Technische Universität Dresden

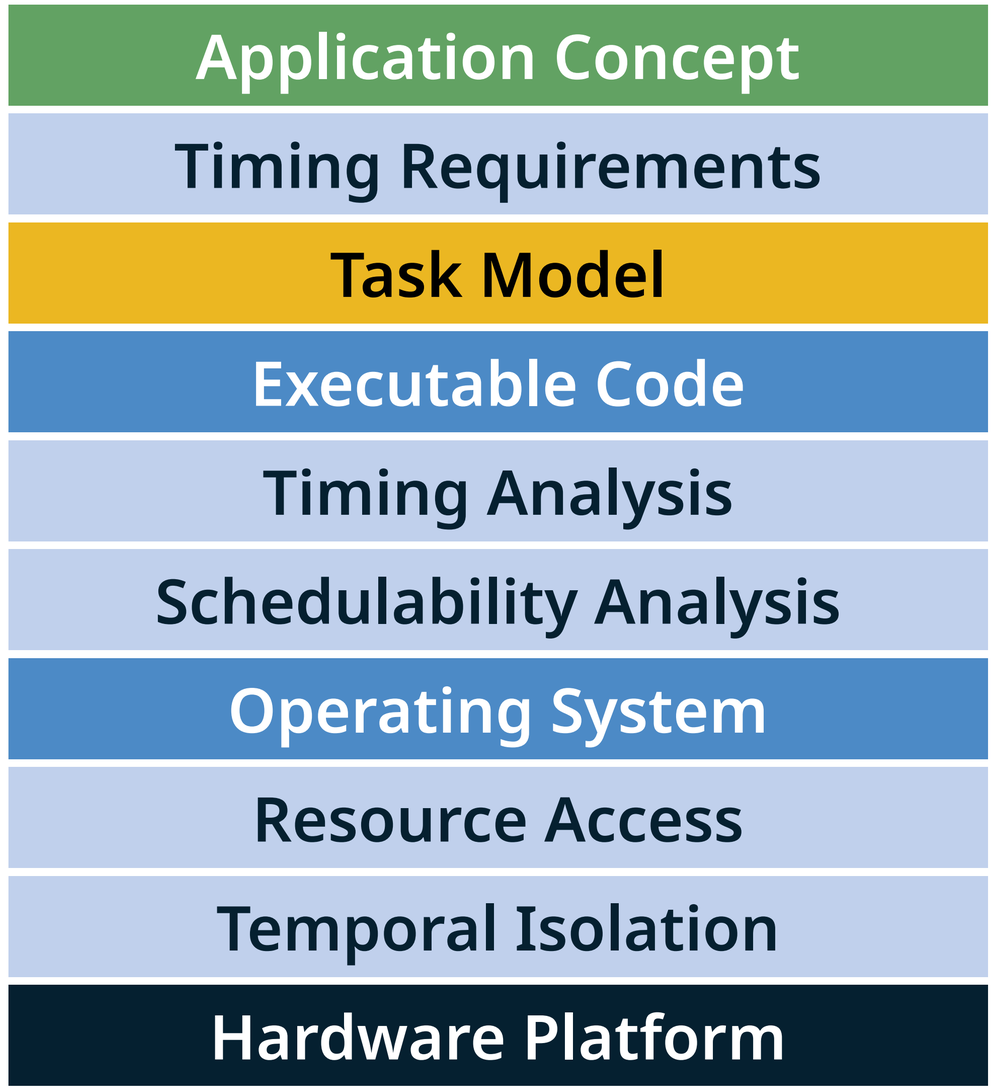
barkhausen institut

Modeling Real-Time Workloads

Real-Time Systems

Michael Roitzsch

Real-Time Systems Technology Stack



Models in Engineering Disciplines ...

- ... look at quantitative properties
- very common in (real) engineering disciplines

Ingredients

- load (or Objective)
- resources
- mapping

e.g. for building bridges:

materials, weight of cars, velocity of wind, structures

Models in Real-Time Systems

Purpose of Models

- describe: certain properties
- derive: knowledge about (same or other) properties (using tools)
- neglect: details not important for property

Real-Time Systems

- properties: timing behavior + system structure
  restrict problem space to simplify NP-hard scheduling
- derive: can timing constraints be met?
  results from analysis research often drive real-time models
- neglect: actual code execution

Resources in Real-Time Systems

Active Resources

- CPUs, networks, disks, ...
- property: speed, bandwidth, ...
- need certain time to achieve objective

Passive Resources

- memory, locks, sequence numbers, ...

Active vs. Passive

- speed/execution vs passive occupation
- passive resources often modelled in terms of additional time for active resources
- distinction not always clear
- both may or may not be reusable and preemptible

Basic Modelling Unit: Job

Set of Jobs

J_i jobs

r_i release times of individual jobs, also called arrival times

d_i deadlines of individual jobs (relative or absolute)

e_i execution time demand of individual jobs

Terminology

- response time: time from job release to completion
- slack: remaining time from (expected) job completion to deadline
- makespan: time from beginning to end of job execution, including gaps

Dependencies between Jobs

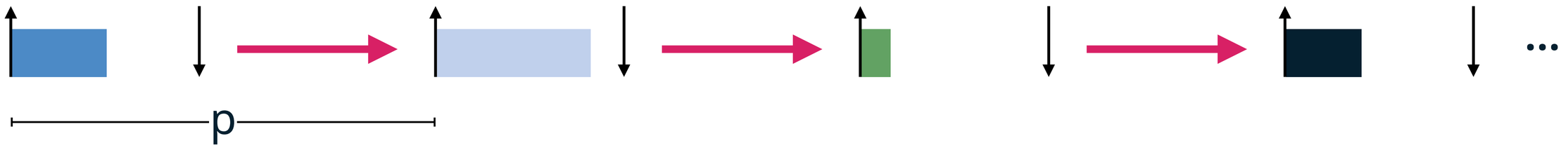
Precedence

- jobs may depend on results of other jobs
- e.g. producer / consumer scenarios
- precedence graph: partial order relation on jobs

Shared Data

- jobs use some resource (e.g. critical section) that can only be used by a single job at a time

Structured Chain of Jobs: Periodic Task



Periodic Task

T task, consisting of a sequence of

J_i jobs, implicit dependencies along the sequence (no parallelism)

p period, inter-release separation between consecutive jobs

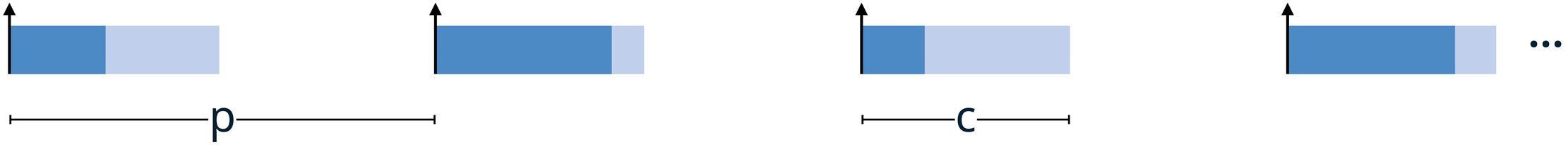
φ release time of first job (phase)

d deadlines of jobs (relative or absolute)

e_i execution time demand of individual jobs

many analysis algorithms assume a periodic task model

Periodic Task – Simplifications



Periodic Task

T task, consisting of a sequence of

J_i jobs, implicit dependencies along the sequence (no parallelism)

p period, inter-release separation between consecutive jobs

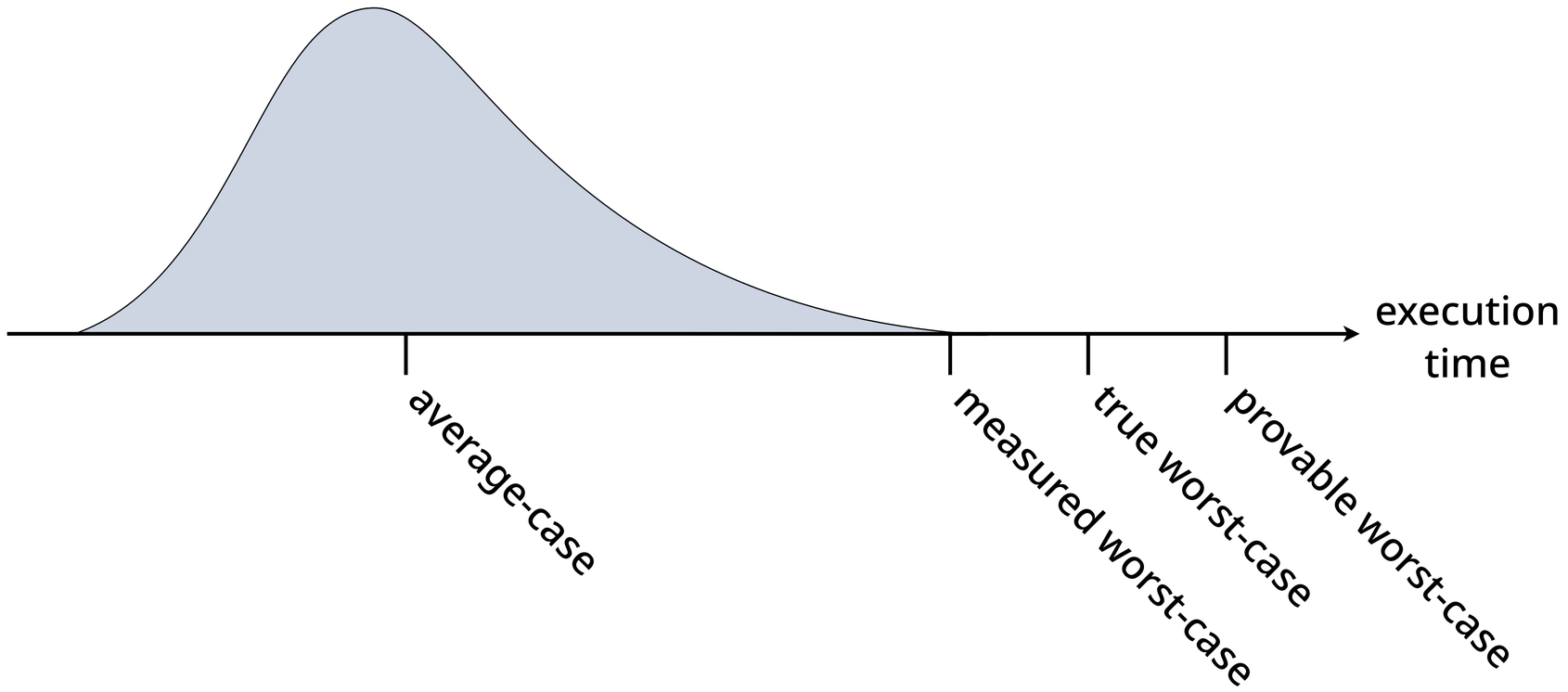
$\varphi = 0$

$d = p$

$e_i = c$ worst-case execution time demand

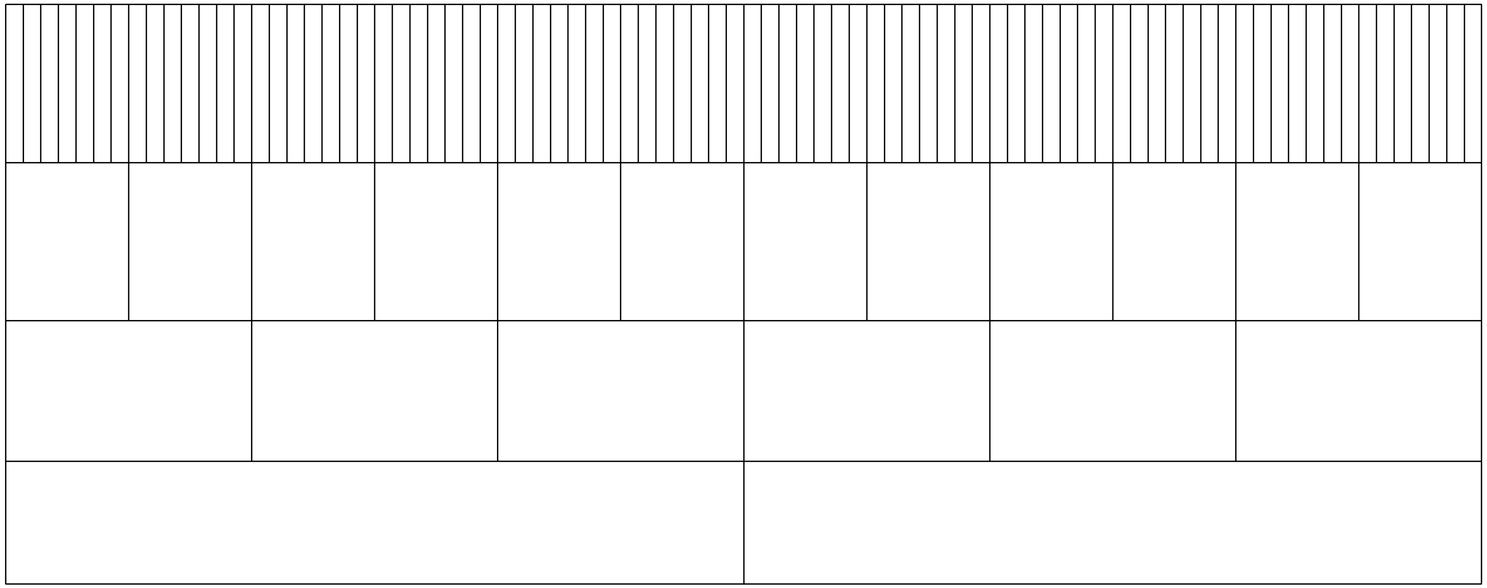
tasks can be expressed as tuples T = (p, c), like $T_1 = (4, 2)$

Parameter: Worst-Case Execution Time



common abbreviation: **WCET**

Parameter: Period – Special Terminology



Harmonic Periods (Simply Periodic)

longer periods are integer multiples of shorter periods

Hyperperiod

least common multiple of all periods in a task set

What about Non-Periodic Workloads?

Real-time systems often respond to events at random points in time.

Examples

- driven by physical events
- input sampling, e.g. from a sensor
  - higher or lower frequency, depending on system status
  - maximum frequency known in advance
- messages or alarms
  - bounded response time required
  - minimum inter-release time known

Sporadic and Aperiodic Tasks

Sporadic Task Model

T task, consisting of a sequence of

J_i jobs

p **minimum** inter-release separation between consecutive jobs

~~φ release time of first job (phase)~~

d deadlines of jobs, **relative to release time of job**

$e_i = c$ worst-case execution time demand

Aperiodic Tasks

p = 0 → unrestricted execution demand,
schedule feasibility can no longer be determined a priori

A Warning about Terminology

Jane Liu's Text Book

- 'periodic' tasks: periodic or sporadic
- 'sporadic/aperiodic' tasks: p can become arbitrarily small
  - sporadic: hard deadlines
  - aperiodic: soft or no deadlines
- better to explicitly say 'tasks with minimum inter-release separation p'

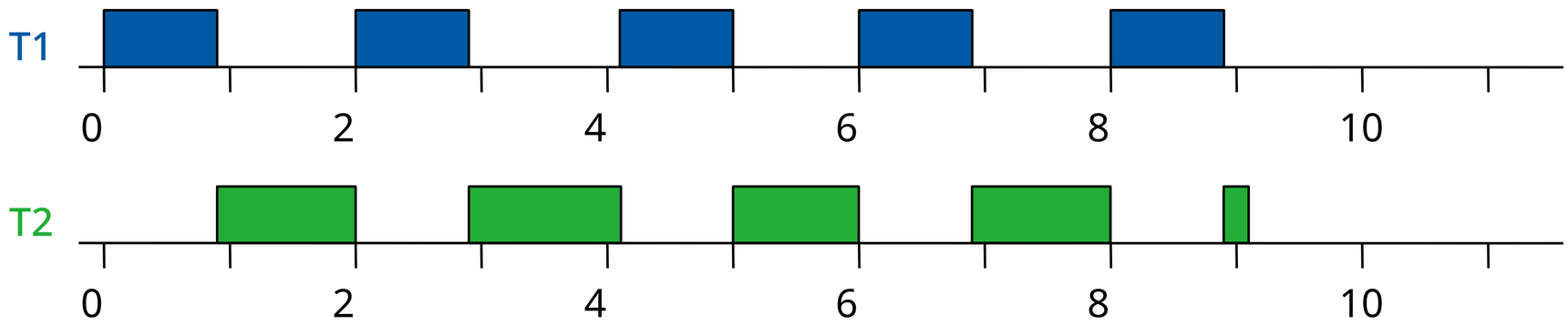
Schedule: Mapping of Load to Resources

Schedule: assignment of a set of tasks onto the available resources

- a schedule is called **valid** if
  - every job is assigned to at most one resource at any time
  - no job is scheduled before its release time
  - all constraints are satisfied:
    precedence, usage of (passive) resources
- a valid schedule is called **feasible** if
  - all deadlines are met
- a feasible schedule is called **sustainable** if
  - all deadlines are met even when parameters change in the “good” direction
  - reason: computers can never precisely follow task parameters

Example schedule

T1: (2,0.9) **T2:** (5,2.3)



Feasibility argument extends to infinity, because schedule can be repeated for each hyperperiod.

Schedulers and Admission

Scheduler

- enacts/enforces/interprets a schedule
- sometimes also called dispatcher

Admission

Can a new task be allowed into the current set of tasks and the current resources such that a feasible schedule exists?

- admission heavily depends on the used scheduler

Schedulers can be based on:

- time tables
- (static, dynamic) priorities

Priority vs. Criticality

Priority-Based Schedulers

- jobs get priorities assigned, for example during admission
- a priority-based scheduler implements:
  If a high-priority job competes with a low-priority job,
  then the high-priority job wins.

Criticality of Tasks

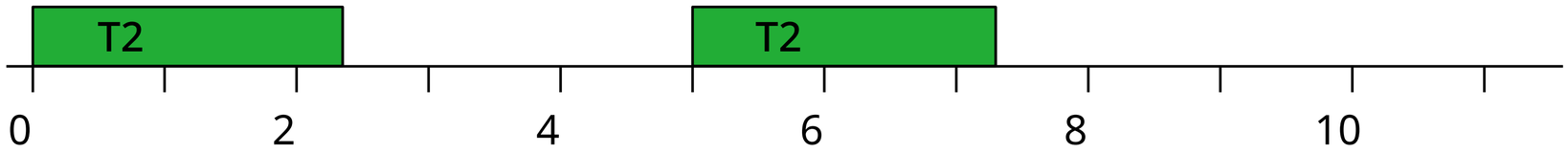
- Some jobs may be more important than others. We want to meet the deadlines of those important jobs, possibly at the expense of less-important jobs.
- criticality becomes a tunable parameter, set by the system designer
- priority however is a result of admission, not tunable

Priority Assignment Following Criticality

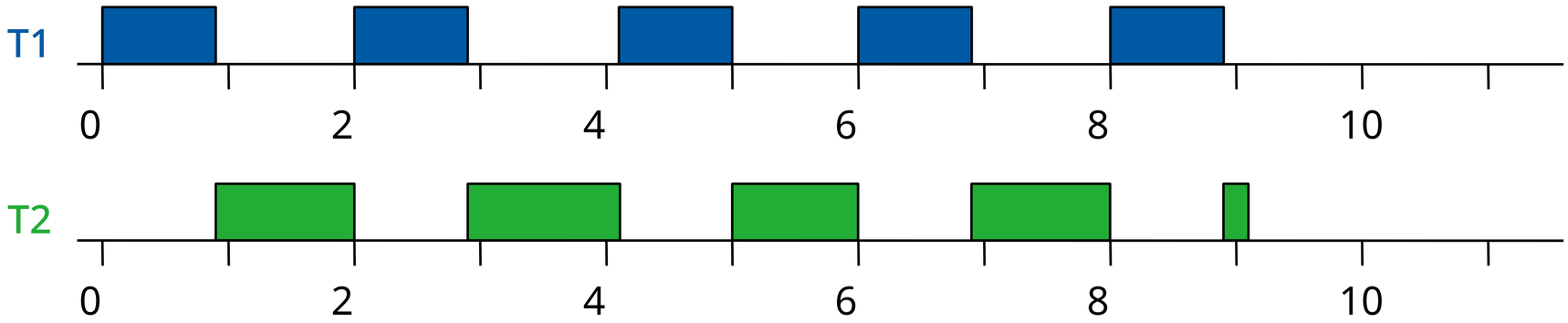
The more critical a task the higher the priority

T1: (2, 0.9) **T2:** (5, 2.3)

T2 more critical than **T1**



T1 misses deadline in Job 1 and 3, unnecessarily …



Preemption

Preemptibility of Jobs

- preempt: stop and resume later
- can jobs be preempted at any time to allow the execution of other jobs?
- non-preemptible jobs must not/cannot be preempted before completion (i.e. cannot release their resource and resume later, examples: SSD, GPU)

Cost of Preemption

- context switch operation
- WCET of jobs may depend on:
  - the number and position of preemptions within the preempted jobs
  - resource usage of other jobs or the operating system

Multiple Processors

Task: (period, WCET)

T1: (2,1) **T2:** (5,3)

- Migration
  - at task granularity in between jobs
  - or while individual jobs are running
- local/global run queues
- more details in dedicated lecture

Modes and Mode Changes

a system may operate in different modes

i.e., characterized by a different set of parameter values

Examples

- aircraft: flying, taxiing, start/land
- car: manual driving, autonomous driving, charging

Mode Change

- transition from one mode to another
- transition itself may have real-time requirements

Execution Time Pessimism

Actual execution times, depend on

- data dependencies: if then else, compression, loops
- hardware: caches, predictors, ...
  (see specific lecture on real-time & hardware)

Actual execution times not known in advance, instead we use:

- pessimistic bounds like worst case execution time
- probabilistic distributions

WCET-based scheduling:

- extremely hard to find “good” WCET
- underutilization of resources
- more models like **hard, firm, soft real-time, mixed criticality**

Soft Real-Time Flavors

- Maximum tardiness t:
  deadlines are never missed by more than t
- m/k systems:
  maximally k of any m consecutive deadlines are missed
- Probabilistic:
  described by distribution functions and expected miss ratio

Uncertain Execution Times

T	task, consisting of a sequence of
J_i	jobs, implicit dependencies along the sequence (no parallelism)
p	period, inter-release separation between consecutive jobs
d	deadlines of jobs
q	**probability that deadline is met**
e	**execution times as probability distribution**

A schedule is called feasible if a q-fraction of the deadlines are met.

Uncertain Job Releases

Periodic Event Stream Model

T stream of events

J_i each event releases a job

p period: expected inter-release separation

Jitter-Constrained Event Stream Model

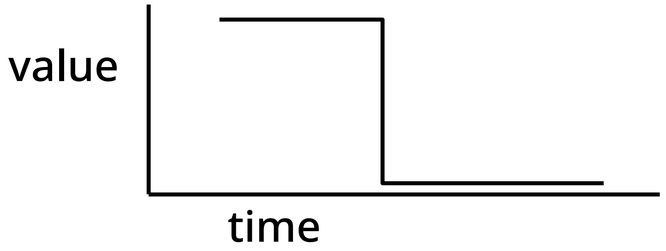
τ allowed deviation from inter-release time p,
jobs arrive in the interval [p – τ, p + τ]
may be larger than p

D minimum inter-release separation

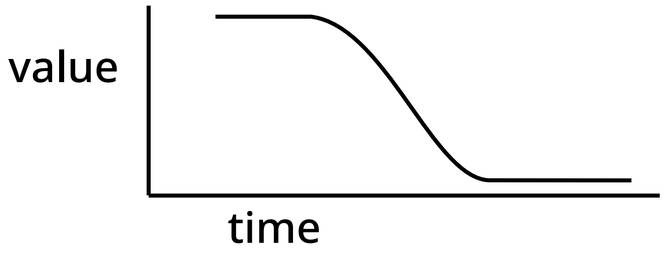
Differentiating Job Importance

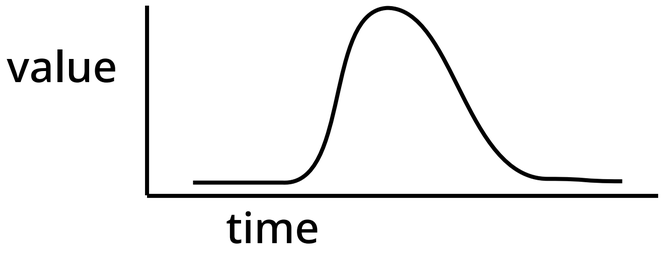
Time-Value-Functions

Hard real-time:



Others:





Differentiating Job Importance

Imprecise Computations

- one mandatory job in each period, followed by sequence of optional jobs
- mandatory job has to complete, optional jobs can be aborted
- optional jobs refine an approximate result
- the longer the optional jobs can run the better the result
- example: computer vision workloads
- flavors: m,k or probabilistic guarantees on optional job completion

Differentiating Job Importance

Task Pairs

- main job and fallback job within each period
- execution time of main job uncertain, WCET of fallback job is known
- try executing the main job first
- abort and switch to fallback at WCET before deadline

Combining Hard and Soft Real-Time

- many system have some core of hard real time tasks
- plus auxiliary tasks with soft requirements or without explicit requirements
- or aperiodic tasks where no formal guarantees can be given

Examples

- traffic control:
  avoid crashes (hard), optimize flow (soft)
- measurement system:
  value of timestamps (hard), deliver data for telemetry (soft)
- quality control on conveyor belt:
  try remove part from belt (soft), otherwise shut down belt (hard)

Simple Solution: Use Slack

Best Effort

- workload without formal timing requirements
- runs in time remaining after all real-time workloads have executed
- i.e. runs in the background
- can result in unnecessarily poor performance for best effort work
- idea: some periodic tasks could be delayed without deadline misses

Structuring the Slack: Server-Based Systems

Server

virtual periodic task as time reservation for aperiodic/background work

P server period: inter-release separation between server jobs

C computation budget: time reserved for aperiodic/background work

Many flavors depending on:

- what priority the server task has
- when the server is invoked
- how the server selects work to execute
- when the server budgets is replenished

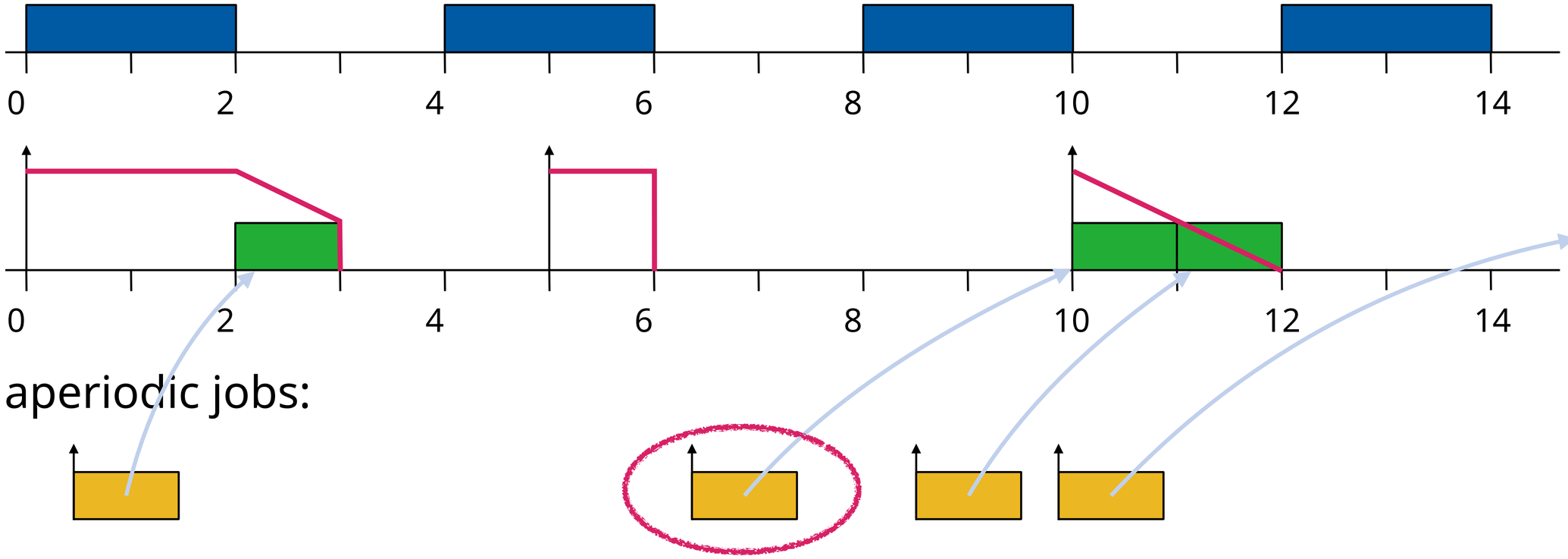
Polling Server

- runs as a regular periodic task
- server budget set to C at the start of each server period
- aperiodic jobs are queued until a server job is invoked
- on invocation, the server **polls the aperiodic job queue** and executes jobs:
  - until the queue is empty OR
  - the server budget is exhausted
- it then suspends itself until the next job release of its periodic task

Polling Server Example

high-priority periodic task: (4, 2)

lower-priority polling server: (5, 2)



could execute immediately?

Deferrable Server

- like polling server: budget replenished to C at the start of server period
- **preserves its budget** if no aperiodic jobs are pending upon invocation
- executes aperiodic jobs arriving at any time during the server period until the server budget is exhausted
- but: jobs arriving close to the end of the server period will be executed
- this may push lower-priority jobs beyond their deadline
- the deferrable server cannot be analysed for schedulability as a regular periodic task
- fixed with **sporadic server:** rules adjusted to ensure it never demand more processing time than a periodic task with the same parameters

Server Designs: Why so many?

More Servers

sporadic, constant bandwidth, weighted fair-queueing, ...

Conflicting Goals

- a virtual periodic task to run aperiodic jobs is a simple idea, but:
- we want to reduce response time for aperiodic jobs
- avoid polling server problem: do not waste unused budget
- avoid deferrable server problem: do not use budget too late
- When does the budget need to expire? When can we safely replenish?
- also relevant: complexity of the implementation

Summary

- modelling maps intended task behaviour to a formal description
  - sets of jobs, dependencies
  - periodic task model
  - sporadic, aperiodic tasks
- scheduling maps formal description to resources
  - admission reasons about feasibility
  - priority vs. criticality
  - preemption, multicore
- task models for soft real-time
- embedding of soft / aperiodic work in servers