

# Refactor your applications using Container and Microservices



Arun Gupta, @arungupta  
Red Hat

## Advantages

### Standardization



ISO standard (modes and equipment). Unique identification number and size type code.

### Flexibility



Commodities, manufactured goods, liquids and refrigerated goods.

### Costs



Low transport costs. Economies of scale at modes and terminals.

### Velocity



Fast transshipment operations. Low terminal turnaround times.

### Warehousing



Own warehouse; simpler and less expensive packaging. Stacking capability.

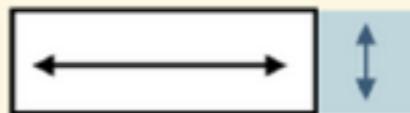
### Security & Safety



Contents unknown to carriers. Reduced spoilage and losses.

## Drawbacks

### Site Constraints



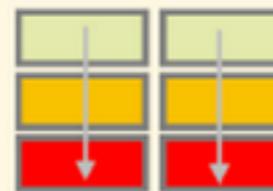
Large consumption of terminal space. Draft issues with larger containerships.

### Capital Intensiveness



Container handling infrastructures and equipment are important investments.

### Stacking



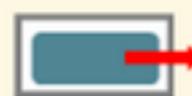
Complexity of arrangement of containers, both on the ground and on modes.

### Repositioning



Divergence between production and consumption; repositioning. 20% of all containers.

### Theft and Losses



High value goods vulnerable to thefts, particularly between terminal and final destination.

### Illicit Trade

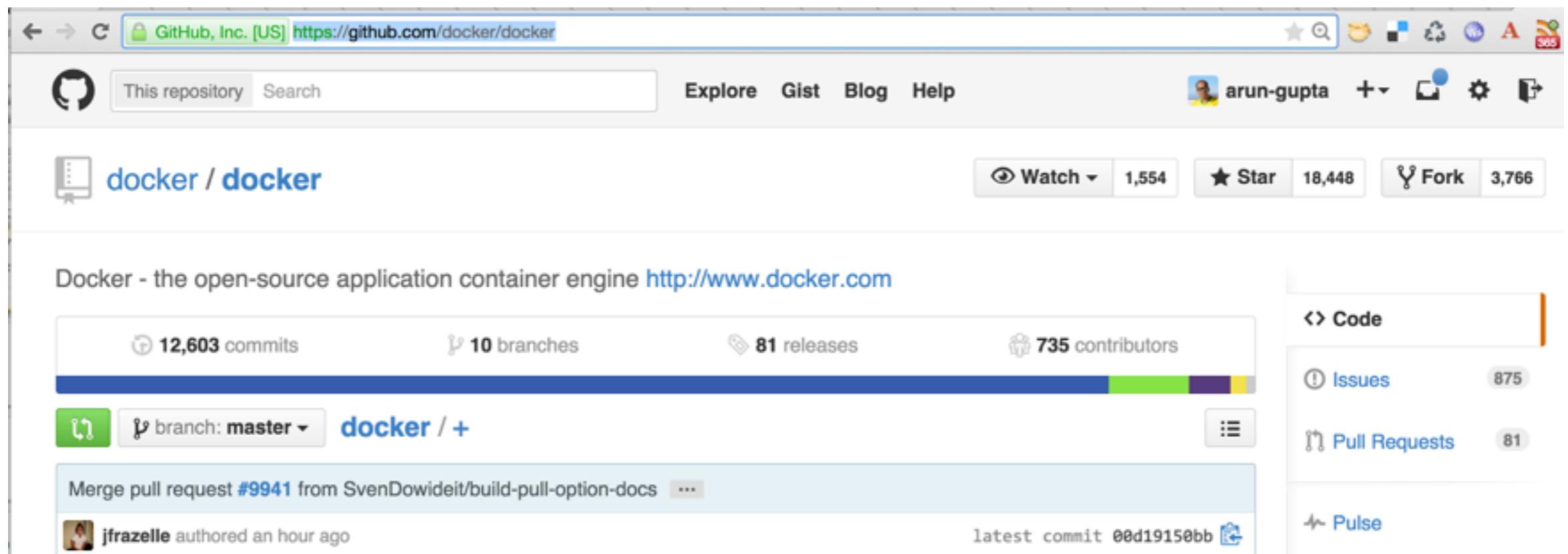


Illicit trade of goods, drugs and weapons, as well as for illegal immigration.

# What is Docker?

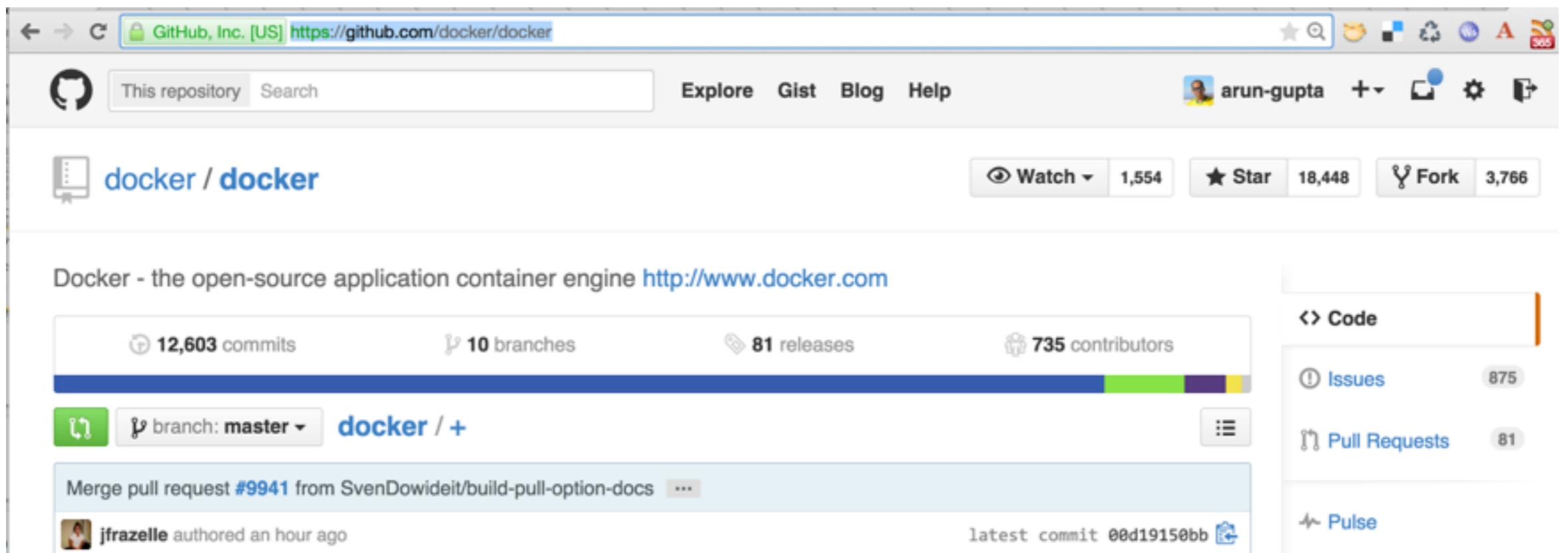
# What is Docker?

- Open source project and company



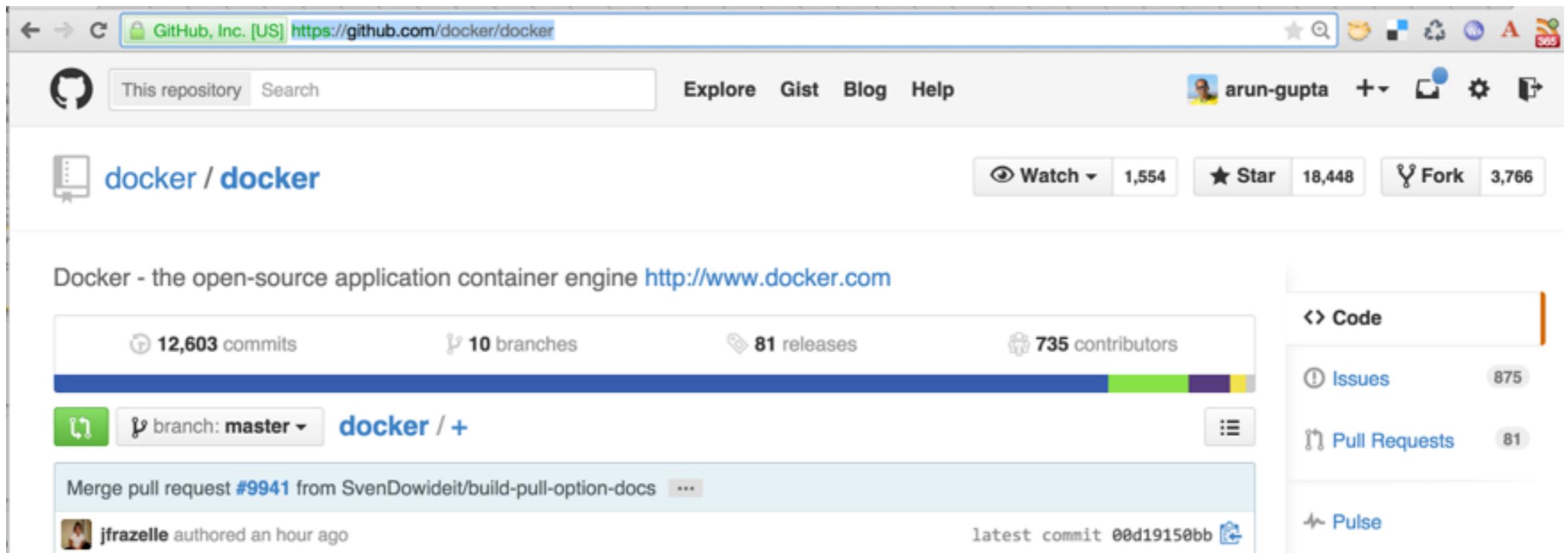
# What is Docker?

- Open source project and company
- Used to create containers for software applications



# What is Docker?

- Open source project and company
- Used to create containers for software applications
- Package Once Deploy Anywhere (PODA)



# Advantages

# Advantages

- Isolation

# Advantages

- Isolation
- Portability - “it works on my machine”

# Advantages

- Isolation
- Portability - “it works on my machine”
- Snapshotting - Upgrade/downgrade application versions

# Advantages

- Isolation
- Portability - “it works on my machine”
- Snapshotting - Upgrade/downgrade application versions
- Security sandbox

# Advantages

- Isolation
- Portability - “it works on my machine”
- Snapshotting - Upgrade/downgrade application versions
- Security sandbox
- Limit resource usage

# Advantages

- Isolation
- Portability - “it works on my machine”
- Snapshotting - Upgrade/downgrade application versions
- Security sandbox
- Limit resource usage
- Lightweight footprint and minimal overhead

# Advantages

- Isolation
- Portability - “it works on my machine”
- Snapshotting - Upgrade/downgrade application versions
- Security sandbox
- Limit resource usage
- Lightweight footprint and minimal overhead
- Simplified dependency

# Advantages

- Isolation
- Portability - “it works on my machine”
- Snapshotting - Upgrade/downgrade application versions
- Security sandbox
- Limit resource usage
- Lightweight footprint and minimal overhead
- Simplified dependency
- Sharing

# Advantages

- Isolation
- Portability - “it works on my machine”
- Snapshotting - Upgrade/downgrade application versions
- Security sandbox
- Limit resource usage
- Lightweight footprint and minimal overhead
- Simplified dependency
- Sharing
- Faster Deployments

# Underlying Technology

# Underlying Technology

- Written in Go

# Underlying Technology

- Written in Go
- Uses several Linux features

# Underlying Technology

- Written in Go
- Uses several Linux features
  - **Namespaces** to provide isolation

# Underlying Technology

- Written in Go
- Uses several Linux features
  - **Namespaces** to provide isolation
  - **Control groups** to share/limit hardware resources

# Underlying Technology

- Written in Go
- Uses several Linux features
  - **Namespaces** to provide isolation
  - **Control groups** to share/limit hardware resources
  - **Union File System** makes it light and fast

# Underlying Technology

- Written in Go
- Uses several Linux features
  - **Namespaces** to provide isolation
  - **Control groups** to share/limit hardware resources
  - **Union File System** makes it light and fast
  - **libcontainer** defines container format

# Is it only Linux?

# Is it only Linux?

- Natively supported in Linux

# Is it only Linux?

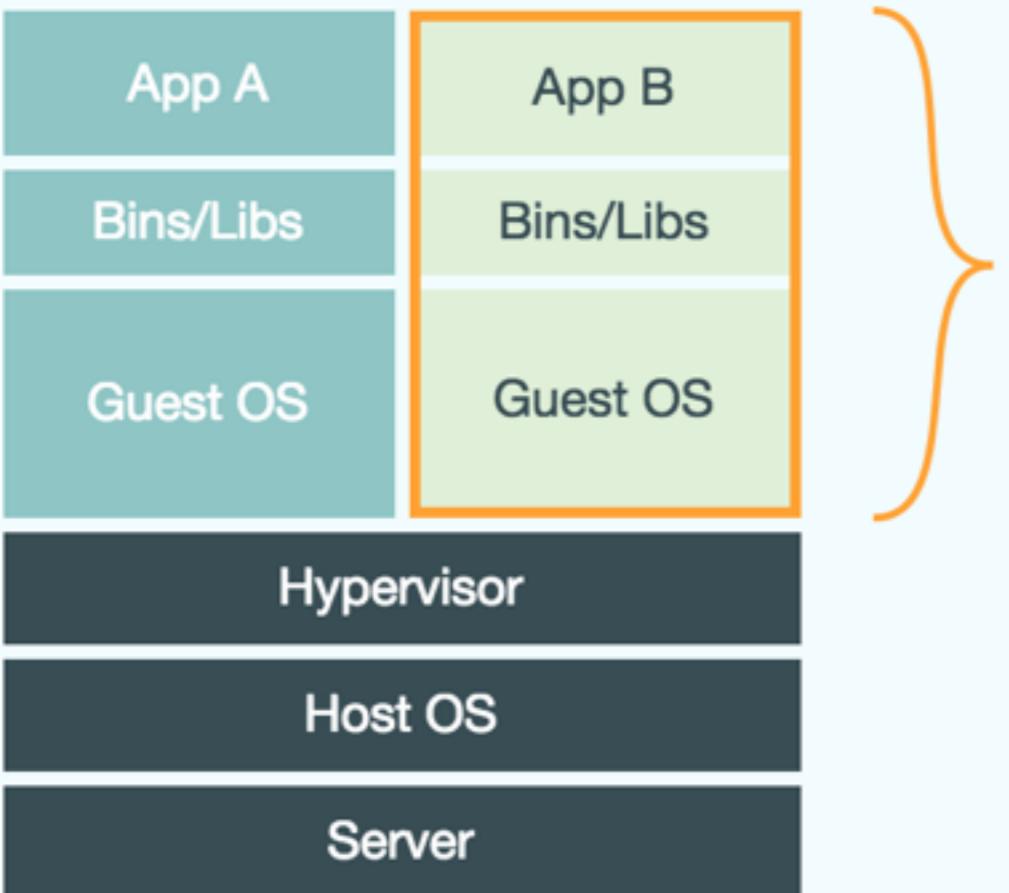
- Natively supported in Linux
- Can be installed on Mac or Windows using `boot2docker`



# Is it only Linux?

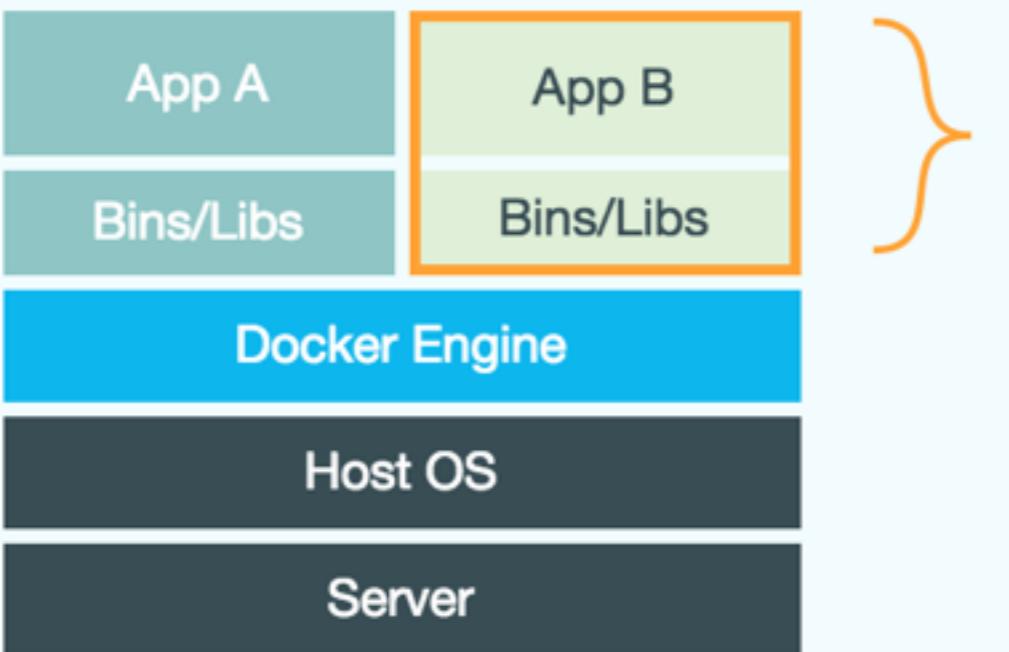
- Natively supported in Linux
- Can be installed on Mac or Windows using `boot2docker`
  - Tiny Core Linux VM





## Virtual Machines

Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.



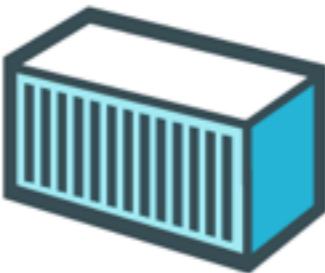
## Docker

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.



## Build

Develop an app using Docker containers with  
any language and any toolchain.



## Ship

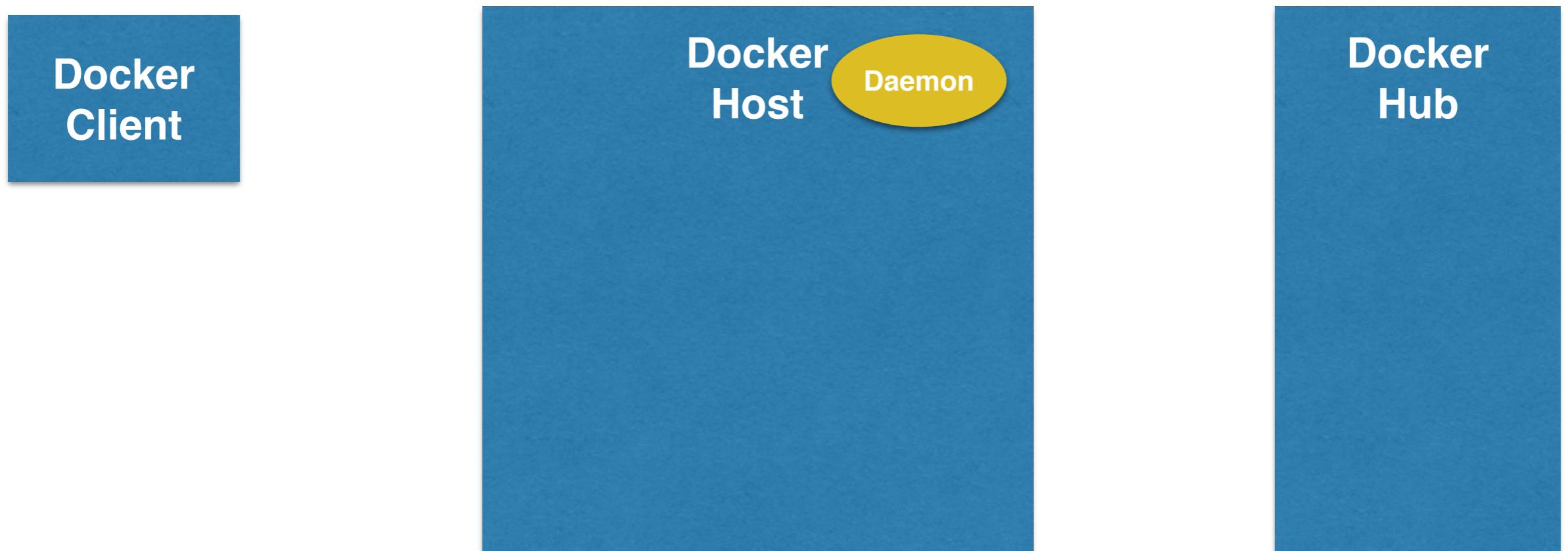
Ship the “Dockerized” app and dependencies  
anywhere - to QA, teammates, or the cloud -  
without breaking anything.



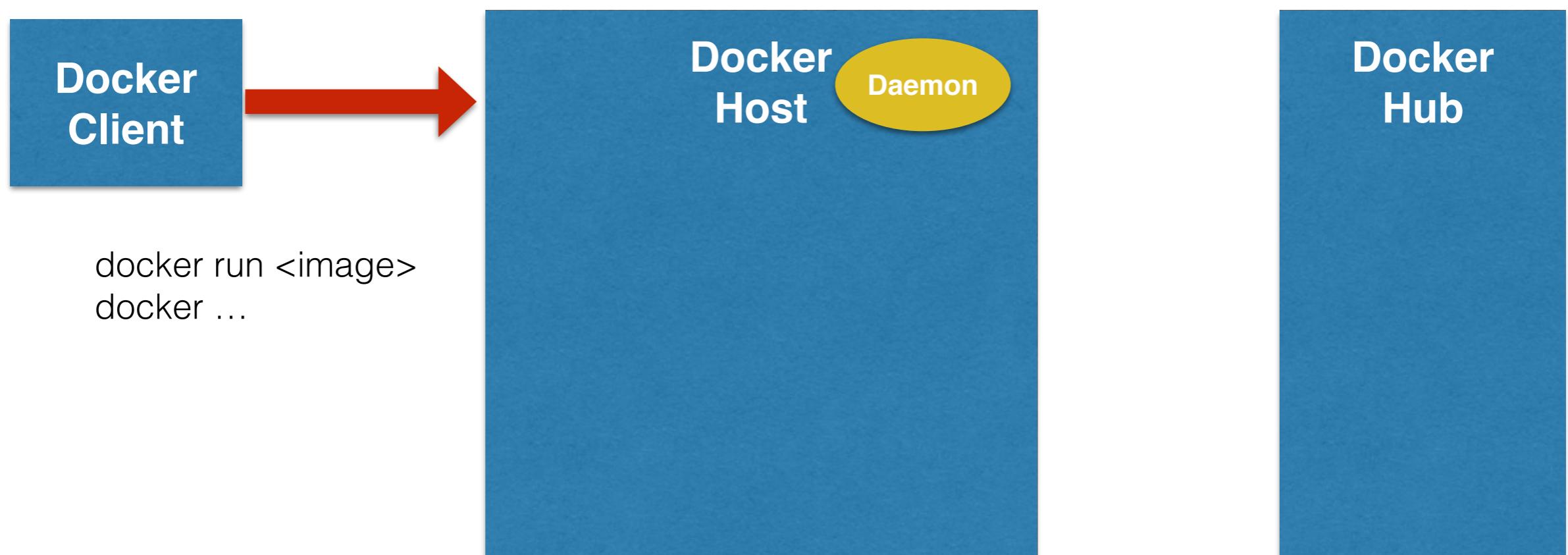
## Run

Scale to 1000s of nodes, move between data  
centers and clouds, update with zero  
downtime and more.

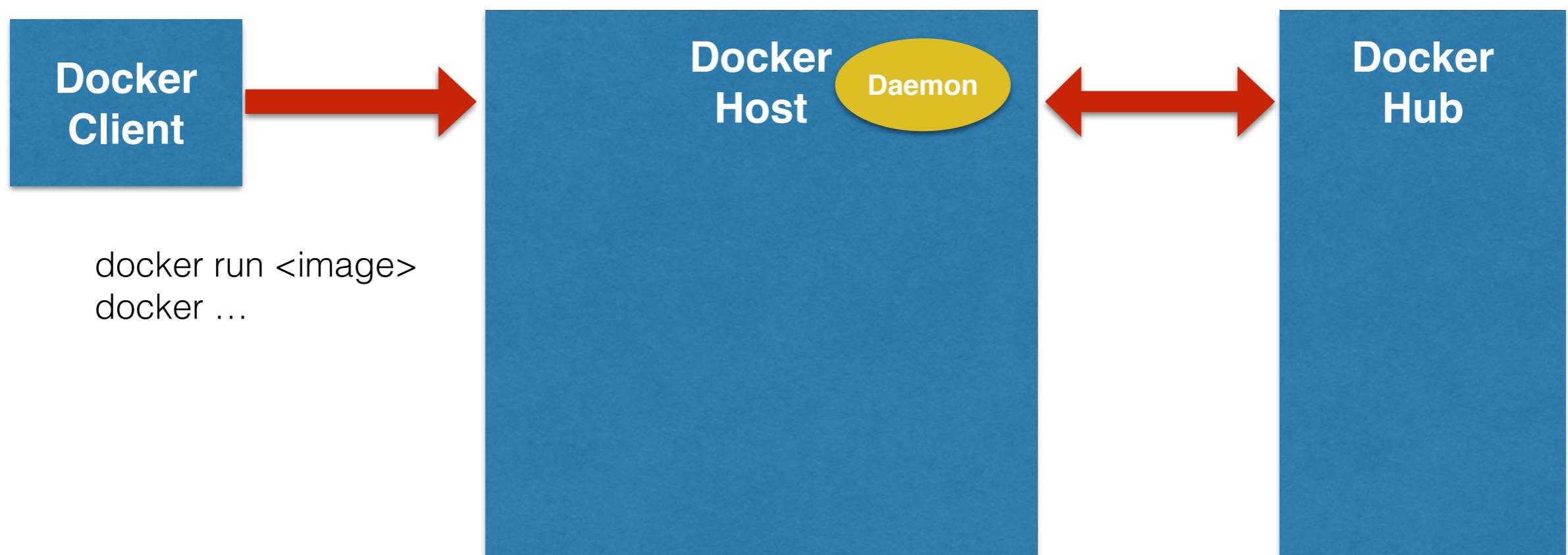
# Docker Workflow



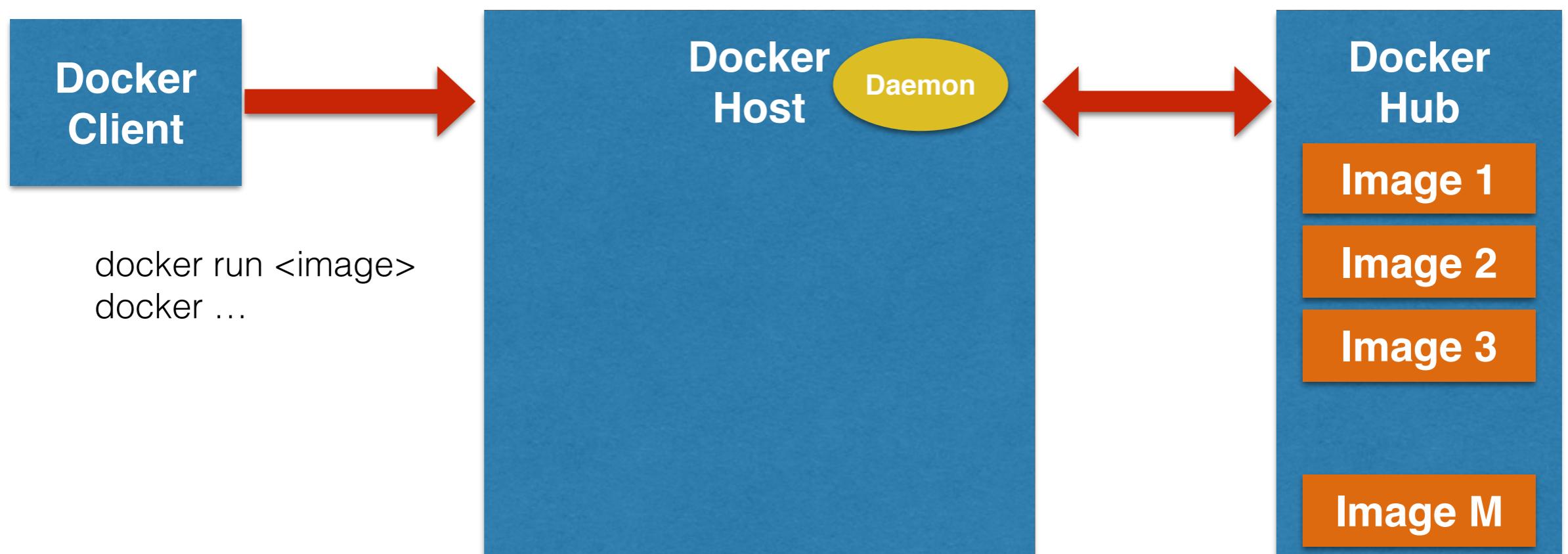
# Docker Workflow



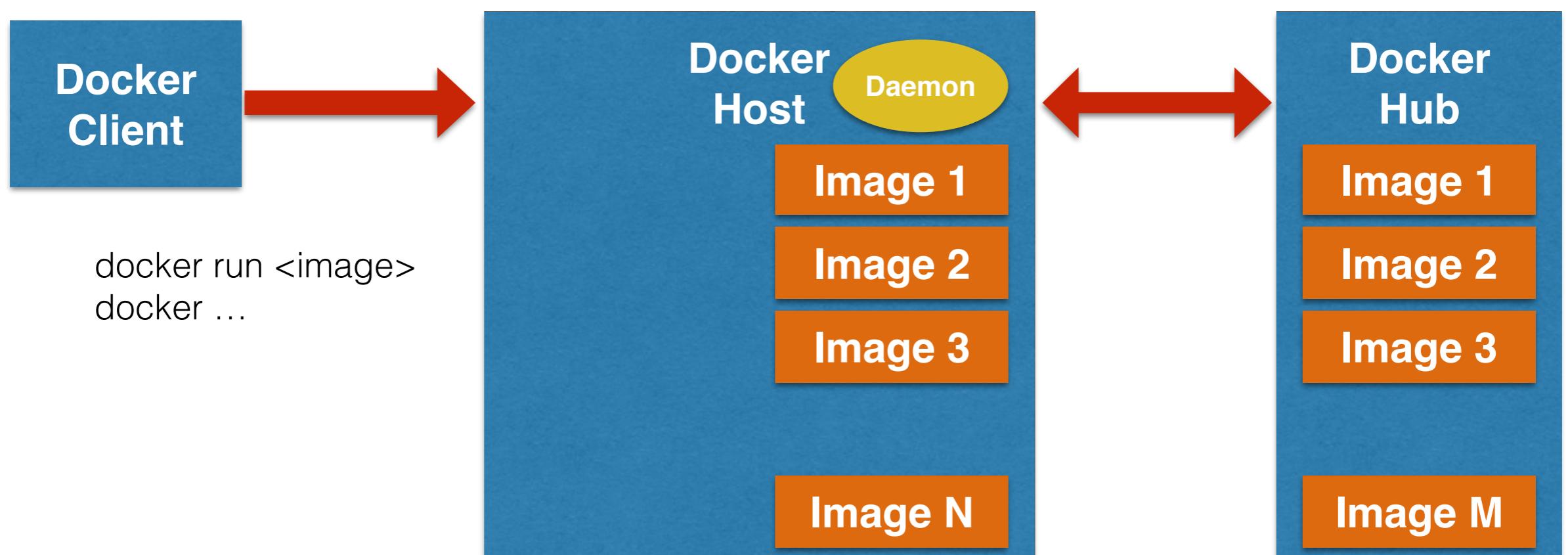
# Docker Workflow



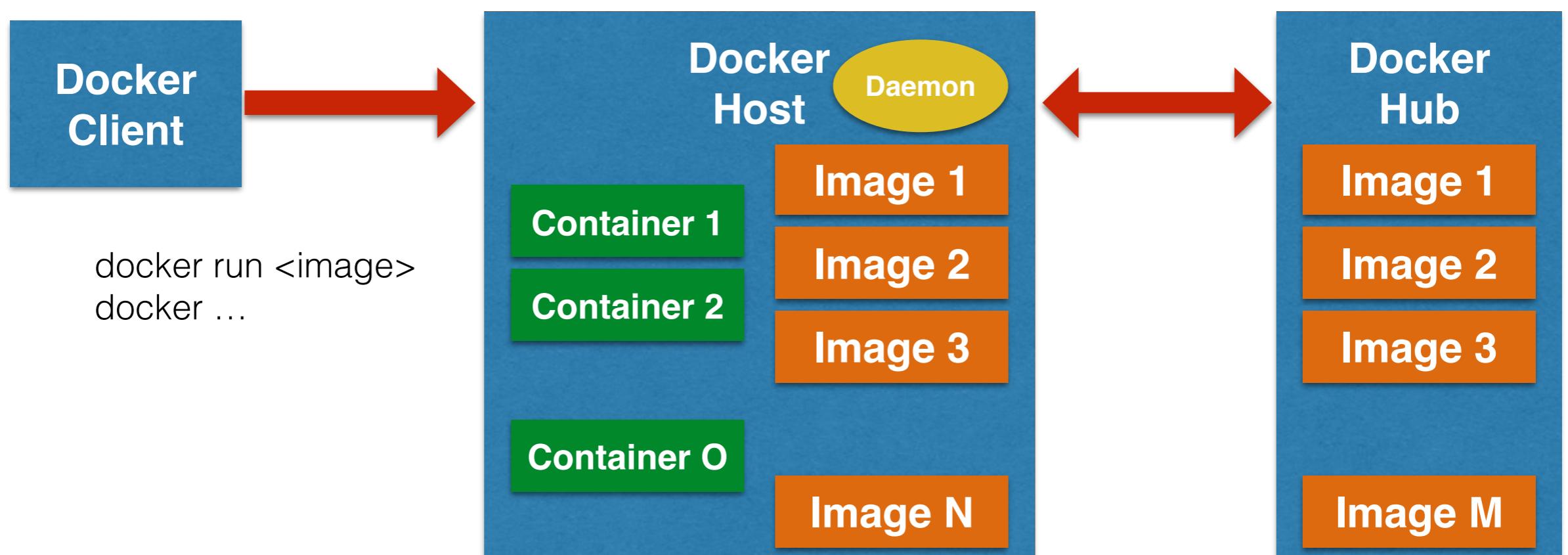
# Docker Workflow



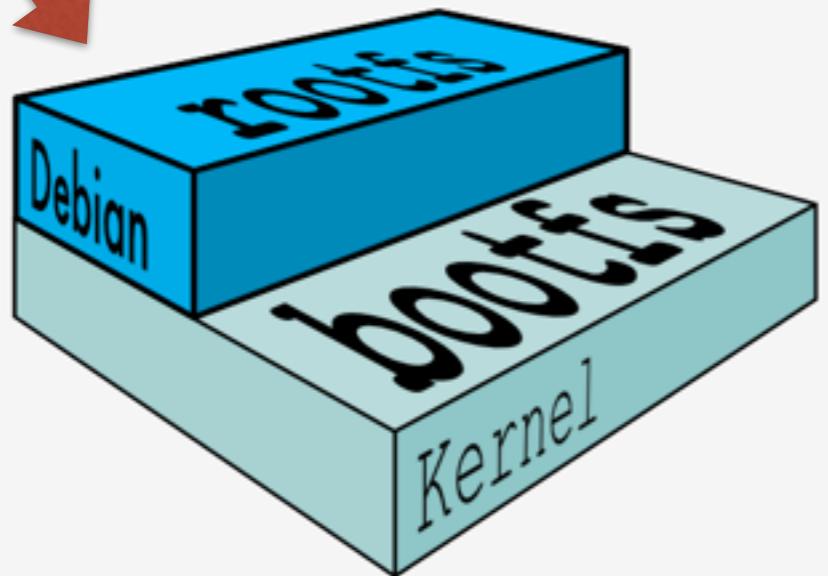
# Docker Workflow



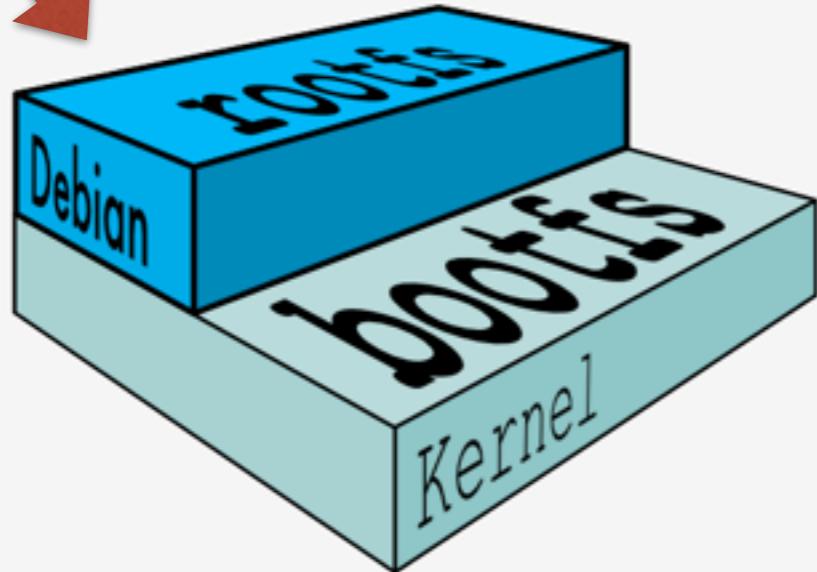
# Docker Workflow



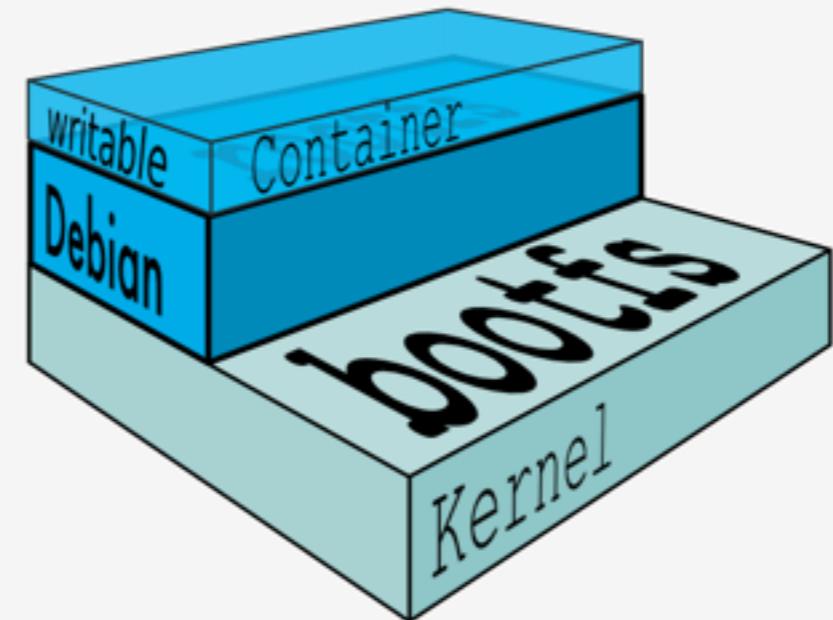
read-only



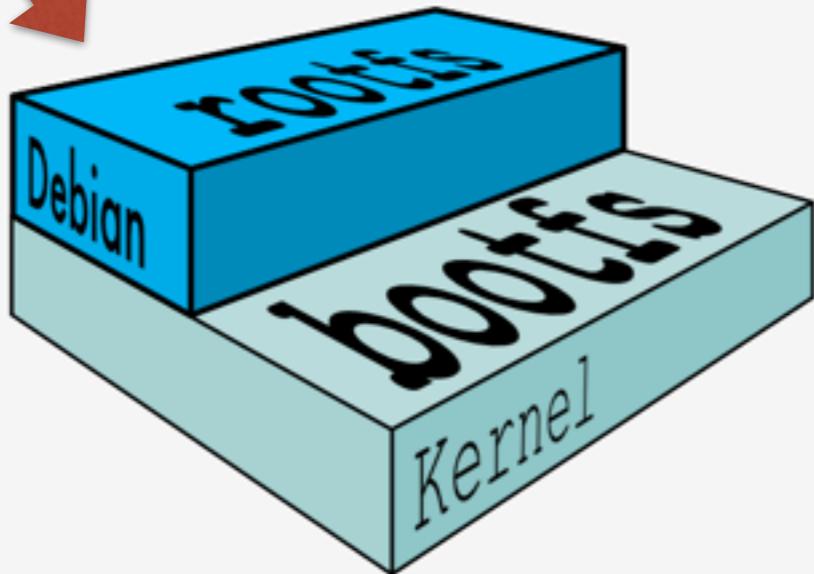
read-only



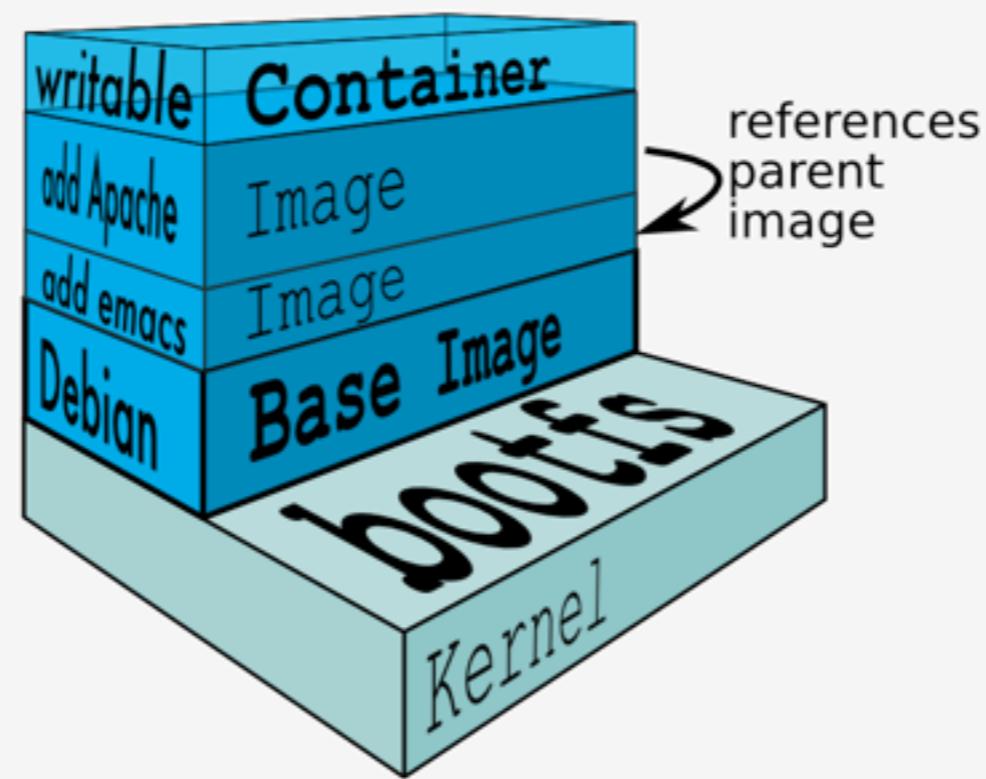
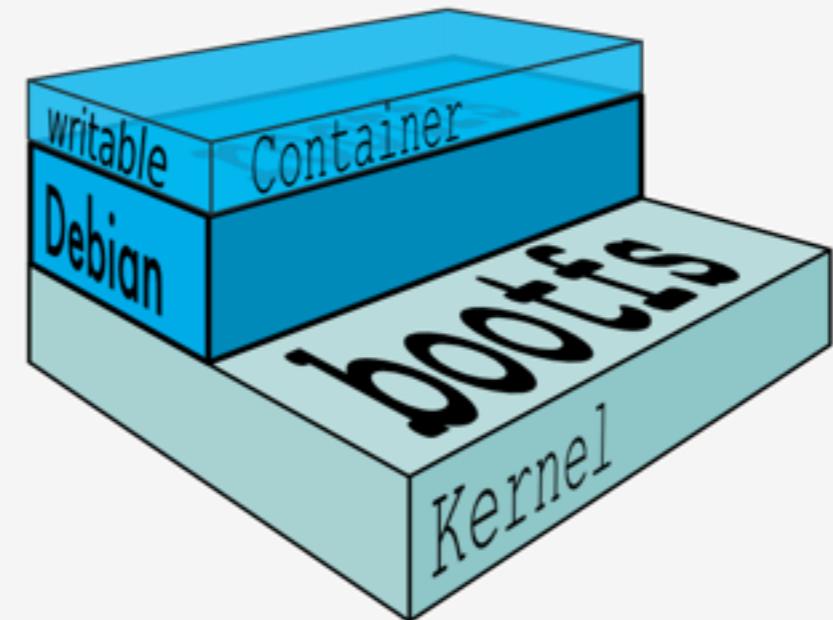
## Union File System



read-only



## Union File System





## Build

Develop an app using Docker containers with  
any language and any toolchain.



- Image defined in text-based **Dockerfile**

Develop an app using Docker containers with  
any language and any toolchain.



Develop an app using Docker containers with  
any language and any toolchain.

- Image defined in text-based **Dockerfile**
- List of commands to build the image



Develop an app using Docker containers with  
any language and any toolchain.

- Image defined in text-based **Dockerfile**
- List of commands to build the image

```
FROM fedora:latest

CMD echo "Hello world"
```



Build

Develop an app using Docker containers with  
any language and any toolchain.

- Image defined in text-based **Dockerfile**
- List of commands to build the image

```
FROM fedora:latest

CMD echo "Hello world"
```

```
FROM jboss/wildfly
MAINTAINER Arun Gupta <arungupta@redhat.com>

RUN curl -L https://github.com/javaee-samples/javaee7-hol/raw/master/solution/
movieplex7-1.0-SNAPSHOT.war -o /opt/jboss/wildfly/standalone/deployments/
movieplex7-1.0-SNAPSHOT.war
```



Build

Develop an app using Docker containers with  
any language and any toolchain.

- Image defined in text-based **Dockerfile**
- List of commands to build the image
- `docker build` or `pull`

```
FROM fedora:latest
```

```
CMD echo "Hello world"
```

```
FROM jboss/wildfly
MAINTAINER Arun Gupta <arungupta@redhat.com>
```

```
RUN curl -L https://github.com/javaee-samples/javaee7-hol/raw/master/solution/
movieplex7-1.0-SNAPSHOT.war -o /opt/jboss/wildfly/standalone/deployments/
movieplex7-1.0-SNAPSHOT.war
```

```
1 FROM centos
2 MAINTAINER Arun Gupta <arungupta@redhat.com>
3
4 # Execute system update
5 RUN yum -y update && yum clean all
6
7 # Install packages necessary to run EAP
8 RUN yum -y install xmlstarlet saxon augeas bsdtar unzip && yum clean all
9
10 # Create a user and group used to launch processes
11 # The user ID 1000 is the default for the first "regular" user on Fedora/RHEL,
12 # so there is a high chance that this ID will be equal to the current user
13 # making it easier to use volumes (no permission issues)
14 RUN groupadd -r jboss -g 1000 && useradd -u 1000 -r -g jboss -m -d /opt/jboss -s /sbin/nologin -c "JBoss user" jboss
15
16 # Set the working directory to jboss' user home directory
17 WORKDIR /opt/jboss
18
19 # User root user to install software
20 USER root
21
22 # Install necessary packages
23 RUN yum -y install java-1.7.0-openjdk-devel && yum clean all
24 #RUN yum -y install java-1.8.0-openjdk-devel && yum clean all
25
26 # Switch back to jboss user
27 USER jboss
28
29 # Set the JAVA_HOME variable to make it clear where Java is located
30 ENV JAVA_HOME /usr/lib/jvm/java
31
32 # Set the WILDFLY_VERSION env variable
33 ENV WILDFLY_VERSION 8.2.0.Final
34
35 # Add the WildFly distribution to /opt, and make wildfly the owner of the extracted tar content
36 # Make sure the distribution is available from a well-known place
37 RUN cd $HOME && curl -O http://download.jboss.org/wildfly/$WILDFLY_VERSION/wildfly-$WILDFLY_VERSION.zip && unzip wildfly-$WILDFLY_
38
39 # Set the JBOSS_HOME env variable
40 ENV JBOSS_HOME /opt/jboss/wildfly
41
42 # Expose the ports we're interested in
43 EXPOSE 8080 9990
44
45 # Set the default command to run on boot
46 # This will boot WildFly in the standalone mode and bind to all interface
47 CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-c", "standalone-full.xml", "-b", "0.0.0.0"]
```



[What is Docker?](#)[Use Cases](#)[Try It!](#)[Browse](#)[Install & Docs](#)[Log In](#)[Sign Up](#)

jboss



Show:

All

Sort by:

Last Updated

Filter by name...

Repositories

**jboss/forge**

8 hours ago



1



17

**jboss/liveoak-server**

4 days ago



3



140

**jboss/switchyard-wildfly**

a month ago



0



49

**jboss/keycloak**

a month ago



0



540

**jboss/keycloak-adapter-wildfly**

a month ago



0



195

**jboss/infinispan-server**

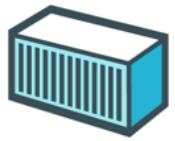
2 months ago



0



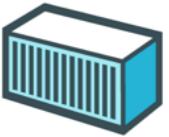
87



## Ship

Ship the “Dockerized” app and dependencies anywhere - to QA, teammates, or the cloud - without breaking anything.

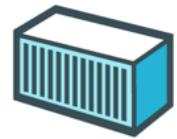
- Images shared using registry



Ship

Ship the “Dockerized” app and dependencies anywhere - to QA, teammates, or the cloud - without breaking anything.

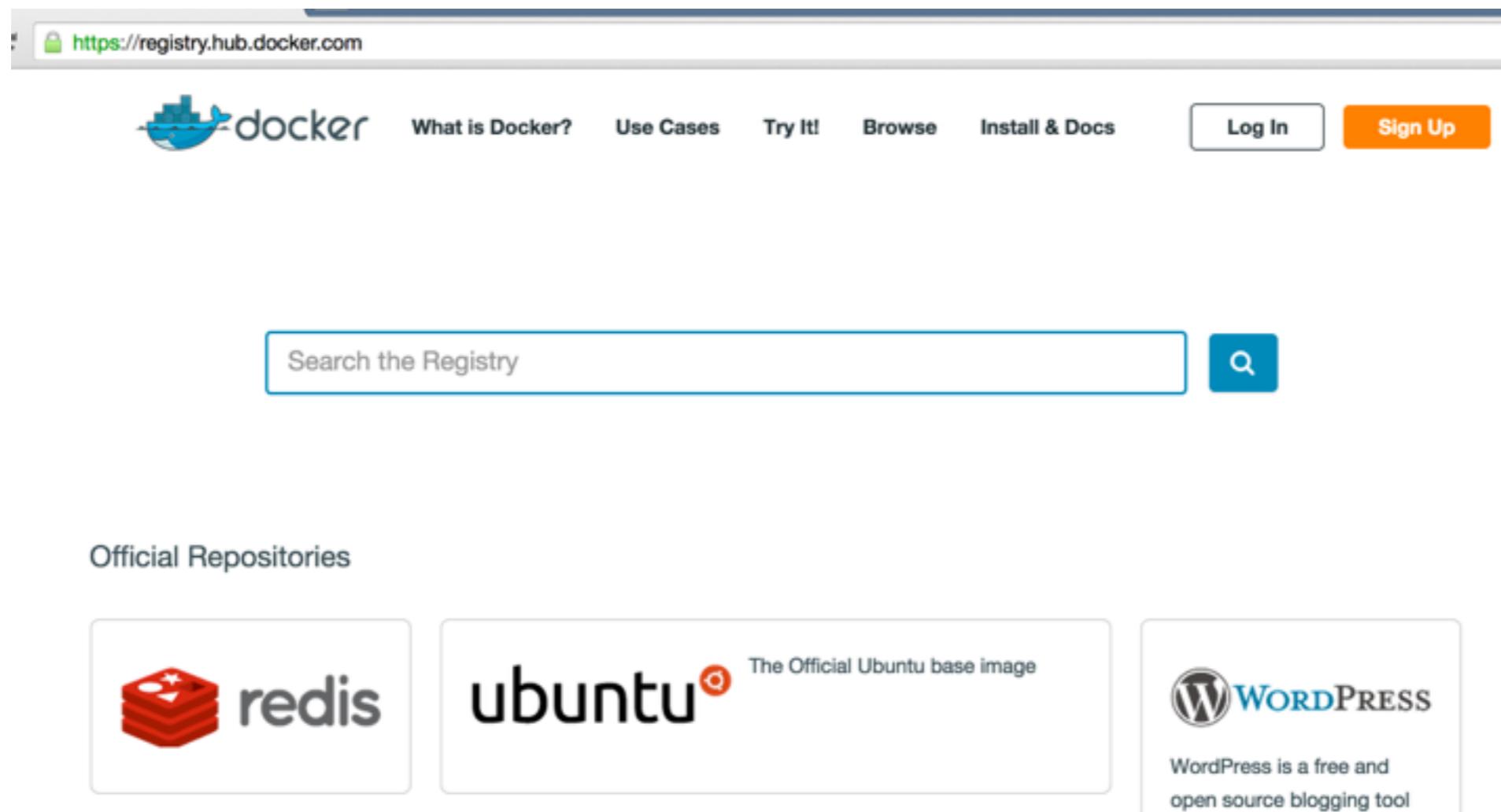
- Images shared using registry



Ship

- Docker Hub is public SaaS

Ship the “Dockerized” app and dependencies anywhere - to QA, teammates, or the cloud - without breaking anything.

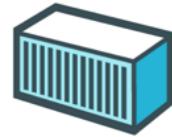


The screenshot shows the Docker Hub Registry homepage. At the top, there's a navigation bar with links for "What is Docker?", "Use Cases", "Try It!", "Browse", "Install & Docs", "Log In", and "Sign Up". Below the navigation is a search bar with the placeholder "Search the Registry" and a magnifying glass icon. The main content area is titled "Official Repositories" and features three cards:

- redis**: Represented by a red cube icon.
- ubuntu**: Represented by the word "ubuntu" in a large, bold font with a small orange circle containing a white dot.
- WORDPRESS**: Represented by the WordPress logo (a stylized 'W' inside a circle).

Below each repository card, there is a brief description: "The Official Ubuntu base image" for Ubuntu and "WordPress is a free and open source blogging tool" for WordPress.

- Images shared using registry
- Docker Hub is public SaaS
- Private registries can be setup inside firewall



Ship

Ship the “Dockerized” app and dependencies anywhere - to QA, teammates, or the cloud - without breaking anything.

The screenshot shows the Docker Hub homepage at <https://registry.hub.docker.com>. The top navigation bar includes links for "What is Docker?", "Use Cases", "Try It!", "Browse", "Install & Docs", "Log In", and "Sign Up". A search bar at the top right contains the placeholder "Search the Registry" and a magnifying glass icon. Below the search bar, the heading "Official Repositories" is displayed. Three repository cards are visible: "redis" (with a red cube icon), "ubuntu" (with a black "ubuntu" logo and the text "The Official Ubuntu base image"), and "WORDPRESS" (with a white "W" icon and the text "WordPress is a free and open source blogging tool").

- Images shared using registry
- Docker Hub is public SaaS
- Private registries can be setup inside firewall
- `docker push or pull <IMAGE_ID>`



Ship the “Dockerized” app and dependencies anywhere - to QA, teammates, or the cloud - without breaking anything.

The screenshot shows the Docker Hub homepage. At the top, there's a navigation bar with links for "What is Docker?", "Use Cases", "Try It!", "Browse", "Install & Docs", "Log In", and "Sign Up". Below the navigation is a search bar with the placeholder "Search the Registry" and a magnifying glass icon. The main content area is titled "Official Repositories" and features three cards: "redis" (with a red cube icon), "ubuntu" (with the word "ubuntu" and "The Official Ubuntu base image"), and "WORDPRESS" (with the WordPress logo and the text "WordPress is a free and open source blogging tool").



Run

Scale to 1000s of nodes, move between data  
centers and clouds, update with zero  
downtime and more.

- Container built from image



Run

Scale to 1000s of nodes, move between data centers and clouds, update with zero downtime and more.

- Container built from image
- Runtime representation of image



Run

Scale to 1000s of nodes, move between data centers and clouds, update with zero downtime and more.



- Container built from image
- Runtime representation of image
- Self contained execution environment

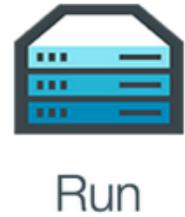


Run

Scale to 1000s of nodes, move between data centers and clouds, update with zero downtime and more.



- Container built from image
- Runtime representation of image
- Self contained execution environment
- `docker run <IMAGE_ID>`



Run

Scale to 1000s of nodes, move between data centers and clouds, update with zero downtime and more.



# Docker commands

- **docker ps**: List running containers
- **docker stop**: Stop a running container
- **docker rm**: Remove a running container
- **docker rmi**: Remove an image
- ...

<https://docs.docker.com/reference/commandline/cli/>

# DOCKER ECOSYSTEM

## YEAR IN REVIEW 2014



\$15  
million

JAN 21  
DOCKER CLOSES  
\$15M FUNDING

Series B round of funding led by  
Greylock Partners

v0.9

MAR 10  
DOCKER 0.9

Drops LXC and replaces it with  
libcontainer library written in Go



MAR 15  
WAVEMAKER PREVIEWS  
DOCKER-ARCHITECTED CLOUD

Developer preview of Wavemaker Cloud.  
Provisions 5k+ instances to 500+ developers  
in over 50 countries

1st  
birthday

MAR 20  
DOCKER TURNS ONE

370 contributors,  
1 million downloads,  
27 countries



dockercon14  
June 9-10, 2014 • San Francisco



GOOGLE ANNOUNCES  
KUBERNETES

An open source orchestration system for  
Docker containers. Receives support from  
Microsoft, IBM, Docker, CoreOS and others



DOCKER 1.0 AND  
ENTERPRISE SUPPORT

Availability of commercial support,  
integration with Docker Hub



ADDITIONAL  
LAUNCHES

libnetwork: network services toolkit  
libcontainer: ultra-lightweight networking library  
libcontainer: standard interface to Linux D-Bus  
sandboxing, made a standalone project



APR 23  
AWS LAUNCHES  
SUPPORT FOR DOCKER

AWS Elastic Beanstalk supports running  
and managing dockerized applications



APR 15  
RED HAT FAST-TRACKS  
DOCKER FOR ENTERPRISE

Integrates Docker and Linux container  
elements in Red Hat Enterprise Linux 7



JUL 23  
DOCKER ACQUIRES  
ORCHARD LABS

Developers of Fig, a composition and  
orchestration tool for multi-container  
Docker apps



AUG 4  
DOCKER SELLS  
DOTCLOUD PLATFORM

Sells dotCloud PaaS to CloudControl to  
focus on the Docker project



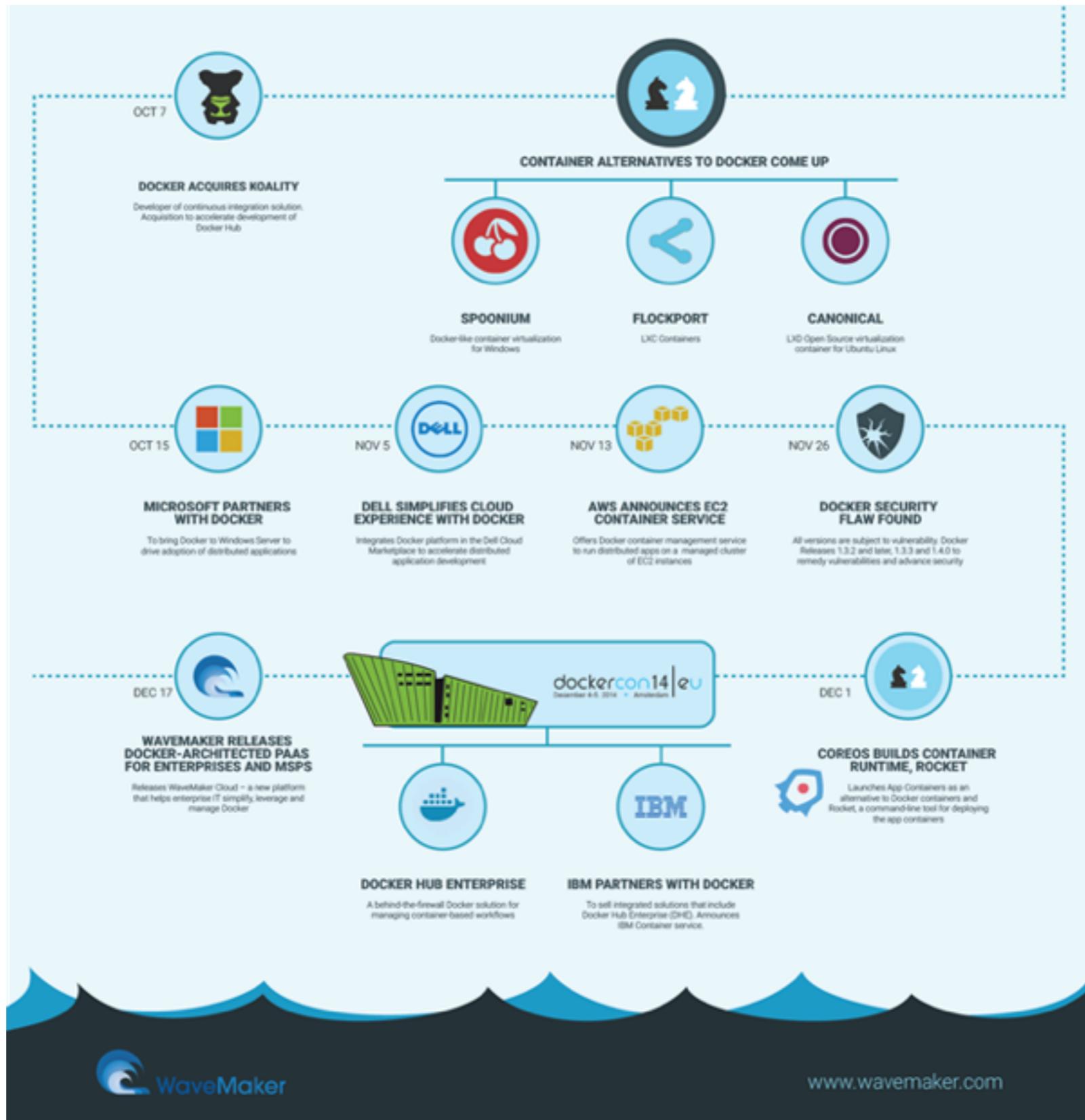
AUG 25  
VMWARE TEAMS  
WITH DOCKER

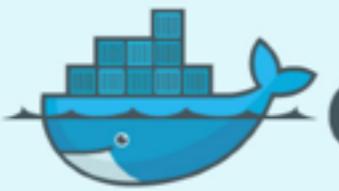
Partners with Docker to help enterprises run  
and manage container-based apps



SEP 16  
DOCKER RAISES  
\$40 MILLION FUNDING

Series C round of funding led by  
Sequoia, with a 3x increase in round size





# docker



## Docker Project: 2014 Year in Review

102.5 Million

Docker Container Downloads

DEC

↑18.8K%  
GROWTH

66.3 Million

NOV

32 Million

OCT

46.8 Million

SEP

18.8 Million

AUG

8.5 Million

JUL

4.3 Million

JUN

2.7 Million

MAY

1.7 Million

APR

1.5 Million

MAR

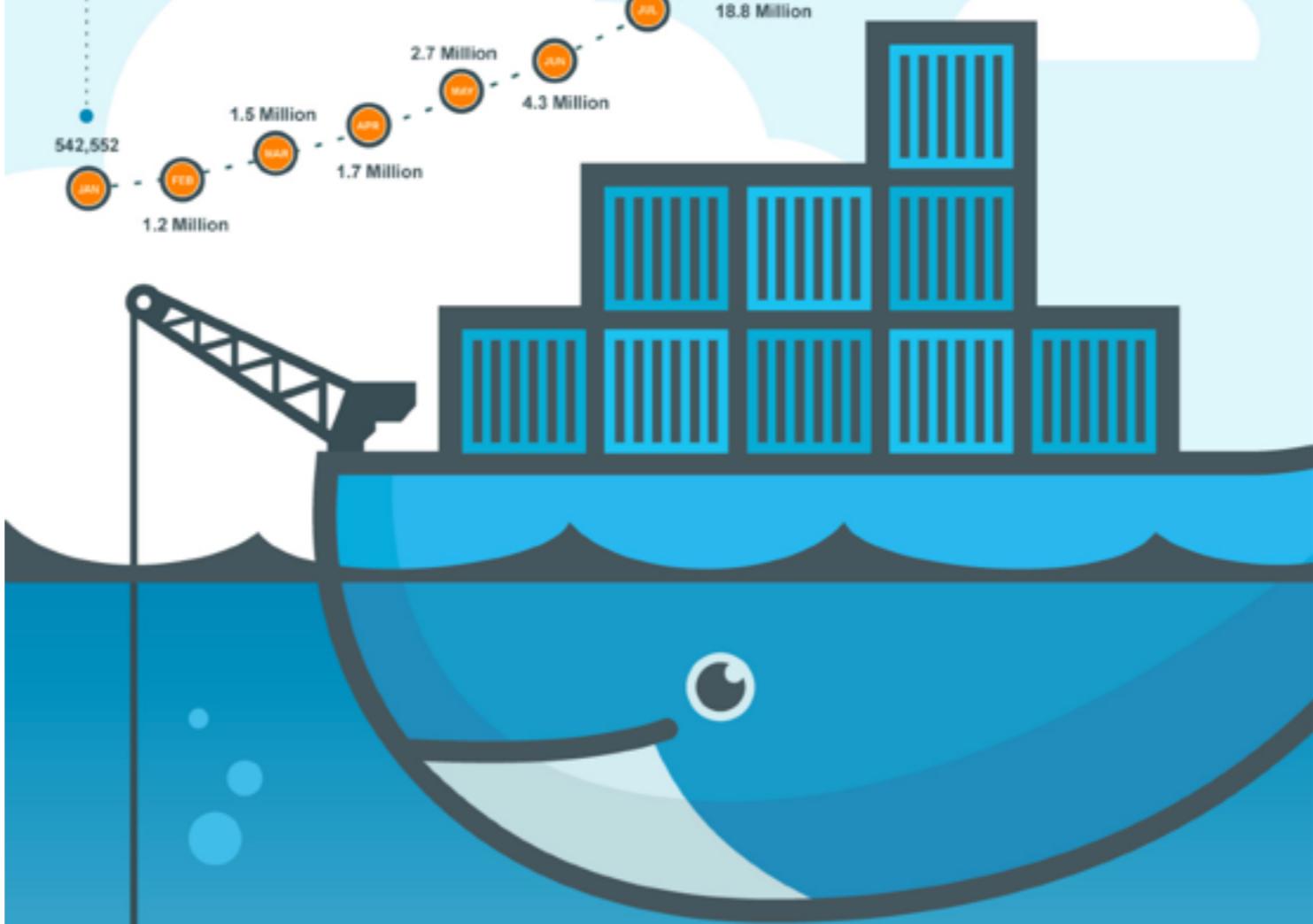
1.2 Million

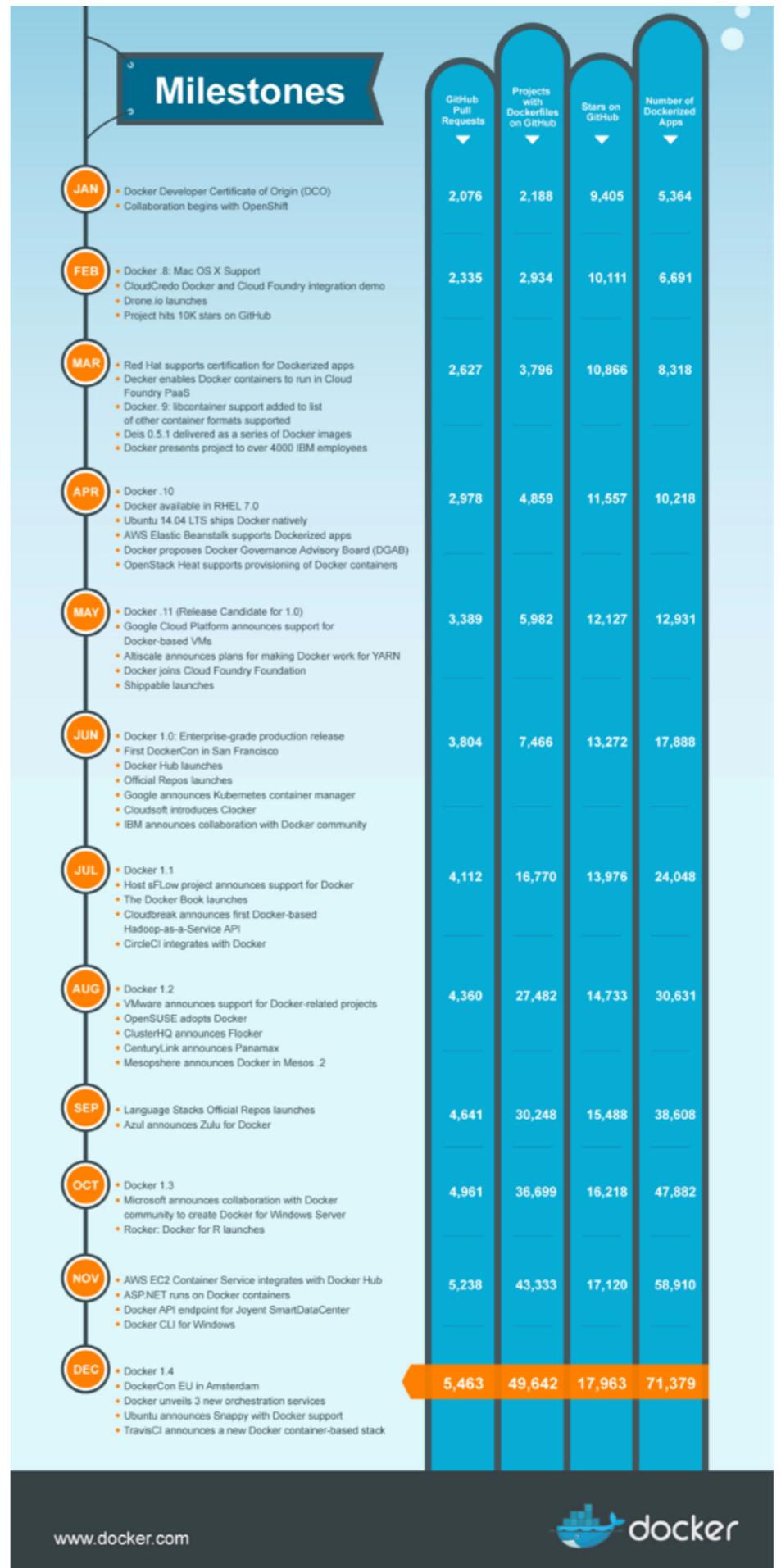
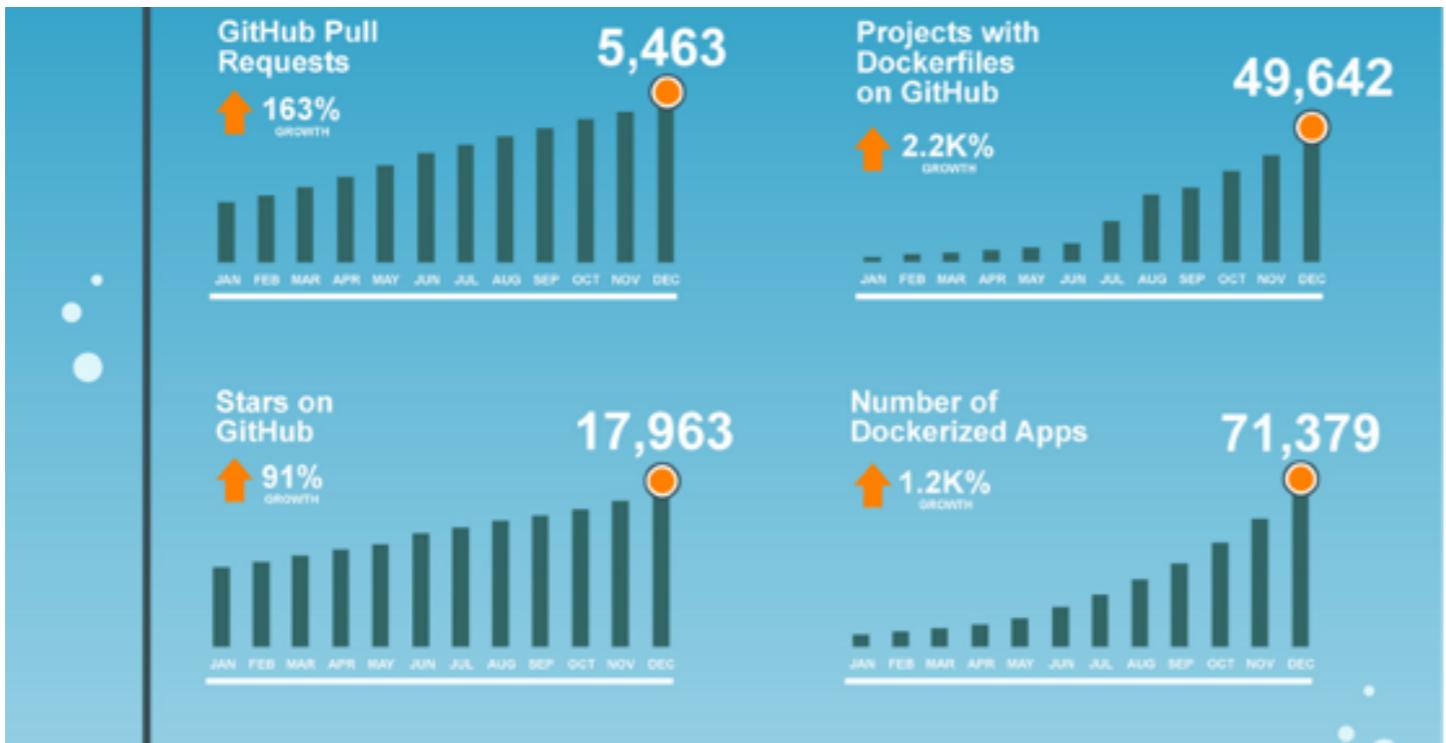
FEB

542,552

JAN

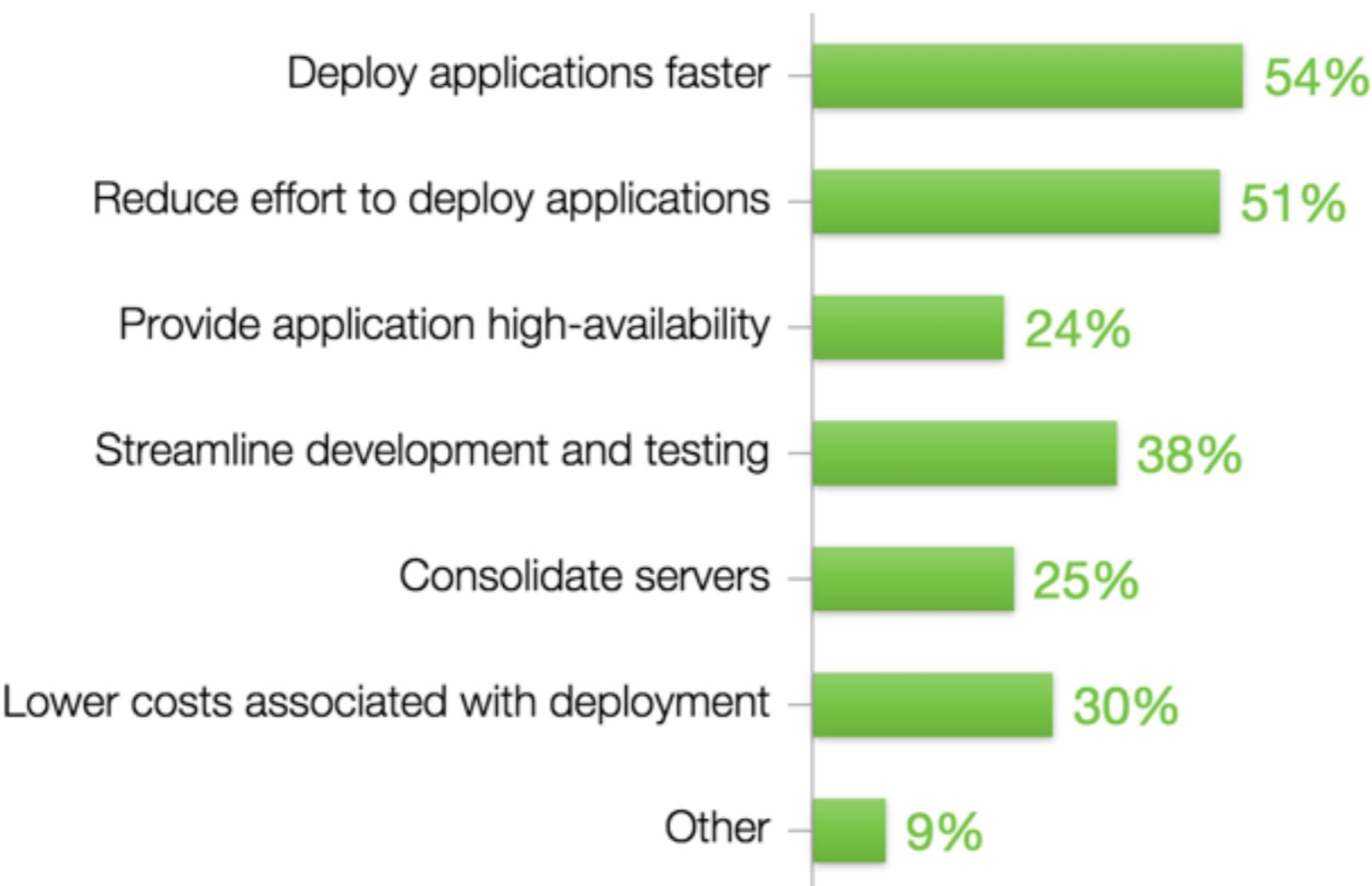
# of Docker Container  
Downloads





# Benefits of containers

What are the top benefits you see with containers?



Note: this is a multiple-choice question – response percentages may not add up to 100.

**Source:** □ TechValidate survey of 79 IT professionals

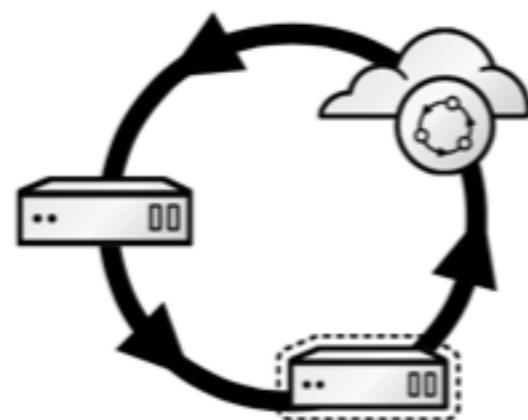
# Benefits of Containers



FASTER APP  
DELIVERY



OPERATIONAL  
EFFICIENCY

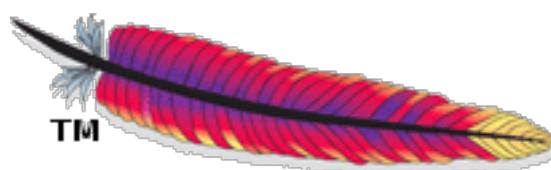


DEPLOYMENT  
FLEXIBILITY



LOWERED  
DEPLOYMENT  
COSTS

# Application Operating Environment



# Kubernetes



# Kubernetes



- Open source orchestration system for Docker containers

# Kubernetes



- Open source orchestration system for Docker containers
- Provide declarative primitives for the “desired state”

# Kubernetes



- Open source orchestration system for Docker containers
- Provide declarative primitives for the “desired state”
  - Self-healing

# Kubernetes



- Open source orchestration system for Docker containers
- Provide declarative primitives for the “desired state”
  - Self-healing
  - Auto-restarting

# Kubernetes



- Open source orchestration system for Docker containers
- Provide declarative primitives for the “desired state”
  - Self-healing
  - Auto-restarting
  - Schedule across hosts

# Kubernetes



- Open source orchestration system for Docker containers
- Provide declarative primitives for the “desired state”
  - Self-healing
  - Auto-restarting
  - Schedule across hosts
  - Replicating

# Key Concepts



# Key Concepts



- **Pods:** Smallest deployable unit that can be created, scheduled, managed (logical collection of containers)

# Key Concepts



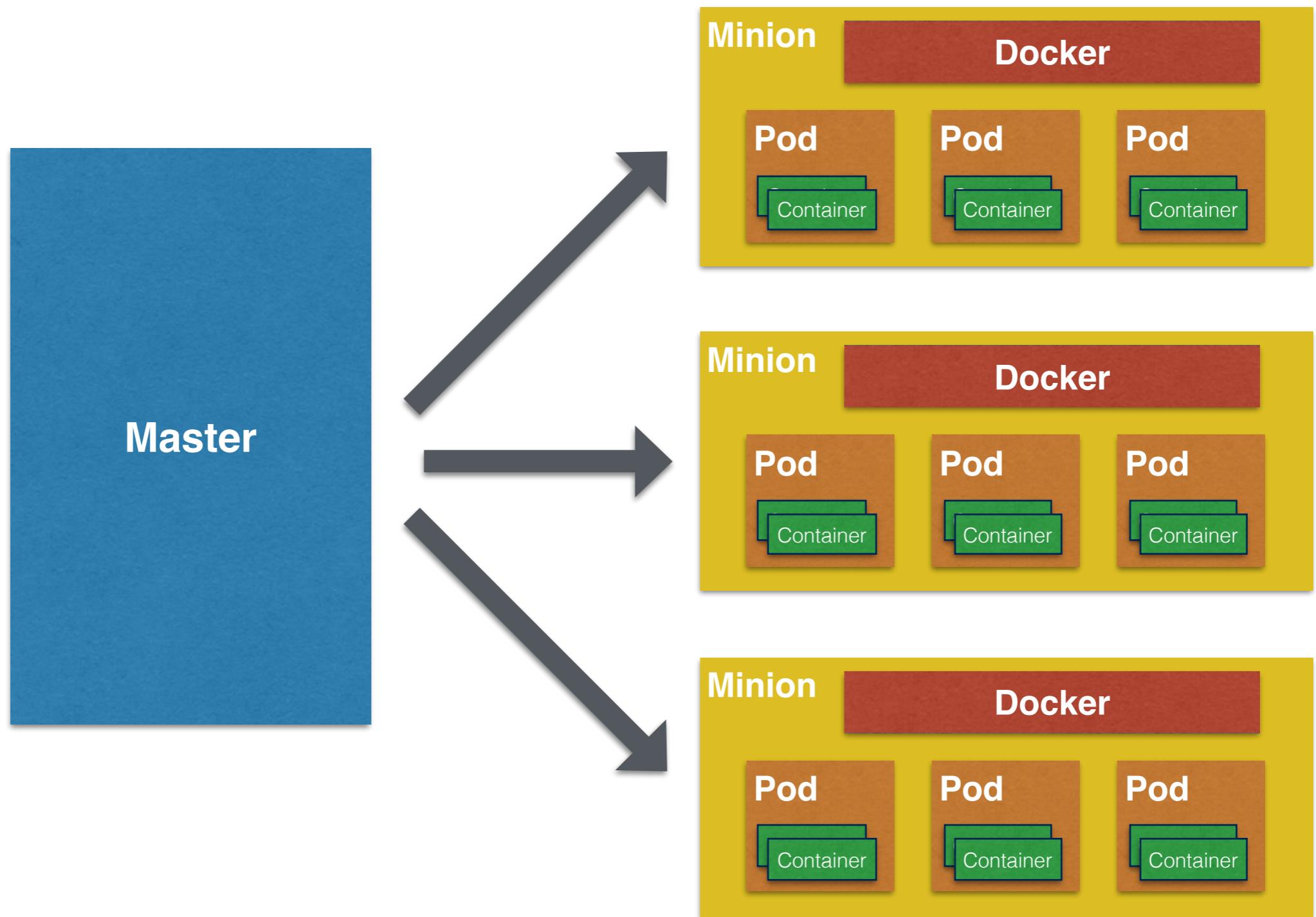
- **Pods:** Smallest deployable unit that can be created, scheduled, managed (logical collection of containers)
- **Master:** Central point that provides unified view of cluster, control one or more minions

# Key Concepts



- **Pods:** Smallest deployable unit that can be created, scheduled, managed (logical collection of containers)
- **Master:** Central point that provides unified view of cluster, control one or more minions
- **Minion:** Run tasks from master

# Kubernetes



# kubectl

# kubectl

- Controls the Kubernetes cluster manager

# kubectl

- Controls the Kubernetes cluster manager
- `kubectl get pods or minions`

# kubectl

- Controls the Kubernetes cluster manager
- `kubectl get pods or minions`
- `kubectl create -f <filename>`

# kubectl

- Controls the Kubernetes cluster manager
- `kubectl get pods or minions`
- `kubectl create -f <filename>`
- `kubectl update or delete`

# kubectl

- Controls the Kubernetes cluster manager
- `kubectl get pods or minions`
- `kubectl create -f <filename>`
- `kubectl update or delete`
- `kubectl resize --replicas=3 replicationcontrollers <name>`

# Kubernetes Config

```
{  
  "id": "mysql",  
  "kind": "Pod",  
  "apiVersion": "v1beta1",  
  "desiredState": {  
    "manifest": {  
      "version": "v1beta1",  
      "id": "mysql",  
      "containers": [ {  
        "name": "mysql",  
        "image": "mysql",  
        "cpu": 100,  
        "ports": [ {  
          "containerPort": 3306,  
          "hostPort": 3306  
        } ]  
      } ]  
    }  
  },  
  "labels": {  
    "name": "mysql"  
  }  
}
```

# Kubernetes Config

```
{  
  "id": "mysql",  
  "kind": "Pod",  
  "apiVersion": "v1beta1",  
  "desiredState": {  
    "manifest": {  
      "version": "v1beta1",  
      "id": "mysql",  
      "containers": [ {  
        "name": "mysql",  
        "image": "mysql",  
        "cpu": 100,  
        "ports": [ {  
          "containerPort": 3306,  
          "hostPort": 3306  
        } ]  
      } ]  
    }  
  },  
  "labels": {  
    "name": "mysql"  
  }  
}  
  
{  
  "id": "wildfly",  
  "kind": "Pod",  
  "apiVersion": "v1beta1",  
  "desiredState": {  
    "manifest": {  
      "version": "v1beta1",  
      "id": "wildfly",  
      "containers": [ {  
        "name": "wildfly",  
        "image": "jboss/wildfly",  
        "cpu": 100,  
        "ports": [ {  
          "containerPort": 8080,  
          "hostPort": 8080  
        } ]  
      } ]  
    }  
  },  
  "labels": {  
    "name": "wildfly"  
  }  
}
```

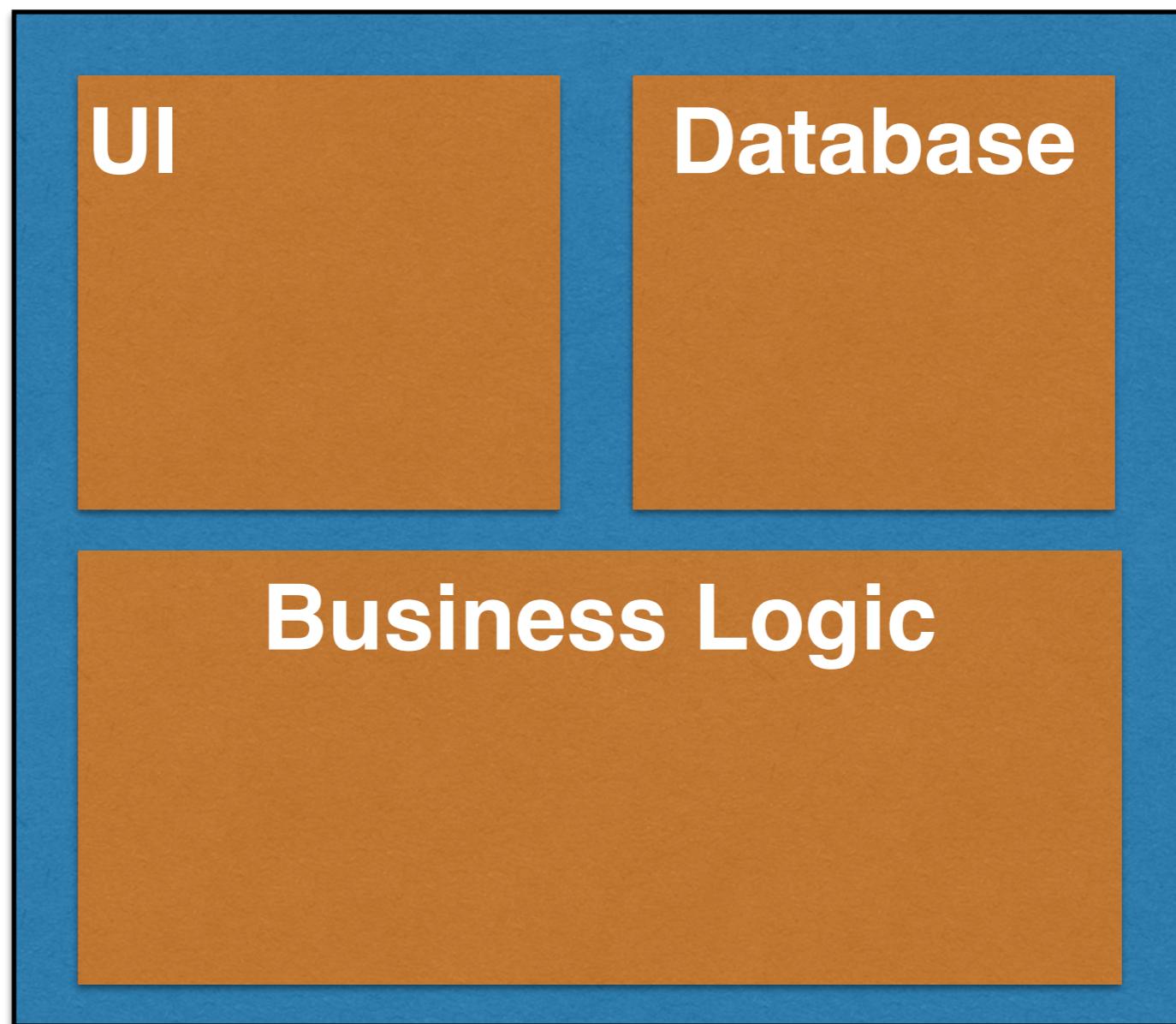
# Microservices

*building applications as **suites of services**. As well as the fact that services are **independently deployable and scalable**, each service also provides a **firm module boundary**, even allowing for different services to be written in **different programming languages**. They can also be **managed by different teams***

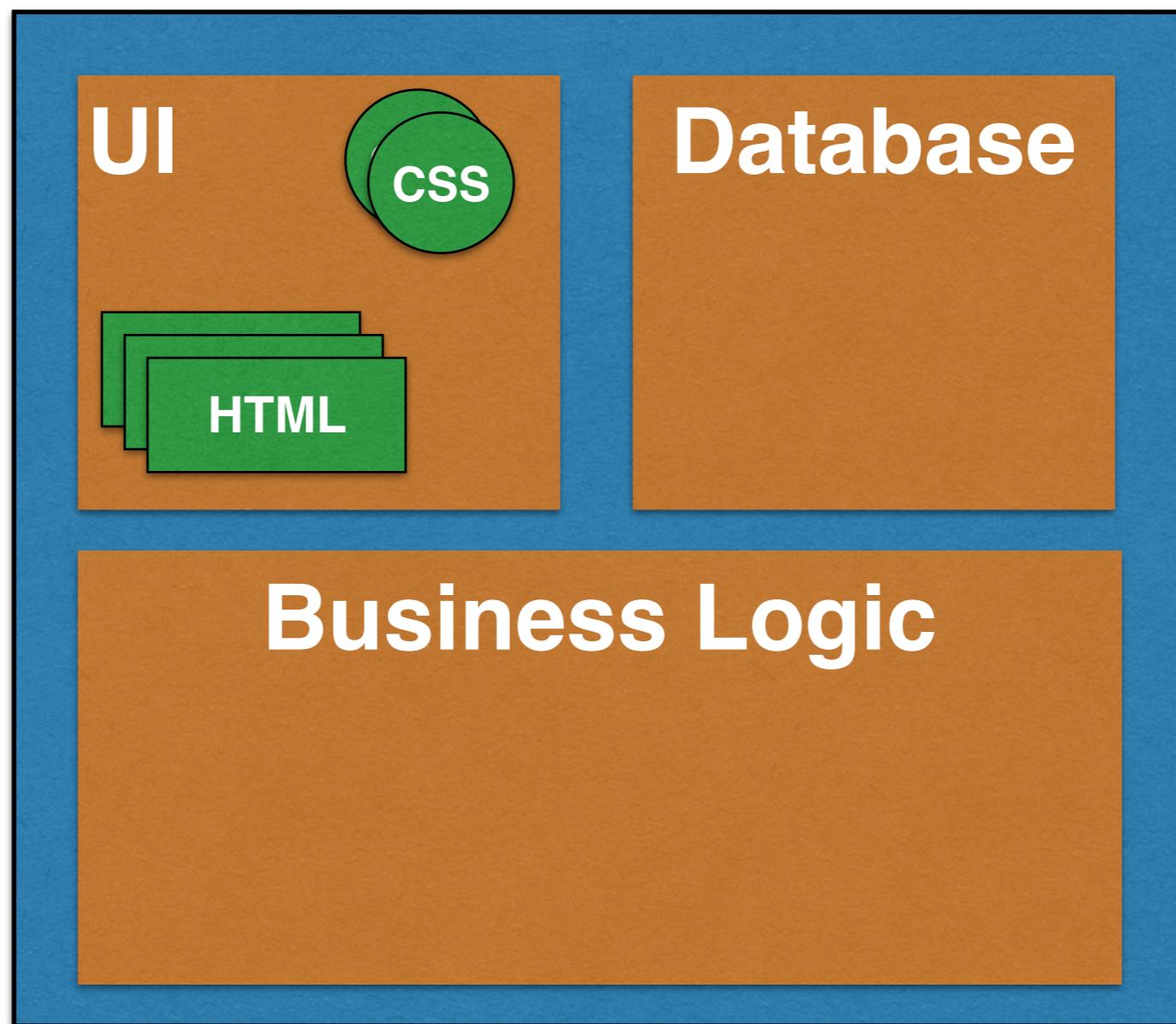
# Is it SOA?

- SOA for hipsters?
- Fine-grained SOA?
- Focus on ESBs in SOA?
- SOA done right?

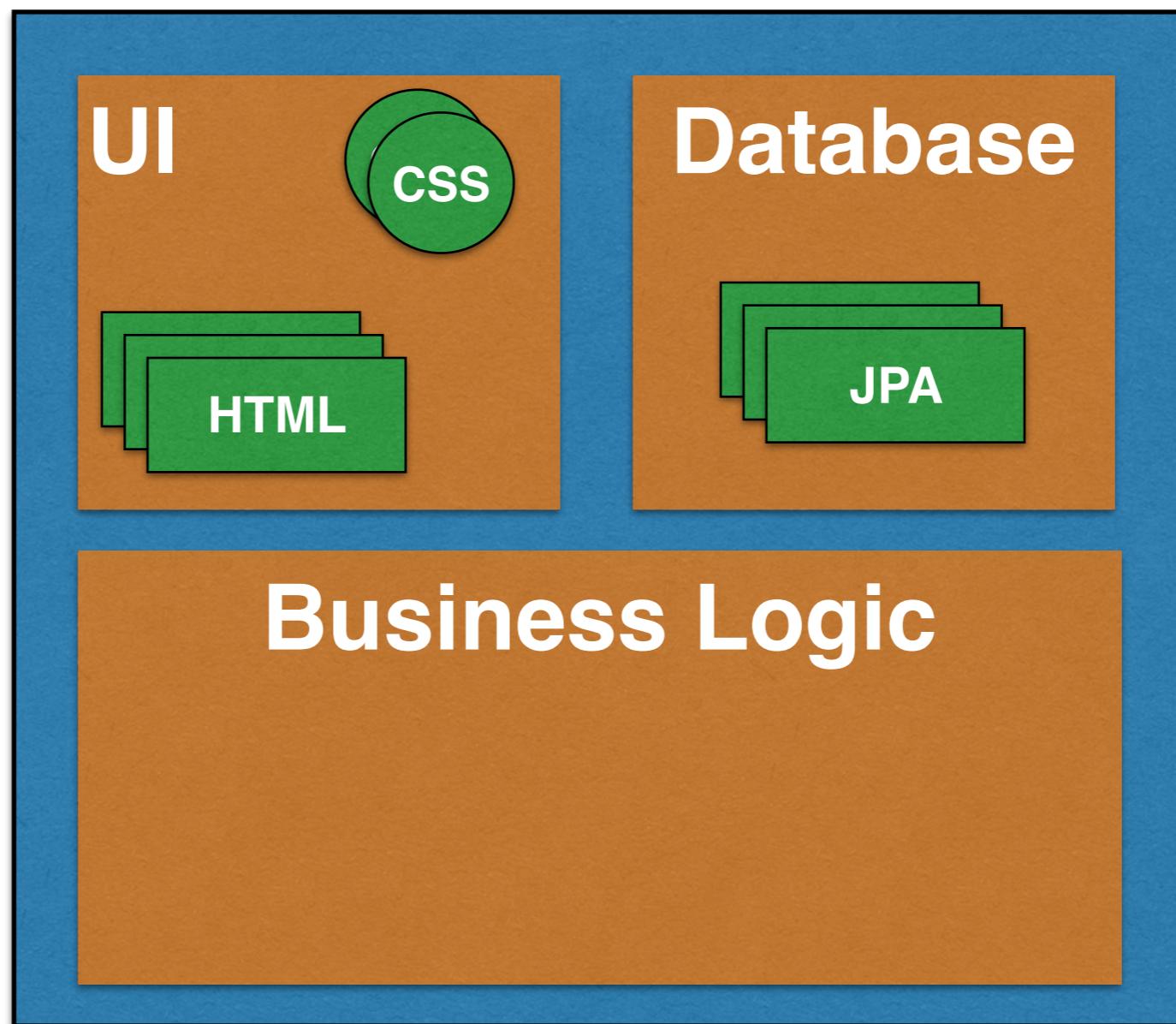
# Monolith Application



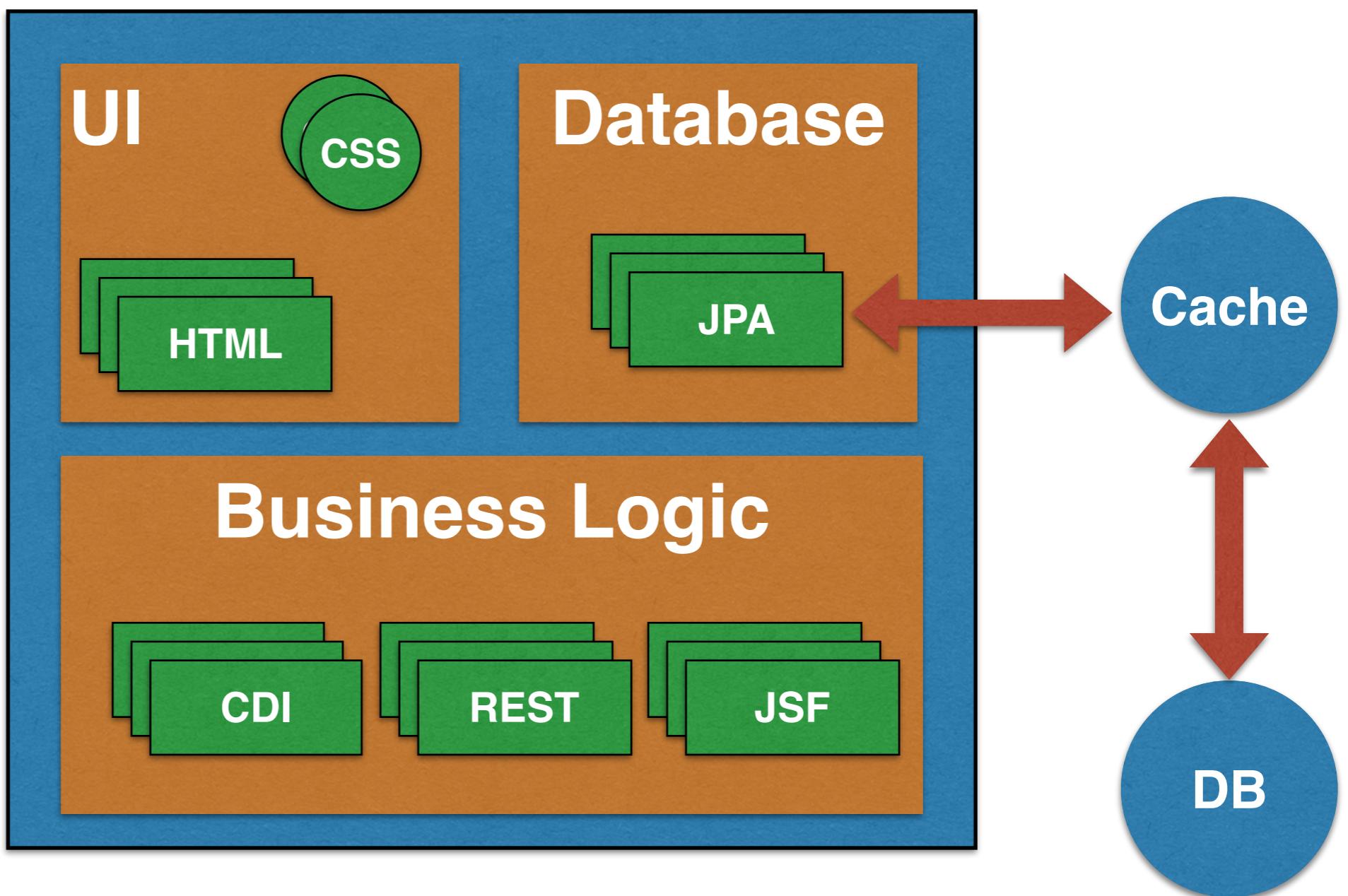
# Monolith Application



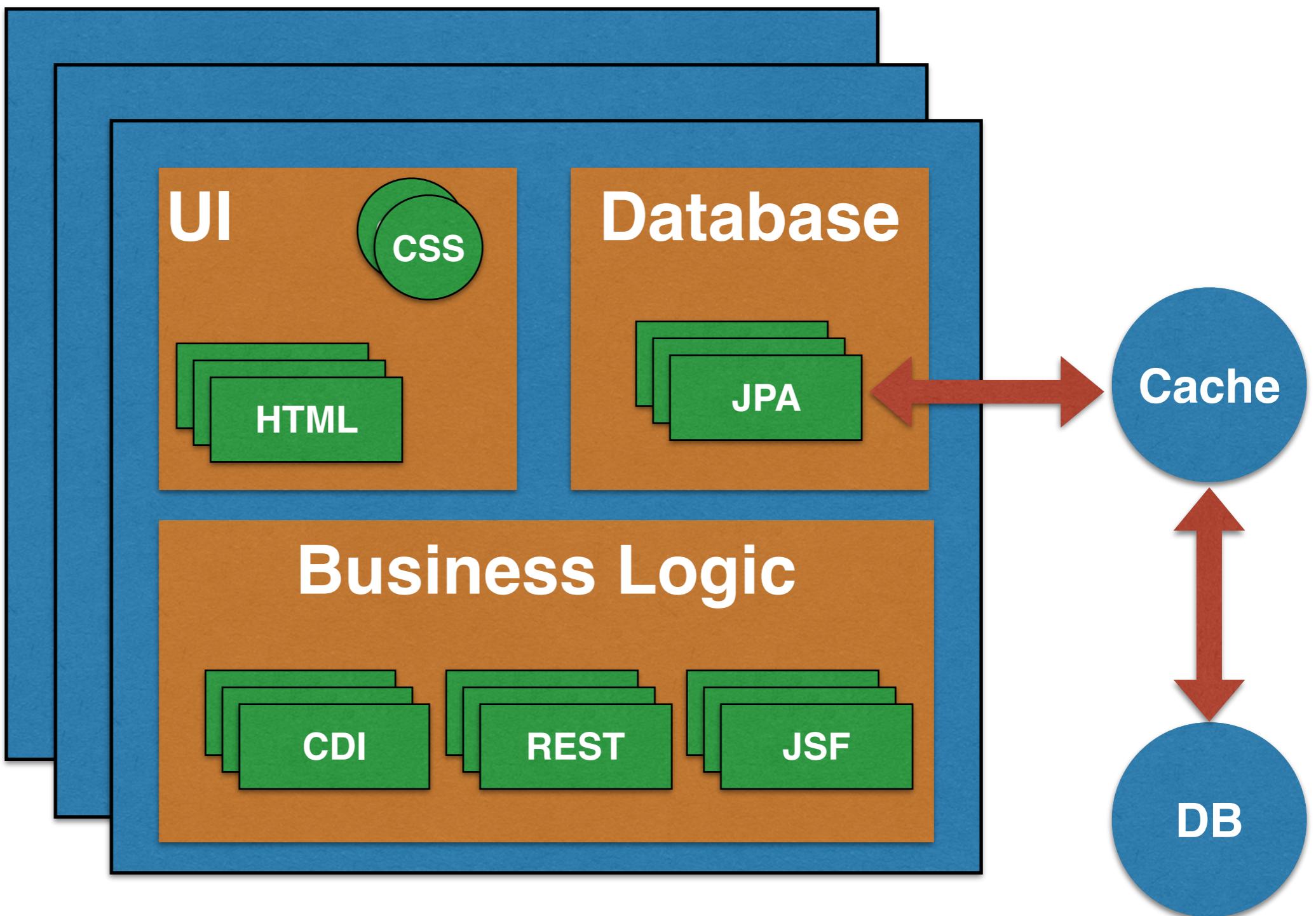
# Monolith Application



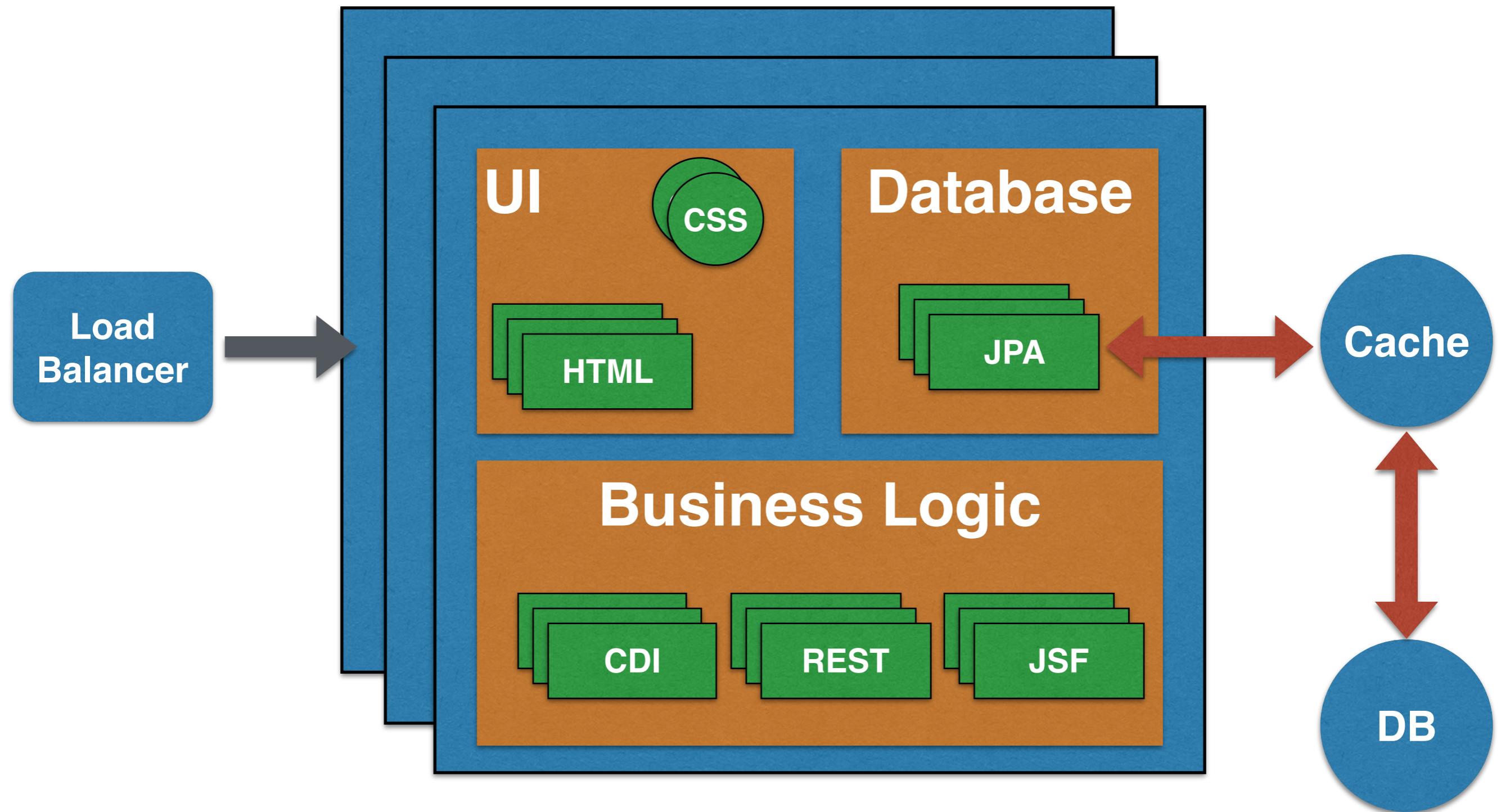
# Monolith Application



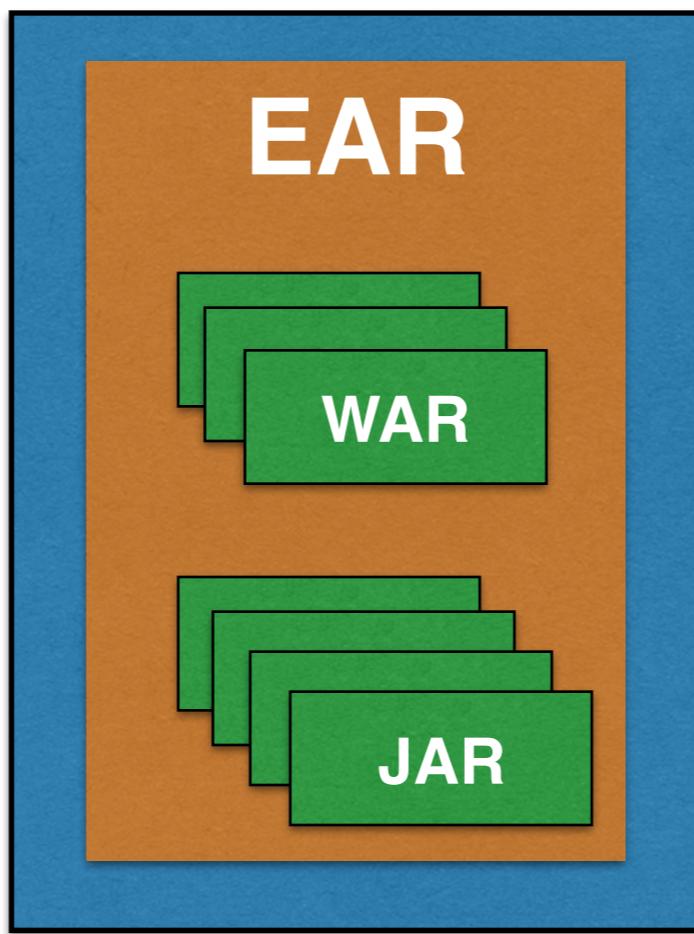
# Monolith Application



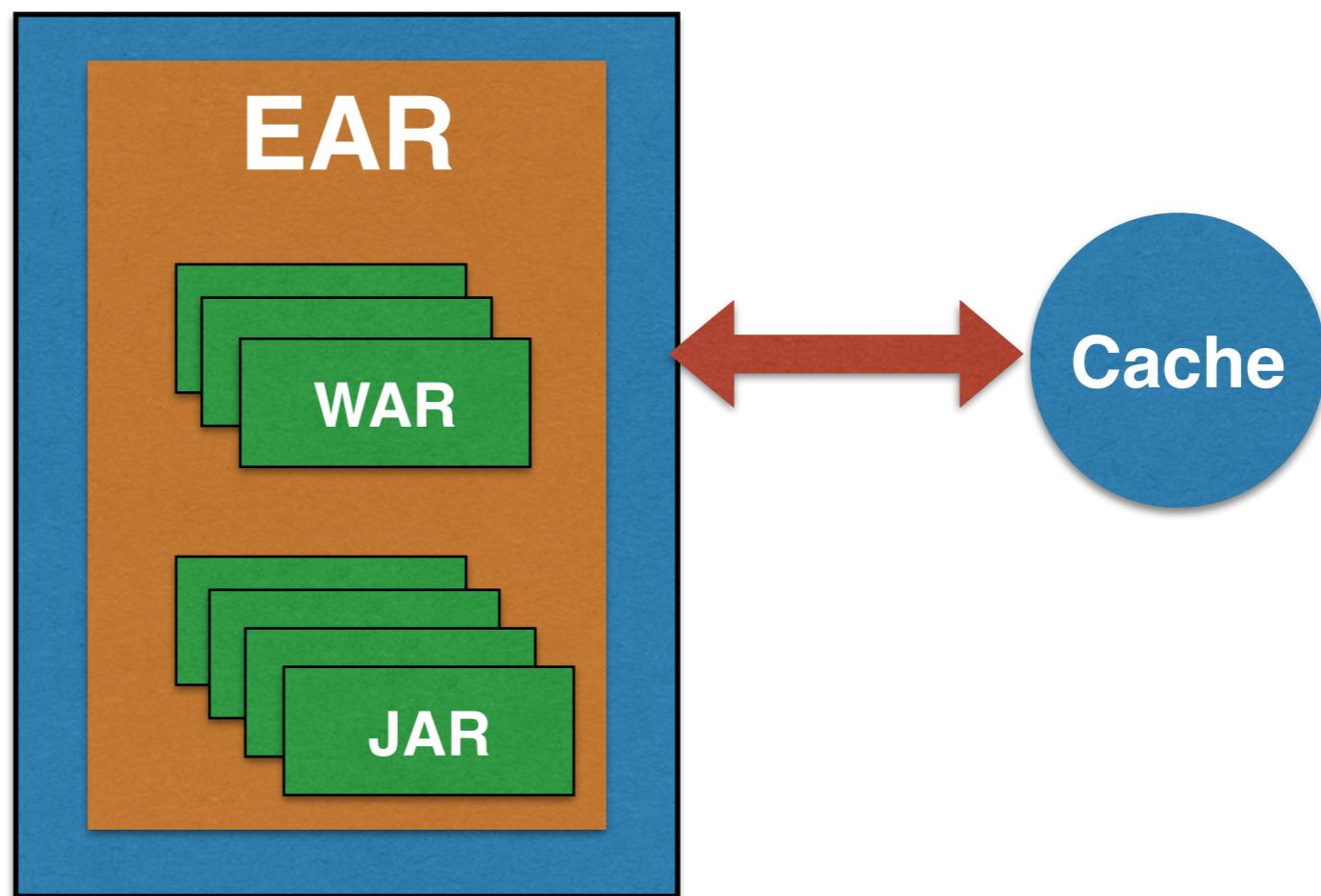
# Monolith Application



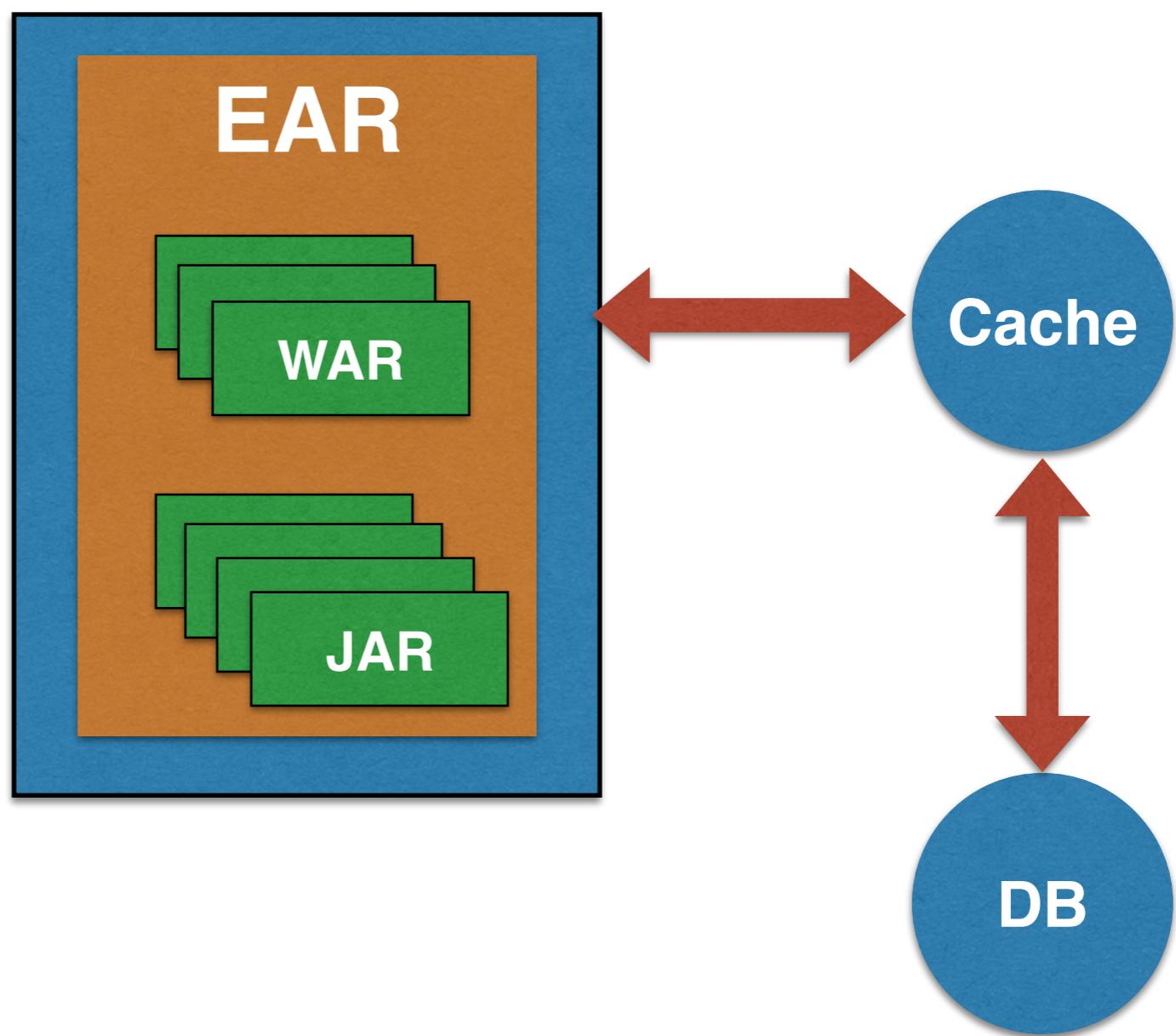
# Before Microservices



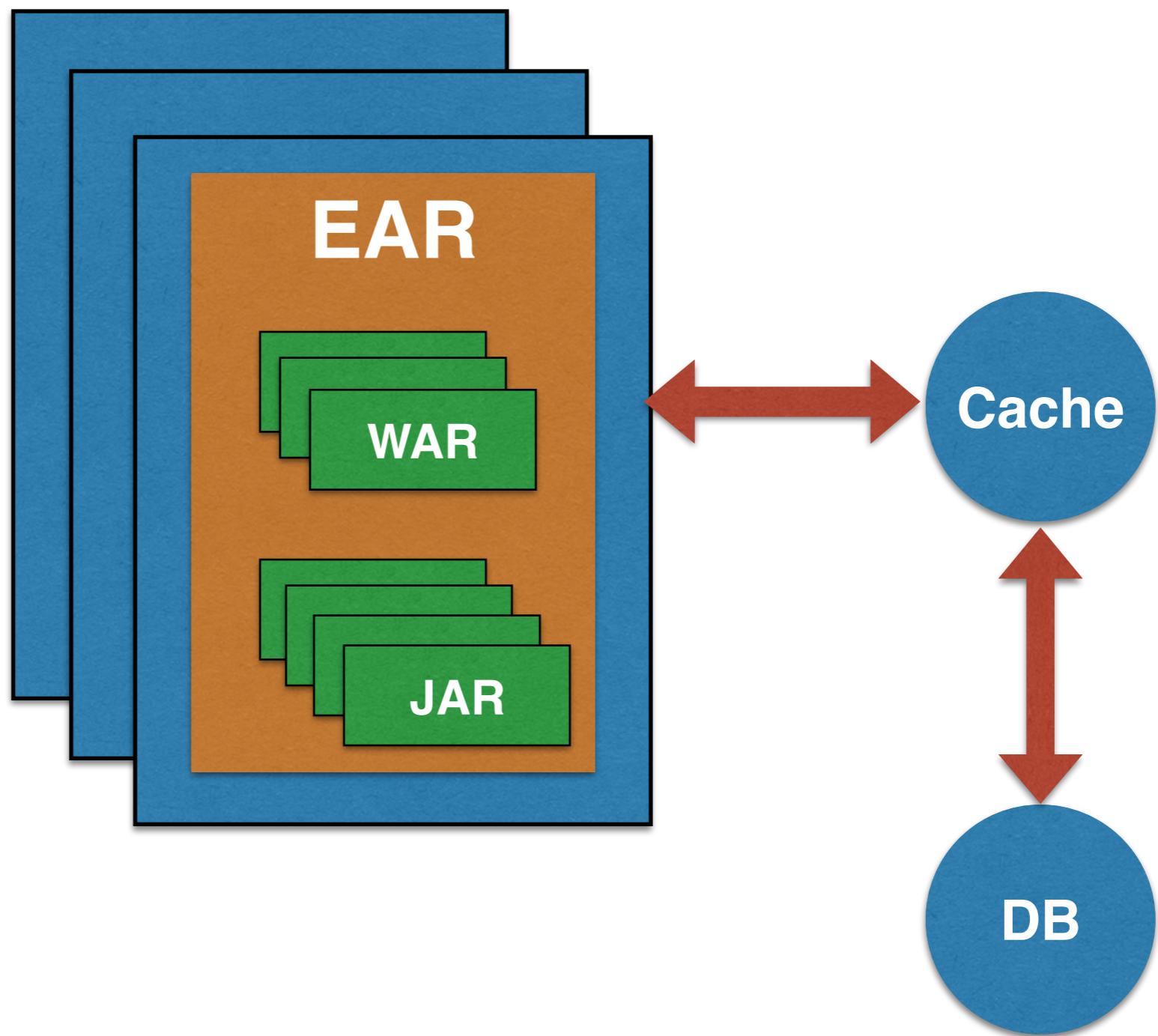
# Before Microservices



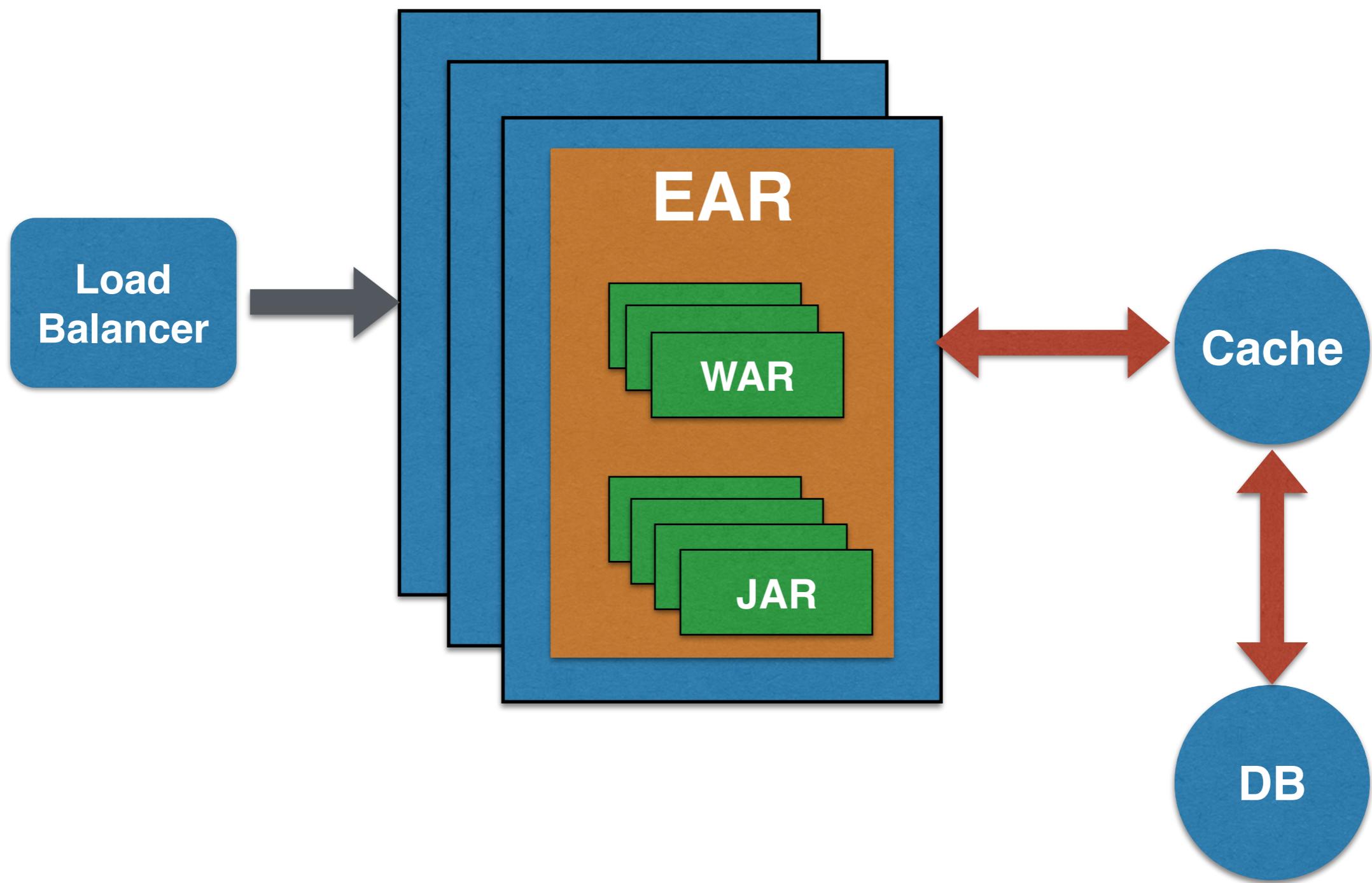
# Before Microservices



# Before Microservices



# Before Microservices



# Advantages of Monolith Application

# Advantages of Monolith Application

- Typically packaged in a single .ear

# Advantages of Monolith Application

- Typically packaged in a single .ear
- Simple to develop/deploy

# Advantages of Monolith Application

- Typically packaged in a single .ear
- Simple to develop/deploy
- Easy to test (all required services are up)

# Disadvantages of Monolith Application

# Disadvantages of Monolith Application

- Difficult to develop and maintain

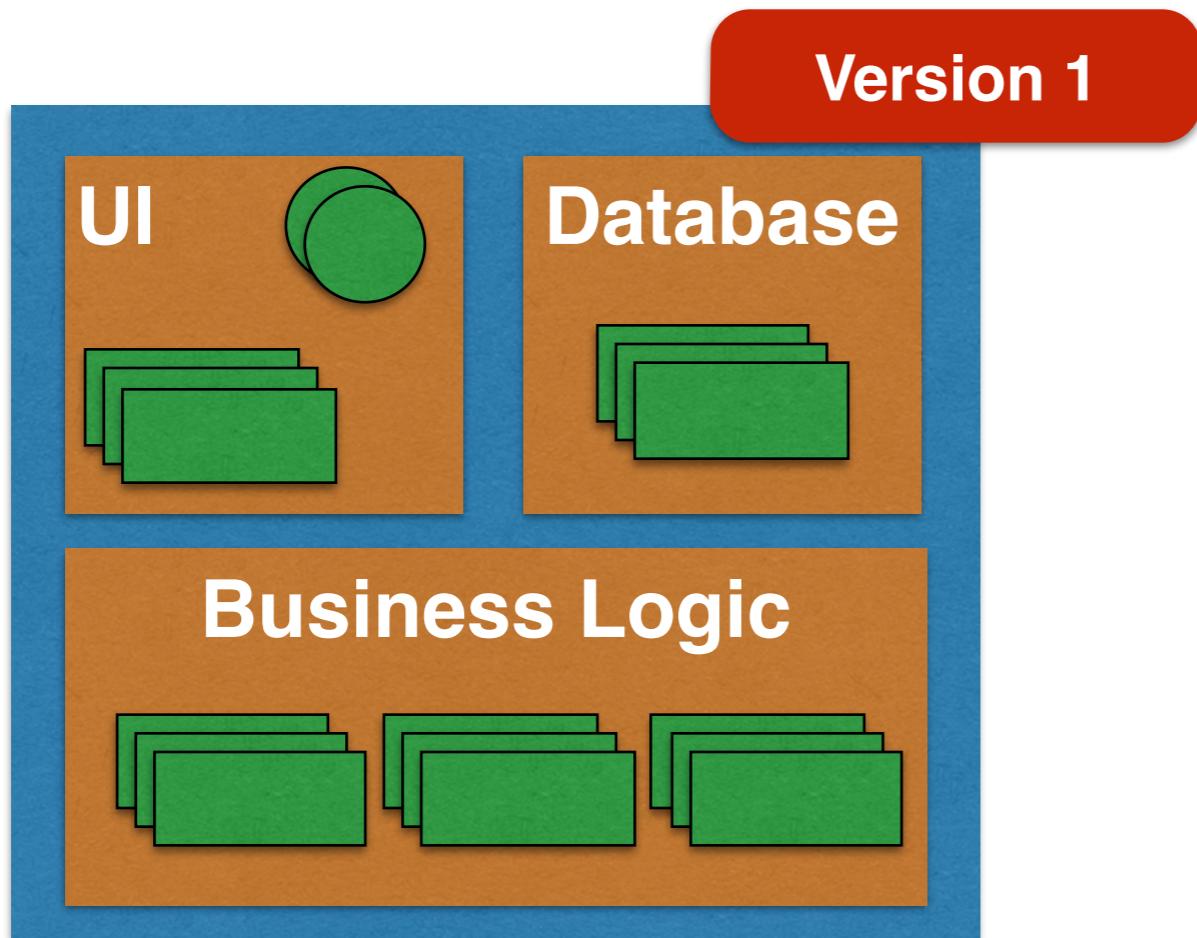
# Disadvantages of Monolith Application

- Difficult to develop and maintain
- Obstacle to frequent deployments

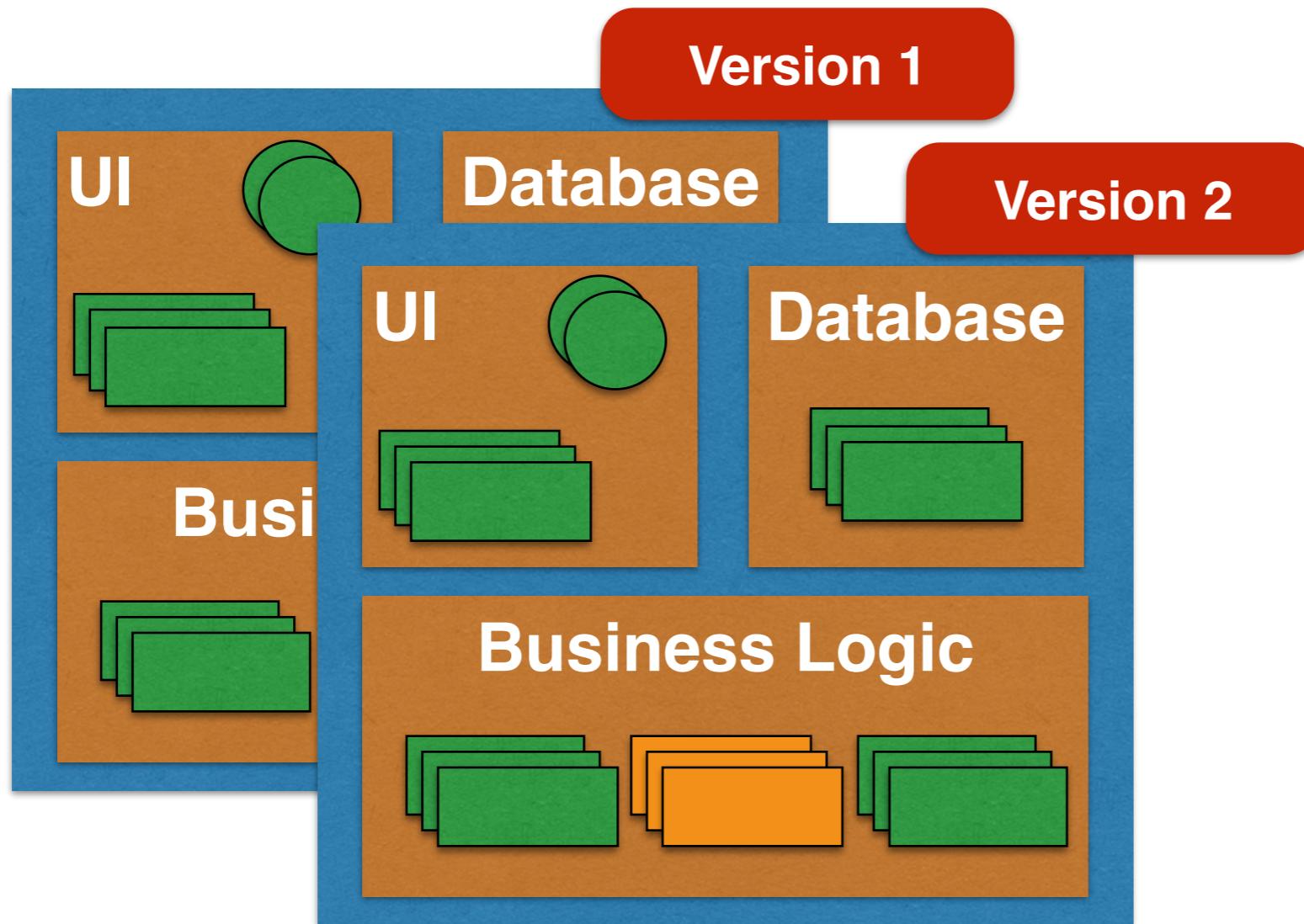
# Disadvantages of Monolith Application

- Difficult to develop and maintain
- Obstacle to frequent deployments
- Makes it difficult to try out new technologies/  
framework

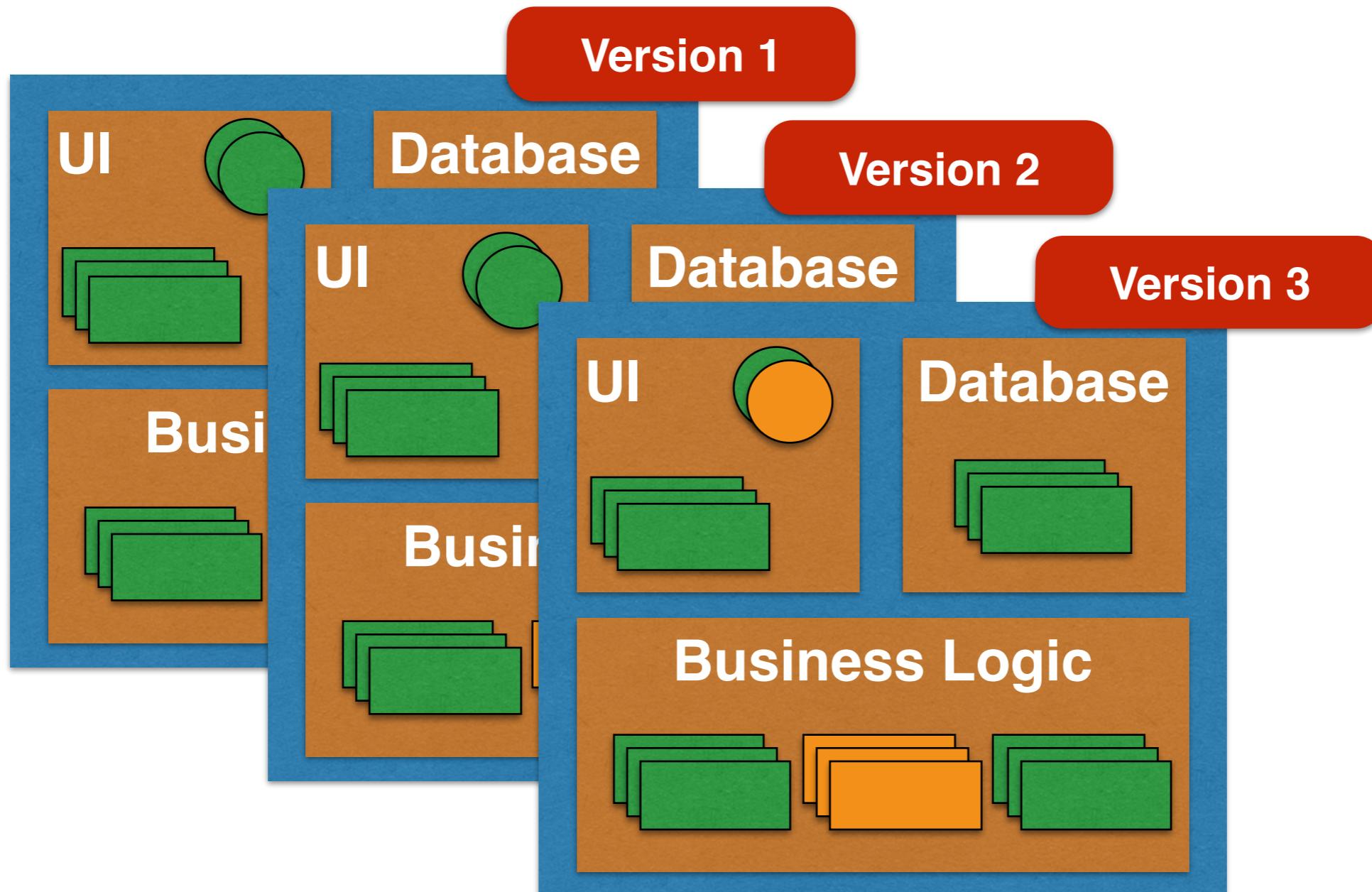
# Monolith Application



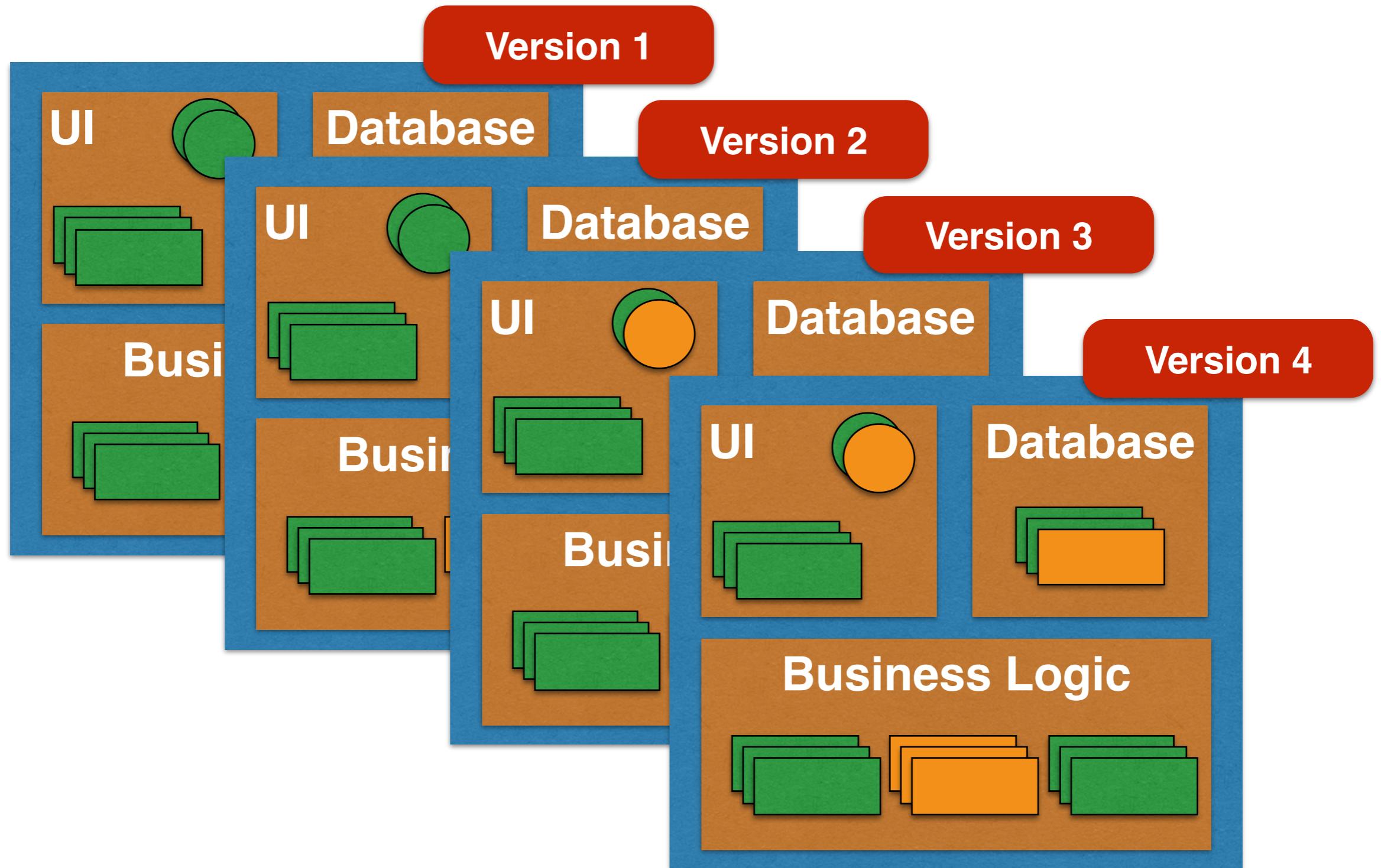
# Monolith Application



# Monolith Application



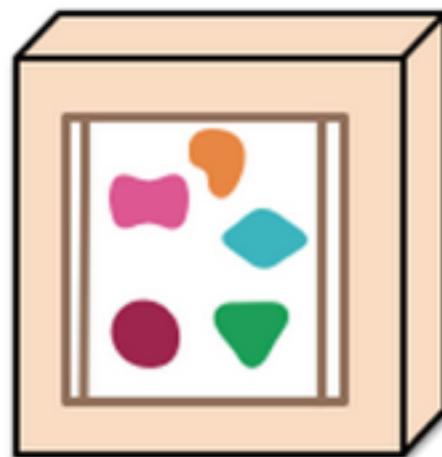
# Monolith Application



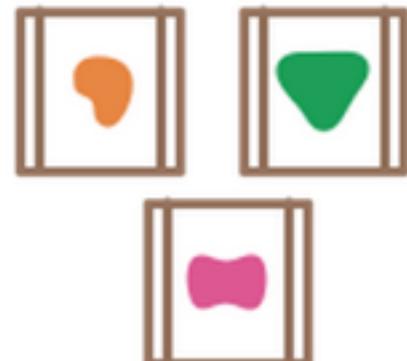
*A monolithic application puts all its functionality into a single process...*



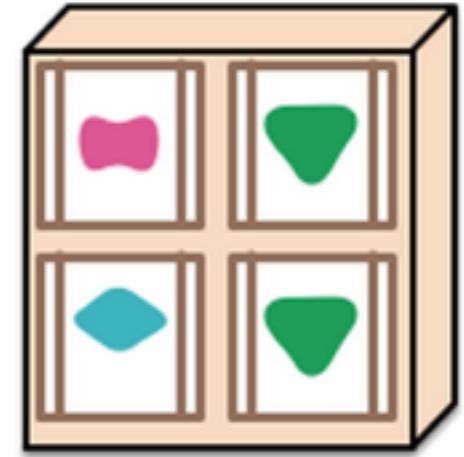
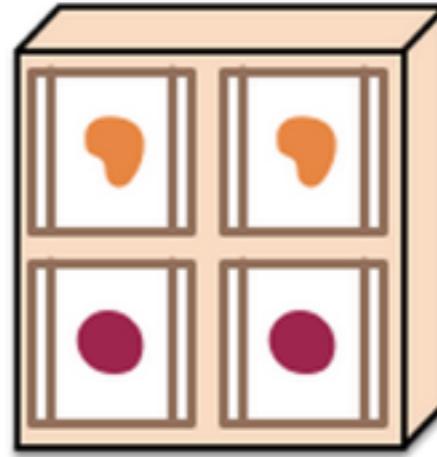
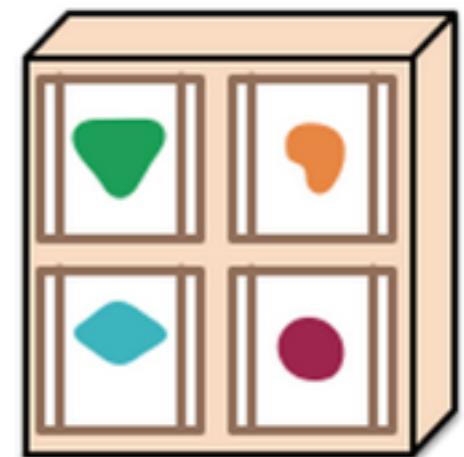
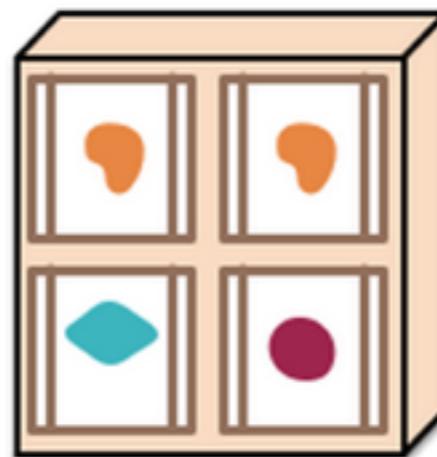
*... and scales by replicating the monolith on multiple servers*



*A microservices architecture puts each element of functionality into a separate service...*



*... and scales by distributing these services across servers, replicating as needed.*



# Characteristics of μservices

# Characteristics of μservices

- Explicitly published interface, bounded context

# Characteristics of μservices

- Explicitly published interface, bounded context
- Domain driven design (full-stack developer)

# Characteristics of μservices

- Explicitly published interface, bounded context
- Domain driven design (full-stack developer)
- Independently deployable and automated (CI / CD)

# Characteristics of μservices

- Explicitly published interface, bounded context
- Domain driven design (full-stack developer)
- Independently deployable and automated (CI / CD)
- Designed for failure

# Characteristics of μservices

- Explicitly published interface, bounded context
- Domain driven design (full-stack developer)
- Independently deployable and automated (CI / CD)
- Designed for failure
- Use IPC to communicate

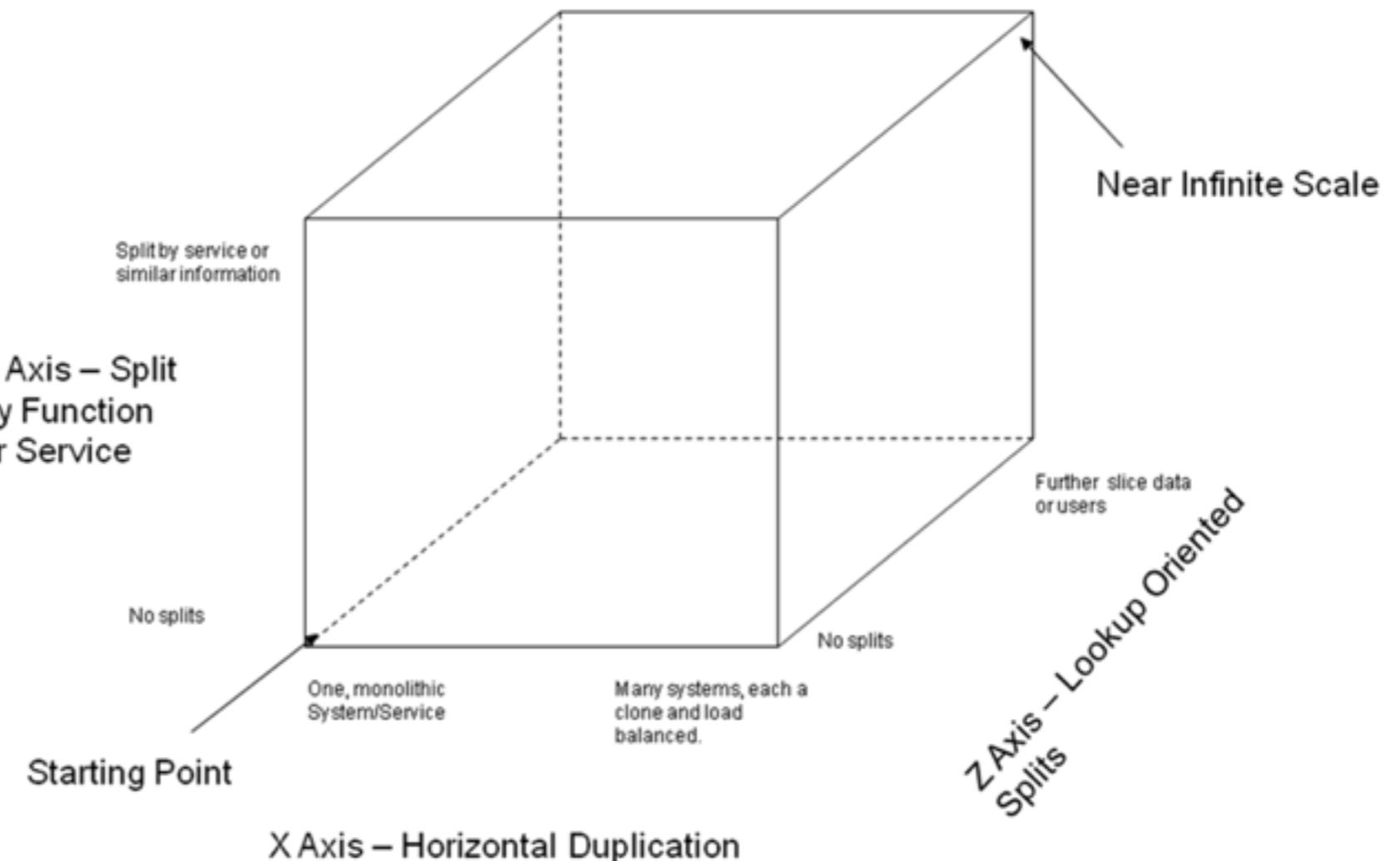
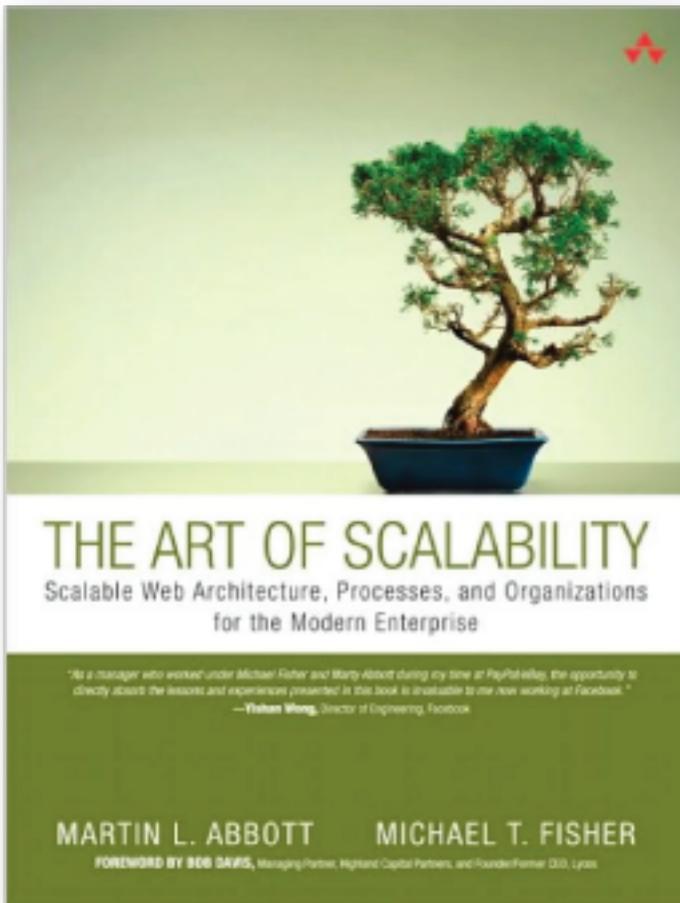
# Characteristics of μservices

- Explicitly published interface, bounded context
- Domain driven design (full-stack developer)
- Independently deployable and automated (CI / CD)
- Designed for failure
- Use IPC to communicate
- Full responsibility - “you build it, you run it”

# Characteristics of μservices

- Explicitly published interface, bounded context
- Domain driven design (full-stack developer)
- Independently deployable and automated (CI / CD)
- Designed for failure
- Use IPC to communicate
- Full responsibility - “you build it, you run it”
- “Smart endpoints, dumb pipes”

# Scaling µservices



# Strategies for decomposing

# Strategies for decomposing

- Verb or usecase - e.g. Checkout UI

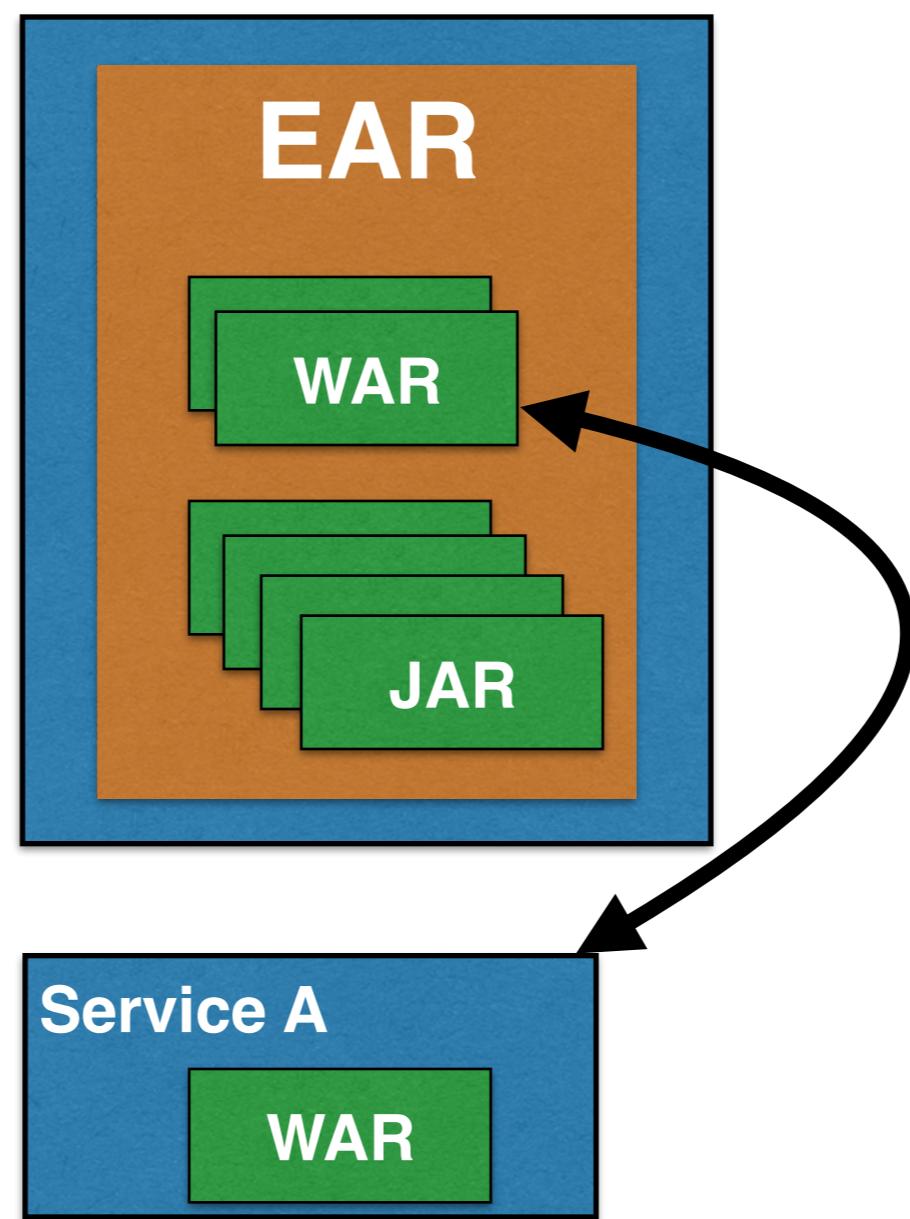
# Strategies for decomposing

- Verb or usecase - e.g. Checkout UI
- Noun - e.g. Catalog product service

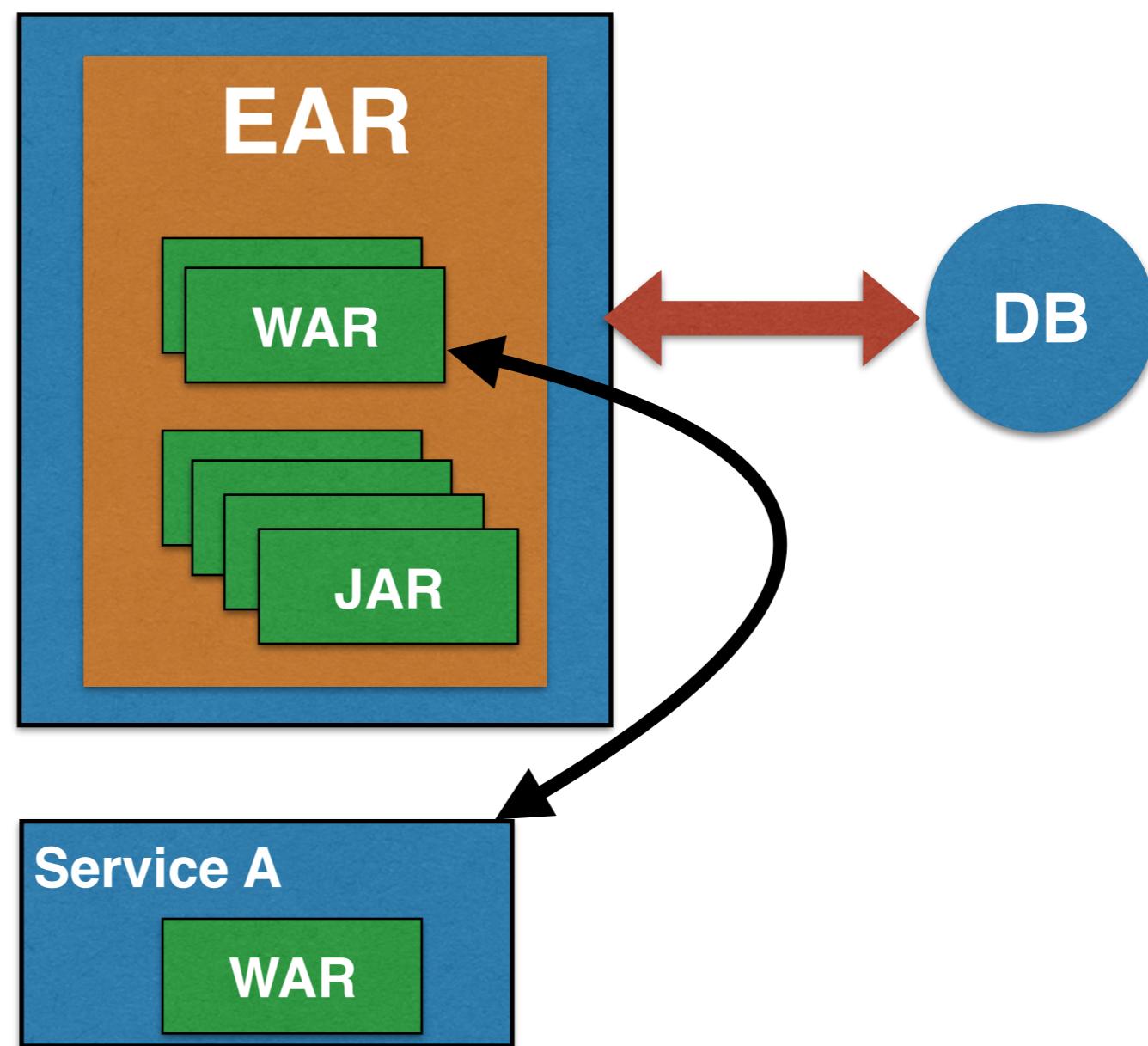
# Strategies for decomposing

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

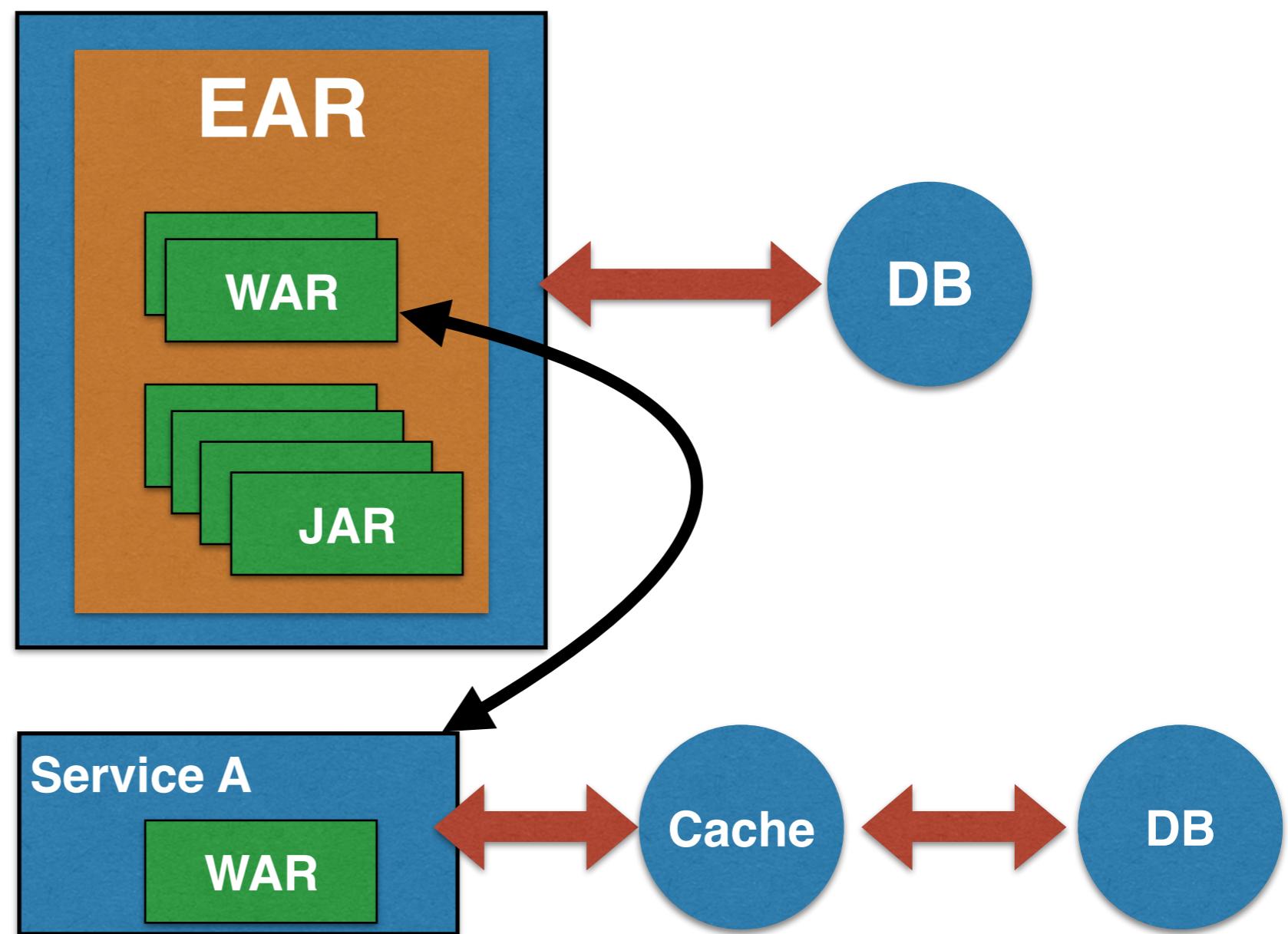
# Towards $\mu$ services



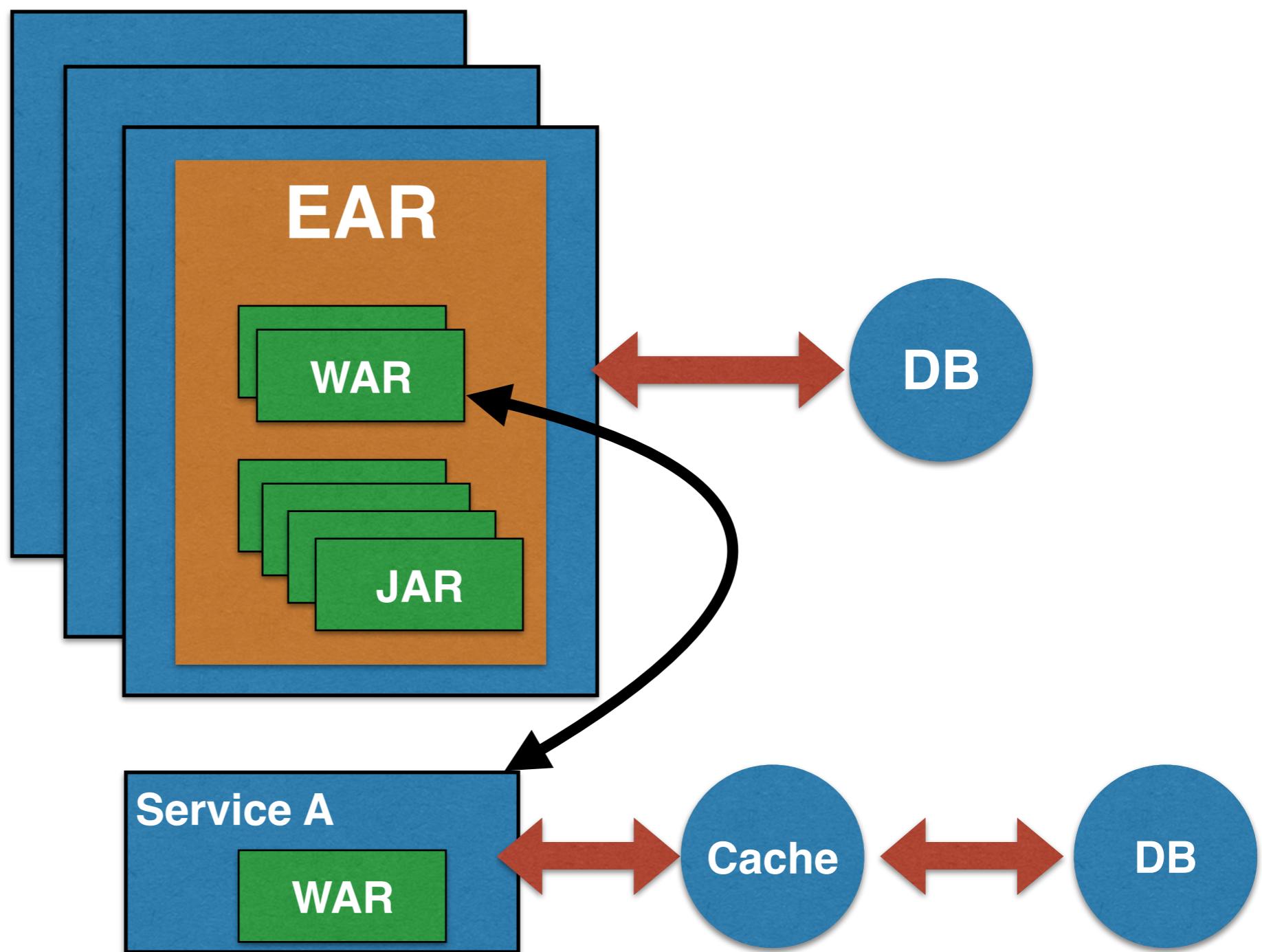
# Towards µservices



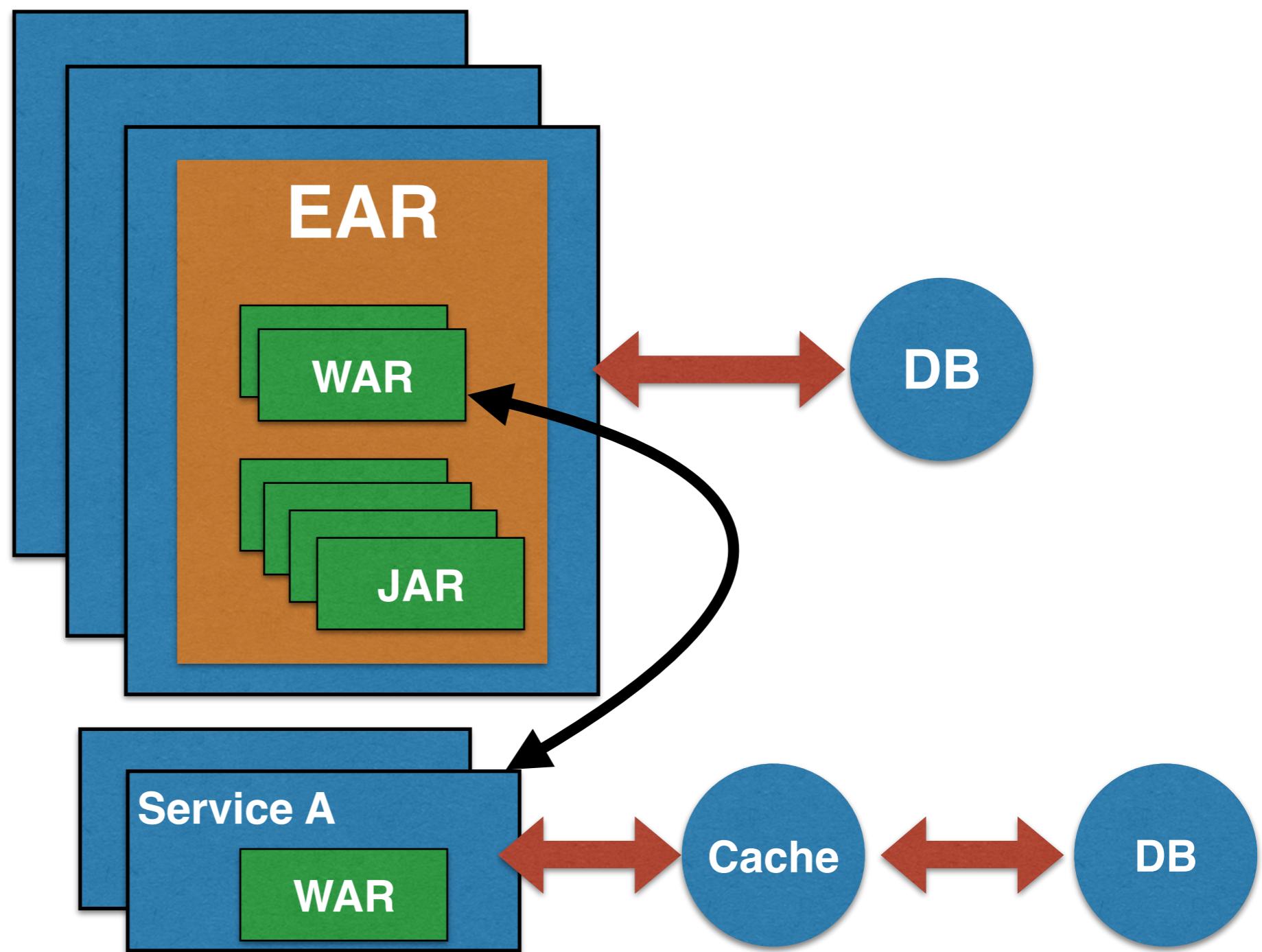
# Towards µservices



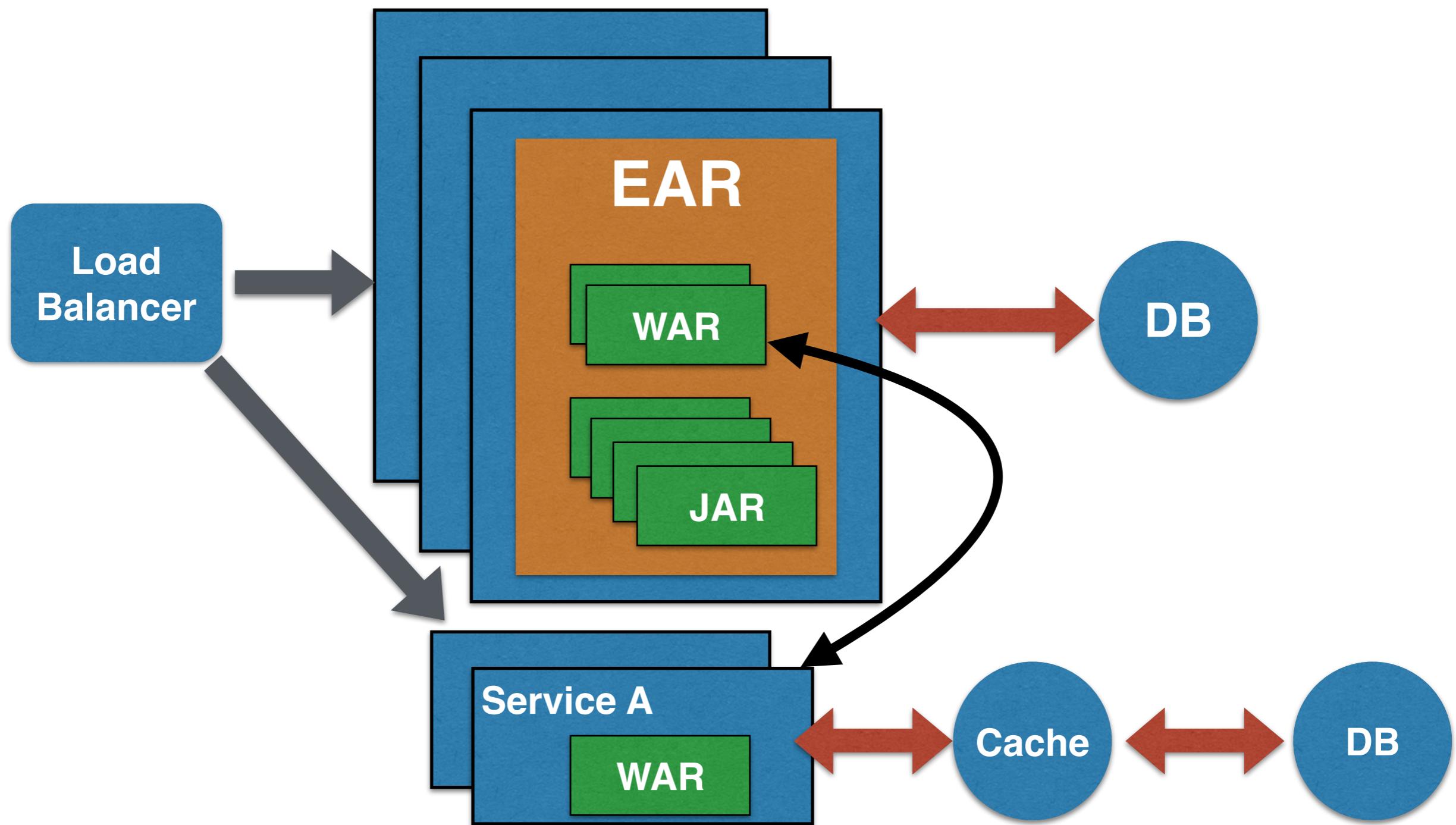
# Towards µservices



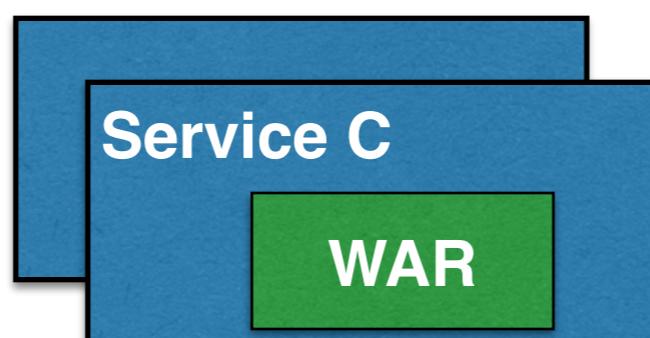
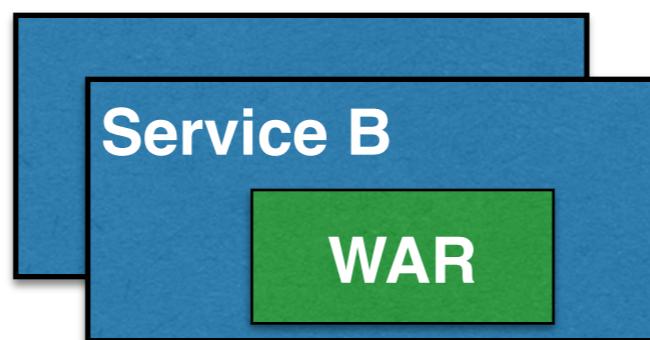
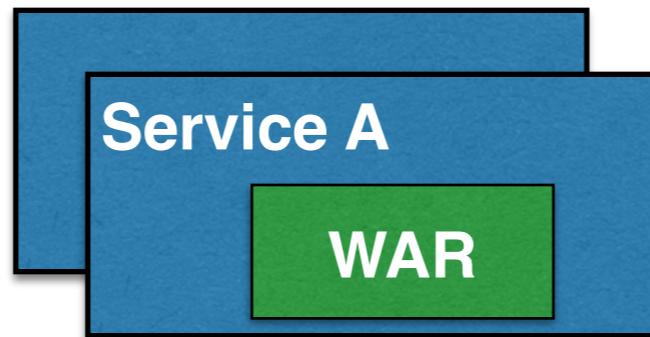
# Towards µservices



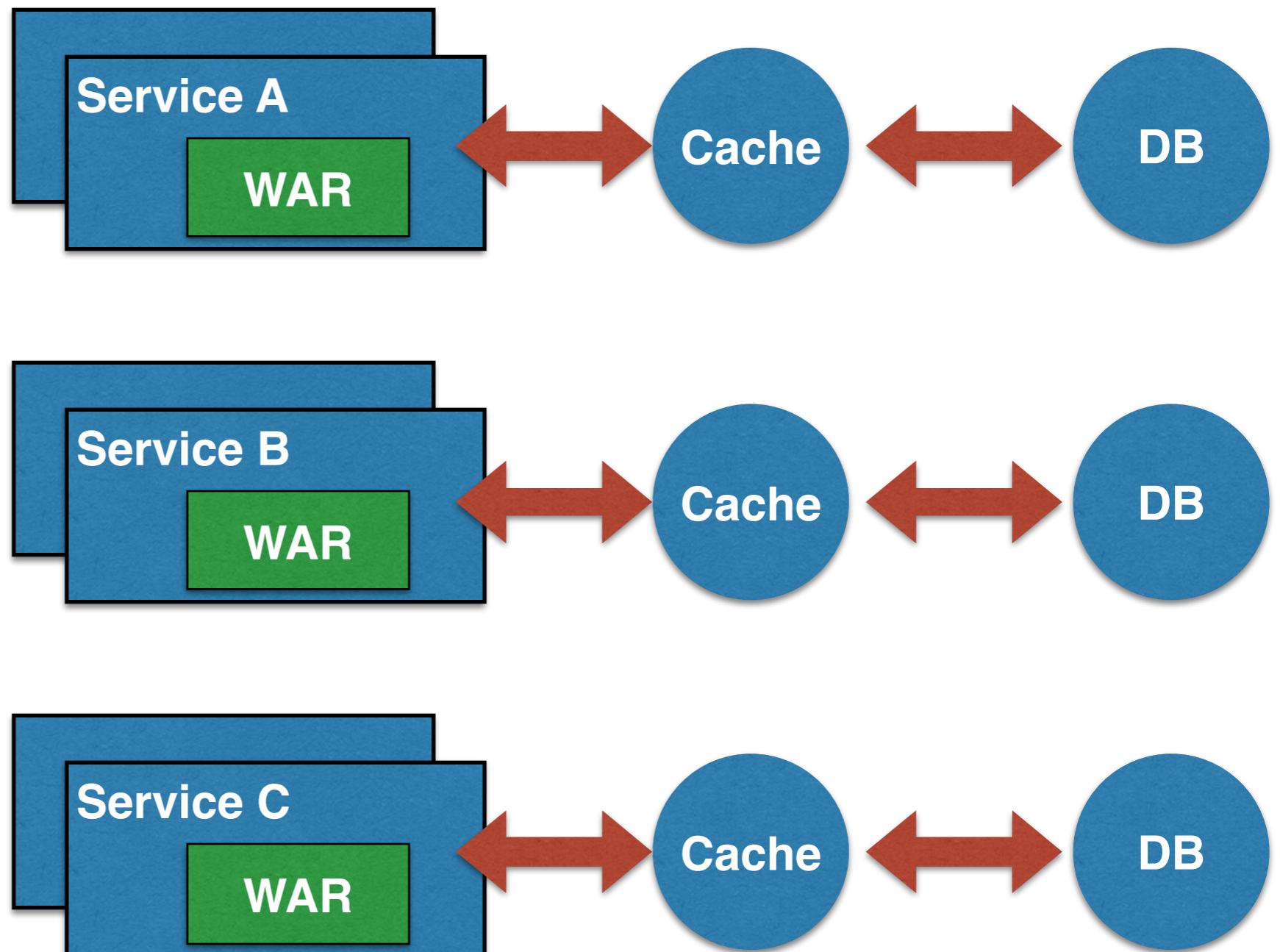
# Towards µservices



# Aggregator Pattern #1

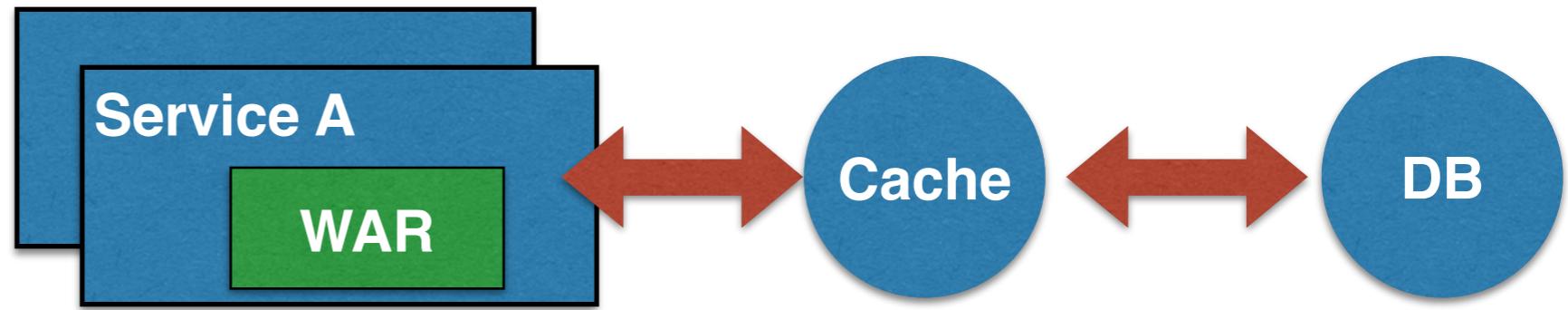


# Aggregator Pattern #1

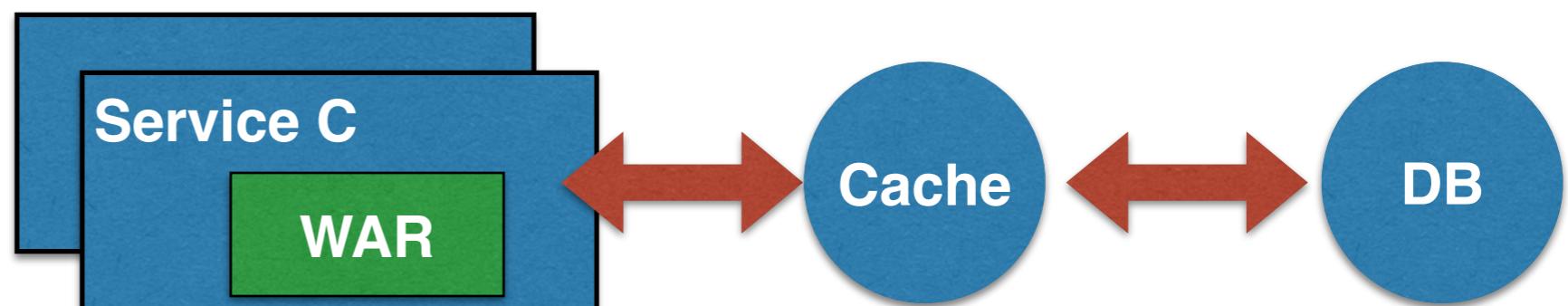
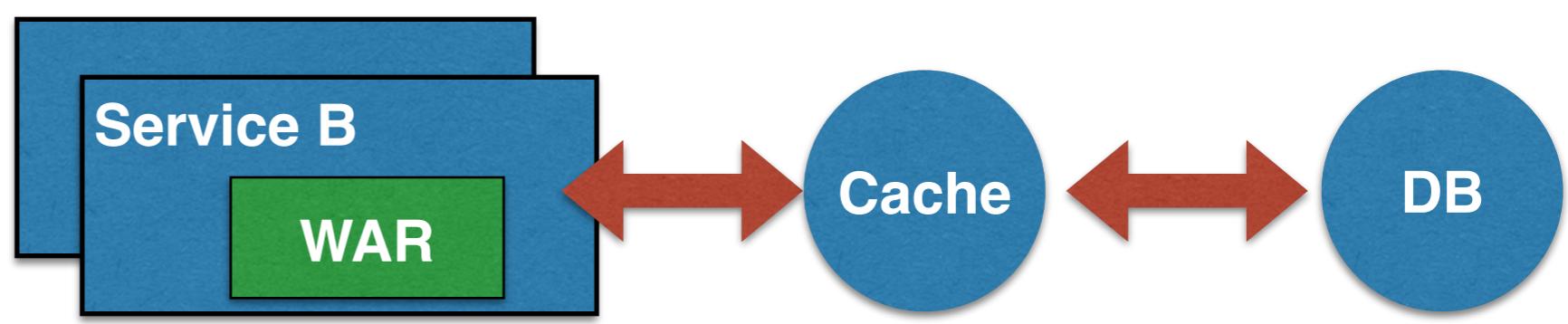


# Aggregator Pattern #1

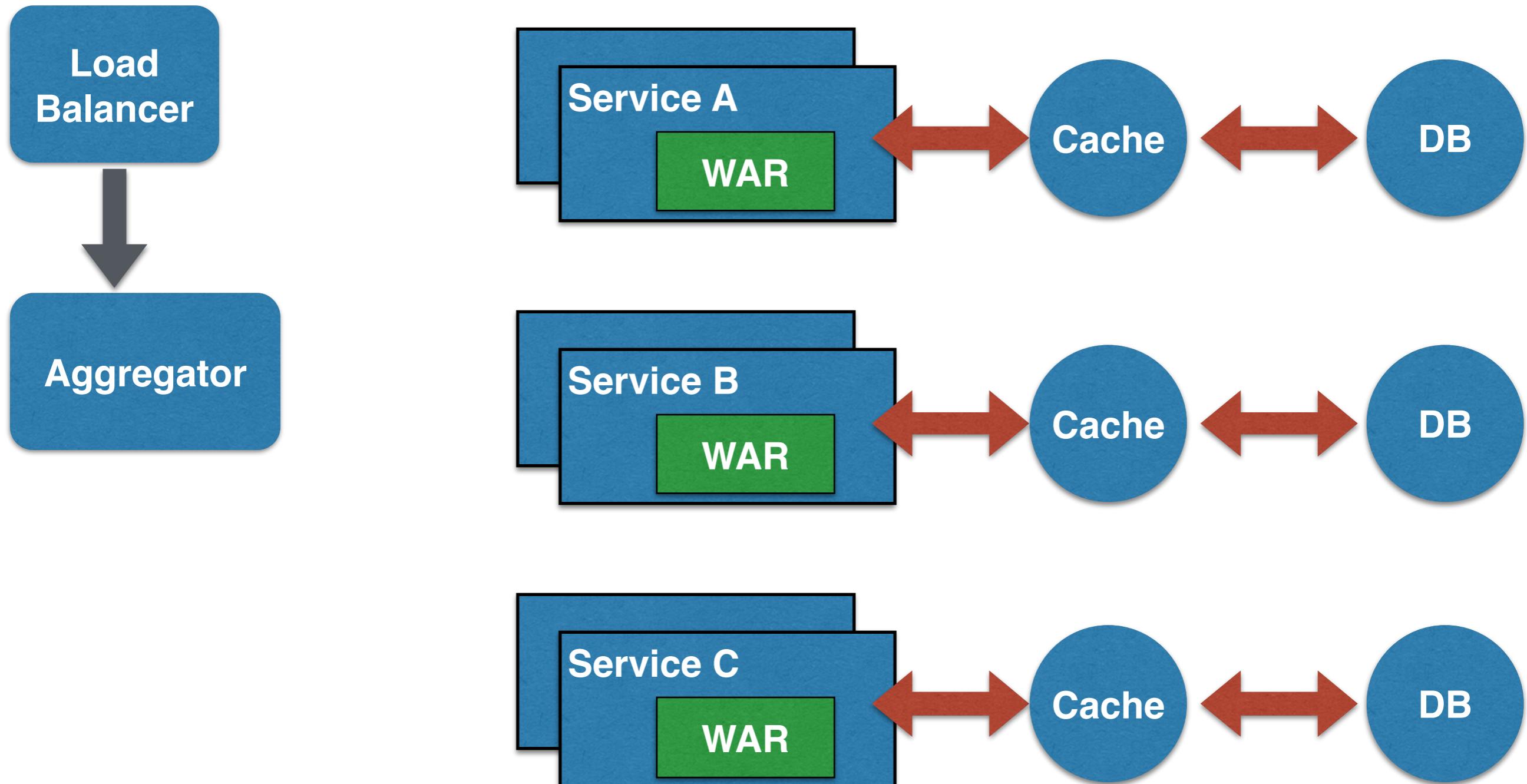
Load  
Balancer



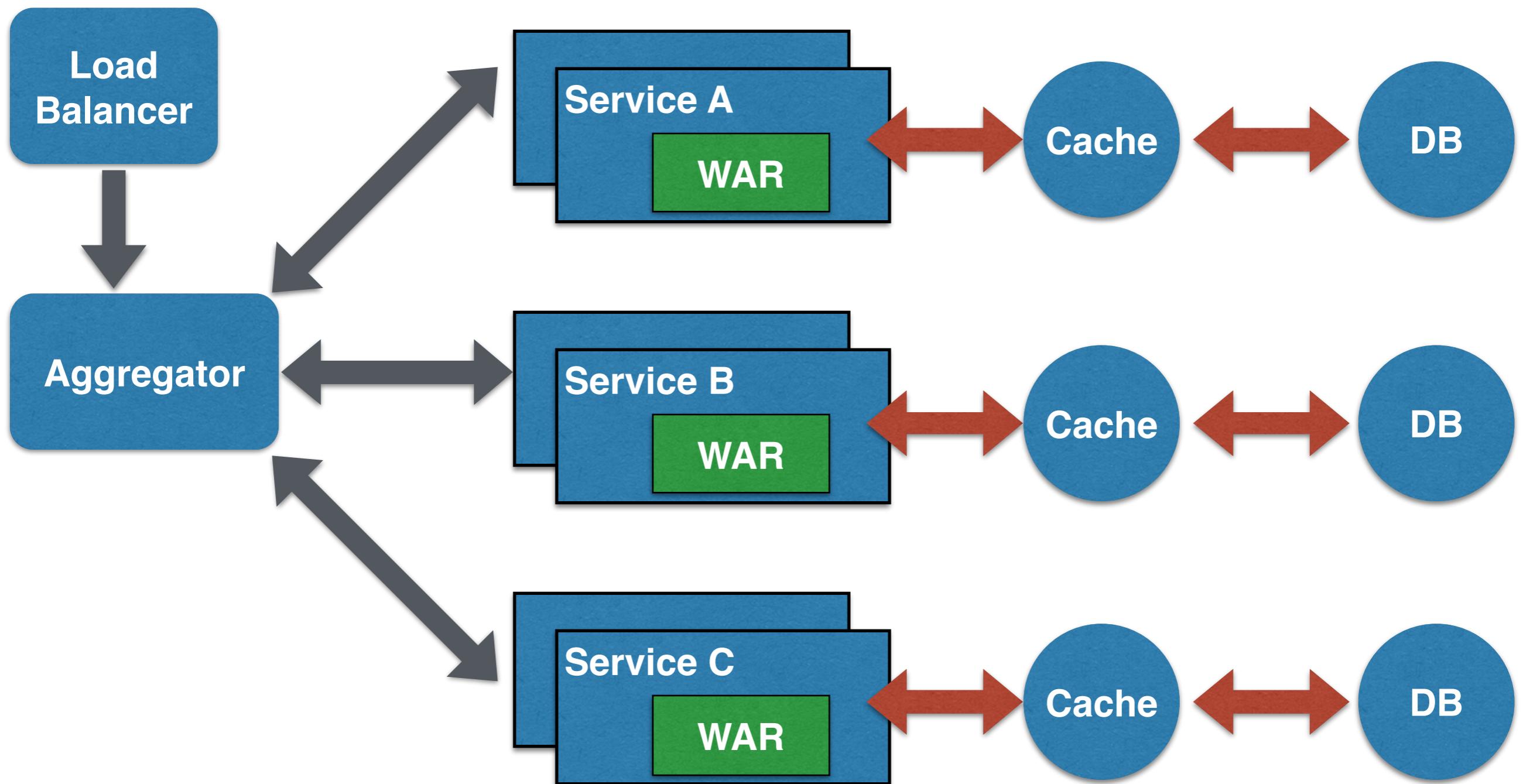
Aggregator



# Aggregator Pattern #1

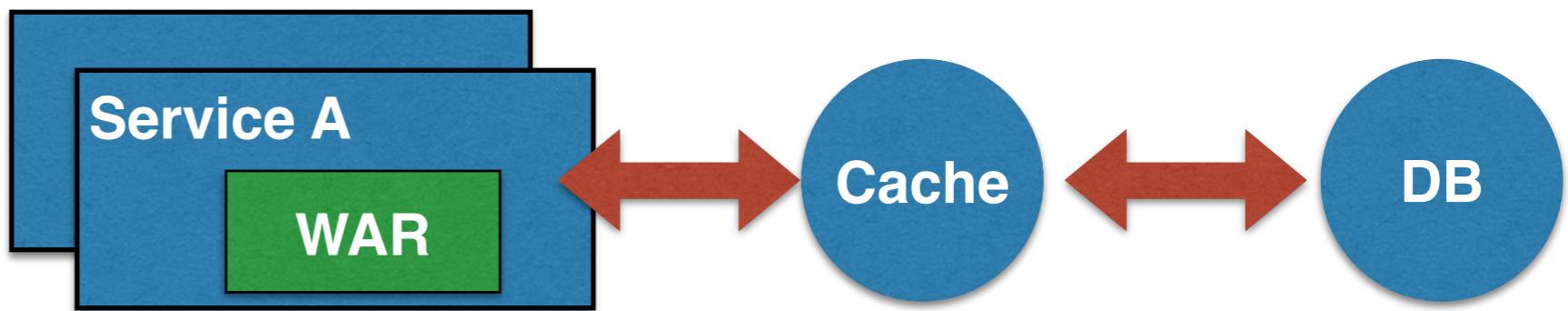


# Aggregator Pattern #1

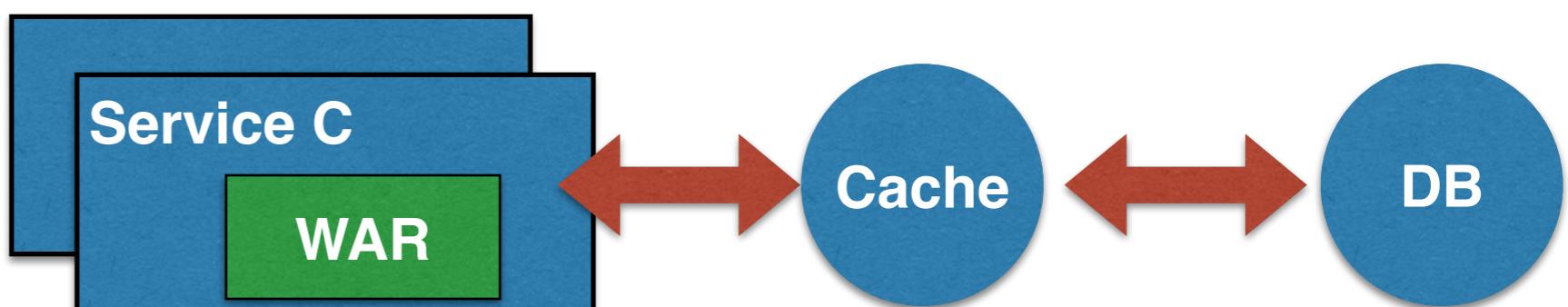
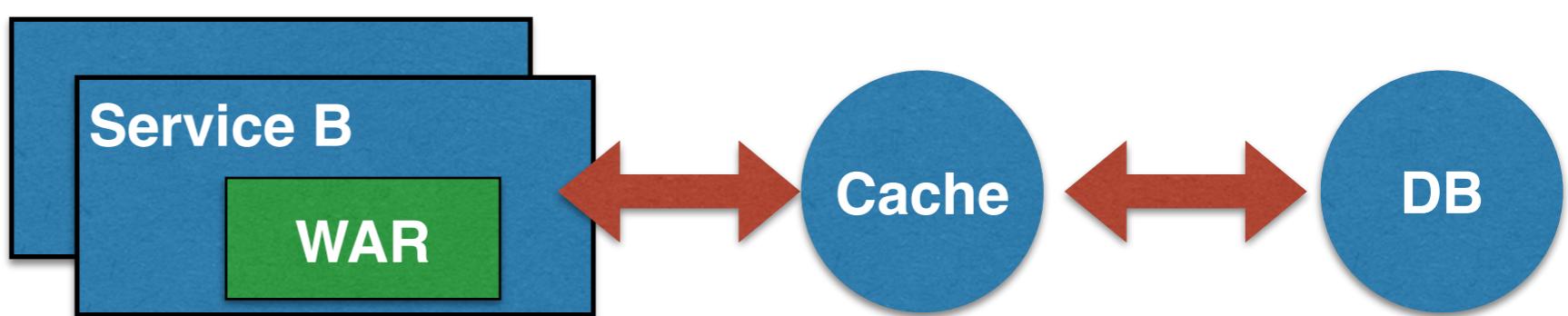


# Proxy Pattern #1.5

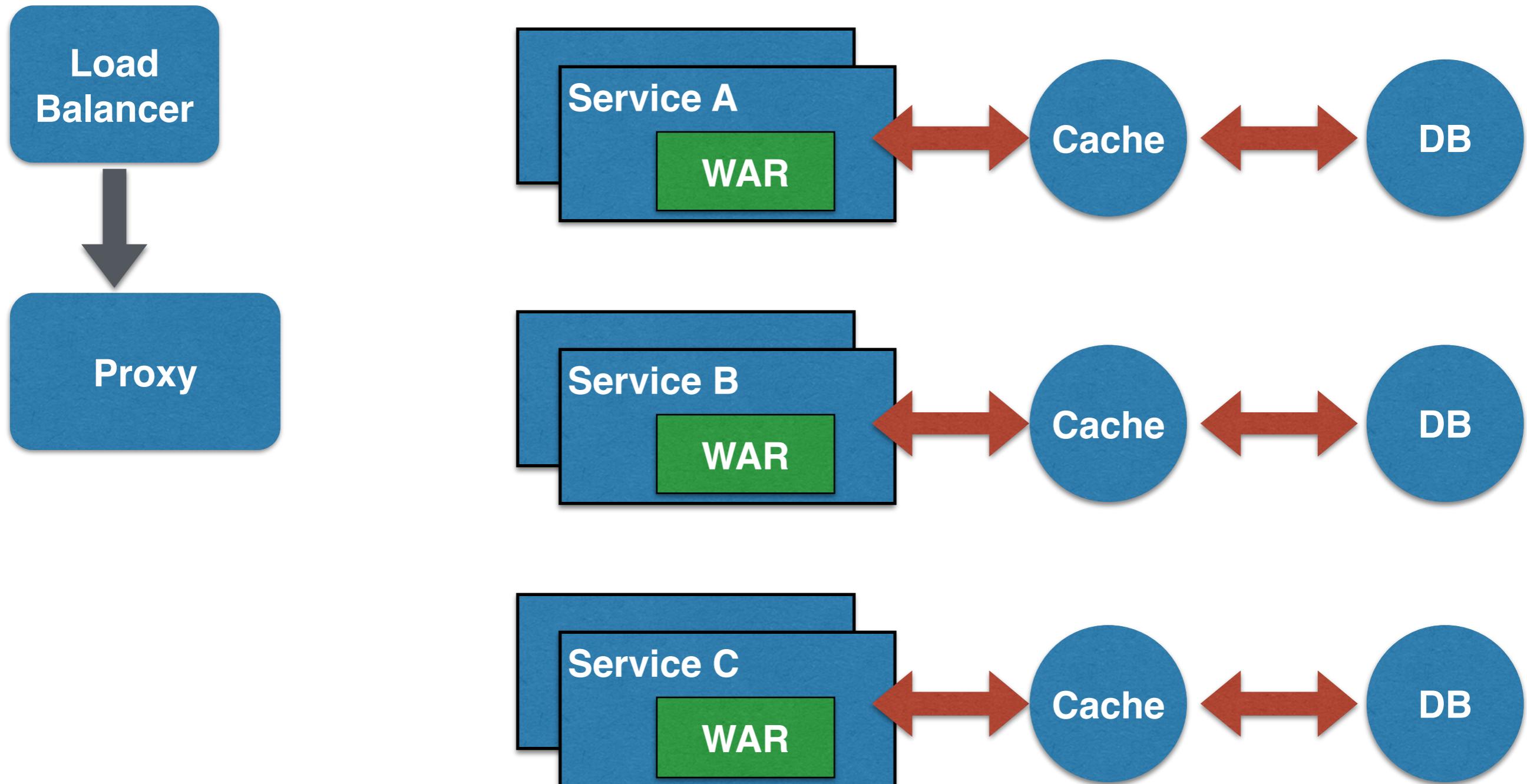
Load  
Balancer



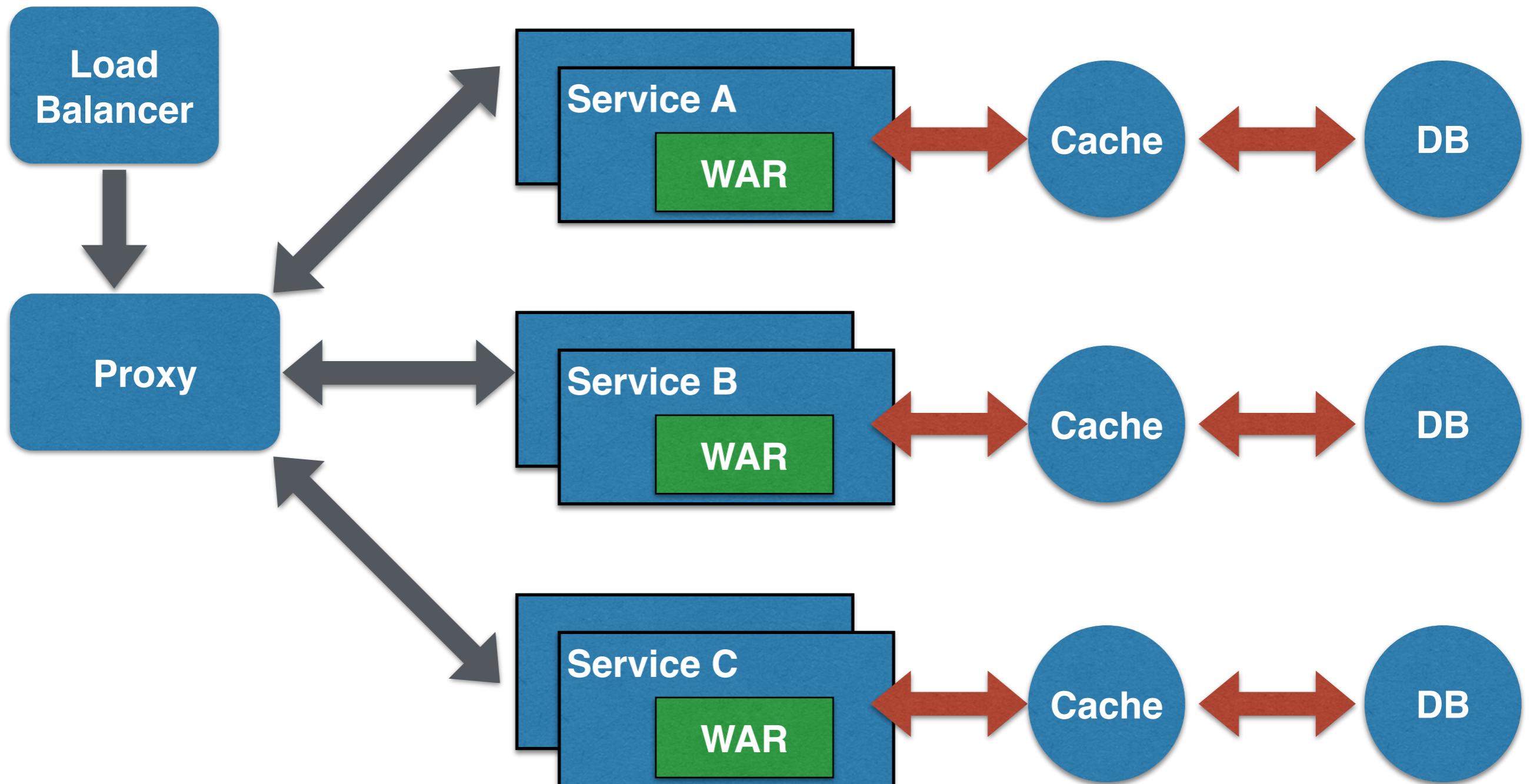
Proxy



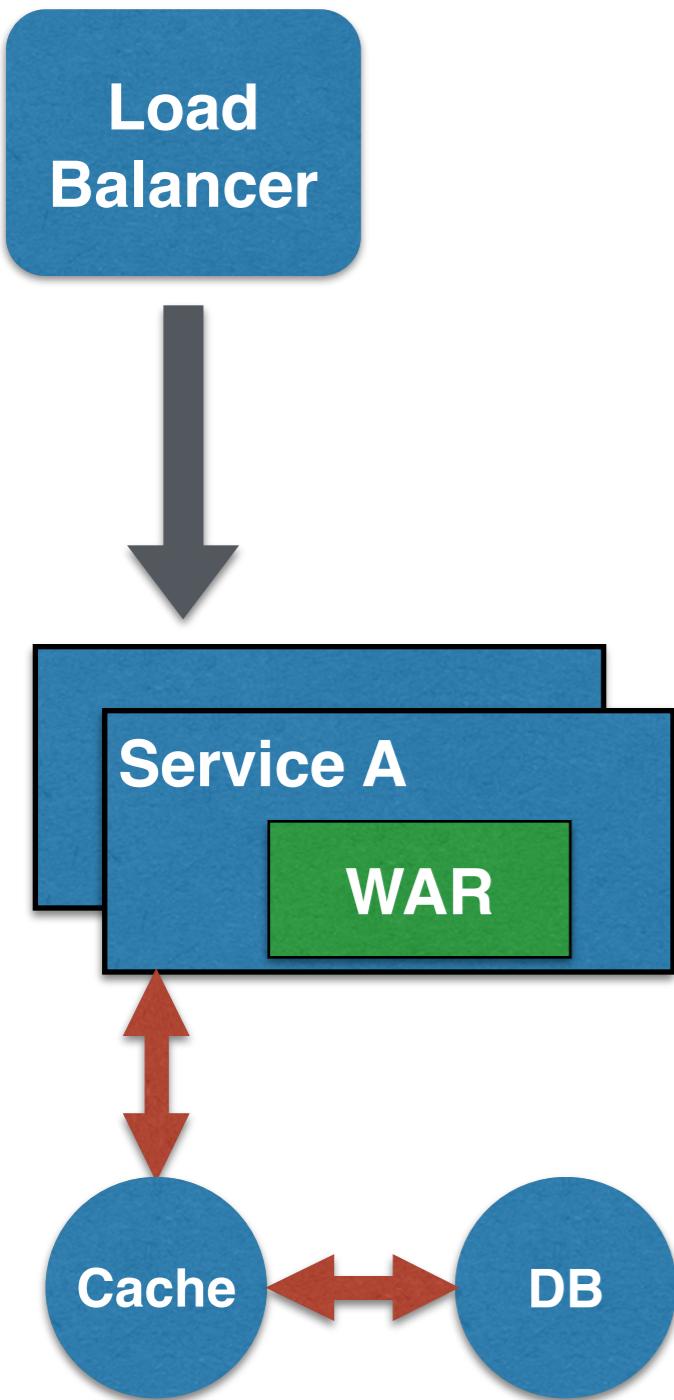
# Proxy Pattern #1.5



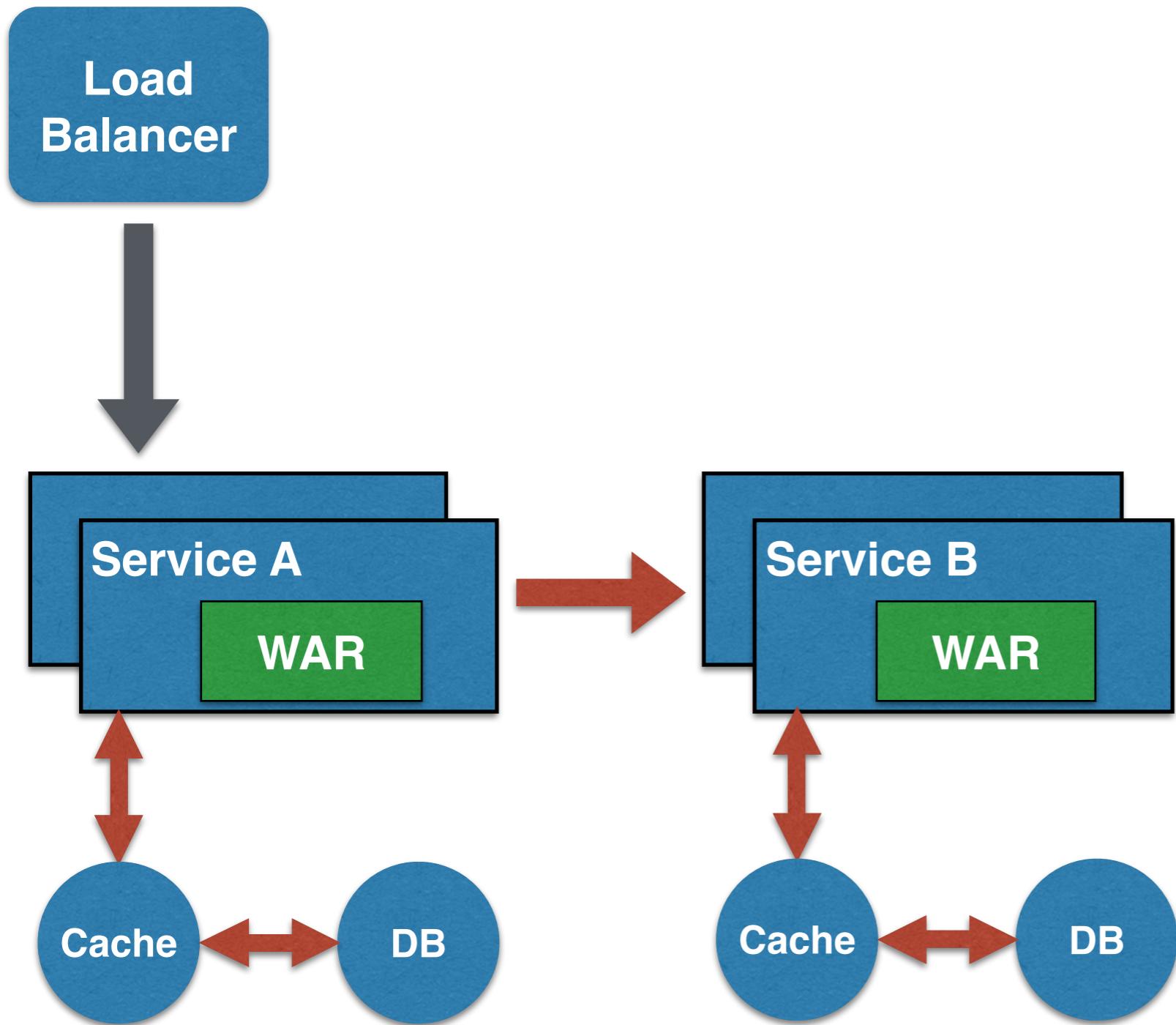
# Proxy Pattern #1.5



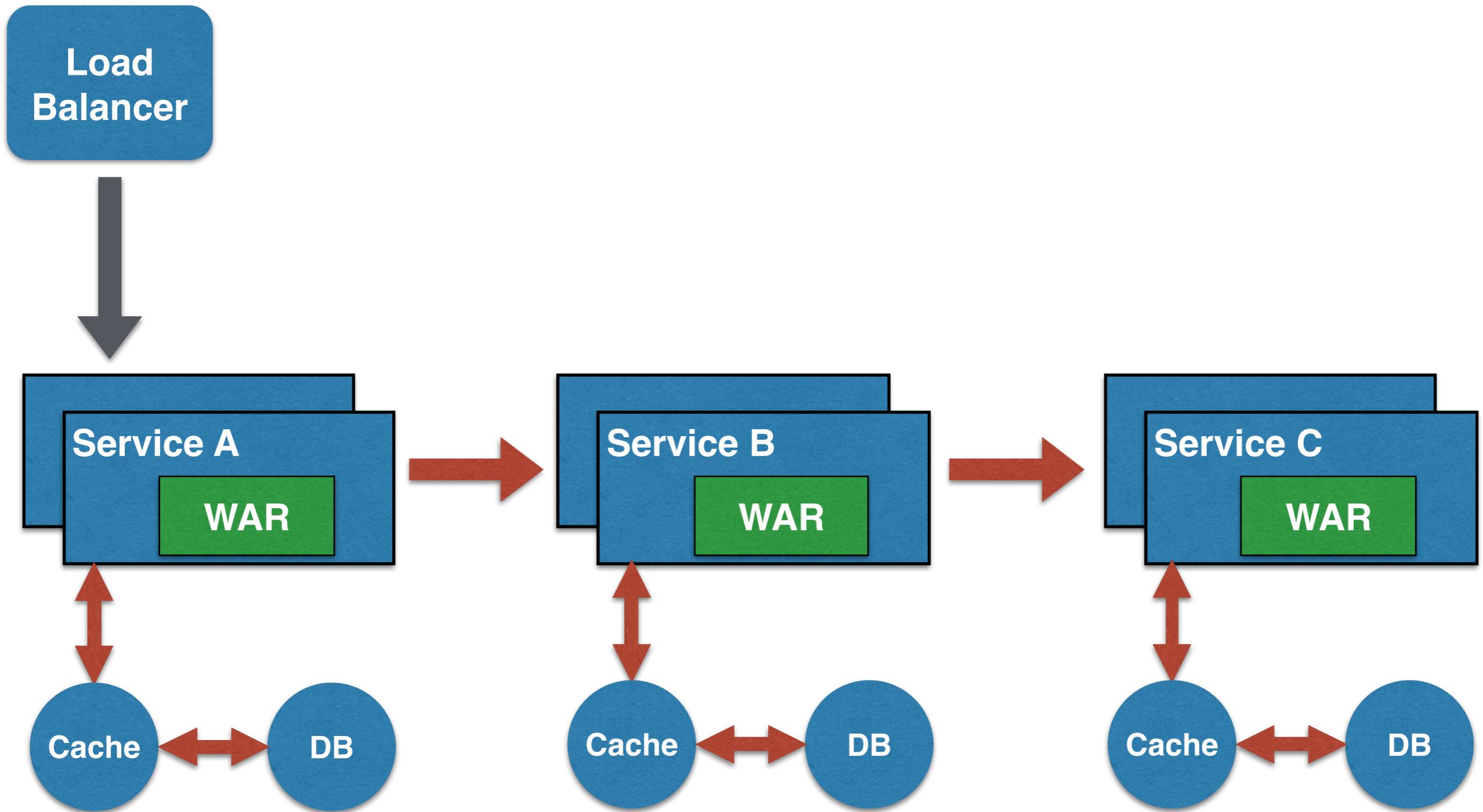
# Chained Pattern #2



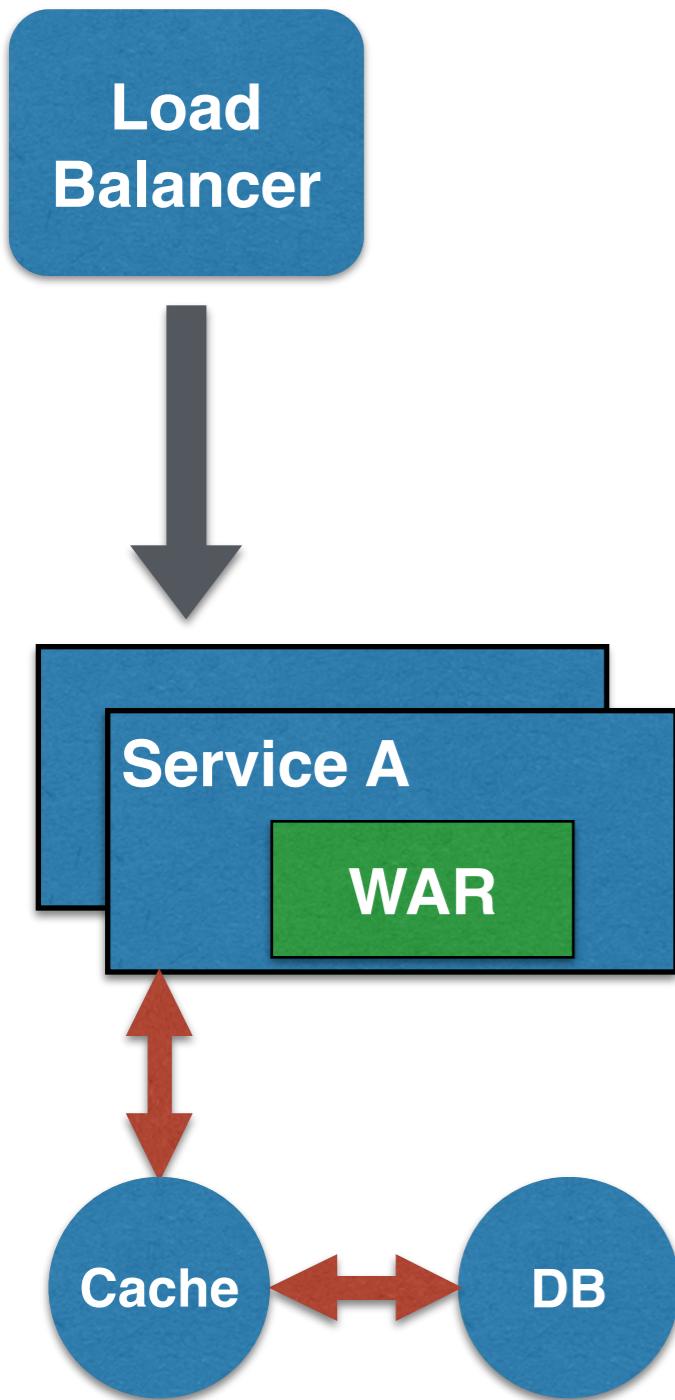
# Chained Pattern #2



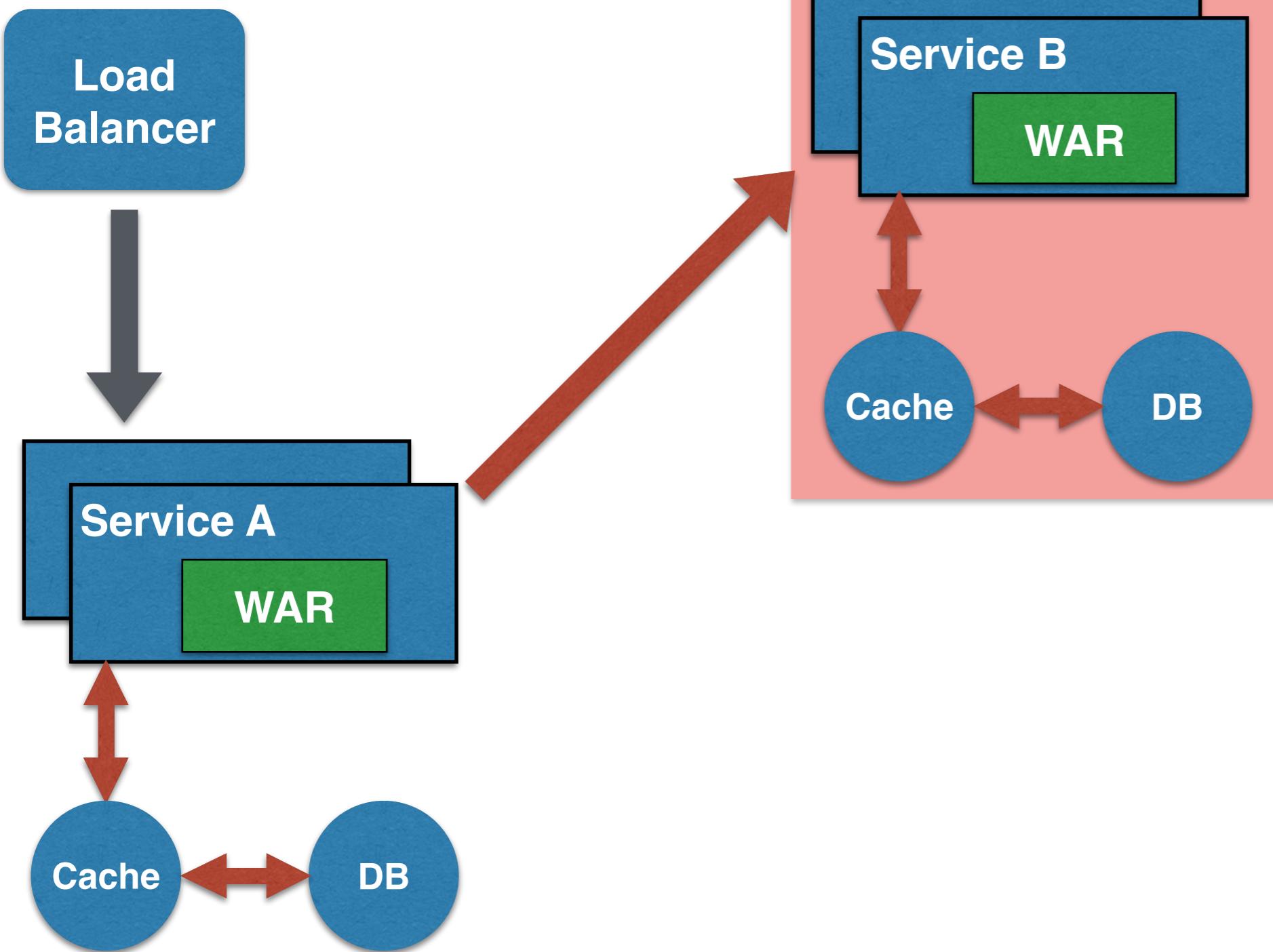
# Chained Pattern #2



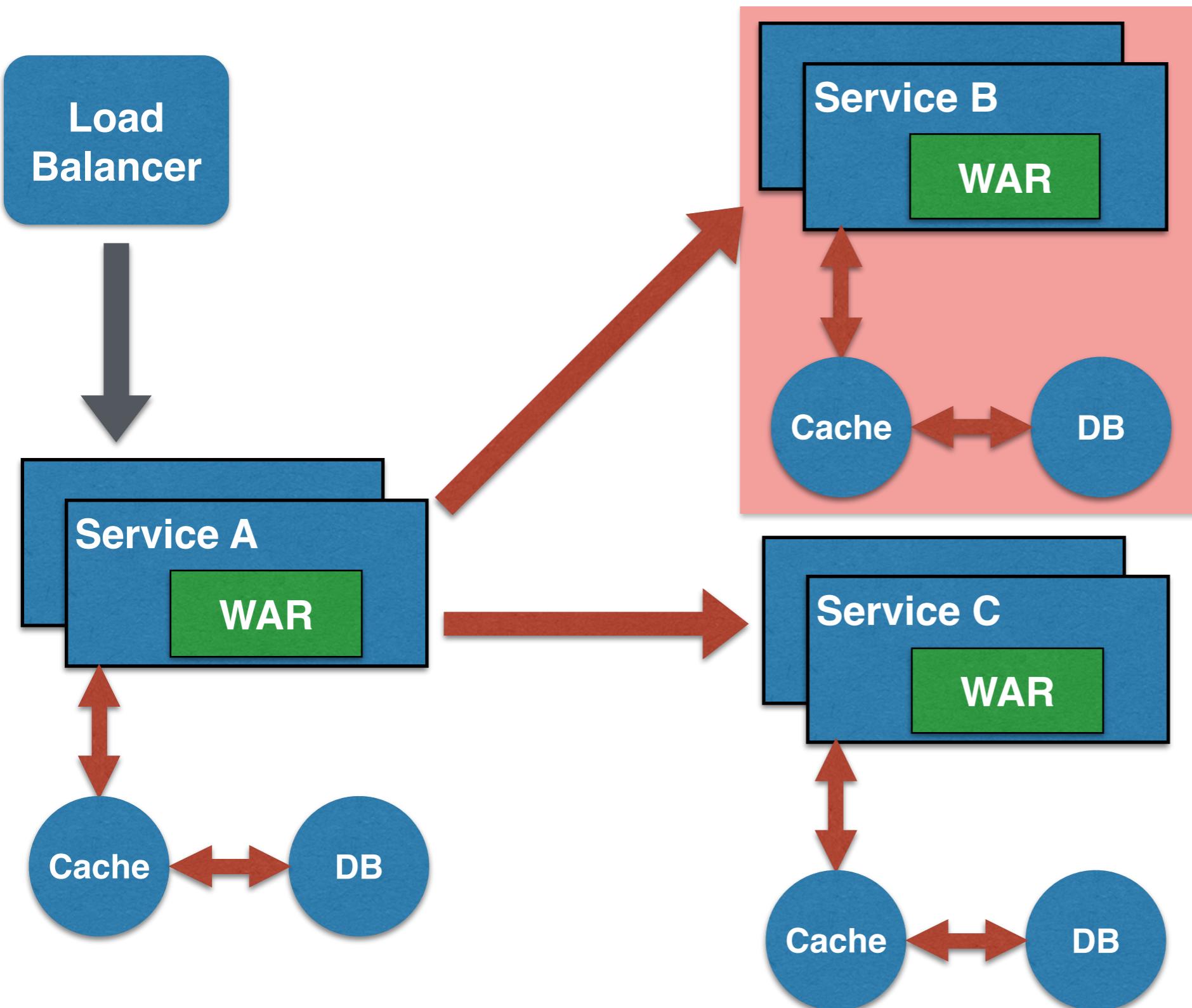
# Branch Pattern #3



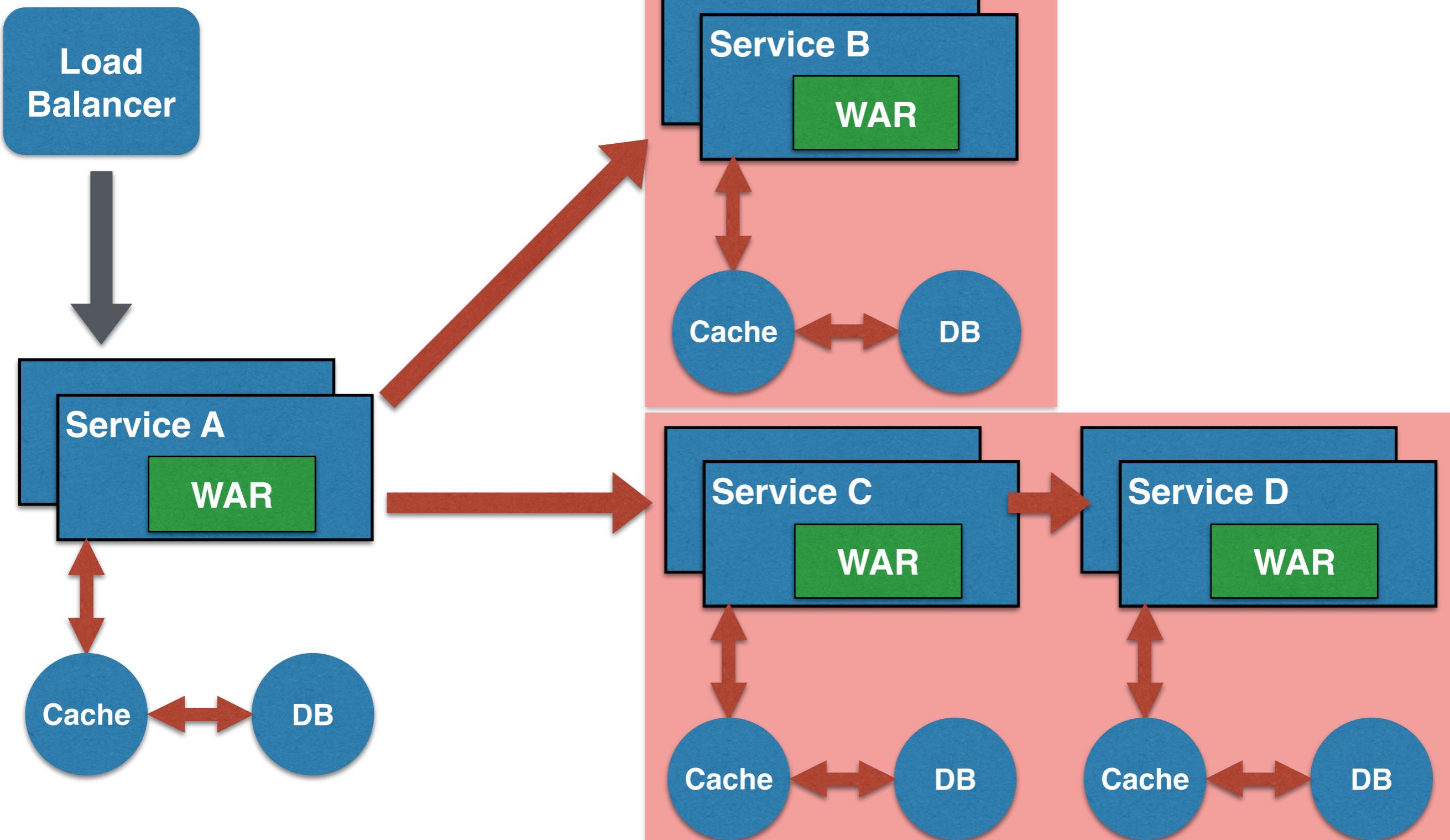
# Branch Pattern #3



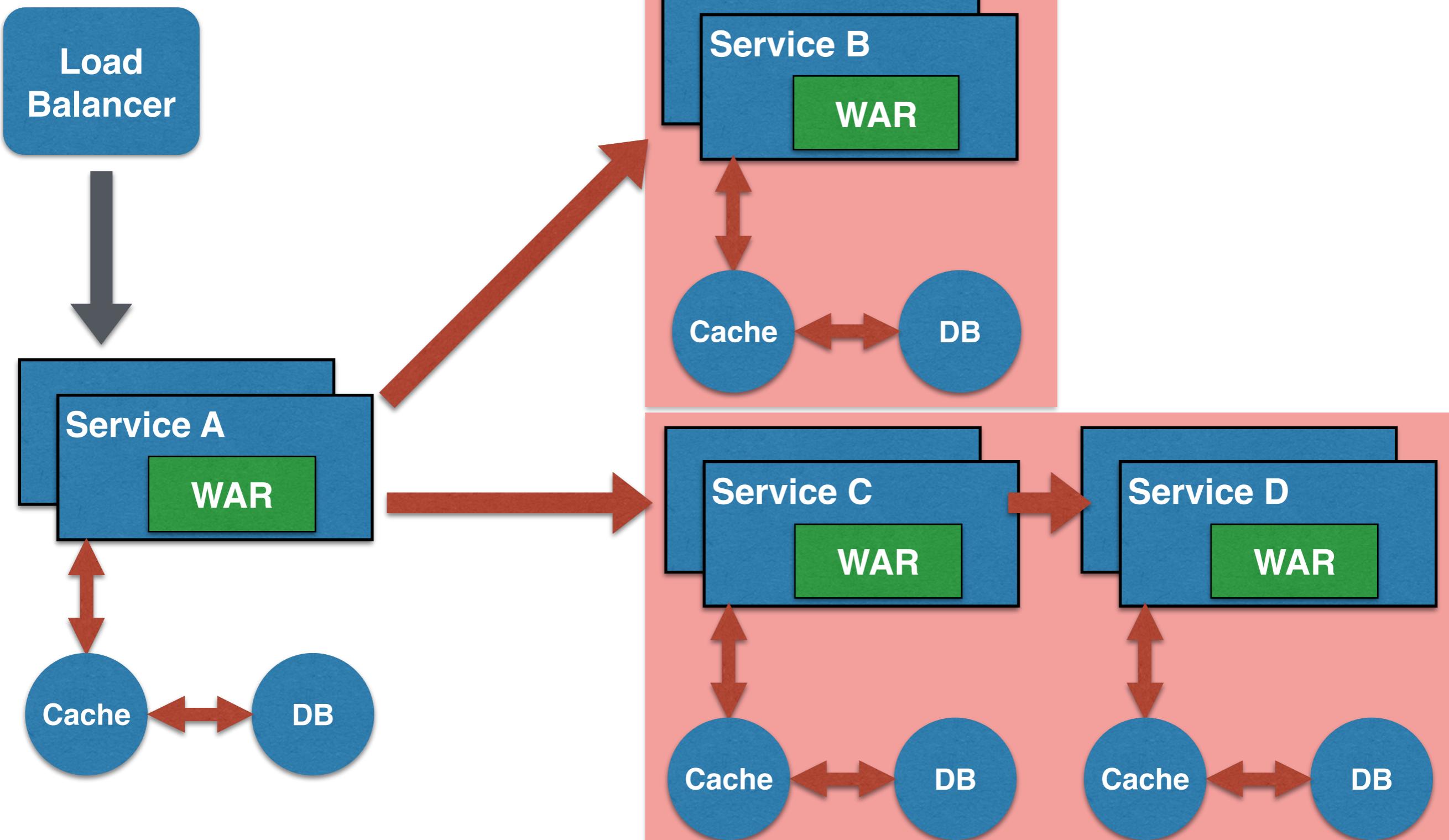
# Branch Pattern #3



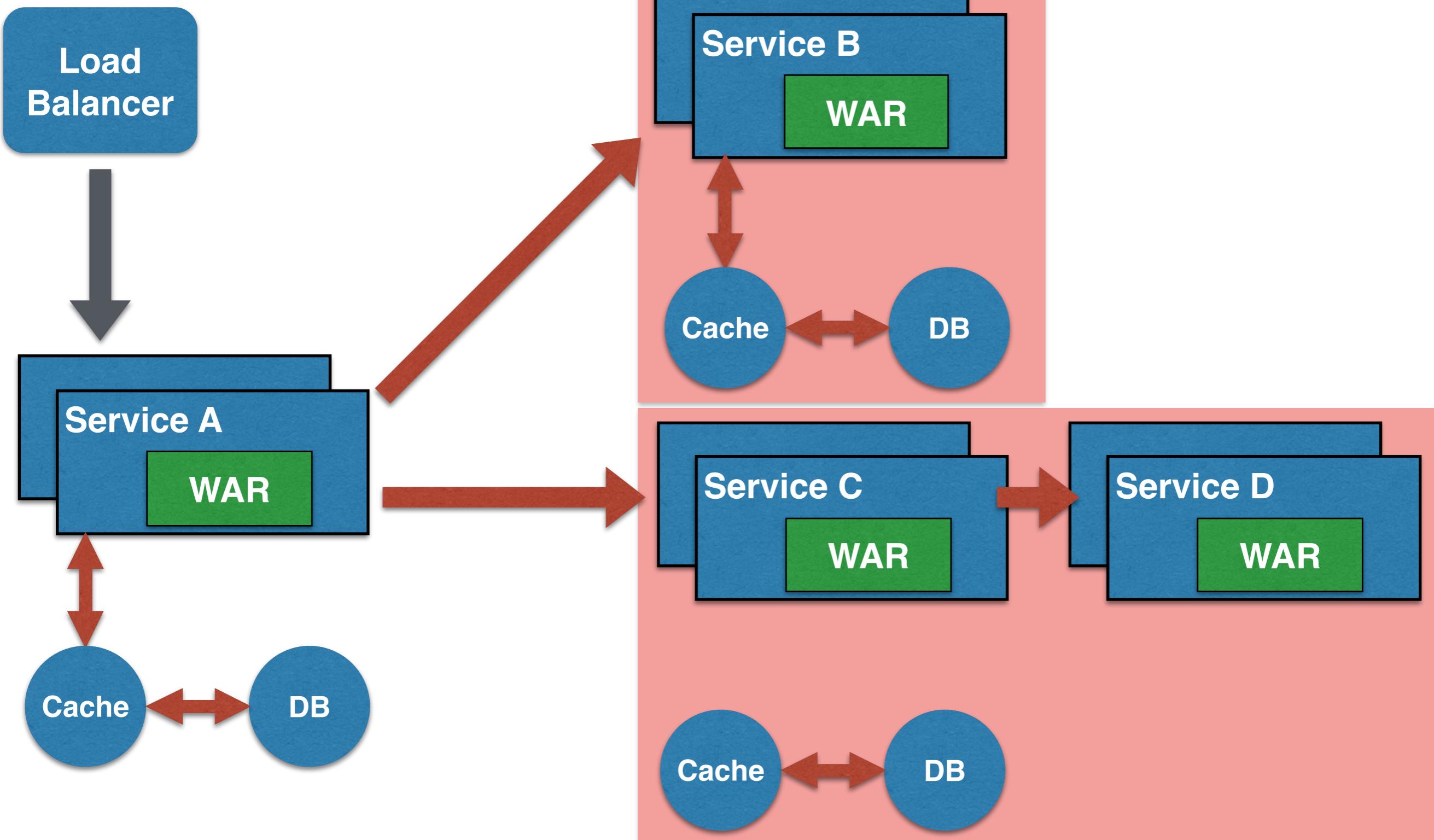
# Branch Pattern #3



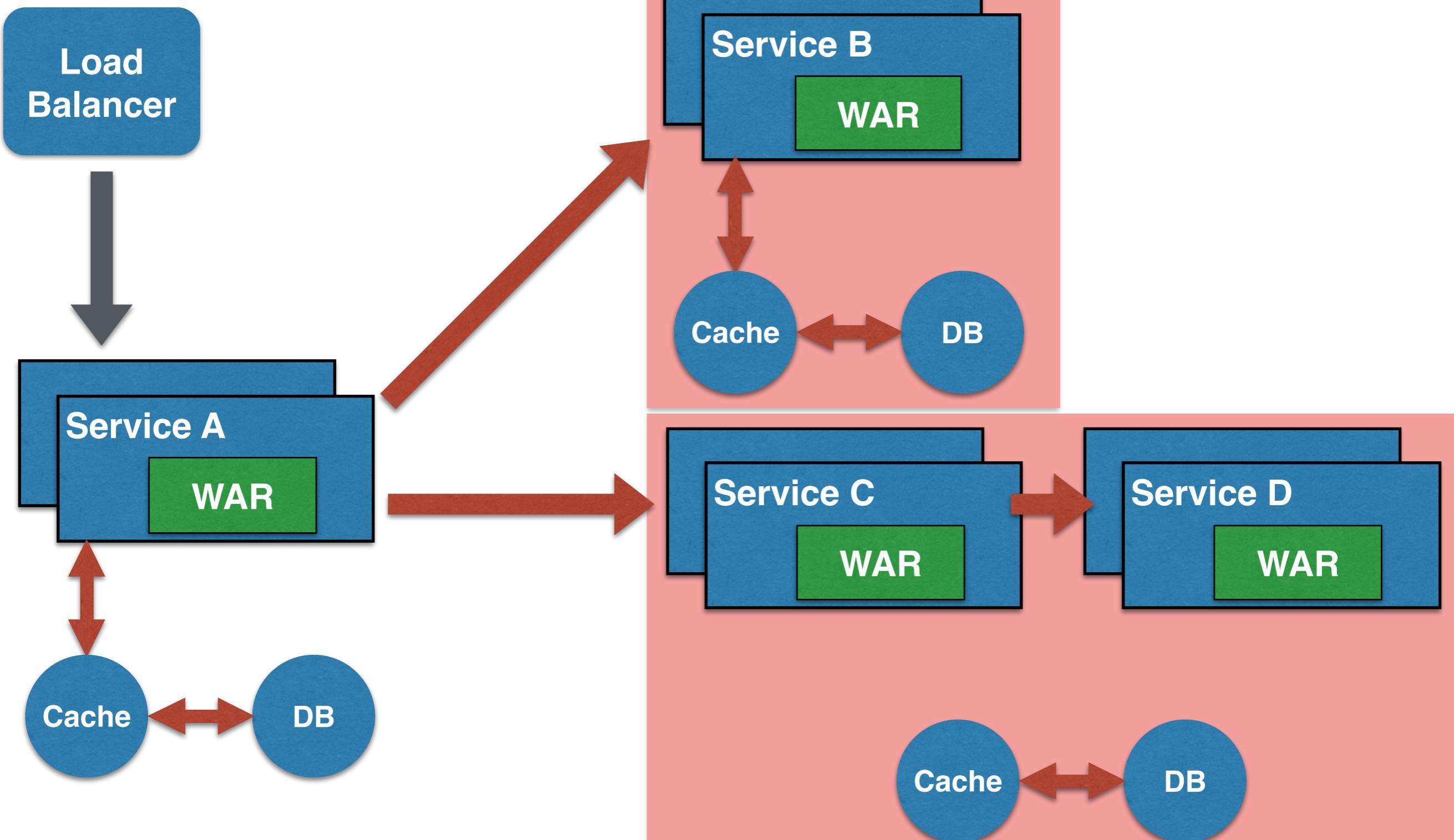
# Shared Resources #4



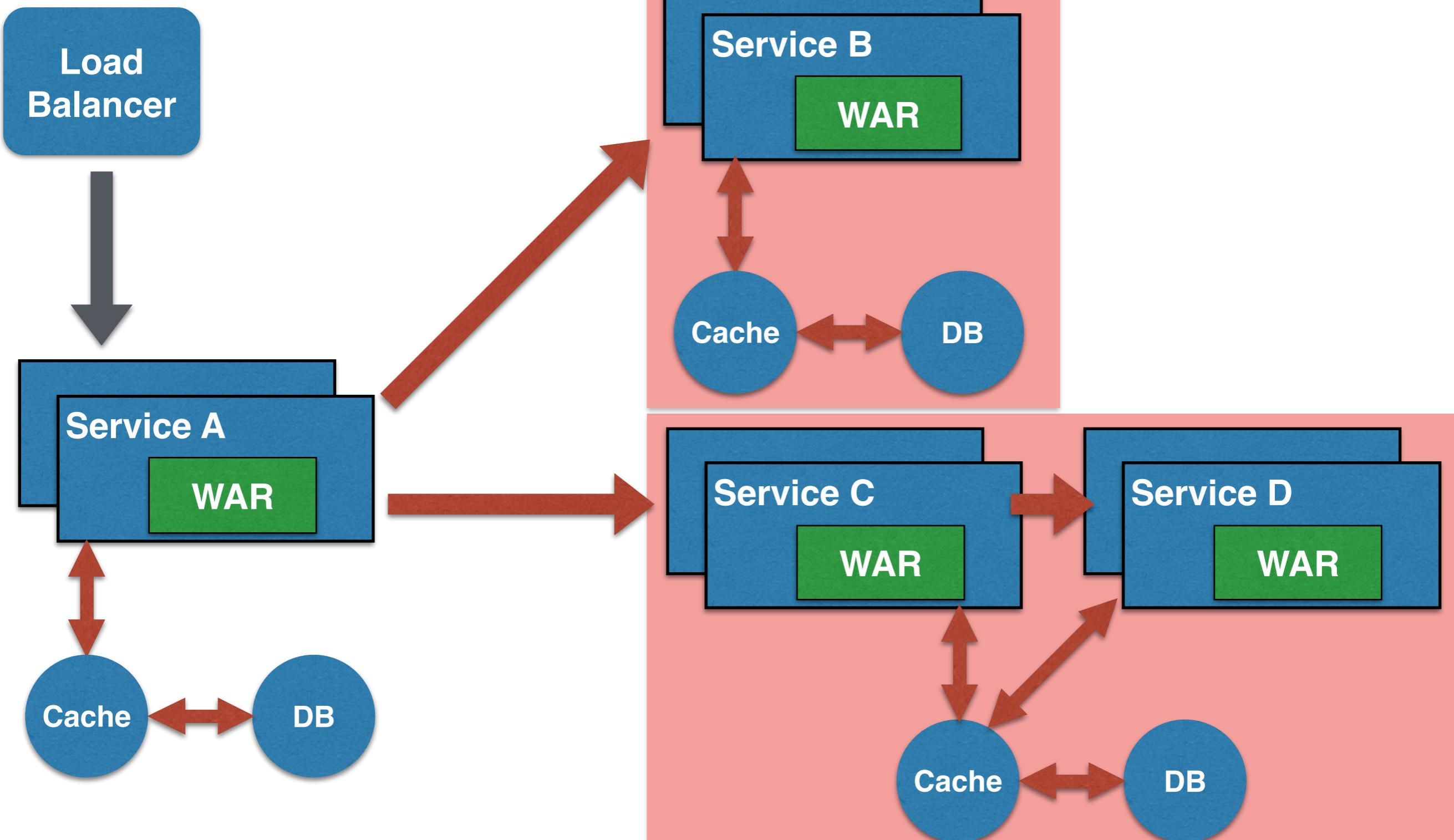
# Shared Resources #4



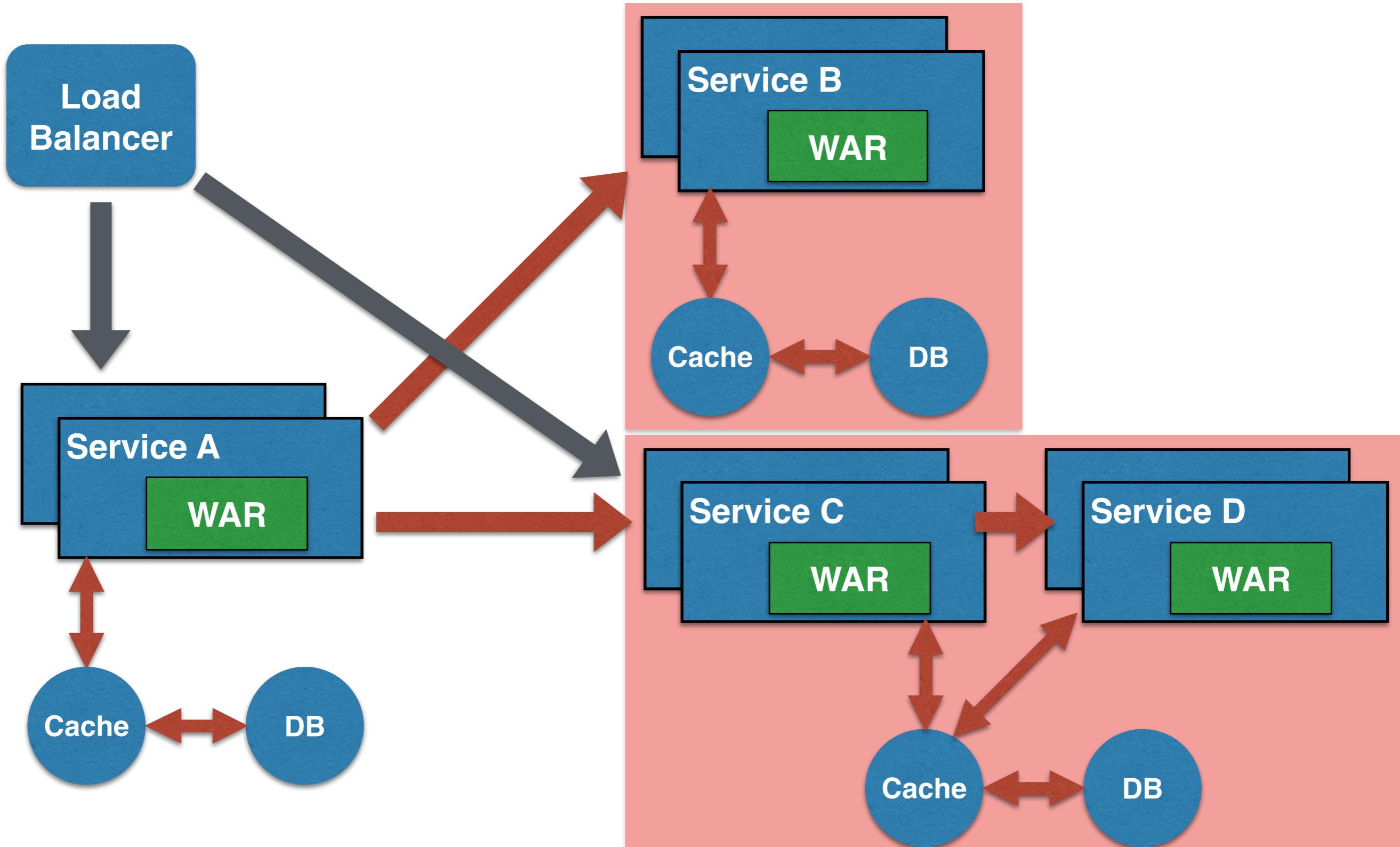
# Shared Resources #4



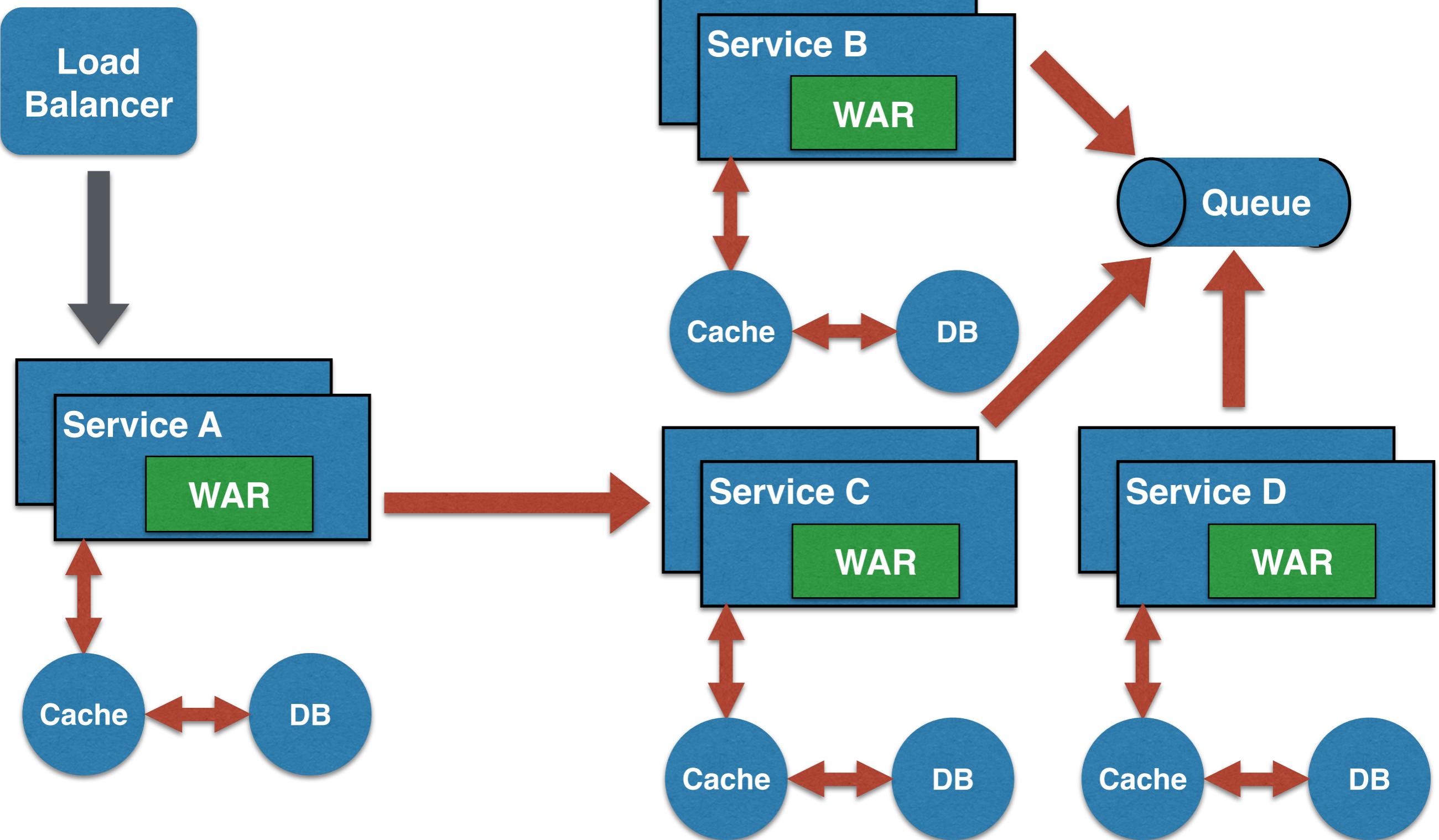
# Shared Resources #4



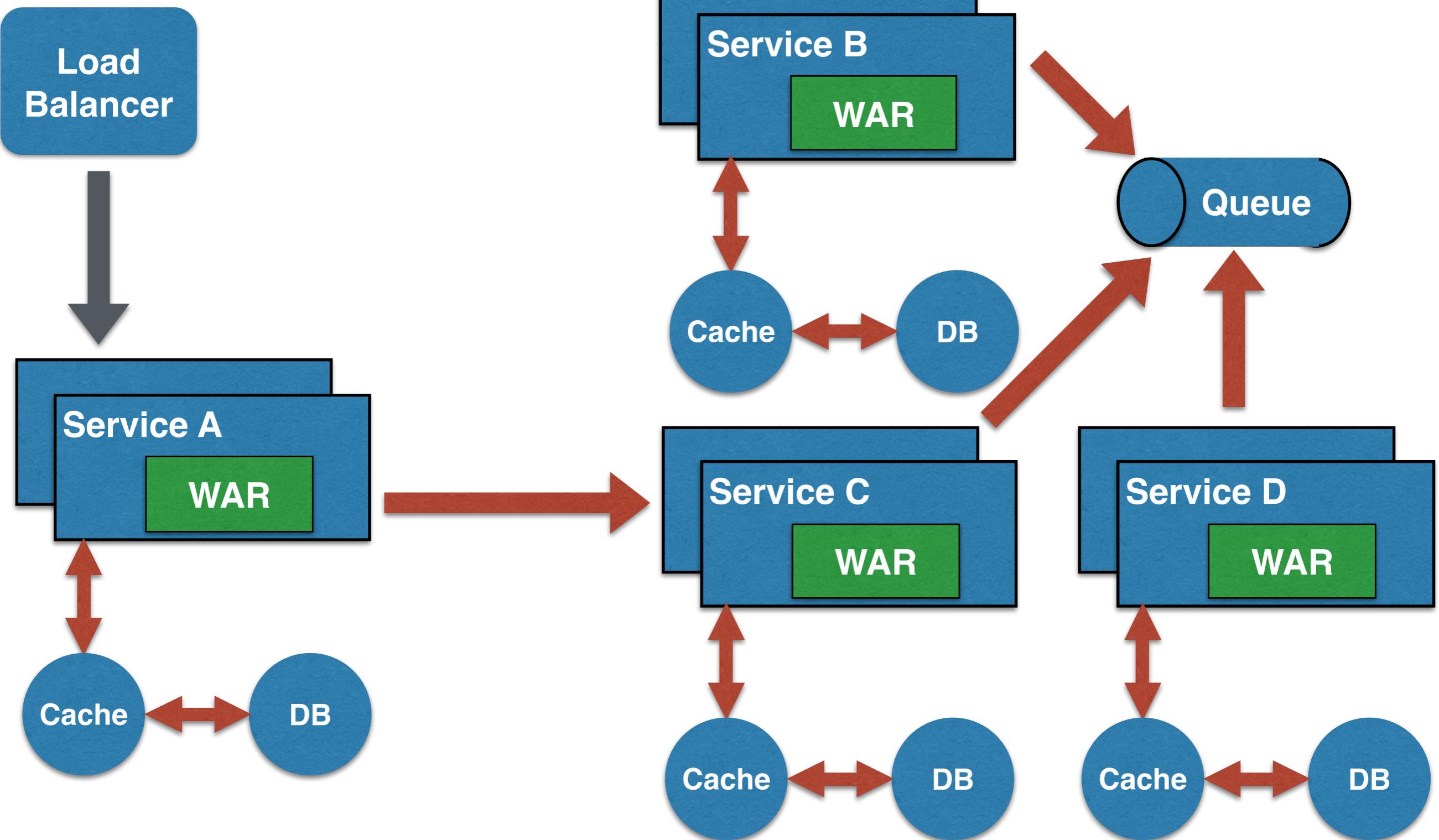
# Shared Resources #4



# Async Messaging #5



# Async Messaging #5



# Decentralized data

# Decentralized data

- Each service has its own schema, loose coupling

# Decentralized data

- Each service has its own schema, loose coupling
- Polyglot persistence

# Decentralized data

- Each service has its own schema, loose coupling
- Polyglot persistence
- Distributed transactions reduces system availability

# Decentralized data

- Each service has its own schema, loose coupling
- Polyglot persistence
- Distributed transactions reduces system availability
- Event-driven asynchronous updates

# Decentralized data

- Each service has its own schema, loose coupling
- Polyglot persistence
- Distributed transactions reduces system availability
- Event-driven asynchronous updates
  - Simplifies development

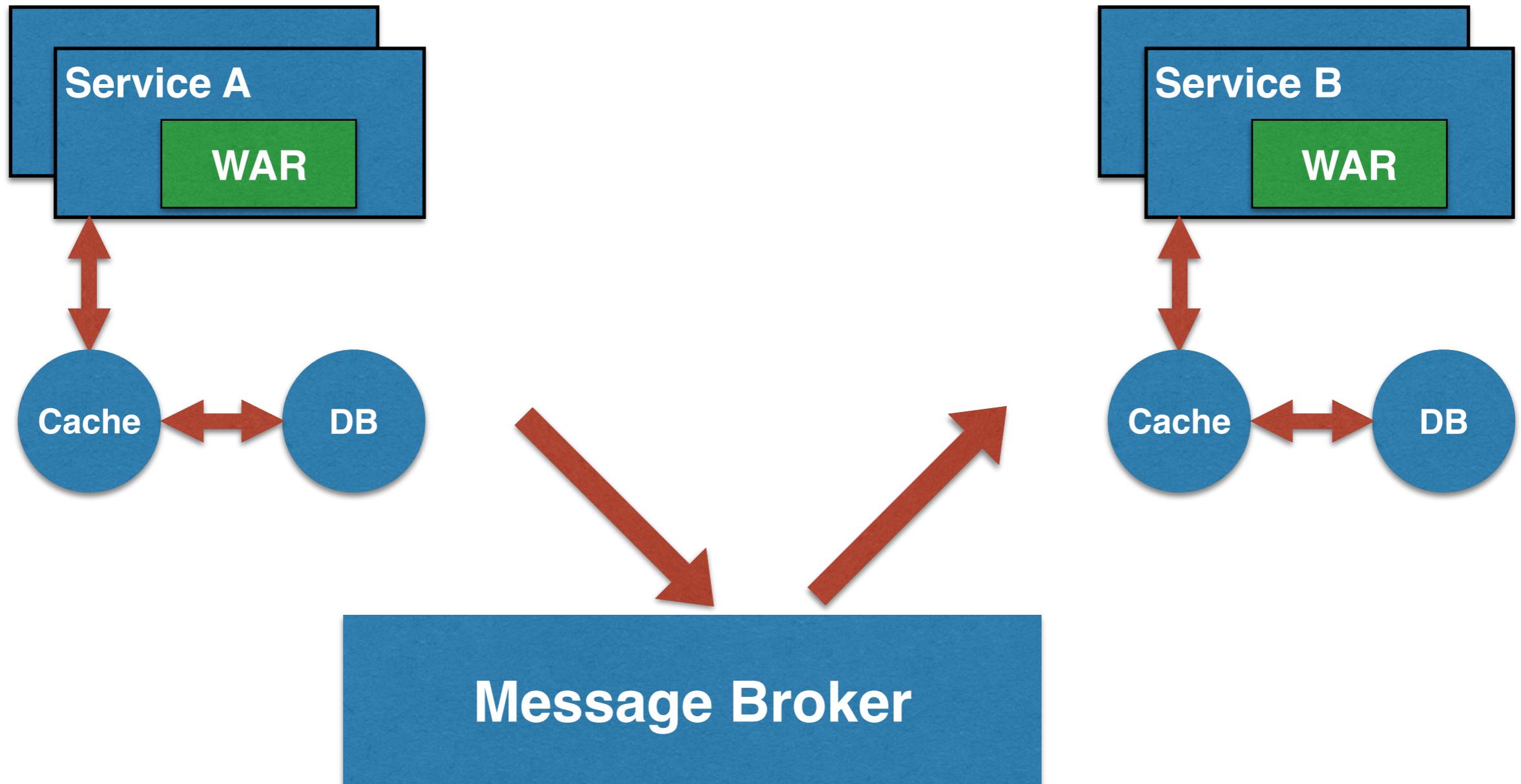
# Decentralized data

- Each service has its own schema, loose coupling
- Polyglot persistence
- Distributed transactions reduces system availability
- Event-driven asynchronous updates
  - Simplifies development
  - Availability over consistency

# Decentralized data

- Each service has its own schema, loose coupling
- Polyglot persistence
- Distributed transactions reduces system availability
- Event-driven asynchronous updates
  - Simplifies development
  - Availability over consistency
  - Implement compensating transactions

# Event-driven async updates



# Advantages of μservices

# Advantages of μservices

- Easier to develop, understand, maintain

# Advantages of μservices

- Easier to develop, understand, maintain
- Starts faster than a monolith, speeds up deployments

# Advantages of μservices

- Easier to develop, understand, maintain
- Starts faster than a monolith, speeds up deployments
- Local change can easily deployed, makes CD feasible

# Advantages of μservices

- Easier to develop, understand, maintain
- Starts faster than a monolith, speeds up deployments
- Local change can easily deployed, makes CD feasible
- Each service can scale on X- and Y-axis

# Advantages of μservices

- Easier to develop, understand, maintain
- Starts faster than a monolith, speeds up deployments
- Local change can easily deployed, makes CD feasible
- Each service can scale on X- and Y-axis
- Improves fault isolation

# Advantages of μservices

- Easier to develop, understand, maintain
- Starts faster than a monolith, speeds up deployments
- Local change can easily deployed, makes CD feasible
- Each service can scale on X- and Y-axis
- Improves fault isolation
- Eliminates any long-term commitment to a technology stack

# Advantages of μservices

- Easier to develop, understand, maintain
- Starts faster than a monolith, speeds up deployments
- Local change can easily deployed, makes CD feasible
- Each service can scale on X- and Y-axis
- Improves fault isolation
- Eliminates any long-term commitment to a technology stack
- Freedom of choice of technology, tools, frameworks

# Drawbacks of μservices

# Drawbacks of $\mu$ services

- Additional complexity of distributed systems

# Drawbacks of μservices

- Additional complexity of distributed systems
- Significant operational complexity, need high-level of automation

# Drawbacks of μservices

- Additional complexity of distributed systems
- Significant operational complexity, need high-level of automation
- Rollout plan to coordinate deployments

# Drawbacks of μservices

- Additional complexity of distributed systems
- Significant operational complexity, need high-level of automation
- Rollout plan to coordinate deployments
- Slower ROI, to begin with

Docker and Kubernetes are well suited to create a  $\mu$ services-based application.

# References

- [github.com/arun-gupta/docker-tutorial](https://github.com/arun-gupta/docker-tutorial)
- [martinfowler.com/articles/microservices.html](https://martinfowler.com/articles/microservices.html)
- [www.infoq.com/articles/microservices-intro](https://www.infoq.com/articles/microservices-intro)

# Refactor your applications using Container and Microservices



Arun Gupta, @arungupta  
Red Hat