Technische Hochschule Rosenheim
Technical University of Applied Sciences

Start: 8:01

Prof. Dr. Florian Künzner
Technical University of Applied Sciences Rosenheim, Computer Science

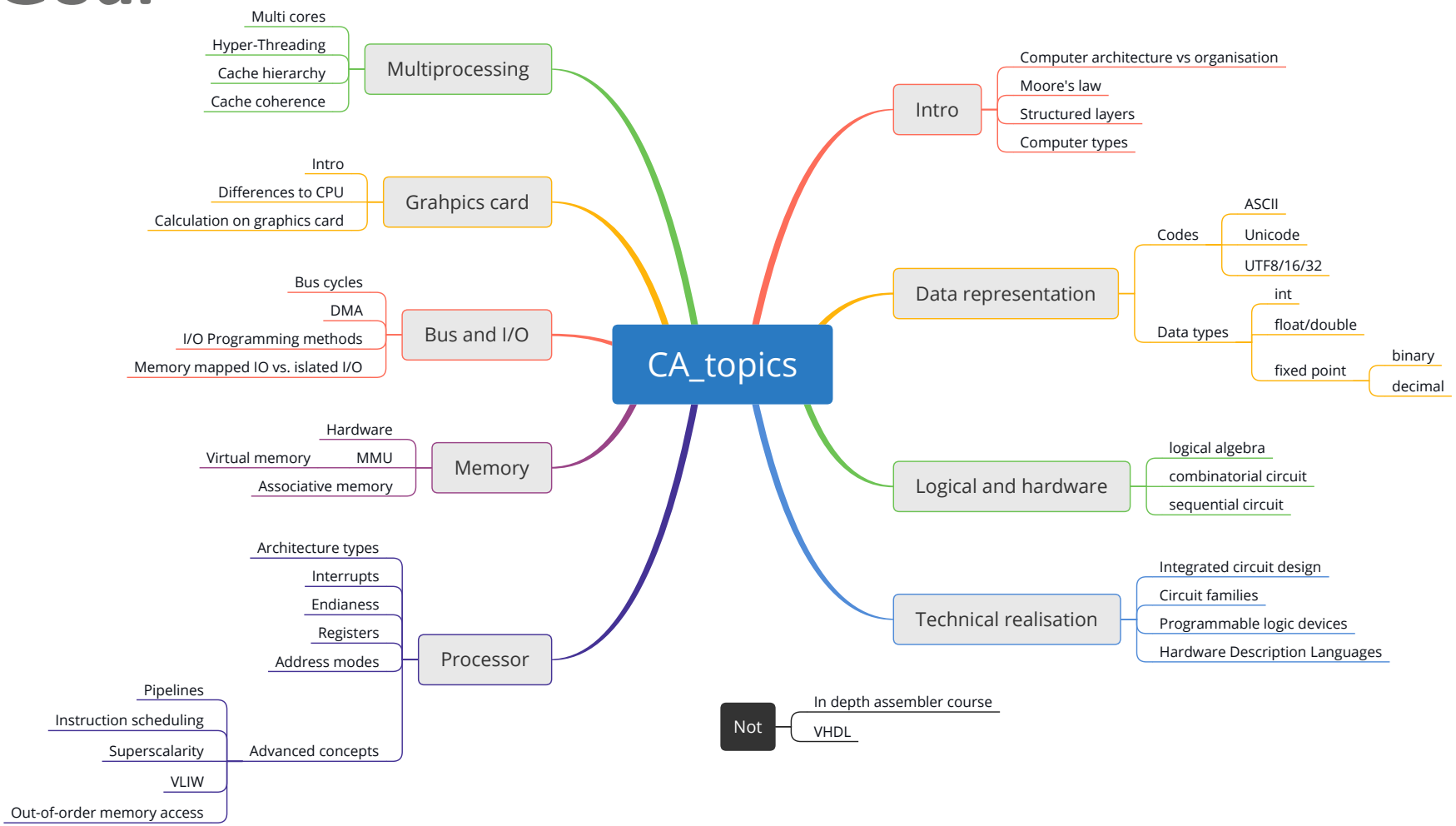
CA 14 – Multiprocessing

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

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Goal



CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Goal

CA::Multiprocessing

- Types of multiprocessing
- CPU performance vs cost
- Memory bottleneck
- Classification of Flynn
- Structures of multi cores and multiprocessors
- MESI protocol

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Types of multiprocessing

CPU core

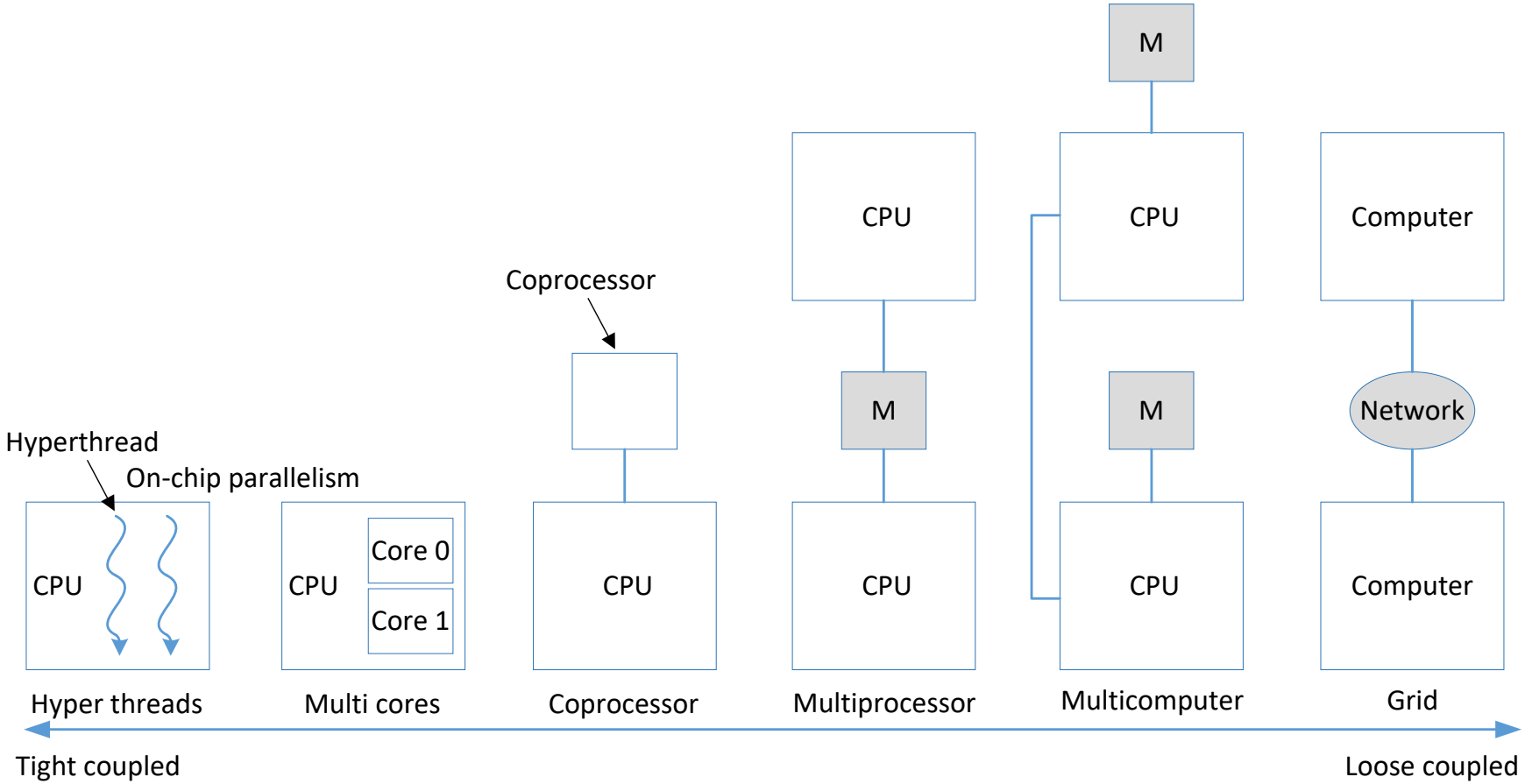
Co-processor

Which types of multiprocessing do you know?

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Types of multiprocessing



[cmp: [1, Fig. 8.1; P. 573]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Hyper threading technology

Observation:

- Not all hardware parts are fully used inside one CPU (core)

Idea:

- Each physical processor is visible as two logical cores to the OS
- Spend some additional hardware to increase hardware utilisation inside one CPU (core)
- The main function of hyper-threading is to increase the number of independent instructions in the pipeline

Only little additional hardware (5%) is required for hyper threading to speed-up execution up to 30%.

Hyper threading technology

Observation:

- Not all hardware parts are fully used inside one CPU (core)

Idea:

- Each physical processor is visible as two logical cores to the OS
- Spend some additional hardware to increase hardware utilisation inside one CPU (core)
- The main function of hyper-threading is to increase the number of independent instructions in the pipeline

Only little additional hardware (5%) is required for hyper threading to speed-up execution up to 30%.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Hyper threading technology

Observation:

- Not all hardware parts are fully used inside one CPU (core)

Idea:

- Each physical processor is visible as two logical cores to the OS
- Spend some additional hardware to increase hardware utilisation inside one CPU (core)
- The main function of hyper-threading is to increase the number of independent instructions in the pipeline

Only little additional hardware (5%) is required for hyper threading to speed-up execution up to 30%.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Hyper threading technology

Observation:

- Not all hardware parts are fully used inside one CPU (core)

Idea:

- Each physical processor is visible as two logical cores to the OS
- Spend some additional hardware to increase hardware utilisation inside one CPU (core)
- The main function of hyper-threading is to increase the number of independent instructions in the pipeline

Only little additional hardware (5%) is required for hyper threading to speed-up execution up to 30%.

Hyper threading technology

Observation:

- Not all hardware parts are fully used inside one CPU (core)

Idea:

- Each physical processor is visible as two logical cores to the OS
- Spend some additional hardware to increase hardware utilisation inside one CPU (core)
- The main function of hyper-threading is to increase the number of independent instructions in the pipeline

Only little additional hardware (5%) is required for hyper threading to speed-up execution up to 30%.

Hyper threading technology

Observation:

- Not all hardware parts are fully used inside one CPU (core)

Idea:

- Each physical processor is visible as two logical cores to the OS
- Spend some additional hardware to increase hardware utilisation inside one CPU (core)
- The main function of hyper-threading is to increase the number of independent instructions in the pipeline

Only little additional hardware (5%) is required for hyper threading to speed-up execution up to 30%.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Hyper threading technology

Observation:

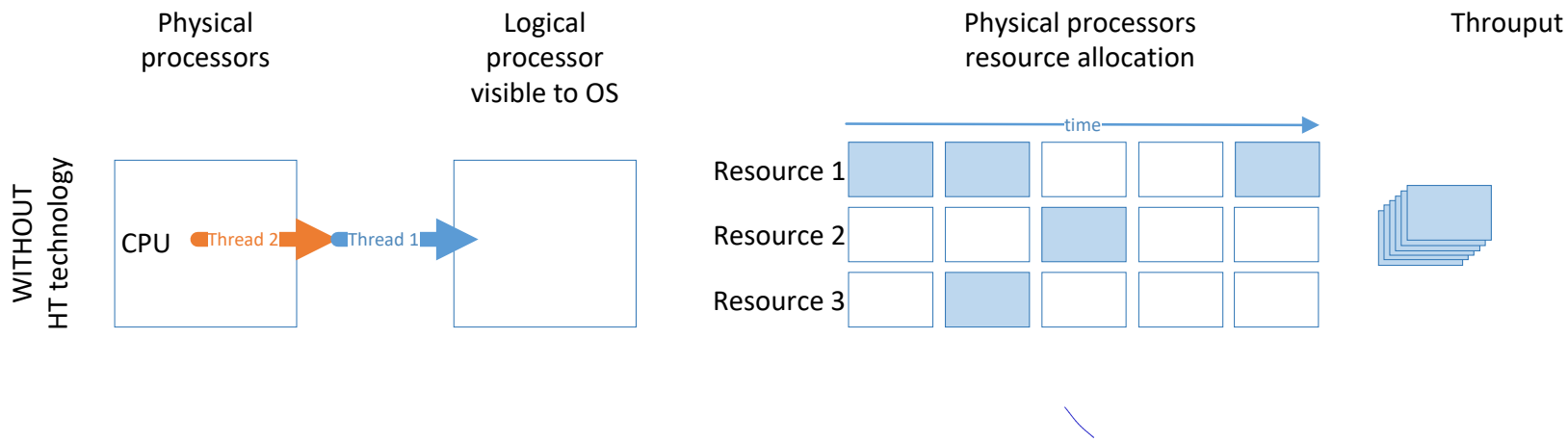
- Not all hardware parts are fully used inside one CPU (core)

Idea:

- Each physical processor is visible as two logical cores to the OS
- Spend some additional hardware to increase hardware utilisation inside one CPU (core)
- The main function of hyper-threading is to increase the number of independent instructions in the pipeline

Only little additional hardware (5%) is required for hyper threading to speed-up execution up to 30%.

Hyper threading technology

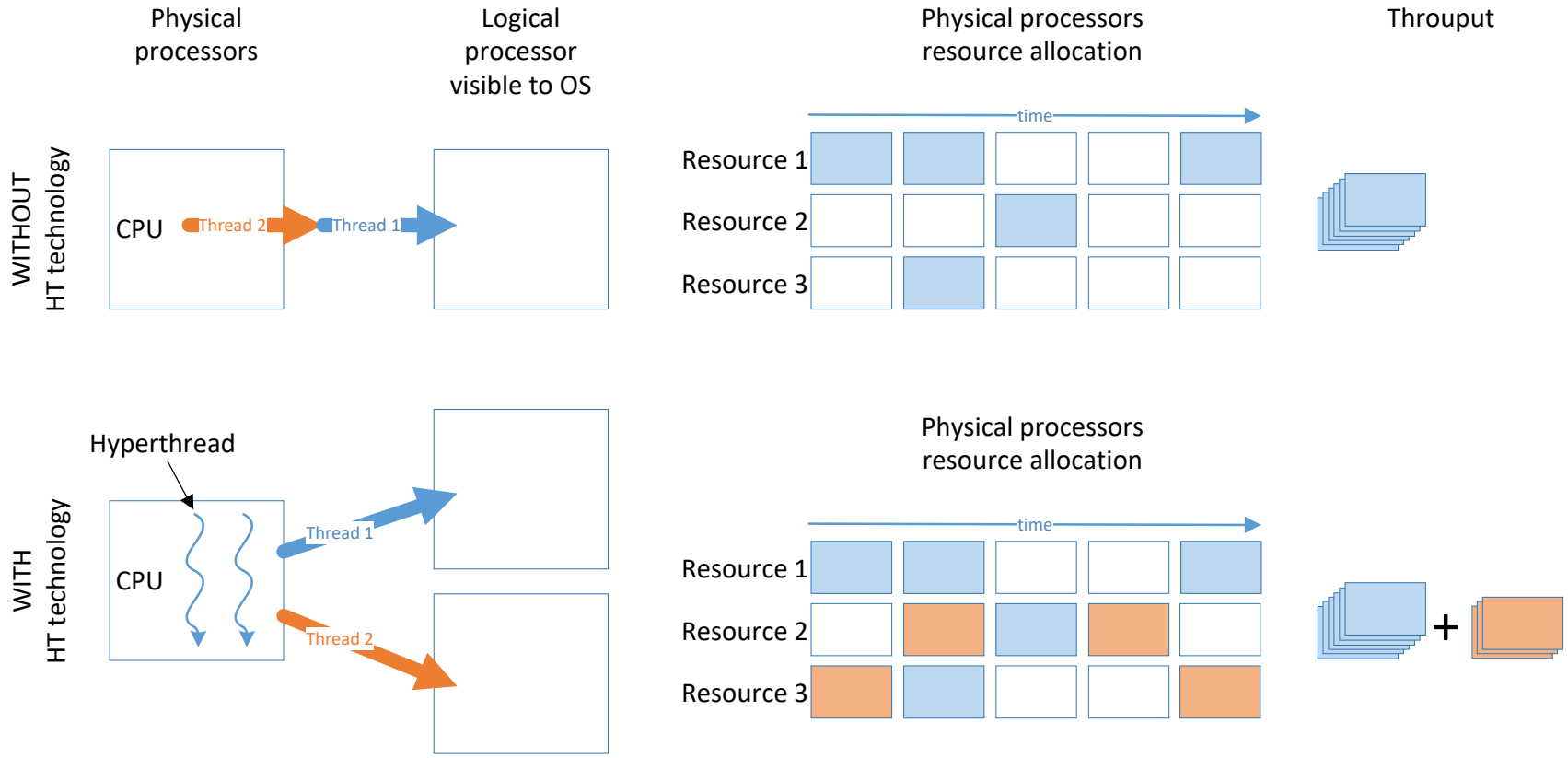


[source (cmp): https://www.techpowerup.com/reviews/Intel/Core_i5_661/14.html]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

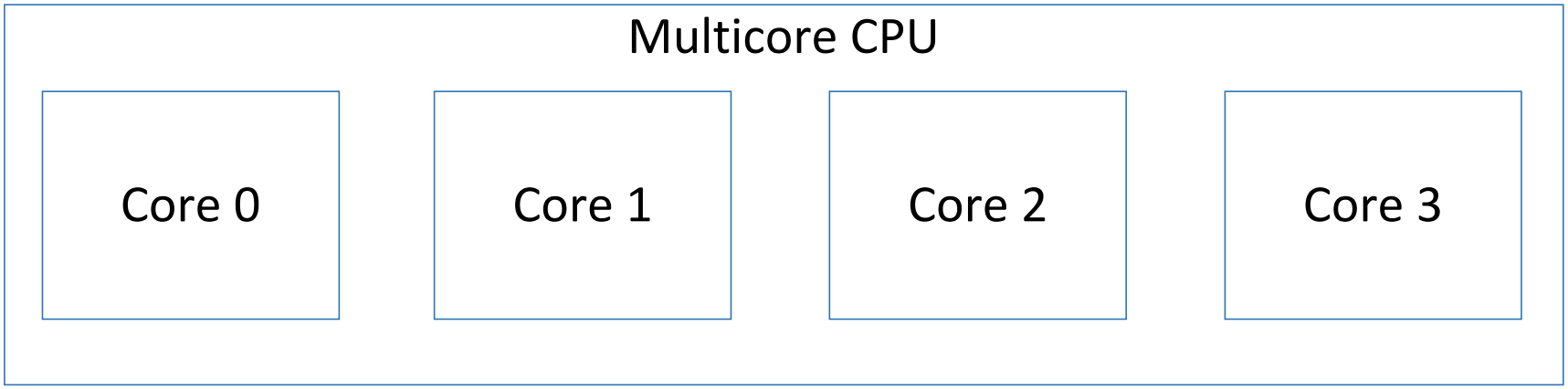
Hyper threading technology



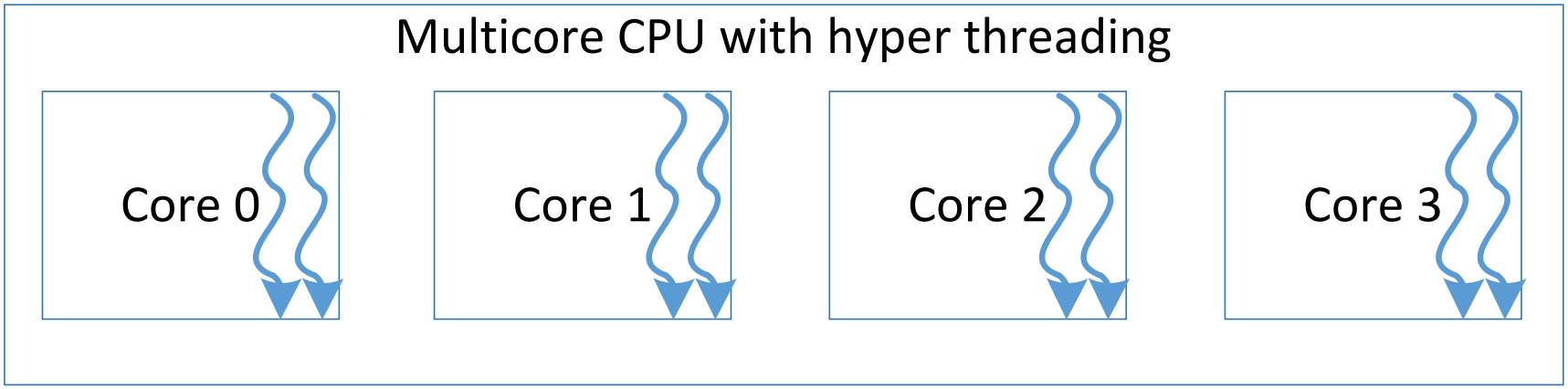
[source (cmp): `https://www.techpowerup.com/reviews/Intel/Core_i5_661/14.html`]

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Multiple cores



Multiple cores with hyper threading



CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Questions?

All right? ⇒ 

Question? ⇒  and use **chat**

or

speak *after* I ask you to

CPU performance vs cost

	Single CPU (Power: 1)	**Multiprocessor** (Power: N)	**Single CPU** (Power: N)
Response time of a **process** (with unloaded machine)	X	X	$\frac{X}{N}$

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

CPU performance vs cost

	Single CPU (Power: 1)	**Multiprocessor** (Power: N)	**Single CPU** (Power: N)
Response time of a **process** (with unloaded machine)	X	X	$\frac{X}{N}$
Throughput of processes (with sufficient processes in ready state)	Y	$\approx N \cdot Y$	$N \cdot Y$

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

CPU performance vs cost

	Single CPU (Power: 1)	**Multiprocessor** (Power: N)	**Single CPU** (Power: N)
Response time of a **process** (with unloaded machine)	X	X	$\frac{X}{N}$
Throughput of processes (with sufficient processes in ready state)	Y	$\approx N \cdot Y$	$N \cdot Y$
Response time of an **application** (consisting of several processes in ready state)	X	$\approx \frac{X}{N}$	$\frac{X}{N}$

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

CPU performance vs cost

	Single CPU (Power: 1)	**Multiprocessor** (Power: N)	**Single CPU** (Power: N)
Response time of a **process** (with unloaded machine)	X	X	$\frac{X}{N}$
Throughput of processes (with sufficient processes in ready state)	Y	$\approx N \cdot Y$	$N \cdot Y$
Response time of an **application** (consisting of several processes in ready state)	X	$\approx \frac{X}{N}$	$\frac{X}{N}$
Cost	$<$	$\ll$	

Questions?

All right? ⇒ 

Question? ⇒  and use **chat**

or

speak *after* I ask you to

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Classification of Flynn

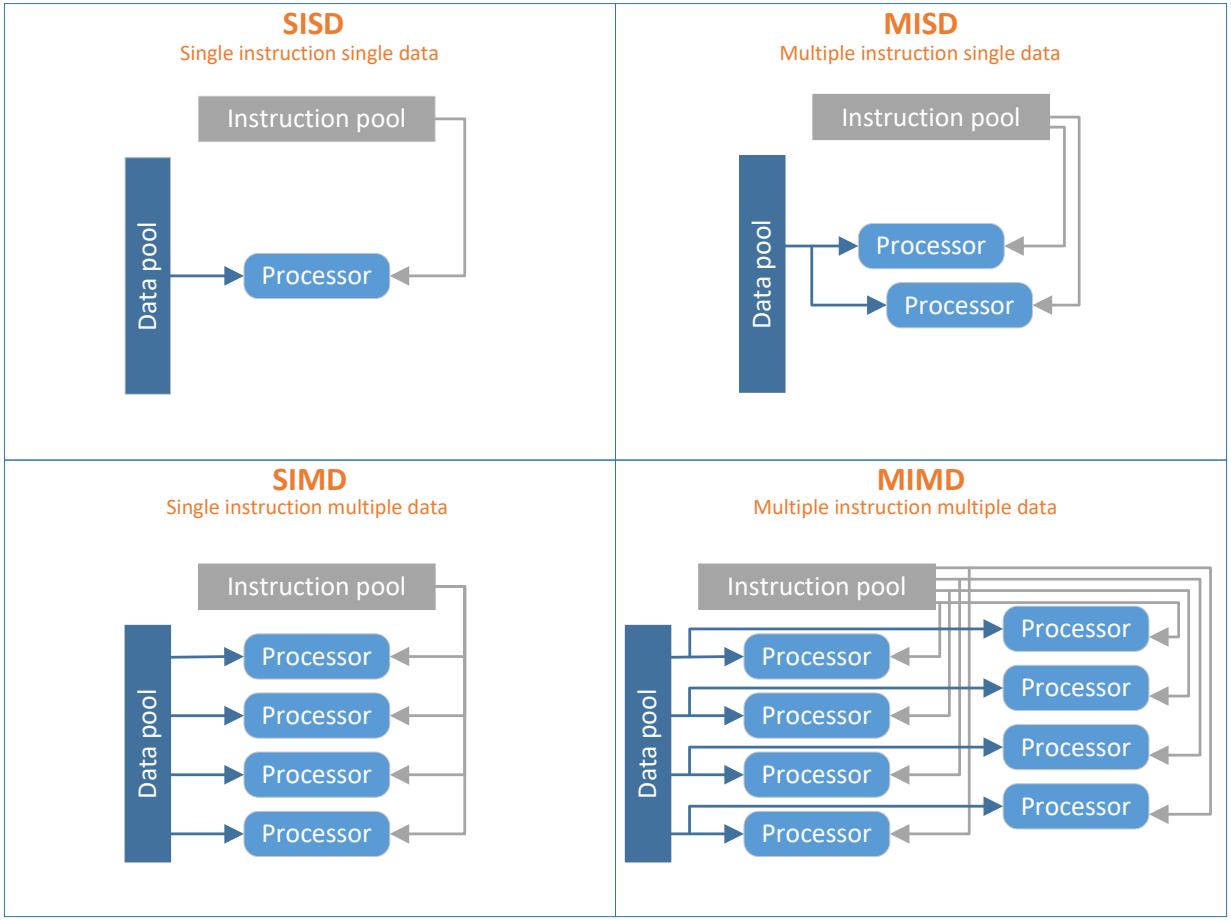
Data streams \ Instruction streams	one (single)	many
one (single)	**SISD** Single instruction single data Traditional von Neumann single CPU computer	**MISD** Multiple instruction single data Not „really“ known
many	**SIMD** Single instruction multiple data Vector processors Fine grained data Parallel computers	**MIMD** Multiple instruction multiple data Multi computers Multiprocessors

[source: https://www.geeksforgeeks.org/computer-architecture-flynns-taxonomy/]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Classification of Flynn

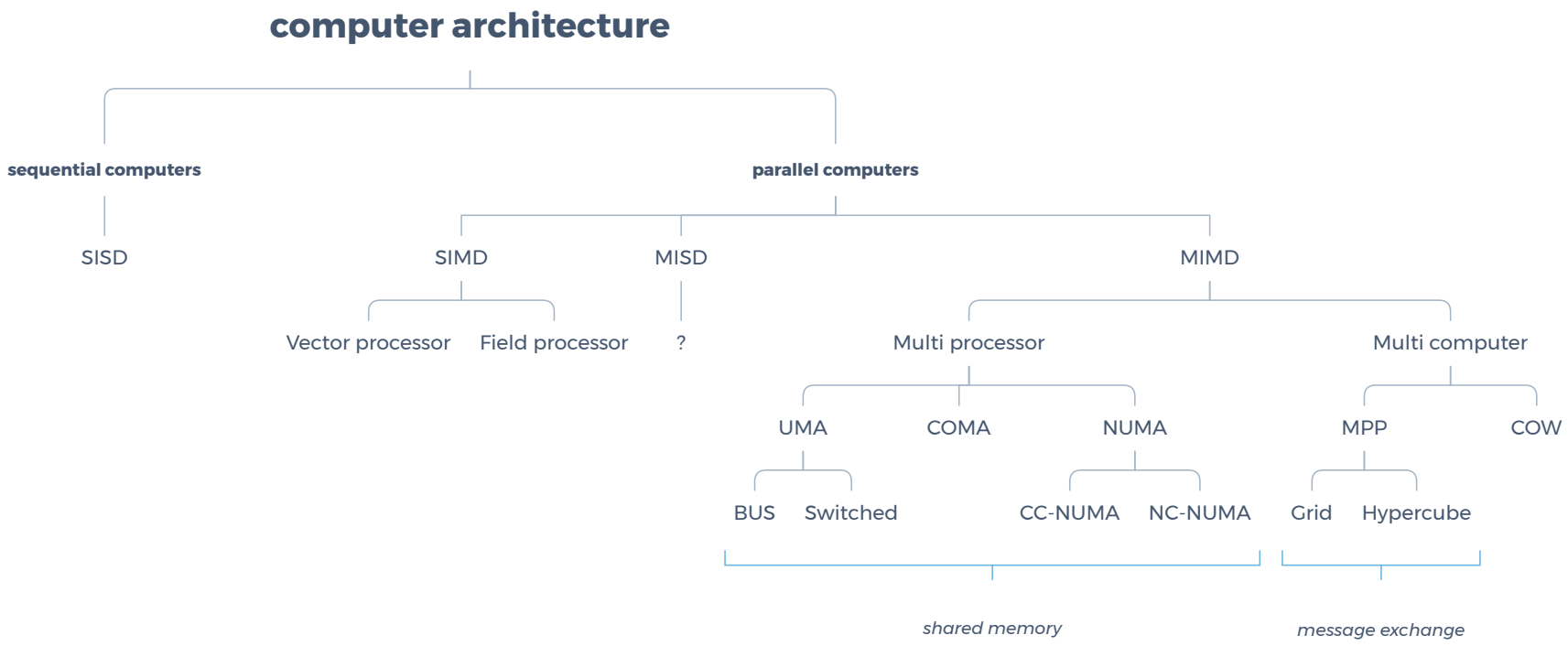


[source: `https://www.geeksforgeeks.org/computer-architecture-flynns-taxonomy/`]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Computer architecture classification



[cmp: [1, P. 609]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Questions?

All right? ⇒ 

Question? ⇒  and use **chat**

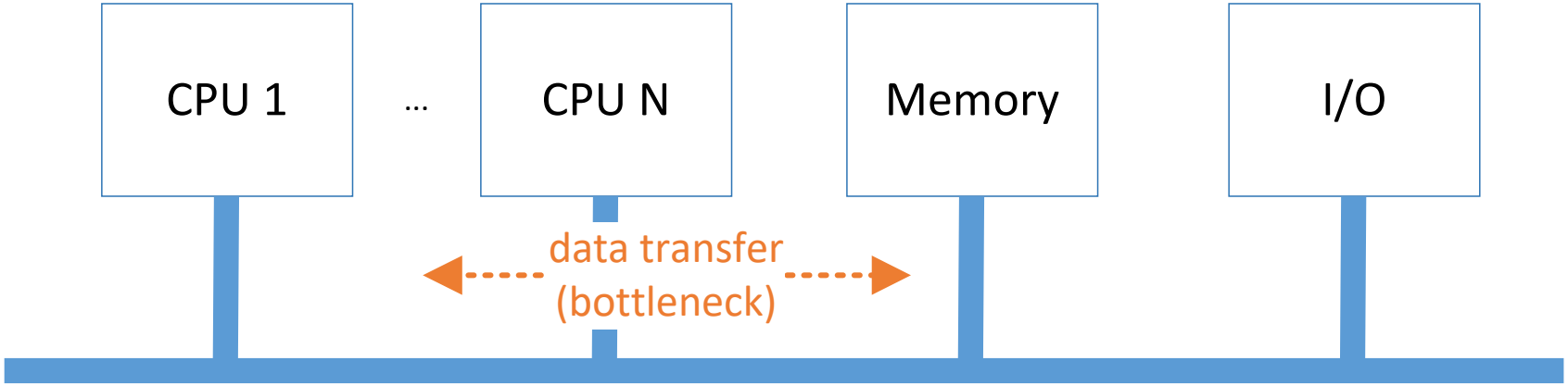
or

speak *after* I ask you to

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

CPU → memory bottleneck



Remember?

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

With more CPUs, the problem gets even worse!

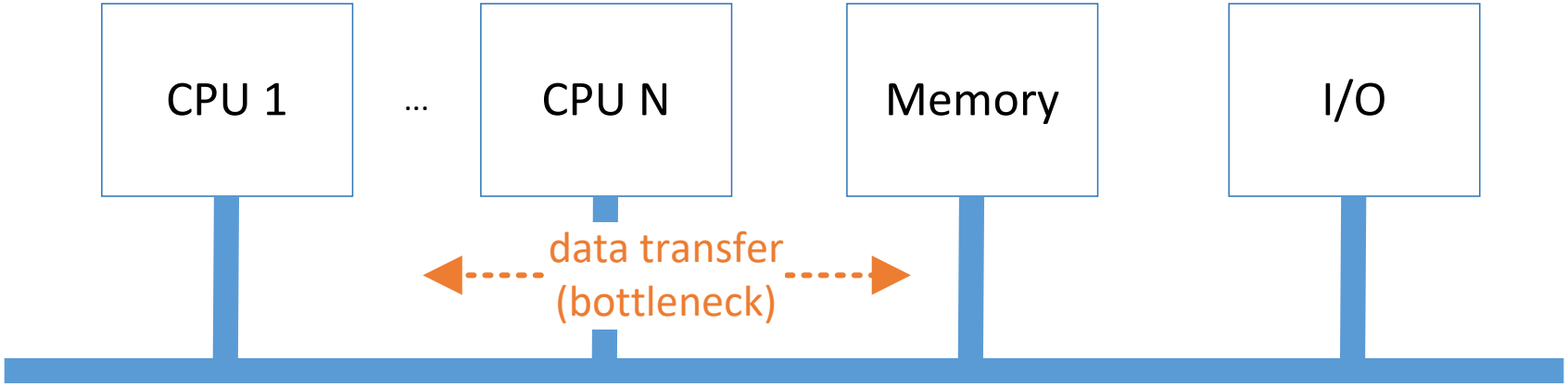
Solution:

- Local caches
- Large caches

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

CPU → memory bottleneck



Remember?

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

With more CPUs, the problem gets even worse!

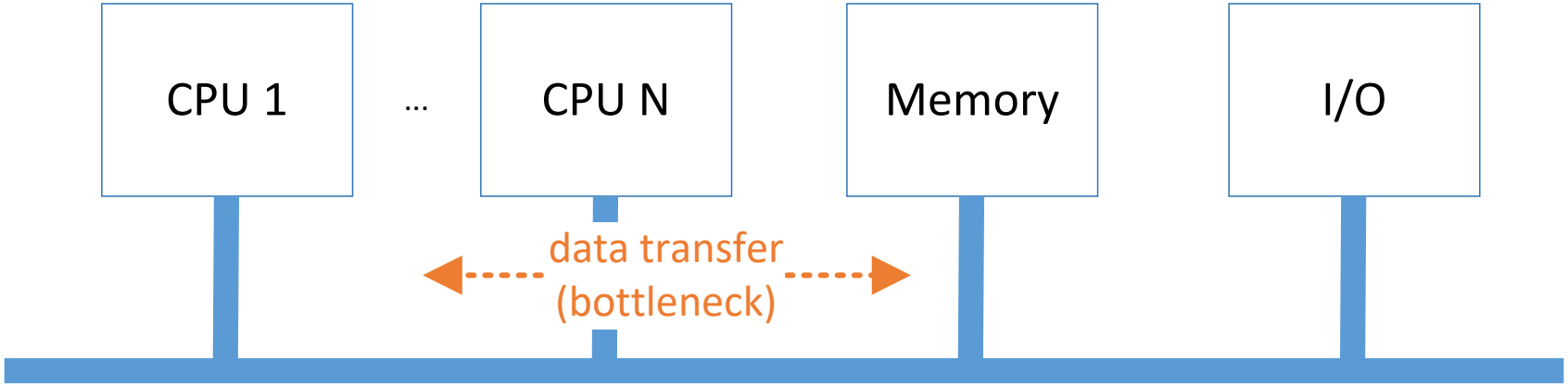
Solution:

- Local caches
- Large caches

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

CPU → memory bottleneck



Remember?

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

With more CPUs, the problem gets even worse!

Solution:

- Local caches
- Large caches

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Questions?

All right? ⇒ 

Question? ⇒  and use **chat**

or

speak *after* I ask you to

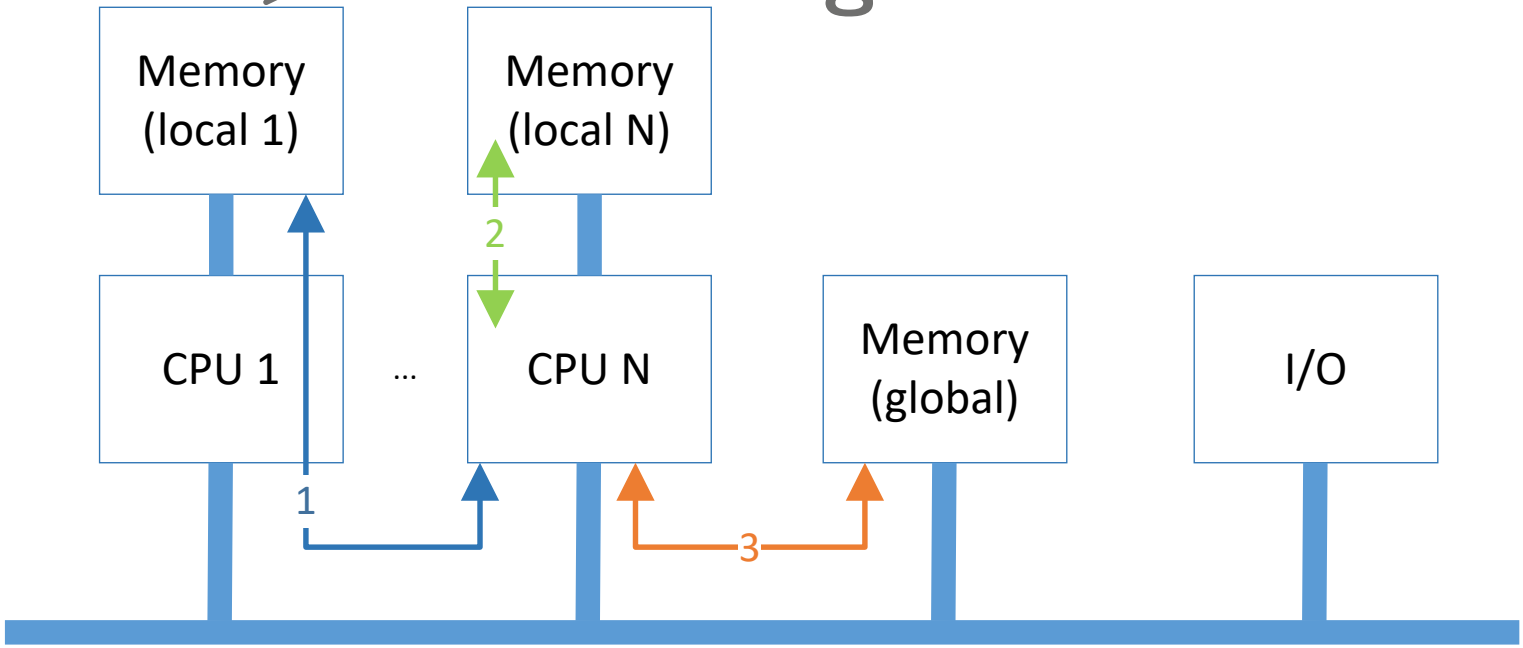
CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Structures of multi cores and multiprocessors

Structure of multi core CPUs with its caches.

Caches -> local storages



NUMA: Non-uniform memory access

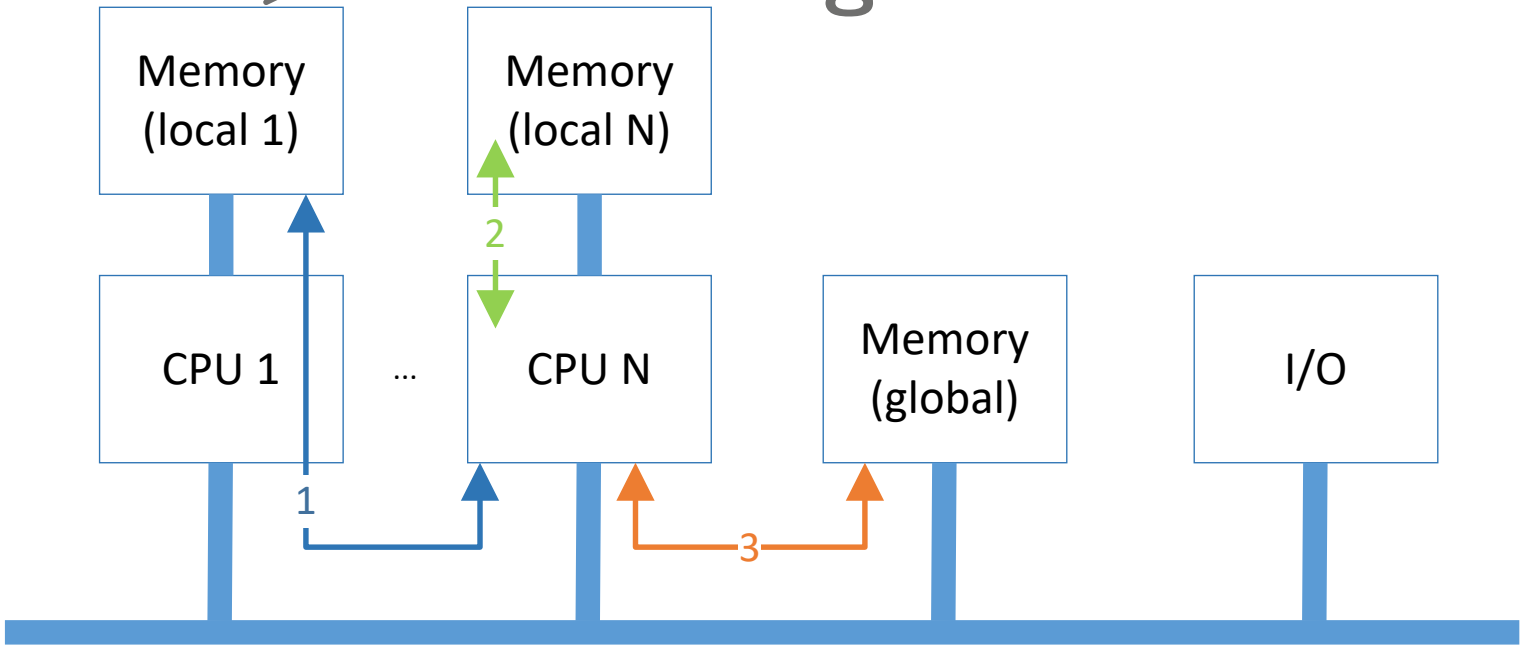
- (1) CPU N access local memory 1
- (2) CPU N access local memory N
- (3) CPU N access global memory

Memory access time depends on the memory location.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Caches -> local storages



NUMA: Non-uniform memory access

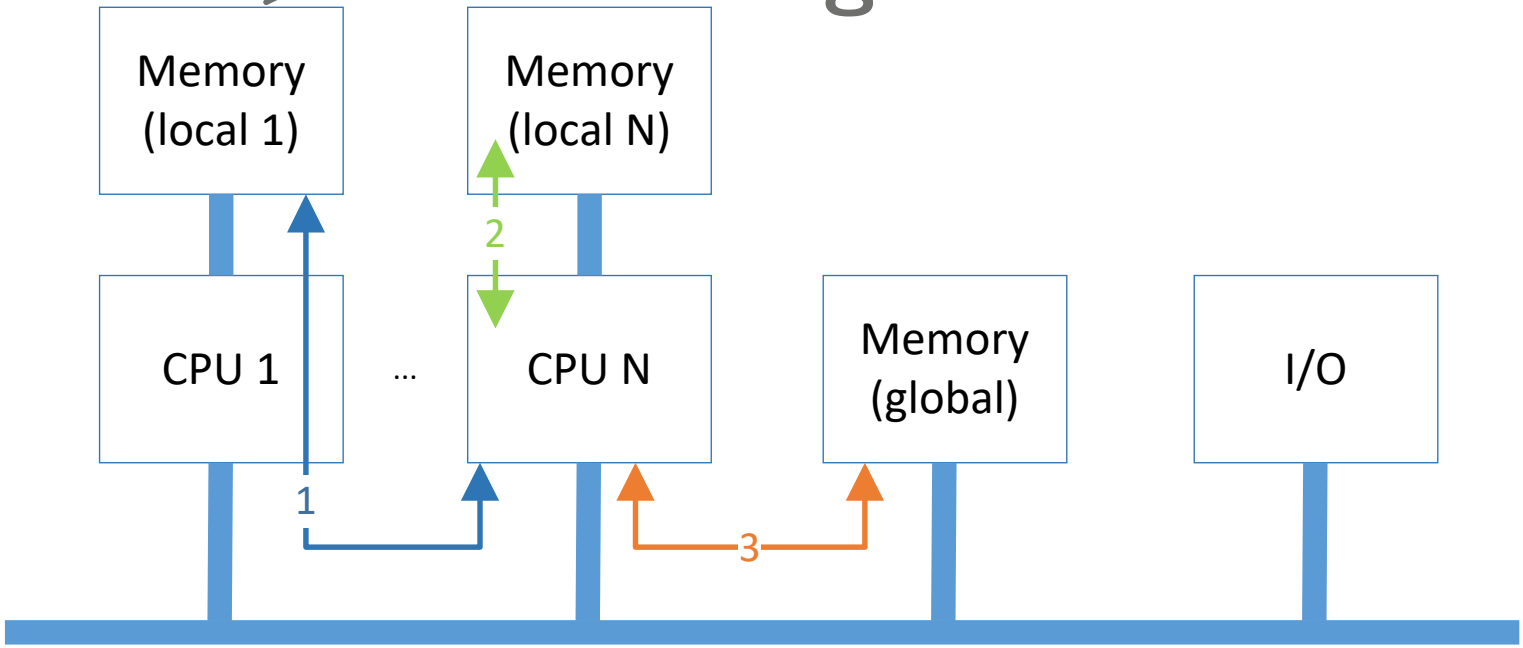
- (1) CPU N access local memory 1
- (2) CPU N access local memory N
- (3) CPU N access global memory

Memory access time depends on the memory location.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Caches -> local storages



NUMA: Non-uniform memory access

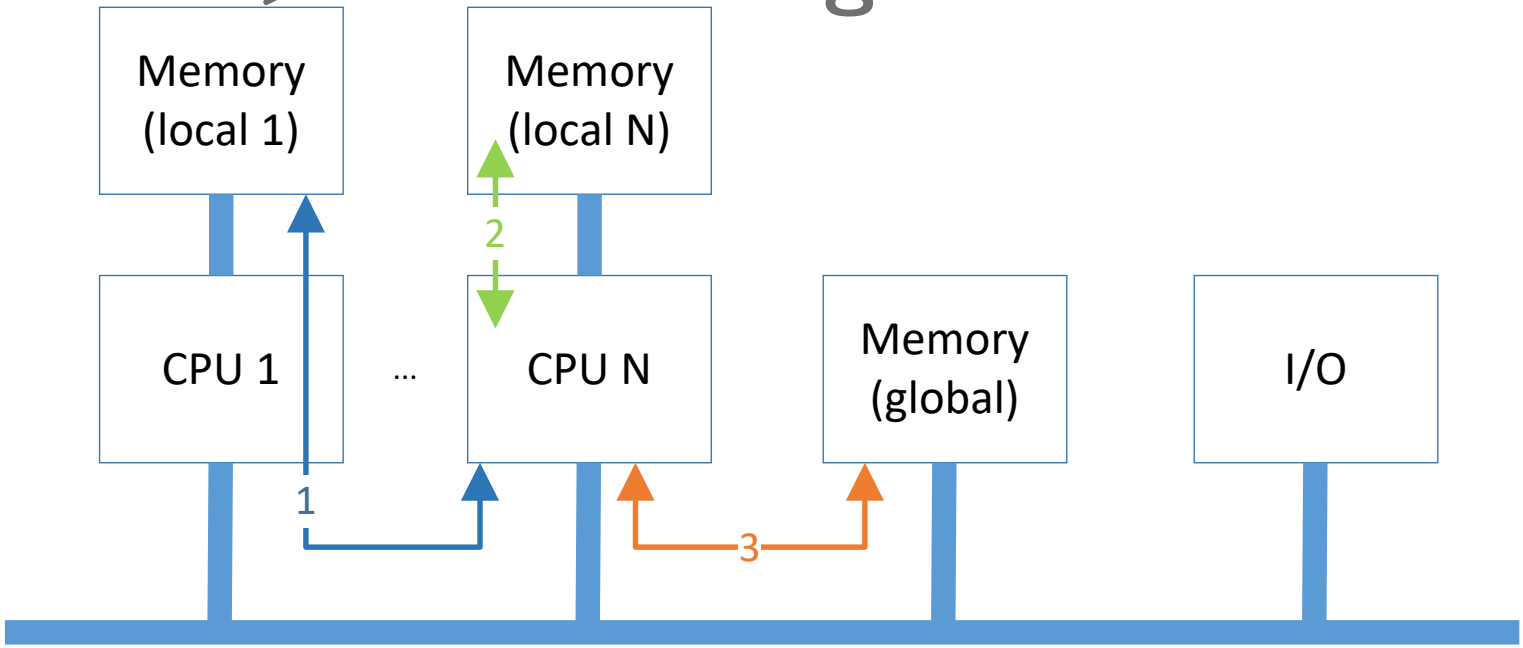
- (1) CPU N access local memory 1
- (2) CPU N access local memory N
- (3) CPU N access global memory

Memory access time depends on the memory location.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Caches -> local storages



NUMA: Non-uniform memory access

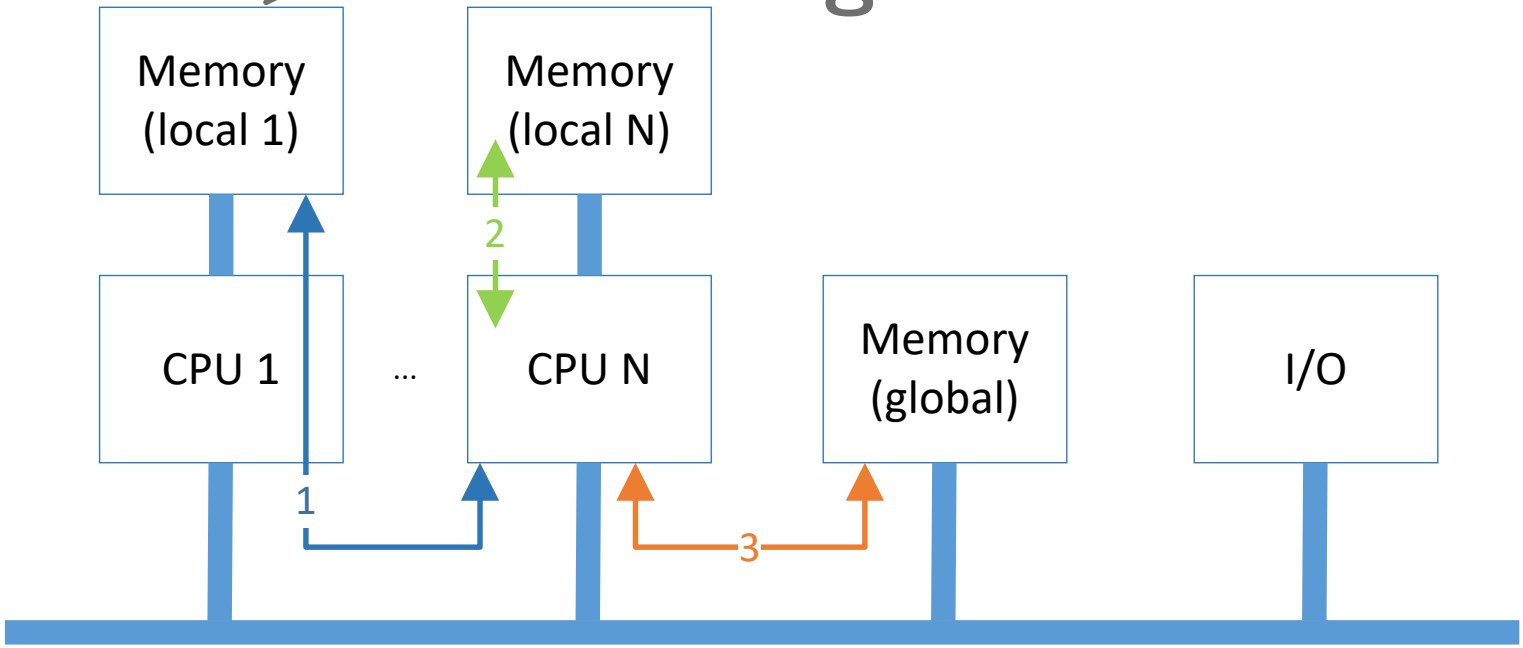
- (1) CPU N access local memory 1
- (2) CPU N access local memory N
- (3) CPU N access global memory

Memory access time depends on the memory location.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Caches -> local storages

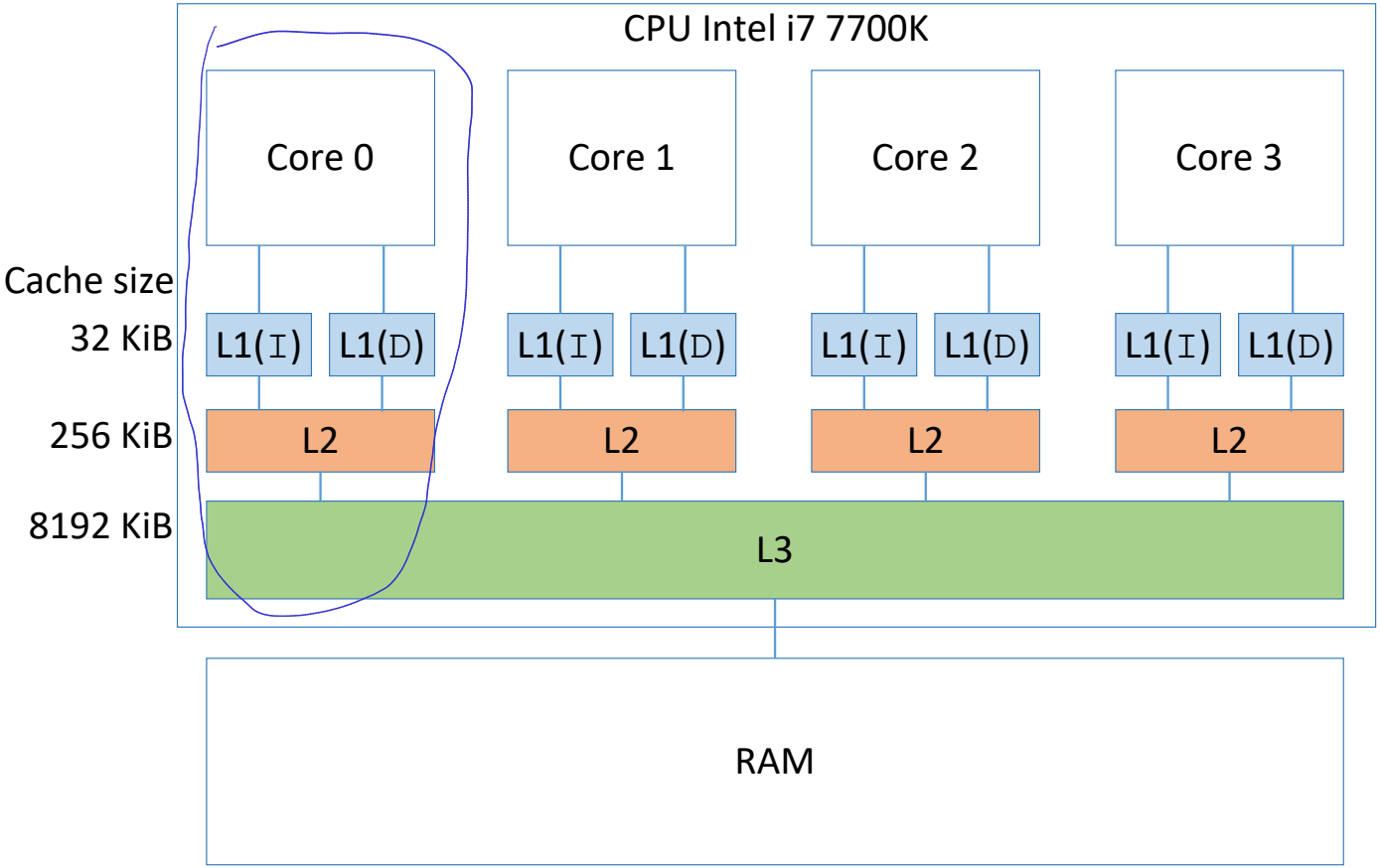


NUMA: Non-uniform memory access

- (1) CPU N access local memory 1
- (2) CPU N access local memory N
- (3) CPU N access global memory

Memory access time depends on the memory location.

Intel i7 7700K example (again)



CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Requirements

Important requirements for multiple cores and multiprocessing

Locking commands

- read-modify-write
- read/write cycles

Example:

- TAS: test and set
- CAS: compare and swap

Communication between processors must be possible

- Global memory
- Interrupt from processor to processor
- Identification of processors

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Requirements

Important requirements for multiple cores and multiprocessing

Locking commands

- read-modify-write
- read/write cycles

Example:

- TAS: test and set
- CAS: compare and swap

Communication between processors must be possible

- Global memory
- Interrupt from processor to processor
- Identification of processors

Requirements

Important requirements for multiple cores and multiprocessing

Locking commands

- read-modify-write
- read/write cycles

Example:

- TAS: test and set
- CAS: compare and swap

Communication between processors must be possible

- Global memory
- Interrupt from processor to processor
- Identification of processors

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Questions?

All right? ⇒ 

Question? ⇒  and use **chat**

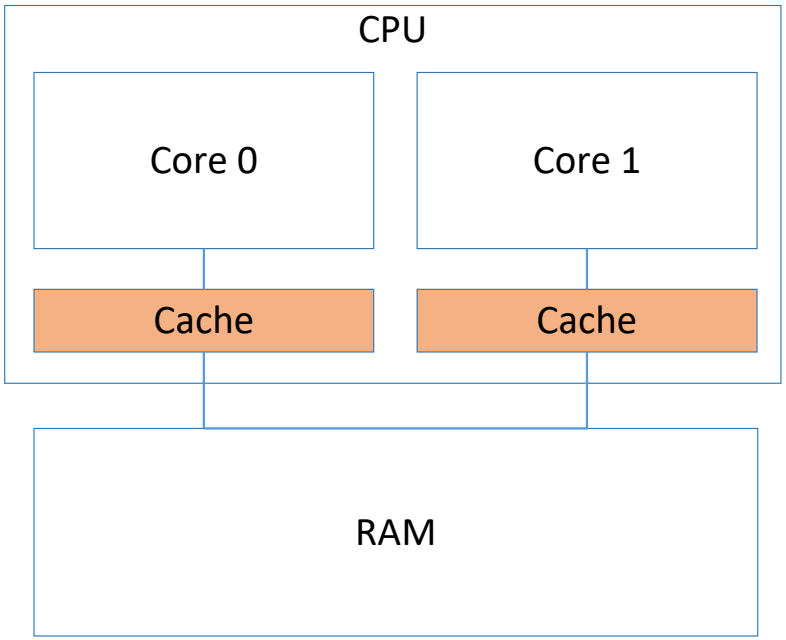
or

speak *after* I ask you to

Cache coherency

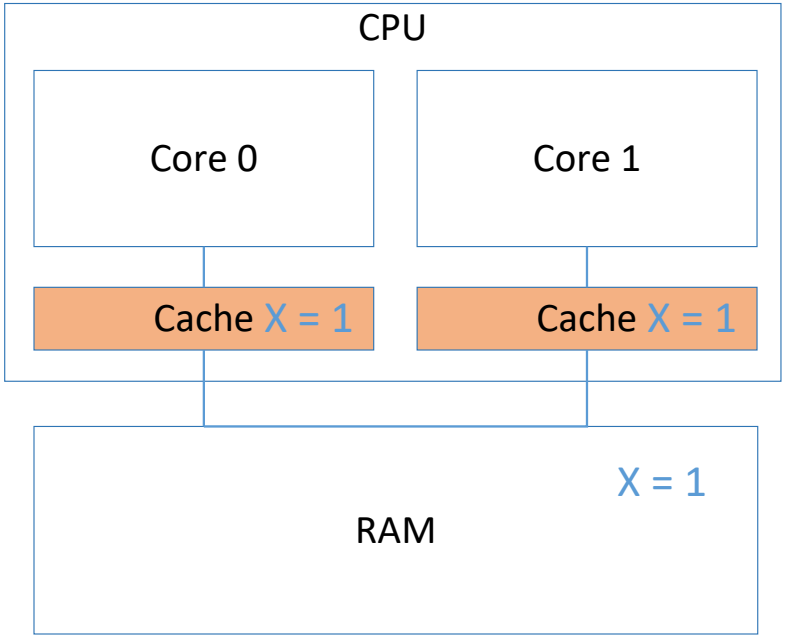
Is there a problem with local memory (caches)?

Cache coherency: Probem!



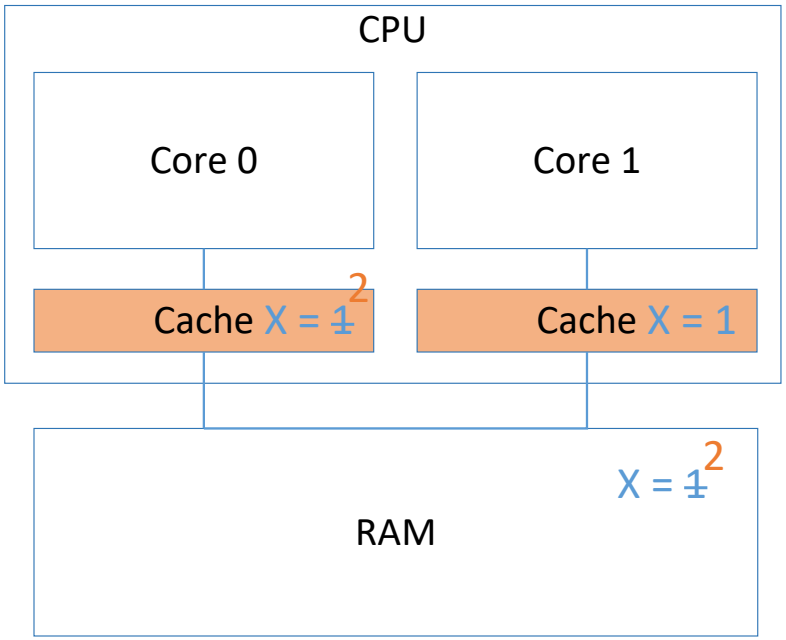
- A dual core CPU with local caches
- Variable X contains value 1
- Both caches contains a copy of X
- Core 0 sets X=2
  (with **write through**)
- **Problem**: Core 1 still reads X==1!!

Cache coherency: Probem!



- A dual core CPU with local caches
- Variable X contains value 1
- Both caches contains a copy of X
- Core 0 sets X=2 (with **write through**)
- **Problem**: Core 1 still reads X==1!!

Cache coherency: Probem!

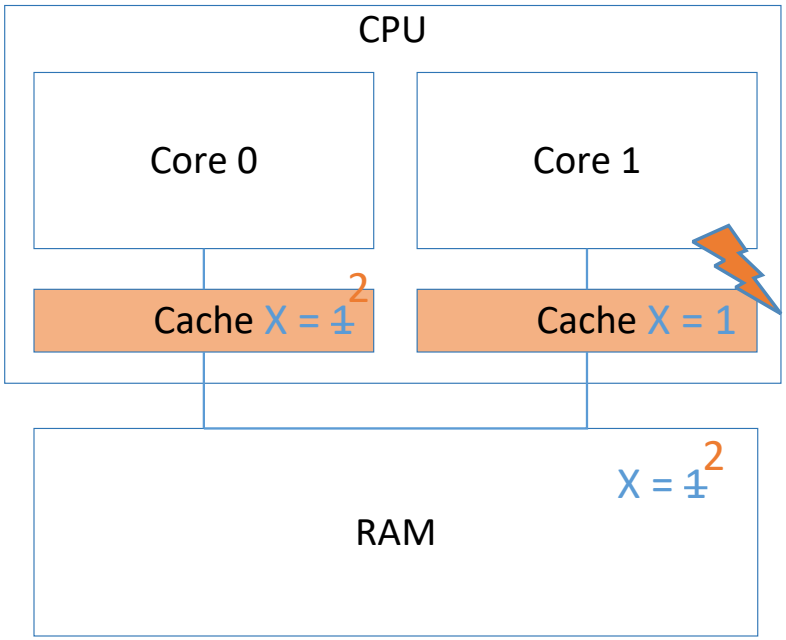


- A dual core CPU with local caches
- Variable X contains value 1
- Both caches contains a copy of X
- Core 0 sets X=2 (with **write through**)
- **Problem**: Core 1 still reads X==1!!

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Cache coherency: Probem!



- A dual core CPU with local caches
- Variable X contains value 1
- Both caches contains a copy of X
- Core 0 sets X=2 (with **write through**)
- **Problem**: Core 1 still reads X==1!!

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Towards a solution

How can we avoid such cache coherency problems?

- **Programming model:** Change the programming model, but than the consistency may not be ensured. Unpleasant thing!
- **Snoop protocols:** All caches operates with **write through** and **listen** to all write cycles on the bus and may set the corresponding entries to invalid. But this is **slow**!
- **MESI protocol:** Various protocols that (at least in some cases) avoid write through.

CC-NUMA: cache coherency with NUMA.

Towards a solution

How can we avoid such cache coherency problems?

- **Programming model:** Change the programming model, but than the consistency may not be ensured. Unpleasant thing!
- **Snoop protocols:** All caches operates with **write through** and **listen** to all write cycles on the bus and may set the corresponding entries to invalid. But this is **slow**!
- **MESI protocol:** Various protocols that (at least in some cases) avoid write through.

CC-NUMA: cache coherency with NUMA.

Towards a solution

How can we avoid such cache coherency problems?

- **Programming model:** Change the programming model, but than the consistency may not be ensured. Unpleasant thing!
- **Snoop protocols:** All caches operates with **write through** and **listen** to all write cycles on the bus and may set the corresponding entries to invalid. But this is **slow**!
- **MESI protocol:** Various protocols that (at least in some cases) avoid write through.

CC-NUMA: cache coherency with NUMA.

Towards a solution

How can we avoid such cache coherency problems?

- **Programming model:** Change the programming model, but than the consistency may not be ensured. Unpleasant thing!
- **Snoop protocols:** All caches operates with **write through** and **listen** to all write cycles on the bus and may set the corresponding entries to invalid. But this is **slow**!
- **MESI protocol:** Various protocols that (at least in some cases) avoid write through.

CC-NUMA: cache coherency with NUMA.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Towards a solution

How can we avoid such cache coherency problems?

- **Programming model:** Change the programming model, but than the consistency may not be ensured. Unpleasant thing!
- **Snoop protocols:** All caches operates with **write through** and **listen** to all write cycles on the bus and may set the corresponding entries to invalid. But this is **slow**!
- **MESI protocol:** Various protocols that (at least in some cases) avoid write through.

CC-NUMA: cache coherency with NUMA.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Cache coherency: The MESI protocol

The MESI protocol **ensures** the **cache coherency** in multi core and multiprocessor systems.

Finite state machine with 4 states:

- `M`: Modified exclusive
- `E`: Exclusive unmodified
- `S`: Shared unmodified
- `I`: Invalid

There is a finite state machine for

- every **cache line**
- in every **cache**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Cache coherency: The MESI protocol

The MESI protocol **ensures** the **cache coherency** in multi core and multiprocessor systems.

Finite state machine with 4 states:

- `M`: Modified exclusive
- `E`: Exclusive unmodified
- `S`: Shared unmodified
- `I`: Invalid

There is a finite state machine for

- every **cache line**
- in every **cache**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Cache coherency: The MESI protocol

The MESI protocol **ensures** the **cache coherency** in multi core and multiprocessor systems.

Finite state machine with 4 states:

- `M`: Modified exclusive
- `E`: Exclusive unmodified
- `S`: Shared unmodified
- `I`: Invalid

There is a finite state machine for

- every **cache line**
- in every **cache**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Cache coherency: The MESI protocol

The MESI protocol **ensures** the **cache coherency** in multi core and multiprocessor systems.

Finite state machine with 4 states:

- `M`: Modified exclusive
- `E`: Exclusive unmodified
- `S`: Shared unmodified
- `I`: Invalid

There is a finite state machine for

- every **cache line**
- in every **cache**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Cache coherency: The MESI protocol

The MESI protocol **ensures** the **cache coherency** in multi core and multiprocessor systems.

Finite state machine with 4 states:

- `M`: Modified exclusive
- `E`: Exclusive unmodified
- `S`: Shared unmodified
- `I`: Invalid

There is a finite state machine for

- every **cache line**
- in every **cache**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Cache coherency: The MESI protocol

The MESI protocol **ensures** the **cache coherency** in multi core and multiprocessor systems.

Finite state machine with 4 states:

- `M`: Modified exclusive
- `E`: Exclusive unmodified
- `S`: Shared unmodified
- `I`: Invalid

There is a finite state machine for

- every **cache line**
- in every **cache**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Cache coherency: The MESI protocol

The MESI protocol **ensures** the **cache coherency** in multi core and multiprocessor systems.

Finite state machine with 4 states:

- `M`: Modified exclusive
- `E`: Exclusive unmodified
- `S`: Shared unmodified
- `I`: Invalid

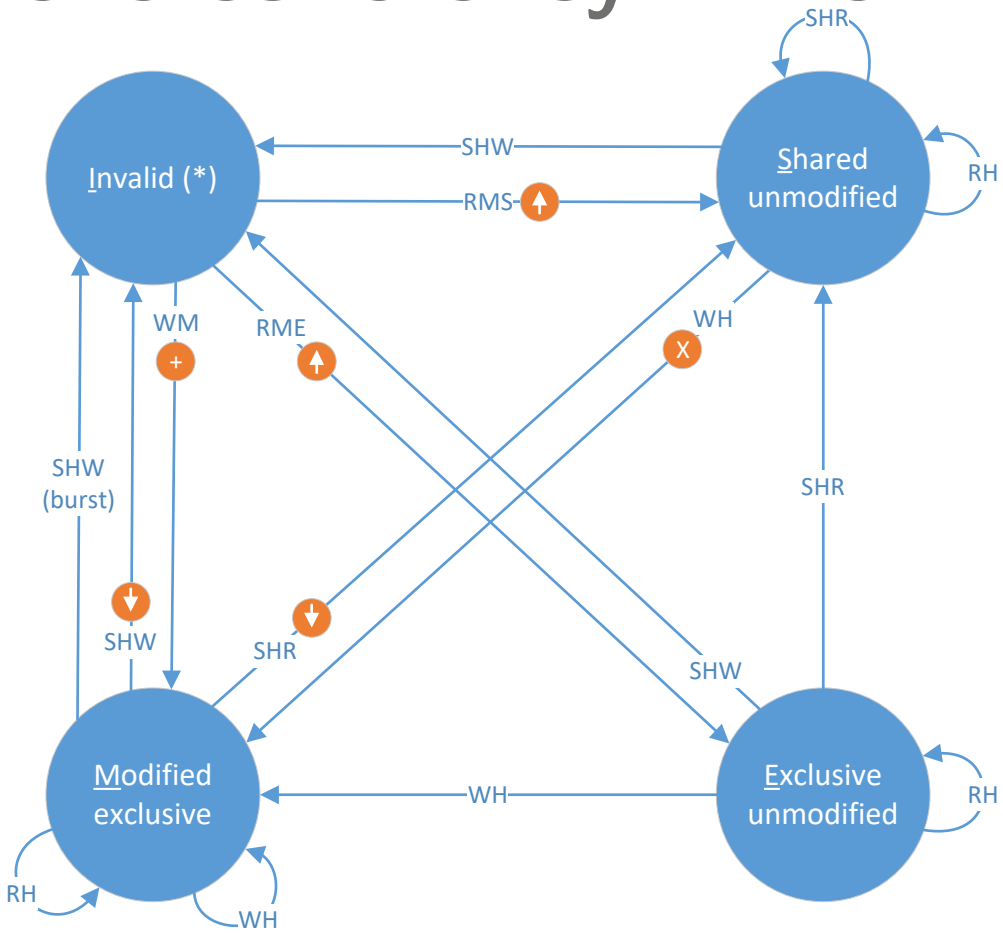
There is a finite state machine for

- every **cache line**
- in every **cache**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Cache coherency: The MESI protocol



RH	Read Hit
RMS	Read miss, shared
RME	Read miss, exclusive
WH	Write hit
WM	Write miss
SHR	Snoop hit on a read
SHW	Snoop hit on a write or Read-with-intent-to-modify or invalidate

- X — Invalidate transaction
- \+ — Read-with-intent-to-modify
- ↑ — Cache line fill
- ↓ — Dirty line copyback

(*) On a miss, the old line is first invalidated and copied-back if exclusive modified

[source: compare: Diefendorff, Allen: Organization of the Motorola 88110, IEEE Micro, April 1992]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Cache coherency: Modern MESI

The MESI protocol is with extensions still in use.

MOESI: AMD64

- MESI states as in the original MESI protocol
- O: Owned
- Allows **moving** a (**dirty**) cache line around caches without updating main memory

MESIF: Intel

- MESI states as in the original MESI protocol
- F: Forward
- Allows **copying** (**clean**) data from one cache to another without accessing main memory

[comparison: https://en.wikipedia.org/wiki/MESIF_protocol]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Cache coherency: Modern MESI

The MESI protocol is with extensions still in use.

MOESI: AMD64

- MESI states as in the original MESI protocol
- O: Owned
- Allows **moving** a (**dirty**) cache line around caches without updating main memory

MESIF: Intel

- MESI states as in the original MESI protocol
- F: Forward
- Allows **copying** (**clean**) data from one cache to another without accessing main memory

[comparison: https://en.wikipedia.org/wiki/MESIF_protocol]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Cache coherency: Modern MESI

The MESI protocol is with extensions still in use.

MOESI: AMD64

- MESI states as in the original MESI protocol
- O: Owned
- Allows **moving** a (**dirty**) cache line around caches without updating main memory

MESIF: Intel

- MESI states as in the original MESI protocol
- F: Forward
- Allows **copying** (**clean**) data from one cache to another without accessing main memory

[comparison: https://en.wikipedia.org/wiki/MESIF_protocol]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Questions?

All right? ⇒ 

Question? ⇒  and use **chat**

or

speak *after* I ask you to

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Summary

Summary

- Types of multiprocessing
- CPU performance vs cost
- Memory bottleneck
- Classification of Flynn
- Structures of multi cores and multiprocessors
- MESI protocol

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Summary

Summary

- Types of multiprocessing
- CPU performance vs cost
- Memory bottleneck
- Classification of Flynn
- Structures of multi cores and multiprocessors
- MESI protocol