



CompSci 401: Cloud Computing

# Monolithic Applications

Prof. Ítalo Cunha



# How should a program be structured?

- Much software engineering research
  - Several trade-offs to consider
  - No clear answer

# Monolithic applications

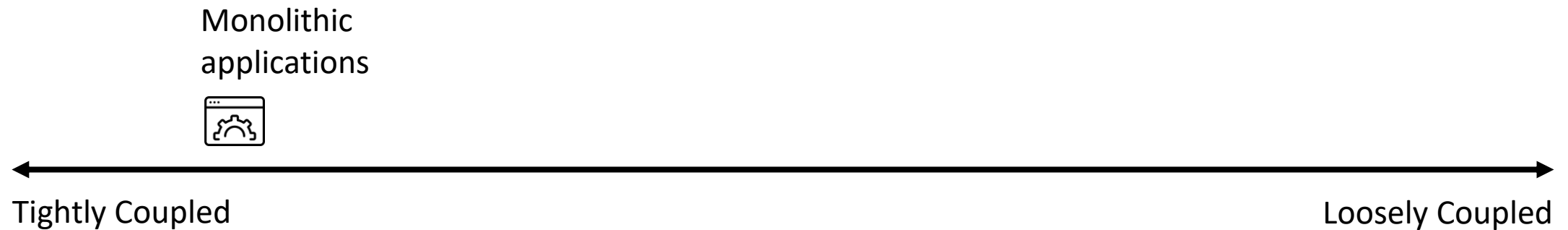
- Tight coupling between components
  - Higher performance (e.g., shared memory, function calls)
- One piece of software to install, configure, use, and update

# Monolithic applications

- Tight coupling between components
  - Higher performance (e.g., shared memory, function calls)
- One piece of software to install, configure, use, and update
- Examples
  - Games
  - Acrobat Reader
  - Word (desktop version)
  - Photoshop

# Monolithic applications

- Tight coupling between components
  - Higher performance (e.g., shared memory, function calls)
- One piece of software to install, configure, use, and update

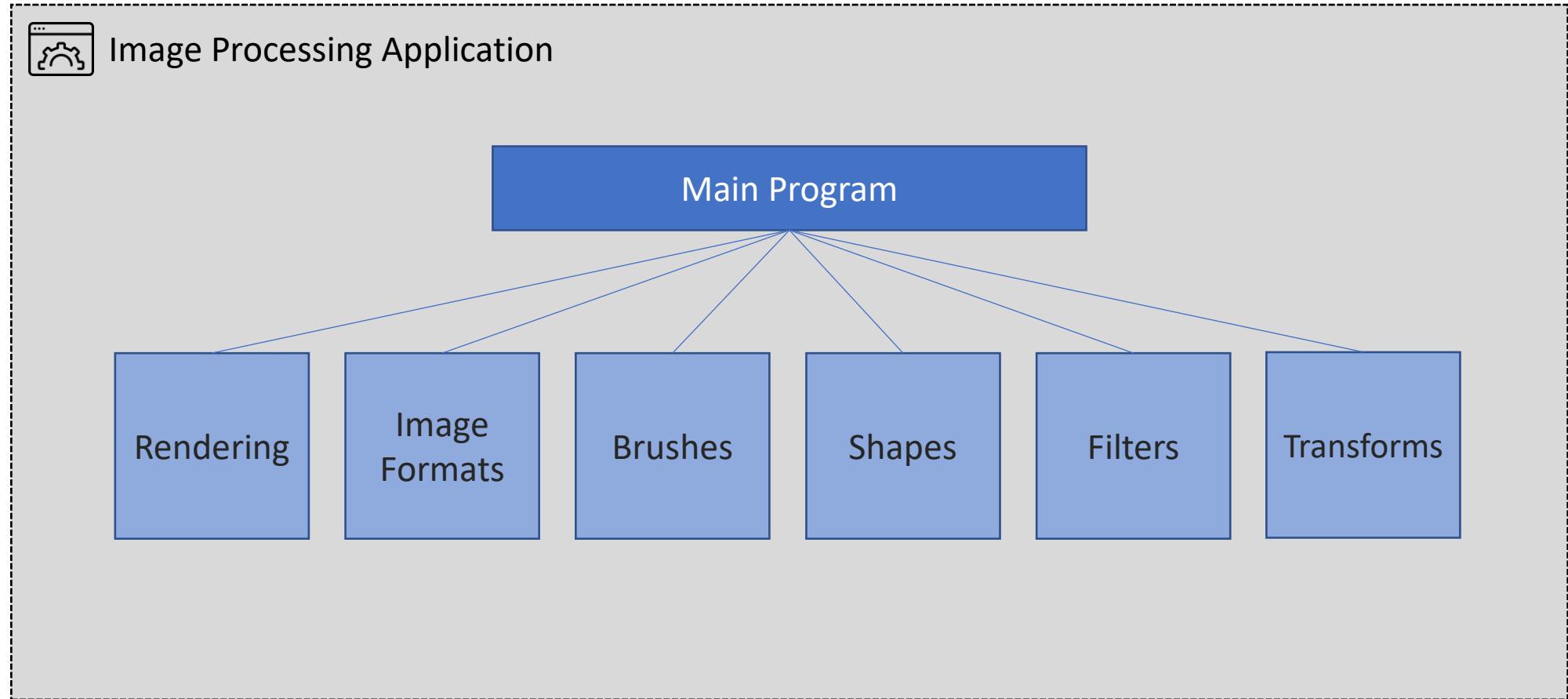


# Monolithic applications

- Tight coupling between components
  - Higher performance (e.g., shared memory, function calls)
- One piece of software to install, configure, use, and update



# Example image processing monolithic app



# Monolithic applications in a data center

- We *can* run monolithic applications in a data center
  - Tenants can rent VMs, install the application in the virtual machine, and use it
- Not the best match for cloud computing
  - Containers are more lightweight
  - Containers can be started and stopped faster than a VM



# Monolithic applications in a data center

- We *can* run monolithic applications in a data center
  - Tenants can rent VMs, install the application in the virtual machine, and use it
- Not the best match for cloud computing
  - Containers are more lightweight
  - Containers can be started and stopped faster than a VM
- Loading the whole application will waste resources
  - Most users will only use a fraction of the functionalities:
    - User A may be rotating images
    - User B may be removing red eyes
    - User C may be recalibrating colors
    - User D may be converting file formats



# CompSci 401: Cloud Computing

# Microservices

Prof. Ítalo Cunha



# Microservice approach

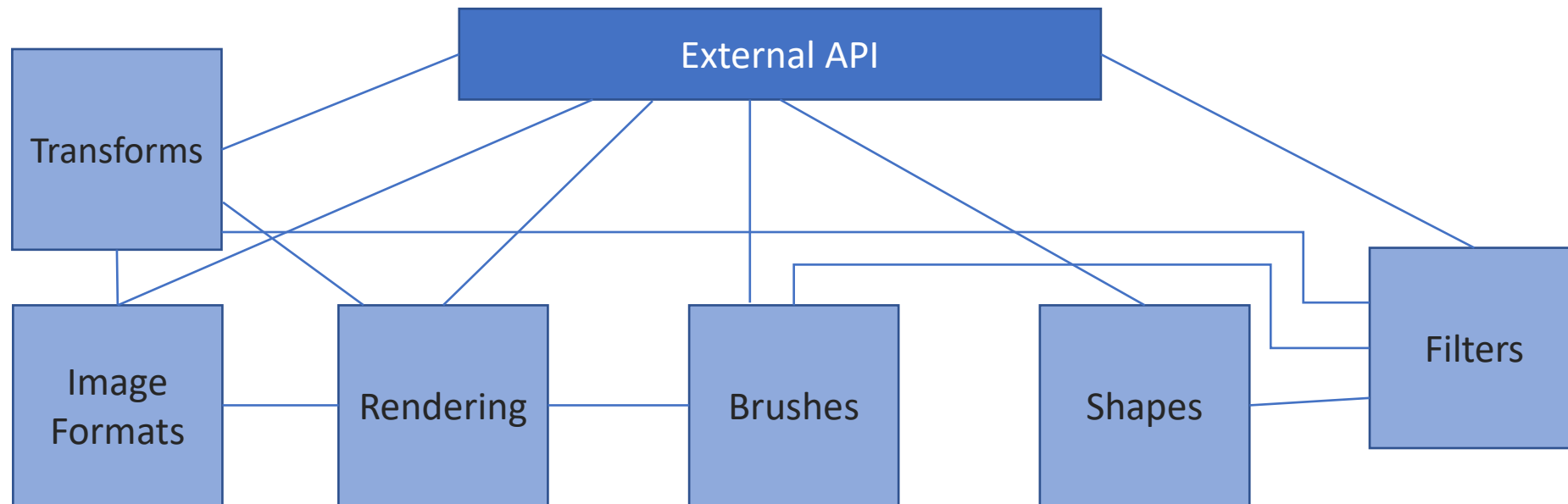
- Split an application's functionality across different programs
  - Each program is small and handles a single or a few related functions
- A monolithic application can be *disaggregated* into microservices

# Microservice approach

- Split an application's functionality across different programs
  - Each program is small and handles a single or a few related functions
- A monolithic application can be *disaggregated* into microservices



Image Processing Application



# Microservice approach

- Split an application's functionality across different programs
  - Each program is small and handles a single or a few related functions
- A monolithic application can be *disaggregated* into microservices



# Advantages of microservices for development

- Smaller scope
  - Do one task and to it well
  - Requires decomposition of problem into parts
- Smaller teams
- Better modularity
- Less complexity
- Choice of programming language
- Extensive testing

# Advantages of microservices for development

- Smaller scope
- Smaller teams
  - Engineers can understand the microservice in its entirety
  - More uniform code and better manageability
  - Less effort in coordinating large teams
- Better modularity
- Less complexity
- Choice of programming language
- Extensive testing

# Advantages of microservices for development

- Smaller scope
- Smaller teams
- Better modularity
  - Define clean interfaces for interaction between microservices
  - Interface only over the network
  - One team cannot change another microservice
- Less complexity
- Choice of programming language
- Extensive testing



# Advantages of microservices for development

- Smaller scope
- Smaller teams
- Better modularity
- Less complexity
  - Complexity may still exist in a microservice, but it's well contained and isolated
- Choice of programming language
- Extensive testing

# Advantages of microservices for development

- Smaller scope
- Smaller teams
- Better modularity
- Less complexity
- Choice of programming language
  - Microservices are independent and can be implemented in any language
- Extensive testing

# Advantages of microservices for development

- Smaller scope
- Smaller teams
- Better modularity
- Less complexity
- Choice of programming language
- Extensive testing
  - Microservices are small, easier to get high coverage in tests
  - Well-defined interfaces (over the network) implies less interactions to check

# Advantages of microservices for operations

- Rapid deployment
  - Small size implies implementation, test, and deployment are quicker
- Improve fault isolation
  - Failures likely confined to one microservice
  - Easier to identify and troubleshoot
- Better control of scaling
  - Finer granularity than monolithic applications
  - Each microservice can be scaled separately
- Compatibility with containers and orchestration systems
- Independent upgrade of each service
  - Update rollout independent of other services
  - Other services can keep running unchanged during upgrade



CompSci 401: Cloud Computing

# Possible Disadvantages of Microservices

Prof. Ítalo Cunha



# Possible disadvantages of microservices

- Cascading errors
  - One failing microservices may induce failures in other microservices
    - For example, excessive requests
  - One failed microservice may be used by many others
    - May induce failures in other microservices
- Management complexity
- Duplication of functionality and overlap
- Replication of data and transmission overhead
- Increased attack surface
- Workforce training

<https://sre.google/books/>

# Possible disadvantages of microservices

- Cascading errors
- Management complexity
  - Interactions between hundreds of microservices get complex
  - Management also becomes more complex and costly
- Duplication of functionality and overlap
- Replication of data and transmission overhead
- Increased attack surface
- Workforce training

# Possible disadvantages of microservices

- Cascading errors
- Management complexity
- Duplication of functionality and overlap
  - Teams might not be aware that functionality exist
  - Duplication increases complexity
- Replication of data and transmission overhead
- Increased attack surface
- Workforce training



# Possible disadvantages of microservices

- Cascading errors
- Management complexity
- Duplication of functionality and overlap
- Replication of data and transmission overhead
  - No shared memory or global variables
  - Requests, responses, and data is transmitted over the network
- Increased attack surface
- Workforce training

# Possible disadvantages of microservices

- Cascading errors
- Management complexity
- Duplication of functionality and overlap
- Replication of data and transmission overhead
- Increased attack surface
  - Microservices are easier to secure, but each microservice is a possible point of attack
- Workforce training

# Possible disadvantages of microservices

- Cascading errors
- Management complexity
- Duplication of functionality and overlap
- Replication of data and transmission overhead
- Increased attack surface
- Workforce training
  - Developing and operating microservices requires complementary skills



CompSci 401: Cloud Computing

# Microservice Granularity

Prof. Ítalo Cunha



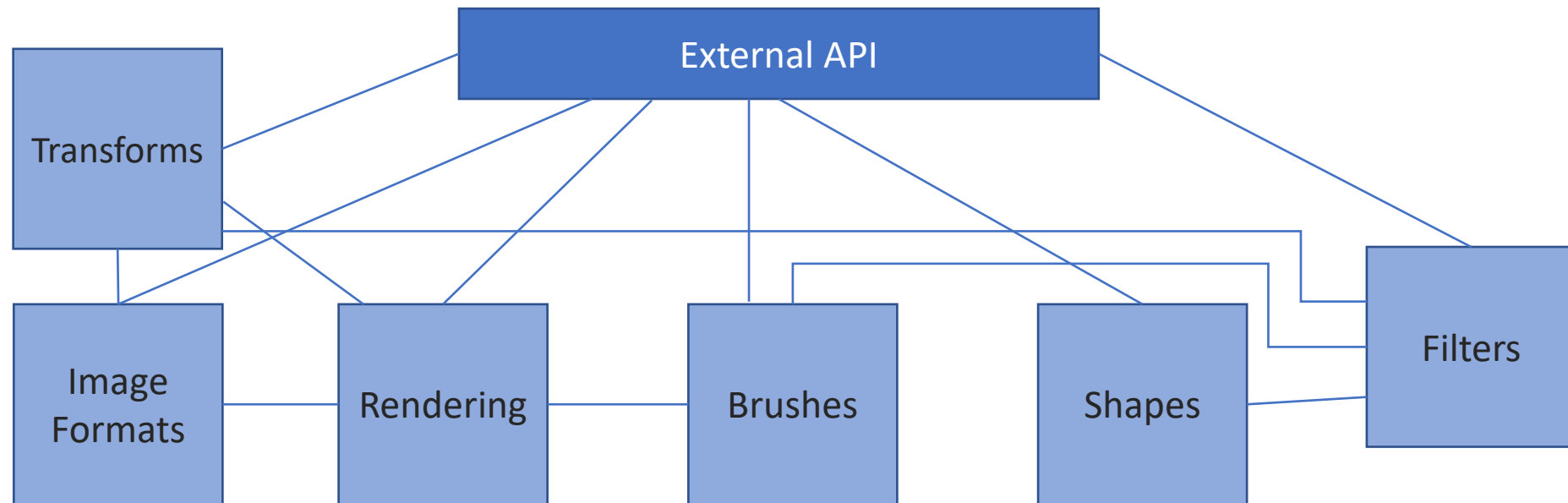
# How much functionality in a microservice?

- A functionality may be broken down into different components
- Multiple implementation decisions are possible

# How to implement the **transforms** service

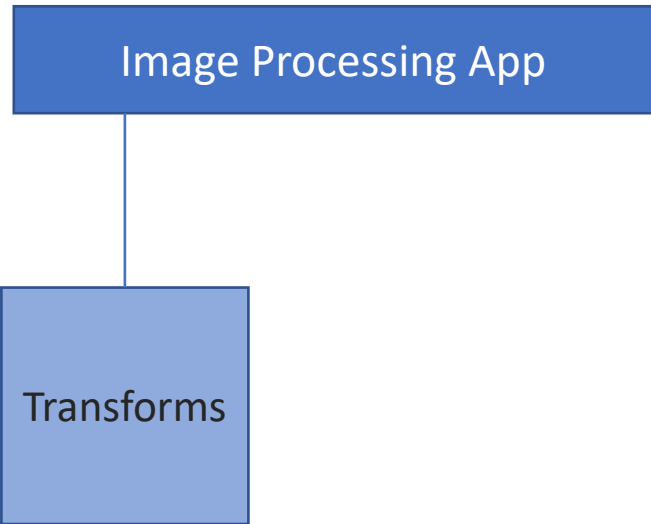


Image Processing Application



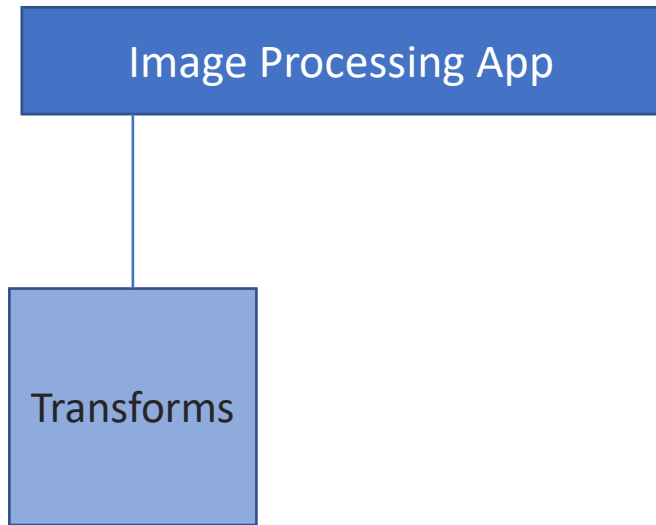
# How to implement the **transforms** service


 Image Processing Application

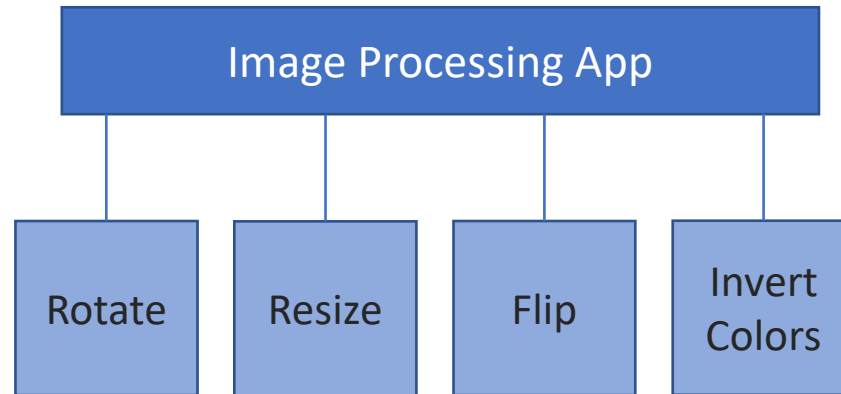


# How to implement the **transforms** service

 Image Processing Application




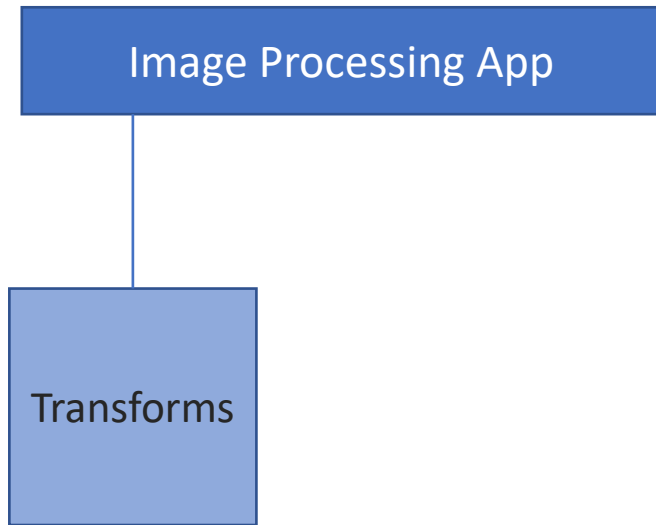
 Image Processing Application




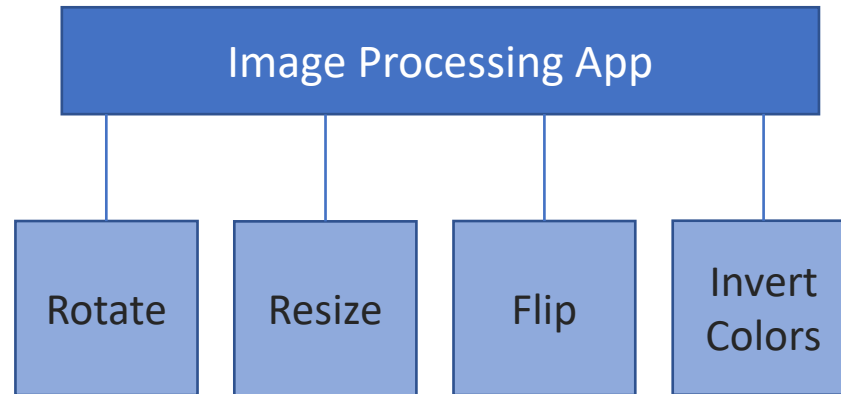



# How to implement the **transforms** service

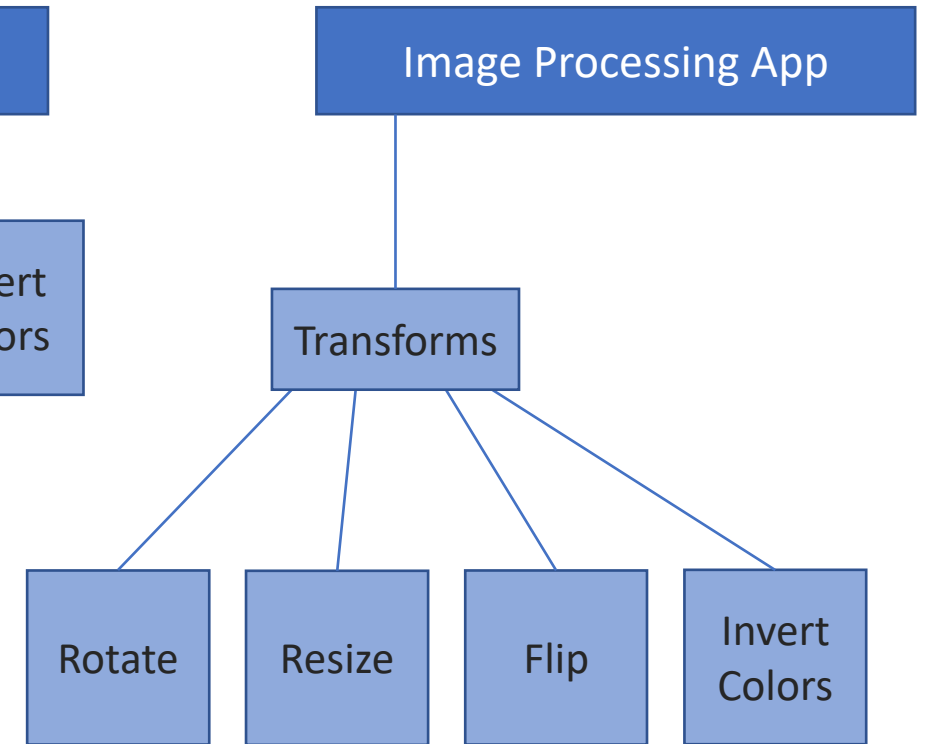
 Image Processing Application



 Image Processing Application



 Image Processing Application



# Microservices can have different granularities



# Microservices can have different granularities

Trade-off between microservice granularity and management complexity



# Heuristics for sizing microservices

- Business process modeling
  - Identify how applications are used and steps in the workflow
  - Try to make one microservice per step
- Identification of common functionality
  - Try to make general microservices that can be used broadly
  - Design interface that supports many different use cases
- Adaptative refactoring
  - Microservices are small, so easier to iterate
  - Can refactor to split or join microservices, as well as add functionalities



CompSci 401: Cloud Computing

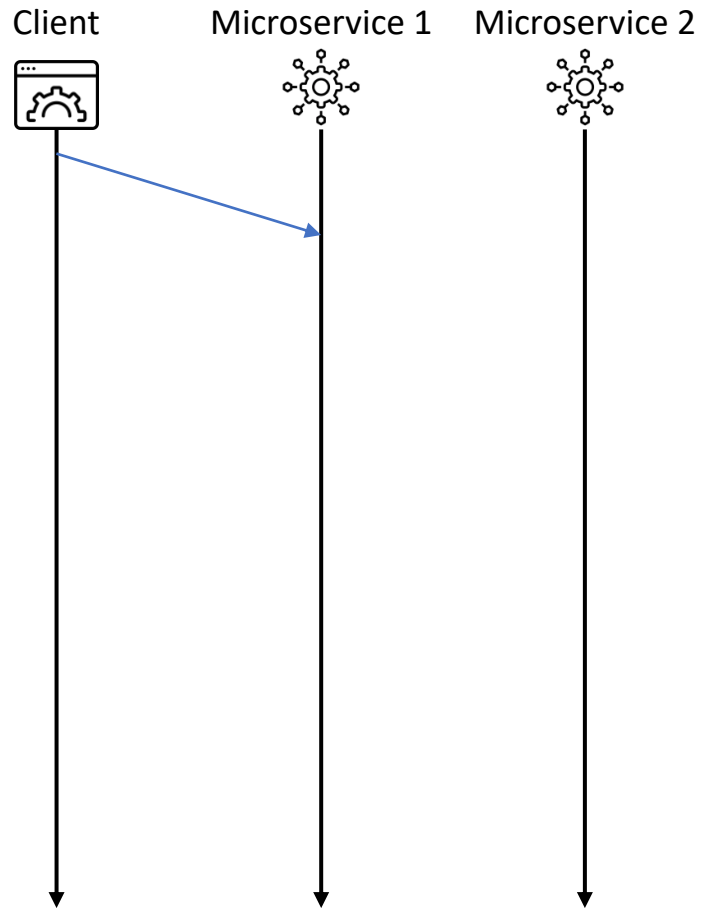
# Microservice Communications

Prof. Ítalo Cunha



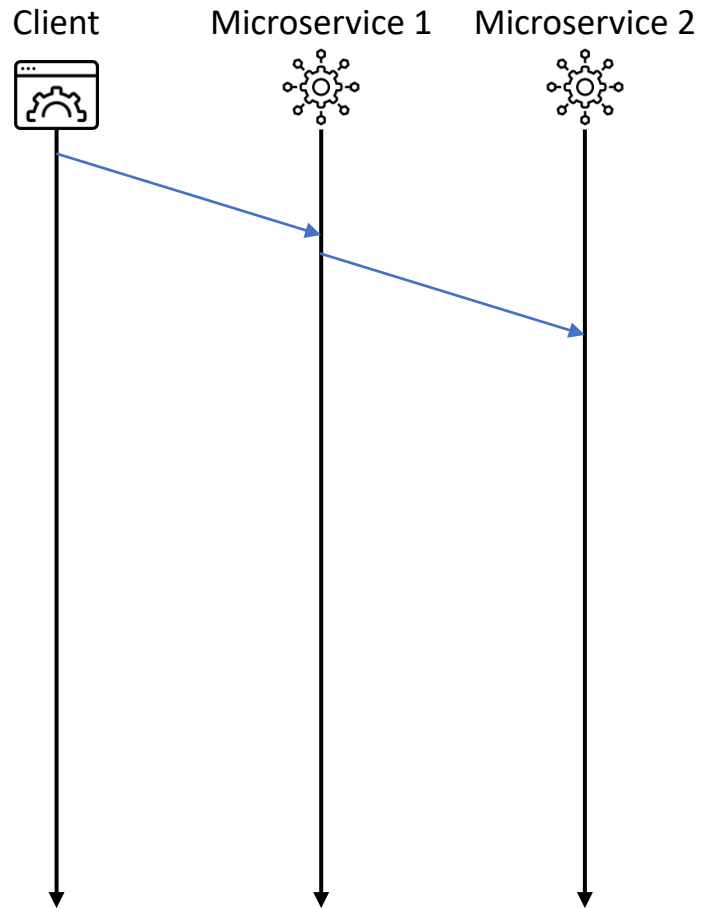
# Synchronous vs asynchronous protocols

## Synchronous



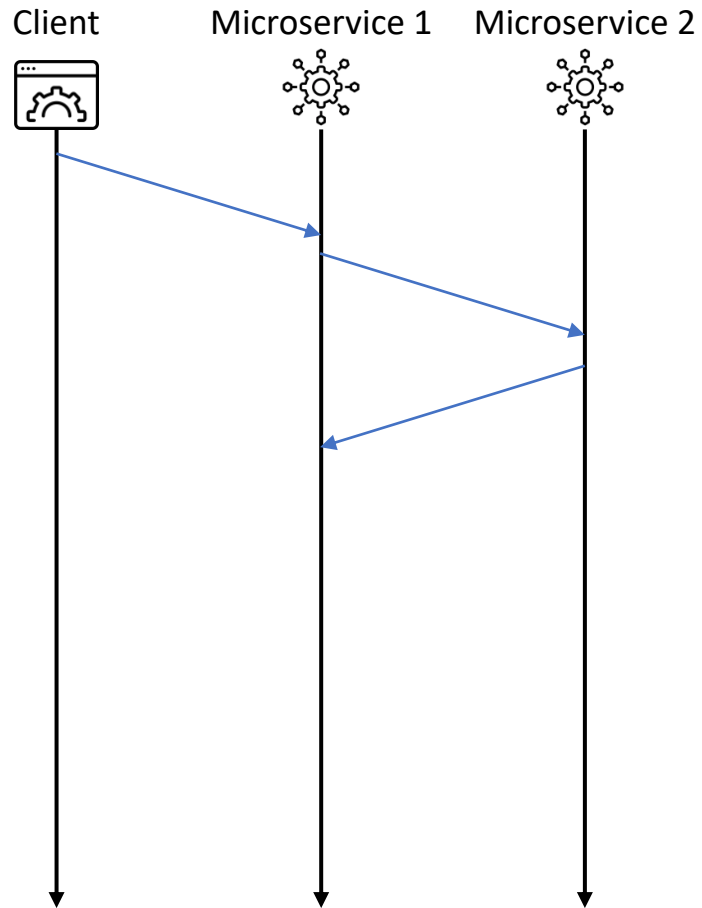
# Synchronous vs asynchronous protocols

## Synchronous



# Synchronous vs asynchronous protocols

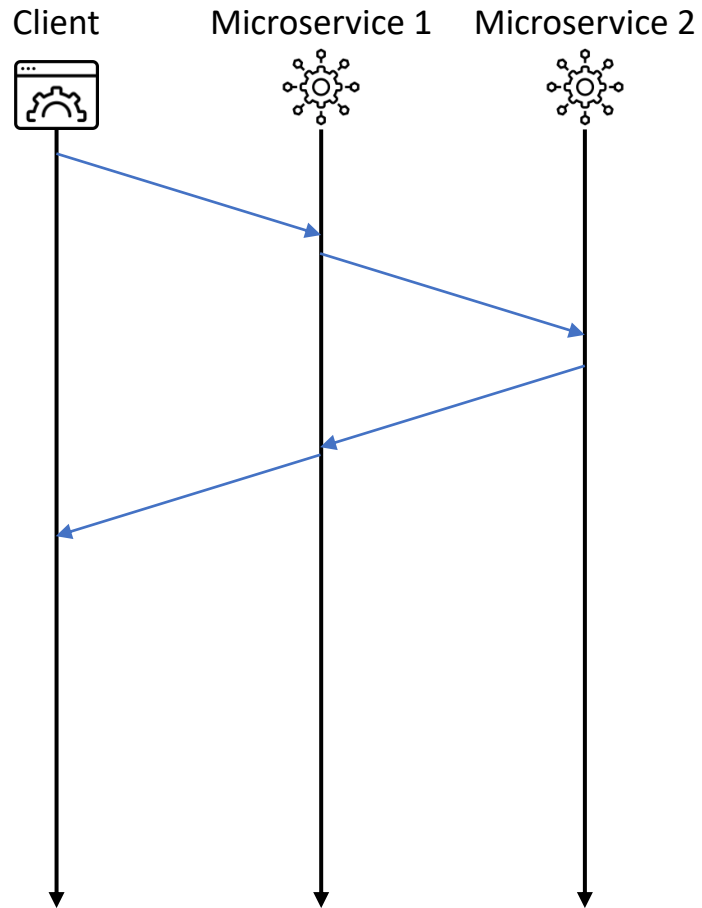
## Synchronous





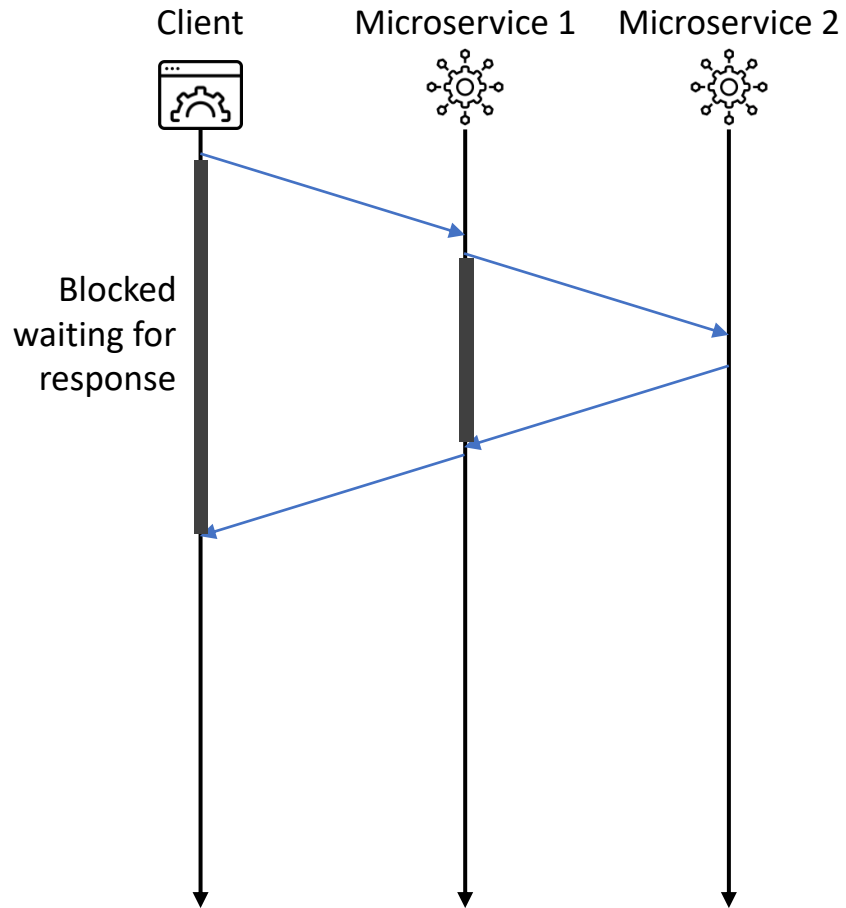
# Synchronous vs asynchronous protocols

## Synchronous



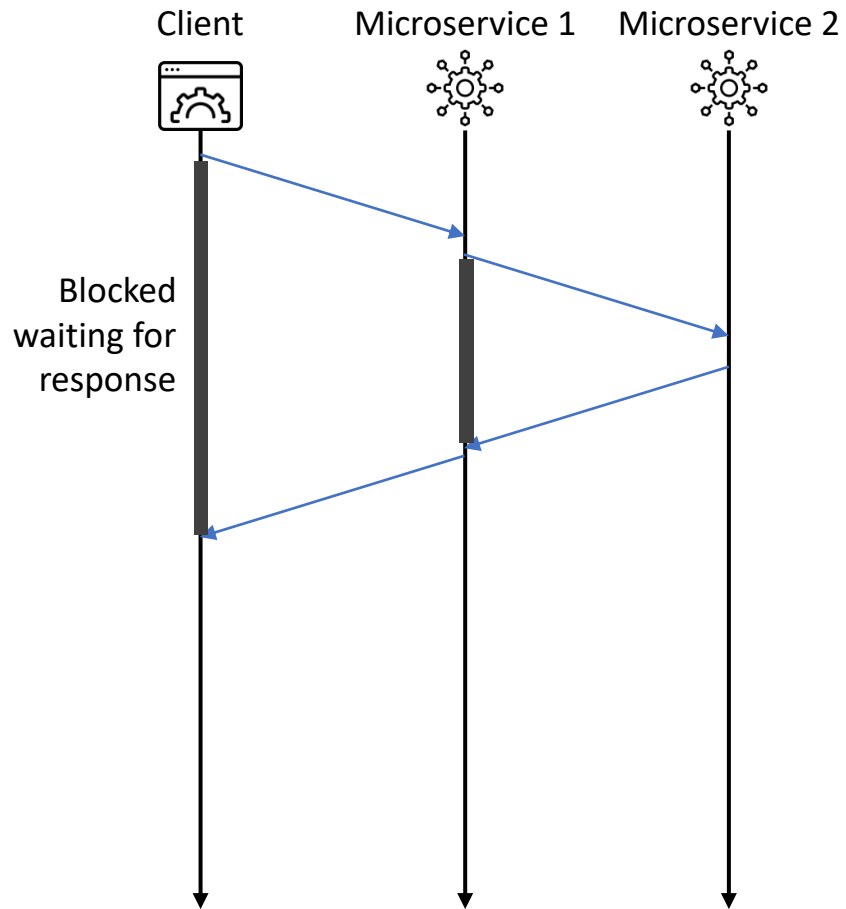
# Synchronous vs asynchronous protocols

## Synchronous

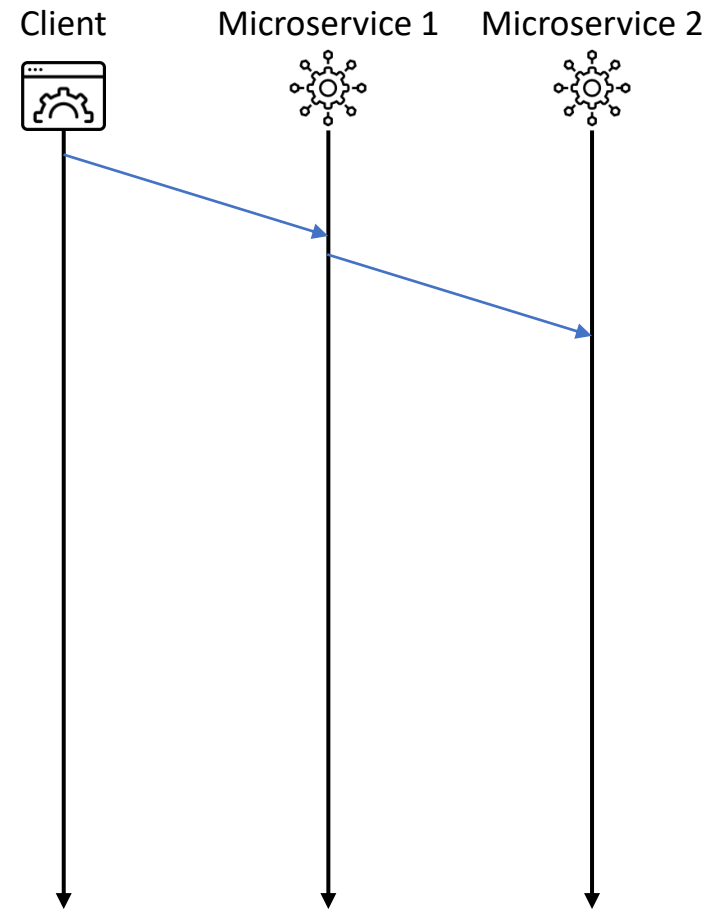


# Synchronous vs asynchronous protocols

## Synchronous

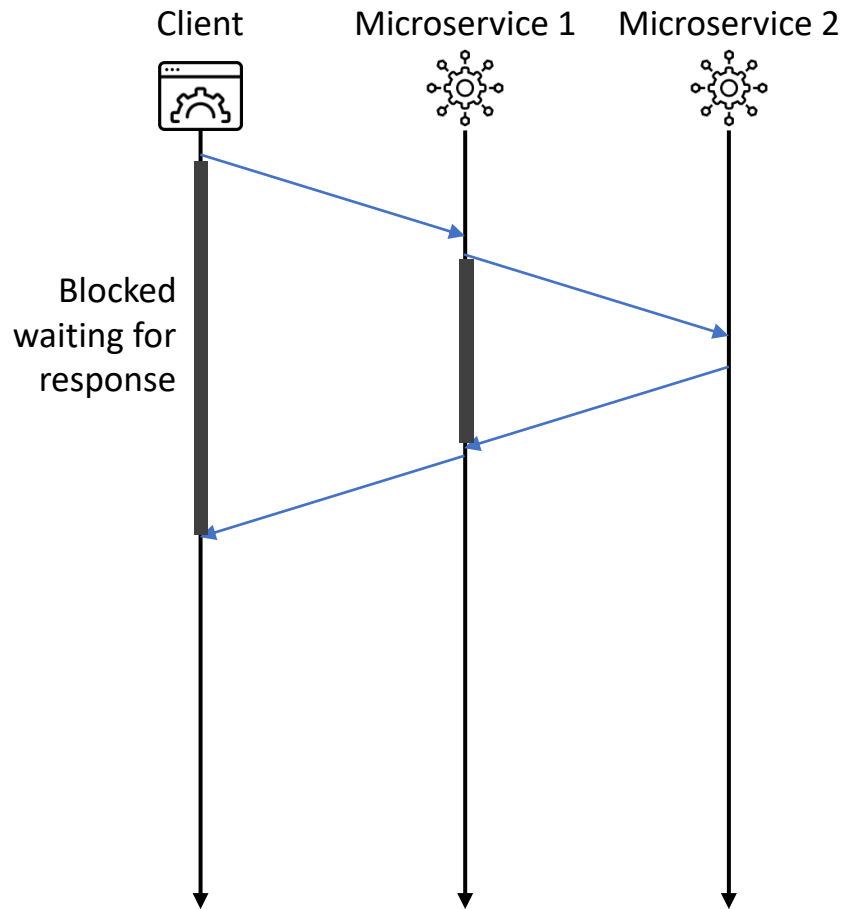


## Asynchronous

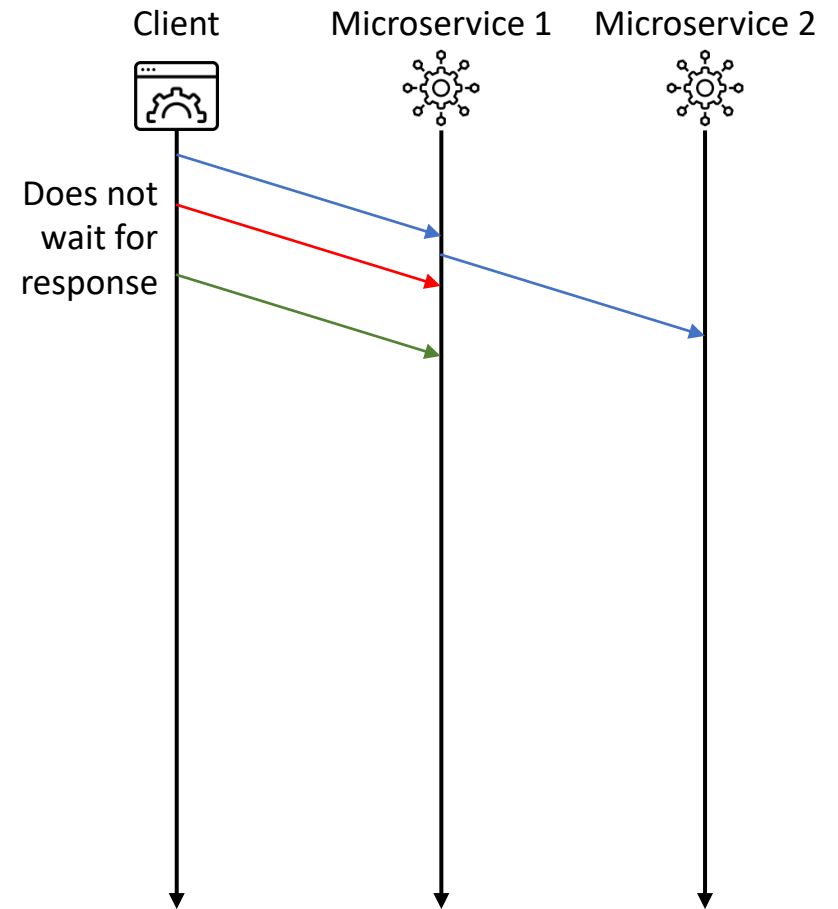


# Synchronous vs asynchronous protocols

## Synchronous

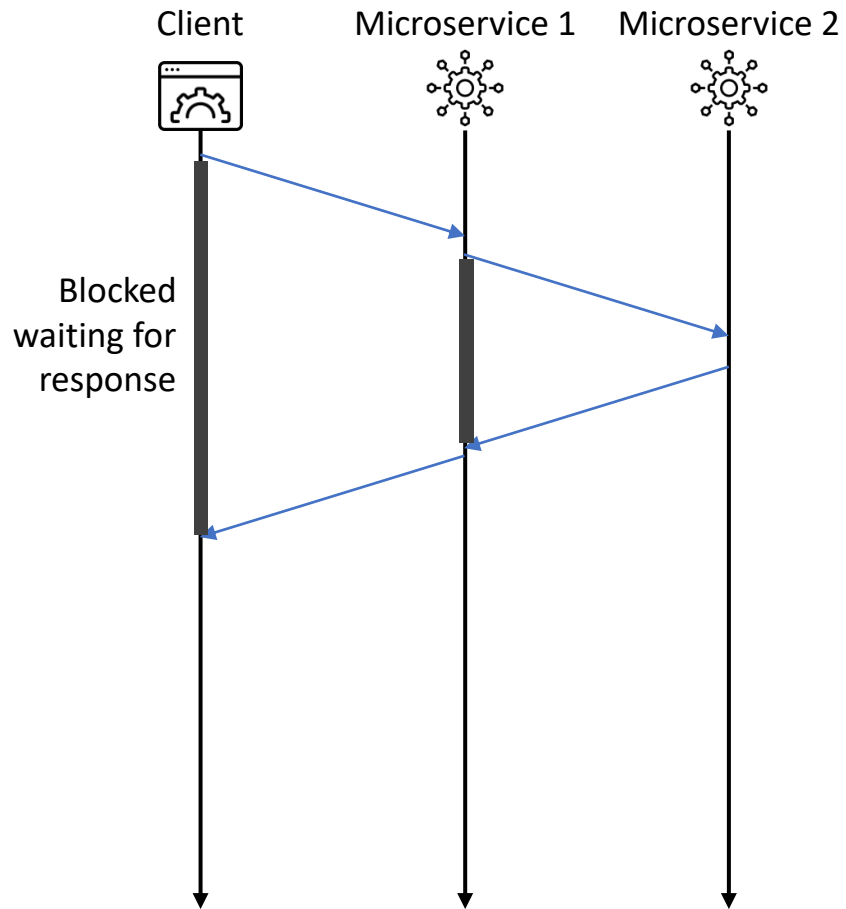


## Asynchronous

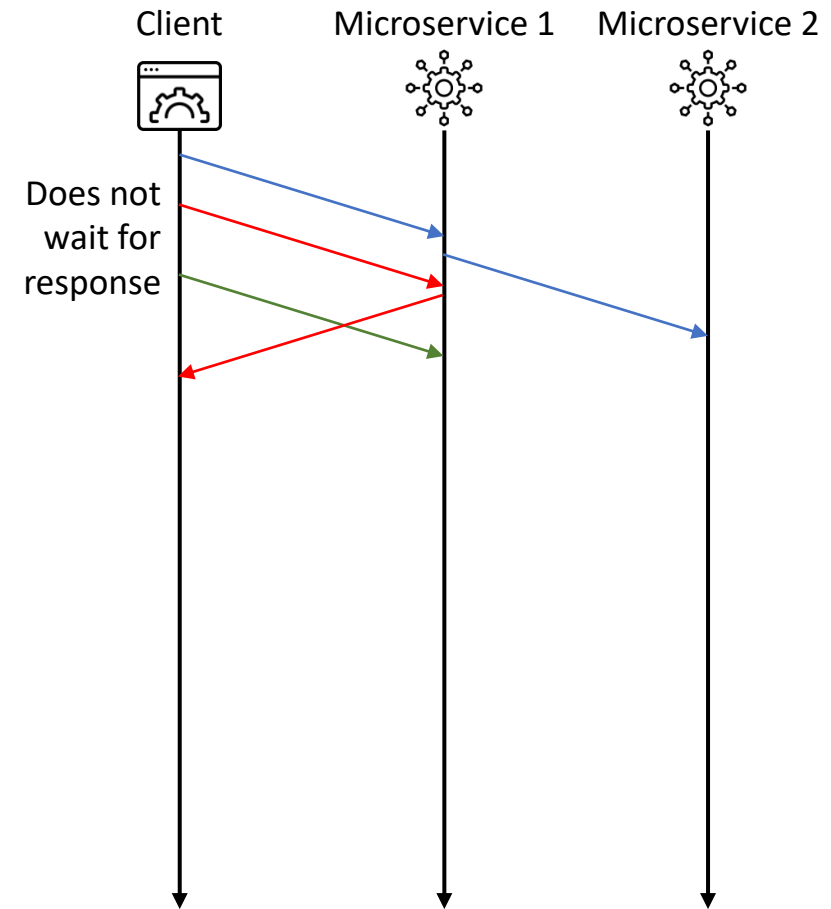


# Synchronous vs asynchronous protocols

## Synchronous

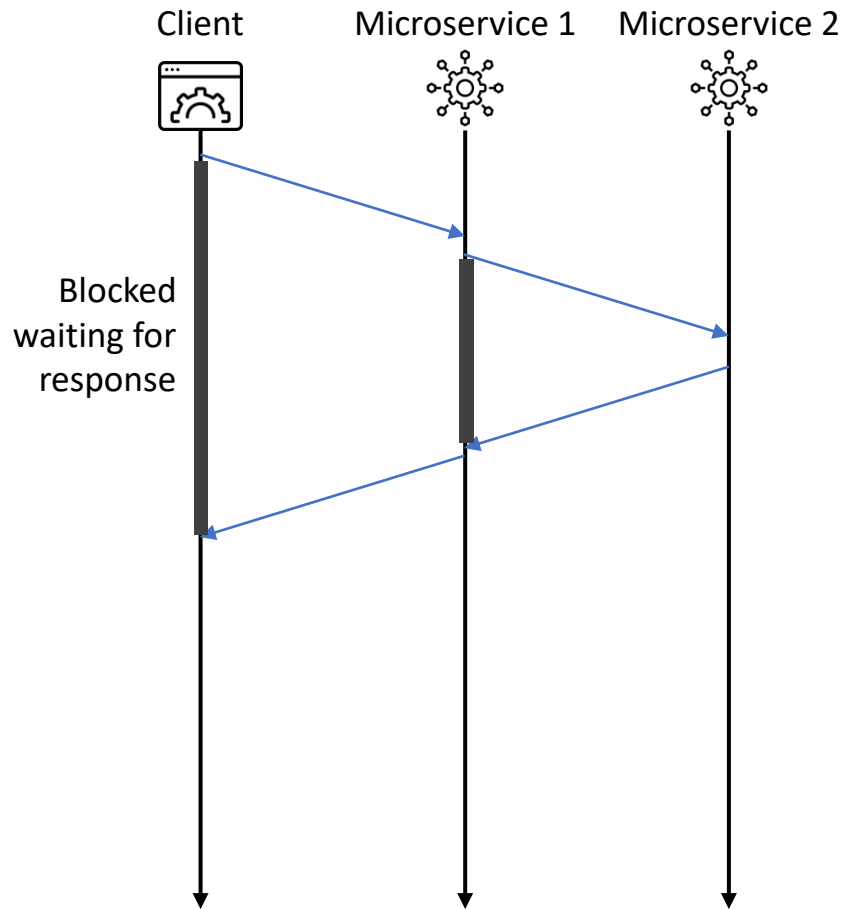


## Asynchronous

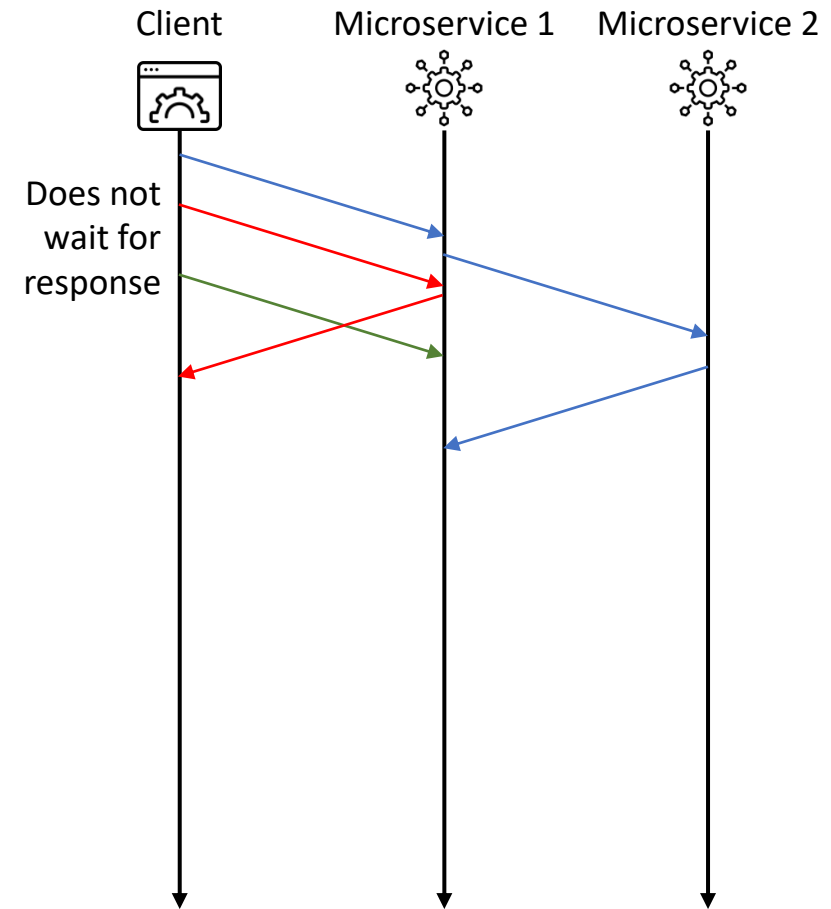


# Synchronous vs asynchronous protocols

## Synchronous

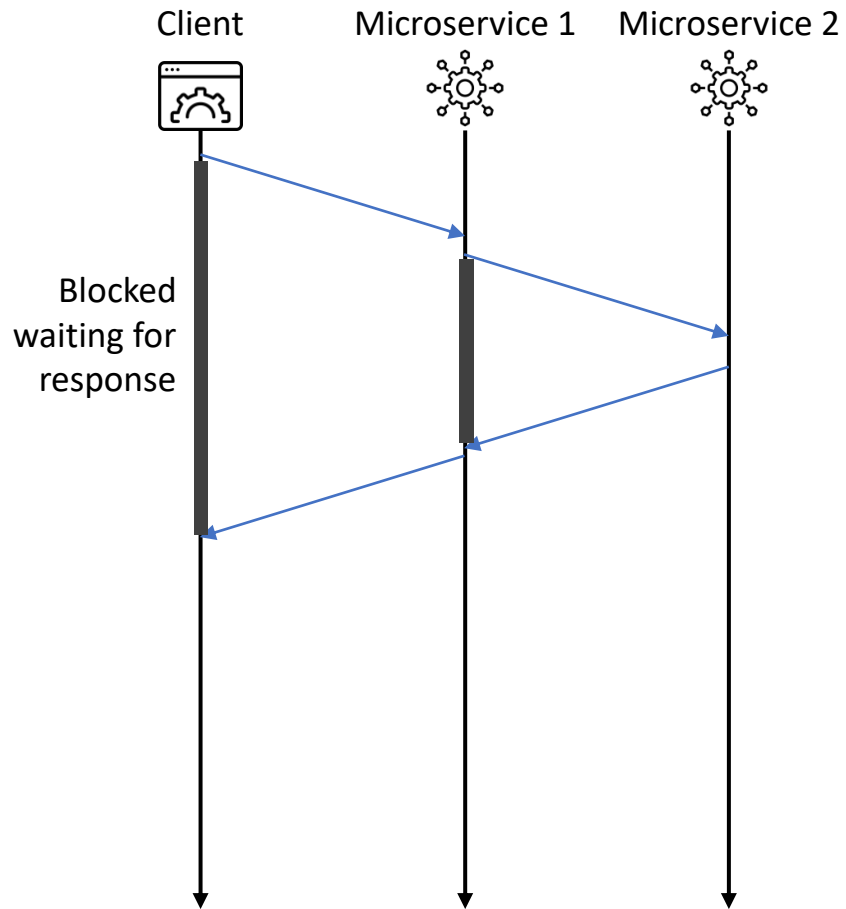


## Asynchronous

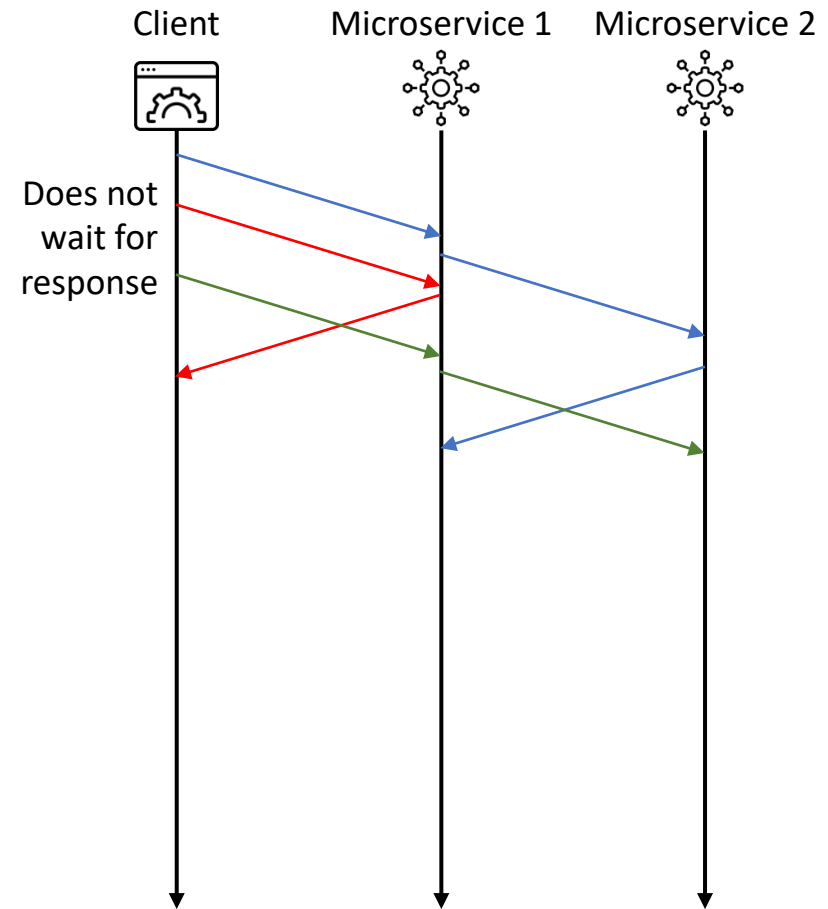


# Synchronous vs asynchronous protocols

## Synchronous

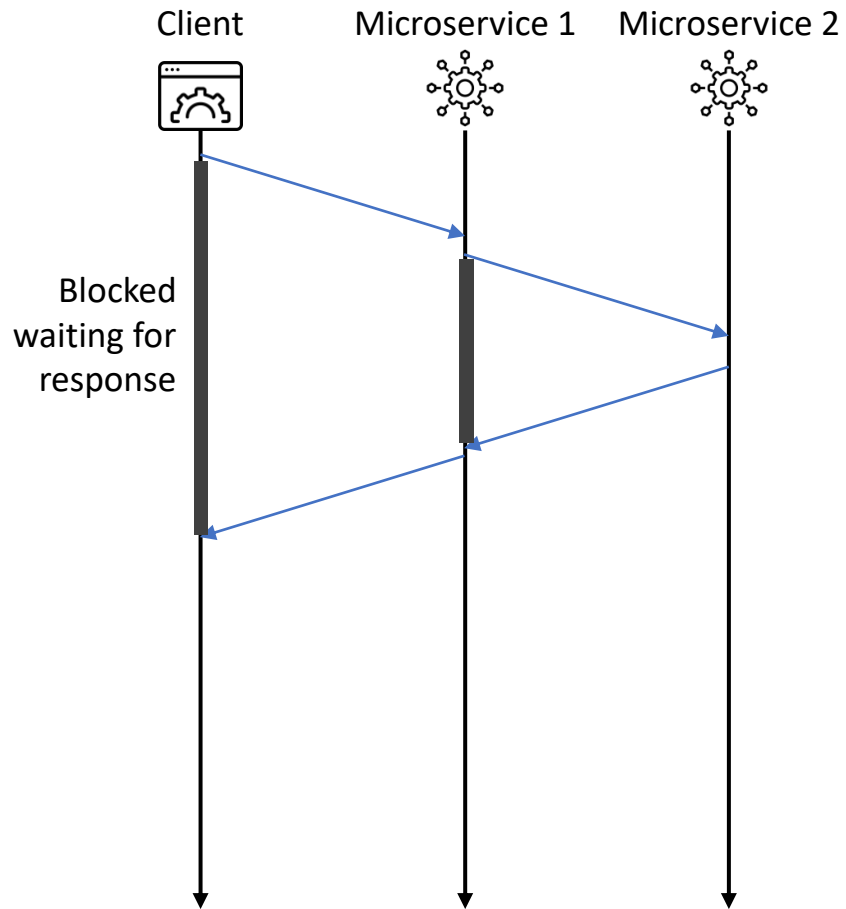


## Asynchronous

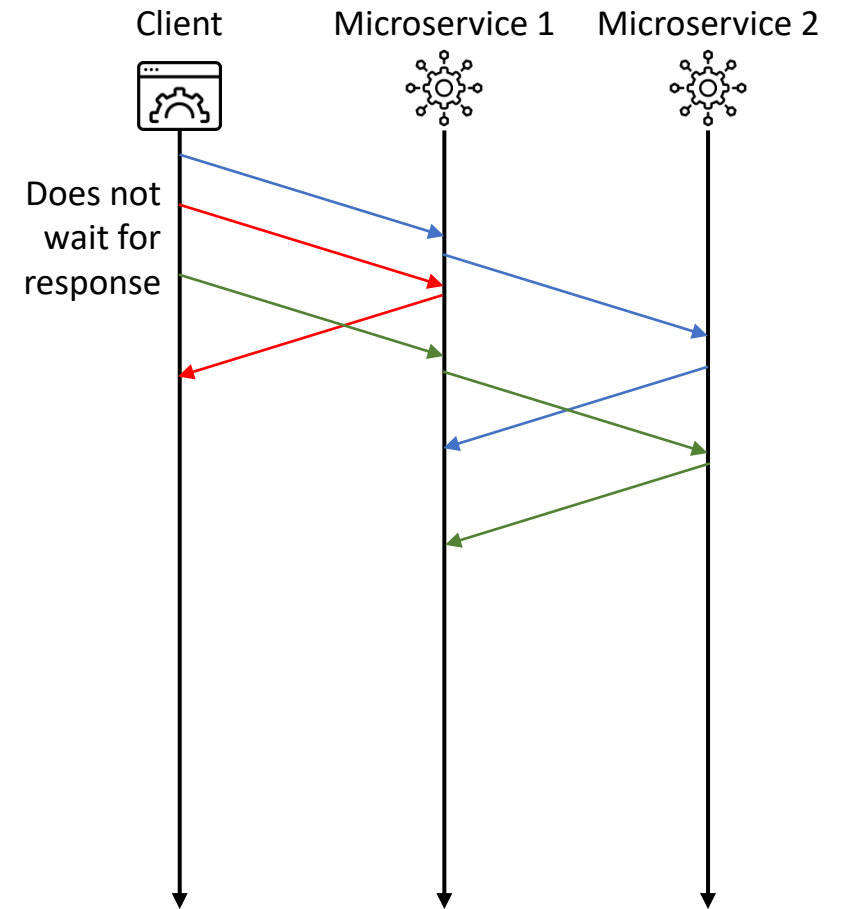


# Synchronous vs asynchronous protocols

## Synchronous



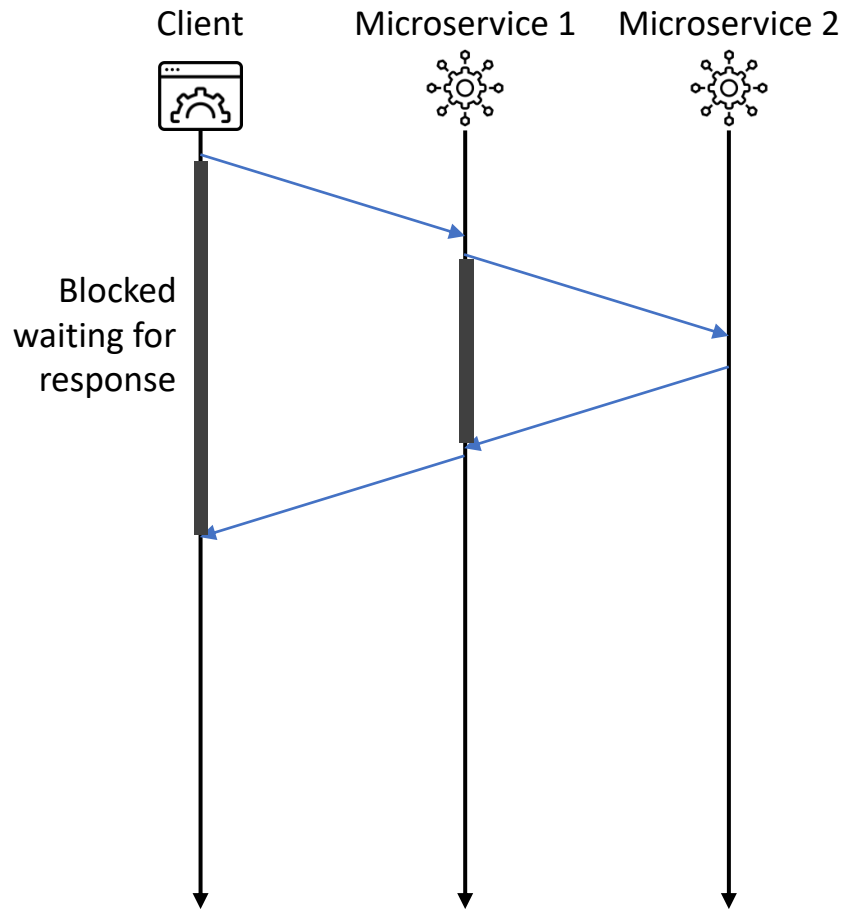
## Asynchronous



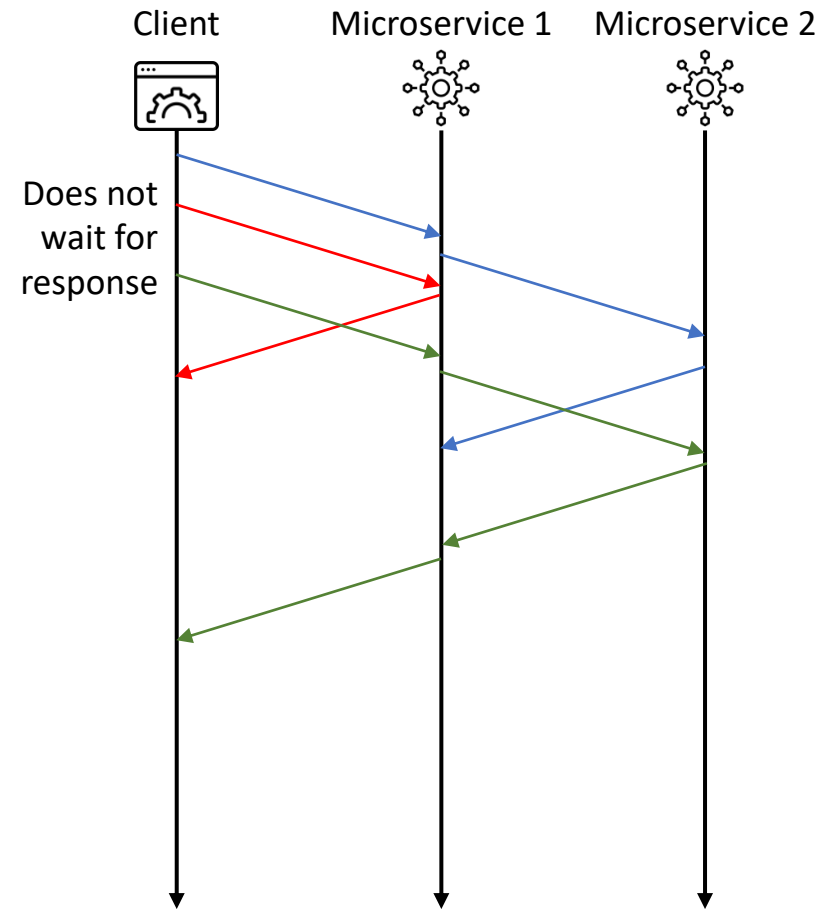


# Synchronous vs asynchronous protocols

## Synchronous

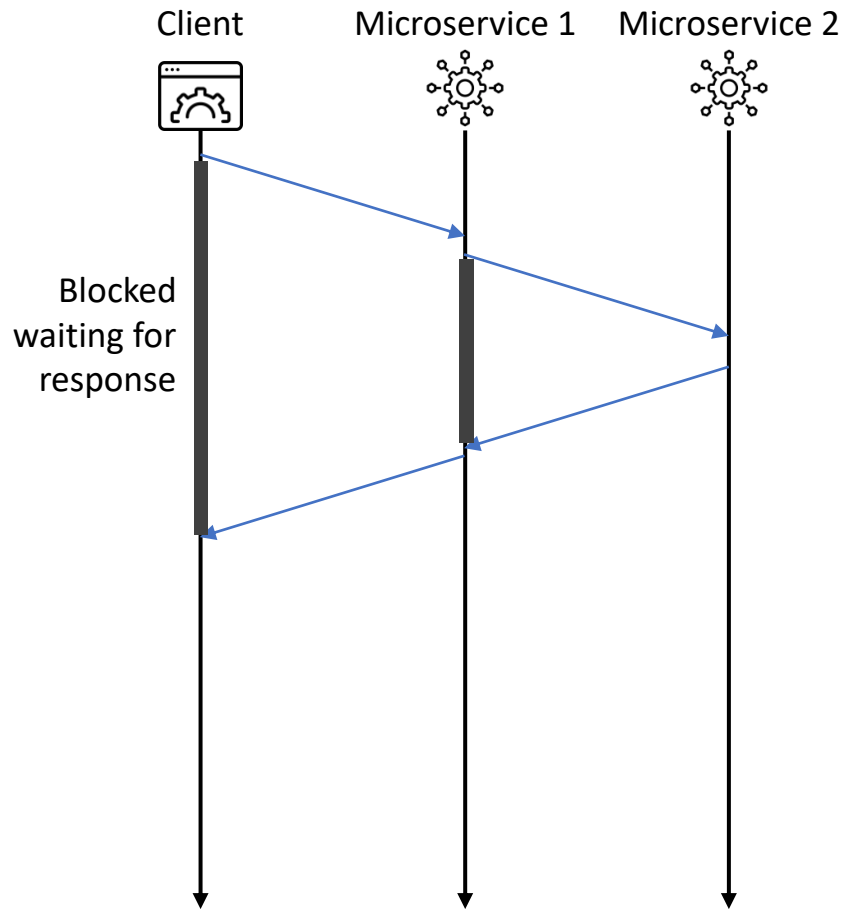


## Asynchronous

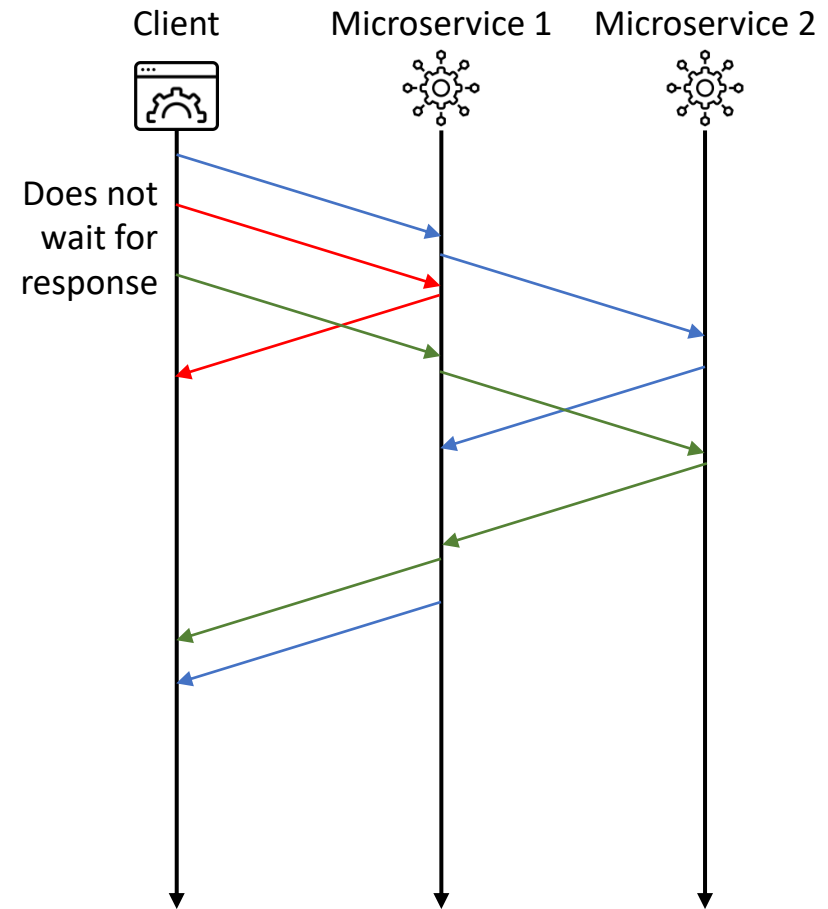


# Synchronous vs asynchronous protocols

## Synchronous

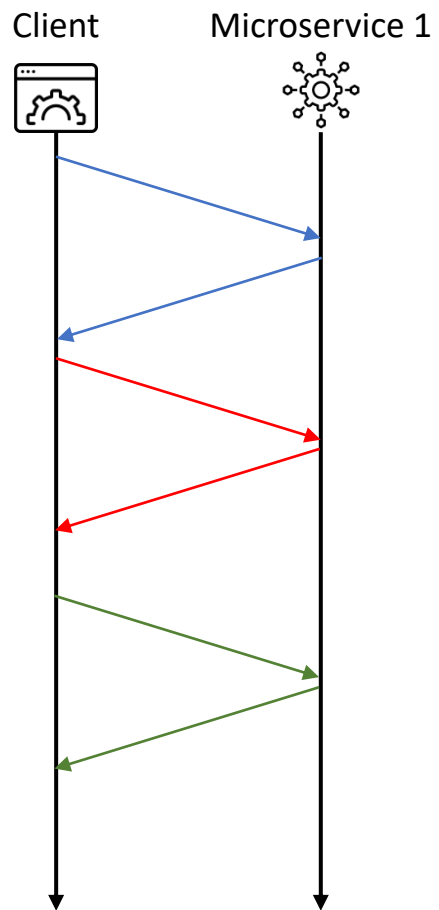


## Asynchronous



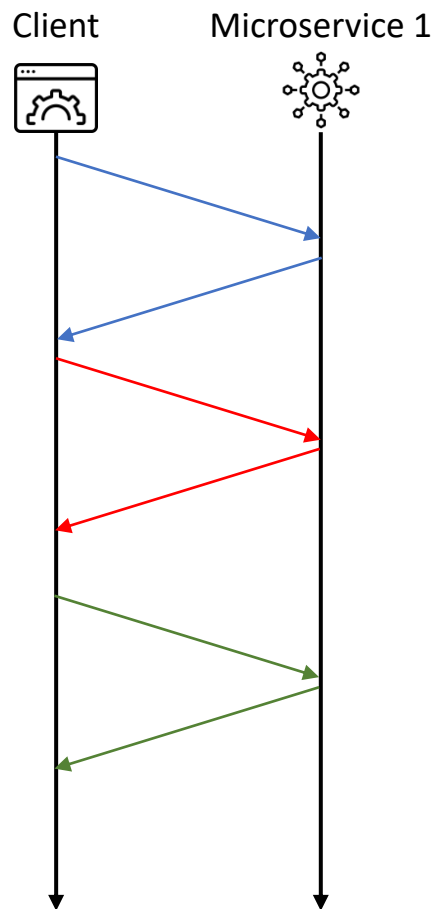
# Communication patterns

## Request/Response

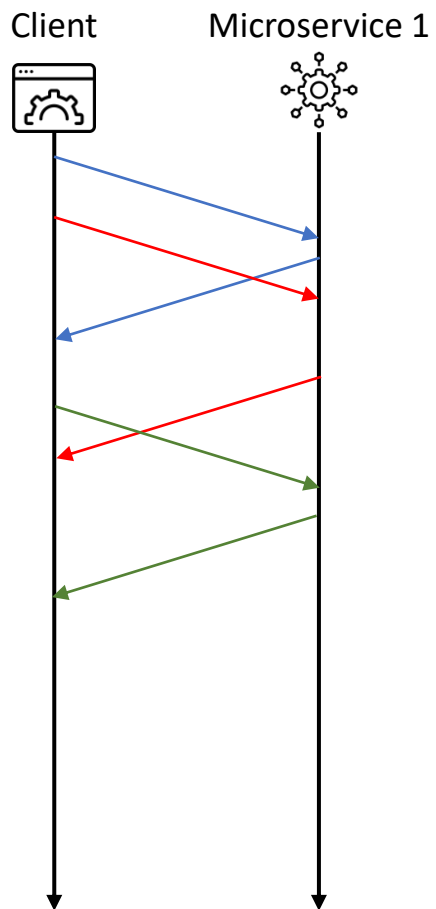


# Communication patterns

## Request/Response

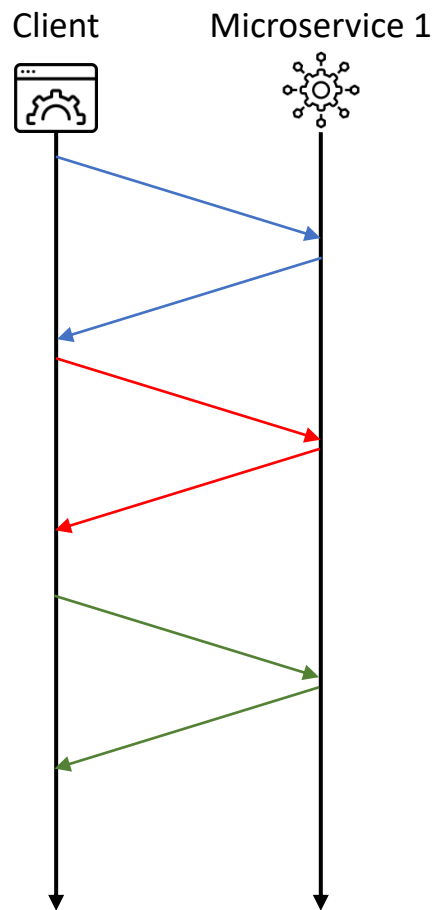


## Request/Response (async)

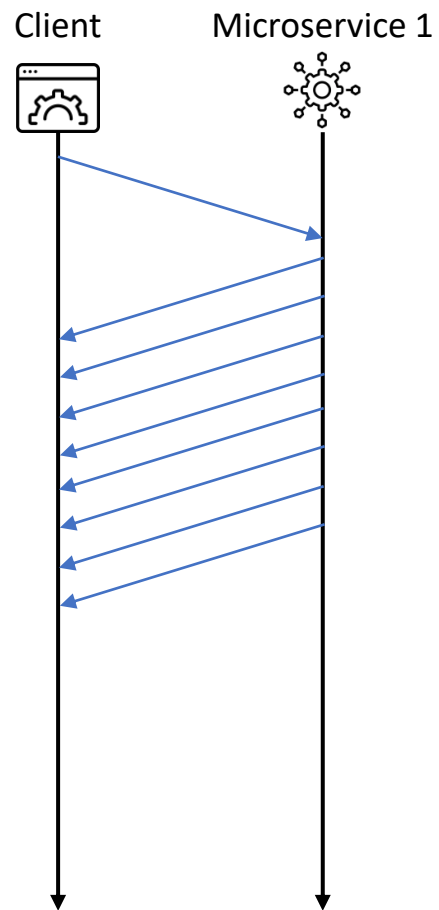


# Communication patterns

## Request/Response

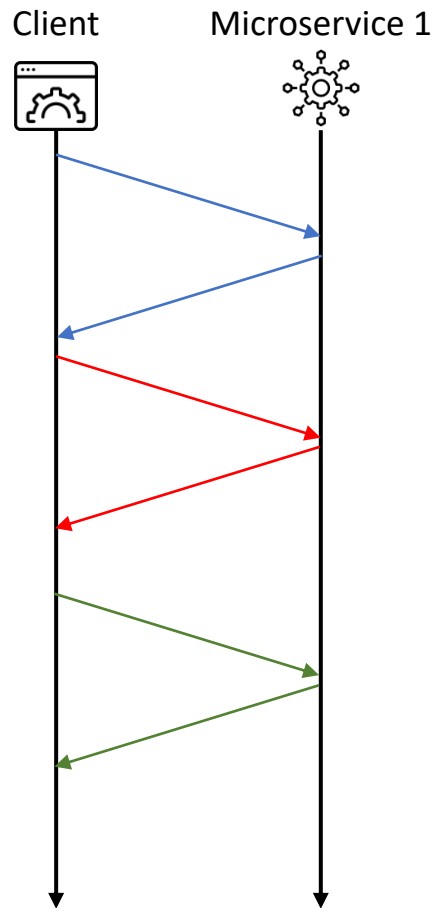


## Streaming

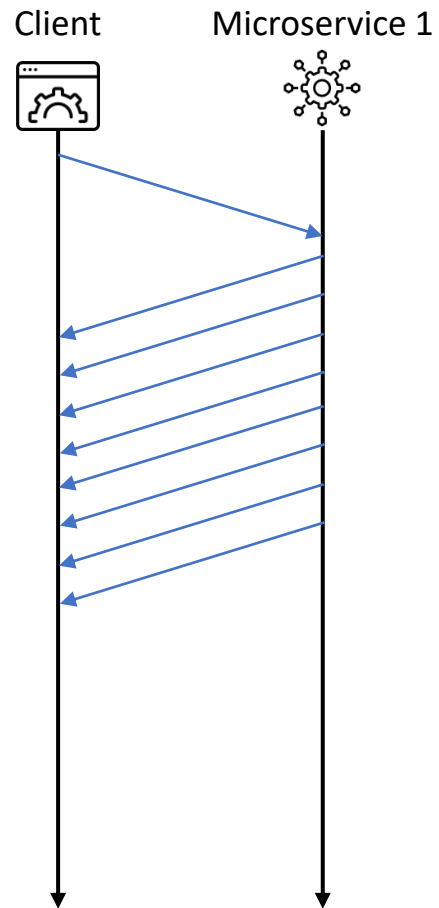


# Communication patterns

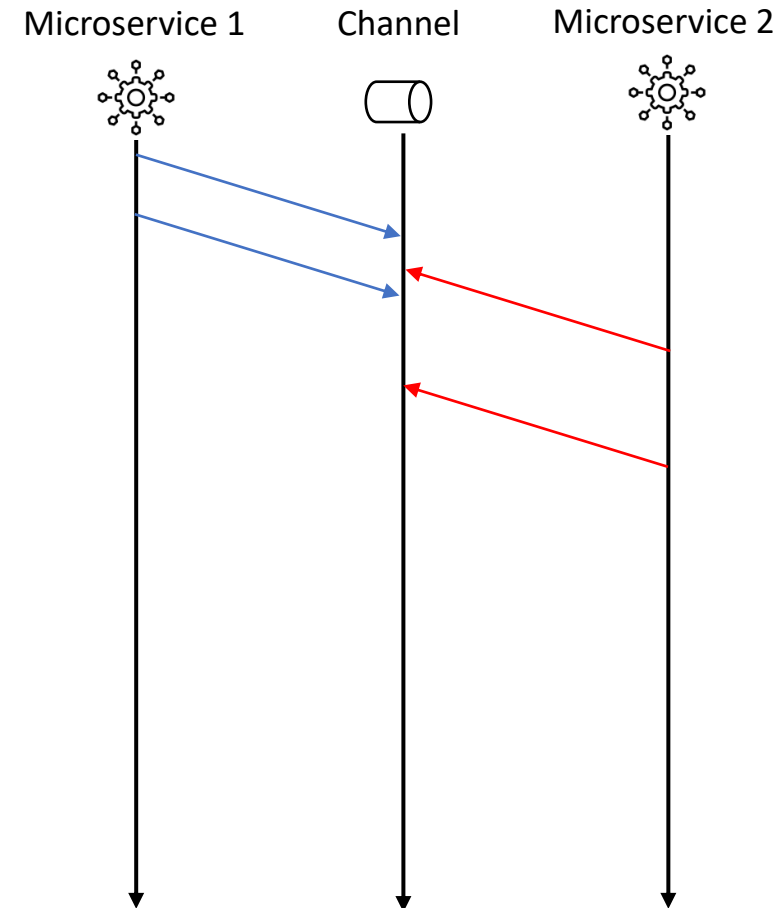
## Request/Response



## Streaming

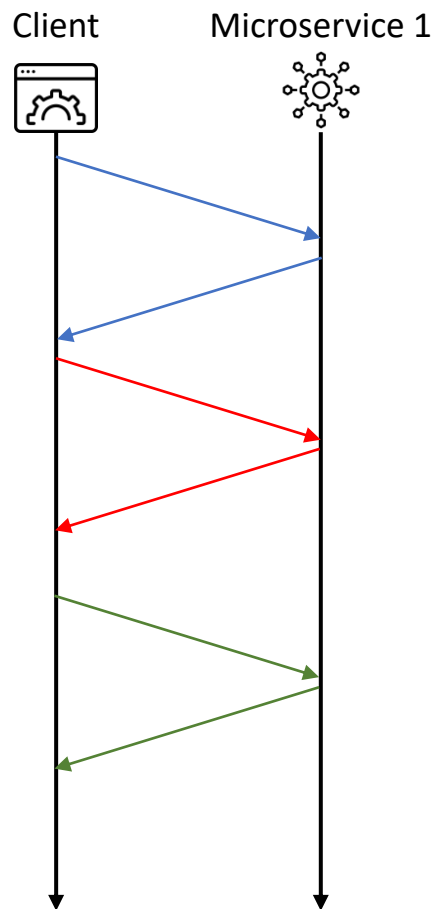


## Message Passing

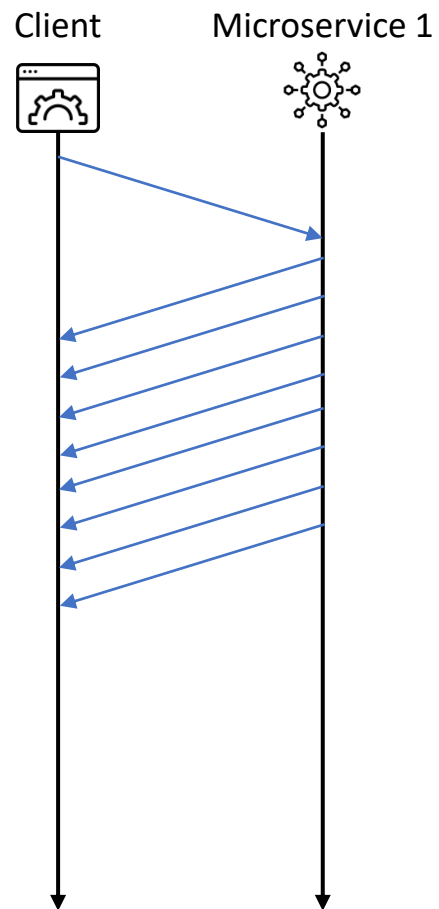


# Communication patterns

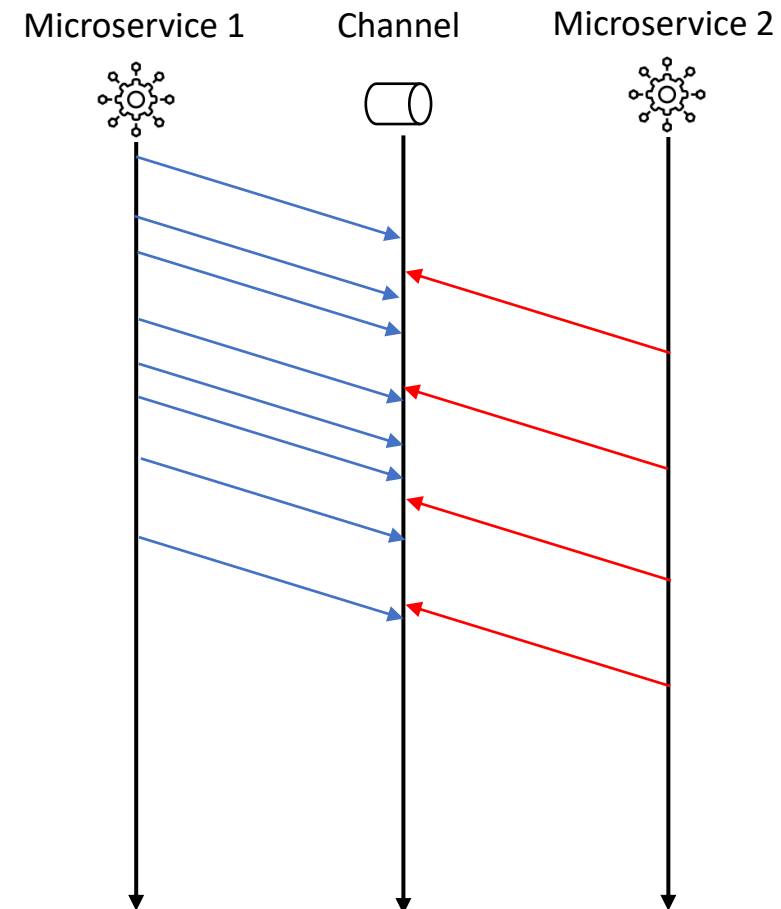
## Request/Response



## Streaming

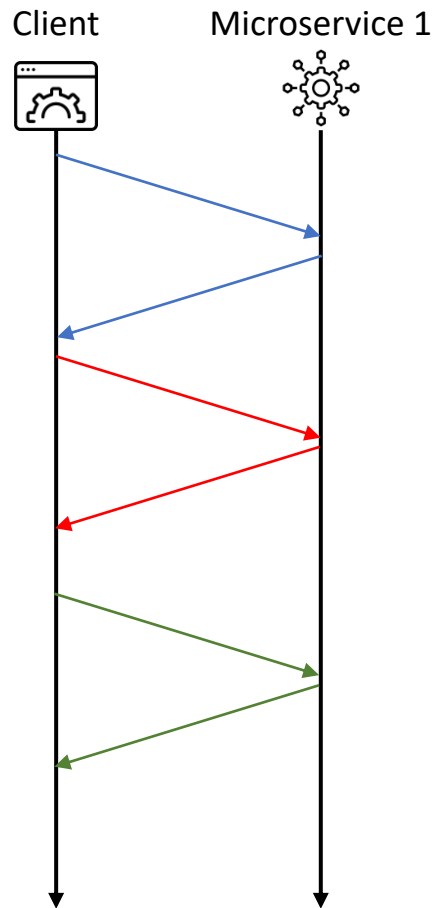


## Message Passing

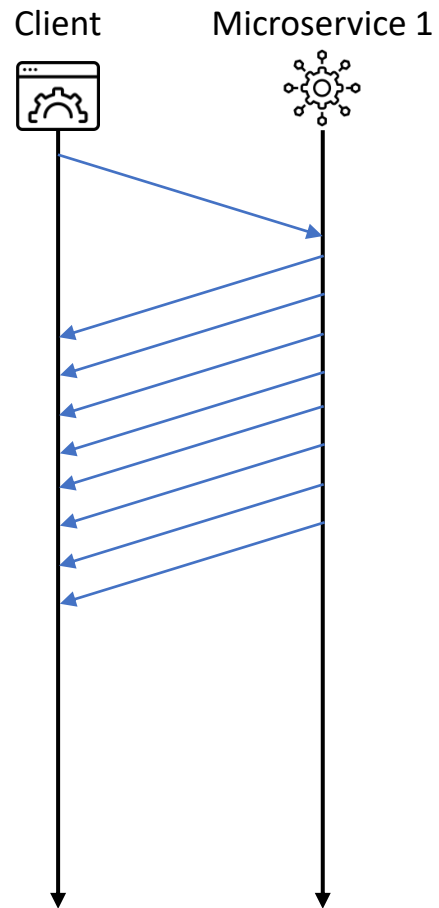


# Communication patterns

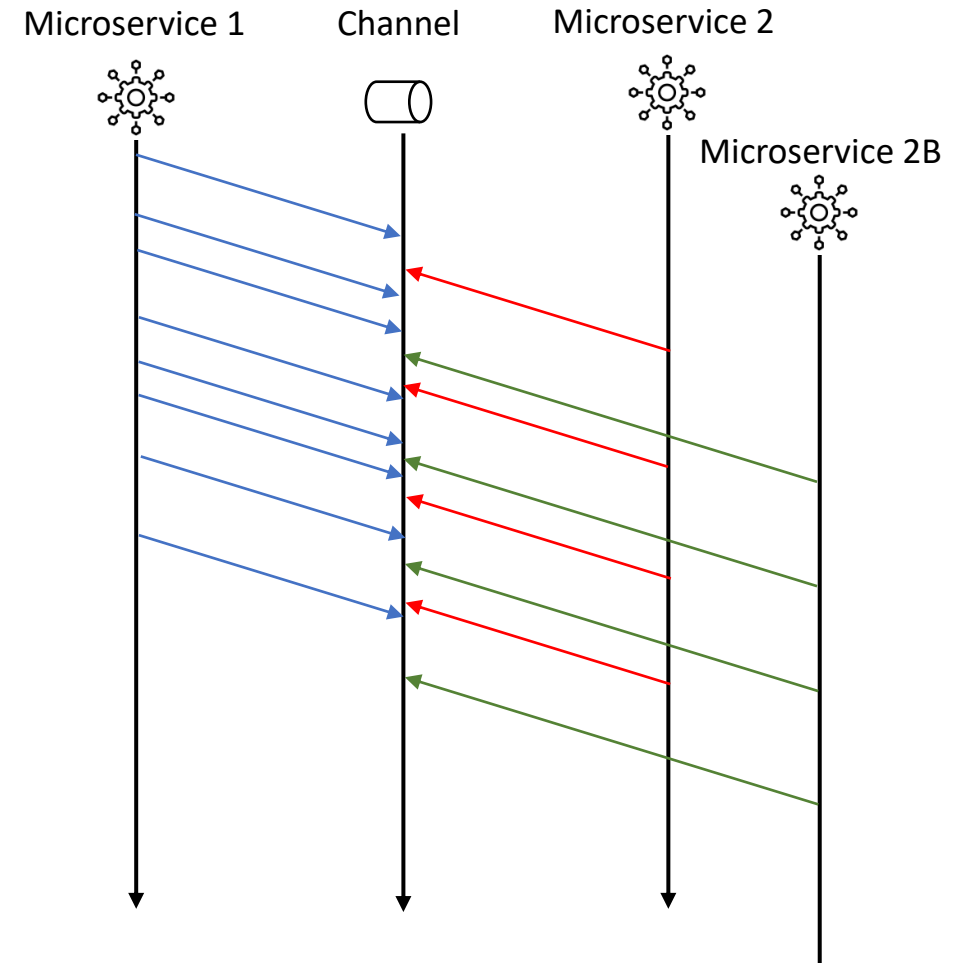
## Request/Response



## Streaming





## Message Passing (one-to-many)





# Communication technologies

	Request/Response	Streaming	Message Passing
Synchronous	<p>{REST}</p> <p>gRPC</p>		
Asynchronous	<p>gRPC</p>	<p>RabbitMQ</p> <p>gRPC</p> <p>APACHE kafka</p>	<p>RabbitMQ</p> <p>APACHE kafka</p>

# Message and data encoding

- Text
  - JSON
  - XML
  - YAML
  - CSV
  - ...
- Binary
  - Pickle
  - ProtoBuf
  - Cap'n Proto
  - ...

# Message and data encoding

- Text

- JSON
- XML
- YAML
- CSV
- ...

Editable, easily viewable

- Binary

- Pickle
- ProtoBuf
- Cap'n Proto
- ...

Smaller size, faster encoding and decoding

# Message and data encoding

- Text

- JSON
- XML
- YAML
- CSV
- ...

- Binary

- Pickle → Language specific (Python)
- ProtoBuf → Language agnostic
- Cap'n Proto
- ...

```
@0xdbb9ad1f14bf0b36;  # unique file ID
```

```
struct Person {  
    name @0 :Text;  
    birthdate @3 :Date;  
  
    email @1 :Text;  
    phones @2 :List(PhoneNumber);
```

```
struct PhoneNumber {  
    number @0 :Text;  
    type @1 :Type;
```

```
enum Type {  
    mobile @0;  
    home @1;  
    work @2;
```

```
    }  
}  
}
```

```
struct Date {  
    year @0 :Int16;  
    month @1 :UInt8;  
    day @2 :UInt8;  
}
```



# CompSci 401: Cloud Computing

# Service Meshes

Prof. Ítalo Cunha



# Service mesh

- Orchestration for microservice communications
  - Load balancing
  - Discovery (compatible with autoscaling)
  - Security
  - Monitoring
- Service-to-service (internal communication)
- Client-to-service (external communication)

# Service mesh

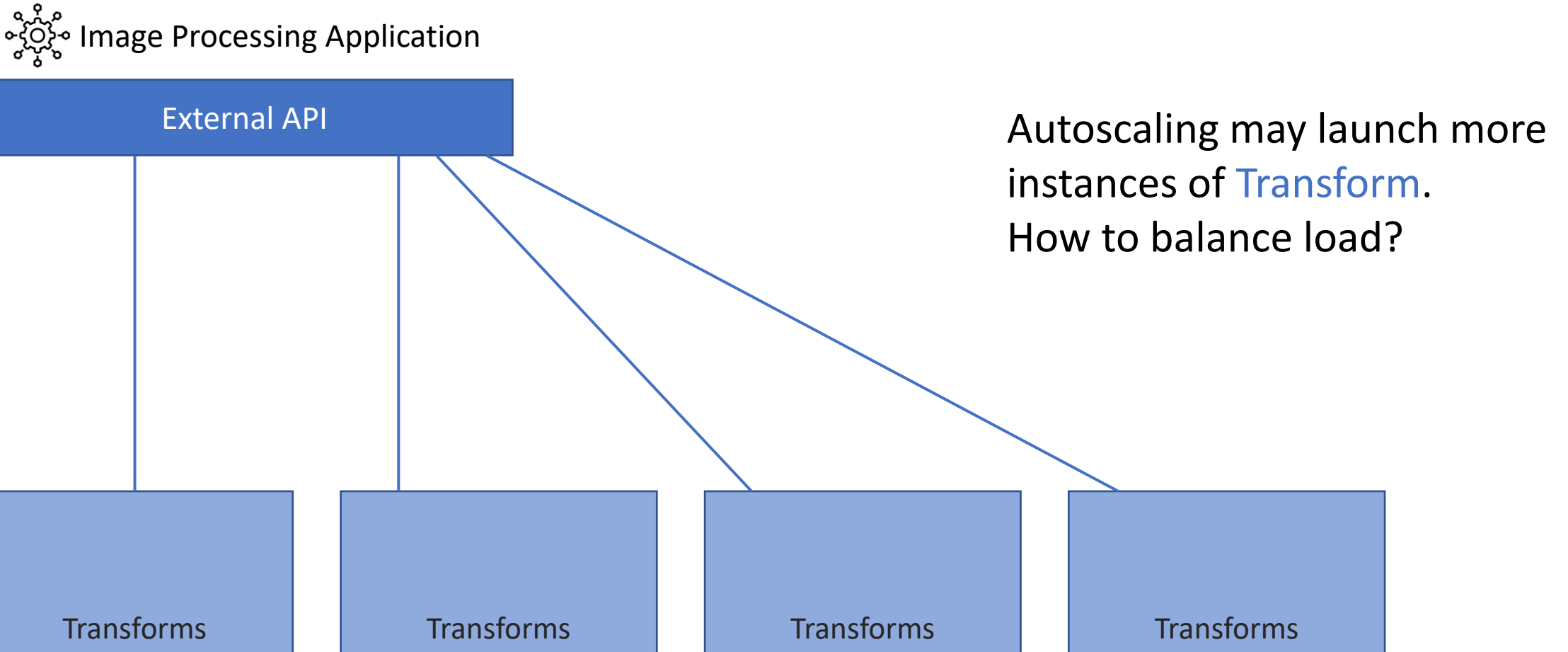


Image Processing Application

External API

Transforms

# Service mesh

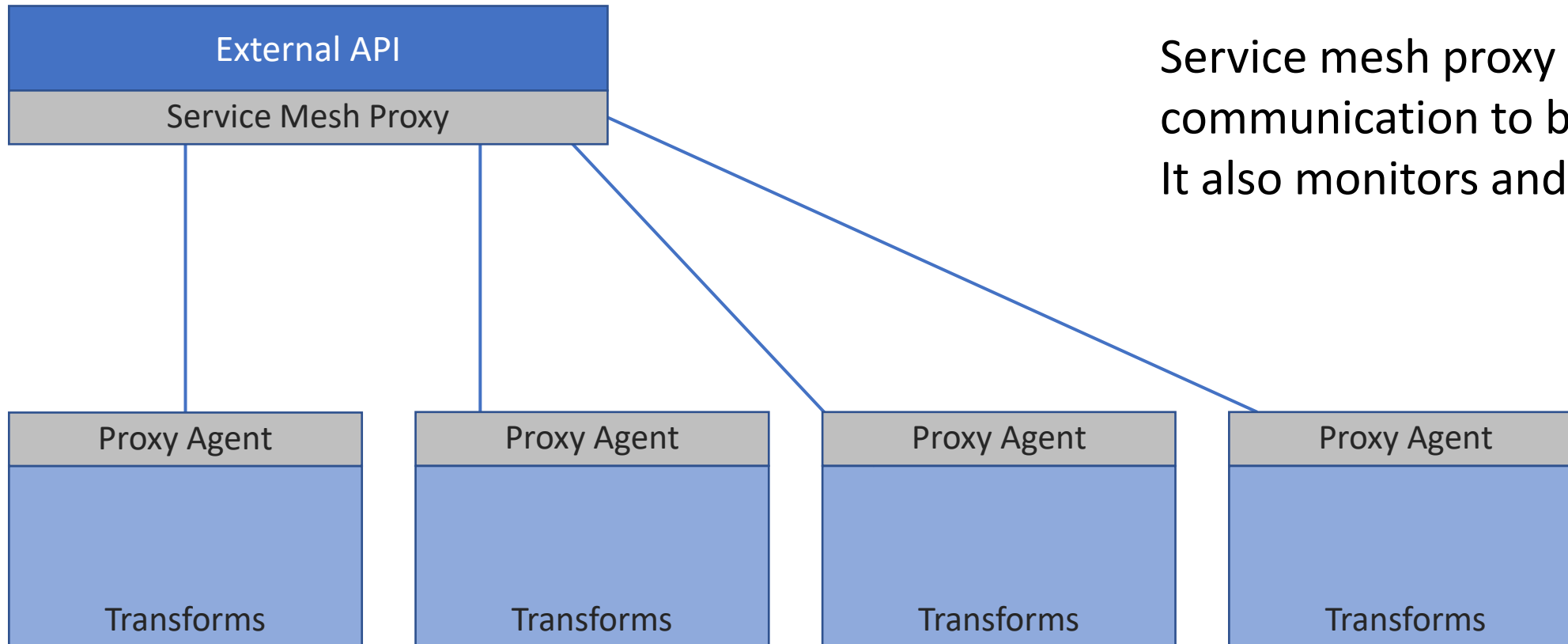




# Service mesh proxy

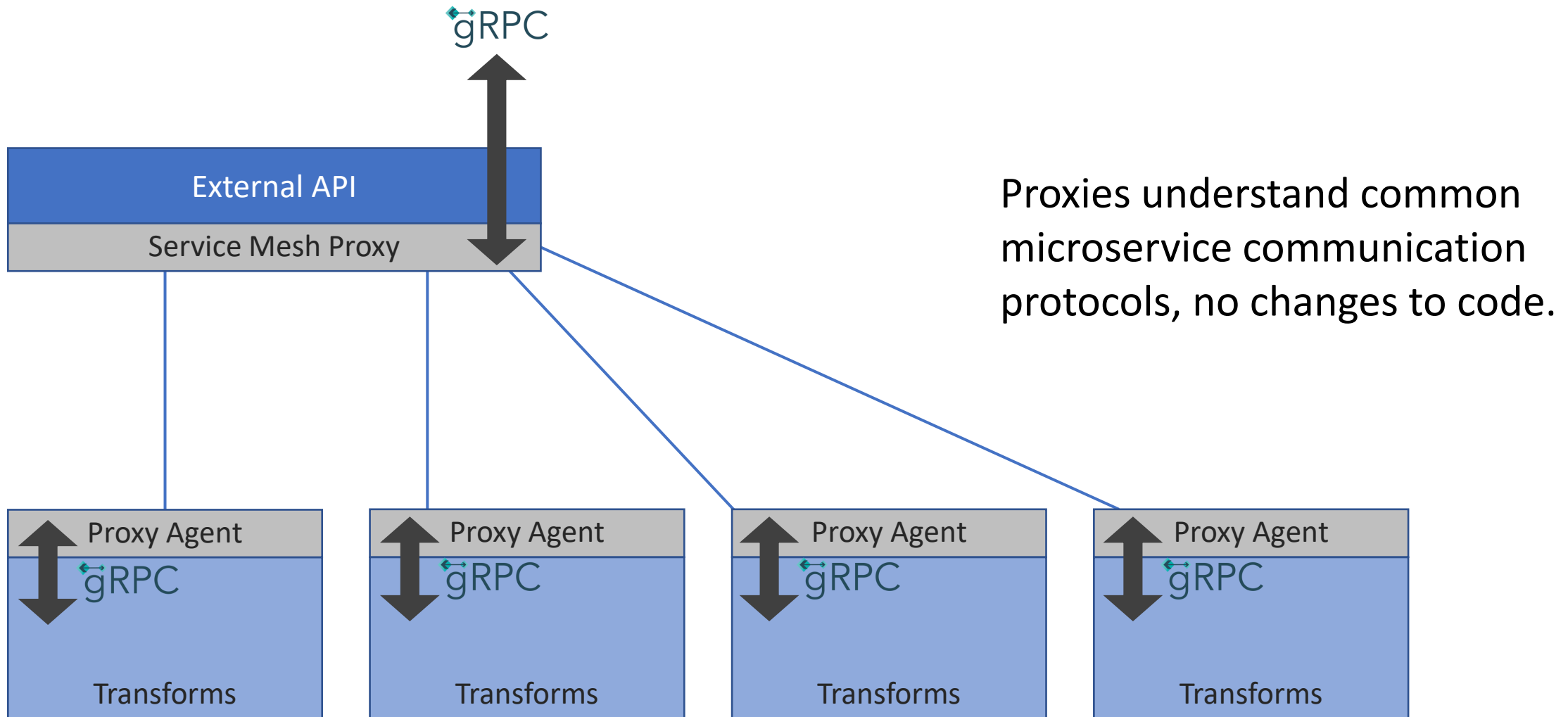


Image Processing Application



Service mesh proxy mediates all communication to balance load. It also monitors and secures traffic.

# Service mesh proxy



# Service mesh proxy

