FUSION FOR ENERGY

MARTe2 Users Meeting
RealTime Applications I

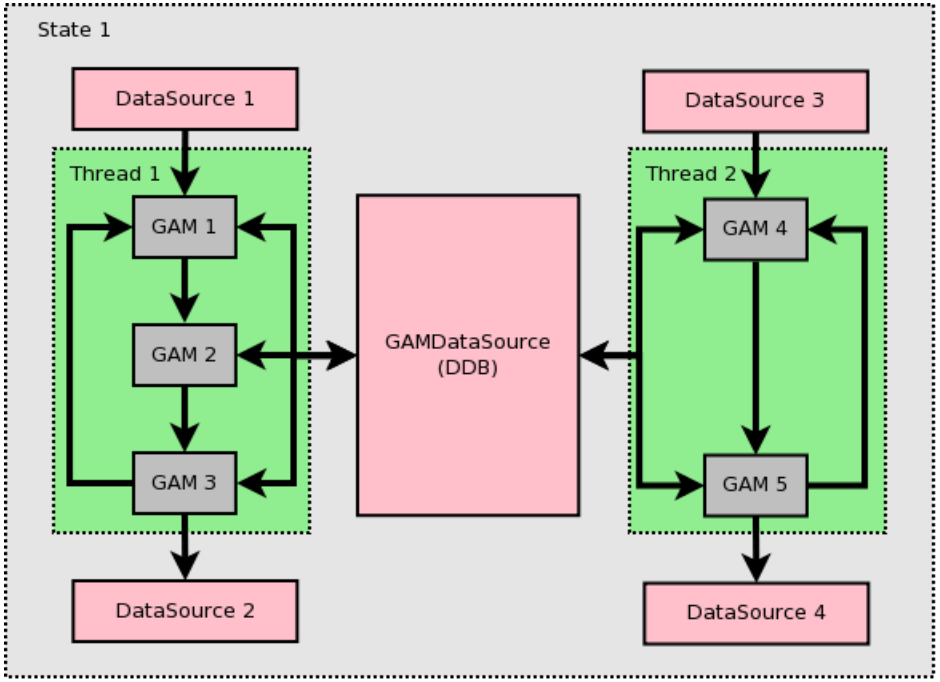
Andre Neto, Filippo Sartori

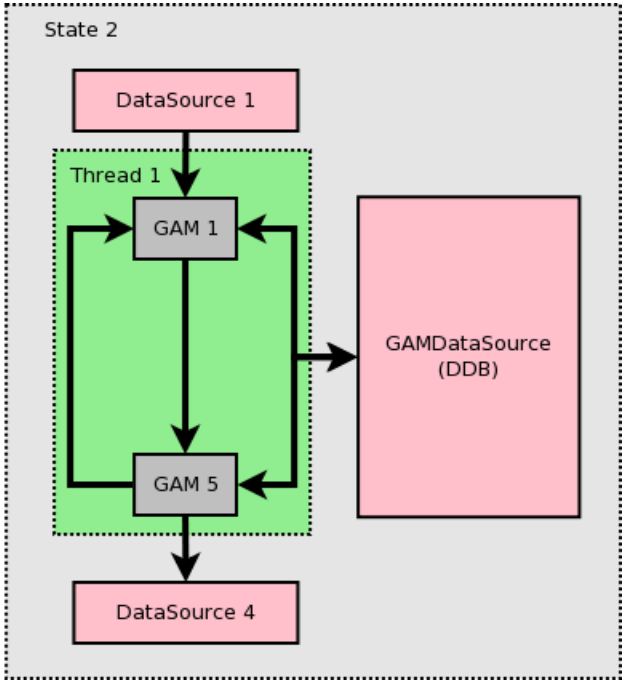
May, 2019

RealTime Application

RealTimeApplication

- Executor of user-functions (GAMs).
- DataSources provide the interface with the hardware.

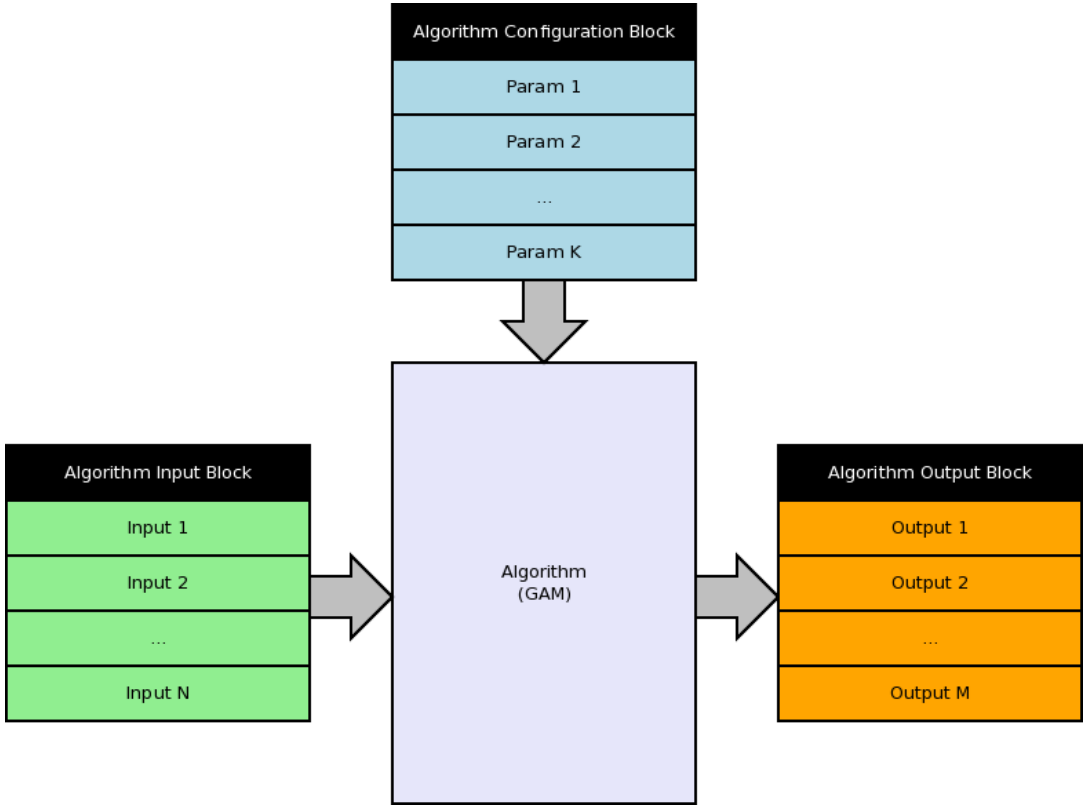




Generic Application Module

Generic Application Module

Component where user-algorithms are to be implemented.



Warning

No interface with operating system (e.g. reading from files/sockets) shall be implemented in the GAMs.
The only exception is memory allocation during configuration

Generic Application Module

Generic Application Module

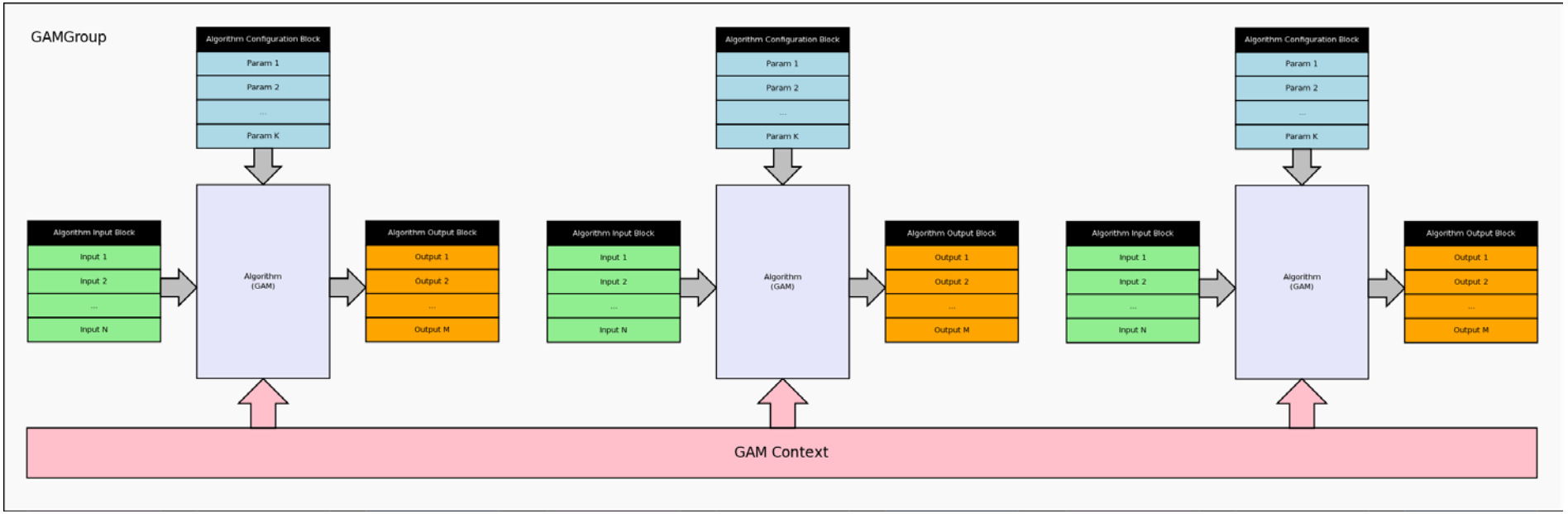
Can be conceptually divided in two sets:

- **Fixed I/O**
  - Inputs and outputs signals (number, type and dimensions) are fixed by design;
  - e.g. integration of existent user-codes with fixed interfaces (e.g. scientific codes);
- **Dynamic I/O**
  - The algorithm behaviour varies with the signal characteristics of a given real-time application.
  - Truly reusable components that can be used across different applications, e.g.:
    - Controllers;
    - Reference generators;
    - Mathematical functions;
    - Type conversion;
    - …

GAMGroup

GAM Group

Component which groups several GAMs together as a single function



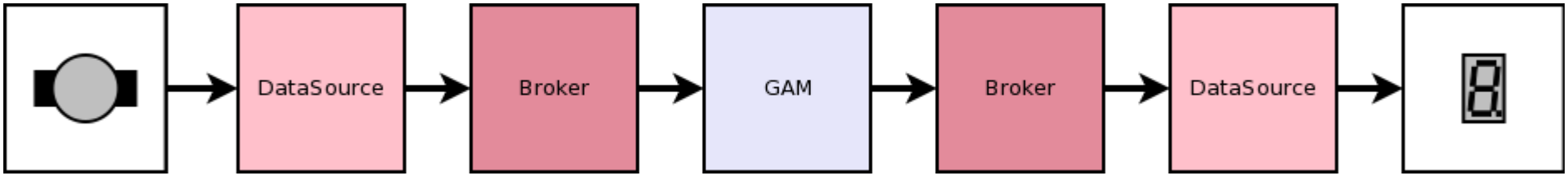
Note

The GAM context can be used to privately share large amounts of information between several GAMs. Typical use-cases: calibration and pre-computation of large matrices.

DataSource & Brokers

DataSources

Real-time interface for the interchange of input and output signals with the hardware.



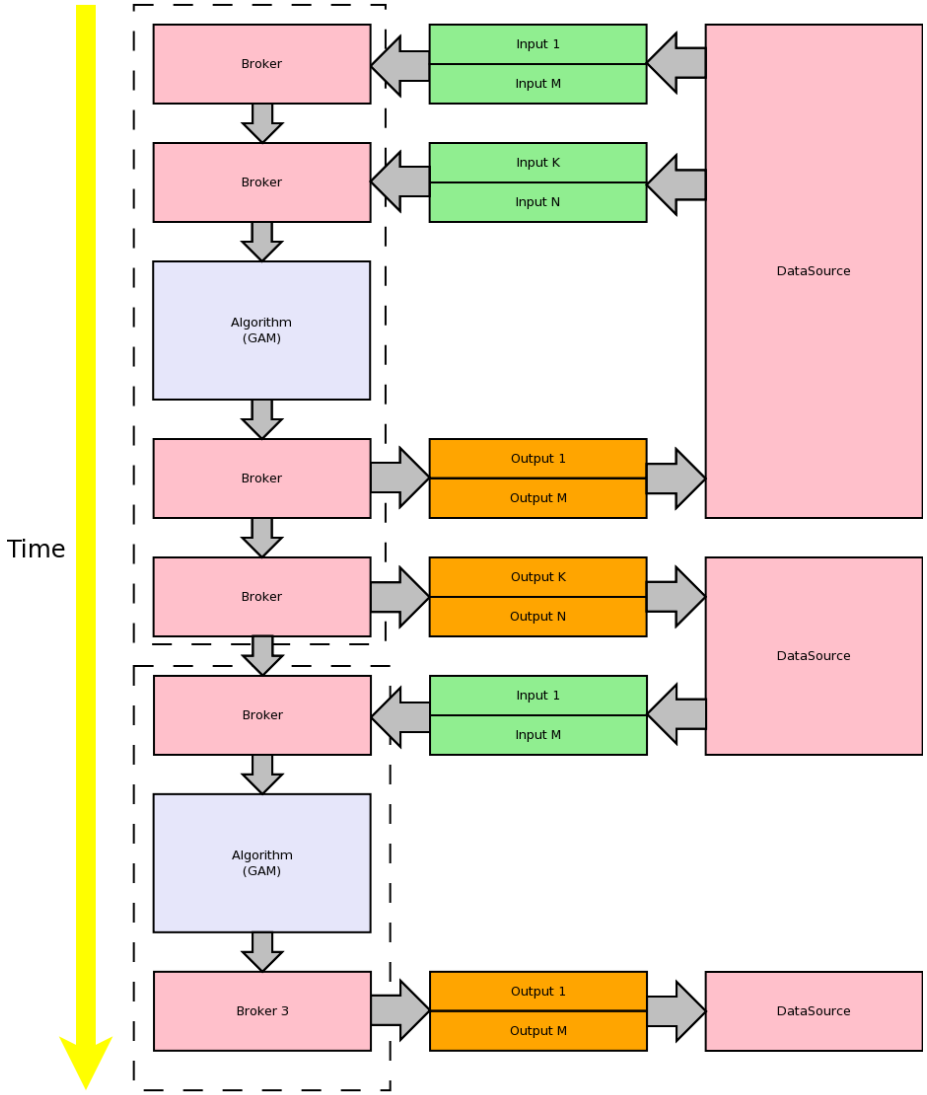
Can *also* be conceptually divided in two sets:

- **Fixed I/O**
  - Inputs and outputs signals (number, type and dimensions) are fixed by design
  - e.g. Timer
- **Dynamic I/O**
  - Behaviour of the hardware is adapted to the signal characteristics of a given real-time application
  - e.g. FileReader

Brokers

Brokers

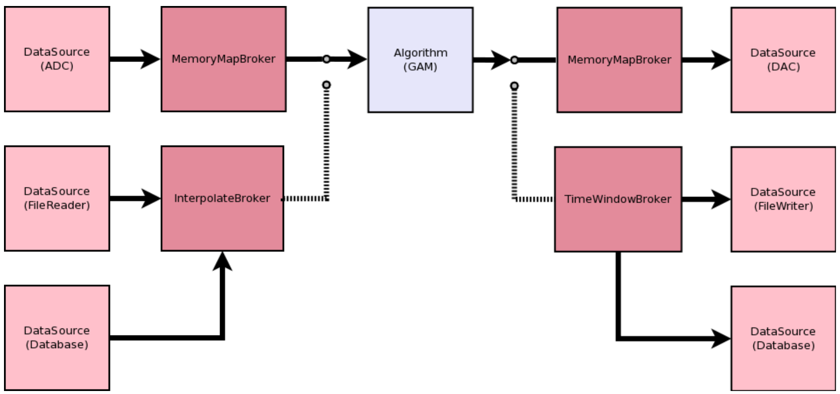
- Provide the interface between the GAMs memory and the DataSource hardware data (typically memory).
- May be specifically developed for a given DataSource
- e.g. because the Broker needs to access an hardware dependent function
- Typically one of the MARTe standard brokers should be used
  - By far one of the most complex interfaces in the library

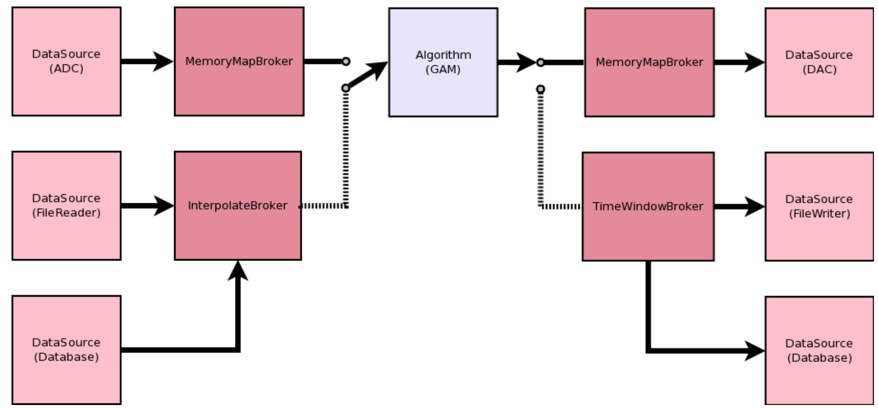


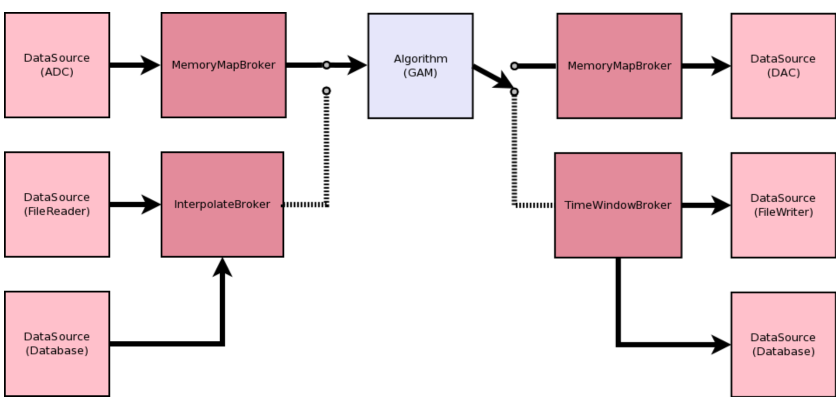
Brokers

Brokers

- All the functions which are related to data transformation should be implemented in a Broker (as opposed to the DataSource)
  - This allows to reuse the same Broker class (e.g. interpolator) in different DataSource implementations.
- Typical use-case: replay experimental data which was not stored at full bandwidth.







Warning

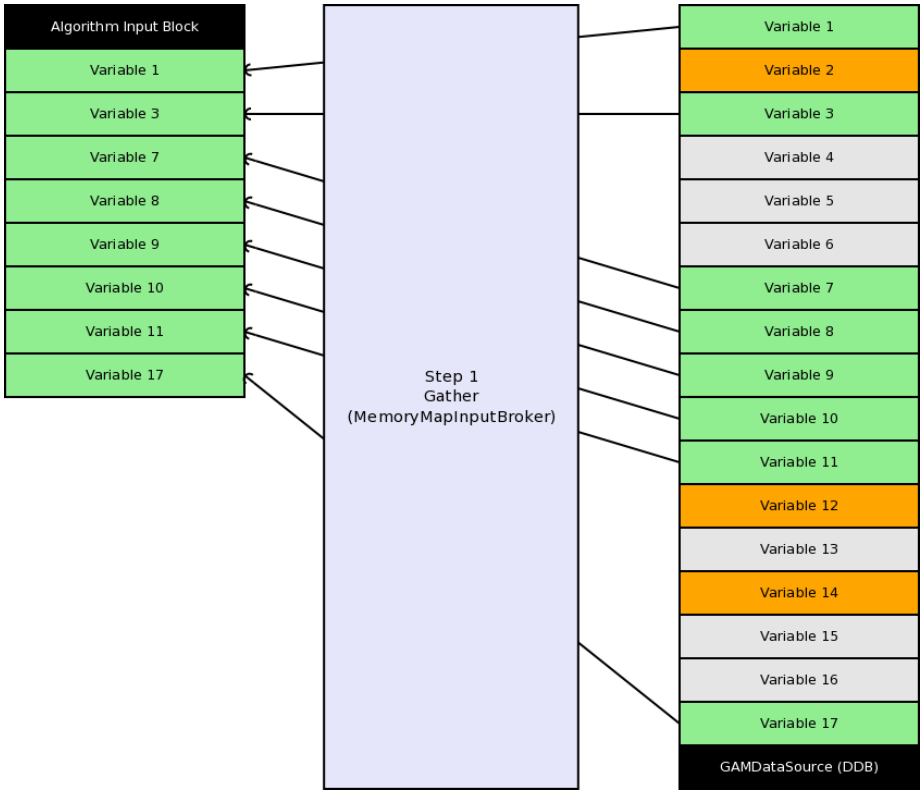
If the synchronisation requires interfacing with the operating system, a decoupling thread shall be used.
The **MemoryMapAsyncOutputBroker** is an example of such type of Broker.

GAMDataSource (DDB)

GAMDataSource (aka DDB)

Framework standard DataSource component for the real-time interchange of data between **GAMs that belong to the same RealTimeThread**.

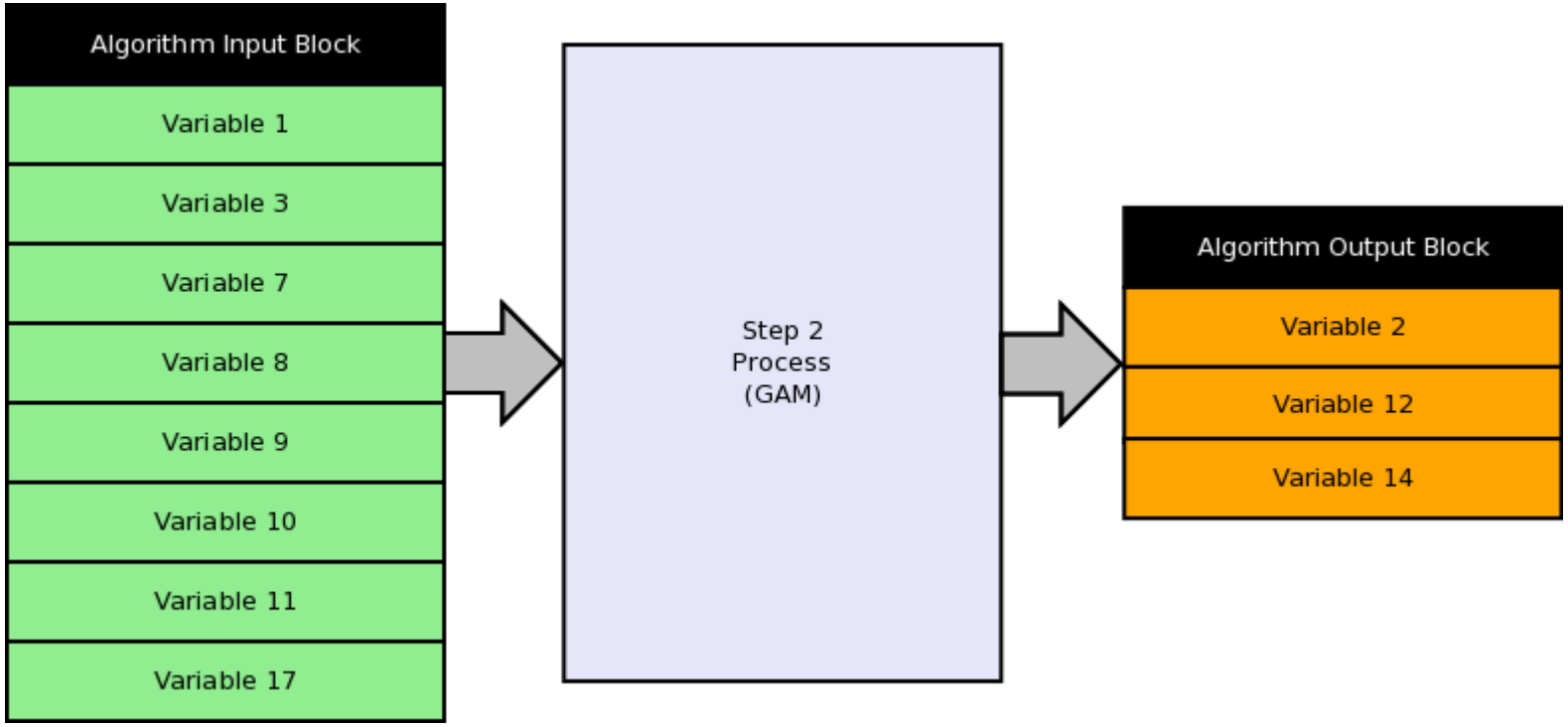
Based on a non-blocking (i.e. without any synchronisation) scatter and gather mechanism.



First the data is copied (gather) to the GAM input signal memory…

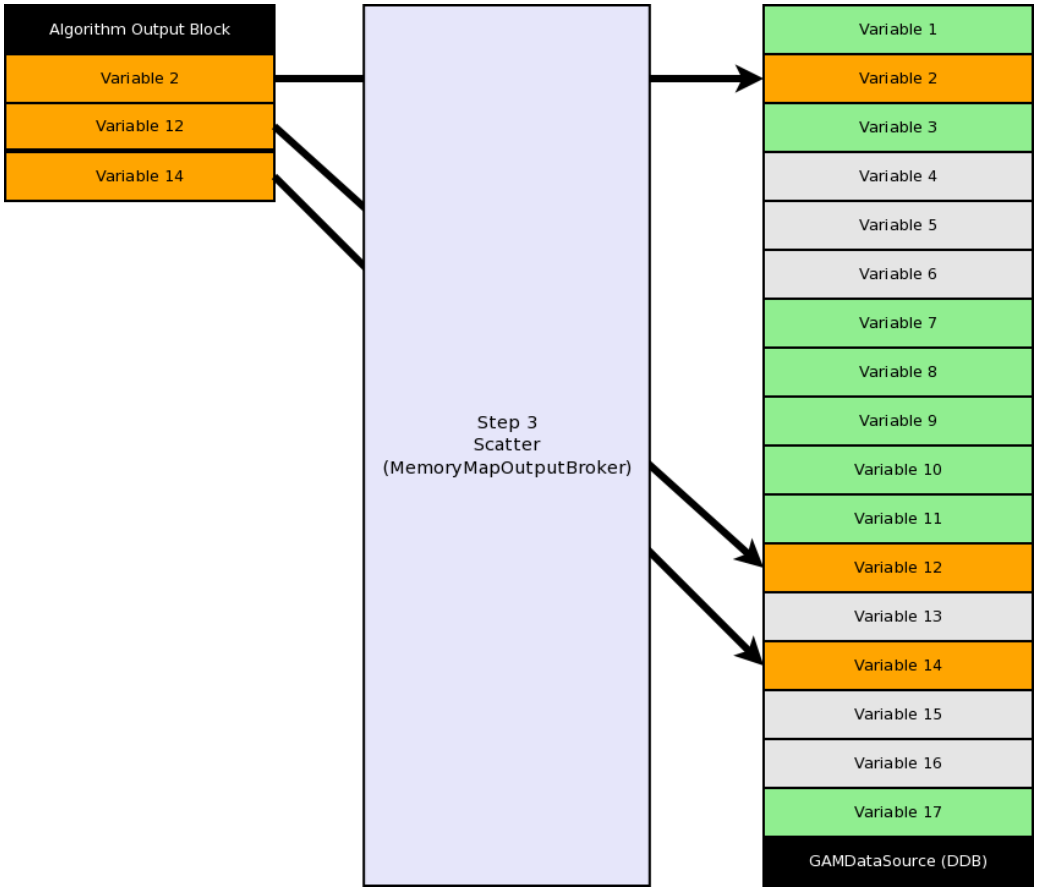
GAMDataSource (DDB)

… after being gathered, the data is processed by the GAM …



GAMDataSource (DDB)

… and finally the data is scattered back to the DDB.



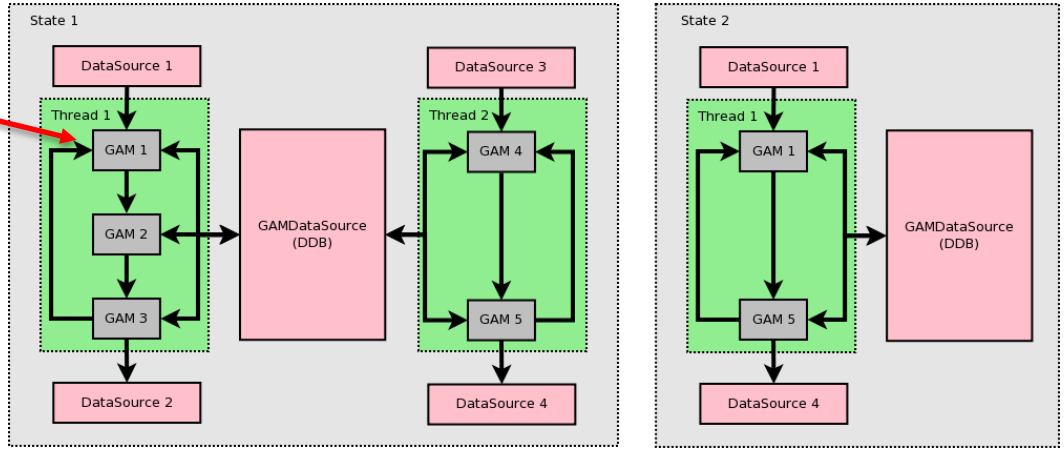
RealTimeApplication Rules

The configuration of a RealTimeApplication has to follow a set of strict rules.

The nodes **Functions**, **Data**, **States** and **Scheduler** shall exist and shall be configured using these rules:

1. GAMs are listed inside a node named Functions of type ReferenceContainer.

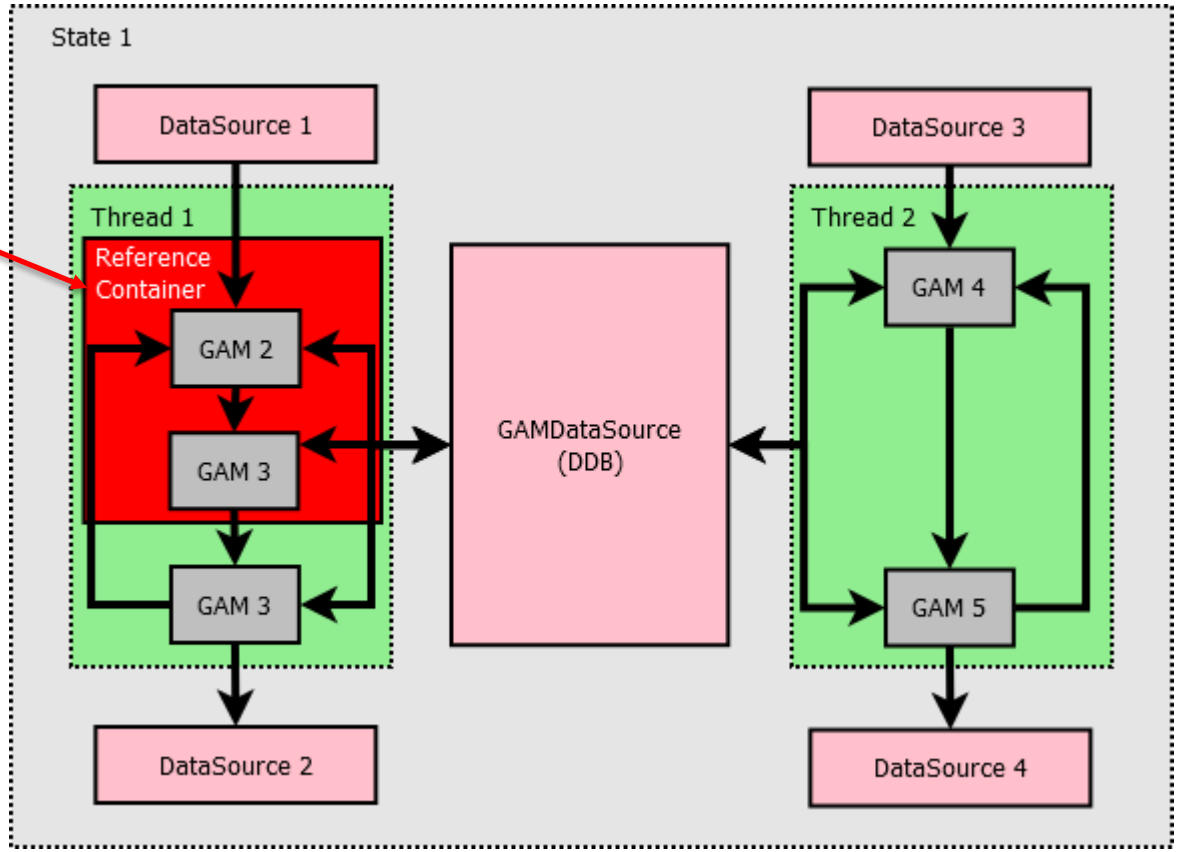
```
$App1 = {
  Class = RealTimeApplication
  +Functions = {
    Class = ReferenceContainer
    +GAM1 = {
      Class = AGAM
      InputSignals = {
        ...
      }
      OutputSignals = {
        ...
      }
    }
    +GAM2 = {
      Class = BGAM
      InputSignals = {
        ...
      }
```



RealTimeApplication Rules

2. GAMs can also be grouped inside reference containers (in order to improve the readability of a configuration stream)

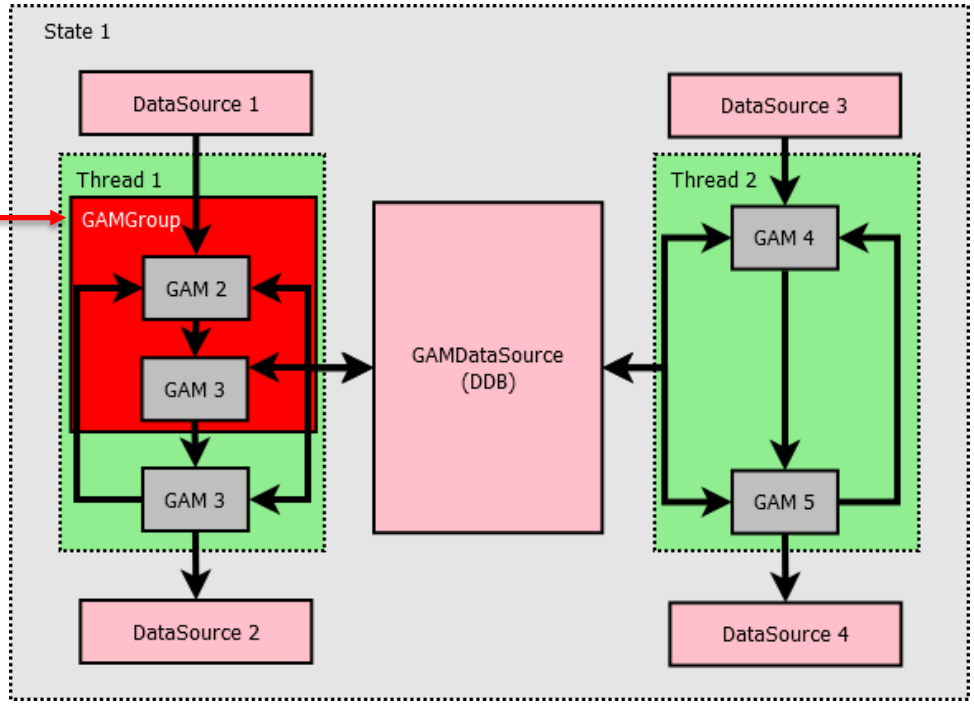
```
+Functions = {
  Class = ReferenceContainer
  +Controllers = {
    Class = ReferenceContainer
    +PID1 = {
      Class = PIDGAM
      InputSignals = {
        ...
      }
      OutputSignals = {
        ...
      }
    }
    +PID2 = {
      Class = PIDGAM
      InputSignals = {
        ...
      }
      OutputSignals = {
        ...
      }
    }
```

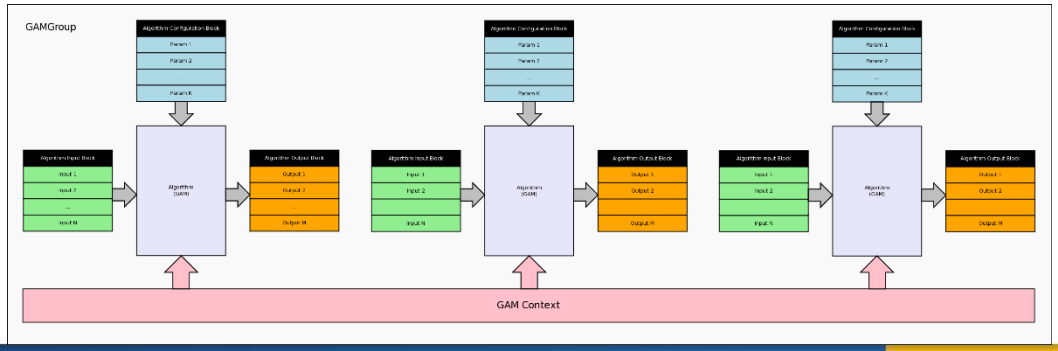


RealTimeApplication Rules

3. GAMs can also be grouped inside reference containers (in order to improve the readability of a configuration stream) and inside GAMGroups.

```
+Functions = {
    Class = ReferenceContainer
    +GAM2 = {
      Class = GAMGroup
      +GAM3 = {
        InputSignals = {
          ...
        }
        OutputSignals = {
          ...
        }
      }
      +GAM4 = {
        InputSignals = {
          ...
        }
        OutputSignals = {
          ...
        }
      }
    ...
```

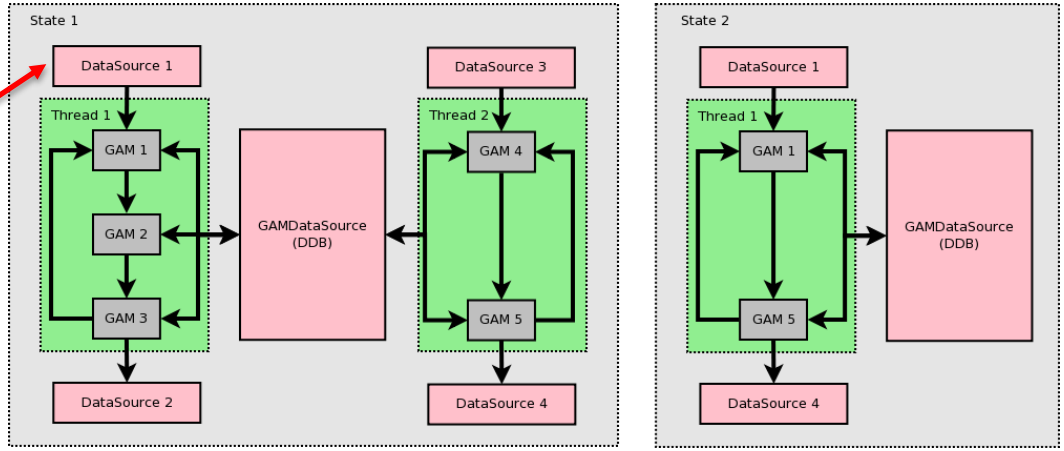




RealTimeApplication Rules

4. DataSources are grouped inside a node named Data of type ReferenceContainer.

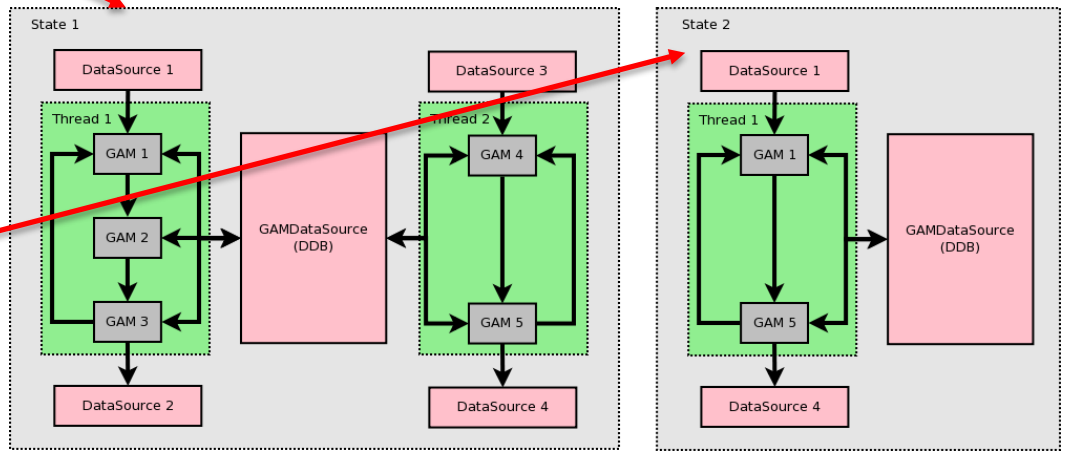
```
$App1 = {
    Class = RealTimeApplication
    +Functions = {
        Class = ReferenceContainer
        ...
    }
    +Data = {
        Class = ReferenceContainer
        DefaultDataSource = DDB1
        +DDB1 = {
            Class = GAMDataSource
        }
        +LoggerDataSource = {
            Class = LoggerDataSource
        }
        +Timings = {
            Class = TimingDataSource
        }
        +Timer = {
            Class = LinuxTimer
            SleepNature = "Default"
            Signals = {
                Counter = {
                    Type = uint32
                }
                Time = {
                    Type = uint32
                }
            }
        }
        ...
```



RealTimeApplication Rules

5. States are listed inside a node named States of type **RealTimeState**. The States nodes is a **ReferenceContainer**.

```
+States = {
    Class = ReferenceContainer
    +State1 = {
      Class = RealTimeState
      +Threads = {
        Class = ReferenceContainer
        +Thread1 = {
          Class = RealTimeThread
          CPUs = 0x1 Functions = {Reference1 Controllers1 GAMGroup1}
        }
        +Thread2 = {
          Class = RealTimeThread
          CPUs = 0x2
          Functions = {Reference2 Controllers2}
        }
      }
    }
    +State2 = {
      Class = RealTimeState
      +Threads = {
        Class = ReferenceContainer
        +Thread1 = {
...
```

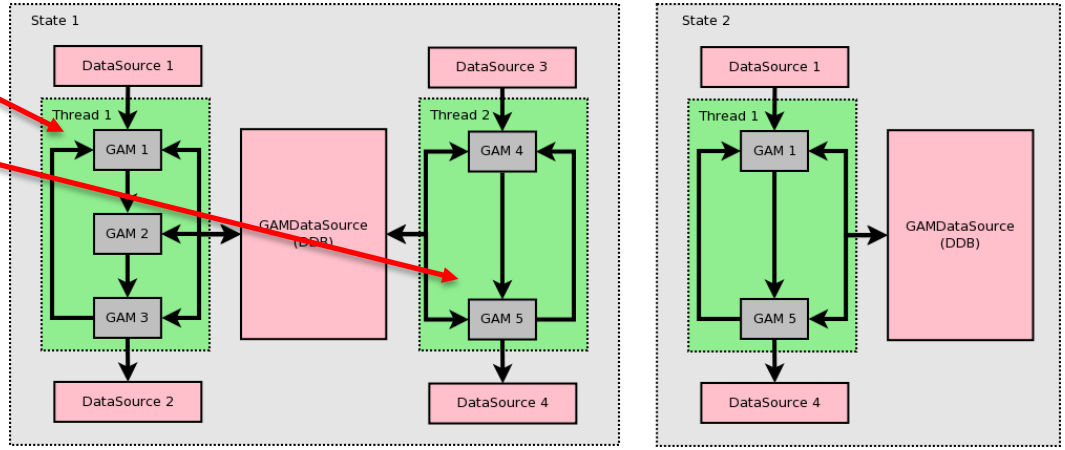


RealTimeApplication states are listed inside a node named States of type RealTimeState.

RealTimeApplication Rules

6. States contain **RealTimeThread** components

```
+States = {
    Class = ReferenceContainer
    +State1 = {
      Class = RealTimeState
      +Threads = {
        Class = ReferenceContainer
        +Thread1 = {
          Class = RealTimeThread
          CPUs = 0x1 Functions = {Reference1 Controllers1 GAMGroup1}
        }
        +Thread2 = {
          Class = RealTimeThread
          CPUs = 0x2
          Functions = {Reference2 Controllers2}
        }
      }
    }
    +State2 = {
      Class = RealTimeState
      +Threads = {
        Class = ReferenceContainer
        +Thread1 = {
...
```



RealTimeApplication Rules

7. The Scheduler defines how the **RealTimeThread** components (and their functions) are executed Standard scheduler executes Functions in sequence – but nothing would prevent the implementation of a parallel scheduler.

```
$App1 = {
  Class = RealTimeApplication
  ...
  +Scheduler = {
    Class = GAMScheduler
    TimingDataSource = Timings
  }
  +Data = {
    Class = ReferenceContainer
    DefaultDataSource = DDB1
    +Timings = {
      Class = TimingDataSource
    }
          ...
```

Note

The TimingDataSource stores performance figures about all the Functions and the RealTime thread

Exercises

- Run basic configuration
- Add GAM to the list of Functions
- Modify existent GAM
- Add State to configuration
- Add Thread to configuration

Make sure you can compile

```
cd ~/Projects/MARTe2-demos-padova/
export MARTe2_DIR=~/Projects/MARTe2-dev
export MARTe2_Components_DIR=~/Projects/MARTe2-components/
make -f Makefile.x86-linux
```

Basic configuration file

Objective: run a simple MARTe real-time application configuration file

```
cd ~/Projects/MARTe2-demos-padova/Startup/
./Main.sh -l RealTimeLoader -f ../Configurations/RTApp-1-1.cfg -s State1
```

Success: application executes and console is regularly updated

Learn more:

- Open Configurations/RTApp-1-1.cfg
- How many GAMs are being executed?
- What is an IOGAM?
  - https://vcis-jenkins.f4e.europa.eu/job/MARTe2-components/doxygen/classMARTe_1_1IOGAM.html

Modify basic configuration file

Objective: modify MARTe real-time application configuration file to add a new GAM

- Modify Configurations/RTApp-1-2.cfg
- Add the **GAMDisplay** to the list of executed Functions
- Run the application

```
cd ~/Projects/MARTe2-demos-padova/Startup/
./Main.sh -l RealTimeLoader -f ../Configurations/RTApp-1-2.cfg -s State1
```

Success: application executes and console is regularly updated

Learn more:

- What happens if the GAMDisplay is not added?

Modify existent GAM

Objective: modify existent GAM

- Modify GAMs/FixedGAMExample1/FixedGAMExample1.cpp
- Multiply the input signals by the **gain** variable
- Compile

```
cd ~/Projects/MARTe2-demos-padova/
export MARTe2_DIR=~/Projects/MARTe2-dev
export MARTe2_Components_DIR=~/Projects/MARTe2-components/
make -f Makefile.x86-linux
```

- Run the application

```
cd ~/Projects/MARTe2-demos-padova/Startup/
./Main.sh -l RealTimeLoader -f ../Configurations/RTApp-1-3.cfg -s State1
```

Success: application executes and console is regularly updated with value multiplied by the gain

Learn more:

- How was the value of the **Gain** set in the configuration file?

Add a new state

Objective: run a MARTe real-time application configuration file with two states

- Modify Configurations/Modify RTApp-1-4.cfg
- Add a new state, named **State2**, which executes the following Functions: GAMTimer FixedGAM2 GAMDisplay
- Run the application (note State2 in the arguments)

```
cd ~/Projects/MARTe2-demos-padova/Startup/
./Main.sh -l RealTimeLoader -f ../Configurations/RTApp-1-4.cfg -s State2
```

Success: application executes and console is regularly updated with value multiplied by the 15x gain

Learn more:

- Did you notice that there are two instances of the same GAM: FixedGAM1 and FixedGAM2 with two different settings?

Add a new thread

Objective: run a MARTe real-time application configuration file with two independent threads

- Modify Configurations/Modify RTApp-1-5.cfg
- Add a new thread, named **Thread2**, which executes the following Functions: GAMTimer2 FixedGAM2 GAMDisplay2
- Run the application

```
cd ~/Projects/MARTe2-demos-padova/Startup/
./Main.sh -l RealTimeLoader -f ../Configurations/RTApp-1-5.cfg -s State1
```

Success: application executes and console is regularly updated with value multiplied by the 5x and by the 15x gain

Learn more:

- Did you notice that:
  - The GAMTimer and the GAMDisplay had to be replicated?
  - There are two GAMDataSources (DDBs)?

FUSION FOR ENERGY

Thank you for your attention

Follow us on:

www.f4e.europa.eu

www.twitter.com/fusionforenergy

www.youtube.com/fusionforenergy

www.linkedin.com/company/fusion-for-energy

www.flickr.com/photos/fusionforenergy