# Föreläsning 7 IS1300 Inbyggda system

- Short note about solving problems with ASIC or microcontroller
- Real time operating system
  - Basic concepts
  - Scheduling principles

- Lab exercise, discussion

# ASIC or microcontroller?

A task can be solved with

- Hardware
  - ASIC (Application Specific Integrated Circuit)
  - Custom chip, large quantities
  - FPGA (Field Programmable Gate Array)
- Software
  - Programming processor or microcontroller
  - MPU (Microprocessor Unit)
  - MCU (Microcontroller Unit)
- MCU or MPU can be integrated in FPGA
  - IP cores (Intellectual Property)
  - Example of State of the art today (Xilinx)

# Tasks and processes

- Task
  - Real time application
- Process
  - Execution of program
- Event
  - Specific time and place

# Time requirements

- Release time
  - Time at which process become ready to execute
- Deadline
  - When the computations has to be finished
- Period
  - Time between successive executions
- Rate
  - Inverse of period (rate=1/period)

# CPU metrics

- Initiation time
  - Time at which process starts executing
- Completion time
  - Time at which process finishes

- Utilization U $$U = \frac{\text{CPU time for useful work}}{\text{total available CPU time}}$$

# RTOS – Real Time Operating Systems

- The RTOS determines which applications should run in what order and how much time should be allowed for each application before giving processor access to another process:
  - manages the sharing of internal memory among multiple tasks.
  - handles input and output to and from attached hardware devices, such as serial ports, buses, and I/O device controllers.
  - sends messages about the status

Source slide 6-12: An introduction to Real-Time Operating Systems www.quadros .com

# Reasons to Use an RTOS

A well-designed RTOS provides a number of tangible benefits to the developer. It

- abstracts away the complexities of the processor,
- provides a solid infrastructure constructed of rules and policies that provide consistency and repeatability
- simplifies development and improves developer productivity by utilizing high level kernel objects to easily handle complex functions
- integrates and manages resources needed by communications stacks and middleware (TCP/IP, USB, SDIO, CAN, FAT and Flash file systems, etc.)
- optimizes use of system resources and improves pro-duct reliability, maintainability and quality

An RTOS can bring all those elements together into a plat-form that allows the application developer to begin devel-opment at a much higher point, enabling a shorter time-to-market with higher reliability and lower risk.

# RTOS properties

RTOS needs to:

- manage the processor and other system resources to meet the requirements of the application
- be able to respond to, and synchronize with, events
- be able to move data efficiently between processes
- be able to manage the demands of the process with respect to an independent variable such as time
- perform in a predictable manner with operations that take place within a predictable about of time

# RTOS kernel

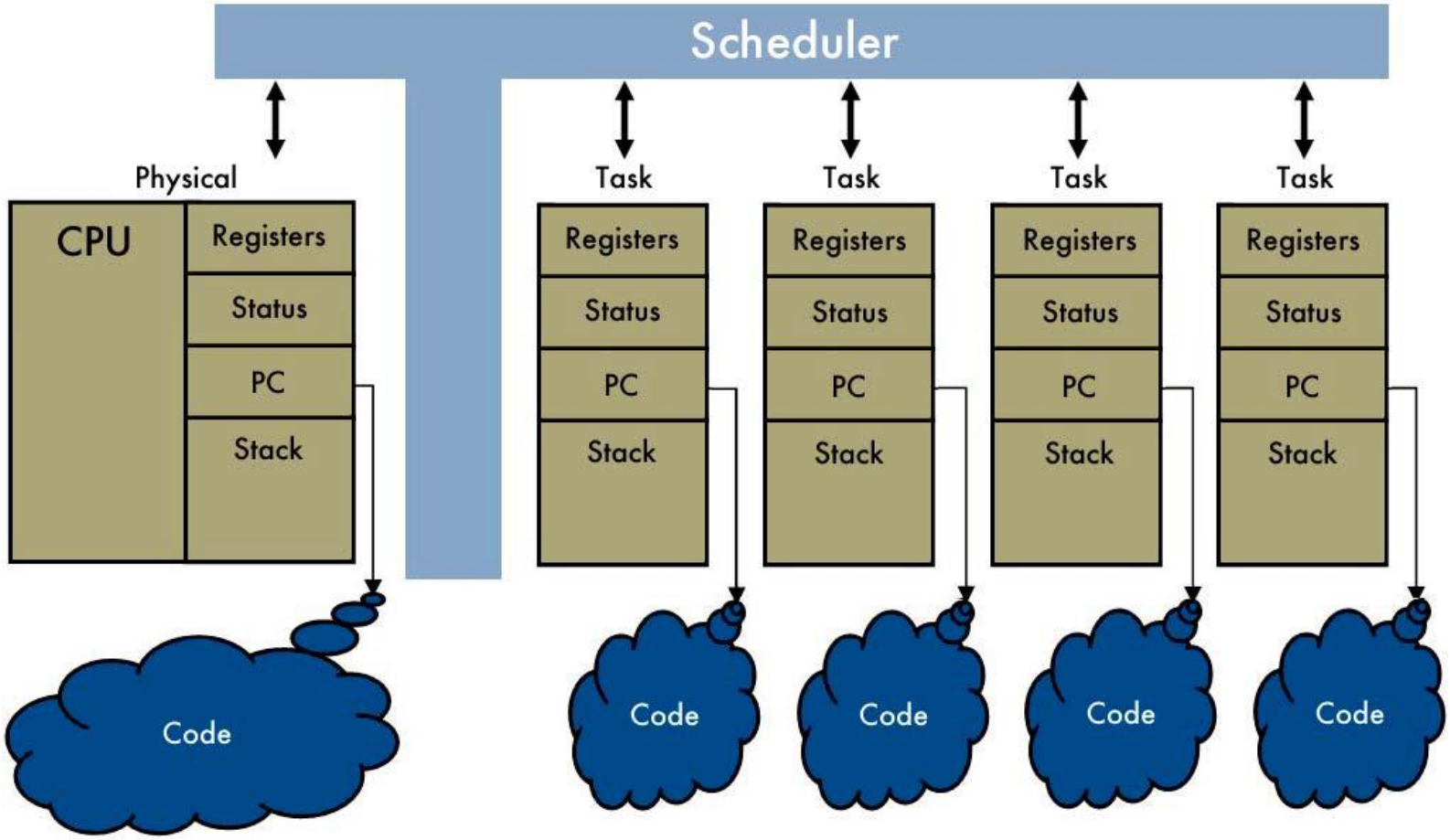
A RTOS kernel includes

- a *scheduler* that determines which programs get access to the CPU and in what order
- a *function library* serving as an interface between the application code and the kernel
- a *library of services* that operate on classes of data objects to cause desired program behavior
- a set of *user-defined data* objects, representing the needs of the application,

# System Resource Management

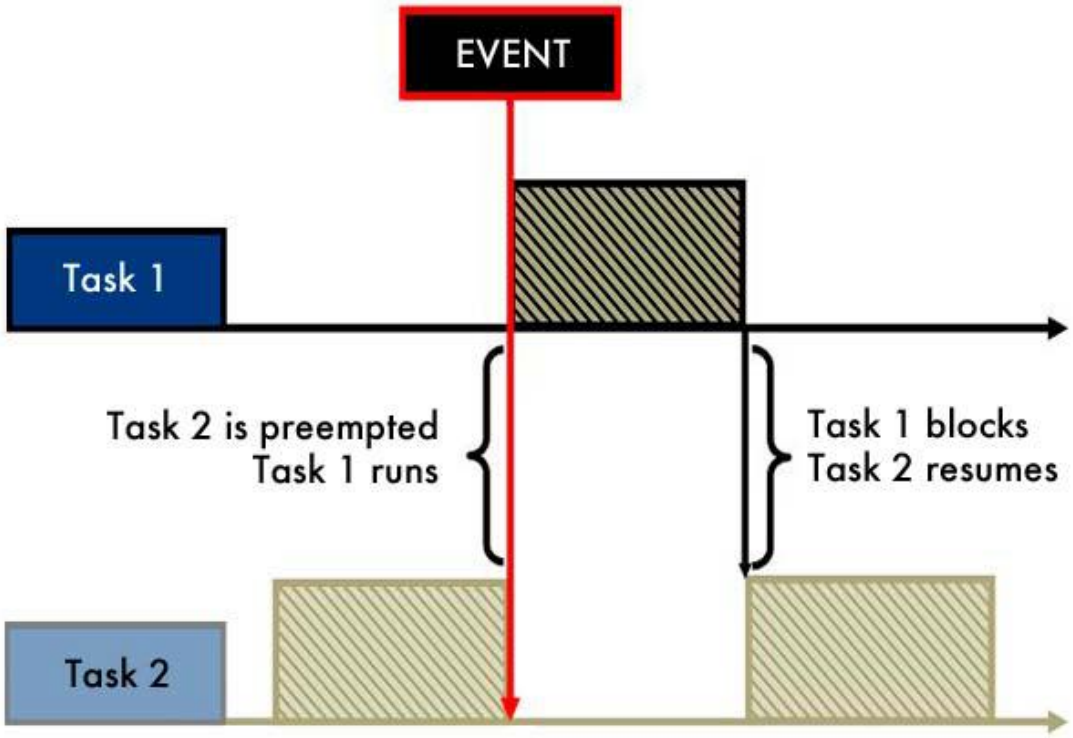
- System memory is a finite resource and therefore must be shared.
- Because the CPU operates much faster than the physical process it is controlling or monitoring, the CPU can be shared to prevent delays in processing. Such delays would violate a basic system policy.
- Time is the most difficult and unforgiving resource managed by the kernel.

# Multitasking

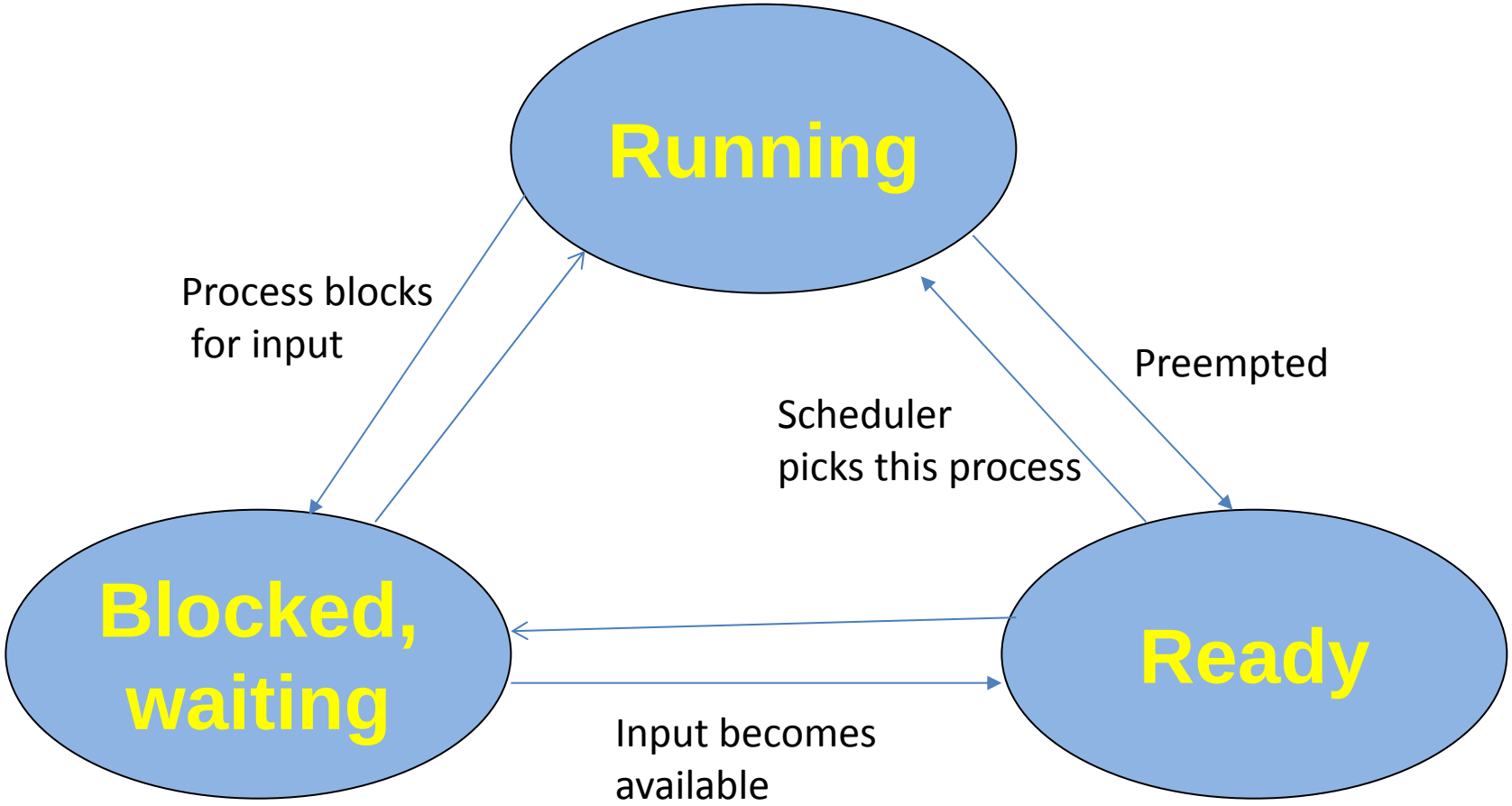


# Preemption

For example, tasks of low priority may have their execution **preempted** by a task of higher priority to permit a high priority task to perform a time-critical function.



# Scheduling state of process



# Cyclostatic scheduling (Time Division Multiple Access)

- Divided over equal-length timeslots over interval equal to hyperperiod H
- Hyperperiod is the least common multiple of the periods for all processes
- Utilization depends on
  - Numbers of timeslots used
  - Fraction of timeslots used for useful work

Example on white board

# Round robin scheduling

- Same hyperperiod as cyclostatic
- If a process do not have any useful work to do the scheduler moves on to the next process in order to fill the time slot with useful work

Example on white board

# Preemptive RTOS

- The most reliable way to meet timing constraints is to build a preemptive OS and to use priorities to control what process to run
- Priority Driven Scheduling
  - Rate-monotic scheduling (RMS)
  - Earliest deadline first (EDF)

- Example: Priority driven scheduling

# Rate-monotic scheduling (RMS)

- Static scheduling policy
  - All processes runs periodically on a single CPU
  - Context switching time is ignored
  - There are no data dependicies between processes
  - The execution time for a process is constant
  - All deadlines are at the end of their periods
  - The highest-priority is always selected for execution

Example

# Earliest deadline first (EDF)

- Dynamic scheduling policy
  - Changes process priorities dynamically during execution based on initiation times
  - Assigns process in order of deadline
  - Highest priority is the one whose deadline is nearest in time

- Implementation more complex than RMS

Example

# Real time lab exercise

16 givare (G01-G16) som känner när lok passerar
5 växlar (V06-V11) som kan styras
Lokets hastighet kan styras
Två lok skall kunna köras samtidigt enligt given bana utan att kollidera