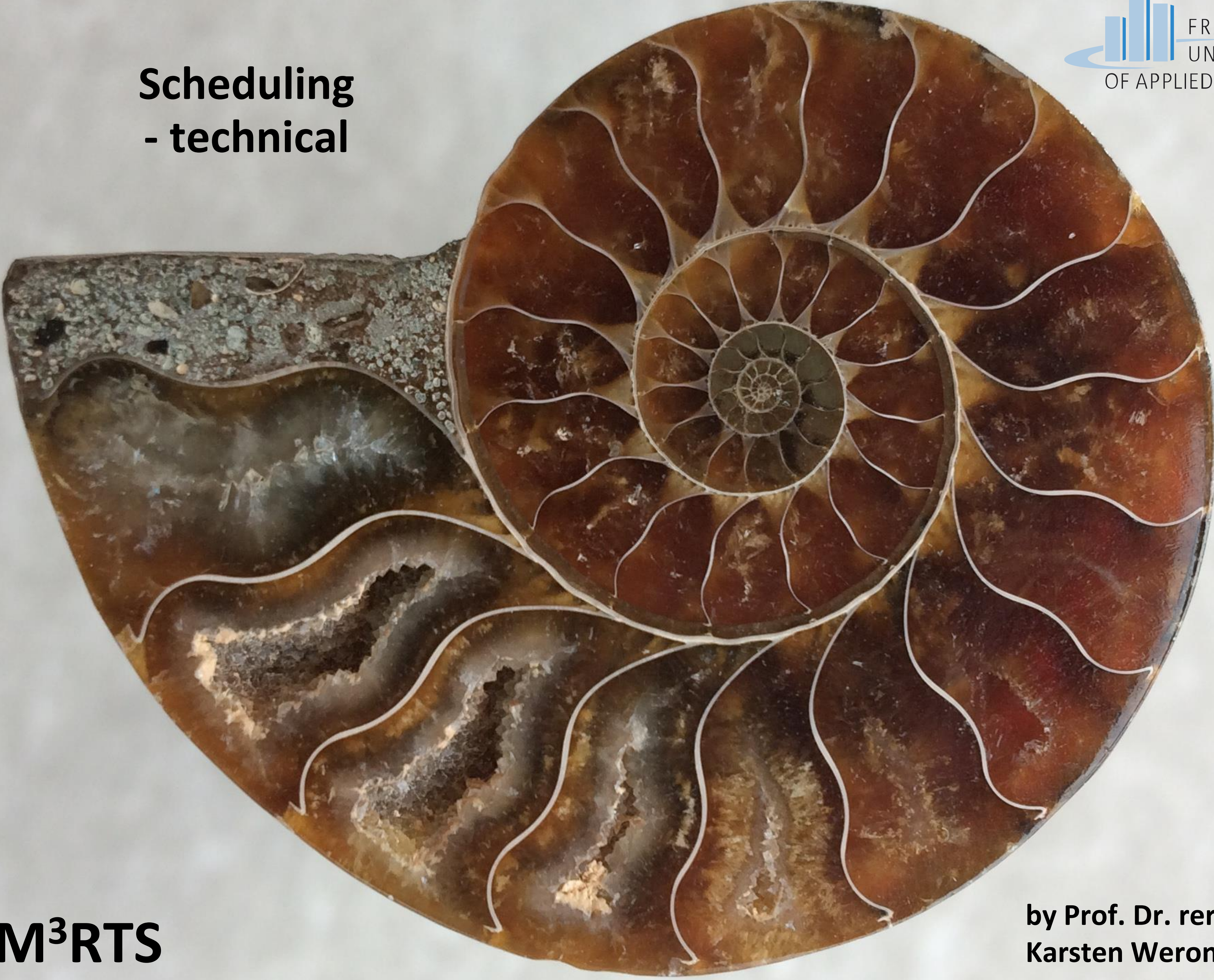
Scheduling - technical

FRANKFURT UNIVERSITY OF APPLIED SCIENCES



M^{3}RTS

by Prof. Dr. rer. nat. Karsten Weronek

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

For interruptible tasks there are four (five) states:

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

States

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Running

When a task is actually executing it is said to be in the Running state. It is currently utilizing 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.

States

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.

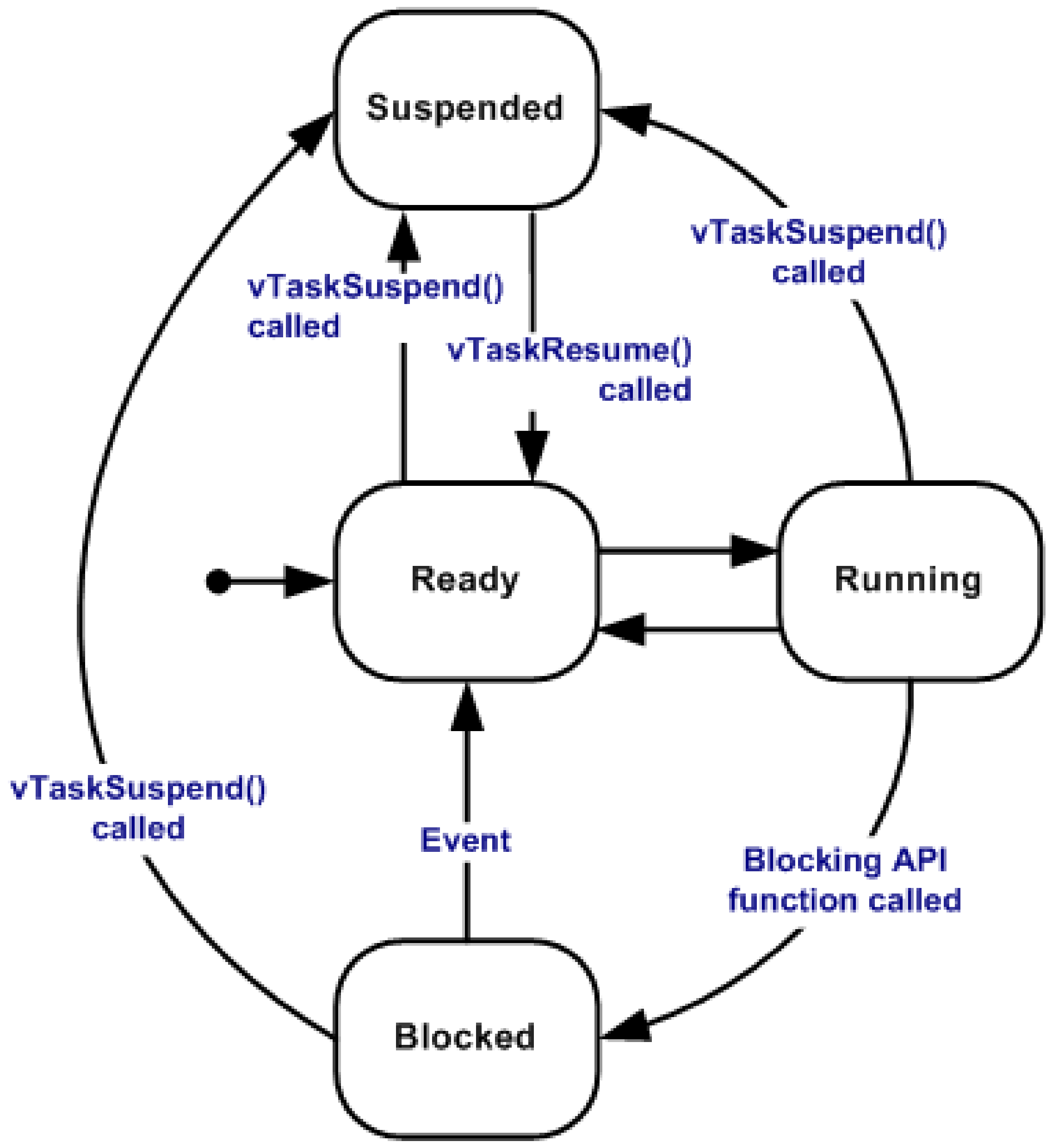
Tasks in the Blocked state do not use any processing time and cannot be selected to enter the Running state.

States

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"

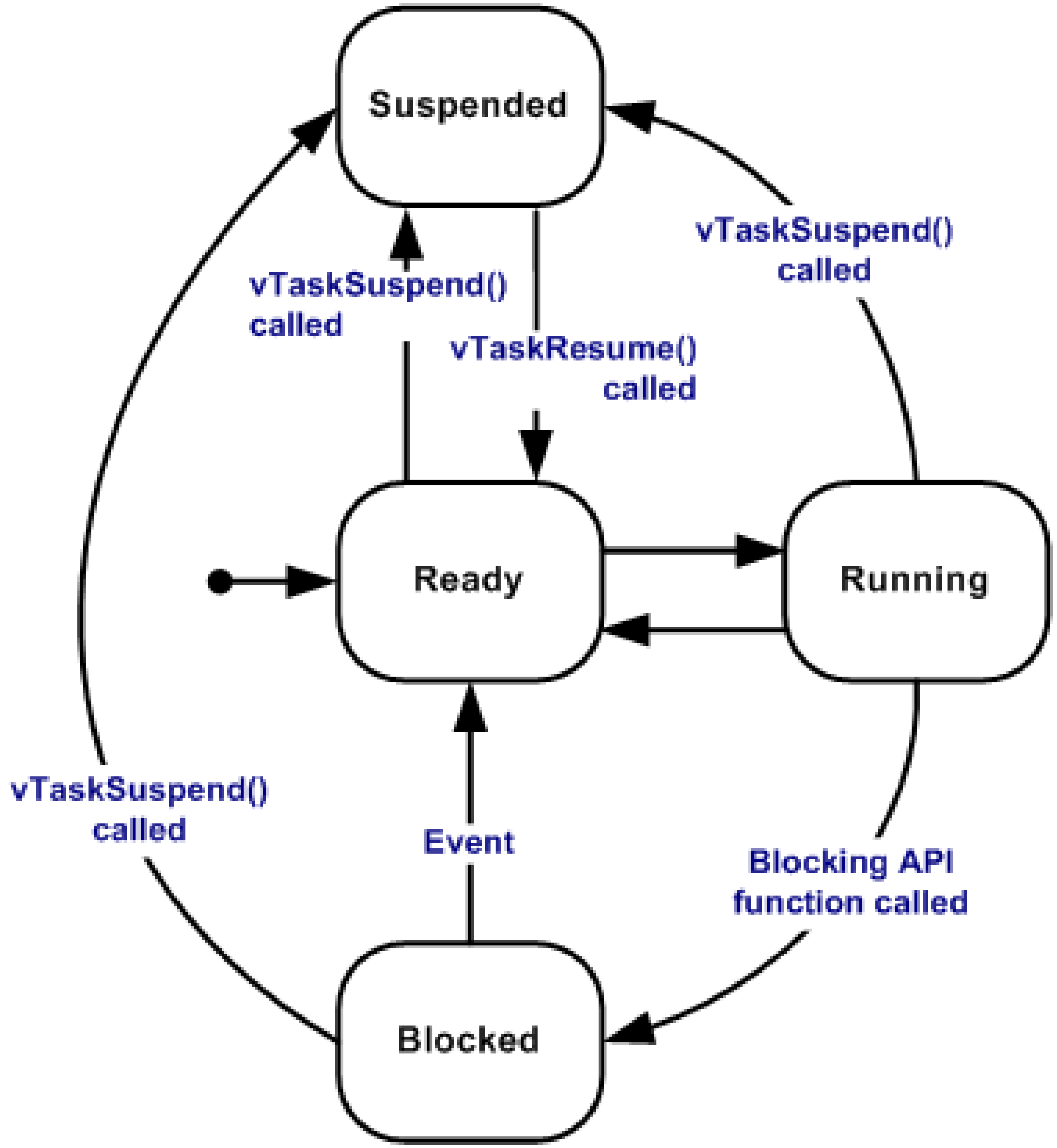
Since the ready State has a central position (most arrows) it is the point where the system / system developer can manage the time behavior best.

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 program/programmer.

Task-State- and Task-State-Transition Diagram



Some "Wordings": Program and Process

Program (Programm)

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

Process is that executing unit of computation,

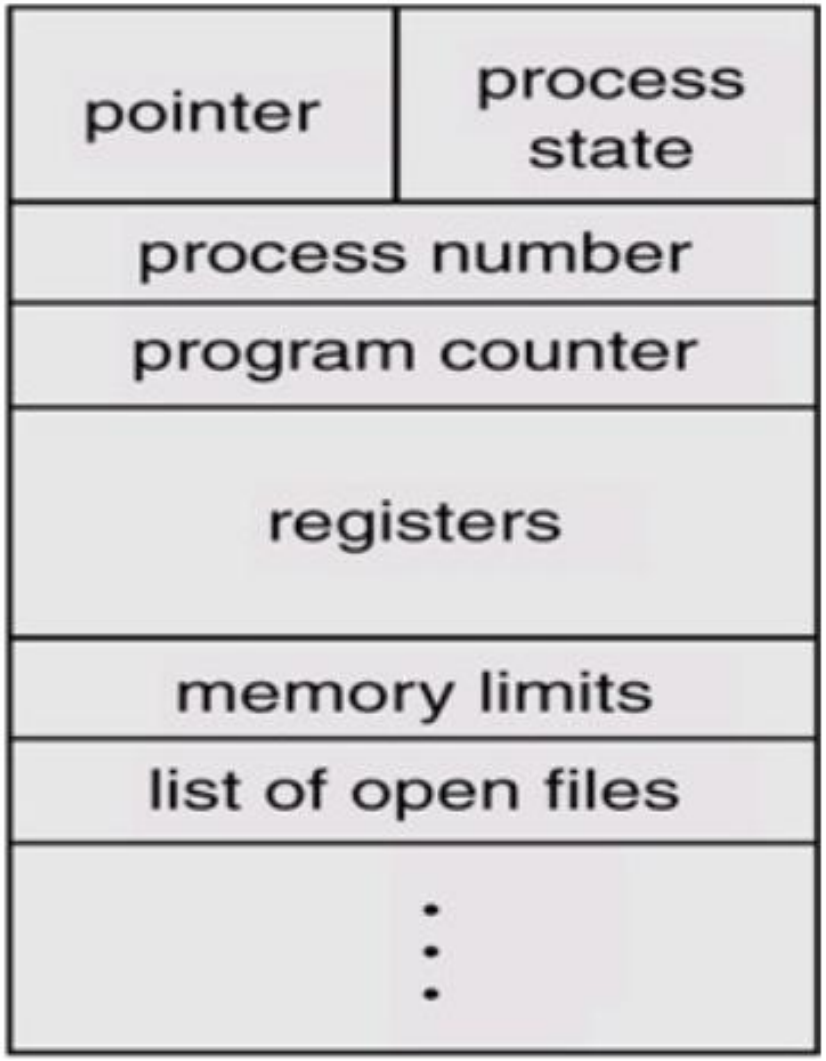
which is controlled by some processes of the OS

- 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 (techn.) user creates a process
- Removed from the system when the process is terminated
- Stores in protected memory area of the kernel



PCB Content (PID)

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.
- others like (Interprocess Communication, Privileges, PPID, etc.)

Context Switching

- 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.

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

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 sequences 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)

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

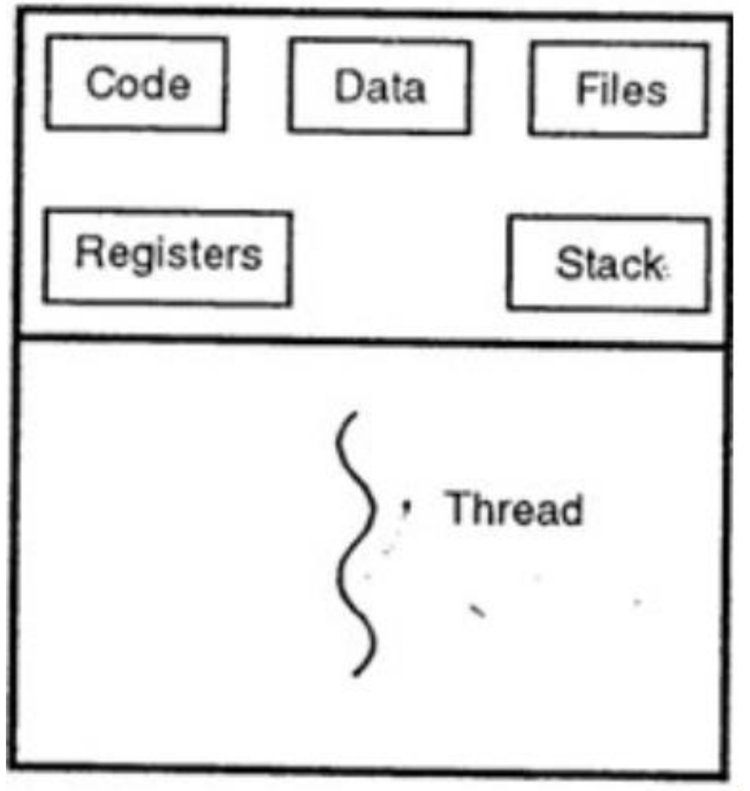
TCB (Thread Control Block) (SPID, LWP) Content

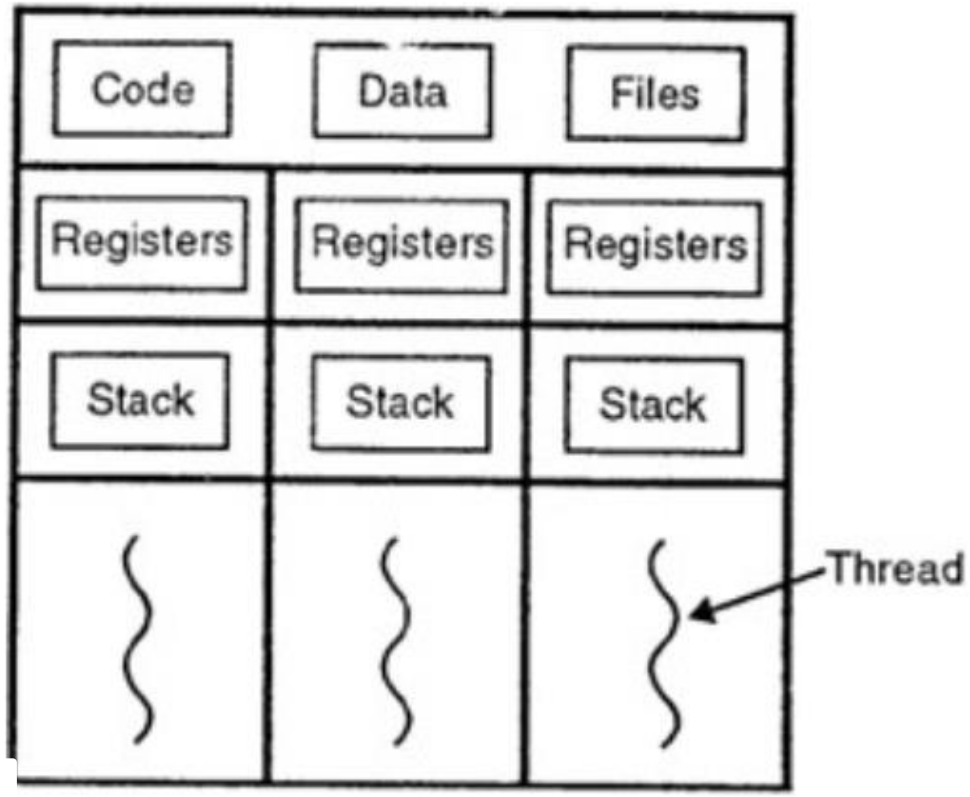
- Thread State: --
  Information about various process states such as new, ready , running, waiting, etc.
- Program Counter: --
  It shows the address of the next instruction to be executed in the process
- CPU registers: -- There are various registers in the CPU such as accumulators, stack pointer, 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.
- 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.

- Pointer to the Process Control Block (PCB)

Single-threaded and multithreaded Processes

- A traditional heavy-weight process (a kernel-level controlled entity) has a single thread of control. If the process has multiple threads 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

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 (fat word with different meanings)

- 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 statuses 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.

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.

Task

Task can be anything…

Task

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)

Task (fat words need to be clarified!)

If you read the word task:

you should figure out what the author means
by getting aware of the context!