Real Time Systems – SS2016

Prof. Dr. Karsten Weronek
Faculty 2
Computer Science and Engineering

Scheduling

The aim for RT-planning

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Find a relation between a set of jobs and
the resources to perform these jobs, that

- all required resources (computation time, memory, devices, etc.) are available to the jobs and
- the achievement of all time-related requirements is garanteed

A Schedule

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

- A schedule or a timetable, as a basic time-management tool, consists of
  - a list of times at which possible tasks, events, or actions that are intended to take place, or/and
  - of a sequence of events in the chronological order in which such things are intended to take place.

The process of creating a schedule
- deciding how to order these tasks and how to commit resources between the variety of possible tasks –

is called scheduling, and
a person responsible for making a particular schedule may be called a scheduler.

Defintion: Schedule

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

For RTS:

Definition:

A schedule of a set of jobs is called feasible(viable) when
each job can be completed with its individual Deadline.

To schedule means
to decide, which process will be processed in which time frame.

But how to find such a feasible schedule ?

Scheduling (non-)optimial Algorithm (Strategy)

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Definition:

An scheduling algorithm is called optimal if
it is able to create a feasible schedule
in those cases in which an useful schedule exists.

A non-optimal scheduling algorithm may not be able
to create an feasible schedule.

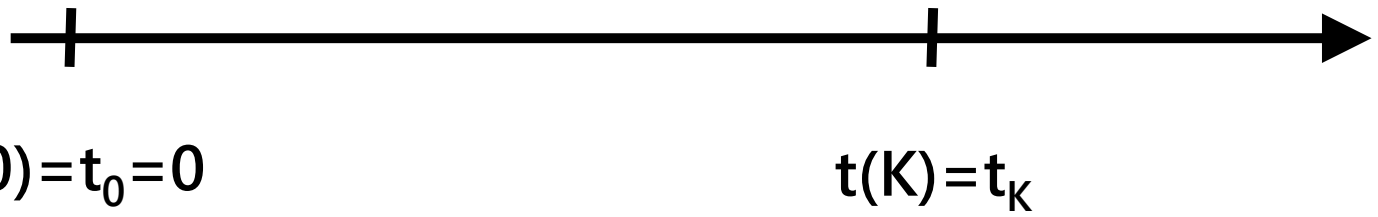
Points in Time, Periods in Time

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Points in time means

there is an event K, that is so short in time, that it can be assumed that the event take no time and can be defined by

a single number t(K) on the **time bar t.**



Periods in time means

there is a defined time frame P, that

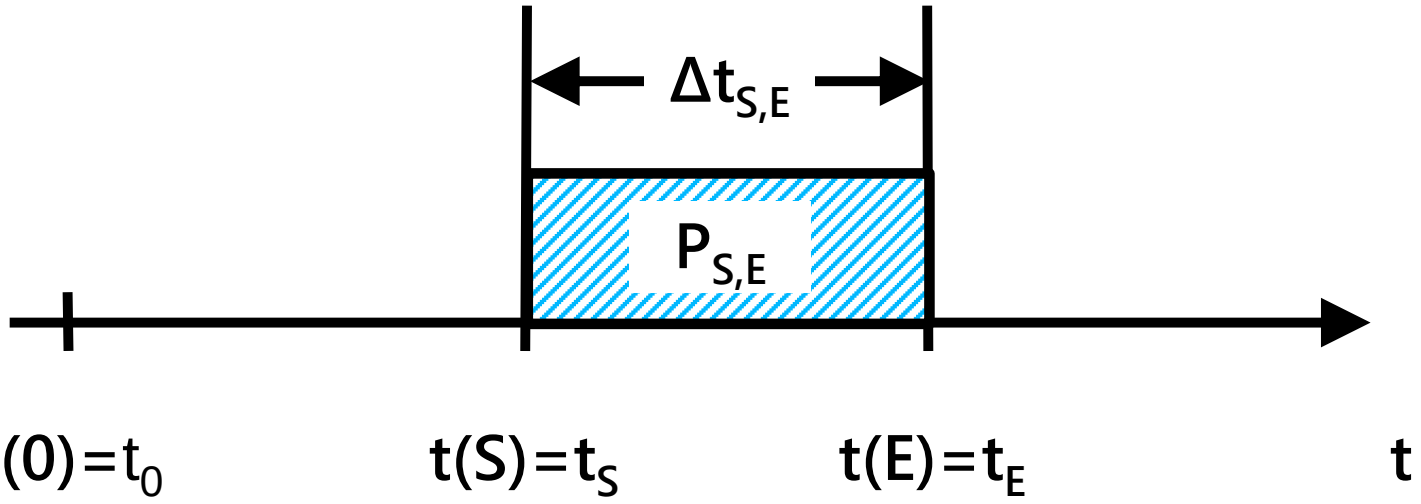
takes some time Δt, and can be defined by

a start point t(S) and an end point t(E) on the time bar.

The period is defined by: $P_{S,E} = P[\, t(S) \mid t(E)\,] = P(\, t_S \mid t_E\,)$

The duration of the time frame P can then be calculated by

$$\Delta t(P_{S,E}) \quad = \quad t(P_E) - t(P_S) \quad = \quad t_E - t_S \quad = \quad \Delta t_{S,E}$$



What is the issue?

Starting from

$$\Delta t(P_{S,E}) = t(P_E) - t(P_S) = t_E - t_S = \Delta t_{S,E}$$

and define $t(P_S) = t_S = 0$ (Start point is set to 0) will become

$$\Delta t(P_{0,E}) = t(P_E) - 0 = t_E - 0 = \Delta t_{0,E}$$

leads to

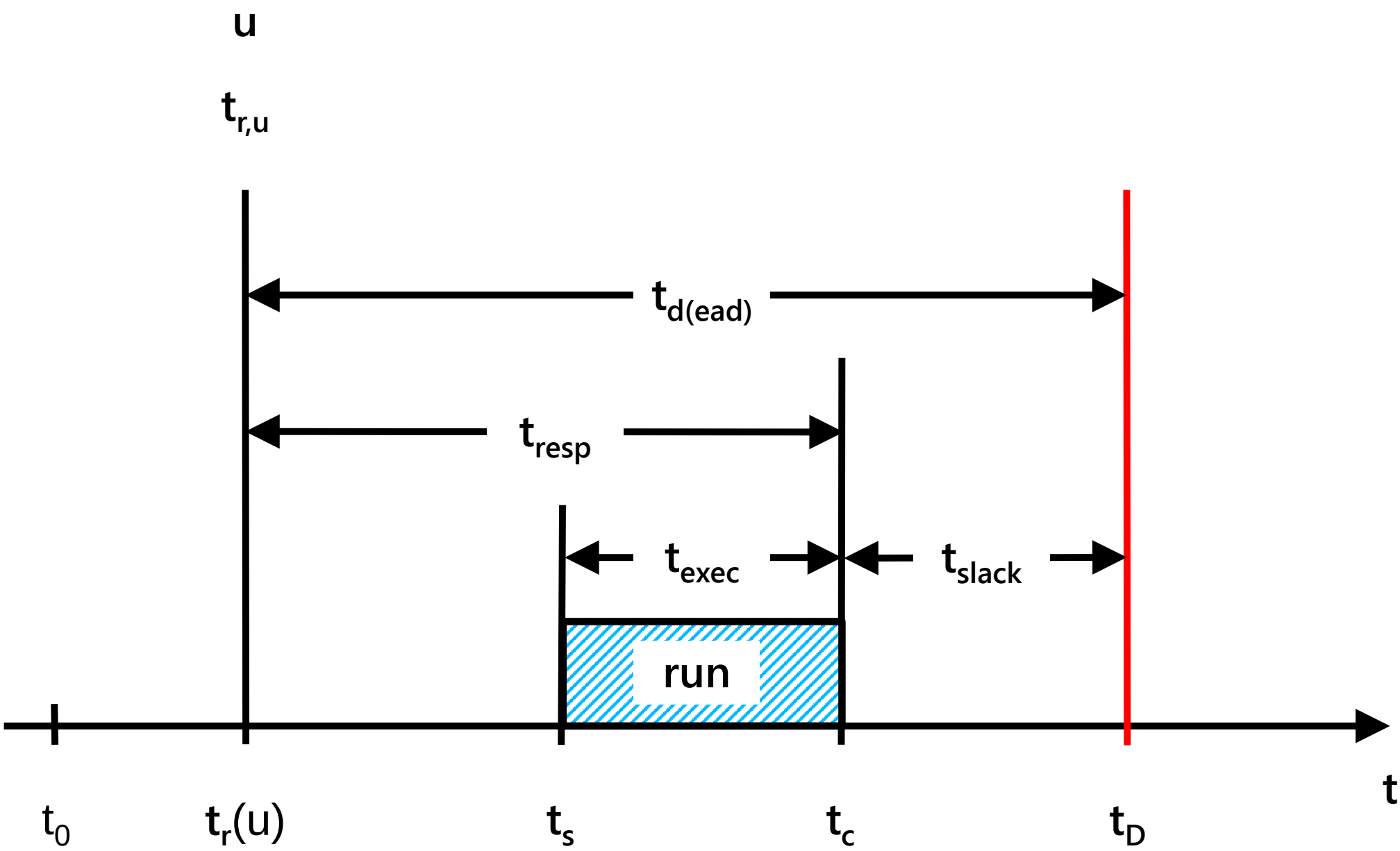
$$\Delta t(P_E) = t(P_E) \quad \text{and:} \quad t_E = \Delta t_E$$

In a lot of documentation (books, scripts, journals, etc.)

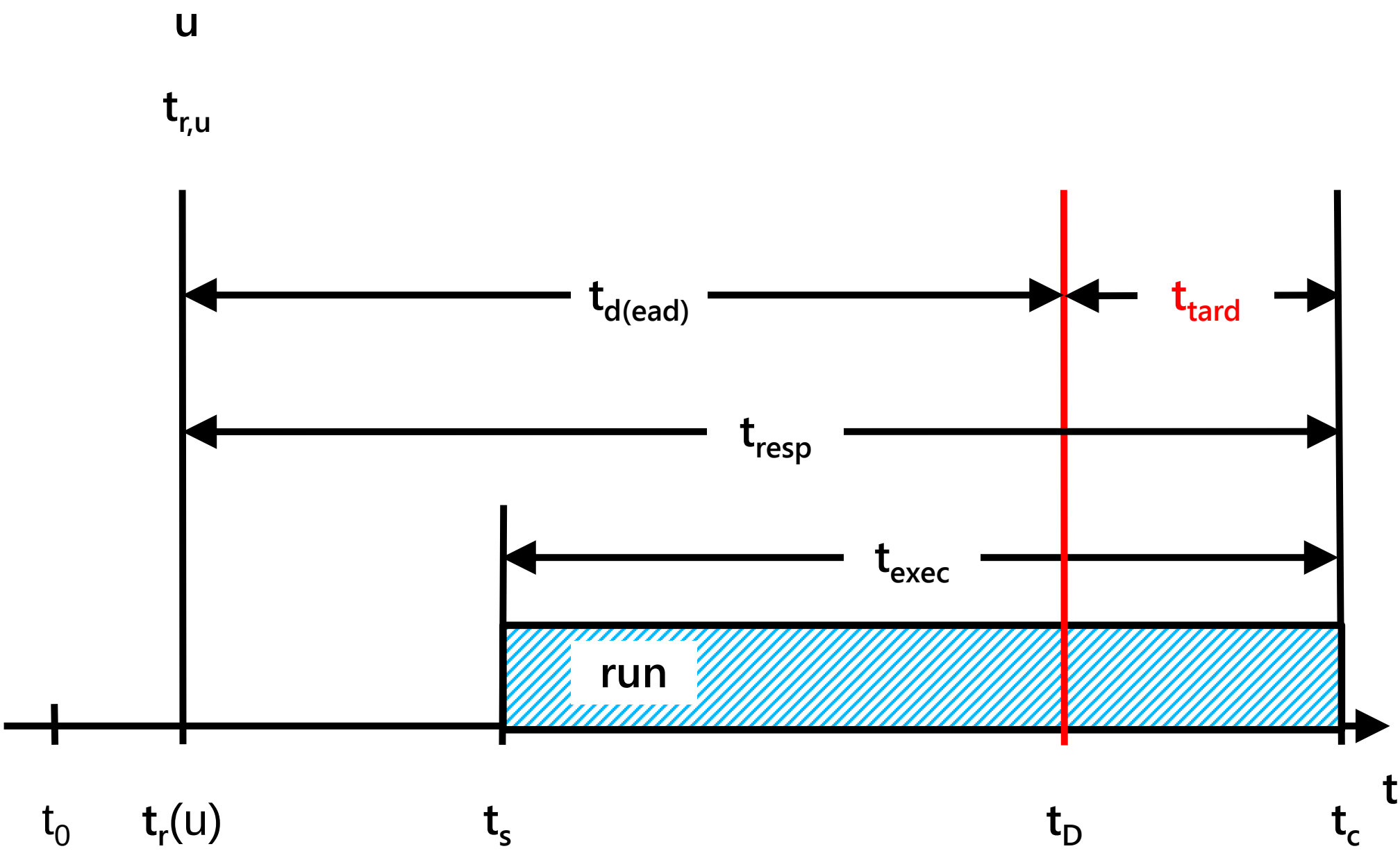
there is no difference between point in time and periods in time.

You have to keep in mind what is meant!

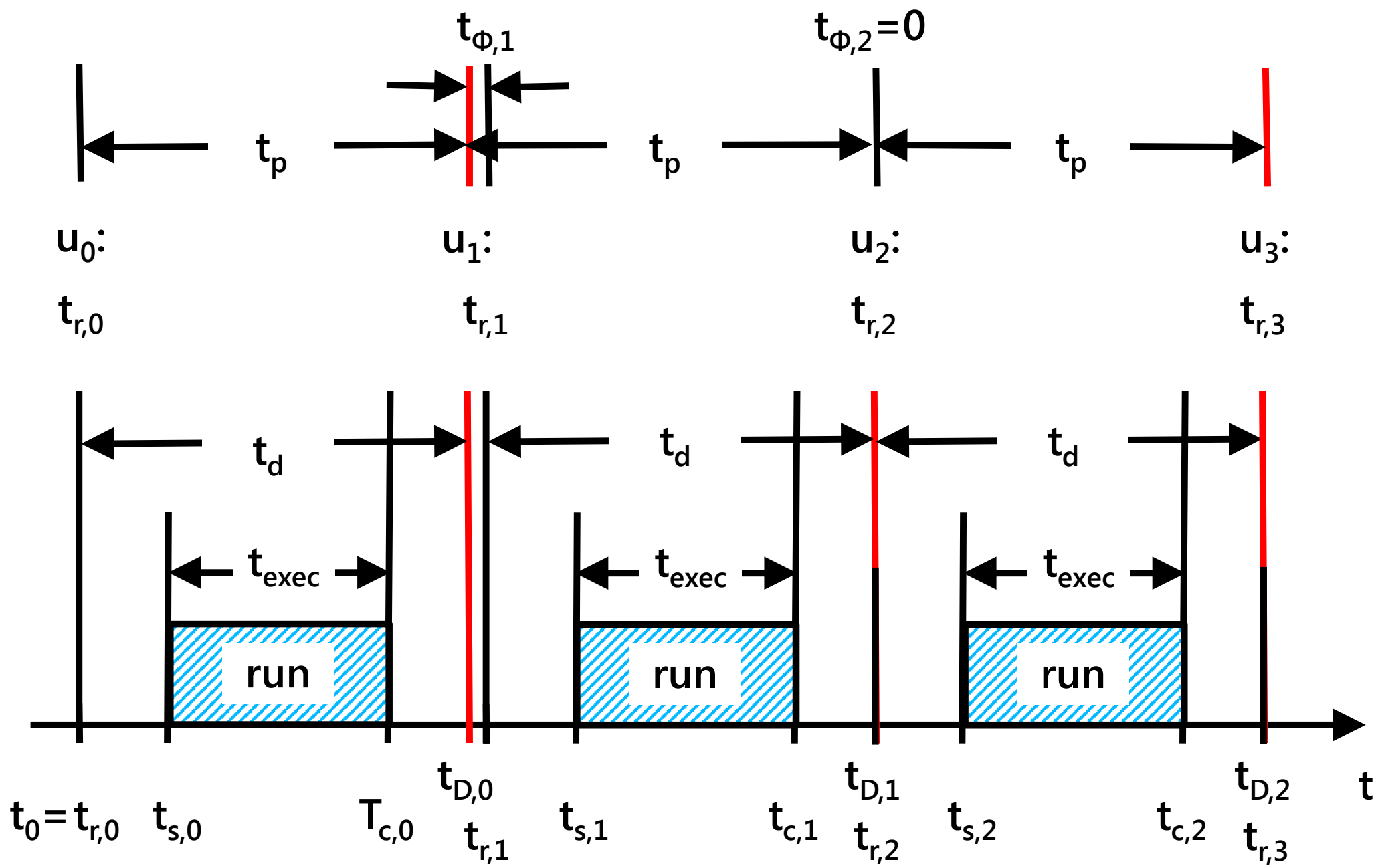
Develop picture on whiteboard



When t_{dead} is missed you get a tardiness



Periodic Tasks with $t0=0=t_{r,0}$



Points in time and time intervals (all named be time!)

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Points in time:

Release time	t_r	time when a job is ready to run
Starting time	t_s	time when job really starts
Completion Time	t_c	time when job is done
Deadline	t_D	time when job has to be completed

Points in time and time intervals (all named be time!)

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Time intervals:

Execution time	t_{exec}	„CPU-time“	$t_c - t_s$
Response time	t_{resp}	„reaction time“	$t_c - t_r$
Tardiness	t_{tard}	„too late“	$t_c - t_D$; for $t_c < t_D = 0$
Slack time	t_{slack}	„time to idle“	$t_D - t_c$ for $t_c < t_D$
Slack time	$t_{d(ead)}$	„dead line“	$t_D - t_c$ for $t_c < t_D$

Warning:

$t_{d(ead)}$ is based on t_r

t_D is based on t_0

Computation-time request

The physical process creates a request for a computation task. This task is identified by a letter (e.g. **u**).

The computation-time request needs a certain a mount of CPU time and has a certain deadline, until the computation must deliver a proper result.

Peridic Requests

The release time is the point in time when the computation-time request appeares.

Release Time: $t_r(u)$ or $t_{r,u}$

In periodically apearing processes the Release Times of requests of the same type can be numbered:

$t_{r,u,1}$, $t_{r,u,2}$, $t_{r,u,3}$, $t_{r,u,4}$, ...

Process Time/Period; Rate

The time difference between two requests of the same type is called **Process Time** or Process Period.

The process time: $t_{p,u}$

The process may vary between: $t_{pmin,u} \leqslant t_{p,u} \leqslant t_{pmax,u}$

For RTS t_{pmax} is not relevant.

The inverse of the period is called rate:

For RTS the maximal Rate is: $r_{max,u} = \frac{1}{t_{pmin,u}}$

Maximum Reaction Time/Deadline

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

The Deadline is the maximum time until a computation-time request needs to be accomplished (i.o.w. needs to be completed in the way that it could be initiated a second time.

In RTS there is a minimal reaction time: t_{dmin}

And a maximum reaction (Deadline): t_{dmax}

Phase

The phase reflects the minimal time delay between a computation-time request and the the time reference point.

Phase: $t_{\Phi min} = \min \{ u(t) - t_0 \}$

In most cases it is assumed to be zero:

$$t_{\Phi min} = 0$$

Execution Time/Computation Time, Runtime,

The execution time is the time that
a computation-time request
is consuming Computer-time.

Is the request for CPU-time only it is equivalent to the CPU-time.

However owing to some latency issues it is slightly higher.

The Execution varies between a
Best Case Execution Time (BCET) and a
Worst Case Execution Time (WCET)

Response Time

The Response Time is defined as the time between the arrival of a computation-time request (Release Time) and the completion time of the computation.

$$t_{resp} = t_c - t_u$$

Don't confuse the Response Time and the waiting time. The waiting is the between the reuquest and the start of execution. So the response includes the waiting time already.

Latency Times

Latency time are the time that is needed for computer internal processing (operation system and processor internal) and leads to couple of time delay between an initiation and the start of the requested task.

There are

- Interrupt-Latency
- Task-Latency
- Kernel-Latency
- Preemption Delay (Verdrängungszeit)

Latencies

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Interrupt Latency

Time between an interrupt request and the start of the related interrupt routine.

Task Latency

Time between the initiation of a task and the start of the task.

Kernel Latency

are OS-internal latencies
owing to blocking sequeces of certain hardware areas.

Preemptive Delay

The delay for stopping and/or removing a currently running code sequence, to load and start a new one (context switch)

Simplification

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

For the scheduling the time will be devided evenly into reasonable time-slices.

For RTS scheduling you aim to describe your problem as a periodic problem to have a finite problem to solve.

If you have a solution for one period you have it for all the time.

For the beginning let assume that all tasks will be independent.

Easy Task model

- Non periodic/aperidoc (three parameters)

  - A: arriving time (of the request)
  - C: computing time
  - D: deadline (relativ to arrival)

  Remark:

  A = $t_A = t_r$ is the release time

  C = $t_{exec} = t_c - t_s$

  D = $t_{execmax}$ = WCET

  Then a computation request/task can be defined as u = (A, C, D)

  A schedule than may be u_1 , u_s , u_1, ... u_n

Scheduling Problems

Given a set of tasks (ready queue)

- Check if the set is schedulable
- If yes, construct a schedule to meet all deadlines
- If yes, construct an optimal schedule
  e.g. minimizing response times

Tasks with the same arrival time

Assume a list of tasks

(A1, C1, D1), (A2, C2, D2) . . . (An, Cn, Dn)

Assume: A1 = A2 = . . . = An = 0 (same arrival time at 0sec)

The you can leave off A an the task list is

→ (C1, D1), (C2, D2) . . . (Cn, Dn)

→ Is there a feasible schedule?

→ How to find a feasible schedule?

→ May be, there are many feasible schedules!

Earlist Due Date fist (EDD) by Jackson 1955

EDD: order tasks with nondecreasing deadlines.

EDD is a simple form of EDF (earliest deadline first).

Example: (1,10)(2,3)(3,5)

Schedule is (2,3)(3,5)(1,10)

EDD is optimal

if EDD can‘t find a feasible schedule

EDD: Schedulability test

IF C1+C2+…+Ck ≤ Dk for all k ≤ n
for the schedule with nondecreasing ordering of deadlines,
then the task set is schedulable

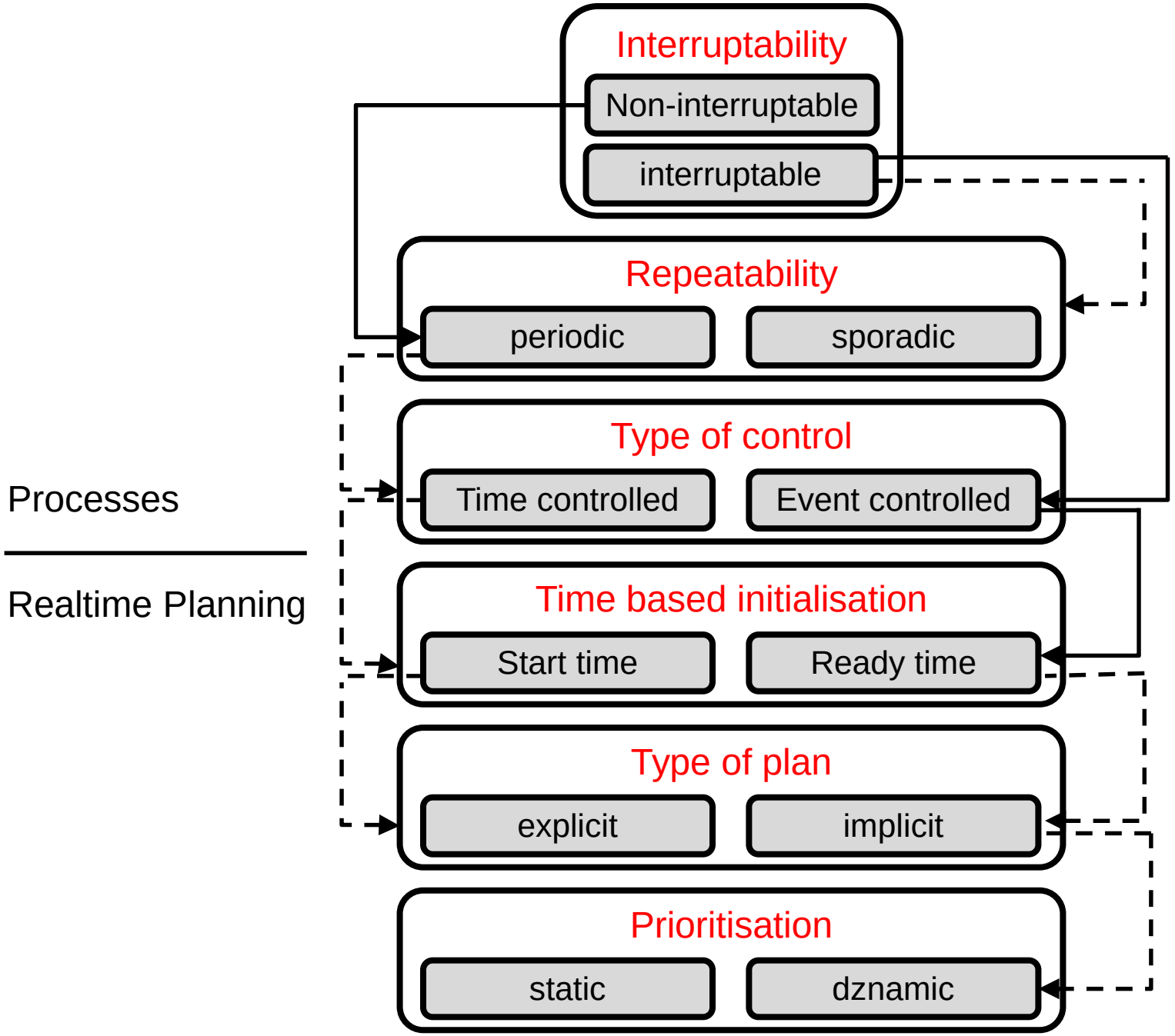
Response time for task i: Ri = C1 + … + Ci

Wenn

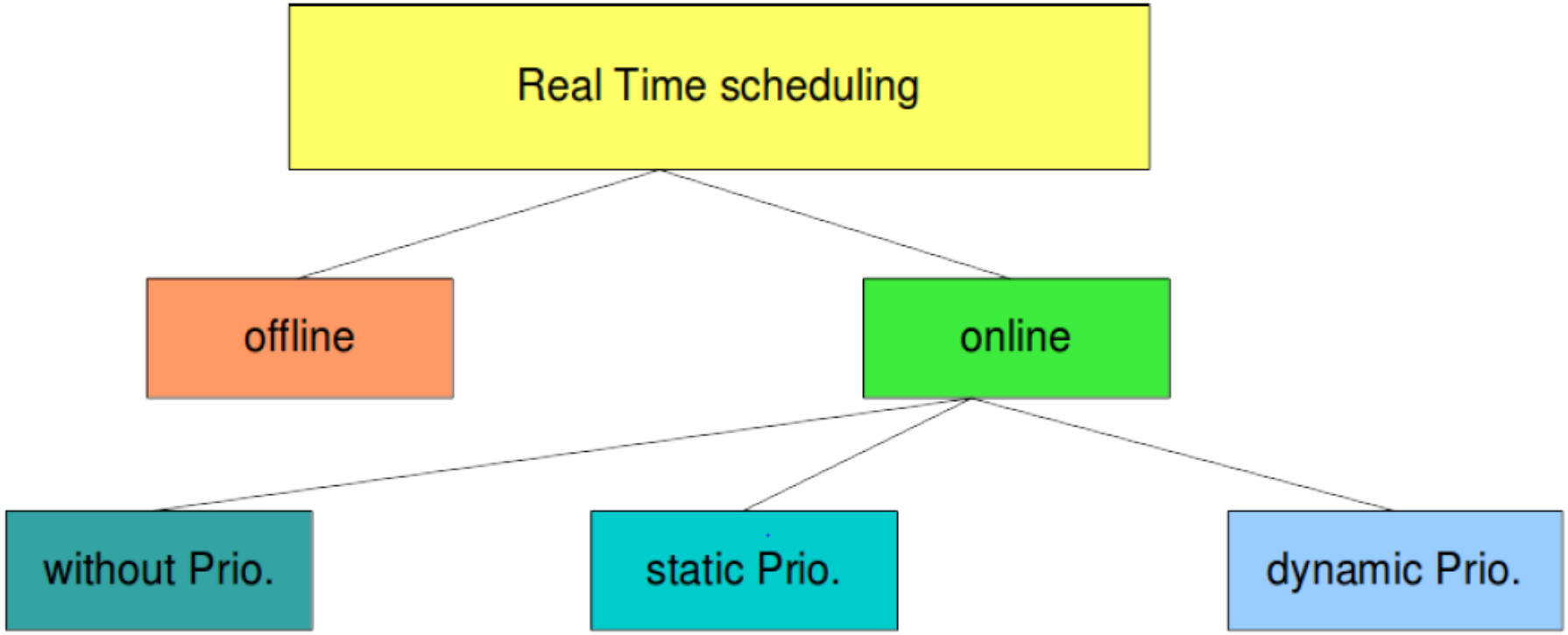
für einen (Ablauf-)Plan mit
nicht-fallender Sortierung nach Deadlines,
C1+C2+…+Ck ≤ Dk für alle k ≤ n
dann ist der Plan ausführbar.

Die Antwortzeit für den task i: Ri = C1 + … + Ci

Combination of Categories of Processes and Plannings



Classification of RT-scheduling Algorithms



FIFO (First in First Out)
Round Robin
SJF (Shortes Job First)
SRT (Shortes Remaining Time)

RMS (Rate Monotonic Sch.)
DMS (Deadline Monotonic Sch.)

EDF (Earliest Deadline First)
LRT (Latest Release Time)
LST (Least Slack Time first)

EDF earliest deadline first

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

EDF has dynamic priorities.

Each tasks priority is calculated depending on deadline:

- An executable task with a shorter deadline has always a higher priority than the other tasks.
- An executable task with higher priority will always interrupt a task with lower priority
- A task with the same priority is not interrupted.

Basic Rate Monotonic Analysis (BRMA)

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Assumptions of rate monotonic Scheduling and Analysis

- All the threads are periodic
- There is no interaction, blocking due to unavailability resources, priority inversion among the threads
- Thread switching in the system is instantaneous
- Each Thread has a constant execution time and the execution time does not change with time
- The deadline for a thread is starting o next period of the thread.
- The priority of each thread is determined by its period. The shorter the execution time period of a thread, the higher the priority
- All threads in the system are equally critical.
- Aperiodic threads are limited to system initialisation and failure recovery and do not have hard deadlines.

Rate Monotonic Scheduling (RMS)

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Basic Rate monotonic analysis formally proves whether a given set of real-time threads can be schedulable to meet their deadlines if the threads were scheduled using rate monotonic scheduling.

RMS is a priority based preemptive scheduling, in which the threads with the lower periods should be given higher priorities and vice versa. In the RMS, the priority is solely determined by its period only with the rule
"lower the period higher the priority and vice versa"

There are some schedulability test for RMS using BRMA.

There is also a method called "Extended RMS". This removes some of the assumptions.

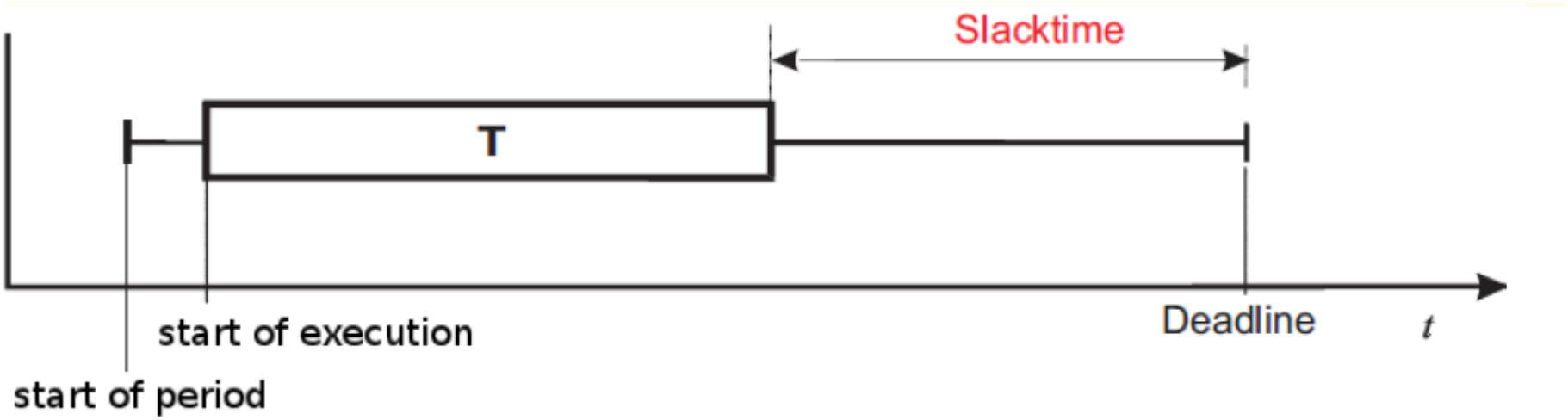
You need to know: If a system can not be schedulable by RMS, the system can not be schedulable with an other static priority assignment

Least Slack-time First

LSF has variable priorities.

Priority is calculated from difference of deadline and (remaining) execution time (Slacktime),

- Shorter Slacktime = higher priority,
- An executable task with higher priority will always interrupt a task with lower priority.



Schedulabilitytest

Before you start to schedule your system you have to check that there exists at least one feasible schedule.

Depending on the set up you have to go through different schedulability tests.

There are two major types of test

- necessary tests (notwendig)
- sufficient tests (hinreichend)

Necessity Tests

Necessary means:

- if one of the appropriate necessity test fails then there is no feasible schedule!
- If one or more or all necessity test are fulfilled then there may be or may not be a feasible schedule!

Sufficient Test

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Sufficient means:

- if you find at least one sufficient necessity test, than the task package is feasible schedulable.

- If you can't find a suffice necessity test, than a feasible schedule may exist or may not exists.

Load test is a necessity requirement:

- make all possible execution requests i periodic
- take this as the worst case maximum for the load

Load of a task:

$$U_i = \frac{e_i}{D_i}$$

System load:

$$U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{e_i}{D_i}$$

Where ei is the execution time and Di time untilexpiration time of dead,

Where Pi = Di

Load U = 1 means the processor never idles

Load U > 1 there is no feasible schedule

Load U < 1 means: this test does not exclude that a feasible schedule may exist

Test for Rate Monotonic Scheduling (RMS)

Schedulability Test:

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

Where n is the number of periodic tasks

Example: (n=2: p_1=(6,3) and p_2=(10,5) [(D_i , e_i)]

with :

$$U_i = \sum_{i=1}^{n} \frac{e_i}{D_i}$$

$$U = \sum_{i=1}^{2} U_i = \frac{3}{6} + \frac{5}{10} \not\leq 2 * (2^{\frac{1}{2}} - 1) = 0,828$$

States of tasks

If a task is non-interruptible it can have only two(three) states:

1. Ready (waiting to start)
2. Running
3. (Completed)

There is only one way to change states:

ready (may be not necessary) → Running → Completed

When you use an explicit plan there are only the states:

running → completed

States of tasks

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

For interruptible tasks there are four (five) states:

1. Ready
2. Running
3. Blocked
4. Suspended
5. (Completed)

Running

When a task is actually executing it is said to be in the Running state. It is currently utilising the processor. If the processor on which the OS is running only has a single core then there can only be one task in the Running state at any given time.

Ready

Ready tasks are those that are able to execute (they are not in the Blocked or Suspended state) but are not currently executing because a different task of equal or higher priority is already in the Running state.

Blocked

A task is said to be in the Blocked state if it is currently waiting for either a temporal or external event. For example, if a task calls “sleep()” it will block (be placed into the Blocked state) until the delay period has expired - a temporal event. Tasks can also block to wait for queue, semaphore, event group, notification or semaphore event.

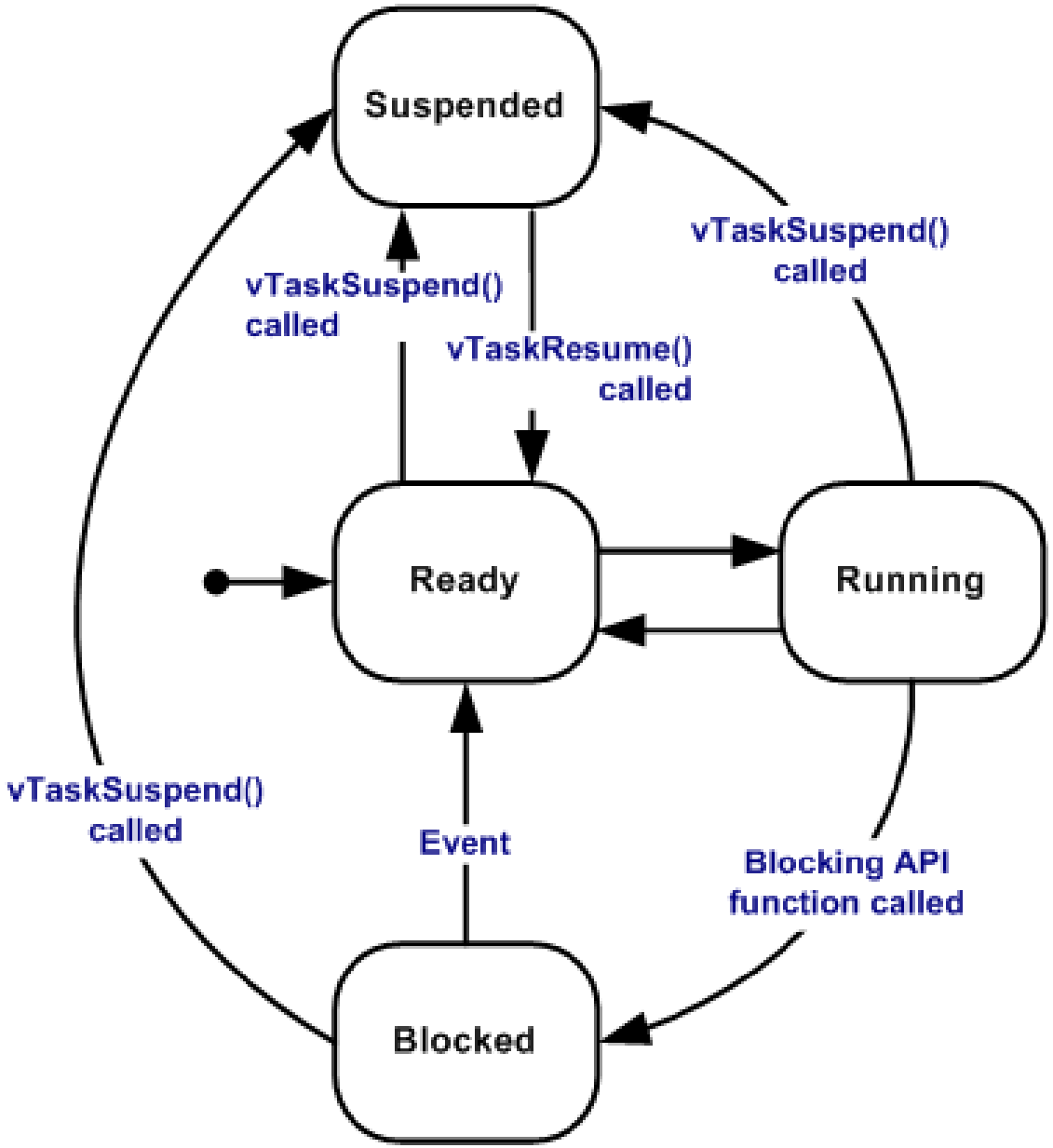
Tasks in the Blocked state normally have a 'timeout' period, after which the task will be timeout, and be unblocked, even if the event the task was waiting for has not occurred.

Tasks in the Blocked state do not use any processing time and cannot be selected to enter the Running state.

Suspended

Like tasks that are in the Blocked state, tasks in the Suspended state cannot be selected to enter the Running state, but tasks in the Suspended state do not have a time out. Instead, tasks only enter or exit the Suspended state when explicitly commanded to do so through the “Suspend()” and “Resume()” API calls respectively.

Task-State- and Task-State-Transition Diagram



Not all theoretical State-Transitions are realised (RTOS)

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Ready	→	Suspended	:	suspend()
Ready	→	Running	:	run()
~~Ready~~	~~→~~	~~Blocked~~		
Suspended	→	Ready	:	resume()
~~Suspended~~	~~→~~	~~Running~~		
Suspended	→	Blocked		
Running	→	Ready	:	yield()
Running	→	Suspended	:	suspend()
Running	→	Blocked	:	block by OS-API
Blocked	→	Ready	:	by event (e.g. timer)
Blocked	→	Suspended	:	suspend()
~~Blocked~~	~~→~~	~~Running~~		

Tasks are initiated by e.g. “start” to queue in the ready state

“Ready-State is key”

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

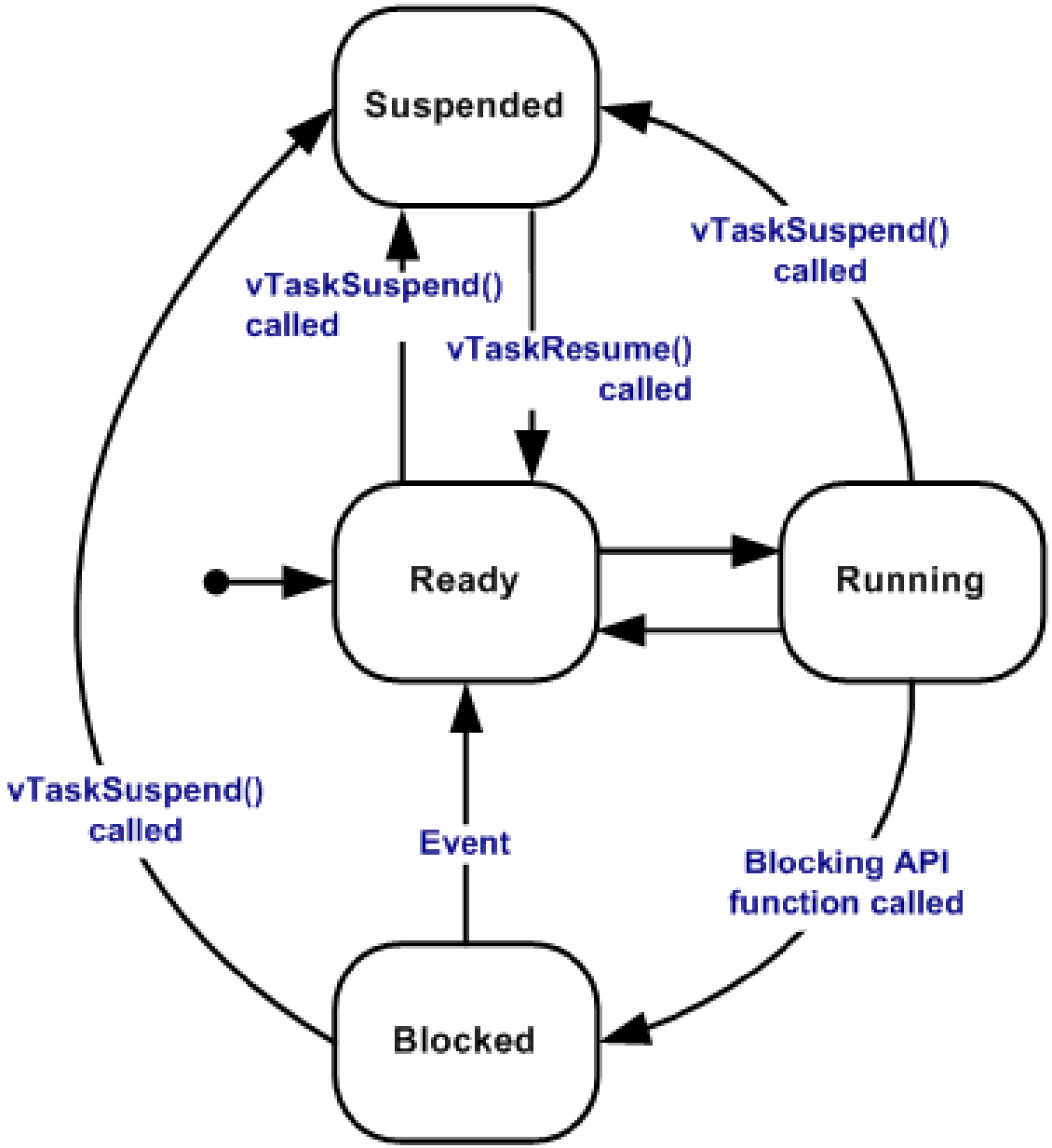
Since the ready State has a central position (most arrows) it is the point where the system / system developer can manage the time behaviour best.

On the other hand the CPU (running state) is the resource to manage.

Therefore scheduler of a Operating System is managing the tasks in the ready state and the running state.

The scheduler decides which tasks is the next to run and it decides (using the scheduling algorithm) when to interrupt a running task and suspend it. All other state transitions are managed by other components of the OS or by the programme/programmer.

Task-State- and Task-State-Transition Diagram



Some “Wordings”: Program and Process

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Program/Programme

A Program contains the instruction.	A process is a sequence of instructions.
A Program is a static entity made up of Program statements.	A process is a dynamic entity that is Program in execution
A Program is a passive part. It does’t give any result but gives a result after starting its execution and becomes process.	A Process is an active part. During execution it gives the result.
A Program is stored on disk.	A Process is store in memory.
A Program does not compete for Computing resources.	A Process compete for computing Resources like CPU-time or memory.

Process Definition

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Process is that executing unit of computation,

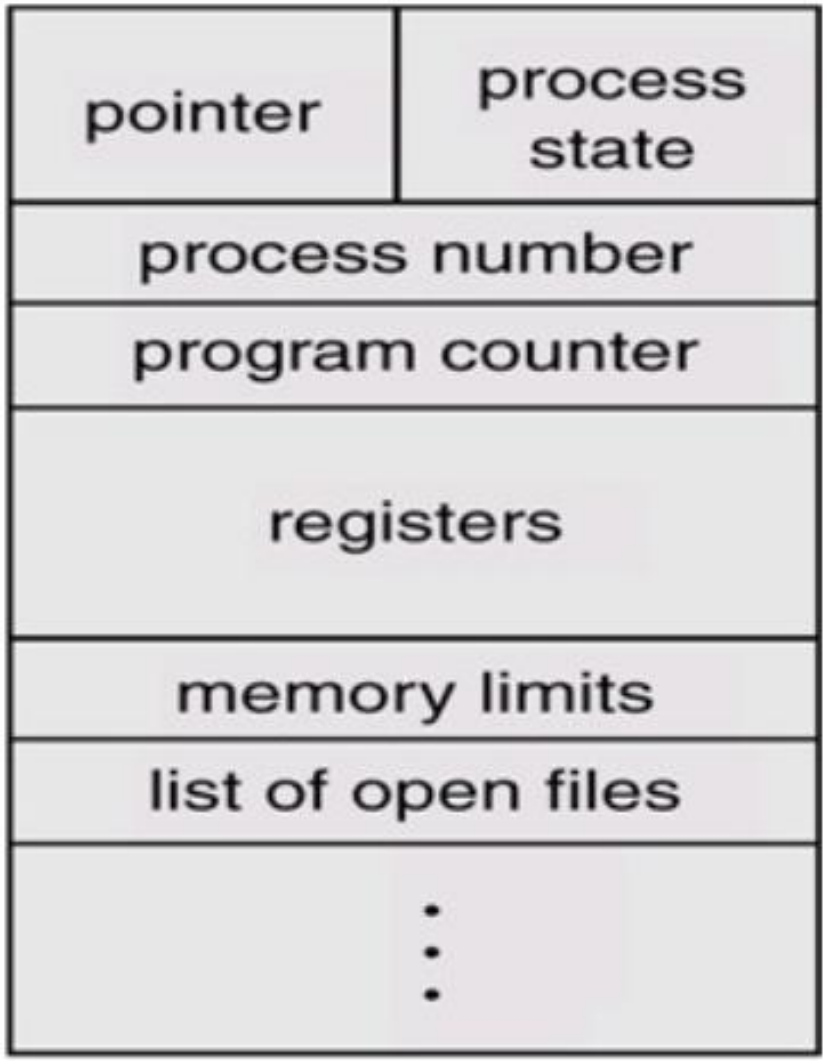
which is controlled by some processes of the OS

- For a scheduling-management mechanism that lets it execute on the CPU
- For a resource-management mechanism that lets it use the system-memory an other system-resources such as network, file , display or printer
- For access control mechanism
- For inter-process communication
- Concurrently

Application program can be said to consist of a number of processes

Process control Block

- Information about each and every Process in our computer is stored in the Process Control Block (PCB)
- Created when a user creates a Process
- Removed from the system when the process is terminated
- Stores in protected memory area of the kernel



PCB Content

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

- Process State: -- Information about various process states such as new, ready , running, waiting, etc.
- Program Counter: -- It shows the address of the next instruction to be executed in the process
- CPU registers: -- There are various registers in the CPU such as accumulators, stack pointer, working register, instruction pointer. The PCB stores the state of all these register when CPU switch from one process to another process.
- CPU Scheduling information : -- It includes process priority, pointer to the ready queue and device queue, and scheduling information.
- Accounting information: -- It includes the total CPU time used, real time used, process number etc.
- I/O status information: -- It includes the list of I/O devices allocated to the process. It also includes the list of opened file by process in disk. File is opened either for read of write.
- Memory-management information: -- The PCB stores the value of base and limit registers, address of page table or segment table, and other memory management information.

Context Switching

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

- When the CPU switches from one process to another process, the CPU saves the information about one process into the PCB (Process Control Block) and ‘ then starts the new process.
- The present CPU registers, which include the program counter and the stack pointer are called context
- When context is saved on the PCB pointed process-stack and register-save area addresses, then the running process stops.
- The other process context now loads and that process runs – this means the context has switched!
- A Context switch is purely overhead because the system does not perform any useful work while the context switches.
- Context switch time are highly dependant on the hardware. Its speed varies from machine to machine depending on the memory speed, registers to be copied and the existence of special instructions.

Thread (1/2)

Two processes can run at the same time but they do not share memory. Suppose we provided software entities that could run at the same time but also share memory. Such entities exist in most of the actual OS. They are called threads. A thread has some of the characteristics of a process, but it is possible to have threads sharing the same memory space.

Many software that runs on desktop PCs are multithreaded. An application typically is implemented as one or more separated processes with several threads.

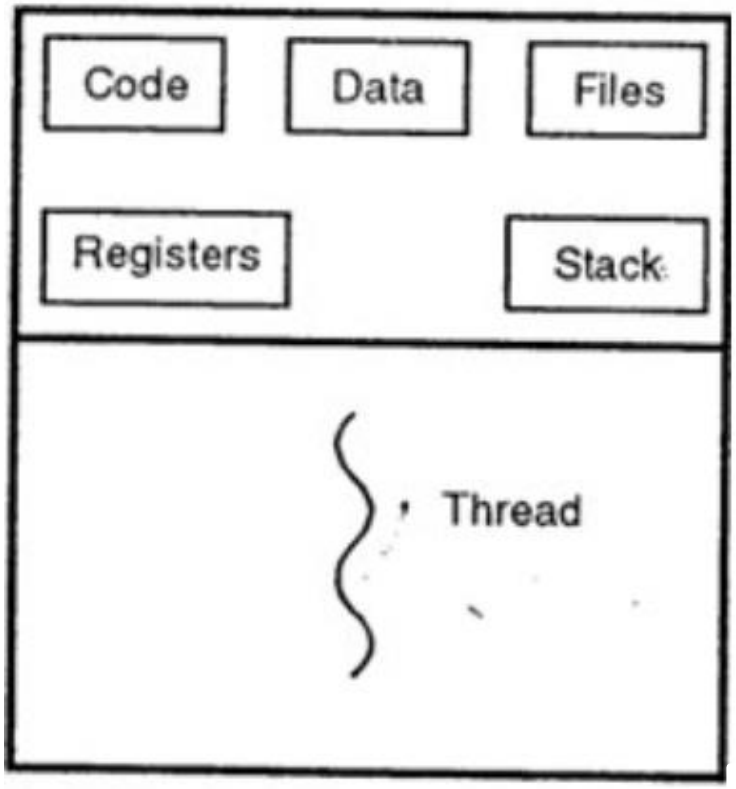
Thread (2/2)

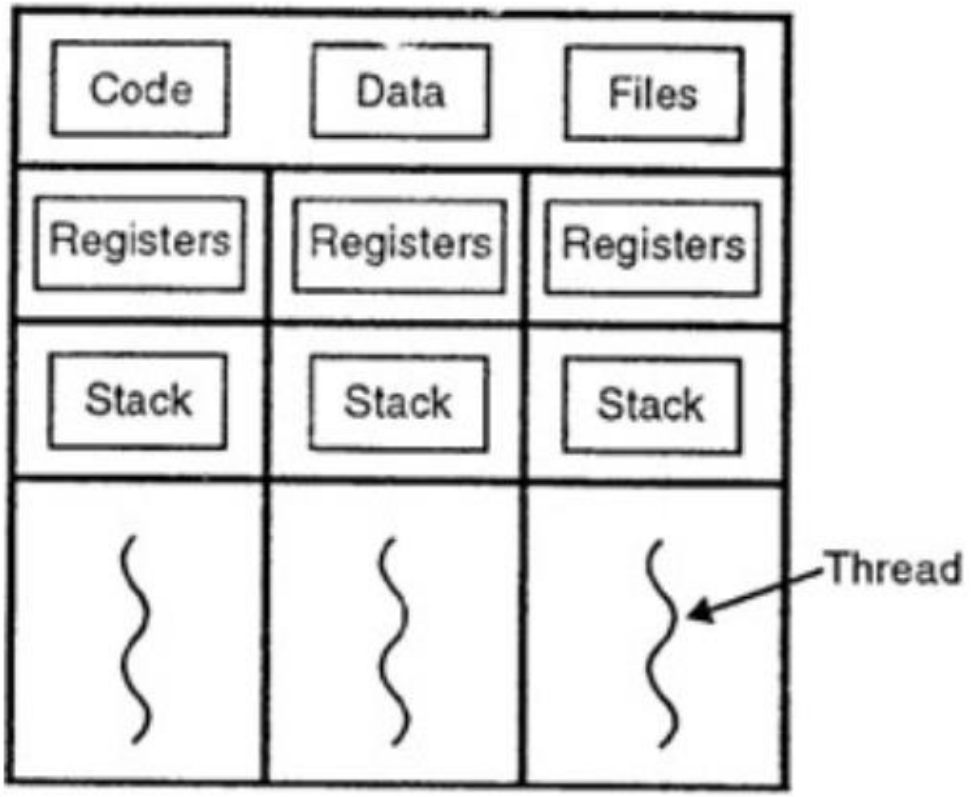
- A thread can either
  - be a sub-process within a process (kernel level thread) or
  - a process within an application program (user level thread)
- A thread does not call another thread to run

- A thread is a process or sub-process within a process that has its own program counter, its own stack pointer, and stack, its own priority parameter for its scheduling by thread-scheduler, and its own variables that load into the process registers on context swithing and is processed concurrently along with other threads

Single-threaded and multithreaded Processes

- A traditional heave-weight process (a kernel-level controlled entity) has a single thread of control. If the process has multiple thread of control, it can do more than one task at a time.
- A Thread is the basic unit of CPU utilisation
- Each thread has independent parameters – a thread ID, a program counter, a register set, and a stack, priority and its present status





Process/ Thread

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Process	Thread
Process is considered heavy	Thread is considered light weight
Unit of Resource Allocation and of Protection	Unit of CPU utilisation.
Process creation is very costly in terms of resources	Thread creation is very economical
Program execution as process is relatively slow	Programs executing using threads are comparatively faster
Process cannot access the memory belonging to another process	Thread can access the memory area belonging to another thread within the same process
Process switching is time consuming	Thread switching is faster
One Process can contain several threads	On thread can belong exactly to one process

Task

- Task – term used for the process in the RTOS e.g. for e.g. an embedded system.
- An application program consists of the tasks and the task behaviours in various stats that are controlled by the OS.
- A task is like a process or thread in an OS.
- Runs when it is scheduled to run by the OS (kernel), which gives the control of the CPU on a task request (system call) or a message.
- Runs by executing the instructions and the continuous changes of its stat takes place as the program counter changes.
- A task – an independent process. No task can call another task.

Task

- Includes the task context and TCB
- TCP – A data structure having the information that the OS is using to control the process state.
- Stores in protected memory area of the kernel
- Consists of the information about the task state

In other words:

Tasks are embedded program computational units that run on a CPU under the state-control using a task control block and are processed concurrently.

The task has the following parameter:

- Task-ID, e.g. a number
- Task name (sometimes)
- Task Priority
- Parent task (if any)
- Child tasks (if any)
- Address to the next task's TCB of the task that will run next

A Task is a process with one or multiple threads

Task = process + 1 Thread (traditionally called process(PCL)/jobs(JCL)

Task = process + >1 Thread (nowadays called Task)