Embedded Systems Design: Concepts and Methods

Todor Stefanov

Leiden Embedded Research Center,
Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands

Outline

- Trends in Embedded Systems Design
- Design Productivity Gap and Design Problem
- Design Methodologies for Embedded MPSoCs
  - Y-chart of Gajski
  - Y-chart of Kienhuis

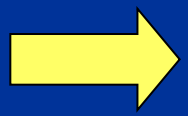
- The DAEDALUS design flow for MPSoCs
  - Introduction and Motivation
  - DAEDALUS flow overview

Trends in Embedded Systems Design (1)

- Modern embedded systems must
  - support multiple applications with limited energy and resource budgets
  - often provide real-time performance/guarantees
- These systems increasingly have heterogeneous system architectures, integrating
  - Dedicated hardware
    - High performance and low power
  - Embedded processor cores
    - High flexibility
  - Reconfigurable components (e.g. FPGAs)
    - Good balance between performance/power/flexibility

Trends in Embedded Systems Design (2)

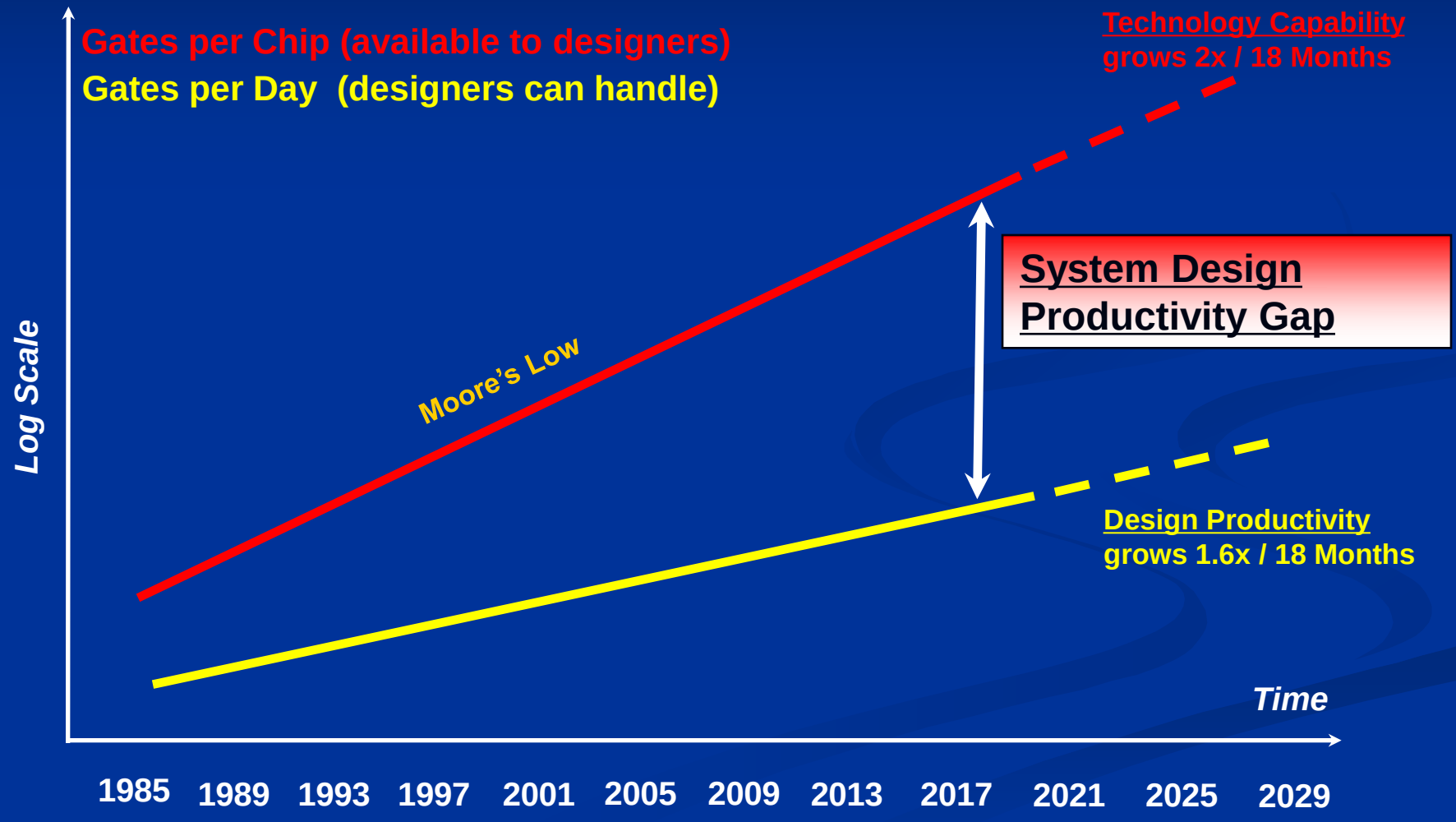
- Silicon budgets are increasing (Moore's Law)
  - High Integration of functions: Systems-on-Chip
  - (Massively) Parallel Systems on a single chip!
- Life cycle of systems is decreasing (e.g., look at cell phones)
  - Short time to market



The design of modern embedded systems becomes increasingly complex

Design Productivity Gap

The *Capability of IC Technology* grows faster than the *Design Productivity*!



The Design Problem

How to design complex embedded systems faster?

The challenge is:

- To increase the system design productivity,
- without sacrificing the quality of the system under design,
- in a *complex design space* with many tradeoffs and *conflicting design objectives*:
  - Low Cost (e.g., small silicon area)
  - High Performance
  - Low Power Consumption
  - High Flexibility

In this course we will study ***only*** Methods, Techniques, and Tools that help solving the Design Problem for Embedded Multi-Processor Systems on Chip (MPSoC)

Evolution of Design Methodologies

Design Methodologies have been drastically changing with the increase in system complexity

Historically, 3 generic evolutionary design methodologies

- Capture-and-Simulate (1960s to 1980s)
  - Designers do complete design manually, no automation
  - Designers validate design through simulation at the end of the design
- Describe-and-Synthesize (early 1980s to late 1990s)
  - Designers describe just functionality, tools synthesize structure
  - Simulation before and after the synthesis
- Specify-Explore-Refine (early 2000 to present)
  - System design performed at several levels of abstraction
  - At each level of abstraction designers:
    - First, specify/model the system under design
    - Then, explore alternative design decisions
    - Finally, refine the model according to their decisions (i.e., put more details)
  - Refined model used as specification for the next lower level

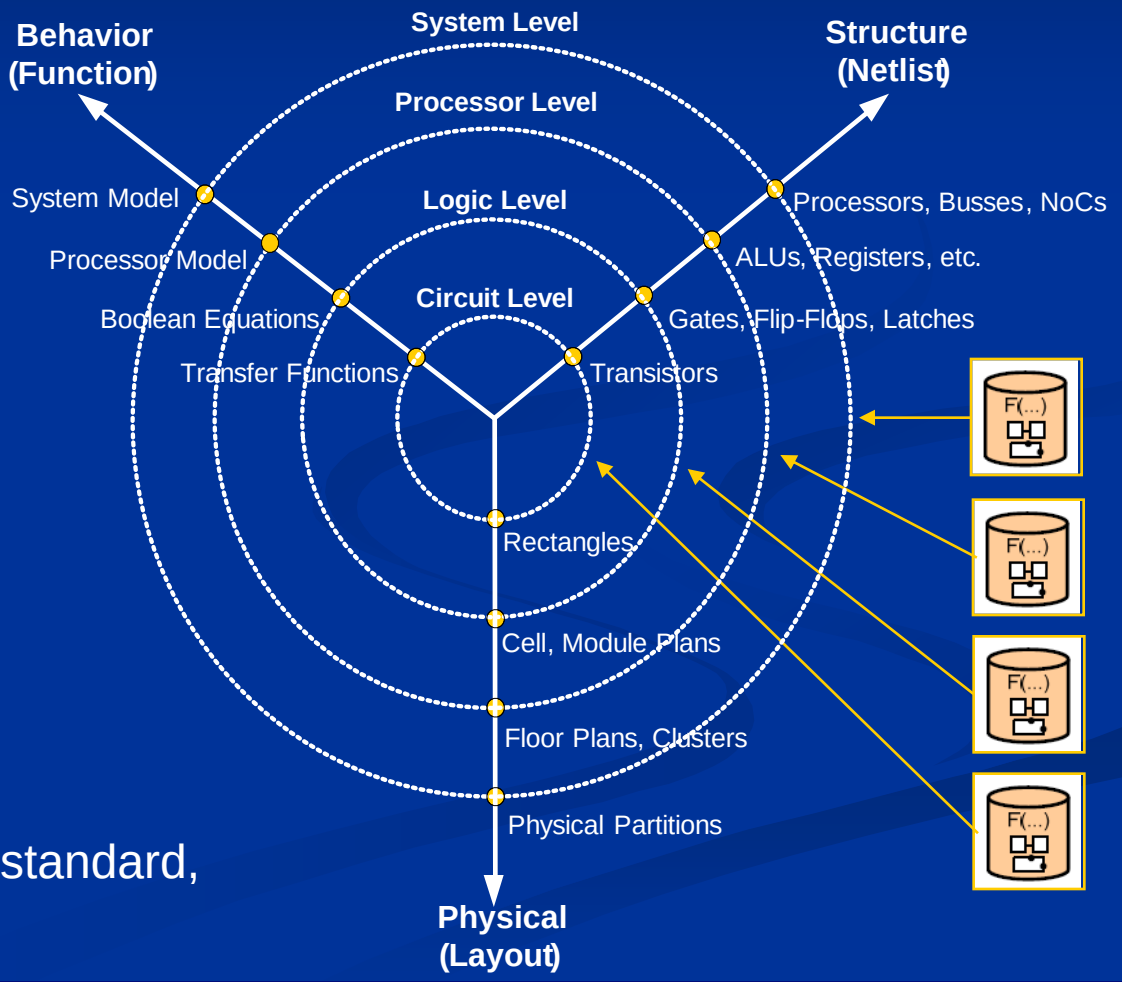
Specify-Explore-Refine Methodology: MPSoC design aspects

- This methodology can be defined in two aspects:
  - ***Synthesis***-oriented design aspect
  - ***Quality Assessment***-oriented design aspect
- For each aspect, there is corresponding so called **Y-chart** design methodology
  - Gajski et al. Y-chart [1983 and modified several times up to now]
    - covers mainly the synthesis aspect
  - Kienhuis et al. Y-chart [1997 and extended later]
    - covers mainly the quality assessment aspect

Y-chart Design Methodology [Gajski et al.] – synthesis aspect

Design methodology is a sequence of models, components and tools used to design the system. Every design has 3 views.

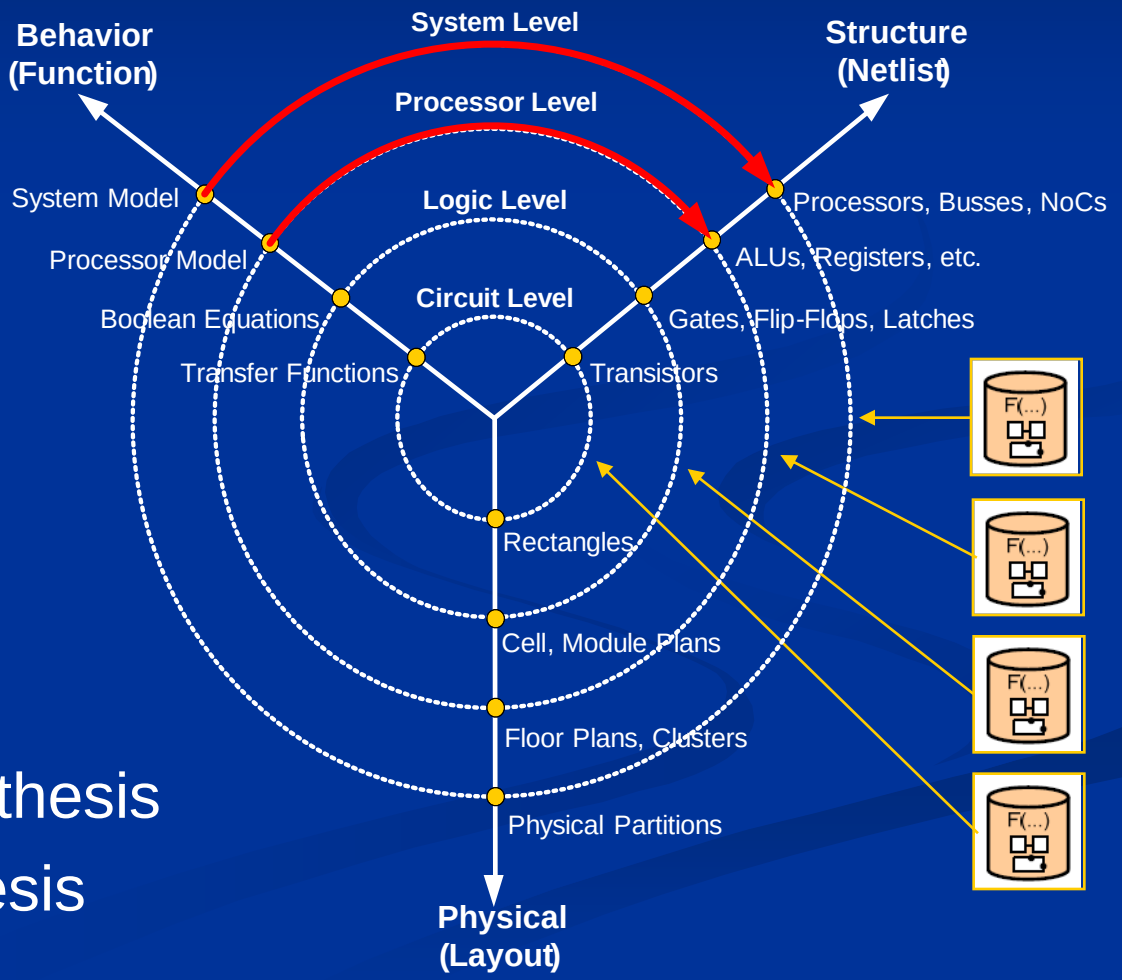
- Three design views
  - Behavior (specification)
  - Structure (block diagram)
  - Physical (floorplan)
- Four abstraction levels
  - Circuit level
  - Logic level
  - Processor (RTL) level
  - System level
- Four component libraries
  - Transistors
  - Logic (standard cells)
  - RTL (ALUs, RFs, …)
  - Processor/Communication (standard, custom)



Synthesis-based Design Methodology

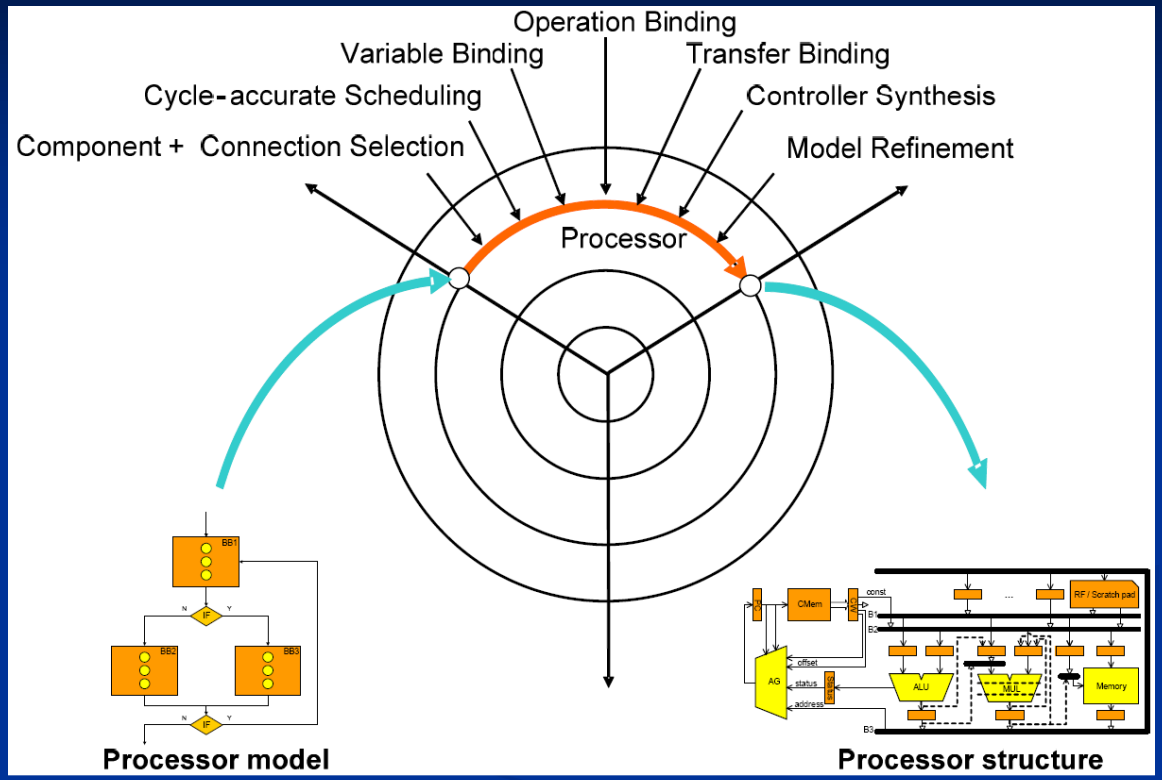
Synthesis is the process of generating the description of a system in terms of related components from a description of the expected system behavior.

- Synthesis can be performed at every level of abstraction
- Examples:
  - Processor Level Synthesis
  - System Level Synthesis



Processor Level Synthesis

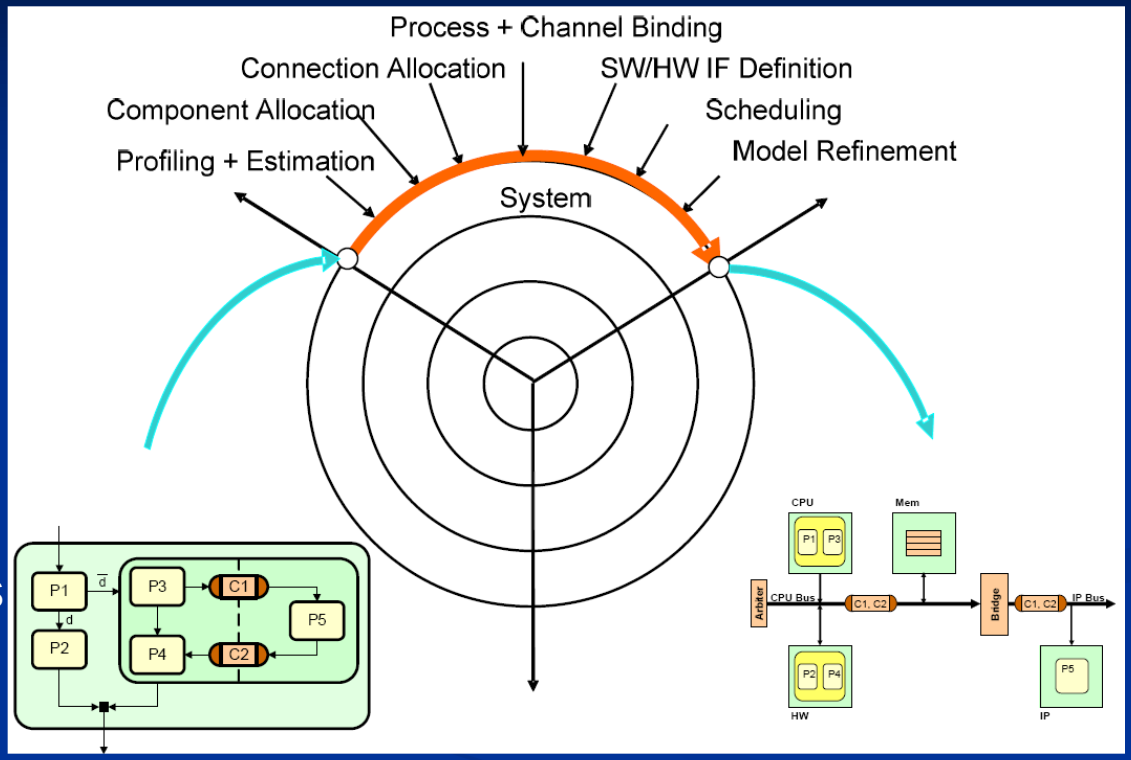
- Processor Model
  - FSM with Datapath
  - CDFG
  - Instruction Set Flow Chart
- Processor Structure
  - Datapath components
    - Storage (registers)
    - Functional units (ALUs, multipliers)
    - Connection (buses)
  - Controller component
    - Registers (PC, Status register, Control word or Instruction register)
    - Others (Control memory or Program memory)



- Synthesis consists of several tasks – see the figure

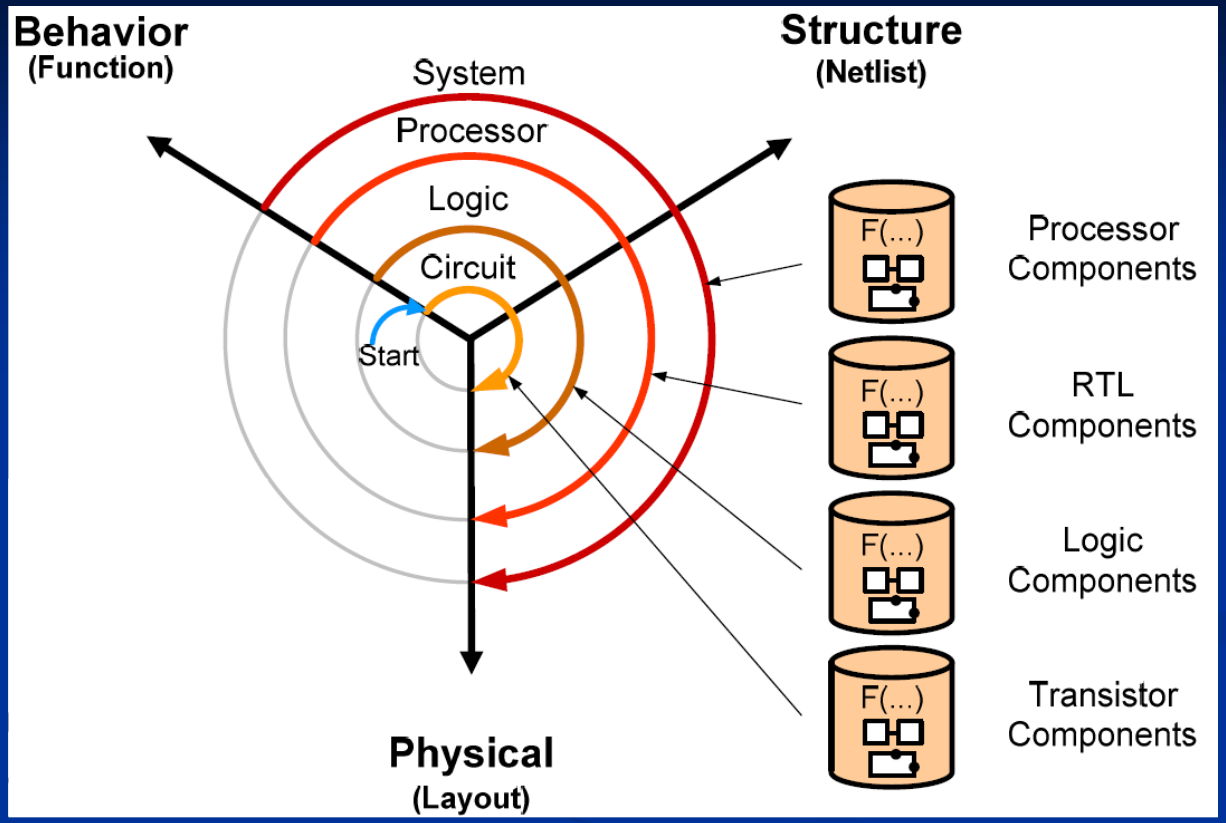
System Level Synthesis

- System Behavior Model
  - Use a MoC
  - Many MoCs exist
- System Structural Model
  - Set of computational components
    - Processors
    - IPs
    - Custom HW components
    - Memories
  - Set of communication components
    - Buses, bridges, arbiters
    - NoCs



- Synthesis consists of several tasks – see the figure

Bottom-up Design Methodology (1)

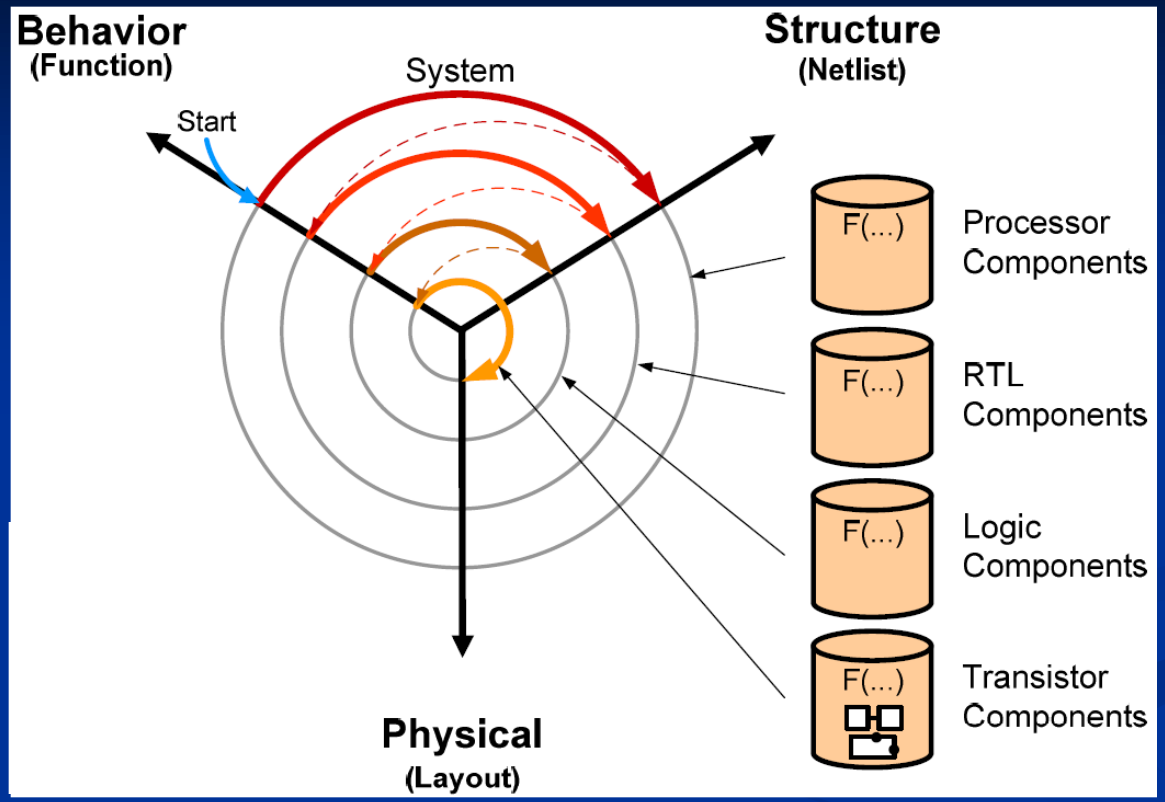


- Starts from bottom level
- Each level generates library for next higher level
  - Circuit level: use transistors to build Gates and Flip-Flops (FF) for Logic level
  - Logic level: use gates, FF to build RTL components for Processor level
  - Processor: build Processing and Communication components for system level
  - System Level: build Embedded System platforms for different applications
- Physical Design (floorplaning and layout) on each level!

Bottom-up Design Methodology (2)

- Pros
  - Abstraction levels clearly separated with its own library
  - Accurate metric estimation (e.g. performance, power)
    - Physical design with layout on each level
- Cons
  - An optimal library for each design is difficult to predict
    - All possible components with all possible parameters
    - All possible optimizations for all possible metrics
  - Library customization is outside the design group
  - Physical design is performed on every level

Top-down Design Methodology (1)

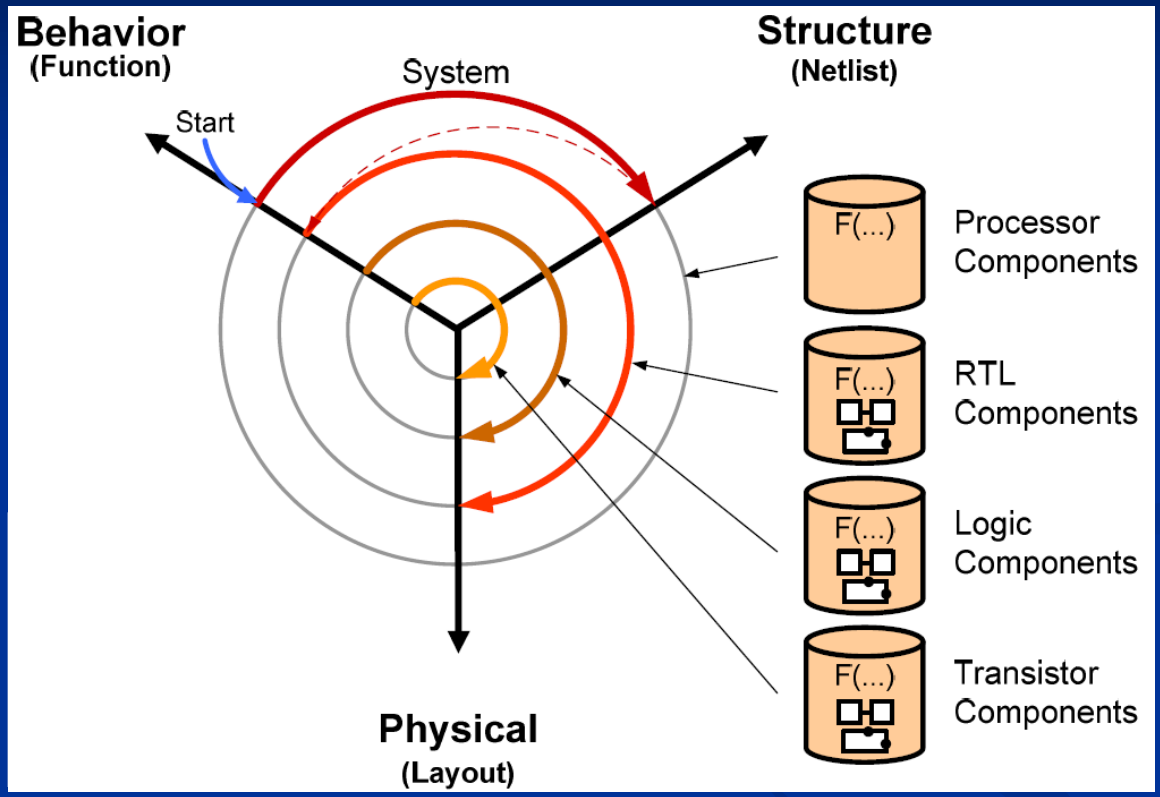


- Starts with the top level
- Functional description is converted into component netlist on each level
- Each component is functionally described and decomposed further on the next abstraction level
- Layout is given only for transistor components

Top-down Design Methodology (2)

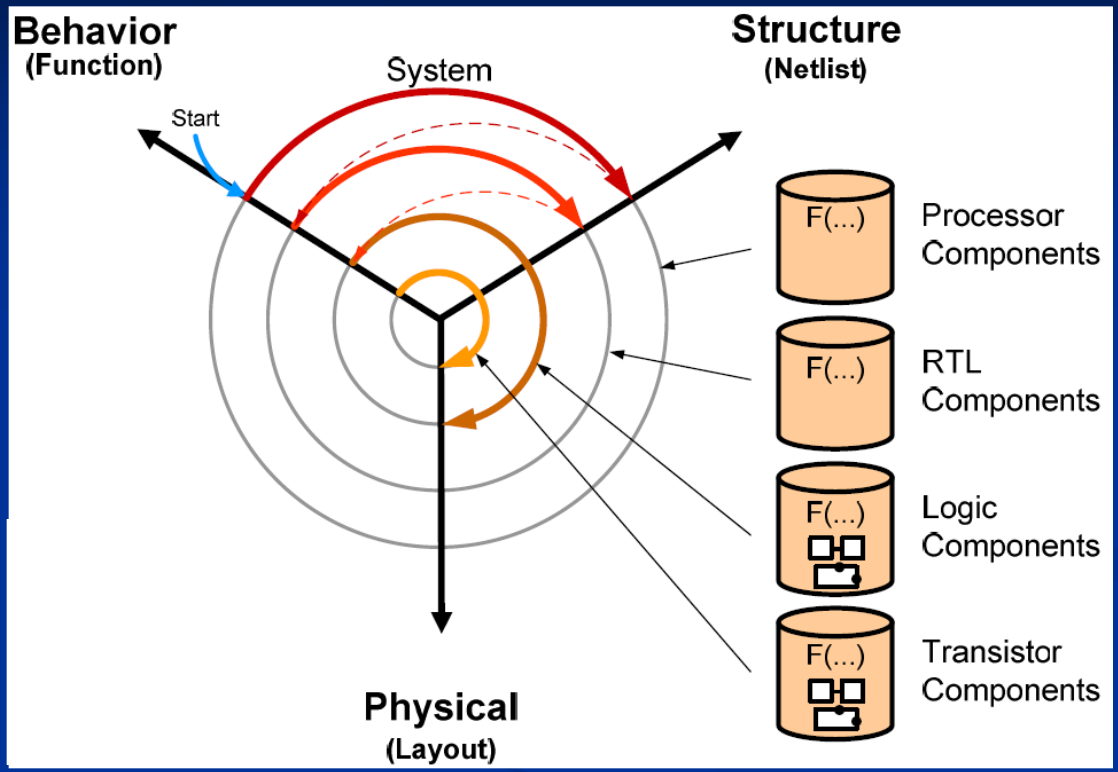
- Pros
  - Highest level of customization possible on each abstraction level
  - Only one small physical transistor library needed
  - Only one physical (layout) design at the end
- Cons
  - Difficult metric estimation (e.g. cost, power, performance) on upper levels because layout is not known until the end
  - Impact of design decisions at higher level not clear

Meet-in-the-Middle Design Methodology: Option 1



- Combines top-down and bottom-up
- Processor level where they meet
- MoC is synthesized into processor components
- Processor components are synthesized with RTL library
- System layout is generated with RTL components

Meet-in-the-Middle Design Methodology: Option 2



- Logic level where they meet
- MoC is synthesized with processor components
- Processor components are synthesized with RTL library
- RTL components are synthesized with standard cells
- System layout is performed with standard cells
- Two levels of layout

Meet-in-the-Middle Design Methodology

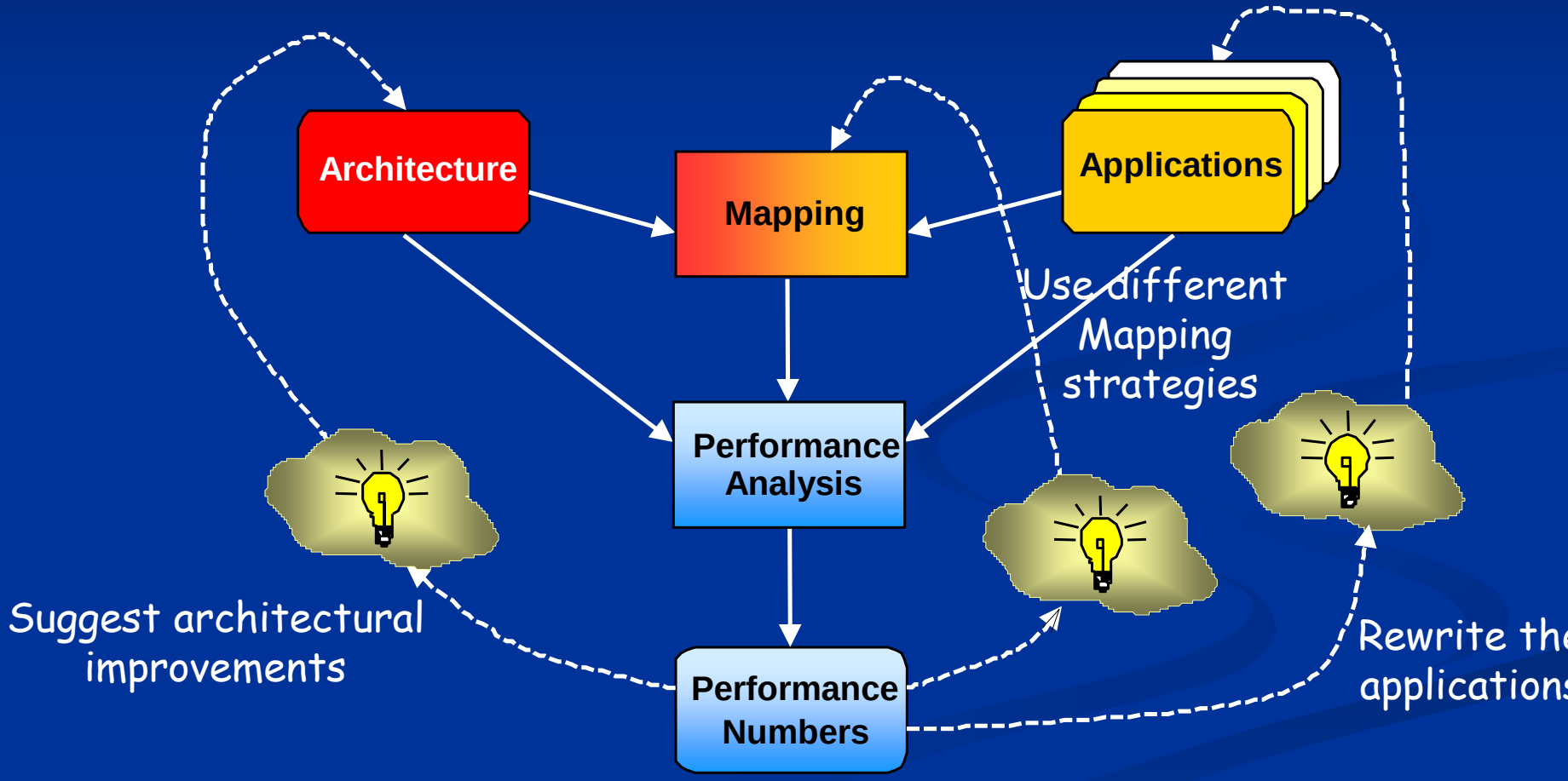
- Pros
  - Shorter synthesis
  - Less layout
  - Less physical libraries
  - Better metric estimation
- Cons
  - Still needs several physical libraries
  - More then one layout
  - Library components may not be optimal

Specify-Explore-Refine Methodology: MPSoC design aspects

- This methodology can be defined in two aspects:
  - ***Synthesis*** oriented design aspect
  - ***Quality Assessment*** oriented design aspect
- For both aspects, there exists corresponding so called **Y-chart** design methodology
  - Gajski et al. Y-chart [1983 and modified several times up to now]
    - covers mainly the synthesis aspect
  - Kienhuis et al. Y-chart [1997 and extended later]
    - covers mainly the quality assessment aspect

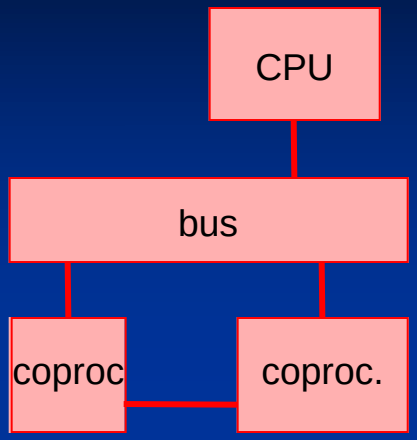
Y-chart Design Methodology [Kienhuis et al.] – quality assesment

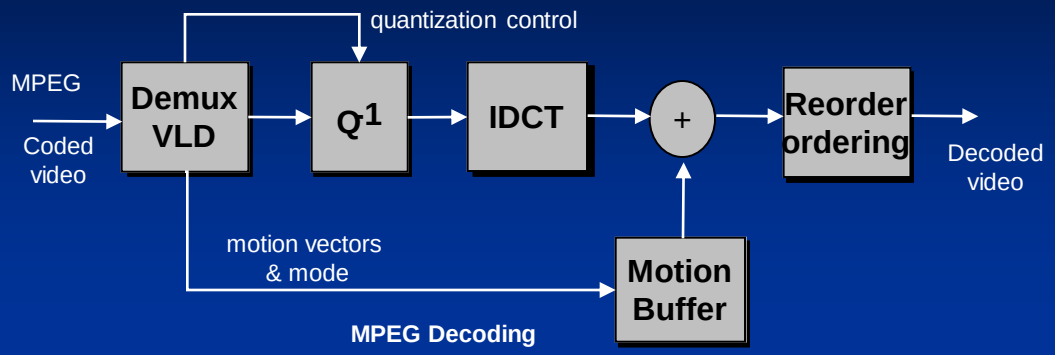
Separation of Concerns: application vs. architecture modeling



Three different ways to improve the performance of a system

Models of Application and Architecture



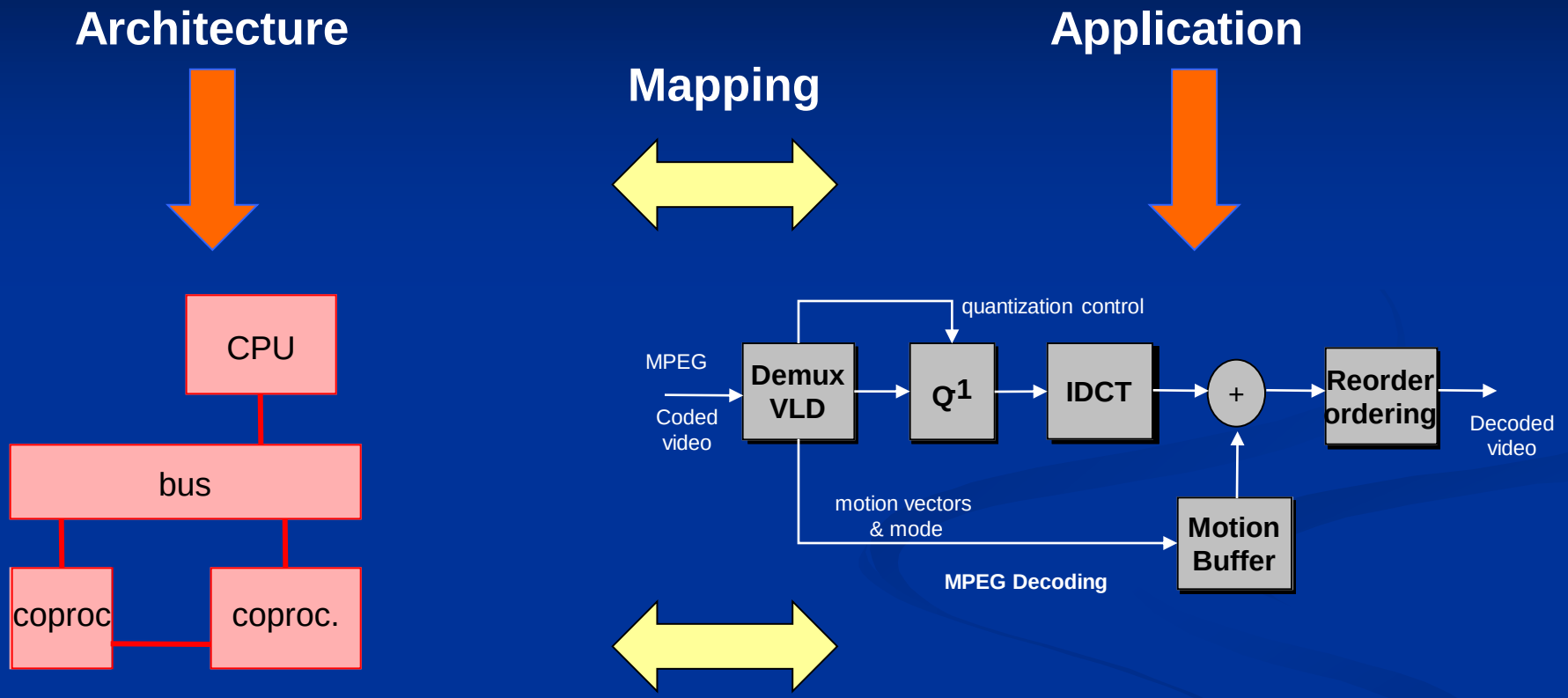


Both describe a network of components that perform a particular function and that communicate in a particular way

- Architecture Model:
  - Resources
    - ALUs, PE, etc.
    - Registers, SRAM, DRAM
    - Busses, Switches
  - Communication
    - Bits, Bytes
- Application Model:
  - Computations
    - IDCT, SQRT, Quantizer
  - Communication
    - Pixels, Blocks

Mapping

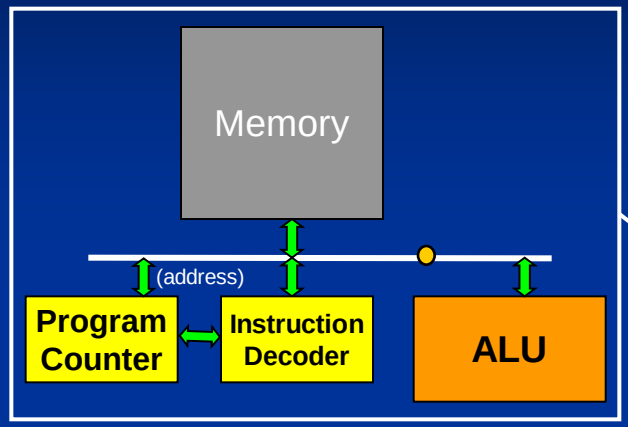
Mapping: Specifies the relation between the two models



- We formalize the descriptions of these 2 networks by using:
  - *Models of Computations* (MoC)
  - *Models of Architectures* (MoA less mature than MoCs)
- When the MoC and MoA match, simple mapping is possile!

Mapping Example 1: MoC and MoA match

Micro Processor



Sequential Program

```
for i=1:1:10
  for j=1:1:10
     A(i,j) =FIR();
  end
end
for i=1:1:10,
  for j=1:1:10,
    A(i,j) =SRC( A(i,j) );
  end
end
```

Compiler (like GCC)

Simulator

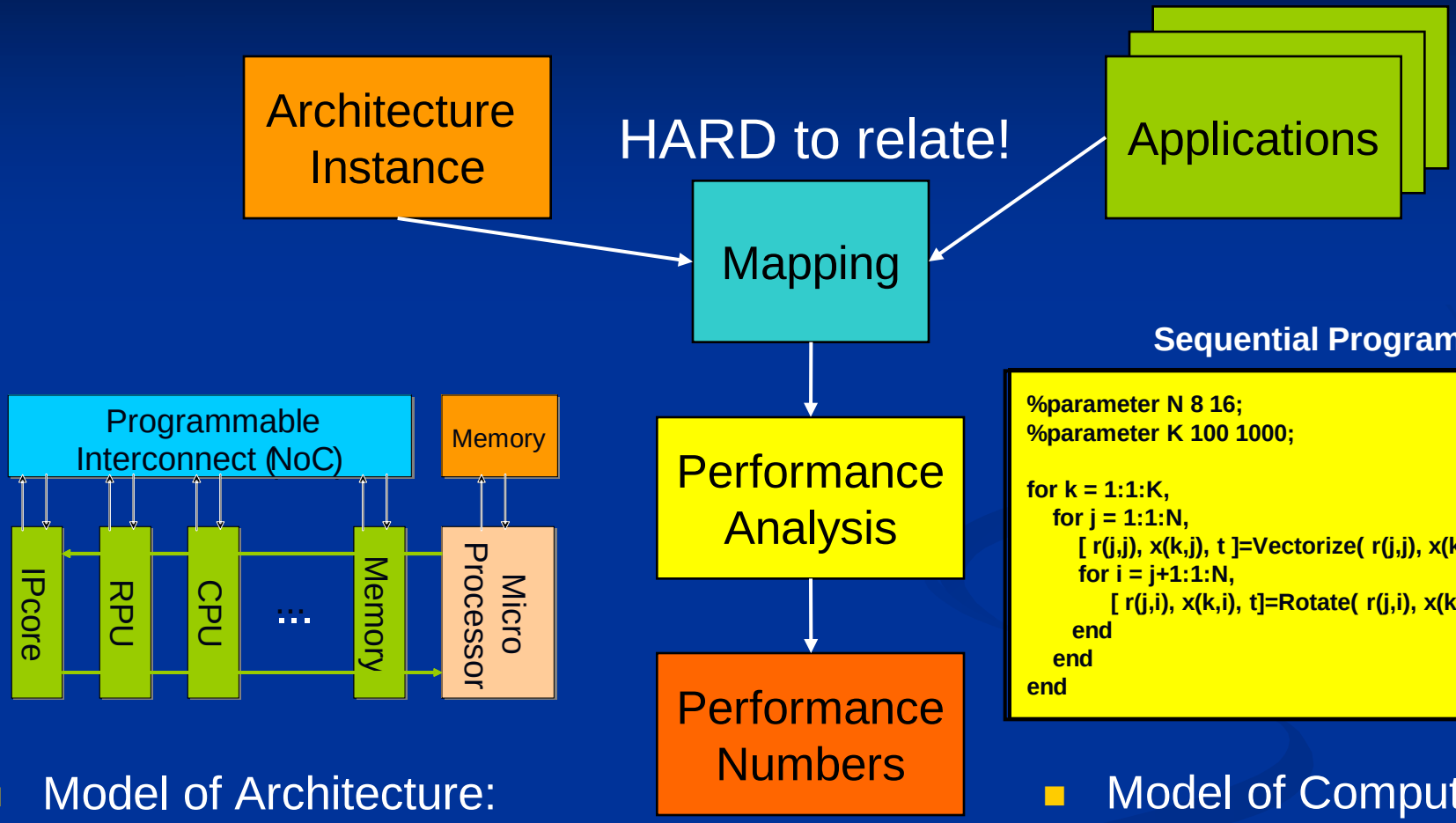
Natural FIT

Performance Numbers

- Model of Architecture:
  - Sequential (Program Counter)
  - One item over the bus at the time
  - Shared Memory

- Model of Computation:
  - Sequential
  - Shared Memory

Mapping Example 2: MoC and MoA do NOT match

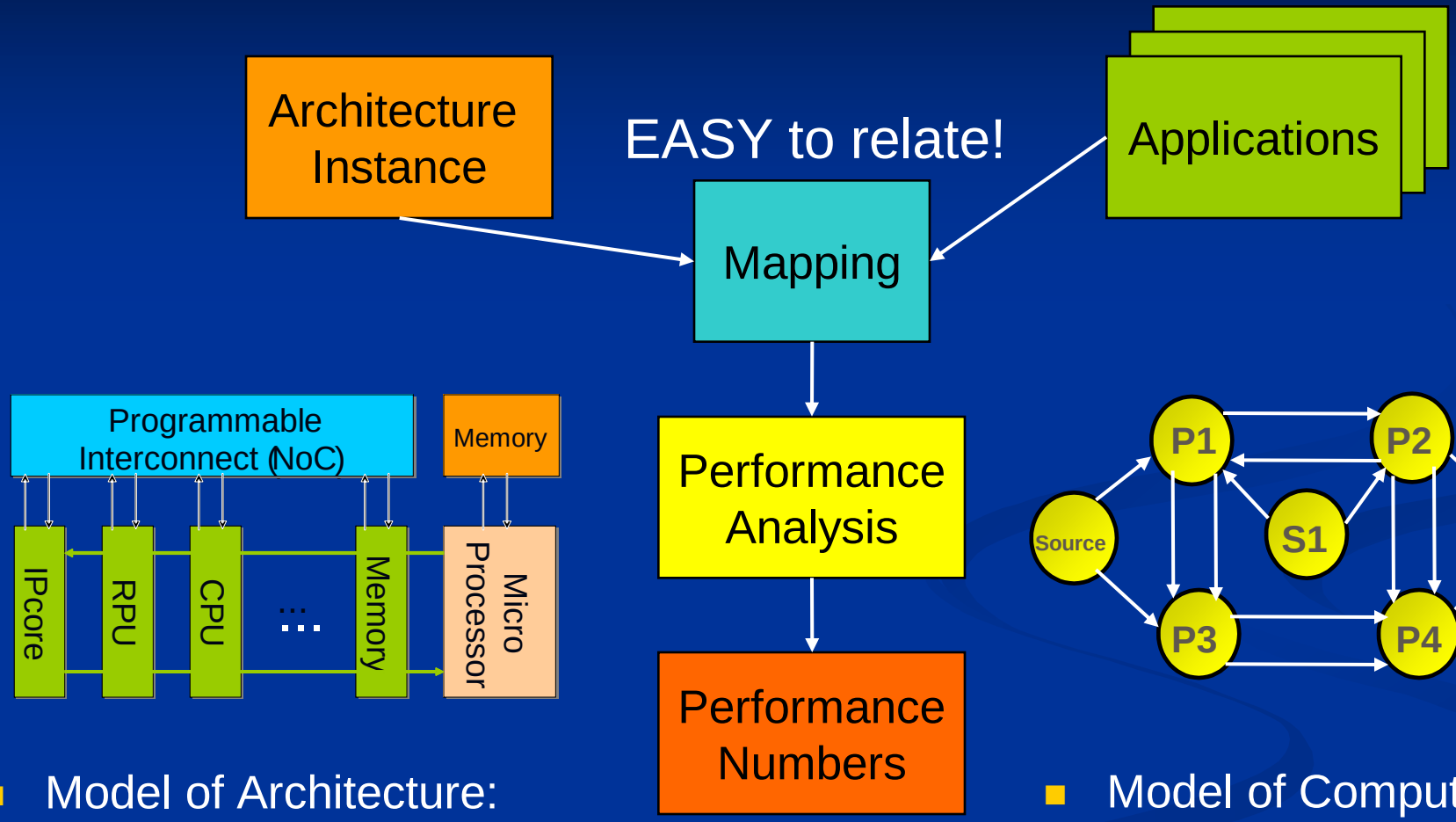


- Model of Architecture:
  - Parallel components
  - Heterogeneity
  - Distributed Memory

NO Natural FIT

- Model of Computation:
  - Sequential Execution
  - Global Memory

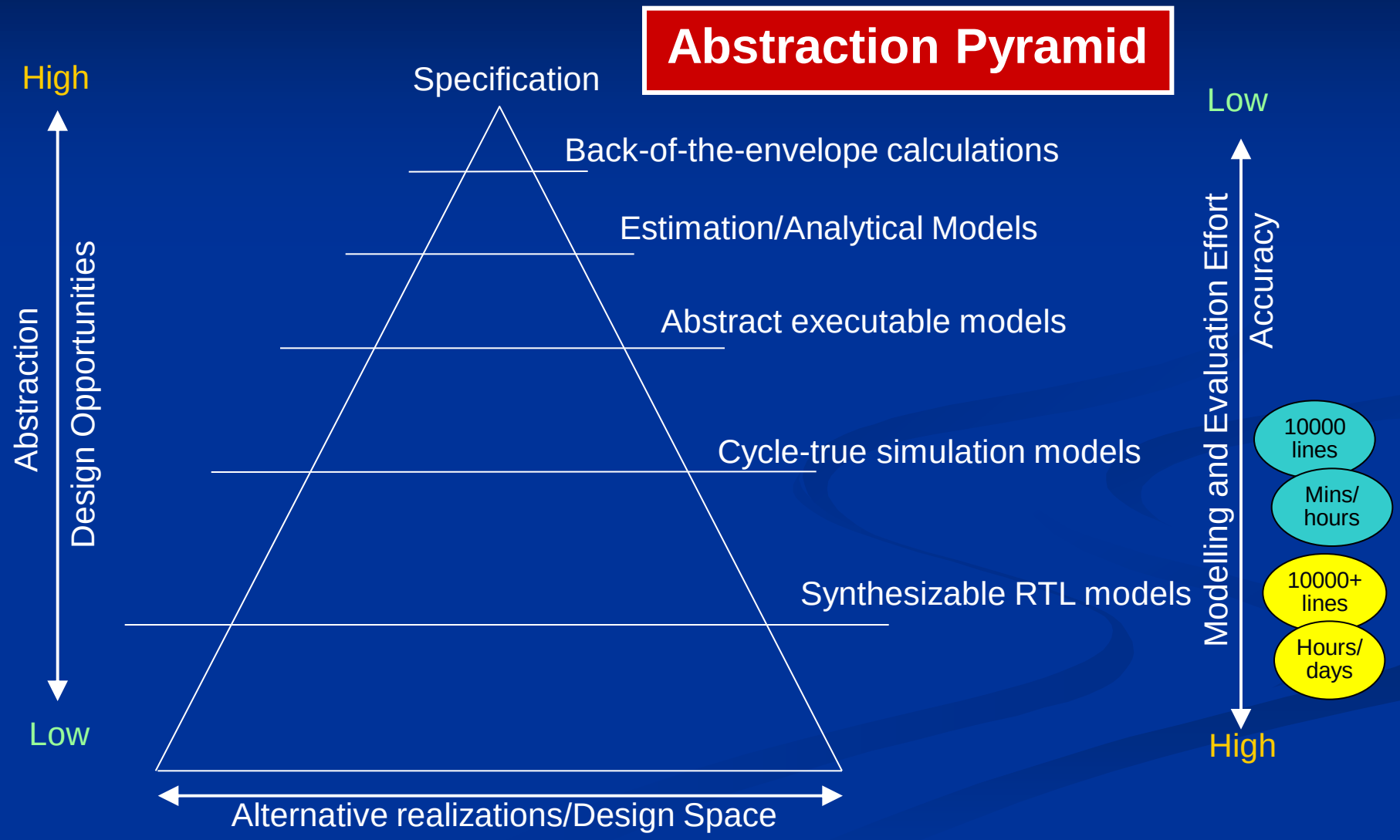
Mapping Example 3: MoC and MoA DO match



- Model of Architecture:
  - Parallel components
  - Heterogeneity
  - Distributed Memory

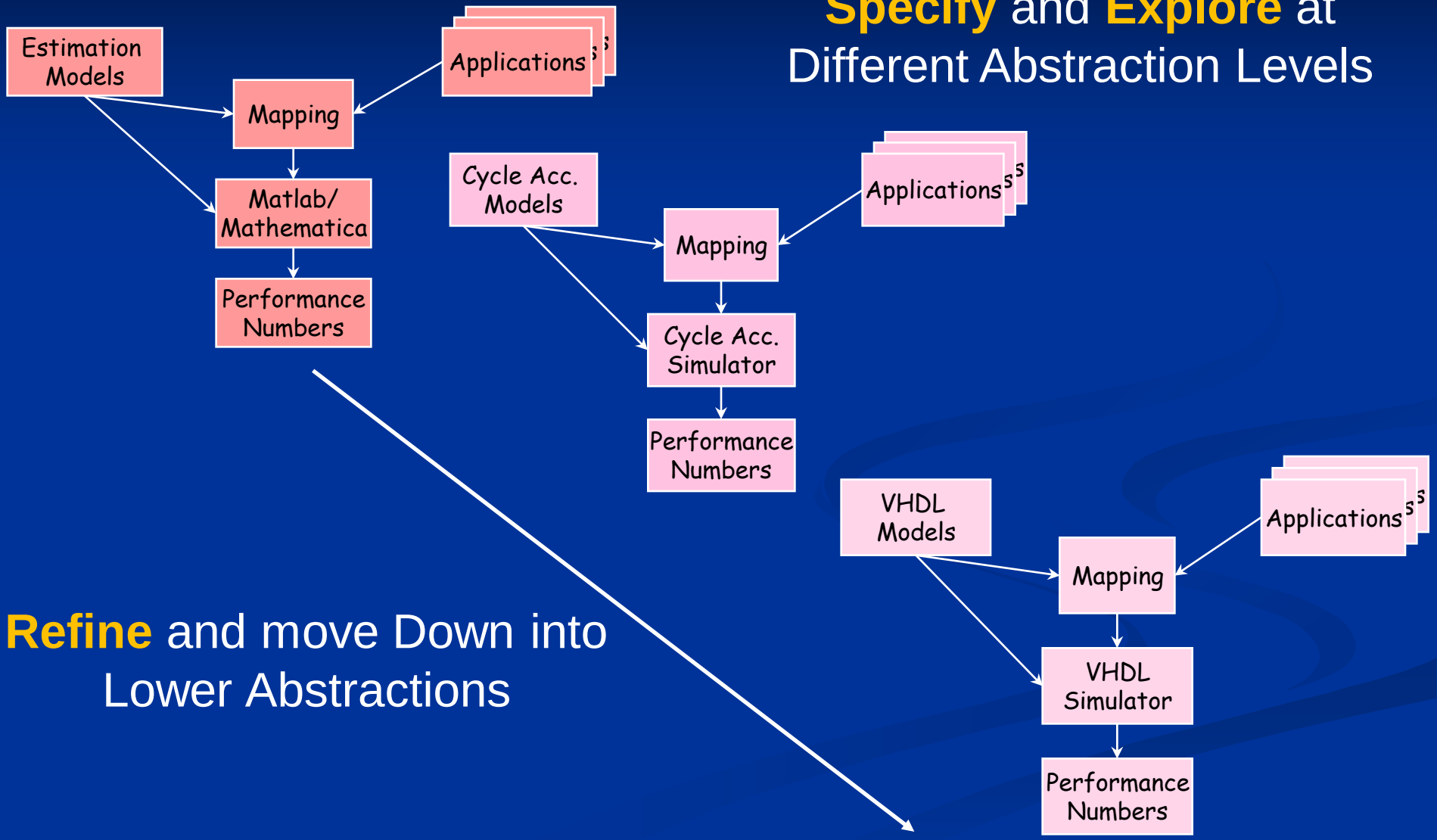
- Model of Computation:
  - Process Networks
  - Distributed Memory
  - Distributed Control

Y-chart MPSoC Design BUT at Which Level of Abstraction?



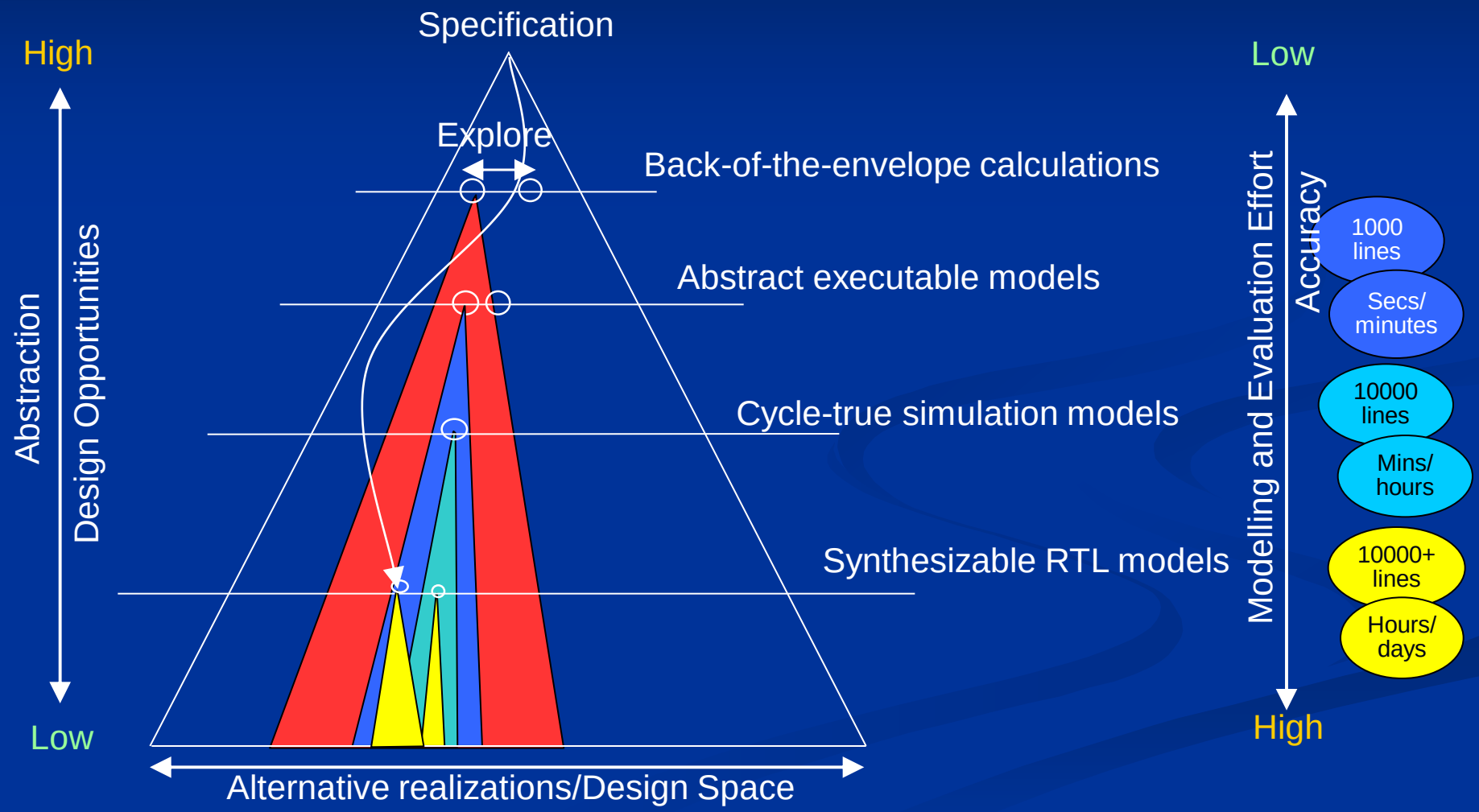
Stack of Y-charts

Specify and **Explore** at Different Abstraction Levels



Refine and move Down into Lower Abstractions

Design by Stepwise Refinement: Narrow-down the Design Space



Some key points …

- Many different design methods are in use
  - One for every group, product, and company
- They differ in:
  - Input specification, MoC
  - Modeling styles and languages
  - Abstraction levels and amount of detail
  - Verification strategy and prototyping
  - CAD tools and component libraries
- They are based on the Y-chart methodologies!

Toward Composable MPSoC Design

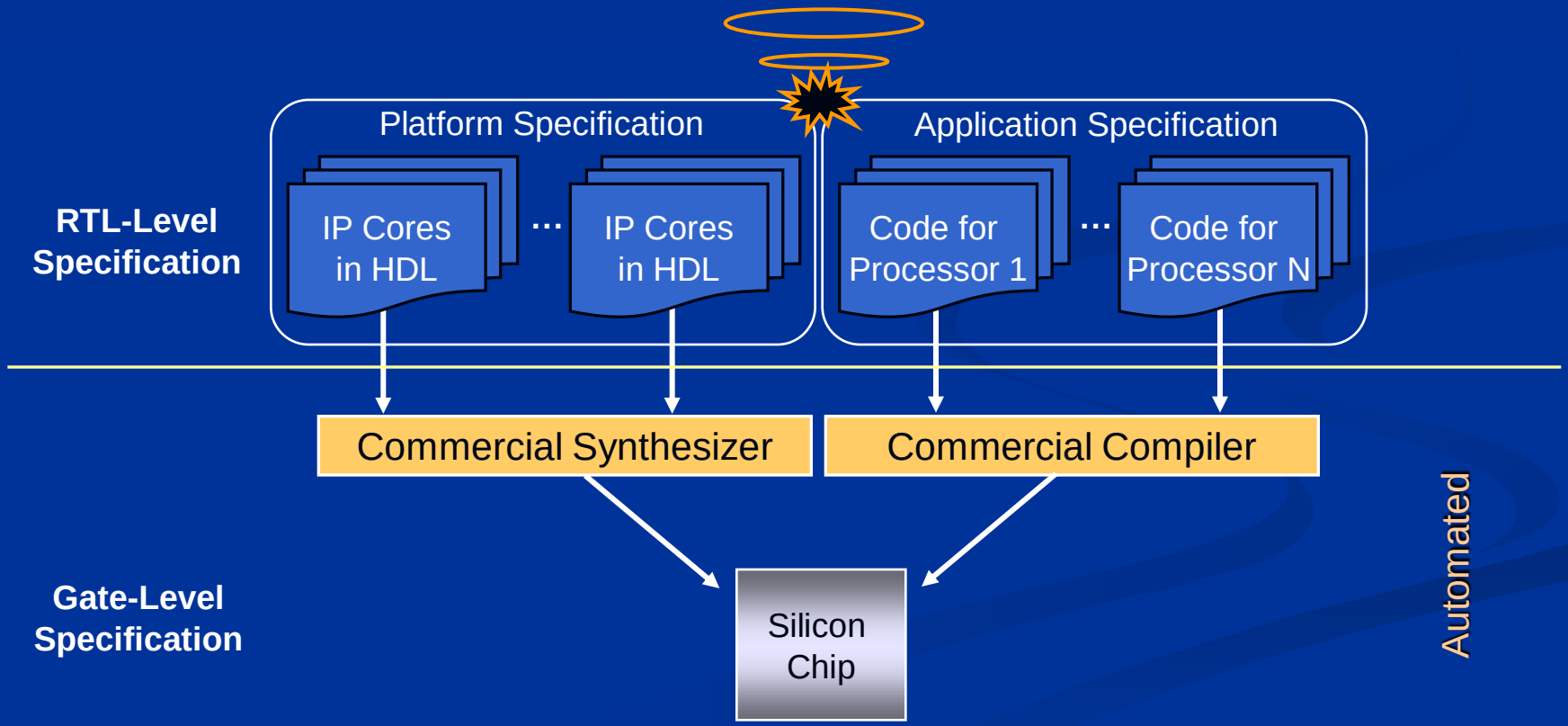
Leiden Embedded Research Center,
Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands

Universiteit van Amsterdam

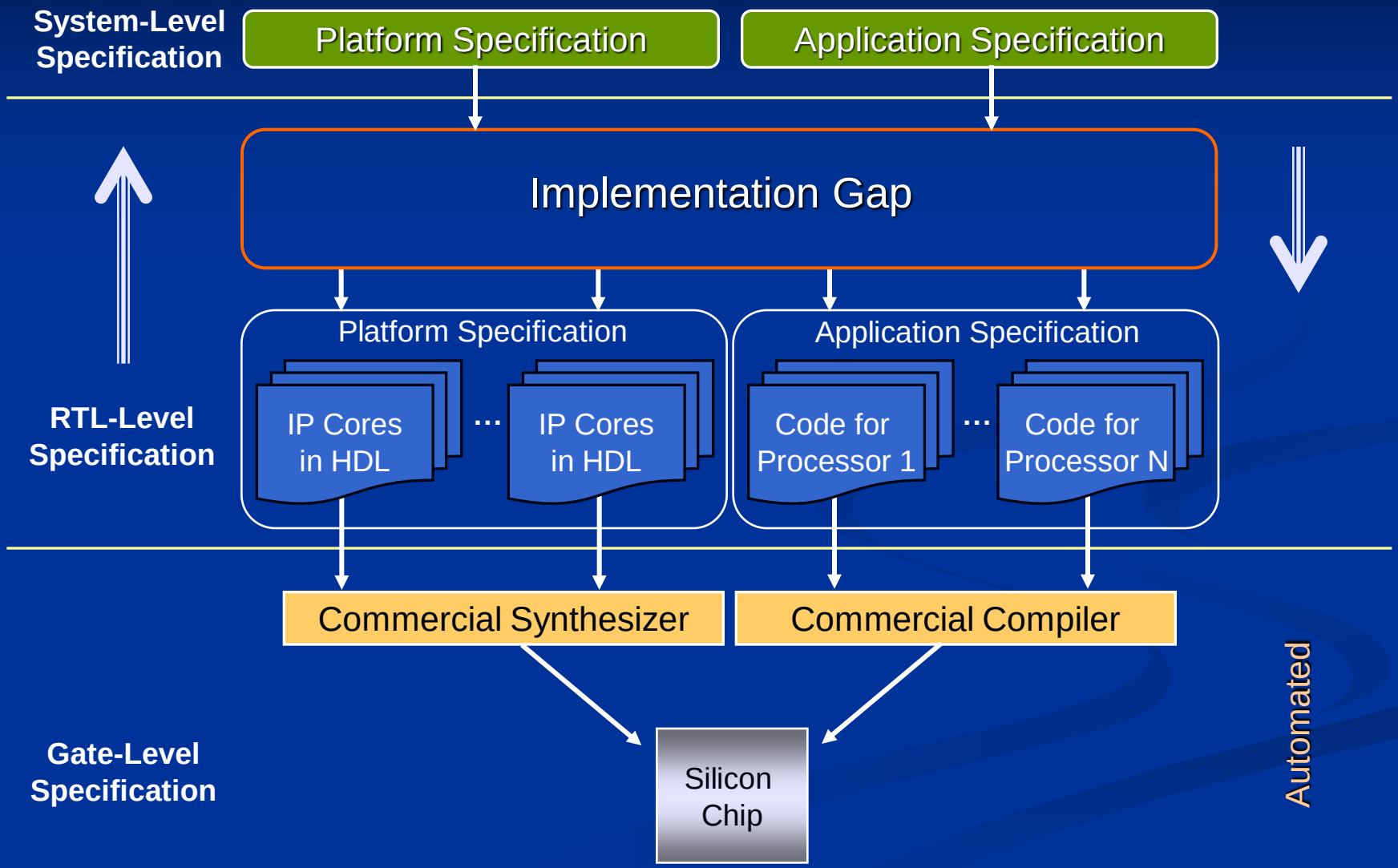
Motivation

Design of Multi-processor System:

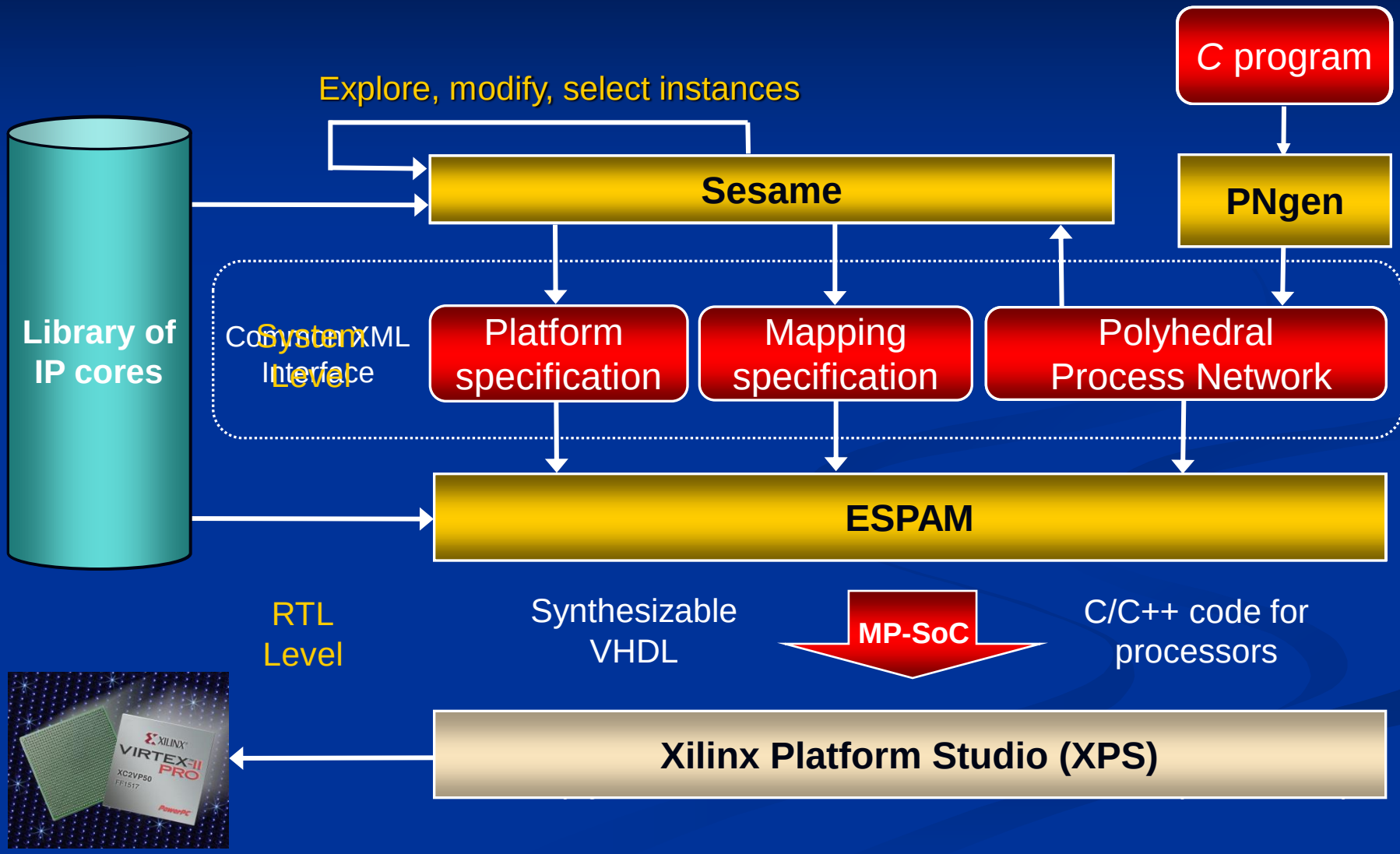
Too detailed, time consuming and error-prone design



Motivation


System-Level Specification
Platform Specification
Application Specification
Implementation Gap
RTL-Level Specification
Platform Specification
IP Cores in HDL
IP Cores in HDL
Application Specification
Code for Processor 1
Code for Processor N
Commercial Synthesizer
Commercial Compiler
Gate-Level Specification
Silicon Chip
Automated

The Daedalus design-flow



Merits of the Daedalus design-flow

- Automated parallelization of applications into parallel specifications (PPNs)
- Automated synthesis of MPSoC platforms at system level, in a plug-and-play fashion
- Automated mapping of parallel application specifications onto MPSoC platforms
- Steering by means of efficient system-level design space exploration

So, what about the name Daedalus?

- Daedalus means "cunning worker" in Latin
- He was an innovator in many arts
- Daedalus was the father of Icarus

- Analogy:
  - It's very good technology
  - But there are still limitations

- "Don't fall into the sea"!

Daedalus and Icarus, by Charles Paul Landon, 1799