

# HW/SW Codesign Concepts & Synthesis

Prof. Dr. Rolf Drechsler  
Dr. Muhammad Hassan  
M.Sc. Jan Zielasko  
M.Sc. Milan Funck

# Announcements

- Do not forget to register for the exams
  - We will also send a reminder email today
  - Tutorial tomorrow?

# Last Time ...

- Why is HW/SW Codesign important?
- Design Methodology
- Specification and Modeling
  - Graph Modeling
    - Control Flow Graph
    - Data Flow Graph
    - Control Data Flow Graph

# Today's Agenda

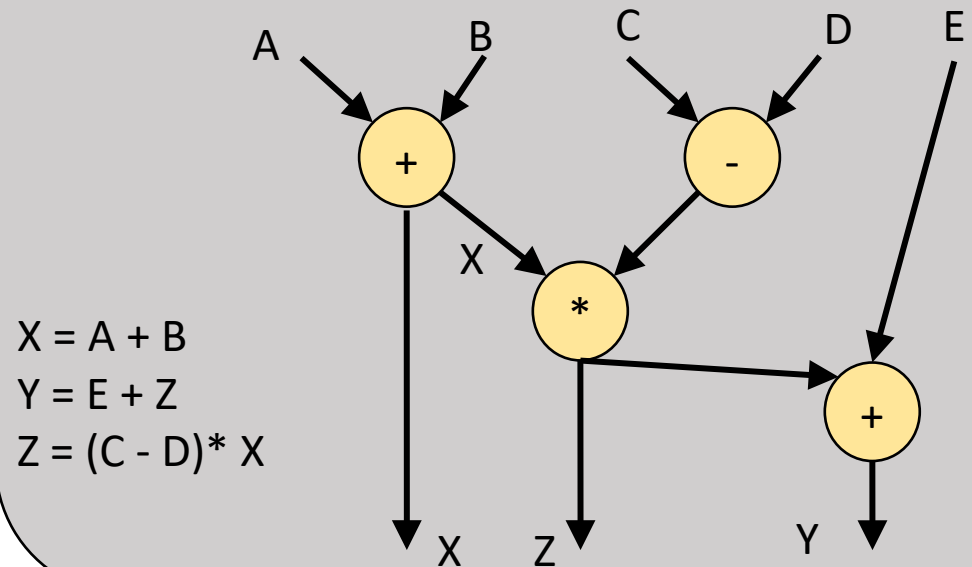
- Why is HW/SW Codesign important?
- Design Methodology
- Specification and Modeling
  - Graph Modeling
    - Control Flow Graph
    - Data Flow Graph
    - Control Data Flow Graph
- Model Characteristics
  - Concurrency
  - Hierarchy
  - Communication
    - Shared Memory
    - Message Passing
  - Synchronization

# Model Characteristics

- **Concurrency:** often simpler to split system into concurrent sub-systems: e.g. 2 FSMs with 1 state is simpler than 1 FSM with 2 states.

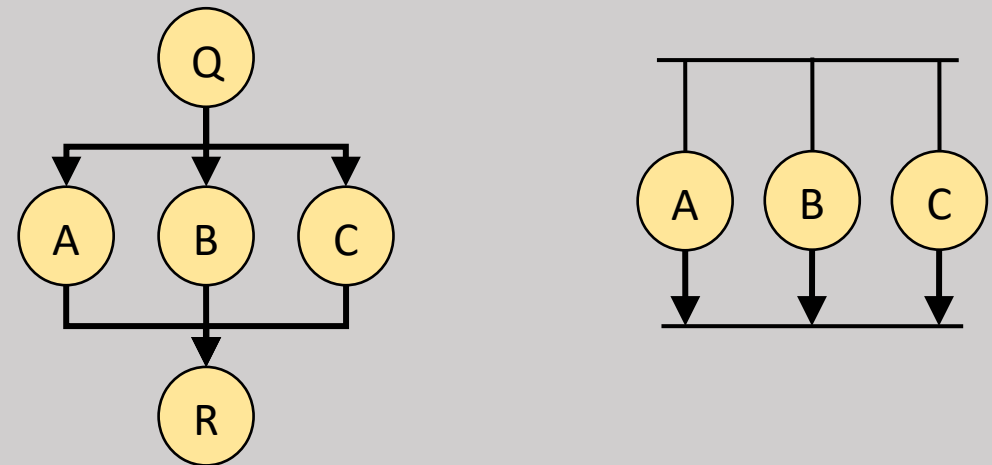
## Data oriented concurrency:

No specific order, single assignment rule



## Control oriented concurrency:

Explicit control instructions (fork-join concurrent behaviour) determine order of operations



# Model Characteristics

- State Transitions

- Transitions depend on conditions/states
- System with  $N$  states can have up to  $N^2$  transitions
- Control centric behavior



- Hierarchy

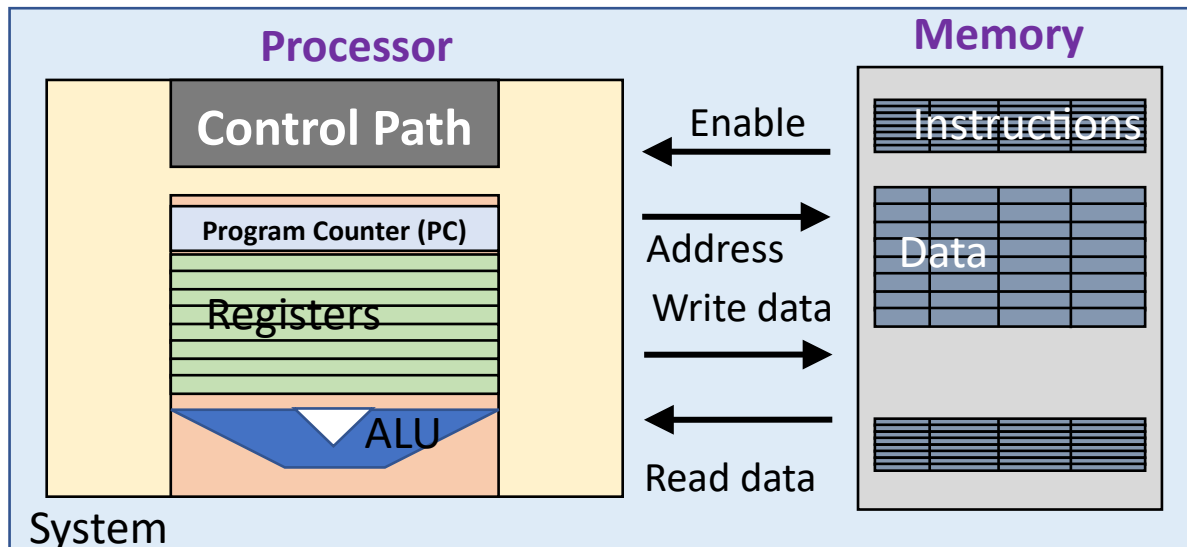
- Real systems are too complex to be viewed in entirety
  - Hierarchy splits system into smaller subsystems so developers can focus on their subsystem
  - Allows reuse, not in depth understanding needed

# Model Characteristics

- **Hierarchy:** real systems are too complex to be viewed in entirety
  - hierarchy splits system into smaller subsystems so developers can focus on their subsystem (allows reuse, not in depth understanding needed)

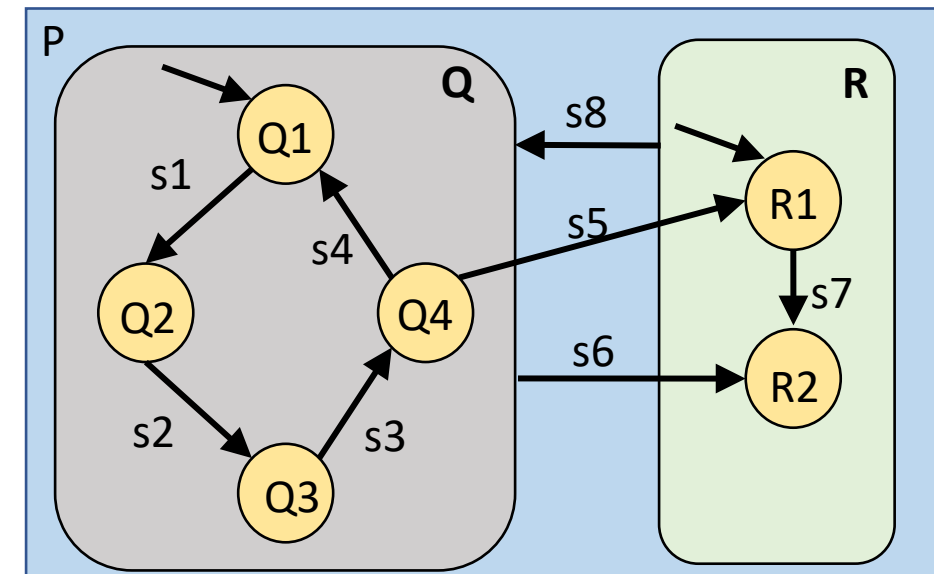
## Structural hierarchy:

Every component is made up of a sub-structure to lower level of abstraction



## Behavioral/ Functional hierarchy

Divides functions into sequential or concurrent sub-functions



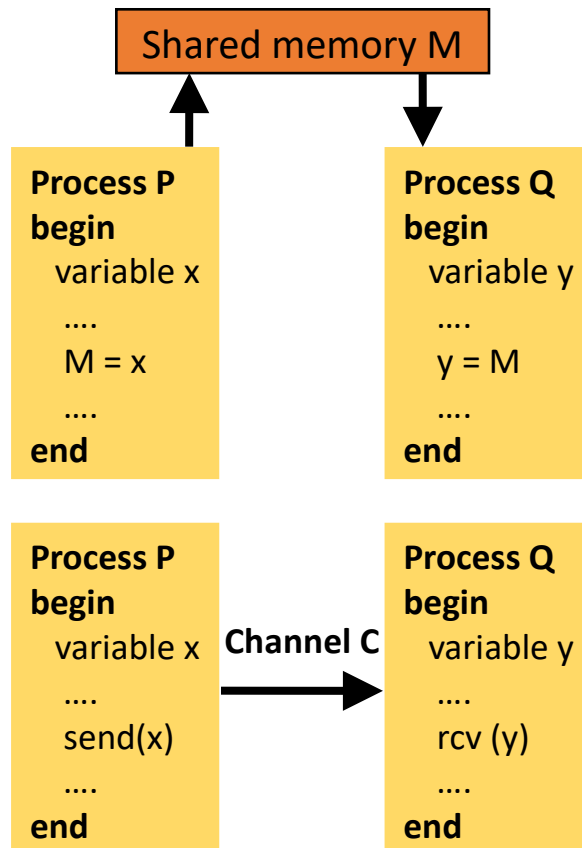
# Model Characteristics ...

- Program Structures/Constructs
  - Many functions can be described best by sequential algorithms including branches, iterations, subroutines, e.g., sorting algorithm
- Completion
  - Process ability to indicate it has stopped
  - All calculations are made or all variables got assigned their new value
  - Sequential processing
    - ensures that the next step in an algorithm does not begin until the current step is complete
    - coordinate processes by ensuring that dependencies are resolved
- Communication
  - Connect HW/SW subsystems



# Model Characteristics ...

- Communication – Connect HW/SW subsystems



## Shared memory

Sending process writes global variable into shared resource; all receiving processes can now read var; sync must be done separate

## Message parsing

- Data between processes is exchanged through communication channels (uni/bidirectional, point-to-point, shared bus)
- Channel can be blocking on non-blocking transfer
  - Blocking-trans: sending process waits until receiving process has accepted data
  - Non-blocking-trans: sending process writes data in queue and continues processing.
    - Receiver can read it at its leisure => standard today, additional memory for queue needed

# Memory-Mapped Input/Output (I/O)

- peripheral devices are mapped into the same address space as the program memory.
- In memory-mapped I/O, reads and writes to specific memory addresses are translated into operations on the peripherals
- What operations does software need to perform on peripherals?
  - Get and set parameters
  - Receive and transmit data
  - Enable and disable functions
- How can we imagine providing this interface to software?
  - Specialized CPU instructions
  - Accessing devices like they are memory

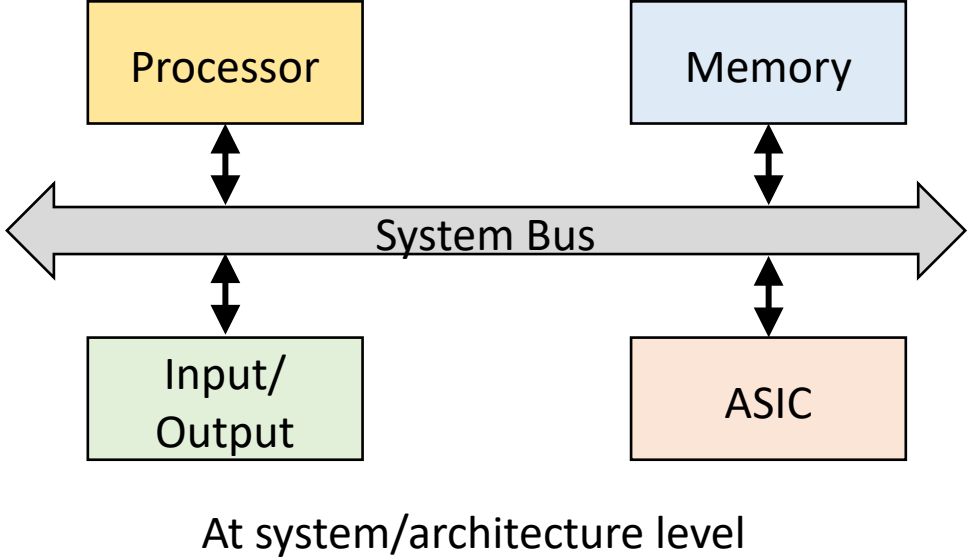
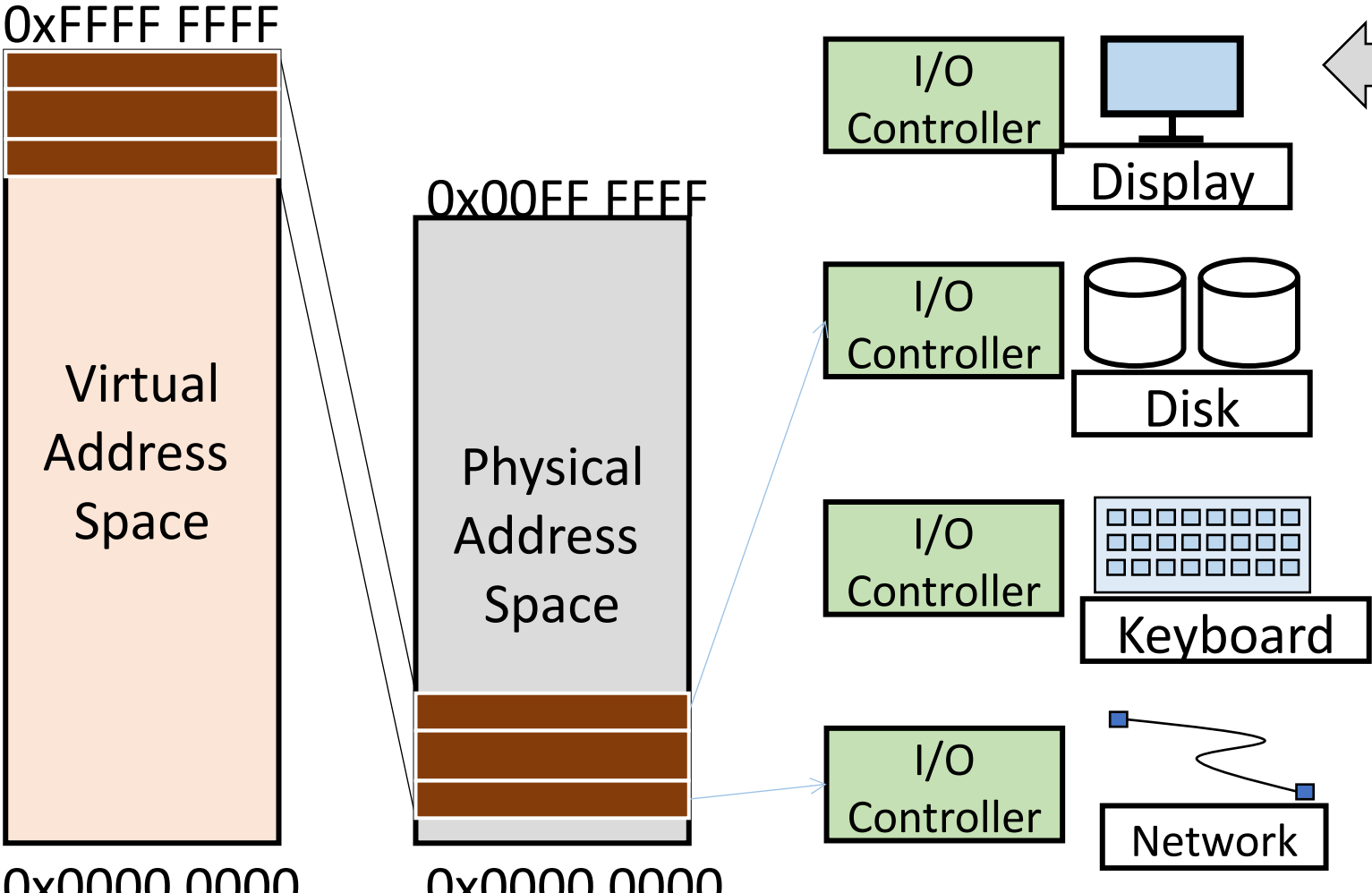
# Memory-Mapped Input/Output (I/O)

- Access I/O devices (like keyboards, monitors, printers) just like it accesses memory
  - Each I/O device assigned one or more address
  - When that address is detected, data is read from or written to I/O device instead of memory
  - A portion of the address space dedicated to I/O devices (for example, addresses 0xFFFF0000 to 0xFFFFFFFF in reserved segment of memory map)
- But we need additional hardware to help us
  - After all we will not really write to and read from memory

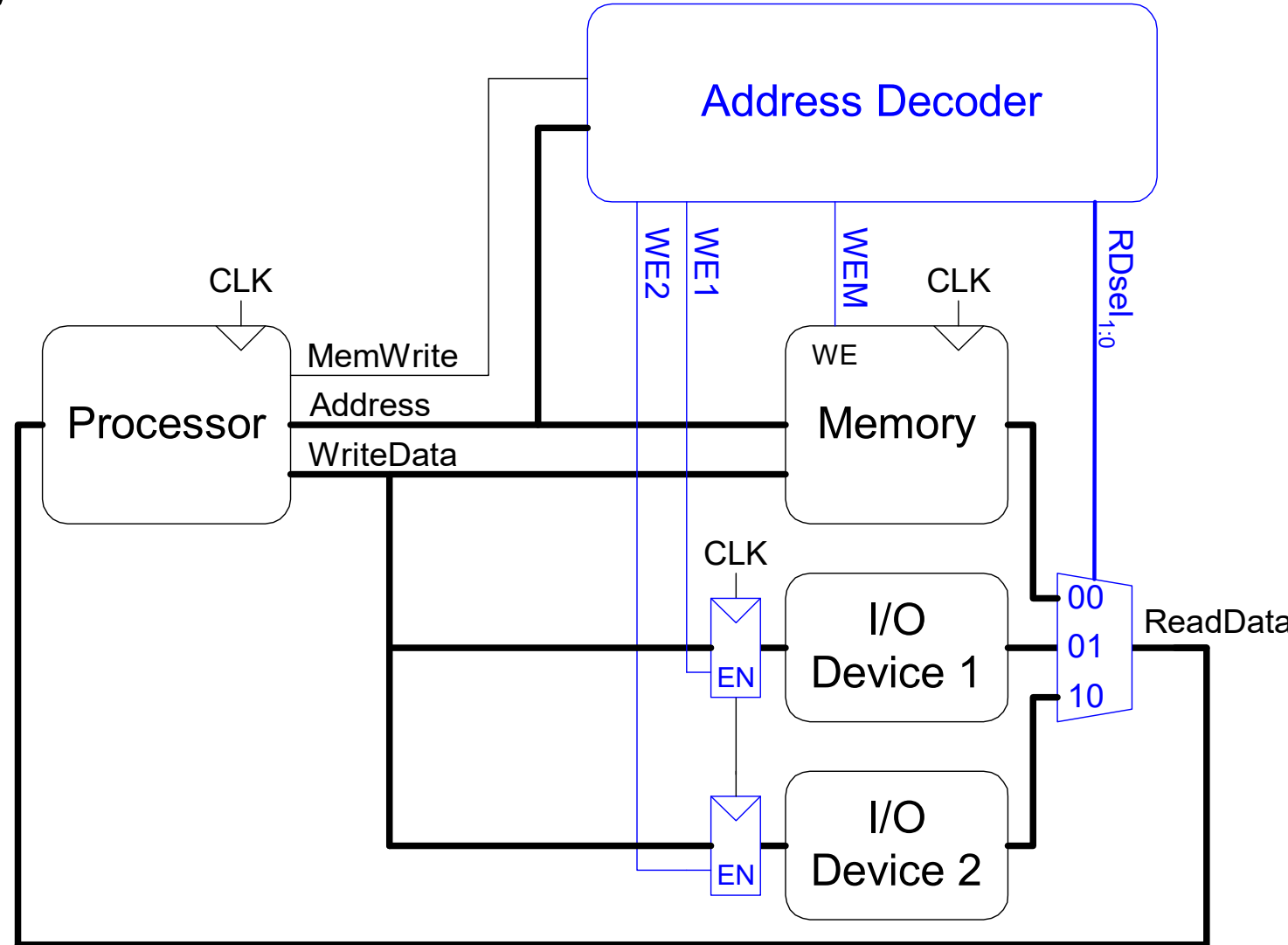
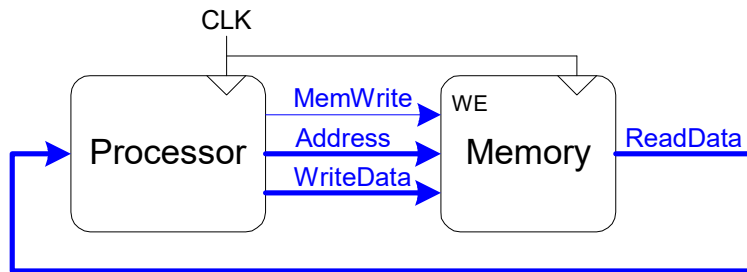
# Memory-Mapped I/O Hardware

- Address Decoder:
  - Looks at address to determine which device/memory communicates with the processor
- I/O Registers:
  - Hold values written to the I/O devices
- ReadData Multiplexer:
  - Selects between memory and I/O devices as source of data sent to the processor

# Memory-Mapped I/O



# Memory-Mapped I/O Hardware



# Memory-Mapped I/O Code

- Suppose I/O Device 1 is assigned the address 0xFFFFFFFF4
  - Write the value 42 to I/O Device 1
  - Read the value from I/O Device 1 and place it in \$t3

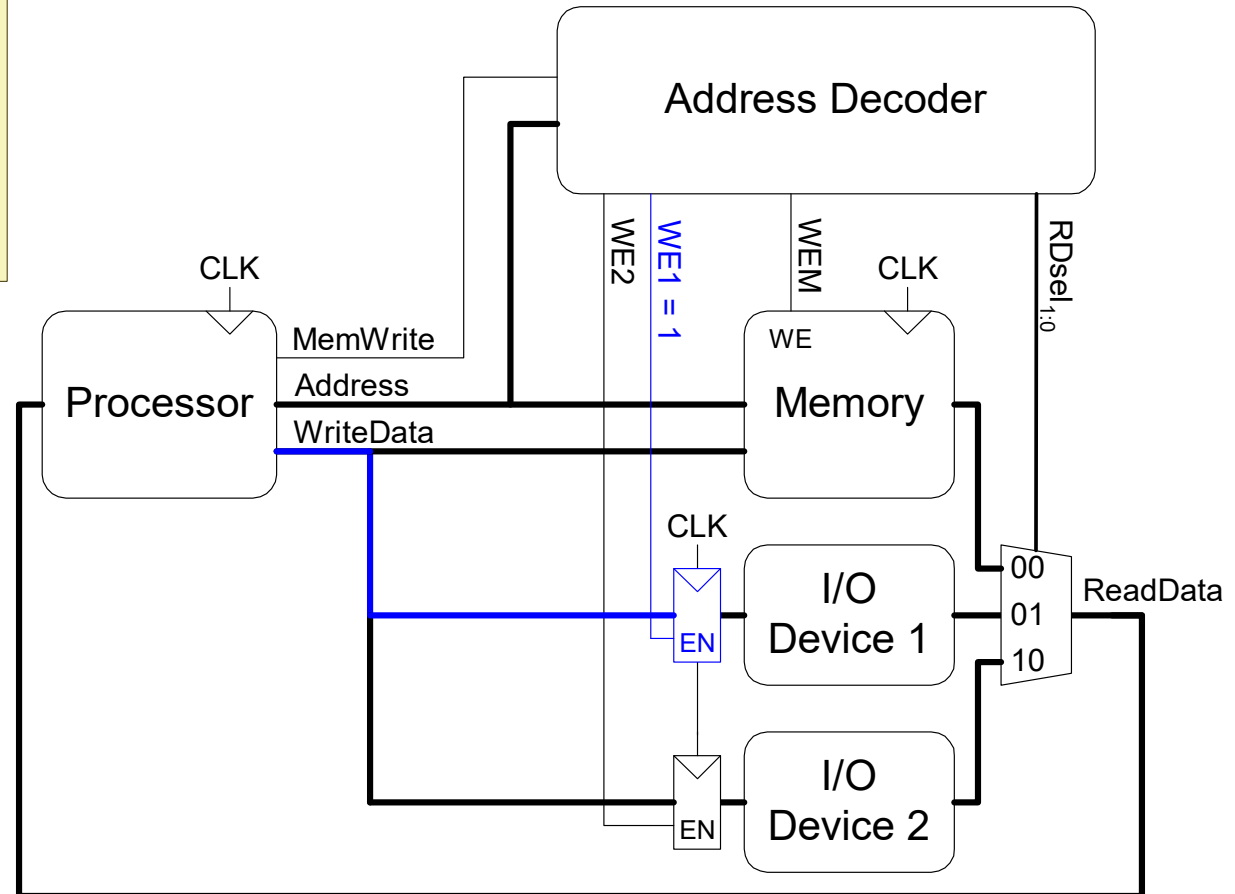
# Memory-Mapped I/O Code: Write

*Write 42 to I/O Device 1 (0xFFFFFFFF4)*

```
addi $t0, $0, 42
sw    $t0, 0xFFF4($0)
```

# Recall that the 16-bit immediate

# is sign-extended to 0xFFFFFFFF4





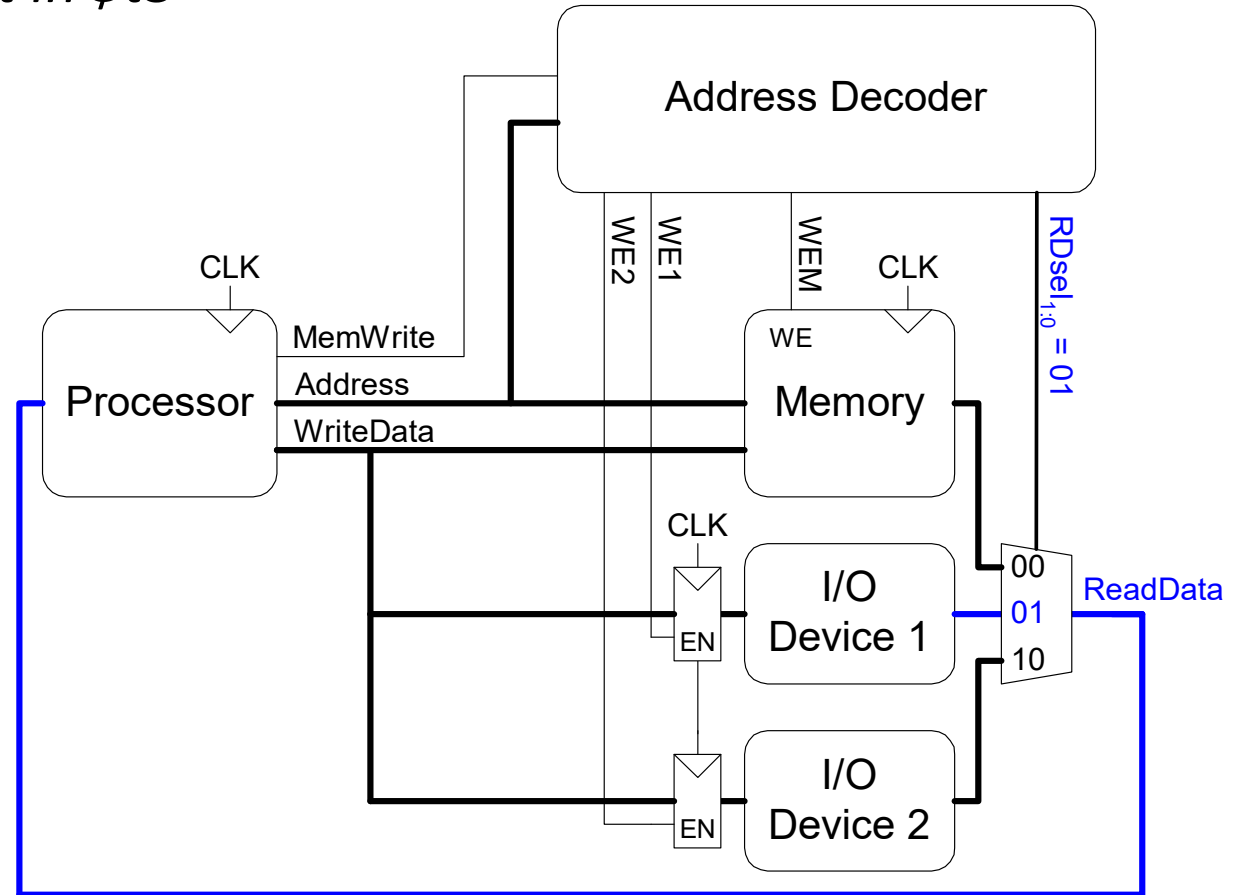
# Memory-Mapped I/O Code: Read

*Read from I/O Device 1 and place it in \$t3*

```
lw $t3, 0xFFF4($0)
```

# Recall that the 16-bit  
immediate

# is sign-extended to 0xFFFFFFFF4

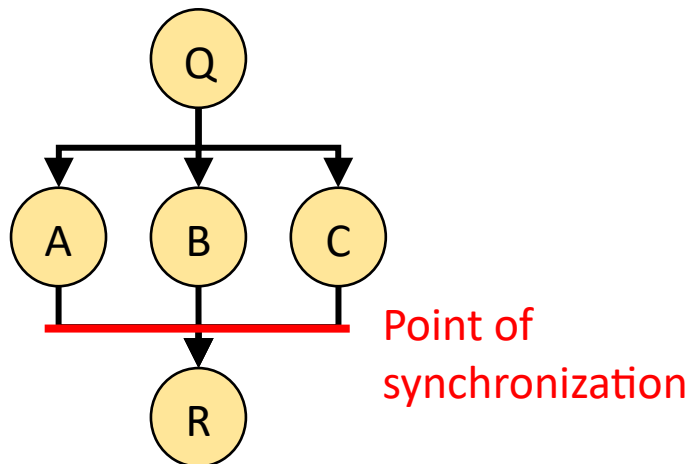


# Model Characteristics ...

- Synchronization
  - Concurrent processes are never fully independent from each other.
  - Sync to exchange data
  - Connect HW/SW subsystems

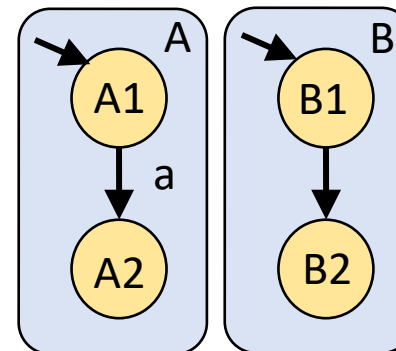
## Control oriented sync

Control structure of functions determine sync



## Data oriented sync

Sync by using inter-process communication  
(shared memory, message passing)



# Model Characteristics ...

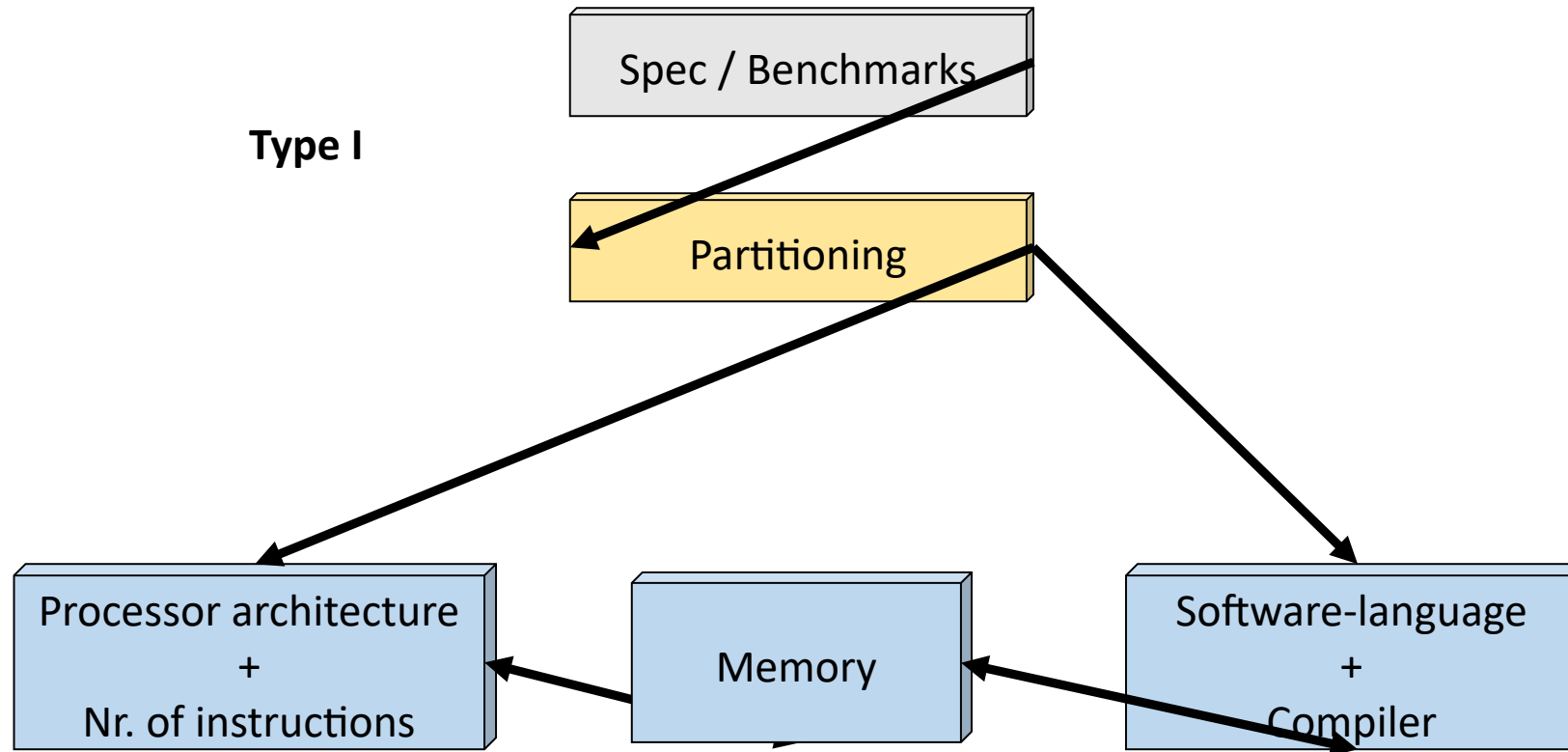
- Exception Handling
  - Events like a reset or interrupt can abruptly terminate a process.
  - If such event/exception occurs control is passed to a pre-defined exception handling routine.
- Non-Determinism
  - Allows specification of multiple options due to unclear best suiting operations for app
    - Put off final decision for later in design process

# HW/SW Codesign Synthesis

# Agenda

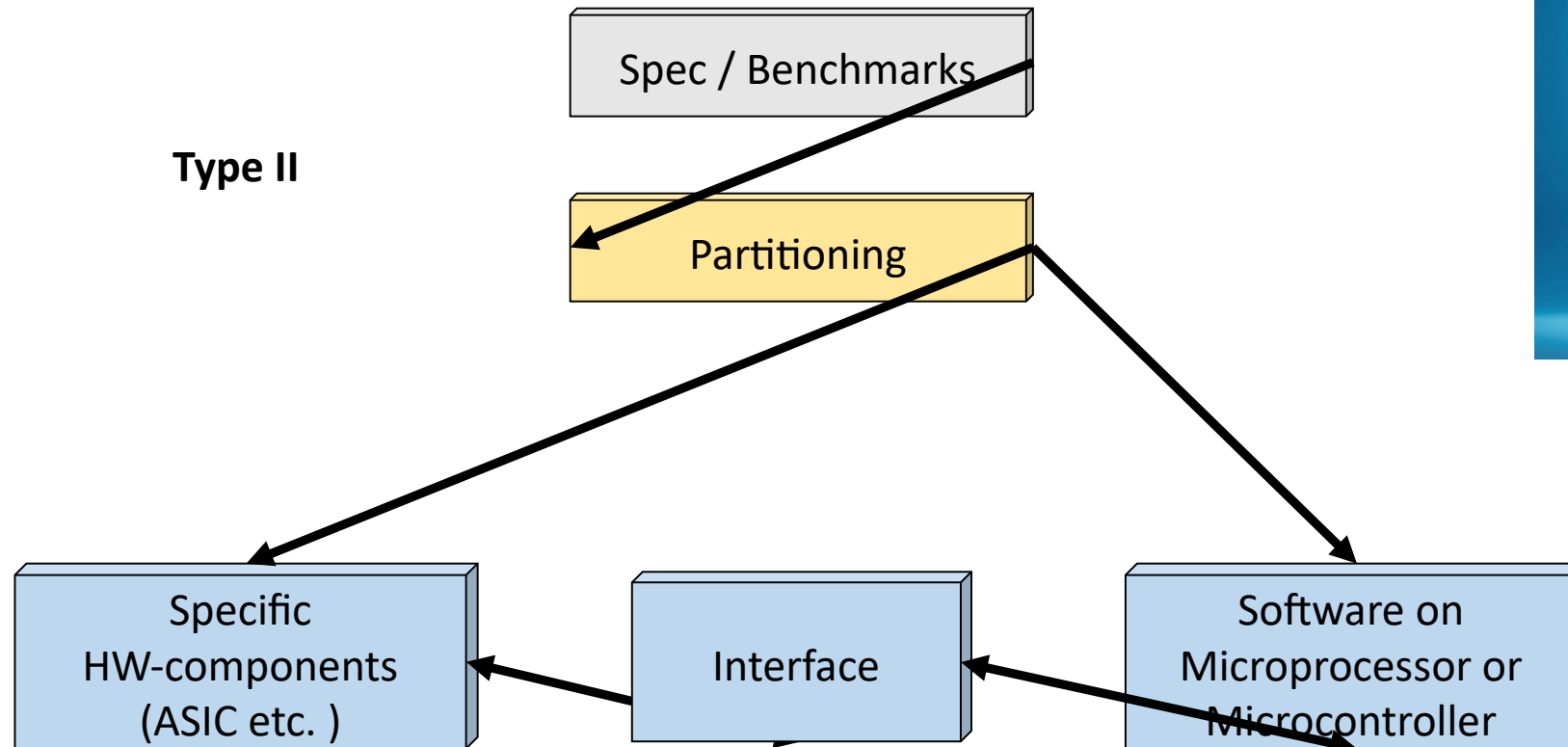
- System Partitioning
  - Partitioning Costs
  - Partitioning Styles
  - Partitioning through ILP Solving
- Scheduling
  - Scheduling without Resource Constraints
  - Scheduling with Resource Constraints
- Resource Sharing and Binding

# Types of Mixed HW/SW Systems (1)



**Type I:** logical borders between HW/SW  
-> SW is developed on the “in parallel implemented HW”  
-> optimized hardware-compiler interface

# Types of Mixed HW/SW Systems (2)

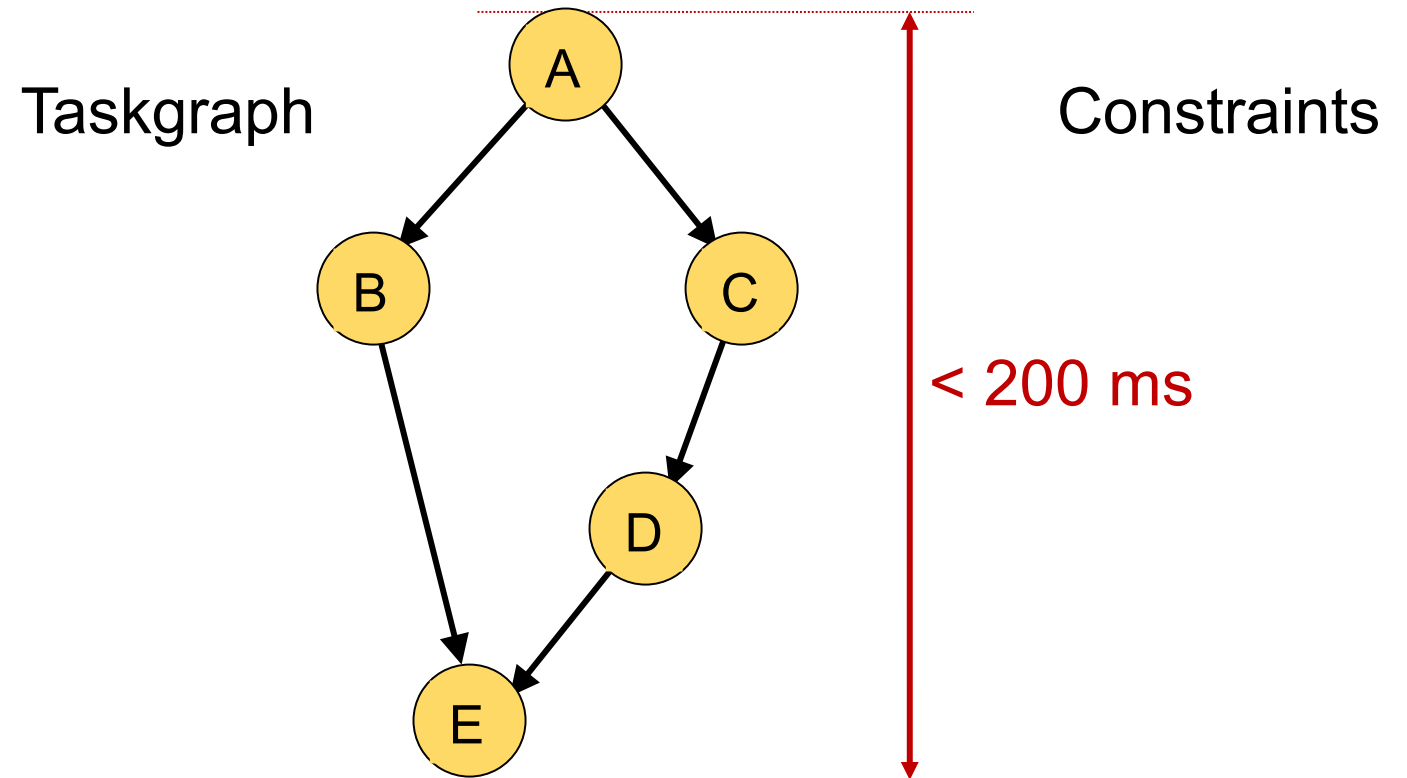


**Type II:** physical borders

- > programmable components and hardware modules
- > to achieve efficient functionality by partitioning cooperating HW/SW-components + interfaces.

# Specification at System-level

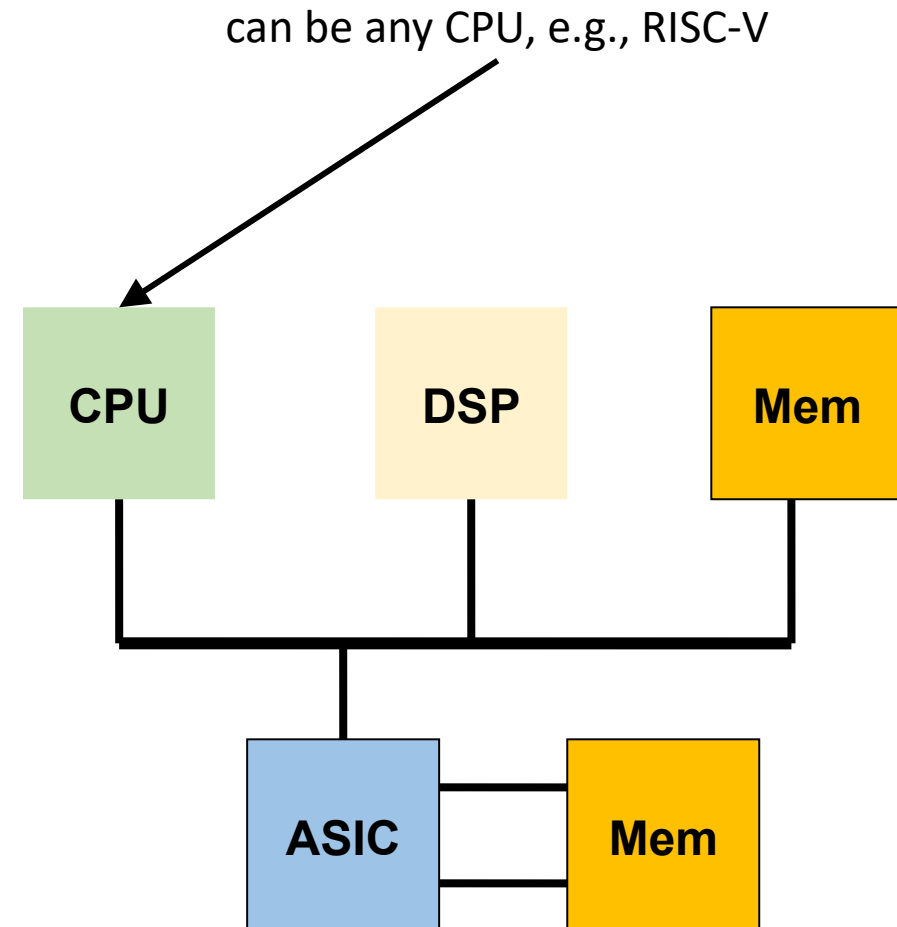
- System-level specification describes the overall behavior and functionality
  - Inputs
  - Outputs
  - Constraints
  - Interfaces
  - protocols
- System-level specification is typically created at the early stages



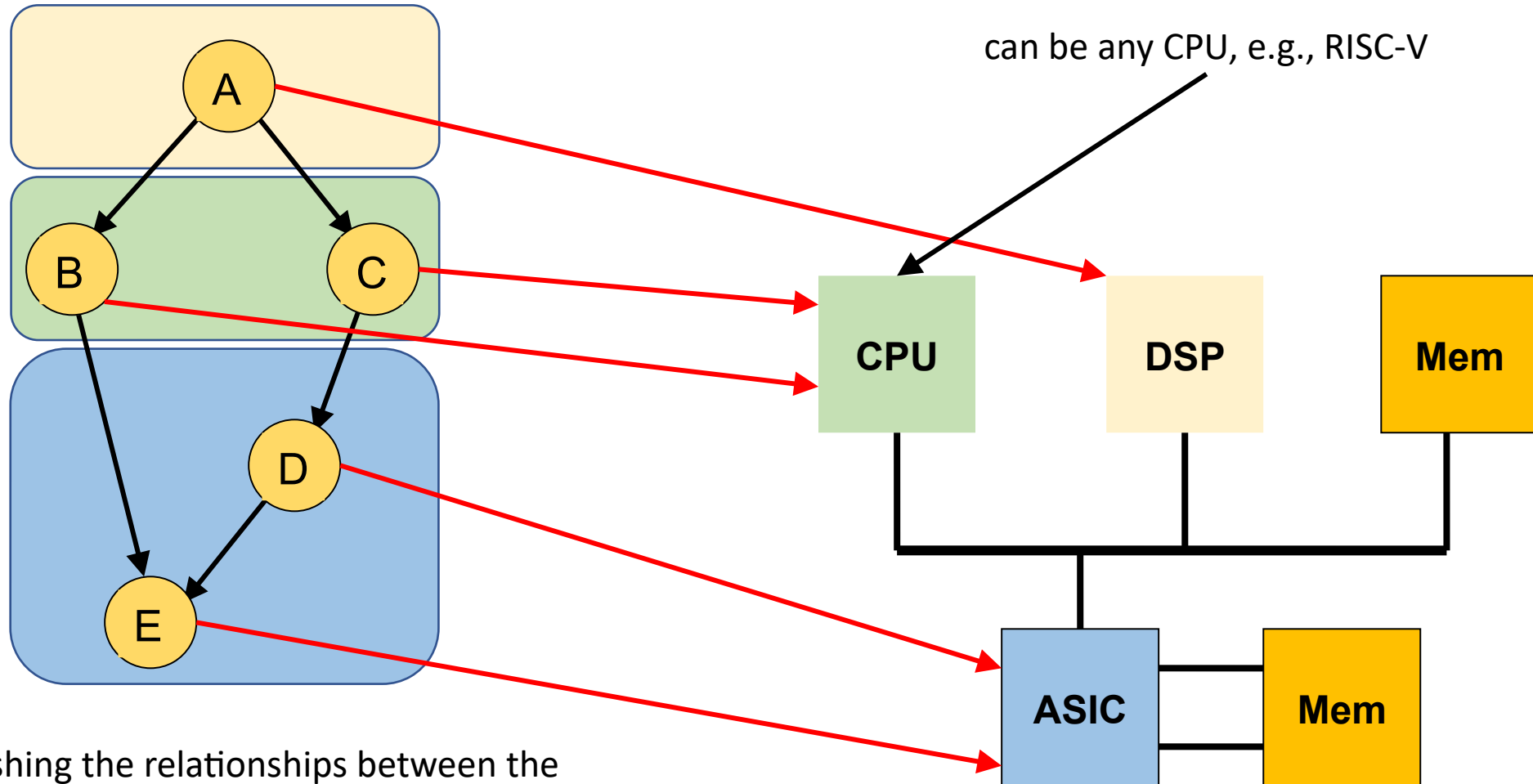


# Allocation at System-level

- Process of dividing the functionality and behavior
  - Processors, dedicated hardware devices
  - memory, I/O
  - Interconnect structures
- Performance requirements
- Power constraints
- Cost
- Availability of hardware components and software libraries
- Iterative process

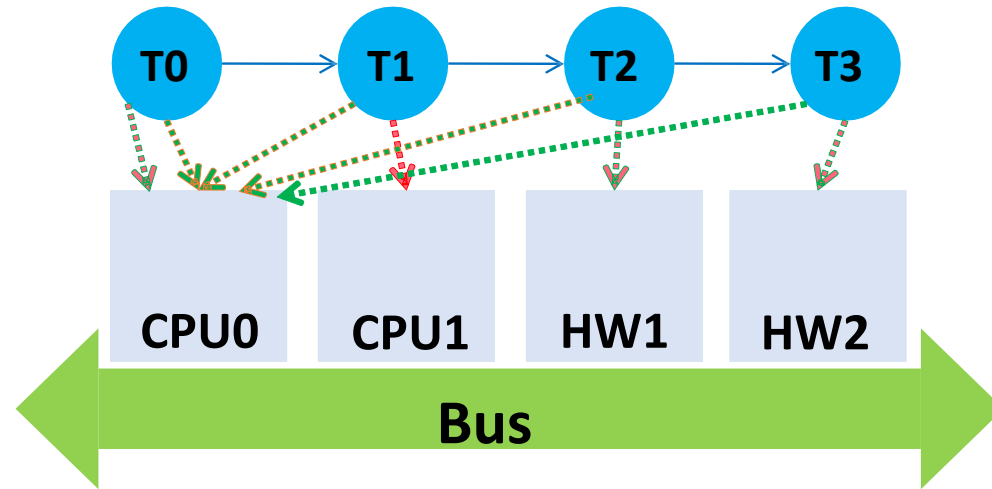


# Binding on System-level



- Process of establishing the relationships between the hardware and software
  - Static binding
  - Dynamic binding

# Partitioning – Problem Description



*optimal V*: N:1 mapping

*optimal L*: 1:1 mapping



In reality it is not like this!

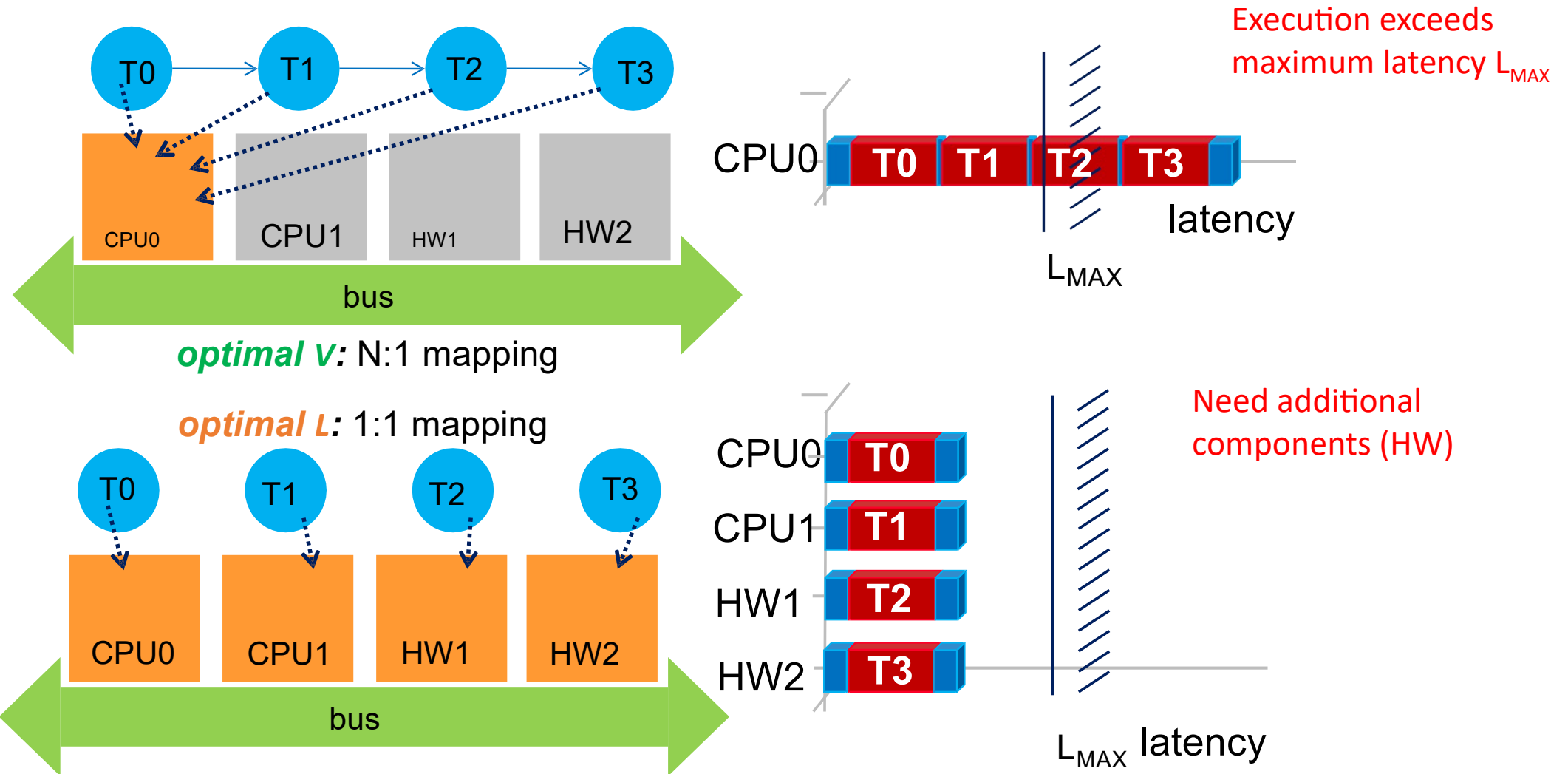
- Considered **performance**

- **Value V**: due to number of **allocated components** (In terms of costs in Euros)
- **Latency L**: due to scheduling (resource sharing) [time in seconds]

- **Conflicting design goals**

- Feasible schedule  $L \leq L_{max}$
- Feasible allocation  $V \leq V_{max}$

# Partitioning – Problem Description



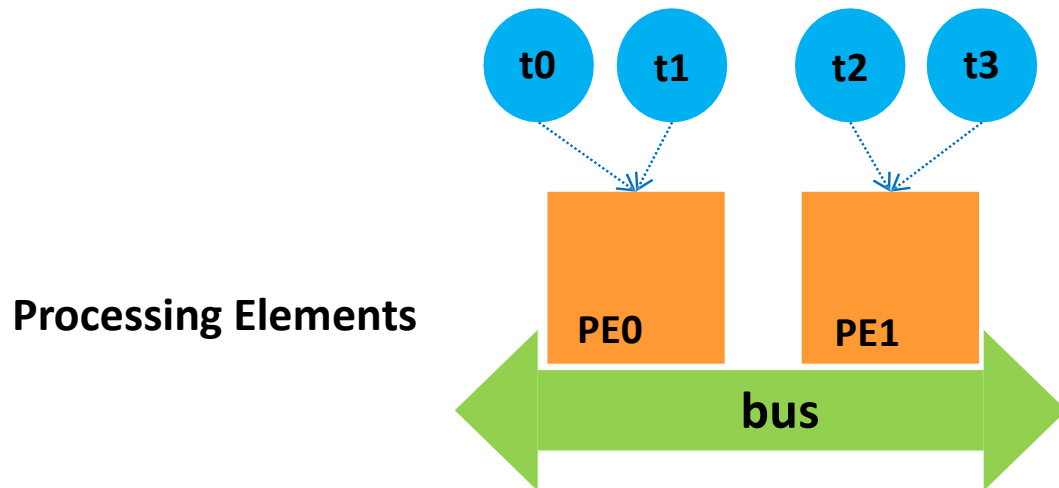
# Partitioning- Cost Function

- **Quantitatively measure performance** of a design point
  - System **value** V [€]
  - **Latency** L [sec]
  - **Energy** consumption E [Wh]
  - ...
- Estimation required to find V, L, E values for each design point
  - E.g.: **linear cost** C function with penalty

$$C = f(V, L, E) = k_1 \cdot h_V(V, V_{max}) + k_2 \cdot h_L(L, L_{max}) + k_3 \cdot h_E(E, E_{max})$$

- Vmax, Lmax, Emax: design **constraints**
- $h_V, h_L, h_E$ : denote how **strong** V, L, E **violate** design constraints
- $k_1, k_2, k_3$ : **weighting** and normalization

# Example 1 – Load Balanced System



Time a PE will take to execute a task

Exec. Time	t0	t1	t2	t3
PE0	5	15	10	30
PE1	10	20	10	10



Optimized for a load balanced system:

task \ PE	t0	t1	t2	t3
PE0	1	1	0	0
PE1	0	0	1	1



*load balancing:*

$$\text{load}_{\text{PE0}} = 5 + 15$$

$$\text{load}_{\text{PE1}} = 10 + 10$$

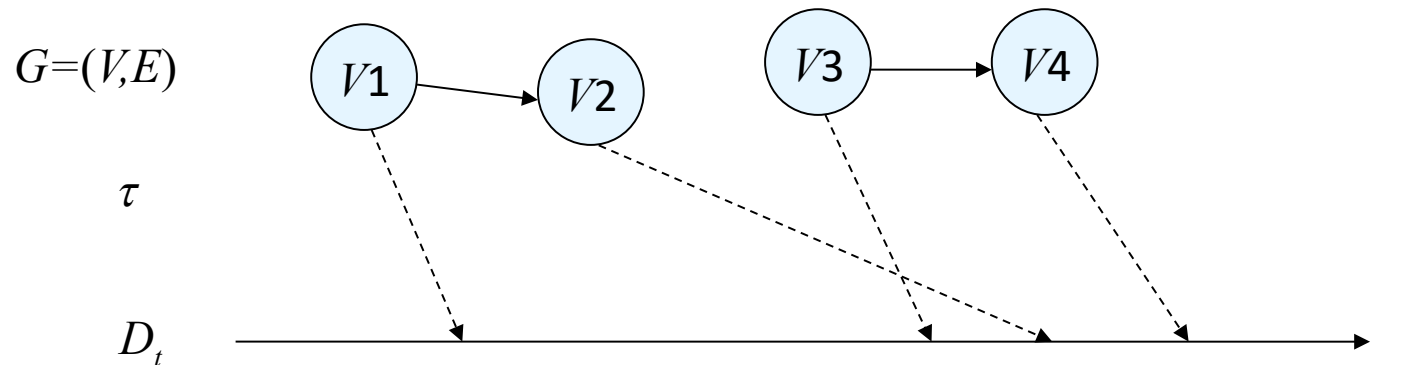
The goal is to ensure that each processing element has an approximately equal share of the workload over time to maximize efficiency and minimize the time taken for all tasks to be completed.

# Scheduling

- Similarities with scheduling and load balancing in real-time operating systems (RTOS):
  - Time constraints, context switching and context switching overhead, process synchronization and communication.
- Differences to sequence planning in RTOS:
  - Larger design space with very different solutions
  - High optimization requirements (motivation for hardware design)
  - Underlying hardware is variable

# Real-time Scheduling

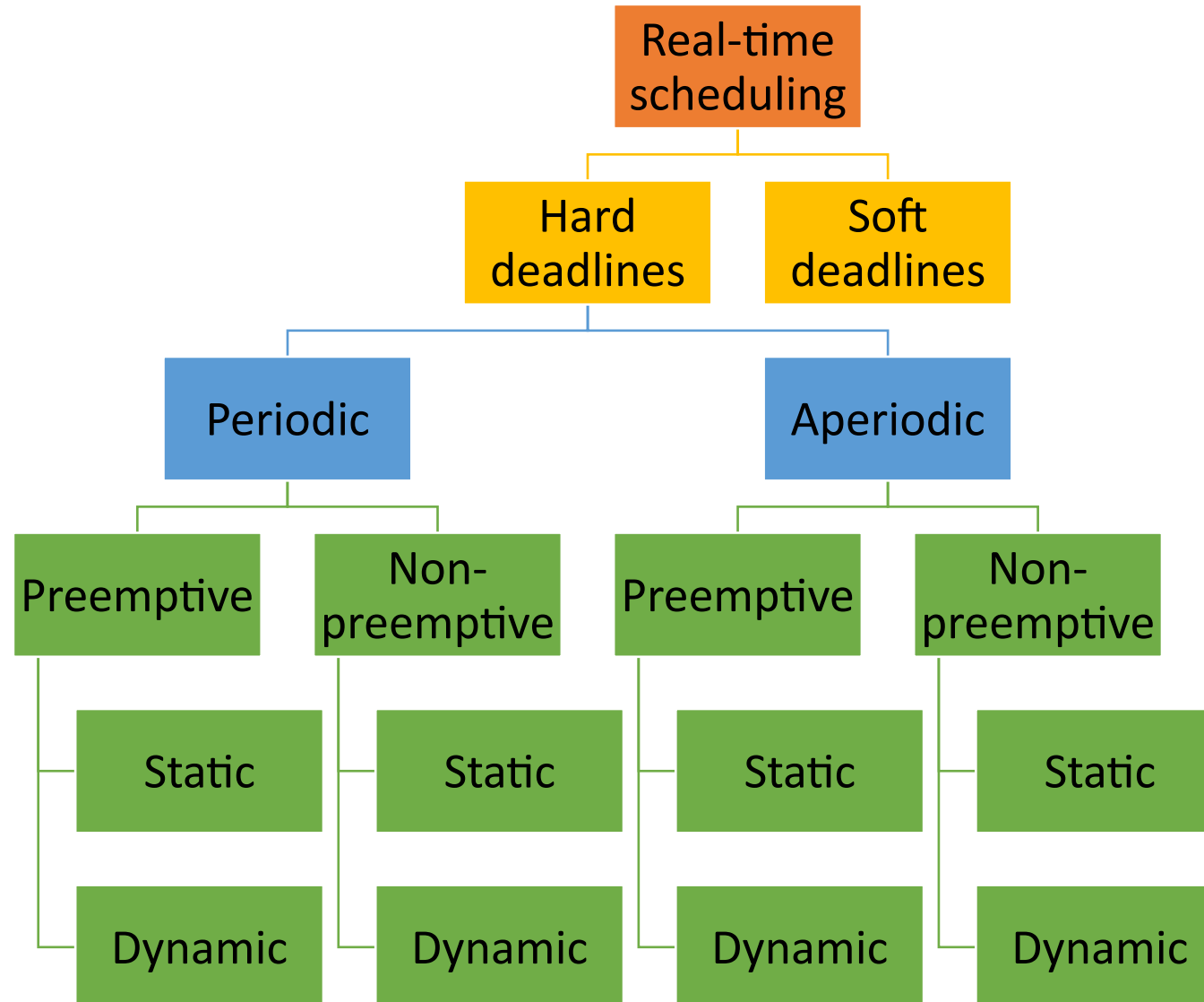
- Given a Task-Graph  $G=(V,E)$ .
- Def.: A schedule  $\tau$  from  $G$  is the Allocation  $V \rightarrow D_t$   
a set of tasks  $V$  at times of the Domain  $D_t$



Various constraints that schedules must take into account: e.g., resource constraints, dependency constraints, deadlines  
Schedule = finding such an assignment

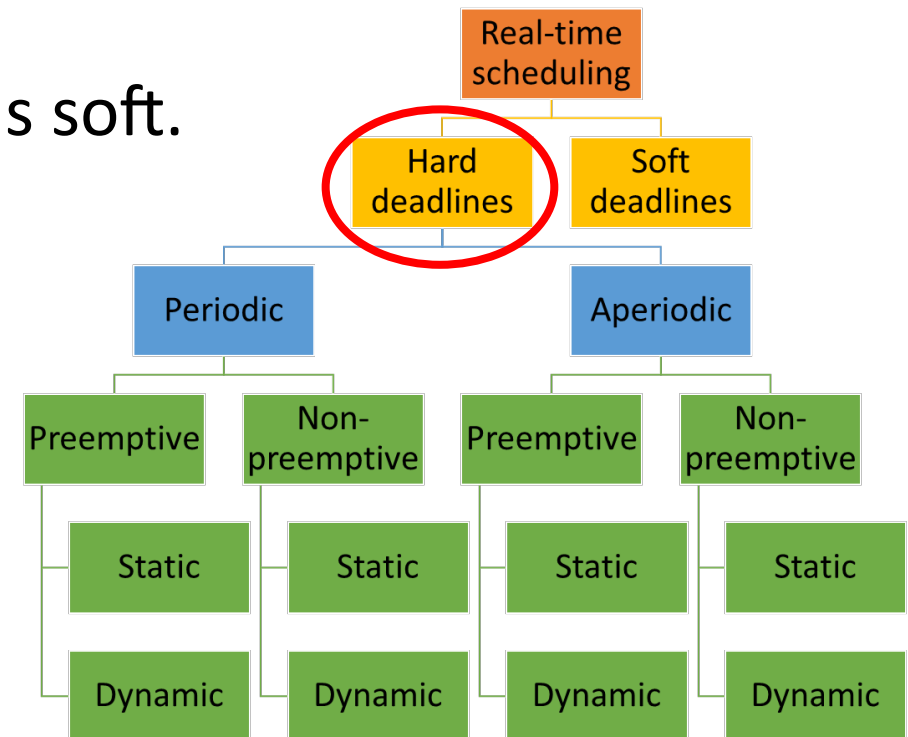


# Real-time Scheduling Strategies



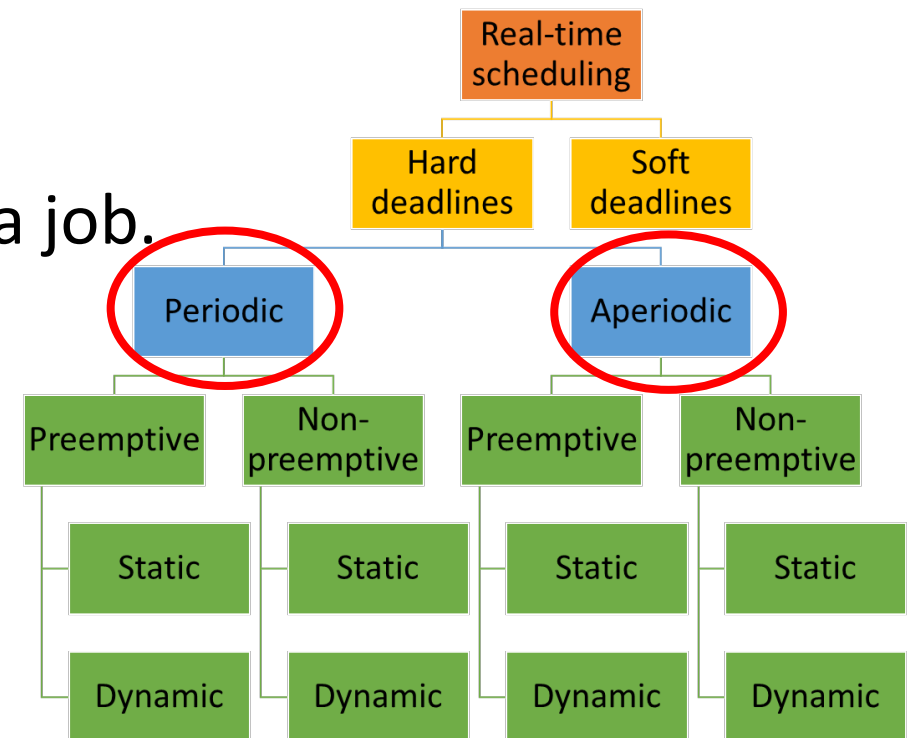
# Hard and Soft Deadlines

- A time constraint (deadline) is called hard if the failure to meet it could lead to a disaster.
- All other time constraints are referred to as soft.
- Focus mostly on hard deadlines



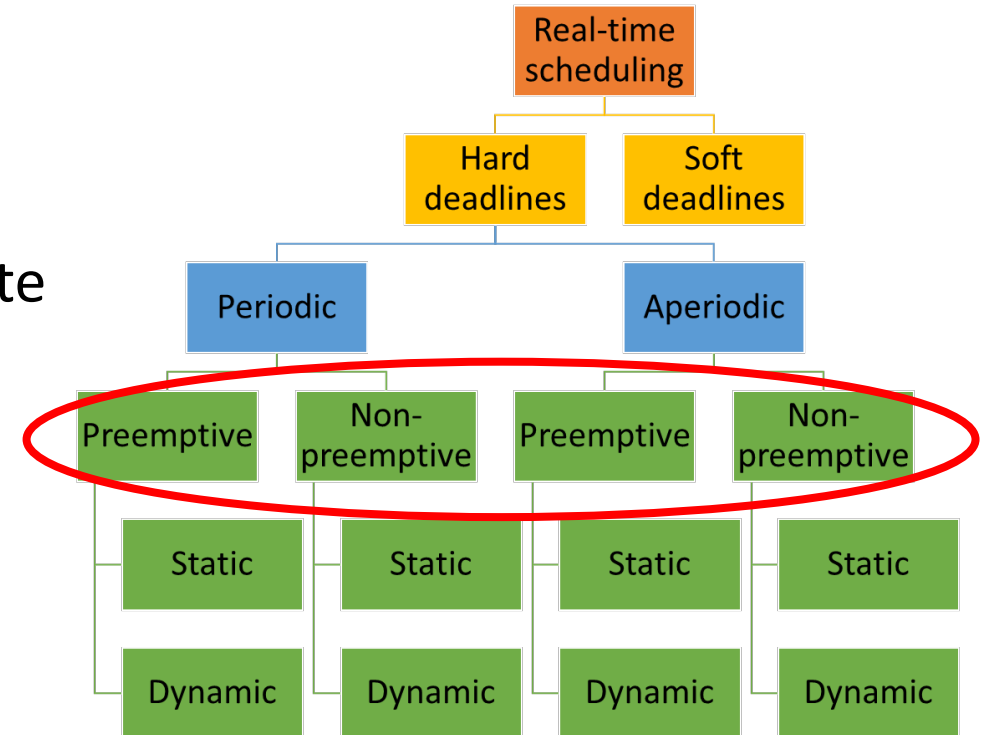
# Periodic and Aperiodic Tasks

- Tasks that must be executed once every  $p$  time units are called periodic tasks.
- $p$  is referred to as their period.
- Each execution of a periodic task is called a job.
- All other tasks are called aperiodic.



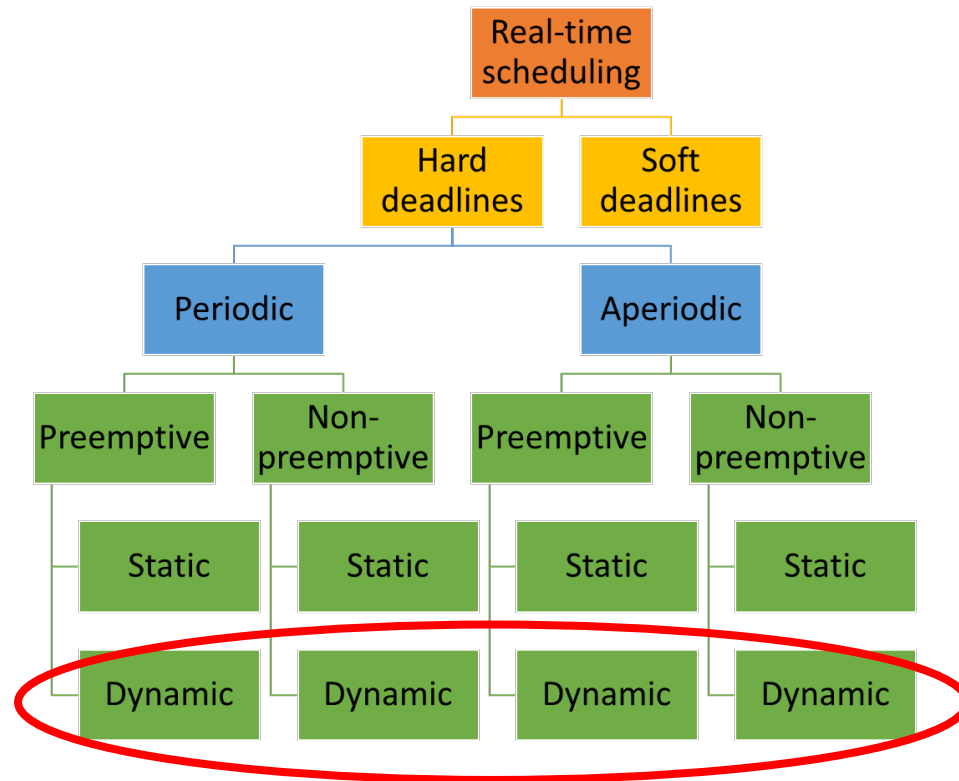
# Preemptive and Non-preemptive Scheduling

- Non-preemptive scheduler:
  - Tasks are executed until they are completed
  - Response time for external events can be quite long
- Pre-emptive schedulers:
  - Tasks can be terminated before they are completed.
    - When some tasks have long execution times
    - When short response time for external events is required



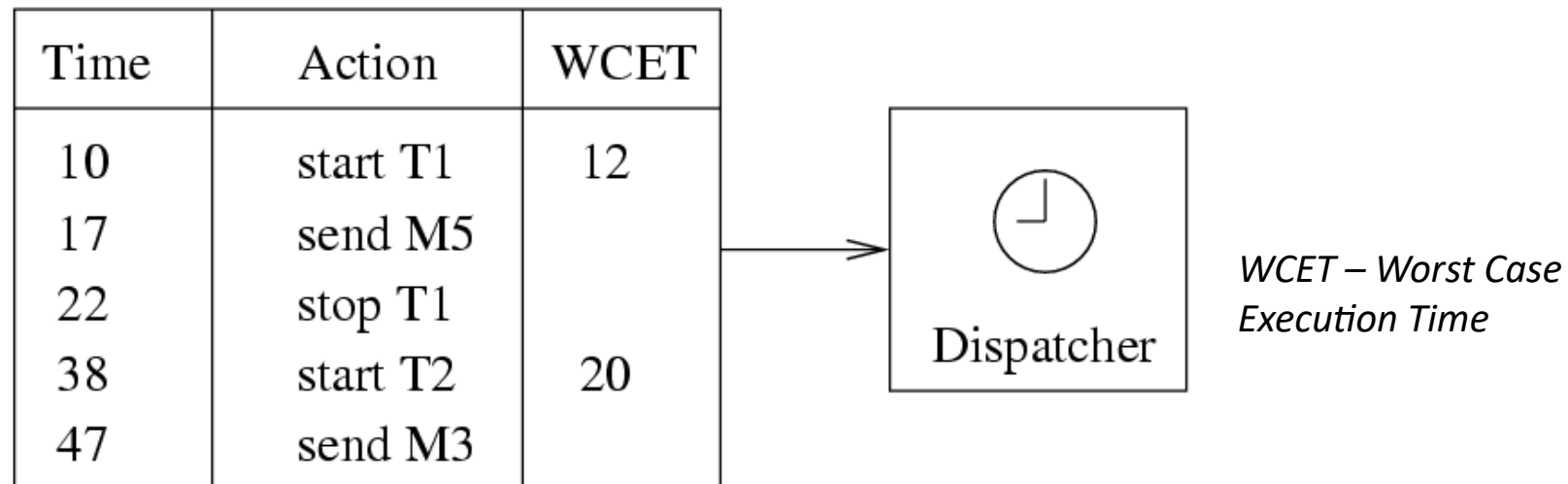
# Dynamic/online Scheduling

- Processor/hardware allocation decisions (scheduling) at runtime.
- Based on information about the tasks that have arrived so far



# Static/offline Scheduling

- Scheduling takes into account a priori knowledge about arrival times, execution times, deadlines
- Dispatcher assigns processor/hardware when interrupted by timer
- Timer is controlled by a table generated at design time

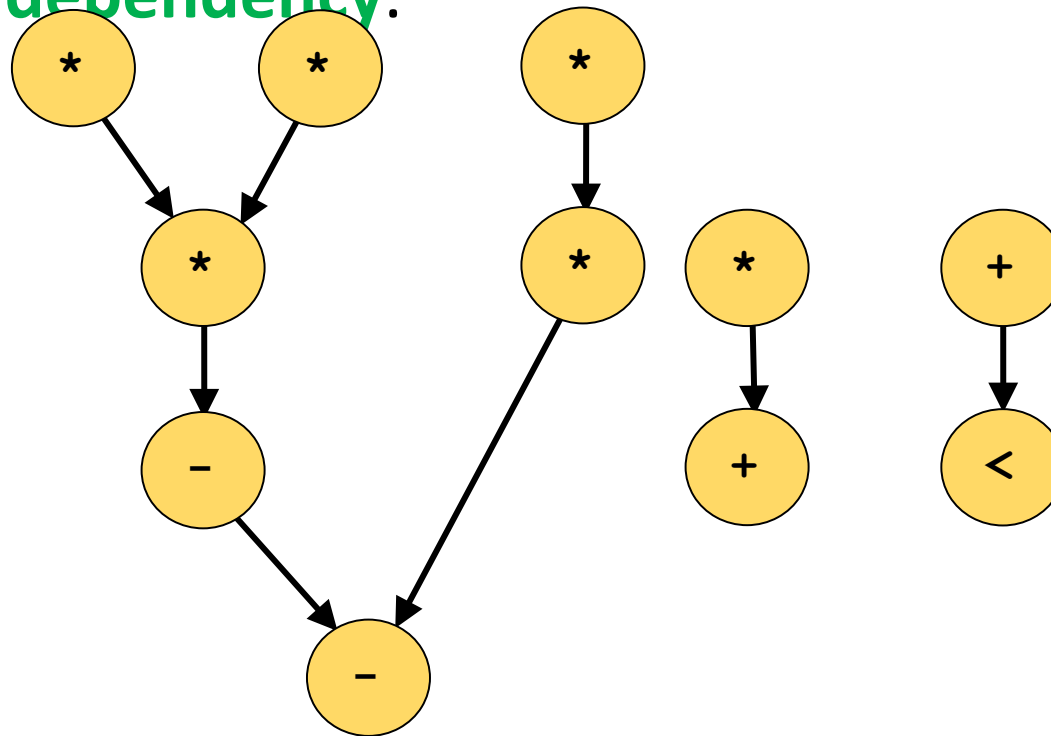


# Without resource constraints

- ASAP (as soon as possible)
  - Determines the earliest possible start times for the tasks
  - Delivers minimum latency
- ALAP (as late as possible)
  - Determines the latest possible start times for the tasks with a given latency
- Mobility of a task is the difference of the start times ALAP-ASAP.
  - Mobility 0  $\rightarrow$  Task is on the critical path

# Problem Graph

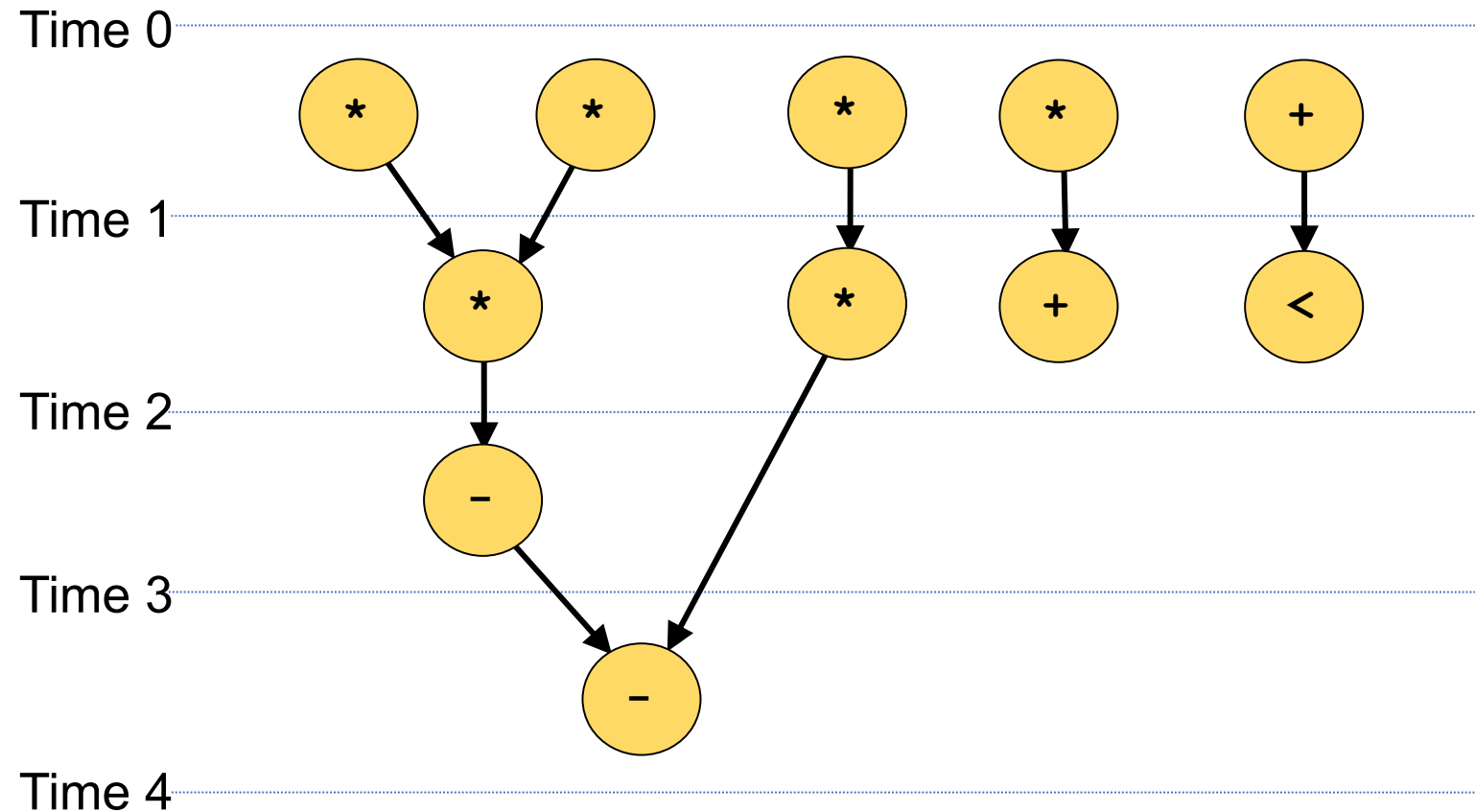
- A problem graph  $G(V,E)$  is a directed acyclic graph with node set  $V$  and edge set  $E$  in which each **node  $v_i \in V$  represents a task** (task, process, instruction, elementary operation) and **each edge  $e=(v_i, v_j) \in E$  represents a data dependency**.





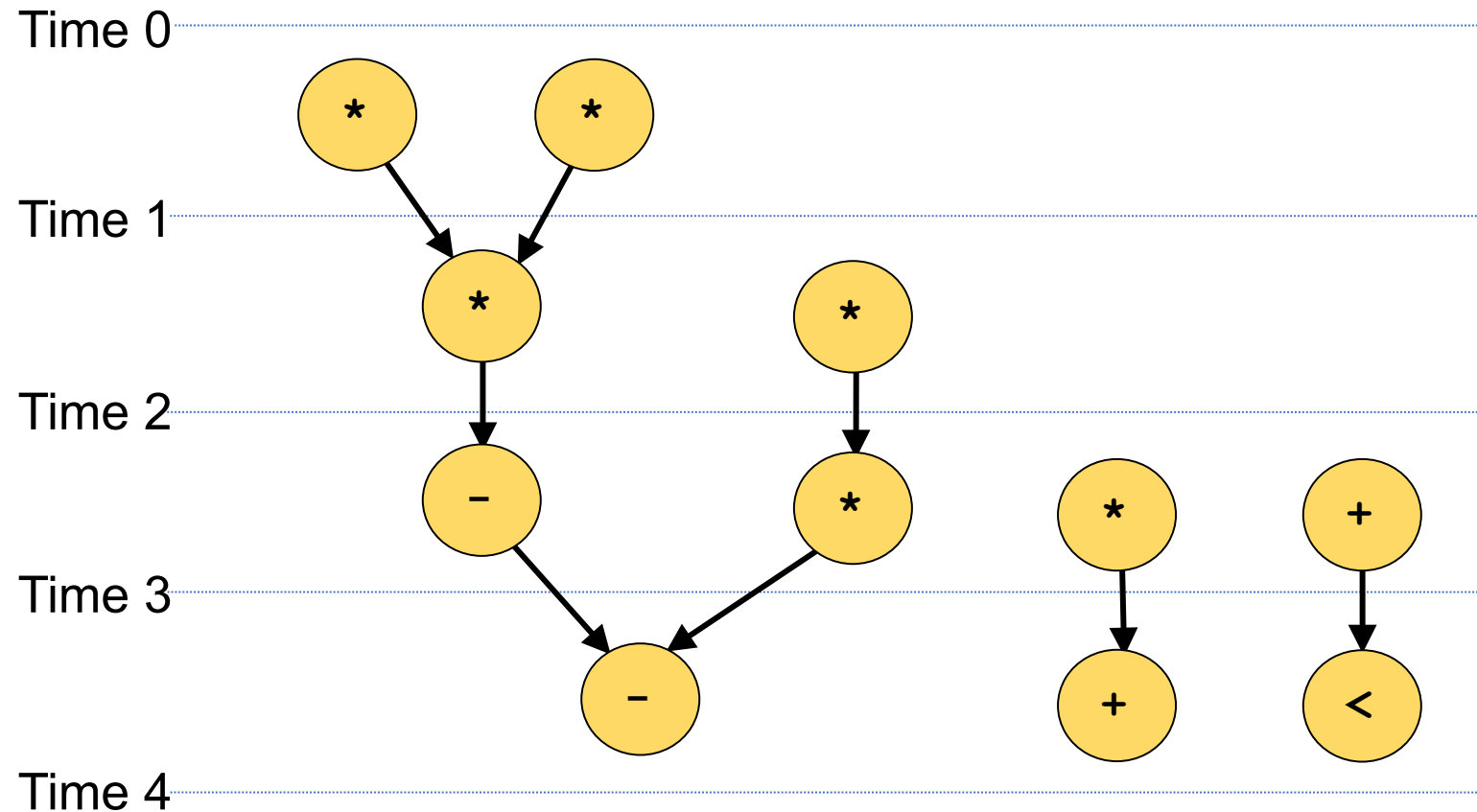
# ASAP (as soon as possible)

- Latency optimal schedule:  $L = 4$



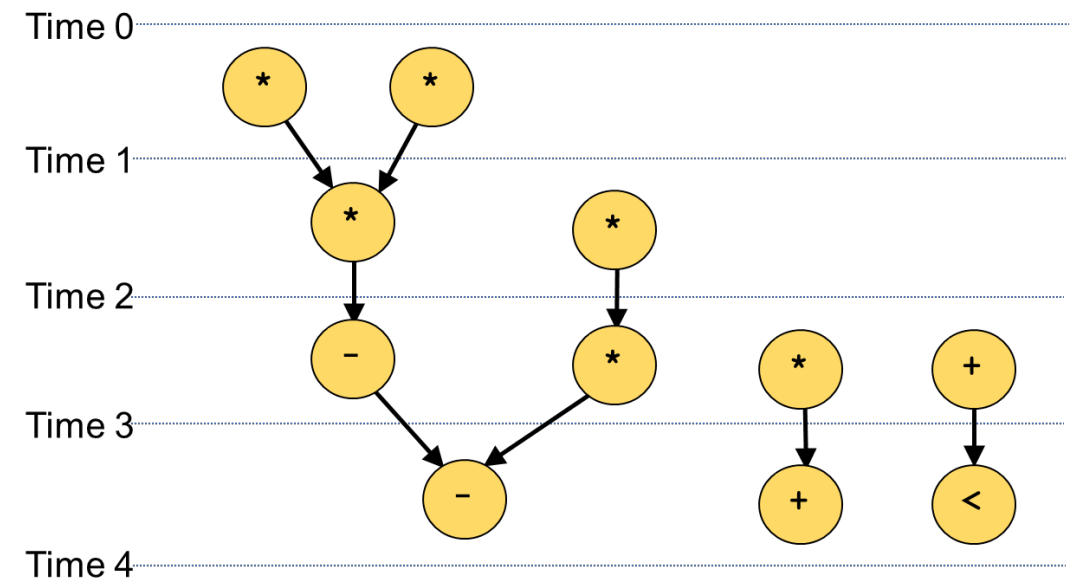
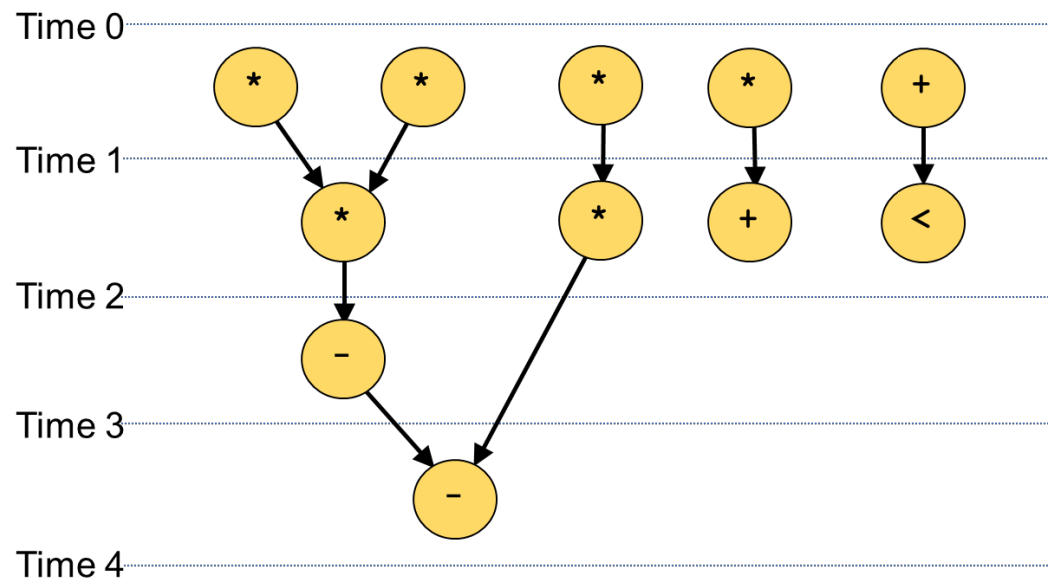
# ALAP (as late as possible)

- Latency limit:  $L = 4$

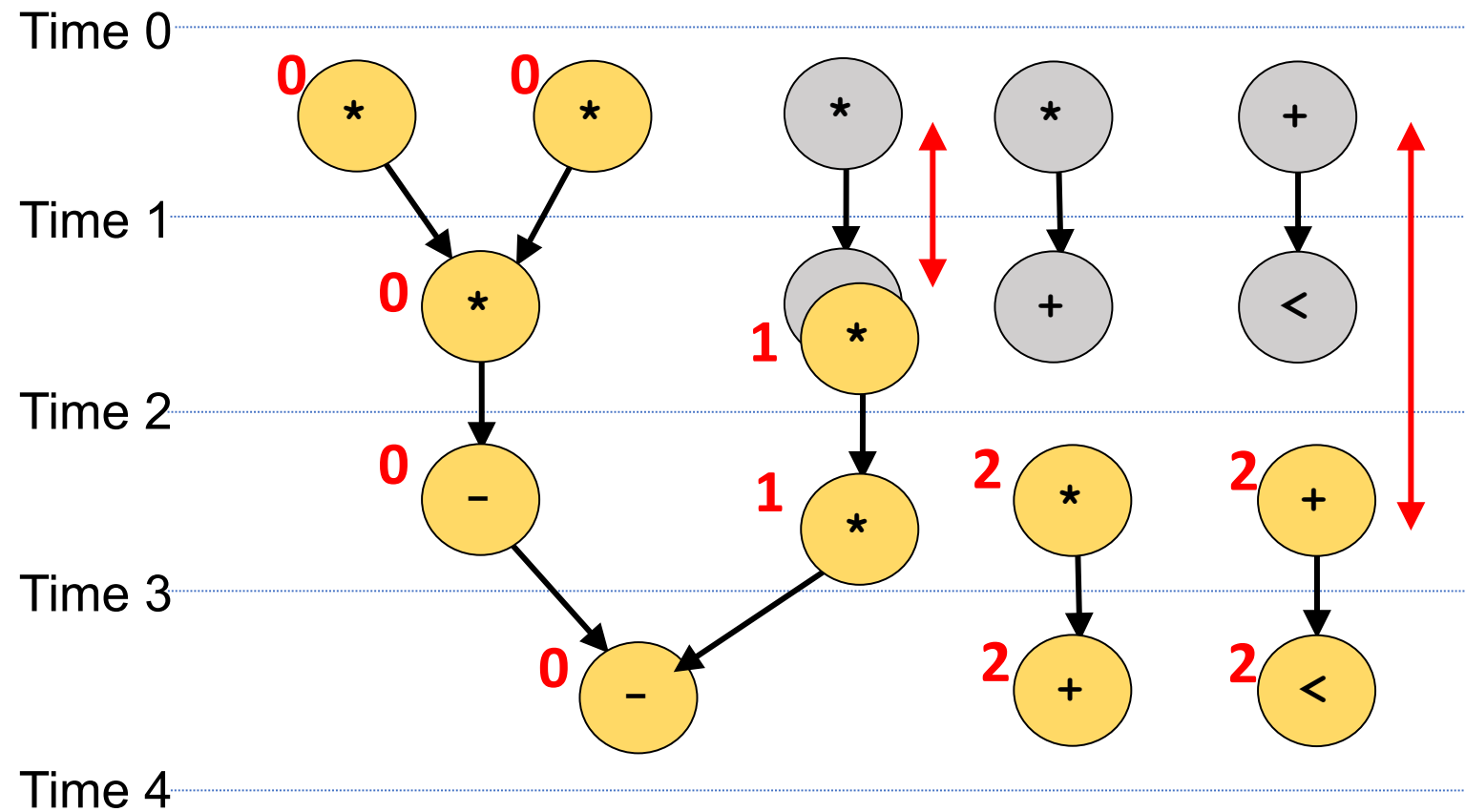


# Mobility

- Mobility of a task is the difference of the start times ALAP-ASAP.



# Mobility



# Summary

- Parallel design of HW and SW offers many advantages
- System design maps behavior to structure
- Synthesis is subdivided into partitioning and mapping
  - subdivided into allocation, binding and scheduling tasks

# HW/SW Codesign Synthesis

Prof. Dr. Rolf Drechsler

Dr. Muhammad Hassan

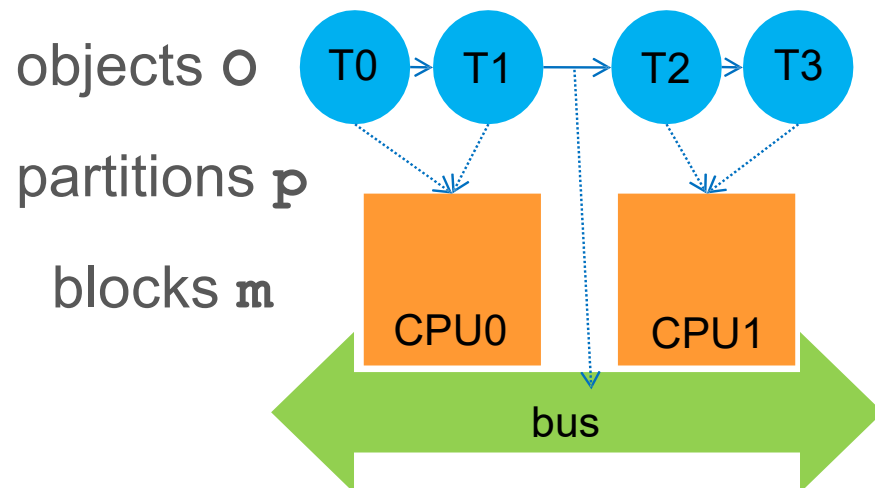
M.Sc. Jan Zielasko

M.Sc. Milan Funck

- Backup

# The Formal Partitioning Problem

- Assign **n objects**  $O = \{o_1, \dots, o_n\}$  to **m blocks** by **partitioning**  $P = \{p_1, \dots, p_m\}$ , such that
  - $P_1 \cup P_2 \cup \dots \cup P_m = O$  (i.e., **all objects** are **assigned**/mapped)
  - $P_i \cap P_j = \{ \} \forall i, j : i \neq j$  (i.e., an **object** is **not assigned**/mapped **twice**)
  - **Costs**  $C(P)$  are **minimized**



- **objects** = process network graph nodes
- **blocks** = architecture graph nodes
- **cost** = measured/estimated with dedicated cost functions (e.g., latency, power, hardware cost)



# Partitioning Methods

- Exact Methods
  - Enumeration
  - Integer Linear Programs (ILP)
  - ...
- Heuristic Methods
  - Simulated annealing
  - Greedy
  - Constructive methods
  - Random mapping
  - Hierarchical clustering
  - Iterative methods
  - Evolutionary algorithms
  - ...

# Integer Linear Programs (1)

- **Binary Variable**  $x_{i,k} = 1$ : Object  $o_i$  in Block  $p_k$   
 $x_{i,k} = 0$ : Object  $o_i$  not in Block  $p_k$
- **Cost**  $c_{i,k}$ , if Object  $o_i$  in Block  $p_k$
- $n$  – Number of objects,  $m$  – Number of blocks
- **Integer Linear Program:**

$$x_{i,k} \in \{0,1\} \quad 1 \leq i \leq n, 1 \leq k \leq m$$

$$\sum_{k=1}^m x_{i,k} = 1 \quad 1 \leq i \leq n$$

$$\text{minimize} \quad \sum_{k=1}^m \sum_{i=1}^n c_{i,k} x_{i,k} \quad 1 \leq k \leq m, 1 \leq i \leq n$$

# Integer Linear Programs (2)

- Constraints are modeled by side conditions to the ILP  
e.g.,: maximum number of  $h_k$  Objects in Block  $k$

$$\sum_{i=1}^n x_{i,k} \leq h_k \quad 1 \leq k \leq m$$

- ILP is an exact procedure, NP-complete
- Suitable only for
  - Small problems
  - Linear constraints

# Simulated Annealing


- Simulated Annealing
  - Metals and glass adopt a state of minimum energy when cooled under certain conditions:
    - **thermodynamic equilibrium** is reached **at any temperature**
    - the **temperature** is **reduced** arbitrarily **slowly**
- Time complexity
  - From **exponential** to **constant**, depending on the implementation of the functions *Equilibrium*, *DecreaseTemp*, *Frozen*
  - The longer the runtime, the better the results
  - Practice: Functions constructed in such a way that polynomial runtime is achieved

# Greedy Algorithms

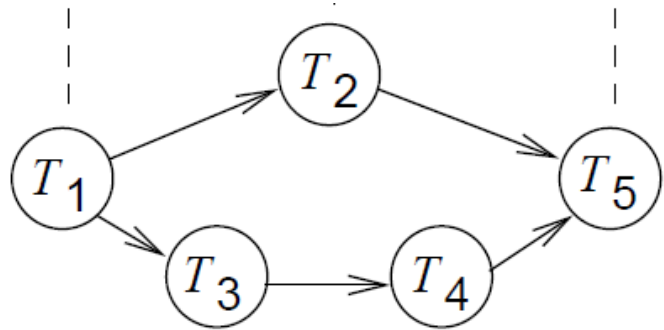
- **Bi-partitioning problem**  $P = \{p_{SW}, p_{HW}\}$
- Migrate objects to the other block until no more improvement occurs

```
REPEAT {  
   $P_{old} = P$ ;  
  FOR  $i = 1$  TO  $n$  {  
    IF ( $f(\text{Move}(P, o_i)) < f(P)$ ) {  
       $P = \text{Move}(P, o_i)$ ;  
    }  
  }  
}  
UNTIL ( $P == P_{old}$ )
```

Cost function



# Example 2: Description



- **Costs:**

- $H1 = 20$
- $H2 = 25$
- $H3 = 30$
- $P = 5$

- HW **types**  $H1$ ,  $H2$  and  $H3$
- Processors of type  $P$
- Tasks  $T1$  to  $T5$
- Static scheduling
- Execution times:

$T$	$H1$	$H2$	$H3$	$P$
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

# Example 2: Operation assignment constraints

$T$	$H1$	$H2$	$H3$	$P$
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

- $x_{i,k}$  :  $k \in \{1,3\}$ : Task  $i$  is mapped to  $H1$ ,  $H2$  or  $H3$
- $x_{i,4}$ : Task  $i$  is mapped to processor  $P$

$$\forall i : \sum_{k=1}^3 x_{i,k} + x_{i,4} = 1$$

$$x_{1,1} + x_{1,4} = 1$$

$$x_{2,2} + x_{2,4} = 1$$

$$x_{3,3} + x_{3,4} = 1$$

$$x_{4,3} + x_{4,4} = 1$$

$$x_{5,1} + x_{5,4} = 1 \text{ (task 5 mapped to } H1 \text{ or to } P)$$

# Example 2: Results

- Time constraint = 100 time units

$T$	$H1$	$H2$	$H3$	$P$
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

Solution (educated guessing) :

$T_1 \rightarrow H_1$

$T_2 \rightarrow H_2$

$T_3 \rightarrow P$

$T_4 \rightarrow P$

$T_5 \rightarrow H_1$

