PROBLEM AREAS

Software

Hardware

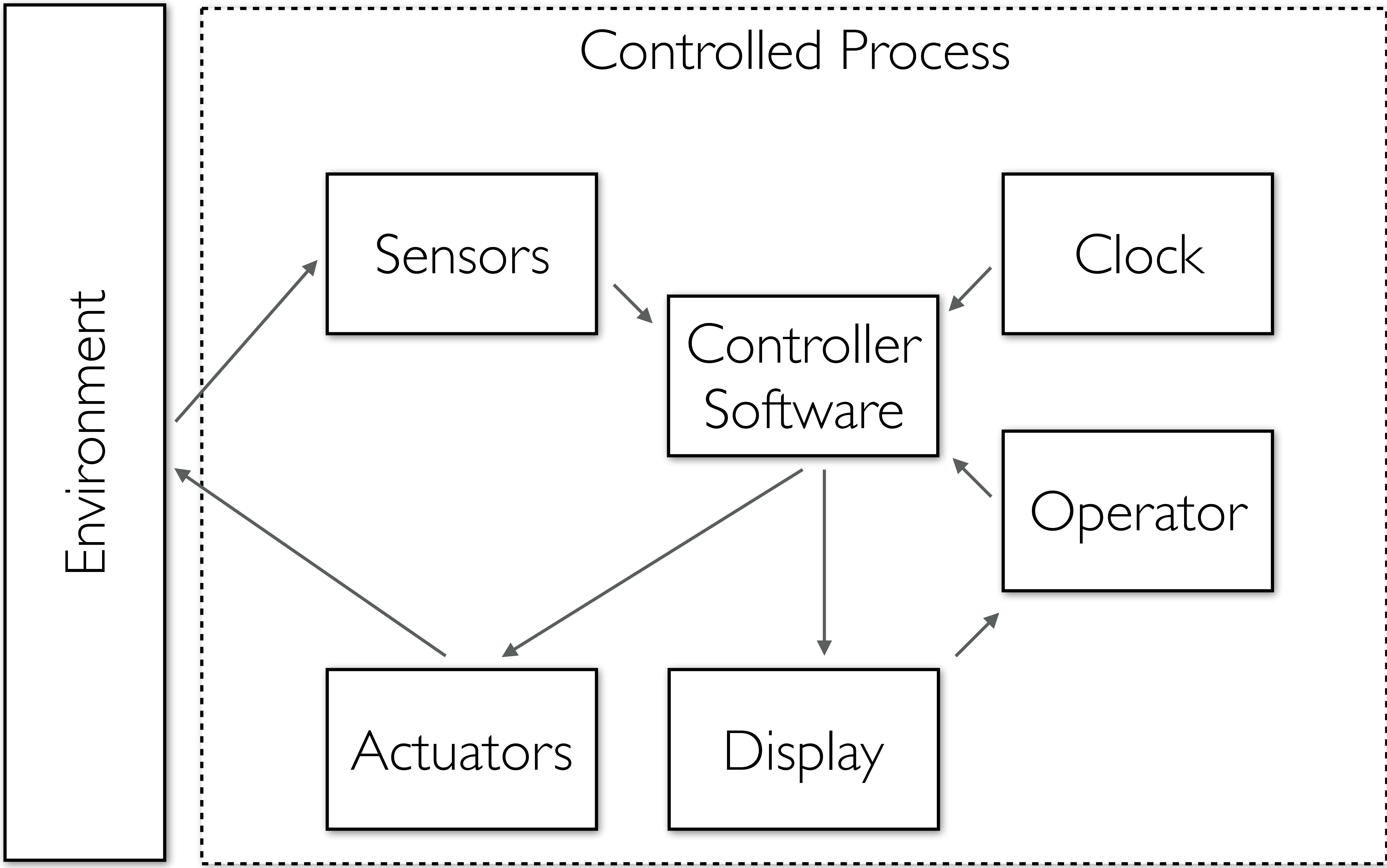
Non-functional properties

EMBEDDED SOFTWARE

- Embedded software interacts with the physical world
  - Often written by domain experts, not computer scientists
  - Timeliness becomes more important (e.g. deadlines)
  - Concurrency becomes more important (e.g. physical events)
  - Liveness becomes crucial (e.g. deadlock prevention)
  - Heterogeneity becomes default
  - In general: Predictable behavior, from nice-to-have to mission-critical

http://ptolemy.eecs.berkeley.edu/publications/papers/02/embsoft/embsoftwre.pdf

TYPICAL STRUCTURE



STARTING POINT

- Understanding of resources
  - Describe resources available to the application (CPU, memory, OS)
  - Driven by cost factors and environmental conditions
- Understanding of algorithms
  - Which resources will be used in which way
  - Relevant resulting performance metrics
- Understanding of workload
  - Must consider control and data dependencies
  - Driven by environmental conditions
  - Describe tasks to be handled + **timeliness constraints**

TIMELINESS

- Embedded systems are often real-time systems
- Hard real-time systems are often embedded systems

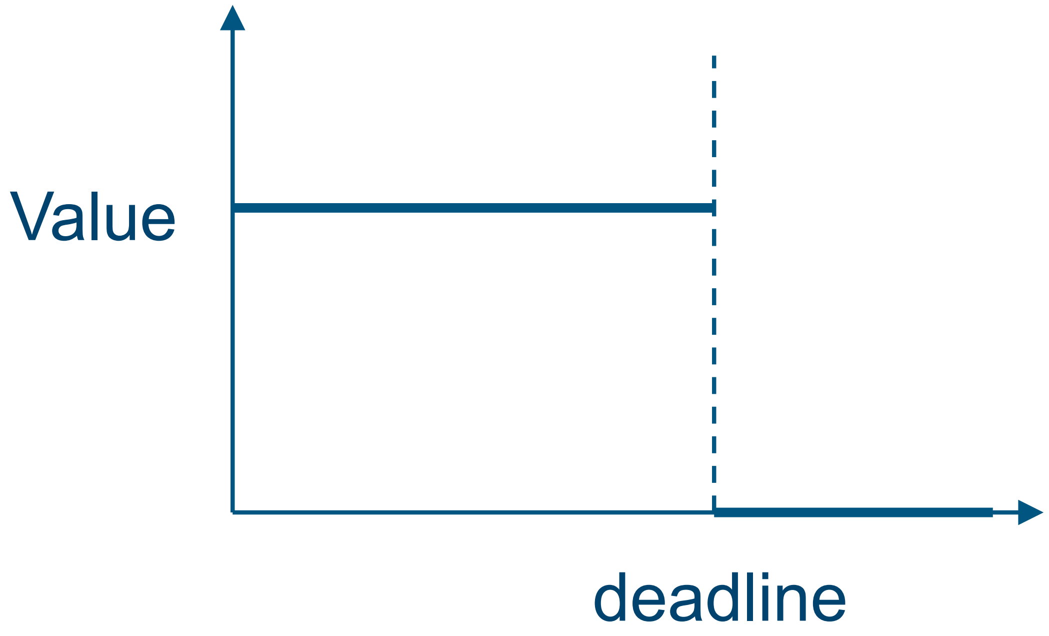
"A real-time system is one in which the ***correctness*** *of the computations not only depends on the logical correctness of the computation, but also on the* ***time at which the result is produced (deadline)****. If the timing constraints of the system are not met, system failure is said to have occurred."*

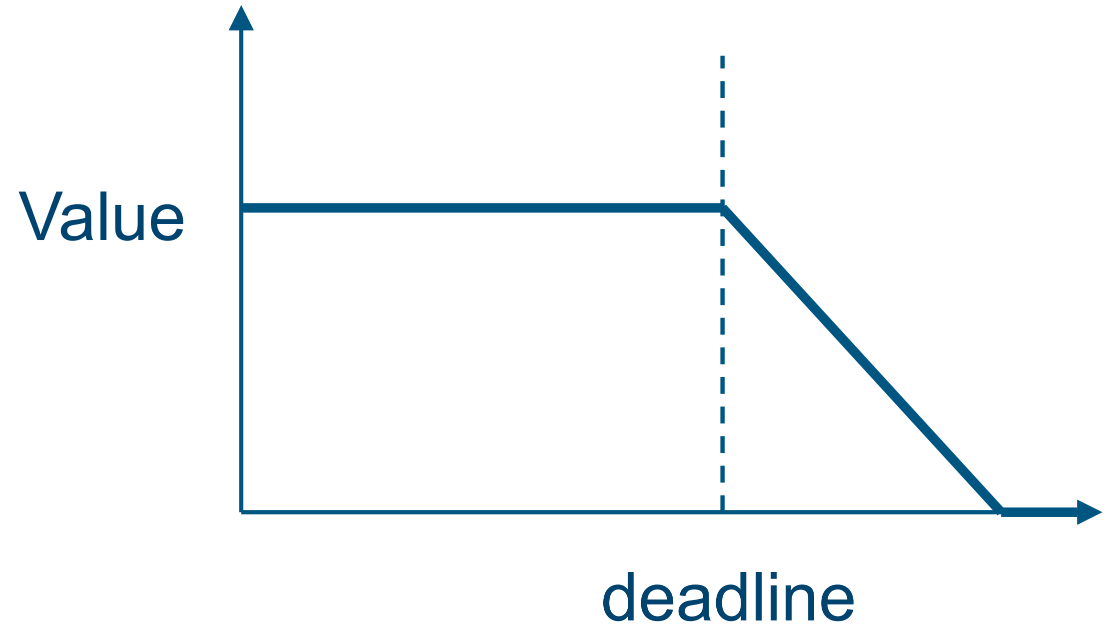
- Autopilot in airplane vs. YouTube video player
  - Position calculation vs. 30 images / s
  - Do all tasks have to be executed before their deadline ?
  - How to deal with missed deadlines ?
  - When is the result produced ?

REAL-TIME

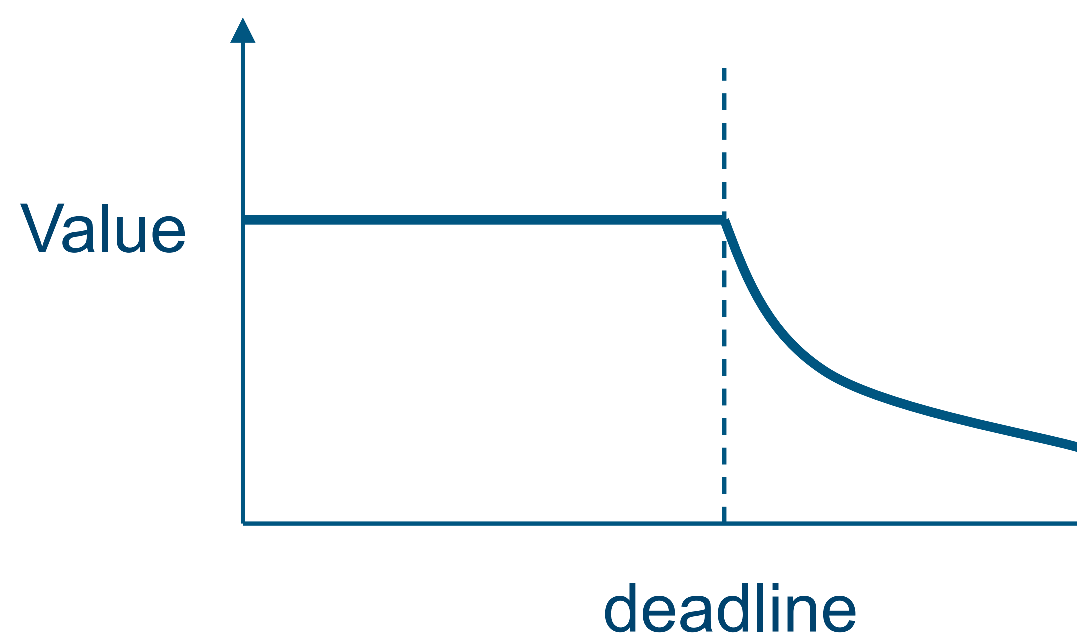
- **Hard real-time**: Missing a deadline is not acceptable
  - Aircraft control systems
  - Nuclear power / chemical plant safety mechanisms
  - Medical devices
- **Soft real-time**: Missing a deadline is undesirable
  - Multimedia
  - Airline reservation
  - High-speed trading applications
- Real-time objectives may change during operation
  - Example: Grounded airplane vs. flying airplane

TASK / VALUE FUNCTIONS

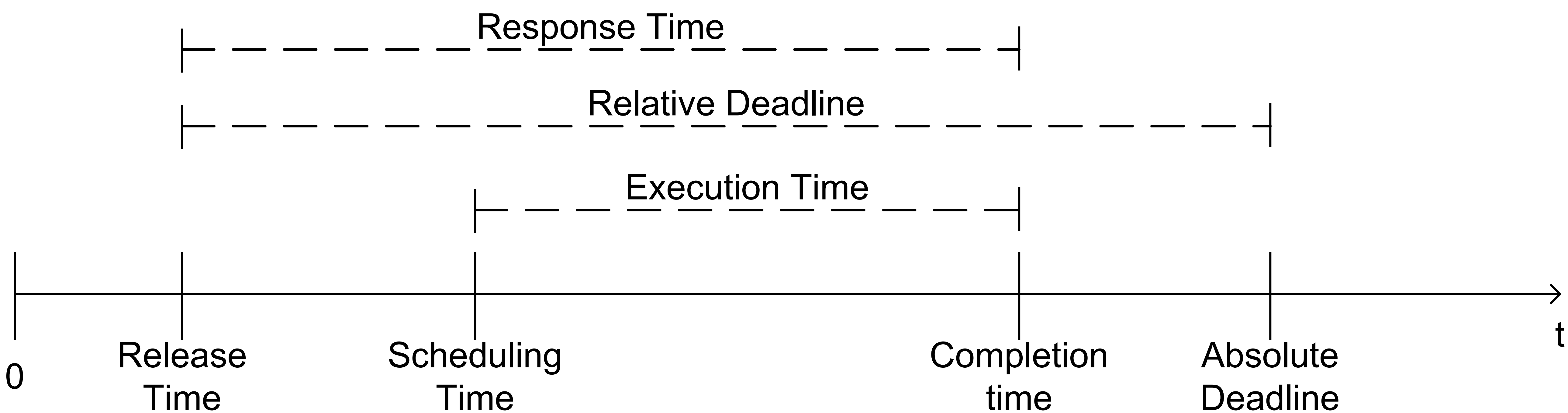




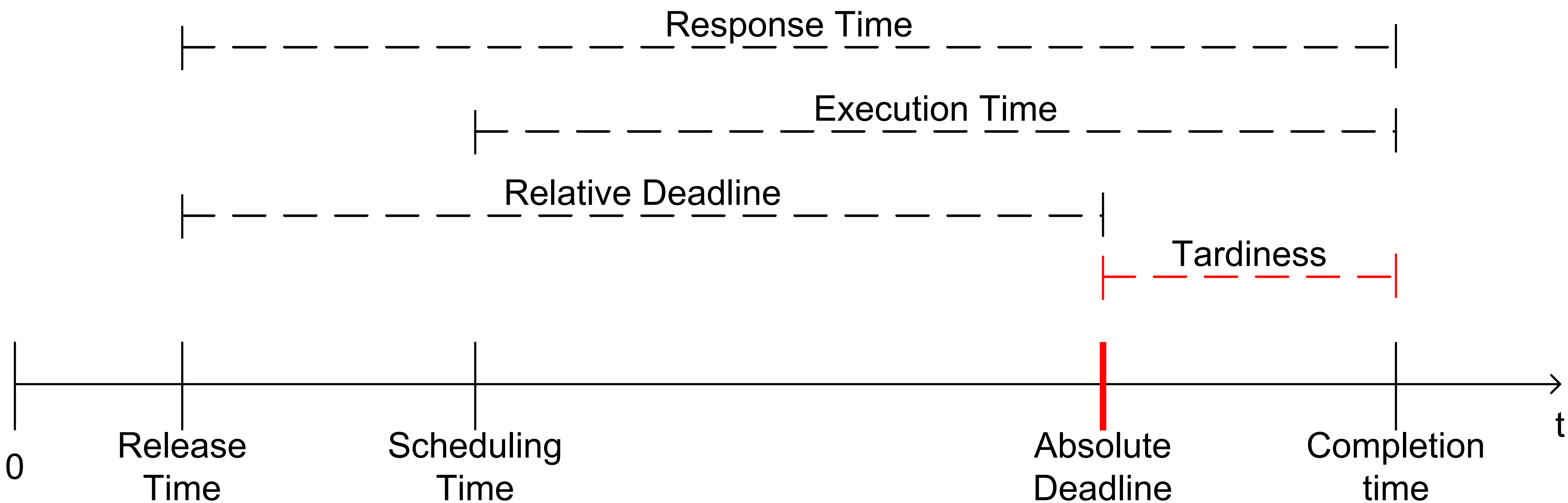
- Deadline missed
  - Hard real-time: Task result has no more value
  - Soft real-time: Task result has reduced value



HARD REAL-TIME



SOFT REAL-TIME



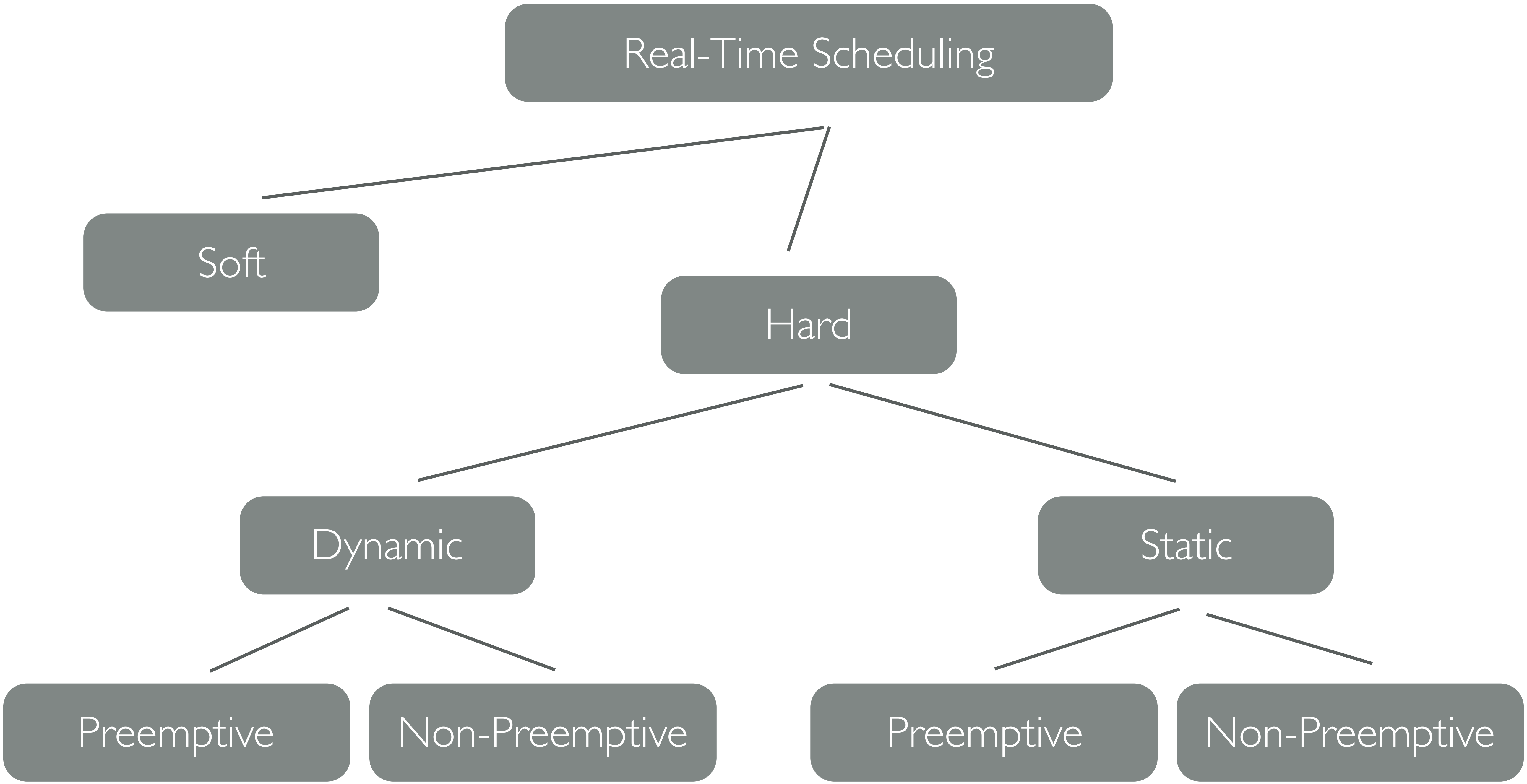
REAL-TIME TASKS

- **Periodic tasks**
  - Examples: Sensor data acquisition, action planning, system monitoring
  - Must be regularly activated (once per **period**)
- **Aperiodic tasks**
  - Example: Background services, logging, operator requests
  - Triggered by well-known event at any time
- **Sporadic tasks**
  - Example: Collision detection in a roboter, I/O device interrupt
  - Aperiodic task with **minimum inter-arrival time** (rate restriction)

REAL-TIME SCHEDULING

- **Scheduling**: Define order of task execution
  - Mature theory for real-time schedules on uniprocessors since 1970's
  - Theory for real-time multiprocessor schedules still under research
- On small embedded systems (micro-controller scale)
  - Only one / a few tasks
  - ‚Manual' scheduling by developer good enough
- On larger embedded systems
  - **Real-time operating system**
  - Implements appropriate scheduling concepts
  - Supports prioritization and synchronization of concurrent tasks

REAL-TIME SCHEDULING



Operating Systems Group

Professorship

Research

Teaching
- Distributed Operating Systems
- Design of Software for Embedded systems
- Dependable Systems
- Seminar "Operating Systems"

Internal

OPAL

stack overflow (Q+A on programming)

Real-Time Systems

Course **565030**

News

- No news.

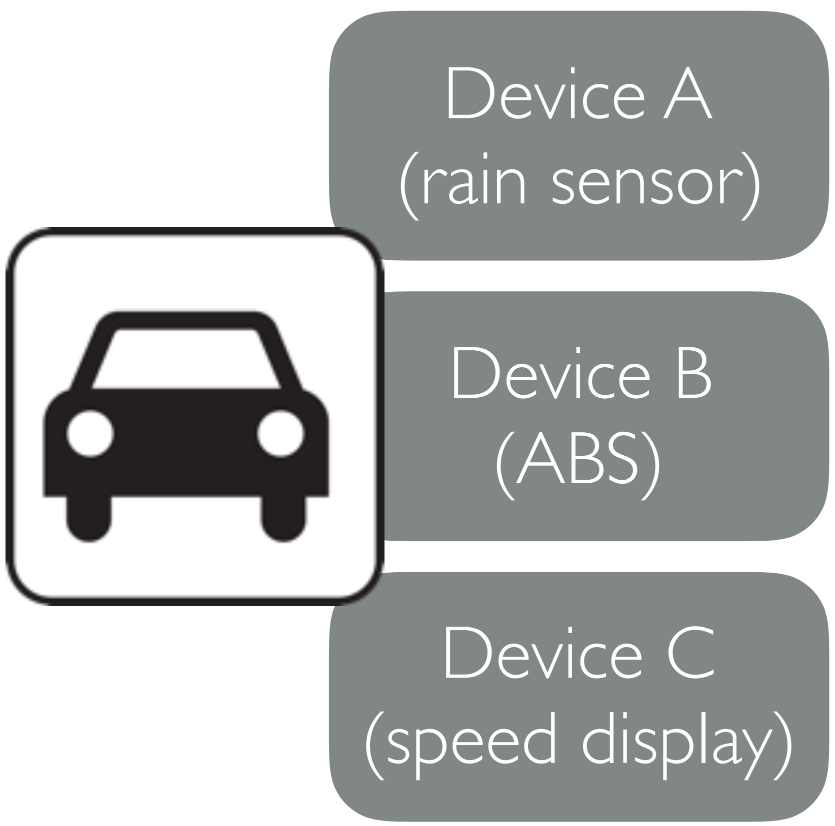
Course Information

Synopsis | Prerequisites | Time and Rooms | Material | Tutorial | Exam

This course introduce concepts and design of real-time systems.

Topics include real-time requirements, scheduling, resource conflicts, real-time communication, and soft real time.

REAL-TIME SCHEDULING




```
void main(void)
{
   while(TRUE)
   {
        if(device A needs service)
                // Handle A -> 5ms
        if(device B needs service)
                // Handle B -> 5ms
        if(device C needs service)
                // Handle C -> 2ms
                              must be serviced all 8 ms
   }
}
```

- ‚Manual scheduling'
  - Simple round-robin implementation, based on **polling**
  - Device C needs periodic attention, A and C are purely event-driven

REAL-TIME SCHEDULING

```
bool handleA,handleB;
void interrupt HandleDeviceA(){
   handleA = true;
}
void interrupt HandleDeviceB(){
   handleB = true;
}
void main(void)
{
   while(true)
   {
        if(handleA){
            handleA = false;
            // handle A }
        if(handleB){
            handleB = false;
            // handle B }
   }
}
```

- ‚Manual scheduling' with round robin with **interrupts**

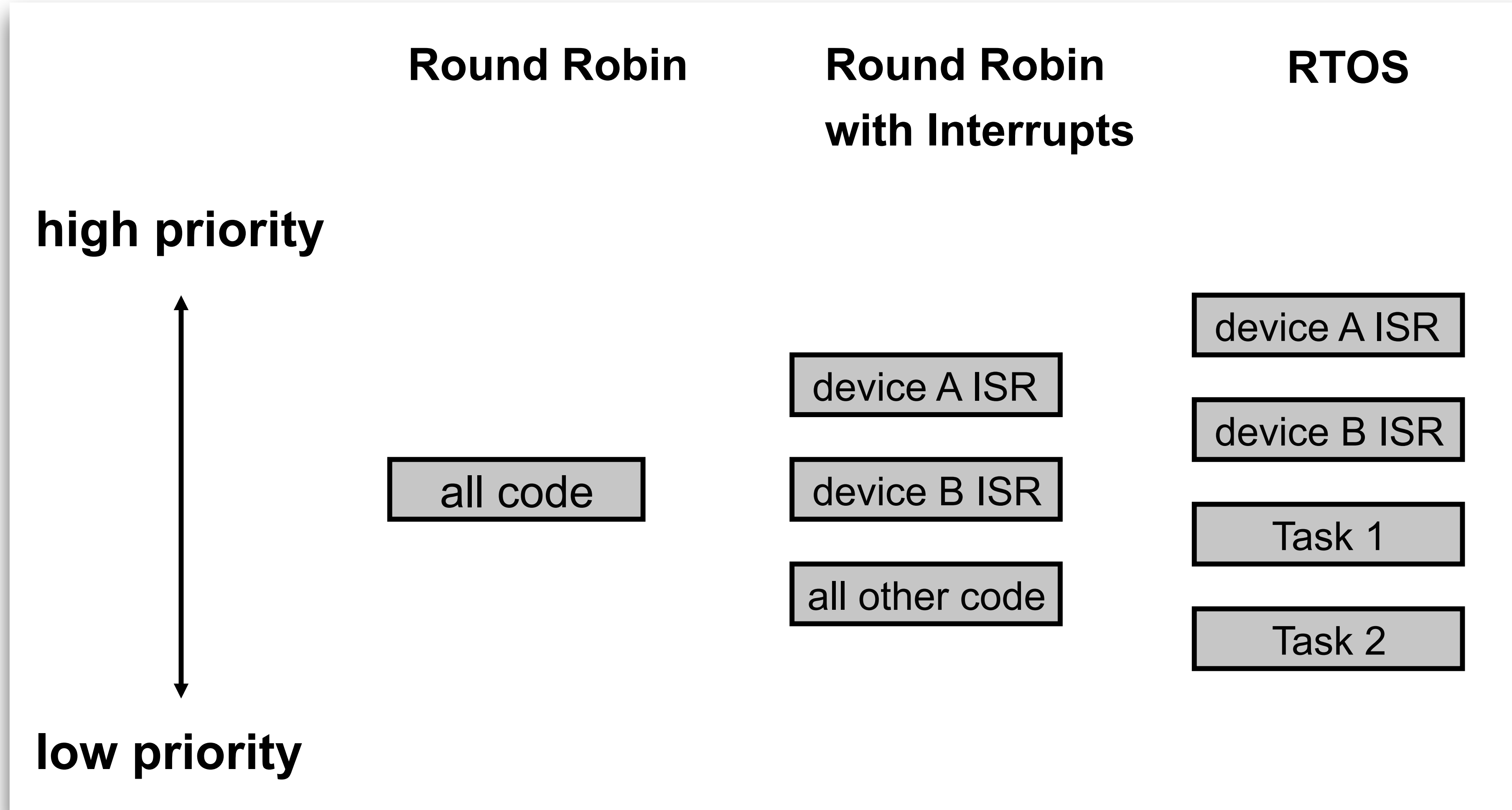
```
void interrupt HandleDeviceA(){
   // set signal X
}
void interrupt HandleDeviceB(){
   // set signal Y
}

void Task1(){
   while(true){
        // wait for signal X
        // handle device A
   }
}

void Task2(){
       while(true){
              // wait for signal Y
              // handle device B
       }
}
```

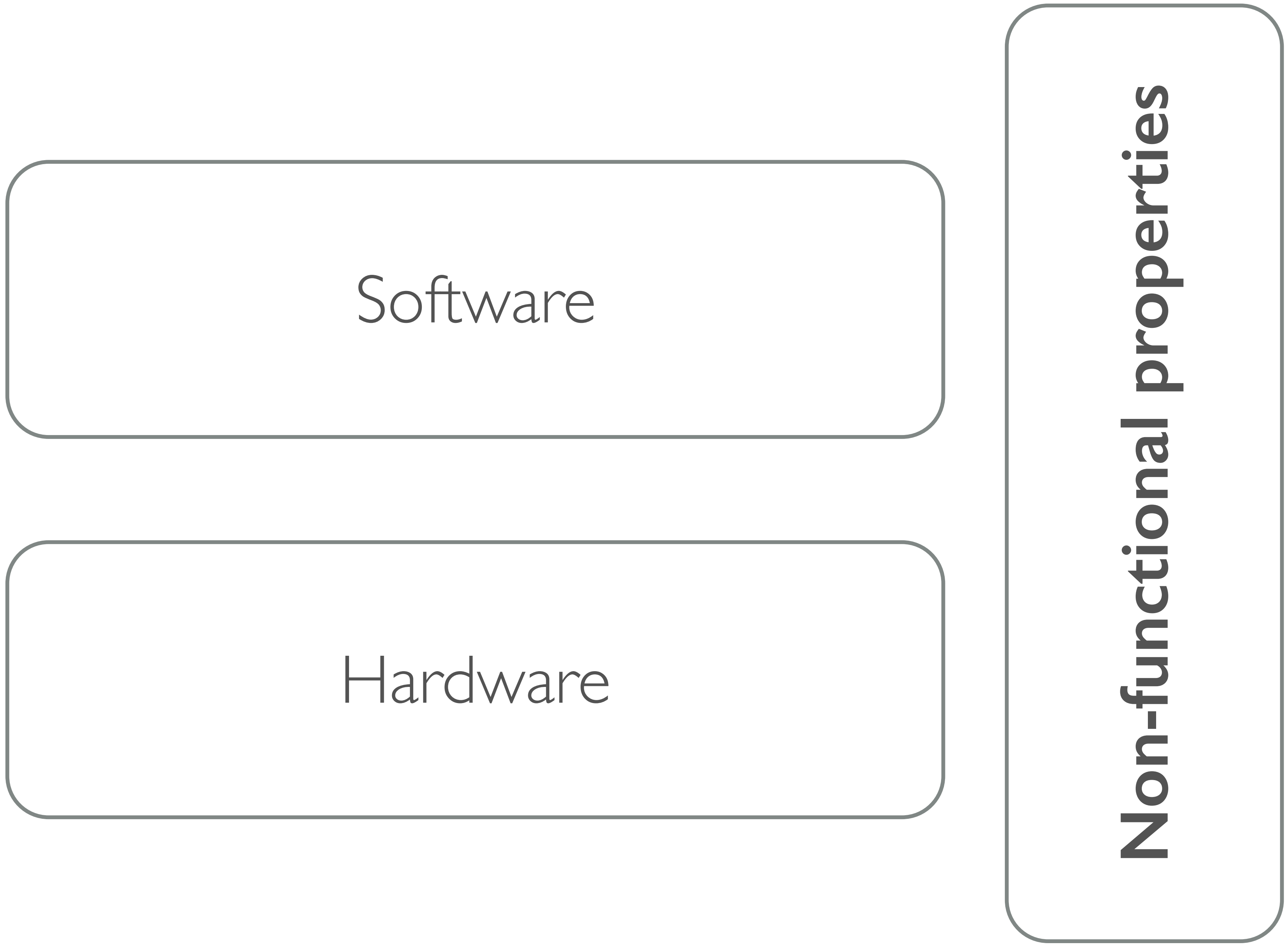
- With support from **real-time operating system**

REAL-TIME SCHEDULING



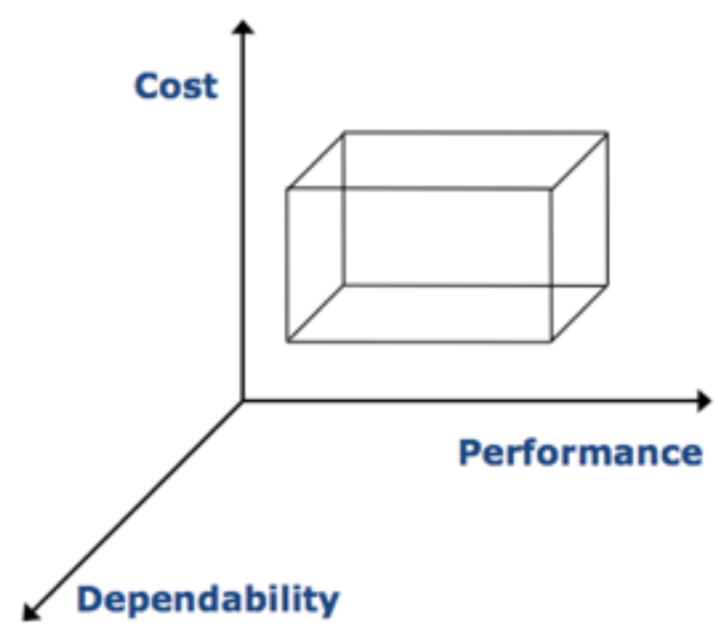
- Real-Time Operating System (RTOS) features
  - Real-time scheduling with priorities
  - Support for concurrency, preemption and prioritization
  - Predictable timing behavior of interrupt routines and system calls

PROBLEM AREAS



DEPENDABILITY

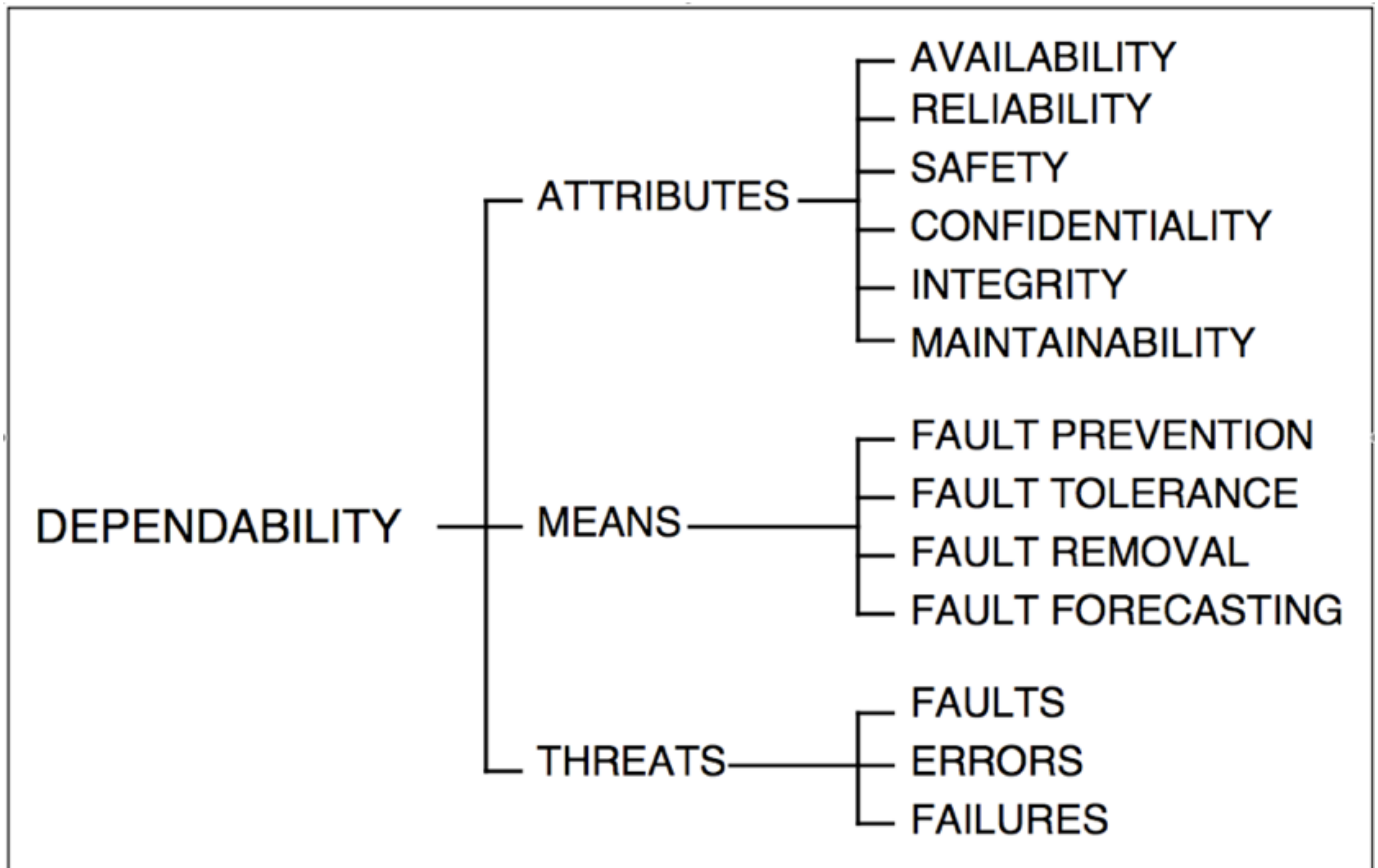
- **Umbrella term** for **operational requirements** on a system
  - Laprie: „*Trustworthiness of a computer system such that reliance can be placed on the service it delivers to the user* “
- Adds a third dimension to system quality
- General question: How to deal with unexpected events ?



DEPENDABILITY IN EMBEDDED

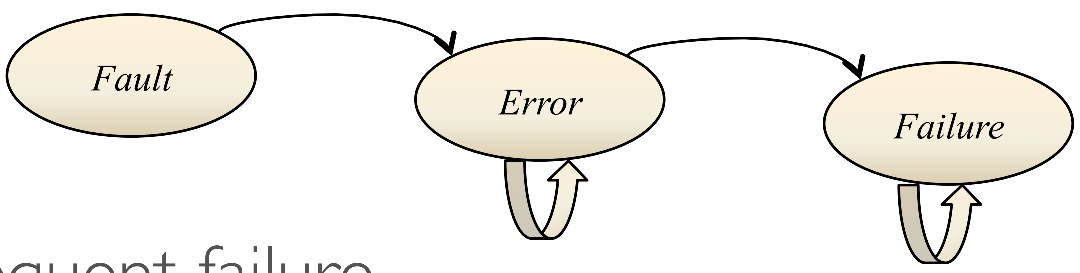
- Critical application domains always considered dependability
  - Aviation industry, power industry, military equipment, …
- But all embedded systems have actuators, people count on them
  - Dangerous real-world interactions may be less explicit
  - Examples: Heating devices, power / water supply devices
- Today more domain experts than software engineering experts
- New challenging through increasingly interconnected devices
  - *Internet of Things (IoT)*

DEPENDABILITY TREE [LAPRIE]



THREATS

- System **failure - ‚Ausfall‘**
  - Event that occurs when the service no longer complies with the specification / deviates from the correct service.



- System **error - ‚Fehler(zustand)‘**
  - Part of system state that can lead to subsequent failure
  - Some sources define errors as active faults - not in this course ...
- System **fault - ‚Fehler(ursache)‘**
  - Adjudged or hypothesized cause of an error
- Failure occurs when error state alters the provided service
- Systems are build from connected components, which are again systems
- Fault is the consequence of a failure of some other system to deliver its service

CONSEQUENCES [KNIGHT]

- Human injury or loss of life
- Damage to the environment
- Damage to or loss of equipment
- Damage to or loss of data
- Financial loss by theft
- Financial loss through production of useless or defective products
- Financial loss through reduced capacity for production or service
- Loss of business reputation, customer base, or jobs

FAULT MODEL

- Faults can be classified into categories on different abstraction levels
  - Physics
  - Circuit level / switching circuit level
    - Interesting for hardware design research (not this course)
    - Investigate logical signals on connections
      - stuck-at-zero, stuck-at-one, bridging faults, stuck-open
  - Register transfer level
  - Processor-memory-switch (PMS) level
  - Hardware system level
  - ... (Software) ...

FAILURE TYPES

- Duration of the failure
  - **Permanent** failures - no possibility for repairing or replacement
  - **Recoverable** failures - back in operation after the system recovered from error state
  - **Transient** failures - short duration, no major recovery action
- Effect of the failure
  - **Functional** failures - system does not operate according to its specification
  - **Performance** failures - performance or SLA specifications not met
- Scope of the failure
  - **Partial** failure - only parts of the system become unavailable
  - **Total** failure - all services go down

FAILURE SEVERITY

- Denotes consequences of failure
- **Benign failures** (‚unkritische Ausfälle‘)
  - Failure costs and operational benefits are similar
  - Sometimes also umbrella term for failures only detected by inspection
  - A system with only such failures is **fail-safe**
- **Catastrophic failures** (‚kritische Ausfälle‘)
  - Costs of failure consequences are much larger than service benefit
- **Significant / serious failures -** Intermediate steps expressing reduced service
- Grading of failure consequences on overall system depends on application
  - Flying airplane - Catastrophic stopping failure, Train - Benign stopping failure
- **Criticality** - Highest severity of possible failure modes in the system

ATTRIBUTES

- **Reliability** - Function R(t)
  - Probability that a system is functioning properly and constantly over time
  - Assumes that system was fully operational at t=0
  - Denotes failure-free interval of operation
- **Availability** - Statement if a system is operational at a point in time / fraction of time
  - Describe system behavior in presence of error treatment mechanisms
  - **Steady-state availability** - Probability that a system will be operational at any random point of time,
    - Fraction of time a system is operational during its expected lifetime: As = Uptime / Lifetime

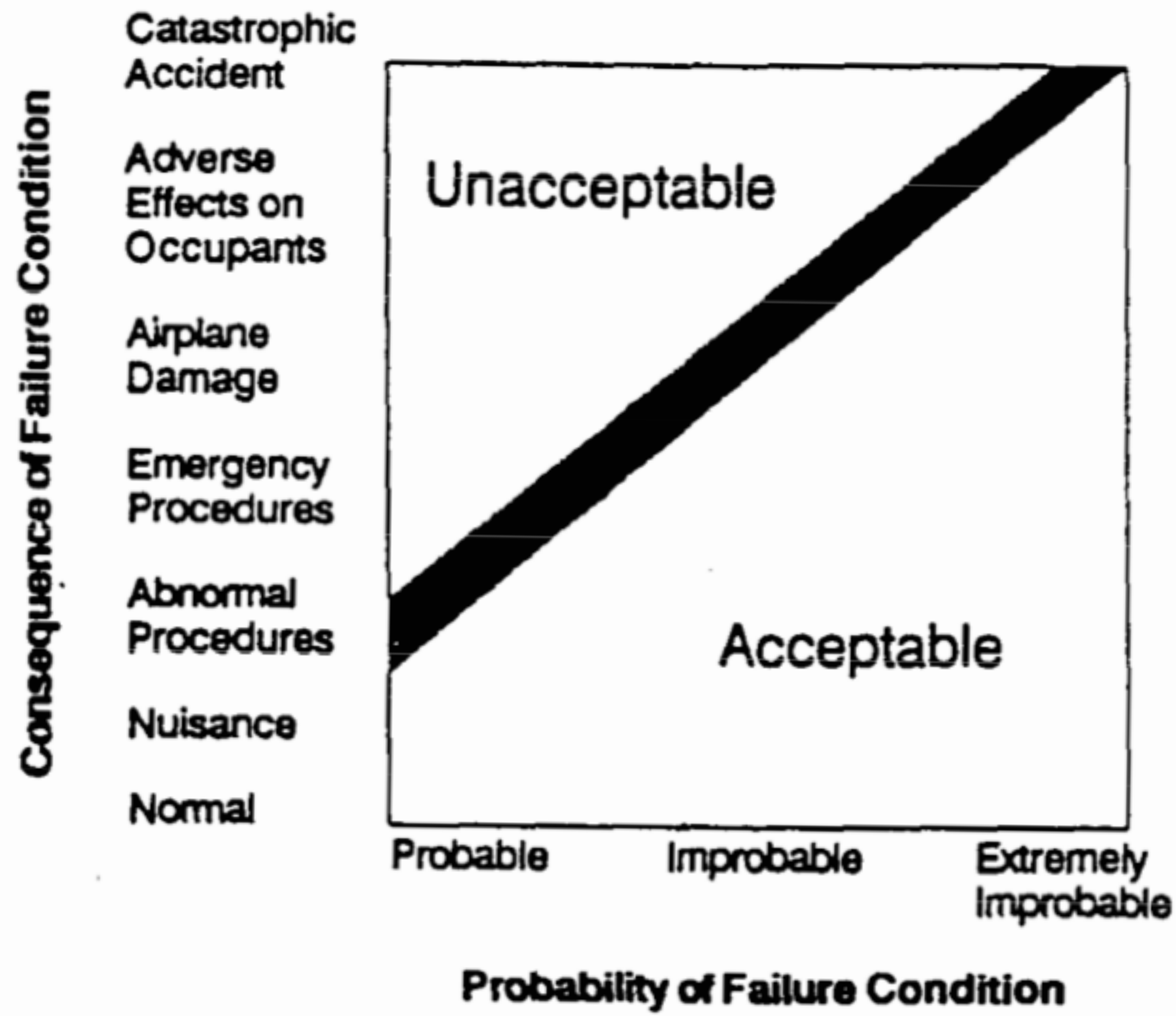
SAFETY

- Different levels of critical participation for a computer system
  - Information provisioning to human controller on request
  - Interpretation of data and presentation to the user
  - Issues command on behalf of the human controller
  - Replaces human controller
- Trend to realize critical systems with commercial-of-the-shelf components
  - Driven by budget cuts and performance advantage
  - Puts sole responsibility on software layer, in contrast to early hardware-only redundancy approaches

EXAMPLE: DO-178D

- Software Considerations in Airborne Systems and Equipment Certification
  - Mature document, developed for more than 20 years
- Definition of **severity of failure** for airplane, crew, and passengers
  - *Catastrophic* - Loss of ability to continue safe flight and landing
  - *Major* - Reduced airplane or crew capability to cope with operating conditions
    - Reduction in safety margins and functional capabilities
    - Higher workload or physical distress for the crew
  - *Minor* - Not significantly reduced airplane safety, slight increase in workload (Example: Change of flight plan)
  - *No effect* - Failure results in no loss of operational capabilities and no increase in crew workload

EXAMPLE: DO-178D



SAFETY VS. SECURITY

Safety	Security
Assumes trustworthy operators	Assumes fault-free system
Assumes closed system	Assumes open, connected system
Existing standards (DO-178C, ISO26262, …)	Existing standards (ISO 27002, Common Criteria, …)

- Different technical foundations, e.g. for recovery from errors
- Embedded system development may need to consider both aspects

PROBLEMS AREAS

- Real-Time
- Code-driven
- Model-driven
- Cross-Compile
- Control loops

Software

- Microprocessor
- CISC vs. RISC
- Microcontroller
- SoC
- ASIC vs. PLC
- ARM

Hardware

Non-functional properties

- Dependability
- Safety
- Security
- Reliability
- Availability