CS-310 Scalable Software Architectures

Lecture 6: Microservices

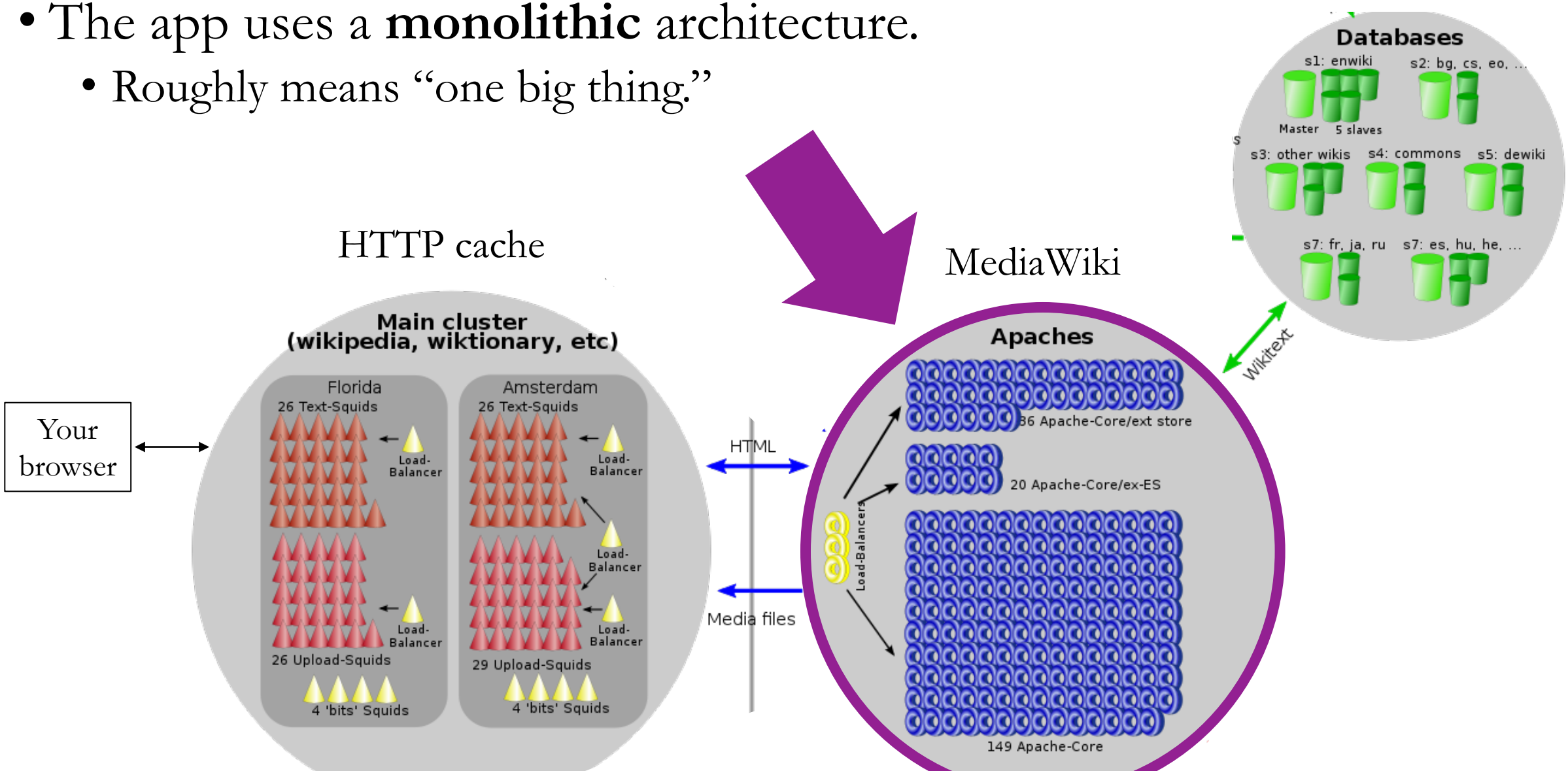
Steve Tarzia

Last time: REST APIs and Data Serialization

- Services are **black boxes**, exposing **network APIs**.
  - Decouples development of different parts of the system.
  - Network APIs define the format and meaning of requests and responses.
- **REST** is the most popular format for network APIs
  - Based on **HTTP** and uses *url*, *method*, *response codes*, usually *JSON bodies*.
- **JSON** is a common data **serialization** format. **XML** is also used.

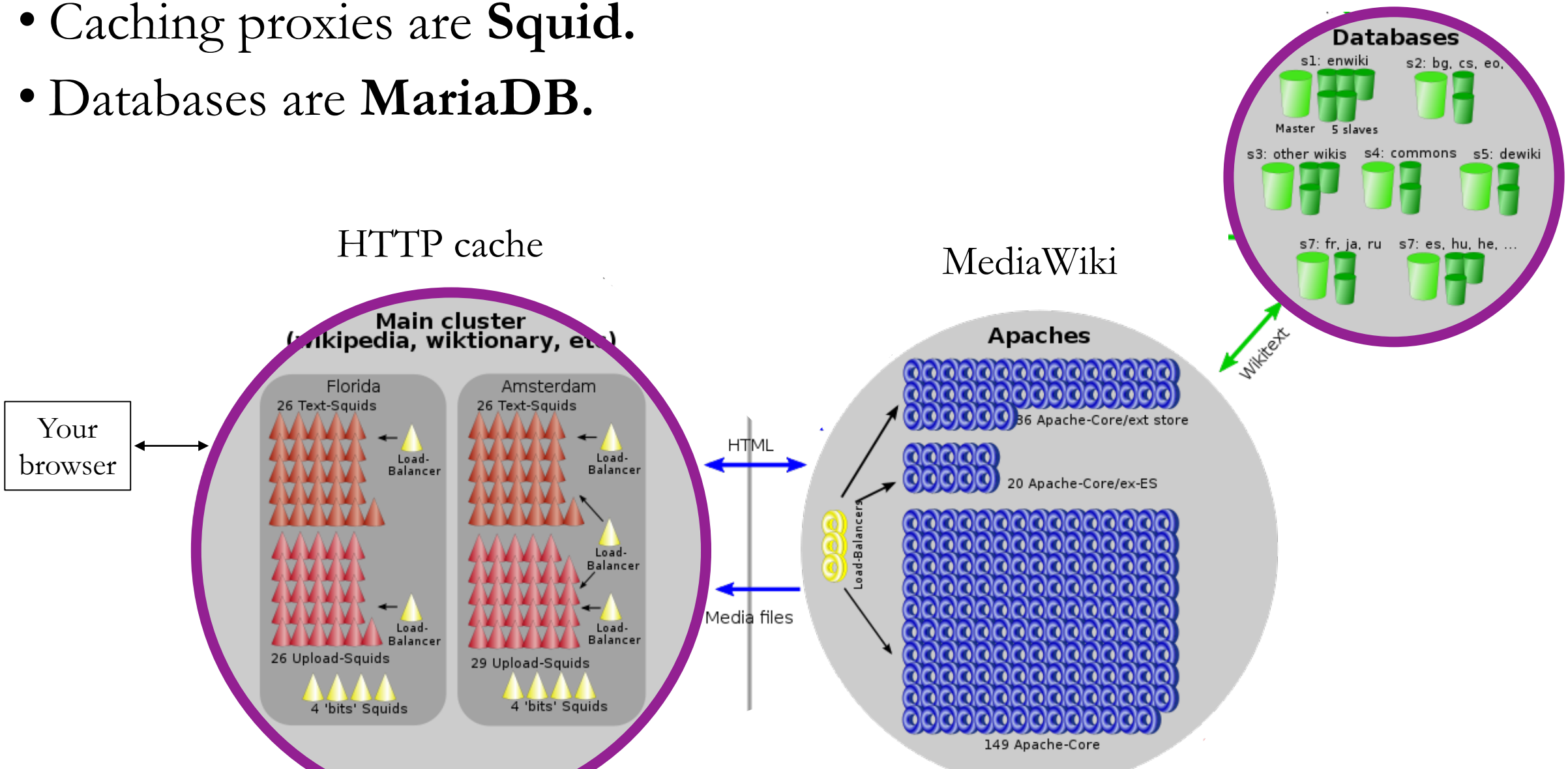
Wikipedia is mostly one big PHP app

- The app uses a **monolithic** architecture.
  - Roughly means "one big thing."



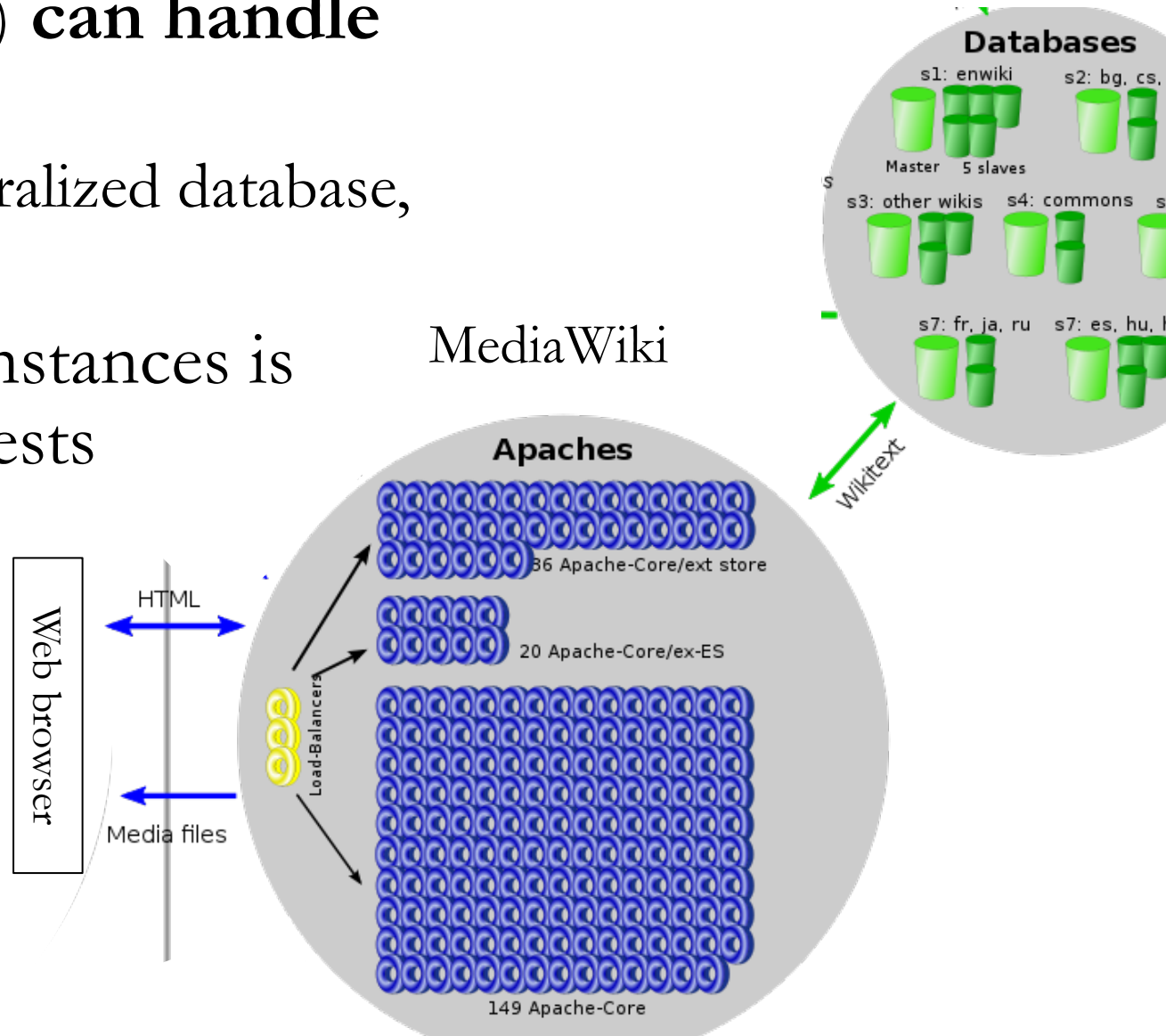
Cache and Database are "off-the-shelf" software

- Caching proxies are **Squid.**
- Databases are **MariaDB.**



Monolithic apps

- Each of instance of Mediawiki (⊙) **can handle any request on its own**.
  - *Caveat:* it needs the help of the centralized database, but let's ignore the DB for now.
- The only reason we have 200 of instances is to handle many independent requests in parallel.
  - They're interchangeable *clones*.
- When a developer is testing new code, they can just run one instance locally.



Advantages of a Monolithic design

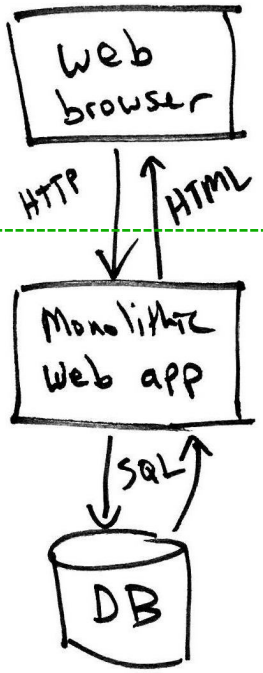
- Easy to build, deploy, test, coordinate, share data.
  - Certainly, the best choice for simple apps.
  - Even some very large services (eg., Facebook) use a monolithic design.

Disadvantages

- Creates bottlenecks in SW development processes.
  - Lots of developers working on one codebase – lots of coordination/merging.
- One huge codebase can lead to messy, fragile code.
  - A change *here* can cause unexpected bugs *there*.
- The whole app must be redeployed for small new functionality.
- Must choose *one* programming language, build system, runtime env.

Breaking up a monolithic architecture, example

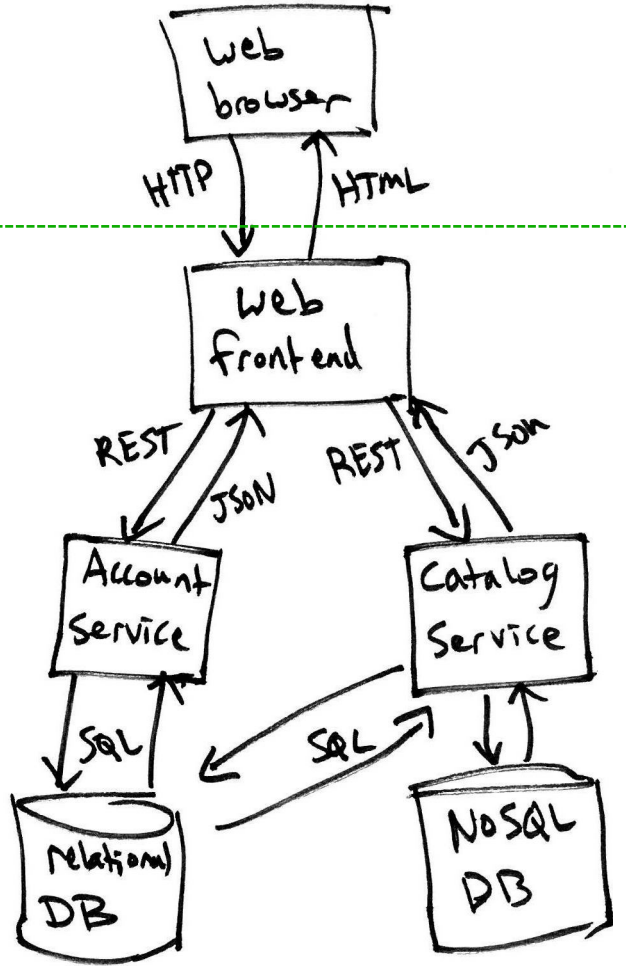
One big service



User's machine

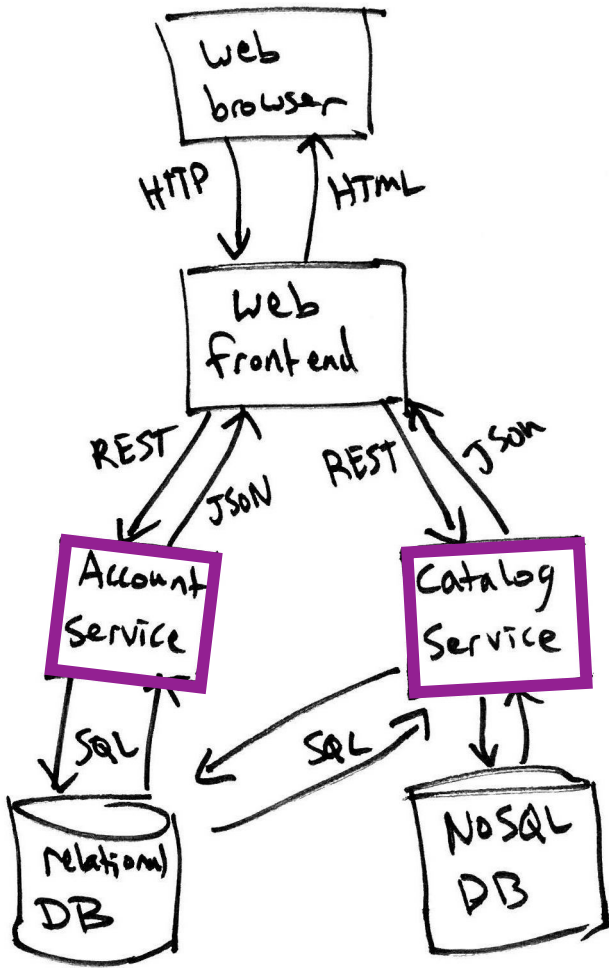
The cloud

Several services



Microservices

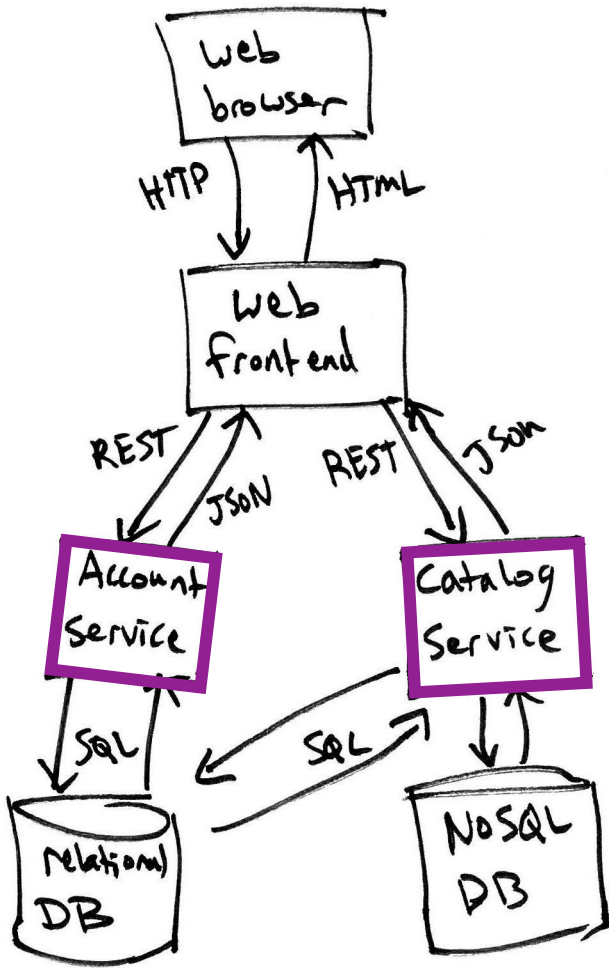
- In a **Service Oriented Architecture** many smaller services work together. Recently, this idea has been re-branded as **Microservices**.
- Responsibilities are split among specialized services.



- To the outside world it still looks like one app, but internally there are many different apps working together.
- Notice that microservices may interact with other services, databases, etc. to do their job.

Microservice interactions

- Each microservice is a **black box** to the rest of the system. It's an independent service.
- Microservice handles requests from the network.
- Implementation (and programming language) details are hidden from the rest of the system.



- However, a clear and language-independent network-level API is needed to specify the format of requests and responses.
  - From last lecture, could be:
    REST, Thrift, ProtoBuf, GraphQL, etc.

A few microservice disadvantages

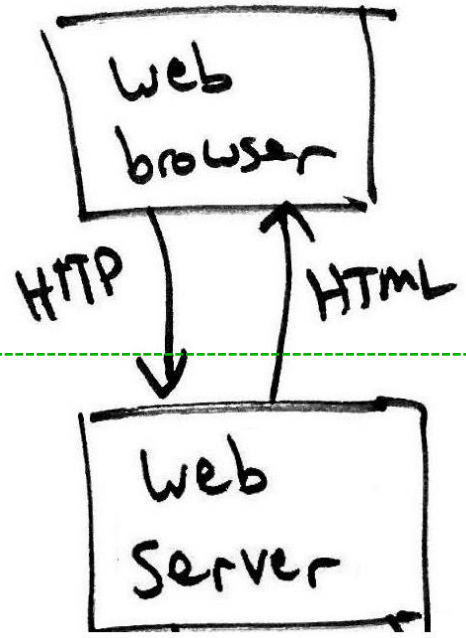
- More difficult to trace through request handling code for debugging, because request handling spans many apps.
  - On the other hand, it's easier to write tests for each smaller service.
- (ACID) **Transactions** are more difficult to support.
  - A monolithic design can manage a single DB connection with a transaction that can be rolled-back.
- Developers get silo-ed/isolated in sub-projects.
  - Collaboration and innovation can be blocked.

Developing a service with a team

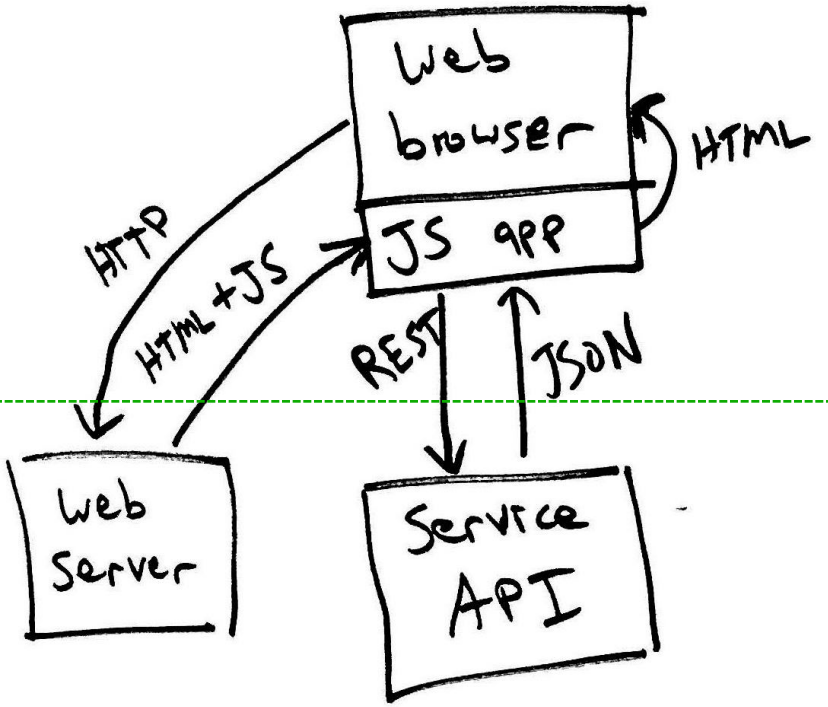
- Microservices isolate codebases with clear network API boundaries, allowing work to proceed in parallel.
- When starting a new project, a typical design process will:
  - Organize the system into several services and databases.
  - Agree on the network-level API for each service.
  - Assign engineers to each service, and build them independently!
- If your application interacts with a service that is under construction, then you can build a quick **mock** of the service.
  - **Mock** services obey the network-level API, but return hard-coded data for testing purposes.

Traditional web app *vs.* JavaScript Single-Page app

- Server generates HTML dynamically.
- Browser doesn't run much JS.
  - Eg., Wikipedia.



- JS app runs in the browser, makes REST requests and generates HTML.
  - Eg., Facebook & other React apps.



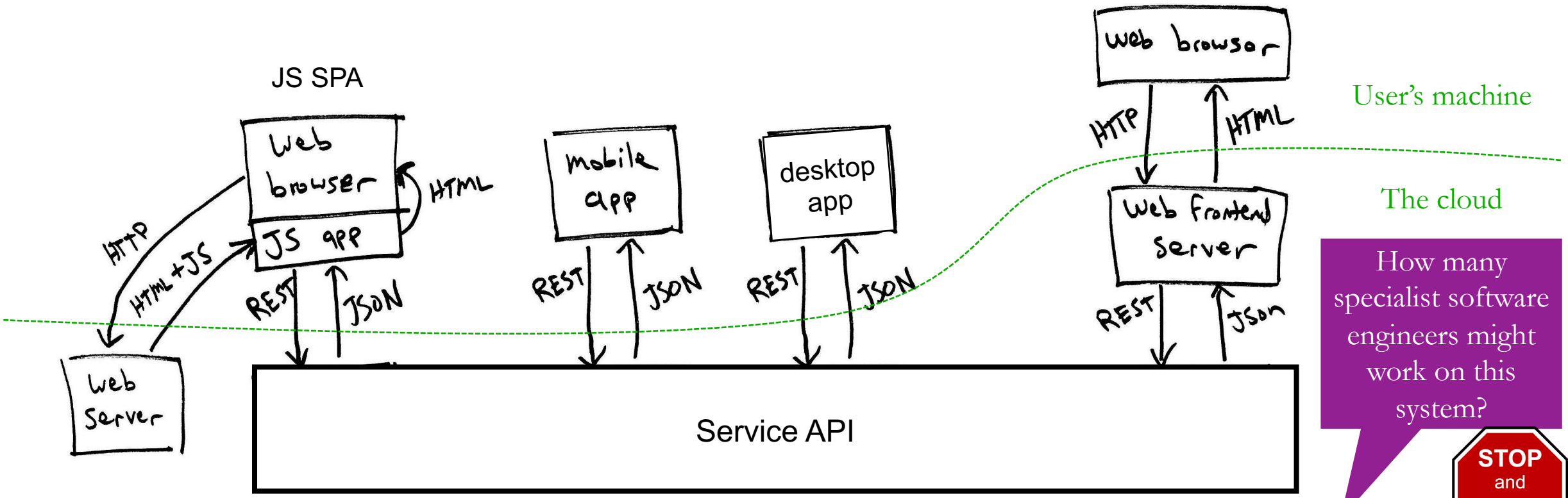
User's machine

The cloud

*Note: many apps combine these two styles.

Cross-platform architecture

- A service API (eg., REST or GraphQL) is essential for supporting mobile and desktop apps with cloud-based data and services.
- A single API can handle all client types.



Review

- Introduced **microservices** as an alternative to **monolithic** design.
- Services are **black boxes**, exposing **network APIs**.
  - Decouples development of different parts of the system.
  - Network APIs define the format and meaning of requests and responses.
- JS **Single-page Applications** (SPAs) interact directly with services.
  - Moves UI concerns away from backend code.
- In a **cross-platform system design**, the same backend service/API can serve mobile, web, and desktop apps.