HW/SW Codesign

May 2001

Axel Jantsch
Royal Institute of Technology

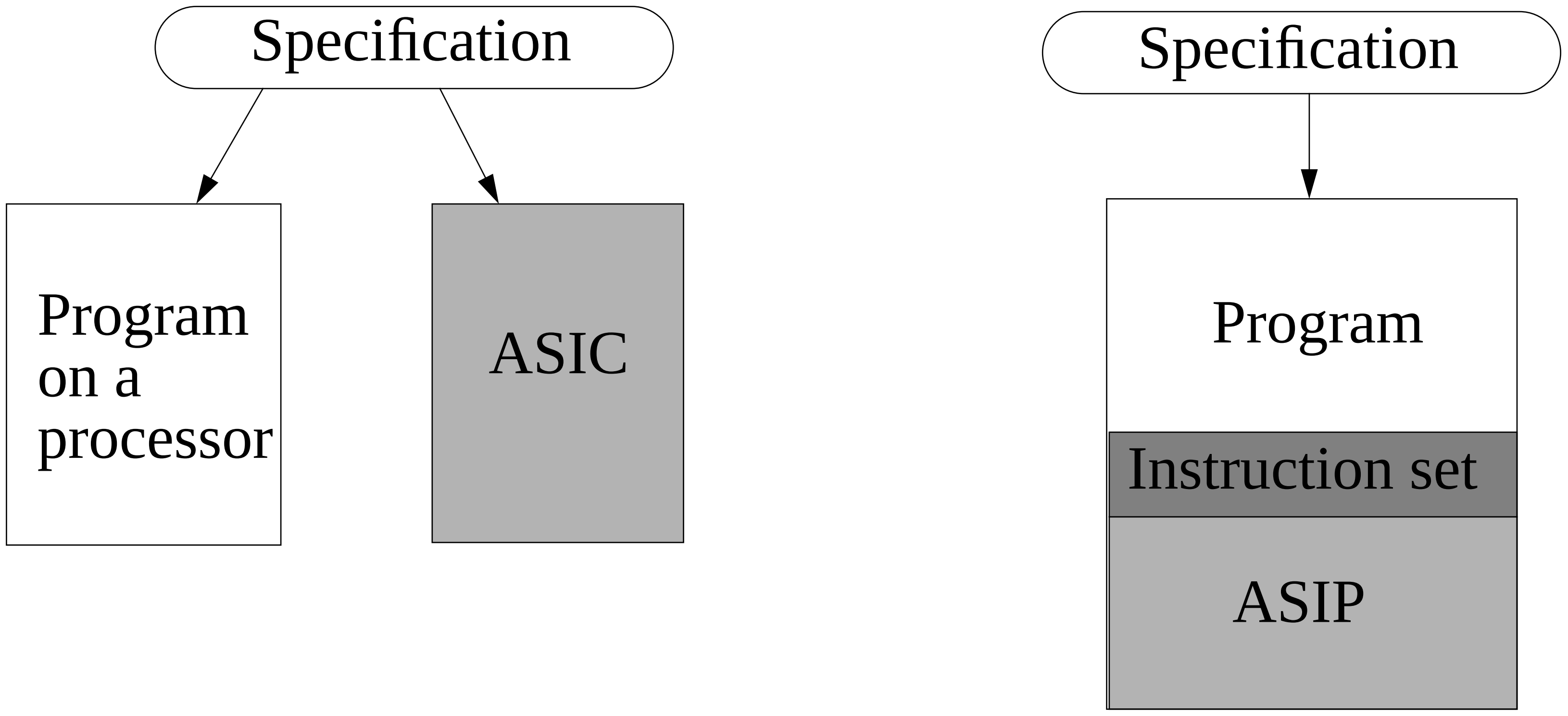
Overview

- Introduction
  - Types of Codesign
  - Main issues and challanges
- Methodology
- HW/SW Cosimulation

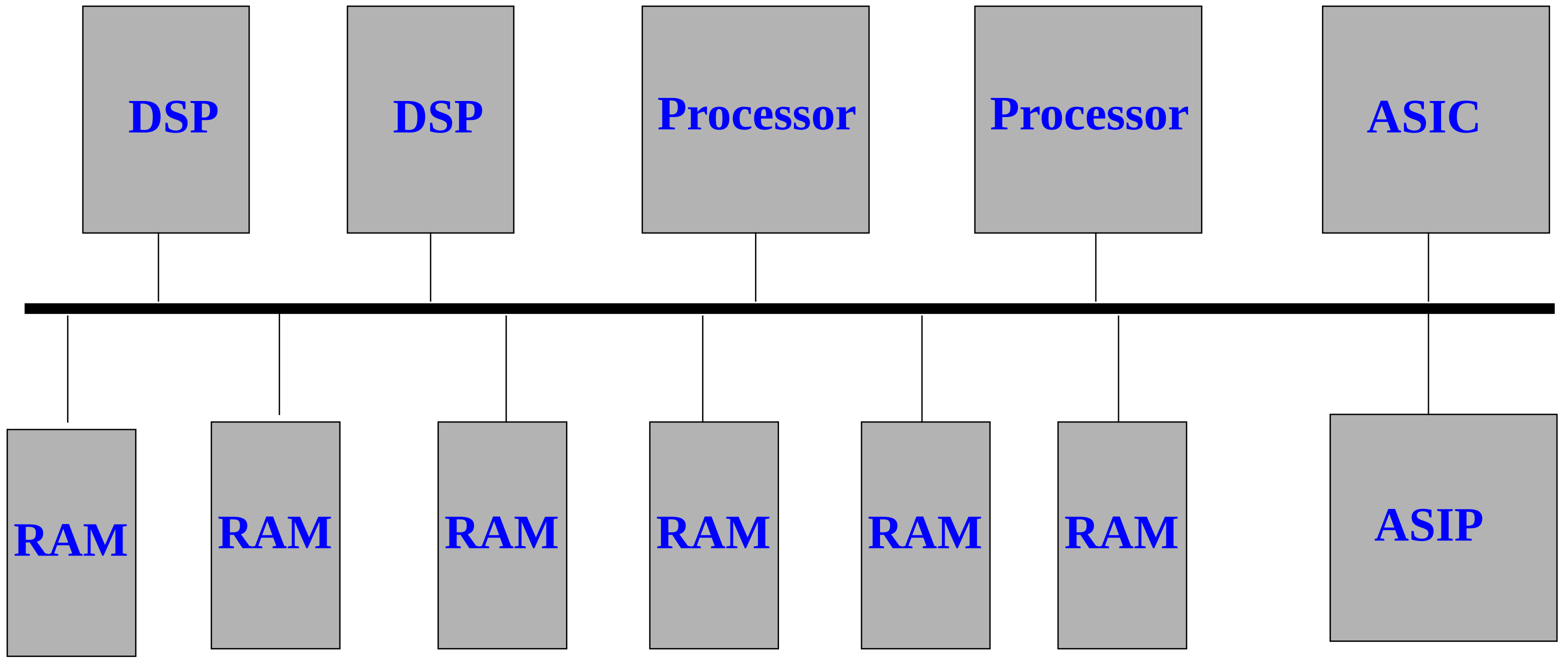
Introduction

- ☆ Vertical and Horizontal Codesign
- ☆ Figures of merits
  - ✣ Favorable for HW
  - ✣ Favorable for SW
- ☆ Intellectual Property (IP) blocks and Reuse
- ☆ Integration of HW design and SW design

Vertical vs. Horizontal Codesign



Vertical vs. Horizontal Codesign



Interfaces between HW and SW

- Instruction set
  - Standard instruction sets
  - Application specific instruction sets
- Bus protocol
- Device drivers
- FPGA configuration file
- Configuration of multiple concurrent resources
- Compilers
- Operating system

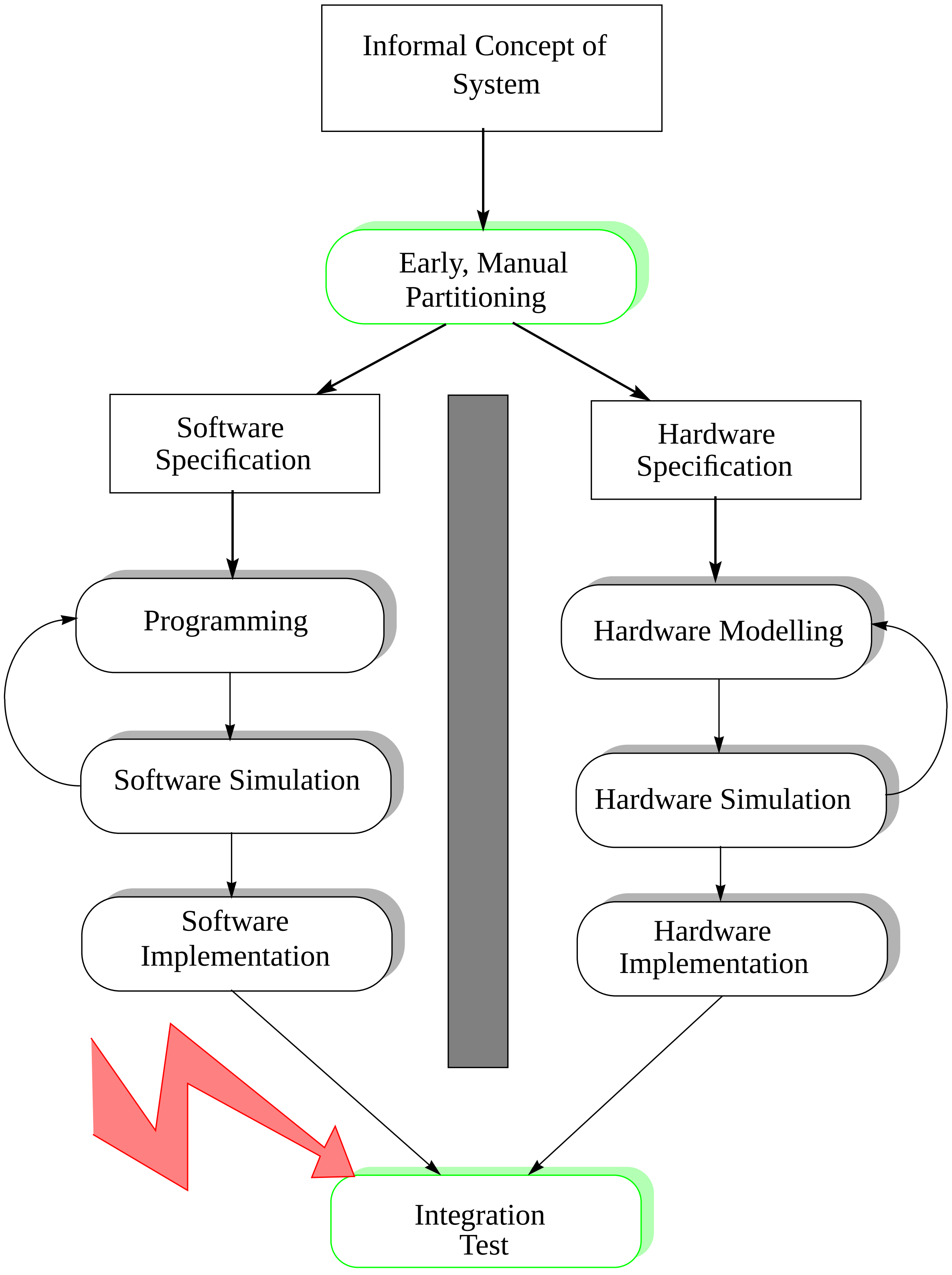
Figures of Merit

- Favorable for HW:
  - Delay
  - Throughput
  - Real-time systems
  - Power
  - Size
- Favorable for SW:
  - Time to market
  - Flexibility
  - Volume
- Others
  - Cost across products and product families - platform
  - Design productivity
  - Available tools and methodologies

IP and Reuse

- Integration of IP blocks
  - Development environment and tools
  - Interfaces: HW interfaces, device drivers
  - Testing
  - Business model and legal issues
- IP Blocks:
  - Processors
  - DSPs
  - Protocol implementations
  - Encryption/decryption blocks
  - I/O devices
  - Operating systems

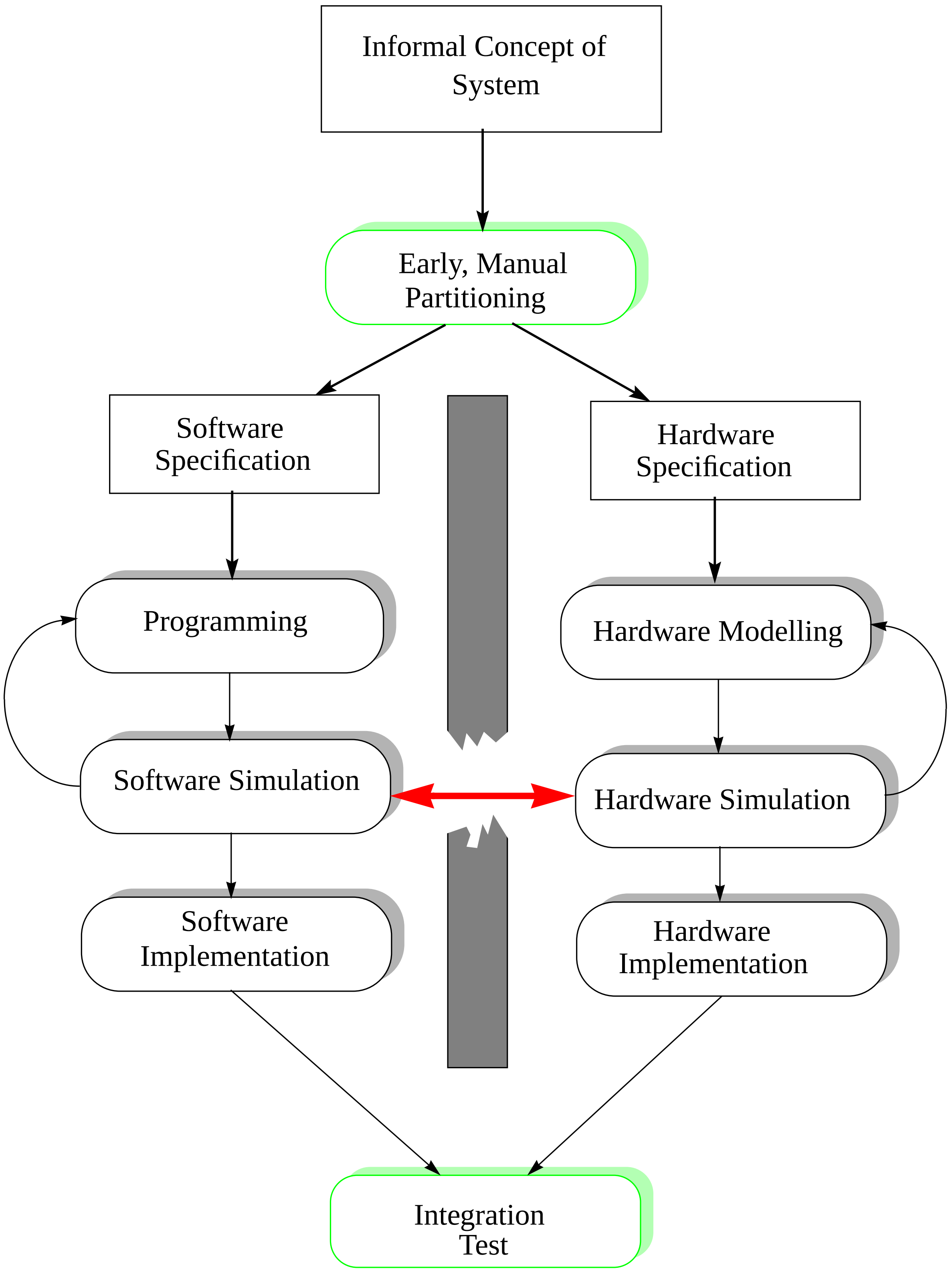
Traditional System Development



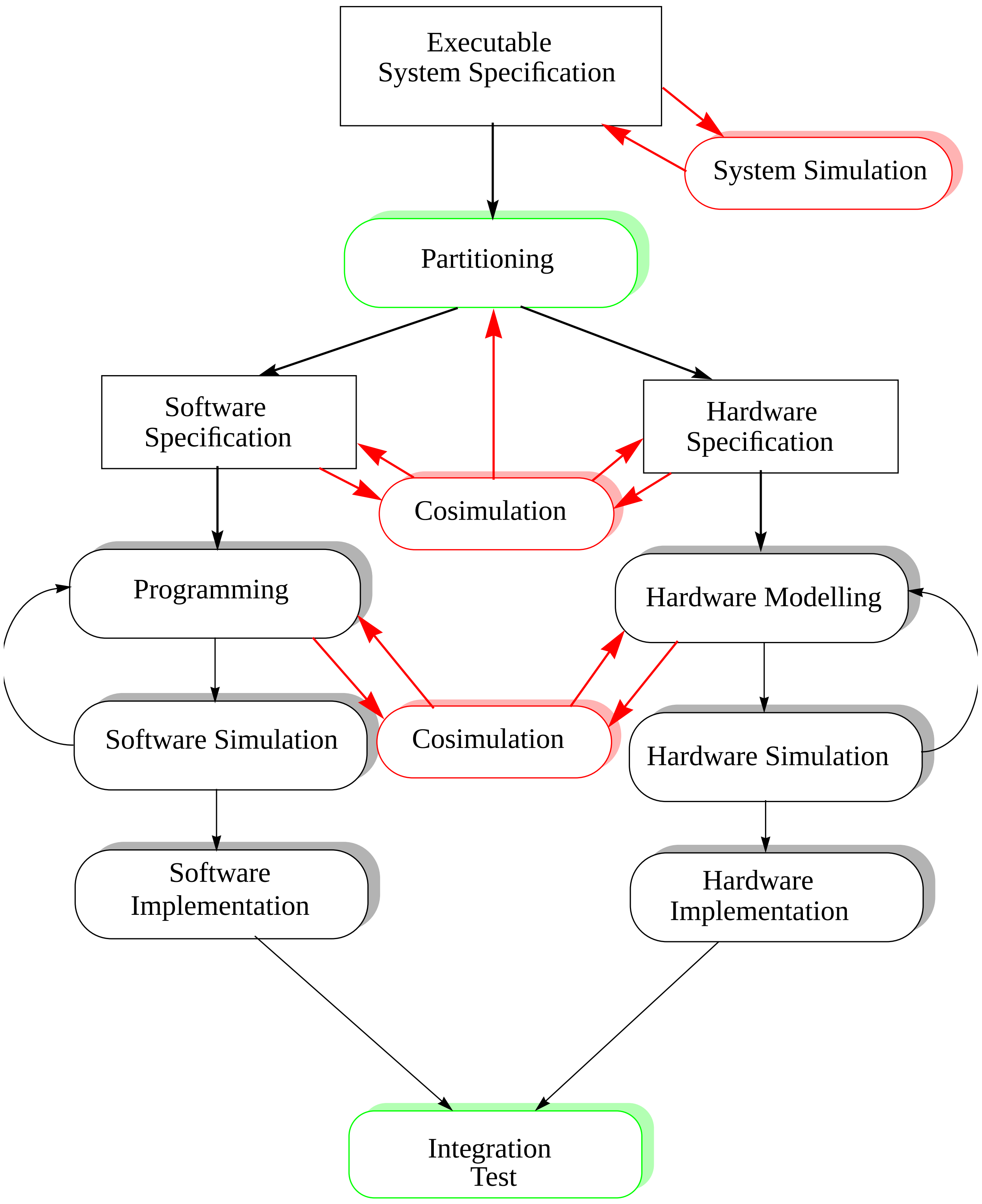
Problems with Traditional System Development

- ☆ Manual partitioning based on a ambiguous and incomplete system specification.
- ☆ Hardware and software development is conducted without sufficient exchange of information.
- ☆ Errors that are detected in the integration phase can be very costly.
- ☆ The cost of correcting errors is an exponential function of the time between error generation and error detection.
- ☆ If possible, integration problems are corrected in the software which reduces maintainability and reusability.
- ☆ Because of this danger the hardware is often over-equipped with features that are never used.

Breaking a Hole



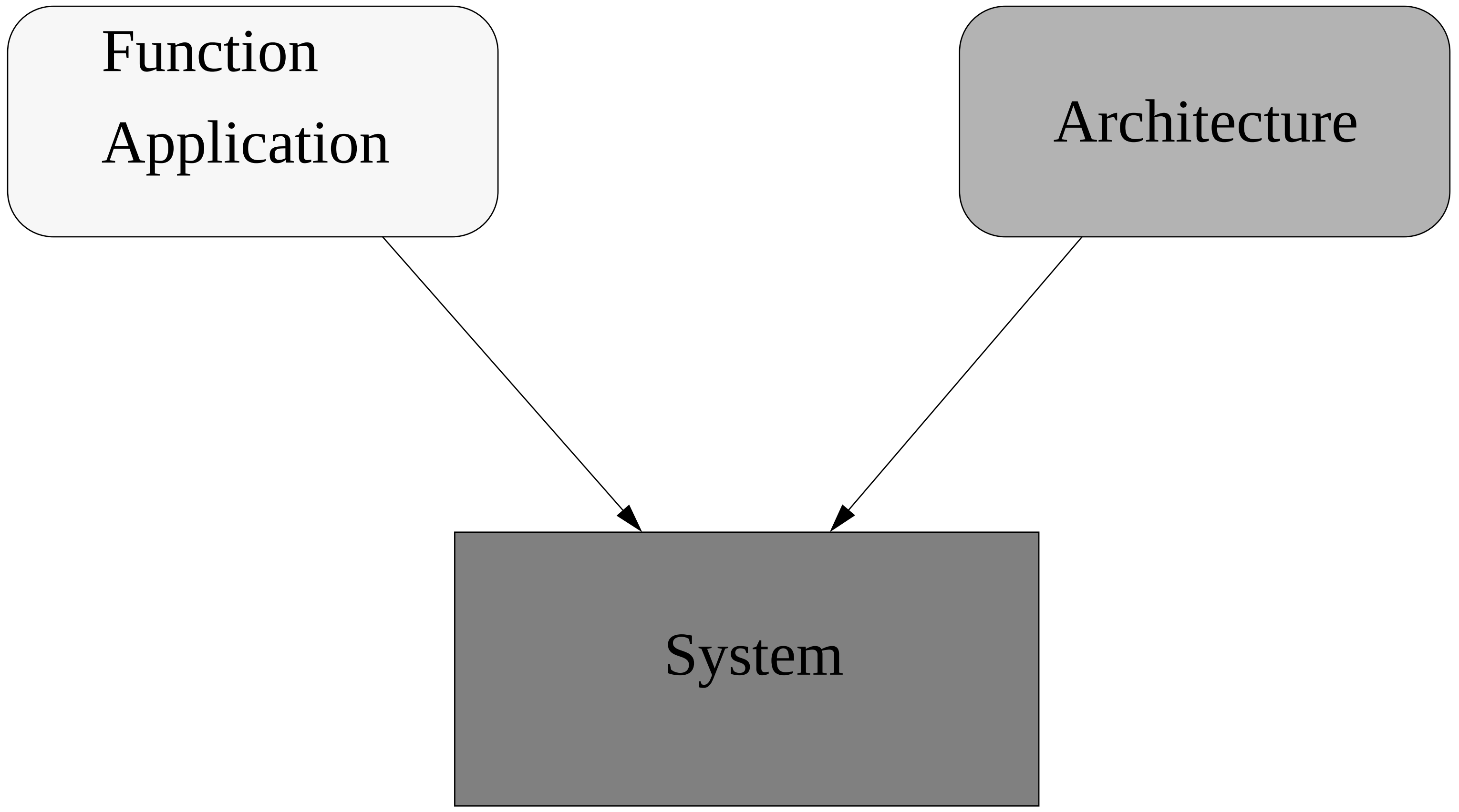
Hardware-Software Codesign



Advantages

- ☆ Specification is more stable.
- ☆ Systematic partitioning is less dependent on intuition.
- ☆ Simulation results from system simulation serve as reference.
- ☆ Avoid multiple test-benches
- ☆ Errors are detected earlier
- ☆ Development time is shorter
- ☆ Reusability and maintainability is higher

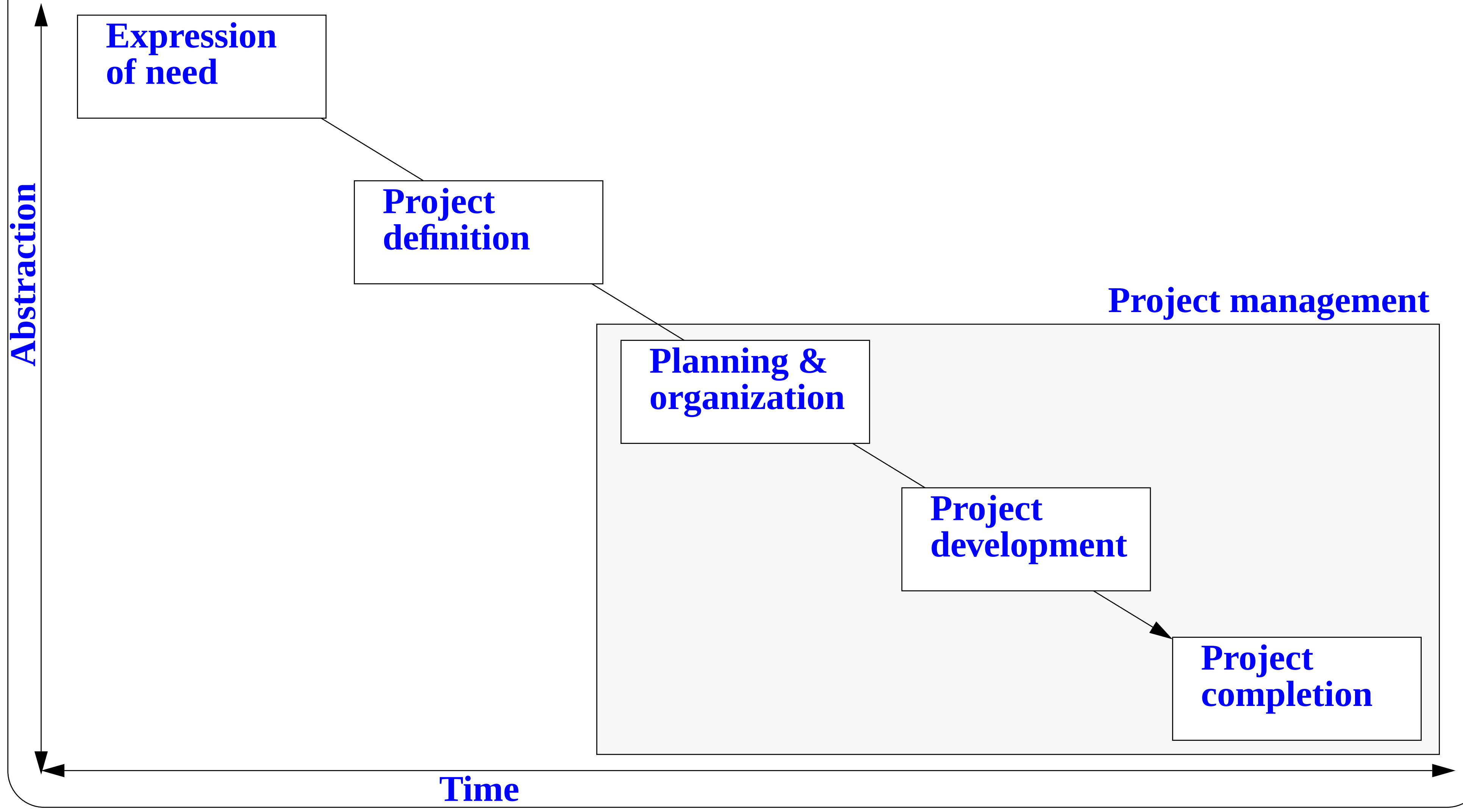
Function - Architecture Codesign



Methodology

- ☆ Development phases
- ☆ Life cycle models
- ☆ Effort distribution in different phases

Project Phases



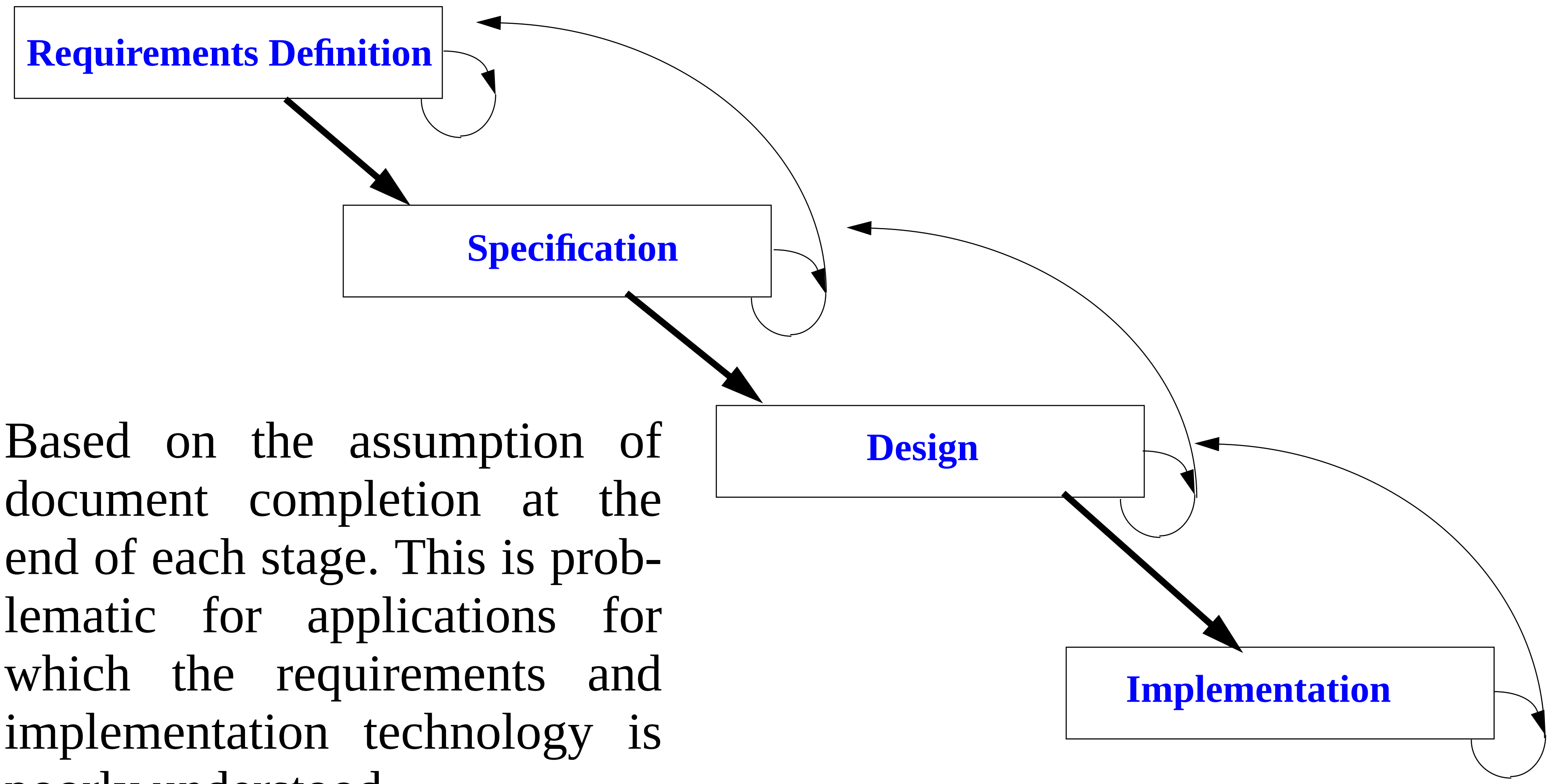
Involved People

- Customers
  - Users
  - Marketing and sales personnel
  - Operators
  - Maintenance personnel
- Product manager
- Project manager
- Requirement definition engineer
- Specification engineer
- Designer
- Implementation engineer
- Test engineer

Life Cycle Models

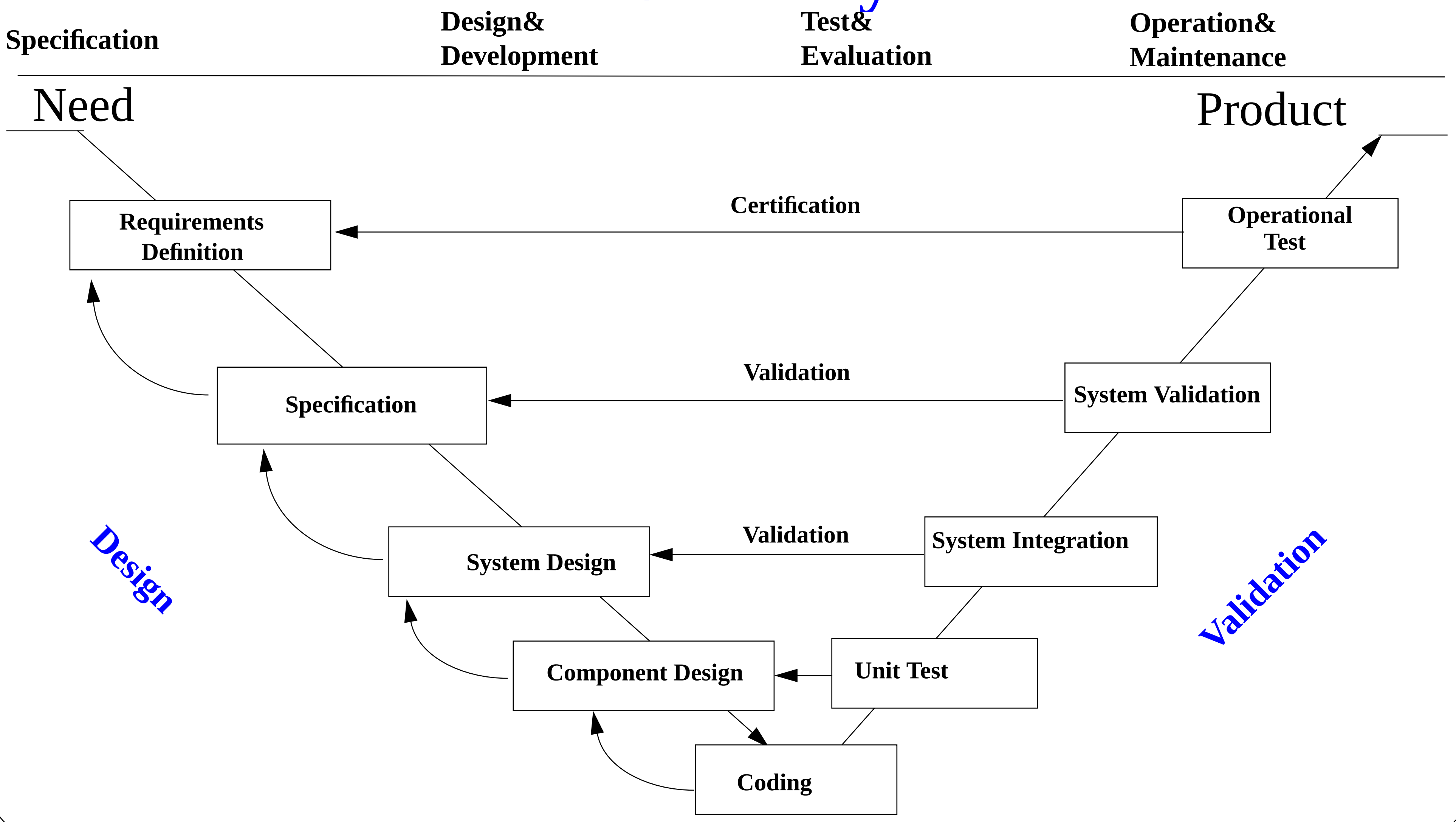
- ☆ Waterfall model
- ☆ V-cycle
- ☆ Spiral model

The Waterfall Model

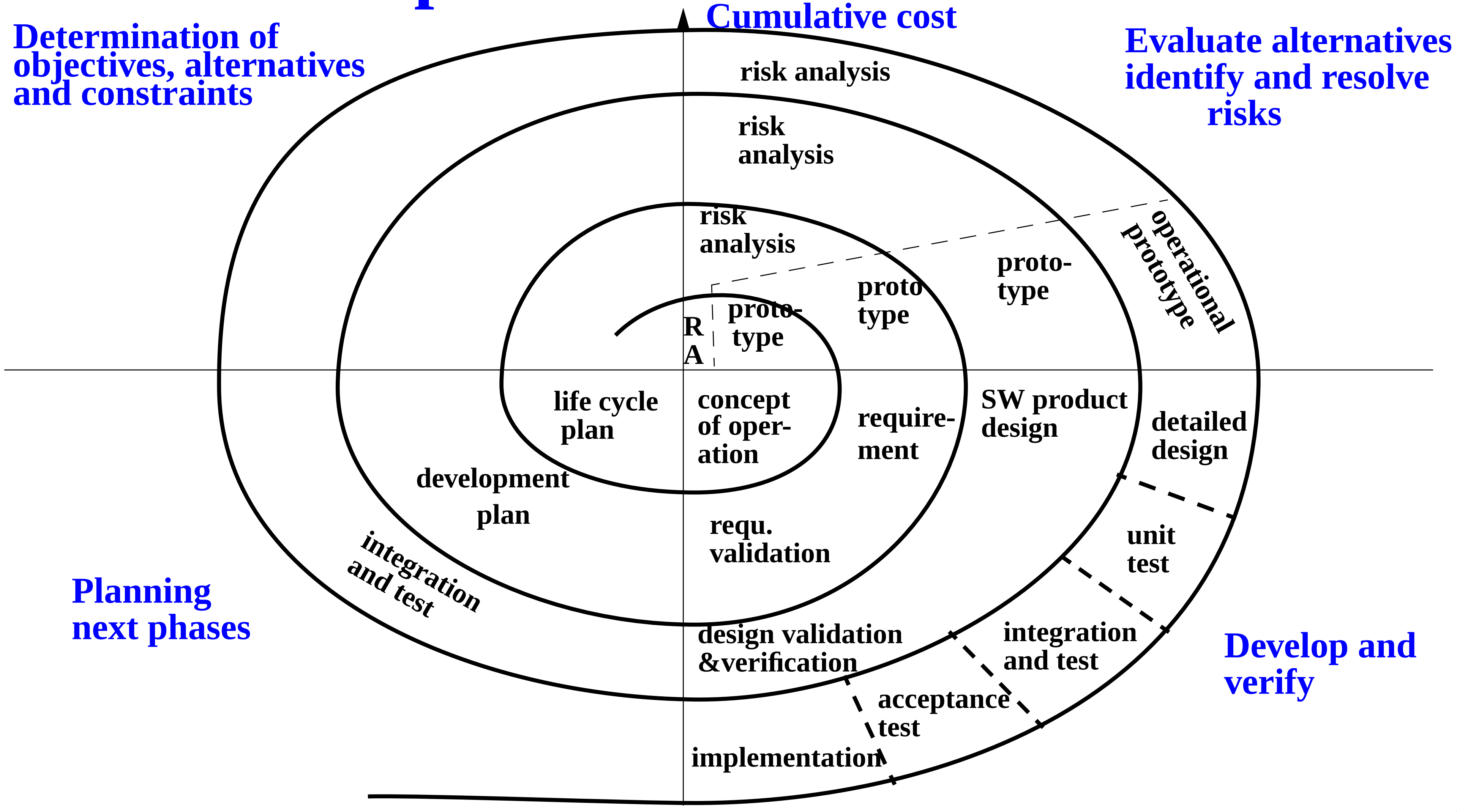


Based on the assumption of document completion at the end of each stage. This is problematic for applications for which the requirements and implementation technology is poorly understood.

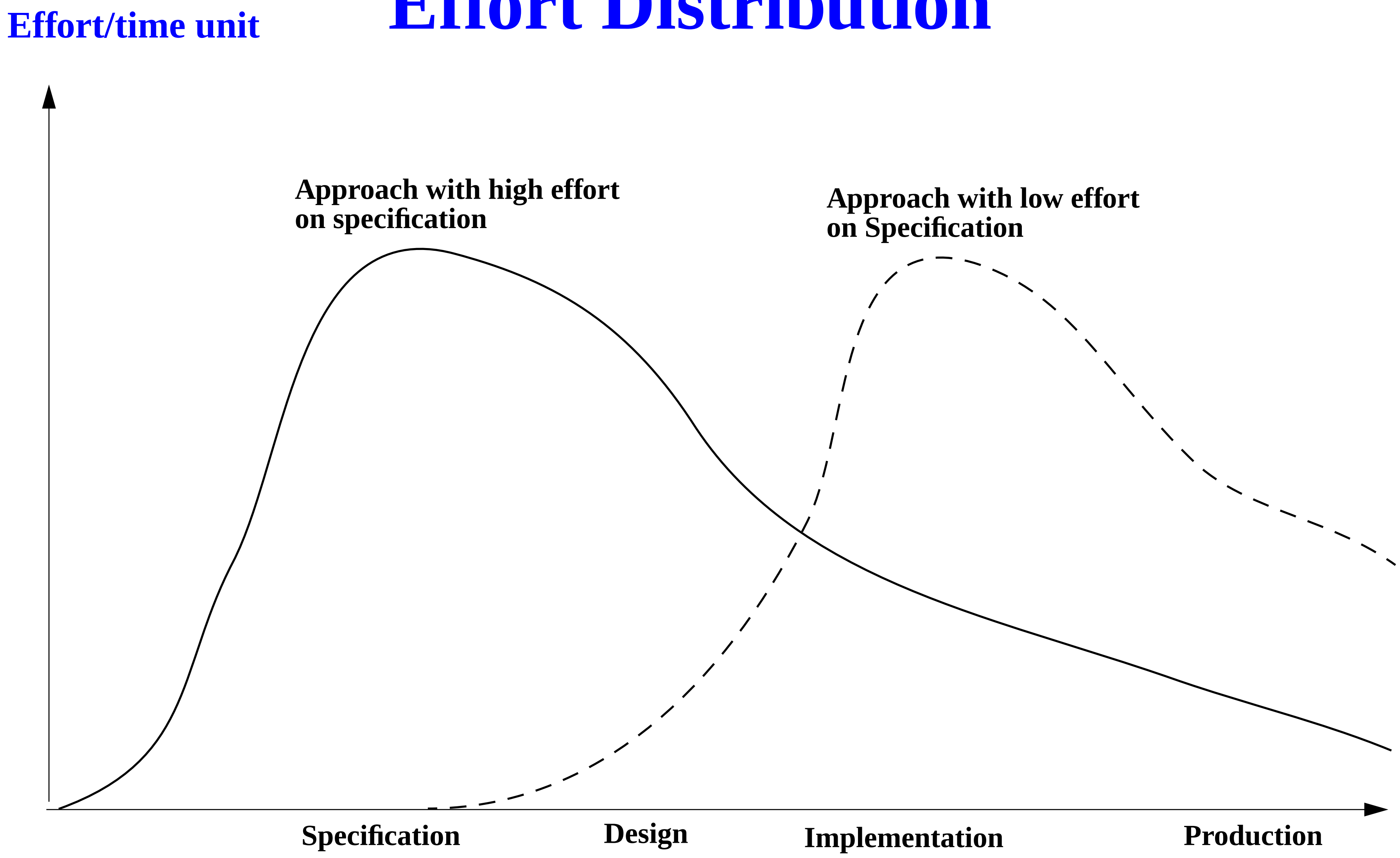
The V life Cycle



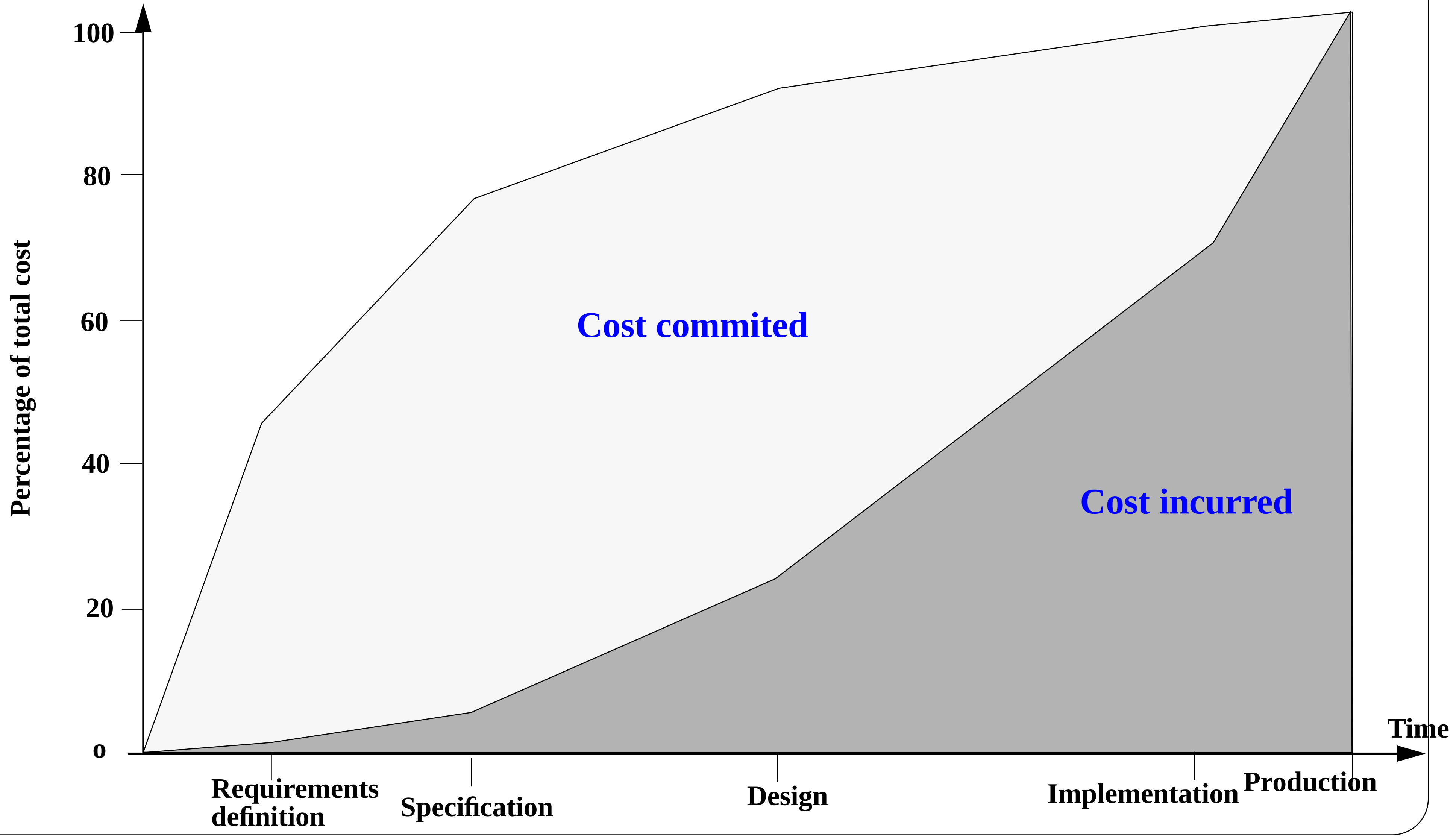
The Spiral Model - Risk Driven



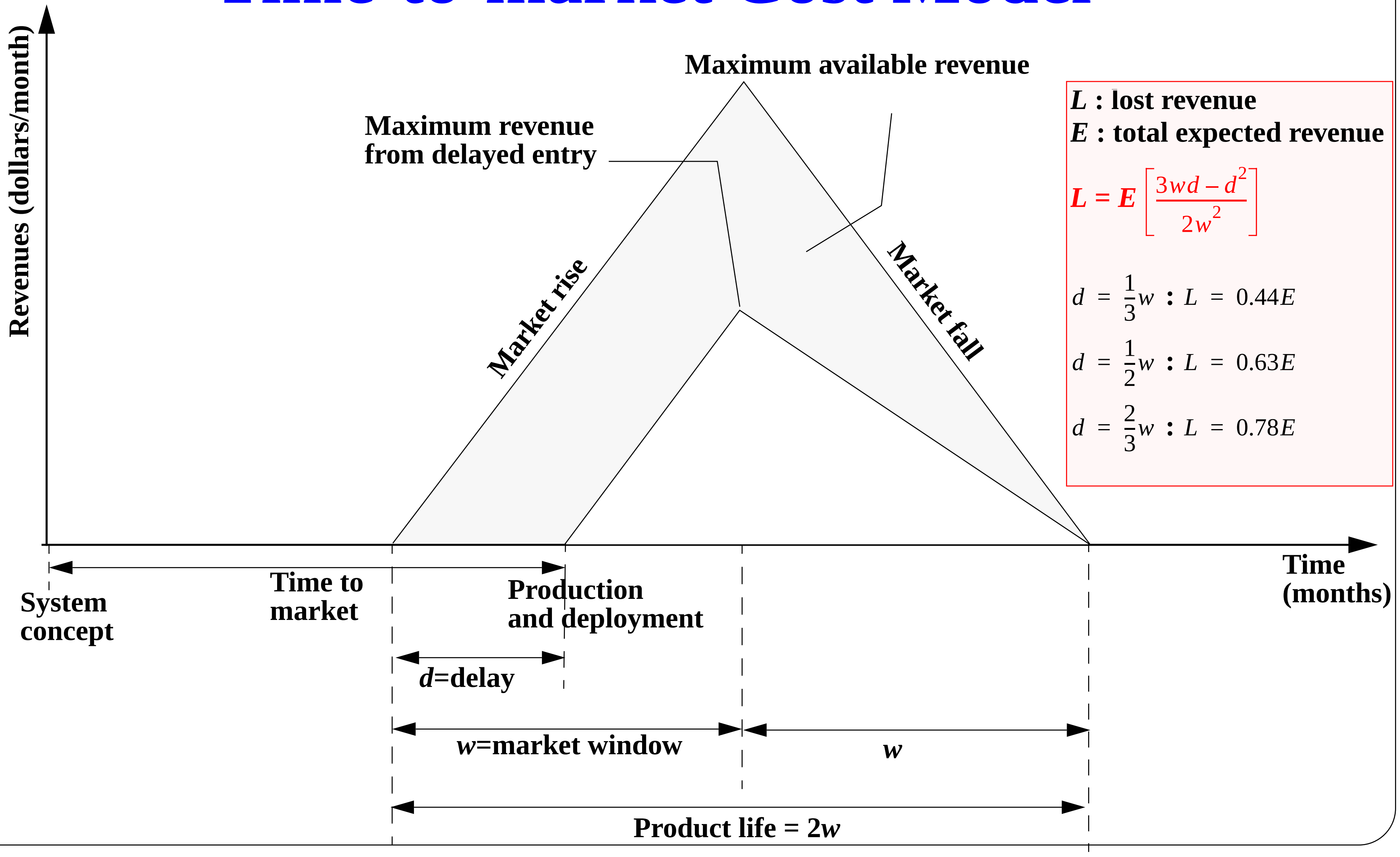
Effort Distribution



Cost Commitment over the Product Life Cycle



Time-to-market Cost Model



L **: lost revenue**
E **: total expected revenue**

$$L = E\left[\frac{3wd - d^2}{2w^2}\right]$$

$$d = \frac{1}{3}w : L = 0.44E$$

$$d = \frac{1}{2}w : L = 0.63E$$

$$d = \frac{2}{3}w : L = 0.78E$$

Codesign Related Tasks - Design

- Specification of functionality as a set of concurrent tasks
- Specification of architecture and resource allocation
  - Type and number of processors
  - Operating system
  - Communication structure
    - One central bus vs Several separated buses
    - Hierarchical bus structure
    - Switches
  - Protocols
    - Data link layer
    - Network layer
    - Application layer
  - Memory structure and hierarchy
    - Shared memory
    - Local memory
    - Caches - Cache coherence
    - Memory allocation to different tasks and protection from other tasks
  - Application specific HW resources
- Task partitioning and binding
- Task synthesis and implementation
- Communication synthesis and implementation
- System integration

Codesign Related Tasks - Performance

- System level performance analysis
  - Error rate, failure rate and reliability, buffering requirements, etc.
- Architecture performance analysis
  - Capacity of computation resources
  - Capacity of communication resources
  - Capacity of storage resources
  - Performance of operating systems (context switching, worst case reaction time, etc.)
- System performance verification
  - Performance for each task
  - Communication performance
  - Memory size and performance, footprint of SW in the memory
  - Overall cost analysis
- Worst case and average case performence
- Static analysis and simulation and profiling based techniques

Codesign Related Tasks - Validation

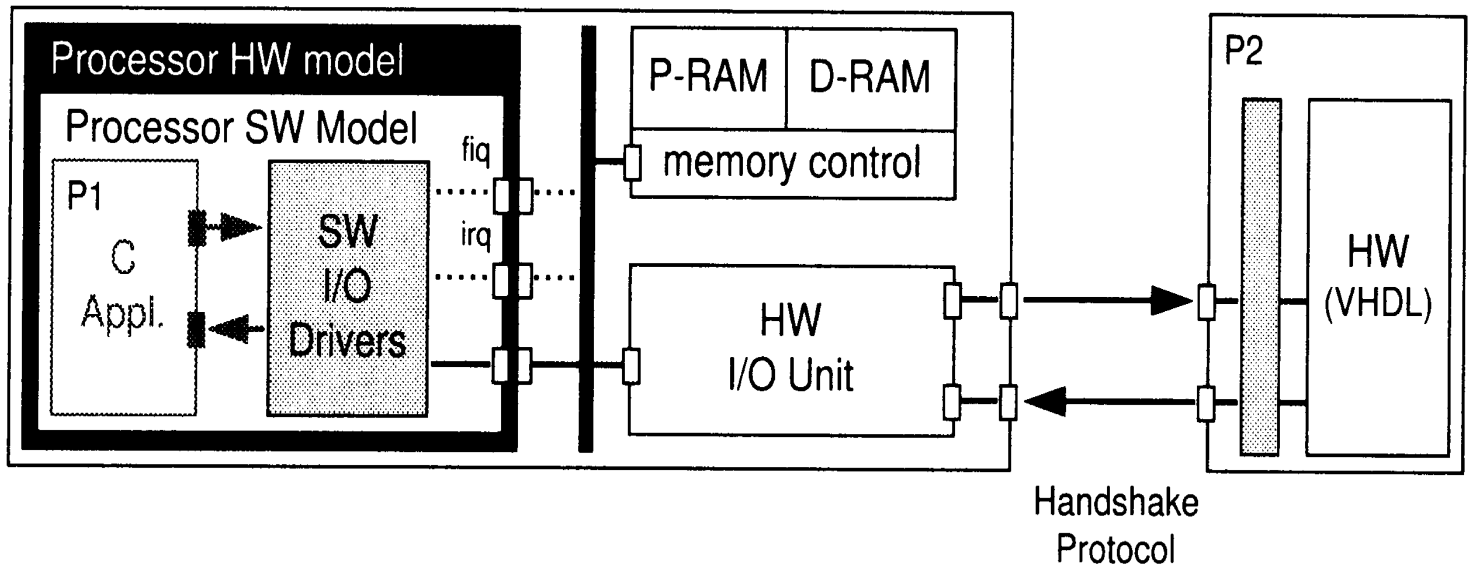
- Validation of requirements (“Do we make the right system?”)
- Functional validation of the task graph
- Validation of each task’s implementation
- Validation of the implementation of the communication between tasks
- System validation (“Did we make the system right?”)

- Simulation
  - Test bench development
  - Testcase development
- Formal verification
  - Equivalence checking
  - Property checking
  - Theorem proving

Models and Languages

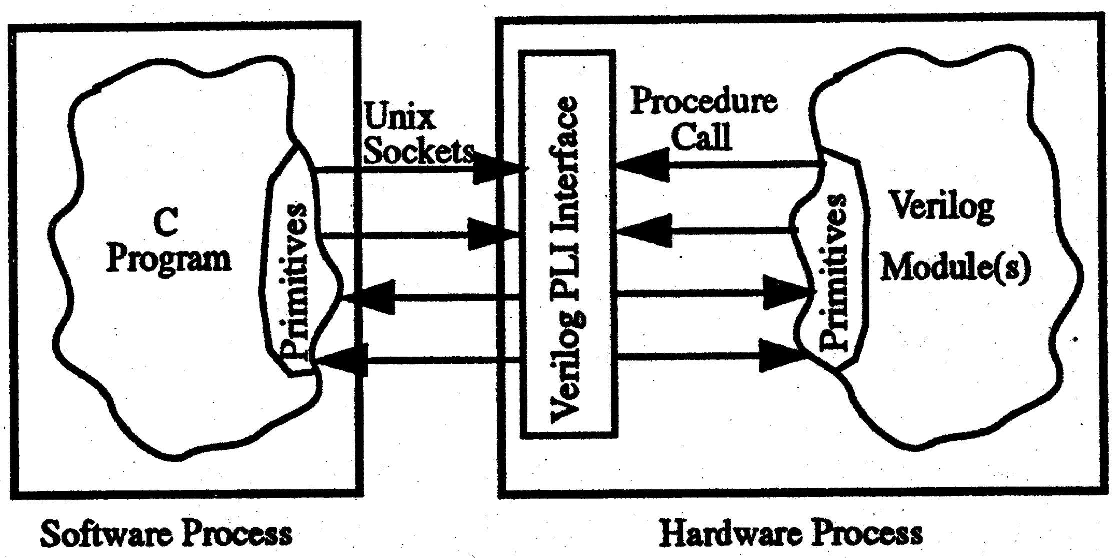
- System level
  - Application oriented (e.g. SDL, Matlab, UML)
  - Co-modelling to integrate different application aspects
- Design and implementation level
  - Implementation oriented (e.g. VHDL, C, C++, Esterel, Java)
  - Modelling of the architecture to gain accurate performance estimates by means of simulation and profiling
  - Co-modeling to integrate different implementation technologies

HW-SW Cosimulation



Contents

Communication



Underlying interconnection:

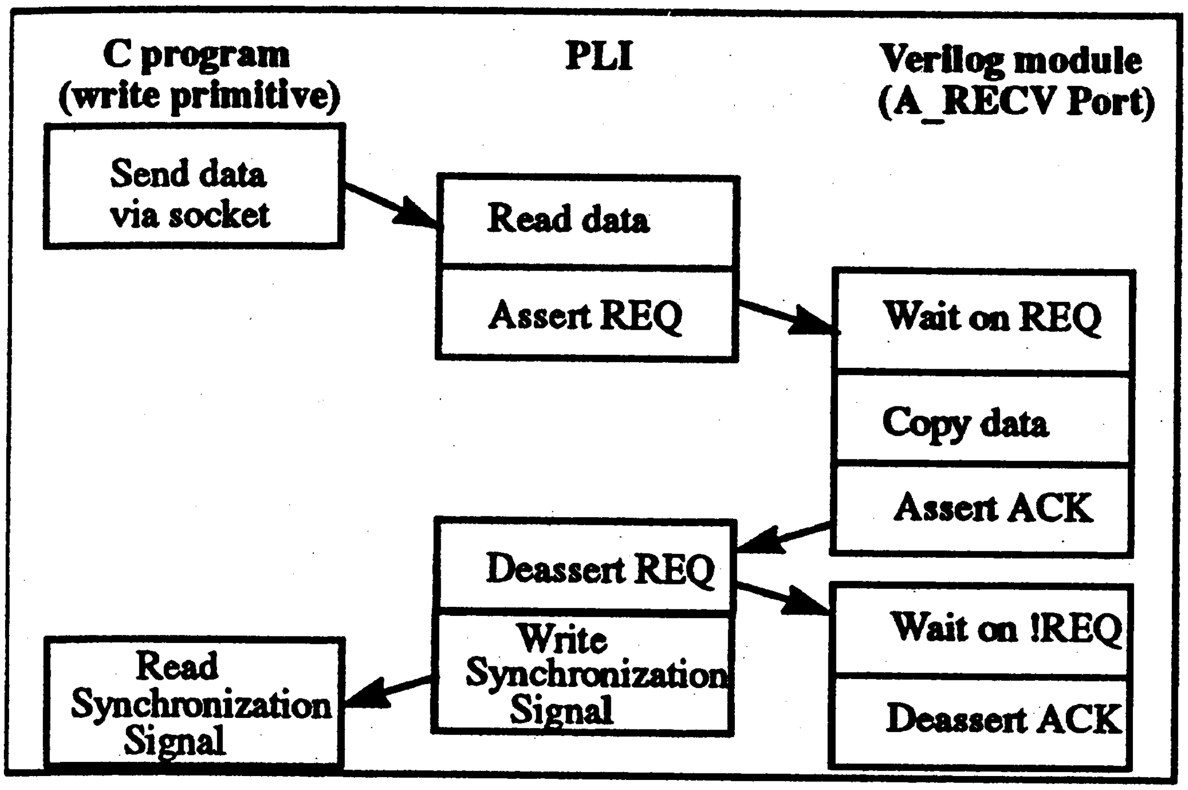
- ☆ Unix InterProcess Communication (IPC)
  - Pipe
  - Socket Interface
  - Remote Procedure Call (RPC)

utilized by:

- ☆ Simulator Interfaces
  - VHDL: Foreign Language Interface
  - Verilog: Programming Language Interface

Communication (cont'd)

- Methods:
  - Shared Memory
  - Message Passing
- Mechanisms:
  - buffered / unbuffered
  - blocking / non-blocking
  - synchronized / unsynchronized data transfer
  - handshaking



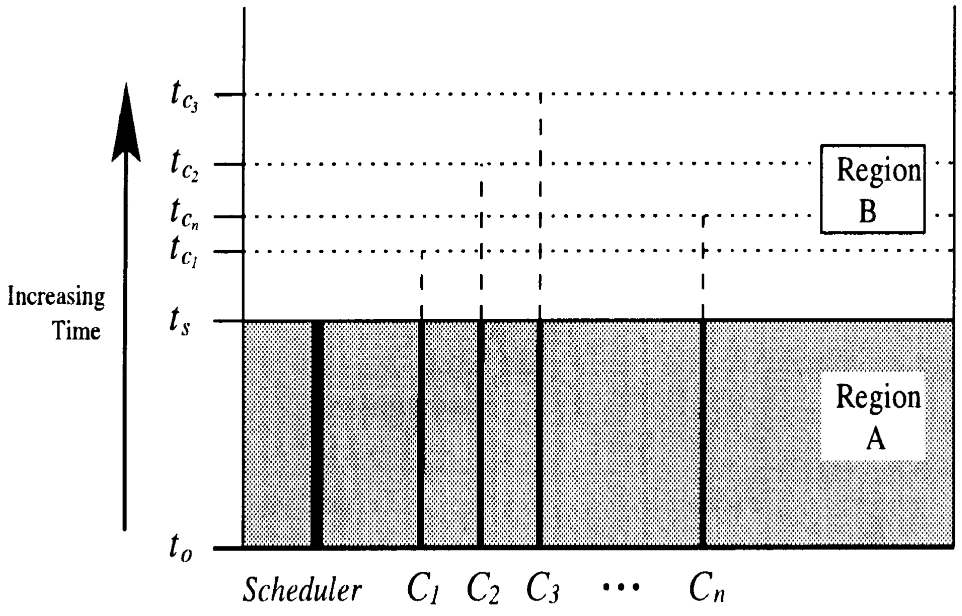
Scheduling

Global Timing Concept:

- scheduler keeps global time
- each component keeps its local time

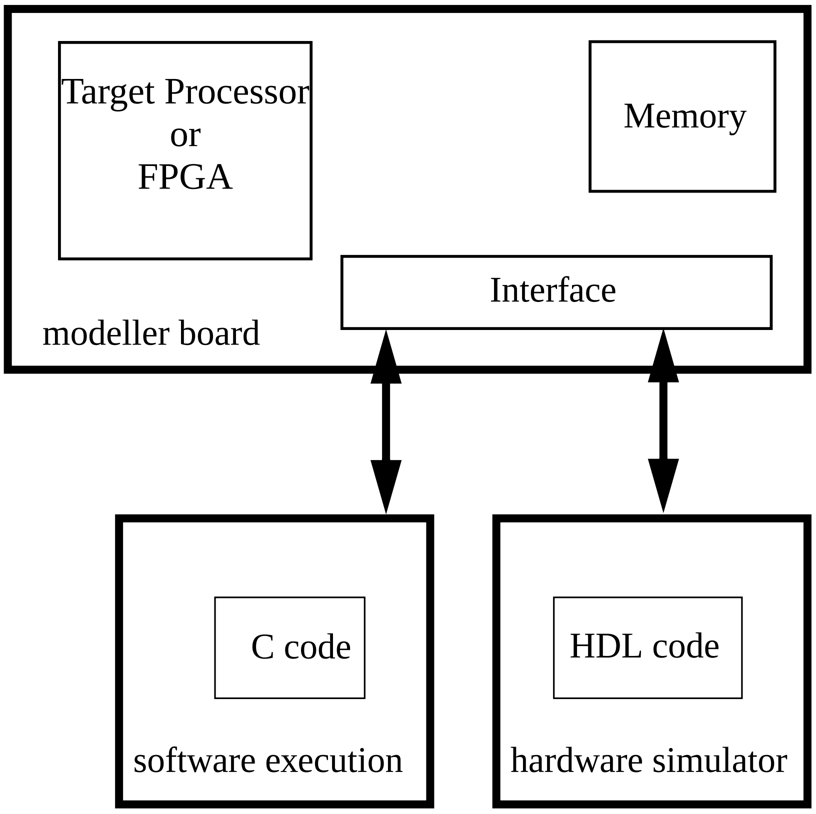
Scheduling Methods:

- conservative scheduling
  - always: global time ≤ local time
  - global time monotonously increasing
  - region A: time has past
  - region B: *send* actions are allowed; *get* actions block;
- optimistic scheduling
  - *get* actions in region B are allowed.



Processor Models I

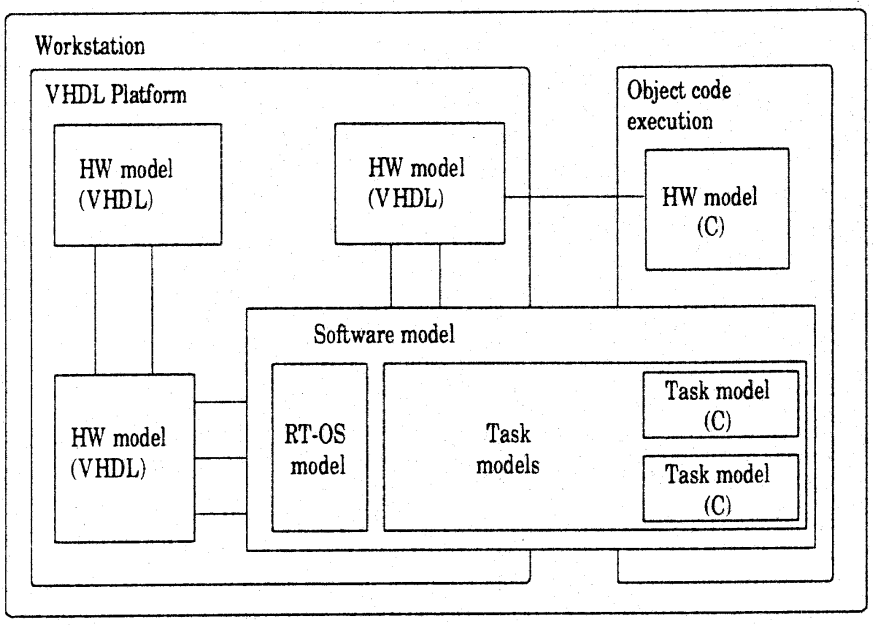
- Hardware and software models exist
- principles of choice:
  - model availability
  - performance
  - timing accuracy
  - debugging features
- Hardware Models
  - target processor
  - logic emulator (FPGA)



Hardware Model

Processor Models II

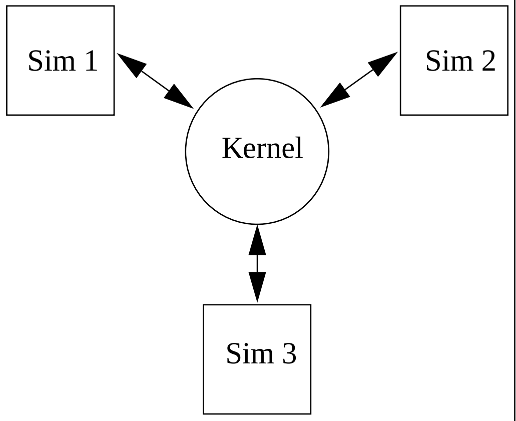
- Software Processor Models
  - nanosecond accurate model
  - cycle accurate model
  - instruction set accurate model (ISS)
  - bus functional model (BFM)
- Techniques Requiring No Processor Model
  - host code execution (HCE)
  - virtual operating system



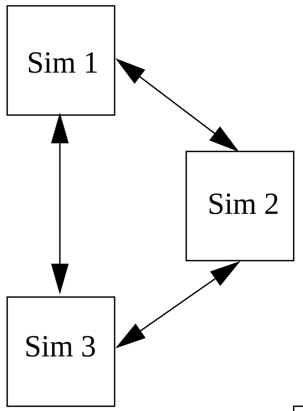
Virtual Operating System

Tool Structures

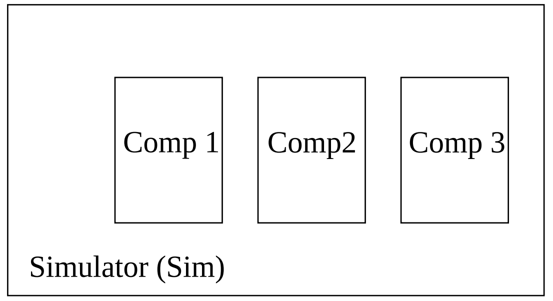
- Backplane Based Simulation
  - contains interconnecting kernel
  - kernel provides all required translations
  - all signals are translation to a general type
- Heterogeneous Simulation
  - simulators are directly connected
  - explicit control and translation of signals in the simulators
- Single Process Simulation
  - entire simulation in a single simulator
  - easy to implement
  - e.g. VHDL-based simulation



Backplane Based Simulation



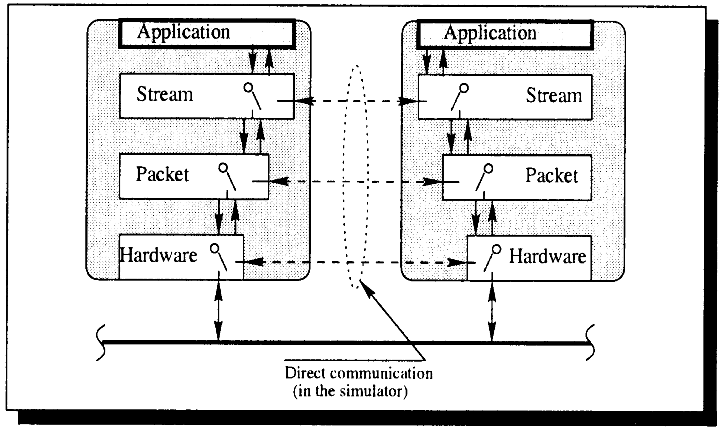
Heterogeneous Simulation



Single Process Simulation

Speed-Up Mechanisms

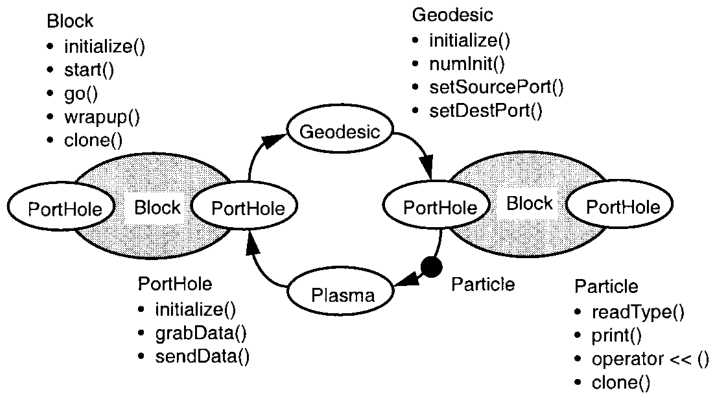
- Multiple Communication Models
  - allow communication on different levels of abstraction

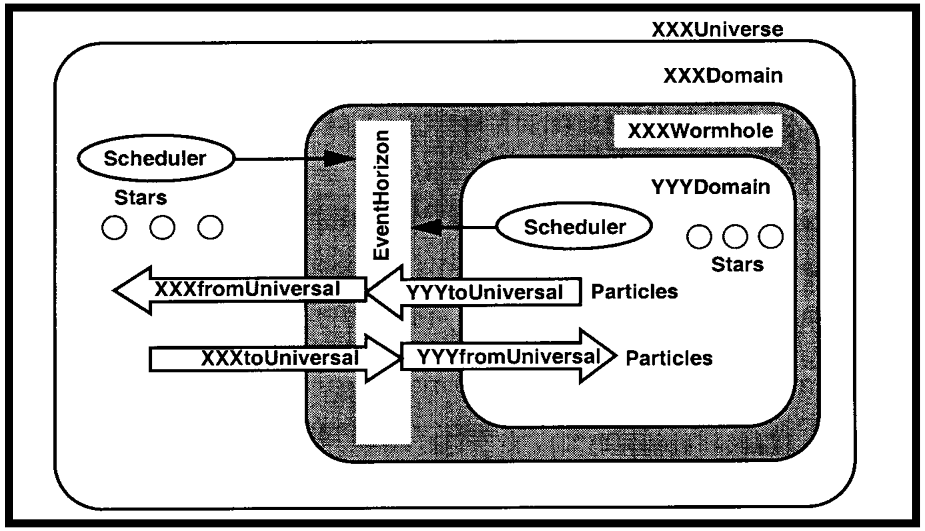


- Memory Image Server
  - maintain two memory models
- Distributed Simulation
  - execute the cosimulation on several machines
- High Powered Coprocessor

Tools I: Ptolemy

- ☆ developed at University of California at Berkeley
- ☆ simulation framework
- ☆ written in object-oriented language (C++)
- ☆ build on basic classes
- ☆ allows design of heterogeneous systems
- ☆ designer is enabled to build arbitrary simulation environments

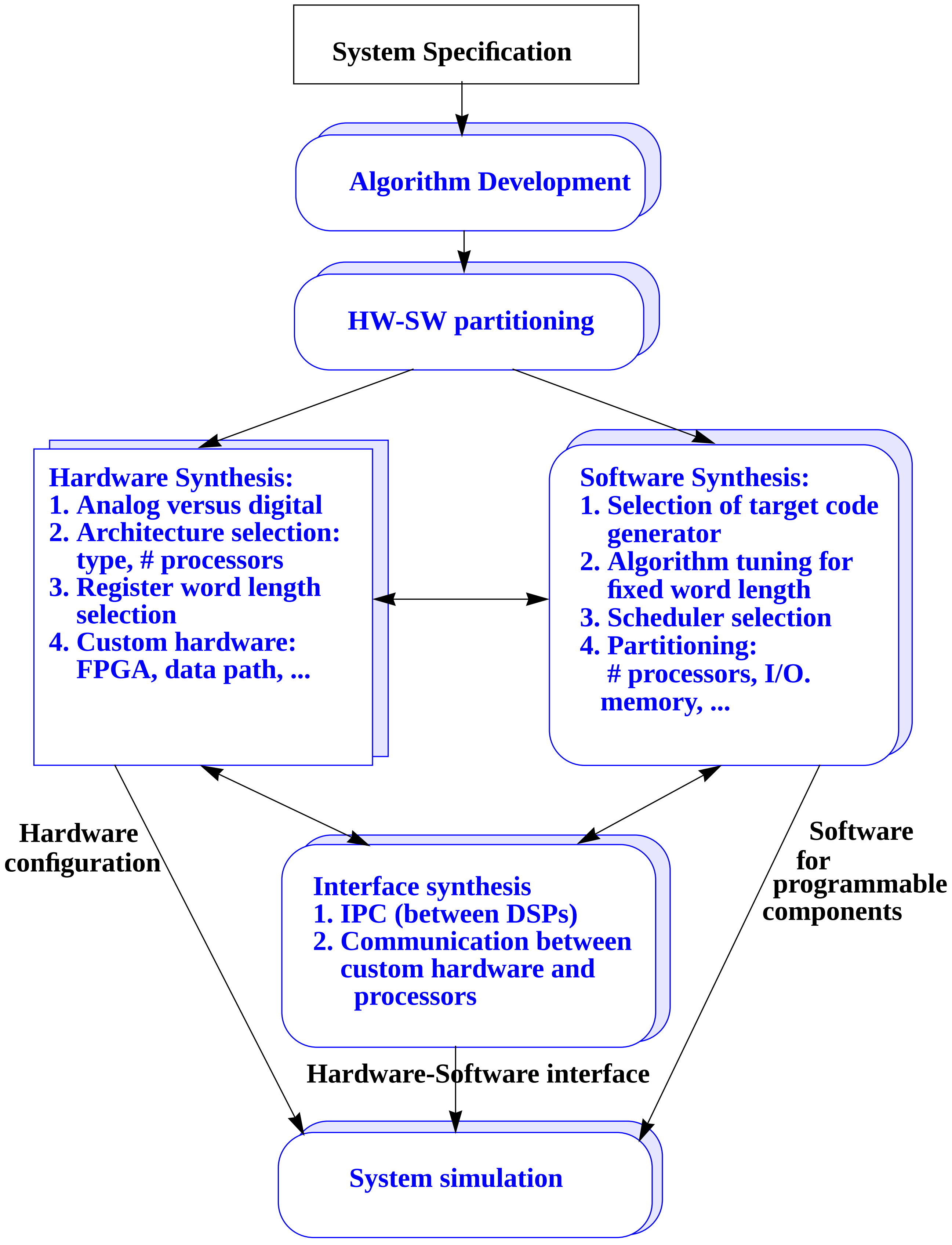




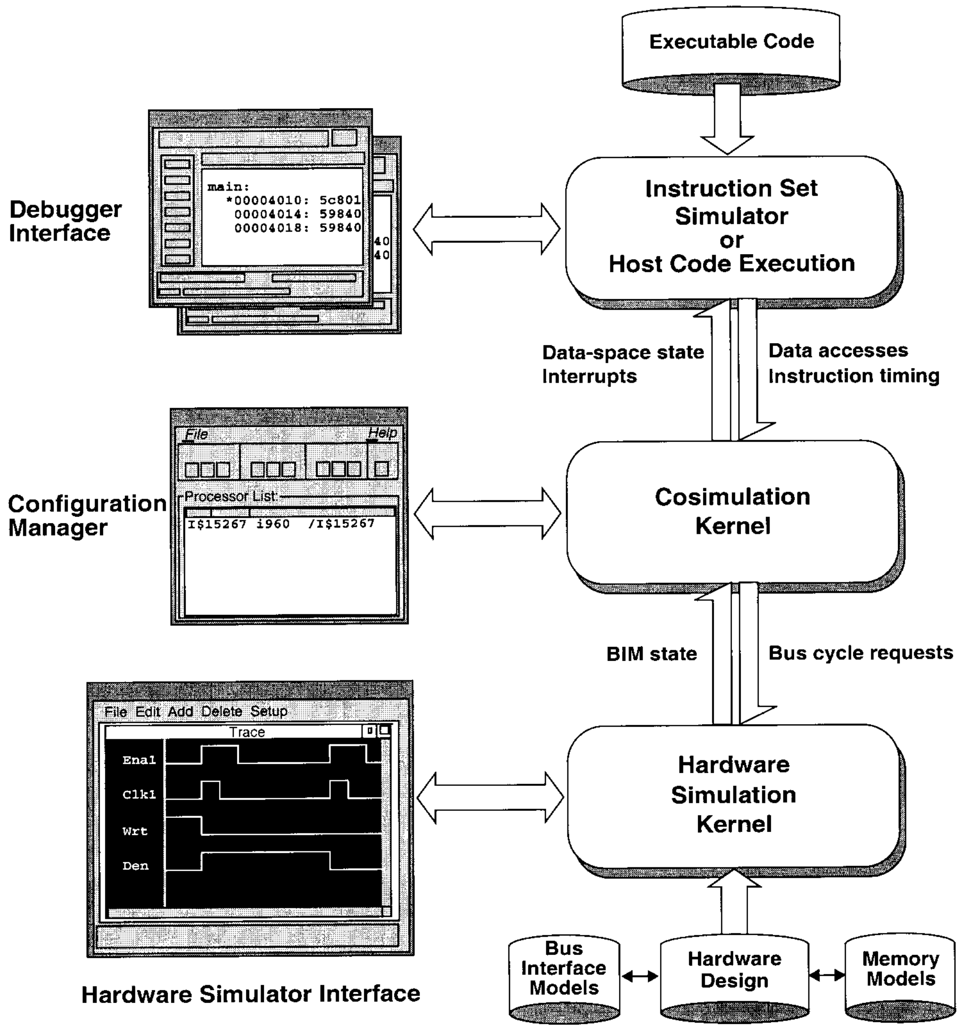
Ptolemy

- Heterogenous simulation environment
- Different domains for different subsystems
- General and flexible interface between domains
- Supported simulation domains:
  - Synchronous data flow
  - Functional simulator for digital hardware
  - Discrete event simulation

Ptolemy

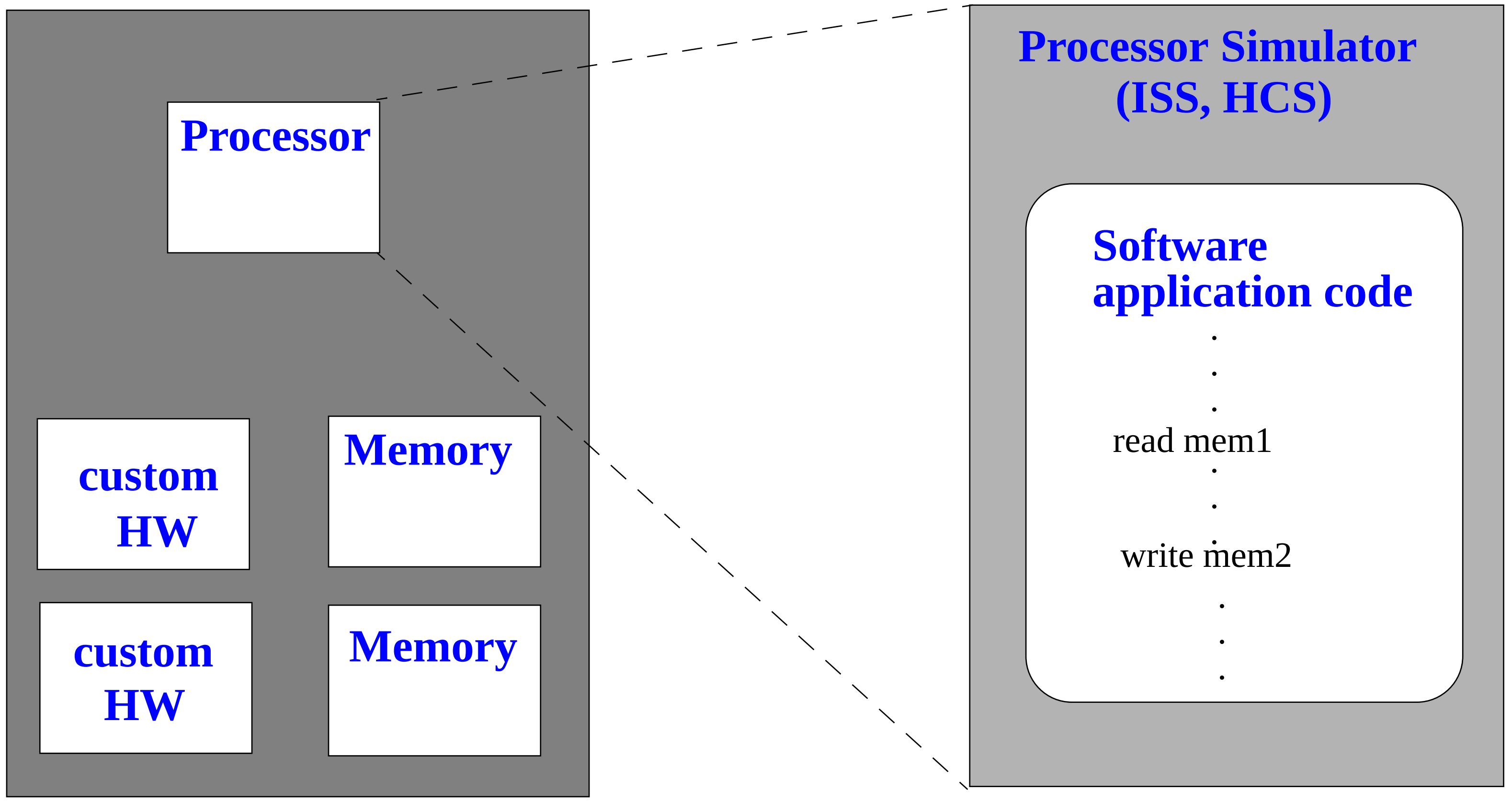

System Specification
Algorithm Development
HW-SW partitioning
Hardware Synthesis:
1. Analog versus digital
2. Architecture selection: type, # processors
3. Register word length selection
4. Custom hardware: FPGA, data path, ...
Software Synthesis:
1. Selection of target code generator
2. Algorithm tuning for fixed word length
3. Scheduler selection
4. Partitioning: # processors, I/O. memory, ...
Hardware configuration
Software for programmable components
Interface synthesis
1. IPC (between DSPs)
2. Communication between custom hardware and processors
Hardware-Software interface
System simulation

Tools III: Seamless

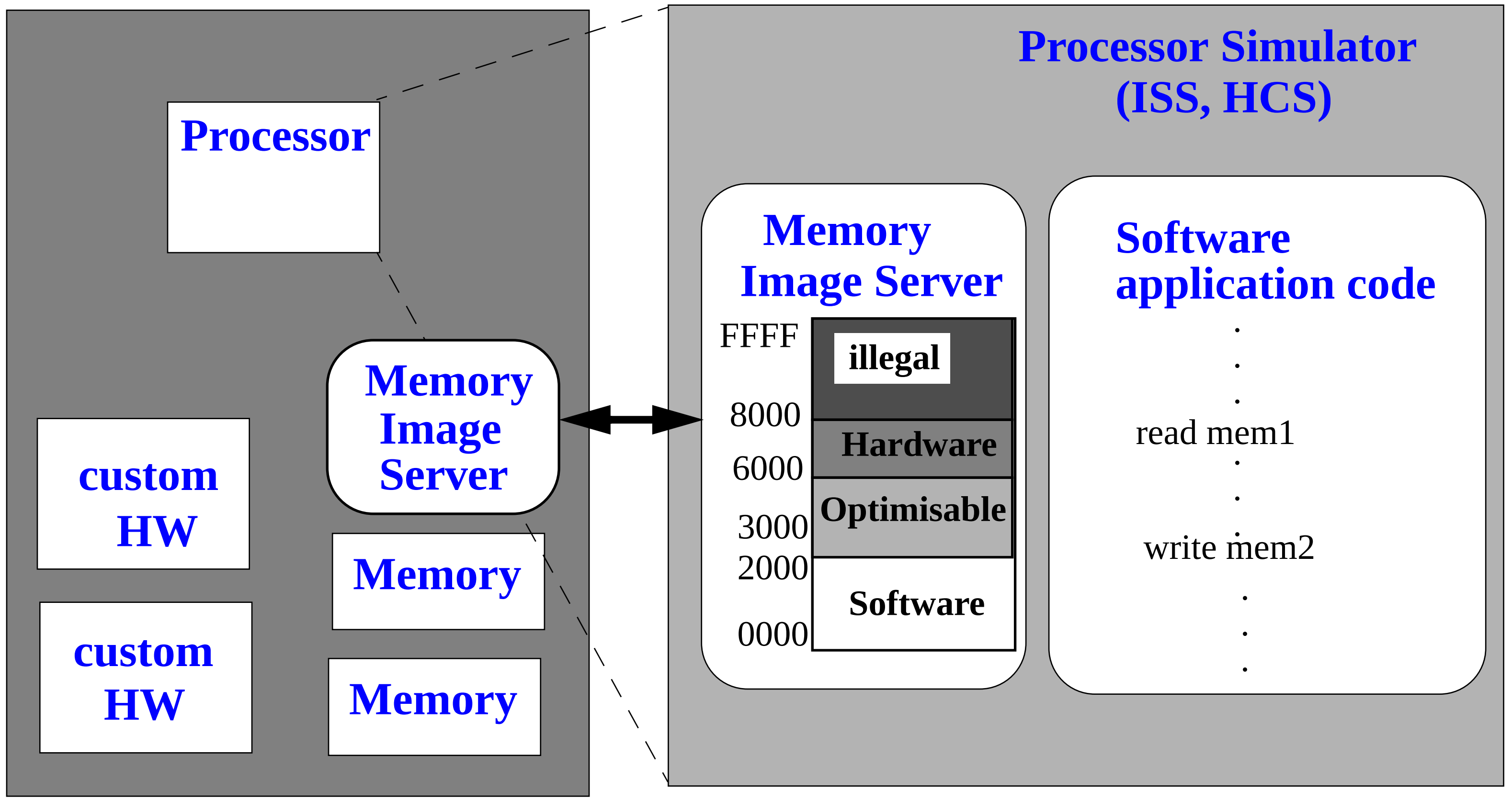


- ☆ distributed by Mentor Graphics
- ☆ contains three parts:
  - software simulator (e.g. ISS)
  - cosimulation kernel
  - hardware simulator (BFM & additional hardware)
- ☆ support speed-up mechanisms
  - unsynchronized simulation
  - memory image server
- ☆ similar tools exist (EagleI, Virtual CPU)

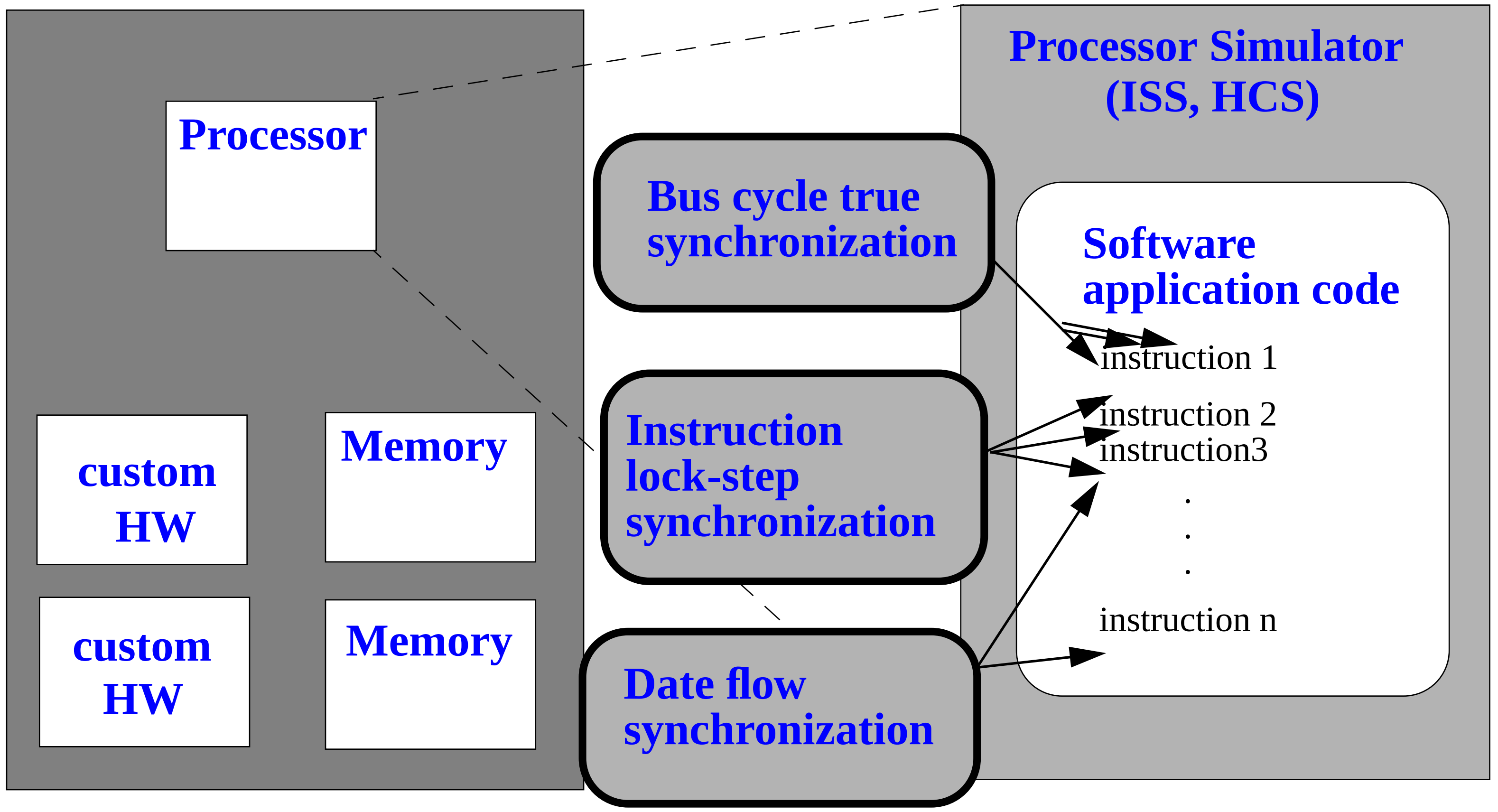
Seamless: Memory Model



Seamless: Memory Model - cont’d



Seamless: Relaxation of Synchronization


Processor
custom HW
Memory
custom HW
Memory
Processor Simulator (ISS, HCS)
Software application code
instruction 1
instruction 2
instruction3
.
.
.
instruction n
Bus cycle true synchronization
Instruction lock-step synchronization
Date flow synchronization

Summary

- Most electronic systems consist of both HW and SW
- HW and SW design have different histories, tools, methodologies concepts and require different competence
- Types of Codeisgn:
  - Vertical Codesign
  - Horizontal Codesign
- Tasks
  - Modelling
  - Design
  - Validation
  - Performance analysis and validation
- HW/SW Cosimulation is the most widely used codesign technique