

Large-Scale Distributed Systems and Networks

TDDE35

Lectures on

Embedded Systems

Petru Eles

**Institutionen för Datavetenskap (IDA)
Linköpings Universitet**

**email: petru.eles@liu.se
<http://www.ida.liu.se/~petel71/>
phone: 28 1396
B building, 329:220**

Information

Lecture notes: available from the course page, latest 24 hours before the lecture.

Recommended literature:

Peter Marwedel: "*Embedded System Design*",
Springer, 2nd edition 2011, 3d edition, 2018.

Edward Lee, Sanjit Seshia: "*Introduction to Embedded Systems - A
Cyber-Physical Systems Approach*",
LeeSeshia.org, 1st edition 2011, 2nd edition 2015.

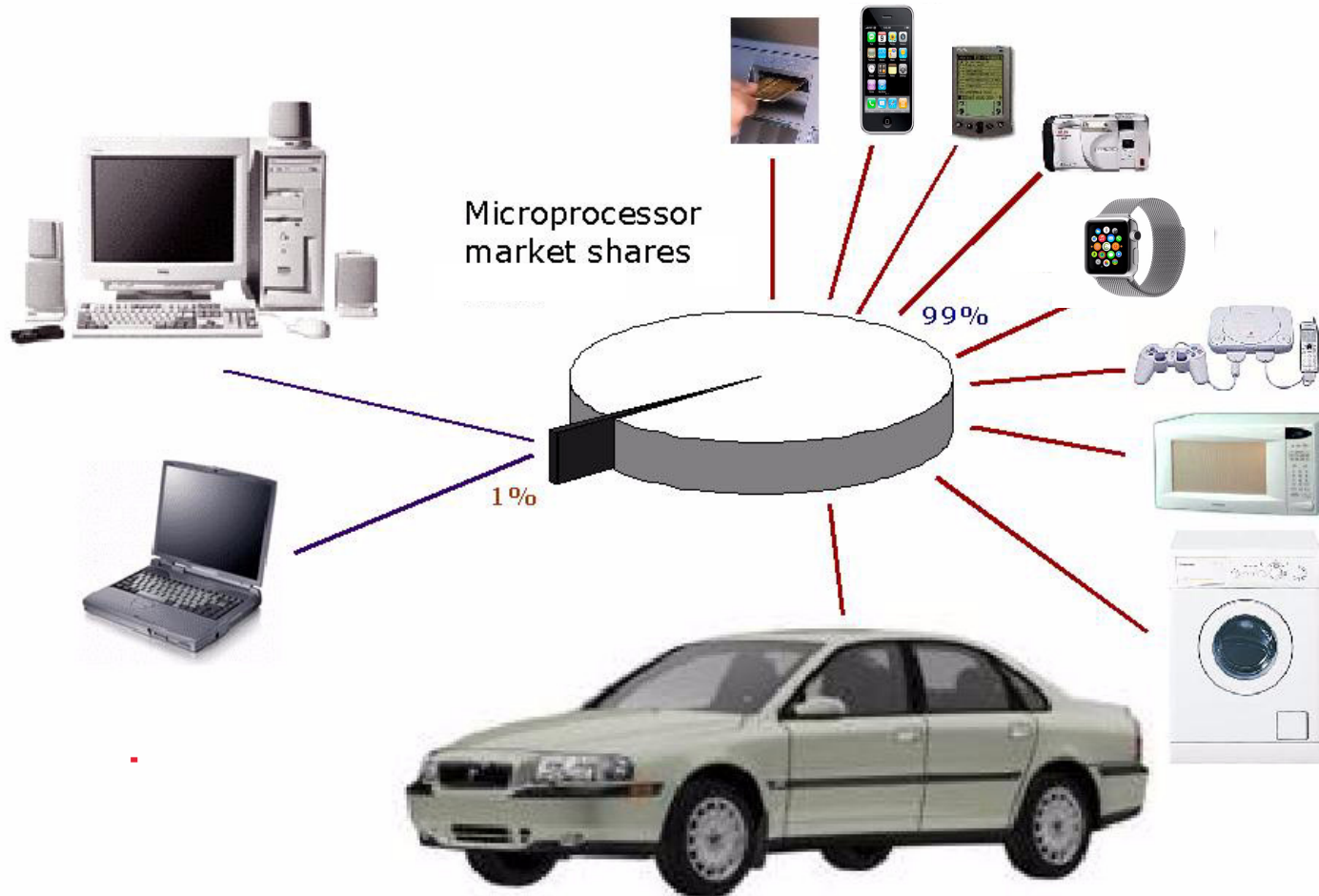
EMBEDDED SYSTEMS AND THEIR DESIGN

1. **What is an Embedded System**
2. **Characteristics of Embedded Applications**
3. **Modeling of Embedded Systems**
4. **The Traditional design Flow**
5. **An Example**
6. **A New Design Flow**
7. **The System Level**
8. **Power/Energy Consumption - a Major Issue**

That's how we use microprocessors

General purpose systems

Embedded systems



What is an Embedded System?

There are several definitions around!

- Some highlight what it is (not) used for:

“An embedded system is any sort of device which includes a programmable component but itself is not intended to be a general purpose computer.”

What is an Embedded System?

There are several definitions around!

- Some highlight what it is (not) used for:

“An embedded system is any sort of device which includes a programmable component but itself is not intended to be a general purpose computer.”

- Some focus on what it is built from:

“An embedded system is a collection of programmable parts surrounded by ASICs and other standard components, that interact continuously with an environment through sensors and actuators.”

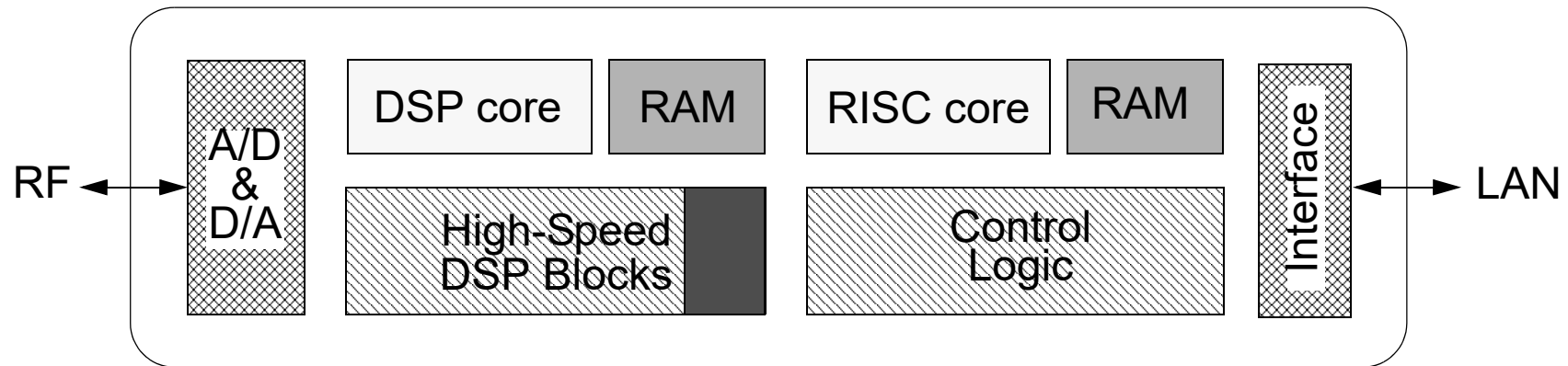
What is an Embedded System?






Some of the main characteristics:

- ❑ **Dedicated (not *general purpose*)**
- ❑ **Contains a programmable component**
- ❑ **Interacts (continuously) with the environment**

Two Typical Implementation Architectures

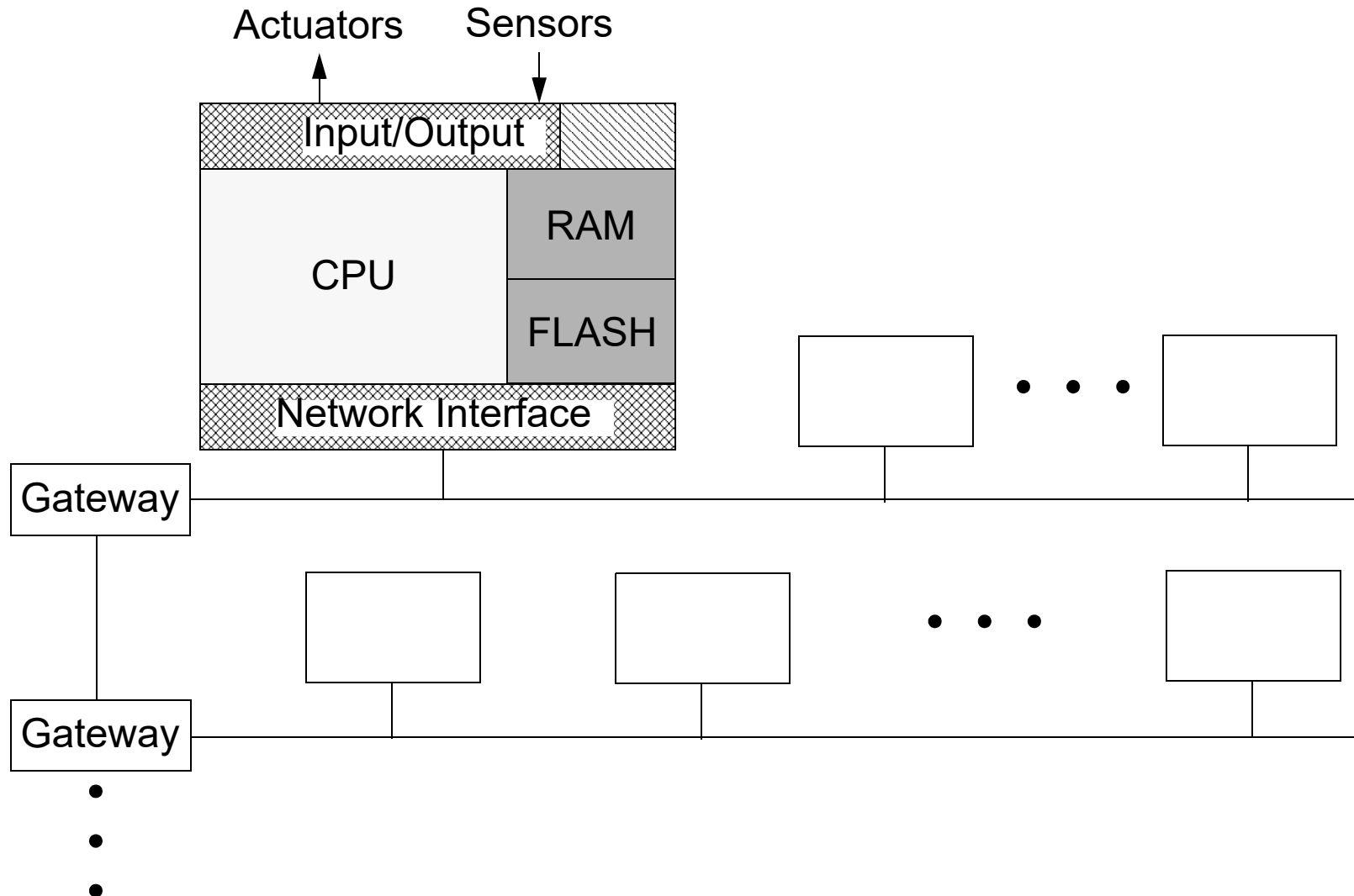
Telecommunication System on Chip



-  Programmable processor
 -  ASIC block (Application Specific Integrated Circuit)
 -  Standard block
 -  Memory
 -  Reconfigurable logic (FPGA)
- } dedicated electronics

Two Typical Implementation Architectures

Distributed Embedded System (automotive application)



The Software Component

Software running on the programmable processors:

- ❑ **Application tasks**
- ❑ **Real-Time Operating System**
- ❑ **I/O drivers, Network protocols, Middleware**

Characteristics of Embedded Applications

What makes them special?

- Like with “ordinary” applications, functionality and user interfaces are often *very complex*.

But, in addition to this:

- Time constraints
- Power constraints
- Cost constraints
- Safety
- Time to market

Time constraints

- **Embedded systems have to perform in real-time: if data is not ready by a certain deadline, the system fails to perform correctly.**
 - ***Hard deadline:* failure to meet leads to major hazards.**
 - ***Soft deadline:* failure to meet is tolerated but affects quality of service.**

Power constraints

- There are several reasons why low power/energy consumption is required:
 - **Cost aspects:**
 - High energy consumption ⇒ large electricity bill
 - expensive power supply
 - expensive cooling system
 - **Reliability**
 - High power consumption ⇒ high temperature that affects life time
 - **Battery life**
 - High energy consumption ⇒ short battery life time
 - **Environmental impact**

Cost constraints

- **Embedded systems are very often mass products in highly competitive markets and have to be shipped at a low cost.**

What we are interested in:

- **Manufacturing cost**
- **Design cost**

Safety

- Embedded systems are often used in life critical applications: avionics, automotive electronics, nuclear plants, medical applications, military applications, etc.
 - *Reliability* and *safety* are major requirements.
In order to guarantee safety during design:
 - Formal verification: mathematics-based methods to verify certain properties of the designed system.
 - Automatic synthesis: certain design steps are automatically performed by design tools.

Short time to market

- In highly competitive markets it is critical to catch the *market window*: a short delay with the product on the market can have catastrophic financial consequences (even if the quality of the product is excellent).
 - Design time has to be reduced!
 - Good design methodologies.
 - Efficient design tools.
 - Reuse of previously designed and verified (hardw&softw) blocks.
 - *Good designers who understand both software and hardware!*

Why is Design of Embedded Systems Difficult?

- ❑ **High Complexity**
- ❑ **Strong time&power constraints**
- ❑ **Low cost**
- ❑ **Short time to market**
- ❑ **Safety critical systems**

In order to achieve these requirements, systems have to be highly optimized.

Why is Design of Embedded Systems Difficult?

- ❑ High Complexity
- ❑ Strong time&power constraints
- ❑ Low cost
- ❑ Short time to market
- ❑ Safety critical systems

In order to achieve these requirements, systems have to be highly optimized.



Both hardware and software aspects have to be considered simultaneously!

From Specifications to Implementations

- **Specification**: An informal description of basic requirements and properties of a system
 - The designer gets a *specification* as an input and, finally, has to produce an *implementation*.
This is usually done as a sequence of *refinement steps*.

System Specifications

- **A specification captures:**
 - **The basic required behaviour of the system**
 - **E.g. as a relation between inputs and outputs**
 - **Other (non-functional) requirements**
 - **time constraints**
 - **power/energy constraints**
 - **safety requirements**
 - **environmental aspects**
 - **cost, weight, etc.**

System Model

- Starting from the informal specification, as an early step in the design flow, a more formal *system model* is produced.
- The model is a description of certain aspects/properties of the system. Models are abstract, in the sense that they omit details and concentrate on aspects that are significant for the design process.
- There are several modeling approaches (and modeling languages) used for embedded system design; examples:
 - Dataflow Models
 - Finite State Machines.

Dataflow Models

- Systems are specified as directed graphs where:
 - *nodes* represent computations (processes);
 - *arcs* represent totally ordered sequences (streams) of data (tokens).

Dataflow Models

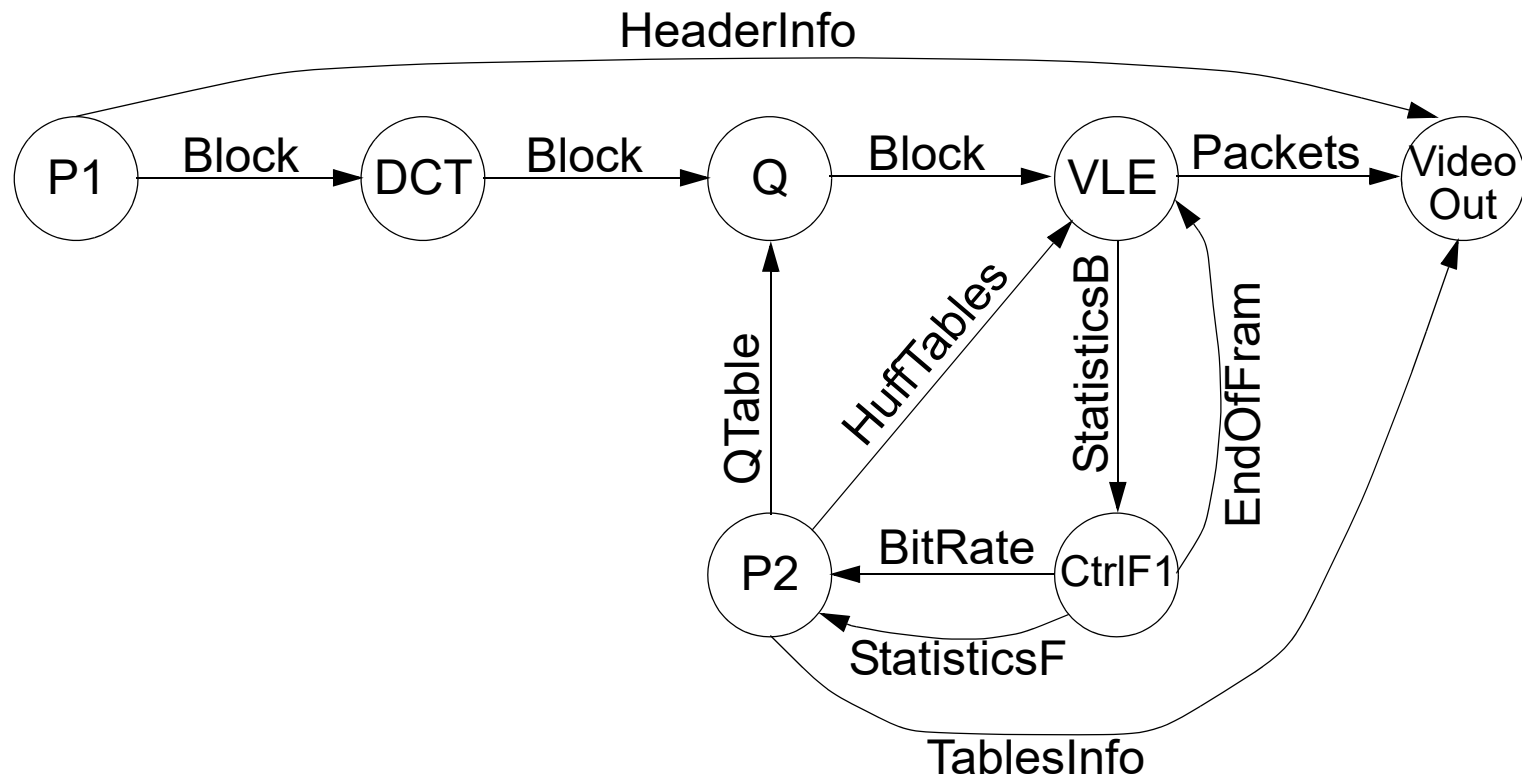
- **Systems are specified as directed graphs where:**
 - *nodes* represent computations (processes);
 - *arcs* represent totally ordered sequences (streams) of data (tokens).
- **Depending on their particular semantics, several models of computation based on dataflow have been defined:**
 - Kahn process networks
 - Dataflow process networks
 - Synchronous dataflow
 - - - - - -

Dataflow Models

- **Systems are specified as directed graphs where:**
 - *nodes* represent computations (processes);
 - *arcs* represent totally ordered sequences (streams) of data (tokens).
- **Depending on their particular semantics, several models of computation based on dataflow have been defined:**
 - Kahn process networks (KPN)
 - Dataflow process networks (DPN)
 - Synchronous dataflow (SDF)
 - - - - - -
- **Dataflow models are suitable for signal-processing algorithms:**
 - Code/decode, filter, compression, etc.
 - Streams of periodic and regular data samples

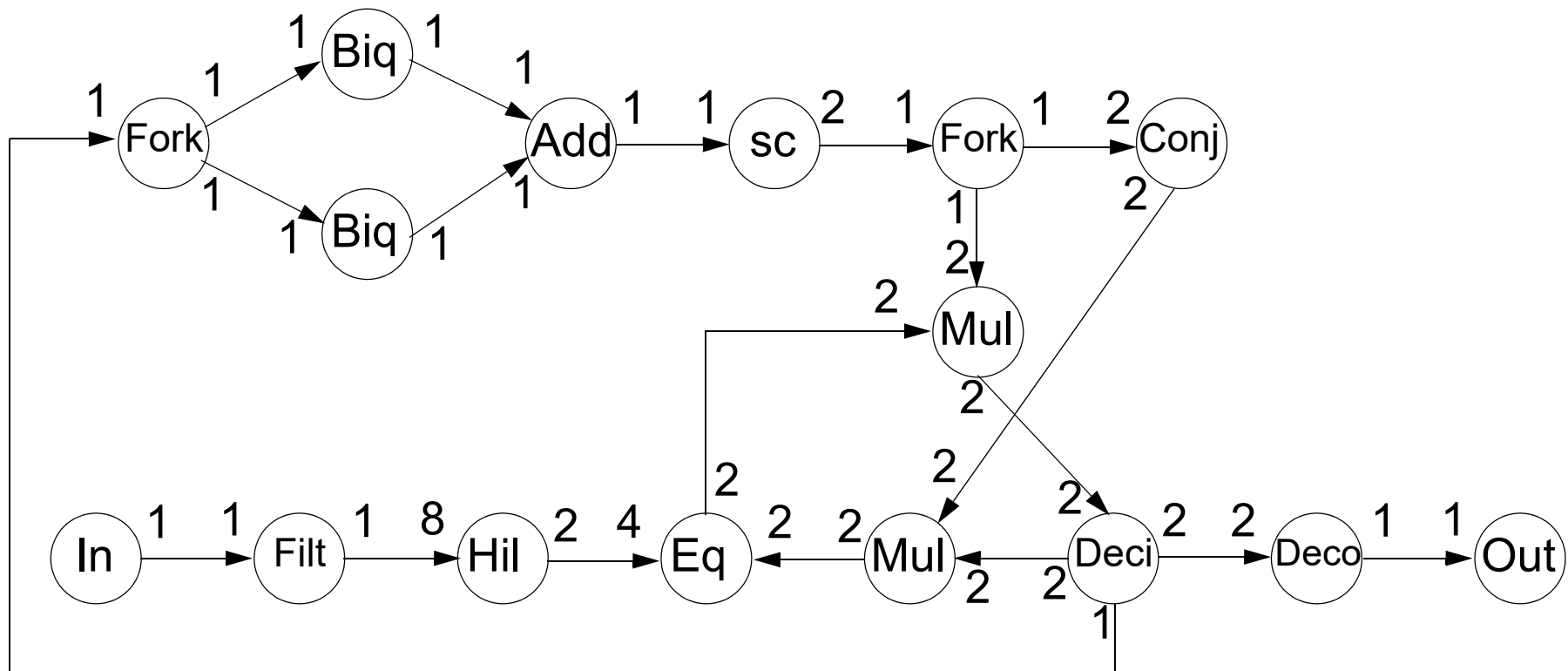
Dataflow Models

KPN model of encoder for Motion JPEG (M-JPEG) video compression format:



Dataflow Models

SDF model of a Modem:



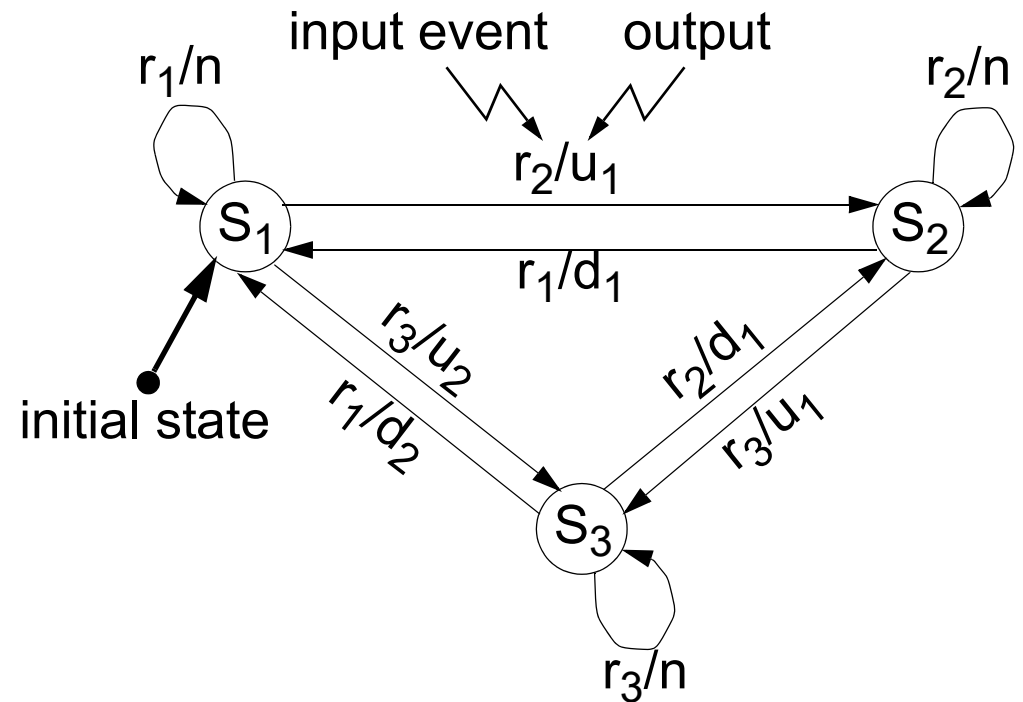
Finite State Machines

- The system is characterised by *explicitly* depicting its states as well as the transitions from one state to another.
- One particular state is specified as the initial one
- States and transitions are in a finite number.
- Transitions are triggered by input events.
- Transitions generate outputs.
- FSMs are suitable for modeling control dominated reactive systems (react on inputs with specific outputs)

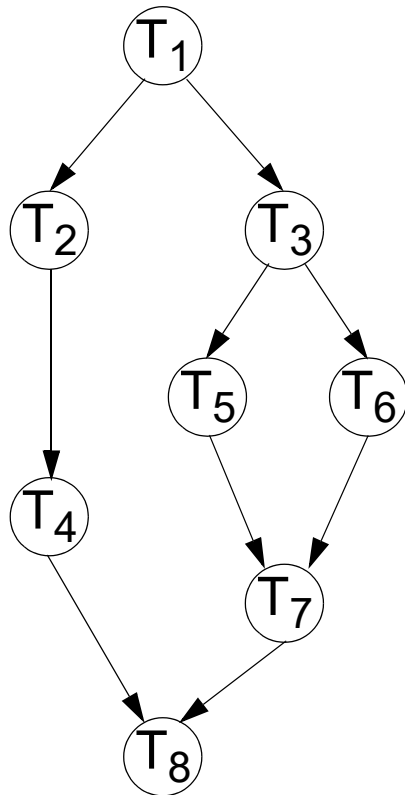
Finite State Machines

Elevator controller

- Input events: $\{r_1, r_2, r_3\}$
 - r_i : request from floor i .
- Outputs: $\{d_2, d_1, n, u_1, u_2\}$
 - d_i : go down i floors
 - u_i : go up i floors
 - n : stay idle
- States: $\{S_1, S_2, S_3\}$
 - S_i : elevator is at floor i .



A Design Example



The system to be implemented is modelled as a *task graph*:

- a node represents a *task* (a unit of functionality activated as response to a certain input and which generates a certain output).
- an edge represents a precedence constraint and data dependency between two tasks.

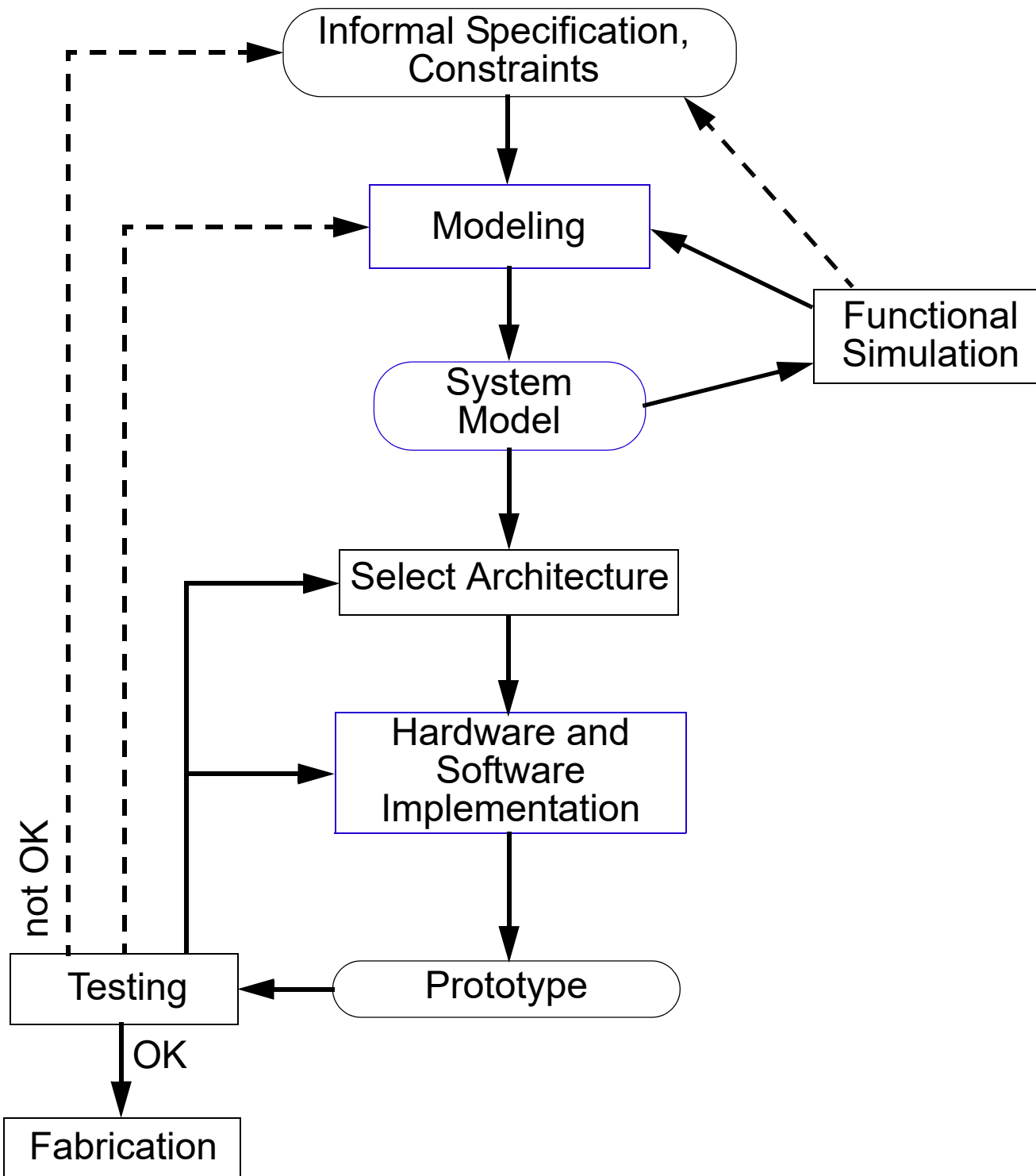
Period: 42 time units

- The task graph is activated every 42 time units \Rightarrow an activation has to terminate in time less than 42.

Cost limit: 8

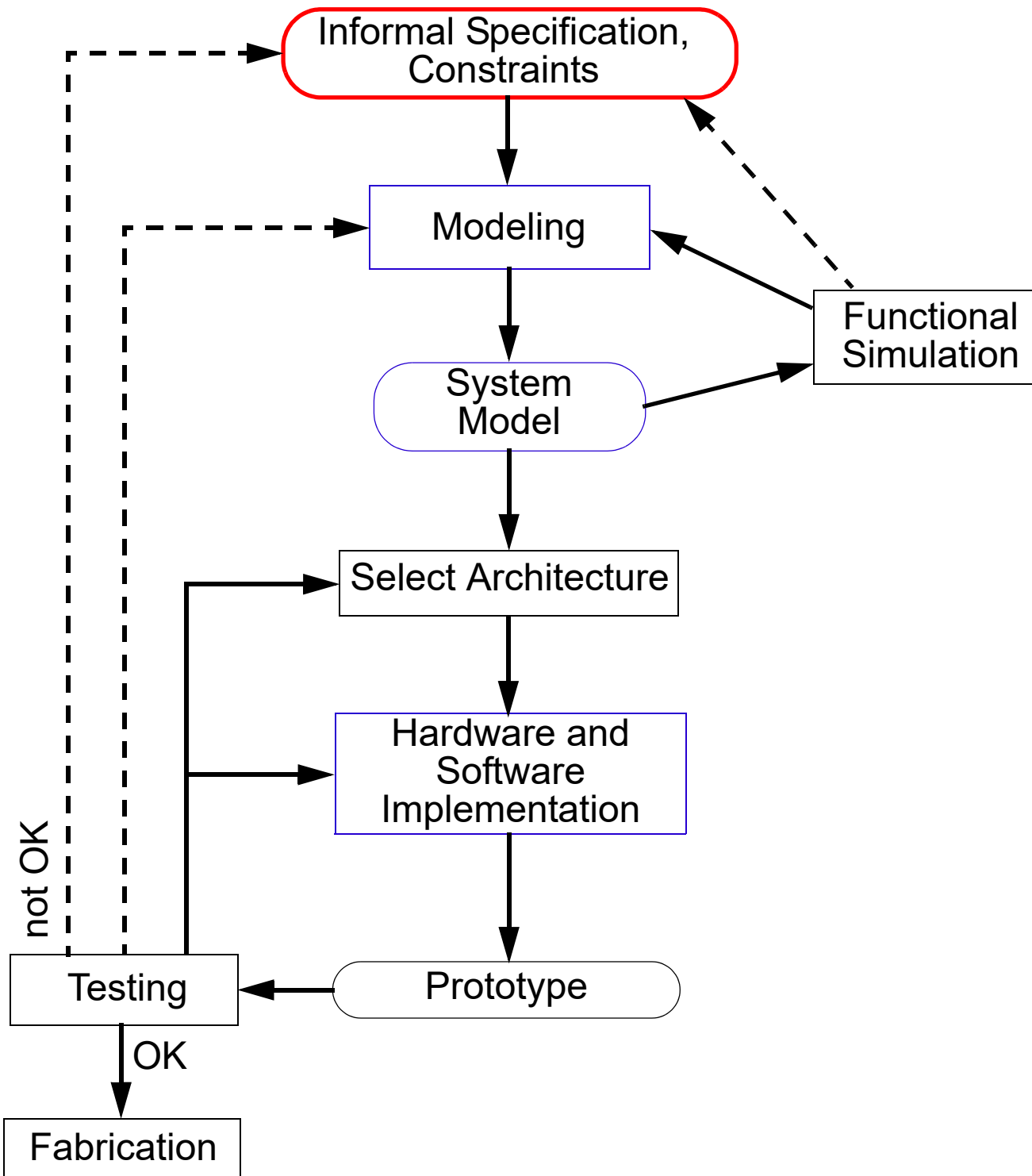
- The total cost of the implemented system has to be less than 8.

Traditional Design Flow



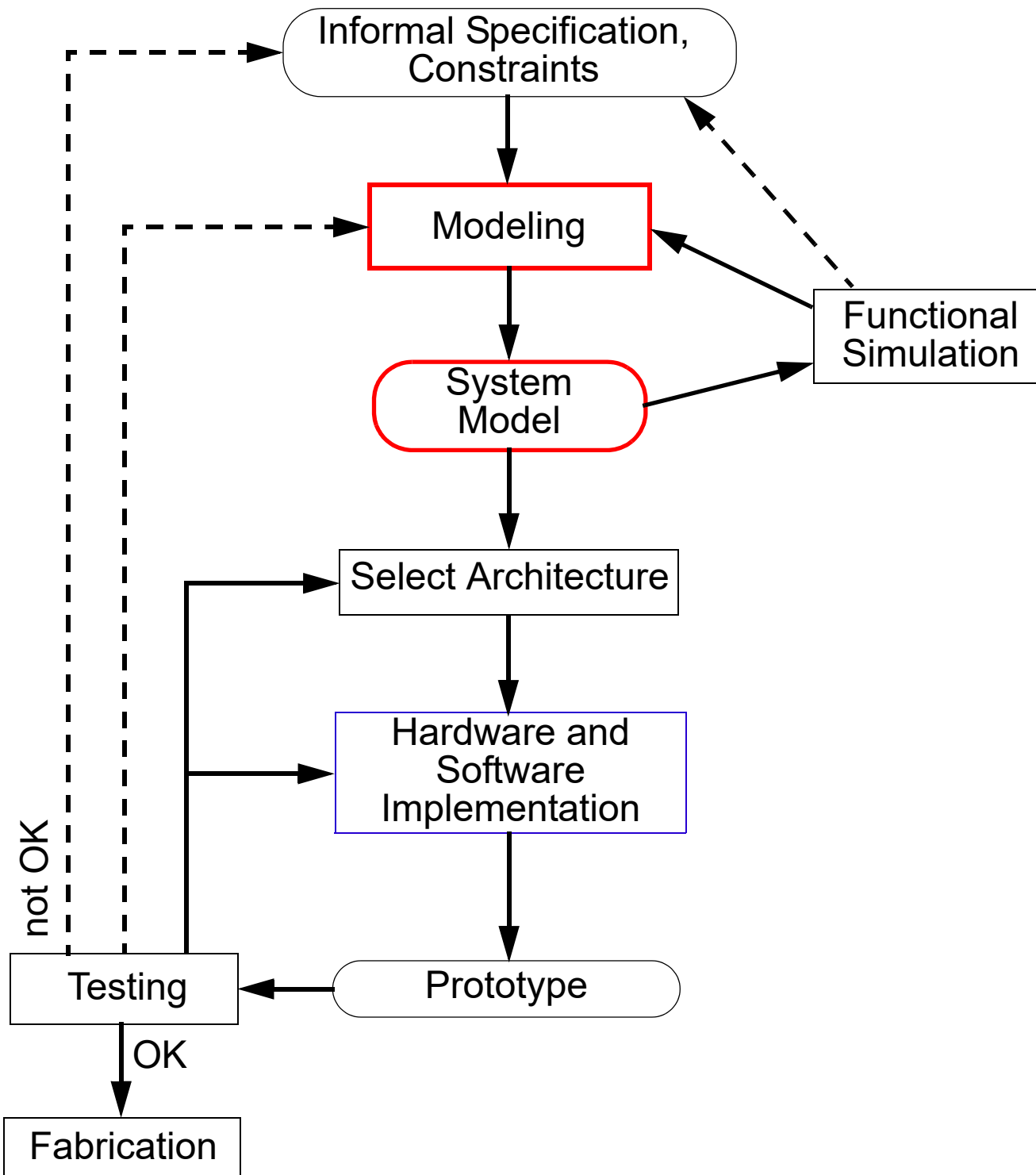
Traditional Design Flow

1. Start from some informal specification of functionality and a set of constraints.



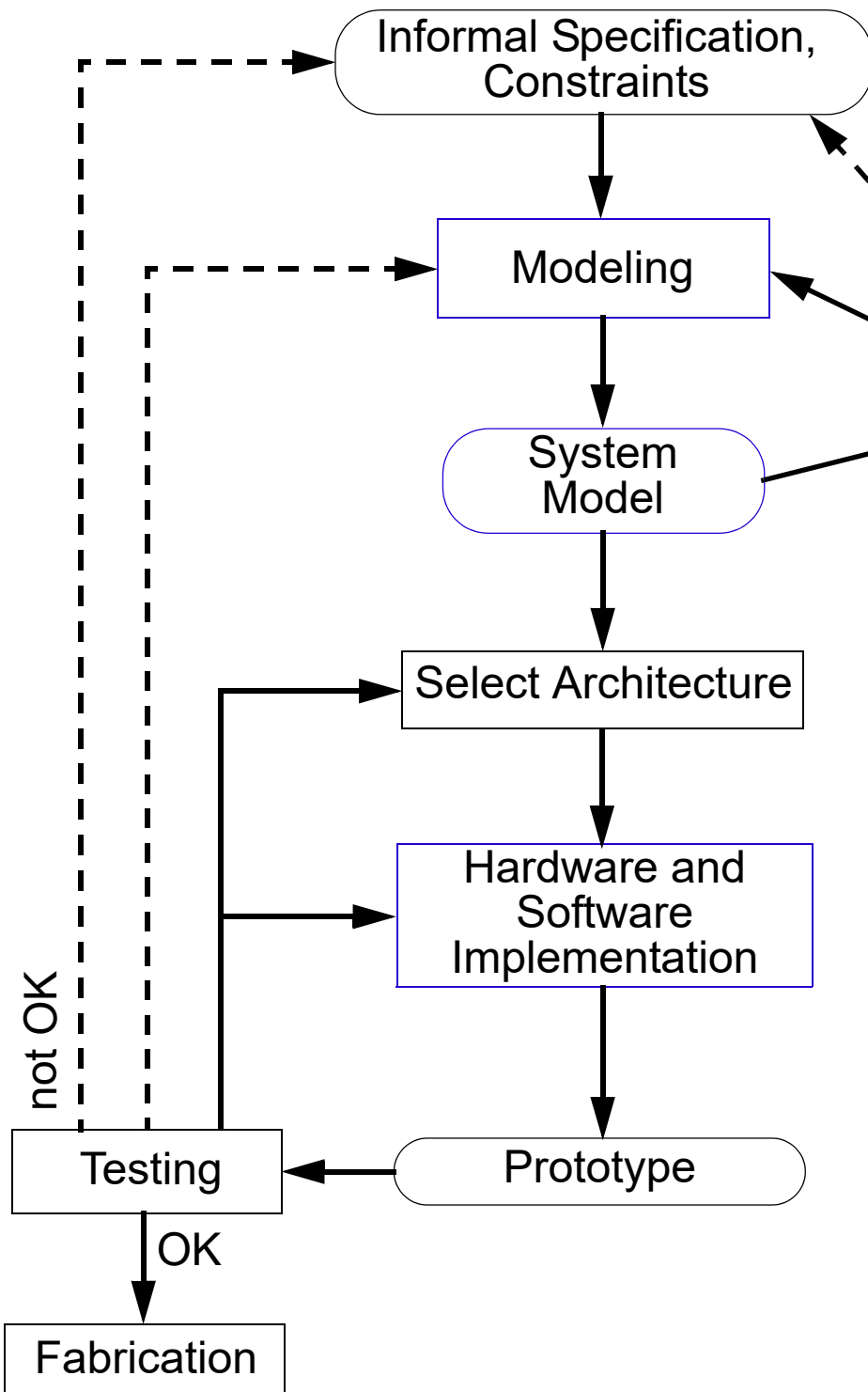
Traditional Design Flow

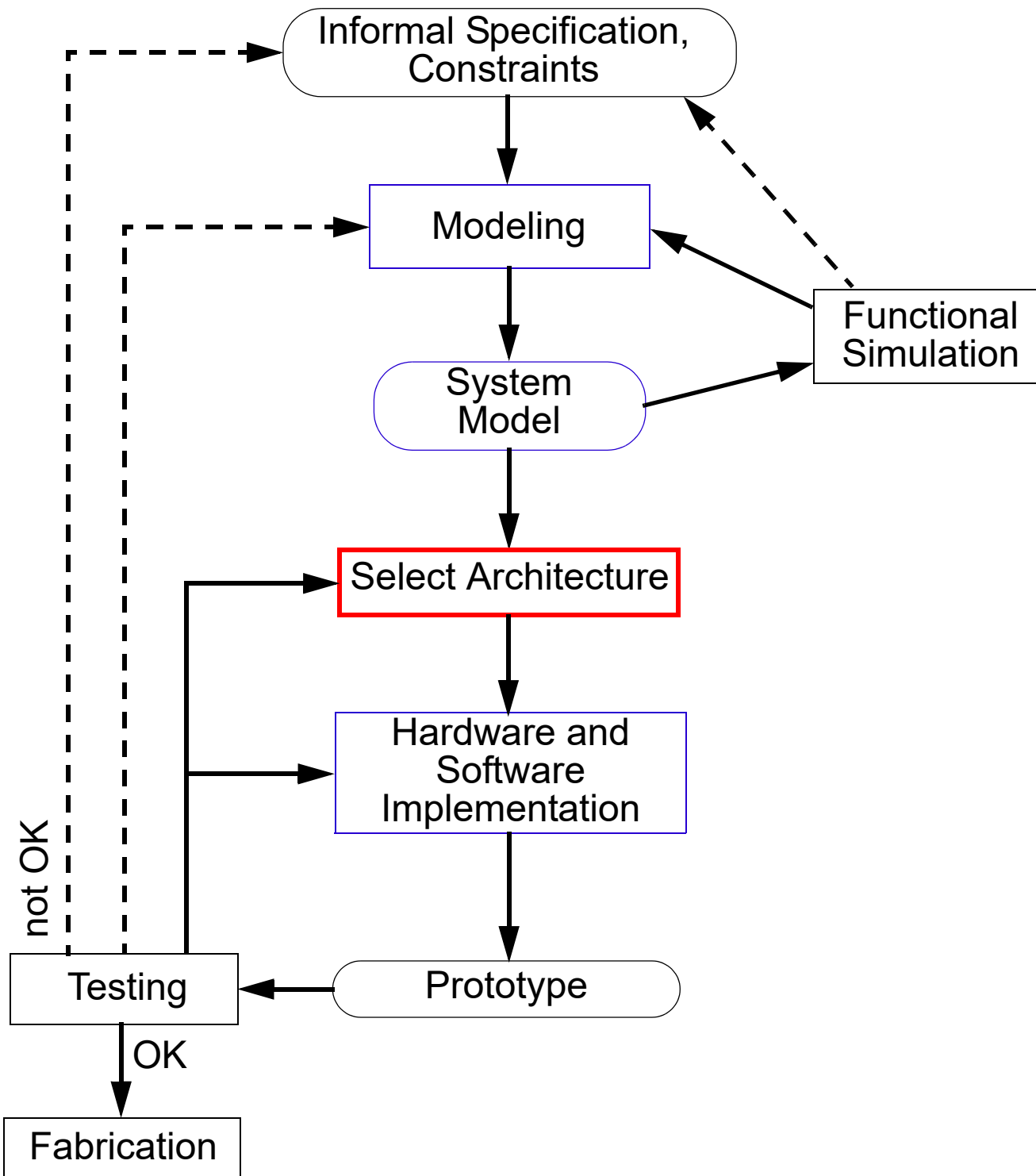
1. Start from some informal specification of functionality and a set of constraints.
2. Generate a more formal model of the functionality, based on some modeling concept. Such model is our task graph.



Traditional Design Flow

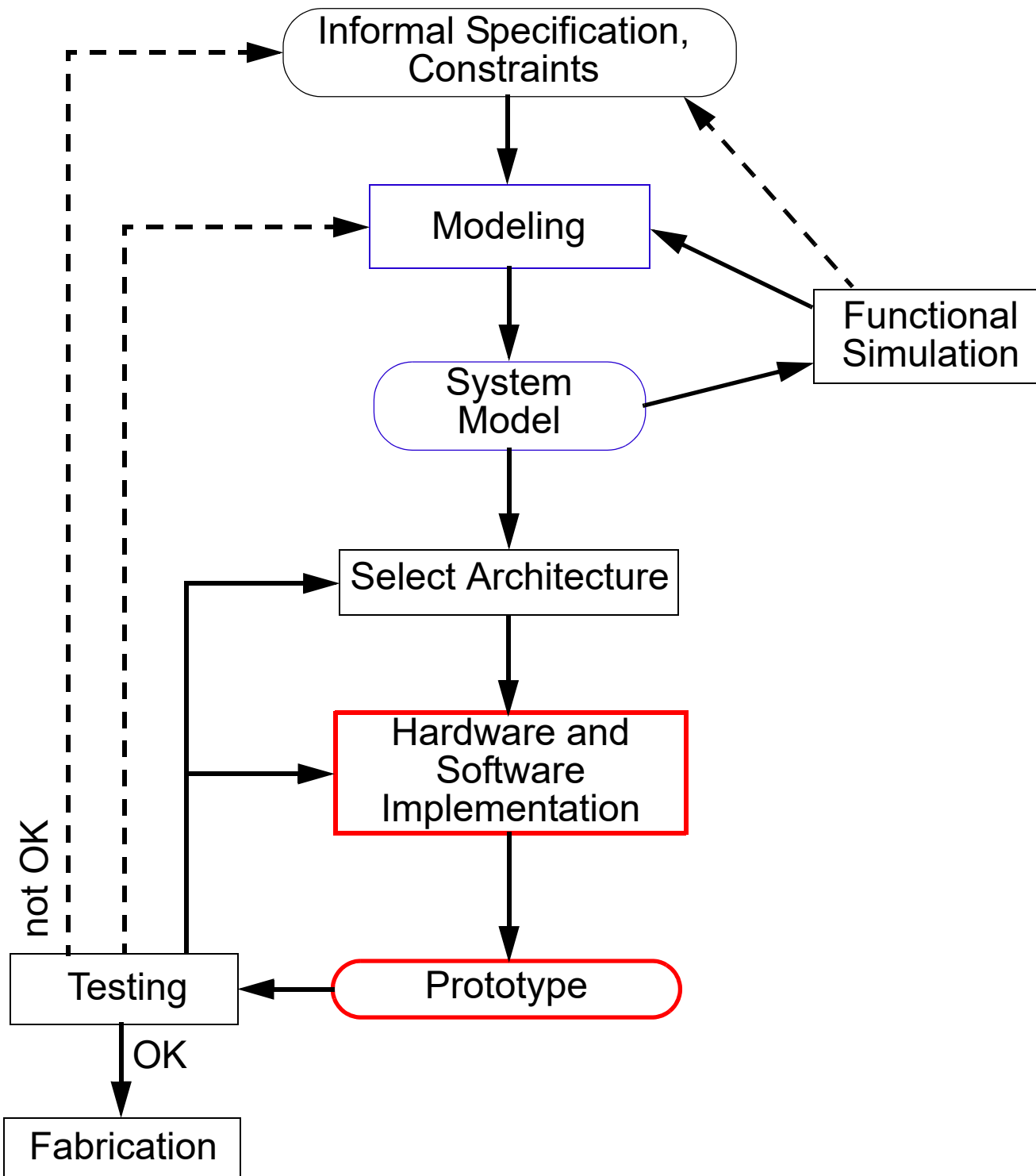
1. Start from some informal specification of functionality and a set of constraints.
2. Generate a more formal model of the functionality, based on some modeling concept. Such model is our task graph.
3. Simulate the model in order to check the functionality. If needed make adjustments.





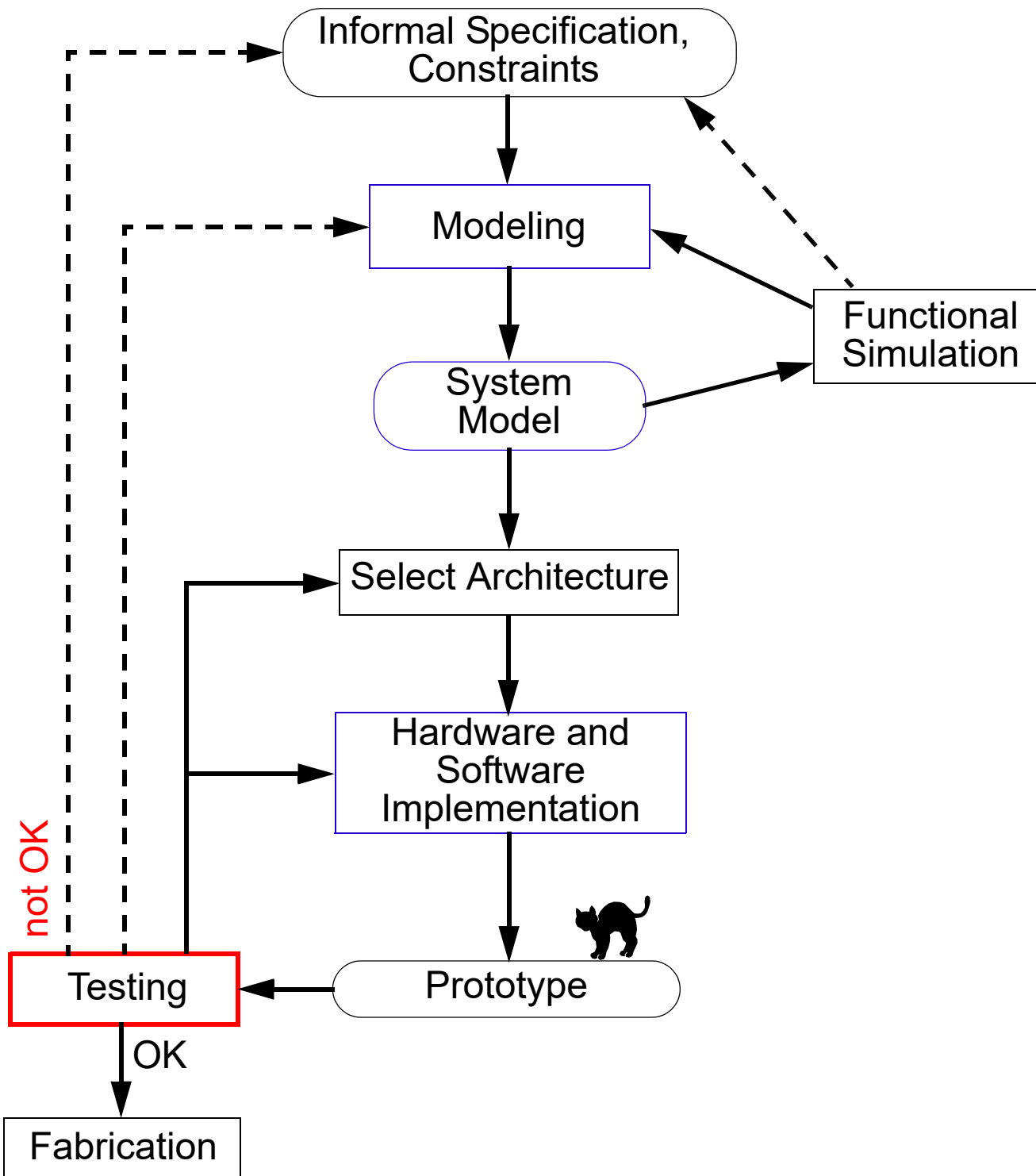
Traditional Design Flow

1. Start from some informal specification of functionality and a set of constraints.
2. Generate a more formal model of the functionality, based on some modeling concept. Such model is our task graph.
3. Simulate the model in order to check the functionality. If needed make adjustments.
4. Choose an architecture (μ processor, buses, etc.) such that cost limits are satisfied and, you hope, time and power constraints are fulfilled.



Traditional Design Flow

1. Start from some informal specification of functionality and a set of constraints.
2. Generate a more formal model of the functionality, based on some modeling concept. Such model is our task graph.
3. Simulate the model in order to check the functionality. If needed make adjustments.
4. Choose an architecture (μ processor, buses, etc.) such that cost limits are satisfied and, you hope, time and power constraints are fulfilled.
5. Build a prototype and implement the system.



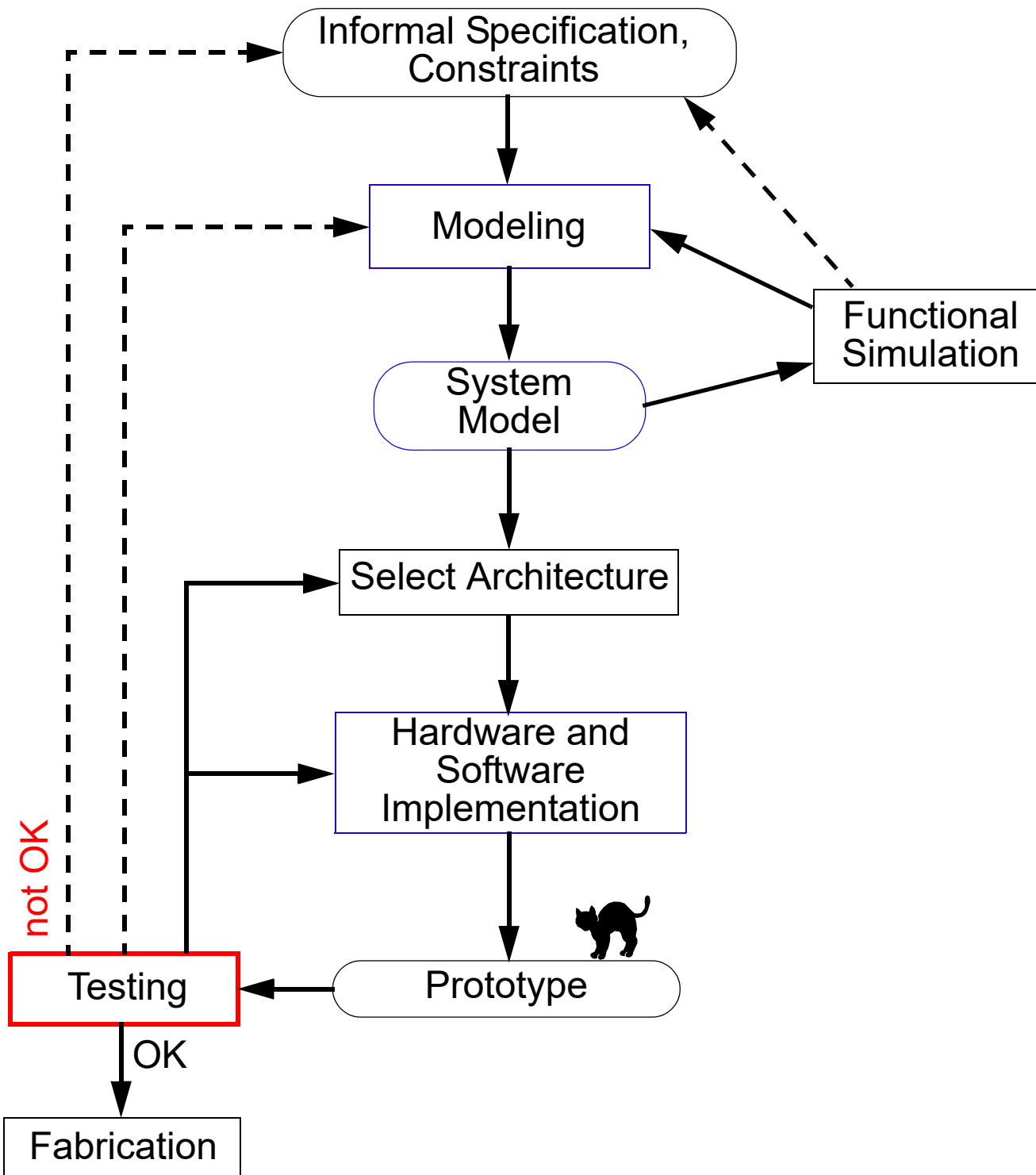
Traditional Design Flow

1. Start from some informal specification of functionality and a set of constraints.
2. Generate a more formal model of the functionality, based on some modeling concept. Such model is our task graph.
3. Simulate the model in order to check the functionality. If needed make adjustments.
4. Choose an architecture (μ processor, buses, etc.) such that cost limits are satisfied and, you hope, time and power constraints are fulfilled.
5. Build a prototype and implement the system.
6. Verify the system: neither time nor power constraints are satisfied!!!

Traditional Design Flow

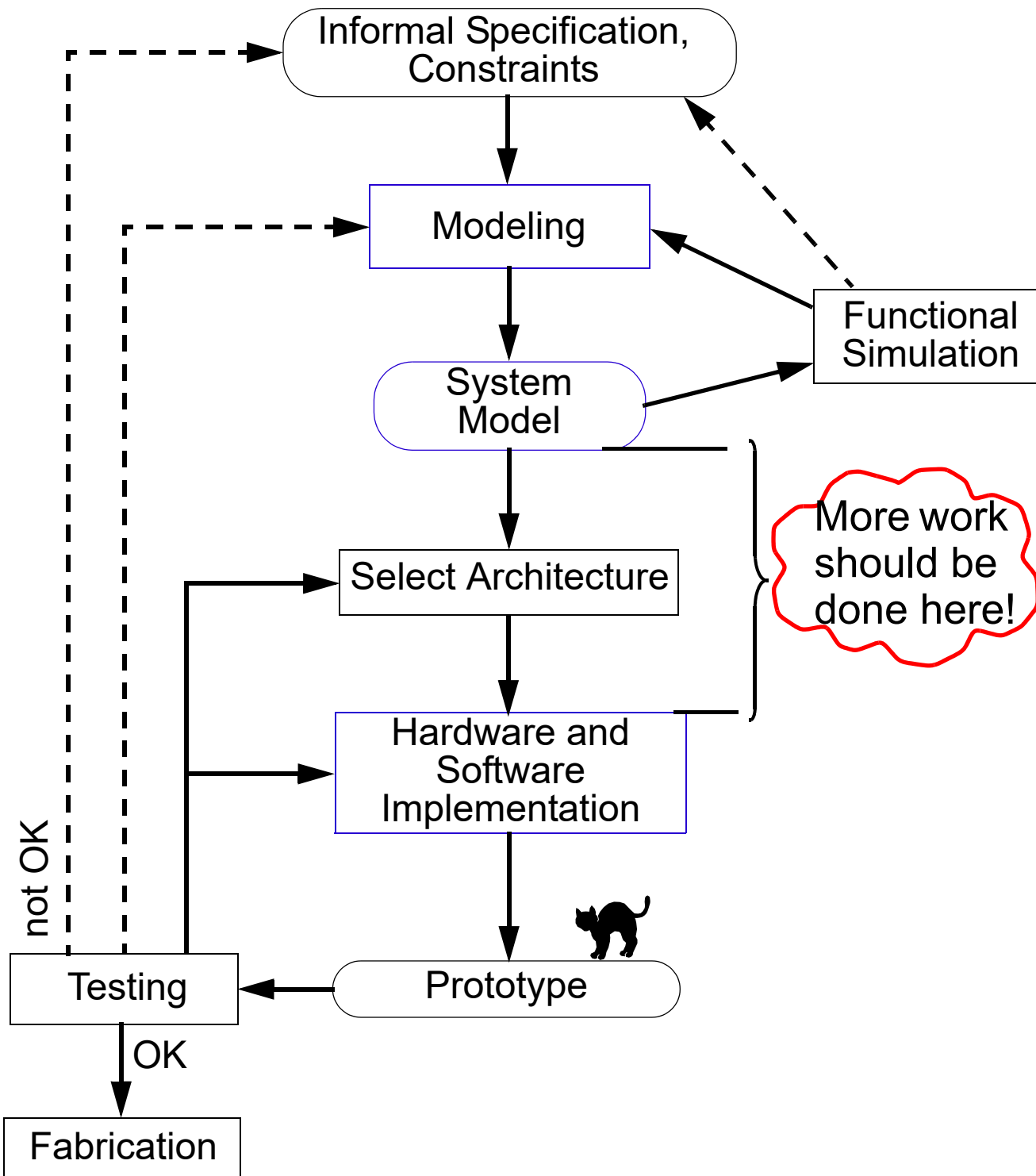
Now you are in great trouble: you have spent a lot of time and money and nothing works!

- ❑ **Go back to 4, choose a new architecture and start a new implementation.**
- ❑ **Or negotiate with the customer on the constraints.**

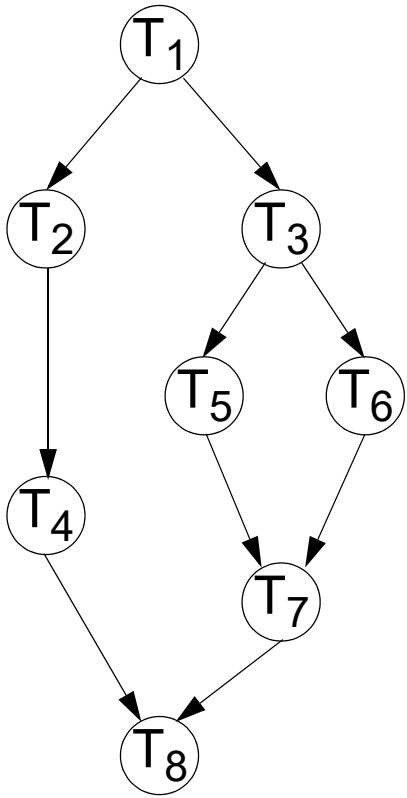


The Traditional Design Flow

- **The consequences:**
 - **Delays in the design process**
 - **Increased design cost**
 - **Delays in time to market \Rightarrow missed market window**
 - **High cost of failed prototypes**
 - **Bad design decisions taken under time pressure**
 - **Low quality, high cost products**

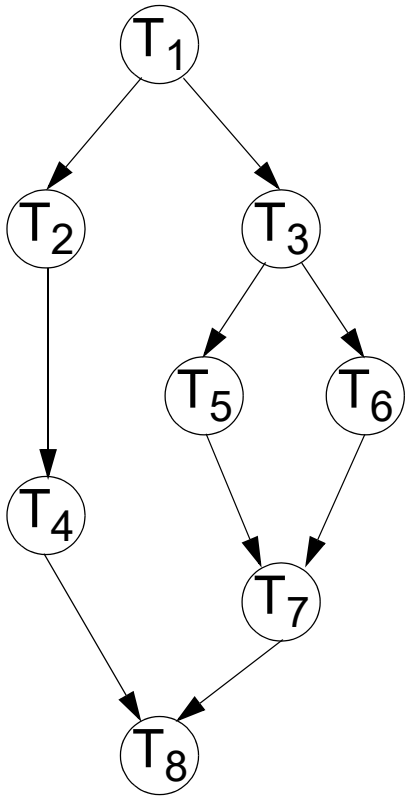


Example

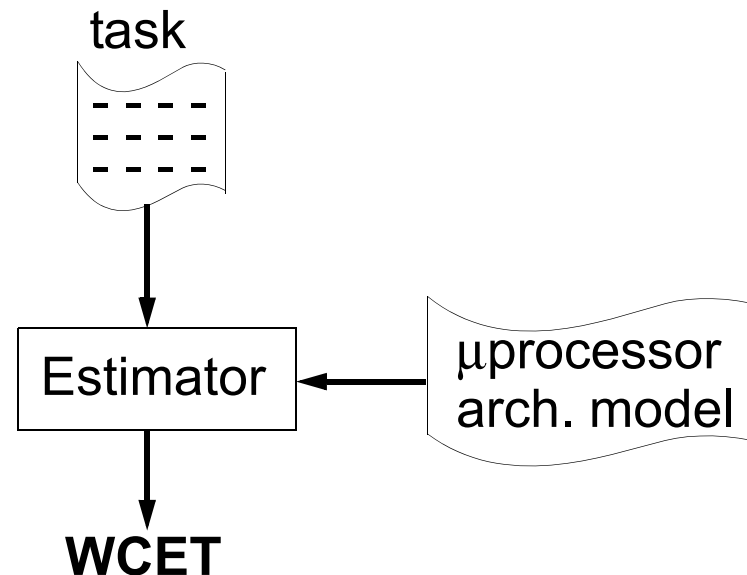


- We have the system model (task graph) which has been validated by simulation.
- We decide on a certain μ processor $\mu p1$, with cost 6.
- For each task the worst case execution time (WCET) when run on $\mu p1$ is *estimated*.

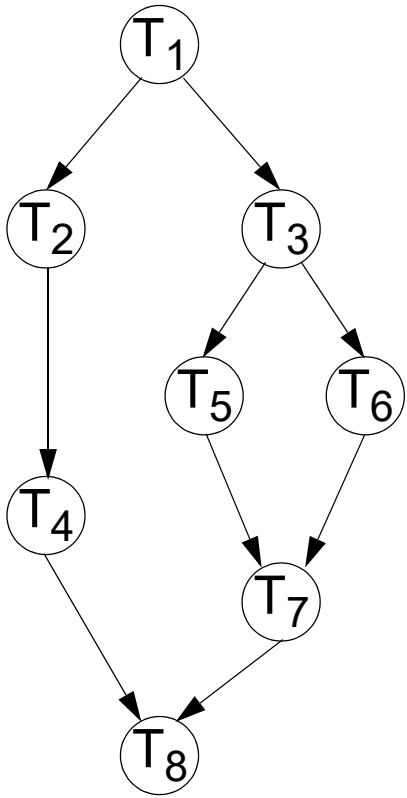
Example



- We have the system model (task graph) which has been validated by simulation.
- We decide on a certain μ processor $\mu p1$, with cost 6.
- For each task the worst case execution time (WCET) when run on $\mu p1$ is *estimated*.

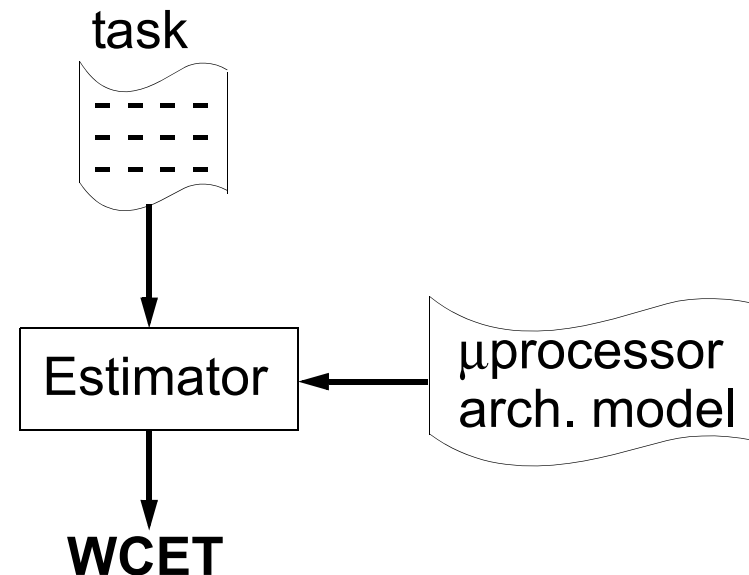


Example



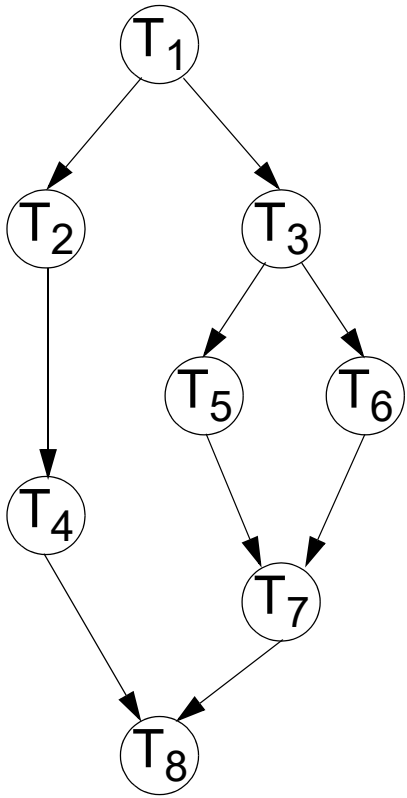
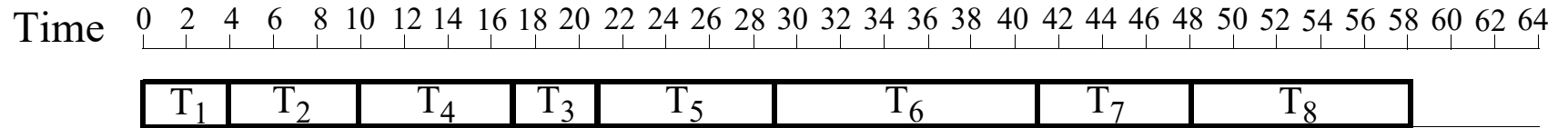
- We have the system model (task graph) which has been validated by simulation.
- We decide on a certain μ processor $\mu p1$, with cost 6.
- For each task the worst case execution time (WCET) when run on $\mu p1$ is *estimated*.

| Task | WCET |
|----------------|------|
| T ₁ | 4 |
| T ₂ | 6 |
| T ₃ | 4 |
| T ₄ | 7 |
| T ₅ | 8 |
| T ₆ | 12 |
| T ₇ | 7 |
| T ₈ | 10 |



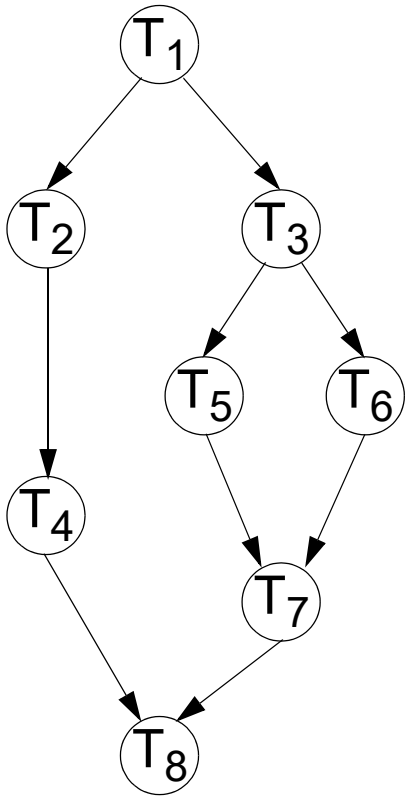
Example

We generate a schedule:

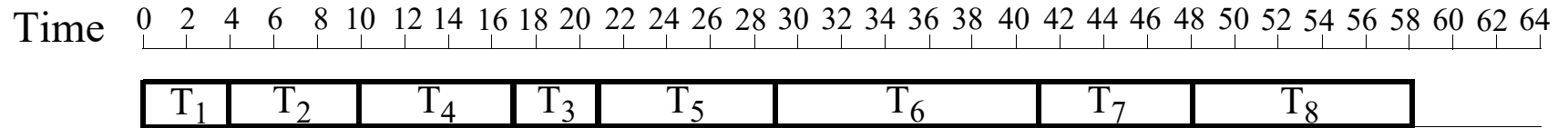


| Task | WCET |
|----------------|------|
| T ₁ | 4 |
| T ₂ | 6 |
| T ₃ | 4 |
| T ₄ | 7 |
| T ₅ | 8 |
| T ₆ | 12 |
| T ₇ | 7 |
| T ₈ | 10 |


Example



We generate a schedule:



Using the architecture with μ processor $\mu p1$ we got a solution with:

- ❑ Execution time: $58 > 42$ 
- ❑ Cost: $6 < 8$

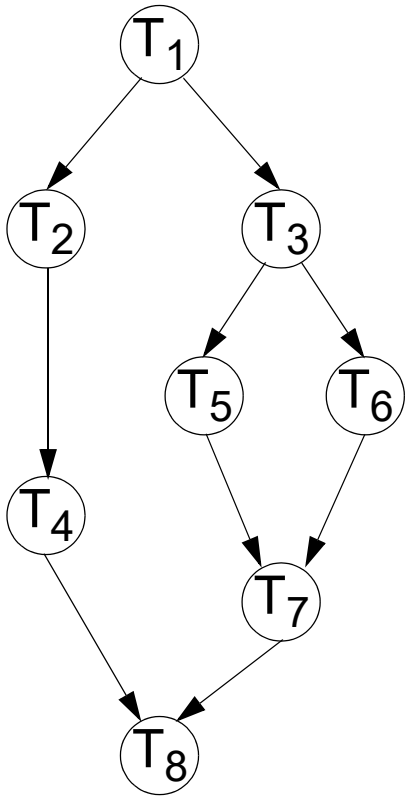


We have to try with another architecture!

| Task | WCET |
|----------------|------|
| T ₁ | 4 |
| T ₂ | 6 |
| T ₃ | 4 |
| T ₄ | 7 |
| T ₅ | 8 |
| T ₆ | 12 |
| T ₇ | 7 |
| T ₈ | 10 |

Example

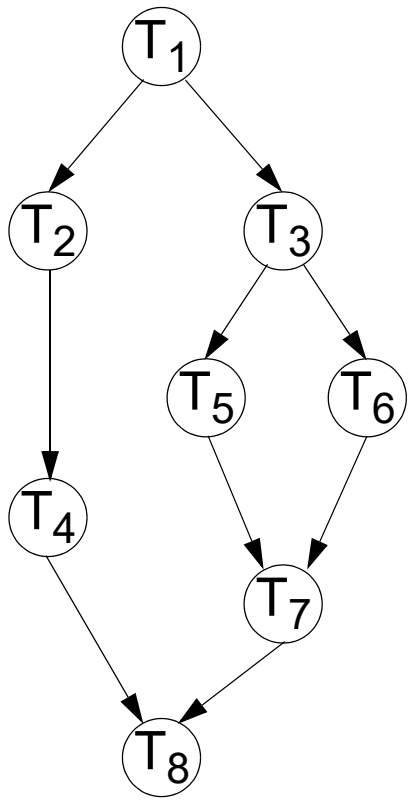
We look after a μ processor which is fast enough: $\mu p2$



Example

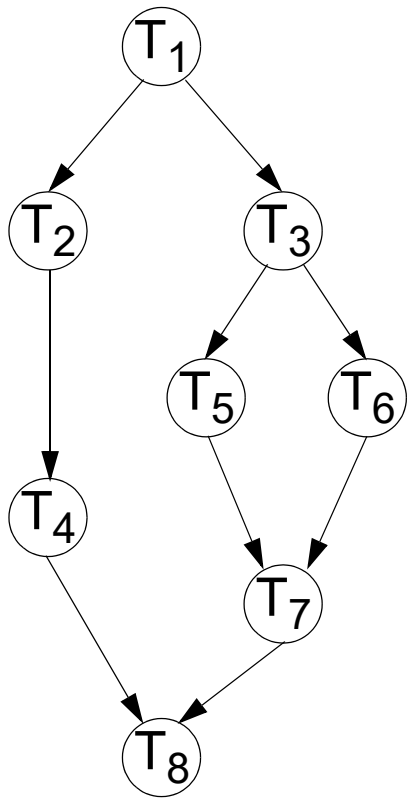
We look after a μ processor which is fast enough: $\mu p2$

For each task the WCET, when run on $\mu p2$, is estimated.



| Task | WCET |
|----------------|------|
| T ₁ | 2 |
| T ₂ | 3 |
| T ₃ | 2 |
| T ₄ | 3 |
| T ₅ | 4 |
| T ₆ | 6 |
| T ₇ | 3 |
| T ₈ | 5 |

Example



We look after a μ processor which is fast enough: $\mu p2$

For each task the WCET, when run on $\mu p2$, is estimated.

Using the architecture with μ processor $\mu p2$ we got a solution with:

❑ Execution time: $28 < 42$

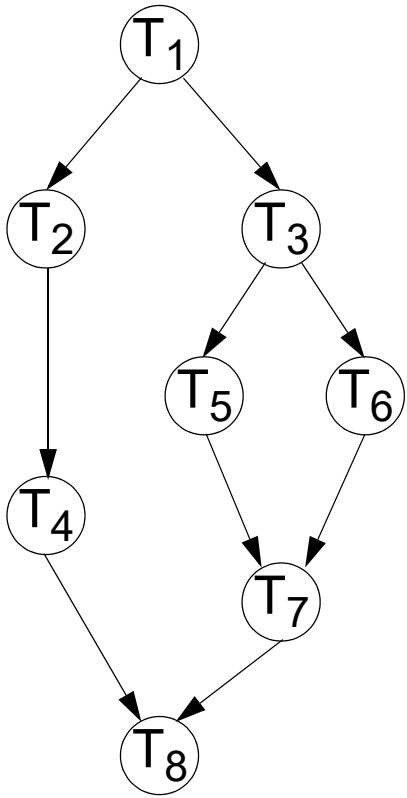
❑ Cost: $15 > 8$ 



We have to try with another architecture!

| Task | WCET |
|----------------|------|
| T ₁ | 2 |
| T ₂ | 3 |
| T ₃ | 2 |
| T ₄ | 3 |
| T ₅ | 4 |
| T ₆ | 6 |
| T ₇ | 3 |
| T ₈ | 5 |

Example



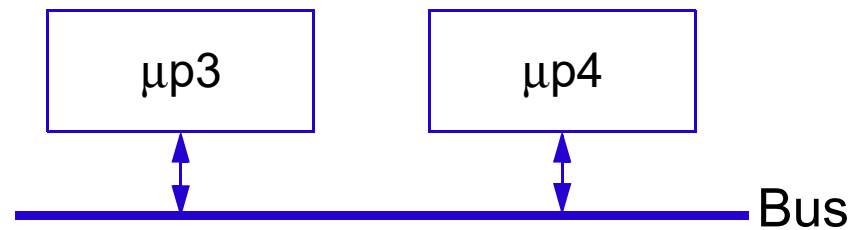
We have to look for a multiprocessor solution

□ In order to meet cost constraints try 2 cheap (and slow) μ ps:

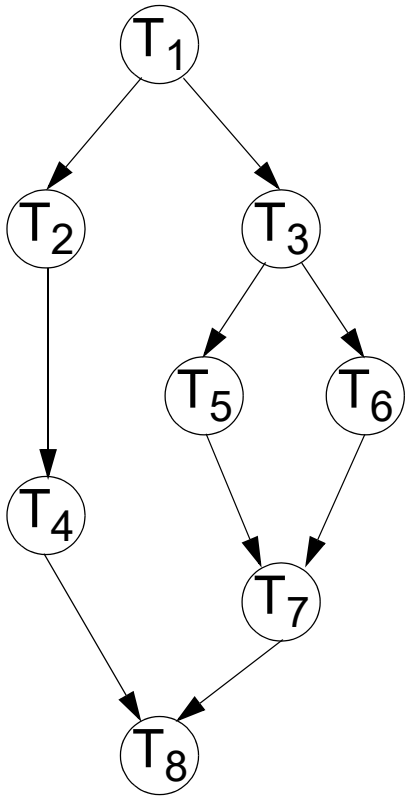
μ p3: cost 3

μ p4: cost 2

interconnection bus: cost 1

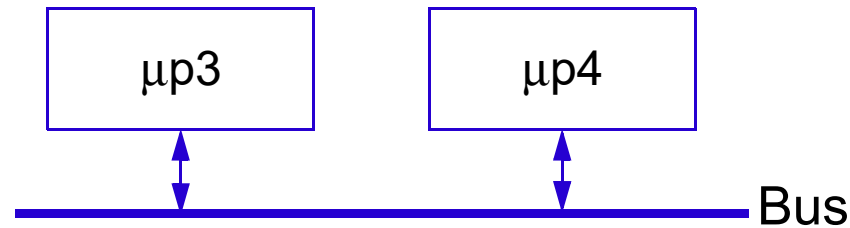


Example



We have to look for a multiprocessor solution

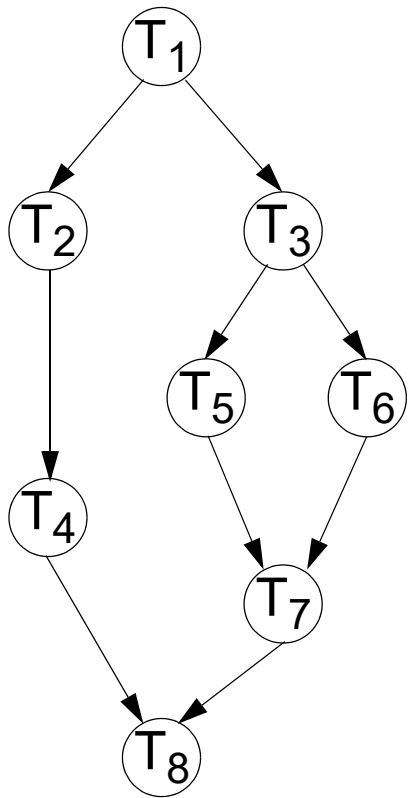
- In order to meet cost constraints try 2 cheap (and slow) μ ps:
 μ p3: cost 3
 μ p4: cost 2
interconnection bus: cost 1



For each task the WCET, when run on μ p3 and μ p4, is estimated.

| Task | WCET | |
|----------------|----------|----------|
| | μ p3 | μ p4 |
| T ₁ | 5 | 6 |
| T ₂ | 7 | 9 |
| T ₃ | 5 | 6 |
| T ₄ | 8 | 10 |
| T ₅ | 10 | 11 |
| T ₆ | 17 | 21 |
| T ₇ | 10 | 14 |
| T ₈ | 15 | 19 |

Example



Now we have to *map* the tasks to processors:

$\mu p3$: $T_1, T_3, T_5, T_6, T_7, T_8$.

$\mu p4$: T_2, T_4 .

If communicating tasks are mapped to different processors, they have to communicate over the bus.

Communication time has to be estimated; it depends on the amount of bits transferred between the tasks and on the speed of the bus.

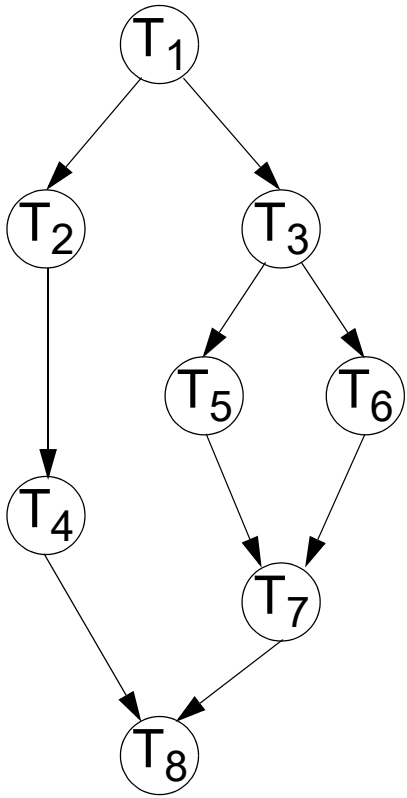
Estimated communication times:

C_{1-2} : 1

C_{4-8} : 1

| Task | WCET | |
|-------|----------|----------|
| | $\mu p3$ | $\mu p4$ |
| T_1 | 5 | 6 |
| T_2 | 7 | 9 |
| T_3 | 5 | 6 |
| T_4 | 8 | 10 |
| T_5 | 10 | 11 |
| T_6 | 17 | 21 |
| T_7 | 10 | 14 |
| T_8 | 15 | 19 |

Example



$\mu p3$: $T_1, T_3, T_5, T_6, T_7, T_8$.

$\mu p4$: T_2, T_4 .

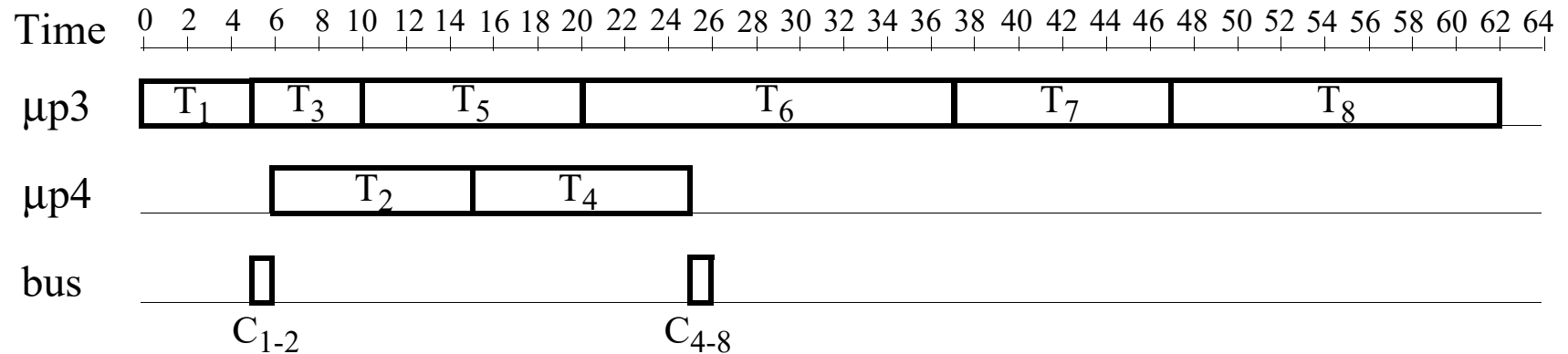
Estimated communication times:

C_{1-2} : 1

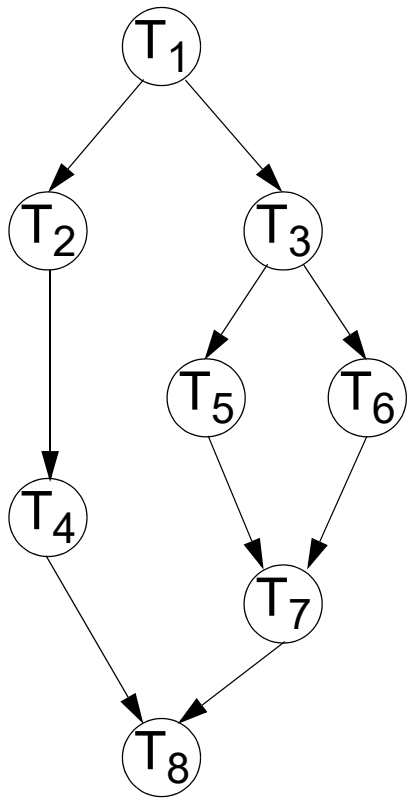
C_{4-8} : 1

We generate a schedule:

| Task | WCET | |
|-------|----------|----------|
| | $\mu p3$ | $\mu p4$ |
| T_1 | 5 | 6 |
| T_2 | 7 | 9 |
| T_3 | 5 | 6 |
| T_4 | 8 | 10 |
| T_5 | 10 | 11 |
| T_6 | 17 | 21 |
| T_7 | 10 | 14 |
| T_8 | 15 | 19 |



Example



$\mu p3$: $T_1, T_3, T_5, T_6, T_7, T_8$.

$\mu p4$: T_2, T_4 .

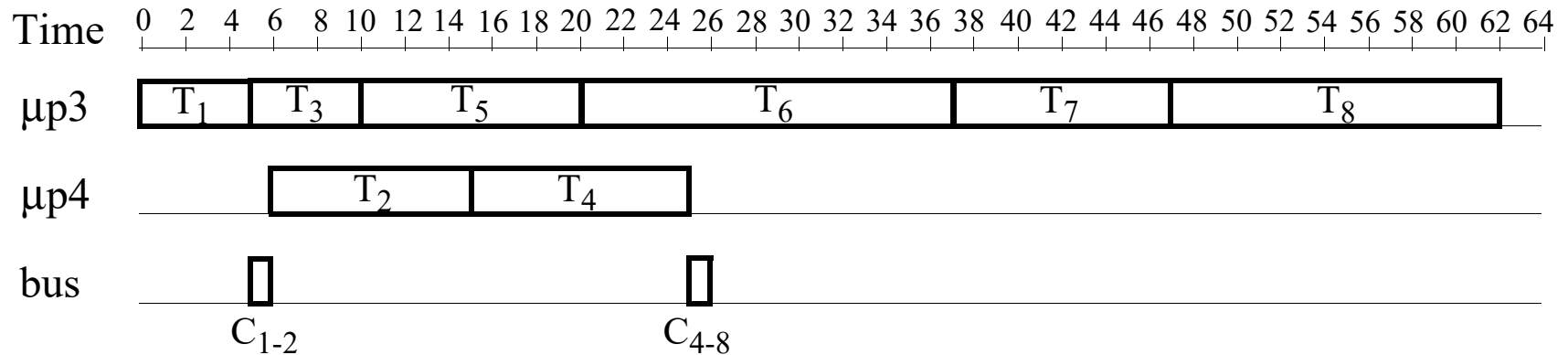
Estimated communication times:

C_{1-2} : 1

C_{4-8} : 1

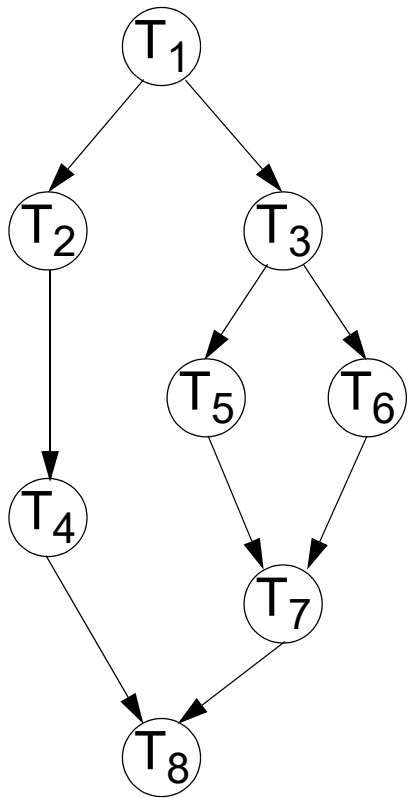
We generate a schedule:

| Task | WCET | |
|-------|----------|----------|
| | $\mu p3$ | $\mu p4$ |
| T_1 | 5 | 6 |
| T_2 | 7 | 9 |
| T_3 | 5 | 6 |
| T_4 | 8 | 10 |
| T_5 | 10 | 11 |
| T_6 | 17 | 21 |
| T_7 | 10 | 14 |
| T_8 | 15 | 19 |



We have exceeded the allowed execution time (42)!

Example



Try a new mapping; T_5 to $\mu p4$, in order to increase parallelism.

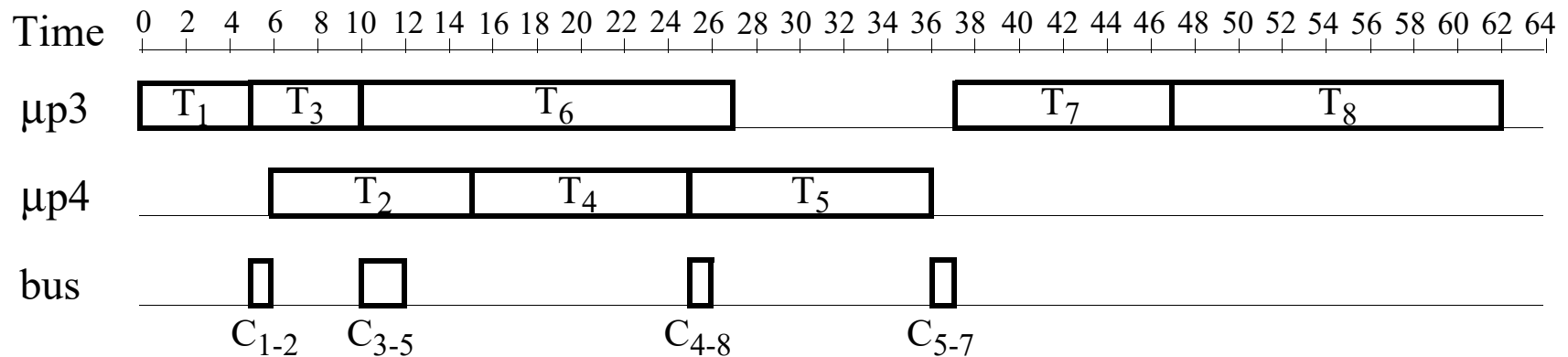
Two new communications are introduced, with estimated times:

C_{3-5} : 2

C_{5-7} : 1

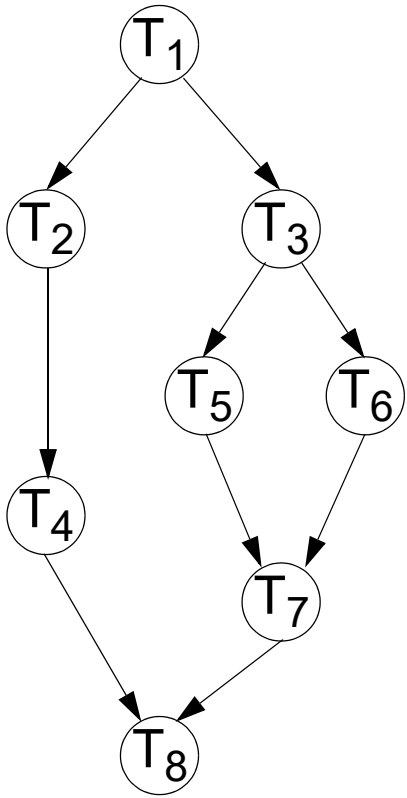
We generate a schedule:

| Task | WCET | |
|-------|----------|----------|
| | $\mu p3$ | $\mu p4$ |
| T_1 | 5 | 6 |
| T_2 | 7 | 9 |
| T_3 | 5 | 6 |
| T_4 | 8 | 10 |
| T_5 | 10 | 11 |
| T_6 | 17 | 21 |
| T_7 | 10 | 14 |
| T_8 | 15 | 19 |



The execution time is still 62, as before!

Example

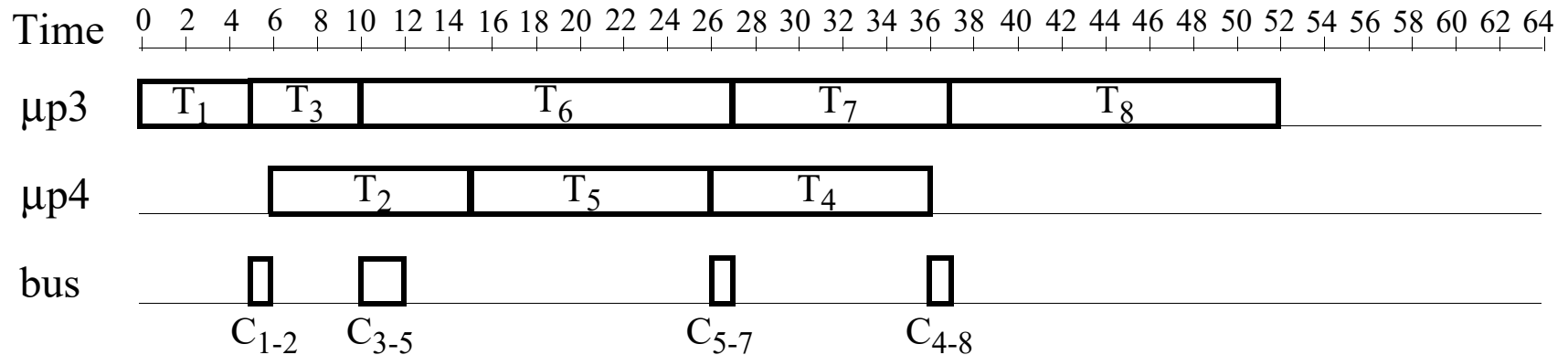


Try a new mapping; T_5 to $\mu p4$, in order to increase parallelism.
 Two new communications are introduced, with estimated times:

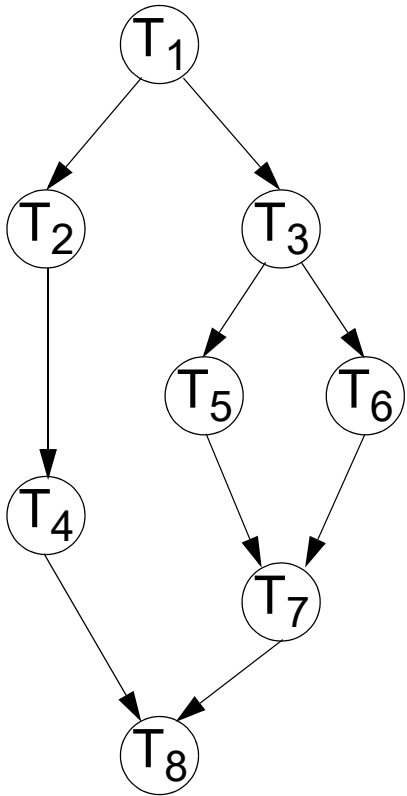
$C_{3-5}: 2$
 $C_{5-7}: 1$

There exists a better schedule!

| Task | WCET | |
|-------|----------|----------|
| | $\mu p3$ | $\mu p4$ |
| T_1 | 5 | 6 |
| T_2 | 7 | 9 |
| T_3 | 5 | 6 |
| T_4 | 8 | 10 |
| T_5 | 10 | 11 |
| T_6 | 17 | 21 |
| T_7 | 10 | 14 |
| T_8 | 15 | 19 |



Example

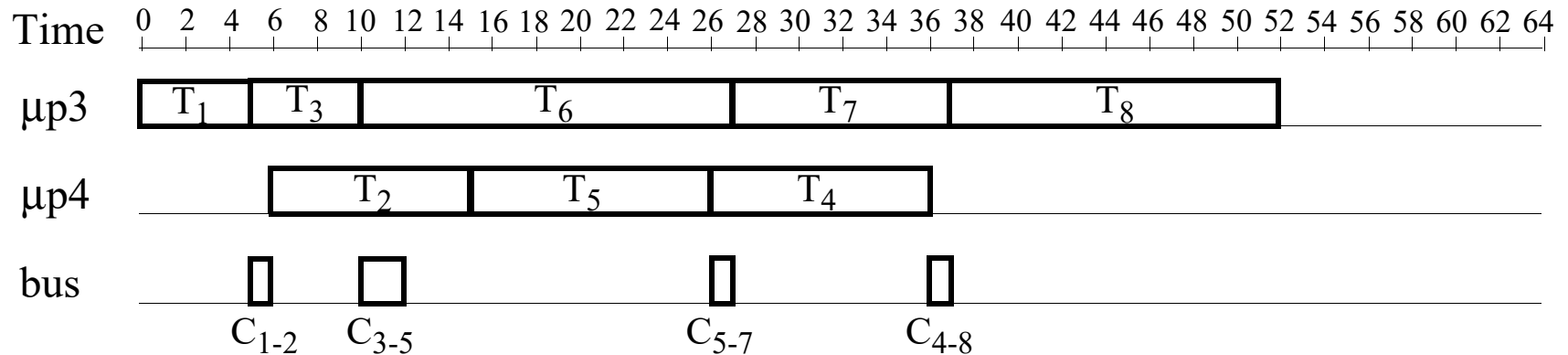


Try a new mapping; T_5 to $\mu p4$, in order to increase parallelism.
 Two new communications are introduced, with estimated times:

$C_{3-5}: 2$
 $C_{5-7}: 1$

There exists a better schedule!

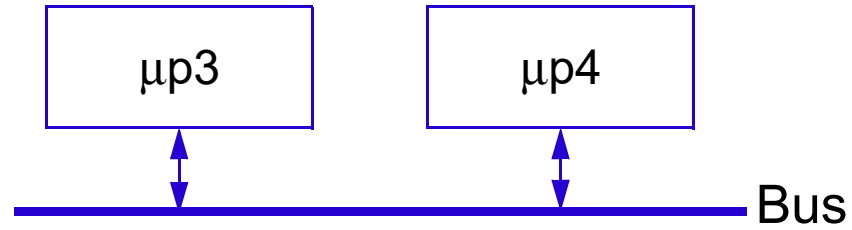
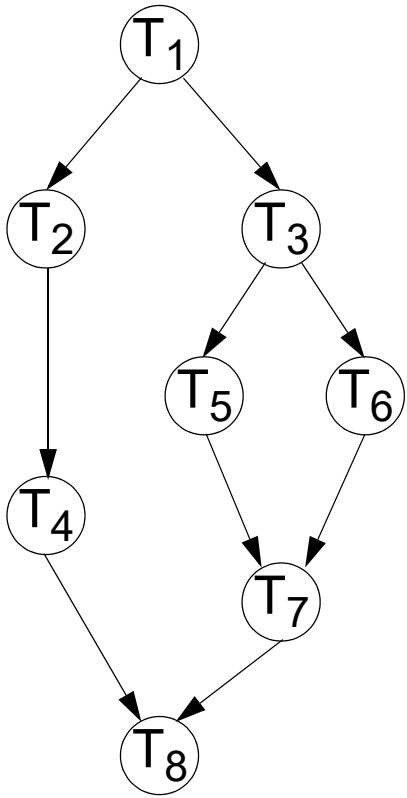
| Task | WCET | |
|-------|----------|----------|
| | $\mu p3$ | $\mu p4$ |
| T_1 | 5 | 6 |
| T_2 | 7 | 9 |
| T_3 | 5 | 6 |
| T_4 | 8 | 10 |
| T_5 | 10 | 11 |
| T_6 | 17 | 21 |
| T_7 | 10 | 14 |
| T_8 | 15 | 19 |



Execution time: $52 > 42$ 

Cost: $6 < 8$

Example

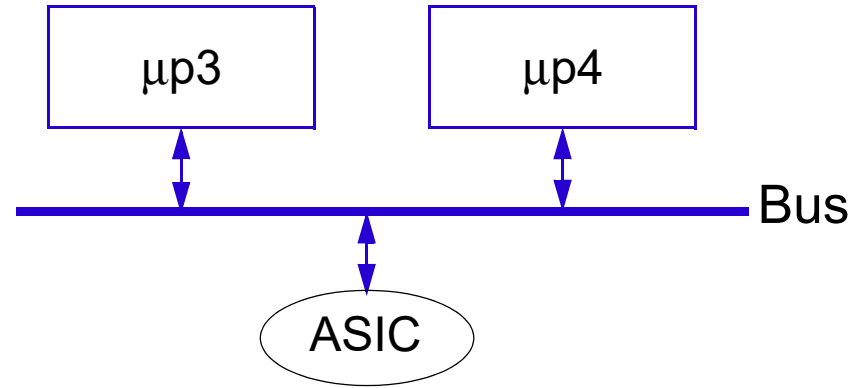


- Possible solutions:

- Change μ proc. μ p3 with faster one \Rightarrow cost limits exceeded

| Task | WCET | |
|----------------|----------|----------|
| | μ p3 | μ p4 |
| T ₁ | 5 | 6 |
| T ₂ | 7 | 9 |
| T ₃ | 5 | 6 |
| T ₄ | 8 | 10 |
| T ₅ | 10 | 11 |
| T ₆ | 17 | 21 |
| T ₇ | 10 | 14 |
| T ₈ | 15 | 19 |

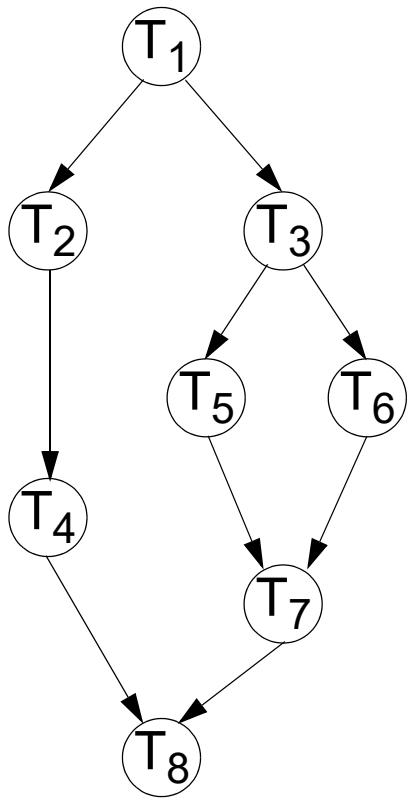
Example



■ Possible solutions:

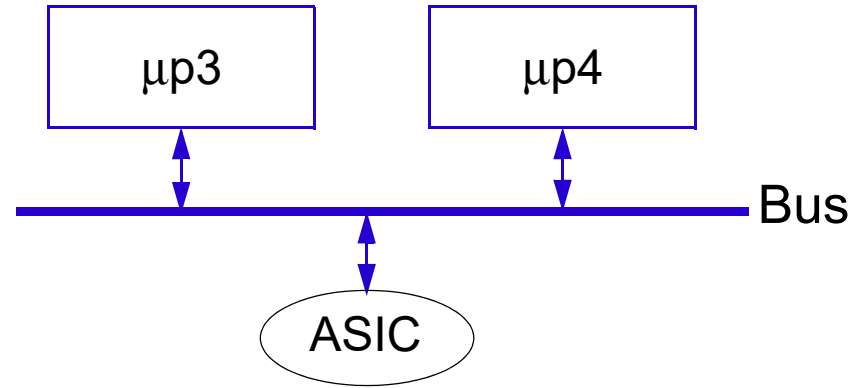
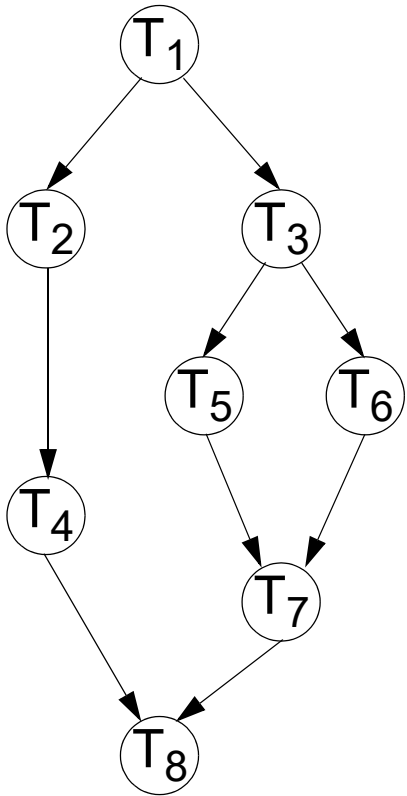
- Change μ proc. $\mu p3$ with faster one \Rightarrow cost limits exceeded
- Implement part of the functionality in hardware as an ASIC

Cost of ASIC: 1



| Task | WCET | |
|----------------|----------|----------|
| | $\mu p3$ | $\mu p4$ |
| T ₁ | 5 | 6 |
| T ₂ | 7 | 9 |
| T ₃ | 5 | 6 |
| T ₄ | 8 | 10 |
| T ₅ | 10 | 11 |
| T ₆ | 17 | 21 |
| T ₇ | 10 | 14 |
| T ₈ | 15 | 19 |

Example



- Possible solutions:

- Change μ proc. μ p3 with faster one \Rightarrow cost limits exceeded
- Implement part of the functionality in hardware as an ASIC

- New architecture

Cost of ASIC: 1

- Mapping

μ p3: T₁, T₃, T₆, T₇.

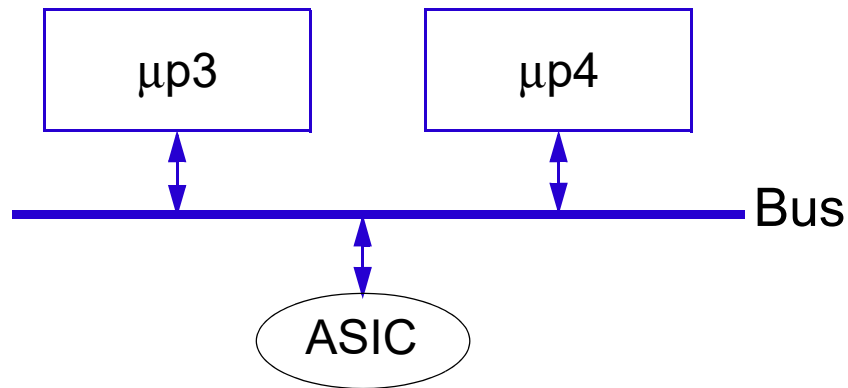
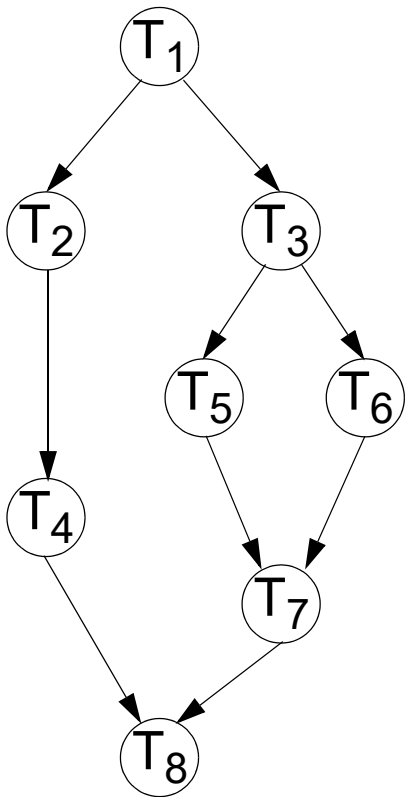
μ p4: T₂, T₄, T₅.

ASIC: T₈ with estimated WCET= 3

- New communication, with estimated time:
C₇₋₈: 1

| Task | WCET | |
|----------------|----------|----------|
| | μ p3 | μ p4 |
| T ₁ | 5 | 6 |
| T ₂ | 7 | 9 |
| T ₃ | 5 | 6 |
| T ₄ | 8 | 10 |
| T ₅ | 10 | 11 |
| T ₆ | 17 | 21 |
| T ₇ | 10 | 14 |
| T ₈ | 15 | 19 |

Example



■ Mapping

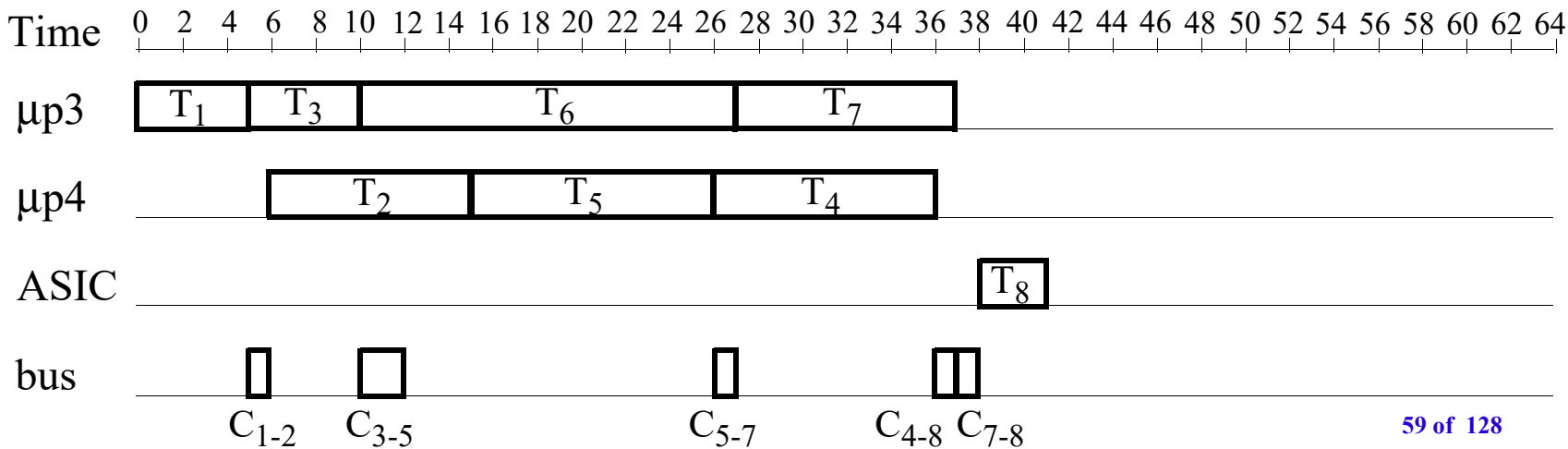
μp3: T₁, T₃, T₆, T₇.

μp4: T₂, T₄, T₅.

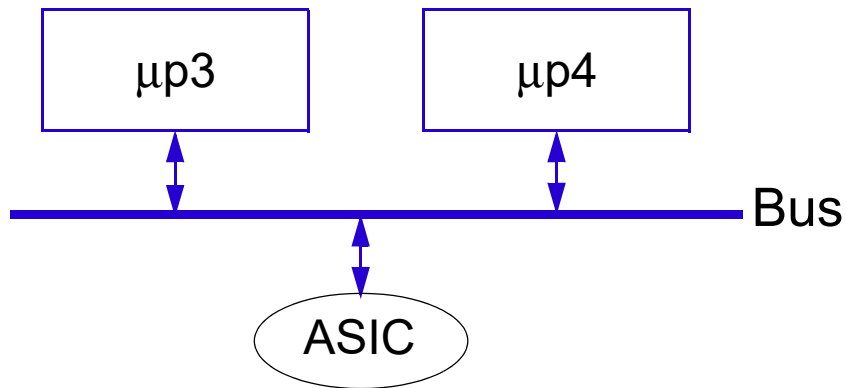
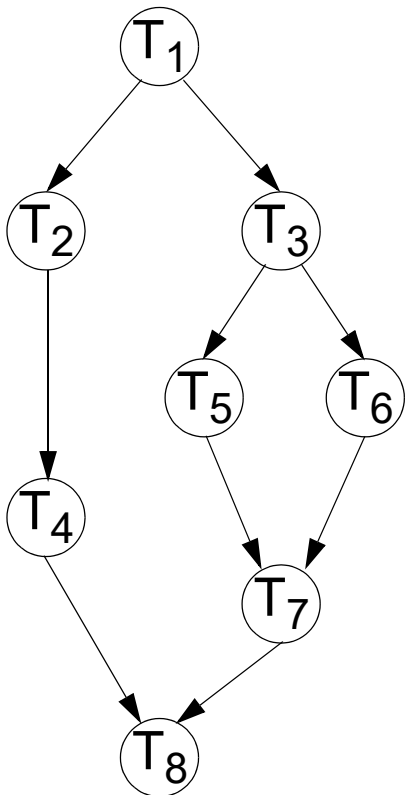
ASIC: T₈ with estimated WCET= 3

- New communication, with estimated time:
C₇₋₈: 1

| Task | WCET | |
|----------------|------|-----|
| | μp3 | μp4 |
| T ₁ | 5 | 6 |
| T ₂ | 7 | 9 |
| T ₃ | 5 | 6 |
| T ₄ | 8 | 10 |
| T ₅ | 10 | 11 |
| T ₆ | 17 | 21 |
| T ₇ | 10 | 14 |
| T ₈ | 15 | 19 |



Example

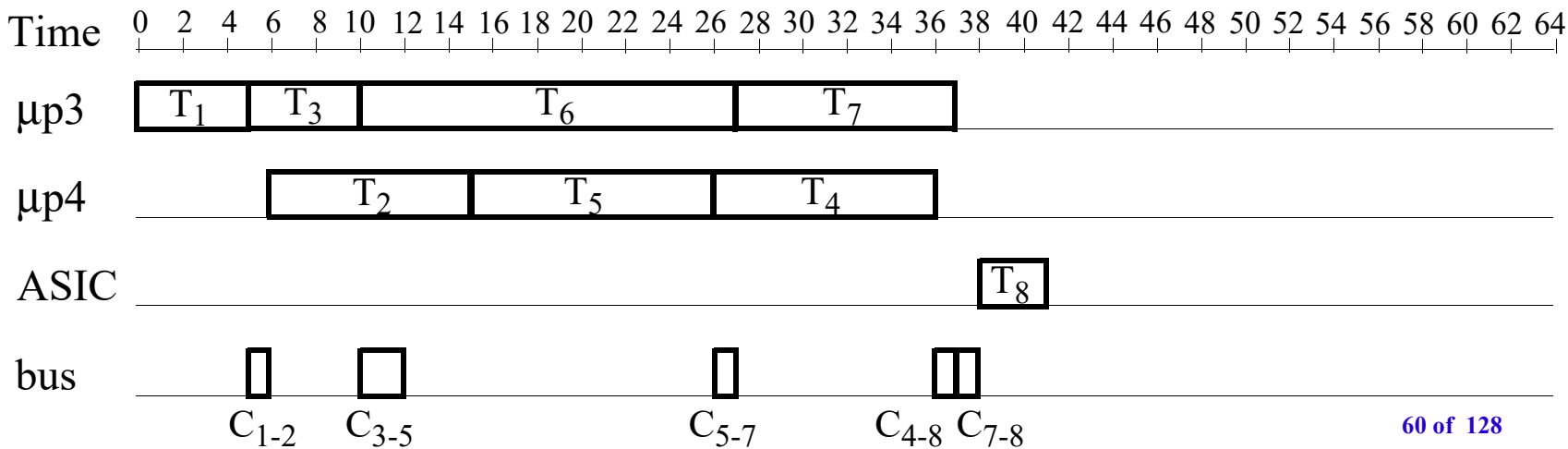


Using this architecture we got a solution with:

- Execution time: 41 < 42
- Cost: 7 < 8



| Task | WCET | |
|----------------|------|-----|
| | μp3 | μp4 |
| T ₁ | 5 | 6 |
| T ₂ | 7 | 9 |
| T ₃ | 5 | 6 |
| T ₄ | 8 | 10 |
| T ₅ | 10 | 11 |
| T ₆ | 17 | 21 |
| T ₇ | 10 | 14 |
| T ₈ | 15 | 19 |



Example

What did we achieve?

- ❑ We have selected an architecture.**
- ❑ We have mapped tasks to the processors and ASIC.**
- ❑ We have elaborated a a schedule.**

Example

What did we achieve?

- ❑ We have selected an architecture.
- ❑ We have mapped tasks to the processors and ASIC.
- ❑ We have elaborated a a schedule.

Extremely important!!!

Nothing has been built yet.

All decisions are based on simulation and estimation.

Example

What did we achieve?

- ❑ We have selected an architecture.
- ❑ We have mapped tasks to the processors and ASIC.
- ❑ We have elaborated a a schedule.

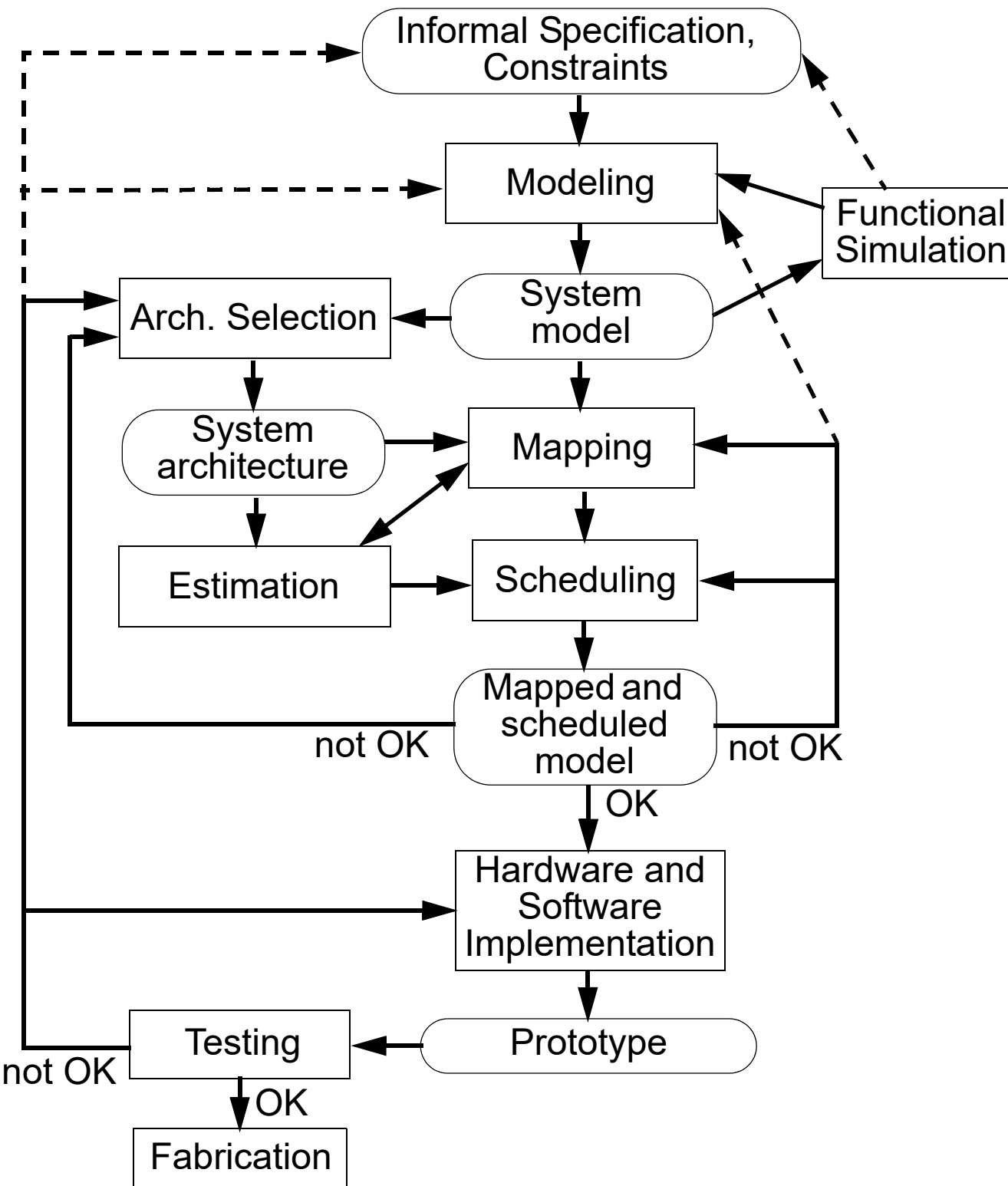
Extremely important!!!

Nothing has been built yet.

All decisions are based on simulation and estimation.

- Now we can go and do the software and hardware implementation, with a high degree of confidence that we get a correct prototype.

What is the essential difference compared to the “traditional” design flow?

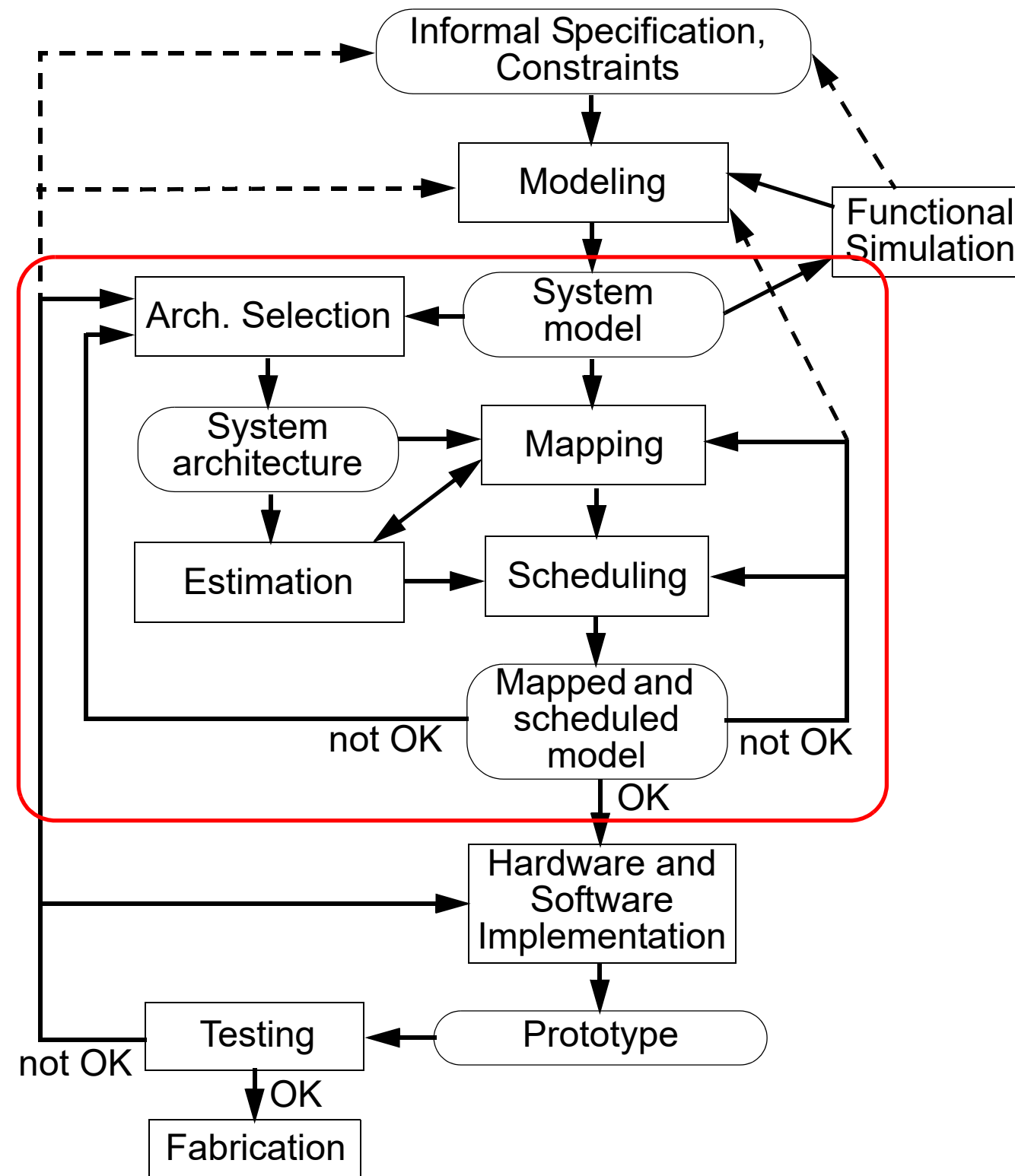


What is the essential difference compared to the “traditional” design flow?

□ The inner loop which is performed before the hardware/software implementation.

This loop is performed several times as part of the design space exploration. Different architectures, mappings and schedules are explored, before the actual implementation and prototyping.

□ We get highly optimized good quality solutions in short time. We have a good chance that the outer loop, including prototyping, is not repeated.



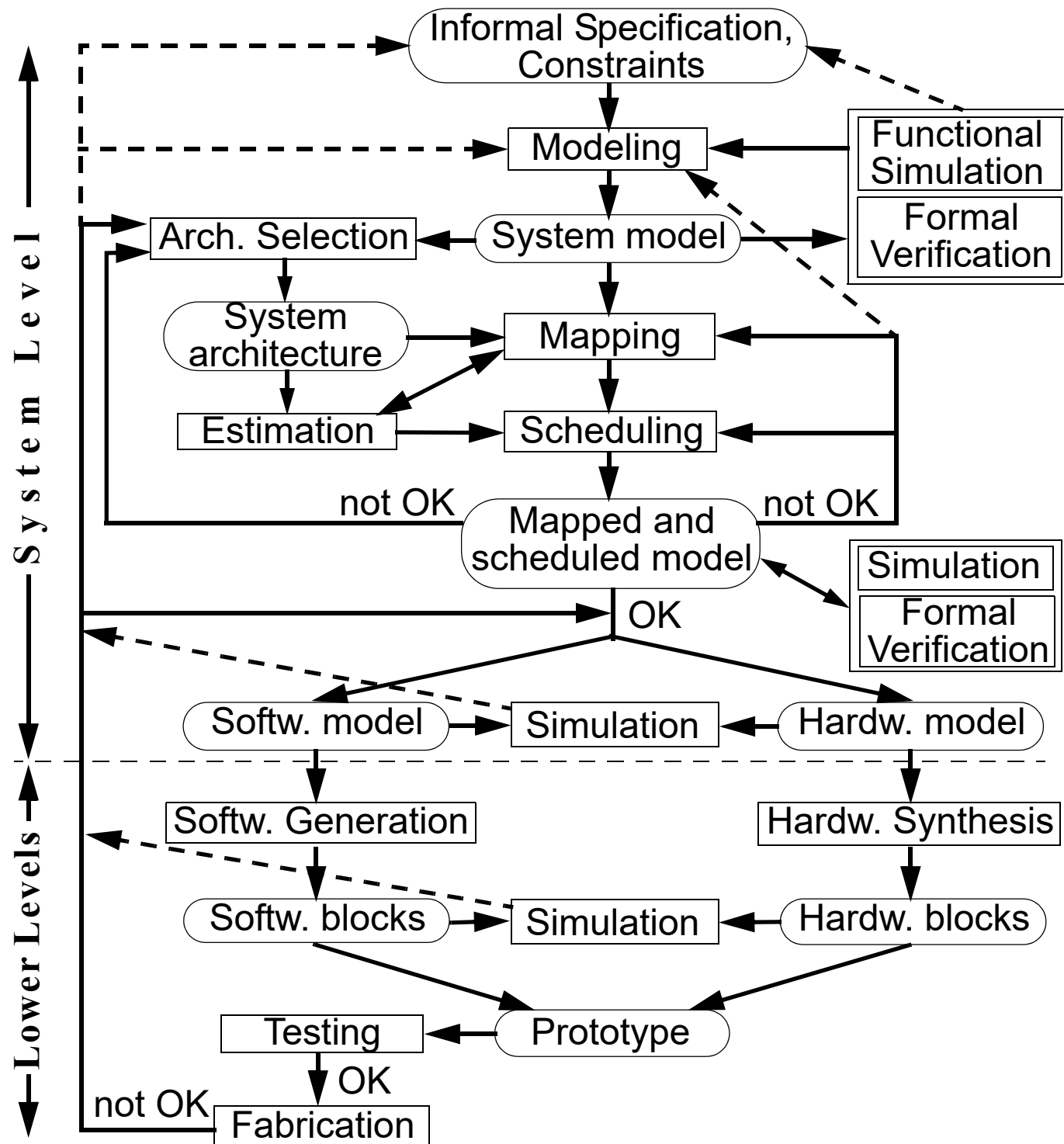
The Design Flow

■ Formal verification

- It is impossible to do an exhaustive verification by simulation!
Especially for safety critical systems formal verification is needed.

■ Hardware/Software codesign

- During the mapping/scheduling step we also decide what is going to be executed on a programmable processor (software) and what is going into hardware (ASIC, FPGA).
- During the implementation phase, hardware and software components have to be developed in a coordinated way, keeping care of their consistency (hardware/software cosimulation)



The “Lower Levels”

- Software generation:
 - Encoding in an implementation language (C, C++, assembler).
 - Compiling (this can include particular optimizations for application specific processors, DSPs, etc.).
 - Generation of a real-time kernel or adapting to an existing operating system.
 - Testing and debugging (in the development environment).

- Several courses are teaching this part: Programming related courses, Algorithms and data structures, Compilers, operating systems, real-time systems,

The “Lower Levels”

- Hardware synthesis:
 - Encoding in a hardware description language (VHDL, Verilog)
 - Successive synthesis steps: high-level, register-transfer level, logic-level synthesis.
 - Testing and debugging (by simulation)

- Several courses are teaching this part: Digital design, Electronics and VLSI related courses, Computer Architectures,

The System Level

- **TDTS07: System Design and Methodology (Modeling and Design of Embedded Systems)**

Bring Power Consumption into the Picture

Why is power consumption an issue?

- **Portable systems: battery life time!**
- **Systems with limited power budget: Mars Pathfinder, autonomous helicopter, ...**
- **Desktops and servers: high power consumption**
 - **raises temperature and deteriorates performance & reliability**
 - **increases the need for expensive cooling mechanisms**
- **One main difficulty with developing high performance chips is heat extraction.**
- **High power consumption has economical and ecological consequences.**

Sources of Power Dissipation in CMOS Devices

$$P = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW} + Q_{SC} \cdot V_{DD} \cdot f \cdot N_{SW} + I_{leak} \cdot V_{DD}$$

C = node capacitances
N_{SW} = switching activities
(number of gate transitions per clock cycle)
f = frequency of operation

V_{DD} = supply voltage
Q_{SC} = charge carried by short circuit current per transition
I_{leak} = leakage current

Sources of Power Dissipation in CMOS Devices

dynamic

$$P = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW} + Q_{SC} \cdot V_{DD} \cdot f \cdot N_{SW} + I_{leak} \cdot V_{DD}$$

C = node capacitances

N_{SW} = switching activities
(number of gate transitions per clock cycle)

f = frequency of operation

V_{DD} = supply voltage

Q_{SC} = charge carried by short circuit current per transition

I_{leak} = leakage current

Sources of Power Dissipation in CMOS Devices

$$P = \underbrace{\frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW}}_{\text{Switching power}} + \underbrace{Q_{SC} \cdot V_{DD} \cdot f \cdot N_{SW}}_{\text{Short-circ. power}} + I_{leak} \cdot V_{DD}$$

Switching power
Power required to charge/discharge circuit nodes

Short-circ. power
Dissipation due to short-circuit current

C = node capacitances
N_{SW} = switching activities
(number of gate transitions per clock cycle)
f = frequency of operation

V_{DD} = supply voltage
Q_{SC} = charge carried by short circuit current per transition
I_{leak} = leakage current

Sources of Power Dissipation in CMOS Devices

$$P = \underbrace{\frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW}}_{\text{dynamic}} + \underbrace{Q_{SC} \cdot V_{DD} \cdot f \cdot N_{SW}}_{\text{dynamic}} + \underbrace{I_{leak} \cdot V_{DD}}_{\text{static}}$$

| | | |
|---|---|---|
| <u>Switching power</u> Power required to charge/discharge circuit nodes | <u>Short-circ. power</u> Dissipation due to short-circuit current | <u>Leakage power</u> Dissipation due to leakage current |
|---|---|---|

C = node capacitances
N_{SW} = switching activities
(number of gate transitions per clock cycle)
f = frequency of operation

V_{DD} = supply voltage
Q_{SC} = charge carried by short circuit current per transition
I_{leak} = leakage current

Sources of Power Dissipation in CMOS Devices

$$P = \underbrace{\frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW}}_{\text{Switching power}} + \underbrace{Q_{SC} \cdot V_{DD} \cdot f \cdot N_{SW}}_{\text{Short-circ. power}} + \underbrace{I_{leak} \cdot V_{DD}}_{\text{Leakage power}}$$

Switching power
Power required to charge/discharge circuit nodes

Short-circ. power
Dissipation due to short-circuit current

Leakage power
Dissipation due to leakage current

- **Earlier:**
Leakage power has been considered negligible compared to dynamic.
- **Today:**
Total dissipation from leakage is approaching the total from dynamic.
- **As transistor sizes shrink:**
Leakage power becomes significant.

Sources of Power Dissipation in CMOS Devices

$$P = \underbrace{\frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW}}_{\text{dynamic}} + \underbrace{Q_{SC} \cdot V_{DD} \cdot f \cdot N_{SW}}_{\text{dynamic}} + \underbrace{I_{leak} \cdot V_{DD}}_{\text{static}}$$

| | | |
|---|---|---|
| <u>Switching power</u> Power required to charge/discharge circuit nodes | <u>Short-circ. power</u> Dissipation due to short-circuit current | <u>Leakage power</u> Dissipation due to leakage current |
|---|---|---|

- **Leakage power** is consumed even if the circuit is idle (standby). The only way to avoid is decoupling from power.

Sources of Power Dissipation in CMOS Devices

$$P = \underbrace{\frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW}}_{\text{dynamic}} + \underbrace{Q_{SC} \cdot V_{DD} \cdot f \cdot N_{SW}}_{\text{dynamic}} + \underbrace{I_{leak} \cdot V_{DD}}_{\text{static}}$$

| | | |
|---|---|---|
| <u>Switching power</u> Power required to charge/discharge circuit nodes | <u>Short-circ. power</u> Dissipation due to short-circuit current | <u>Leakage power</u> Dissipation due to leakage current |
|---|---|---|

- **Leakage power** is consumed even if the circuit is idle (standby). The only way to avoid is decoupling from power.
- **Short circuit power** is up to 10% of total.

Sources of Power Dissipation in CMOS Devices

$$P = \underbrace{\frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW}}_{\text{dynamic}} + \underbrace{Q_{SC} \cdot V_{DD} \cdot f \cdot N_{SW}}_{\text{dynamic}} + \underbrace{I_{leak} \cdot V_{DD}}_{\text{static}}$$

| | | |
|--|--|--|
| <u>Switching power</u> Power required to charge/discharge circuit nodes | <u>Short-circ. power</u> Dissipation due to short-circuit current | <u>Leakage power</u> Dissipation due to leakage current |
|--|--|--|

- Leakage power is consumed even if the circuit is idle (standby). The only way to avoid is decoupling from power.
- Short circuit power can be around 10% of total.
- Switching power is still the main source of power consumption.

Power and Energy Consumption

$$P = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW}$$

$$E = P \cdot t = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot N_{CY} \cdot N_{SW}$$

N_{CY} = number of cycles needed for the particular task.

Power and Energy Consumption

$$P = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW}$$

$$E = P \cdot t = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot N_{CY} \cdot N_{SW}$$

N_{CY} = number of cycles needed for the particular task.

- In certain situations we are concerned about power consumption:
 - heath dissipation, cooling:
 - physical deterioration due to temperature.
- Sometimes we want to reduce total energy consumed:
 - battery life.

Power and Energy Consumption

$$P = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW}$$

$$E = P \cdot t = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot N_{CY} \cdot N_{SW}$$

- Reducing power/energy consumption:
 - Reduce supply voltage

Power and Energy Consumption

$$P = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW}$$

$$E = P \cdot t = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot N_{CY} \cdot N_{SW}$$

- Reducing power/energy consumption:
 - Reduce supply voltage
 - Reduce switching activity

Power and Energy Consumption

$$P = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW}$$

$$E = P \cdot t = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot N_{CY} \cdot N_{SW}$$

- Reducing power/energy consumption:
 - Reduce supply voltage
 - Reduce switching activity
 - Reduce capacitance

Power and Energy Consumption

$$P = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f \cdot N_{SW}$$

$$E = P \cdot t = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot N_{CY} \cdot N_{SW}$$

■ Reducing power/energy consumption:

- Reduce supply voltage
- Reduce switching activity
- Reduce capacitance
- Reduce number of cycles

System Level Power/Energy Optimization

- Dynamic techniques: applied at run time.

These techniques are applied at run-time in order to reduce power consumption by exploiting idle or low-workload periods.

- Static techniques: applied at design time.

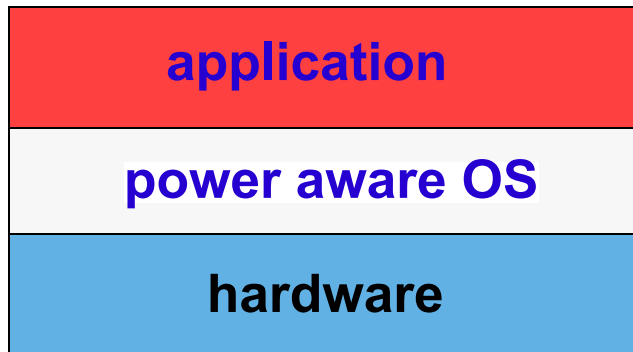
- Compilation for low power: instruction selection considering their power profile, data placement in memory, register allocation.
- Algorithm design: find the algorithm which is the most power-efficient.
- Task mapping and scheduling.

System Level Power/Energy Optimization

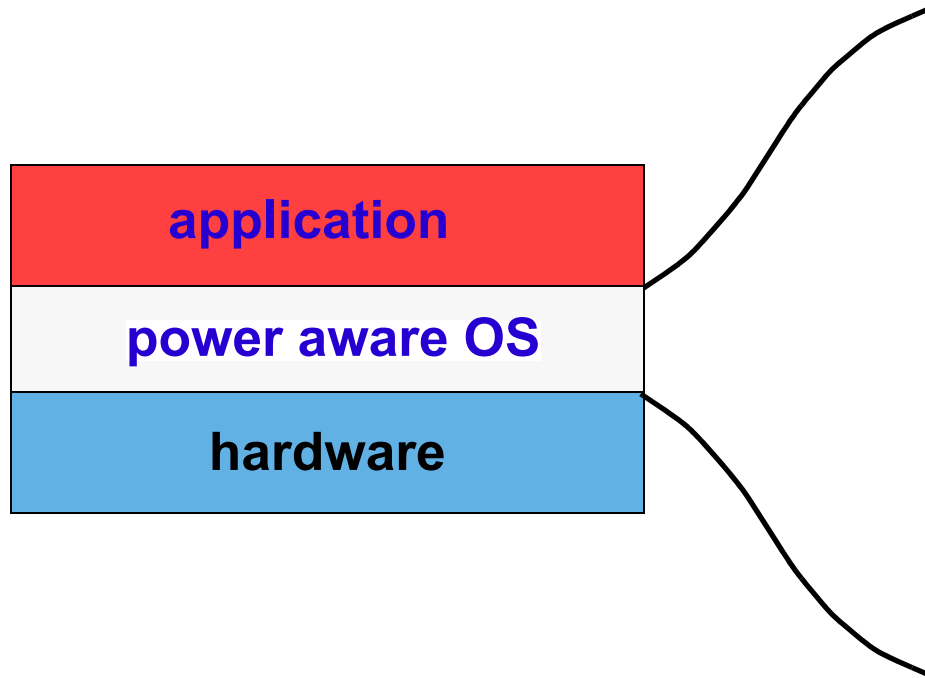
Three techniques will be discussed:

1. **Dynamic power management: a dynamic technique.**
2. **Task mapping: a static technique.**
3. **Task scheduling with dynamic power scaling: static & dynamic.**

Dynamic Power Management (DPM)



Dynamic Power Management (DPM)

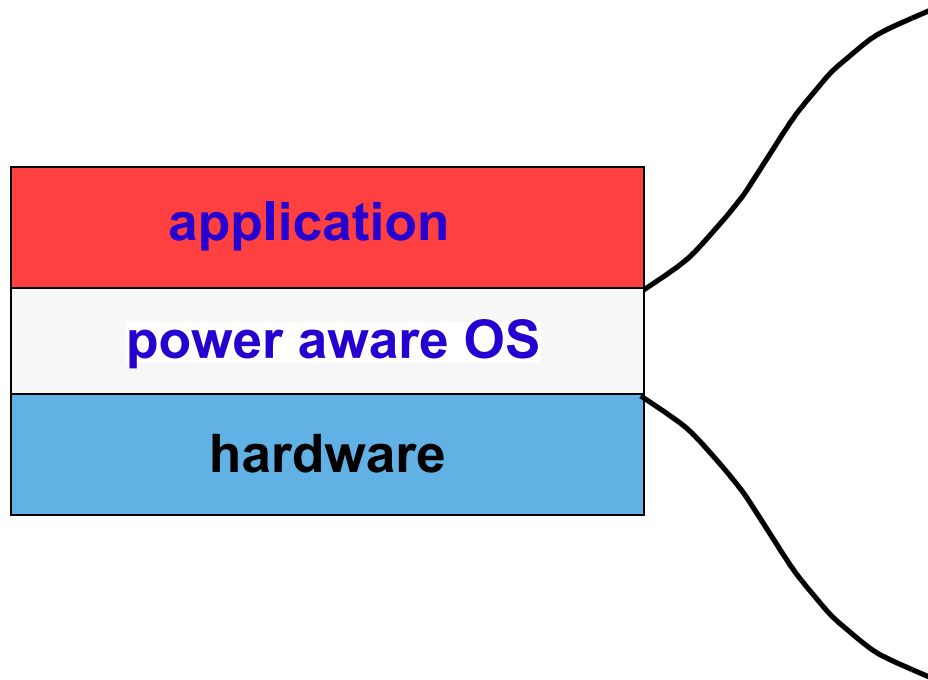


Decisions:

- Switching among multiple power states:
 - idle
 - sleep
 - run

- Switching among multiple frequencies and voltage levels.

Dynamic Power Management (DPM)



Decisions:

- Switching among multiple power states:
 - idle
 - sleep
 - run
- Switching among multiple frequencies and voltage levels.

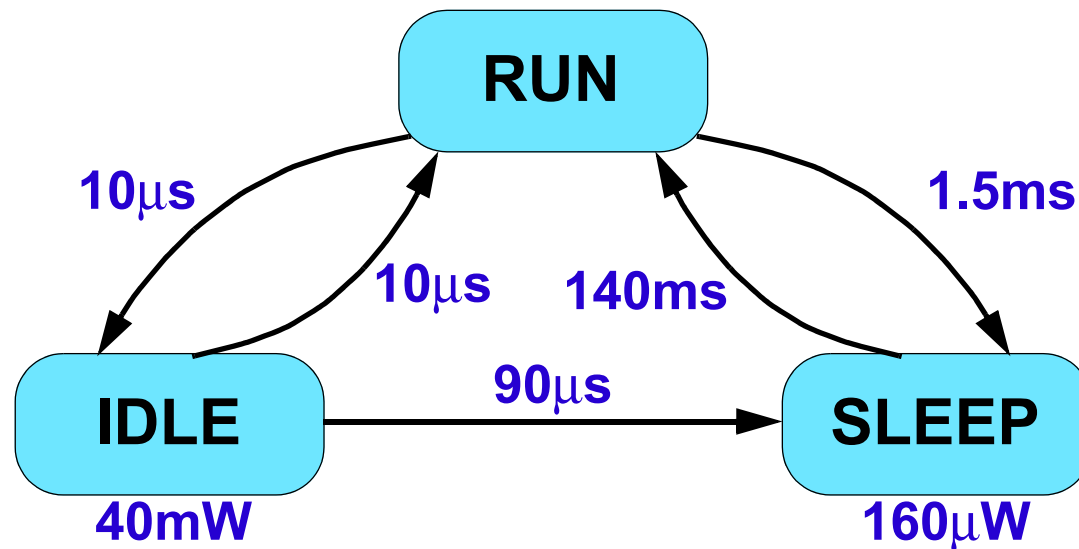
Goal:

- Energy optimization
- QoS constraints satisfied

Dynamic Power Management (DPM)

Intel Xscale Processor

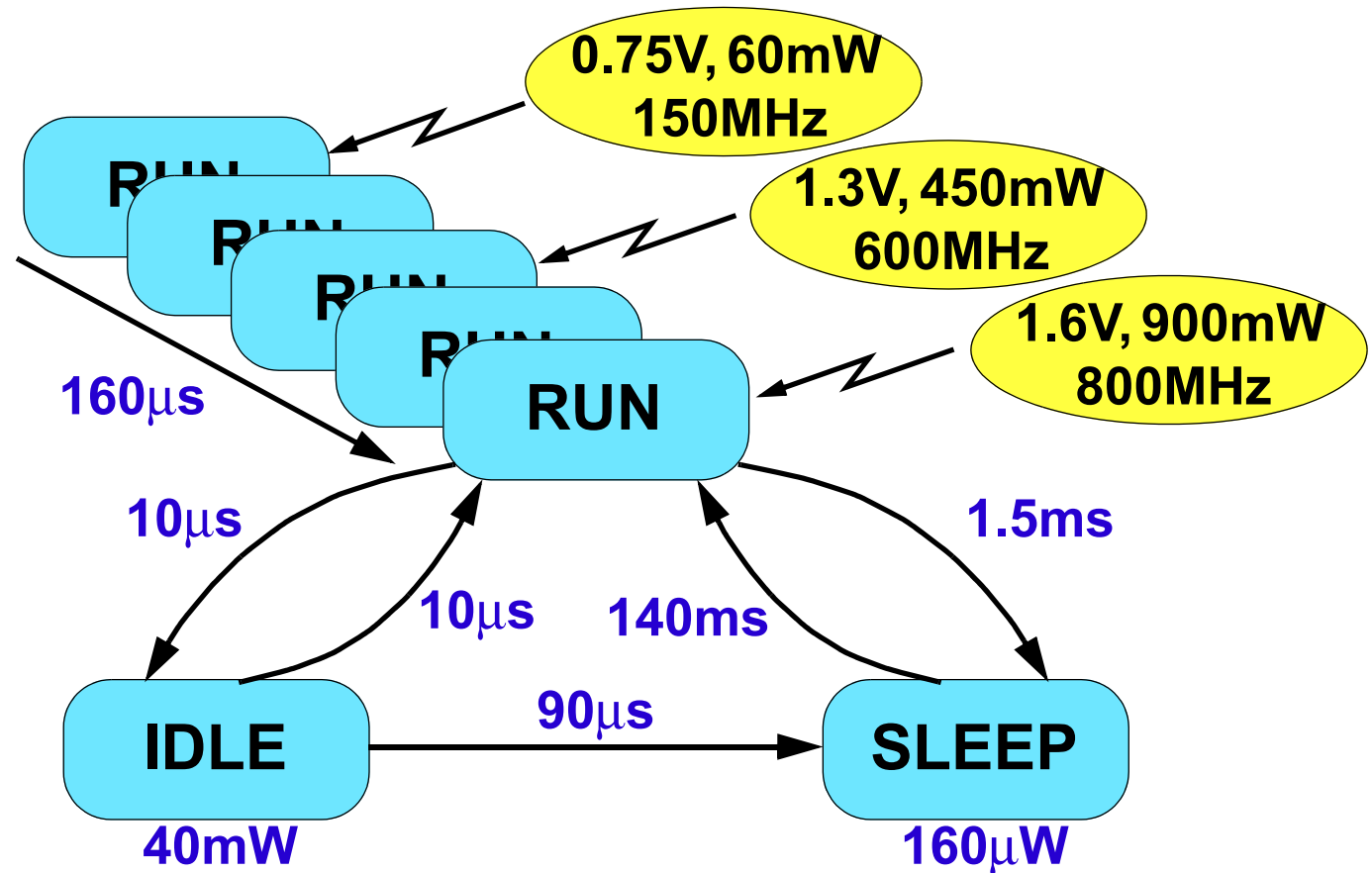
- **RUN:** operational
- **IDLE:** Clocks to the CPU are disabled; recovery is through interrupt.
- **SLEEP:** Mainly powered off; recovery through wake-up event.
- Other intermediate states: **DEEP IDLE**, **STANDBY**, **DEEP SLEEP**



Dynamic Power Management (DPM)

Intel Xscale Processor

- **RUN:** operational
- **IDLE:** Clocks to the CPU are disabled; recovery is through interrupt.
- **SLEEP:** Mainly powered off; recovery through wake-up event.
- Other intermediate states: **DEEP IDLE**, **STANDBY**, **DEEP SLEEP**

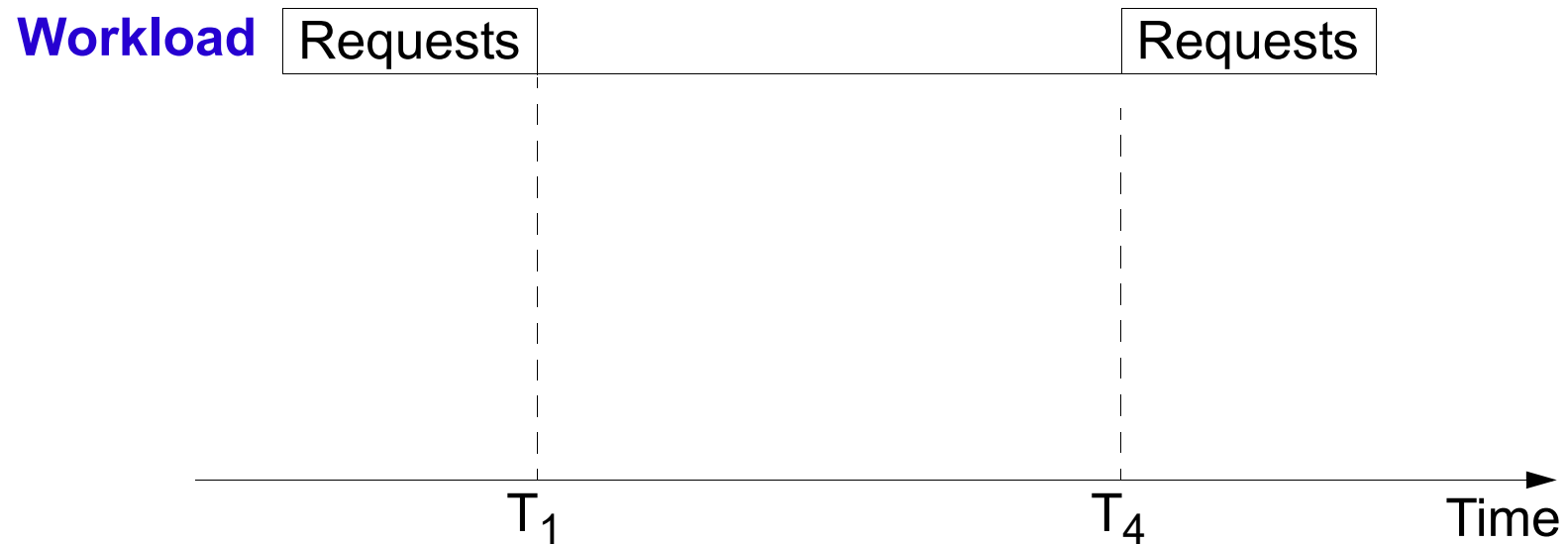


The Basic Concept of DPM

- When there are requests for a device \Rightarrow the device is *busy*; otherwise it is *idle*.
- When the device is idle, it can be shut down to enter a low-power sleeping state.

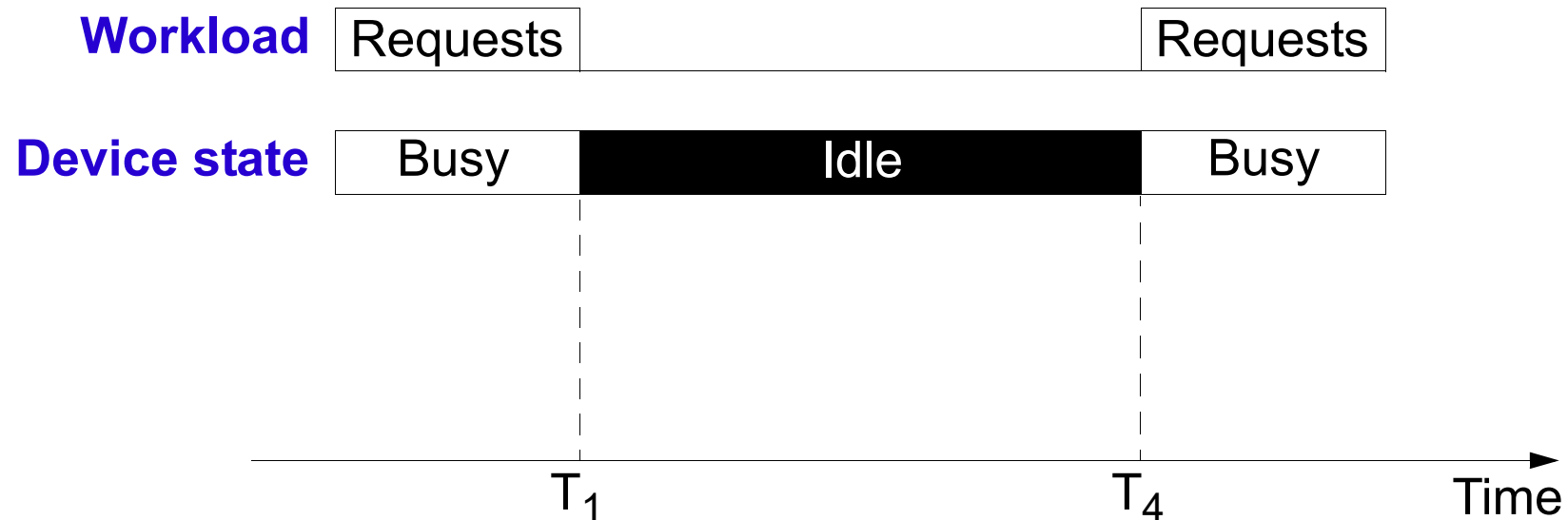
The Basic Concept of DPM

- When there are requests for a device \Rightarrow the device is *busy*; otherwise it is *idle*.
- When the device is idle, it can be shut down to enter a low-power sleeping state.



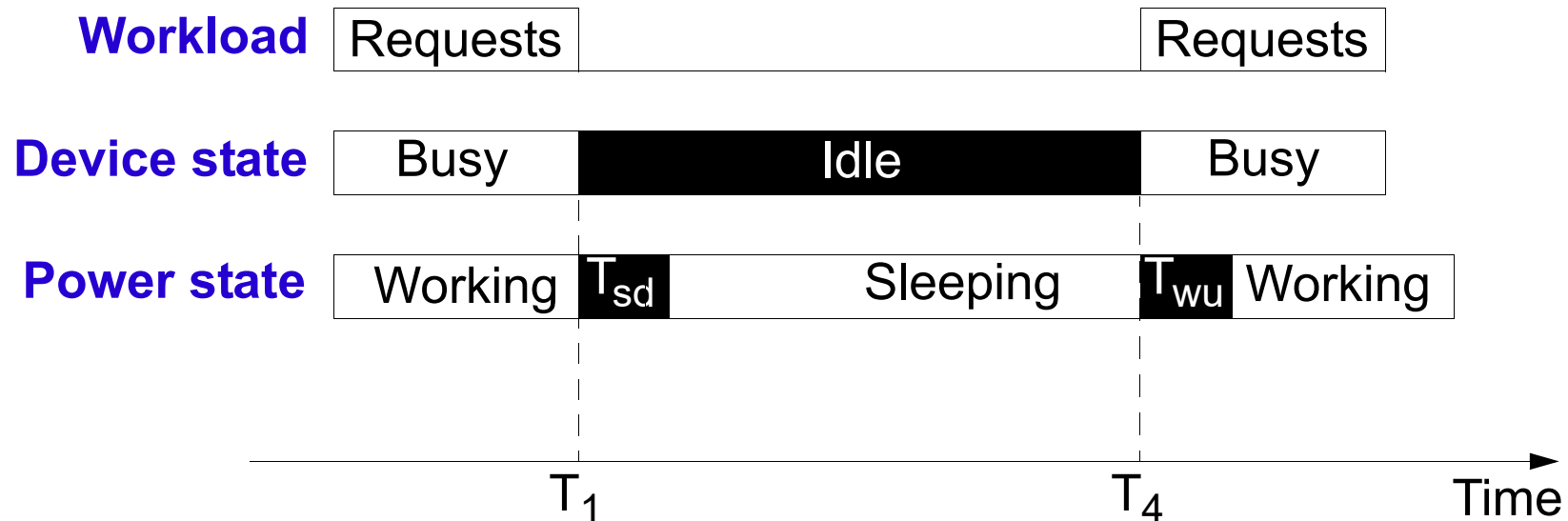
The Basic Concept of DPM

- When there are requests for a device \Rightarrow the device is *busy*; otherwise it is *idle*.
- When the device is idle, it can be shut down to enter a low-power sleeping state.



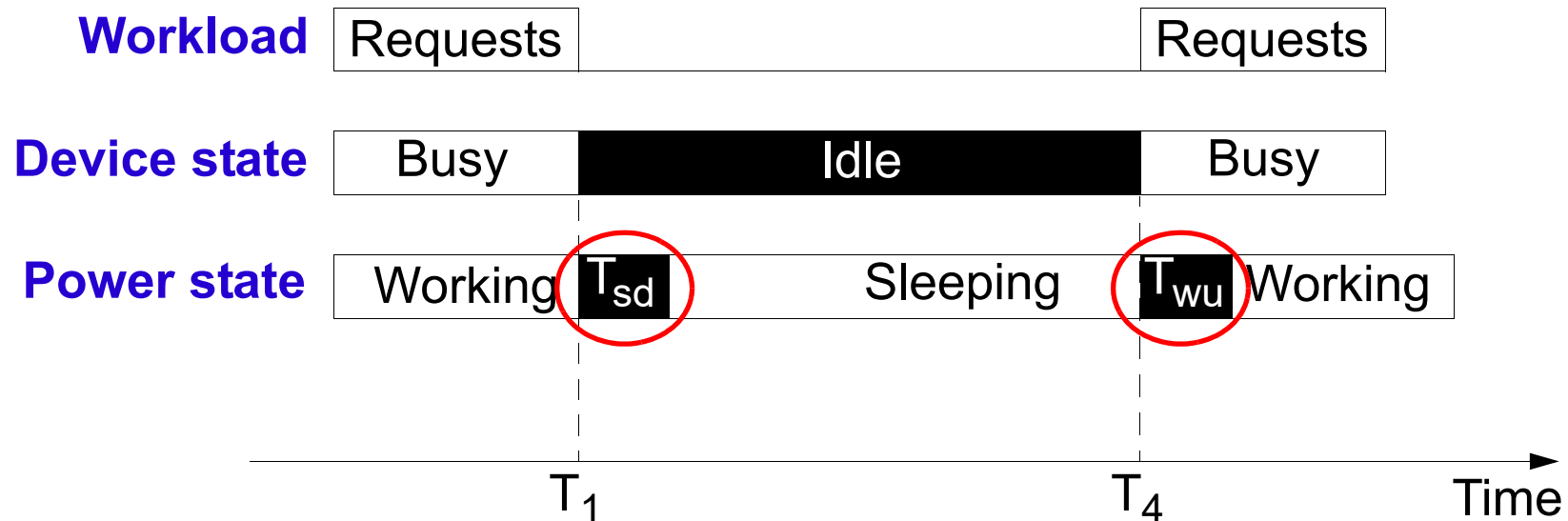
The Basic Concept of DPM

- When there are requests for a device \Rightarrow the device is *busy*; otherwise it is *idle*.
- When the device is idle, it can be shut down to enter a low-power sleeping state.



The Basic Concept of DPM

- When there are requests for a device \Rightarrow the device is *busy*; otherwise it is *idle*.
- When the device is idle, it can be shut down to enter a low-power sleeping state.



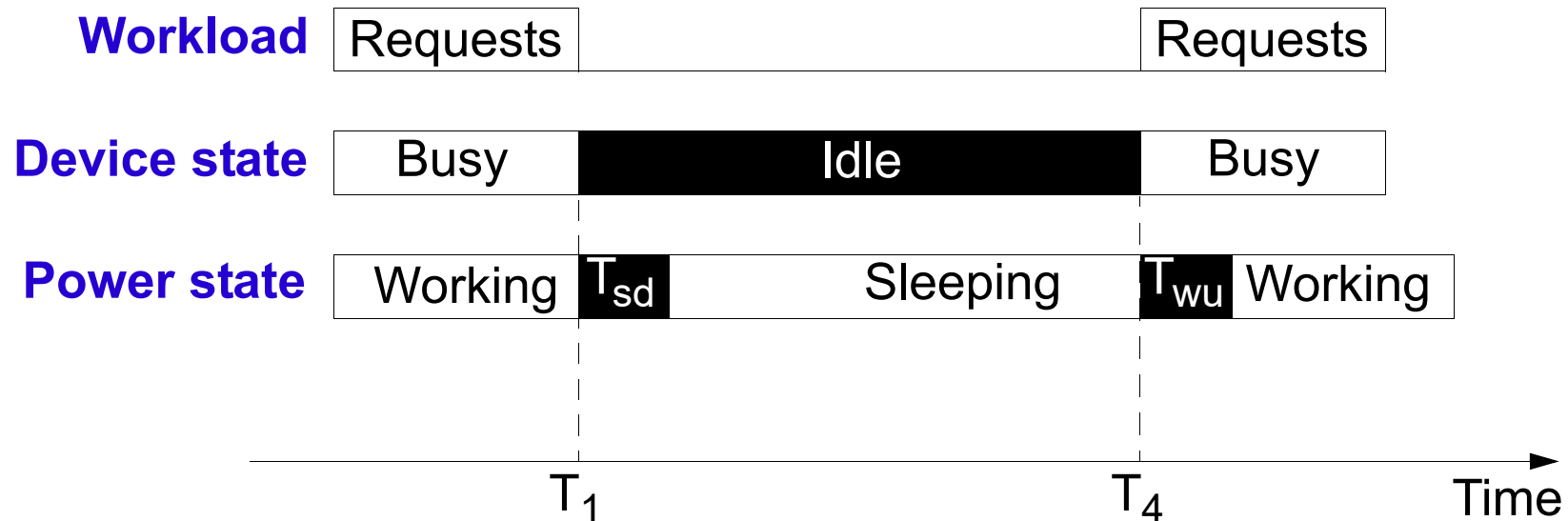
- Changing the power state takes *time and extra energy*.
 - T_{sd} : shutdown delay
 - T_{wu} : wake-up delay



Send the device to sleep only if the saved energy justifies the overhead!

The Basic Concept of DPM

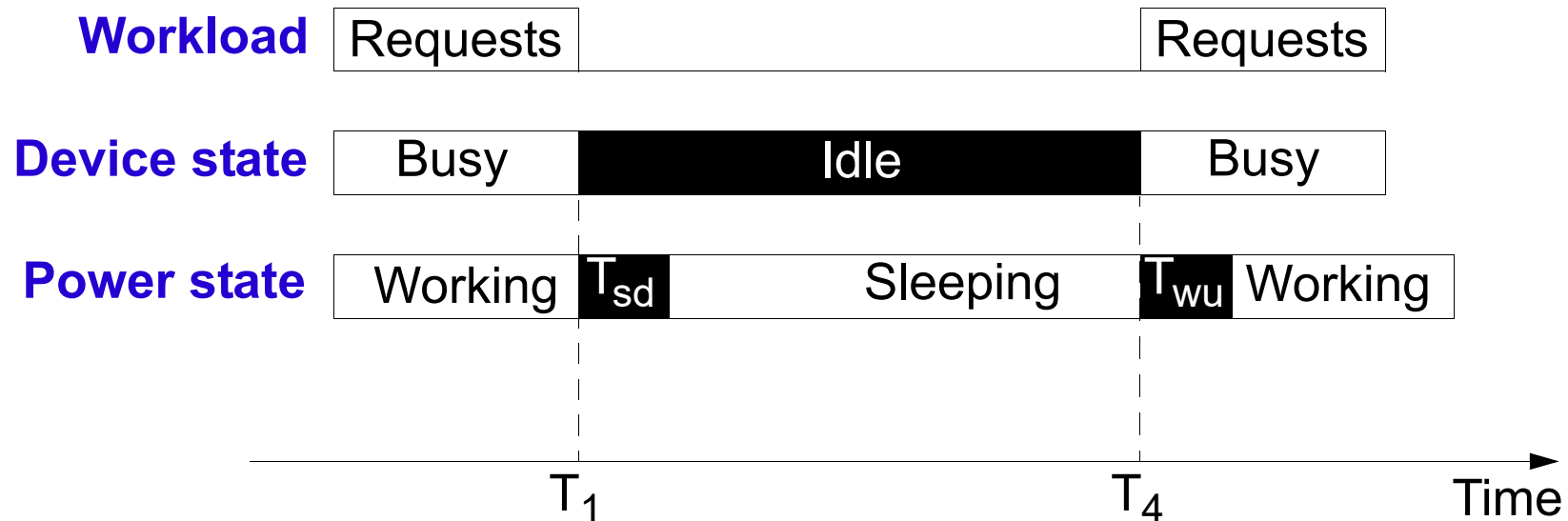
- When there are requests for a device \Rightarrow the device is *busy*; otherwise it is *idle*.
- When the device is idle, it can be shut down to enter a low-power sleeping state.



- The main Problems:
 - Don't shut down such that delays occur too frequently.
 - Don't shut down such that the savings due to the sleeping are smaller than the energy overhead of the state changes.

Power Management Policies

- When there are requests for a device \Rightarrow the device is *busy*; otherwise it is *idle*.
- When the device is idle, it can be shut down to enter a low-power sleeping state.

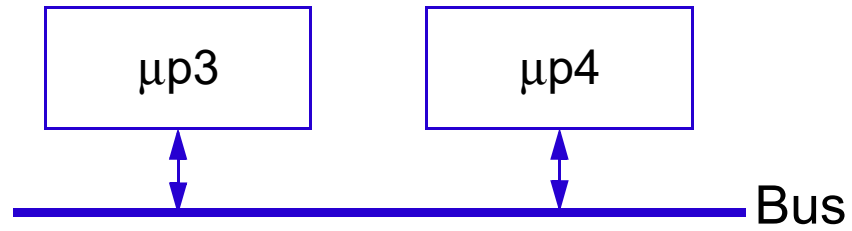
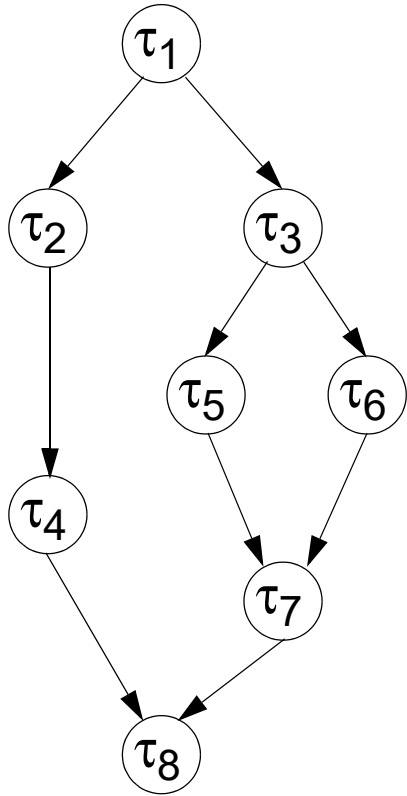


- Power management policies are concerned with predictions of idle periods:
 - For shut-down: try to predict how long the idle period will be in order to decide if a shut-down should be performed.
 - For wake-up: try to predict when the idle period ends, in order to avoid user delays due to T_{wu} . - Very difficult!

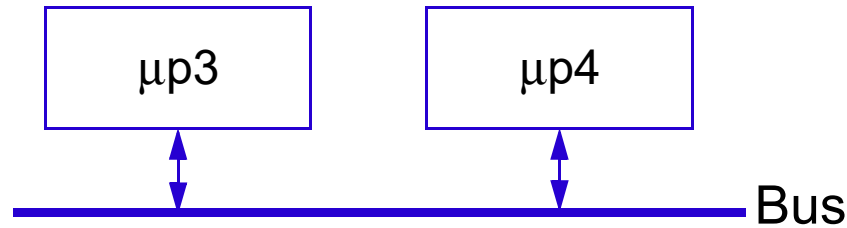
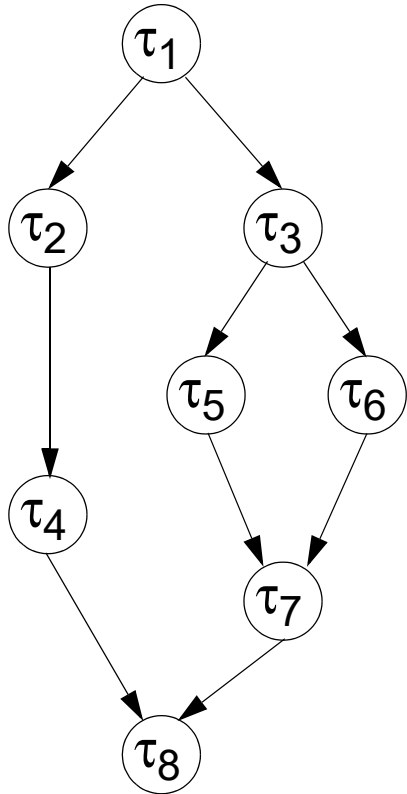
Dynamic Power Management (DPM)

- **For many embedded systems DPM techniques, like presented before, are not appropriate:**
 - **They have time constraints \Rightarrow we have to keep deadlines (usually we cannot afford shut-down and wake-up times).**
 - **The OS is simple&fast \Rightarrow no sophisticated run-time techniques.**
 - **The application is known at design time \Rightarrow we know a lot about the application and optimize already at design time.**

Mapping for Low Energy



Mapping for Low Energy



Consider a mapping:

$\mu p3$: $\tau_1, \tau_3, \tau_6, \tau_7, \tau_8$.

$\mu p4$: τ_2, τ_4, τ_5 .

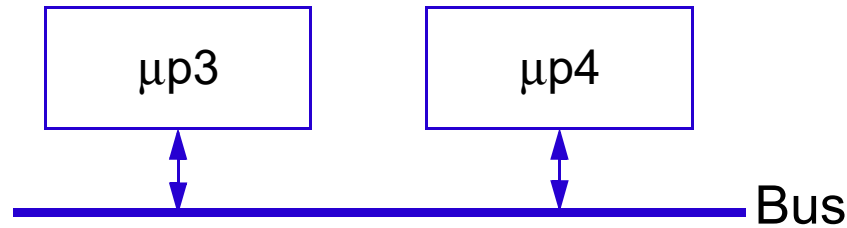
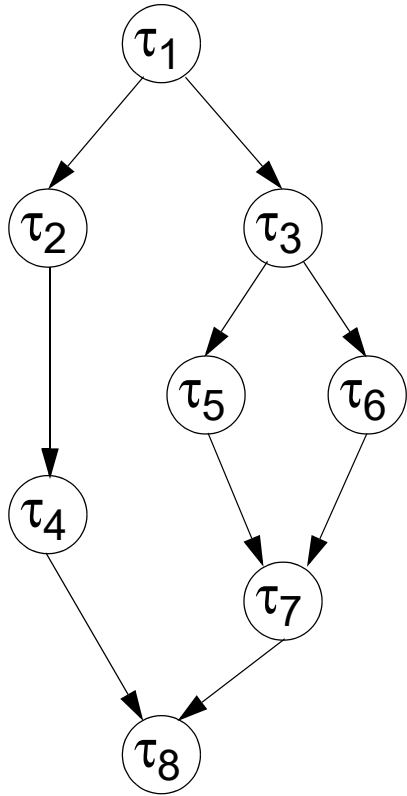
Communication times and energy:

C_{1-2} : $t = 1$; $E = 3$. C_{3-5} : $t = 2$; $E = 5$.

C_{4-8} : $t = 1$; $E = 3$. C_{5-7} : $t = 1$; $E = 3$.

| Task | WCET | | Energy | |
|----------|----------|----------|----------|----------|
| | $\mu p3$ | $\mu p4$ | $\mu p3$ | $\mu p4$ |
| τ_1 | 5 | 6 | 5 | 3 |
| τ_2 | 7 | 9 | 8 | 4 |
| τ_3 | 5 | 6 | 5 | 3 |
| τ_4 | 8 | 10 | 6 | 4 |
| τ_5 | 10 | 11 | 8 | 6 |
| τ_6 | 17 | 21 | 15 | 10 |
| τ_7 | 10 | 14 | 8 | 7 |
| τ_8 | 15 | 19 | 14 | 9 |

Mapping for Low Energy



Consider a mapping:

μp3: $\tau_1, \tau_3, \tau_6, \tau_7, \tau_8$.

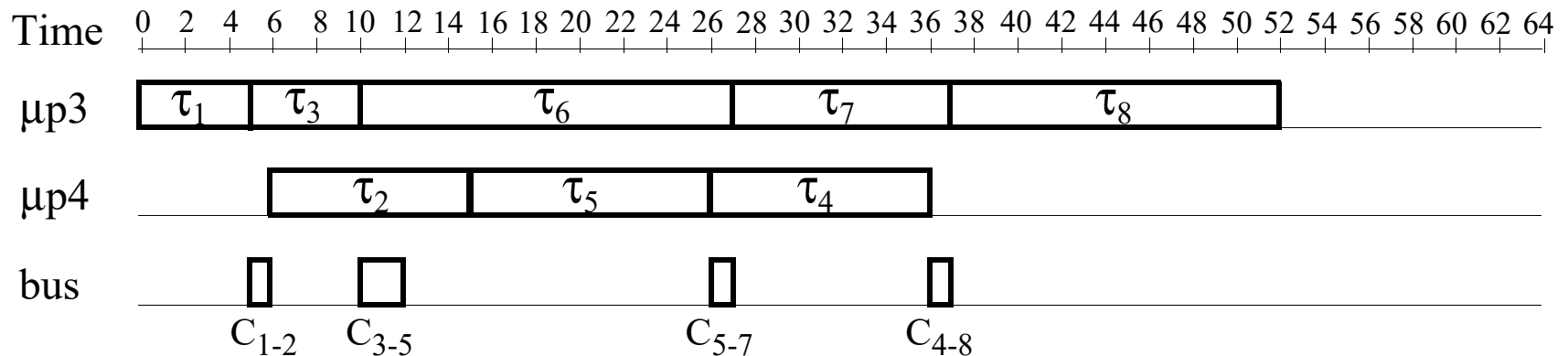
μp4: τ_2, τ_4, τ_5 .

Communication times and energy:

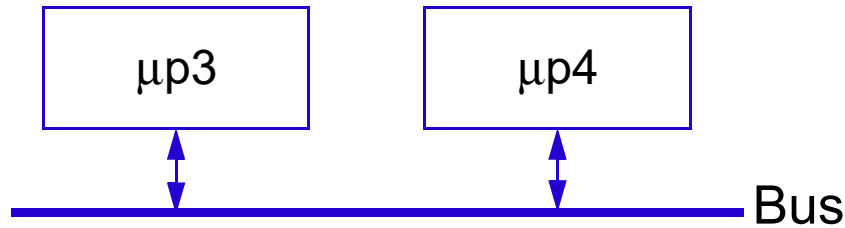
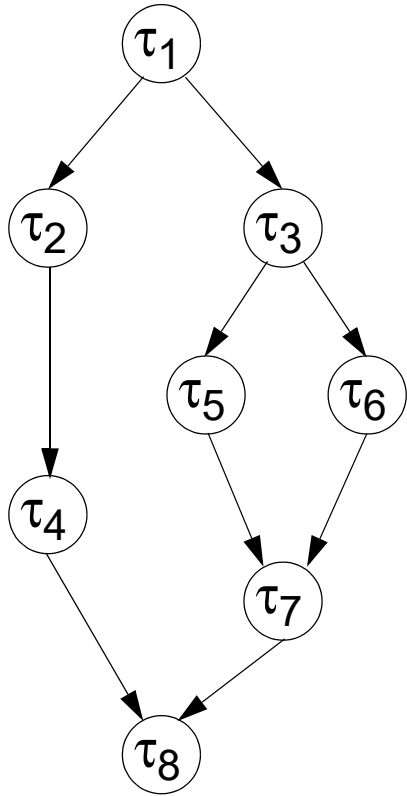
C₁₋₂: $t = 1; E = 3$. **C₃₋₅:** $t = 2; E = 5$.

C₄₋₈: $t = 1; E = 3$. **C₅₋₇:** $t = 1; E = 3$.

| Task | WCET | | Energy | |
|----------|------|-----|--------|-----|
| | μp3 | μp4 | μp3 | μp4 |
| τ_1 | 5 | 6 | 5 | 3 |
| τ_2 | 7 | 9 | 8 | 4 |
| τ_3 | 5 | 6 | 5 | 3 |
| τ_4 | 8 | 10 | 6 | 4 |
| τ_5 | 10 | 11 | 8 | 6 |
| τ_6 | 17 | 21 | 15 | 10 |
| τ_7 | 10 | 14 | 8 | 7 |
| τ_8 | 15 | 19 | 14 | 9 |



Mapping for Low Energy



Consider a mapping:

μp3: $\tau_1, \tau_3, \tau_6, \tau_7, \tau_8$.

μp4: τ_2, τ_4, τ_5 .

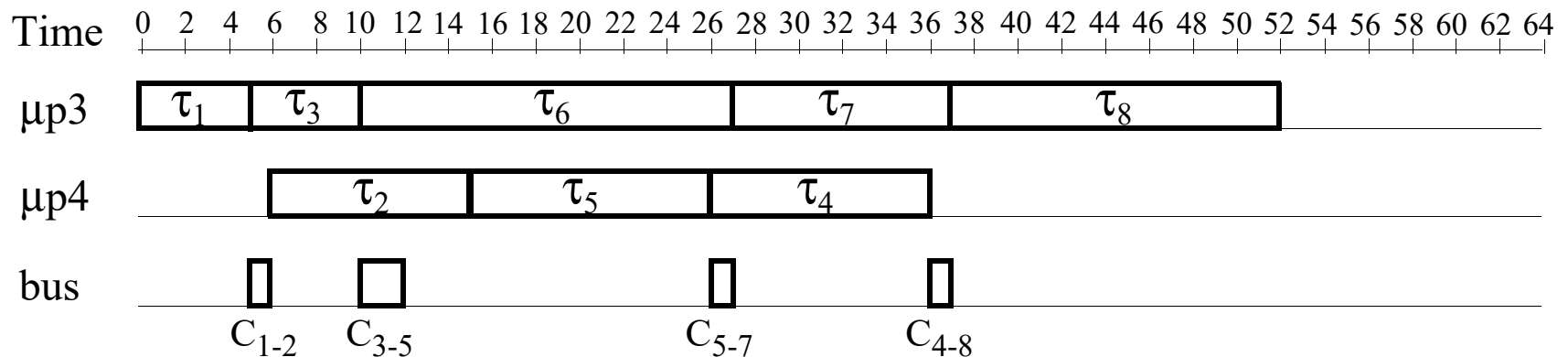
Communication times and energy:

C_{1-2} : $t = 1; E = 3$. **C_{3-5} :** $t = 2; E = 5$.

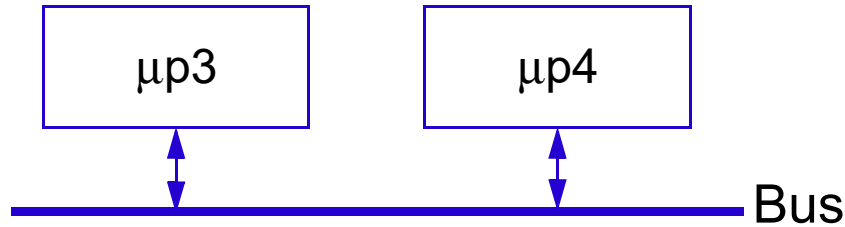
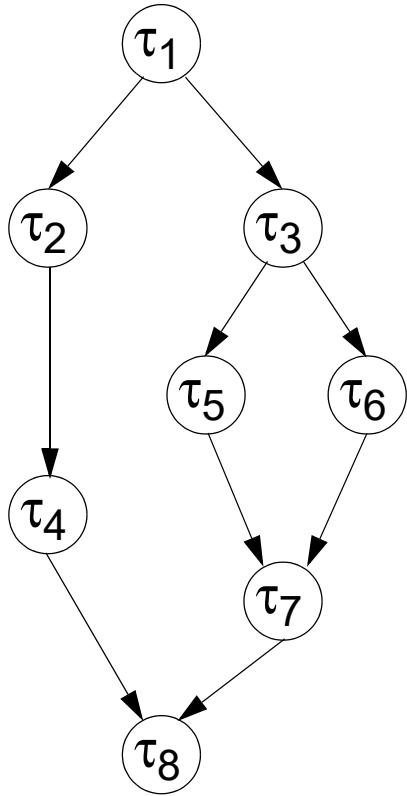
C_{4-8} : $t = 1; E = 3$. **C_{5-7} :** $t = 1; E = 3$.

| Task | WCET | | Energy | |
|----------|------|-----|--------|-----|
| | μp3 | μp4 | μp3 | μp4 |
| τ_1 | 5 | 6 | 5 | 3 |
| τ_2 | 7 | 9 | 8 | 4 |
| τ_3 | 5 | 6 | 5 | 3 |
| τ_4 | 8 | 10 | 6 | 4 |
| τ_5 | 10 | 11 | 8 | 6 |
| τ_6 | 17 | 21 | 15 | 10 |
| τ_7 | 10 | 14 | 8 | 7 |
| τ_8 | 15 | 19 | 14 | 9 |

Execution time: 52; Energy consumed: 75



Mapping for Low Energy



Consider another mapping:

$\mu p3$: $\tau_1, \tau_3, \tau_6, \tau_7, \tau_8$.

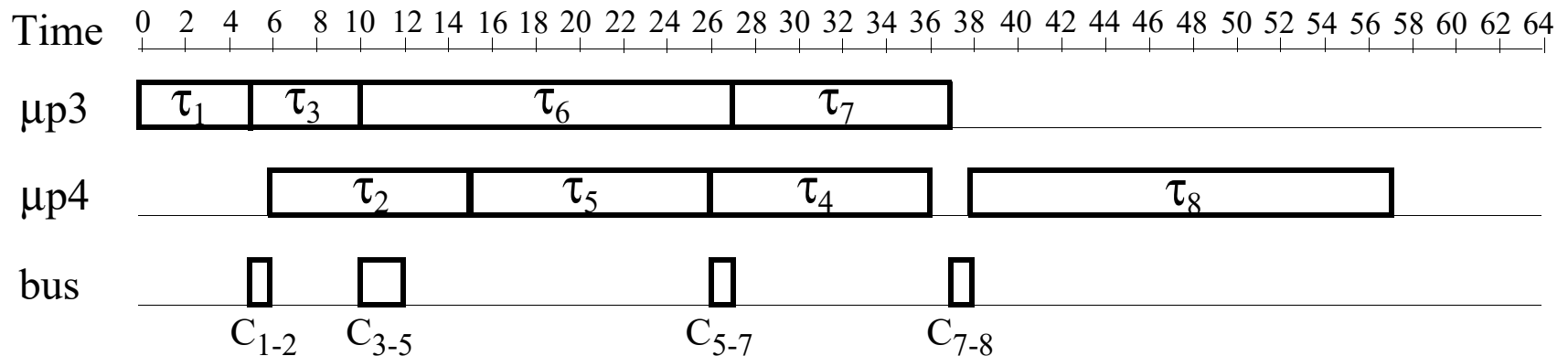
$\mu p4$: $\tau_2, \tau_4, \tau_5, \tau_8$.

Communication times and energy:

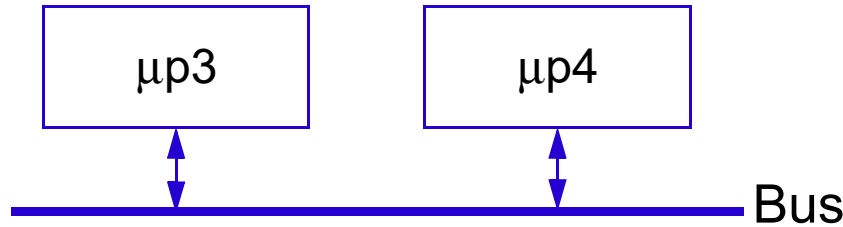
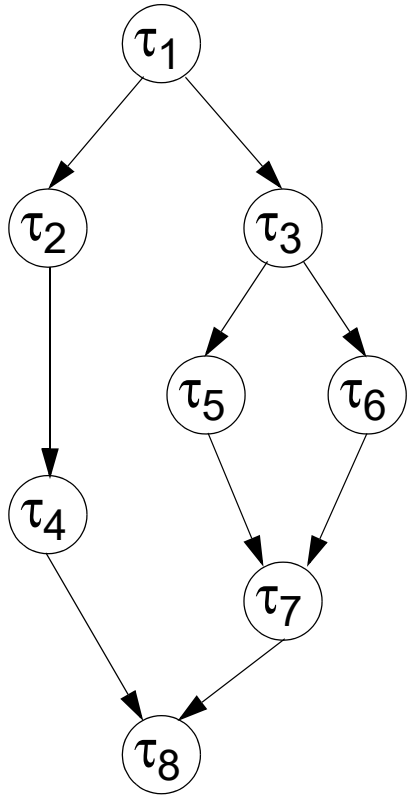
C_{1-2} : $t = 1$; $E = 3$. C_{3-5} : $t = 2$; $E = 5$.

C_{7-8} : $t = 1$; $E = 3$. C_{5-7} : $t = 1$; $E = 3$.

| Task | WCET | | Energy | |
|----------|----------|----------|----------|----------|
| | $\mu p3$ | $\mu p4$ | $\mu p3$ | $\mu p4$ |
| τ_1 | 5 | 6 | 5 | 3 |
| τ_2 | 7 | 9 | 8 | 4 |
| τ_3 | 5 | 6 | 5 | 3 |
| τ_4 | 8 | 10 | 6 | 4 |
| τ_5 | 10 | 11 | 8 | 6 |
| τ_6 | 17 | 21 | 15 | 10 |
| τ_7 | 10 | 14 | 8 | 7 |
| τ_8 | 15 | 19 | 14 | 9 |



Mapping for Low Energy



Consider a mapping:

μp3: $\tau_1, \tau_3, \tau_6, \tau_7$.

μp4: $\tau_2, \tau_4, \tau_5, \tau_8$.

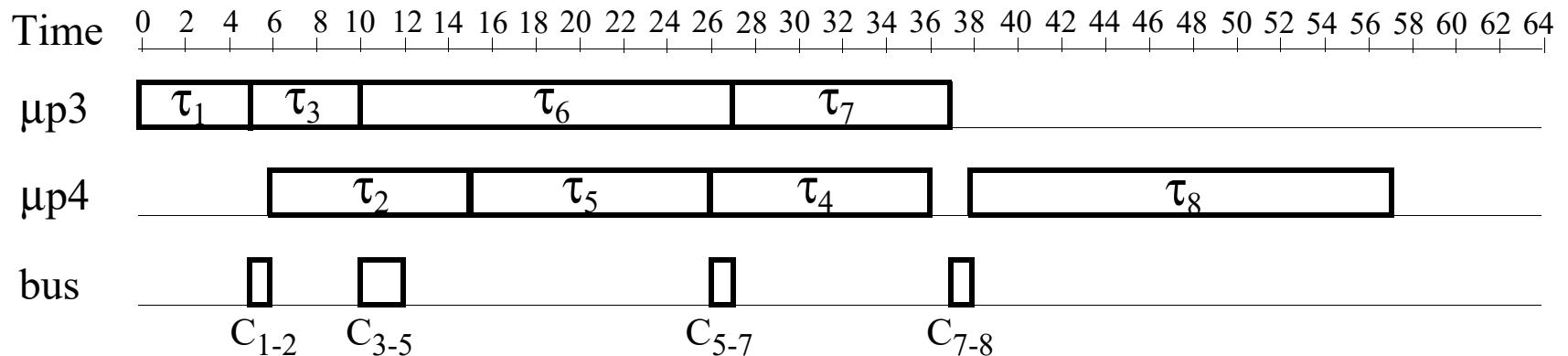
Communication times and energy:

C_{1-2} : $t = 1; E = 3$. **C_{3-5} :** $t = 2; E = 5$.

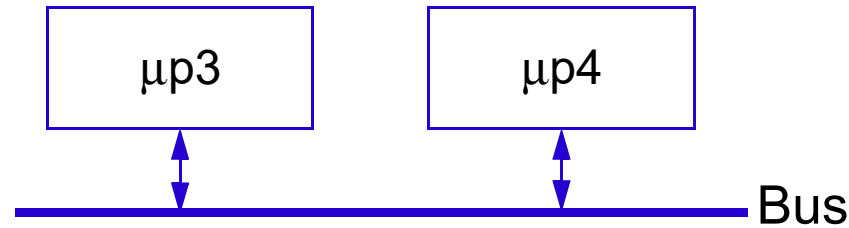
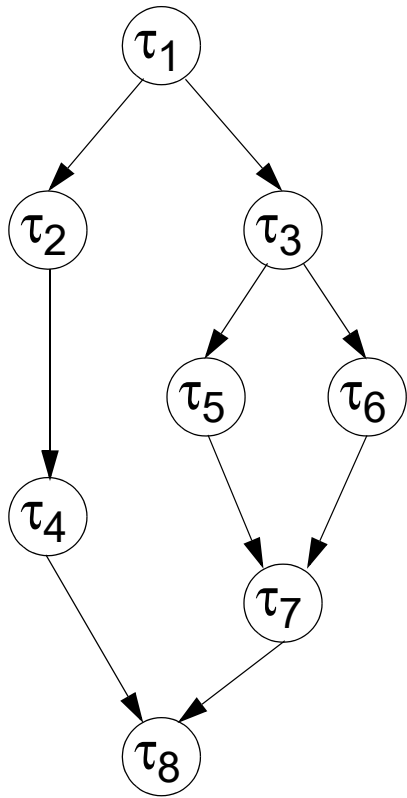
C_{7-8} : $t = 1; E = 3$. **C_{5-7} :** $t = 1; E = 3$.

| Task | WCET | | Energy | |
|----------|------|-----|--------|-----|
| | μp3 | μp4 | μp3 | μp4 |
| τ_1 | 5 | 6 | 5 | 3 |
| τ_2 | 7 | 9 | 8 | 4 |
| τ_3 | 5 | 6 | 5 | 3 |
| τ_4 | 8 | 10 | 6 | 4 |
| τ_5 | 10 | 11 | 8 | 6 |
| τ_6 | 17 | 21 | 15 | 10 |
| τ_7 | 10 | 14 | 8 | 7 |
| τ_8 | 15 | 19 | 14 | 9 |

Execution time: 57; Energy consumed: 70



Mapping for Low Energy



| Task | WCET | | Energy | |
|----------------|------|-----|--------|-----|
| | μp3 | μp4 | μp3 | μp4 |
| τ ₁ | 5 | 6 | 5 | 3 |
| τ ₂ | 7 | 9 | 8 | 4 |
| τ ₃ | 5 | 6 | 5 | 3 |
| τ ₄ | 8 | 10 | 6 | 4 |
| τ ₅ | 10 | 11 | 8 | 6 |
| τ ₆ | 17 | 21 | 15 | 10 |
| τ ₇ | 10 | 14 | 8 | 7 |
| τ ₈ | 15 | 19 | 14 | 9 |

- The second mapping with τ₈ on μp4 consumes less energy;
 - Assume that we have a maximum allowed delay = 60.



This second mapping is preferable, even if it is slower!

Real-Time Scheduling with Dynamic Voltage Scaling

- The energy consumed by a task, due to switching power:

$$E = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot N_{CY} \cdot N_{SW}$$

N_{SW} = number of gate transitions per clock cycle.

N_{CY} = number of cycles needed for the task.

Real-Time Scheduling with Dynamic Voltage Scaling

- The energy consumed by a task, due to switching power:

$$E = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot N_{CY} \cdot N_{SW}$$

N_{SW} = number of gate transitions per clock cycle.
 N_{CY} = number of cycles needed for the task.

- Reducing supply voltage V_{DD} is the efficient way to reduce energy consumption.
 - The frequency at which the processor can be operated depends on V_{DD} :

$$f = k \cdot \frac{(V_{DD} - V_t)^2}{V_{DD}}$$

k : circuit dependent constant; V_t : threshold voltage.

Real-Time Scheduling with Dynamic Voltage Scaling

- The energy consumed by a task, due to switching power:

$$E = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot N_{CY} \cdot N_{SW}$$

N_{SW} = number of gate transitions per clock cycle.
 N_{CY} = number of cycles needed for the task.

- Reducing supply voltage V_{DD} is the efficient way to reduce energy consumption.

- The frequency at which the processor can be operated depends on V_{DD} :

$$f = k \cdot \frac{(V_{DD} - V_t)^2}{V_{DD}}, \quad k: \text{circuit dependent constant}; V_t: \text{threshold voltage.}$$

- The execution time of the task: $t_{exe} = N_{CY} \cdot \frac{V_{DD}}{k \cdot (V_{DD} - V_t)^2}$

Depends on V_{DD} !

Real-Time Scheduling with Dynamic Voltage Scaling

- The (classical) scheduling problem:

Which task to execute at a certain moment on a certain processor so that time constraints are fulfilled?

Real-Time Scheduling with Dynamic Voltage Scaling

- The (classical) scheduling problem:

Which task to execute at a certain moment on a certain processor so that time constraints are fulfilled?

- The scheduling problem with voltage scaling:

Which task to execute at a certain moment on a certain processor, *and at which voltage level*, so that time constraints are fulfilled and *energy consumption is minimised*?

Real-Time Scheduling with Dynamic Voltage Scaling

- The (classical) scheduling problem:

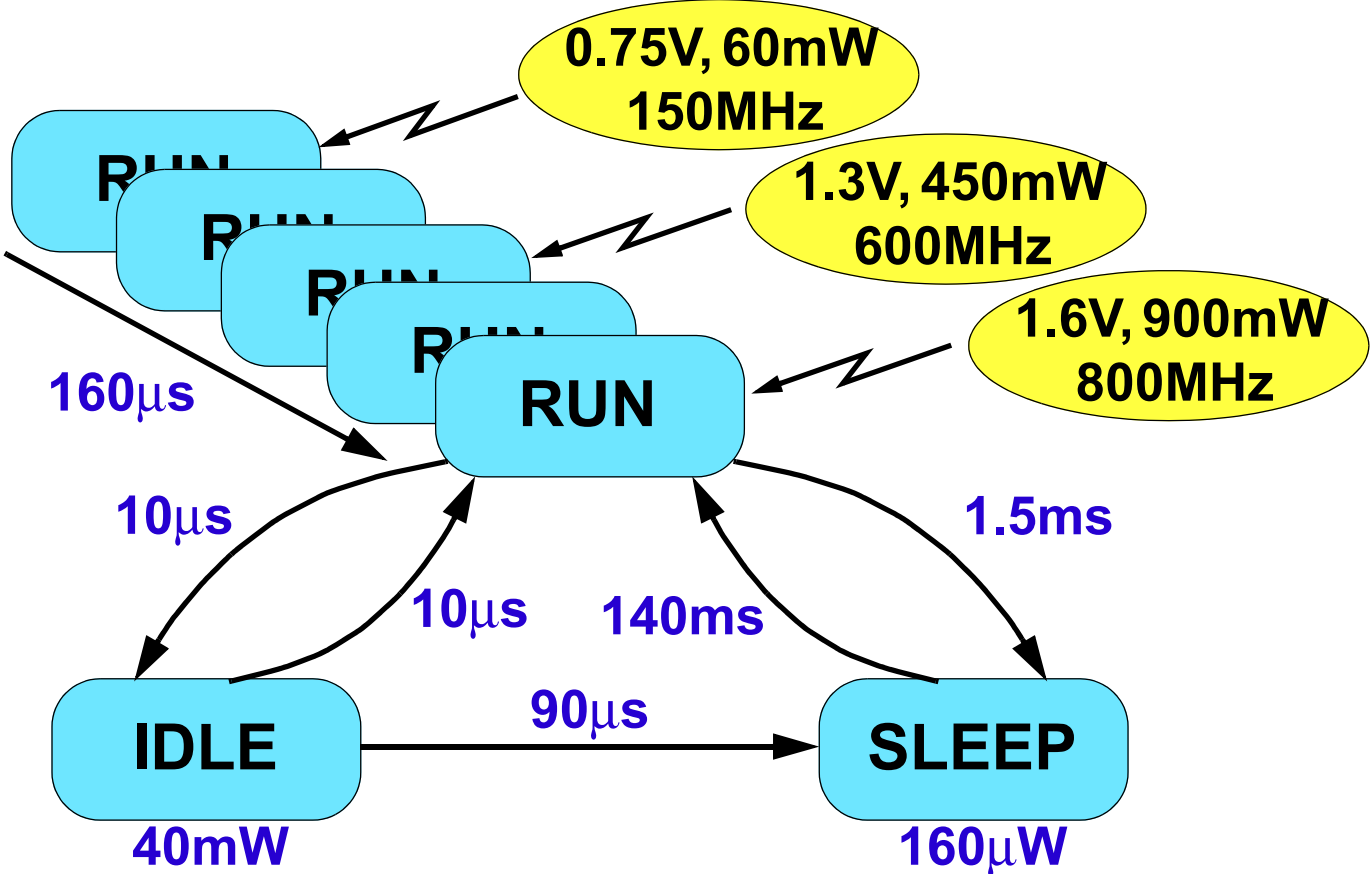
Which task to execute at a certain moment on a certain processor so that time constraints are fulfilled?

- The scheduling problem with voltage scaling:

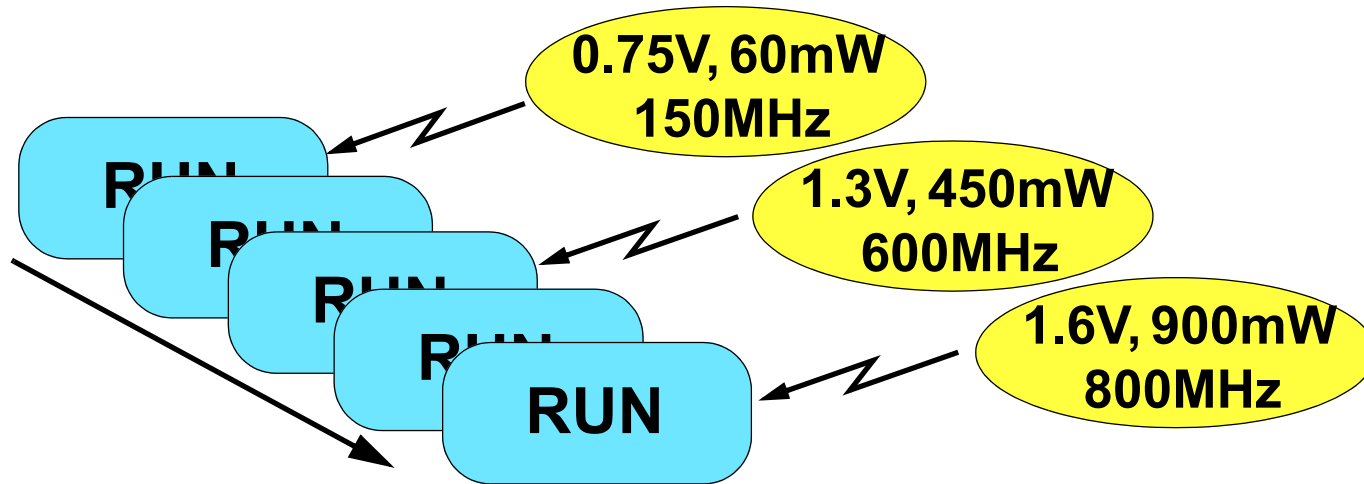
Which task to execute at a certain moment on a certain processor, *and at which voltage level*, so that time constraints are fulfilled and *energy consumption is minimised*?

- The problem: reducing supply voltage extends execution time!

Variable Voltage Processors



Variable Voltage Processors



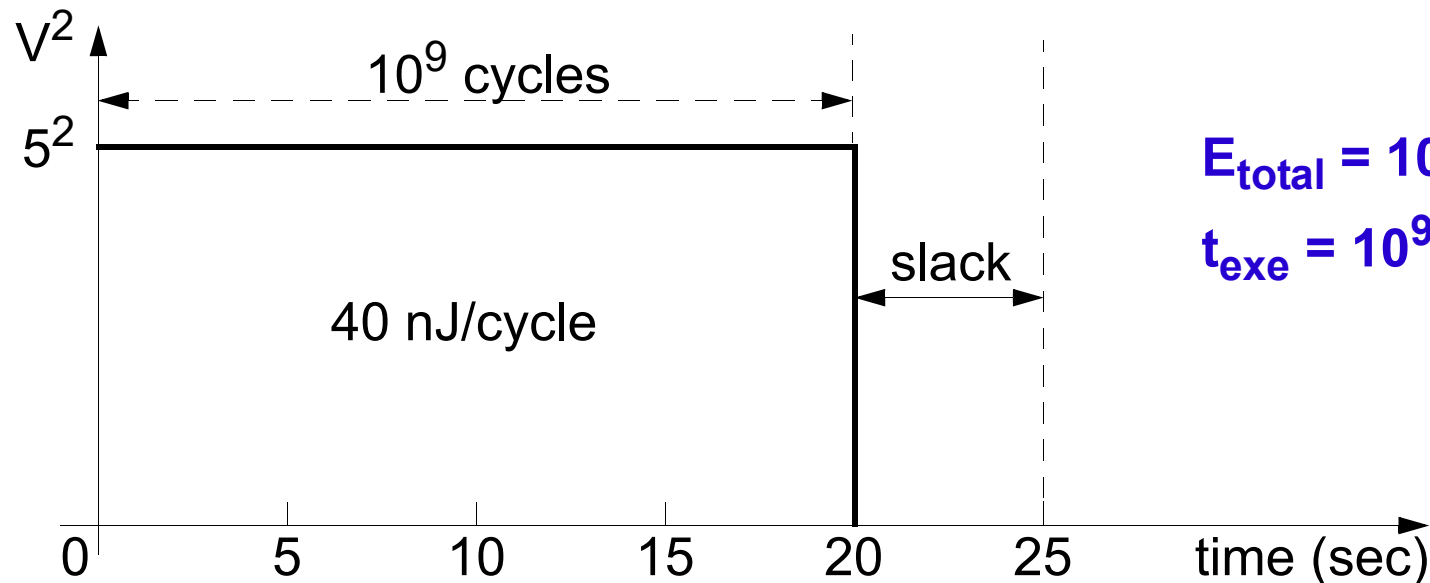
- Several supply voltage levels are available.
- Supply voltage can be changed during run-time.
- Frequency is adjusted to the current supply voltage.

The Basic Principle

- We consider a single task τ :
 - total computation: 10^9 execution cycles.
 - deadline: 25 seconds.
 - processor nominal (maximum) voltage: 5V.
 - energy: 40 nJ/cycle at nominal voltage.
 - processor speed: 50MHz (50×10^6 cycles/sec) at nominal voltage.

The Basic Principle

- We consider a single task τ :
 - total computation: 10^9 execution cycles.
 - deadline: 25 seconds.
 - processor nominal (maximum) voltage: 5V.
 - energy: 40 nJ/cycle at nominal voltage.
 - processor speed: 50MHz (50×10^6 cycles/sec) at nominal voltage.

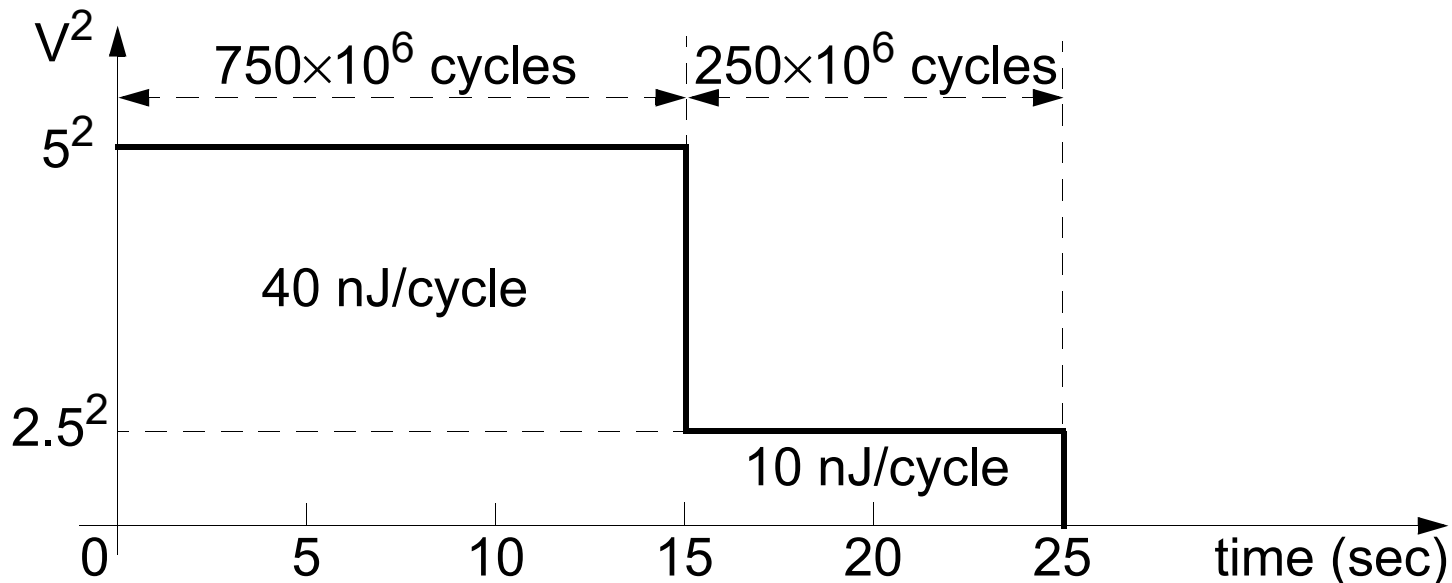


$$E_{\text{total}} = 10^9 \times (40 \times 10^{-9}) = 40 \text{ J}$$

$$t_{\text{exe}} = 10^9 / (50 \times 10^6) = 20 \text{ sec}$$

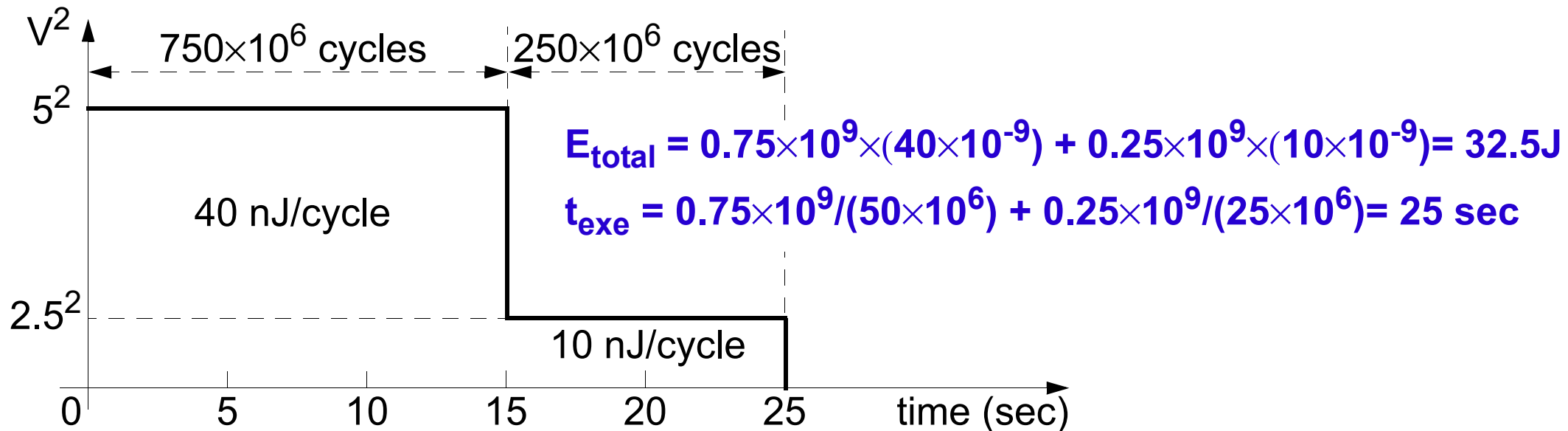
The Basic Principle

- We consider a single task τ :
 - total computation: 10^9 execution cycles.
 - deadline: 25 seconds.
 - processor nominal (maximum) voltage: 5V.
 - energy: 40 nJ/cycle at nominal voltage; **at 2.5V: $40 \times 2.5^2 / 5^2 = 10$ nJ/cycle**
 - processor speed: 50MHz (50×10^6 cycles/sec) at nominal voltage;
at 2.5V: $50 \times 2.5 / 5 = 25$ MHz (25×10^6 cycles/sec).



The Basic Principle

- We consider a single task τ :
 - total computation: 10^9 execution cycles.
 - deadline: 25 seconds.
 - processor nominal (maximum) voltage: 5V.
 - energy: 40 nJ/cycle at nominal voltage; **at 2.5V: $40 \times 2.5^2 / 5^2 = 10 \text{ nJ/cycle}$**
 - processor speed: 50MHz (50×10^6 cycles/sec) at nominal voltage;
at 2.5V: $50 \times 2.5 / 5 = 25 \text{ MHz}$ (25×10^6 cycles/sec).



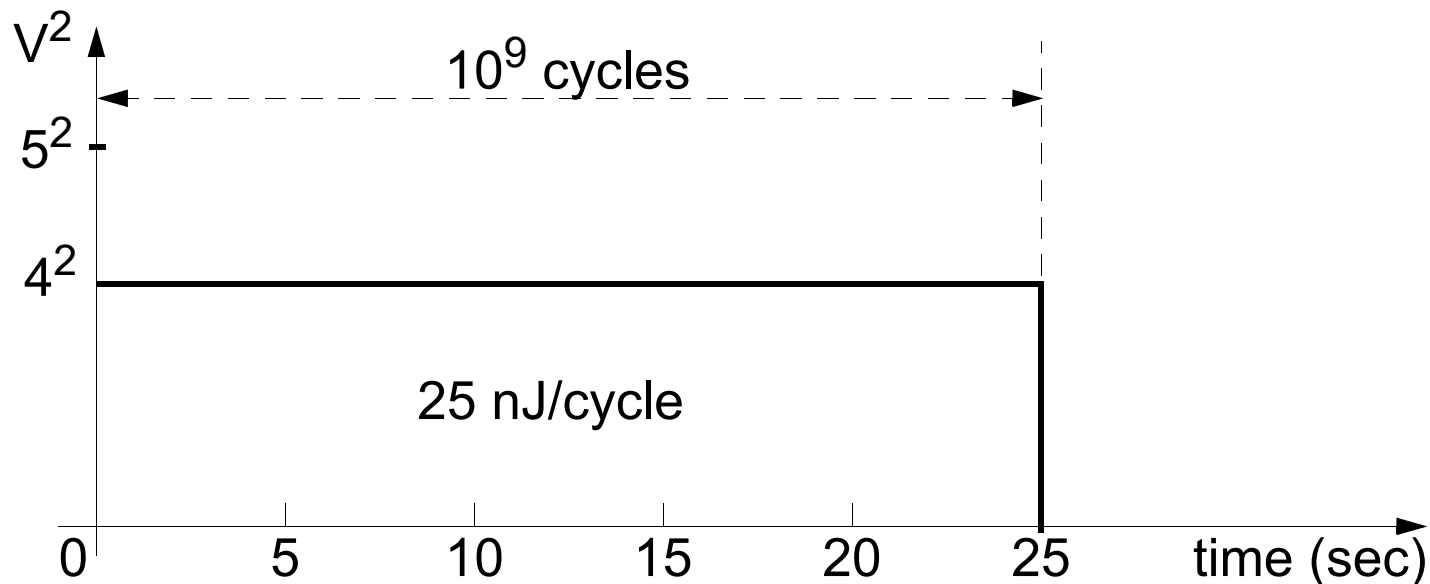
The Basic Principle

- We consider a single task τ :
 - total computation: 10^9 execution cycles.
 - deadline: 25 seconds.
 - processor nominal (maximum) voltage: 5V.
 - energy: 40 nJ/cycle at nominal voltage; at 2.5V: $40 \times 2.5^2 / 5^2 = 10 \text{ nJ/cycle}$
 - processor speed: 50MHz (50×10^6 cycles/sec) at nominal voltage;
at 2.5V: $50 \times 2.5 / 5 = 25 \text{ MHz}$ (25×10^6 cycles/sec).

Let's try a different solution!

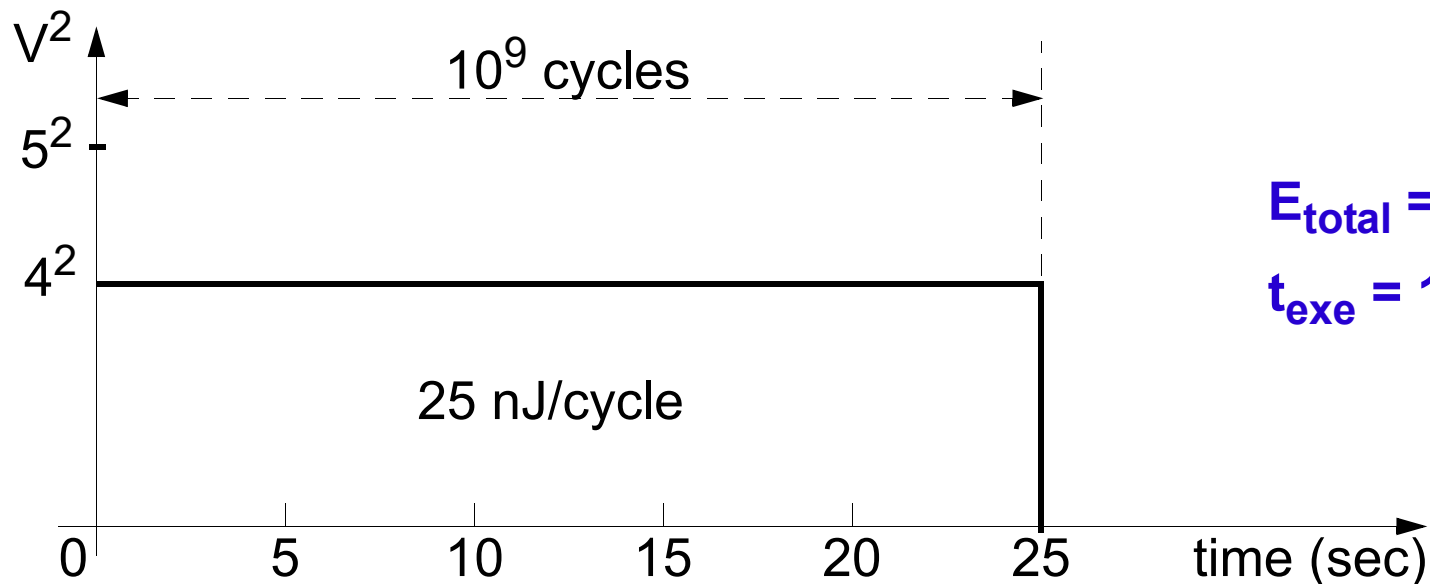
The Basic Principle

- We consider a single task τ :
 - total computation: 10^9 execution cycles.
 - deadline: 25 seconds.
 - processor nominal (maximum) voltage: 5V.
 - energy: 40 nJ/cycle at nominal voltage; **at 4V: $40 \times 4^2 / 5^2 = 25$ nJ/cycle**
 - processor speed: 50MHz (50×10^6 cycles/sec) at nominal voltage;
at 4V: $50 \times 4 / 5 = 40$ MHz (40×10^6 cycles/sec).



The Basic Principle

- We consider a single task τ :
 - total computation: 10^9 execution cycles.
 - deadline: 25 seconds.
 - processor nominal (maximum) voltage: 5V.
 - energy: 40 nJ/cycle at nominal voltage; **at 4V: $40 \times 4^2 / 5^2 = 25$ nJ/cycle**
 - processor speed: 50MHz (50×10^6 cycles/sec) at nominal voltage;
at 4V: $50 \times 4 / 5 = 40$ MHz (40×10^6 cycles/sec).



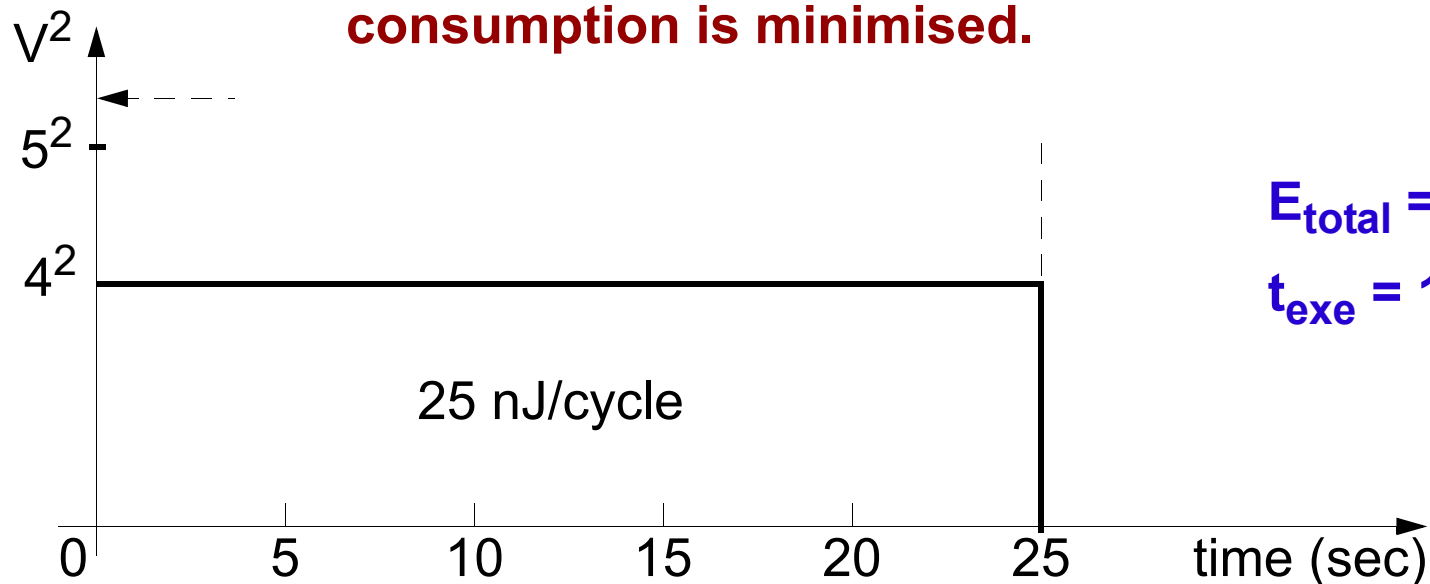
$$E_{\text{total}} = 10^9 \times (25 \times 10^{-9}) = 25 \text{ J}$$

$$t_{\text{exe}} = 10^9 / (40 \times 10^6) = 25 \text{ sec}$$

The Basic Principle

- We consider a single task τ :
 - total computation: 10^9 execution cycles.
 - deadline: 25 seconds.
 - processor nominal (maximum) voltage: 5V.
 - energy: 40 nJ/cycle at nominal voltage; at 4V: $40 \times 4^2 / 5^2 = 25 \text{ nJ/cycle}$
 - processor speed: 50MHz (50×10^6 cycles/sec) at nominal voltage;

If a processor uses a single supply voltage and completes a program just on deadline, the energy consumption is minimised. :).

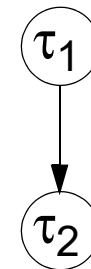


$$E_{\text{total}} = 10^9 \times (25 \times 10^{-9}) = 25 \text{ J}$$

$$t_{\text{exe}} = 10^9 / (40 \times 10^6) = 25 \text{ sec}$$

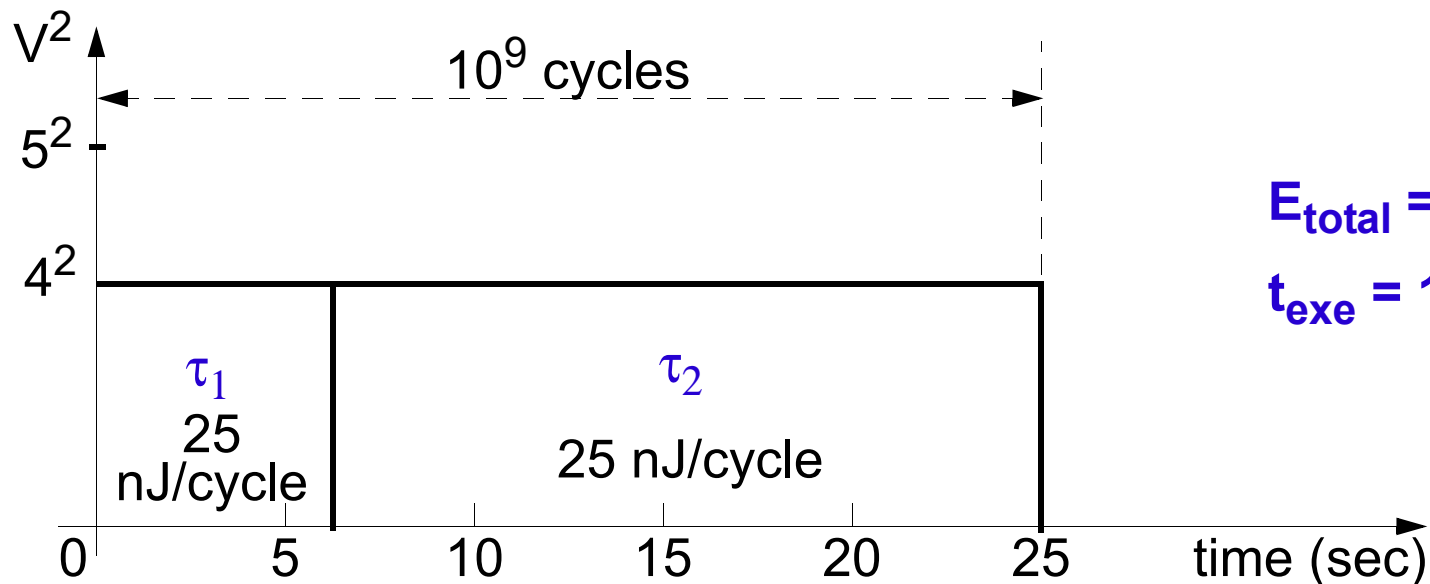
The Basic Principle

- We consider two tasks τ_1 and τ_2 :
 - Computation τ_1 : 250×10^6 execution cycles; τ_2 : 750×10^6 execution cycles
 - deadline: 25 seconds.
 - processor nominal (maximum) voltage: 5V.
 - energy: 40 nJ/cycle at nominal voltage; at 4V: $40 \times 4^2 / 5^2 = 25$ nJ/cycle
 - processor speed: 50MHz (50×10^6 cycles/sec) at nominal voltage; at 4V: $50 \times 4/5 = 40$ MHz (40×10^6 cycles/sec).



The Basic Principle

- We consider two tasks τ_1 and τ_2 :
 - Computation τ_1 : 250×10^6 execution cycles; τ_2 : 750×10^6 execution cycles
 - deadline: 25 seconds.
 - processor nominal (maximum) voltage: 5V.
 - energy: 40 nJ/cycle at nominal voltage; at 4V: $40 \times 4^2 / 5^2 = 25 \text{ nJ/cycle}$
 - processor speed: 50MHz (50×10^6 cycles/sec) at nominal voltage; at 4V: $50 \times 4/5 = 40 \text{ MHz}$ (40×10^6 cycles/sec).



$$E_{\text{total}} = 10^9 \times (25 \times 10^{-9}) = 25 \text{ J}$$

$$t_{\text{exe}} = 10^9 / (40 \times 10^6) = 25 \text{ sec}$$

Considering Task Particularities

- Energy consumed by a task:

$$E = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot N_{CY} \cdot N_{SW}$$

N_{SW} = number of gate transitions per clock cycle.

C = switched capacitance per clock cycle.

- Average energy consumed by task per cycle:

$$E_{CY} = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot N_{SW}$$

- Often tasks differ from each other in terms of executed operations \Rightarrow N_{SW} and C differ from one task to the other.



The average energy consumed per cycle differs from task to task.

Considering Task Particularities

- If power consumption per cycle differs from task to task the “basic principle” is not longer true!

Voltage levels have to be reduced with priority for those tasks which have a larger energy consumption per cycle.

- One individual voltage level has to be established for each task, so that deadlines are just satisfied.

Conclusions

- **Embedded systems are everywhere.**
- **They have to satisfy strong timing, safety, power, and cost constraints.**
- **An efficient design flow, with iterations at the system level, is needed in order to support the design of complex embedded systems.**
- **System level design steps are performed *before* the start of the actual implementation of hardware and software components!**
- **The input to the actual design flow is an abstract model of the system.**
- **Power consumption becomes a central issue of the design process.**