Distributed Systems for Real-Time Applications: Analysis and Synthesis

Petru Eles

Department of Computer and Information Science (IDA)
Linköpings universitet
http://www.ida.liu.se/~petel/


Linköping
Stockholm

- **Linköping University**

  23000 students, 1400 PhD students, 3500 staff&faculty

  - **Department of Computer and Information Science**

    207 employees

    16 professors, 23 associate & assistant professors

    90 PhD students

    - **Embedded Systems Laboratory**

      2 professors

      1 assistant professor

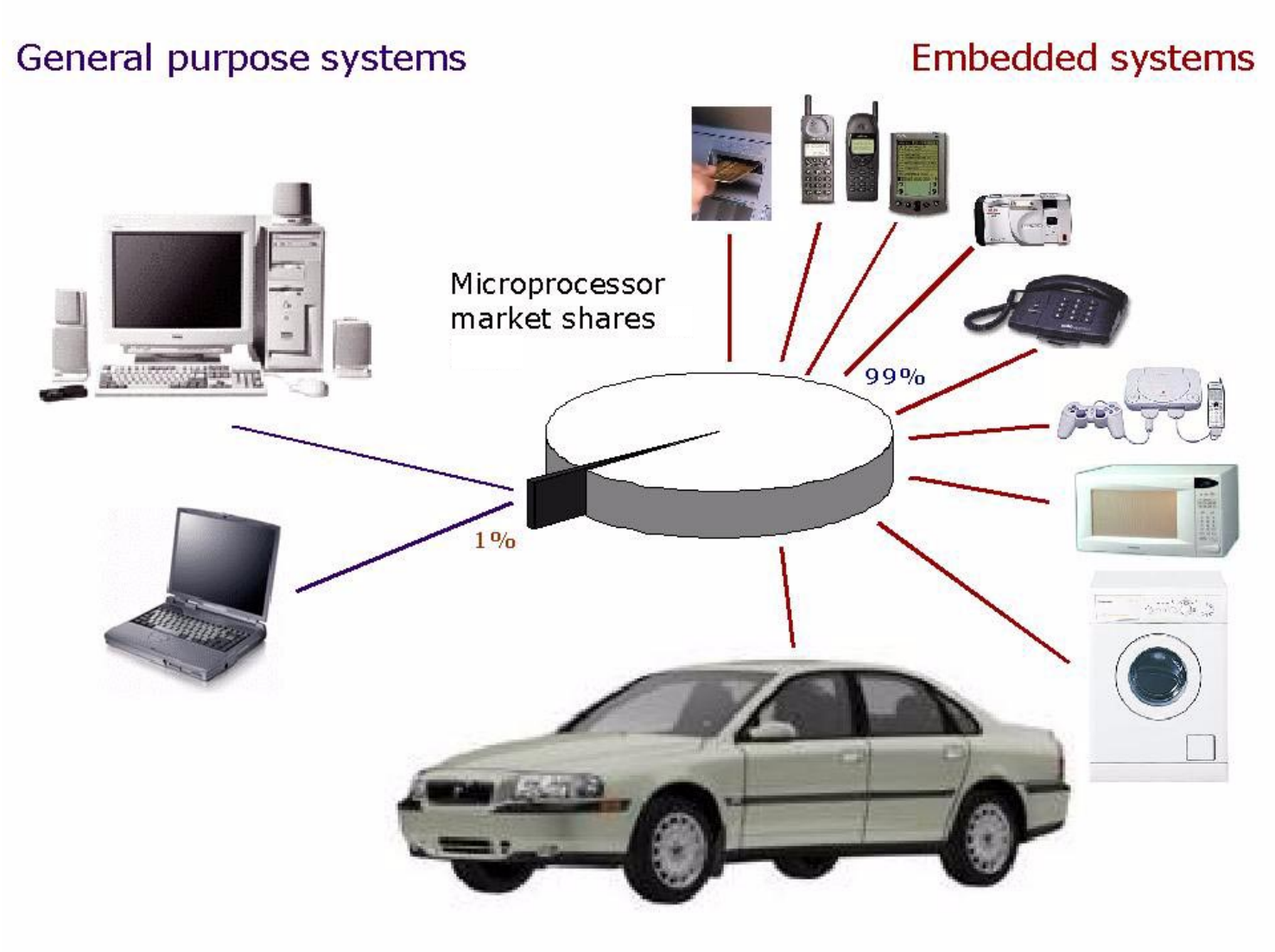
      10 PhD students

- National Graduate School in Computer Science
- Socware - the Swedish Systems-on-Chip initiative
- Centres of excellence
  - Stringent - the Strategic Integrated Electronic Systems Research Center at LiTH
  - ISIS Information Systems for Industrial Control and Supervision

Outline

- Embedded Real-Time System
- System-level Design Flow
- Distributed Embedded Real-Time Systems
  - Application Model
  - Heterogeneous Systems
  - Time/Event Triggered Tasks
  - Static/Dynamic Communication
  - Analysis&Optimization
- Single/Multi-cluster Heterogeneous Distributed Architectures
  - Analysis&Optimization
- Incremental Design Process

Embedded Real-Time Systems



Embedded Real-Time Systems

- Dedicated (not *general purpose*)
- Interact with the environment
- Consist of a collection of
  - programmable parts
  - ASICs and other standard components
  - sensors and actuators

What Makes Them Different?

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

But, in addition to this:

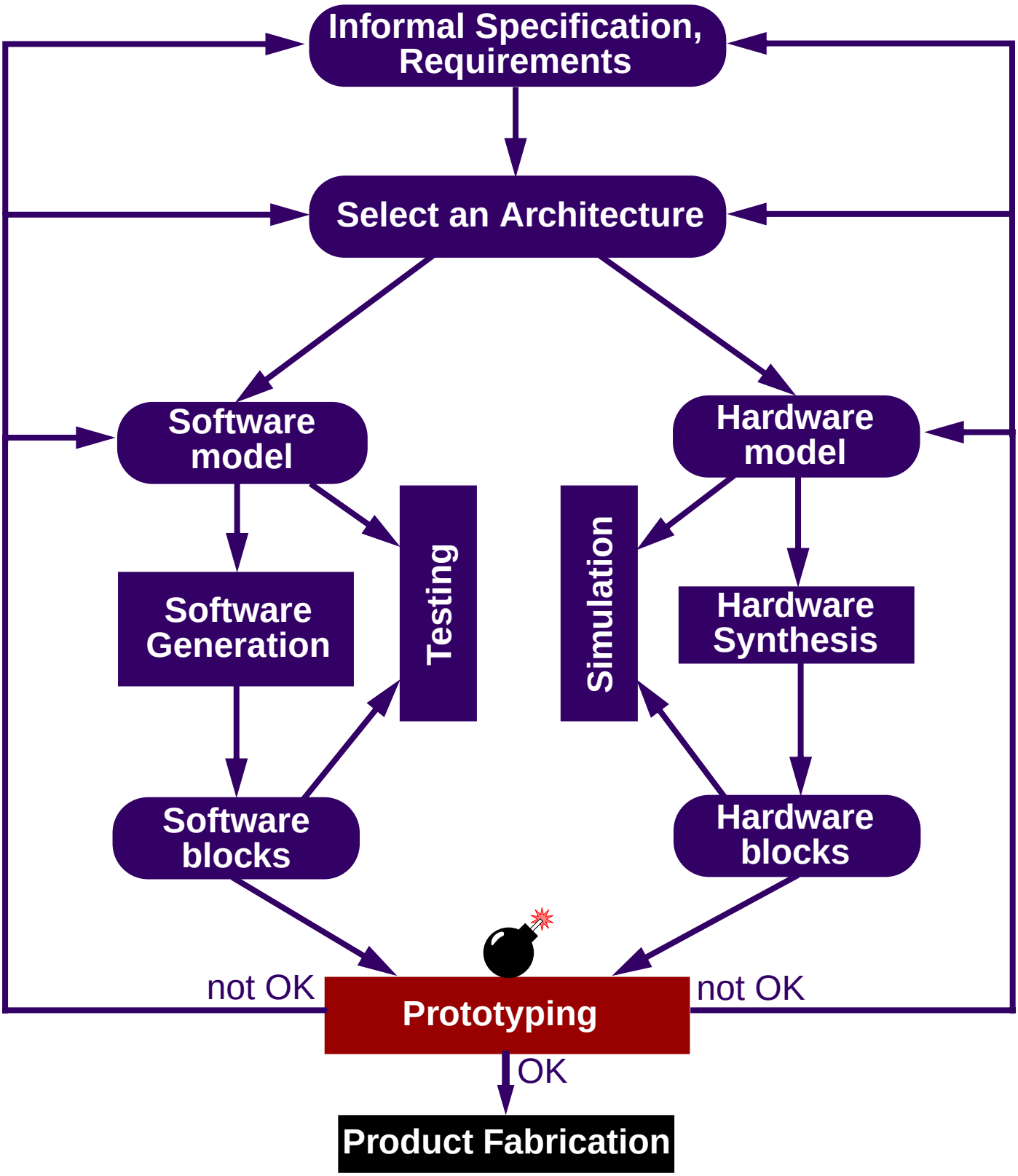
- Time constraints
- Power constraints
- Cost constraints
- Safety
- Time to market

Why Is the Design of Embedded Systems Difficult?

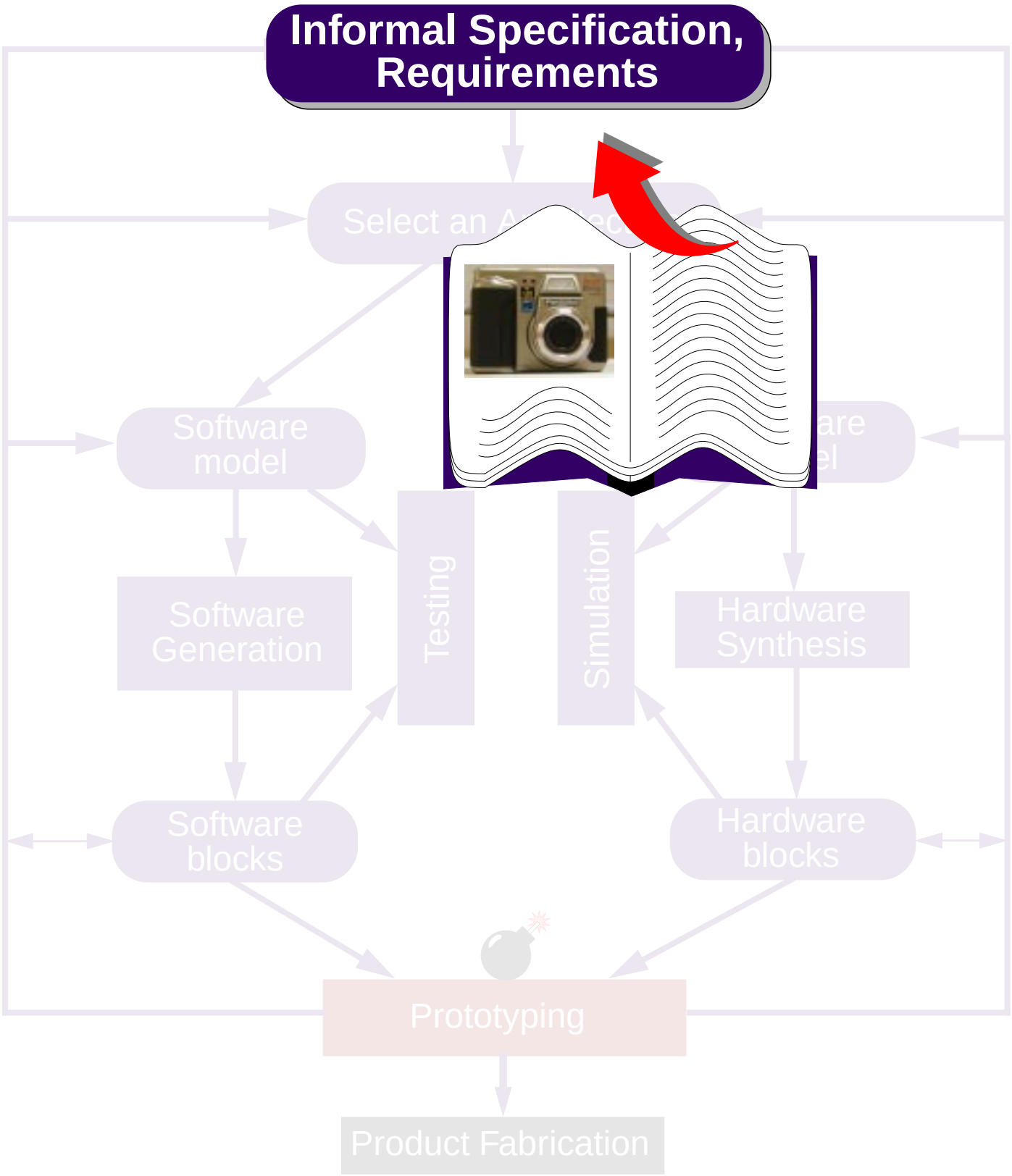
- In order to achieve all these constraints, systems have to be
  - highly optimized
  - verified for safety
  - delivered in (short) time

☞ Both hardware and software aspects have to be considered simultaneously!

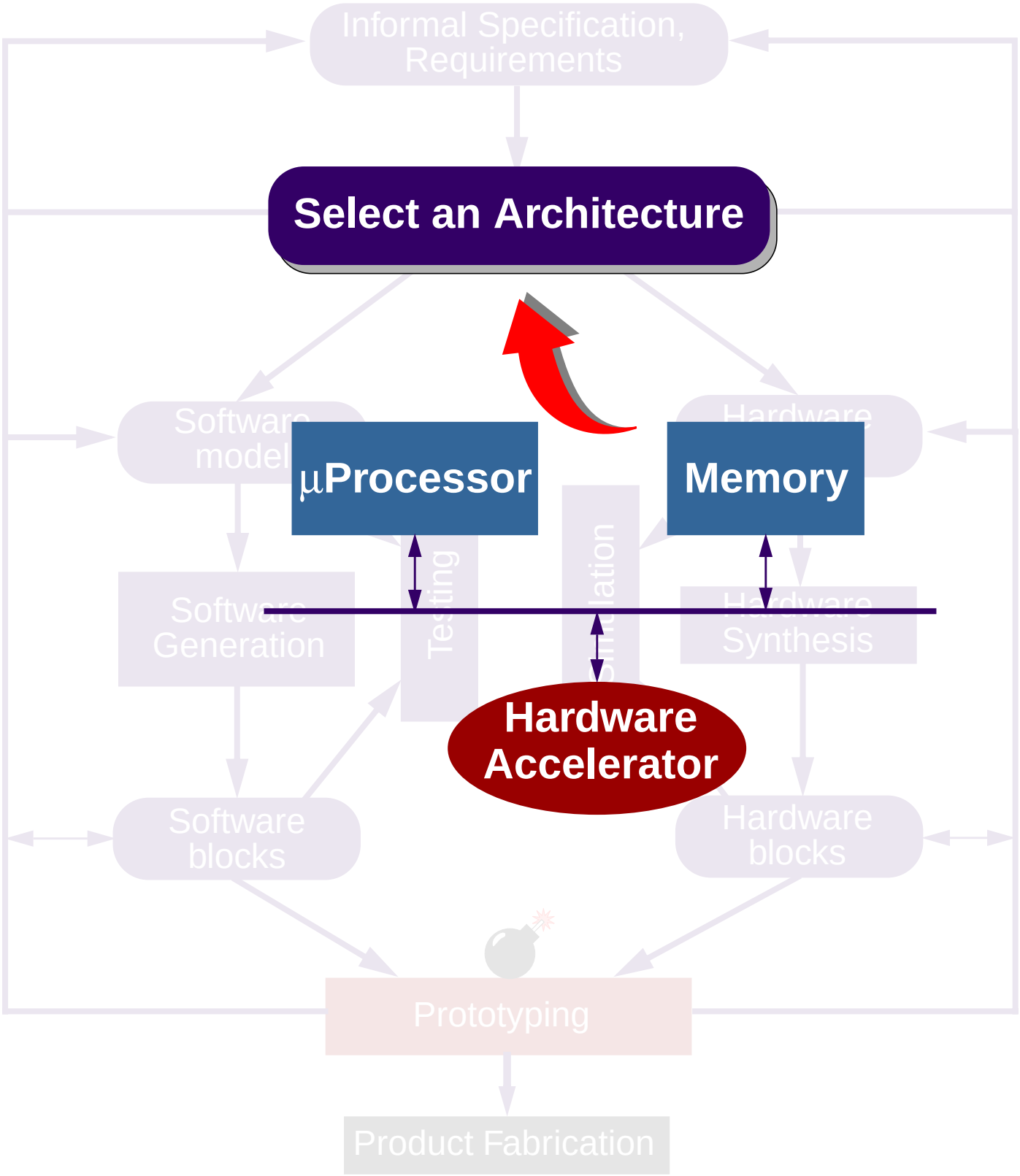
A Design Flow



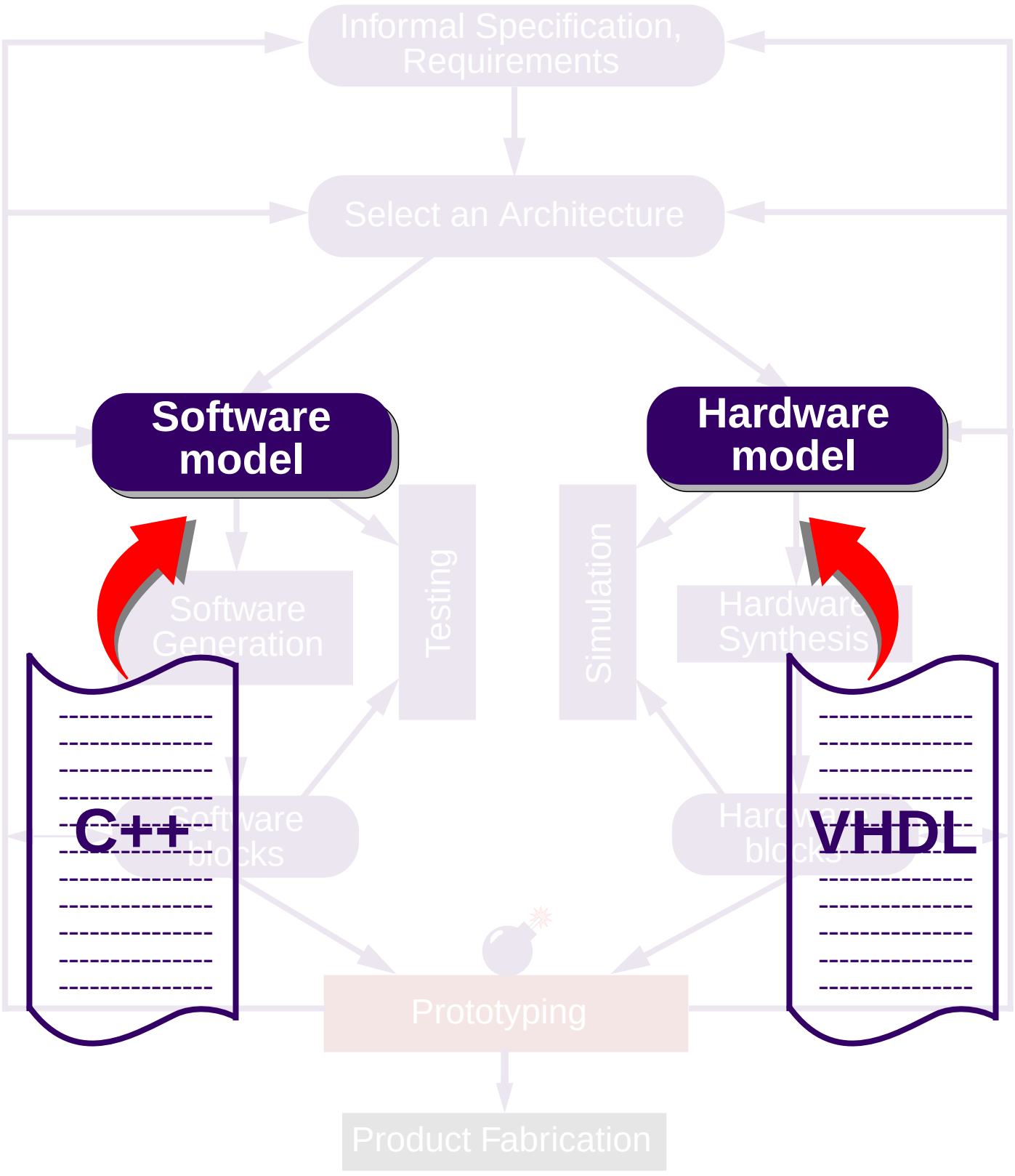
A Design Flow



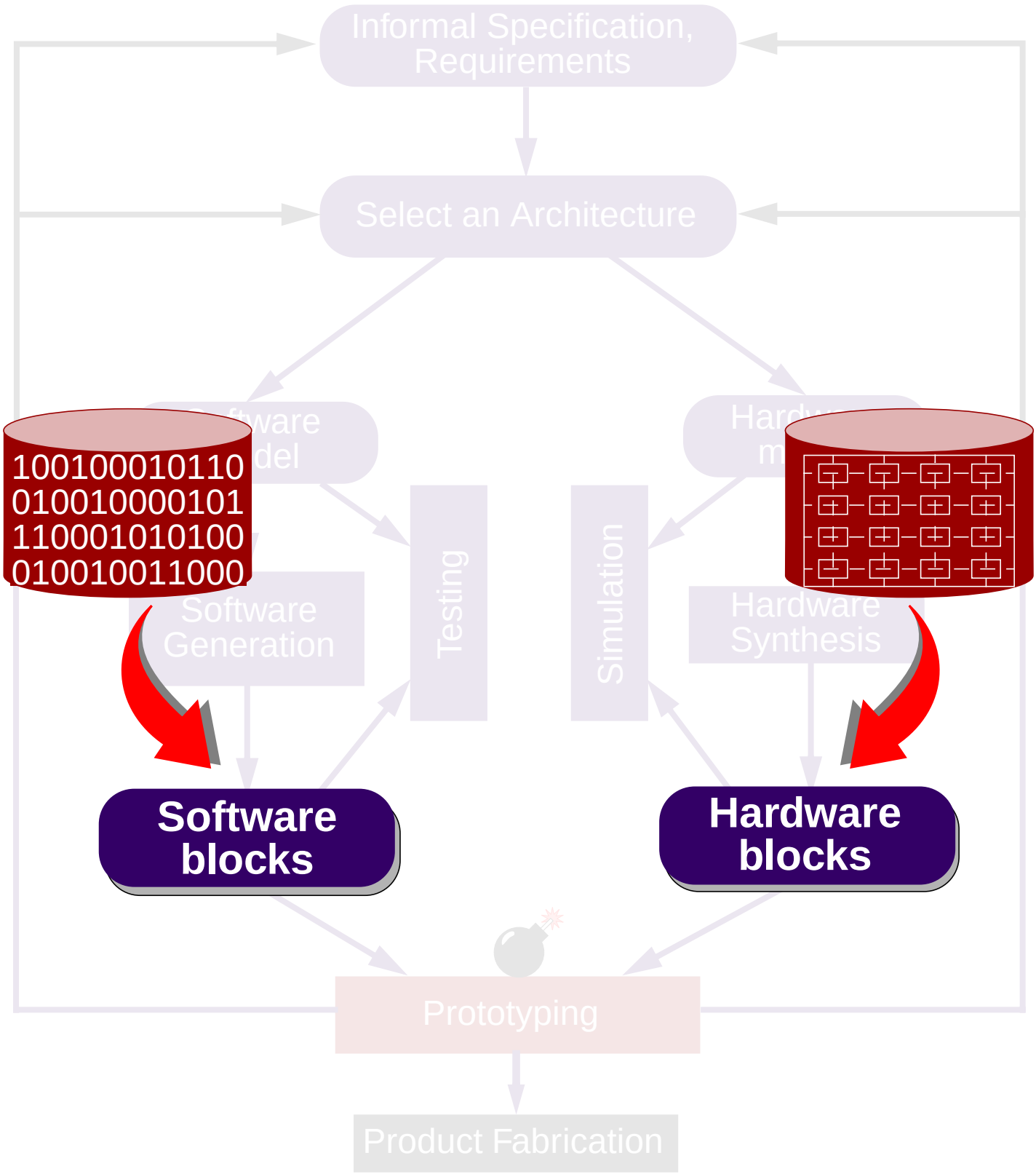
A Design Flow



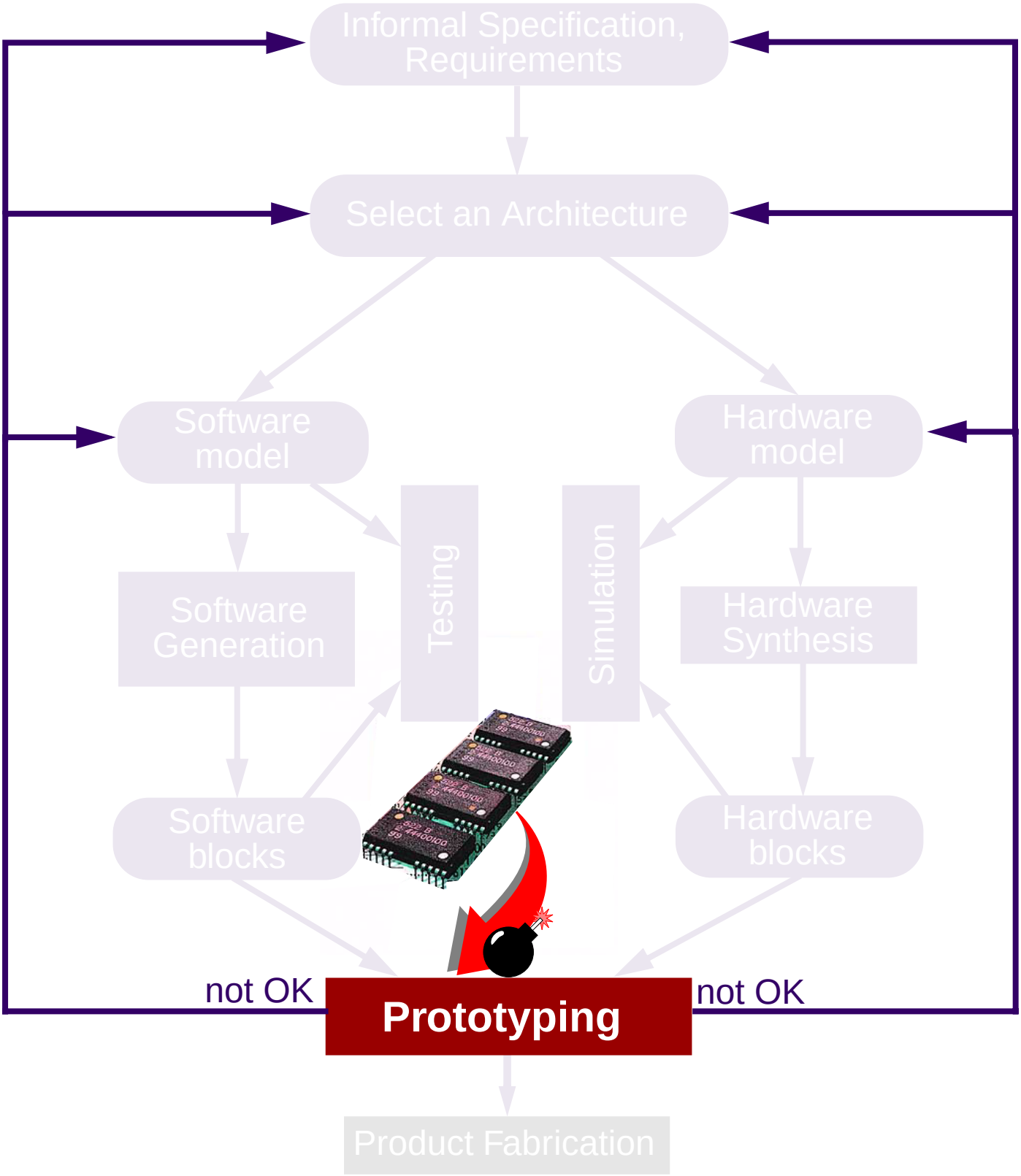
A Design Flow



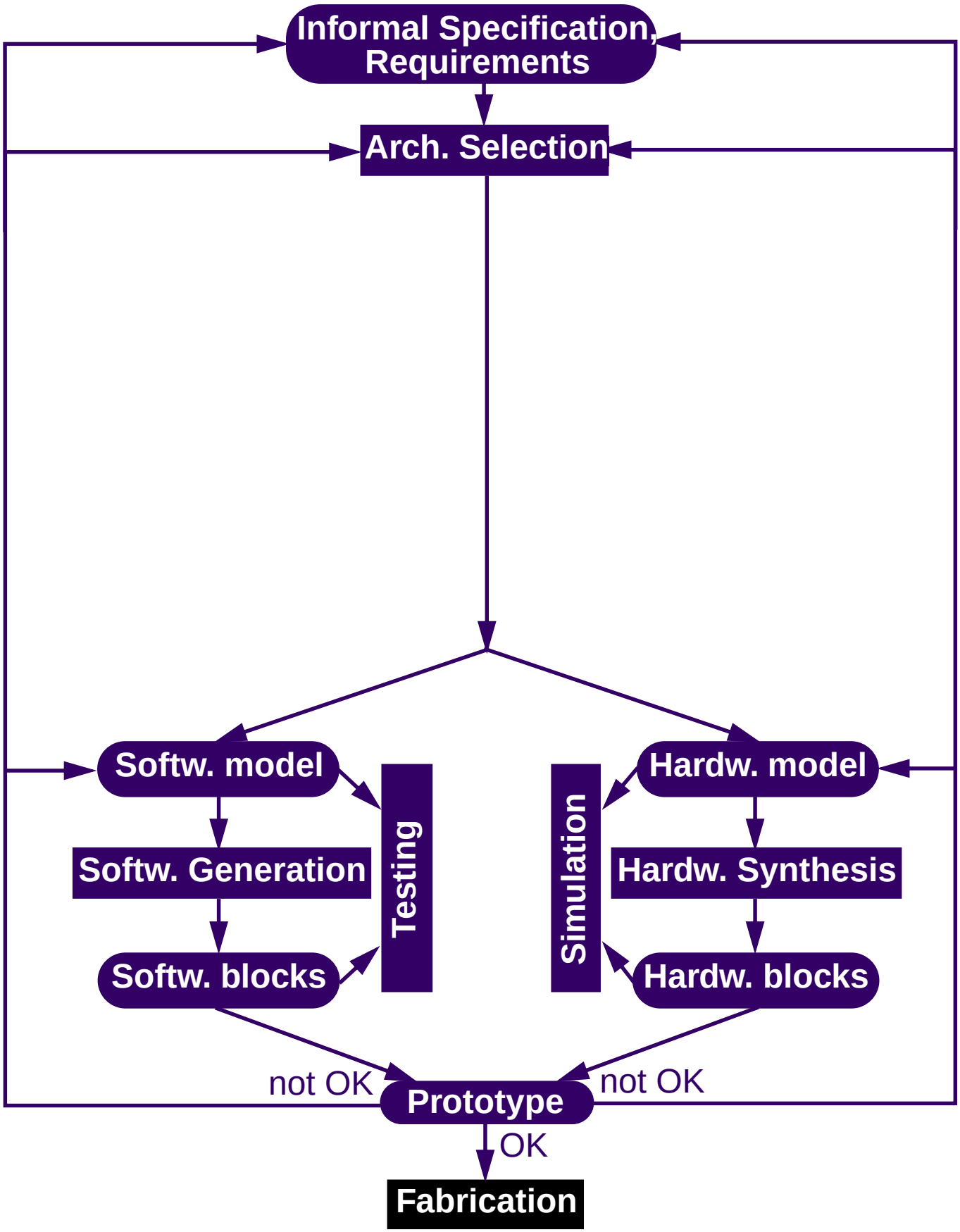
A Design Flow



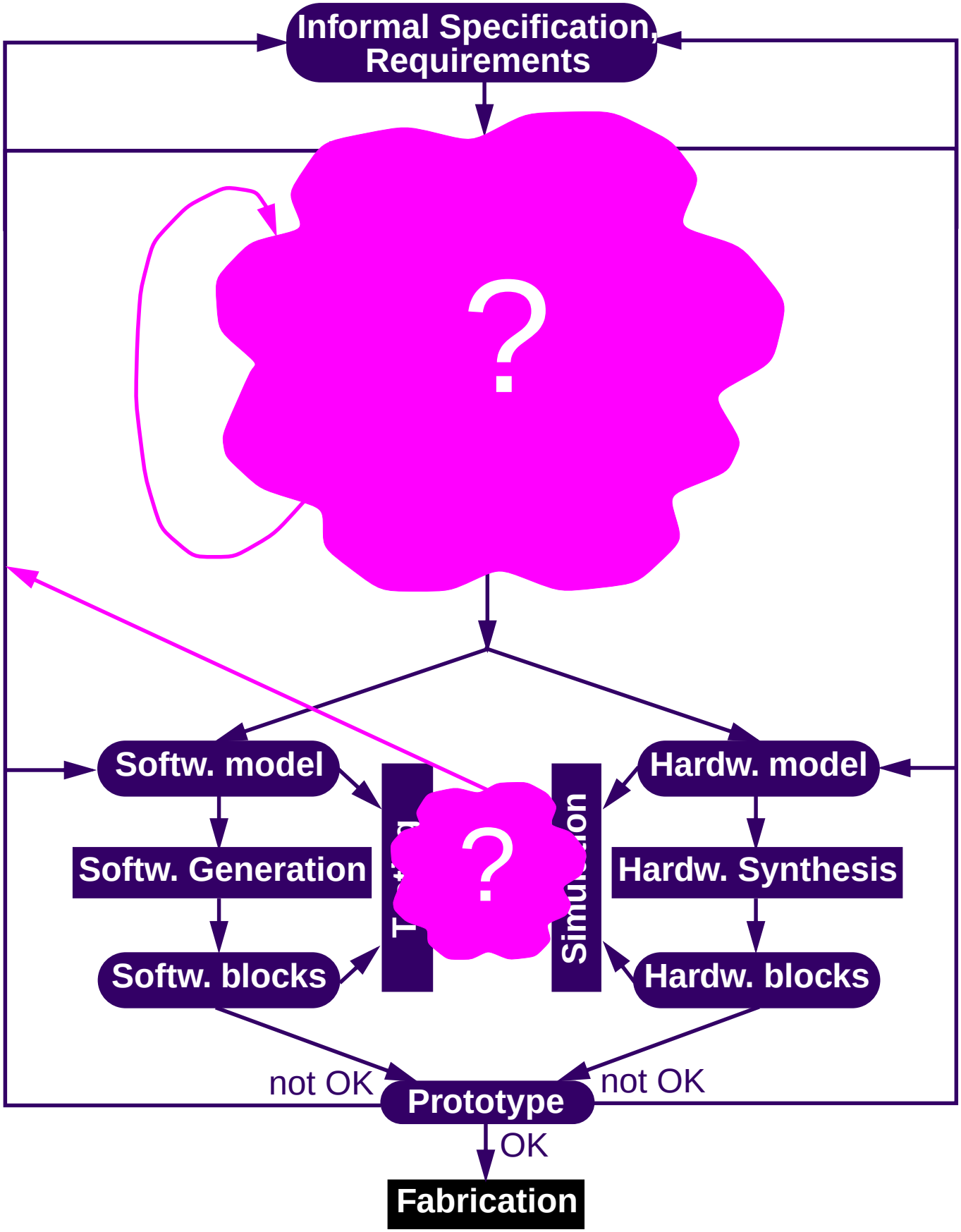
A Design Flow



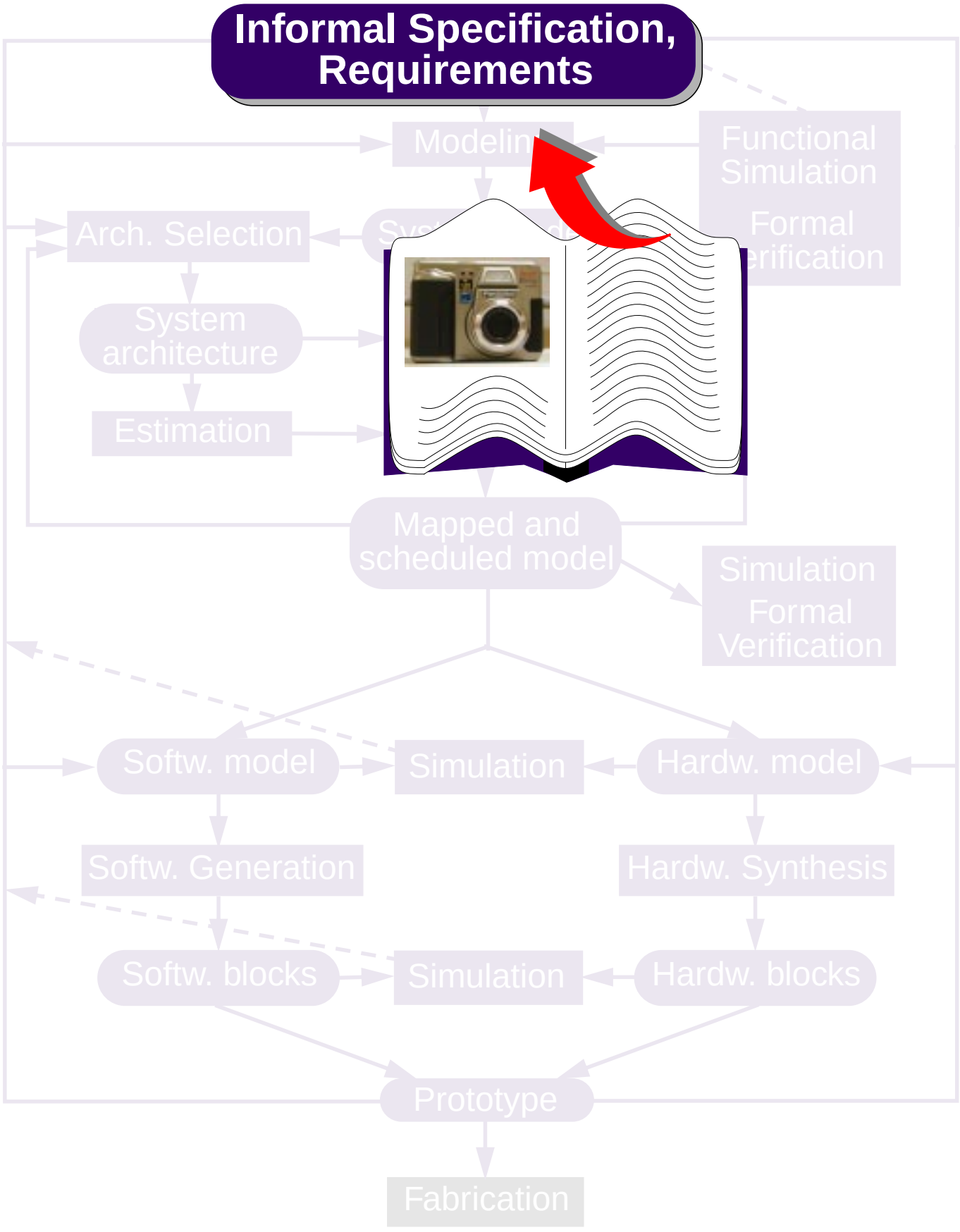
What Is Missing?



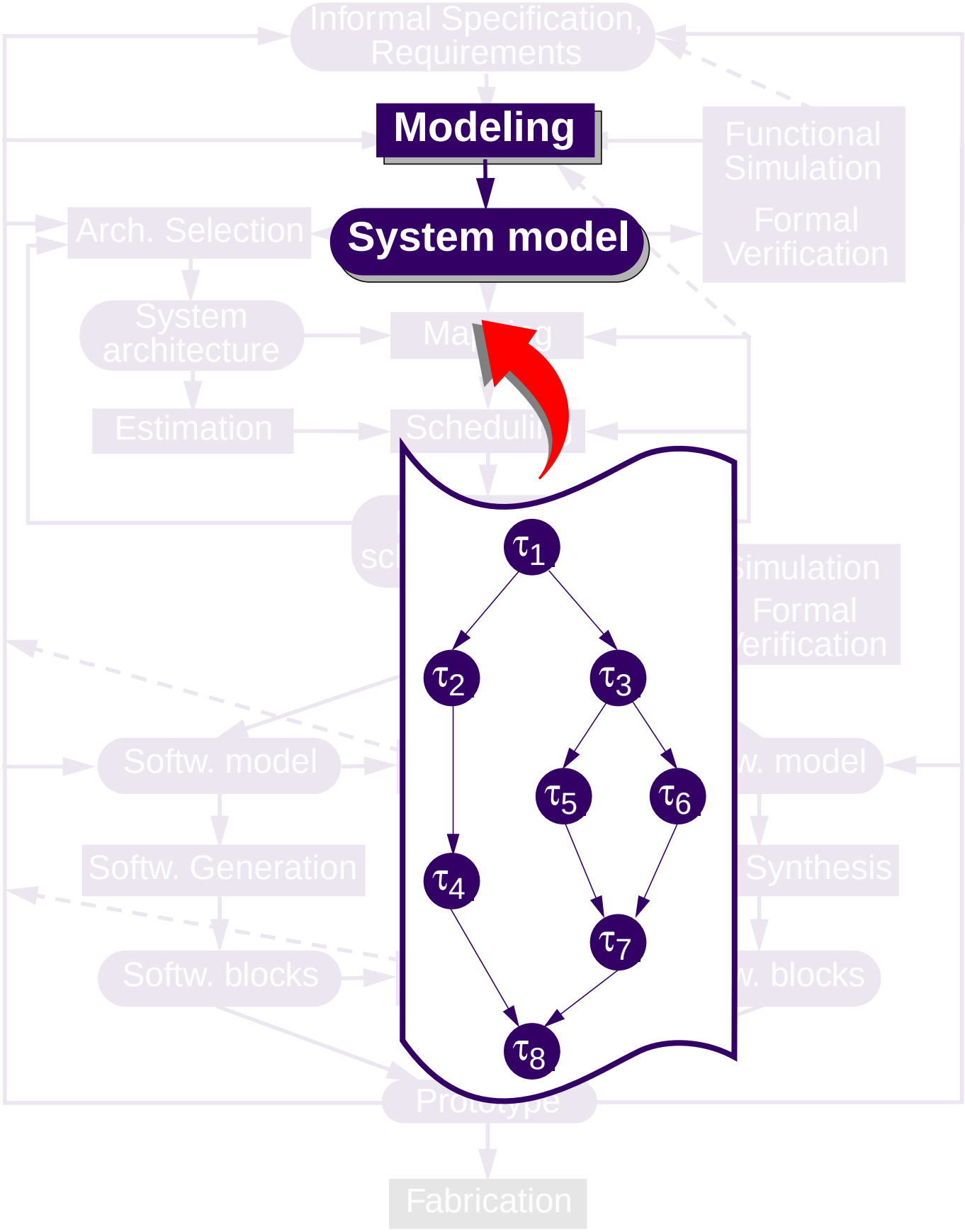
What Is Missing?



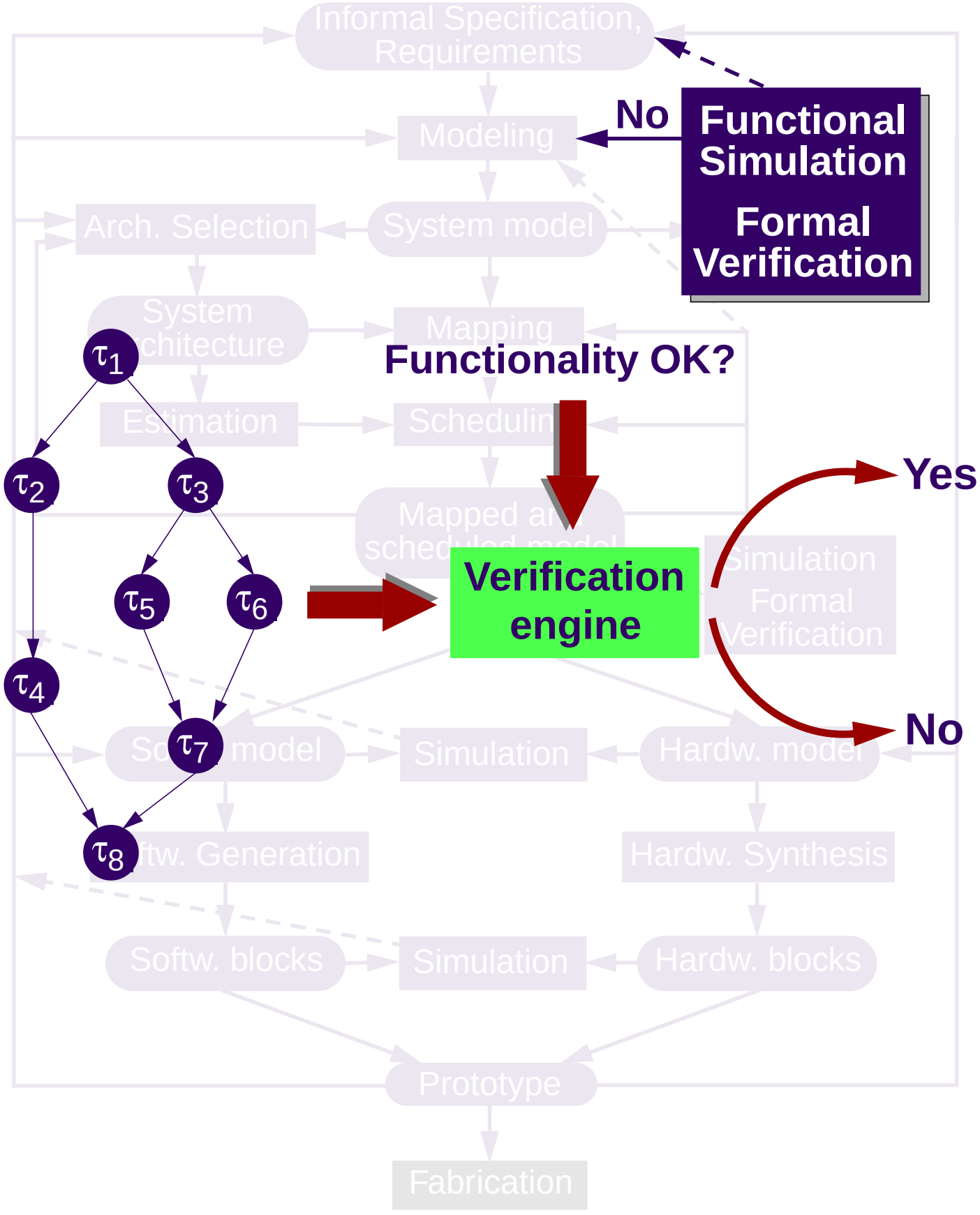
The System Level



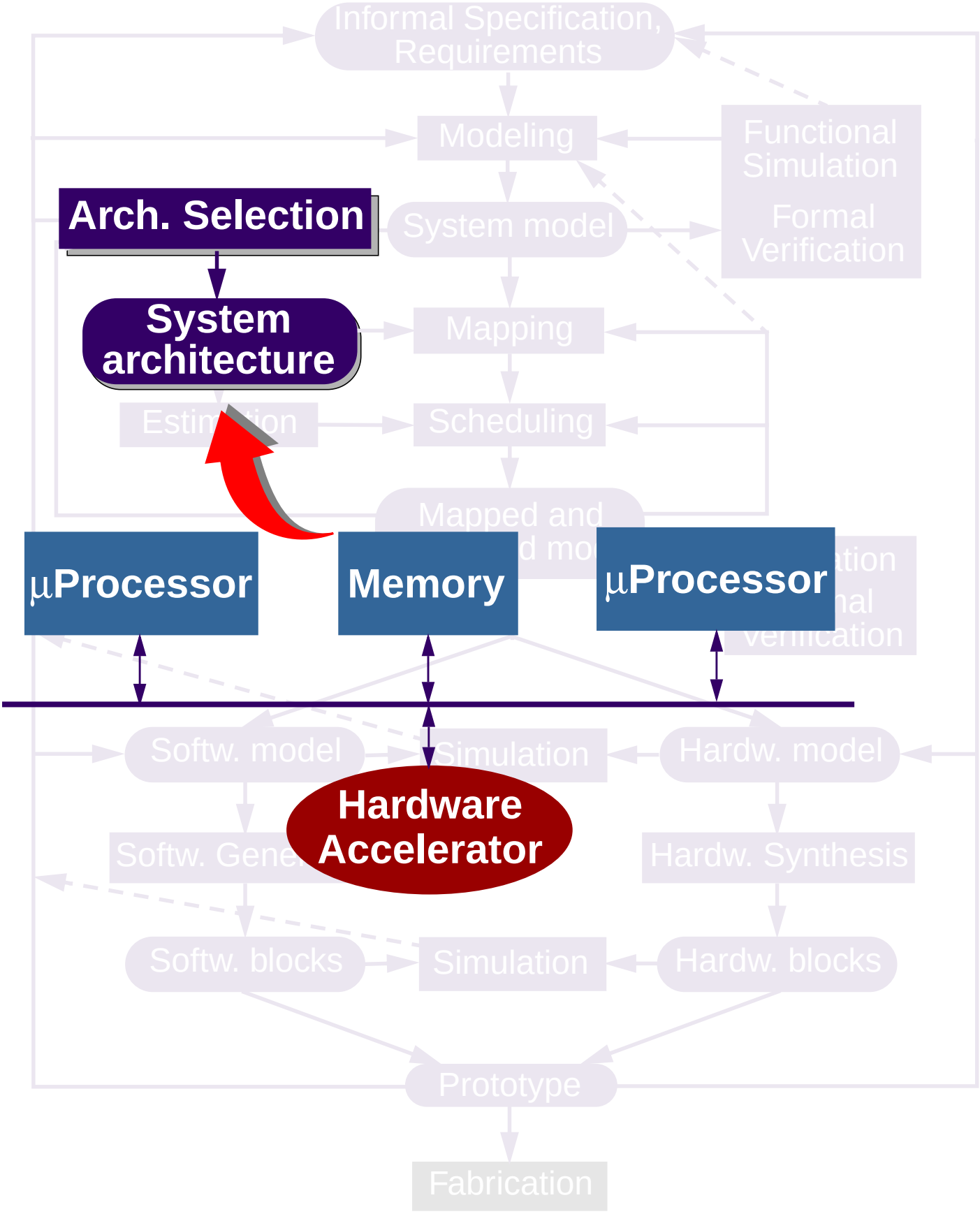
The System Level



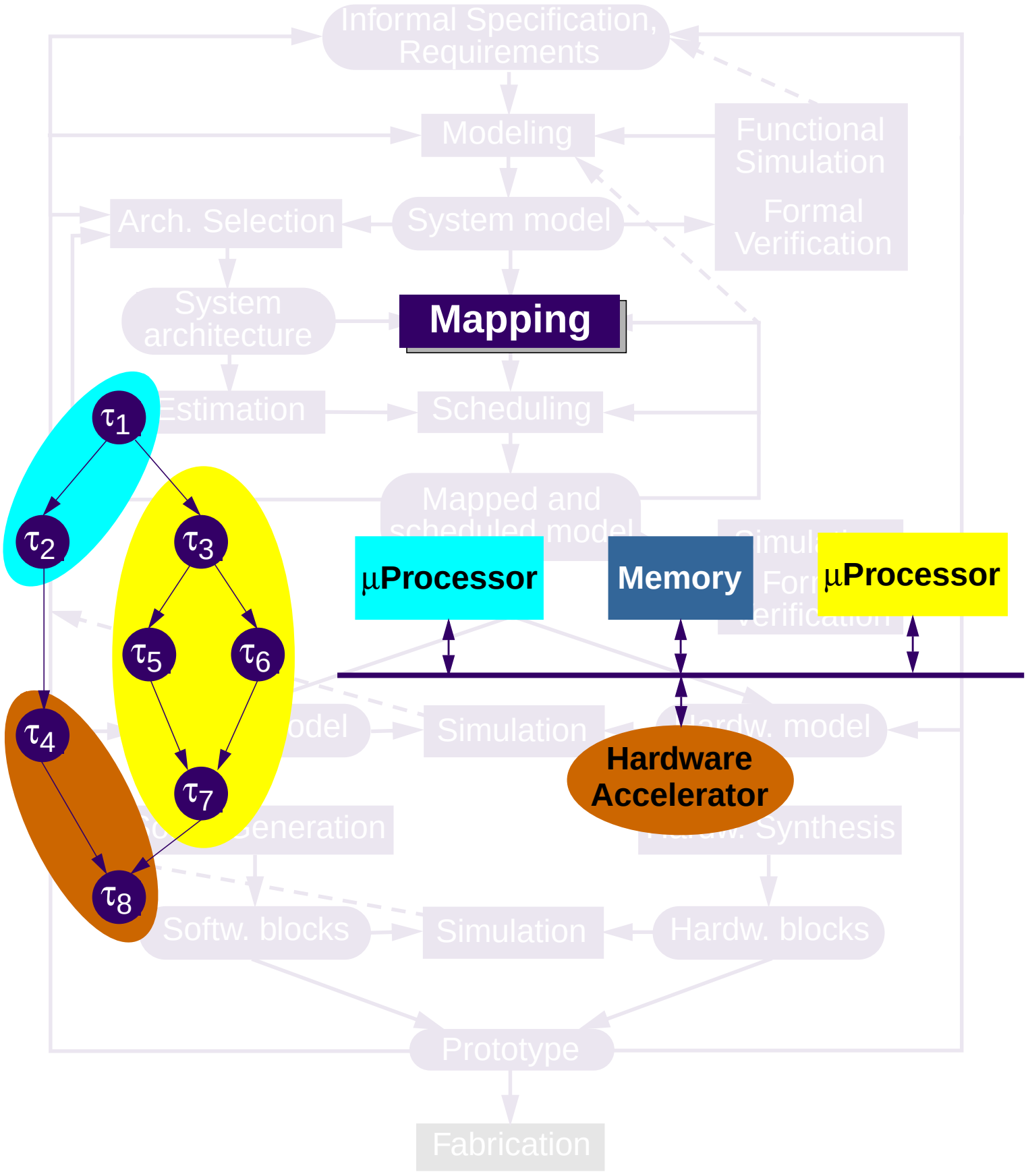
The System Level



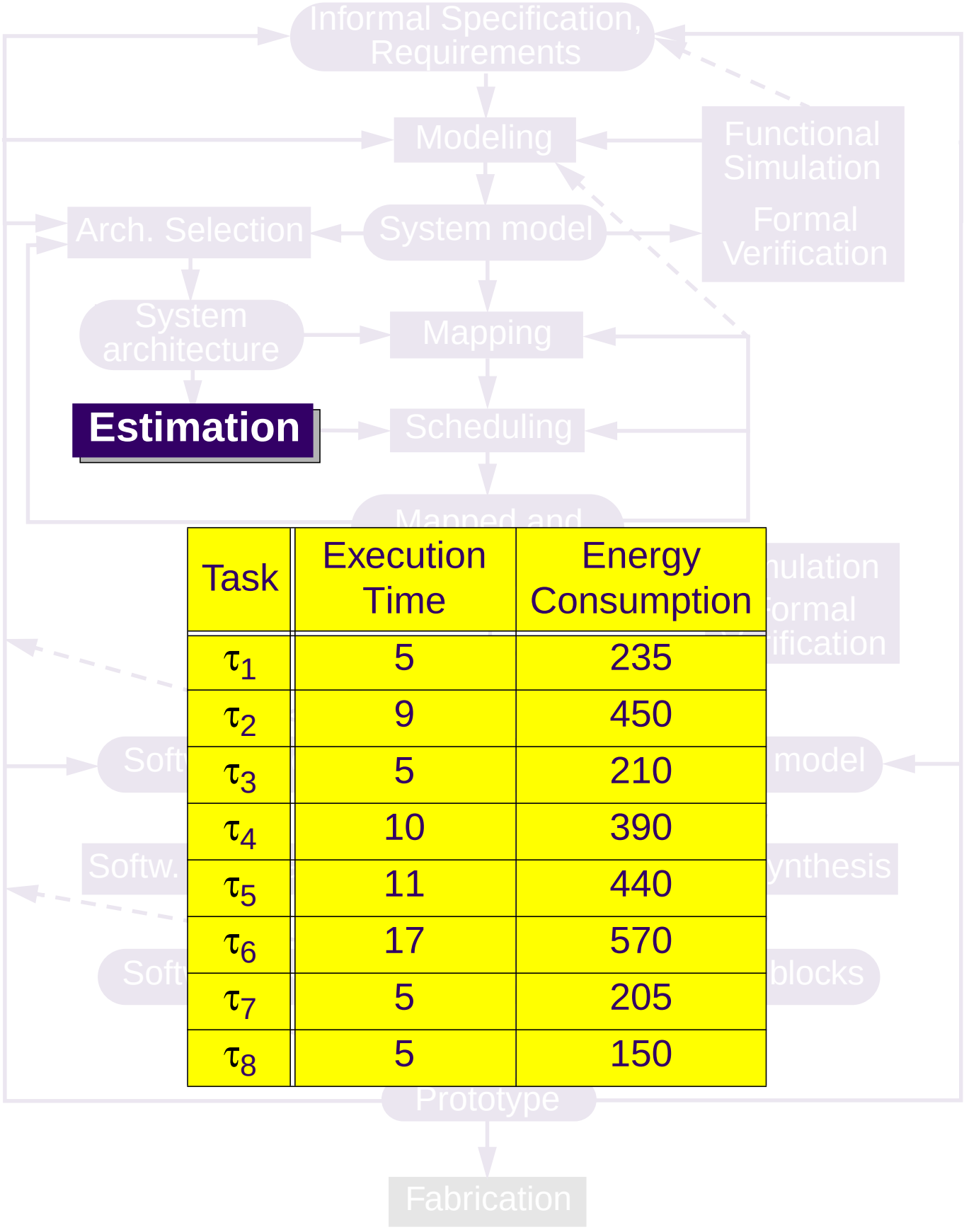
The System Level



The System Level

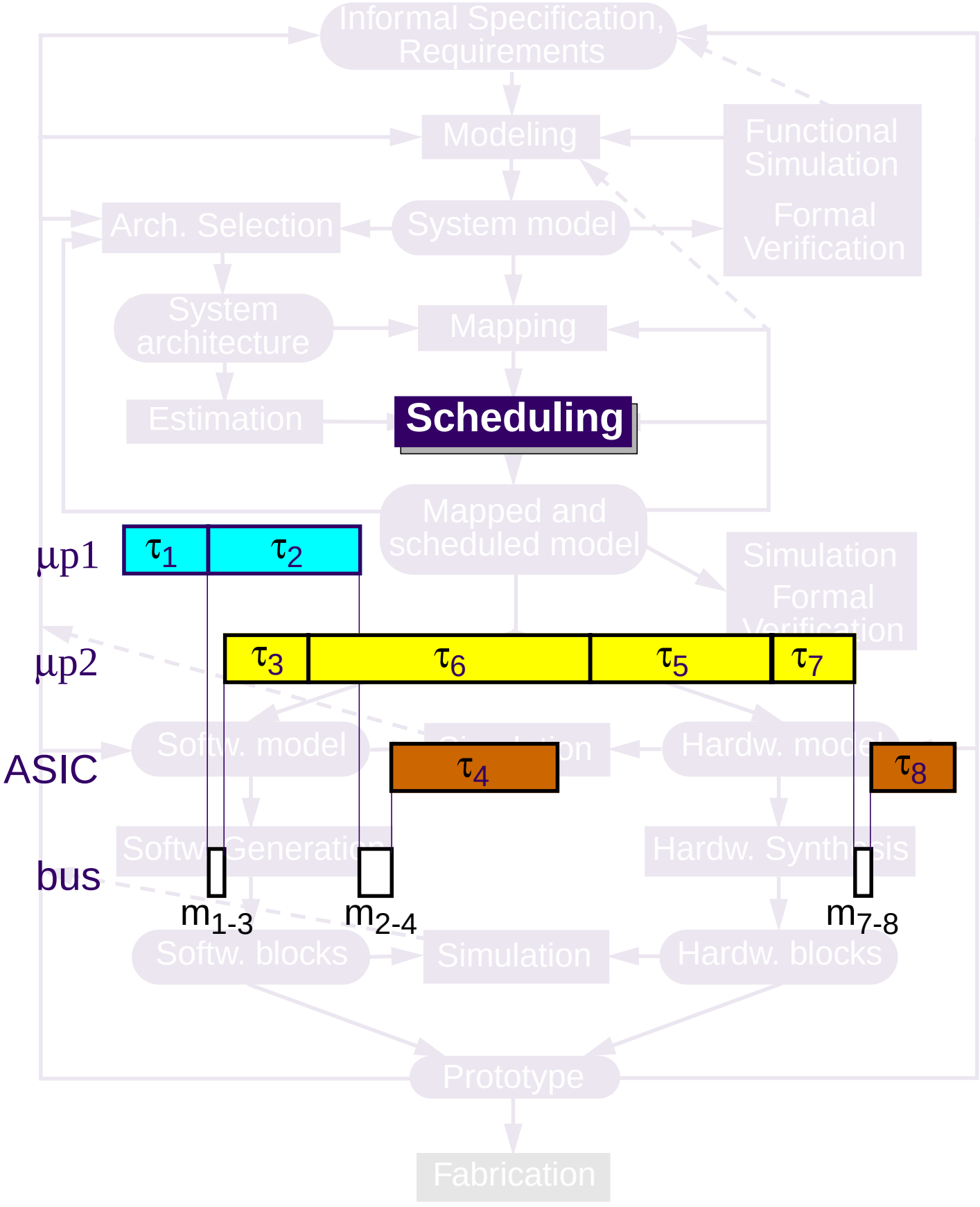


The System Level

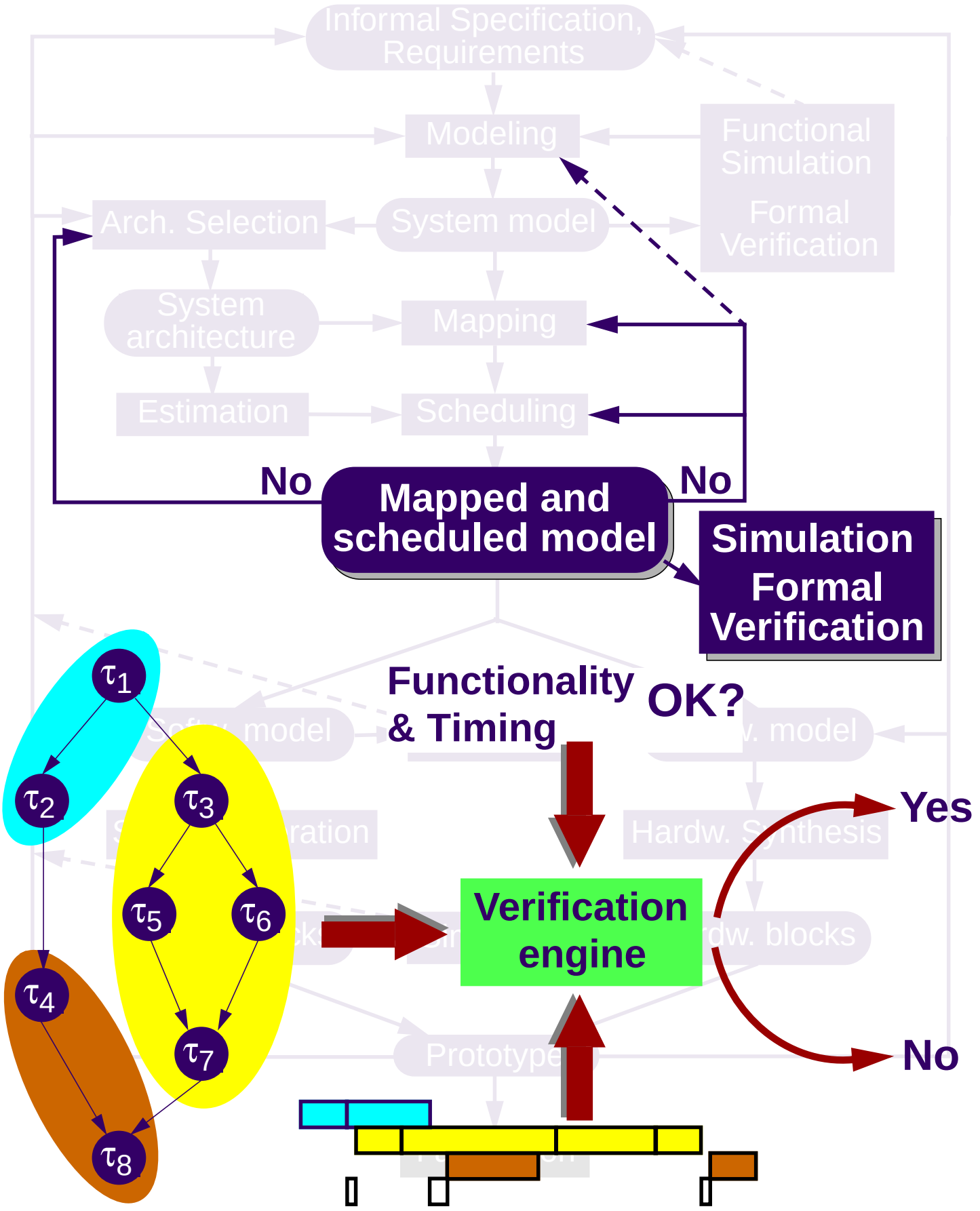


Task	Execution Time	Energy Consumption
τ_1	5	235
τ_2	9	450
τ_3	5	210
τ_4	10	390
τ_5	11	440
τ_6	17	570
τ_7	5	205
τ_8	5	150

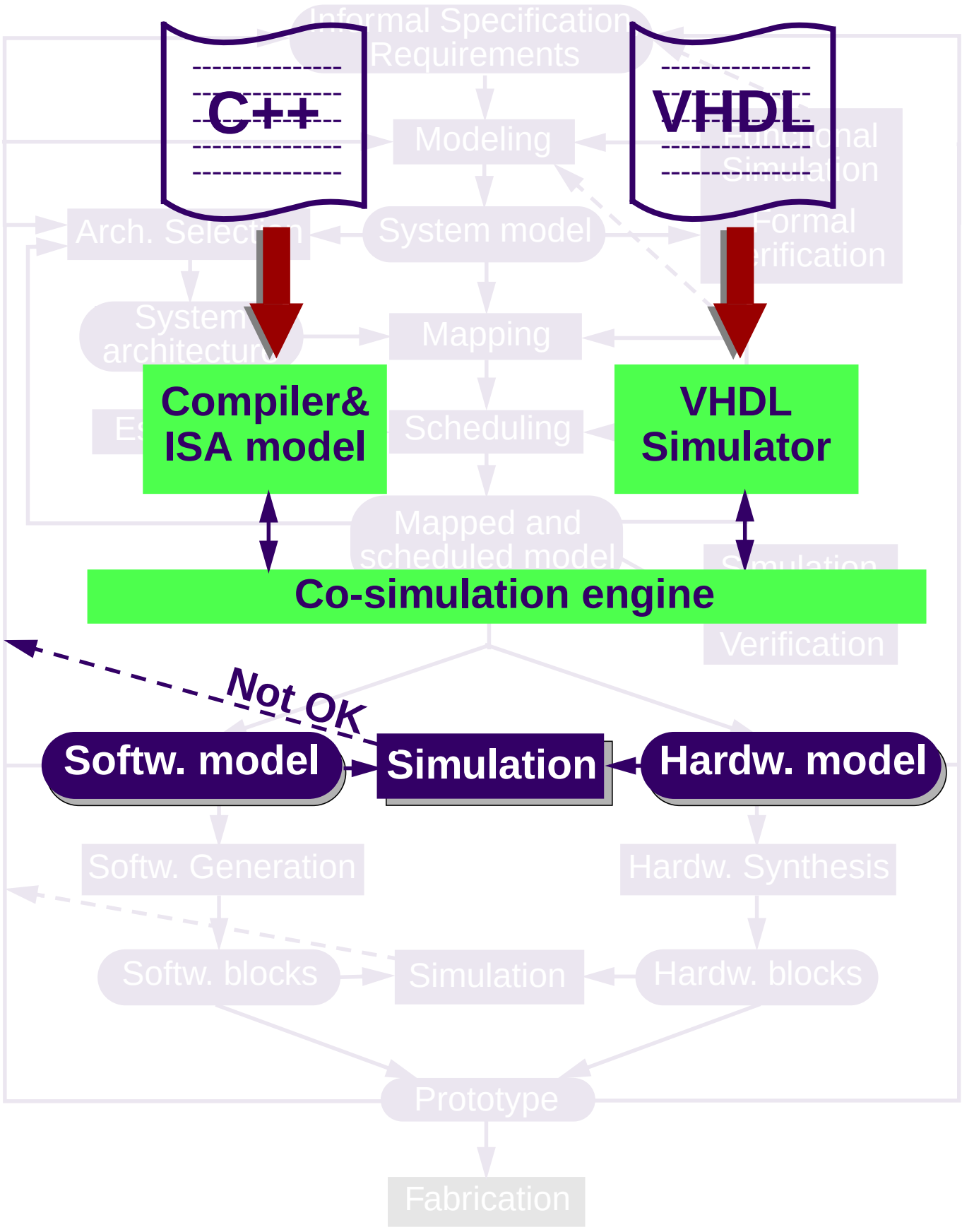
The System Level



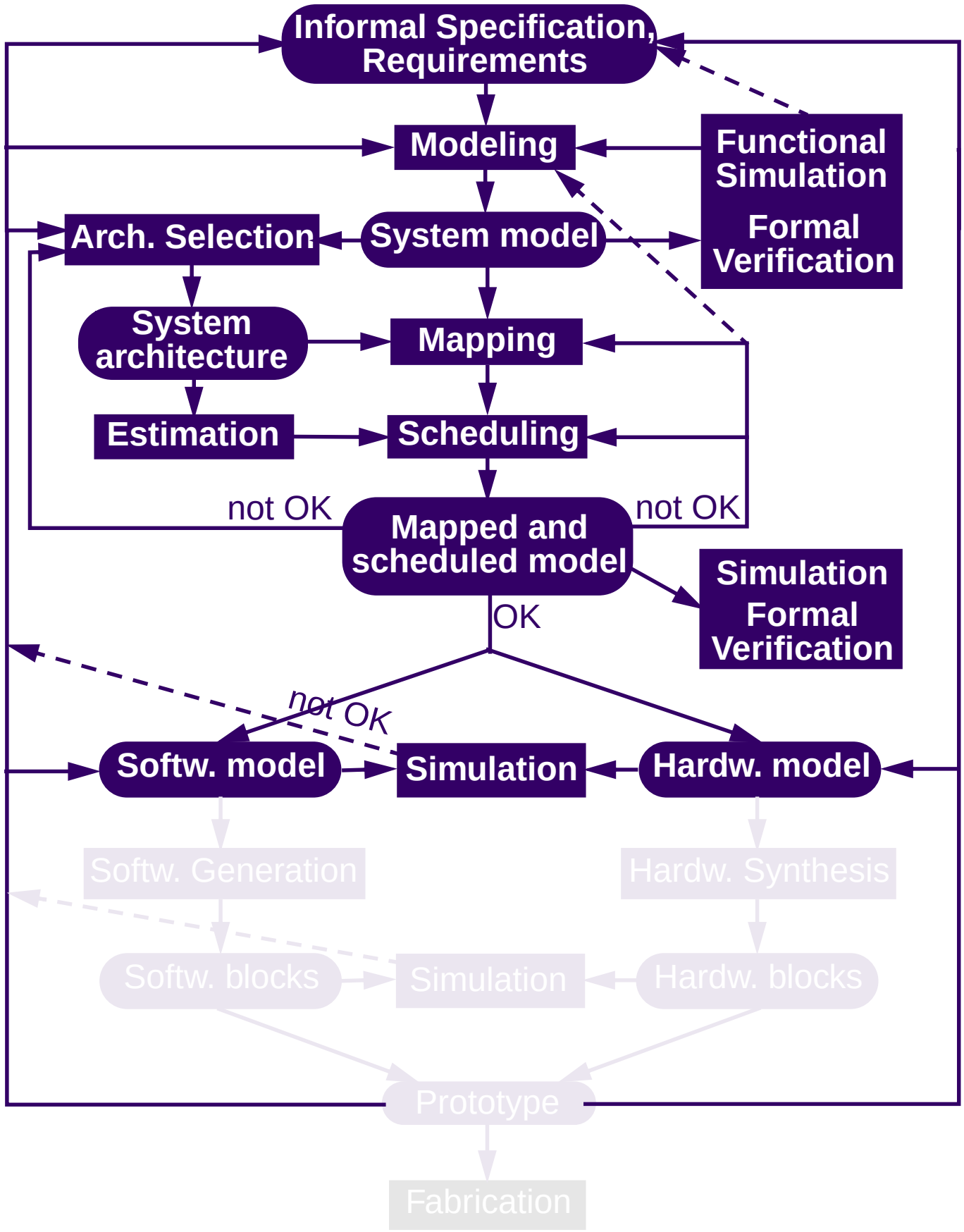
The System Level



The System Level



System Level Design Flow



System-level Design

☞ System-level design is performed before any effective hardware/software implementation has been generated.

- It is a design loop including the exploration of different
  - architectures (including communication infrastructure)
  - mappings
  - schedules
- It is supported by
  - system level models
  - estimation
  - analysis & formal verification
  - simulation

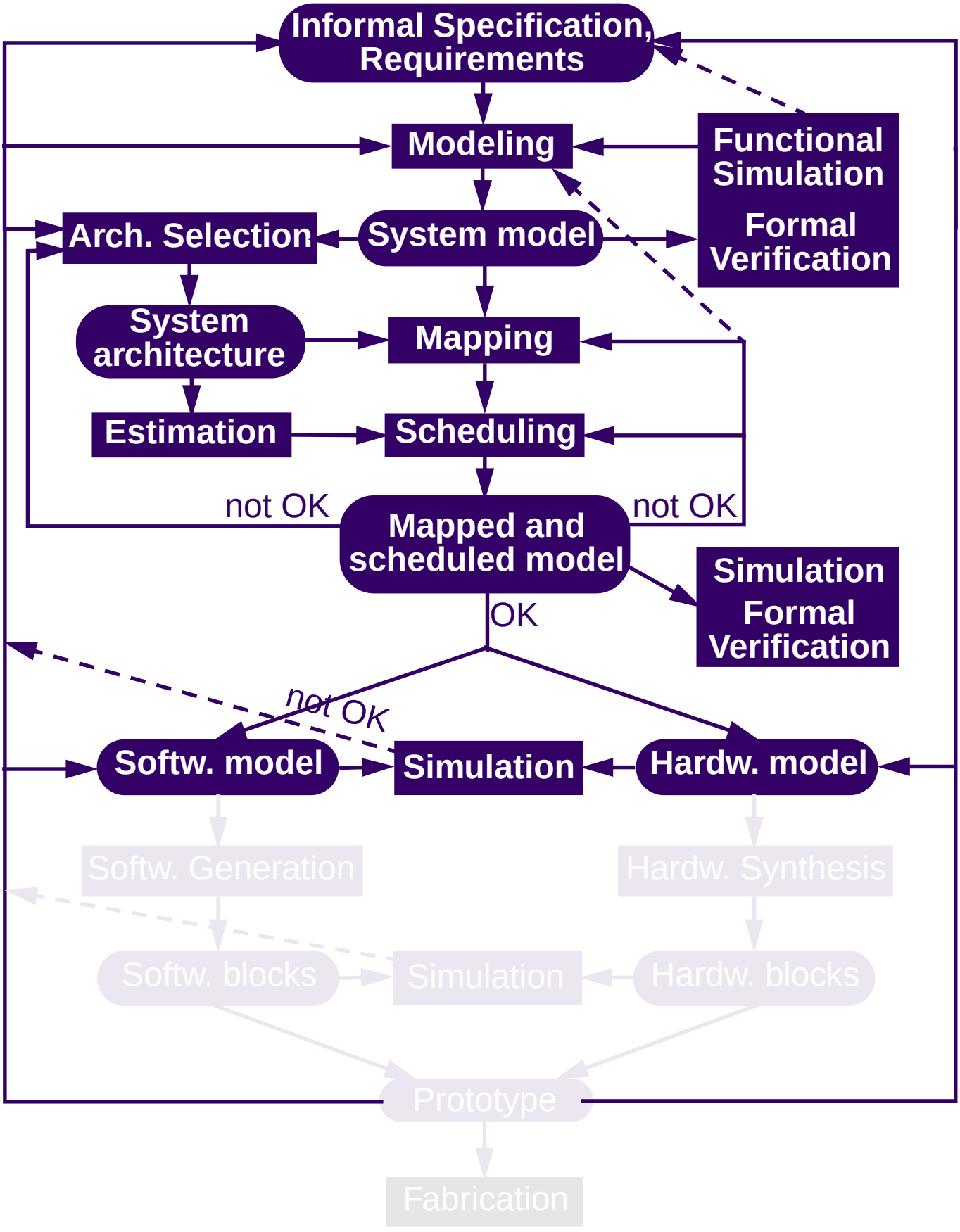
System-level Design

☞ System-level design is performed before any effective hardware/software implementation has been generated.

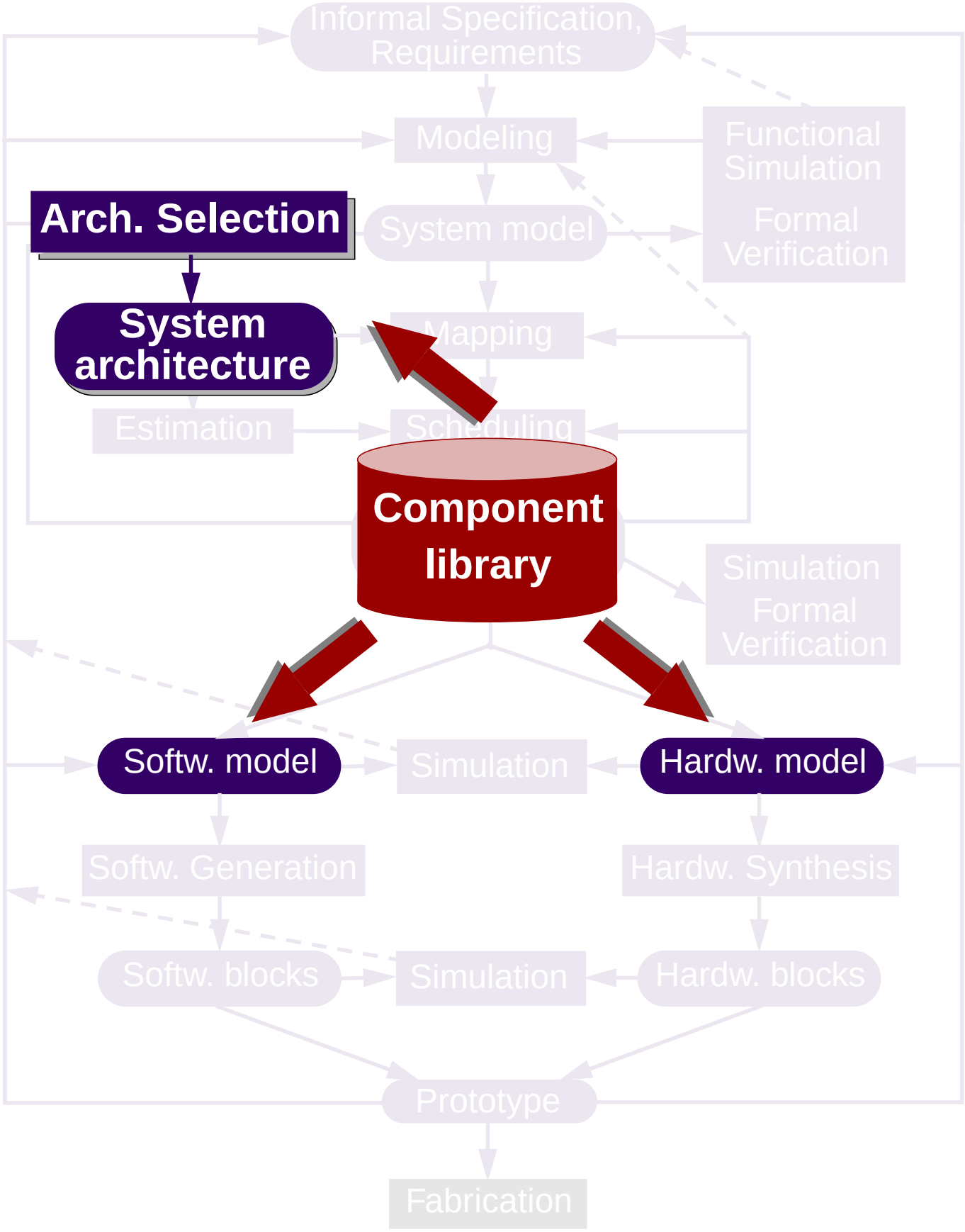
- It is a design loop including the exploration of different
  - architectures (including communication infrastructure)
  - mappings
  - schedules
- It is supported by
  - system level models
  - estimation/analysis
  - analysis & formal verification
  - simulation

Hardware Architecture and Software are jointly developed!

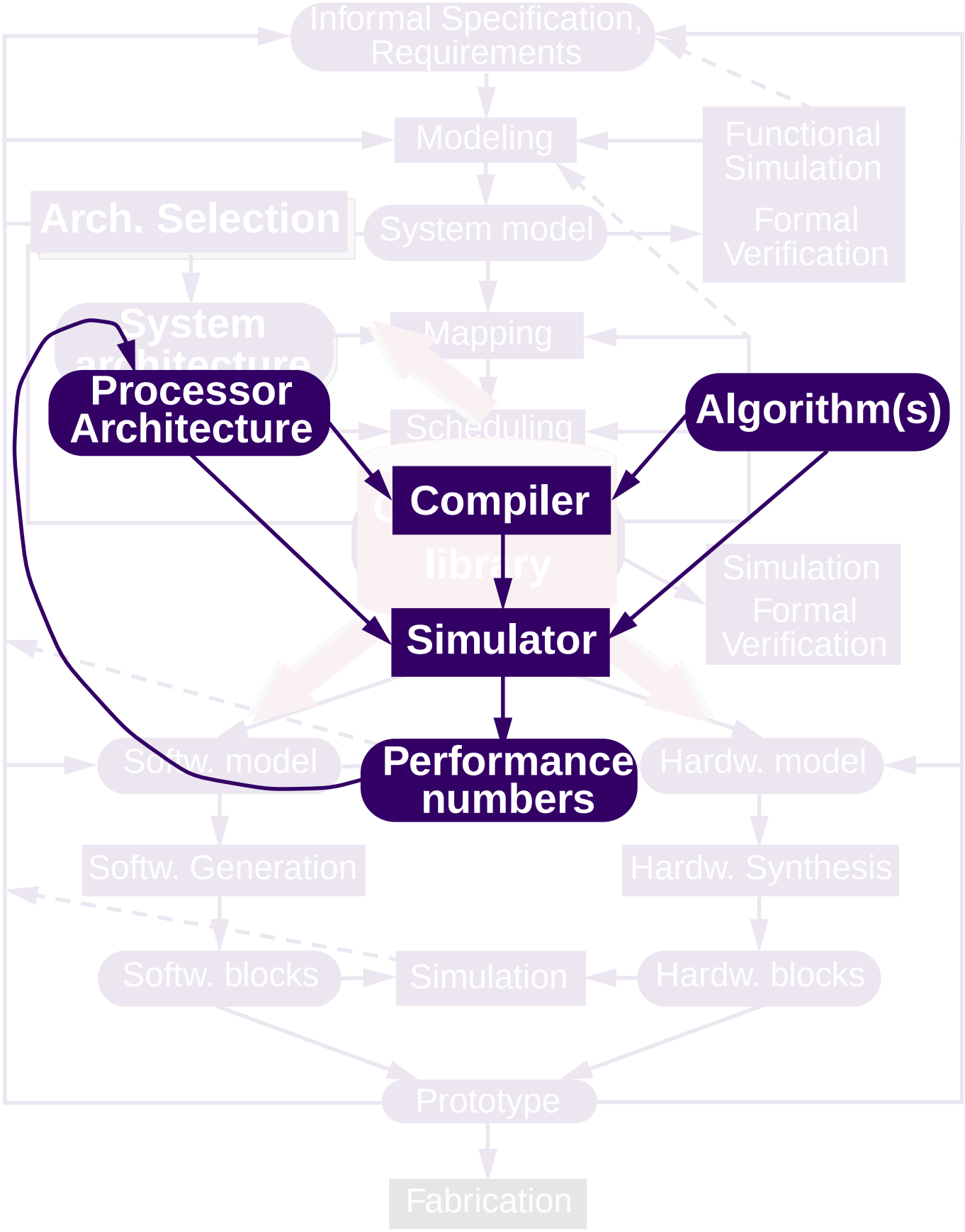
Platforms and IP-blocks



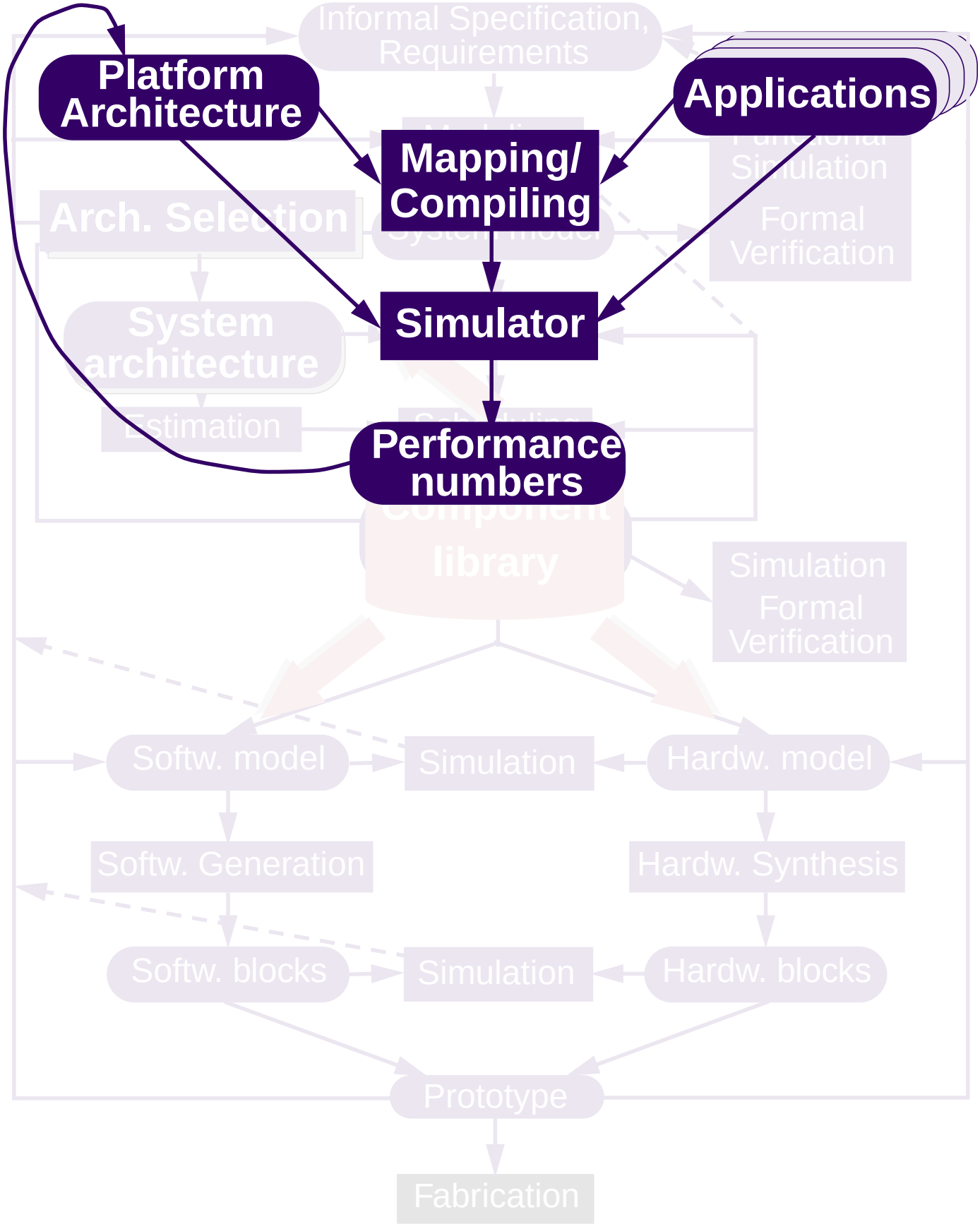
Platforms and IP-blocks



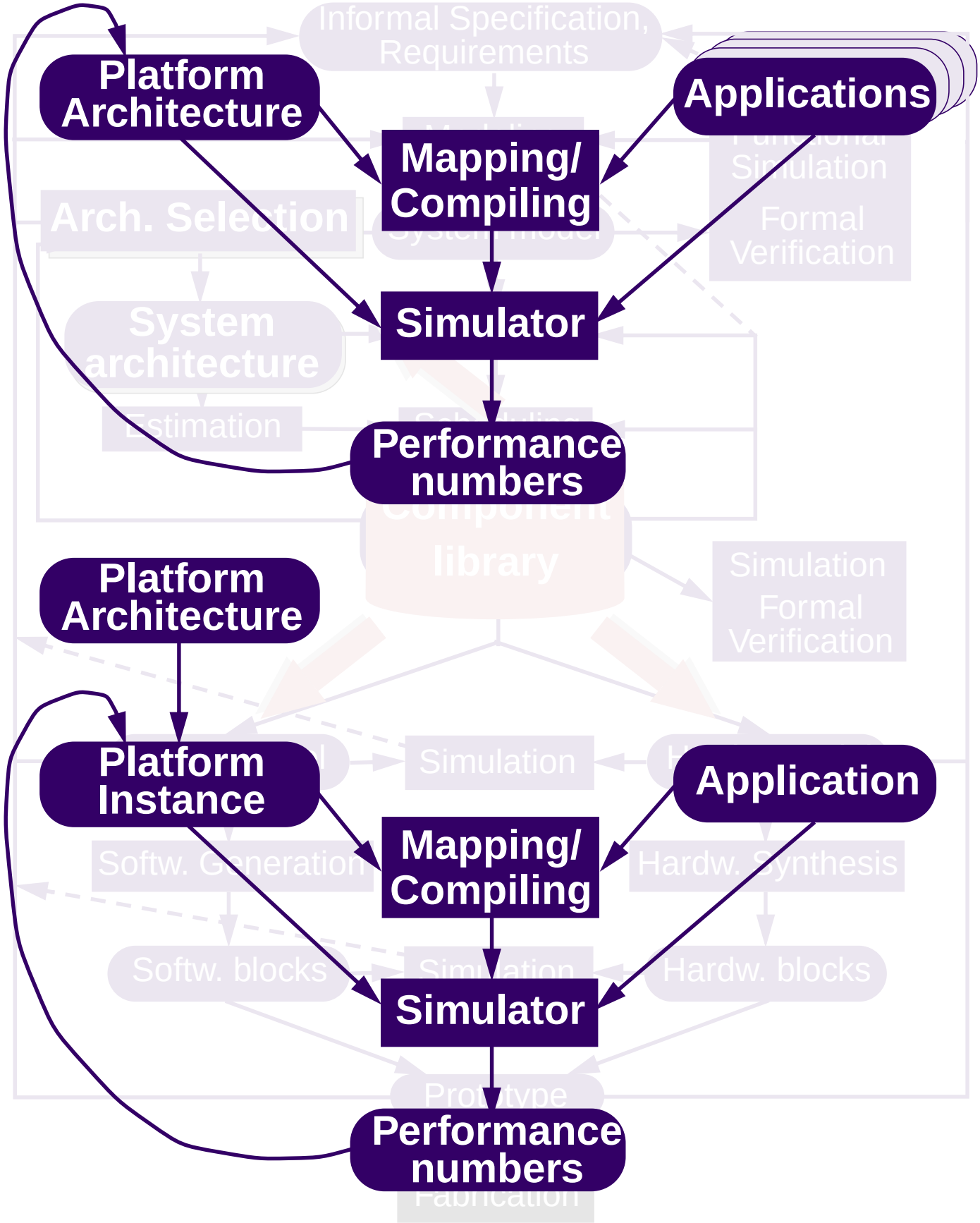
Platforms and IP-blocks



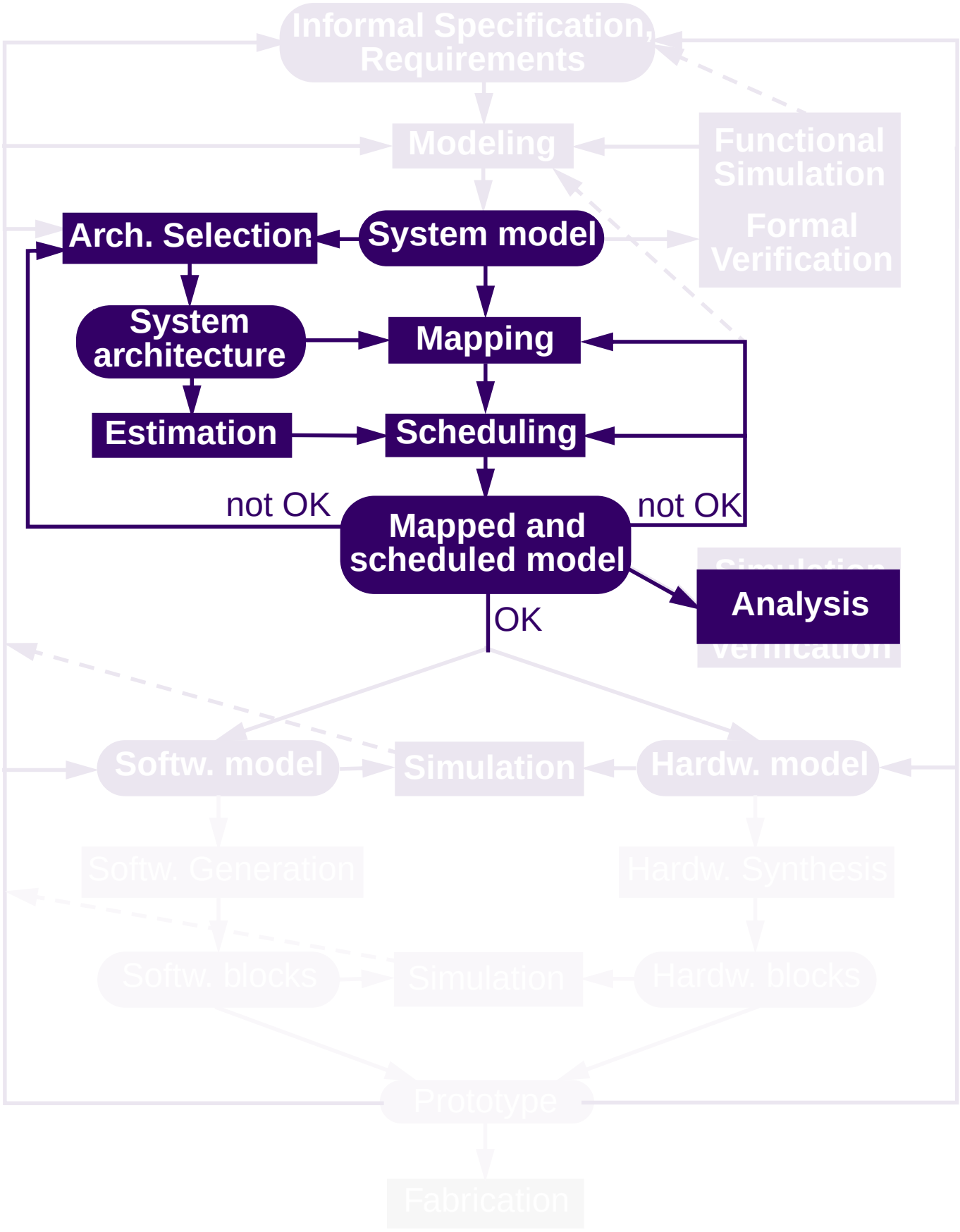
Platforms and IP-blocks



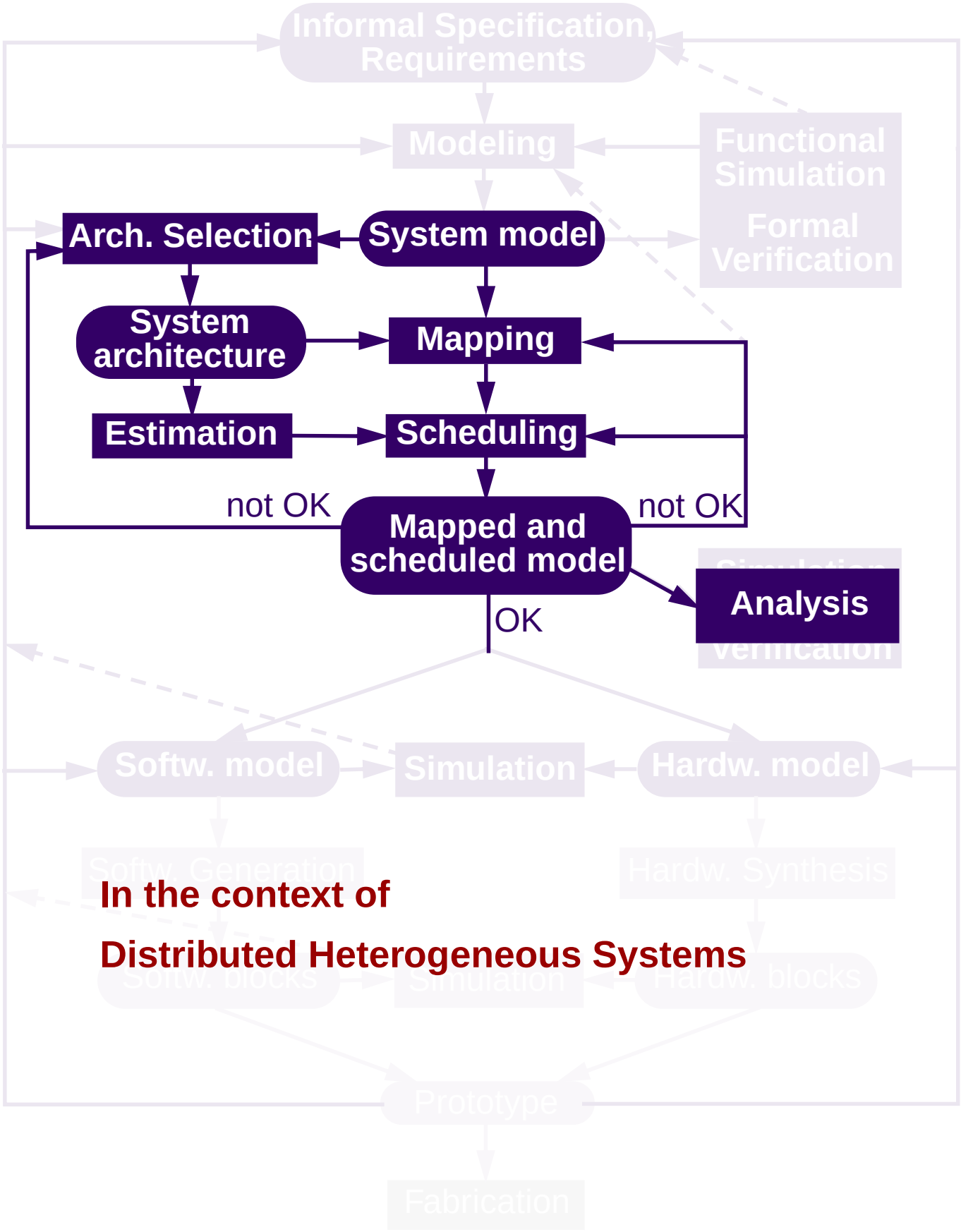
Platforms and IP-blocks



That's what's we are looking at



That's what's we are looking at



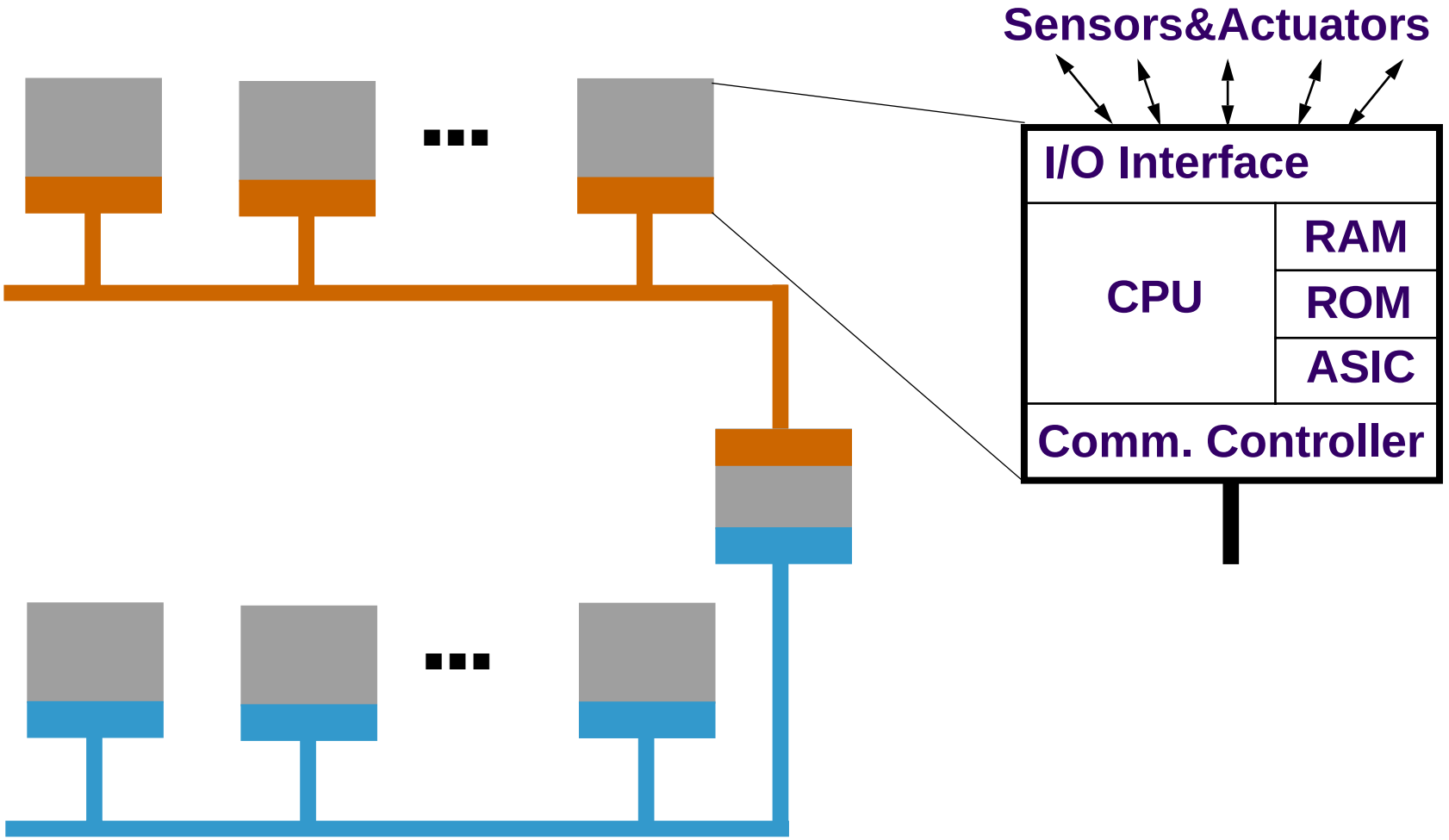
In the context of

Distributed Heterogeneous Systems

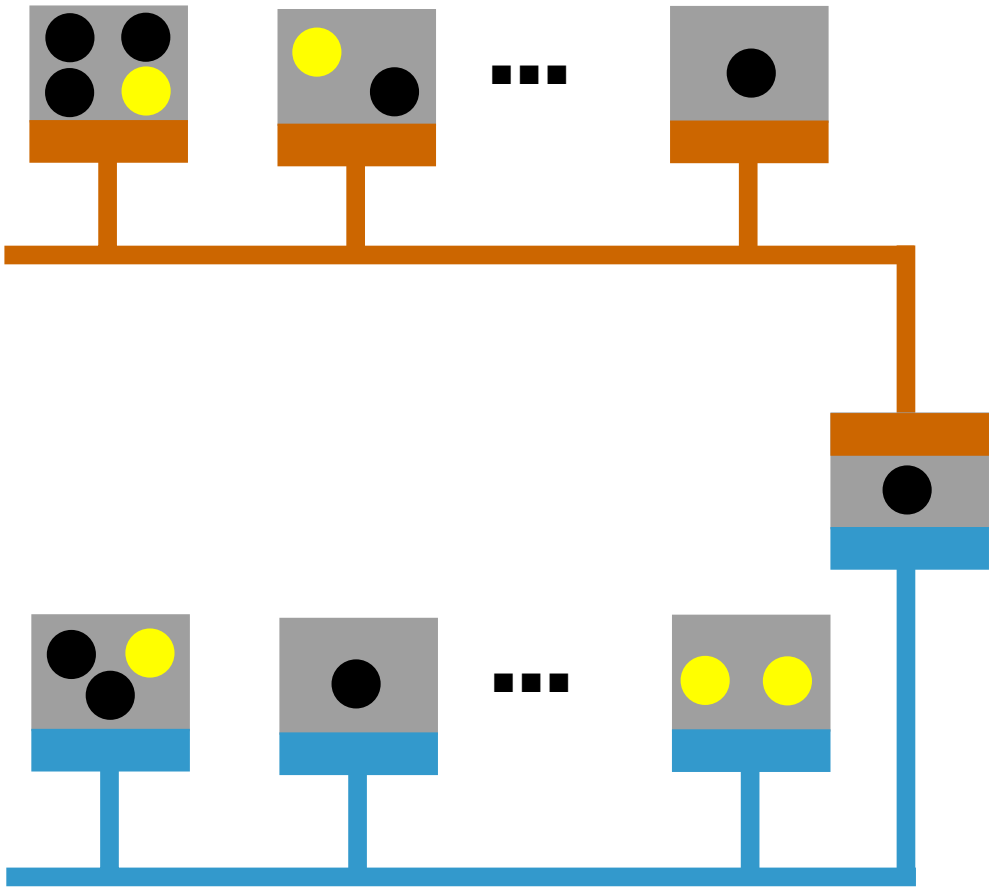
Outline

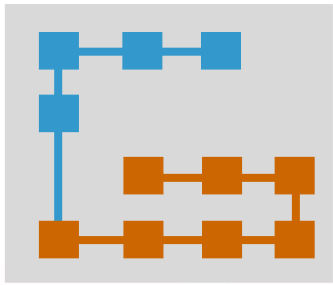
- Embedded Real-Time System
- System-level Design Flow
- Distributed Embedded Real-Time Systems
  - Application Model
  - Heterogeneous Systems
  - Time/Event Triggered Tasks
  - Static/Dynamic Communication
  - Analysis&Optimization
- Single/Multi-cluster Heterogeneous Distributed Architectures
  - Analysis&Optimization
- Incremental Design Process

Distributed Embedded Systems

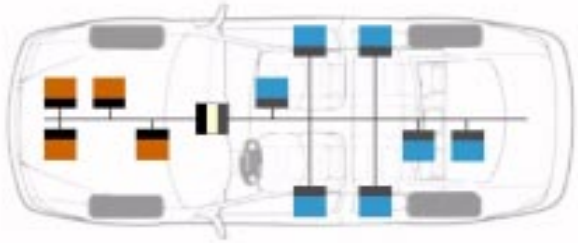


Distributed Embedded Systems

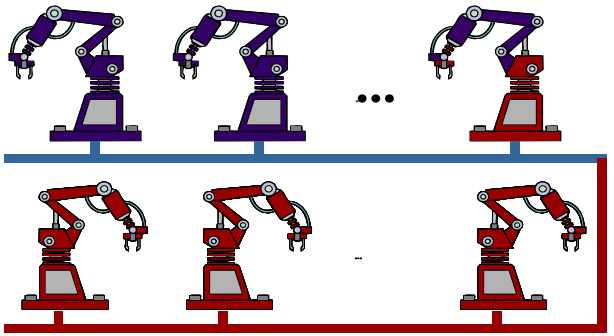




NoCs

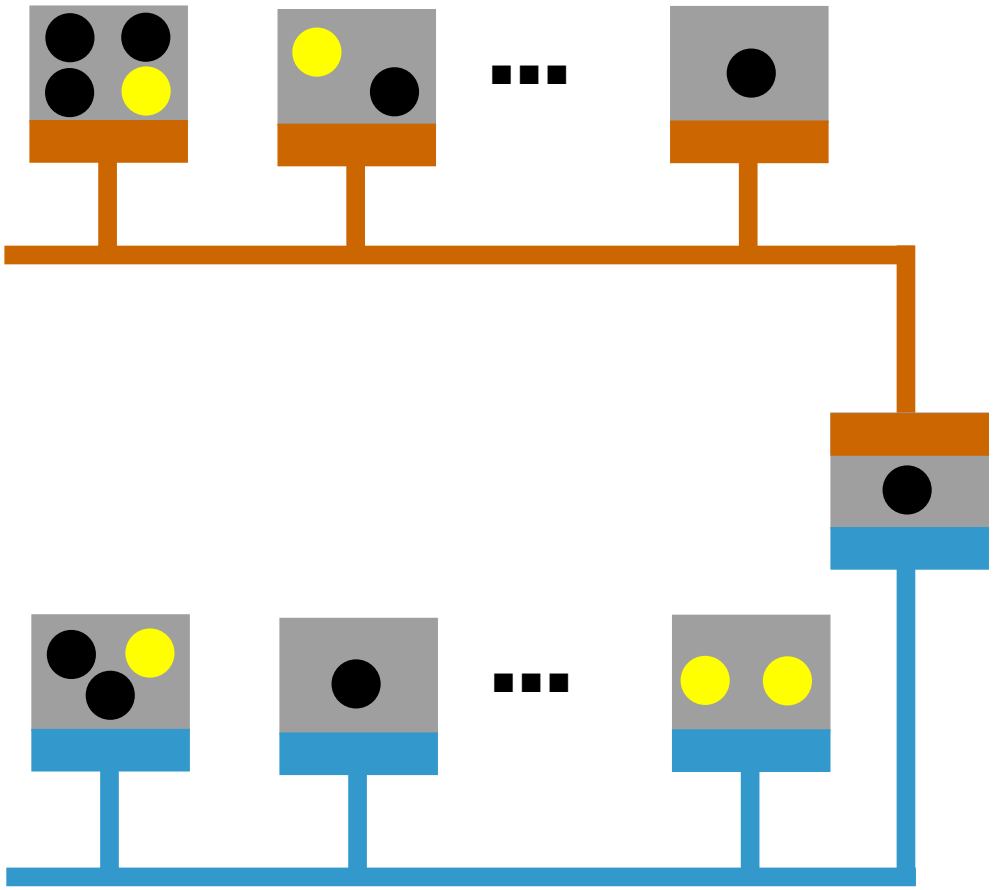


Automotive Electronics



Factory Systems

Distributed Embedded Systems



Why?

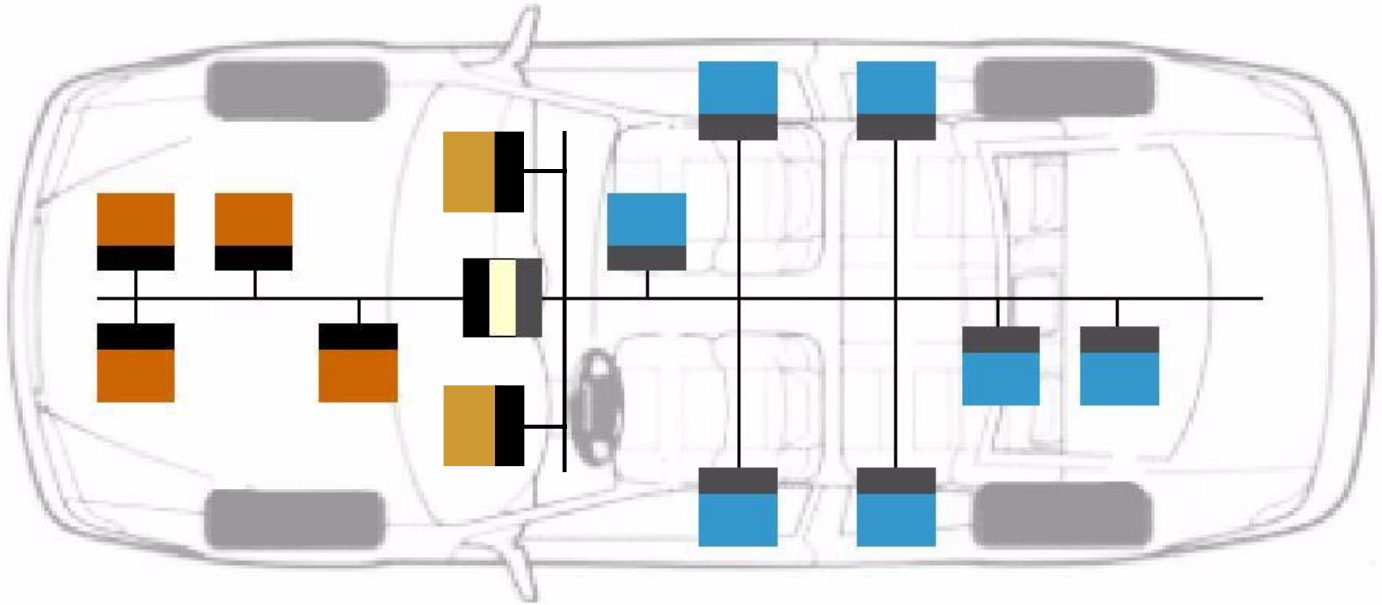
- **Physical constraints**
  - **Operation close to sensor;**
- **Modularity constraints**
- **Safety Constraints**
- **Performance**

Distributed Embedded Systems

- Dimensions of Heterogeneity
  - Architectural Components
  - Hardware/Software
  - Software Implementation (language, OS)
  - Data/Control Dominated
  - Continuous/Discrete
  - Network Protocol
  - RT Design Approach/Scheduling Policy
  - - - - - - - - - - - - - - -

Distributed Embedded Systems

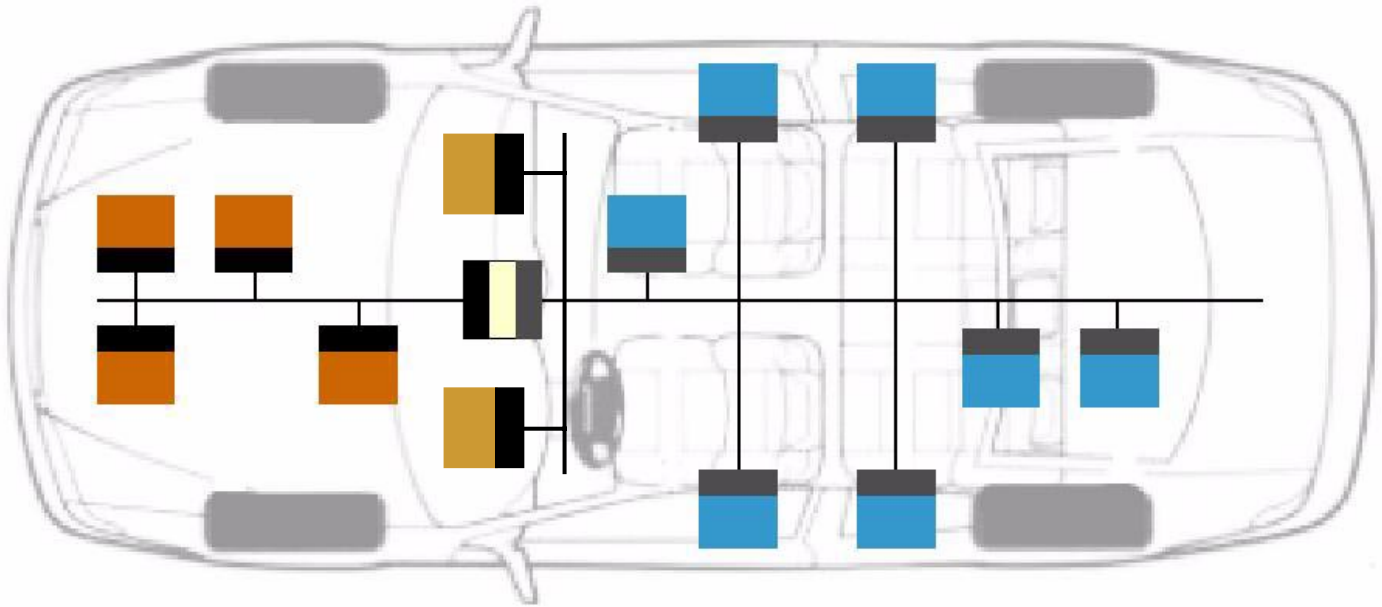
- Heterogeneous Nature of Implemented Functions



Distributed Embedded Systems

Engine Control

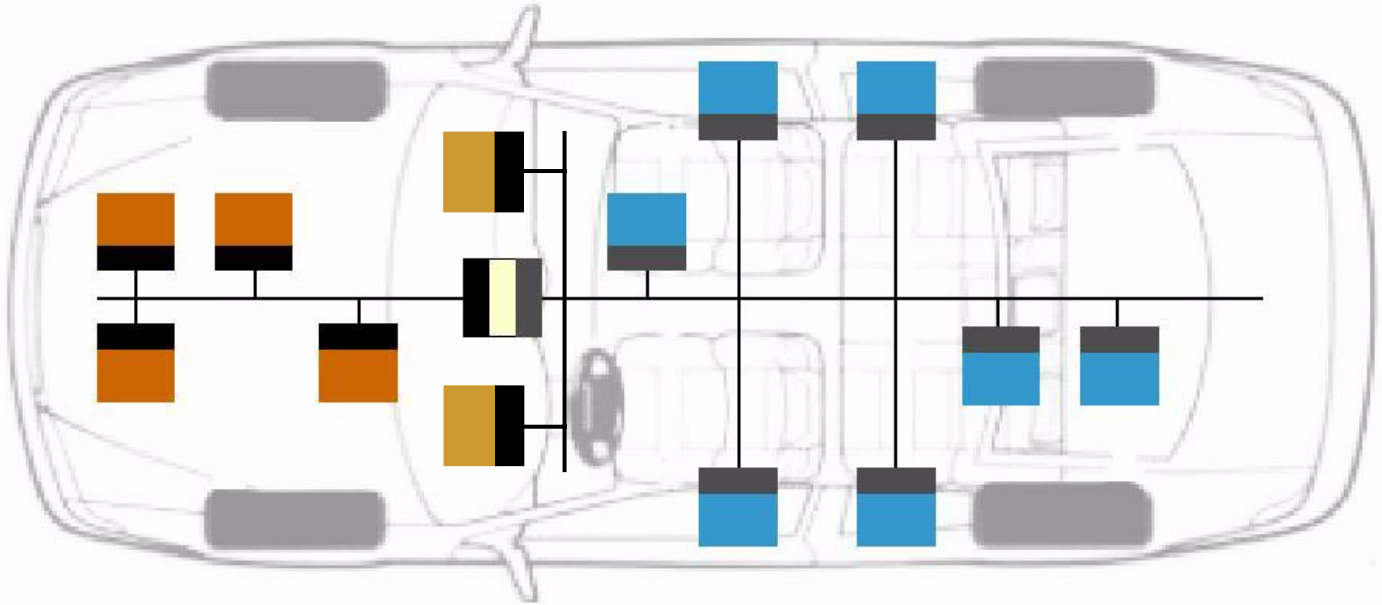
- hard real-time



Distributed Embedded Systems

Engine Control
- hard real-time

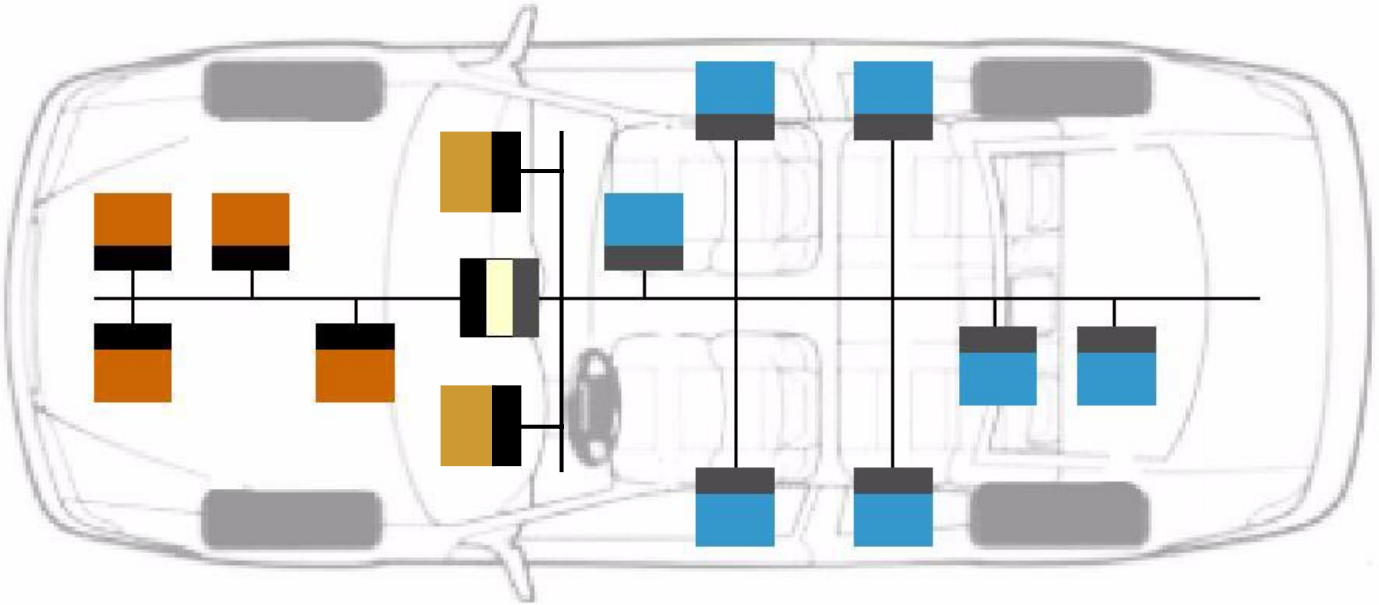
Power Train (break-by-wire, ABS)
- hard real-time
- highly safety-critical



Distributed Embedded Systems

Engine Control
- hard real-time

Power Train (break-by-wire, ABS)
- hard real-time
- highly safety-critical



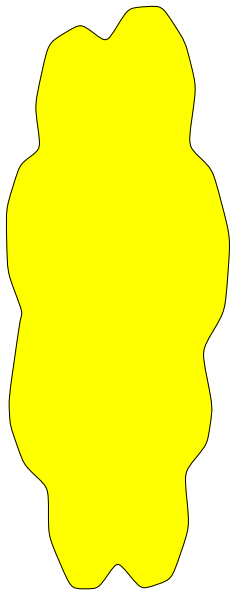
Air Conditioning
- soft real-time

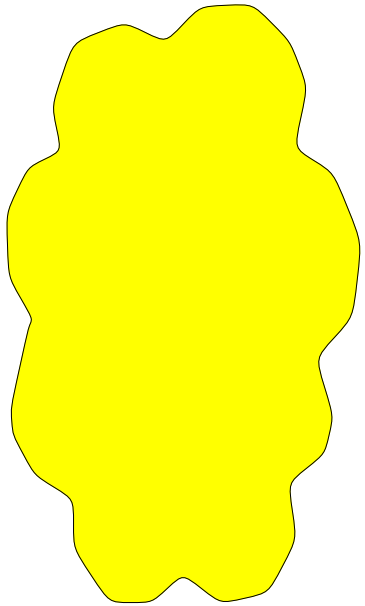
Distributed Embedded Systems

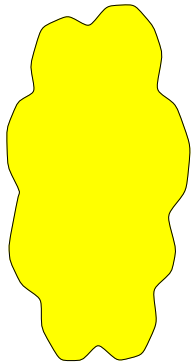
- Dimensions of Heterogeneity
  - Architectural Components
  - Hardware/Software
  - Software Implementation (language, OS)
  - Data/Control Dominated
  - Continuous/Discrete
  - Network Protocol
  - RT Design Approach/Scheduling Policy
  - - - - - - - - - - - - - - -

Application Model

- An application is modelled as a set of task graphs:







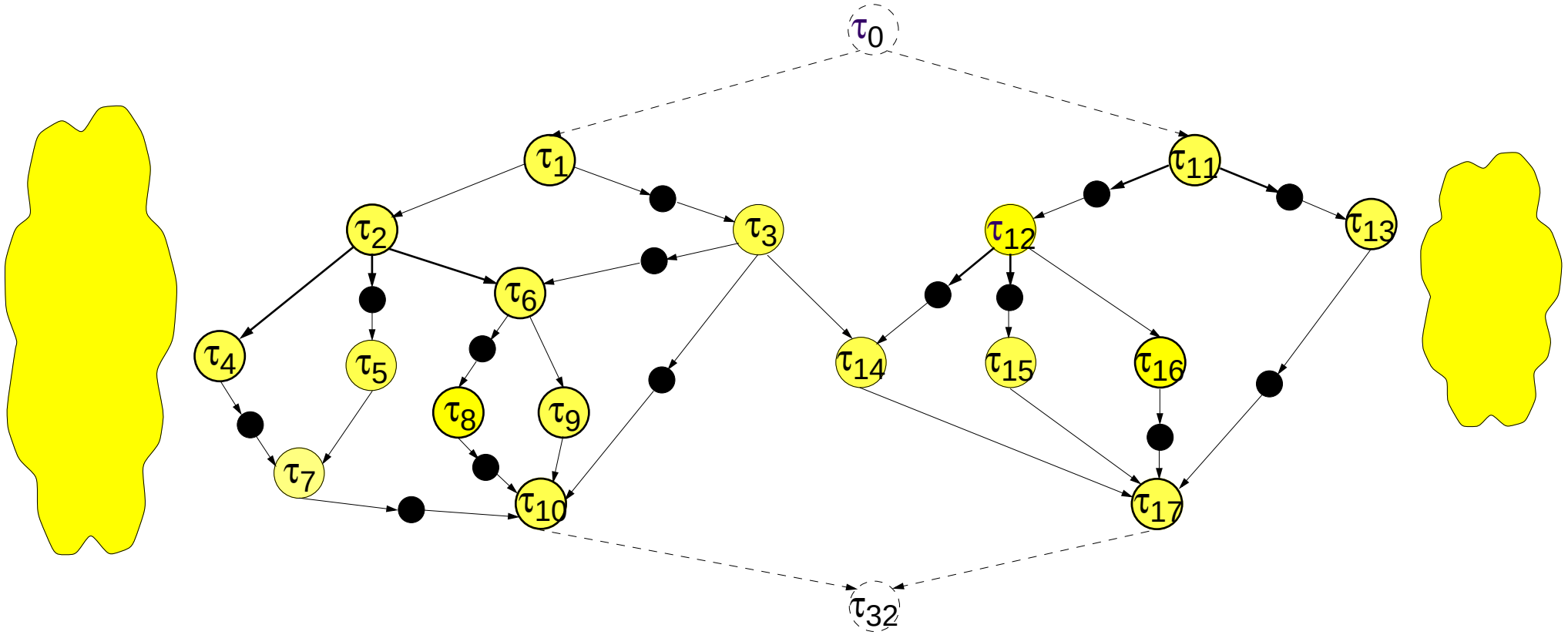
Γ_1
Period: $T_{\Gamma 1}$
Deadline: $D_{\Gamma 1}$

Γ_2
Period: $T_{\Gamma 2}$
Deadline: $D_{\Gamma 2}$

Γ_3
Period: $T_{\Gamma 3}$
Deadline: $D_{\Gamma 3}$

Application Model

- An application is modelled as a set of task graphs:



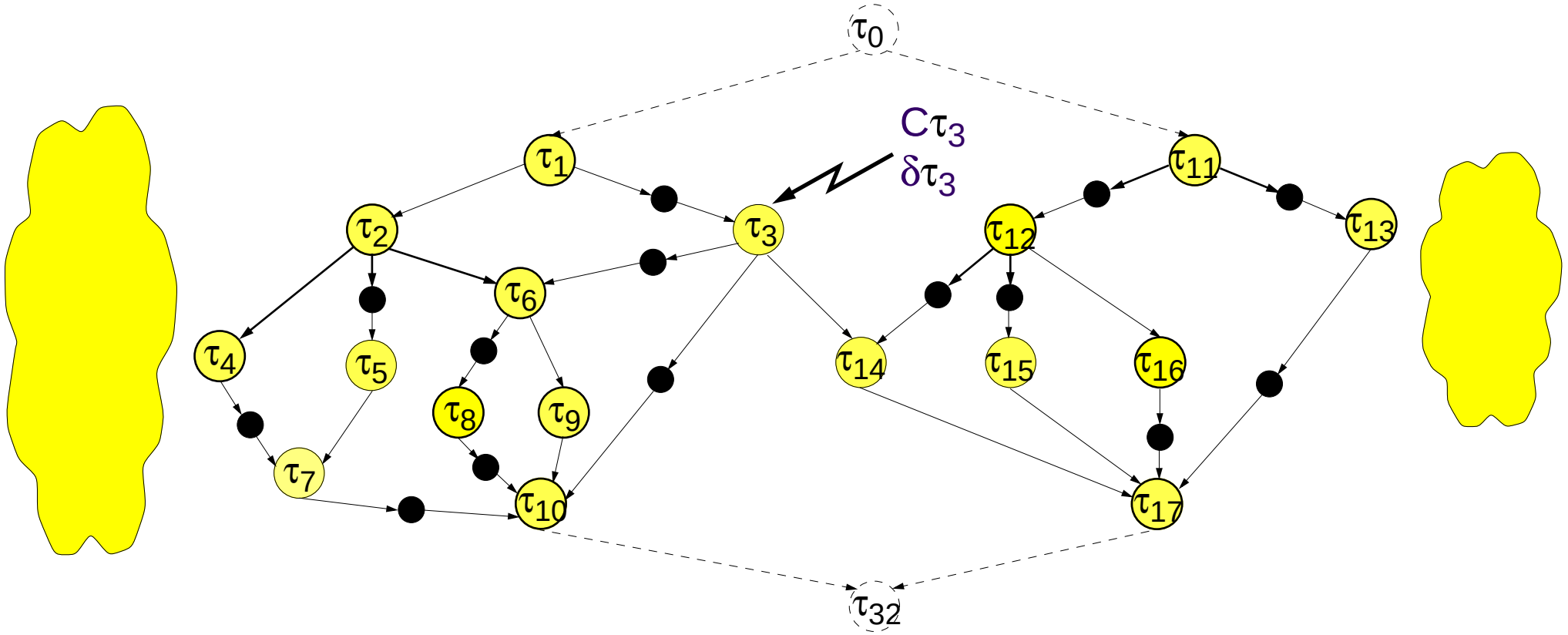
Γ_1
Period: $T_{\Gamma 1}$
Deadline: $D_{\Gamma 1}$

Γ_2
Period: $T_{\Gamma 2}$
Deadline: $D_{\Gamma 2}$

Γ_3
Period: $T_{\Gamma 3}$
Deadline: $D_{\Gamma 3}$

Application Model

- An application is modelled as a set of task graphs:



Γ_1
Period: $T_{\Gamma 1}$
Deadline: $D_{\Gamma 1}$

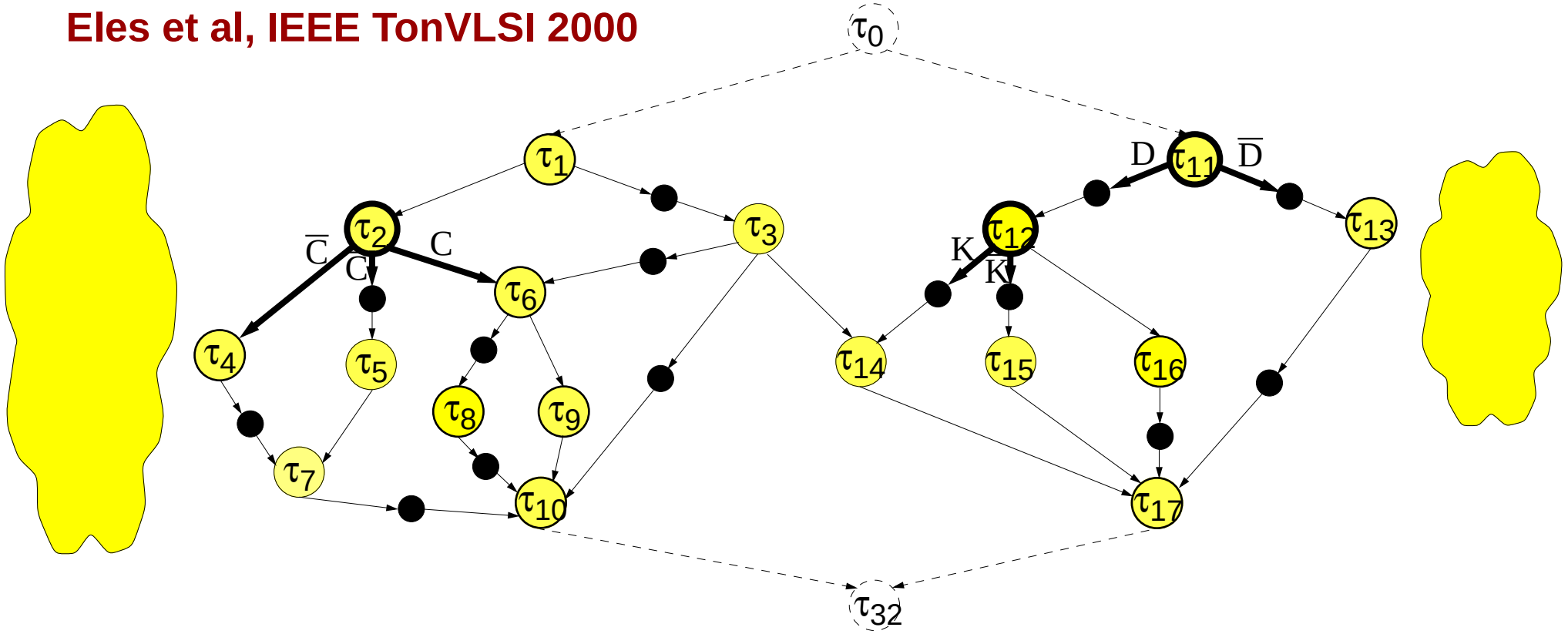
Γ_2
Period: $T_{\Gamma 2}$
Deadline: $D_{\Gamma 2}$

Γ_3
Period: $T_{\Gamma 3}$
Deadline: $D_{\Gamma 3}$

Application Model

- An application is modelled as a set of task graphs:

  Eles et al, IEEE TonVLSI 2000



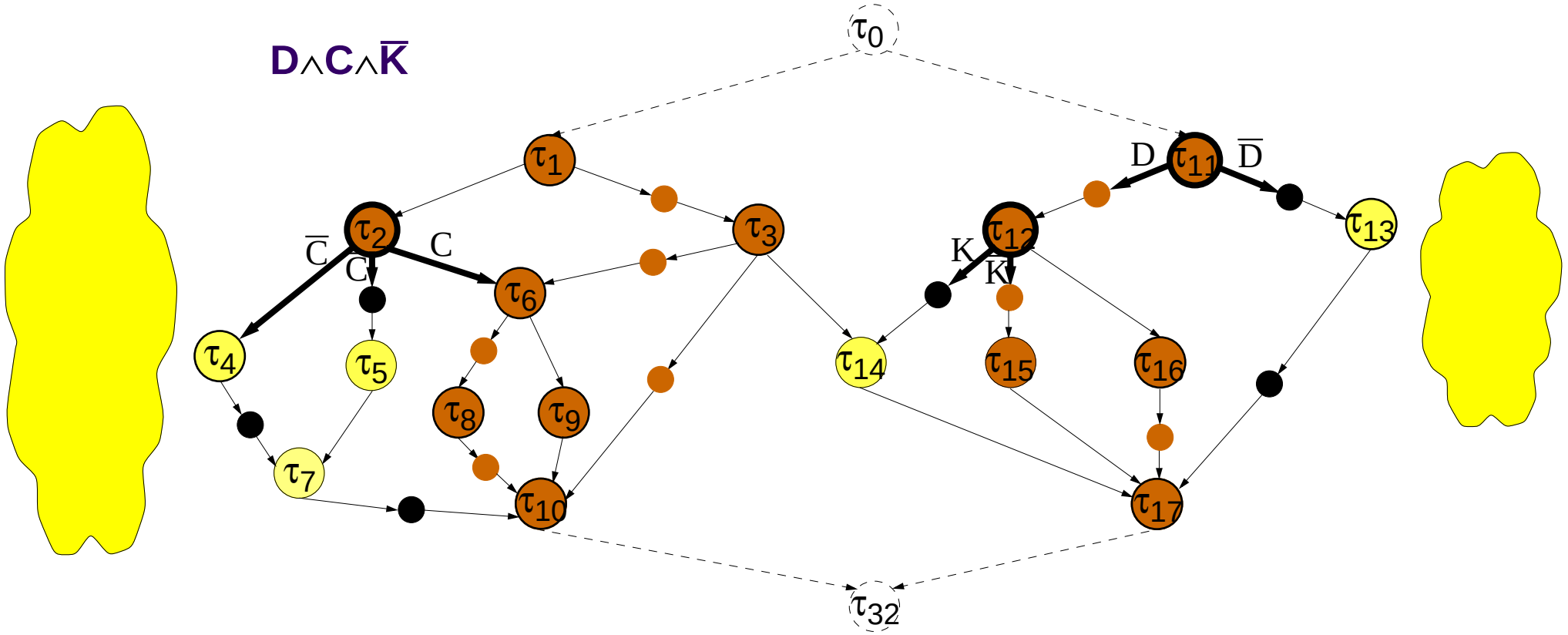
Γ_1
Period: $T_{\Gamma 1}$
Deadline: $D_{\Gamma 1}$

Γ_2
Period: $T_{\Gamma 2}$
Deadline: $D_{\Gamma 2}$

Γ_3
Period: $T_{\Gamma 3}$
Deadline: $D_{\Gamma 3}$

Application Model

- An application is modelled as a set of task graphs:



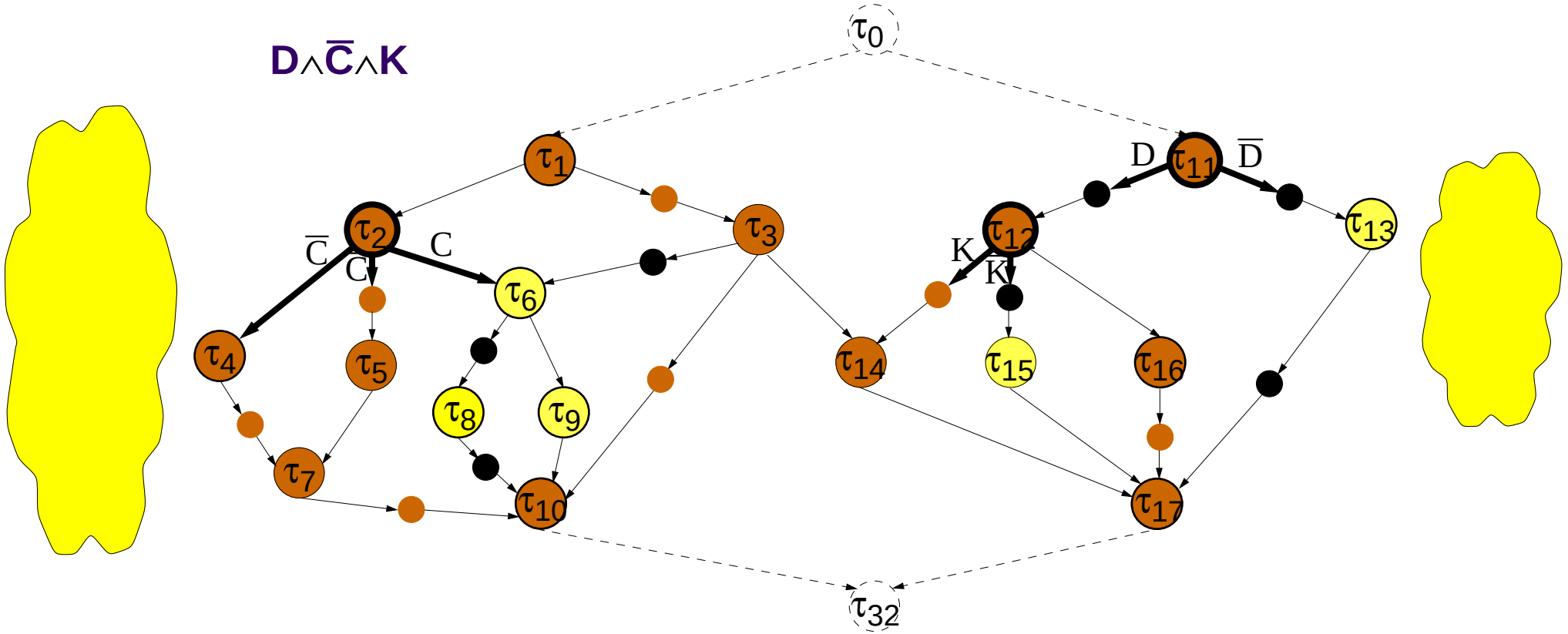
Γ_1
Period: $T_{\Gamma 1}$
Deadline: $D_{\Gamma 1}$

Γ_2
Period: $T_{\Gamma 2}$
Deadline: $D_{\Gamma 2}$

Γ_3
Period: $T_{\Gamma 3}$
Deadline: $D_{\Gamma 3}$

Application Model

- An application is modelled as a set of task graphs:



Γ_1
Period: $T_{\Gamma 1}$
Deadline: $D_{\Gamma 1}$

Γ_2
Period: $T_{\Gamma 2}$
Deadline: $D_{\Gamma 2}$

Γ_3
Period: $T_{\Gamma 3}$
Deadline: $D_{\Gamma 3}$

Heterogeneous Distributed Real-Time Systems

- RT Design Approach

- Network Protocol

Heterogeneous Distributed Real-Time Systems

- RT Design Approach
  - Time triggered
  - Event triggered
- Network Protocol

Heterogeneous Distributed Real-Time Systems

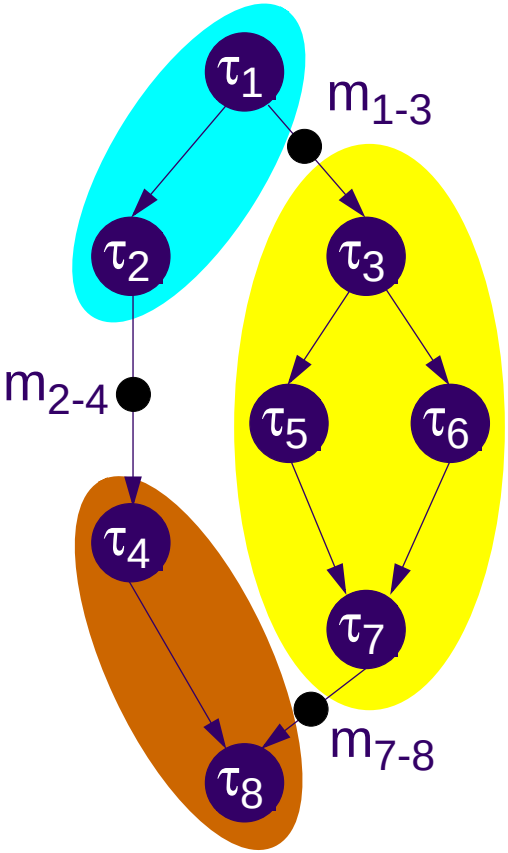
- RT Design Approach
  - Time triggered
  - Event triggered
- Network Protocol
  - Static
  - Dynamic

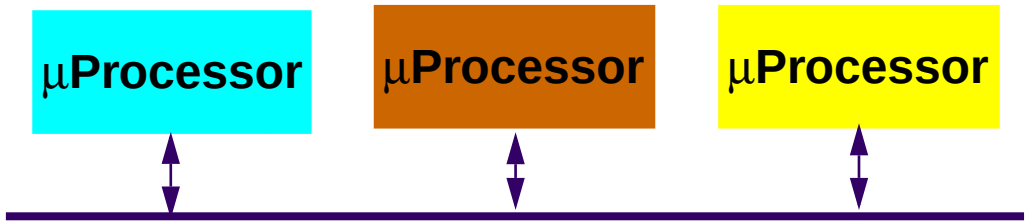
Time Triggered Systems

- The execution of tasks is initiated at pre-determined moments in time.

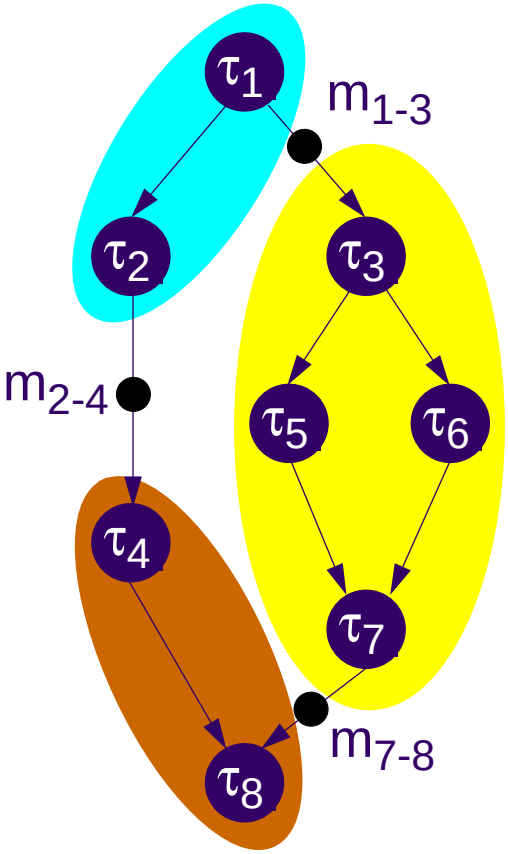
- Task initiation is performed by the real-time kernel typically based on information stored in a schedule table.

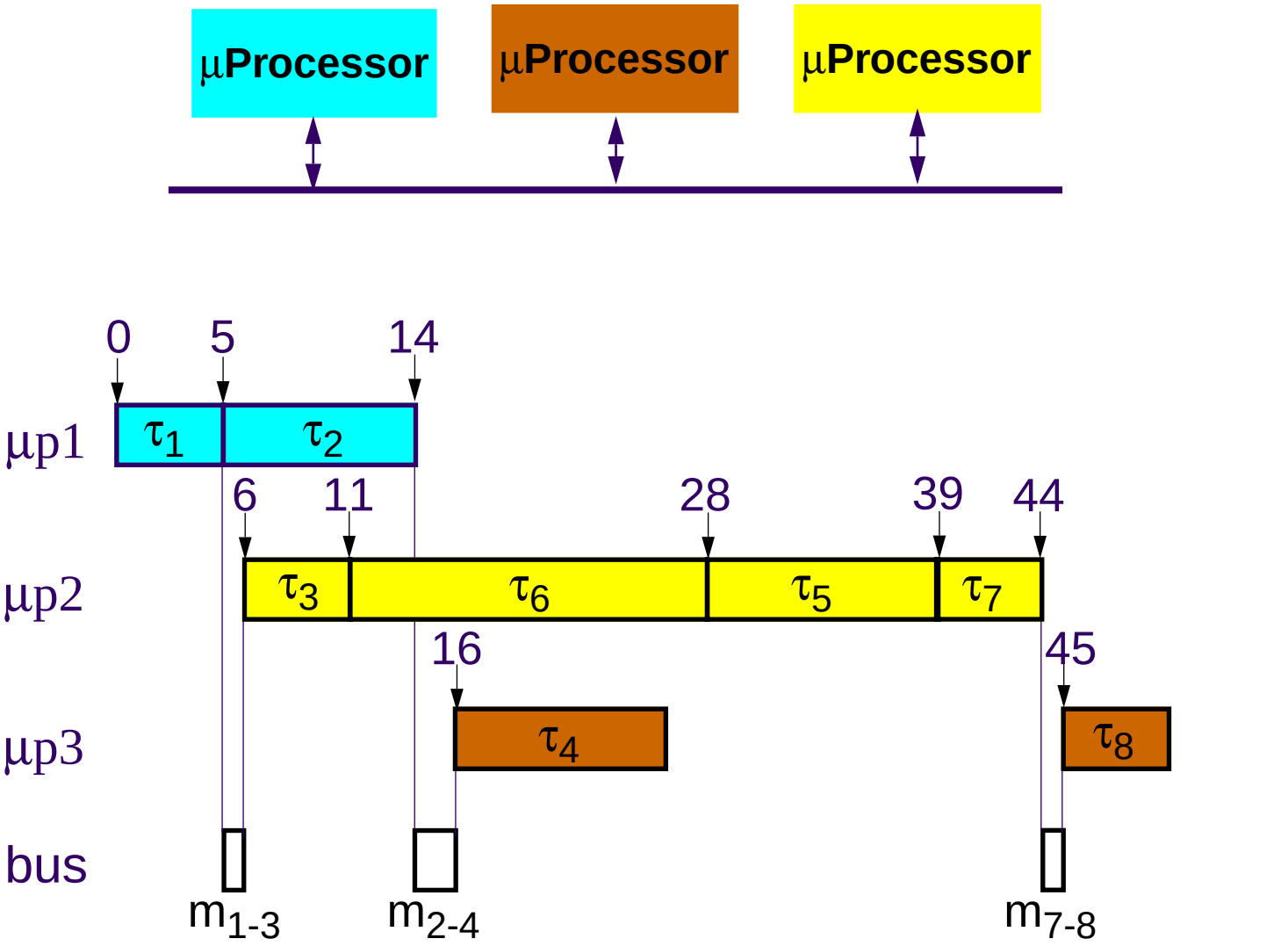
Time Triggered Systems





Time Triggered Systems





Time Triggered Systems

Scheduling → **Schedule table**

Static cyclic schedule

Task/Comm.	Start Time
τ_1	0
τ_2	5
τ_3	6
τ_4	16
τ_5	28
τ_6	11
τ_7	39
τ_8	45
$m_{1\text{-}3}$	5
$m_{2\text{-}4}$	14
$m_{7\text{-}8}$	44

Over a Hyperperiod

Time Triggered Systems

Scheduling → **Schedule table**

Static cyclic schedule

The system is schedulable if it is possible to build a schedule table such that all deadlines are satisfied.

Task/Comm.	Start Time
τ_1	0
τ_2	5
τ_3	6
τ_4	16
τ_5	28
τ_6	11
τ_7	39
τ_8	45
$m_{1\text{-}3}$	5
$m_{2\text{-}4}$	14
$m_{7\text{-}8}$	44

Over a Hyperperiod

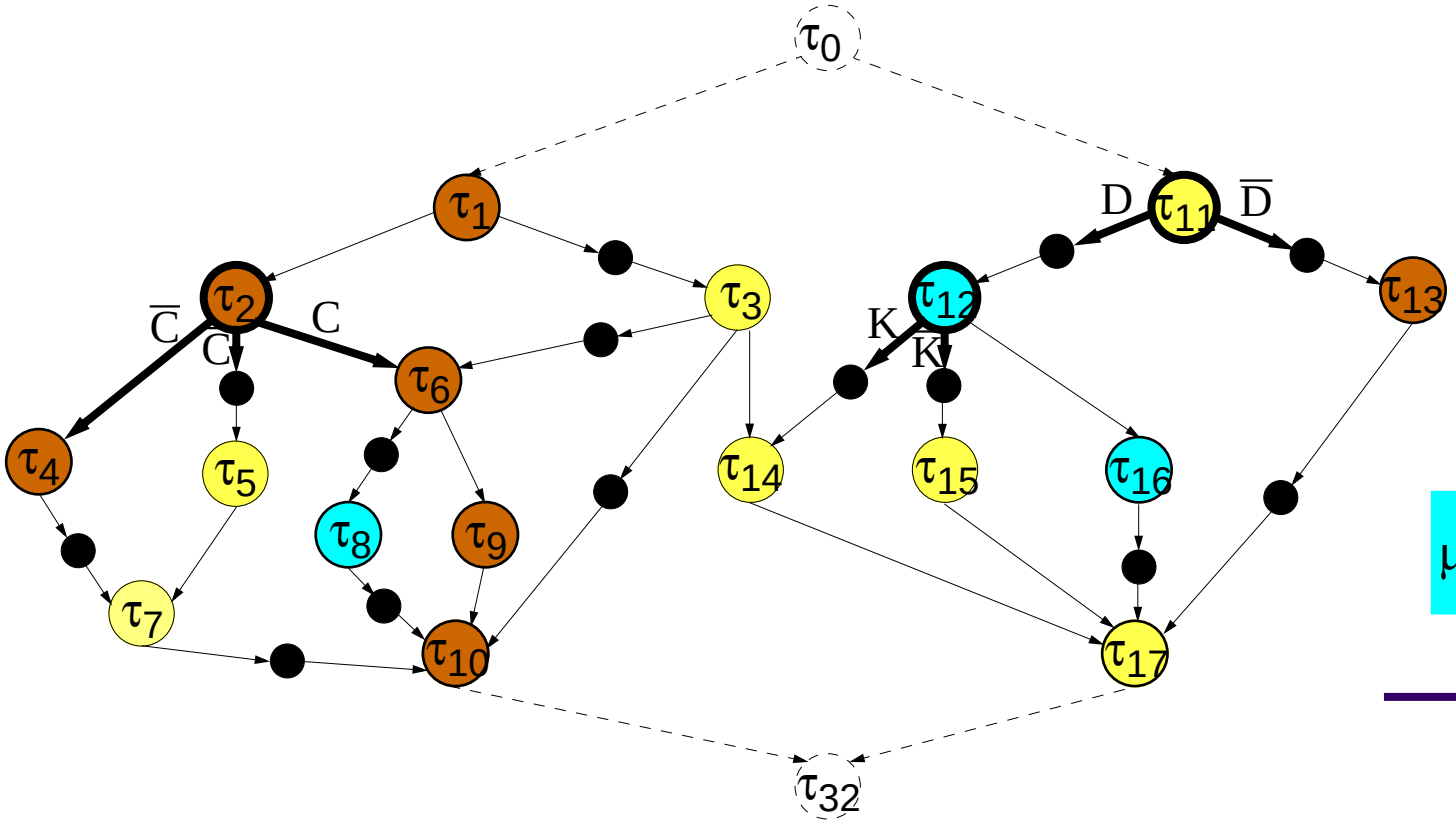
Time Triggered Systems

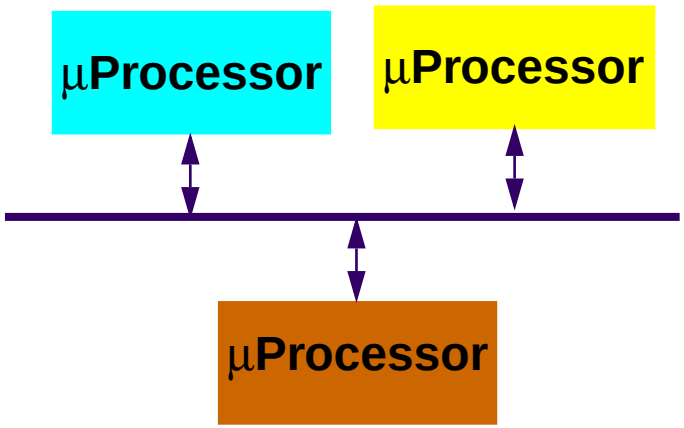
- In the case of Conditional Task Graphs we cannot build a static schedule

Quasi-static scheduling

Time Triggered Systems

- In the case of Conditional Task Graphs we cannot build a static schedule





Eles et al, IEEE TonVLSI 2000

	true	*D*	*D*∧*C*	*D*∧*C*∧$\overline{K}$	*D*∧*C*∧*K*	*D*∧$\overline{C}$	*D*∧$\overline{C}$∧$\overline{K}$	*D*∧$\overline{C}$∧*K*	$\overline{D}$	$\overline{D}$∧*C*	$\overline{D}$∧$\overline{C}$
τ_1	0										
τ_2	3										
τ_{10}				34	34		26	26		34	26
τ_{11}	0										
τ_{14}					35			24			
τ_{17}				29	37		30	26		22	24
τ_{18} (1→3)	3										
τ_{19} (2→5)						9					10
τ_{20} (3→10)				28	20		21	21		22	18
D	6										
C		7							7		
K			15			15					

Time Triggered Systems

Why do we like (quasi)static cyclic scheduling?

- **High predictability**
- **Easy to debug/validate**
- **Low execution time overhead**

Suitable for safety-critical applications

Time Triggered Systems

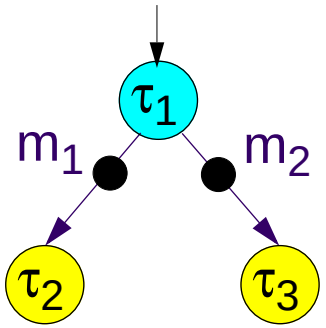
Some problems with (quasi)static cyclic scheduling:

- Not flexible:
  - quality degrades rapidly if periods and execution times deviate from those predicted;
  - if new tasks are added, the whole schedule has to be regenerated.
- Static scheduling for large task sets is computationally very expensive.
- Urgent events (interrupts) are handled purely:
  - time slots are statically allocated for polling and handling such events.
- Very long hyper-periods have to be avoided:
  - the periods of individual tasks have to be adjusted;
    this can lead to artificially reduced periods $\Rightarrow$ artificially increased load
    $\Rightarrow$ waste of processor time.
- Tasks have to be “manually” split, in order to make the system schedulable.

Event Triggered Systems

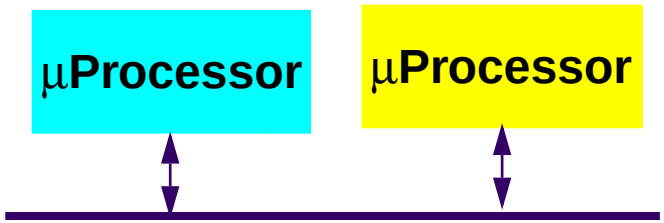
- The execution of tasks is initiated by the occurrence of a certain event.
- Task initiation is performed by the real-time kernel which selects and executes the ready task with the highest priority.
- Task scheduling is, typically, preemptive.
- No schedule (predetermined activation times) is generated off-line.

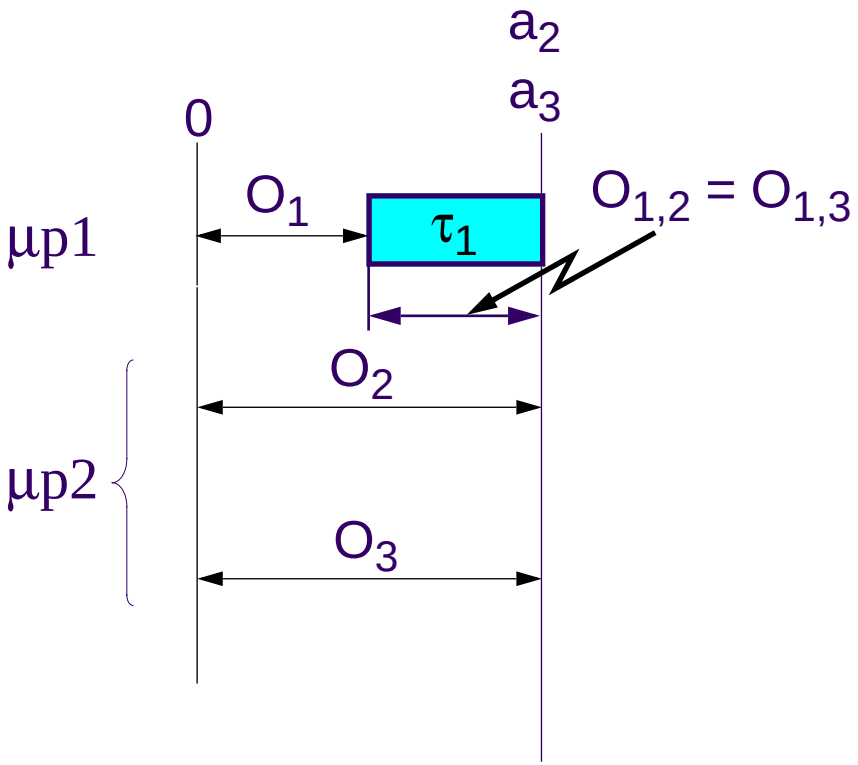
Event Triggered Systems



priority$_{m1}$ < priority$_{m2}$

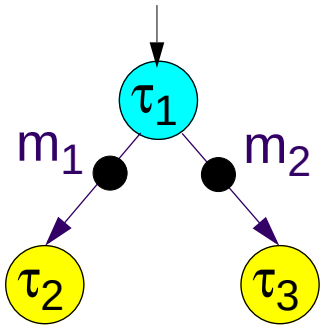
priority$_{\tau 2}$ > priority$_{\tau 3}$





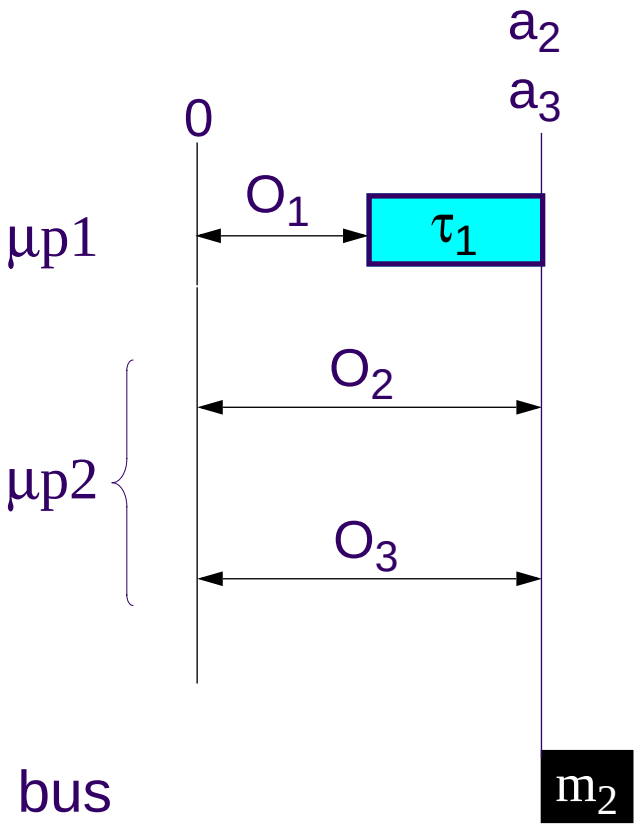
- **Arrival time a_i: The time at which τ_i becomes ready for execution.**
- **Offset O_i: The earliest possible arrival time of τ_i.**

Event Triggered Systems

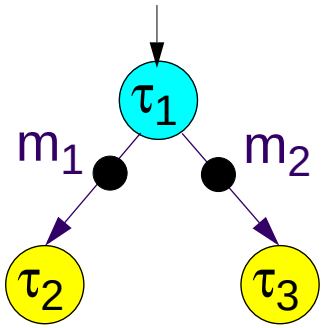


$priority_{m1} < priority_{m2}$

$priority_{\tau 2} > priority_{\tau 3}$

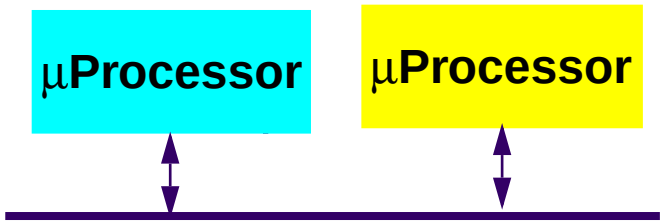


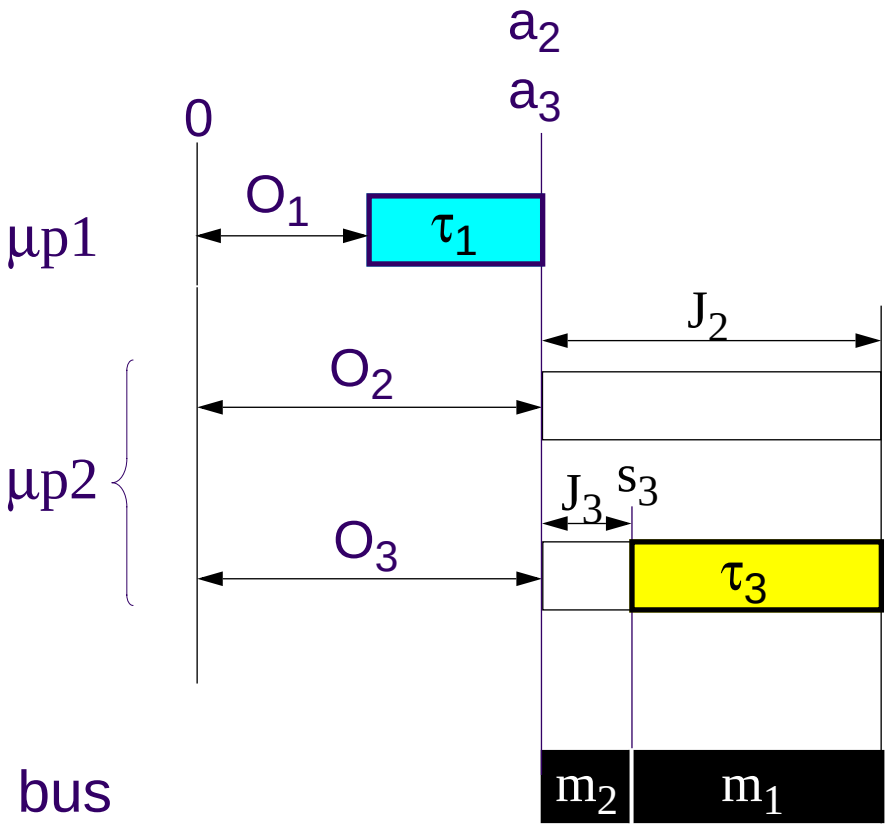
Event Triggered Systems



priority$_{m1}$ < priority$_{m2}$

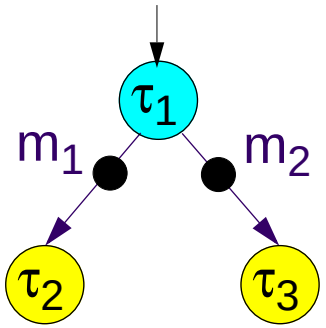
priority$_{\tau 2}$ > priority$_{\tau 3}$





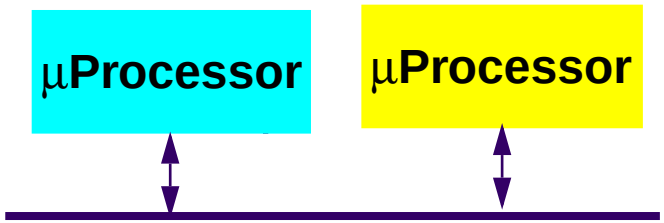
- **Start time s_i: Time when a task starts execution.**
- **Release jitter J_i: The delay between the arrival of τ_i and the start of its execution.**

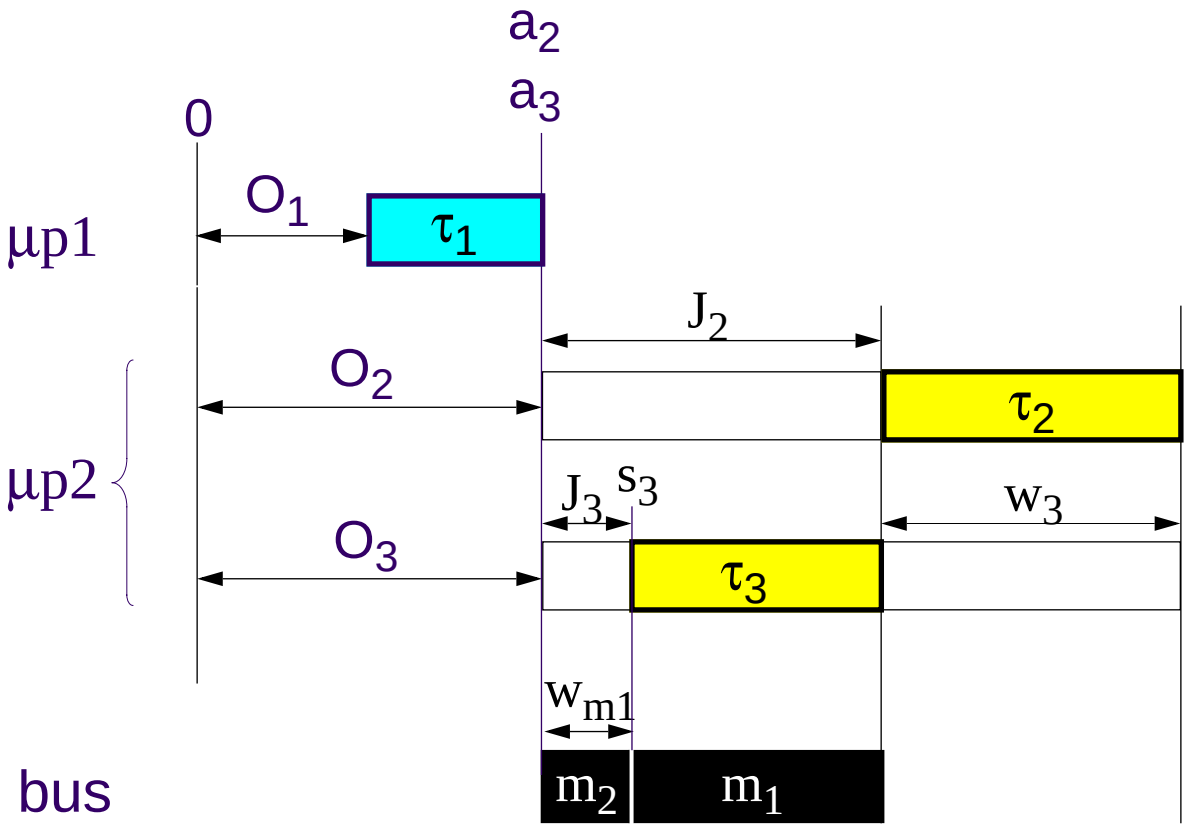
Event Triggered Systems



$priority_{m1} < priority_{m2}$

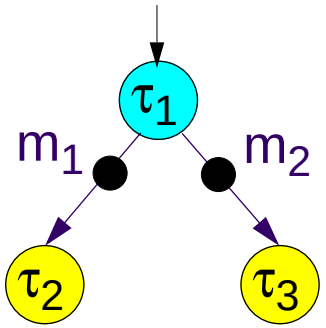
$priority_{\tau_2} > priority_{\tau_3}$





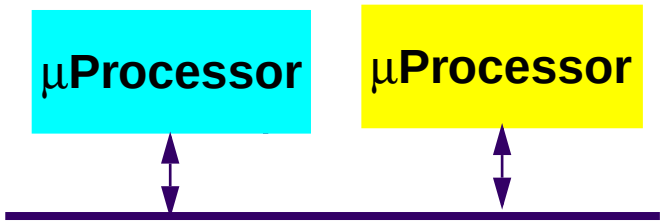
- Interference w_i: The time τ_i is preempted by higher priority tasks.
- Queuing delay w_{mi}: The delay experienced by m_i before being sent.

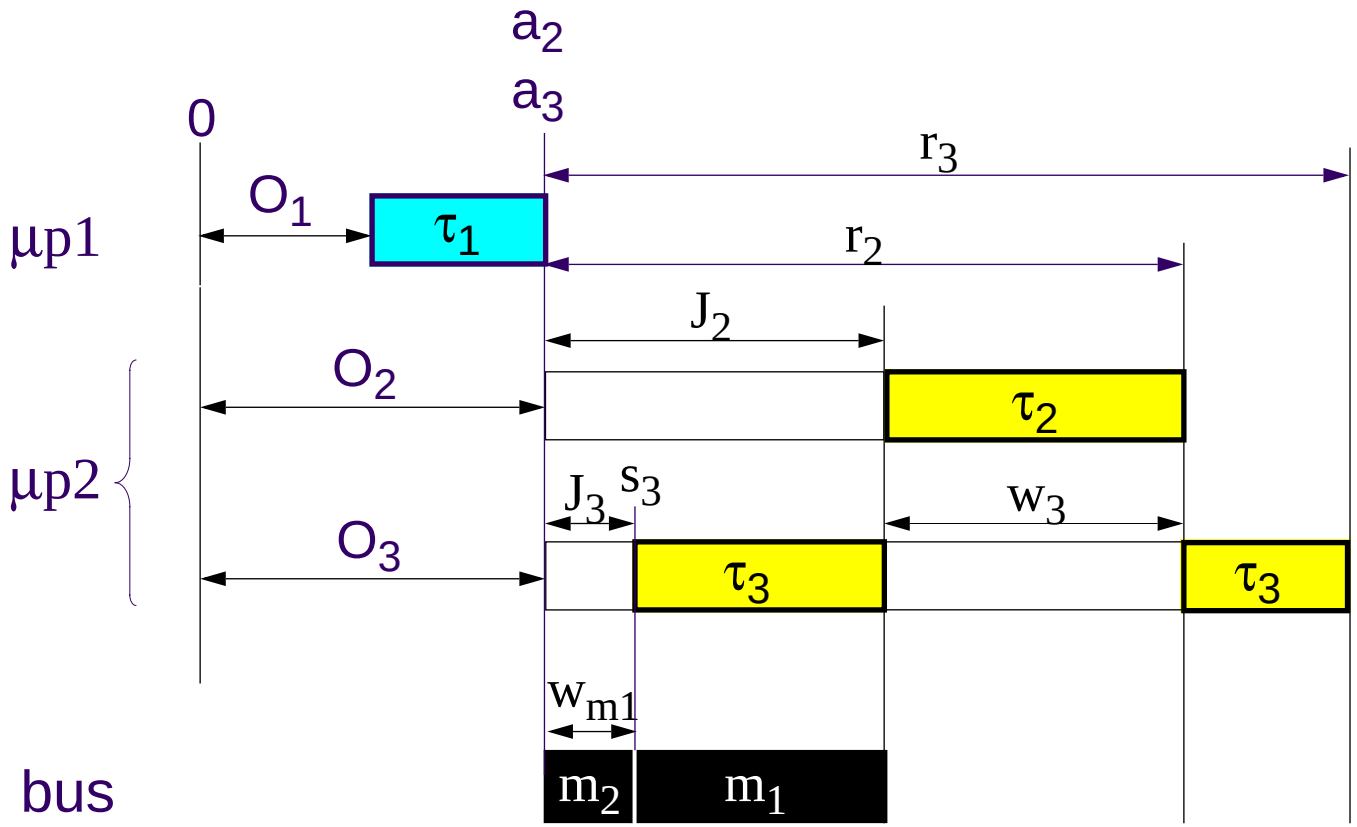
Event Triggered Systems



priority$_{m1}$ < priority$_{m2}$

priority$_{\tau 2}$ > priority$_{\tau 3}$





- **Response time r_i: The time from the arrival of τ_i until it finishes execution.**

Schedulability analysis (is the system schedulable?)

Schedulability analysis (is the system schedulable?)

- Utilisation based tests
  - Feasibility test for processes with RM priority assignment (Liu&Layland '73):

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$$

  - Problems:
    - Such tests are valid only under restrictive conditions.
    - They are pessimistic.
    - Necessary and sufficient tests (if available) are computationally expensive.

Schedulability analysis (is the system schedulable?)

- Response time analysis
  - Calculate worst case response time r_i for each task τ_i.
    If, for each task $r_i \leq D_i$, then and only then the task set is schedulable.
  - The basic equation (Audsley et al '91):

$$r_i = C_i + \sum_{\forall j \in hp(\tau_i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j$$

This is the interference w_i from higher priority tasks.

  - Problem
    - The above formula is exact (necessary and sufficient) only in particular cases (monoprocessor, independent tasks, etc.).
    - In more general cases (e.g. data dependencies) it is very pessimistic.

Response time calculation for data dependent tasks (Tindell '94):

$r_i = J_i + w_i + C_i$ **, where the interference is**

$$w_i = \sum_{\forall j \in hp(\tau_i)} \left\lceil \frac{w_i + J_j - O_{ij}}{T_j} \right\rceil_0 C_j$$

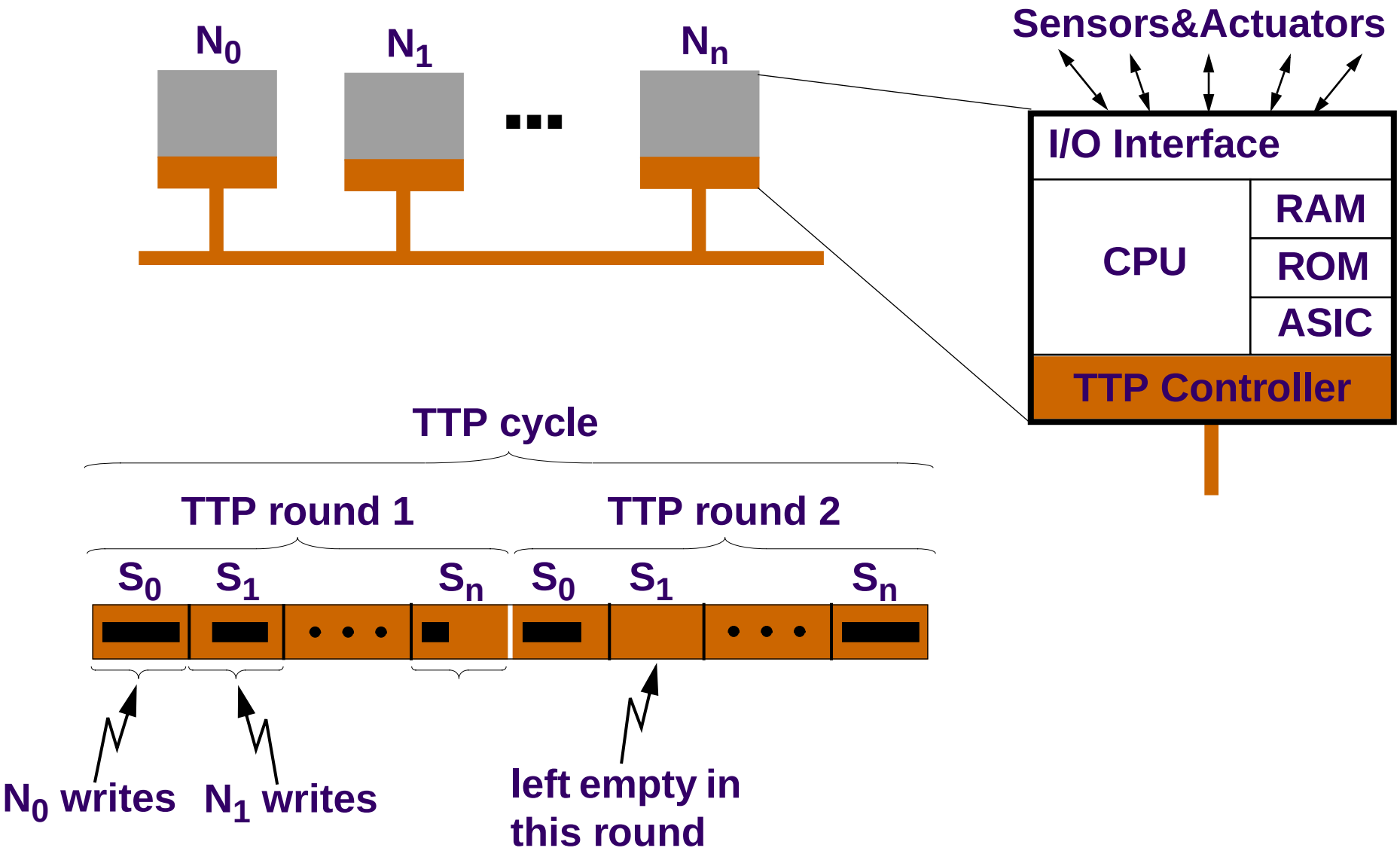
Response time calculation for data dependent tasks (Tindell '94):

$$r_i = J_i + w_i + C_i$$ **, where the interference w_i is:**

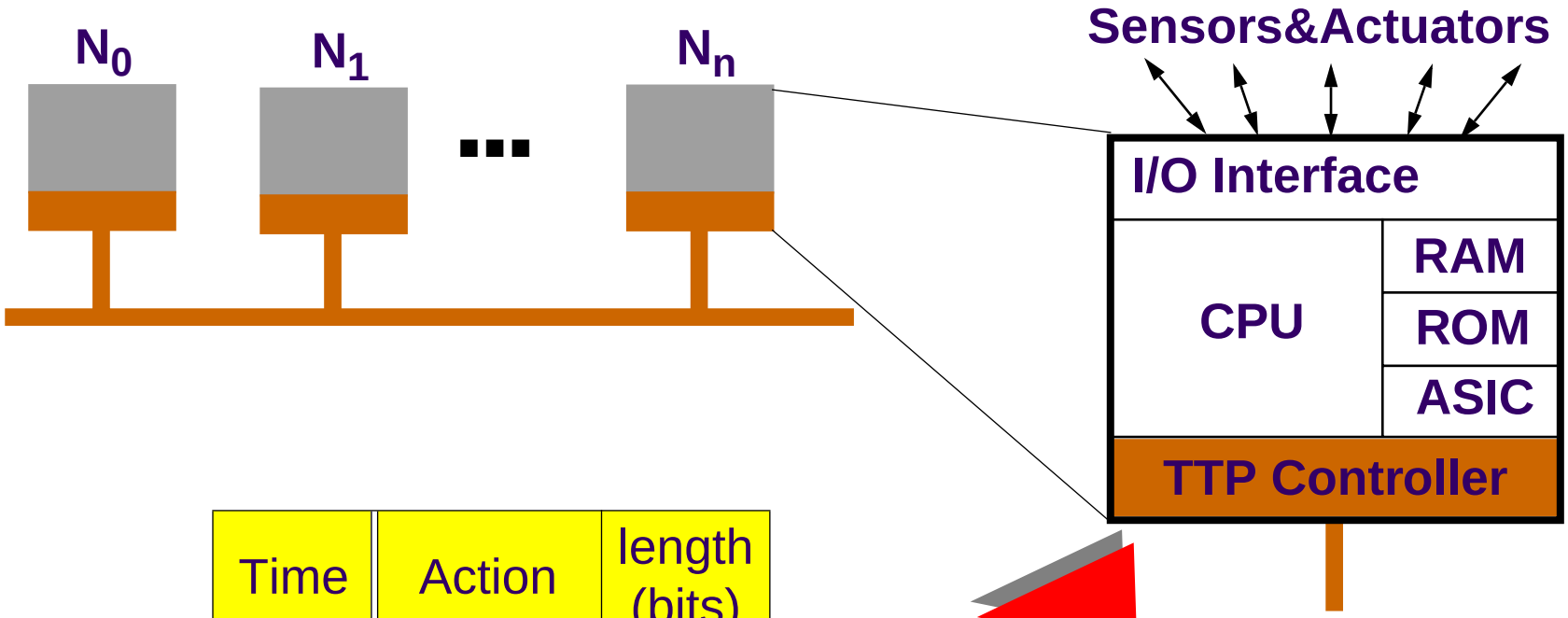
$$w_i = \sum_{\forall j \in hp(\tau_i)} \left\lceil \frac{w_i + J_j - O_{ij}}{T_j} \right\rceil_0 C_j$$

- **Main idea:**
  - **Use offsets and jitter to model dependencies**
- **The same idea can be used to model communication in distributed systems (the release jitter of the receiver task depends on the communication delay).**
- **Further reduce pessimism in the case with conditional execution (Pop, Eles, Peng, CODES 2000).**

Static Communication: TTP Network



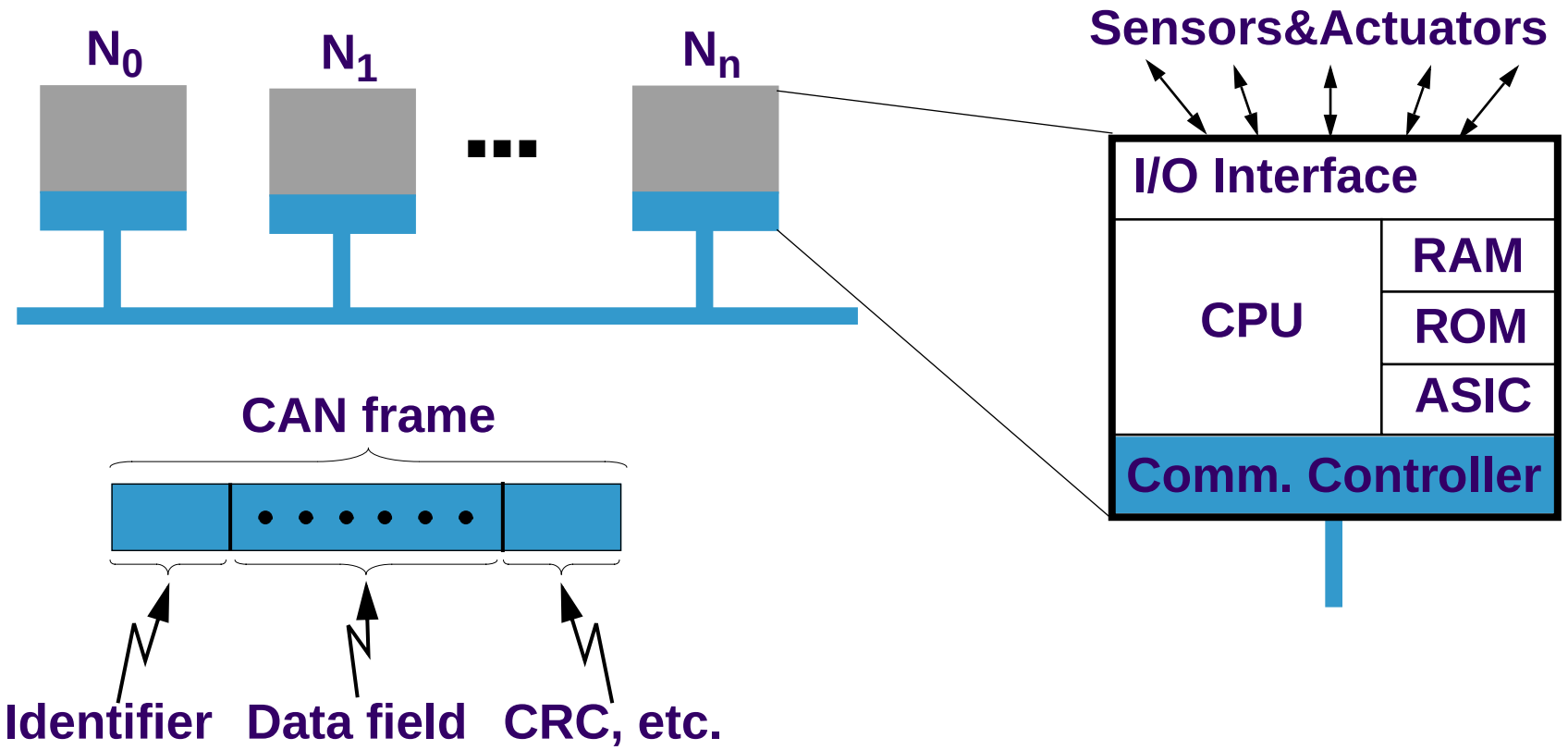
Static Communication: TTP Network



Over a TTP cycle

Time	Action	length (bits)
t_1	rd-frame	16
t_2	wr-frame	32
t_3	rd-frame	32
...	...	...
t_x	wr-frame	16

Dynamic Communication: CAN Network



- Priority bus with collision avoidance mechanism:

  The node that transmits the highest priority frame wins the contention.

Scheduling of Distributed RT Systems

- The classical approaches:
  - TT tasks over static network.

    (Eles et al, IEEE TonVLSI 2000)
  - ET tasks over dynamic network.

    (Tindell et al ‘95)

Scheduling of Distributed RT Systems

- The classical approaches:
  - TT tasks over static network.

    (Eles et al, IEEE TonVLSI 2000)
  - ET tasks over dynamic network.

    (Tindell et al '95)

- A first step towards heterogeneous systems:
  - TT tasks over dynamic network.

    (Dobrin et al, 2001)
  - ET tasks over static networks.

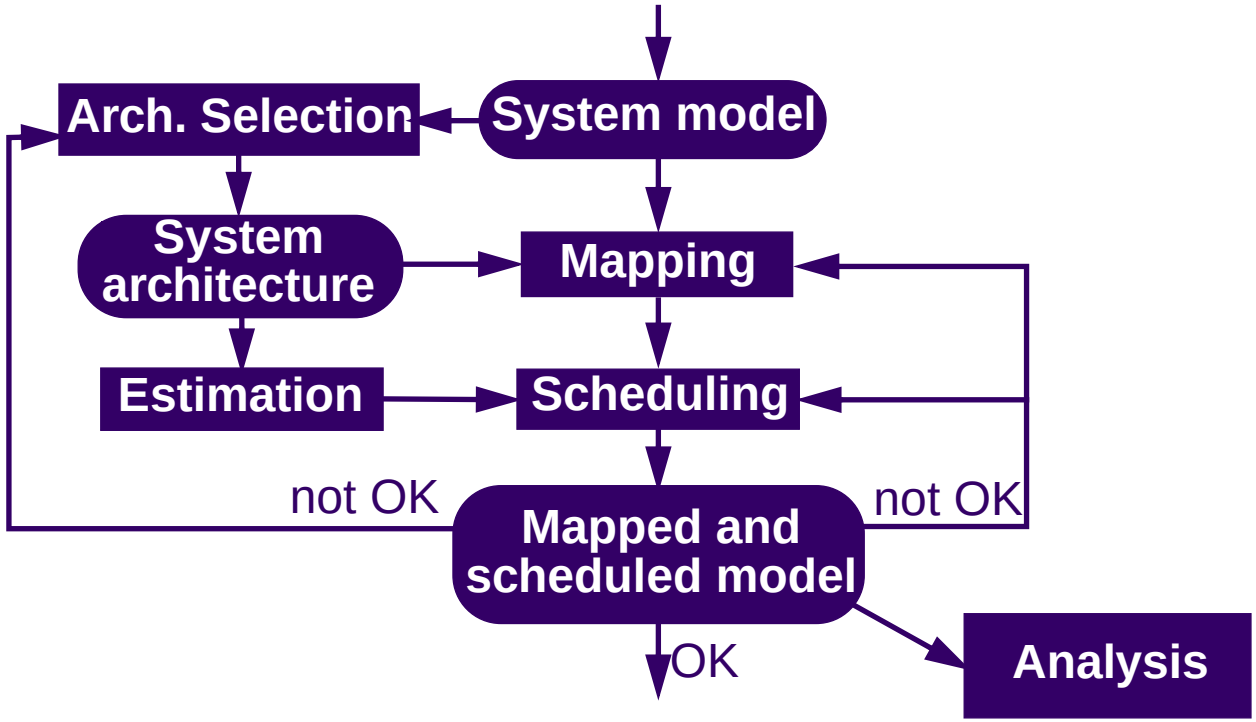
    (Tindell '94; Pop, Eles, Peng, DATE 2000 & RT Systems Journal 2003)

Optimization of Distributed RT Systems

- Once a scheduling/schedulability analysis approach is in place, several optimization tasks can be performed.

Optimization of Distributed RT Systems

- Once a scheduling/schedulability analysis approach is in place, several optimization tasks can be performed.



Optimization of Distributed RT Systems

- Once a scheduling/schedulability analysis approach is in place, several optimization tasks can be performed.

- Task mapping

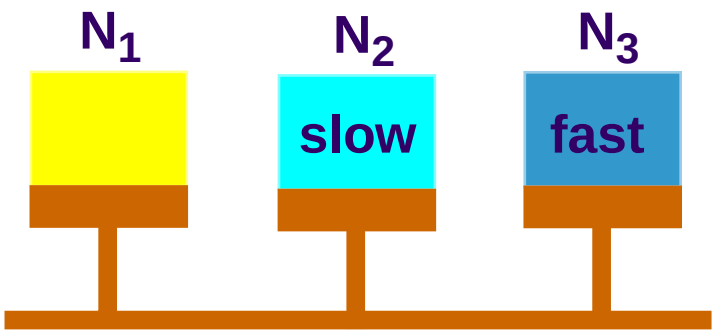
- Bus access optimization

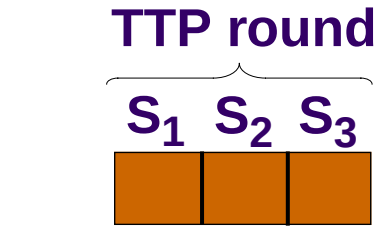
Pop, Eles, Peng, CODES ‘99

Pop, Eles, T. Pop, Peng, DAC ‘01

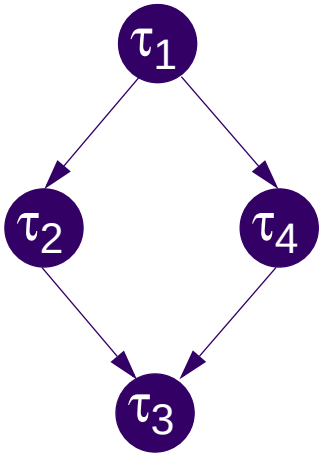
Pop, Eles, Peng, RT Systems Journal 2003

Task Mapping





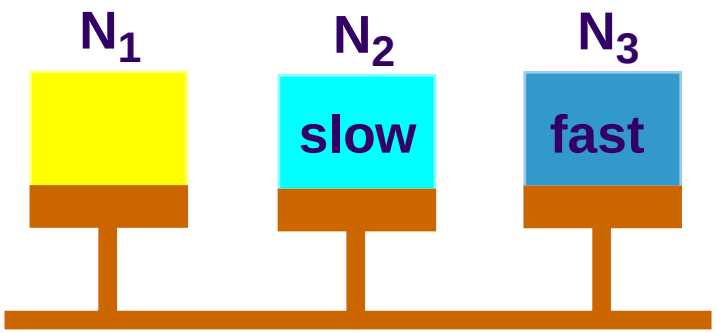
S1 = S2 = S3 = 4ms

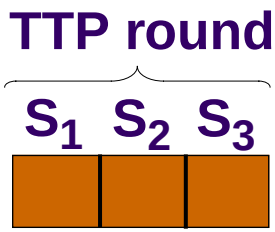


T = D = 50ms

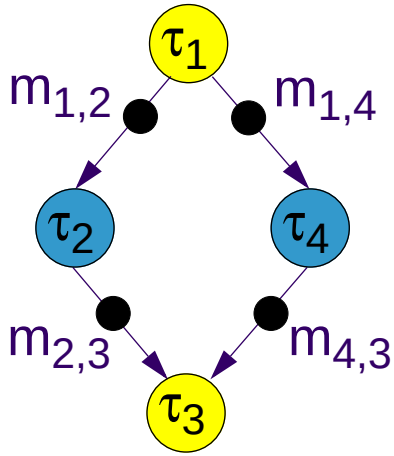
	N_1	N_2	N_3
τ_1	4ms	-	-
τ_2	-	12ms	8ms
τ_3	4ms	-	-
τ_4	-	12ms	8ms

Task Mapping

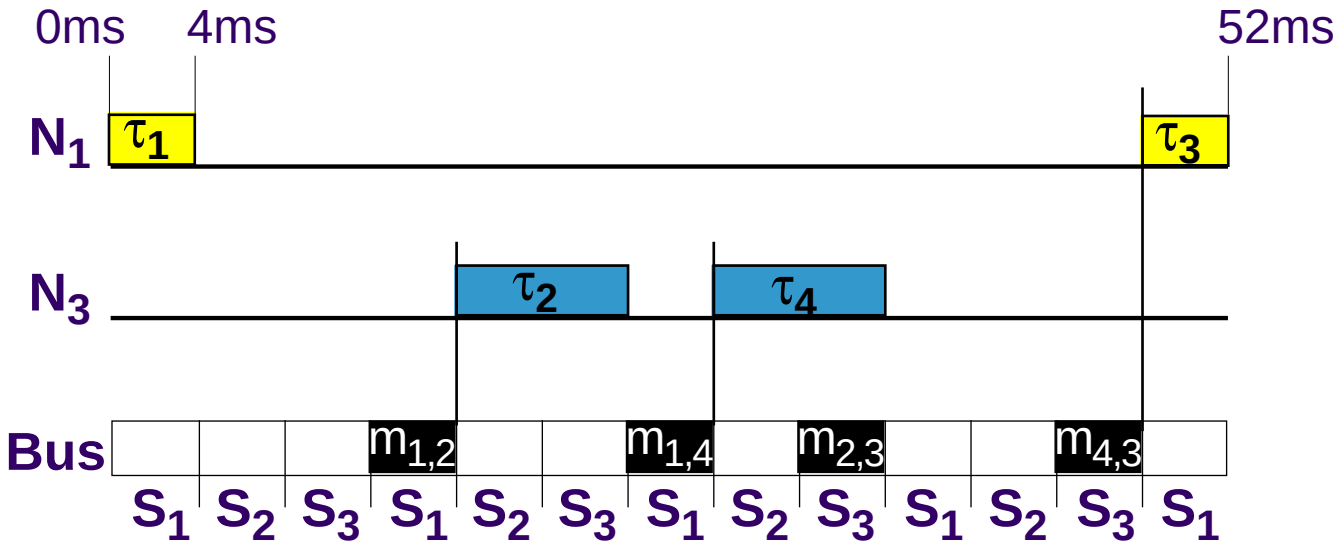




S1 = S2 = S3 = 4ms

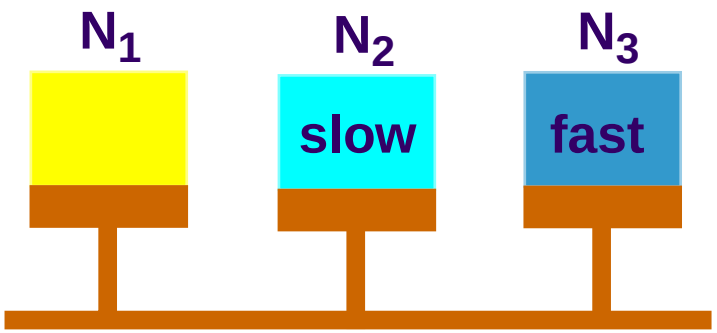


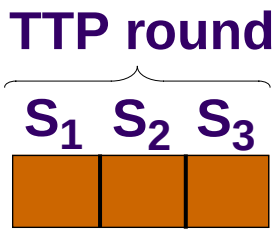
T = D = 50ms



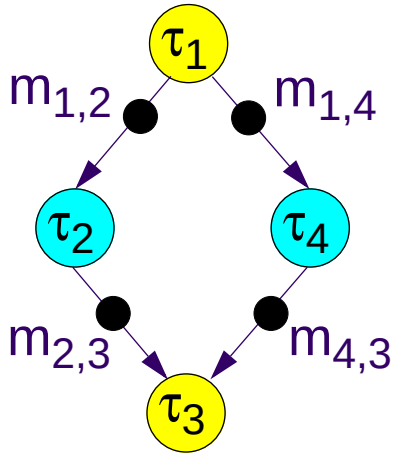
	N_1	N_2	N_3
τ_1	4ms	-	-
τ_2	-	12ms	8ms
τ_3	4ms	-	-
τ_4	-	12ms	8ms

Task Mapping

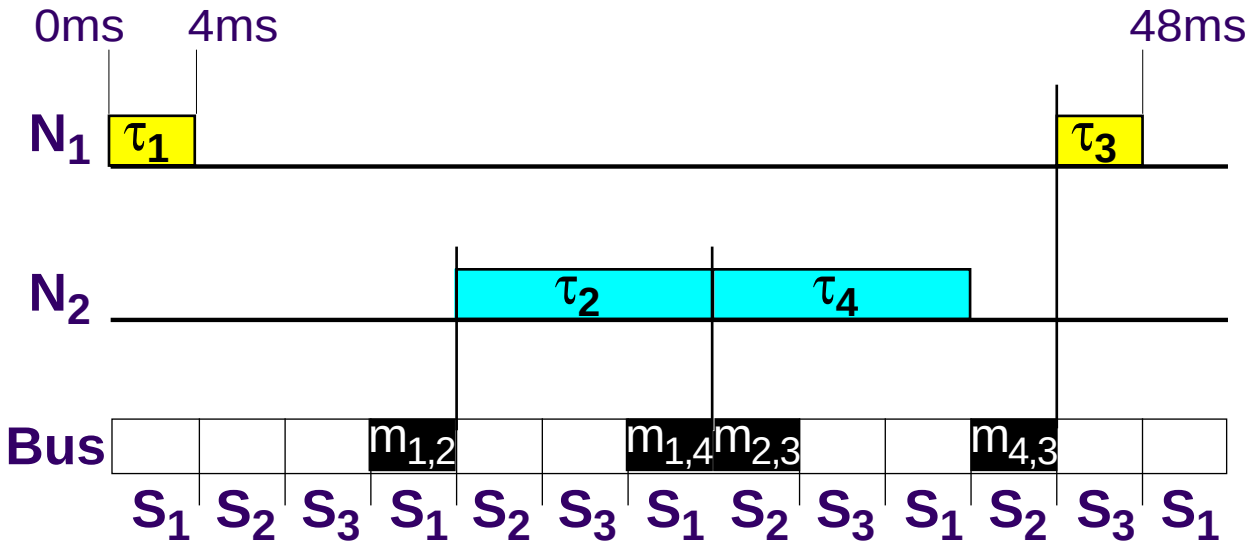




S1 = S2 = S3 = 4ms

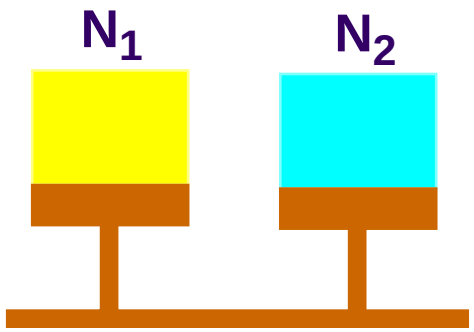


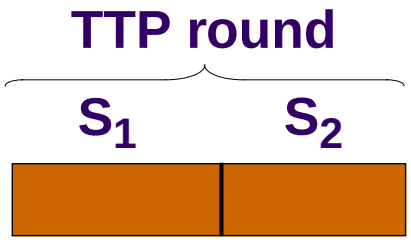
T = D = 50ms

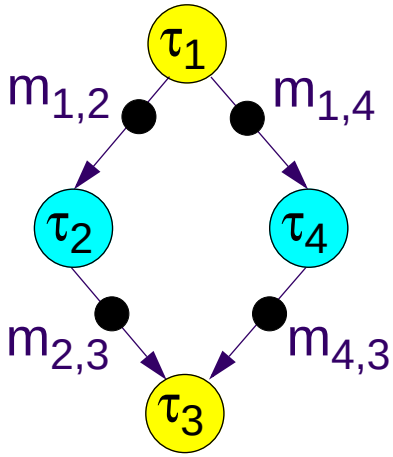


	N_1	N_2	N_3
τ_1	4ms	-	-
τ_2	-	12ms	8ms
τ_3	4ms	-	-
τ_4	-	12ms	8ms

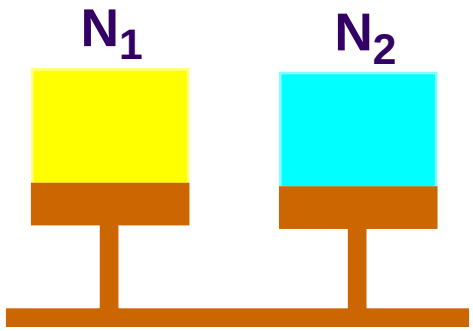
Bus Access Optimization

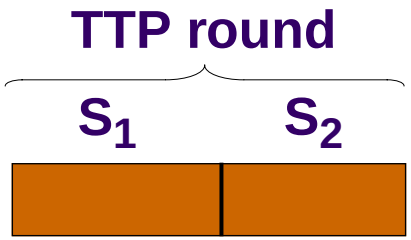


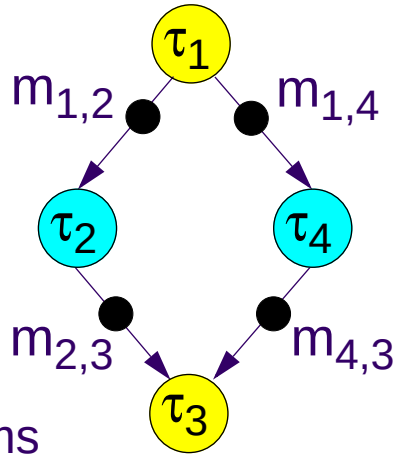


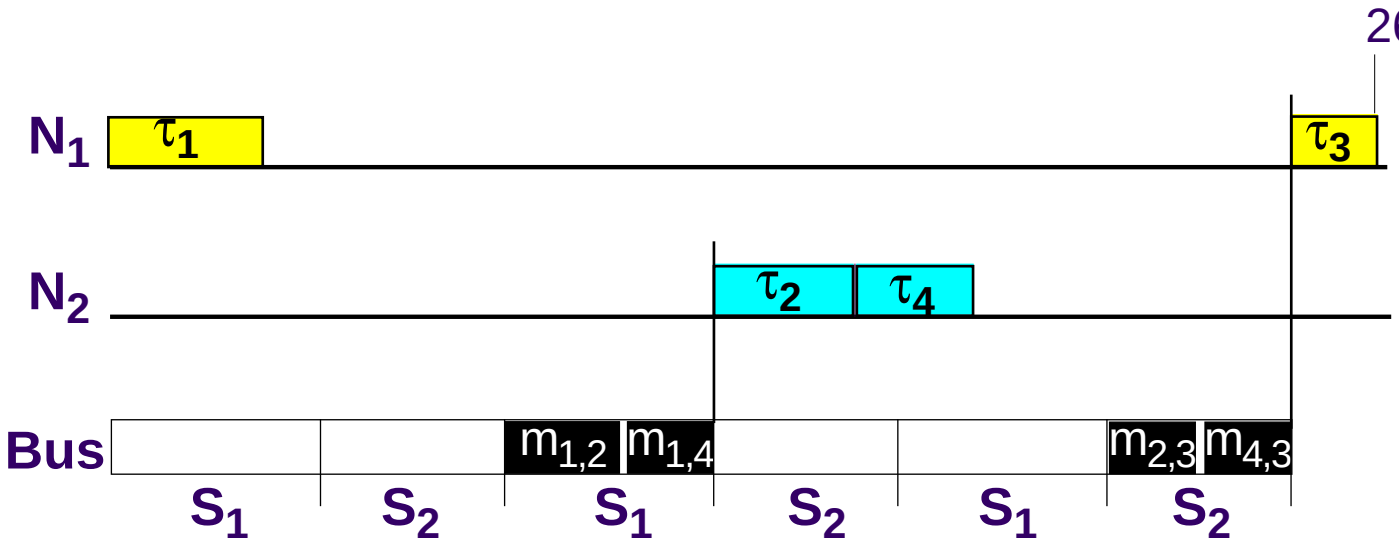


Bus Access Optimization

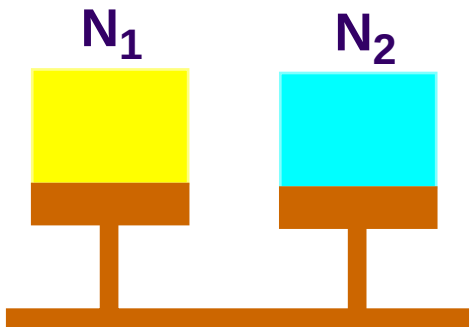


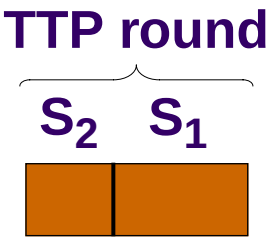


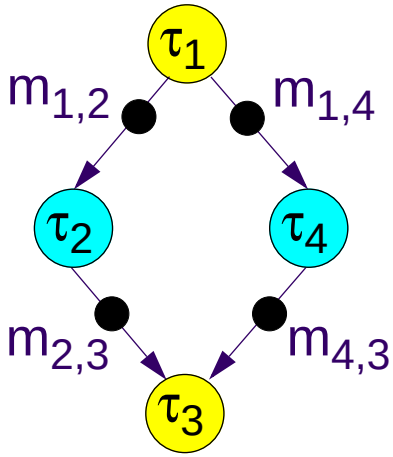


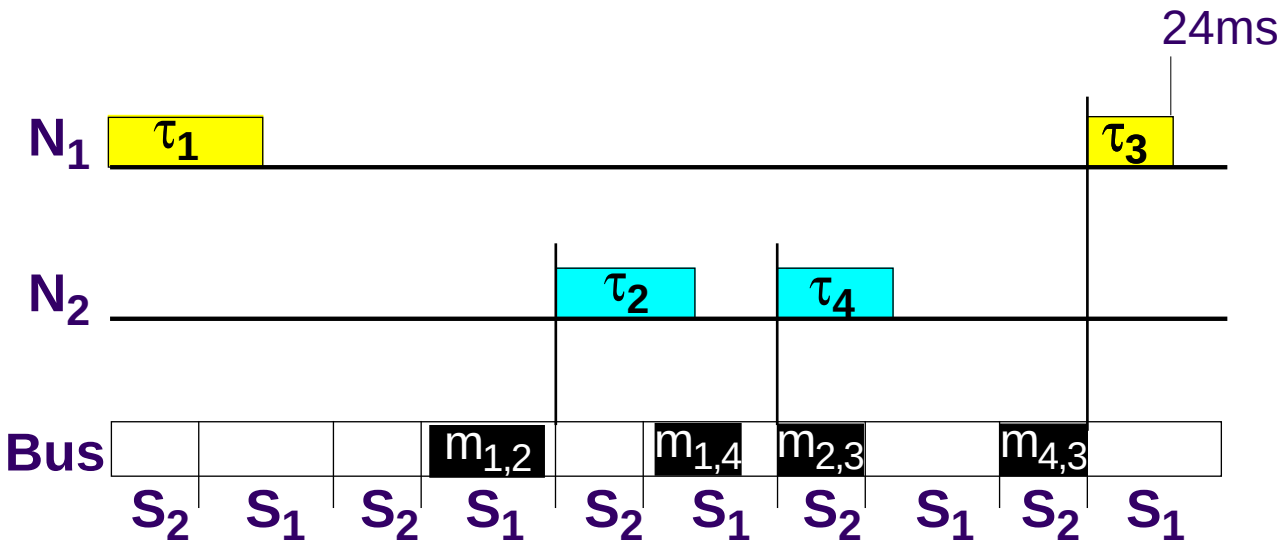


Bus Access Optimization

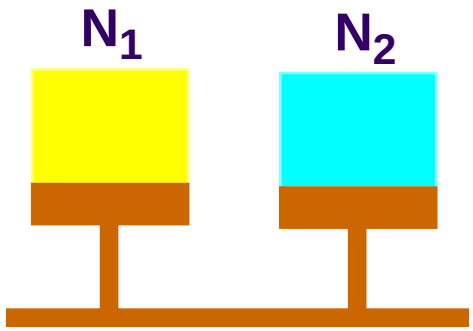


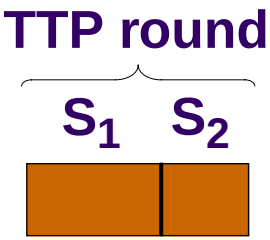


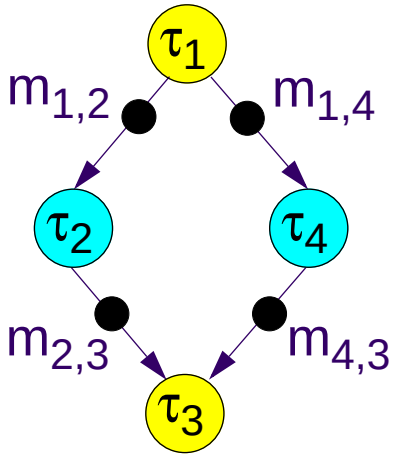


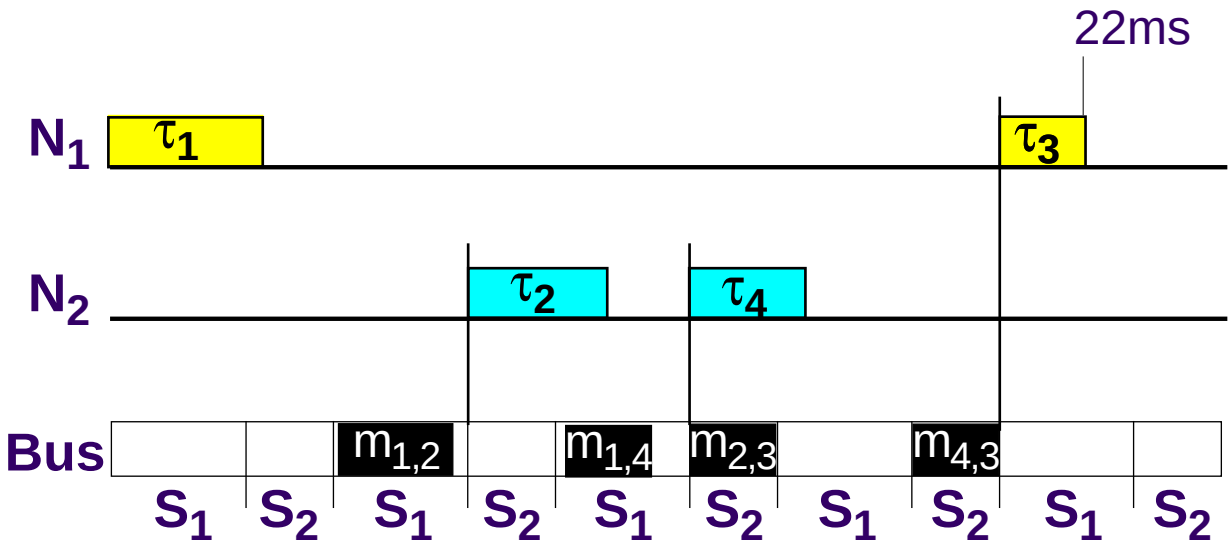


Bus Access Optimization







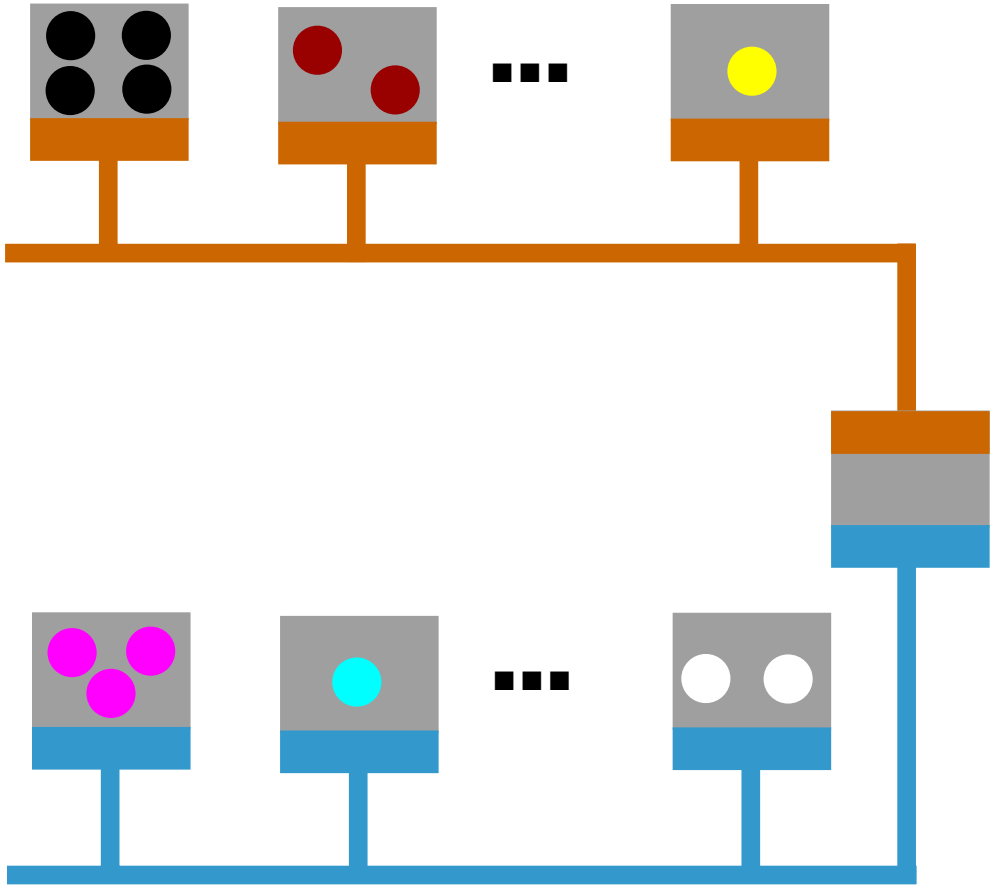


Outline

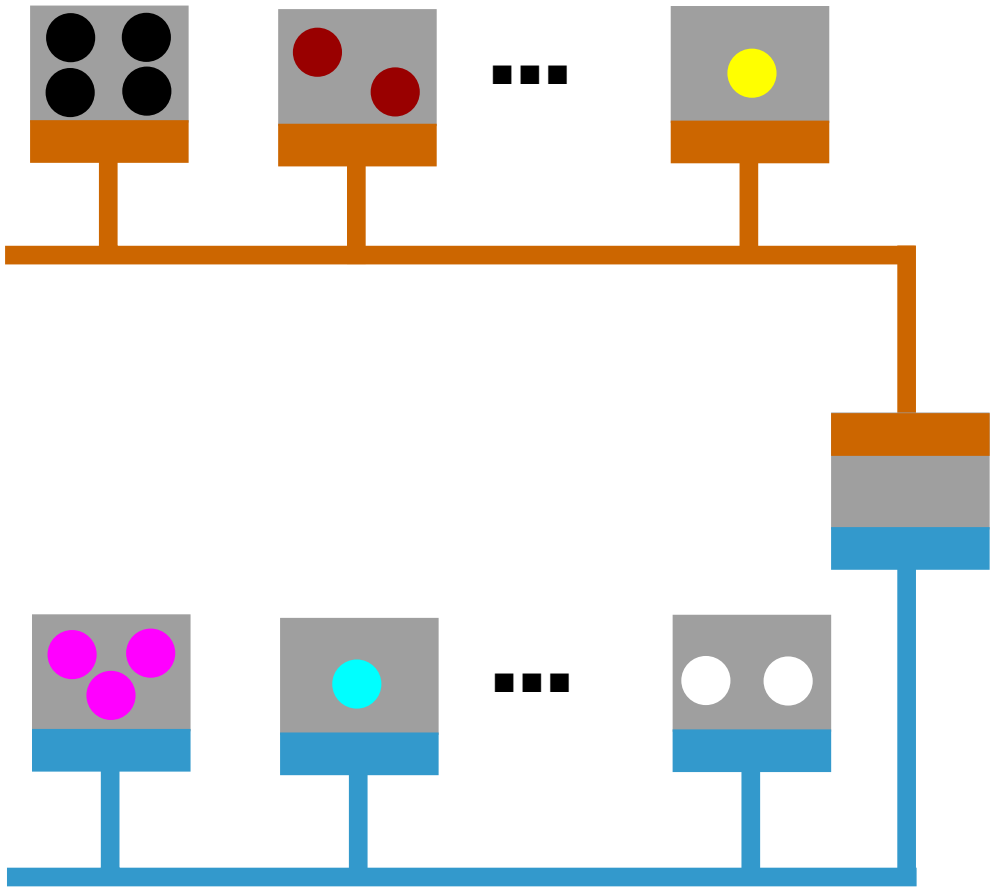
- Embedded Real-Time System
- System-level Design Flow
- Distributed Embedded Real-Time Systems
  - Application Model
  - Heterogeneous Systems
  - Time/Event Triggered Tasks
  - Static/Dynamic Communication
  - Analysis&Optimization
- **Single/Multi-cluster Heterogeneous Distributed Architectures**
  - **Analysis&Optimization**
- Incremental Design Process

The Traditional Approach

One node ⟷ One function



The Traditional Approach



One node ⟷ One function

- Purchase node&function and integrate into system
- Sophistication grew quickly

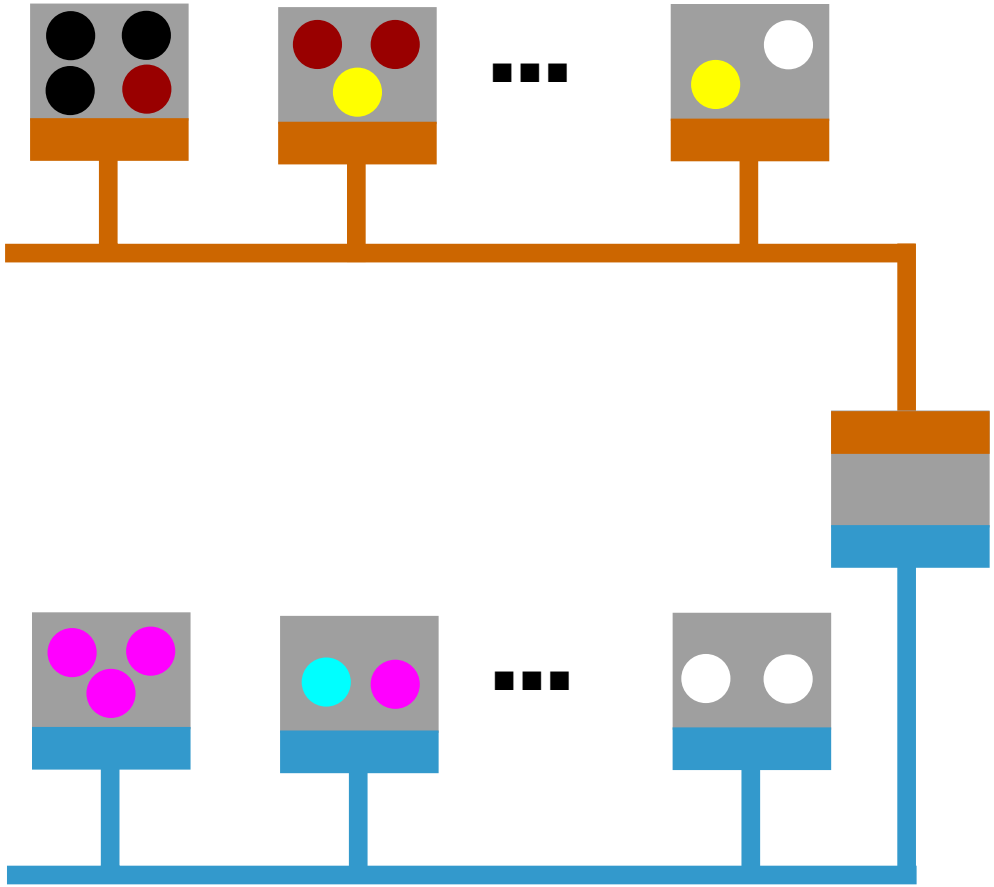
↓

More then 100 nodes

↓

Resources have to be used more efficiently

What Comes



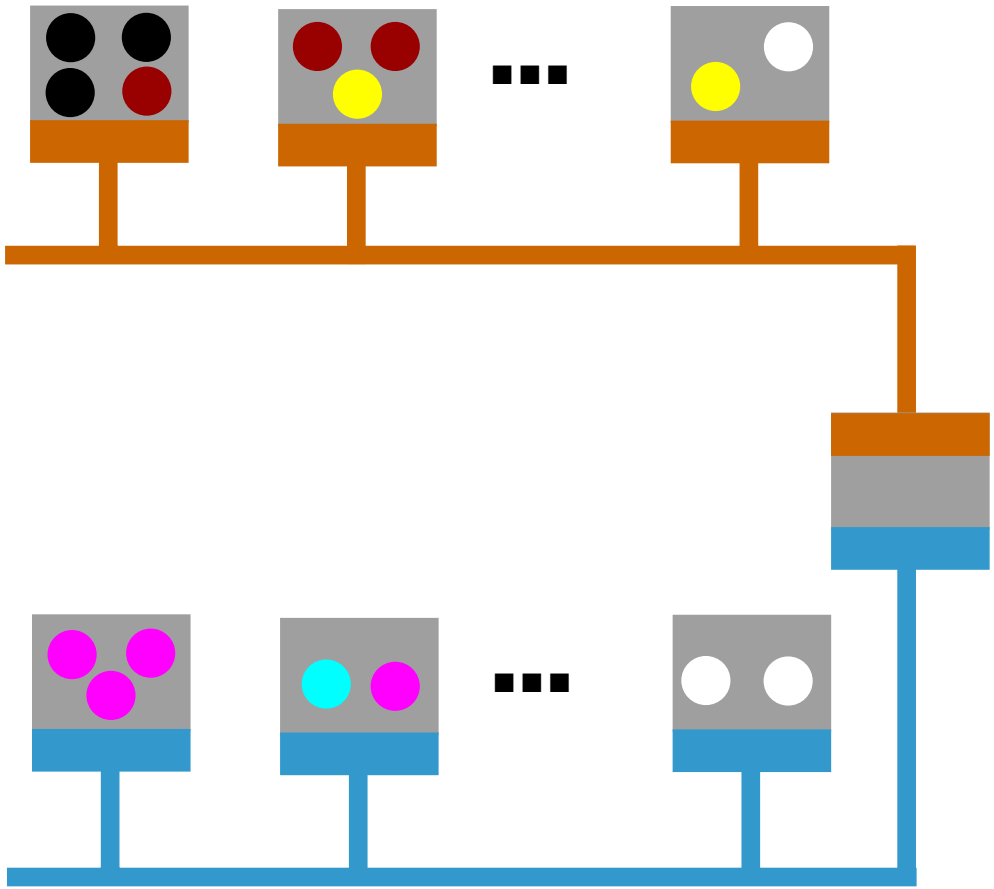
One node ⟷ Several functions

Several nodes ⟷ One function

Flexibility!

- Reduce cost
- Improve resource usage
- Function close to sensor

What Comes



One node ⟷ Several functions

Several nodes ⟷ One function

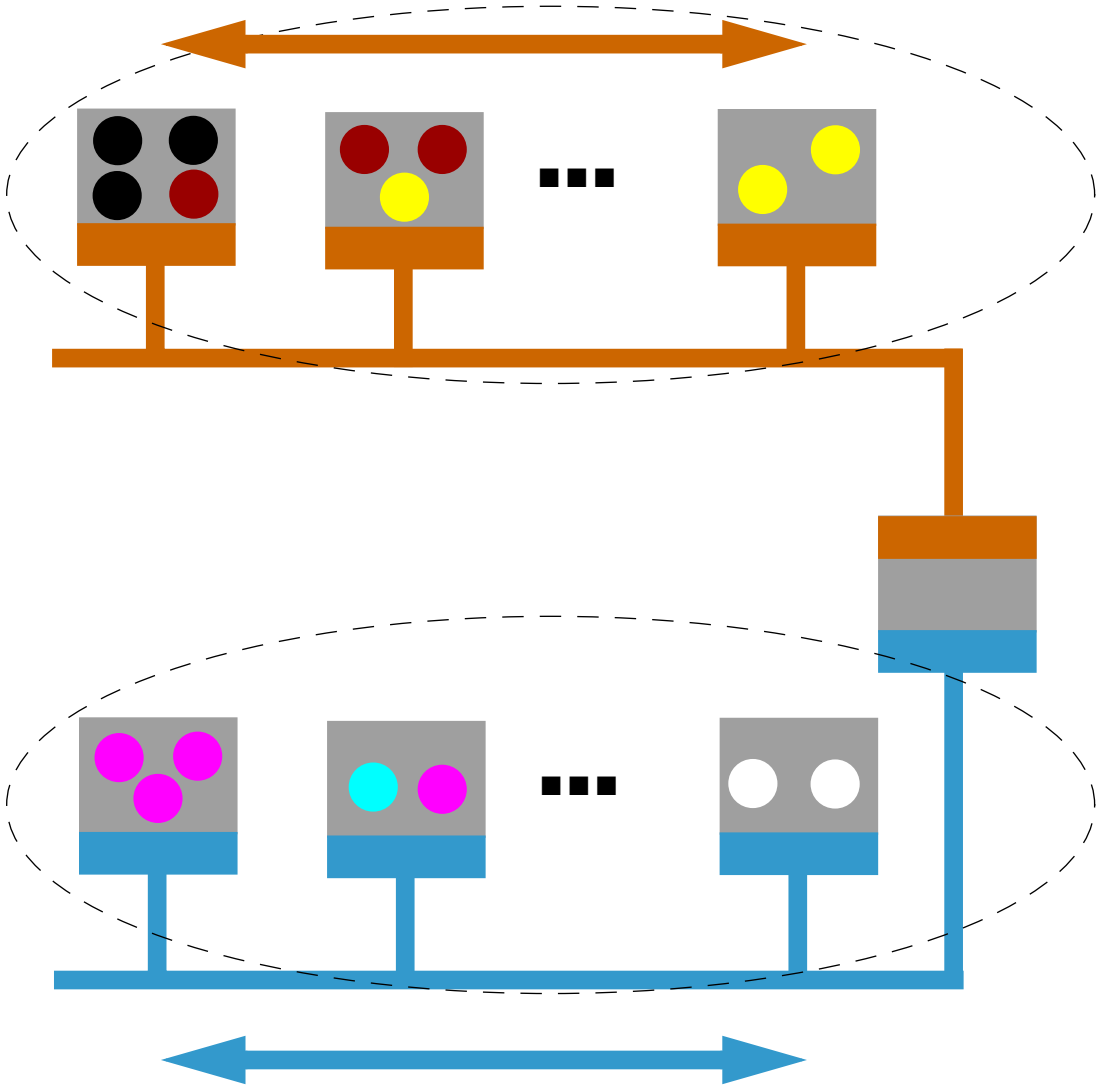
Flexibility!

- Reduce cost
- Improve resource usage
- Function close to sensor

☞ Needed:

- Middleware software
- New analysis
- New system optimization

Multi-cluster Heterogeneous Distributed Architecture



Time triggered cluster:

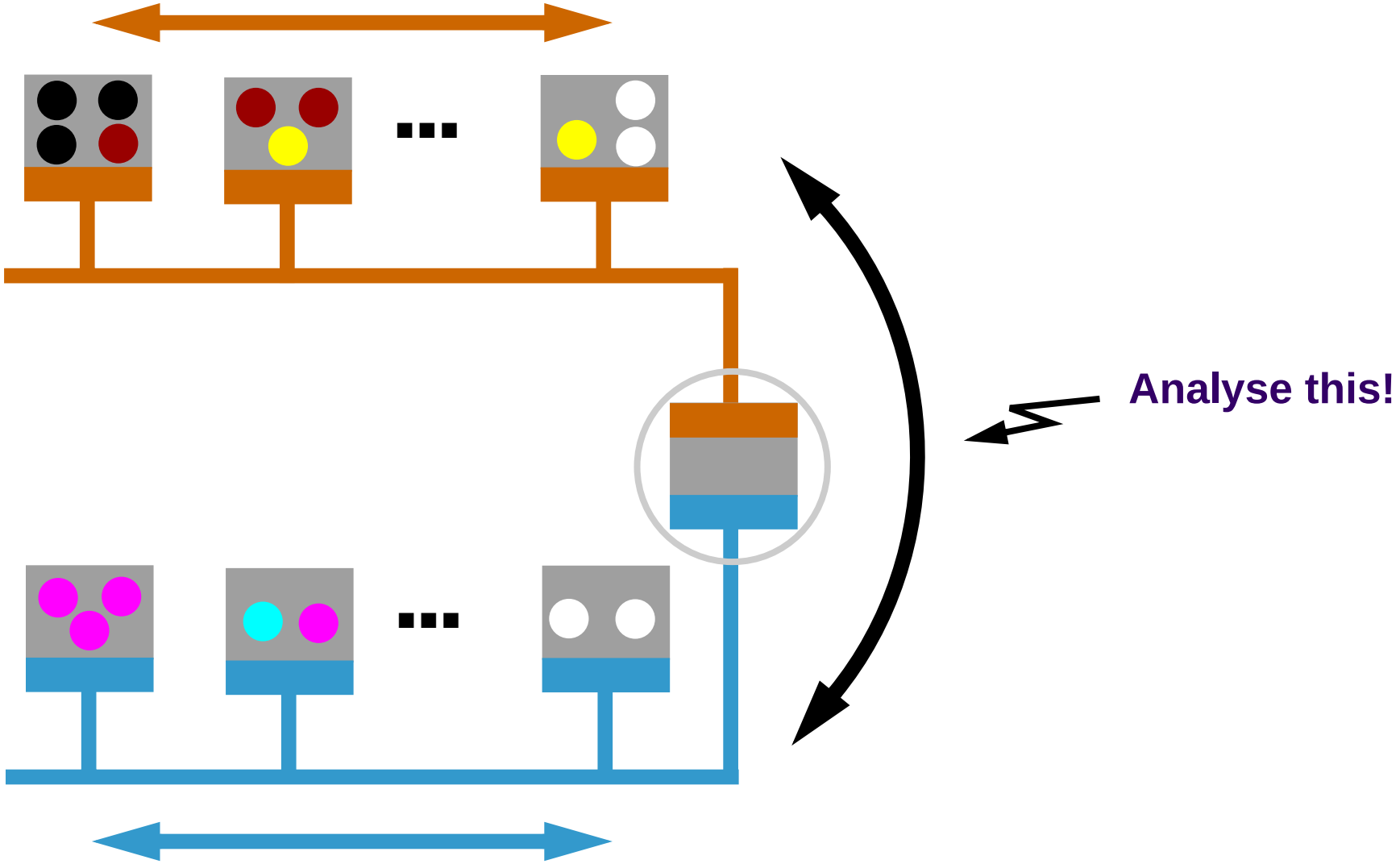
- TT tasks
- Static communication

That we have seen.

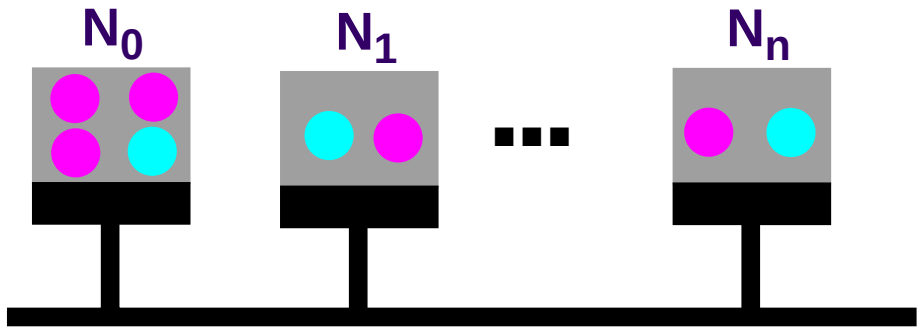
Event triggered cluster:

- ET tasks
- Dynamic communication

Multi-cluster Heterogeneous Distributed Architecture

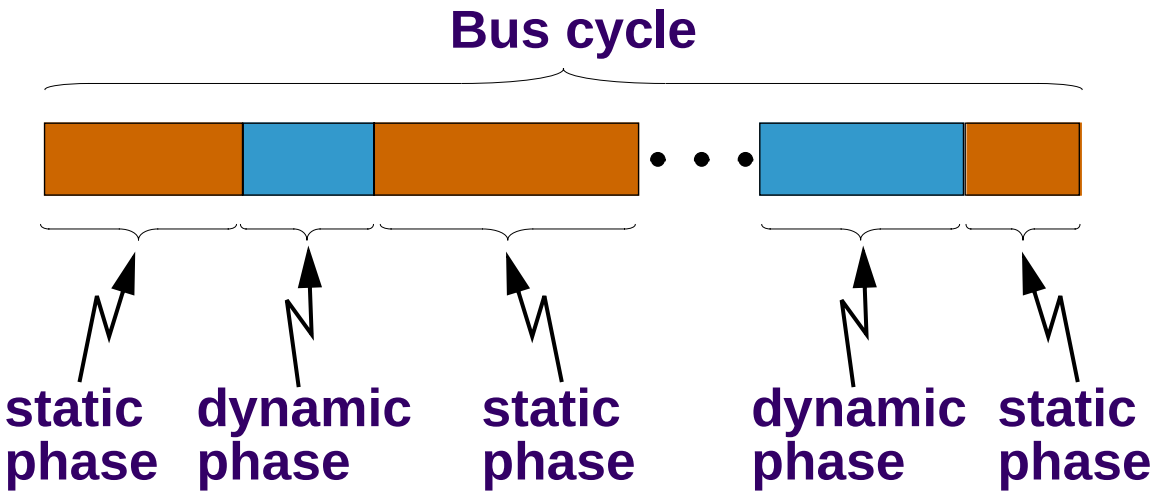


Single-cluster Heterogeneous Distributed Architecture

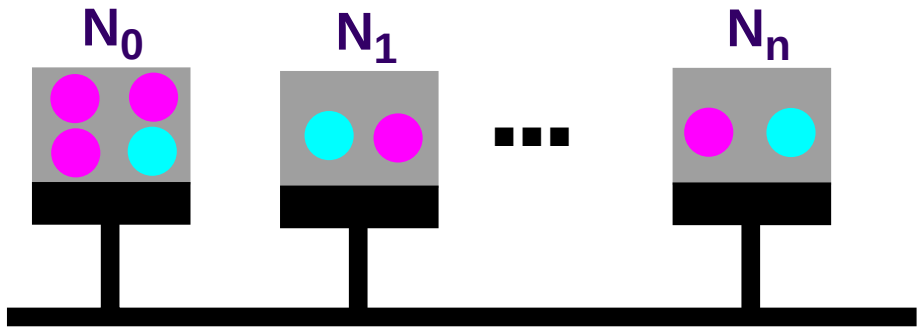


ET tasks

TT tasks

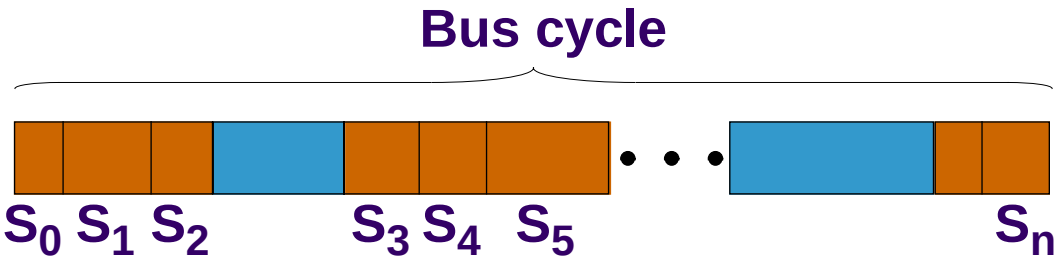


Single-cluster Heterogeneous Distributed Architecture

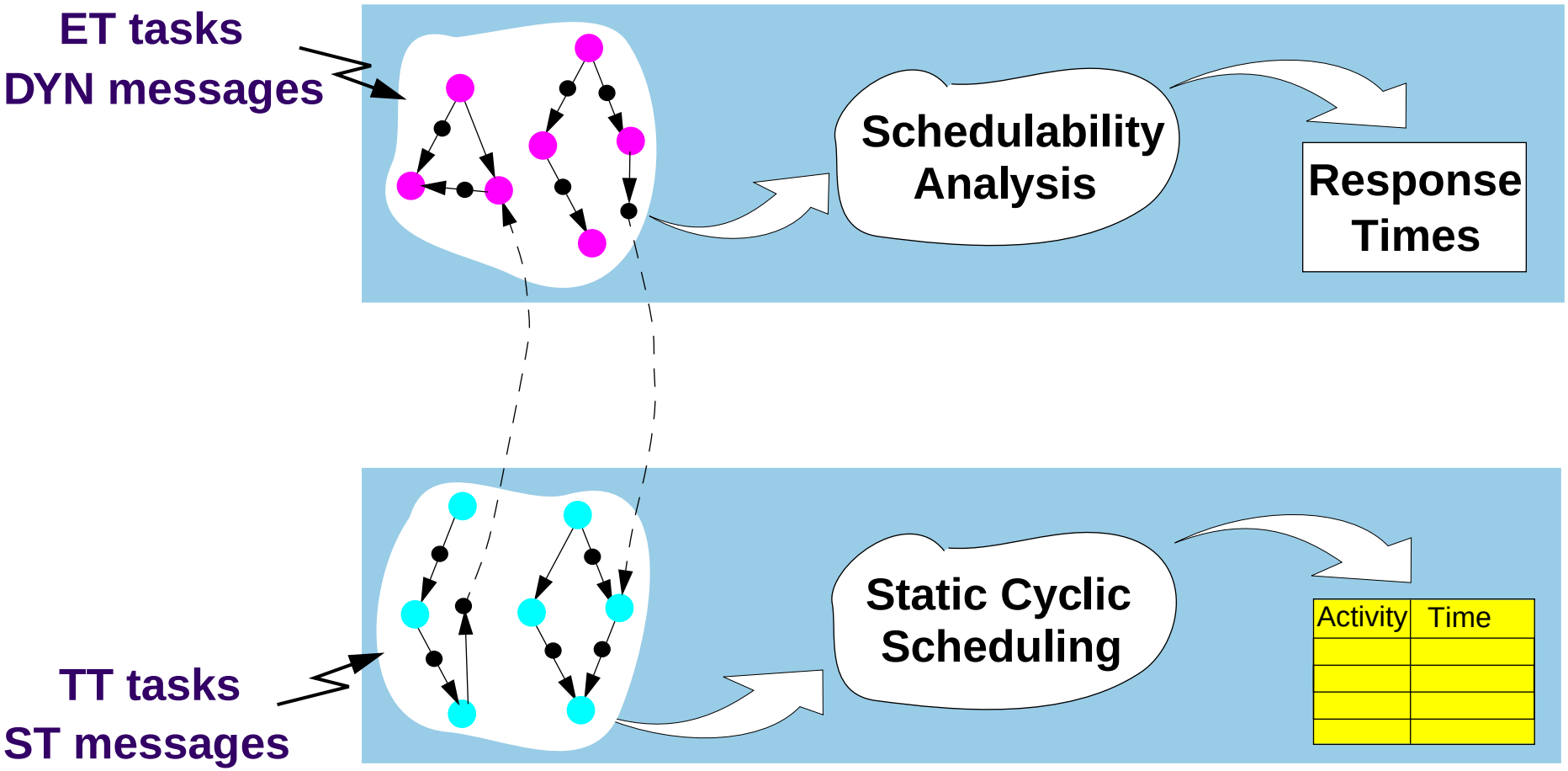


ET tasks

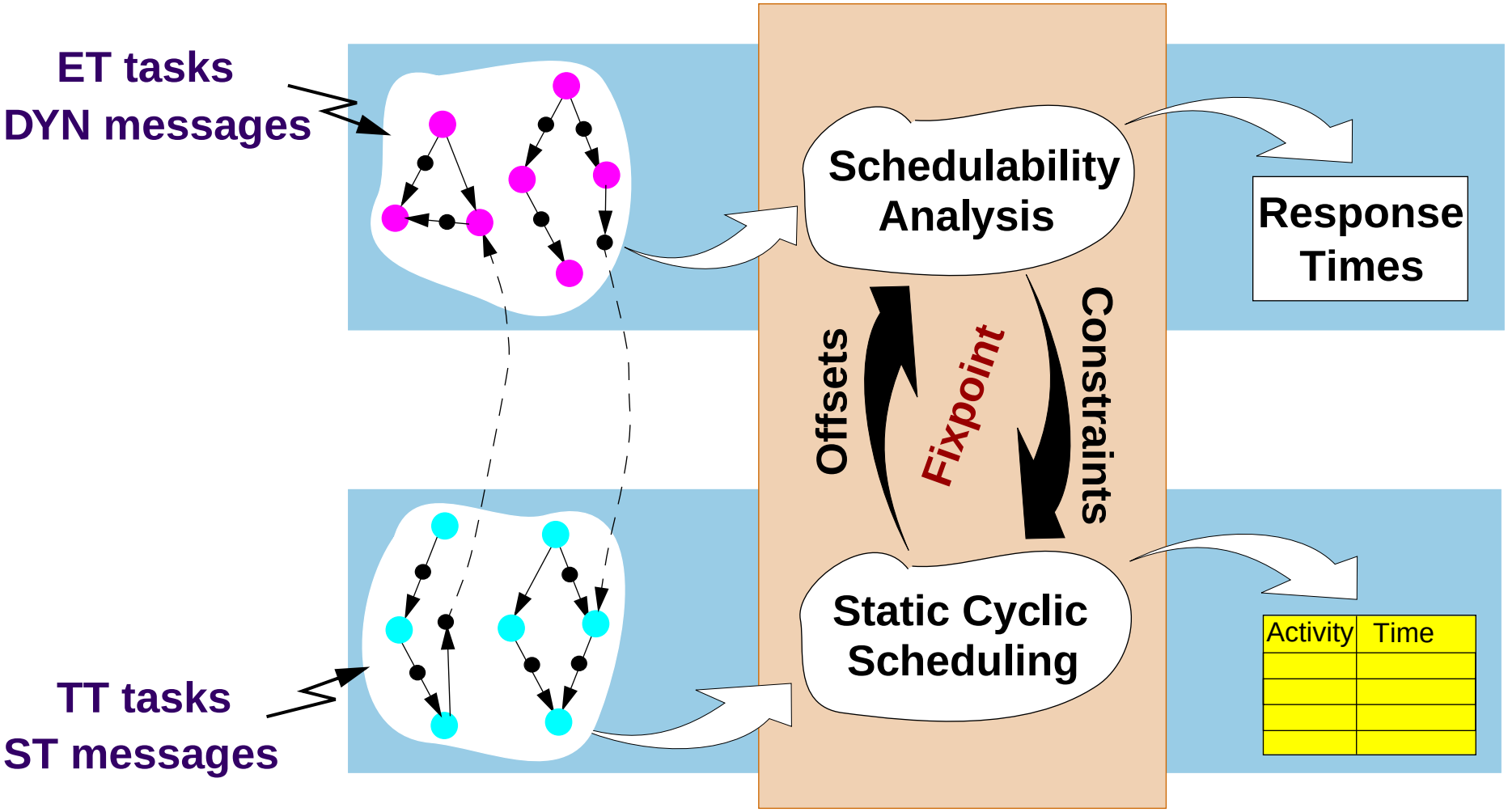
TT tasks



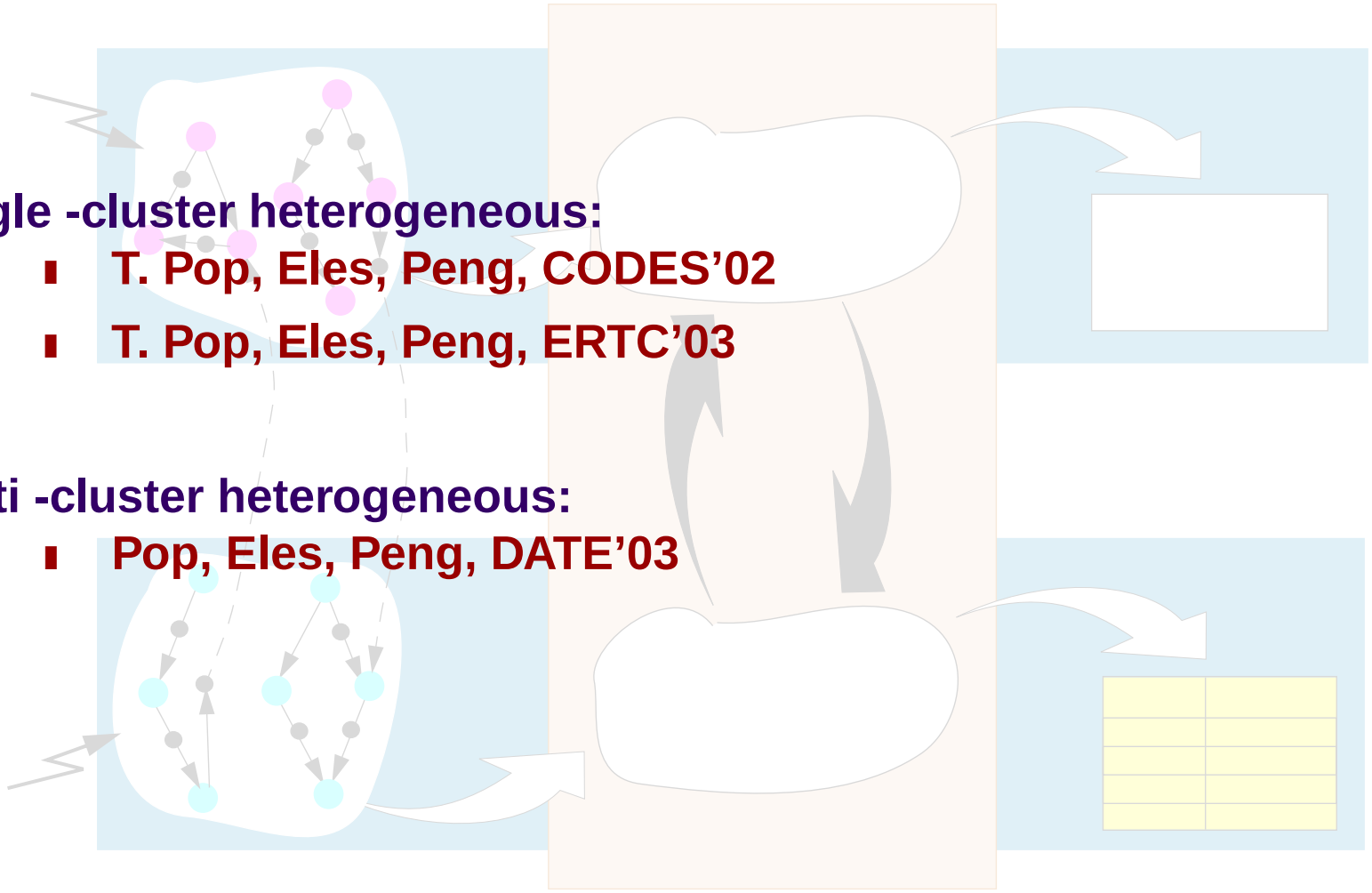
Analysis


ET tasks
DYN messages
Schedulability Analysis
Response Times
TT tasks
ST messages
Static Cyclic Scheduling
Activity
Time

Analysis


ET tasks
DYN messages
Schedulability Analysis
Response Times
Offsets
Fixpoint
Constraints
TT tasks
ST messages
Static Cyclic Scheduling
Activity
Time

Analysis



Single -cluster heterogeneous:

- T. Pop, Eles, Peng, CODES'02
- T. Pop, Eles, Peng, ERTC'03

Multi -cluster heterogeneous:

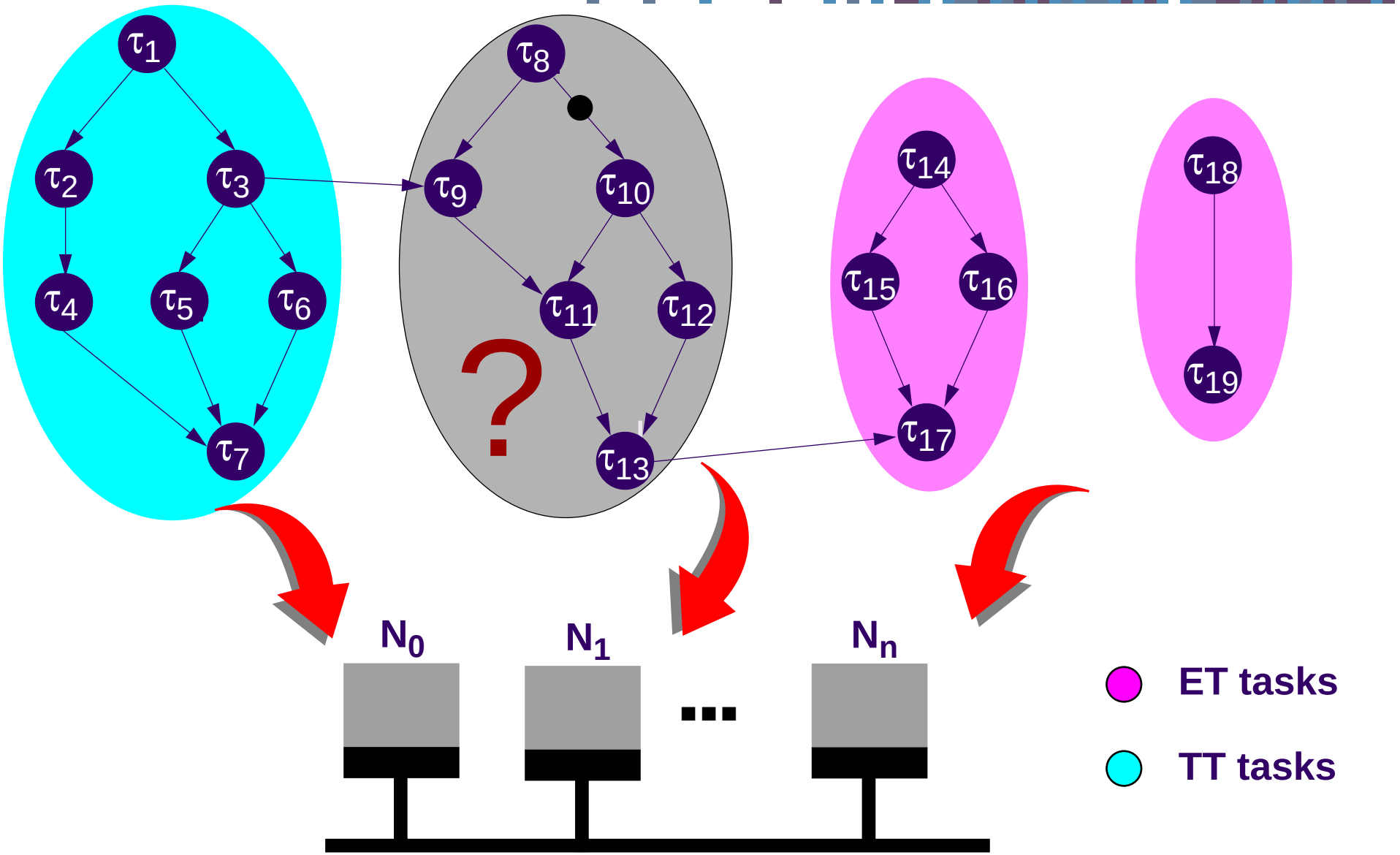
- Pop, Eles, Peng, DATE'03

Optimization

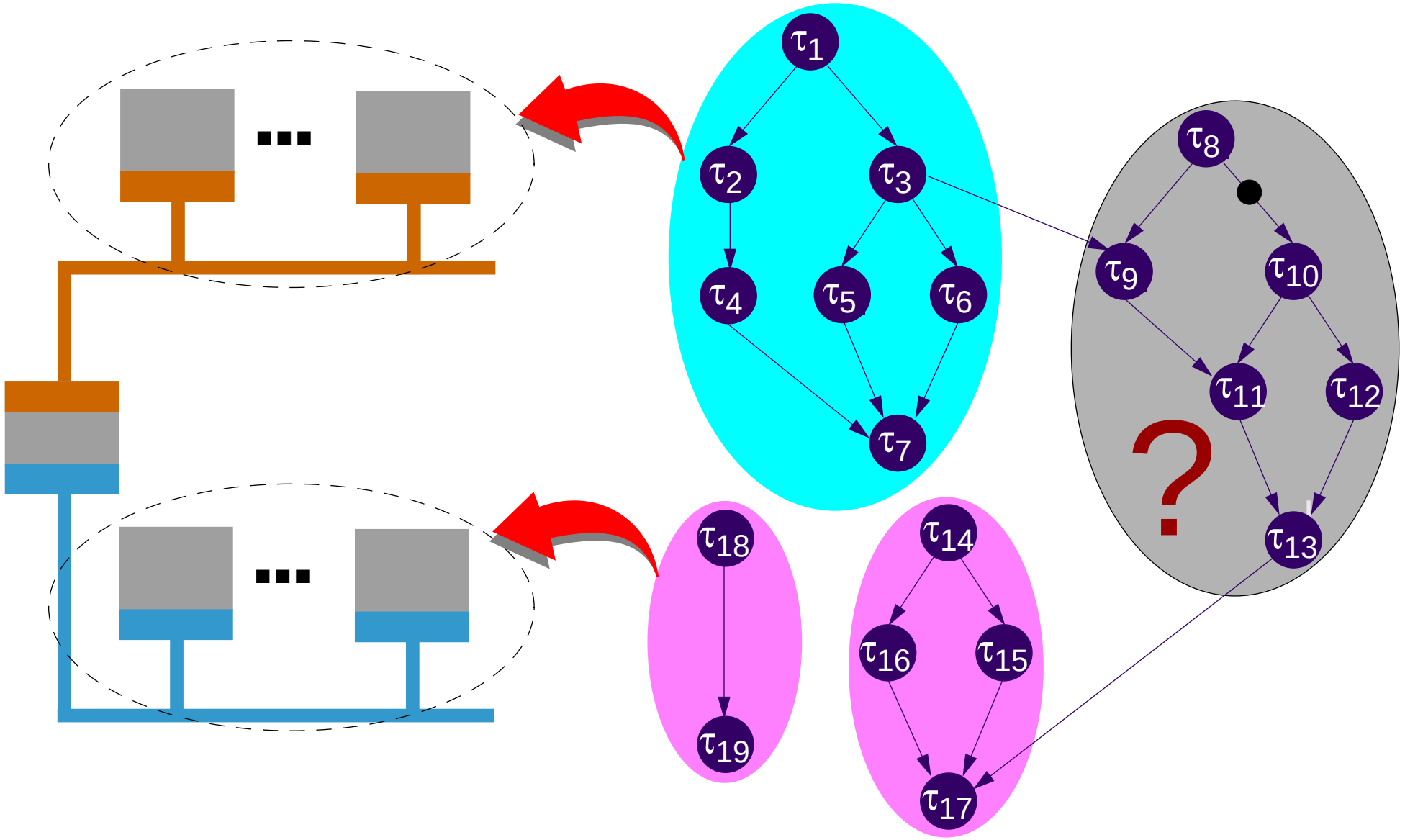
- Once the analysis approach is in place, several new, specific optimization tasks can be performed:

  - Task partitioning into TT, ET
  - Cluster mapping
  - Bus access optimization (static, dynamic phases)
  - Buffer minimisation

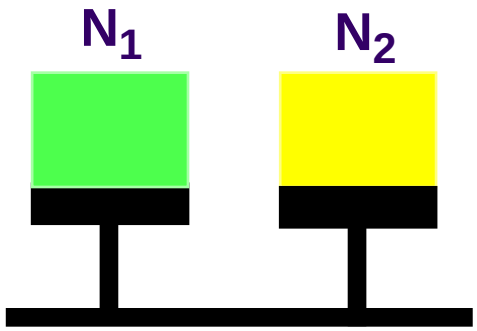
Mapping & Task Partitioning

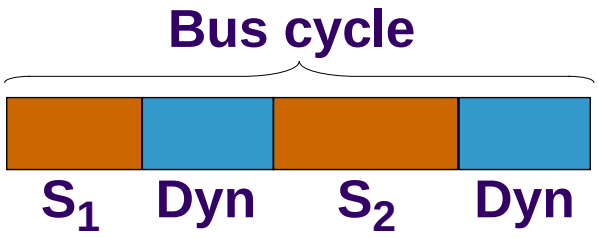


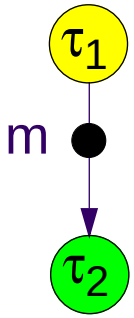
Mapping & Task Partitioning



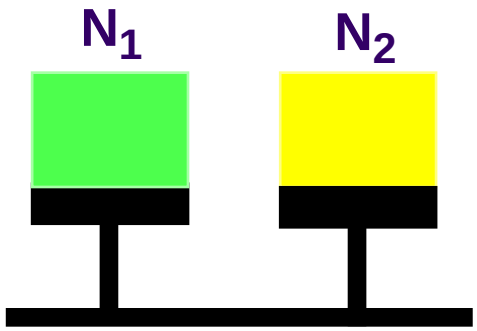
Bus Access Optimization

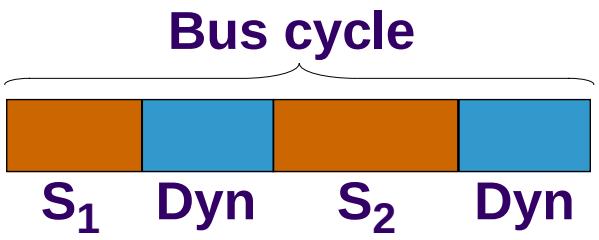


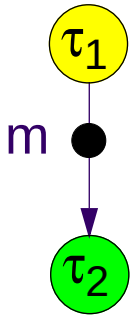


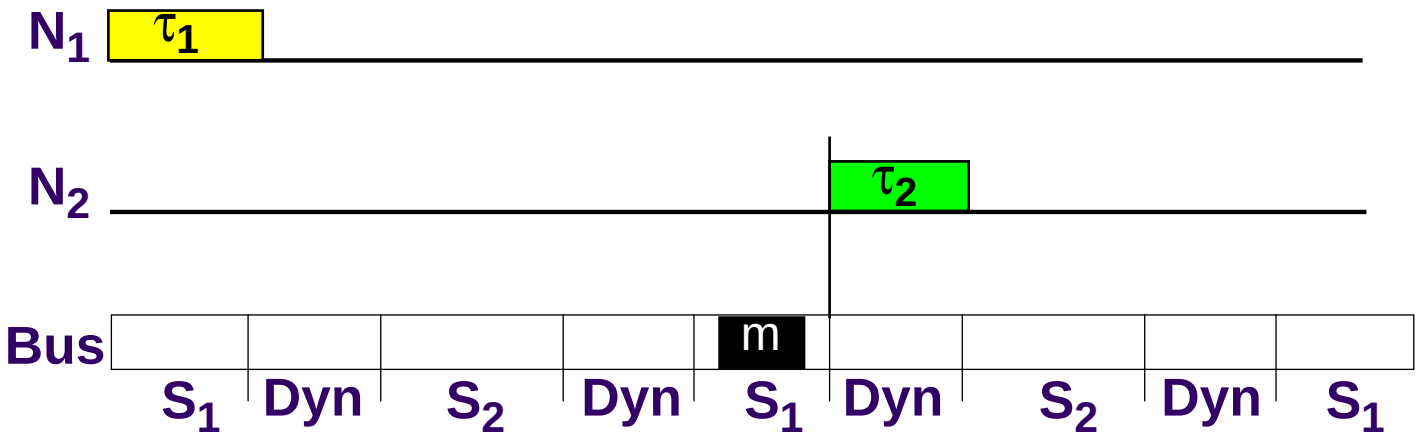


Bus Access Optimization

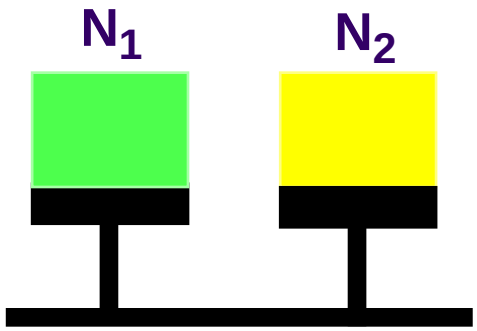


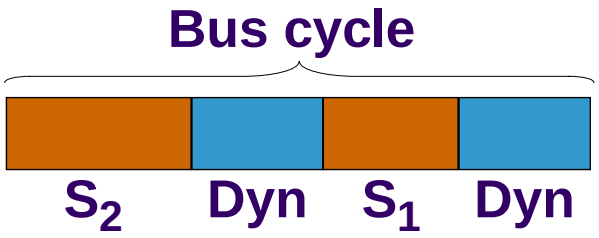


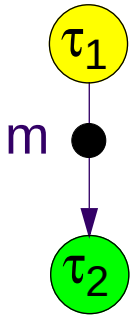


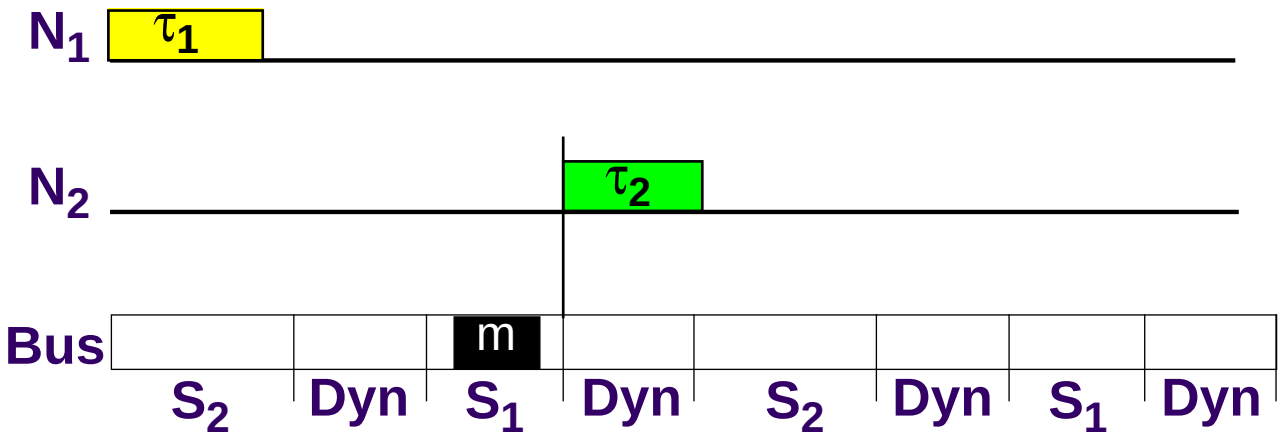


Bus Access Optimization

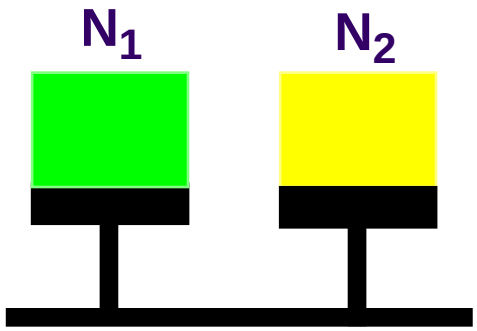


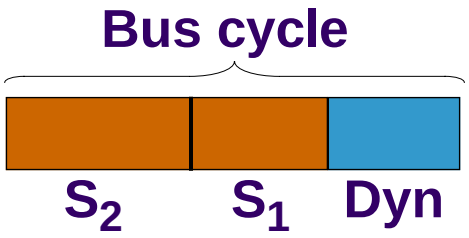


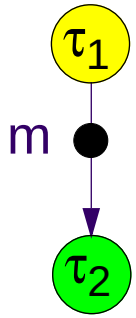


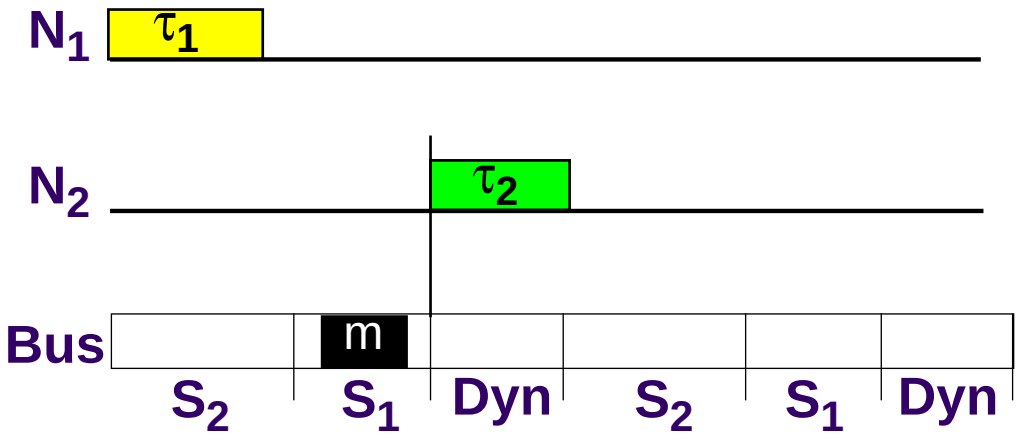


Bus Access Optimization

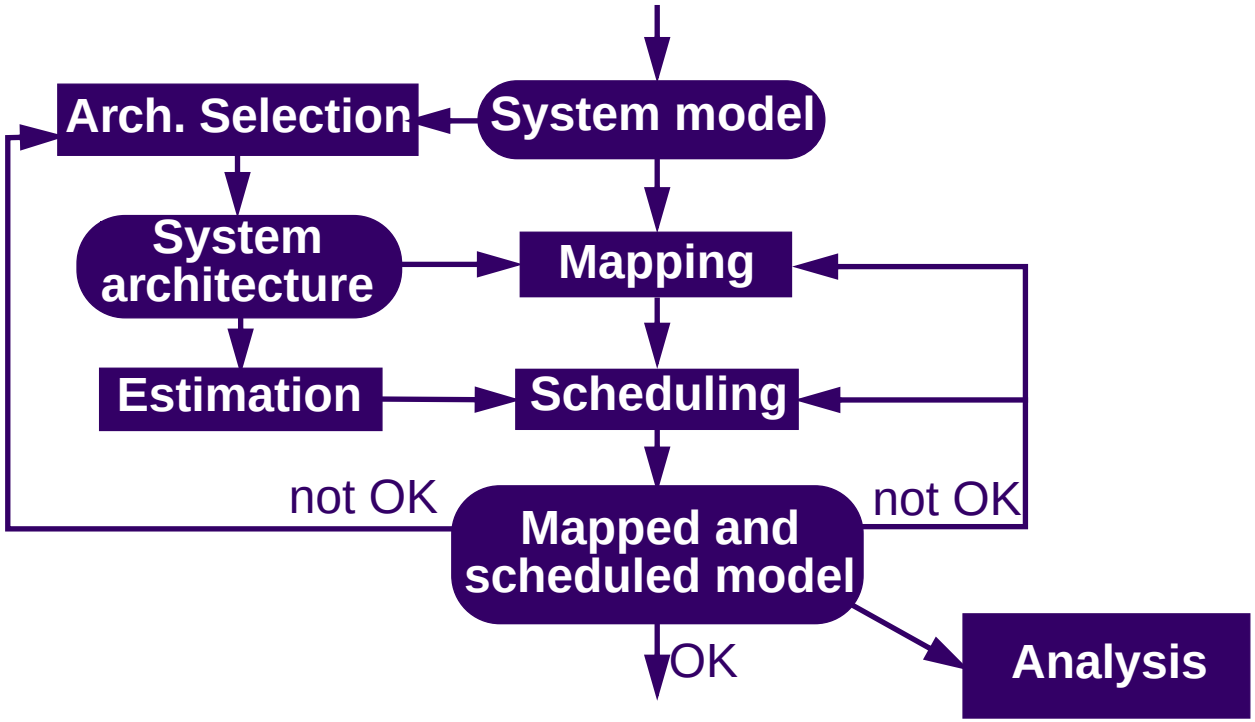








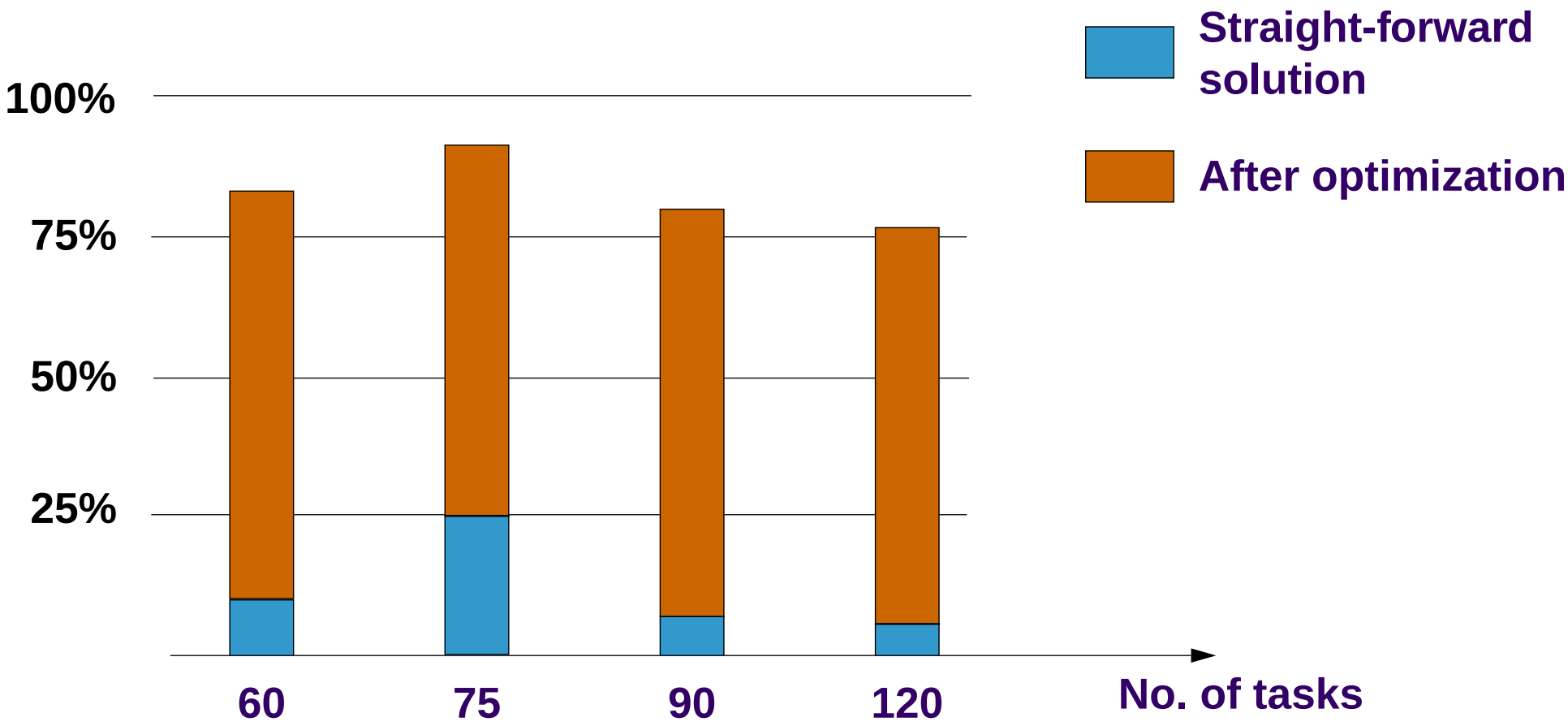
Optimization Loop



Multi-cluster heterogeneous:

- **Pop, Eles, Peng, DATE'03**

Improved Schedulability by Optimization



- 6 nodes (single-cluster)
- 60% average processor utilisation

Outline

- Embedded Real-Time System
- System-level Design Flow
- Distributed Embedded Real-Time Systems
  - Application Model
  - Heterogeneous Systems
  - Time/Event Triggered Tasks
  - Static/Dynamic Communication
  - Analysis&Optimization
- Single/Multi-cluster Heterogeneous Distributed Architectures
  - Analysis&Optimization
- **Incremental Design Process**

Incremental Design Process

- **In practice we almost never start from scratch!**

- **Start from an already existing system running certain applications.**

- **The design problem:**
  - **Implement new functionality or/and upgrades to the existing ones (without adding resources, if possible).**

Incremental Design Process

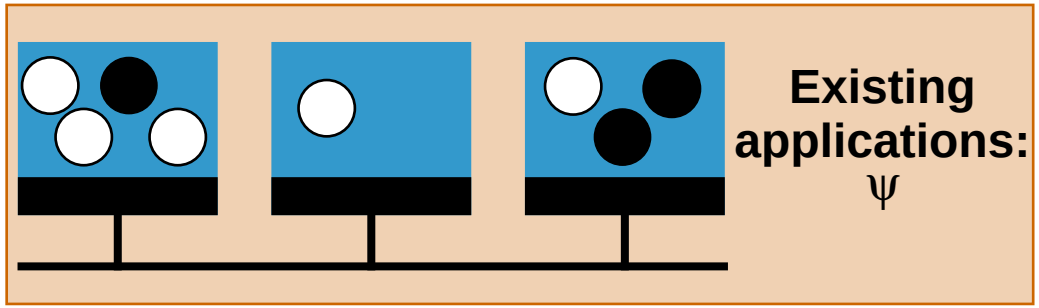
- In practice we almost never start from scratch!

- Start from an already existing system running a certain application

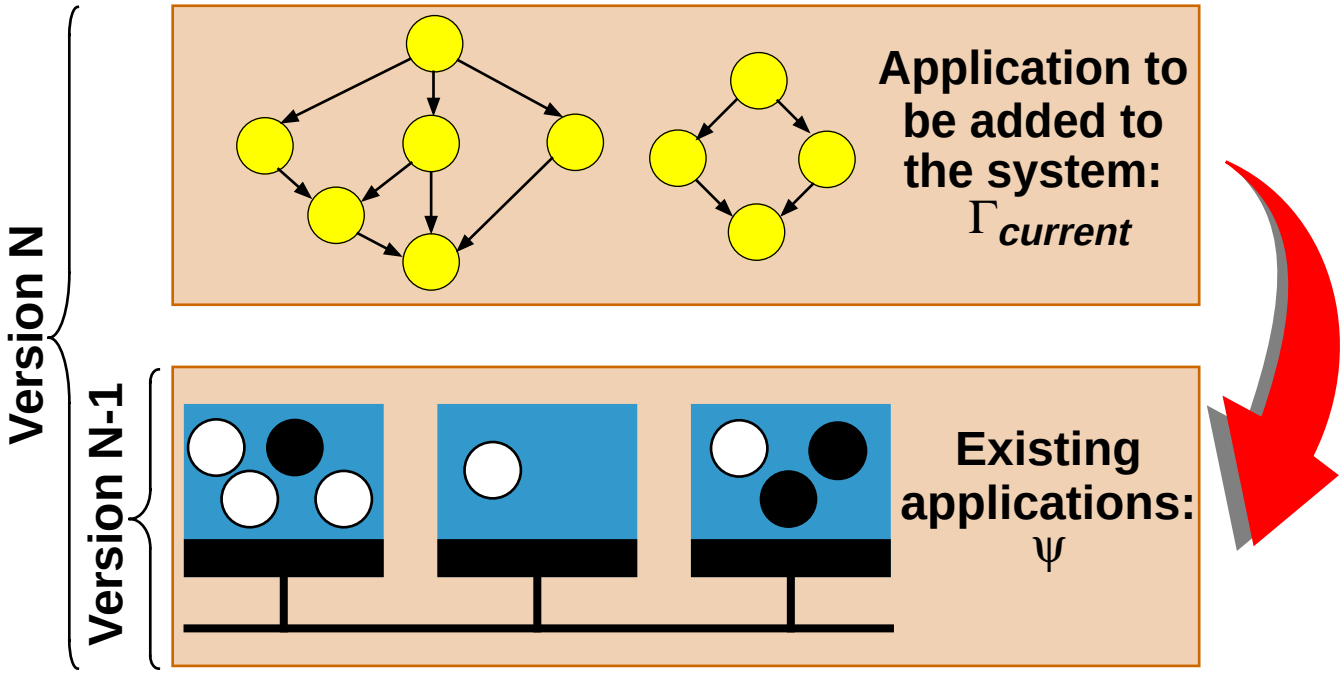
- The design problem:
  - Implement new functionality or/and upgrades to the existing ones (without adding resources, if possible).

- The same problems have to be solved as discussed before and in addition:
  - Produce as few as possible modifications to the running applications.
  - The resulting system should be such that, later, it is easy to add new functionality.

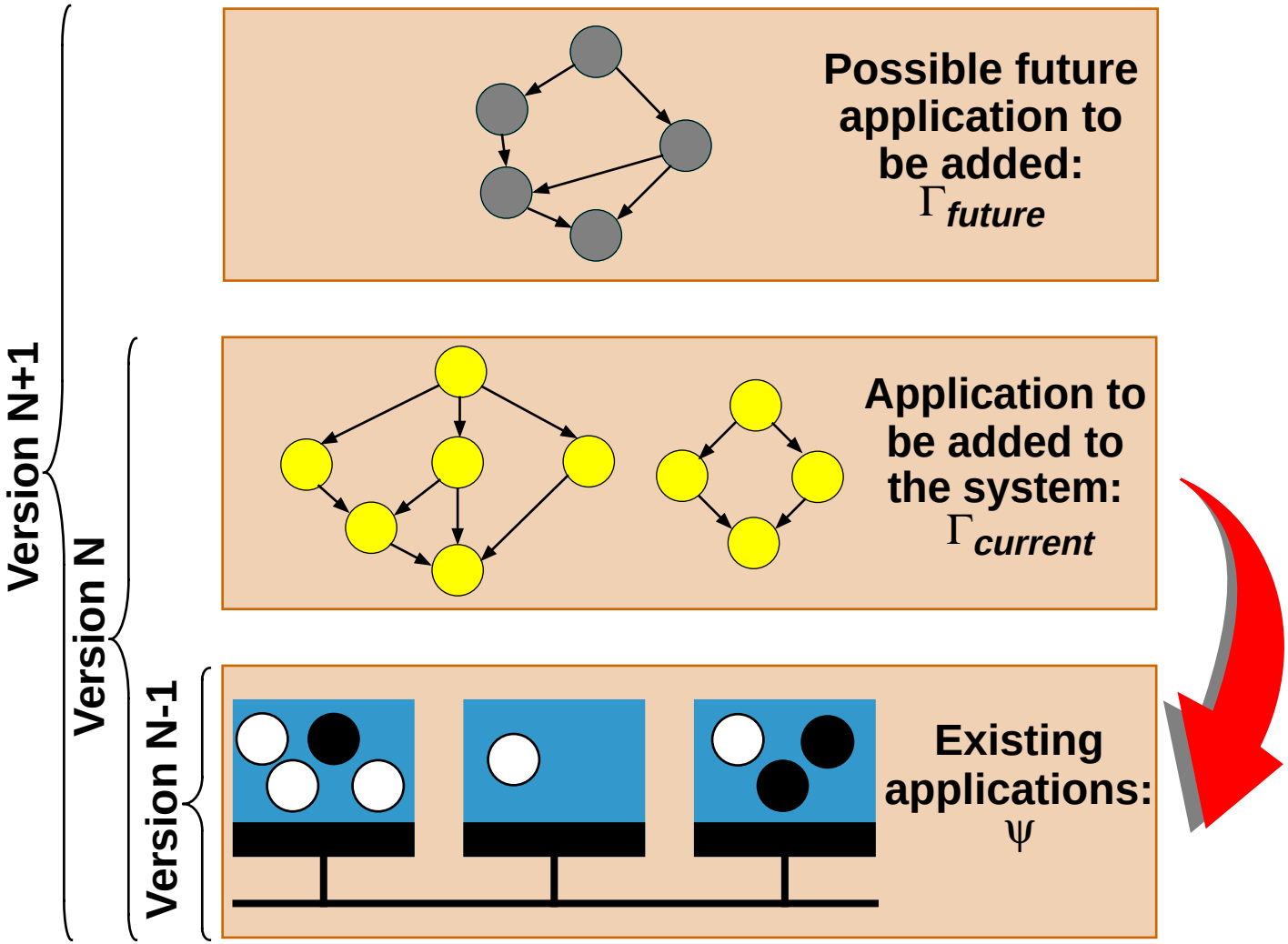
Incremental Design Process



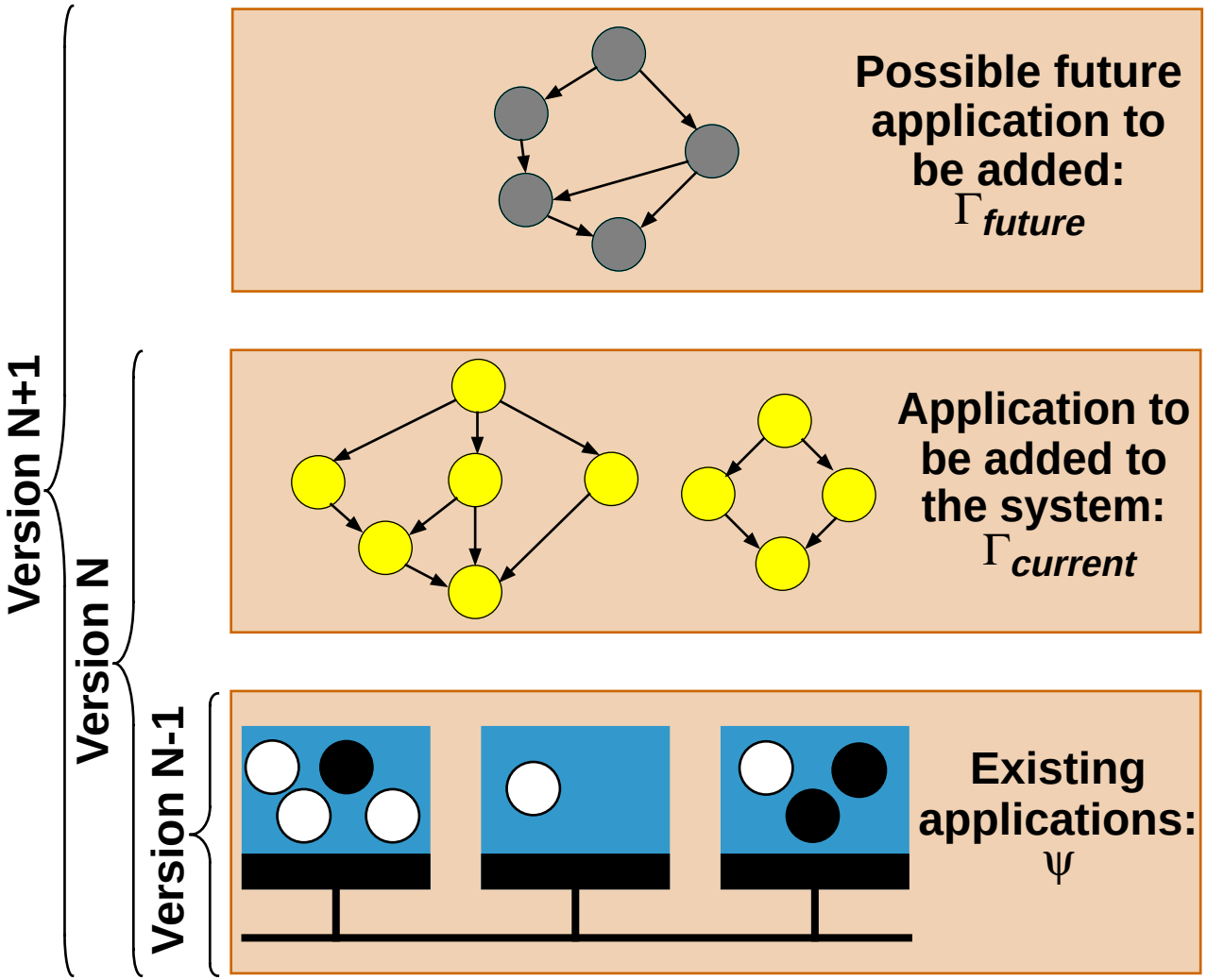
Incremental Design Process



Incremental Design Process



Incremental Design Process



Implement $\Gamma_{current}$ so that:

1. constraints on $\Gamma_{current}$ satisfied
2. modifications of ψ minimized
3. possible to implement Γ_{future}

Incremental Design Process

Pop, Eles, T. Pop, Peng, CODES '01

Pop, Eles, T. Pop, Peng, DAC '01

☞ **The goal:**

- **Minimise modification related costs:**
  - **Modify as few as possible from ψ.**
  - **Allocate resources so that it will be easy to implement Γ_{future}.**

- **Resources that have to be allocated in the right way:**
  - **Processor utilisation**
  - **Bus bandwidth**
  - **Time slots on processors and buses**
  - **Memory**

The Future Application

What do we know?

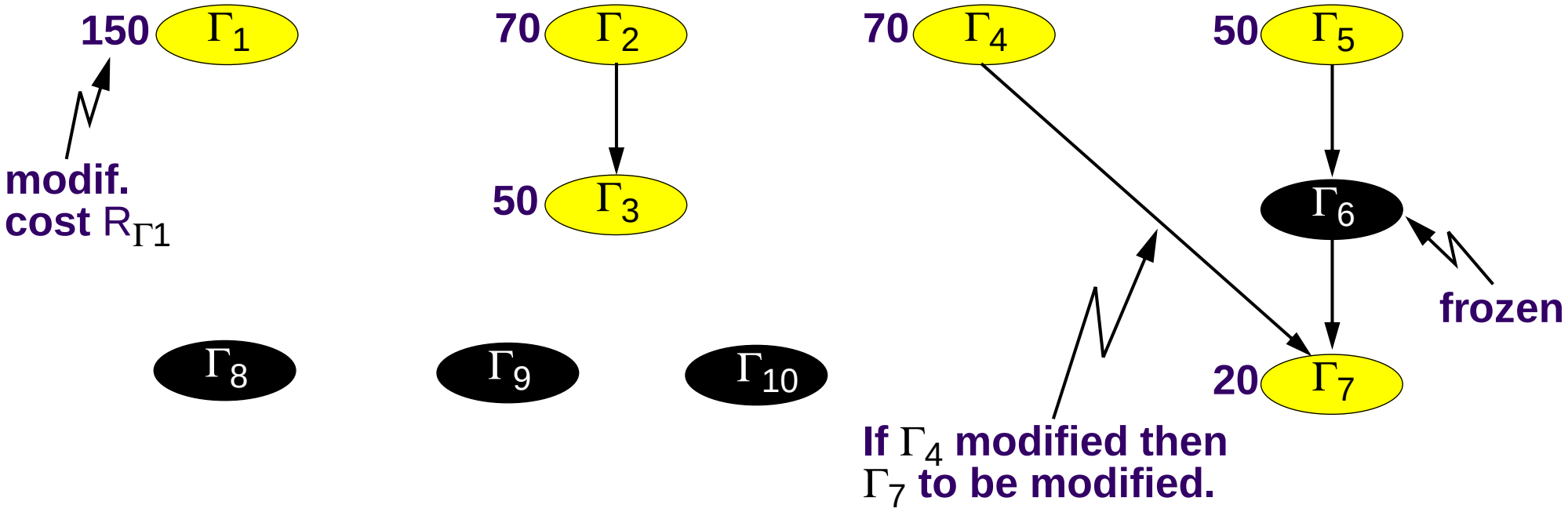
- About the future application:
  - Probability distributions regarding the need for
    - Processor
    - Bus bandwidth
    - Memory

The Existing Applications

What do we know?

- About the existing applications:
  - The application graph captures
    - modification costs of applications (e.g. COCOMO, Boehm et al, 2000);
    - dependencies: modifying an application can trigger the need to modify other applications.

The Existing Applications



Total modification cost of a subset of applications $\Omega \in \psi$.

$$R(\Omega) = \sum_{\Gamma_i \in \Omega} R_{\Gamma_i}$$

Conclusions

- System-level design; the early design steps.
- Heterogeneous distributed real-time systems:

  How to guarantee timing constraints?
- It’s not sufficient to analyse! Help the designer to efficiently implement.

  Both functionality and communication!
- You don’t start from scratch and there always comes another version.