

Real-Time Systems

VO Embedded Systems Engineering

Benedikt Huber

WS 2010/11

Overview

- Definition
- Tasks & Scheduling
- Worst-Case Execution Time Analysis
- Distributed RTS and Clock Synchronization

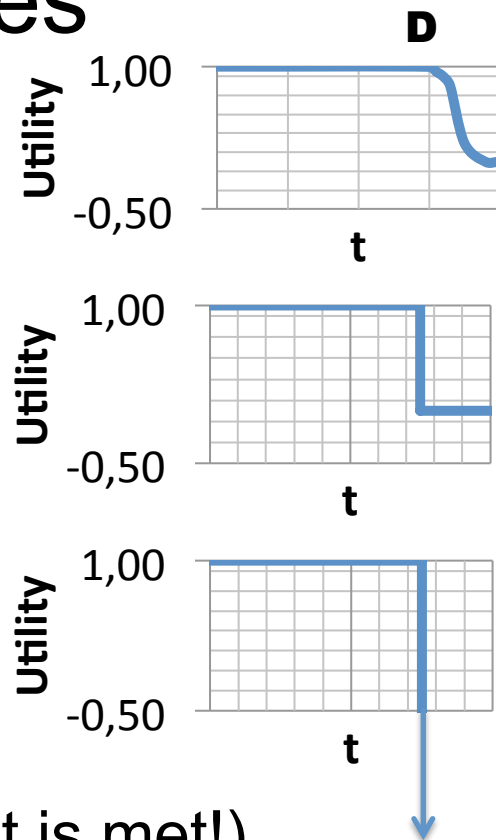
Real-Time Systems

In a real-time computer system **the correctness of the system behavior depends** not only on the logical results of the computations, but also **on the physical instant** at which these results are produced.

- Timing Constraints dictated by the environment
- *Response Time*: The time span between a request and the corresponding response
- *Deadline*: The point in time when a result has to be produced

Classification of Deadlines

- **Soft Deadline**
result has utility after deadline
- **Firm Deadline**
no utility after deadline
- **Hard Deadline**
deadline miss can have catastrophic results
- **Hard Real-Time System**
At least one deadline is hard (ensure it is met!)
- Design of hard/soft RTS fundamentally different

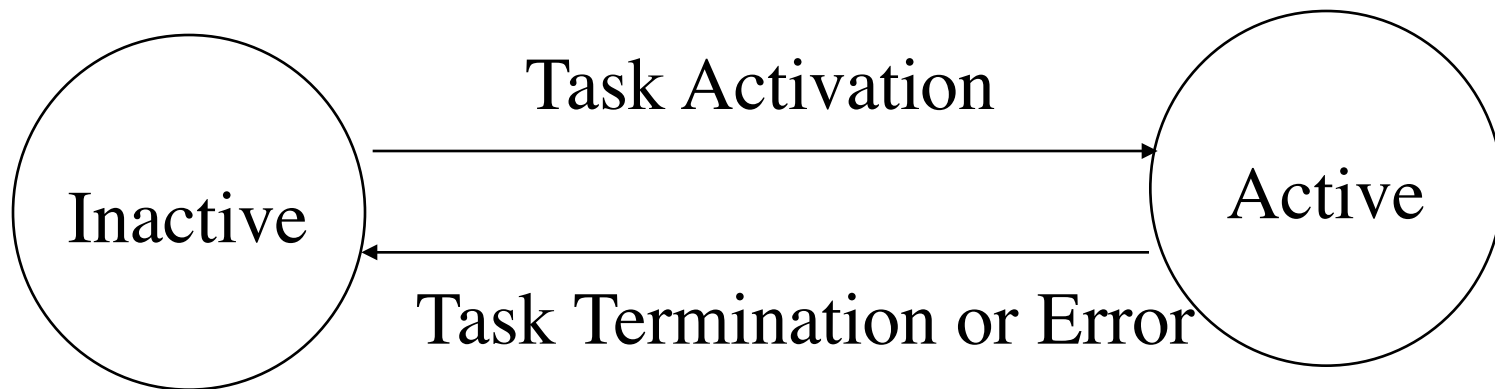


Tasks

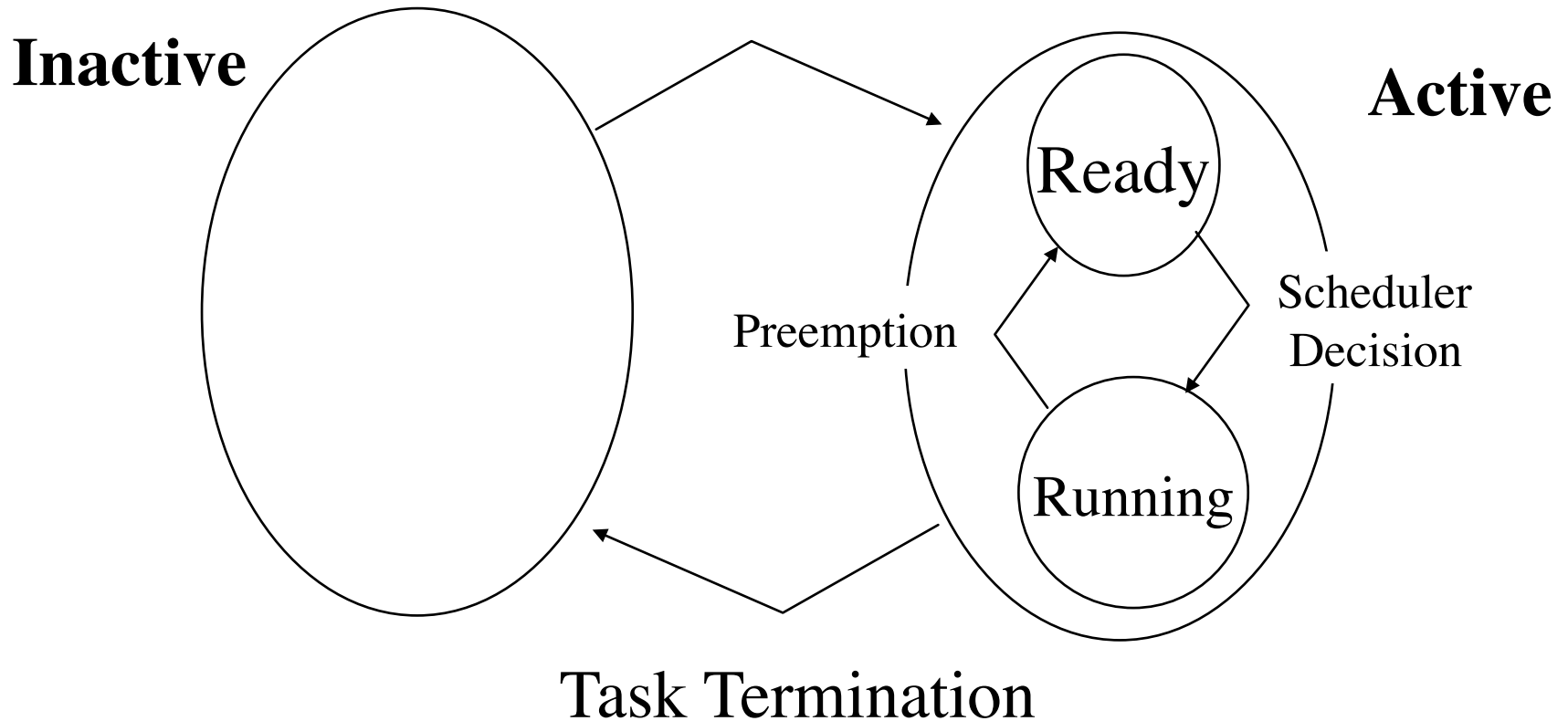
- Execution of program triggered by some event
 - Activation: Periodic vs. Sporadic
 - Synchronization: S-Task vs. C-Task
 - Parameters: period (offset), deadline, execution time
- Typical RT Task: Control Automation
 - Periodic Activation
 - Read inputs (from sensors)
 - Compute
 - Produce Outputs

Non Preemptive S-Tasks

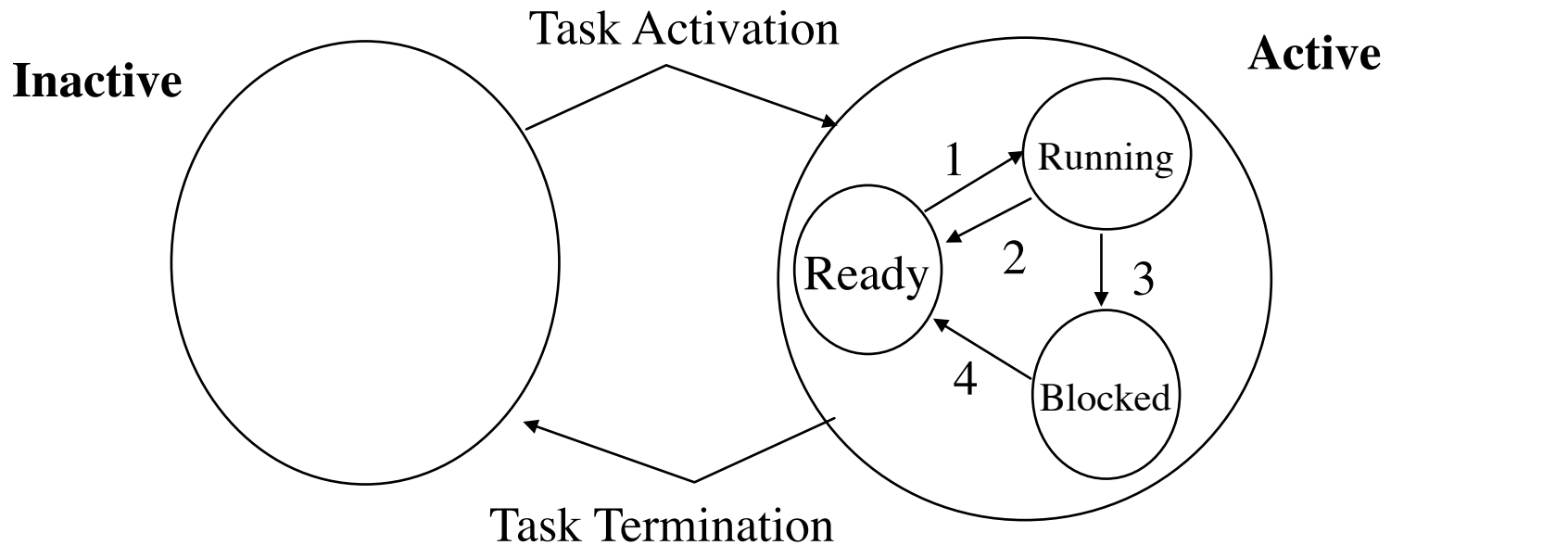
- No preemption, no blocking (Simple Task)



Task States: Preemptive S-Tasks



Task States: Preemptive C-Tasks



- 1 Scheduler Decision
- 2 Task Preemption

- 3 Task executes WAIT for Event
- 4 Blocking Event occurs

Real-Time Scheduling

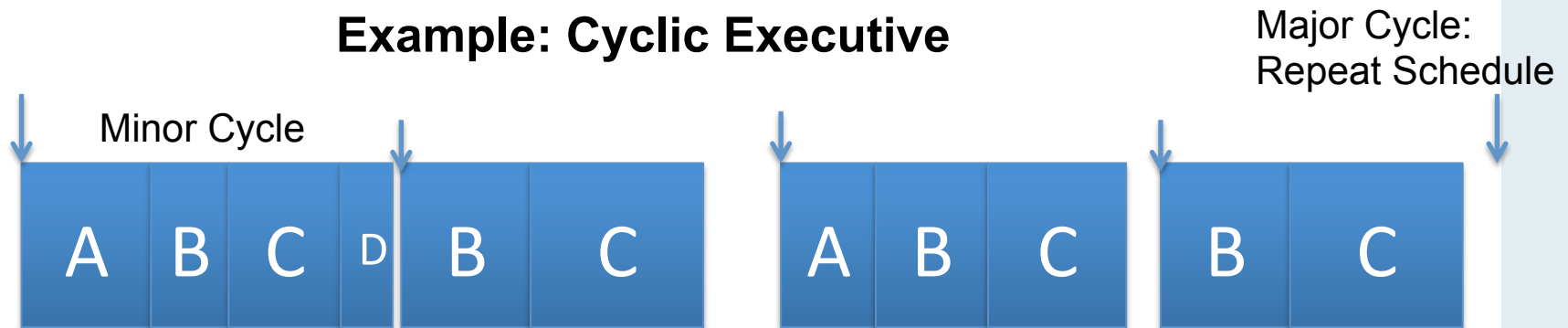
- Assign tasks to CPUs
- Classification
 - Best Effort versus Guaranteed
 - Offline (static) versus Online (dynamic)
 - Preemptive versus Nonpreemptive
 - Central versus Distributed

Offline and Online Scheduling

- Offline Scheduling

- Scheduling decisions are carried out before runtime

Example: Cyclic Executive



- Online Scheduling

- Flexibility (e.g., sporadic tasks) vs. Predictability
 - Mixed Offline/Online Scheduling

The Scheduling Problem

- Scheduling Test
 - Feasible Schedule: No Deadline Misses
 - Given a set of tasks $\{T_i\}$, is there a feasible schedule?
- Scheduling Algorithms
 - Is a certain algorithm guaranteed to produce a feasible schedule for some task set?
 - Optimality: Will the algorithm produce a feasible schedule whenever there exist one?

Utilization

- Fraction of available processor time used ($t \rightarrow \infty$)

$$U = \sum_{j=1}^n \frac{C_j}{T_j}$$

- Hyperperiod H: least common multiple of task periods
 - Workload in Hyper Period: $H * U$
 - $U > 1 \rightarrow$ there will be a deadline miss eventually
- Harmonic Periods: Task periods are multiples of each other

Rate-Monotonic Scheduling

Static Priority Scheduler [Liu and Layland '73]

- Given: n tasks, WCET C_i , period T_i .
- Tasks with the shorter period gets the higher priority (static priority)
- Sufficient Condition that RMS schedule is feasible:

$$U \leq n (2^{1/n} - 1)$$

| |
|---|
| $n = 2 \approx 0.8284$ |
| $n = 3 \approx 0.7798$ |
| $n \rightarrow \infty \approx 0.6933 = \log(2)$ |

- Harmonic periods: Optimal algorithm (100% utilization)
- Overload: Tasks with long period miss deadlines

Earliest-Deadline First

Dynamic Priority Scheduling

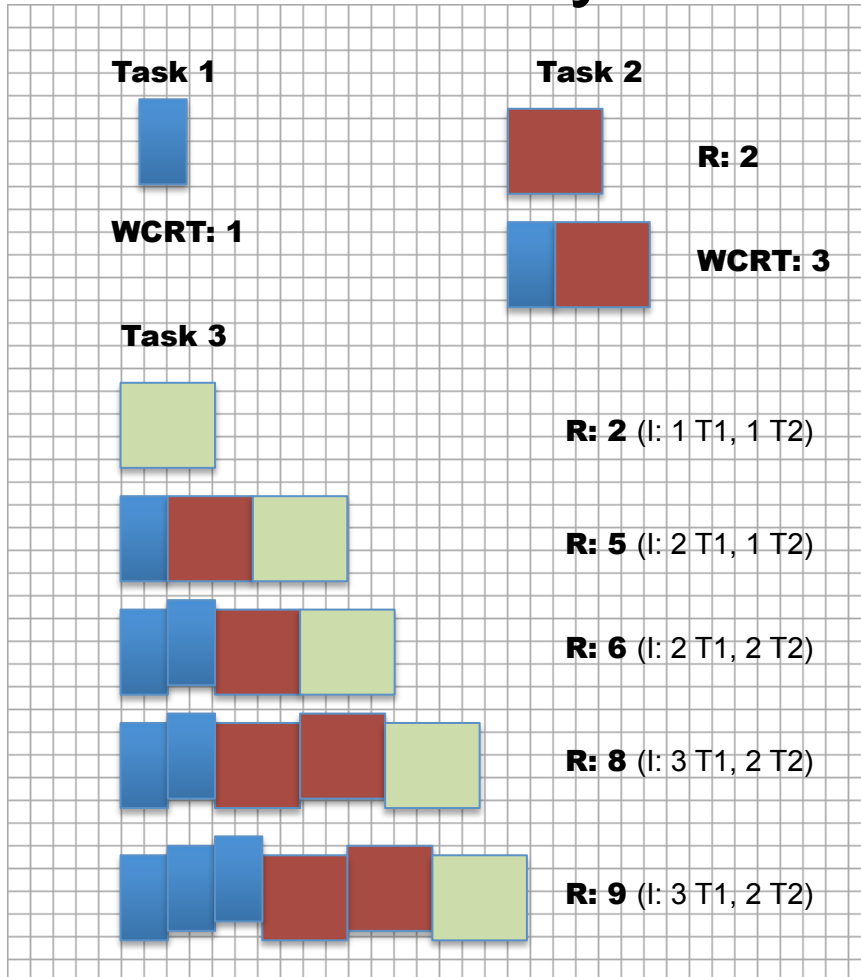
- Same assumptions as rate monotonic
- The tasks with the earliest deadline gets the highest (dynamic) priority
- Optimal for preemptive uniprocessor scheduling (without synchronization) → schedulable if $U \leq 1$
- Overload: All tasks might miss their deadlines

Worst-Case Response Time Analysis

- Goal: More general and precise feasibility test
- WCRT R_i : Max time from activation to completion
 - Feasible Schedule if R_i of all tasks \leq relative deadline
- Example: Rate-monotonic scheduling
 - $R_i = C_i + \text{max. interference from higher-priority tasks}$
 - Calculate smallest solution solution to fixed-point equation

$$R_i^{n+1} = C_i + \underbrace{\sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j}_{\text{Interference}}$$

WCRT Analysis: Example



| Task | Period | WCET |
|------|--------|------|
| 1 | 3 | 1 |
| 2 | 5 | 2 |
| 3 | 9 | 2 |

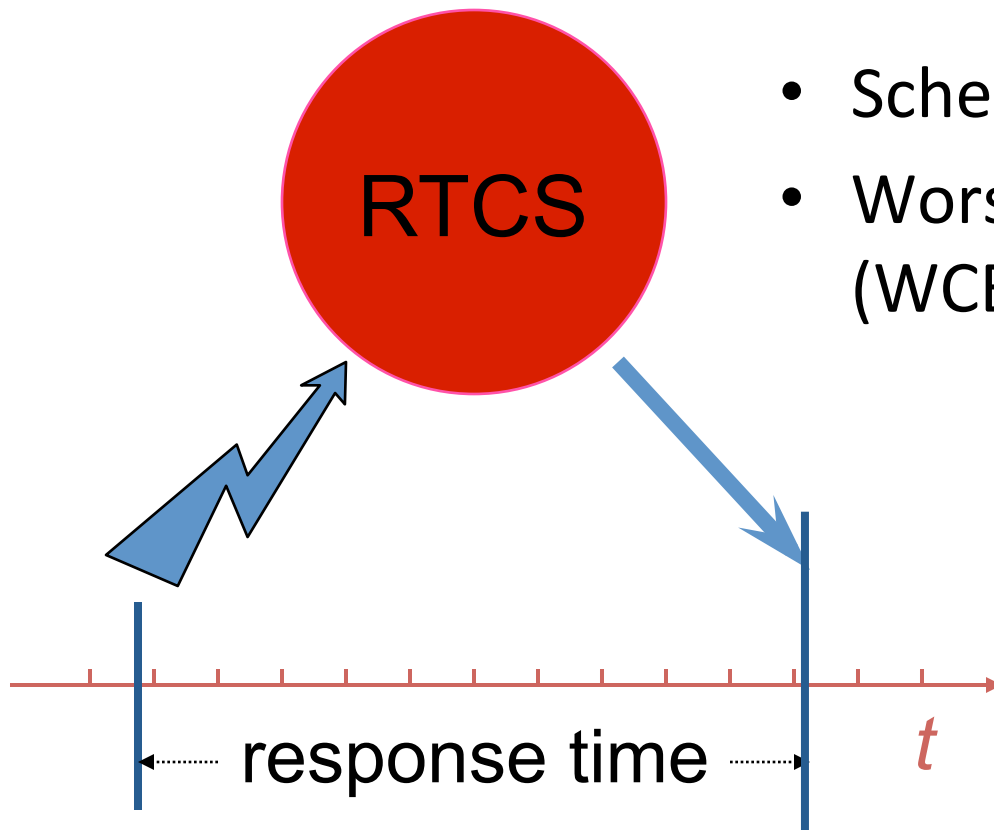
Hyperperiod: 45 (43 work)
Utilization: $0.956 > 0.78$

| Task | Period | WCRT |
|------|--------|------|
| 1 | 3 | 1 |
| 2 | 5 | 3 |
| 3 | 9 | 9 |

Other topics in Real-Time Scheduling

- Model: Offsets, Precedence Constraints, Release Jitter
- Synchronization (critical sections)
 - Blocking times
 - Priority Inheritance Protocol, Priority Ceiling Protocol
- Scheduling for soft real-time and mixed criticality RTS
- Multiprocessor Scheduling
- Distributed Scheduling

Timing Analysis



- Scheduling +
- Worst Case Execution Time (WCET) Analysis

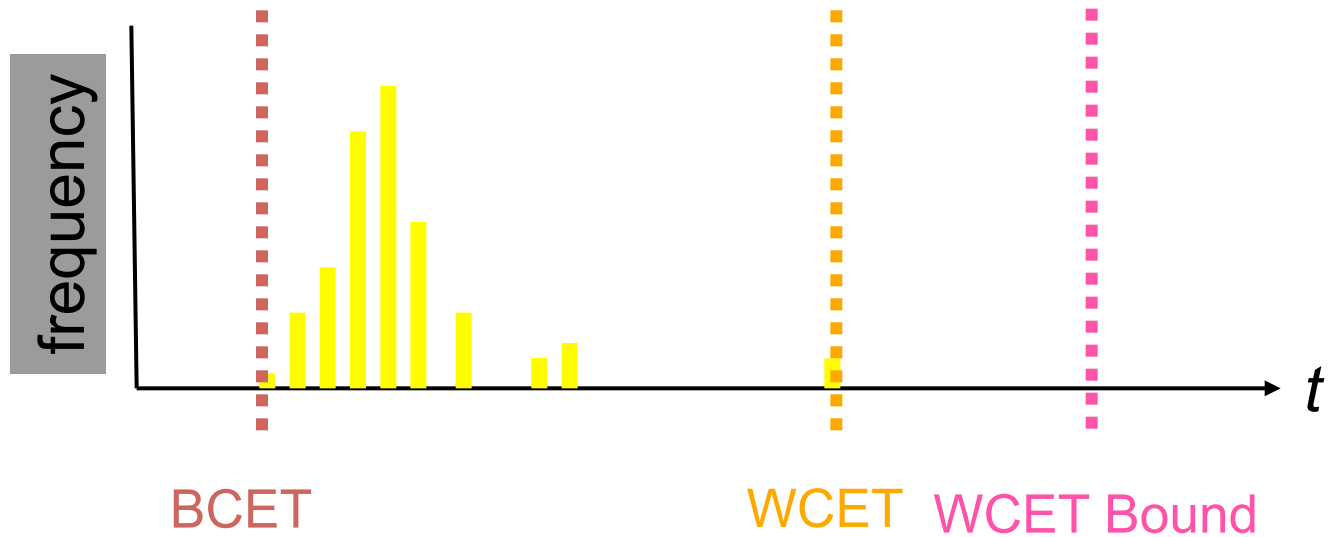
Scheduling vs. WCET Analysis

- Scheduling objects
 - Units of execution (tasks) with WCET
 - Precedence relations
 - Synchronization (critical sections), communication
- WCET-analysis objects
 - Analysis of one task / part of a task (e.g., a function)
 - Usual Assumption: No blocking, No preemption
 - Execution time variations due to initial state and input data

WCET Analysis

- Motivation: We need to know maximum execution times to proof schedule is feasible
 - ... or to calculate schedule (e.g., least-laxity first scheduling)
- WCET Analysis: Obtain **upper bounds** for the execution time of pieces of code
- Execution time depends on
 - (machine) code to be analyzed
 - on a given machine (CPU, Memory, etc.)
 - in a given application context (restrictions on initial state)

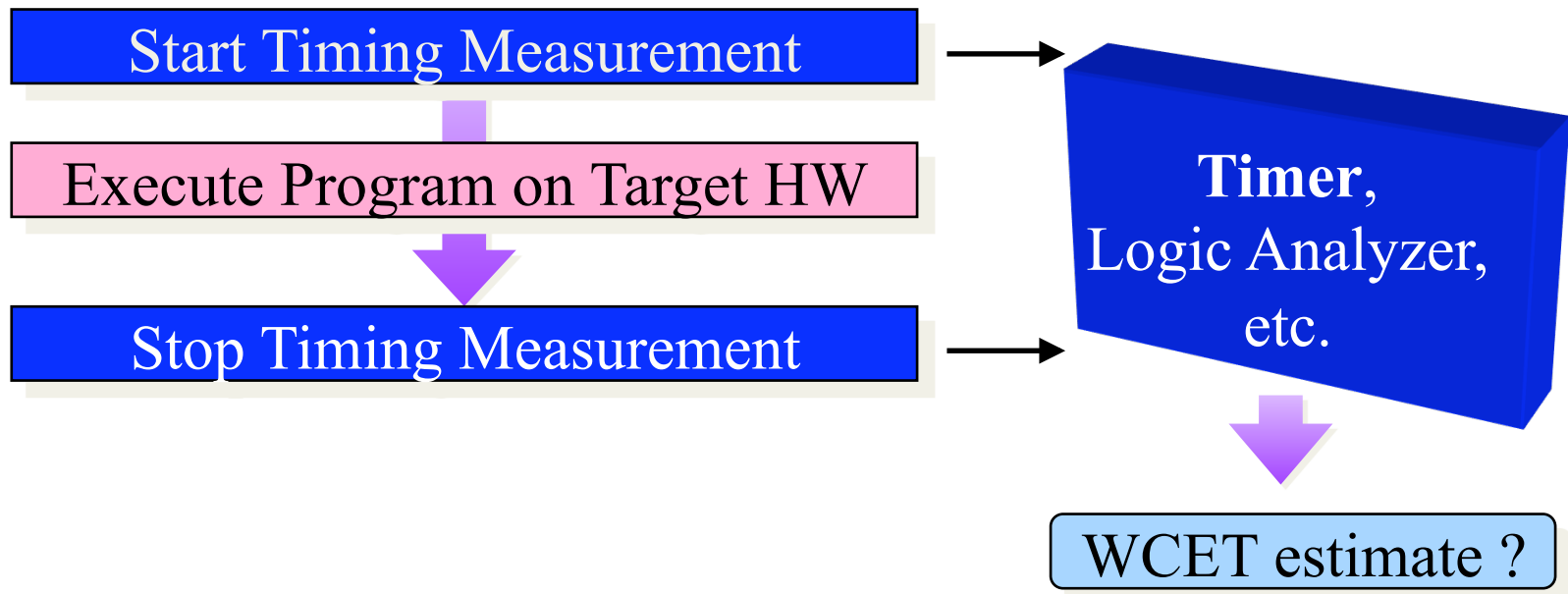
Quality of WCET Analysis



Pursuing a Simple Solution

End-To-End Measurements

- Idea: Obtain WCET by measuring execution time



Problem: Complex Architectures

Pursuing a Simple Solution (2)

- Simple Architecture: Constant execution time for one instructions
- Complex Architectures
 - Instructions with variable timing (e.g. multiplier, division)
 - Instruction- and Data Caches
 - Superscalar Out-Of-Order Execution
 - Branch Predictors, Branch Target Buffers
 - ...
- Initial state of hardware might not trigger worst case

Problem: Testdata Selection

Pursuing a Simple Solution (3)

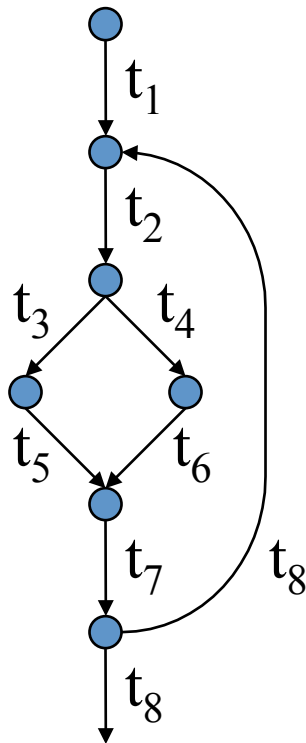
- Measuring **all** different execution traces of a real-size program is **intractable in practice**
- Selected test data for measurement may **fail** to trigger the **longest execution trace**
 - Intuition fails on complex architectures
 - **Rare execution scenarios** may have been **overlooked** when selecting test data (e.g., exception handling, ...)

End-To-End Measurements?

Pursuing a Simple Solution (4)

- Simple measurements are useful to get a first rough estimate of the execution time.
 - More systematic WCET analysis techniques are required to obtain a trustworthy WCET bound!
-
- Static WCET Analysis
 - Hybrid Measurement-Based WCET Analysis

Static WCET Analysis



- Goal: Calculate safe and precise upper bound for execution time by means of static analysis
- Determinants
 - Possible sequences of program actions (= **execution paths**) in given application
 - Duration of each occurrence of an action on each possible (= **feasible**) path

Calculating the WCET

- High-Level Analysis
 - Model set of possible execution paths
 - Control Flow Graphs, Flow Facts (Loop Bounds, etc.)
- Low-Level Analysis
 - Context dependent execution times of basic blocks
 - Includes global Cache and Pipeline Analysis
- WCET calculation
 - E.g., using an ILP solver

Challenges in Path Analysis

- Control Flow Graph / Call Graph construction
 - Targets of indirect jumps and indirect function calls
- All execution paths in model need to be finite
 - Loop and recursion depth bound
 - Problematic: Busy Waiting (manual annotations)
- Should exclude infeasible path
 - E.g., some parts of a function might not be executed depending on the actual parameters

Challenges due to Complex HW

- In modern architectures, it is too pessimistic to just use the worst-case timing of a single instructions
 - Caches (need global analysis)
 - Speculation
 - Complex pipelines
- Cache analysis classifies cache accesses as hit or miss (depending on the context)
- Pipeline analysis simulates all possible pipeline behaviors to find the worst-case timing

WCET-oriented programming

- Do not optimize for the average case!
 - Try to produce code that is free from input-data dependent control decisions
 - Keep number of operations that are only executed for a subset of the input-data space small
- Note: On simple architectures, the ‚simple solution‘ works if we have a only one (a small, well-defined set of) execution path(s)

Distributed Real-Time Systems

Distributed Real-Time Systems

- System of multiple, autonomous, cooperating nodes
- Advantages
 - Use of computing power where it is needed
 - Scalability/Performance (Exploitation of Parallelism)
 - Availability/Reliability (Exploitation of Redundancy)
- Challenges
 - E.g., Clock Synchronization, Scheduling & Communication

Distributed Real-Time Systems

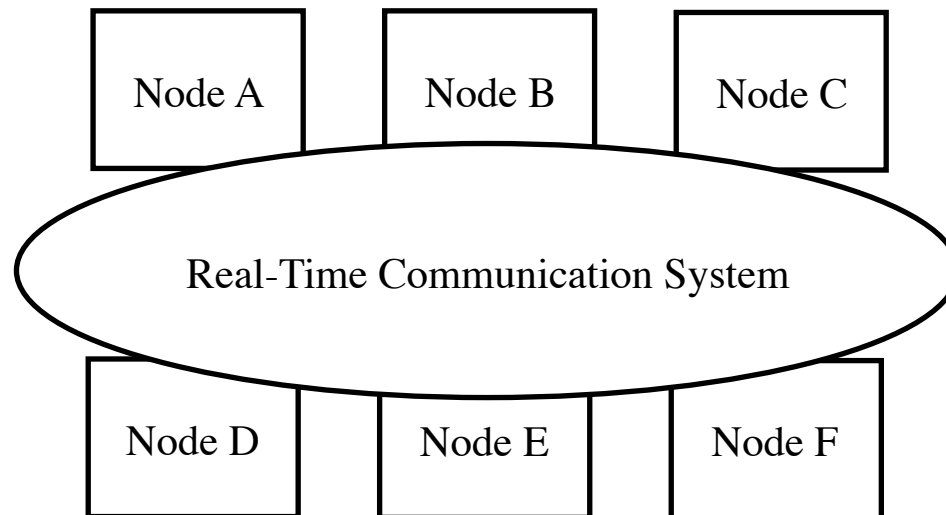
- ‘Ingredients’ of Distributed RT Systems:

Network

Node

Messages

Tasks



Message Semantics

- **Event messages** – contain event information
 - Every event is significant, loss of a message can lead to the loss of the synchronization in state between sender/receiver
 - Not idempotent, requires implementation of message queues
- **State messages** – contain state information, e.g., current temperature
 - Old state is overwritten with new state
 - Idempotent
- In RT control systems state semantics is more important than event semantics

Time and Order

- In Distributed RT systems different functions are executed on different nodes
- To guarantee consistent behavior, all nodes should be able to process events into consistent temporal order
- A global time base helps to establish such a consistent order

Clocks and Time Stamps

- A *clock* is a device that contains a counter and increments this counter periodically according to some law of physics (*microticks*).
- The *granularity* of a clock: interval between two consecutive microticks
- Given a clock and an event, a *timestamp* of the event is the state of clock immediately after the event occurrence, denoted by $clock(event)$.
- *Reference clock*: A clock that has a granularity that is much smaller than the duration of any intervals of interest

Clock Drift

actual reference clock μ ticks per granule

- Clock Drift

$$drift_i^k = \frac{z(\text{microtick}_{i+1}^k) - z(\text{microtick}_i^k)}{n^k}$$

nominal reference clock μ ticks per granule

- microtick_i^k : i^{th} microtick of clock k

- Drift Rate

$$\rho_i^k = \left| \frac{z(\text{microtick}_{i+1}^k) - z(\text{microtick}_i^k)}{n^k} - 1 \right|$$

- Perfect clock has drift rate of 0
- Real clocks have drift rates from 10^{-2} to 10^{-8}

Clock Synchronization

- Periodically: Resynchronization interval
- Requirements
 - Bounded drift rates
 - Bounded transmission delay
- Internal Clock Synchronization
 - Objective: Ensure bounded internal deviation (Precision Π)
- External Clock Synchronization
 - Ensure bounded deviation between any clock and reference time server (Accuracy A)
 - External Clock e.g. provided by GPS receiver

Central Clock Synchronization

- Drift offset (Γ): maximum divergence of any two good clocks from each other during the resynchronization interval
- Master periodically sends synchronization messages to all slave nodes
- Deviation Measure: Master/Slave clock difference corrected by latency
- Slave corrects its clock by this deviation to bring it into agreement with the master's clock.
- Precision of Central Master Algorithm (ε ...jitter, Γ ...drift offset): $\Pi_{central} = \varepsilon + \Gamma$

Distributed Clock Synchronization

- Typically, distributed fault-tolerant clock resynchronization proceeds in three distinct phases:
 - Every node acquires knowledge about the state of the global time counters in all other nodes by message exchanges among the nodes.
 - Every node analyzes the collected information to detect errors and executes the convergence function to calculate a correction value for the node's local global time counter.
 - The local time counter of the node is adjusted by the calculated correction value.

Distributed Clock Synchronization (2)

- The algorithms differ in the way in which
 - they collect the time values from the other nodes,
 - in the type of convergence function used (e.g., midpoint), and
 - in the way in which the correction value is applied to the time counter (e.g. clock amortization, rate correction).

References

■ Books and Articles

- Deadline Scheduling for Real-Time Systems - EDF and Related Algorithms (Stankovic, Spuri, Ramamritham, Butazzo)
- The Worst-Case Execution Time Problem – Overview of Methods and Survey of Tools (TECS, Volume 7, Issue 3)
- Real-Time Systems: Design principles for distributed embedded applications (Kopetz)

■ Lectures

- Real-Time Systems (Prof. Kopetz)
- Real-Time Scheduling (Prof. Schmid)
- Timing Analysis for Safety-Critical Systems (Prof. Puschner)

ENDE

Danke für die
Aufmerksamkeit!

