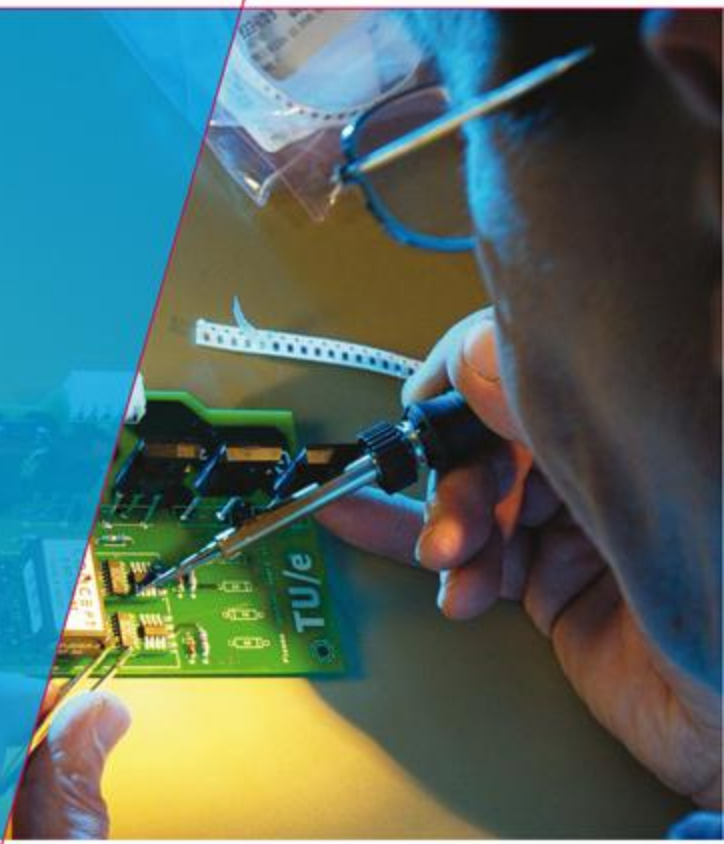
# Tasks

Ruxandra Bobiti
Royce Luo
George Rascanu
Alexandru Pustianu
Tsvetan Baliovsky




TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

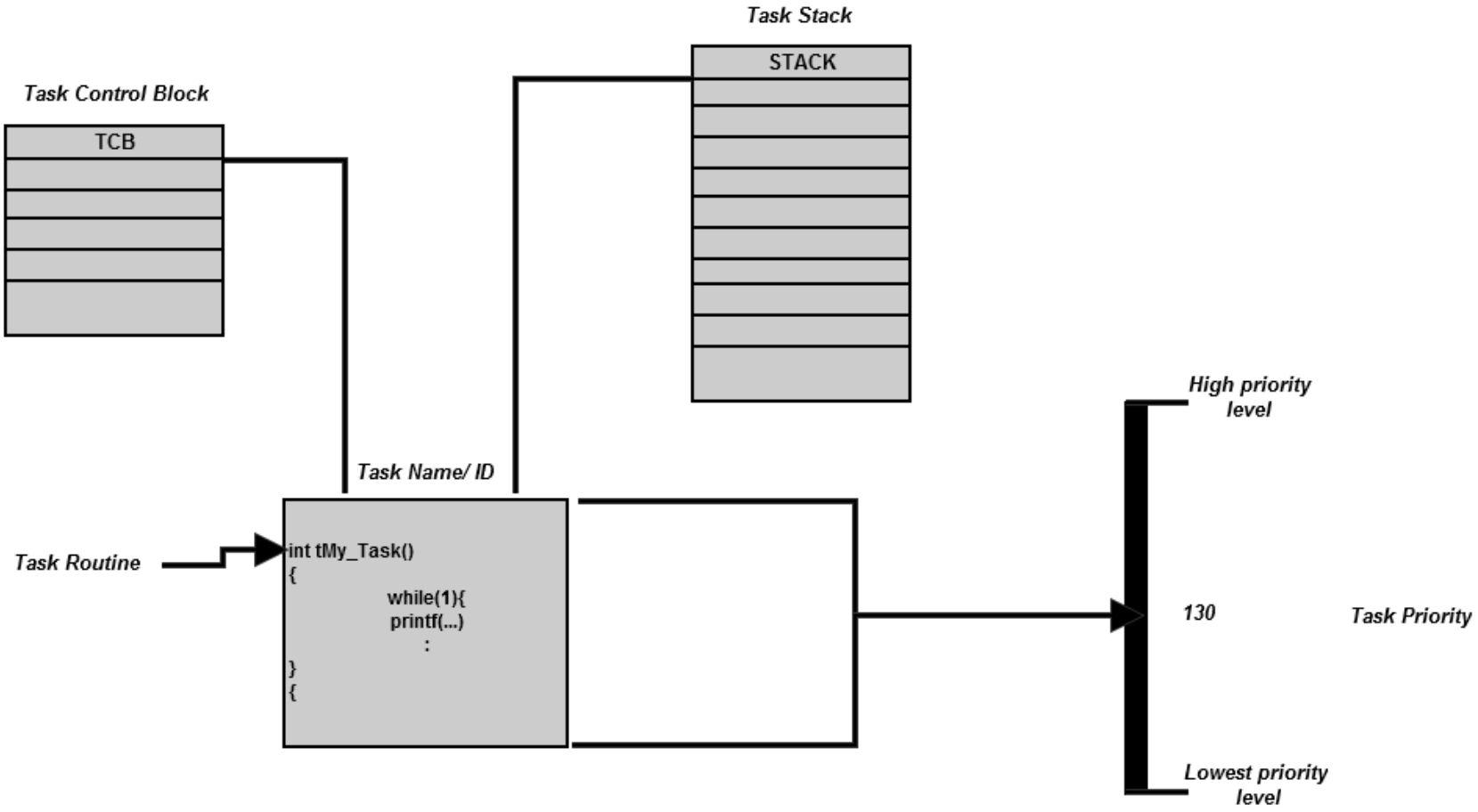
# Topics

- Task definition
- Task states and scheduling
- Typical task operations
- Typical task structure
- Task coordination and concurrency
- Tasks in ROS

# Task definition

- **Independent thread of execution**
- **Main property – schedulable**
- **Defined by:**
  - **Supporting data structures (Task stack, TCB, task routine)**
  - **Set of parameters (Name, ID, Task priority)**

# Task object


Task Stack
STACK
Task Control Block
TCB
Task Name/ ID
Task Routine
int tMy_Task()
{
while(1){
printf(...)
:
}
{
High priority level
130
Task Priority
Lowest priority level

# System tasks

- Initialization/startup task
- Idle task
- Logging task
- Exception-handling task
- Debug agent tasks

- Reserved priority levels for system tasks.

# Task states and Scheduling

- **Ready state**
- **Blocked state**
- **Running state**

**Ready state**

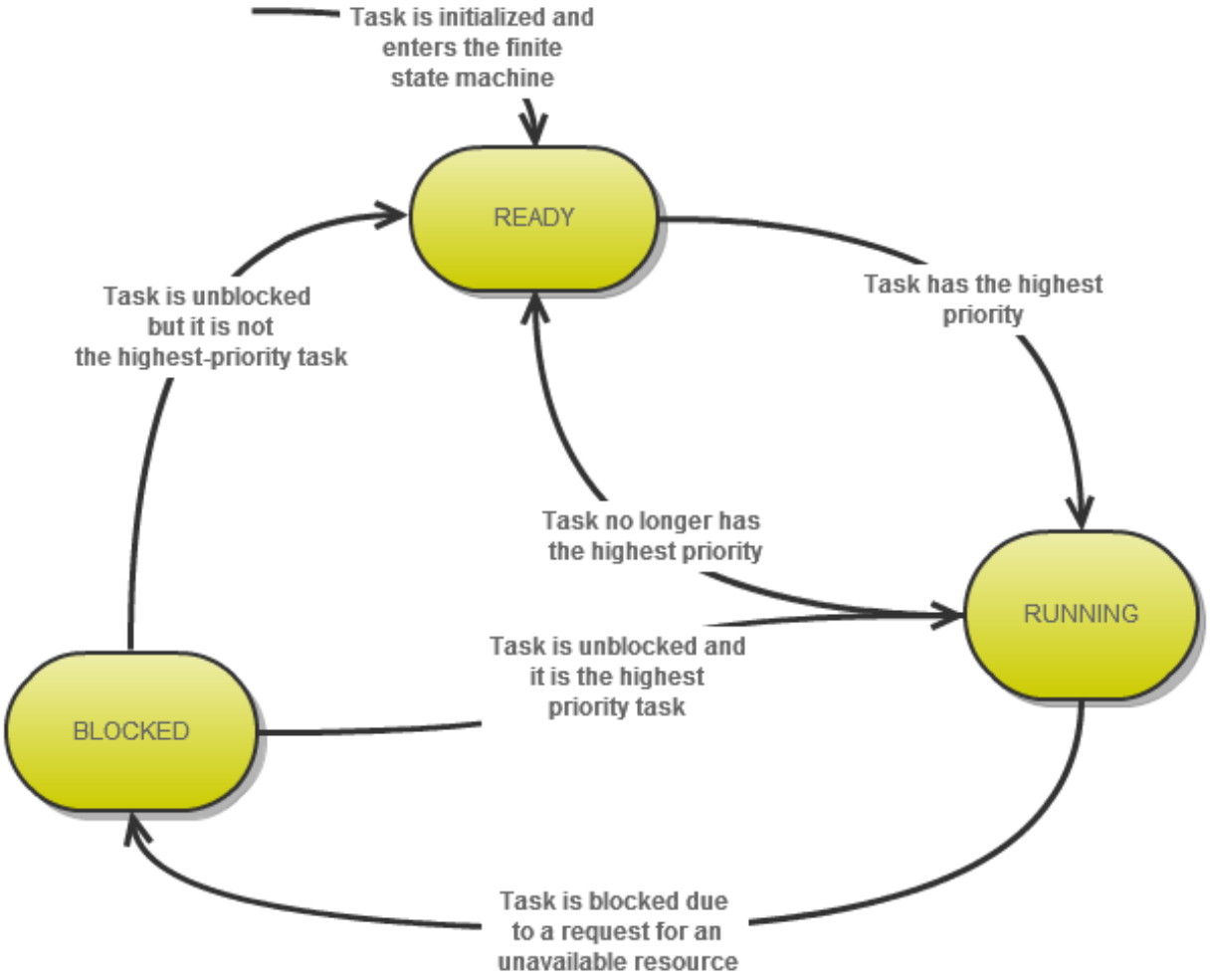
**Pended state**

**Delayed state**

**Running state**

**Suspended state**

# Task states and Scheduling



# Task states and Scheduling

## State of Task-Ready list

| | | | | | |
|---|---|---|---|---|---|
| | Task 1<br>Priority=70 | Task 2<br>Priority=80 | Task 3<br>Priority=80 | Task 4<br>Priority=80 | Task 5<br>Priority=90 |
| Task 1 executed | | | | | |
| | Task 2<br>Priority=80 | Task 3<br>Priority=80 | Task 4<br>Priority=80 | Task 5<br>Priority=90 | |
| Task 1 blocked, Task 2 executed | | | | | |
| | Task 3<br>Priority=80 | Task 4<br>Priority=80 | Task 5<br>Priority=90 | | |
| Task 2 blocked, Task executed | | | | | |
| | Task 4<br>Priority=80 | Task 5<br>Priority=90 | | | |
| Task 2 unblocked | | | | | |
| | Task 4<br>Priority=80 | Task 2<br>Priority=80 | Task 5<br>Priority=90 | | |
| Task 1 unblocked | | | | | |
| | Task 4<br>Priority=80 | Task 2<br>Priority=80 | Task 3<br>Priority=80 | Task 5<br>Priority=90 | |

# Task blocked

- A call to request an unavailable resource
- A call to request to wait for an event to occur
- A call to delay the task

# Task unblocked

- **A semaphore token**
- **A waiting message arrives**
- **A time delay imposed**

# Task Creation and Deletion

- **Fundamental operations:**

- **Create**

**Why?**

- **Useful for debugging.**
- **Special initialization needs to occur between the times that a task is created and started.**

# Fundamental operations:

- **Delete**

**Why?**

- **Limited memory !**

**Be careful!**

- **Deleting one task can result in memory or resources leaks.**

# Task Scheduling

- done automatic by the kernel
- manual scheduling for developers using an API

**Why we need task scheduling ?**

**Debugging and avoiding deadlocks or starvation in the scheduler**

- Fundamental operations:
  - Suspend
  - Resume
  - Delay
  - Restart
  - Get priority
  - Set priority
  - Preemption lock
  - Preemption unlock

# Obtaining Task information

**Information useful for debugging and monitoring**

- **Get ID - Get the current task s ID**
- **Get TCB - Get the current task s TCB**

# Typical Task Structure

- run to completion
- endless loop.

# Run-to-Completion Tasks

```
RunToCompletionTask ()
{
   Initialize application
   Create endless loop tasks'
   Create kernel objects
   Delete or suspend this task
}
```

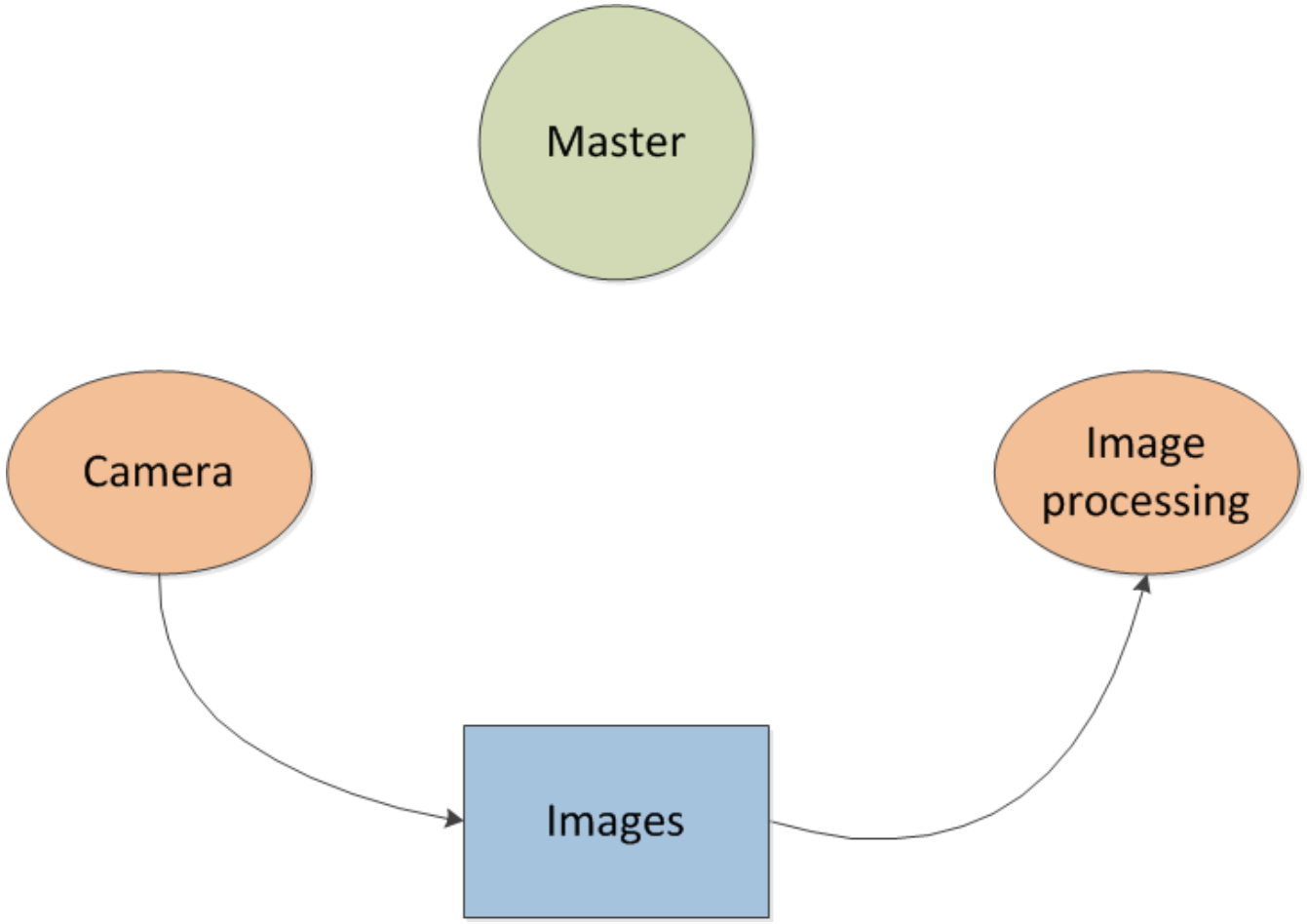
# Endless-Loop Tasks

```
EndlessLoopTask ()
{
    Initialization code
    Loop Forever
   {
    Body of loop
    Make one or more blocking calls
    }
}
```

# Task implementation in ROS

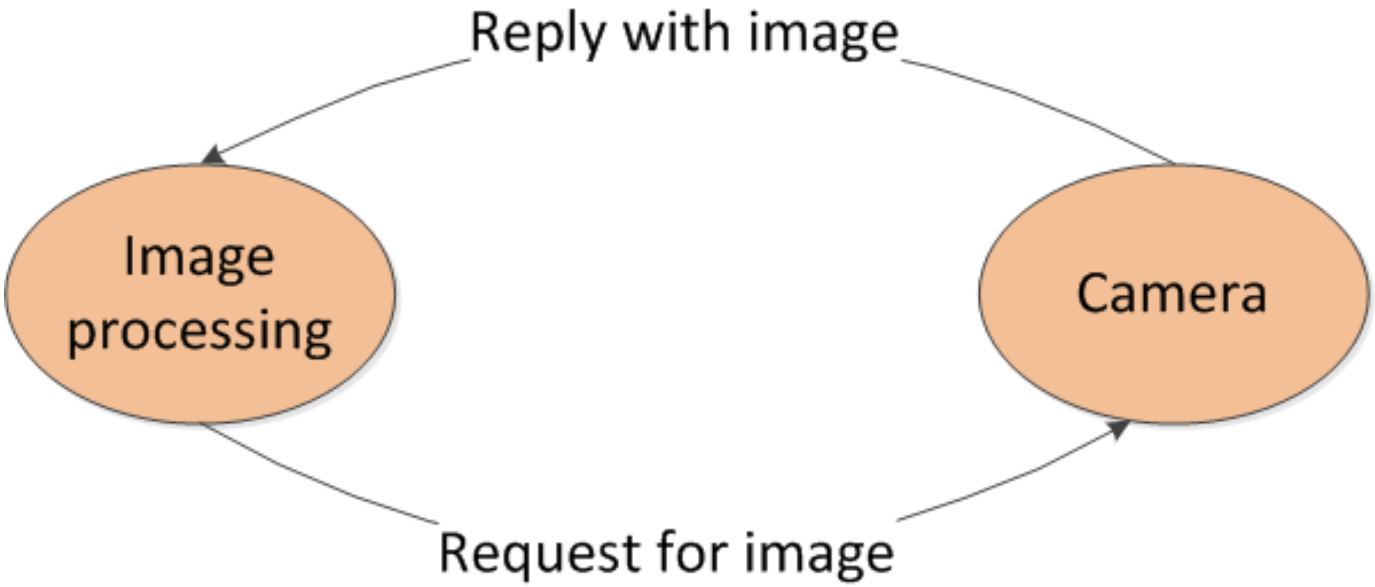
- ***Nodes*- processes that perform computation**
- **Defining a node and communication**
  - **Peer-to-peer communication**
    - **One-to-one**
    - **One-to-many**
    - **Many-to-many**
    - **Request-reply**
    - **Request-reply with feedback**
- **Scheduling**
  - **Possibilities for real-time control**
- **Node operations**

# Defining a node



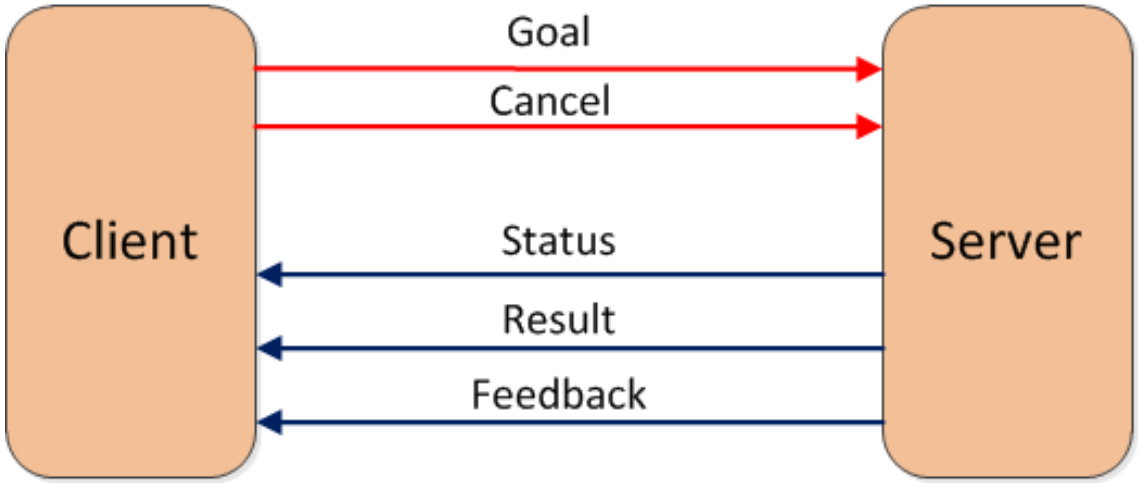
# Request-reply communication

- A way for synchronous communication



# Preemptible Request-reply

- For time consuming services
- Allows preemption of required services->Preemption results in cancellation of the previous goal
- Provides feedback of the current status and results



# Benefits of the peer-to-peer

- In distributed systems

802.11x

Offboard computation

Onboard computation

# Real-time possibilites

- PR2 Controller manager
  - Hard real-time loop at 1000Hz
  - Calls and schedules controllers