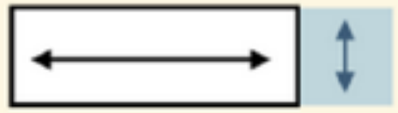
Refactor your applications using Container and Microservices

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

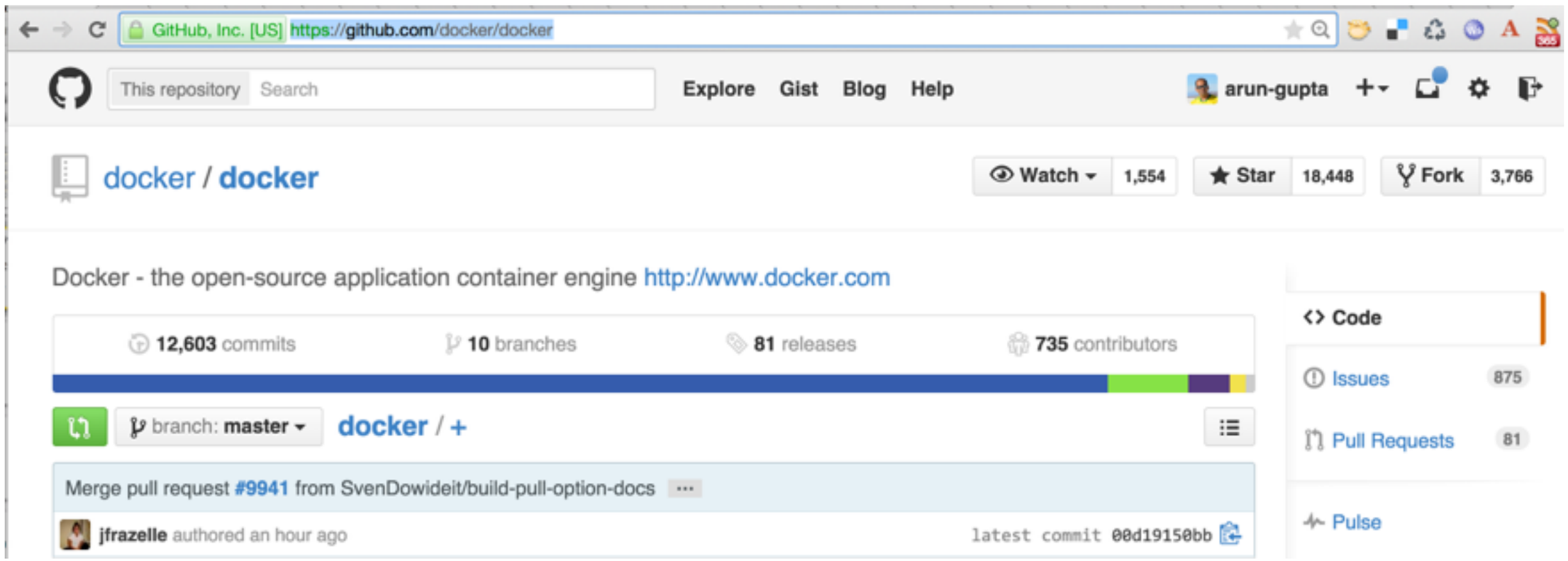
Site Constraints 	Large consumption of terminal space. Draft issues with larger containerships.
Capital Intensiveness 	Container handling infrastructures and equipment are important investments.
Stacking 	Complexity of arrangement of containers, both on the ground and on modes.
Repositioning 	Divergence 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.

http://people.hofstra.edu/geotrans/eng/ch3en/conc3en/table_advantageschallengescont.html

What is Docker?

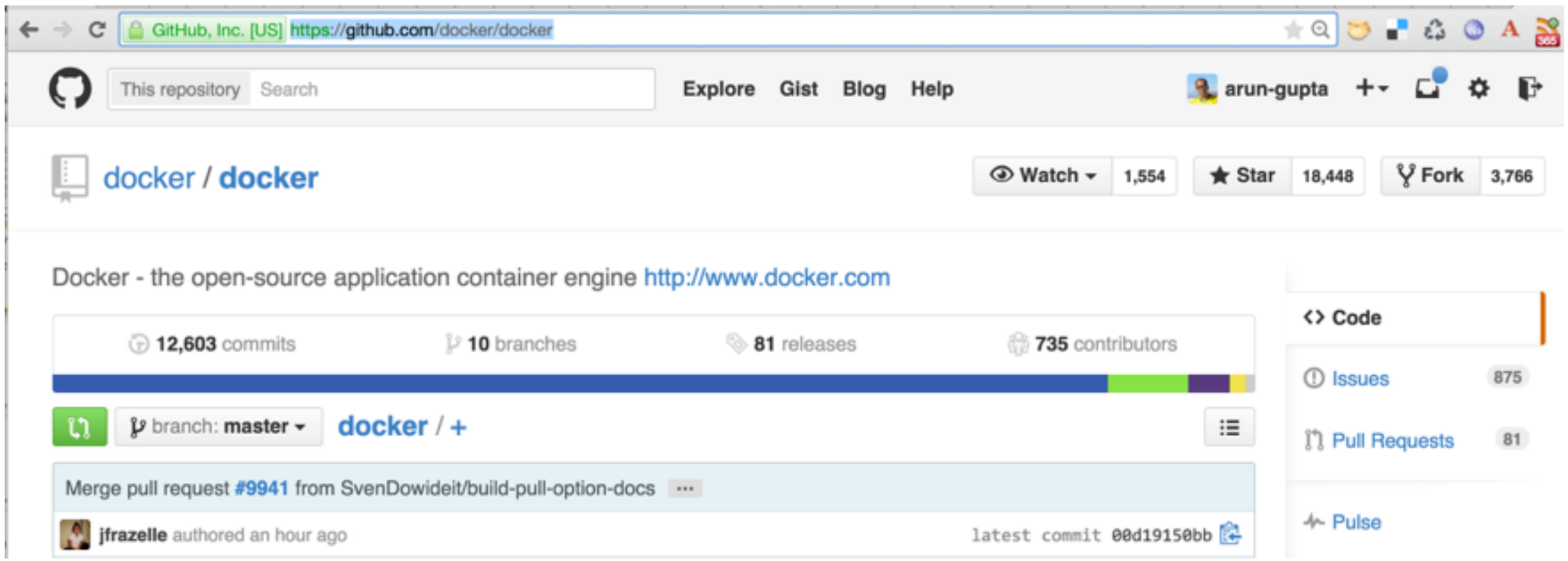
What is Docker?

- Open source project and company



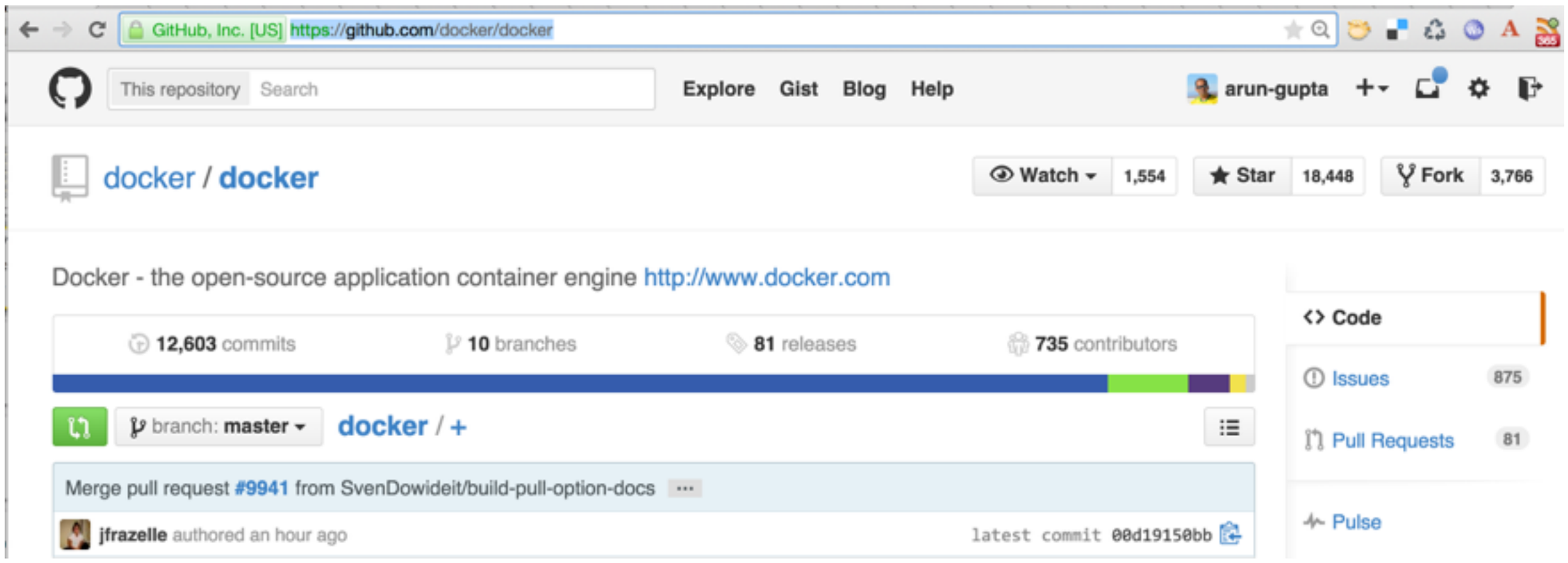
What is Docker?

- 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?

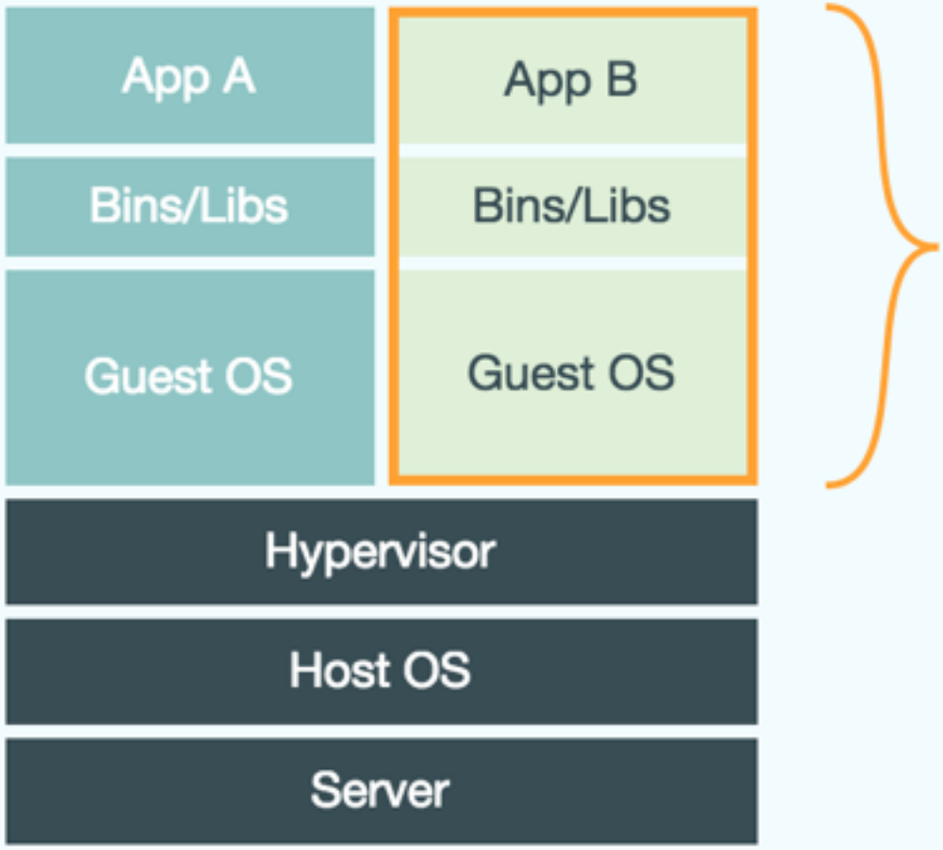
- Natively supported in Linux

Is it only Linux?

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

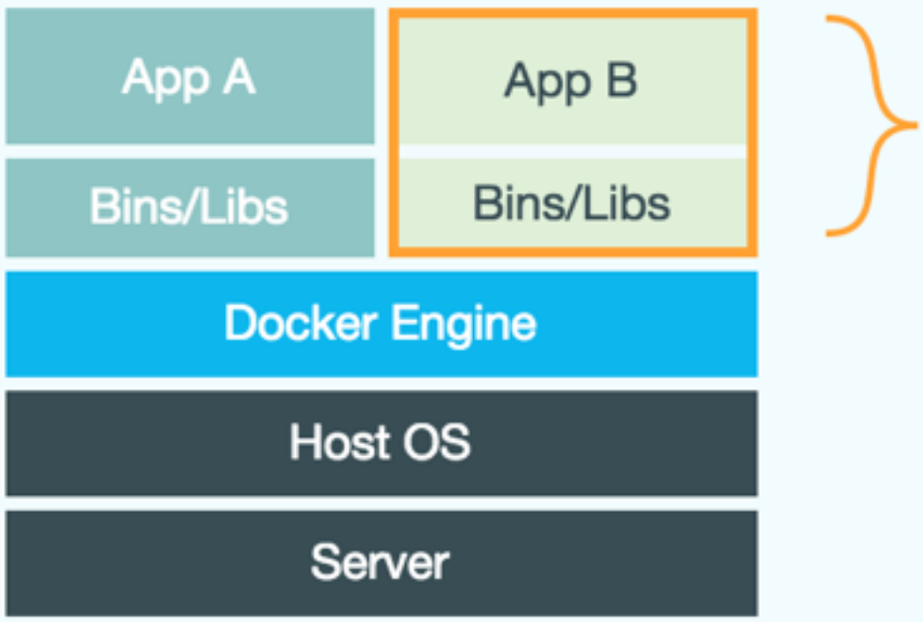
Is it only Linux?

- 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.

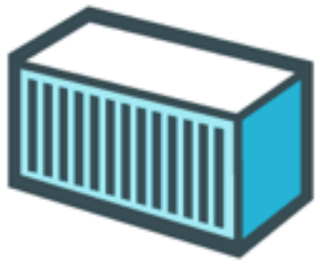


Docker

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.



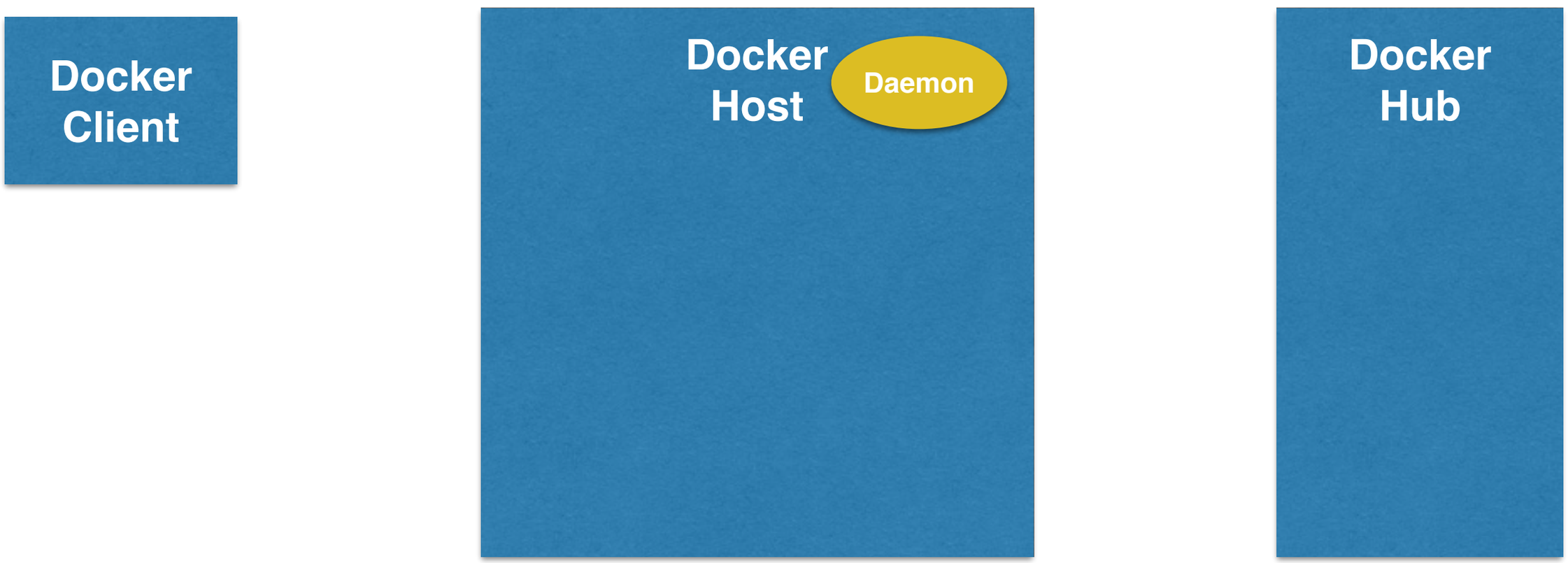
Ship

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

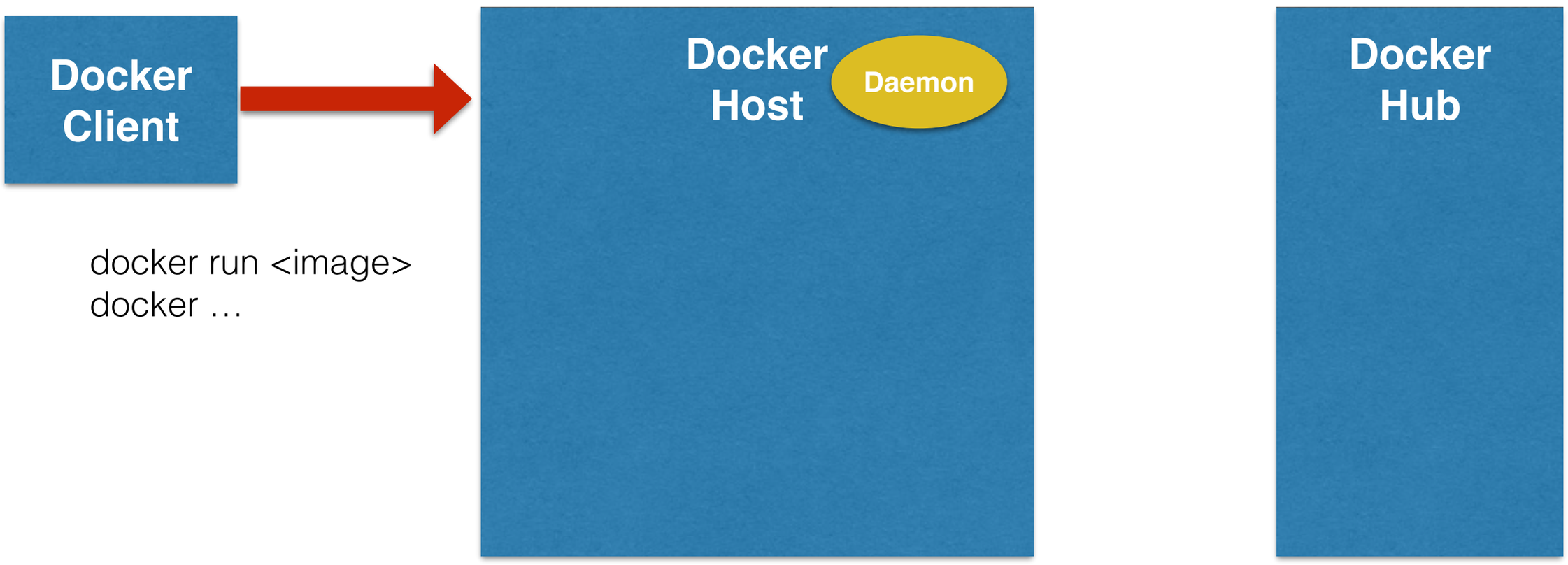
Run

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

Docker Workflow

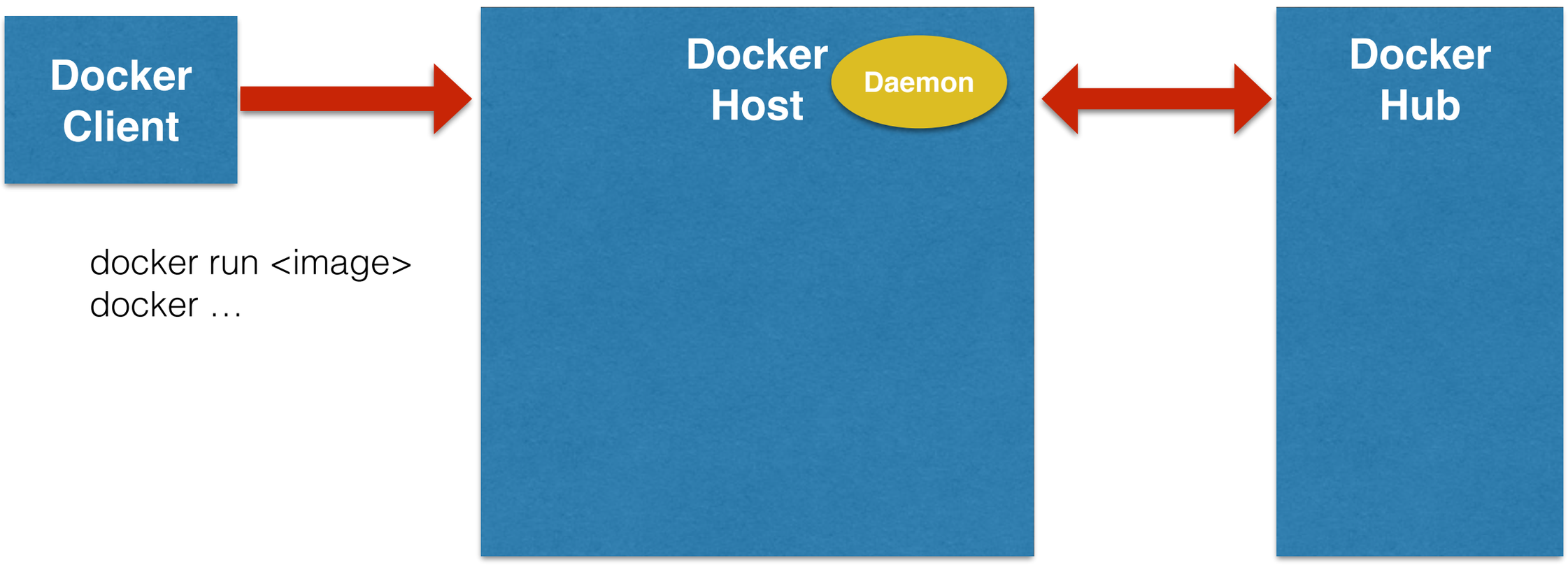


Docker Workflow



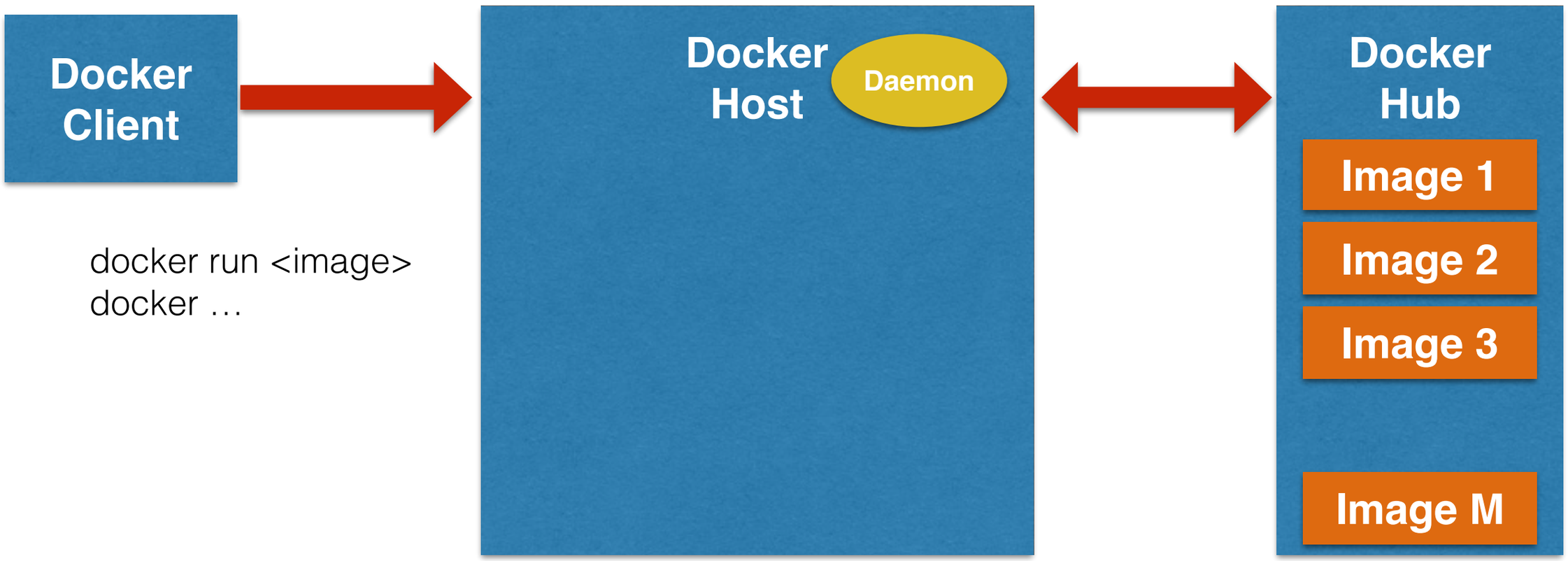
docker run <image>
docker …

Docker Workflow



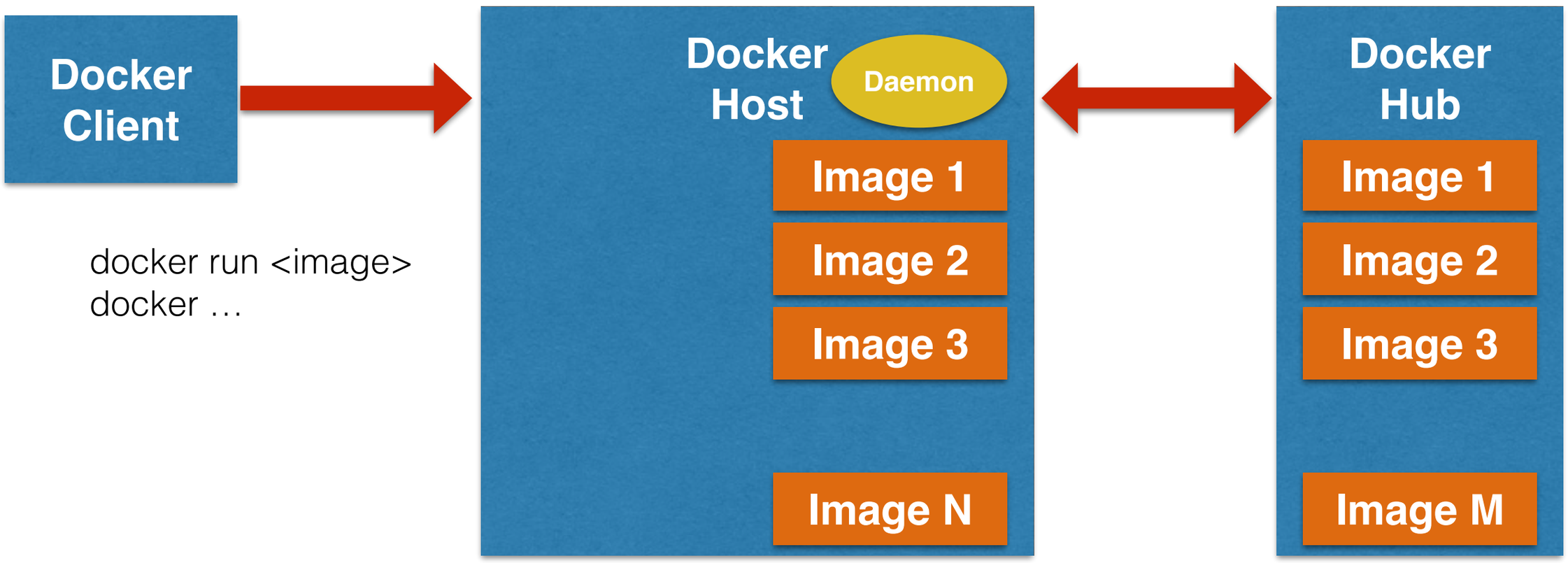
docker run <image>
docker …

Docker Workflow



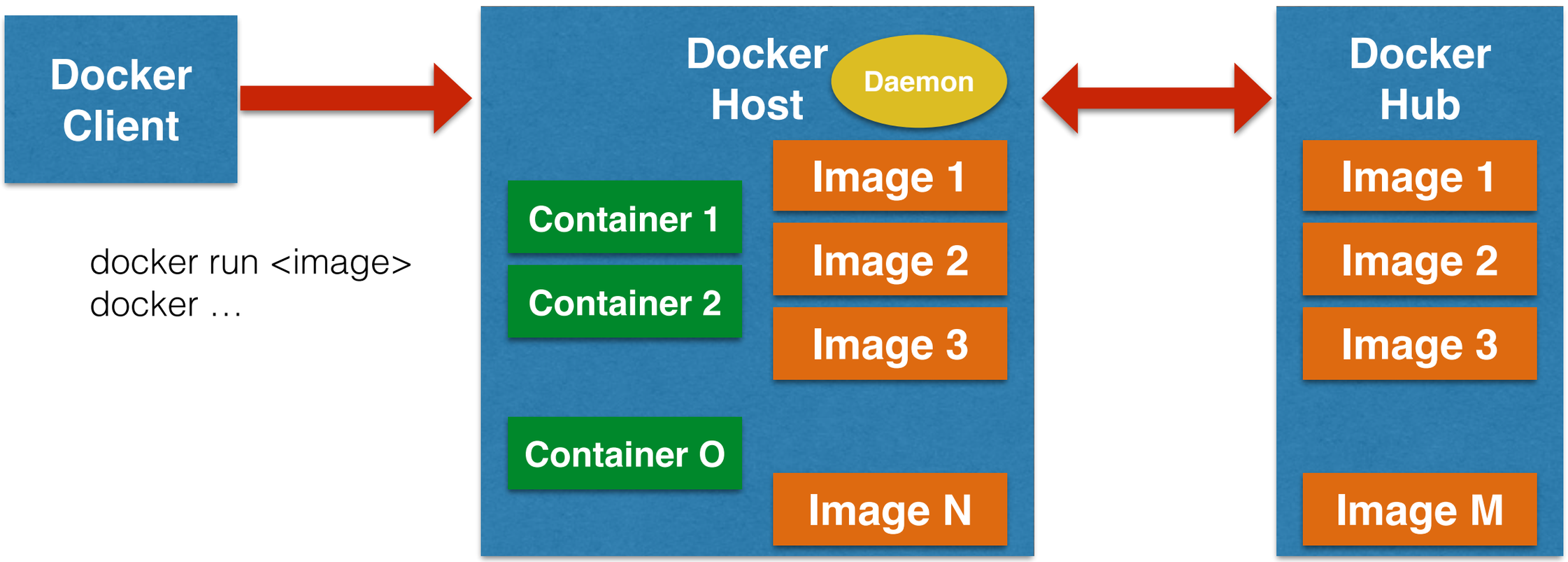
docker run <image>
docker …

Docker Workflow



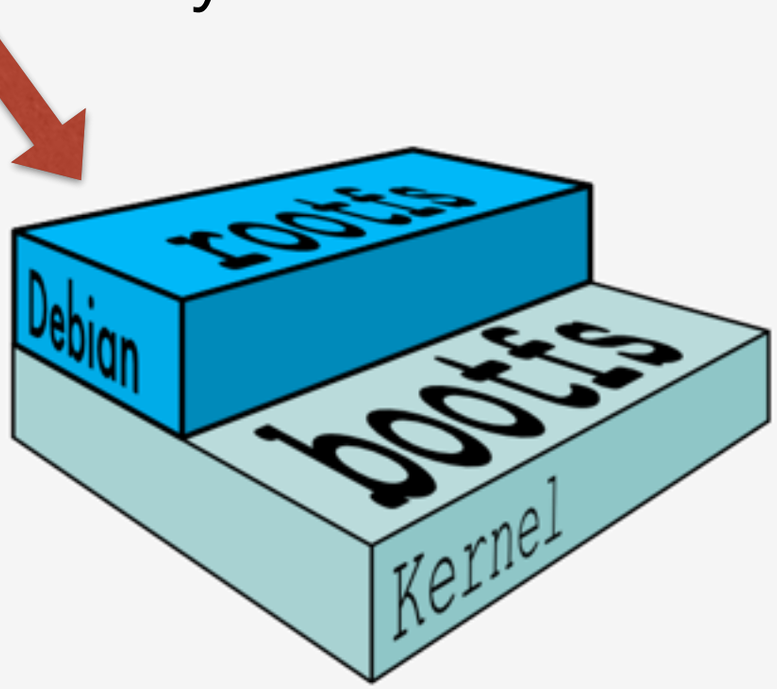
docker run <image>
docker …

Docker Workflow

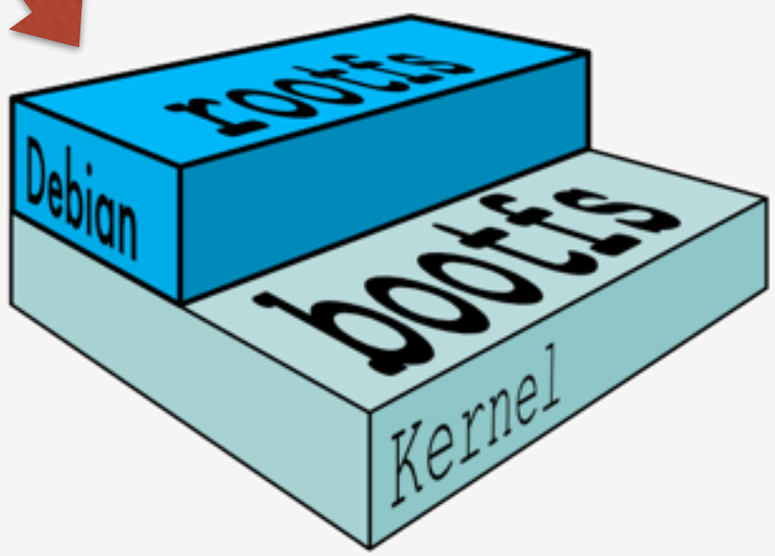


docker run <image>
docker …

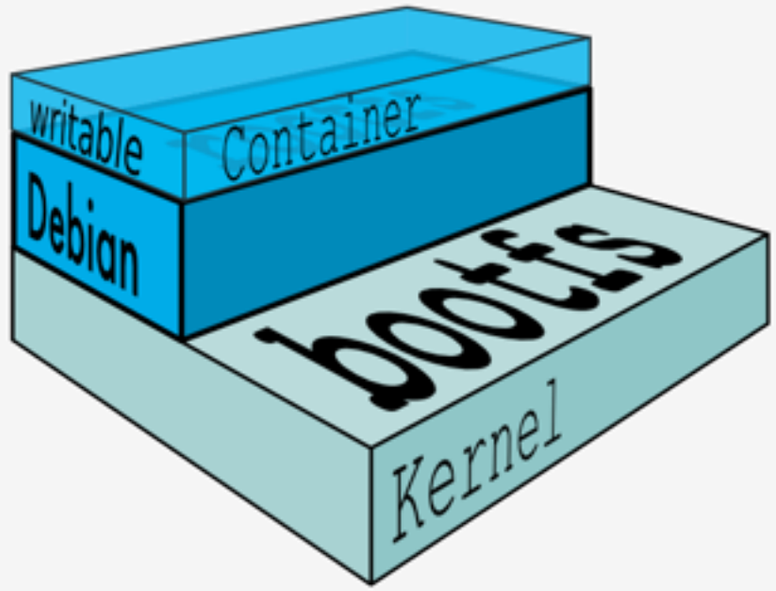
read-only



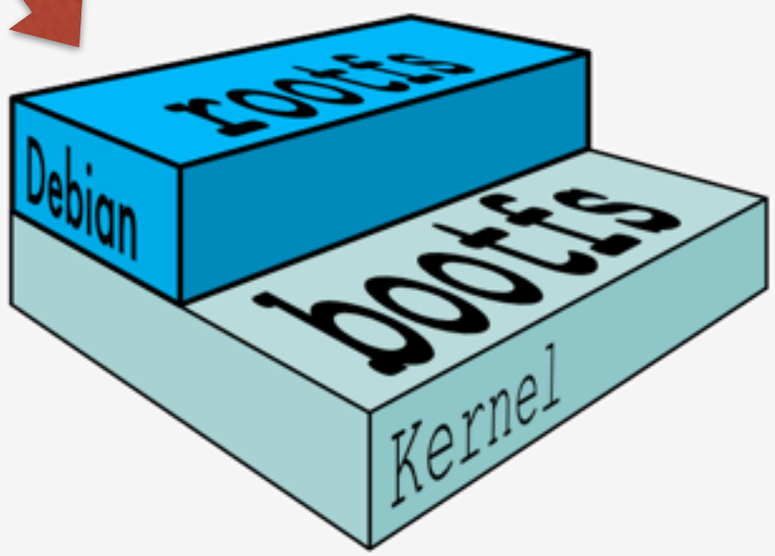
read-only



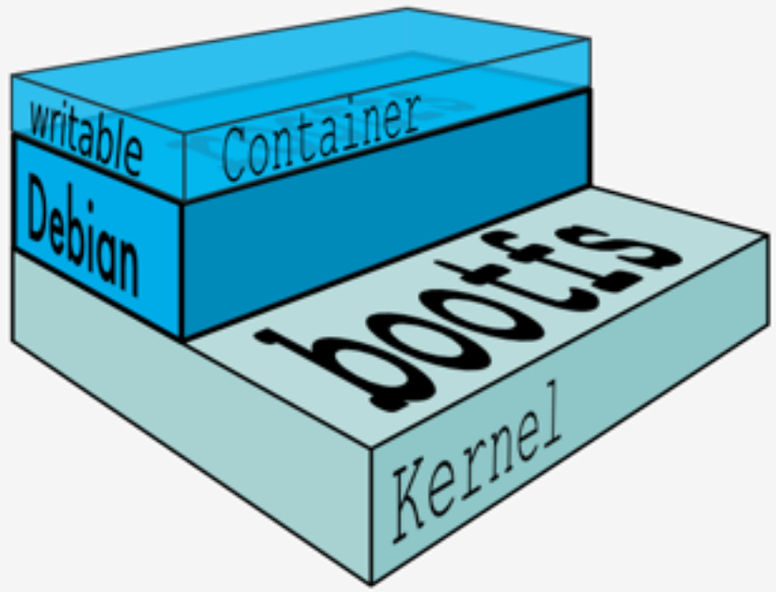
Union File System

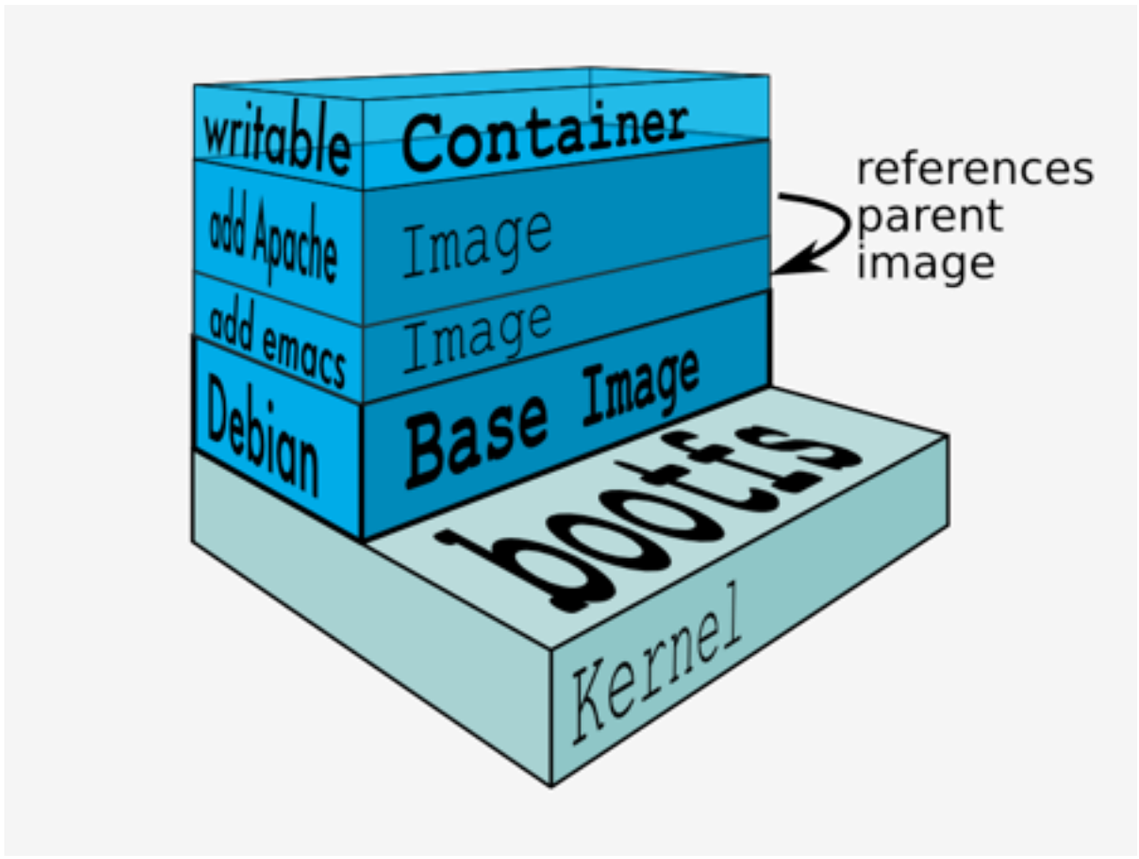


read-only



Union File System





Build

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

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

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

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


```
FROM fedora:latest

CMD echo “Hello world”
```

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


```
FROM fedora:latest

CMD echo “Hello world”
```

```
FROM jboss/wildlfy
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
```

- 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/wildfy
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
```


```
FROM centos
MAINTAINER Arun Gupta <arungupta@redhat.com>

# Execute system update
RUN yum -y update && yum clean all

# Install packages necessary to run EAP
RUN yum -y install xmlstarlet saxon augeas bsdtar unzip && yum clean all

# Create a user and group used to launch processes
# The user ID 1000 is the default for the first "regular" user on Fedora/RHEL,
# so there is a high chance that this ID will be equal to the current user
# making it easier to use volumes (no permission issues)
RUN groupadd -r jboss -g 1000 && useradd -u 1000 -r -g jboss -m -d /opt/jboss -s /sbin/nologin -c "JBoss user" jboss

# Set the working directory to jboss' user home directory
WORKDIR /opt/jboss

# User root user to install software
USER root

# Install necessary packages
RUN yum -y install java-1.7.0-openjdk-devel && yum clean all
#RUN yum -y install java-1.8.0-openjdk-devel && yum clean all

# Switch back to jboss user
USER jboss

# Set the JAVA_HOME variable to make it clear where Java is located
ENV JAVA_HOME /usr/lib/jvm/java

# Set the WILDFLY_VERSION env variable
ENV WILDFLY_VERSION 8.2.0.Final

# Add the WildFly distribution to /opt, and make wildfly the owner of the extracted tar content
# Make sure the distribution is available from a well-known place
RUN cd $HOME && curl -O http://download.jboss.org/wildfly/$WILDFLY_VERSION/wildfly-$WILDFLY_VERSION.zip && unzip wildfly-$WILDFLY

# Set the JBOSS_HOME env variable
ENV JBOSS_HOME /opt/jboss/wildfly

# Expose the ports we're interested in
EXPOSE 8080 9990

# Set the default command to run on boot
# This will boot WildFly in the standalone mode and bind to all interface
CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-c", "standalone-full.xml", "-b", "0.0.0.0"]
```

https://registry.hub.docker.com/repos/jboss/

docker

What is Docker? Use Cases Try It! Browse Install & Docs Log In Sign Up

jboss

Repositories

Show: All Sort by: Last Updated

Filter by name...

jboss/forge 8 hours ago — 1 — 17

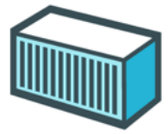
jboss/liveoak-server 4 days ago
Backend as a Service — 3 — 140

jboss/switchyard-wildfly a month ago — 0 — 49

jboss/keycloak a month ago — 0 — 540

jboss/keycloak-adapter-wildfly a month ago — 0 — 195

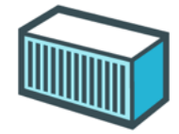
jboss/infinispan-server 2 months ago — 0 — 87



Ship

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

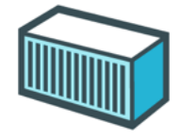
- Images shared using registry



Ship

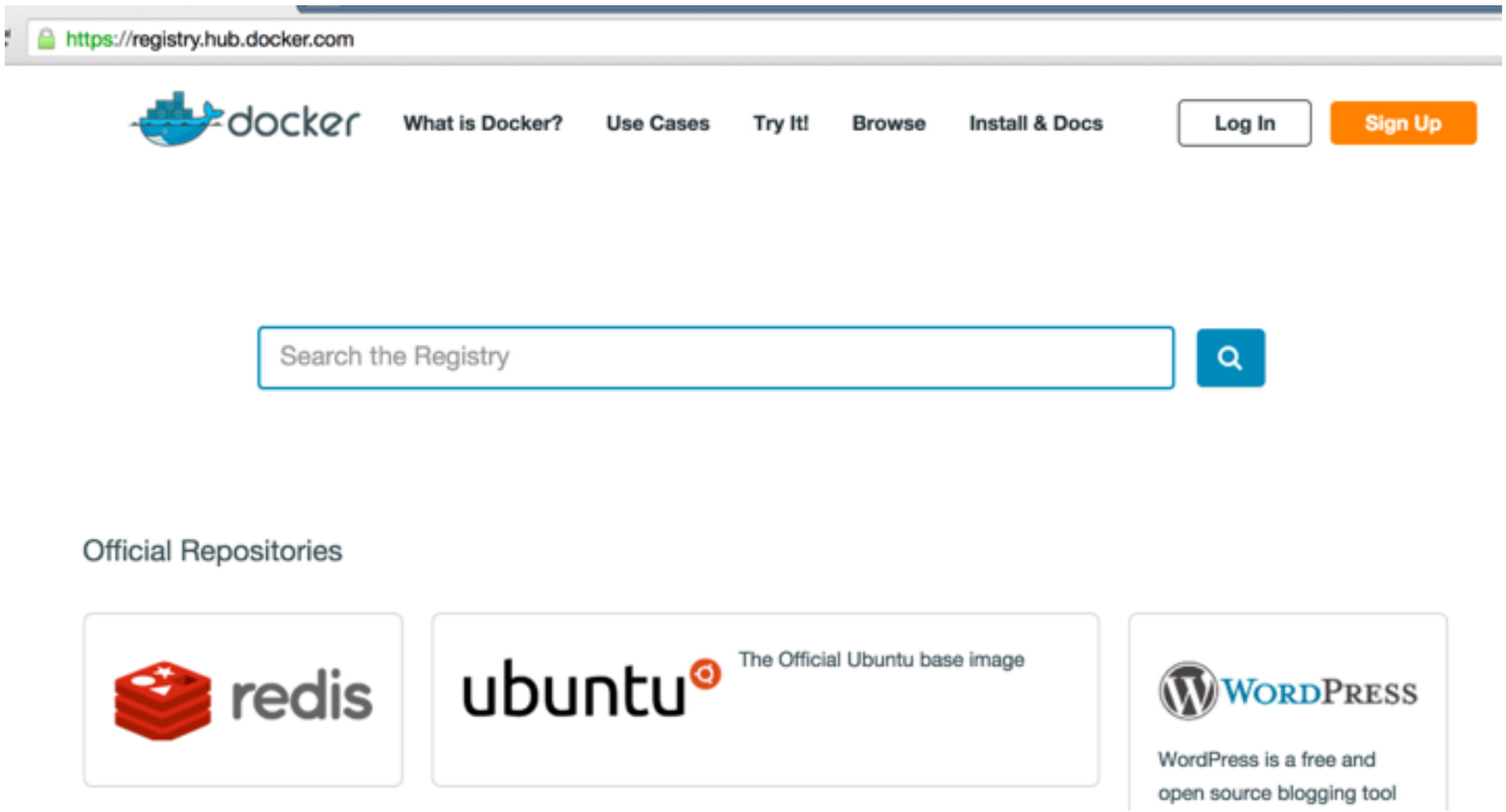
Ship the “Dockerized” app and dependencies anywhere - to QA, teammates, or the cloud - without breaking anything.

- Images shared using registry
- Docker Hub is public SaaS

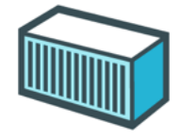


Ship

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

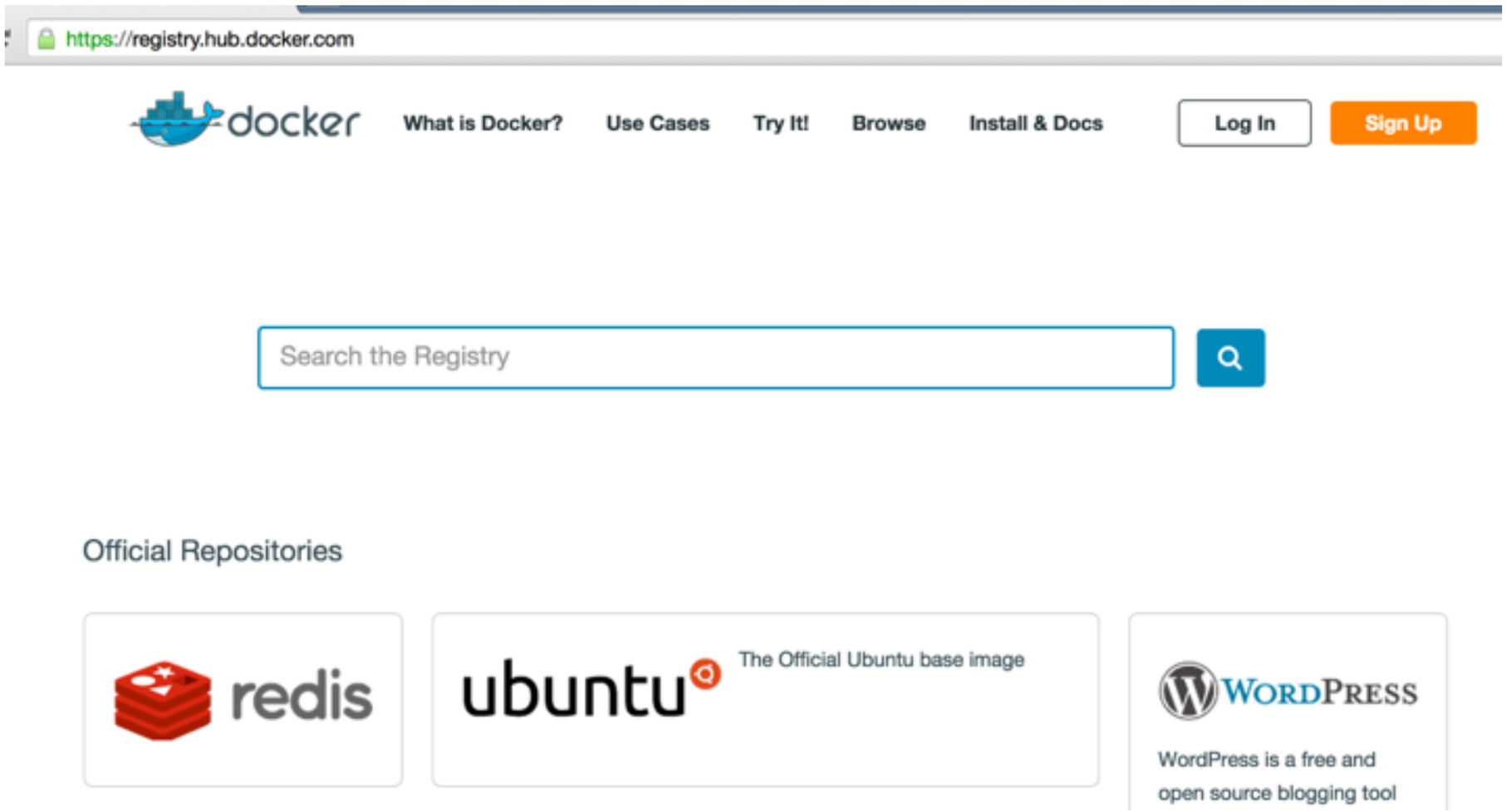


- Images shared using registry
- Docker Hub is public SaaS
- Private registries can be setup inside firewall

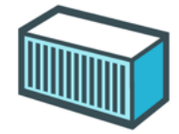


Ship

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

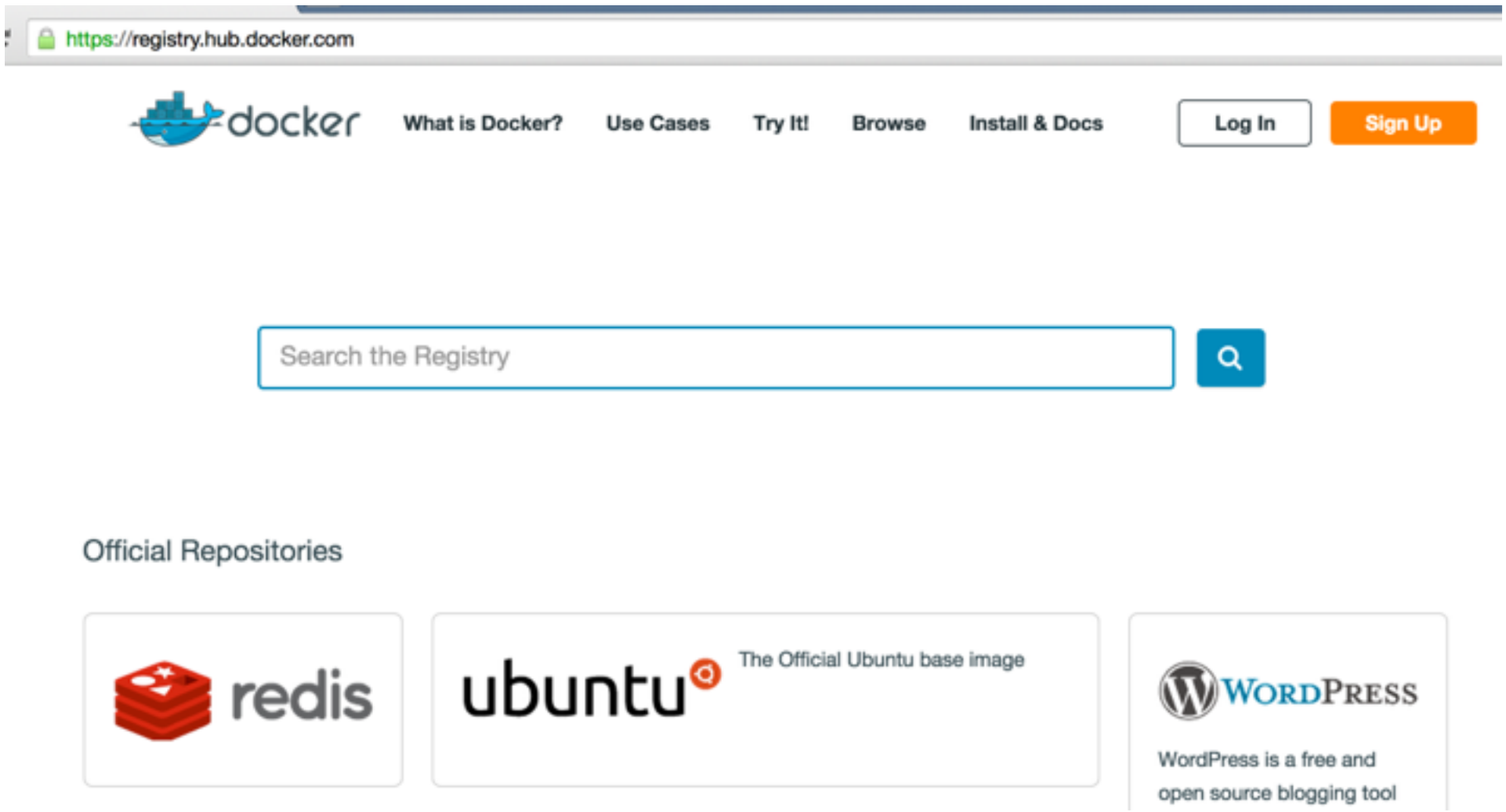


- Images shared using registry
- Docker Hub is public SaaS
- Private registries can be setup inside firewall
- `docker push` or `pull <IMAGE_ID>`



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.

- 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>`**

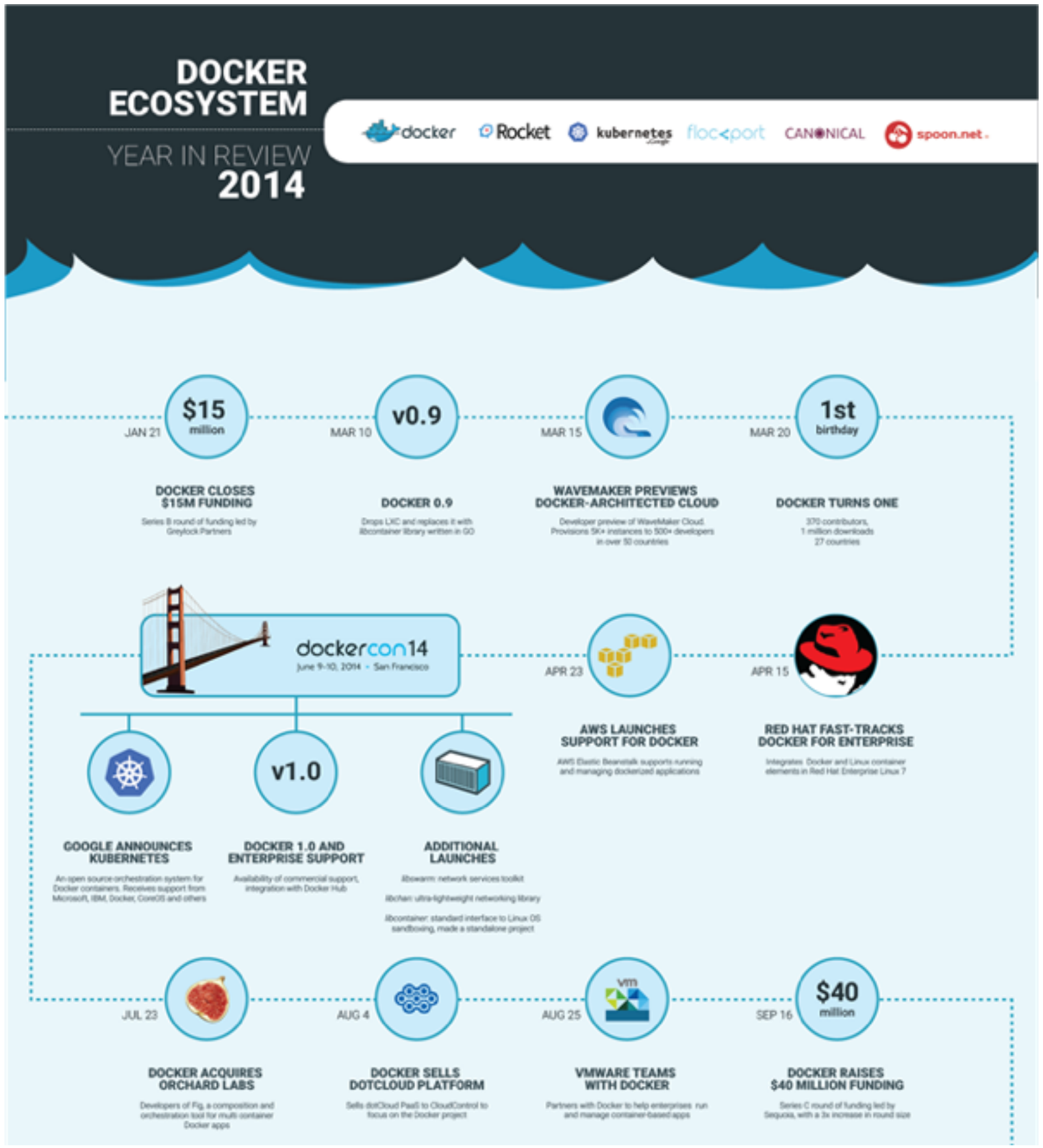
Run

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

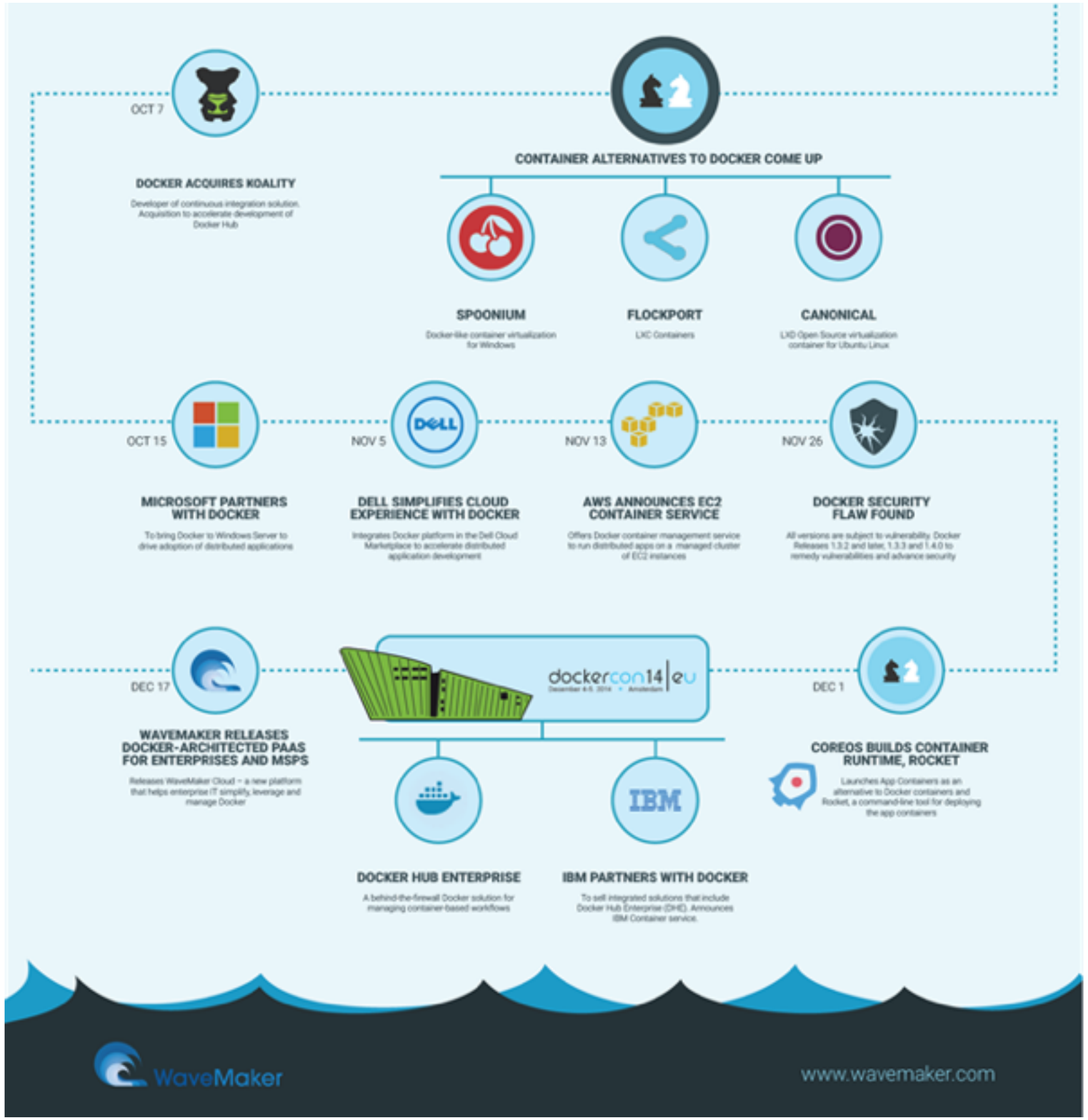
Docker commands

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

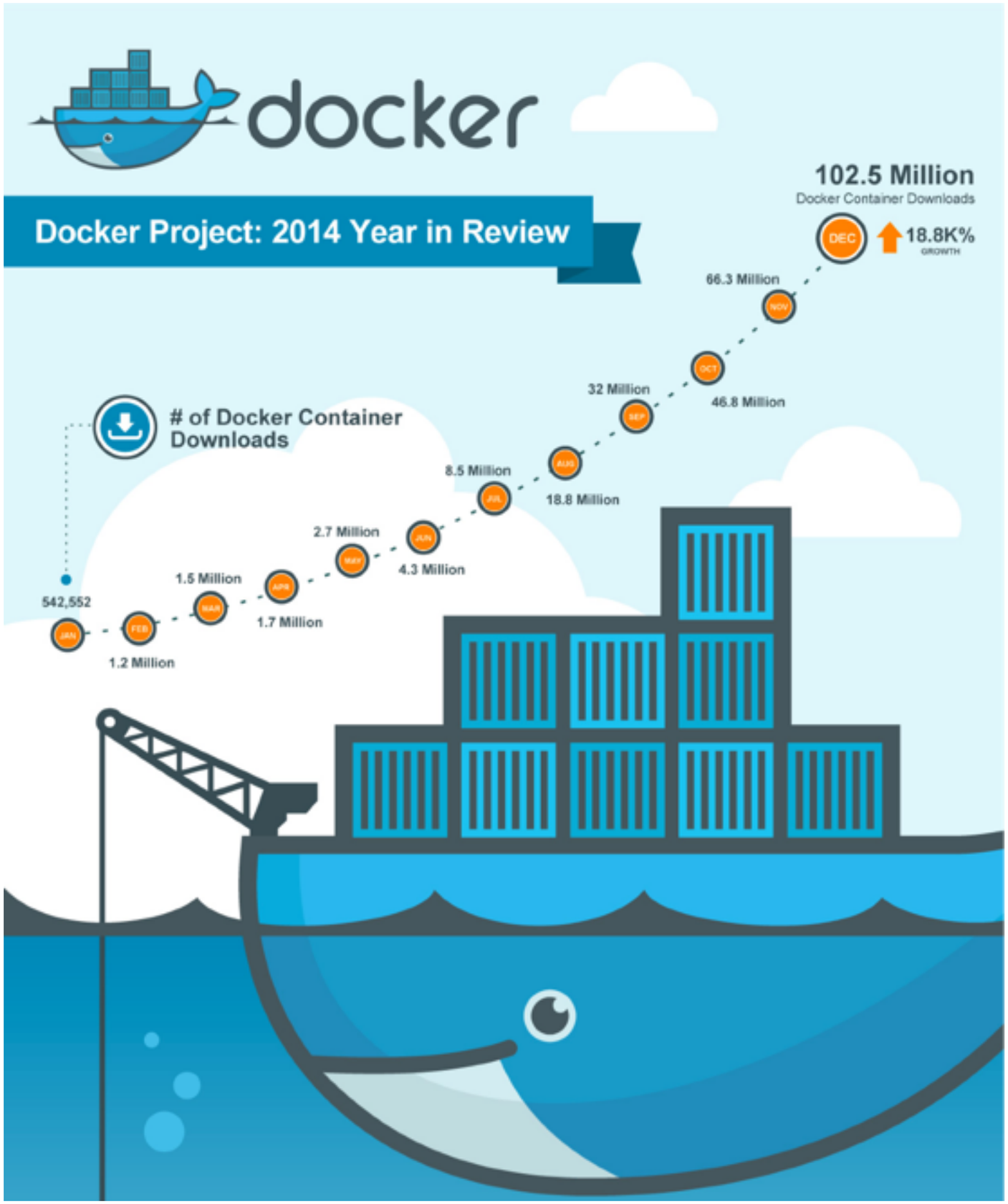
https://docs.docker.com/reference/commandline/cli/



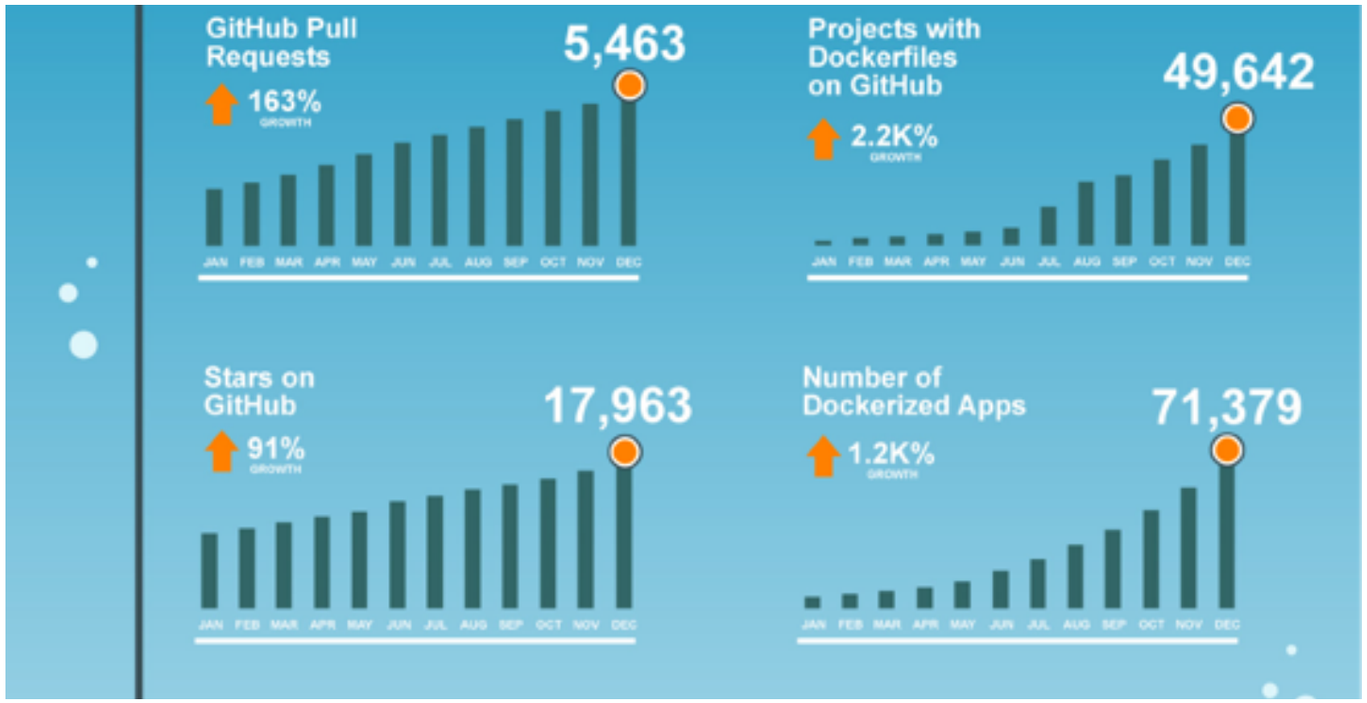
http://www.wavemaker.com/cloud/docker-ecosystem-year-in-review-2014/

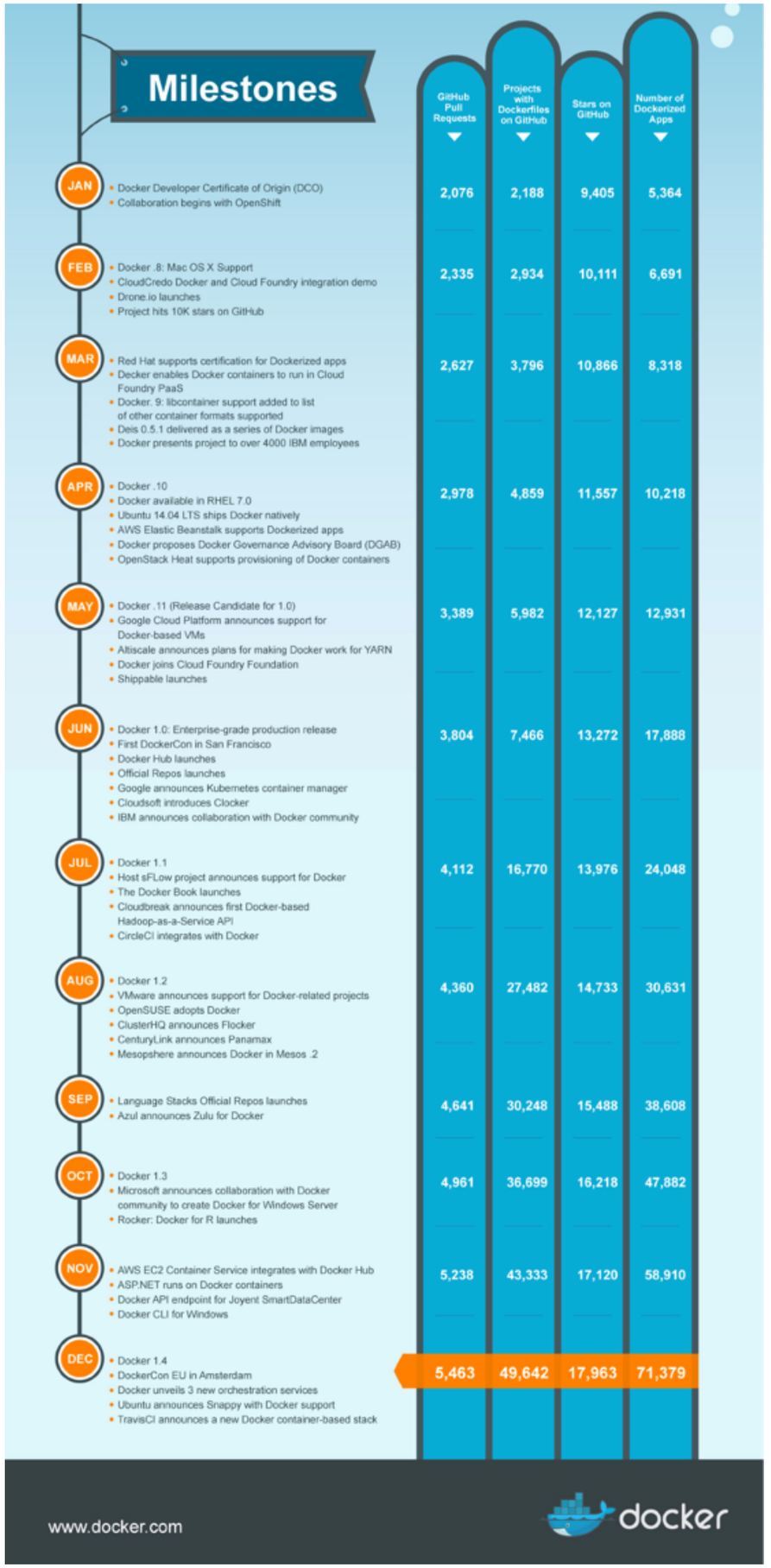


http://www.wavemaker.com/cloud/docker-ecosystem-year-in-review-2014/



http://venturebeat.com/2015/01/15/heres-how-big-docker-got-last-year/

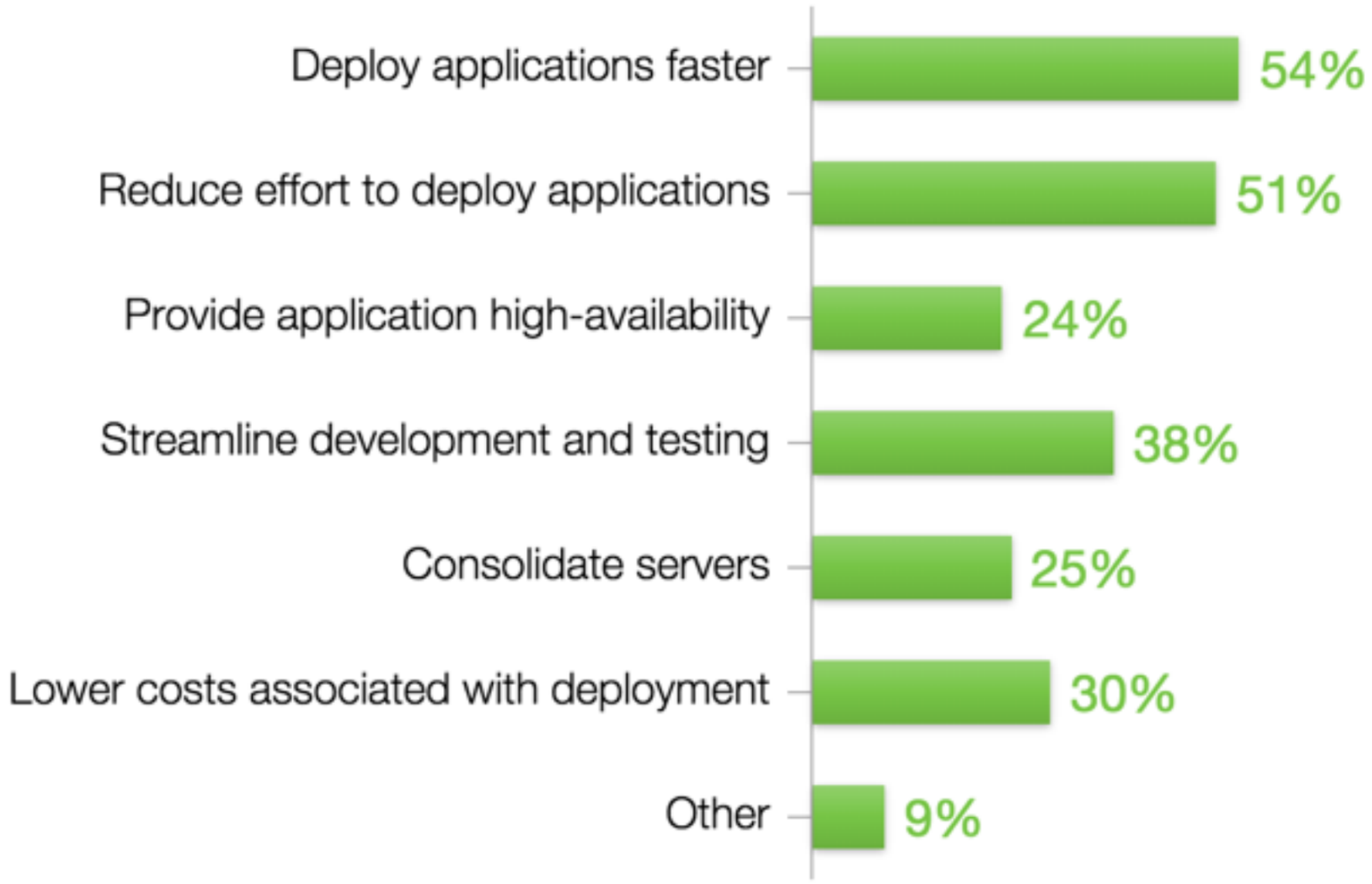




http://venturebeat.com/2015/01/15/heres-how-big-docker-got-last-year/

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.

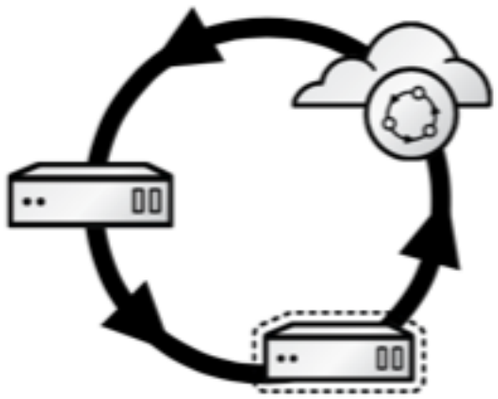
Source: TechValidate survey of 79 IT professionals

http://www.techvalidate.com/product-research/red-hat-enterprise-linux/charts/E21-1EE-BF2

Benefits of Containers

FASTER APP DELIVERY

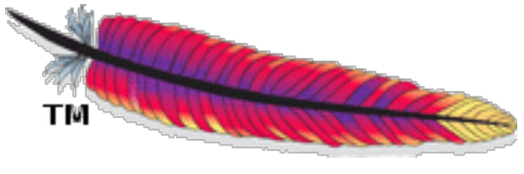
OPERATIONAL EFFICIENCY



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

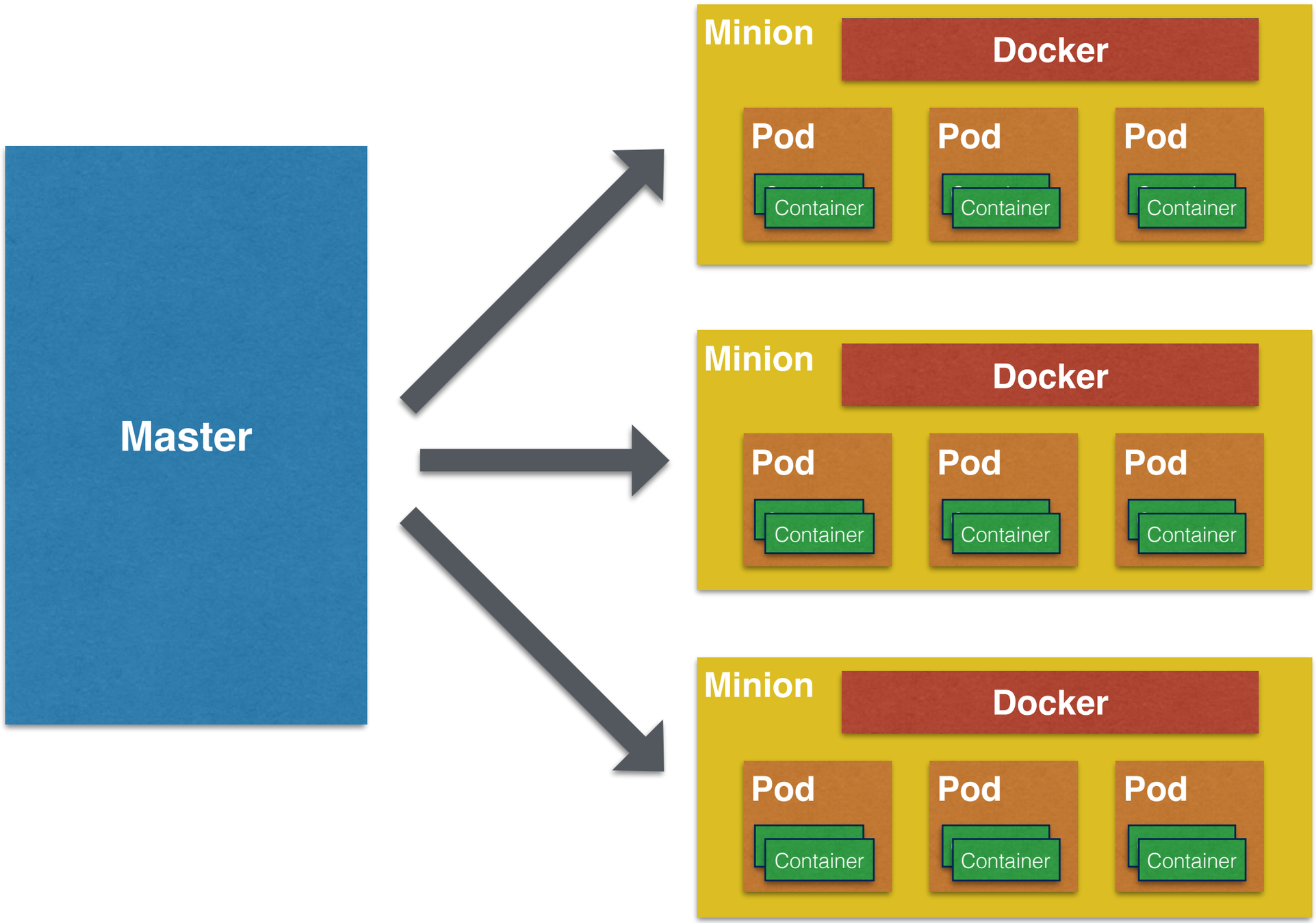
Key Concepts

- **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


Master
Minion
Docker
Pod
Container
Pod
Container
Pod
Container
Minion
Docker
Pod
Container
Pod
Container
Pod
Container
Minion
Docker
Pod
Container
Pod
Container
Pod
Container

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>`
- `lubectl update` or `delete`

kubectl

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

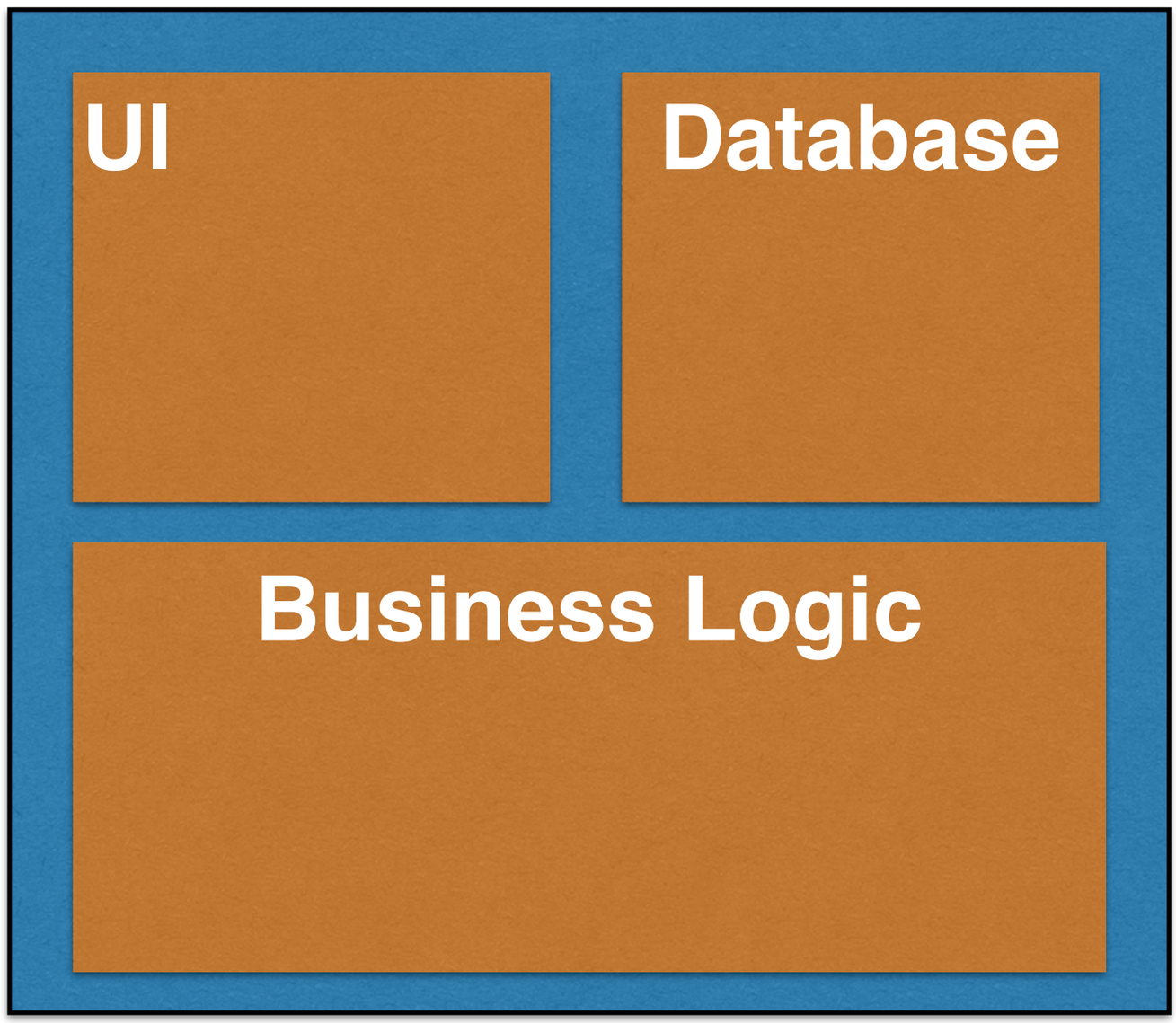
*building applications as **suites of services**. As well as the fact that services are **independently deployable and scalable**, each service also provides a **firm module boundary**, even allowing for different services to be written in **different programming languages**. They can also be **managed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

http://martinfowler.com/articles/microservices.html

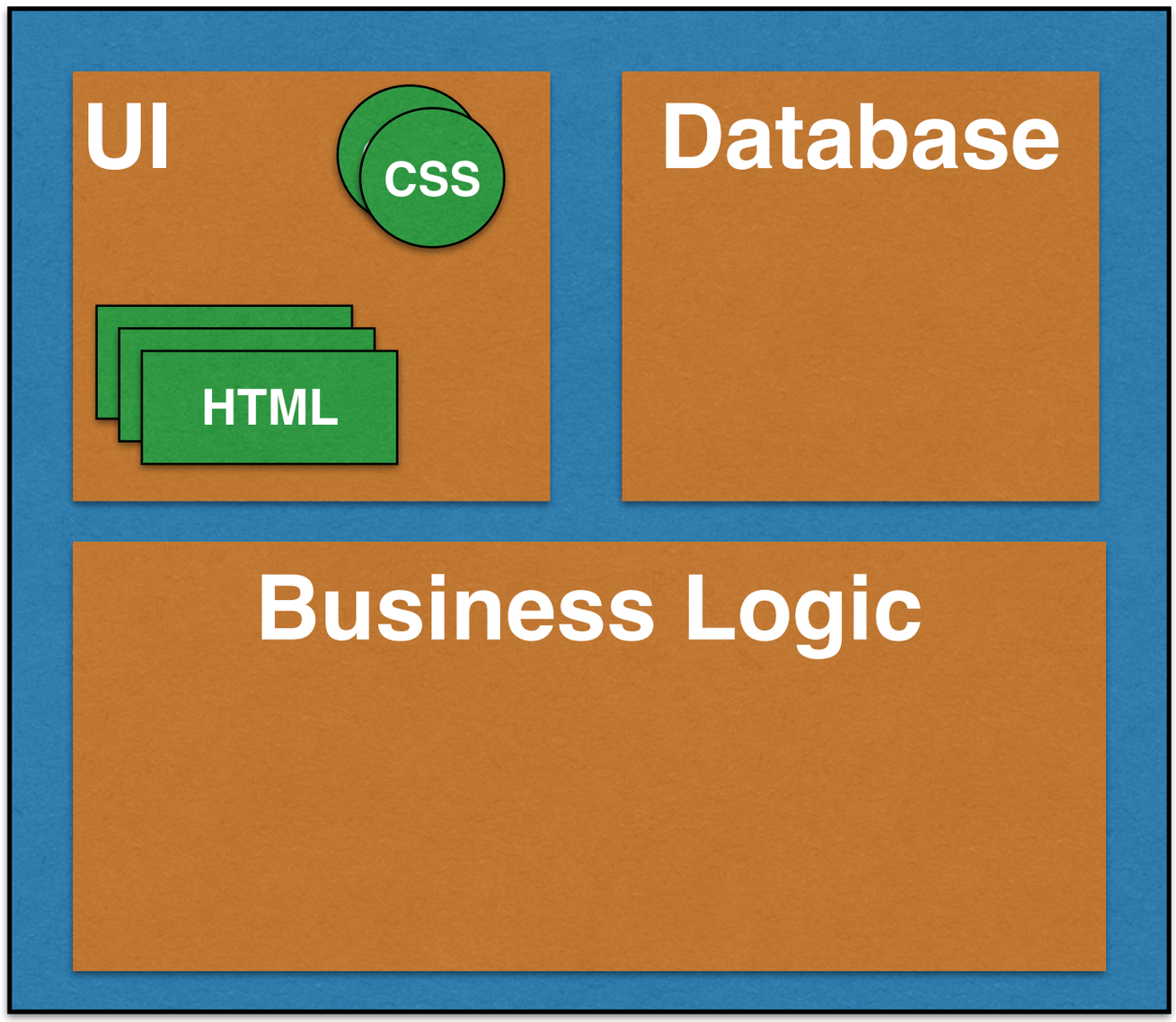
Is it SOA?

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

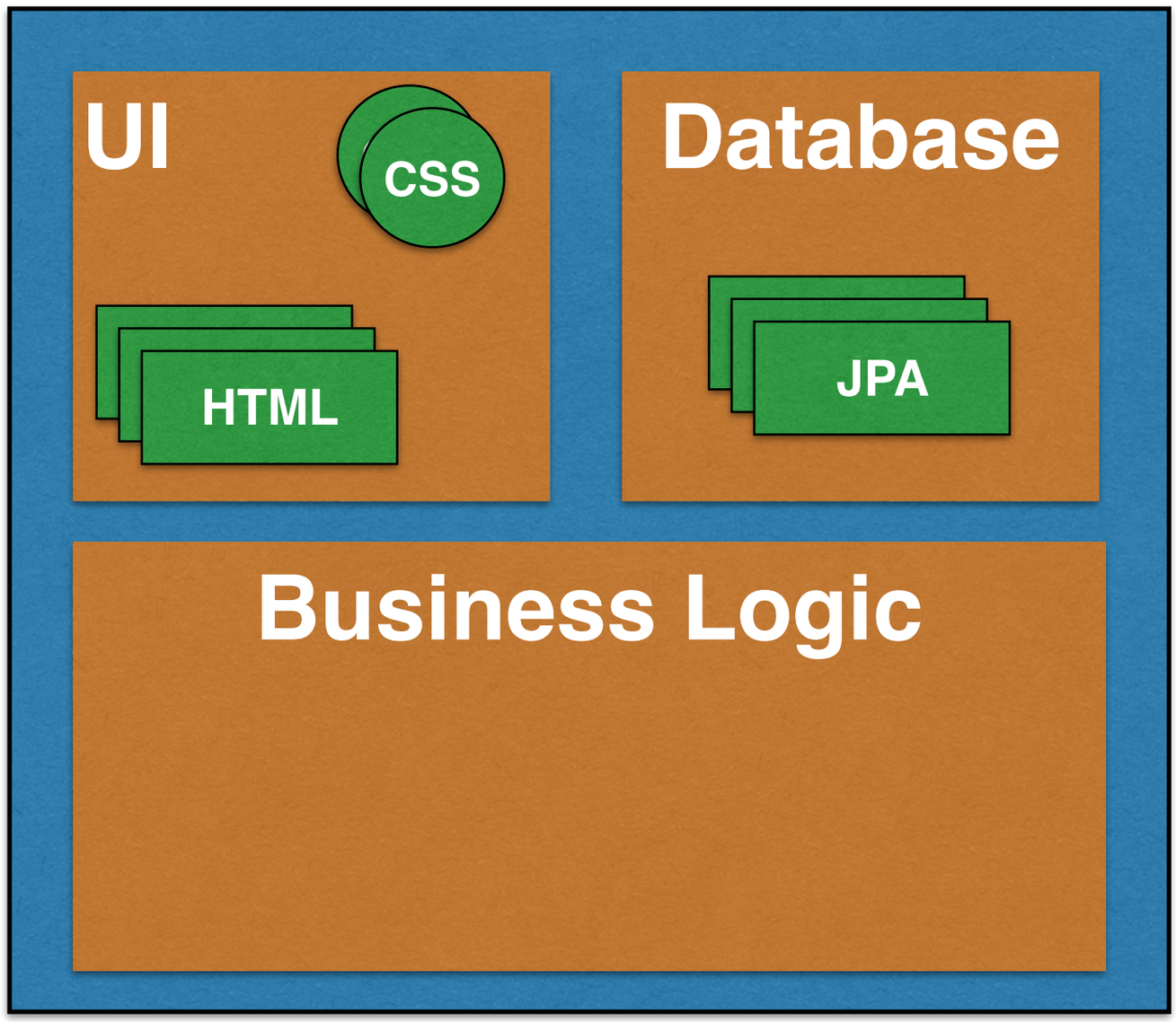
Monolith Application



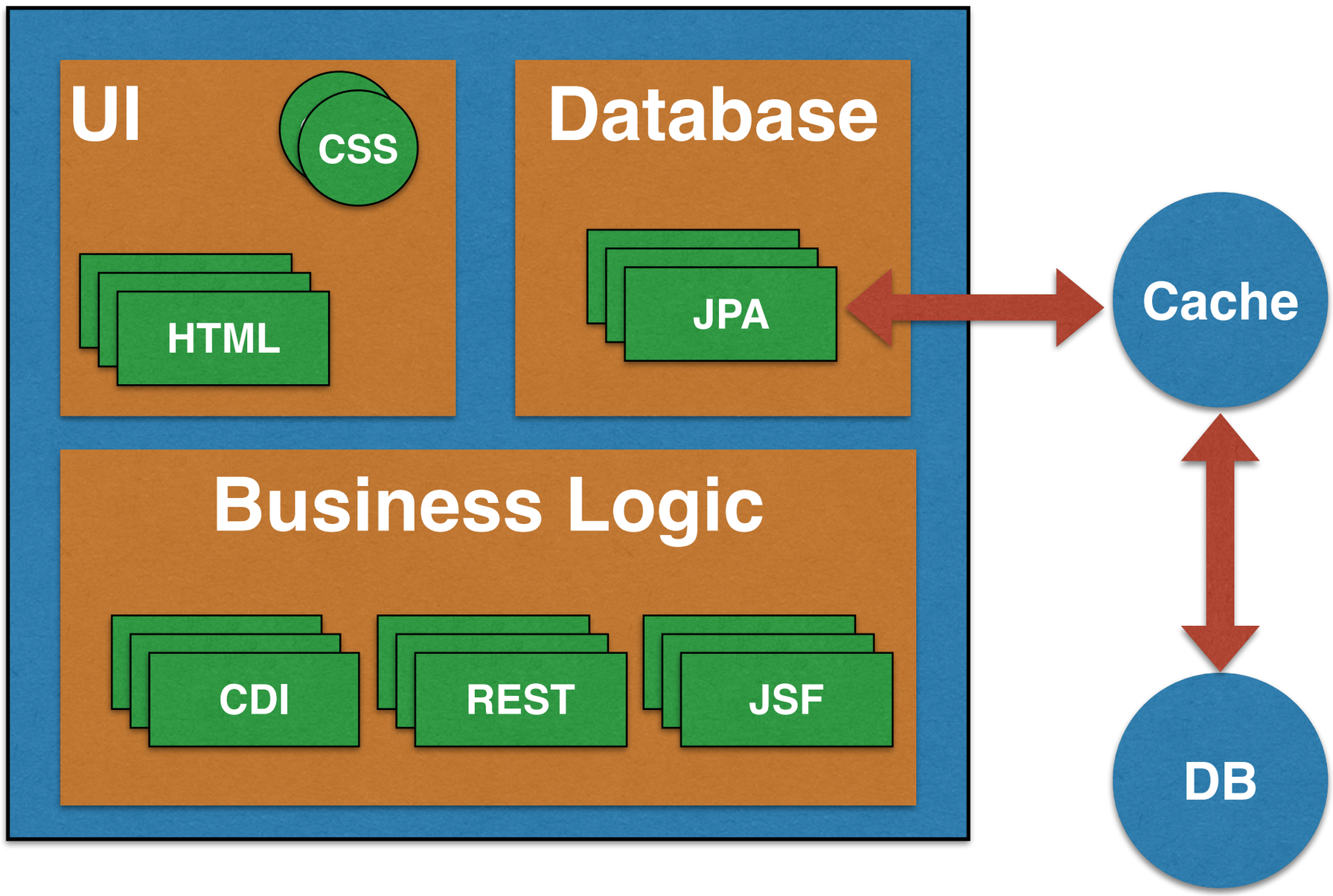
Monolith Application



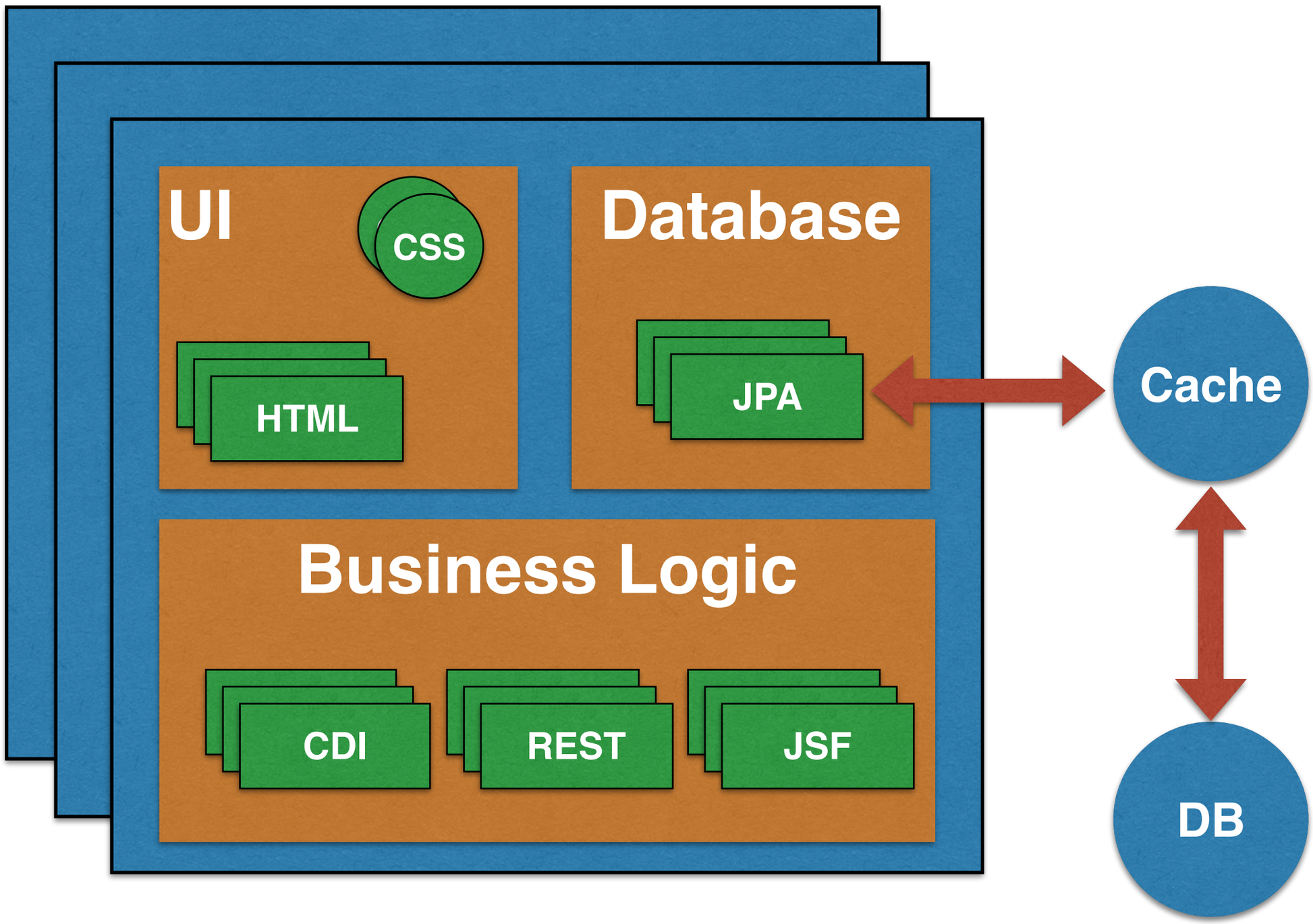
Monolith Application



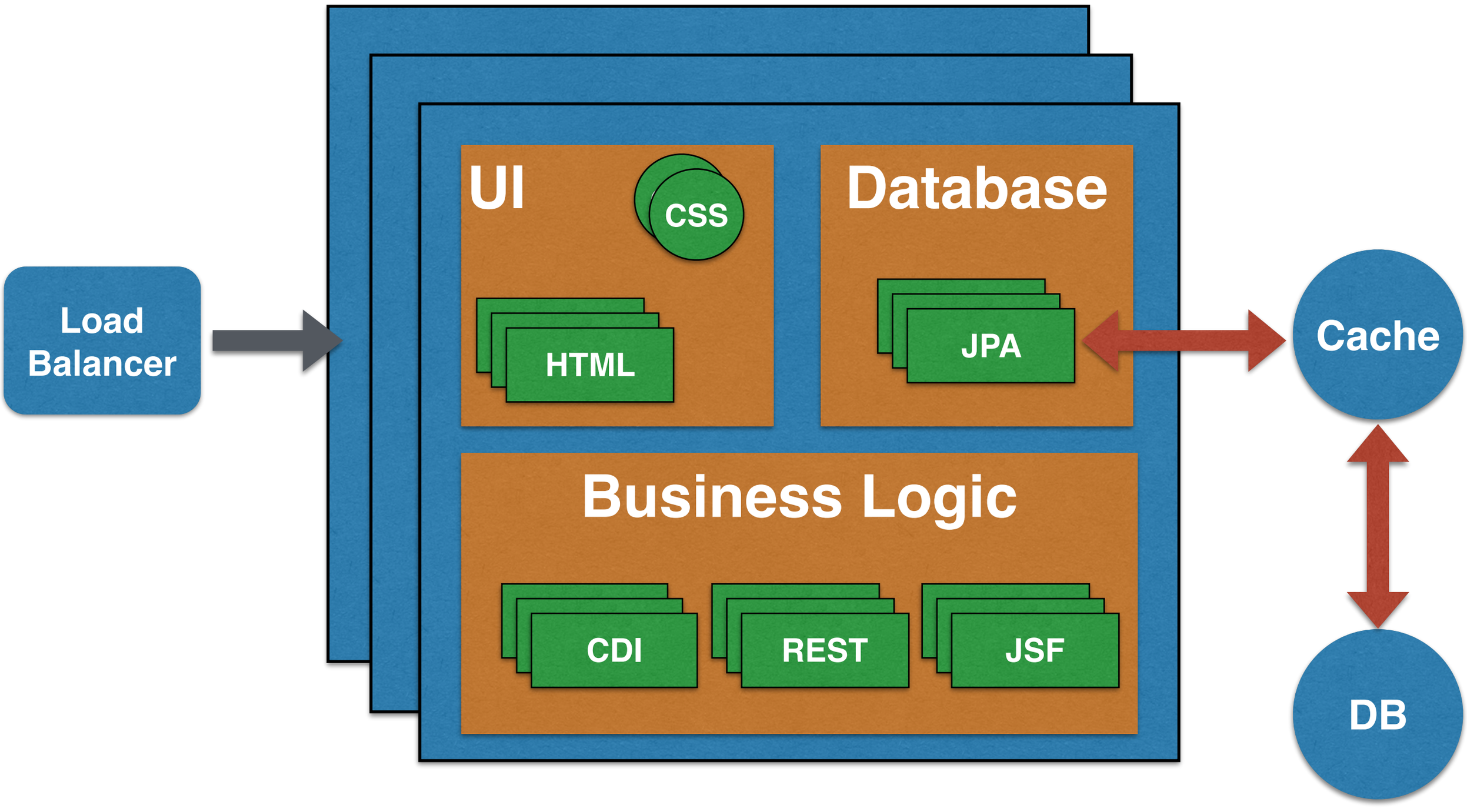
Monolith Application



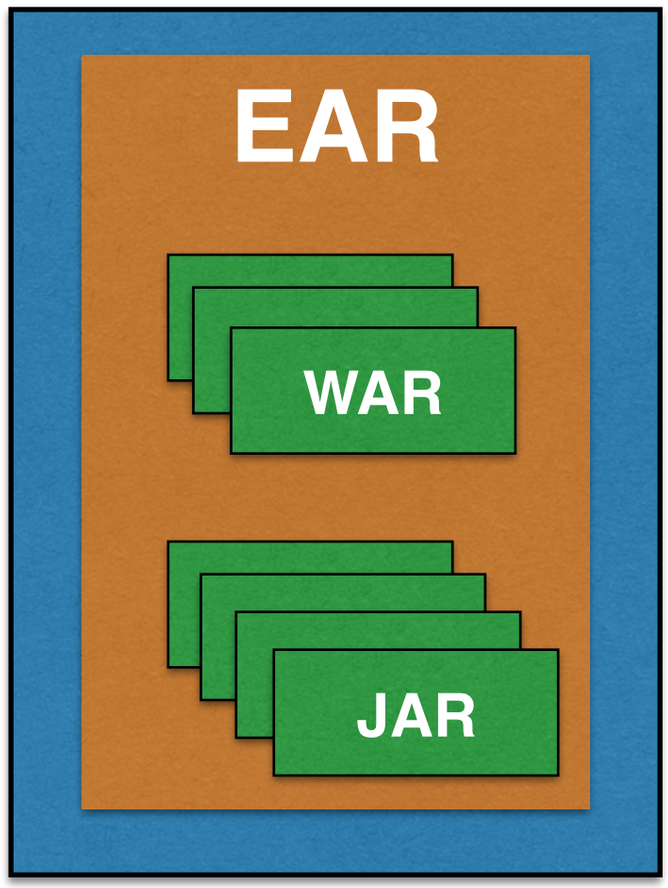
Monolith Application



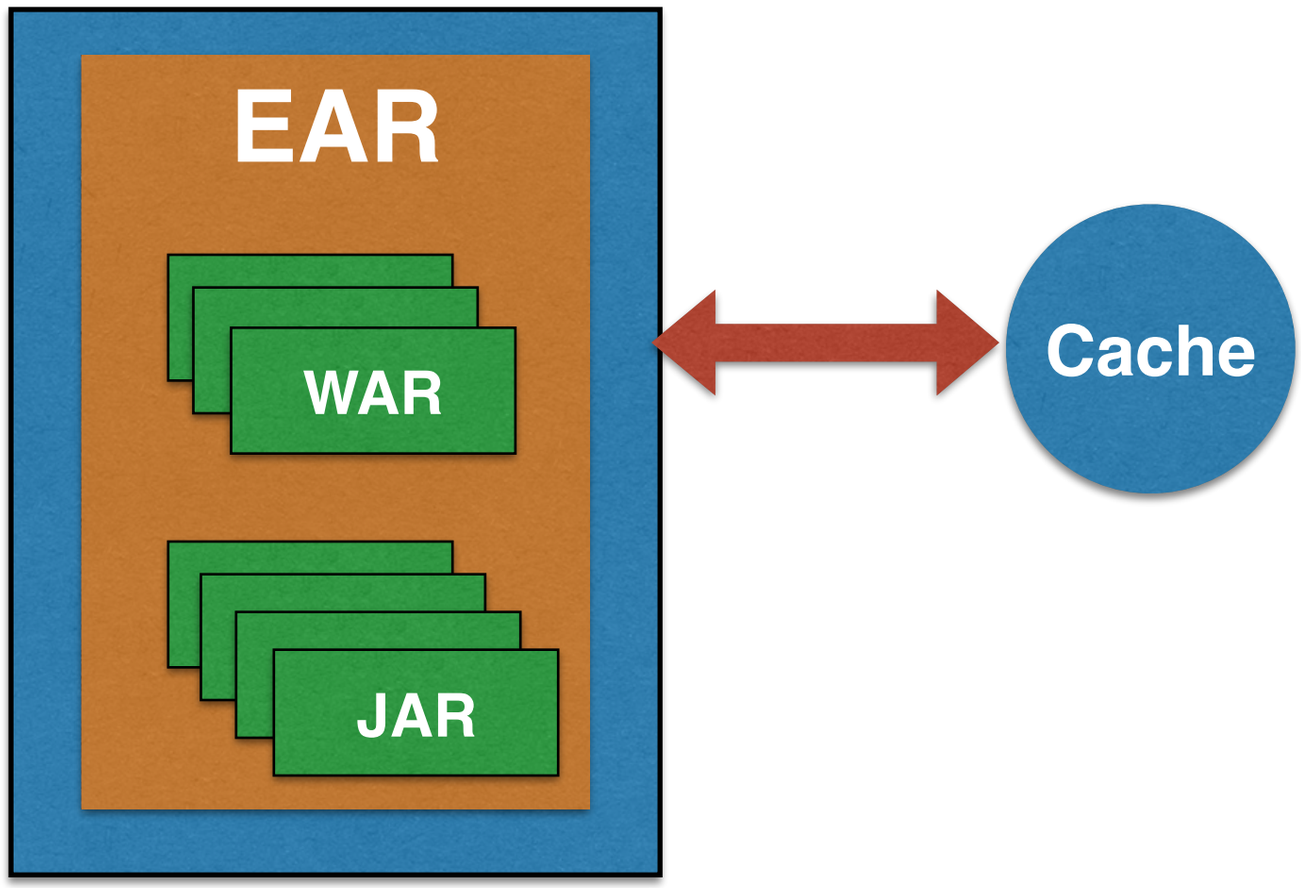
Monolith Application



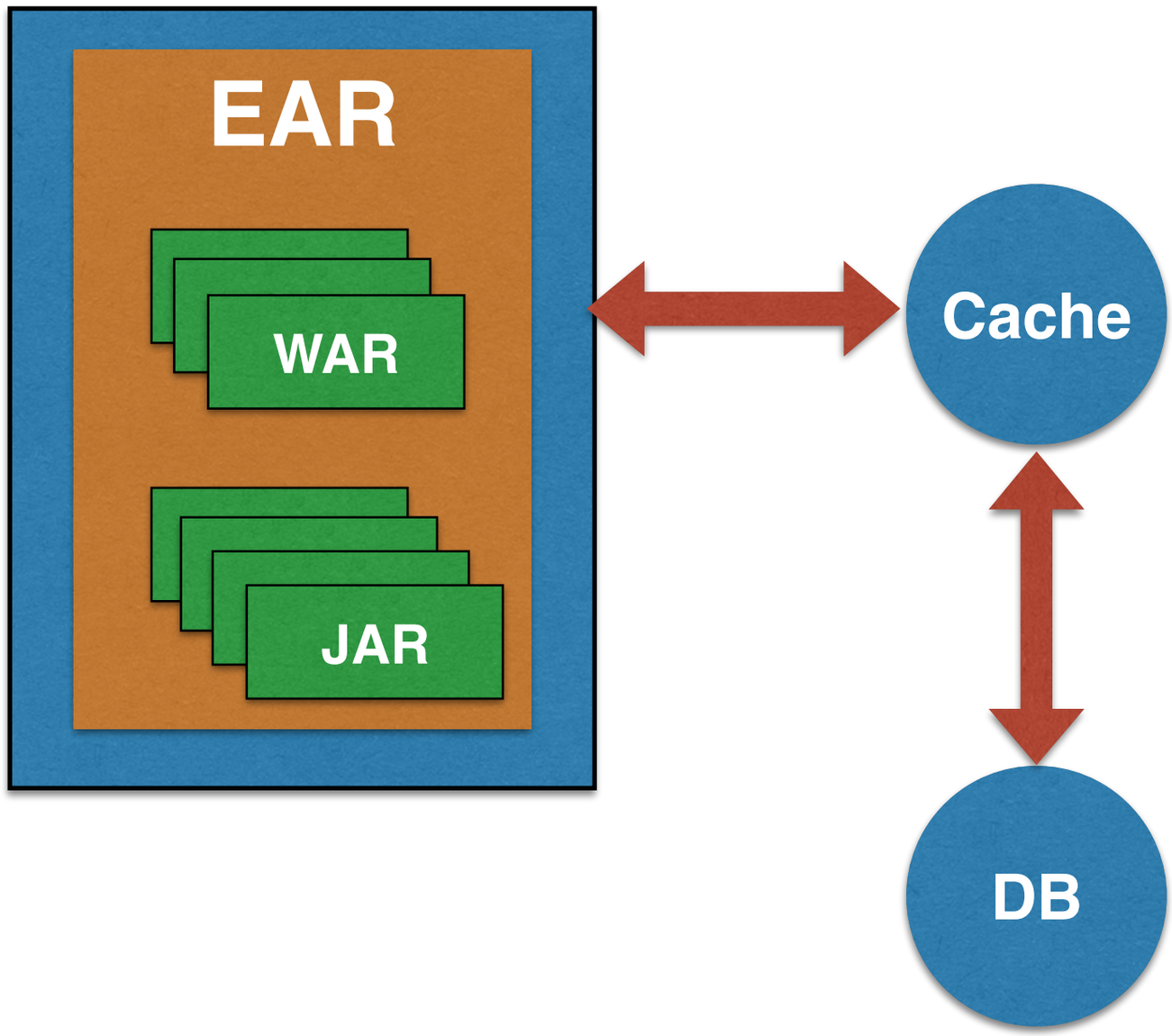
Before Microservices



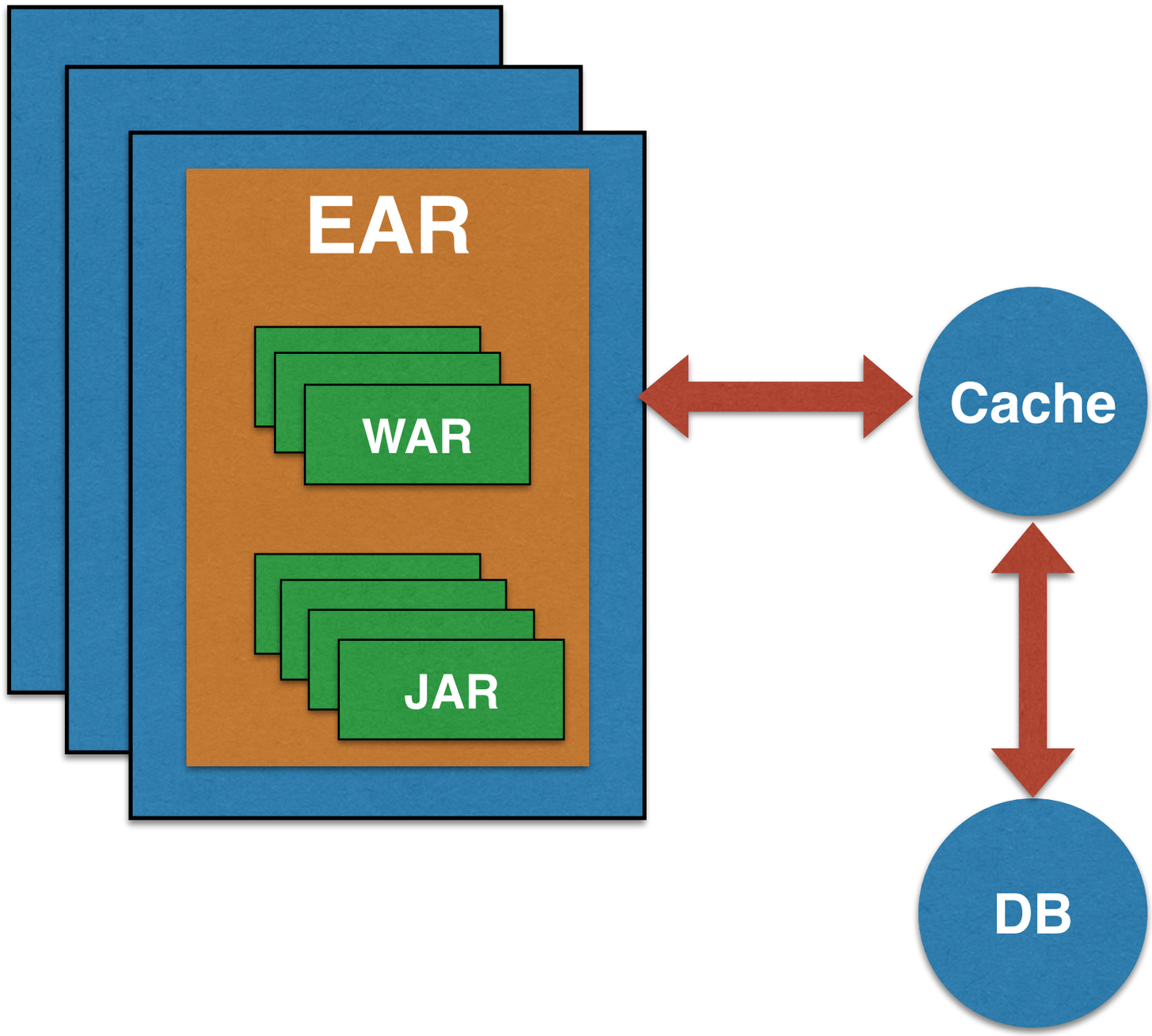
Before Microservices



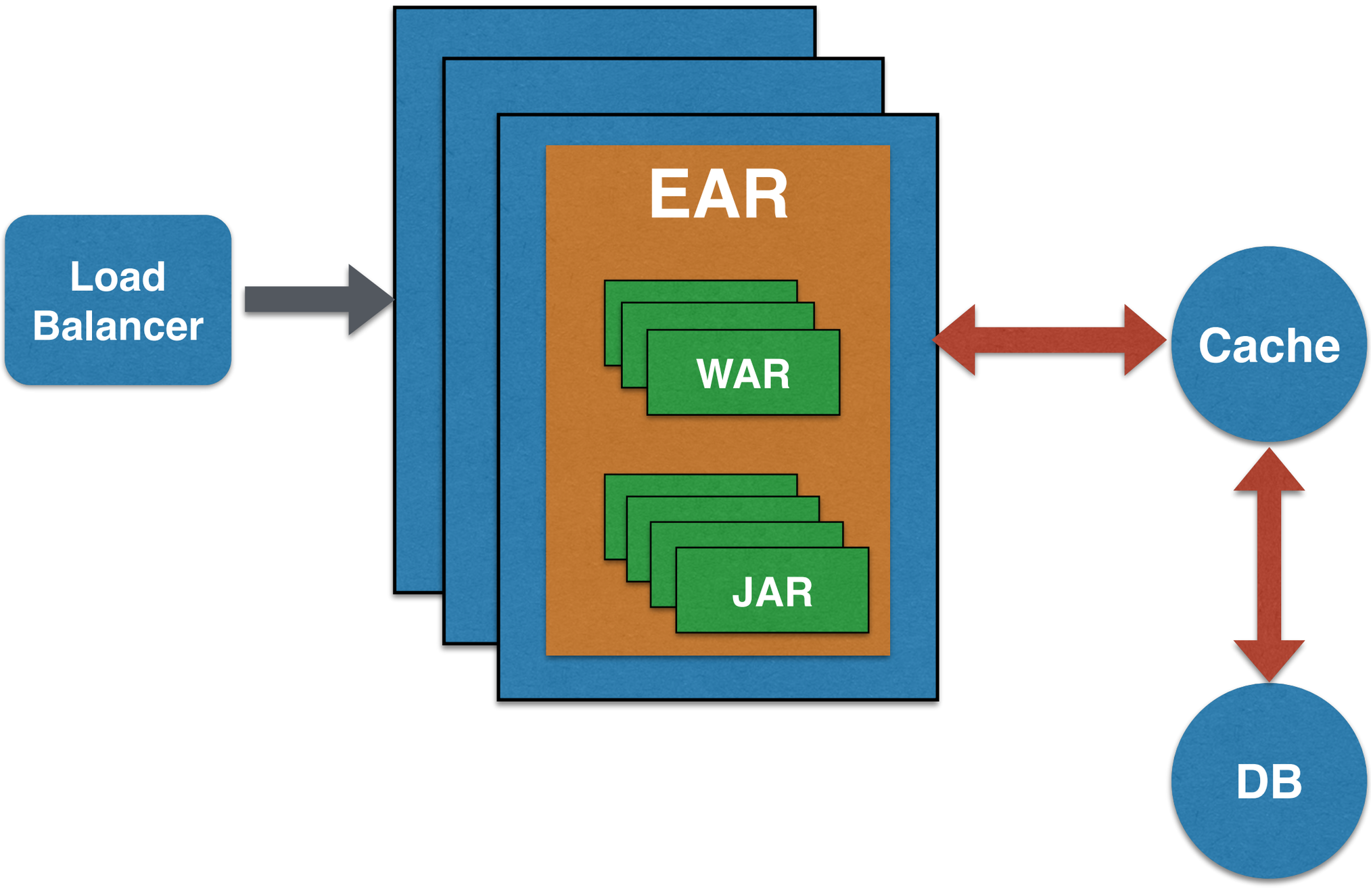
Before Microservices



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

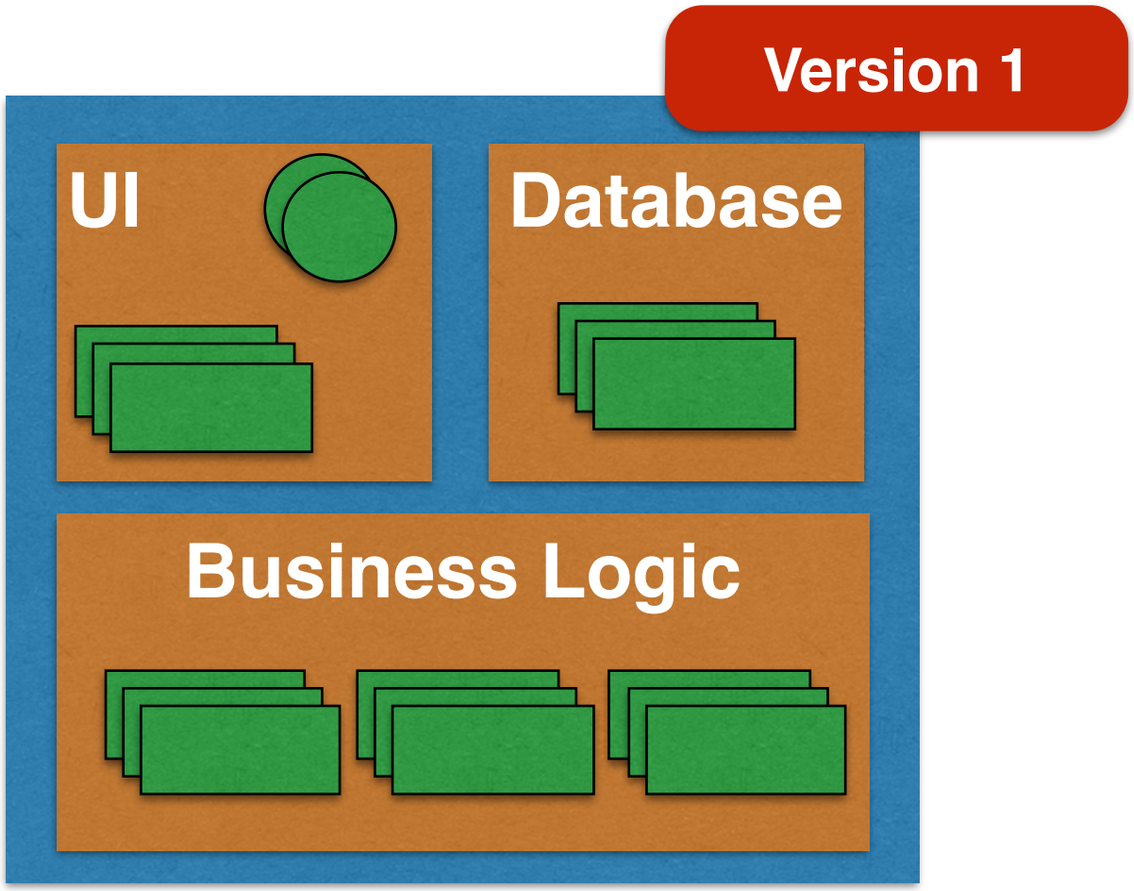
Disadvantages of Monolith Application

- Difficult to develop and maintain
- Obstacle to frequent deployments

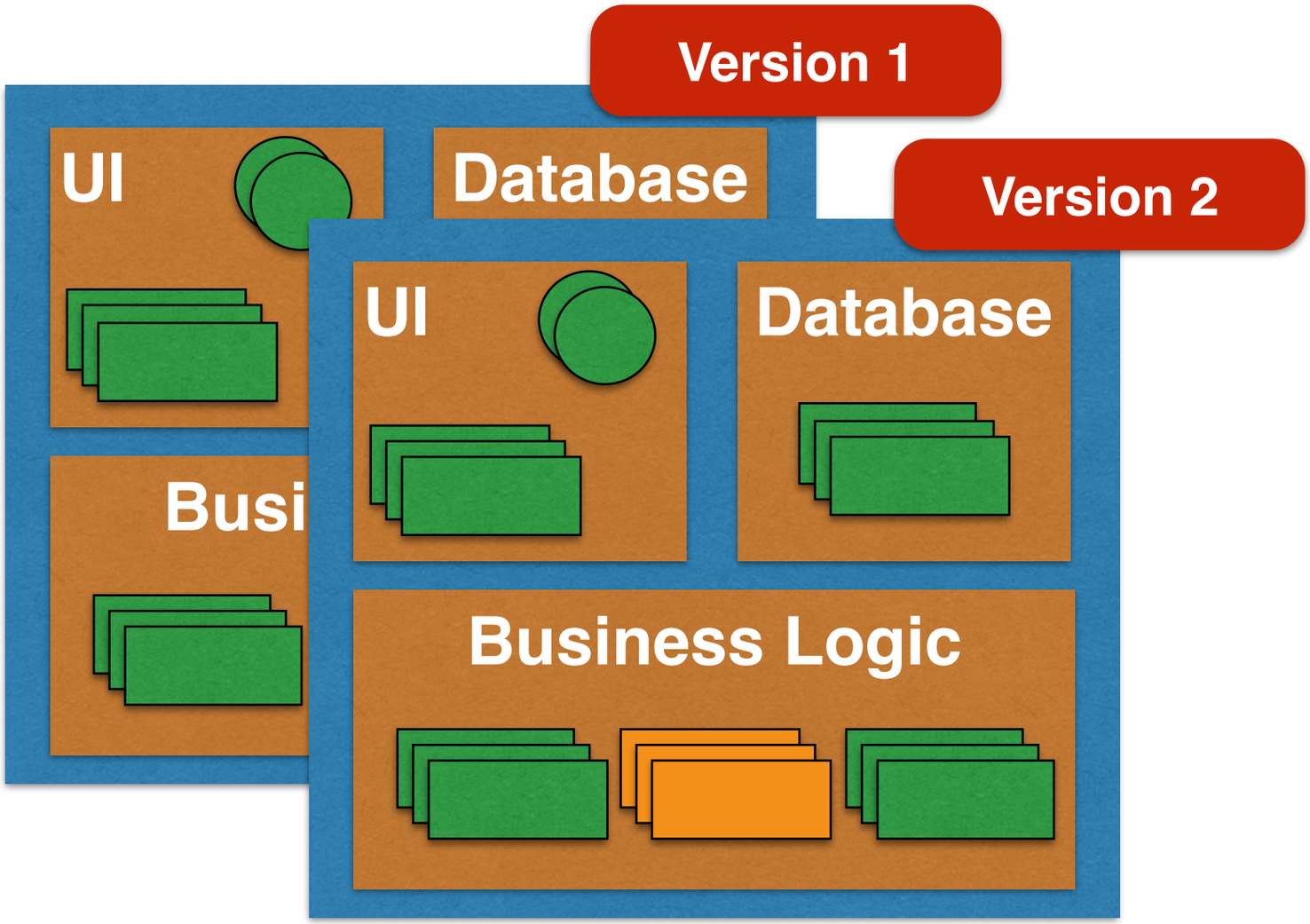
Disadvantages of Monolith Application

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

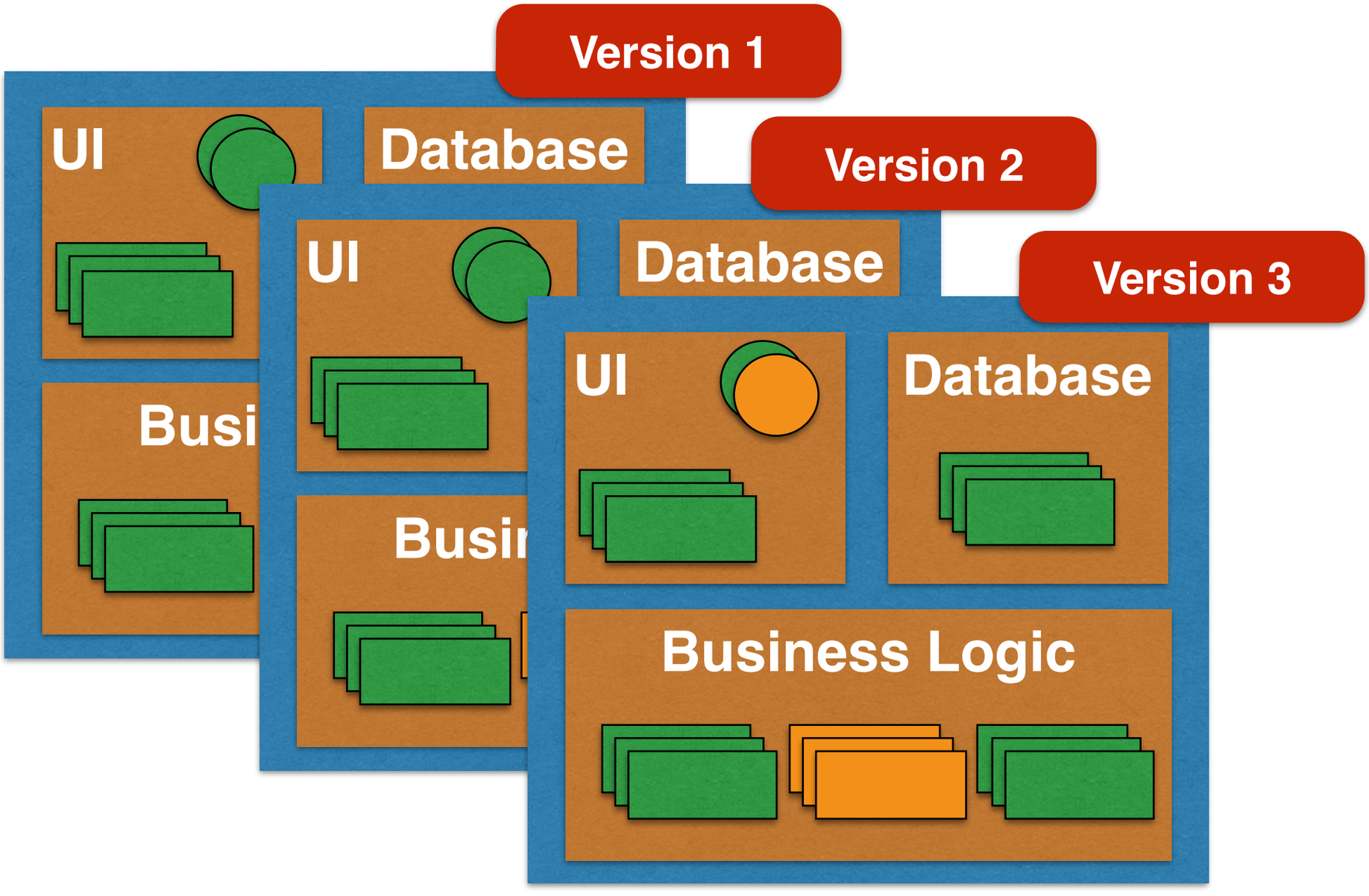
Monolith Application



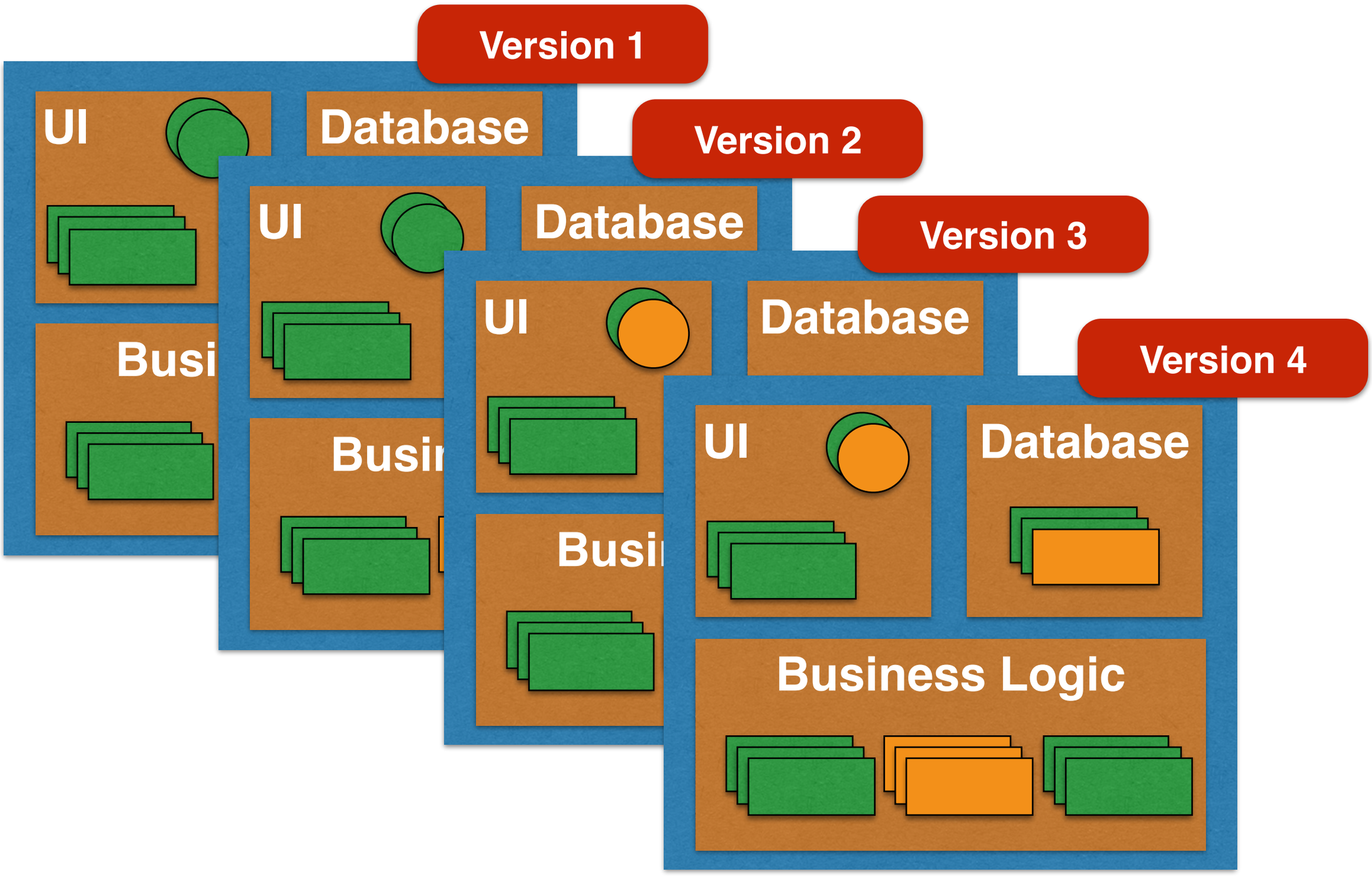
Monolith Application



Monolith Application

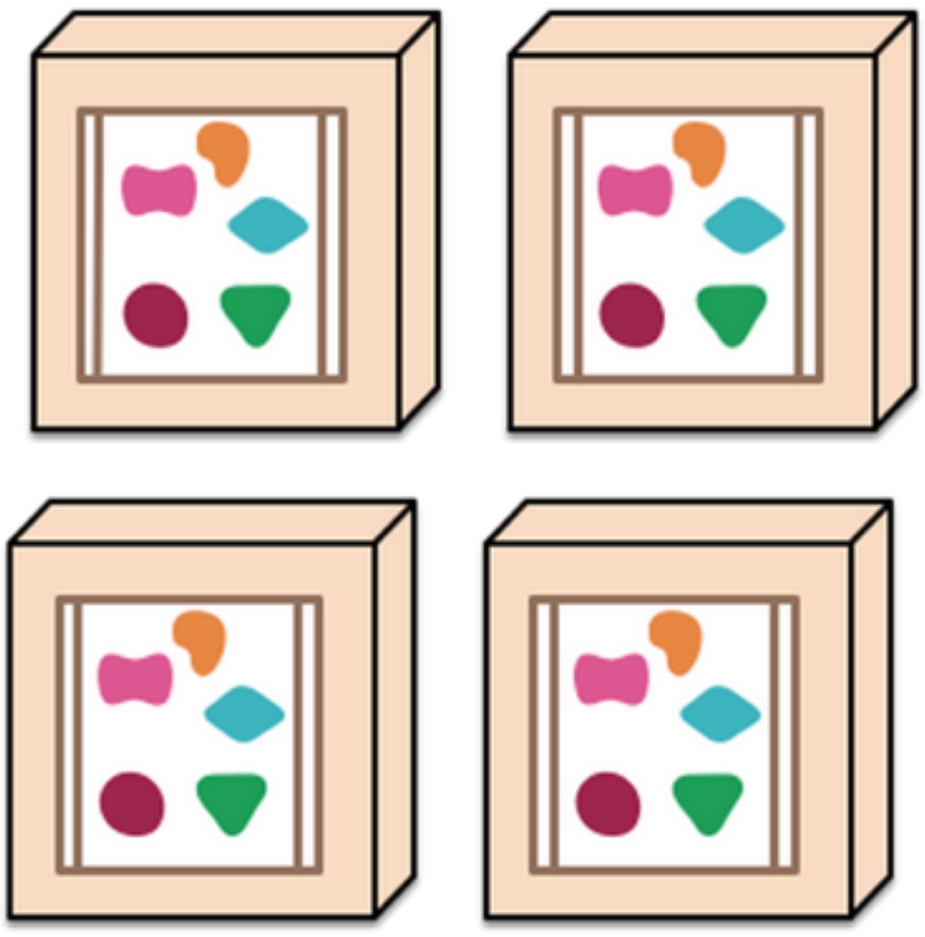


Monolith Application

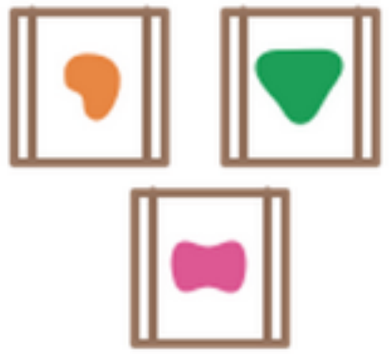


A monolithic application puts all its functionality into a single process...

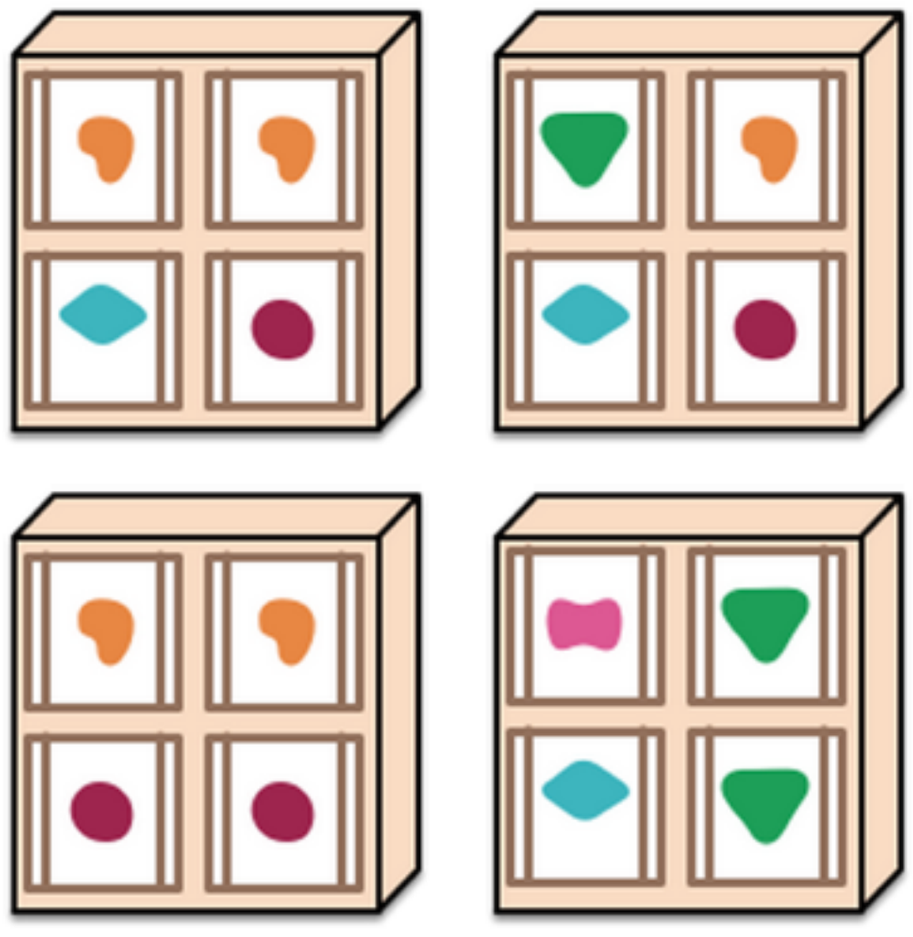
... and scales by replicating the monolith on multiple servers



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



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



http://martinfowler.com/articles/microservices.html

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

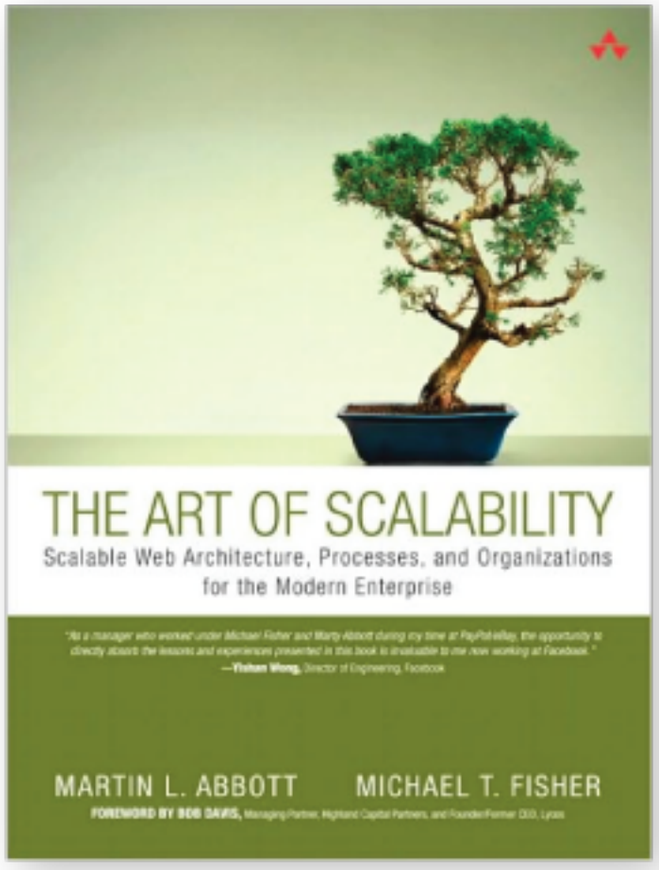
Characteristics of µservices

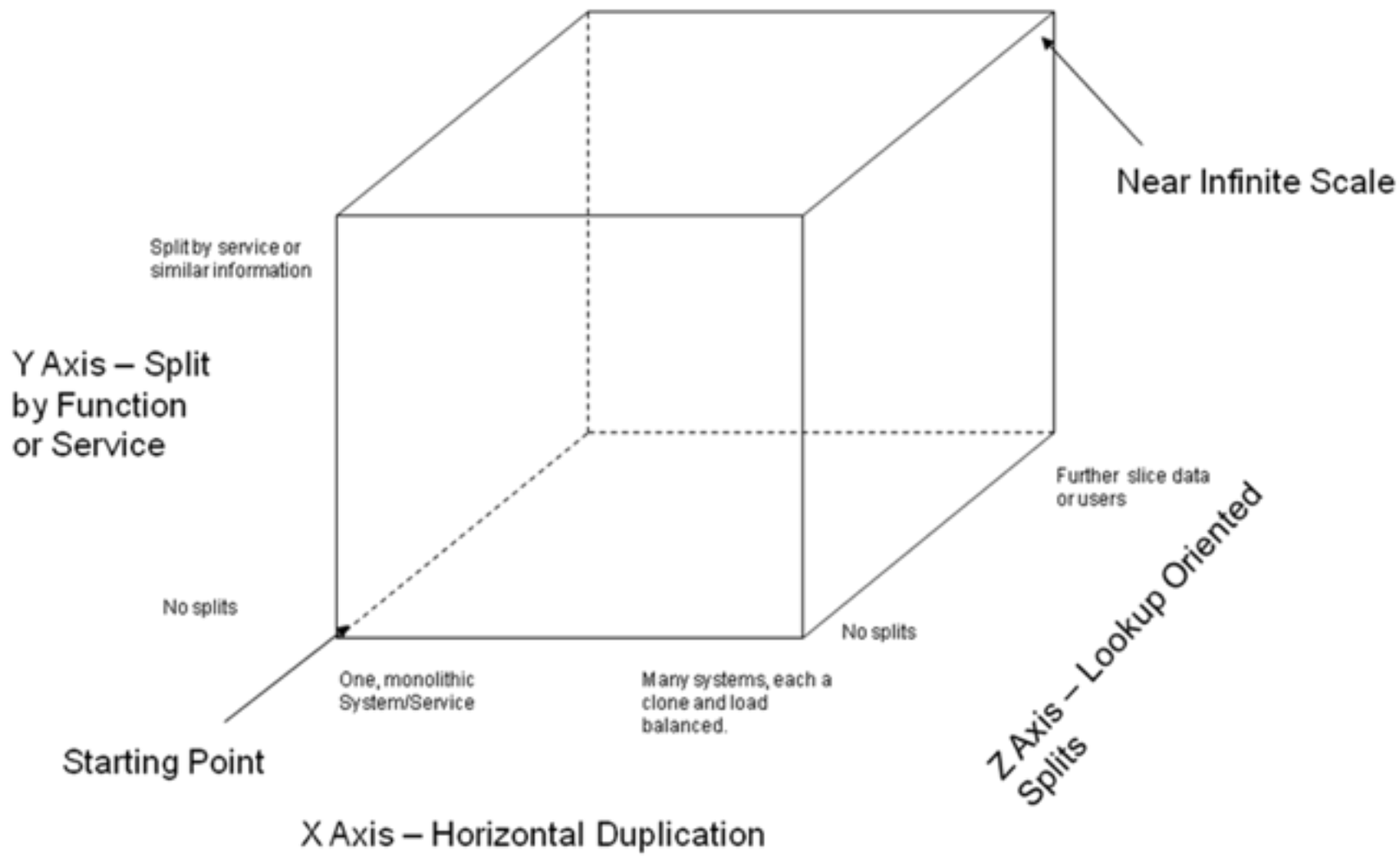
- Explicitly published interface, bounded context
- Domain driven design (full-stack developer)
- Independently deployable and automated (CI / CD)
- Designed for failure
- Use IPC to communicate
- Full responsibility - “you build it, you run it”

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





http://akfpartners.com/techblog/2008/05/08/splitting-applications-or-services-for-scale/

Strategies for decomposing

Strategies for decomposing

- Verb or usecase - e.g. Checkout UI

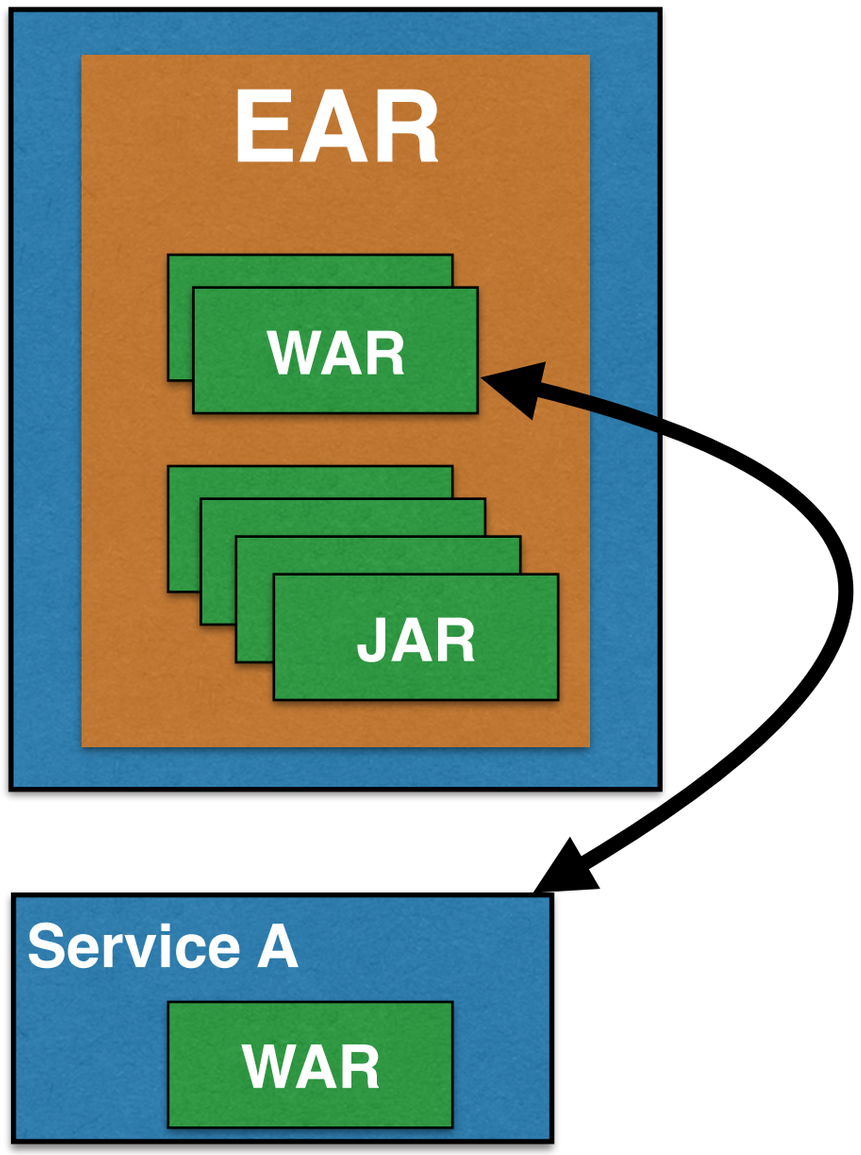
Strategies for decomposing

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

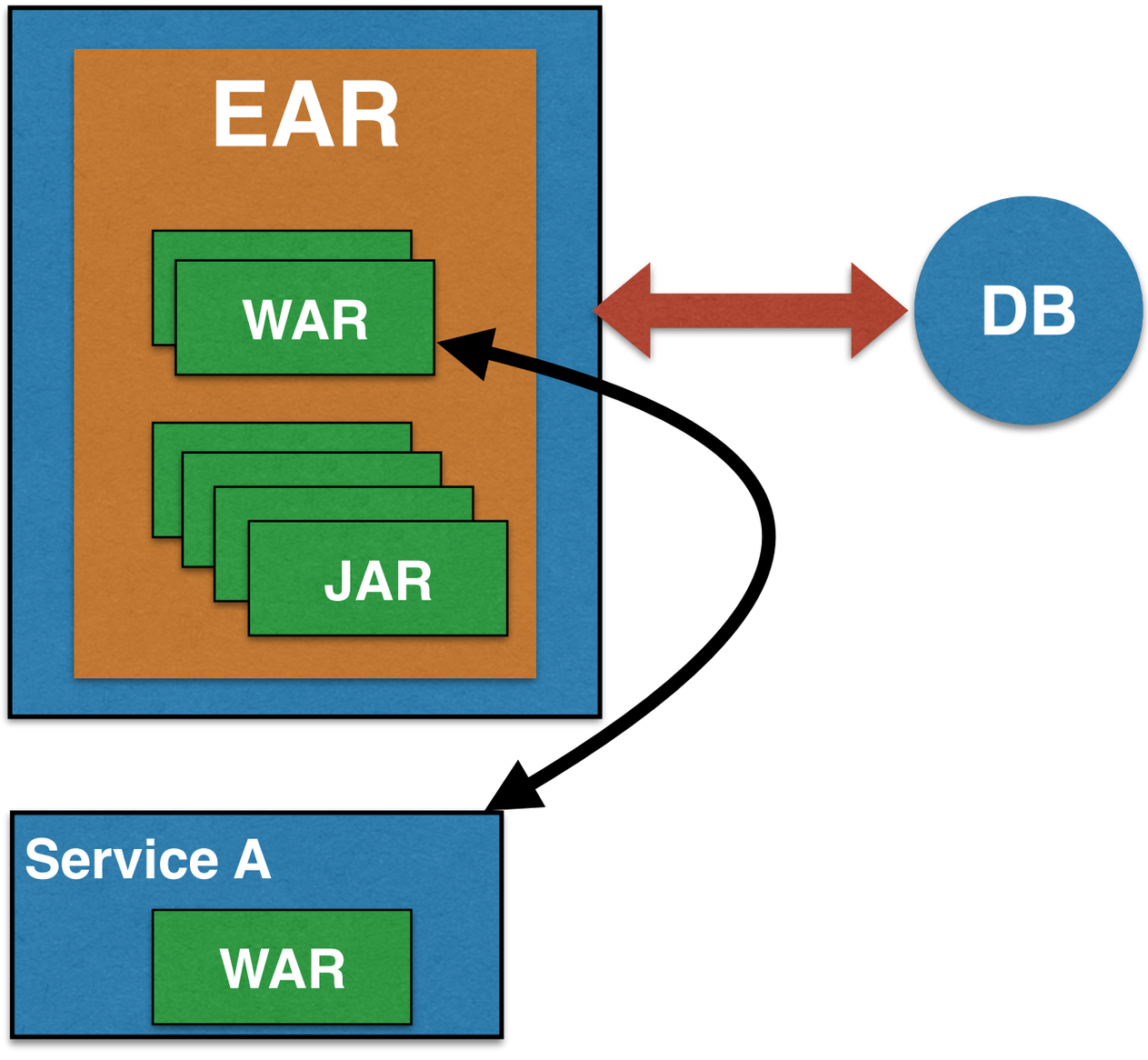
Strategies for decomposing

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

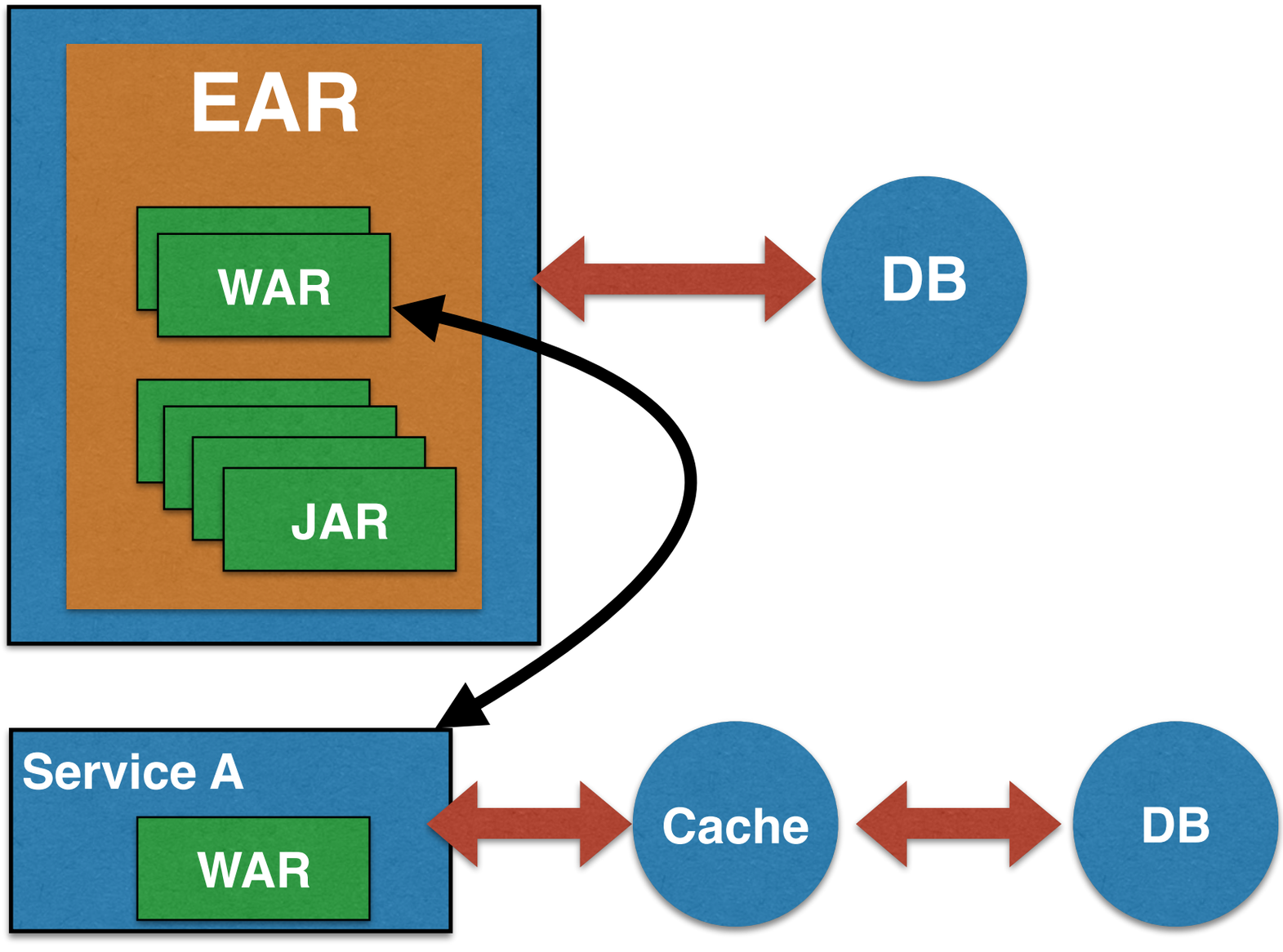
Towards μservices



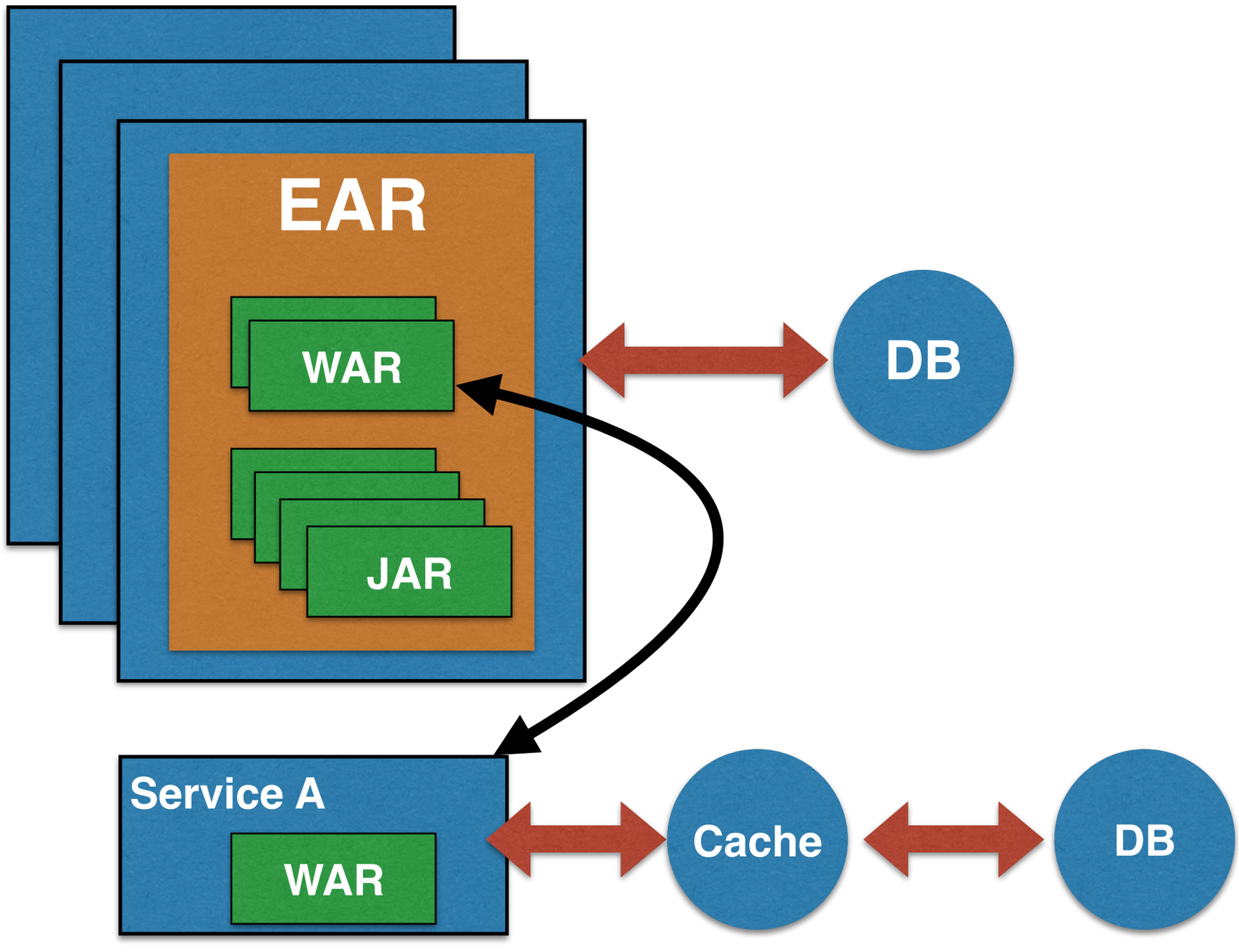
Towards μservices



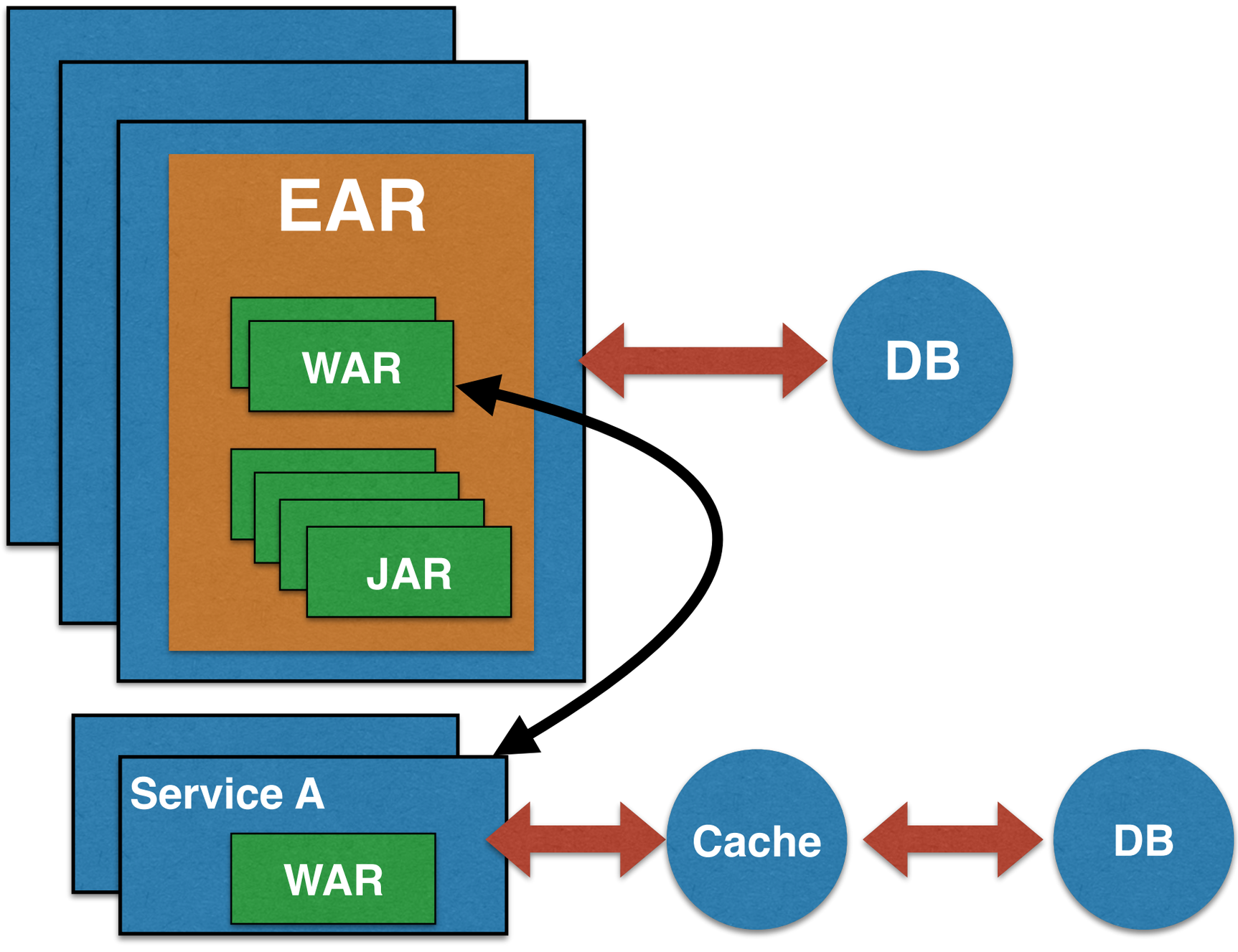
Towards μservices



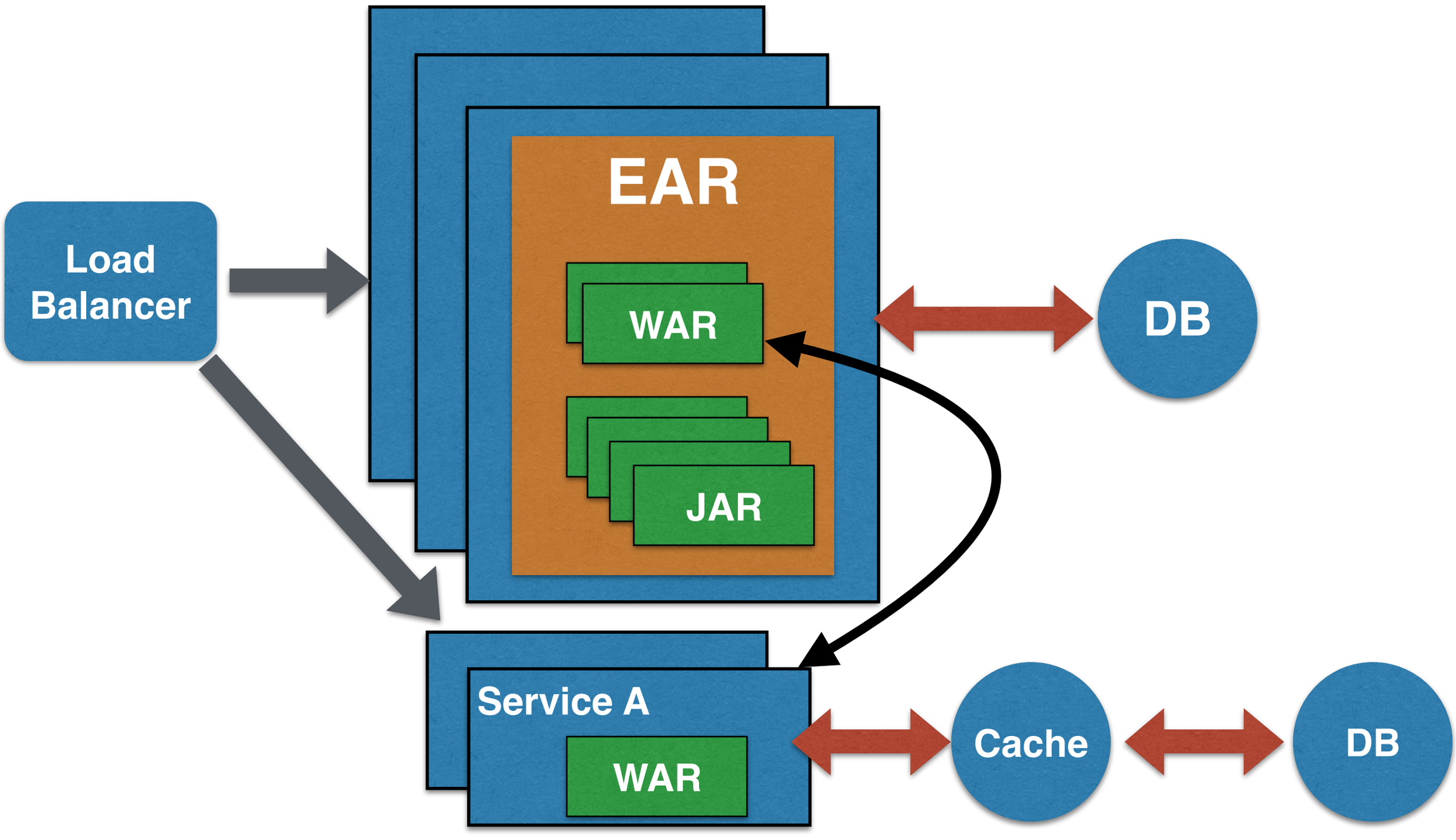
Towards μservices



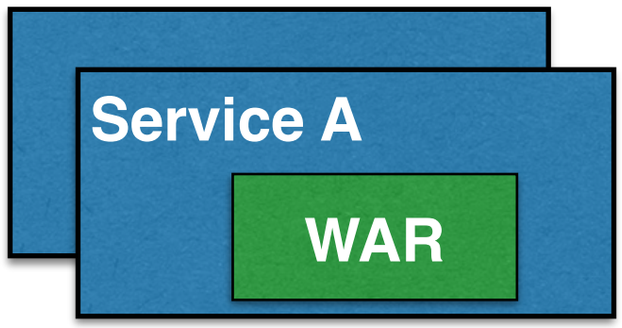
Towards μservices

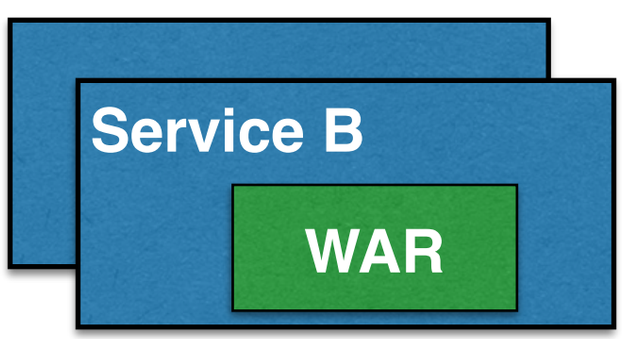


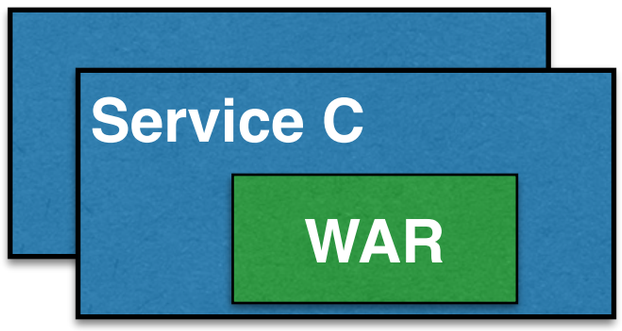
Towards μservices



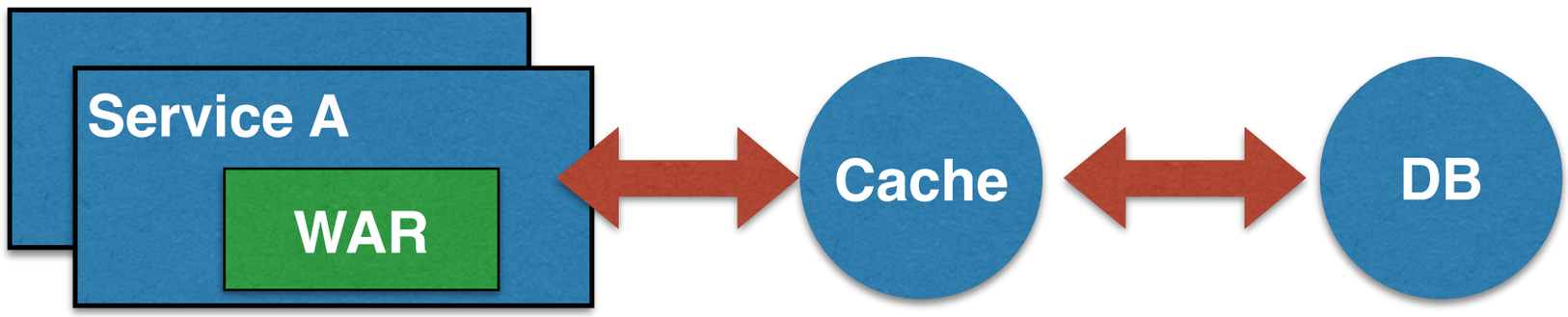
Aggregator Pattern #1

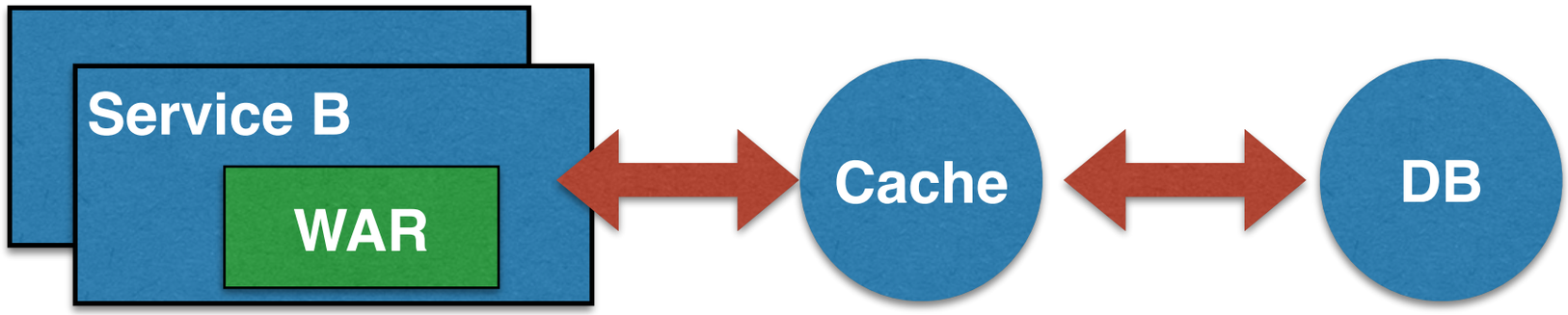


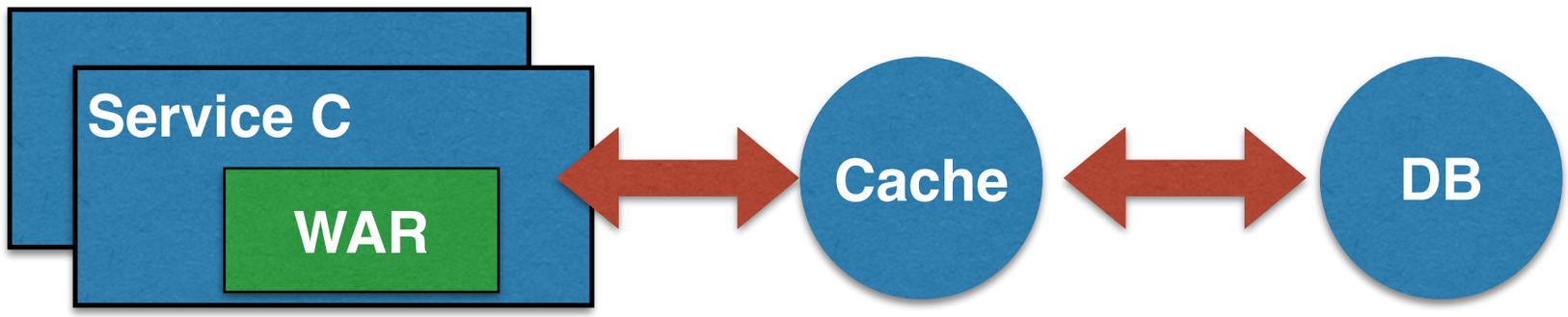




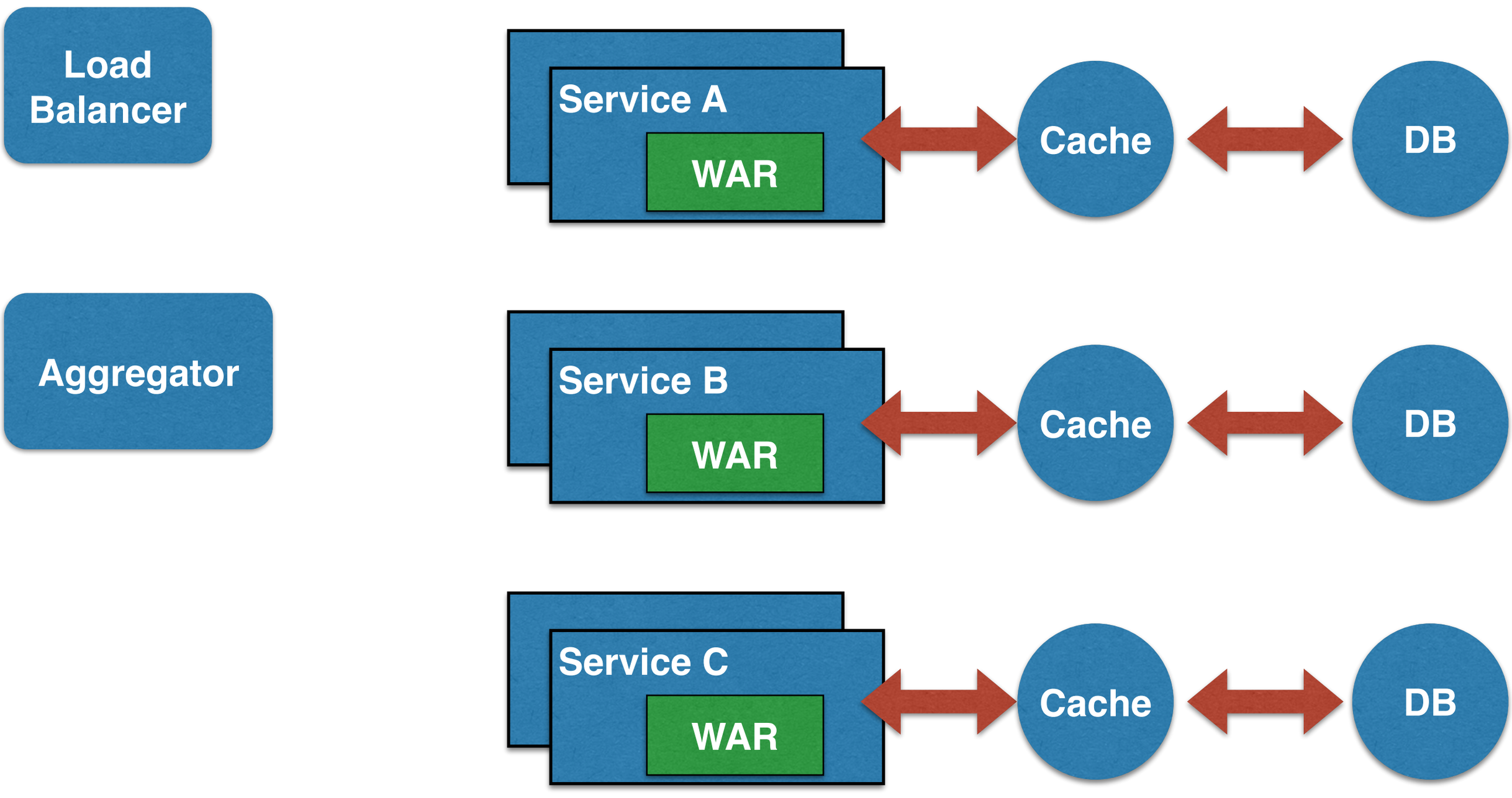
Aggregator Pattern #1



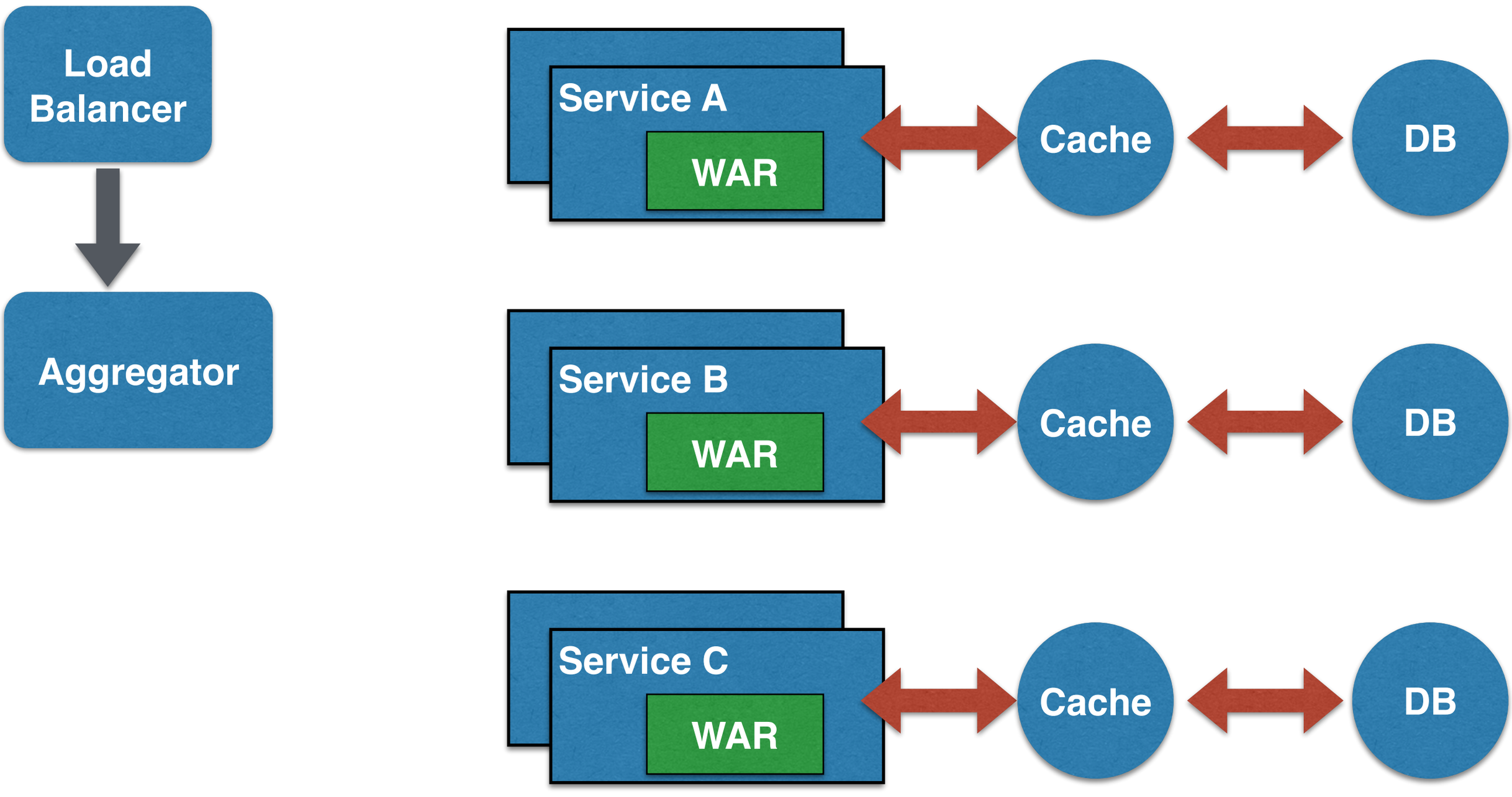




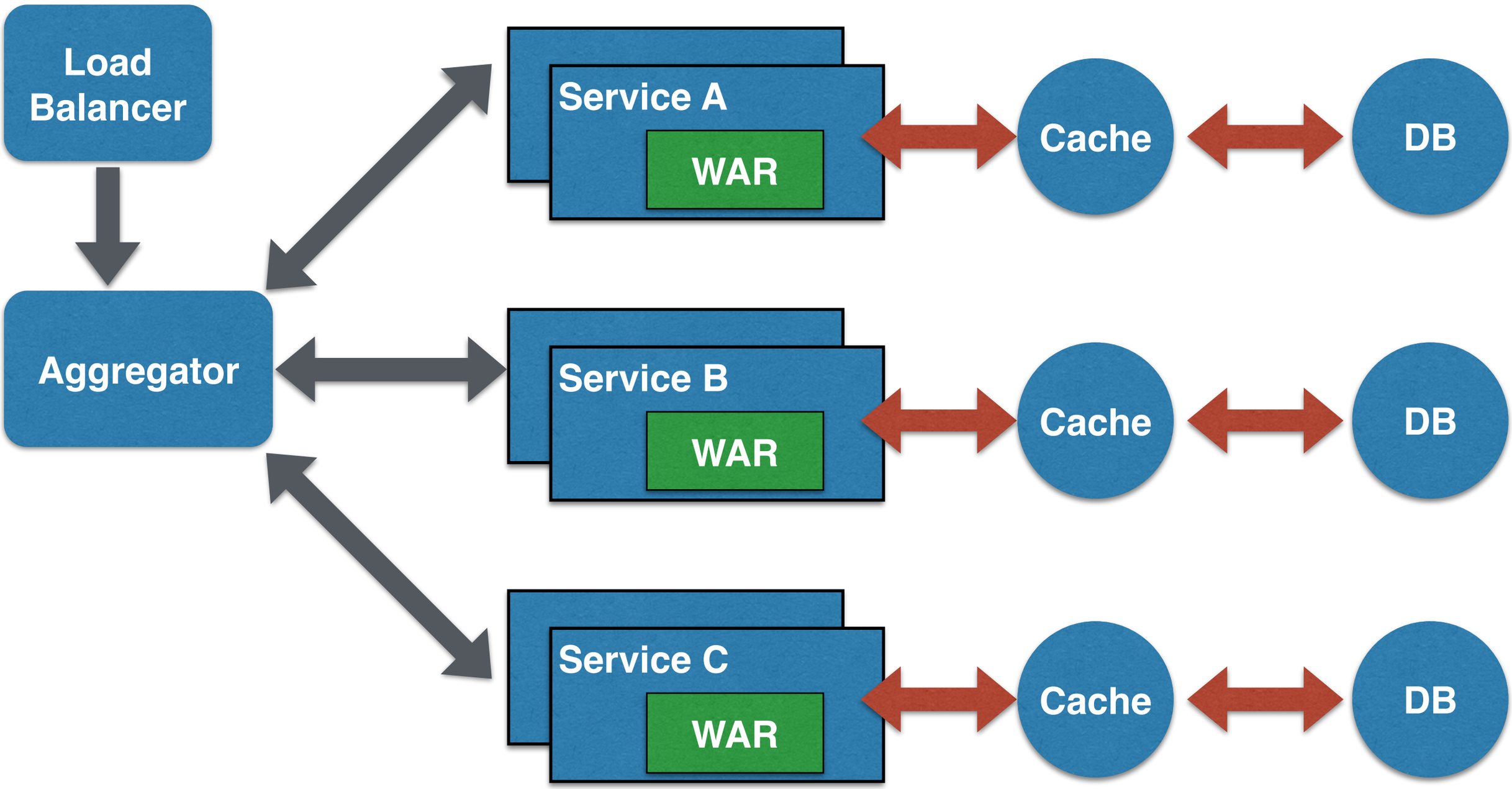
Aggregator Pattern #1



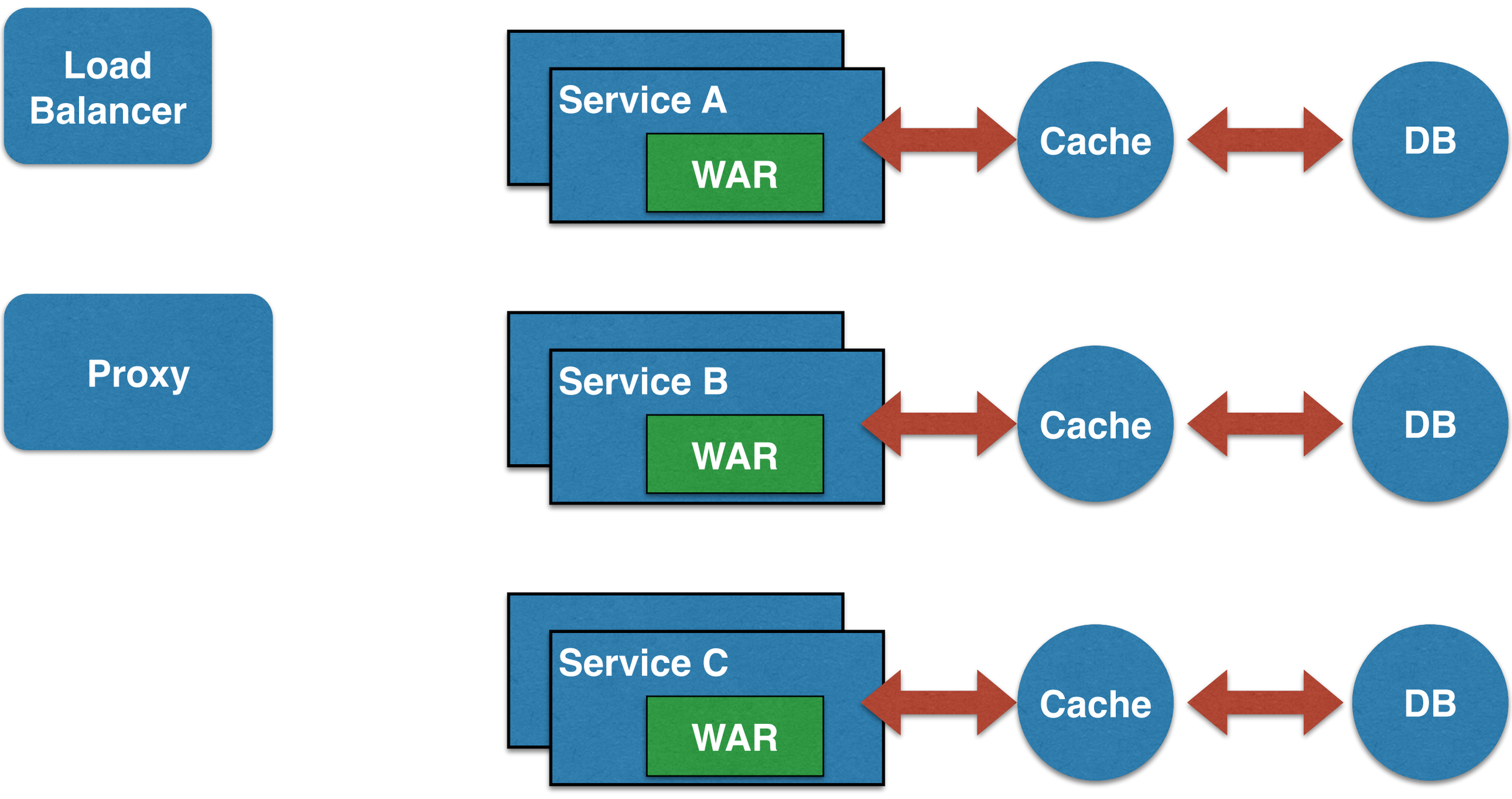
Aggregator Pattern #1



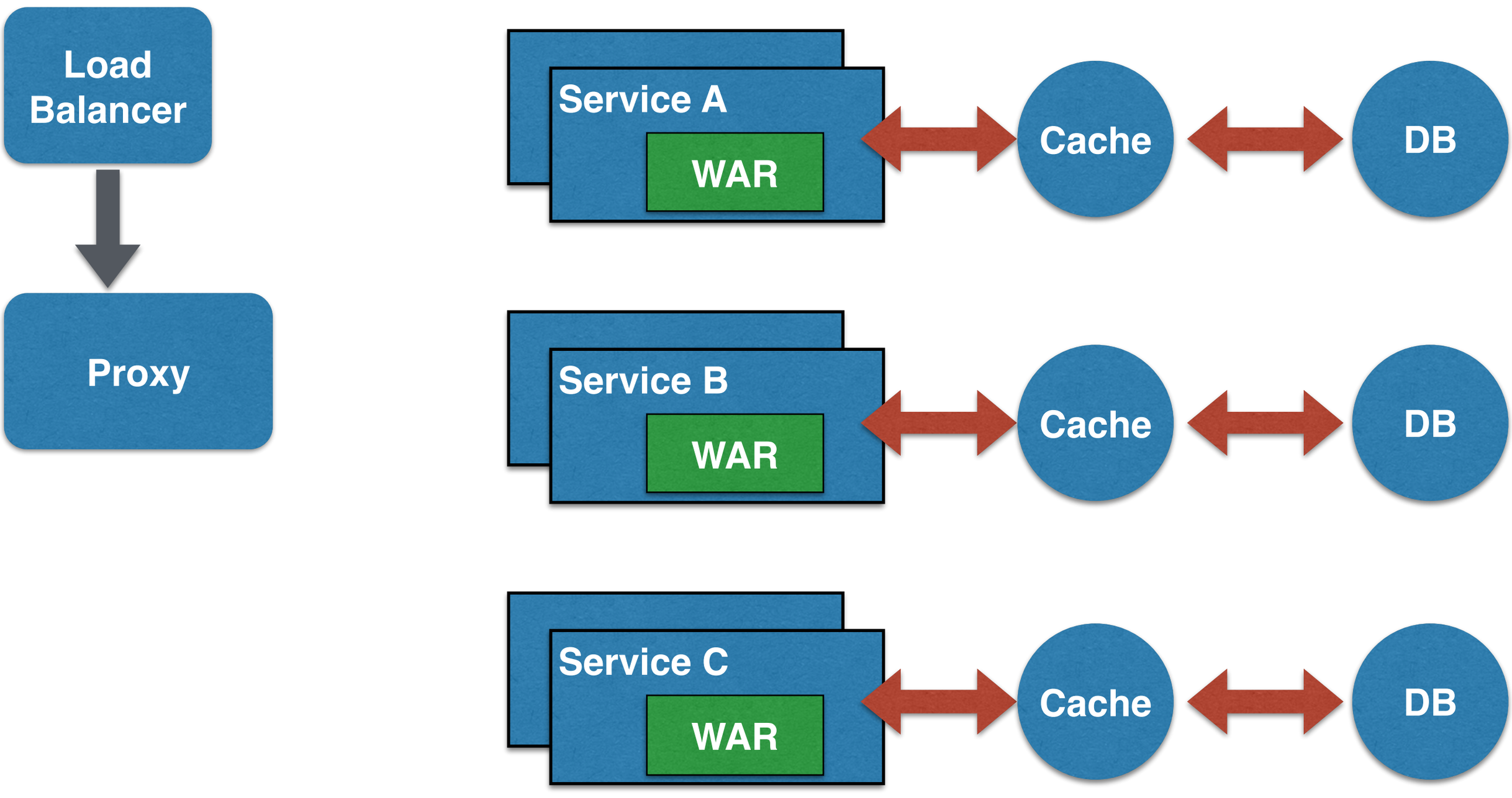
Aggregator Pattern #1



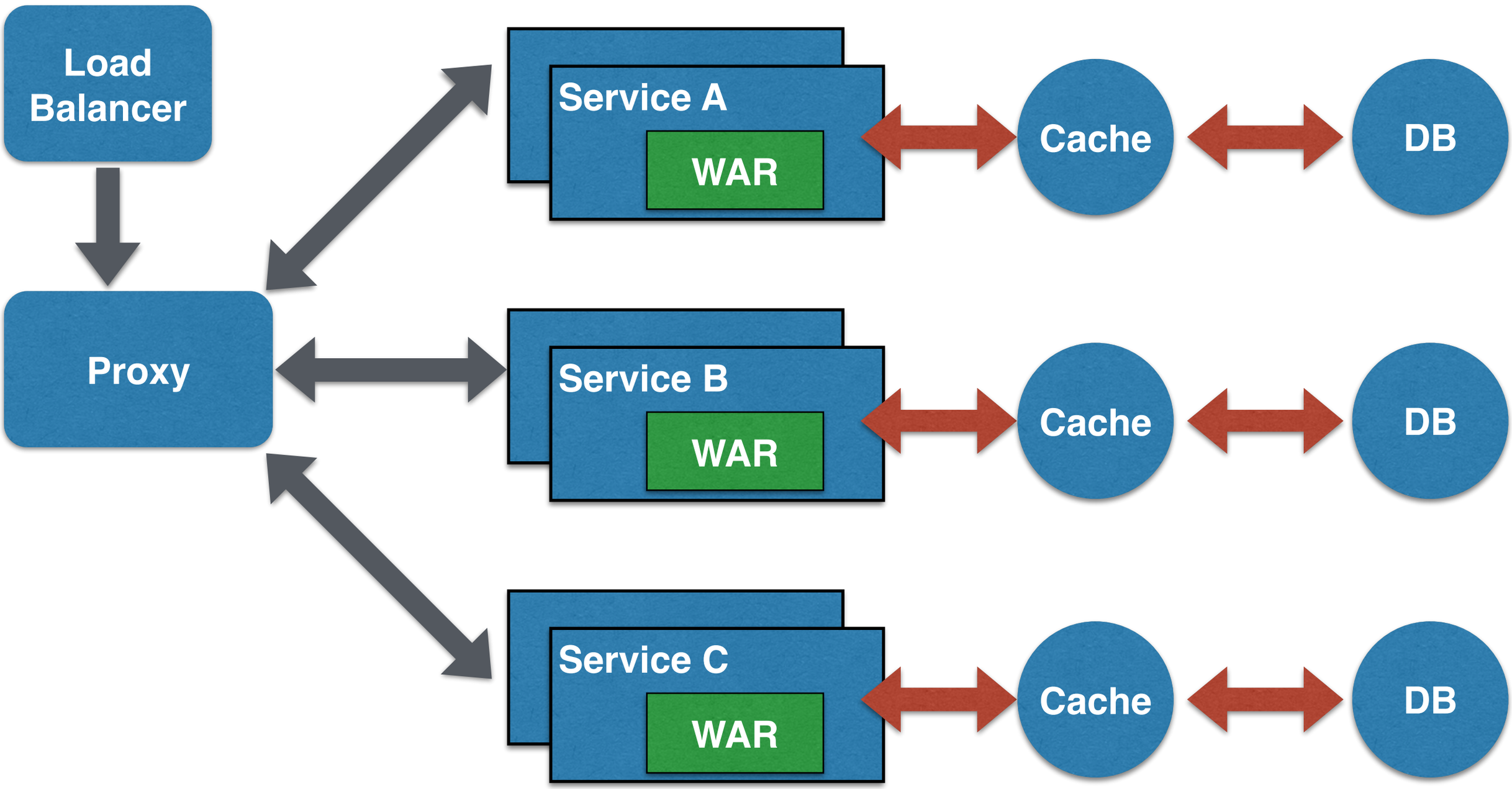
Proxy Pattern #1.5



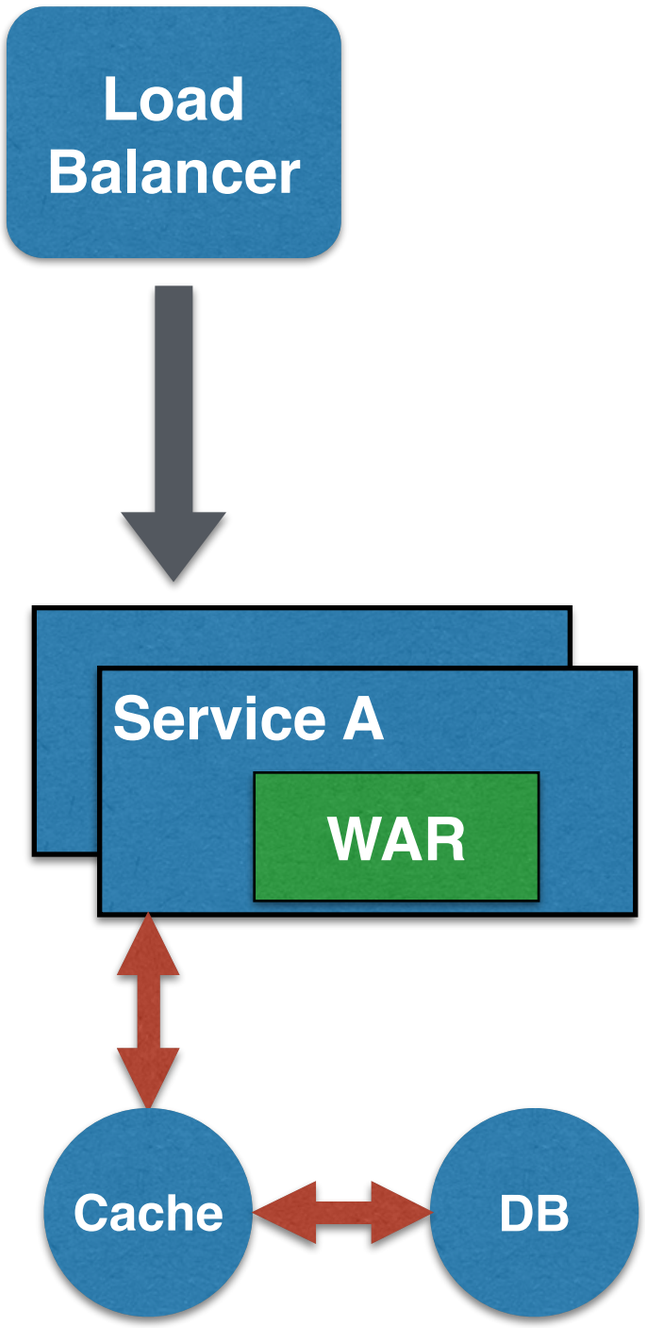
Proxy Pattern #1.5



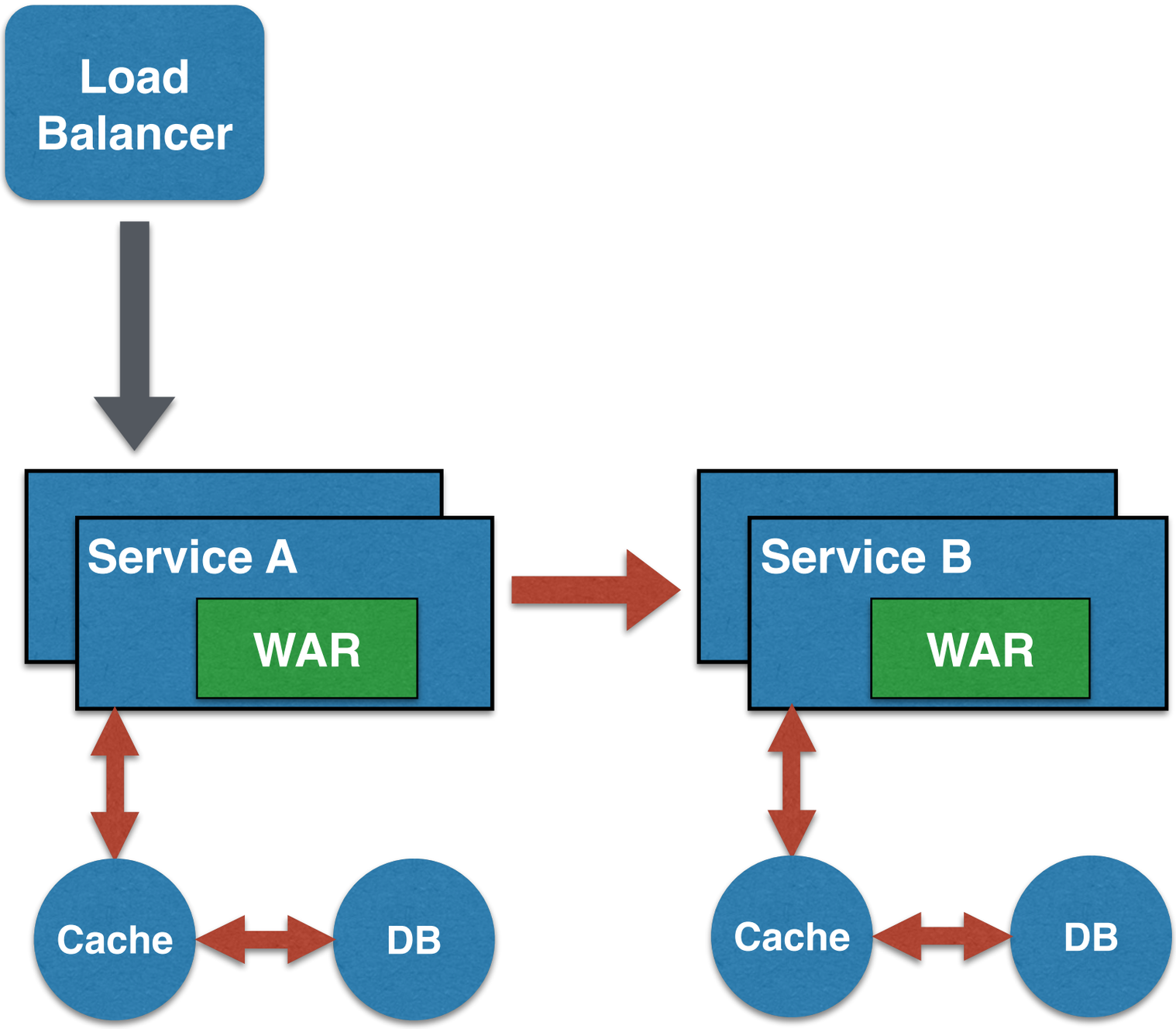
Proxy Pattern #1.5



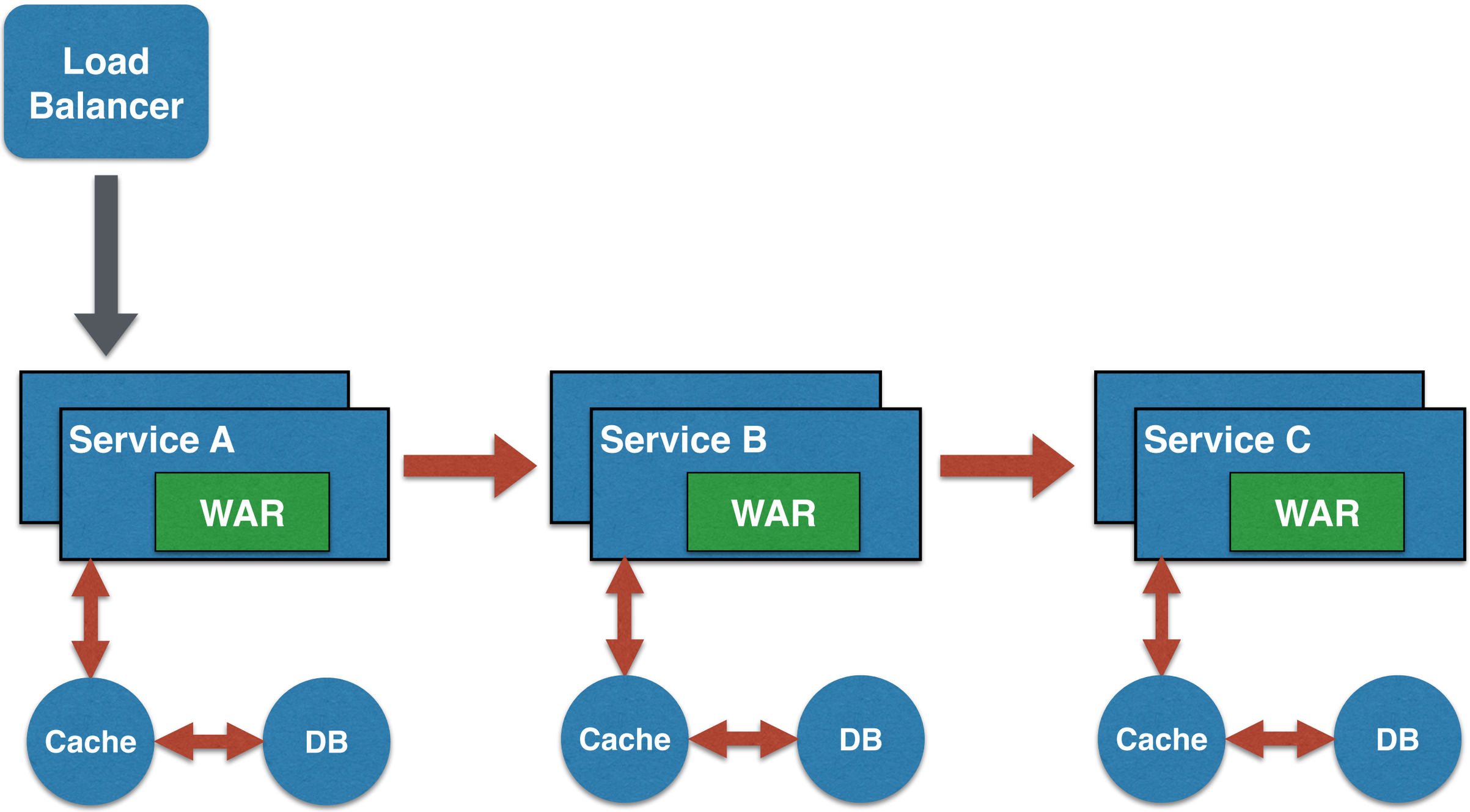
Chained Pattern #2



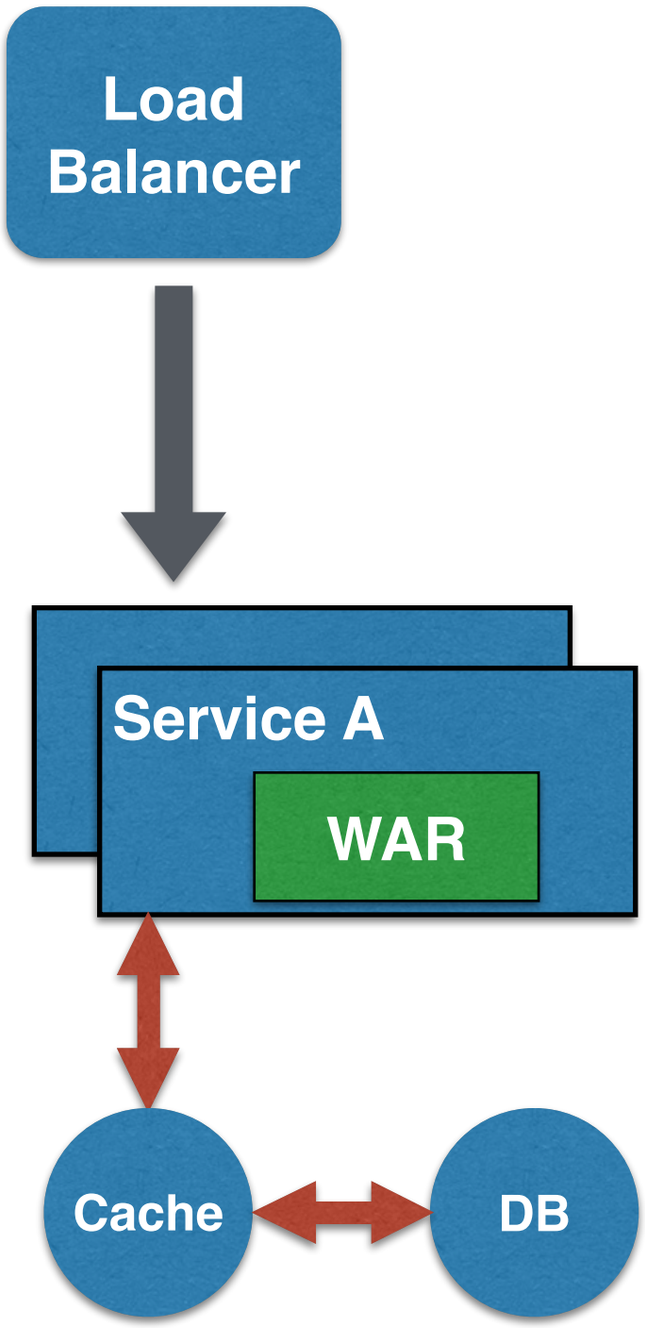
Chained Pattern #2



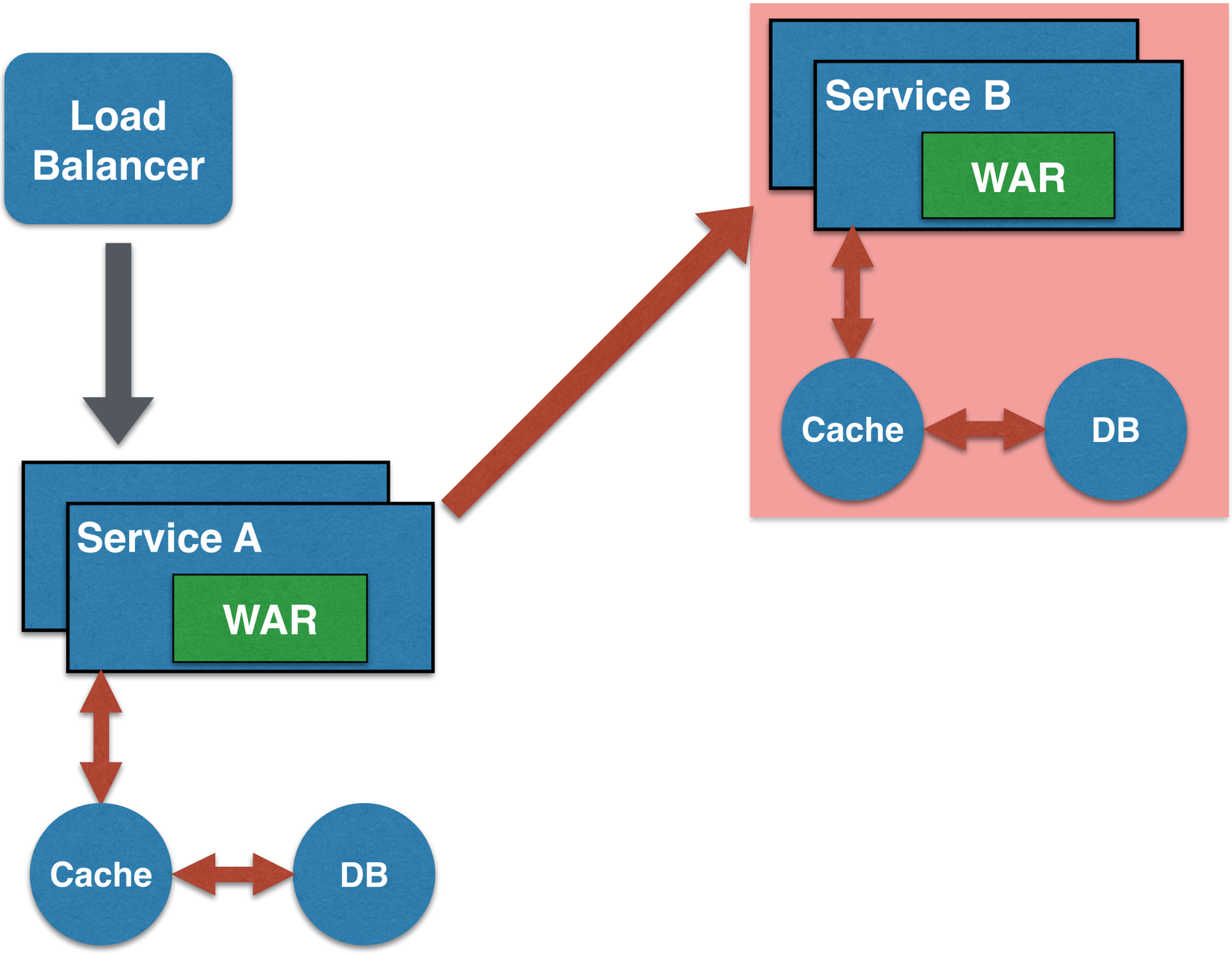
Chained Pattern #2



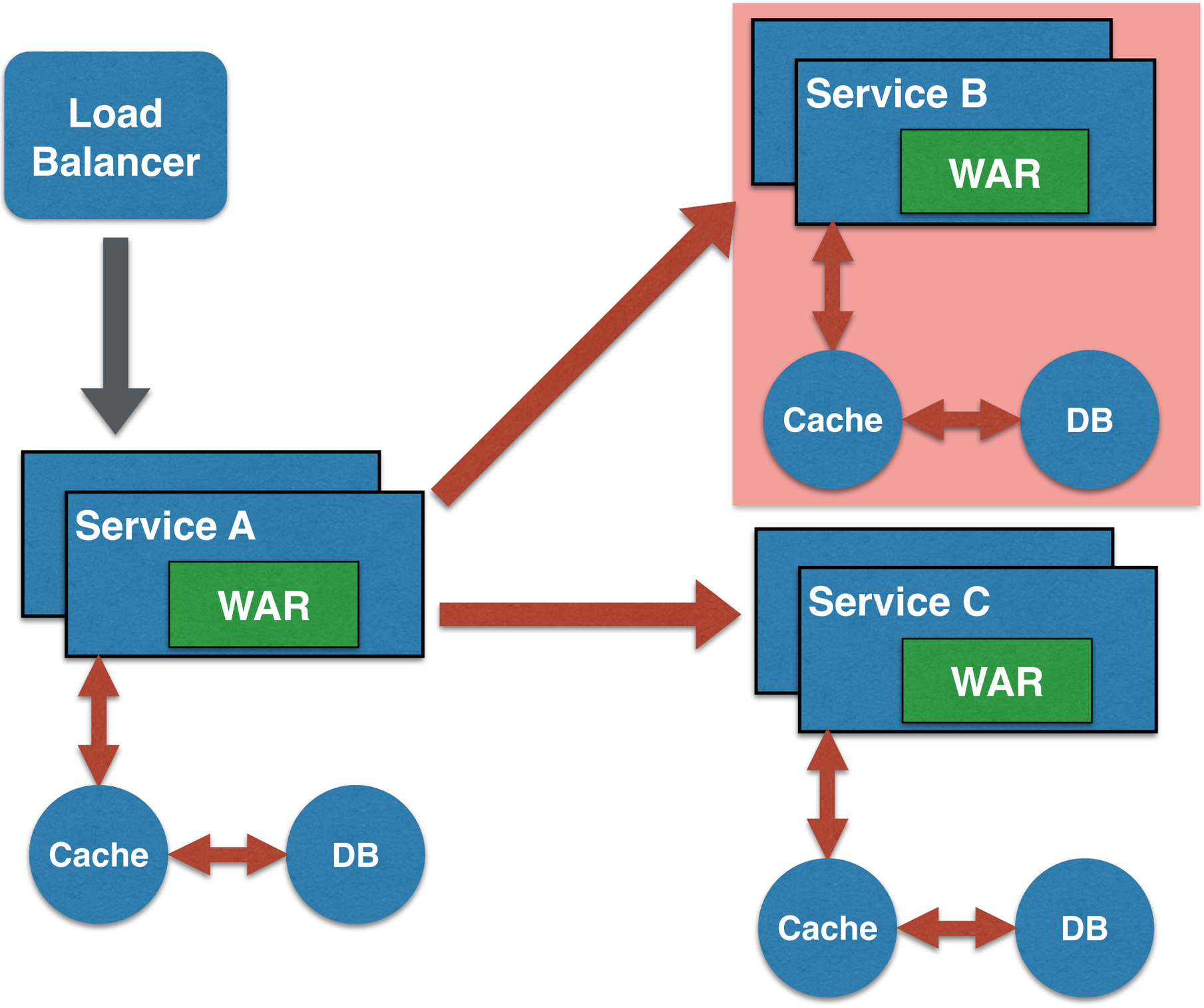
Branch Pattern #3



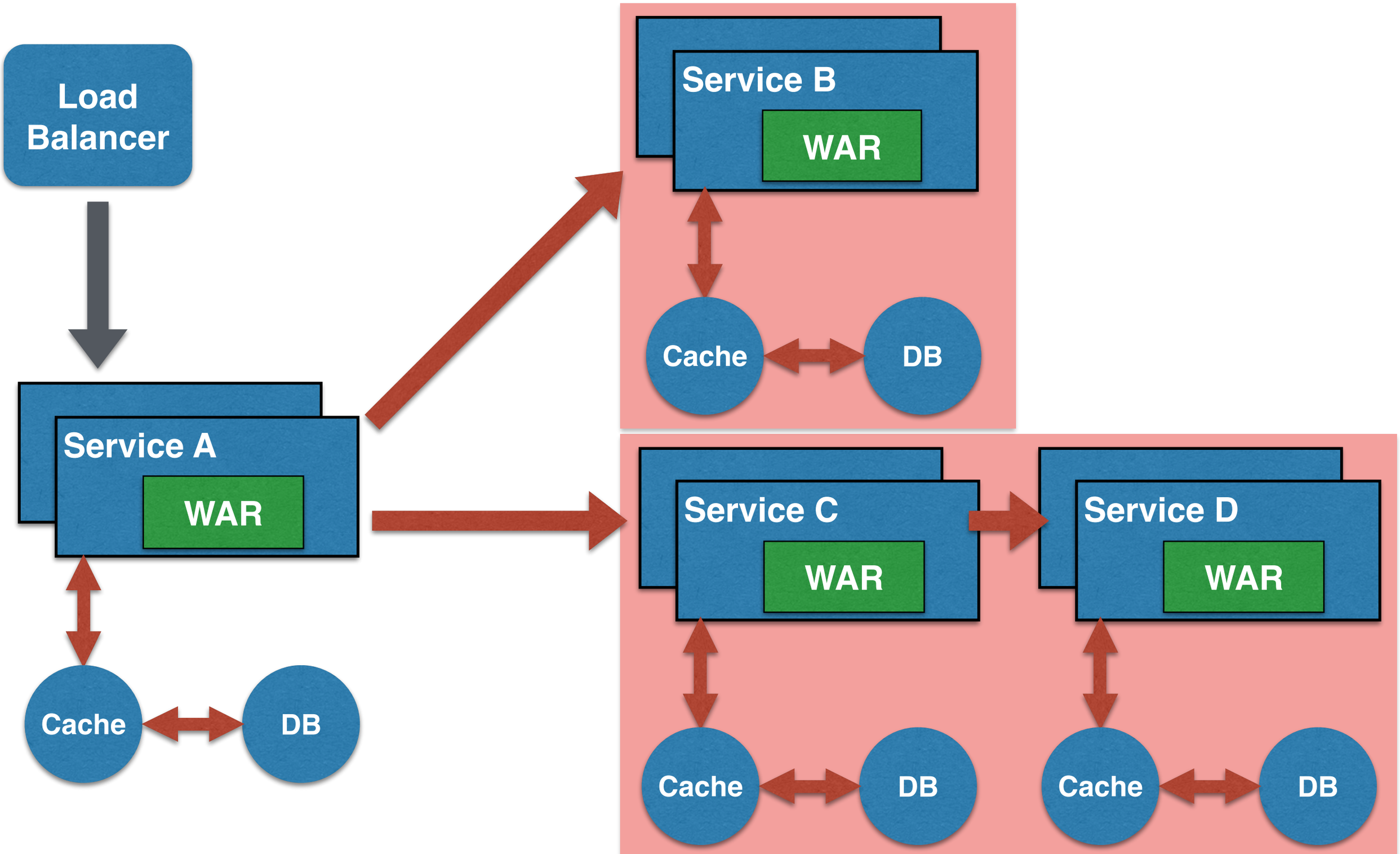
Branch Pattern #3



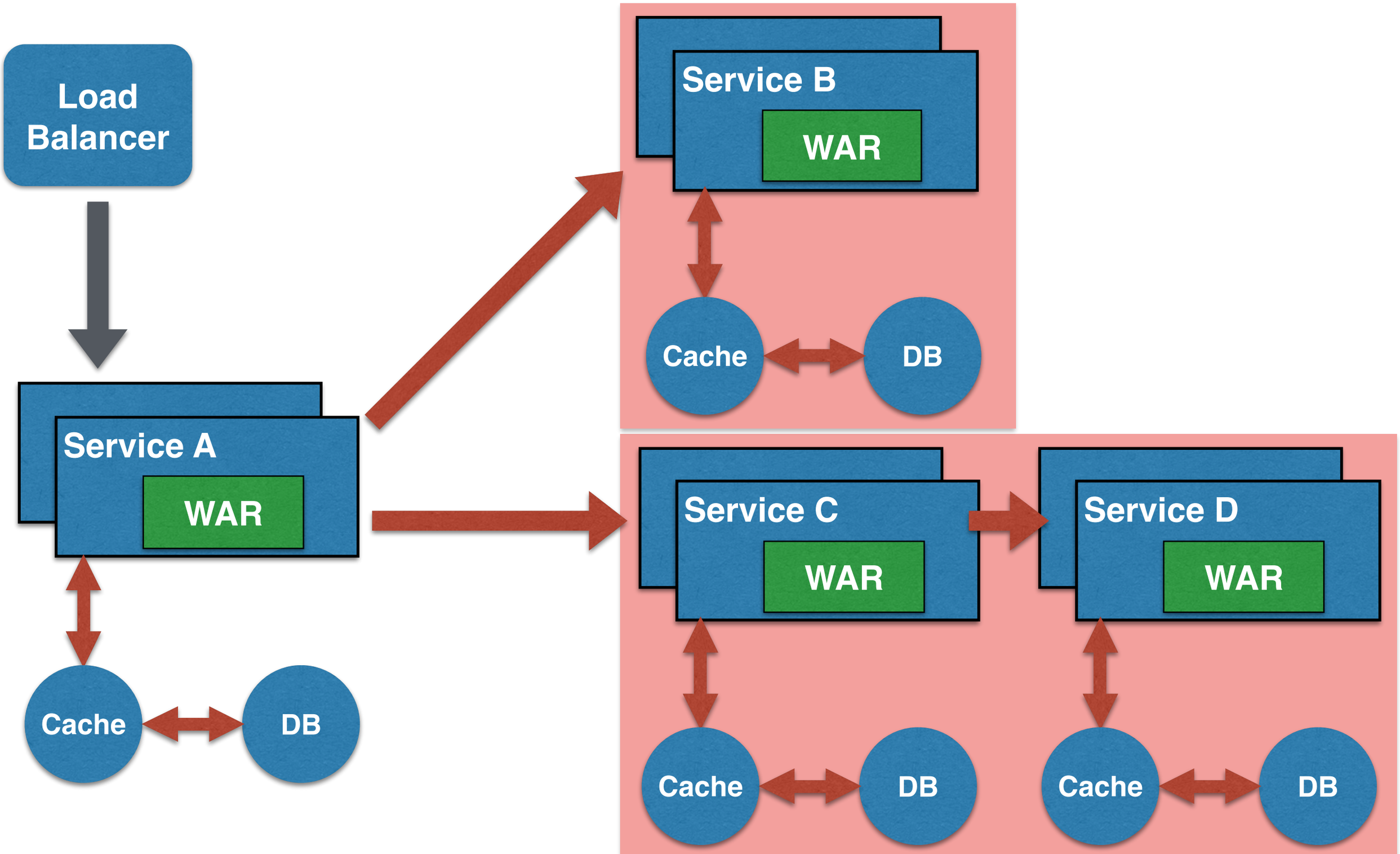
Branch Pattern #3



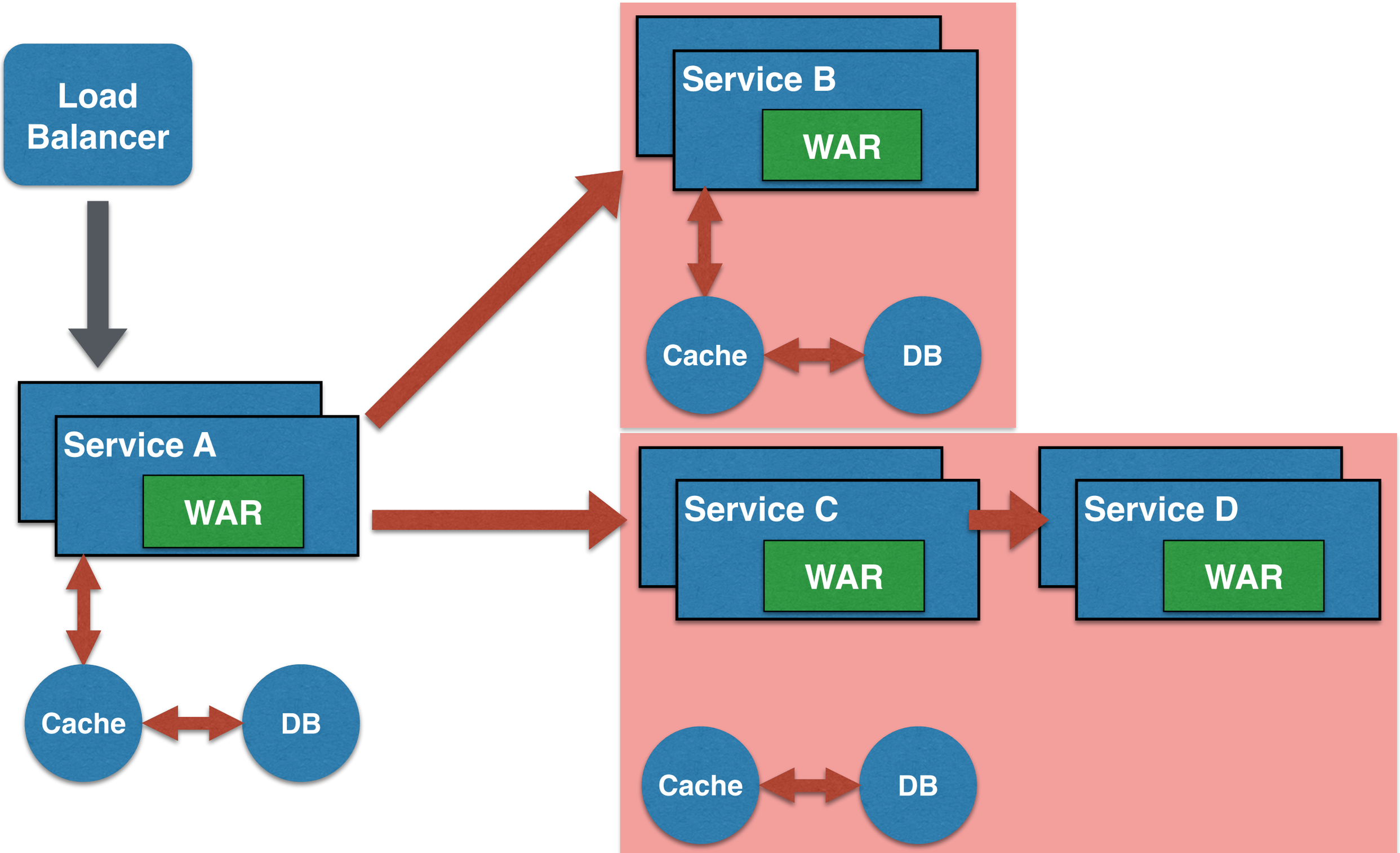
Branch Pattern #3



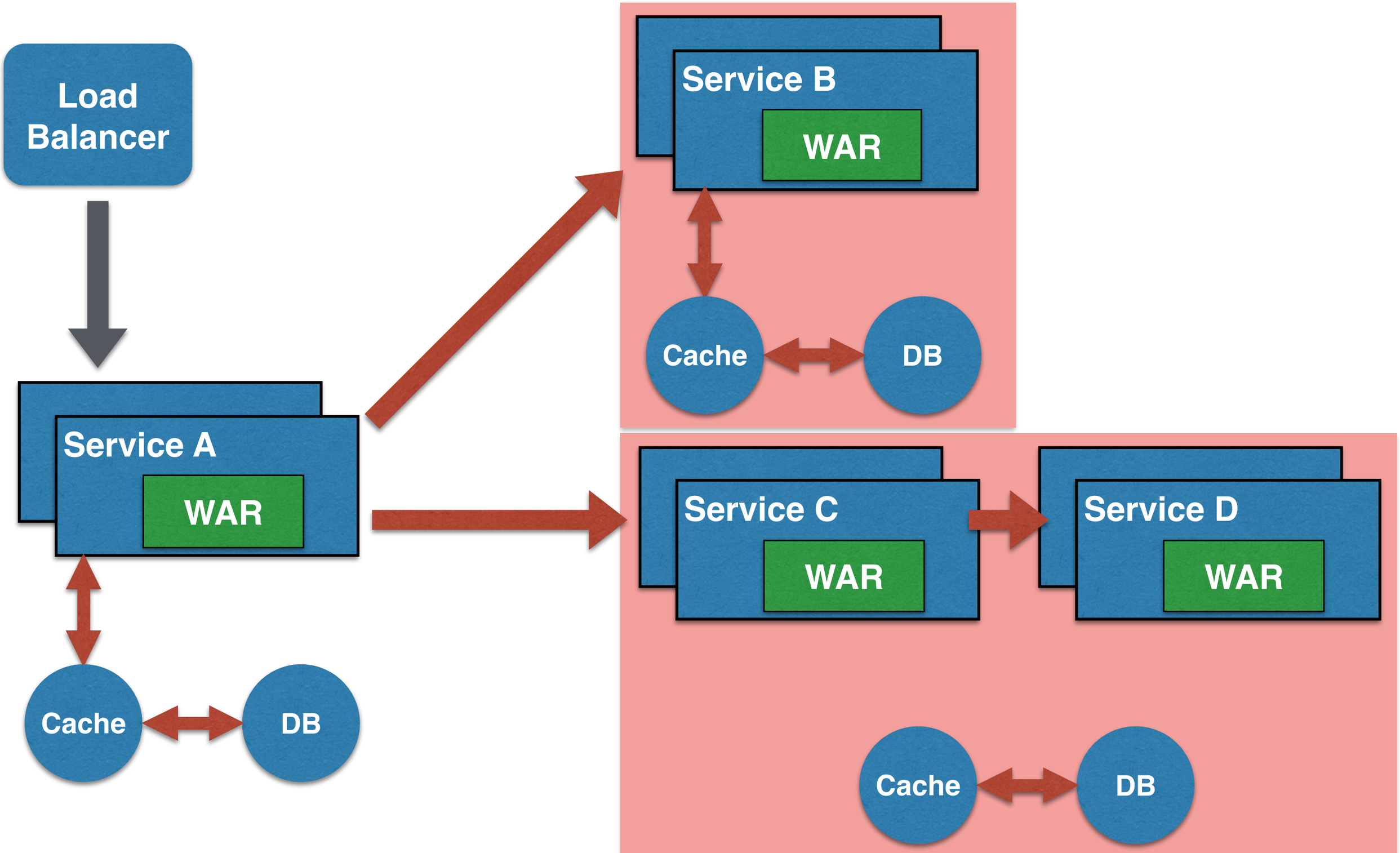
Shared Resources #4



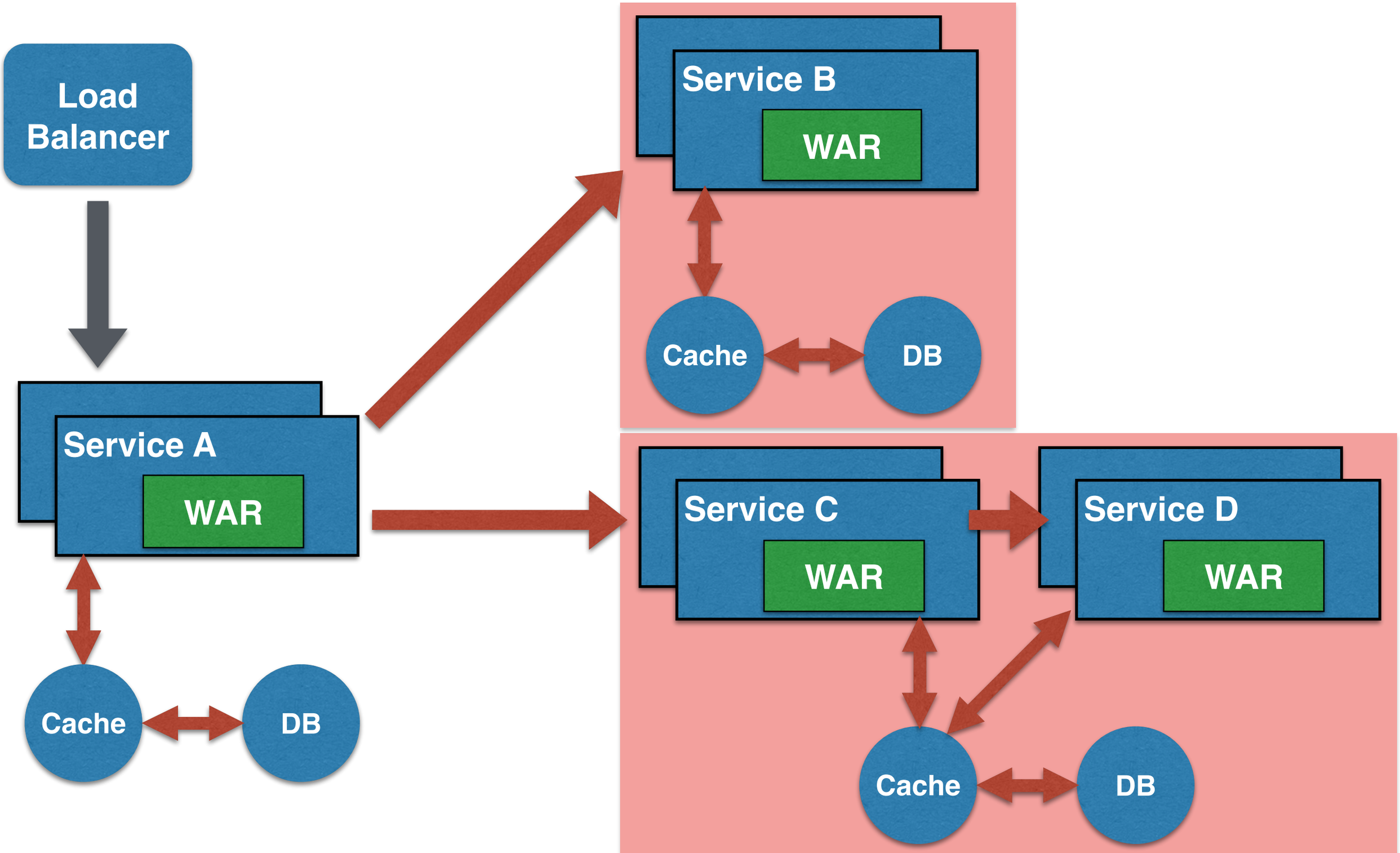
Shared Resources #4



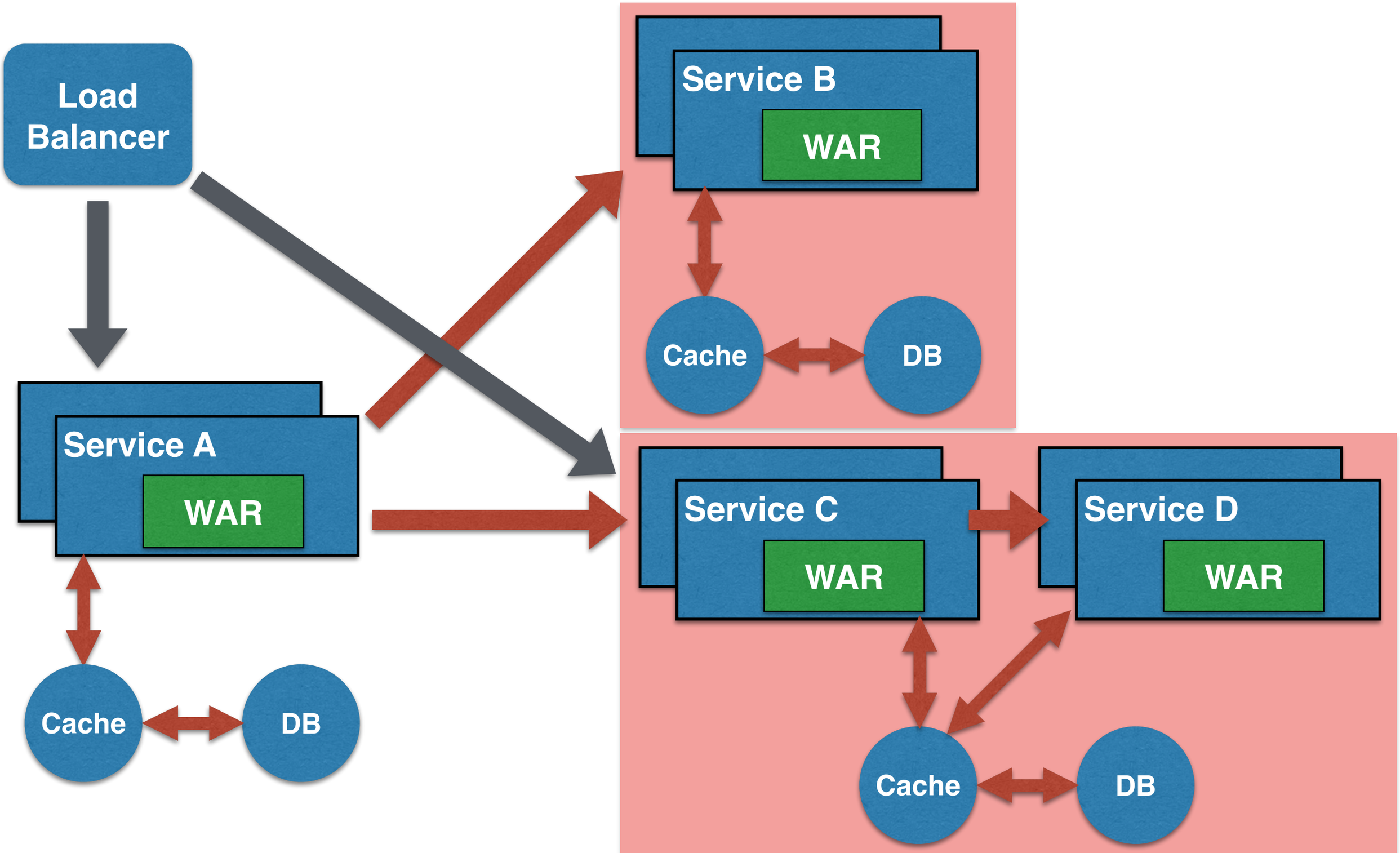
Shared Resources #4



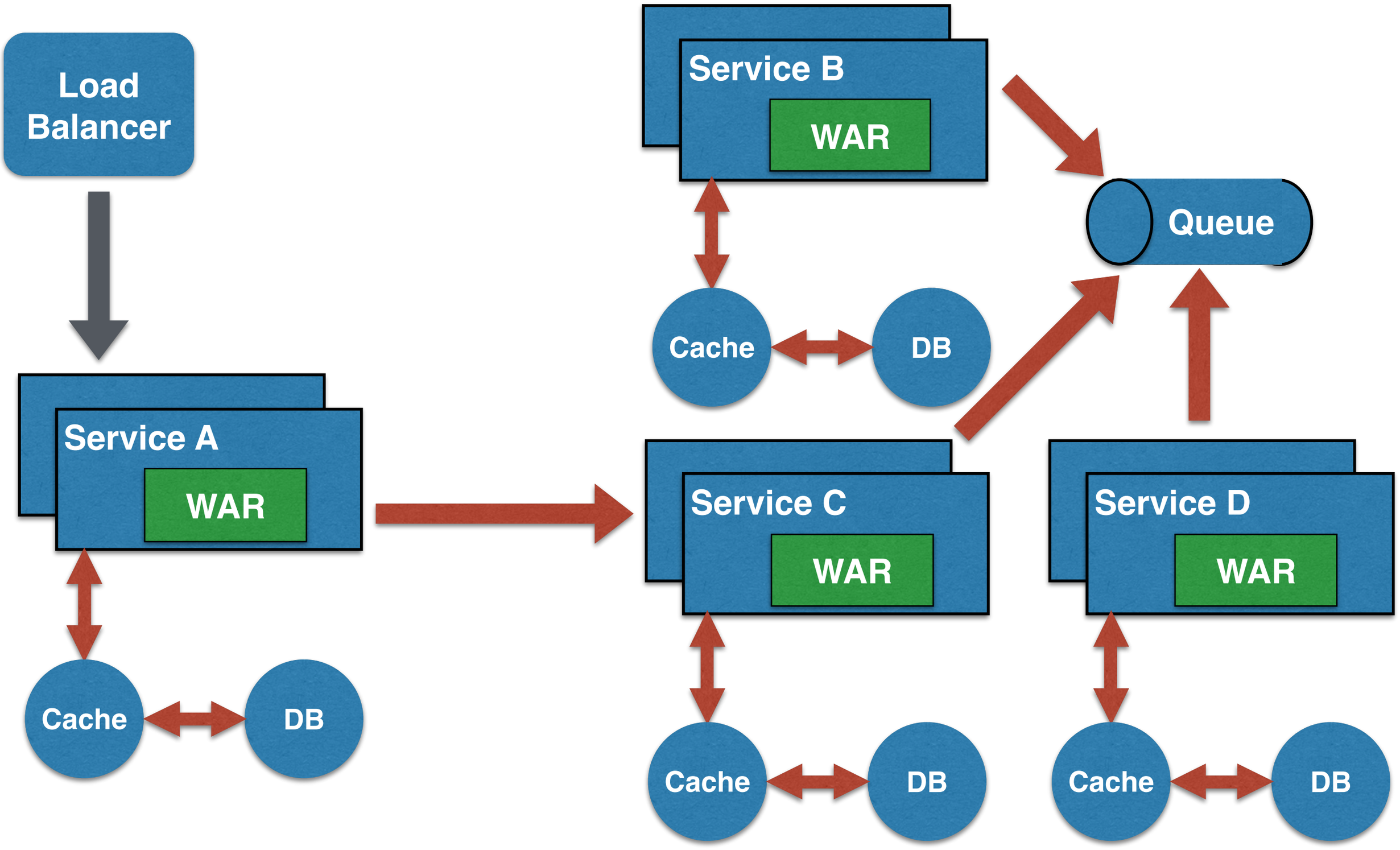
Shared Resources #4



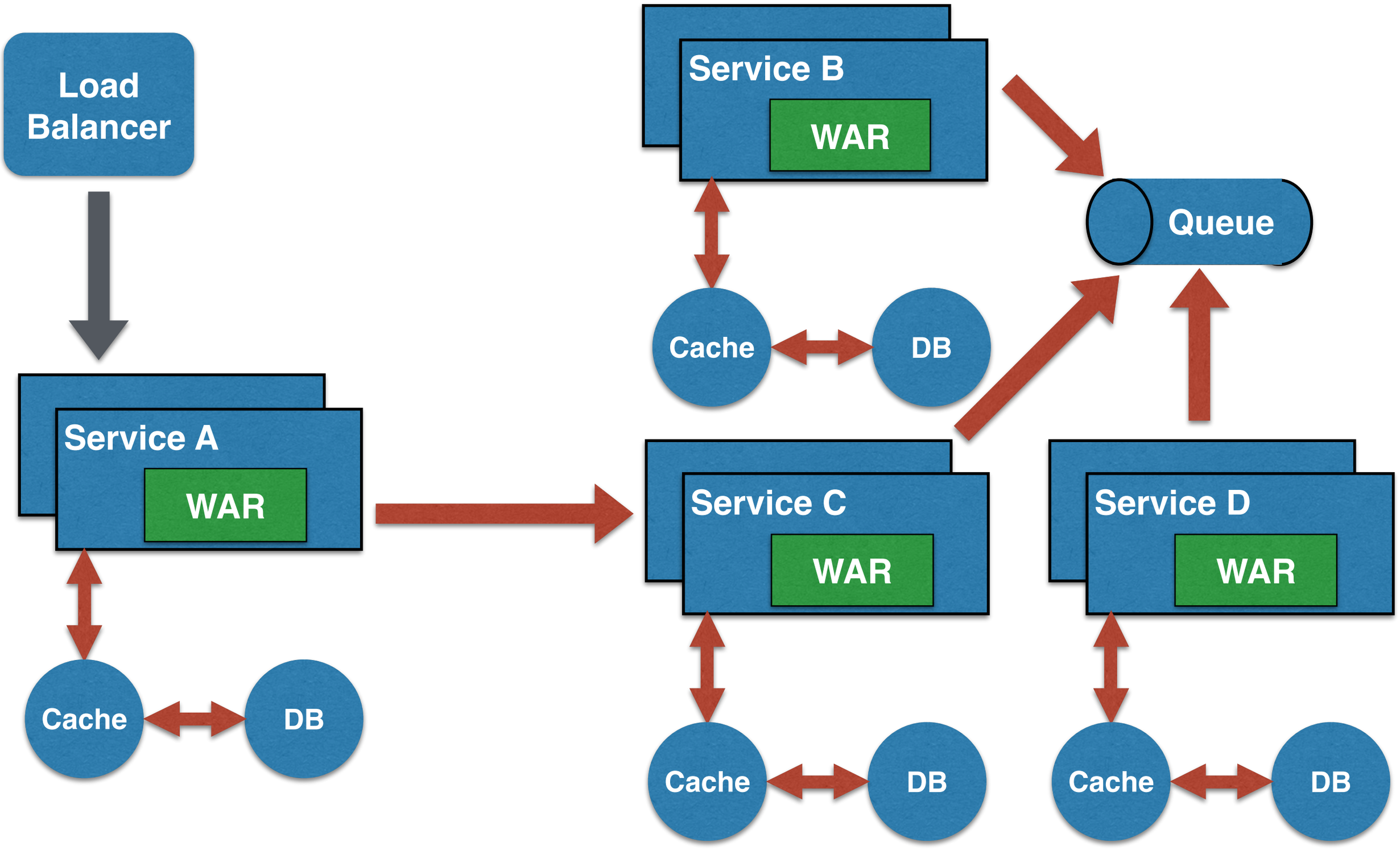
Shared Resources #4



Async Messaging #5



Async Messaging #5



Decentralized data

Decentralized data

- Each service has it owns schema, loose coupling

Decentralized data

- Each service has it owns schema, loose coupling
- Polyglot persistence

Decentralized data

- Each service has it owns schema, loose coupling
- Polyglot persistence
- Distributed transactions reduces system availability

Decentralized data

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

Decentralized data

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

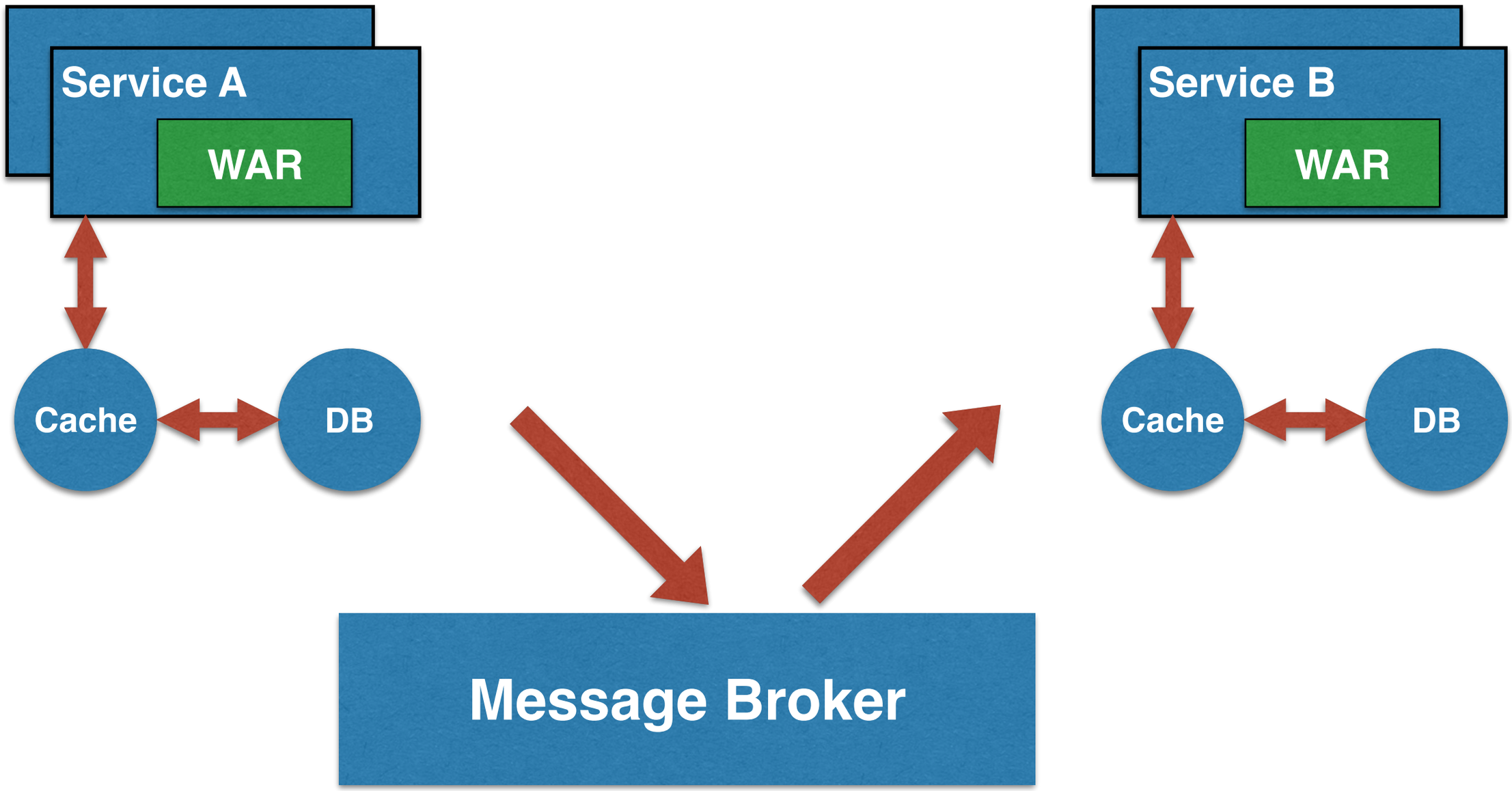
Decentralized data

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

Decentralized data

- Each service has it owns 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 μservices

- 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 *μ*services-based application.

References

- github.com/arun-gupta/docker-tutorial
- martinfowler.com/articles/microservices.html
- www.infoq.com/articles/microservices-intro

Refactor your applications using Container and Microservices

Arun Gupta, @arungupta
Red Hat