POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA,
INFORMAZIONE E BIOINGEGNERIA

HEAP LAB

Tutorial: A Run-time Managed Programming Approach to Computing Continuum

Giuseppe Massari, Domenico Iezzi,
Michele Zanella, William Fornaciari

<name.surname@polimi.it>

MANGO

HiPEAC2020 – Bologna (Italy) – 20-22 January 2020

RECOPE

Agenda

10.00 – 10.30: Introduction to Computing Continuum

10.30 – 11.00: Frameworks overview

11.00 – 11.30: Coffee break

11.30 – 12.30: Programming with MANGO

12.30 – 13.00: Demos

Introduction to Computing Continuum

Prof. William Fornaciari

Associate Professor at DEIB, Politecnico di Milano

The Problem

Devices and sensors pervasiveness

GENERATE

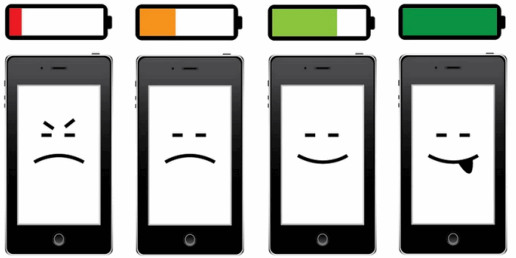
Huge amount of raw data

(about 850 Zettabytes by 2021)

High rate

They need to be processed

Increased capabilities and computational power of mobile and embedded devices



BUT

Battery-powered devices -> *Energy/Thermal constraints*

Evolution of Computing Paradigms

Paradigm	Pros	Cons
Cloud Computing (Centralized)	• User's costs • **Performance** • Ideal for Big Data analysis	• Long-term scalability • Server farm power consumption • **Internet connection (latency, availability)** • **High Internet traffic** • **No Real-time**
Edge Computing (Local single-node)	• Reduces Internet traffic • **Local computation** and data aggregation/filtering	• **Embedded limited devices** • **Internet connection still required** • **Cloud-based**
Mobile Cloud Computing/Fog (Decentralized Local multi-node)	• Mixes and extends Cloud and Edge • **No internet connection Real-time** • **Privacy-aware**	• **Architectural models** • **Availability of nodes** • **Computation management** • **Open issues**

Fog Computing: towards the continuum

Proposed in 2012 by Cisco:

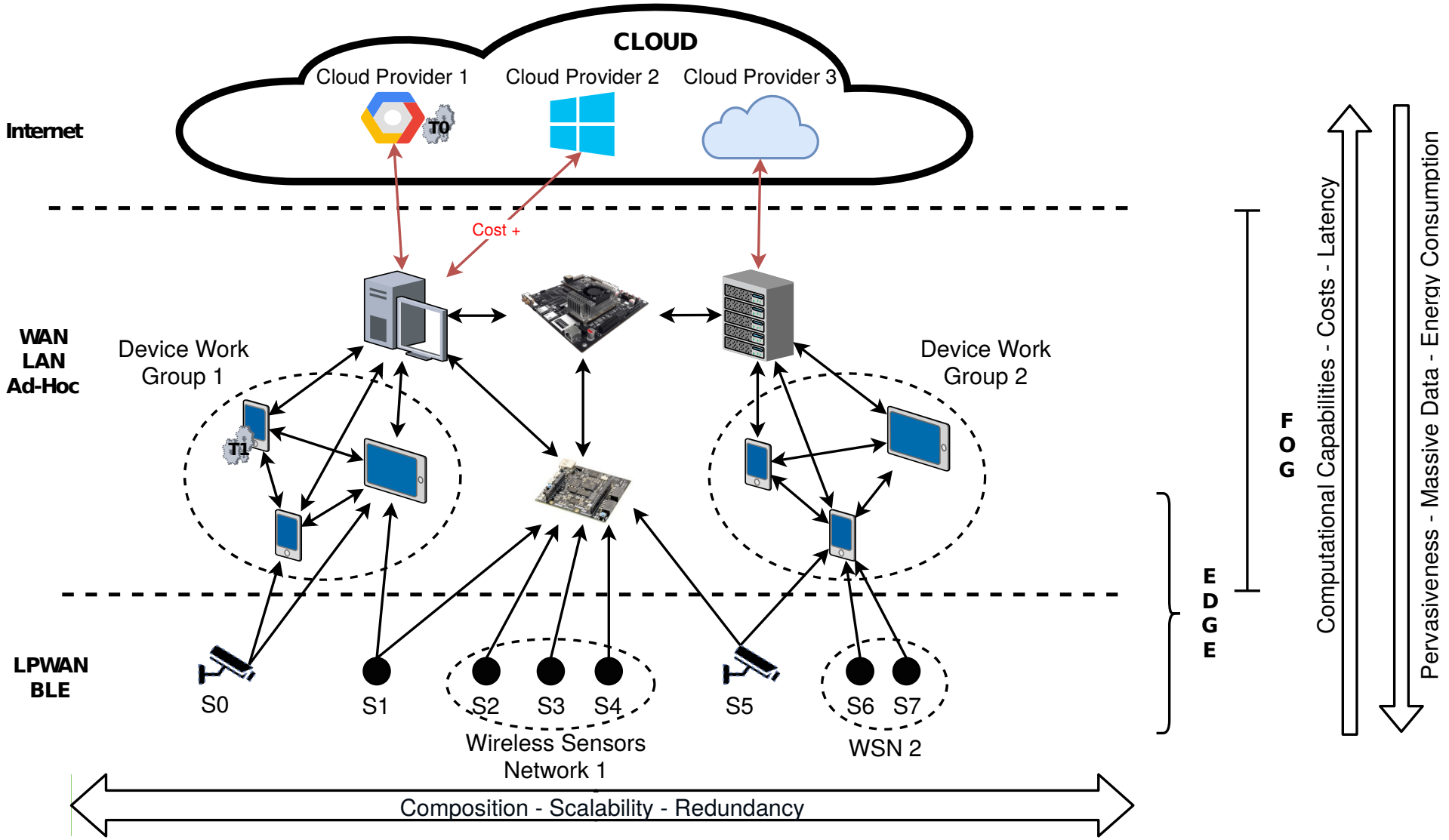
- Initially as a computational layer below the Cloud
- Heterogeneous (wireless and sometimes autonomous) ubiquitous and decentralized devices that can communicate and cooperate
- Storage and processing without third parties

Then extended by the OpenFog Consortium in 2017:

*a system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the **continuum** from Cloud to Things.*

Hot research topic:
More than 15 surveys from 2014- 2018
+ 500% articles from 2017 to now w.r.t 2014-2017
50 billion of connected entities by 2021
768 Million USD market by 2025

A Reference Architecture [1]



[1] M. Zanella, G. Massari, W. Fornaciari, Back to the future: Resource Management in Post-Cloud Solutions. *INTESA Workshop 2018*

Goals

- Efficiently heterogeneous Cloud-to-Edge **resources integration and management** in a scalable *infrastructure prototype*
- Transparent and secure **allocation of application's tasks** among the devices
- Taking into account **application's requirements** and single **device's constraints**.
- Showing **benefits** in term of energy consumption, resource utilization and performance of both devices and applications
- Providing an evaluation on a **platform using real-world use cases**

Challenges

1) Poor or lack of **fine-grained distributed resource management** in Fog and Cloud-to-Edge solutions
- Devices' mobility and availability
- Resource heterogeneity
- Energy consumption
- Workload allocation and balancing
- Single node optimization

2) Application programmability and task distribution capability
- Unified programming model
- Transparent tasks distribution
- Security and Privacy aspects

3) Need of a **test-bed for real-world use case** scenarios
- Enrich simulations with realistic power/energy model

Addressing the challenges

Challenge	Proposed solution
Fine-grained Distributed Resource Management Device Mobility and availability Resource Heterogeneity	*The use of the Distributed capabilities of the **Barbeque Run-Time Resource Manager** that supports heterogeneous and mobile platforms.*
Energy consumption Workload allocation and balancing	*The resource management can be enforced with customized **allocation policies** that optimize energy efficiency and/or performance.*
Single node optimization	*Local optimization is done by each instances of the **resource manager** running on the devices.*
Unified programming model	*The **libmango**: a resource-aware task-based programming model. It hides the underlying resource heterogeneity.*
Transparent task distribution Security and privacy awareness	*The **BeeR framework** for task distribution. It has been currently extending with cryptography features.*
Test-bed with real-world use-case scenarios	*We built a real Fog test-bed with power probing capabilities: the **SmokyGrill**.*

Addressing the challenges

	Fine-grained Distributed Resource Management					Application programmability			Real testbed
	Device mobility	Resource Heterogeneity	Energy awareness	Allocation and balancing	Node optimization	Unified model	Transparent task distribution	Security and privacy	
Barbeque RTRM	✓	✓	✓	✓	✓				
libmango		✓				✓	✓		
BeeR							✓	✓	
SmokyGrill		✓							✓

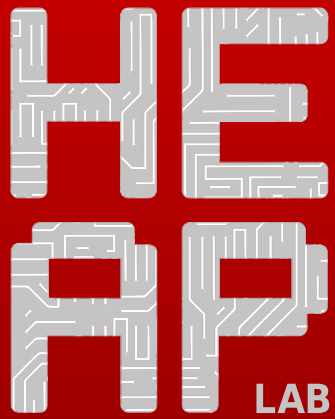
What we are going to present

A full-stack framework to manage and enable a Resource Continuum scenario, composed by:

- The Barbeque Run-Time Resource Manager
- The libmango programming model
- The BeeR task distribution framework
- Programming tutorial with sample applications
- Demos

POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA,
INFORMAZIONE E BIOINGEGNERIA



The MANGOLIBS Project

Giuseppe Massari

<giuseppe.massari@polimi.it>

HiPEAC2020 – Bologna (Italy) – 20-22 January 2020

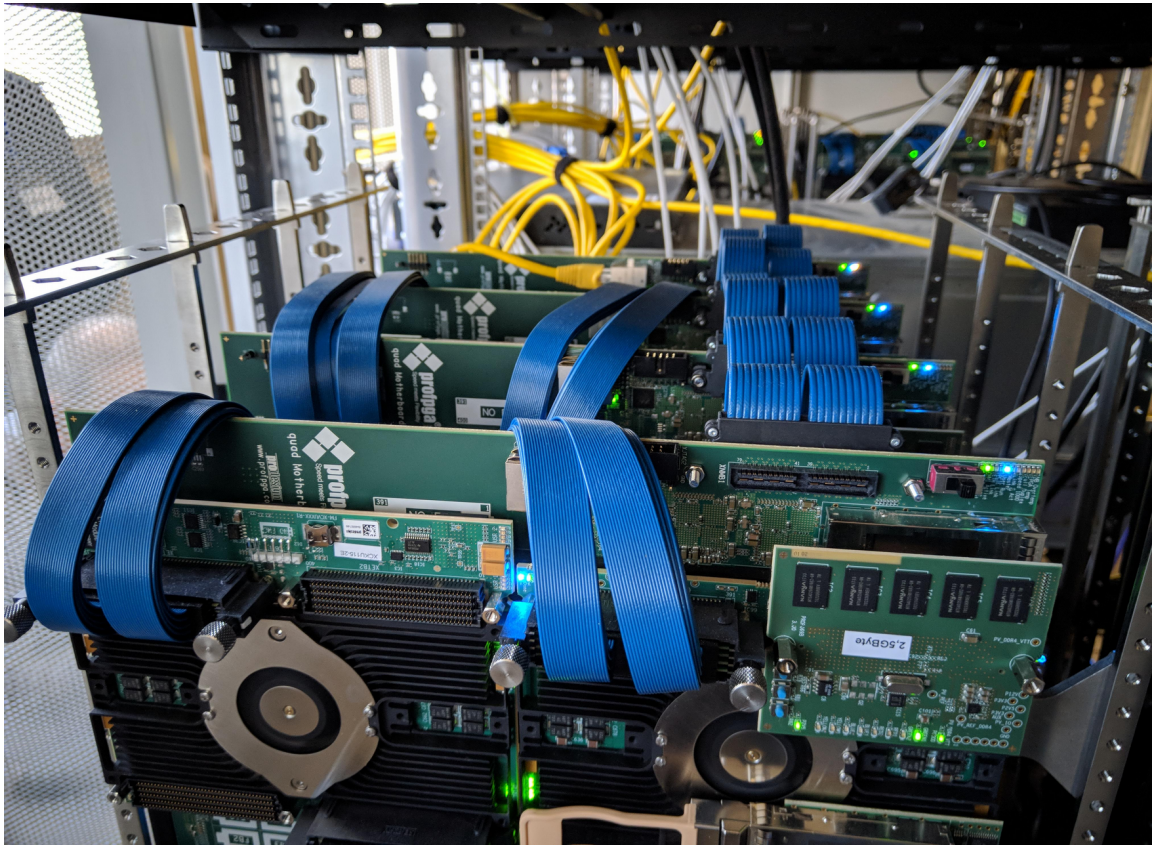
Outline

- The road to MANGOLIBS
- Architecture
- Download and installation
- Usage

The road to MANGOLIBS

The H2020 MANGO project inheritance

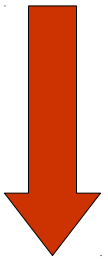
- Experimental heterogeneous HPC target platform
- http://mango-project.eu/



The road to MANGOLIBS

The H2020 MANGO project inheritance

- What about the programming model?
- Developers: “We don’t want to use OpenCL”

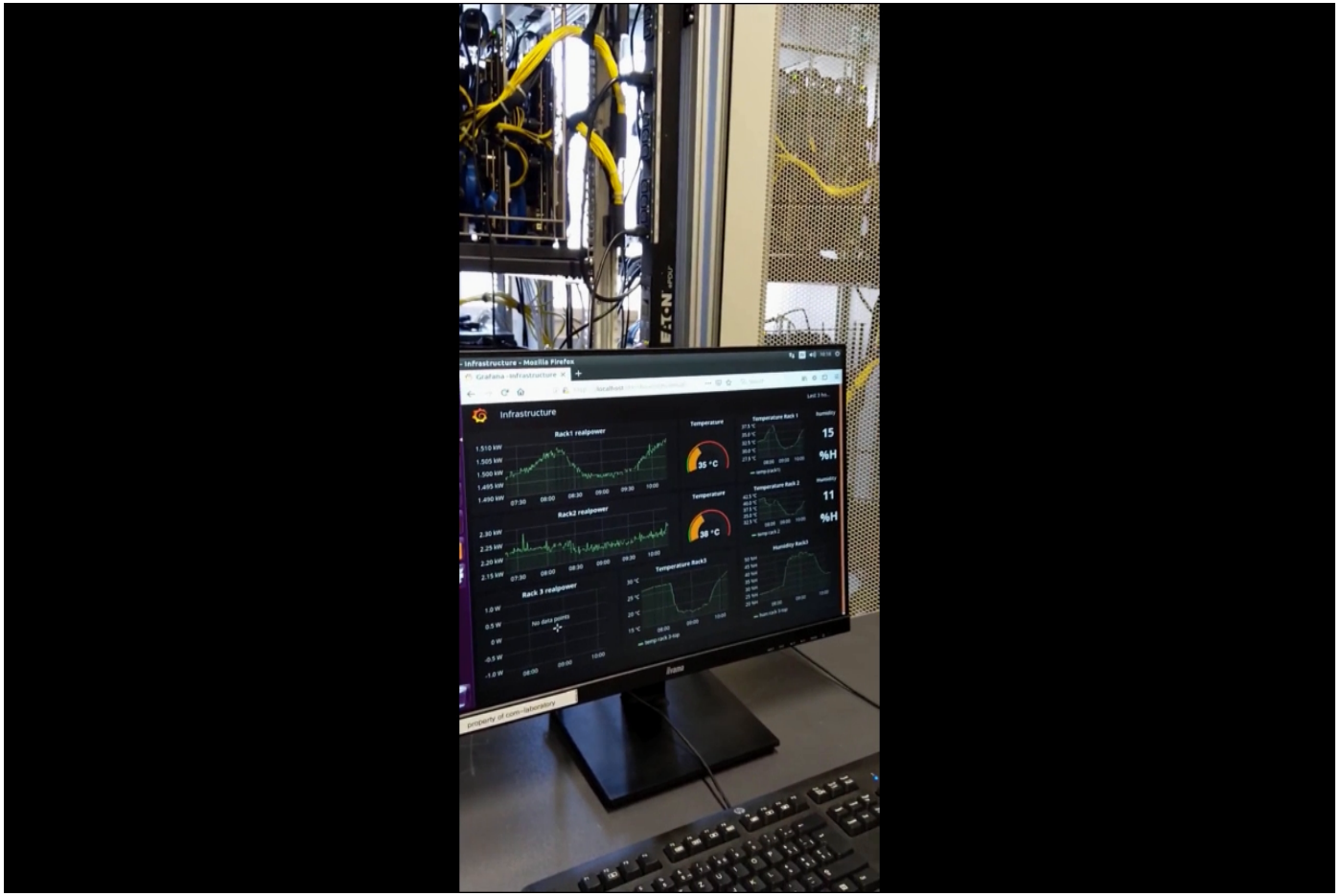


- **MANGO Programming Model**
  - ➔ Kernels offloading and buffers allocation (resource mapping) managed by the BarbequeRTRM framework
  - ➔ Resource manager ↔ Programming library integration

The road to MANGOLIBS

The H2020 MANGO project inheritance

- The prototype



The road to MANGOLIBS

The H2020 MANGO project inheritance

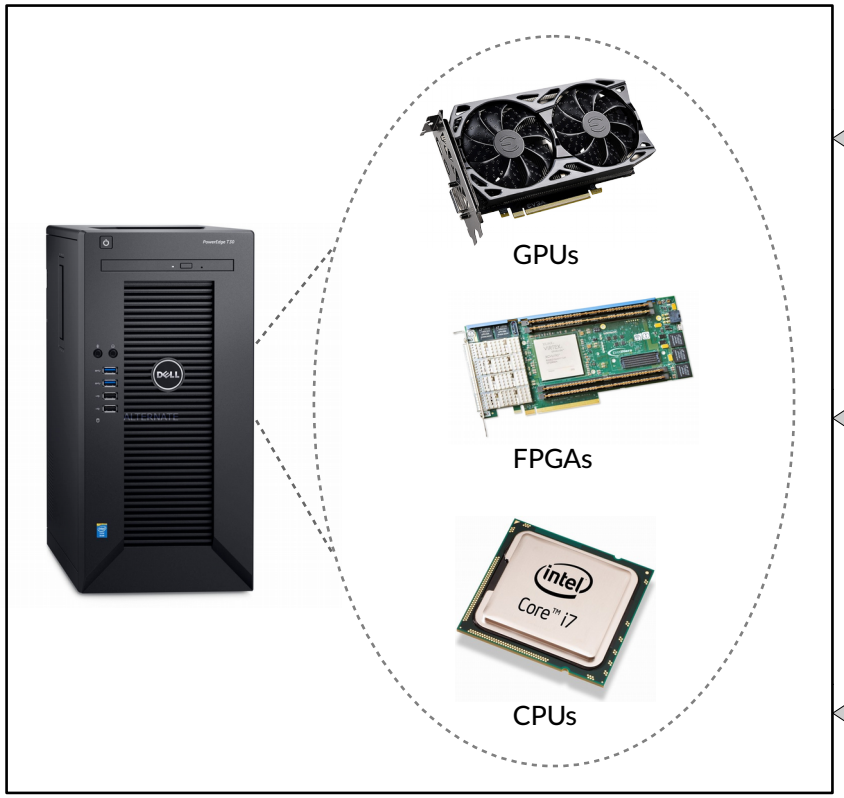
- The driving idea…

Why don't we use the same programming model for both heterogeneous HPC and Fog/Edge distributed systems?

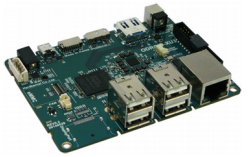
The road to MANGOLIBS

Fog/Edge scenario

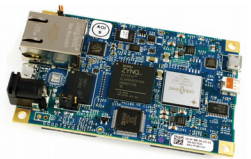
- Distributed and heterogeneous set of nodes



Heterogeneous
workstation or server



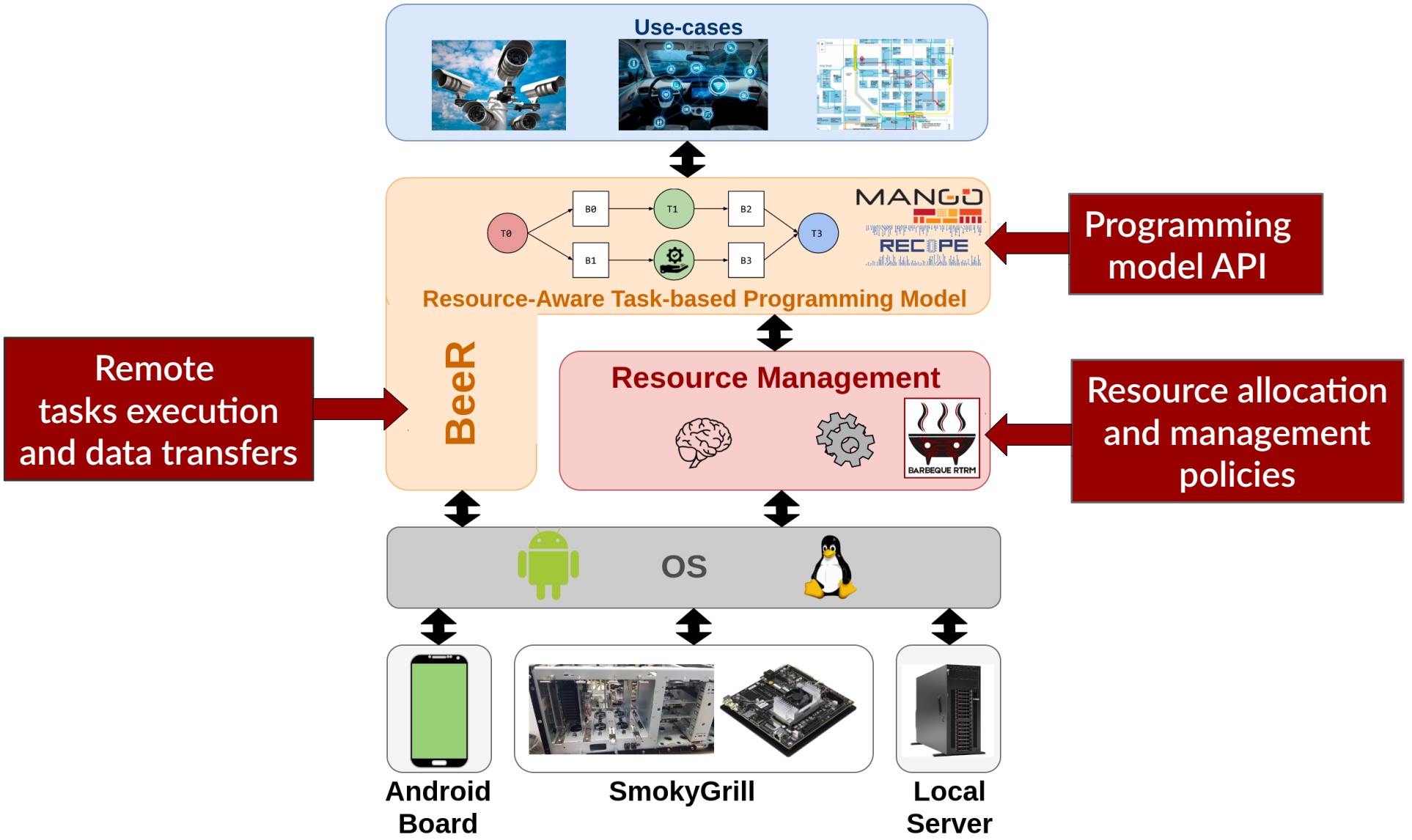
8-core ARM 32 bit
+ embedded GPU



2-core ARM (ZynQ)
+ HW accelerator

4-core ARM 64 bit
+ embedded GPU

Architecture



MANGO Programming Library

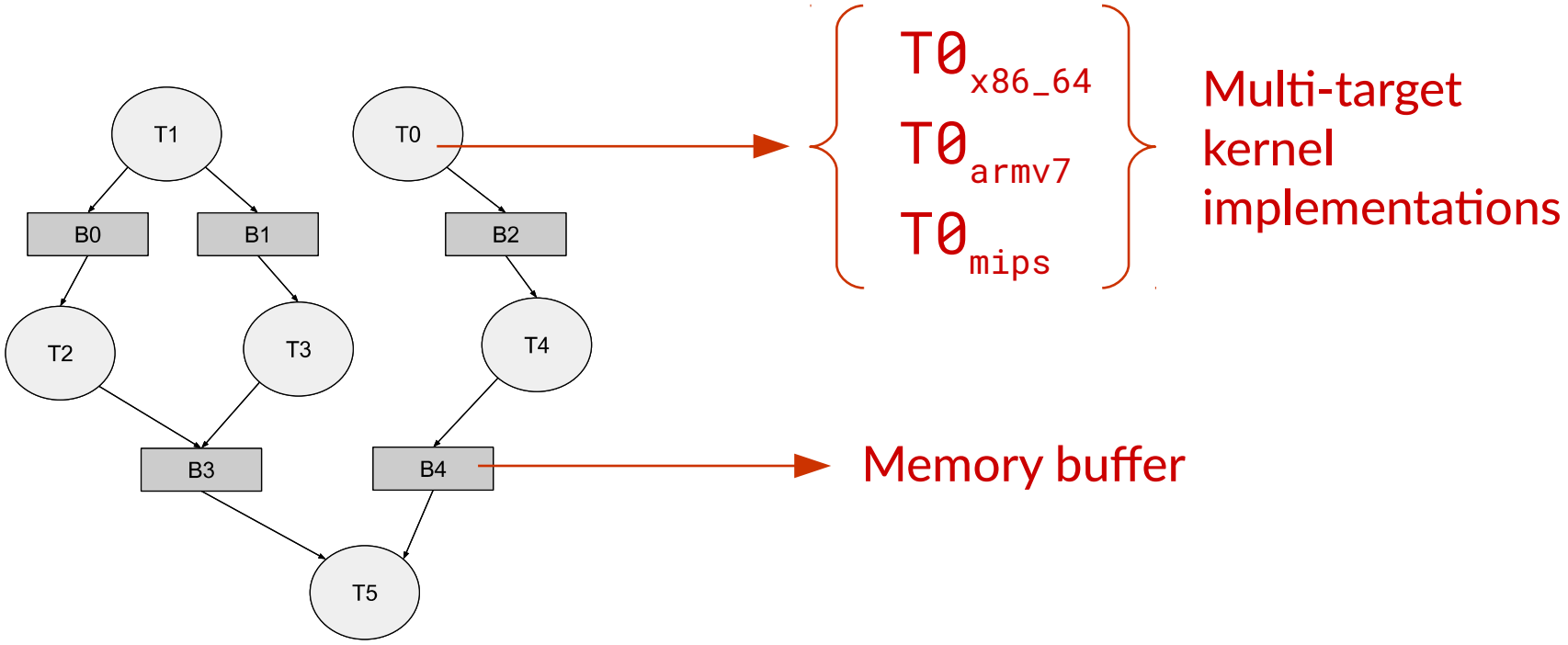
MANGO Programming Model

- C and C++ API
  - Python wrapper also available
- Multi-tasking programming
- Multi-target tasks implementation (different kernel compilations)
- Similar to OpenCL but with some differences
  - Developer has to provide a task-graph based description of the application
  - Platform/device selection is left to the resource manager
  - Possibility of specifying per-task requirements
  - Just-In-Time compilation NOT available yet
- Profiling support included

MANGO Programming Library

Task-graph based representation

- Tasks (kernels) and memory buffers

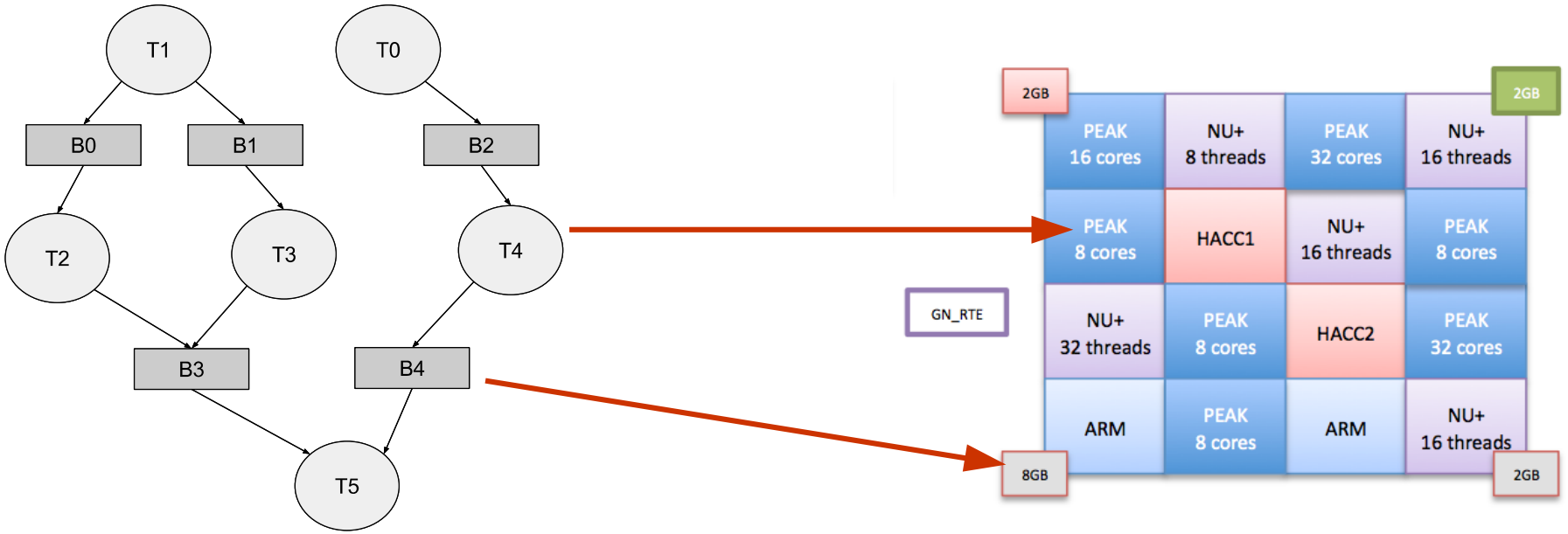


Application

MANGO Programming Library

Task-graph based representation

- Run-time resource mapping



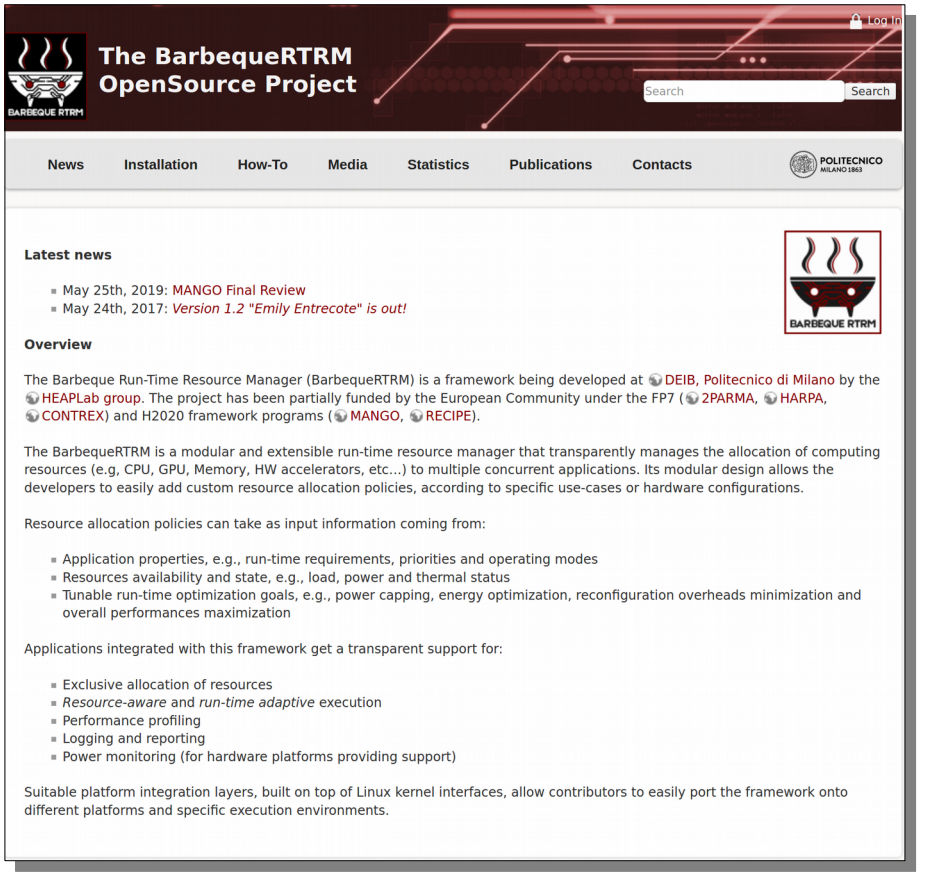
Application

Target platform

Barbeque Run-Time Resource Manager

BarbequeRTRM

- Project was born in 2012 and has been involved in several FP7 (2PARMA, HARPA, CONTREX) and H2020 (MANGO, RECIPE) projects
- https://bosp.deib.polimi.it/
  - ➔ Subscribe to the mailing lists for the update



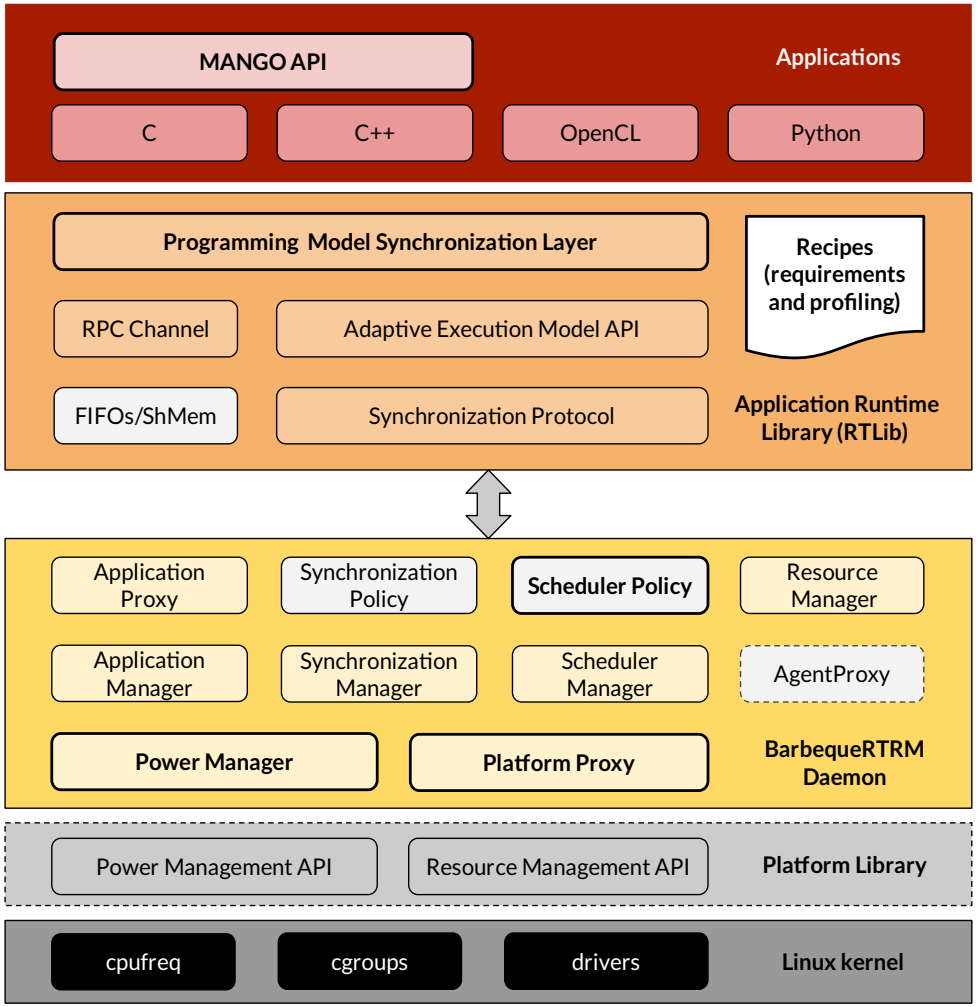
Barbeque Run-Time Resource Manager

BarbequeRTRM

- Run-time resource manager for multi/many-core platforms
  - Processing units, memory and network bandwidth reservation
  - Power management
- HPC and multi-core embedded target platforms
  - Homogeneous and heterogeneous ISA
  - Single node and distributed systems
- Modular design
  - Resource allocation policy plug-in
  - Minimum porting effort to support new target hardware platforms
- Programming models
  - OpenMP, OpenCL, ...
  - *Adaptive Execution Model*
  - *MANGO*

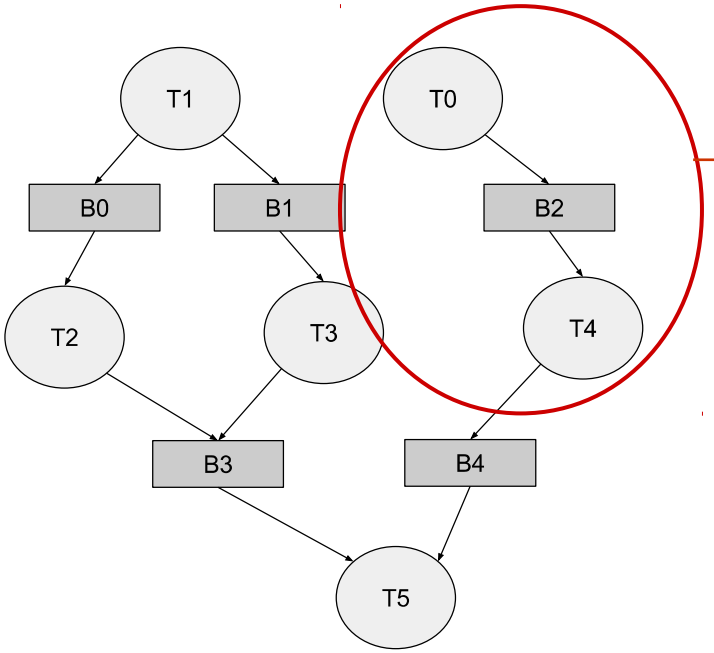
Barbeque Run-Time Resource Manager

Modular design



Goals

- Enable the “task offloading” towards remote nodes
- Transparent management of remote task/kernel execution
- Transparent management of data transfers

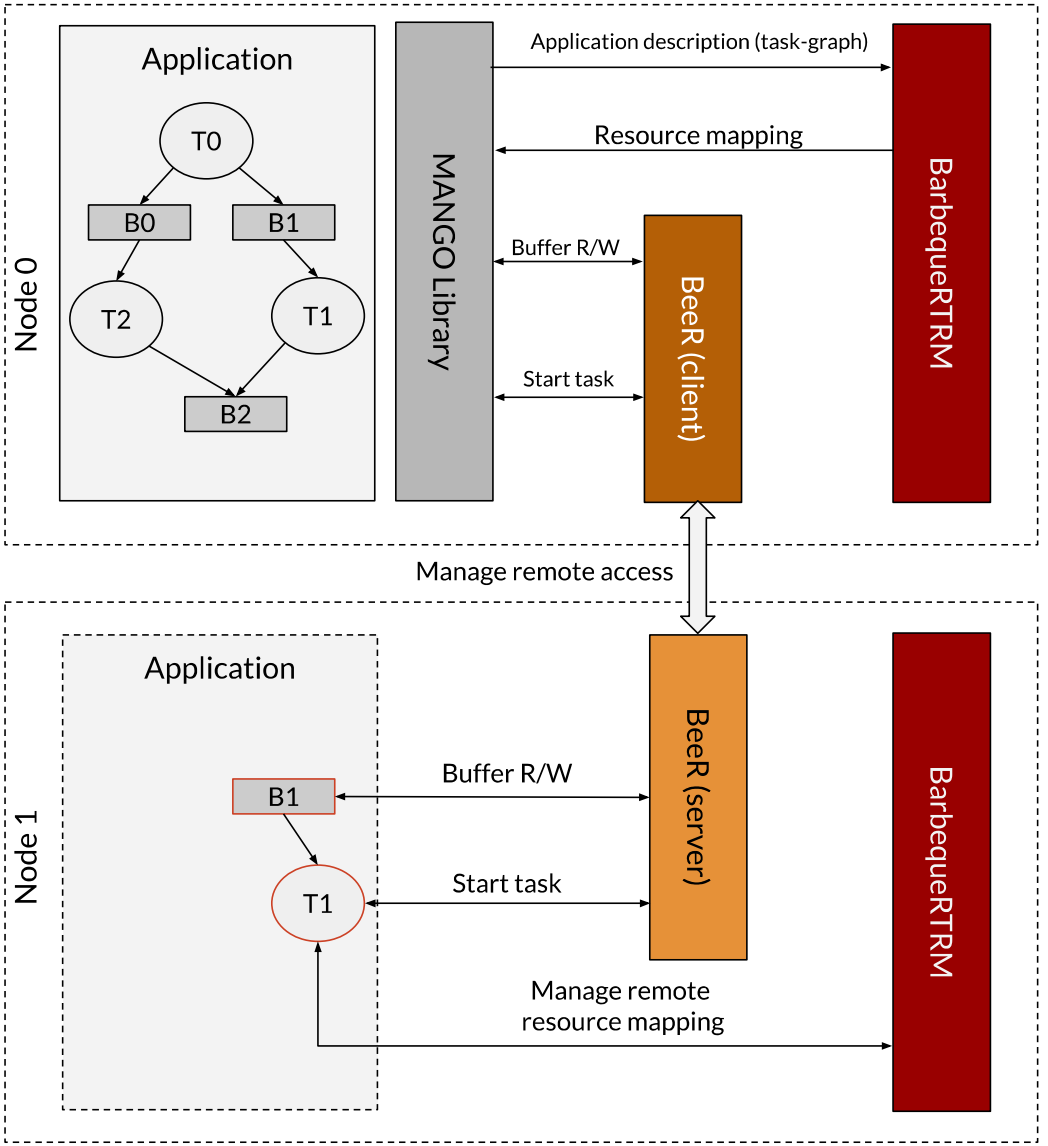


Example

→ T1 and T4 will run on different nodes

→ Buffer B2 must be allocated considering the distributed context

Integration

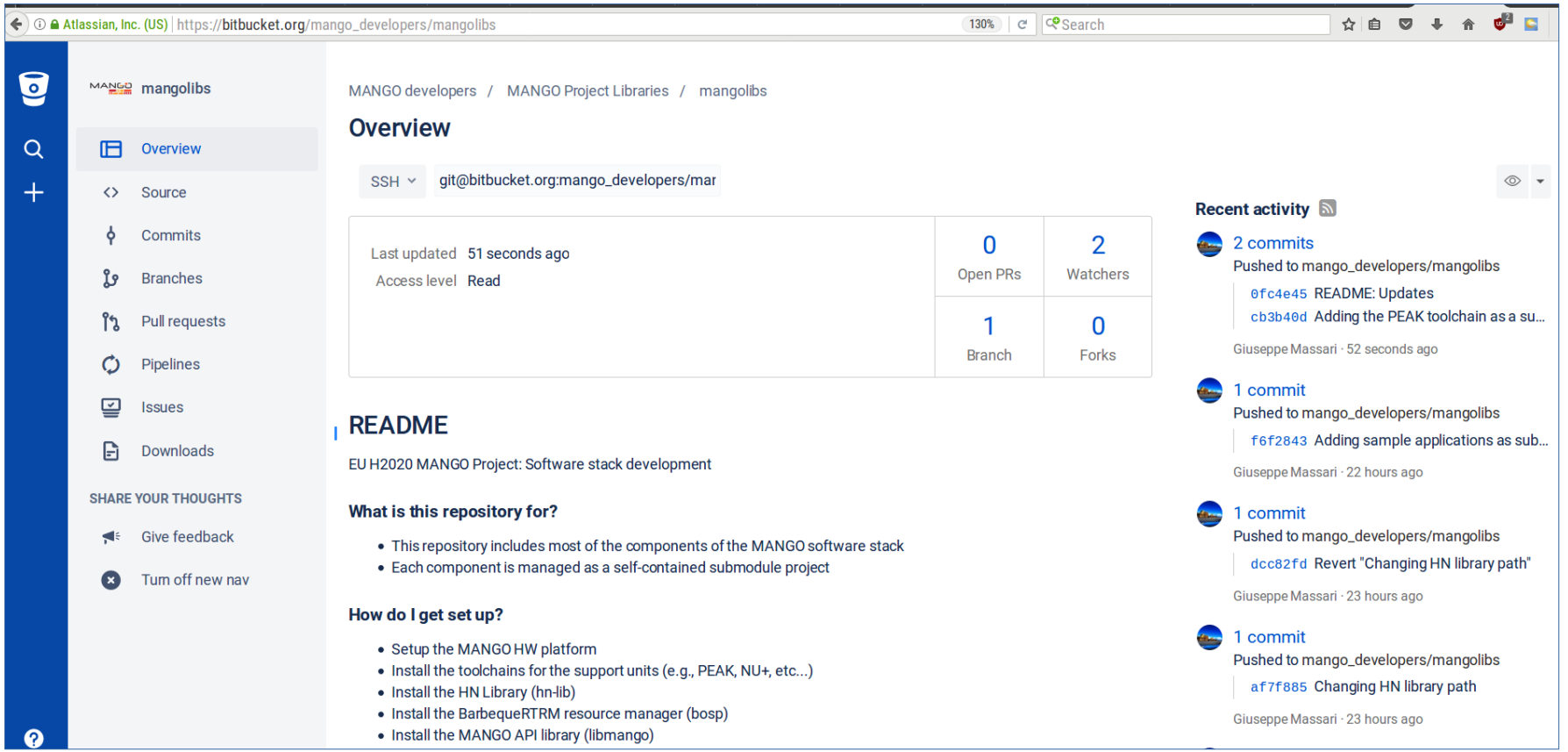


Download and installation

Download and installation

Download

- GIT repository: https://bitbucket.org/mango_developers/mangolibs



Download and installation

Download

- GIT repository: https://bitbucket.org/mango_developers/mangolibs
  - **beer**
  - **bosp**
  - libhn
  - lbgn
  - **libmango**
  - toolchains
  - samples
  - sdk-*
  - ...

Download and installation

Download

- Clone the GIT repository

```
[...~$] git clone https://bitbucket.org/mango_developers/mangolibs.git
```

- Initialize the submodules

```
[... ~$] cd mangolibs
[mangolibs $] ./setup.sh
```

- Getting updates

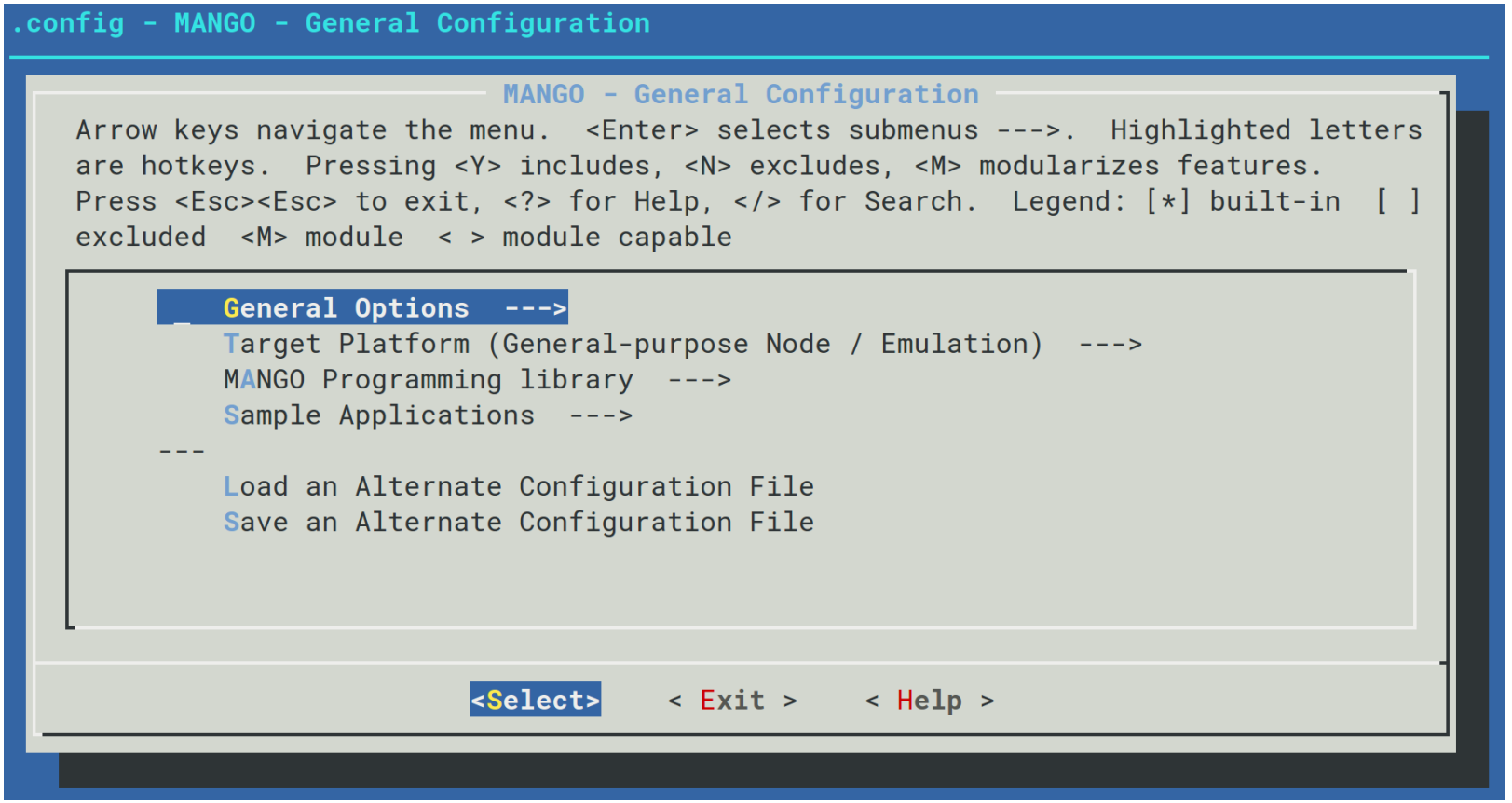
```
[mangolibs $] ./update.sh [branch-name]
```

Download and installation

Compilation

- Open the Kconfig menu for setting the build options

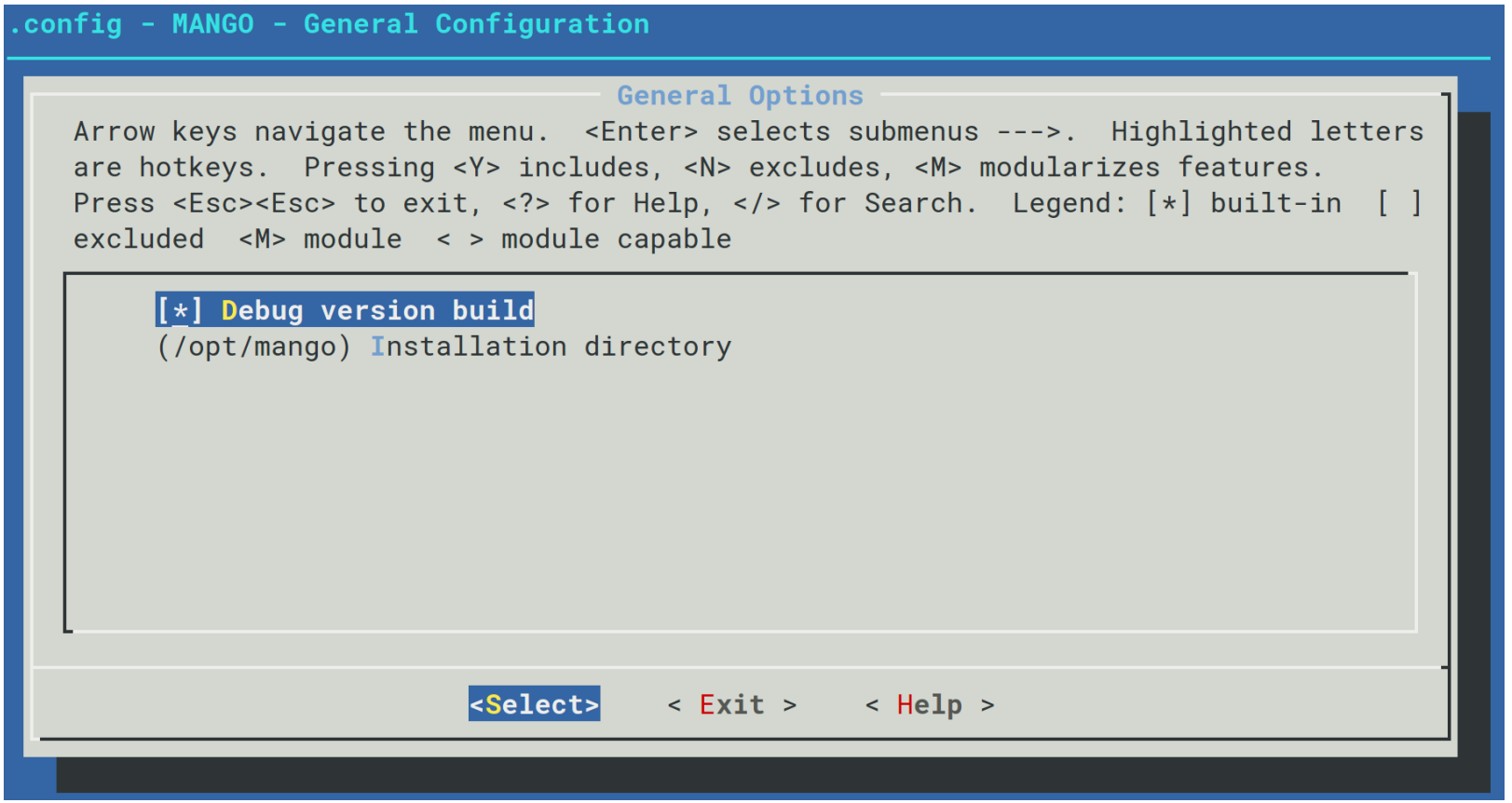
```
[mangolibs $] make menuconfig
```



Download and installation

Compilation

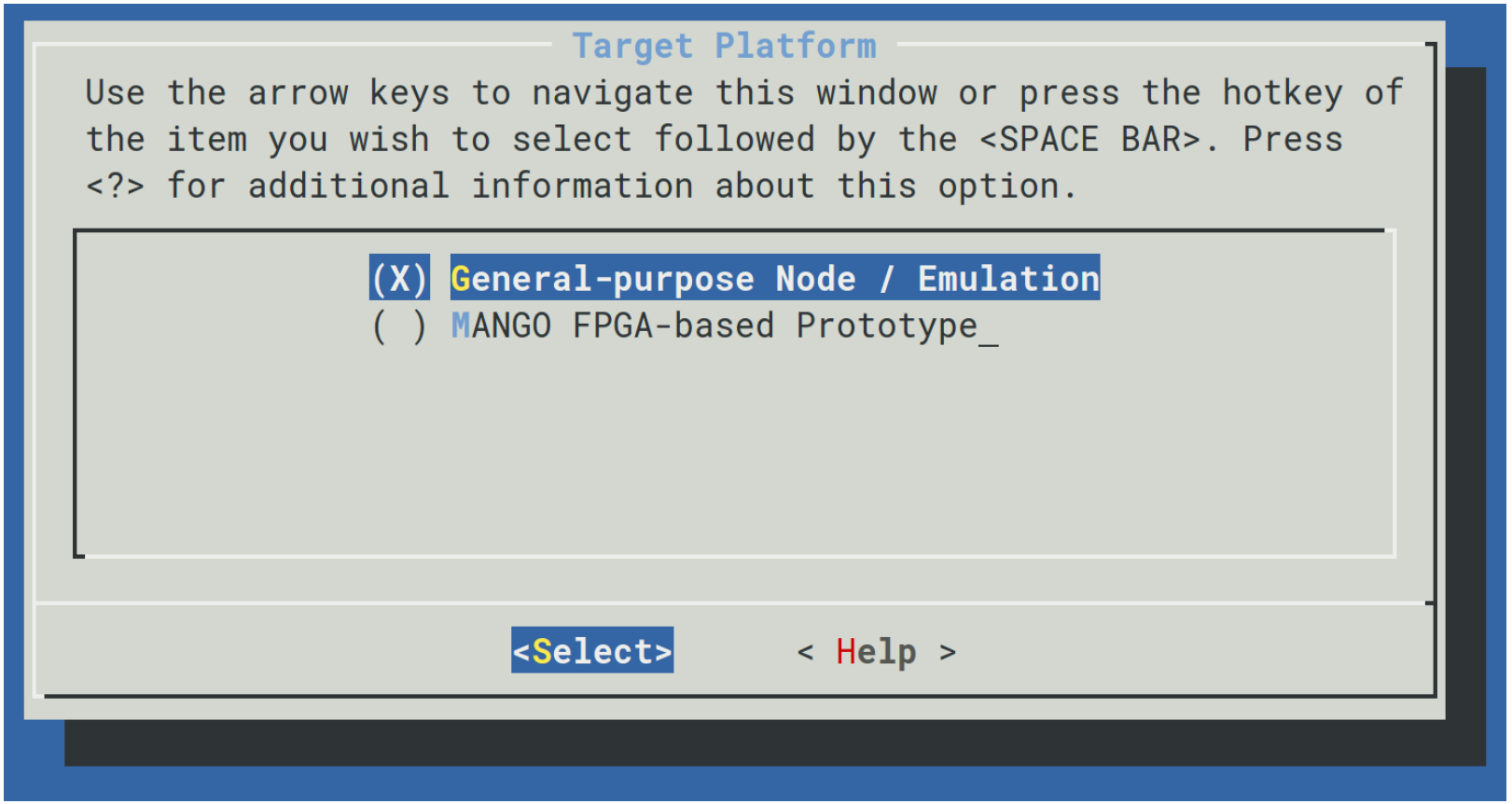
- General Options
  - ➔ Build mode (Debug vs Release) and installation directory setting



Download and installation

Compilation

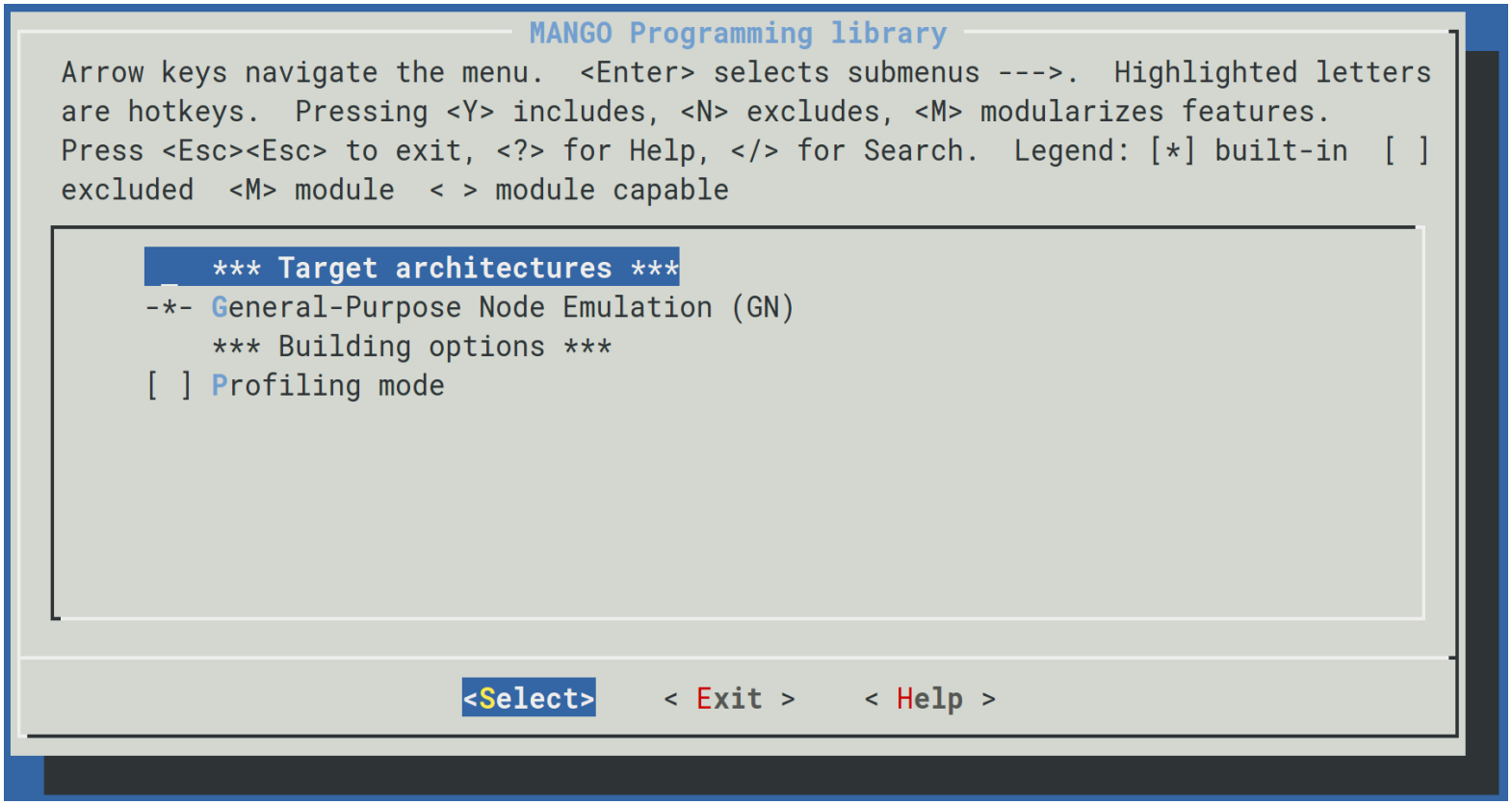
- Target Platform
  - ➔ Select “General-purpose” if the target is not a MANGO prototype



Download and installation

Compilation

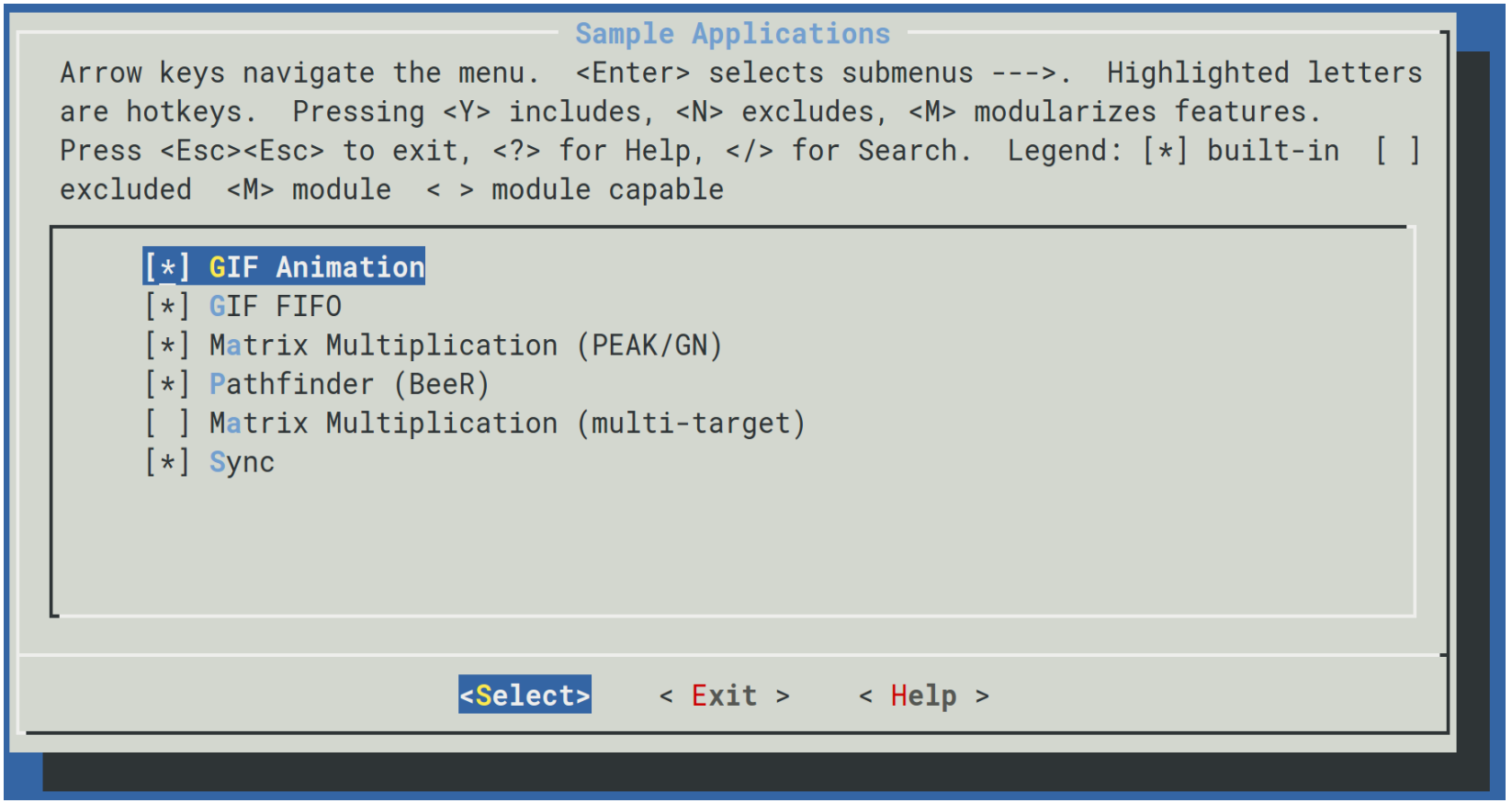
- MANGO Programming library
  - “General-purpose Node” is the only target option (already set)
  - We could enable “Profiling mode” for timing profiling



Download and installation

Compilation

- Sample Applications
  - ➔ Enable the compilation of some sample applications...



Download and installation

Compilation

- Save the Kconfig configuration and exit
- Launch the compilation (add ‘sudo’ if the installation path requires it)

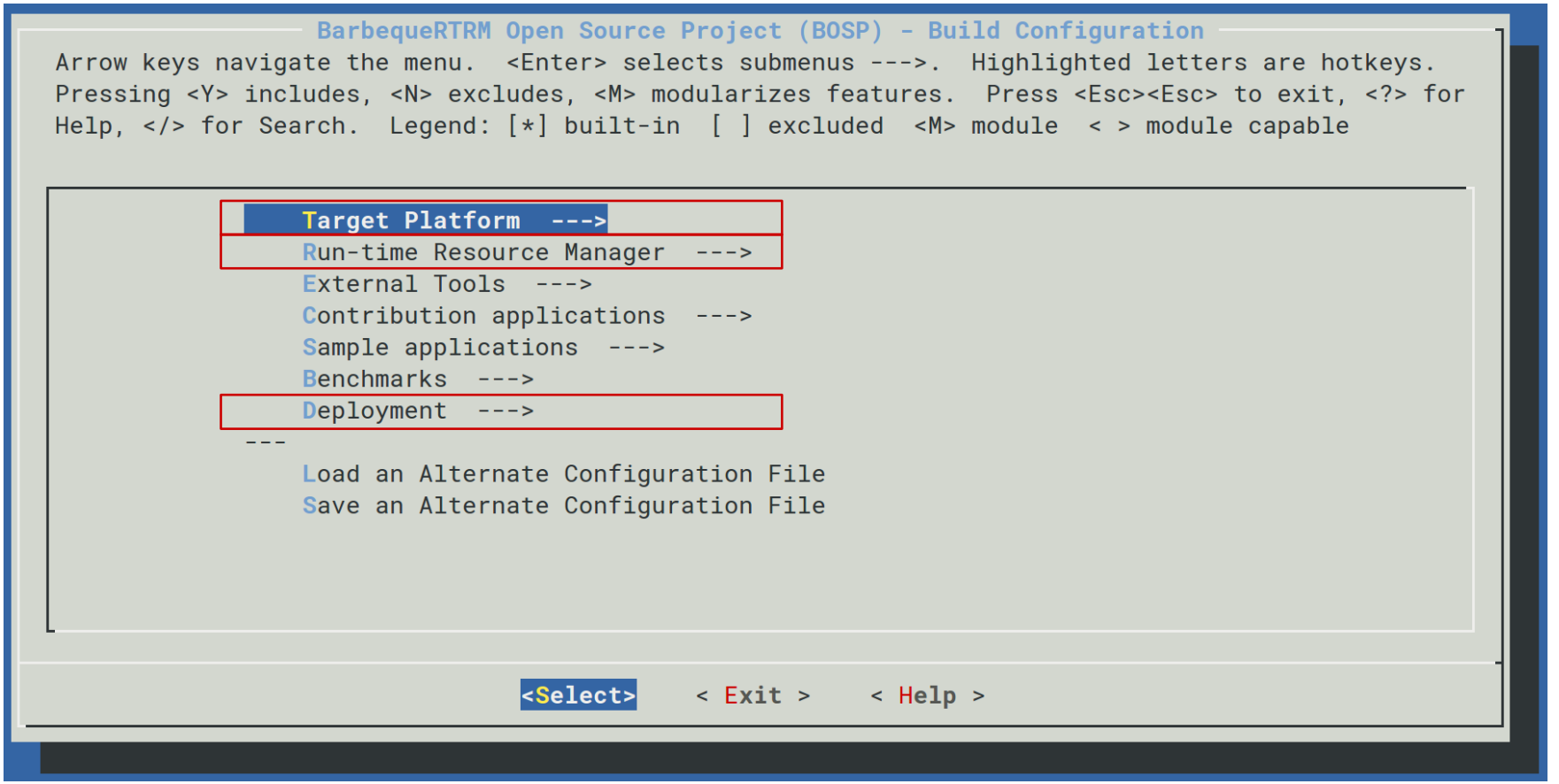
```
[mangolibs $] MANGO_ROOT=/opt/mango make
```

```
---------------------------------------------
        Generating configuration
---------------------------------------------
make[1]: Entering directory '/home/giuseppe/Development/mangolibs/bosp/external/tools/kconfig'
make[2]: Entering directory '/home/giuseppe/Development/mangolibs/bosp/external/tools/kconfig'
make[2]: Leaving directory '/home/giuseppe/Development/mangolibs/bosp/external/tools/kconfig'
make[1]: Leaving directory '/home/giuseppe/Development/mangolibs/bosp/external/tools/kconfig'
/home/giuseppe/Development/mangolibs/bosp/external/tools/kconfig//scripts/kconfig/mconf Kconfig
#
# configuration written to .config
#

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

cat .config | sed -f /home/giuseppe/Development/mangolibs/bosp/external/tools/kconfig//makefile_to_cmake.sed > global.cmake
giuseppe@jupiter:~/Development/mangolibs$ sudo make
```

Download and installation

Compilation

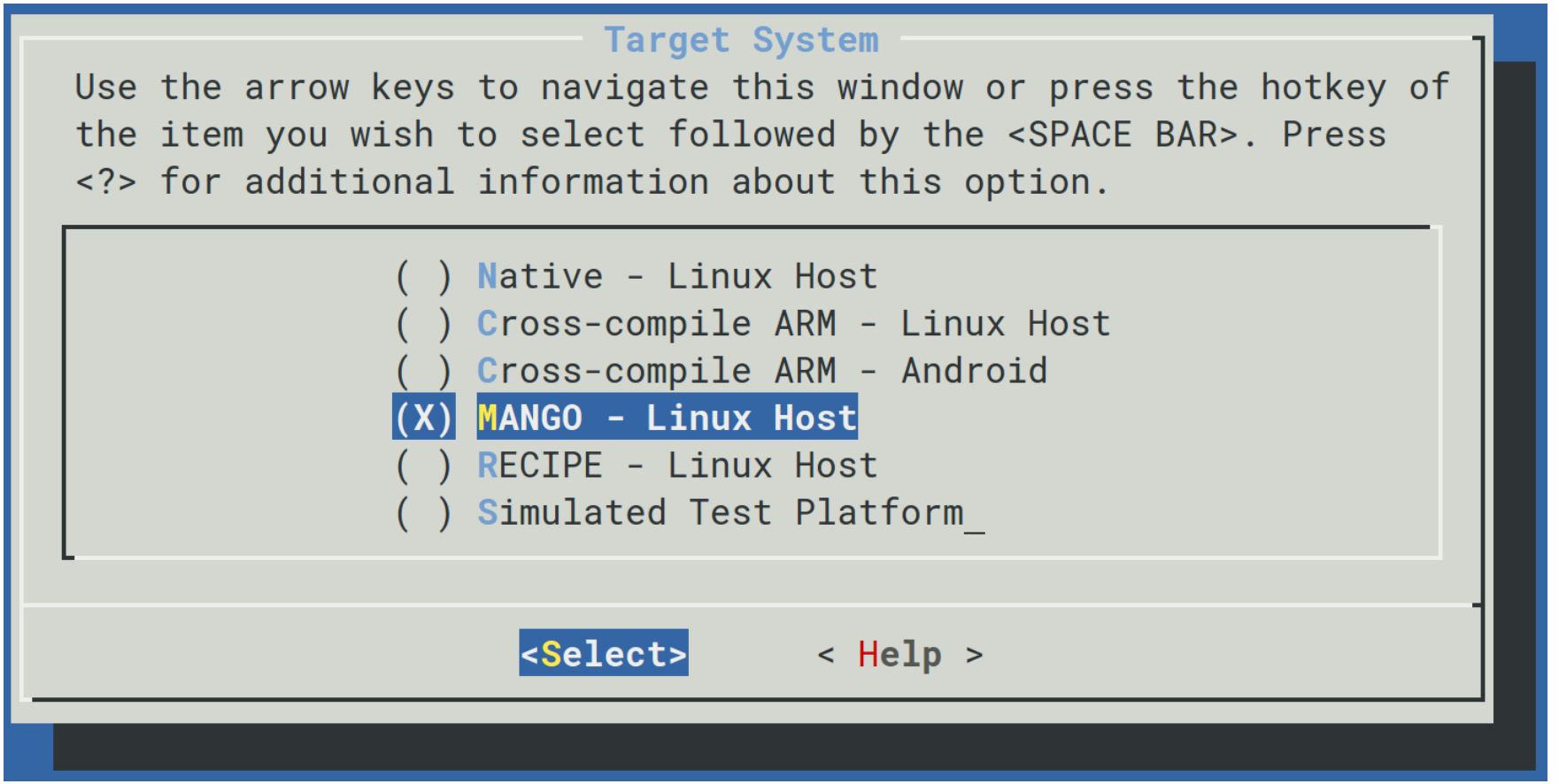
- A new configuration menu appears to check the BarbequeRTRM configuration



Download and installation

Compilation

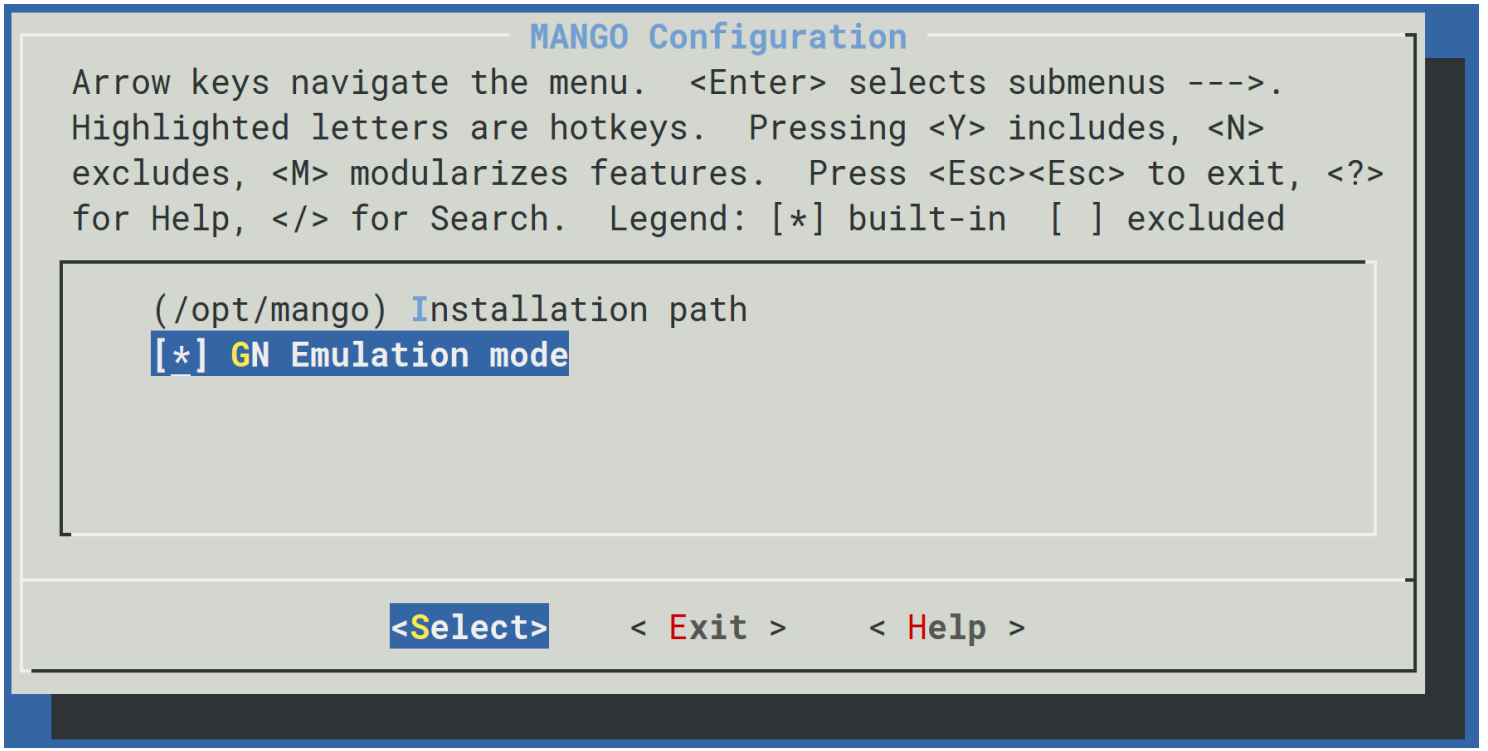
- Target Platform → Target Sytem
  - ➔ Select “MANGO – Linux Host”



Download and installation

Compilation

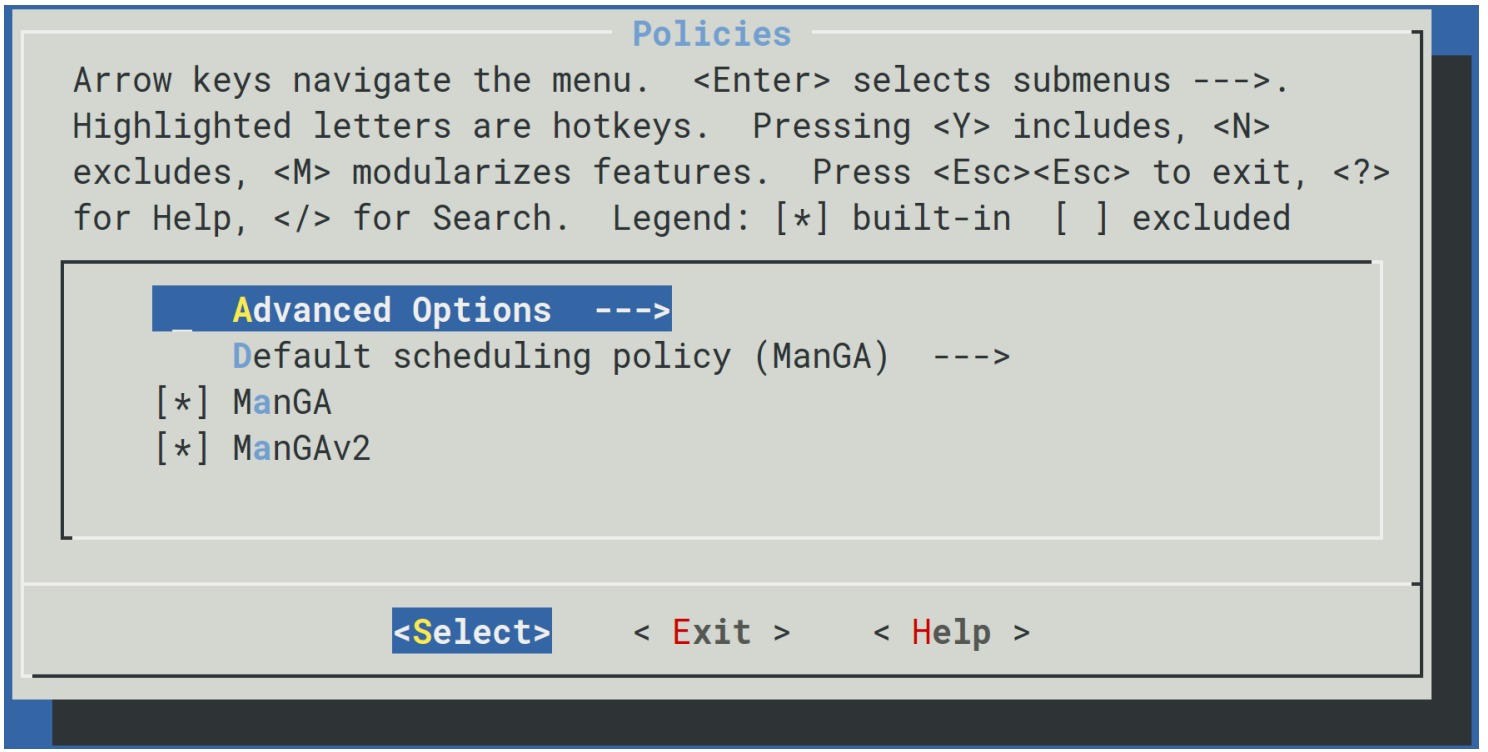
- Target Platform → MANGO Configuration
  - Check the installation path
  - “GN Emulation mode” can be selected if you want to run the resource manager in a test mode (not actual CPU/Memory assignment performed)



Download and installation

Compilation

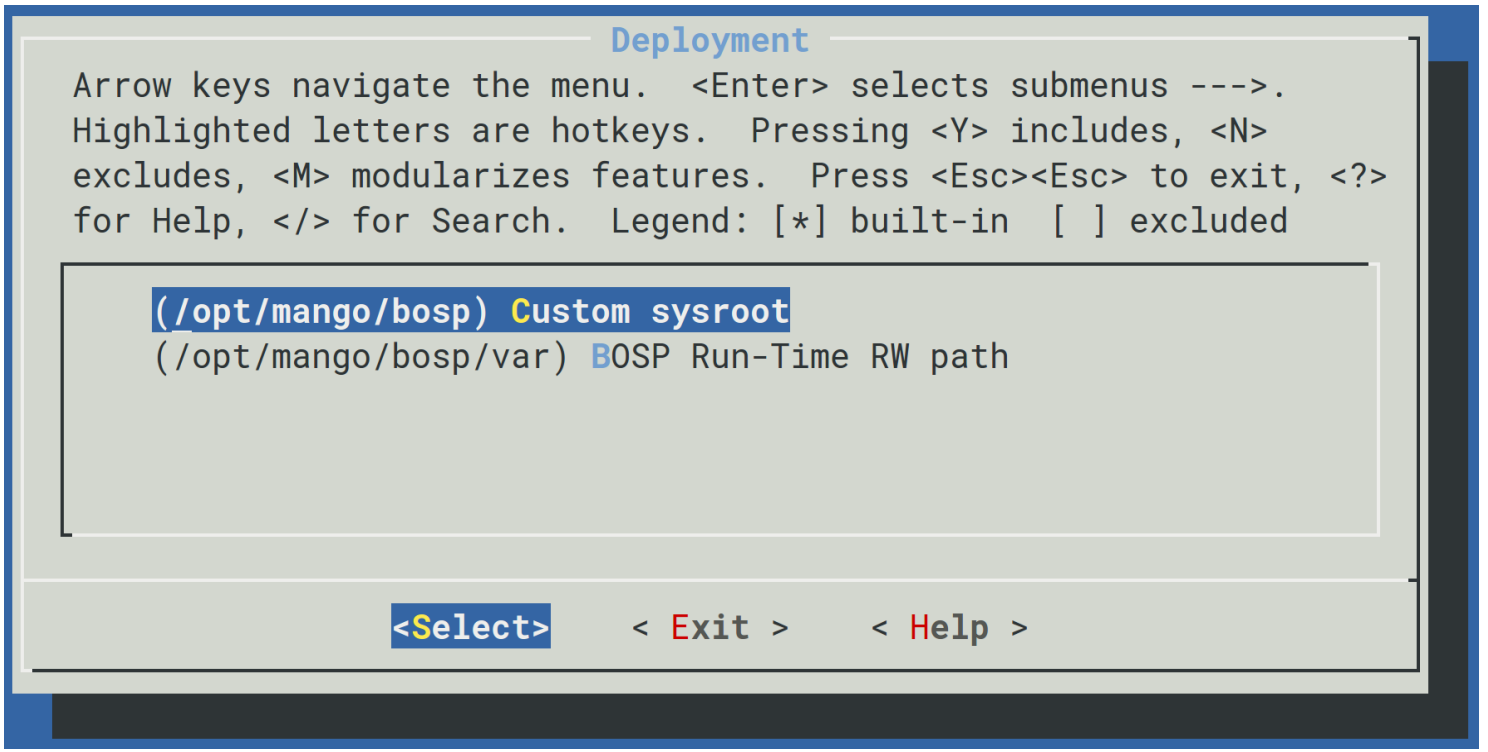
- Run-time Resource Manager → Policies
  - ManGA: assign the first available processor/ memory node / system node
  - ManGAv2: try to apply the resource mapping choices already provided in the “recipe” file



Download and installation

Compilation

- Deployment
  - ➔ Set the path as it follows (assuming '/opt/mango/' as the installation directory)



Download and installation

Compilation

- Save the configuration and exit
  - ➔ The compilation process will resume

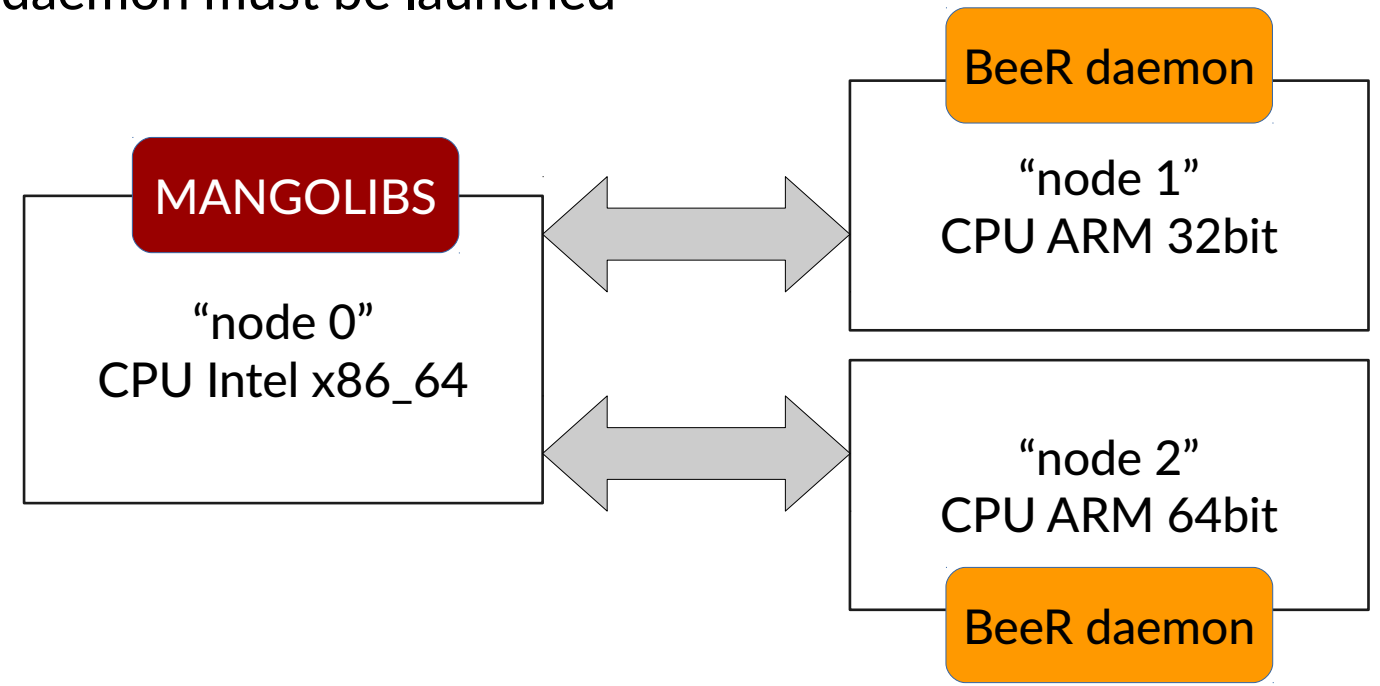
```
[ 20%] Built target bbque_rpc_fifo
make[4]: Entering directory '/home/giuseppe/Development/mangolibs/bosp/barbeque/build/Debug'
Scanning dependencies of target bbque_resources
make[4]: Leaving directory '/home/giuseppe/Development/mangolibs/bosp/barbeque/build/Debug'
make[4]: Entering directory '/home/giuseppe/Development/mangolibs/bosp/barbeque/build/Debug'
[ 21%] Building CXX object bbque/res/CMakeFiles/bbque_resources.dir/bitset.cc.o
[ 22%] Building CXX object bbque/res/CMakeFiles/bbque_resources.dir/binder.cc.o
[ 23%] Linking CXX static library libbbque_ploader_rxml.a
make[4]: Leaving directory '/home/giuseppe/Development/mangolibs/bosp/barbeque/build/Debug'
[ 23%] Built target bbque_ploader_rxml
make[4]: Entering directory '/home/giuseppe/Development/mangolibs/bosp/barbeque/build/Debug'
make[4]: Leaving directory '/home/giuseppe/Development/mangolibs/bosp/barbeque/build/Debug'
[ 26%] Built target bbque-plp
make[4]: Entering directory '/home/giuseppe/Development/mangolibs/bosp/barbeque/build/Debug'
Scanning dependencies of target bbque_utils
[ 28%] Building CXX object bbque/res/CMakeFiles/bbque_resources.dir/identifier.cc.o
make[4]: Leaving directory '/home/giuseppe/Development/mangolibs/bosp/barbeque/build/Debug'
make[4]: Entering directory '/home/giuseppe/Development/mangolibs/bosp/barbeque/build/Debug'
[ 29%] Building CXX object bbque/utils/CMakeFiles/bbque_utils.dir/timer.cc.o
[ 30%] Building CXX object bbque/res/CMakeFiles/bbque_resources.dir/resource_assignment.cc.o
[ 31%] Building CXX object plugins/rloader/rxml/CMakeFiles/bbque_rloader_rxml.dir/rxml_plugin.cc.o
[ 32%] Building CXX object bbque/utils/CMakeFiles/bbque_utils.dir/deferrable.cc.o
[ 33%] Building CXX object bbque/utils/CMakeFiles/bbque_utils.dir/worker.cc.o
[ 34%] Building CXX object bbque/utils/CMakeFiles/bbque_utils.dir/metrics_collector.cc.o
[ 35%] Building CXX object plugins/schedpol/mangav2/CMakeFiles/bbque_schedpol_mangav2.dir/mangav2_plugin.cc.o
[ 37%] Building CXX object bbque/res/CMakeFiles/bbque_resources.dir/resource_path.cc.o
[ 38%] Linking CXX shared module libbbque_syncpol_sasb.so
[ 39%] Building CXX object bbque/res/CMakeFiles/bbque_resources.dir/resource_tree.cc.o
make[4]: Leaving directory '/home/giuseppe/Development/mangolibs/bosp/barbeque/build/Debug'
[ 39%] Built target bbque_syncpol_sasb
[ 40%] Building CXX object bbque/res/CMakeFiles/bbque_resources.dir/resources.cc.o
[ 41%] Linking CXX shared module libbbque_schedpol_manga.so
make[4]: Leaving directory '/home/giuseppe/Development/mangolibs/bosp/barbeque/build/Debug'
[ 41%] Built target bbque_schedpol_manga
[ 42%] Building CXX object bbque/utils/CMakeFiles/bbque_utils.dir/extra_data_container.cc.o
[ 43%] Building CXX object bbque/utils/CMakeFiles/bbque_utils.dir/schedlog.cc.o
[ 44%] Building CXX object bbque/utils/CMakeFiles/bbque_utils.dir/assert.cc.o
[ 46%] Linking CXX static library libbbque_rloader_rxml.a
make[4]: Leaving directory '/home/giuseppe/Development/mangolibs/bosp/barbeque/build/Debug'
[ 46%] Built target bbque_rloader_rxml
```

Usage

Usage

Target hardware configuration

- Simple example of platform including a workstation (master node) and two embedded boards with ARM processors (slave nodes)
- Master node must include a MANGOLIBS installation
- Slave nodes must include at least the BeeR framework (at this moment)
  - ➔ BeeR daemon must be launched



Usage

System hardware configuration

- Create a **systems.xml** file as root of the description hierarchy

```
[mangolibs #] xdg-open /opt/mango/bosp/etc/bbque/pil/systems.xml
```

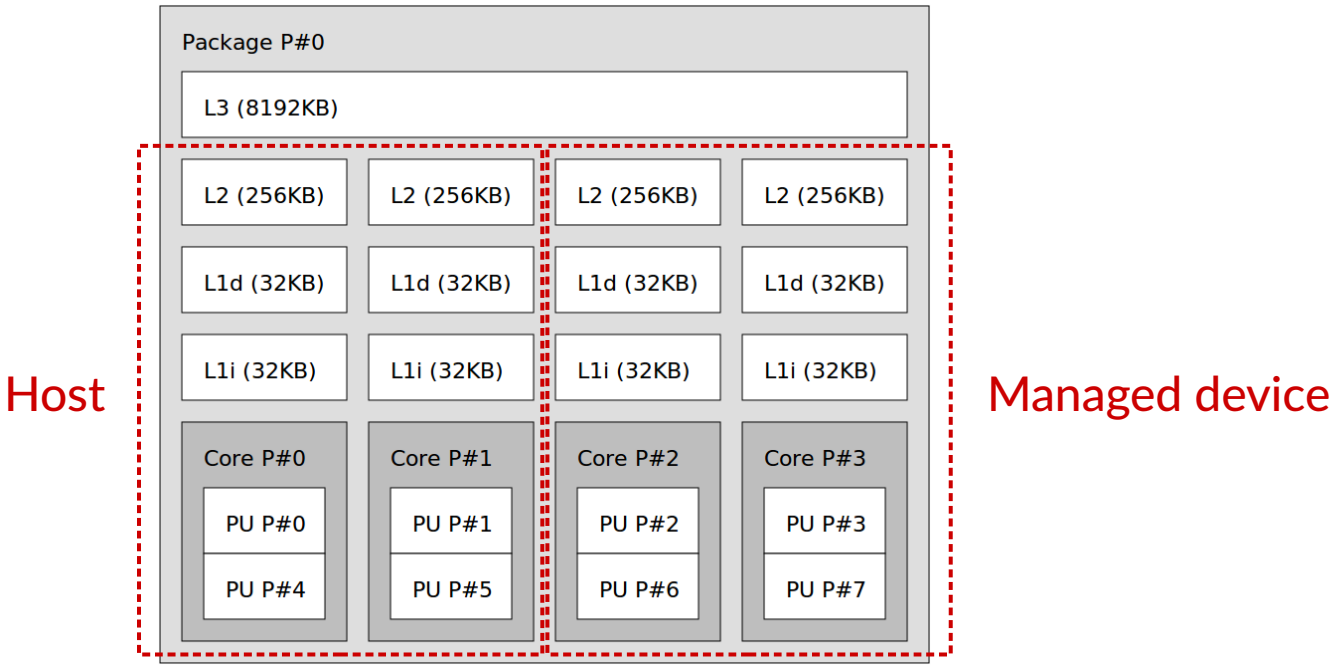
```
<?xml version="1.0" encoding="UTF-8"?>
<systems version="1.0">
    <include local="true">node0.xml</include>
    <include local="false">node1.xml</include>
    <include local="false">node2.xml</include>
</systems>
```

- Our system includes 3 nodes (by default filename = hostname)
- *node0* is the master node, since local to the master instance of the BarbequeRTRM
- We need to create the description file for each node

Usage

System hardware configuration

- CPU partitions concepts
  - **HOST** = set of cores reserved to system processes and resource manager
  - **MANAGED DEVICE** = set of cores reserved to managed applications
  - **SHARED** = cores shared between system processes and managed applications



Usage

System hardware configuration

- Create a **node0.xml** file describing the master node

```
[mangolibs #] xdg-open /opt/mango/bosp/etc/bbque/pil/node0.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<system id="0" hostname="node0" address="10.79.4.243">
    <cpu arch="x86_64" id="0" socket_id="0" mem_id="0">
        <pe id="0" core_id="0" share="100" partition="host"/>
        <pe id="3" core_id="0" share="100" partition="host"/>
    </cpu>
    <cpu arch="x86_64" id="1" socket_id="0" mem_id="0">
        <pe id="1" core_id="1" share="100" partition="mdev"/>
        <pe id="2" core_id="1" share="100" partition="mdev"/>
    </cpu>
    <mem id="0" quantity="16142016" unit="KB"/>
    <netif id="3" name="wlp2s0" />
</system>
```

Usage

System hardware configuration

- Create a **node1.xml** file describing slave node 1

```
[mangolibs #] xdg-open /opt/mango/bosp/etc/bbque/pil/node1.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<system id="1" hostname="node1" address="10.79.4.244">
    <cpu arch="arm_v7" id="0" socket_id="0" mem_id="0">
        <pe id="0" core_id="0" share="100" partition="shared"/>
        <pe id="1" core_id="1" share="100" partition="shared"/>
    </cpu>
    <mem id="0" quantity="2017752" unit="KB"/>
    <netif id="1" name="wlp2s0" />
</system>
```

- Create a **node2.xml** file describing slave node 2
  - → (...similar...)

Usage

BarbequeRTRM start

- Initialize the BOSP shell on the master node
  - ➔ (assumption: /opt/mango/ is the installation path)

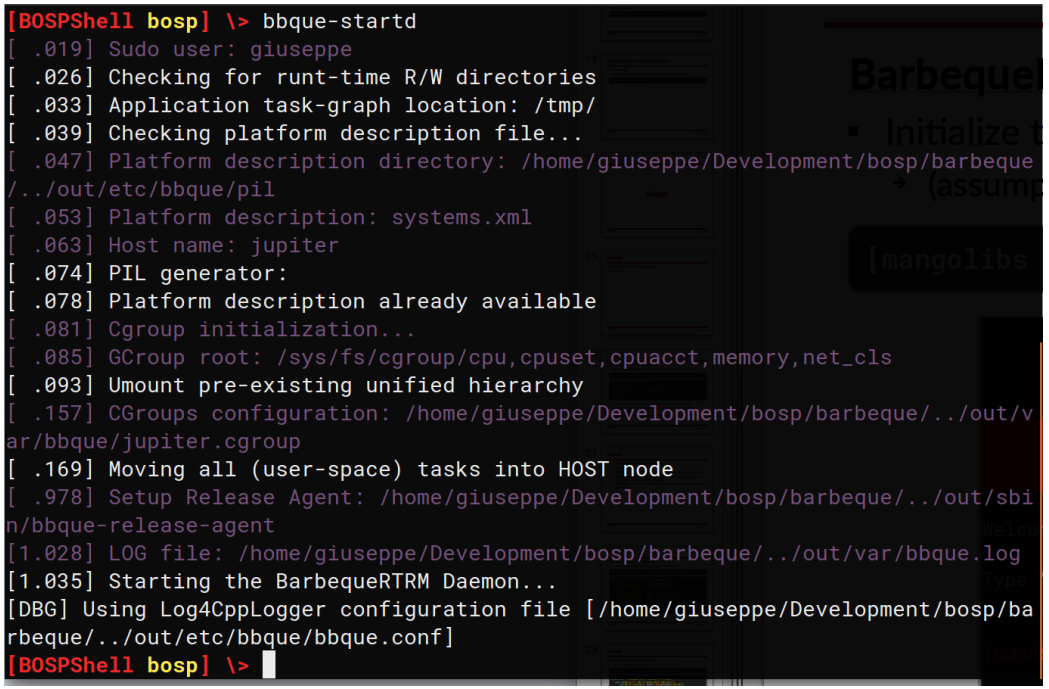
```
[mangolibs $] source /opt/mango/bosp/etc/bbque/bosp_init.env
```

Usage

BarbequeRTRM start

- Launch the daemon
  - Append the option `--nocg` if BarbequeRTRM as been compiled in "GN Emulation Mode"

```
[bosp $] \> bbque-startd
```



Usage

BarbequeRTRM start

- Check the system description loaded through the log file

```
[bosp $] \> tail -n 2000 /opt/mango/bosp/var/bbque.log
```

```
...
... : ================================================================================
... : |   RESOURCES                 I/O | MOD |   USED     | UNRESERVED |    TOTAL    |
... : |-----------------------------+-----+------------+------------+-------------|
... : | sys0.cpu1.pe1                 I |     |        0 % |      100 % |      100 % |
... : | sys0.cpu1.pe2                 I |     |        0 % |      100 % |      100 % |
... : | sys0.mem0                     I |     |        0   |    16.52 G |    16.52 G |
... : | sys0.net3                     I |     |        0   |        0   |        0   |
... : | sys1.cpu0.pe0                 I |     |        0 % |      100 % |      100 % |
... : | sys1.cpu0.pe1                 I |     |        0 % |      100 % |      100 % |
... : | sys1.mem0                     I |     |        0   |     2.01 G |     2.01 G |
... : | sys1.net2                     I |     |        0   |        0   |        0   |
... : | sys2.cpu0.pe1                 I |     |        0 % |      100 % |      100 % |
... : | sys2.cpu0.pe2                 I |     |        0 % |      100 % |      100 % |
... : | sys2.cpu0.pe3                 I |     |        0 % |      100 % |      100 % |
... : | sys2.mem0                     I |     |        0   |     4.05 G |     4.05 G |
... : | sys2.net2                     I |     |        0   |        0   |        0   |
... : ================================================================================
...
```

Usage

BarbequeRTRM configuration

- Edit the 'opt/mango/**bosp/etc/bbque/bbque.conf**' file
  - Change the default resource allocation policy
  - Change the logger verbosity
  - …

```
################################################################################
# Scheduler Manager Options
################################################################################
[SchedulerManager]
#policy = tempura

…
################################################################################
# Log4CPP Logger Configuration
################################################################################
[log4cpp]
###
# Set root category priority and its appender
# By default all logs are only from level greater than or equal to INFO
###
rootCategory = NOTICE, raFile
#rootCategory = DEBUG, raConsole
```

Usage

Launch the BeeR deamon

- The daemon must be in execution on all the slave nodes

```
user@node1:~$ /opt/mango/usr/bin/beer-daemon
```

```
INFO: Started listening on port 9876
INFO: Starting worker thread
INFO: Starting worker thread
INFO: Starting worker thread
INFO: Starting worker thread
```

```
user@node2:~$ /opt/mango/usr/bin/beer-daemon
```

```
INFO: Started listening on port 9876
INFO: Starting worker thread
INFO: Starting worker thread
INFO: Starting worker thread
INFO: Starting worker thread
```

Usage

Launch a sample application

- Once the BarbequeRTRM daemon is up and running…
- We can launch a sample, e.g., **matrix_multiplication**

```
[bosp $] \> /opt/mango/usr/bin/matrix_multiplication 100 100
```

```
…
17:11:18,587 - INFO   exc             : onRelease: Monitoring thread joined
17:11:18,587 - NOTICE exc             : ==============+==========+======================================+======================================
17:11:18,587 - NOTICE exc             : | Application            |          Completion time (ms)        |         Avg. Throughput (CPS)        |
17:11:18,587 - NOTICE exc             : --------------+----------+--------------------------------------+--------------------------------------|
17:11:18,587 - NOTICE exc             : | matrix_multiplication  |                                    2 |                                 0.00 |
17:11:18,587 - NOTICE exc             : ==============+==========+======================================+======================================
17:11:18,587 - INFO   exc             : Control-loop for EXC [matrix_multiplication] TERMINATED
17:11:18,587 - INFO   actrl           : [matrix_multiplication] closing the controller...
17:11:18,587 - INFO   exc             : Unregistering EXC [matrix_multiplication] (@0xa63ca0)...
17:11:18,587 - NOTICE rpc             : Execution statistics:


Cumulative execution stats for 'matrix_multiplication':
  TotCycles    :       1
  StartLatency :      52 [ms]
  AwmWait      :      52 [ms]
  Configure    :       1 [ms]
  Process      :       1 [ms]

# EXC    AWM   Uses Cycles   Total |      Min      Max |      Avg      Var
#================================+==================+==================
matrix_multiplication 001      1      1       1 |    1.841    1.841 |    1.841    0.000
#------------------------+        +------------------+------------------
matrix_multiplication 001         onRun       1 |    1.483    1.483 |    1.483    0.000
matrix_multiplication 001     onMonitor       0 |    0.357    0.357 |    0.357    0.000
#------------------------+--------+------------------+------------------
matrix_multiplication 001   onConfigure       1 |    1.907    1.907 |    1.907    0.000
Matrix multiplication correctly performed
…
```

Release

Current stable release

- Version 0.4 target is the MANGO HPC prototype

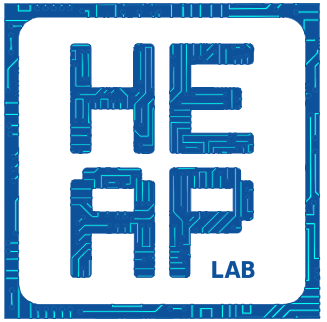
Next release

- Version 0.5 including BeeR framework integration
- **Expected in February 29th, 2020**

Links

HEAPLab Webpage

- http://heaplab.deib.polimi.it
  - Research → Project → MANGO | **MANGOLIBS**
- Facebook: “HEAPLab”



- **BOSP website**
- https://bosp.deib.polimi.it/

POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA,
INFORMAZIONE E BIOINGEGNERIA

HEAP LAB

Programming with MANGO/BeeR

Michele Zanella, Domenico Iezzi

PhD Students at HEAPLab – DEIB, Politecnico di Milano

<name.surname@polimi.it>

MANGO

HiPEAC2020 – Bologna (Italy) – 20-22 January 2020

RECOPE

MANGO API

Different supported languages:

- C++ (main implementation)
- C
- OpenCL
- Python

Application requirements:

- Host-side code
- A Task-Graph definition
- Device-side code (kernels)
- Recipe (optional)

C++ API

mango::TaskGraph

- Class to represent the application Task-Graph composed by Kernels, Buffers and Events.
- Possibility to create an empty TG and then add elements using operators (i.e., **+=**, **-=**). In this case Kernel and Buffer events has to be added manually (see further example)

- **TaskGraph** (std::initializer list< std::shared ptr< Kernel >> lkernels,
  std::initializer list< std::shared ptr< Buffer >> lbuffers,
  std::initializer list< std::shared ptr< Event >> levents={})
  Define a Task-Graph passing all the elements

- **TaskGraph** ()
  Define an empty Task-Graph

C++ API

mango::Event

- Represent a generic Event used to synchronize the application execution
- 3 types of events:
  - Kernel-based (e.g., end of execution)
  - Buffer-based (e.g., finishing of a write or read)
  - Custom-defined
- The first two are generated by the libmango and obtained respectively from the two classes

- **Event** (mango_id_t kernel_id)

- uint32_t **wait**()
  Wait for an event to occur. E.g., waiting the termination of a Kernel.

C++ API

mango::Kernel

- Class to represent and interact with each device-side kernel

- std::shared_ptr<KernelCompletionEvent> **get_termination_event** ()
- std::vector<std::shared_ptr<Event> > & **get_tasks_events**()

mango::KernelFunction

- Class to represent an available kernel binary for a specific architecture

- mango_exit_code_t **load** (const std::string &kernel_file,
  mango_unit_type_t unit,
  mango_file_type_t type)

  Load a new kernel image for a specific architecture type

C++ API

mango::Buffer

- Class to represent each buffer used by the kernels for read/write operations

- std::shared ptr< const Event > **write** (const void* GN_buffer,
  mango_size_t global_size=0)
  Memory transfer from general-purpose node (GN) to the heterogeneous node (HN) in DIRECT mode.

- std::shared ptr< const Event > **read** (void* GN_buffer,
  mango_size_t global_size=0)
  Memory transfer from heterogeneous node (HN) to the general-purpose node (GN) in DIRECT mode.

- std::shared ptr< Event > **get_event** ()

C++ API

Defining a Task Graph

- *Creates an empty Task Graph and adds elements sequentially*

```
mango::TaskGraph *_tg = new mango::TaskGraph()

*_tg += kernel;
*_tg += kernel->get_termination_event();
*_tg += buffer_in;
*_tg += buffer_in->get_event();
*_tg += buffer_out;
*_tg += buffer_out→get_event();
```

- *Creates a completed Task Graph*

```
mango::TaskGraph *_tg = new mango::TaskGraph({kernel},
                                             {buffer_in, buffer_out});
```

C++ API

mango::BBQContext (1/2)

- Interface with the framework
- Application must initialize one object of this class

- **BBQContext** (std::string const &_name="app",
  std::string const &_recipe="generic")
  initialization of the libmango

- virtual mango_exit_code_t **resource_allocation** (TaskGraph &tg)
  Requests TG resource allocation to the resource manager

- virtual mango_exit_code_t **resource_deallocation** (TaskGraph &tg)

- virtual std::shared_ptr<Event> **start_kernel** (std::shared_ptr<Kernel> kernel,
  KernelArguments &args,
  std::shared_ptr<Event> _e=nullptr)
  Launches a given kernel with parameters. Needs a previous successful allocation

C++ API

mango::BBQContext (2/2)

- Interface with the framework
- Application must initialize one object of this class

- std::shared_ptr<Kernel> **register_kernel** (mango_id_t kid, KernelFunction *k,
  std::vector<mango_id_t> in_buffers={},
  std::vector<mango_id_t> out_buffers={})

- std::shared_ptr<Buffer> **register_buffer** (mango_id_t bid,
  mango_size_t size,
  std::initializer_list<mango_id_t> kernels_in={},
  std::initializer_list<mango_id_t> kernels_out={},
  BufferType bt=BufferType::BUFFER)

- std::shared_ptr<Event> **register_event** (std::shared_ptr<Event> event)

BeeR API

mango::BeerContext

- Interface with the BeeR framework to enable a transparent task distribution over different devices
- Application must initialize one object of this class
- Same methods of the mango::BBQContext
- The developers can use the same APIs of the libmango for host and device parts of the application

- **BeerContext** (std::string const &_name=”app”,
  std::string const &_recipe=”generic”)
  initialization of the libmango with BeeR functionalities

C API

- Synchronization primitives for execution and the write/read operations
- Used both for the Host and Device-side

- void **mango_wait** (mango_event_t e)
  Wait for an event to occur. E.g., waiting the termination of a Kernel.

- void **mango_write_synchronization** (mango_event_t event,
  uint32_t value)
  Initialize an event

- uint32_t **mango_read_synchronization** (mango_event_t event)
  Read and reset an event

- uint32_t **mango_lock** (mango_event_t e)
  Lock and read an event

C API

- Library initialization and shutdown

- mango_exit_t **mango_init** (const char *application_name,
  const char *recipe)

- mango_exit_t **mango_release** ()

- Kernel loading

- kernelfunction* **mango_kernelfunction_init** ()
  Initialize a kernel function data structure

- mango_exit_t **mango_load_kernel** (const char *kname,
  kernelfunction *kernel,
  mango_unit_type_t unit,
  Filetype t)

C API

- Registration of task graph components

- mango_kernel_t **mango_register_kernel** (uint32_t kernel_id,
  kernelfunction *kernel,
  unsigned int nbuffer_in,
  unsigned int nbuffer_out,...)

- mango_buffer_t **mango_register_memory** (uint32_t buffer_id,
  size_t size,
  mango_buffer_type_t mode,
  int nkernels_in,
  int nkernels_out, ...)
- mango_event_t **mango_register_event** (int nkernels_in,
  Int nkernels_out,...)

C API

- Task Graph definition

- mango_task_graph_t * **mango_task_graph_vcreate** (mango_kernel_t **kernels,
  mango_buffer_t **buffers,
  mango_event_t ** events)

- mango_task_graph_t * **mango_task_graph_create** (int k, int b, int e,...)

- mango_task_graph_t * **mango_task_graph_add_kernel**(mango_task_graph_t *tg,
  mango_kernel_t *kernel)

- mango_task_graph_t * **mango_task_graph_add_buffer**(mango_task_graph_t *tg,
  mango_buffer_t *buffer)

- mango_task_graph_t * **mango_task_graph_add_event**(mango_task_graph_t *tg,
  mango_event_t *event)

C API

- Resource allocation

- mango_exit_t **mango_resource_allocation** (mango_task_graph_t *tg)
- mango_exit_t **mango_resource_deallocation** (mango_task_graph_t *tg)

- Kernel launch

- mango_event_t **mango_start_kernel** (mango_kernel_t kernel,
  mango_args_t *args,
  mango_event_t event)

C API

- GN-HN data transfer

- mango_event_t **mango_write** (const void *GN_buffer,
  mango_buffer_t HN_buffer,
  mango_communication_mode_t mode,
  size_t global_size)

  Memory transfer from GN to HN

- mango_event_t **mango_read** (void *GN_buffer,
  mango_buffer_t HN_buffer,
  mango_communication_mode_t mode,
  size_t global_size)

  Memory transfer from HN to GN

Device-side API

- For each kernel the developer has to write only the kernel functional code with at least one function marked with the **#pragma mango_kernel** directive

```
#include “dev/mango_hn.h”

#pragma mango_kernel
void kernel_function(uint8_t *buffer_out, uint8_t *buffer_in, ...) {...}
```

Device-side API

- Then during compilation an automated script will create the main function for each kernel with the memory mapping for buffer and events and generate the executable.

```
#include “dev/mango_hn.h”
extern void kernel_function(uint8_t *buffer_out, uint8_t *buffer_in, …);

int main(int argc, char **argv) {
      mango_init(argv);
      uint8_t out=(uint8_t) mango_memory_map (strtol(argv[5], NULL, 16));
      uint8_t in=(uint8_t) mango_memory_map (strtol(argv[5], NULL, 16));
      … // Operation on other parameters
      kernel_function(out, in, …);
      mango_close(42);
}
```

Application Recipe

- An XML file containing a set of resource assignment configurations of the application and per-task requirements suggested to the resource manager.

- It has to be installed in the <MANGOROOT>/bosp/etc/bbque/recipes/ directory by the developer

- Some important sections:
  - **<platform>**: for each target platform the list of resource configurations
  - **<tg>**: specified for each platform
    - **<reqs>** specify the application requirements at task level with these metrics (ctime_ms, throughput_cps, hw_prefs, inbw_kbps, outbw_kbps) *(Not exploited yet)*
    - **<mapping>** kernel-HW mapping specifications *(used by MANGAv2 policy)*

Writing a MANGO/BeeR Application (1/2)

1) Initialize the **libmango**

2) For each Kernel:
   1) Instantiate a KernelFunction object
   2) Load the Kernel
   3) Register the Kernel to the libmango

3) Register each Buffer

4) Create the TaskGraph object

5) Request the Task-Graph resource allocation

6) Setting the Buffer and Event arguments

Writing a MANGO/BeeR Application (2/2)

7) Setting the Kernel arguments

8) Start the Kernels

9) Perform synchronized Buffer writing/reading operations

10) Wait the completion of each Kernels

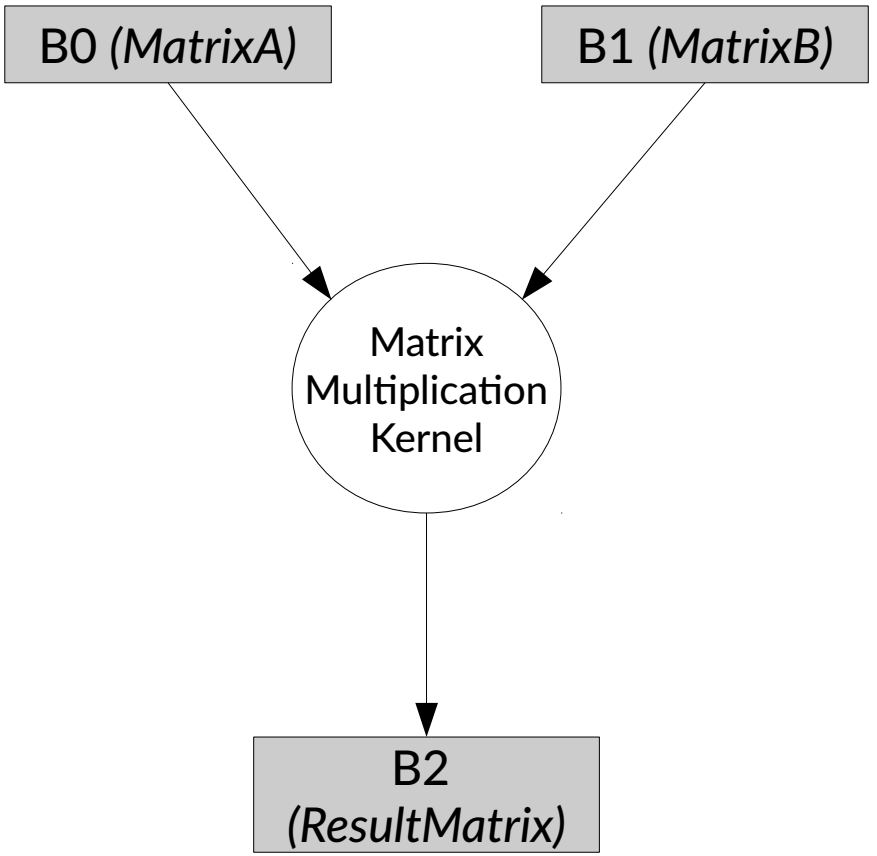
11) Deallocate the resources of the Task-Graph

Sample application: Matrix Multiplication

Host-side: C

Device-side: C

Matrix Multiplication

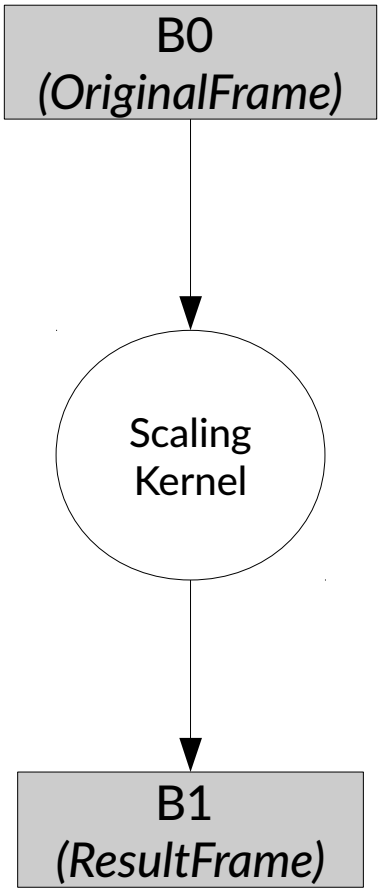


Sample application: GIF Scaling

Host-side: C++

Device-side kernel: C

Gif Scaling



Sample application: Pathfinder

Using The BeeR Framework

Host-side: C++

Device-side kernel: C

Pathfinder

- Adapted from the Rodinia[1] benchmark suite
- Find the path with the lowest accumulated weight from the bottom to the top of a grid, each step moving straight ahead or diagonally
- Work in parallel on different subgrids

1) http://rodinia.cs.virginia.edu/doku.php?id=pathfinder

Pathfinder

Source grid

6	5	1	4	9	6
7	0	2	2	3	8
5	3	4	2	2	3
6	9	8	7	2	5
3	1	5	6	7	3
2	1	8	5	7	9

Result

13	**12**	**19**	**22**	**17**	**19**

Pathfinder

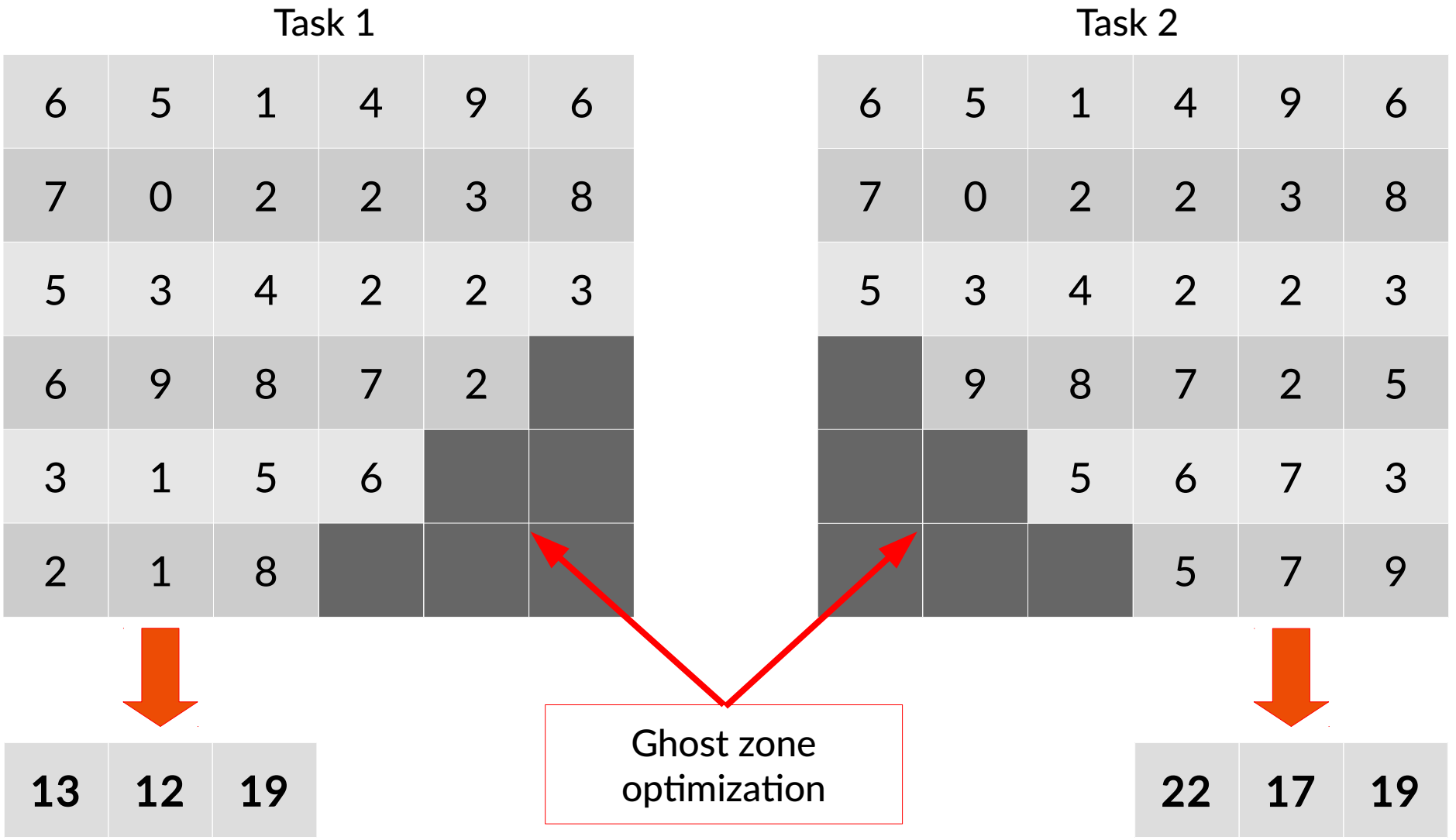
Source grid

6	5	1	4	9	6
7	0	2	2	3	8
5	3	4	2	2	3
6	9	8	7	2	5
3	1	5	6	7	3
2	1	8	5	7	9

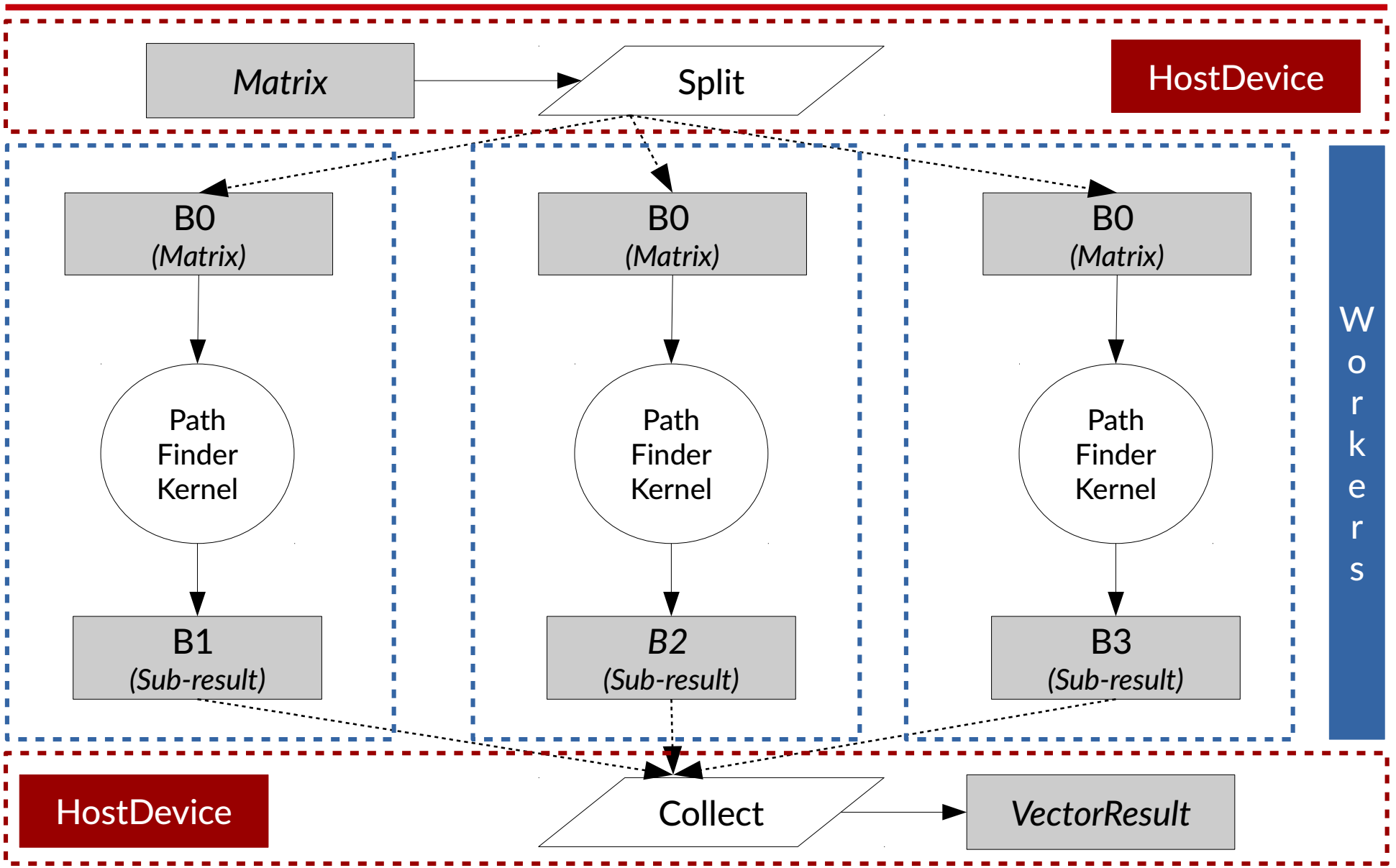
Result

13	**12**	**19**	**22**	**17**	**19**

Pathfinder


Task 1
6 5 1 4 9 6
7 0 2 2 3 8
5 3 4 2 2 3
6 9 8 7 2
3 1 5 6
2 1 8
13 12 19
Task 2
6 5 1 4 9 6
7 0 2 2 3 8
5 3 4 2 2 3
9 8 7 2 5
5 6 7 3
5 7 9
22 17 19
Ghost zone optimization

Pathfinder


Matrix
Split
HostDevice
B0
(Matrix)
B0
(Matrix)
B0
(Matrix)
Path
Finder
Kernel
Path
Finder
Kernel
Path
Finder
Kernel
B1
(Sub-result)
B2
(Sub-result)
B3
(Sub-result)
Workers
HostDevice
Collect
VectorResult

POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA,
INFORMAZIONE E BIOINGEGNERIA

HEAP LAB

DEMO

Giuseppe Massari, Domenico Iezzi,
Michele Zanella, Filippo Sciamanna

<name.surname@polimi.it>

MANGO

HiPEAC2020 – Bologna (Italy) – 20-22 January 2020

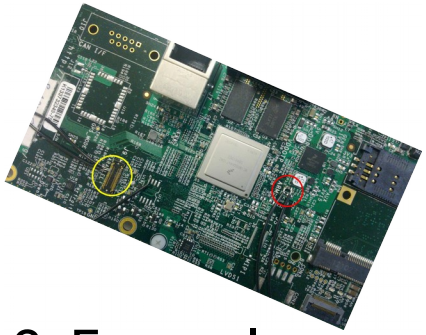
RECOPE

BeeR Pathfinder

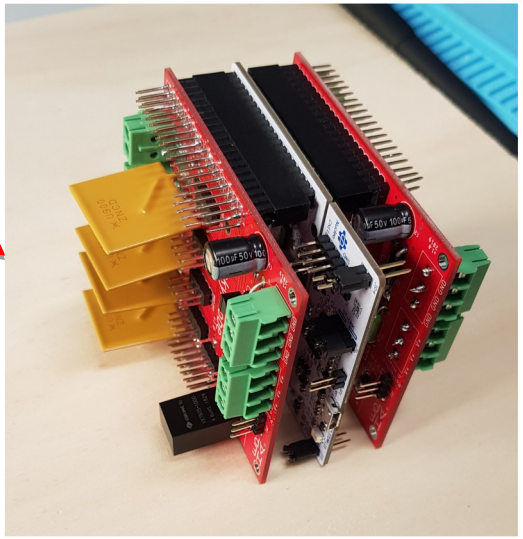
Domenico Iezzi

PhD Student at HEAPLab – DEIB, Politecnico di Milano

SmokyGrill: a Fog cluster test-bed

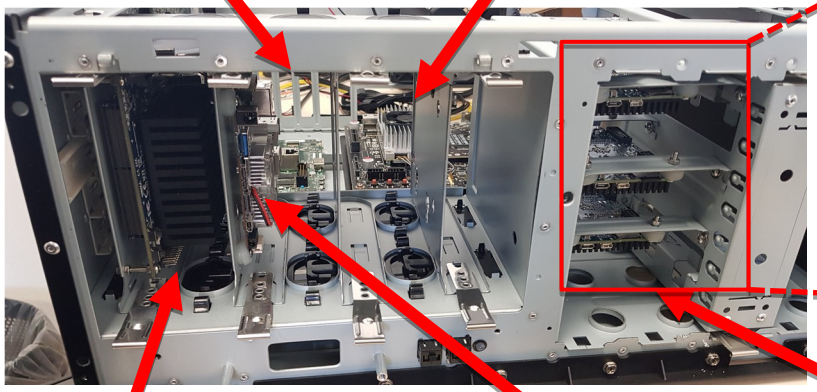


Smoky-Probe



2xFreescale
i.MX6Q

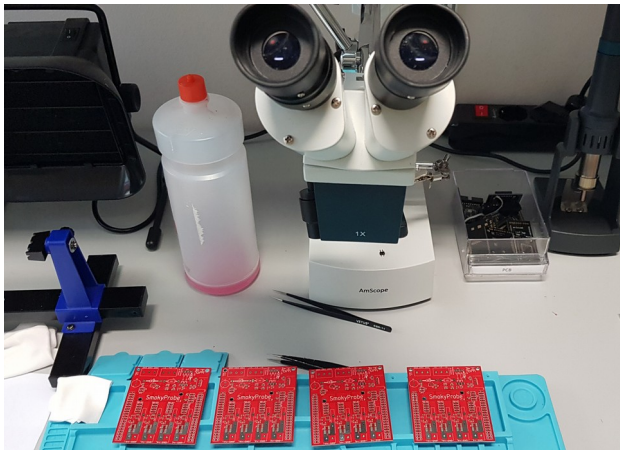
Nvidia
Jetson
TX2



Odroid
H2

Odroid
XU3

3xAdapteva
Parallela

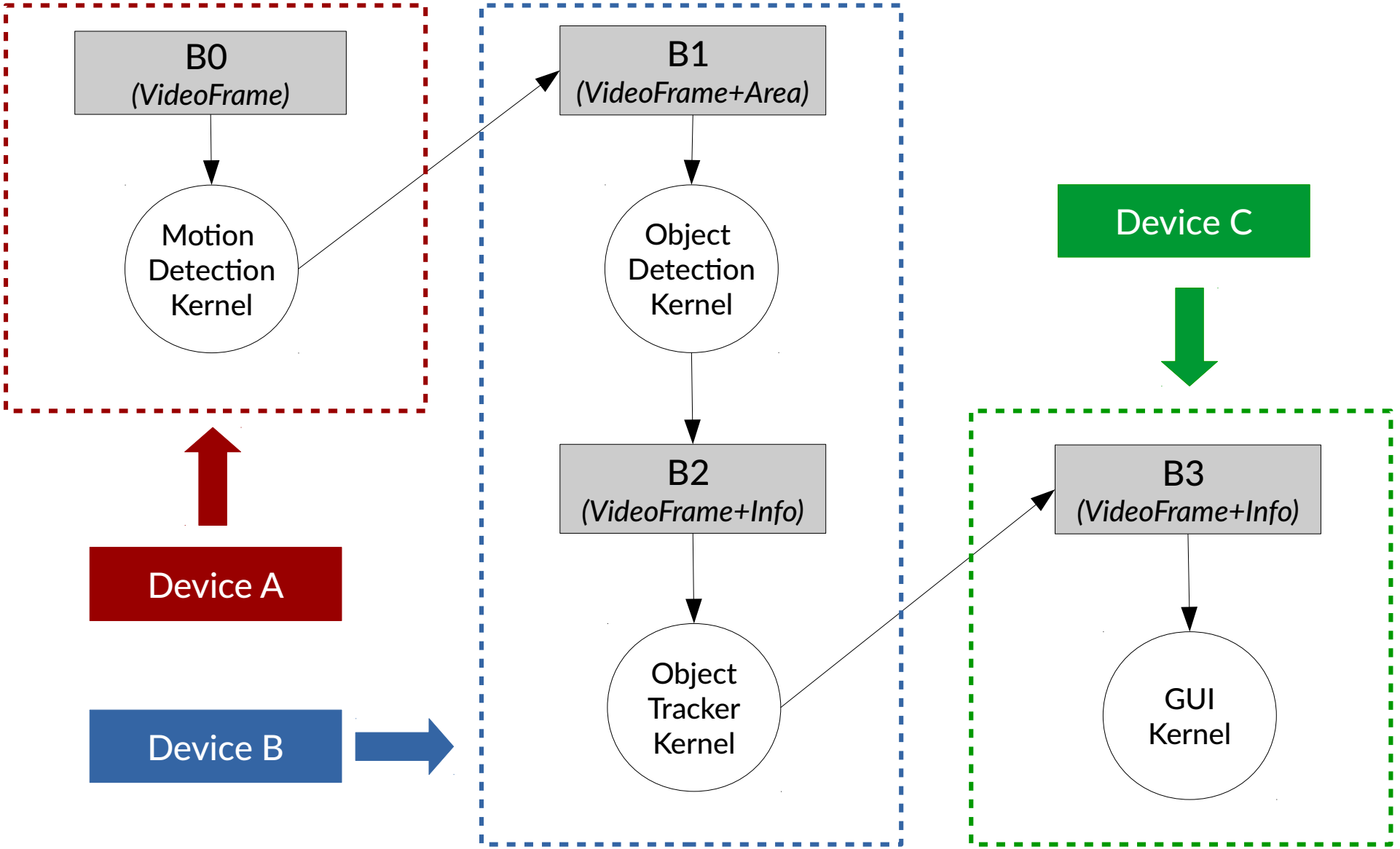


Video Surveillance Fog Application

Filippo Sciamanna

Master Thesis Student at HEAPLab – DEIB, Politecnico di Milano

Michele Zanella

PhD Student at HEAPLab – DEIB, Politecnico di Milano

Video Surveillance