System Deployment & Benchmarking

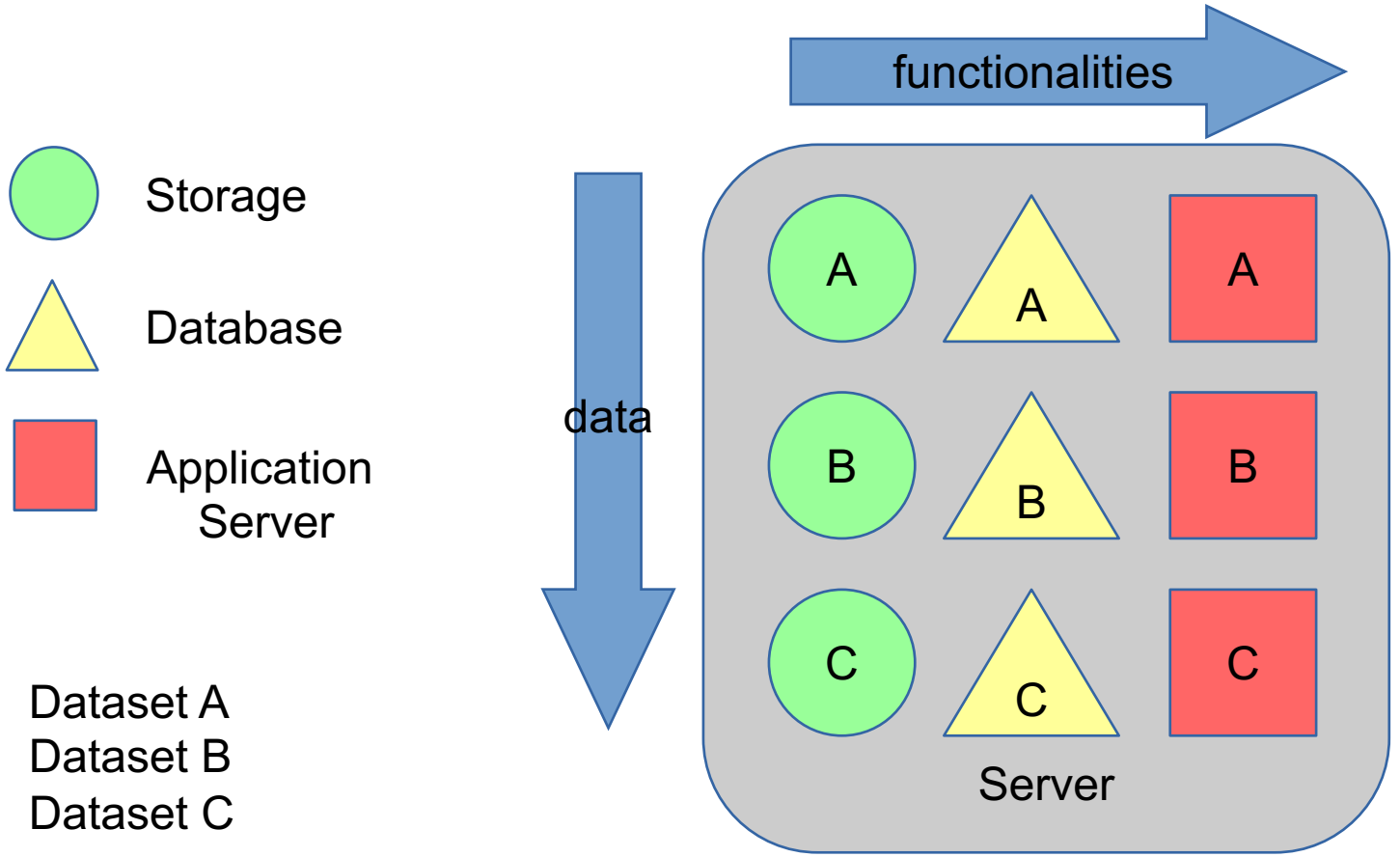
Goals

- Main concerns: **why distributed systems**?
  - Modularity, decoupling different concerns.
  - Performance.
  - Dependability.

- Main architectures: **how to distribute**?

Monolithic system

- Multiple services for multiple targets in the same server.

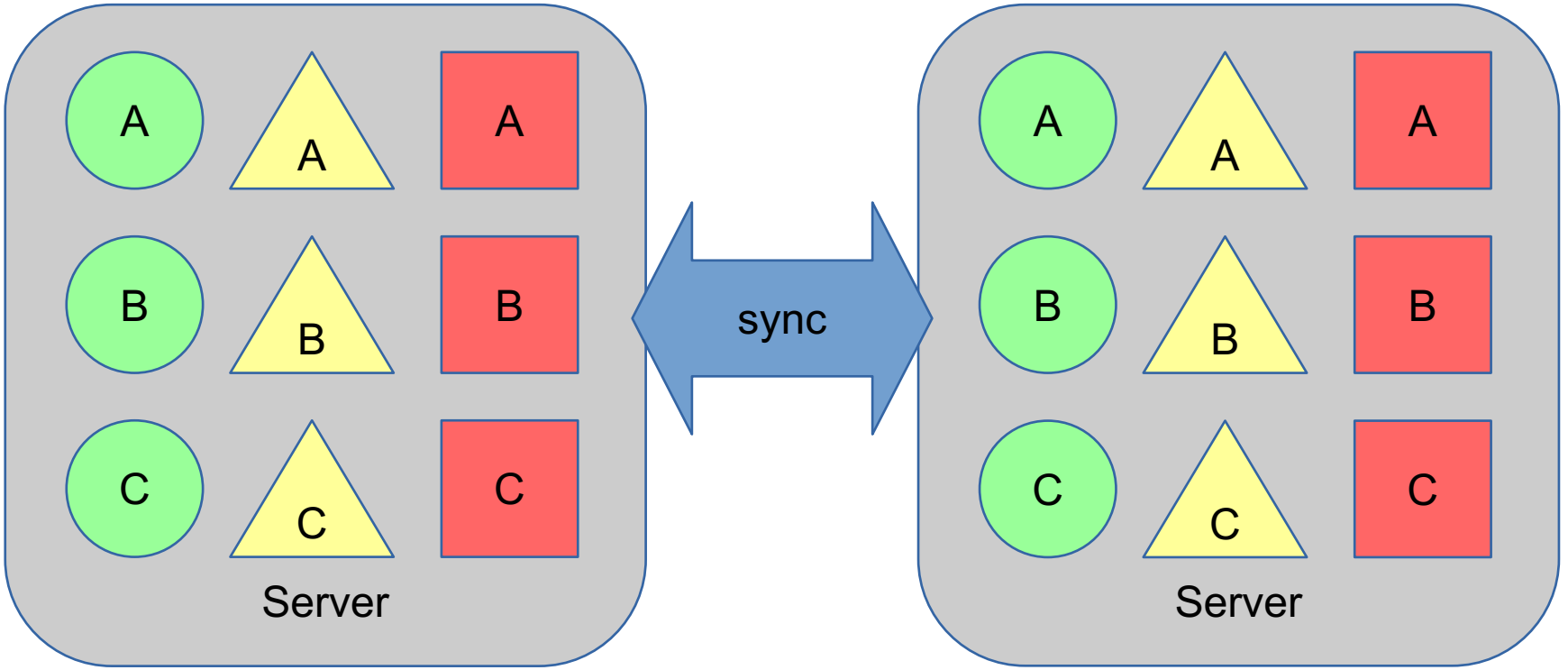


Distributed systems

- Main distribution concerns:
  - Replication
  - Partitioning
  - Service-orientation
- All of these address scaling out a service/application.
- Not mutually exclusive, can be combined.

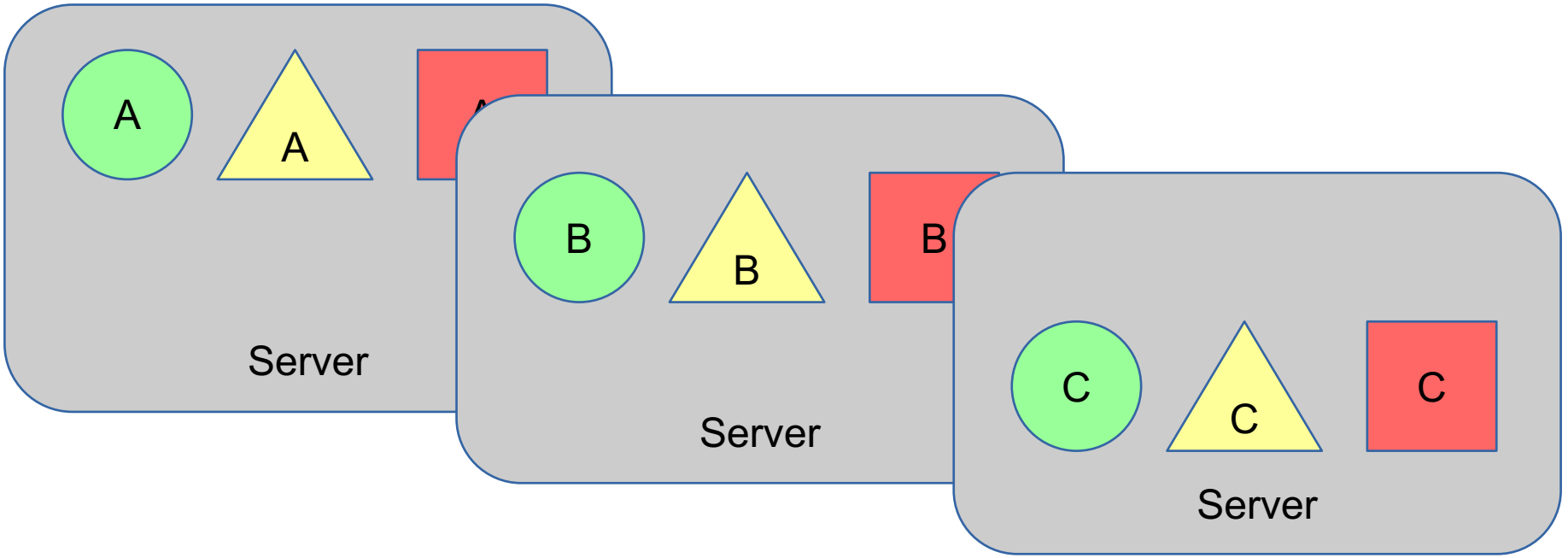
Replication

- Multiple copies of the same data and functionality.
- Addresses resilience and scale-out.



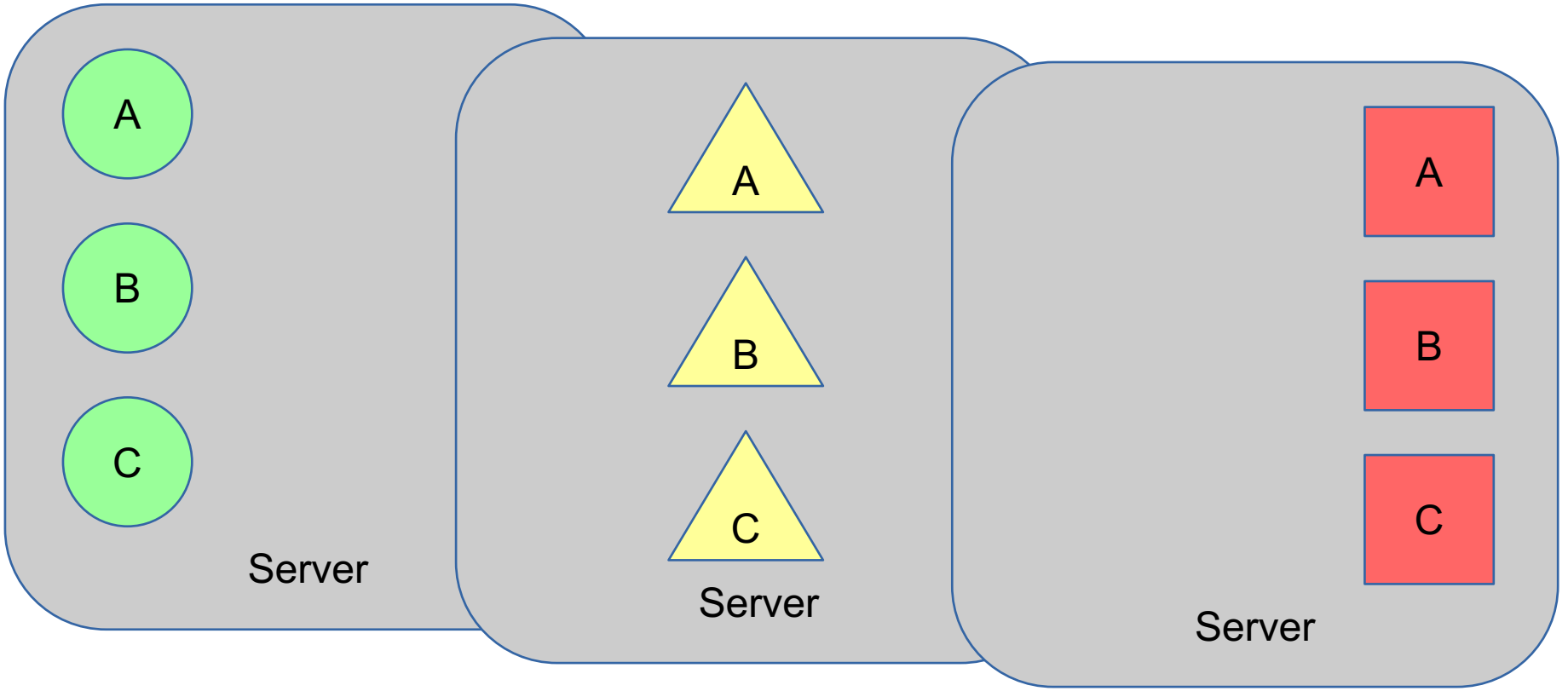
Partitioning

- A server is split horizontally.
- Addresses scale-out.
- Radical partitioning is *sharding.*

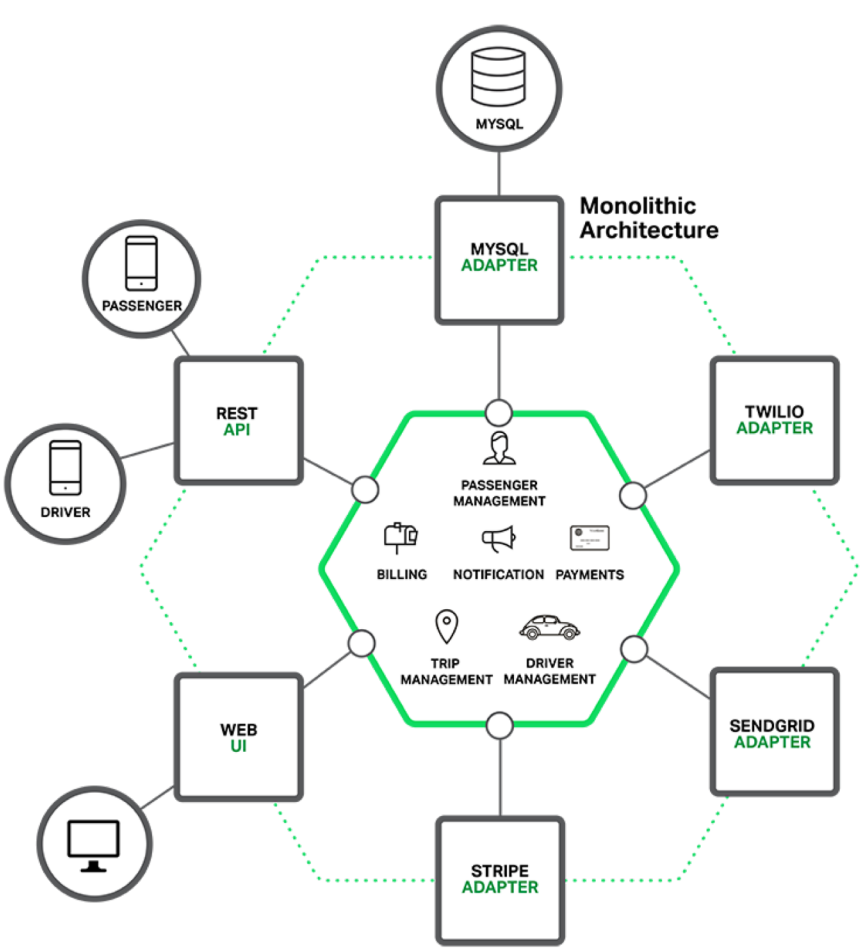


Service-Oriented Architecture (SOA)

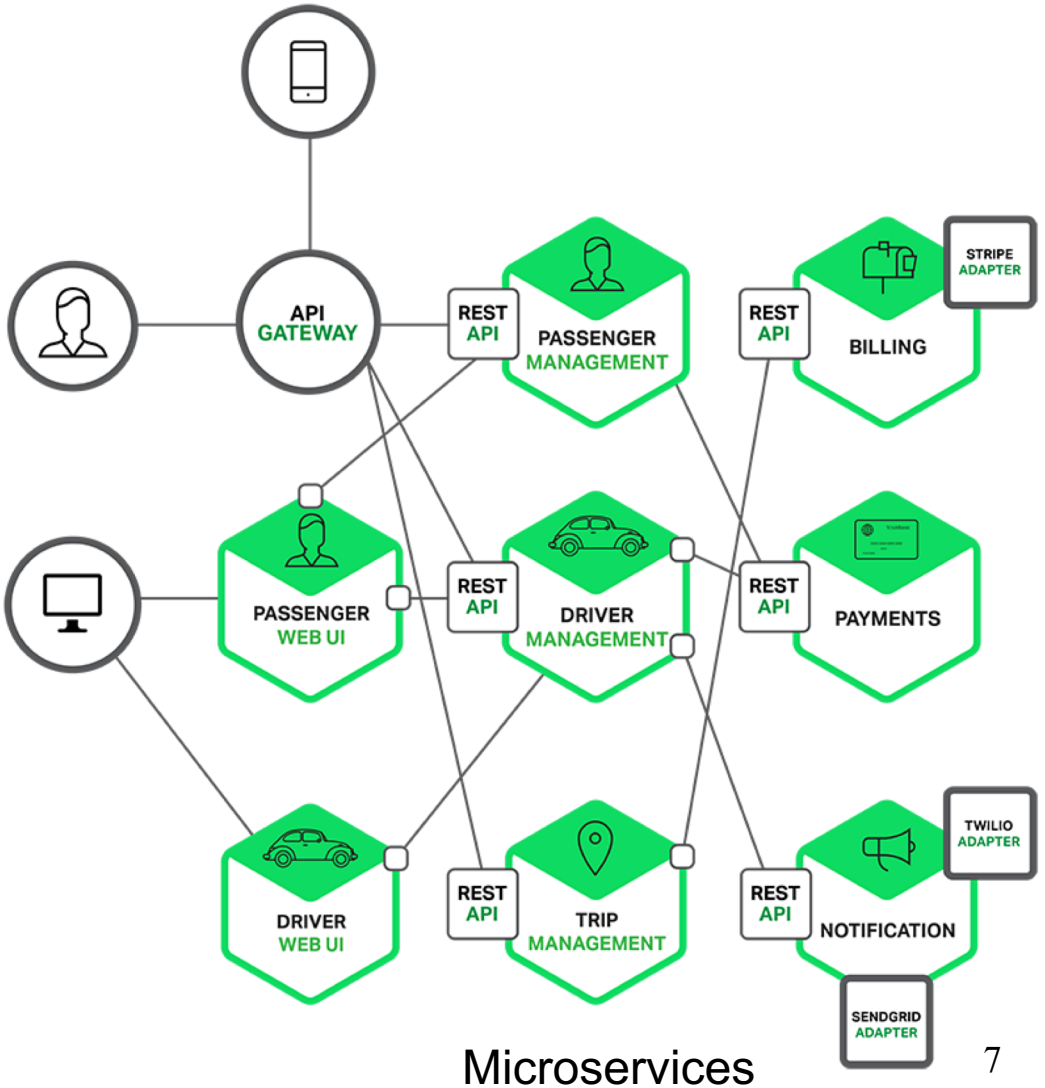
- A server is split vertically.
- Addresses scale-out and modularity.
- Radical service-orientation is *micro-services.*



Monolithic to Microservices



Monolithic

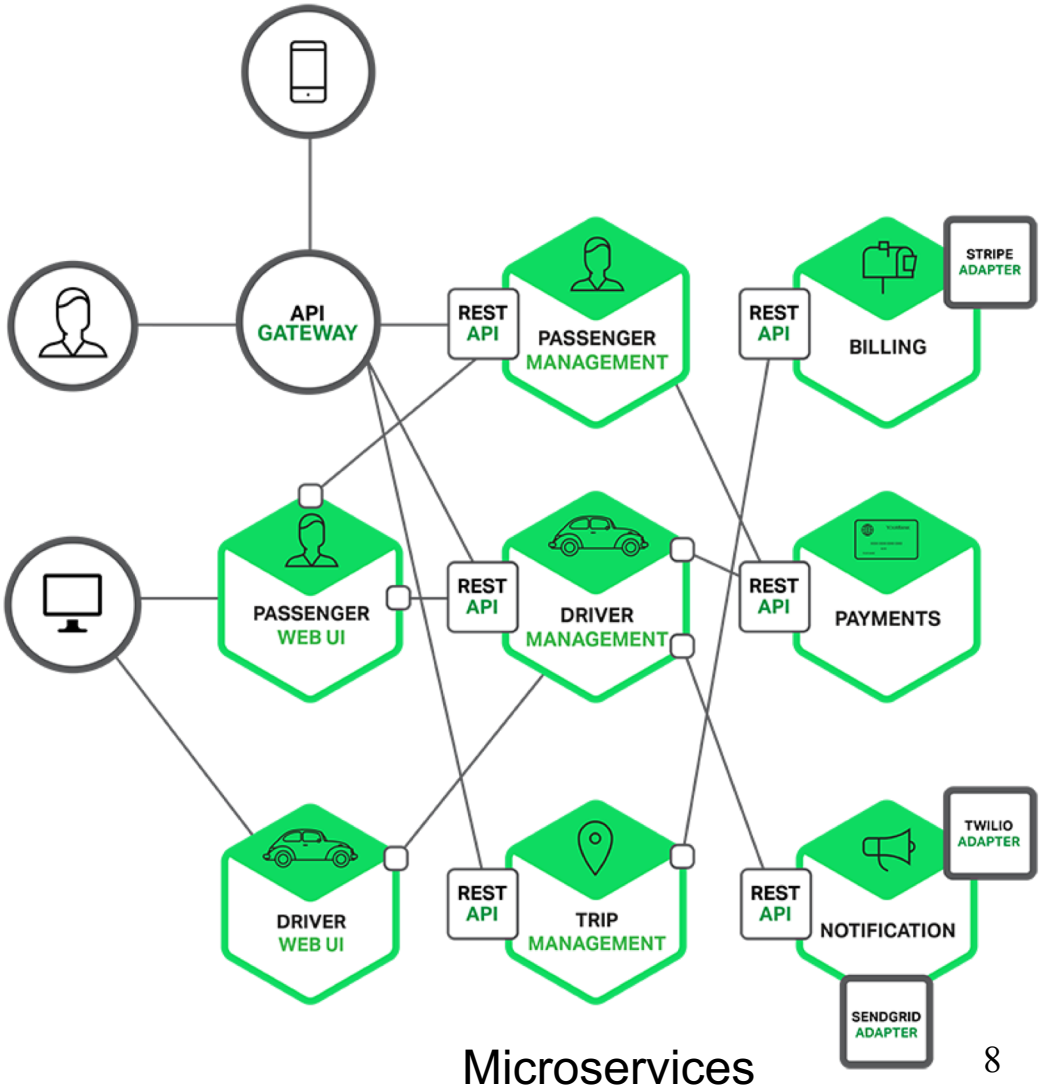


Microservices

Source: https://www.nginx.com/blog/introduction-to-microservices/

Microservices

- Each service implements specific functionality.
- Services can scale independently.
- Decomposition may be troublesome: how micro is micro?
- Consistency.
- Complex deployment and testing



Microservices

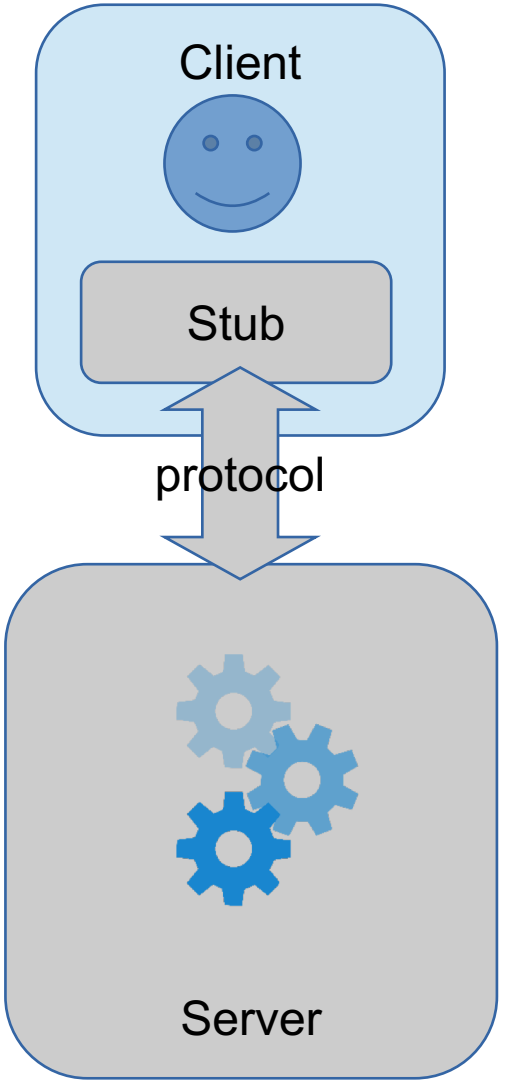
Source: https://www.nginx.com/blog/introduction-to-microservices/

Distributed architectures

- Client-server
- Proxy server
- Master-server
- Server group
- Bus
- Multi-tier

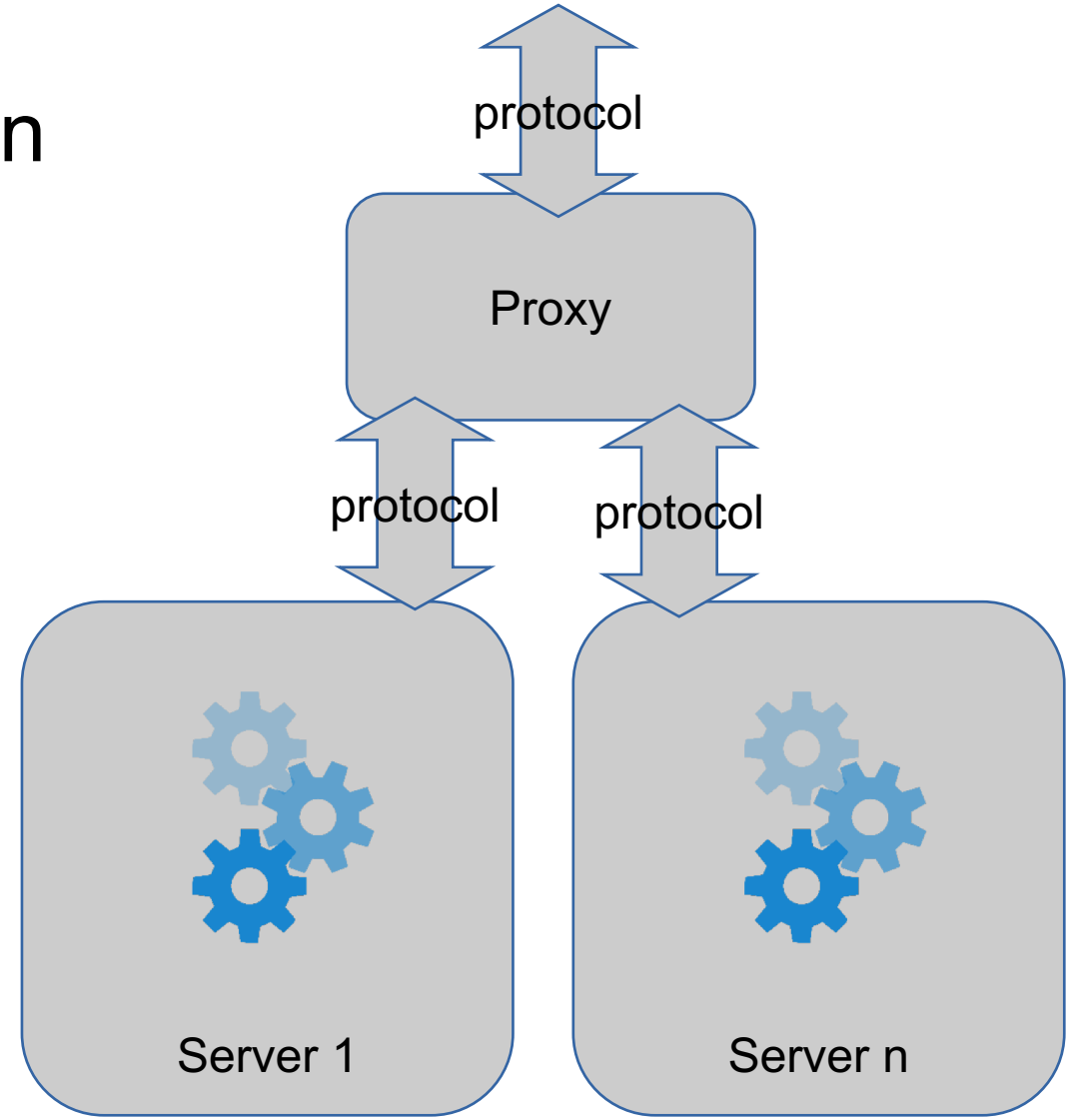
Client-server

- Functionality and data are in the server.
- A stub runs embedded in the client.
- The stub is part of the server software package
- E.g., the Web
  - “protocol” is HTTP



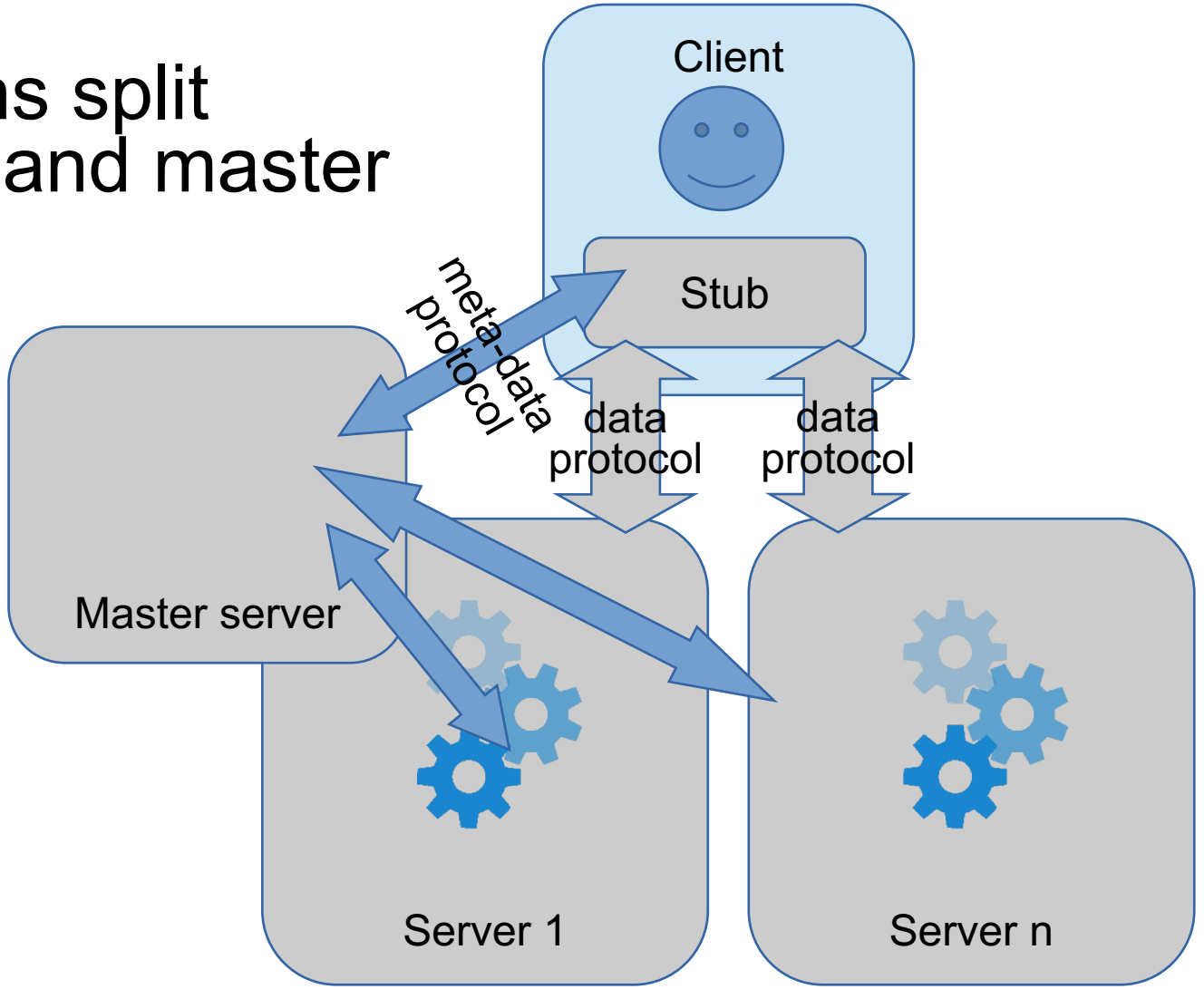
Proxy server

- Multiple servers can be used transparently.
- The proxy is a performance and availability bottleneck.
- E.g. MongoDB



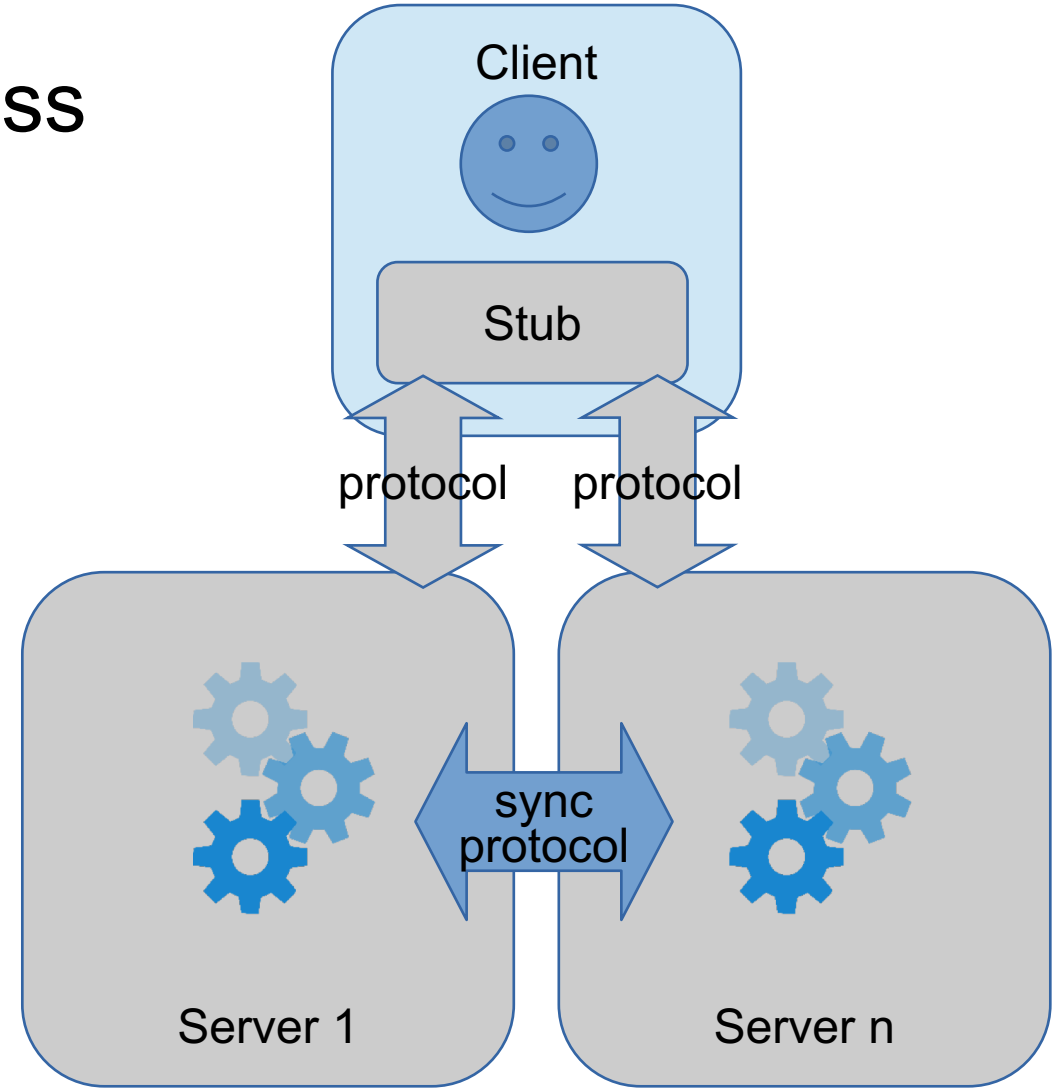
Master server

- Proxy functions split between stub and master server
- **Scale out!**
- E.g. HDFS



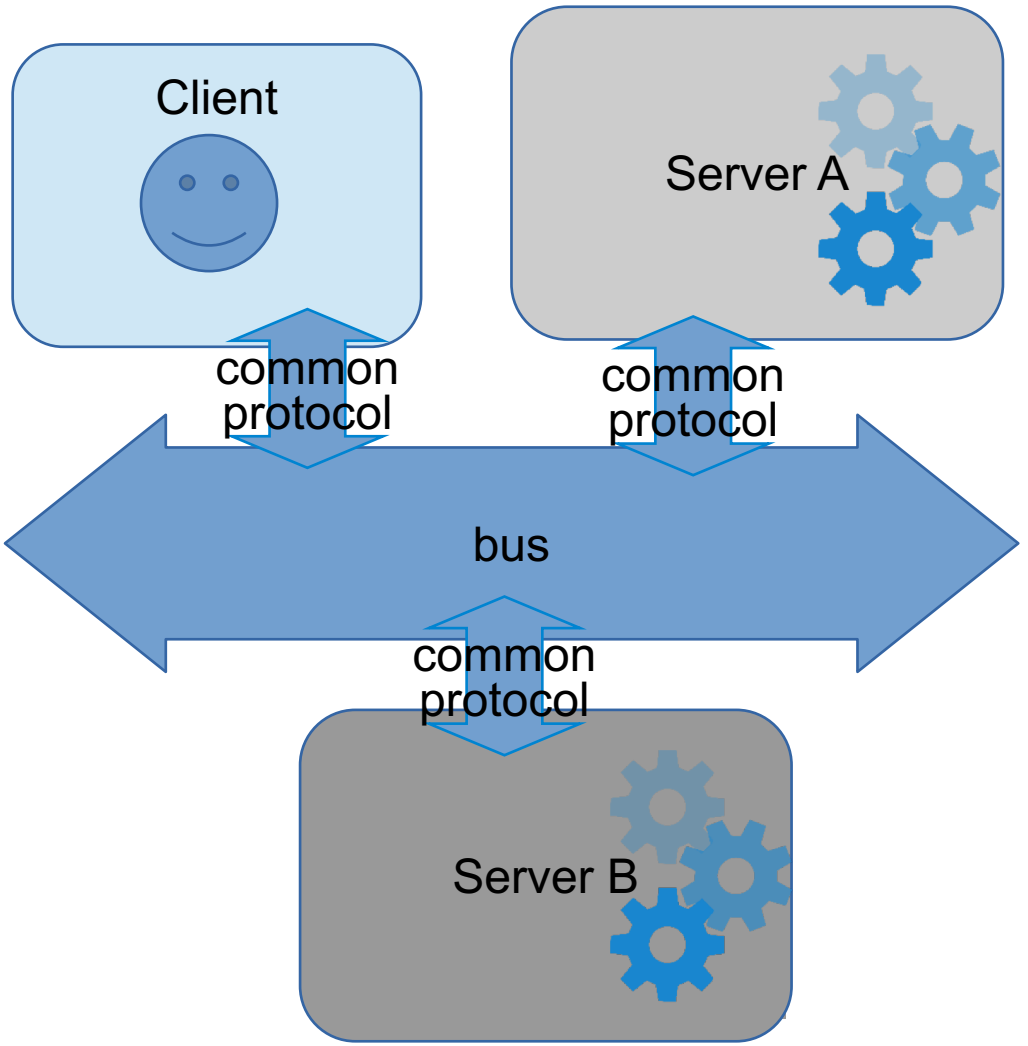
Server group

- All servers can process requests.
- Coordination may be necessary.
- **Resillience!**
- E.g. ZooKeeper

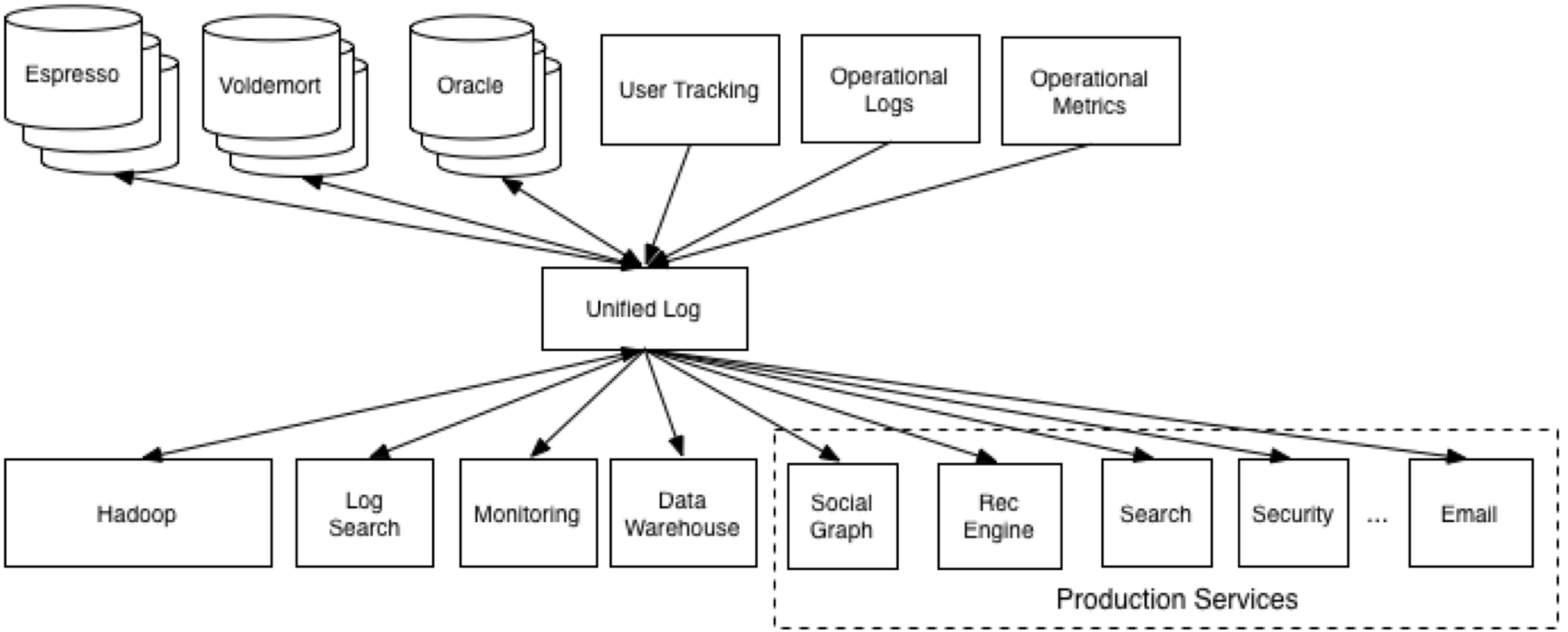


Bus

- The bus routes messages.
- Participants publish and consume messages to/from the bus.
- Decouples producers from consumers.
- **Flexibility!**
- E.g. Kafka



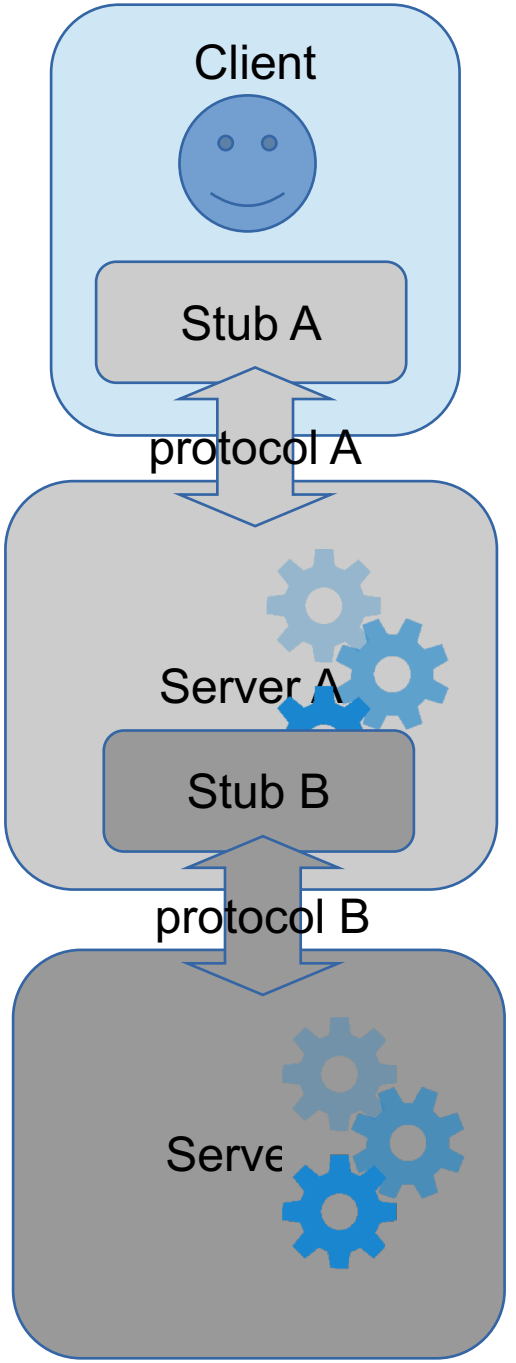
Apache Kafka



LinkedIn

Multi-tier

- Each server acts as a client of the next tier.
- Allows independent deployment and scaling of different functionality.
- E.g. AS + DBMS:
  - “protocol A” == Web (e.g.)
  - “Stub B” == Database Driver!
  - “protocol B” uses SQL



State in multi-tier

- Persistent state is harder to replicate and shard.
- Computation is easier to replicate and shard.
- No state in upper tiers:
  - Web browser!
- Transient / cached state in middle tiers:
  - Application server
- Persistent state at lower tiers:
  - Database