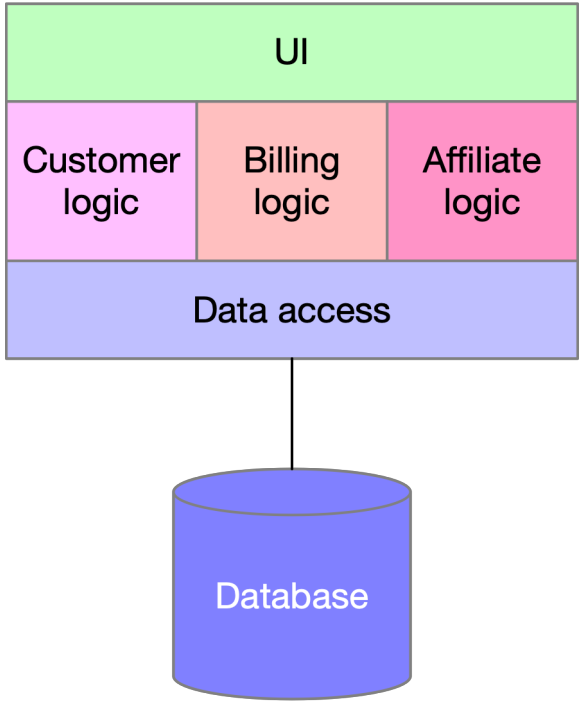
Santeri Paavolainen

How we define microservices?

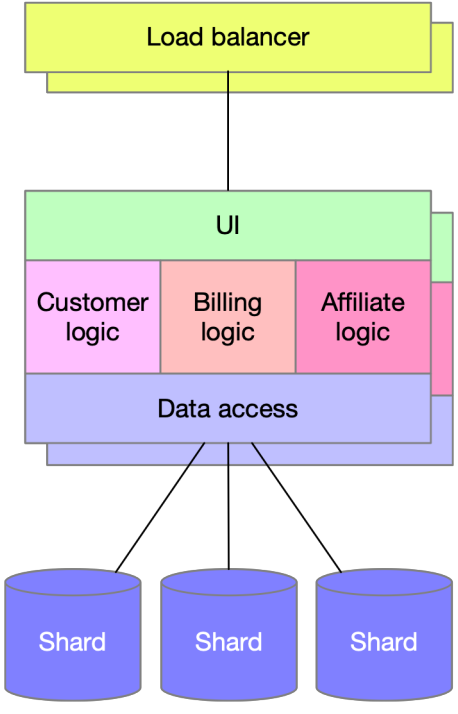
- **From first lecture:**
  - Technical e.g. architectural viewpoint (boxes and arrows)
  - Operational viewpoint (patterns to solve real-world problems)
  - Organizational viewpoint (how to organize work)

- **Let's focus on the architectural viewpoint**

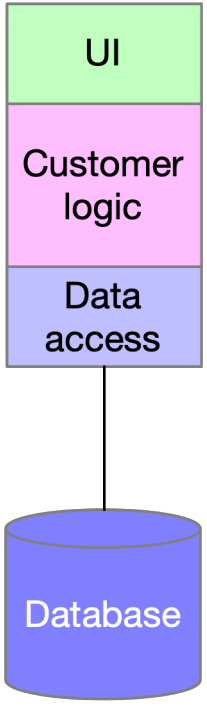
Monolith: Idealized case



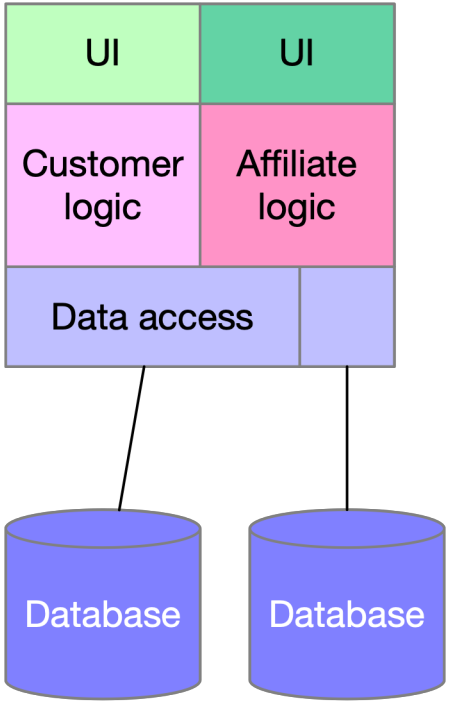
Monolith: Idealized (scaled) case



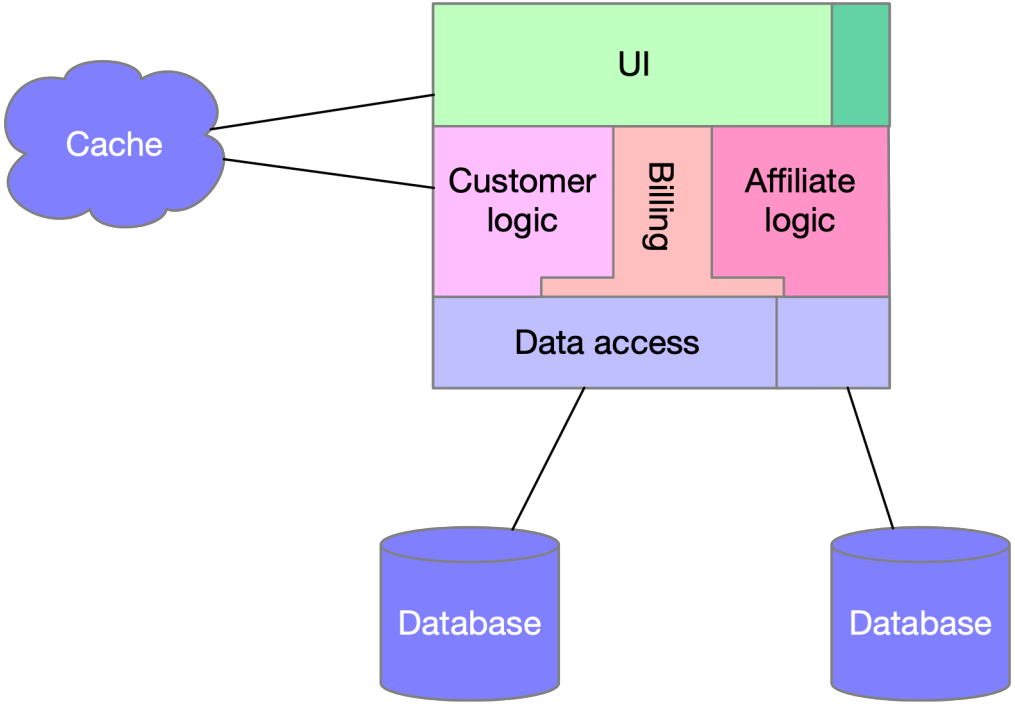
Monolith aggregation



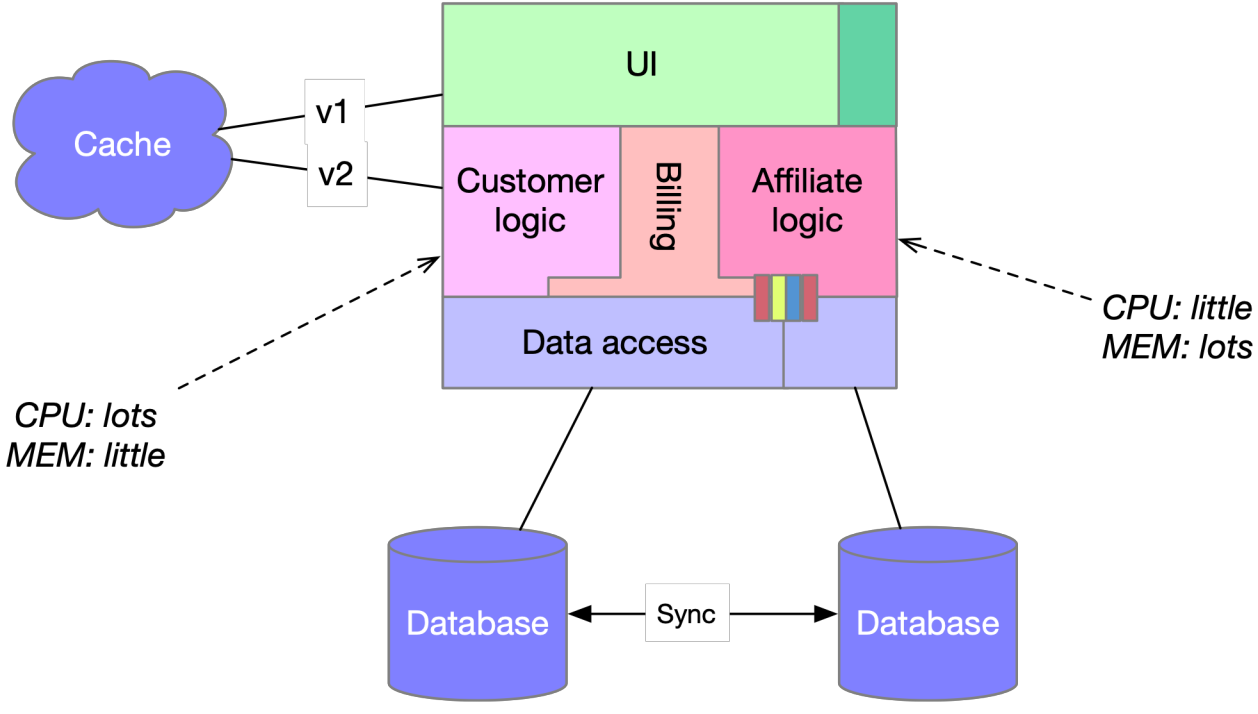
Monolith aggregation



Monolith aggregation



Monolith aggregation



Monolith: Reality

- **Accumulated growth**
- **Ad hoc solutions**
- **Cross-module dependencies**
  - Dave Device-Teamer: “I need to verify device token … Ulf User-Interfacer has a user token verifier in UserInterface, I’ll just instantiate with dummy parameters and use that” (anyone spot a problem?)
- **Cross-component depencies**
  - Dave uses Redis set up by Ulf, Ulf needs to upgrade to a newer version, but this breaks Dave’s application
- **Different CPU and memory requirements**
  - Horizontal scaling scales all at the least common denominator pace, wastes resources

Service Orientation: The Idea

- **Define services by a logical boundaries**
  - Each service responsible for anything “inside”
  - Interaction via well-defined interfaces (APIs or other)

  - “Separation of concerns”

- **How to define a service boundary?**

Service boundaries

- **Each service focuses on**
  - … one service? (what is a service?)
  - … a functionality? (a metric service needs to ingest, store, collate, condense, search and do lots of other things)
  - … responsibility?
- **Often a "service boundary" is defined by organization!**
  - "What can we manage with 8 people?"
  - Blindly taken implies start-up teams must build monoliths …
  - Sort of putting the cart before the horse

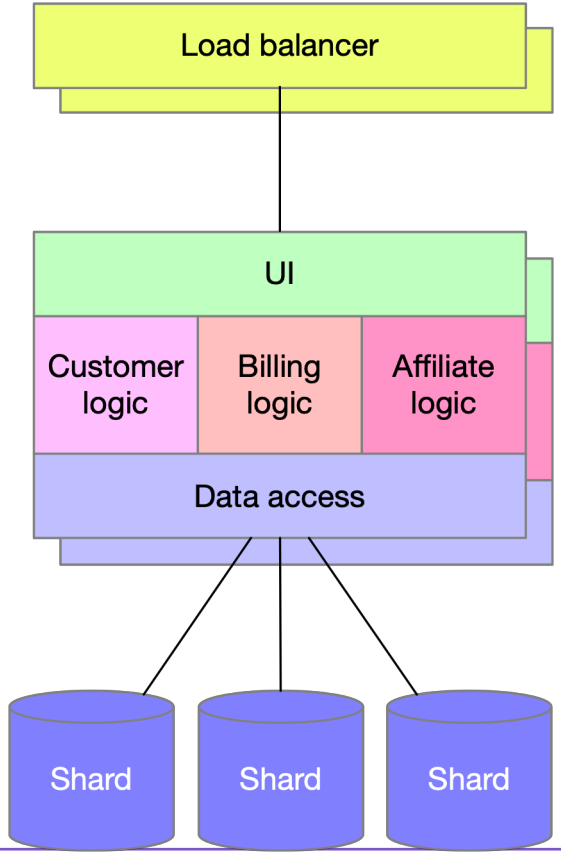
Service boundaries

- **Some rules of thumb**
  - Segment services by boundaries which make sense, where
  - Service focuses on handling all responsibilities for some functional need, and
  - “A two-pizza team” can handle design, development, testing, deployment and operations of the service

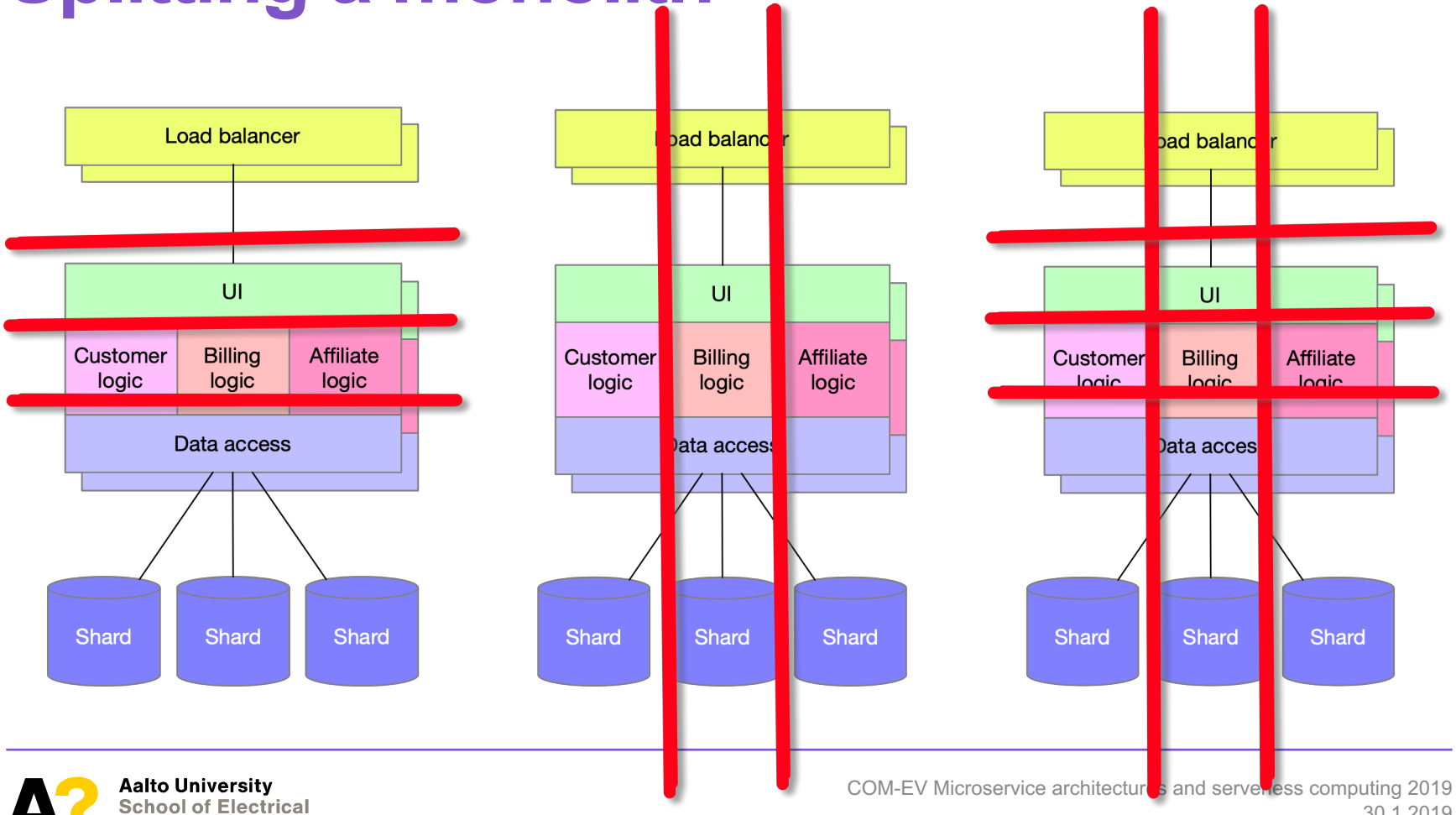
“Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.”

- Antoine de Saint-Exupery

Splitting a monolith



Splitting a monolith

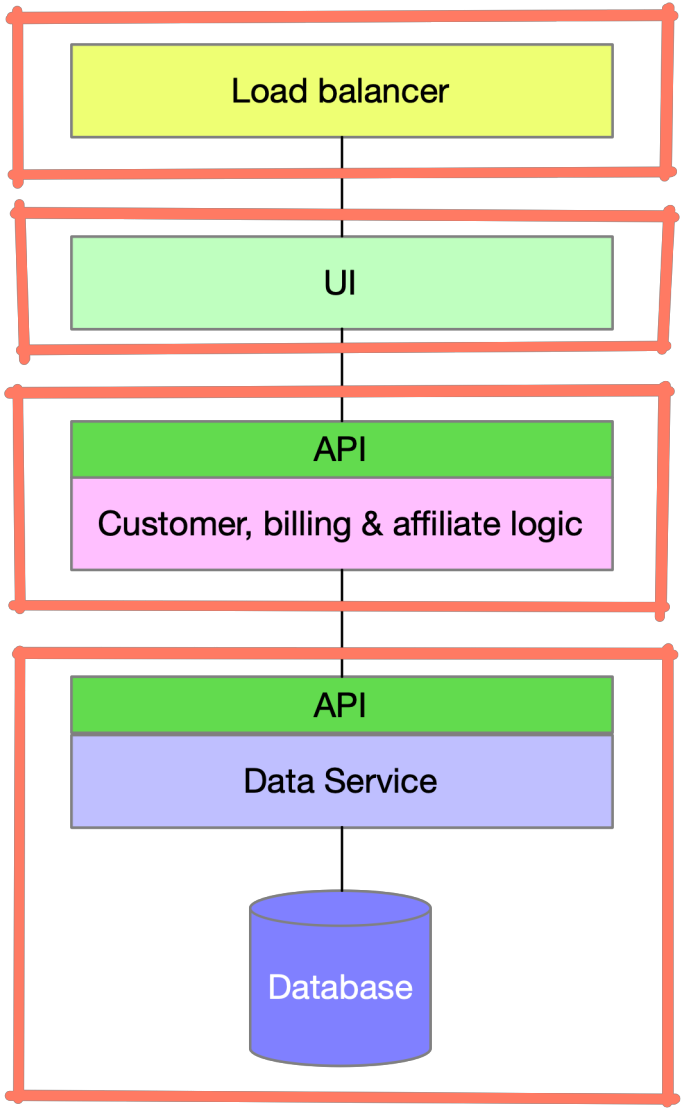


“Request distribution service”

“User interaction service”

“Business logic service”

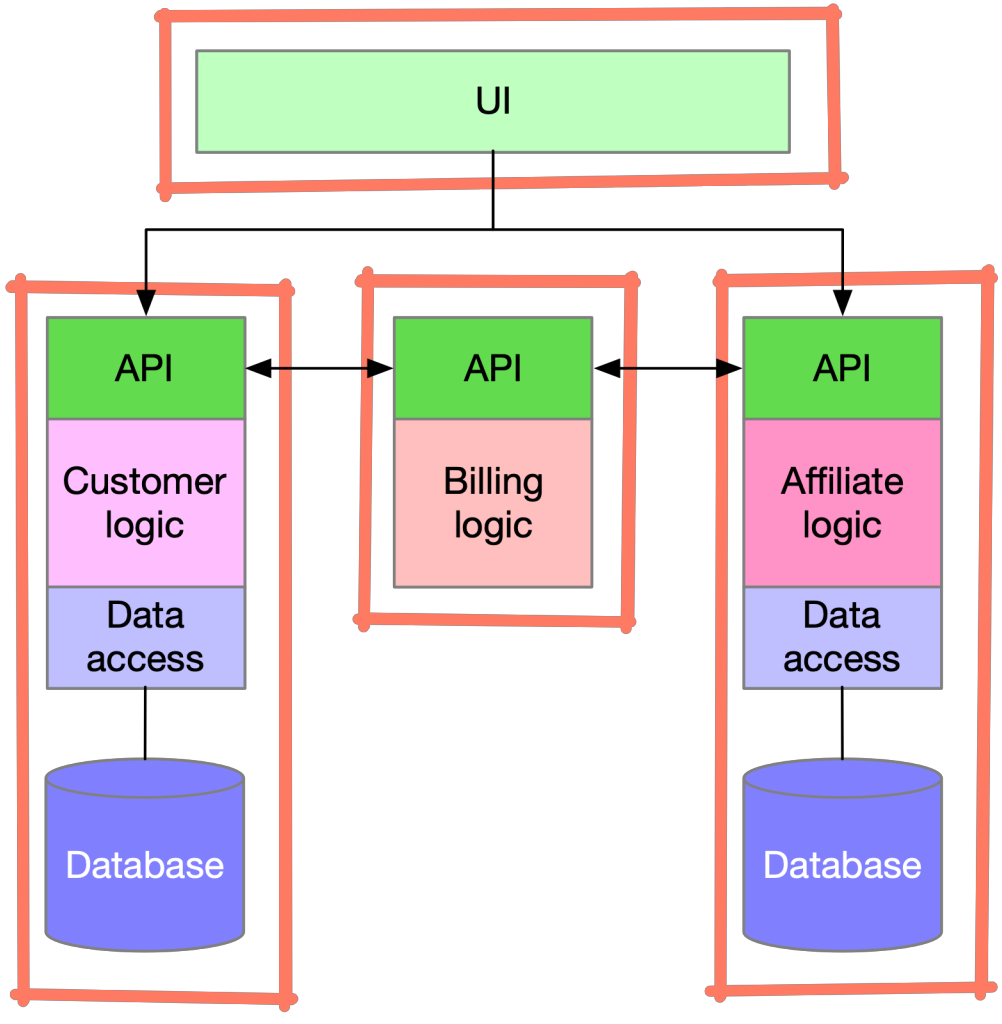
“Persistence service”



Is this a microservice architecture?

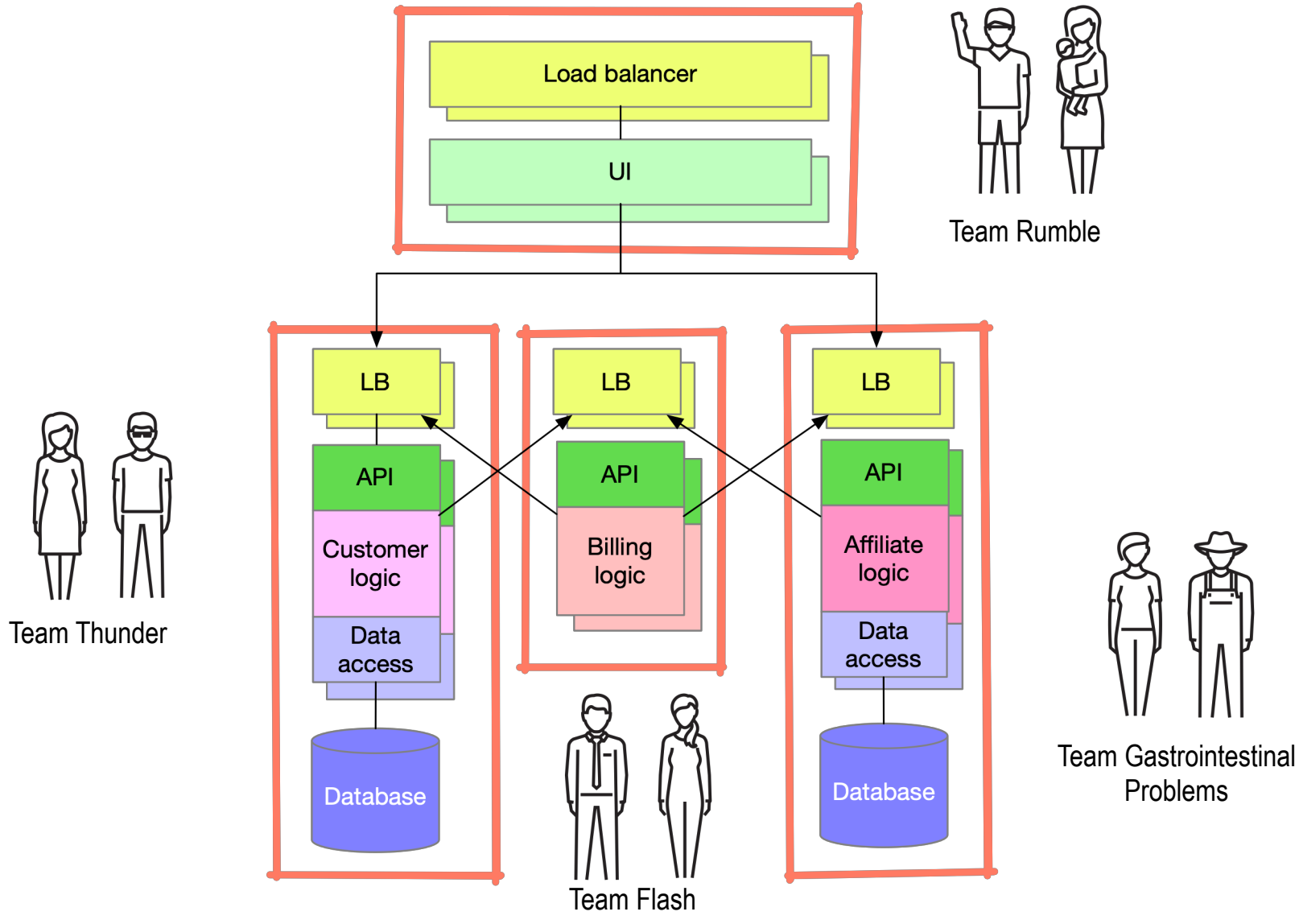
NO!

Why not?



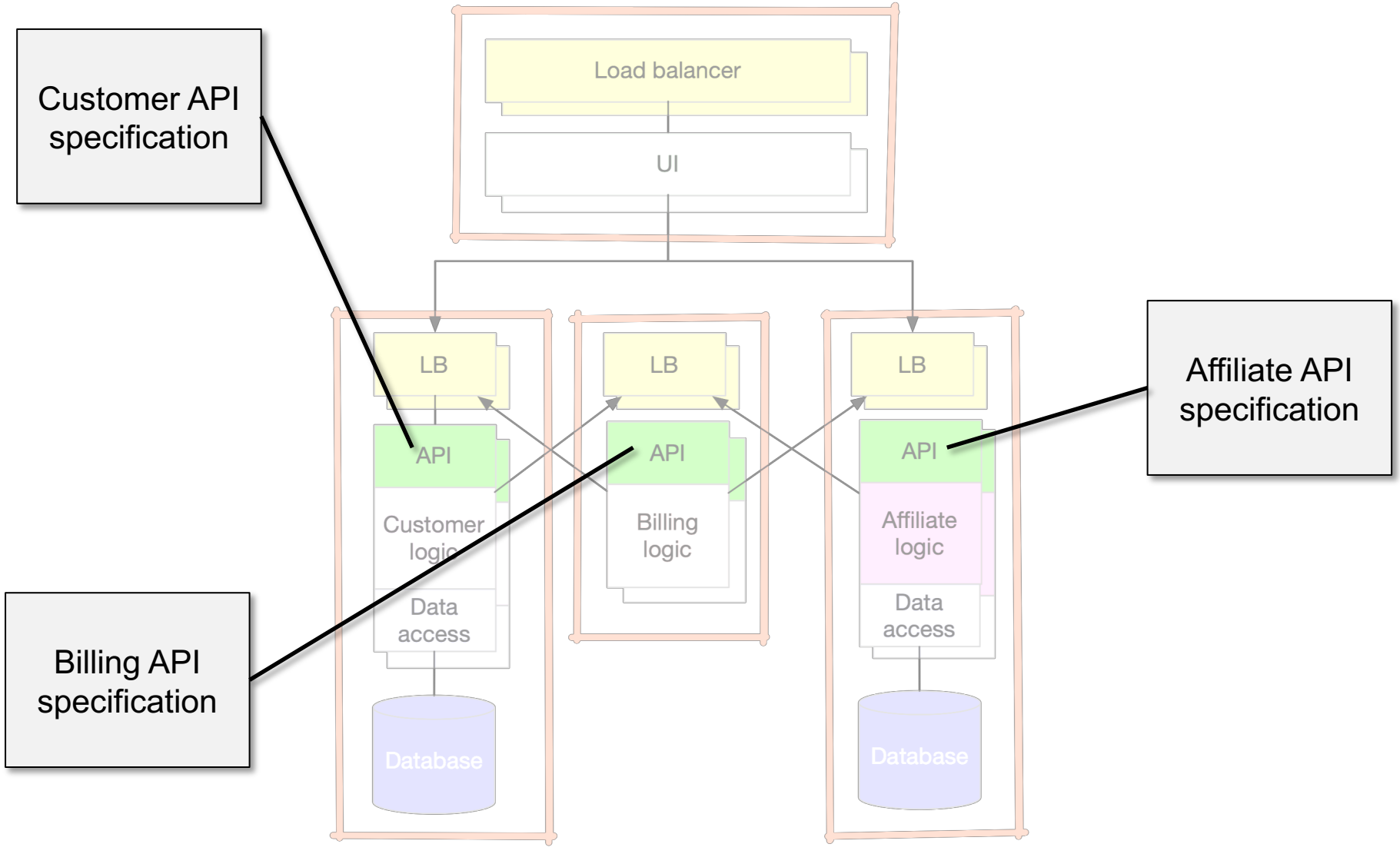
Why not a single database?

Where did load balancer go?


Load balancer
UI
Team Rumble
LB
LB
LB
API
API
API
Customer logic
Billing logic
Affiliate logic
Data access
Data access
Database
Database
Team Thunder
Team Flash
Team Gastrointestinal Problems

Service description: Interface

- **How does a microservice interact with another?**
  - Service interface description
  - May be anything that allows control and data transfer
    - *REST is a practical default*
  - Formal interface definitions: Interface Definition Languages
    - *(WSDL and SOAP)*
    - *OpenAPI and Swagger for REST*
    - *gRPC and Thrift inherently IDL protocols*


Customer API specification
Billing API specification
Affiliate API specification
Load balancer
UI
LB
LB
LB
API
API
API
Customer logic
Billing logic
Affiliate logic
Data access
Data access
Database
Database

Service description: SLAs

- **Interfaces do not tell about non-functional requirements**
  - Availability
  - Reliability
  - Response time (distribution)
  - Security guarantees
  - Interface stability (obsolescence)

- **These are often highly situational**

The Big Kahuna

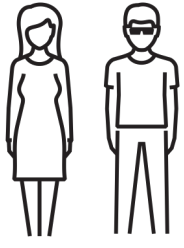
1M users
100k daily
99.9%
availability

Hi Thunder!
We need
99.95% and
800 r/s peak

Team Rumble

Hi old farts!
We need
99.5% and
30 req/s
sustained

Yep Rumble,
can do but it'll
cost you 200
donuts/day

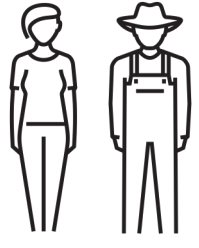


Team Thunder

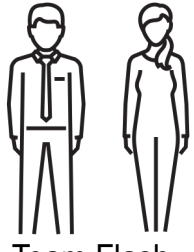
grumble grumble
Only if you can submit
requests using JCL

Either 99% at 50ms
or 99.99% at 5s?

Pssst Flash, what
can you do for 500
donuts/year?



Team Gastrointestinal
Problems



Team Flash

Service description: SLAs

- **Availability, response time (95%, mean etc.) most common**
- **Rarer: adaptability (variable load), persistence, maximum continuous unavailability, …**
- **SLA implies monitoring**
  - Either producer or consumer of the service (often both, former explicit and latter implicit)
- **Most often for humans only (not machine-readable)**
- **Internal vs. external SLAs**
  - Internal SLAs weighed against business risks (losses)
  - External SLAs full of weasel-words

SLA weaseling: AWS EC2

"AWS will use commercially reasonable efforts to make the Included Products and Services each available with a Monthly Uptime Percentage (defined below) of at least **99.99%** …"

""Monthly Uptime Percentage" is calculated by subtracting from 100% the percentage of minutes during the month in which any of the Included Products and Services, as applicable, was in the state of **"Region Unavailable."**""

""Region Unavailable" and "Region Unavailability" mean … when **more than one AZ** within the same Region, in which you are running **an instance** …, as applicable, are concurrently "Unavailable" to you"

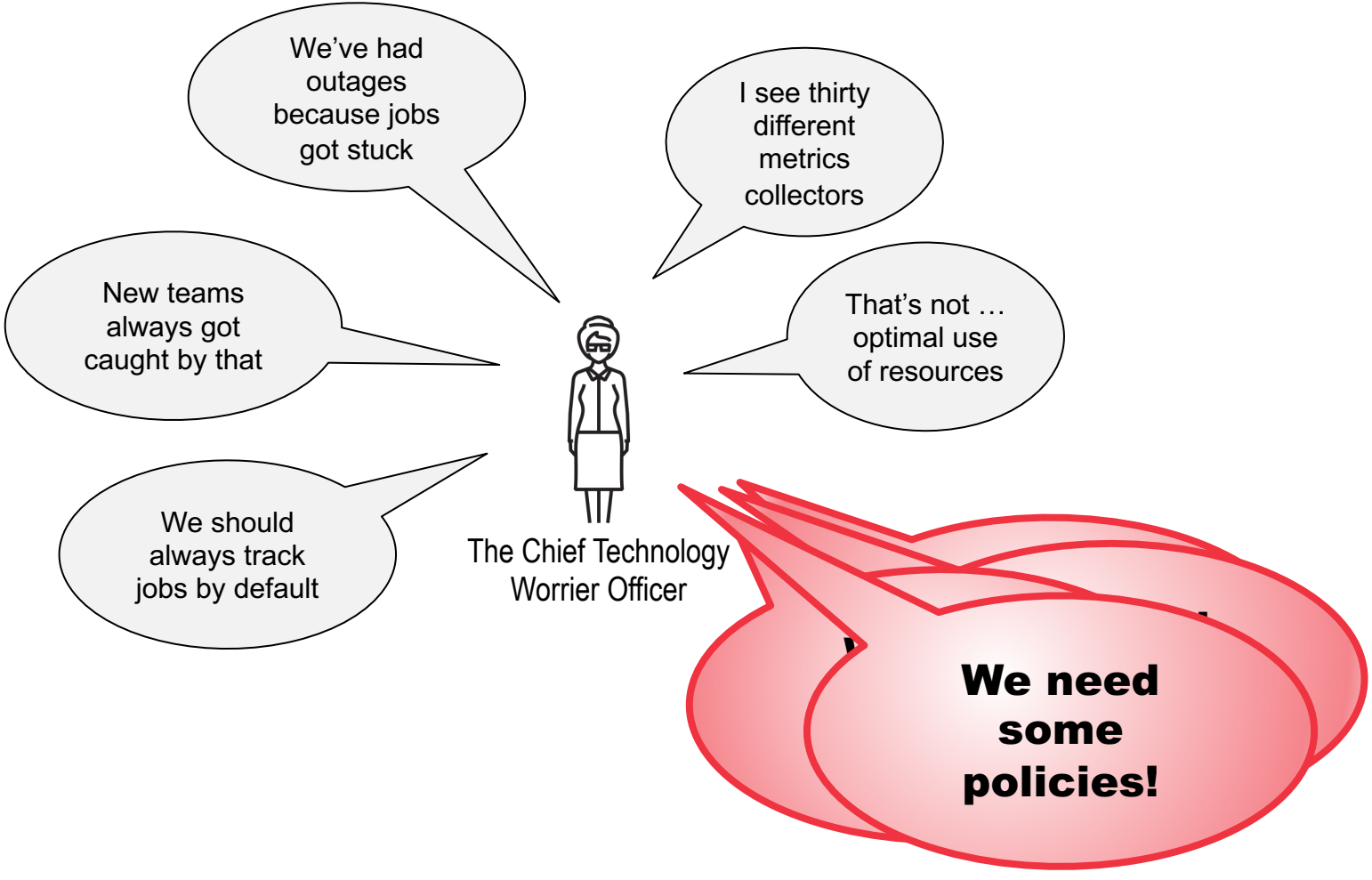
""Unavailable" and "Unavailability" mean: … when **all of your running instances** … have no **external connectivity**."

[More on AWS service availability]

Service description: API and SLA

- **API can be formally defined in machine-readable format**
  - Generally a good idea (but human-only versions common)
  - Either via external description language, or as part of the protocol specification itself (IDL)
  - Offers a chance of API conformance monitoring
- **SLAs are formally defined in human-readable format**
  - Clear distinction between internal and external SLAs
  - Internal SLAs generally more strict and relevant

- **Do these completely define a service?**


We've had outages because jobs got stuck
I see thirty different metrics collectors
New teams always got caught by that
That's not … optimal use of resources
We should always track jobs by default
The Chief Technology Worrier Officer
We need some policies!

Organizational requirements

- **Are these part of a service definition or not?**
  - Tricky …
  - These requirements can be vague and/or aspirational
  - Requirements for an implementation or for a service?

- **Is a contra-example of "microservices decide things themselves"**
  - Real-world constraints — resources, time, risk management, …

- **See Good Eggs example**

API spec + SLA + org requirements = we done?..?!?!?!

I have 1M users getting 99.99% available pictures of cats?!?!?!

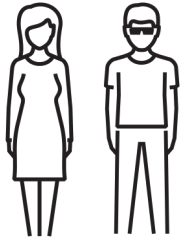
The Big Kahuna

Thunder ahoy, did donuts buy us only cat pictures?

Team Rumble

You scoundrels! Where are job results?

Yep, we've got best cat pics and all to API spec and SLA!

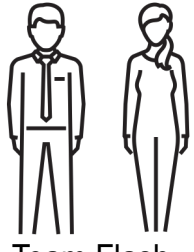


Team Thunder

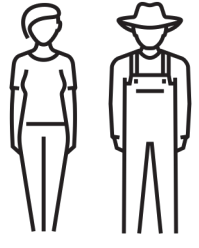
You didn't know results are mailed as printouts monthly? Only requests in SLA!

What? Our API is just returning random numbers. SLA's good, tho.

Pssst Flash, why is your parrot ML model so bad?



Team Flash



Team Gastrointestinal Problems

Forgot the actual functionality.

What does it actually DO?

Service description: Functionality

- **Often implicit in API specification, introductory text etc.**
- **Easily overlooked**
  - In monoliths often organizational knowledge

    "Oh the Frobnitzor! You know, aside from Frobnitzing it also does Xuulization and Memengosplanning."
- **Being able to actually ~~describe~~ sell the service is important in an organization**
  - Helps people use it — they're not experts on it!
  - Decreases replication of effort (NIH syndrome)

API + SLA + org req + *tell others what it actually does* = done?

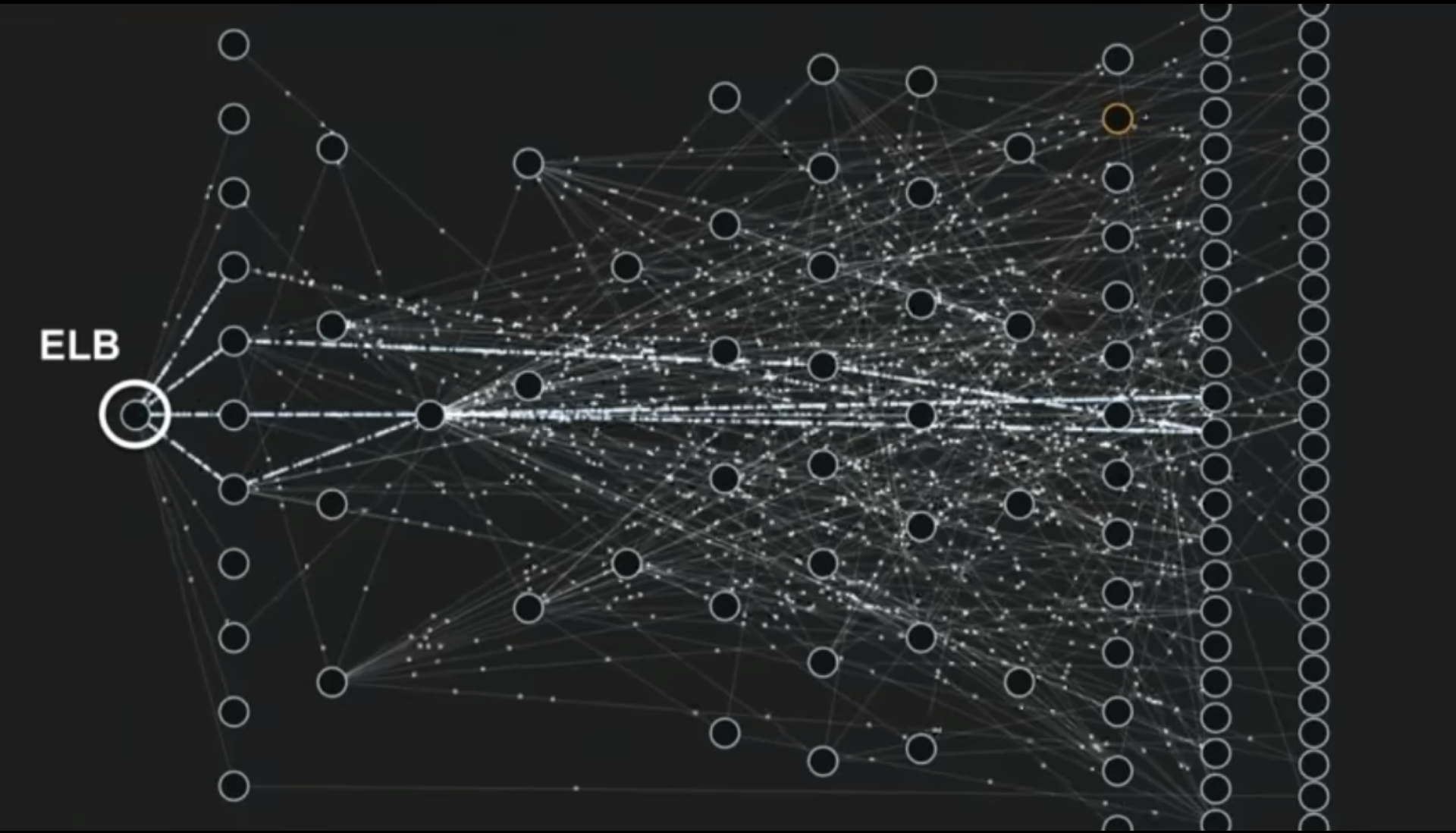
What does a software architect do?

- **In a larger company**
  - Draw fancy pictures ← **Communication**
  - Talk to lots of stakeholders ← **Communication**
  - Attend lots of meetings ← **Communication**
  - Talk to other engineers ← **Communication**
  - Draw lots of stuff on whiteboard ← **Communication**
- **In a start-up**
  - Less talking and less meetings ← **Communicate with actual customers**
  - Less people to catch your mistakes

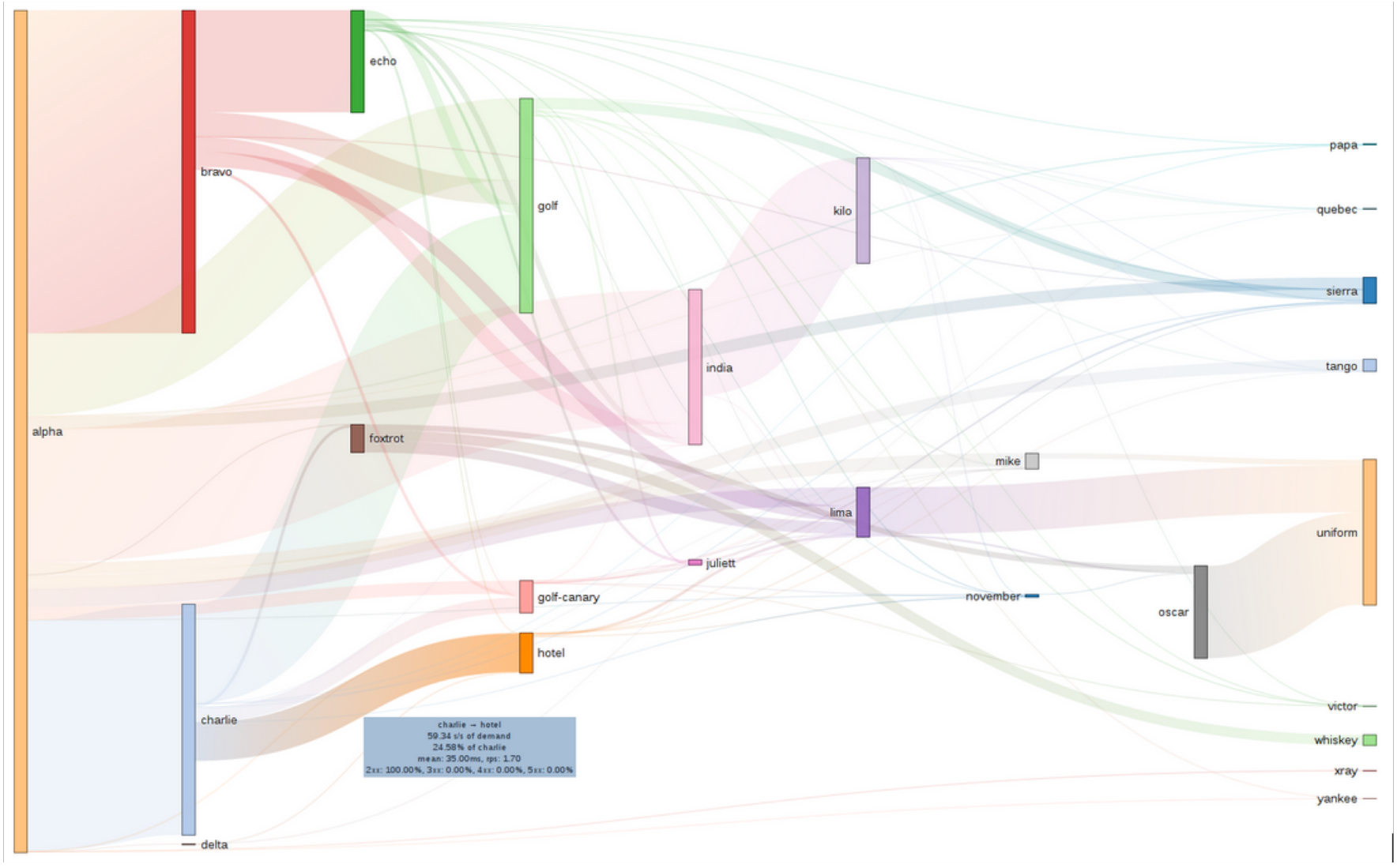
Technical difficulty for SWA is attaining knowledge, applying the knowledge to technical problems is usually straightforward.

Microservices are not systems

- **Systems comprise of multiple services**
  - This was true even decades ago, nothing new about microservices
- **Often multi-faceted**
  - Serving different types of users, different workloads
  - Overall goal is to support an organization's goals (business, academic, ..)



Source: InfoQ @ Youtube



Source: Netflix Technology Blog