Tasking Framework from A to Z

Embedded System Week 2020

Zain Hammadeh

DLR

Tasking Framework

- ✔ Open-source
- ✔ Event-driven
- ✔ Multithreading
- ✔ C++
- ✔ Development library
- ✔ Execution platform

- ✔ Embedded systems with static memory requirements
- ✔ Non-embedded software
- ✔ Compatible with POSIX-like OS: Linux, RTEMS

- X Operating system
- X Testing platform

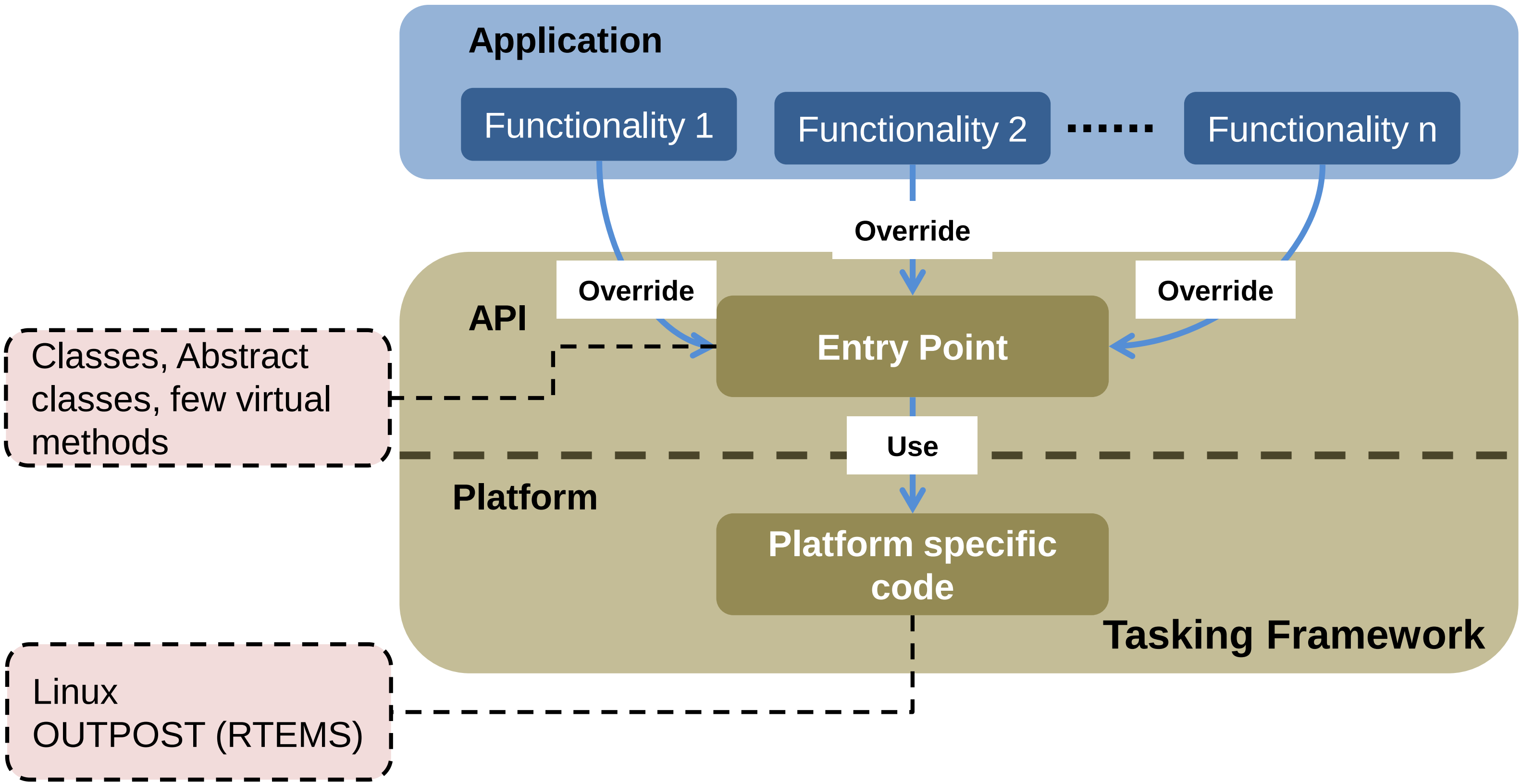
Open-source

On-line

https://github.com/DLR-SC/tasking-framework

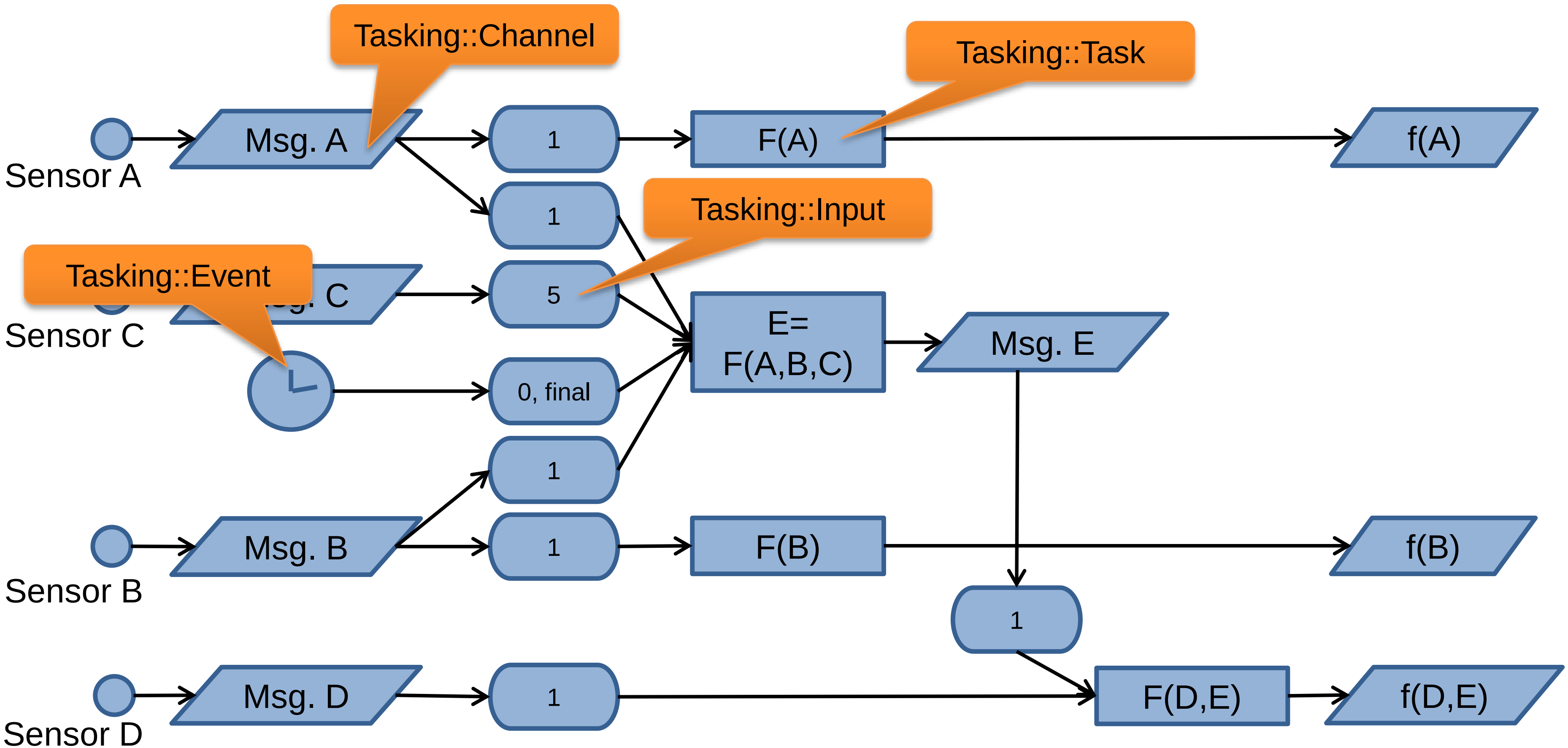
Copyrights

Solution strategy



Application Programing Interface (API)

Elements


Tasking::Channel
Tasking::Task
Tasking::Input
Tasking::Event
Sensor A
Msg. A
1
1
F(A)
f(A)
Sensor C
5
0, final
E=
F(A,B,C)
Msg. E
1
Sensor B
Msg. B
1
F(B)
f(B)
1
Sensor D
Msg. D
1
F(D,E)
f(D,E)

Concept

Separation between data and functionality

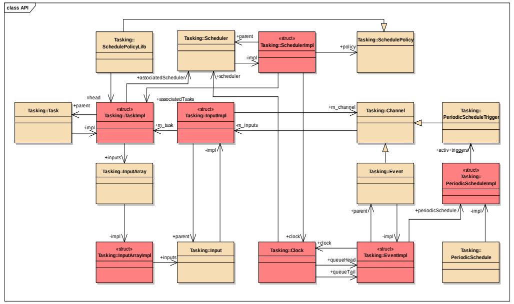
Channel_1 → Task_1 → Channel_2 → Task_2

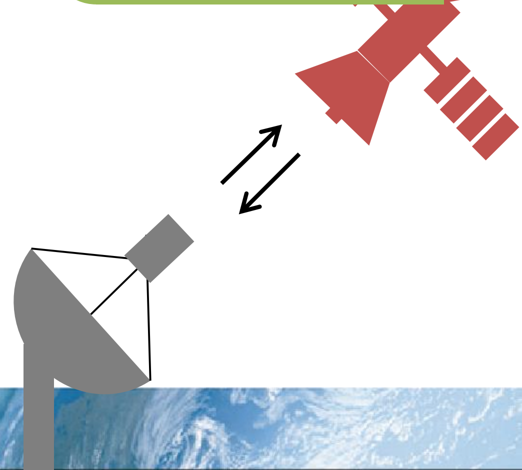
- Channel_1: Data container
- Channel_2: Interface

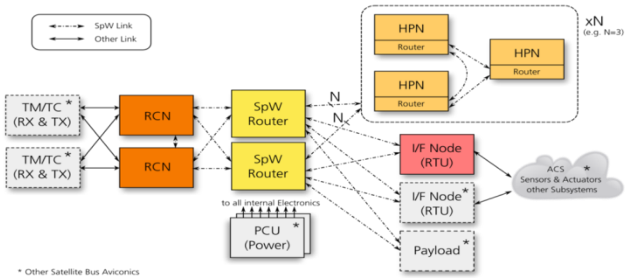
✔ Reusability

✔ Data handling

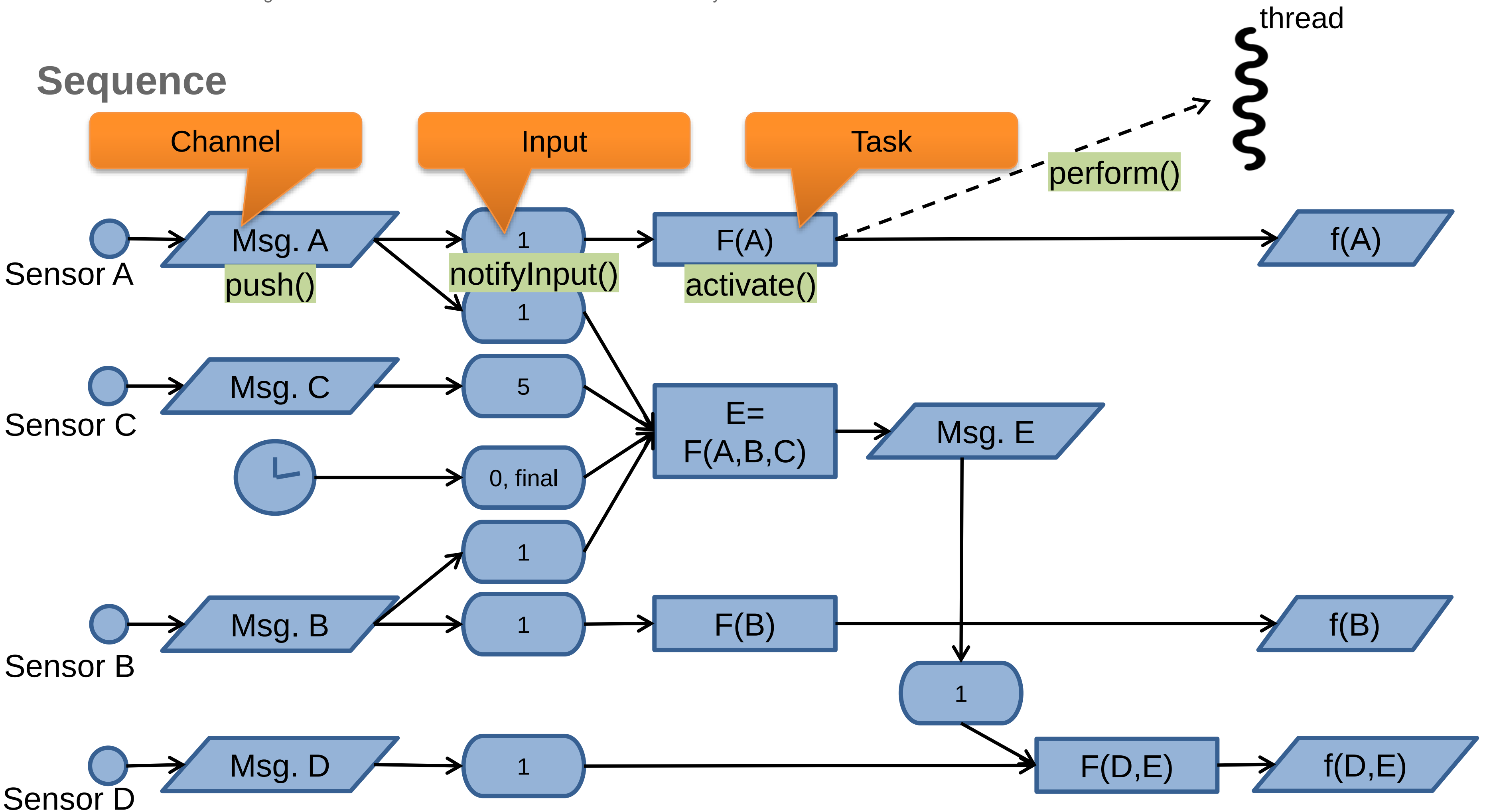
✔ Distributed systems





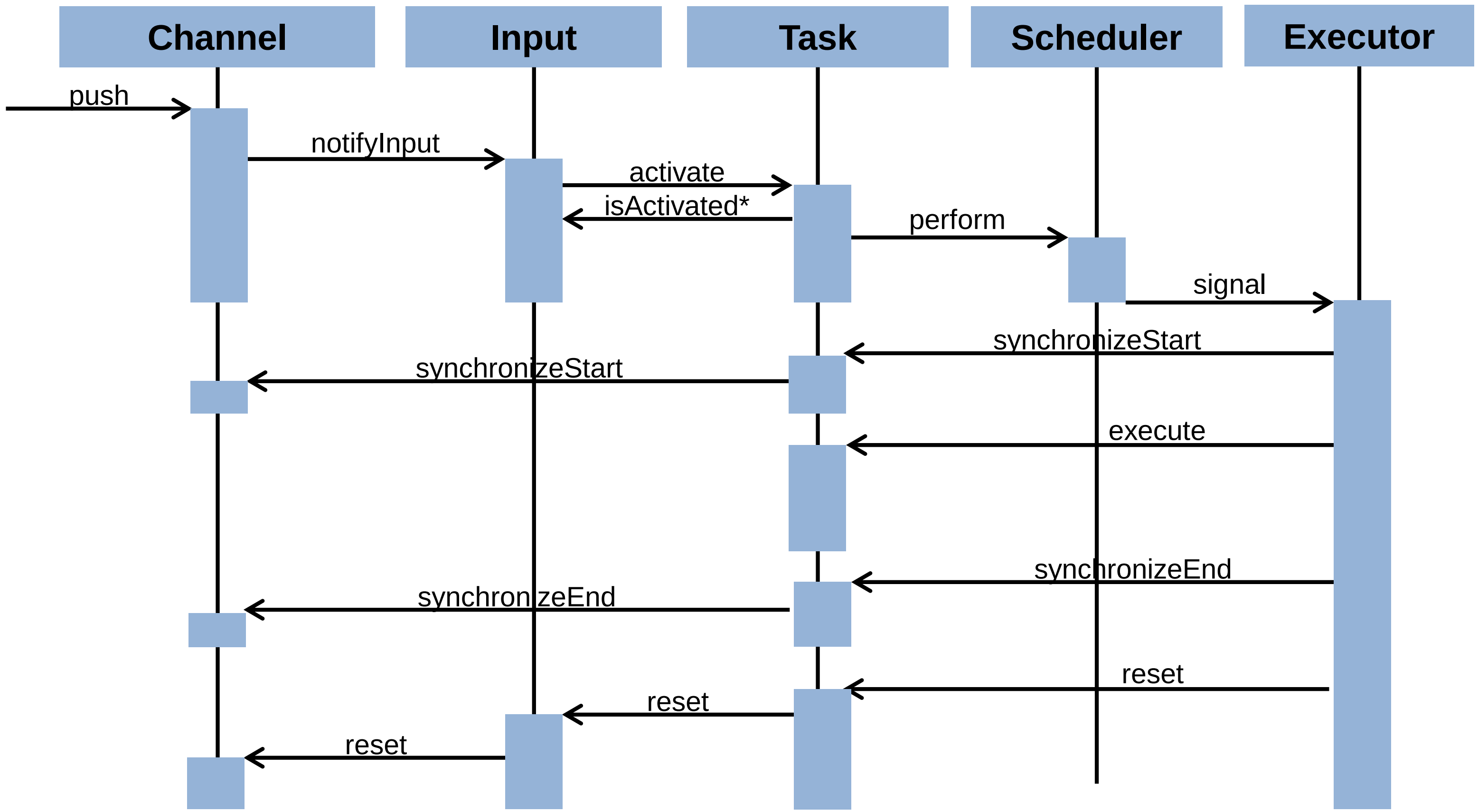


Sequence

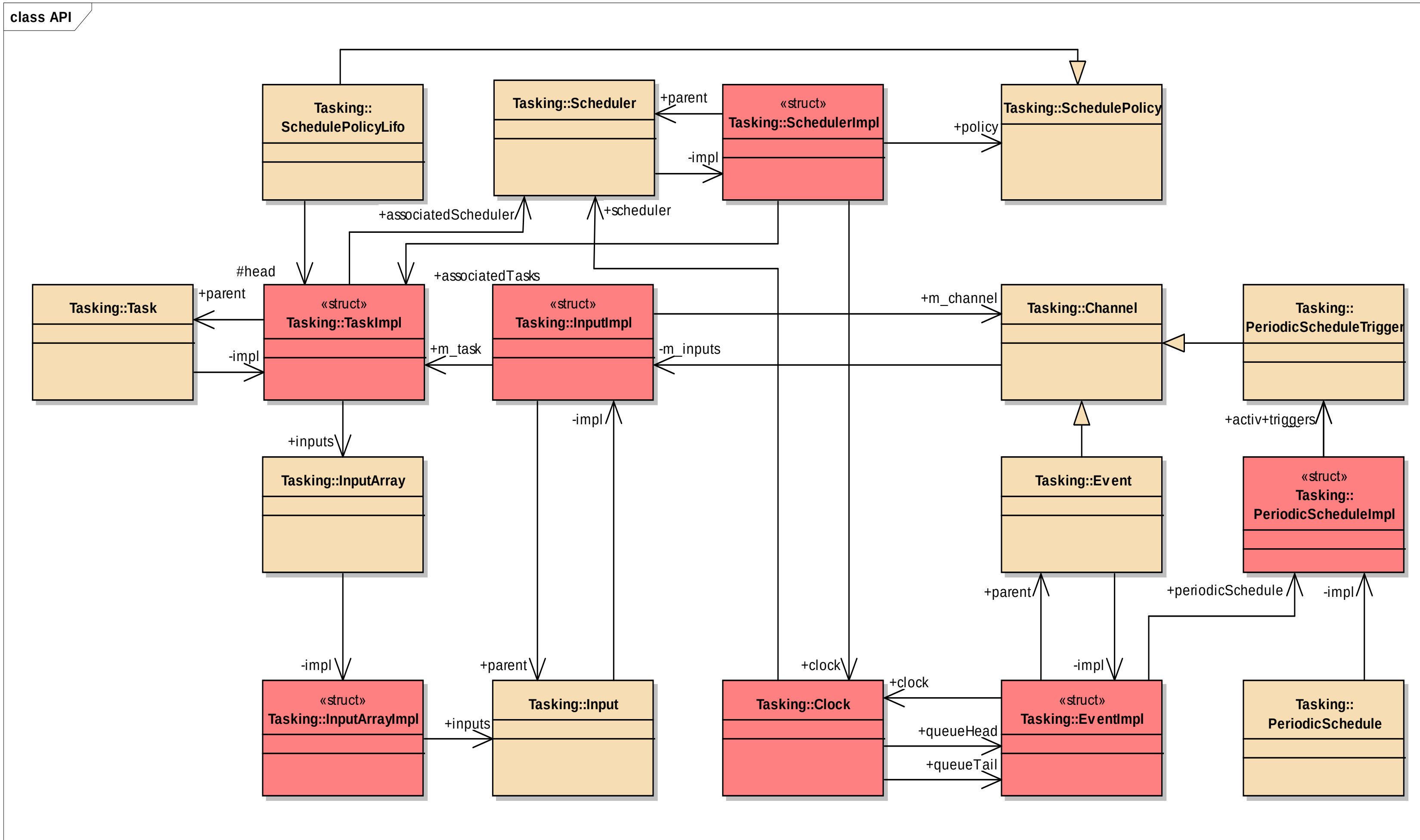


thread
Channel
Input
Task
perform()
Sensor A
Msg. A
push()
1
notifyInput()
1
F(A)
activate()
f(A)
Sensor C
Msg. C
5
0, final
E=
F(A,B,C)
Msg. E
1
Sensor B
Msg. B
1
F(B)
f(B)
1
Sensor D
Msg. D
1
F(D,E)
f(D,E)

Sequence cont'd

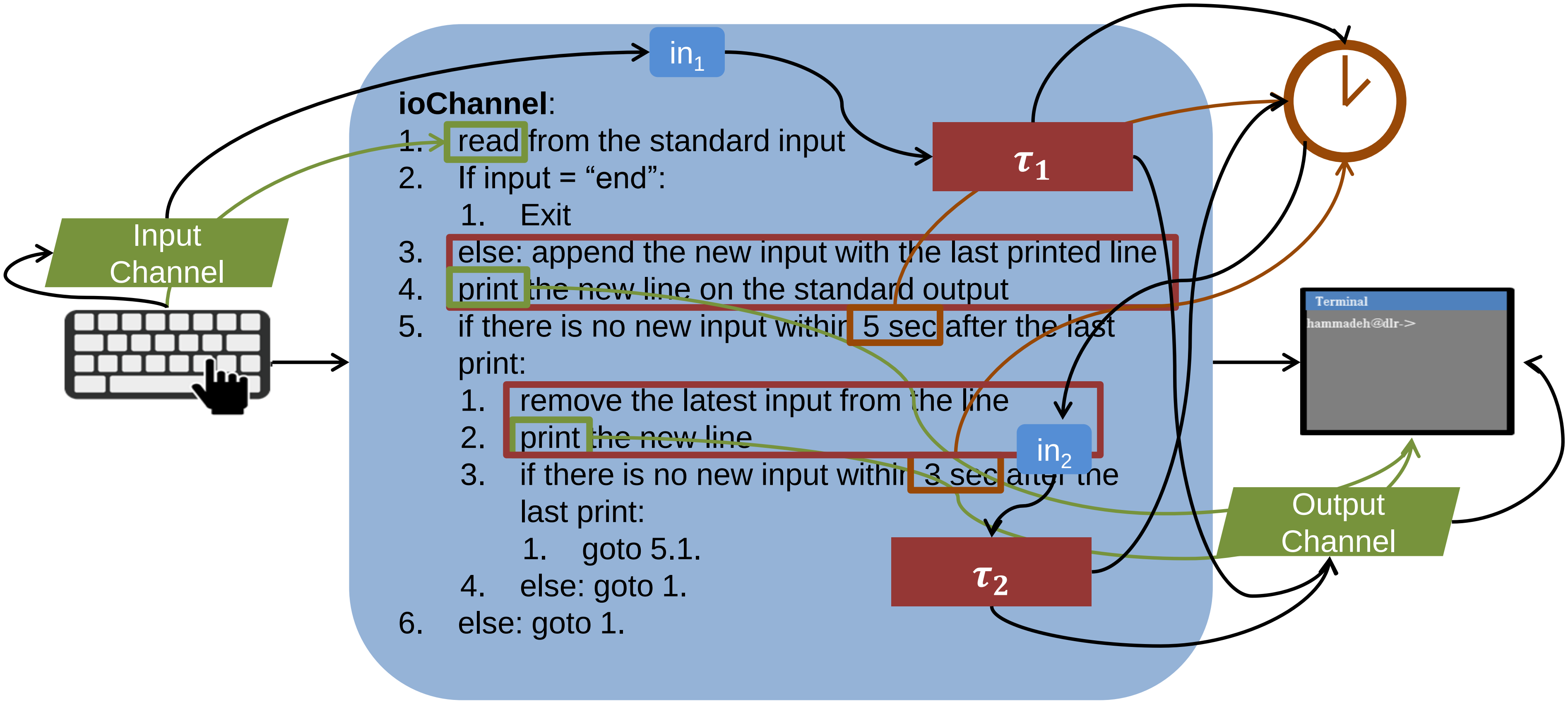


API

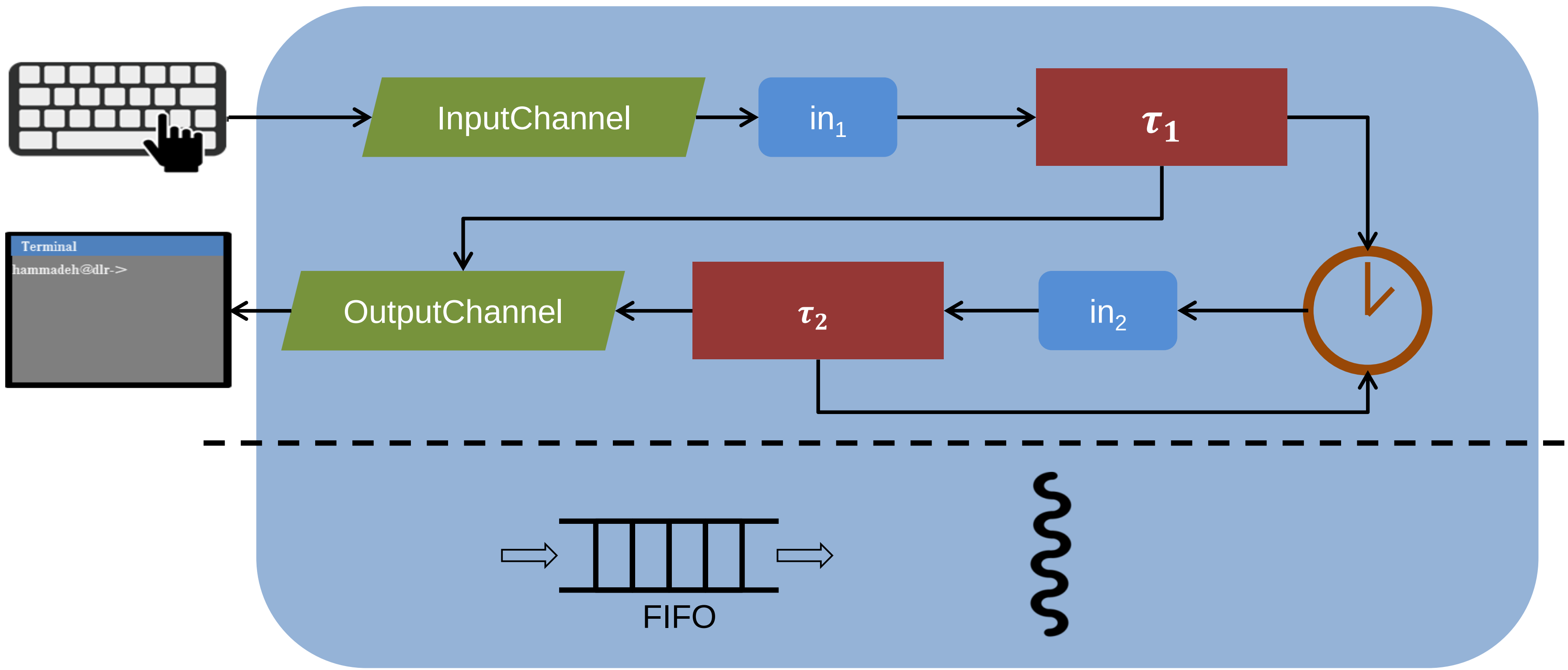

class API
Tasking::SchedulePolicyLifo
Tasking::Scheduler
«struct» Tasking::SchedulerImpl
Tasking::SchedulePolicy
+parent
-impl
+policy
+associatedScheduler
+scheduler
#head
+associatedTasks
Tasking::Task
«struct» Tasking::TaskImpl
«struct» Tasking::InputImpl
Tasking::Channel
Tasking:: PeriodicScheduleTrigger
+parent
-impl
+m_task
-m_inputs
+m_channel
+inputs
-impl
+activ+triggers
Tasking::InputArray
Tasking::Event
«struct» Tasking:: PeriodicScheduleImpl
+parent
+periodicSchedule
-impl
-impl
+parent
+clock
-impl
+clock
«struct» Tasking::InputArrayImpl
Tasking::Input
Tasking::Clock
«struct» Tasking::EventImpl
Tasking:: PeriodicSchedule
+inputs
+queueHead
+queueTail

Example

Example

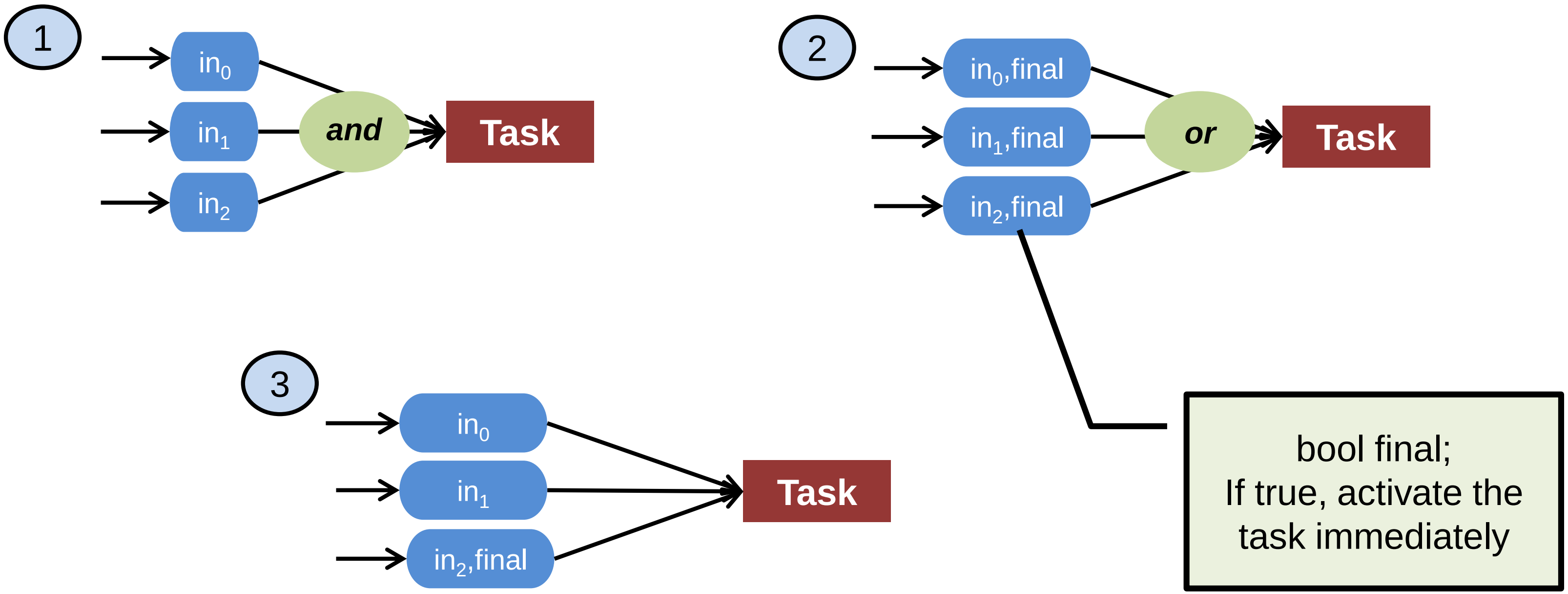

Input Channel
in$_1$
ioChannel:
1. read from the standard input
2. If input = “end”:
1. Exit
3. else: append the new input with the last printed line
4. print the new line on the standard output
5. if there is no new input within 5 sec after the last print:
1. remove the latest input from the line
2. print the new line
3. if there is no new input within 3 sec after the last print:
1. goto 5.1.
4. else: goto 1.
6. else: goto 1.
τ_1
in$_2$
τ_2
Terminal
hammadeh@dlr->
Output Channel

Example


InputChannel
in_1
τ_1
Terminal
hammadeh@dlr->
OutputChannel
τ_2
in_2
FIFO

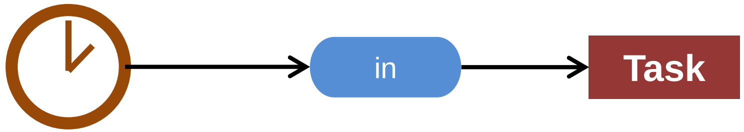
Activation model

Activation model

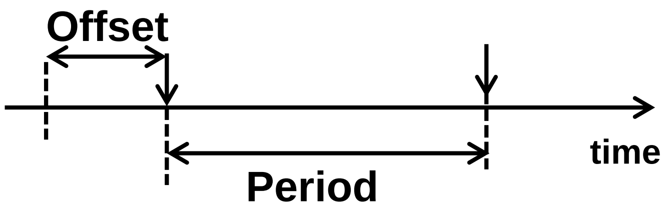


Activation model – Time driven

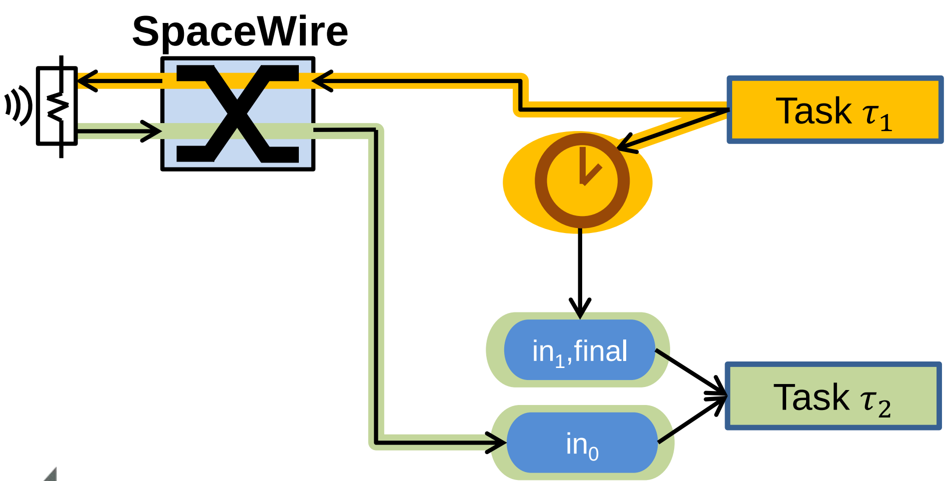
class Event : **public** Channel

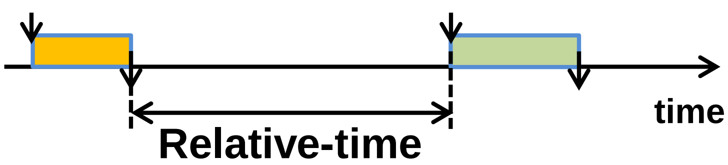


Period



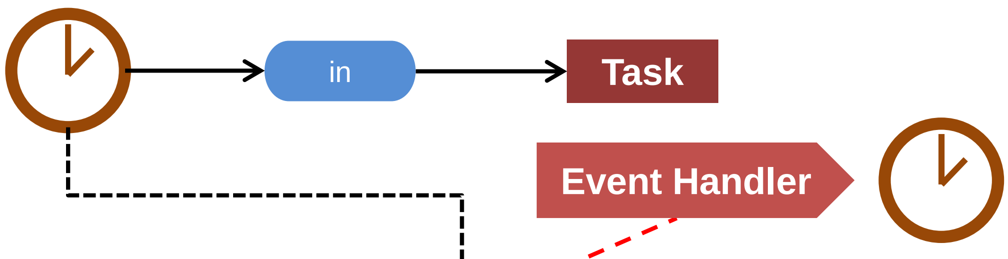
Relative-time





Tasking::Event

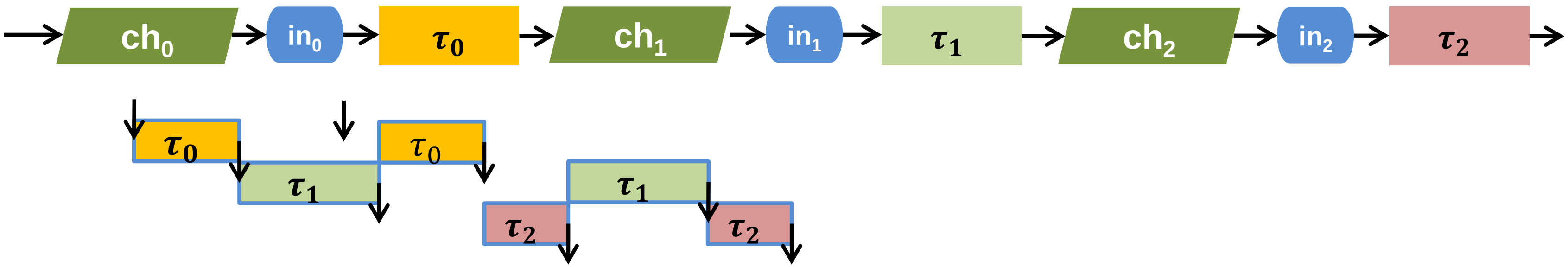
```
class Event : public Channel
```



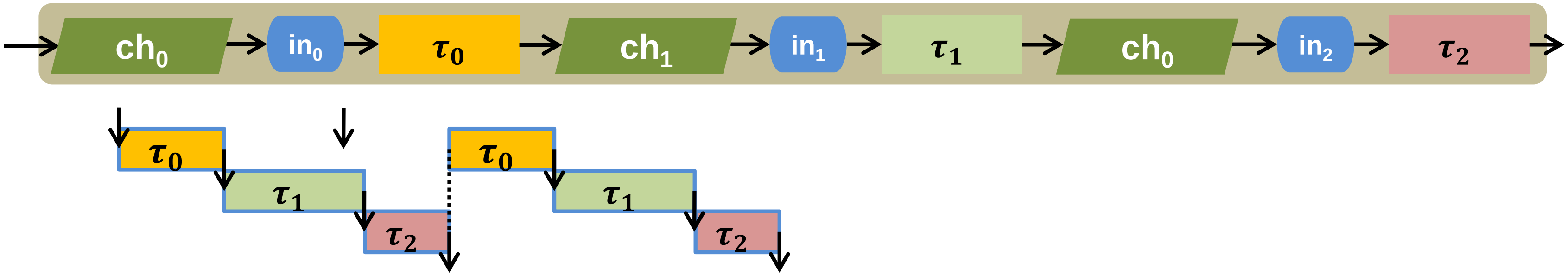

```
if (parent.shallFire())
  {
     parent.onFire();
     static_cast<UnprotectedChannelAccess&>(parent).push();
  }
```

Call semantics

1 Asynchronous (Default)

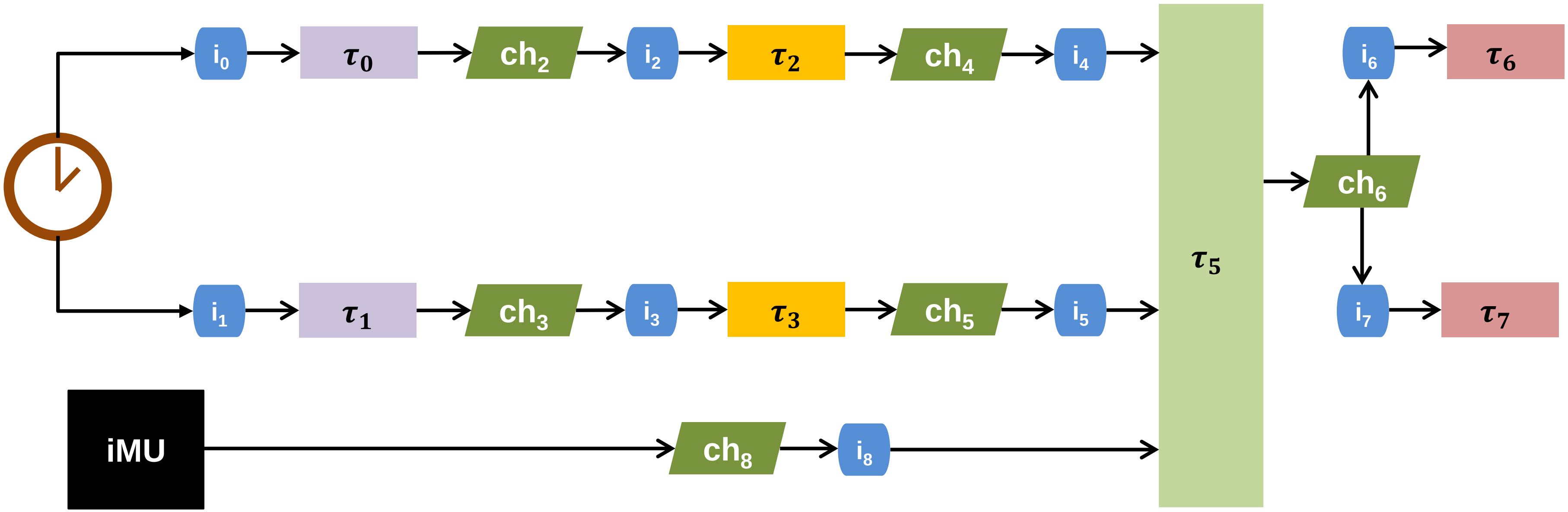


2 Synchronous (Tasking::Group)



Task model

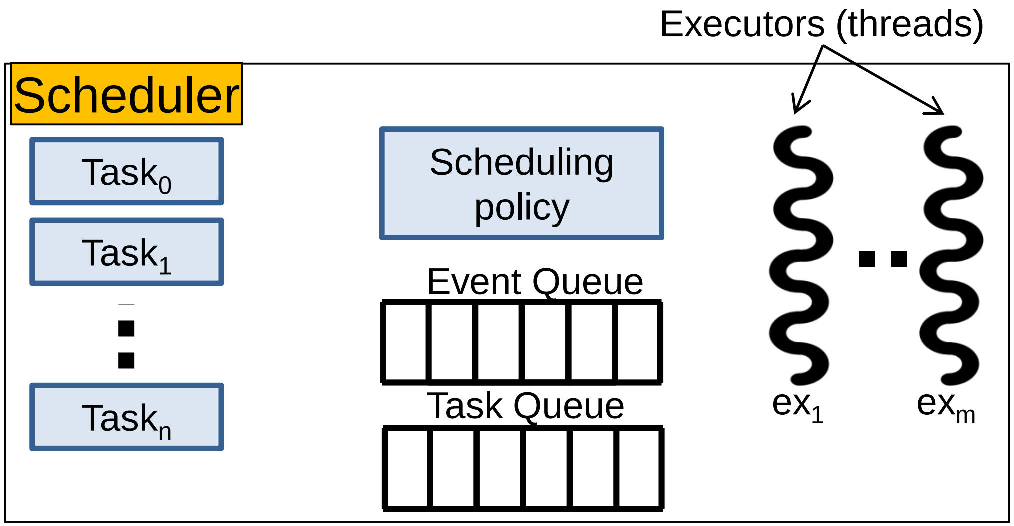
Directed Acyclic Graph (DAG)



Optical navigation system (ATON)

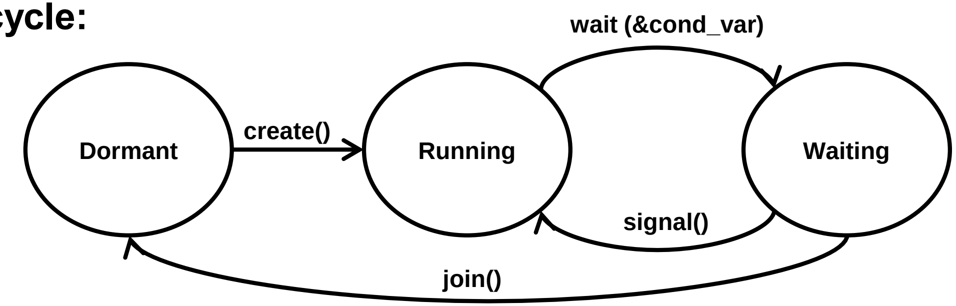
Execution model

Execution model

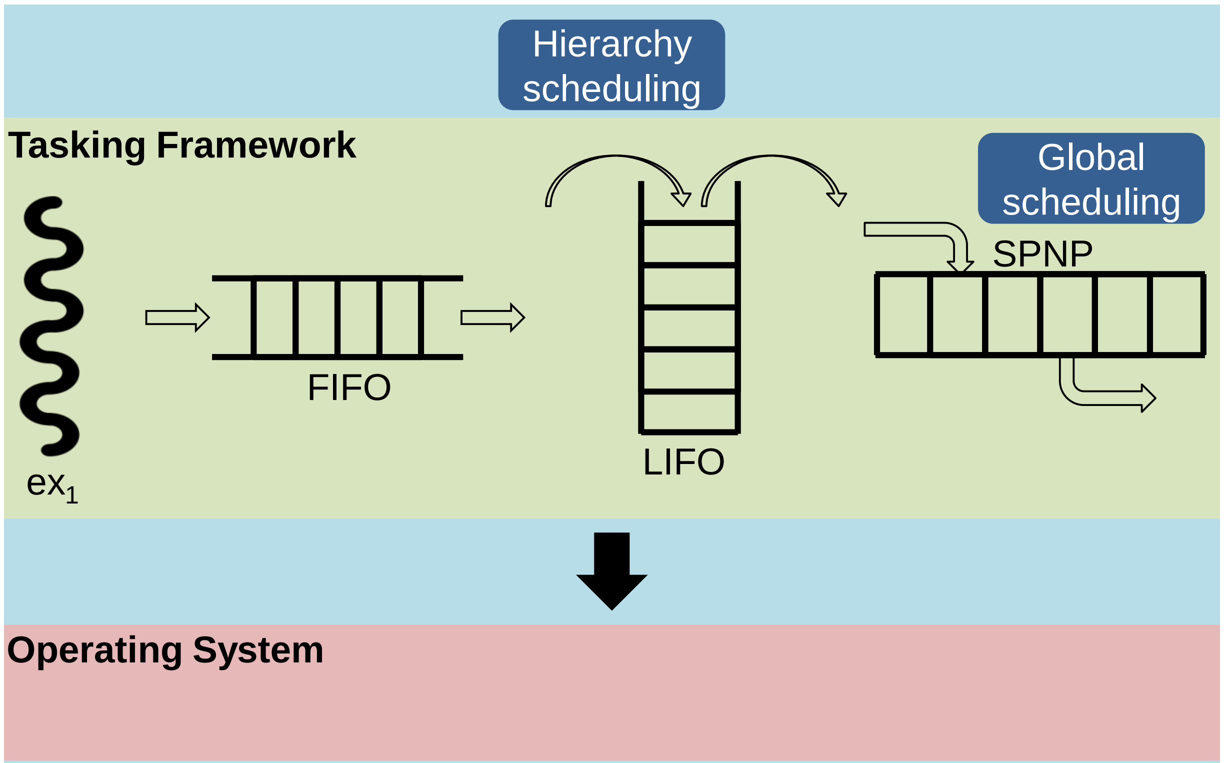


- Work-conserving scheduling
- The load is balanced on the available executors

Thread's life cycle:

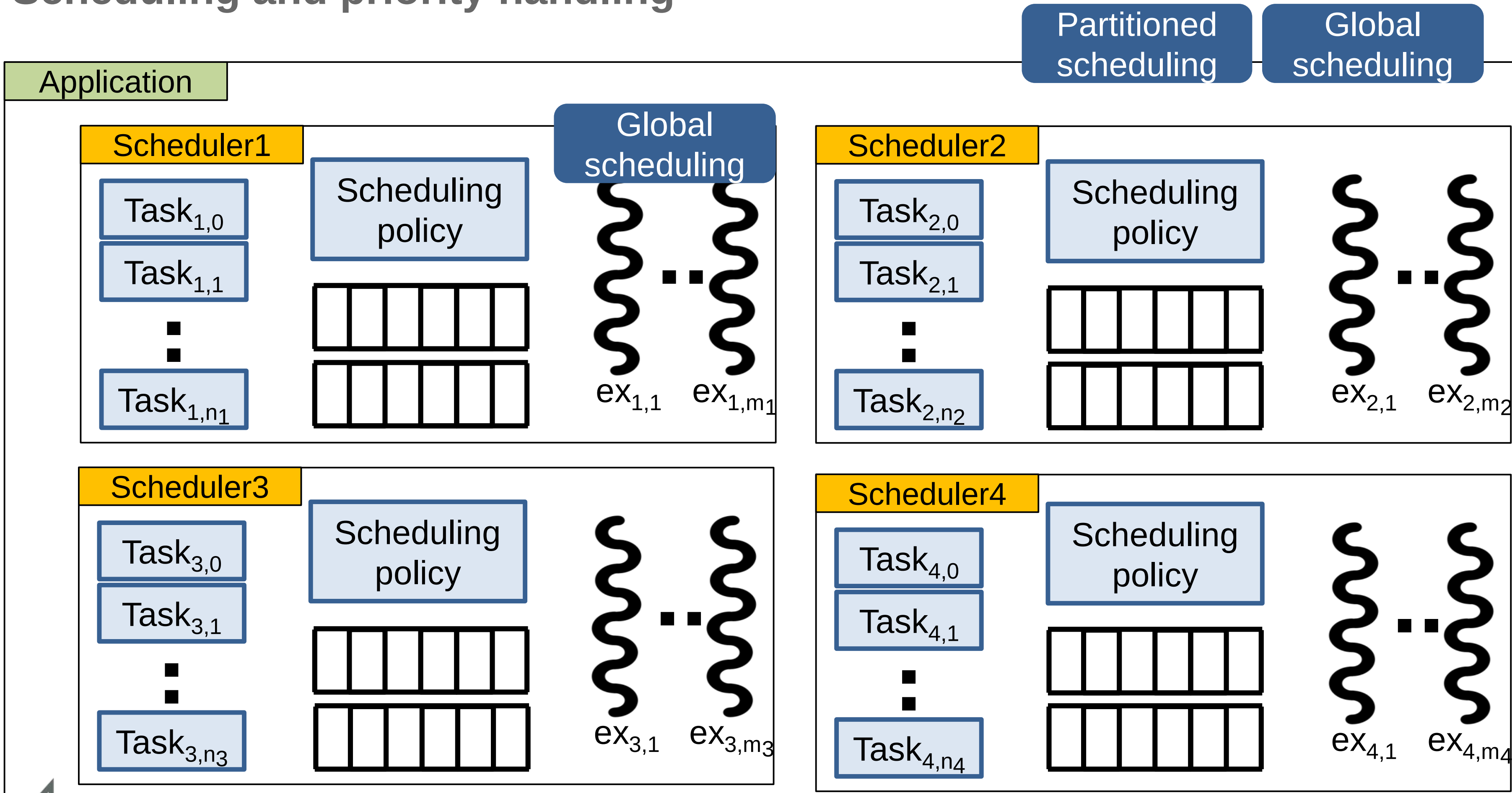


Execution model – Scheduling



- Work-conserving scheduling
- The load is balanced on the available executors
- Global scheduling in tasking framework
- Hierarchy scheduling

Scheduling and priority handling

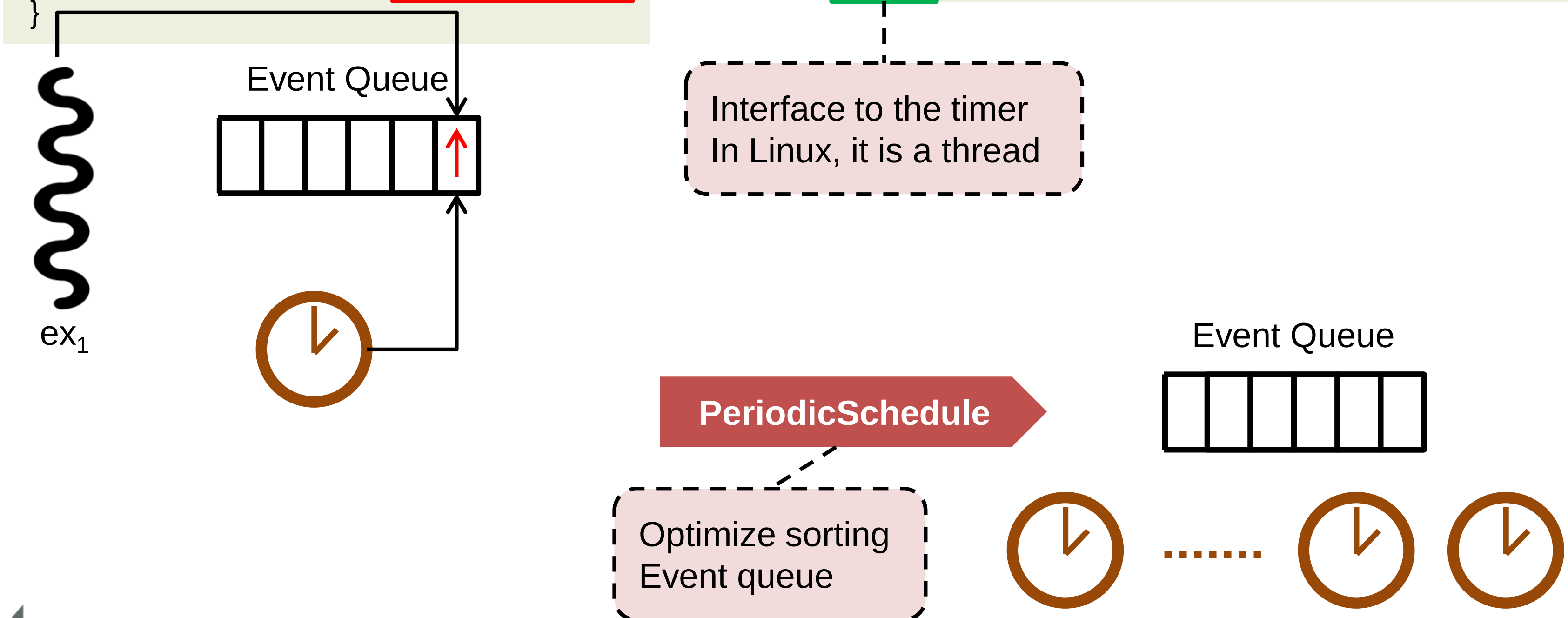


Handling Events

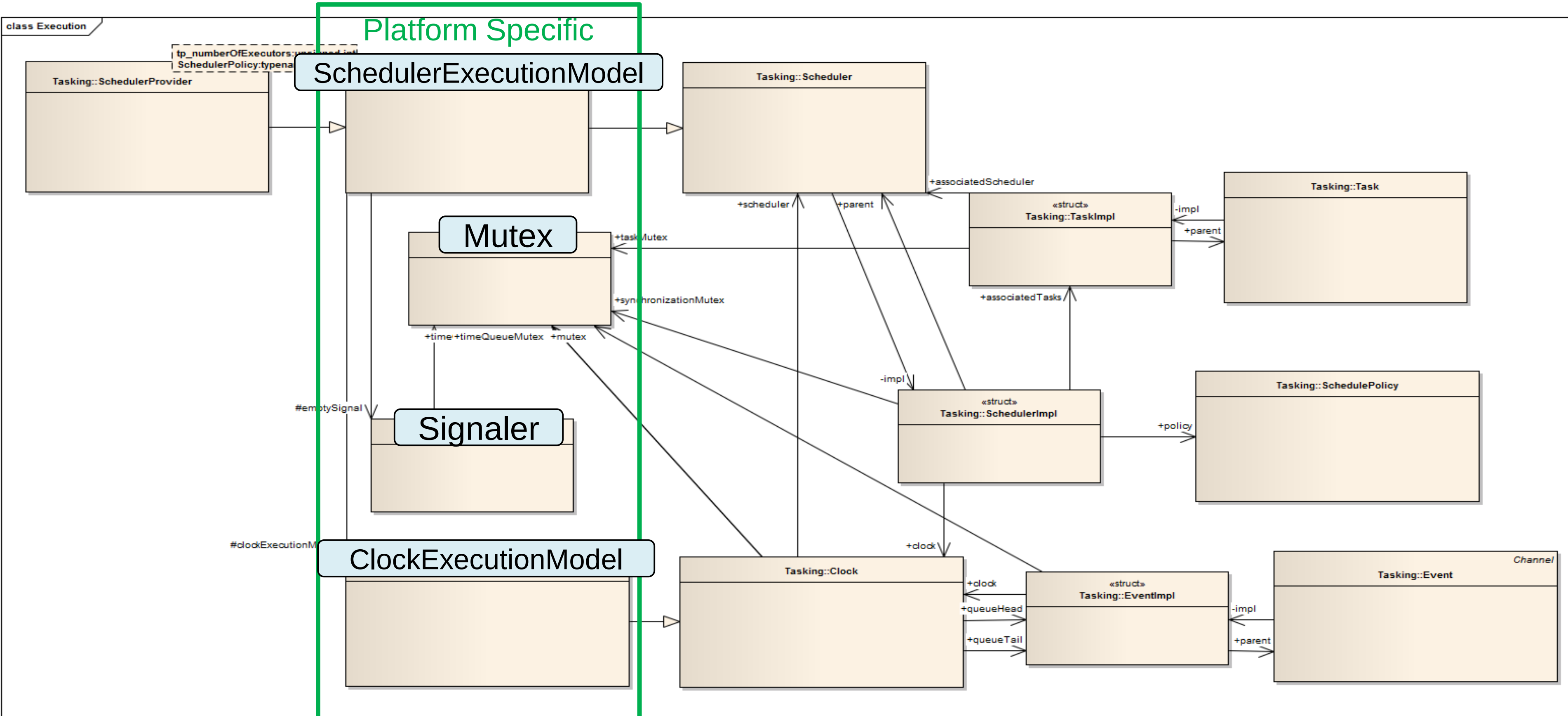
```
if (data->schedulerImpl->clock.isPending())
{
    data->schedulerImpl->handleEvents();
}
```

Periodic

```
clock.startAt(*this, (nextActivation_ms + period_ms));
```



Execution Platform



Unit Test

Unit tests

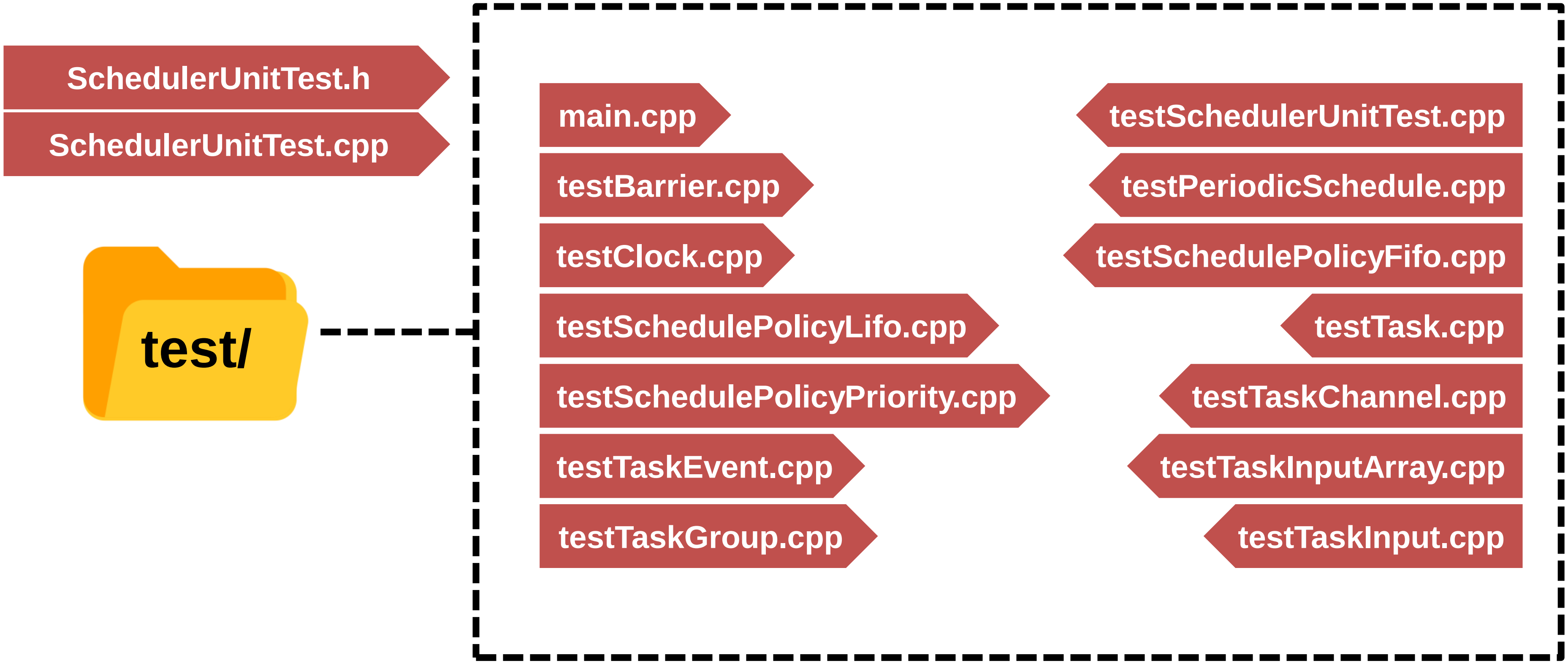
SchedulerUnitTest.h

SchedulerUnitTest.cpp

test/

```cpp
void
Tasking::SchedulerUnitTest::schedule(Tasking::Time timeSpan)
{
   // Step forward in time
   unitTestclock.step(timeSpan);
   // Doing execution
   SchedulerImpl& implementation = getImpl();
   do
   {
      implementation.handleEvents();
      for (TaskImpl* task = implementation.policy.nextTask(); (task != nullptr);
          task = implementation.policy.nextTask())
      {
         implementation.execute(*task);
      }
   } while (implementation.clock.isPending());
}
```

Unit tests


SchedulerUnitTest.h
SchedulerUnitTest.cpp
test/
main.cpp
testBarrier.cpp
testClock.cpp
testSchedulePolicyLifo.cpp
testSchedulePolicyPriority.cpp
testTaskEvent.cpp
testTaskGroup.cpp
testSchedulerUnitTest.cpp
testPeriodicSchedule.cpp
testSchedulePolicyFifo.cpp
testTask.cpp
testTaskChannel.cpp
testTaskInputArray.cpp
testTaskInput.cpp

Unit tests

SchedulerUnitTest.h

SchedulerUnitTest.cpp

test/

main.cpp

```
#include <gtest/gtest.h>

int
main(int argc, char **argv)
{
   ::testing::InitGoogleTest(&argc, argv);

   return RUN_ALL_TESTS();
}
```

testSchedulerUnitTest.cpp

testPeriodicSchedule.cpp

estSchedulePolicyFifo.cpp

testTask.cpp

testTaskChannel.cpp

testTaskInputArray.cpp

testTaskInput.cpp

Documentation

Documentation

doc/Tasking_framework.docx

In-line comments

Doxygen

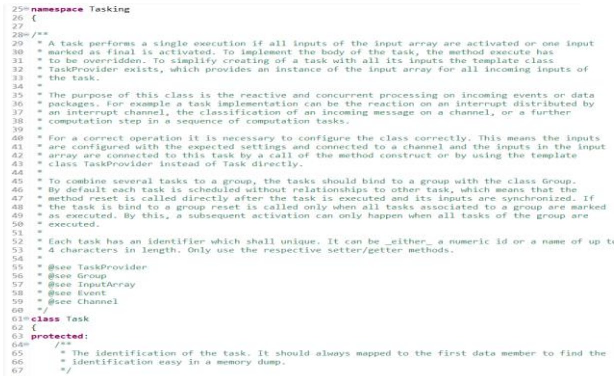
Deutsches Zentrum für Luft- und Raumfahrt
German Aerospace Center

Tasking Framework – Documentation

SC-SRV-OSS

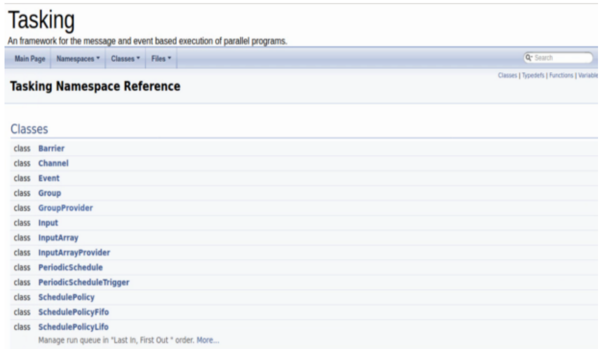
Date: 27.08.2019

Author: Zain Alabedin Haj Hammadeh



```
namespace Tasking
{

/**
 * A task performs a single execution if all inputs of the input array are activated or one input
 * marked as final is activated. To implement the body of the task, the method execute has
 * to be overridden. To simplify creating of a task with all its inputs the template class
 * TaskProvider exists, which provides an instance of the input array for all incoming inputs of
 * the task.
 *
 * The purpose of this class is the reactive and concurrent processing on incoming events or data
 * packages. For example a task implementation can be the reaction on an interrupt distributed by
 * an interrupt channel, the classification of an incoming message on a channel, or a further
 * computation step in a sequence of computation tasks.
 *
 * For a correct operation it is necessary to configure the class correctly. This means the inputs
 * are configured with the expected settings and connected to a channel and the inputs in the input
 * array are connected to this task by a call of the method construct or by using the template
 * class TaskProvider instead of Task directly.
 *
 * To combine several tasks to a group, the tasks should bind to a group with the class Group.
 * By default each task is scheduled without relationships to other task, which means that the
 * method reset is called directly after the task is executed and its inputs are synchronized. If
 * the task is bind to a group reset is called only when all tasks associated to a group are marked
 * as executed. By this, a subsequent activation can only happen when all tasks of the group are
 * executed.
 *
 * Each task has an identifier which shall unique. It can be _either_ a numeric id or a name of up to
 * 4 characters in length. Only use the respective setter/getter methods.
 *
 * @see TaskProvider
 * @see Group
 * @see InputArray
 * @see Event
 * @see Channel
 */
class Task
{
protected:
    /**
     * The identification of the task. It should always mapped to the first data member to find the
     * identification easy in a memory dump.
     */
```



Tasking

An framework for the message and event based execution of parallel programs.

Main Page | Namespaces | Classes | Files | Search

Classes | Typedefs | Functions | Variable

Tasking Namespace Reference

Classes

class **Barrier**
class **Channel**
class **Event**
class **Group**
class **GroupProvider**
class **Input**
class **InputArray**
class **InputArrayProvider**
class **PeriodicSchedule**
class **PeriodicScheduleTrigger**
class **SchedulePolicy**
class **SchedulePolicyFifo**
class **SchedulePolicyLifo**
Manage run queue in "Last In, First Out " order. More...

Live Q&A session

Sunday 20.09.2020
at 11:00 – 12:00 am EDT
at 05:00-06:00 pm Berlin

11:00am

Tutorial 4: Tasking Framework: An open-source software development library for on-board software systems

11:00am - 12:00pm, Sep 20

Tutorials

* This tutorial is **a live Q&A session** with **pre-recorded videos made available before the event**.

Tasking Framework is a C++ software development library and an event-driven multithreading execution platform. It is developed by the Institute for Software Technology, German Aerospace Center (DLR). Tasking Framework is dedicated to improve the reusability in developing embedded software systems and to reconcile the embedded software with model-driven software development. It can be used to develop, but not dedicated for, critical as well as non-critical embedded software on single-core as well as parallel architectures. Tasking Framework gives software developers the ability to implement their applications as task graphs with arbitrary activation patterns (periodic, aperiodic and sporadic) using a set of abstract classes with virtual methods. It is compatible with the POSIX-based operating systems, mainly Linux and RTEMS. The Tasking Framework was successfully used in, for instance, the attitude orbit

Read more