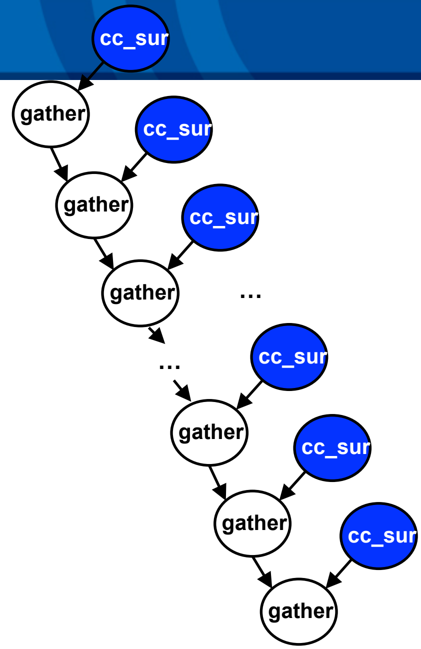
Programmatic workflows in PyCOMPSs



Tasks definition

```
@task(cc_original = INOUT, cc_surrs = INOUT)
def gather(result, cc_original, cc_surrs, start, end):
    cc_original[start:end,:] = result[0]
    cc_surrs[start:end,:,:] = result[1]

@task(returns = list)
def cc_surrogate_range(start_idx, end_idx, seed, num_neurons,
num_surrs, num_bins, maxlag):
    …
```

- Programmatic workflows
  - Standard sequential coordination scripts and applications in Python
  - Incremental changes: Task annotations + directionality hints
- Runtime
  - DAG generation based on data dependences: files and objects
  - Tasks and objects offload
- Platform unaware
  - Clusters (also MIC)
  - Clouds, distributed computing

Main program

```
from pycompss.api.api import compss_wait_on

cc_original = zeros((num_ccs,2*maxlag+1))
cc_surrs = zeros((num_ccs,2*maxlag+1,2))
for frag in range(num_frags):
    …
    result = cc_surrogate_range(start_idx, end_idx, seed, …
    gather(result, cc_original, cc_surrs, start_idx, end_idx)
    seed = seed + delta

f = open('./result_cc_originals.dat','w')
cc_original = compss_wait_on(cc_original)
pickle.dump(cc_original,f)
f.close()

…
```

PyCOMPSs integrated with persistent storage

```
from pycompss.api.task import task

@task()
def cc_surrogate_range(block_i, block_j, nd, correlation,
seed, num_surrs, num_bins, maxlag):

...
```

```
import sys

neuron_data_name = sys.argv[1]
correlation_name = sys.argv[2]

nd = NeuronData(neuron_data_name )
correlation = Correlation()


seed = 2398645
delta = 1782324

for block_i in nd.spikes.keys():
  for block_j in nd.spikes.keys():
      cc_surrogate_range(block_i, block_j, nd, …)
       seed = seed + delta
```

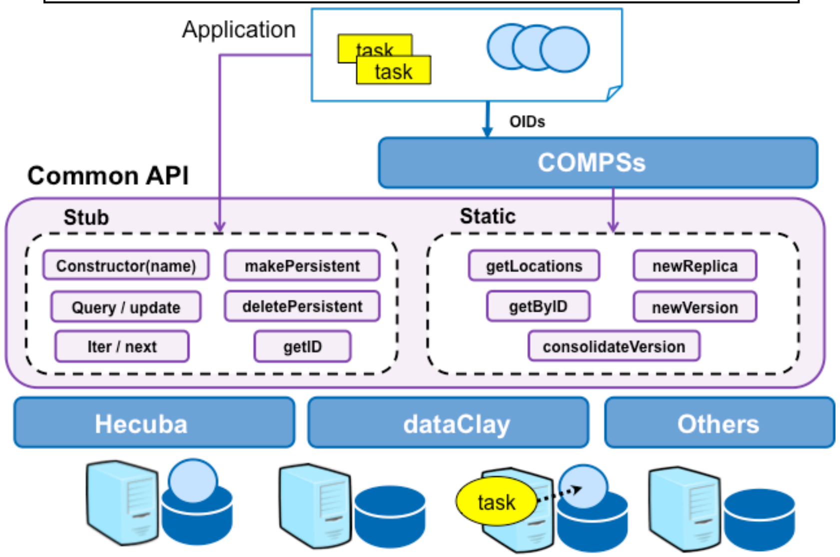
PyCOMPSs integrated with persistent storage

```
from pycompss.api.task import task

@task()
def cc_surrogate_range(block_i, block_j, nd, correlation,
seed, num_surrs, num_bins, maxlag):

...
```

```
import sys

neuron_data_name = sys.argv[1]
correlation_name = sys.argv[2]

nd = NeuronData(neuron_data_name )
correlation = Correlation()
correlation.make_persistent(correlation_name)

 seed = 2398645
 delta = 1782324

for block_i in nd.spikes.keys():
   for block_j in nd.spikes.keys():
       cc_surrogate_range(block_i, block_j, nd, …)
        seed = seed + delta
```

- Data remains persistent
  - Can be shared by several applications
    - Producer/Consumer
    - In-situ data-processing
  - Can remain after execution
  - Can be deleted by another application

- Implementation
  - Hecuba backend can transparently map Python dictionaries into Cassandra tables
  - Python iterators redefined for blocking



Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

Summary: BSC vision

Applications

Regular programming languages
+ light API

Intelligent runtimes

Storage

Computing

- Use of regular traditional programming languages
  - Python, Java, C → **COMPSs**
  - C, C++, Fortran → **OmpSs**
- Tight, natural integration of Concurrency and data model
  - Data flow annotations
  - Persistent objects
    - Self-contained objects → **dataClay**
    - Key-value databases → **Hecuba**
    - …

Traditional look and feel …

… revolutionary under the covers

www.bsc.es

Barcelona Supercomputing Center

Centro Nacional de Supercomputación

Writing Efficient Computational Workflows in Python

compss.bsc.es