NTNU | Norwegian University of Science and Technology

Operating Systems

Lecture 3: Challenges and tasks of operating systems

Michael Engel

Different views of an operating systems

- Abstractions
  - Processes, virtual memory, file systems
- Tasks
  - CPU scheduling, synchronization
  - Inter-process communication
  - Memory management
- Problems
  - Deadlocks
  - System security
- Challenges
  - Multiprocessor systems
  - Cloud computing and virtualization

A process…

Processes

- Horning/Randell, Process Structuring:

> A *process* is a triple (S, f, s), where S is a state space, f is an action function in that space, and s is the subset of S which defines the initial states of the process. A process generates all the computations generated by its action function from its initial states.

- Dennis/van Horn, Programming Semantics for Multiprogrammed Computations

> A *process* is a locus of control within an instruction sequence. That is, a process is that abstract entity which moves through the instructions of a procedure as the procedure is executed by a processor.

- Habermann, Introduction to Operating System Design

> A *process* is controlled by a program and requires a processor to execute that program.

A process…

Processes

- „***…is a program in execution***“ [unknown source]

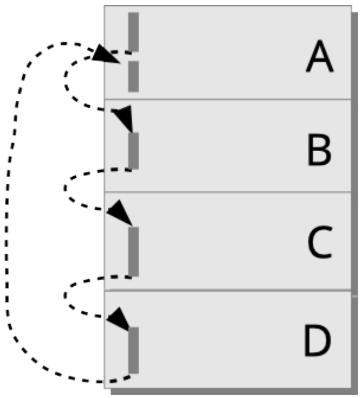
- This requires a **process context**, which consists of...
- Memory: **code**, **data** and **stack segment** (text, data, bss, stack, heap)
- Contents of processor registers
  - Instruction pointer
  - Stack pointer
  - General purpose registers
  - …
- Process state
- User ID
- Access permissions
- Currently used resources
  - Files, I/O devices, etc.
- ...

represented in the **process control block** (PCB)

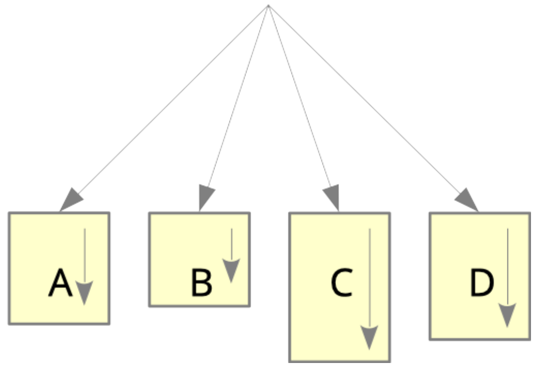
The process model

Processes

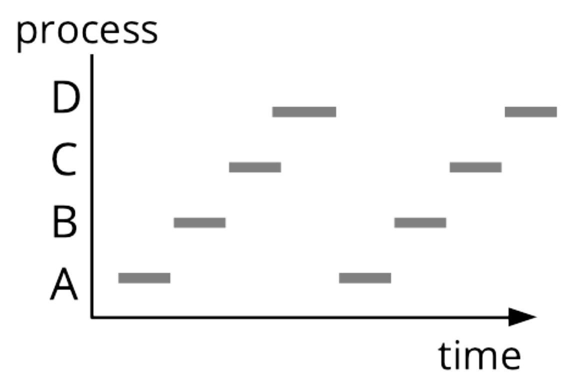
multiprogramming



concurrent processes



CPU multiplexing



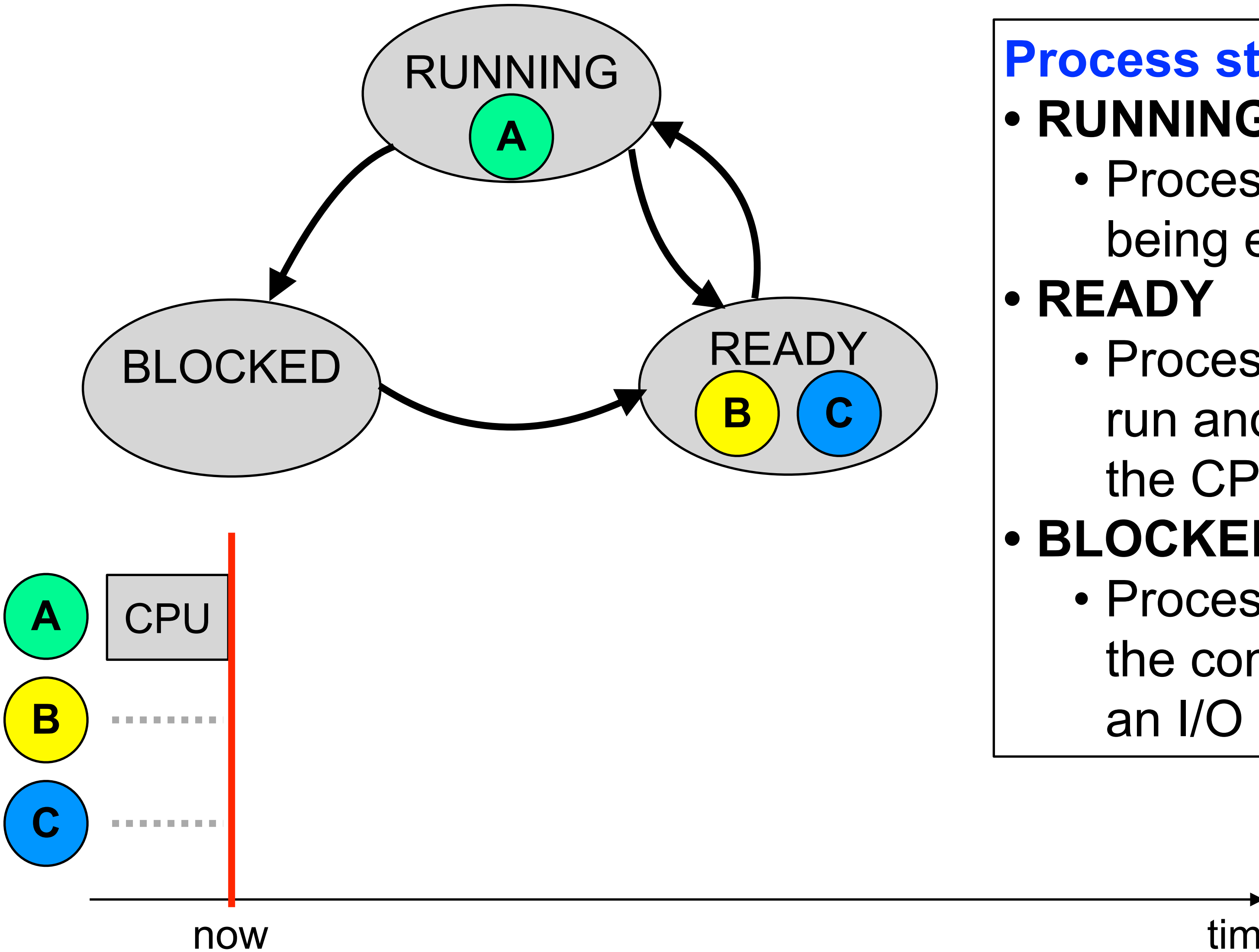
Technical view:
- 1 instruction pointer
- Context switching

Conceptional view:
- 4 independent sequential control flows

Real time view:
(Gantt diagramm)
- Only one process is active at any given point in time (on a single processor system)

Process behavior and states (1)

Processes

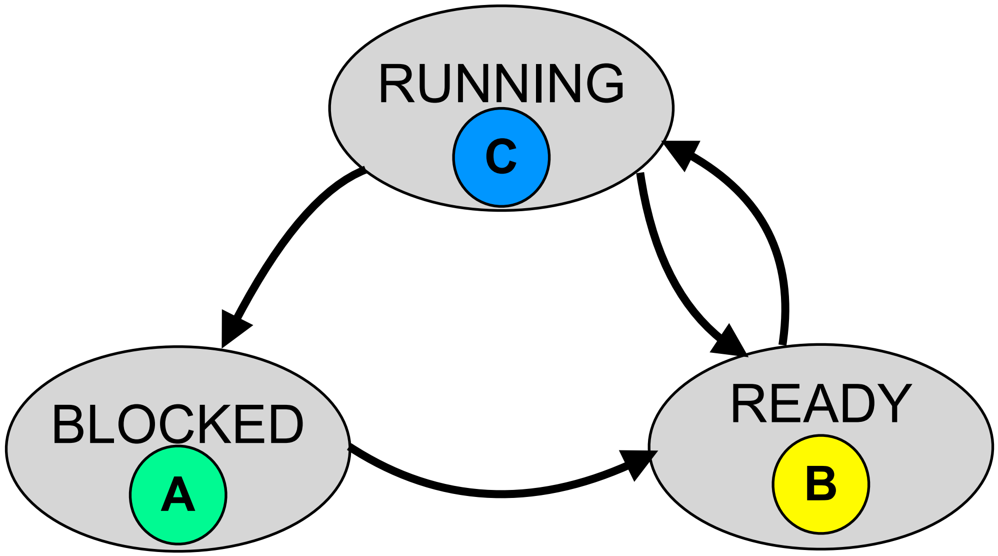


Process states:

- **RUNNING**
  - Process is currently being executed
- **READY**
  - Process is ready to run and waits for the CPU
- **BLOCKED**
  - Process waits for the completion of an I/O operation

Process behavior and states (2)

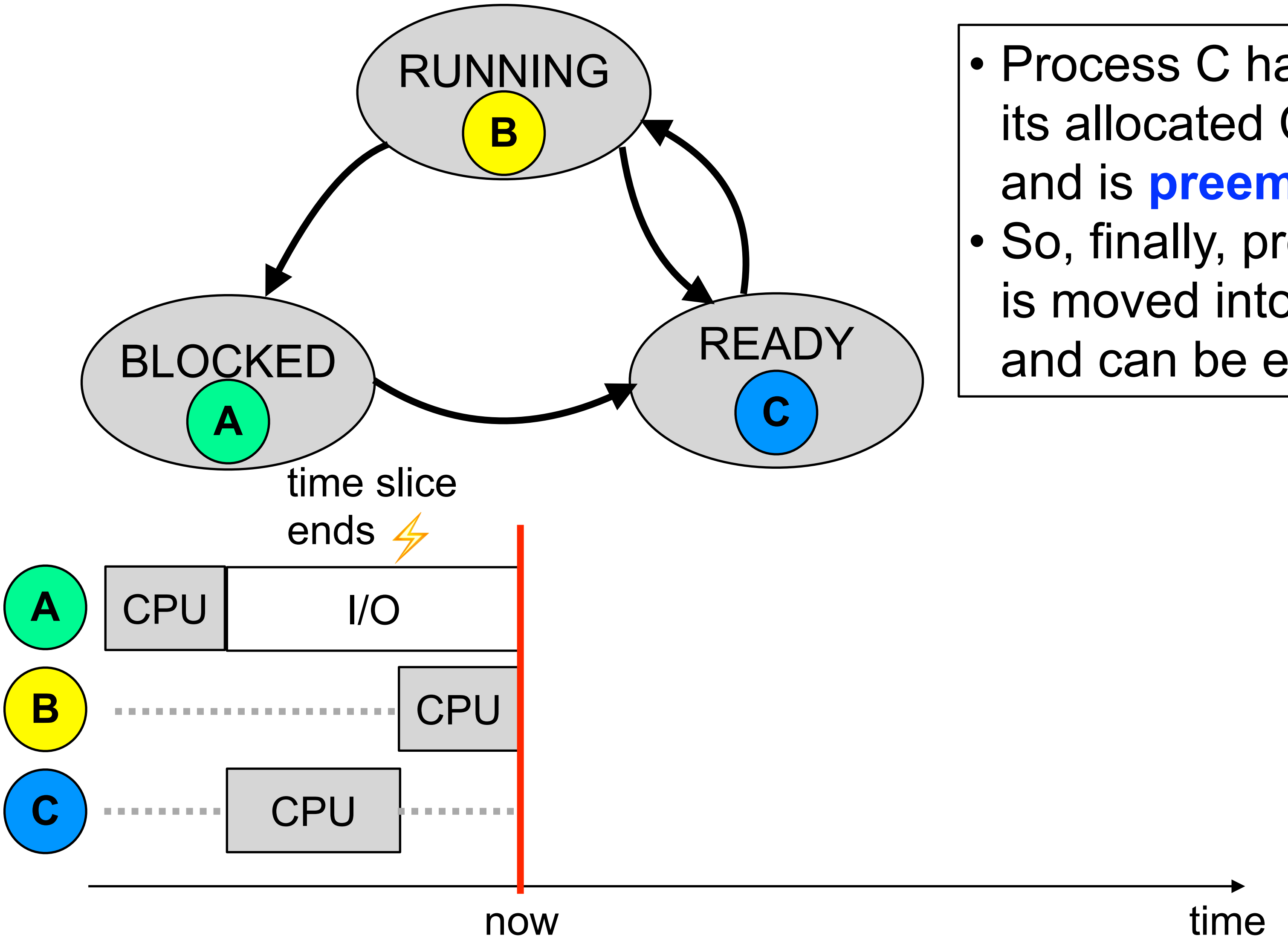
Processes



- Process A has started an I/O operation and was moved to the BLOCKED state.
- Since A does not make use of the CPU now, the OS chose process C and moved it from READY to RUNNING.
- This is a **context switch** from A to C

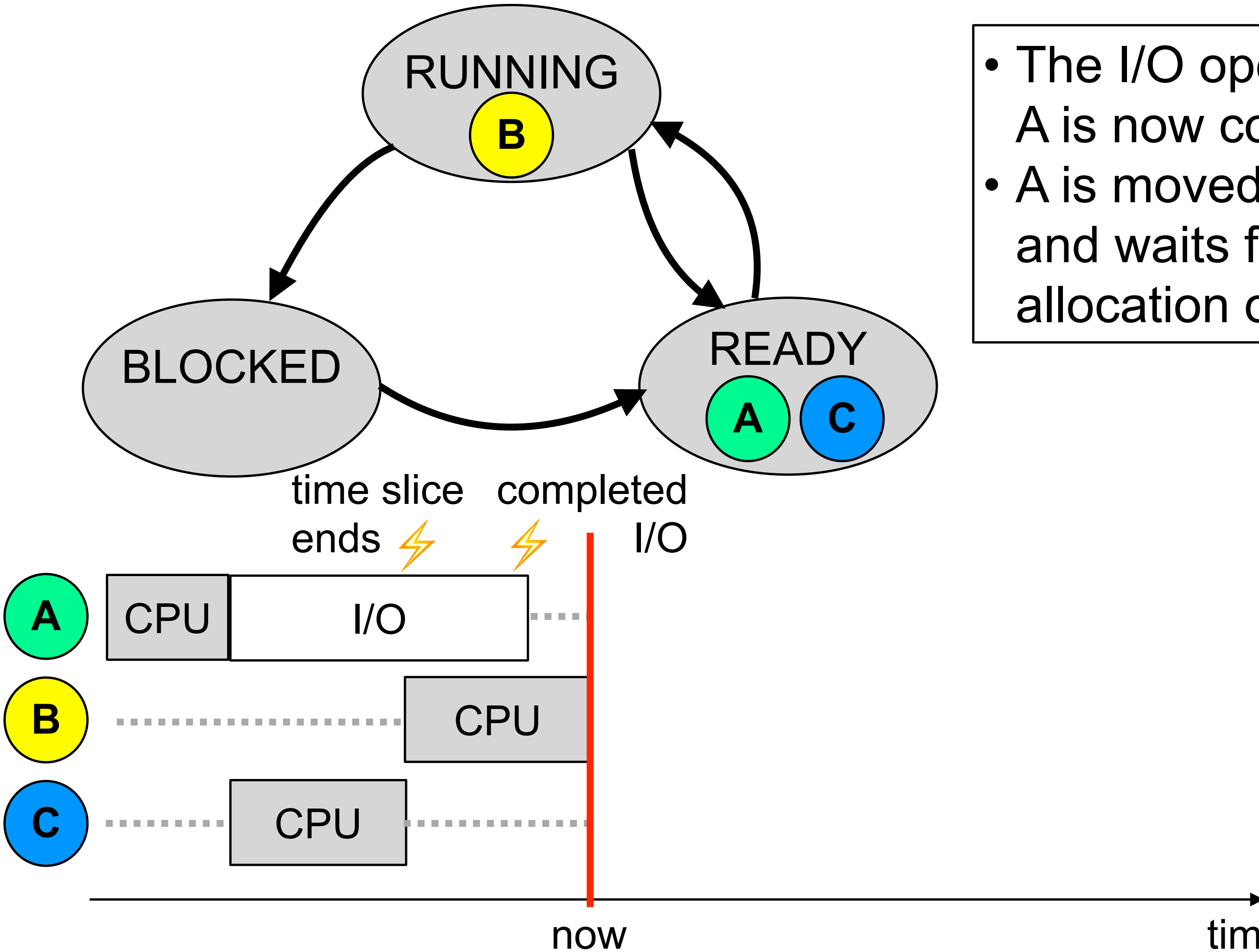
Process behavior and states (3)

Processes



- Process C has used up its allocated CPU time and is **preempted**
- So, finally, process B is moved into RUNNING and can be executed

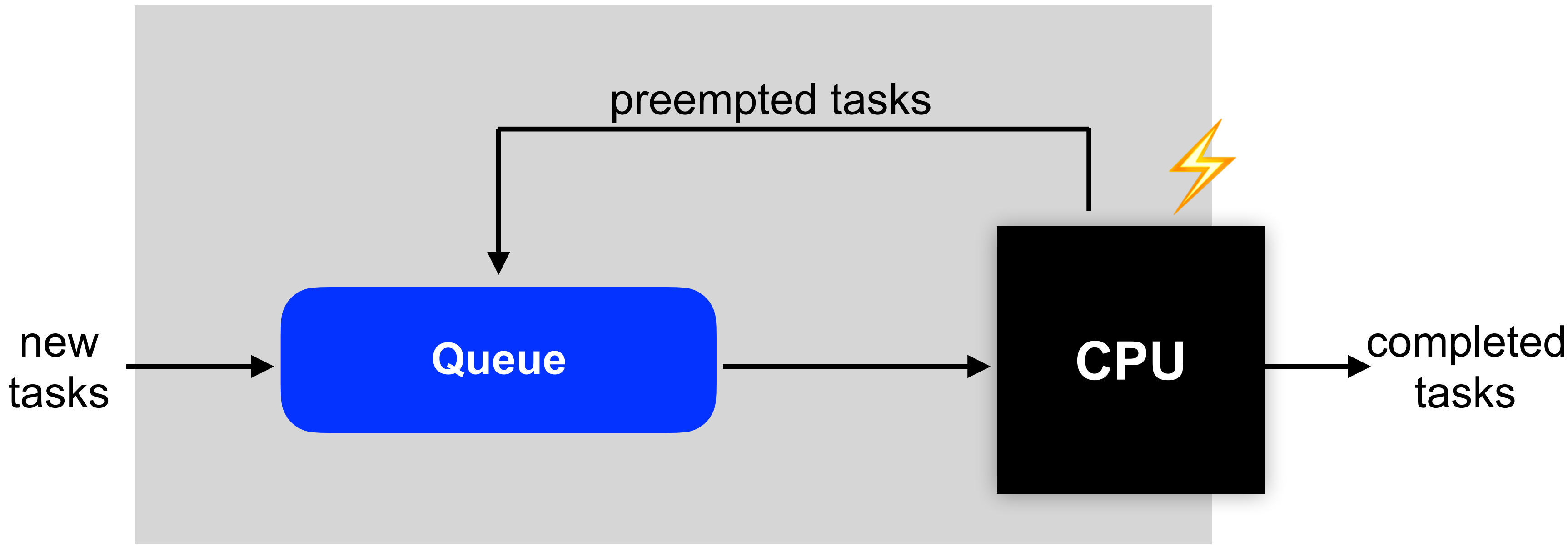
Process behavior and states (4)



- The I/O operation of A is now completed
- A is moved to READY and waits for the allocation of the CPU

CPU scheduling

Scheduling



A single **scheduling algorithm** is characterized by the order of processes in the queue and the conditions under which the processes are added to the queue.

CPU scheduling

Scheduling

- Scheduling enables the coordination of concurrent processes
- Basic questions:
  - Which sorts of events can cause preemption?
  - In which order should processes be executed?
- Objectives of a scheduling algorithm
  - user oriented ➛ short reaction times
  - system oriented ➛ optimal CPU utilization
- No single scheduling algorithm can fulfill all requirements!

Process synchronization

Synchronization

- Example: non coordinated access to a printer

Process A

```
...
print("Hei Olav\n");
print("Call me.\n");
print("Phone: 422342\n");
...
```

Process B

```
...
print("Tor \n");
print("I like you!\n");
...
```

Process synchronization

Synchronization

- Reason for the problem: **critical sections**
- Solution approach: **mutual exclusion**
  - Using the **mutex** abstraction

Process A

```
...
lock(&printer_mutex);
print("Hei Olav\n");
print("Call me.\n");
print("Phone: 422342\n");
unlock(&printer_mutex);
...
```

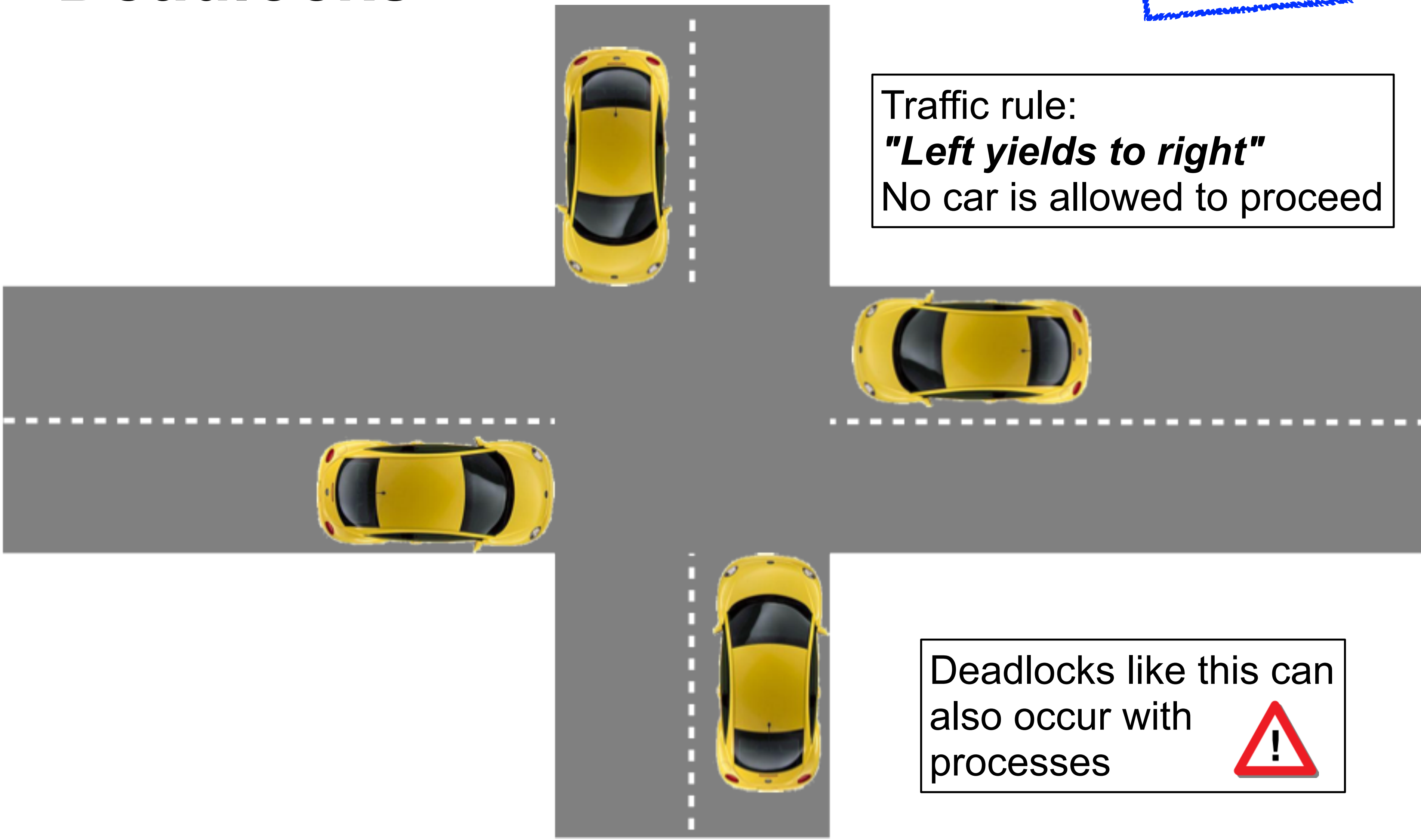
Process B

```
...
lock(&printer_mutex);
print("Tor \n");
print("I like you!\n");
unlock(&printer_mutex);
...
```

If one of the processes A or B is in between the calls to **lock** and **unlock**, the other cannot pass the **lock** and blocks at the lock until the critical section is left by the other process calling **unlock**.

Deadlocks

Deadlocks



Traffic rule:
"Left yields to right"
No car is allowed to proceed

Deadlocks like this can also occur with processes

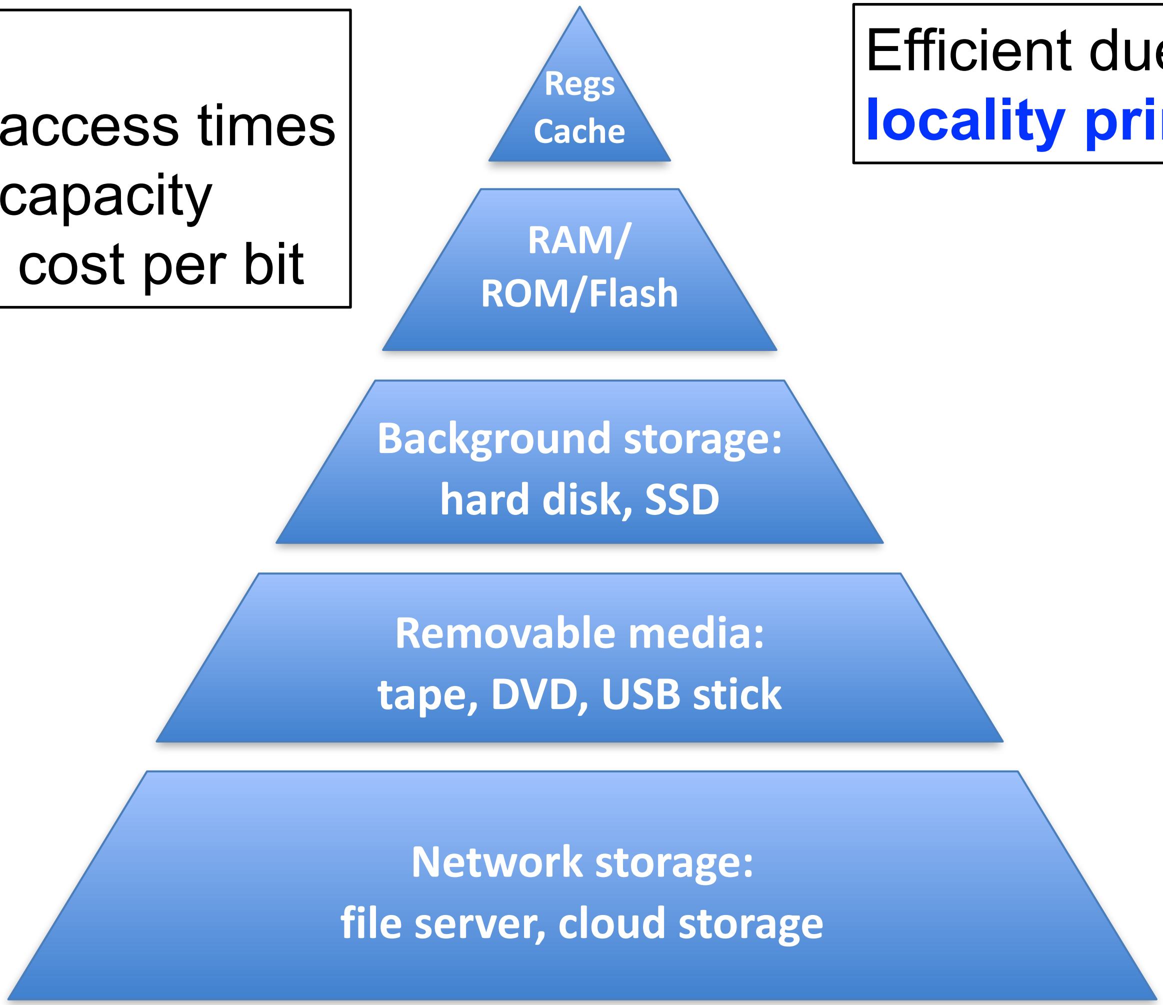
Inter-process communication (IPC) IPC

- ... enables the collaboration of multiple processes
  - **local**, e.g. printing daemon, X window server
  - **remote**, e.g. web server, database server, ftp server
    - *“client/server systems”*
- Abstractions/programming models
  - **Shared memory**
    - multiple processes can use the *same memory area* at the same time
    - additional synchronisation is required
  - **Message passing**
    - *copy semantics:*
      recipient receives a copy of the message
    - can be synchronous or asynchronous

The memory hierarchy

Memory

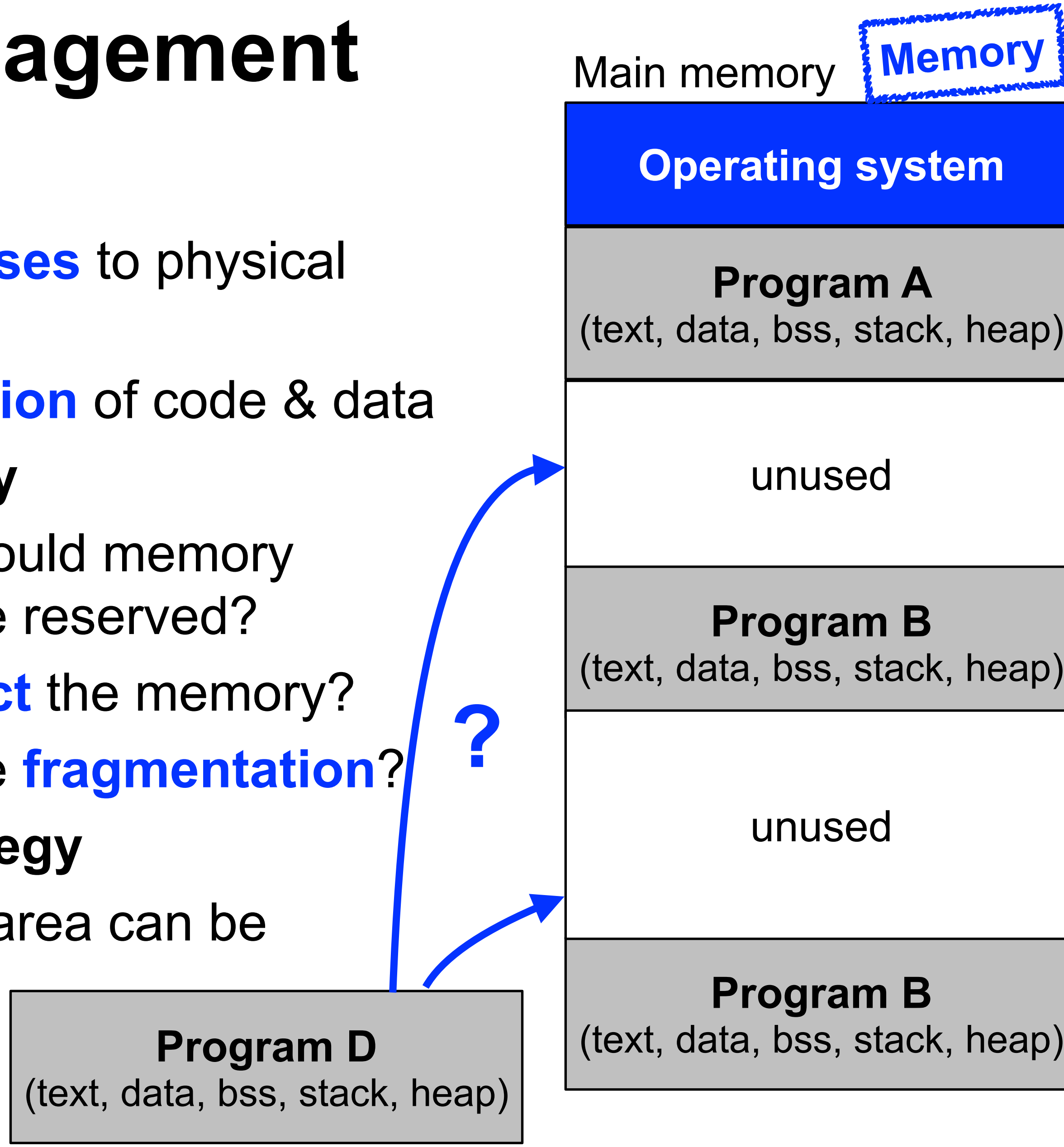
Properties:
- increasing access times
- increasing capacity
- decreasing cost per bit



Efficient due to the **locality principle**!

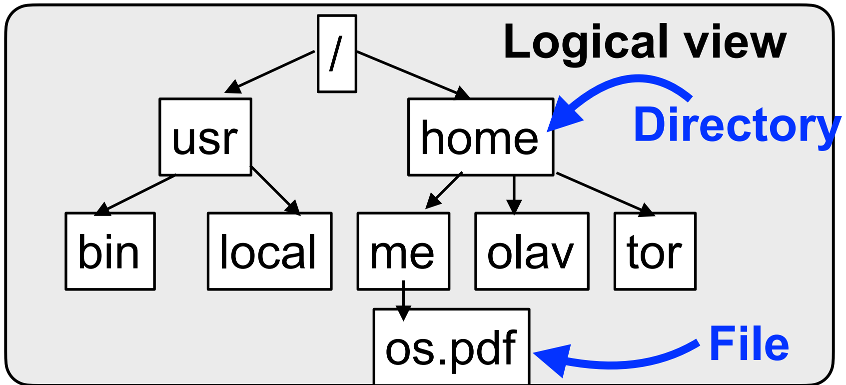
Memory management

- **Address mapping**
  - **Logical addresses** to physical addresses
  - Enables **relocation** of code & data
- **Placement strategy**
  - In which gap should memory for process D be reserved?
  - Can we **compact** the memory?
  - How to minimize **fragmentation**?
- **Replacement strategy**
  - Which memory area can be swapped out?



Background storage

Storage

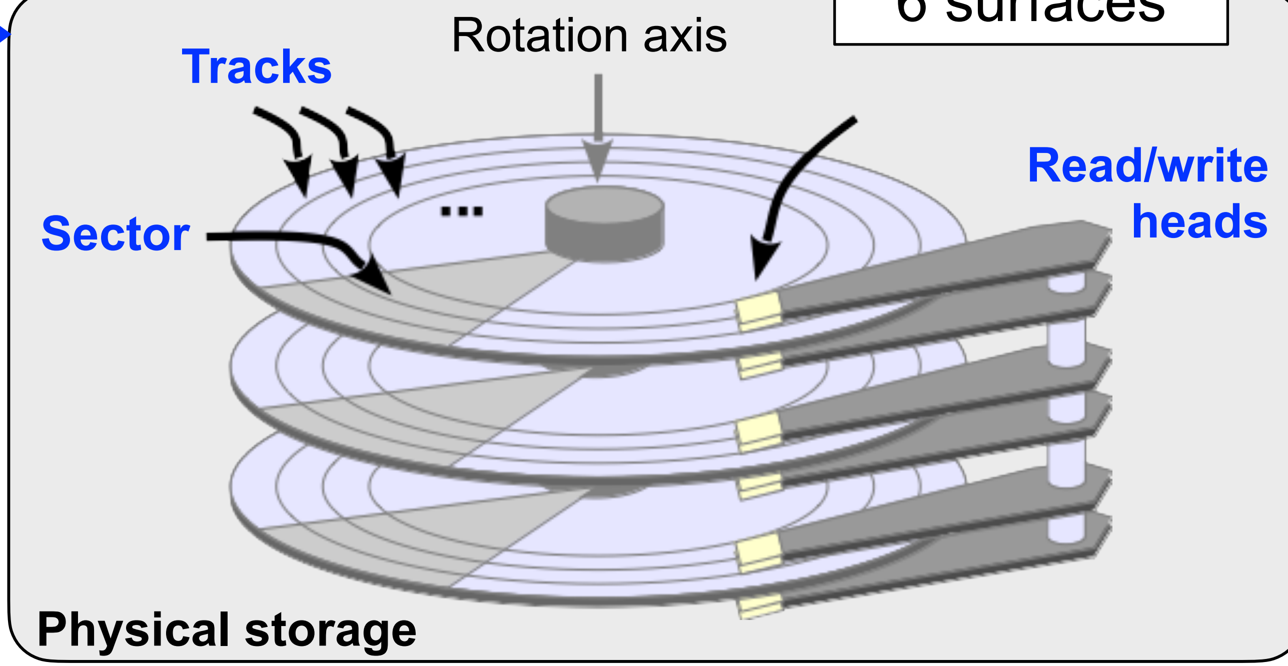


File systems enable permanent storage of large amounts of data

mapping

The operating system provides a logical view to applications and has to implement this efficiently

Hard disk with 6 surfaces



Access matrix

Security

- Elements of the matrix:
  - Subjects (persons/users, processes)
  - Objects (data, devices, processes, memory, …)
  - Operations (read, write, delete, execute, …)
- Question: Is `operation(subject, object)` permitted?

		Objects	
Subjects		**Permissions**	

Basic model: file/process attributes

Security

- Properties related to a user:
  - for which user is the process being executed?
  - which user is the **owner** of a file?
  - which permissions does the owner of a file give to him/ herself and which permissions to other users?
- Permissions of a process when accessing a file
  - Attributes of processes: **user ID**
  - Attributes of files: **owner ID**

	file 1	file 2	file 3
user 1			
user 2		read	
user 3			
user 4			

Unix access permissions

Security

- Unix: simple access control lists
- Processes have a **user ID** and a **group ID**
- Files have an **owner** and a **group**
- Permissions are related to the
user (owner), **g**roup, and all **o**thers

file.tex		
`rw-`	`r--`	`---`
		others
	group: staff	
user: michael		

File attributes:

`rwx`

- **x** → execute: yes/no
- **w** → write: yes/no
- **r** → read: yes/no

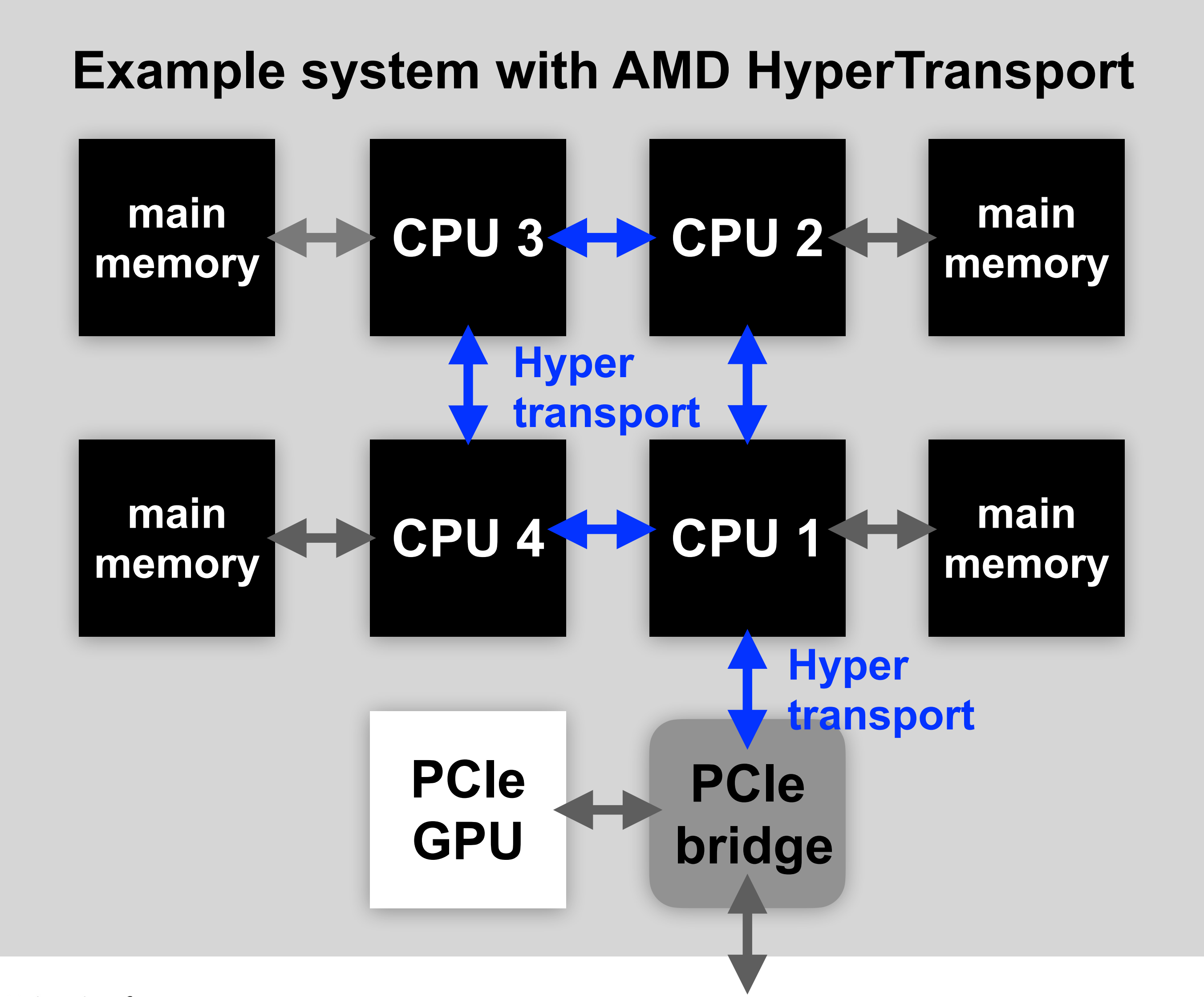
Multiprocessors

NUMA architectures (non uniform memory architecture)

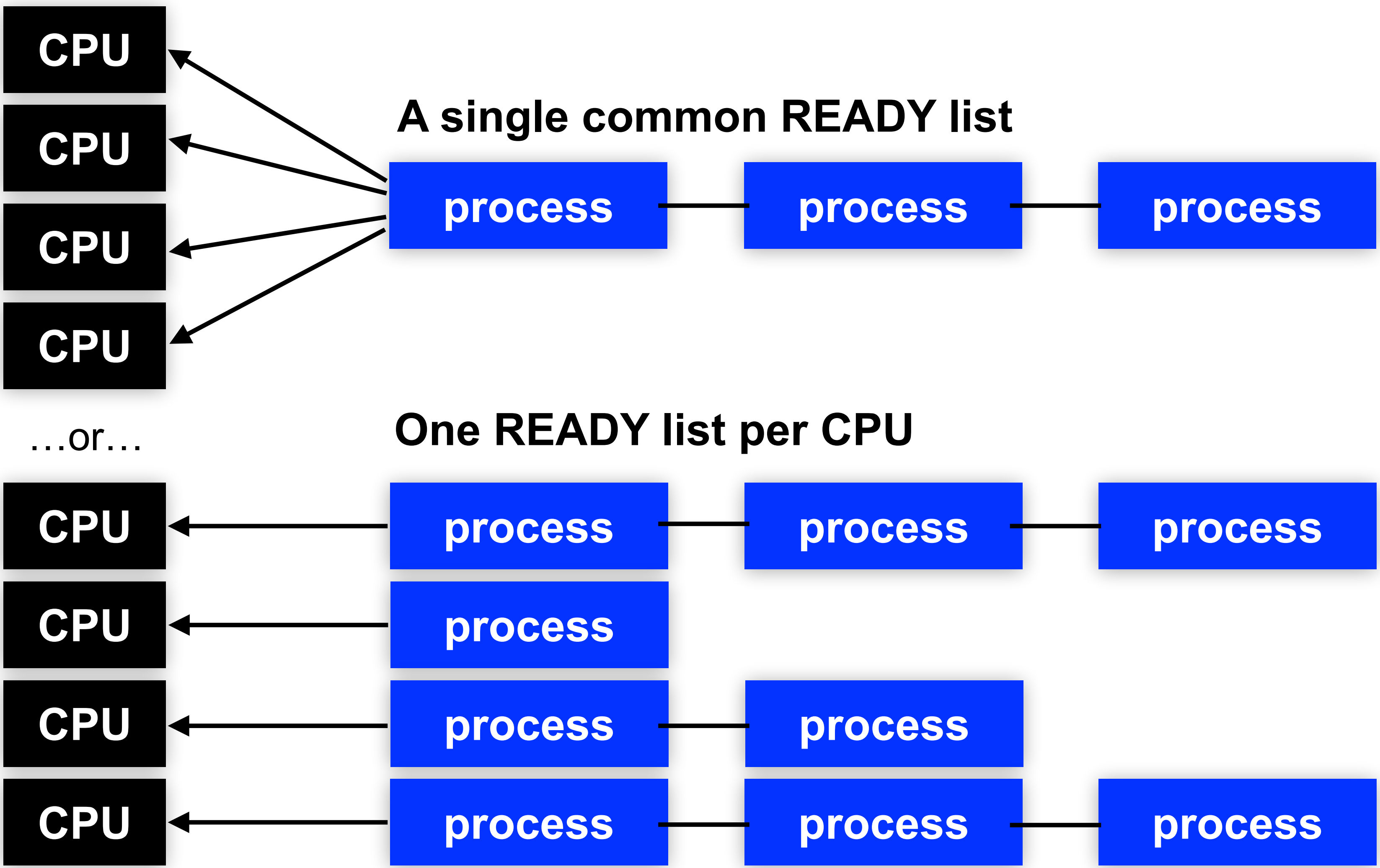
The CPUs (which can have multiple cores each) communicate via HyperTransport.

Global address space:
Memory connected to a different CPU can be accessed, but the **latency is higher**.

Approach enables better **scalability**, since parallel memory accesses are possible .



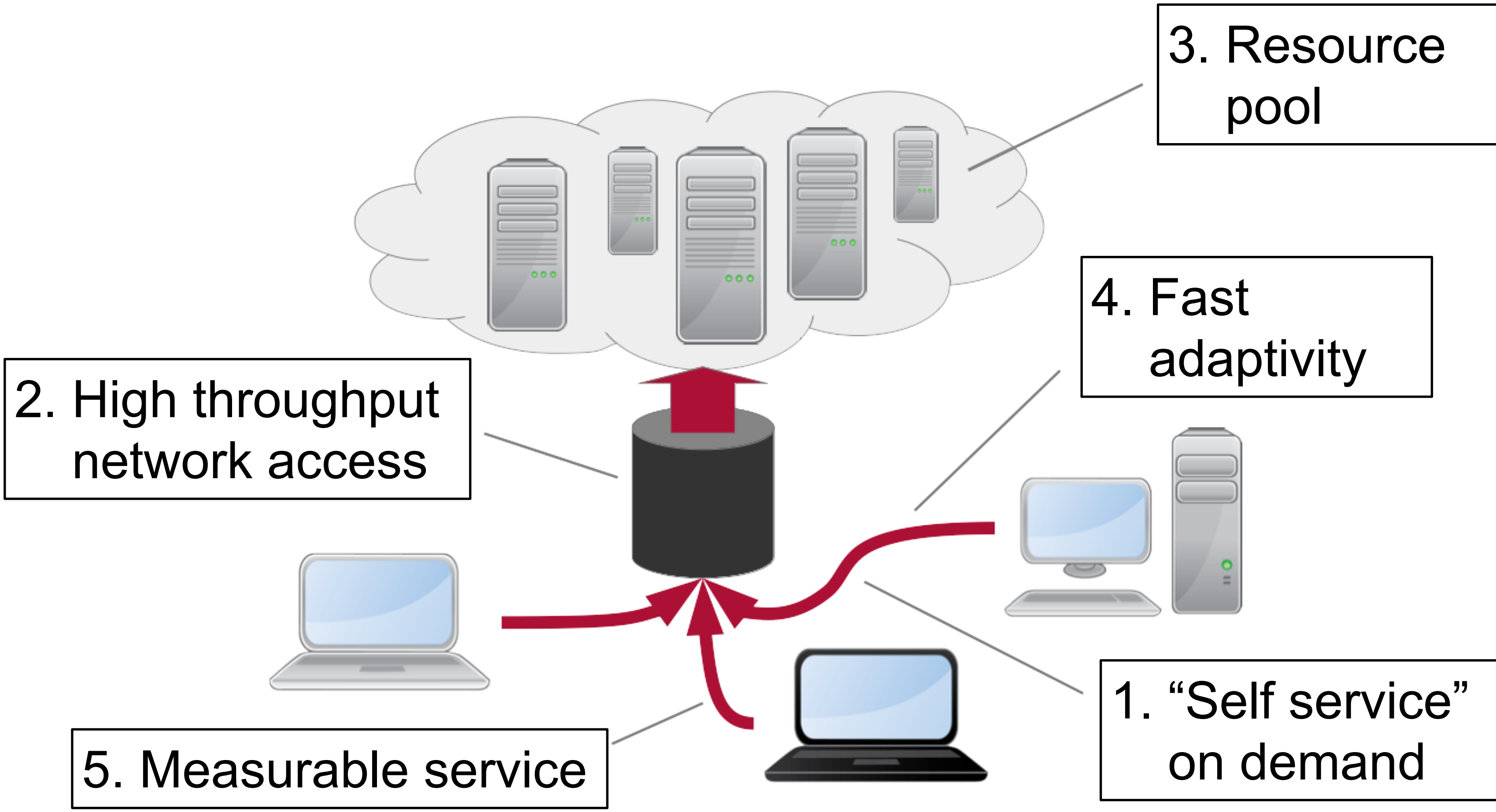
CPU allocation for multiprocessors



Cloud computing

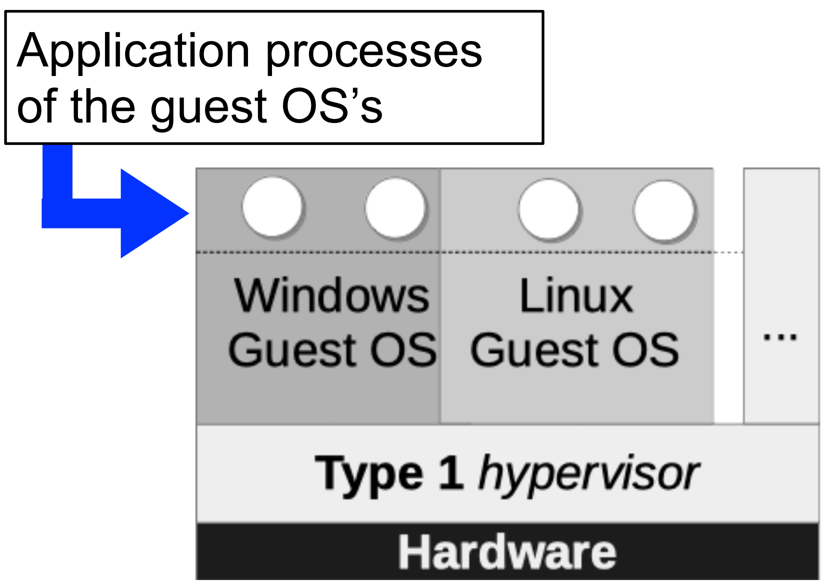
Cloud & virtualization

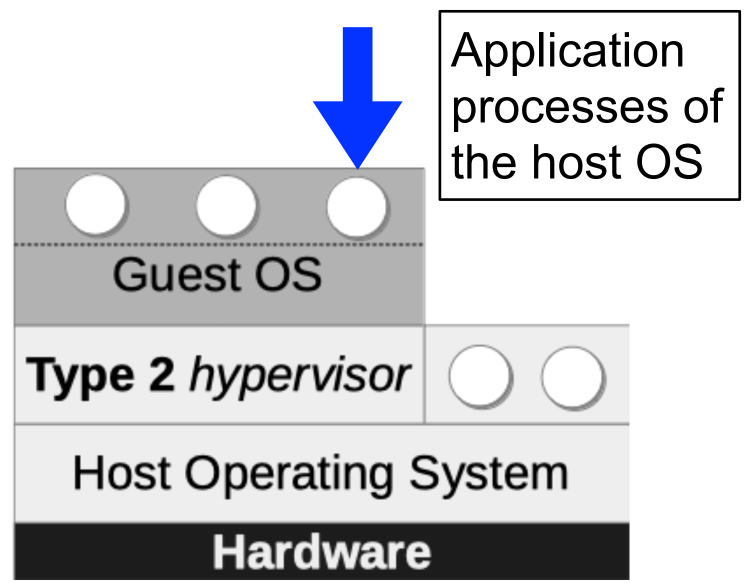
- According to the US *National Institute of Standards and Technology*, a **Cloud** has five properties:



Hardware virtualization

- …enables the creation of multiple **virtual machines** on one physical computer. Each virtual machine can have its own OSn.
- Important foundation technology for Cloud computing and server consolidation
- Technical basis: **hypervisor** / **virtual machine monitor**





Conclusion: the OS…

- administers resources, especially the CPU(s) and memory
- provides abstractions, e.g. …
  - the process concept
  - files and directories
  - permission conecpt
- is optimized for the specific application profile
- It is impossible to satisfy the requirements of all applications to 100%. We can approach this goal using virtualization.

Operating systems, typical applications and the hardware have evolved together during the last few decades.
The system abstractions available today are the result of an evolution which is still ongoing.