A Self Balancing Robot

Or

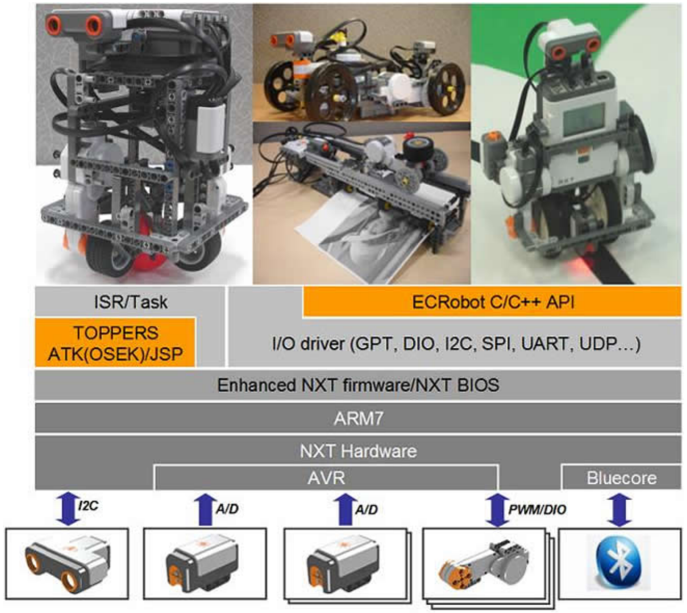
A lesson on OSEK/VDX

Agenda

- Introduction.
- The robot & the model.
- Installing and the that jazz.
- Deconstructing the OS.
- A series of interesting topics.
- Concluding remarks.

Introduction

- What is a (Reactive) Real Time System?
- Com**ponen**ts of RTS development:
  - Known hardware platform,
  - OS that allows ”proper” scheduling.

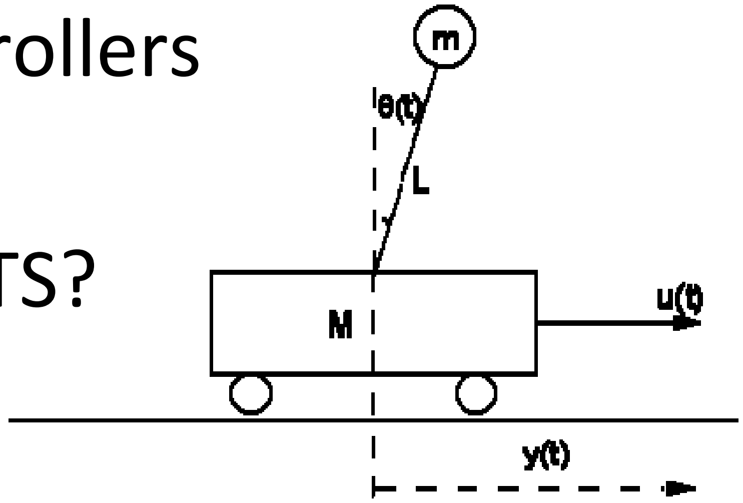


The Robot

- This is old news….
- NXTway-GS
- Building instructions, software etc is trivially found online at lejos-osek website.

The Model

- Any real autonomous robot has a model.
  - This is the central point of control theory.
  - Models for a 2-D moving robot is?
  - A self balancing robot is…?
- Models are used for controllers
  - PID, Fuzzy Logic, etc.
- How is this related to a RTS?



Installing and all the Jazz

- The intention was…
- Building, compiling etc. is easy.
- YouTube…

Deconstructing the OS (1)

- Why is OSEK/VDX interesting?
  - It is build for a purpose by the German automotive industry.
  - Used in the industry.
  - Designed for small embedded devices.
  - Depending on HW hard real-time is expected.
    - Airbags, ABS, ESP.
  - Distributed via CAN-bus etc.

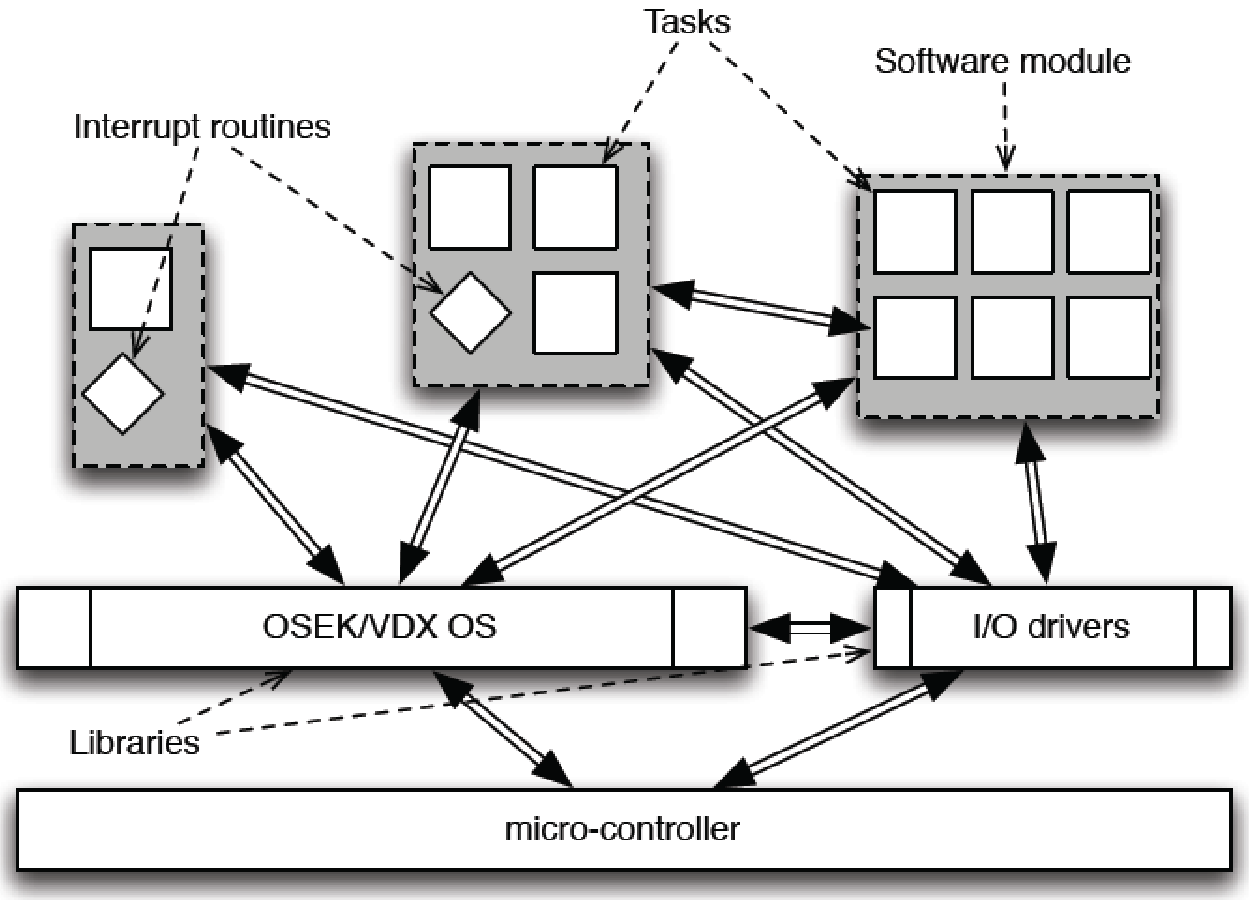
Deconstructing the OS (2)

- What is OSEK/VDX?
  - "Öffene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug / Vehicle Distributed Executive”
  - Open Systems and their Interfaces for the Electronics in Motor Vehicles.
  - This means that it is a specification more than an implementation.

Deconstructing the OS (3)

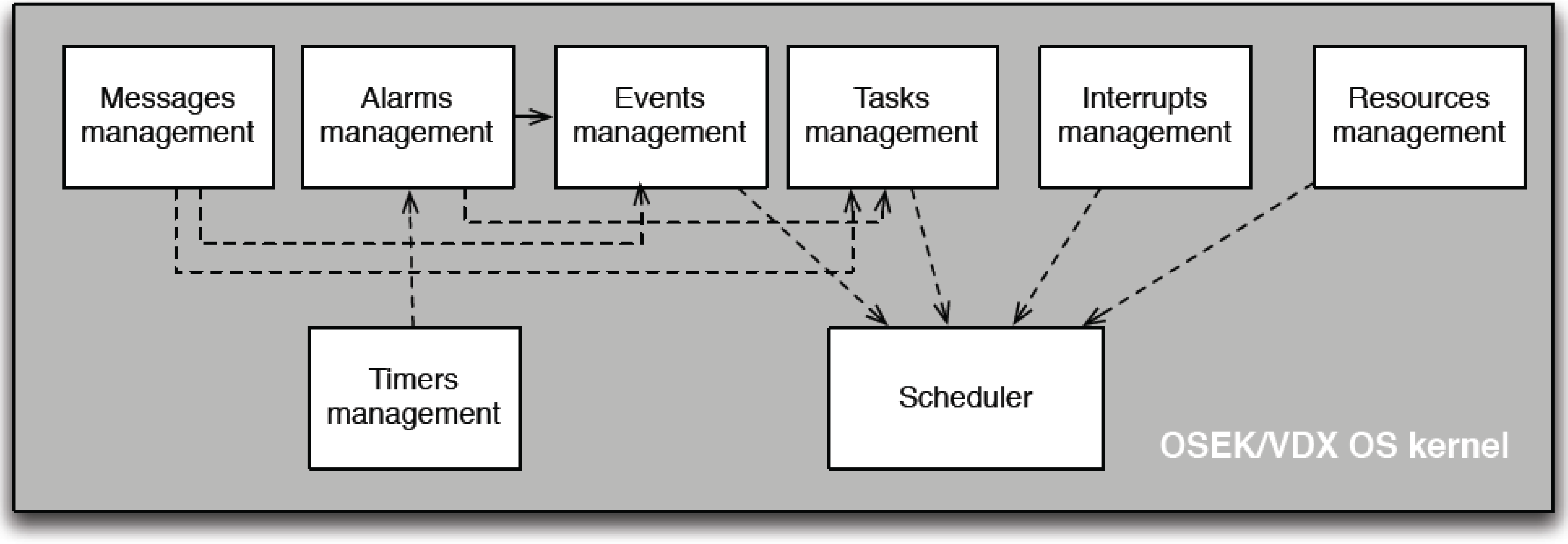
- Great stuff, point me to the documentation…
- OSEKnxt has certain limitations.
  - The TOPPERS/ATK implementation is not complete.
  - Based on:
    - OSEK OS Version 2.2.1
    - OSEK OIL Verion 2.5
- Where does this leave us?

This is regular OS knowledge



- OSEKnxt does not support real ISR’s!

Submit to the scheduler



No ’new’ constructs

- static configuration (offline) : The application architecture is completely known.
  - Greatly simplify the design and the writing of the kernel.
- allow to embed only the functions of the OS that are really used.
- Predictable behavior. Fit requirements of real-time applications.

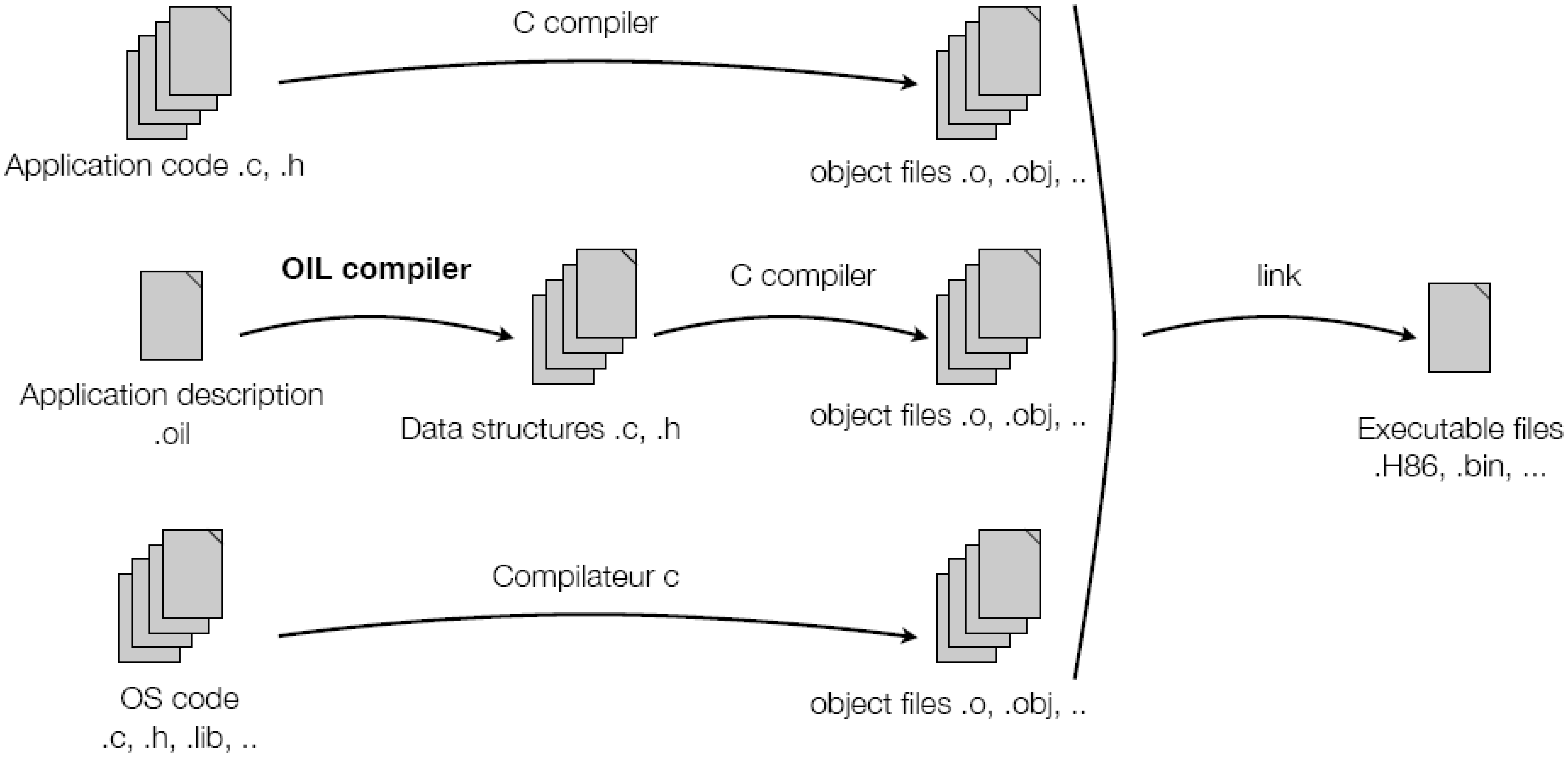
OSEK OS and OIL

- Objects of an OSEK application are all defined when the application is designed.
  - Objects are static. i.e: there are no creation/deletion of tasks, resources, etc dynamically during the execution of the application.
- Data structures are used to store the properties of the objects and are defined statically when the application is built.
- A language has been defined (and standardized) to define the attributes of the objects in a simple way:
  - OIL: OSEK Implementation Language

A tad more on OIL

- The OIL syntax is a simple one: based on objects (tasks, resources, …) with a value for each attribute.
  - Some attributes have sub-attributes.
- Starting from the description of the application (text file), data structures are automatically generated:
  - fast;
  - less error prone;
  - Independant of the OSEK vendor (the data structures are not included in the standard);
  - easy to update.

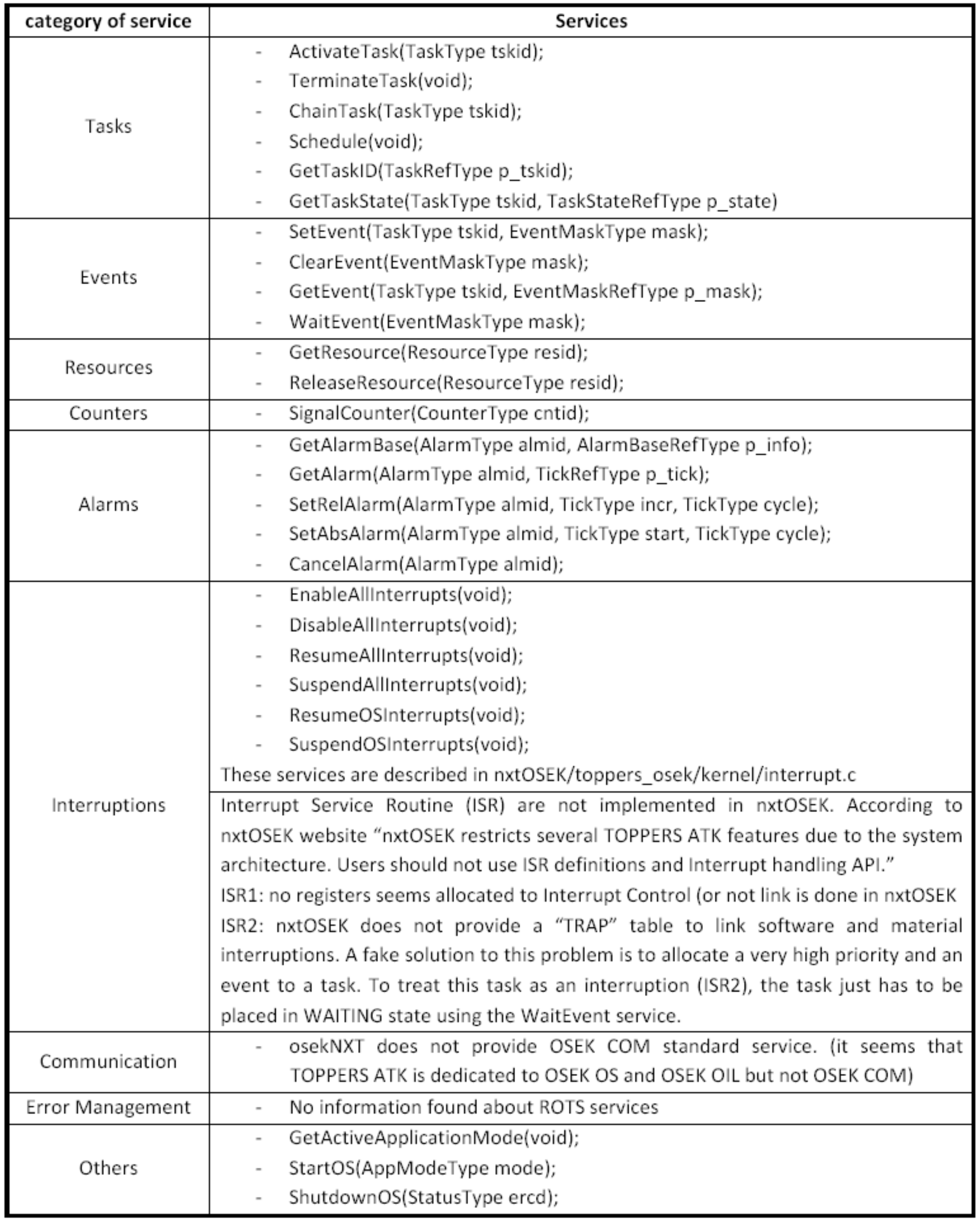
Adding it all together



Phew, thats it?

- No, now the real techical issues are coming.
- OSEKnxt < OSEK/VXD spec.
- Our focus of today is OSEKnxt

OSEKnxt Implements



category of service	Services
Tasks	- ActivateTask(TaskType tskid); - TerminateTask(void); - ChainTask(TaskType tskid); - Schedule(void); - GetTaskID(TaskRefType p_tskid); - GetTaskState(TaskType tskid, TaskStateRefType p_state)
Events	- SetEvent(TaskType tskid, EventMaskType mask); - ClearEvent(EventMaskType mask); - GetEvent(TaskType tskid, EventMaskRefType p_mask); - WaitEvent(EventMaskType mask);
Resources	- GetResource(ResourceType resid); - ReleaseResource(ResourceType resid);
Counters	- SignalCounter(CounterType cntid);
Alarms	- GetAlarmBase(AlarmType almid, AlarmBaseRefType p_info); - GetAlarm(AlarmType almid, TickRefType p_tick); - SetRelAlarm(AlarmType almid, TickType incr, TickType cycle); - SetAbsAlarm(AlarmType almid, TickType start, TickType cycle); - CancelAlarm(AlarmType almid);
Interruptions	- EnableAllInterrupts(void); - DisableAllInterrupts(void); - ResumeAllInterrupts(void); - SuspendAllInterrupts(void); - ResumeOSInterrupts(void); - SuspendOSInterrupts(void); These services are described in nxtOSEK/toppers_osek/kernel/interrupt.c
	Interrupt Service Routine (ISR) are not implemented in nxtOSEK. According to nxtOSEK website "nxtOSEK restricts several TOPPERS ATK features due to the system architecture. Users should not use ISR definitions and Interrupt handling API." ISR1: no registers seems allocated to Interrupt Control (or not link is done in nxtOSEK ISR2: nxtOSEK does not provide a "TRAP" table to link software and material interruptions. A fake solution to this problem is to allocate a very high priority and an event to a task. To treat this task as an interruption (ISR2), the task just has to be placed in WAITING state using the WaitEvent service.
Communication	- osekNXT does not provide OSEK COM standard service. (it seems that TOPPERS ATK is dedicated to OSEK OS and OSEK OIL but not OSEK COM)
Error Management	- No information found about ROTS services
Others	- GetActiveApplicationMode(void); - StartOS(AppModeType mode); - ShutdownOS(StatusType ercd);

Tasks

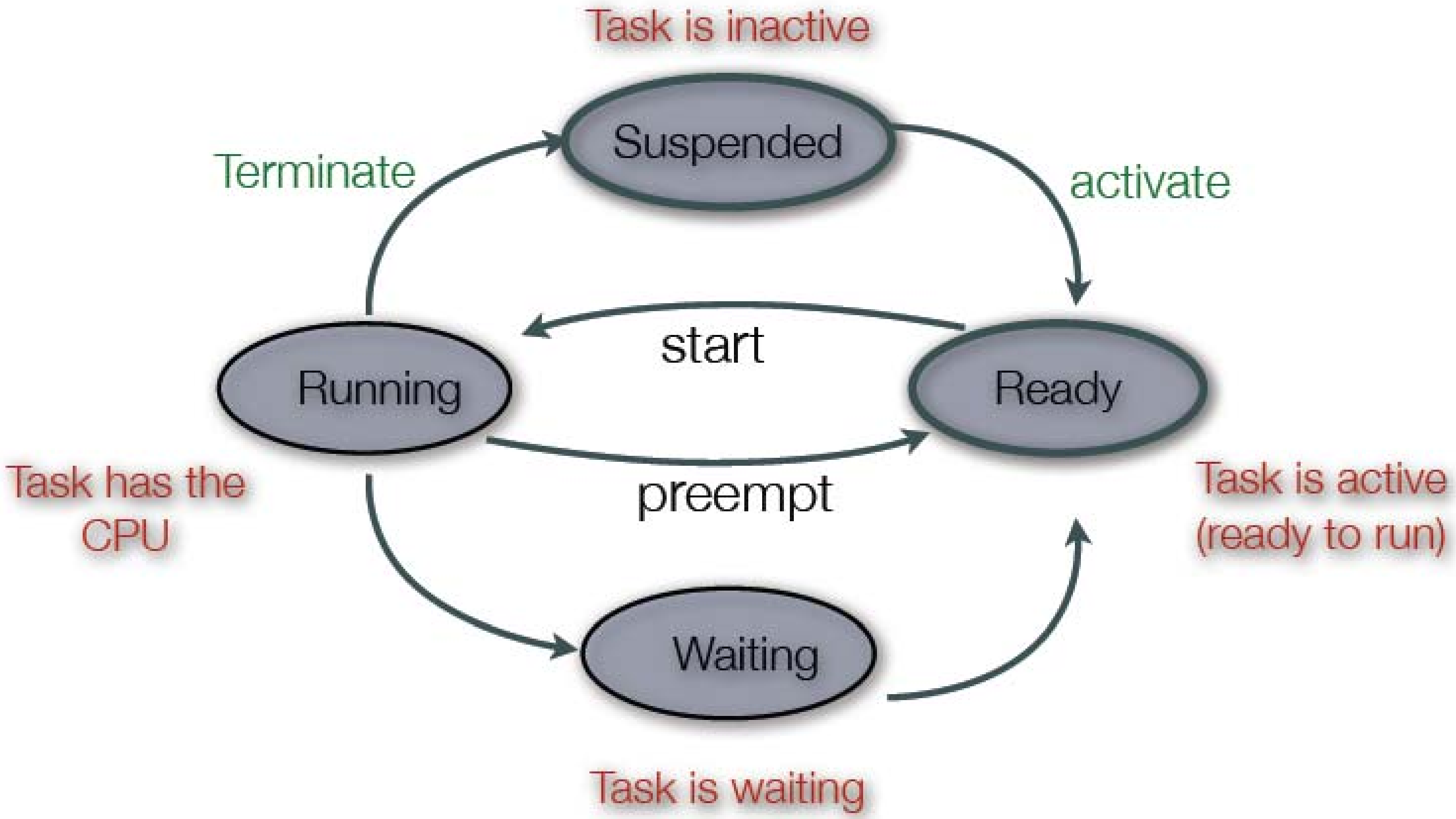
- Task has a state model:
  - Basic
  - Extended
- Tasks are finite series of C instructions.

Task keyword

```
TASK(myTask){
  //Task instructions
  …
  TerminateTask();
}
```

Call to service that
Termintas a task

Skipping to Extended tasks

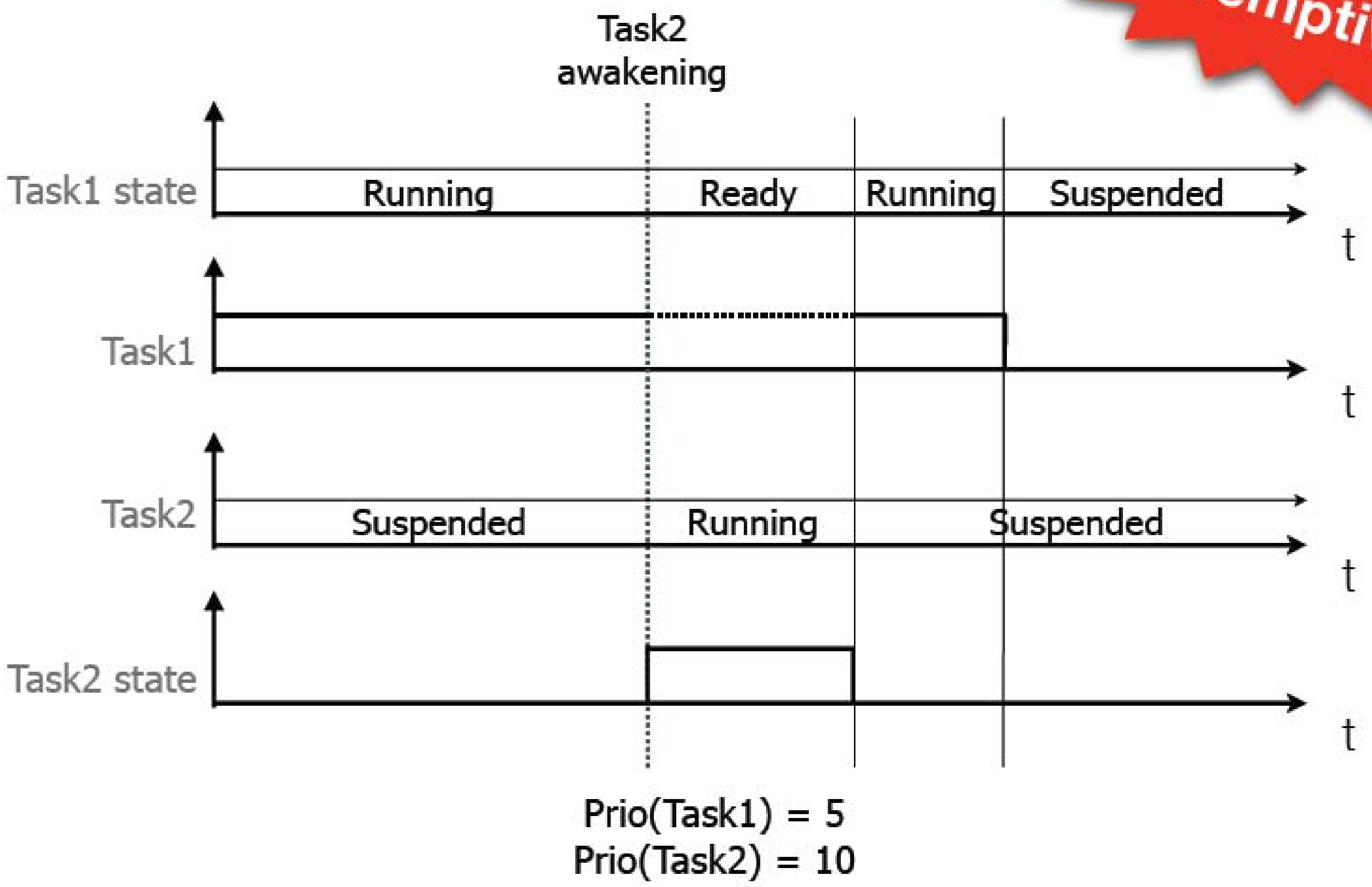


Tasks are flexible entities

- Programs build of tasks may be:
  - Full preemptive
    - It is the most reactive model because any task may be preempted. The highest priority Task is sure to get the CPU as soon as it is activated.
  - Non-preemptive
    - It is the most predictive model because a task which get the CPU will never be preempted. Scheduling is a straightforward and the OS memory footprint may be smaller.
  - Mixed setting
    - For instance, a very short task (in execution time) may be configured as non-preempteble because the context switch is longer than its execution.
- Decided at design time in the OIL file

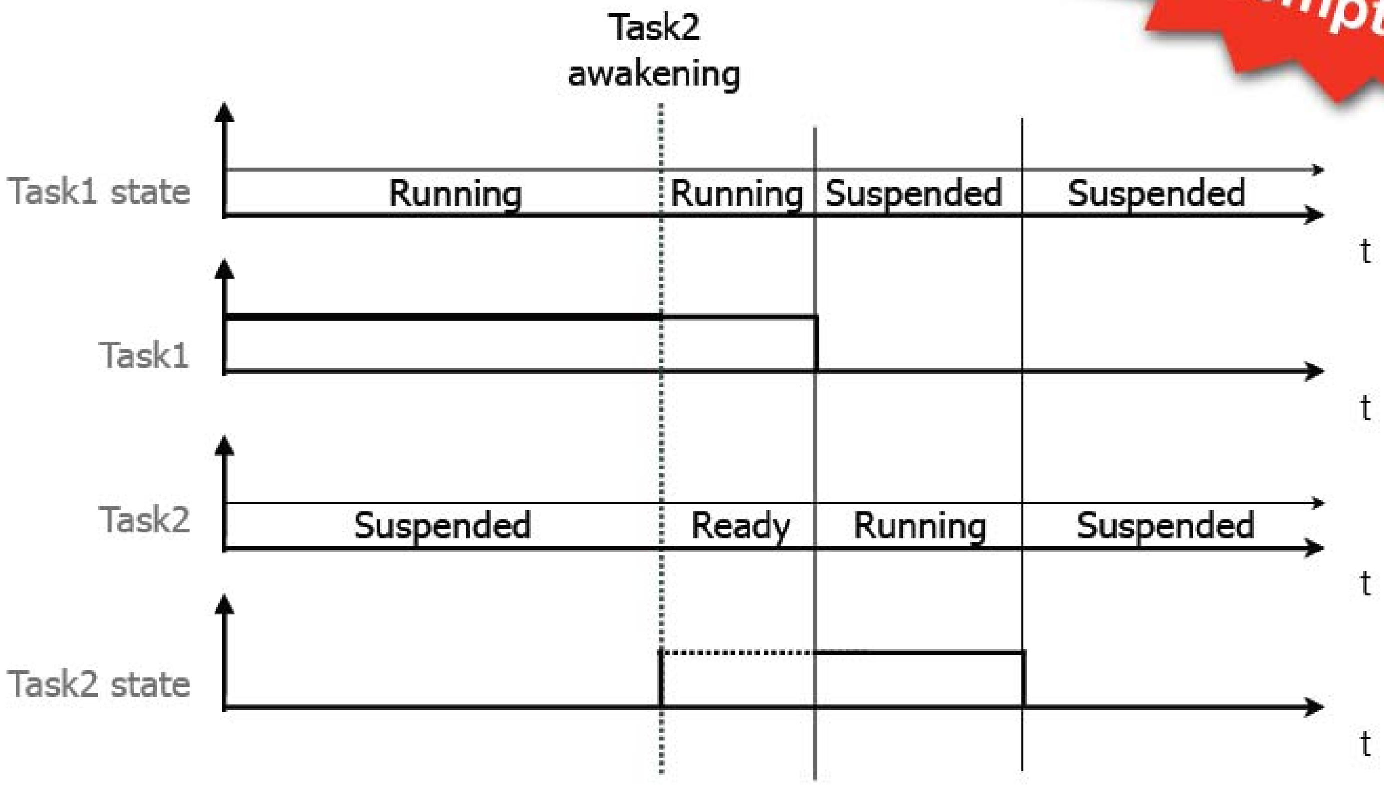
- Example: 2 tasks (Task1 and Task2).
  At start, Task1 runs. Then Task2 is activated

Full Preemptive



- Example: 2 tasks (Task1 and Task2).
  At start, Task1 runs. Then Task2 is activated

Full Non-preemptive

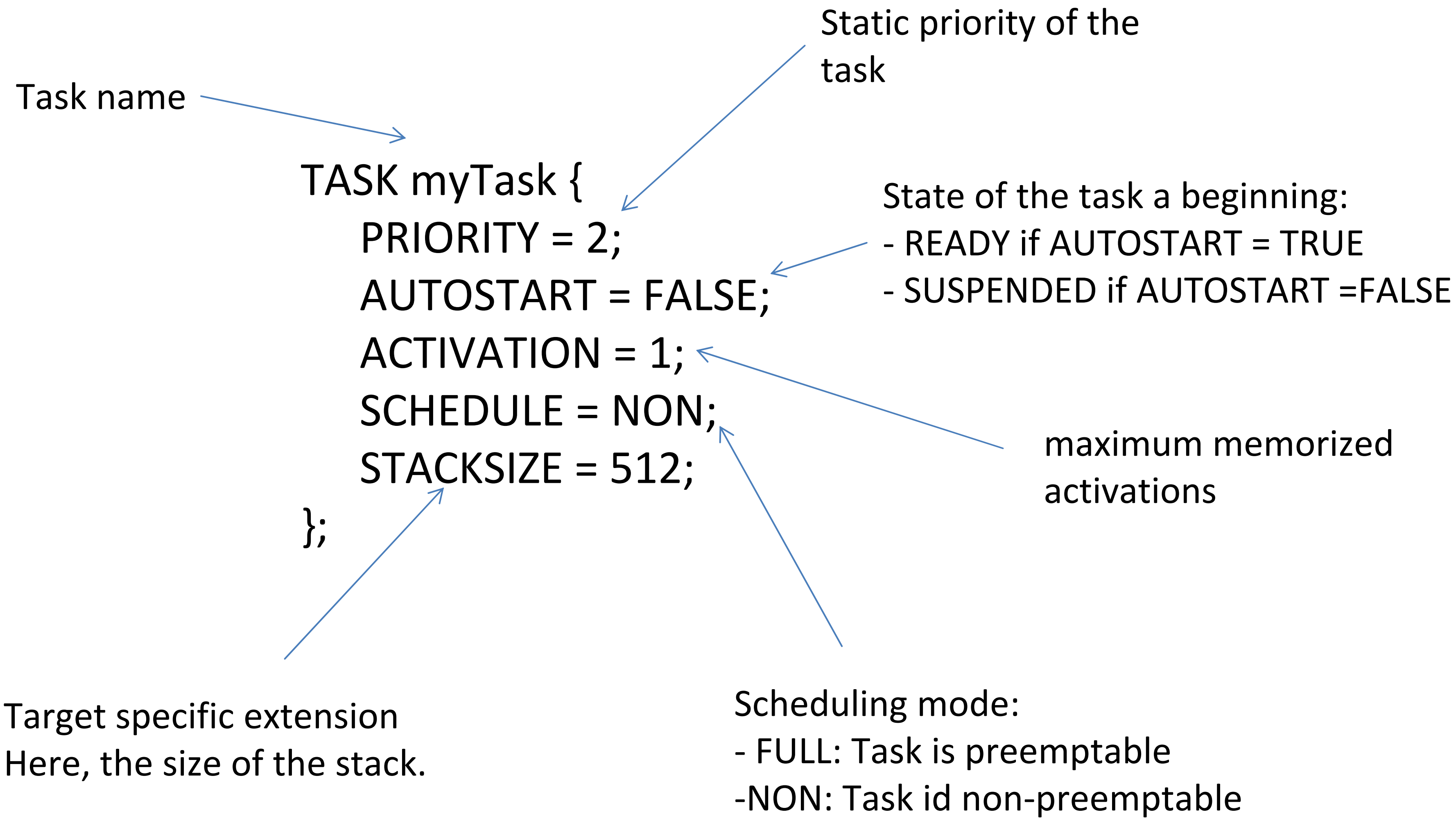


Prio(Task1) = 5
Prio(Task2) = 10

Task Services

- Manipulate tasks in various ways.
  - Terminate task
    - OS terminates tasks. All tasks must terminate!
  - Activate task
    - Puts a new task in running state
  - Chain tasks
    - Replaces terminate task by explicitly calling a new task

OIL Description of task



```
TASK myTask {
    PRIORITY = 2;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = NON;
    STACKSIZE = 512;
};
```

Task name

Static priority of the task

State of the task a beginning:
- READY if AUTOSTART = TRUE
- SUSPENDED if AUTOSTART =FALSE

maximum memorized activations

Target specific extension
Here, the size of the stack.

Scheduling mode:
- FULL: Task is preemptable
-NON: Task id non-preemptable

Events, Alarms, Counters and Hooks

All of these gives us:

- Synchronization.
- Discrete counter to drive the system.
- Time based events.
- Running code at various interesting points.

This should also be (well) known by you.

Events

- An event is like a flag that is raised to signal something just happened.
- An event is private: It is a property of an Extended Task. Only the owning task may wait for the event.
- It is a N producers / 1 consumer model
  - Any task (extended or basic) may invoke the service which set an event.
  - One task (an only one) may get the event (ie invoke the service which wait for an event.
- The implementation defines a max number of events pr. taks.

Event masks

- Any extended task has:
  - A bit vector coding events set
  - A bit vector coding events it waits for
- To implement this feature, an event corresponds to a binary mask: 0x01,0x02, 0x04.
  - Fun eh?
  - Bit operations are error prone!
  - OS to the rescue.

Event Services (1)

- **SetEvent**
  - StatusType SetEvent(TaskType <TaskID>, EventMaskType <Mask>);
  - This service is not blocking and may be called from any task.
- **ClearEvent**
  - StatusType ClearEvent(EventMaskType <Mask>);
  - non-blocking service.
  - May be called by the owning task (only)

Event Services (2)

- **GetEvent**
  - StatusType GetEvent(TaskType <TaskId>, EventMaskRefType event);
  - The event mask of the task <TaskId> is copied to the variable event (A pointer to an EventMaskType is passed to the service);
  - Non-blocking service, my be called from a task.

Event Services (3)

- **WaitEvent**
  - StatusType WaitEvent(EventMaskType <EventID>);
  - Put the calling task in the WAITING state until one of the events is set.
  - May be called by the event owning (extended) task only.
  - Blocking service.

OIL Description of Events

```
EVENT ev1 {
  MASK = AUTO;
};
```

Definition of the mask. It is:
- AUTO, the actual value is computed by the OIL compiler.
- A litteral value which is the binary mask.

```
EVENT ev2 {
  MASK = 0x4;
};
```

List of the event the task uses.
The task is the owner of these events

```
TASK myTask {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = NON;
  STACKSIZE = 512;
  EVENT = ev1;
  EVENT = ev2;
};
```

If an event is used in more than one task, only the name is shared: **An event is private.**

myTask is automatically an **Extended task because it** uses at least one event.

Code with events

```
TASK(Task1)
{
…
SetEvent(Task2, EV1);
…
TerminateTask();
}
```

Set EV1 which is owned by Task2

Wait for 2 events simultaneously
The task will be waked up when at least one of the 2 events will be set

```
TASK(Task2)
{
EventMaskType event_got;
…
WaitEvent(EV1 | EV2);
GetEvent(Task2, &event_got);
if (event_got & EV1) {
//manage EV1
}
if (event_got & EV2) {
//manage EV2
}
…
TerminateTask();
```

Useful to know what event has been set

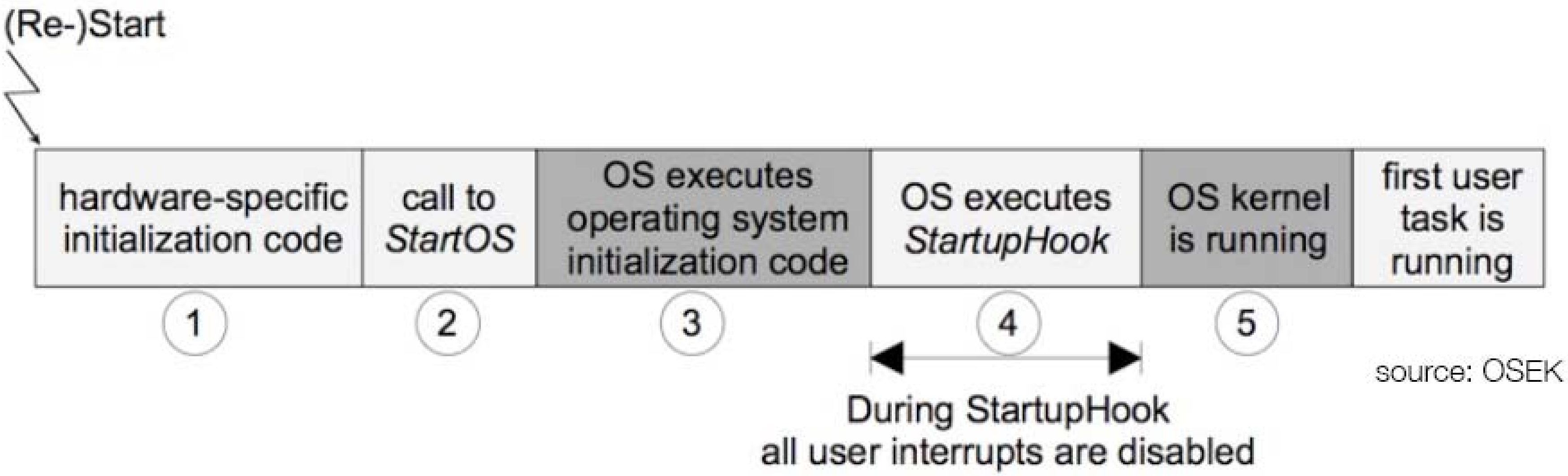
```
TASK(Task3)
{
…
SetEvent(Task2, EV2);
…
TerminateTask();
}
```

Hook Routines

Features

- OSEK proposes dedicated routines (or functions) to allow the user to «hook » an action at important stages in system calls.
- “hook routines” are/have:
  - called by the operating system,
  - a priority greater than all tasks,
  - a standardized interface,
  - able to call a subset of the operating system services.

Startup/shutdown hook


(Re-)Start
hardware-specific initialization code
call to *StartOS*
OS executes operating system initialization code
OS executes *StartupHook*
OS kernel is running
first user task is running
1
2
3
4
5
During StartupHook all user interrupts are disabled

source: OSEK 2.2.3

- ShutdownHook
  - This routine is called when ShutdownOS() is called and should be used for fatal error handling.

Errorhook/pre-post task hooks

ErrorHook:

- This routine is called when a system call does not return E_OK, that is if an error occurs during a system call(recursive calls).

PreTaskHook and PostTaskHook:

- PreTaskHook is called just before a task goes from READY state to RUNNING state.
- PostTaskHook is called just before a task goes from RUNNING state to
- READY or SUSPENDED state.

OIL and C hook example

- The hooks which are used must be declared in the OS object in the mplementation part of the OIL file

```
OS config {
  STATUS = EXTENDED;
  ERRORHOOK = TRUE;
  PRETASKHOOK = TRUE;
} ;
```

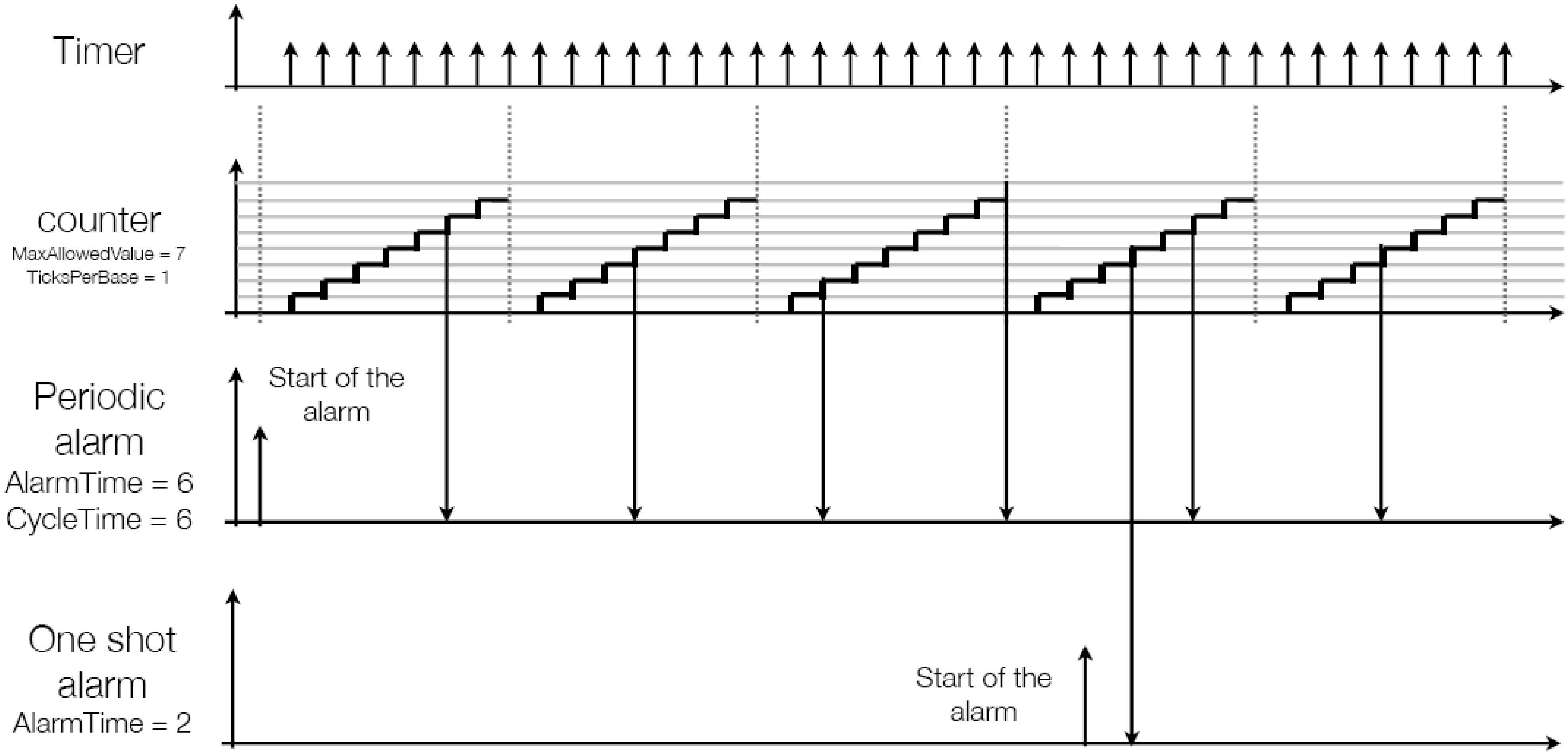
In the C source:

```
void ErrorHook(StatusType error)
{}
void PreTaskHook(void)
{
TaskType id;
GetTaskID(&id);
printf("Before %d\n",id);
}
```

Counters and Alarms

- Goal: perform an “action” after a number of “ticks” from an hardware device:
  - Typical case: periodic activation of a task with a hardware timer.
- The “action” may be:
  - signalization of an event.
  - activation of a task.
  - function call (a callback since it is a user function). The function is executed on the context of the running task.
- The hardware device may be:
  - a timer.
  - any periodic interrupt source (for instance an interrupt triggered by the low position of a piston of a motor. The frequency is not a constant in this case.

Alarms



Timer
counter
MaxAllowedValue = 7
TicksPerBase = 1
Periodic
alarm
AlarmTime = 6
CycleTime = 6
Start of the
alarm
One shot
alarm
AlarmTime = 2
Start of the
alarm

Alarm services

SetAbsAlarm

```
StatusType SetAbsAlarm (
    AlarmType <AlarmID>,
     TickType <start>,
     TickType <cycle>)
```

- AlarmID is the id of the alarm to start.
- start is the absolute date at which the alarm expire
- cycle is the relative date (counted from the start date) at which the alarm expire again. If 0, it is a one shot alarm.

Alarm Services

SetRelAlarm

```
StatusType SetRelAlarm (
    AlarmType <AlarmID>,
    TickType <increment>,
    TickType <cycle>)
```

- AlarmID is the id of the alarm to start.
- increment is the relative date at which the alarm expire
- cycle is the relative date (counted from the start date) at which the alarm expire again. If 0, it is a one shot alarm.

Alarm Services

CancelAlarm

StatusType CancelAlarm (AlarmType <AlarmID>)

- AlarmID is the id of the alarm to stop.

GetAlarm

- Get the remaining ticks before the alarm expires.

StatusType GetAlarm (larmType <AlarmID>,TickRefType <tick>)

- AlarmID is the id of the alarm to get.
- tick is a pointer to a TickType where GetAlarm store the remaining ticks before the alarm expire.

Shared Resources

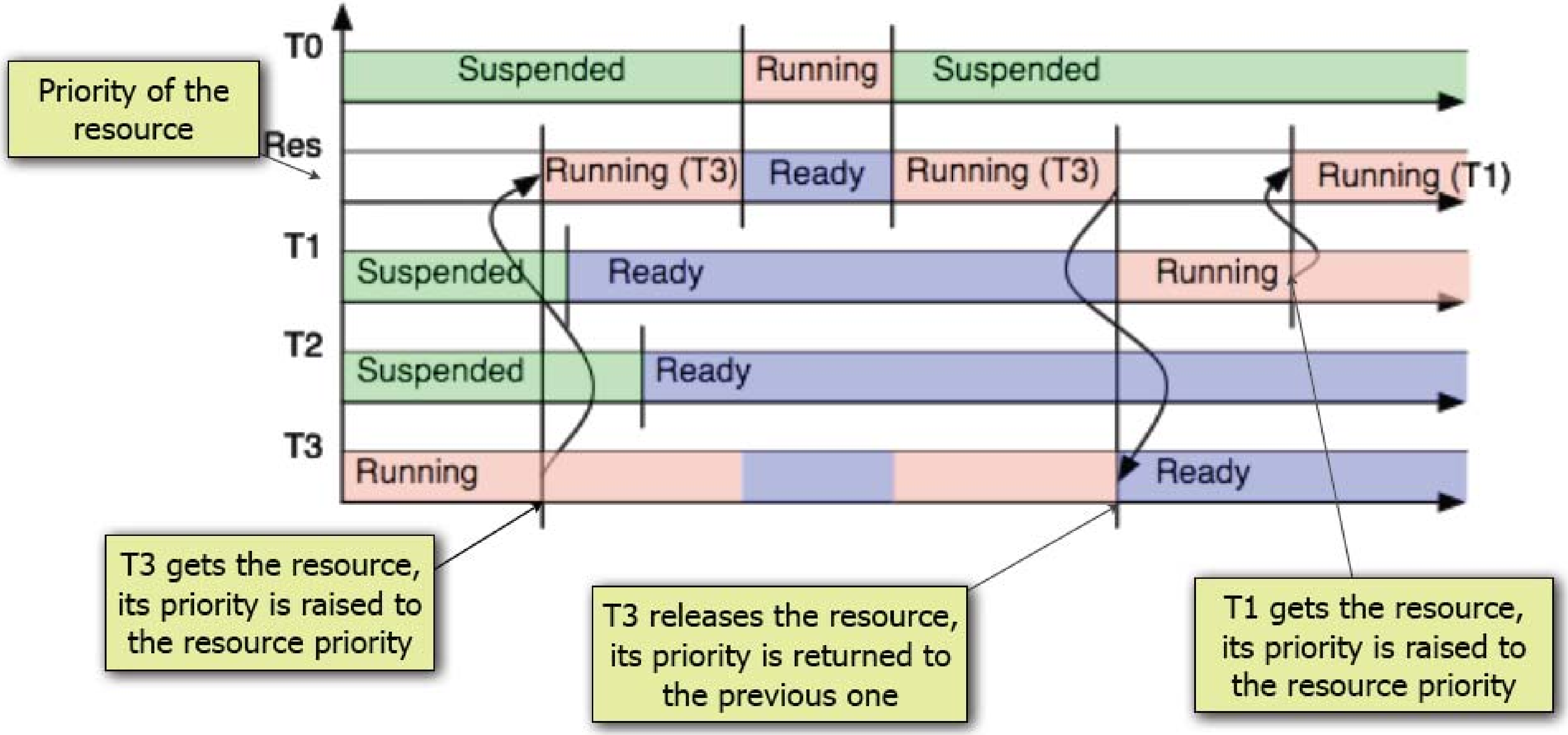
- All the usual problems,
- Along with the solutions.
- A ressource is defined in the OIL file.
- A task kan get or release a resource

Ressource Scheduling

To take resources into account in scheduling, a slightly modified PCP (Priority Ceiling Protocol) is used.

- Each resource has a priority such as:
  - The priority is ≥ to max of priorities of tasks which may get the resource;
  - When a task get a resource, its priority is raised to the priority of the resource
  - When a task release the resource, its priority is lowered to the previous one.

Ressource example



Priority of the resource

T3 gets the resource, its priority is raised to the resource priority

T3 releases the resource, its priority is returned to the previous one

T1 gets the resource, its priority is raised to the resource priority

Is there more time?

- If yes, we take a look at the code for balancing robot.

Conlusion

- OSEK/VDX is powerful
  - We have left out everything on conformance classes.
- OSEKnxt is good learning experience.
- Gives all scheduling needs for soft real time.

Further Reading

- http://tiresias.nuxit.net/chuwiki/download/lego/LEGO_and_real_time.pdf
- http://portal.osek-vdx.org/files/pdf/specs/deprecated/os221.pdf
- http://portal.osek-vdx.org/files/pdf/specs/oil25.pdf
- http://lejos-osek.sourceforge.net/
- http://www.embedded-computing.com/pdfs/Metrowerks.Win03.pdf

Credits

- Trampoline teaching slides: diagrams and structure.

Excercises

- Download and install OSEKnxt
- Find StatusType for the services mentioned in the slides.
- Review:
  - Alarmtest
  - Eventtest
  - Resourcetest
- Consider if OSEKnxt is the right way for your project.
- Compile and run the bluetooth tests.