

SYSTEM-LEVEL DESIGN: Only the Radical Will Survive

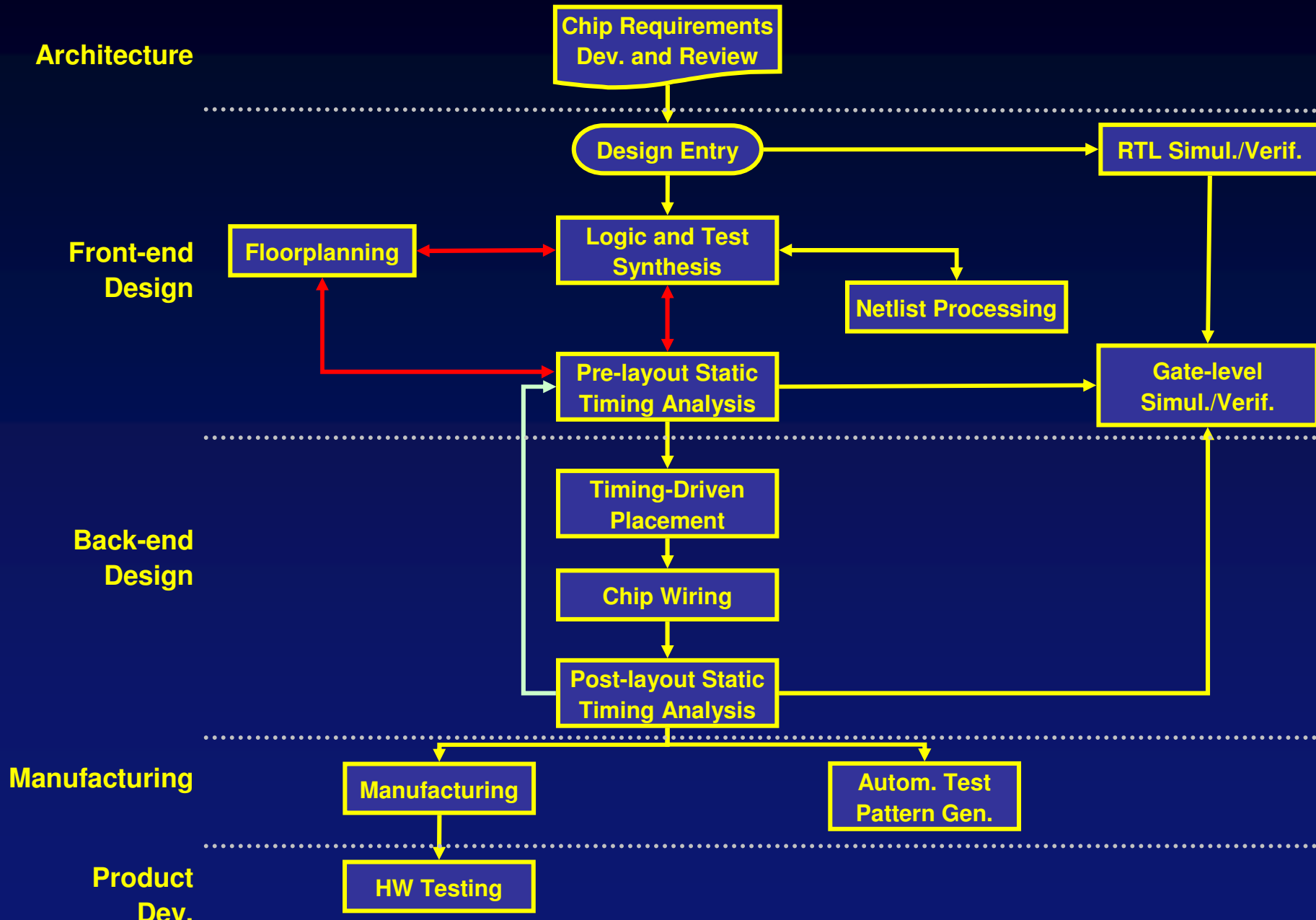


Reinaldo Bergamaschi
IBM T. J. Watson Research Center
Yorktown Heights, NY
berga@us.ibm.com

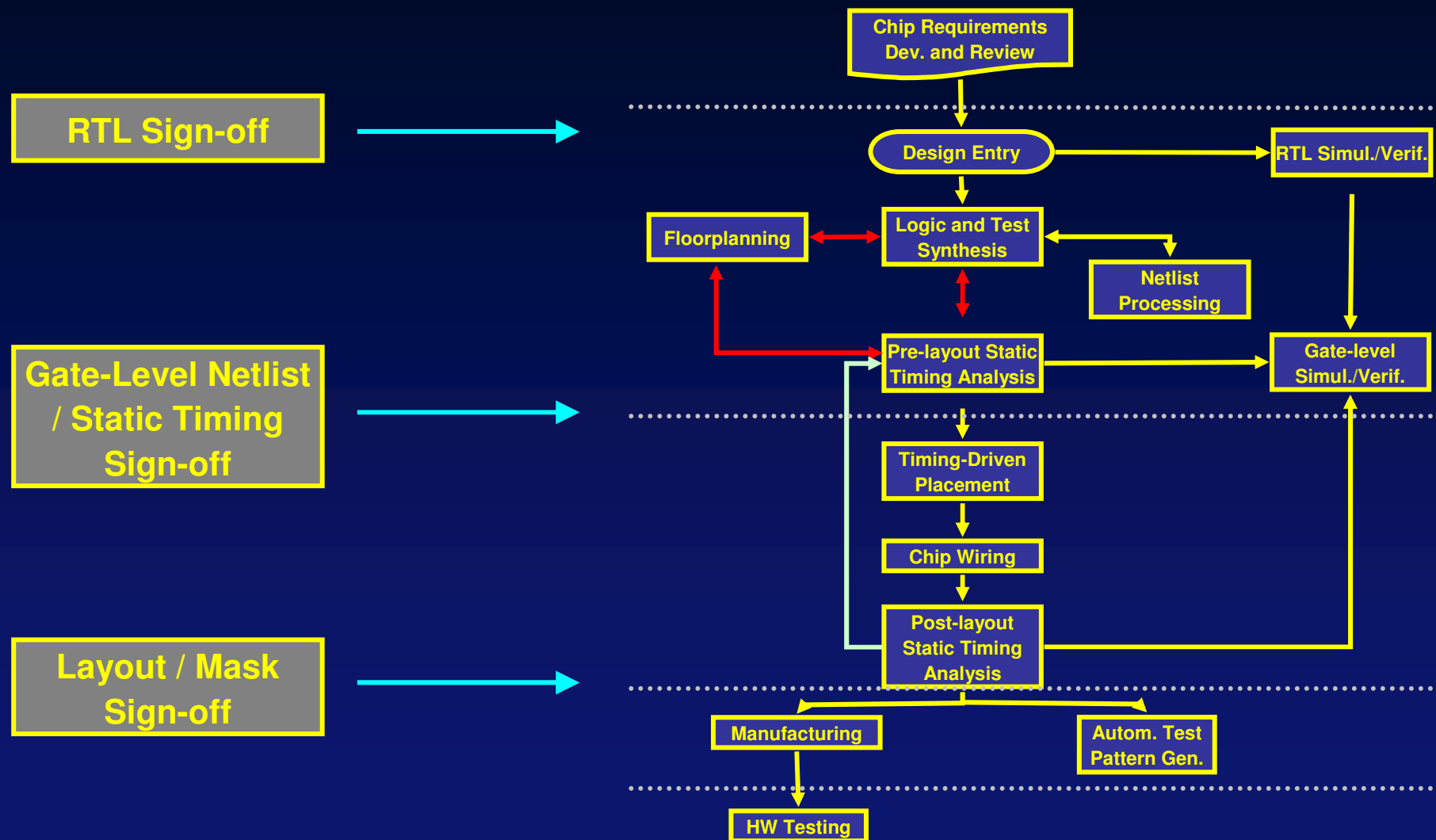
Contents

- ASIC Methodology
- Sign-off Approaches
- From ASICs to Systems (using same old methodology)
- SoC Methodology Evolution
- Productivity and the Design Gap
- The Wisdom of designing HW as SW (or lack thereof)
- The Language Fallacy
- What is missing in current SLD Tools/Approaches
- System-level design tools at IBM Research
- Conclusions

ASIC Methodology (circa 1999)



Sign-off Approaches

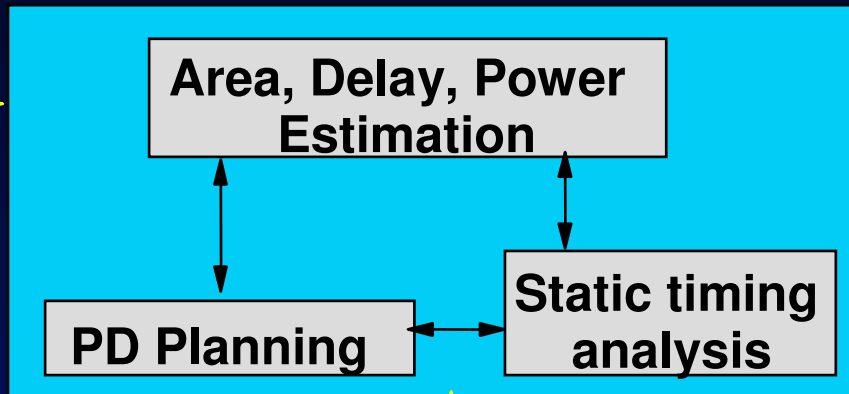


RTL Sign-off (IBM 1997)

USER

Sign-off Kit

VHDL
Verilog



Constraints files

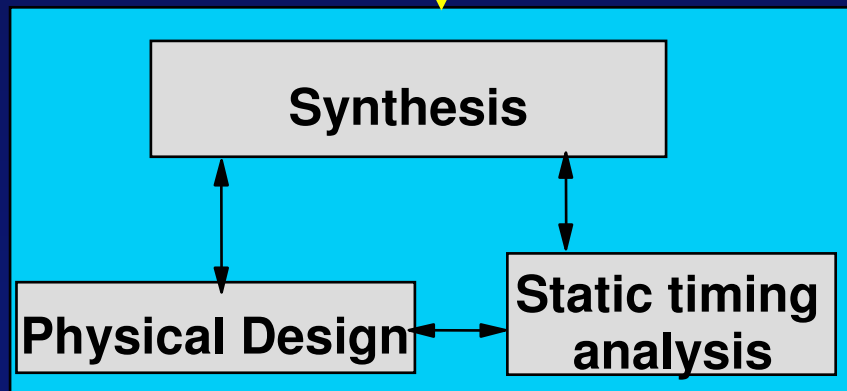
meet requirements ?

No

Yes

ASICS Foundry

Completed design



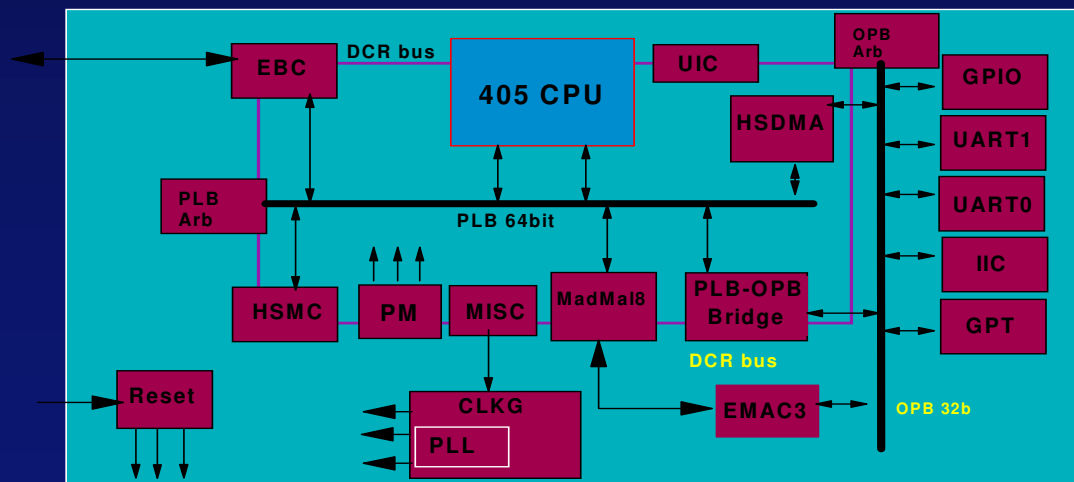
From ASICs to SoCs

Much higher levels of integration (< .25um)

High Performance Circuits and Tools

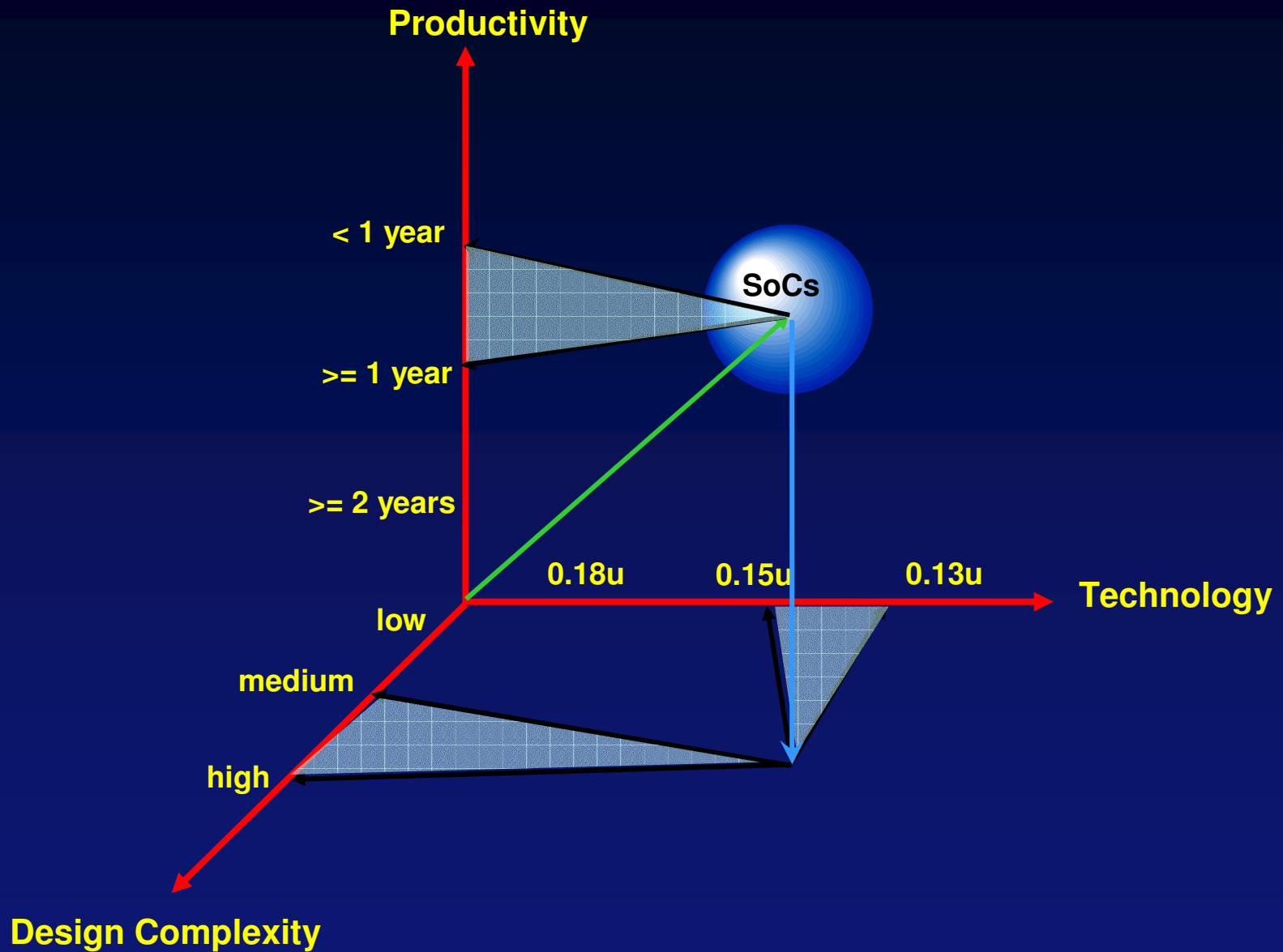
Common Architectural Blocks

Shorter design cycle + Lagging Productivity

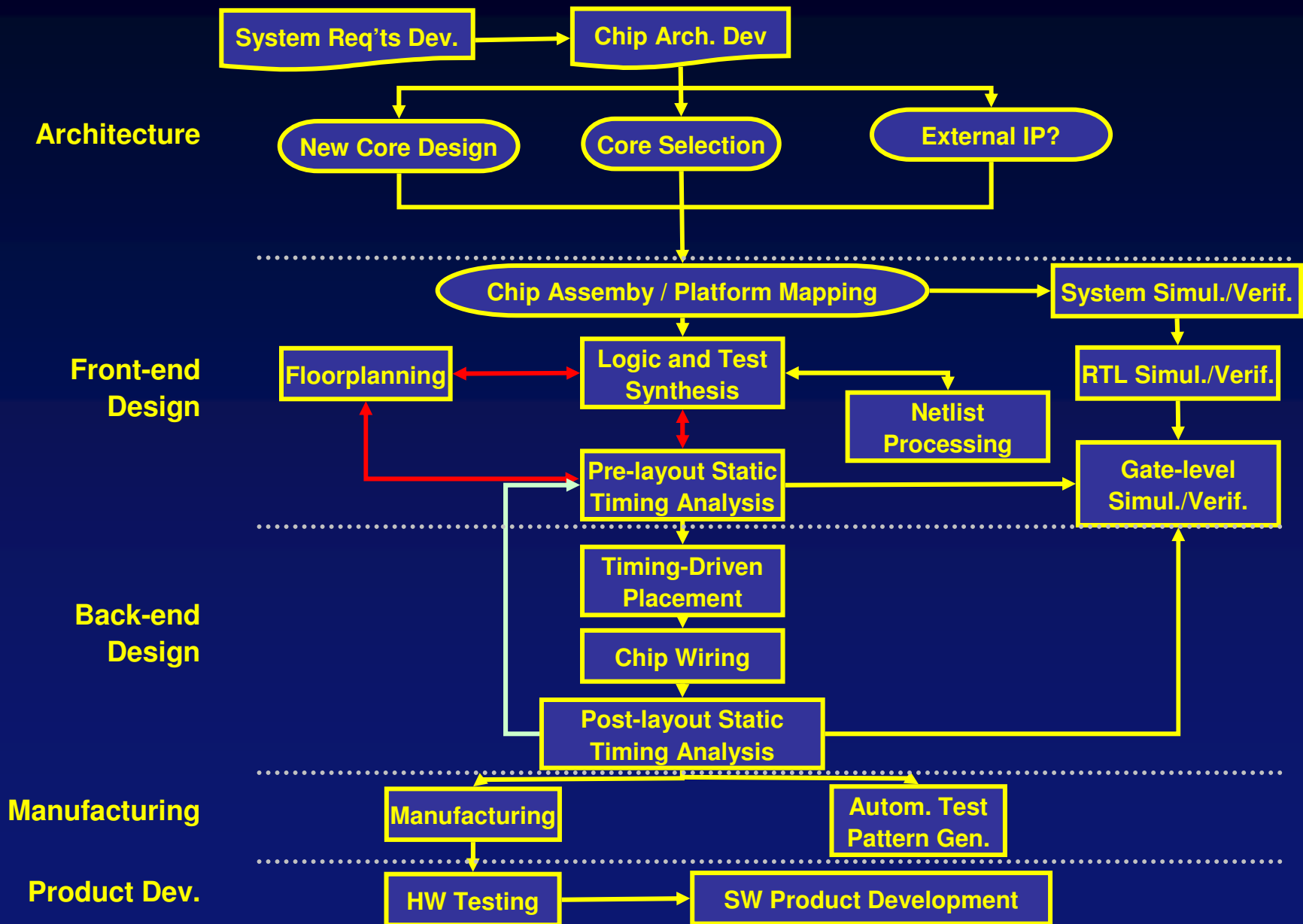


- IP Cores
- Platforms
- Reuse

Technology vs. Productivity vs. Complexity

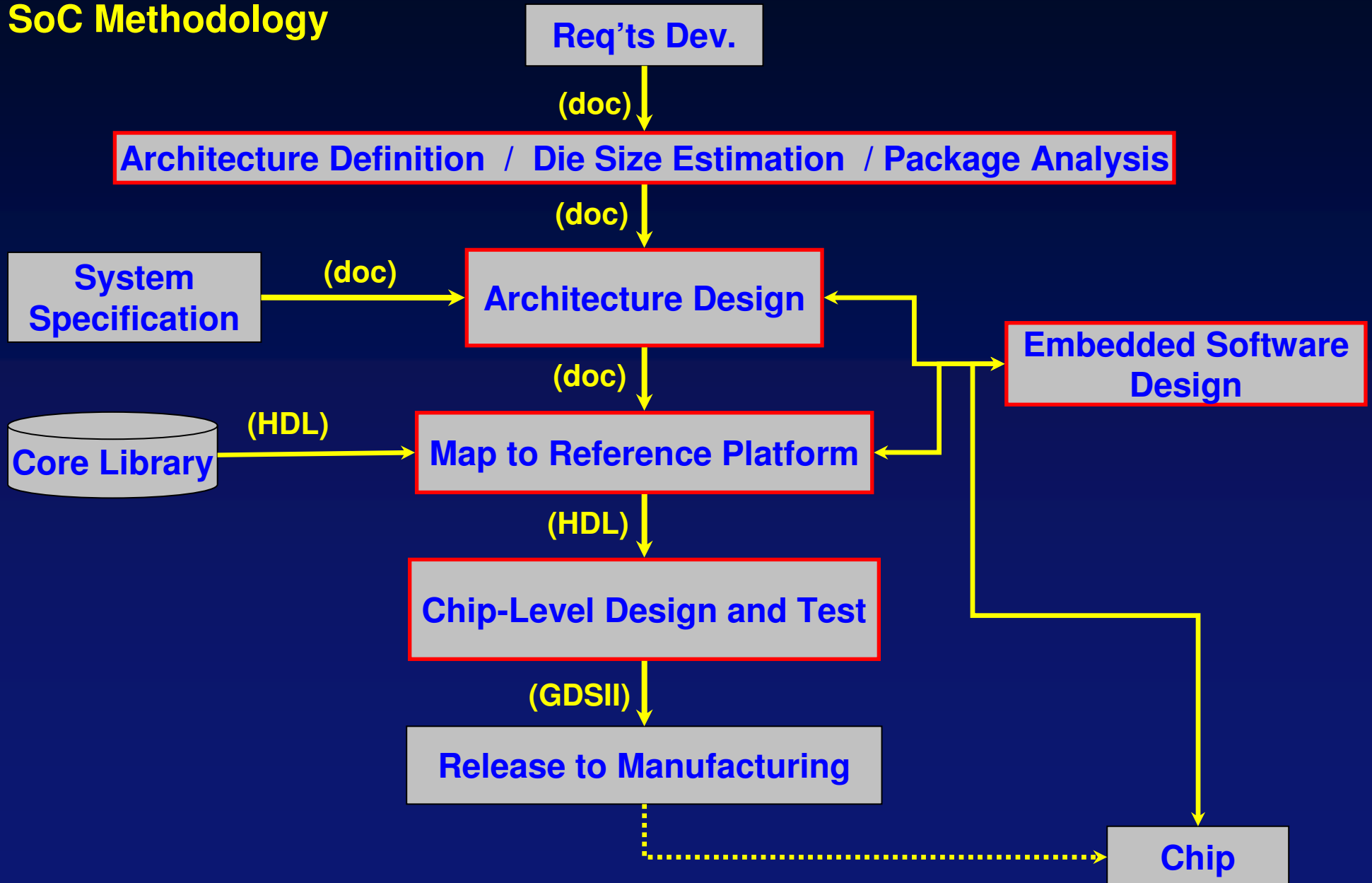


SoC Methodology



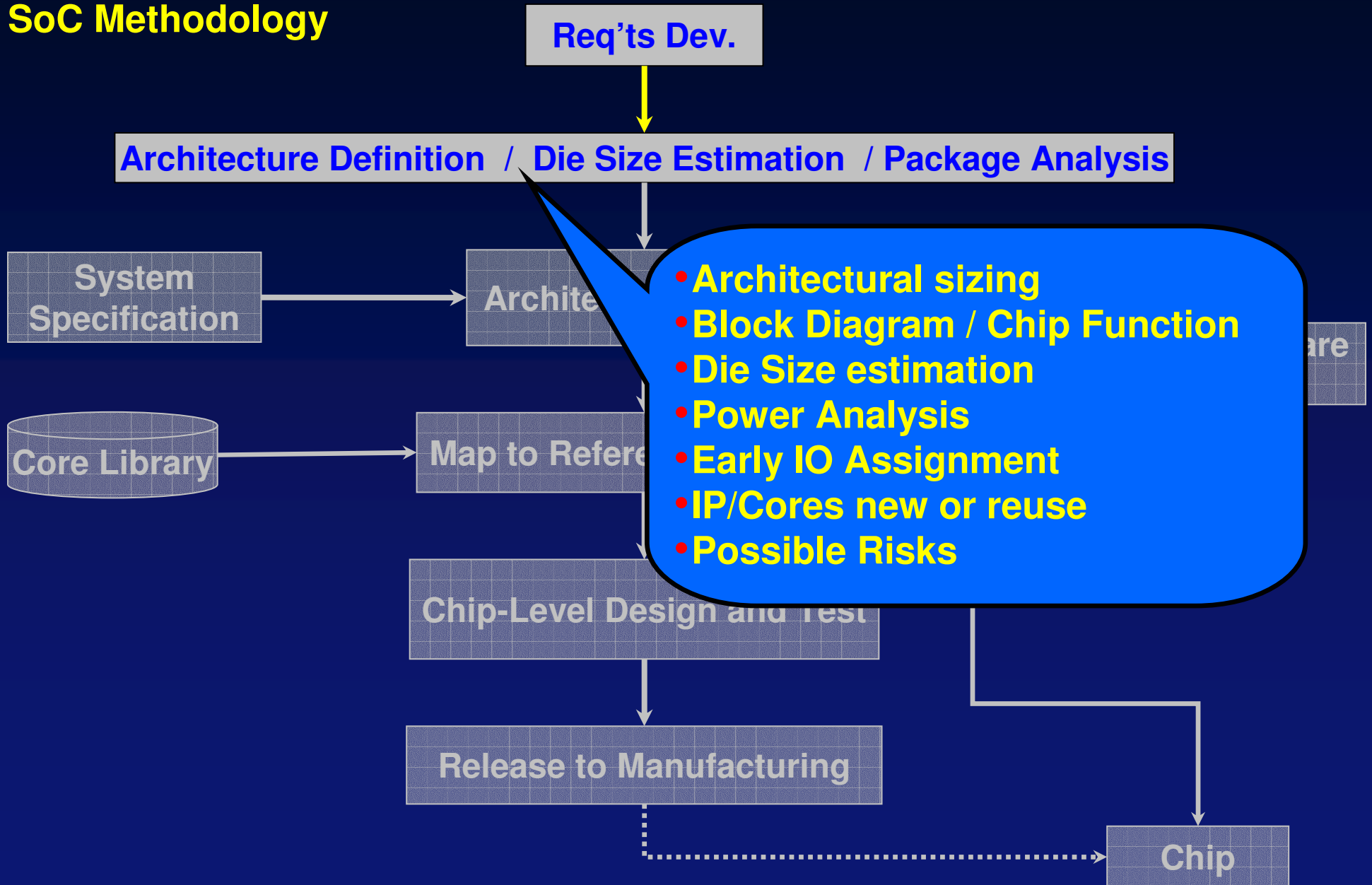
SoC Methodology Evolving...

SoC Methodology



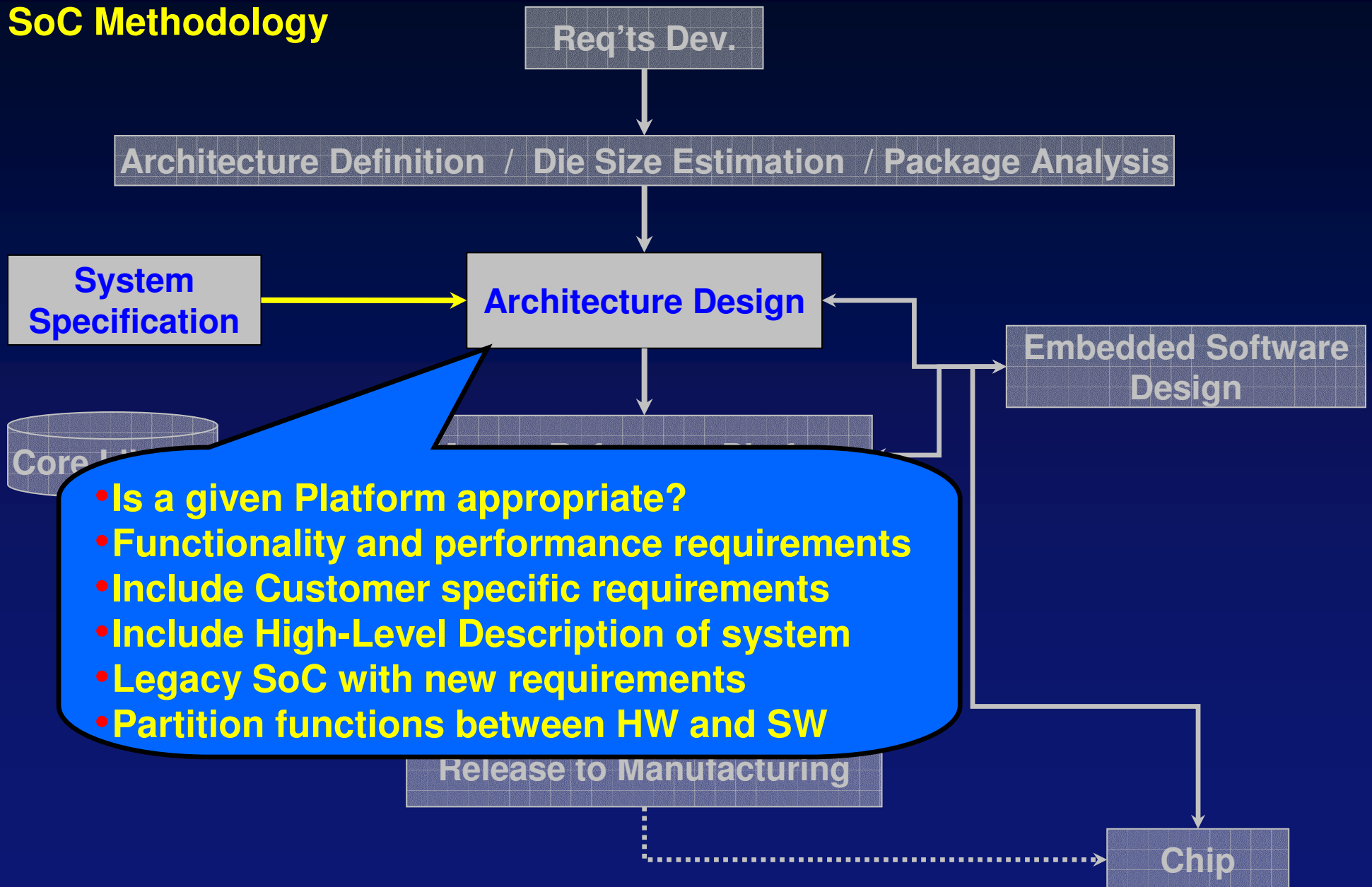
How to Design an SoC

SoC Methodology



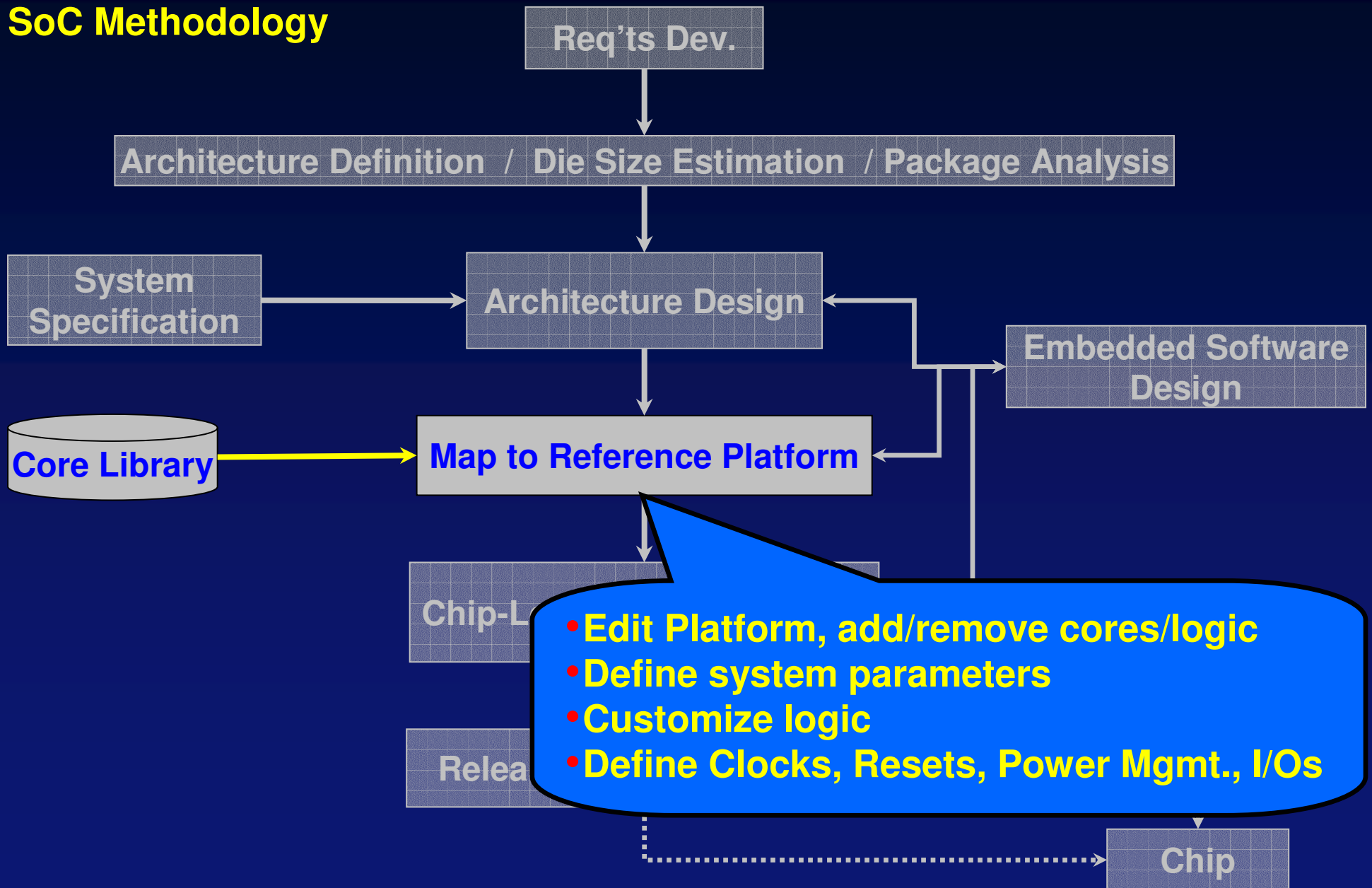
How to Design an SoC

SoC Methodology



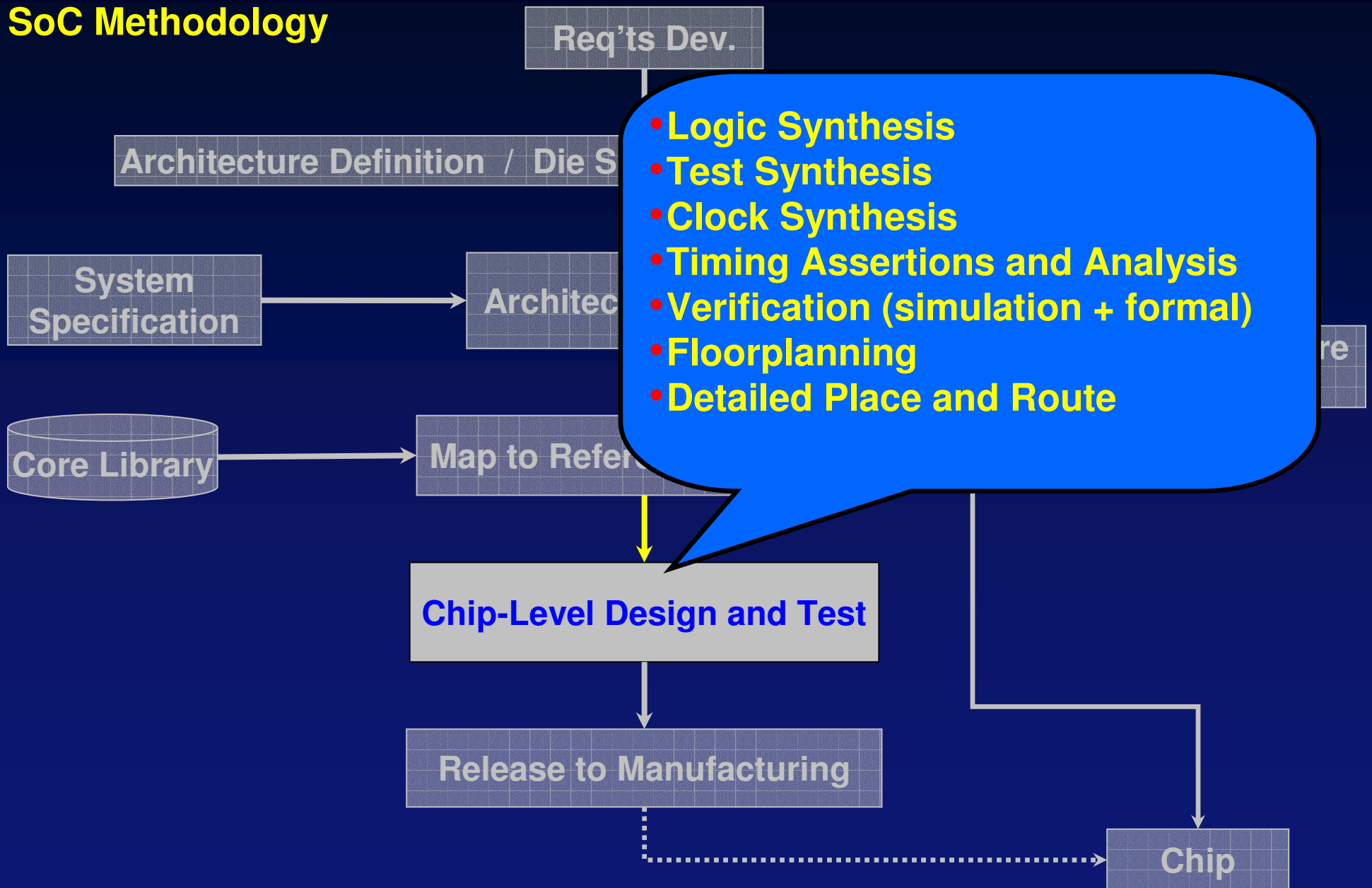
How to Design an SoC

SoC Methodology



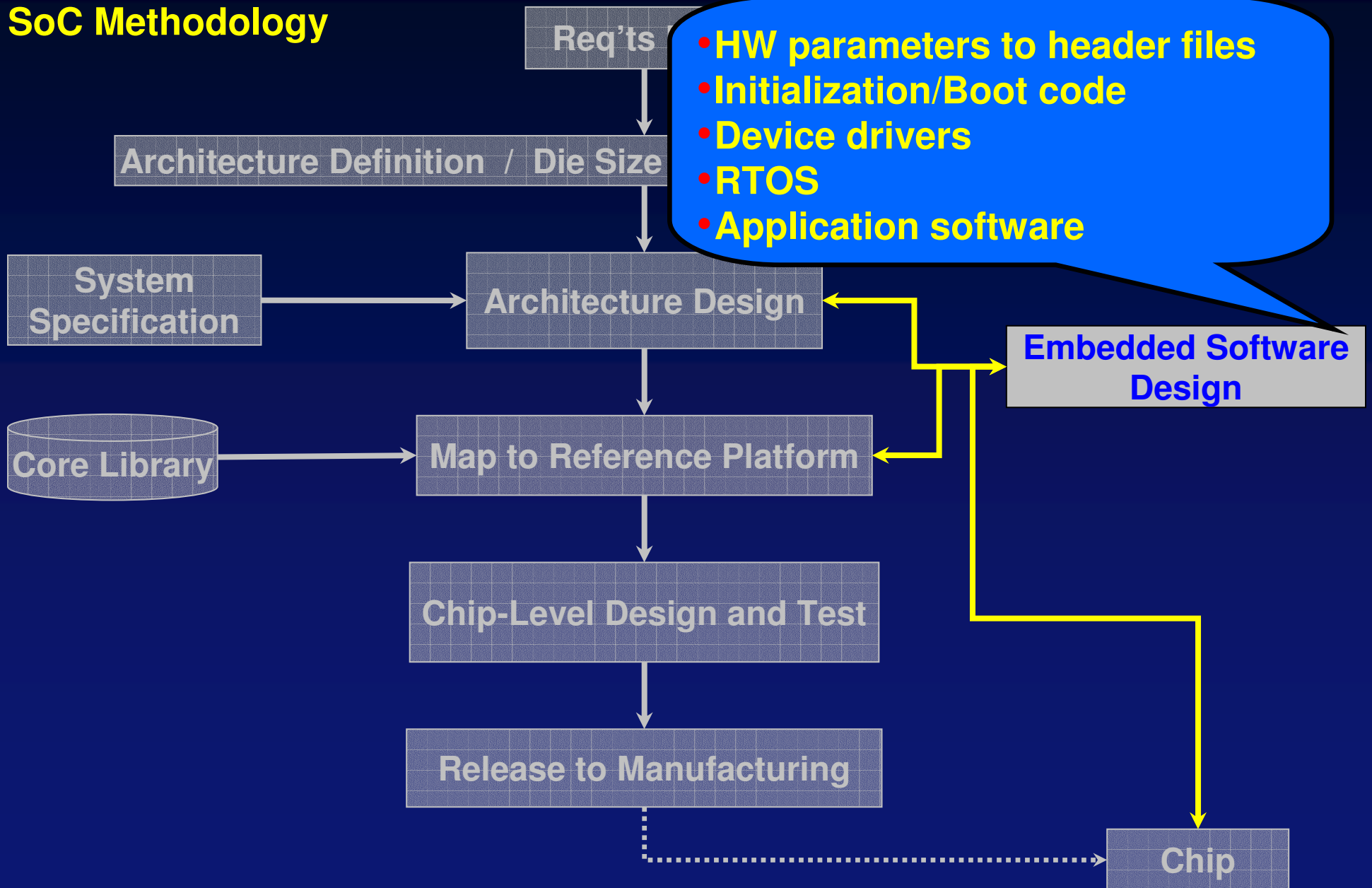
How to Design an SoC

SoC Methodology



How to Design an SoC

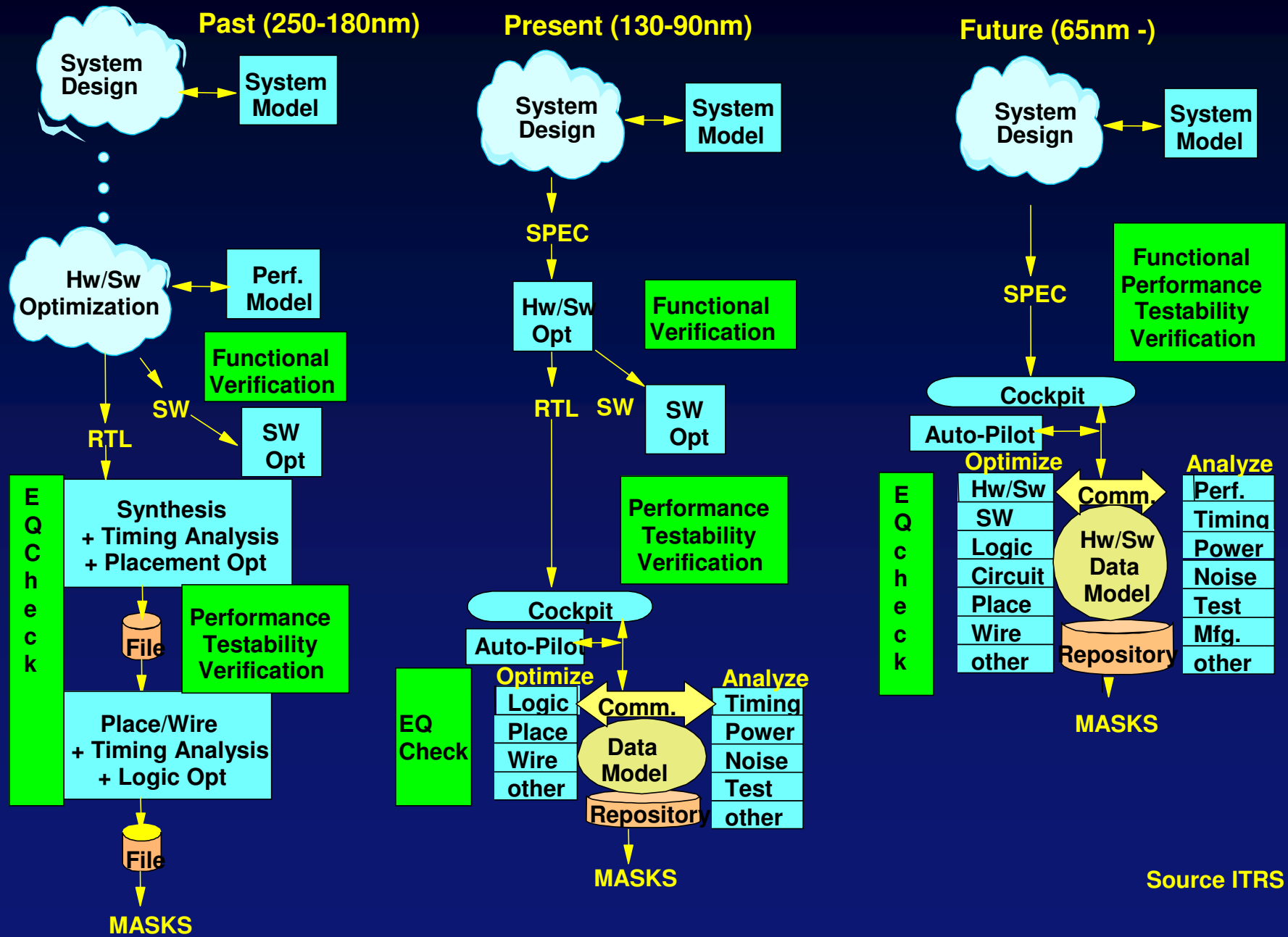
SoC Methodology



SoCs vs. ASICs

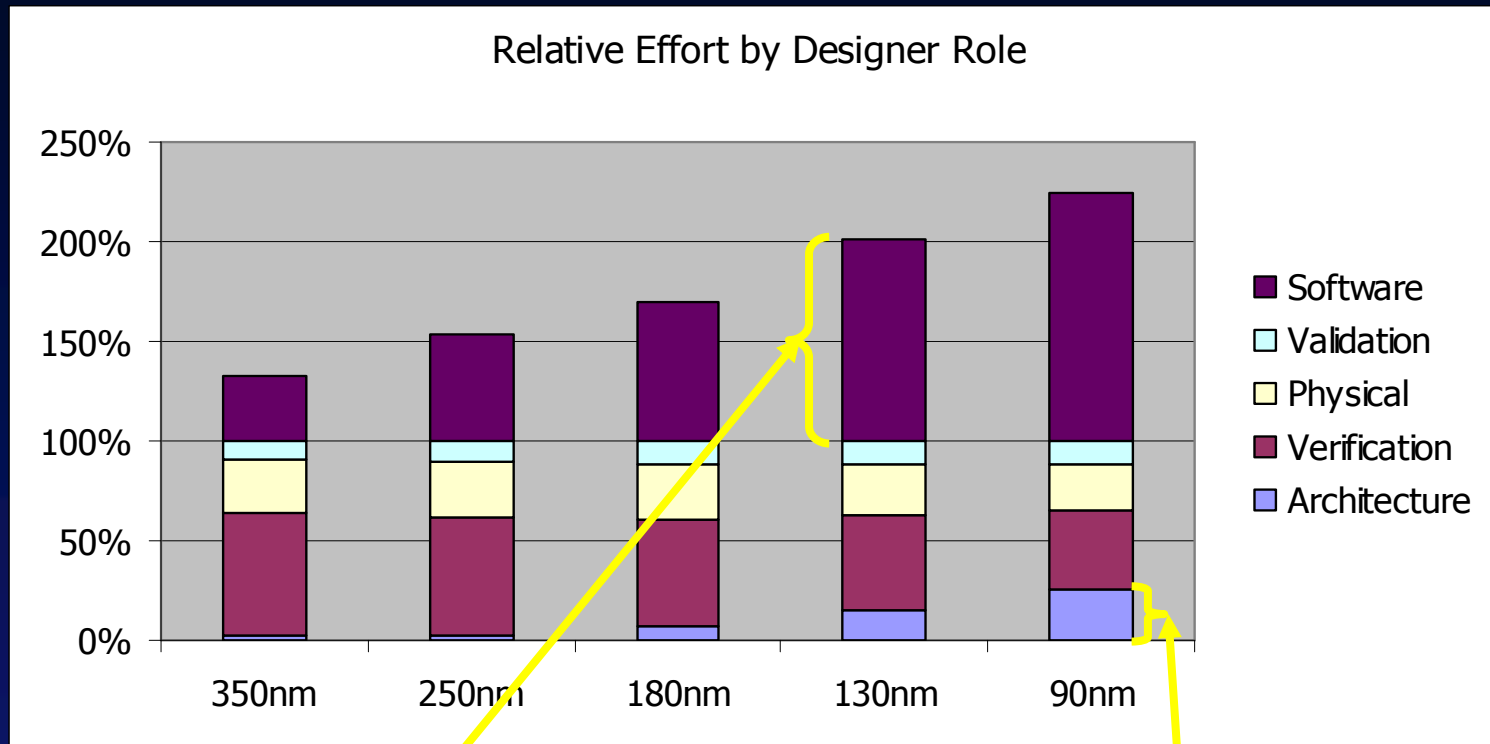
- SoC Methodology is an incremental step over ASIC Methodology
- But SoC design is significantly more complex
 - ◆ Need cross-domain optimizations
 - ◆ IP reuse and Platform-based design increase productivity, but not enough
 - ◆ Even with extensive IP reuse, many of the ASICs design problems remain, plus many more...
- Productivity increase far from closing design gap

From ASICs to Systems: Methodology Evolution



Source ITRS

Key Challenges for Systems Design



**Software effort
overtakes hardware effort
at 130 nm**

**Architecture effort
overtakes physical design
at 90 nm**

Source: IBS, November 2002, Coware Presentation

Increasing Software Complexity

Typical product, circa 1995:

100,000 lines of code. Standalone, unmanaged, fixed function



Typical product, circa 2001:

1,000,000 lines of code. Networked, managed, upgradable



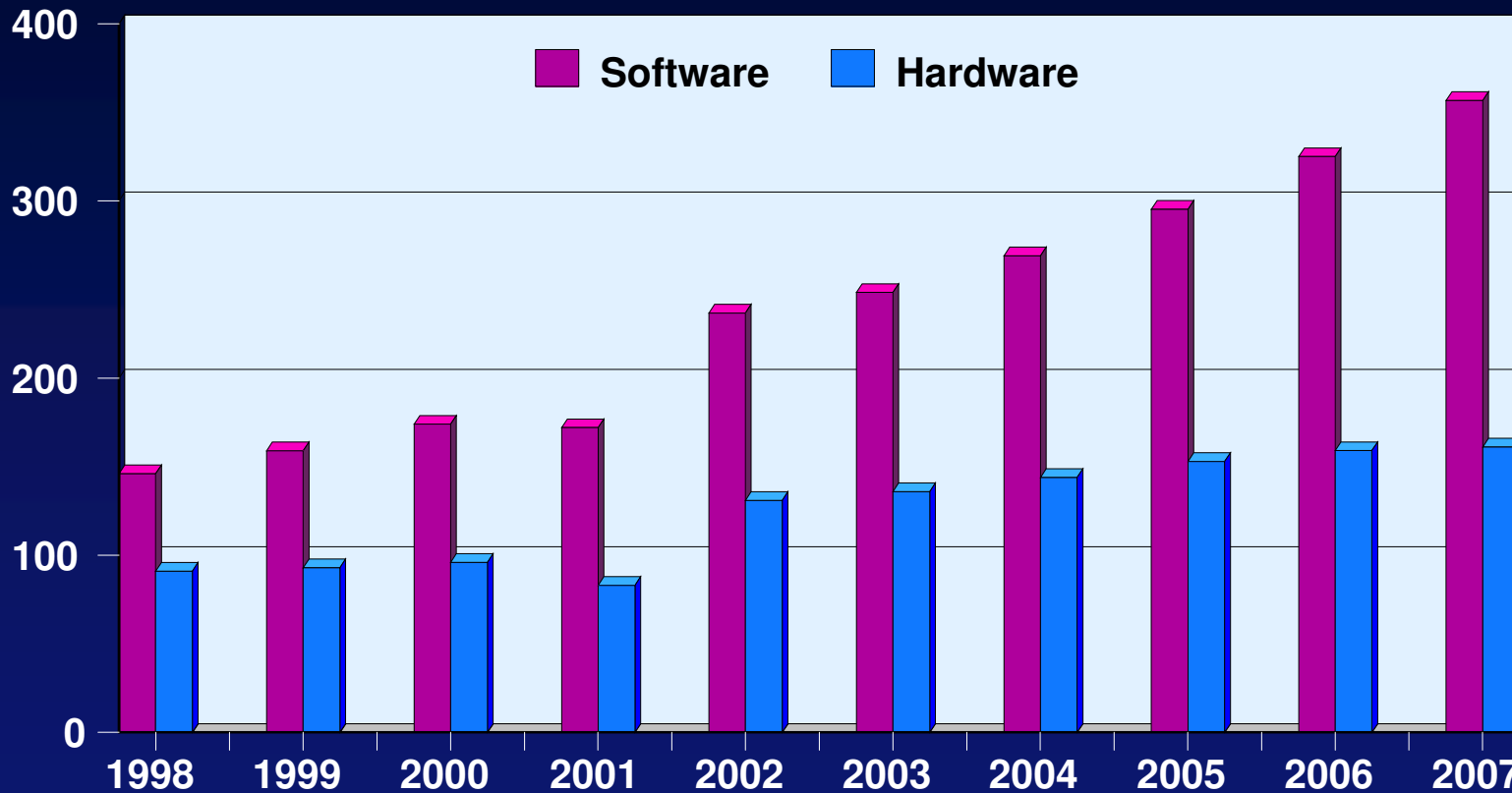
A tenfold increase in software content

Rapid growth will continue in software complexity for high value industries such as aerospace, consumer electronics, network, industrial and automobiles

Source: Wind River, BCG

Embedded Systems Developers - HW vs. SW

Worldwide Population of Embedded Software and Hardware Developers, 2002-2007 (Thousands of Developers)

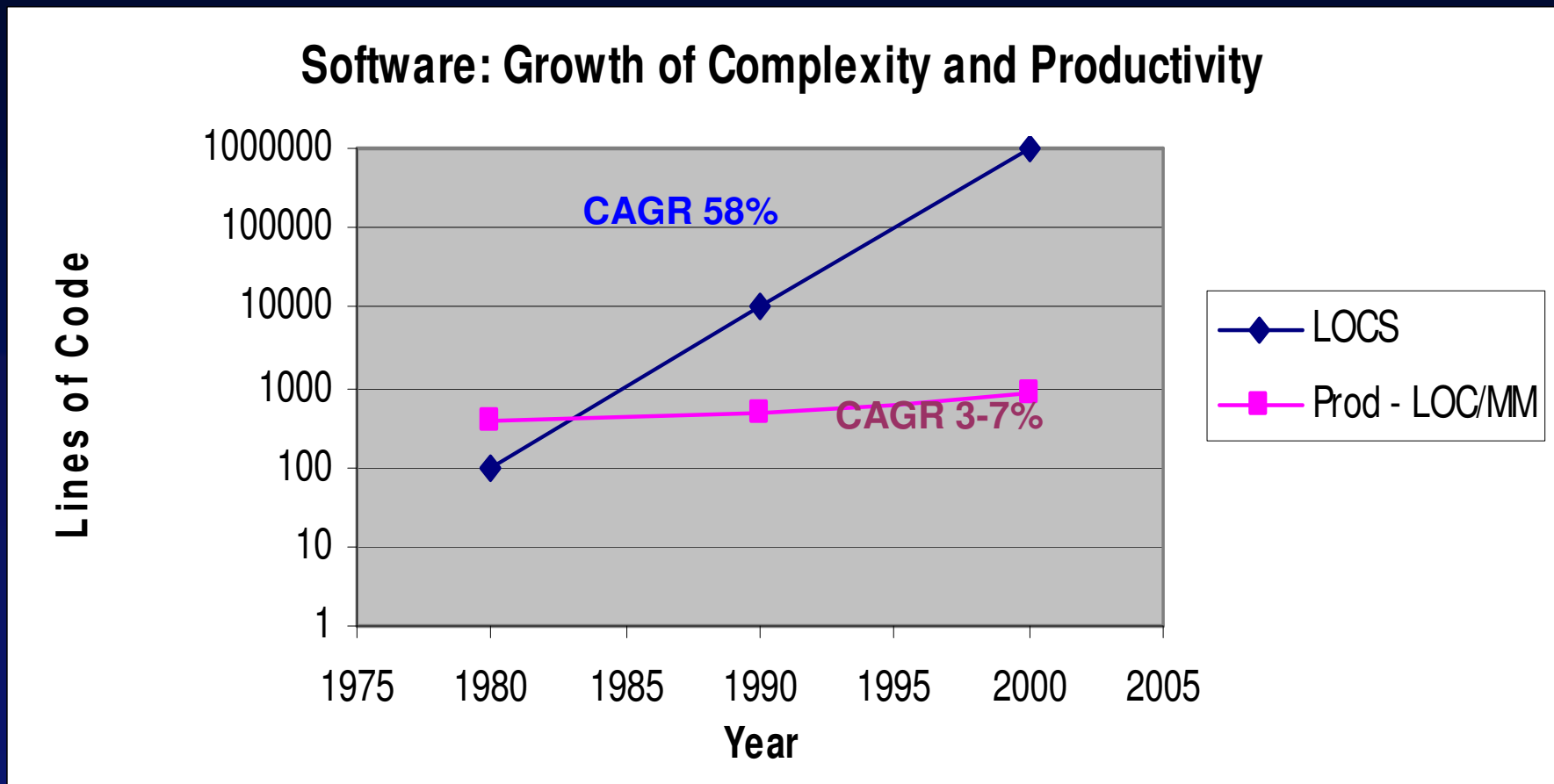


Software developer population growing at 8.3% vs 4.0% for hardware

Courtesy of Jack Kouloheris, IBM Research

Source: VDC Embedded Systems Market Analysis 2002/3

Software: Growth of Complexity and Productivity

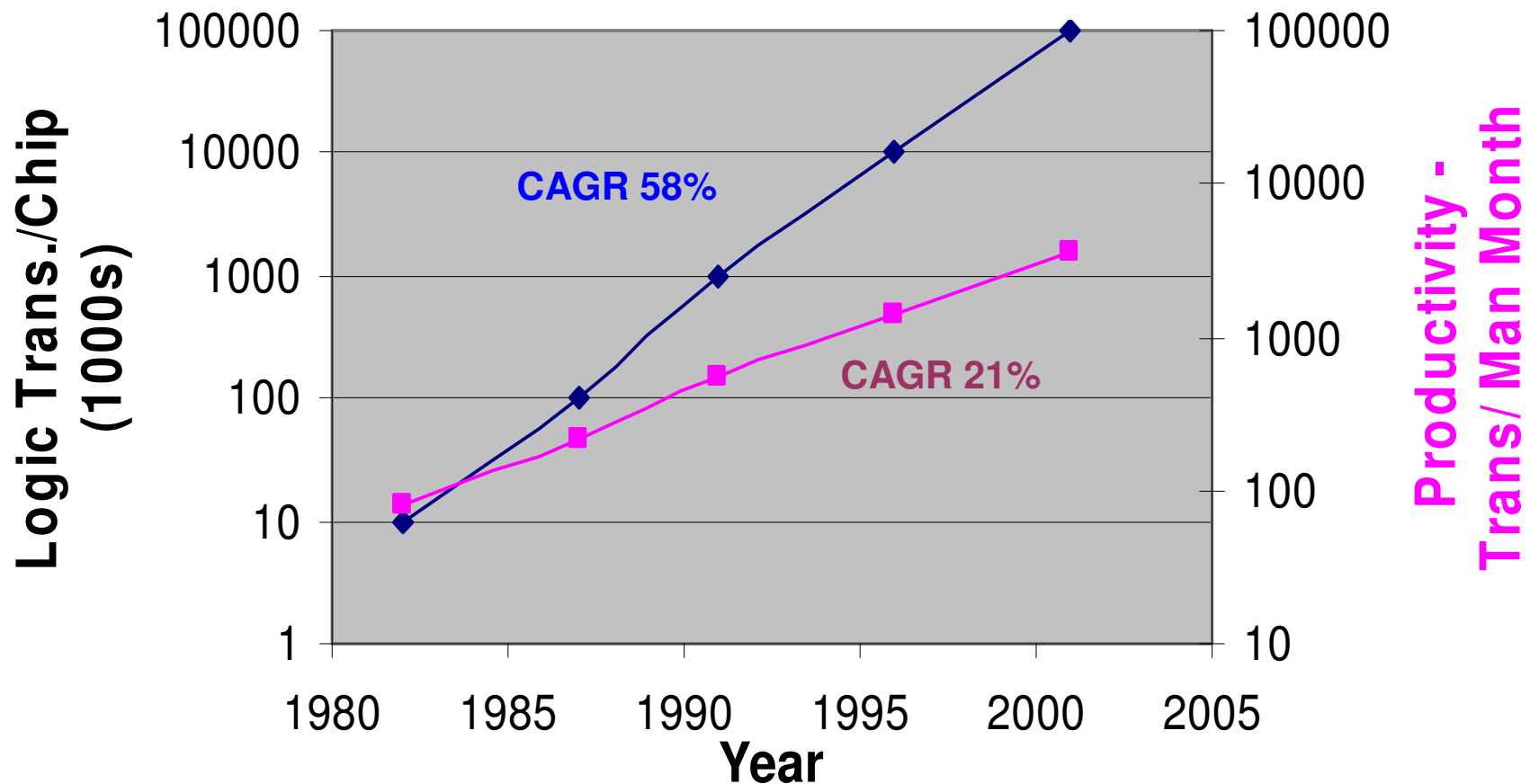


Source: Applied SW Measurement (Capers Jones), IEEE

Courtesy of Jack Kouloheris, IBM Research

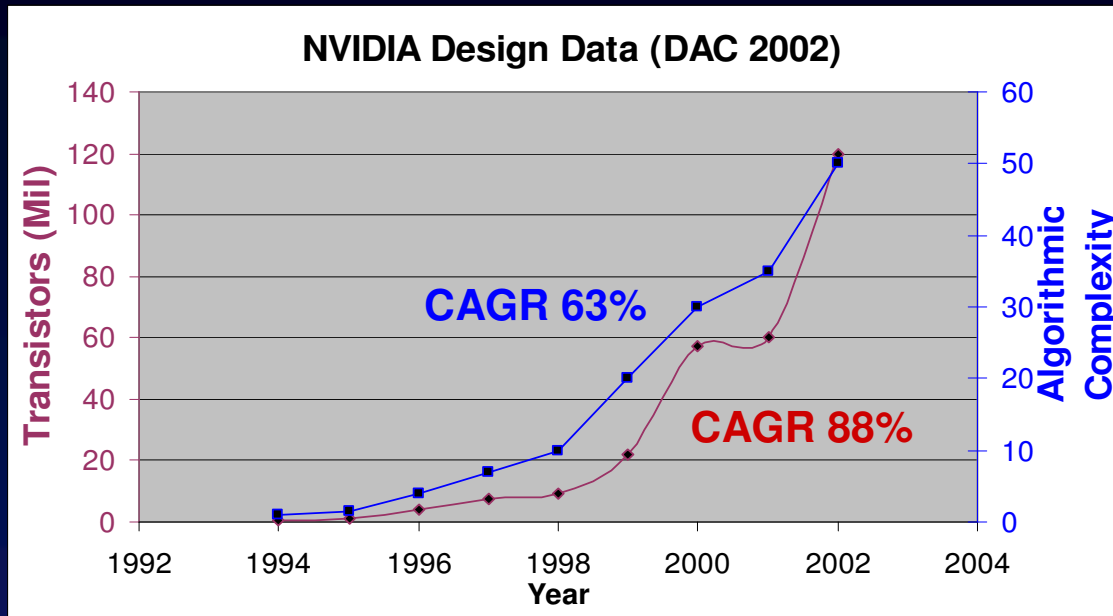
Hardware: Growth of Complexity and Productivity

Hardware- Chip Complexity and Productivity Growth



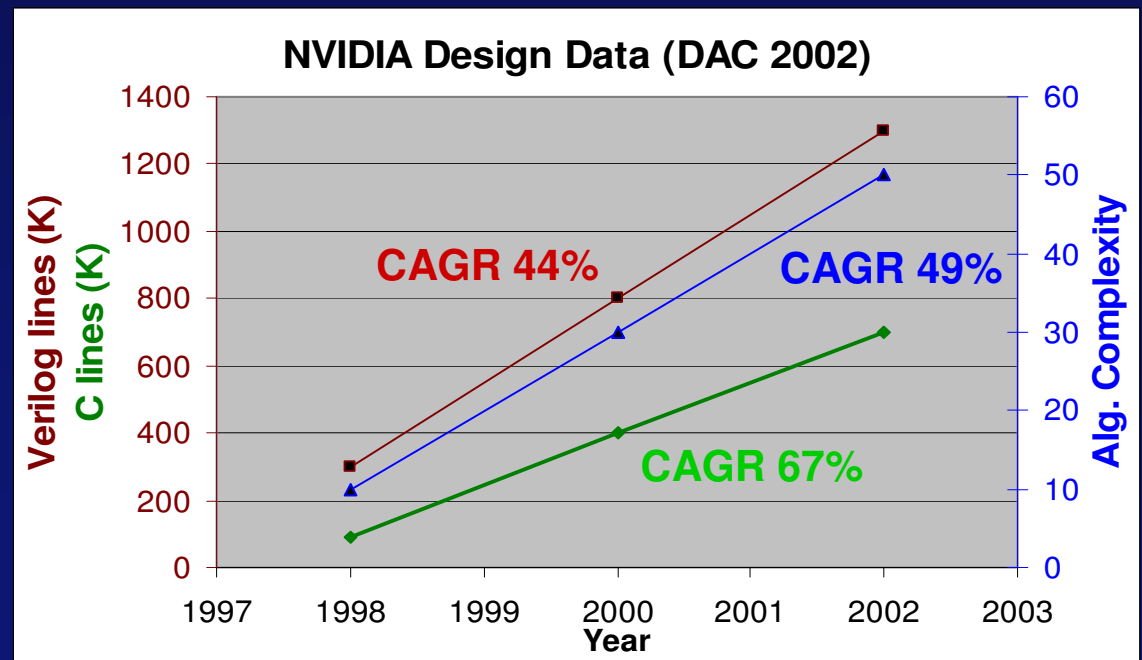
Source: Sematech

Productivity issues



- Low design (21% CAGR) productivity is usually blamed on tools

- Algorithmic complexity keeps growing (>49% CAGR)
- No tools to help



The Wisdom of Designing HW as SW (or lack thereof)

- **Systems (and SOCs) becoming increasingly programmable**
 - ◆ Support multiple standards
 - ◆ Differentiation is more in SW (less in expensive HW)
 - ◆ “Faster time to market”
- **General trend in making HW design mimic SW design**
- **Why ☺?**
 - ◆ SW design is easier than HW design
 - ◆ HW designers are fed up with bad tools and want something as straightforward as writing a program
 - ◆ Lots of programmers around, not enough good HW designers
 - ◆ HW and SW design are similar on the surface, but significantly different when it comes to implementation

Raising the Level of Abstraction

- Our favorite approach to raising productivity
 - ◆ From Gate-Level to RTL
 - ◆ From RTL to Behavioral
 - ◆ From Behavioral to C++/SystemC/SpecC, Transaction-level
- How much productivity have we really gained by raising the abstraction level?
 - ◆ Is it a problem of the approach
 - ◆ Or is it really a tools problem?
- Problems with current approaches:
 - ◆ C++/SystemC is a complex language
 - ◆ Code-reuse is non-trivial, libraries inflate rapidly, etc.
 - ◆ Maybe cleanly designed languages for specific applications (e.g., data path, pipelines, controllers) are better

The Language Fallacy

- **Myth**

- ◆ Raising the abstraction level of the input description will significantly increase productivity

- **Reality**

- ◆ It does help but up to a point (not enough to close the productivity gap!)
- ◆ Describing a complex chip is hard enough in any language
- ◆ Main design problems remain:
 - Verification (Functional, Performance, HW + SW)
 - Timing closure
 - Architecture design

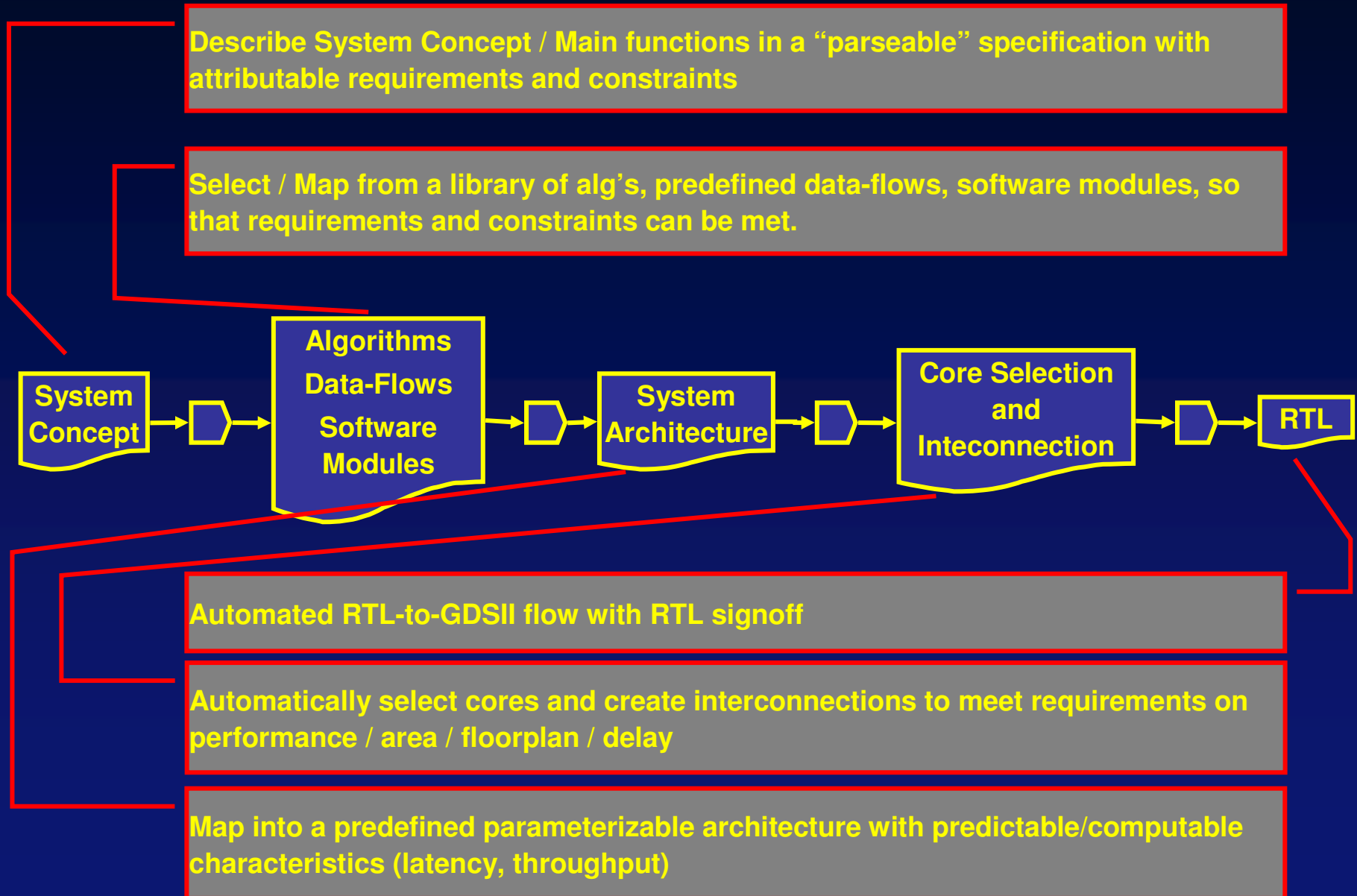
Problems with Existing Approaches

- **Specification complexity not decreasing**
 - ◆ Higher abstraction levels not catching up with higher design complexity
 - ◆ Need for new specification abstraction ideas that reduce complexity and ease the verification burden
- **Design stages still disconnected**
 - ◆ Data is not always automatically passed down
 - ◆ Manual intervention needed in several steps
 - ◆ Decisions made in one stage not always satisfied by lower stages
 - ◆ Design iterations must be avoided at all costs
- **Tools are not able to predict their effects down the line**
 - ◆ Design predictability needed to avoid bad early decisions
 - ◆ Better understanding of cross-domain optimizations needed

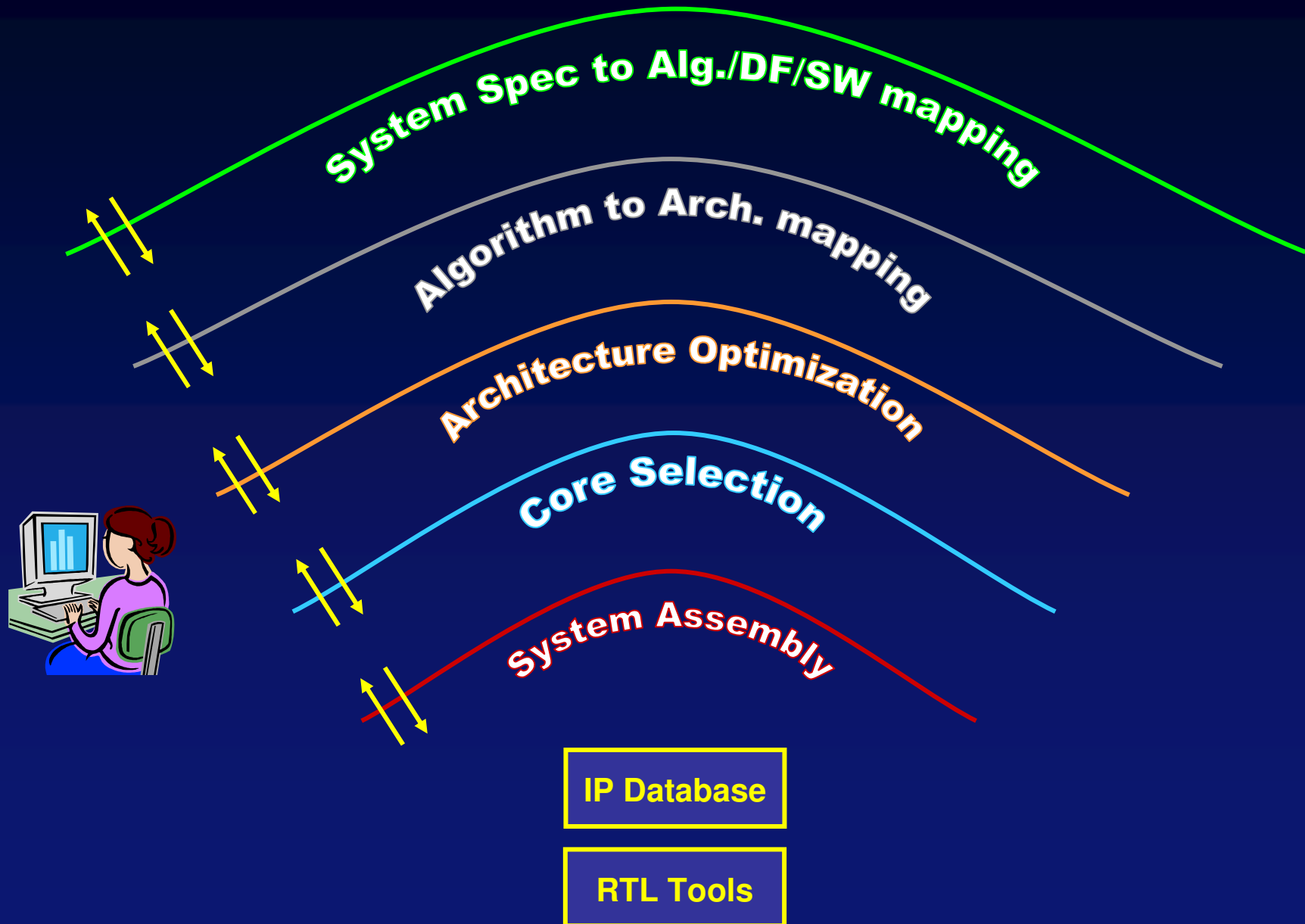
Radical Approaches Needed

- Specify a design by its main functions only, NOT by everything it has to do! Reduce complexity!
- Reliable IP-based, Platform-based, Application-based design style. Maximize reuse, minimize new custom logic. HW+SW IP blocks needed.
- Much more interoperability needed, to be able to leverage multiple IP repositories
- Automation at every step (not manual refinement w/ verification)
- Predictors, Estimators that can look ahead several design decisions

Radical Approaches Needed



SLD Tools at IBM Research



Pins Properties for Interconnection

- **BUS_TYPE** : PLB, OPB, AMBA, APB, ...
- **INTERFACE_TYPE** : Master, Slave, ...
- **FUNCTION_TYPE** : Read, Write, Interrupt, ...
- **OPERATION_TYPE** : Request, Acknowledge, ...
- **DATA_TYPE** : Address, Instruction, Data, ...
- **RESOURCE_TYPE** : Bus, Peripheral, ...
- **PIN_GROUP** : M0, M1, ICU, DCU, ...

Reasoning about Cores and Pins Properties

- Properties are encoded using Boolean equations, represented as Binary Decision Diagrams (BDDs)
- Relationships among cores and pins can be established by comparing Boolean functions
- Two pins with “compatible” properties may be connected together
 - ◆ functionally, structurally and electrically compatible

CORAL's Virtual Design Representation

- Block Diagram
 - ◆ From "back-of-the-envelope" to a synthesizable description
- Virtual Design
 - ◆ Structural and functional encapsulation of the real design
 - ◆ Virtual Components, Virtual Interfaces, Virtual Nets

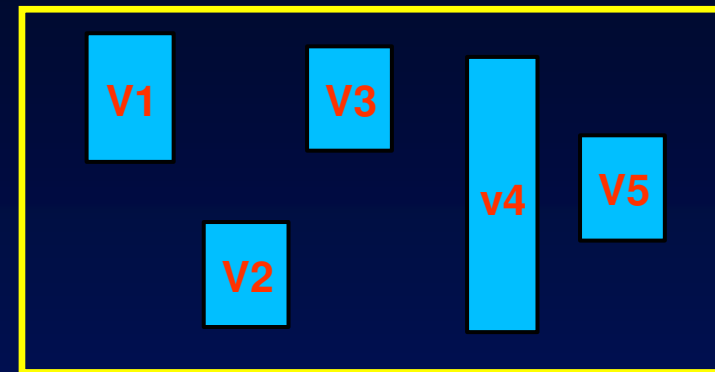
```
Entity PPC_VC is    - - Virtual Power PC
Port (
  PLB_M_ICU_interface    : in std_logic;
  PLB_M_DCU_interface    : in std_logic;
  ISOCM_interface        : in std_logic;
  DSOCM_interface        : in std_logic;
  APU_interface          : in std_logic;
  RESET_interface        : in std_logic;
  INTERRUPT_interface    : in std_logic;
  CLOCK_interface        : in std_logic;
  DCR_interface          : in std_logic;
  JTAG_interface         : in std_logic );
```

- Virtual PowerPC
10 interfaces
- Real PowerPC 405
215 interfaces

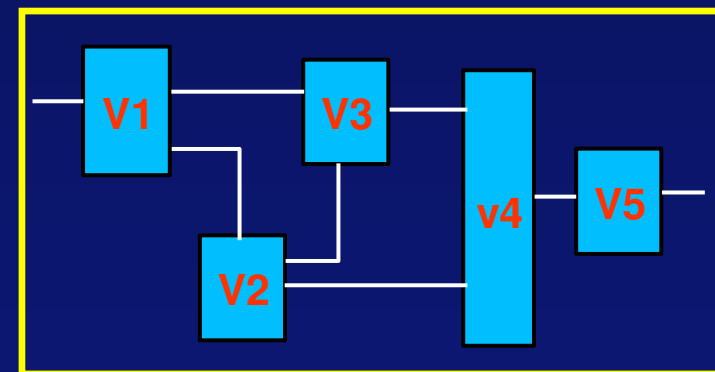
Configuration and Integration Steps

- User selects virtual cores
- By querying properties, tool displays only cores/pins associated with a given configuration
- User defines
 - ◆ Masters and Slaves
 - ◆ Address maps
 - ◆ Interrupt priorities
 - ◆ DMA channel assignments
 - ◆ Clock domains
 - ◆ I/O mapping
- Tool generates Virtual Design Automatically!
- Guide the user through configuring the design

Virtual Cores

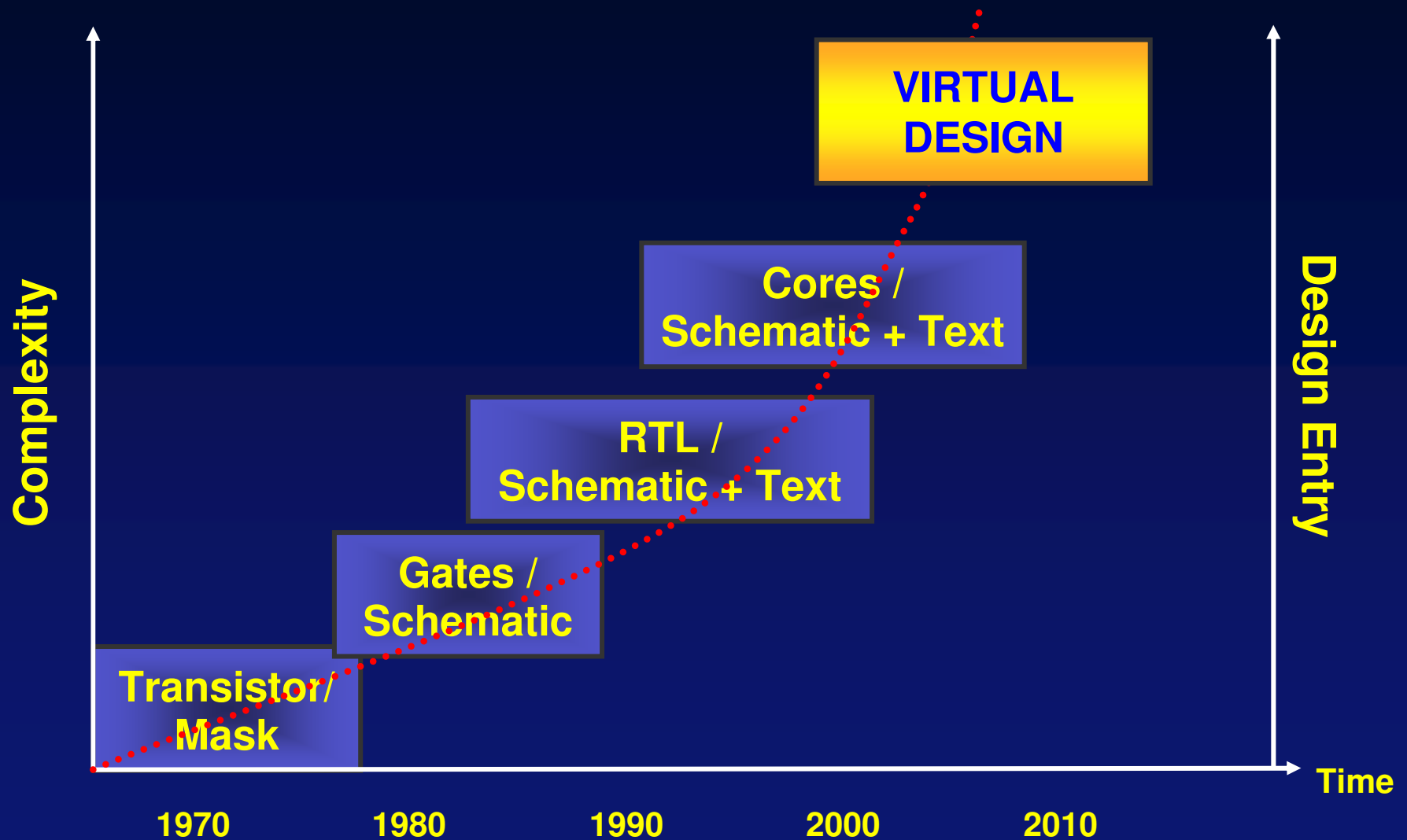


Configuration Wizard



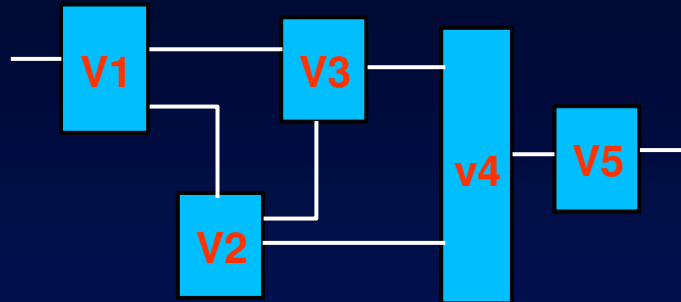
Virtual Design

Next Design Entry Evolution

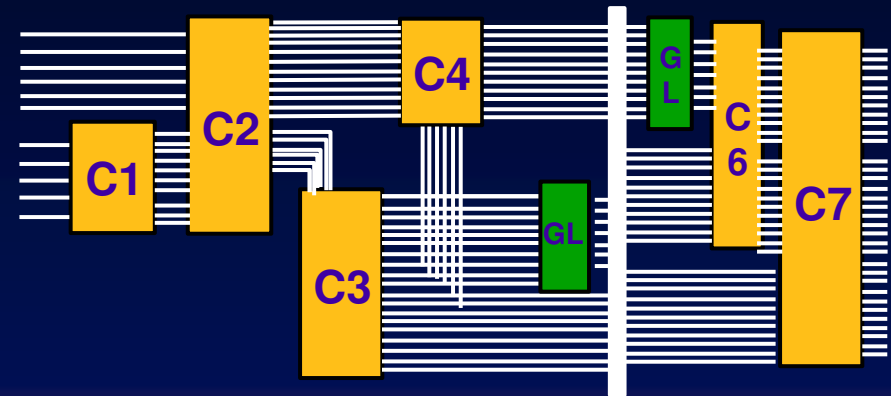


Real Design Generation

Virtual Design

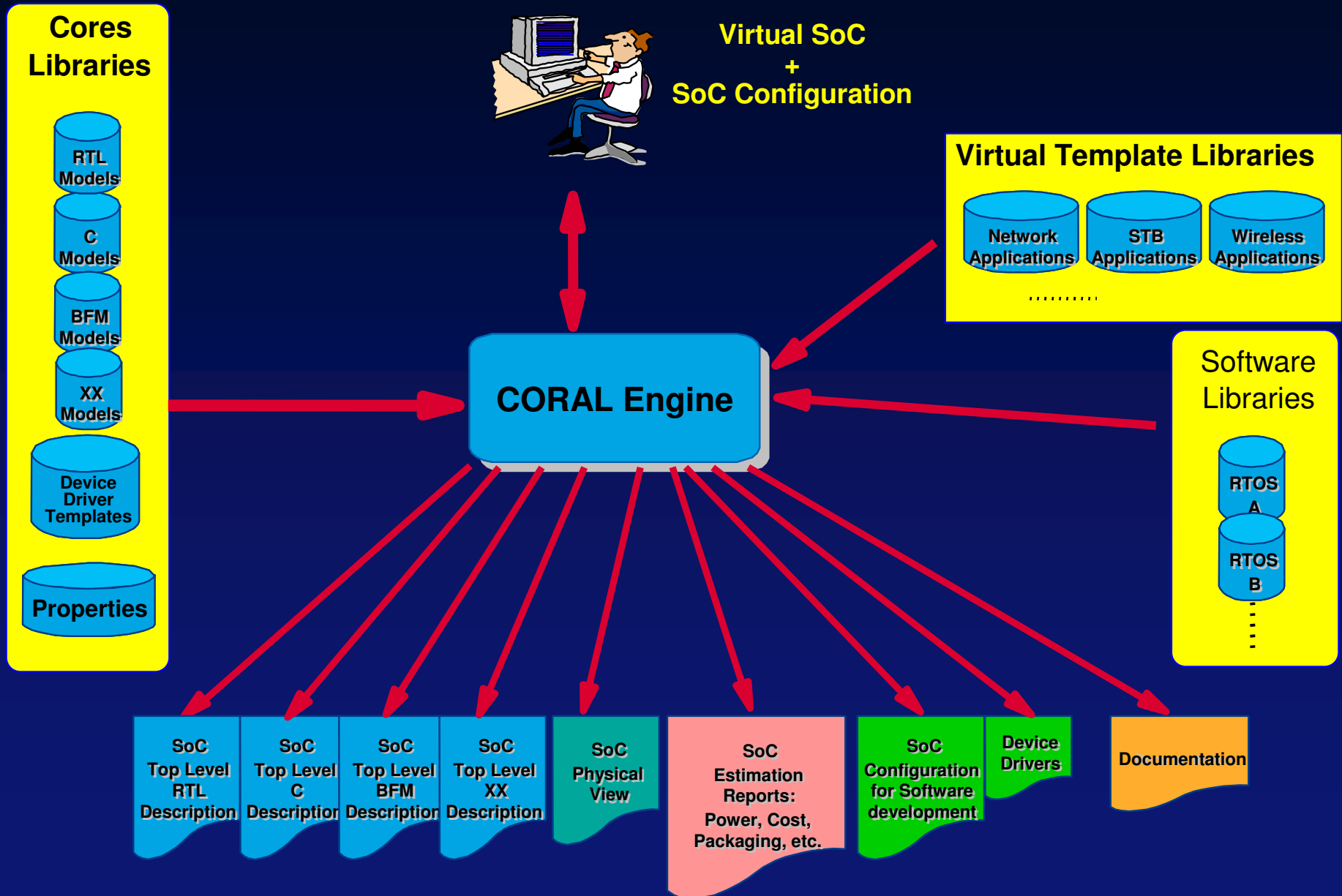


Real Design



- From a Virtual Design to a Real Design:
 - ◆ Map virtual components to real components and interface logic
 - ◆ Expand virtual pins into real pins
 - ◆ Expand virtual nets into real nets
 - ◆ Recognize the function of each virtual component, pin and net and create the real components, pins and nets implementing that function

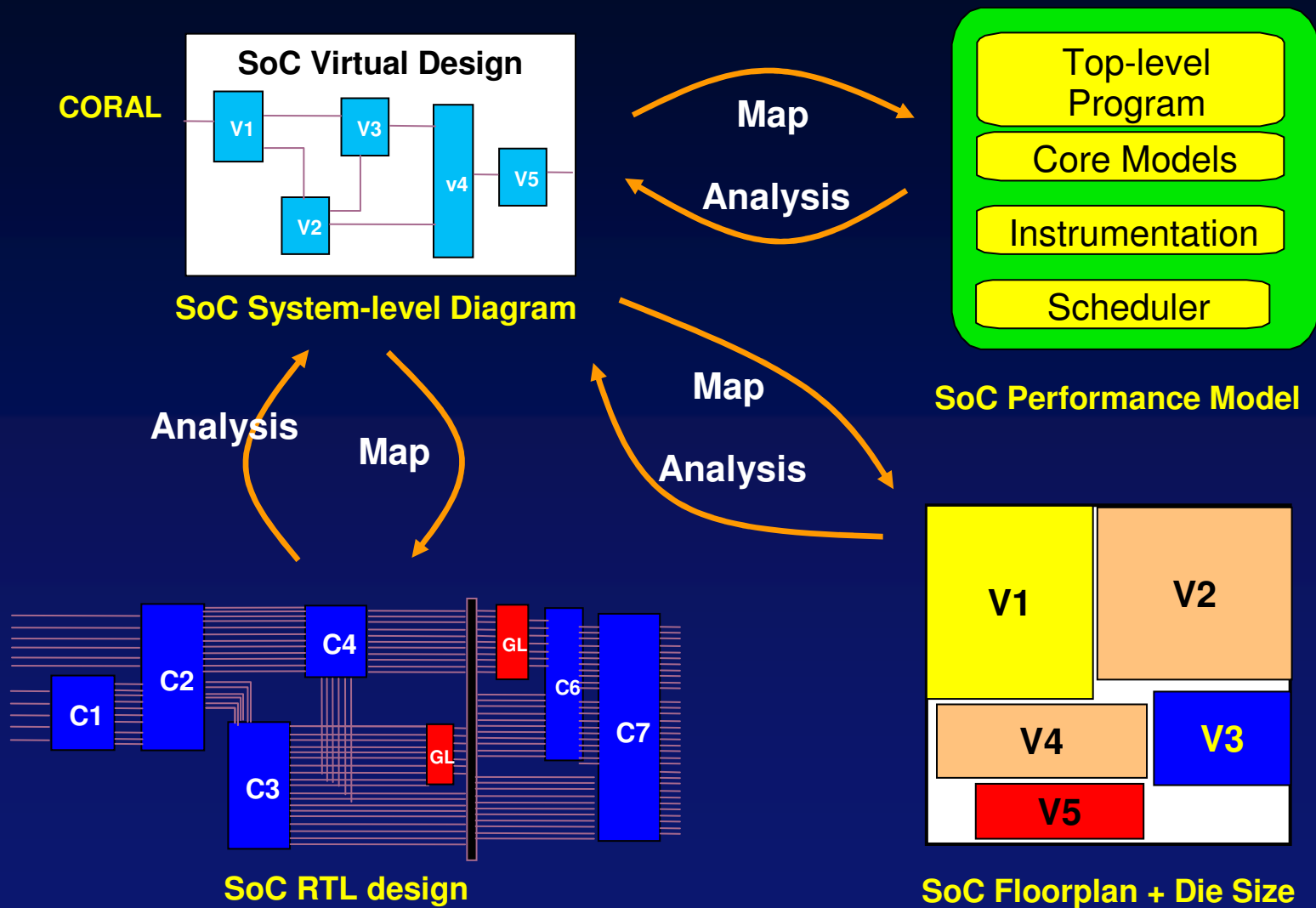
Using Coral for Systems Design



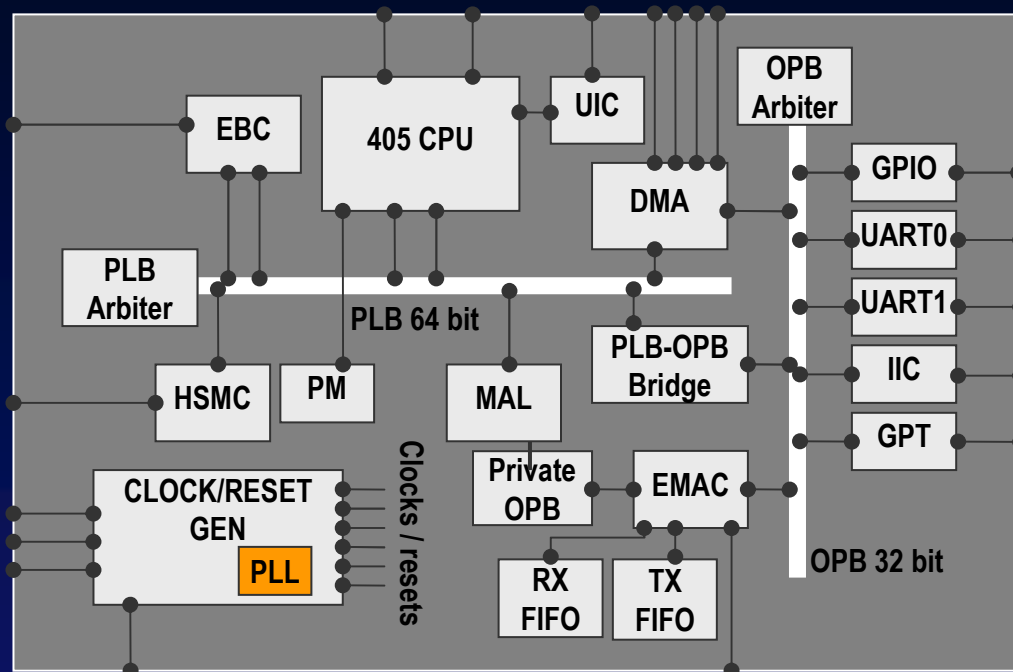
SEAS - a System for Early Analysis of SoCs

- An environment to allow early analysis of SoCs
- How early and how quickly?
 - ◆ Prior to the existence of any detailed spec (beh, rtl)
 - ◆ One day turnaround from creating the design spec to generating analyses results
 - ◆ A couple of hours turnaround in evaluating changes
- Accurate enough to allow decisions to be made on:
 - ◆ Architecture
 - ◆ Components
 - ◆ Floorplan (Area + timing)
 - ◆ Chip/Die Size and Cost
- Direct links to implementation (e.g., detailed synthesis, P&R)

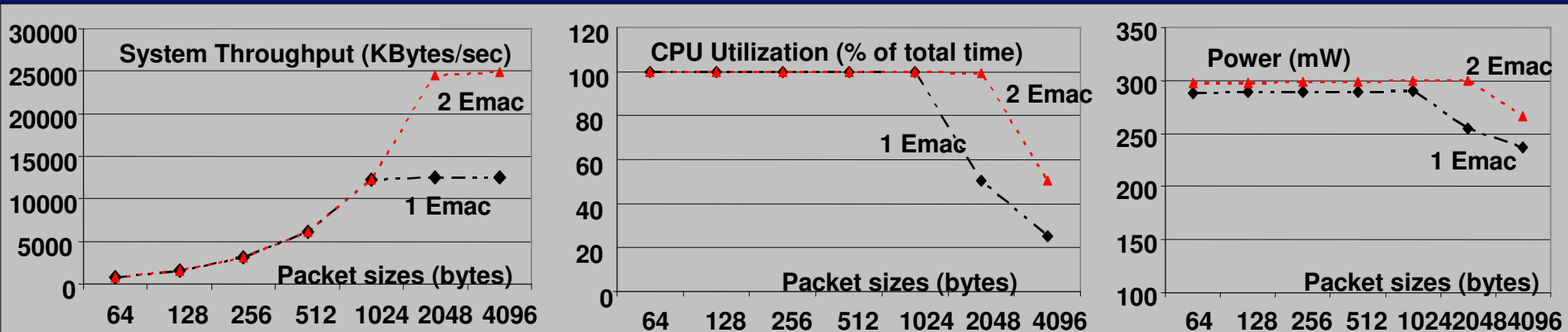
SEAS - a System for Early Analysis of SoCs



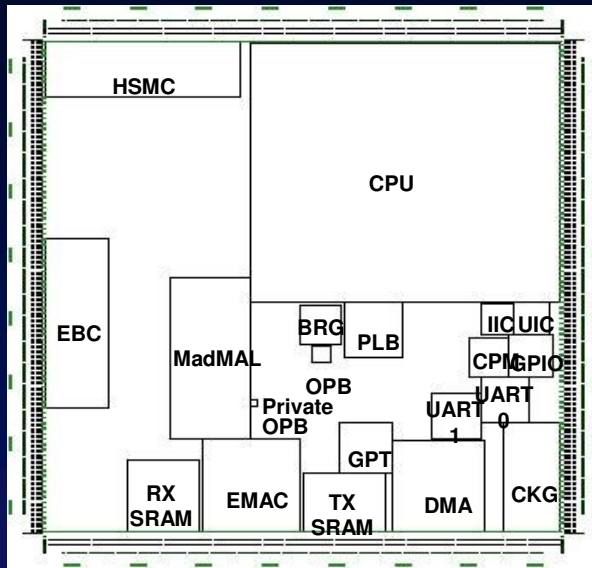
SEAS - a System for Early Analysis of SoCs



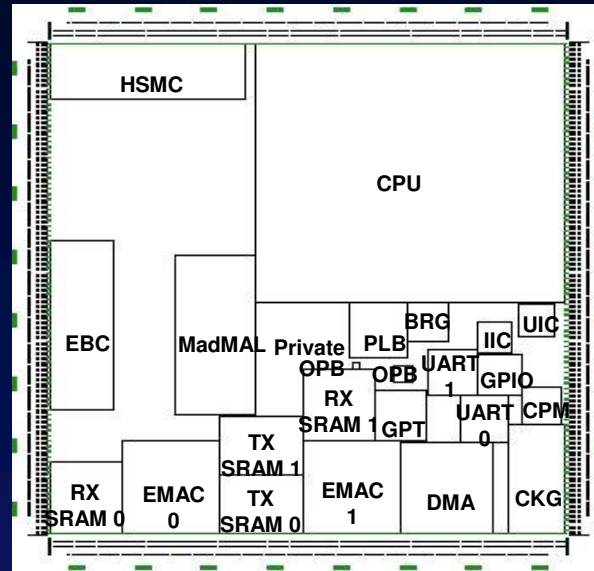
- 405PBD
 - ◆ Ethernet Subsystem
 - 1 EMAC
 - 1 Madmal
- Change to improve performance
 - ◆ Added an extra EMAC + Fifos
- Measure effects on die-size, fp, timing, power



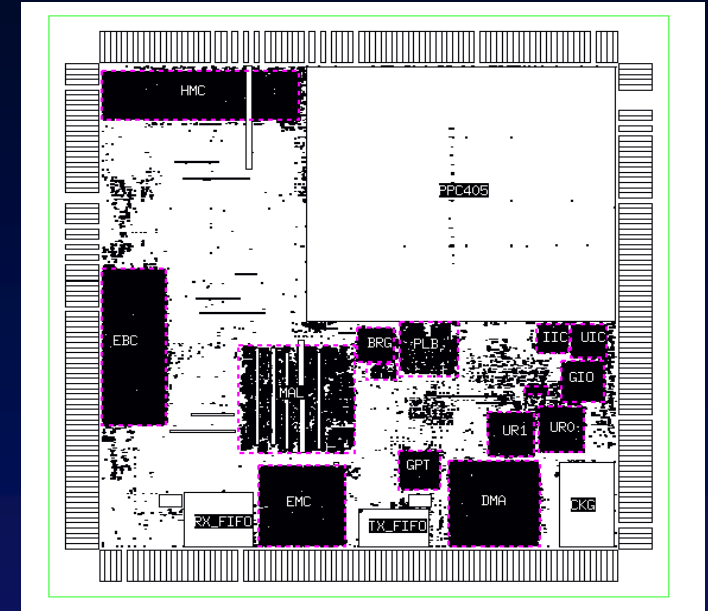
SEAS - a System for Early Analysis of SoCs



1 EMAC
6.05mmx6.05mm



2 EMACs
6.05mmx6.05mm



Real Design Layout (1 EMAC)
6.05mmx6.05mm

• Results

- ◆ Two Emac solution delivered the required performance
- ◆ Could fit in the same die-size as the original one
- ◆ Met the same timing requirements as the original one

Conclusions

- Current SoC / Systems Design tools and methodologies are incremental improvements over ASIC's
- Higher-levels of abstraction in the specification help up to a point. Design complexity is growing faster than the language's ability to express it succinctly
- Usual show stoppers (verification, timing closure, etc.) not really being addressed by system-level tools
- New (more radical) approaches are needed to cope with design complexity and productivity