CHULA ENGINEERING
Foundation toward Innovation

dianahep

Investigation in Query System Framework for HEP

Query engine for distributed system

By Thanat Jatuphattharachat ,Computer Engineering Chulalongkorn University

Supervisor: David Lange and Jim Pivarski, DIANA-HEP

CONTENT OVERVIEW

1 **Problem**
Distributed system is hard to be built from scratch

2 **Distributed query**
An overview of distributed Query Design

3 **Query Server**
How should the system server serves distributed query system

4 **3rd Party Framework**
An overview of investigated framework

5 **Design**
A distributed engine design using 3rd party framework

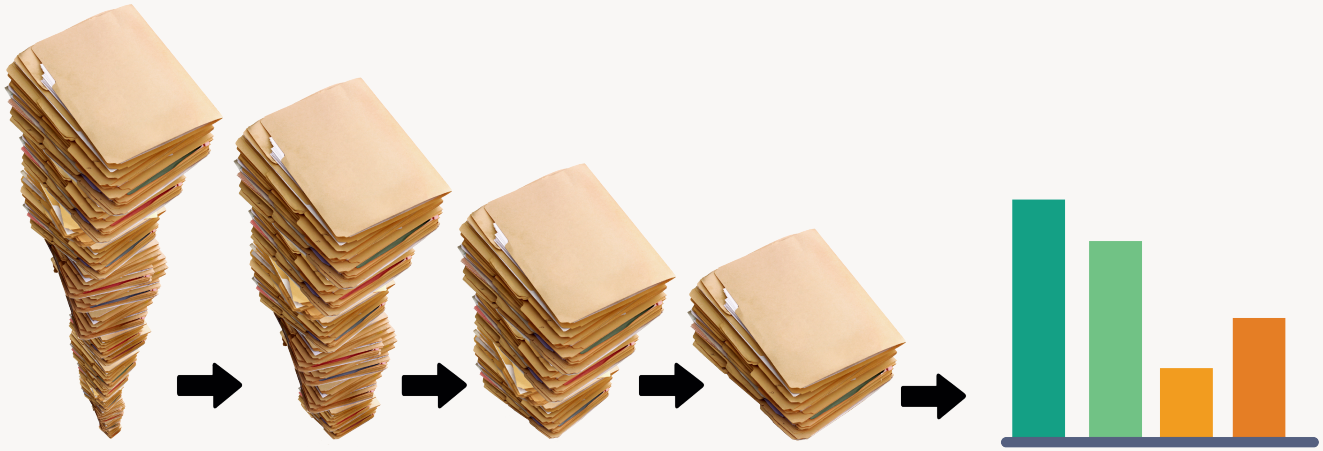
6 **Conclusion and Future**
How should the system server serves distributed query system

Problem

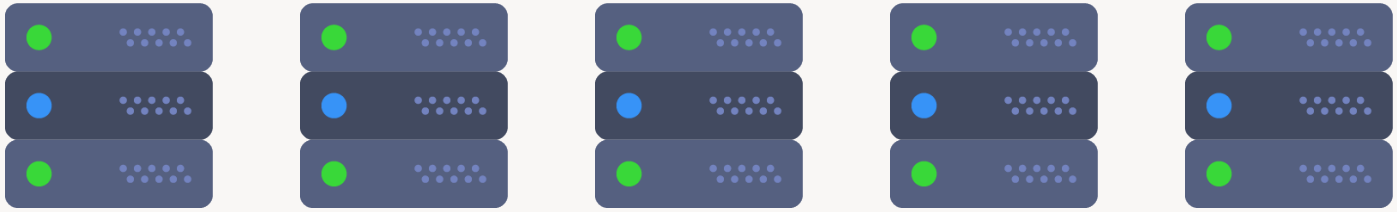
Background

femtocode provides the ability to make plots (and other aggregations for statistical analysis) directly from the collaboration's Analysis Object Data in real time

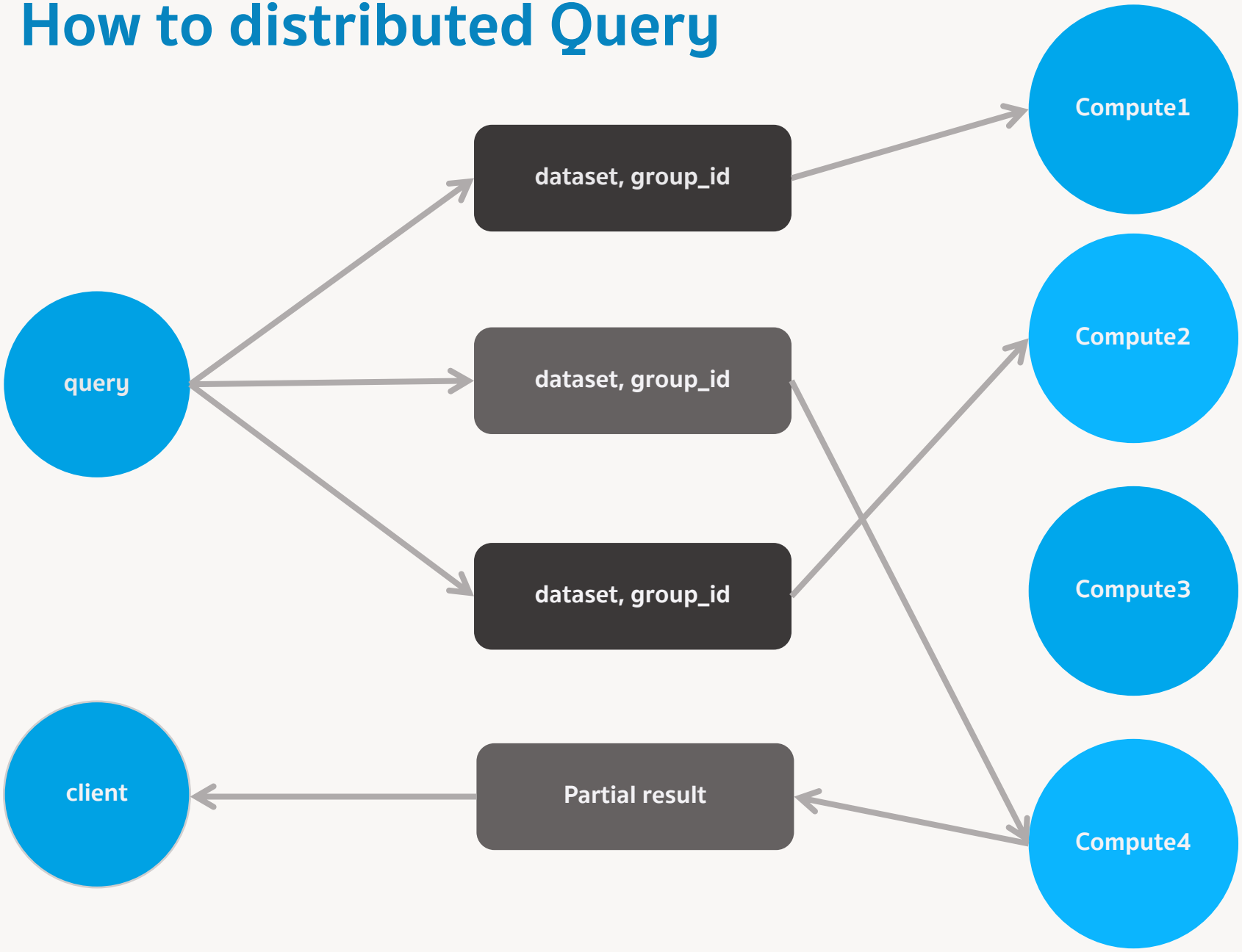
Problem



+



How to distributed Query



3rd Party Framework

Investigated Framework

Apache **Zookeeper**

Zookeeper allows distributed processes to coordinate with each other through a shared hierarchichal namespace

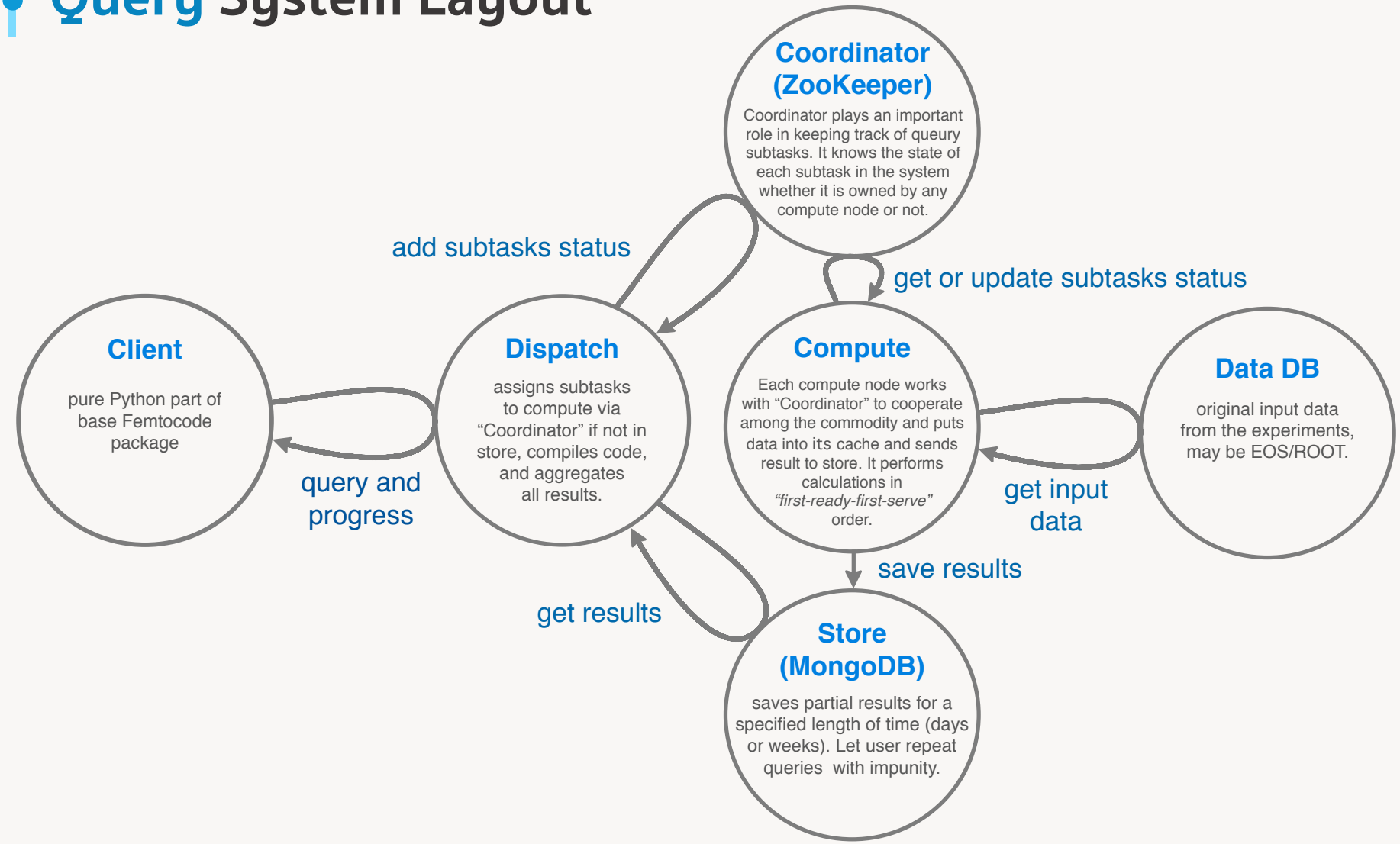
Mesos

Mesos works as a distributed system Kernel which can make us write a distributed application like a single machine application. It reduce the complexity in distributed programming at scale.

Marathon

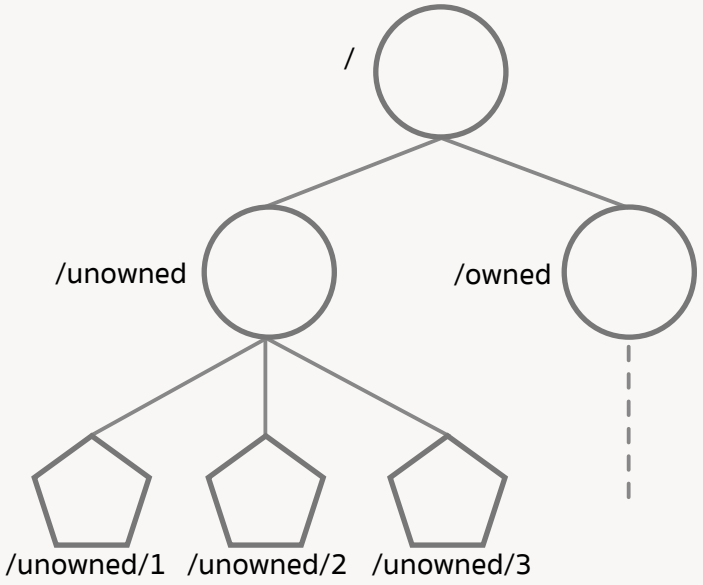
Marathon is container orchestration tool built on Mesos. it help us manage and lauch many container or application at once and let us scale it easily.

Query System Layout

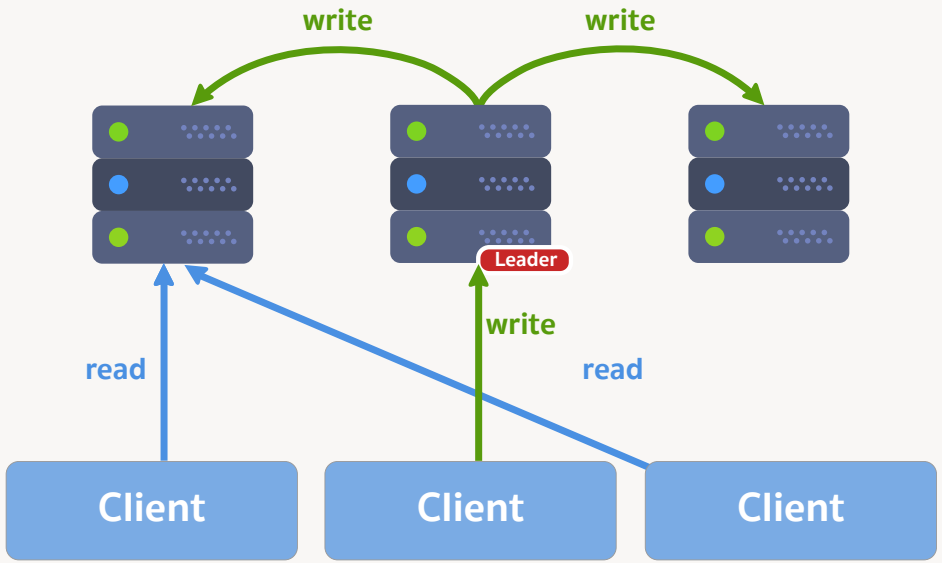


Apache Zookeeper

Apache Zookeeper File Format



Zookeeper Read and Write



Apache Zookeeper

Timeliness

The clients view of the system is guaranteed to be up-to-date within a certain time bound.

Sequential Consistency

Updates from a client will be applied in the order that they were sent.

Atomicity

Updates either succeed or fail. No partial results.

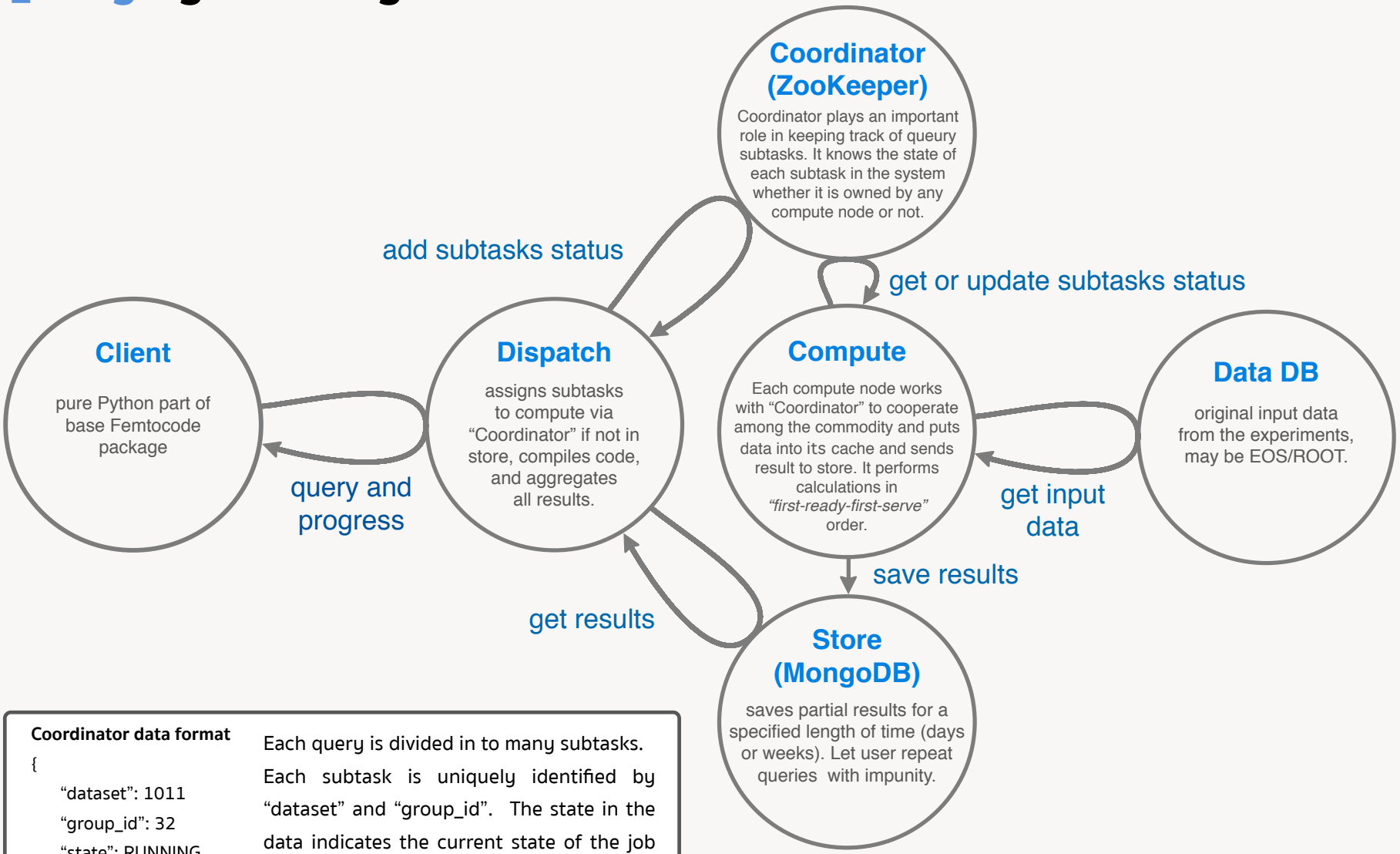
Single System Image

A client will see the same view of the service regardless of the server that it connects to.

Reliability

Once an update has been applied, it will persist from that time forward until a client overwrites the update.

Query System Layout

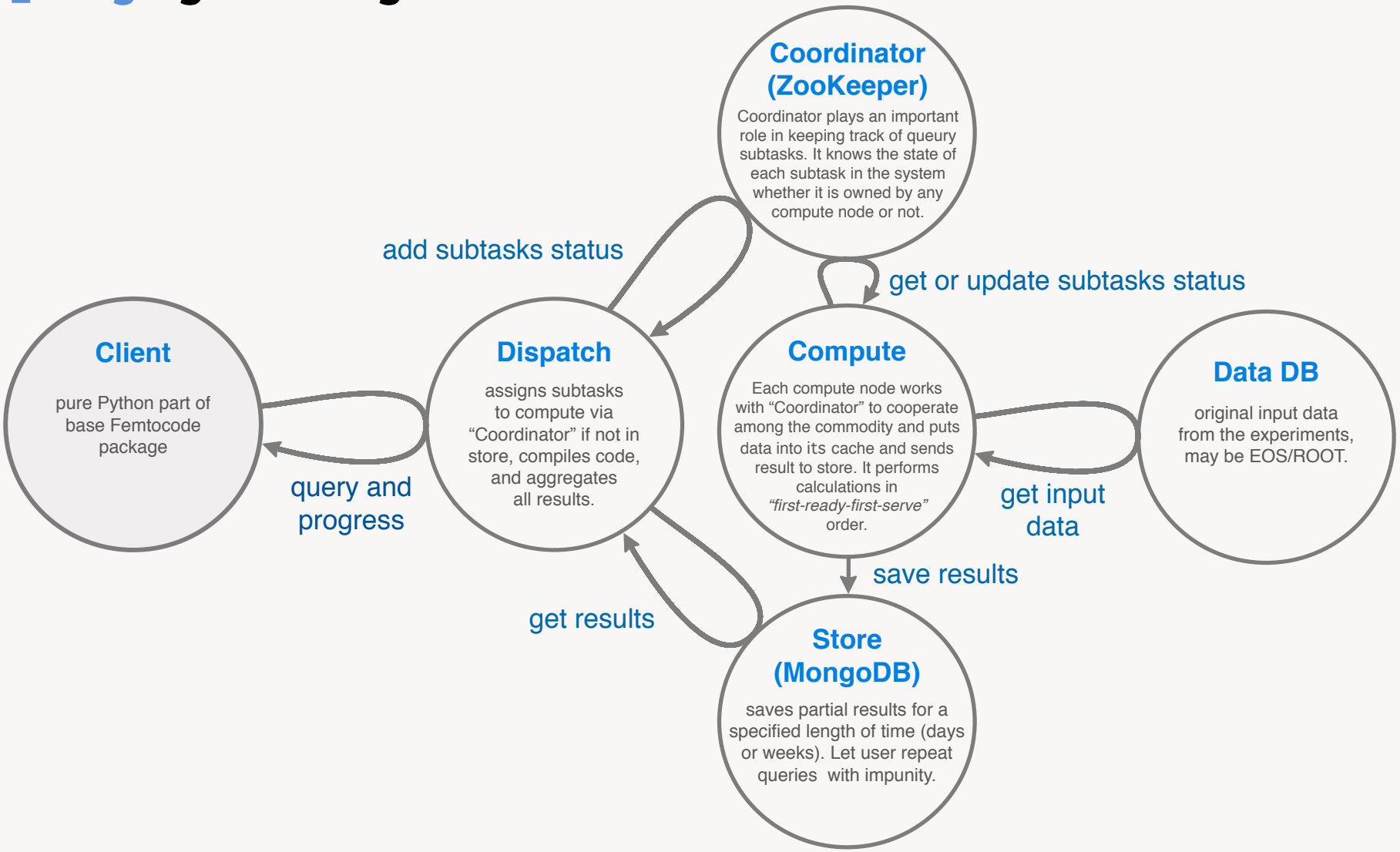


Coordinator data format

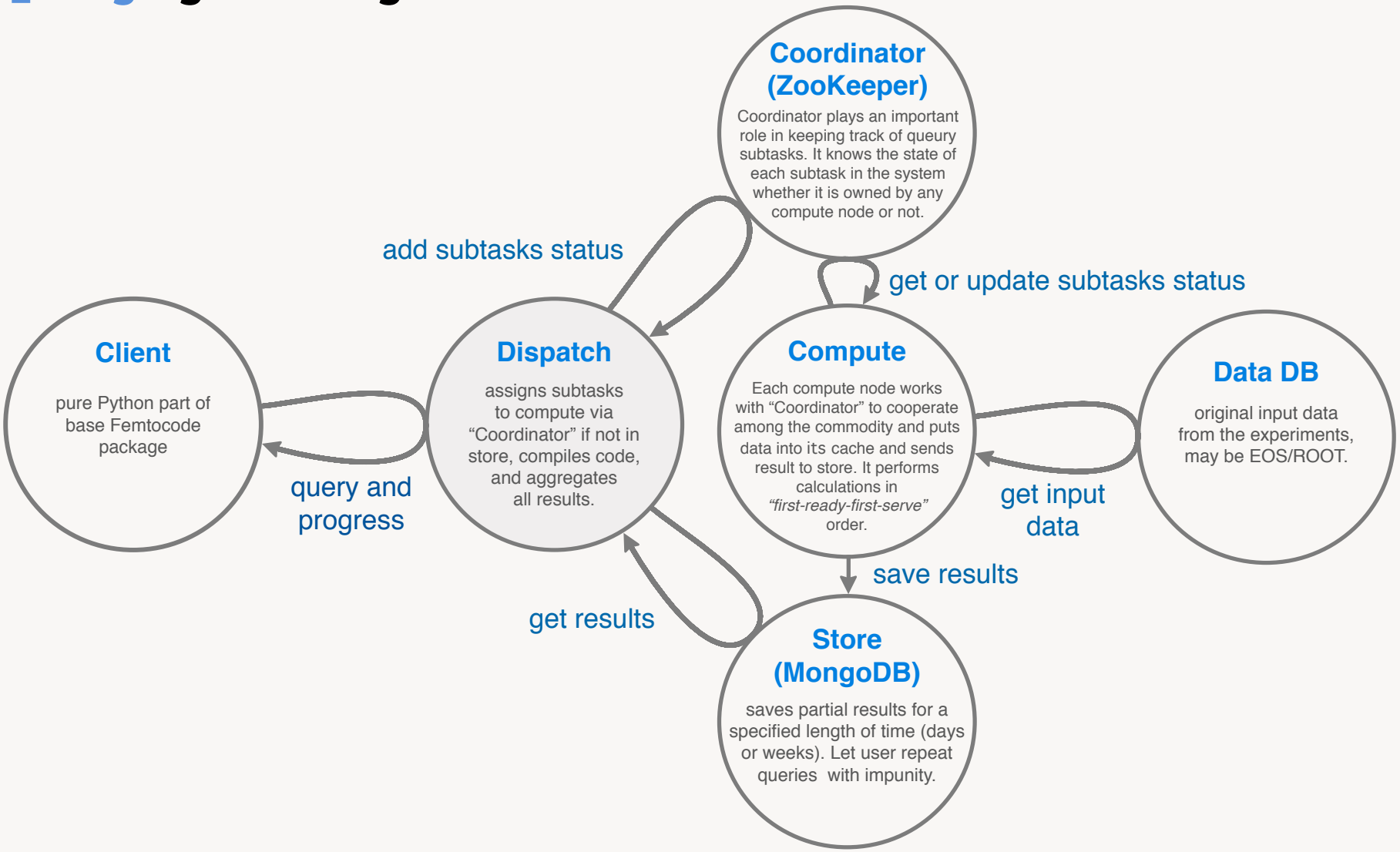
```
{
    "dataset": 1011
    "group_id": 32
    "state": RUNNING
    "worker": wk-0001
}
```

Each query is divided in to many subtasks. Each subtask is uniquely identified by "dataset" and "group_id". The state in the data indicates the current state of the job while worker stores the resoponsible worker for the task.

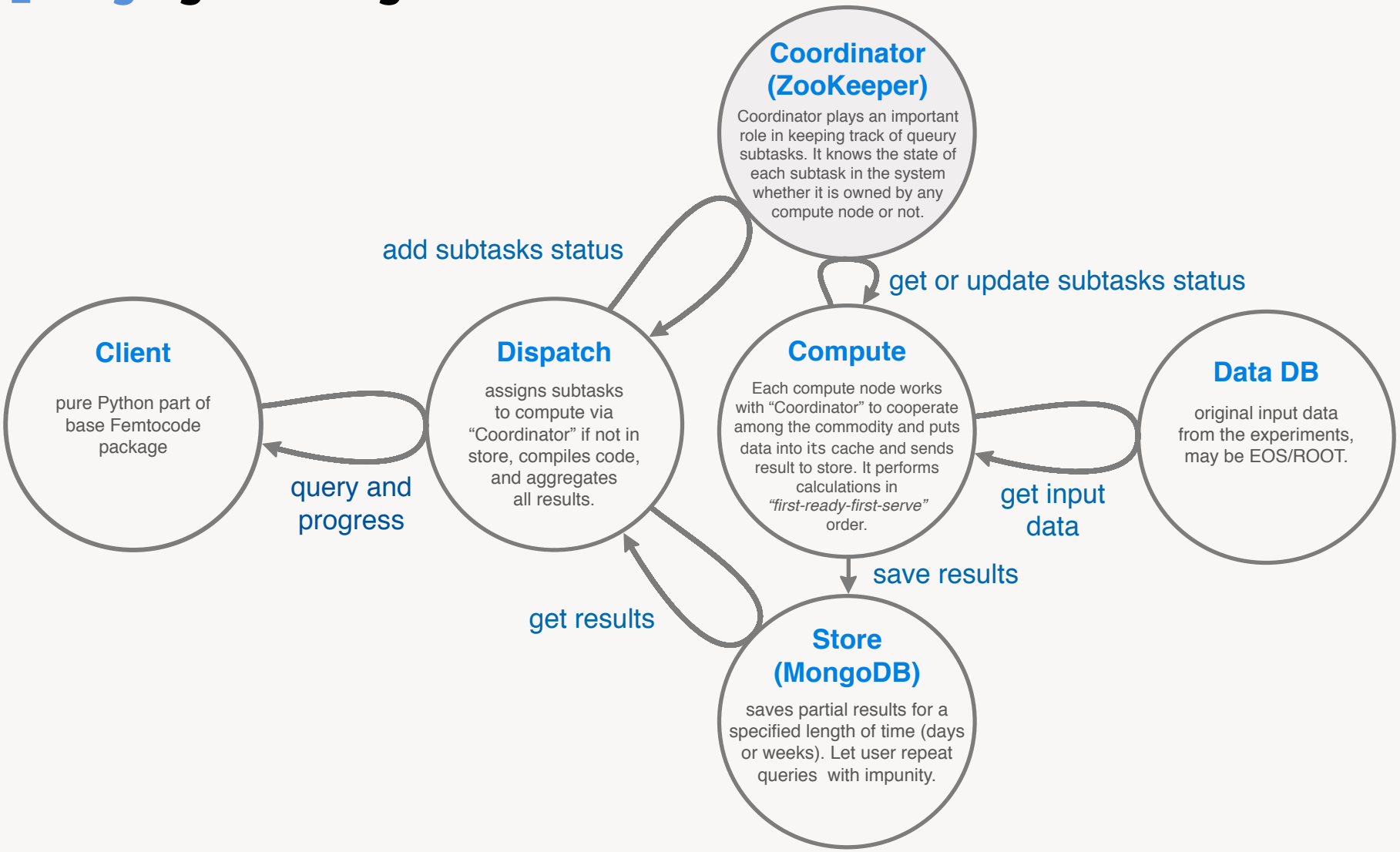
Query System Layout



Query System Layout



Query System Layout

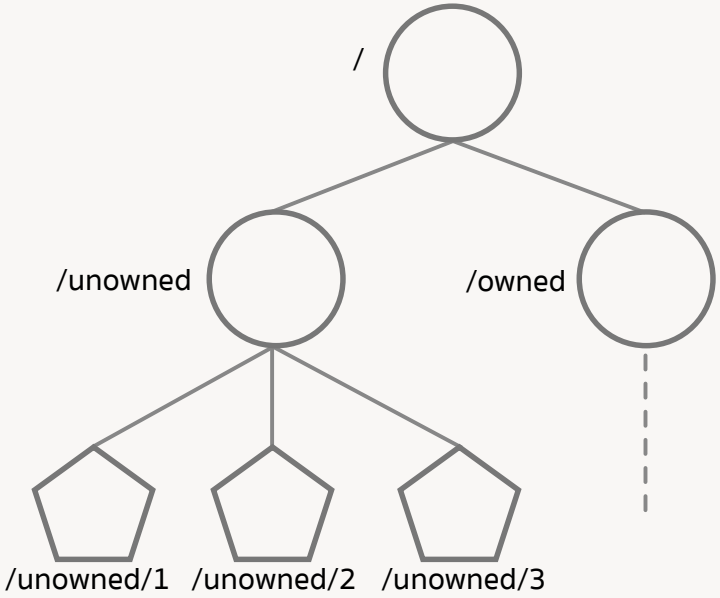


Query System Layout

Coordinator data format

```
{
    “dataset”: 1011
    “group_id”: 32
    “state”: RUNNING
    “worker”: wk-0001
}
```

Each query is divided in to many subtasks. Each subtask is uniquely identified by “dataset” and “group_id”. The state in the data indicates the current state of the job while worker stores the resoponsible worker for the task.



Coordinator (ZooKeeper)

Coordinator plays an important role in keeping track of queury subtasks. It knows the state of each subtask in the system whether it is owned by any compute node or not.

add subtasks status

get or update subtasks status

Dispatch

assigns subtasks to compute via “Coordinator” if not in store, compiles code,

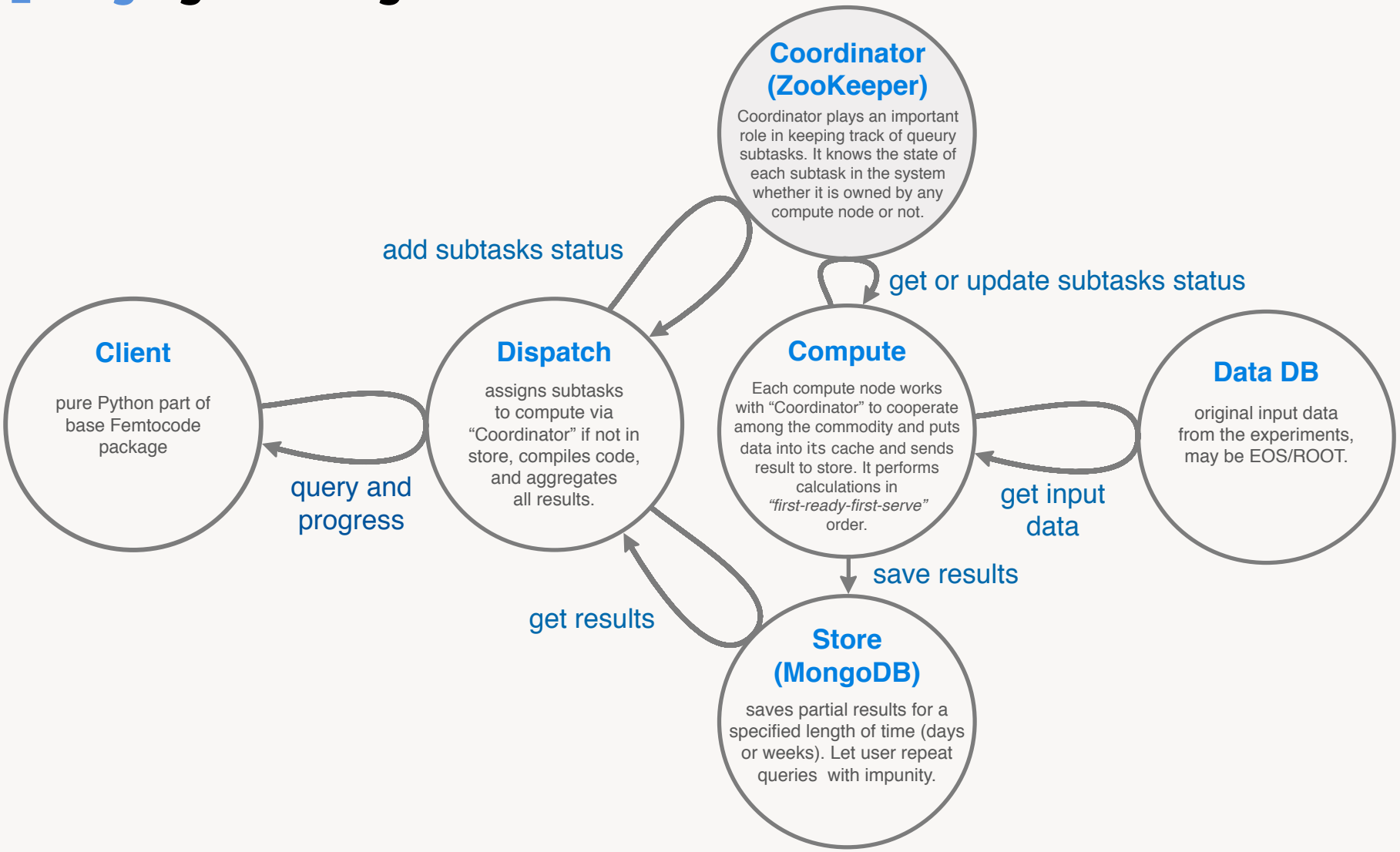
Compute

Each compute node works with “Coordinator” to cooperate among the commodity and puts data into its cache and sends result to store. It performs

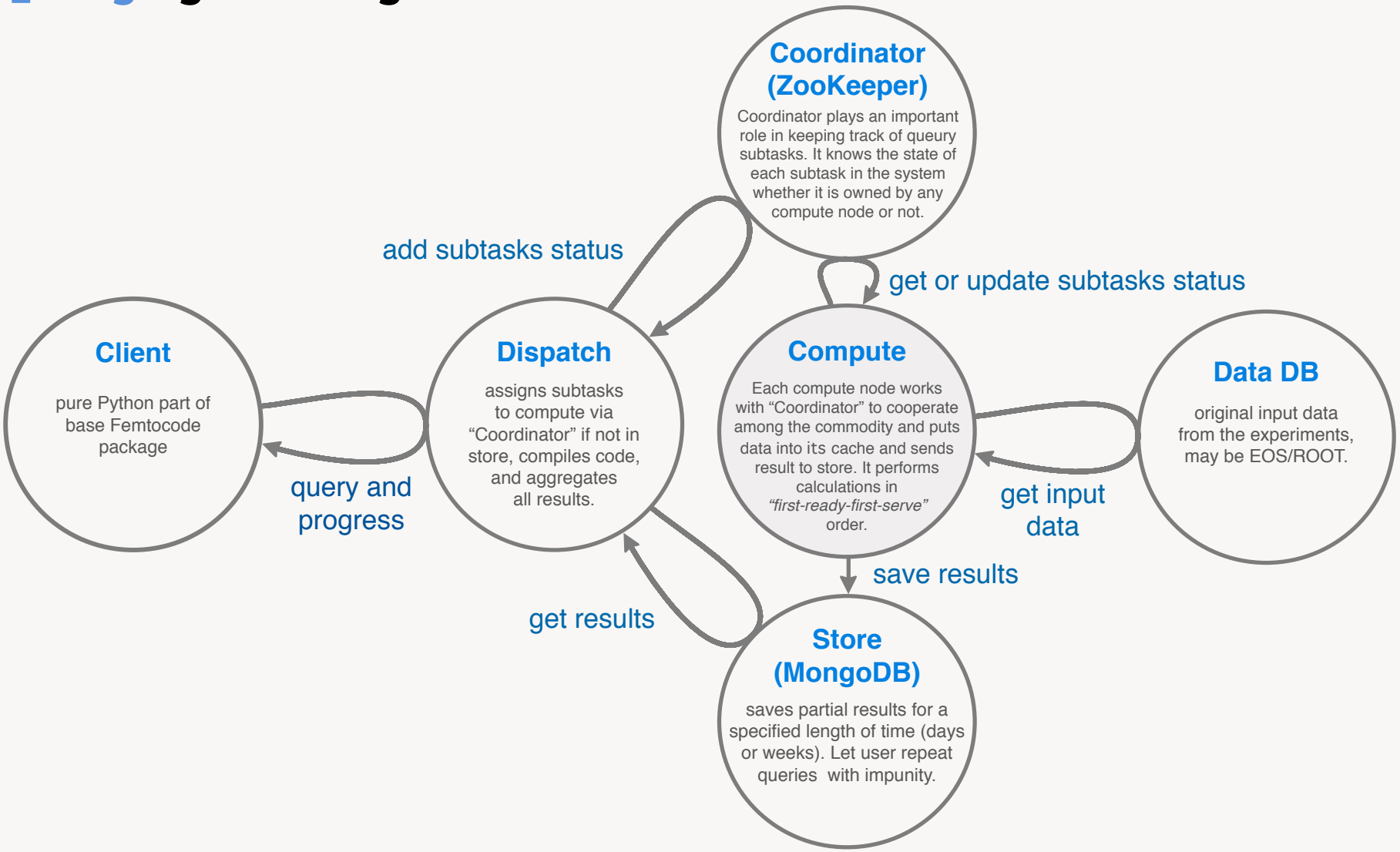
Data DB

original input data from the experiments, may be EOS/ROOT.

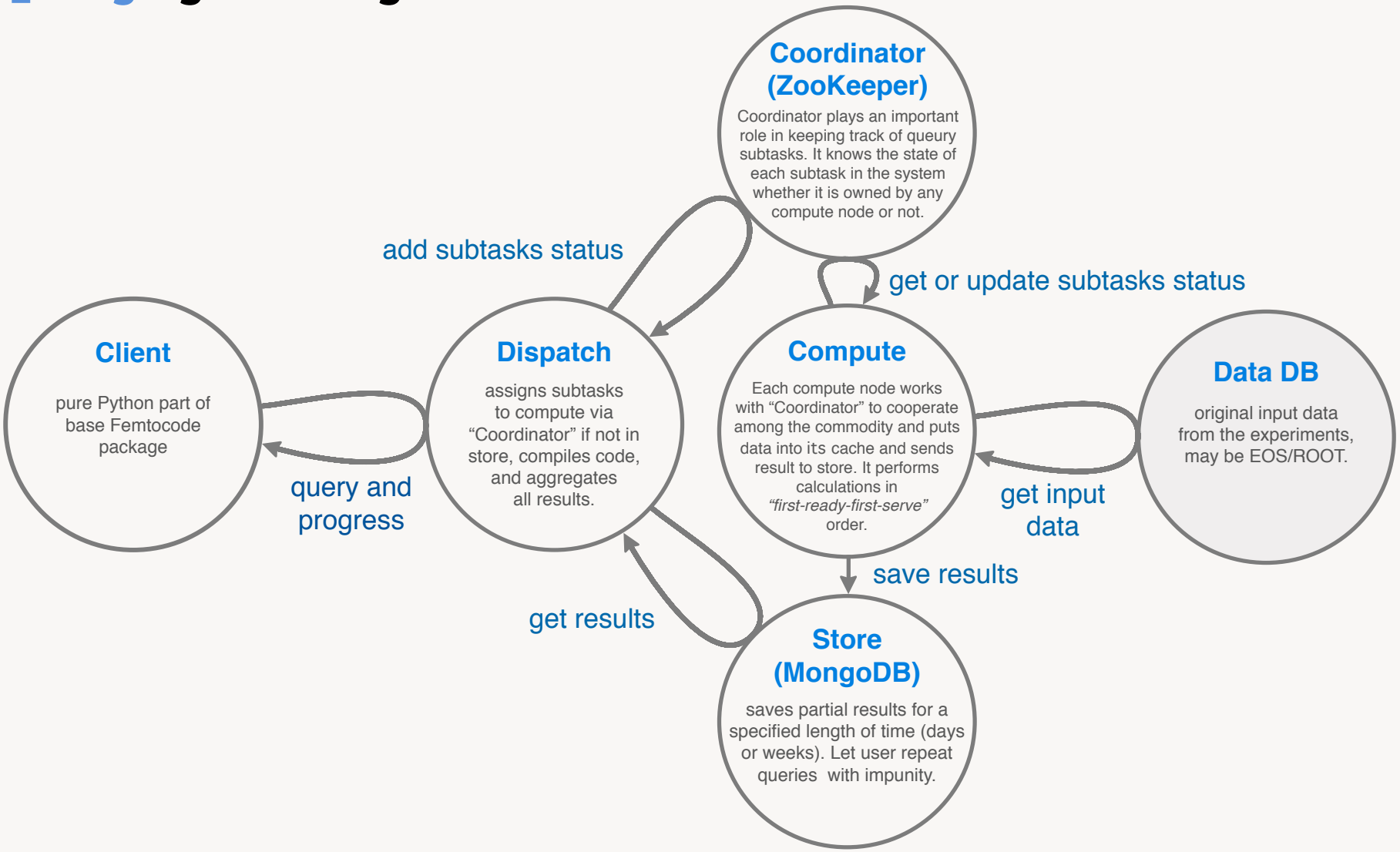
Query System Layout



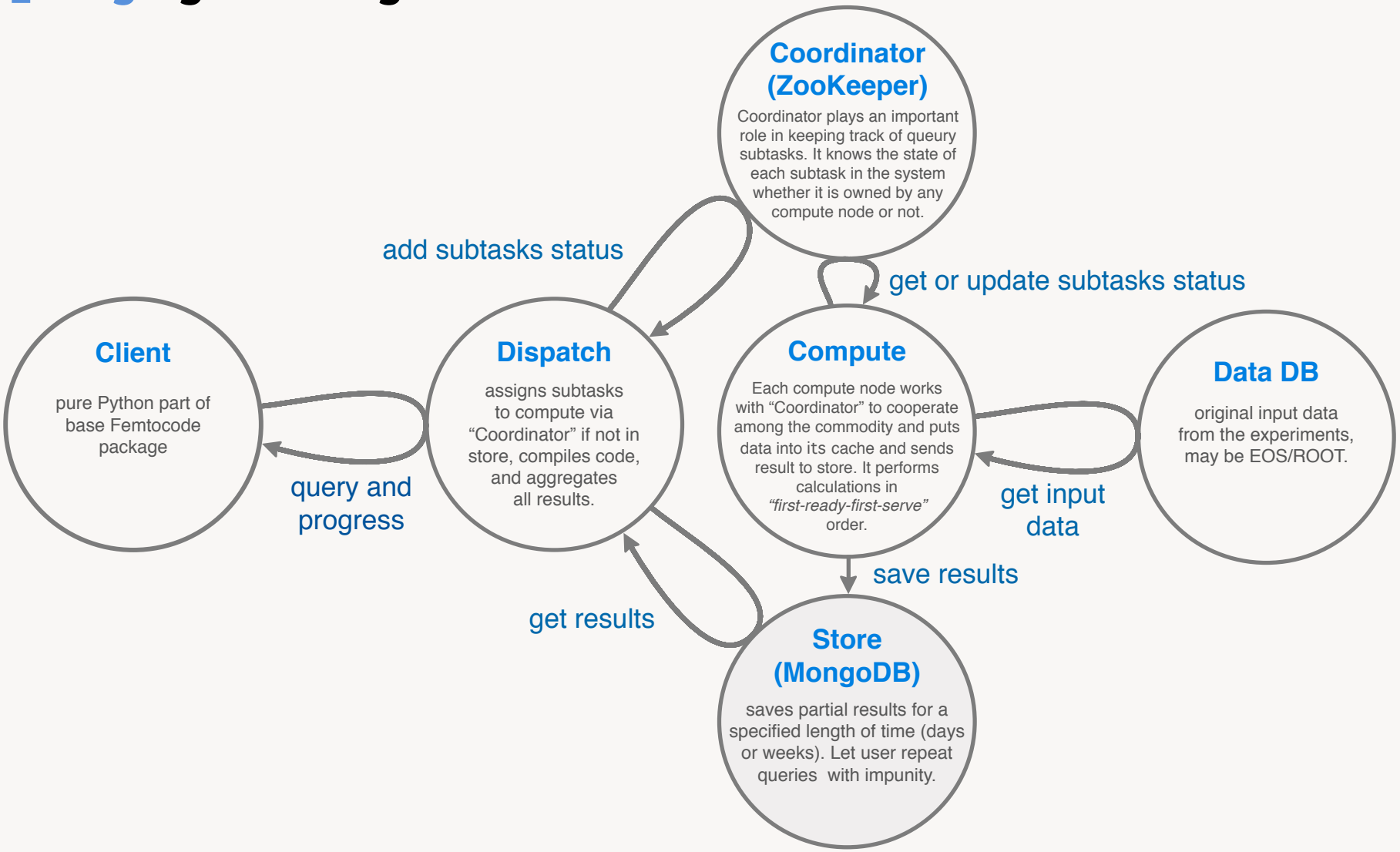
Query System Layout



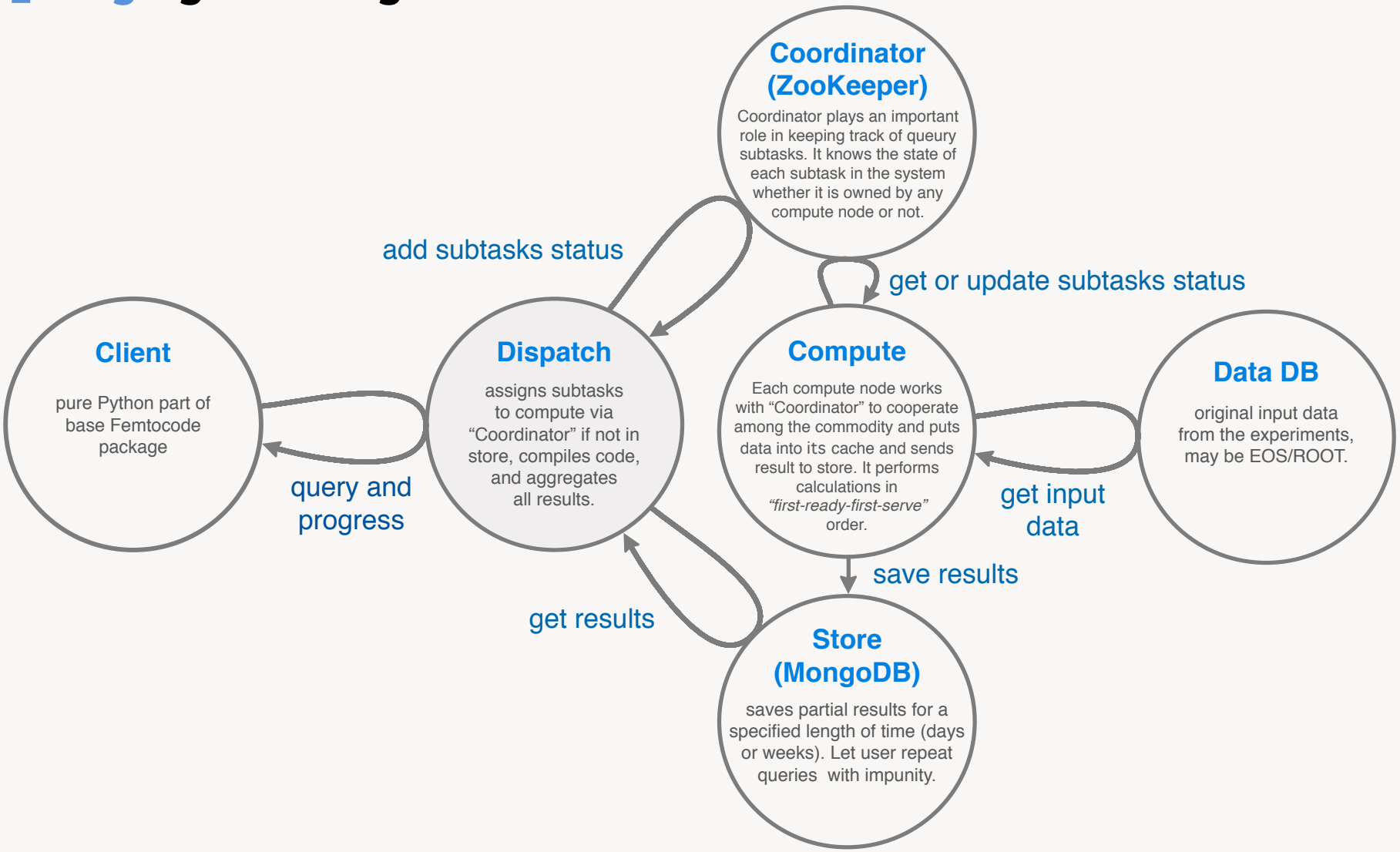
Query System Layout



Query System Layout



Query System Layout



Result from the design

Machine for System Testing

Local Development

Local Machine (Laptop)

It is used as a development machine for testing. The distributed programming is tested by using multiple threads.

Cloud Development

Private Cloud

CERN Openstack is being used for the current development of project. It is used to test development scale. Multiple instances are launched to test distributed query and high availability

Plan for Production

Private Cloud

Padova cluster will be used as a production test. It is Marathon Cluster. Hence, the application will be launched with Docker and scale with Marathon.

Example on local machine (laptop)

Put new job on Zookeeper example

```
(Thread-3  ) Received response(xid=150): []
(Thread-3  ) Sending request(xid=151): Create(path='/unowned/entry-100-3:3-',
data='{"state": 1, "groupid": 3, "dataset": 3}', acl=[ACL(perms=31, acl_list=['ALL'],
id=Id(scheme='world', id='anyone'))], flags=2)
(Thread-3  ) Sending request(xid=152): Create(path='/unowned/entry-100-2:2-',
data='{"state": 1, "groupid": 2, "dataset": 2}', acl=[ACL(perms=31, acl_list=['ALL'],
id=Id(scheme='world', id='anyone'))], flags=2)
(Thread-3  ) Sending request(xid=153): GetChildren(path='/unowned', watcher=None)
(Thread-3  ) Sending request(xid=154): GetChildren(path='/unowned', watcher=None)
(Thread-3  ) Sending request(xid=155): GetChildren(path='/owned', watcher=None)
(Thread-3  ) Received response(xid=151): u'/unowned/entry-100-3:3-0000000000'
(Thread-3  ) Received response(xid=152): u'/unowned/entry-100-2:2-0000000001'
(Thread-3  ) Received response(xid=153): [u'entry-100-3:3-0000000000', u'entry-100-2:2-0000000001']
(dispatcher-worker) putting job 1:1 into zookeeper
(Thread-3  ) Received response(xid=154): [u'entry-100-3:3-0000000000', u'entry-100-2:2-0000000001']
```

Code: https://github.com/JThanat/femto-mesos

Example on local machine (laptop)

Create a new data when owning the job

```
(Thread-3  ) Sending request(xid=172): Create(path='/owned/entry-100-3:3-',
data='{"state": 3, "worker": "819dee4c-1806-403c-8fcb-0b4f8e4b6e2c", "groupid": 3, "dataset": 3}',
acl=[ACL(perms=31, acl_list=['ALL'], id=Id(scheme='world', id='anyone'))], flags=2)
(Thread-3  ) Received response(xid=172): u'/owned/entry-100-3:3-0000000000'
(Thread-3  ) Sending request(xid=173): GetChildren(path='/unowned', watcher=None)
(Thread-3  ) Received response(xid=173): [u'entry-100-1:1-0000000002', u'entry-100-2:2-0000000001', u'entry-100-4:4-0000000004', u
(Thread-5  ) waiting 100 ms for available thread
(Thread-3  ) Sending request(xid=174): GetChildren(path='/owned', watcher=None)
(Thread-3  ) Received response(xid=174): [u'entry-100-3:3-0000000000']
(Thread-3  ) Sending request(xid=175): GetChildren(path='/unowned', watcher=None)
(Thread-3  ) Received response(xid=175): [u'entry-100-1:1-0000000002', u'entry-100-2:2-0000000001', u'entry-100-4:4-0000000004', u
(dispatcher-worker) waiting 100ms for 3:3 to be executed
(Thread-6  ) waiting 100 ms for available watcher
(Thread-3  ) Sending request(xid=176): GetChildren(path='/done', watcher=None)
(Thread-3  ) Received response(xid=176): []
(watcher-worker) No data to return
(dispatcher-worker) waiting 100ms for 0:0 to be executed
(dispatcher-worker) waiting 100ms for 2:2 to be executed
(dispatcher-worker) waiting 100ms for 1:1 to be executed
(Thread-4  ) waiting for a new job that satisfy
(Thread-3  ) Sending request(xid=177): GetChildren(path='/unowned', watcher=None)
(Thread-3  ) Received response(xid=177): [u'entry-100-1:1-0000000002', u'entry-100-2:2-0000000001', u'entry-100-4:4-0000000004', u
(Thread-3  ) Sending request(xid=178): GetData(path='/unowned/entry-100-2:2-0000000001', watcher=None)
(Thread-3  ) Received response(xid=178): ('{"state": 1, "groupid": 2, "dataset": 2}', ZnodeStat(czxid=6834, mzxid=6834, ctime=1502
(Thread-3  ) Sending request(xid=179): Delete(path='/unowned/entry-100-2:2-0000000001', version=-1)
(Thread-3  ) Received response(xid=179): True
(Thread-3  ) Sending request(xid=180): Create(path='/owned/entry-100-2:2-',
data='{"state": 3, "worker": "819dee4c-1806-403c-8fcb-0b4f8e4b6e2c", "groupid": 2, "dataset": 2}',
acl=[ACL(perms=31, acl_list=['ALL'], id=Id(scheme='world', id='anyone'))], flags=2)
```

Code: https://github.com/JThanat/femto-mesos

Example on local machine (laptop)

Put a successfully done job into Zookeeper for the watcher to poll the job

```
(Thread-3  ) Received response(xid=189): ('{"state": 3, "worker": "819dee4c-1806-403c-8fcb-0b4f8e4b6e2c", "groupid": 3,
(Thread-3  ) Sending request(xid=190): Create(path='/done/entry-100-3:3-',
data='{"state": 4, "worker": "819dee4c-1806-403c-8fcb-0b4f8e4b6e2c", "groupid": 3, "dataset": 3}',
acl=[ACL(perms=31, acl_list=['ALL'], id=Id(scheme='world', id='anyone'))], flags=2)
(Thread-3  ) Received response(xid=190): u'/done/entry-100-3:3-0000000000'
(Thread-4  ) waiting for a new job that satisfy
(Thread-3  ) Sending request(xid=191): GetChildren(path='/unowned', watcher=None)
(Thread-3  ) Received response(xid=191): [u'entry-100-4:4-0000000004', u'entry-100-0:0-0000000003']
(Thread-3  ) Sending request(xid=192): GetData(path='/unowned/entry-100-0:0-0000000003', watcher=None)
(Thread-3  ) Received response(xid=192): ('{"state": 1, "groupid": 0, "dataset": 0}', ZnodeStat(czxid=6836, mzxid=6836,
(Thread-3  ) Sending request(xid=193): Delete(path='/unowned/entry-100-0:0-0000000003', version=-1)
(Thread-3  ) Received response(xid=193): True
(Thread-3  ) Sending request(xid=194): Create(path='/owned/entry-100-0:0-',
data='{"state": 3, "worker": "819dee4c-1806-403c-8fcb-0b4f8e4b6e2c", "groupid": 0, "dataset": 0}',
acl=[ACL(perms=31, acl_list=['ALL'], id=Id(scheme='world', id='anyone'))], flags=2)
(Thread-3  ) Received response(xid=194): u'/owned/entry-100-0:0-0000000003'
(worker    ) finish saving job_id:5986e40d64b4385ee6b9984b in mongo
(Thread-3  ) Sending request(xid=195): Delete(path=u'/owned/entry-100-3:3-0000000000', version=-1)
(Thread-3  ) Received response(xid=195): True
(Thread-5  ) waiting 100 ms for available thread
(dispatcher-worker) return executed 3:3 objectid:5986e40d64b4385ee6b9984b after waiting
(Thread-6  ) waiting 100 ms for available watcher
(Thread-3  ) Sending request(xid=196): GetChildren(path='/done', watcher=None)
(Thread-3  ) Received response(xid=196): [u'entry-100-3:3-0000000000']
(Thread-3  ) Sending request(xid=197): GetData(path='/done/entry-100-3:3-0000000000', watcher=None)
(Thread-3  ) Received response(xid=197): ('{"state": 4, "worker": "819dee4c-1806-403c-8fcb-0b4f8e4b6e2c", "groupid": 3,
(Thread-3  ) Sending request(xid=198): Delete(path='/done/entry-100-3:3-0000000000', version=-1)
```

Code: https://github.com/JThanat/femto-mesos

Conclusion

From the investigation Mesos/Marathon might be good when we want to build the application at the scale. ZooKeeper can also be used as a part of Mesos High availability and a shared storage coordinator as well.

As of now, ZooKeeper is enough to satisfy the design as a job coordinator for a computing cluster.

Future Work

1. Implement and test the engine on a big cluster using Marathon and Docker.
2. Test performance and system scalability at large scale.
3. Integrate the query engine with the language to test system compatibility
4. Design MongoDB Sharding for large scale servcie