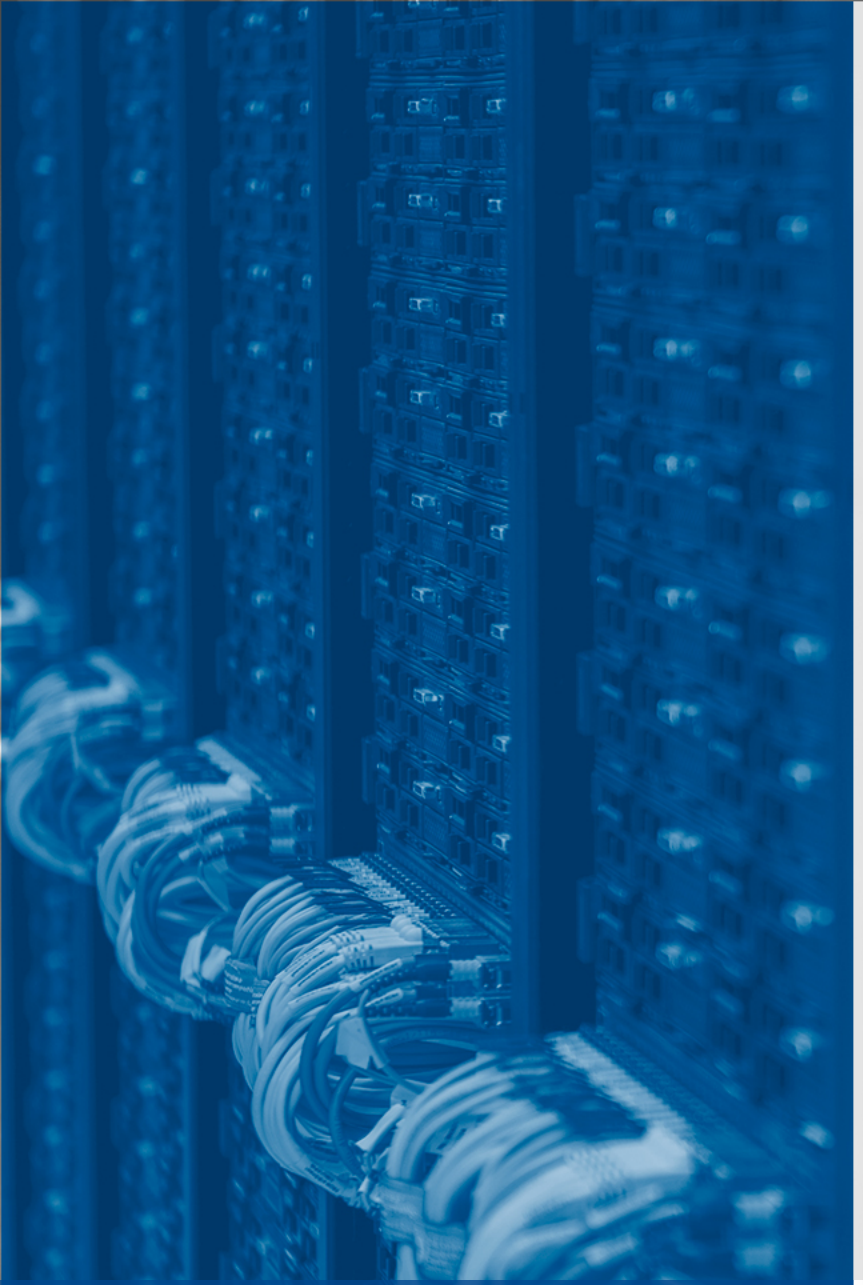


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Evolving COMPSs to embrace the Compute Continuum

Francesc Lordan

15/09/2022 - Cáceres

RED ESPAÑOLA DE SUPERCOMPUTACIÓN

16th Users Conference

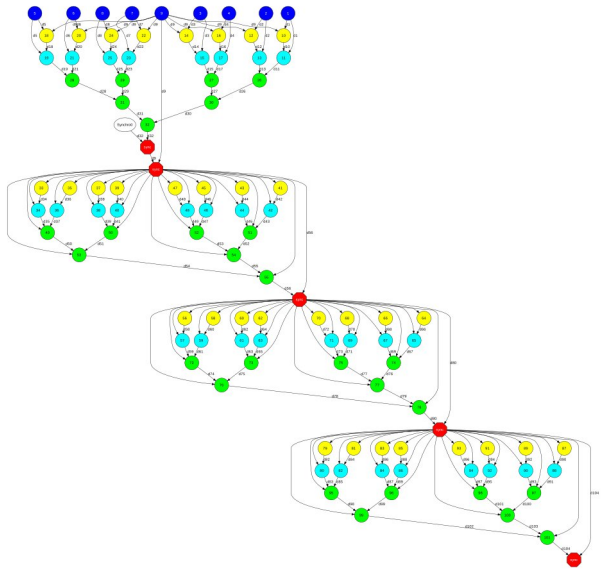
COMP Superscalar (COMPSs)

http://compss.bsc.es
https://compss.readthedocs.io

- Programming model for distributed computing
  - Sequential programming
  - Agnostic of the computing infrastructure
  - General purpose programming language

```
@task()
def increment(counterFile) :
   // Parse the content of a file and increment the value
   …

def main():
   increment(counter1)
   increment(counter2)
   increment(counter3)
   printCounters(counter1, counter2, counter3);
```

- At runtime, an intelligent engine
  - Identifies tasks with dependencies and builds a workflow
  - discover the implicit parallelism
  - distributes the workload



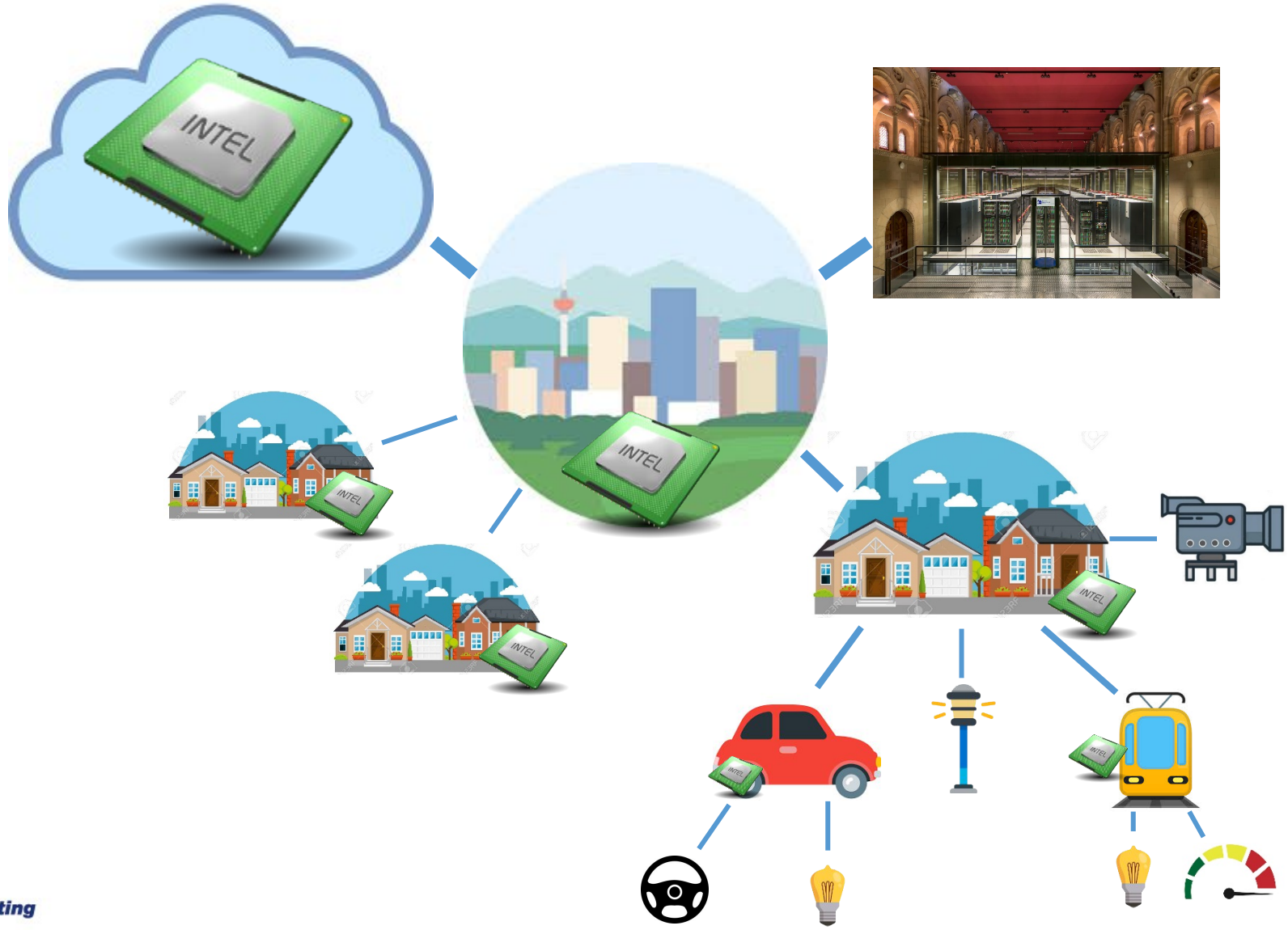
- Tasks => function invocation
  - Code annotations/hints

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Workflow Managers' Traditional Approach

- The infrastructure is stable (except for sporadic network disruptions)
  - May scale-in/out depending on the workload in a controlled manner
- There is a single instance of the application running
- 1 node (master) decomposes the application and spawns tasks
- The master is aware of the whole infrastructure and orchestrates the execution of the tasks
  - Schedules task executions on the available resources
    - Runs tasks on the local resources
    - Submits the task execution to the remote nodes
  - Orders data transfers

Compute Continuum



Compute Continuum

Three computation scenarios:

- Sense-process-actuate
- Stream processing
- Batch jobs

Compute Continuum's Approach

- The infrastructure is dynamic because of mobile devices
- There are several requests of the same service running at a time
- Any node of the infrastructure may trigger a code execution and generate workload
- The execution of tasks is a shared responsability
  - Schedules task executions on the available resources
  - Share data

Solution overview

- Computation triggered at any node of the infrastructure **Standalone** devices (agents)
- Multiple requests at a time **Functions-as-a-Service**
- Hardware heterogeneity **Serverless** functions
- Software heterogeneity and device multi-tenancy **Virtualized** environments
- Resource exploitation Convert functions into task-based **workflows**
- Application development complexity Single Programming Model

Resource hierarchies

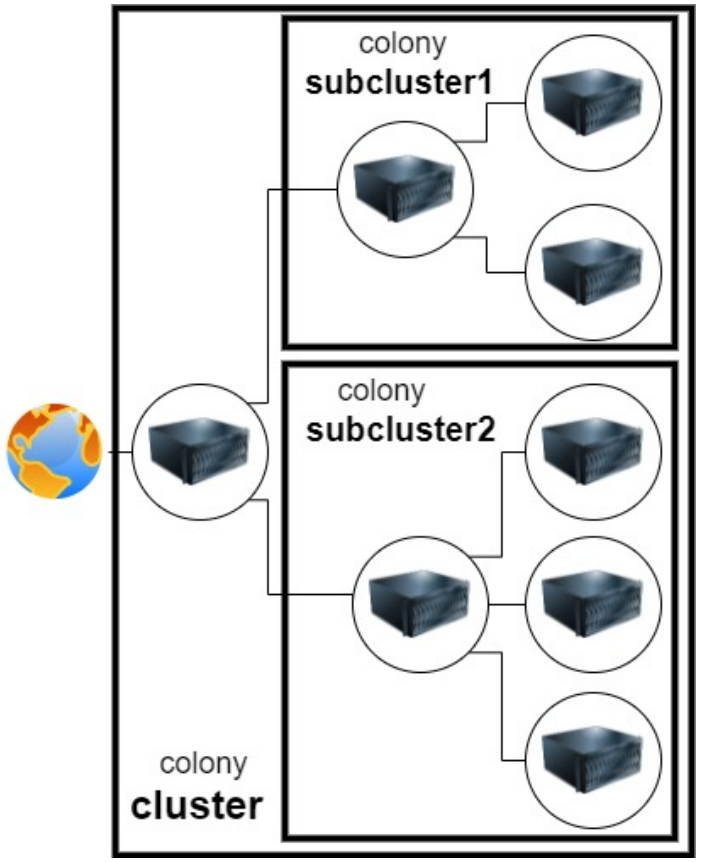
Agents organize resembling organic colonies

- Each member is an autonomous individual capable of processing information
- Members offer the colony their embedded resources to run functions in a serverless, stateless manner
- Members receive a platform where to offload computation

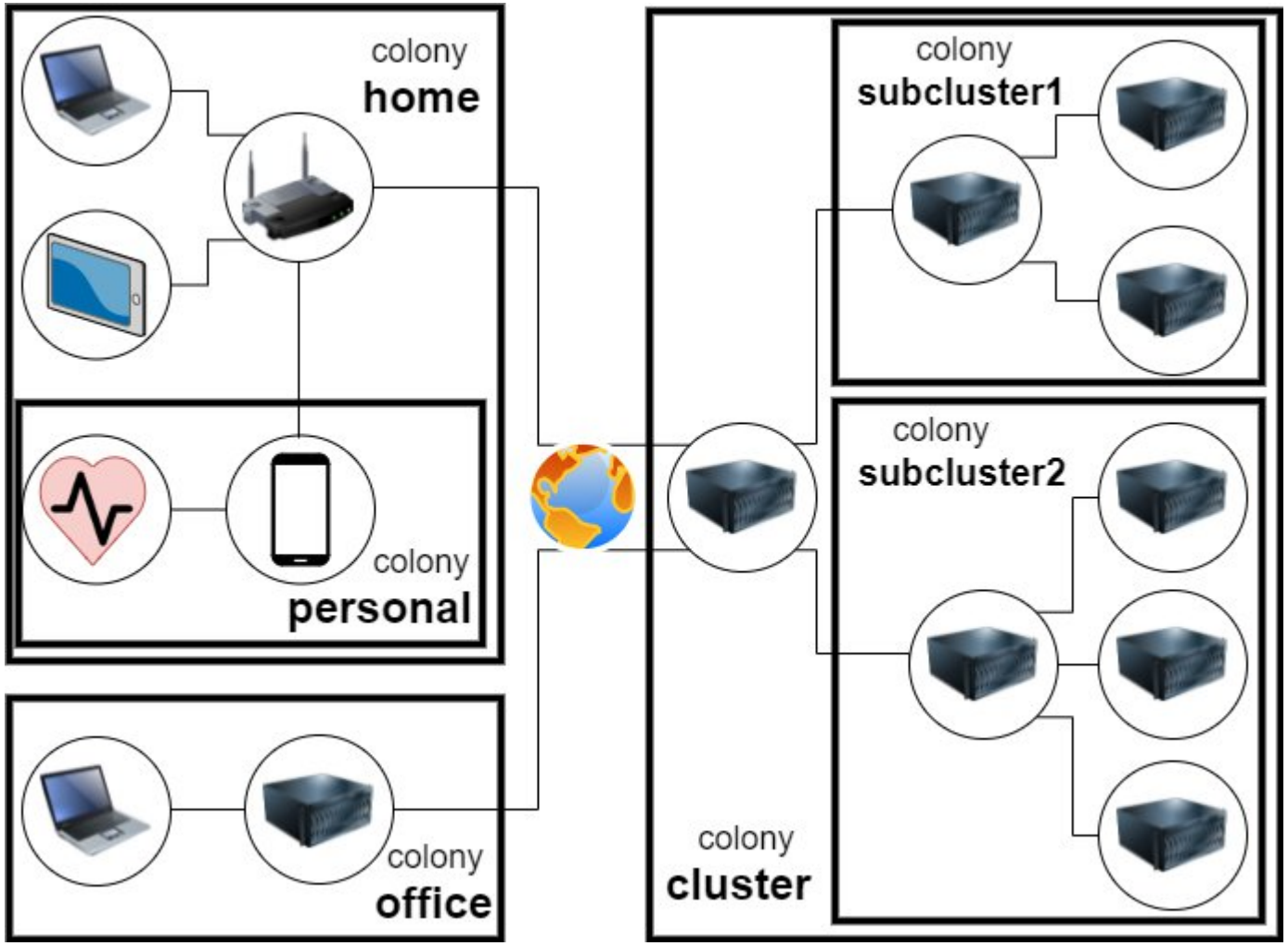
Colonies can be divided recursively forming hierarchies

Each colony selects one node to act as

- Nexus
- Interaction gateway to the Colony

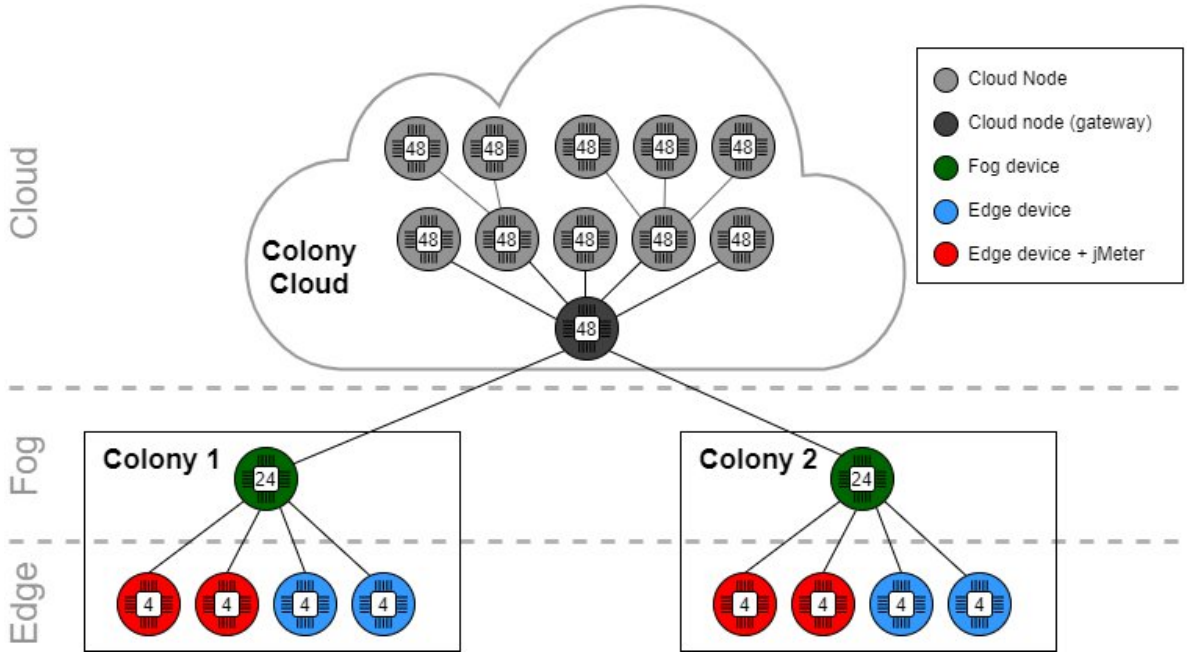


Hierarchy Malleability



Preliminary Results - Classification Service

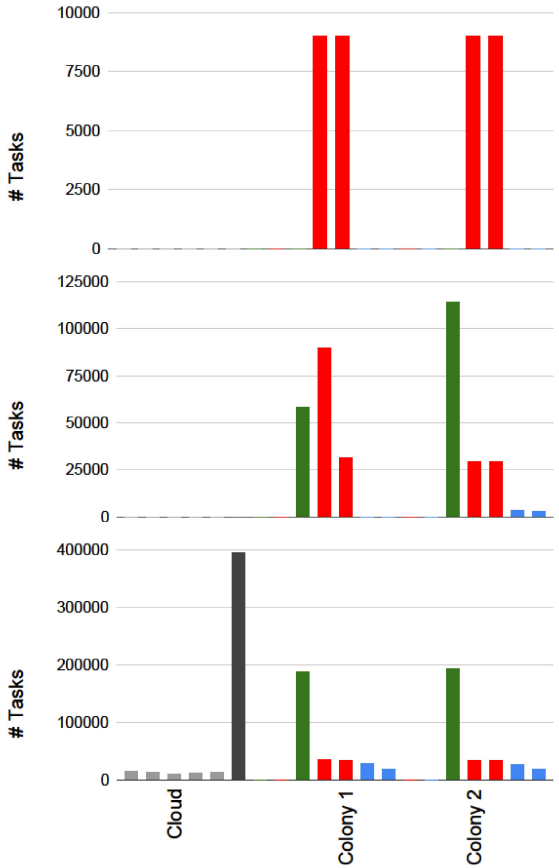
(sense-process-actuate scenario)



10 users

100 users

300 users



Avg. Response time

104 ms

Avg. Response time

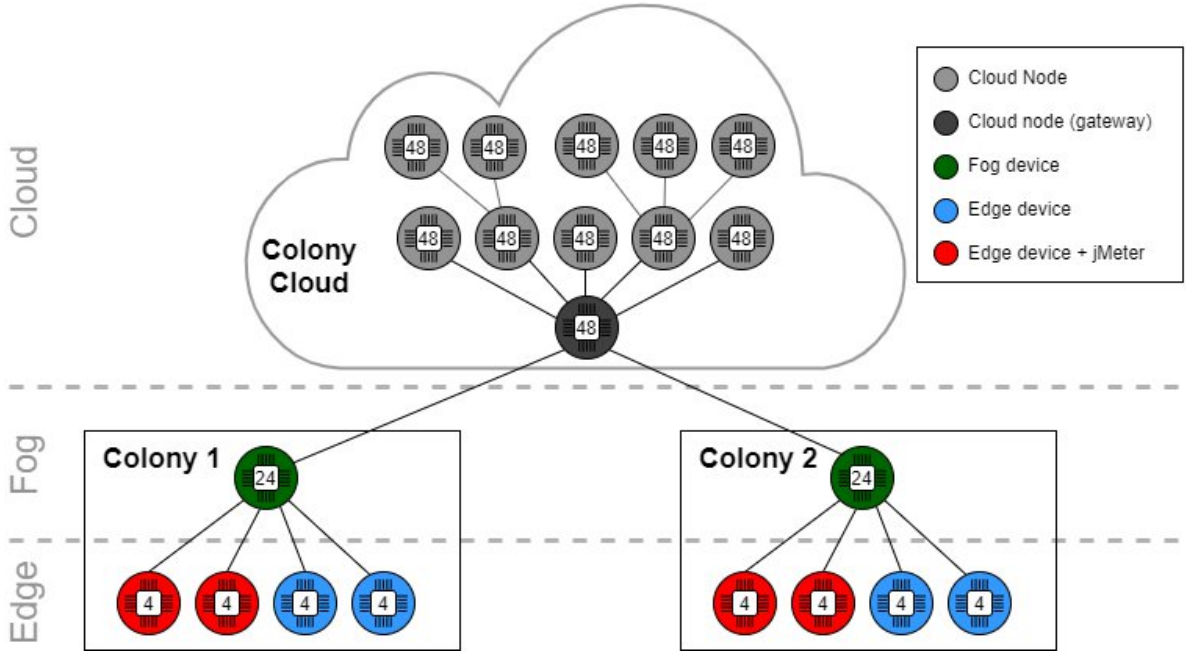
104 ms

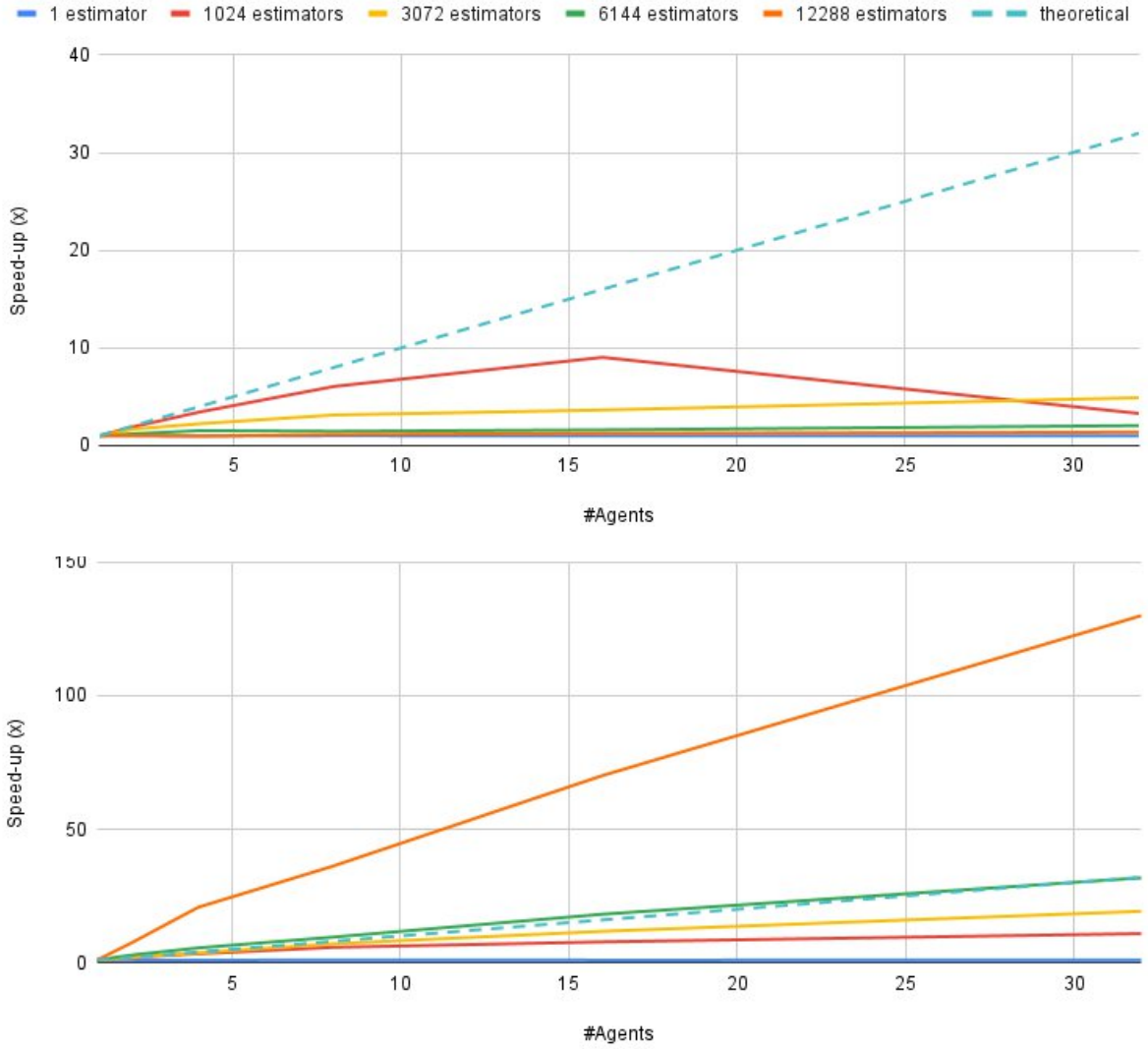
Avg. Response time

109 ms

Preliminary Results - Classification Service

(batch processing scenario)





Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Projects where this technology is being applied

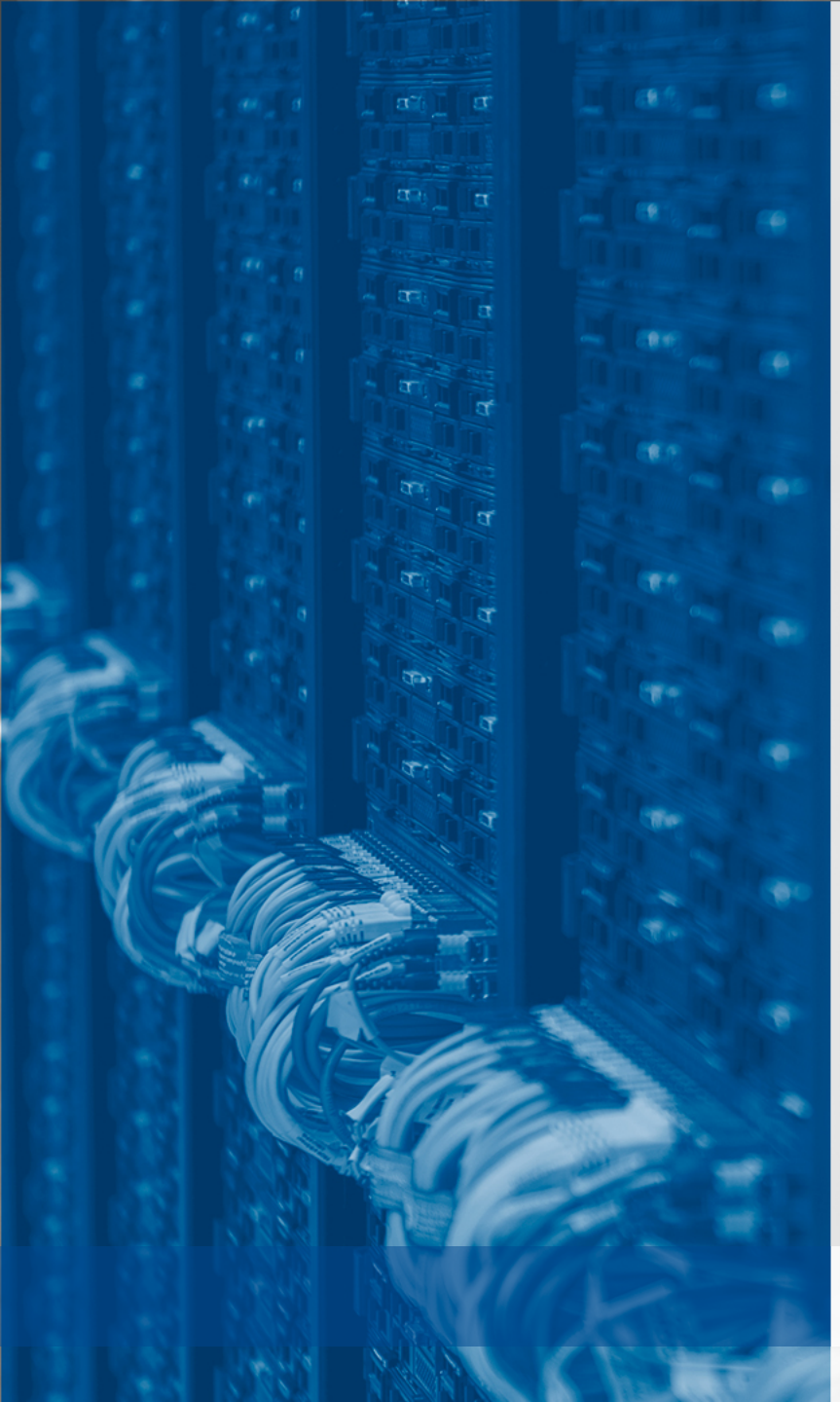
- Personalised alerts and recommendation system for airport passengers

- GeoSpatial Decision Support system

- Mask detection (Image processing)
- AF detection (eHealth)

Open Challenges

- Agent Deployment
- Resource discovery and hierarchy setup
- Scheduling policies
  - QoS-based for real-time
- Data Management
  - IP, privacy
  - Integration with data sharing mechanisms (currently, support for Hecuba and dataClay)
- Eventing: data triggers computation
  - Monitoring changes on storage services (MINIO - UPV)
  - Monitoring resources
  - External services such as IFTTT
  - Notifications from publish-subscribe system

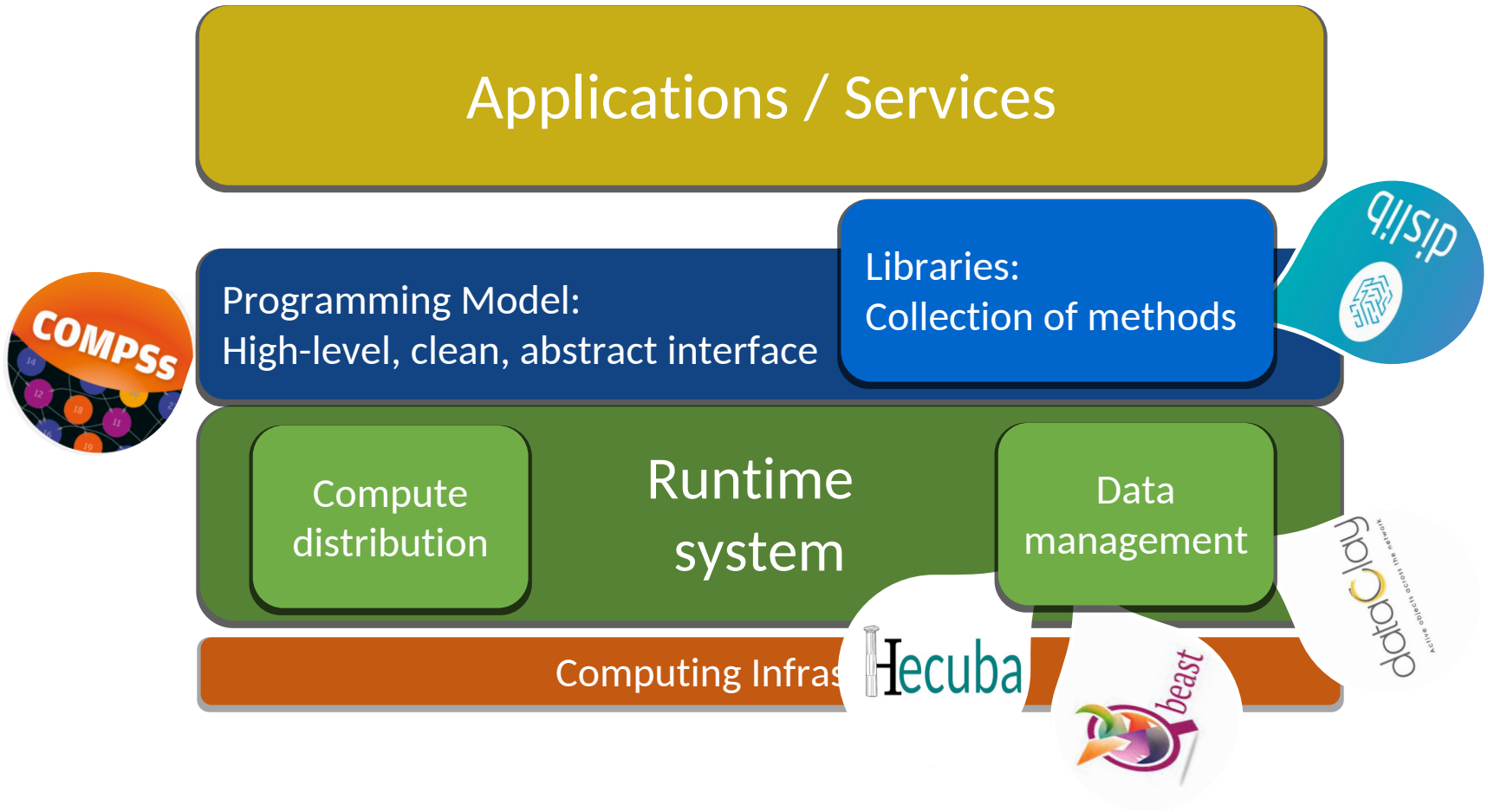


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

For further information about the project:

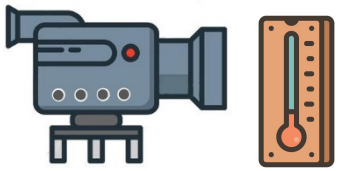
francesc.lordan@bsc.es

Tools by WDC group



Device-Edge-Cloud Infrastructures

3 types of elements composing the system:



Sensors (Events Detectors/Data Generators)

elements monitorizing certain condition and informing about it

- Continuosly generating information Streams of data
- Eventually notifying a change Events

Computing and Storage elements

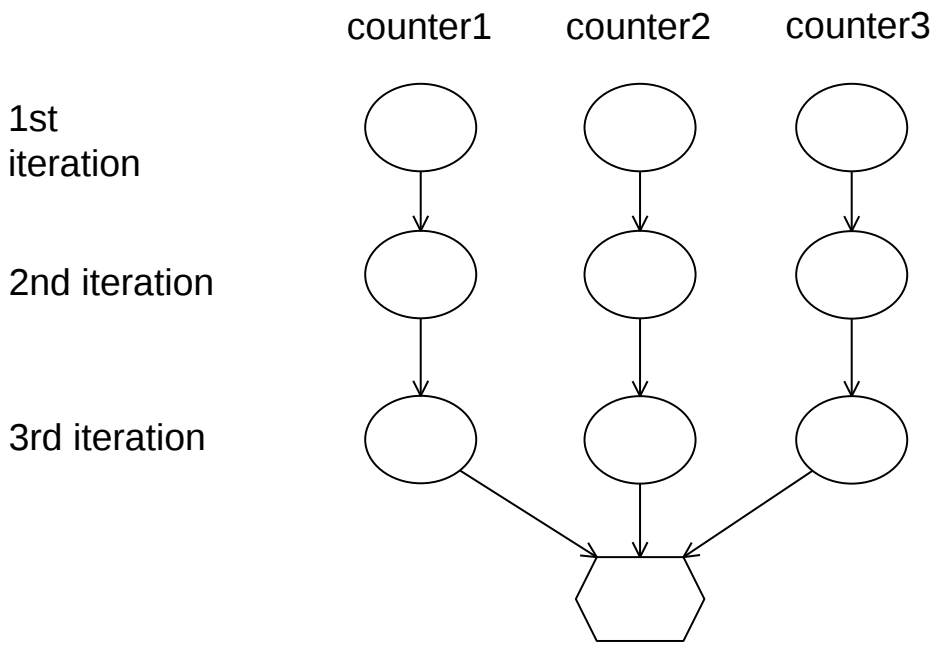
elements within the infrastructure with ability to process and transform information

Actuators

elements on which the platform realizes actions (changing its state)

Programming model example (Python)

```
class Simple(object):

    @task( counterFile = FILE_INOUT)
    def increment(counterFile) :
        // Parse the content of a file and increment the value
        ...

    def main():
        for i in range(3) :
            increment(counter1)
            increment(counter2)
            increment(counter3)

        printCounters(counter1, counter2, counter3);
```



Task types (Python)

```python
@binary(binary="mybinary.bin")
@task()
def binary_func():
    pass

@constraint(computingUnits="2")
@binary(binary="otherbinary.bin")
@task()
def binary_func2():
    pass
```

```python
@mpi(processes=4, processes_per_node=2)
@task()
def layout_test_with_all():
    from mpi4py import MPI
    rank = MPI.COMM_WORLD.rank
    return rank
```

```python
from pycompss.api.implement import implement

@implement(source_class="sourcemodule", method="main_func")
@constraint(app_software="numpy")
@task(returns= list)
def myfunctionWithNumpy(list1, list2):
    # Operate with the lists using numpy
    return resultList

@task(returns=list)
def main_func(list1, list2):
    # Operate with the lists using built-in functions
    return resultList
```

Constraints (Python)

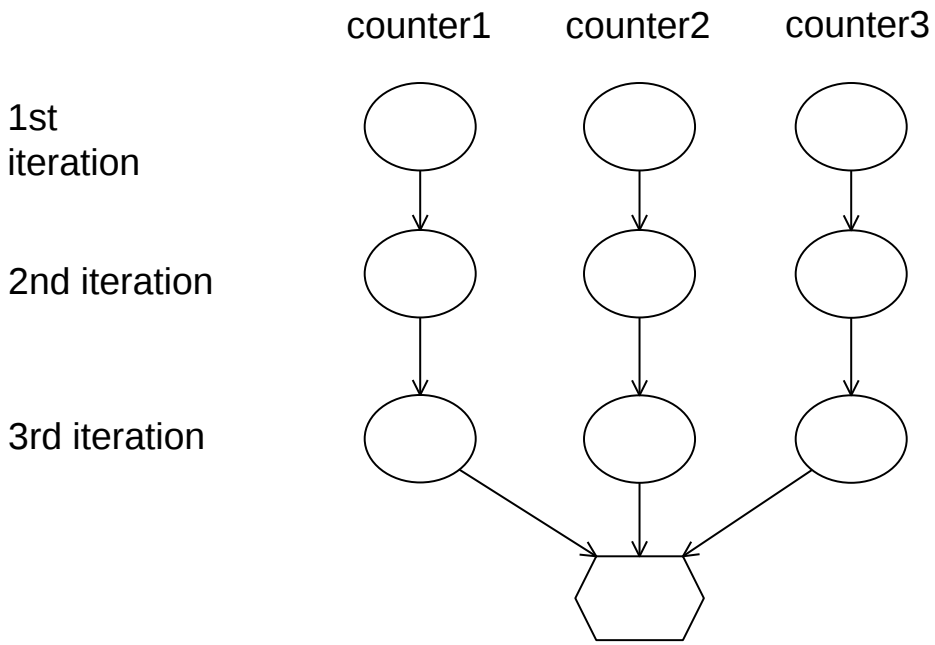
```
@constraint(computing_units="4")
@task(c=INOUT)
def func(a, b, c):
   c += a * b
   ...
```

```
@constraint(computing_units="4",
        app_software="numpy,scipy,gnuplot",
        memory_size="$MIN_MEM_REQ")
@task(c=INOUT)
def func(a, b, c):
   c += a * b
   ...
```

```
@constraint(processors=[{'processorType':'CPU', 'computingUnits':'1'},
                {'processorType':'GPU', 'computingUnits':'1'}])
@task(returns=1)
def func(a, b, c):
   ...
   return result
```

Programming model example (Java)

```
public class Simple {

   public static void increment(String counterFile) {
      // Parse the content of a file and increment the value
      ...
   }

   public static void main(String[] args) {
      for (i = 0; i < 3; i++) {
         increment(counter1);
         increment(counter2);
         increment(counter3);
      }
      printCounters(counter1, counter2, counter3);
   }
}
```

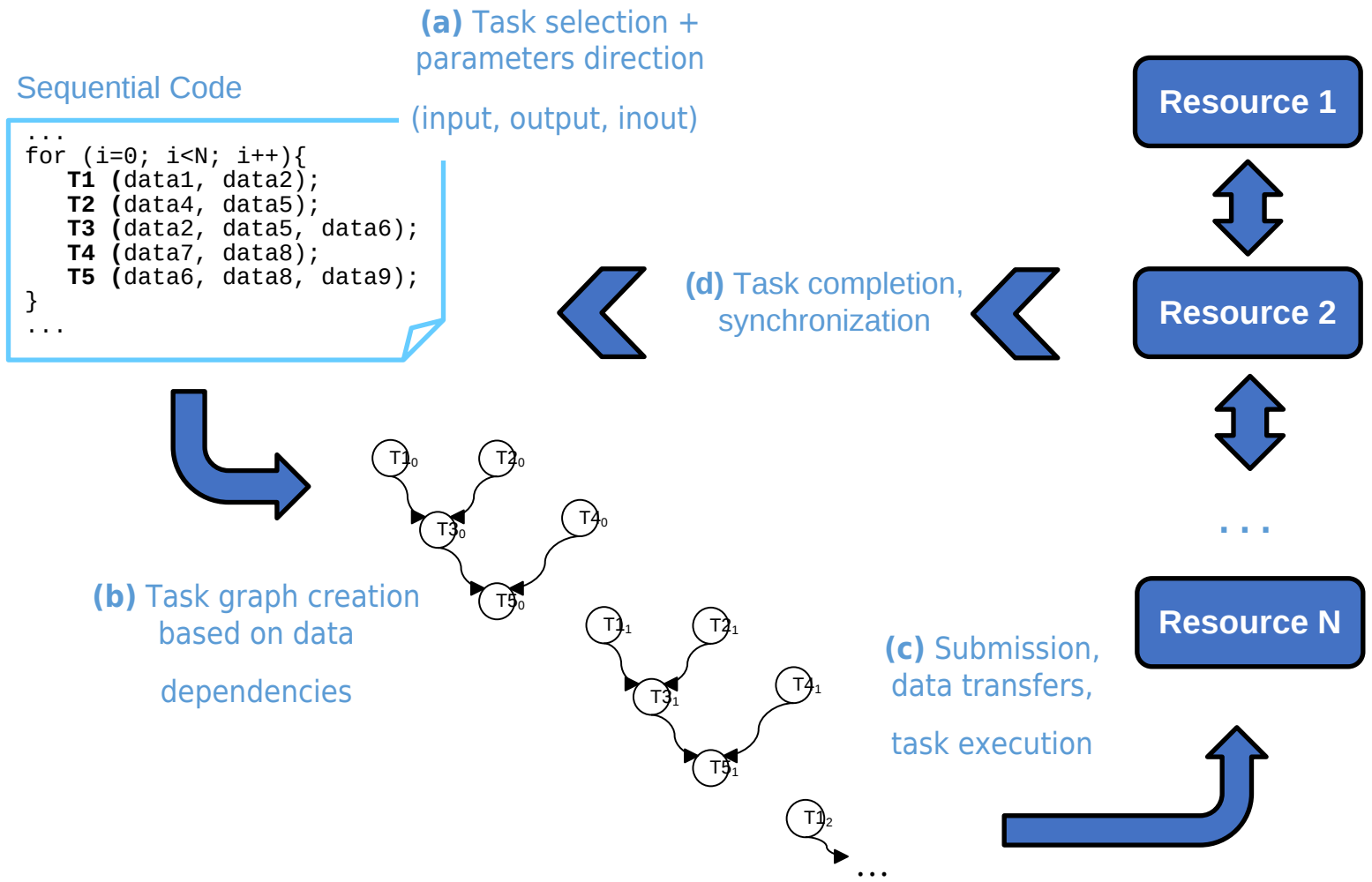



```
public interface SimpleItf {

   @Method(declaringClass = "es.bsc.Simple")
   void increment(
            @Parameter(type = FILE, direction = INOUT)
            String counterFile
   );

}
```

Implementation

Parameter metadata

Programming model runtime: summary



Constraints (Java)

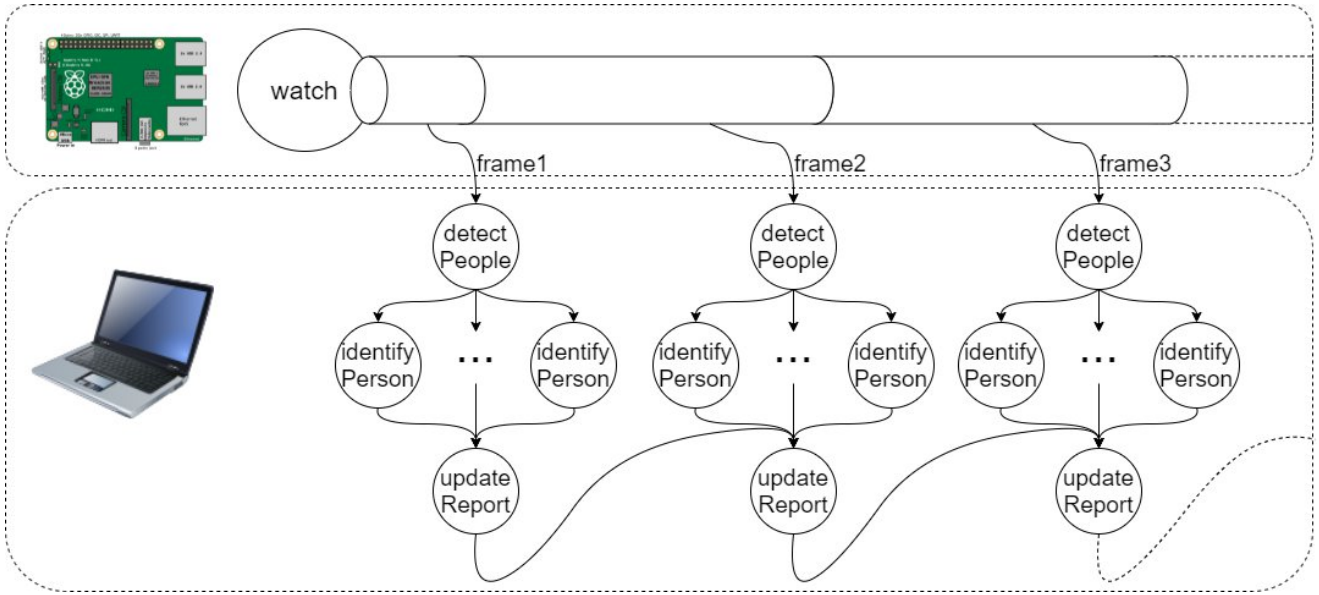
- Allow the application developer to exploit system's heterogeneity

- Specifying:
  - Processor
    - Number of computing units (cores)
    - Architecture
    - Type
  - Memory
    - Size
  - Storage
    - Disk Size
    - Bandwith
  - Operating System
  - Installed Software

```
public interface SimpleItf {

   @Method(declaringClass = "Simple")
   void increment(
            @Constraints(computingUnits = 4, processorArchitecture = RISC-V)
            @Parameter(type = FILE, direction = INOUT)
            String counterFile
   );

}
```

Preliminary Results – Real-time Video Processing

(stream processing scenario)

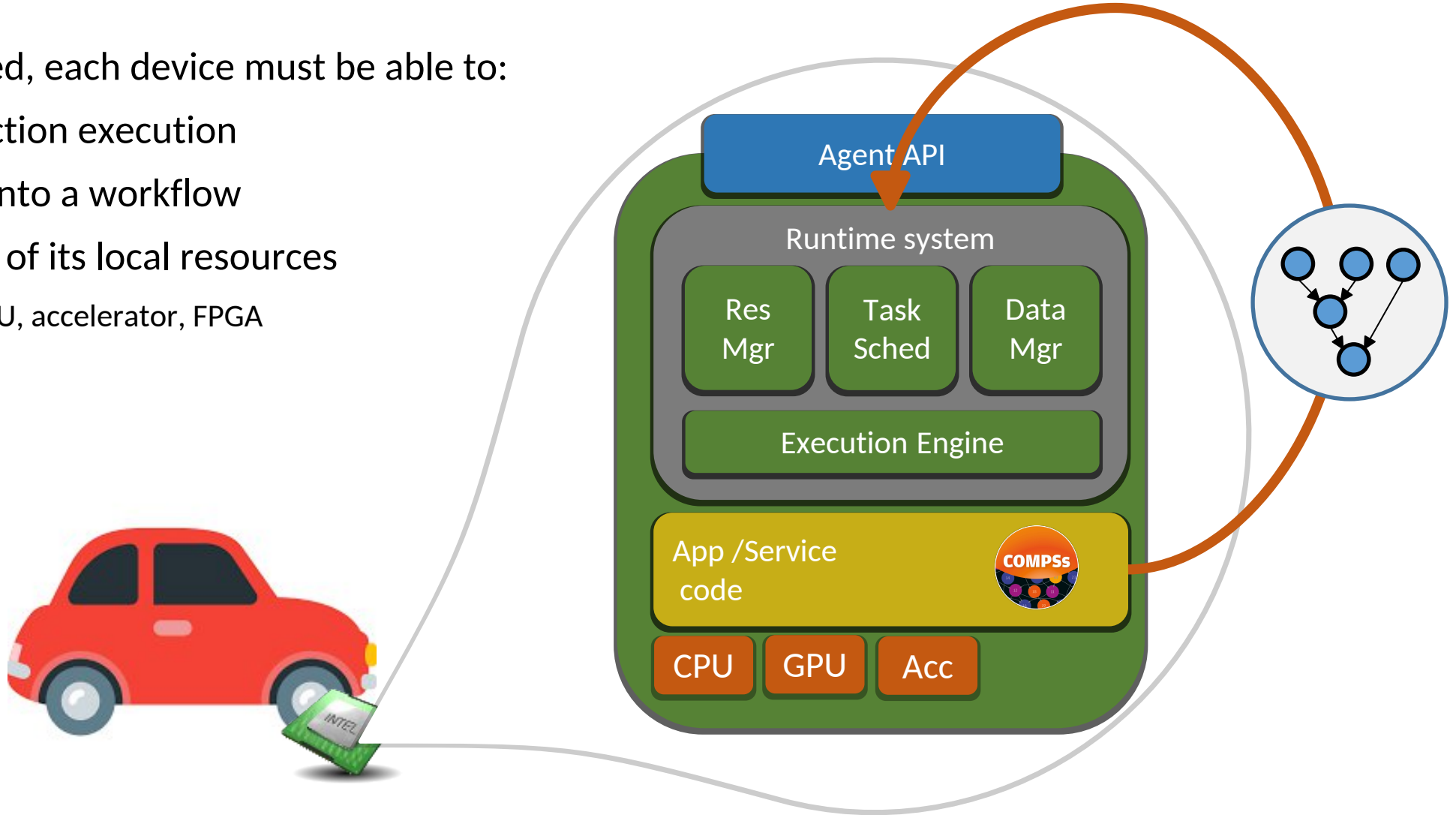


- rPi alone:
  - 1 frame each 41 seconds
  - Frame rate: 0,024 fps
- Colony:
  - rPi: 1 frame 353 ms
  - Frame rate: 2,79 fps

Standalone Agent

Even when isolated, each device must be able to:

- Start a function execution
- Convert it into a workflow
- Run on top of its local resources
  - CPU, GPU, accelerator, FPGA



Agents interaction

Agents can offload part of the computation onto other agents by adding them onto their resource pool