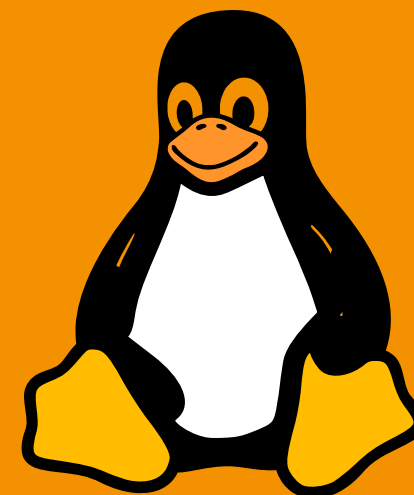




Prof. Dr. Florian Künzner

Technical University of Applied Sciences Rosenheim, Computer Science

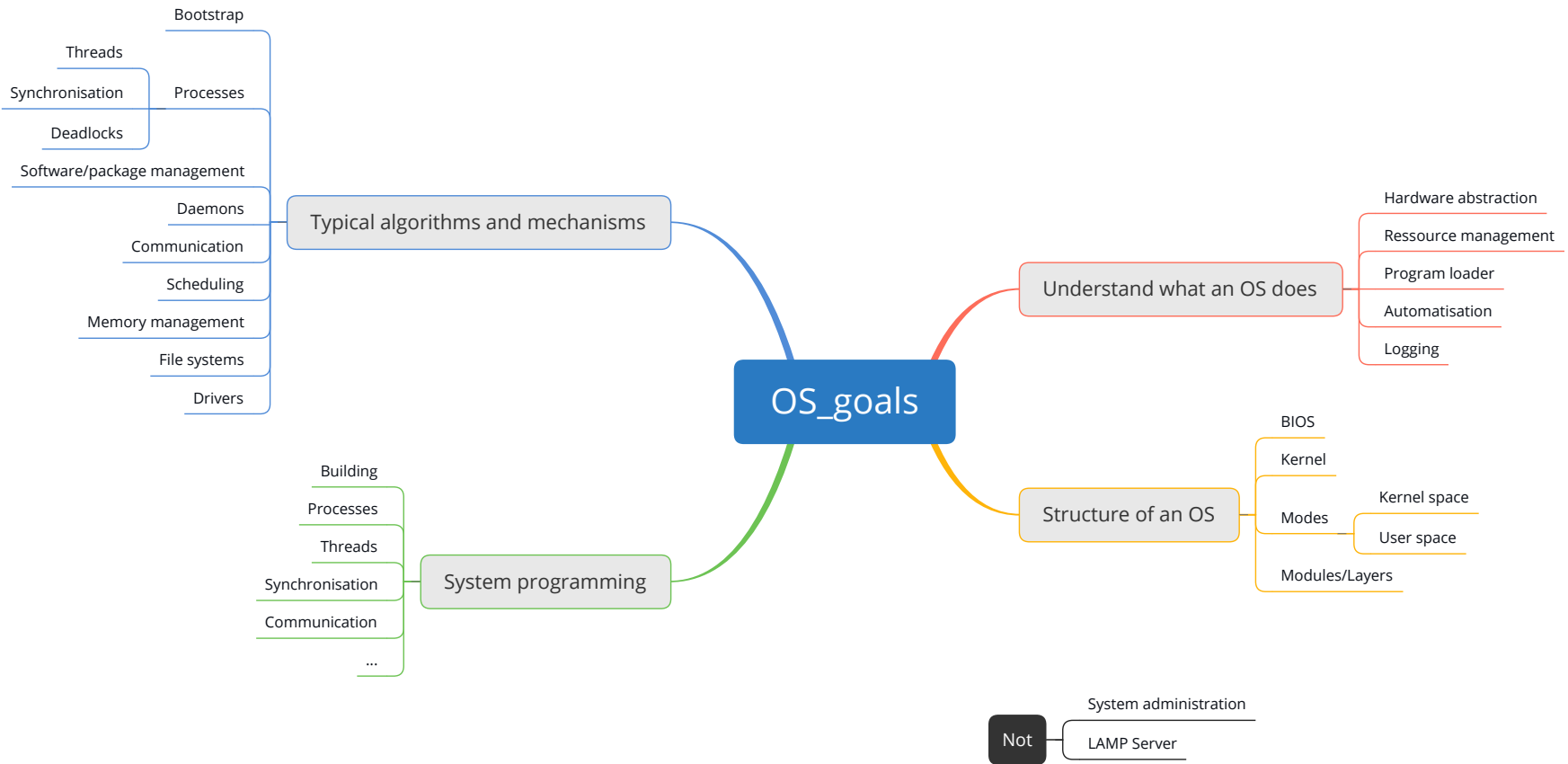
OS 5 – Process/Thread



source: icons.png.com

The lecture is based on the work and the documents of Prof. Dr. Ludwig Frank

Goal



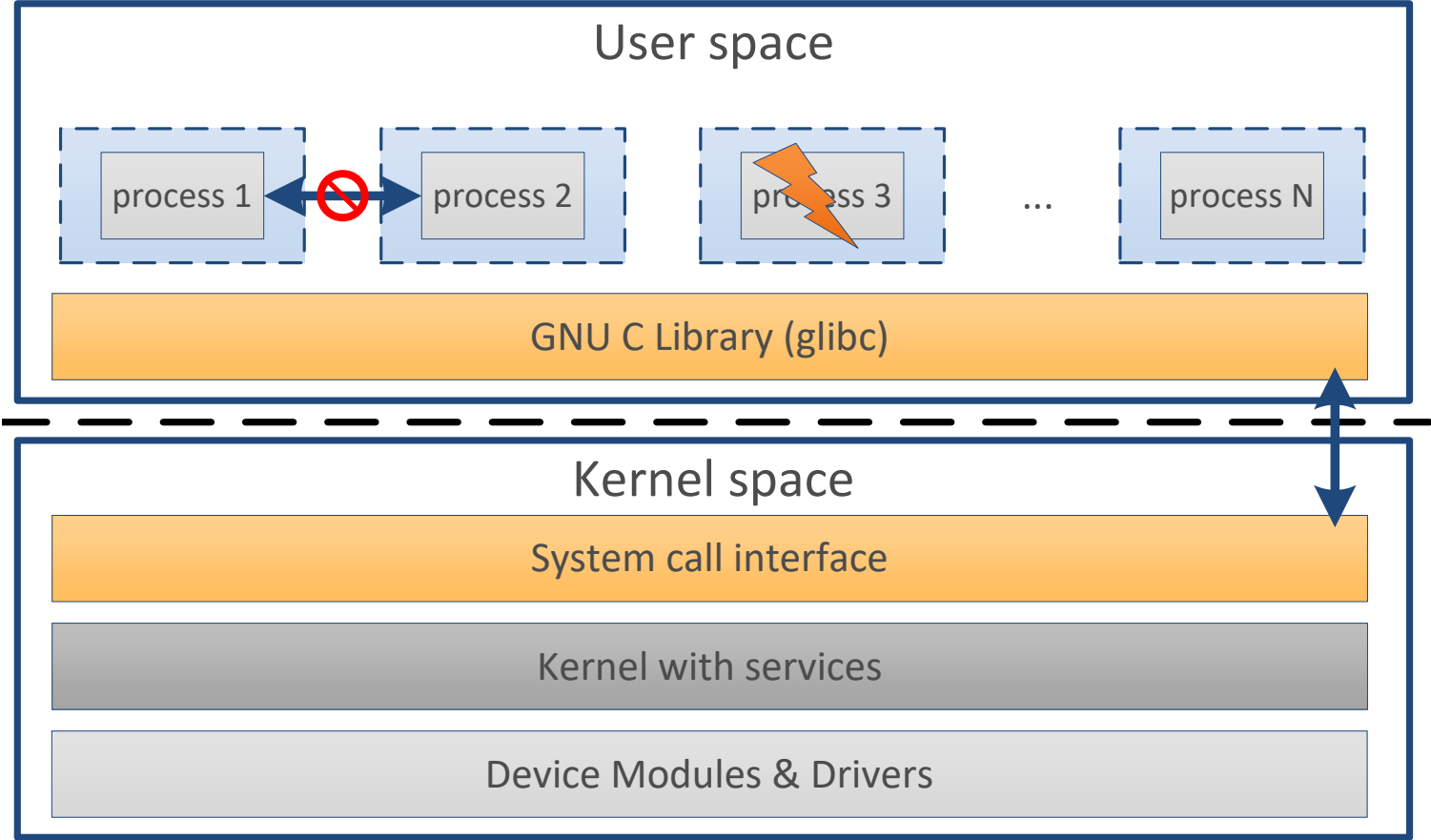
Intro

What is a process?

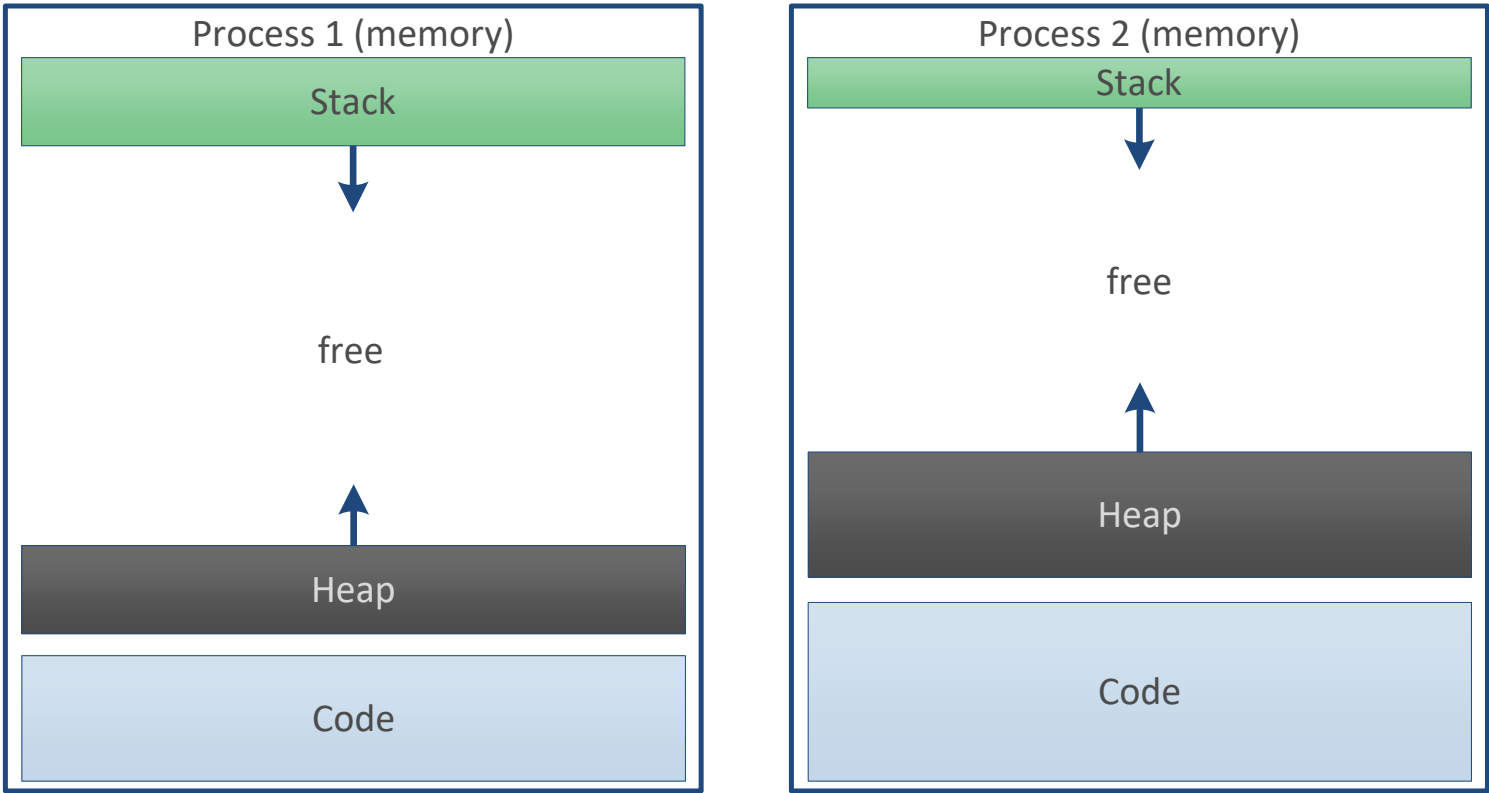
Process definition

A process is an **instance of a computer program** that is being executed.

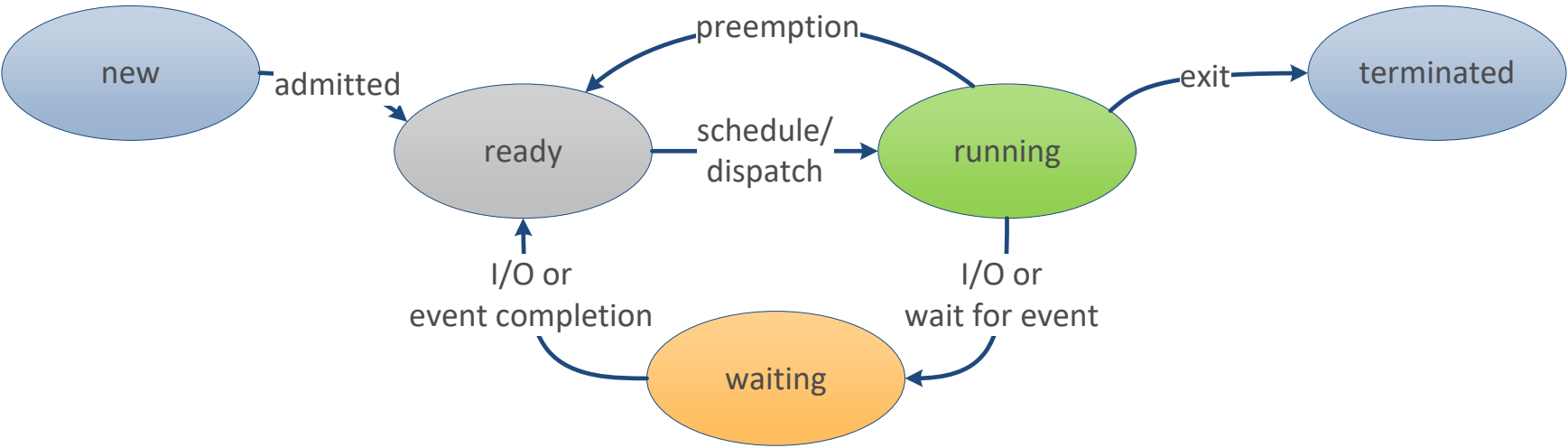
Process isolation



Process memory view

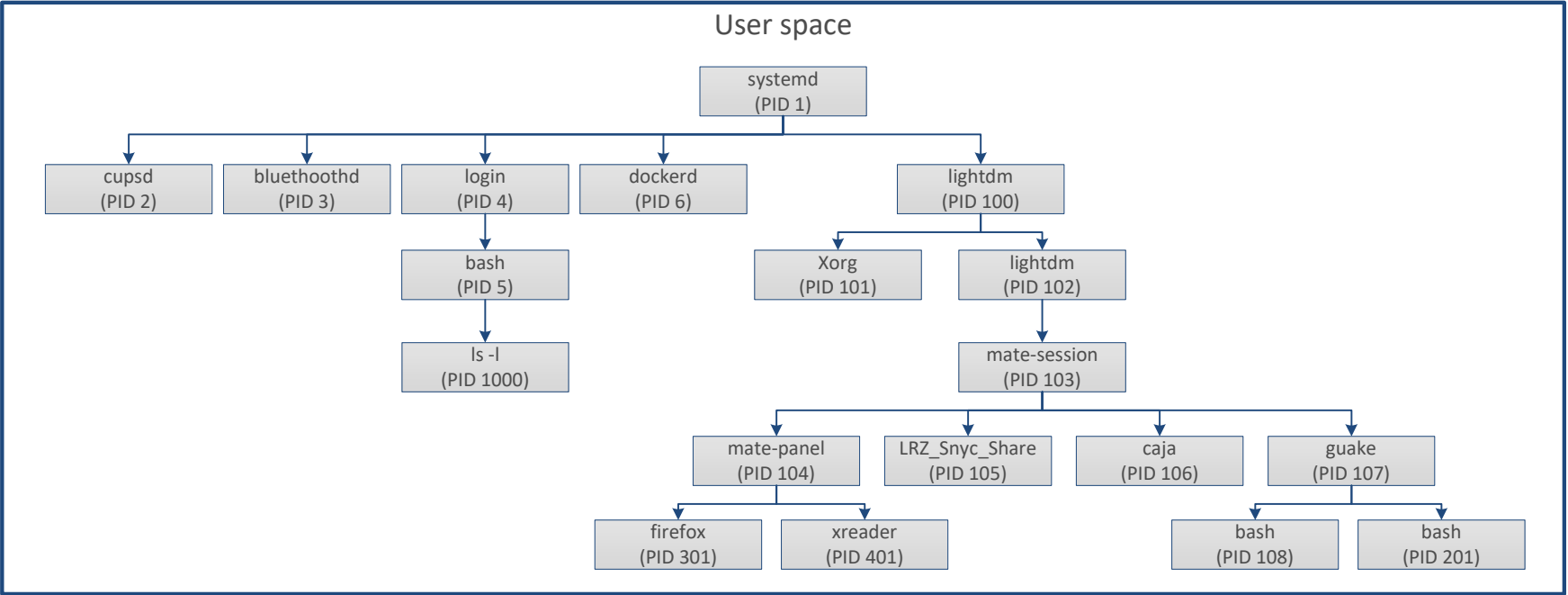


Process states



It is still a simplified view!

Process hierarchy



Daemon: a special kind of a process

Daemon - disk and execution monitor (bacronym)

- A daemon is a process
- Operates in the background
- A daemons parent PID is 1 -> `systemd`
- Usually started from `systemd` as part of the boot procedure
- A daemon has no direct interaction (shell, keyboard, mouse) with the user
- Communication with a daemon: network, signals, pipes, shared memory, ...
- Working directory is /
- Usually uses a logfile to log events and errors

Process specific properties

Process management

- Process id (PID)
- Parent process id (PPID)
- State
- Register entries
- Priority
- Signals
- Start time
- Environment variables

Process memory

- Code (pointer)
- Stack (pointer)
- Heap (pointer)

File management

- Root directory (/)
- Working directory
- File descriptors
- User id
- Group id

OS process table

Process control block (PCB)

- The **kernel manages** the different **processes**
- Each process has its own **process control block (PCB)**
 - Contains all **process specific properties**
 - The process table contains all PCBs
- In Linux: **struct** `task_struct` {...}
- <https://github.com/torvalds/linux/blob/master/include/linux/sched.h>



Advantages of multiple processes

- **Independent start** of different processes
- Can be **executed** in **parallel**
- If a process **crashes** the others can **continue** their work
- **No overwriting** of the memory
- Security: **No read** of another process memory possible
- **Independent development!**
- **Independent dependencies!**
- **Each user** can have its **own processes**.

Programmatic: execute a command

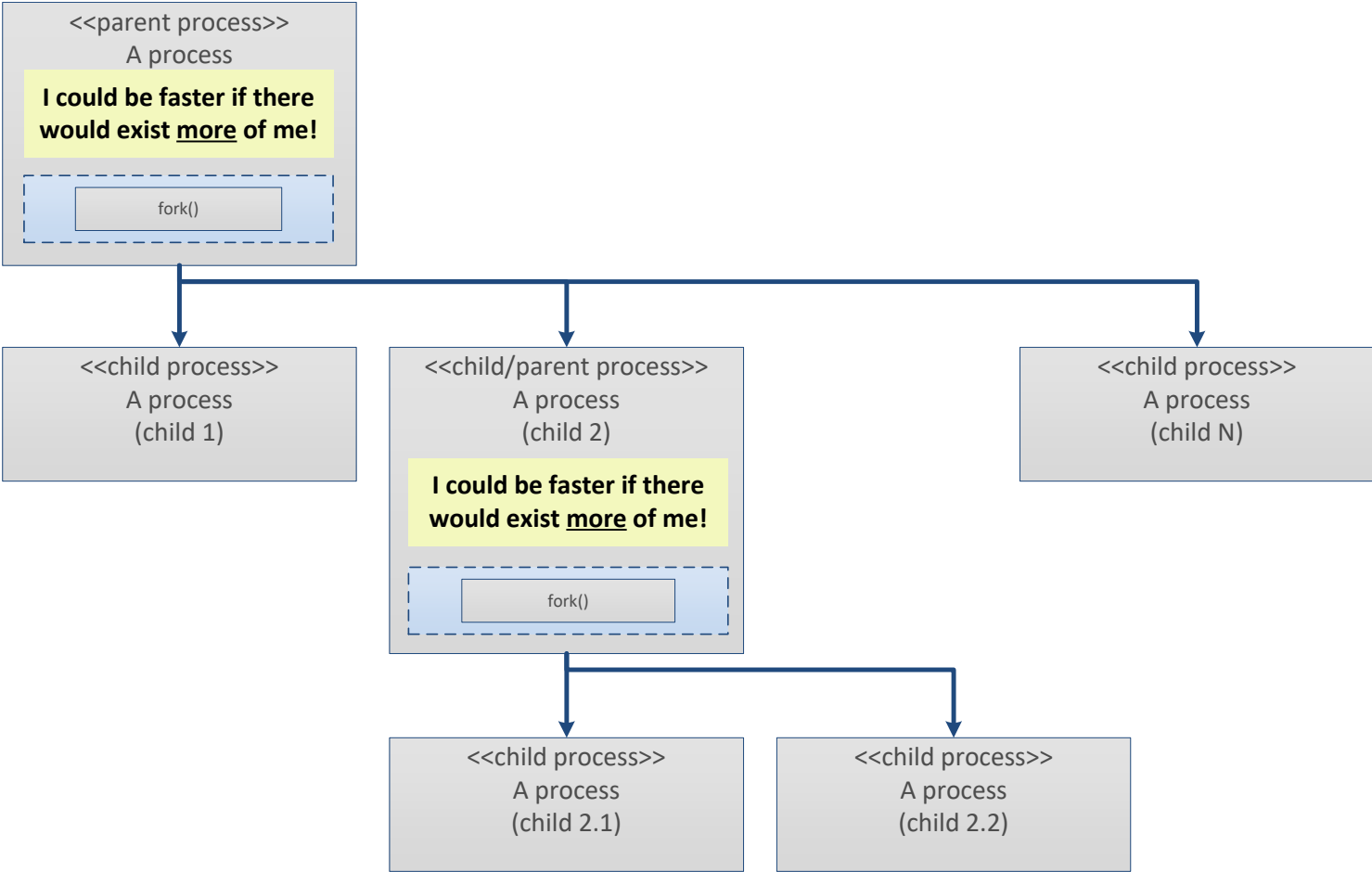
```

1  #include <stdio.h>      //printf
2  #include <stdlib.h>     //EXIT_SUCCESS, system
3
4  int main(int argc, char** argv)
5  {
6      //executes a command specified in command by calling /bin/sh -c command
7      const char* const command = "ls -l /";
8      int exit_status = system(command);
9
10     if(exit_status == -1) {
11         printf("%s can't be started.\n", command);
12     } else {
13         printf("%s exited with status: %d.\n", command, exit_status);
14     }
15
16     return EXIT_SUCCESS;
17 }

```

■ system (man): <http://man7.org/linux/man-pages/man3/system.3.html>

Programmatic: fork idea



Programmatic: fork example

```

1  #include <stdio.h>      //printf
2  #include <stdlib.h>     //EXIT_SUCCESS, EXIT_FAILURE, system
3  #include <unistd.h>     //fork
4  #include <sys/wait.h>   //waitpid
5
6  int main(int argc, char** argv)
7  {
8      pid_t pid = fork();
9
10     switch(pid){
11         case -1: //error
12             printf("Error: fork failed.\n");
13             exit(EXIT_FAILURE);
14             break;
15         case 0: //child
16             printf("Hi, I'm the fork with the PID %d!\n", getpid());
17             break;
18         default: //parent
19             printf("Parent waits until child process with PID %d ends.\n", pid);
20             waitpid(pid, NULL, 0);
21             printf("Child process with PID %d exited.\n", pid);
22             break;
23     }
24
25     return EXIT_SUCCESS;
26 }

```

■ fork (man): <http://man7.org/linux/man-pages/man2/fork.2.html>

Processes management on shell

cmd	Description
<code>./command</code>	Start a process.
<code>kill</code>	Stop (exit) a process.
<code>wait</code>	Wait until a child process has stopped.
<code>ps aux</code>	Show information about started processes.
<code>top</code>	Show live information about processes.
<code>pstree</code>	Show the process hierarchy .
<code>renice</code>	Change the priority of a process.

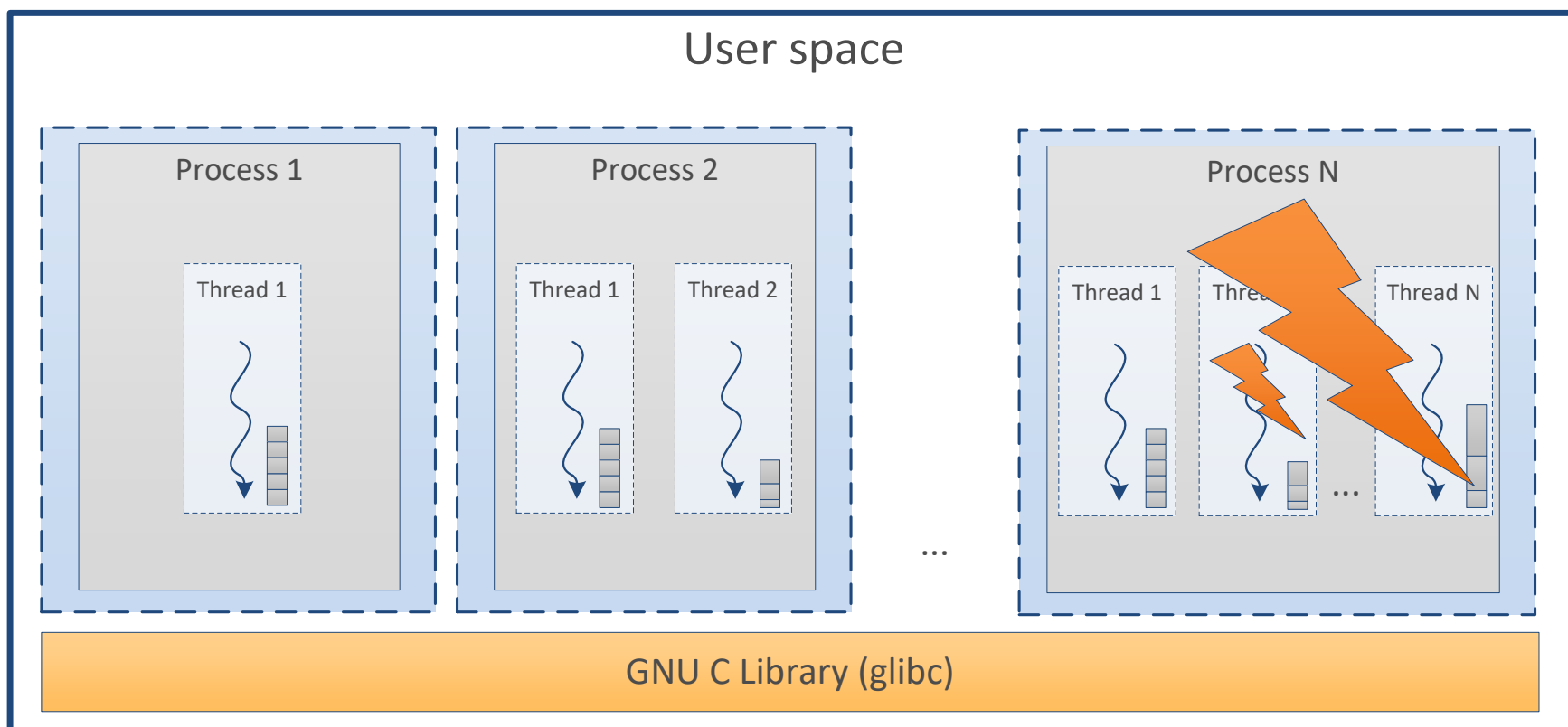
Thread definition

A thread is an **entity within a process** that can be **scheduled independently** of other threads for execution.

In Linux: A thread is a **lightweight process!**



Thread illustration



Thread properties

Properties for each thread

- Thread id (TID)
- State
- Register entries
- Stack (pointer)

Shared process properties

- Address space (code/heap)
- Global variables
- Opened files
- Child processes
- Signals
- Working directory
- Environment variables

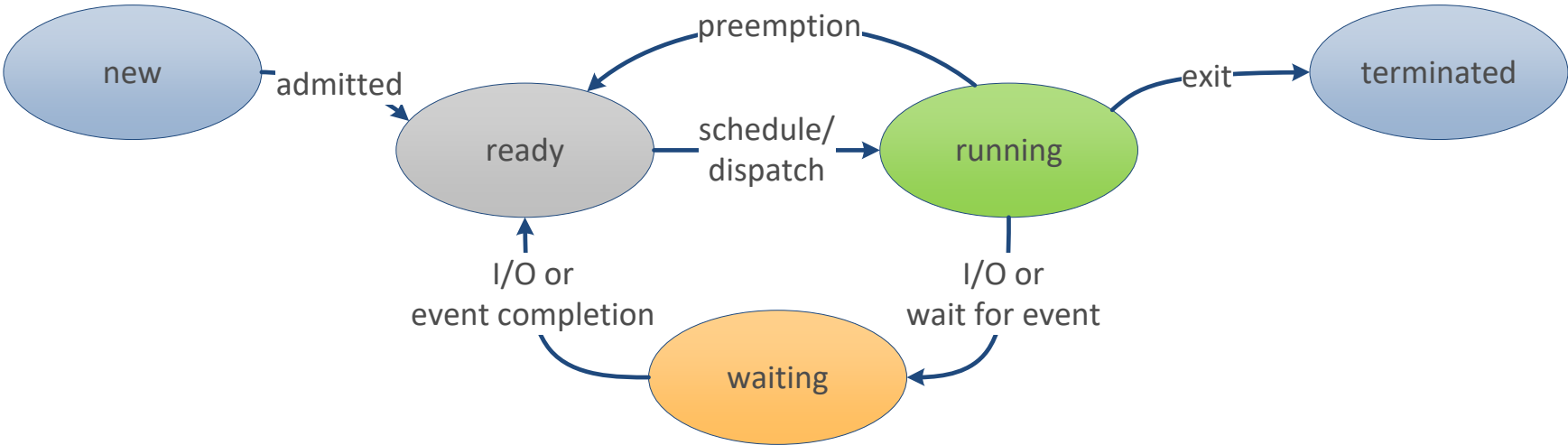
OS thread/process table

Thread control block (TCB)

- The **kernel manages** the different **threads**
(similar to a process)
- Each thread has its own **thread control block (TCB)**
 - Contains all **thread specific properties**
 - Each process knows the thread TCBs in its PCB
- In Linux:
 - **struct** `task_struct` {...}
 - It is **used** for **processes and threads**
 - This is the reason for the term: **“lightweight process”**
 - <https://github.com/torvalds/linux/blob/master/include/linux/sched.h>

Thread states

Similar to the process states.



It is still a simplified view!

Programmatic: pthread example

```

1 #include <stdio.h>      //printf
2 #include <stdlib.h>     //EXIT_SUCCESS, EXIT_FAILURE
3 #include <pthread.h>    //pthread_create, pthread_join
4
5 void* thread_work() {
6     printf("Hi, I'm the thread with the TID %lu!\n", pthread_self());
7     return NULL; //the thread ends here
8 }
9
10 int main(int argc, char** argv) {
11     //create thread
12     pthread_t thread_id;
13     int thread_create_state = pthread_create(&thread_id, NULL, &thread_work, NULL);
14     if( thread_create_state != 0) {
15         printf("Failed creating thread\n");
16         exit(EXIT_FAILURE);
17     }
18
19     //join thread (wait until the thread has finished)
20     printf("Main thread waits until child thread with TID %lu ended.\n", thread_id);
21     int thread_exit_state = pthread_join(thread_id, NULL);
22     if(thread_exit_state != 0){
23         printf("Thread exited with failure %d.\n", thread_exit_state);
24     }
25
26     printf("Child thread with TID %lu exited.\n", thread_id);
27     return EXIT_SUCCESS;
28 }

```

- pthreads (man): <http://man7.org/linux/man-pages/man7/pthreads.7.html>
- pthread_create(): https://man7.org/linux/man-pages/man3/pthread_create.3.html
- pthread_join(): https://man7.org/linux/man-pages/man3/pthread_join.3.html

Process vs thread

Quiz: When to use a process or a thread?



Summary and outlook

Summary

- What is a process/thread
- Process hierarchy
- Processes management
- Thread management
- Parallelisation

Outlook

- Process synchronisation
- Semaphore
- Mutual exclusion