Marie Delavergne

Cheops, a service mesh to geo-distribute (micro-)service applications at the Edge

Supervisor: Adrien LEBRE - Professor, IMT Atlantique
Co-supervisor: Ronan-Alexandre CHERRUEAU (2019-2020) - DGFiP
Reviewers :
Noël DE PALMA - Professor, Université Grenoble Alpes
Pierre SENS - Professor, Sorbonne Université
Examiners :
Sara BOUCHENAK - Professor, INSA Lyon
Thomas LEDOUX - Professor, IMT Atlantique

Inria informatics mathematics
IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom
LS2N Laboratoire des Sciences du Numérique de Nantes

Research question

Is it possible to use applications developed for the Cloud on Edge infrastructures without changing their code?

I Context: Cloud Computing
II State-of-the-art
III The approach: Cheops and scope-lang
IV Conclusion & perspectives

Context:
Cloud Computing

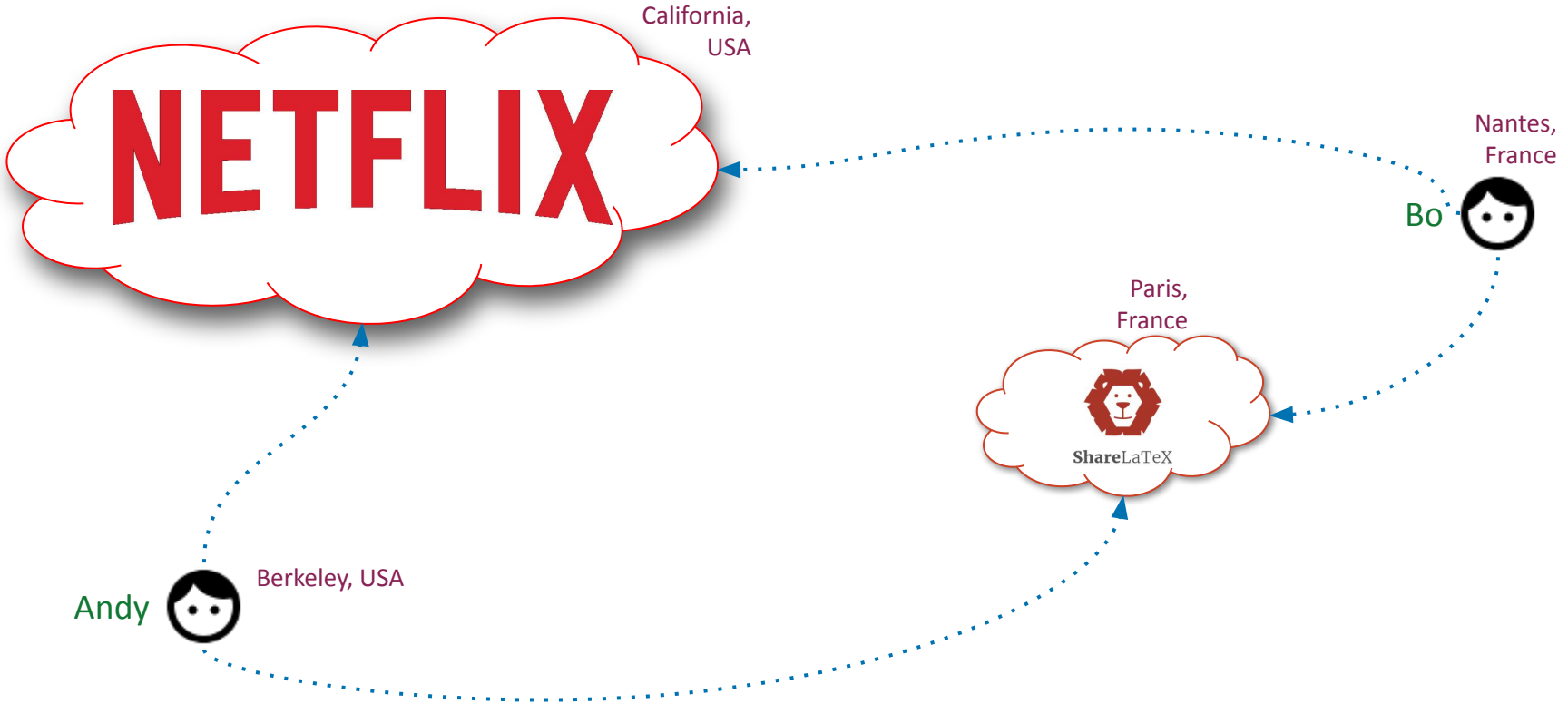
A simplified view of the Cloud

Source

NETFLIX

ShareLaTeX

A simplified view of the Cloud

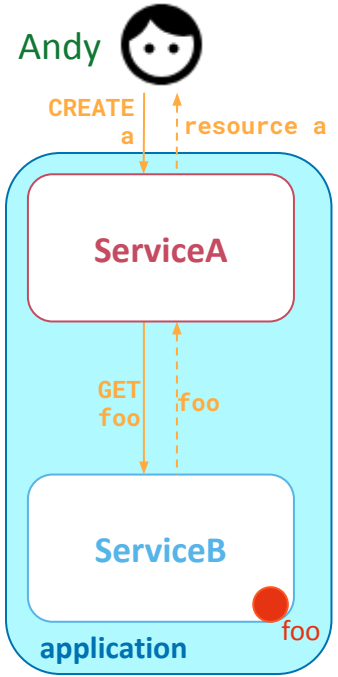


Cloud applications are (often) made of services

Services manage a subset of resources of the applications.

To create a **resourceA** on **ServiceA** using a *sub-resource* foo from **ServiceB**, Andy needs to give a command like this:

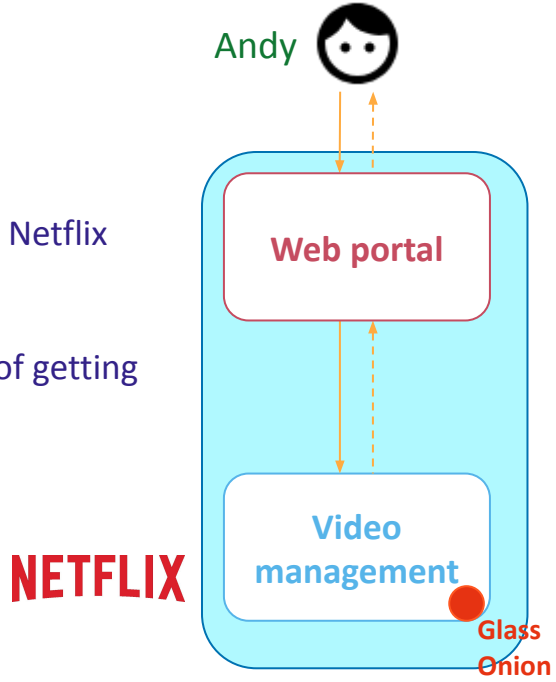
```
application create resourceA --sub-resourceB foo
```



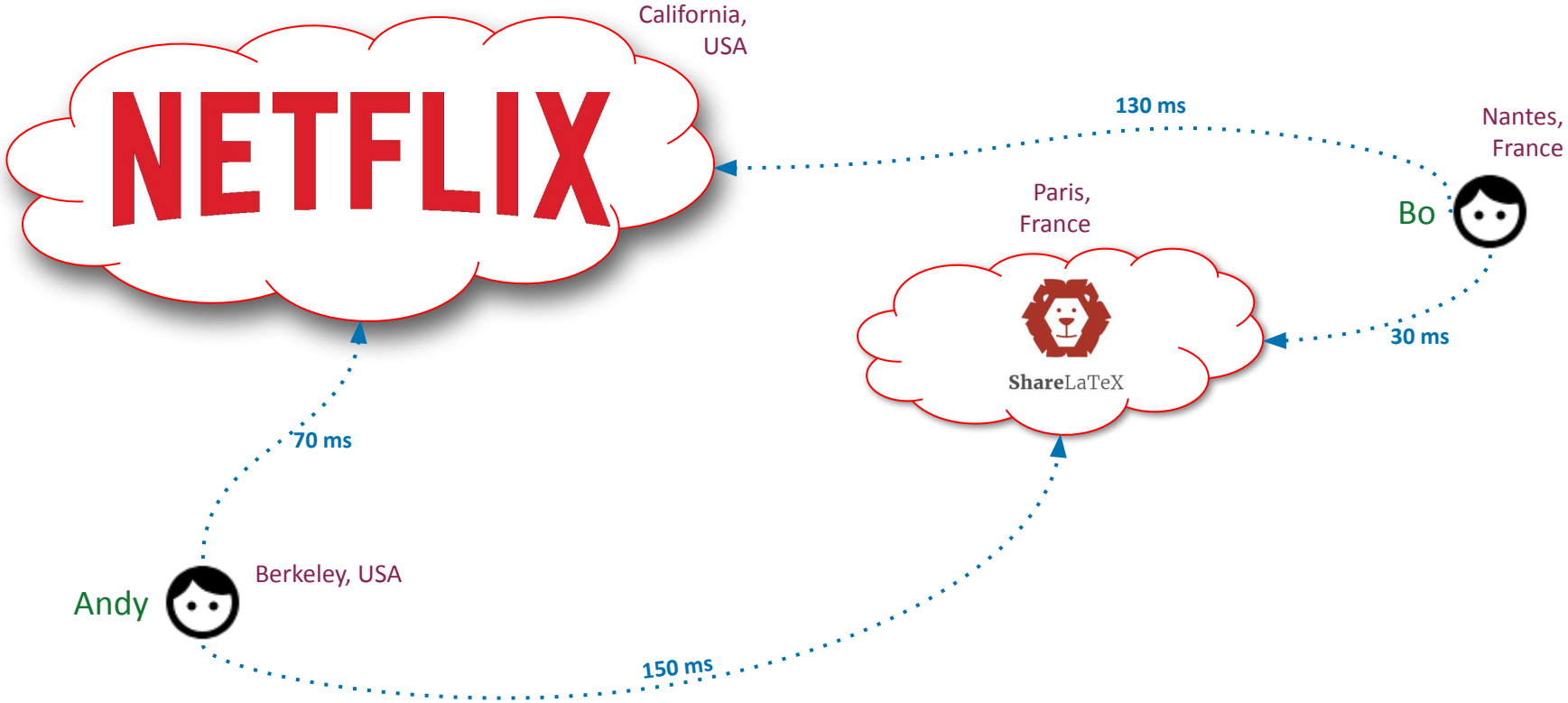
Cloud applications are (often) made of services

Andy wants to watch Glass Onion on Netflix

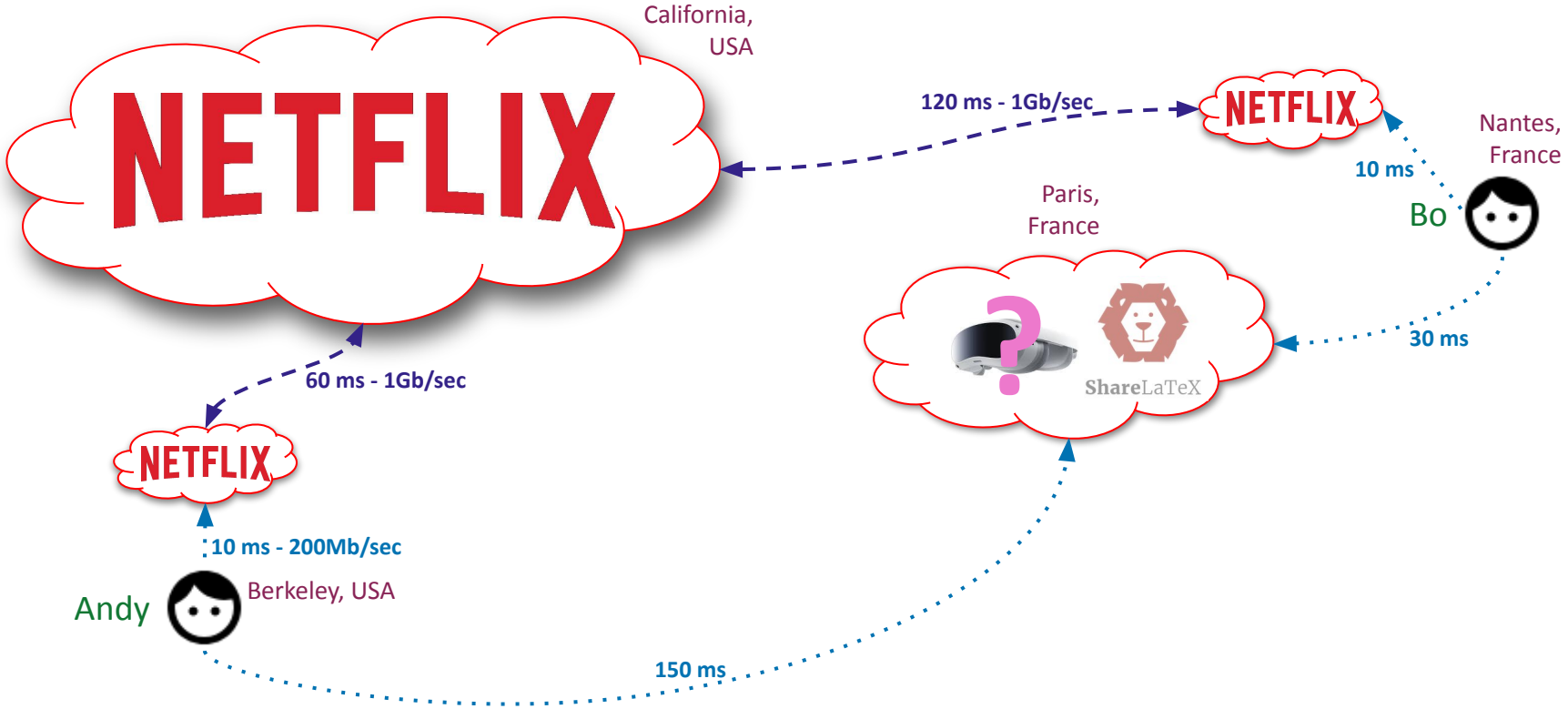
She reaches the web portal, in charge of getting the video



Cloud vs latency/throughput



Cloud vs latency/throughput



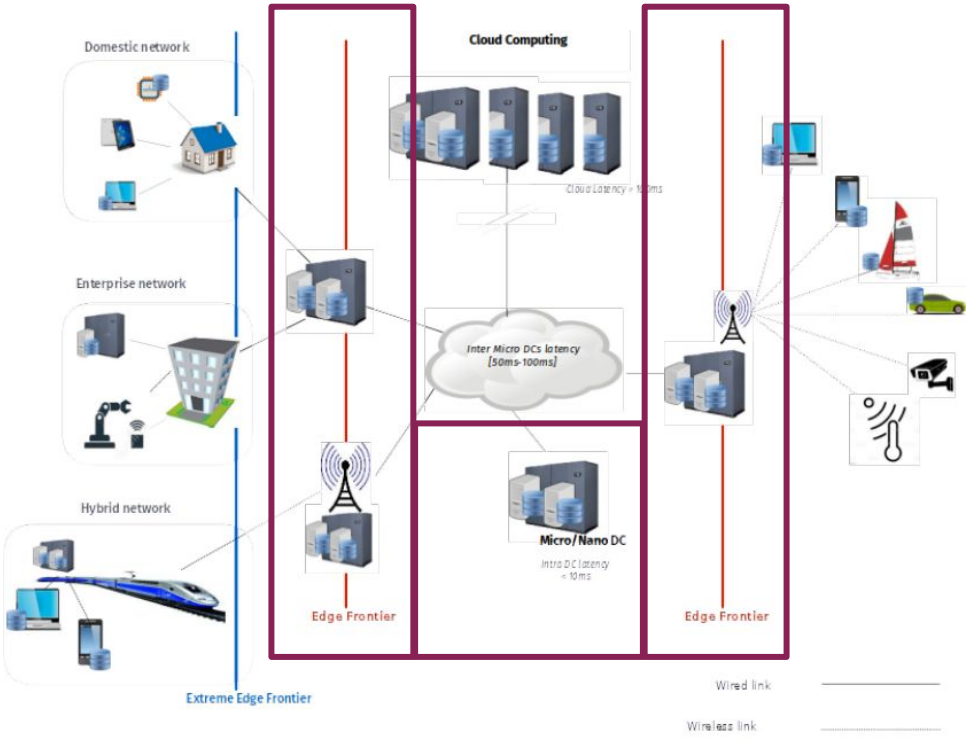
Edge Computing: What is the purpose?

Source

Reduce the latency between the Cloud clients and the datacenters

for **latency-sensitive** and **data-intensive** applications

Many Edge definitions: our vision



A geo-distributed Cloud:

An infrastructure of micro DCs

- Closer to users
- Managed by one operator

Challenges:

- **High latency** between far sites
- **Disconnections** between sites

How can we geo-distribute an application?

NETFLIX is natively geo-distributed

Two approaches in the academic literature:

- **ShareLatex**[1]: efforts to geo-distribute services
- **OpenStack**[2]: change the DB to a geo-distributed one

We do not want to change their code,

nor put solutions to push the applications in the business code

[1] Tato et al., *ShareLatex on the Edge* [...], ME'CC18 (ACM Middleware workshops) https://dl.acm.org/doi/10.1145/3286685.3286687

[2] Lebre et al., *Revising OpenStack to Operate Fog/Edge Computing infrastructures*, IC2E 2017 https://hal.inria.fr/hal-01273427

A geo-distributed version of ShareLaTeX?

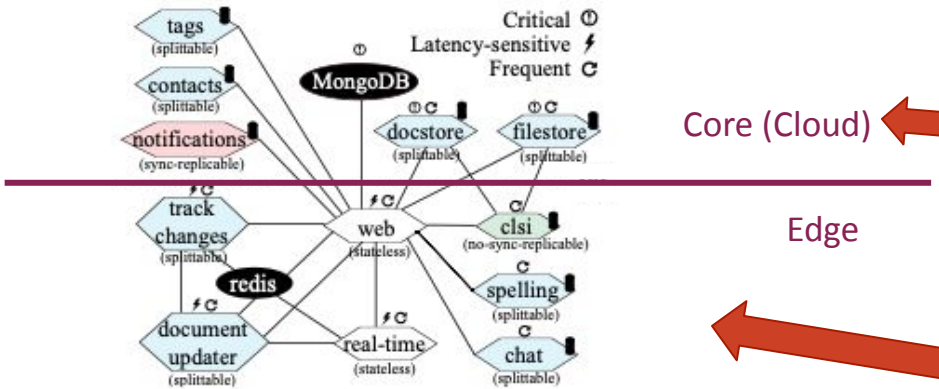
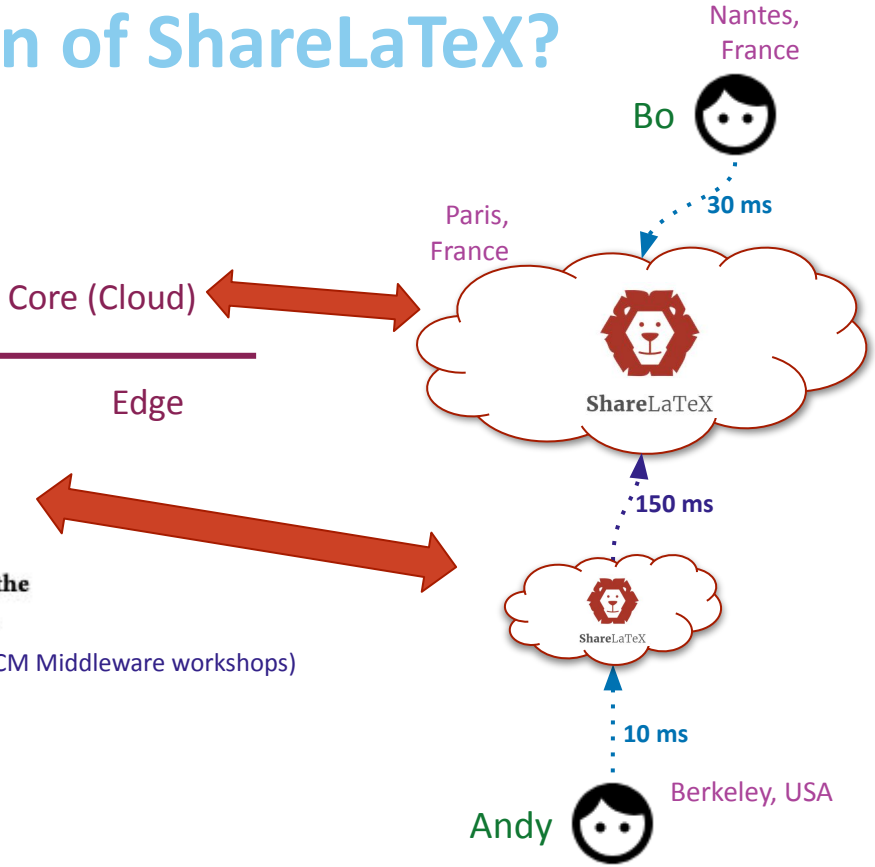


Figure 3: Our split and placement. Critical services in the core, latency-sensitive and frequent services at the edge.

Source: Tato et al., *ShareLatex on the Edge*, ME'CC18 (ACM Middleware workshops)



- Locality?
- Network partitions?

Bases for an approach: locality is crucial

- **Local-first**:
  - Always able to serve local requests
  - Minimize communications between DCs
- **Collaborative-then**:
  - Share resources across sites on demand
  - Replicate them where needed

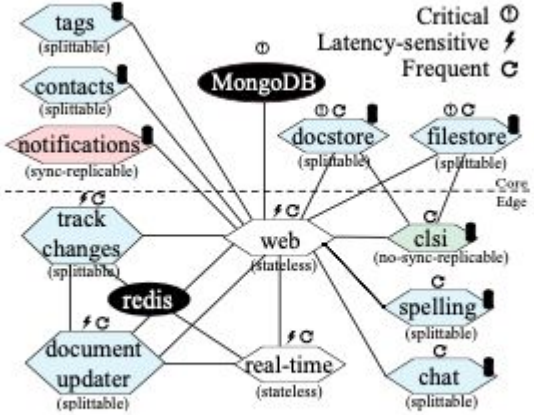


Figure 3: Our split and placement. Critical services in the core, latency-sensitive and frequent services at the edge.

Autonomous Cloud Applications: a first step

Autonomous instances: provides **locality** and **robustness**

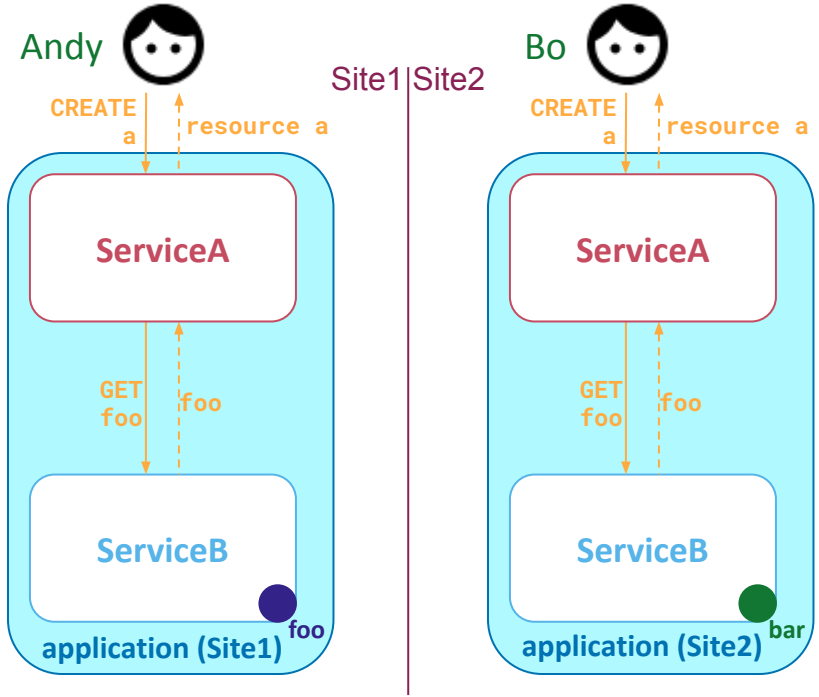
Andy and Bo use their own application, the closest:

Andy: `application create resourceA --sub-resourceB foo`

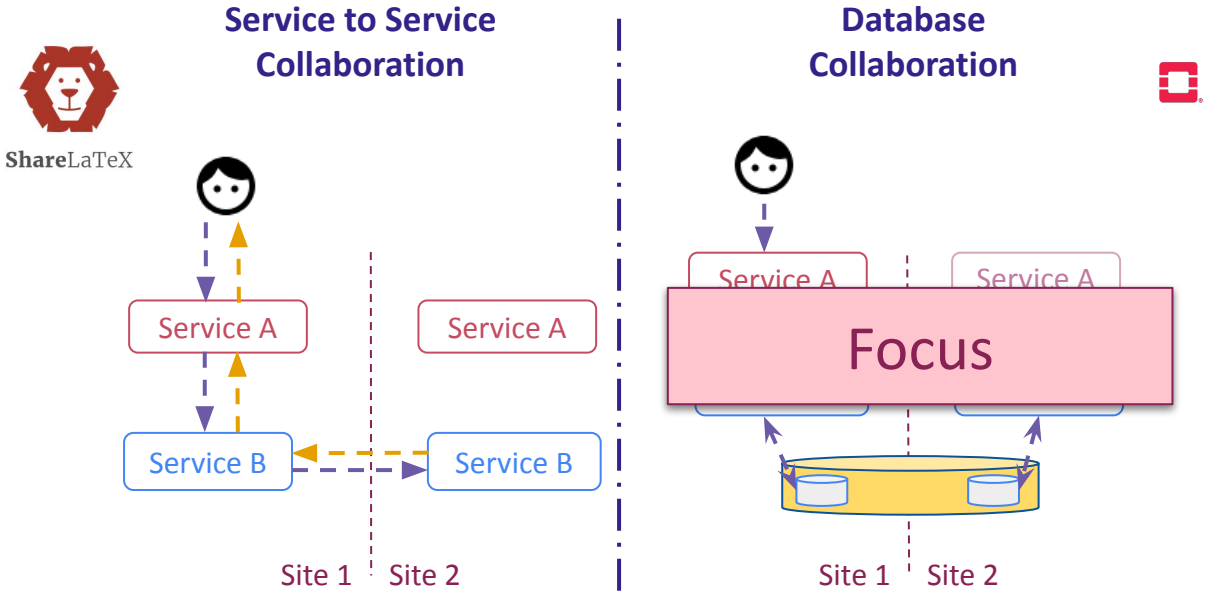
Bo: `application create resourceA --sub-resourceB bar`

Without collaborations between instances, what is the point?

Andy: `application create resourceA --sub-resourceB bar` ?



Different approaches for collaborations

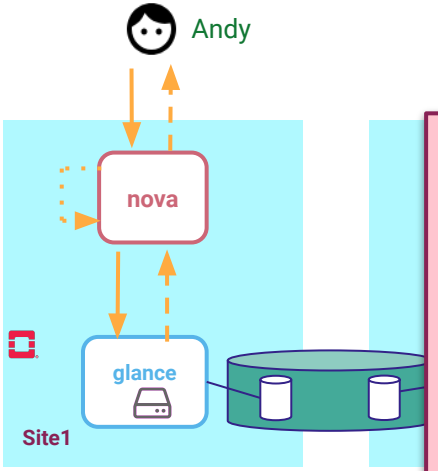


Problems:

The collaboration code is tangled in the business code

Resources have a context and a scope, sometimes side effects

Unanticipated issue



Coupling domain and collaboration code is not *always* an option:

- OpenStack is 13M LoC of Python (1.5M for core projects)‡
- Core developers do not want to add complexity (geo-distribution is not that important for them)

```
- import os
+ import urllib

  @route('/image/{name}')
  def get_image(name: String) -> Blob:
    # Get the path of the image stored

                                  le:///path/debian.qcow
                                  ry(f'''
                                  .path FROM images
                                  IS "{name}";''')
                                  oto://url/debian.qcow
                                  th(name)

                                  e path on the disk to get
                                  ge
-   image = os.open(path, 'rb')
+   # Follow the link to get the qcow image
+   image =  urllib.urlopen(path)
    return image
```

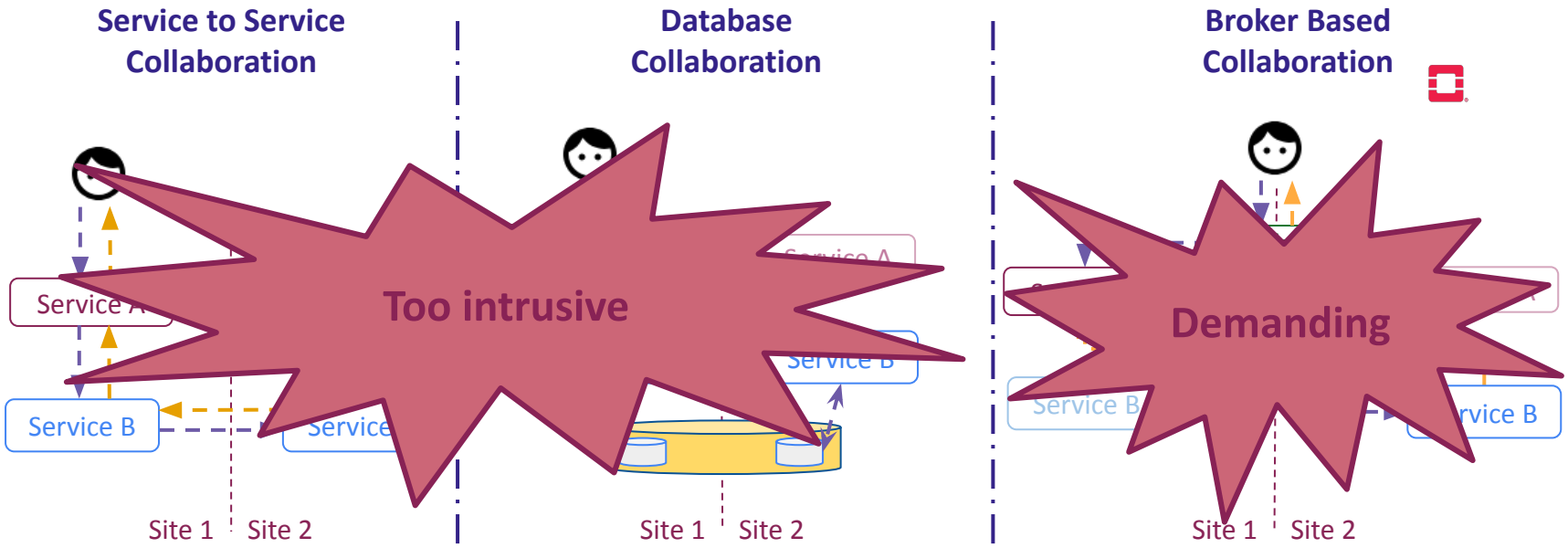
Data sharing ≠ resource sharing

- A **resource** is **data (+ effects)**
- OpenStack VMs, nets, images, volumes **do not fit** in the DB
- **App. code** has to **take into account** the distribution

‡. https://www.openhub.net/p/openstack/analyses/latest/languages_summary

Slide originally made by R.-A. Cherrueau

Different approaches for collaborations



Problems:

The collaboration code is tangled in the business code

Resources have a context and a scope, sometimes side effects

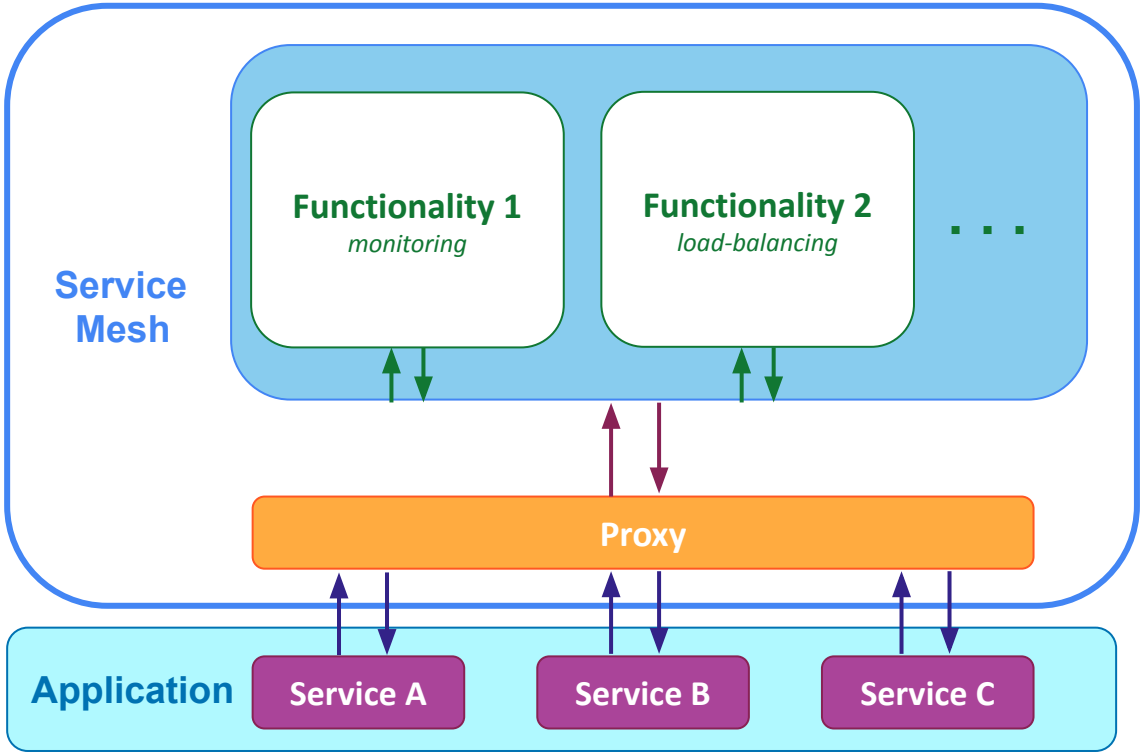
Brokers need to be developed for each service

The different approaches are either intrusive or tedious

Is it possible to find a **generic** solution to **externalize geo-distribution** concerns?

Lifecycle of service based applications can be managed externally through mechanisms such as a **service mesh.**

A simplified view of a service mesh



Purpose:
Manage some functionalities outside of the business code

Example of functionalities:

- load-balancing
- monitoring

  .

  .

  .
- geo-distribution ?

Research questions, more specifically

- **Is it possible to use applications developed for the Cloud on Edge infrastructures without changing their code *leveraging a service mesh-like approach ?***

- Use case: OpenStack, a Cloud infrastructure manager (and then Kubernetes)
  - OpenStack: complex application / 186 services (8 core services)
  - If OpenStack can be geo-distributed, we can propose a solution to manage an **Edge infrastructure** (which is an open challenge too)

State of the Art (cont)
Other academic and industrial solutions

Criteria

Non-intrusive	no changing the code
Generic	should work on as many applications as possible
Local-first	ability of the solution to use only the local site
Collaborative-then	collaboration between sites
Network partition	robust to network partition
Dynamic	execution of requests handled dynamically
On-demand	execution chosen per request by the users
Decentralized	decentralization of the solution
Peer-to-Peer	equality of privileges on the different sites

Comparison (more ☁ = more 😀)

	Approach	Non-intrusive	Genericity	Local-First	Collaborative-then	Resiliency to Network Partitions	Dynamicity	On-demand	Decentralized	P2P
Resource management on Edge Infrastructures	**Decentralized CloudLightning**	☁☁☁	☁☁☁	-	☁☁☁	☁	☁☁☁	☁	☁☁☁	☁☁☁
	Liqo	☁☁☁	☁☁	☁☁☁	☁☁☁	☁☁☁	☁☁	☁☁	☁☁☁	☁☁
	HYDRA	☁☁☁	☁☁	☁	☁☁☁	☁☁☁	☁☁	☁	☁☁☁	☁☁☁
	OneEdge	☁☁☁	☁☁☁	☁☁	☁☁☁	☁	☁☁☁	☁	☁☁	☁☁
	OpenYurt	☁☁☁	☁☁	☁☁	☁☁☁	☁	☁☁	☁	☁	☁
	P2P OpenStack	☁☁☁	☁☁	☁☁☁	☁☁	☁☁☁	☁☁	☁☁	☁☁☁	☁☁☁
Service meshes	**Istio**	☁☁☁	☁☁	☁☁☁	☁☁	☁	☁☁	☁	☁	-
	Linkerd	☁☁☁	☁☁	☁☁☁	☁☁	☁☁	☁☁	☁	☁☁	-
Towards Edge-native applications	**Scalable Gabriel**	☁	☁☁	☁	☁☁	☁	-	-	☁	☁
	Colony	☁	☁☁☁	☁☁☁	☁☁☁	☁☁☁	☁☁☁	☁☁	☁☁	☁☁

The approach:
Cheops and scope-lang

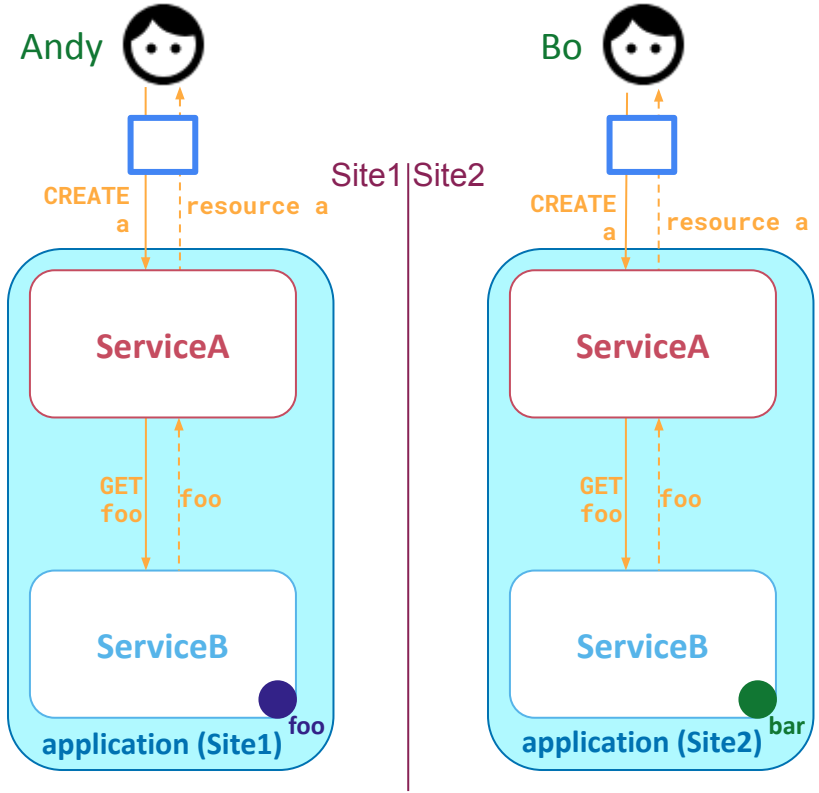
Autonomous Cloud Applications: a first step

Andy and Bo use their own application, the closest:

Andy: `application create resourceA --sub-resourceB foo`

Bo: `application create resourceA --sub-resourceB bar`

We need a solution to manage the geo-distribution externally and in a generic manner

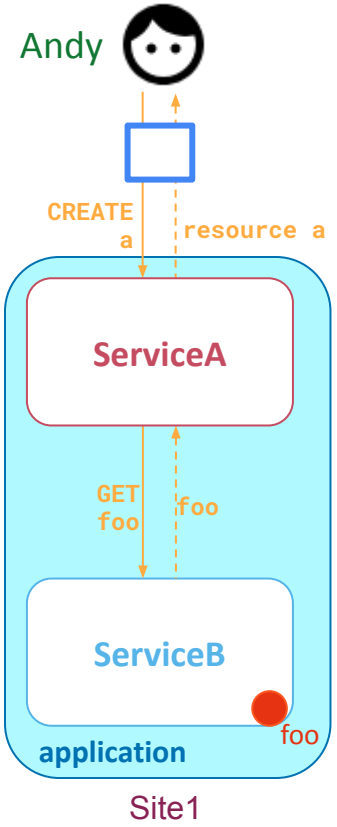


Scope-lang

- A DSL to **manage resources** & **integrated with native CLI**
- A scope-lang expression contains:
  - **Collaboration** & **service** information
  - **Location** information
- Gives users a set of operations they can use to decide **where** and **how** a request will be executed.

For example:

```
application create resourceA  --sub-resourceB foo
                              ≡
application create resourceA  --sub-resourceB foo
                              --scope {ServiceA: Site1,
                                       ServiceB: Site1}
```

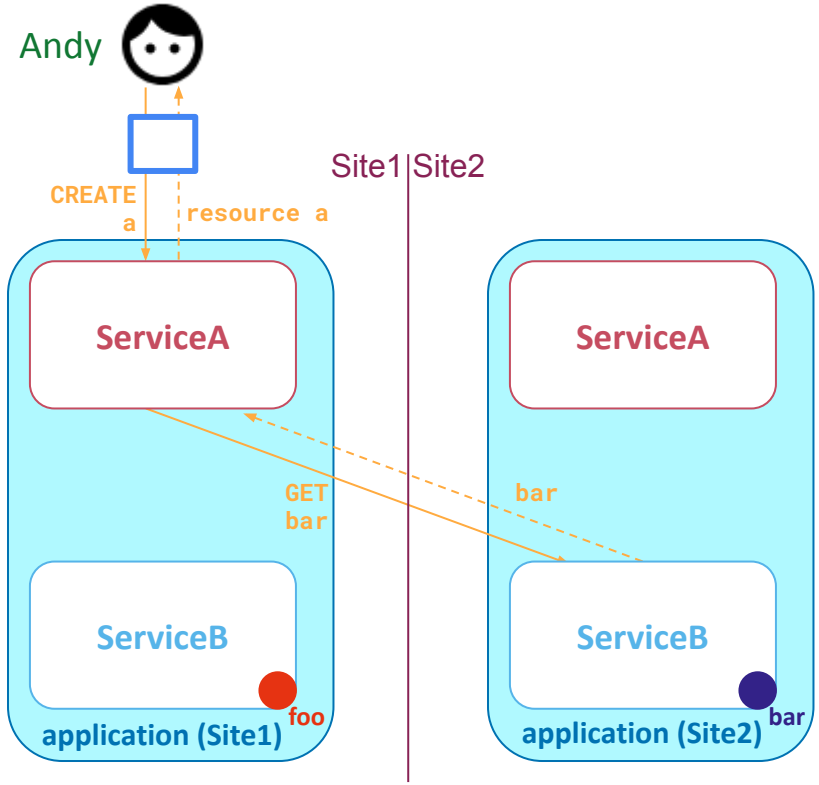


Scope-lang

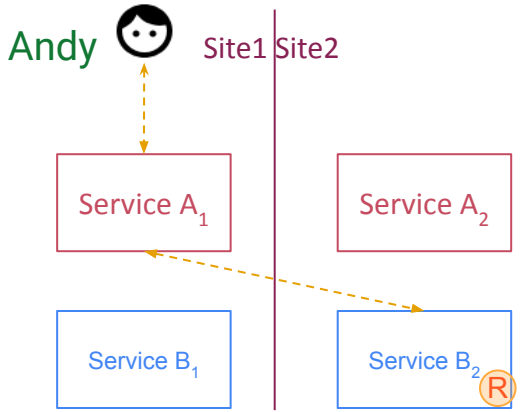
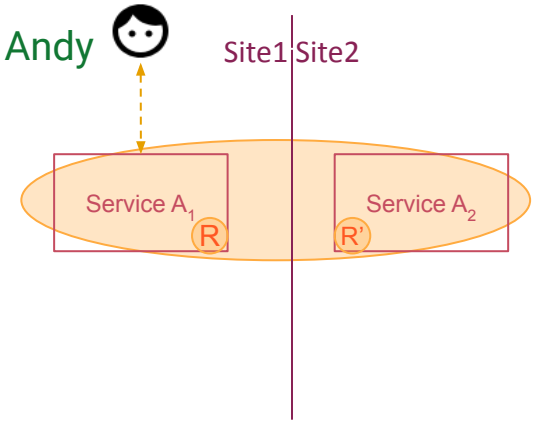
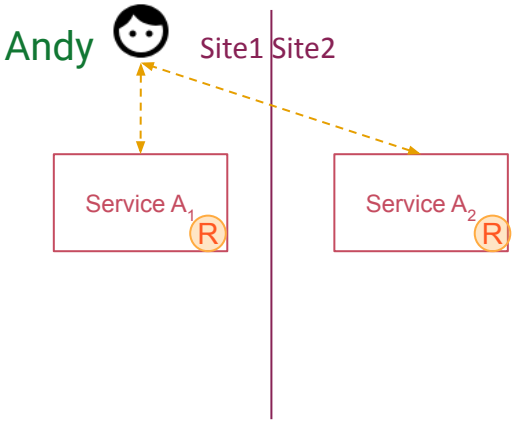
Scope-lang gives users a set of operations they can use to decide where and how a request will be executed.

For example:

```
application create resourceA
                  --sub-resourceB bar
                  --scope
                    {ServiceA: Site1,
                     ServiceB: Site2}
```



3 types of Collaborations[1]

Resource ***sharing***	Resource spanning a***cross*** different instances	Resource ***replication***
		
Sharing service B from Site2 has the required resource	**Cross** Andy creates a resource that span on every involved sites	**Replication** Andy creates identical resources on different sites

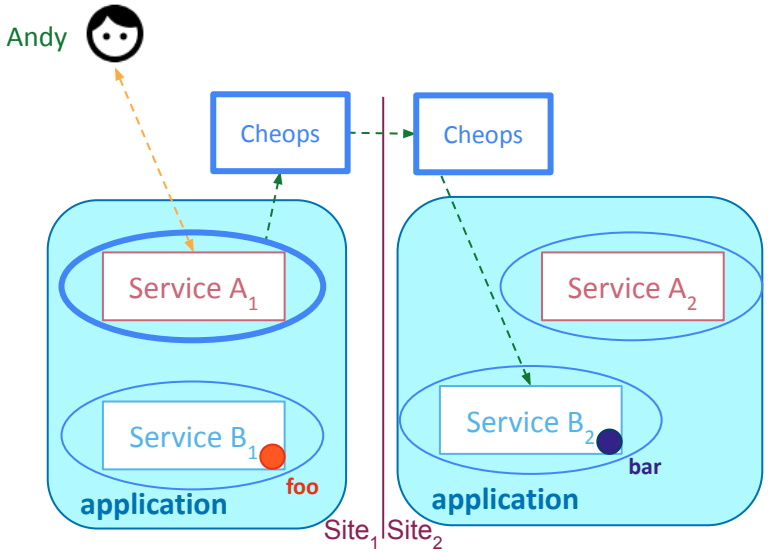
[1] Euro-Par 2021

Sharing[1]

Service B from Site2 has the **required resource (bar)**

Andy defines the scope of the request into the CLI. The scope specifies **where** the request applies.

```
application create resourceA
                  --sub-resourceB bar
                  --scope { serviceA: Site1 ,
                            serviceB: Site2 }
```



Sharing has been validated on OpenStack

```
openstack server create [...] myvm --scope {Nova: Berlin, Glance: Paris}
```

[1] Euro-Par 2021

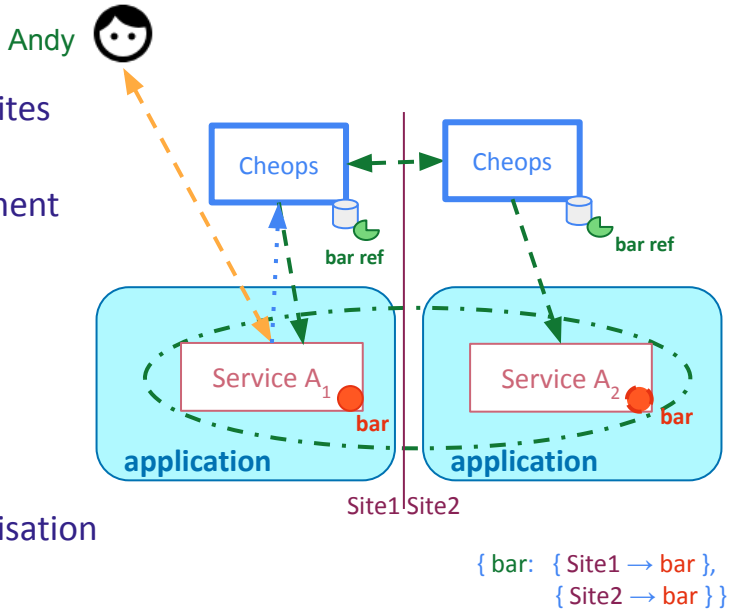
Cross[1]

Cross creates the illusion of a **Single Resource** across the involved sites

Initial proposal made by Sarmiento et al.[2], with intrusive development efforts made for Openstack

- Cross maintains the availability of the resource
- Cross is based on **Aggregation** & **Divisibility** principles

Currently still a work in progress by Geo Johns Antony for a generalisation



[1] **ICSOC 2022**
[2] Multi-site Connectivity for Edge Infrastructures : DIMINET: DIstributed Module for Inter-site NETworking

Replication[1,2]: a theory

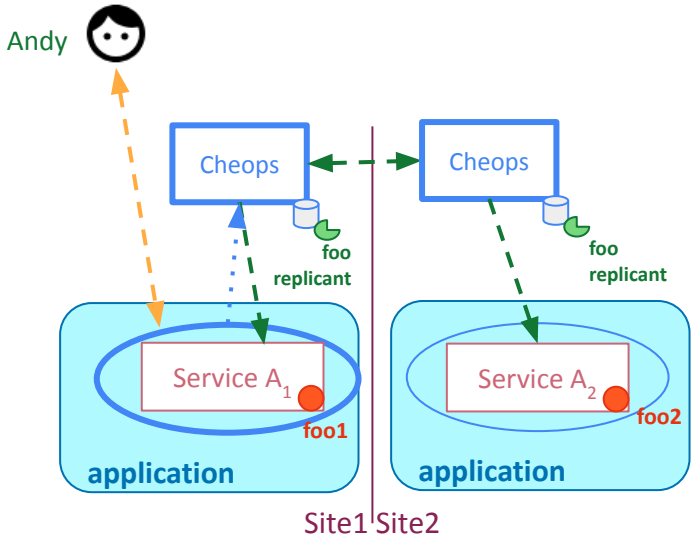
Andy defines in the scope **where the resource will be replicated**
The resource (managed by Service A) will be created on both sites

```
application create foo --scope { serviceA: Site1 & Site2 }
```

The goals of the replication are:

- **lowering latency** when using a resource close
- **increasing robustness** towards partition by allowing the use of the resource locally

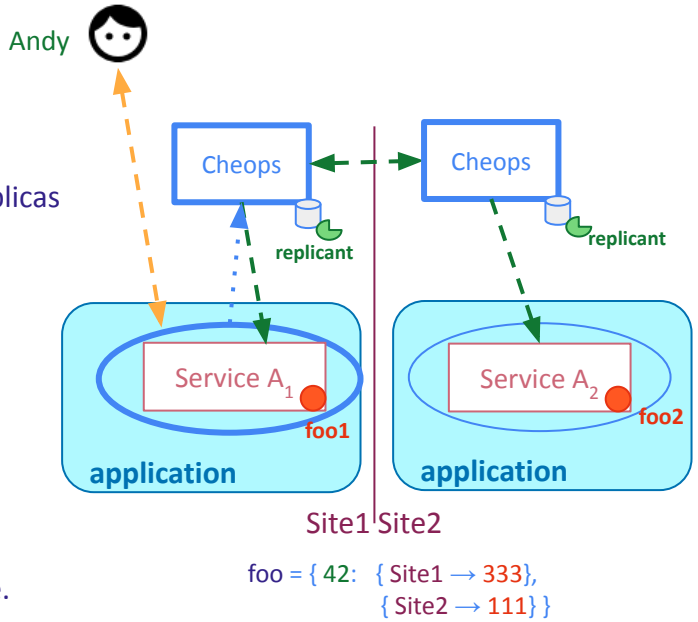
Replicas information are stored in a **replicant**, associating the resource to **the location of the replicas** and **how to find them locally**.



[1] Euro-Par 2021
[2] XP 2021 Workshops

A view of the replica model: replicants

- These replicants store:
  - a meta ID (identifiant), generated by Cheops to keep track of the replicas
  - each site where the replicas are located
  - for each site, the local ID to find the replica when needed

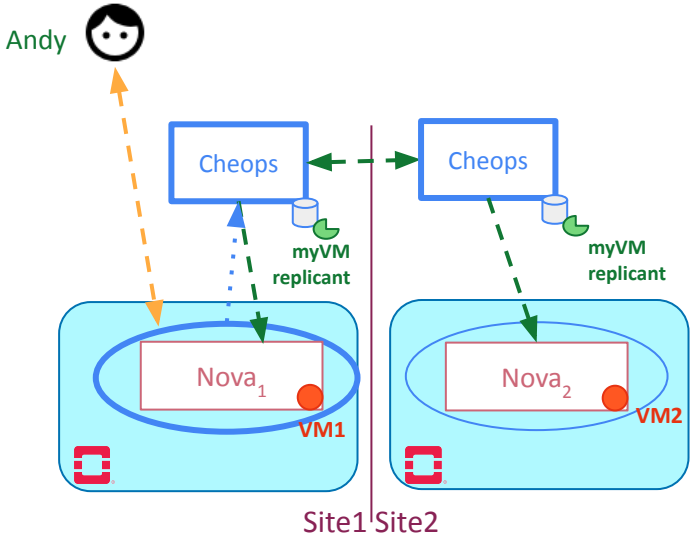


- Example:
  - Andy wants two replicas of foo on Site1 and Site2
  - Site1 creates the resource and assign 333 as local ID to this resource.
    Site2 does the same with the ID 111.
  - Cheops generates a meta ID: 42
  - The replicant is then: { 42: { Site1: 333, Site2: 111 } }

Replication, in the flesh

```
openstack server create myVM --scope { Nova: Site1 & Site2 }
```

- Replication follows an **eventual consistency** (Raft)
- Consistency is maintained **through the API** (at the CRUD level, resources are considered as black boxes)



Like for sharing, resources are considered as black boxes for **genericity**

The premises for an implementation of Cheops

Our specification/requirements for the execution:

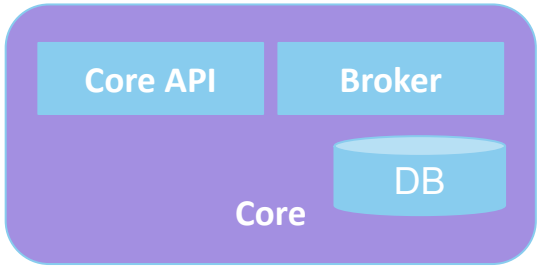
- Follows the **requirements** defined in the State-of-the-Art
  - Non-intrusive
  - Generic
  - Local-first
  - Collaborative-then
  - Network partitions tolerance
  - Dynamic
  - On-demand
  - Decentralized and P2P
- **Modular**
- Ability to integrate **scope-lang** and the **collaborations**

Primary targets: OpenStack and Kubernetes

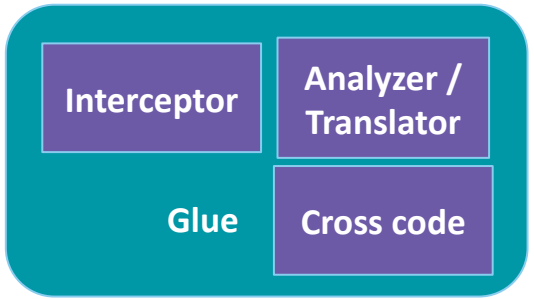

Our solution: Cheops

Cheops divided into 2 major modules:

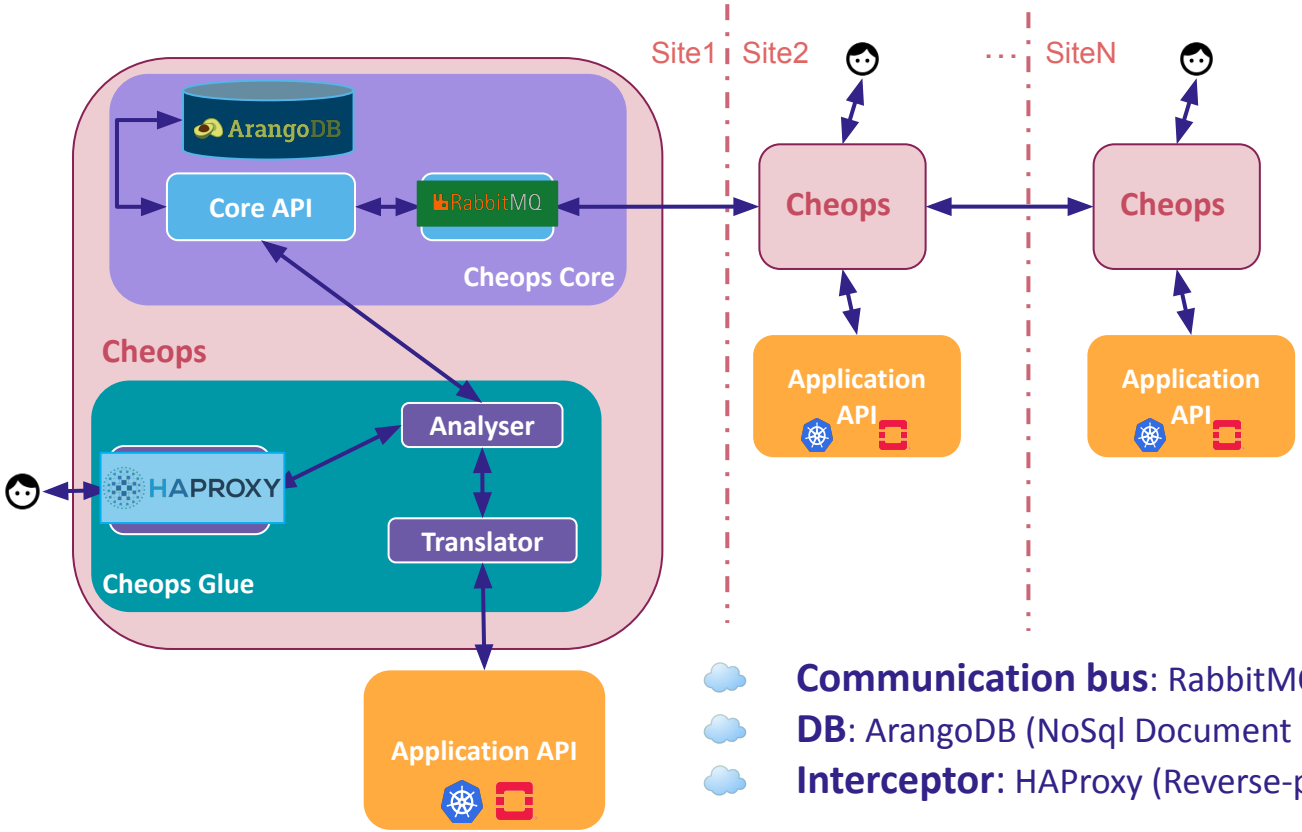
- **Cheops Core:**
  - Management layer responsible for the deployed application metadata & P2P interactions
  - Generic across applications



- **Cheops Glue:**
  - Intermediate layer responsible for translating interactions with the application
  - Dependent to an application

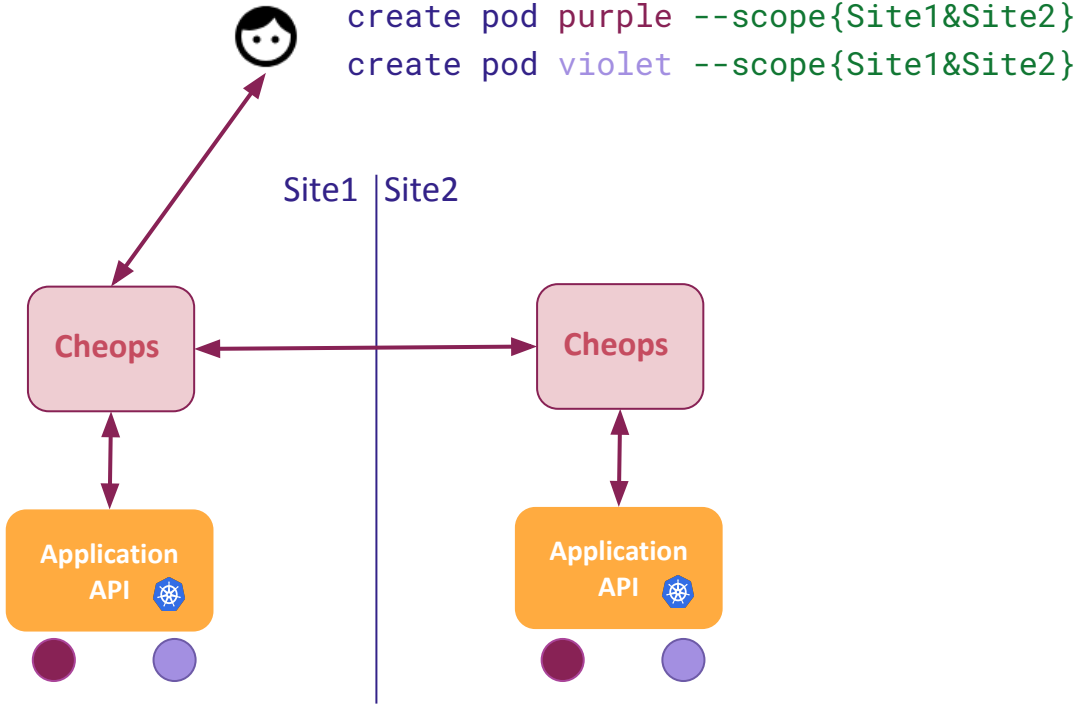


Cheops architecture



- **Communication bus**: RabbitMQ (P2P AMQP)
- **DB**: ArangoDB (NoSql Document based)
- **Interceptor**: HAProxy (Reverse-proxy for request capture)

Sandbox experiments on Kubernetes



Note: Sharing does not make sense on Kubernetes because it only has one service

Sandbox experiments on Kubernetes

Operation	Location	Result
`kubectl create pod purple --scope{Site1&Site2}`	Site1	Pod purple created on Site1 and Site2
`kubectl create pod violet --scope{Site1&Site2}`	Site2	Pod violet created on Site1 and Site2
`kubectl get pod violet`	Site1	Pod violet from Site1 is displayed
`kubectl get pod violet`	Site2	Pod violet from Site2 is displayed

Note: ongoing experimentations (scalability, network partitions, ...)

Resource dependencies Model

A Generic Pattern

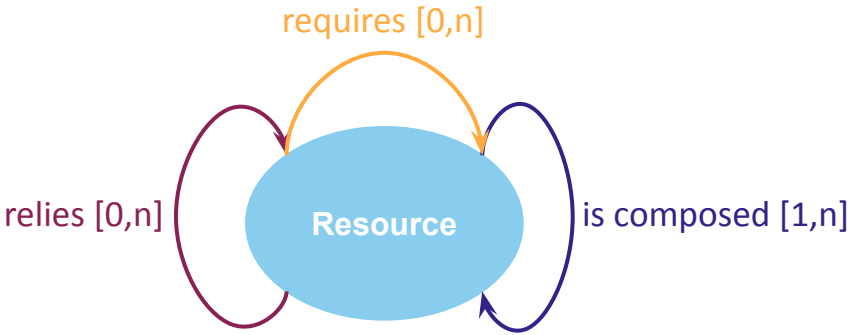
Services relies on resources like VMs, pods, services, networks, etc. that may be **interdependent.**

To geo-distribute applications in a non-intrusive manner:

- Involves handling resources dependencies between these micro-services
- This dependencies handling must be also generic

Can we find a generic pattern from multiple applications to manage these **dependencies**?

Patterns in dependencies[1]



Example:

A vm **requires** an image

Example:

A vm **relies** on an IP

Example:

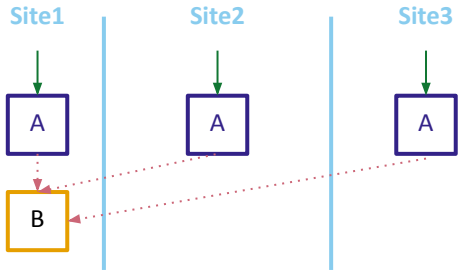
A heat stack **is composed of** vms

[1] **ICSOC 2022**

How to manage dependencies - Replication

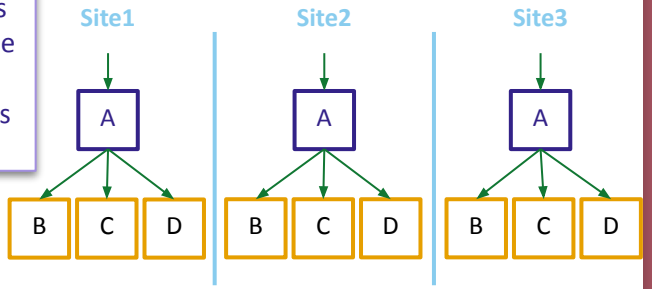
REQUIREMENT

Replicate A everywhere using sharing (with B on Site1)

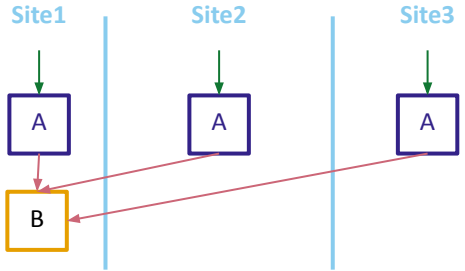


COMPOSITION

Replicate A everywhere, as an elementary resource, the application will create the sub-resource in the process



RELIANCE



Ideally, replicate B everywhere where A will be

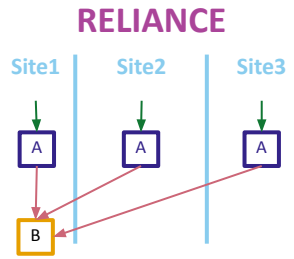
Store the dependency information on B to warn against/prevent deletion

creation

sharing (requirement / reliance)

!! Some resources are only valid in a specific context !!

Invalid collaborations



Ideally, replicate B everywhere where A will be

Store the dependency information on B to warn against/prevent deletion

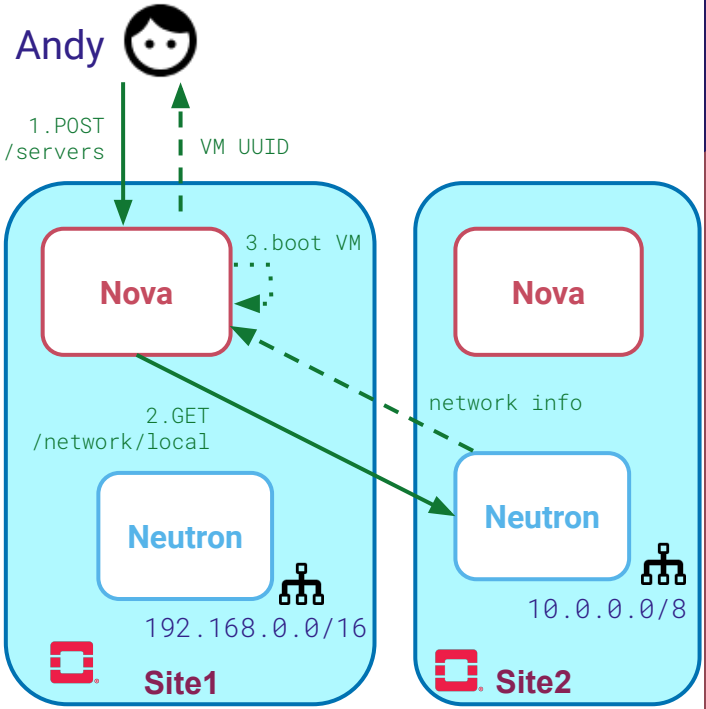
!! A vm relies on an IP !!

```
openstack server create myVM --ip myIPSite2 ??
                            --scope { Nova: Site1 & Site2,
                                      Neutron: Site2 }
```

➔ **Wrong net info** for the compute service (*e.g.,* `10.0.0.2` instead of `192.168.0.2`)

☹ the VM on Site1 **is unreachable**

Resources are valid only in a specific context



Boot VM on Site1 with a network from Site2

Conclusion and Perspectives

Conclusion on scope-lang and Cheops

Cheops/scope-lang is a Proof of concept to geo-distribute Cloud applications at the Edge. It is:

- **Non-intrusive** no change has been made to the application code
- **Generic** to applications made of services
- **Local-first** autonomous instances of the application everywhere
- **Collaborative-then** collaborations thanks to scope-lang
- **Network partitions tolerance** local-first and collaborations allows some tolerance
- **Dynamic** scope-lang applies to any fine-grained request
- **On-demand** scope-lang specifies only location on-demand, otherwise, the request is local
- **Decentralized and P2P** fully decentralized with no central authority or hierarchy

Cheops: ongoing/future work

- More **experimentations** on partitioning, scaling, tests on OpenStack
- **Dependencies management** to be formalised and **invalid collaborations** prevented with a dependency tree
- More **collaborations** (with Cross under way)

Collaborations and Scope-lang

- **Or**
  - `application resourceA create --scope { serviceA: Site1 || Site2 }`
- **Around**
  - `application resourceA create --scope { around: Site1, 50ms }`
- **&set** (or replicaset, inspired by Kubernetes)
  - `application resourceA create --scope { set: 2@Site1, 3@Site2 }`
  - managed by an autonomous control loop

Research question

Is it possible to use applications developed for the Cloud on Edge infrastructures without changing their code?

Through this Phd Defense I hope I demonstrated that it is indeed possible 😀

Publications

Conferences

- **ICSOC 2022** Delavergne, M., Antony, G.J., Lebre, A. (2022). Cheops, a Service to Blow Away Cloud Applications to the Edge. In: Troya, J., Medjahed, B., Piattini, M., Yao, L., Fernández, P., Ruiz-Cortés, A. (eds) Service-Oriented Computing. ICSOC 2022. Lecture Notes in Computer Science, vol 13740. Springer, Cham., doi: 10.1007/978-3-031-20984-0_37.
- **Euro-Par 2021** Cherrueau, RA., Delavergne, M., Lebre, A. (2021). Geo-distribute Cloud Applications at the Edge. In: Sousa, L., Roma, N., Tomás, P. (eds) Euro-Par 2021: Parallel Processing. Euro-Par 2021. Lecture Notes in Computer Science(), vol 12820. Springer, Cham., doi: 10.1007/978-3-030-85665-6_19.

Journal

- **IEEE TPDS 2022** R.-A. Cherrueau et al., "EnosLib: A Library for Experiment-Driven Research in Distributed Computing", in IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 6, pp. 1464-1477, 1 June 2022, doi: 10.1109/TPDS.2021.3111159.

Workshop

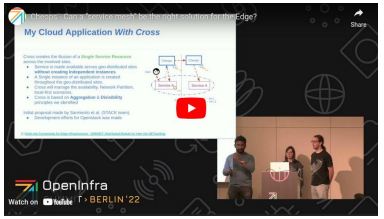
- **XP 2021 Workshops** Delavergne, M., Cherrueau, RA., Lebre, A. (2021). A Service Mesh for Collaboration Between Geo-Distributed Services: The Replication Case. In: Gregory, P., Kruchten, P. (eds) Agile Processes in Software Engineering and Extreme Programming – Workshops. XP 2021. Lecture Notes in Business Information Processing, vol 426. Springer, Cham. doi: 10.1007/978-3-030-88583-0_17.125

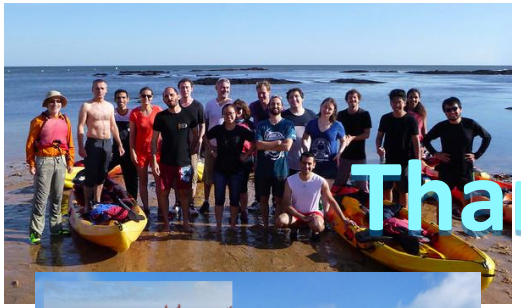
Research reports

- **RR 2022** Marie Delavergne, Geo Johns Antony, Adrien Lebre. Cheops, a service to blow away Cloud applications to the Edge. [Research Report] RR-9486, Inria Rennes - Bretagne Atlantique. 2022, pp.1-16. ⟨hal-03770492v2⟩.
- **RR 2020** Ronan-Alexandre Cherrueau, Marie Delavergne, Adrien Lebre, Javier Rojas Balderrama, Matthieu Simonin. Edge Computing Resource Management System: Two Years Later!. [Research Report] RR-9336, Inria Rennes Bretagne Atlantique. 2020. ⟨hal-02527366v2⟩.

Outside of the scientific community

- **OpenInfra Summit 2022** Geo Johns Antony, Marie Delavergne and Baptiste Jonglez. Cheops - Can a "service mesh" be the right solution for the Edge?, https://www.youtube.com/watch?v=7EZ63DMRJhc, OpenInfra Summit 2022, Berlin





Thanks for your attention!