Rosenheim
Technical University
of Applied Sciences

Prof. Dr. Florian Künzner

Technical University of Applied Sciences Rosenheim, Computer Science

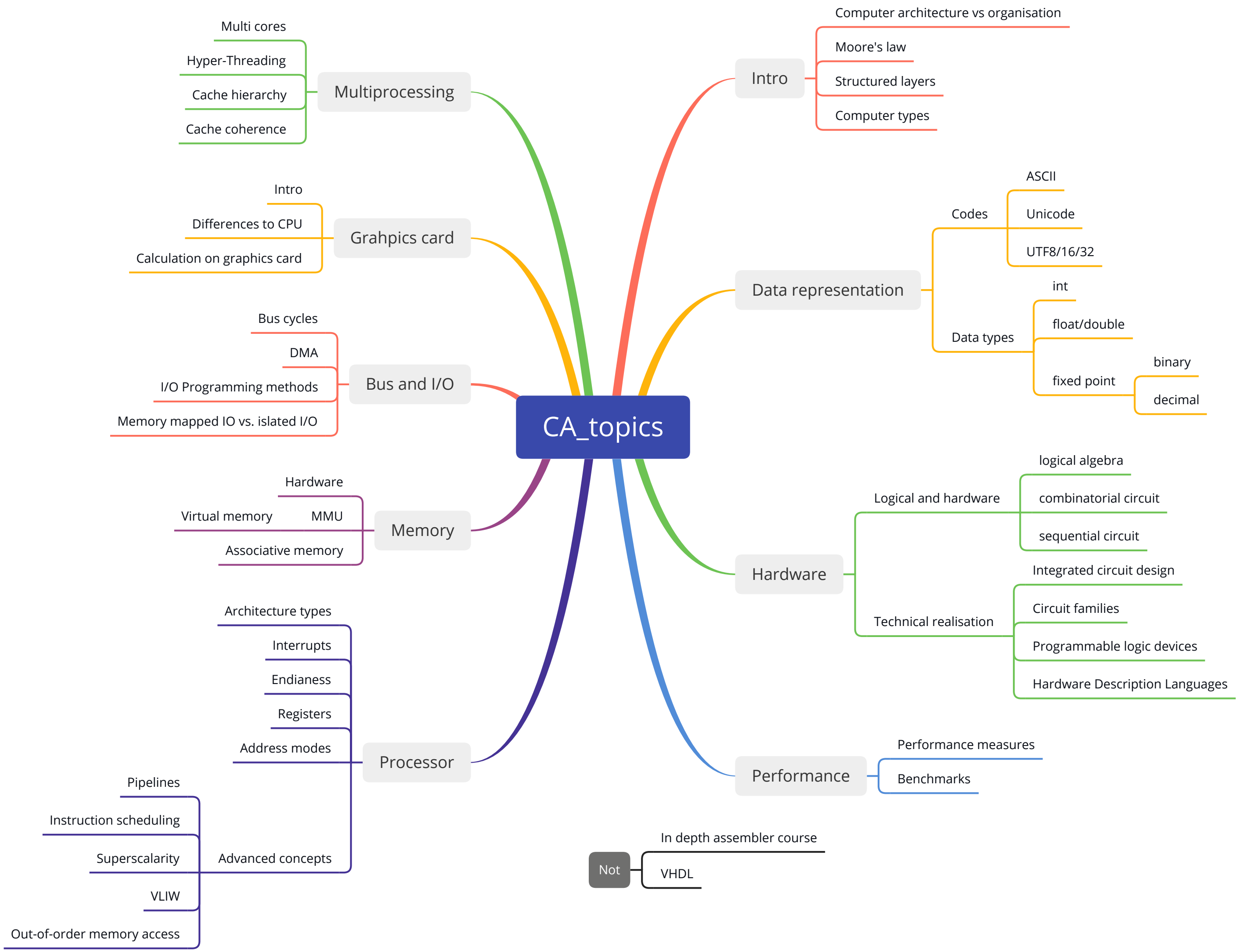
CA 10 – Associative memory

The lecture is based on the work and the documents of Prof. Dr. Theodor Tempelmeier

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

Goal



- CA_topics
  - Multiprocessing
    - Multi cores
    - Hyper-Threading
    - Cache hierarchy
    - Cache coherence
  - Grahpics card
    - Intro
    - Differences to CPU
    - Calculation on graphics card
  - Bus and I/O
    - Bus cycles
    - DMA
    - I/O Programming methods
    - Memory mapped IO vs. islated I/O
  - Memory
    - Hardware
    - MMU
      - Virtual memory
    - Associative memory
  - Processor
    - Architecture types
    - Interrupts
    - Endianess
    - Registers
    - Address modes
    - Advanced concepts
      - Pipelines
      - Instruction scheduling
      - Superscalarity
      - VLIW
      - Out-of-order memory access
  - Intro
    - Computer architecture vs organisation
    - Moore's law
    - Structured layers
    - Computer types
  - Data representation
    - Codes
      - ASCII
      - Unicode
      - UTF8/16/32
    - Data types
      - int
      - float/double
      - fixed point
        - binary
        - decimal
  - Hardware
    - Logical and hardware
      - logical algebra
      - combinatorial circuit
      - sequential circuit
    - Technical realisation
      - Integrated circuit design
      - Circuit families
      - Programmable logic devices
      - Hardware Description Languages
  - Performance
    - Performance measures
    - Benchmarks
- Not
  - In depth assembler course
  - VHDL

CAMPUS Rosenheim
Computer Science

Rosenheim
Technical University
of Applied Sciences

Goal

CA::Associative memory

- Memory hierarchy
- Associative memory
- Translation lookaside buffer
- Cache
- Memory protection

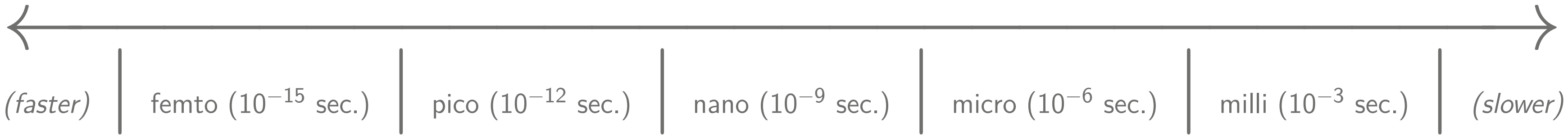
Memory hierarchy

Different kind of memory exists

- On-chip: embedded into IC
- Off-chip: stand alone as a separate hardware
- The more embedded
  - the **smaller** in **hardware** size
  - the **less memory storage** is available
  - the **faster** the **memory**
  - the more **expensive** in price
- The more stand alone
  - the **bigger** in **hardware** size
  - the **more memory storage** is available
  - the **slower** the **memory**
  - the **cheaper** in price

How long does a CPU instruction take?

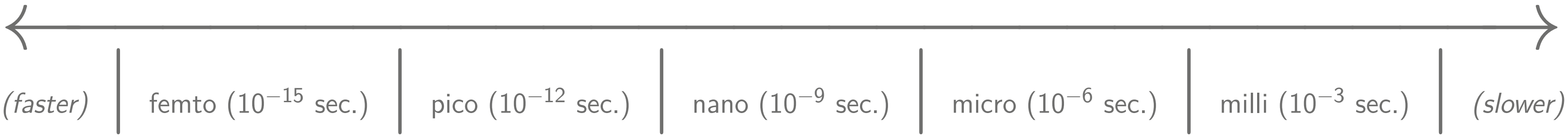
Consider a modern CPU for a notebook or a workstation (e.g. Intel Core i7/i9). **How long** does a **single instruction take** until it is fully executed?



(faster) | femto (10^{-15} sec.) | pico (10^{-12} sec.) | nano (10^{-9} sec.) | micro (10^{-6} sec.) | milli (10^{-3} sec.) | *(slower)*

How long does a memory access take?

Consider a modern memory module for a notebook or a workstation (e.g. DDR4). **How long** does a **load** or **store** from/to **memory take**?



(faster) | femto (10^{-15} sec.) | pico (10^{-12} sec.) | nano (10^{-9} sec.) | micro (10^{-6} sec.) | milli (10^{-3} sec.) | *(slower)*

CAMPUS Rosenheim
Computer Science

Rosenheim
Technical University
of Applied Sciences

Memory hierarchy

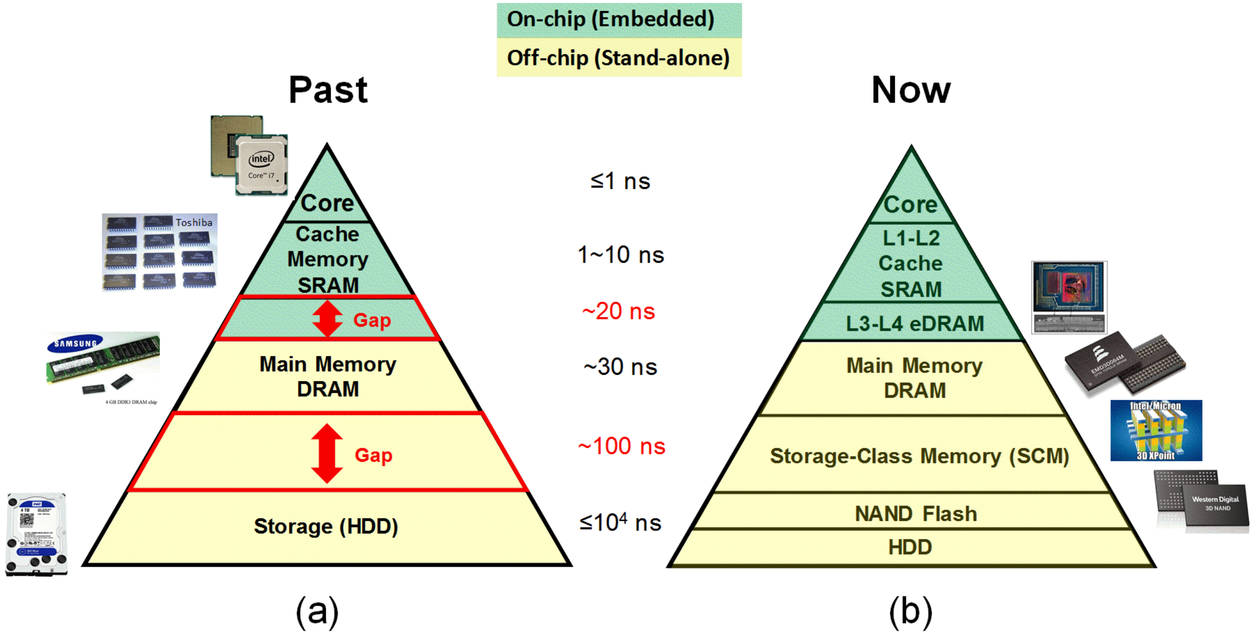


Figure 1.2: Example of memory hierarchy in an ICT system for the past (a) and for now (b). The speed of the component from bottom to top increases, while the storage volume decreases. Those components are divided into on-chip (embedded with compute circuitry on the same chip) and off-chip (stand-alone as a separate chip).

[source: Dissertation, Enlong LIU, Materials and designs of magnetic tunnel junctions with perpendicular magnetic anisotropy for

Associative memory

Key (address)	Value (information)
key_0	$value_0$
key_1	$value_1$
key_2	$value_2$
key_3	$value_3$
key_4	$value_4$
key_5	$value_5$
...	...

In principle

- A key to value store (comparable to a JAVA hashtable/dictionary)
- Key: address
- Value: some information

Properties

- Search for key (address) is done in parallel in hardware!
- Access to information is **very fast**

Usage

- TLB
- Cache

CAMPUS Rosenheim
Computer Science

Rosenheim
Technical University
of Applied Sciences

TLB

Translation lookaside buffer

Address translation

Procedure

1. Load page table(s)
2. Lookup inside page table(s)
3. Address translation

Problem

- Address translation from: virtual to real address required
- Memory access may be required to obtain real address

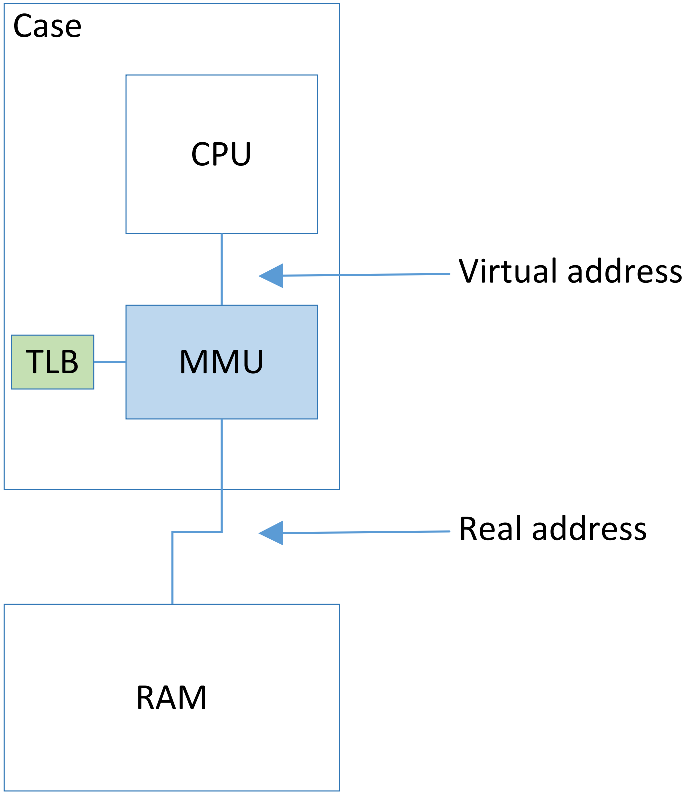
All that takes a lot of time–even with the MMU!

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

Translation lookaside buffer

Idea: Use an associative memory for address translation from virtual to real addresses: **TLB - Translation lookaside buffer**



Key (virt. adr.)	Value (real. adr.)
virtual_base_address$_0$	*real_base_address*$_0$
virtual_base_address$_1$	*real_base_address*$_1$
virtual_base_address$_2$	*real_base_address*$_2$
virtual_base_address$_3$	*real_base_address*$_3$
virtual_base_address$_4$	*real_base_address*$_4$
virtual_base_address$_5$	*real_base_address*$_5$
...	...

virtual_base_address: Virtual address without offset
real_base_address: Real (frame) address without offset

Translation lookaside buffer

Address translation: virtual address to real address

Step 1 (fast way):

- Try to obtain the real address through the TLB
- If the TLB
    - contains the entry: **done**!
    - doesn't contain the entry: go to step 2!

Step 2 (slow way):

- Load page table(s)
- Lookup inside page table(s)
- Address translation
- Store address into TLB

Address translation with TLB always tries step 1 first!

CAMPUS Rosenheim
Computer Science

Rosenheim
Technical University
of Applied Sciences

Cache

Caches inside the CPU

Loading of data and instructions

Before the CPU can process data, it must first be loaded from memory into the registers.

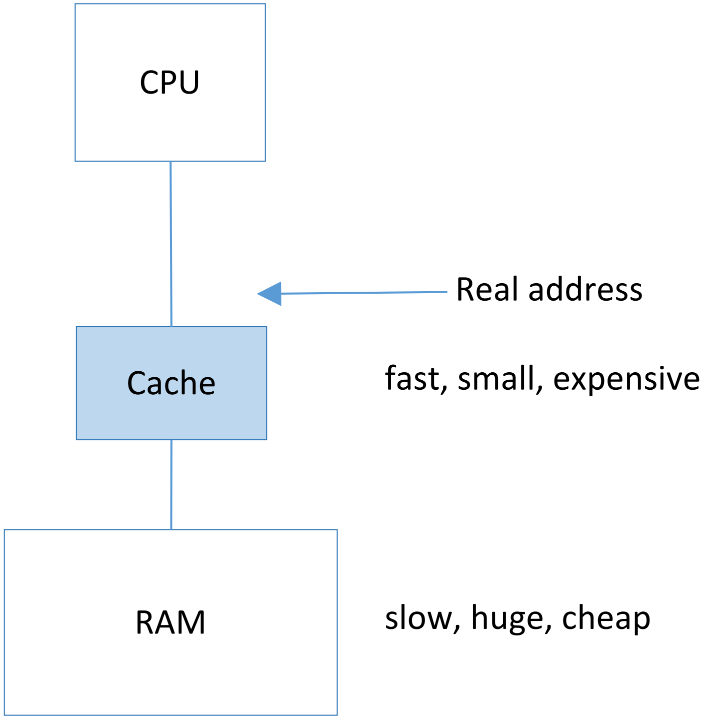
Problem:

- CPU instructions are very fast ($< 1ns$)
- Memory access is slow ($< 30ns$)

We should try to bring the data closer to the CPU!

Cache

Idea: Use an associative memory to store data (parts of the main memory) closer to the CPU: the **cache**!



Key (real adr.)	Value (data)
$real\_address_0$	$data_0$
$real\_address_1$	$data_1$
$real\_address_2$	$data_2$
$real\_address_3$	$data_3$
$real\_address_4$	$data_4$
$real\_address_5$	$data_5$
...	...

Cache details (example)

Given details:

- 16 bit system
- Cache line size: 4 bytes
- Real address: 0x0100
- Data (for given address): 0x1234

Key (real adr.)	Value (data: byte 0 to 3)			
	#0	#1	#2	#3
0x0100	0x12	0x34	?	?

This is the view for a BE (big endian) architecture.

Cache

Data access: read/write from/to memory through the cache

Step 1 (fast way):

- Try to obtain the data through the cache
- If the cache
  - (cache **hit**) contains the entry: **done**!
  - (cache **miss**) doesn't contain the entry: go to step 2!

Step 2 (slow way):

- Load data from memory or store into memory
- Store data into cache

Data access with cache always tries step 1 first!

- If new data is stored in the cache, old data may have to be replaced.
- A cache hit rate of at least 90% should be achieved.

Cache writing strategies

The modified data in the cache have to be written back to the memory at some time.

Write through

- On a write into a word, the data is **immediately transferred** into **cache** and the **memory**.

Write back

- On a write into a word, the data is **only changed in the cache**.
- On the corresponding cache line (entry), the **modified bit** is set.
- Temporarily, the **memory contains invalid data** (the old version(s))
- If the cache line (entry) is **invalidated**, the **data is written back** to the memory

CAMPUS Rosenheim
Computer Science

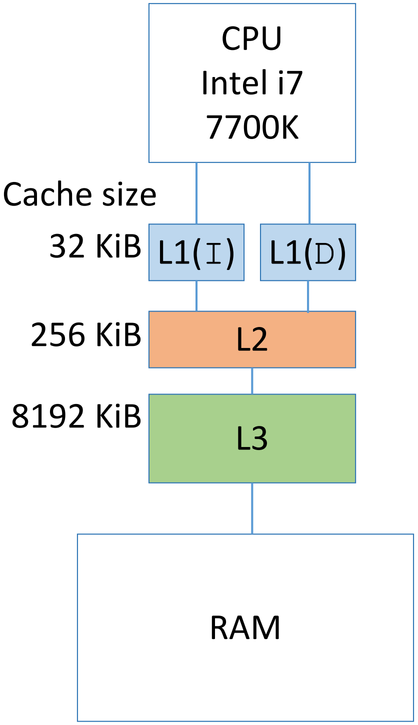
Rosenheim
Technical University
of Applied Sciences

Intel Core i7 caching

How works the Intel Core i7 caching hierarchy?

- Multiple caches with different sizes
- Von Neumann architecture with Harvard architecture ideas!

Cache example Intel Core i7



[simplified schematic view for 1 core]

Intel Core i7 7700K:

- **Split** cache: separate cache for data (D) and instructions (I)
- Cache **hierarchy** with different sizes: L1, L2, and L3
- Cache **line** width 64 bytes

[More infos on cache hierarchy behaviour]

Cache latency:

- `L1(D):  4 cycles`
- `L1(I):  5 cycles`
- `L2   : 12 cycles`
- `L3   : 38 cycles`

[source: https://www.7-cpu.com/cpu/www.7-cpu.com]

CAMPUS Rosenheim
Computer Science

Rosenheim
Technical University
of Applied Sciences

Memory protection

How to protect the memory?

Memory protection unit

A **memory protection unit** (MPU) is a smaller version of a MMU that only contains the **memory protection support**.

- **Privileged** software can define the **memory regions** and its **attributes**.
- If an **access violation** is detected by the MPU a **fault exception** is triggered.

Properties

- Memory region: A fixed base address and a fixed size
- Memory attributes: shared, cached, ...
- Access rights: read, write, execute

A **MPU** can be used for

- Increased security during code execution
- Different privilege levels in an application
- Strict separation of code, data and stack (also between different tasks)

[see: ARM Cortex M3 - MPU, MPU peripheral]

Memory protection

Memory protection with virtual memory and the MMU

For each page the following information is saved

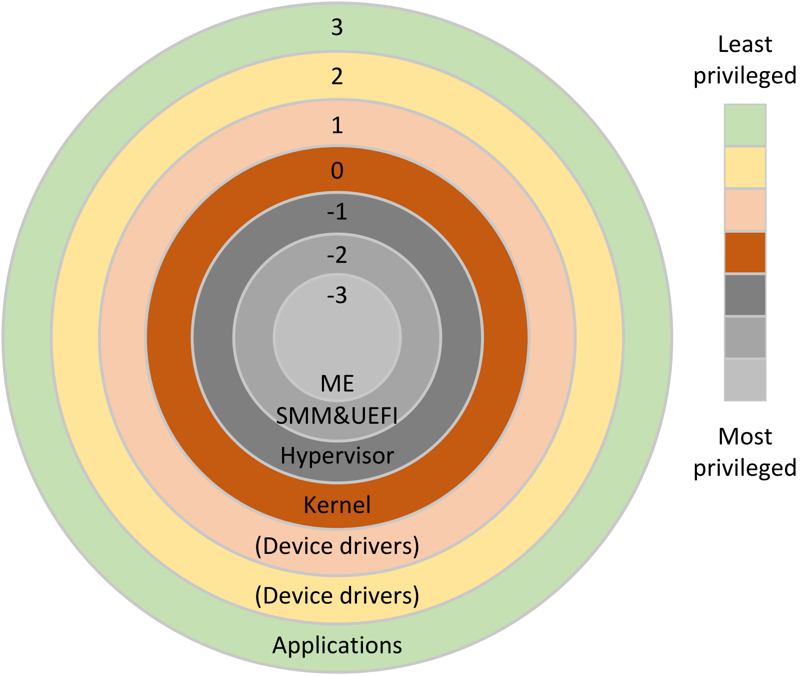
- `R/W` = read/write
- `RO` = read only
- `EO` = execute only
- `U/S` = user/supervisor

This is a basis for memory protection.

A **process** can only access memory through the virtual memory mechanism (MMU) and **can** therefore **only access memory assigned** by the OS.

Memory protection

Intel and AMD x86/64 protection



[source: Is hardware the black hole of computing?]

Privilege rings

- **Ring 3**: User processes: memory management and I/O access only through Ring 0 kernel possible.
- **Ring 1,2**: Device drivers (usually unused)
- **Ring 0**: Supervisor mode: OS kernel
- **Ring -1**: Hypervisor mode
- **Ring -2**: System management mode (power management) + firmware (UEFI)
- **Ring -3**: Manageability engine: remote access, turn computer on/off (MINIX 3)

CAMPUS Rosenheim
Computer Science

Rosenheim
Technical University
of Applied Sciences

Summary and outlook

Summary

- Memory hierarchy
- Associative memory
- Translation lookaside buffer
- Cache
- Memory protection

Outlook

- Bus and I/O