Microservices Design Patterns for Java Applications

Vineet Reynolds · Red Hat

Who am I ?

What does this talk cover ?

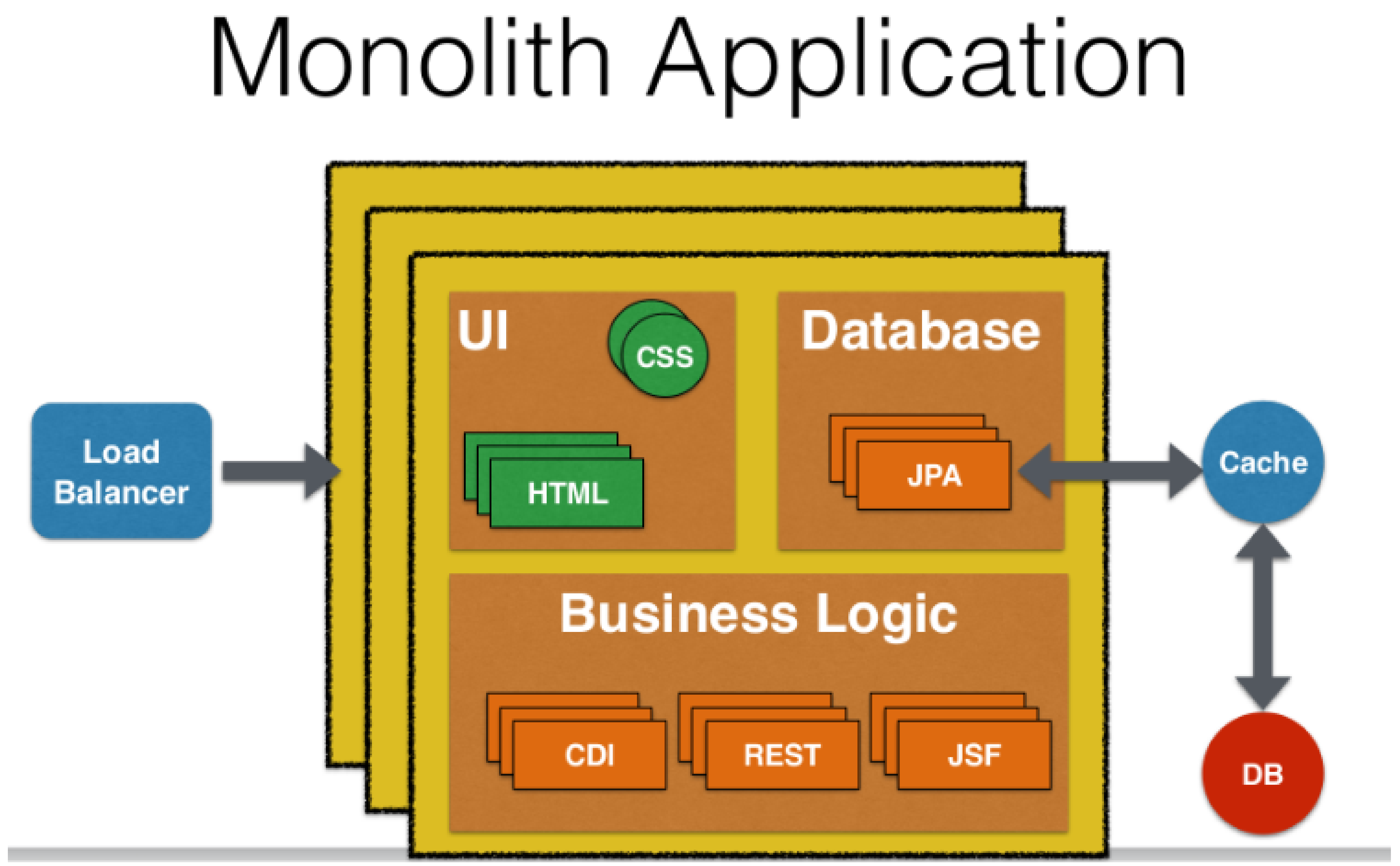
What does this talk cover ?

- MicroServices - a brief introduction
    - The monolith
    - Decomposing monoliths

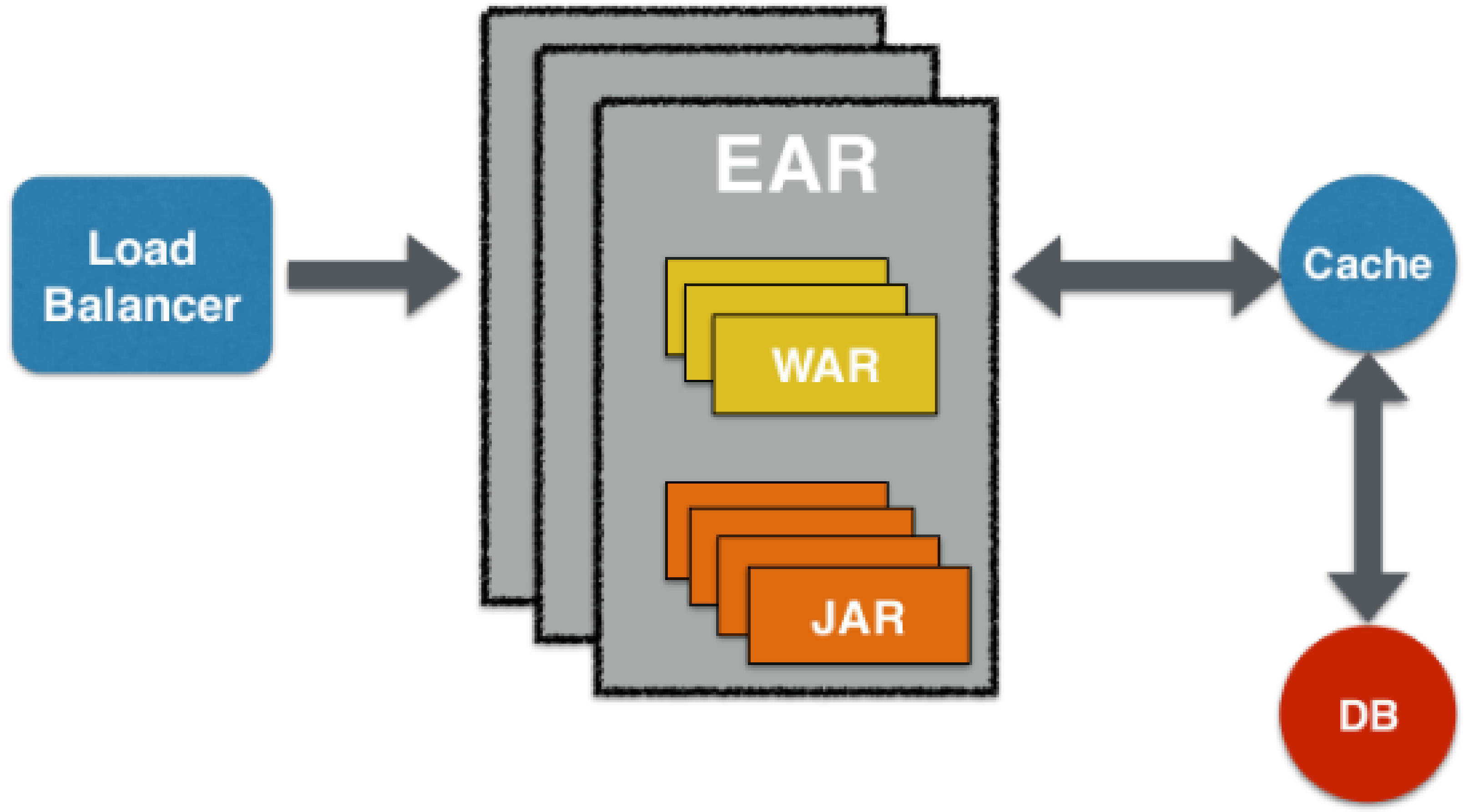
What does this talk cover ?

- MicroServices - a brief introduction
  - The monolith
  - Decomposing monoliths
- Patterns
  - Aggregator
  - Proxy
  - Chained services
  - Branch
  - Shared data
  - Asychronous messaging

The monolith



Monolith Application



The monolith

- Do they have advantages ? Oh yes.
  - Packaged and deployed as a single unit. Relative ease of rolling back from failure.
  - Homogenous design - could be a good thing to counter chaos.
  - Easy to test - services are always available.
  - Simple to develop - single codebase.
  - Easy to scale horizontally.
  - Organize operations around a single team.

The monolith

- Do they have disadvantages ? Oh, yes, a lot more.
    - Long deployment cycles (lasting hours or even days)
    - May not use the right framework/tool/language for each domain
    - Acquires too many responsibilities over time

Characteristics

- Many smaller (fine grained), clearly scoped services
  - Single Responsibility Principle (`S` in `SOLID`, but for services)
  - Independently managed
- Clear ownership for each service
  - Independently deployable, leading easily to CI+CD
  - Typically need/adopt the "DevOps" model

What are Microservices?

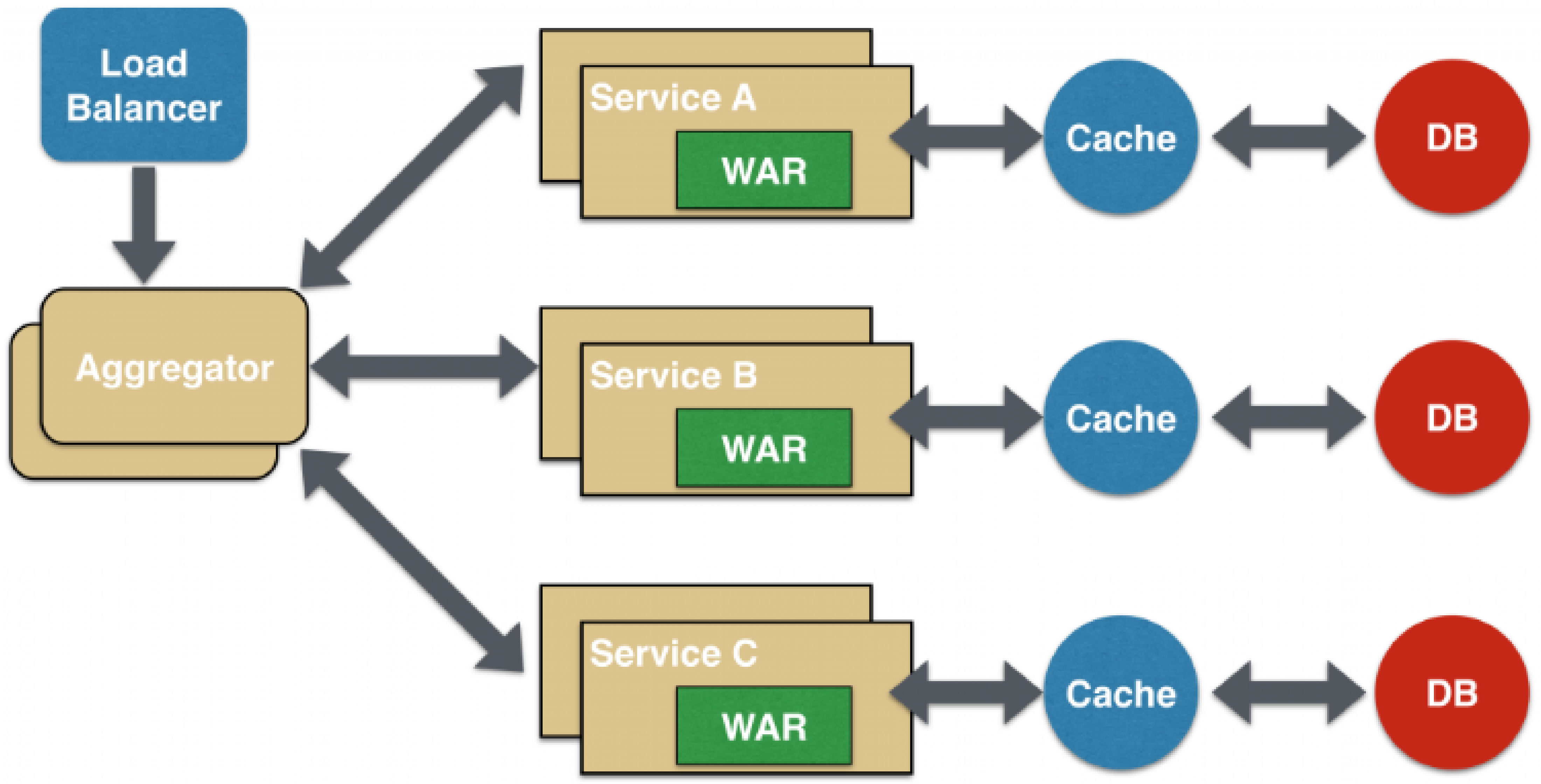
- Hipster SOA?
- Fine-grained SOA?
- Focus on ESBs in SOA?
- SOA done right?

Decomposing the monolith into services

- Verb or usecase - e.g. Checkout UI
- Noun - e.g. Catalog product service
- Single Responsible Principle - e.g. Unix utilities

Microservices Patterns

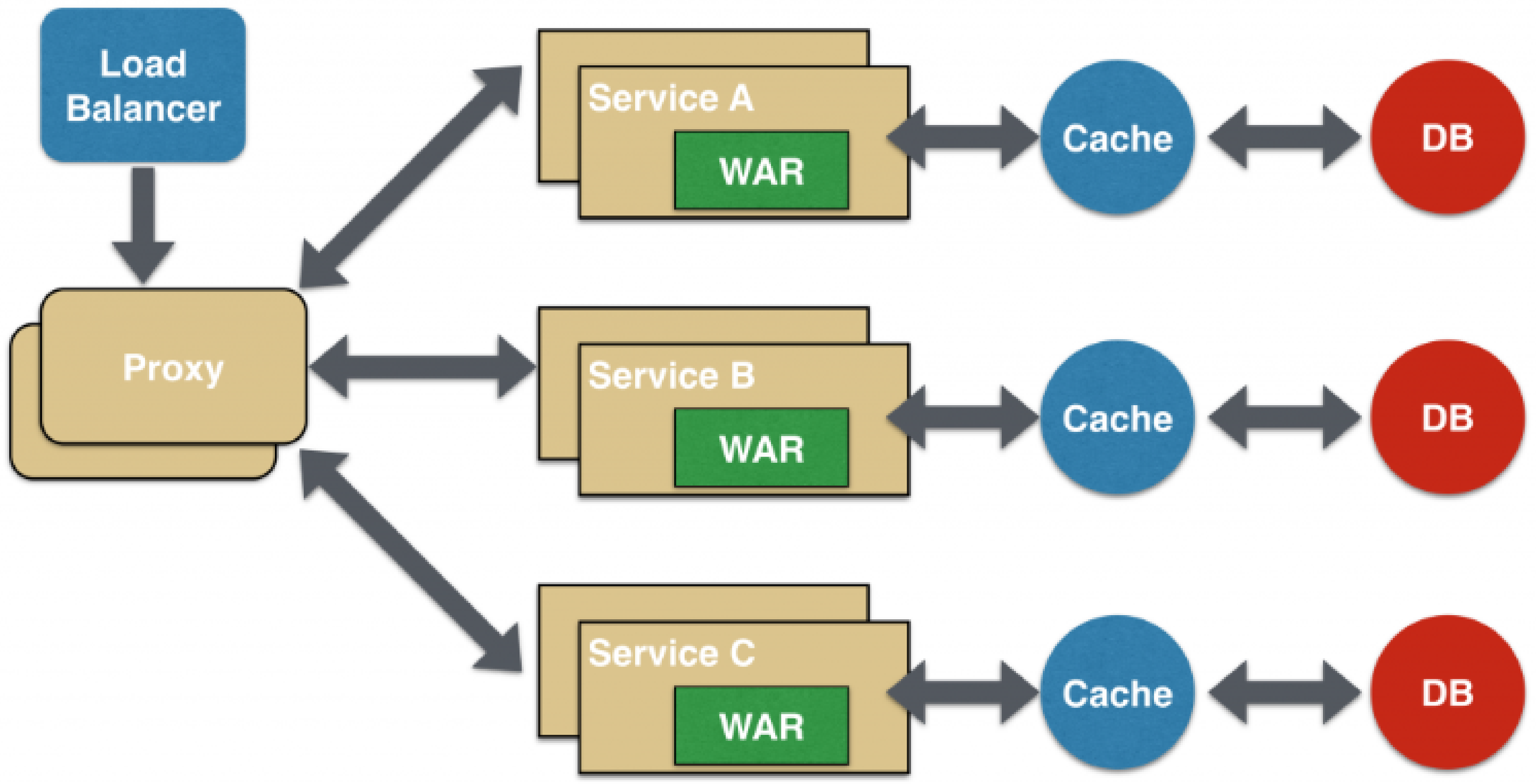
Aggregator


Load Balancer
Aggregator
Service A
WAR
Cache
DB
Service B
WAR
Cache
DB
Service C
WAR
Cache
DB

Aggregator

- Similar to an API gateway.
- Provides a simple interface to a complex system.
- Can tranform data from downstream services.

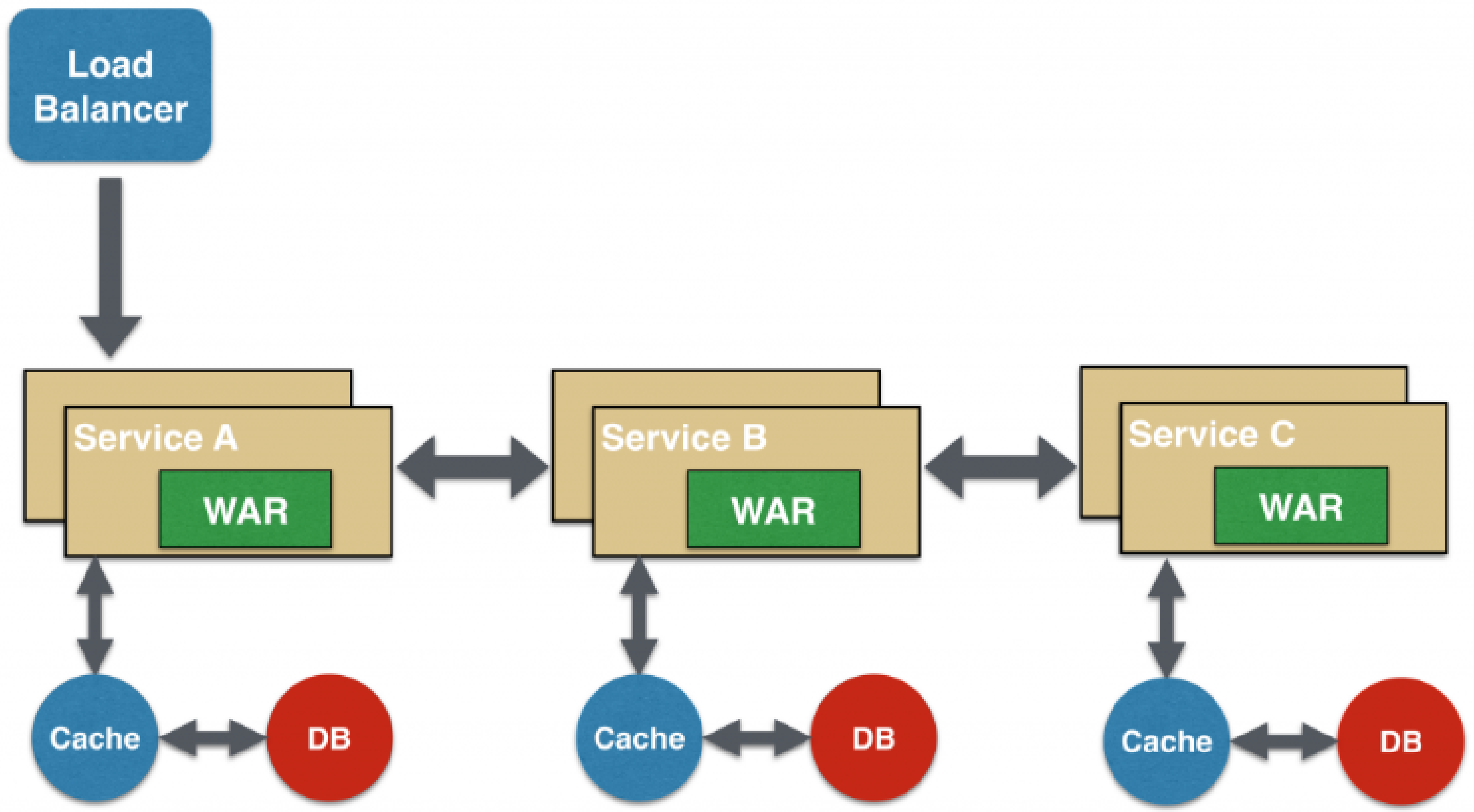
Proxy


Load Balancer
Proxy
Service A
WAR
Service B
WAR
Service C
WAR
Cache
Cache
Cache
DB
DB
DB

Proxy

- Similar to the aggregator.
- Proxies don’t aggregate; they merely delegate to a downstream service.
- A smart proxy, may transform data from downstream responses.

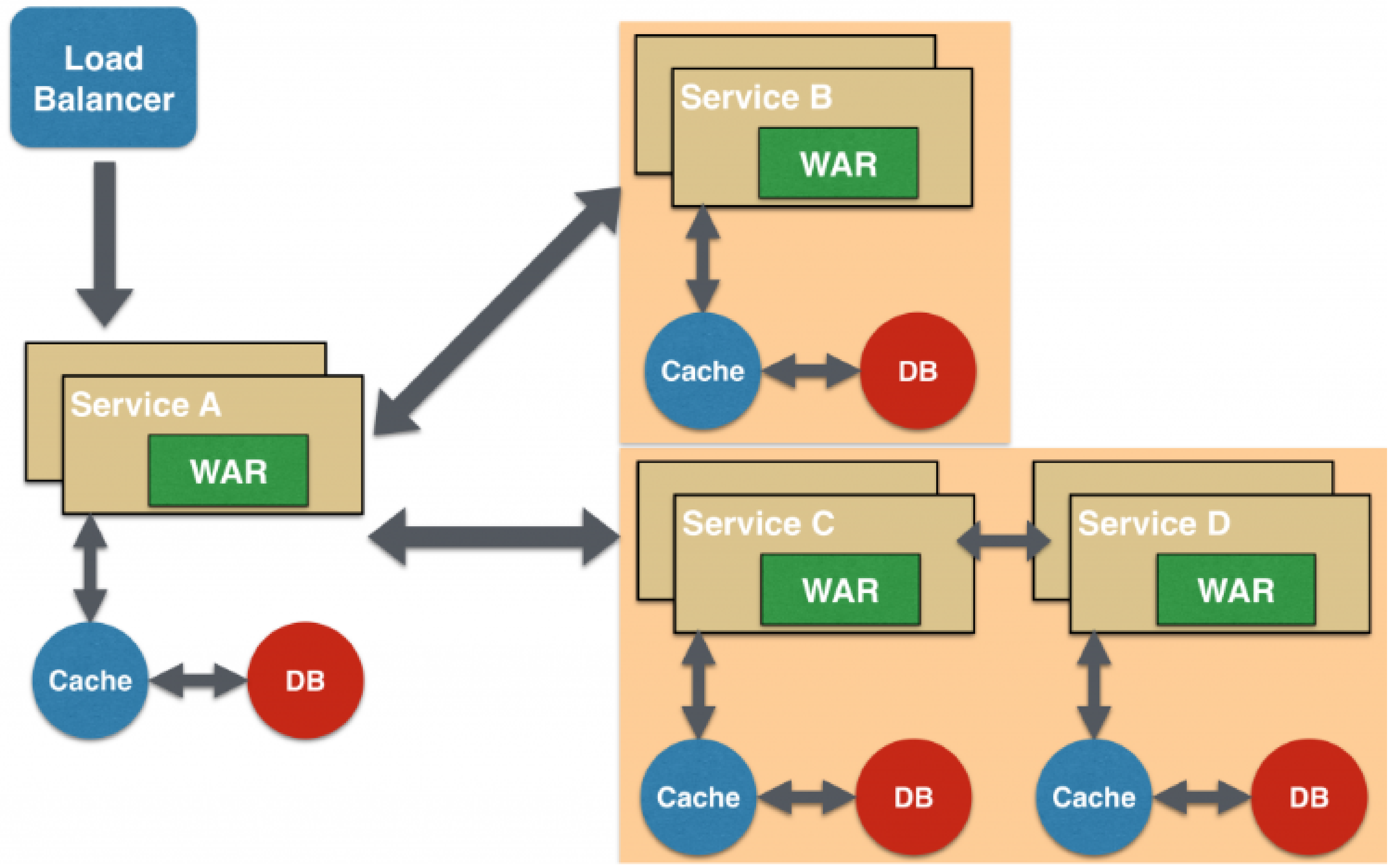
Chained services



Chained services

- Commonly used when business workflows are to be modelled in service interactions.
- Each service adds value in the transaction.
- Asynchronosity may need to be introduced in long-running transactions; use in combination with asynchronous messaging.

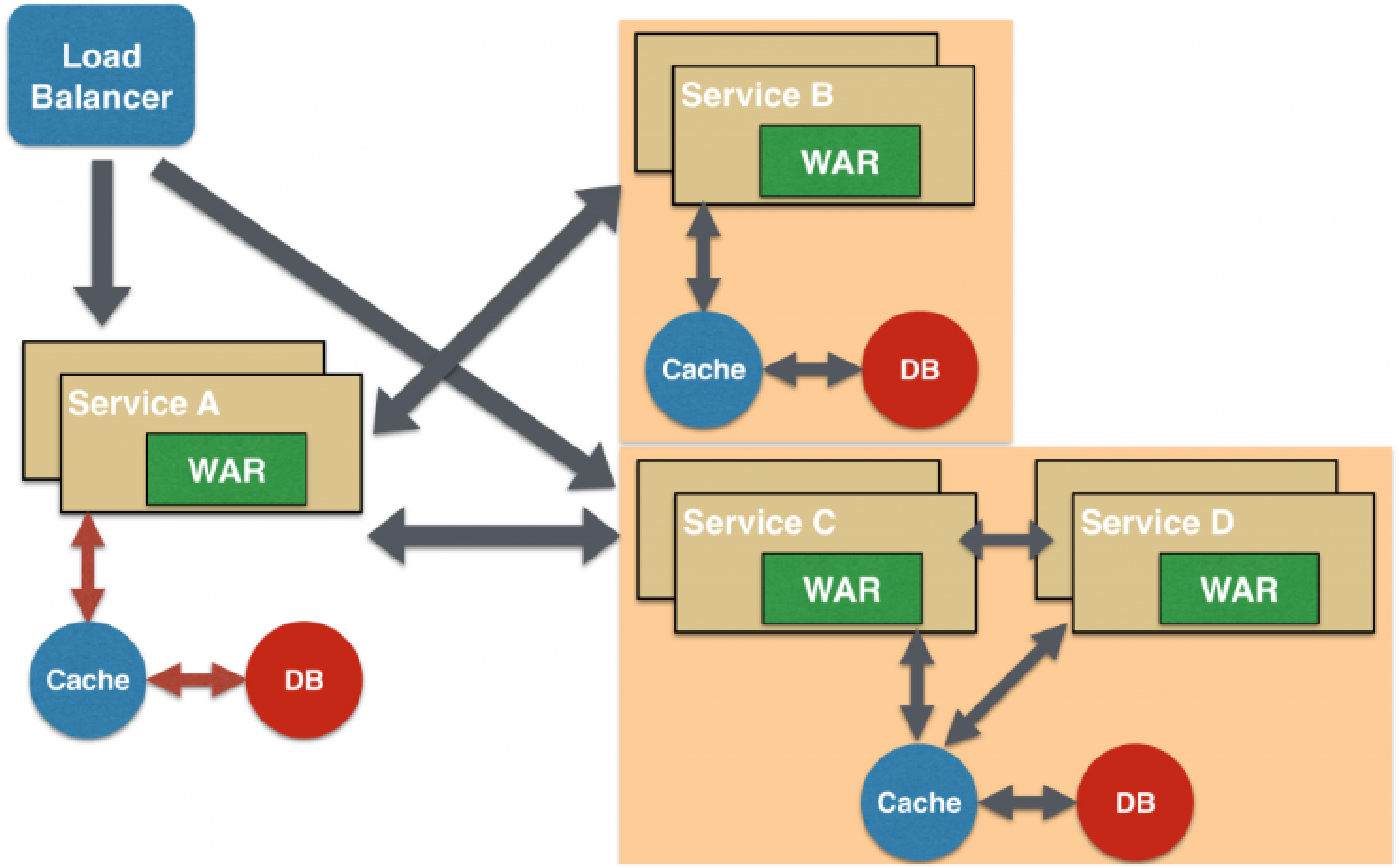
Branch


Load Balancer
Service A
WAR
Cache
DB
Service B
WAR
Cache
DB
Service C
WAR
Cache
DB
Service D
WAR
Cache
DB

Branch

- Similar to the Content-based Router pattern.
- Requests/Messages are routed based on content like URL, headers or request body.
- Different downstream services exist due to siloed nature of business operations.
- Data may be siloed, in conjunction with processing logic, at downstream services.

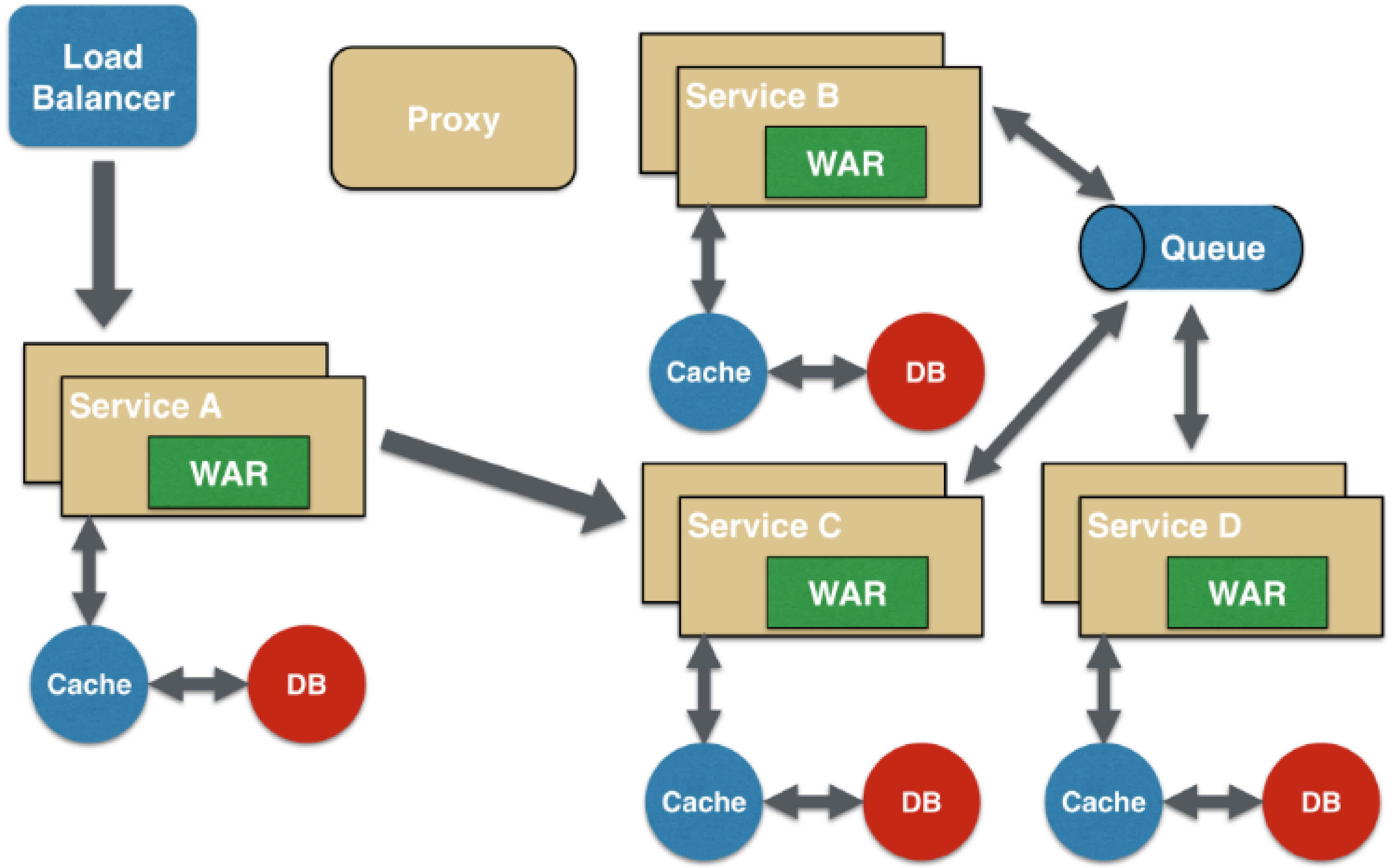
Shared data


Load Balancer
Service A
WAR
Cache
DB
Service B
WAR
Cache
DB
Service C
WAR
Service D
WAR
Cache
DB

Shared data

- Susceptible to mis-use; can devolve to an anti-pattern.
- Locking and transactional semantics may be unclear.
- Much safer to use when semantics are understood.
- Perfect when shared data is read-only.
- Necessary when microservices are strongly coupled to avoid data duplication.

Asynchronous messaging



Asynchronous messaging

- Preferred way to update data not owned by a service.
- Instead, publish events to the owning service.
- Eventually, leads to patterns like event sourcing.

Questions?