Cyber Security Summer School 2019

CENTER FOR CYBERSIKKERHED

DTU

kea KØBENHAVNS ERHVERVSAKADEMI

CBS

AALBORG UNIVERSITET

SDU

SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Microservice Security Concepts

Presentation: Me

Name: **Saverio**

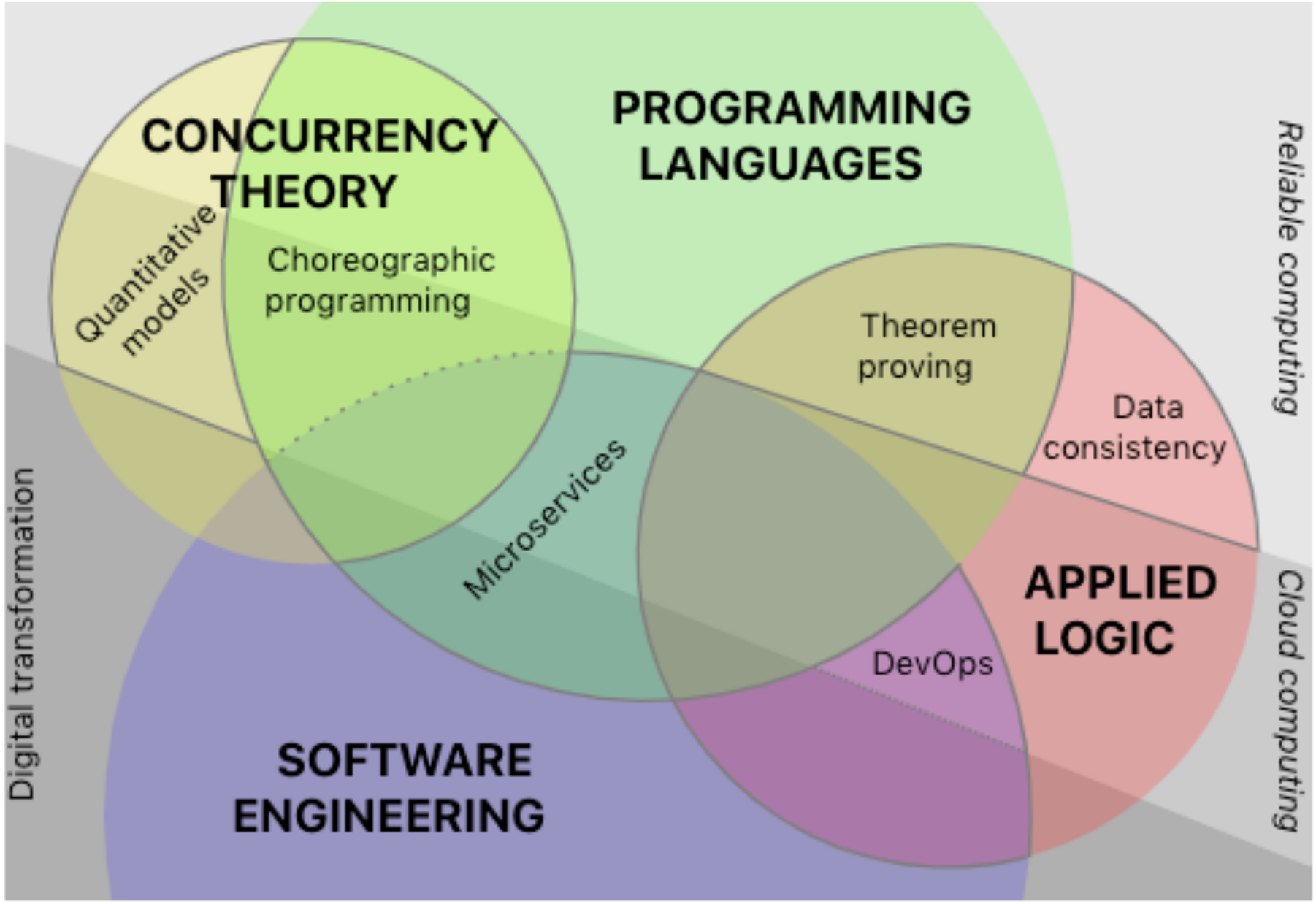
From: **University of Southern Denmark (Odense),**
Department of Mathematics and Computer Science

Research group: **Concurrency and Logic**

Expertise: **Programming Languages, Microservices, Internet of Things, Security**

SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Presentation: the CL group



https://concurrency.sdu.dk

Microservice Security Concepts

SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

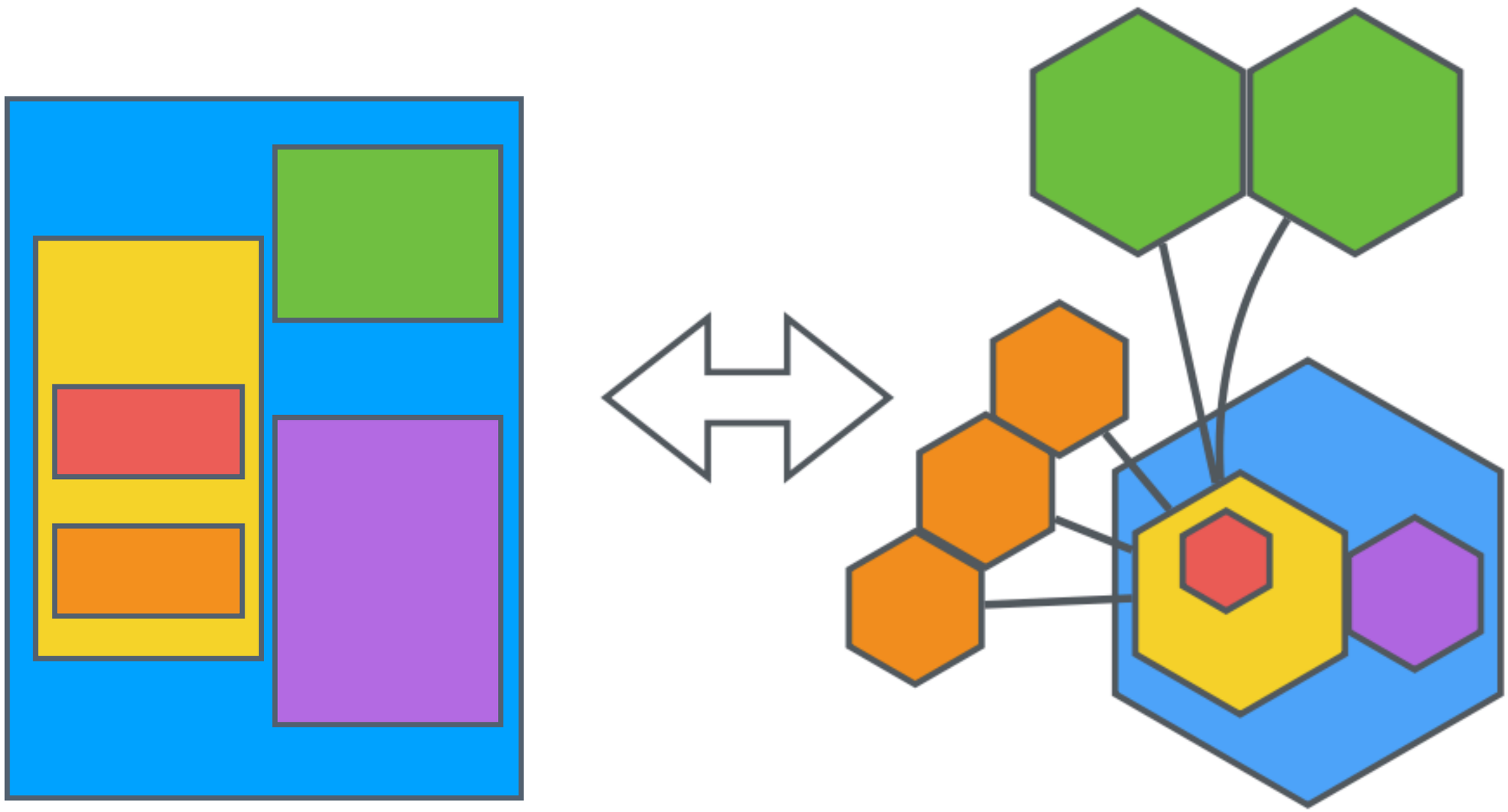
Microservice Security Concepts

Target Audience: CTOs, CSOs, Project Managers, Developers

Overview:

- Microservices, from 10 to 1 km high
- Microservices and Containers
- Microservice Security

Microservices, from 10km high

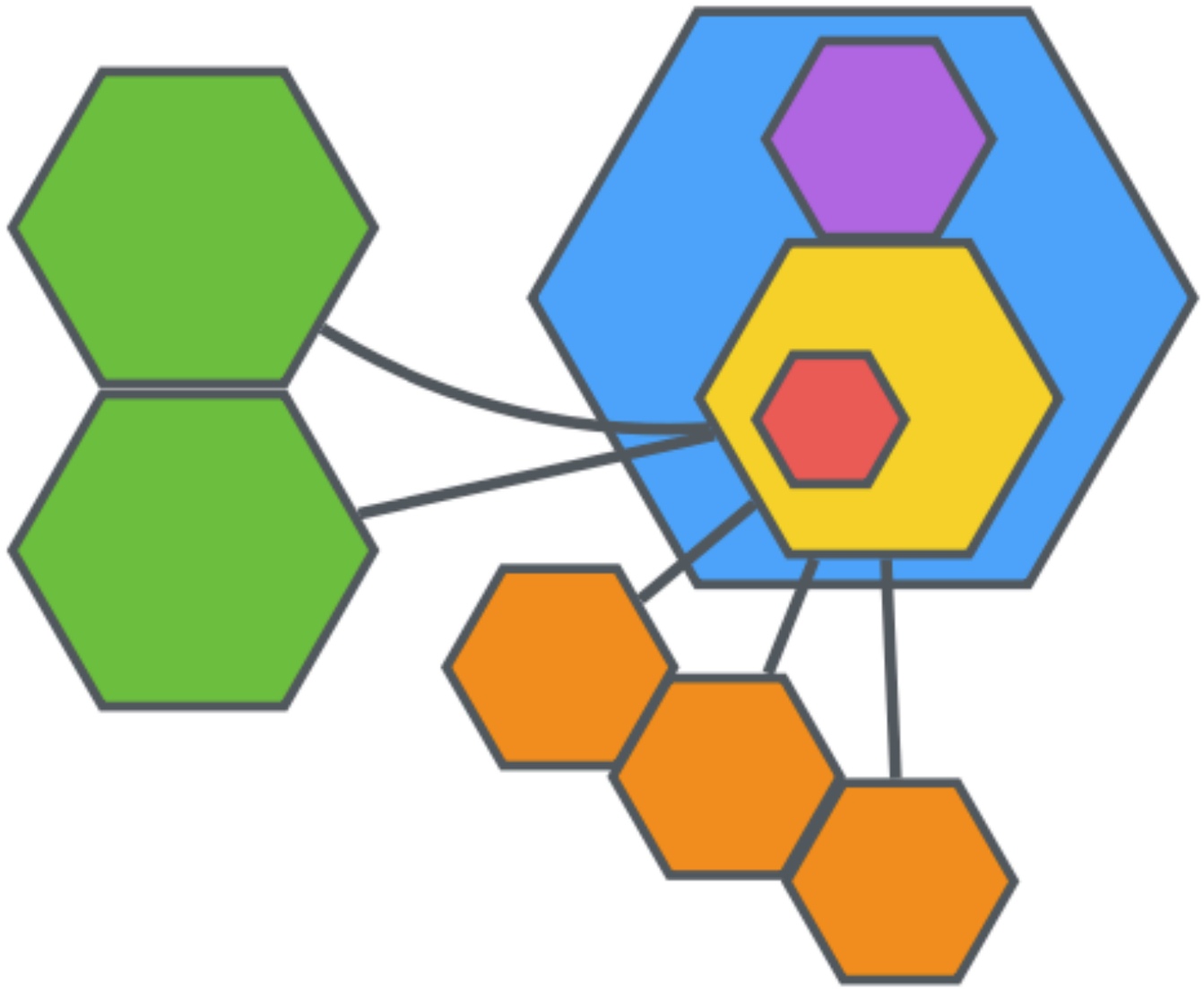


Monolith

Microservices

SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Microservices, from 1km high

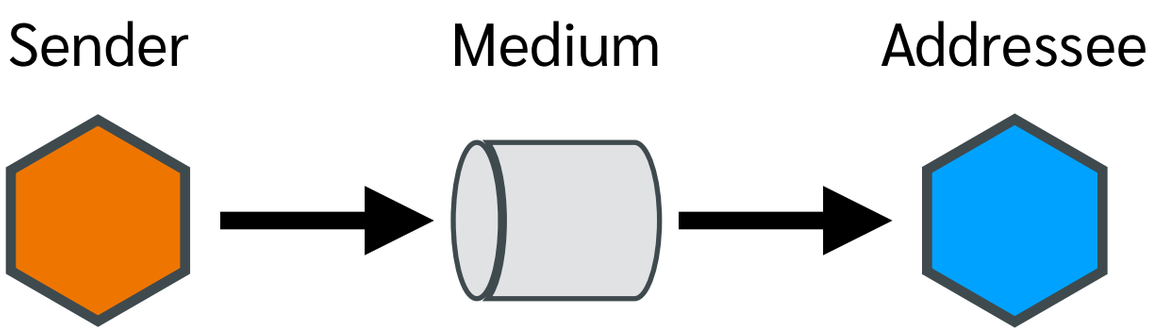


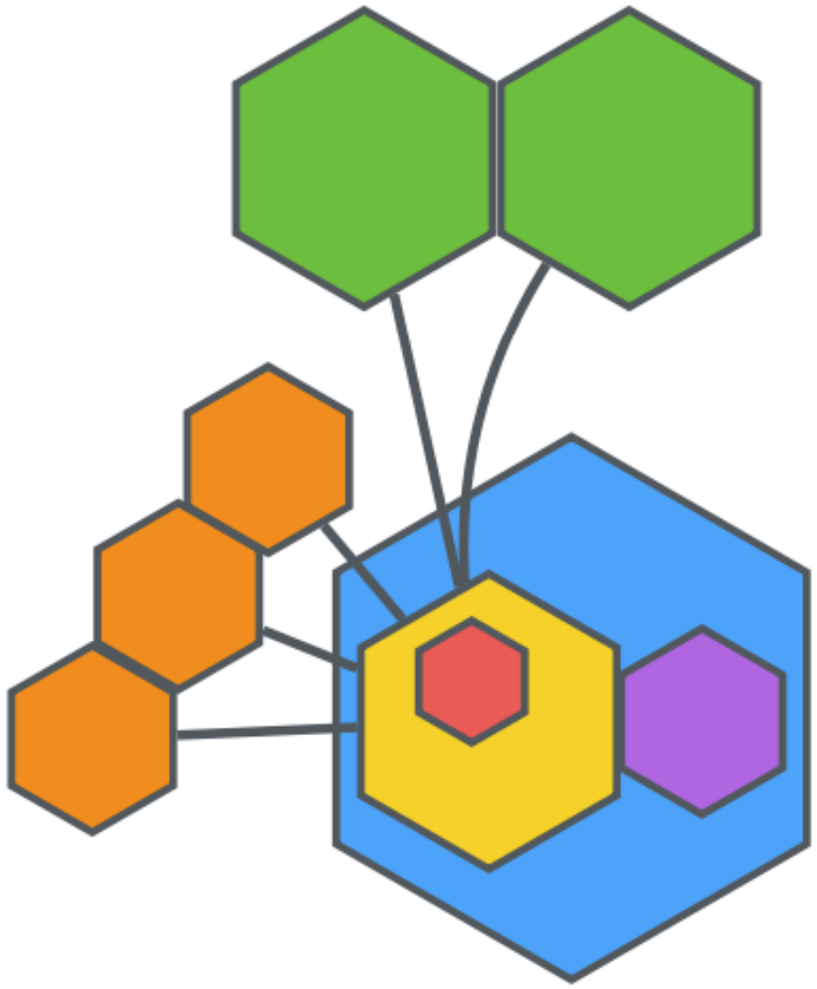
Independent, Scalable Software Components

SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

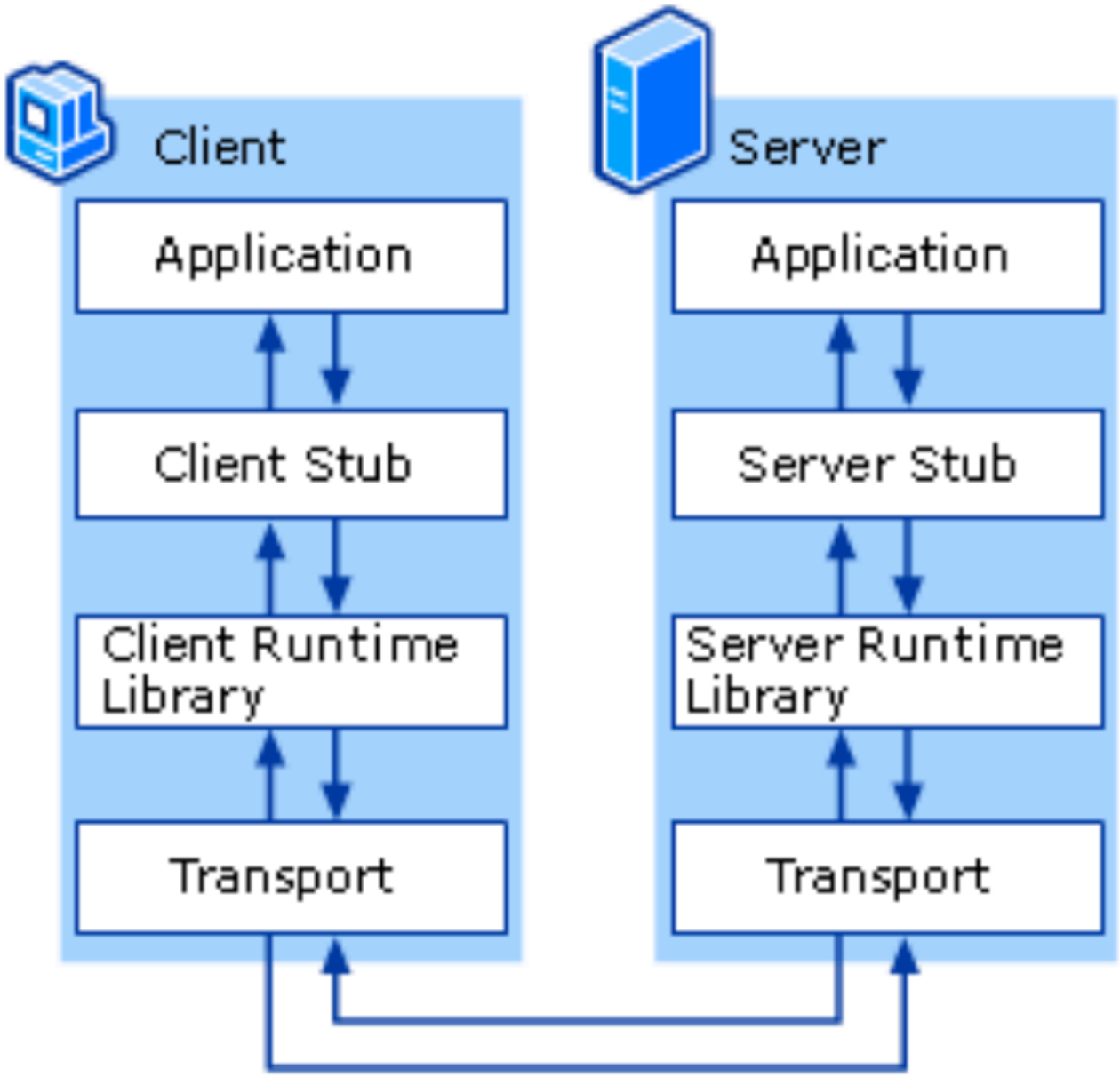
Microservices, from 1km high

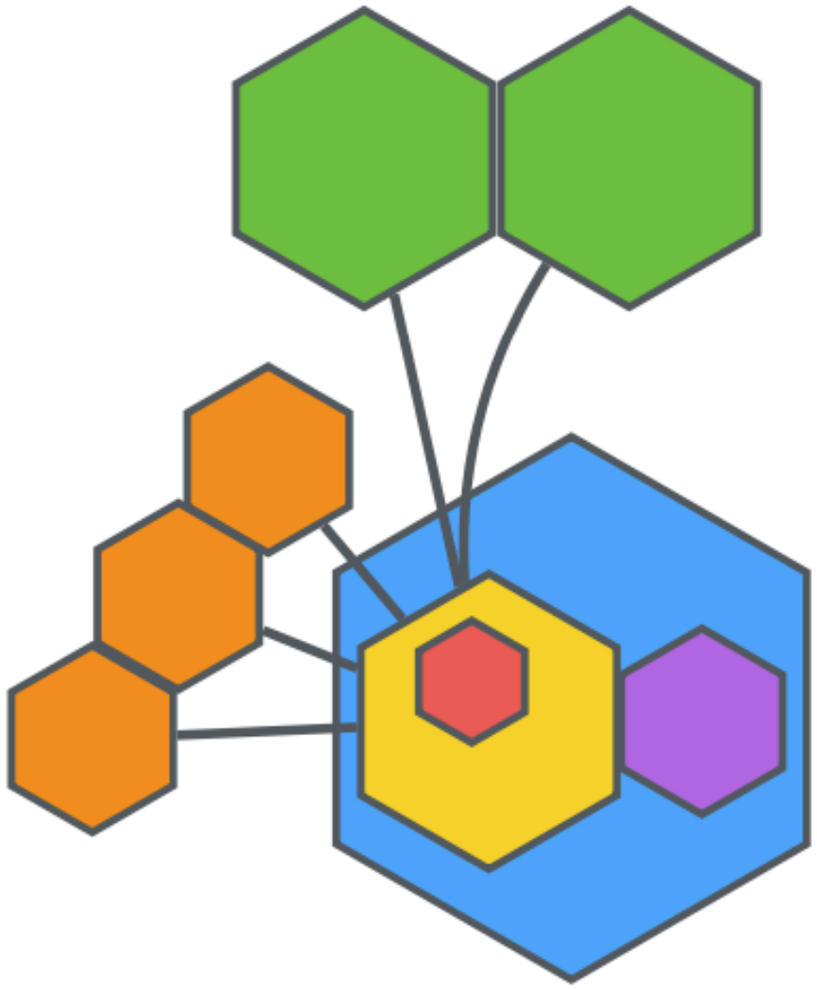
Message-based
Inter-process
Communications





Microservices, from 1km high



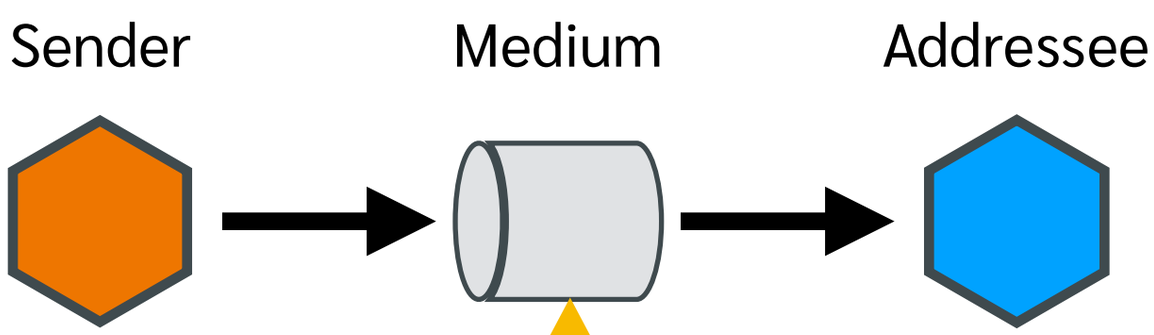


Message-based Inter-process Communications

SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Microservices, from 1km high

Message-based Inter-process Communications

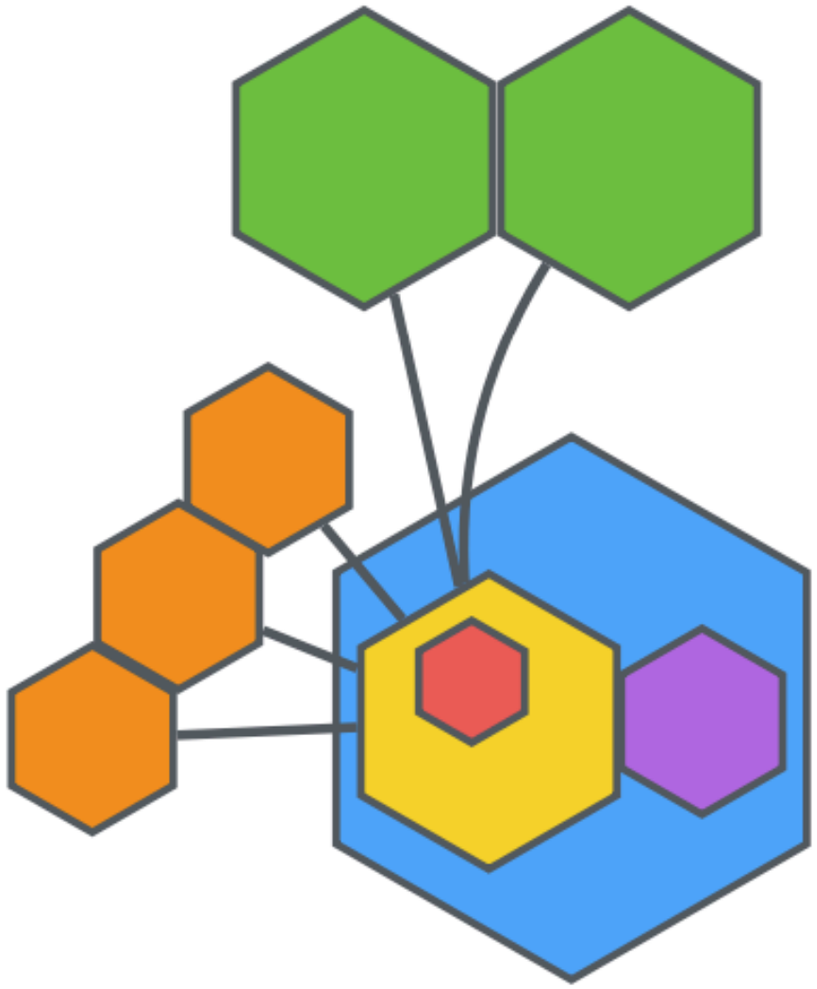


Pattern: Request-Response, Publish/Subscribe

Application: HTTP, SOAP, RabbitMQ, MQTT, COAP

Transport: TCP, UDP, Hybrids (QUIC), Serial, RAW

Link: IEEE 802 (.3 Ethernet, .11 WiFi), Unix Sockets



SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

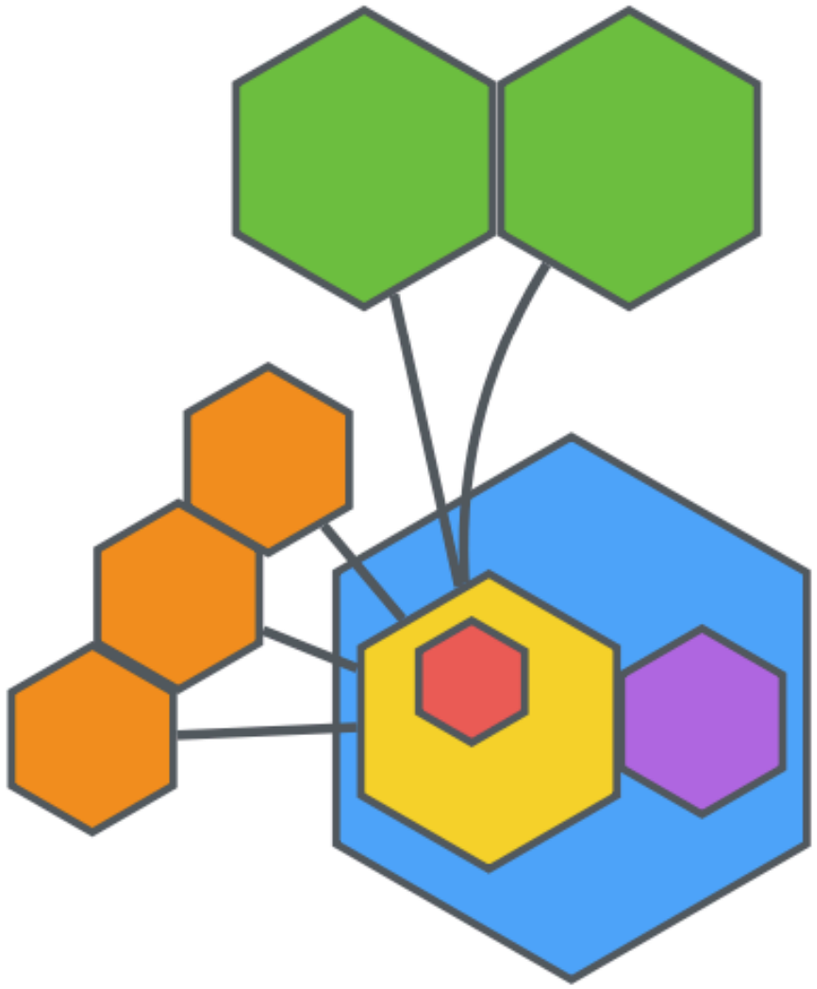
Microservices, from 1km high

Composition via Application Programming Interfaces


Sapori in cantina
AGRITURISMO

I Nostri Piatti

Antipasto Rustico (a-b) (9-10)	€ 10,00
Antipasto di Verdure dell'Orto* (a) (9-10)	€ 10,00
Selezione di Salumi (a-b) con la Giardiniera (a) (9)	€ 10,00
Polenta (c) e Salame (a-b)	€ 8,00
Tomino alla Griglia (b) con Misticanza (a) (7)	€ 8,00
Tortino al Pomodoro e Basilico (a) con Crostini (c) (1)	€ 8,00



SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Microservices, from 1km high

Machine Processable

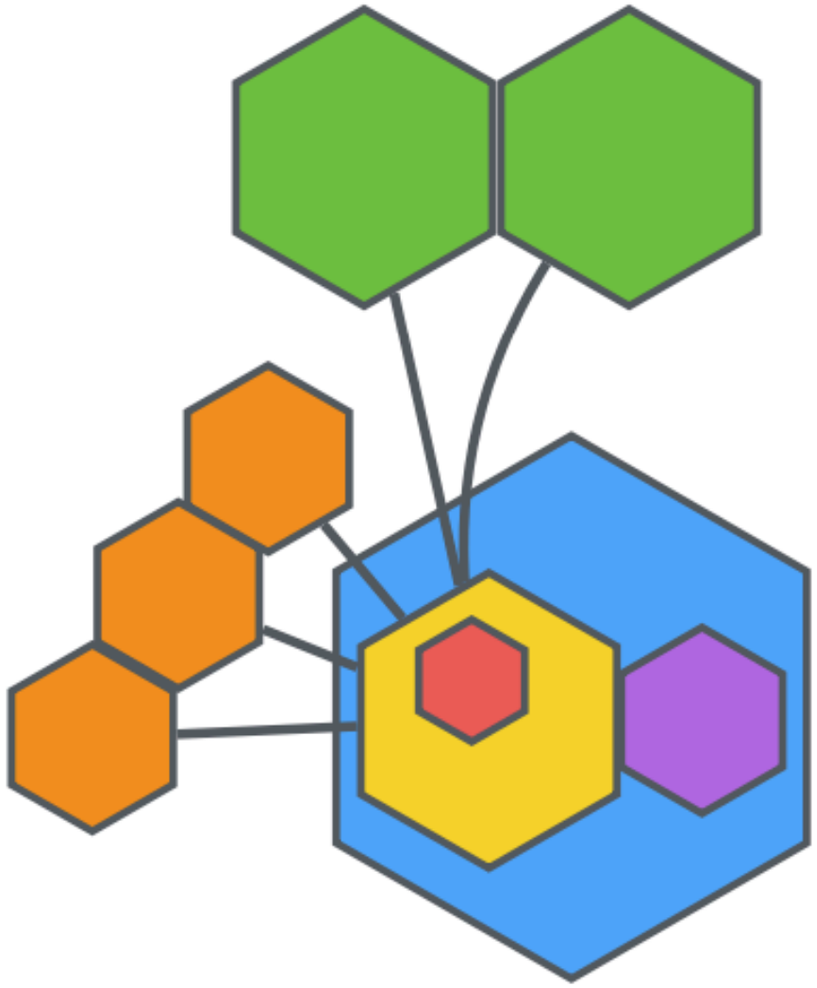
Composition via Application Programming Interfaces

They are in mainstream languages

```
Interface I {
 List< Integer > getRequest()
}
```

Why not to program microservices?

```
type RequestType: { item*: int }
interface I {
 One-Way: getRequest( Request )
}
```

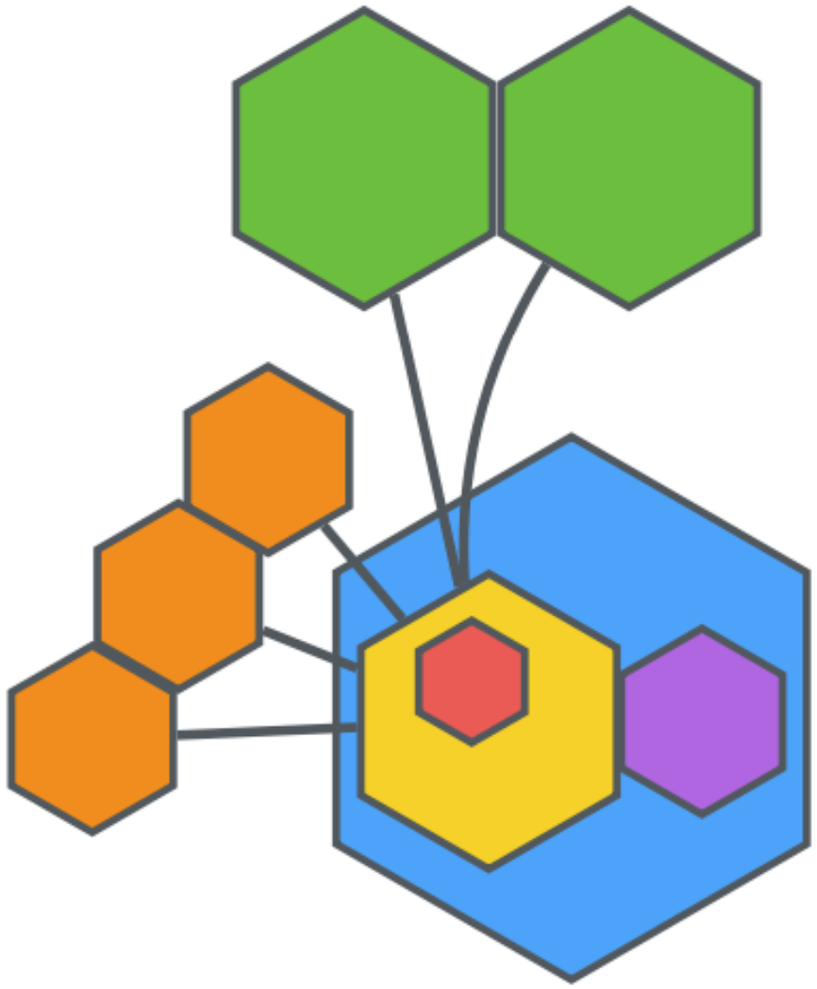


SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Microservices, from 1km high

😩 Disadvantages

- **Increased Complexity**
  - of unit- and integration-testing
  - of monitoring
  - of ensuring availability (non-binary system status)

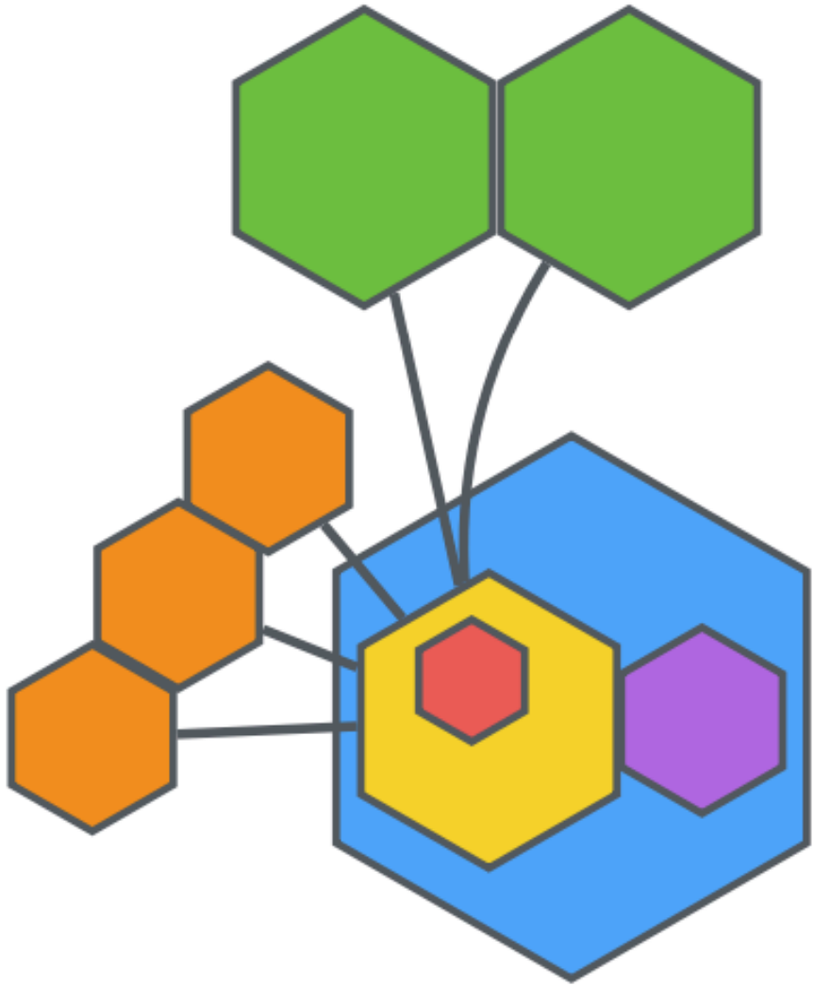


SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

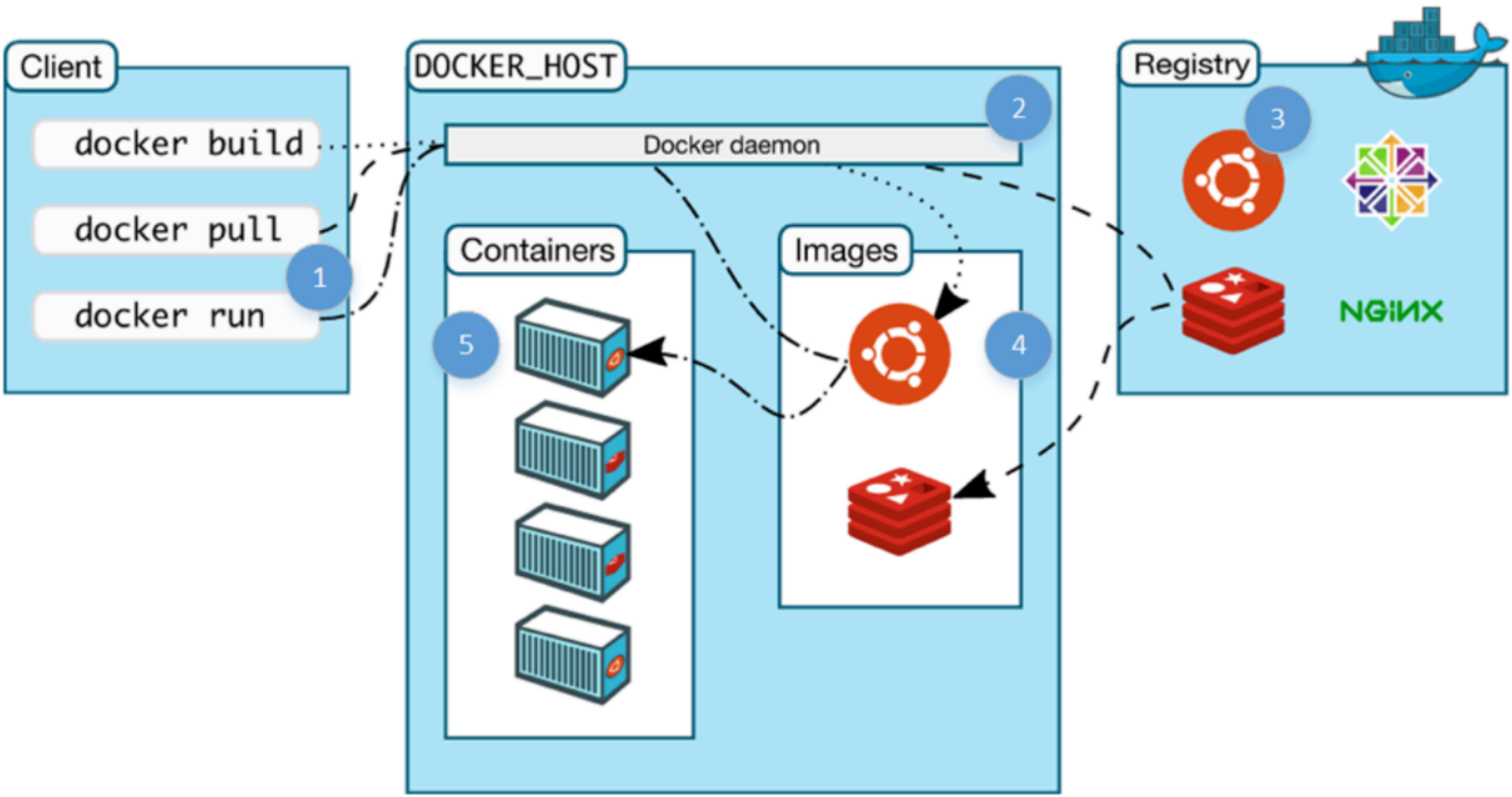
Microservices, from 1km high

💪 Advantages

- **Independence**
  - of development
  - of scalability
  - of reuse
- **Agility**
  - small codebase to maintain
  - contained outages
- **Flexibility** to match the business capabilities/structure



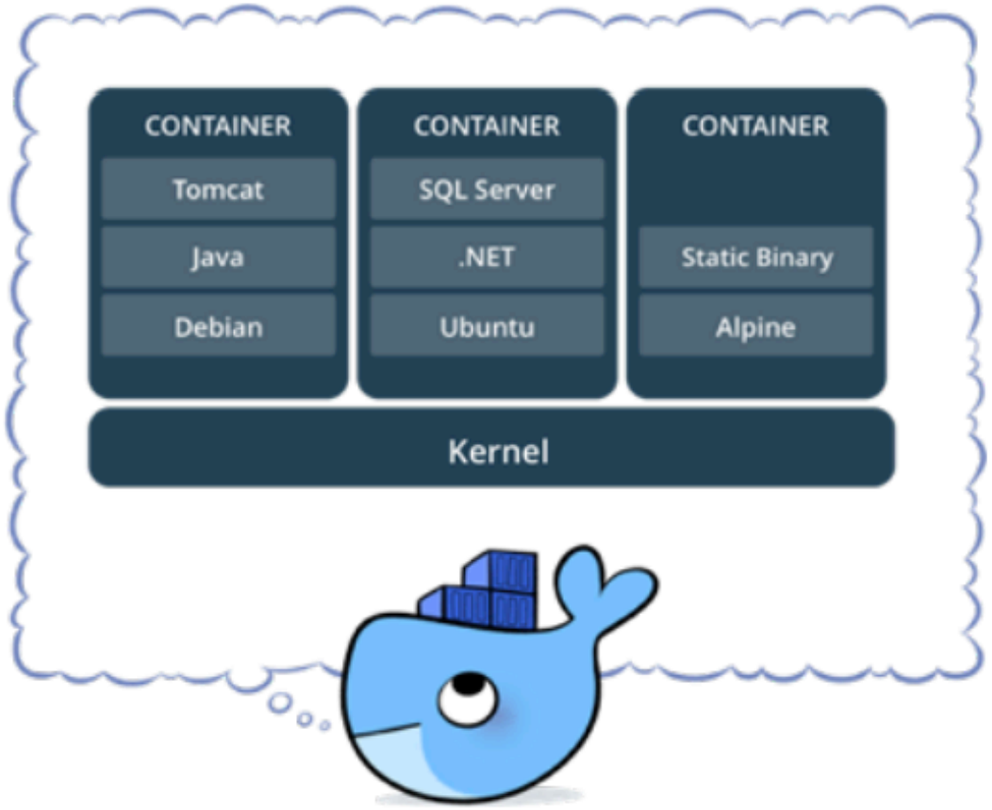
Microservices ≠ Containers



Microservices ≠ Containers

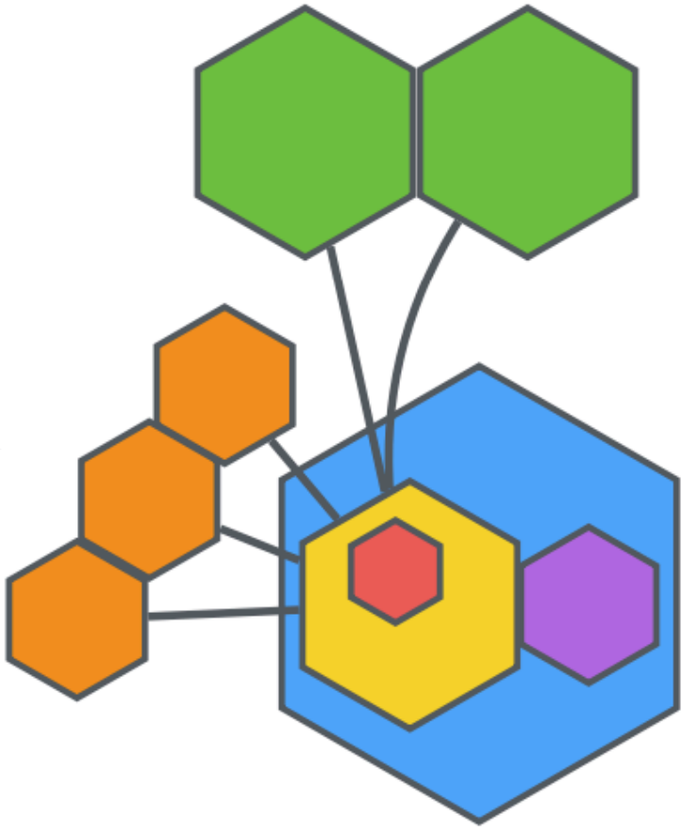
System Deployment

Independent applications enclosed within **containers**.



System Programming

Independent **microservices**, possibly enclosed within containers.

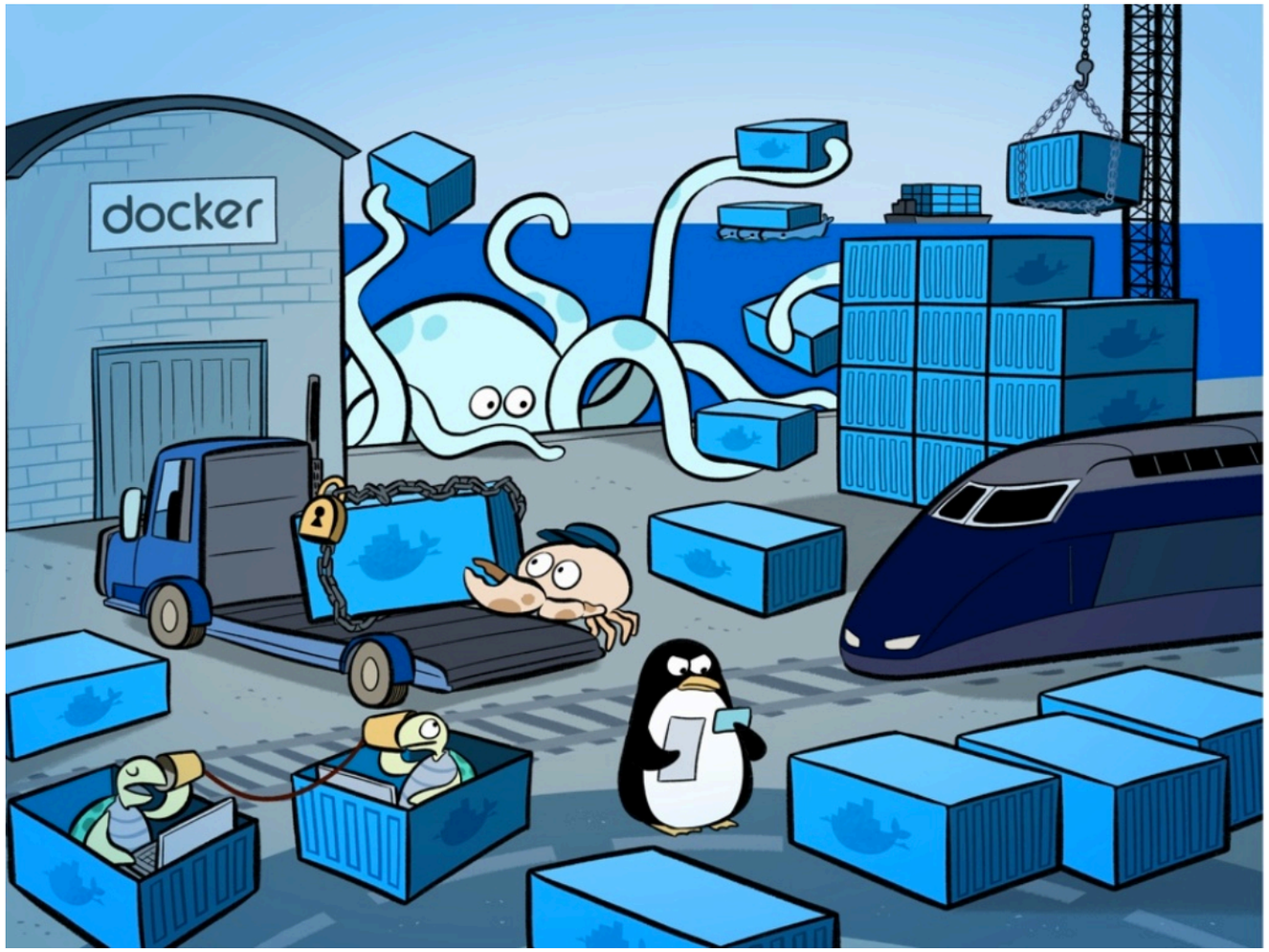


Microservices ≠ Containers

Here, we do not focus on containers. However, they play well with microservices.

Some pointers on security (by NIST):

- **Application Container Security Guide**
- **Security Assurance Requirements for Linux Application Container Technologies**



Microservices Security

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

"Promote U.S. innovation and industrial competitiveness ***by advancing*** *measurement science,* ***standards,*** *and* ***technology*** *in ways that* ***enhance economic security*** *and* ***improve*** *our* ***quality of life.****"*

- NIST's official mission

Microservices Security

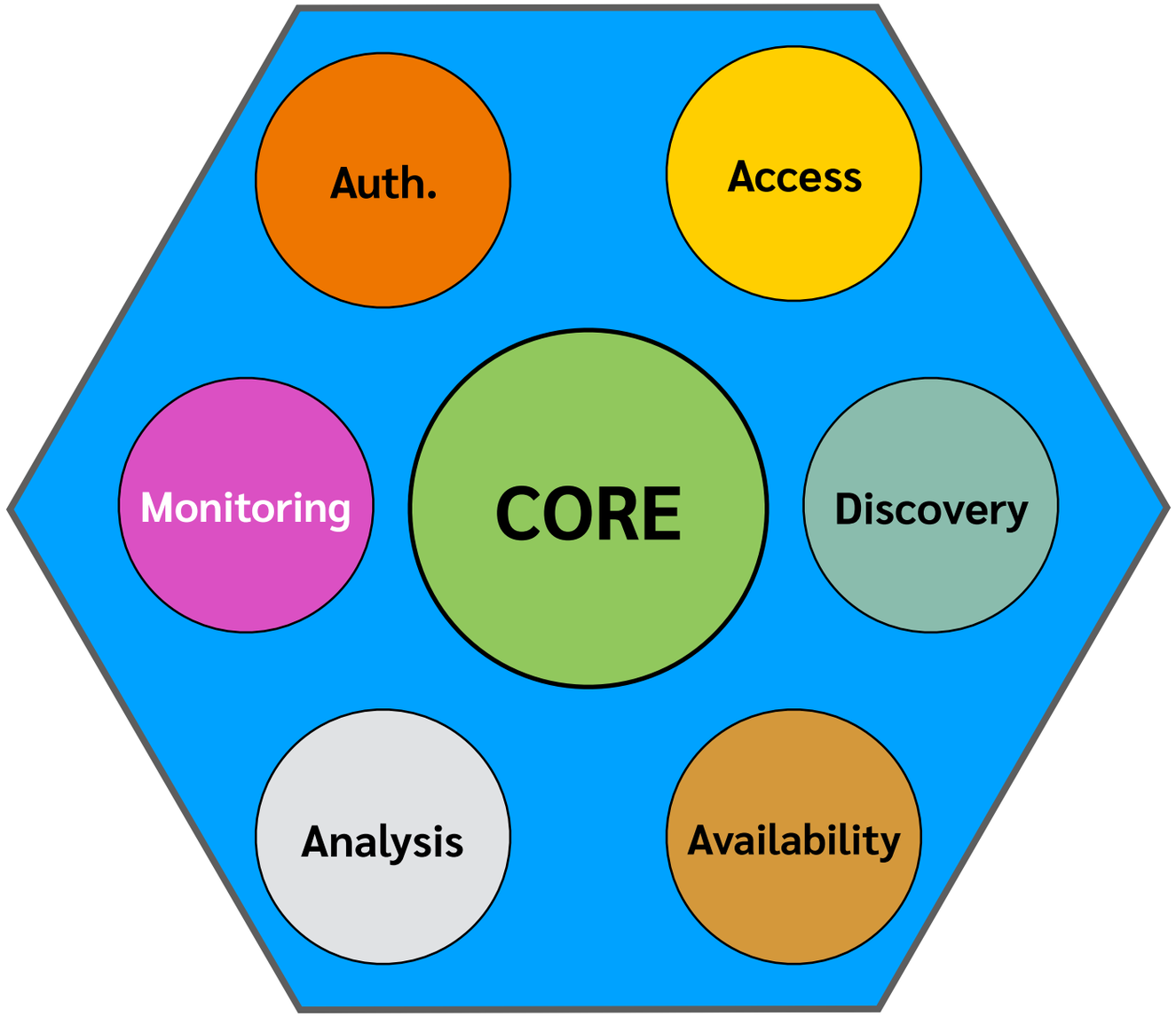
Reference source for this seminar

NIST

Security Strategies for Microservice-based Application Systems

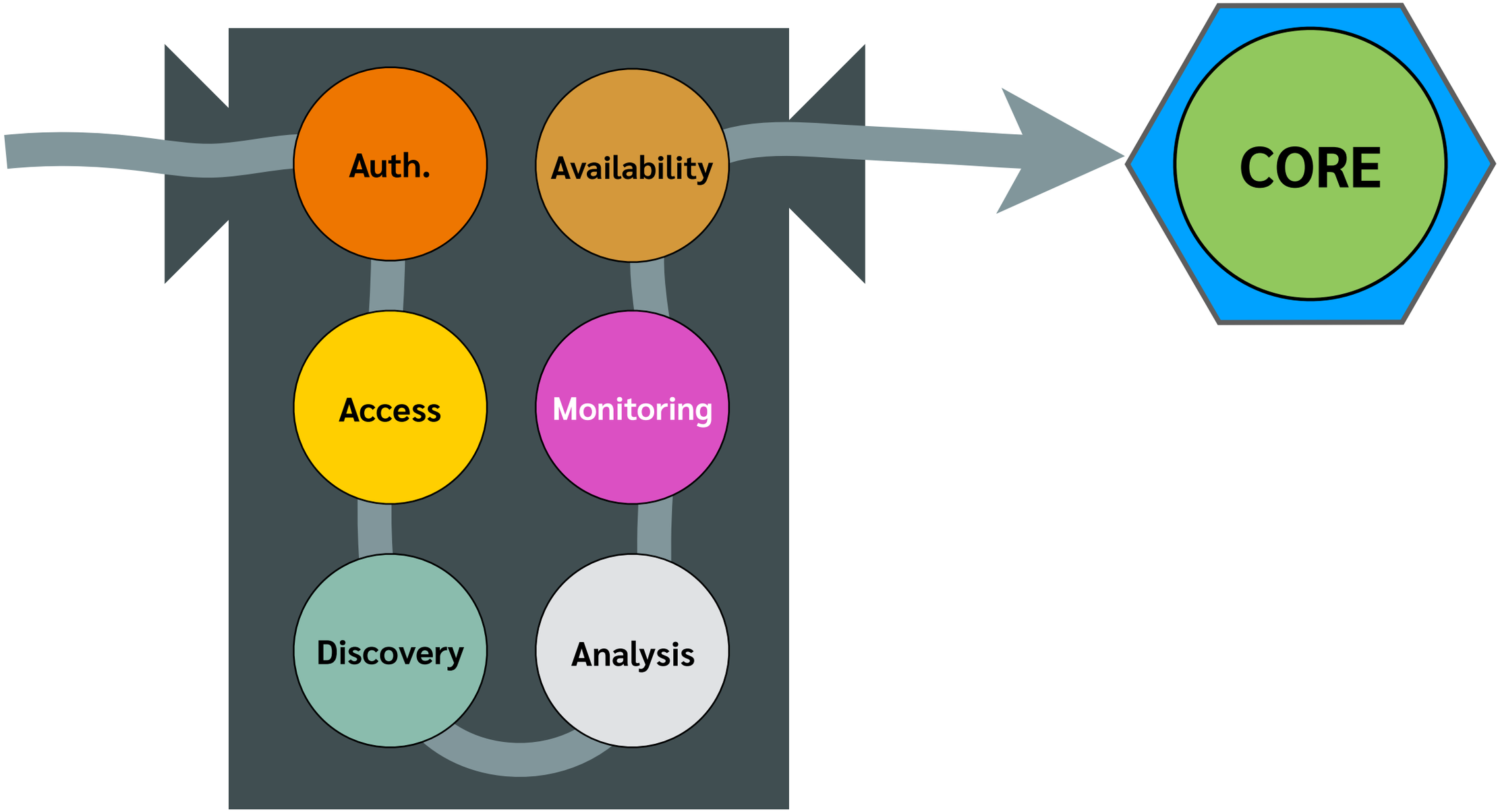
Approaches to control

Standalone



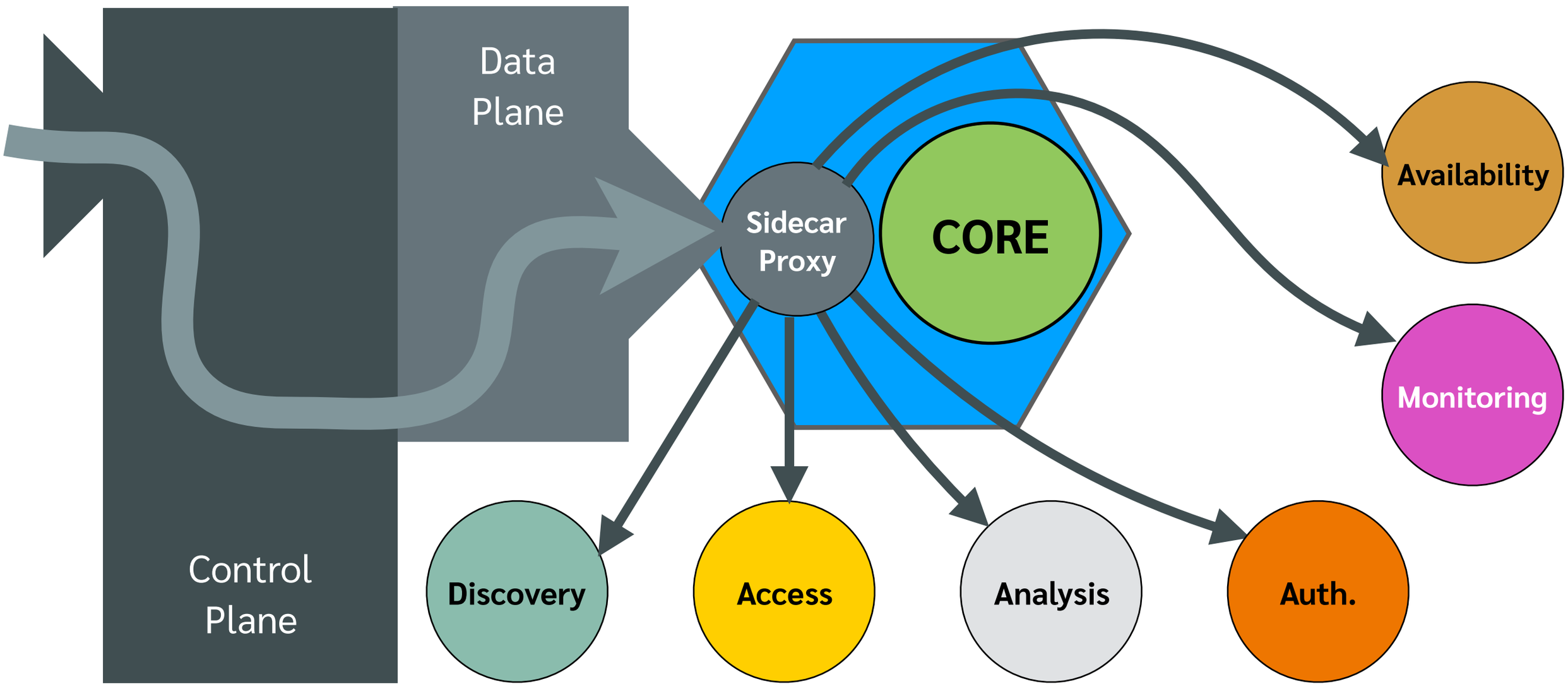
Approaches to control

Gateway



Approaches to control

Service Mesh



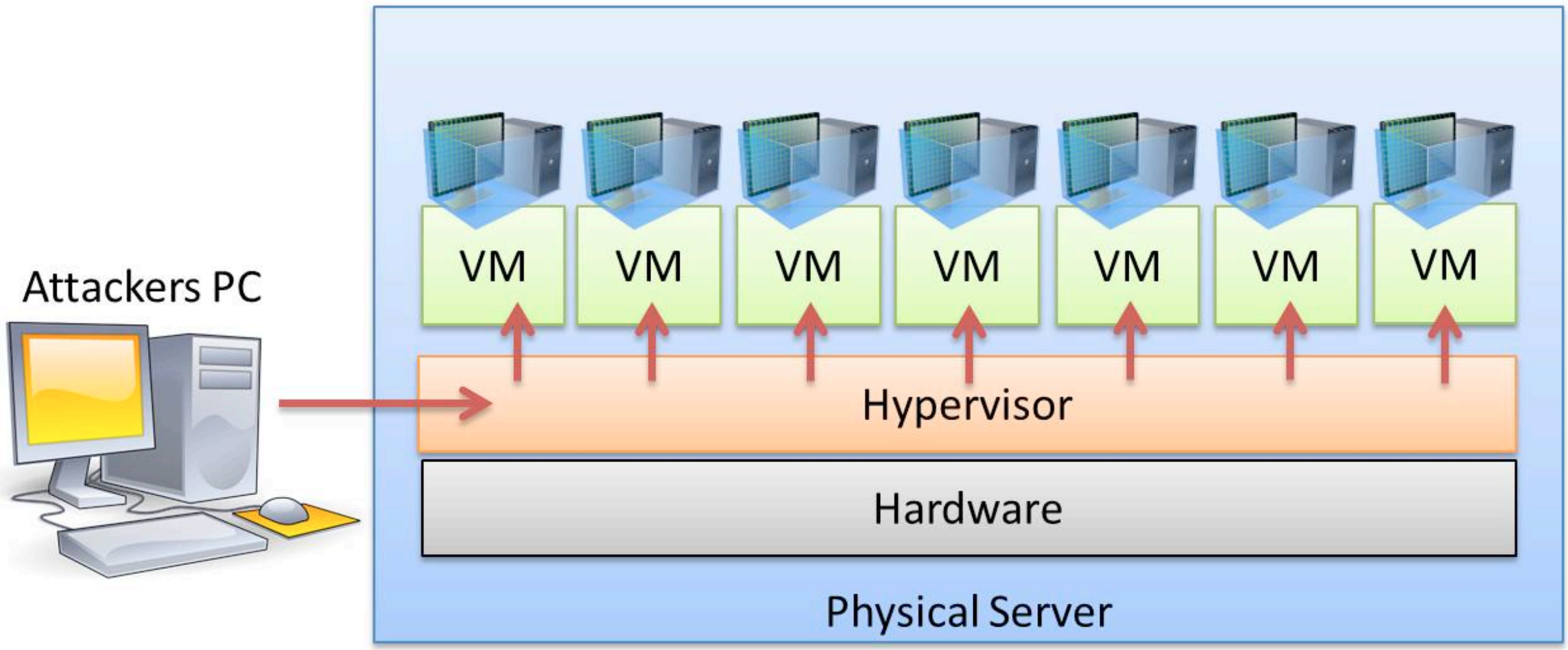
Generic Threats

Hardware

SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

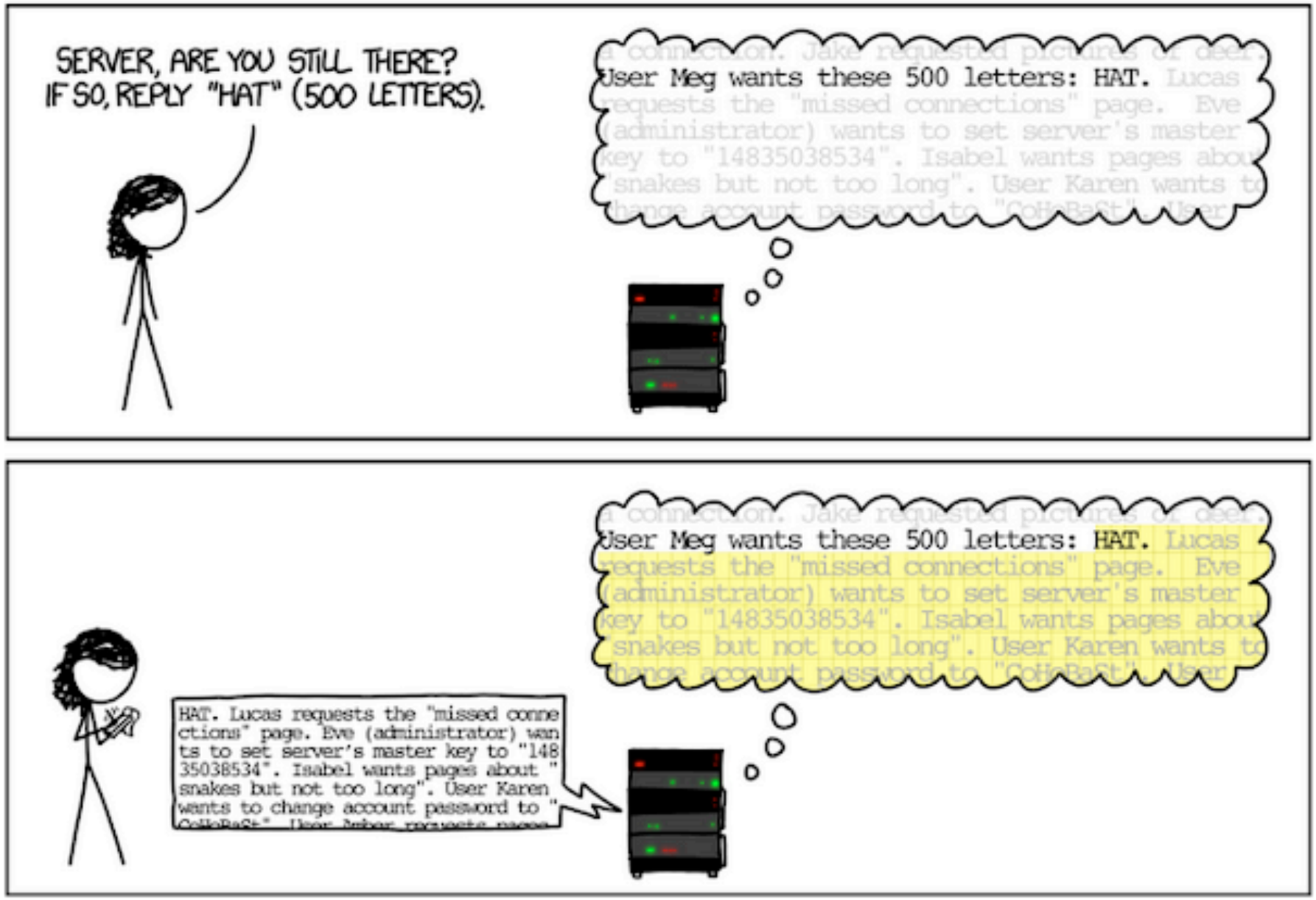
Generic Threats

Virtualisation, Containers, and the Cloud Infrastructure



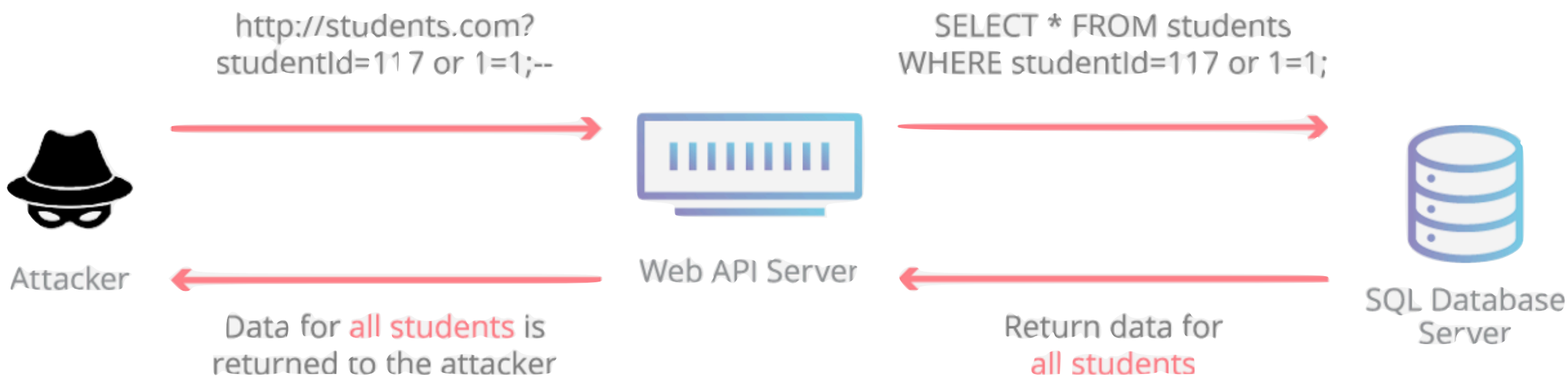
Generic Threats

Communication and Application Layers



Generic Threats

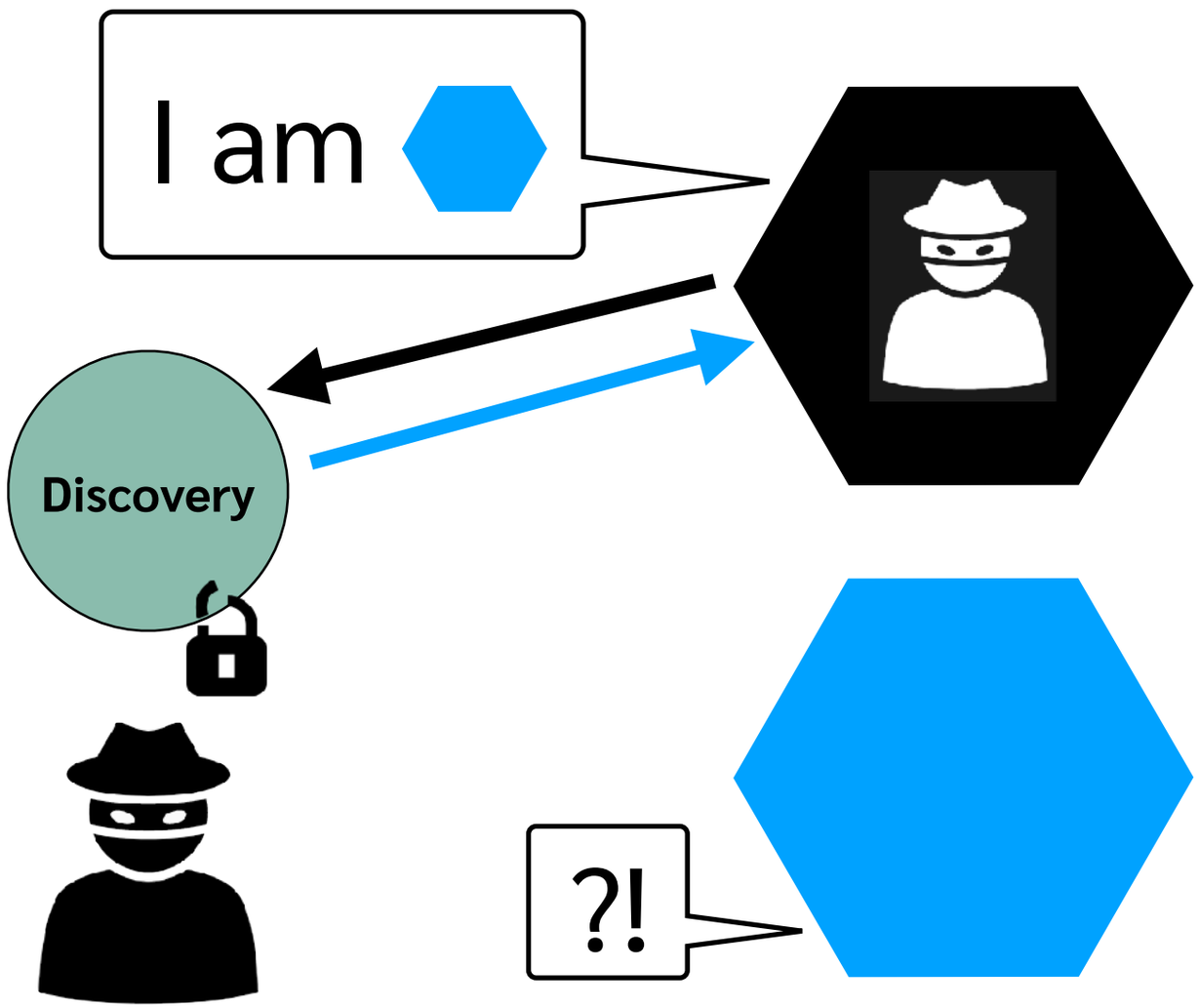
Communication and Application Layers



SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Specific Threats

Service Discovery

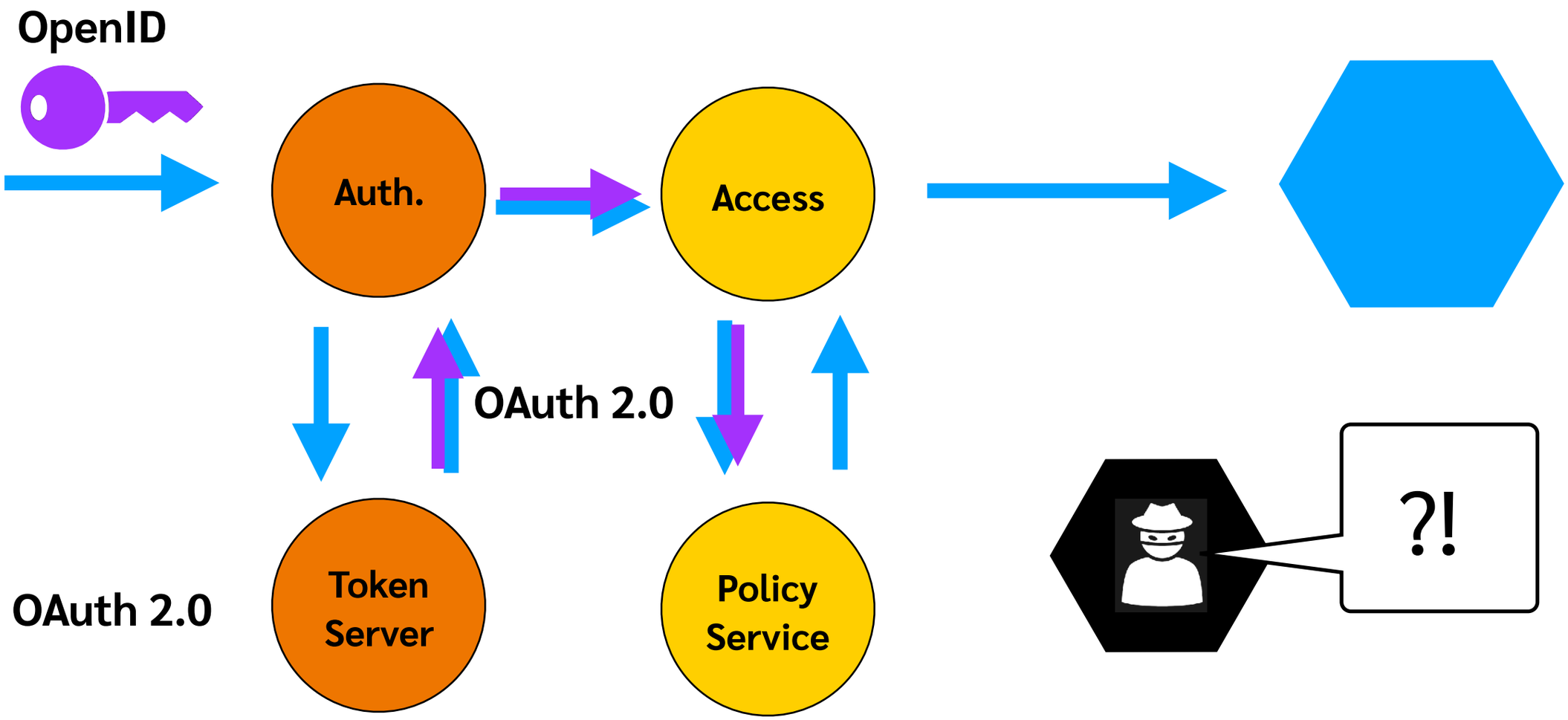


Mitigations

- DMZ for discovery servers
- cluster of distributed discovery servers
- authenticated and secured communications for service registration/discovery
- sanity/health service check before registry modification

Specific Threats

Authentication and Access Control



Specific Threats

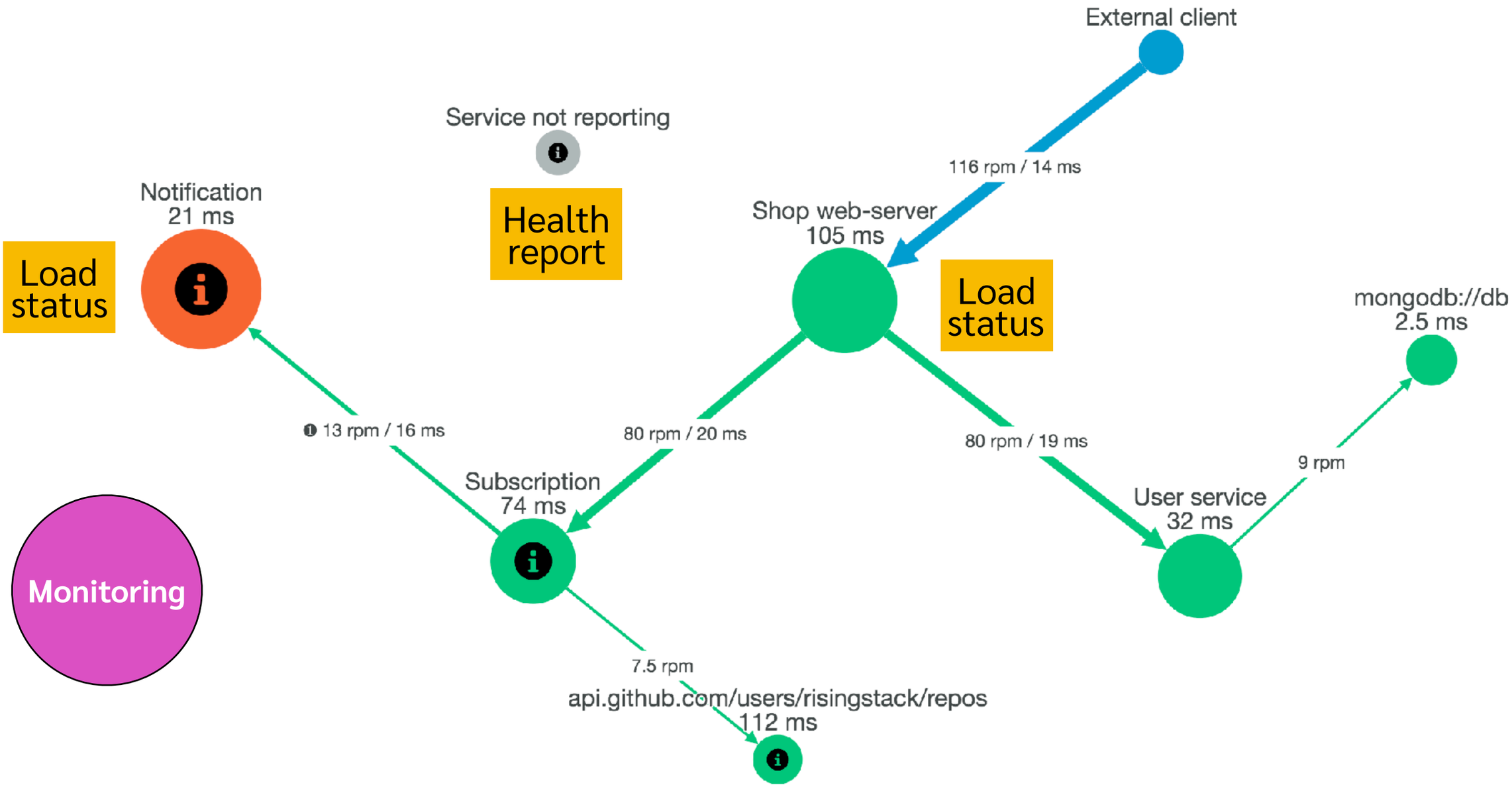
Secure Communication Protocols

Each microservice is an SSL/TLS endpoint.

Use keep-alive connections to mitigate handshake overhead

Specific Threats

Security Monitoring/Analysis



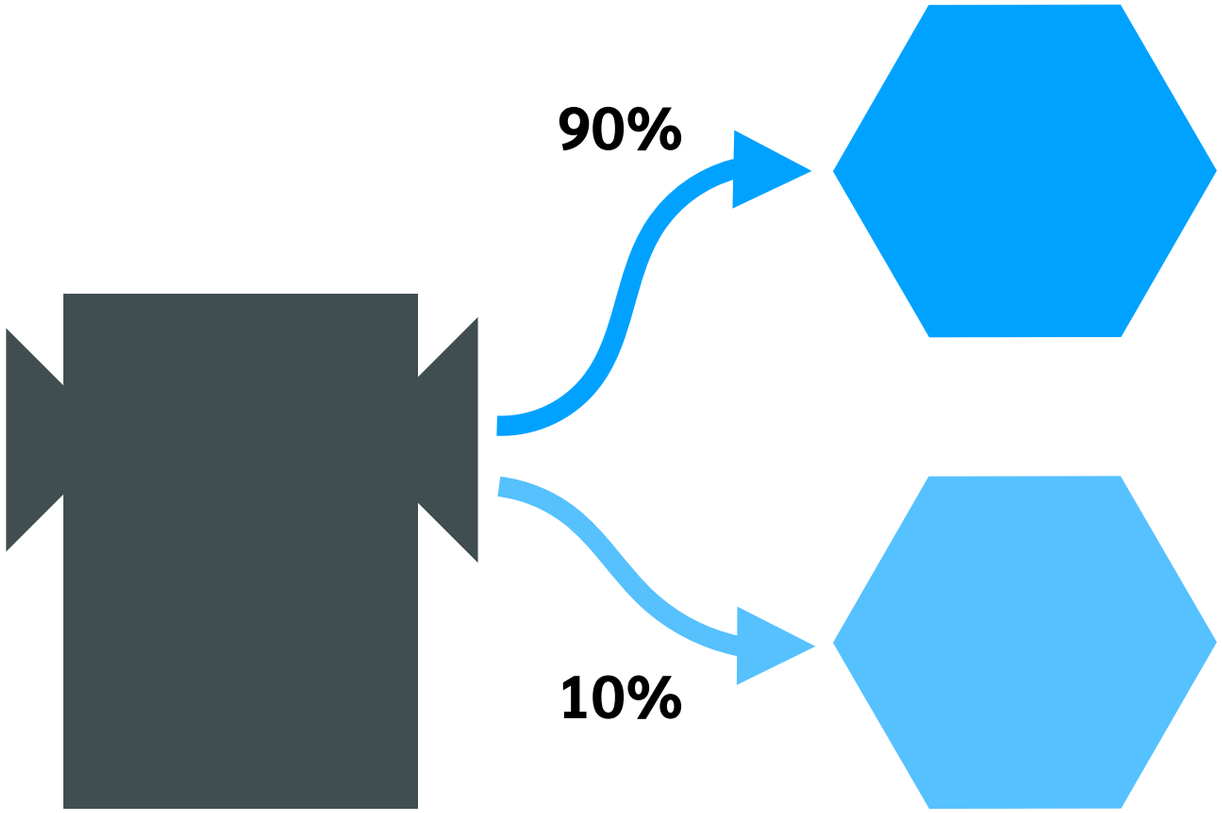
SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Specific Threats

Availability & Resilience

Integrity Assurance

Availability



Canary Release

- Gradually transition clients and sessions (next slide)
- Ensure expected behaviour of new release

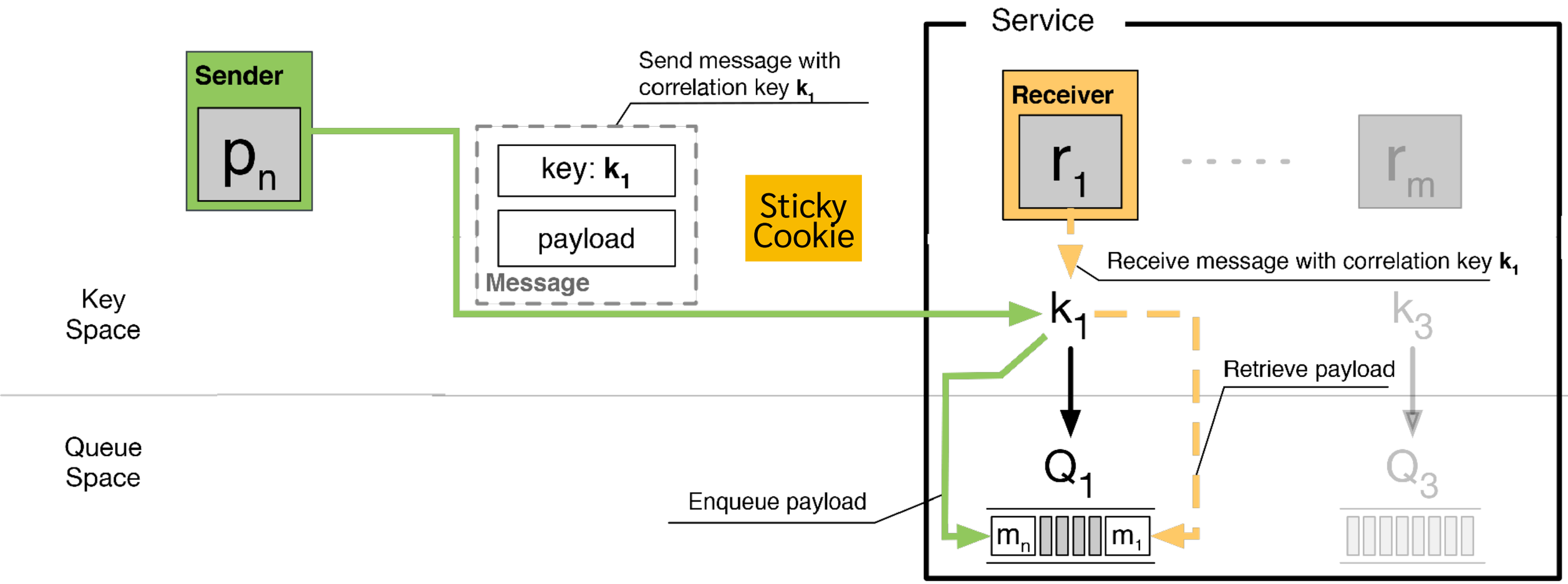
SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Specific Threats

Availability & Resilience

Session Persistence

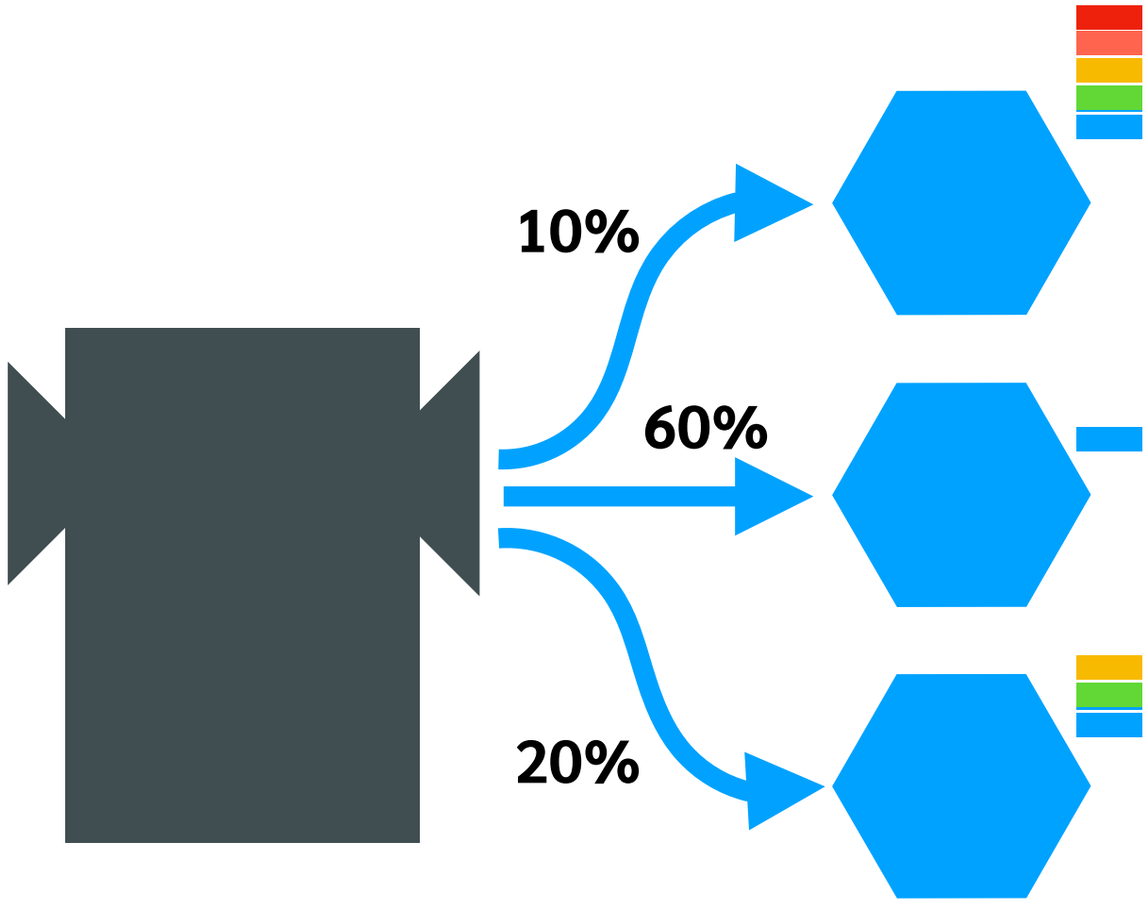
Availability



Specific Threats

Availability & Resilience

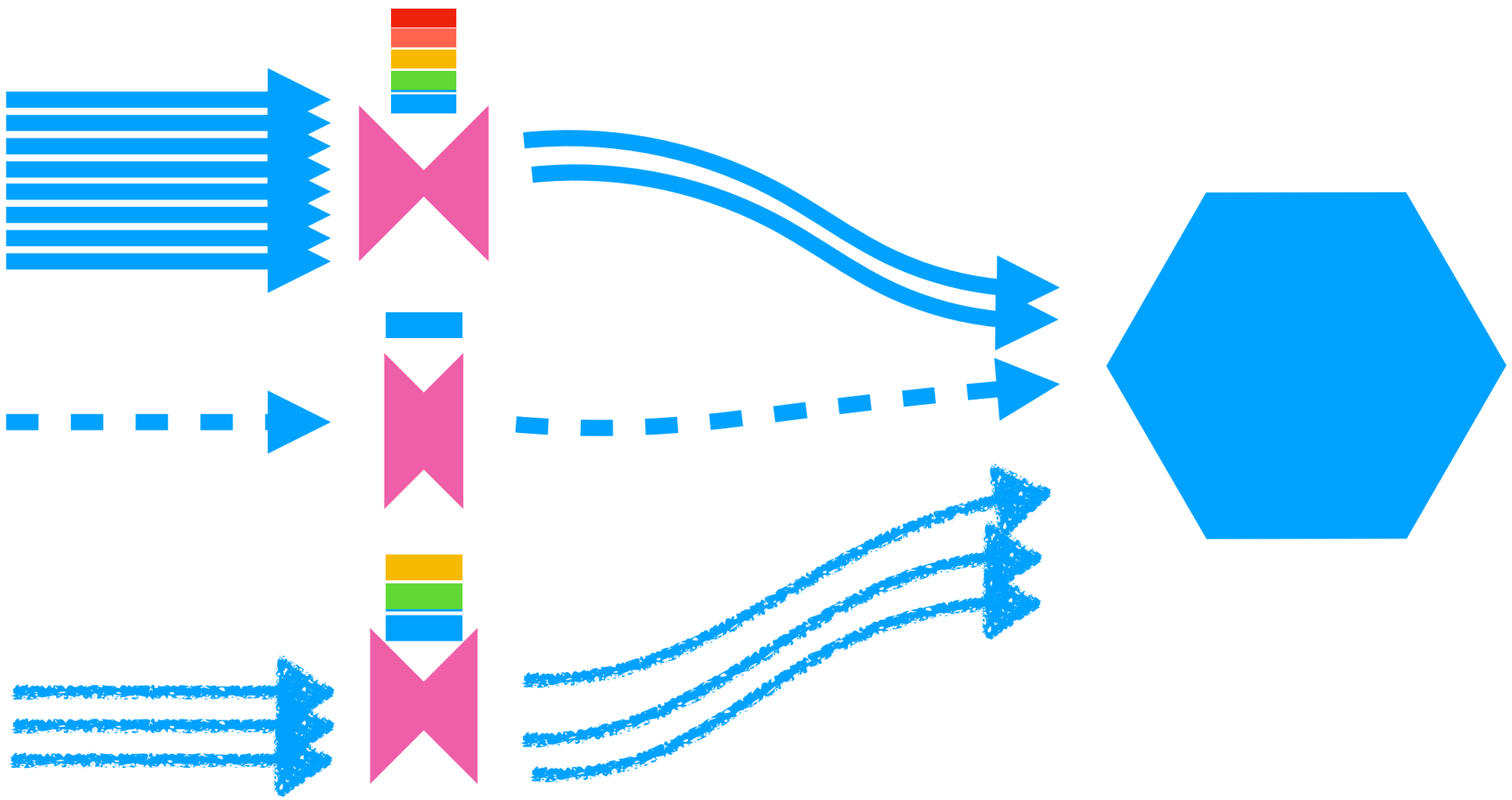
Load Balancers



Specific Threats

Availability & Resilience

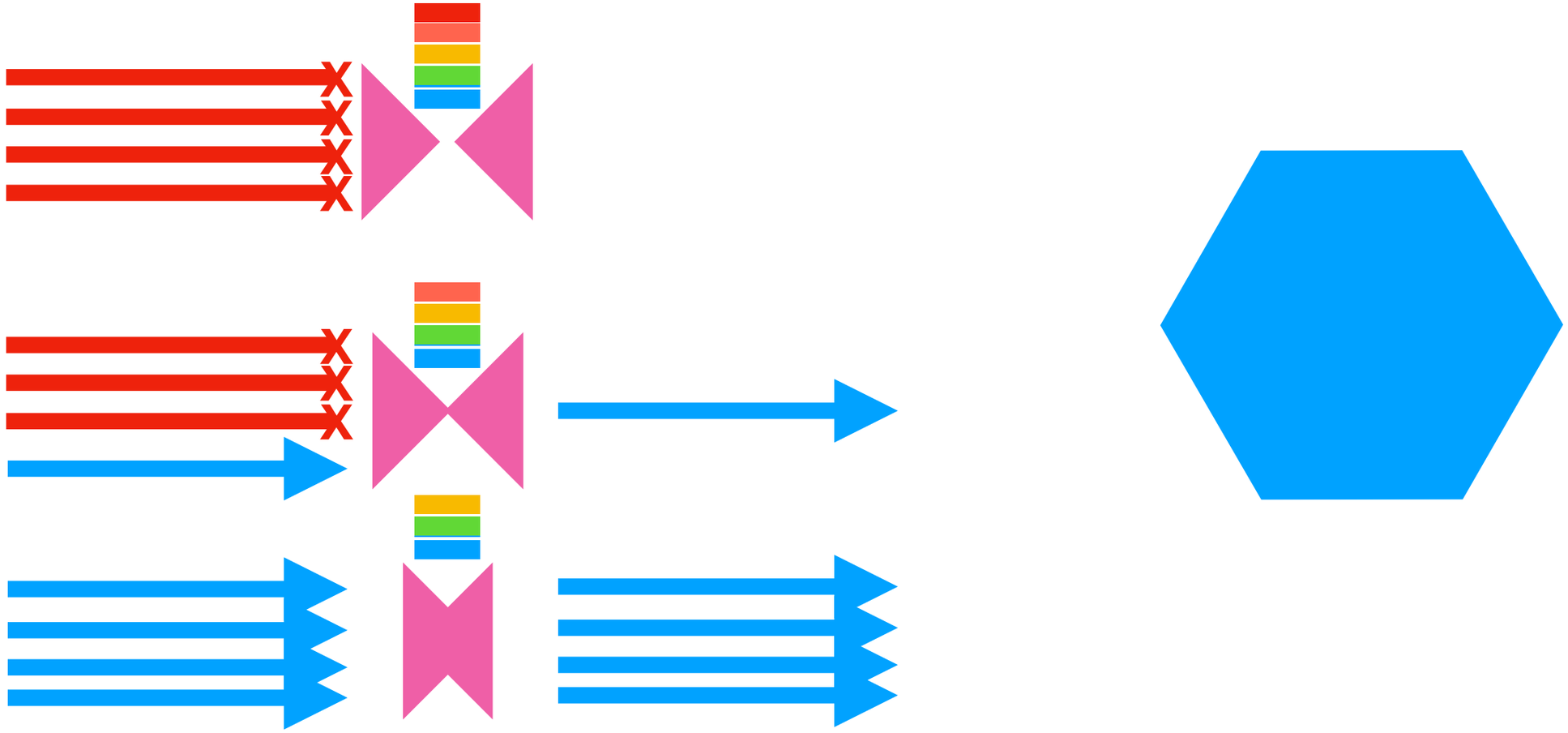
Throttlers



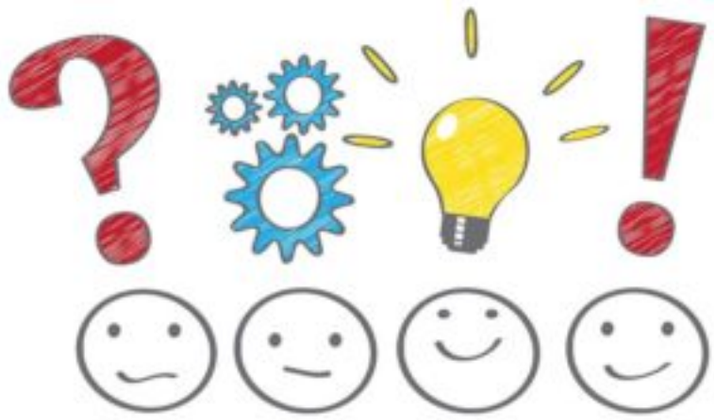
Specific Threats

Availability & Resilience

Circuit Breakers



Microservice Security Concepts



The first language for Microservices

https://www.jolie-lang.org

Why Jolie?

Jolie is perfect for fast prototyping. In little time a small team of developers can build up a full-fledged distributed system.

But I already know Java!
Why shall I use Jolie?

Why Jolie?

```
SocketChannel socketChannel = SocketChannel.open();
  socketChannel.connect(
new InetSocketAddress("http://someurl.com", 80));
  Buffer buffer = . . .; // byte buffer
  while( buffer.hasRemaining() ) {
     channel.write( buffer );
}
```

Happy?

Ok, but you did not even close the channel or handled exceptions

Why Jolie?

```
SocketChannel socketChannel = SocketChannel.open();
try {
  socketChannel.connect(new InetSocketAddress("http://someurl.com",
80));
  Buffer buffer = . . .; // byte buffer
  while( buffer.hasRemaining() ) {
    channel.write( buffer );
} }
catch( UnresolvedAddressException e ) { . . . }
catch( SecurityException e ) { . . . }
/* . . . many catches later . . . */
catch( IOException e ) { . . . }
finally { channel.close(); }
```

Happier now?

Yes, but what about the **server**?

Why Jolie?

```
Selector selector = Selector.open();
channel.configureBlocking(false);
SelectionKey key = channel.register(selector, SelectionKey.OP_READ);
while(true) {
  int readyChannels = selector.select();
  if(readyChannels == 0) continue;
  Set<SelectionKey> selectedKeys = selector.selectedKeys();
  Iterator<SelectionKey> keyIterator = selectedKeys.iterator();
  while(keyIterator.hasNext()) {
    SelectionKey key = keyIterator.next();
    if(key.isAcceptable()) {
        // a connection was accepted by a ServerSocketChannel.
    } else if (key.isConnectable()) {
        // a connection was established with a remote server.
    } else if (key.isReadable()) {
        // a channel is ready for reading
    } else if (key.isWritable()) {
        // a channel is ready for writing
    }
    keyIterator.remove();
  }
}
```

Here you are

SDU
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Why Jolie?

Well, ok, but again, you are not **handling exceptions**.

And what about if **different operations** use the **same channel**?

And if we wanted to use **RMIs** instead of **Sockets**?

In what **format** are you transmitting data? And if we need to **change** the **format** after we wrote the application? Do you **check** the **type of data** you receive/send?

Why Jolie?

Programming distributed systems is usually harder than programming non distributed ones.

Concerns of **concurrent** programming.

Plus (not exhaustive):

- handling **communications**;
- handling **heterogeneity**;
- handling **faults**;
- handling the **evolution** of systems.

Why Jolie?

Applications in a distributed system can perform a **distributed transaction**.

Example:

- a client asks a store to buy some music;
- the store opens a request for handling a payment on a bank;
- the client sends his credentials to the bank for closing the payment;
- the store sends the goods to the client.

Looks good, but a lot of things **may go wrong**, for instance:

- the store (or the bank) could be offline;
- the client may not have enough money in his bank account;
- the store may encounter a problem in sending the goods.

Why Jolie?

Things can be made easier by **hiding the low-level** details.

Two main approaches:

- make a library/tool/framework for an existing programming language;
- make a new programming language.

Can you tell the difference between the two approaches?

Why Jolie?

Strong foundations from Academia

ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA A.D. 1088

IT University of Copenhagen

Taught also in:

UNIVERSITAS STUDIORUM UTINENSIS

DTU

UNIVERSITAS STUDII PADUANI MCCXXII

innopolis university

Why Jolie?

It is a live **open source** project with continuous updates and a well documented codebase

https://github.com/jolie/jolie

"This *is* the programming language you are looking for"

Why Jolie?

"Hello World!" is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"

main
{
  println@Console( "Hello, world!" )()
}
```

Include a service

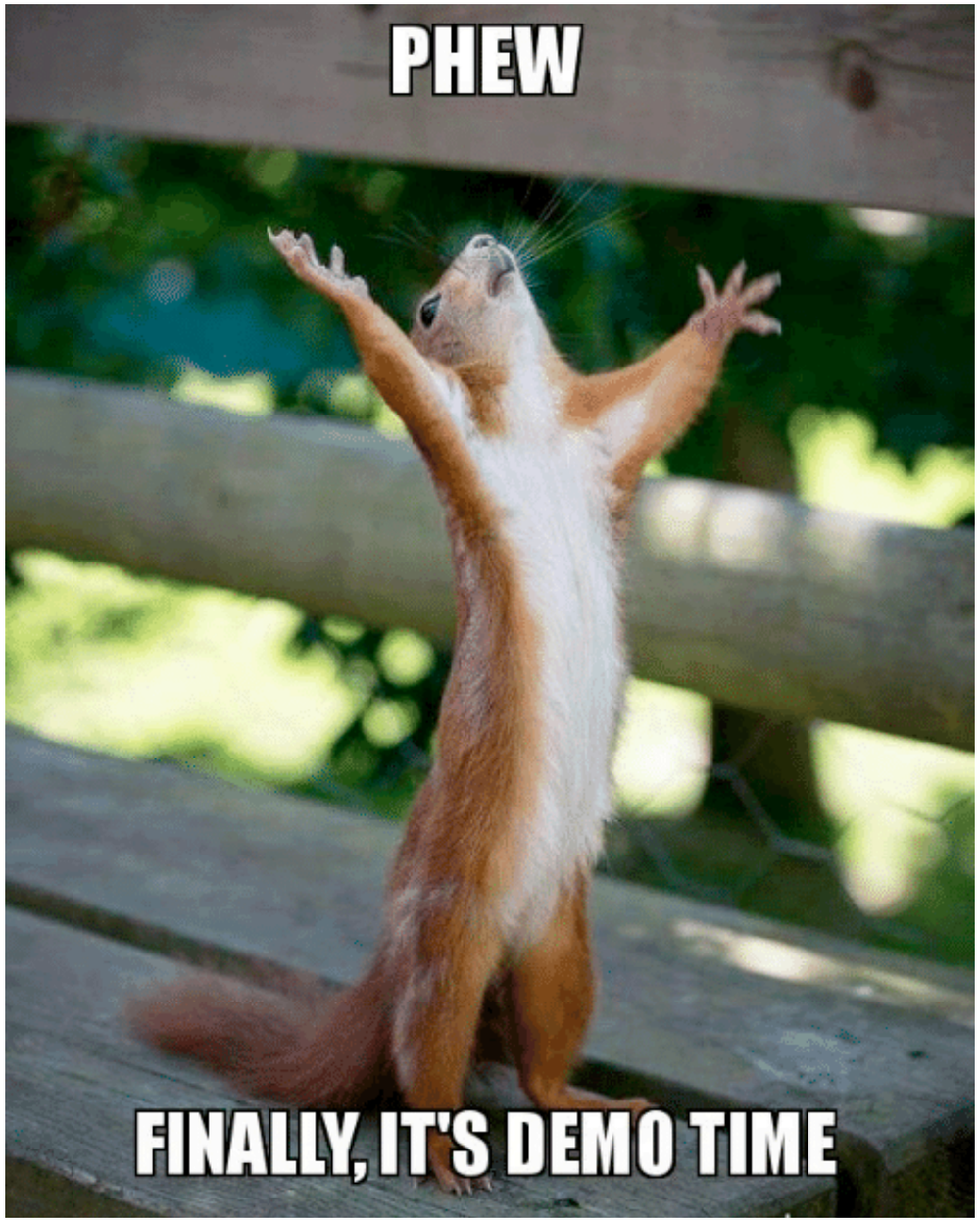
program entry point

operation

service

Resources | **Online**

- Official Website:
  - **http://www.jolie-lang.org**
- Official Docs:
  - **http://docs.jolie-lang.org**
- Official Codebase:
  - **https://github.com/jolie/jolie**



https://github.com/thesave/cybersecurity_summer_school_cph_2019