Real-Time Systems

RTS10

Minor Embedded Systems

Week 3
Cooperative Scheduling

versd@hr.nl
brojz@hr.nl

Planning RTS10

Week 3: Cooperative Scheduling

Week 4: Pre-emptive Scheduling

Week 5: Using Free-RTOS + POSIX

Week 6: Schedulability Analyses, Priority Assignment

Overview

Scheduling

- Problem
- Goal
- Possible solution

Scheduling

Problem

- Multiple processes require CPU time
  - Some processes need it asap
  - Some processes just need to happen at some point in time
- Multiple processes require bandwidth
  - USB, Serial, SPI ....
  - Prioritization?

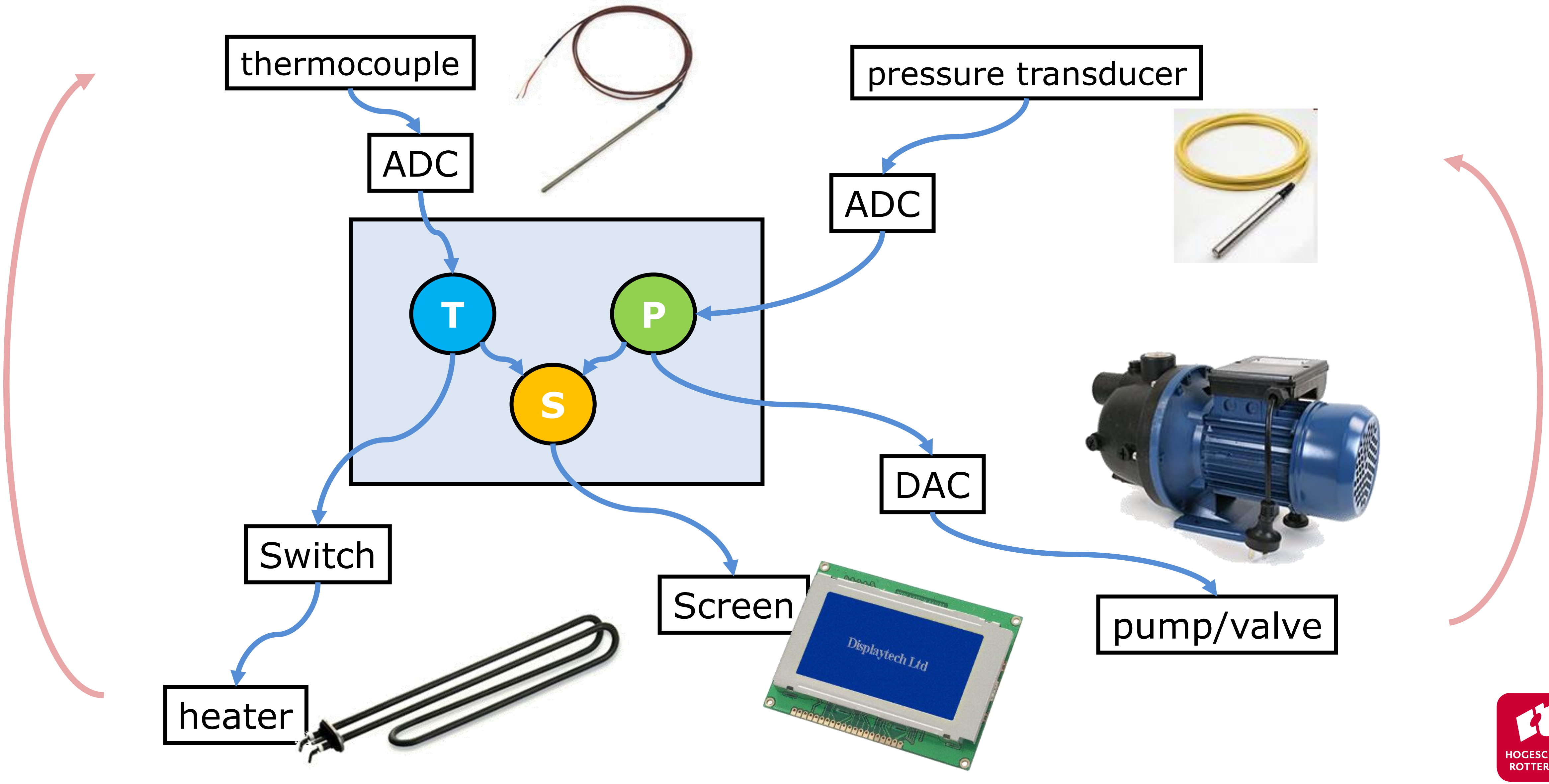
Goal

- Create a framework that'll ease (CPU) time management
- Easy to add new processes and to share resources

Cooperative Solution

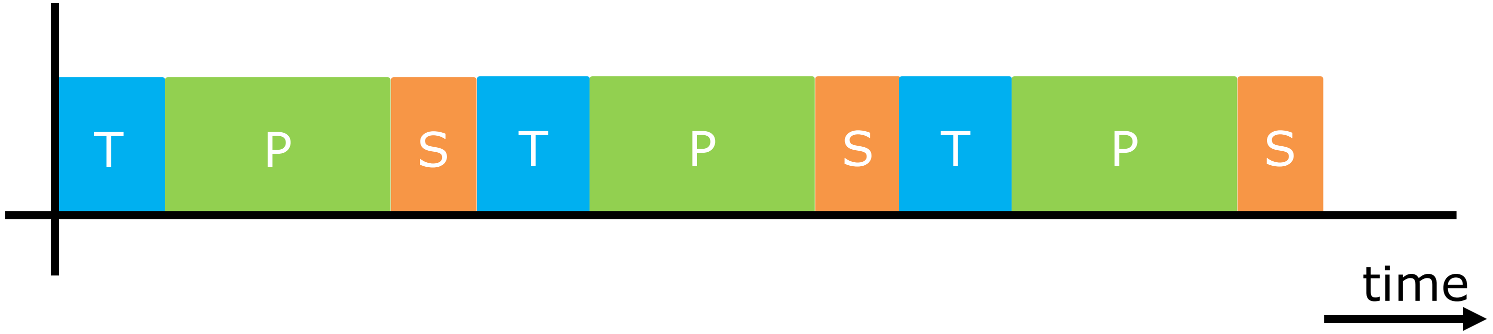
- Superloop ‘scheduler’
  - Systick
- Cooperative scheduler

Voorbeeld embedded system



What is the biggest flaw with this architecture?

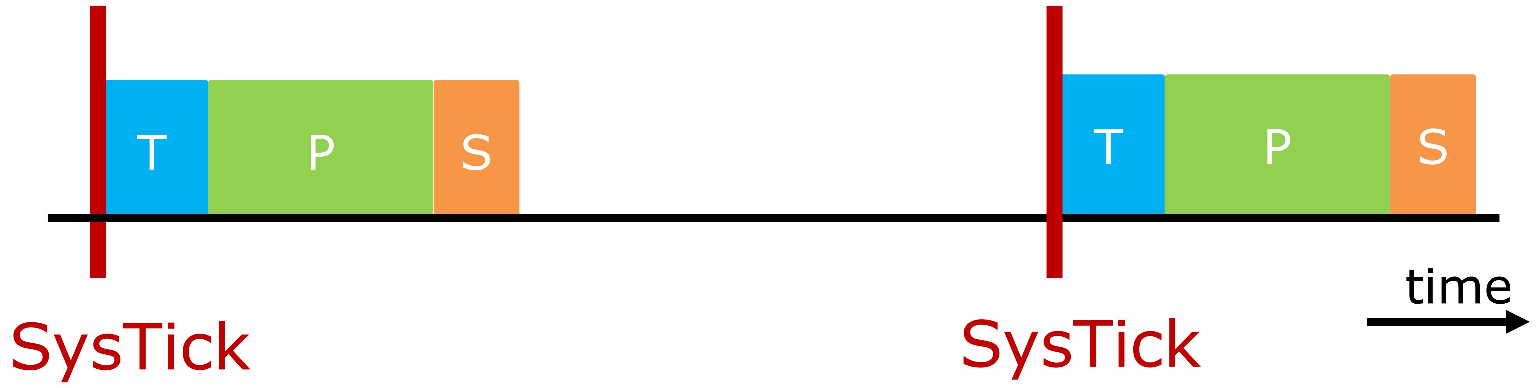
- NO control over time
  - Not deterministic
    - e.g. sample time of T and P depends on execution times
  - No response time promises
  - Wasting CPU cycles



T = temperature control
P = pressure control
S = screen update

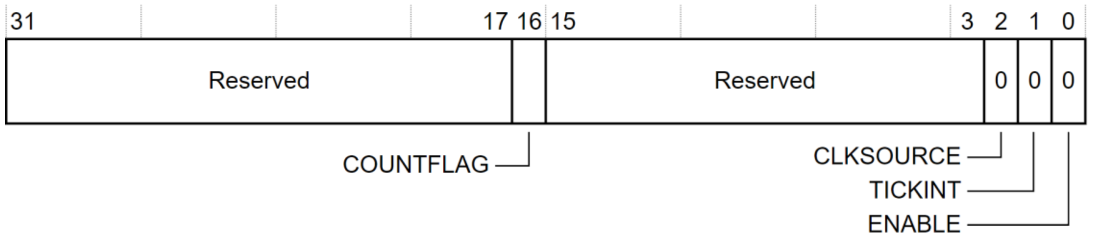
Superloop Construct

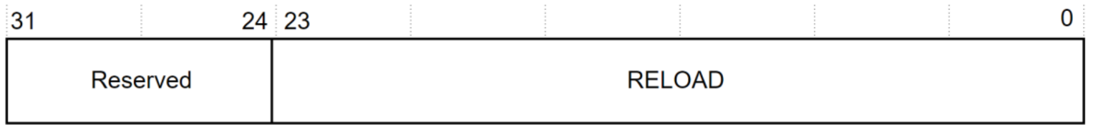
- Simple, deterministic
- Fixed time scheduling using SysTick
- Sleep until next SysTick (save energy)



Systick

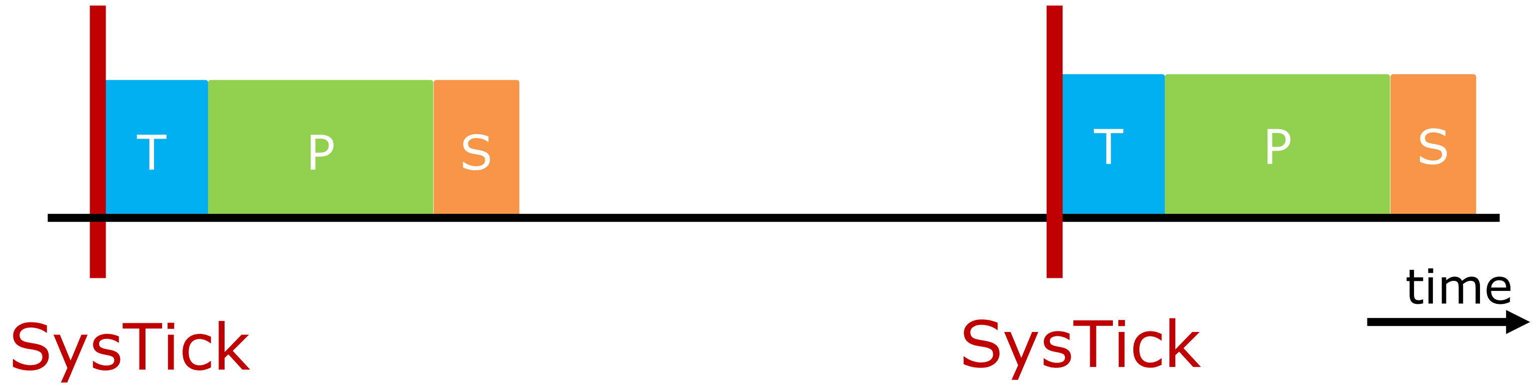
- Peripheral as part of the ARM CPU core
- Meant to be used by an OS
- Simple timer
  - 24 bits (16777216 ticks)
  - Own high priority interrupt
  - Has a ‘reload’ register





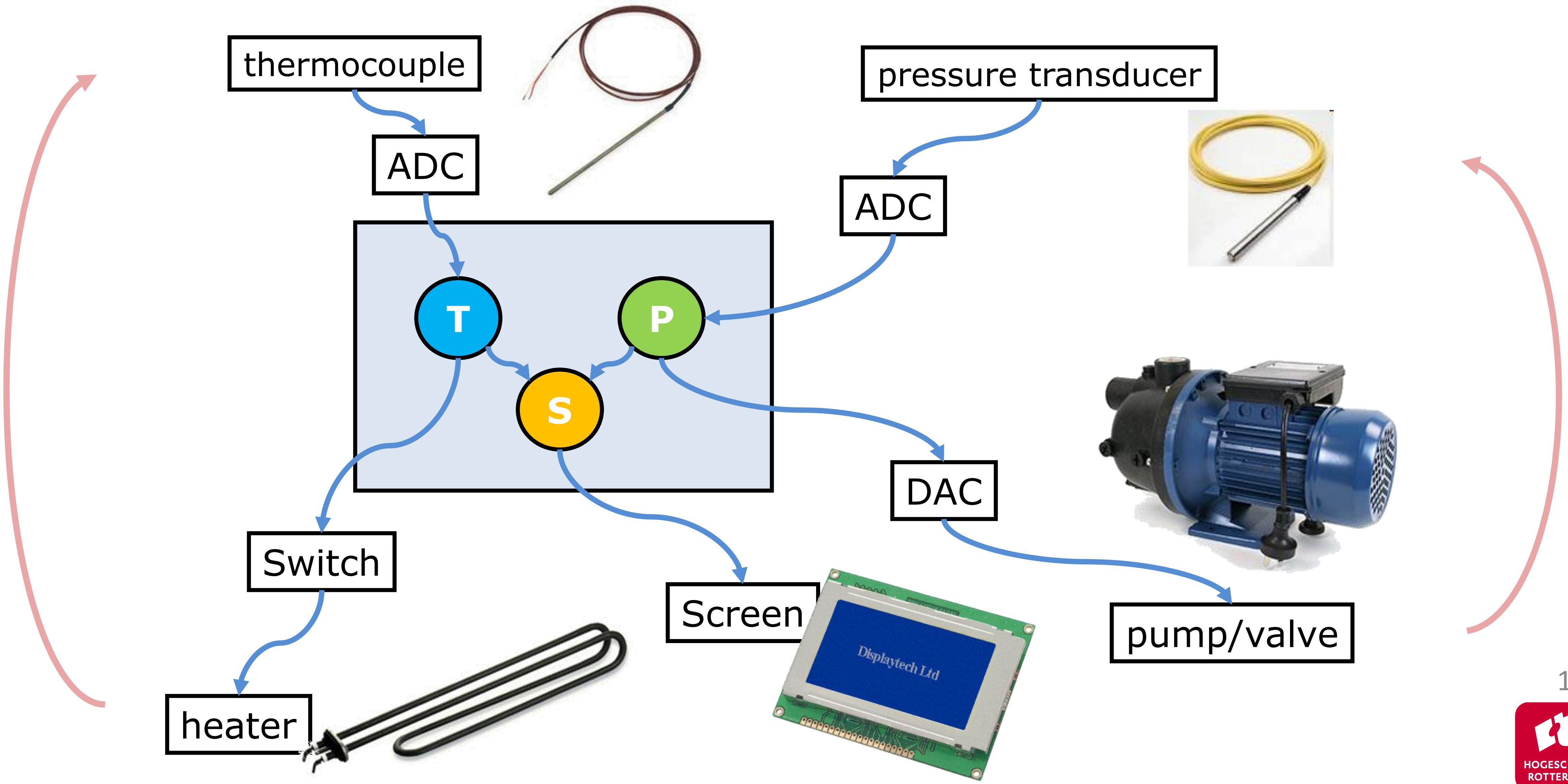
Superloop Construct

- Should we run every task every tick?



HOGESCHOOL ROTTERDAM

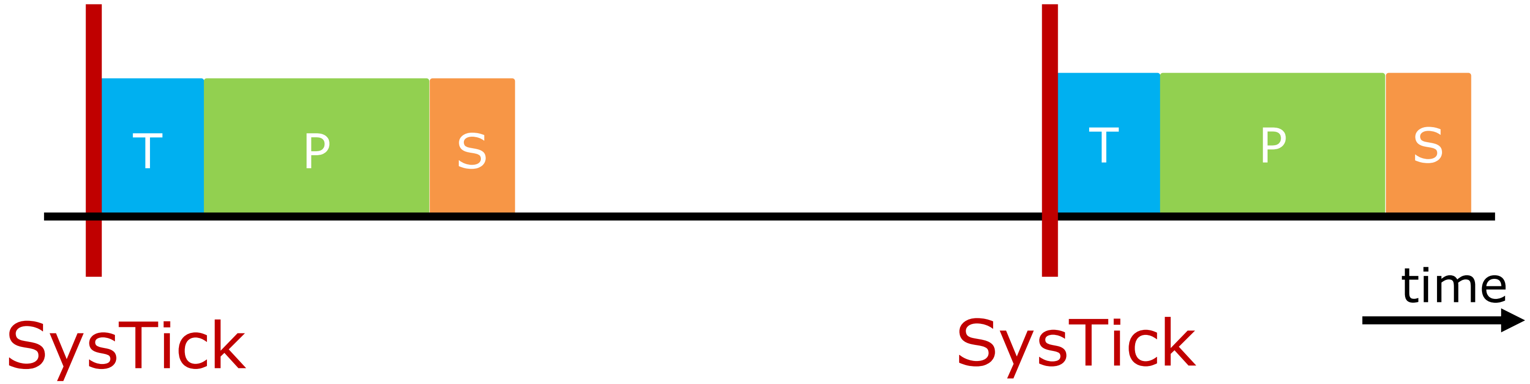
Voorbeeld embedded system



HOGESCHOOL ROTTERDAM

Superloop Construct

- Unnecessarily runs all tasks every tick
- $\sum_t time$ < SysTick time
  - Limited amount of tasks
  - Blocking tasks will cause problems: `while (buttonIsPressed())`



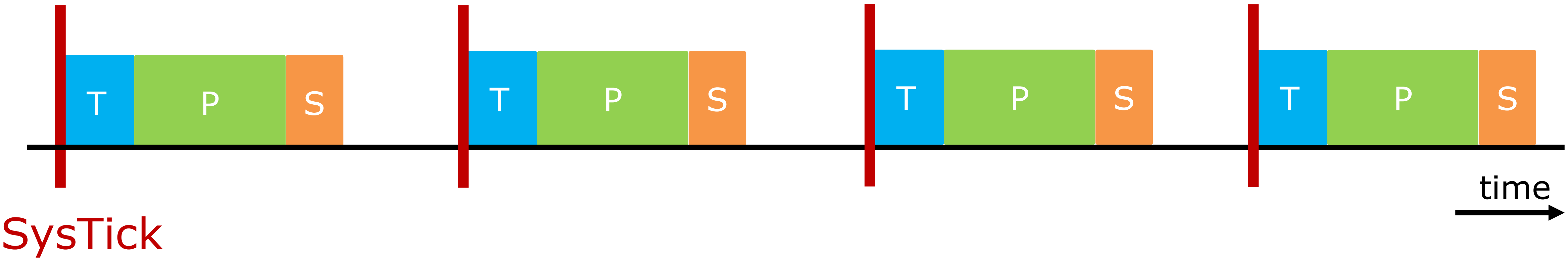
‘Complex’ Cooperative Scheduler

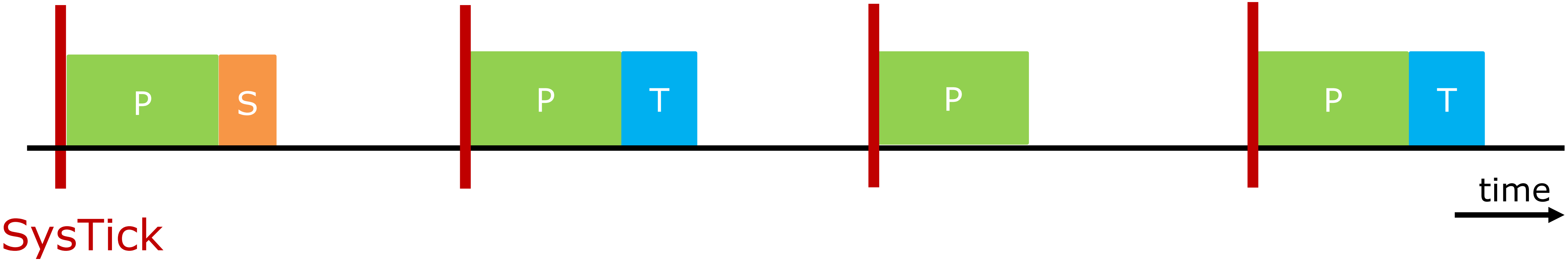
Cooperative:

- Task finishes
- then transfers control back to the scheduler

- No fights over concurrent use of hardware
- Perfect for small amount of tasks
- Easy to maintain
- Adaption to simple version:
  - Each task gets its own period (e.g. 400 SysTicks)
  - Each task could have a priority, state, etc

Difference





Overview

- Tasklist
  - Structure (`struct`) for each task
  - Ordered by priority
  - Only execute task when ready

- Use SysTick to determine which task is READY
- Use main loop to execute all ready tasks.
- Sleep until next SysTick

Interim : Function pointer

Syntax

```
void func(int);
void (*pointerNaarFunc)(int);
```

To run

```
pointerNaarFunc = &func;
(*pointerNaarFunc)(42);
```

To run (alternative)

```
pointerNaarFunc = func;
pointerNaarFunc(42);
```

Suspending a task

Implementing a delay using SysTick

- Change state to WAITING
- Initialize a counter, or add to the existing period
- Decrement the counter each tick
- When reached zero, put into ready mode

Scheduling

The process of selecting the task to execute next

- What if 3 tasks are READY at the same time?
- Which one will be selected first?

Scheduling algorithms

- FIFO (Round robin)
- Priority based

FIFO – Scheduling Algorithm

Tasks are run in order of task-creation

- Add most important tasks first
- Add less important tasks later

Pro

- Easy!
- No overhead in selecting

Con

- Fixed solution, pre-determined at compile time
- Tasks created run-time are always last

Priority based – Scheduling Algorithm

Use ‘priority’ number over position in task list

- Highest priority task goes first of all READY tasks

Pro

- Ability to work with more tasks
- Possible to change priority real time
- Most demanding tasks run first

Cons

- Means either sorting a list or traversing it
  - Increases scheduling time

Next week

Pre-emptive scheduling