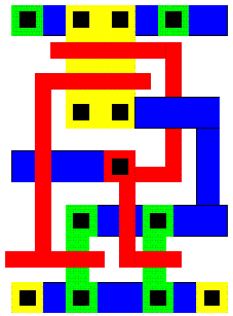
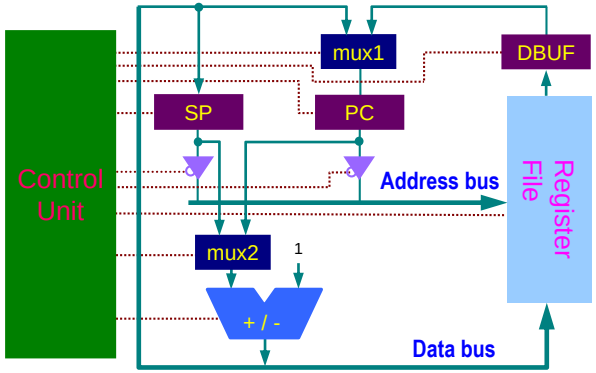
Introduction to System Level Design

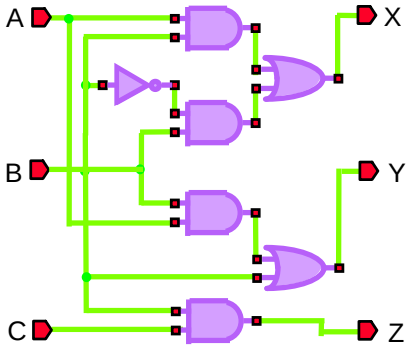
Outline

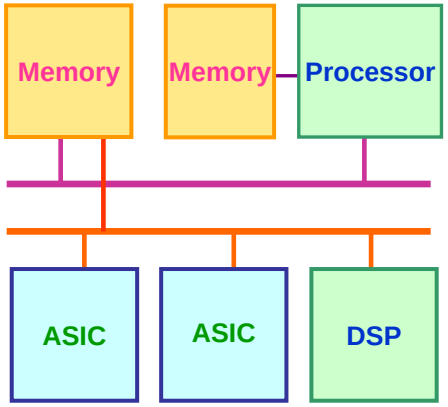
- ***SoC Design Flow***
- ***System Architecture Development***
- ***System-Level Verification***
- ***Electronic System-Level Design***

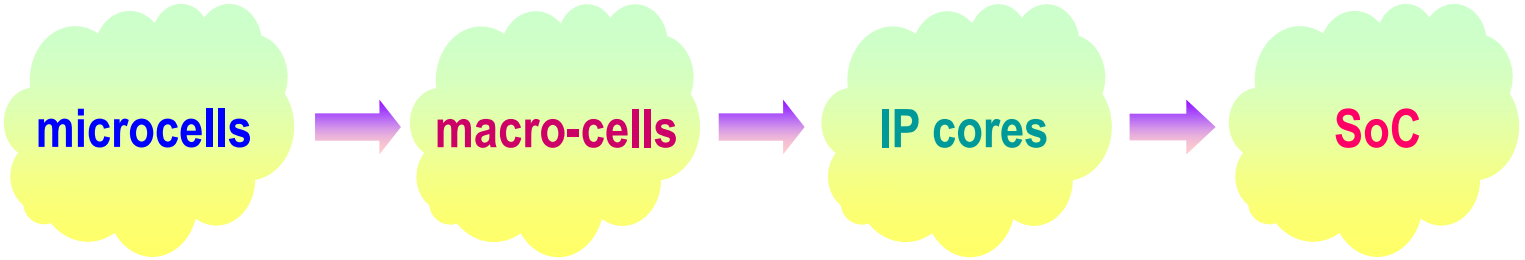
Evolution of IC Design



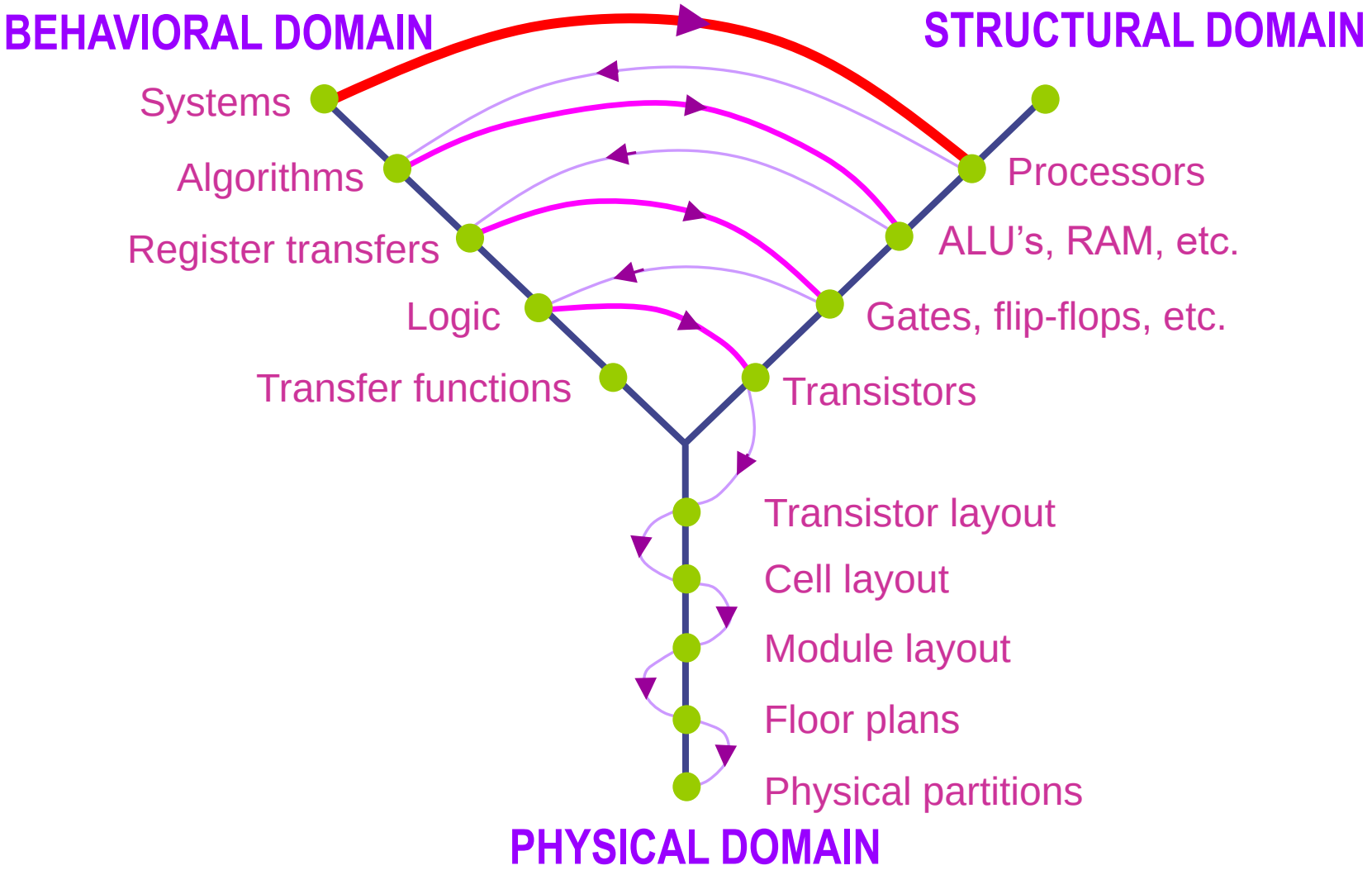






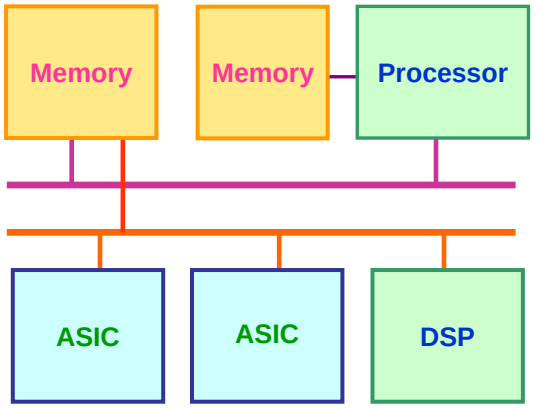


Design Flow


BEHAVIORAL DOMAIN
STRUCTURAL DOMAIN
Systems
Algorithms
Register transfers
Logic
Transfer functions
Processors
ALU's, RAM, etc.
Gates, flip-flops, etc.
Transistors
Transistor layout
Cell layout
Module layout
Floor plans
Physical partitions
PHYSICAL DOMAIN

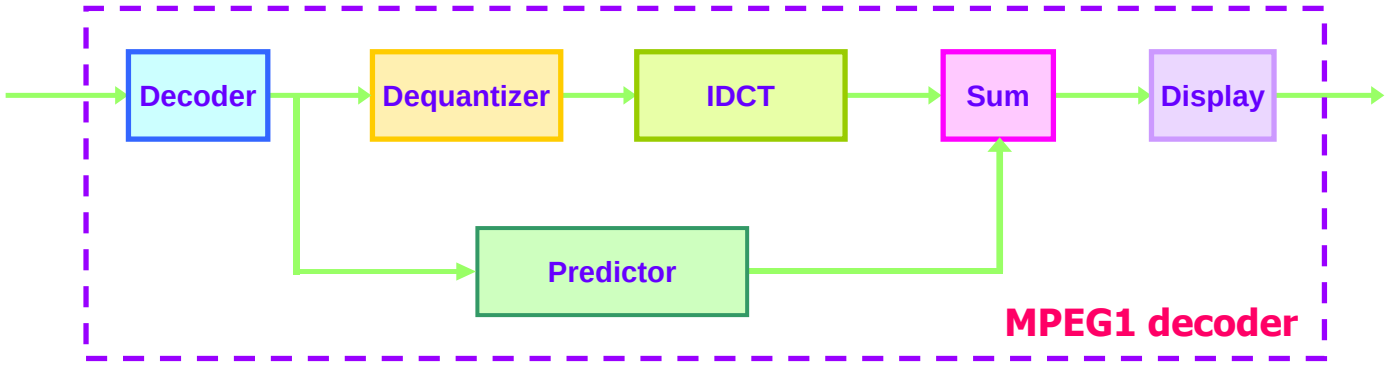
System-Level Design (1/2)

- **Design starting from a system specification**
  - C/C++, SystemC, SystemVerilog, …
  - Constraints: performance, resource, area, power, …
- **Algorithm Development**
- **Architecture Development**
  - HW/SW Partitioning
  - Communication Synthesis
  - Memory Infrastructure
  - Multiple Supply Voltages, … …
  - Synthesis-based and Simulation-based
- **System-level Verification**

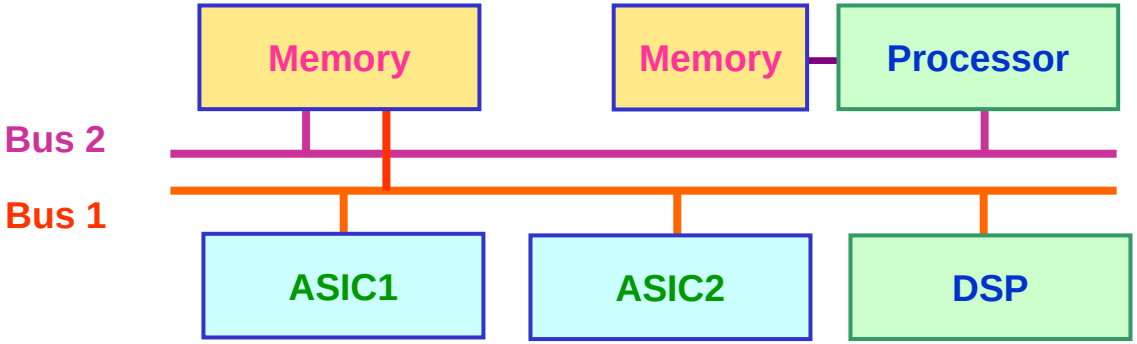


System-level Design (2/2)

system specification



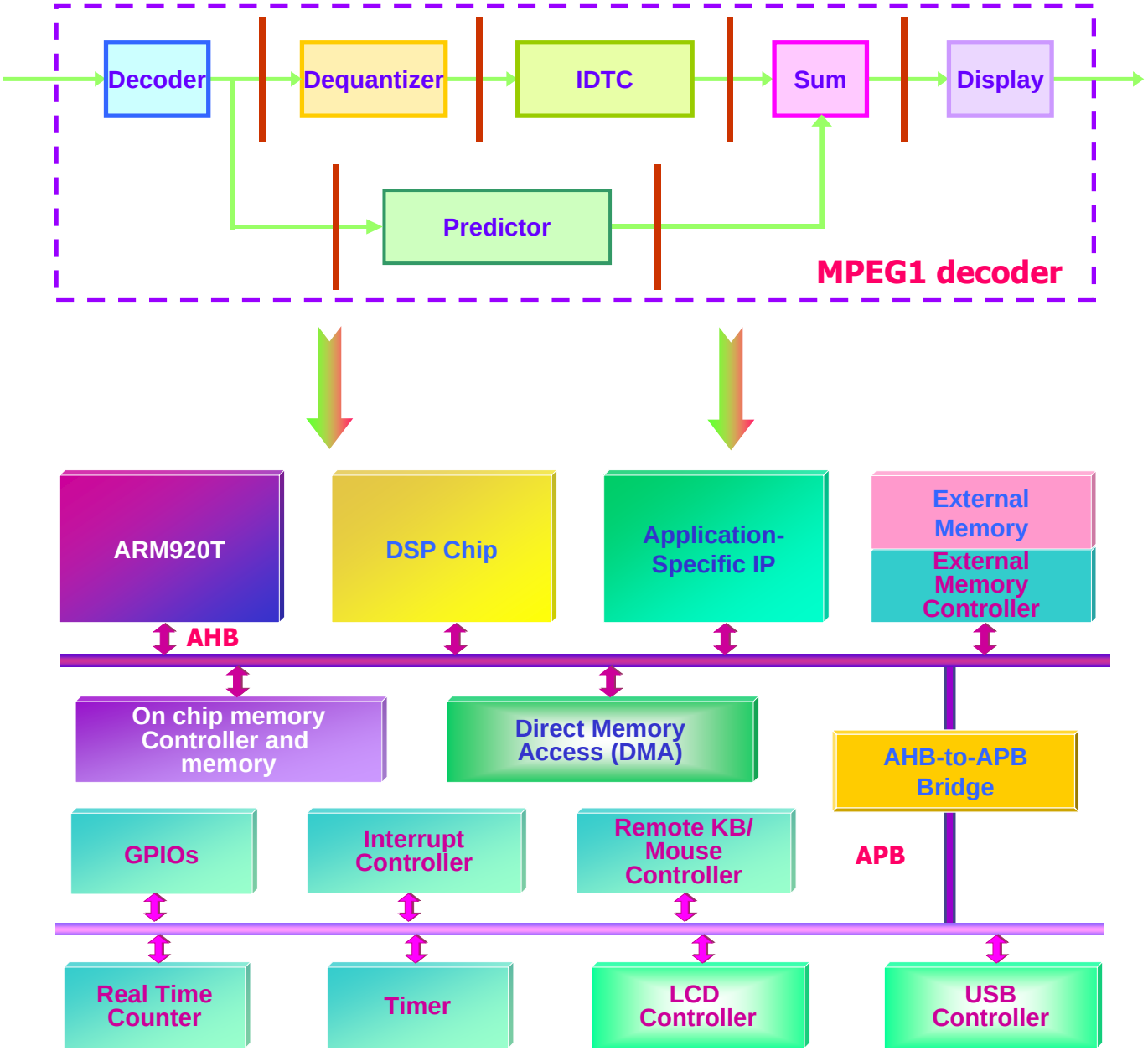
Hardware-software co-design



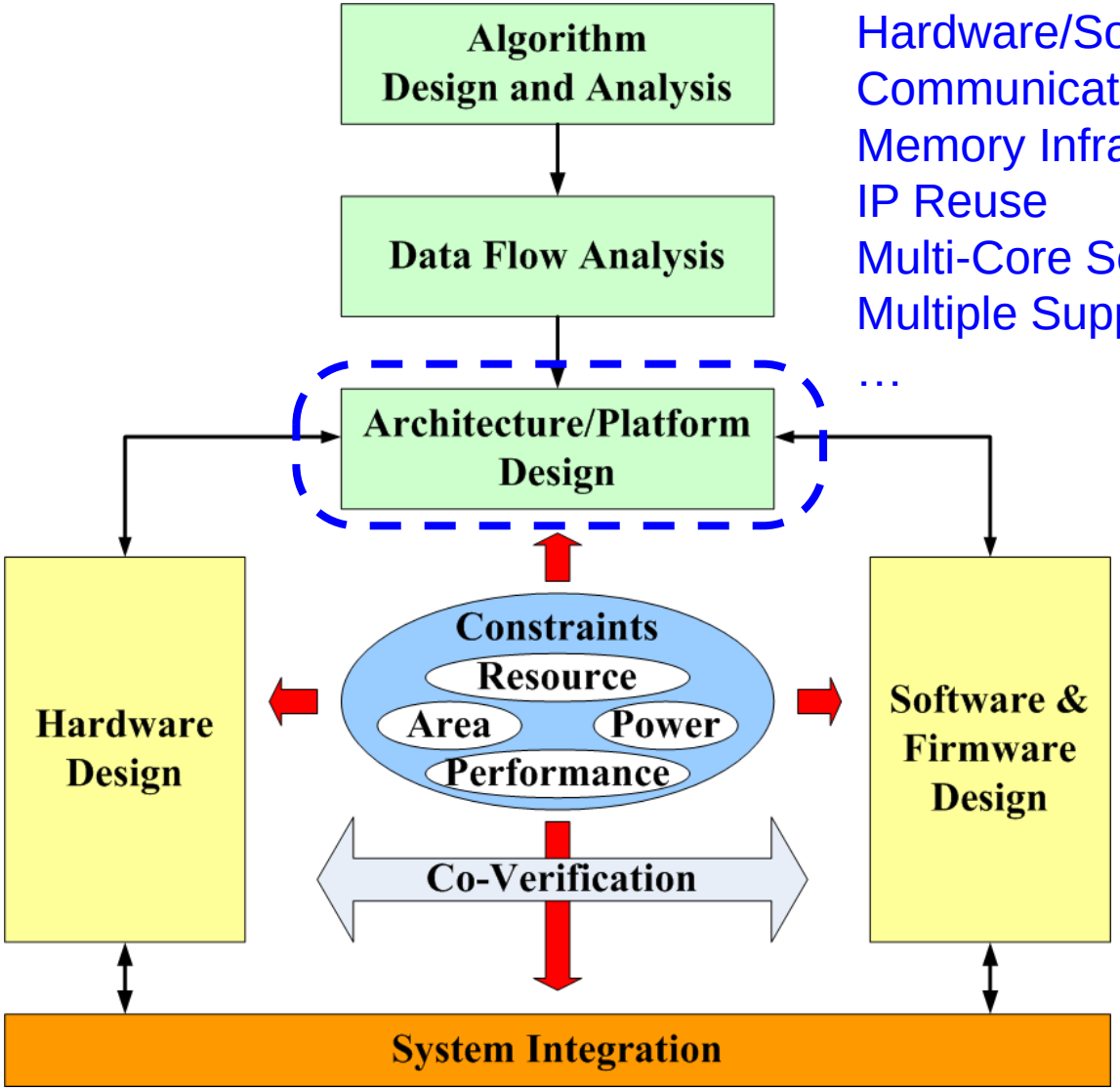
system architecture

SoC Design Challenges

- **Design Complexity & Verification**
  - Microprocessors, DSPs, RTOS's, Digital/Analog IPs, On-chips buses, …
  - HW/SW co-verification, Digital/analog/memory circuit verification, …
- **How to Conquer the Complexity**
  - Use a Known Real Entity
    - Pre-designed component: Semiconductor **intellectual property (IP)** reuse
    - Platform: Architecture reuse (Platform-based design)
  - Design Methodology
    - Design flow and EDA tools
    - Hierarchical design methodology

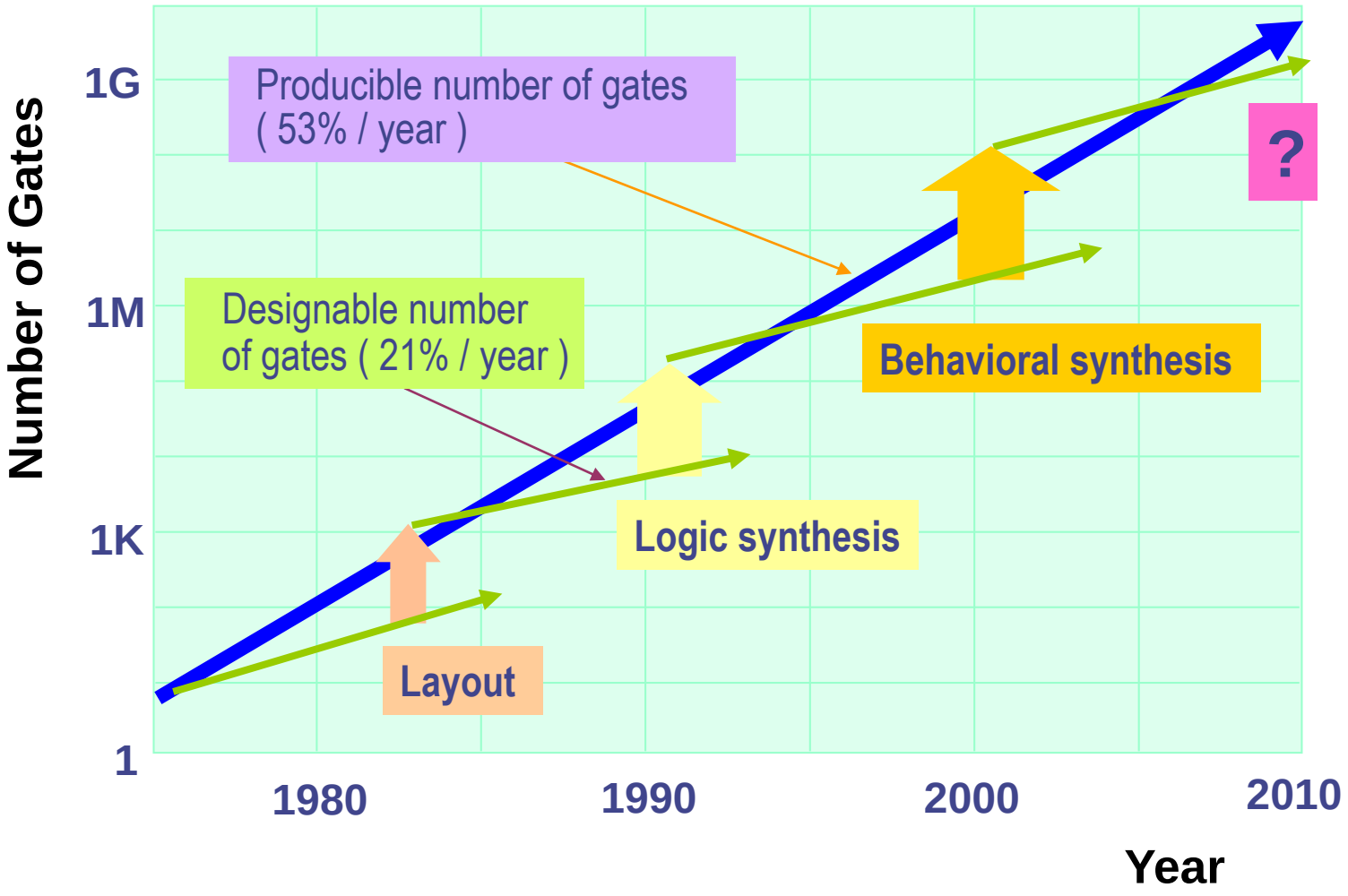

Decoder
Dequantizer
IDTC
Sum
Display
Predictor
MPEG1 decoder
ARM920T
DSP Chip
Application-Specific IP
External Memory
External Memory Controller
AHB
On chip memory Controller and memory
Direct Memory Access (DMA)
AHB-to-APB Bridge
GPIOs
Interrupt Controller
Remote KB/ Mouse Controller
APB
Real Time Counter
Timer
LCD Controller
USB Controller

Design Flow for a Typical SoC



Hardware/Software Partitioning
Communication Synthesis
Memory Infrastructure
IP Reuse
Multi-Core SoC
Multiple Supply Voltages
…

Electronic Design Automation (EDA) tools



Design Flow Styles (1/2)

- **Top-down approaches**
  - start design from the system behavior representing the design's functionality
  - generate a system architecture from the behavior
  - gradually reach the implementation model by adding implementation details
  - the initial version of design can be designed using the approach
  - after this step we have a predefined platform which can be used further

Design Flow Styles (2/2)

- **Meet-in-the-middle approaches**
  - map the system behavior to the predefined system architecture, rather than generating the architecture from the behavior
- **Bottom-up approaches**
  - if designers want to replace an old IP by a new IP for the designed system, the approach can be exploited
  - assemble the existing computation components by inserting wrappers among them
  - focus on component reuse and wrapper generation

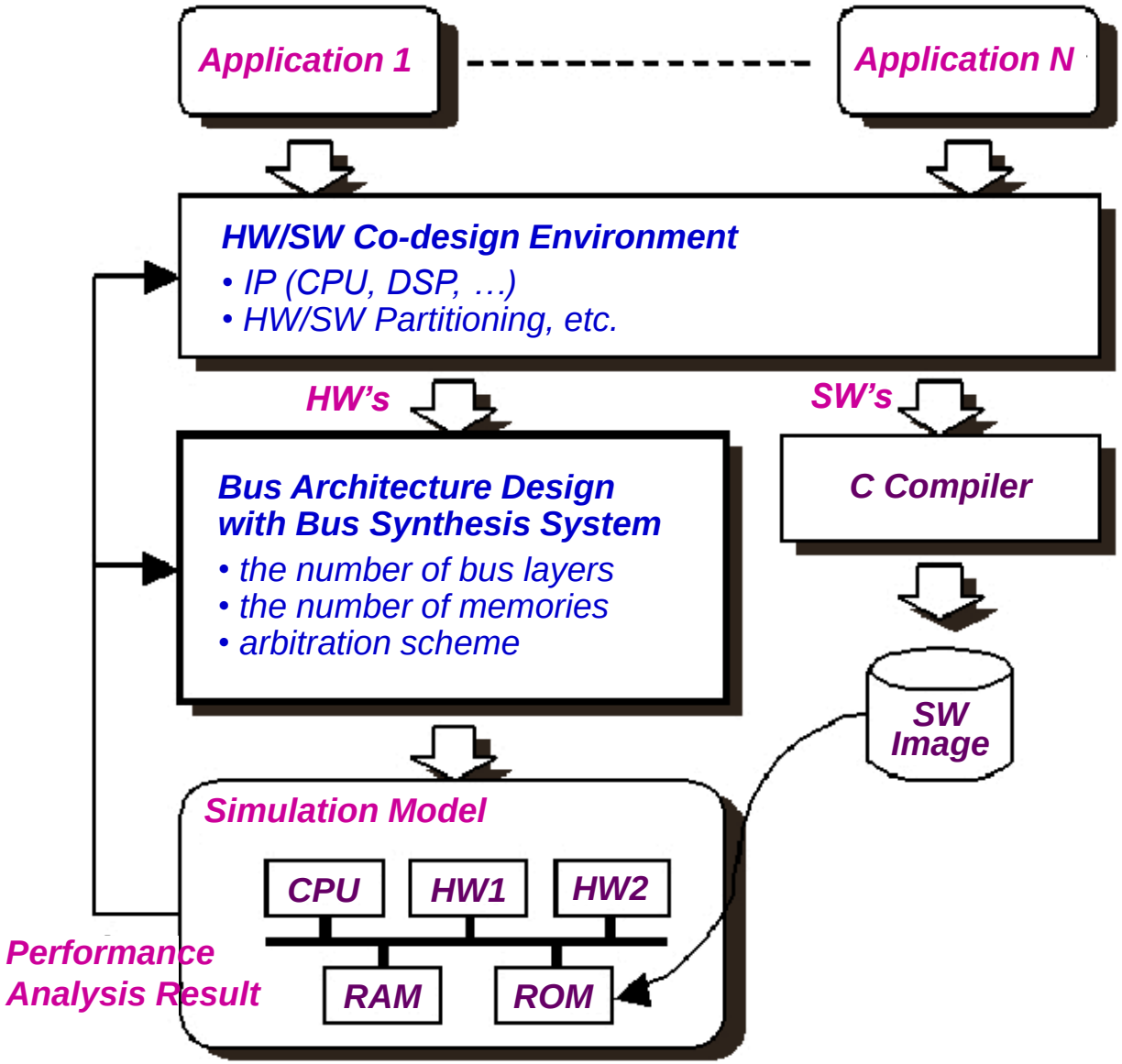
Outline

- SoC Design Flow
- System Architecture Development
- System-Level Verification
- Electronic System-Level Design
- Transaction-Level Model

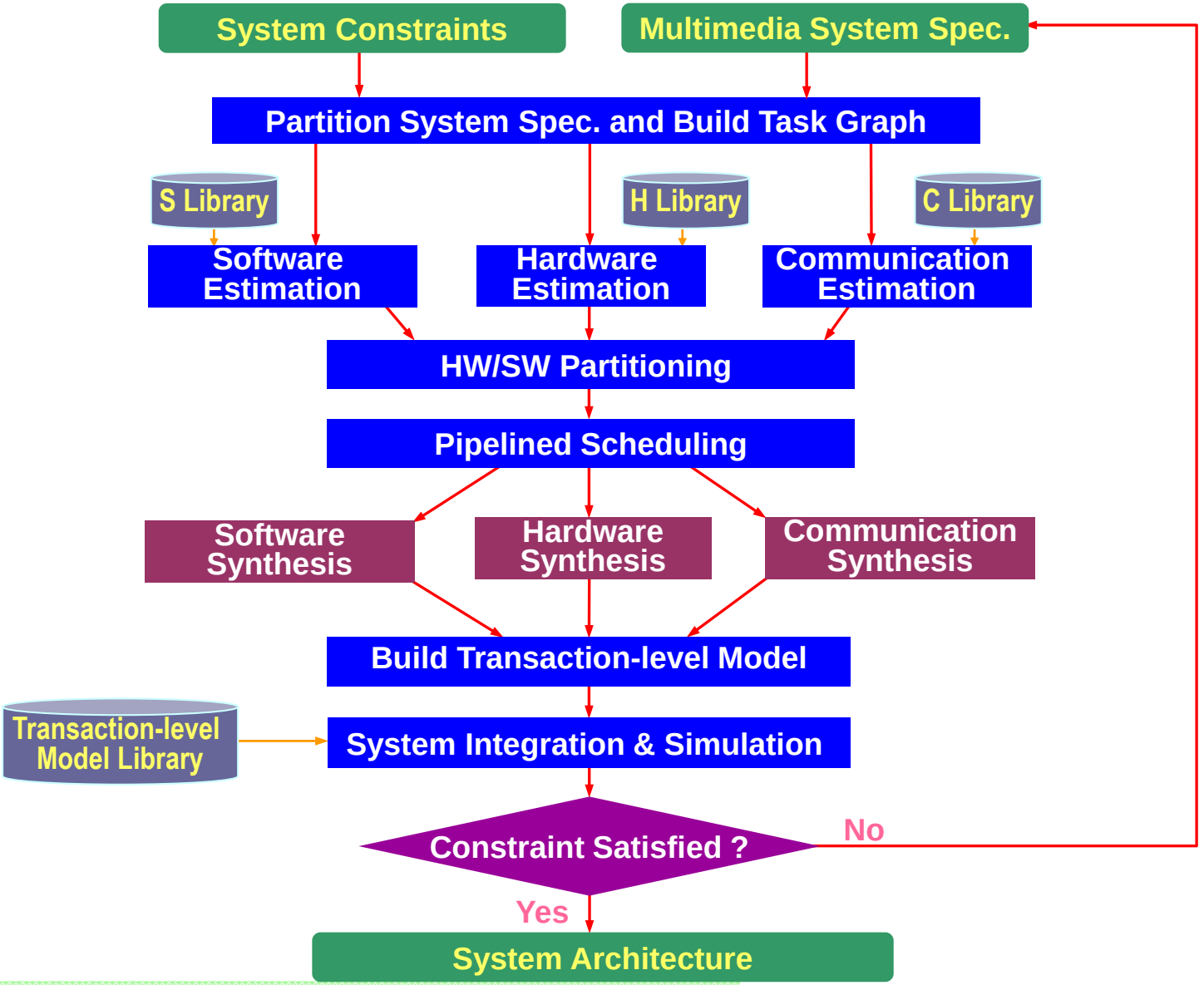
Architecture Development

- ***Synthesis-based*** **Methods**
  - often assume statically scheduled systems
  - cannot accurately model dynamic effects such as bus contention
  - estimates can be conservative (worst case)
- ***Simulation-based*** **Methods**
  - simulate the entire system using **models** of the components and their communication at different levels of abstraction
  - allows tradeoff between accuracy and simulation time
  - every evaluation of an alternative communication architecture will require a complete system simulation (components + bus)

Static System Architecture Synthesis



HW/SW Co-design



Definition of Co-design

- **Satisfy the system-level objectives by exploiting the trade-offs between hardware and software in a system through their concurrent design**

- **Key concepts**
  - Concurrent: hardware and software developed at the same time on parallel paths
  - Integrated: interaction between hardware and software development to produce design meeting performance criteria and functional specs

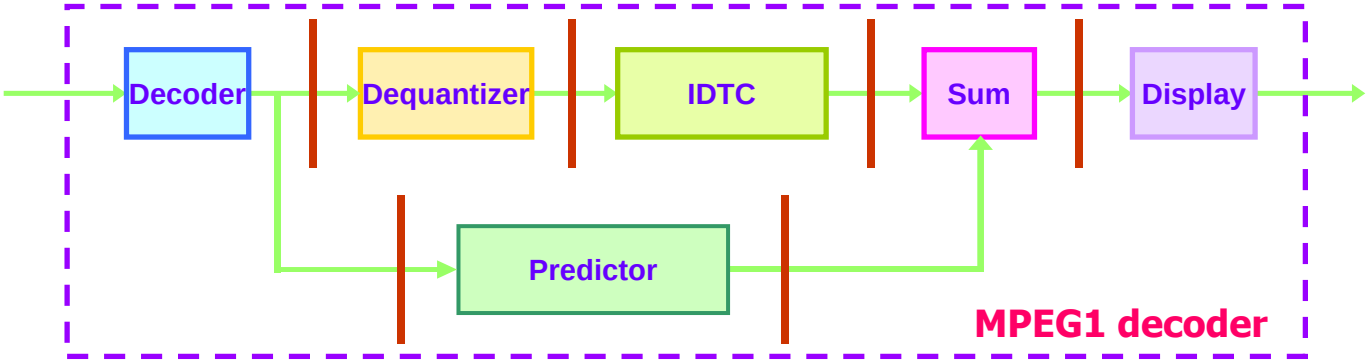
Co-simulation & Co-verification

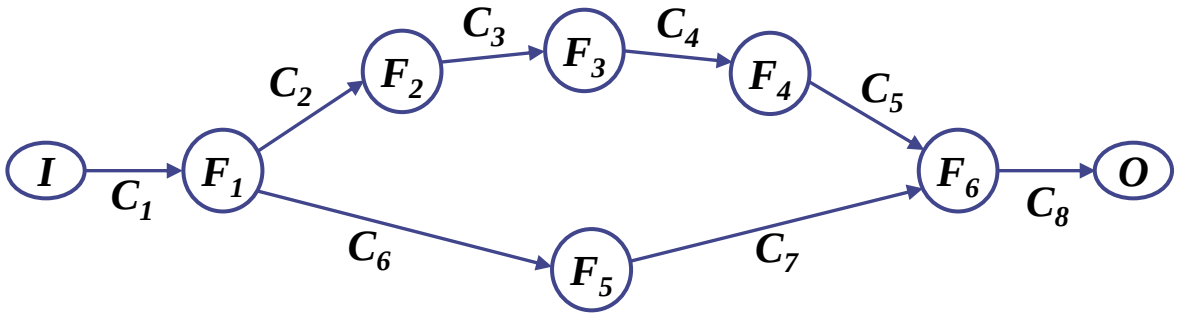
Co-simulation

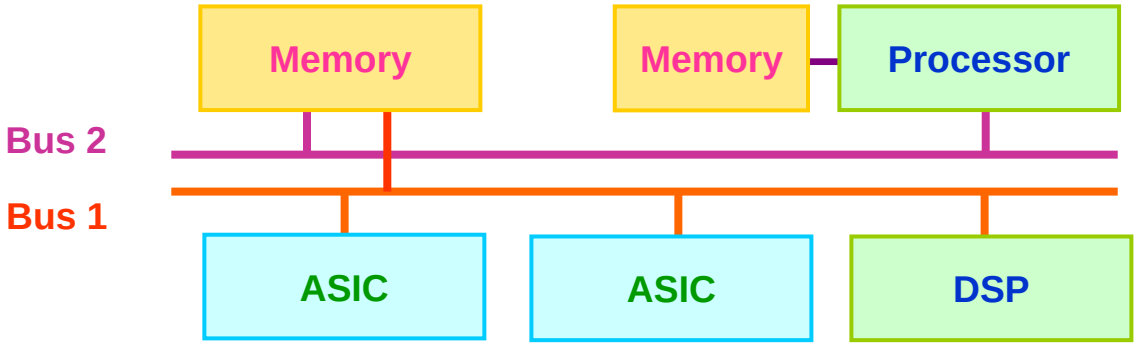
- two or more heterogeneous simulators working together to produce a complete simulation result
- Ex: a Verilog simulator working with a VHDL simulator, or a digital logic simulator working with an analog simulator

Co-verification

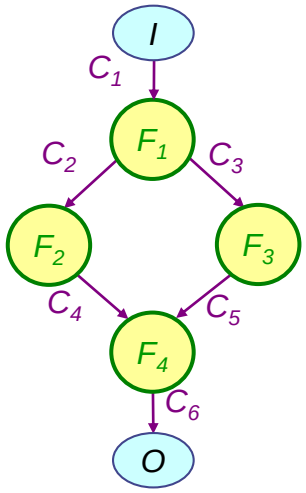
- means verifying SoC software executes correctly on SoC hardware
- To making sure that you are indeed implementing what you want


Decoder
Dequantizer
IDTC
Sum
Display
Predictor
MPEG1 decoder


I
F1
F2
F3
F4
F5
F6
O
C1
C2
C3
C4
C5
C6
C7
C8

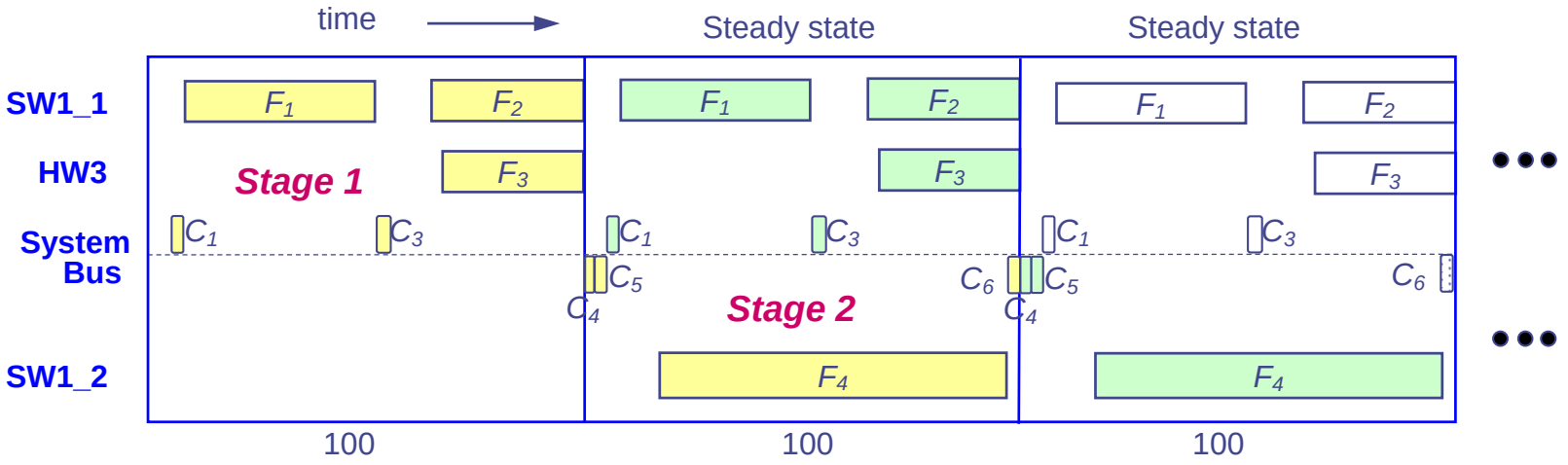

Memory
Memory
Processor
Bus 2
Bus 1
ASIC
ASIC
DSP

An Simple Example



F_i	ML_j	MA_j	$FT_{i,j}$
F_1	SW1	450	44
	SW2	600	36
	HW1	310	18
F_2	SW1	450	35
	SW2	600	25
	HW2	270	12
F_3	SW1	450	105
	HW3	560	32
F_4	SW1	450	80
	SW2	600	60

C_n	CT_n
C_1	3
C_2	3
C_3	4
C_4	2
C_5	3
C_6	3



Processor Selection

- **General-Purpose Processor**
  - extensible instruction sets
  - single-issue pipelined, simple superscalar pipelined (dual issue), VLIW
- **Application-Specific Instruction-set Processor (ASIP)**
  - has a configurable instruction set
  - provides a tradeoff between the flexibility of a general purpose CPU and the performance of an ASIC
- **Selecting a Processor**
  - based on performance, cost, power consumption
  - familiarity with the core family (and tools)
  - tools, operating system, support, documentation availability and cost
  - licensing and royalties, bus structure and performance, …

IP Selection

- **IP selection includes many concerns**
  - Functionality and requirements
  - Type of IP and its distribution format
    - **Soft IP**: synthesizable RTL (Verilog or VHDL)
    - **Hard IP**: fully designed, placed, and routed
    - **Firm IP**: provided as a net-list
  - IP protection from vendor and it's impact on licensing model, reluctance of vendor to let you evaluate before buying, etc.
  - Licensing and restrictions placed on IP and it's affect on product cost
  - Quality of IP and it's documentation

HW/SW Partitioning (1/2)

- **The process of deciding whether the required functionality is more advantageously implemented in hardware or software**
- **Implementation by software**
  - Pros: May run on high-performance processors at low cost (due to high-volume production)
  - Cons: Incurs high cost of developing and maintaining (complex) software
- **Implementation by hardware**
  - Pros: Provides higher performance via hardware speeds and parallel execution of operations
  - Cons: Incurs additional expense of fabricating ASICs

HW/SW Partitioning (2/2)

- The overall system cost and performance are affected by its partition into hardware and software components
- To achieve a partition that will give us the required performance within the overall system requirements (in size, weight, power, cost, etc.)
  - NP-complete problem
- The quality of a partition depends on performance/cost estimators
- Designer may be more interested in coarse-grained partition of the system

Software Estimation

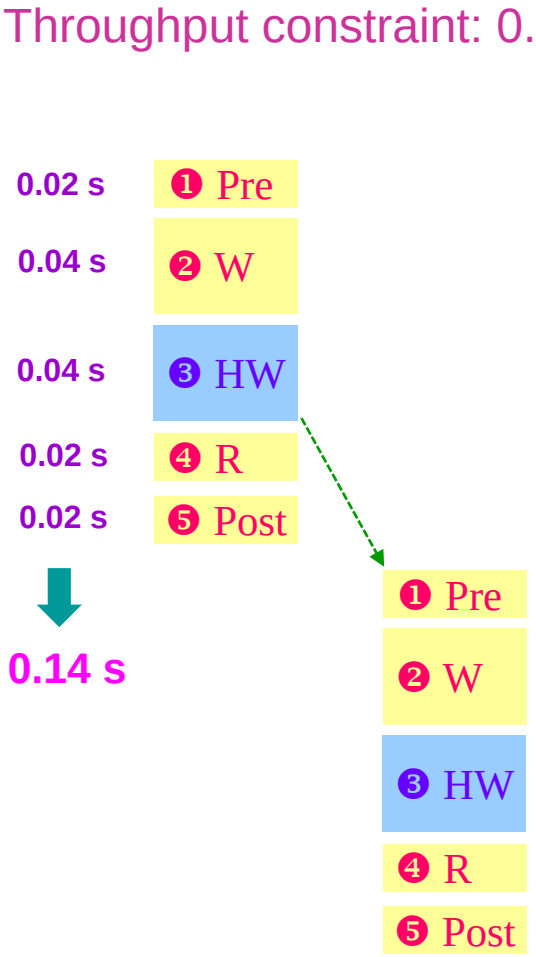
- **Estimate the execution time and memory size of each node in task graph G(V, E) when it is implemented by SW processor**
- **Instruction Set Simulator**
- **For example**
  - ARM Symbolic Debugger (armsd) simulator in ARM Developer Suite (ADS)

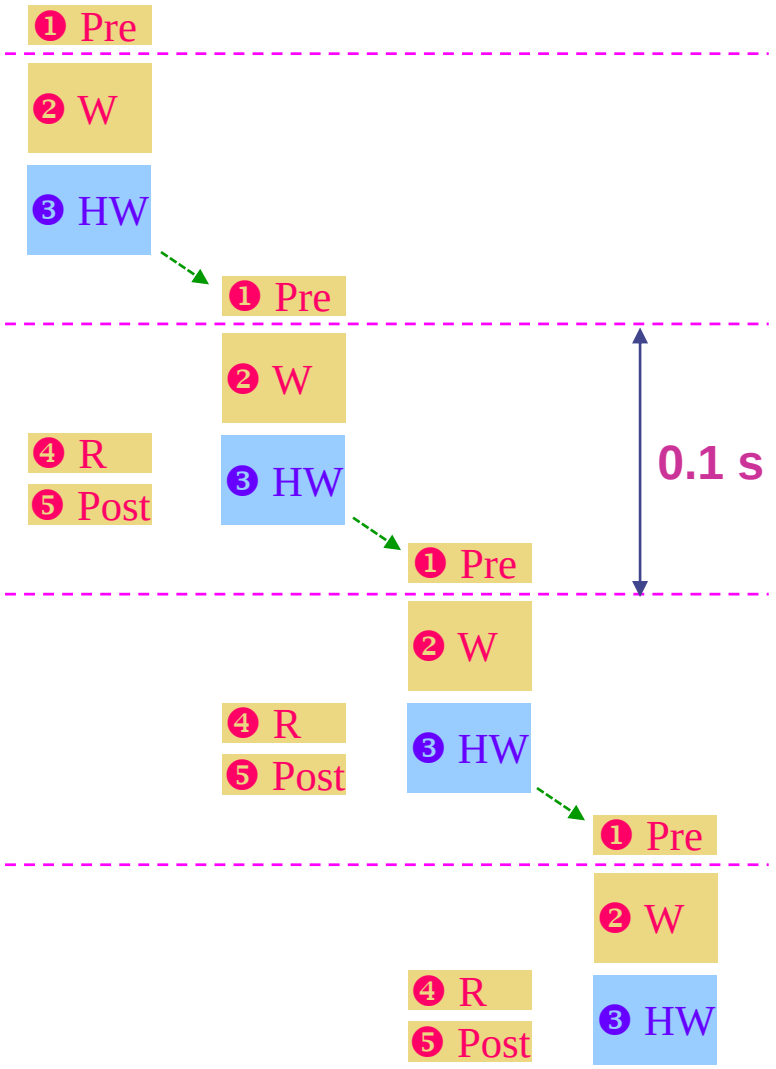
Hardware Estimation

- **Estimate the execution time and required area of every node in G(V, E) when it is implemented by hardware components**
- **Schedule the operations in each node of G(V, E) and bind them to available function units by using high-level synthesis**
- **For example**
  - Synopsys Behavioral Compiler with DesignWare library

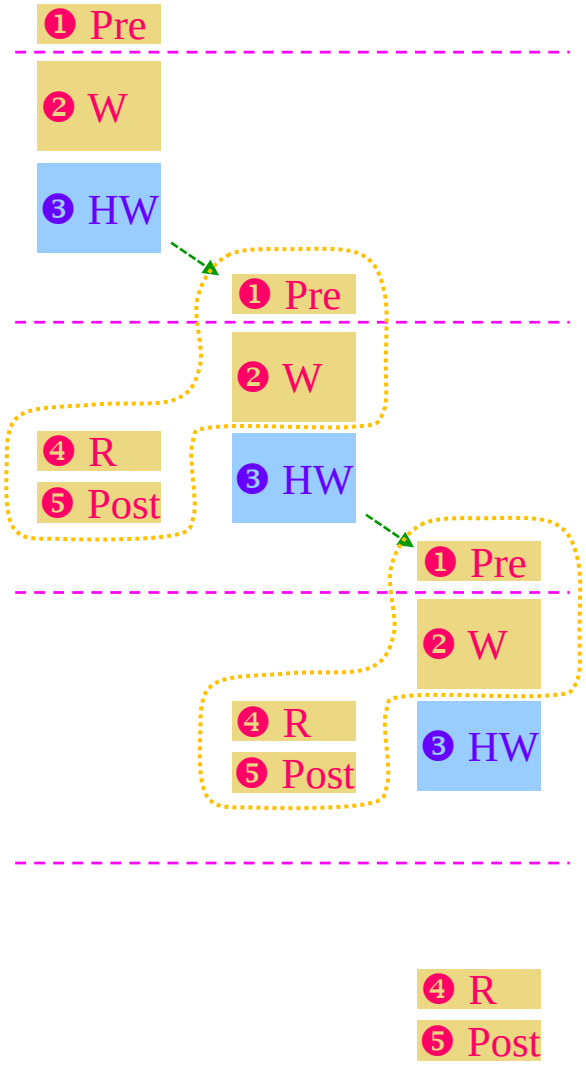
Pipelined Scheduling (1/2)

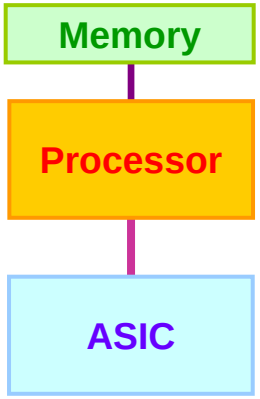
Throughput constraint: 0.1s

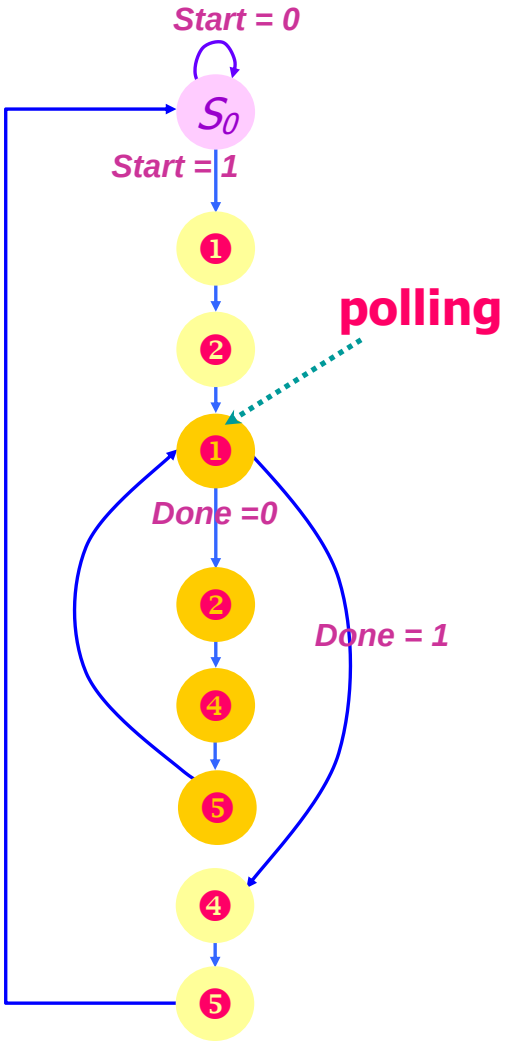




Pipelined Scheduling (2/2)



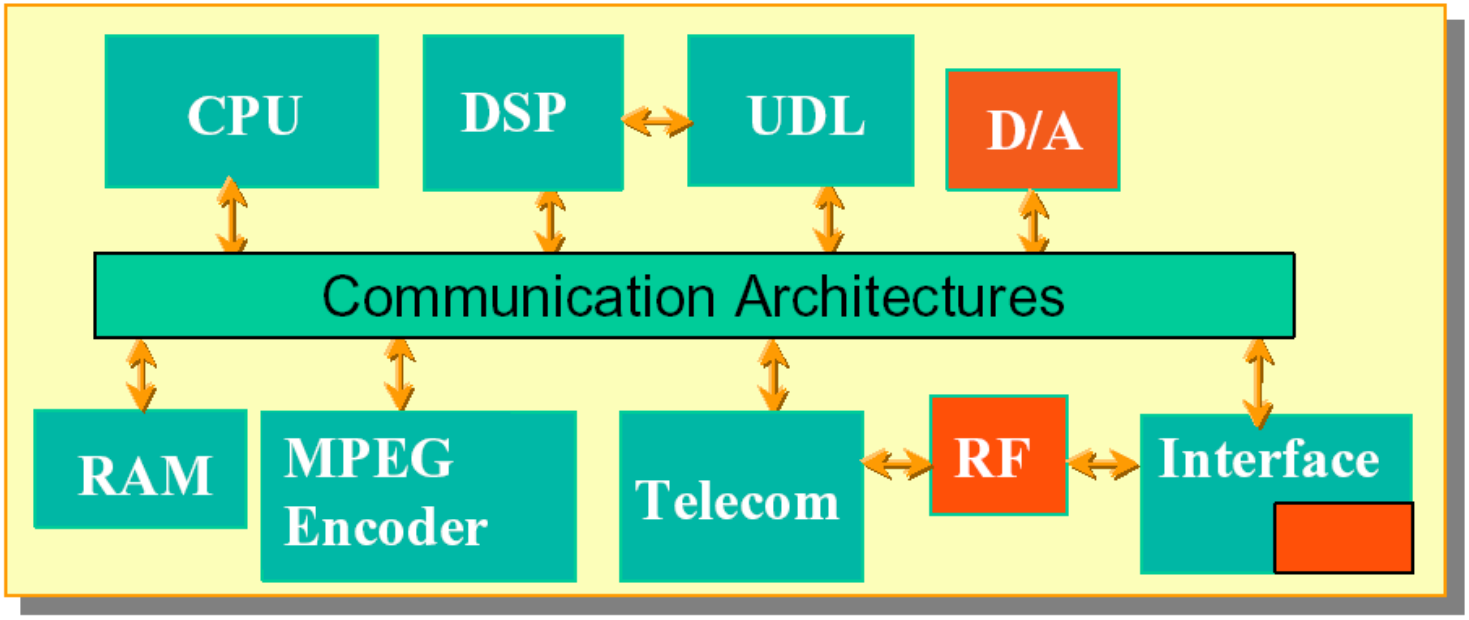




Communication Synthesis

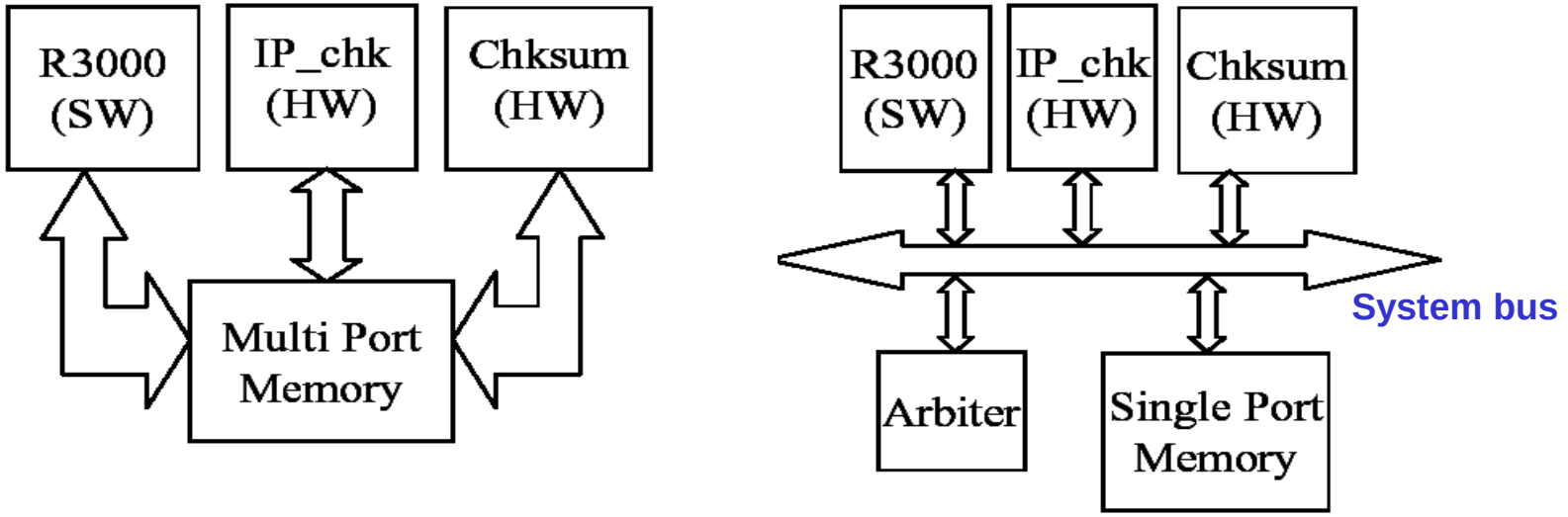
SoC Communication Architectures

- determine the way in which the components communicate with each other to synchronize and exchange data
- key to high performance
- bus-based architecture or network-on-a-chip (NoC)



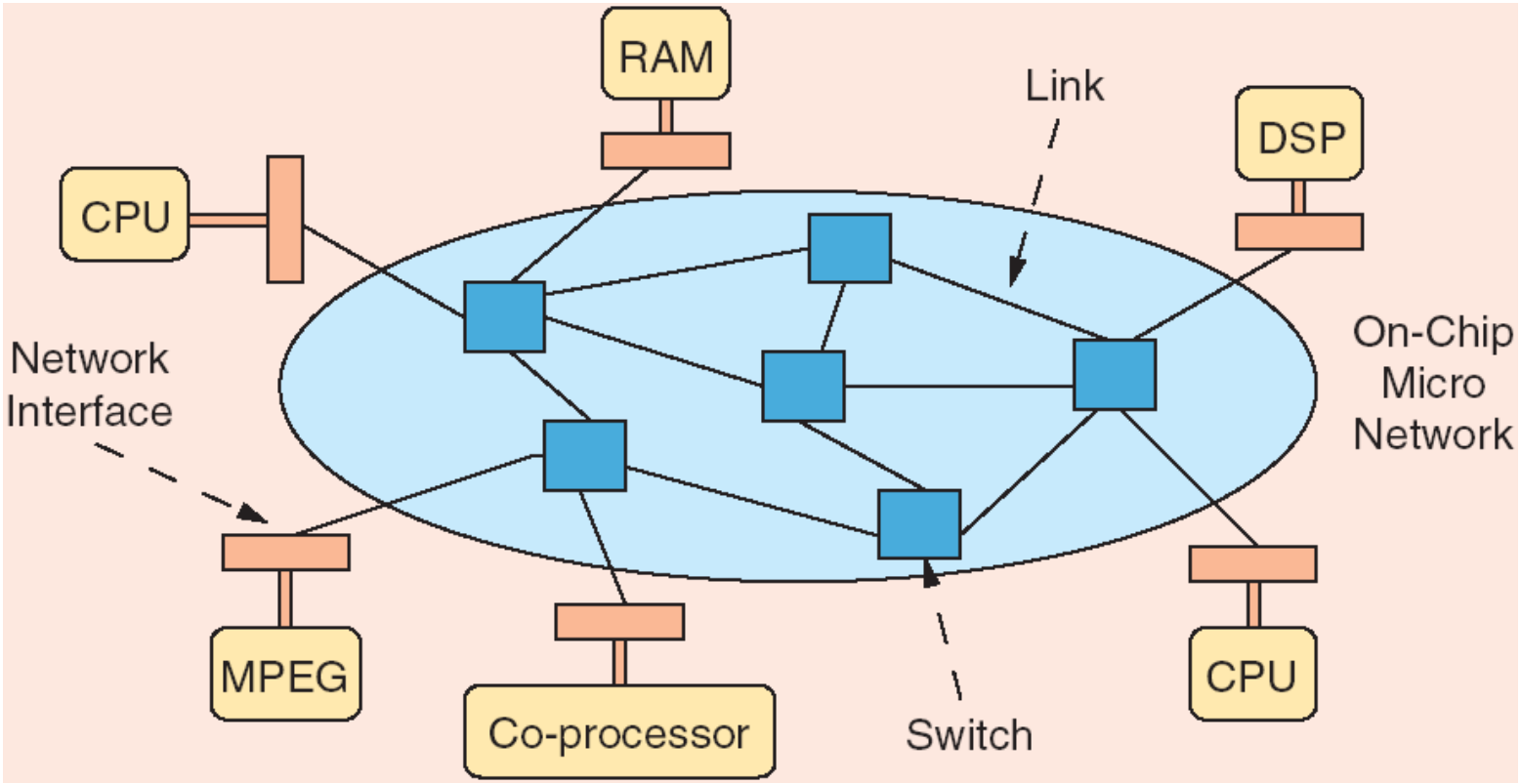
Bus-based Architecture

- **Point-to-Point or Shared Bus**

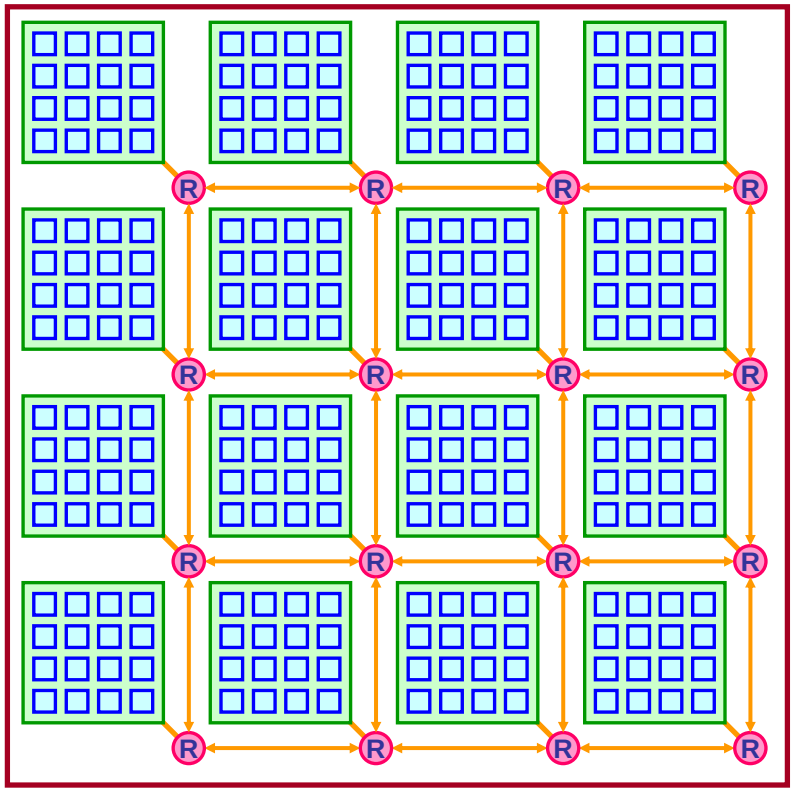


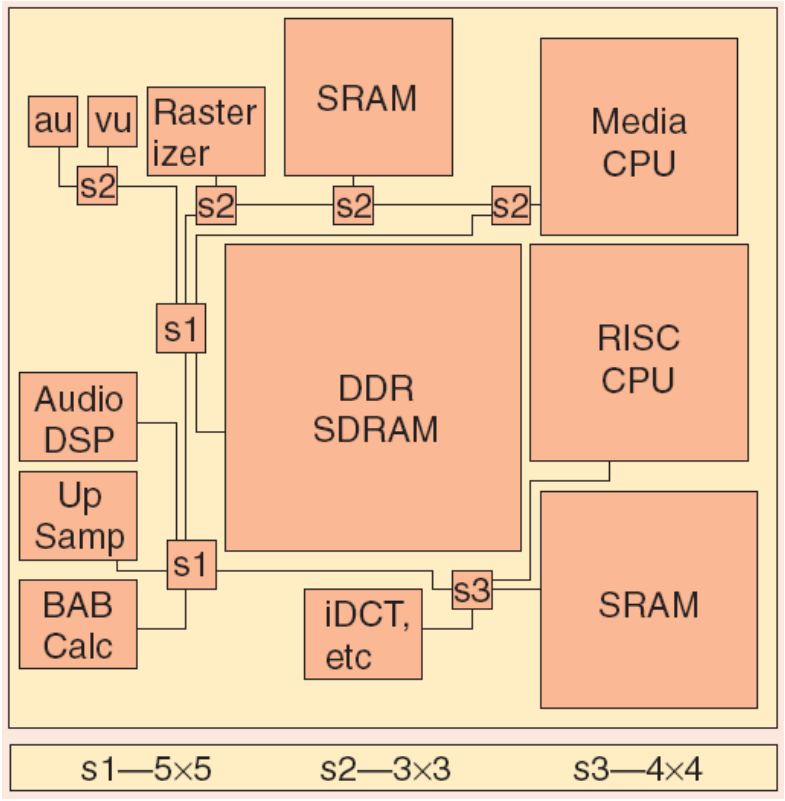
- **Bus Parameters**
  - bus width, arbitration protocols, support for DMA, DMA block size, bus speed, the number of bus layers, etc.
  - optimality of different architectures depends on the communication traffic generated by the components

Network-on-a-chip (NoC)



NoC Topology

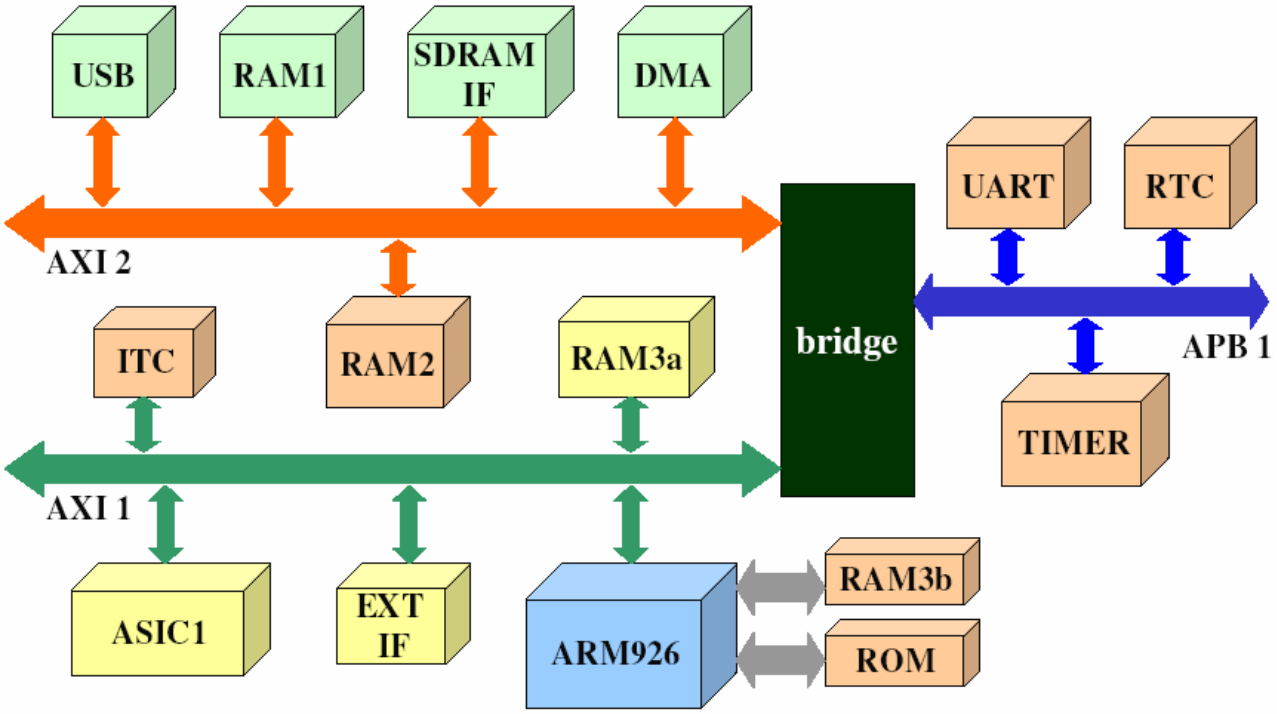




NoC vs. Bus

- **Packet-based**
  - no distinction address/data, only packets (but many types)
  - complete separation between end-to-end transactions and data delivery protocols
- **Distributed vs. centralized**
  - no global control bottleneck
  - better link with placement and routing
- **Bandwidth scalability**

Simulation-based Architecture Synthesis



Parameter	Values		
	AXI 1	AXI 2	APB
Bus width	32	32	32
Bus speed	66	133	33
arb scheme	static ASIC1>DMA>USB>ARM		
DMA size	16		

Outline

- SoC Design Flow
- System Architecture Development
- System-Level Verification
- Electronic System-Level Design
- Transaction-Level Model

System-Level Verification

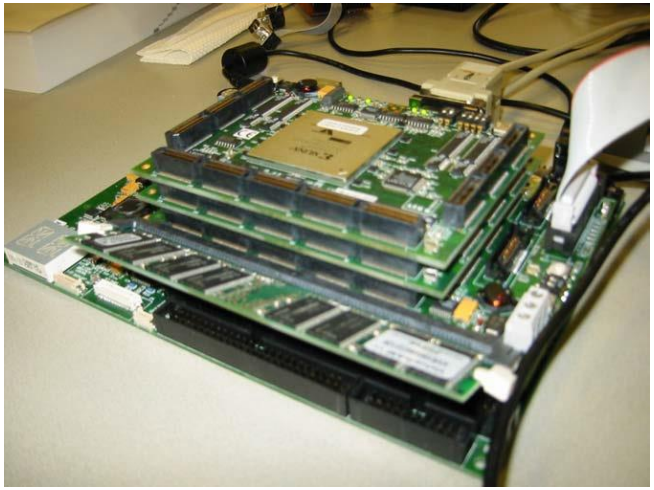
- **SoCs increase in complexity**
  - Embedded software + Heterogeneous components
  - Validation becomes more and more difficult, time consuming, and error prone
- **System-level verification**
  - Gain a reasonable certainty that the design is free from errors
  - Can be done by means of
    - Formal verification
    - Simulation
    - Emulation (hardware prototyping)

Hardware Prototyping

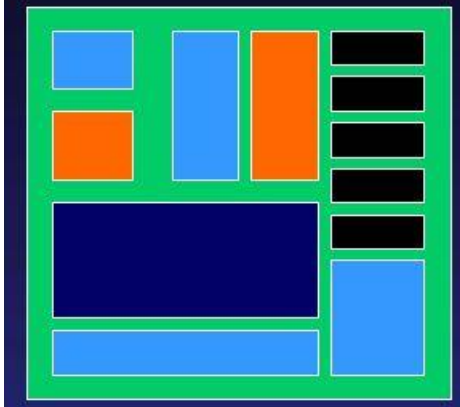
- **Required properties of SoC Development Platforms**
  - Flexibility and scalability $\Rightarrow$ reuse the platform
  - Enough software and hardware prototyping nodes
  - Communications prototyping network can be configured to perform various protocols (point-to-point or/and bus)
  - Enable simultaneously hardware and software development
    - An early software debug
  - Enable an accurate micro-architecture model as close as possible to the designer view
    - To obtain first-time silicon success

ARM-based SoC Development Platform

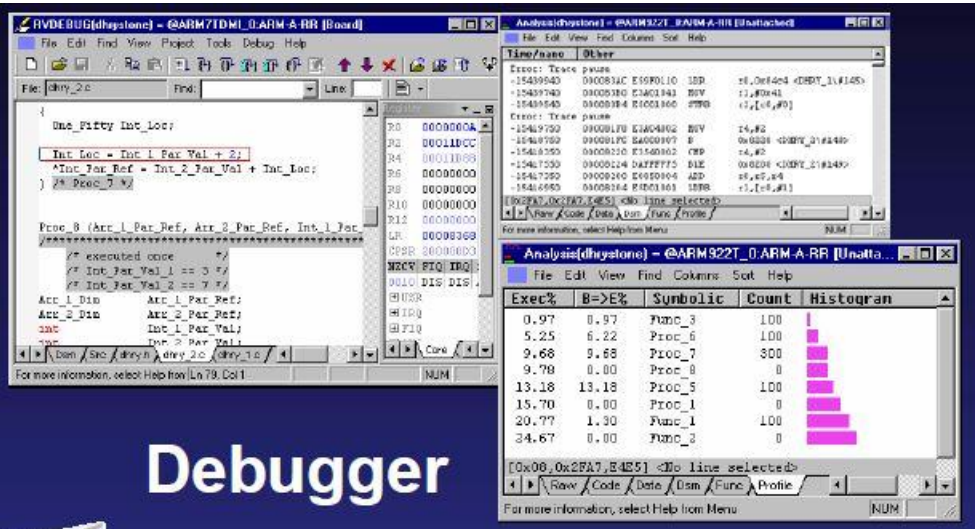
- **ARM-based System Development**
  - Processor Cores
  - On-Chip Bus: AMBA
  - System building blocks: PrimeCell
  - Development tools
    - Software development, Debug tools, Development kits
    - EDA models, Development boards



A System Solution



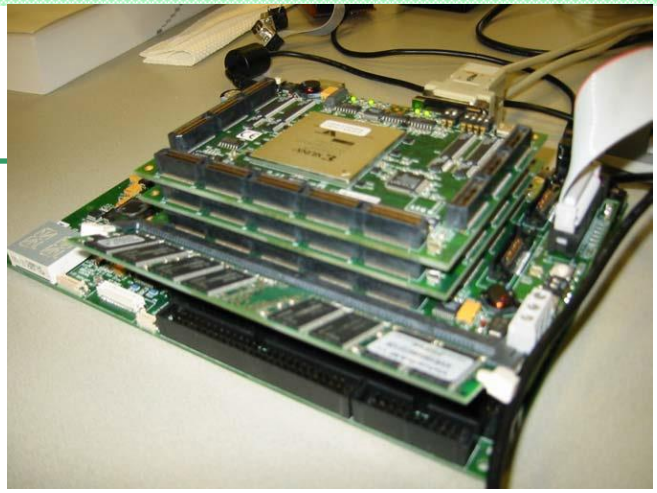
On-chip features

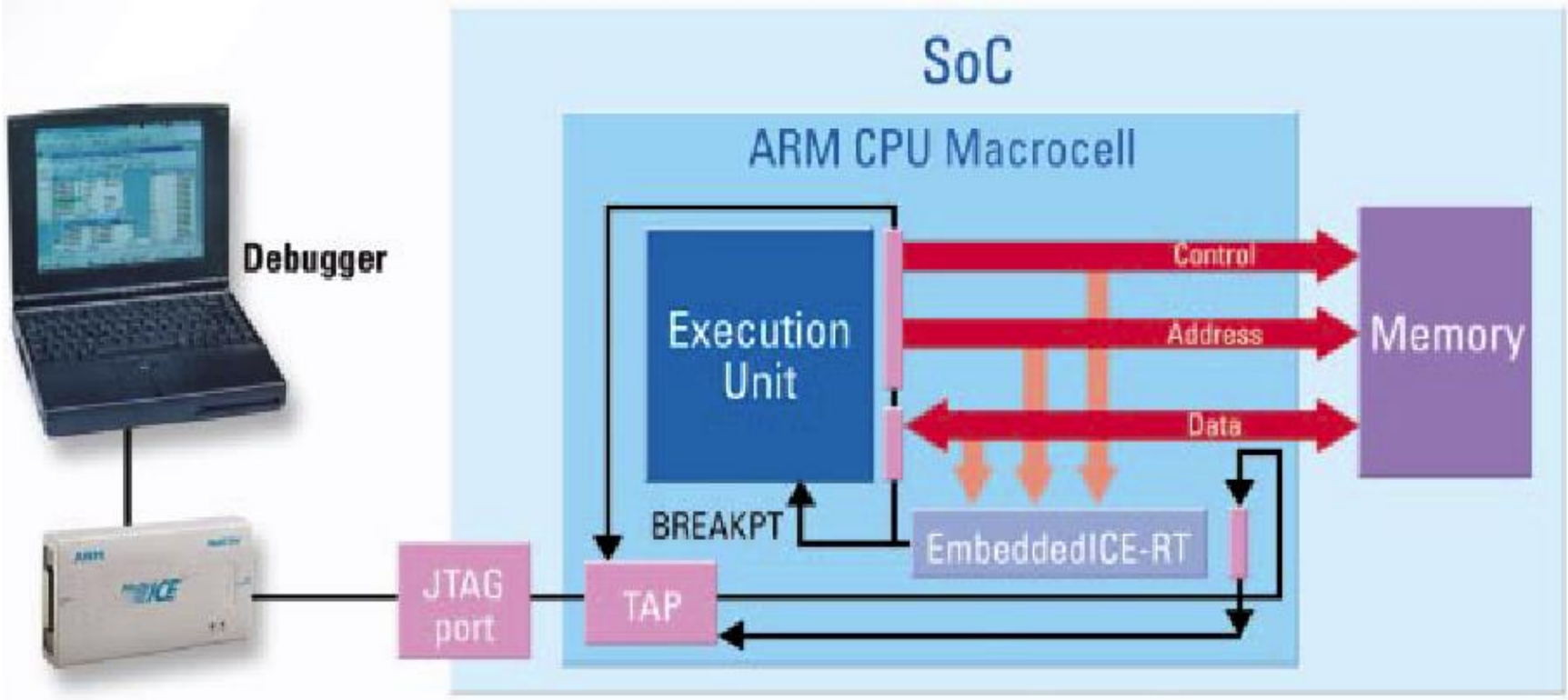


Debugger

JTAG Emulator
& Trace capture

Debugging with Multi-ICE





Simulation (1/2)

- **Simulation is traditional way of validating hardware correctness, by examining a set of output responses to input stimuli**
- **Simulation can't insure the correctness of the design**
- **Nevertheless, simulation tools are very useful for co-design**
- **A simulator should provide the following desirable features**
  - Adequate timing accuracy
  - Fast execution
  - Visibility of the internal registers

Simulation (2/2)

- **SystemC**
  - For system-level design and modeling
  - A library of C++ classes
  - C++ does not support
    - Hardware style communication, Concurrency, Reactivity, Notion of time, Hardware data types
  - SystemC 2.0
    - support generalized modeling for communication and synchronization with channels, interfaces, and events
    - support transaction-level modeling, communication refinement, executable specification modeling, HW/SW co-design

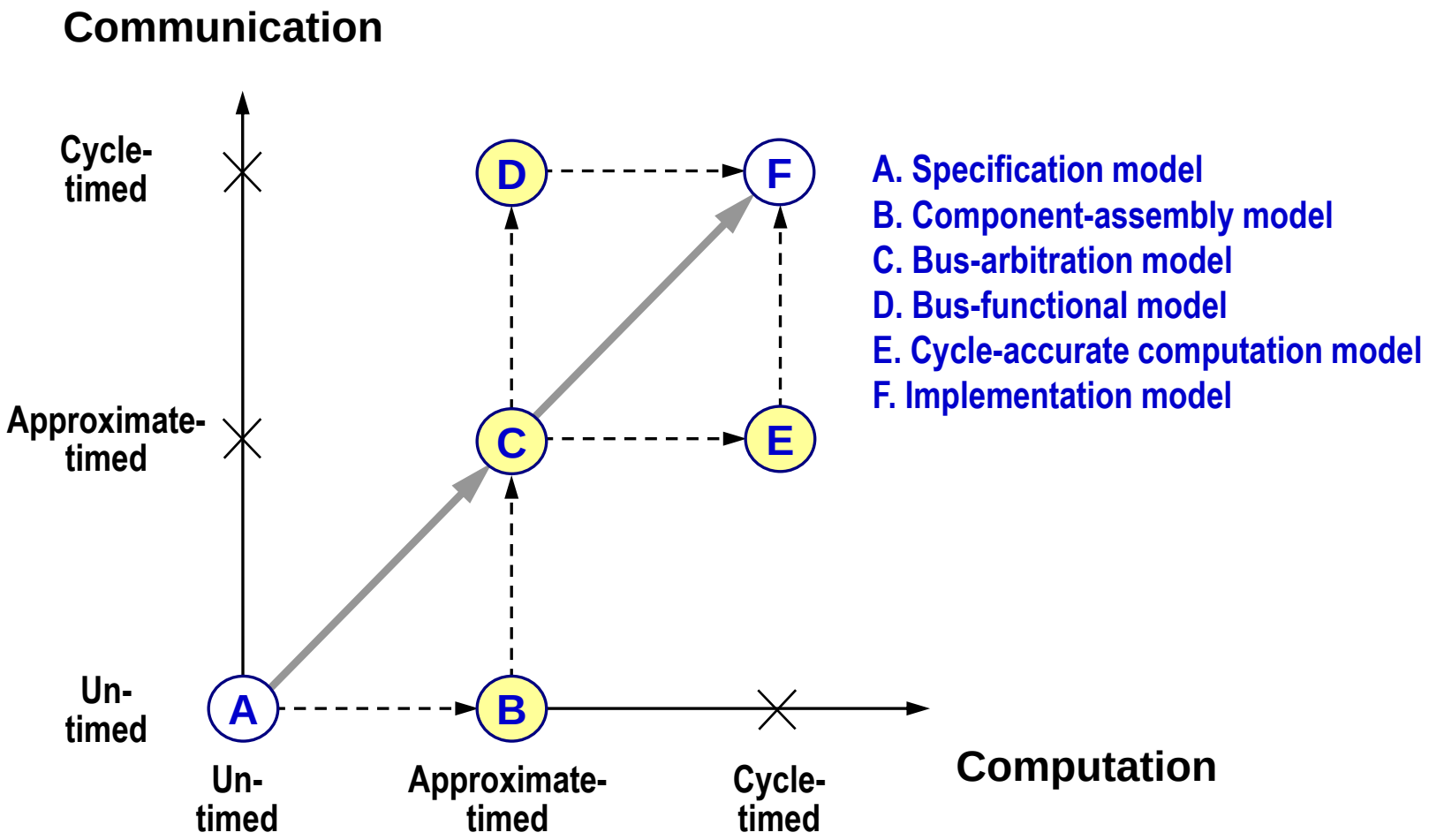
Transaction-level Modeling (1/3)

- **Transaction-level model (TLM)**
  - operates at the level of function calls and data packet transfers
  - provides SoC designers with a direct and clear view of system behavior
- **SystemC TLMs of silicon IP are easily integrated into the SoC architecture TLM**
  - this enables the SoC architect to rapidly explore and analyze multiple candidate hardware architectures and HW/SW partitioning schemes (each with different performance and economic trade-offs) to identify the optimum architecture

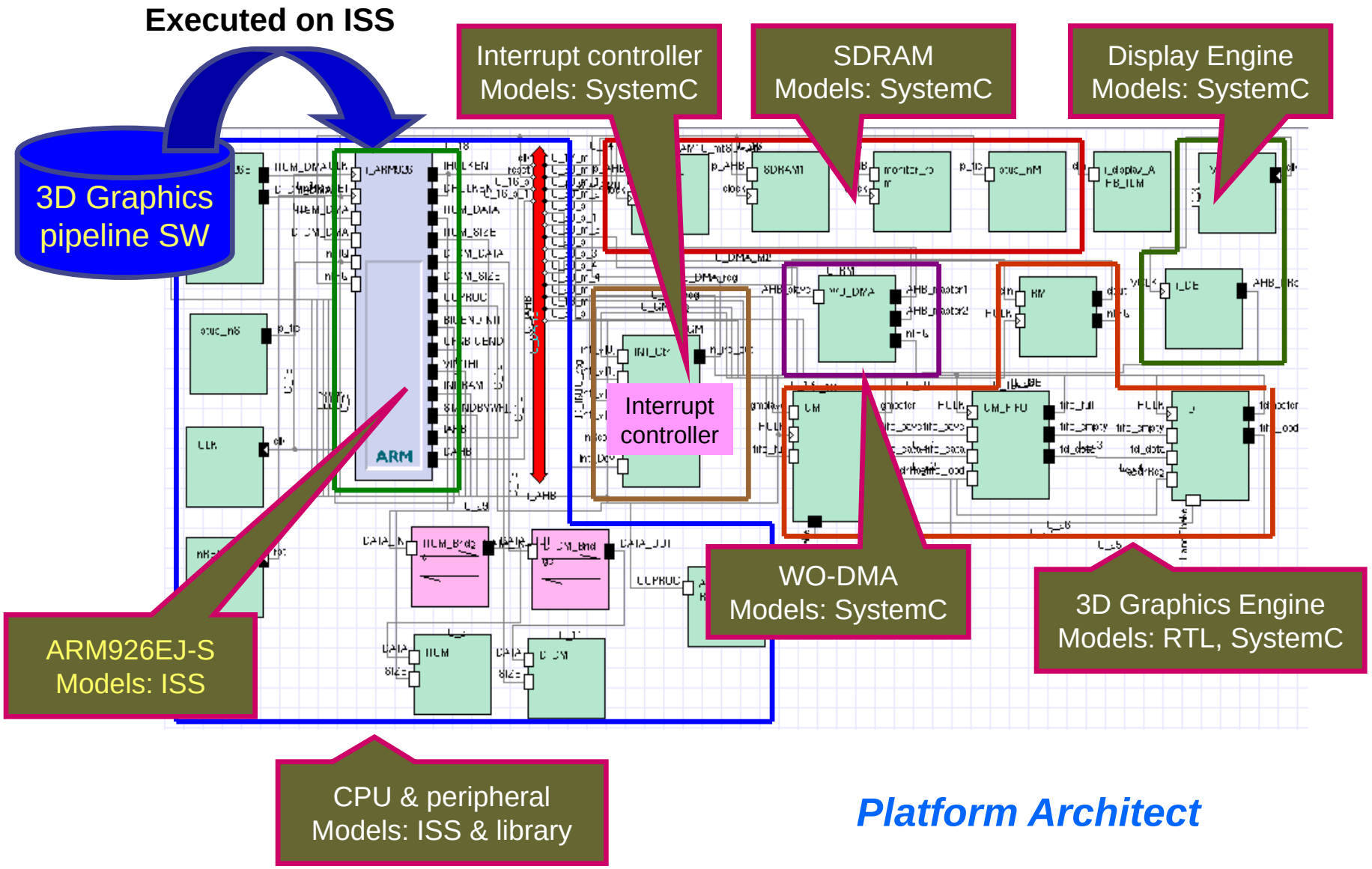
Transaction-level Modeling (2/3)

- **Register-transfer Level (RTL)**
  - intra-block circuit states, nanosecond-accurate transitions and bit-accurate bus behavior
- **Cycle-accurate TLM**
  - speeds up hardware verification and HW/SW co-verification by a factor of 1,000 or more over RTL
  - enables the generation of a functional testbench to verify system behavior and RTL implementation
  - supports the co-simulation of SystemC with RTL

Transaction-level Modeling (3/3)

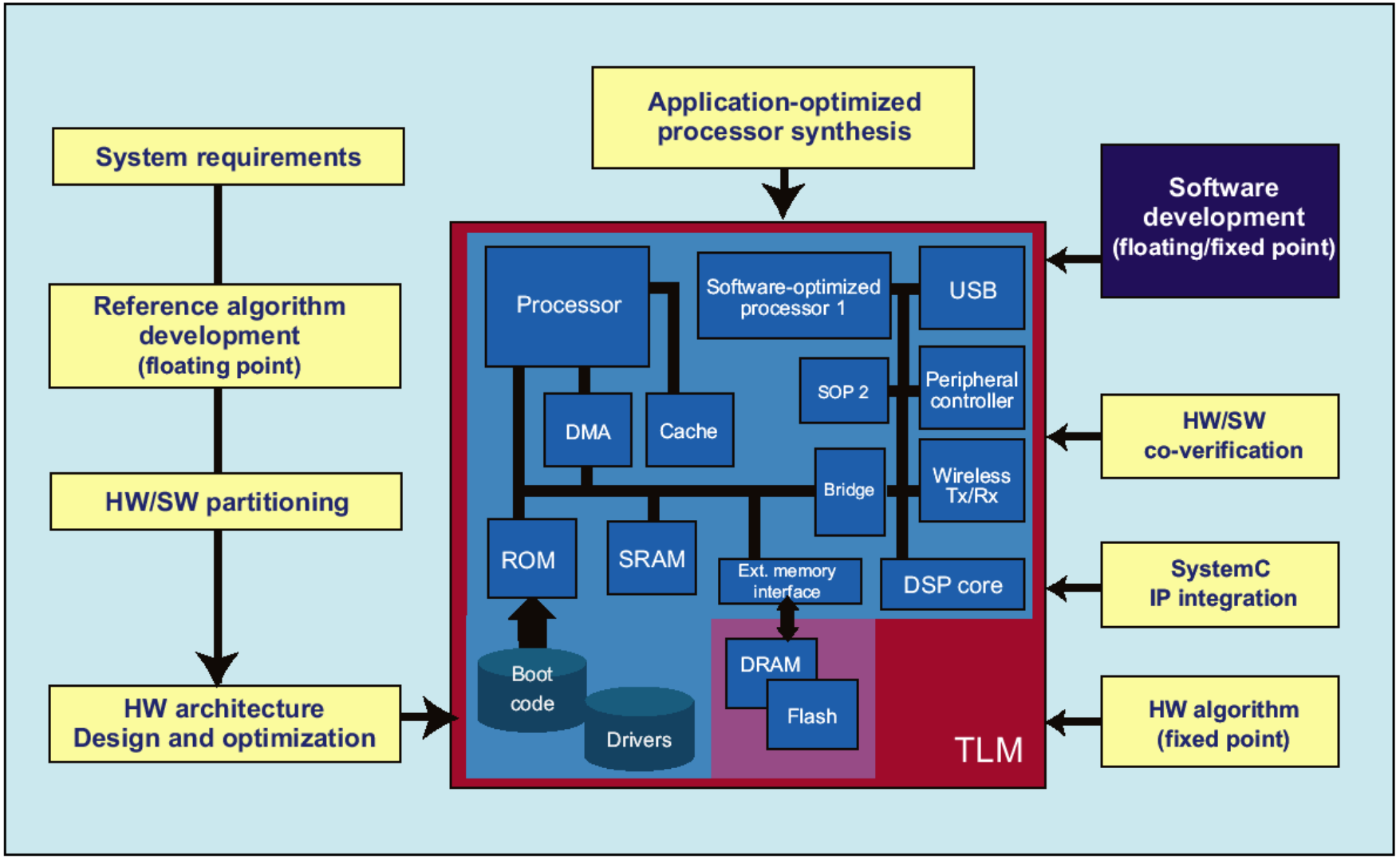


3D Graphics SoC



Platform Architect

TLM Methodology



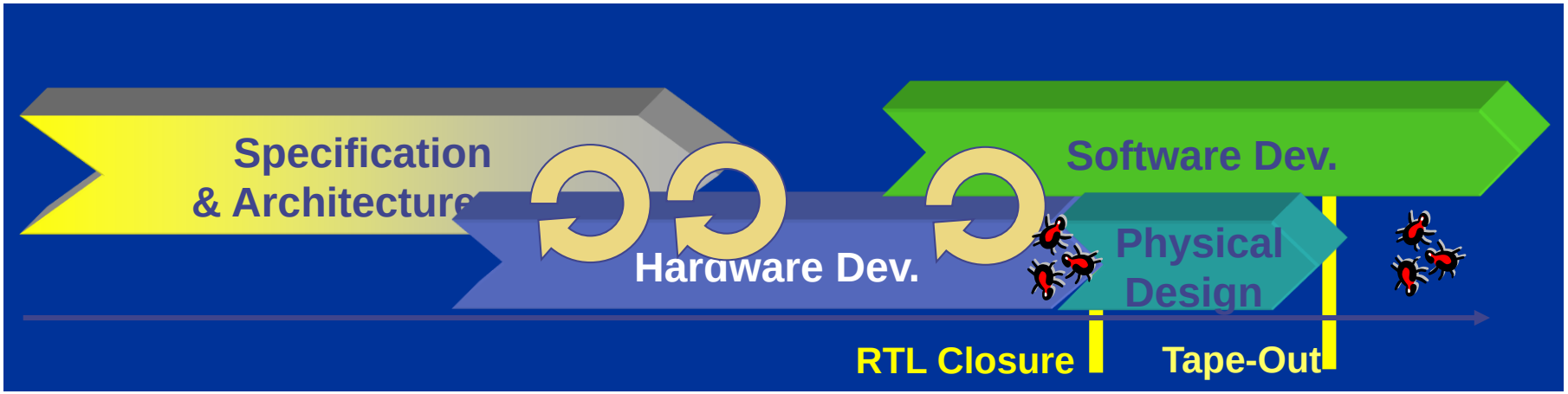
Source: CoWare Inc.

Outline

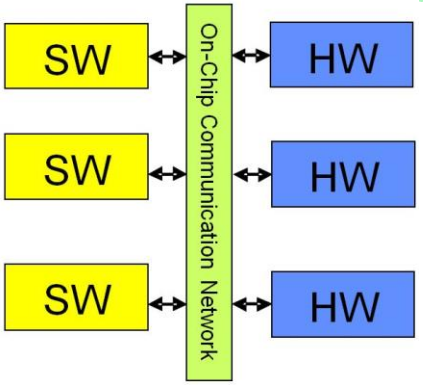
- SoC Design Flow
- System Architecture Development
- System-Level Verification
- Electronic System-Level Design

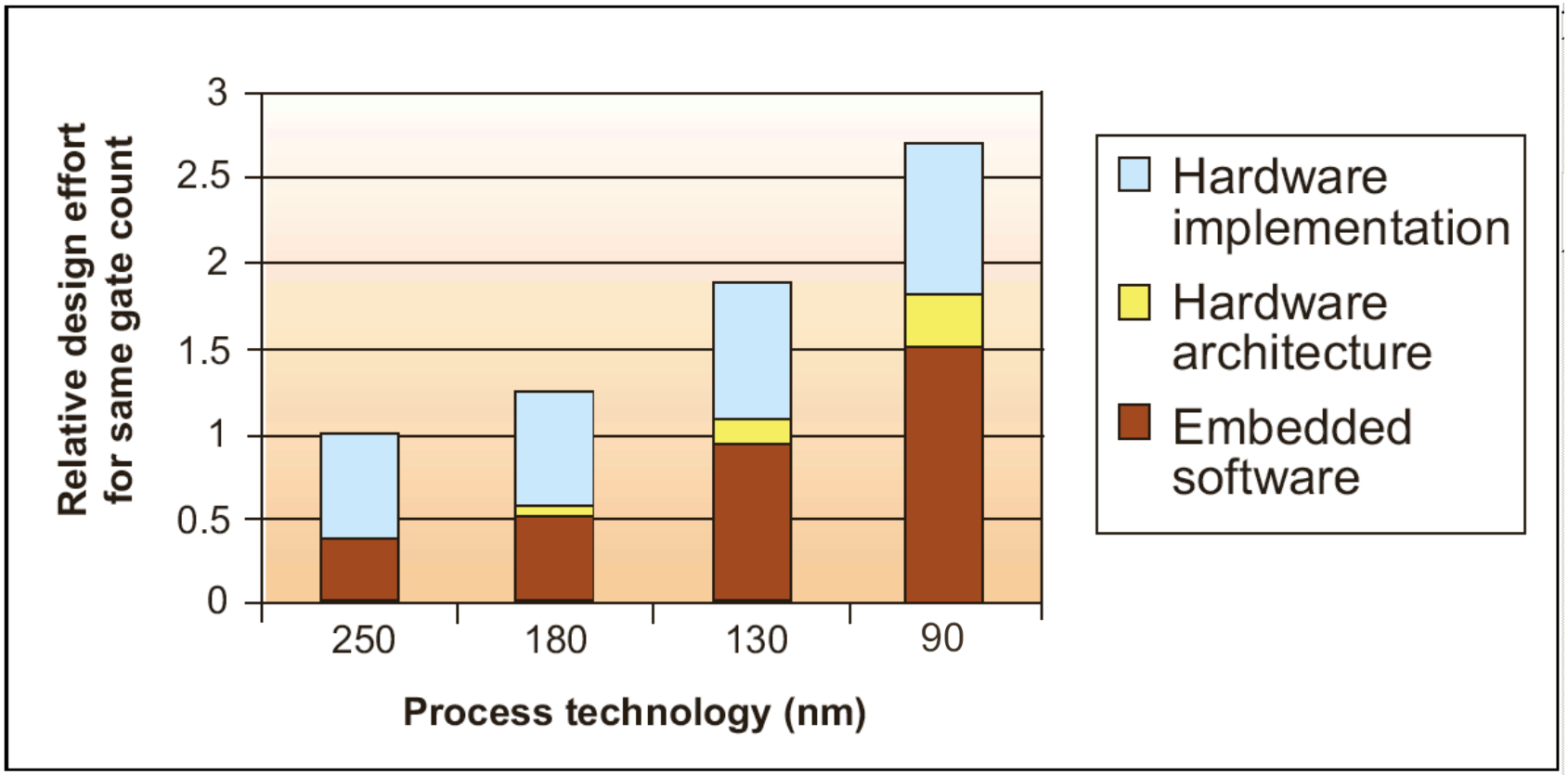
Conventional SoC Design Flow

- **Overlap in specification/architecture phase and RTL-design phase; multiple design changes**
  - Architecture design done informally
- **SW development starting late in the project**
- **Little SW simulation pre-silicon (takes too long)**



Relative Design Efforts

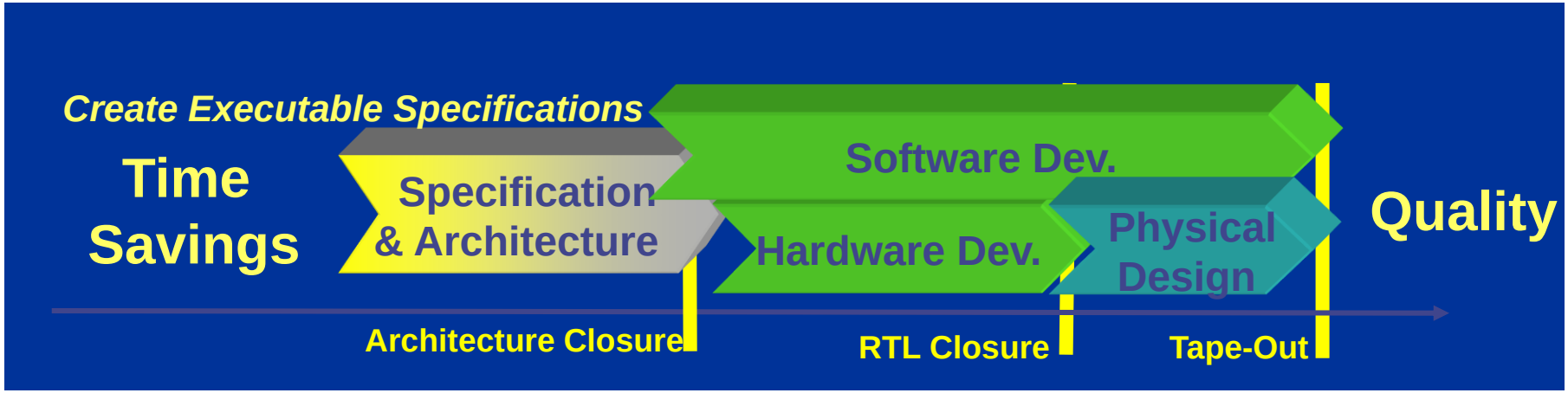




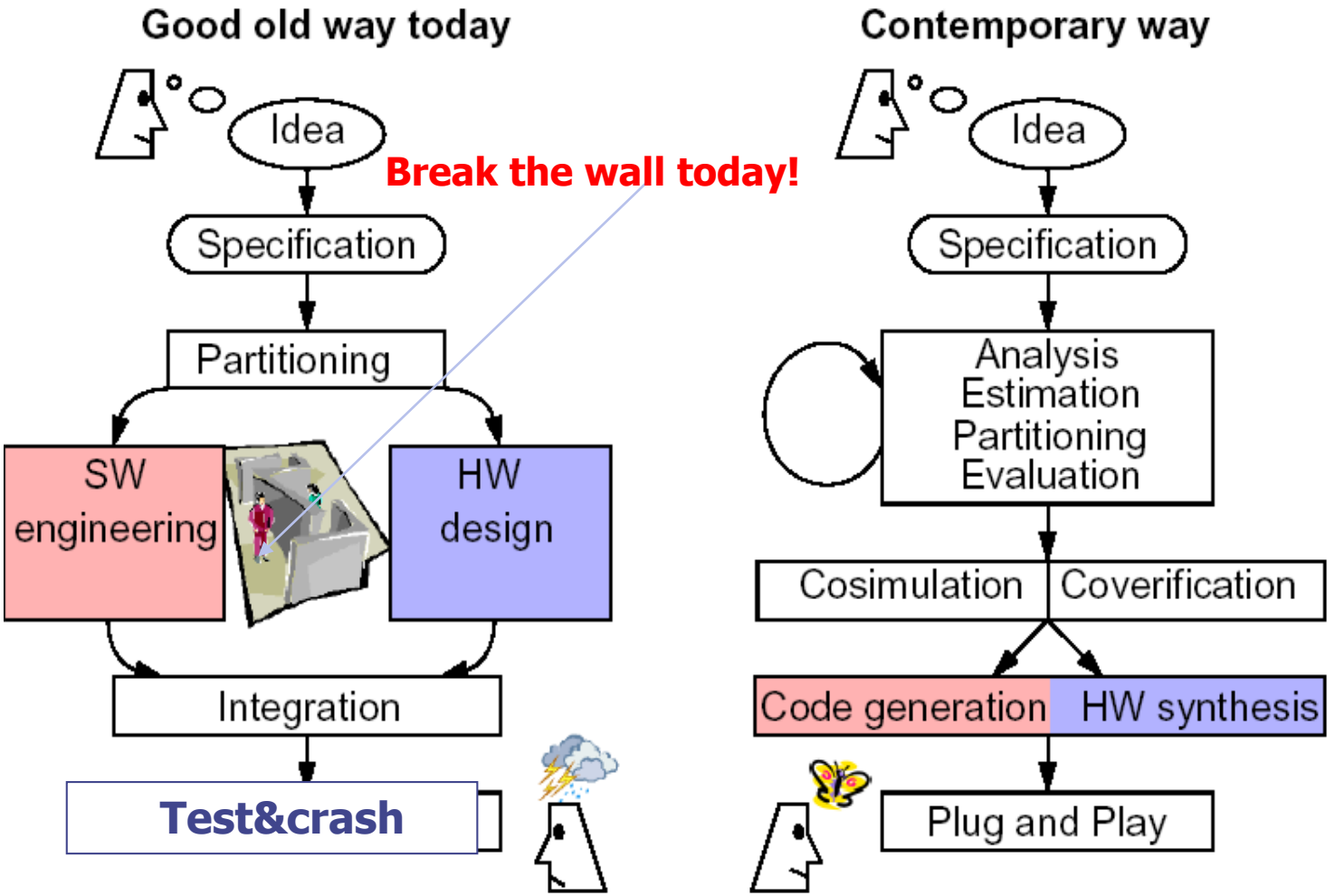
Source: International Business Strategies

Ideal SoC Design Methodology

- **Architecture closure**
  - Achieve a reduction # of RTL iterations
  - Can perform concurrent HW and SW design
  - Shorten the time it takes to get to golden RTL
- **Run SW on the architecture model**



HW/SW Co-Design & Co-Verification



[1]

Emerging SoC Design Flow (1/2)

- **The design methodologies developed for earlier SoC technology are inadequate to the task of designing a multiprocessor SoC**
  - Electronic system-level (ESL) design methodology has been devised to solve these problems
  - a set of methodologies that enables SoC engineers to efficiently develop, optimize and verify complex system architectures and embedded software
  - the foundation for the continuously verifying downstream register-transfer level (RTL) implementation and subsequent software development

Emerging SoC Design Flow (2/2)

- **Virtual Platform**
  - A SW model of a HW SoC platform
- **Enables...**
  - HW platform architecture exploration and optimization
  - SW development, debugging, and optimization
  - Concurrent HW/SW design (HW/SW codesign)
- **Requirements**
  - High simulation speed
  - Speed/accuracy trade-off
  - Flexibility
  - Usability for non-HW-experts

Success Factors for ESL Tools (1/2)

- **Quickly assemble virtual prototypes**
  - a variety of IP (processors, DSPs, etc.) and infrastructure options (bus configurations)
- **Run software on the virtual prototypes**
  - include operating systems and application software, and hence requires high performance
- **Analyze performance and perform power/size estimates**
  - generate alternative HW/SW partitions
  - optimize the architecture and the software

Success Factors for ESL Tools (2/2)

- **Reuse the testbench/stimulus environment**
  - for verification in the downstream RTL design flow
- **Generate/assemble the corresponding RTL**
  - designers or system architects define the configuration parameters for the IP and the on-chip buses or other infrastructure
  - automatically generate the corresponding RTL representation for the IP blocks, including the infrastructure
- **Create the RTL for the new logic blocks**
  - automatically map the new logic blocks into RTL