ascender

www.proyecto-ascender.com

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXTRACT
A distributed data-mining software platform for extreme data across the compute continuum

The Compute Continuum: An Efficient Use of Edge-to-Cloud Computing Resources

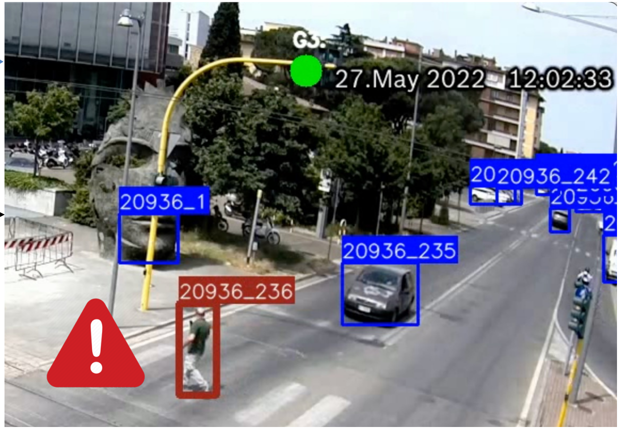
Eduardo Quiñones

{eduardo.quinones@bsc.es}

Ada-Europe, Barcelona, 2024

From Data to (Real-time) Knowledge

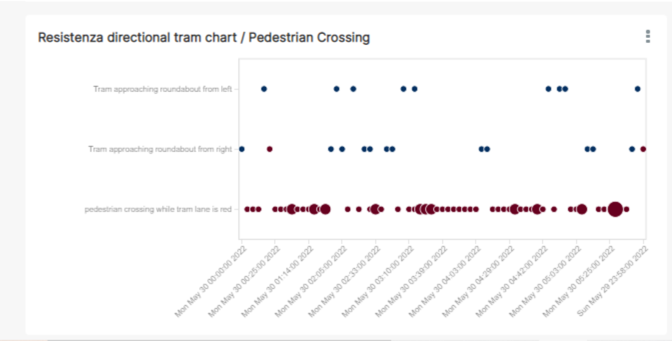
Data **Knowledge**



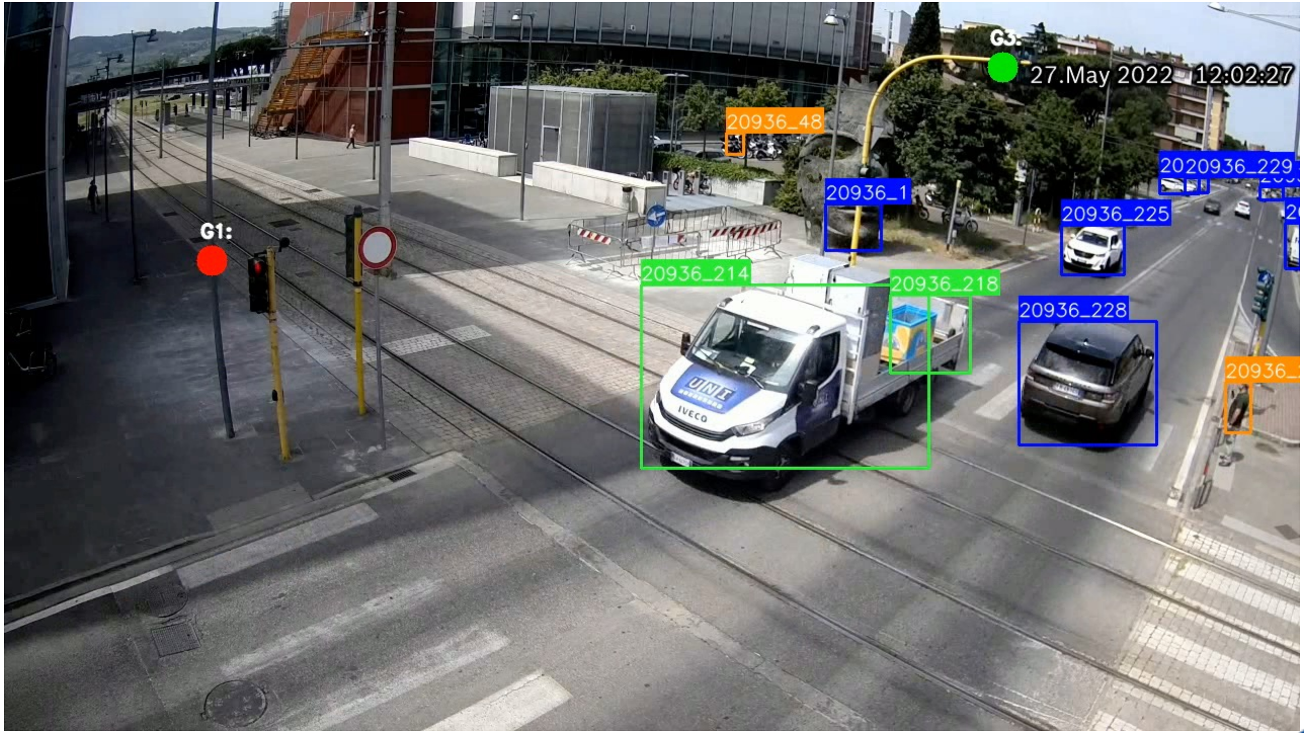
Object Detection & tracking → Semantic Extraction → **Hazard Detection: PersonAtCrossWalk-SemaphoreRed**

Guaranteed End-to-end response time

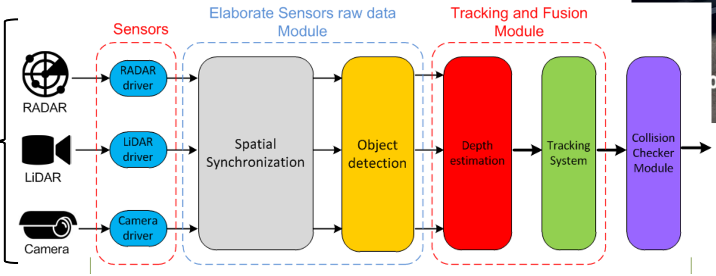
Data-Analytics Workflow executed by city resources

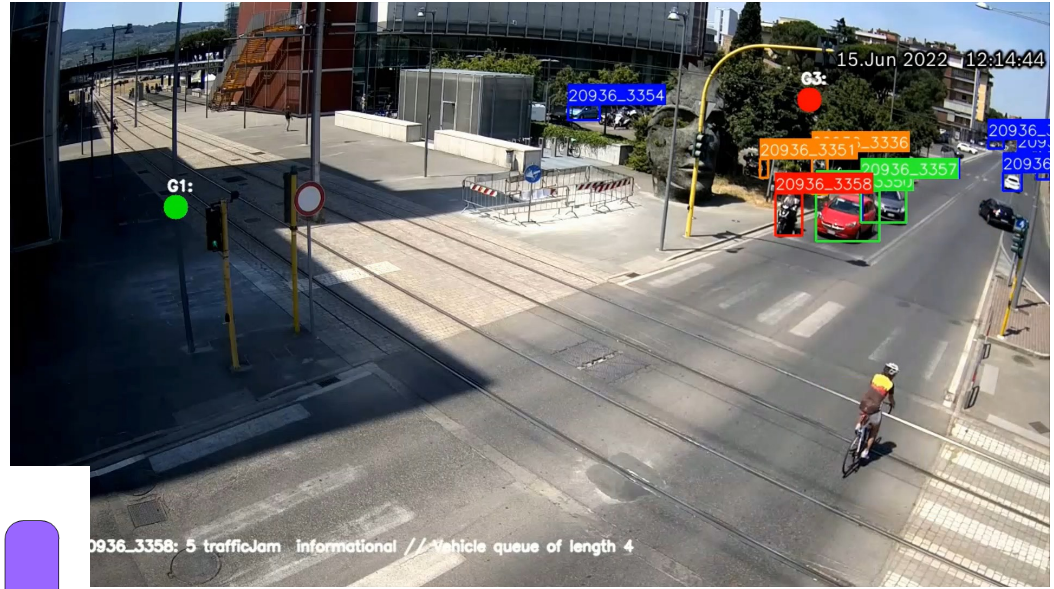


From Data to (Real-time) Knowledge



From Data to (Real-time) Knowledge



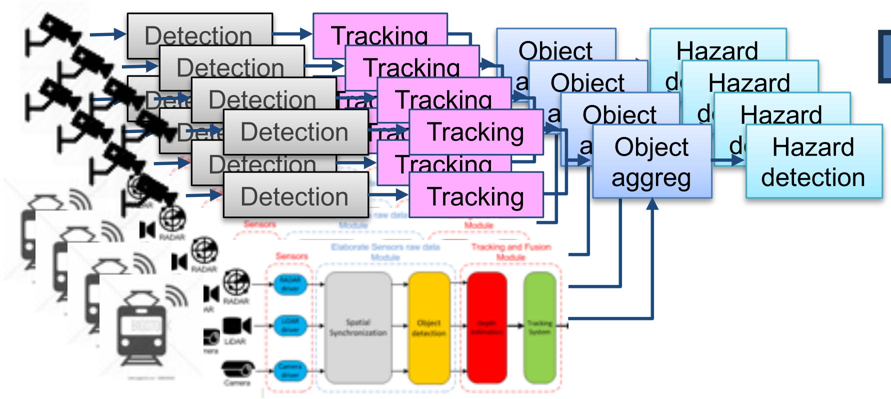


Data-Analytics Workflow executed by the tramway

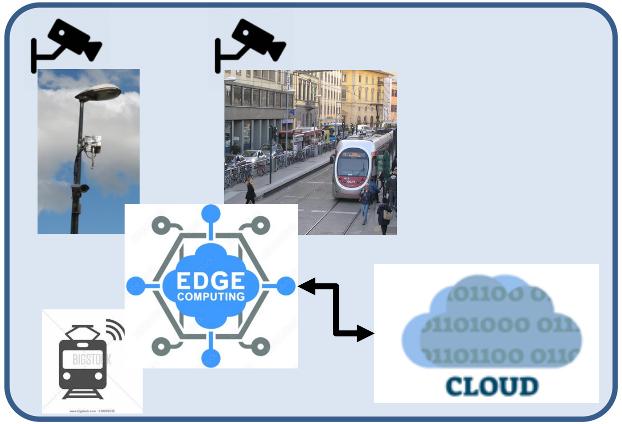
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Edge and Cloud Computing

Sensing capabilities of vehicles and cities can be effectively combined to identify hazardous situations



Deploy and execute

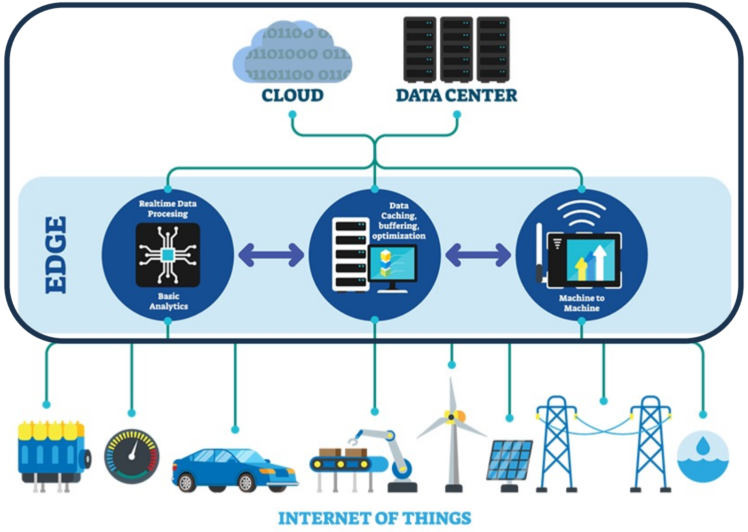


Computing/Communication Resources

Compute Continuum: From Edge to Cloud

Abstracts the edge/cloud complexity

- Edge computing allows moving analytics close to data-sources
  - Enables faster **real-time processing**, higher privacy control and lower network costs
  - The use of powerful **heterogeneous and parallel embedded processor architectures** becomes fundamental
- Cloud computing provides computational intensive, batch processes and storage

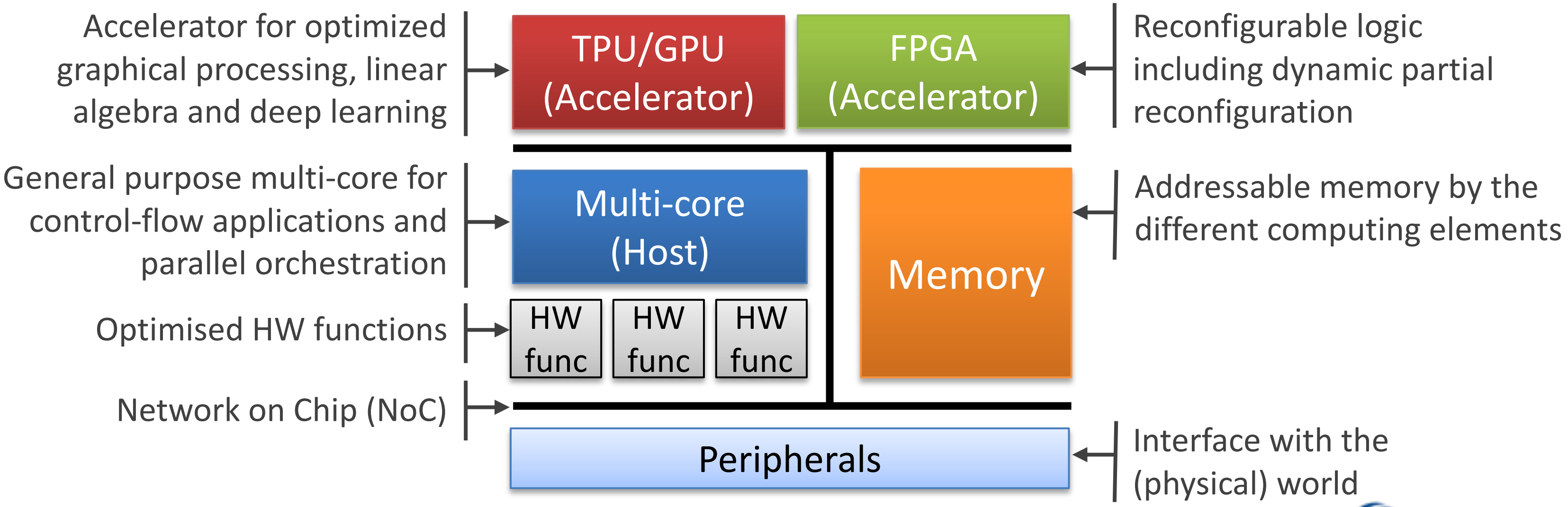


Compute Continuum

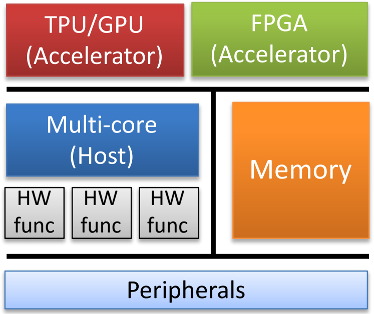
How are the data-analytics workflows <u>**developed, deployed and efficiently executed**</u> on highly heterogeneous **computing/communication resources?**

Edge Computing

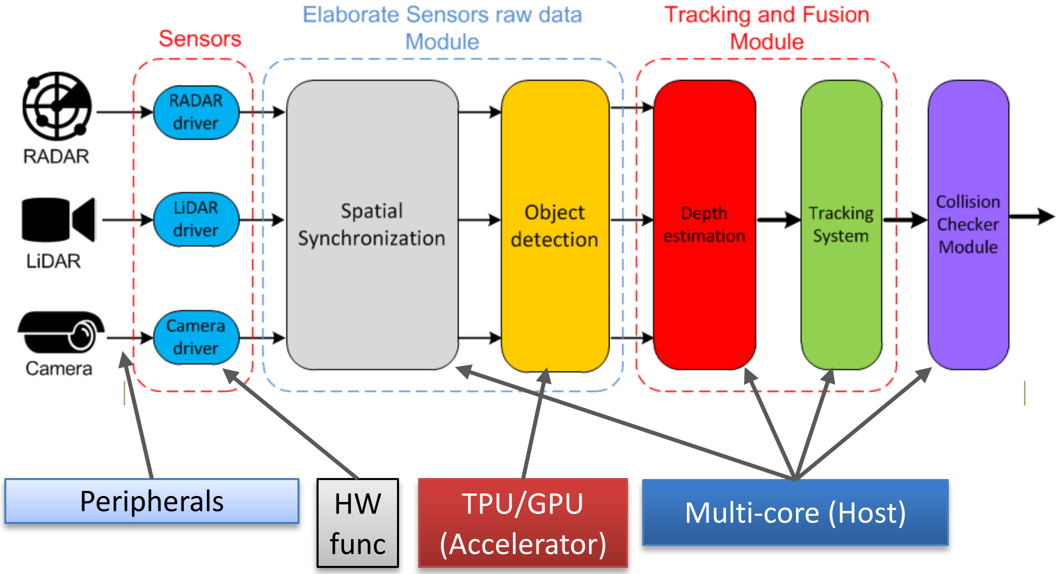
Host-centric paradigm: The parallel computation is orchestrated by the general-purpose multi-core



Deploying on Edge Computing

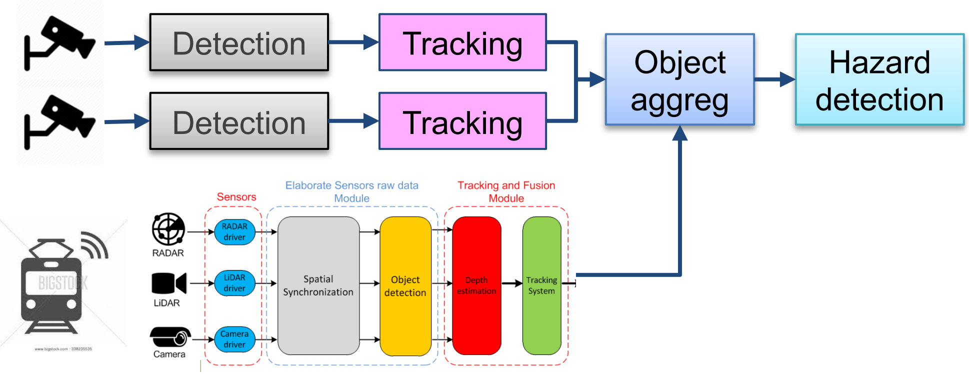


NVIDIA CUDA
RTEMS
AdaCore
POSIX pthreads
OpenMP

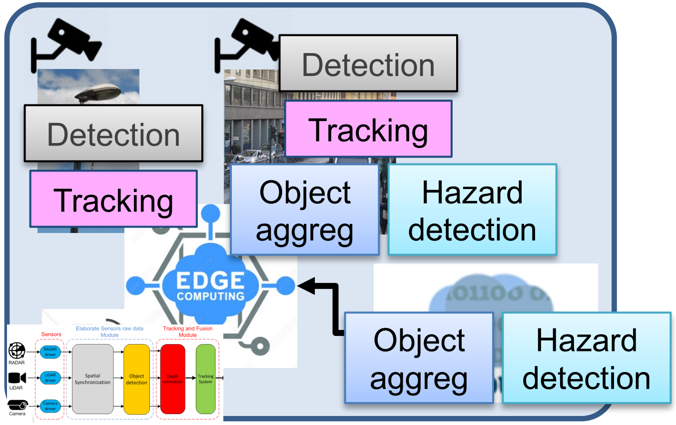


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Deploying on Cloud Computing



Computing/Communication Resources



kubernetes S3 docker MINIO Skupper LITHOPS

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

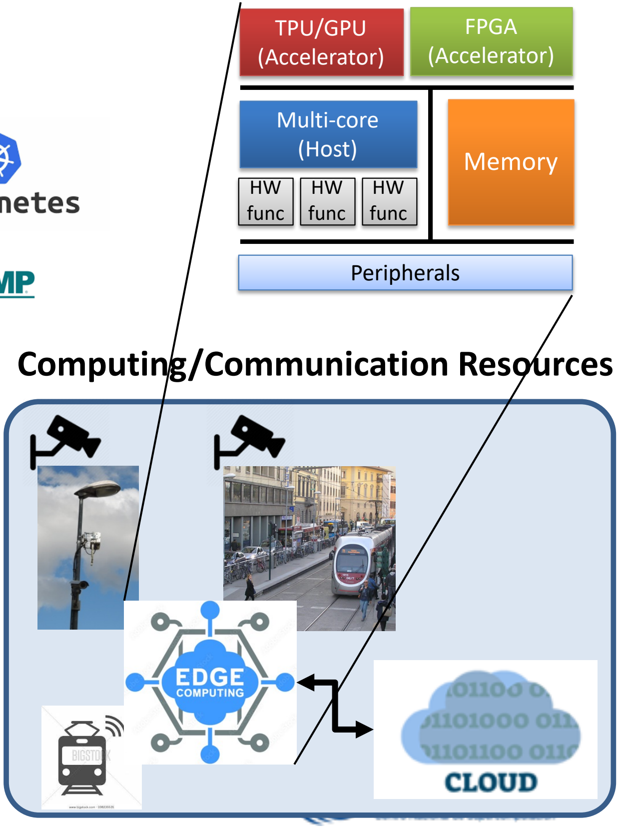
Non-functional Requirements (NFR)

- Inherited due to the cyber-physical interactions, e.g.,
  - **Real-time:** The end-to-end response time (from sensor to actuator) must be within a given time budget
  - **Power/Thermal:** The energy/temperature of the computing elements must be within a given budget due to power supply/operational environment limitations
  - **Safety:** Built guaranteeing the correctness and integrity of its operation
  - **Security:** Prevent external elements not to affect the correctness and integrity of the system

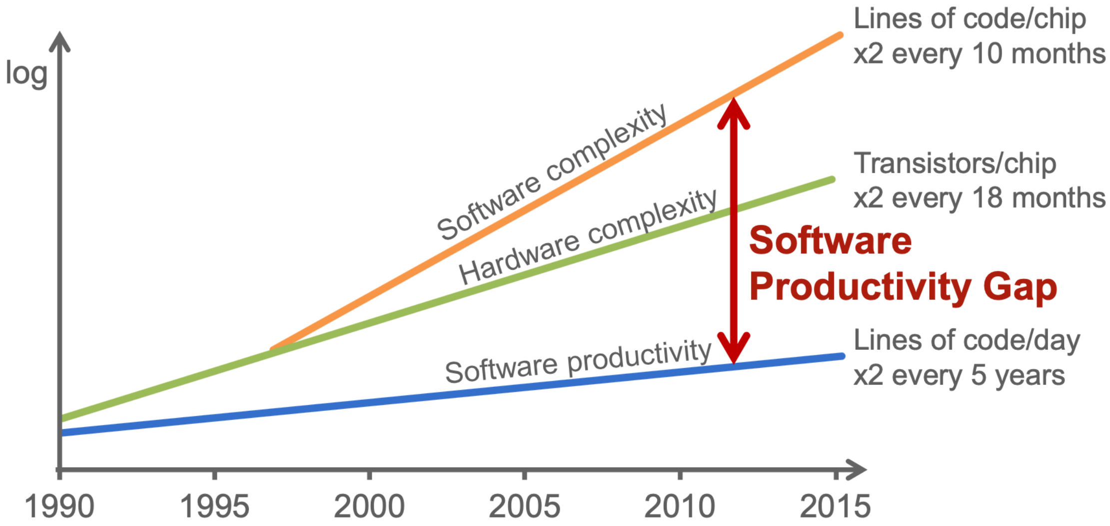
How to develop workflows on the compute continuum?

1. **Exploit the parallel performance capabilities** of the (different) processor architectures
2. **Efficiently distribute** the data-analytics workflow across the compute continuum
3. **Guarantee** functional correctness and the non-functional requirements

POSIX pthreads



SW Development Complexity



Source: ITRS & Hardware-dependent Software, Ecker et al., Springer

Task-based Parallel Programming Models

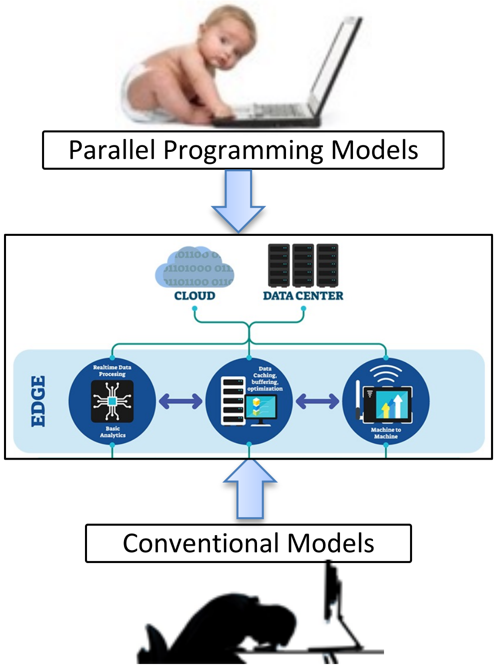
Parallel Programming Models

- A set of programming elements to **describe** the parallel behaviour of an application and **abstract** the complexities of the underlying parallel platform
  - **Granularity level of parallelism** exploited: instruction, statement, loop, procedural
  - **Synchronization** model: coarse-grain, fine-grain
  - **Execution model**: fork-join, thread-pool, etc.
  - **Memory model**: Shared, distributed
- Commonly built on top of a base programming language

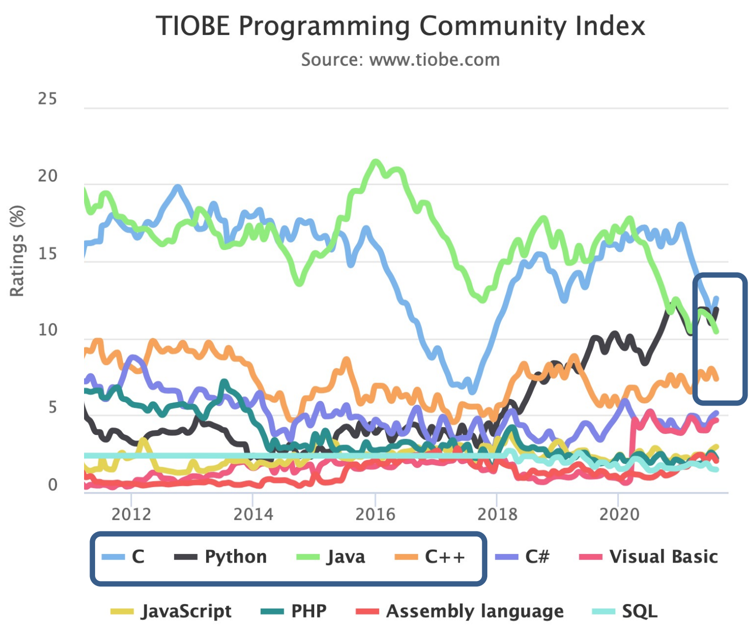
Parallel Programming Models

Mandatory to enhance **productivity**

- **Programmability.** Abstracts the parallelism while hiding the underlying computing platform complexities
- **Portability/scalability.** The same source code is valid in different parallel platforms
- **Performance.** Rely on run-time mechanisms to exploit the performance capabilities of parallel platforms



Parallel Programming Models and Programming Languages



Model	Base Language	Type of PPM	Type of architect	Type of Parallelism
CUDA	C/C++, Python	HW-centric	NVIDIA GPU	Struct/ Unstruct
OpenCL	C/C++	App-centric	GPU/ FPGAs	Struct
OpenMP	C/C++	Parallel-centric	Shared mem	Struct/ **Unstruct**
Pthreads	C/C++	Parallel-centric	Shared mem	Unstruct
MPI	C/C++, Python	Parallel-centric	Distributed mem	Unstruct
COMPSs	C++, Java Python	Parallel-centric	Distributed mem	**Unstruct**
Spark	Java, Python	Parallel-centric	Distributed mem	Struct
Ray	C++,Java Python	Parallel-centric	Distributed mem	Unstruct

Why OpenMP?

Mature language constantly reviewed (last release Nov 2024, v6.0)

- Defacto industrial standard in HPC
- Active research community with an **increasing interest** on the edge domain

Productivity in parallel programming

- Portability
  - Supported by many chip vendors
- Programmability
  - Interoperability with other models (CUDA, OpenCL)
  - Allows incremental parallelization

Why COMPSs?

Programming distribute framework highly inspired in the OpenMP tasking model

- The code is annotated to describe task and data dependencies

Productivity in distributed programming

- Performance
- Programmability
  - Interoperability with other programming models (OpenMP)
  - Allows incremental parallelization

You can choose any other task-based parallel programming model you like!

OpenMP Tasking Model

Sequential version

```
void main() {
    int x,y;
    f1(&x,&y);
    f2(x);
    f3(y);
}
```

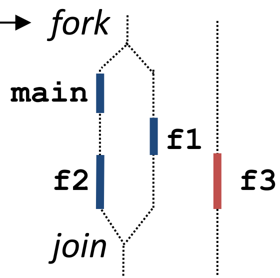
Executes on the host

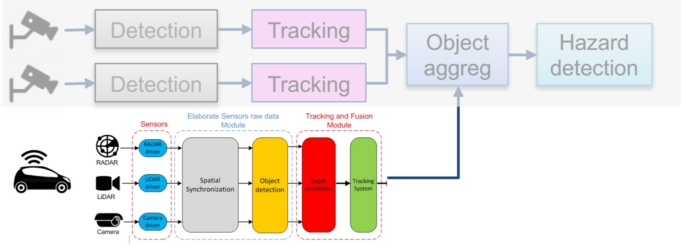
Executes on the accelerator

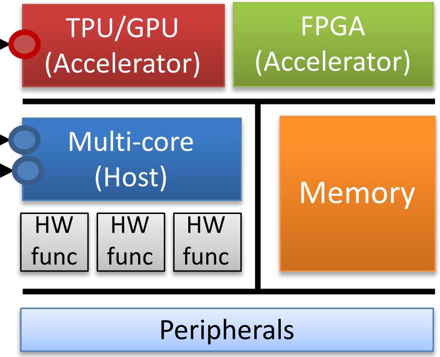
OpenMP version

```
void main() {
    #pragma omp parallel
    #pragma omp single
    {
        int x,y;
        #pragma omp task depend(out:x,y)
        { f1(&x,&y); }
        #pragma omp task depend(in:x)
        { f2(x); }
        #pragma omp target map(to:y) depend(in:y)
        { f3(y); }
    }
}
```

1. Open parallelism







2. Tasks executed on the host

3. Tasks executed on the host and accelerator when f1 completes

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

COMPSs Tasking Model

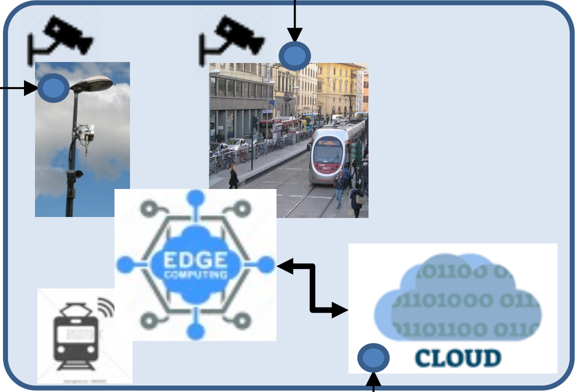
<u>Sequential version</u>

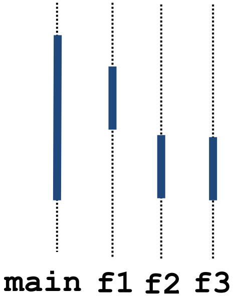
```
def f1():
    …
    return x,y
def f2(x):
    …
def f3(y)
    …
def main():
    x,y=f1()
    f2(x)
    f3(y)
```

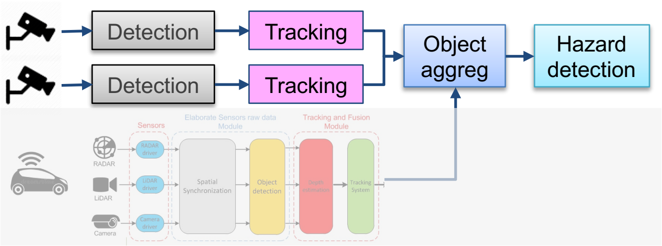
<u>COMPSs version</u>

```
@task(x=OUT,y=OUT)
def f1():
    …
    return x,y
@task(x=IN)
def f2(x):
    …
@task(y=IN)
def f3(y)
    …
def main():
    x,y=f1()
    f2(x)
    f3(y)
    compss_wait()
```

Tasks executed across the compute continuum





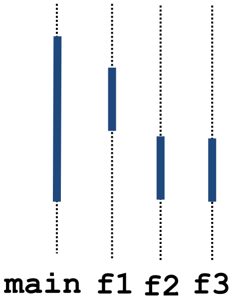


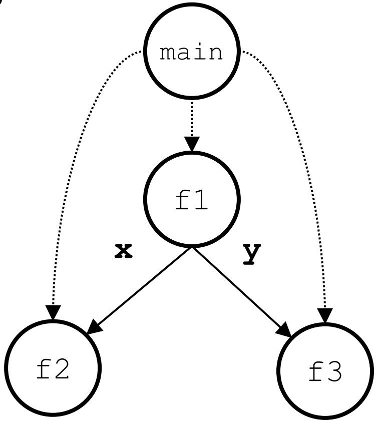
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Task Dependency Graph (TDG)

COMPSs version

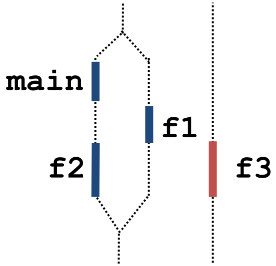
```
@task(x=OUT,y=OUT)
def f1():
    …
    return x,y
@task(x=IN)
def f2(x):
    …
@task(y=IN)
def f3(y)
    …
def main():
    x,y=f1()
    f2(x)
    f3(y)
```





OpenMP version

```
void main() {
    #pragma omp parallel
    #pragma omp master
    {
        int x,y;
        #pragma omp task depend(out:x,y)
        { f1(&x,&y); }
        #pragma omp task depend(in:x)
        { f2(x); }
        #pragma omp target map(to:y) depend(in:y)
        { f3(y); }
```

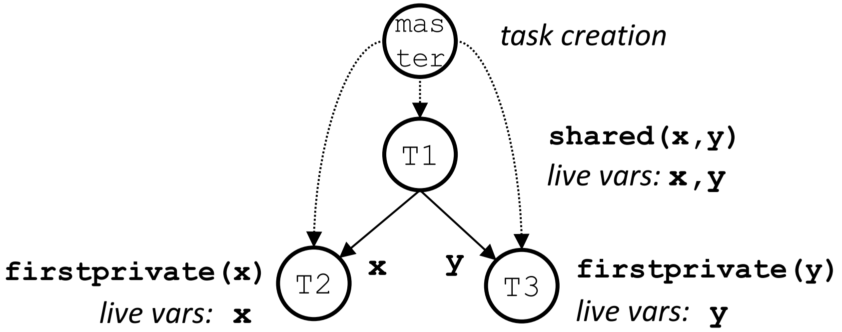


Task Dependency Graph (TDG)

A representation of the parallel nature of a workflow

- Includes all the information for functional and non-funcional correctness
  - **Parallel units** and **synchronization** dependencies
  - **Liveness analysis of variables** and **data-sharings** involved in the parallel execution
- Independent from the targeted parallel platform (but can include HW dependent information)
  - **Execution characterisation** of parallel units (e.g., time, energy, memory behaviour)

```
#pragma omp parallel
#pragma omp master
{
    int x,y;
    #pragma omp task depend(out:x,y) shared(x,y) // T1
    { f1(&x,&y); }
    #pragma omp task depend(in:x) firstprivate(x) // T2
    { f2(x); }
    #pragma omp task depend(in:y) firstprivate(y) // T3
    { f3(y); }
}
```



Principle behind Tasking Models

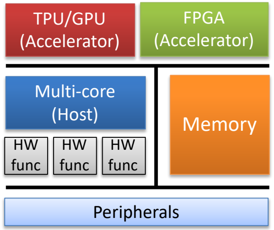
- Tasking provides a great expressiveness to describe the parallel nature of applications
  - It specifies **what** the application does and not **how** it is done
  - The framework is responsible of orchestrating the execution

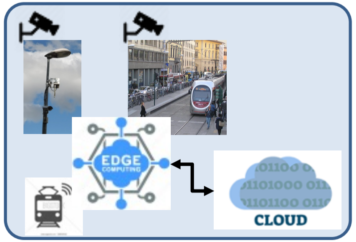
interoperability

OpenMP ⟷ COMPSs

NVIDIA CUDA, POSIX pthreads, RTEMS, AdaCore

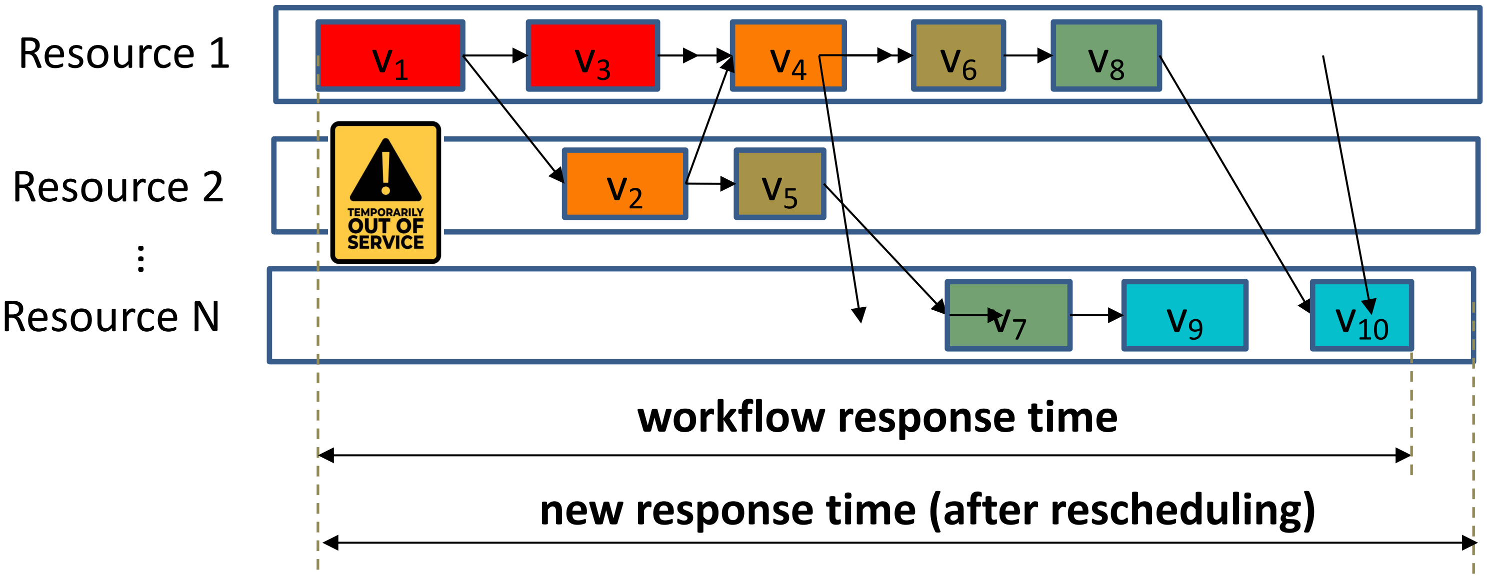
MINIO, S3, docker, kubernetes, LITHOPS, Skupper





Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Orchestration of resources



Conclusions

1. The compute continuum provides the computing capabilities to cope with the performance requirements of complex data-analytics workflows, and…
2. … task-based parallel programming models allows to **reasoning about functional and time predictability** while removing from developers the responsibilty of managing the complexity of the compute continuum

VERY INTERESTING RESEARCH IS STILL PENDING!

ascender

www.proyecto-ascender.com

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXTRACT

www.extract-project.eu

The Compute Continuum: An Efficient Use of Edge-to-Cloud Computing Resources

Eduardo Quiñones

{eduardo.quinones@bsc.es}

Ada-Europe, Barcelona, 2024