The Real-Time Thing

What the hack is real-time and what to do with it

22C3
30. December 2005

Erwin Erkinger
vindaome@p-e.at

Content

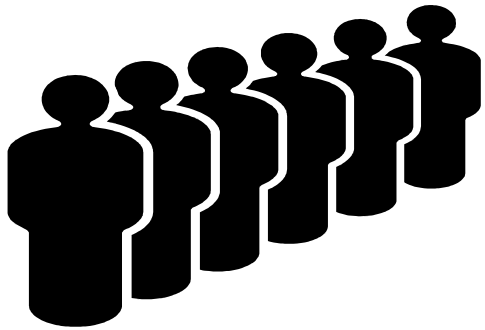
- Part 1: Introduction
  - the vocabulary and the concepts
- Part 2: Practical
  - the tools
  - with RTAI Linux
- Part 3: Pitfalls and design limits
- Conclusion

Part 1

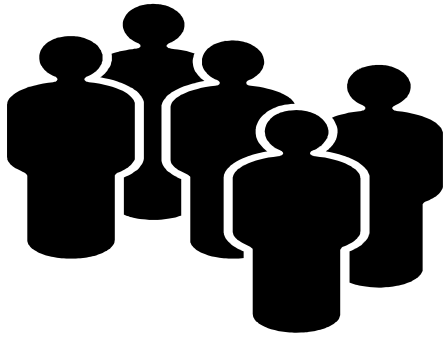
The Introduction

Software Design

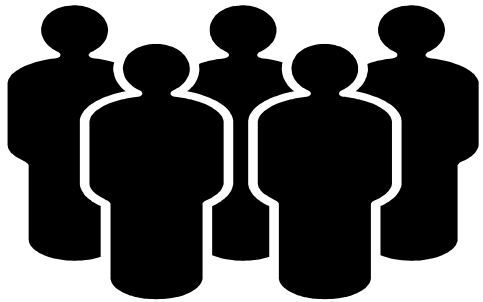
- Sequential Programs
  - Execution is in a predetermined chain
  - Results independent of execution speed
  - Can often be analytically verified on correctness



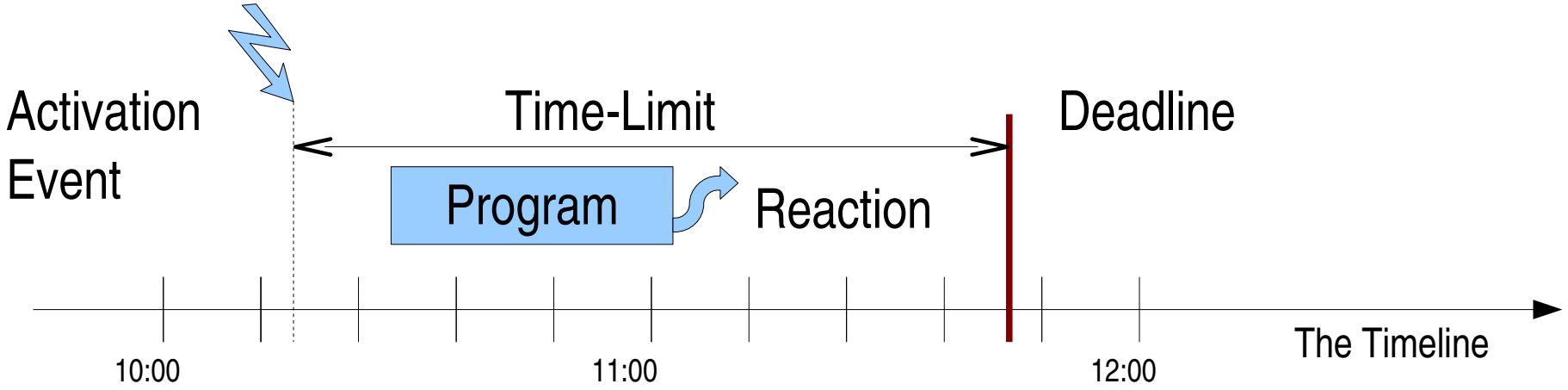
- Concurrent Programs
  - Multiple sequential programs
  - Execution in pseudo-parallel – concurrent
  - Results may dependent of execution speed
  - Correctness of is difficult to determine



- Real-Time Programs
  - As concurrent programs PLUS
  - Correctness must be provided at “deadline”
  - Results now dependent on absolute execution speed



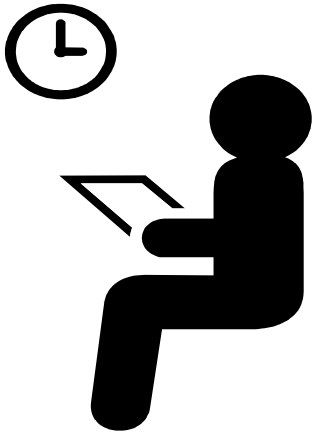
Real-Time – a guarantee



- Real-Time programming guarantees the reaction on an event before the deadline
- The time-limit depends only on the application
- The exact time of the program execution is not defined

Real-Time = Computer reacts in time

Hard and Soft



- Hard-Real-Time
  - Will never fail
  - Correctness may be verified by analysis
  - A RT-OS is needed
  - Typical application: Fly-By-Wire
- Soft-Real-Time
  - May fail (and will)
  - Correctness can not be verified
  - No OS is need at all
  - Typical application: Linux, Windows...

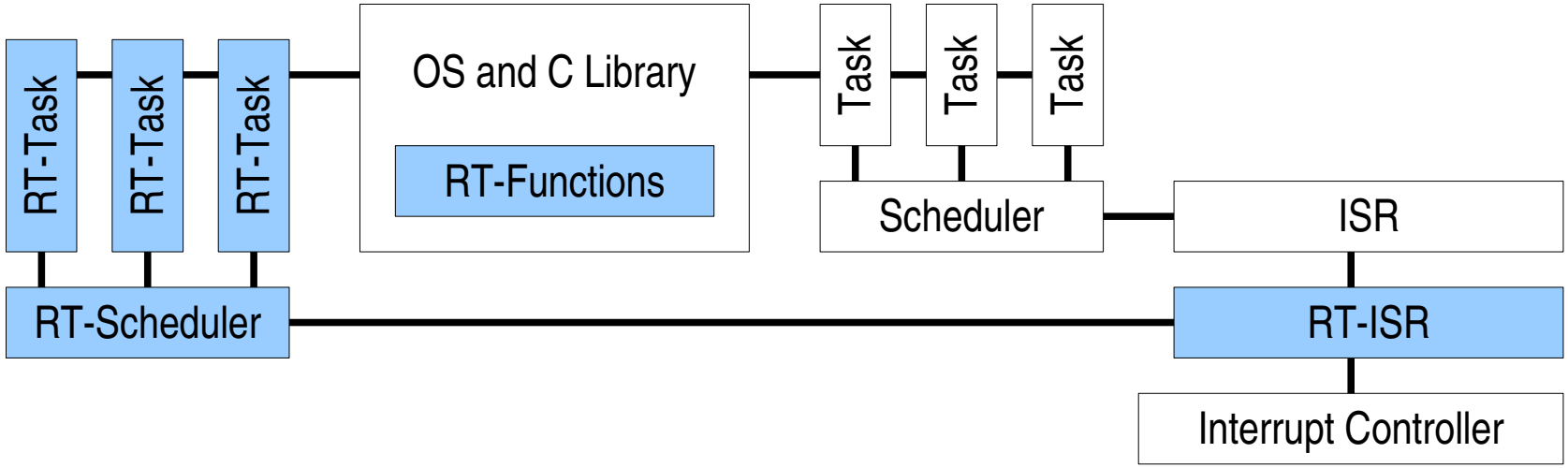
Embedded and Real-Time

Embedded and Real-Time go together like a horse and a chariot ...

But Embedded ≠Real-Time

- Embedded is a special-purpose computer system which is completely encapsulated by the device it controls
- Real-Time can be achieved on every computer system with the appropriate means

Linux and Real-Time

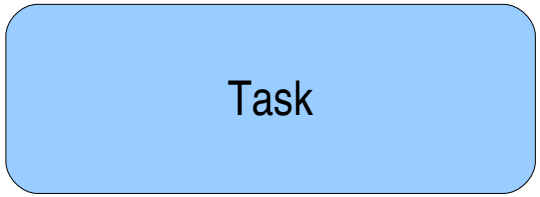


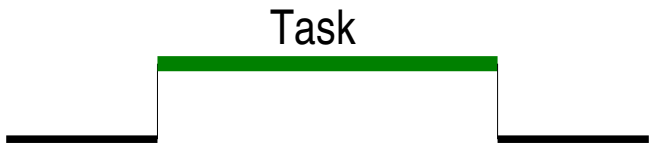
- Standard Linux is a classic concurrent system
  - Execution in concurrent pseudo-parallel
  - No Deadline guarantees
  - Strong variant of execution time
- “Real-Timing” of Linux
  - Remove reason for unreliable of task-scheduler
  - Add/change real-time functions

Part 2

The Practical Part

The Task

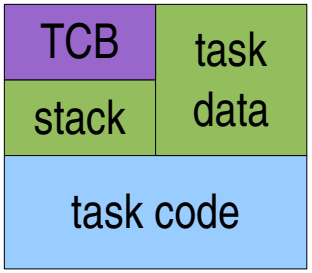




- A Task ...
  - is an independent thread of execution
  - contains or uses functions (task ≠ function)
  - may be synchronised and/or communicate to another task
  - may be non-preemtible, preemtible or timesliced
  - has a priority
  - usually has states: Blocked, Ready, Running
  - is maintained by the RT-OS (Stack, TDB, State...)

The Task - Example

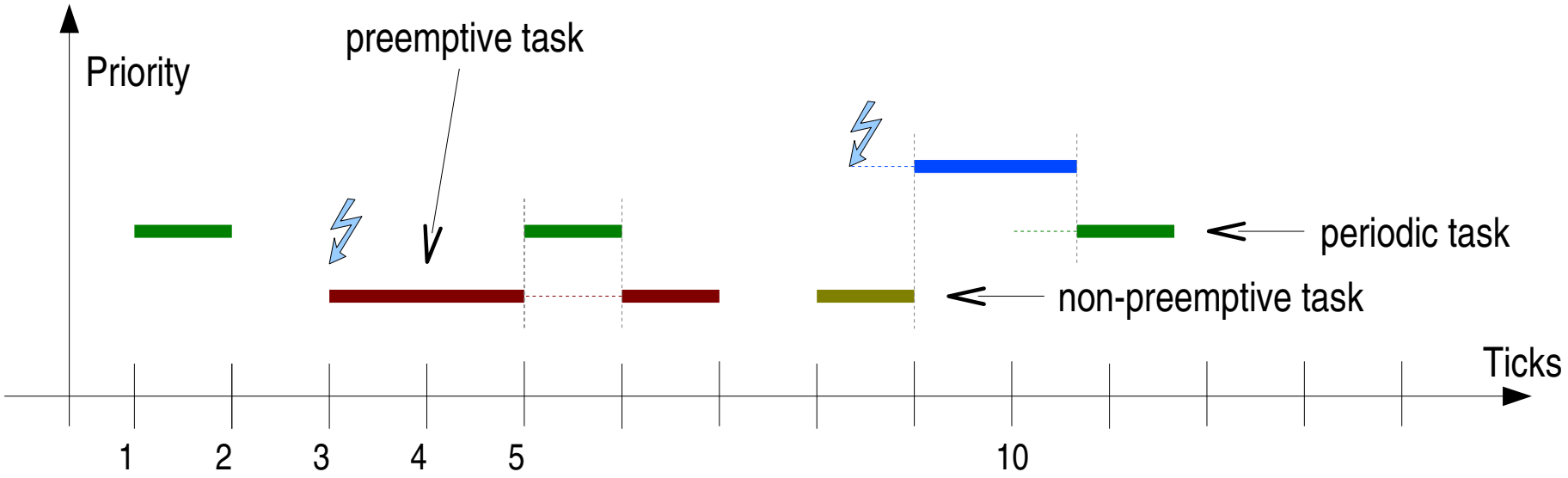
- A Task ...
  - contains an “endless” loop
  - “blocks” on resources
  - is setup by init function (insmod in case of Linux)
  - may be marked as periodic




```
RTAI example of init_module:

    rt_task_init(   & mytask_env
                  , mytask_func
                  , task_data
                  , STACK_SIZE
                  , TASK_PRIO
                  , USES_FPU
                  , mytask_act_func
                );
```

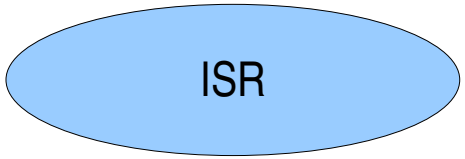
```
void mytask_func (int data)
{
    init_mytask(data);

    while(no_error)
    {
        wait_on_message();

        do_something();
    }
}
```

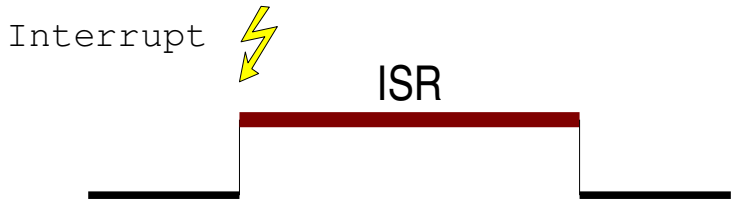
The Task - Scheduling



- A Task may be scheduled ...
  - periodic or aperiodic
  - preemptive or non-preemptive

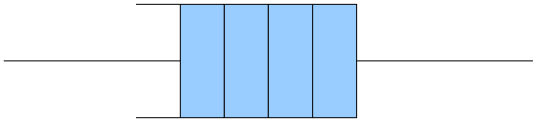
The ISR





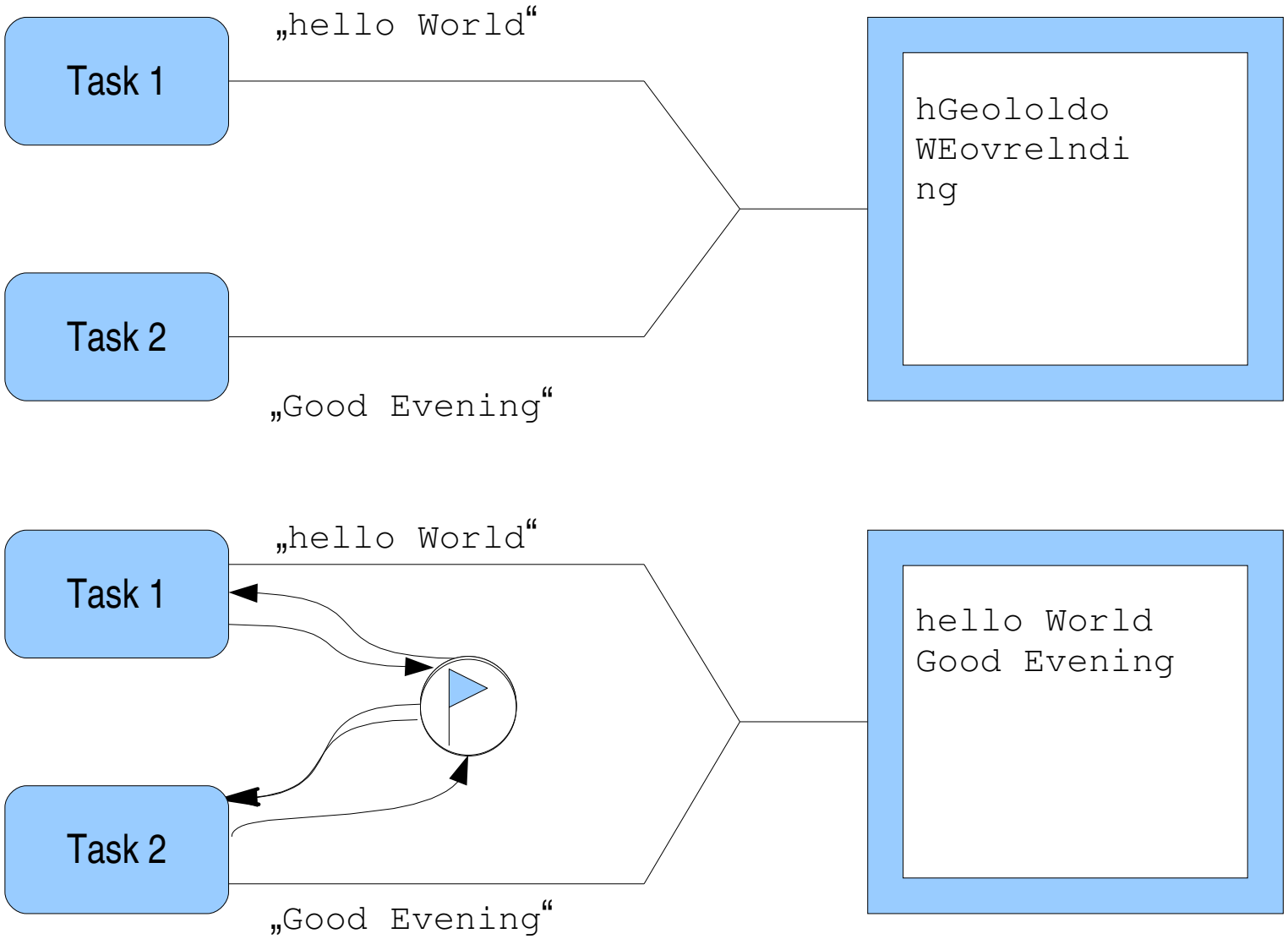
- An Interrupt Service Routine ...
  - is a function which is activated on an interrupt
  - will return after servicing the interrupt
  - may use an subset of the RT-OS Library
  - is usually used to activate RT-OS signals for further processing by tasks
  - has the priority of the interrupt
  - is usually non-preemptive

Message Queues

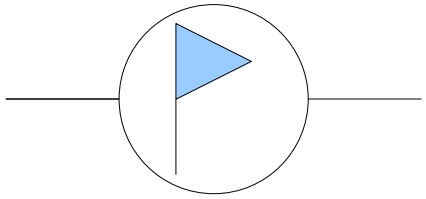


- The Message Queue ...
  - is used for inter-task data communication
  - latches data between two tasks
  - usually fixed length data elements are used
  - implementation and additional features varies strong between RT-OS's
    - mailbox, queue and pipes in RTAI RT-Linux
    - static and queued messages in ARINC 653
    - FIFO, FILO feature in pSOS ...
  - but common function for all RT-OS's
    - to hold N elements of size M

Semaphore principle

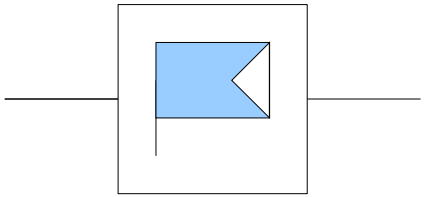


Semaphore



- The Semaphore ...
  - is used for inter-task control communication
  - is often used for resource protection
  - may count
  - functions:
    - Generating and Deletion
    - Signalling - “v”, “give” operation – increases count
    - Wait - “p”, “take” operation – decrease and block
- The Mutex ...
  - is a semaphore which is owned by one task
    - task which locks semaphore has to release it later

Events



- The Event ...
  - is used for inter-task synchronisation
  - is used to trigger a task on combinations
  - reflect a change of an state
    - e.g. event engine is shut-down – vs. state engine is running and engine is stopped.
  - is not count – a repeated occurrence of an event only triggers it once

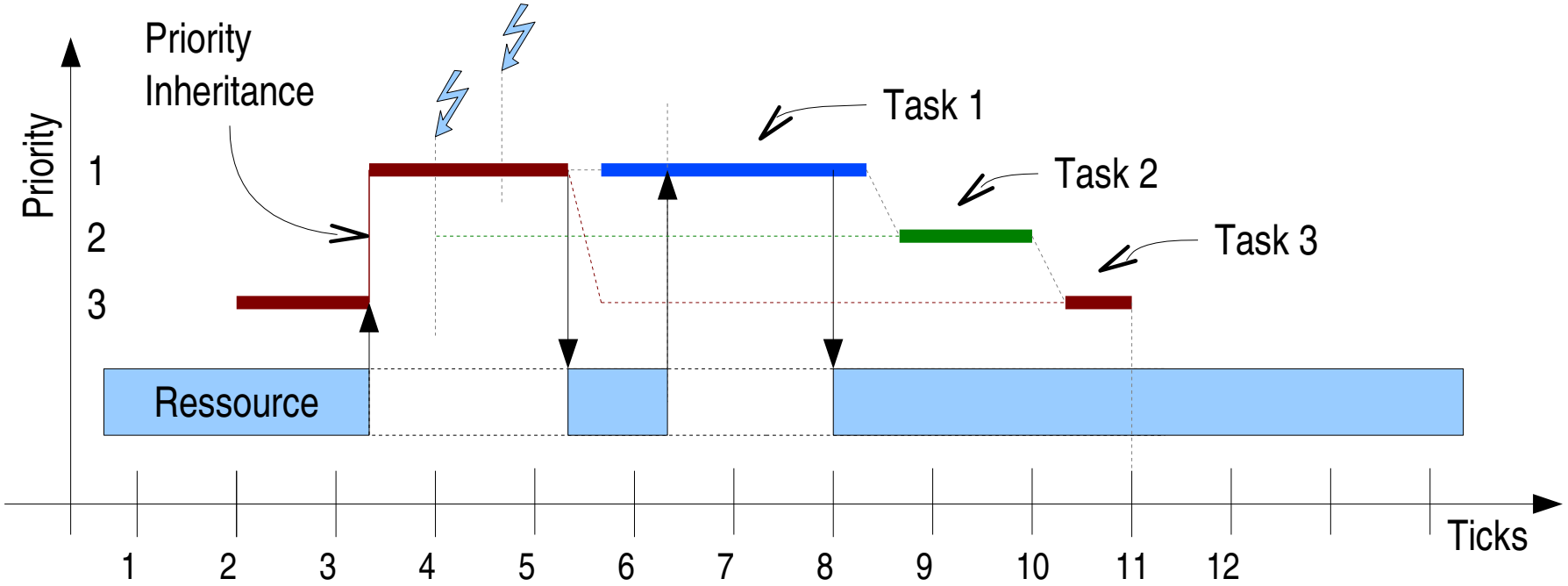
Others

- The Timer ...
  - is used to generate single-shot or periodic events
  - is used to delay or wait
  - usually one system timer is used by the RT-OS to provide timer services
- Time-out on blocking
  - The RT-OS provides a time-out for the blocking on semaphores and message queues
- User/Kernelspace communication
  - RTAI provides RT-FIFO's and RT-shared-memory for user/kernelspace communication

Part 3

The Traps and the Pitfalls

Priority Inversion



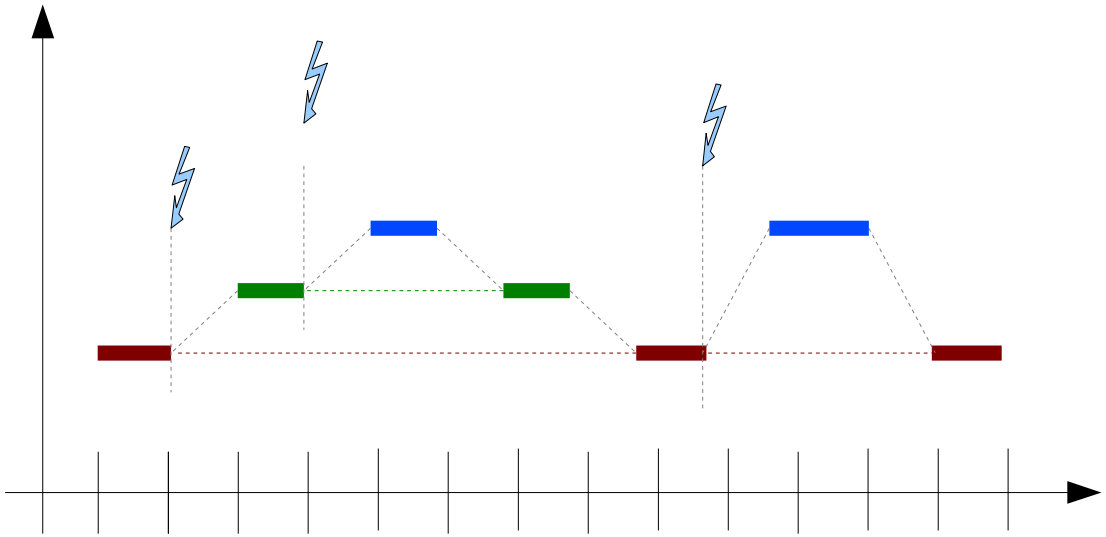
- Priority inversions is when a lower priority task blocks a higher priority task
- Prevention - priority inheritance
  - the blocking task inheritances the priority of the highest priority blocking on the same resource

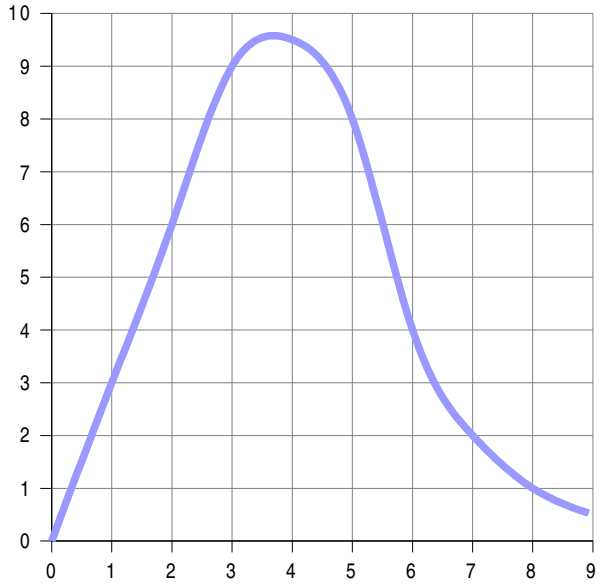
Deadlock

Picture source: wikipedia.org

- Tasks block mutual on resources
- Prevention
  - system design (e.g. banker's algorithm)
  - usage of additional synchronisation objects

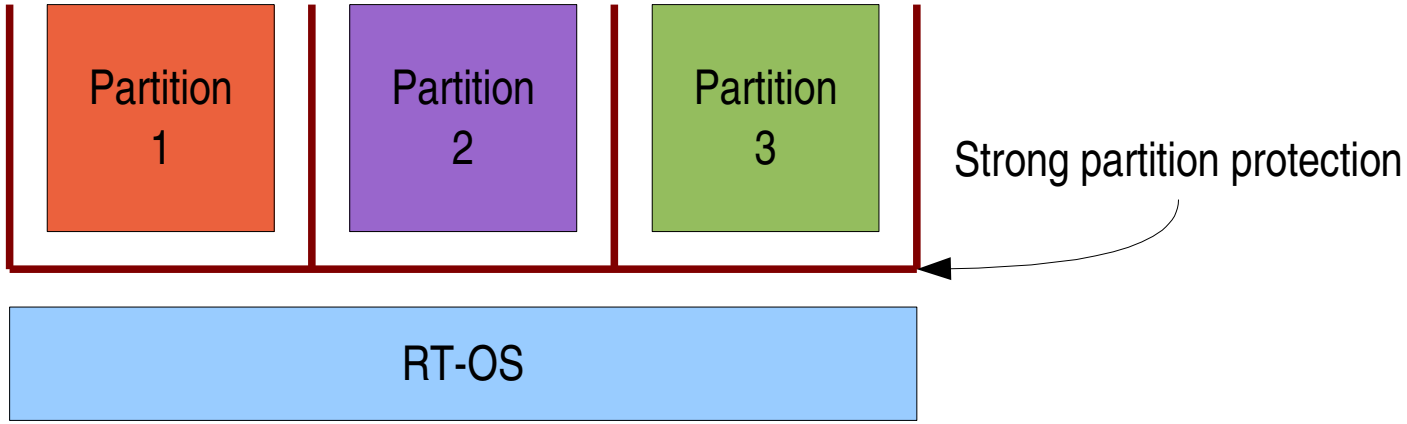
High Load Scenario





- High Load decreases overall CPU throughput
  - task switch time exceeds task execution time
  - system starvation possible
- Alternative Concept - fixed time schedule
  - time triggered – instead of event activation

Protection



- System is fragmented in partitions
  - “fixed” schedule “timeslices” the CPU time to the partitions
  - strong protection in time and space domain
  - each partition is independent from others
  - prevents error propagation through system
  - used in high reliability designs (e.g. Fly-By-Wire)

Links

- RTAI
  - Current stable version 3.1-r2
  - http://www.rtai.org/
- Wikipedia
  - http://de.wikipedia.org/wiki/Realtime
- DENX – embedded Linux
  - http://denx.de/
- ADEOS – An alternative Linux Interrupt handler
  - http://home.gna.org/adeos/
- LynxOS – An Linux call compatible RT-OS
  - http://www.lynuxworks.com/rtos/rtos.php
- IBM – Linux articles – eg. posix threads
  - http://www-128.ibm.com/developerworks/library/l-posix2