System Design and Methodology/ Embedded Systems Design (Modeling and Design of Embedded Systems)

TDTS07/TDDI08
VT 2023

Soheil Samii

Institutionen för datavetenskap (IDA)
Linköpings universitet

email: soheil.samii@liu.se
B building, 329:220

Acknowledgment: All lecture notes are developed by Prof. Petru Eles

Course Information

Web page: http://www.ida.liu.se/~TDTS07
http://www.ida.liu.se/~TDDI08

Examination: written, March 22, 14:00-18:00
labs (see course page and lesson notes)

Lecture notes: available on the web page, before the lecture.

Course Information

Recommended literature:

Peter Marwedel: "*Embedded System Design*",
Springer, 2nd edition 2011, 3d edition 2018, 4th edition 2021.

The 4th edition is open access and available online via Springer.com

Edward Lee, Sanjit Seshia: "*Introduction to Embedded Systems - A Cyber-Physical Systems Approach*",
1st edition 2011, 2nd edition 2017 '
(available online: LeeSeshia.org)

Course Information

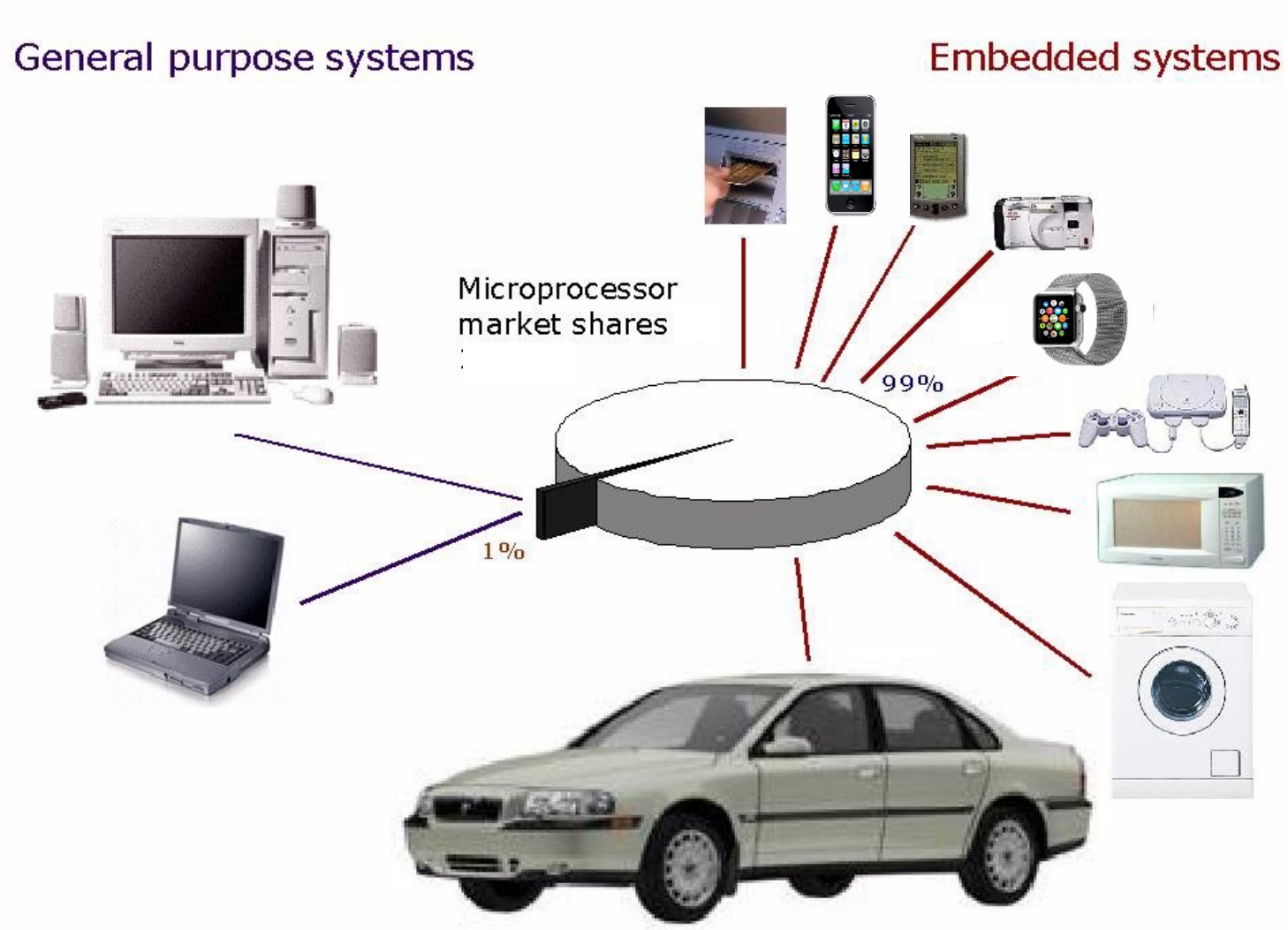
Lessons&Labs:

Yungang Pan
Institutionen för datavetenskap (IDA)
email: yungang.pan@liu.se
B building, 329:202

EMBEDDED SYSTEMS AND THEIR DESIGN

1. What is an Embedded System
2. Characteristics of Embedded Applications
3. The Traditional Design Flow
4. An Example
5. A New Design Flow
6. The System Level
7. Course Topics

That’s how we use microprocessors



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.”

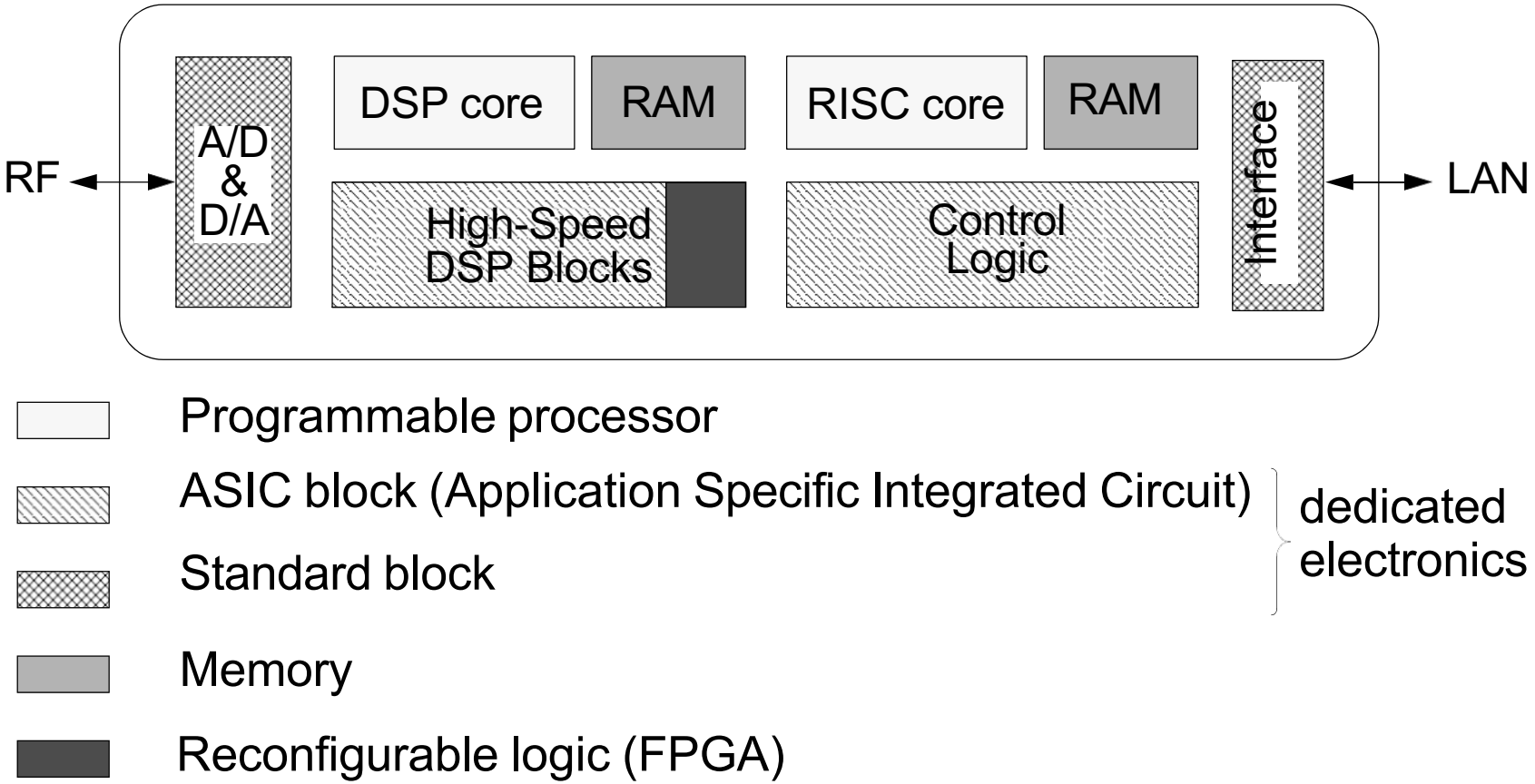
What is an Embedded System?

Some of the main characteristics:

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

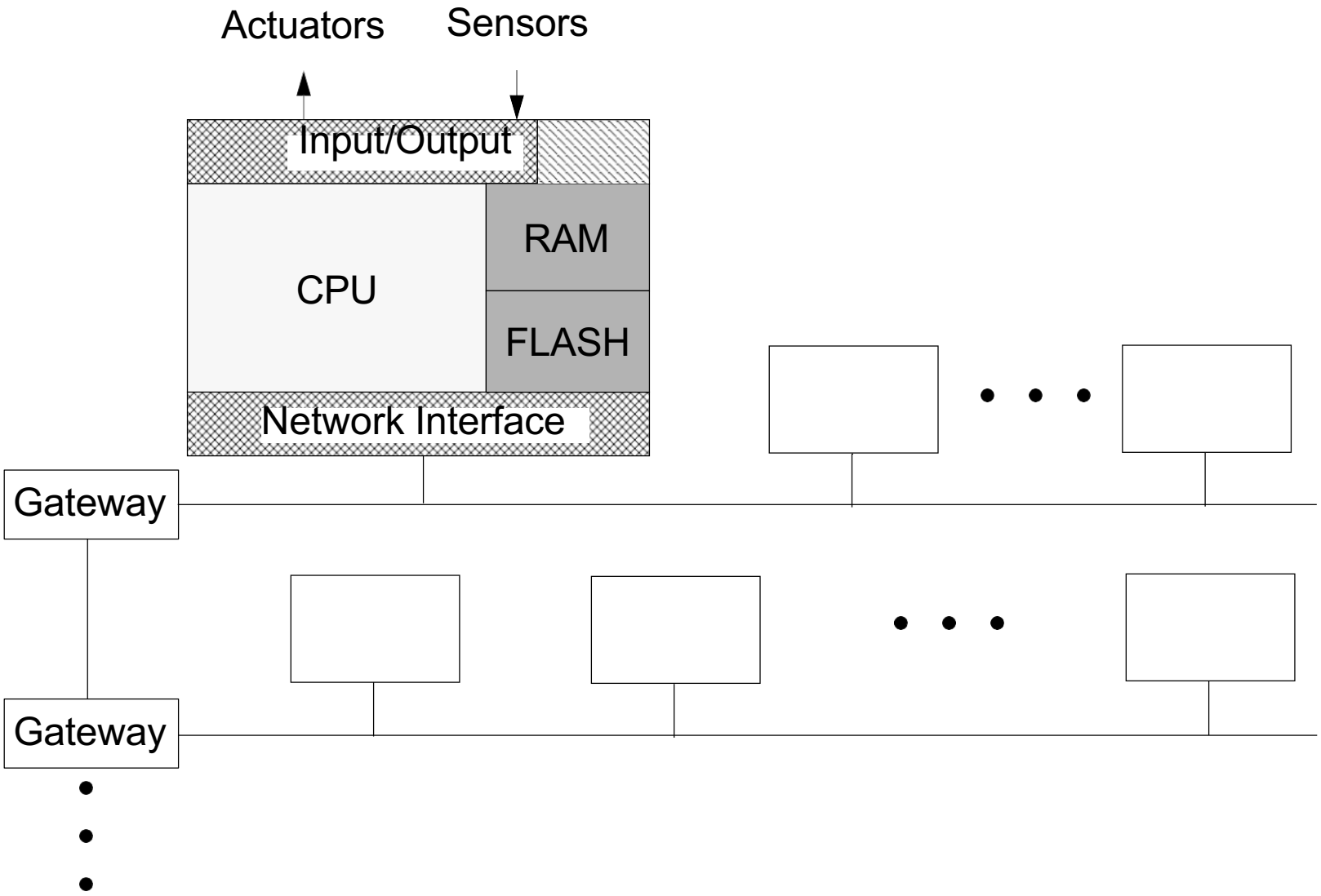
Two Typical Implementation Architectures

Telecommunication System on Chip



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 $\Rightarrow$ large electricity bill
    expensive power supply
    expensive cooling system
  - Reliability

    High power consumption $\Rightarrow$ high temperature that affects life time
  - Battery life

    High energy consumption $\Rightarrow$ 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
  - Material cost (Bill of Material)
  - Warranty 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.

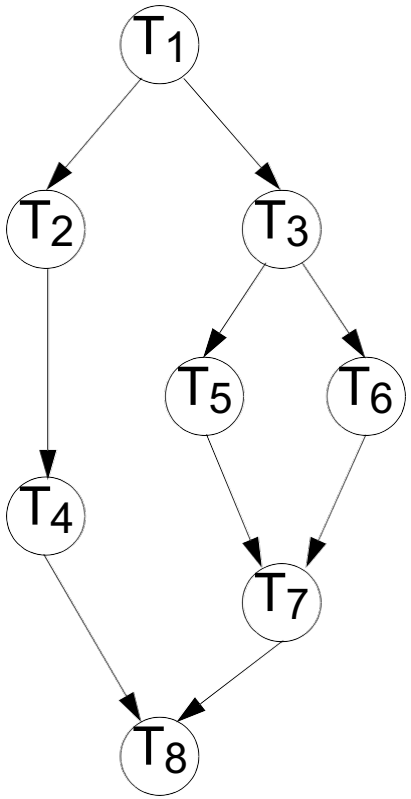
Why is Design of Embedded Systems Difficult?

- 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!

An 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.

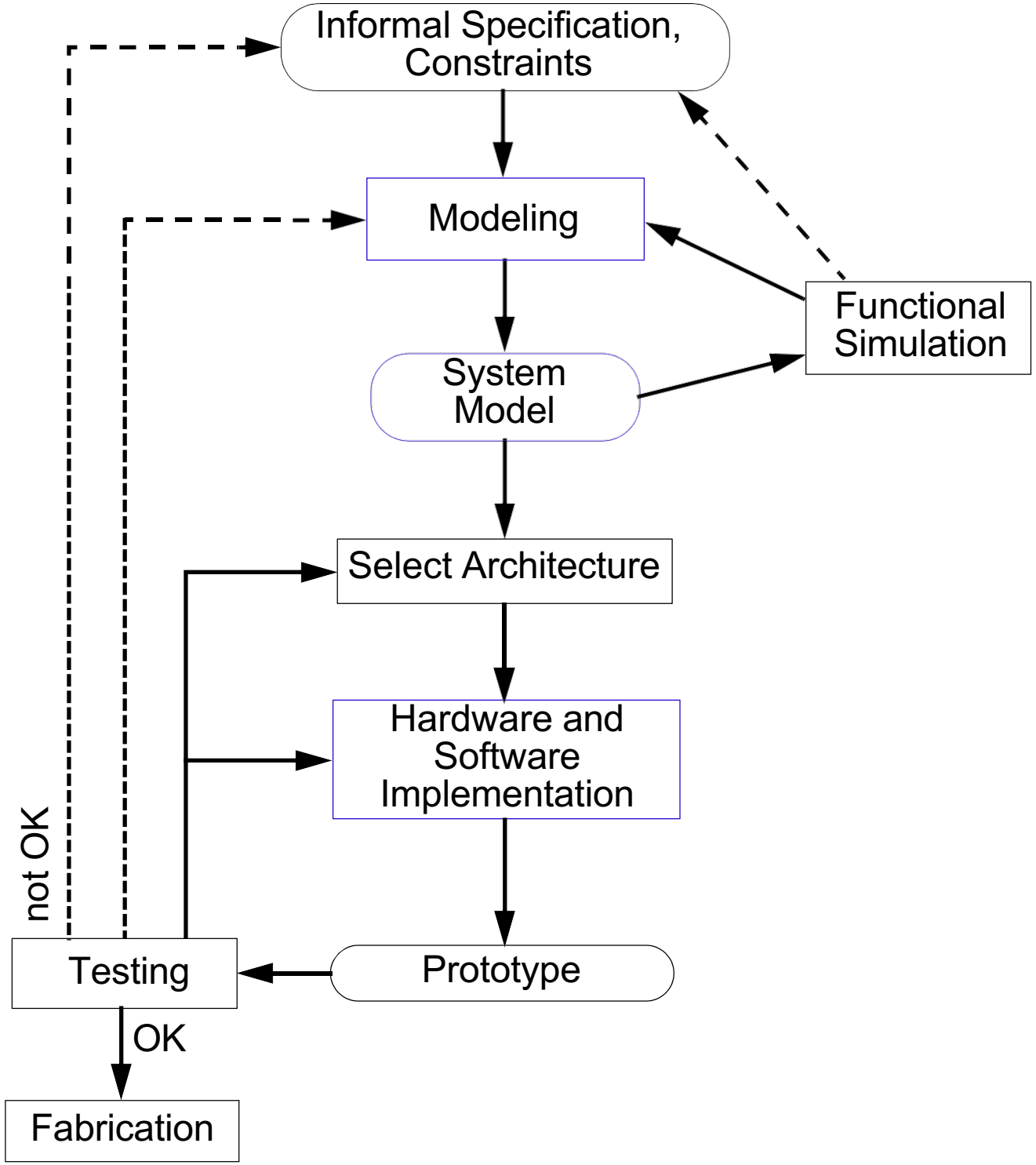
Period: 42 time units

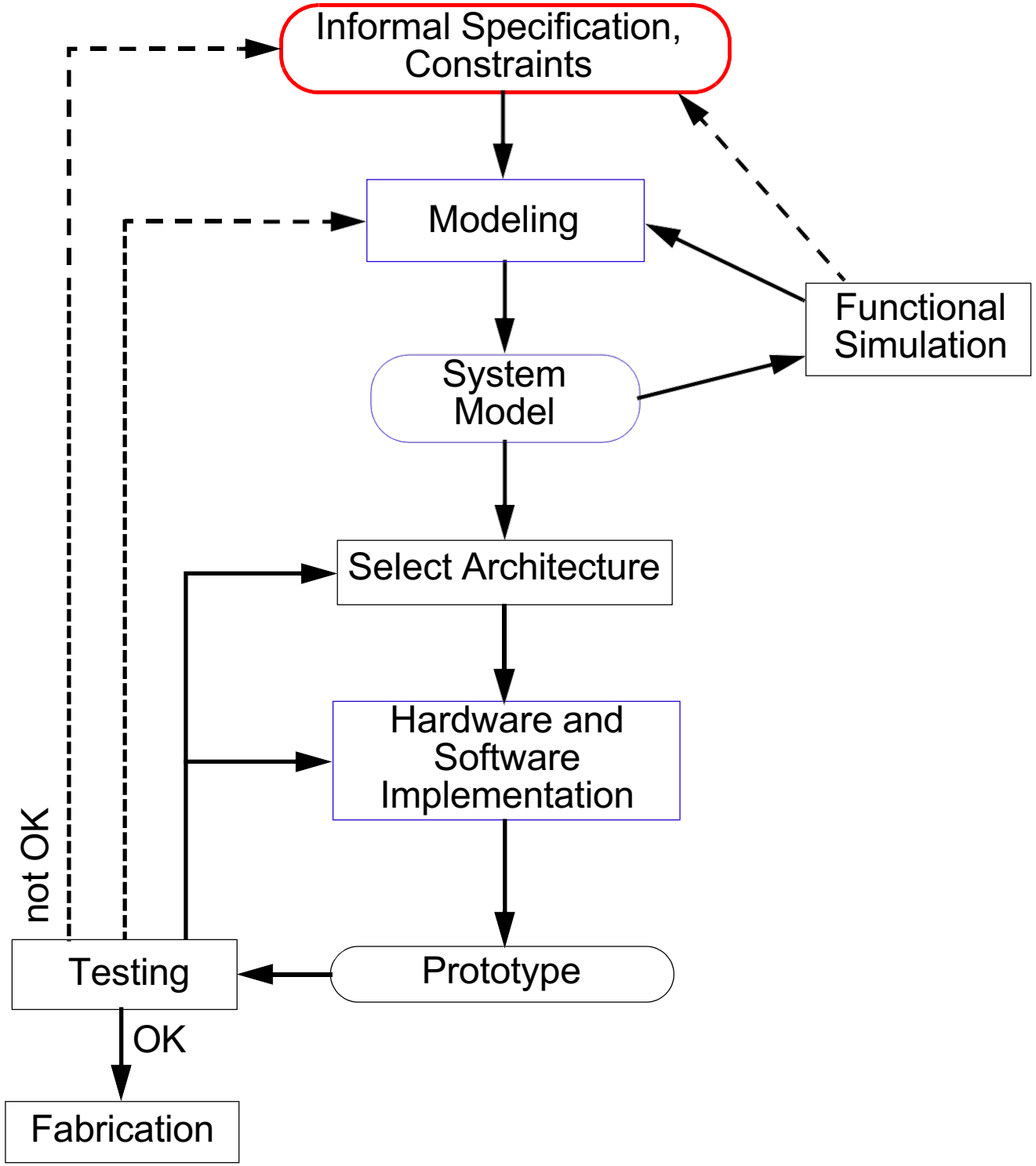
- The task graph is activated every 42 time units $\Rightarrow$ an activation has to terminate in time less than 42.

Cost limit: 8

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

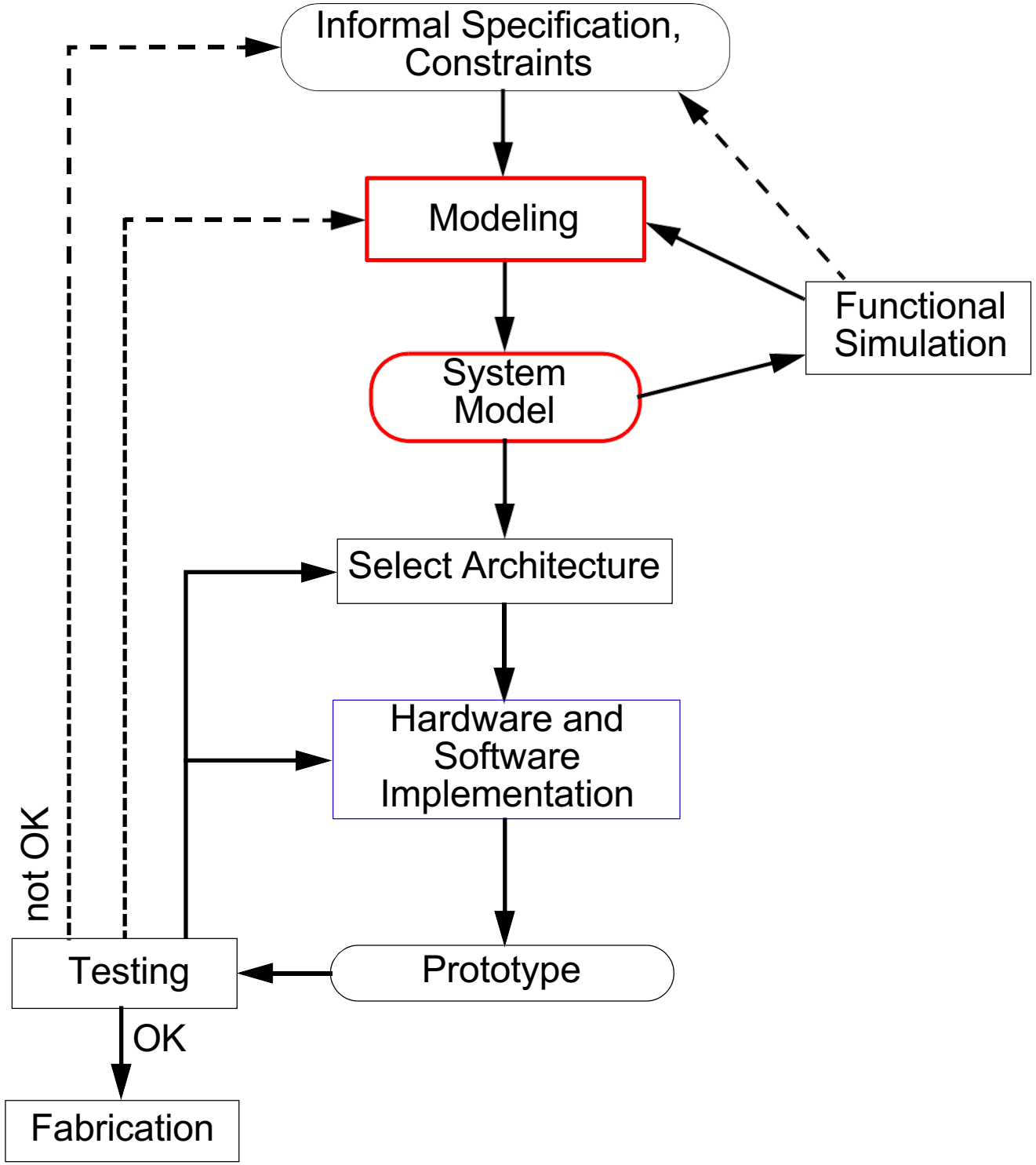
Traditional Design Flow





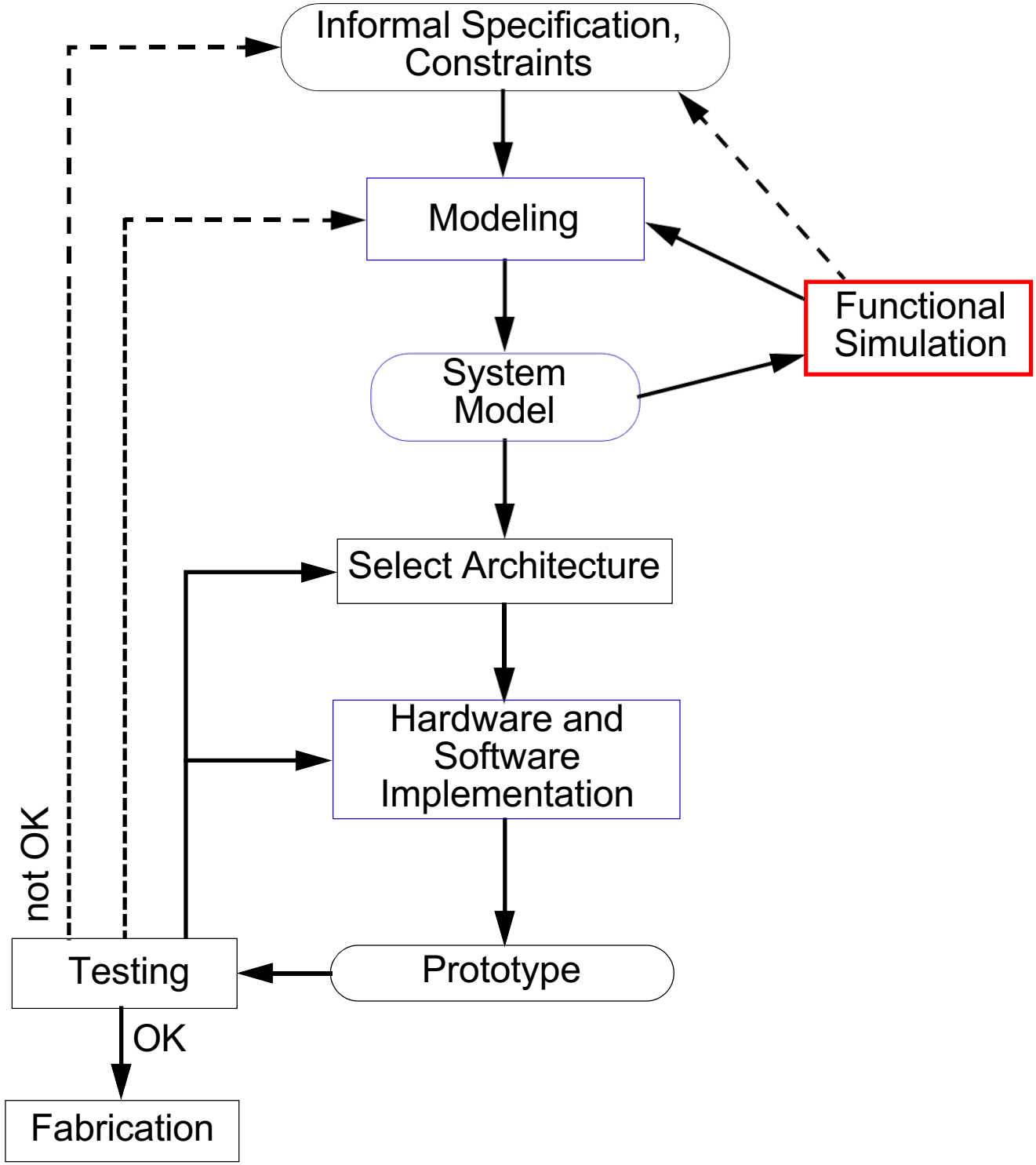
Traditional Design Flow

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



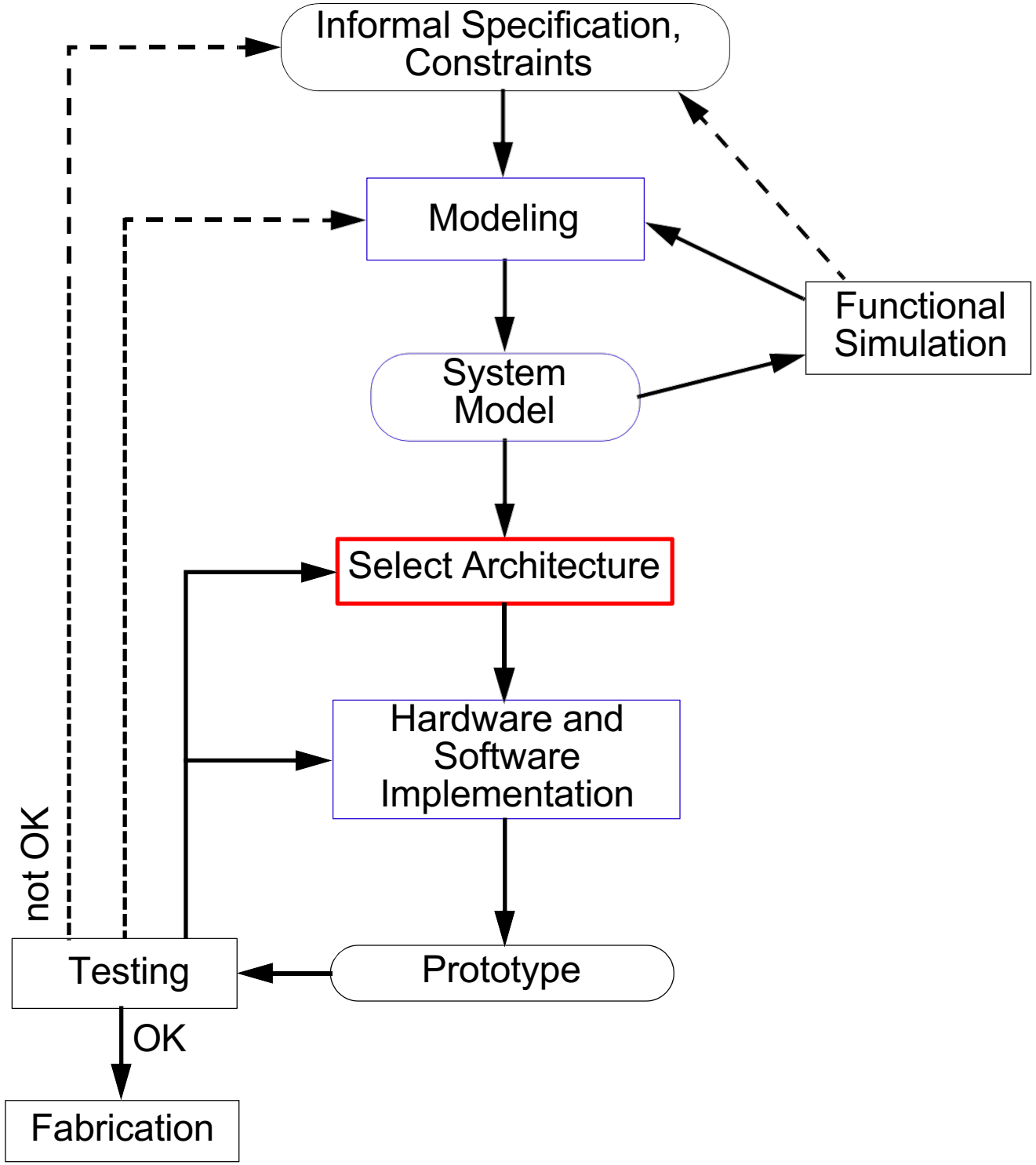
Traditional Design Flow

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



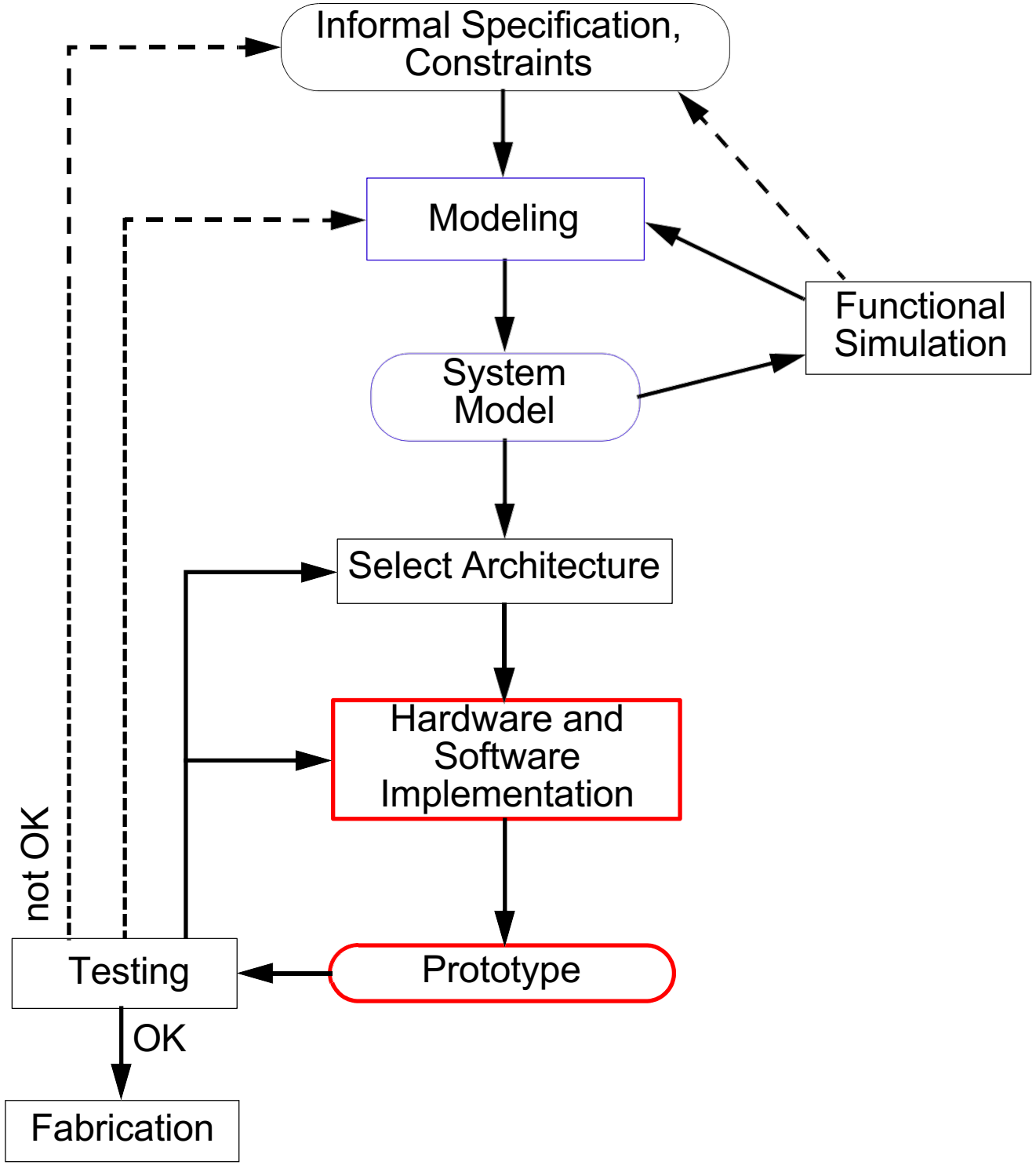
Traditional Design Flow

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.



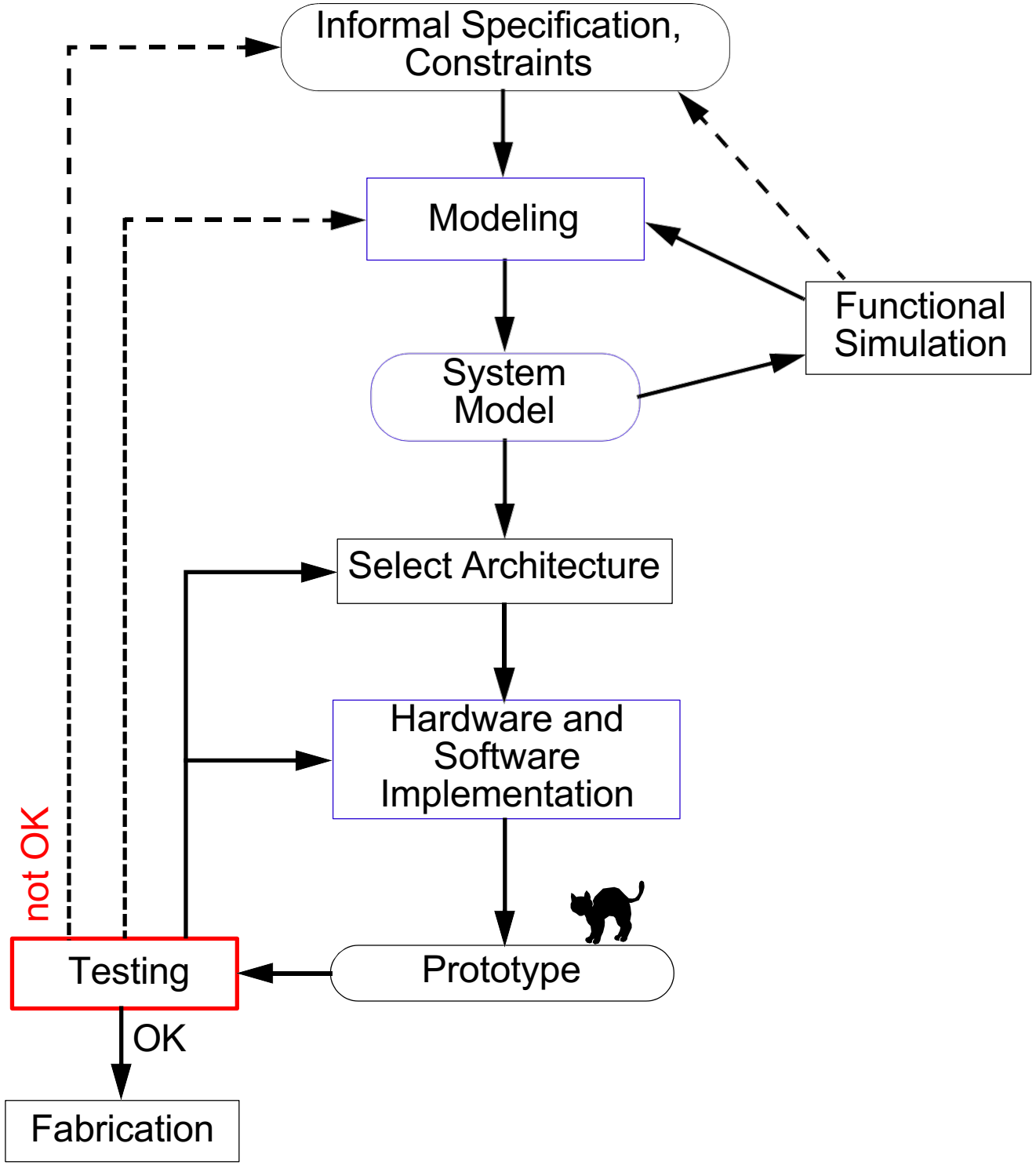
Traditional Design Flow

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 (μprocessor, buses, etc.) such that cost limits are satisfied and, you hope, time and power constraints are fulfilled.



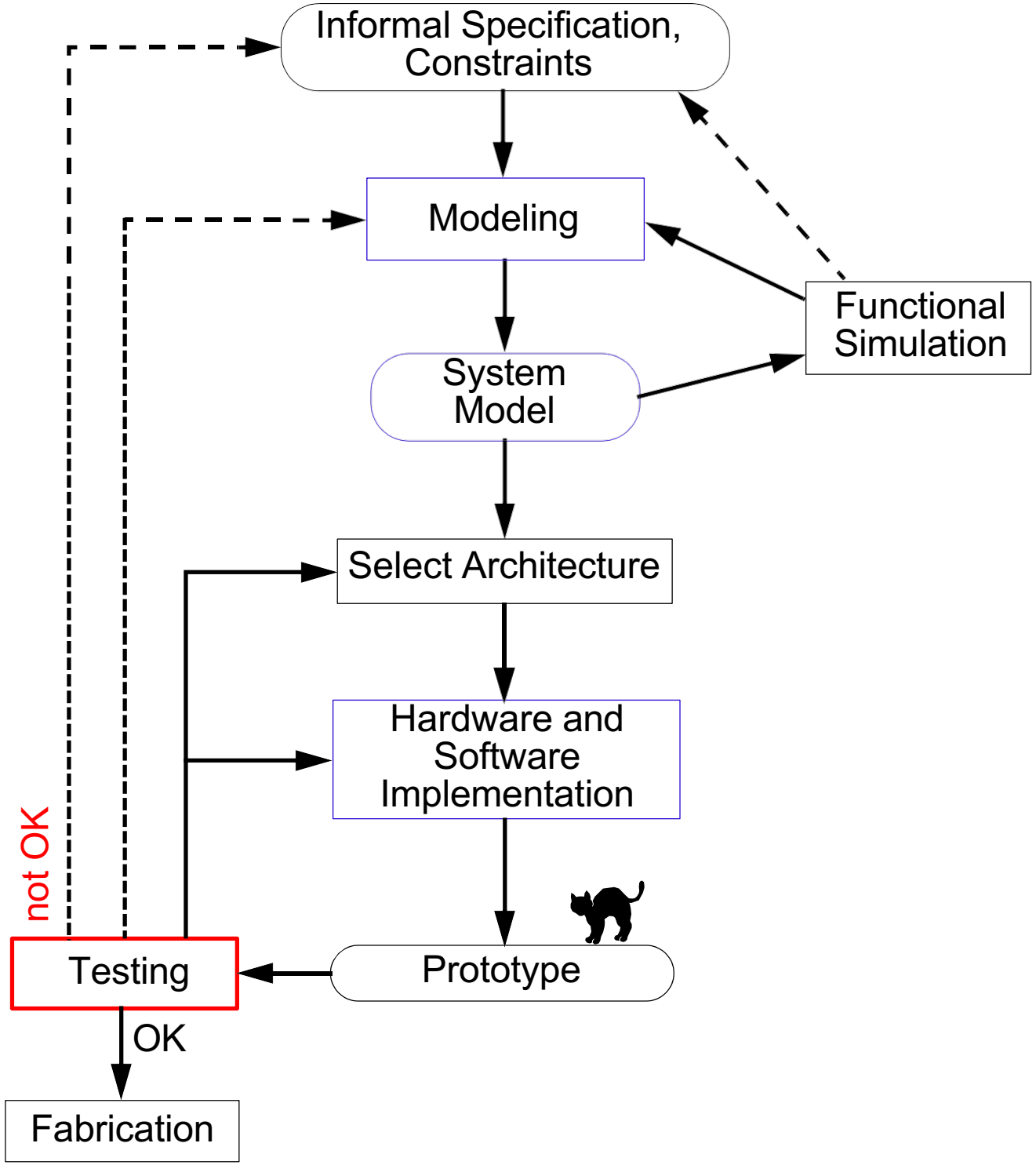
Traditional Design Flow

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 (μprocessor, 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 (μprocessor, 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!!!



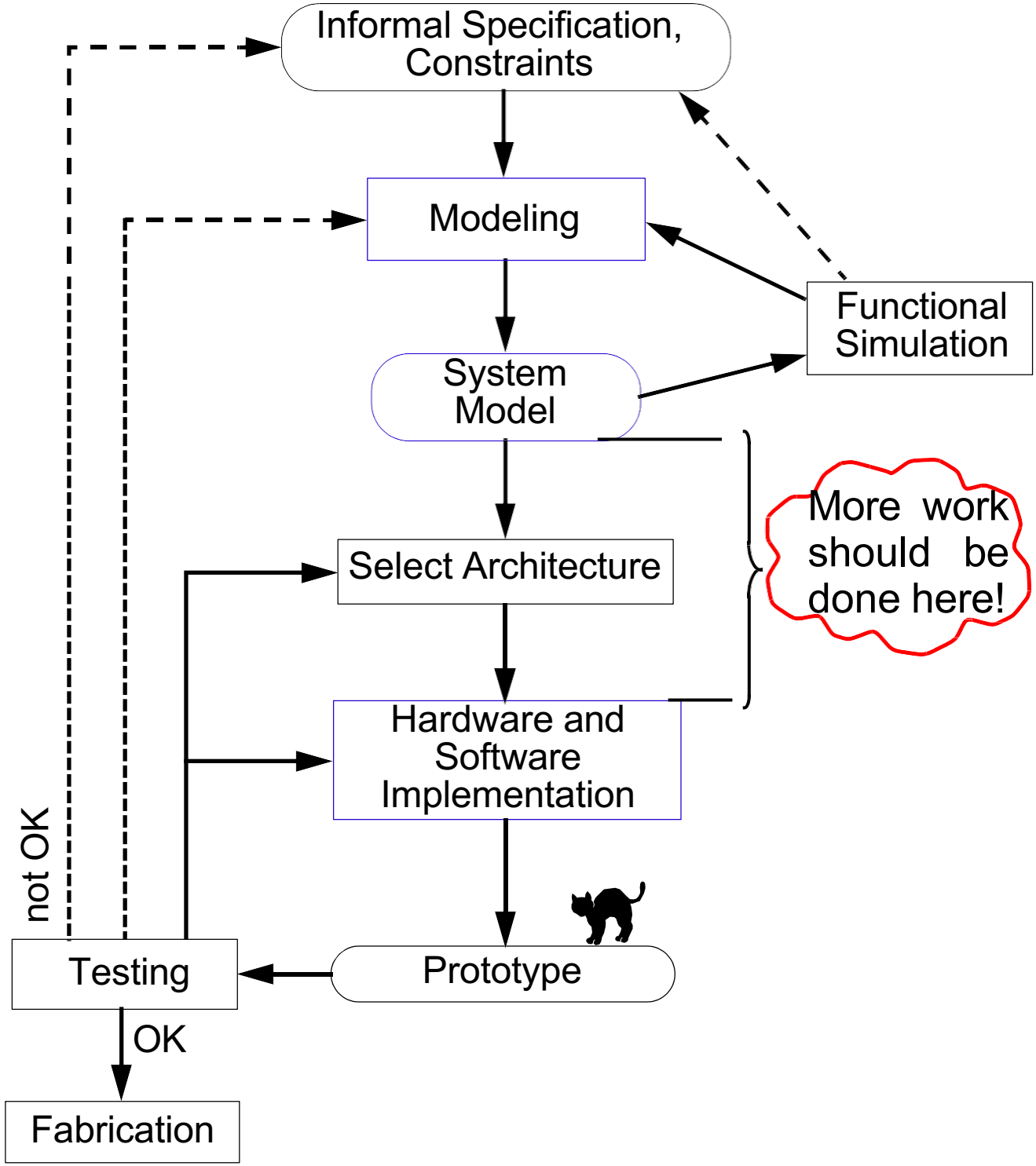
Traditional Design Flow

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

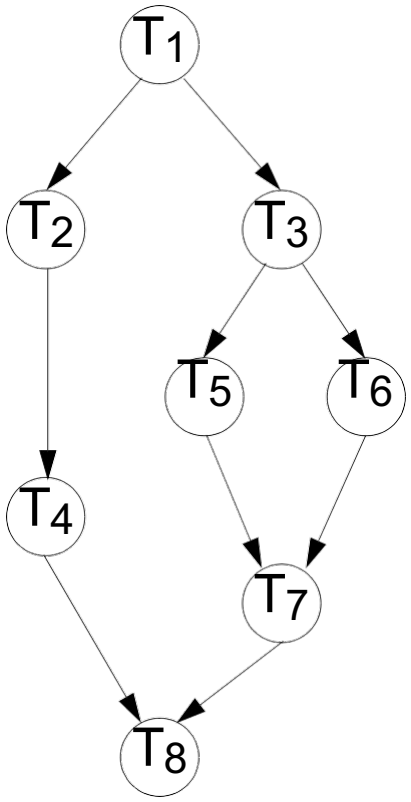
- **Go back to 4, choose a new architecture and start a new implementation.**
- **Or negotiate with the customer on the constraints.**

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

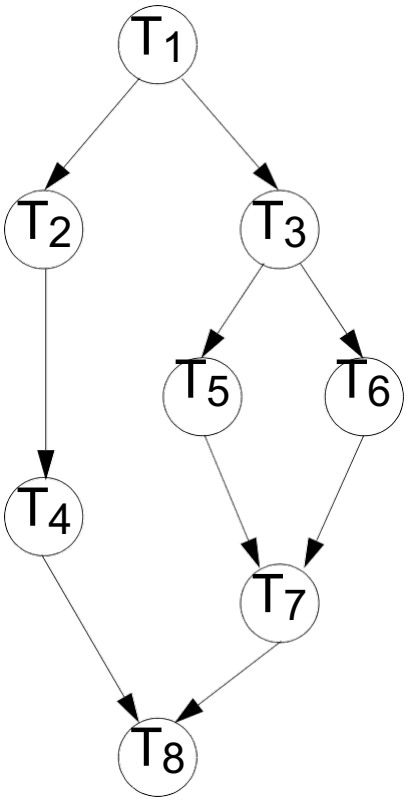

Informal Specification, Constraints
Modeling
Functional Simulation
System Model
More work should be done here!
Select Architecture
Hardware and Software Implementation
Prototype
Testing
not OK
OK
Fabrication

Example

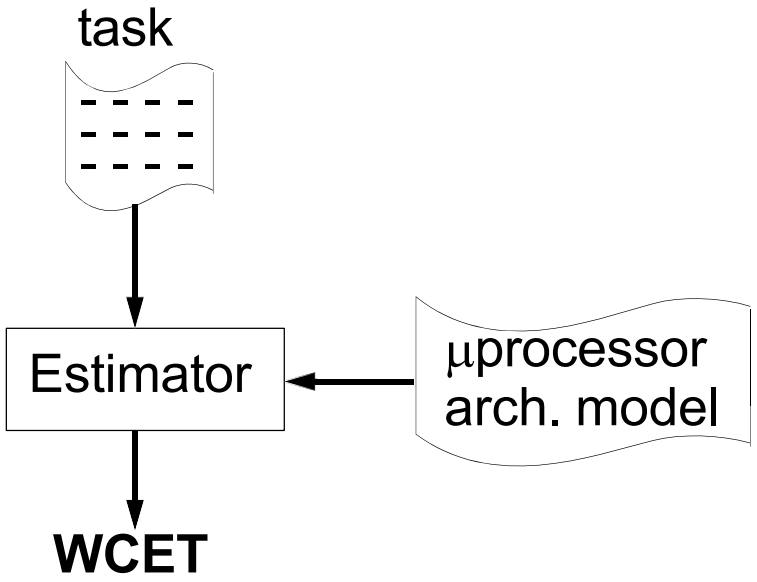


- We have the system model (task graph) which has been validated by simulation.
- We decide on a certain μprocessor μp1, with cost 6.
- For each task the worst case execution time (WCET) when run on μp1 is *estimated*.

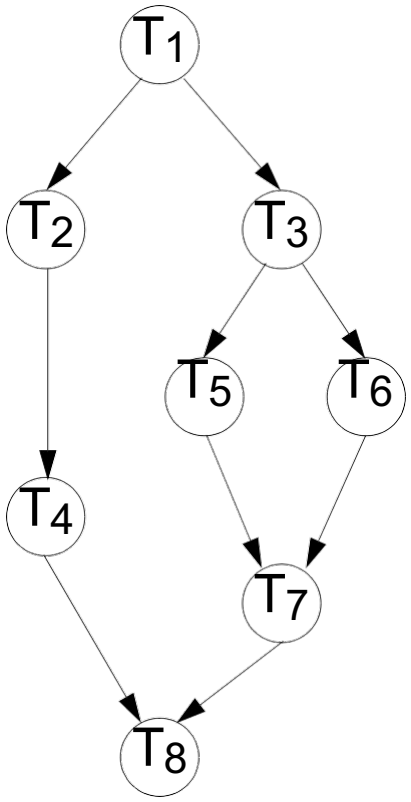
Example



- We have the system model (task graph) which has been validated by simulation.
- We decide on a certain μprocessor μp1, with cost 6.
- For each task the worst case execution time (WCET) when run on μp1 is *estimated*.

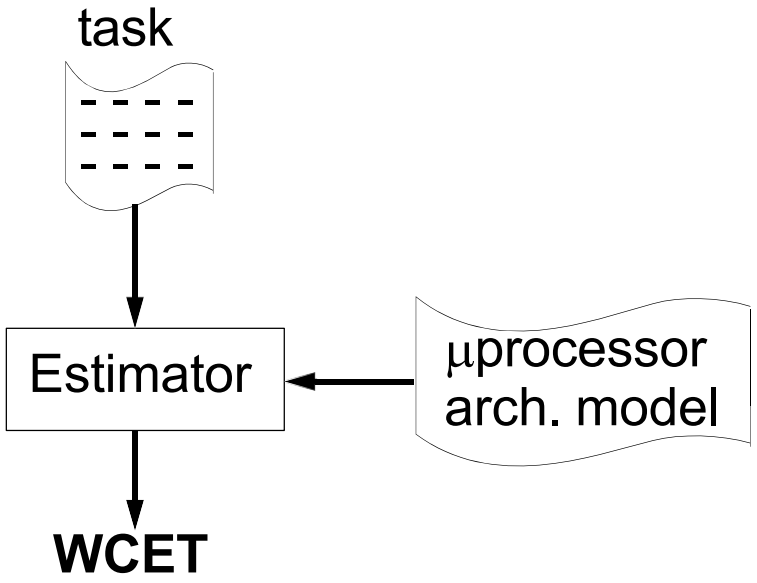


Example

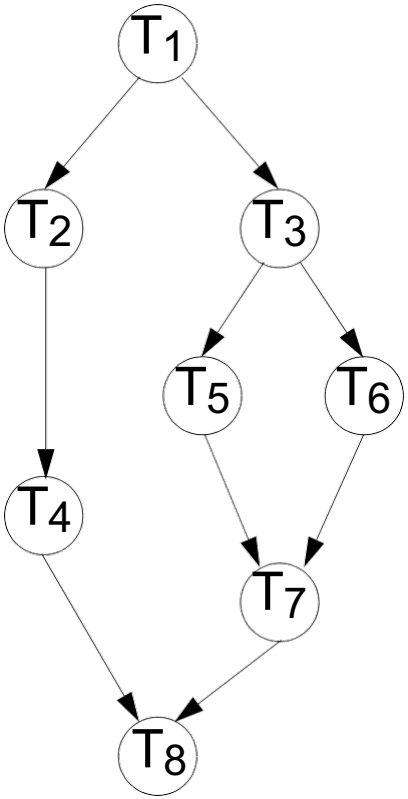


- We have the system model (task graph) which has been validated by simulation.
- We decide on a certain μprocessor μp1, with cost 6.
- For each task the worst case execution time (WCET) when run on μp1 is *estimated*.

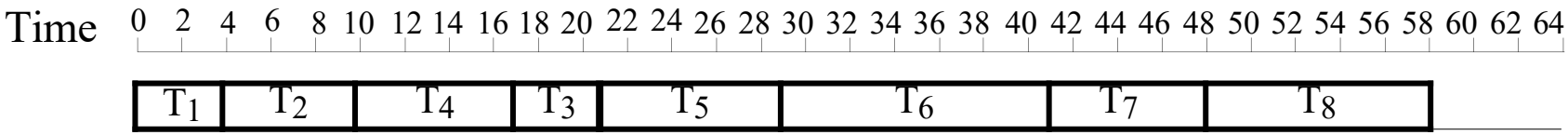
Task	WCET
T_1	4
T_2	6
T_3	4
T_4	7
T_5	8
T_6	12
T_7	7
T_8	10



Example

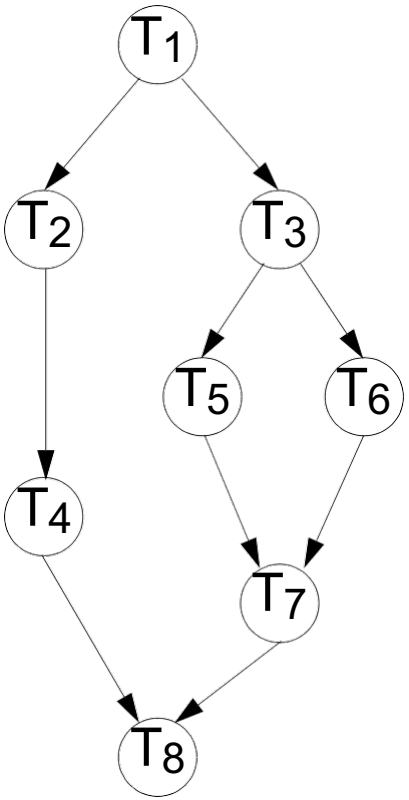


We generate a schedule:



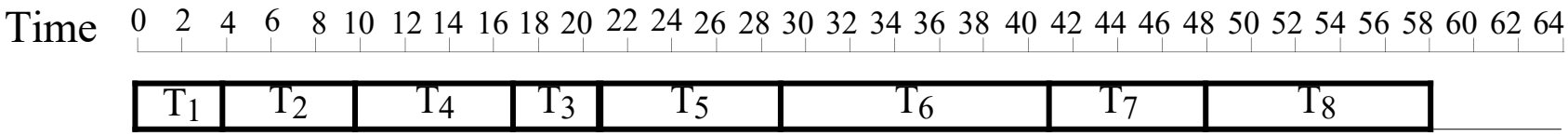
Task	WCET
T_1	4
T_2	6
T_3	4
T_4	7
T_5	8
T_6	12
T_7	7
T_8	10

Example



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

We generate a schedule:



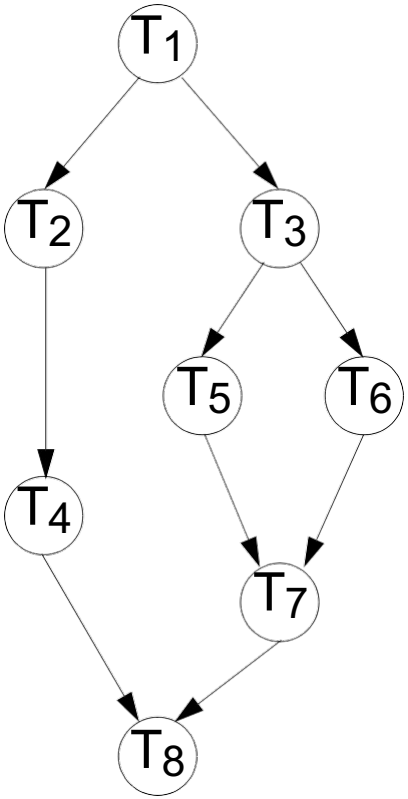
Using the architecture with μprocessor μp1 we got a solution with:

- **Execution time: 58 > 42**

- **Cost: 6 < 8**

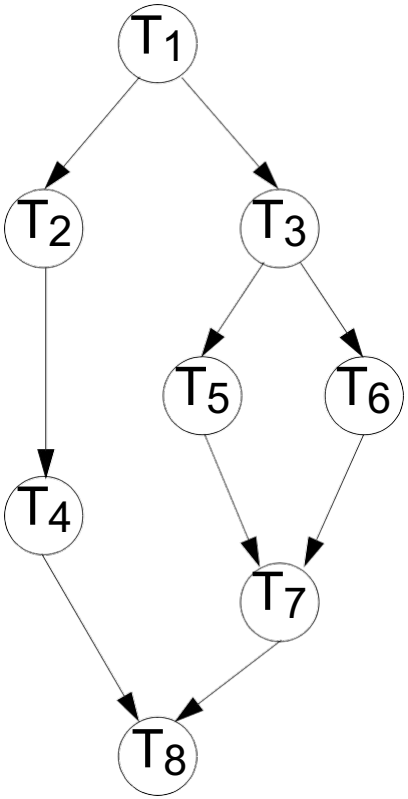
We have to try with another architecture!

Example



We look after a μprocessor which is fast enough: μp2

Example

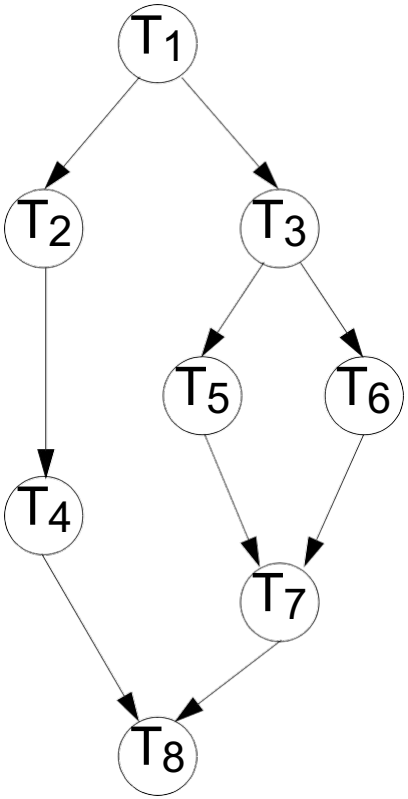


We look after a μprocessor which is fast enough: μp2

For each task the WCET, when run on μp2, 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



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

We look after a μprocessor which is fast enough: μp2

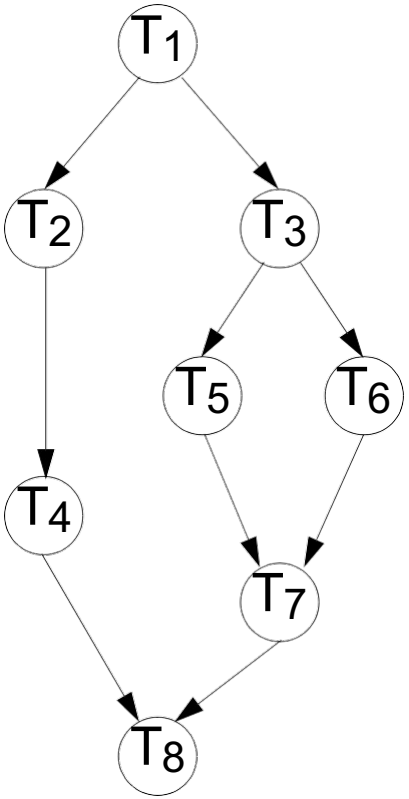
For each task the WCET, when run on μp2, is estimated.

Using the architecture with μprocessor μp2, after generating a schedule, we got a solution with:

- Execution time: 28 < 42
- Cost: 15 > 8 

We have to try with another architecture!

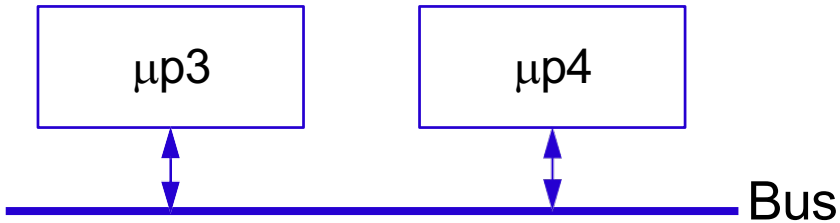
Example



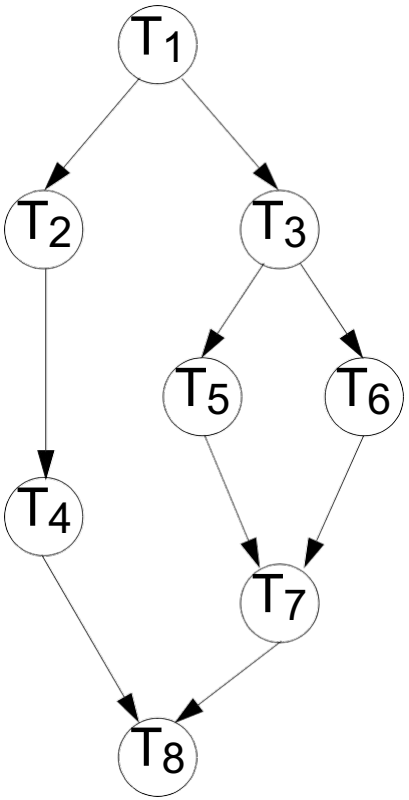
We have to look for a multiprocessor solution

- **In order to meet cost constraints try 2 cheap (and slow) μps:**

  μp3: cost 3
  μp4: cost 2
  interconnection bus: cost 1



Example

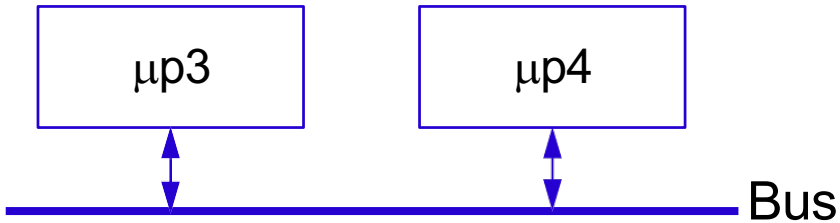


Task	WCET	
	μp3	μp4
T_1	5	6
T_2	7	9
T_3	5	6
T_4	8	10
T_5	10	11
T_6	17	21
T_7	10	14
T_8	15	19

We have to look for a multiprocessor solution

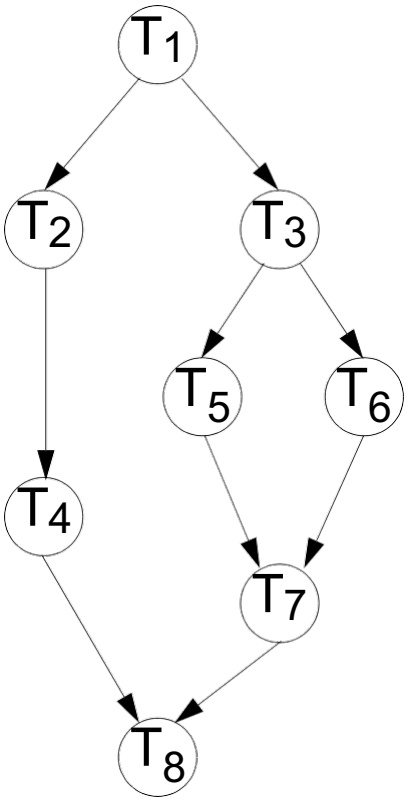
- **In order to meet cost constraints try 2 cheap (and slow) μps:**

  μp3: cost 3
  μp4: cost 2
  interconnection bus: cost 1



For each task the WCET, when run on μp3 and μp4, is estimated.

Example



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

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

μp3: T_1, T_3, T_5, T_6, T_7, T_8.
μp4: T_2, T_4.

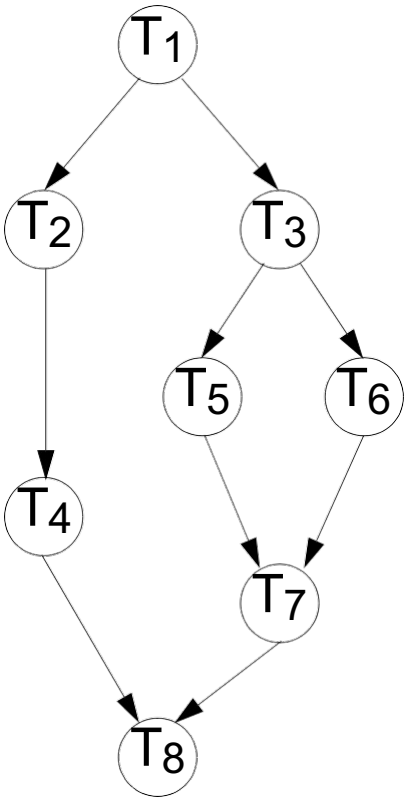
If communicating tasks are mapped to different processors, they have to communicate over the bus.

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:

$C_{1\text{-}2}$: 1
$C_{4\text{-}8}$: 1

Example



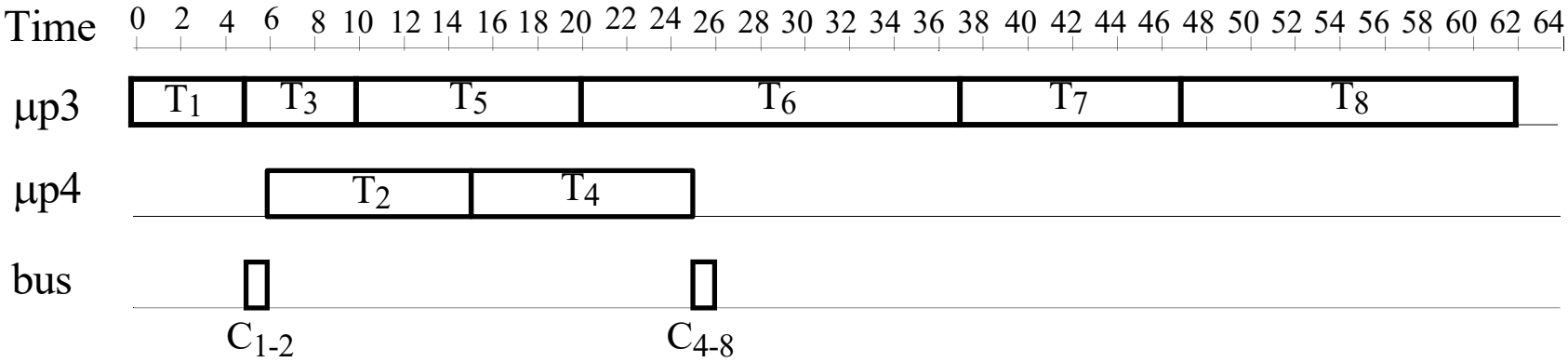
Task	WCET	
	μp3	μp4
T_1	5	6
T_2	7	9
T_3	5	6
T_4	8	10
T_5	10	11
T_6	17	21
T_7	10	14
T_8	15	19

μp3: T_1, T_3, T_5, T_6, T_7, T_8.
μp4: T_2, T_4.

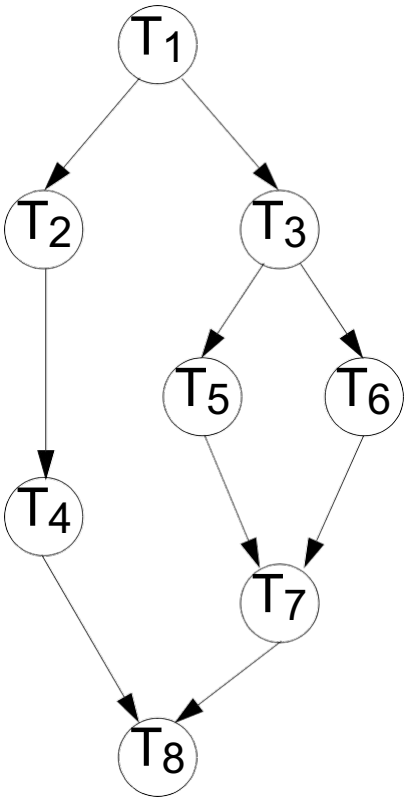
Estimated communication times:

$C_{1\text{-}2}$: 1
$C_{4\text{-}8}$: 1

We generate a schedule:



Example



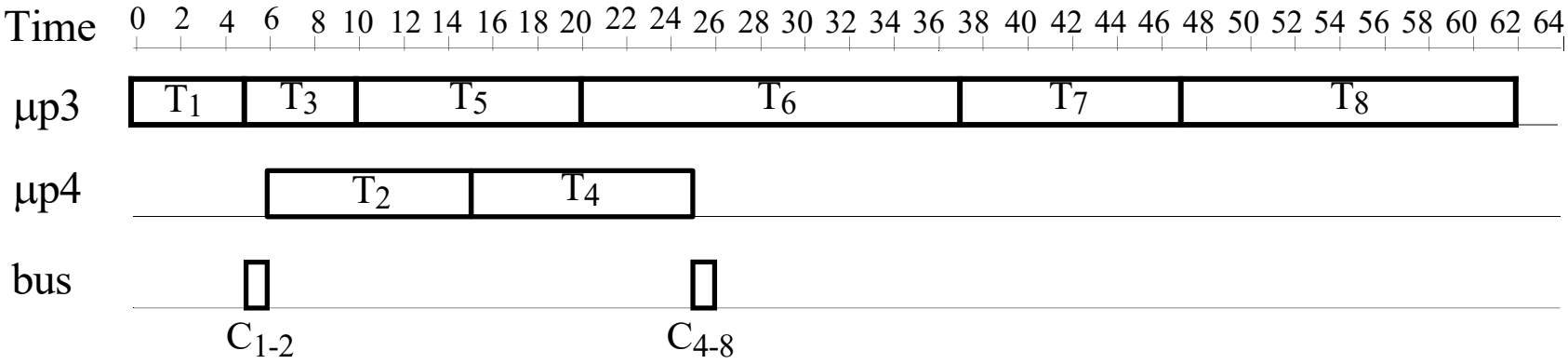
Task	WCET	
	μp3	μp4
T_1	5	6
T_2	7	9
T_3	5	6
T_4	8	10
T_5	10	11
T_6	17	21
T_7	10	14
T_8	15	19

μp3: T_1, T_3, T_5, T_6, T_7, T_8.
μp4: T_2, T_4.

Estimated communication times:

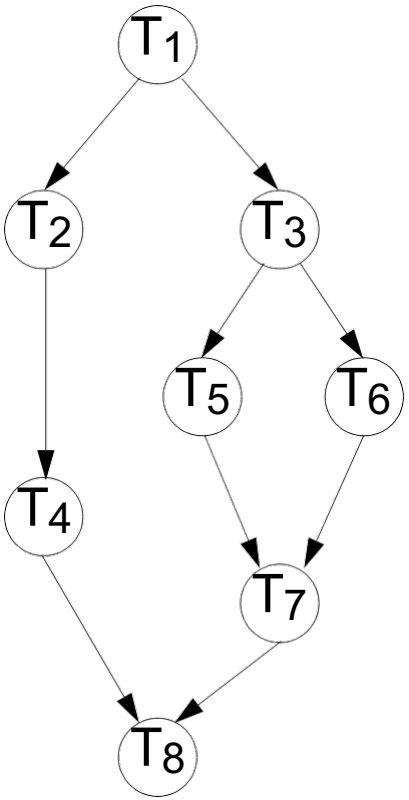
$C_{1\text{-}2}$: 1
$C_{4\text{-}8}$: 1

We generate a schedule:



We have exceeded the allowed execution time (42)!

Example



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

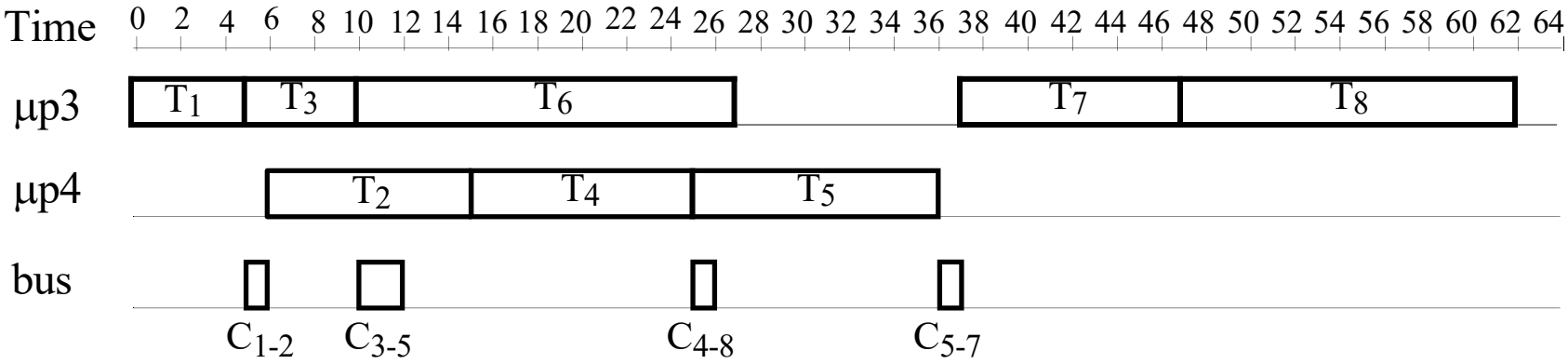
Try a new mapping; T_5 to μp4, in order to increase parallelism.

Two new communications are introduced, with estimated times:

$C_{3\text{-}5}$: 2

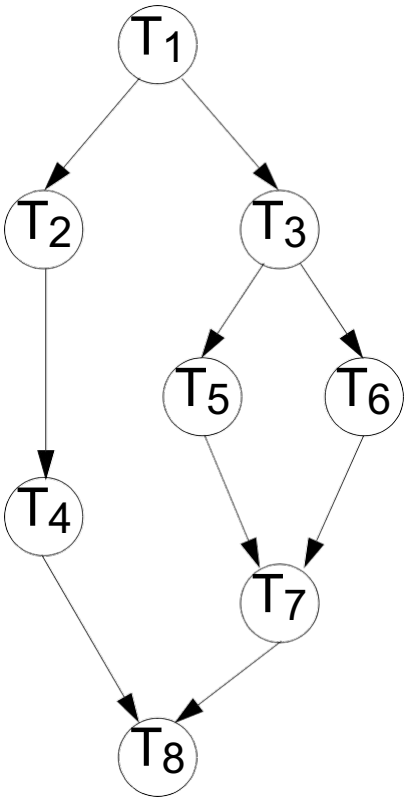
$C_{5\text{-}7}$: 1

We generate a schedule:



The execution time is still 62, as before!

Example



Try a new mapping; T_5 to μp4, in order to increase parallelism.

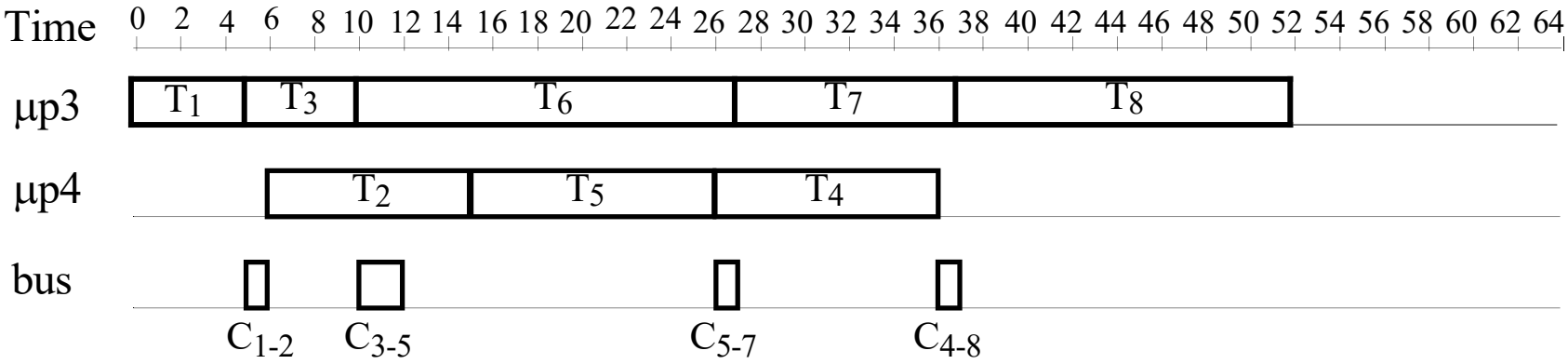
Two new communications are introduced, with estimated times:

$C_{3\text{-}5}$: 2

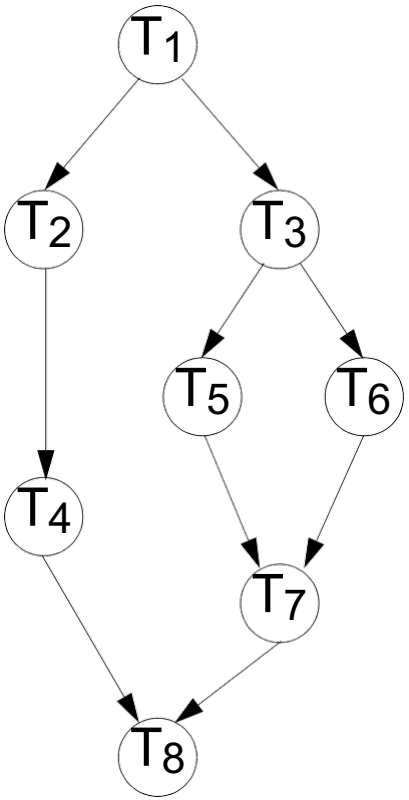
$C_{5\text{-}7}$: 1

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

There exists a better schedule!



Example



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

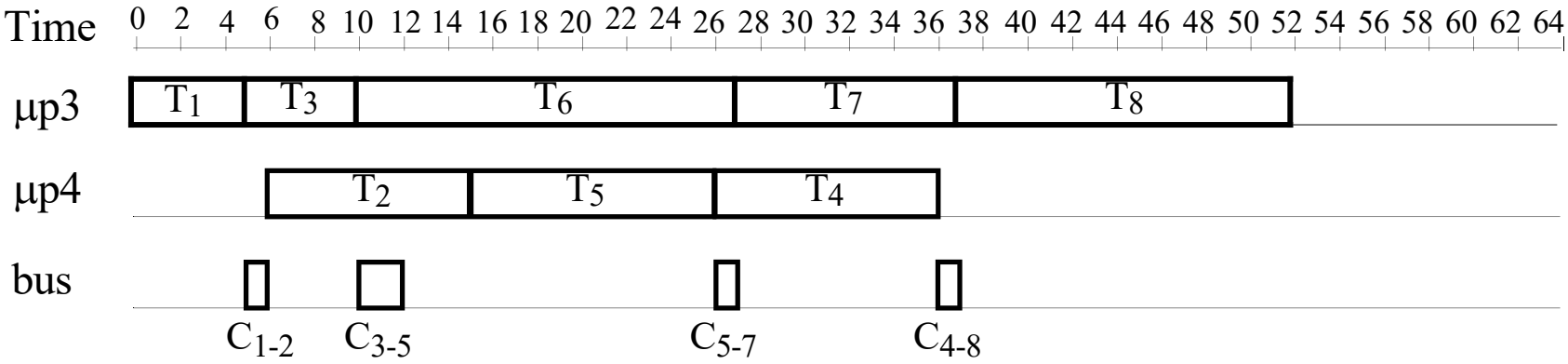
Try a new mapping; T_5 to μp4, in order to increase parallelism.

Two new communications are introduced, with estimated times:

$C_{3\text{-}5}$: 2

$C_{5\text{-}7}$: 1

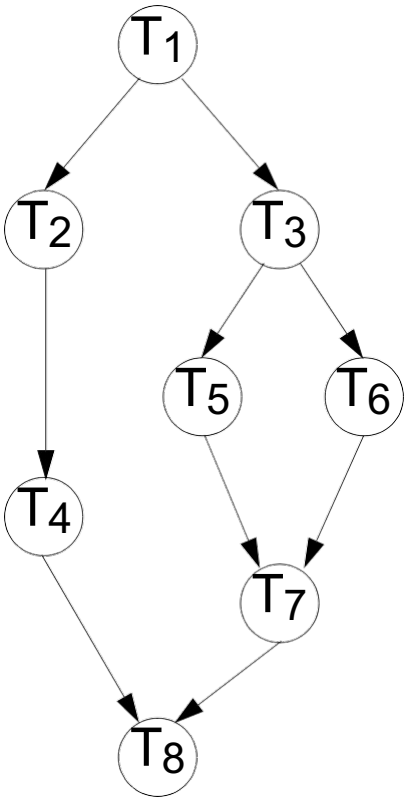
There exists a better schedule!

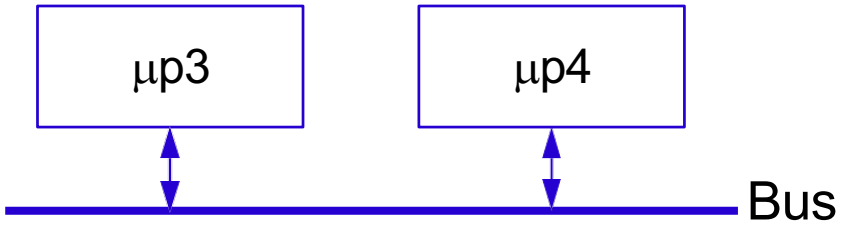


Execution time: 52 > 42

Cost: 6 < 8

Example

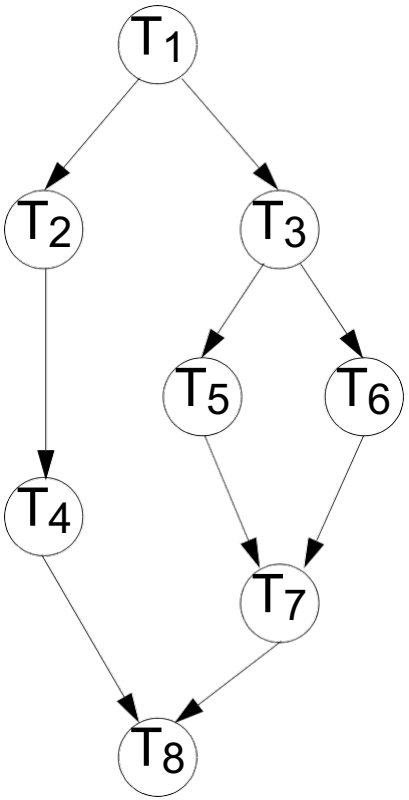


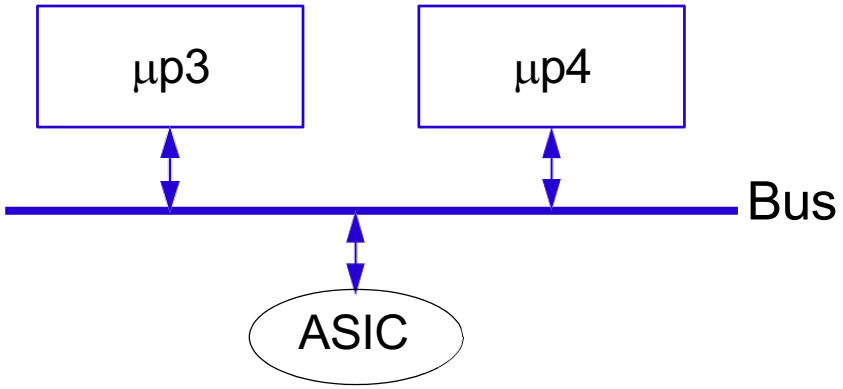


Task	WCET	
	µp3	µp4
T_1	5	6
T_2	7	9
T_3	5	6
T_4	8	10
T_5	10	11
T_6	17	21
T_7	10	14
T_8	15	19

- **Possible solutions:**
  - **Change µproc. µp3 with faster one ⇒ cost limits exceeded**

Example



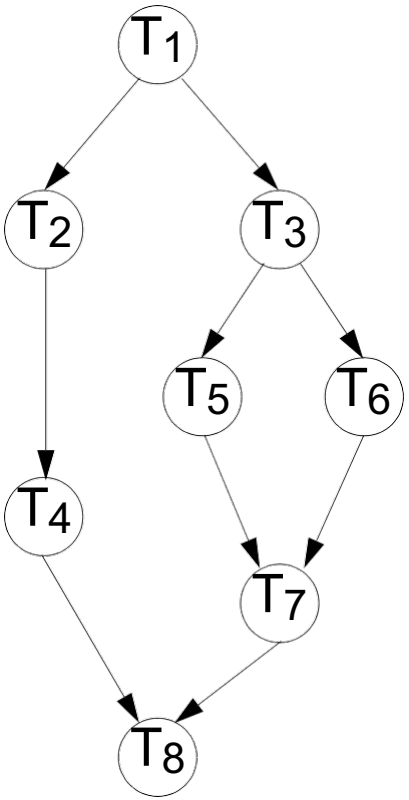


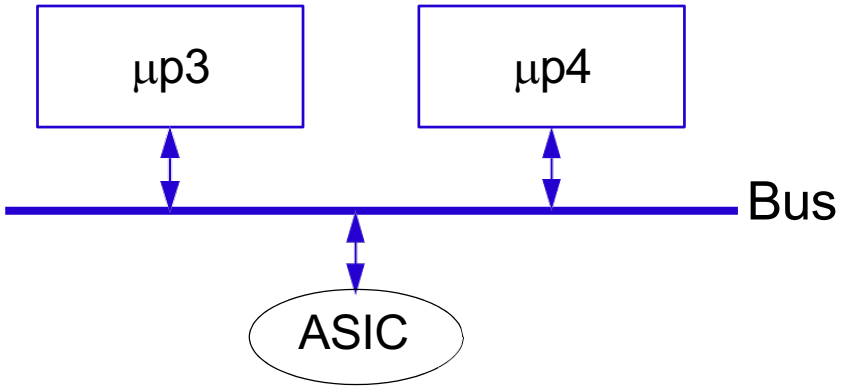
Task	WCET	
	µp3	µp4
T_1	5	6
T_2	7	9
T_3	5	6
T_4	8	10
T_5	10	11
T_6	17	21
T_7	10	14
T_8	15	19

- **Possible solutions:**
  - **Change µproc. µp3 with faster one ⇒ cost limits exceed**
  - **Implement part of the functionality in hardware as an ASIC**

    Cost of ASIC: 1

Example





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

- **Possible solutions:**
  - **Change µproc. µp3 with faster one ⇒ cost limits exceed**
  - **Implement part of the functionality in hardware as an ASIC**
- **New architecture**

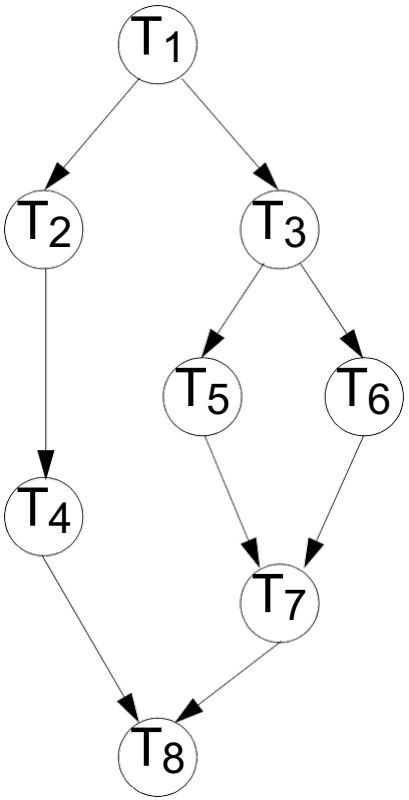
  Cost of ASIC: 1
- **Mapping**

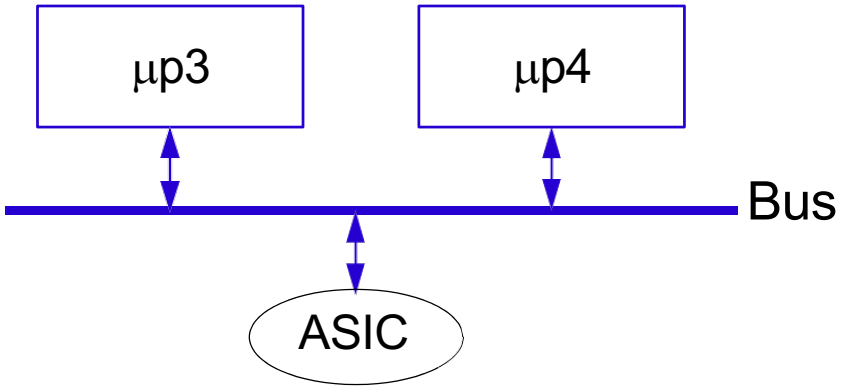
  µp3: T_1, T_3, T_6, T_7.

  µp4: T_2, T_4, T_5.

  ASIC: T_8 with estimated WCET= 3
  - **New communication, with estimated time: $C_{7\text{-}8}$: 1**

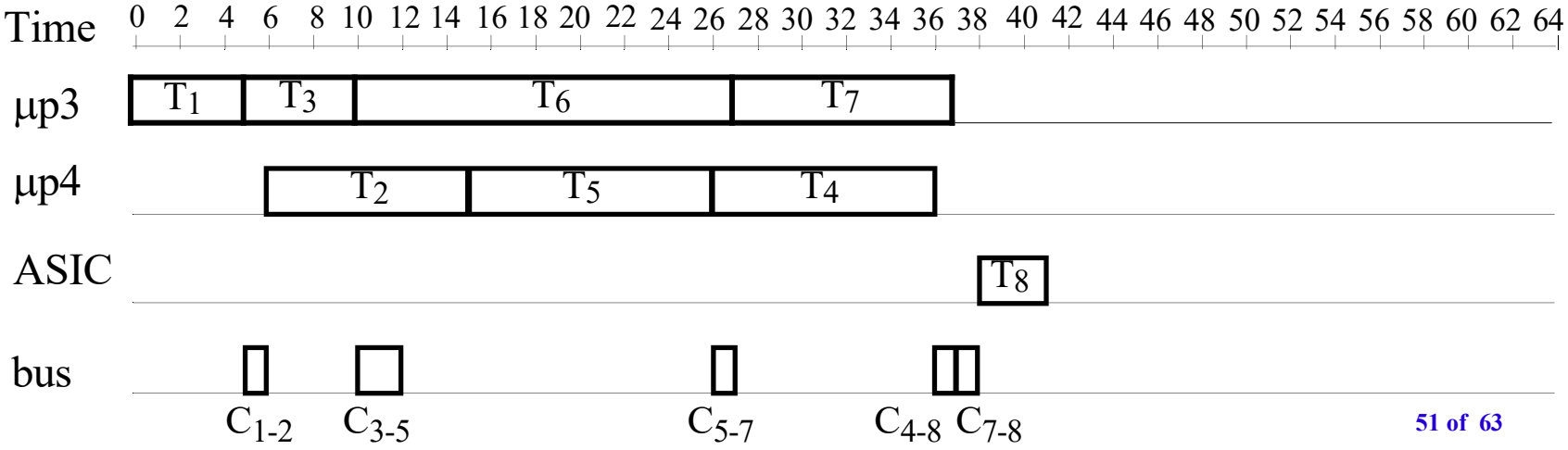
Example



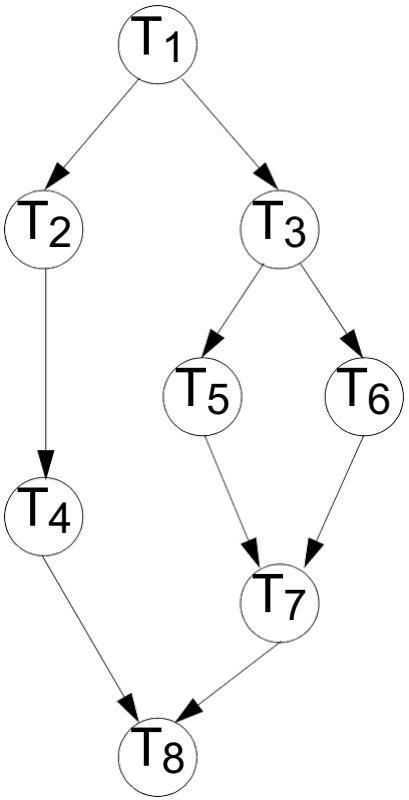


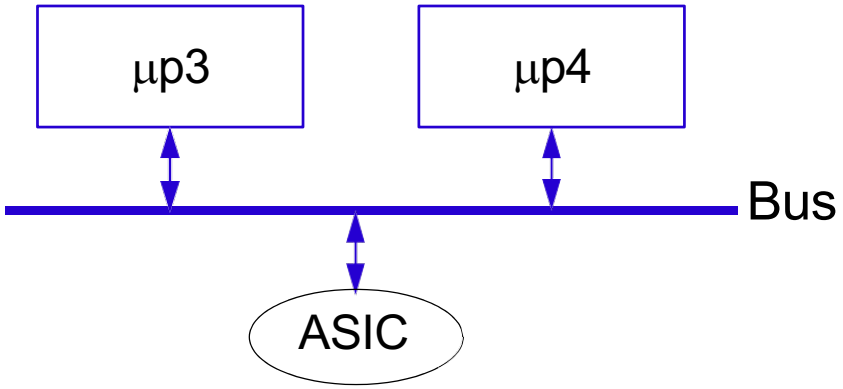
Task	WCET	
	µp3	µp4
T_1	5	6
T_2	7	9
T_3	5	6
T_4	8	10
T_5	10	11
T_6	17	21
T_7	10	14
T_8	15	19

- **Mapping**
  - **µp3: T_1, T_3, T_6, T_7.**
  - **µp4: T_2, T_4, T_5.**
  - **ASIC: T_8 with estimated WCET= 3**
    - **New communication, with estimated time: $C_{7\text{-}8}$: 1**



Example

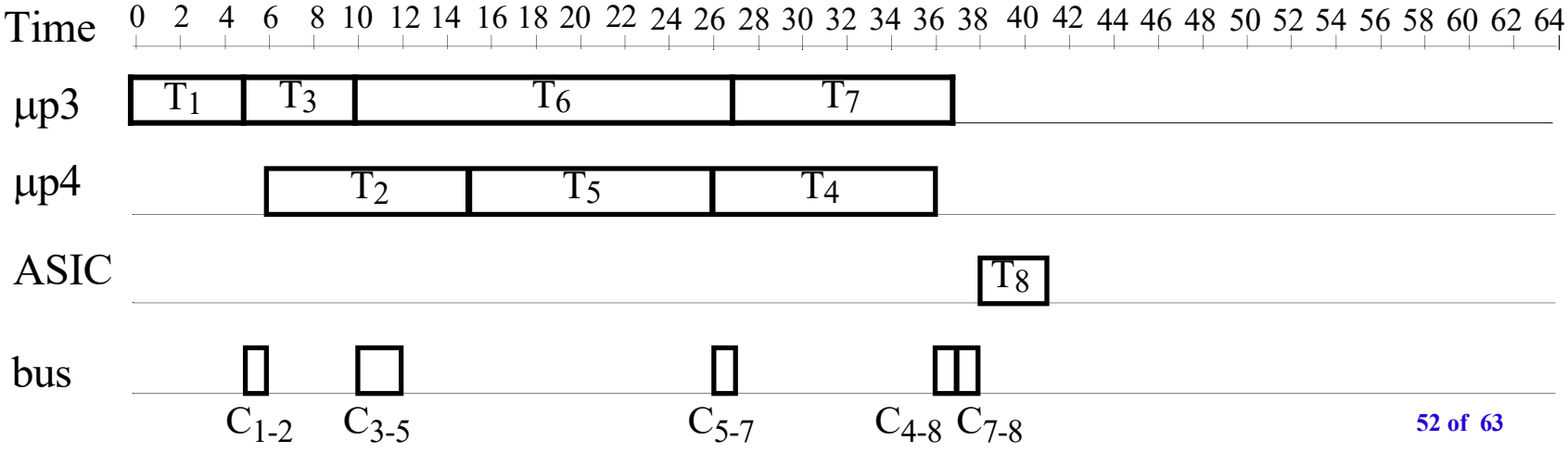




Task	WCET	
	µp3	µp4
T_1	5	6
T_2	7	9
T_3	5	6
T_4	8	10
T_5	10	11
T_6	17	21
T_7	10	14
T_8	15	19

Using this architecture we got a solution with:

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



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?

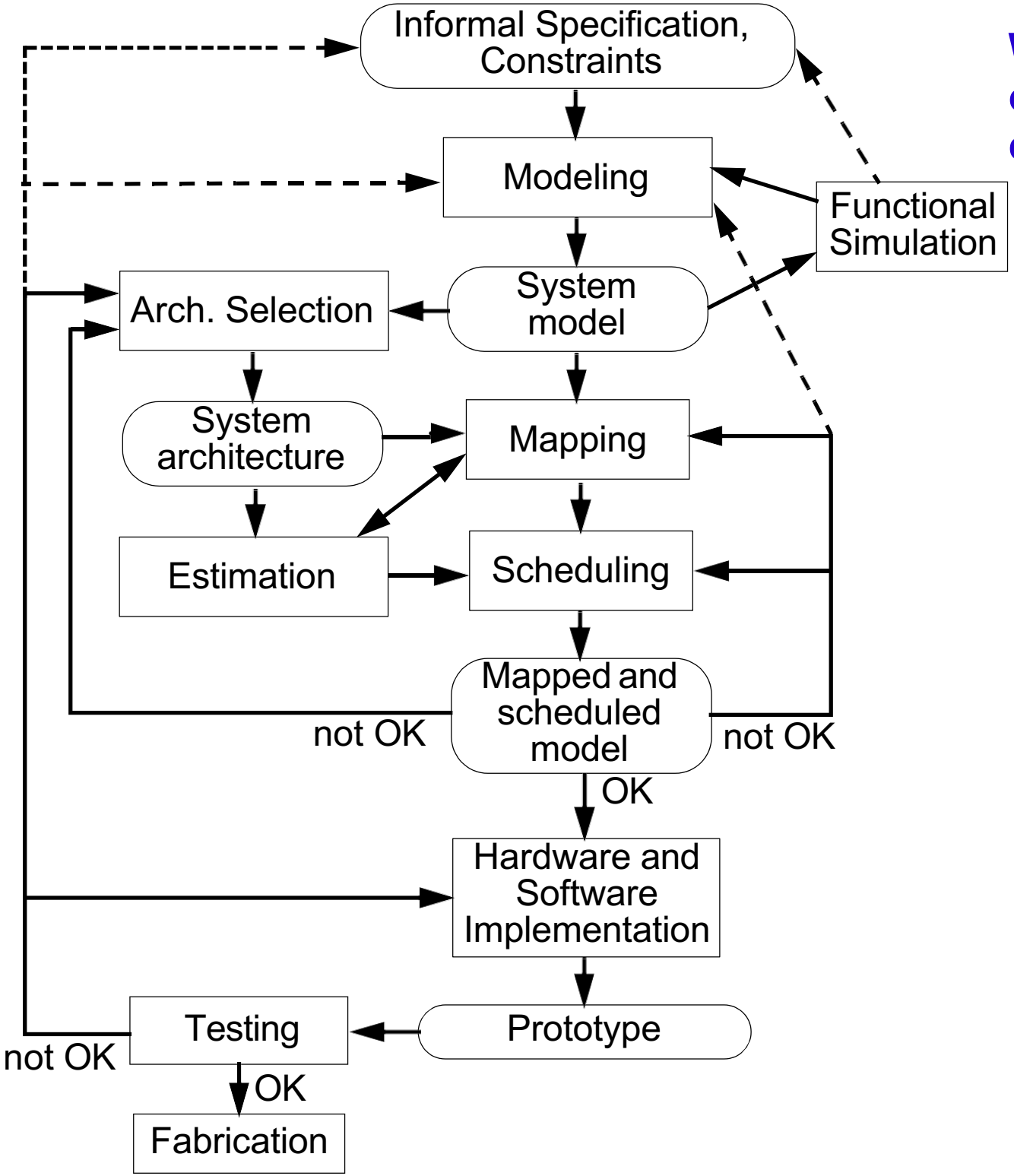
- We have selected an architecture.
- We have mapped tasks to the processors and ASIC.
- We have elaborated a a schedule.

Extremely important!!!

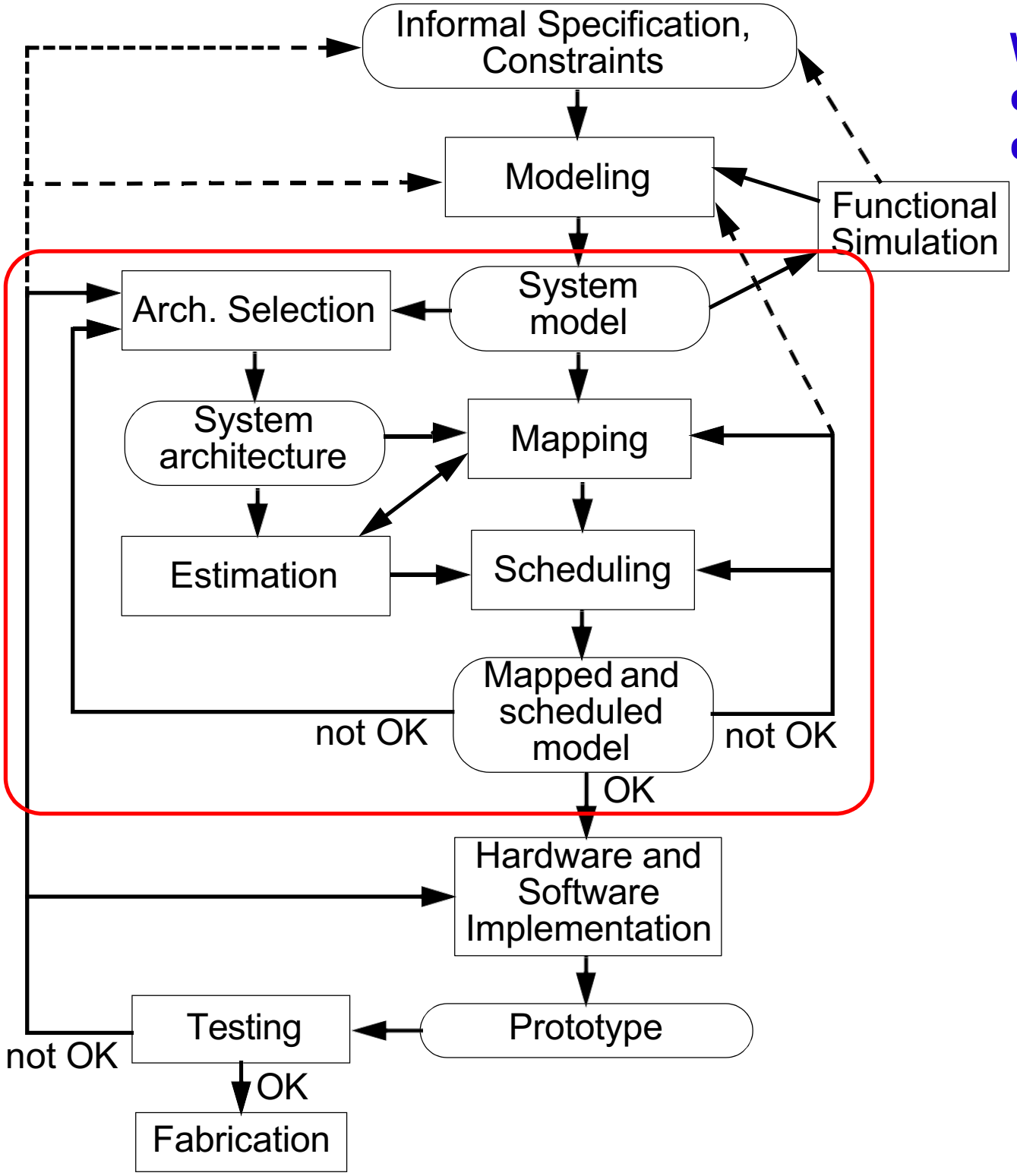
Nothing has been built yet.

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.**

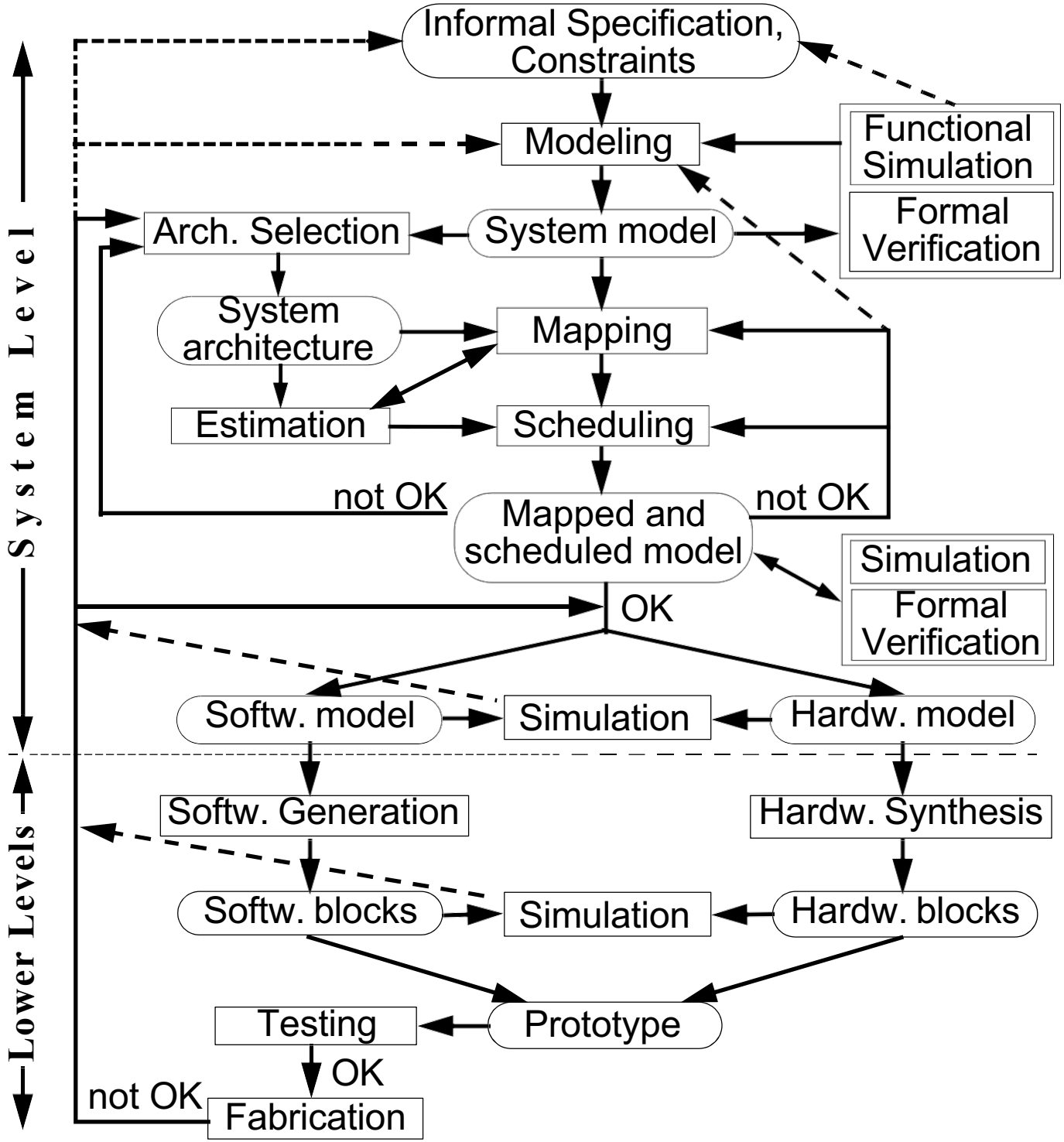
  This loop is performed several times as part of the *design space exploration*. Different architectures, mappings and schedules are explored, before the actual implementation and prototyping.

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

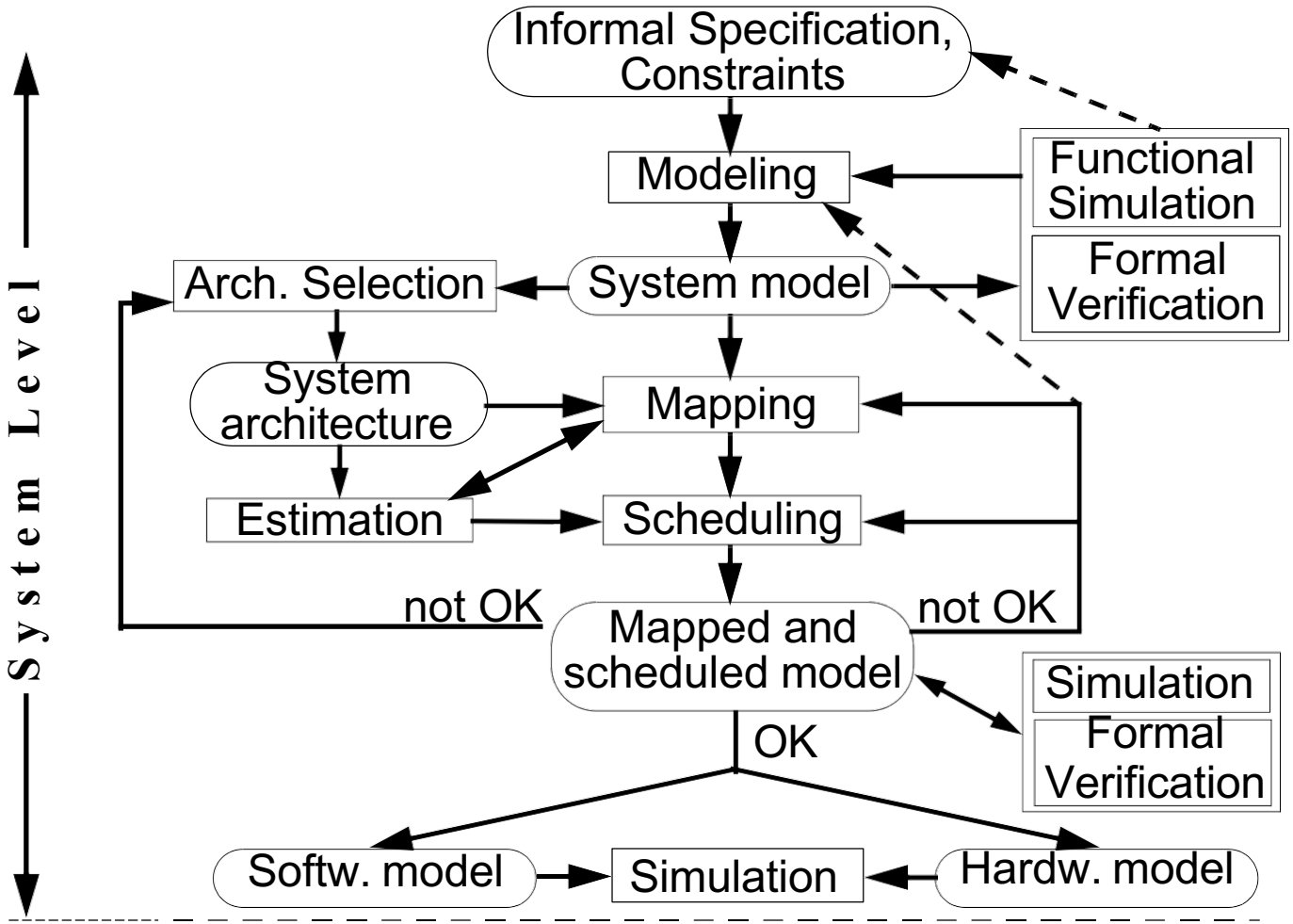
The Design Flow

- 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)


Informal Specification, Constraints
Modeling
Functional Simulation
Formal Verification
Arch. Selection
System model
System architecture
Mapping
Estimation
Scheduling
not OK
not OK
Mapped and scheduled model
Simulation
Formal Verification
OK
Softw. model
Simulation
Hardw. model
Softw. Generation
Hardw. Synthesis
Softw. blocks
Simulation
Hardw. blocks
Testing
Prototype
OK
not OK
Fabrication
System Level
Lower Levels

System Level Design Flow



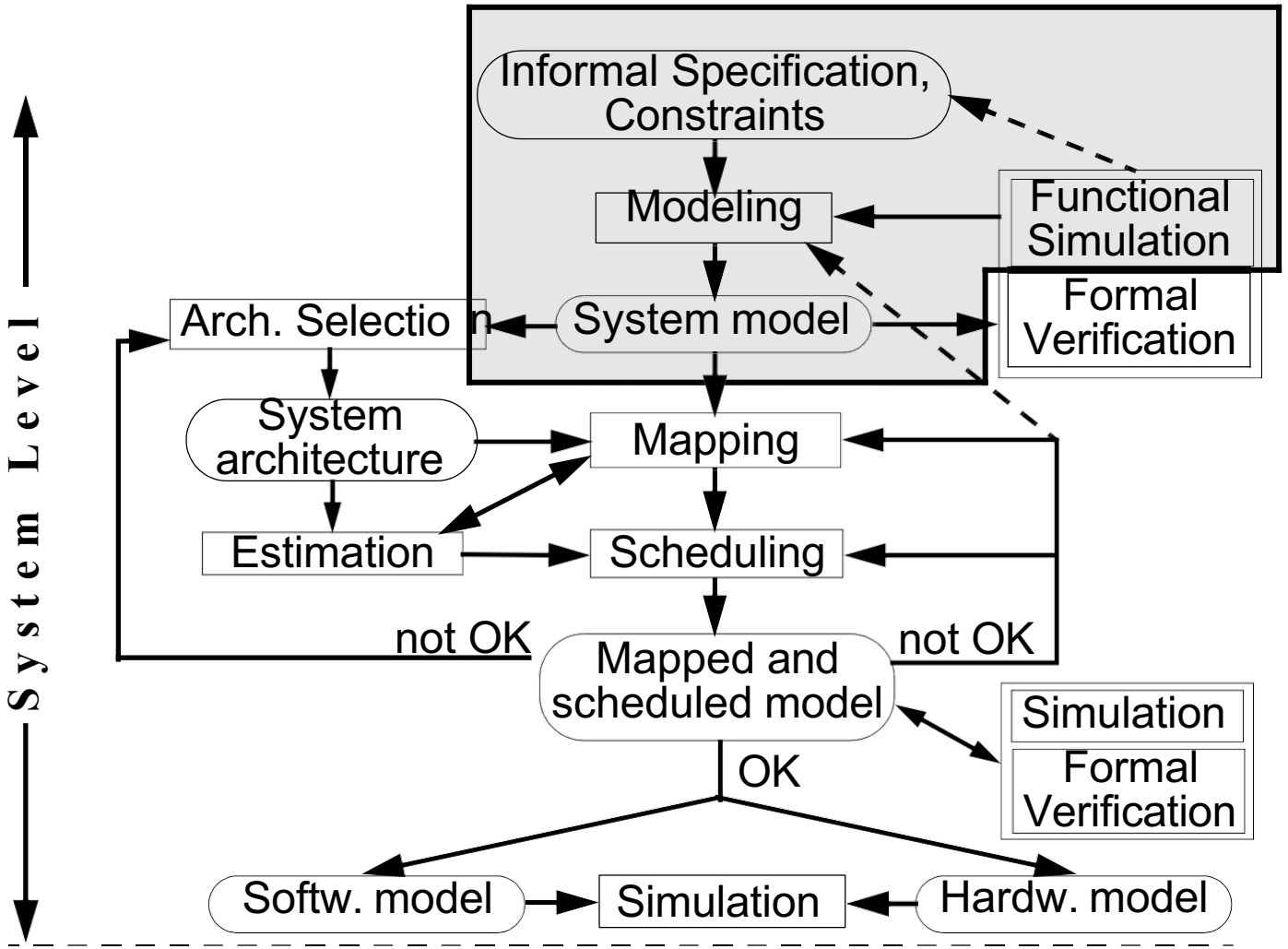
This is what we are interested in, in this course!

Course Topics at a Glance

- Introduction: Embedded Systems and Their Design (*just finished!*)
- Models of Computation and Specification Languages
  - Dataflow Models, Petri Nets, Discrete Event Models, Synchronous Finite State Machines & Synchronous Languages, Globally Asynchronous Locally Synchronous Systems, Timed Automata, Hybrid Automata.
- Architectures and Platforms for Embedded Systems Design
  - General Purpose vs. Application Specific Architectures, Component and Platform-based Design, Reconfigurable Systems, Functionality Mapping.
- Real-Time Embedded Systems
- System-Level Power/Energy Optimization

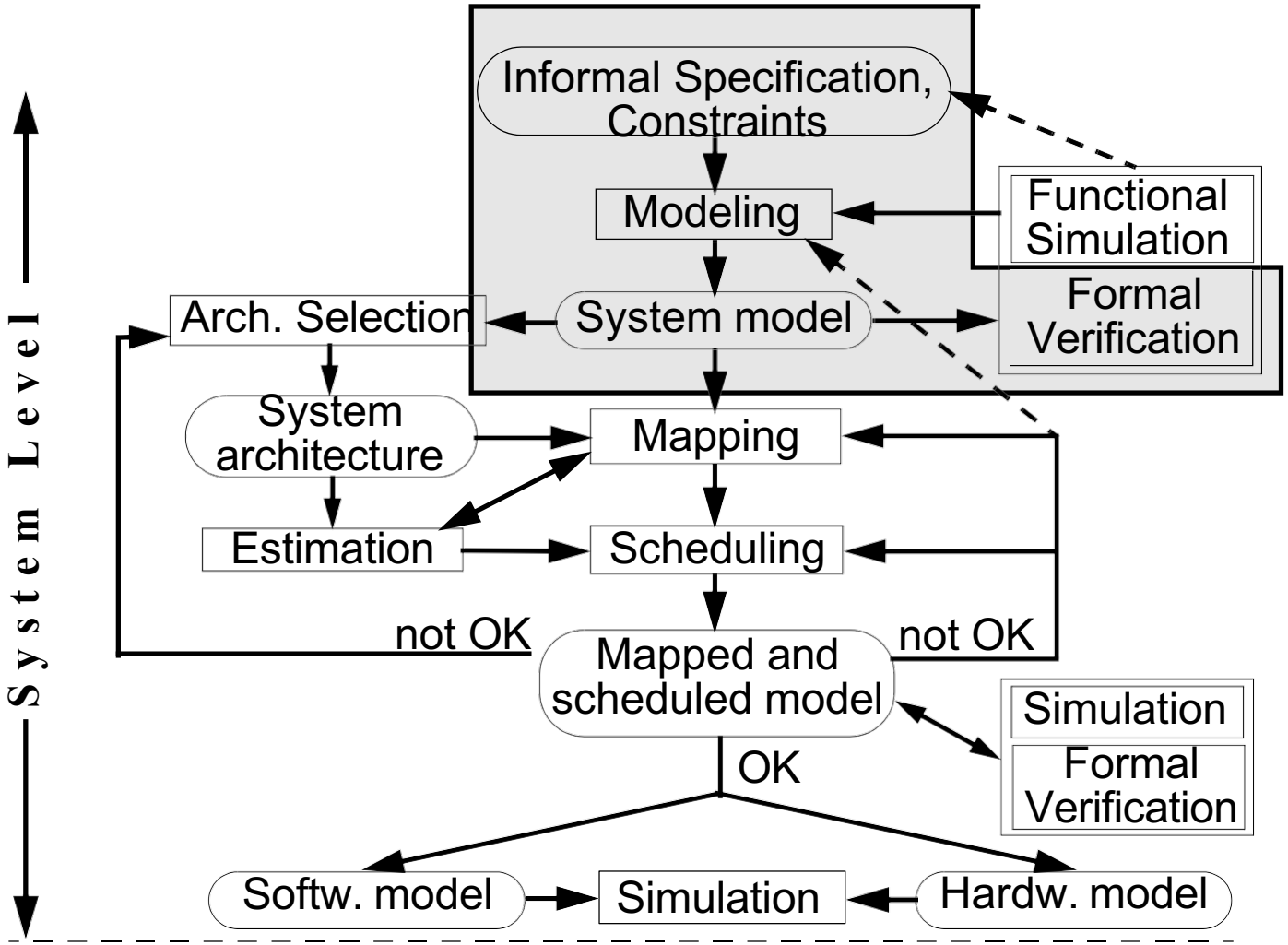
Lab Assignment 1

- **Modeling and simulation with System C**



Lab Assignment 2

- **Formal verification with UPPAAL (TDTS07 only)**



Lab Assignment 3

- **Design space exploration with an MPARM simulator.**