REAL TIME SYSTEMS

1. Real Time Operating systems: an overview

Brief overview of Real-Time Systems

Objective

Hide the particularities of the hardware from the application

=> more or less complex virtual machine

OS Classification :

Generalist (UNIX...)

Real-time extended generalists (Linux, POSIX...)

Original real time

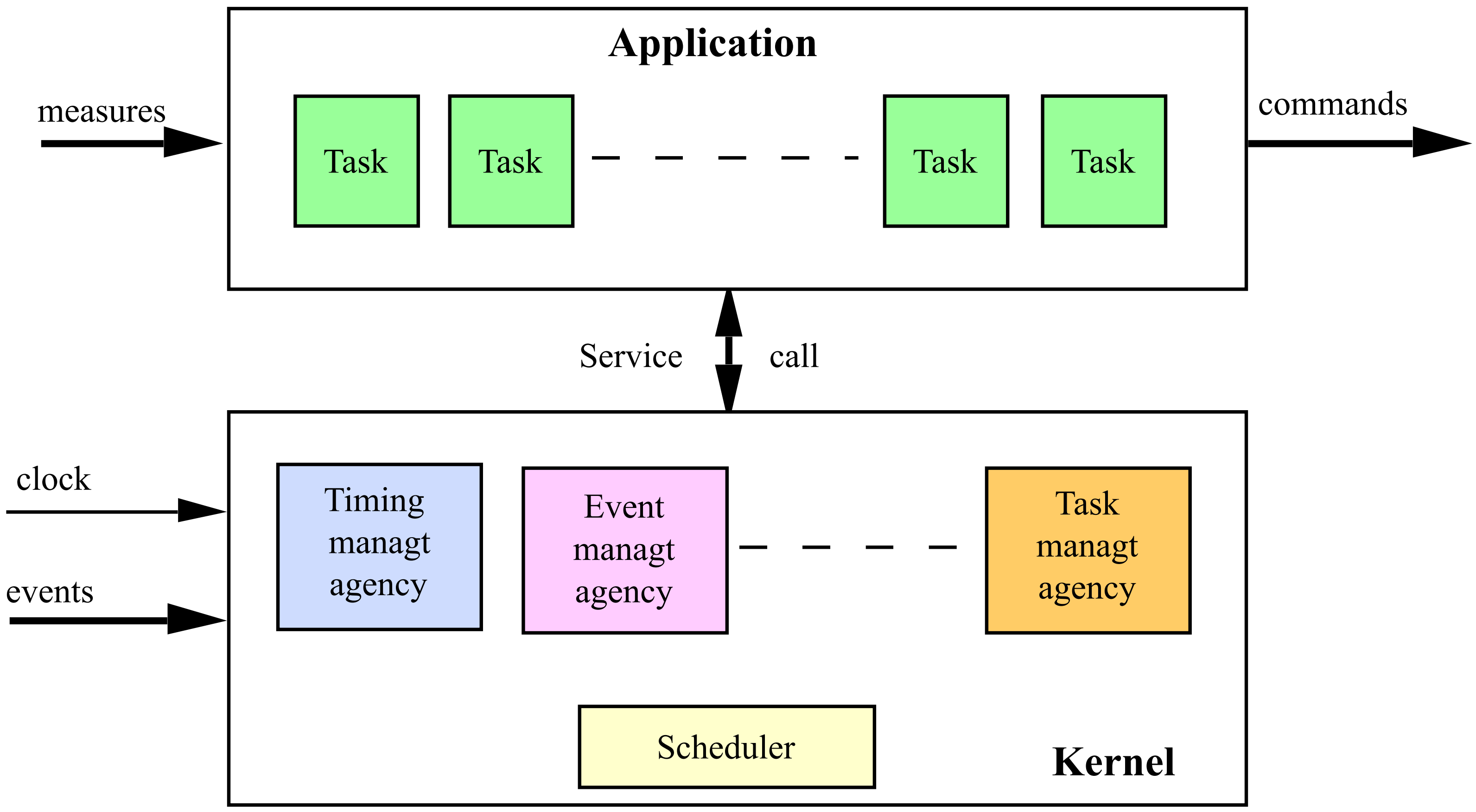
Small kernels for limited embedded applications

=> APEX

=> OSEK

Brief overview of Real-Time Systems

General structure:



Generalities about Real-Time OS

The main characteristics of real-time kernels

- **Conformity to a standard or pseudo-standard** (POSIX, Sceptre project)
- **Compactness** (for embedded applications)
- **Target environment** (microprocessors, architecture, …)
- **Host environment** (OS type)
- **Development tools** (debug, online analysis, …)
- **Real-time functions** (list of all services provided)
- **Characteristics of the scheduler** (scheduling policies)
- **Temporal characteristics :**
  - **interrupt latency:** time during which interrupts are masked and therefore cannot be taken into account (execution of atomic primitives, manipulation of critical structures, ...)
  - **preemptive latency:** the maximum amount of time the kernel can delay the scheduler.
  - **task response time:** time between the occurrence of an interruption and the execution of the woken up task.

Generalities about Real-Time OS

Two main types of real-time OS:

- the original Real-Time OS:
  - Domain-oriented OS (aeronautics, automotive…)
  - General real-time OS (Tornado, QNX, …)

  - allow a fine management of priorities
  - offer fast system primitives, in limited time (management of interrupts, semaphores...)
  - no virtual memory, but locking pages in main memory
  - minimizing overhead (the time taken by the system to run and manage itself)

=> the solution to Hard Real-Time

Generalities about Real-Time OS

Two main types of real-time OS:

=> the classical O.S. (Unix...) extended for real time

Enable concurrent development of real-time and non-real-time applications in a standard and comfortable environment.

- But it took:
  - review the scheduling policies
  - reinforce the notion of preemption
  - define reentrant system primitives
  - define the notion of thread (to facilitate preemption with context saving and then recovery with context restitution)

  => important and complex modifications

  RTAI

  RTLinux

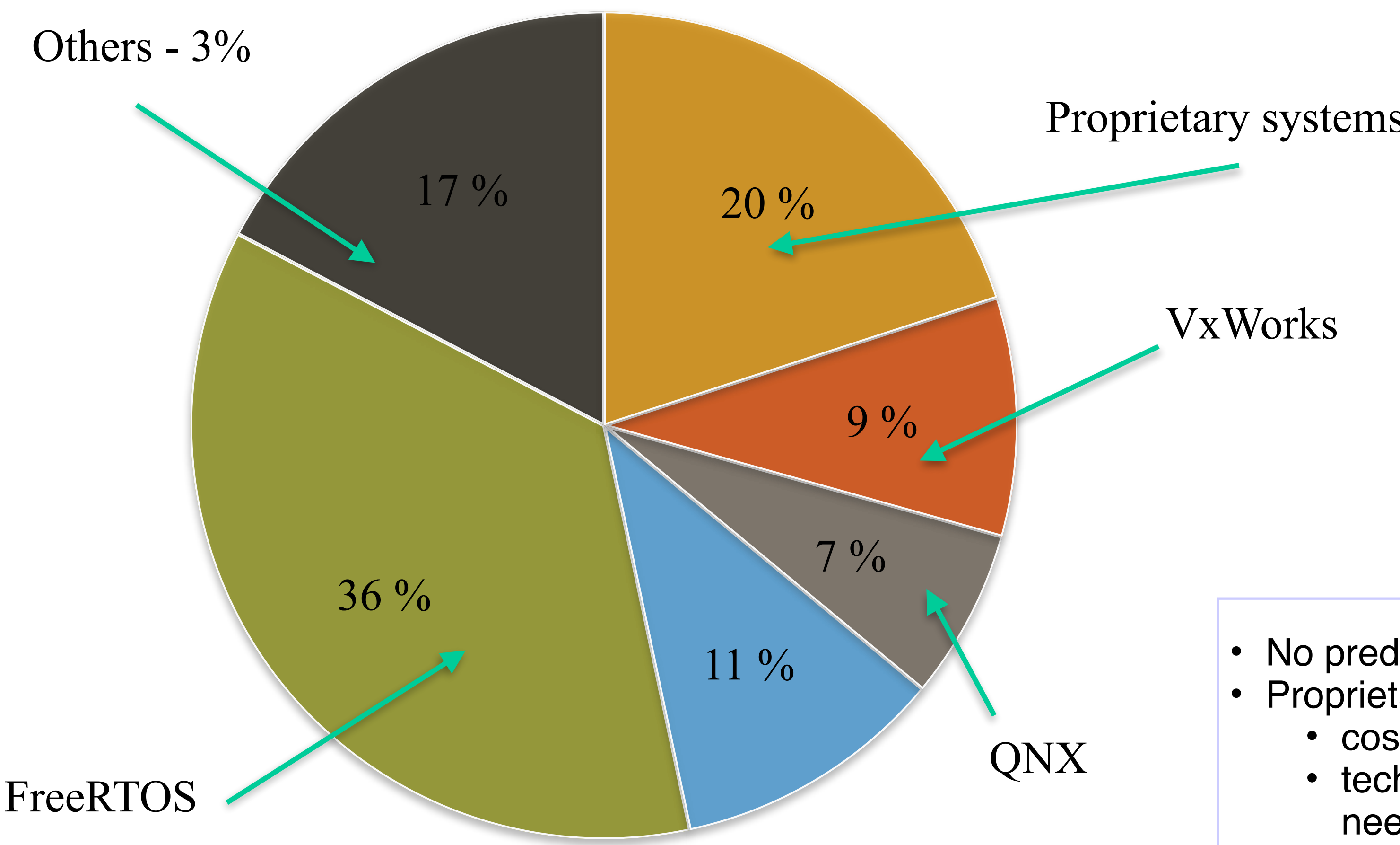
  Windows CE

  => for Soft Real-Time

Generalities about Real-Time OS

Situation of the industrial supply of real-time kernels

Embedded Market Study (USA - 2014)



- No predominance of one system
- Proprietary systems :
  - cost (licence)
  - techniques (adaptation to needs)
  - strategy (control)

2. The Real-Time kernel OSEK/VDX

The OSEK context

Context: the embedded "electronics" in vehicles

- real-time **constraints (hard and soft)**
- high safety
- **minimal hardware support (little RAM, 8- and 16-bit ECUs)**
- distributed architecture **around ≠ networks (CAN, VAN, LIN ...)**
- **cross-cutting functions (interoperability of subsystems)**
- **flexibility of the architecture (addition of functions,** portability **and** reusability **of software functions)**

OSEK / VDX: history

- Proposal of the OSEK group, consisting of:
  - manufacturers (BMW, DaimlerChrysler, Renault, PSA, etc.)
  - equipment manufacturers (Bosch, Siemens, etc.)
  - academics (Univ. Karlsruhe)

- Merger of the OSEK (German) and VDX (GIE PSA-Renault) projects

- Includes the European project MODISTARC (certification process of compliant implementations)

- Work started in 1995

The reference website: http://www.osek-vdx.org

Main OSEK OS services

- Services for tasks
- Synchronization services (events)
- Mutual Exclusion Services
- Services for recurring phenomena (counters and alarms)
- Service for communication (in OSEK/VDX COM)
- Interruption management Services
- System services and Error management

=> All objects are static

Tasks

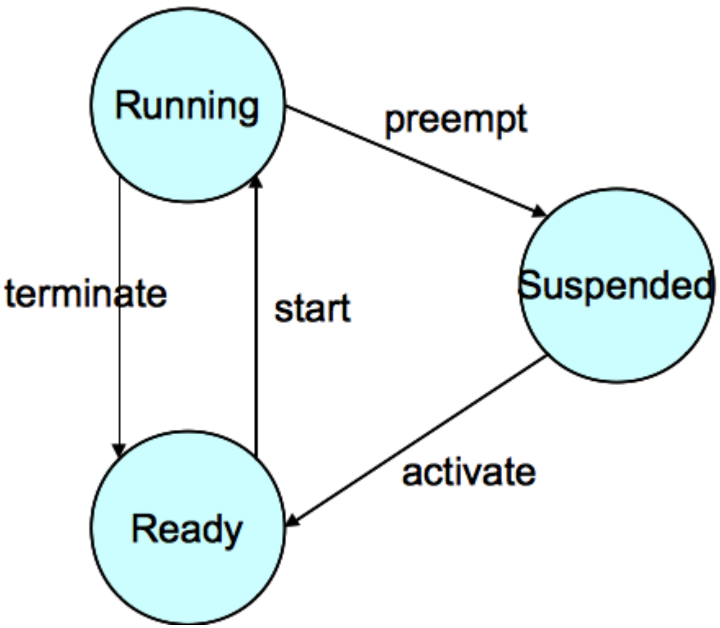
tâches basiques

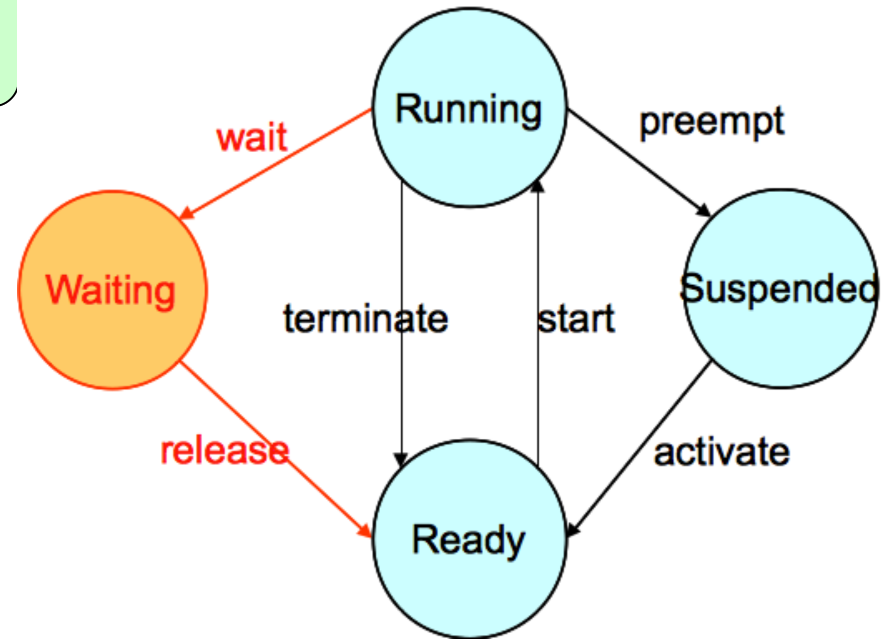
= tâche sans points bloquants

ActivateTask
TerminateTask
ChainTask
Schedule

tâches étendues

= tâche basique comportant des points bloquants





=> Possibility to memorize activation requests

Scheduling

- Fixed priority that cannot be changed

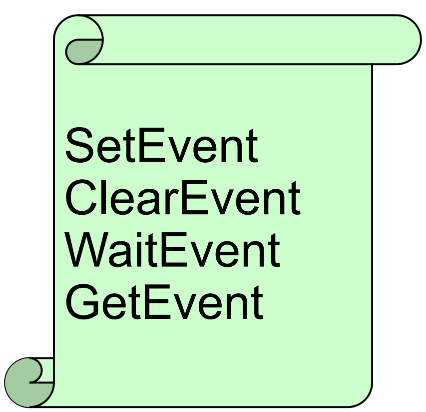
- 3 scheduling modes:
  - preemptive
  - non-preemptive
  - mixed (preemptive for some tasks and non-preemptive for others)

- Management of shared resources with Priority Ceiling Protocol

  (priority inheritance + avoidance of inter-blocking by assigning priority to resources and resource allocation rule)

Event-driven synchronization

- Private events: 1 event is the property of a task (extended)
- Model n producers / 1 consumer
- Explicit consumer expectation (synchronous)
- Memorized events, explicit deletion
- Waiting possible in OR on a list of events



Mutual exclusion

- Services for explicitly taking and releasing a resource
- Using the PCP protocol to avoid:
    - priority inversion
    - inter-task blocking
- A standard resource: Res_scheduler (switch to non-preemptive mode)

GetResource
ReleaseResource

Alarms and counters

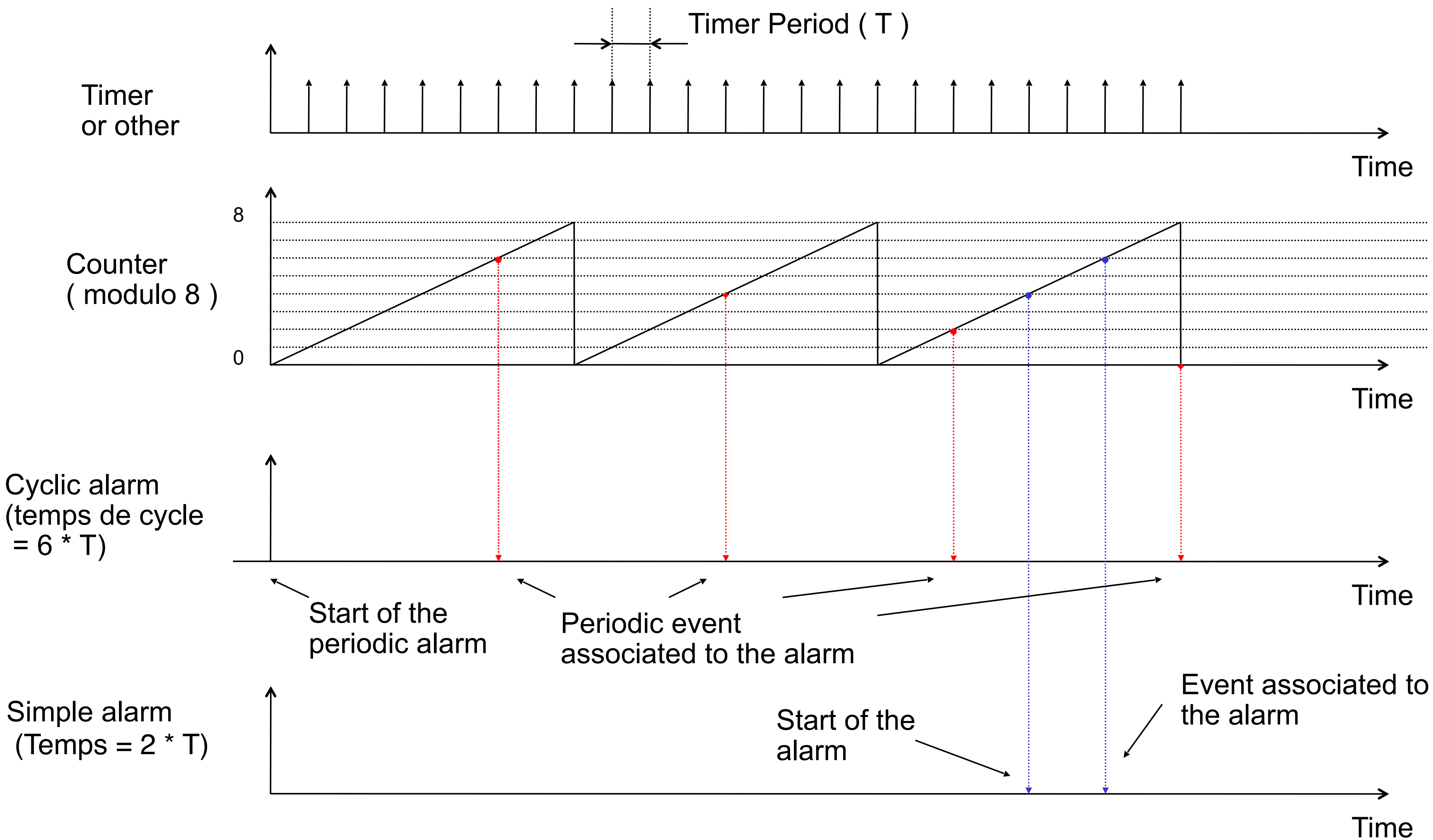
Counters

- recording of external ticks, possibly pre-divisional tick
- finished counter with automatic reset

Alarms

- attached to 1 counter and 1 task
- single or cyclic, absolute or relative triggering

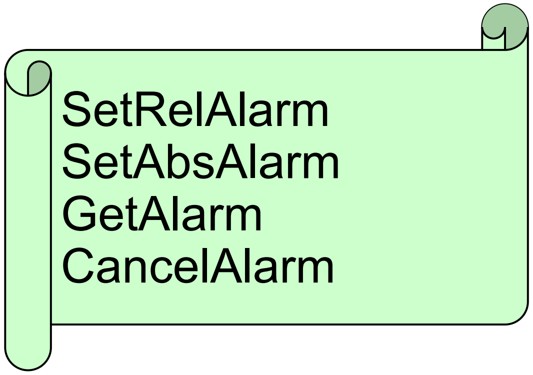
Alarms and counters



Alarms and counters

Applications:

- activation of periodic or non-periodic tasks
- reporting of periodic or non-periodic event occurrences
- watchdog



Interruptions

Interruption categories

Category 1	**Category 2**

ISR (isr_name)
{
 	
code **without calls**
to services of OS
 	
}
	ISR (isr_name)
	
	{
	
	code **with calls**
	to service of OS
	
	}

Inter-process communication

- Communication through messages
  - 1 message = 1 name + 1 data type + attributes

- Asynchronous communication model:
  - `SendMessage` : writing a new value
  - `ReceiveMessage` : reading of the current value

- Effective transmission according to attributes

➡ Resynchronization required :
  - polling of the state of the message, activation of the task or occurrence of event on end of sending or reception...

Inter-process communication

Some attributes of the messages:

- UnqueuedMessage : management type "blackboard"
- QueuedMessage : "mailbox" type management
- On-demand transmission
- Periodic transmission
- Mixed transmission
  ➡ periodical + communication in case of change

Inter-process communication

3 communication types:

- 1 : 1 (1 static receiver)
- 1 : 1 among n (dynamic selection from a static list)
- 1 : n (distribution to n recipients)

2 protocols (unacknowledged) :

- UUDT : non-segmented
- USDT : segmented

OSEK Task configuration: OIL

- OIL language: OSEK Implementation Language
- Hook Routines to temporary control the system
- Task types: conformity classes
  - BCC 1 : Only basic tasks
    --- 1 active task and 1 task per priority level
  - BCC 2 : Only basic tasks
    --- several active tasks + several tasks per priority level
  - ECC 1 : BCC 1 + extended tasks
  - ECC 2 : BCC 2 + extended tasks

OSEK Task configuration: OIL

- Can be split in several OIL file

  Ex: #include « implementation.oil »

- Objects:
  - define the configuration parameters
  - one main object: CPU

```
CPU ATMEL_AT91SAM7S256
{
    OS TRAMPOLINE {
    …
    };
    APPMODE appmode1;
    TASK task1 {
    …
    };
 …
}
```

Some objects

- Object OS
  - Only one OS object per CPU
  - Defines the OSEK configuration(hook routines, scheduler)

```
OS TRAMPOLINE {
    STATUS              = EXTENDED;
    STARTUPHOOK         = FALSE;
    ERRORHOOK           = FALSE;
    SHUTDOWNHOOK        = FALSE;
    PRETASKHOOK         = FALSE;
    POSTTASKHOOK        = FALSE;
    USEGETSERVICEID     = FALSE;
    USEPARAMETERACCESS  = FALSE;
    USERESSCHEDULER     = FALSE;
    };
```

- Object APPMODE (= `appmode1 {};`)
  - Groups task in a same set

Task definition

- Object Task

```
TASK taskname {
    AUTOSTART = FALSE /* task waits until alarm to
                         start */
    PRIORITY = 3; /* task priority,
                     1 = lowest priority */
    ACTIVATION = 1; /* number of occurrence of
                       the task in the ready
                       list  */
    SCHEDULE = FULL; /* NONE = no preemption
                        FULL = possible preemption*/
    STACKSIZE = 512; /* Size of the stack, ie 512*/
    };
```

- Autres champs
  - RESSOURCE = ressource1 (work with ressource n°1)
  - EVENT = event1 (waits event 1)
  - MESSAGE = message1 (synchronization using message1)

Task definition

- Tick definition : object COUNTER

```
COUNTER SysTimerCnt {
    MINCYCLE = 1; /* tick period */
    MAXALLOWEDVALUE = 100; /* max value of the cpt*/
    TICKPERBASE = 1; /* cpt step */
    };
```

- Periodic activation of tasks:
  - One alarm per task
  - One counter associated to an alarm

```
ALARM cyclic_alarm1 {
    ACTION = ACTIVATETASK {
        TASK = taskname;
    };
    AUTOSTART = TRUE {
        ALARMTIME = 1; /* 1st alarm instance */
        CYCLETIME = 1; /* period, here 1 tick */
        APPMODE = appmode1;
    };
};
```

Conclusion on OSEK

- **A mature and complete proposal**
- **Industrial products**
- **An important step towards application portability, reuse of components in ECUs ...**
- **Variants : OSEKtime OS**
  - ➡TT (time triggered) tasks for critical applications
  - ➡joins the APEX model for the partition level

3. APEX : ARINC 653 standard

Introduction

Specify an interface between an OS of an avionics resource and application software: APEX interface (APplication / EXecutive)

Start of work: 1991

Publication of the first ARINC 653 standard: Summer 1996

Definitions

Application :

— Software realization of an avionics function

— consisting of one or more partitions

Example: Automatic Pilot

Partition :

- Entity of execution of an application
- Runs on a single processor
- Deterministic allocation of resources to partitions
- Spatial (memory) and temporal (CPU) segregation unit
- Consisting of one or more processes

Example : FM1, FM2, FE1-COM, FE1-MON, FE2-COM, FE2-MON, FG1-COM, FG1-MON, FG2-COM, FG2-MON

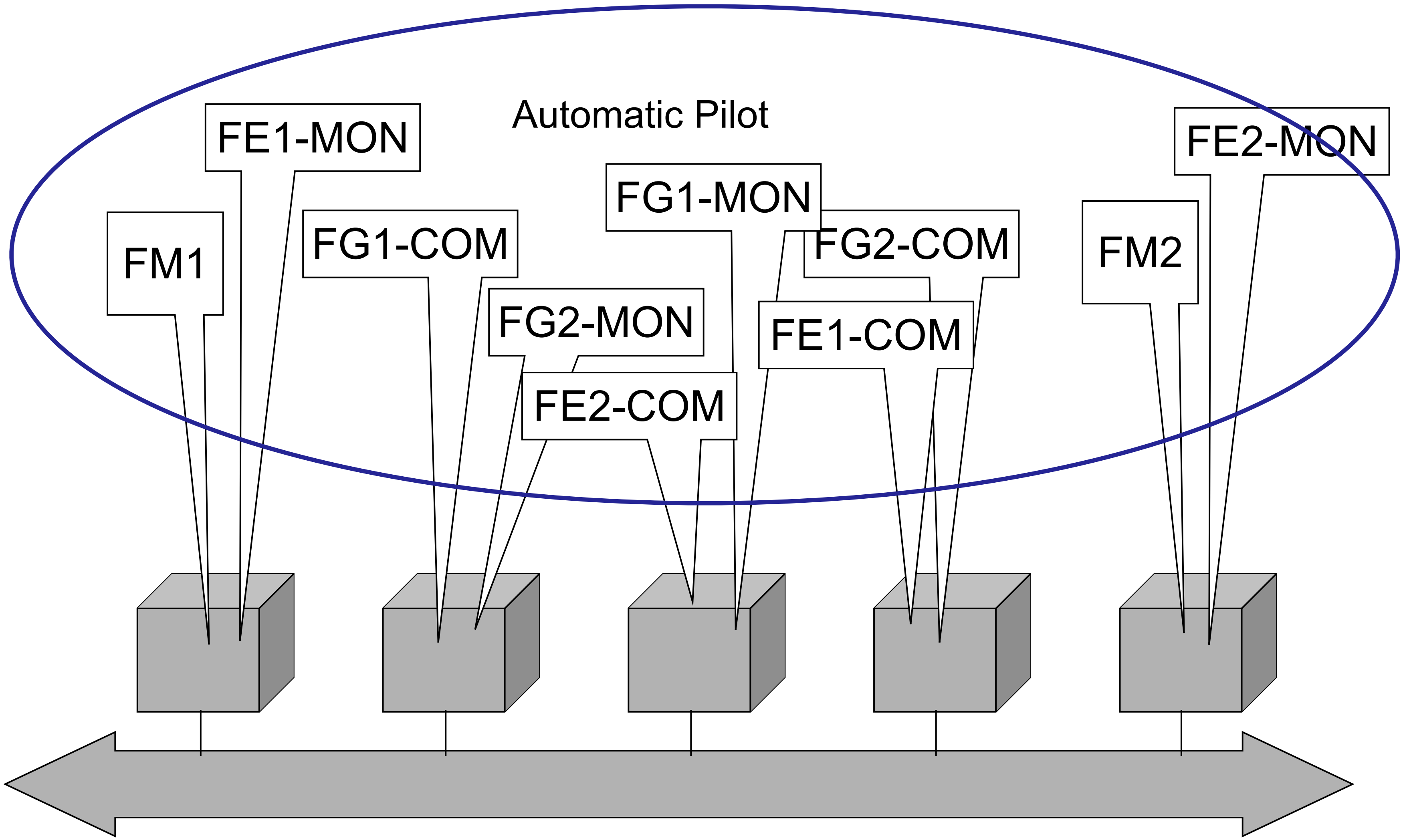
Definition

Process:

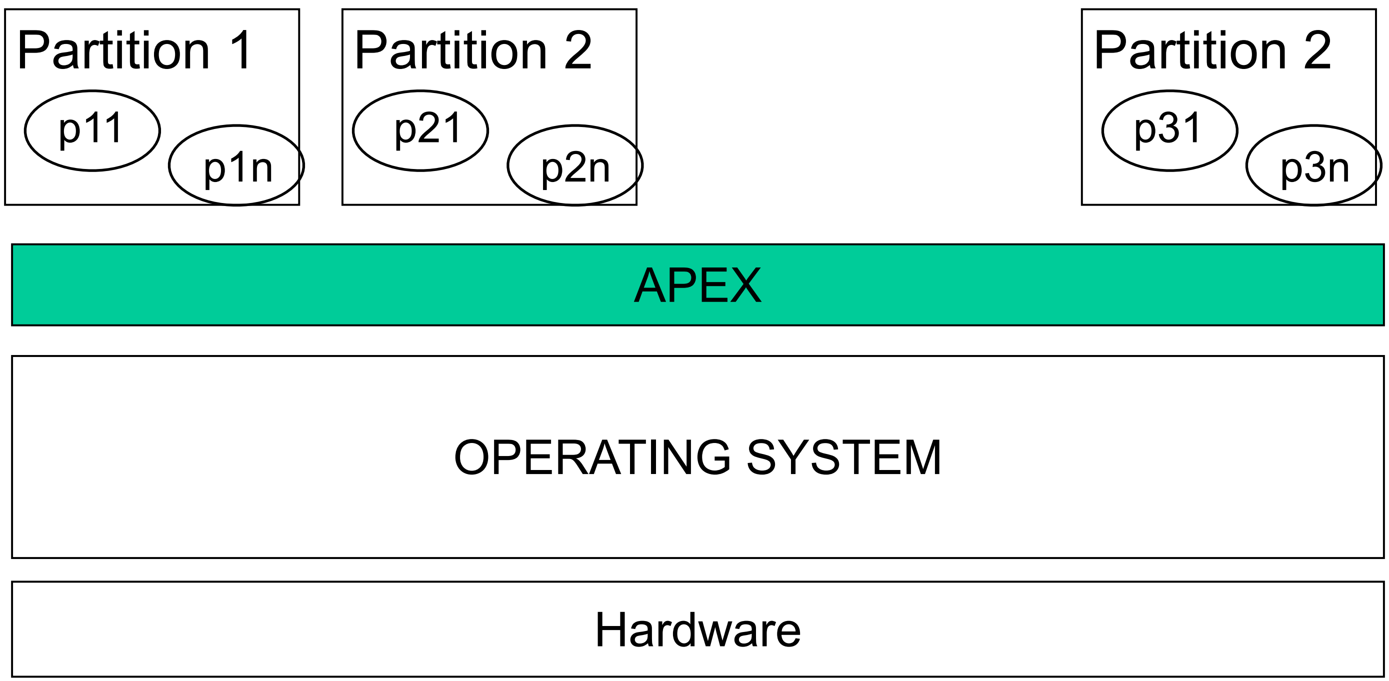
- A program unit that runs in the partition environment.
- Concurrent execution of processes to perform the avionics function

Example: a control law, a control logic…

Architecture



Architecture



Operating system :

- Schedules the partitions of the module
- Schedules the processes in the partition
- Ensures spatial and temporal segregation (partitioning)

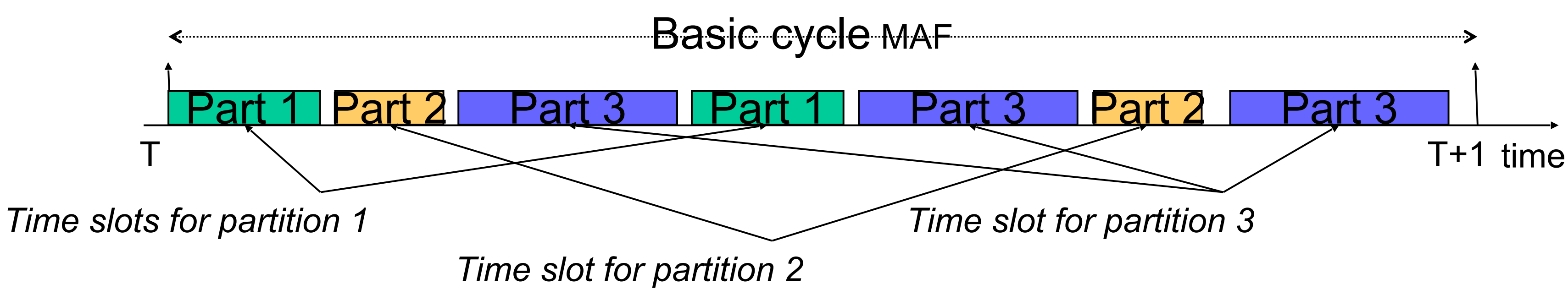
Resource Sharing

Spatial sharing:

Predetermined memory area for each partition

Time sharing (CPU):

- A partition has no priority
- Deterministic and cyclic allocation of the processor to partitions:
  - => The OS repeats a basic cycle (MAjor time Frame: MAF) of fixed duration.
  - => Assign one or more time slots in the MAF to each partition.



=> Resource allocation is defined by configuration and cannot be changed dynamically.

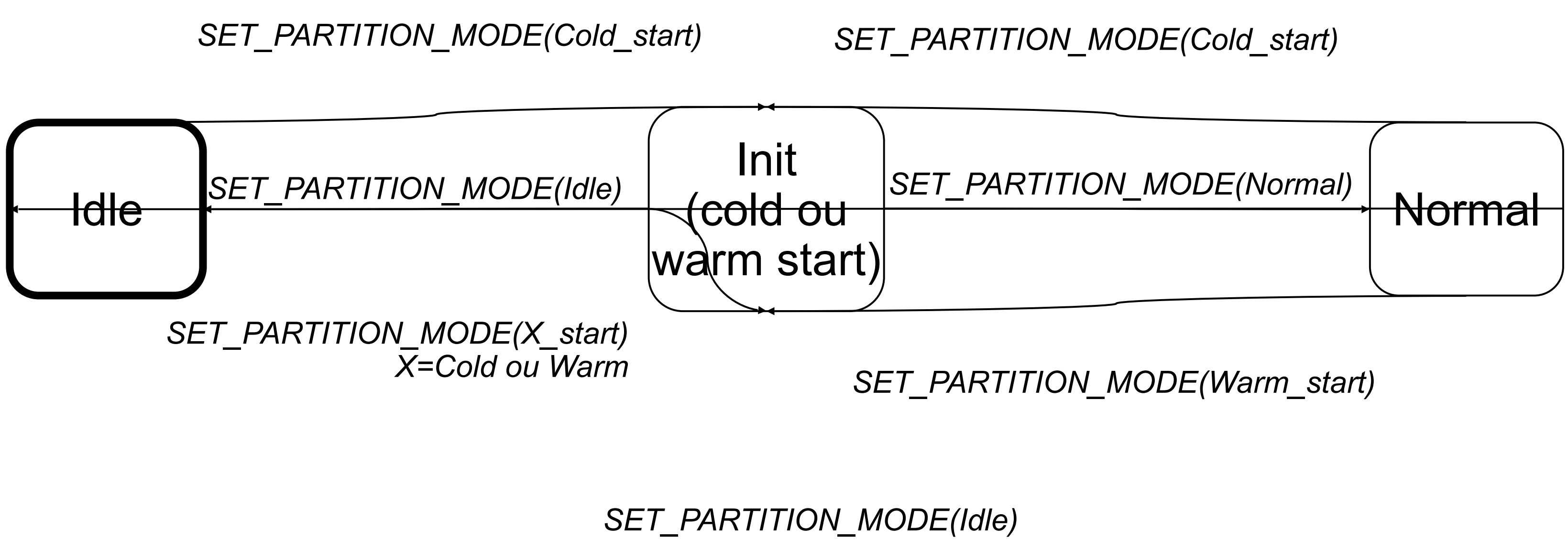
Partitions management

- The OS starts the partition scheduler at the end of module initialization.
- During its time slots, a partition has the following states
  - Idle :
    - No process is running.
  - Cold_start ou Warm_start : initialization of the partition
    - Cold start: abstraction of the previous context of the partition
    - Warm start: recovering the previous context of the partition
  - Normal : starting process scheduling
- Outside its time slots a partition is suspended.
- The partition creates all its objects (ports, process, semaphores...) during its initialization phase.
- The partition starts process scheduling at the end of its initialization phase.

Partitions management

APEX service for partitions

- SET_PARTITION_MODE: changes the mode (Idle, Cold or Warm start, Normal)
- GET_PARTITION_STATUS: provides the status of the partition



Process management

- 2 process types
  - Periodic: execution in regular intervals
  - Apériodic : execution on event occurrence
- No segregation between processes within the same application
- Processes on one partition are not visible to other processes on other partitions.
- Each process has a priority
- Process scheduling is based on a preemption algorithm by priority level and current process state
- Only one process per partition is "running" at any given time.

Process management

APEX services for processes

CREATE_PROCESS :

- Creation of a link between the name of the process and its reservation on partition initialization
- Putting the process in a S*leeping* state and returning an identifier (id)

START :

- Initialization of the process
- Change from *Sleeping* to *Ready* state

STOP :

- stopping a process by removing access rights to the processor
- Change from *Ready* or *Waiting* to *Sleeping* state

STOP_SELF :

- Auto-stop of the running process
- Change from *Running* to *Sleeping* state

SUSPEND :

- Suspension of the process until resumed by another process
- Change from *Running* to *Waiting* state

SUSPEND_SELF :

- self-suspension of the process
- Change from *Running* to *Waiting* state

Process management

APEX service for processes

- RESUME :
  - Restarting a suspended process
  - If the process is already waiting for a resource, it remains in the *Waiting* state, otherwise it switches to the *Ready* state
- SET_PRIORITY :
  - Changing the priority of a process
- LOCK_PREEMPTION :
  - Incrementing the preemption counter of the partition and blocking the preemption mechanism
- UNLOCK_PREEMPTION :
  - Decrementing the preemption counter of the partition and unlocking the preemption mechanism if this counter is null
- GET_PROCESS_ID :
  - returns the identifier of a process by giving its name
- GET_PROCESS_STATUS :
  - returns status information about a process

Process management

Time management

Time is expressed in real time (absolute).

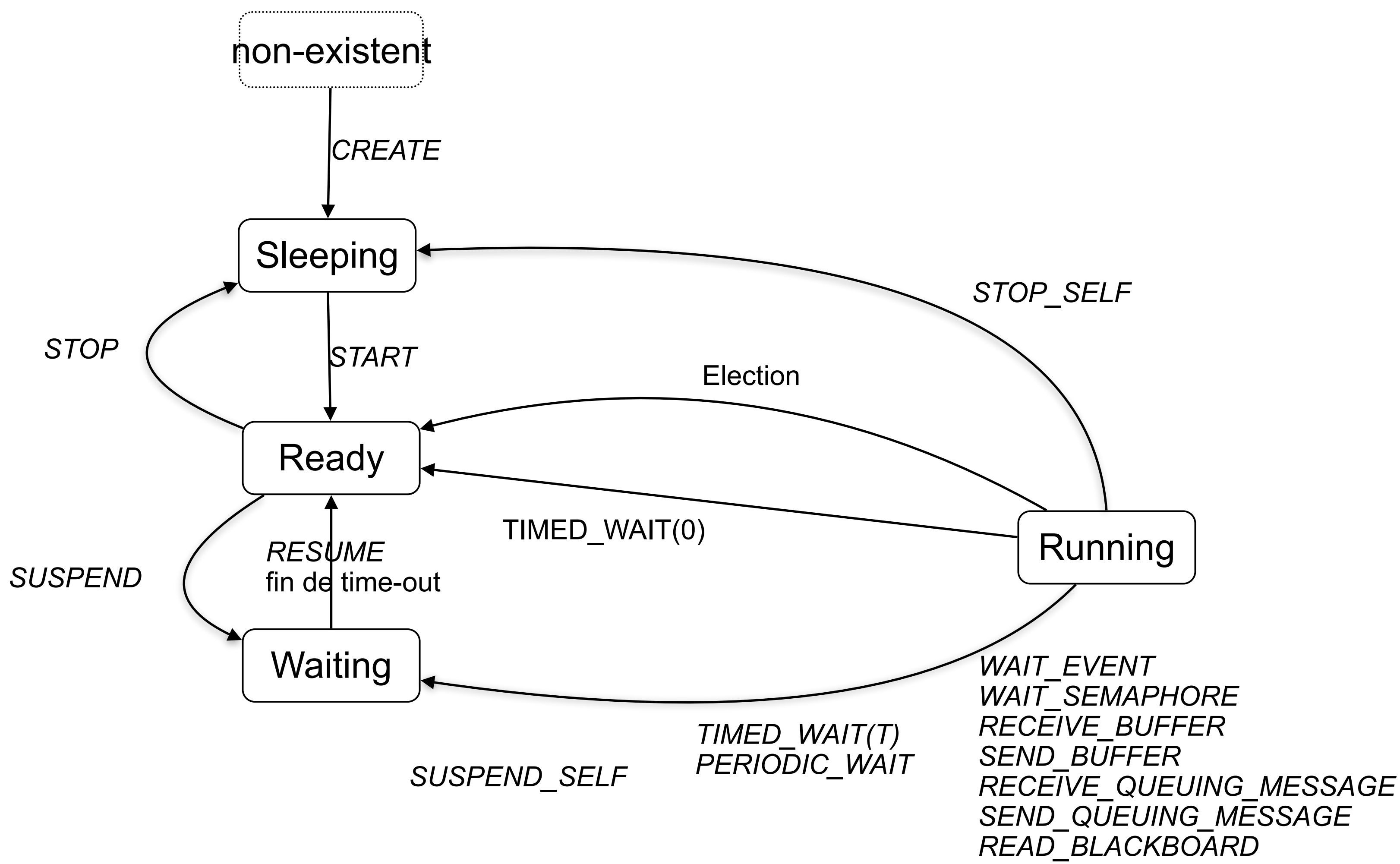
The OS ensures time segregation between partitions

APEX services for time management :

- TIMED_WAIT :
  - Suspension of a process for a given minimum amount of time
  - Change from Running to Waiting state
  - A time equal to zero performs a "Round Robin" on processes with the same priority.
- PERIODIC_WAIT
  - Suspension of a periodic process until the next period
  - reallocation of the process time budget at the beginning of the period
- GET_TIME
  - returns time (absolute) and local time to the module

Process management

State graph of the processes



Inter-process communication

Processes on the same partition can communicate and synchronize through

- communication via Buffer or Blackboard type mailboxes
- use of global variables to the partition
- synchronization by semaphore and event mechanisms

A mailbox allows the exchange of messages between several processes without indicating the names of the sending and receiving processes.

A memory area is reserved for the initialization of the partition for these communication / synchronization objects.

Inter-process communication

The "Buffer" type mailboxes:

communication by message that can carry different data

a message occurrence does not overwrite previous occurrences

messages are stored in FIFO message queues

the message tails are limited in length

a message queue can be full

processes trying to read from an empty message queue are put on hold

processes seeking to transmit in a full message queue are put on hold

Corresponding APEX services:

CREATE_BUFFER

SEND_BUFFER

RECEIVE_BUFFER

GET_BUFFER_ID

GET_BUFFER_STATUS

a timeout can be specified

Inter-process communication

"Blackboard" type mailboxes:

- data write communication
- a data entry overwrites the previous data
- a written data remains displayed until the next data is written or a delete request is made.
- all processes waiting on an empty "Blackboard" are woken up when writing data (change from *Waiting* to *Ready* status)

Corresponding APEX services:

- CREATE_BLACKBOARD
- DISPLAY_ BLACKBOARD
- READ_ BLACKBOARD → a timeout can be specified
- CLEAR_ BLACKBOARD
- GET_ BLACKBOARD _ID
- GET_ BLACKBOARD _STATUS

Inter-process communication

Smaphores:

=> semaphores with counter

resource and signals the semaphore when it releases the resource

if counter > 0: the value represents the number of processes that can access the resource.

if counter = 0: resource unavailable

processes waiting on a semaphore are ordered either by FIFO or by priority

Corresponding APEX services:

CREATE_SEMAPHORE

WAIT_ SEMAPHORE → a timeout can be specified

SIGNAL_ SEMAPHORE

GET_ SEMAPHORE _ID

GET_ SEMAPHORE _STATUS

Inter-process communication

Events

=> event at level (UP and DOWN)

allows to synchronize processes on the partition

when an event is set to UP, all processes pending on that event are set to *Ready* and a new process scheduling takes place

when a process is waiting for an event set to DOWN, it goes to the *Waiting* state with possibly waiting for a time out.

Services APEX correspondants :

- CREATE_ EVENT
- SET_ EVENT
- RESET_ EVENT
- WAIT_EVENT → a timeout can be specified
- GET_ EVENT _ID
- GET_ EVENT _STATUS

Inter-partition communication

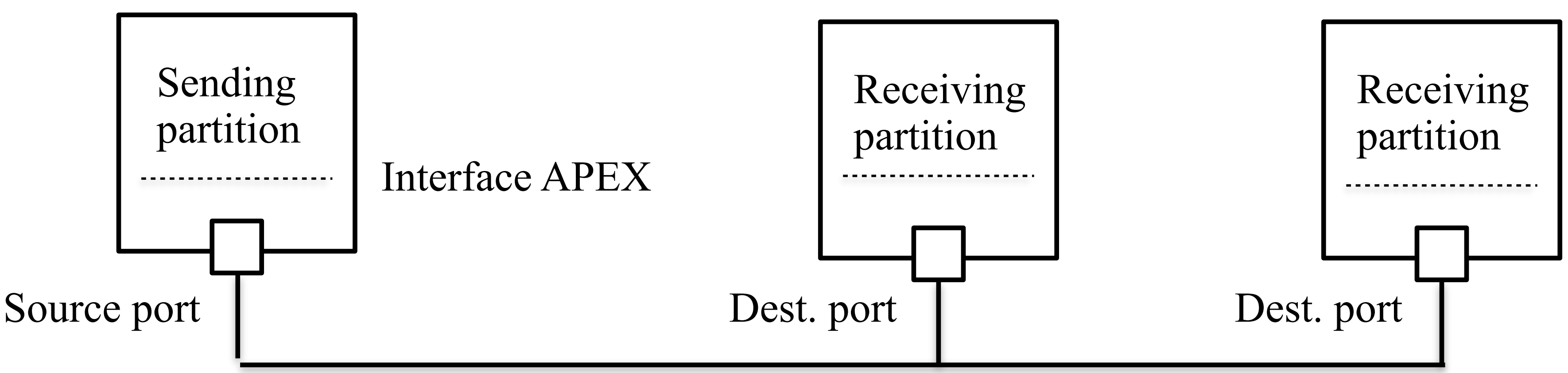
Communication between partitions (of the same module or not) is only done by message exchange.

A transmitter is connected to one or more receivers by a logical channel called "Channel"

The transmitting or receiving entities correspond to the partitions

A partition accesses a channel via a "Port

=> APEX service for sending and receiving via ports



Inter-partition communication

- The transmitting partition does not know the name and location of the receiving partitions.

- The physical connection between the sending and receiving ports is established by static network configuration.

- The communication protocol between transmitter and receiver must be the same.

  (same message format, same acknowledgement policy...)

- The periodic call of an APEX transmission service generates a periodic message, and conversely, an aperiodic call generates an aperiodic message.

  => the periodicity is not an attribute of the port

Inter-partition communication

Transfert protocol:

- Sampling mode:
  - the message always conveys the same updated data
  - each occurrence of a message overwrites the previous one
  - messages are of fixed size
  - the messages are not segmentable by the OS

- Queuing mode:
  - the message can convey different data
  - occurrences are saved in a FIFO
  - messages can be of variable length
  - messages are segmentable by the OS

Inter-partition communication

Ports attributes

- — Identifier (id): value returned when creating the port
- — Name
- — Transfer protocol: sampling or queuing
- — Direction: input or outpout
- — Maximum message size
- — Port size
  - identical to the message size for sampling ports
  - FIFO depth for queuing ports
- — Refresh period for sampling ports
  - allows to monitor the freshness of the last received message
- — Queuing mode waiting policy:
  - FIFO or priority

Inter-partition communication

Sampling APEX services

CREATE_ SAMPLING_PORT

WRITE_ SAMPLING_MESSAGE

READ_ SAMPLING_MESSAGE

reads the last message received in the port and indicates if the age of the message is consistent with the Refresh period attribute of the port

GET_ SAMPLING _ID

GET_ SAMPLING _STATUS

Service APEX queuing

CREATE_ QUEUING_PORT

SEND_ QUEUING _MESSAGE

RECEIVE_ QUEUING _MESSAGE → a timeout can be specified

GET_ QUEUING _ID

GET_ QUEUING _STATUS

A conclusion

Summary:

- A two-level OS:
  - partition + static cyclic scheduling
  - process + dynamic scheduling by priority

- Intuitive objective: emulate classic federated architectures where each function has its own resources and manages its own processes as it sees fit

- Partition level required for certification requirements

  => guarantee that a faulty function will never be able to disrupt another one

  => notion of strict partitioning

Question: Wouldn't it have been simpler to remove the partition level and multiply the physical resources?

=> Non-Integrated Modular Avionics

=> Only resource sharing = the network