A service mesh for collaboration between geo-distributed services: the replication case

Marie Delavergne under the supervision of
Adrien Lebre and **Ronan-Alexandre Cherrueau**

STACK

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Inria
inventeurs du monde numérique

LS2N
Laboratoire des sciences du numérique de Nantes

AMP 2021

Context

Existing cloud apps are going to the Edge

- **What is the Edge?**
    - Bring compute and storage equipments as close as possible to discrete data sources, physical elements or end users (to deal with latencies)
    - Limited network connections with disconnections between (edge) sites
- **How to deal with disconnections?**
    - Distribute cloud application instances on every involved sites
    - Each instance works **autonomously**, but should be able to ***collaborate*** with others when needed

Existing cloud apps are going to the Edge

- **Unfortunately, most Cloud applications do not follow this design**

- **Intrusive modifications, when possible, are tedious** [1,2]
    - Thousands of LoCs: ShareLatex, Kubernetes, …, OpenStack

⇒ We do not want to change their code

[1] Revising OpenStack to Operate Fog/Edge Computing infrastructures https://hal.inria.fr/hal-01273427
[2] ShareLatex on the Edge [...] https://dl.acm.org/doi/10.1145/3286685.3286687

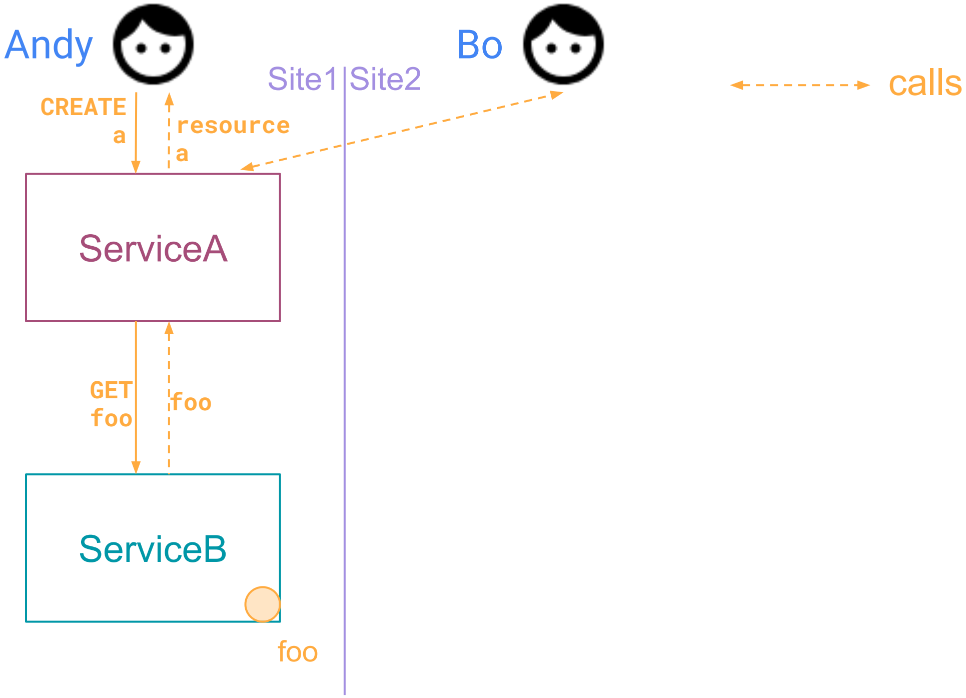
Problem:

How can we use Cloud Applications in Edge Infrastructure?

My Cloud application

Andy and Bo use the same application, even though Bo is far

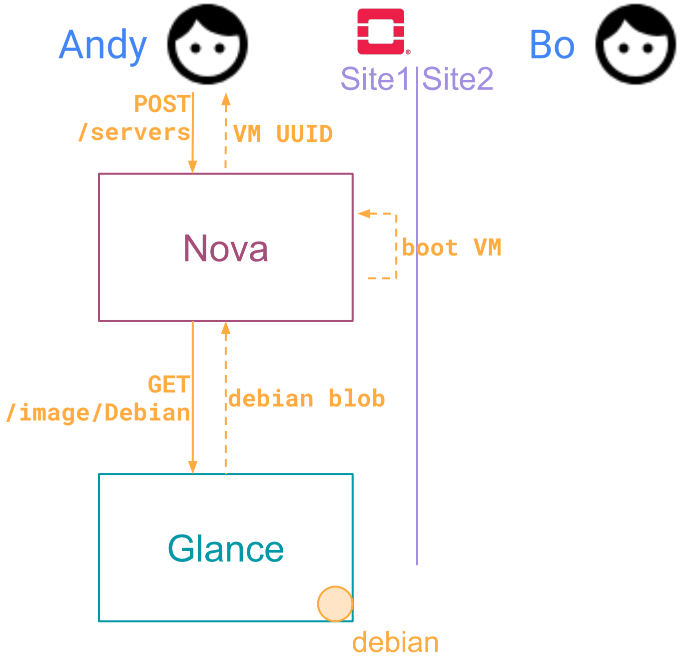
```
resourceA a = application
              resourceA create
              --sub-resourceB foo
```



My Cloud application, example: OpenStack

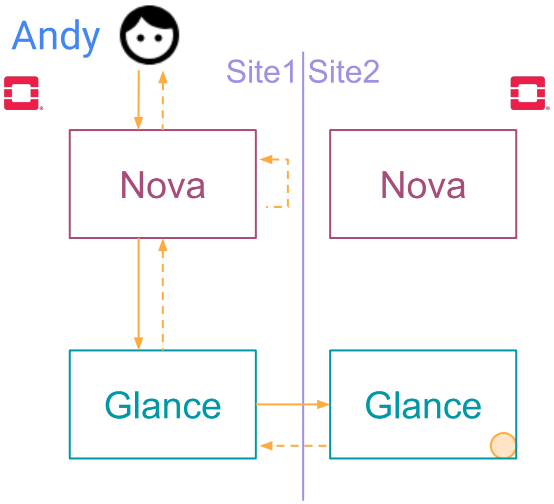
Andy and Bo use the same Openstack, even though Bo is far

```
server a = openstack
           server create my-vm
           --image debian
```

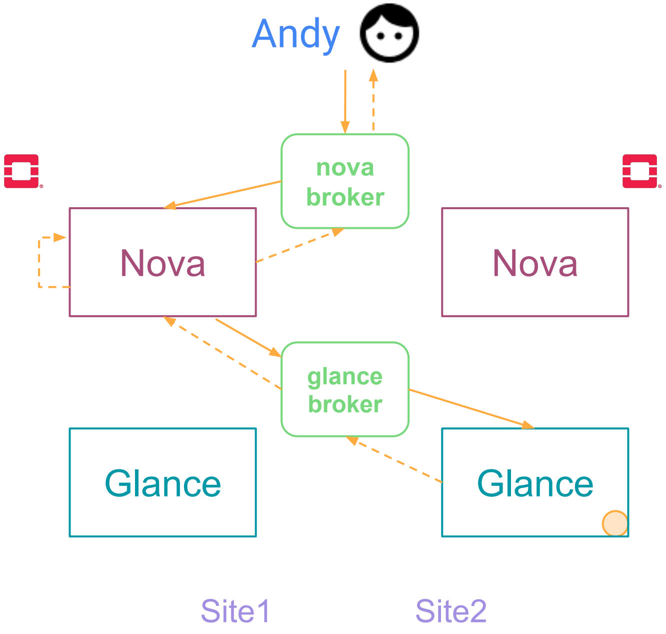


Different types of collaborations

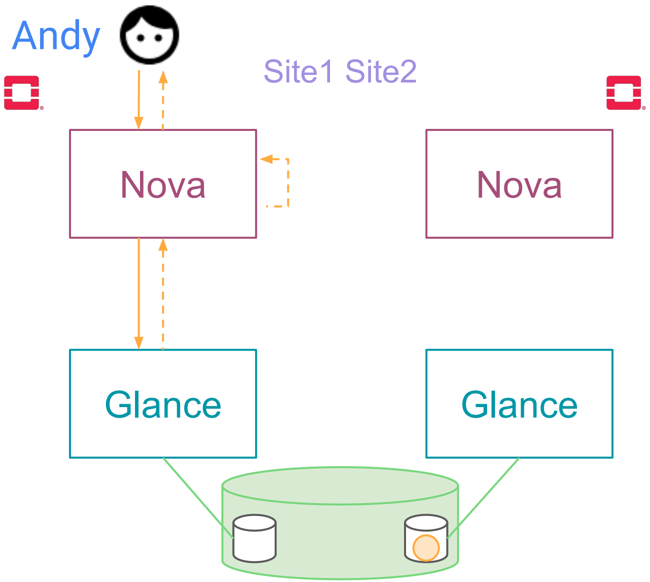
Service-to-service collaboration



Broker-based collaboration



Database collaboration



How to make a cloud app edge compliant?

design principles

- autonomous instances
- on-demand collaboration
- no touching the code
- generic

How to make a cloud app edge compliant with our design principles?

- ❖ genericity ?
- ❖ no touching the code ?
- ❖ autonomous instances?
- ❖ collaboration ?

- The answer lies -in part- in service-based application modularity
- Those applications are composed of services that:
  - allows separation of concerns (application domain vs deployment, monitoring, etc.)
  - are generic and can be used in other applications
  - expose an API to communicate with each other

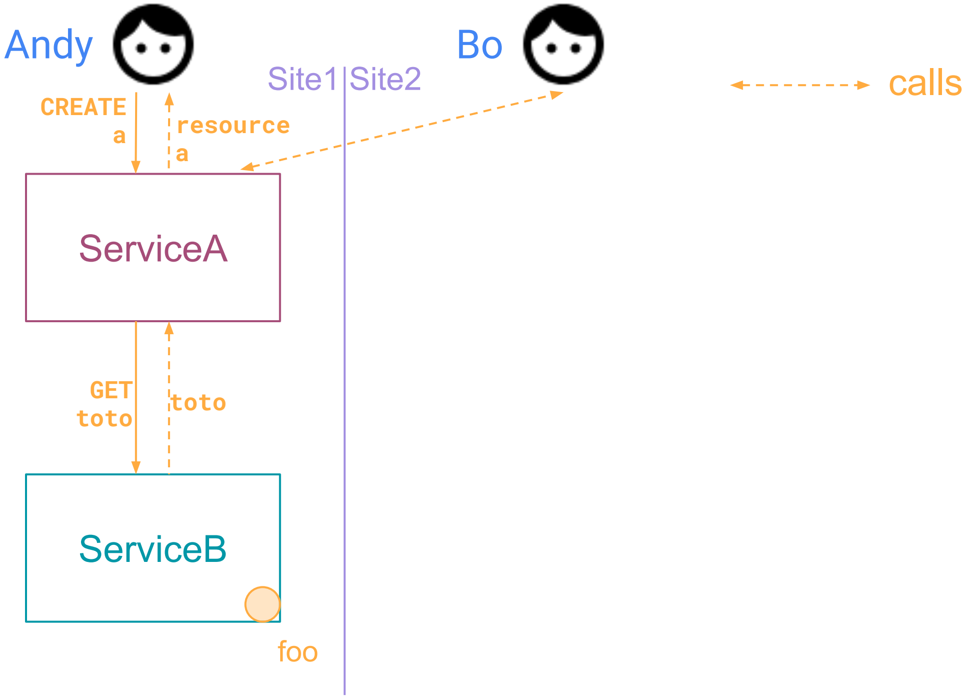
Solution

My Cloud application

- generic □
- no touching the code □
- autonomous instances ✖
- collaboration ✖

Andy and Bo use the same application, even though Bo is far

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

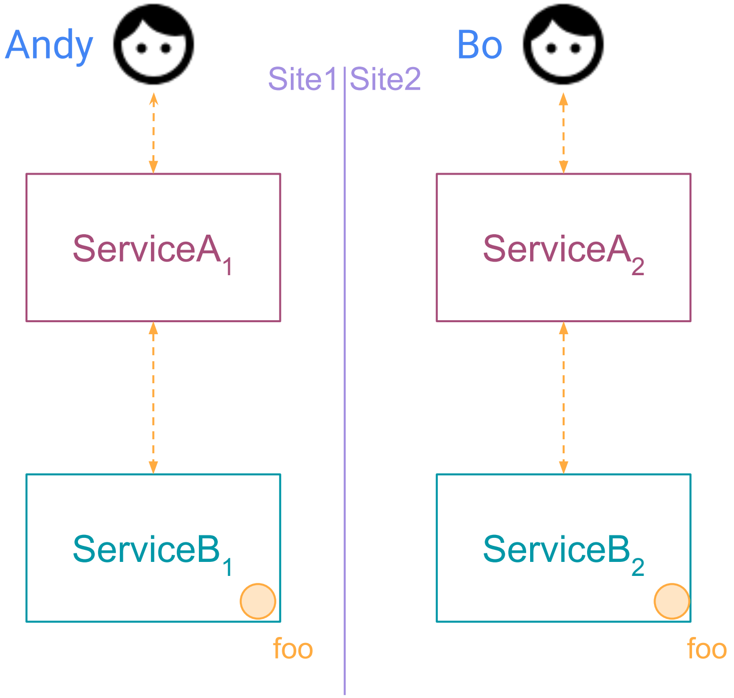


My Cloud application *instantiated everywhere*

- generic □
- no touching the code □
- autonomous instances □
- collaboration ✖

Andy and Bo use their own application, closer to them

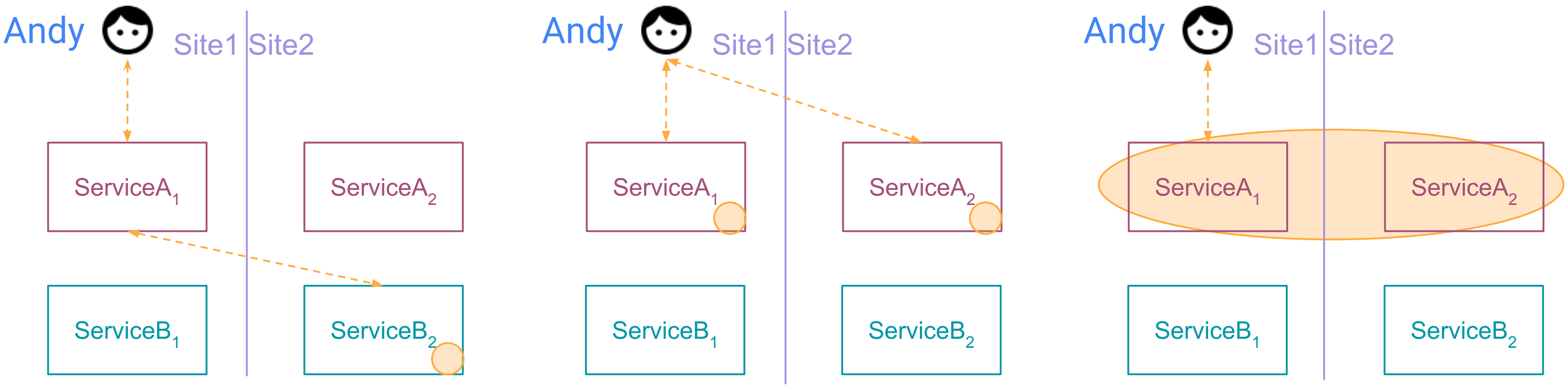
```
resourceA a = application
              serviceA create
              --sub-resourceB foo
```



Focus on 3 collaborations

- Between services of different instances for **sharing**: ,
- Resource **replication**: &
- Resource spanning a**cross** different instances: +

- genericity
- no touching the code
- autonomous instances
- collaboration
  - sharing
  - replication
  - cross



Sharing ,:
A required resource is on another site

Replication &:
Andy creates identical resources on different sites

Cross +:
Andy creates a resource that span on every involved sites

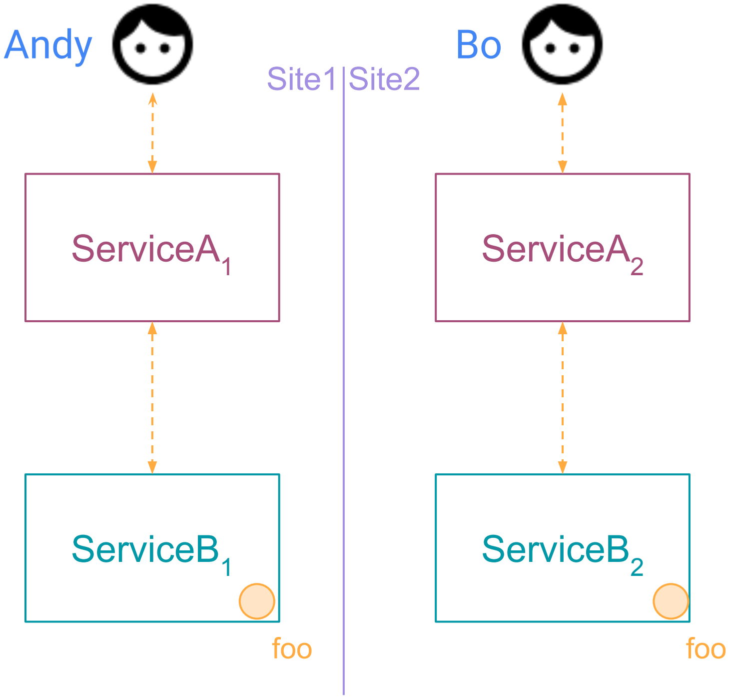
Scope-lang

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

```
resourceA a = application
              serviceA create
              --sub-resourceB foo
```

≣

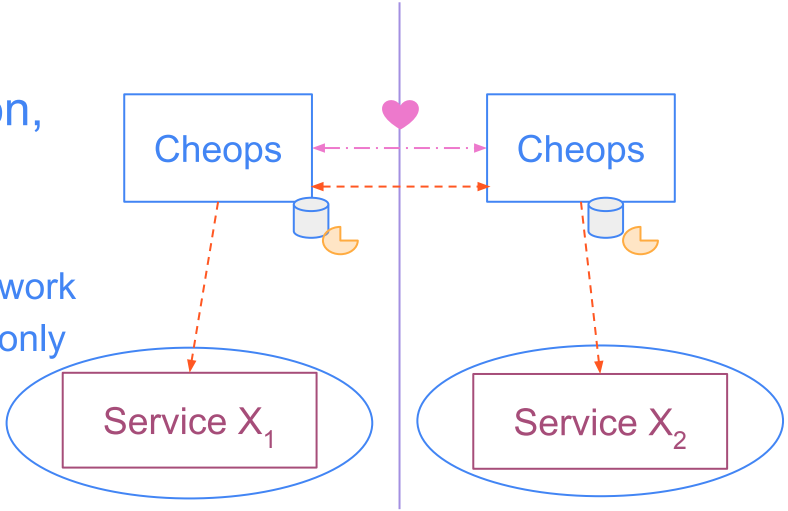
```
resourceA a = application
              serviceA create
              --sub-resourceB foo
              --scope {ServiceA: Site1,
                       ServiceB: Site1}
```



Cheops as a building block to deal with geo-distribution

- To forward requests between services
- To manage creations, updates and deletions of resources in a consistent manner on multiple sites

- Cheops is a service to manage geo-distribution, considering each resource as a black box.
  - Agents are located on each site
  - Uses heartbeat to check if sites are up and in the network
  - Uses its own database to store resource information only where relevant

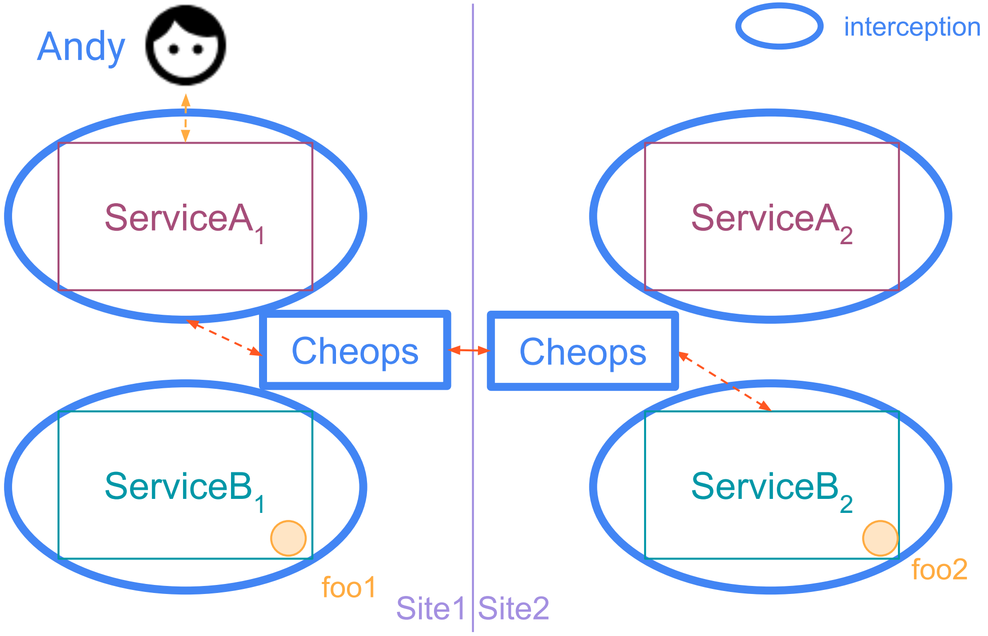


My cloud application *with sharing*

- generic □
- no touching the code □
- autonomous instances □
- collaboration
  - sharing □
  - replication ✖

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

```
resourceA a = application
              resourceA create
              --sub-resourceB foo2
              --scope { serviceA: Site1 ,
                        serviceB: Site2 }
```



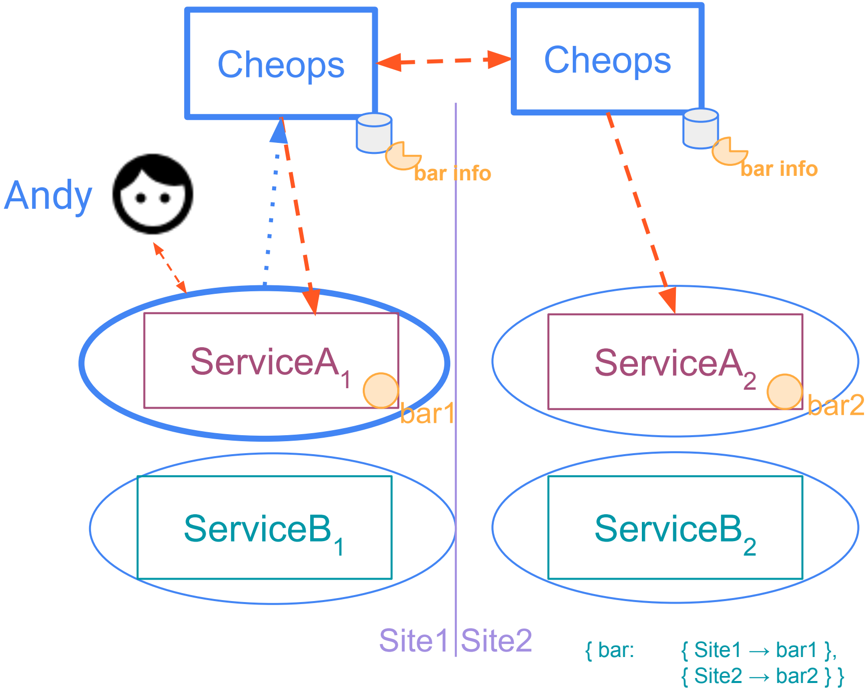
Replication

My cloud application *with replication*

- generic
- no touching the code
- autonomous instances
- collaboration
  - sharing
  - replication ?

Andy defines the scope of the request into the CLI. She defines that the resource (managed by Service A) will be created on both sites.

```
ressourceA bar  = application
                  resourceA create
                  --scope { serviceA: Site1 &
                                      Site2 }
```



- Stores only generic information about the resources (e.g. its unique id, where is it located, information to retrieve it locally)
- The resource: {meta-uid: {site-uid: local-uid}}

Different consistencies

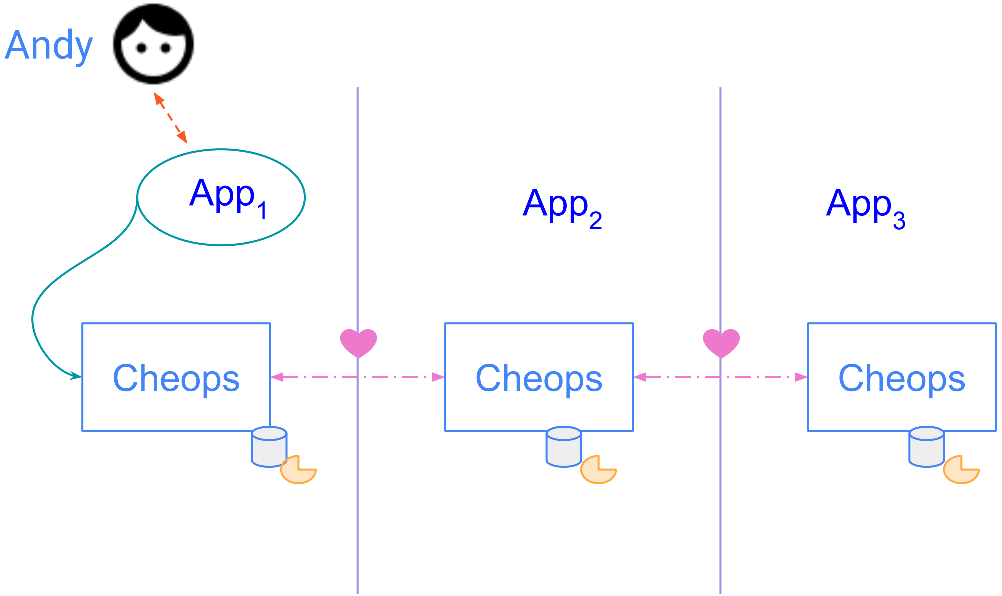
- **None**: No guarantees (operations is trigger and that's all).
- **Eventual**: every operation on a replica will be applied to the others eventually.
- **Transactional Eventual**: either with two phases commit or long-lived transactions, depending on the resources involved. Ensures transactions while still being available. (cf Cure[1] and Sagas[2])
- **Strong Serializable**: strongest consistency, but the system might be unavailable a lot.

→ **this is currently the focus**

Requires transactions

1: https://pages.lip6.fr/Marc.Shapiro/papers/Cure-final-ICDCS16.pdf
2: http://www.amundsen.com/downloads/sagas.pdf

Eventual consistency with Raft



- The replicant where Cheops intercepted the creation request becomes the leader for this resource.
- It stores a log of the updates made to the replicas.
- Its Cheops is in charge of trying to apply the updates on all replicas.
- When a replica is separated from the quorum, it works only in read mode

Going further

Cheops, for a fine-grained control

- **Vanilla request**
  - `openstack server create my-vm --image debian`
- **The same, with scope**
  - `openstack server create my-vm --image debian --scope { Nova: Site1, Glance: Site1 }`
- **Sharing**
  - `openstack server create my-vm --image debian --scope { Nova: Site1, Glance: Site2 }`
- **Replication with eventual consistency**
  - `openstack image create debian --file ./debian .qcow2 --scope {Nova: Site1 & Site2}`
- Extend to any kind of multi-sites operations
  - **otherwise** operator, **around** operator
    - `server create --scope { Nova: Site1 ; Site2 }`
    - `server create --scope { Nova: around(Site1, 10ms) }`

Thanks for your attention!