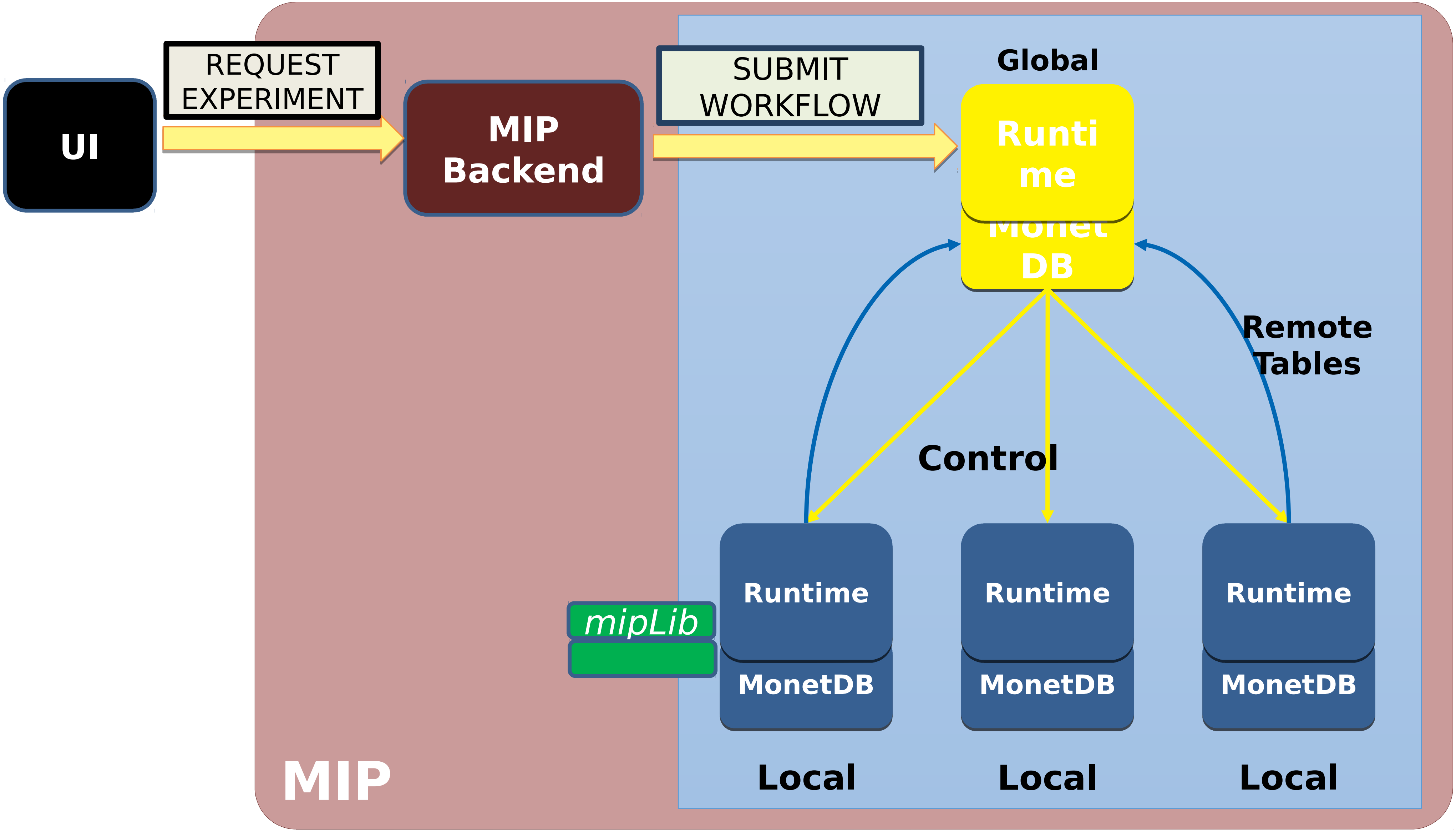
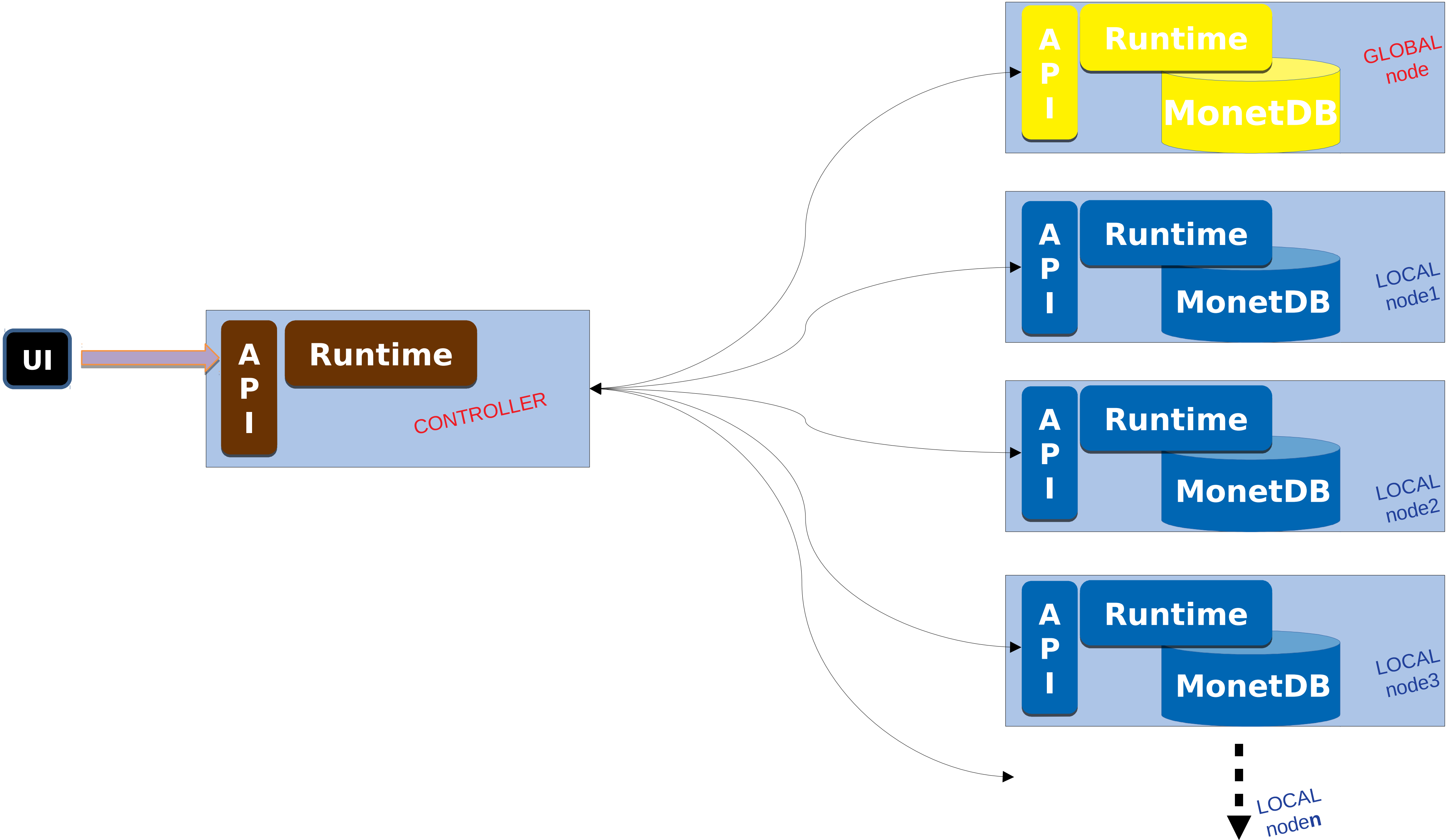
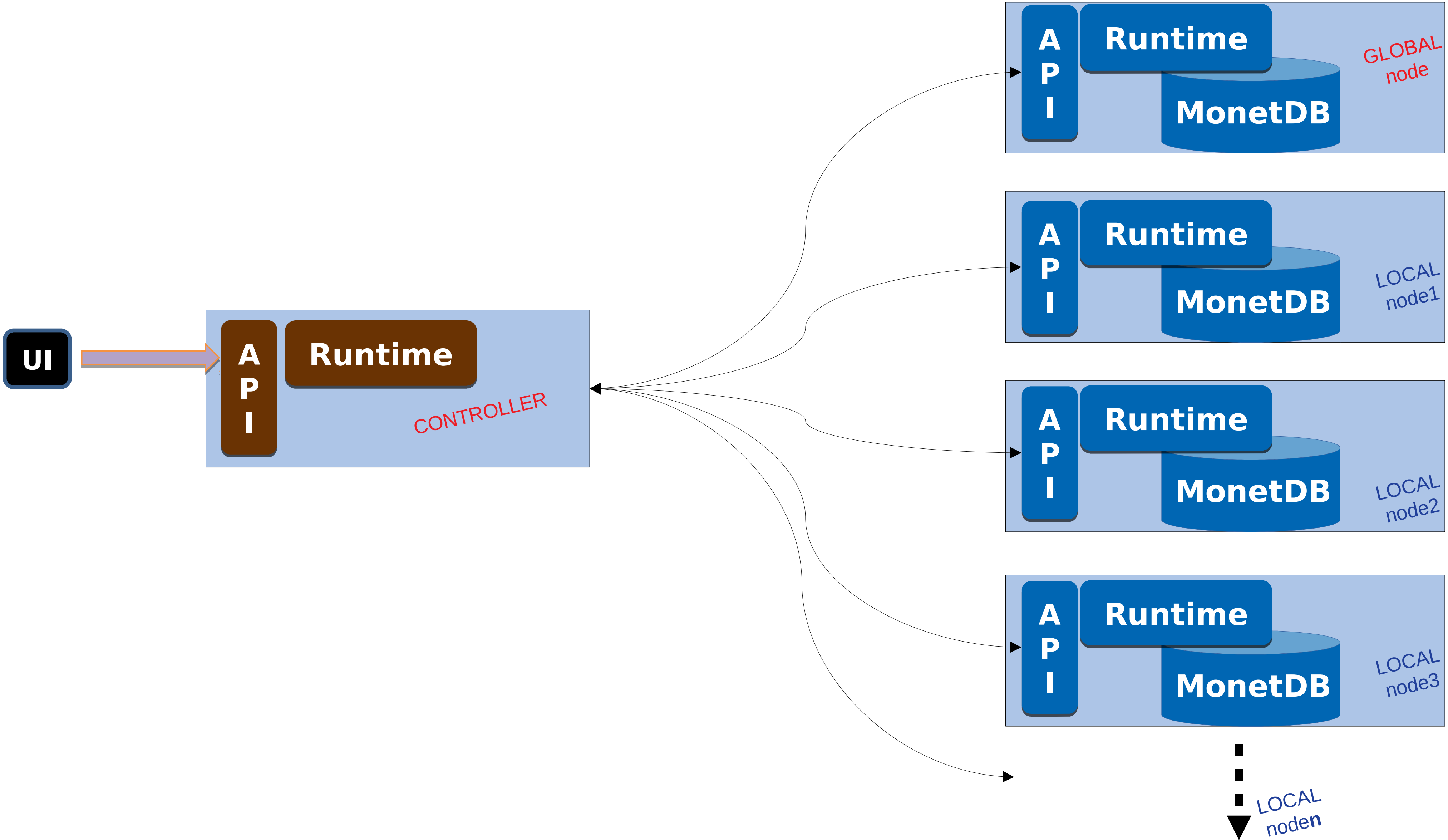
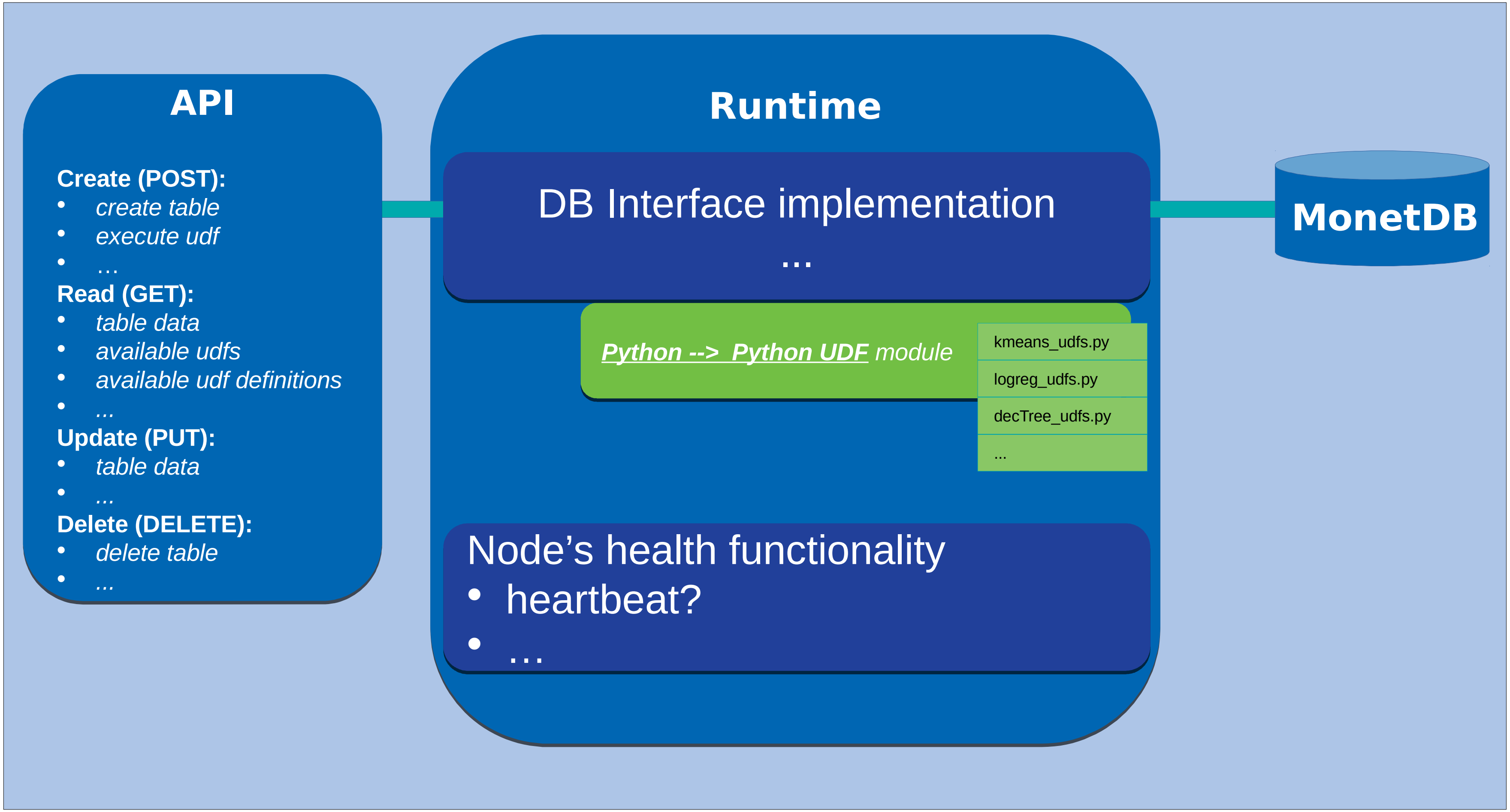

UI
REQUEST EXPERIMENT
MIP Backend
SUBMIT WORKFLOW
Global
Runtime
MonetDB
Remote Tables
Control
mipLib
Runtime
MonetDB
Runtime
MonetDB
Runtime
MonetDB
MIP
Local
Local
Local


UI
A P I
Runtime
CONTROLLER
A P I
Runtime
MonetDB
GLOBAL node
A P I
Runtime
MonetDB
LOCAL node1
A P I
Runtime
MonetDB
LOCAL node2
A P I
Runtime
MonetDB
LOCAL node3
LOCAL noden


UI
A P I
Runtime
CONTROLLER
A P I
Runtime
MonetDB
GLOBAL node
A P I
Runtime
MonetDB
LOCAL node1
A P I
Runtime
MonetDB
LOCAL node2
A P I
Runtime
MonetDB
LOCAL node3
LOCAL noden

GLOBAL/LOCAL node



API

Create (POST):

- *create table*
- *execute udf*
- …

Read (GET):

- *table data*
- *available udfs*
- *available udf definitions*
- *…*

Update (PUT):

- *table data*
- *…*

Delete (DELETE):

- *delete table*
- *…*

Runtime

DB Interface implementation

…

***Python --> Python UDF** module*

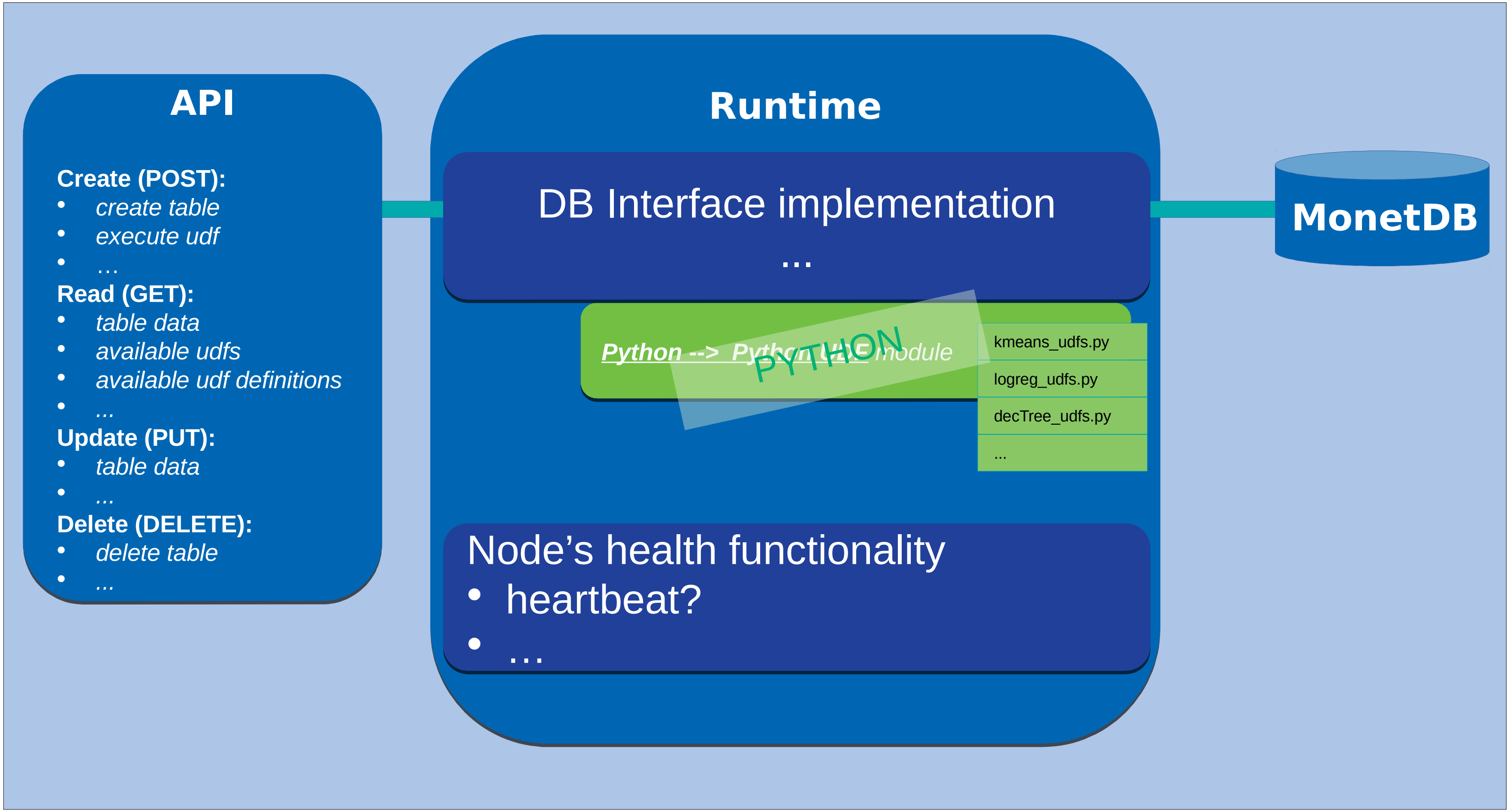
- kmeans_udfs.py
- logreg_udfs.py
- decTree_udfs.py
- ...

Node’s health functionality

- heartbeat?
- …

MonetDB

GLOBAL/LOCAL node



API

Create (POST):
- *create table*
- *execute udf*
- …

Read (GET):
- *table data*
- *available udfs*
- *available udf definitions*
- *…*

Update (PUT):
- *table data*
- *…*

Delete (DELETE):
- *delete table*
- *…*

Runtime

DB Interface implementation

…

***Python --> Python UDF** module*

PYTHON

kmeans_udfs.py

logreg_udfs.py

decTree_udfs.py

...

Node’s health functionality
- heartbeat?
- …

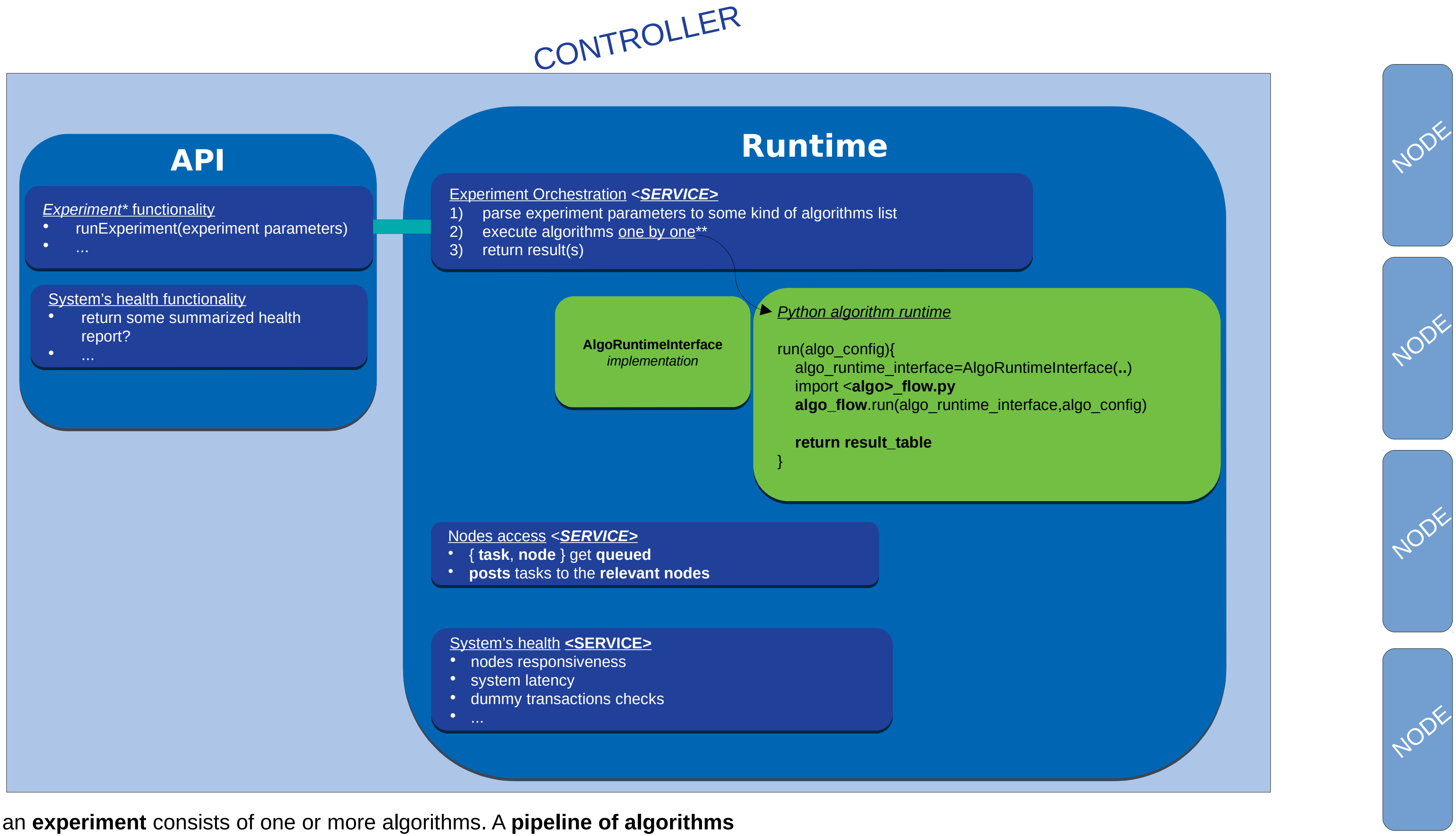
MonetDB

kmeans_flow.py

```
class AlgorithmFlow(AlgorithmFlow_abstract):
    async def run(self):
        runtime_interface=self.runtime_interface

        num_of_clusters=self.algorithm_parameters["numOfClusters"]
        num_of_iterations=self.algorithm_parameters["numOfIterations"]
        initial_min=self.algorithm_parameters["initialMin"]
        initial_max=self.algorithm_parameters["initialMax"]

        centroids_table_name=runtime_interface.execute_udf_global("generate_initial_centroids",udf_result_schema,[num_of_dimensions,num_of_clusters,initial_min,initial_max])
        runtime_interface.create_remote_global_to_locals(centroids_table_name)

        for i in range(num_of_iterations):
            local_centroids=runtime_interface.execute_udf_locals("local_calc",result_table_schema,[self.data_table_name,centroids_table_name])
            merged_table_name=runtime_interface.create_remote_and_merge_global(local_centroids)
            centroids_table_name= runtime_interface.execute_udf_global("global_calc",num_of_clusters,merged_table_name)
            runtime_interface.create_remote_global_to_locals(table,centroids_table_name)

        return centroids_table_name
```

kmeans_udfs.py

```
def generate_initial_centroids(num_of_dimensions:int, num_of_clusters:int, min_val:float, max_val:float) -> Dict:
    ##IMPLEMENTATION##

def local_calc(centroids_data_cross_prod:ArrayBundle):
    ##IMPLEMENTATION##

def global_calc(num_of_clusters:int, local_centroids:List[ArrayBundle]):
    ##IMPLEMENTATION##
```



CONTROLLER

API

*Experiment** functionality
- runExperiment(experiment parameters)
- …

System's health functionality
- return some summarized health report?
- …

Runtime

Experiment Orchestration <***SERVICE>***
1) parse experiment parameters to some kind of algorithms list
2) execute algorithms one by one**
3) return result(s)

AlgoRuntimeInterface
implementation

Python algorithm runtime

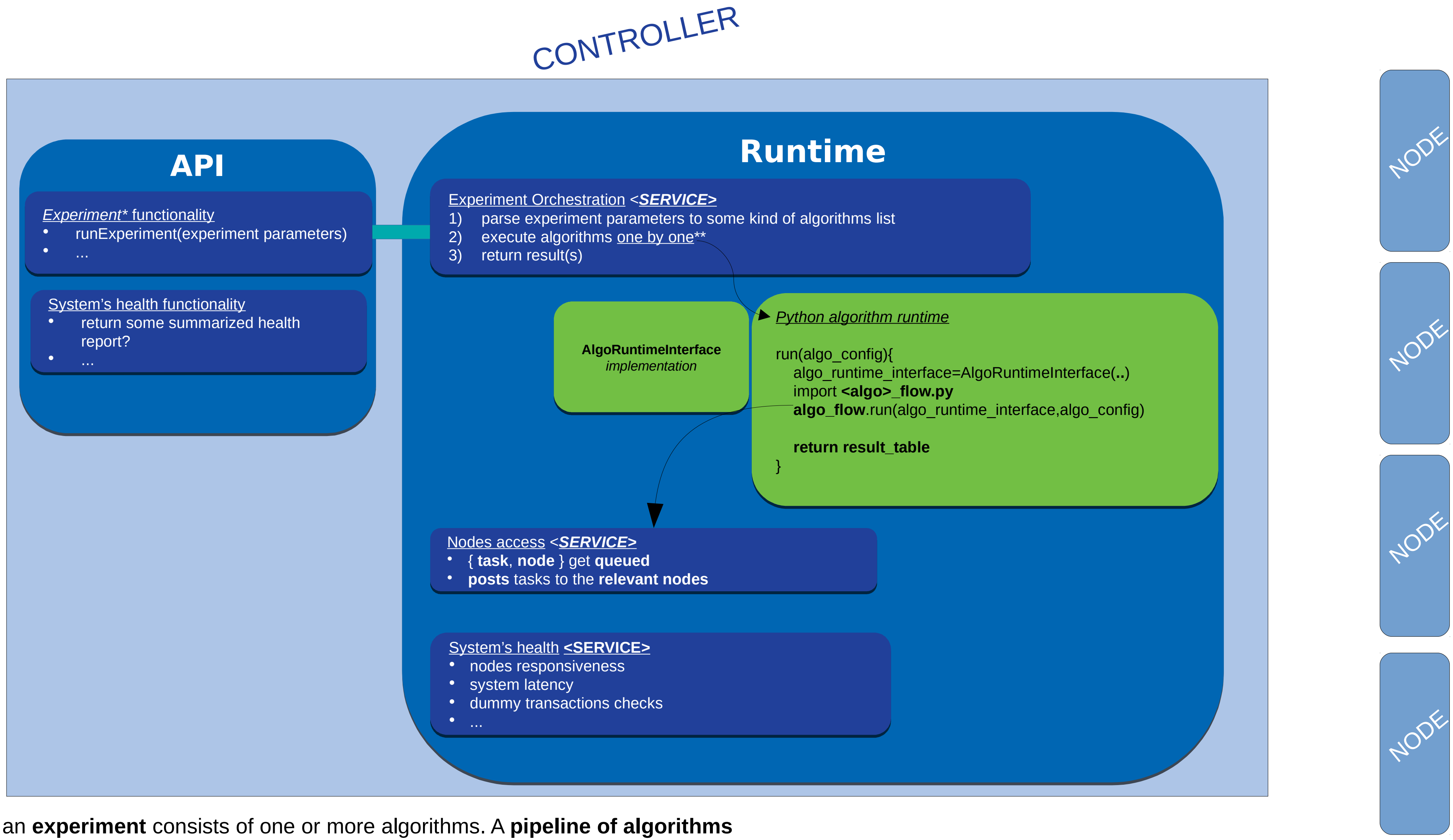
```
run(algo_config){
    algo_runtime_interface=AlgoRuntimeInterface(..)
    import <algo>_flow.py
    algo_flow.run(algo_runtime_interface,algo_config)

    return result_table
}
```

Nodes access <***SERVICE>***
- { **task**, **node** } get **queued**
- **posts** tasks to the **relevant nodes**

System's health **<SERVICE>**
- nodes responsiveness
- system latency
- dummy transactions checks
- …

* an **experiment** consists of one or more algorithms. A **pipeline of algorithms**

** algorithm flow design must allow pipelining



CONTROLLER

API

*Experiment** functionality

- runExperiment(experiment parameters)
- …

System's health functionality

- return some summarized health report?
- …

Runtime

Experiment Orchestration <***SERVICE>***

1) parse experiment parameters to some kind of algorithms list
2) execute algorithms one by one**
3) return result(s)

AlgoRuntimeInterface
implementation

Python algorithm runtime

```
run(algo_config){
    algo_runtime_interface=AlgoRuntimeInterface(..)
    import <algo>_flow.py
    algo_flow.run(algo_runtime_interface,algo_config)

    return result_table
}
```

Nodes access <***SERVICE>***

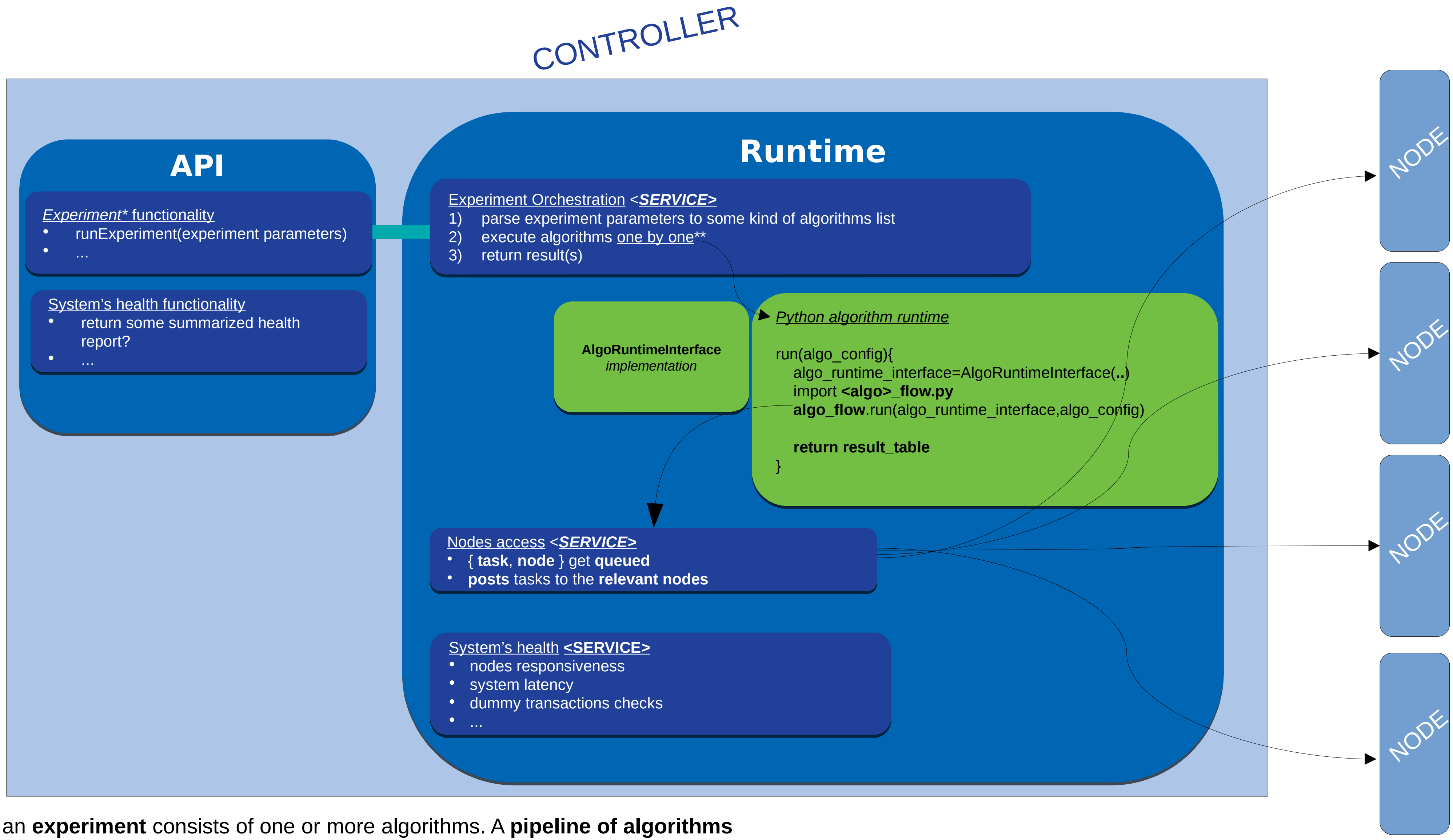
- { **task**, **node** } get **queued**
- **posts** tasks to the **relevant nodes**

System's health **<SERVICE>**

- nodes responsiveness
- system latency
- dummy transactions checks
- …

* an **experiment** consists of one or more algorithms. A **pipeline of algorithms**

** algorithm flow design must allow pipelining



CONTROLLER

API

*Experiment** functionality
- runExperiment(experiment parameters)
- …

System's health functionality
- return some summarized health report?
- …

Runtime

Experiment Orchestration <***SERVICE>***
1) parse experiment parameters to some kind of algorithms list
2) execute algorithms one by one**
3) return result(s)

AlgoRuntimeInterface
implementation

Python algorithm runtime

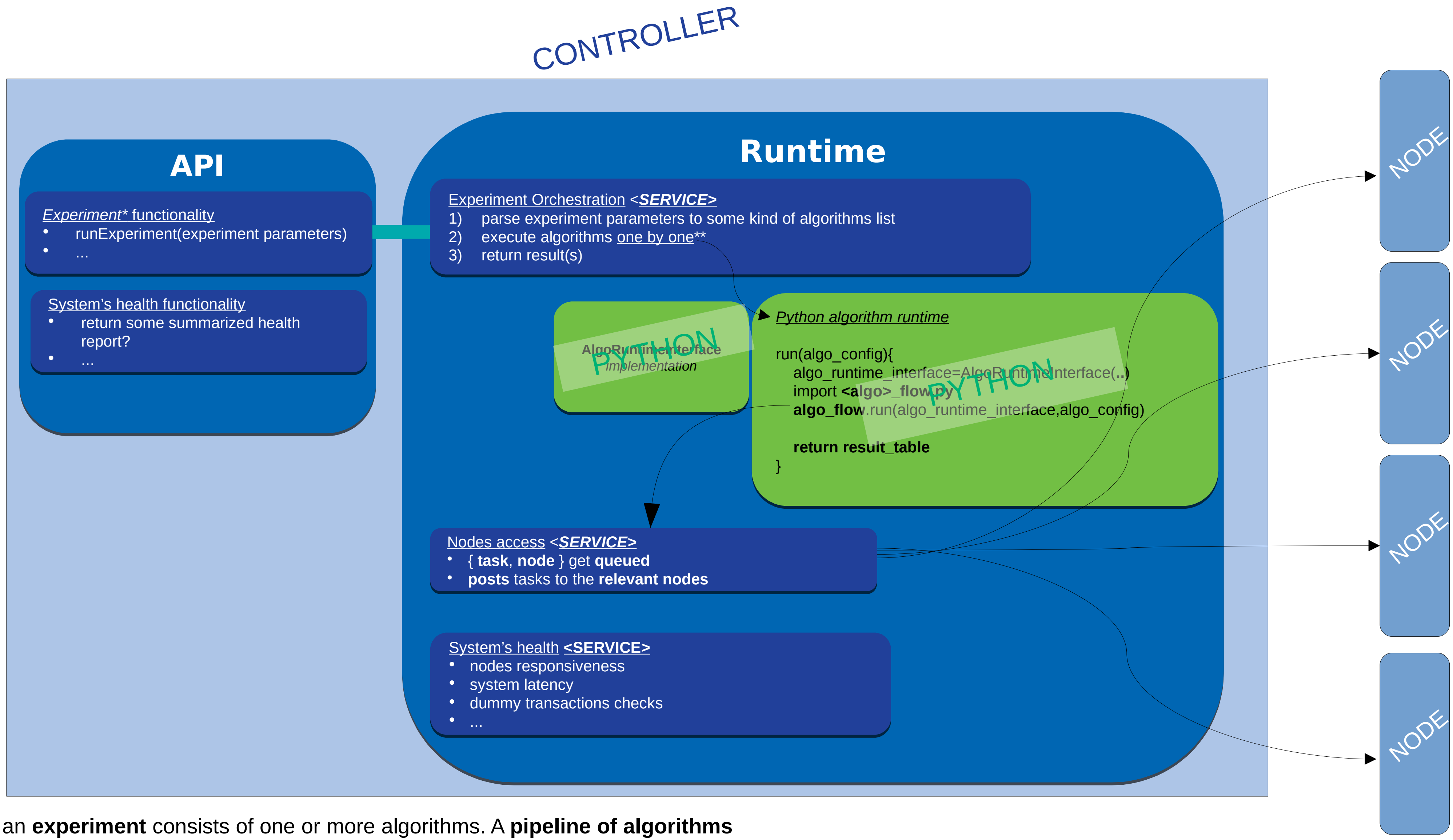
```
run(algo_config){
    algo_runtime_interface=AlgoRuntimeInterface(..)
    import <algo>_flow.py
    algo_flow.run(algo_runtime_interface,algo_config)

    return result_table
}
```

Nodes access <***SERVICE>***
- { **task**, **node** } get **queued**
- **posts** tasks to the **relevant nodes**

System's health **<SERVICE>**
- nodes responsiveness
- system latency
- dummy transactions checks
- …

* an **experiment** consists of one or more algorithms. A **pipeline of algorithms**

** algorithm flow design must allow pipelining



API

*Experiment** functionality
- runExperiment(experiment parameters)
- …

System's health functionality
- return some summarized health report?
- …

Runtime

Experiment Orchestration <***SERVICE>***
1) parse experiment parameters to some kind of algorithms list
2) execute algorithms one by one**
3) return result(s)

AlgoRuntimeInterface
implementation

Python algorithm runtime

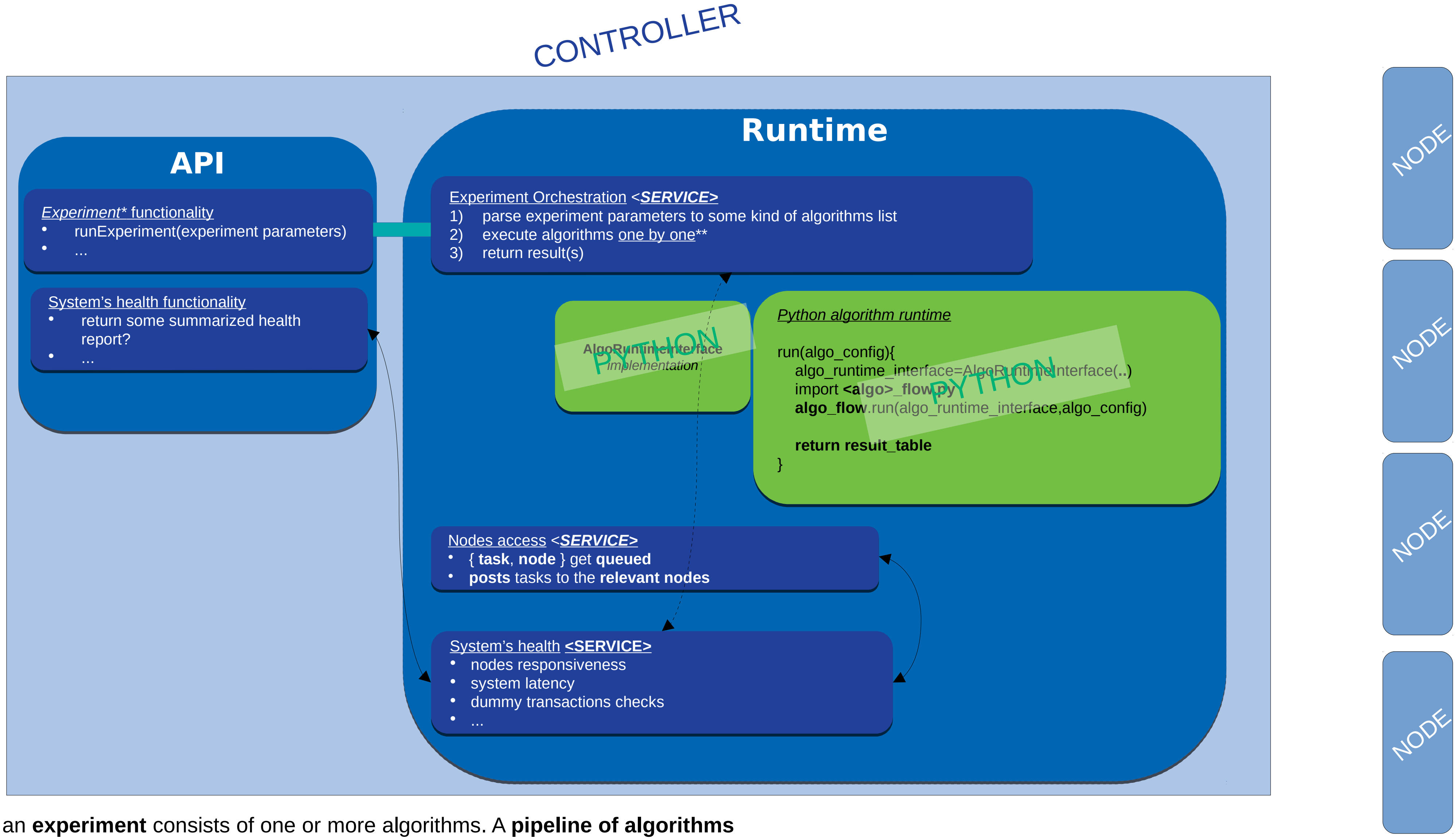
```
run(algo_config){
    algo_runtime_interface=AlgoRuntimeInterface(..)
    import <algo>_flow.py
    algo_flow.run(algo_runtime_interface,algo_config)

    return result_table
}
```

Nodes access <***SERVICE>***
- { **task**, **node** } get **queued**
- **posts** tasks to the **relevant nodes**

System's health **<SERVICE>**
- nodes responsiveness
- system latency
- dummy transactions checks
- …

* an **experiment** consists of one or more algorithms. A **pipeline of algorithms**

** algorithm flow design must allow pipelining



CONTROLLER

API

*Experiment** functionality*
- runExperiment(experiment parameters)
- …

System's health functionality
- return some summarized health report?
- …

Runtime

Experiment Orchestration <***SERVICE>***
1) parse experiment parameters to some kind of algorithms list
2) execute algorithms one by one**
3) return result(s)

AlgoRuntimeInterface
implementation

Python algorithm runtime

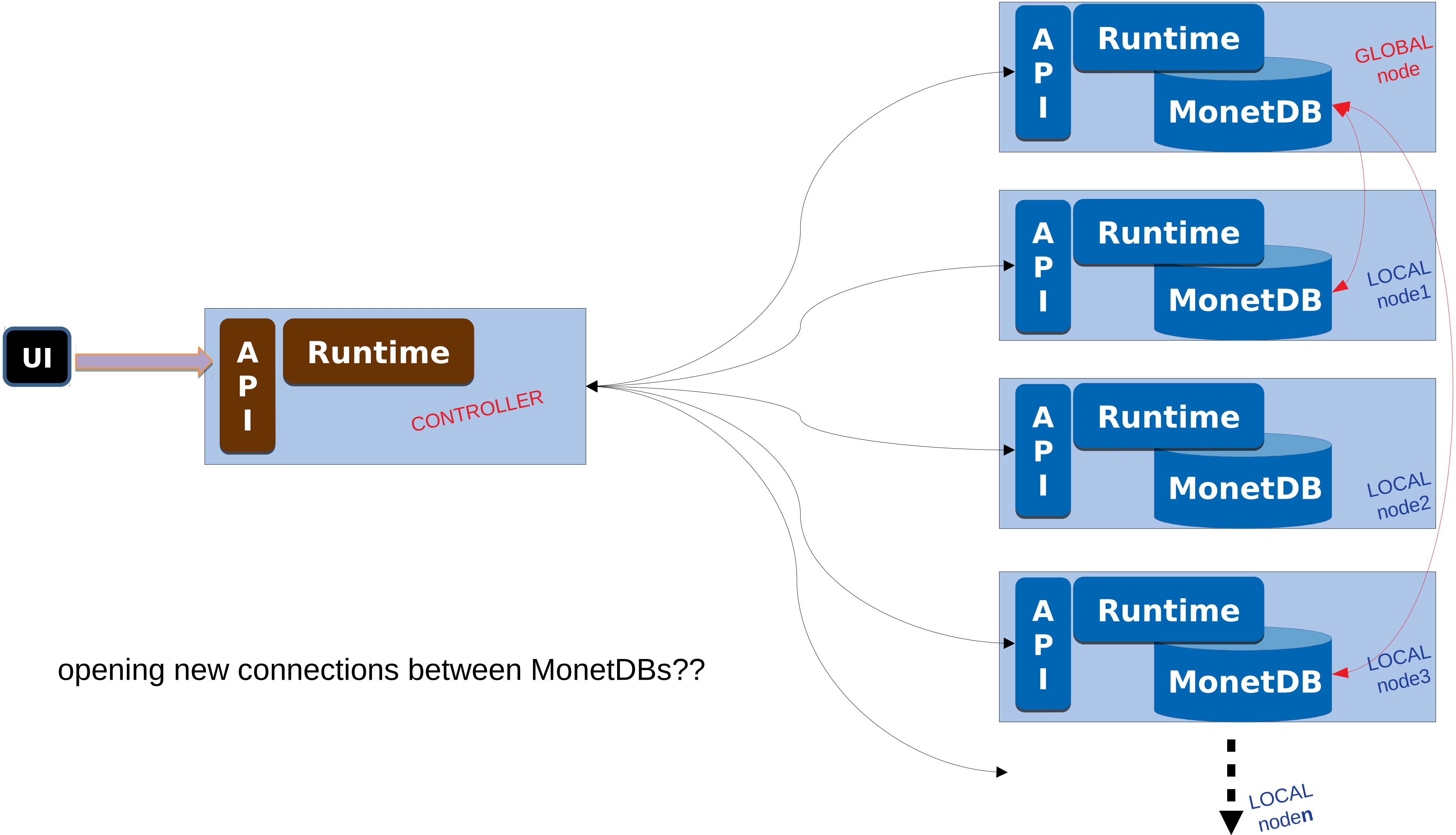
```
run(algo_config){
    algo_runtime_interface=AlgoRuntimeInterface(..)
    import <algo>_flow.py
    algo_flow.run(algo_runtime_interface,algo_config)

    return result_table
}
```

Nodes access <***SERVICE>***
- { **task**, **node** } get **queued**
- **posts** tasks to the **relevant nodes**

System's health **<SERVICE>**
- nodes responsiveness
- system latency
- dummy transactions checks
- …

* an **experiment** consists of one or more algorithms. A **pipeline of algorithms**

** algorithm flow design must allow pipelining



opening new connections between MonetDBs??