DATA FAIRNESS

LECCE 1-5 JUL 2019 INTERNATIONAL SUMMER SCHOOL FOR ENVIRONMENTAL & EARTH SCIENCE INFRASTRUCTURES
LIFEWATCH.EU/ISS-DATA-FAIRNESS

PLUS ENVRI

LifeWatch ERIC

Service oriented architecture and virtualized infrastructure for data management

Zhiming Zhao

z.zhao@uva.nl

UvA Universiteit van Amsterdam

About data management

Provenance…

Software engineering

Big data

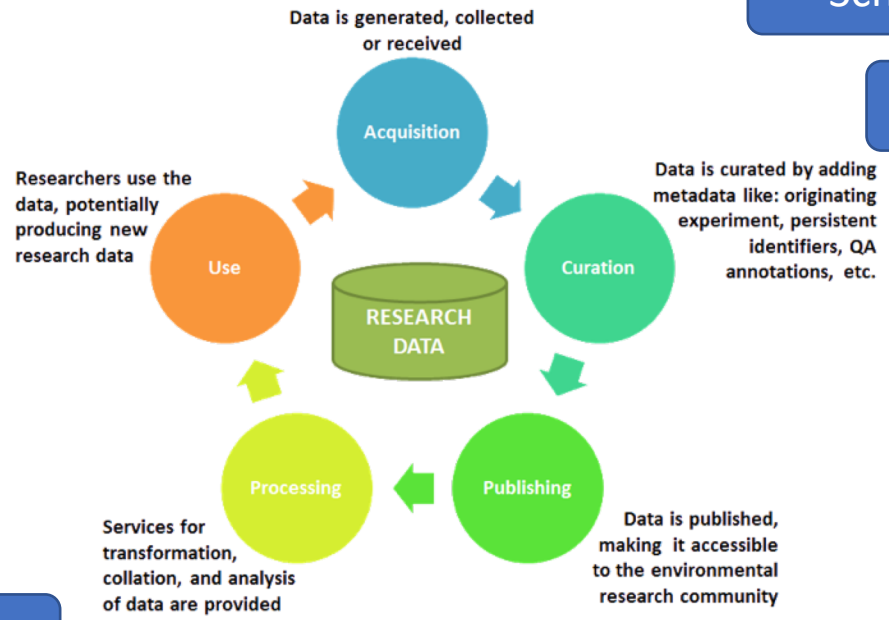
Cloud/HPC/HTC

E-Infrastructures

Service orientation

Workflow

Data Discovery



metadata

Schema, standards

cataloguing

Identification

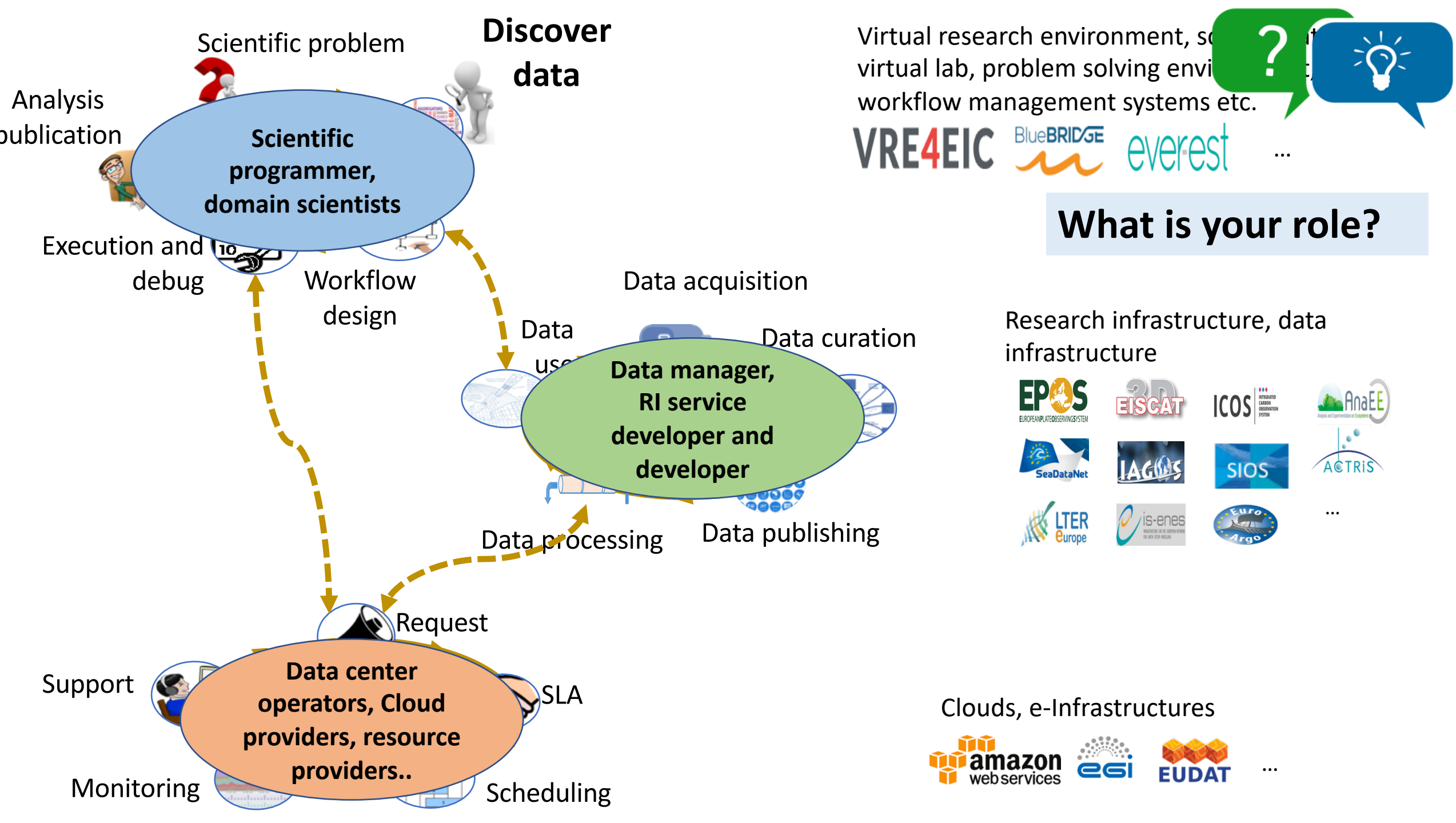
Publishing

Ontology

Semantics

License

Data management plan



Virtual research environment, s
virtual lab, problem solving envi
workflow management systems etc.

VRE4EIC BlueBRIDGE everest ...

What is your role?

Research infrastructure, data infrastructure

EPOS EISCAT ICOS AnaEE SeaDataNet IAGOS SIOS ACTRiS LTER Europe is-enes Euro-Argo ...

Clouds, e-Infrastructures

amazon web services egi EUDAT ...

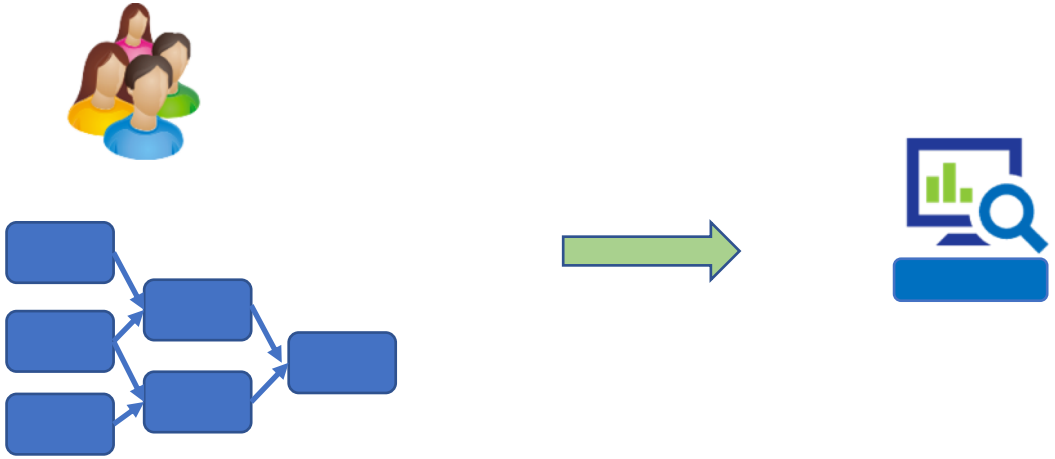
Objectives

- Scientific programmer, domain scientists
  - How to development process components and make them FAIR
  - How to effectively execute them using remote resources
  - A rough landscape of data ***management*** technologies
- Data manager, RI data management service developers/operators
  - How to ***deploy*** and operate data management services?
  - AAI infrastructure (authentication, authorization
- Data center operators, infrastructure resource providers
  - Virtual infrastructure provisioning
  - **Service level agreement (SLA)**
  - ***Autoscaling*** of infrastructure resources at runtime

1. E-Infrastructures and virtualization

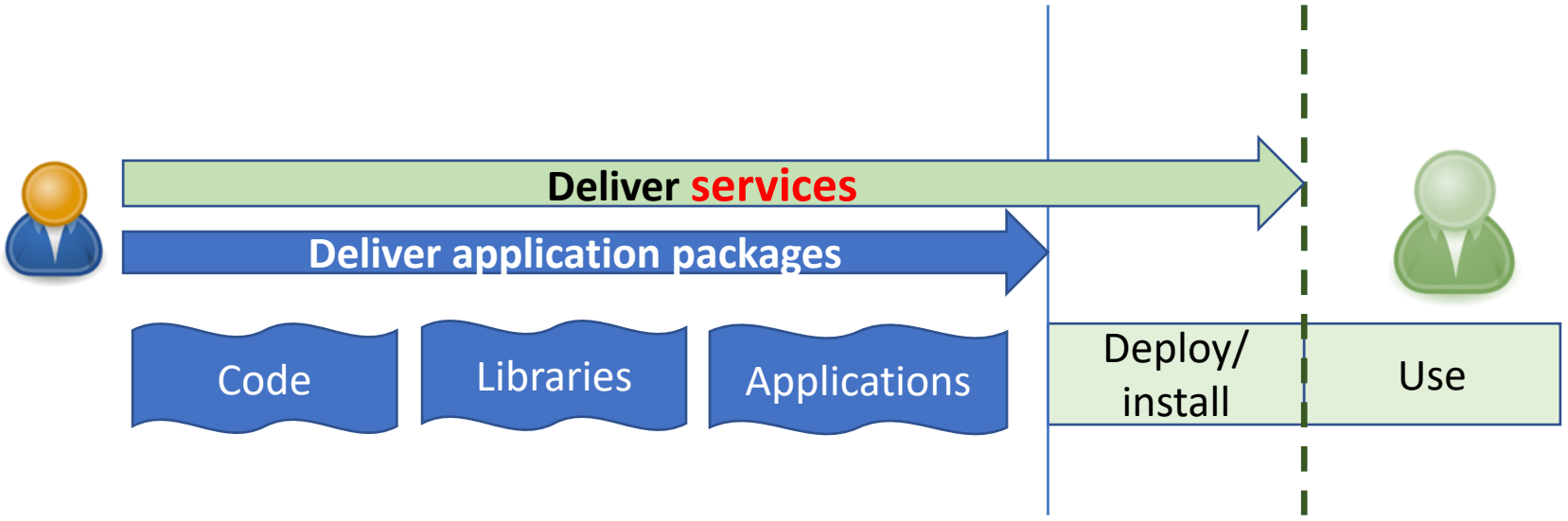
Scenario 1: data pipeline execution

- Heavy pipeline
  - Simulations (e.g., fluid dynamics)
  - Where to run them?



Scenario 2: service delivery

- Share software components
  - Options: software code, binary, self-contained environments …
  - How to package and share it?



Infrastructures are needed for data centric sciences

- High performance computers
  - Super computers
  - Clusters
- Virtualized resources
  - Infrastructure as a service (Virtual Machines)
  - Containers
  - Platform as a service
  - Software as a service
- Storage
  - Cloud storage, distributed file systems,
- Advanced network
  - Light paths
  - Software defined networking

Infrastructures are needed for data centric sciences

- High performance cor
  - Super computers
  - Clusters
- Virtualized resources
  - Infrastructure as a se
  - Containers
  - Platform as a service
  - Software as a service
- Storage
  - Cloud storage, distrib
- Advanced network
  - Light paths
  - Software defined net

e-Infrastructures address the needs of European researchers for digital se
networ ing, computing and da

aborative
a Infrastructure

EOSC-hub

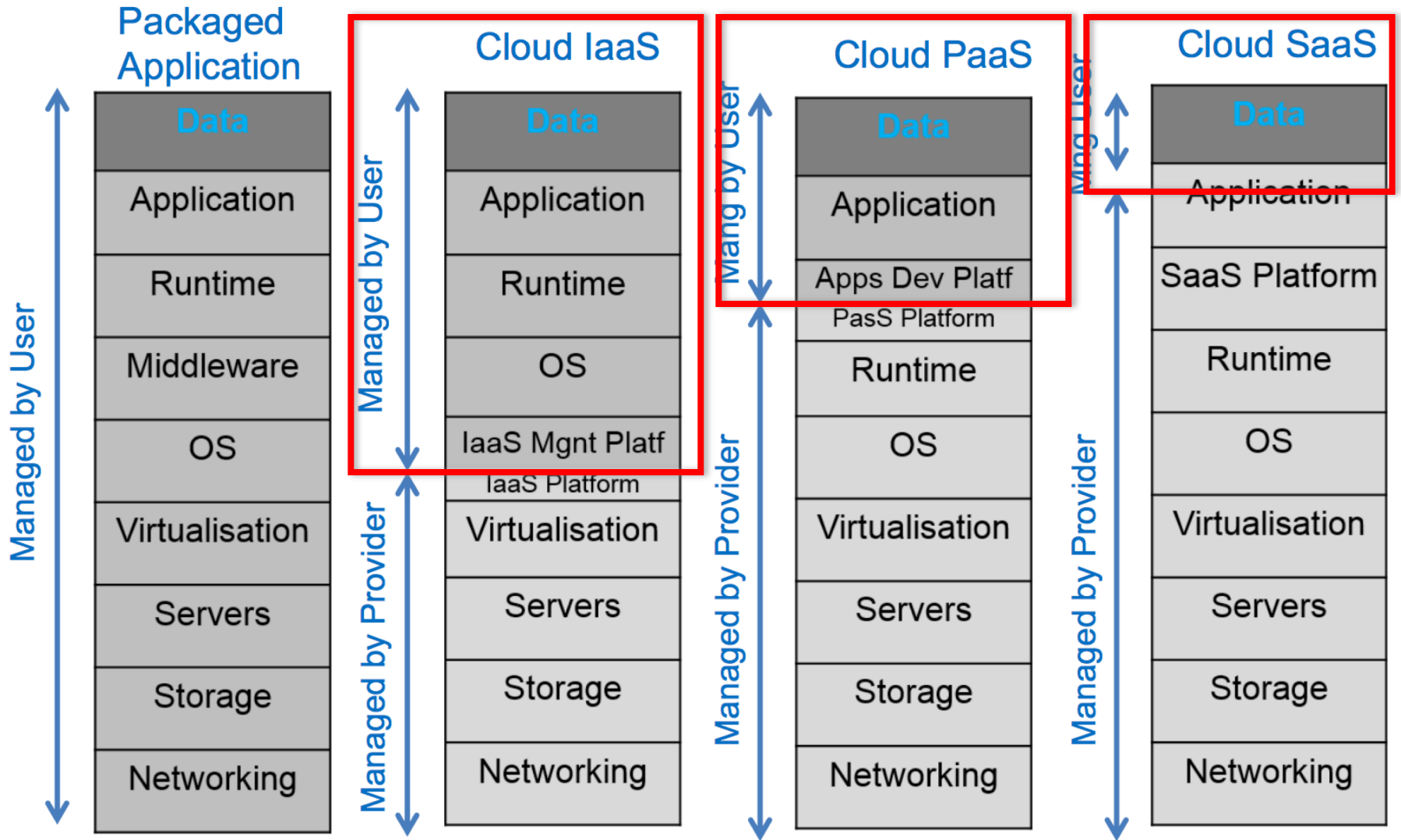
EUROPEAN OPEN SCIENCE CLOUD

Provider: Virtualize one computer as many computers

Data Center

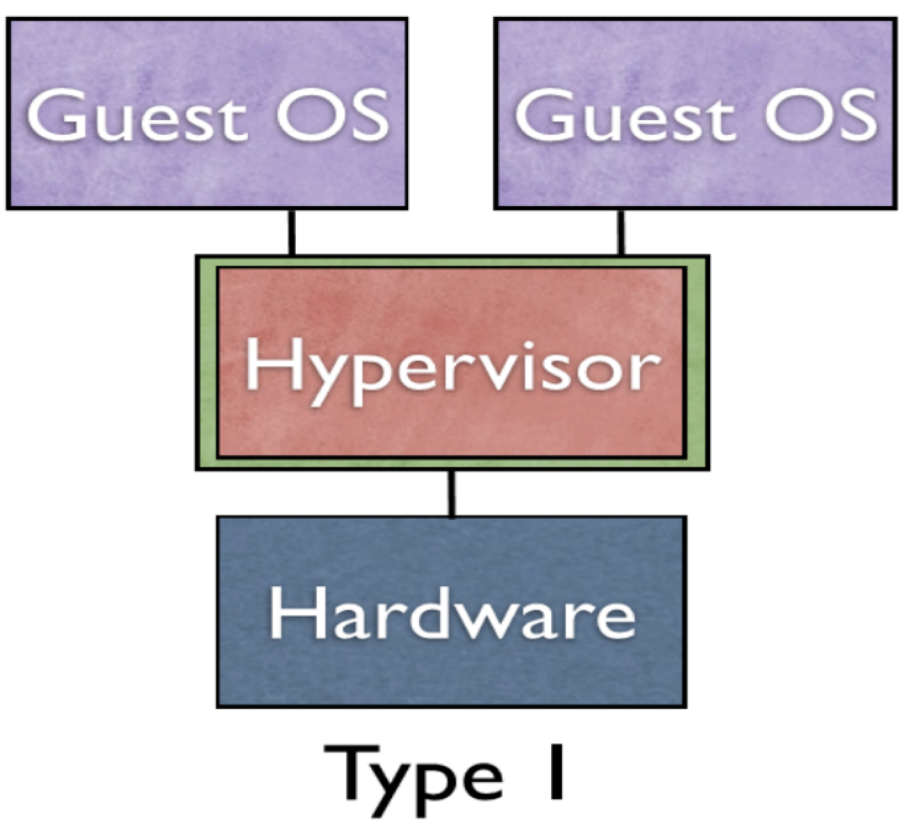
- Traditional way: many users share a server without isolation
- Cloud: many users share a server with isolated resources

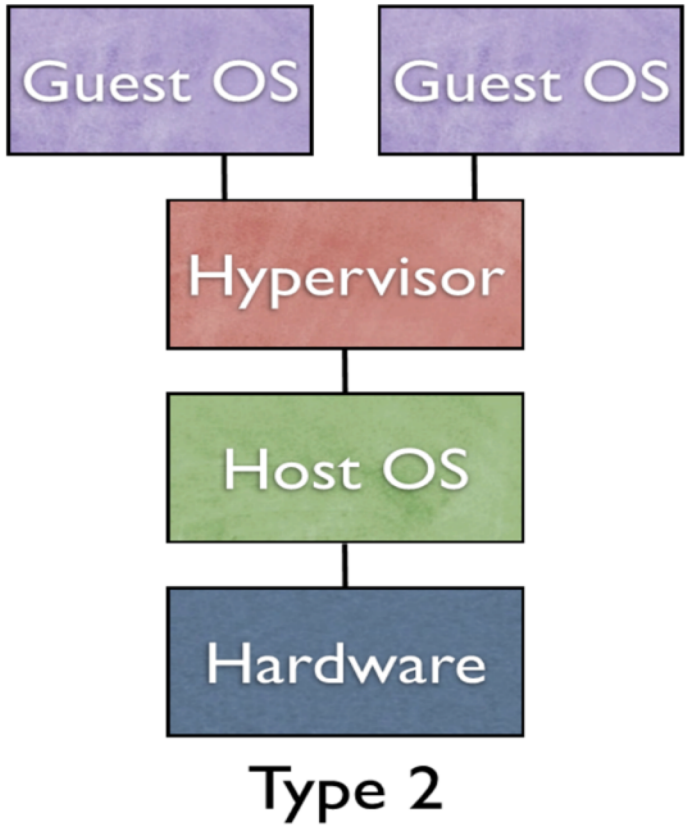
User: outsource management



How does a Cloud work?

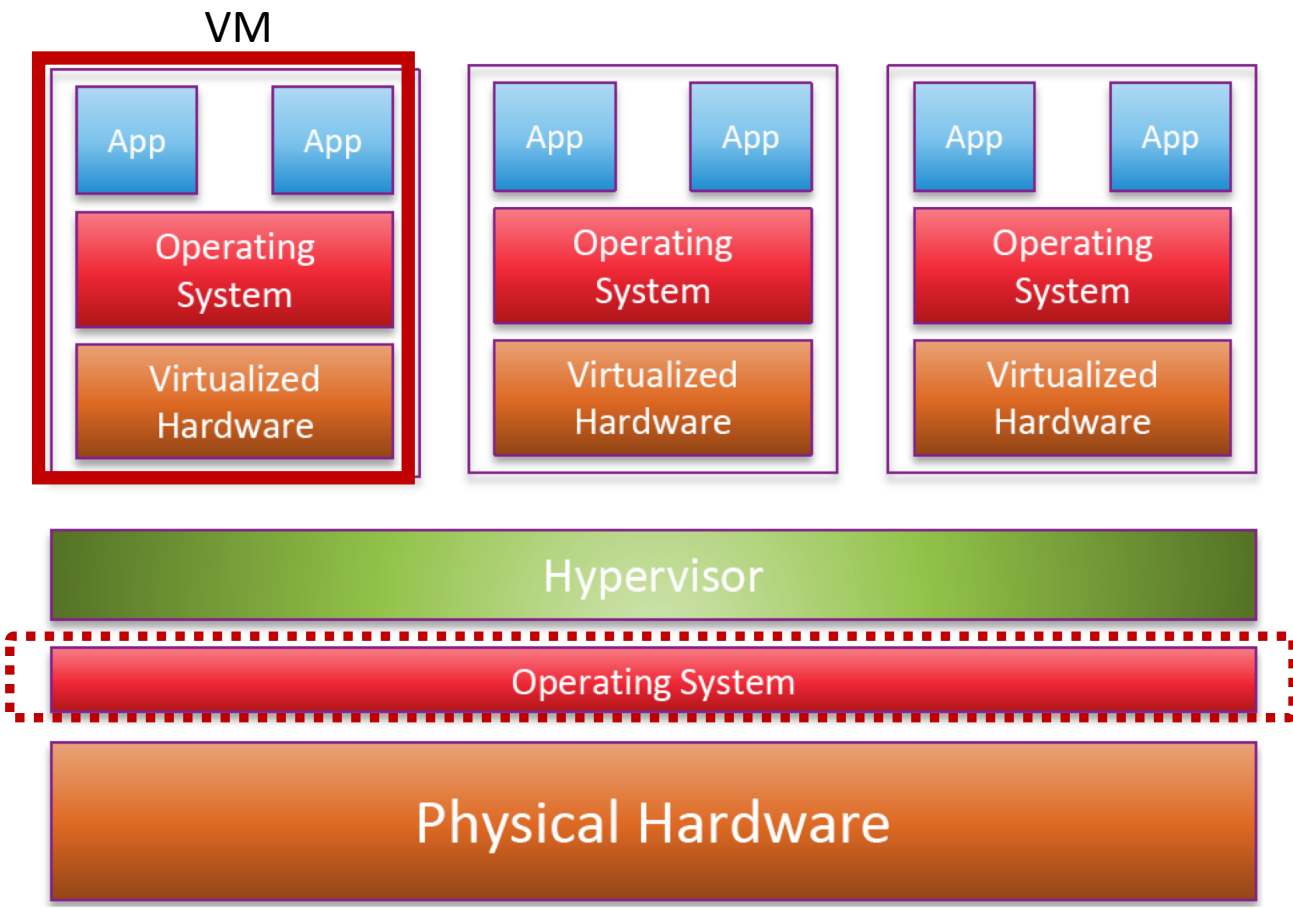
- Virtualization: one computer can be isolated as multiple *computers*





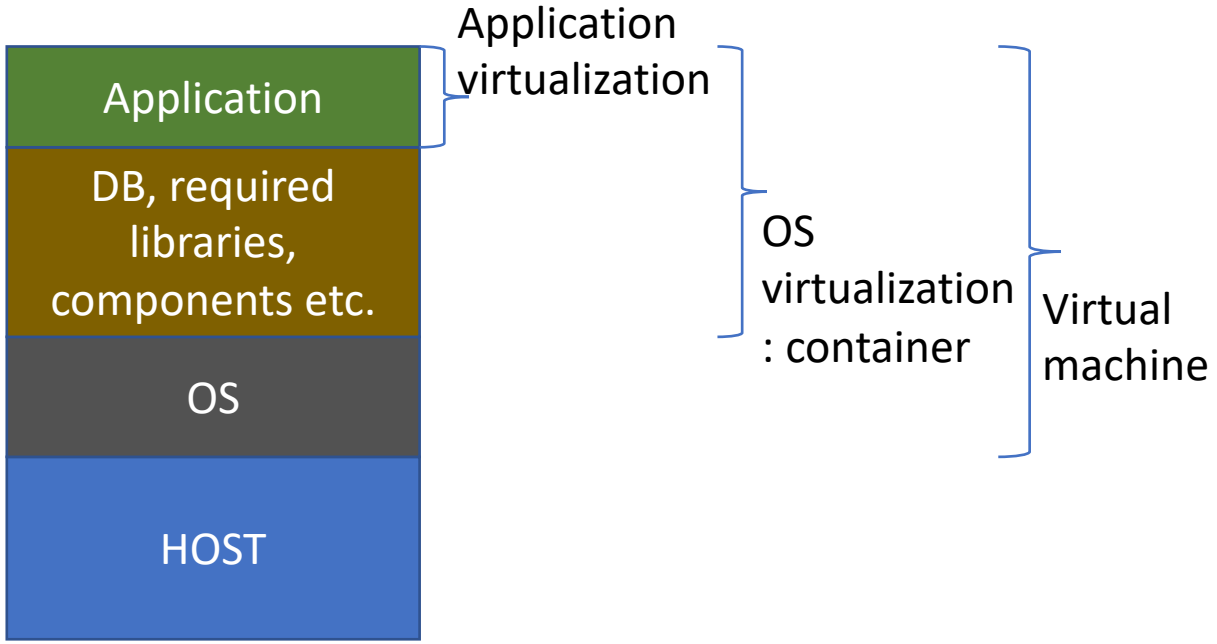
From virtualization point of view

- **Cloud IaaS**: VM (often contains a complete guest OS)



Application delivery and deployment

A **container** is a unit of **software** that packages up code and all its dependencies so the application runs quickly and reliably from **one computing** environment **to another**.



Applications in Cloud

Cloud service model

Complexity hidden

Programmability & Controllability

SaaS

(Software as a Service)

- Directly software to use
- Limited service interfaces

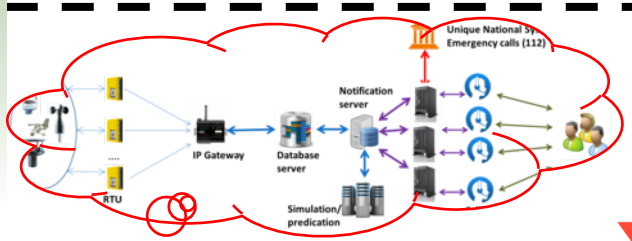
PaaS

(Platform as a Service)

- Limited by the platform

IaaS

(Infrastructure as a Service)



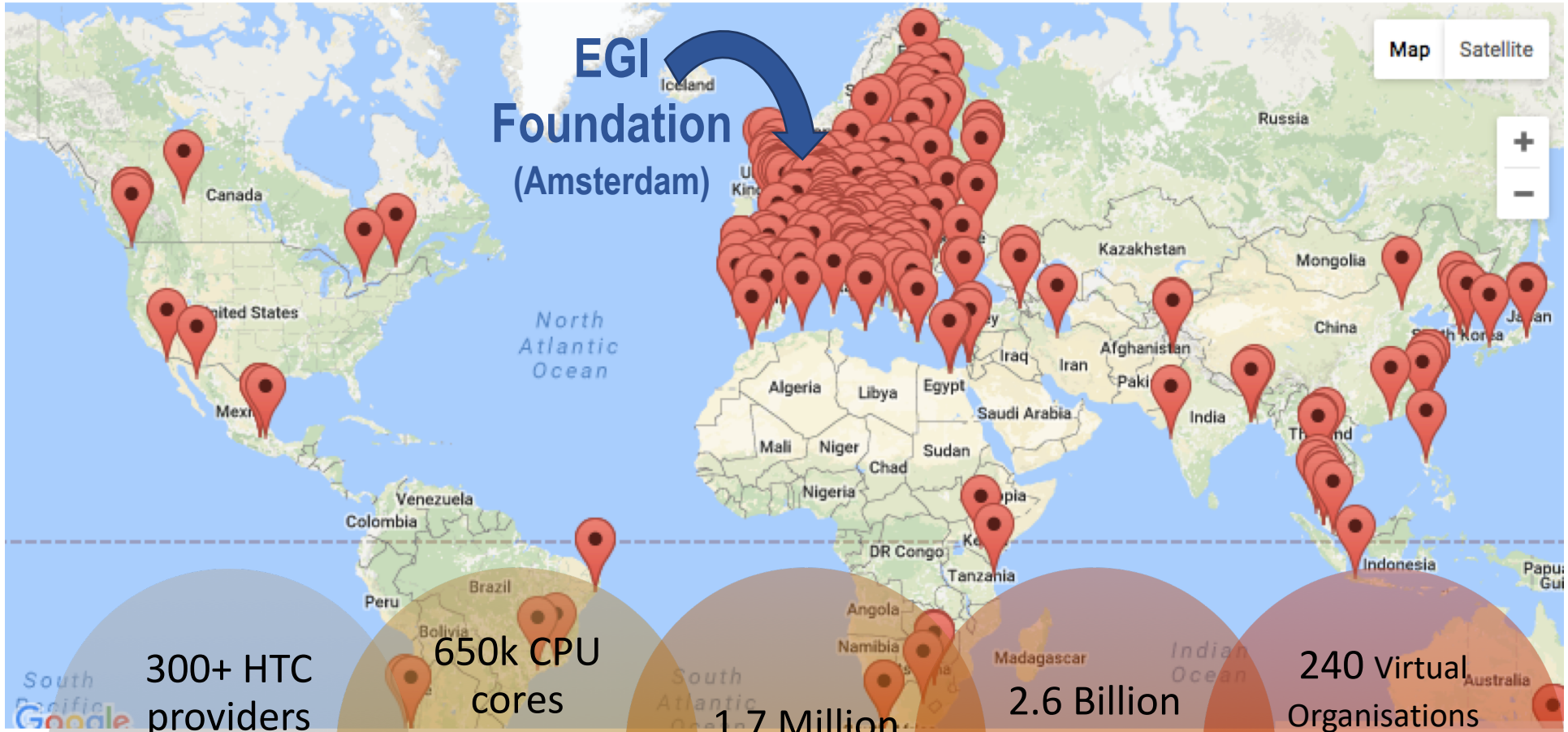
- Request a VM with a specified OS
- Able to run most applications
- The user has the controllability to configure the network

Cloud classification

- Private cloud
- Public cloud
- Hybrid cloud
- Community cloud

EGI Federation

Largest distributed compute e-Infrastructure of the world



300+ HTC providers

25 Cloud providers

650k CPU cores

285 PB online storage

1.7 Million jobs/day

2.6 Billion CPU hours/year

240 Virtual Organisations

>48 000 users

The EGI Service Catalogue

www.egi.eu/services

Compute

Cloud Compute

Run virtual machines on demand with complete control over computing resources

Cloud Container Compute BETA

Run Docker containers in a lightweight virtualised environment

High-Throughput Compute

Execute thousands of computational tasks to analyse large datasets

Storage and Data

Online Storage

Store, share and access your files and their metadata on a global scale

Archive Storage

Back-up your data for the long term and future use in a secure environment

Data Transfer

Transfer large sets of data from one place to another

Security

Check-in BETA

Login with your own credentials

Applications

Applications on Demand BETA

Use online applications for your data & compute intensive research

Training

FitSM Training

Learn how to manage IT services with a pragmatic and lightweight standard

ISO 27001 Training

Learn how to manage and secure information assets

Training Infrastructure

Dedicated computing and storage for training and education

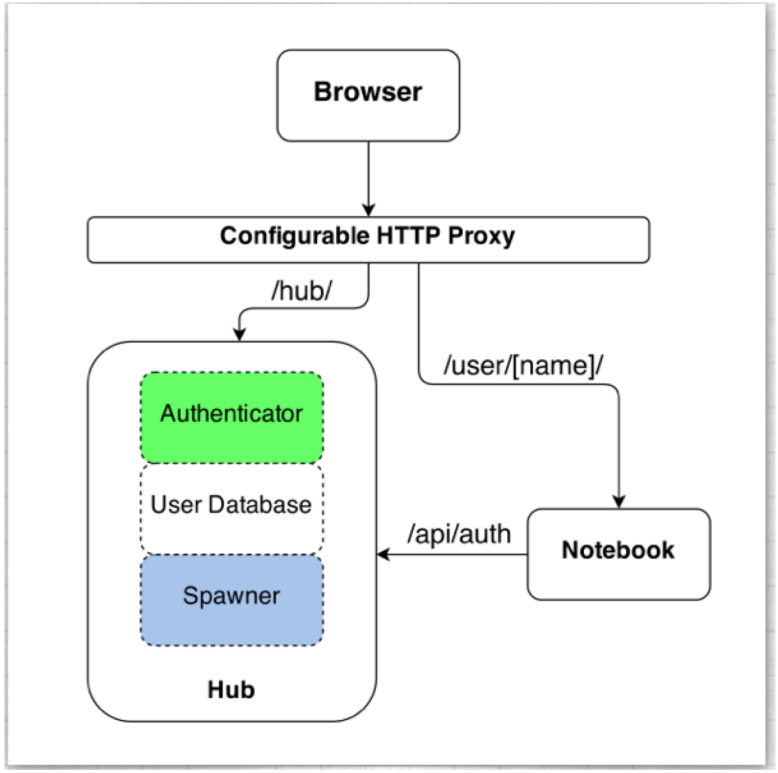
Jupyter notebook: https://jupyter.org/

- Install on your own laptop
  - Windows
  - MAC:
  - Linux
- Load notebook
- Create python helloword

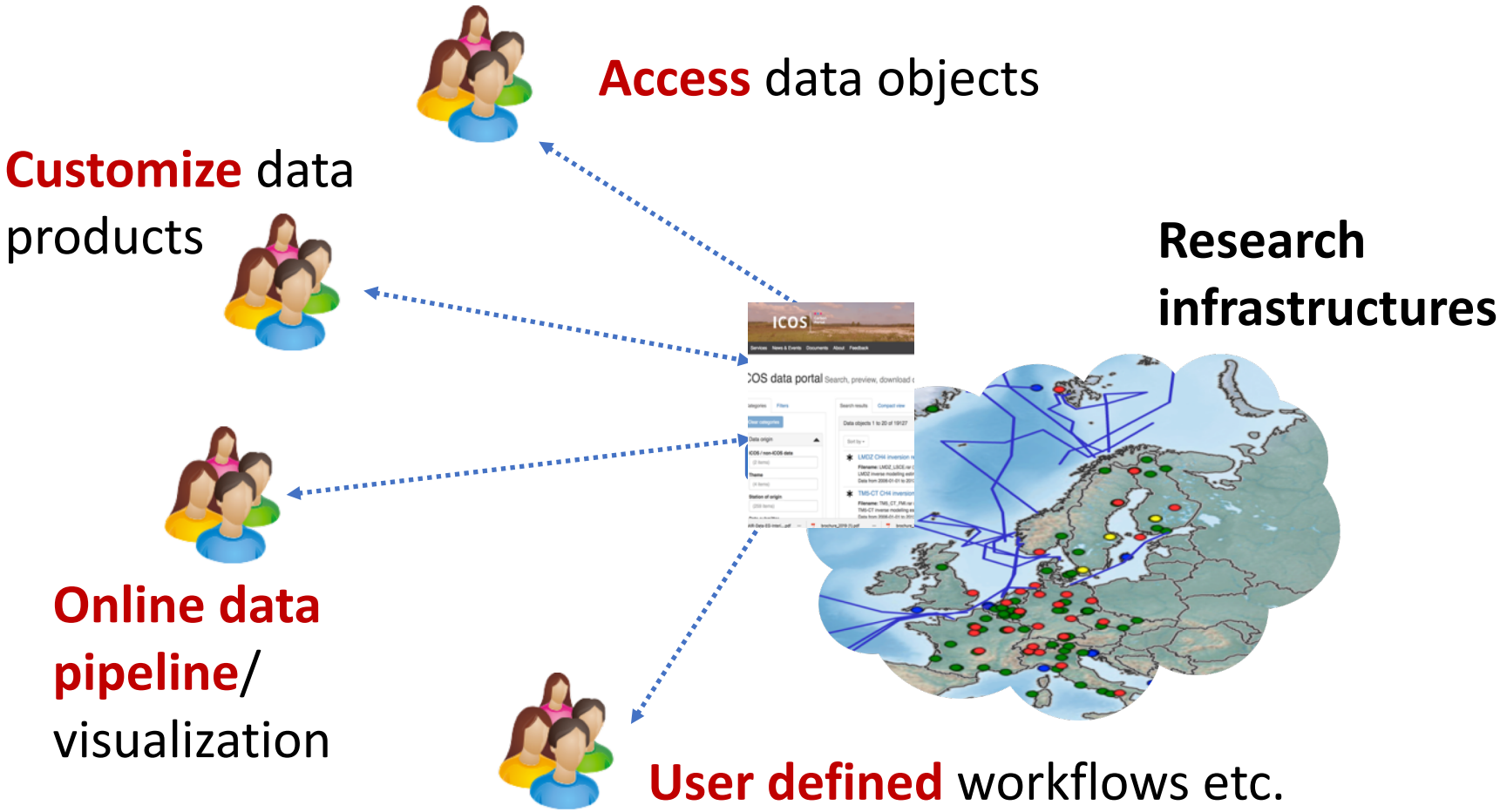
Jupyter and Cloud

- Jupyter is single user by design
- JupyterHub is a multi-user version of notebook designed for companies, classrooms and research labs

- JupyterHub capabilities:
  - Manages Authentication
  - Spawns single-users notebooks servers on-demand
  - Gives each user a complete Jupyter server



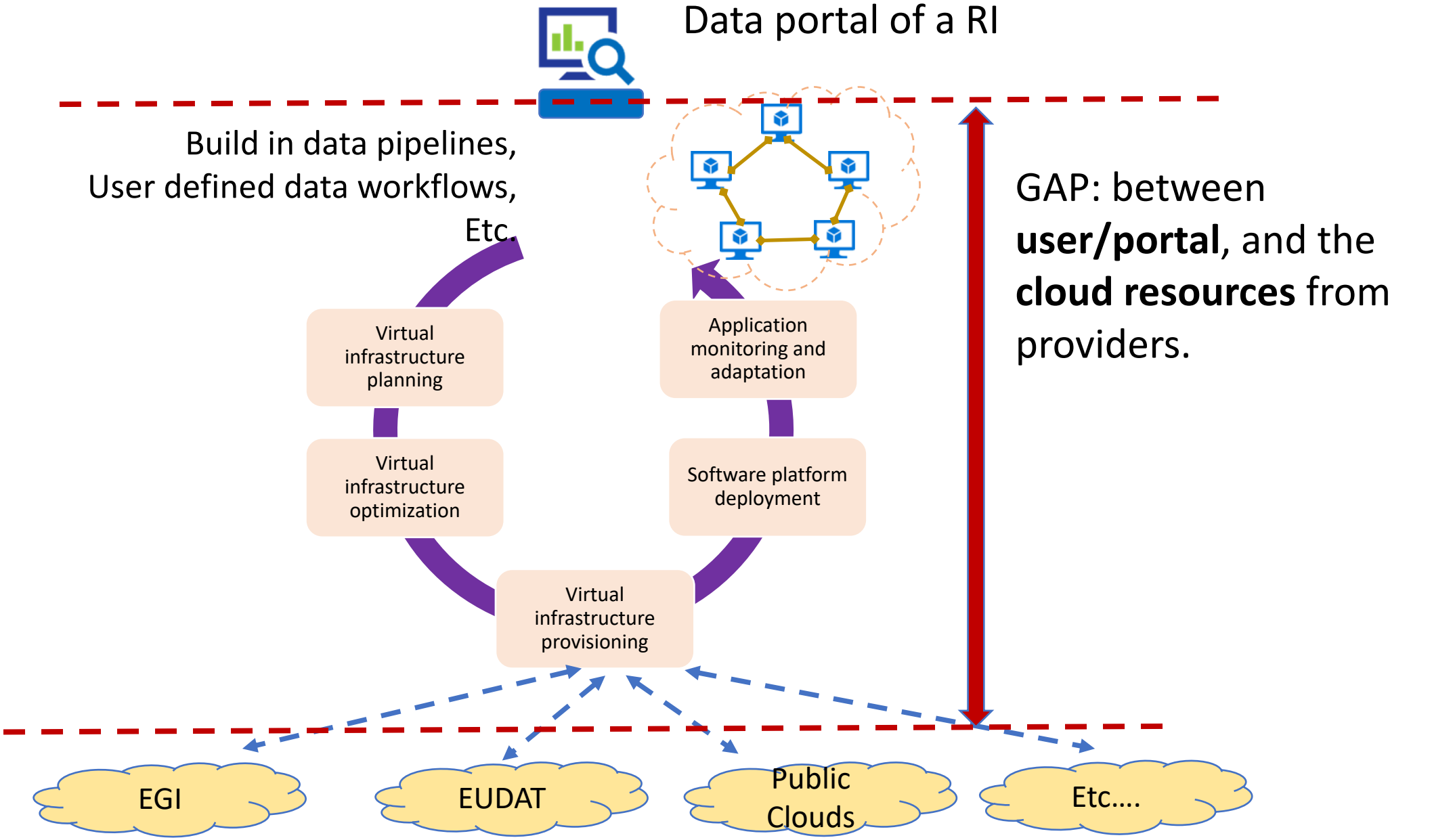
2. Using Cloud resources



Challenges of infrastructure as a service

- **Difficult to bridge the gap** between **complex data processing workflow** required by RI with the **cost effective virtualized resources** provided by different providers
  - **Data processing workflows** are distributed, with different data locality, real-time constraints,
  - **Virtualize resources** need to be i) networked, ii) provided by different data centers or providers, iii) integrated with dedicated infrastructures, and iv) further customized for different software stack
- **Requirement**:
  - Needs to be **customizable** for the characteristics of data, services, real-time constraints,
  - Needs to be **automated** to hide the diversity, of underlying providers, APIs. Ideally, provide application profile, return an accessible virtual infrastructure.
  - Needs to be **adaptable** to handle dynamic changes of the workflow loads (increasing sensors, data volume, parameter configurations etc.)

A solution to provide:

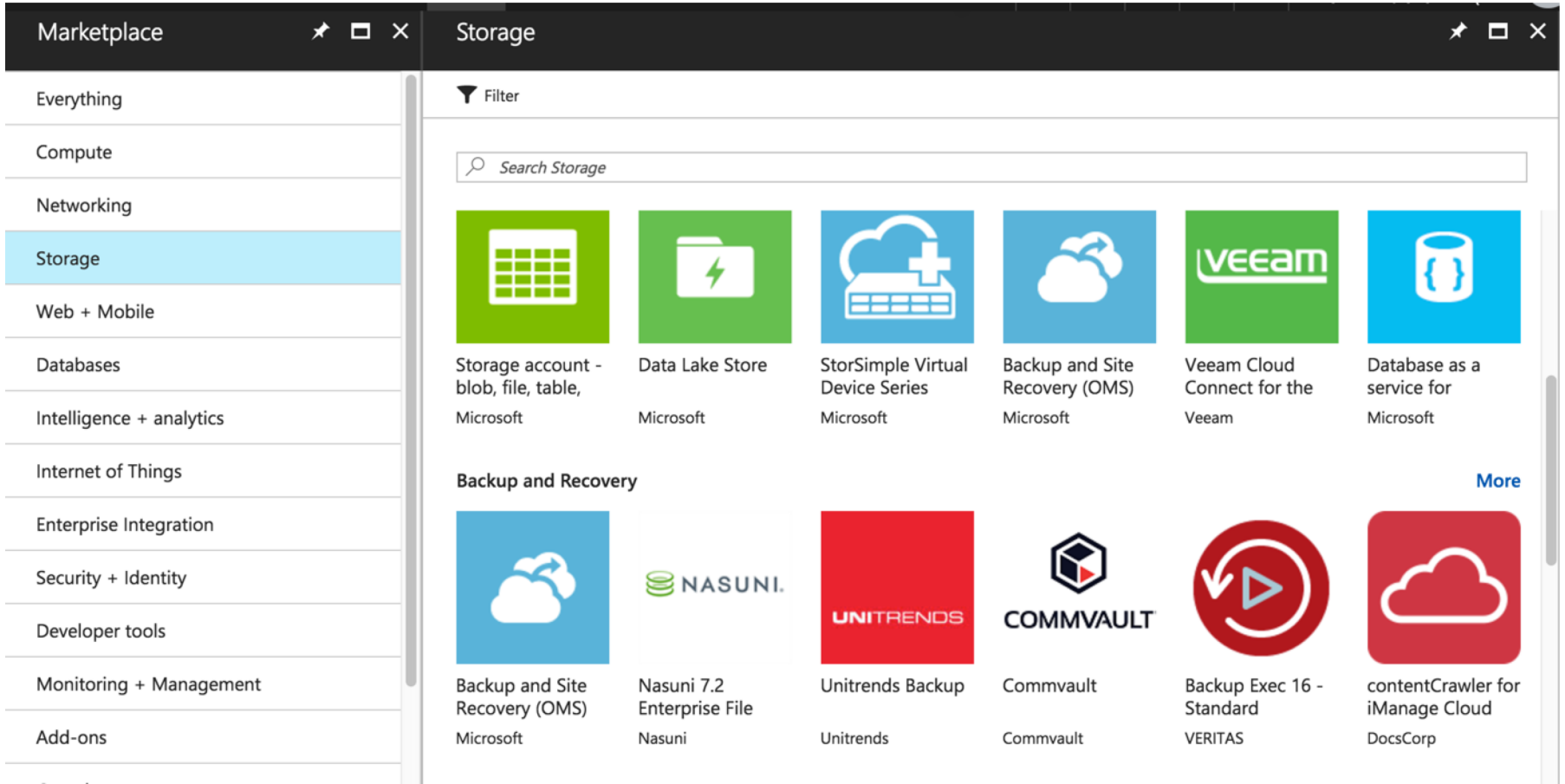


Virtual Infrastructure capacity planning

- Business requirements
- Workload analytics
- Application utilization patterns
- Data management policies
- What to put in Cloud
- Disaster recovery concerns
- Select appropriate cloud machines with price considerations
- Customize virtual infrastructure

Infrastructure as a Service

- How are they offered?



VM selection

- Cpu (>=4cores)
- Disk (>=200G)
- Memory (>=32G)

Size	vCPU	GiB	GiB	Core	/ hour	Credits	disks
Standard_B1s	1	1	4	10%	6	144	2
Standard_B1ms	1	2	4	20%	12	288	2
Standard_B2s	2	4	8	40%	24	576	4
Standard_B2ms	2	8	16	60%	36	864	4
				90%	54	1296	8
				135%	81	1944	16

Size	vCPU	Memory: GiB	Temp storage (SSD) GiB	Max data disks	Max temp storage throughput: IOPS / MBps	Max uncached disk throughput: IOPS / MBps
Standard_L4s	4	32	678	16	20,000 / 200	5,000 / 125
Standard_L8s	8	64	1,388	32	40,000 / 400	10,000 / 250
Standard_L16s	16	128	2,807	64	80,000 / 800	20,000 / 500
Standard_L32s [1]	32	256	5,630	64	160,000 / 1,600	40,000 / 1,000

Size	vCPU's	Memory: GiB	(SSD) GiB	data disks	(cache size in GiB)	bandwi (Mbps)
Standard_F2s_v2	2	4	16	4	4000 (32)	2 / 875
Standard_F4s_v2	4	8	32	8	8000 (64)	2 / 1,75
Standard_F8s_v2	8	16	64	16	16000 (128)	4 / 3,50
Standard_F16s_v2	16	32	128	32	32000 (256)	4 / 7,00
Standard_F32s_v2	32	64	256	32	64000 (512)	8 / 14,0
Standard_F64s_v2	64	128	512	32	128000 (1024)	8 / 28,0
Standard_F72s_v2[2,3]	72	144	576	32	144000 (1520)	8 / 30,0

Standard_D2_v2	2	7	100	6000 / 93 / 46	8	8x500
Standard_D3_v2	4	14	200	12000 / 187 / 93	16	16x500
Standard_D4_v2	8	28	400	24000 / 375 / 187	32	32x500
Standard_D5_v2	16	56	800	48000 / 750 / 375	64	64x500

Cloud pricing models

- **Fix pricing model**. The Provider determines the price based on service quality, cost of resource, provisioning, maintenance etc. in advance.
  - Static price
  - Long time
  - Simple profit estimation
- **Dynamic pricing model**. Dynamically changing based on market, available services, quality, etc.

Service level agreement (SLA)

- SLA is the contract between the provider and its users
  - For quality of the services

Size	vCPU's	Memory: GiB	(SSD) GiB	data disks	(cache size in GiB)	bandwi (Mbps)
Standard_F2s_v2	2	4	16	4	4000 (32)	2 / 875
Standard_F4s_v2	4	8	32	8	8000 (64)	2 / 1,75
Standard_F8s_v2	8	16	64	16	16000 (128)	4 / 3,50
Standard_F16s_v2	16	32	128	32	32000 (256)	4 / 7,00
Standard_F32s_v2	32	64	256	32	64000 (512)	8 / 14,0
Standard_F64s_v2	64	128	512	32	128000 (1024)	8 / 28,0
Standard_F72s_v2[2, 3]	72	144	576	32	144000 (1520)	8 / 30,0

What is Cloud SLA?

Cloud SLA (Service Level Agreement) is a **business** concept which defines the contractual and financial agreements between the Cloud **customer** and **provider**.

Cloud Service Customer

What is Cloud SLA?

Cloud SLA (Service Level Agreement) is a **business** concept which defines the contractual and financial agreements between the Cloud **customer** and **provider**.

Cloud Service

Provider is in a centralized and dominated position:

- **Less fair;**
- **Lack of violation proof;**
- **Manual enforcement.**

Clo

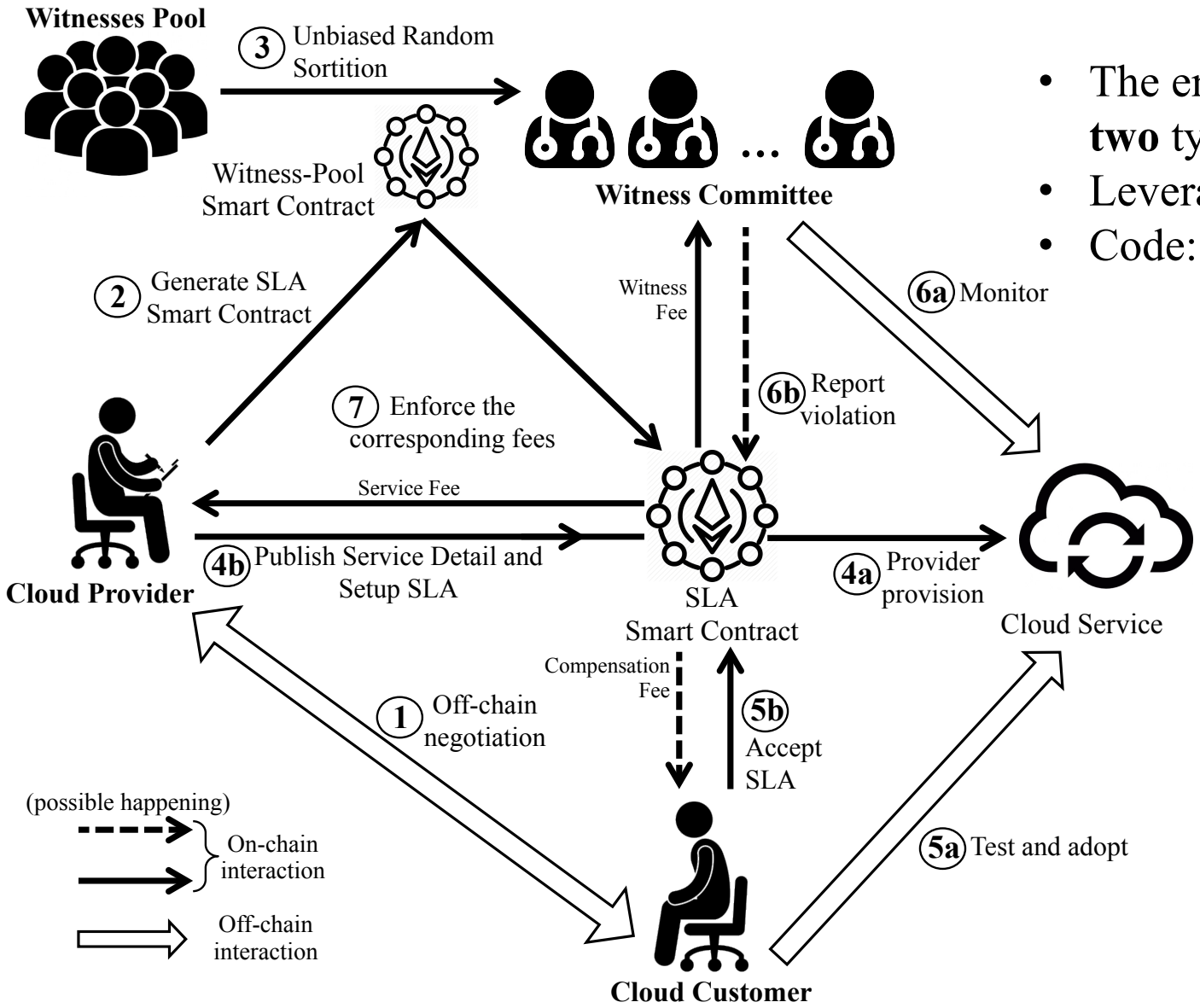
Ex

A

Th

- If the VM, ***X***, does not crash, ***C*** → ***P*** 1000 credits. (payment)
- If the VM, ***X***, crashes, ***C*** → ***P*** 500 credits. (compensation)

Implementation: Ethereum

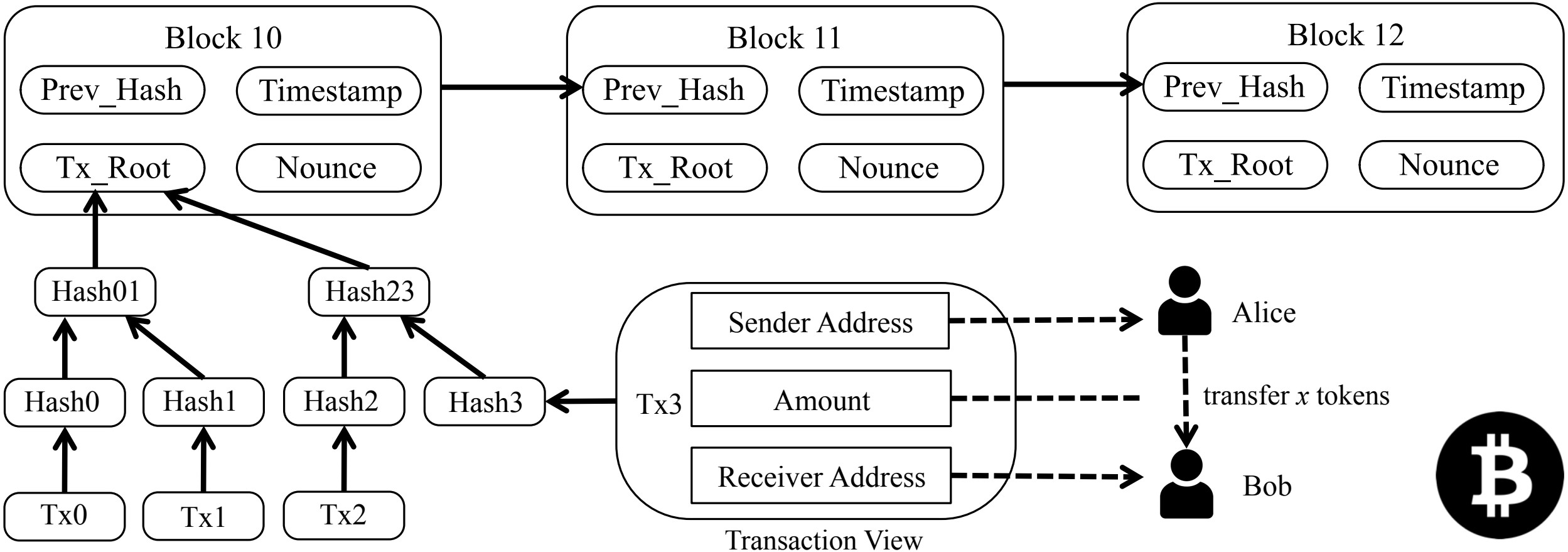


- The entire system is implemented based on the **two** types of smart contracts
- Leverage **Solidity** to program smart contracts
- Code: https://github.com/zh9314/SmartContract4SLA

[1]Huan Zhou, Xue Ouyang, Zhijie Ren, Jinshu Su, Cees d e Laat and **Zhiming Zhao** (2019) *A Blockchain based Witness Model for Trustworthy Cloud Service Level Agreement Enforcement,* IEEE International Conference on Computer Communications (INFCOM2019), Paris, France [10.1109/INFOCOM.2019.8737580]

Blockchain:
decentralized and **immutable** ledger

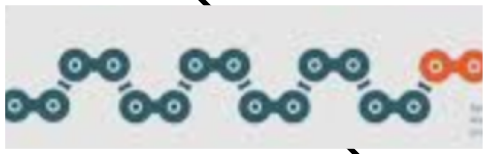
- Blockchain is a technique, which makes every participant having consensus on a decentralized ledger, e.g., through PoW (Proof of Work).
- Bitcoin is the first generation application of blockchain, from 2009.



Via friends

Via authority

Via history

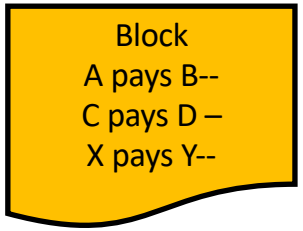


.....

Images are from internet

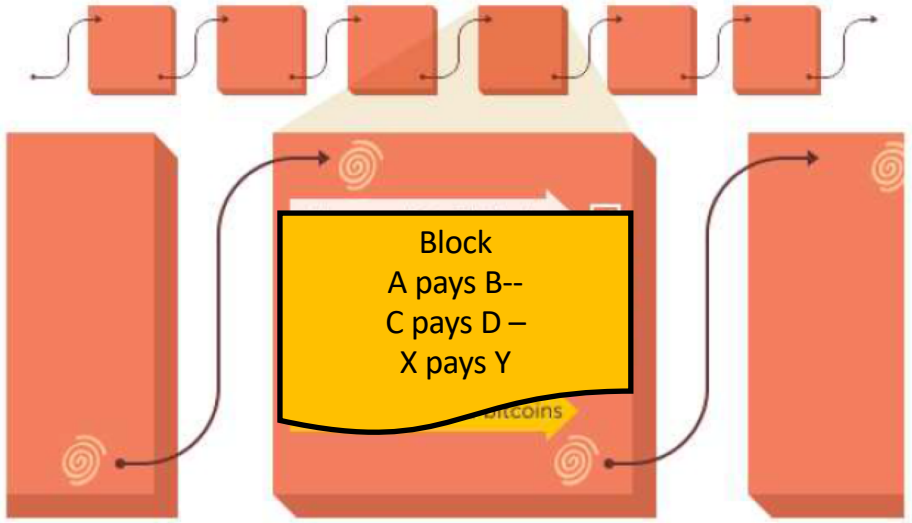
Block

A block contains *transactions*



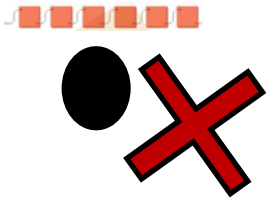
Chain of blocks

- **Blocks are chained**: a block contains the hash code of the previous block

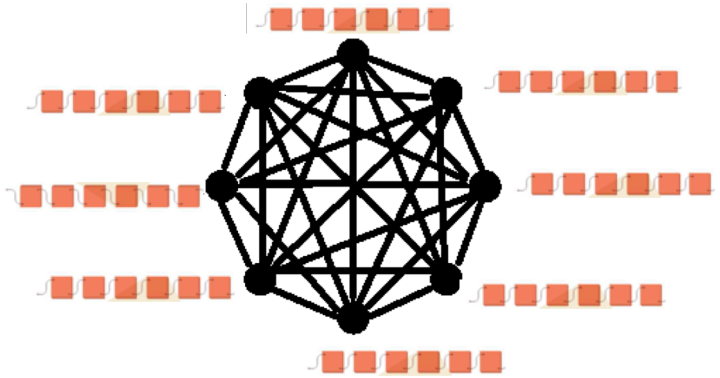


Blockchain

1. Distributed Ledger



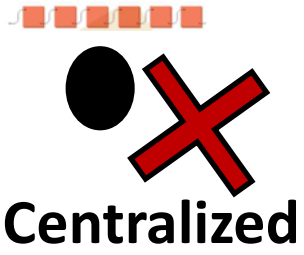
Centralized



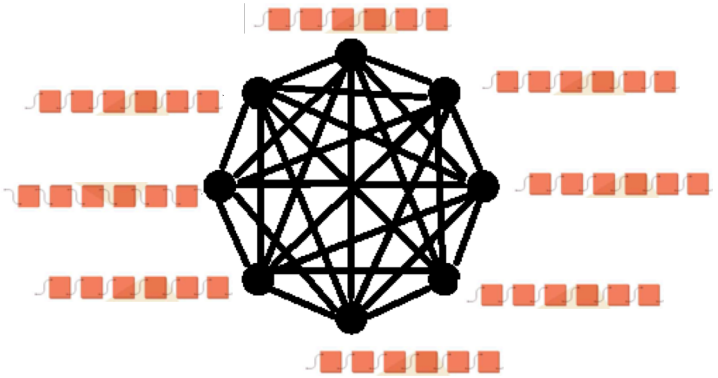
Distributed

Blockchain

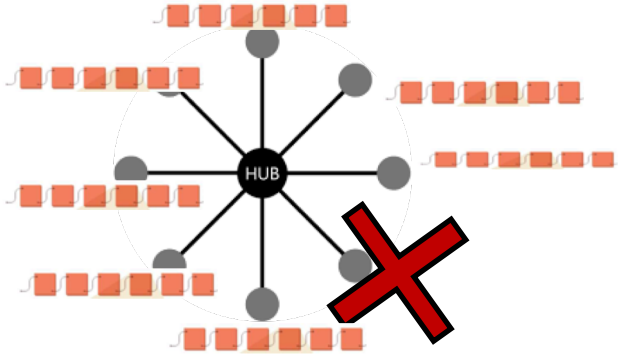
1. Distributed Ledger
2. **Decentralized**



Centralized

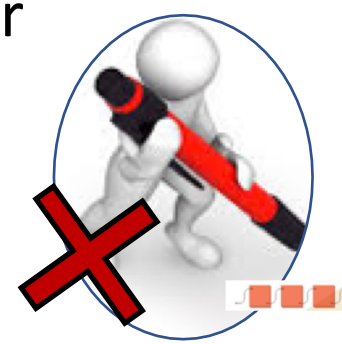


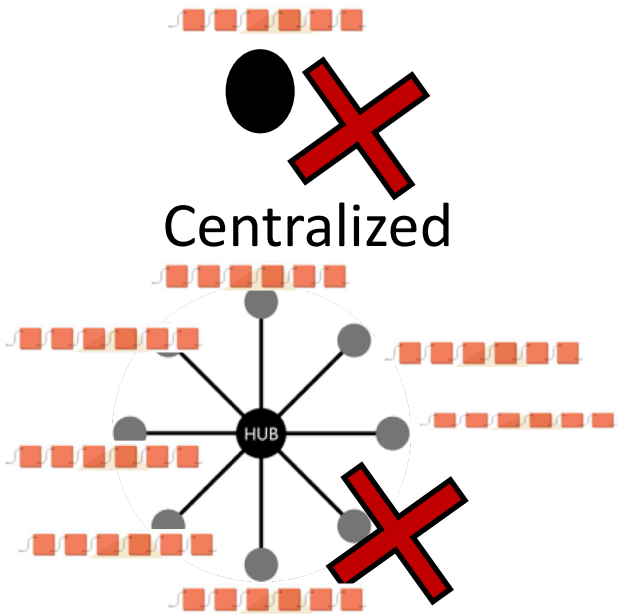
Decentralized



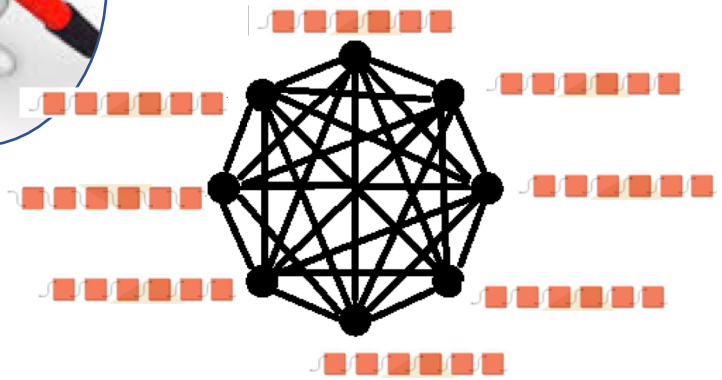
Blockchain

1. Distributed Ledger
2. Decentralized
3. **Immutable**





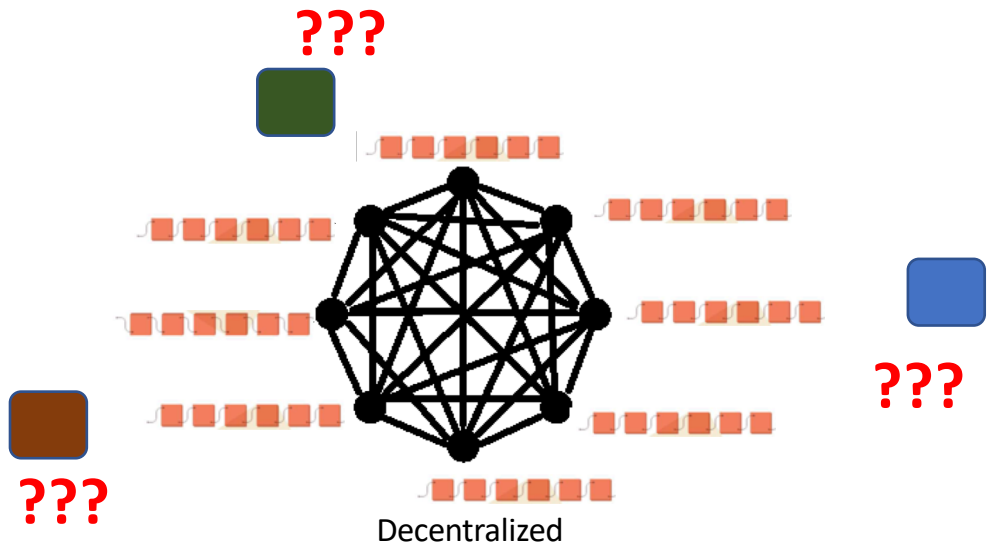
Centralized



Decentralized

Blockchain

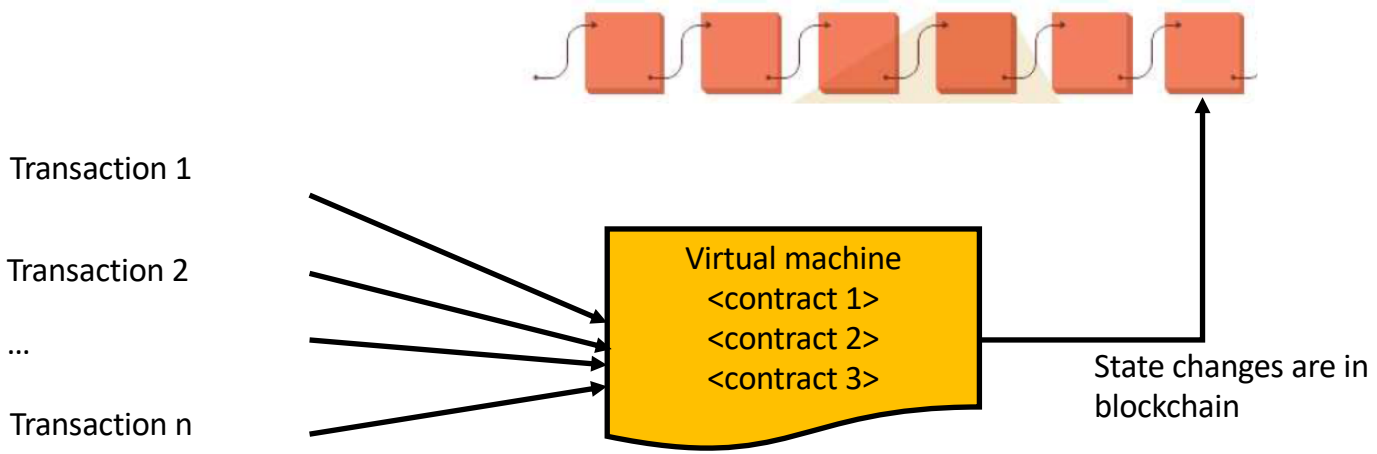
1. Distributed Ledger
2. Decentralized
3. Immutable
4. **Consensus**



Images are from internet

Smart contract: enhance transactions with *operations*

- Extend **transactions** in a block to **operations** (smart contracts)
  - create **smart contract** (code)
  - Invoke **smart contract**
- Node → Virtual machines



Some facts

Bitcoin
First block time: **2009-01-09**
Current Blockchain data size: **243.95 GB**
Max TPS (transaction per second): **7**
Blocks interval: around **10 mins**
Transaction to be confirmed: **6 blocks depth**

- **Ethereum**

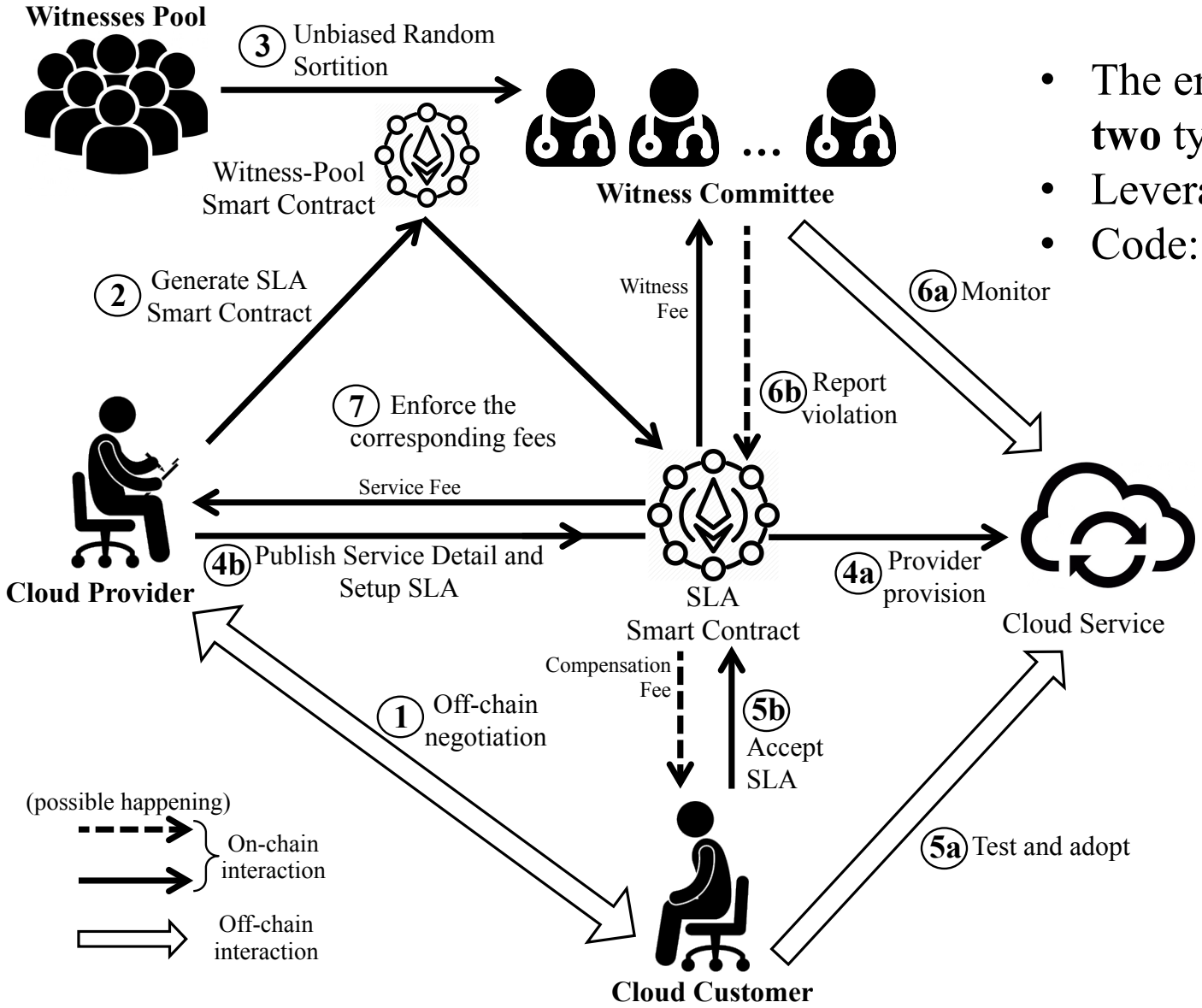
First block time: **2015-07-30**
Current Blockchain data size: **197.04 GB**
Max TPS: around **25**
Blocks interval: around **15s**
Transaction to be confirmed: **12 blocks depth**

Blockchain ecosystem

- **From the scope**
  - Public chain
  - Community/Consortium chain
  - Private chain
- **From Consensus protocol**
  - Proof of work
  - Proof of stake
  - BFT (Byzantine Fault Tolerance)
  - Proof of retrievability
- From the data structure
  - ….

Images are from internet

Implementation: Ethereum



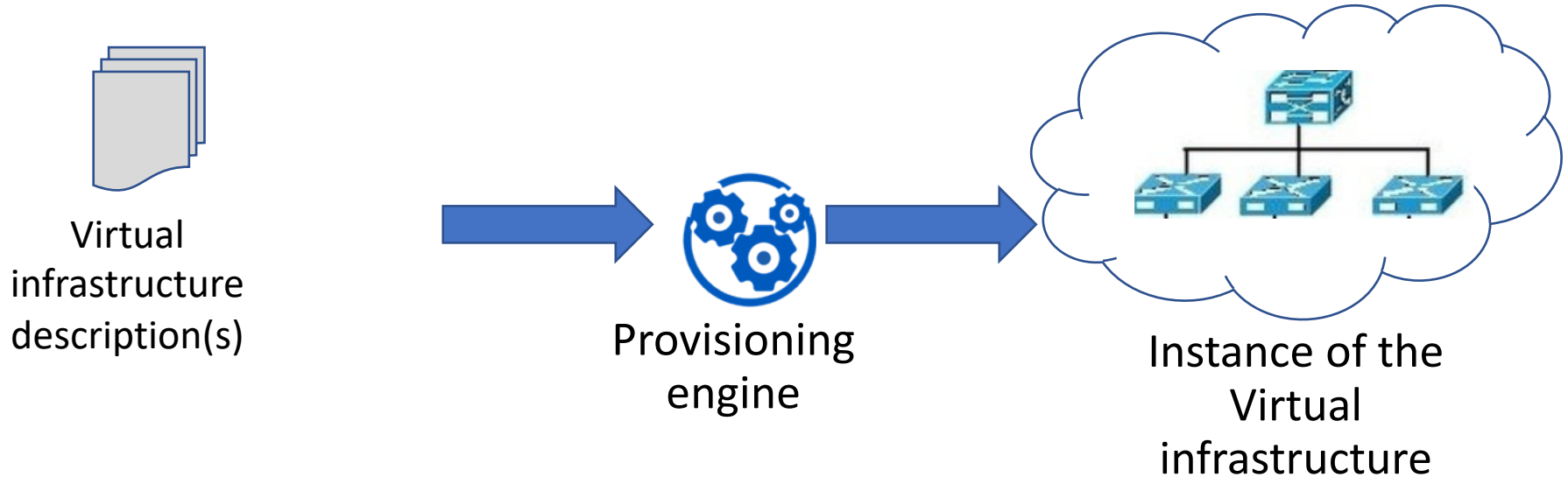
- The entire system is implemented based on the **two** types of smart contracts
- Leverage **Solidity** to program smart contracts
- Code: https://github.com/zh9314/SmartContract4SLA

[1]Huan Zhou, Xue Ouyang, Zhijie Ren, Jinshu Su, Cees d e Laat and **Zhiming Zhao** (2019) *A Blockchain based Witness Model for Trustworthy Cloud Service Level Agreement Enforcement,* IEEE International Conference on Computer Communications (INFCOM2019), Paris, France [10.1109/INFOCOM.2019.8737580]

Provisioning

Provision:

to create an instance of a virtual infrastructure based on its description via certain API(s) or engine(s).



Scenario

- Directly provision via provider's portal
  - GUI
- Provision via the API of a provider
  - All providers provide certain API/tools for remote provisioning
- Provision via engines/APIs, which might be provided by third party
  - E.g., Cloudify, Juju, JCloud, CHEF, PUPPET, **DRIP**.

Describe a virtual infrastructure

- Describe virtual machines, configurations, topologies etc.

```
components:
  - name: nodeA
    type: switch/compute
    nodetype: t2.medium
    OStype: "Ubuntu 14.04"
    domain: "ec2.us-east-1.amazonaws.com"
    script: /Users/zh9314/SWITCH_Provision/topology/t1/script/install.sh
    installation: /Users/zh9314/SWITCH_Provision/topology/t1/installation/Server
    public_address: TBD
    ethernet_port:
      - name: p2
        connection_name: c1.source
  - name: nodeB
    type: switch/compute
    nodetype: t2.medium
    OStype: "Ubuntu 14.04"
    domain: "ec2.us-east-1.amazonaws.com"
    script: /Users/zh9314/SWITCH_Provision/topology/t1/script/install.sh
    installation: /Users/zh9314/SWITCH_Provision/topology/t1/installation/Client
    public_address: TBD
    ethernet_port:
      - port_name: p2
        connection_name: c1.target
      - port_name: p3
        subnet_name: s1
        address: 192.168.10.10
subnets:
  - name: s1
    subnet: 192.168.10.0
    netmask: 255.255.255.0
connections:
  - name: c1
    source:
      component_name: nodeA
      port_name: p1
      netmask: 255.255.255.0
      address: 192.168.3.11
    target:
      component_name: nodeB
```

Other examples

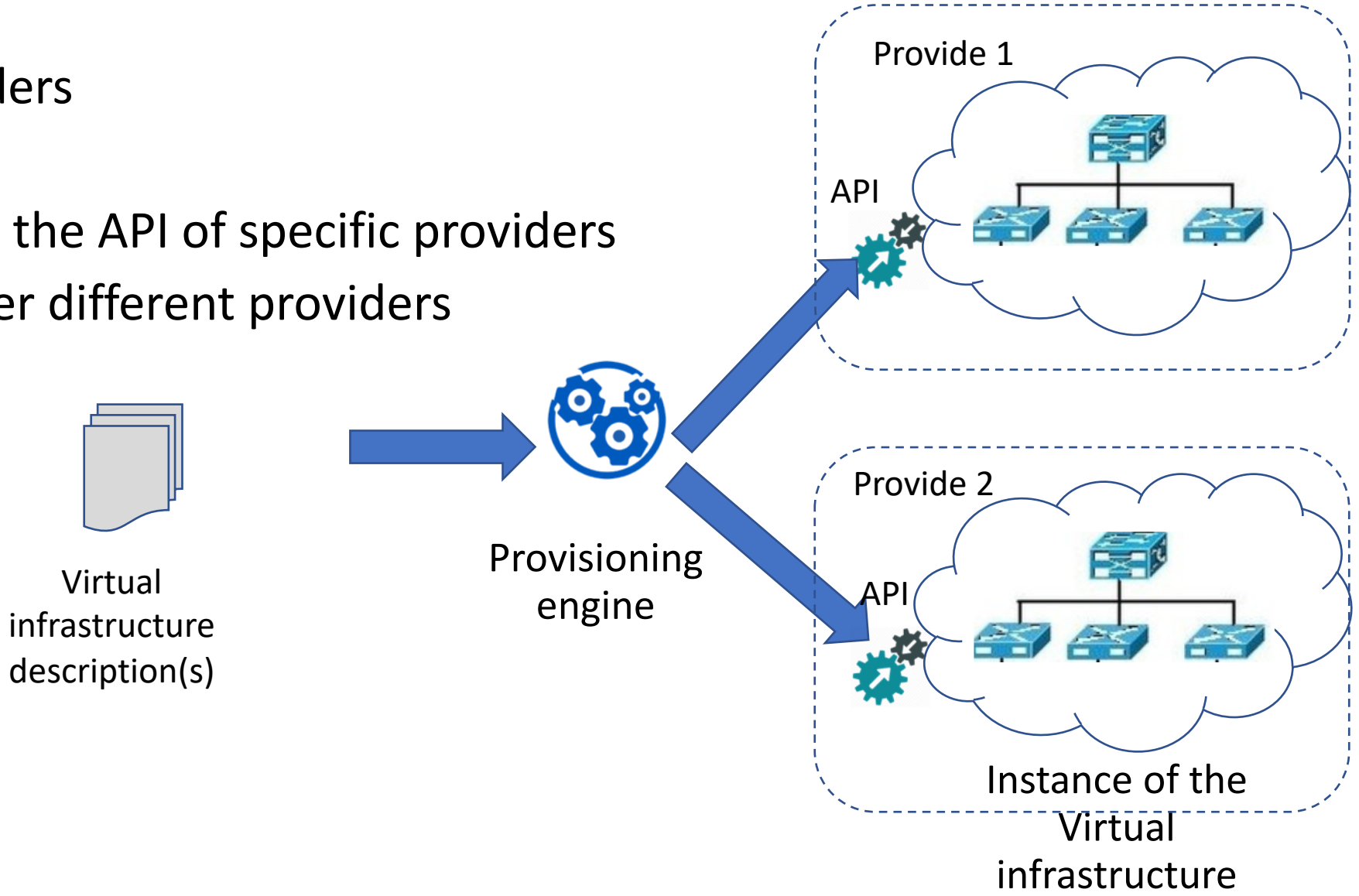
Further reading:

ACM Computing Surveys (CSUR) Surveys Homepage archive, Volume 51 Issue 1, April 2018 Article No. 22

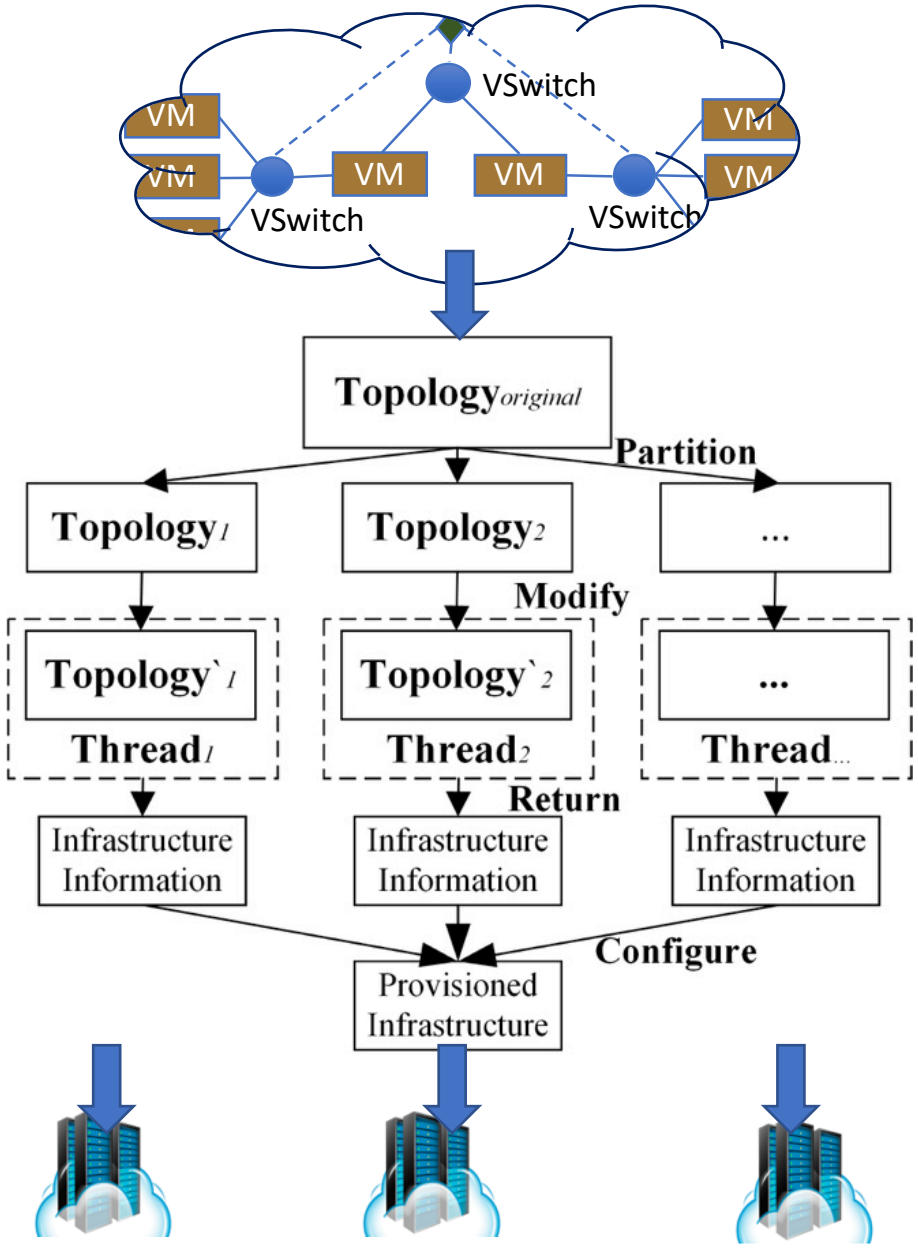
CML	Pragmatics	Target
Blueprint	Cloud service composition and description of deployment configurations	XaaS
Caglar *et al.*	Cloud service simulation and description of deployment plan configurations	IaaS
CAML	Cloud application architecture description and refinement of deployment configurations towards target cloud environment	XaaS
clADL	Architecture description of interactive cloud services and generation of implementations for the cyber-physical systems domain	XaaS
CloudDSL	Description of deployment configurations	XaaS
CloudMIG	Application migration to the cloud with emphasis on optimal deployment configurations and their conformance with target cloud environments	PaaS I
CloudML-SINTEF	Automated provisioning of multi-cloud applications and re-configuration of provisioned cloud services at run-time	XaaS
CloudML-UFPE	Description of cloud services	IaaS
CloudNaaS	Description of deployment configurations with emphasis on network aspects	IaaS
GENTL	Description of deployment configurations with emphasis on cost-efficient application provisioning	XaaS
Holmes	Description of deployment configurations and their automated provisioning	XaaS
MOCCA	Optimal (re)arrangement of (existing) deployment configurations for application provisioning to multiple target cloud environments	XaaS
MULTI-CLAPP	Application code generation for target cloud environments from component configurations	XaaS
Nhan *et al.*	Feature model based software stack (re-)configuration and their automated provisioning	IaaS
PDS	Deployment plan generation from described deployment configurations	IaaS
RESERVOIR-ML	Description of deployment configurations with emphasis on application-triggered elasticity rules for infrastructure-related cloud services	IaaS
StratusML	Generation of executable deployment descriptor and run-time adaptation rule from described deployment configurations	XaaS
TOSCA	Description of portable composite cloud applications for their automated provisioning and life-cycle management	XaaS
VAMP	Automated provisioning of distributed cloud applications with emphasis on support for establishing communication between components	IaaS

Provision using third party tools

- Select providers
- SLA
- Provision via the API of specific providers
- Provision over different providers



DRIP solution



Zhou, H., Hu Y., Wang, J., Martin, P., de Laat, C. and **Zhao, Z.**, (2016) *Fast and Dynamic Resource Provisioning for Quality Critical Cloud Applications*, IEEE International Symposium On Real-time Computing (ISORC) 2016, York UK [doi:10.1109/ISORC.2016.22]

3. Controlling applications in Cloud

- When a data catalogue is accessed by many visitors?
- When you simulation is too heavy to run on your own computer?

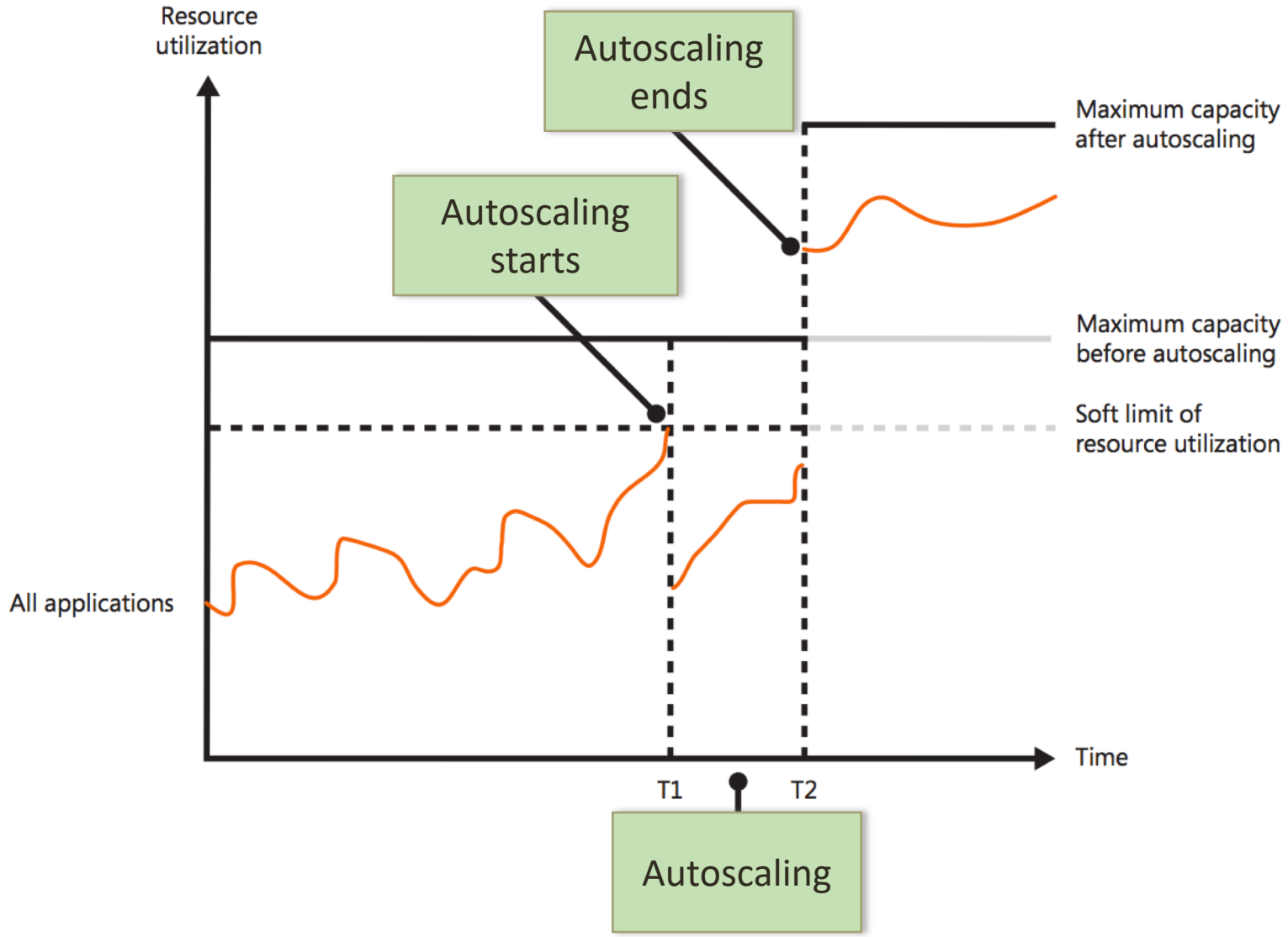
Auto-scaling

- How to dynamically allocate resources required by an application?
- Autoscaling
  - **Vertical scaling**- ***scaling up***: redeploy solutions using different hardware (adding more power, cpu, memory etc.). A disruptive process, temporal unavailable while redeploying.
  - **Horizon scaling**- ***scaling out***: requires deploying the system on additional resources. Continue without interruption.

How does elastic computing work?

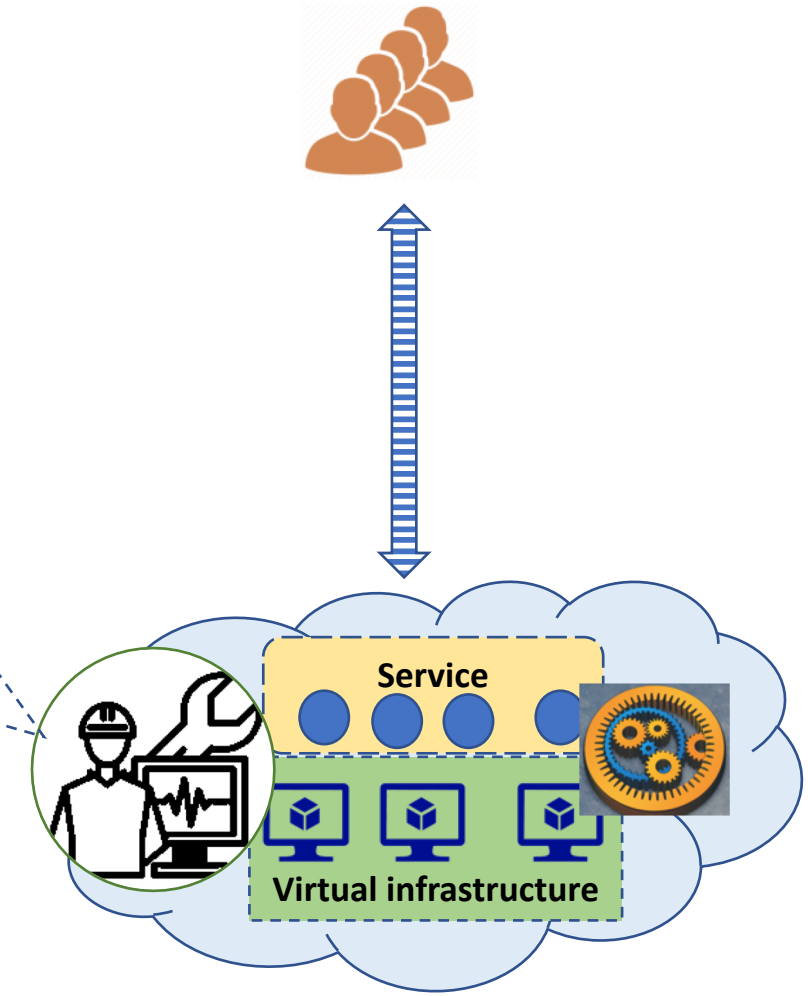
- Virtual machine scaling
- Application scaling

How to implement auto scaling?



How to maintain the performance?

1. What should the *"operator"* check?

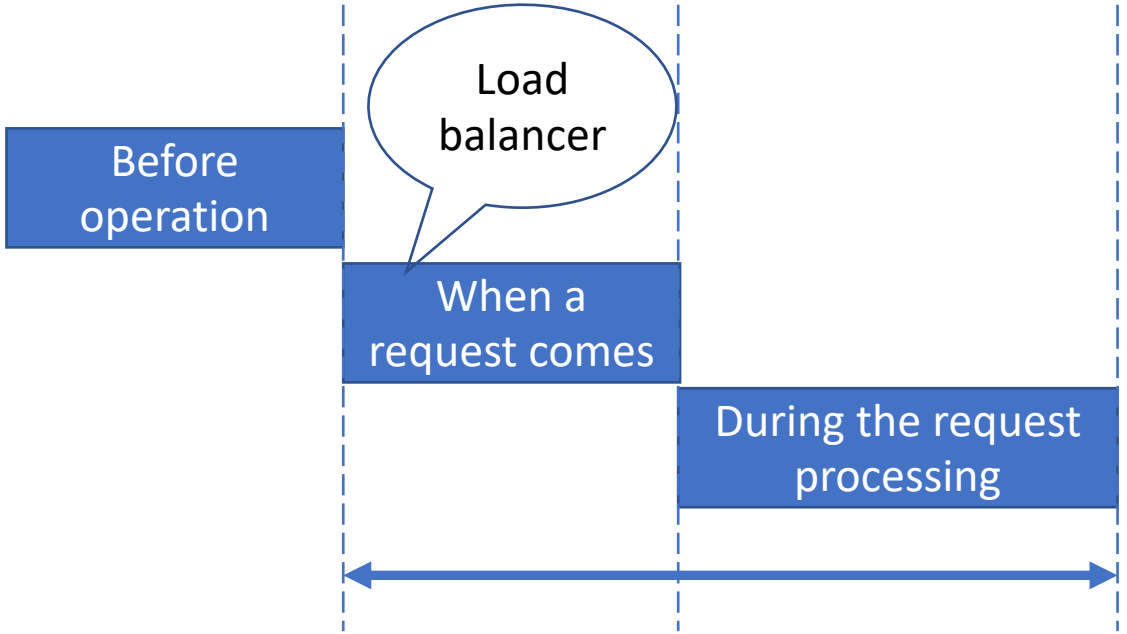


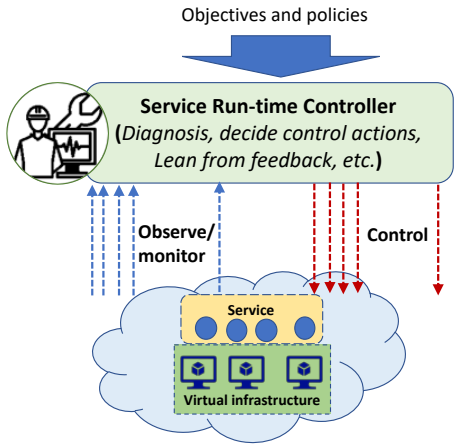
Different control options

- Load balancer
- Scaling
- Other adaptation

Load balancer

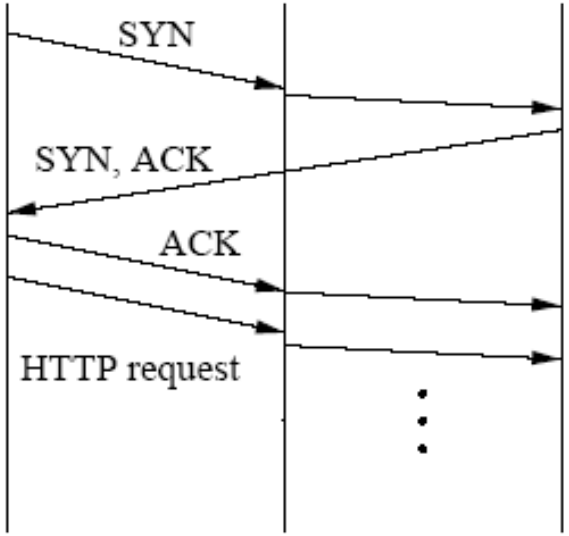
- Observe/Monitor
  - the input request
- Diagnosis
  - Monitor the healthy
- Decision: Balance the load
  - Static
  - Dynamic
- Control activities
  - Dispatching traffic
- Learn from feedback



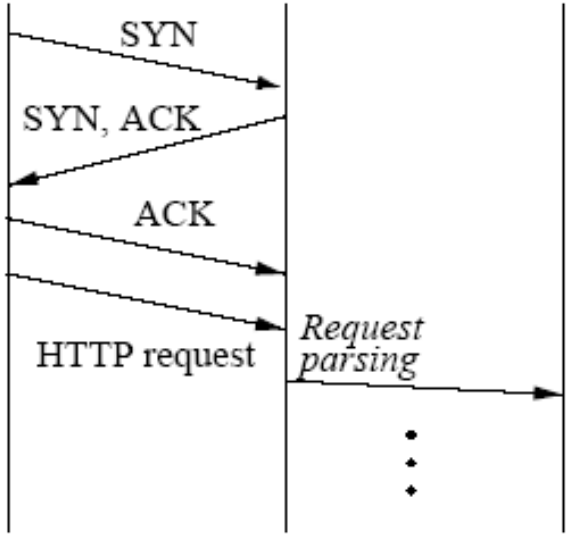


Handling the incoming requests

- **Content blind**– Determines the target server when a network packet is received (e.g., TCP SYN), the server selection policy is not based on http contents at the application level. Also called **layer-4** load balancing.
- **Content aware**– examines the request at the application level and then selects a server. It supports sophisticated dispatching policies, but large latency for moving to application level – Also called ***Layer7*** *or* ***Layer 5*** **in TCP/IP** protocol load balancing.

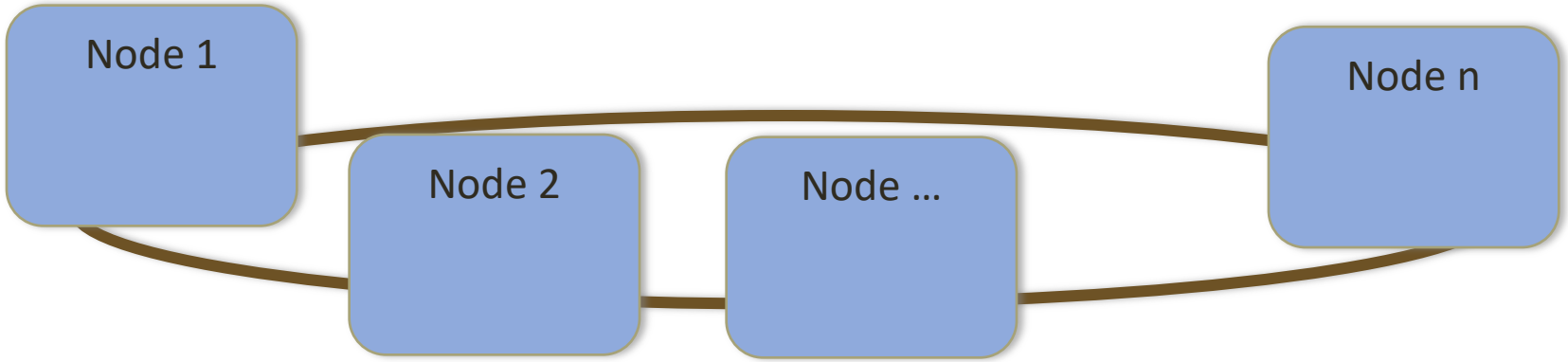


Content blind



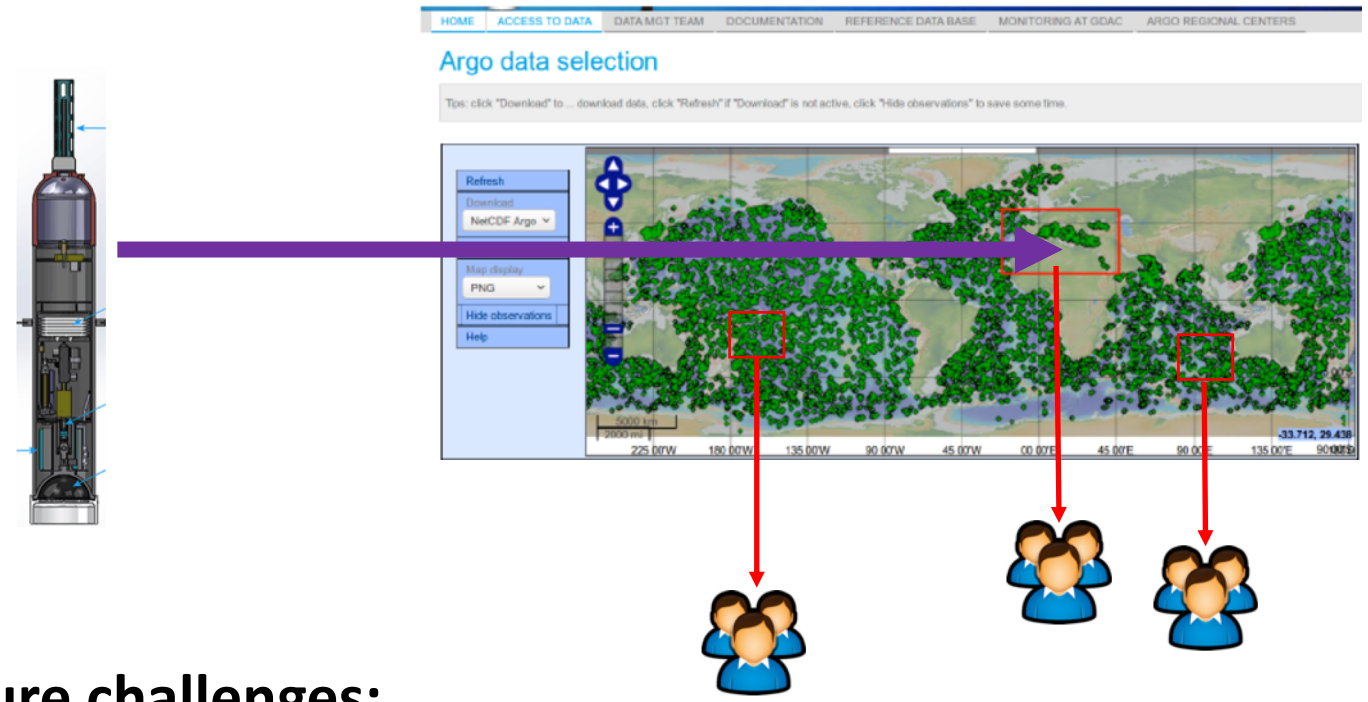
Content aware

How does it work?



- Cluster is a collection of nodes
- A load balancer schedules the service/components

Example: subscribing data from a RI

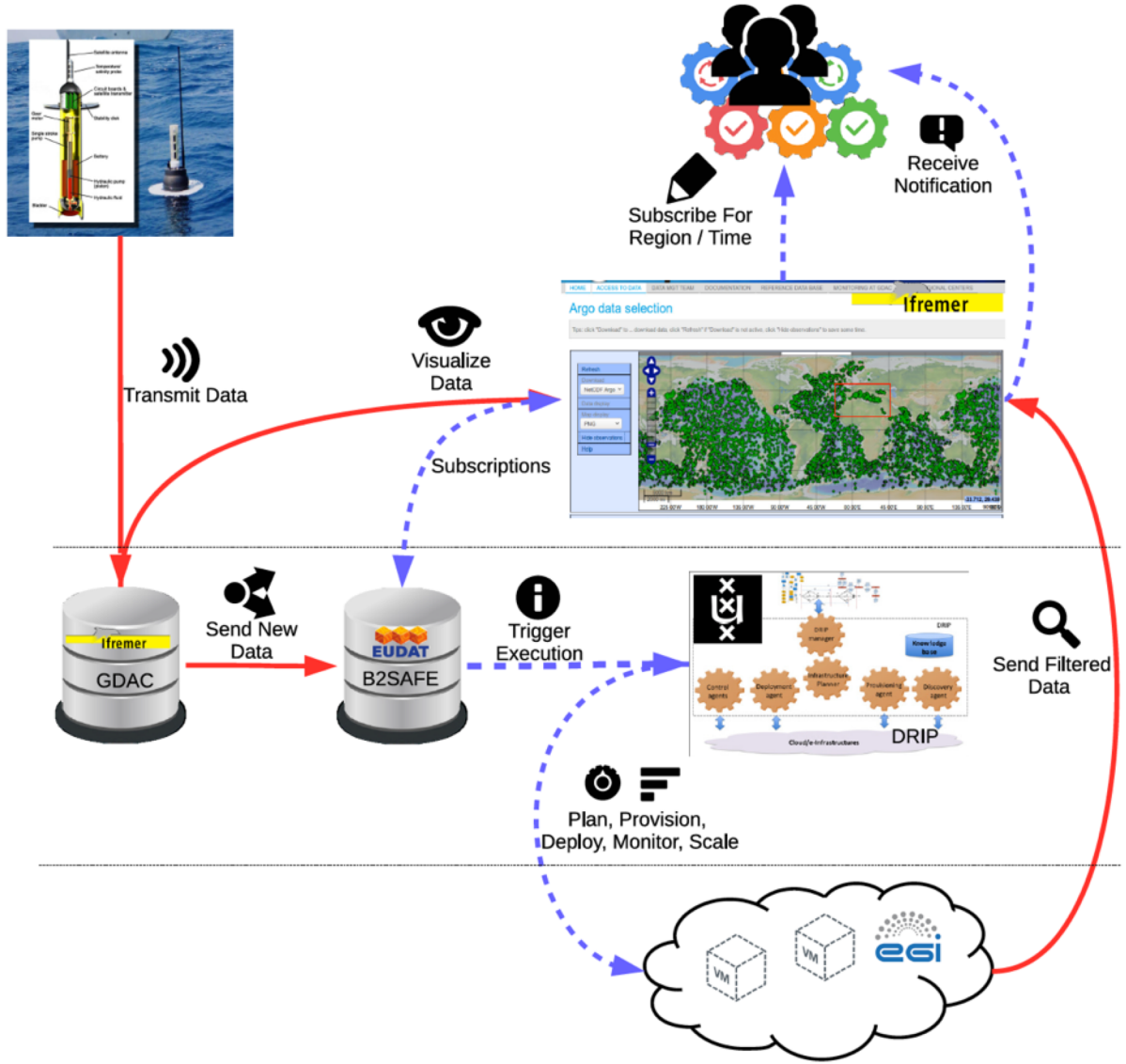


Infrastructure challenges:

1. Subscriptions are diverse and changing
2. Subscriptions require different data region, time duration and computation
3. Subscriptions are for different purposes: simulation models, real-time decisions, or research etc.
4. Subscriptions have different requirements: e.g., time critical or high QoS
5. Etc.

Data processing using Cloud resources

- On demand virtual infrastructure planning and provisioning
- Schedule the subscriptions based on the time requirements, and data dependencies

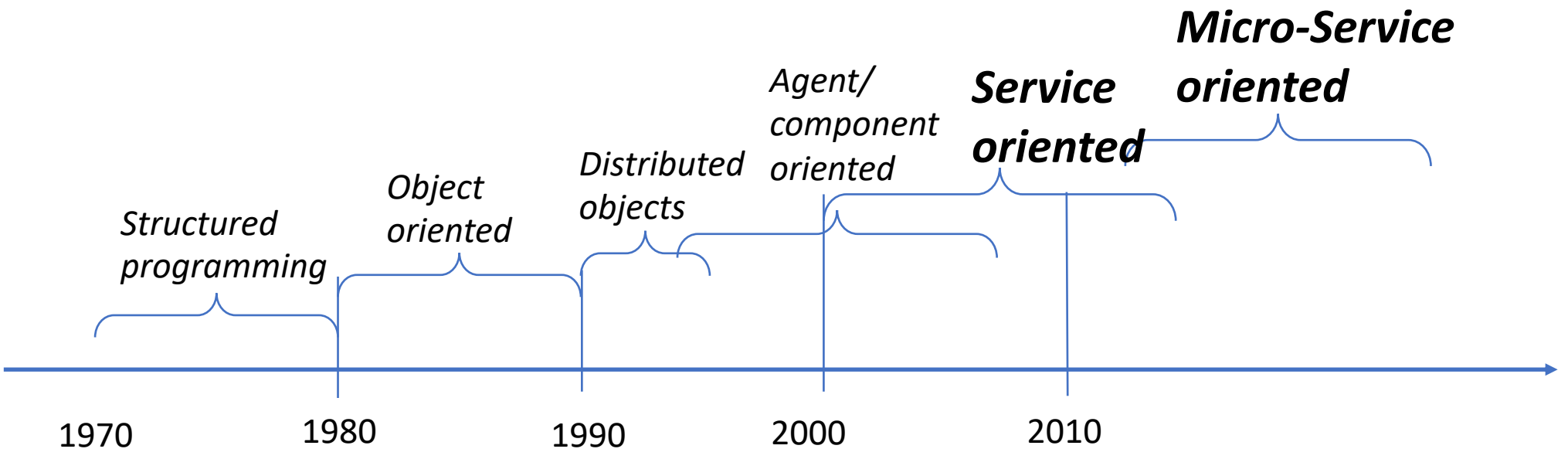


4. Service oriented architecture in scientific computing?

- Service oriented architecture, Web service and RESTful service
- How do services work
- Service composition

In the evolution of programming paradigms

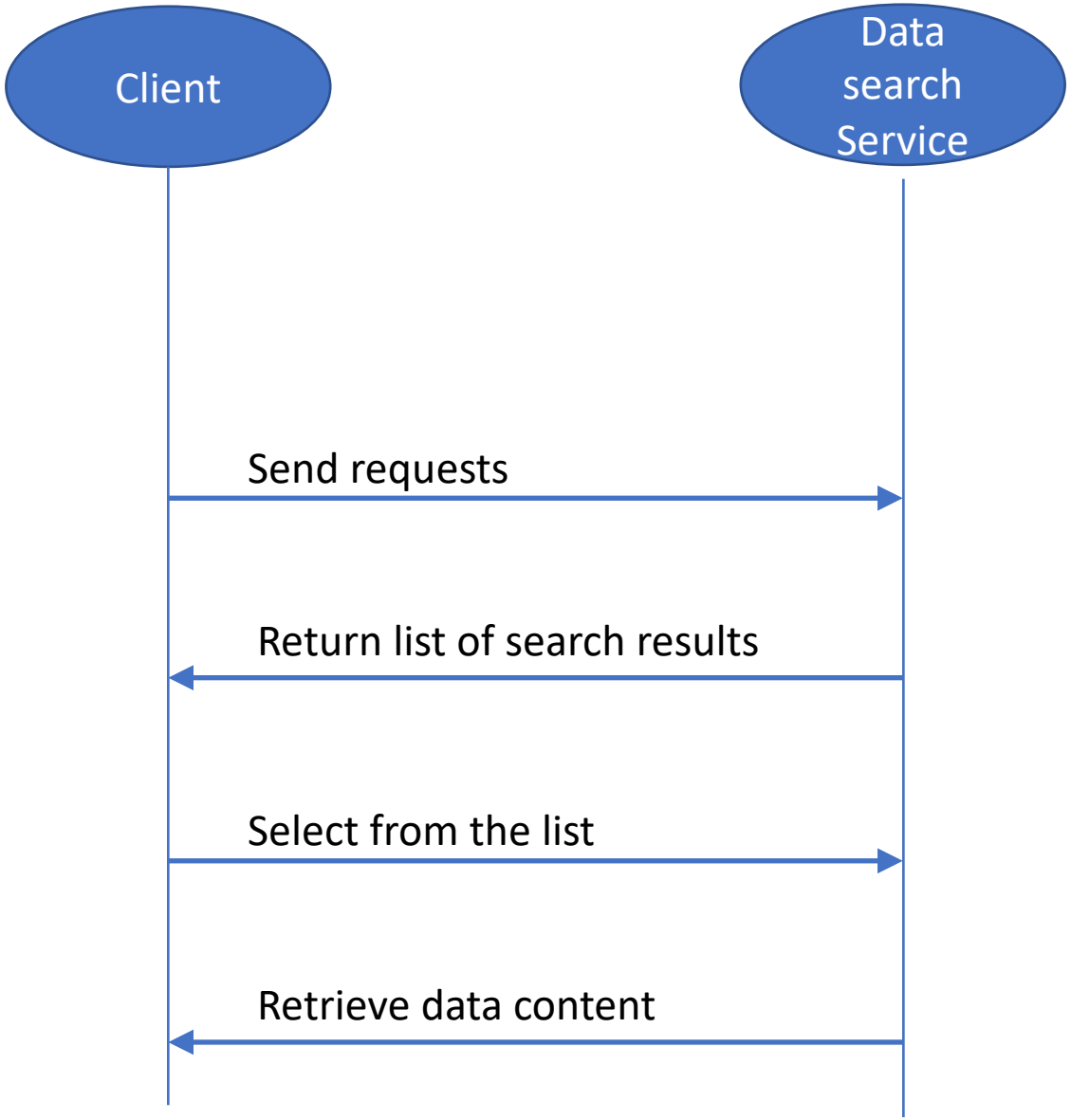
- A **programming paradigm** is a **fundamental style** of computer programming that deals with how solutions to problems should be **formulated** in a **programming language**.



Services and service orientation

A typical scenario:

- **Send query (e.g., ocean data in North sea)**
- **Receive search results**
- **Select result**
- **Retrieve the data content of a selected item**

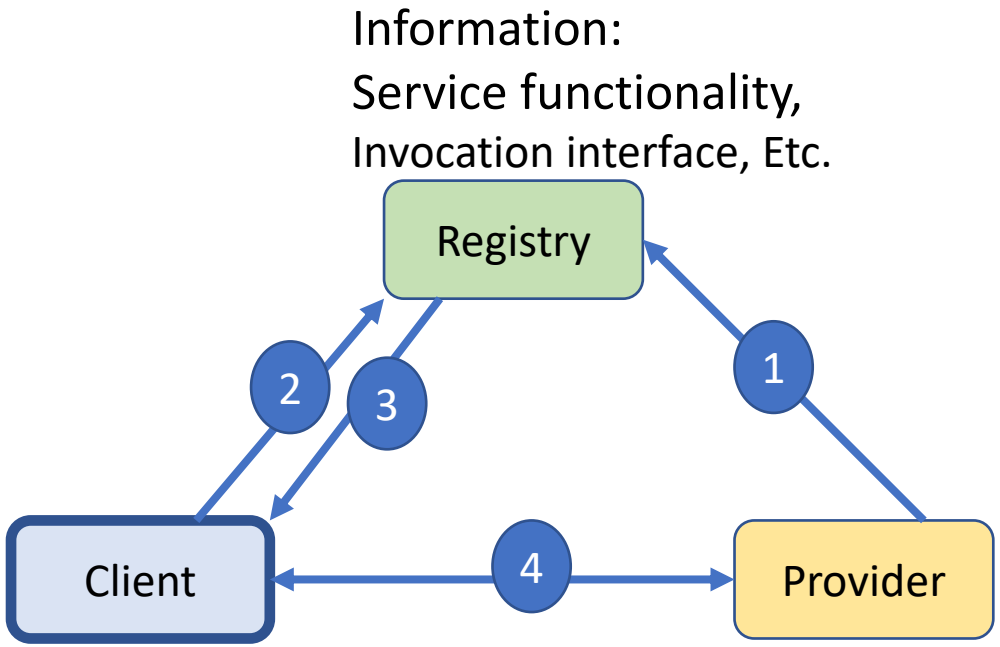


Key aspects

- Web services are ***resources,*** which can be discovered and shared
- Web services provide standardized interfaces
- Web services can be invoked and integrated via a workflow engine

Service discovery

- A developer (*provider*) can **publish** your components as services
- A application developer (*consumer*) can **discover and reuse** existing services
- Dynamic **binding** between client/provider

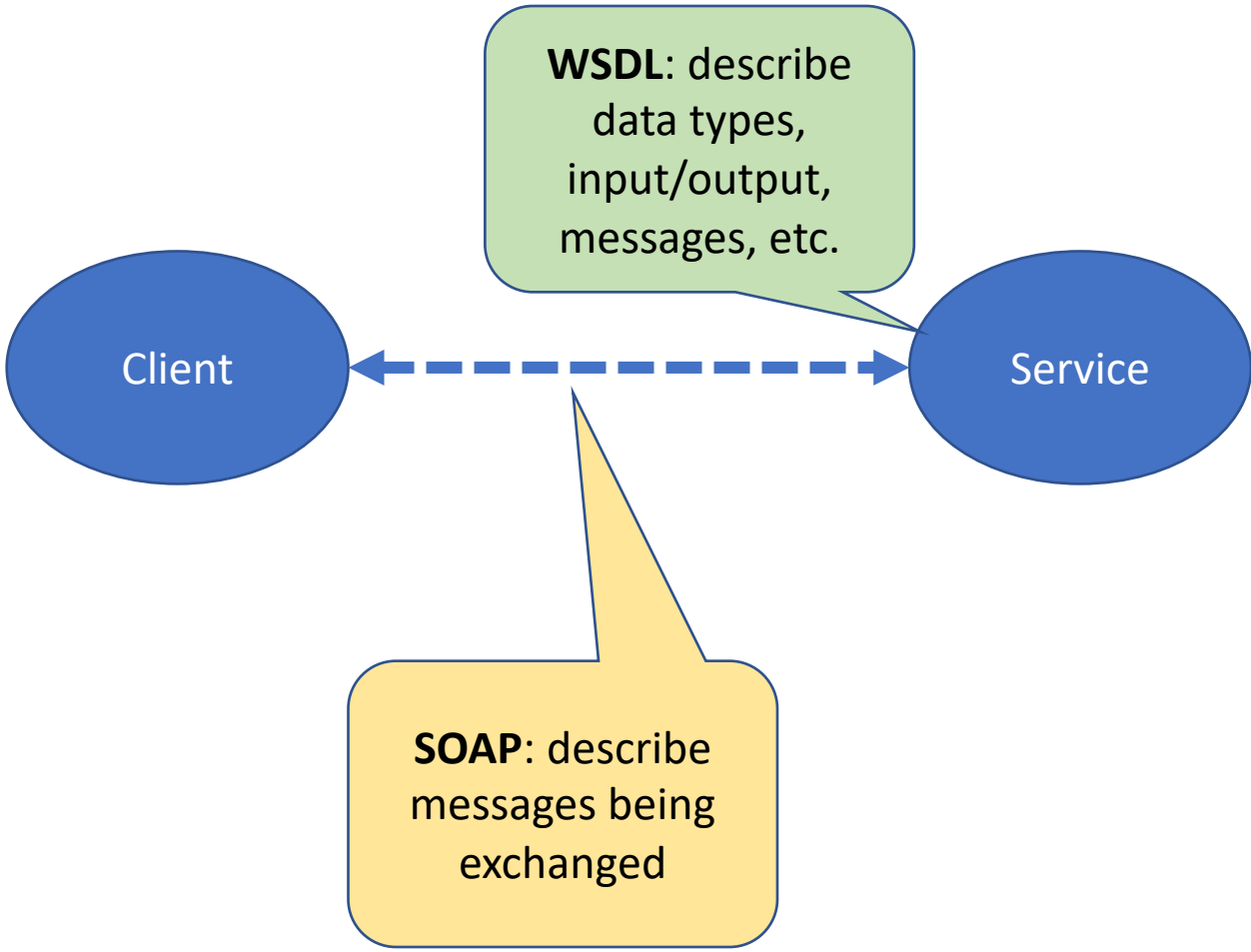


Service descriptions

- **Service description**
  - Functionality (interface)
  - Quality of Service (QoS)
  - *Others (operational constraints etc.)*
- **Service level agreement**
  - Contract between service provider/user

Web service using SOAP/WSDL

- WSDL
- SOAP



Web service

- WSDL for interface description

```
<definitions>
 <types>
      definition of types.....
 </types>
 <message>
      definition of a message....
 </message>
 <portType>
      <operation>
         definition of a operation.....
      </operation>
 </portType>
 <binding>
      definition of a binding....
 </binding>
 <service>
  definition of a service....
 </service>
</definitions>
```

Web service

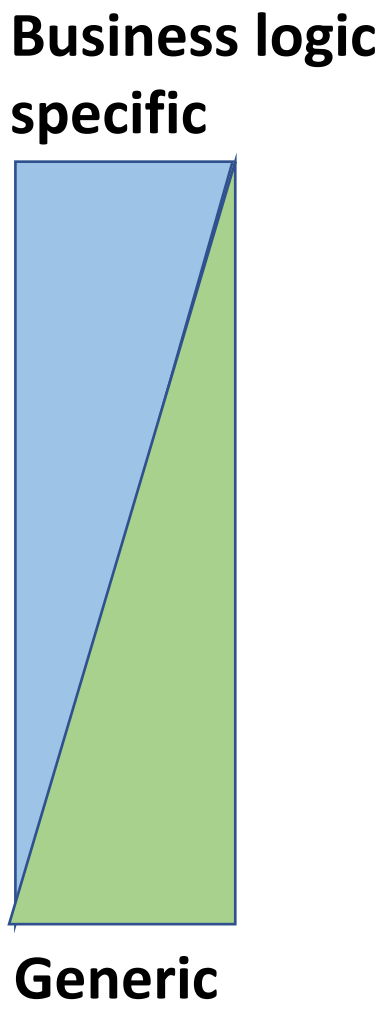
- WSDL for interface description
- SOAP for communication

```
<definitions>
 <types>
     definition of types.....
 </types>
 <message>
     definition of a message....
 </message>
 <portType>
      <operation>
        definition of a operation.....
     </operation>
 </portType>
 <binding>
     definition of a binding....
 </binding>
 <service>
  definition of a service....
 </service>
</definitions>
```

```
<soap:Envelope>

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>

</soap:Body>

</soap:Envelope>
```

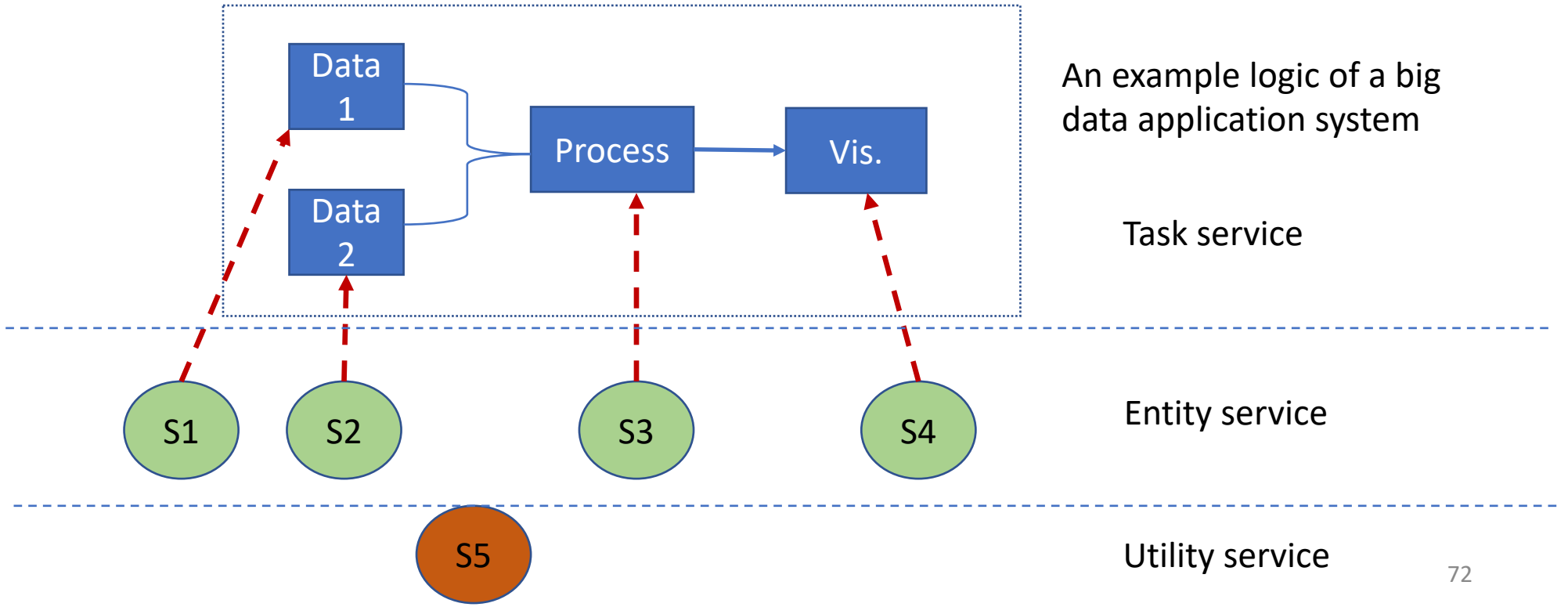
Types of services

- **Task service:**
  - directly related to a business task of a process.
  - Often contains specific business logic
- **Entity service**
  - Business centric, reusable within certain business boundary
- **Utility service**
  - business-logic agnostic

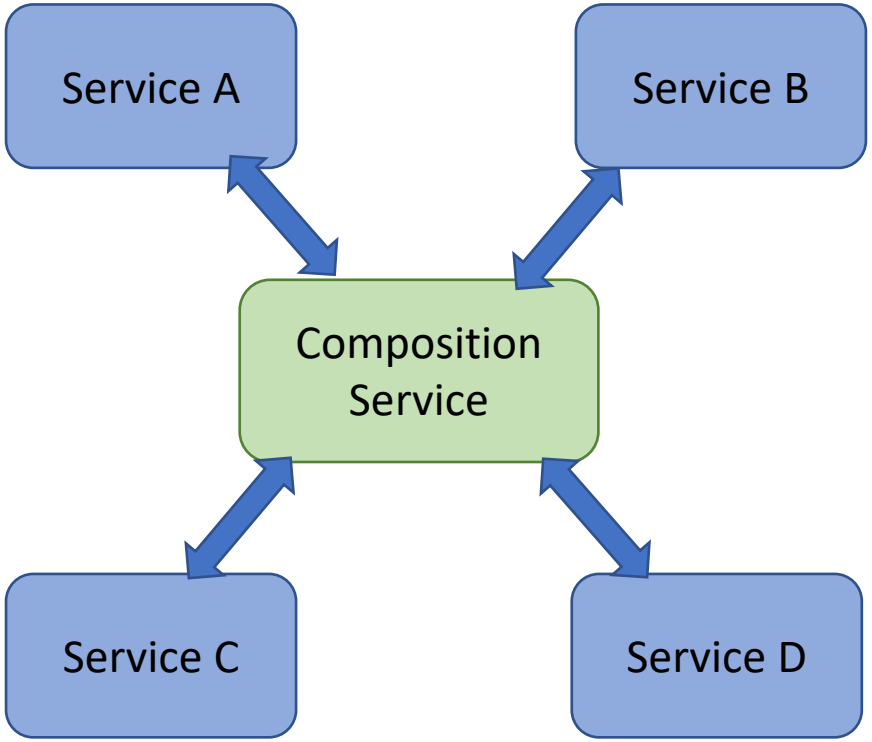


Services can be composed

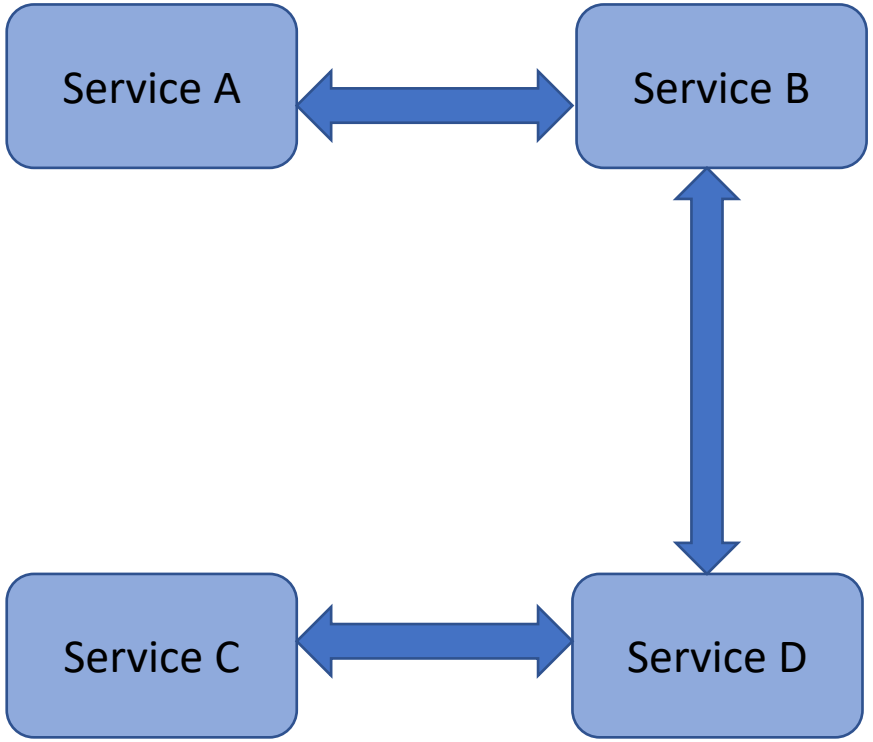
- Service can be a building block for larger services
  - Aggregation based on process logic or data
  - Integration based on workflows



Service orchestration and choreography



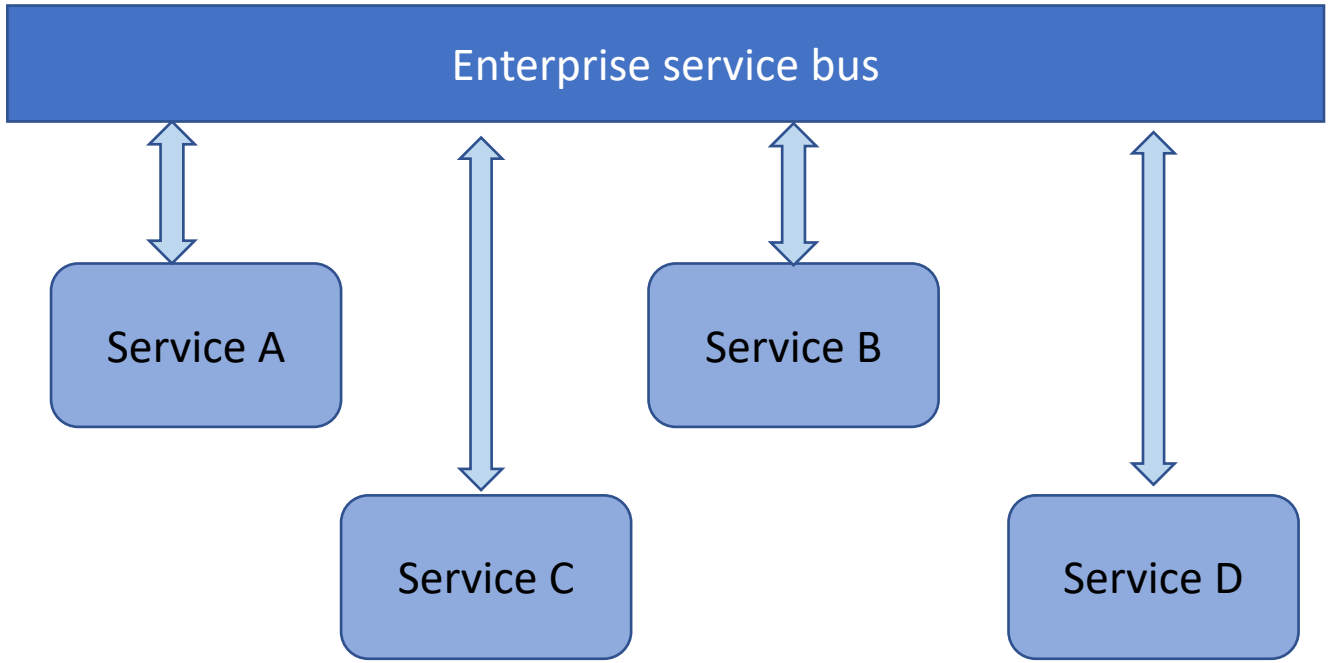
Orchestration



Choreography

Service oriented architecture

- Services interact via a “software bus”
- Messages are queued and distributed via bus
- Have security check at the bus
- Service bus can be federated



Service implementation

- Communication protocol
  - HTTP based: web services
  - Or other protocols
- Message schema
  - XML, JSON, or others
- Interface description

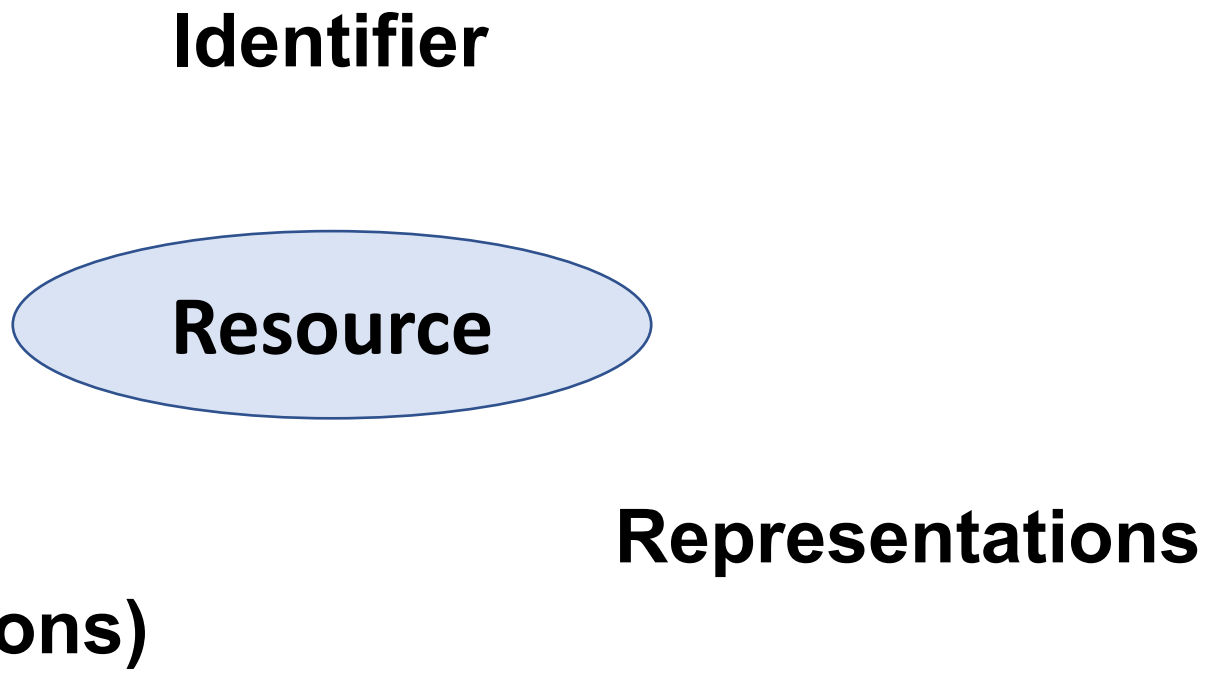
Web service implementation

- RPC: remote procedure call
  - **JSON-RPC, XML-RPC:** Using HTTP/POST, simple, using JSON/XML,
  - **SOAP**: an extension of XML-RPC, with XML schematic restrictions
- REST: Representational State Transfer

REST: Representational State Transfer

- **Representational State Transfer (REST)** is a software architectural style for networked system.
- **Web services** that conform to the REST architectural style, termed **RESTful web services （Restful Web API）**

Main Concepts in REST



Resources: basic abstraction

- **Any thing that can be named** can be a resource: a document, a service, a collection of other resources, a non-virtual object (e.g. a device), and so on
- Represented with a global identifier (**URI** in HTTP)
  - *http://www.dataportal.com/ocean/europe*

Naming Resources

- REST uses URI to identify resources
  - http://www.anexampledataportal.com/ocean
  - http:// www.anexampledataportal.com/ocean/europe
  - http:// www.anexampledataportal.com/ocean/europe/20101101

Verbs: actions performed on resources

- **GET**: Retrieve
  - GET http:// www.anexampledataportal.com/ocean/Europe/20120118
- **POST**: create new record
  - POST http:// www.anexampledataportal.com/ocean/europe
  - Content: (time, location, parameter, value…)
- **PUT**: Update a record
  - PUT http:// www.anexampledataportal.com/ocean/europe
  - Content: (id, time, location, parameter, value)
- **DELETE**: Delete a record
  - DELETE http:// www.anexampledataportal.com/ocean/Europe/20110123

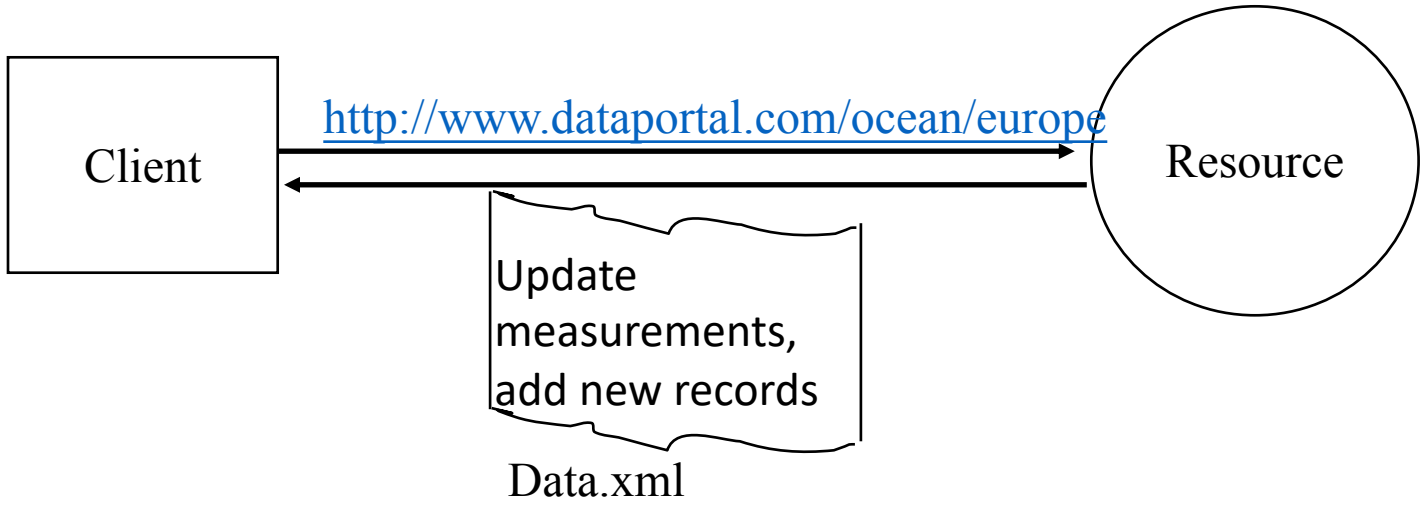
Representations: data returned to the client

- Two main formats:
  - XML
  - JavaScript Object Notation (JSON)
- It is common to have multiple representations of the same data

```
<Measurement>
        <ID>20110202</ID>
        <Parameter>Temperature</ Parameter >
        <Value>XXX</Value>
</Measurement>
```

```
{Measurement
        {id: 20110202}
        {parameter: Temperature}
        {value: xxx}
}
```

State Transfer



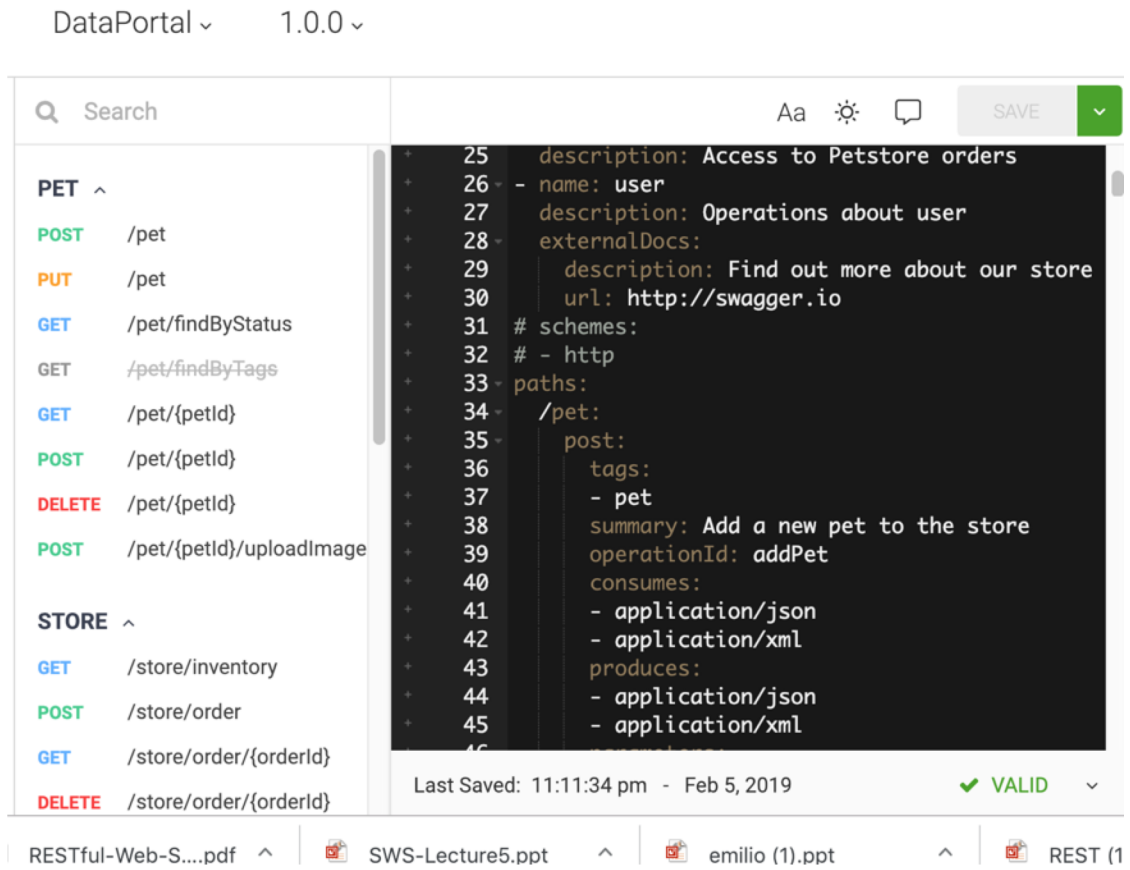
The Client references a Web resource using a URI. A **representation** of the resource is returned (in this case as a XML document).
The client application changes (**transfer**s) state with each resource representation of the resource --> Representation State Transfer!

Development

- Define API interface
- Generate code
- Develop code

RESTful web service definition

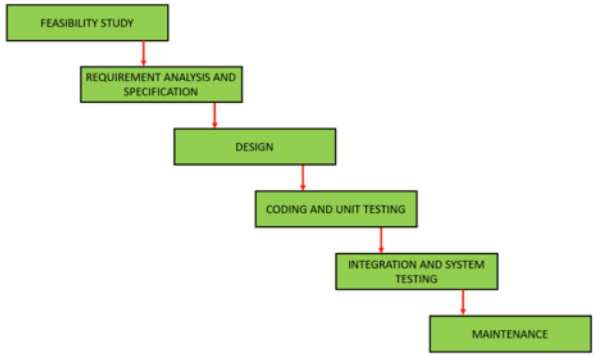
- Swagger environment
- Export to the server/client side code

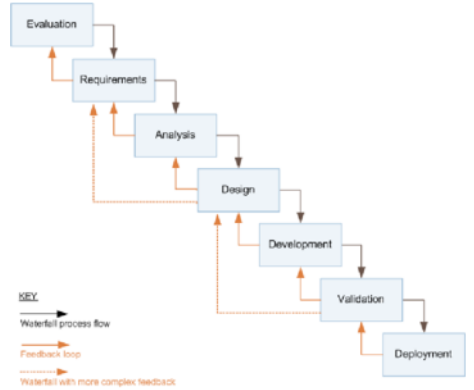


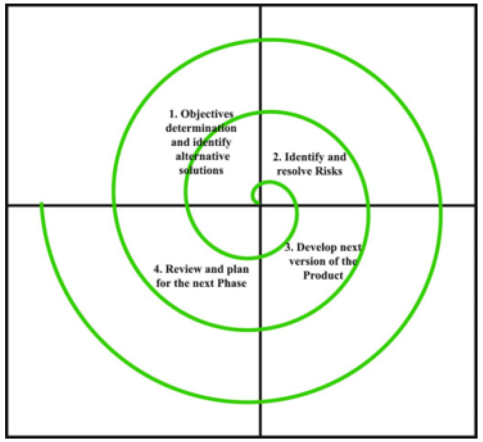
5. Developing data management services using software engineering approach

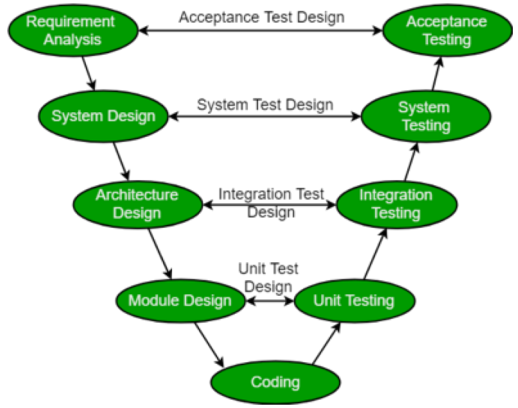
Can we use classical model?

- Water-fall
- Iterative
- Spiral
- V-shaped

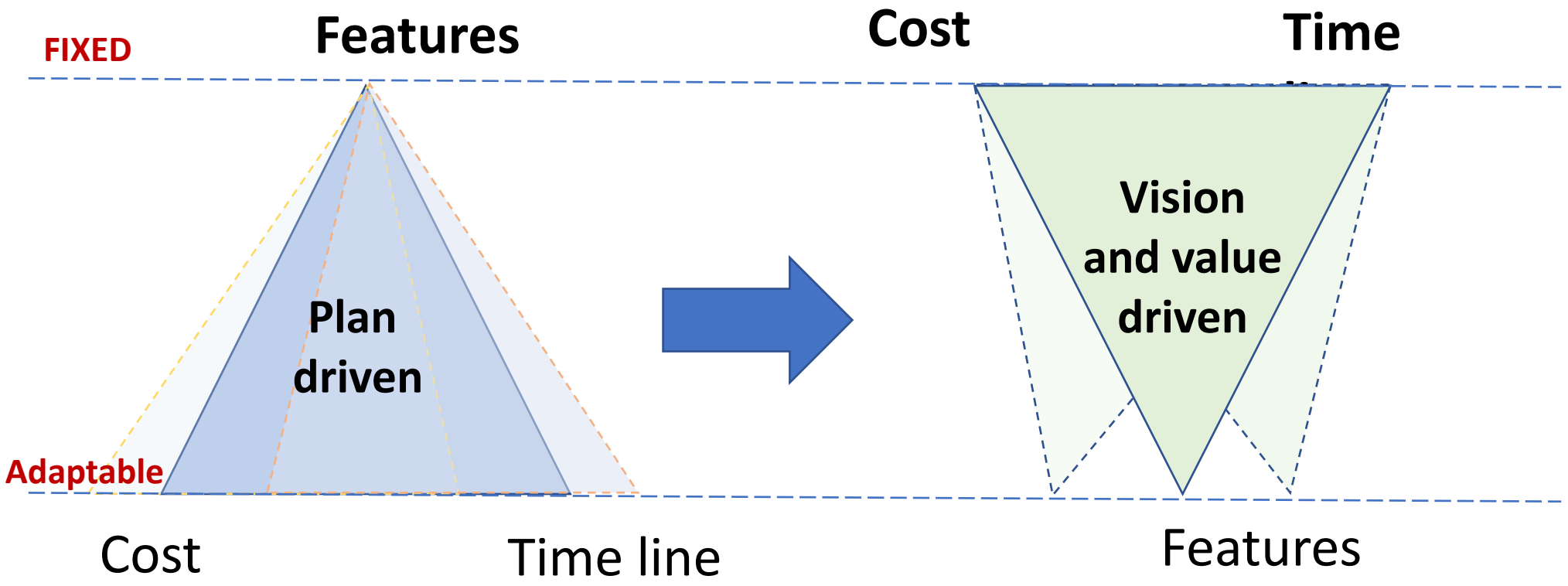








Think it in agile way?



Agile Project Management: Creating Innovative Products (2nd Edition) 2nd Edition by Jim Robert Highsmith

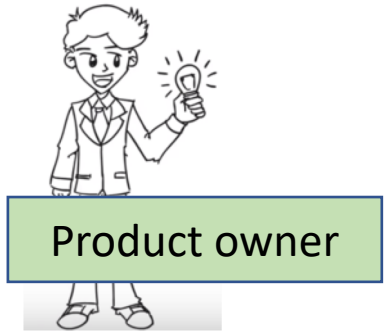
Think it in agile way?

- Business **goal**, **user stories and stakeholders (Customer in the loop)**
- **Make a prioritized feature list (decompose the user story)**
- **Prepare task lists** for developers

Using scrum

- Role:
  - Product owner, Scrum master and Development team
- Artifacts
  - Product backlogs, user stories, sprint backlogs, and burndown chat
- Meetings
  - Sprint planning, Daily scrum, Sprint review
- Workflow

Scrum workflow (customizable)

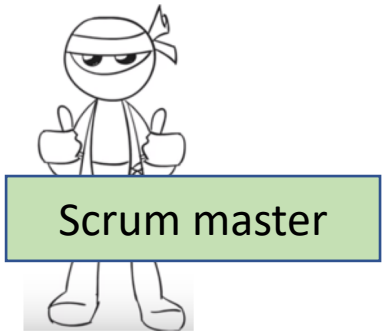


Product owner

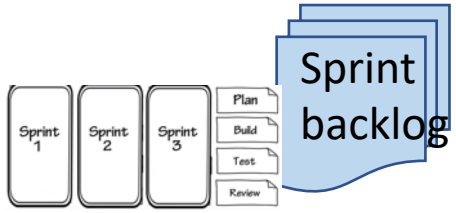
Product backlog

As a___
I need ___
So that ___

User stories



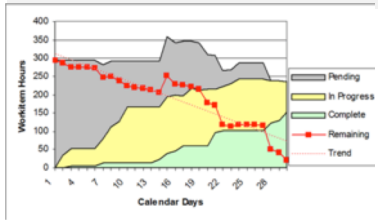
Scrum master



Sprint backlog

Sprint planning

Burn down chart



Did yesterday,
To do today
Obstacles etc.

Daily Scrum

Development team

Sprint review, retrospective

From features to development tasks

- **Decomposition feature/function**
  - Information viewpoint
  - Computation viewpoints
- **Putting things back into a system**
  - What technology should we choose?
  - What architecture should we use?

Make your RESTful services

Instruction:

https://bit.ly/2LyyV6s