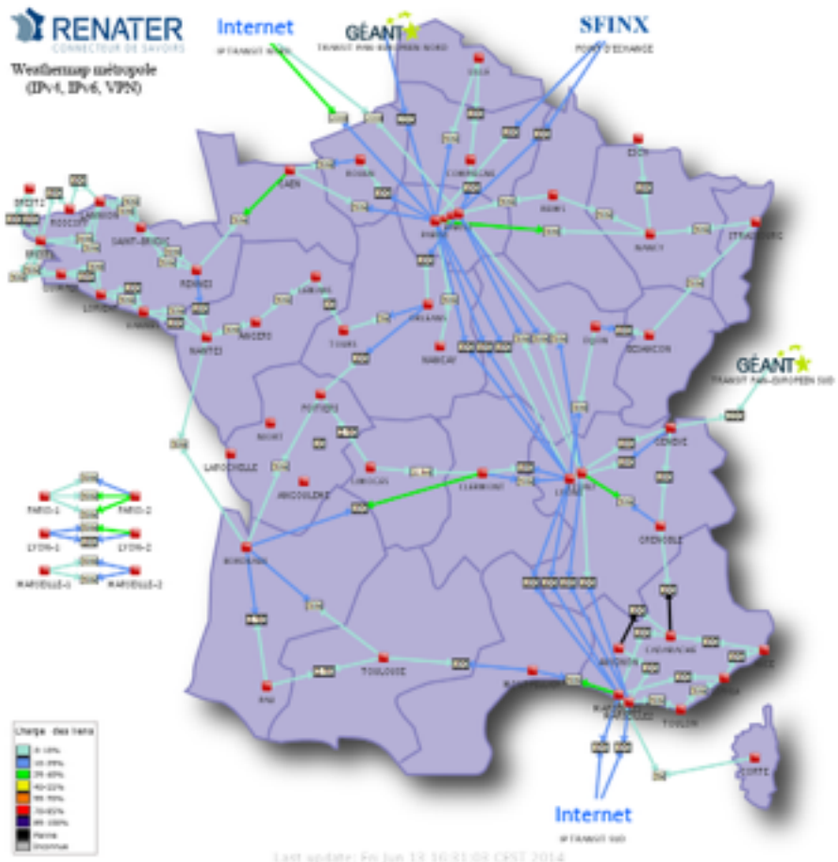
Distributing OpenStack

To build geographically distributed clouds

Jonathan Pastor
jonathan.pastor@inria.fr

Context

- Cloud computing has become very popular.
- Ever-increasing demand => ever-increasing infrastructure size.
- PB: scalability, reliability, energy but also security, juridiction and network overhead.
- Decentralise the production of computing ressources (Discovery project, http://beyondtheclouds.github.io/).

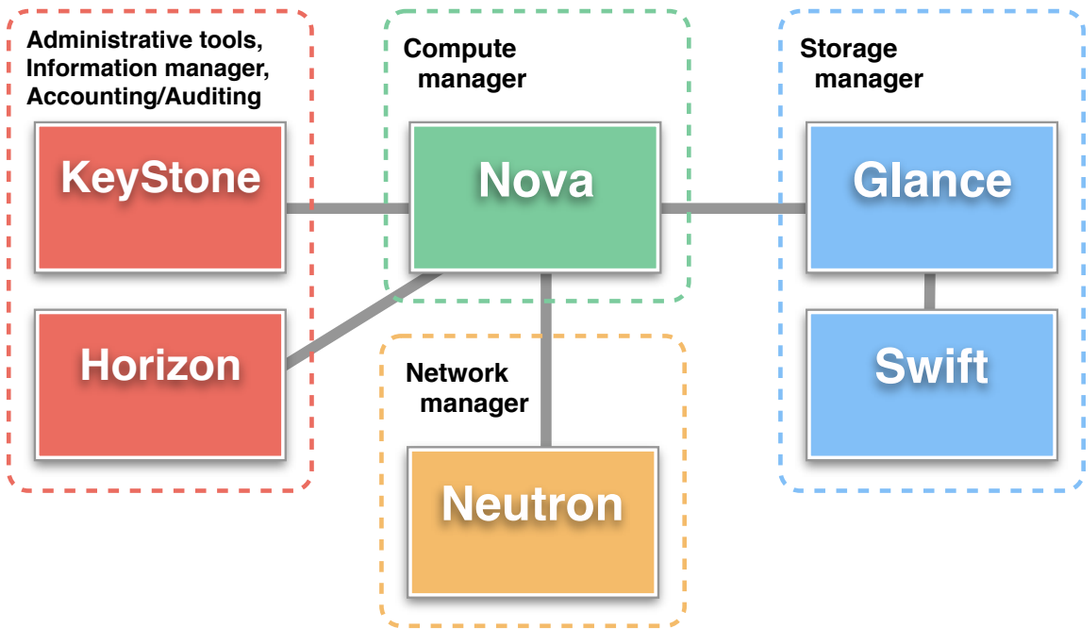


LUC-OS

- Locality Based Utility Computing OS (LUC-OS):
- A **fully distributed** Cloud-OS that enables to use and operate a massively distributed infrastructure at WAN scale, leveraging **locality properties** in order to organise efficient **cooperations**.
- To address fault tolerance and energy concerns.
- To address the network overhead, micro/nano DCs will be located on ISP point of presence.

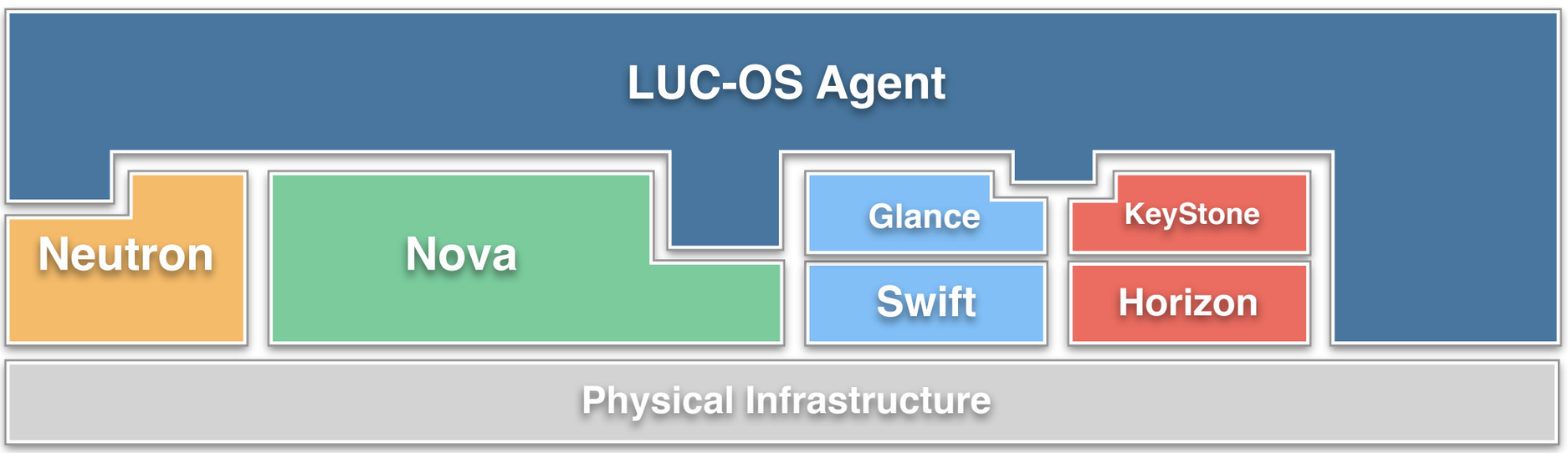
OpenStack

- OpenStack is an IaaS manager of choice.
- It leverages an architecture composed of several services.
- Nova is the controlling service of OpenStack.



Designing the LUC-OS on top of OpenStack

- The LUC-OS will rely on a multi-agent architecture.
- Some services of the LUC-OS may entirely reuse implementation from OpenStack (***Swift***).
  https://www.swiftstack.com/docs/admin/cluster_management/regions.html
- Some services will “adapt” OpenStack to the LUC-OS (***Nova***).



How to distribute OpenStack?

- We focus on distributing Nova

- Several ways to reach this objective:

1) Leverage Classic HA deployment to build distributed Clouds (Standard).

2) Using Cells (Cern).

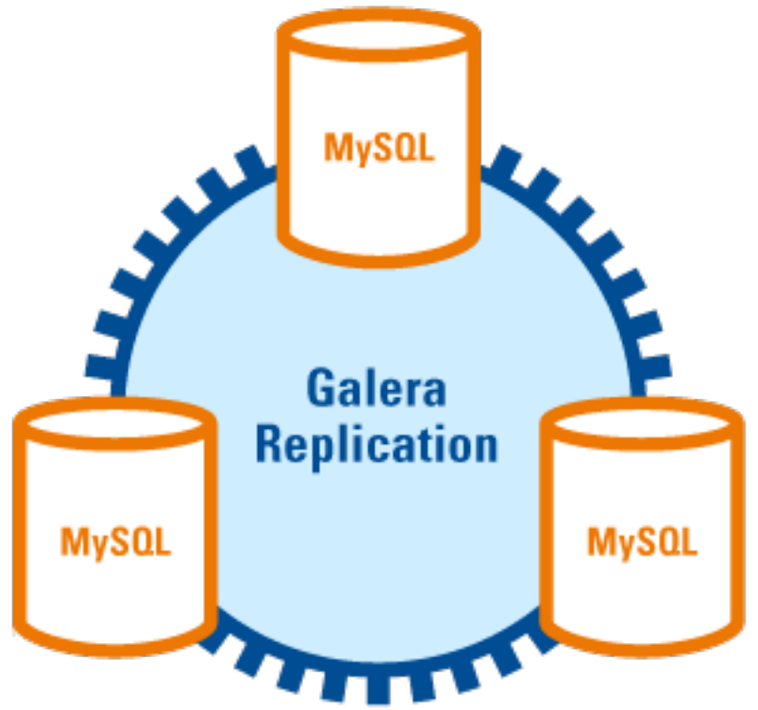
3) Replacing MySQL.

Classic HA deployment

- Nova uses relational databases as local backend (sqlalchemy).

- Database is replicated and synchronised on each nova controller nodes.

- Proposal: Use HA and deploy at least one node controller per site.



https://github.com/madkiss/openstackinaction4

Classic HA deployment

+

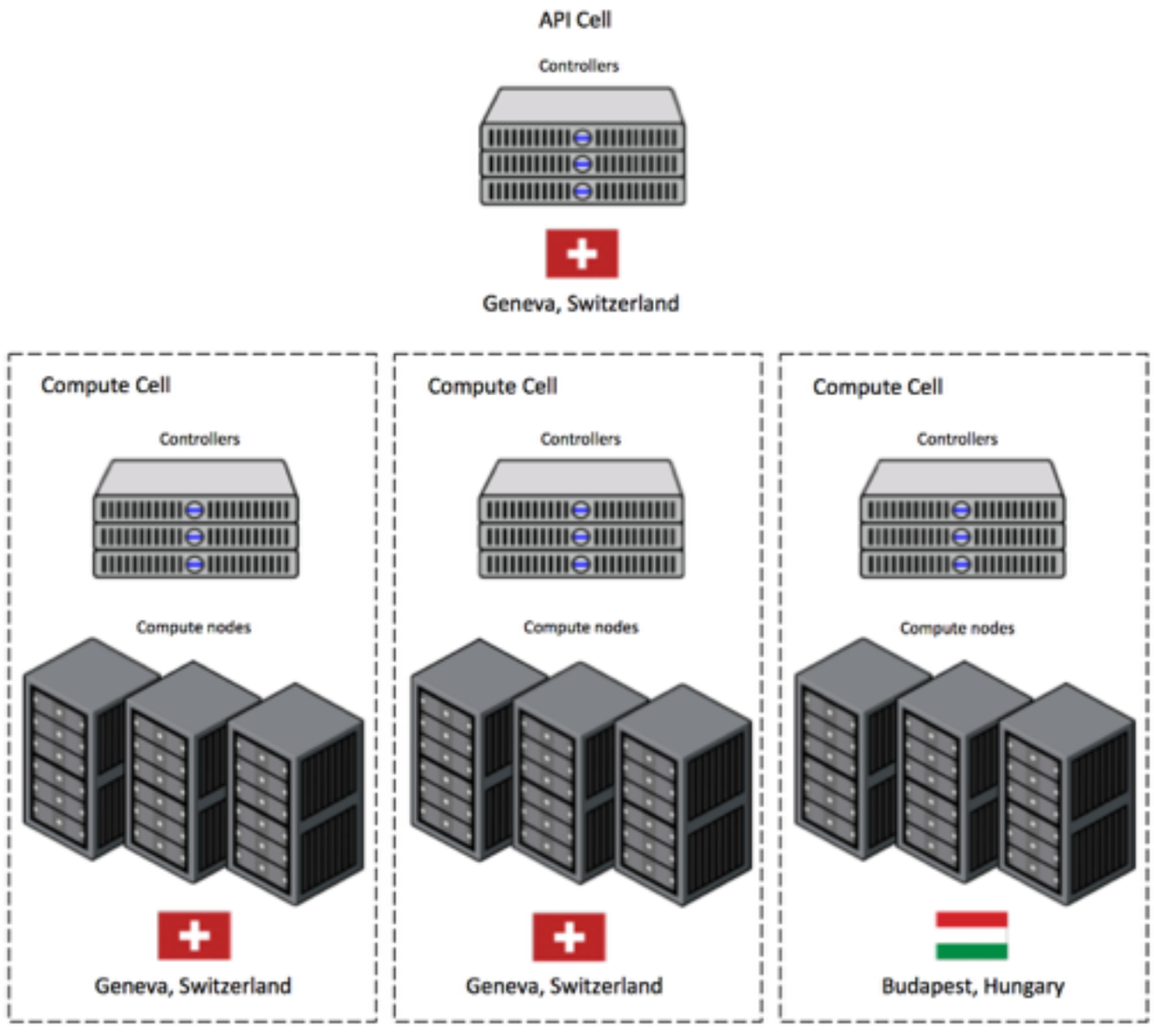
- Already working (with few controller nodes)

-

- Scalability:
  “**n controller nodes** means that for each write, there will be **n-1 writes** of other nodes”

Using Cells (Cern)

- Hierachical structure: a top cell contains nova-api and children cells contains remaining services including MySQL.
- Already used in Cern infrastructure:
    - *50 000 cores*
    - *3 different sites*
- http://openstack-in-production.blogspot.fr/2014/03/cern-cloud-architecture-update-for.html



Using Cells (Cern)

- It is possible to build the LUC-OS (flat infrastructure) on top of cells:

- **n servers** means **1 top cell**, and **n-1 cells that contains nova-controller and nova-compute**.

- It doesn't fit with the LUC-OS needs: top cell is a centralised (scalability, SPOF, …).

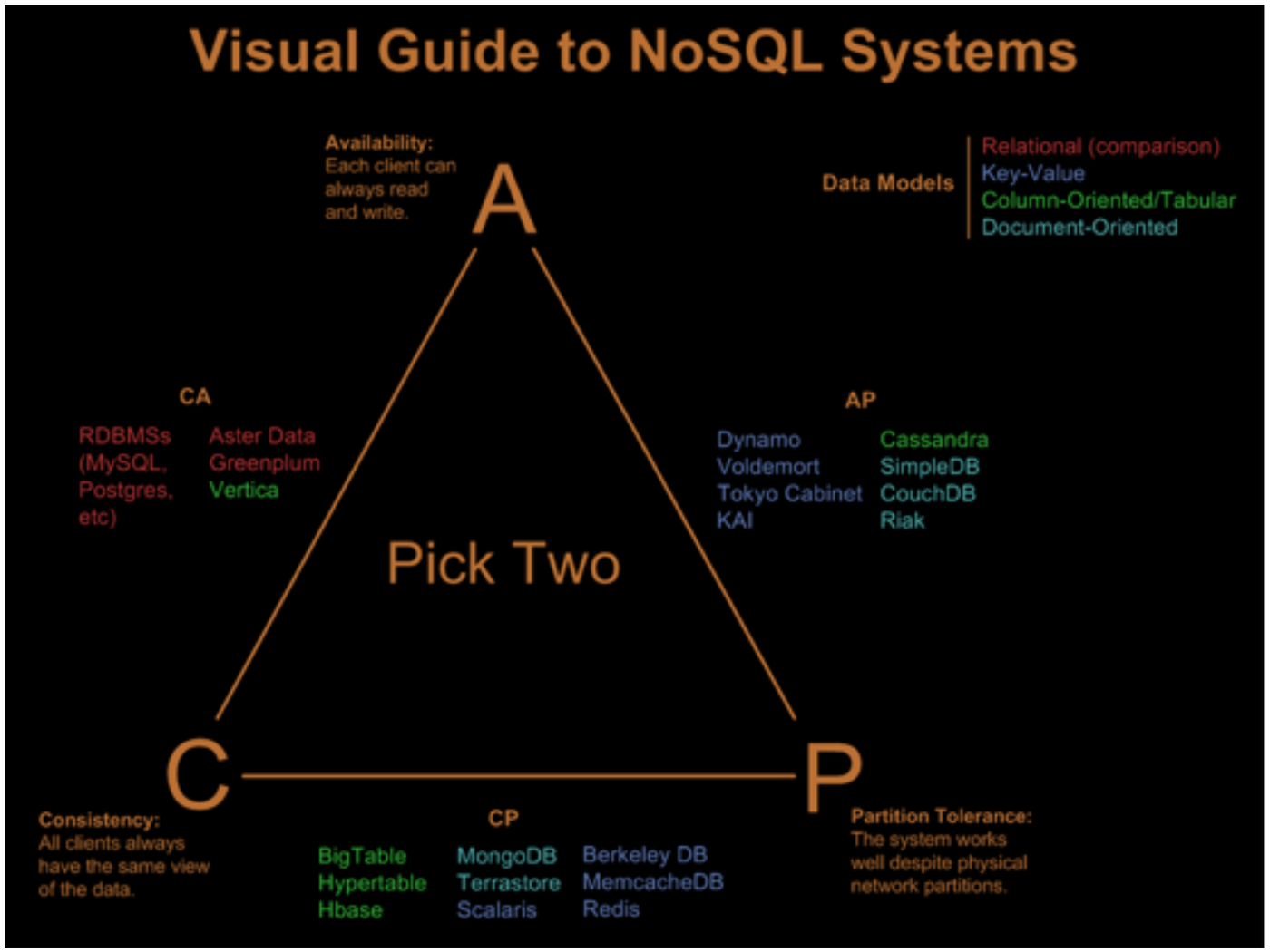
- Proposal: distributing the top cell and cells become the pivot between the LUC-OS and cloud infrastructure

Using Cells (Cern)

- ***Drawbracks:***
  - Require to modify the code of top cell (fork of OpenStack cells and follow each modification).
- ***Long term:***
  - Require to extend the protocol of cells each time we add capabilities to Nova compute (LUC-OS).

Replacing MySQL

- MySQL + Galera: Availability and consistency.
- Do we need consistency or eventual consistency?
- Replacement of MySQL by a NoSQL solution.



Replacing MySQL

- Nova services doesn't access directly the database: they calls functions located in: **nova/nova/db/api.py**

- These function manipulates databases and return python objects from:

```
/Users/jonathan/vm3/stack/nova/nova/objects
__init__.py             external_event.py     instance_fault.py      quotas.py
aggregate.py            fields.py             instance_group.py      security_group.py
base.py                 fixed_ip.py           instance_info_cache.py security_group_rule.py
block_device.py         flavor.py             keypair.py             service.py
compute_node.py         floating_ip.py        migration.py           virtual_interface.py
dns_domain.py           instance.py           network.py
ec2.py                  instance_action.py    pci_device.py
```

Replacing MySQL

- Nova already include the possibility of using a different db backend (only sqlalchemy yet).

- **nova/nova/db/api.py** => provides functions that manipulates the database backend

```
def instance_get(context, instance_id, columns_to_join=None):
    """Get an instance or raise if it does not exist."""
    print "Hello from ->instance_get"
    return IMPL.instance_get(context, instance_id,
                             columns_to_join=columns_to_join)
```

- It means that we can add a new backend:

```
_BACKEND_MAPPING = {
    'sqlalchemy': 'nova.db.sqlalchemy.api',
    'discovery': 'nova.db.discovery.api'
}
```

Replacing MySQL

- We propose to implement all functions contained in **nova/nova/db/api.py** to use a NoSQL database such as Riak, Scalaris or Cassandra.

- As current implementation (sqlalchemy) stores object with relation, we have to keep these relations with the new implementation.

- This would enable to use existing HA deployment (cf slide 4) without the active replication.

Conclusion

- We propose to replace MySQL by a NoSQL solution.

- If this works with Nova, we can extend this to KeyStone.

- No dependency with the Cell concept.

Bibliography

- https://www.swiftstack.com/docs/admin/cluster_management/regions.html

- https://github.com/madkiss/openstackinaction4

- http://openstack-in-production.blogspot.fr/2014/03/cern-cloud-architecture-update-for.html