Designing Programmable Platforms: From ASIC to ASIP

MPSoC 2005

Heinrich Meyr

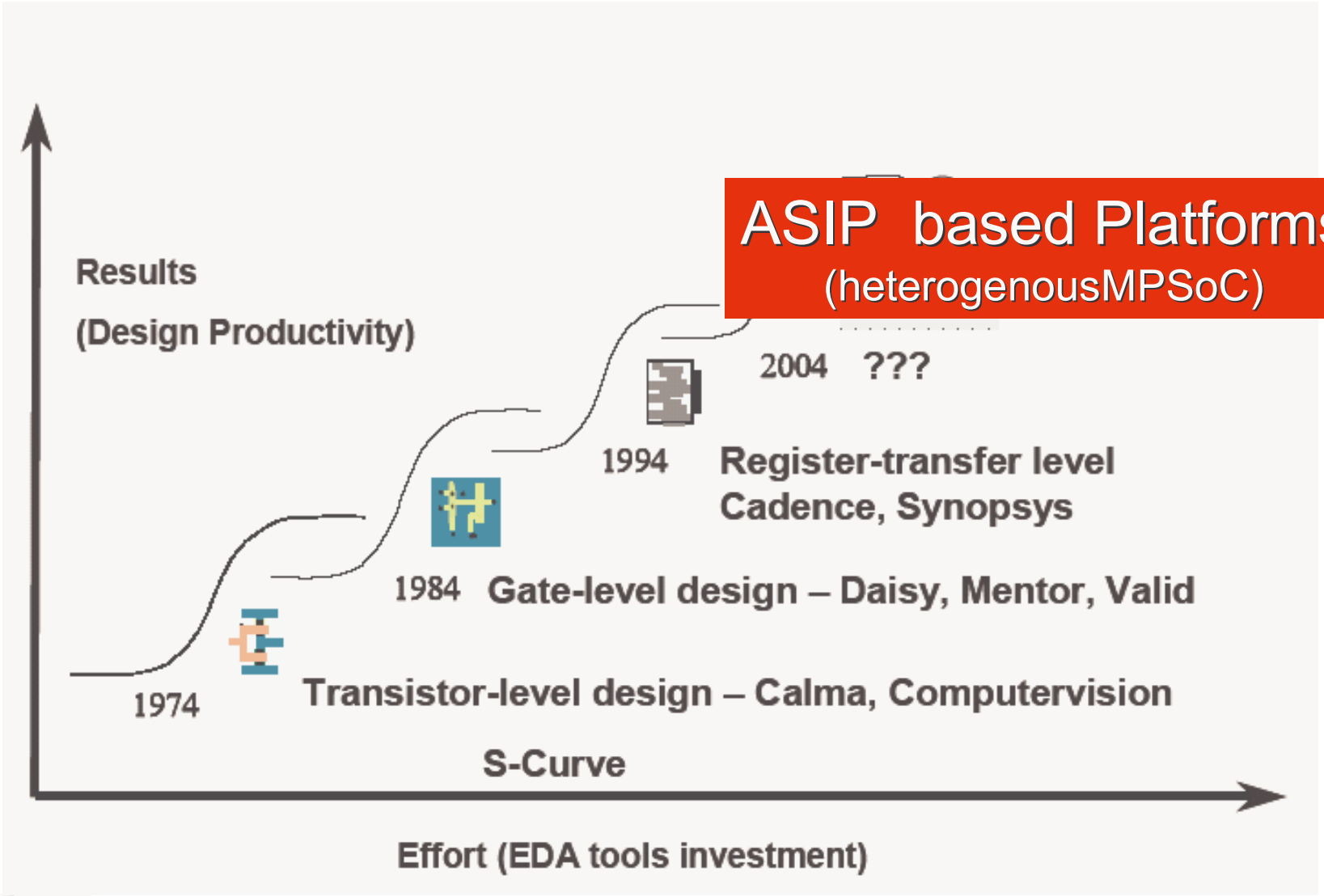
CoWare Inc., San Jose
and
Integrated Signal Processing Systems (ISS),
Aachen University of Technology,
Germany

Agenda

- Facts & Conclusions
- Heterogeneous MPSoC
  - » Energy Efficiency vs.Flexibility
  - » How to explore the Design Space?
- ASIP Design
- Economics of SoC Development
- Conclusions

Facts & Conclusion

Core Proposition



Agenda

- Facts & Conclusions
- Heterogeneous MPSoC
  - » Energy Efficiency vs.Flexibility
  - » How to explore the Design Space?
- ASIP Design
- Economics of SoC Development
- Conclusions

Trade-off between Flexibility and Energy -Efficiency

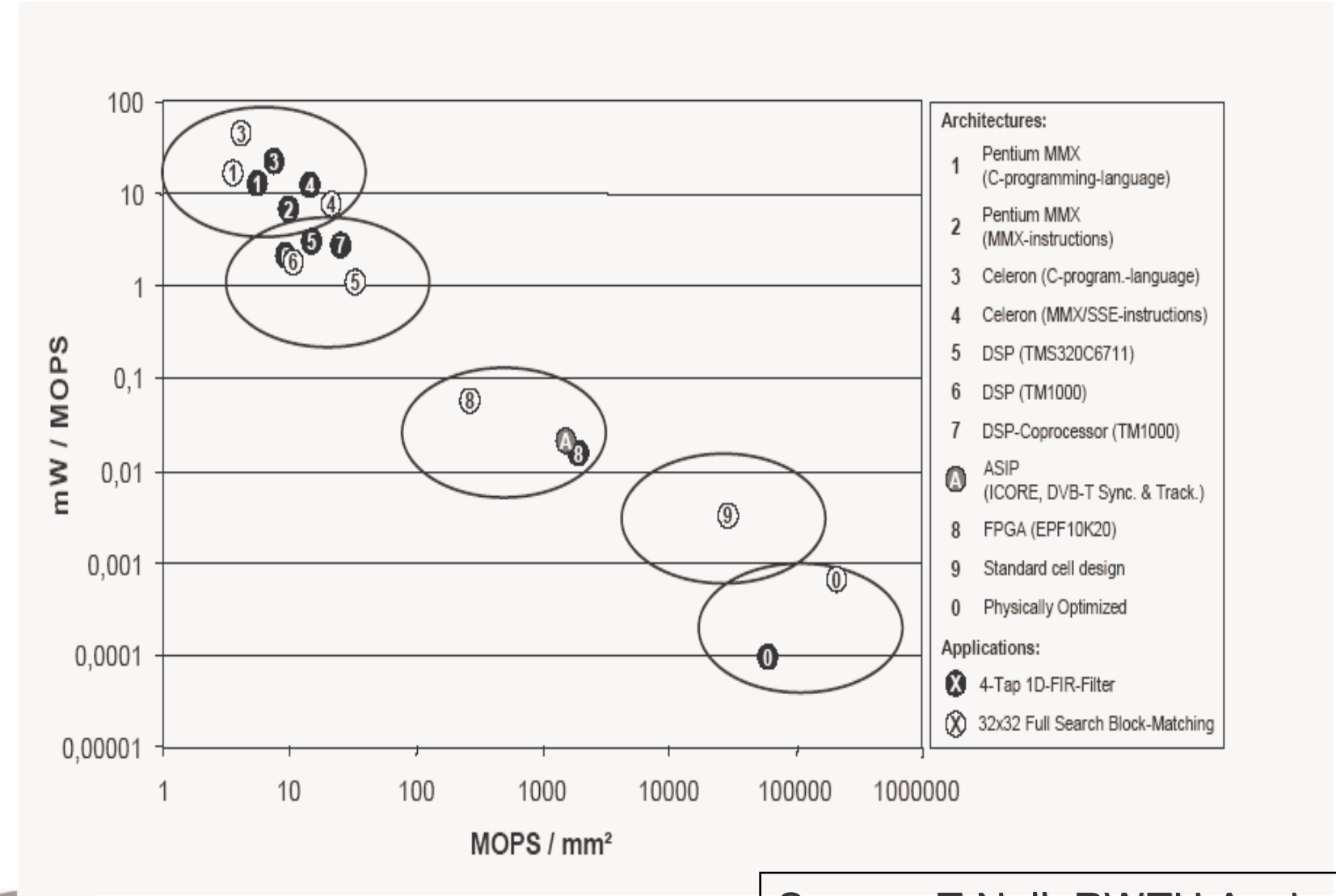
Architectural Objectives

Need more MOPS/Watt and MOPS/mm² to minimize the global performance measure for battery driven devices

Energy / decoded Bit = (Joule/Bit)

ISS

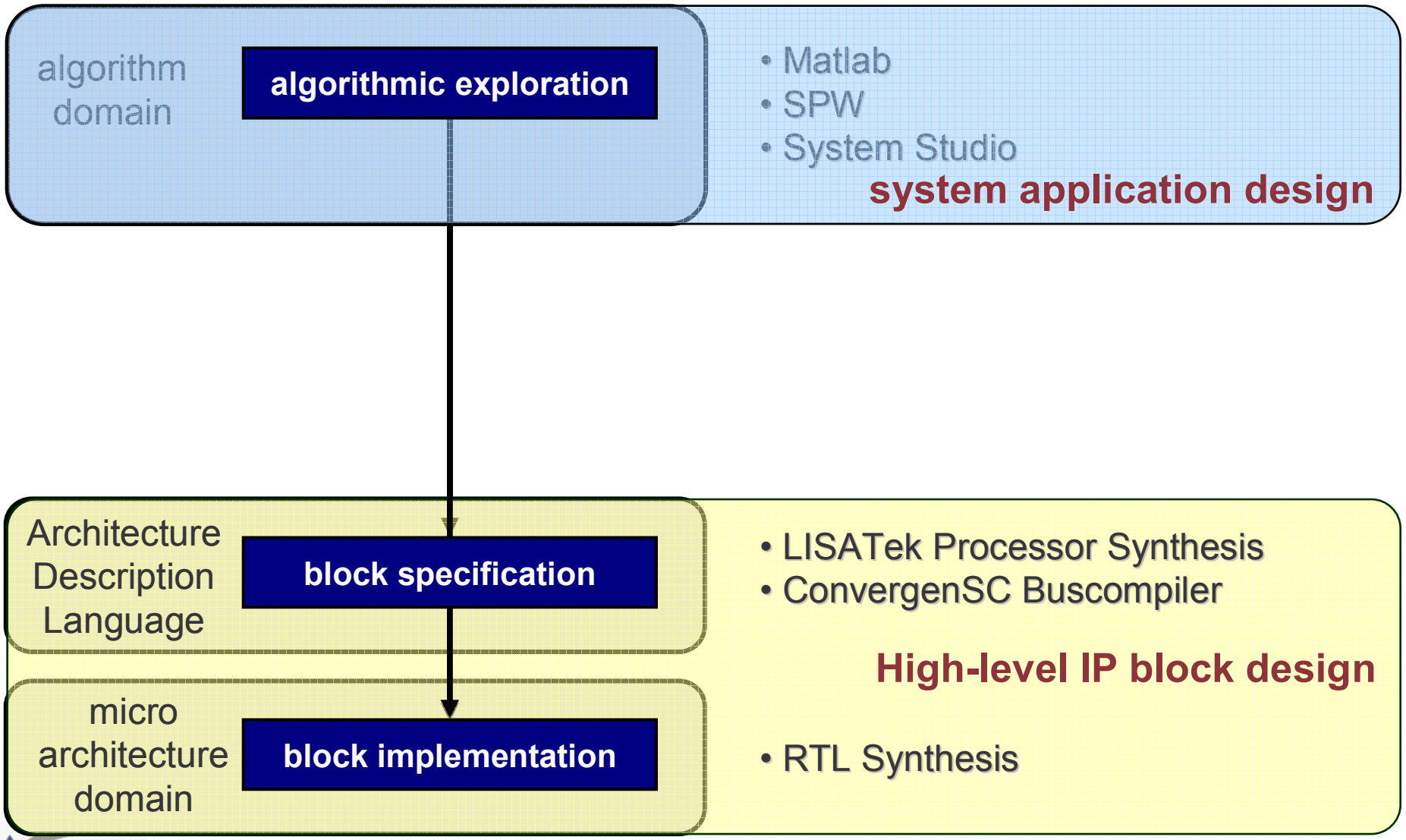
Computational Effiency vs. Flexibility



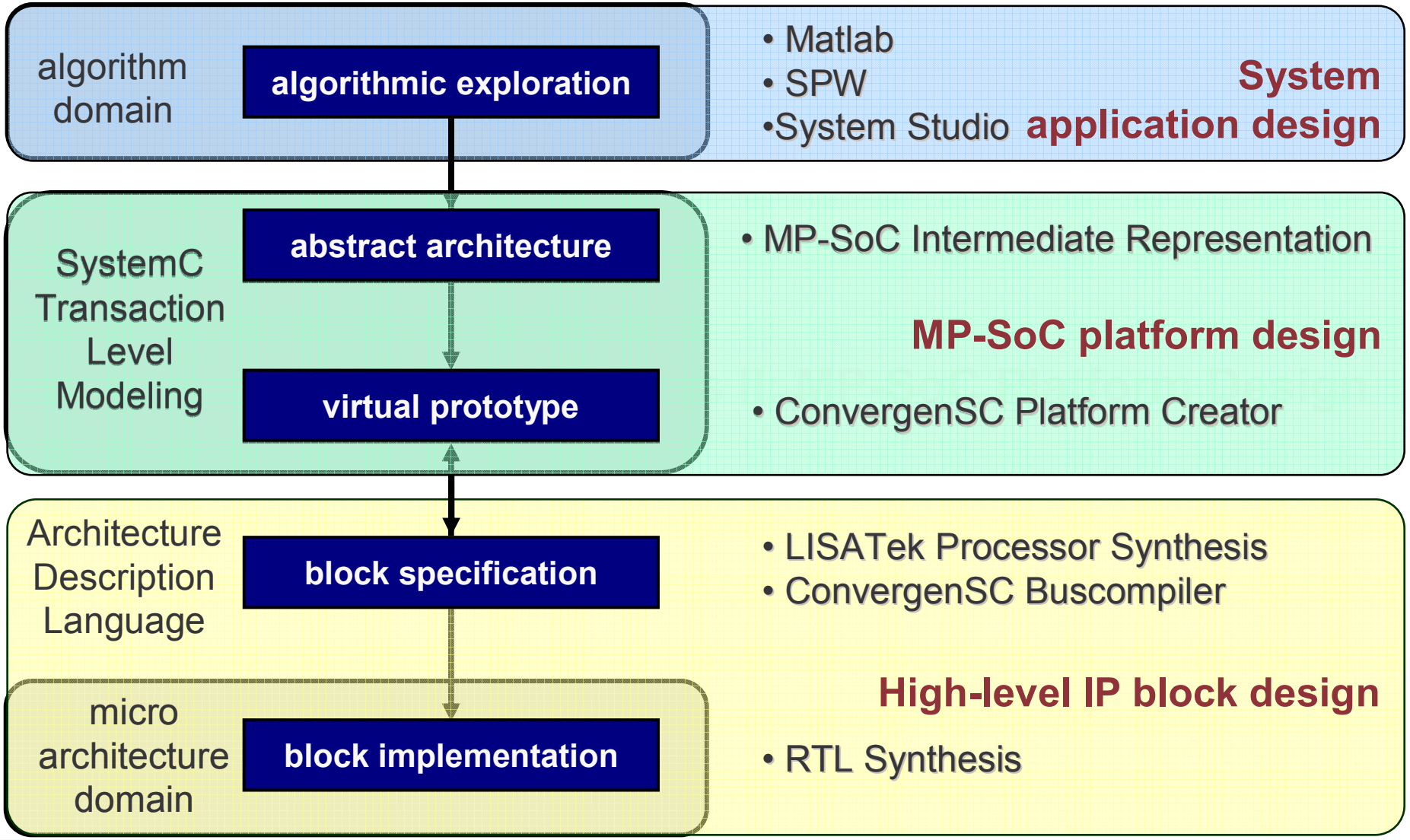
Source: T.Noll, RWTH Aachen

Enabling MP-SoC Design

System Level Tools I: Application & IP Creation



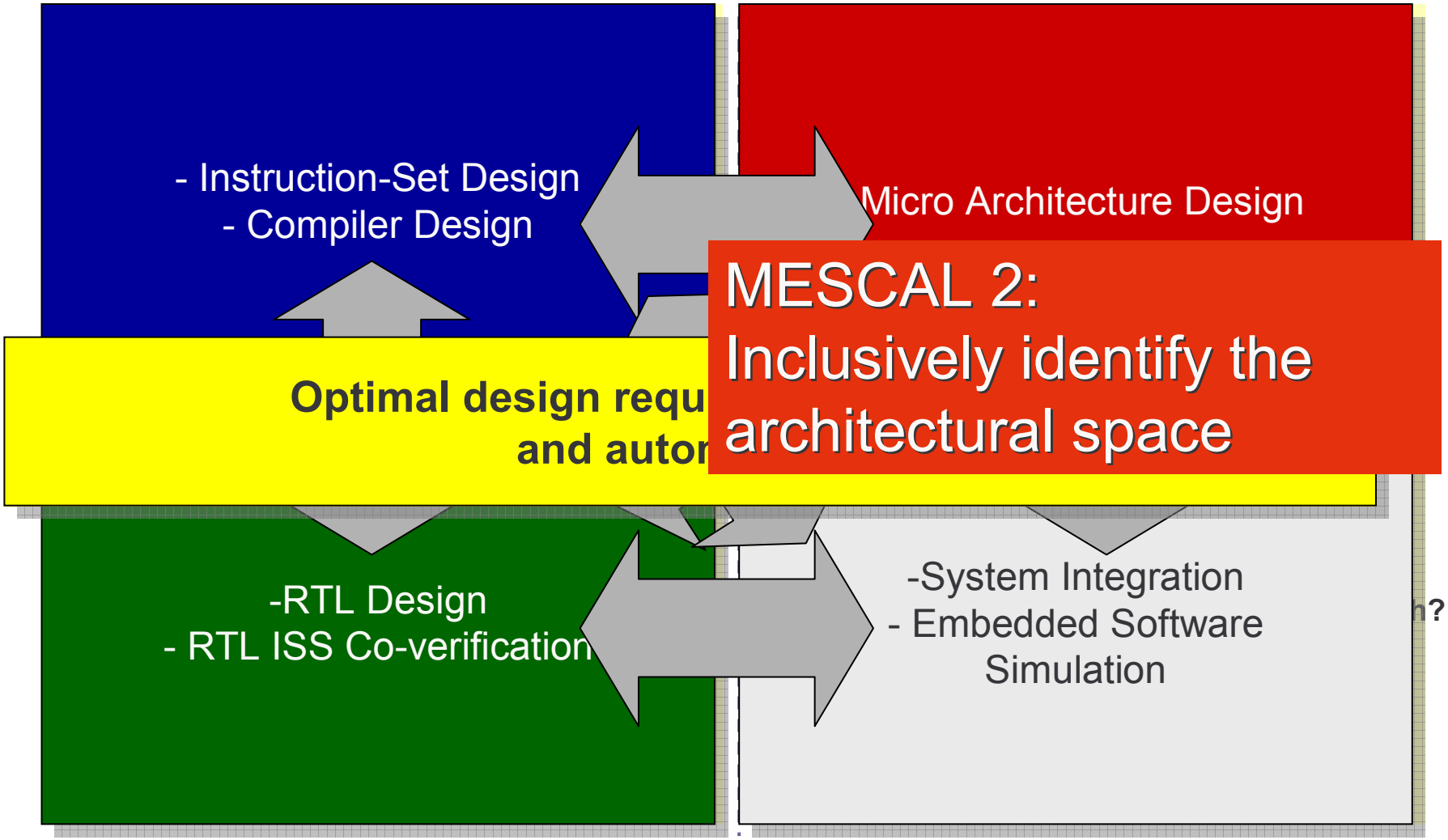
System Level Tools I: Application & IP Creation



Agenda

- Facts & Conclusions
- Heterogeneous MPSoC
  - » Energy Efficiency vs.Flexibility
  - » How to explore the Design Space?
- **ASIP Design**
- Economics of SoC Development
- Conclusions

Processor Design Space



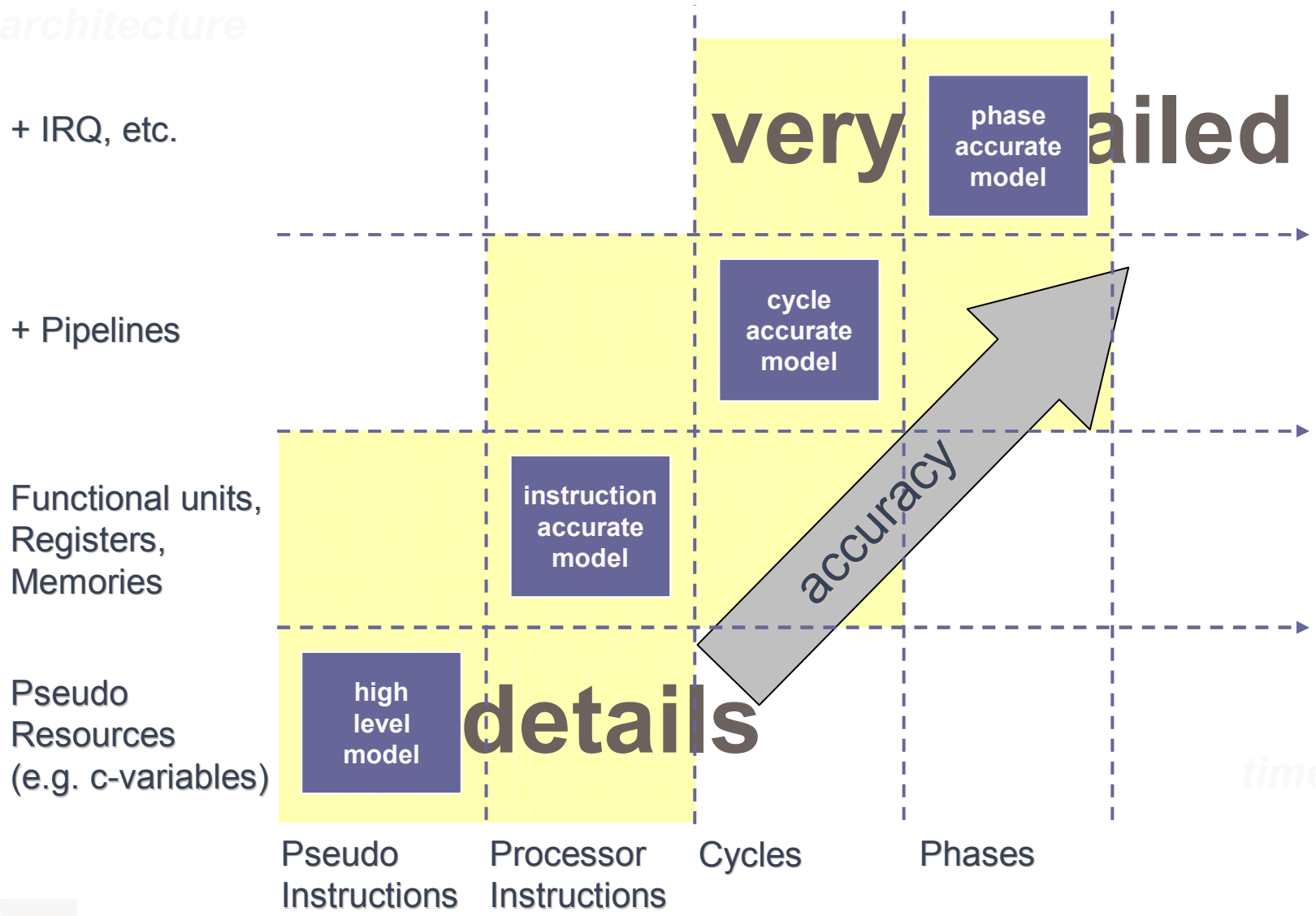
Architecture Description Language based Processor Design

- The purpose of an **architecture description language** (e.g LISA) is:
  - » To allow for an iterative design to efficiently explore architecture alternatives
  - » To jointly design "…hip communication
  - » To automatically g… implementation)
  - » To automatically generate tools
    - » Assembler ,Linker, Compiler, Simulator, co-simulation interfaces

MESCAL 3:
Efficiently describe the ASIP

- From a **single** model at various level of temporal and spatial abstraction

LISA 2.0 - Abstraction Levels



Empty Model
RISC Sample
VLIW Sample
DSP Sample
FFT Processor

LISATek IP

Describe/Adopt Processor Model

Generate Tools

Application

Custom Processor Model (LISA 2.0 language)

LISATek

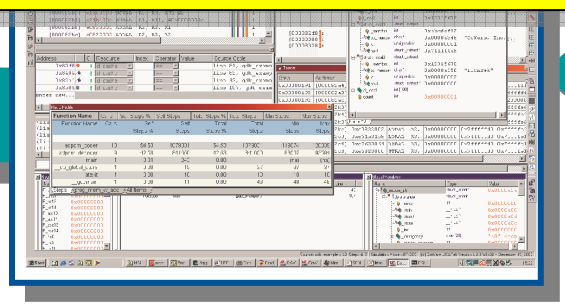
Rapid modeling and re-targetable simulatio

joint optimization of applicatio

MESCAL 3: Efficiently describe and evaluate the ASIP

Generate...

RTL

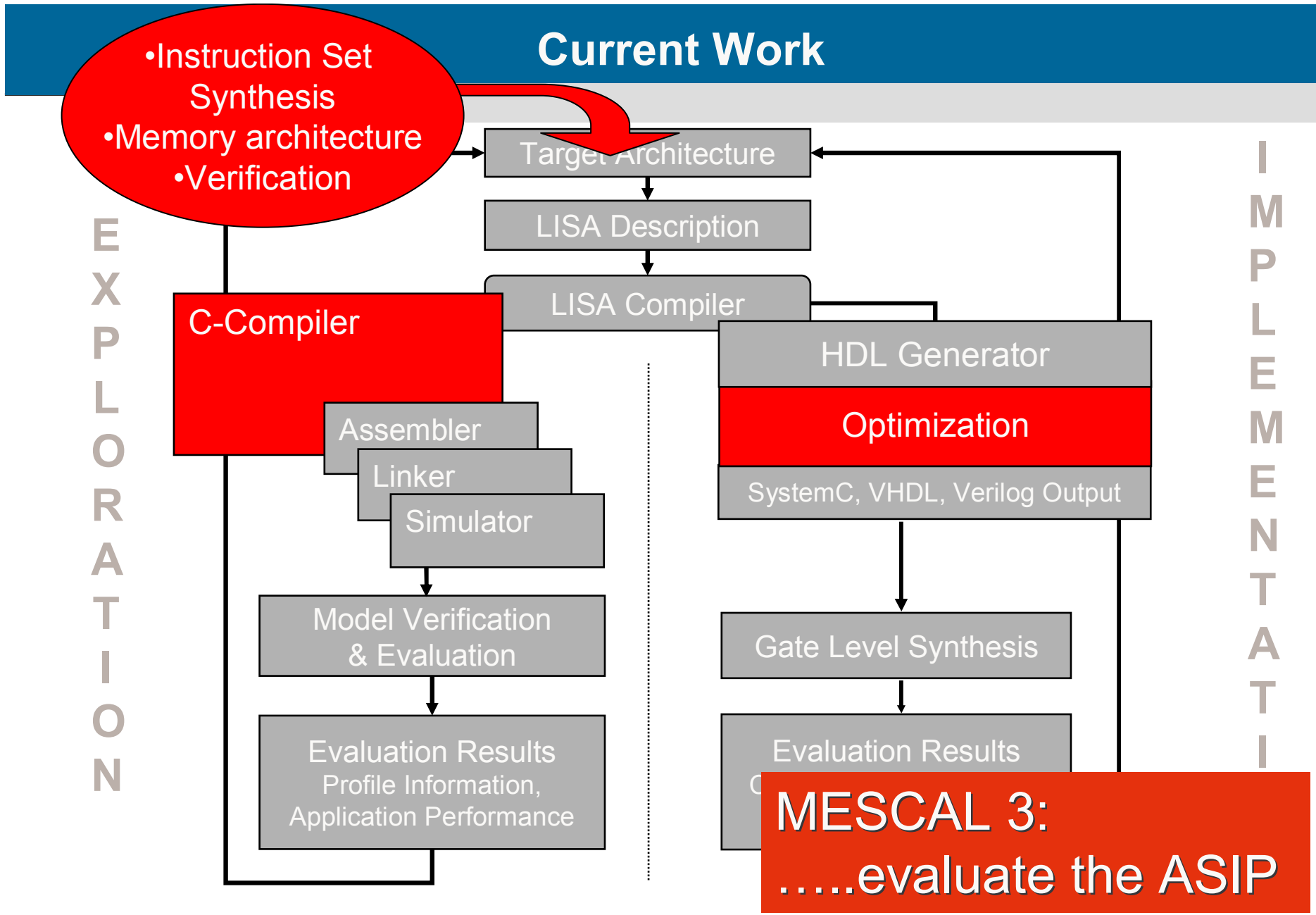


Function and instruction level profiling reveals hot-spots -> special purpose instructions

MESCAL 5: Sucessfully deploy the ASIP

Int
(e.g

Current Work



A Novel Approach for Flexible and Consistent ADL-driven ASIP Design

Gunnar Braun

Achim Nohl

CoWare, Inc

DAC Booth #1844

www.CoWare.com

Weihua Sheng, Jianjiang Ceng, Manuel Hohenauer, Hanno Scharwächter, Rainer Leupers, Heinrich Meyr

Integrated Signal Processing Systems (ISS)
Aachen University of Technology
Germany

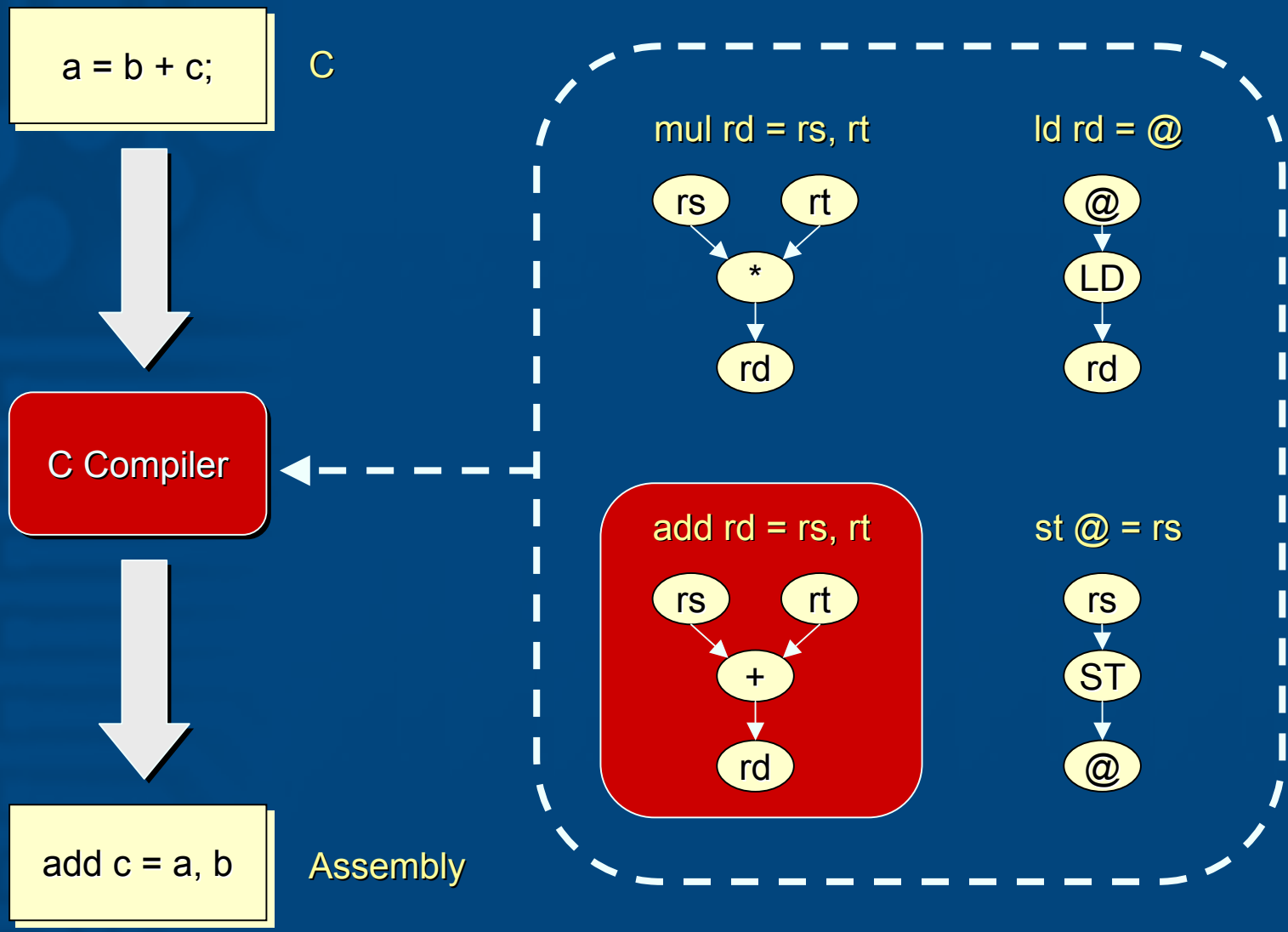
Architecture Description Languages (ADL)

- Automatic generation of Software Toolkit (Compiler, Assembler, Linker, IS-Simulator)
- Architecture Exploration
- SystemC models, RTL code, verification tools, ...

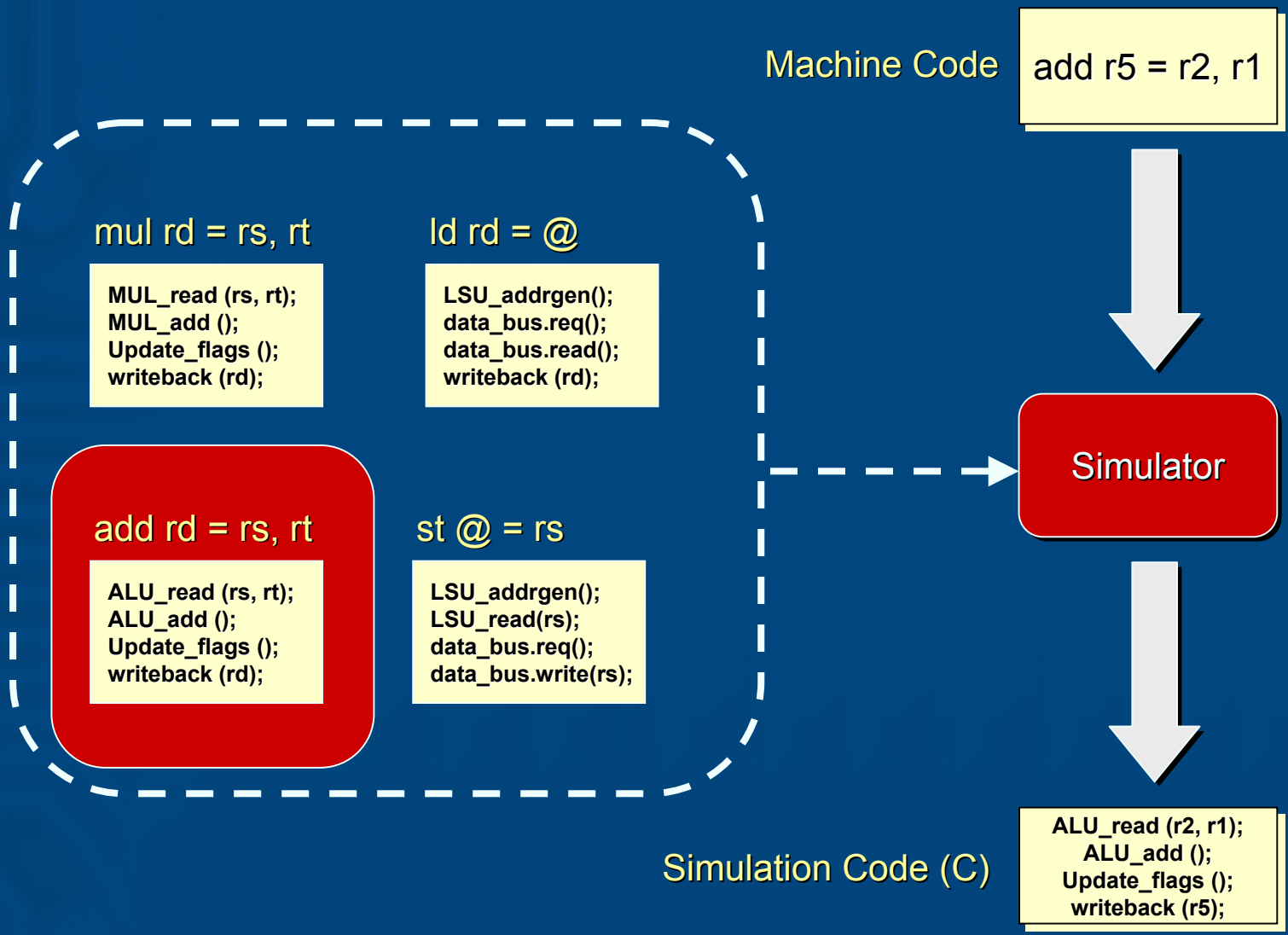
Challenges:

- Different tools need different information
- Unambiguous, redundancy-free architecture model (rather than tools description)
- Multiple abstraction levels (instruction-accurate and/or cycle-accurate)

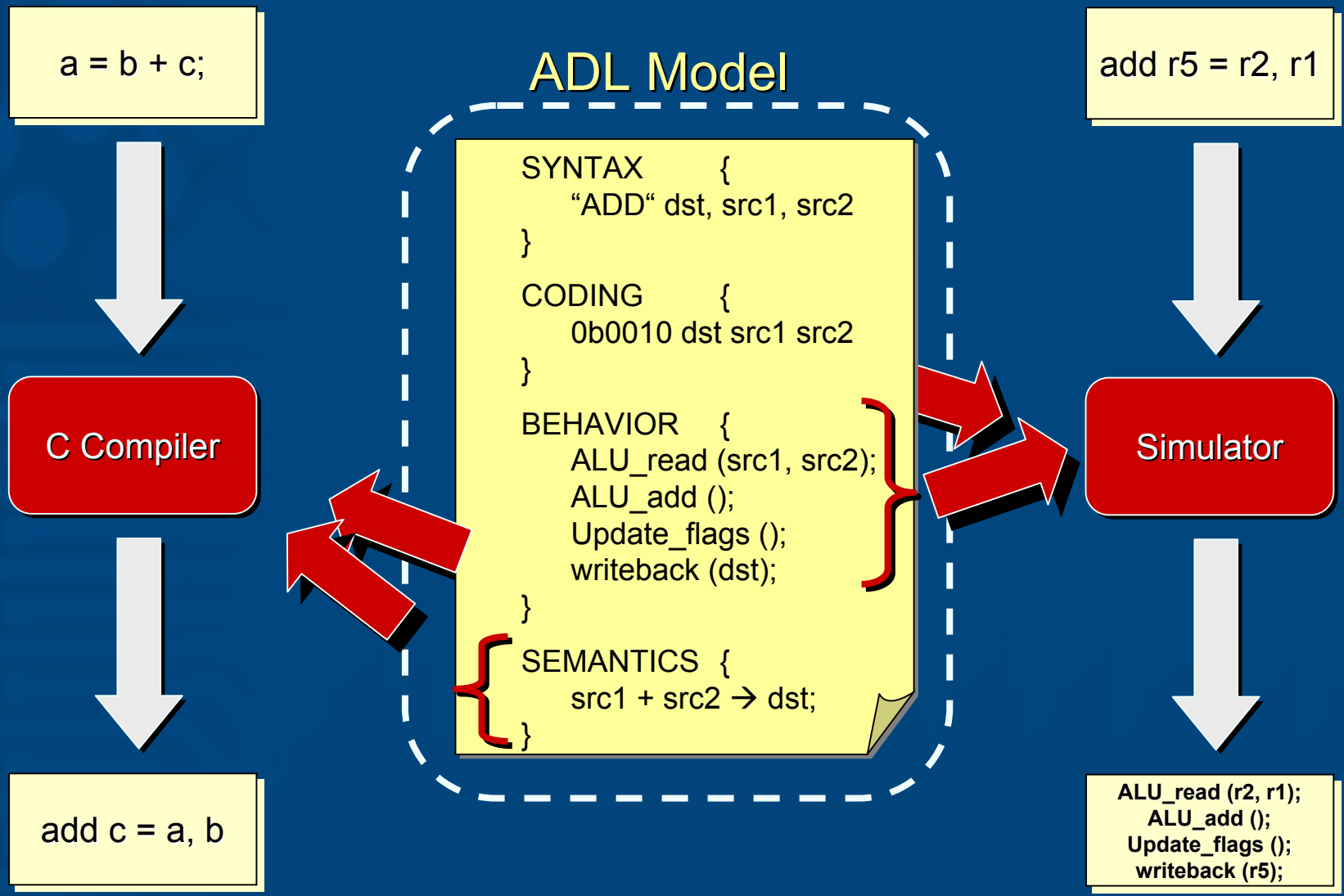
Tool Requirements: Compiler



Tool Requirements: Simulator

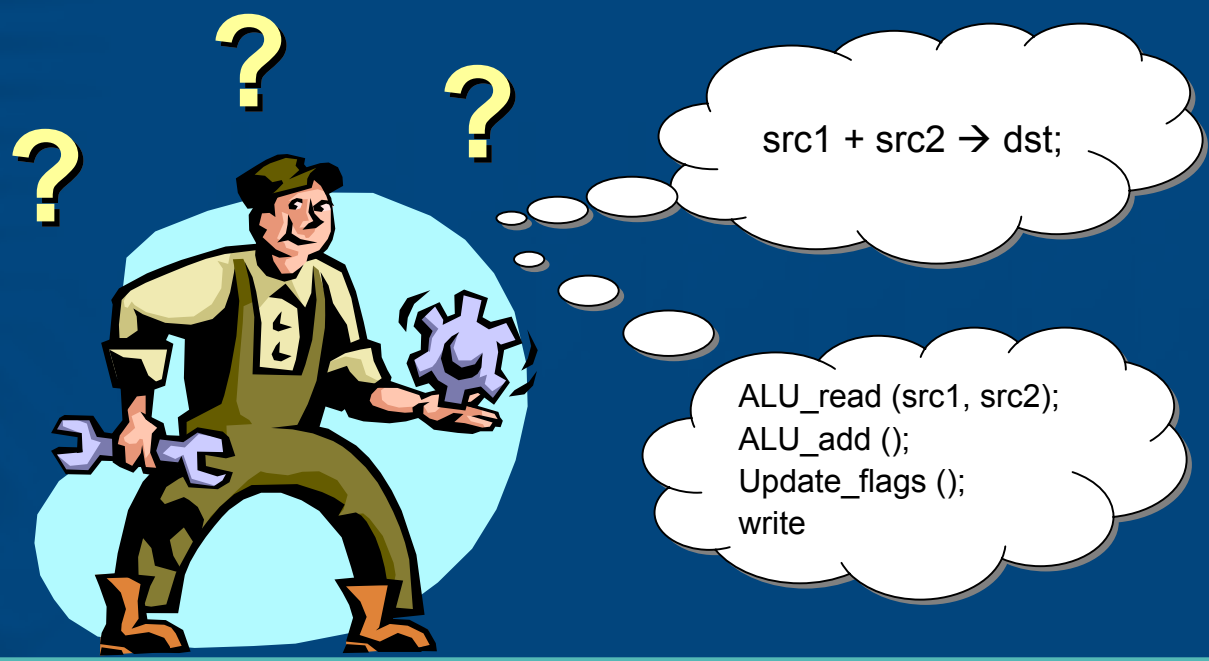


ADL Model


a = b + c;
C Compiler
add c = a, b
ADL Model
SYNTAX {
"ADD" dst, src1, src2
}
CODING {
0b0010 dst src1 src2
}
BEHAVIOR {
ALU_read (src1, src2);
ALU_add ();
Update_flags ();
writeback (dst);
}
SEMANTICS {
src1 + src2 → dst;
}
add r5 = r2, r1
Simulator
ALU_read (r2, r1);
ALU_add ();
Update_flags ();
writeback (r5);

Problem Statement

- **Compiler and Simulator need different information:**
- **Compiler: C operation to instruction(s)**
  - WHAT is the instruction good for? Purpose?
- **Simulator: instructions to sequence of operations**
  - HOW is the instruction executed? What actions to perform?
- **Architecture Designer‘s Perspective:**



Examples

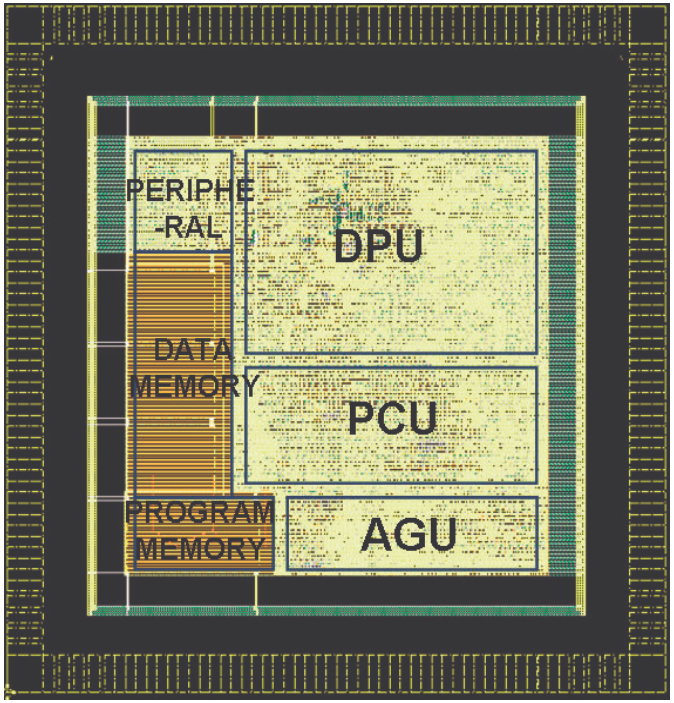
ASDSP FPGA Implementation

Myjung Sunwoo, Ajiou University,

ASDSP Core Design

✓**SEC 0.18um Synthesis**

- Gate : 77,000
- Program Memory : 4 Kbyte, Data Memory : 8 Kbyte
- Frequency : 290MHz
- Power consumption : 0.87W (3mW/MHz)



FPGA Implementation

✓ iProve Xilinx xc2v6000

Support the Special Instruction Set for FFT Operation and the BMU Instruction
Improve the Performance for OFDM Communication

The ICORE

A low-power ASIP for Infineon DVB-T 2nd generation Single-Chip Receiver:

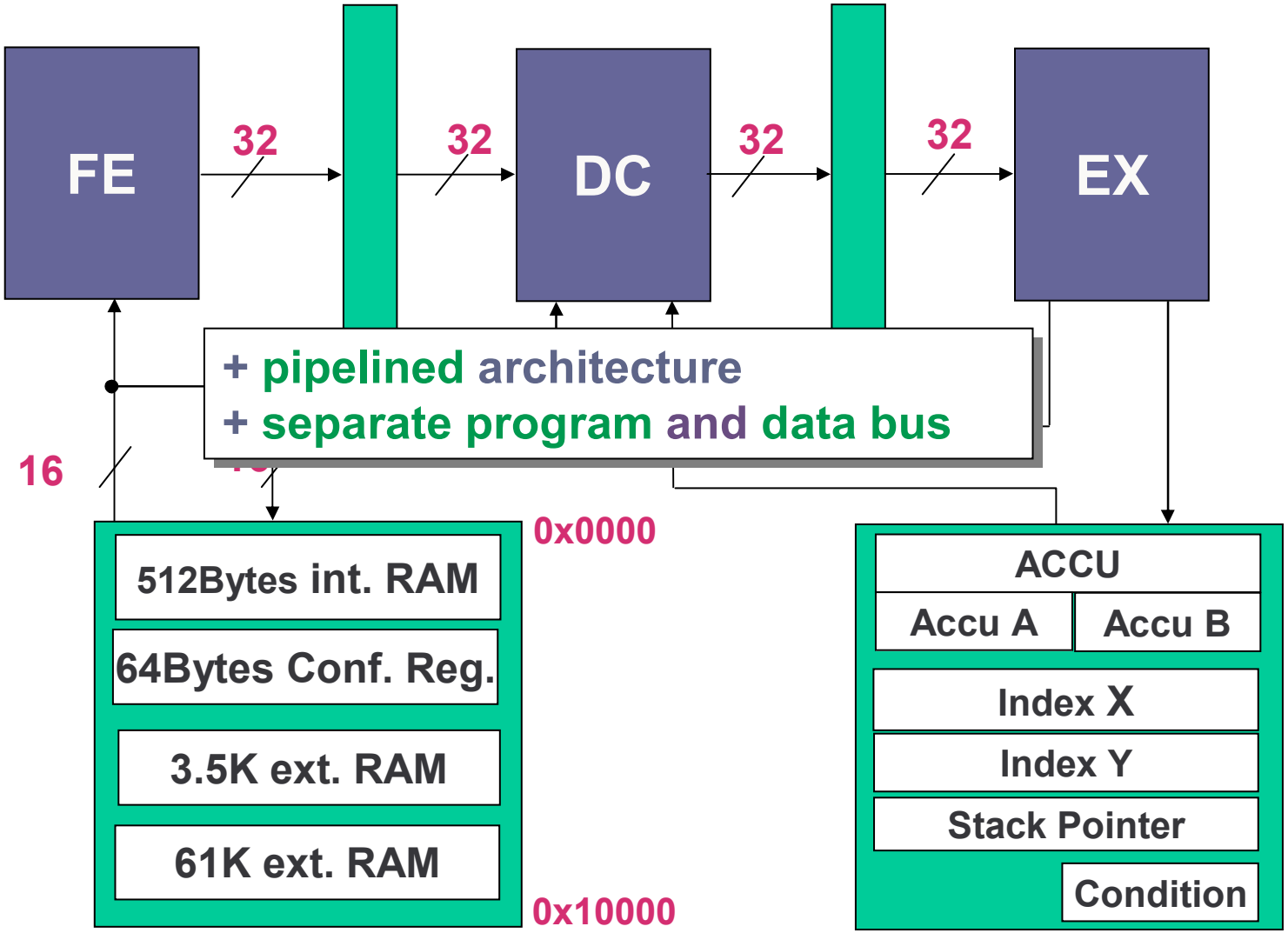
- ASIP for DVB-T acquisition and tracking algorithms (sampling-clock-synchronization, interpolation / decimation, carrier frequency offset estimation)
- Harvard Architecture
- 60 mostly RISC-like Instructions & Special Instructions for CORDIC-Algorithm
- 8x32-Bit General Purpose Registers, 4x9-Bit Address Registers
- 2048x20-Bit Instruction ROM, 512x32-Bit Data Memory
- I2C Registers and dedicated interfaces for external communication

The Motorola M68HC11 Architecture

Architecture Overview

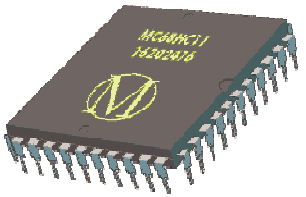
- **M68HC11 CPU Architecture : Hot spots**
  - » 8-bit micro-controller.
    - » Harvard Architecture
  - » 7 CPU Registers.
  - » 6 different Addressing Modes.
  - » Shared data and program bus. : stalled data access
  - » Instruction width : 8,16, 24, 32, 40 : multi-cycle fetch
  - » 8-bit opcode : 181 instructions
  - » Clock speed : ~200 MHz
  - » Performance : : non-pipelined
  - » Area : 15K to 30K (DesignWare® Library)

Architecture Development with LISA

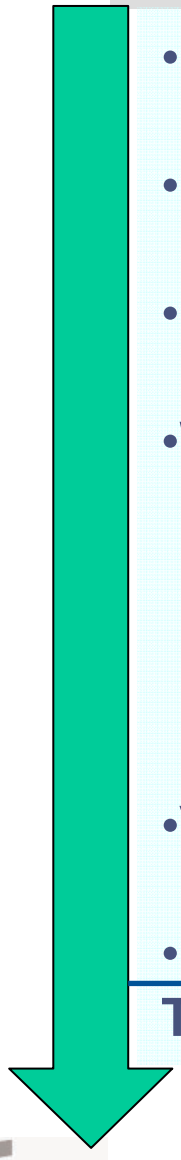


Results

- Area
  - < 23k gates
- Clock speed
  - ~ 200 MHz
- Execution time speed up
  - 62 % for spanning tree application
- Mapped onto Xilinx FPGA



Architecture Development with LISA



•Studying the architecture	4 days
•Basic architecture modifications	2 days
•Grouping and coding of the instructions	1 day
•Writing the LISA model	
-basic syntax and coding	4 days
-behavior section	6 days
•Validation	4 days
•HDL Generation	2 days
Total	23 days

Design of Application Specific Processor Architectures

Rainer Leupers
RWTH Aachen University
Software for Systems on Silicon
leupers@iss.rwth-aachen.de

Overview

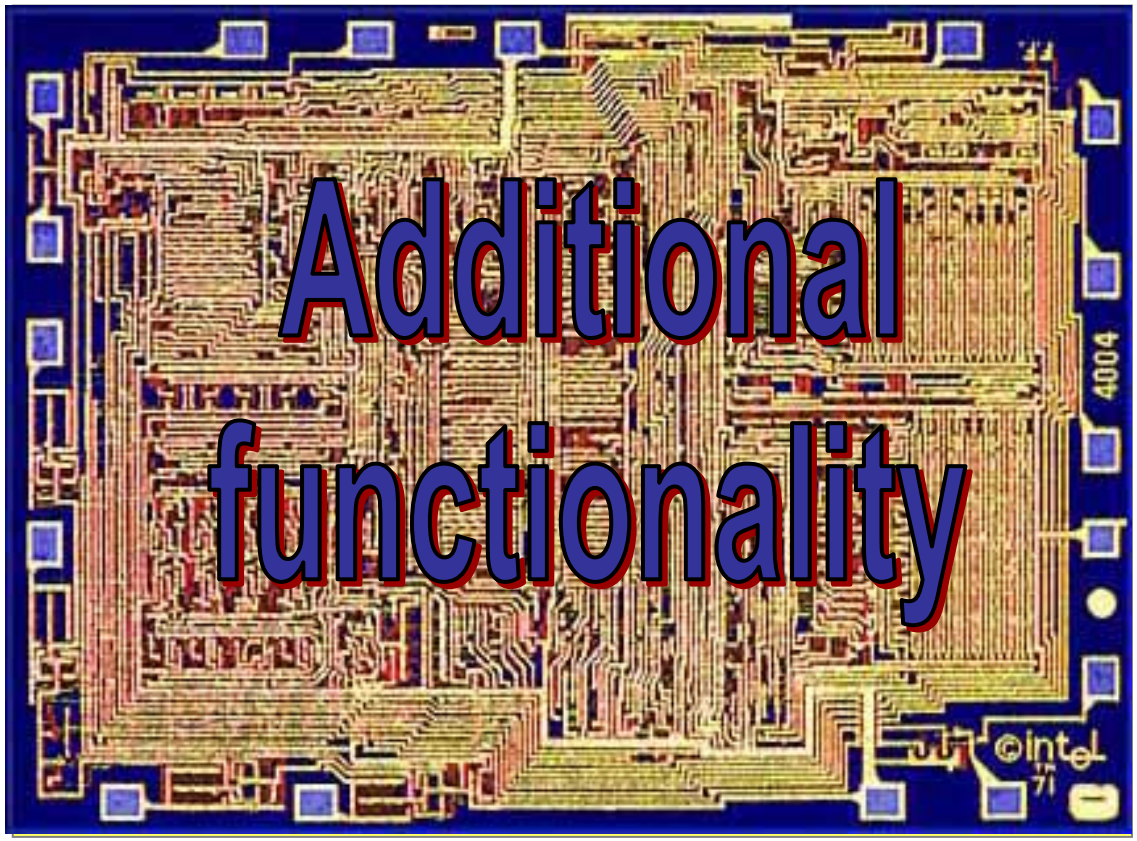
1. Introduction
2. ASIP design methodologies
3. Software tools
4. ASIP architecture design
5. Case study
6. Advanced research topics

1. Introduction

Embedded system design automation

- Embedded systems
  - Special-purpose electronic devices
  - Very different from desktop computers
- Strength of European IT market
  - Telecom, consumer, automotive, medical, ...
  - Siemens, Nokia, Bosch, Infineon, ...
- New design requirements
  - Low NRE cost, high efficiency requirements
  - Real-time operation, dependability
  - Keep pace with Moore´s Law

What to do with chip area ?



Example: wireless multimedia terminals

- Multistandard radio
  - UMTS
  - GSM/GPRS/EDGE
  - WLAN
  - Bluetooth
  - UWB
  - …
- Multimedia standards
  - MPEG-4
  - MP3
  - AAC
  - GPS
  - DVB-H
  - …

Key issues:

- Time to market (≤ 12 months)
- Flexibility (ongoing standard updates)
- Efficiency (battery operation)

Application specific processors (ASIPs)

„As the performance of conventional microprocessors improves, they first meet and then exceed the requirements of most computing applications. Initially, performance is key. But eventually, other factors, like customization, become more important to the customer...“

[M.J. Bass, C.M. Christensen: The Future of the Microprocessor Business, IEEE Spectrum 2002]

design budget = (semiconductor revenue) × (% for R&D)

growth ≈ 15% ≈ 10%

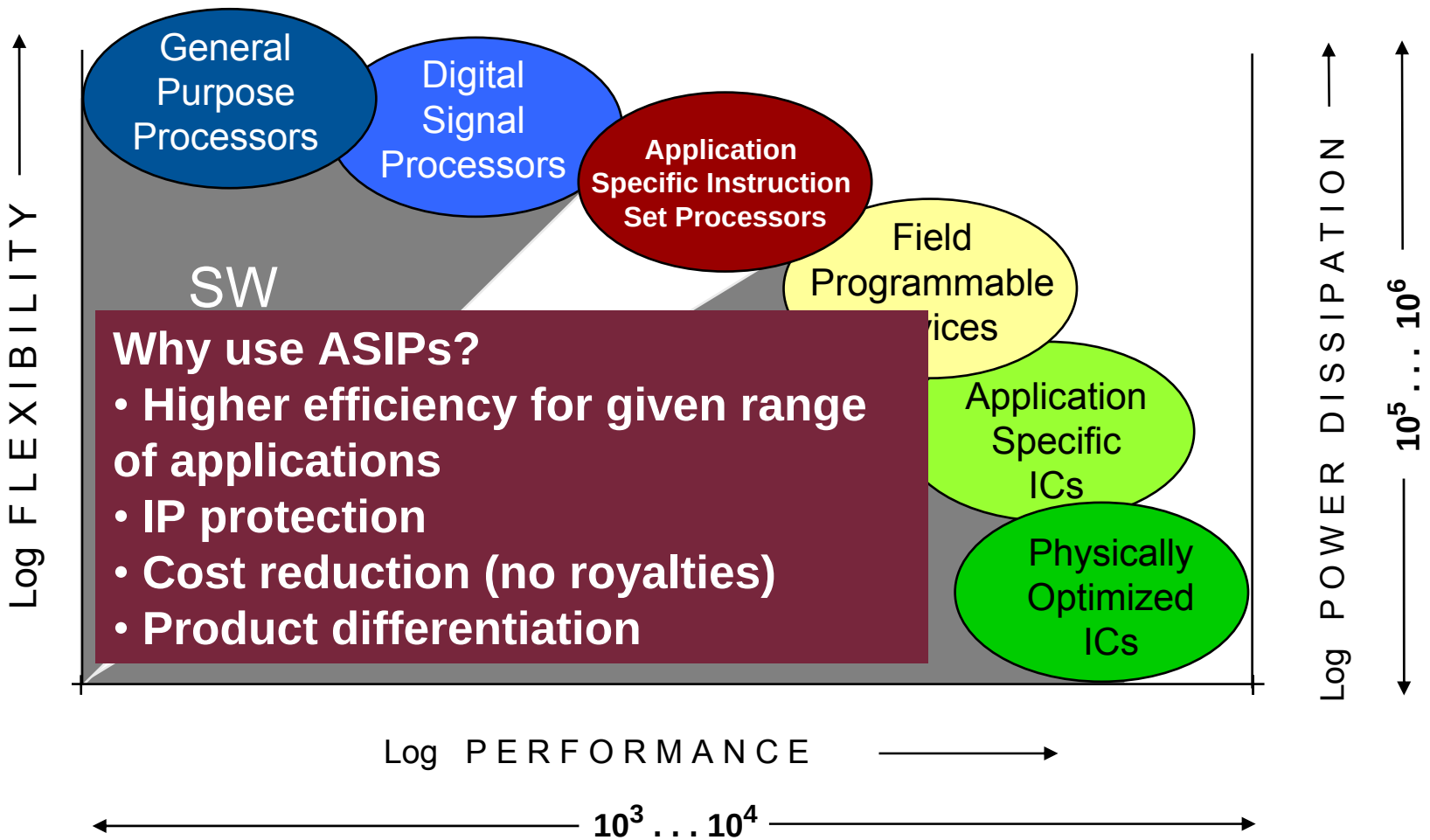
IC designs = (design budget) / (design cost per IC)

growth ≈ 15% growth ≈ 50-100%

[Keutzer05]

→ ***Customizable application specific processors as reusable, programmable platforms***

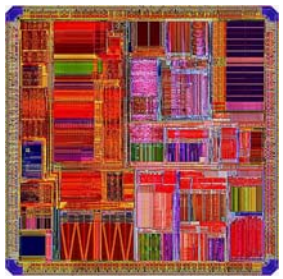
Efficiency and flexibility



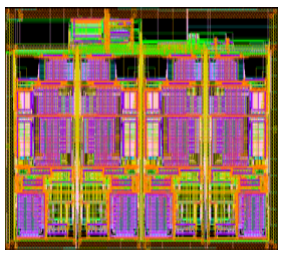
Source: T.Noll, RWTH Aachen

2. ASIP design methodologies

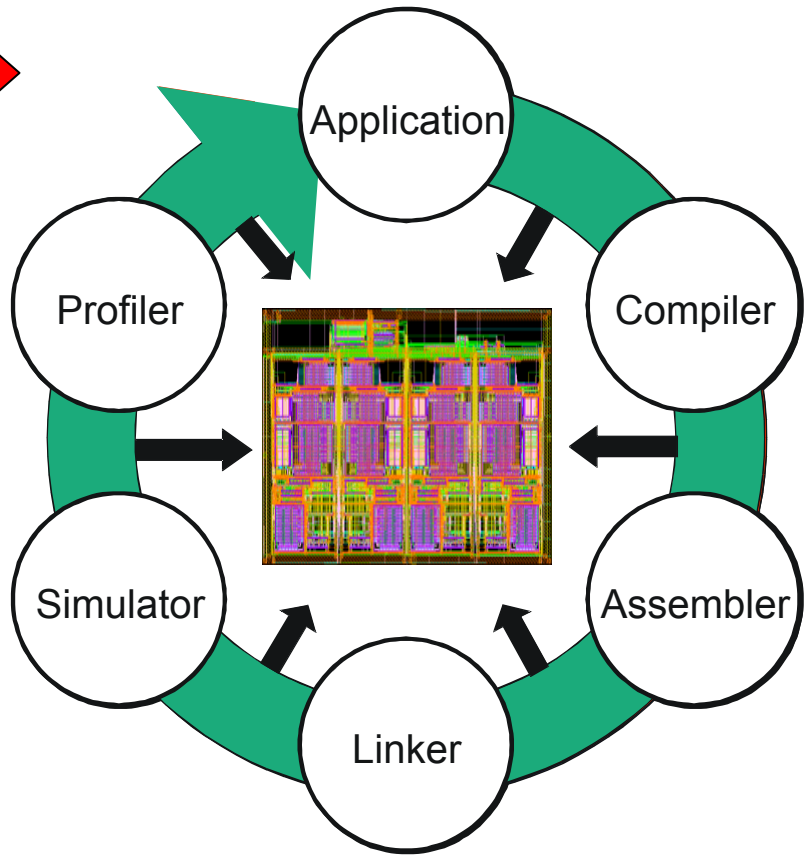
ASIP architecture exploration



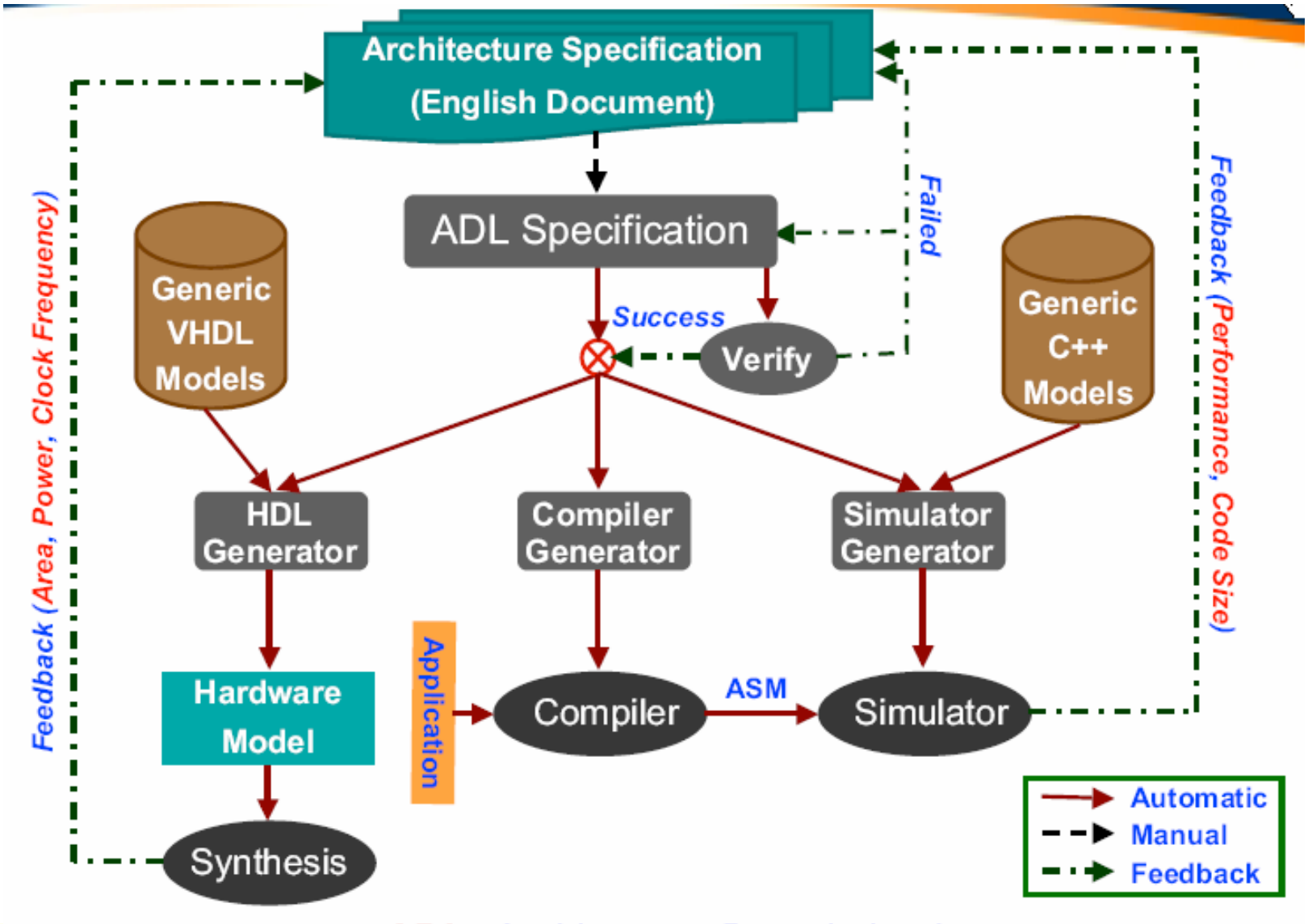
initial processor architecture



optimized processor architecture

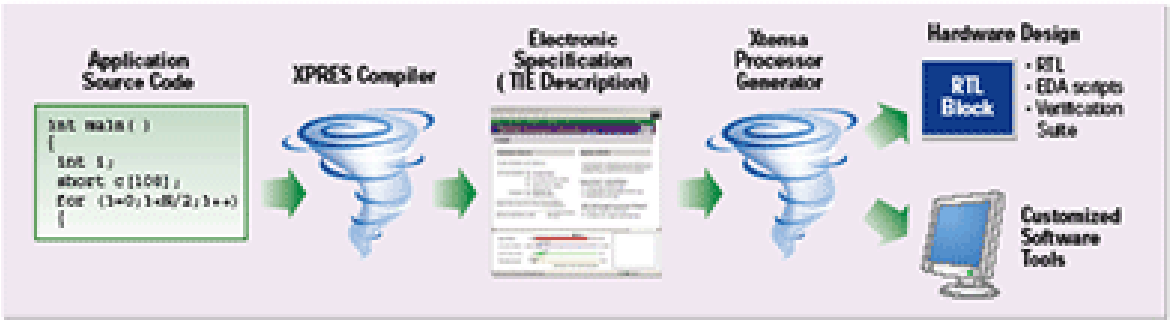


Expression (UC Irvine)



Tensilica Xtensa/XPRES

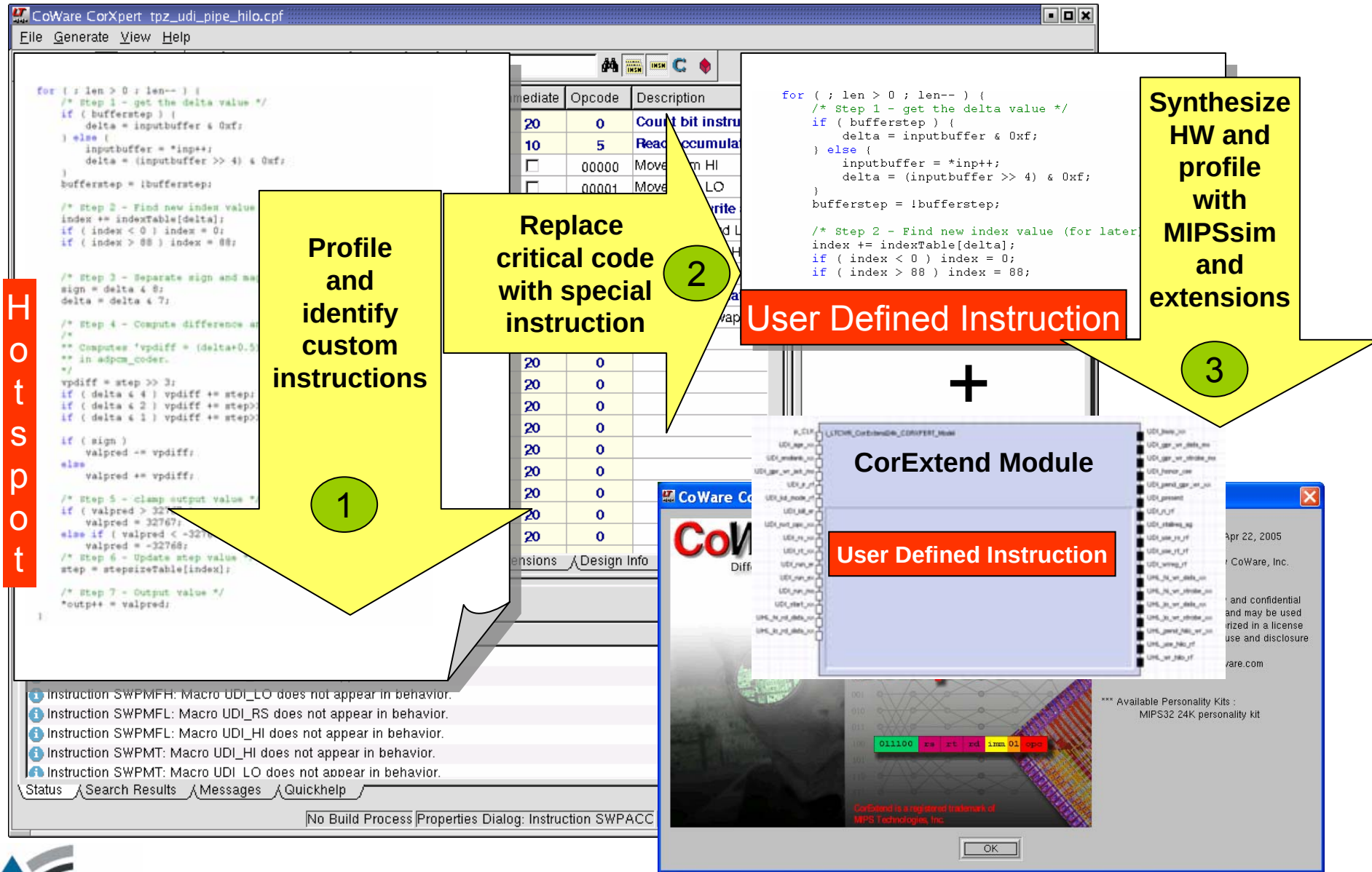
From an ANSI C / C++ application the XPRES Compiler generates an optimized set of processor extensions …



… that is reusable over a range of similar application software code.

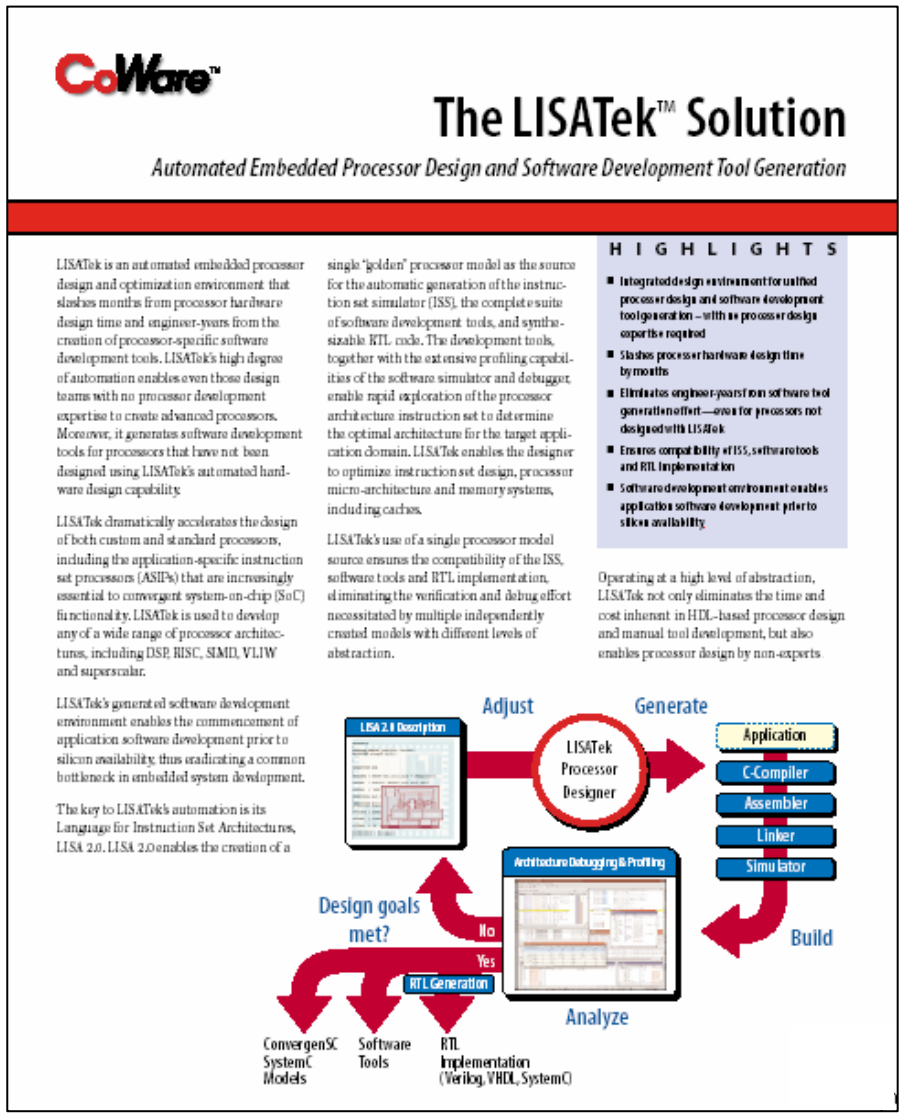
Source: Tensilica Inc.

MIPS CorXtend/CoWare CorXpert



CoWare LISATek ASIP architecture exploration

- Integrated embedded processor development environment
- Unified processor model in LISA 2.0 architecture description language (ADL)
- Automatic generation of:
  - SW tools
  - HW models



CoWare™

The LISATek™ Solution

Automated Embedded Processor Design and Software Development Tool Generation

LISATek is an automated embedded processor design and optimization environment that slashes months from processor hardware design time and engineer-years from the creation of processor-specific software development tools. LISATek's high degree of automation enables even those design teams with no processor development expertise to create advanced processors. Moreover, it generates software development tools for processors that have not been designed using LISATek's automated hardware design capability.

LISATek dramatically accelerates the design of both custom and standard processors, including the application-specific instruction set processors (ASIPs) that are increasingly essential to convergent system-on-chip (SoC) functionality. LISATek is used to develop any of a wide range of processor architectures, including DSP, RISC, SIMD, VLIW and superscalar.

LISATek's generated software development environment enables the commencement of application software development prior to silicon availability, thus eradicating a common bottleneck in embedded system development.

The key to LISATek's automation is its Language for Instruction Set Architectures, LISA 2.0. LISA 2.0 enables the creation of a single "golden" processor model as the source for the automatic generation of the instruction set simulator (ISS), the complete suite of software development tools, and synthesizable RTL code. The development tools, together with the extensive profiling capabilities of the software simulator and debugger, enable rapid exploration of the processor architecture instruction set to determine the optimal architecture for the target application domain. LISATek enables the designer to optimize instruction set design, processor micro-architecture and memory systems, including caches.

LISATek's use of a single processor model source ensures the compatibility of the ISS, software tools and RTL implementation, eliminating the verification and debug effort necessitated by multiple independently created models with different levels of abstraction.

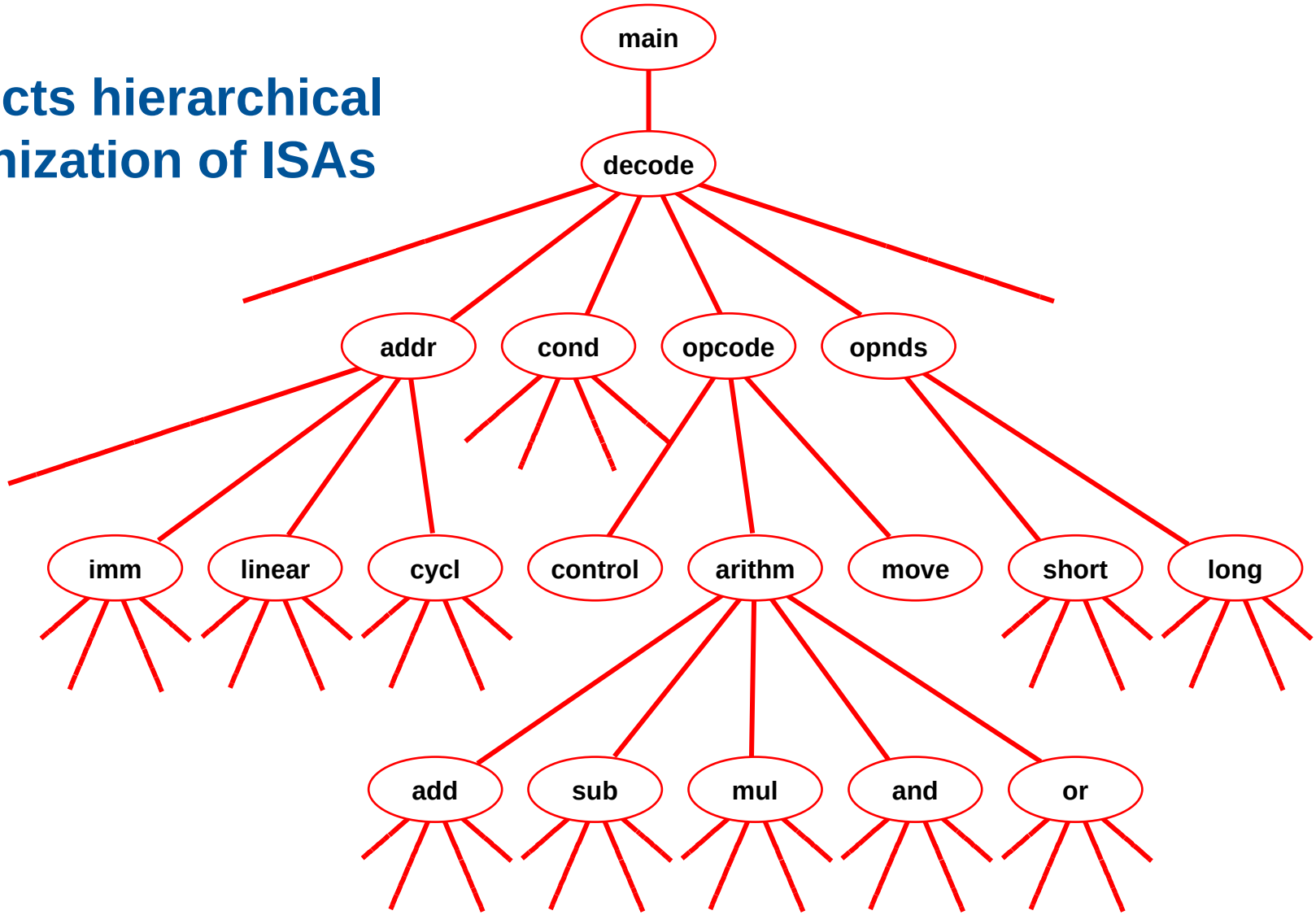
HIGHLIGHTS

- Integrated design environment for unified processor design and software development tool generation – with no processor design expertise required
- Slashes processor hardware design time by months
- Eliminates engineer-years from software tool generation effort—even for processors not designed with LISATek
- Ensures compatibility of ISS, software tools and RTL implementation
- Software development environment enables application software development prior to silicon availability

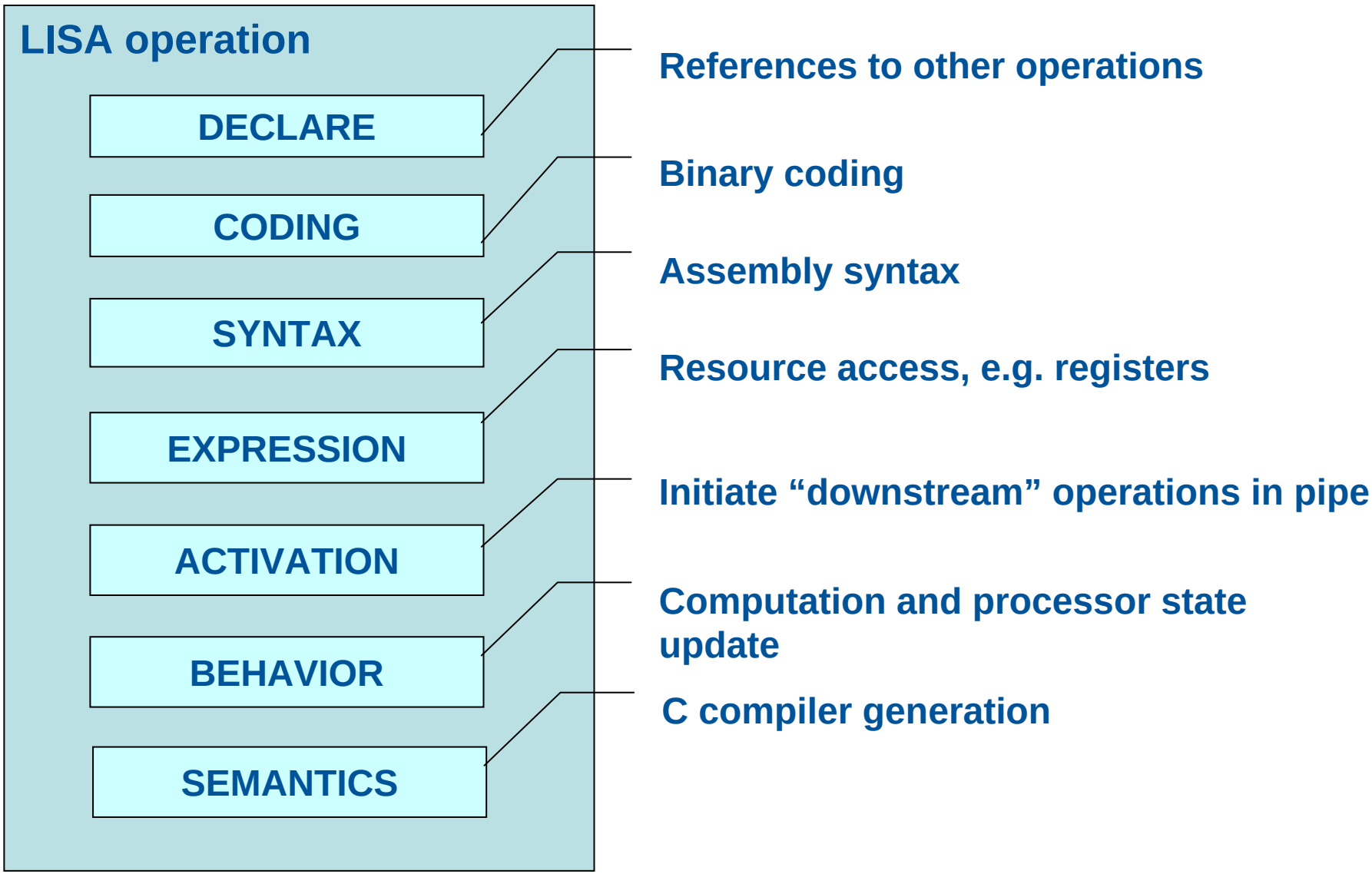
Operating at a high level of abstraction, LISATek not only eliminates the time and cost inherent in HDL-based processor design and manual tool development, but also enables processor design by non-experts.

LISA operation hierarchy

Reflects hierarchical organization of ISAs



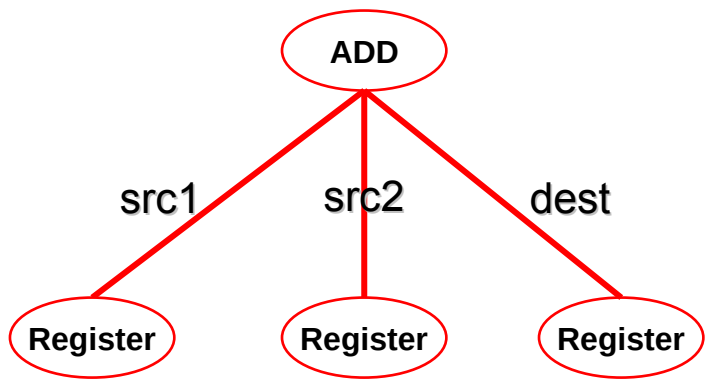
LISA operations structure



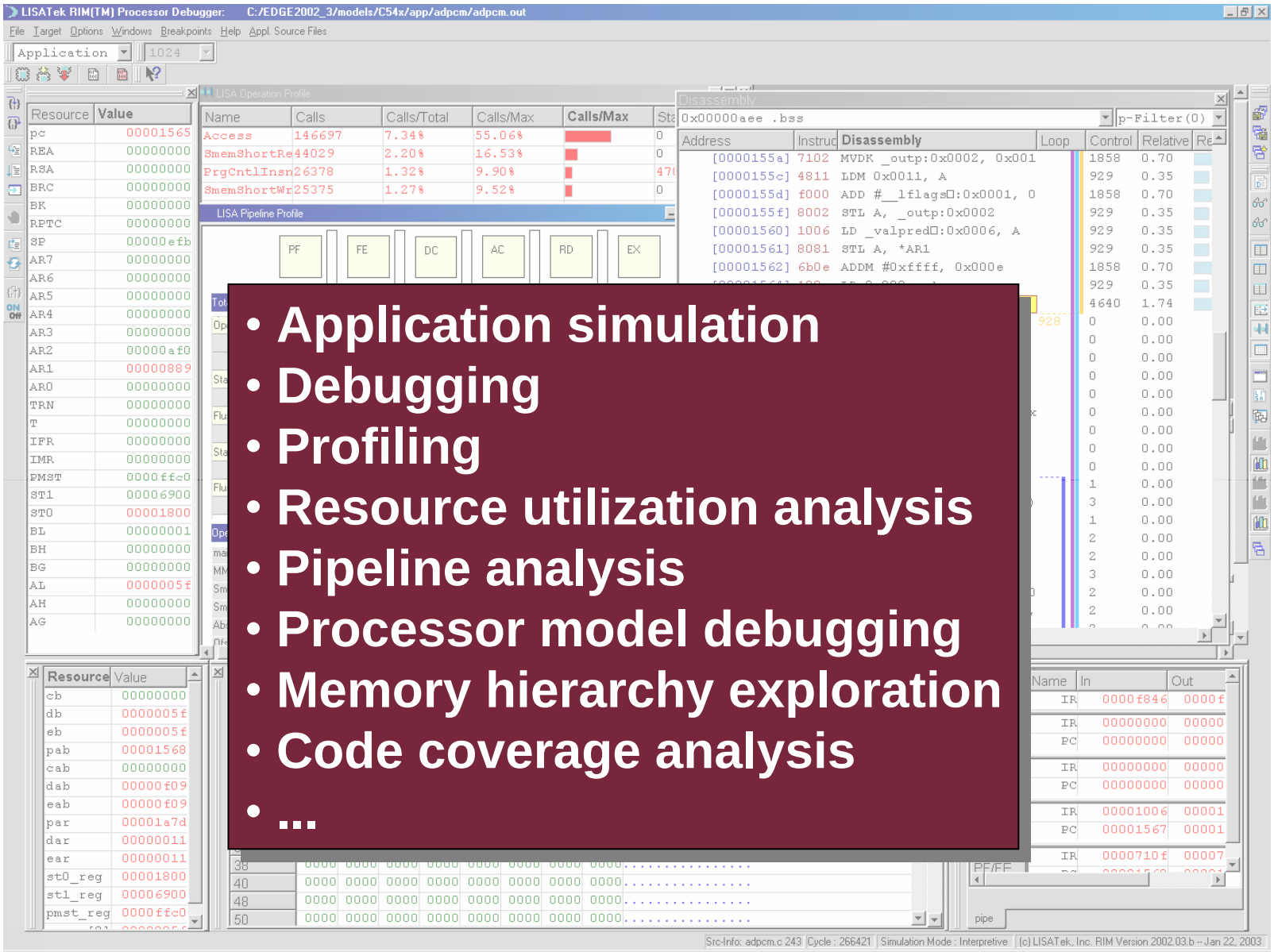
LISA operation example

```
OPERATION ADD
    {
        DECLARE
        {
                GROUP src1, src2, dest = { Register }
        }
        CODING { 0b1011 src1 src2 dest }

        SYNTAX { “ADD” dest “,” src1 “,” src2 }

        BEHAVIOR { dest = src1 + src2; }
    }
OPERATION Register
    {
        DECLARE
        {
                LABEL  index;
        }

        CODING { index }

        SYNTAX { “R”  index }
        EXPRESSION{ R[index] }
    }
```

C/C++ Code



Exploration/debugger GUI

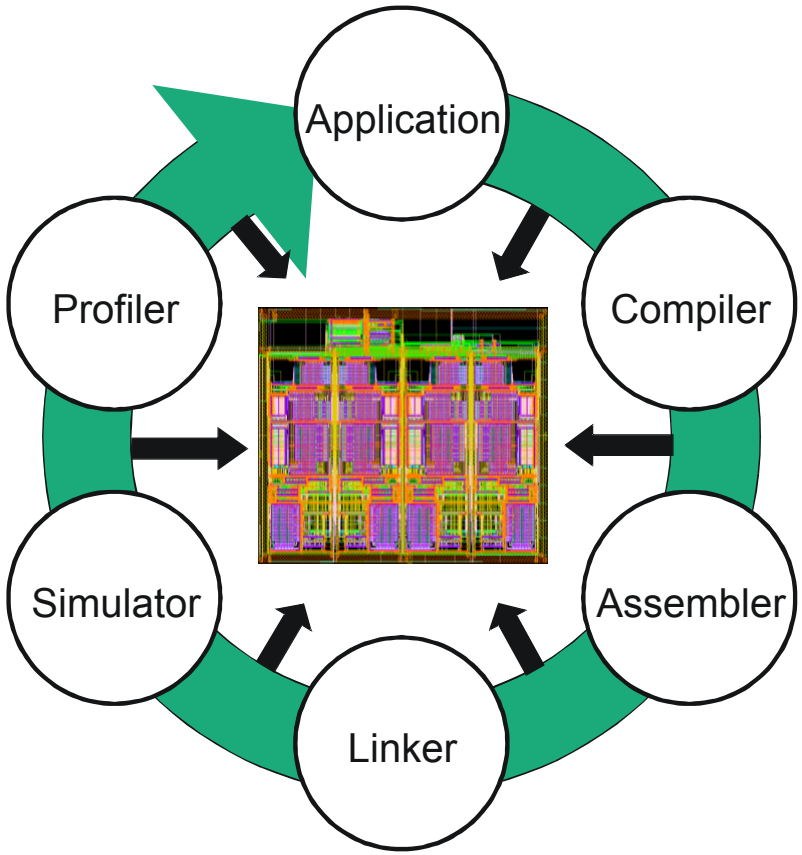


Some available LISA 2.0 models

- DSP:
  - Texas Instruments TMS320C54x
  - Analog Devices ADSP21xx
  - Motorola 56000
- RISC:
  - MIPS32 4K
  - ESA LEON SPARC 8
  - ARM7100
  - ARM926
- VLIW:
  - Texas Instruments TMS320C6x
  - STMicroelectronics ST220
- µC:
  - MHS80C51
- ASIP:
  - Infineon PP32 NPU
  - Infineon ICore
  - MorphICs DSP

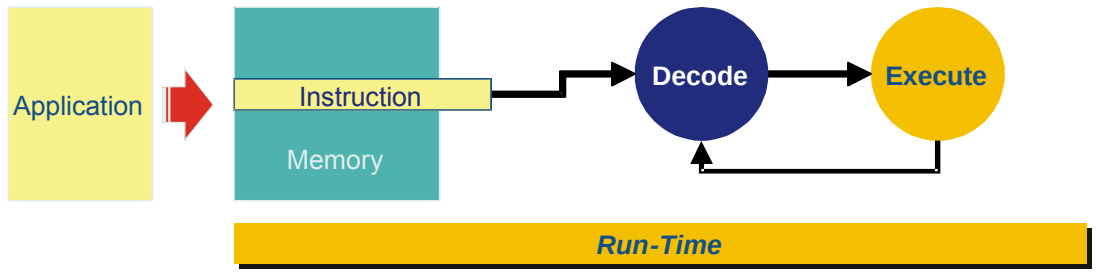
3. Software tools

Tools generated from processor ADL model



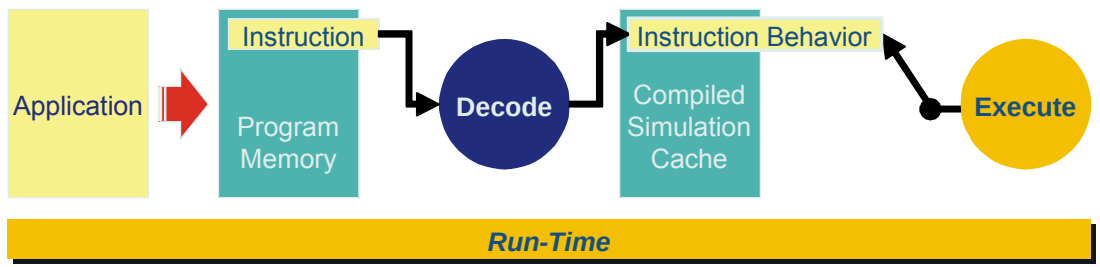
Instruction set simulation

Interpretive:
- flexible
- slow (~ 100 KIPS)



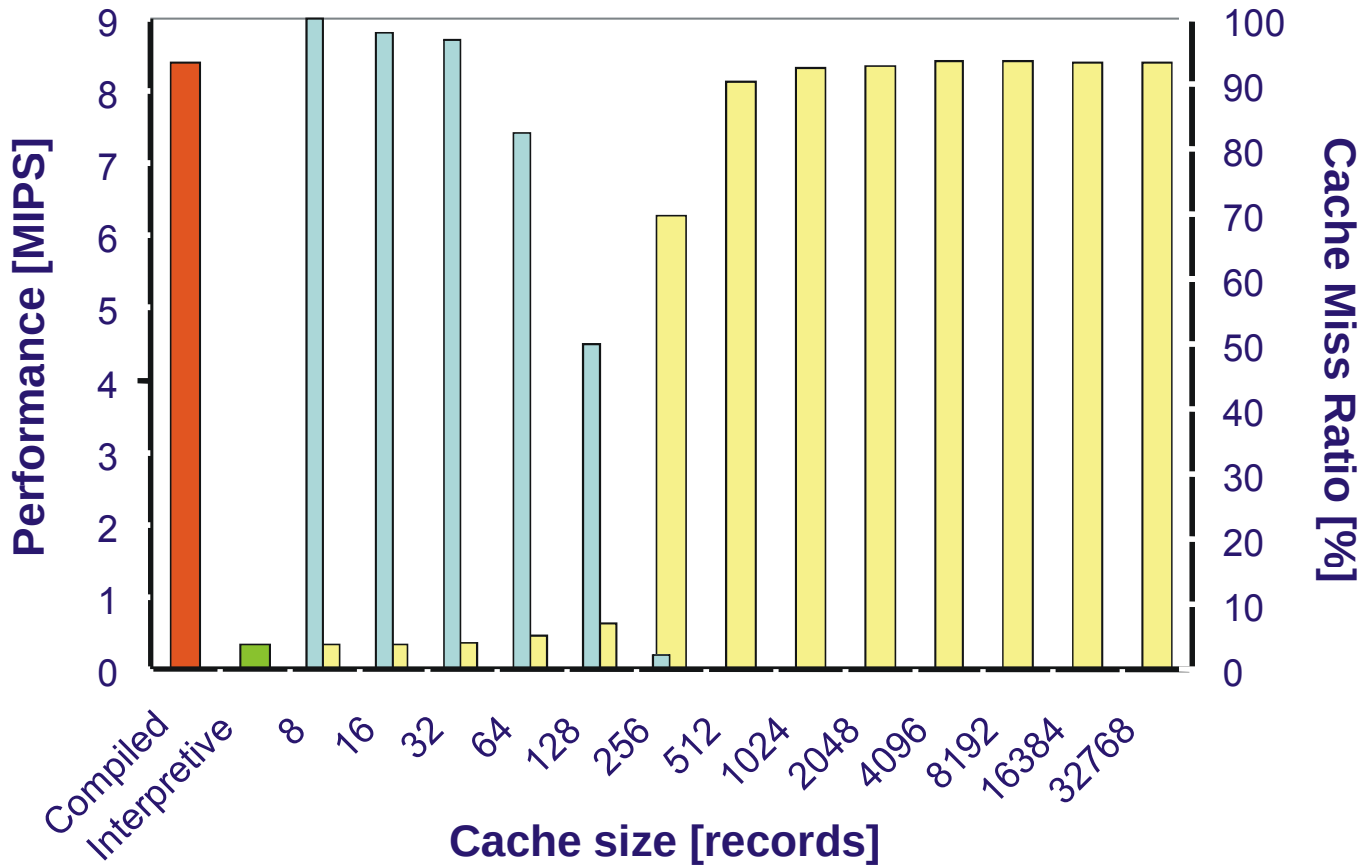
Compiled:
- fast (> 10 MIPS)
- inflexible
- high memory consumption

JIT-CCS™:
- „just-in-time“ compiled
- SW simulation cache
- fast and flexible



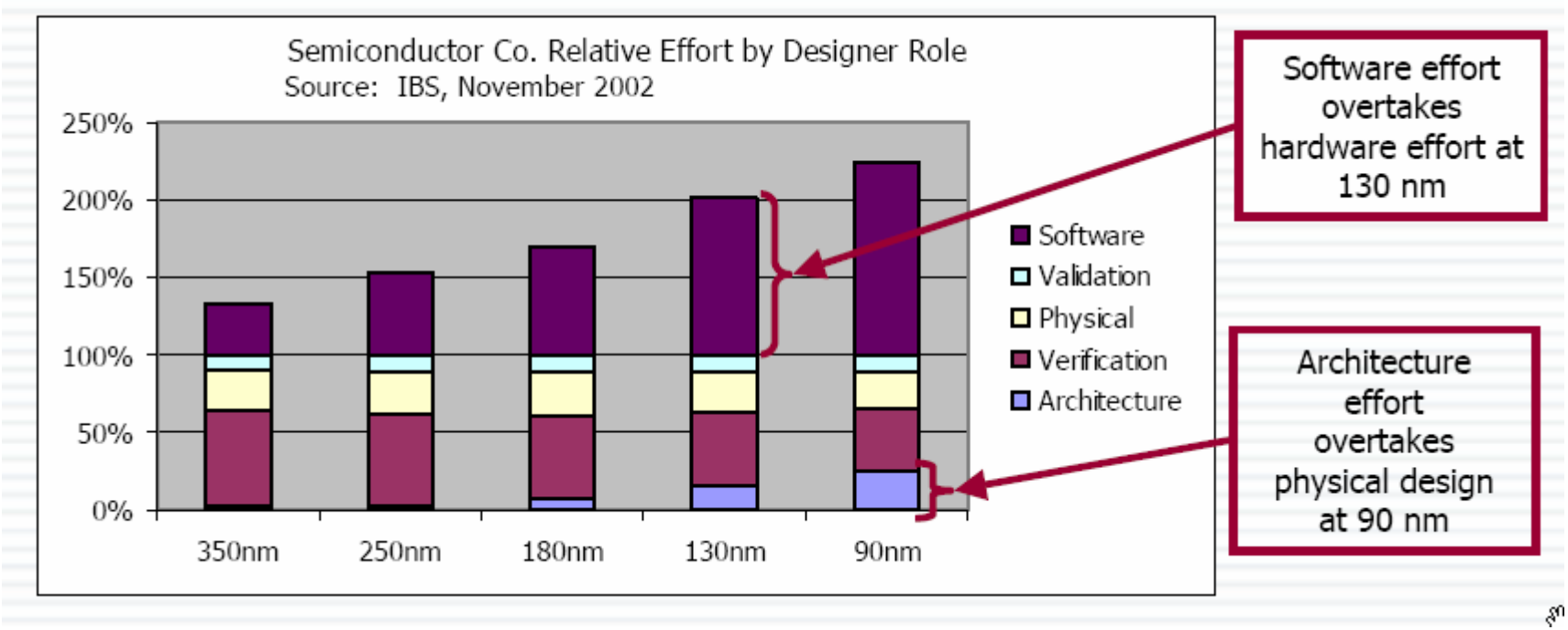
JIT-CC simulation performance

- Dependent on simulation cache size
- 95% of compiled simulation performance @ 4096 cache blocks (10% memory consumption of compiled sim.)
- Example: ST200 VLIW DSP



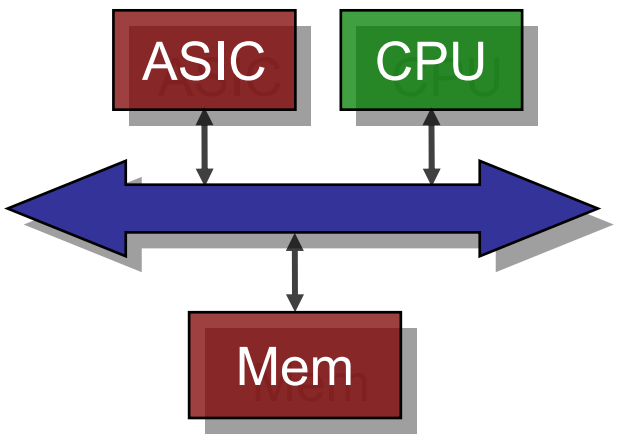
Why care about C compilers?

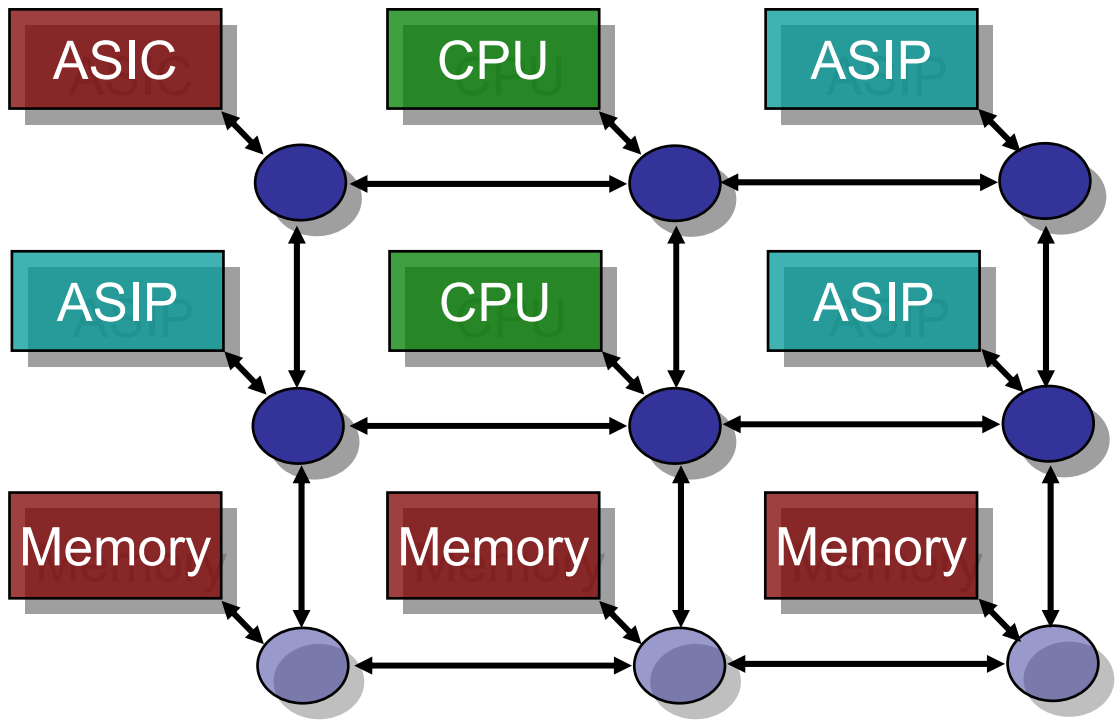
- Embedded SW design becoming predominant manpower factor in system design
- Cannot develop/maintain millions of code lines in assembly language
- Move to high-level programming languages



Why care about compilers?

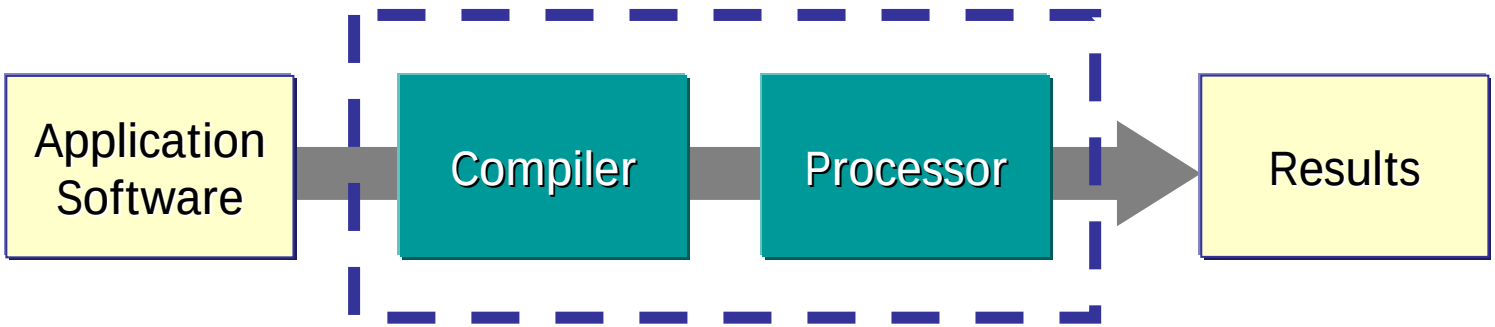
- Trend towards heterogeneous multiprocessor systems-on-chip (MPSoC)
- Customized application specific instruction set processors (ASIPs) are key MPSoC components
- How to achieve efficient compiler support for ASIPs?





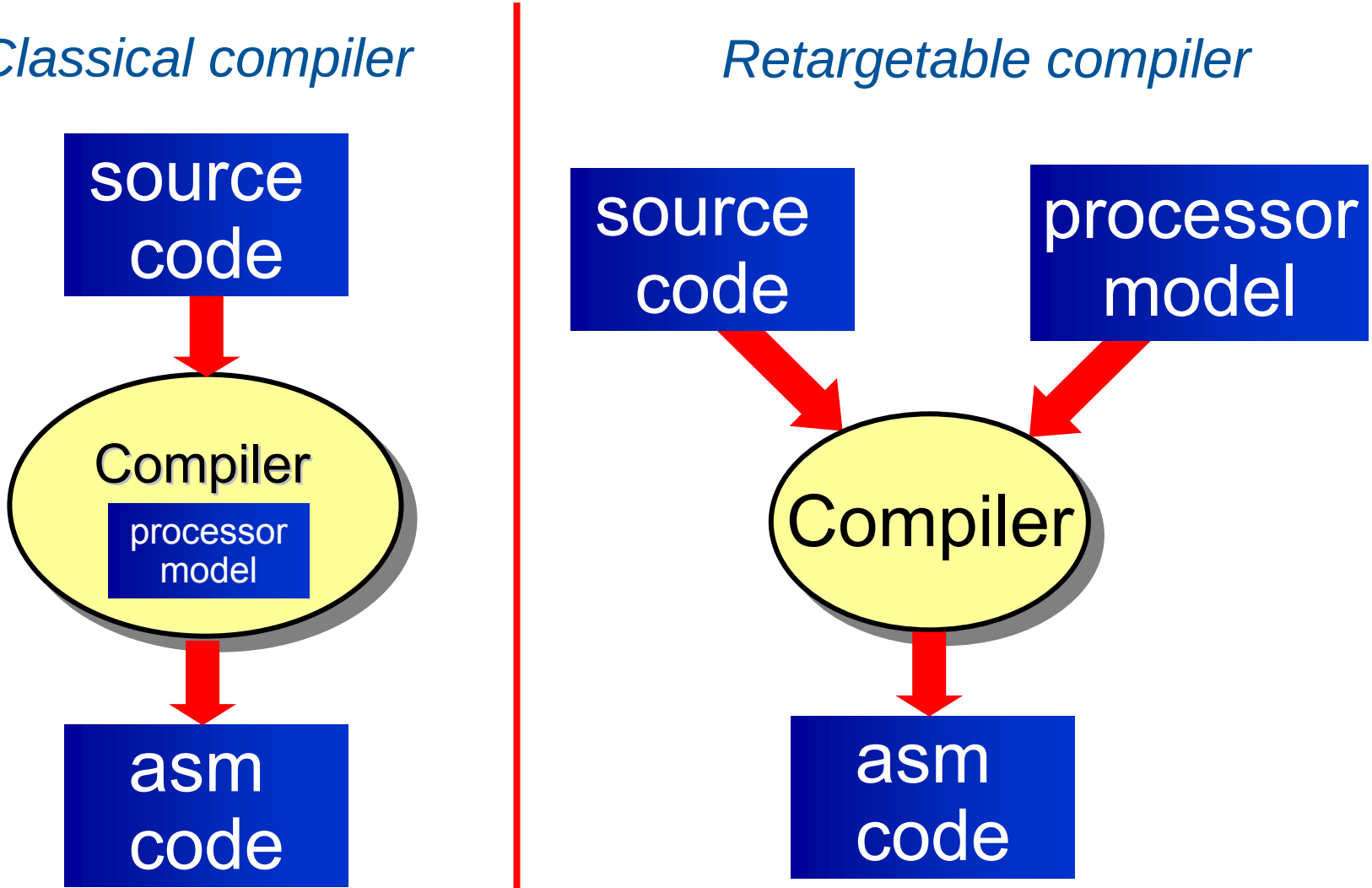
„Compiler/Architecture Co-Design“

- Efficient C-compilers cannot be designed for ARBITRARY architectures!



- Compiler and processor form a UNIT that needs to be optimized!
- “Compiler-friendliness“ needs to be taken into account during the architecture exploration!

Retargetable compilers



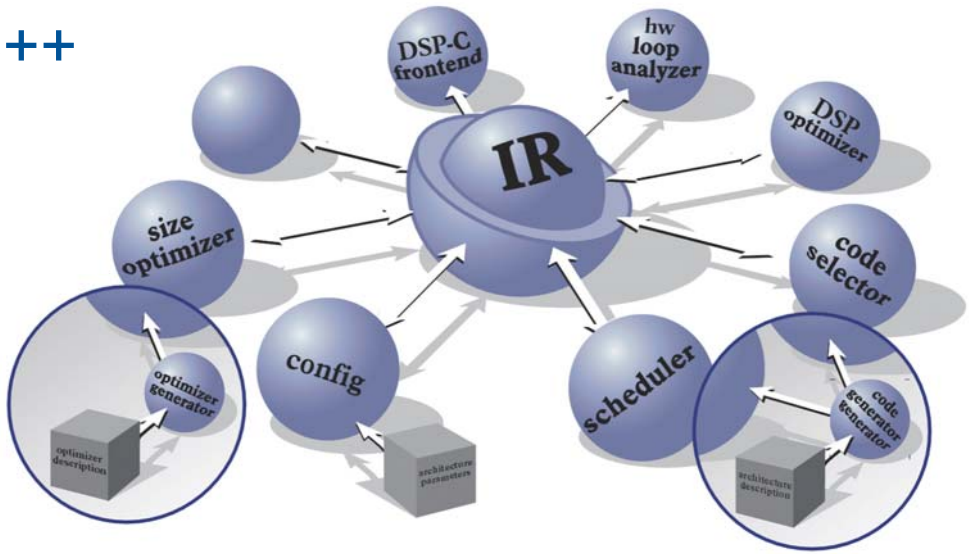
GNU C compiler (gcc)

- Probably the most widespread retargetable compiler
- Mostly used as a native Unix/Linux compiler, but may operate as a cross-compiler, too
- Support for C/C++, Java, and other languages
- Comes with comprehensive support software, e.g. runtime and standard libraries, debug support
- Portable to new architectures by means of machine description file and C support routines

> *"The main goal of GCC was to make a good, fast compiler for machines in the class that the GNU system aims to run on: 32-bit machines that address 8-bit bytes and have several general registers. Elegance, theoretical power and simplicity are only secondary."*

CoSy compiler system (ACE)

- Universal retargetable C/C++ compiler
- Extensible intermediate representation (IR)
- Modular compiler organization
- Generator (BEG) for code selector, register allocator, scheduler



LISATek C compiler generation

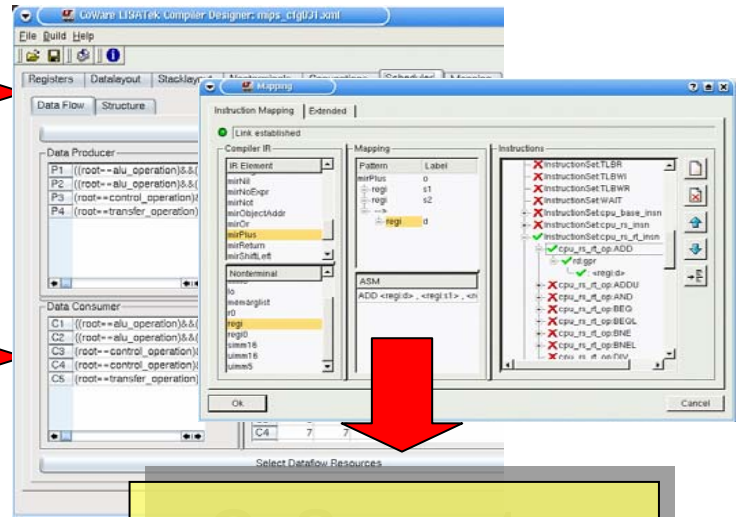
LISA processor model

```
SYNTAX      {
      "ADD" dst, src1, src2
}
CODING      {
      0b0010 dst src1 src2
}
BEHAVIOR    {
      ALU_read (src1, src2);
      ALU_add ();
      Update_flags ();
      writeback (dst);
}
SEMANTICS   {
      src1 + src2 → dst;
}
...
```

Autom. analyses

Manual refinement

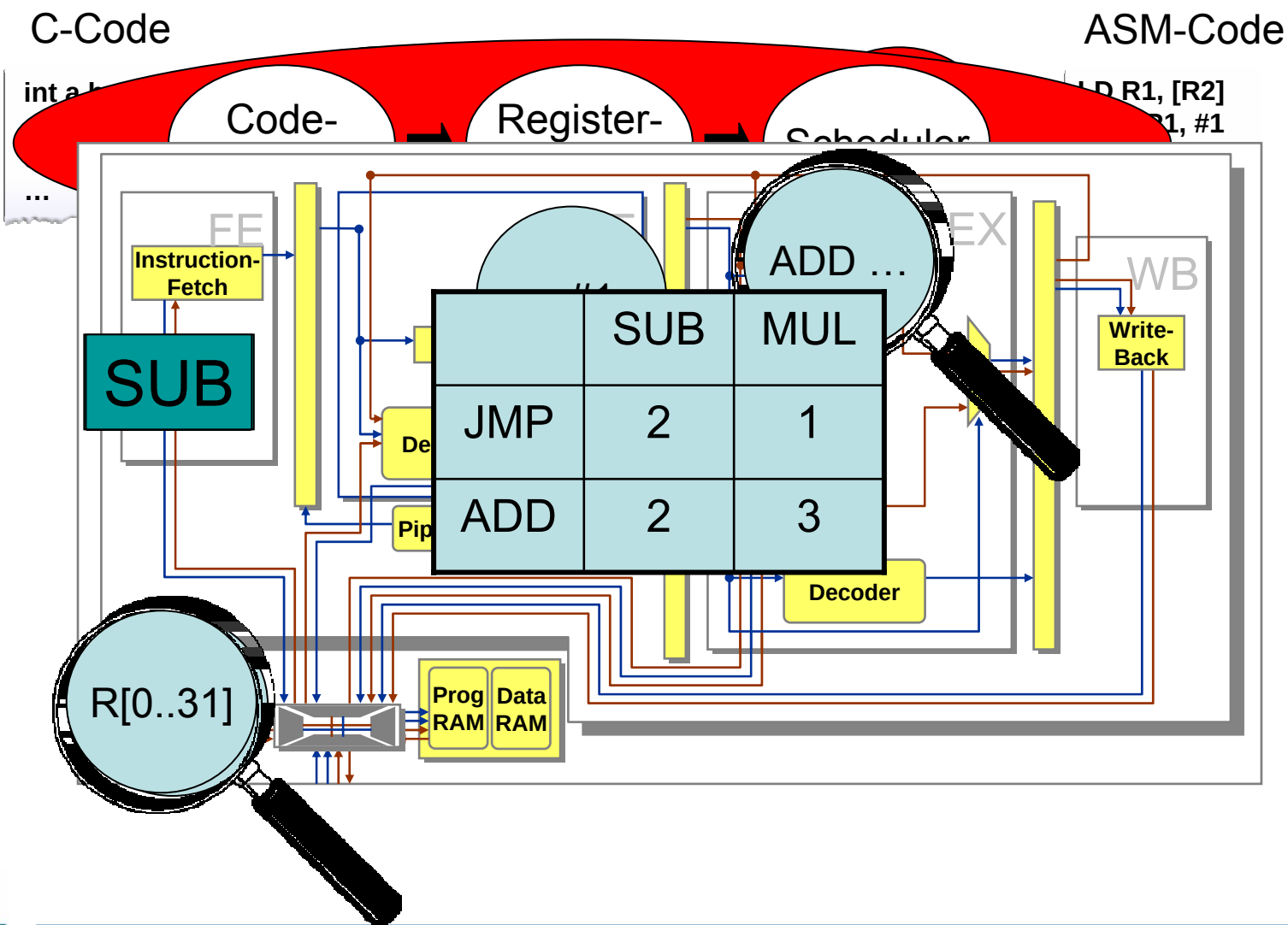
GUI



CoSy system

C Compiler

LISATek compiler generation

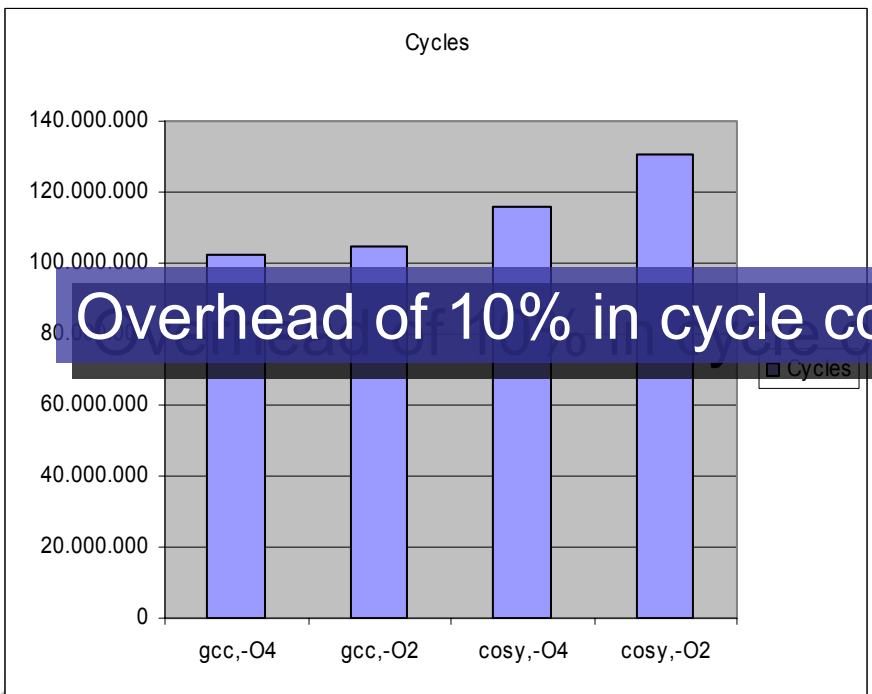


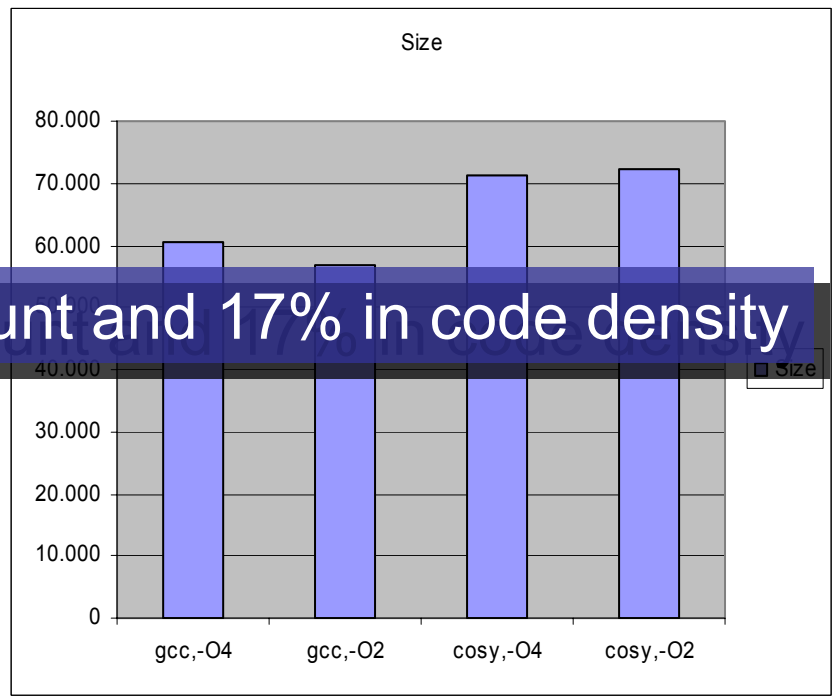
Compiled code quality: MIPS example

- LISATek generated C-Compiler
- Out-of-the-box C-Compiler
- No manual optimizations
- Development time of model approx. 2 weeks

gcc C-Compiler

- gcc with MIPS32 4kc backend
- Used by most MIPS users
- Large group of developers, several man-years of optimization



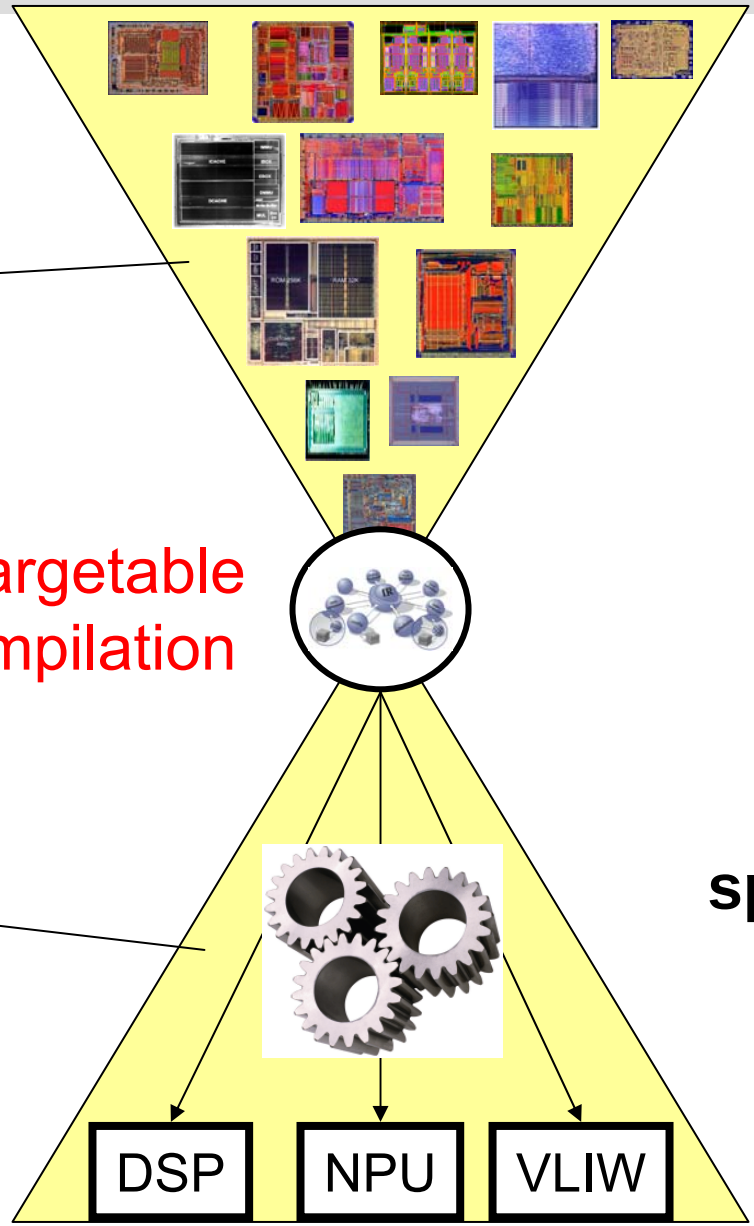


Overhead of 10% in cycle count and 17% in code density

Demands on code quality

- Compilers for embedded processors have to generate extremely efficient code
  - Code size:
    - » system-on-chip
    - » on-chip RAM/ROM
  - Performance:
    - » real-time constraints
  - Power/energy consumption:
    - » heat dissipation
    - » battery lifetime

Compiler flexibility/code quality trade-off

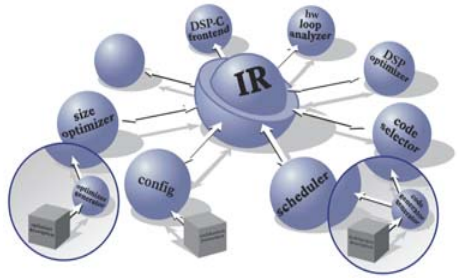


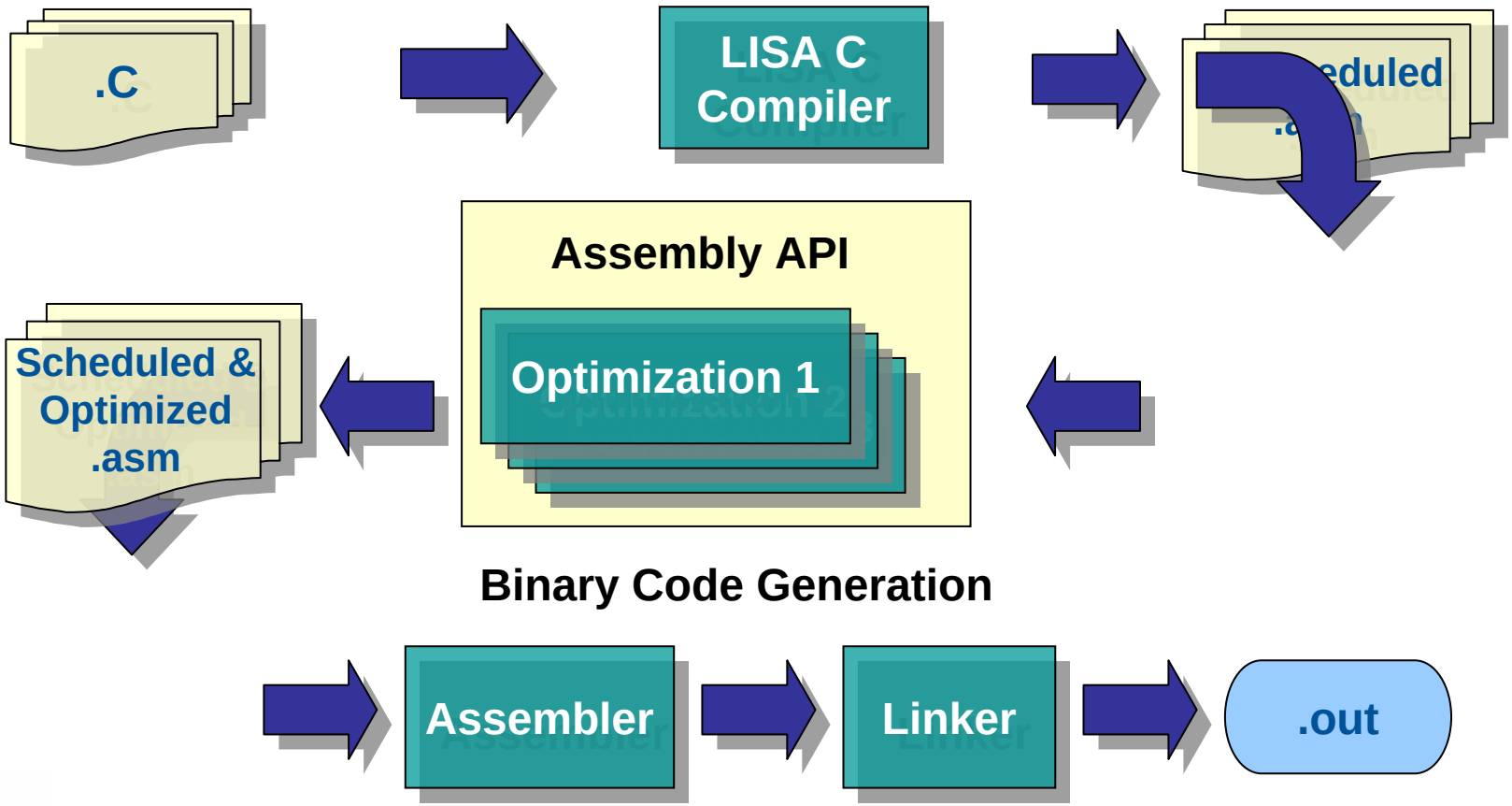
unification

specialization

Adding processor-specific code optimizations

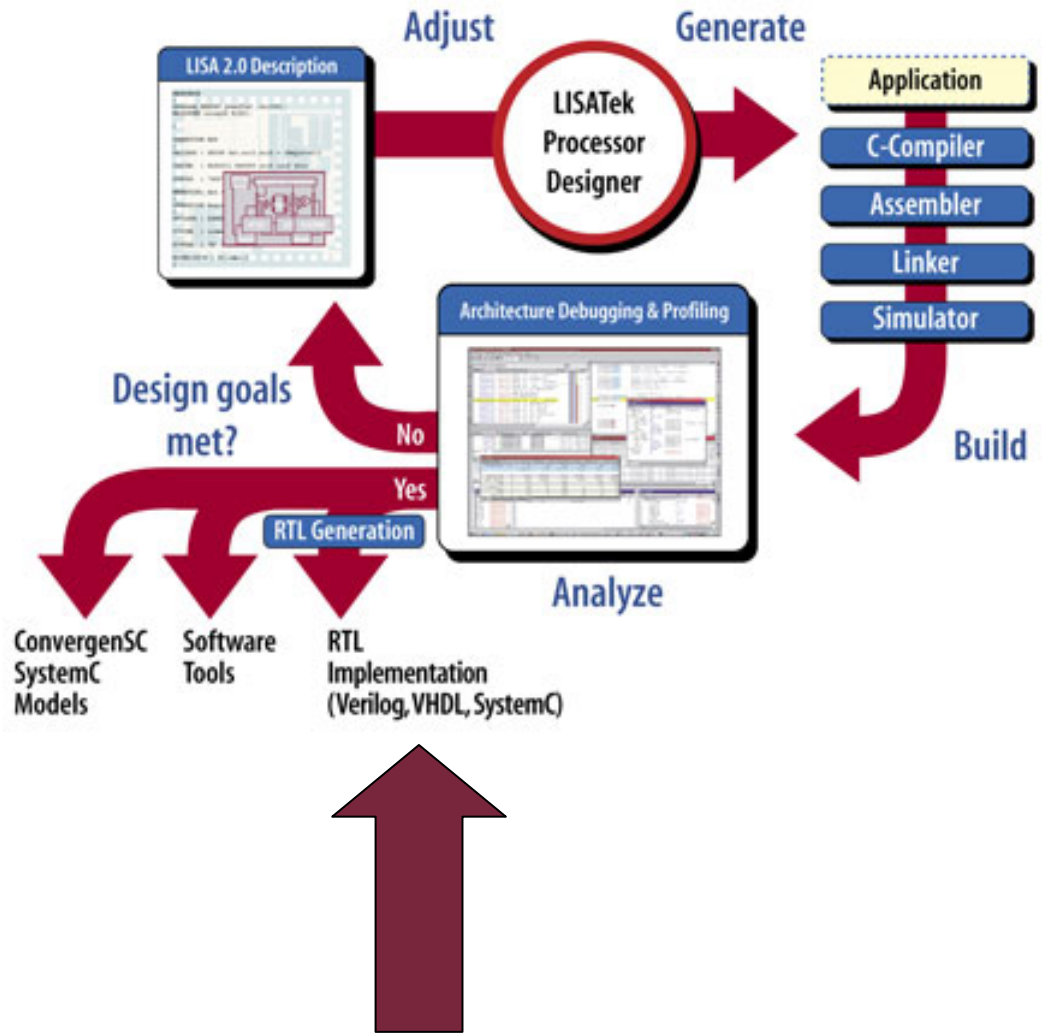
- High-level (compiler IR)
  - Enabled by CoSy´s engine concept
- Low-level (ASM):



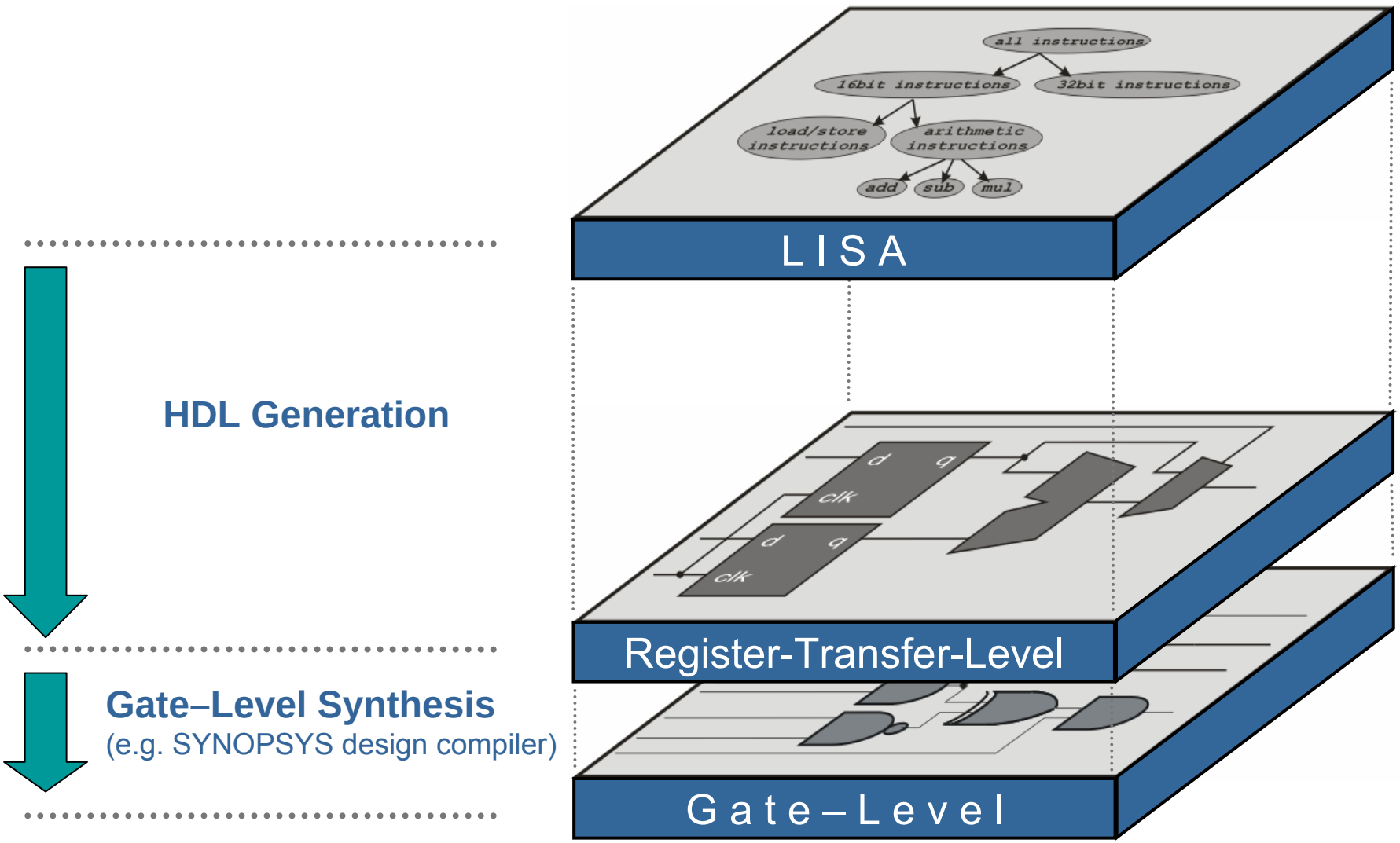


4. ASIP architecture design

ASIP implementation after exploration

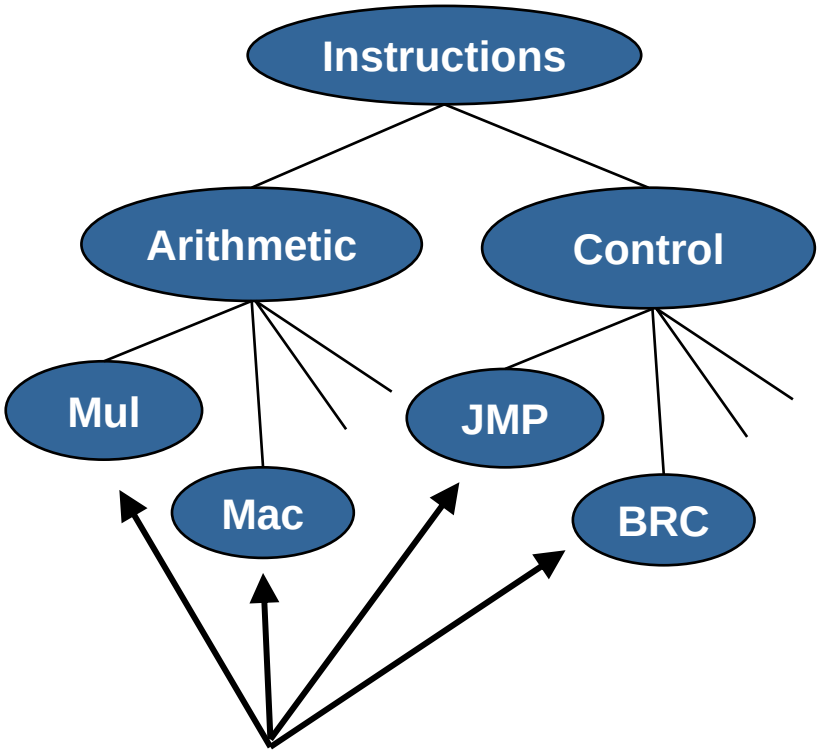


Unified Description Layer

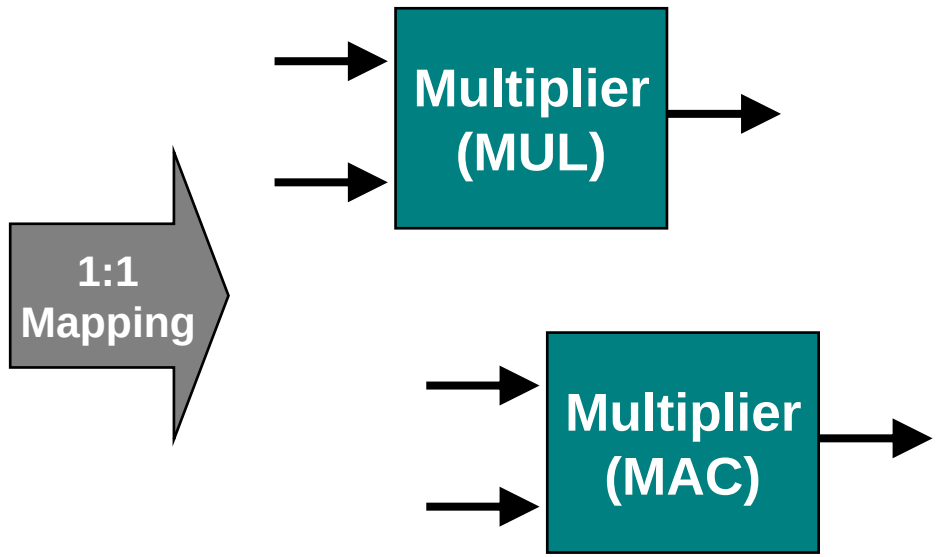


Challenges in Automated ASIP Implementation

ADL:



HDL:



Independent description of instruction behavior:

+ Efficient Design Space Exploration

Independent mapping to hardware blocks:

- Insufficient architectural efficiency by 1:1 mapping

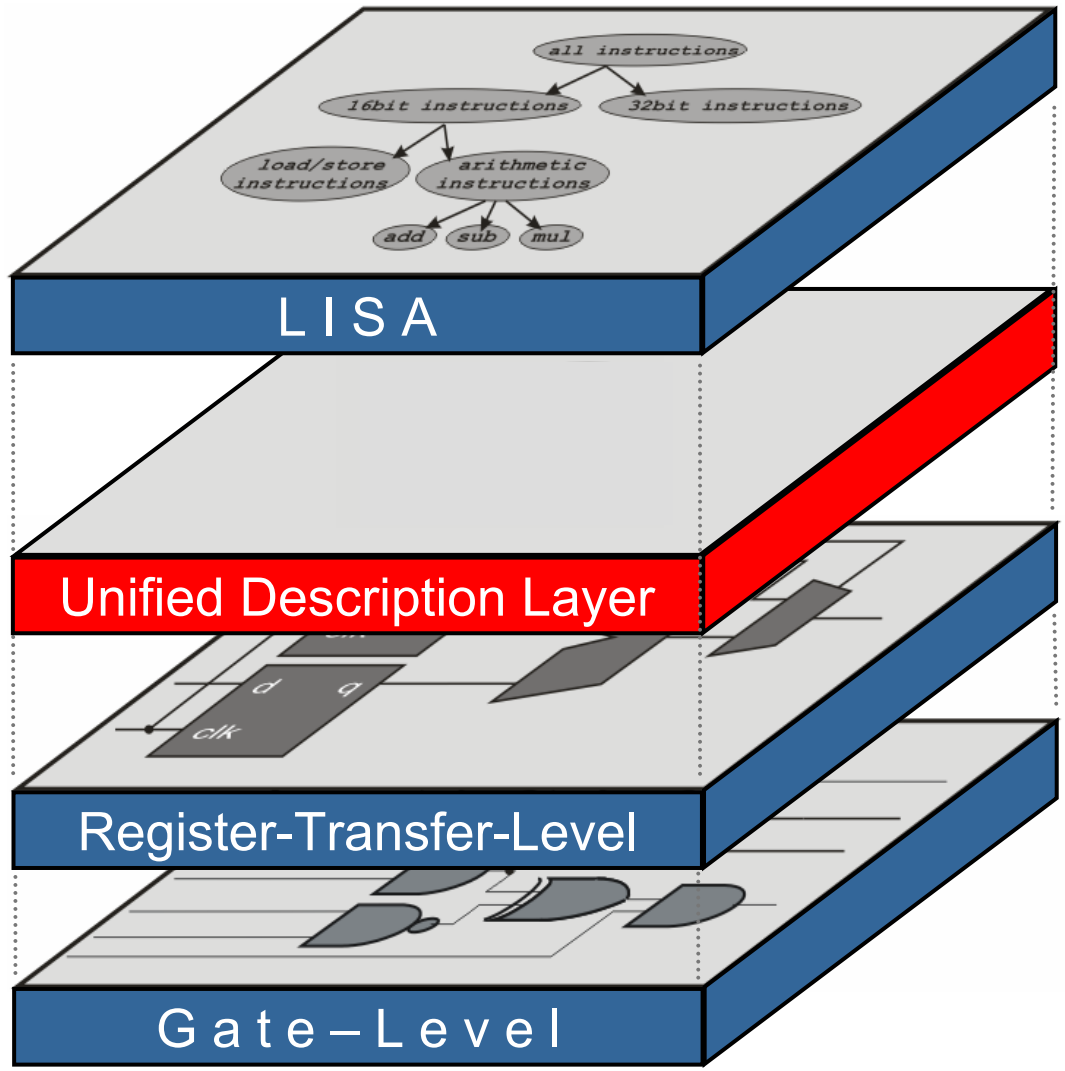
Unified Description Layer

Structure & Mapping
(incl. JTAG/DEBUG)

Optimizations

Backend (VHDL, Verilog, SystemC)

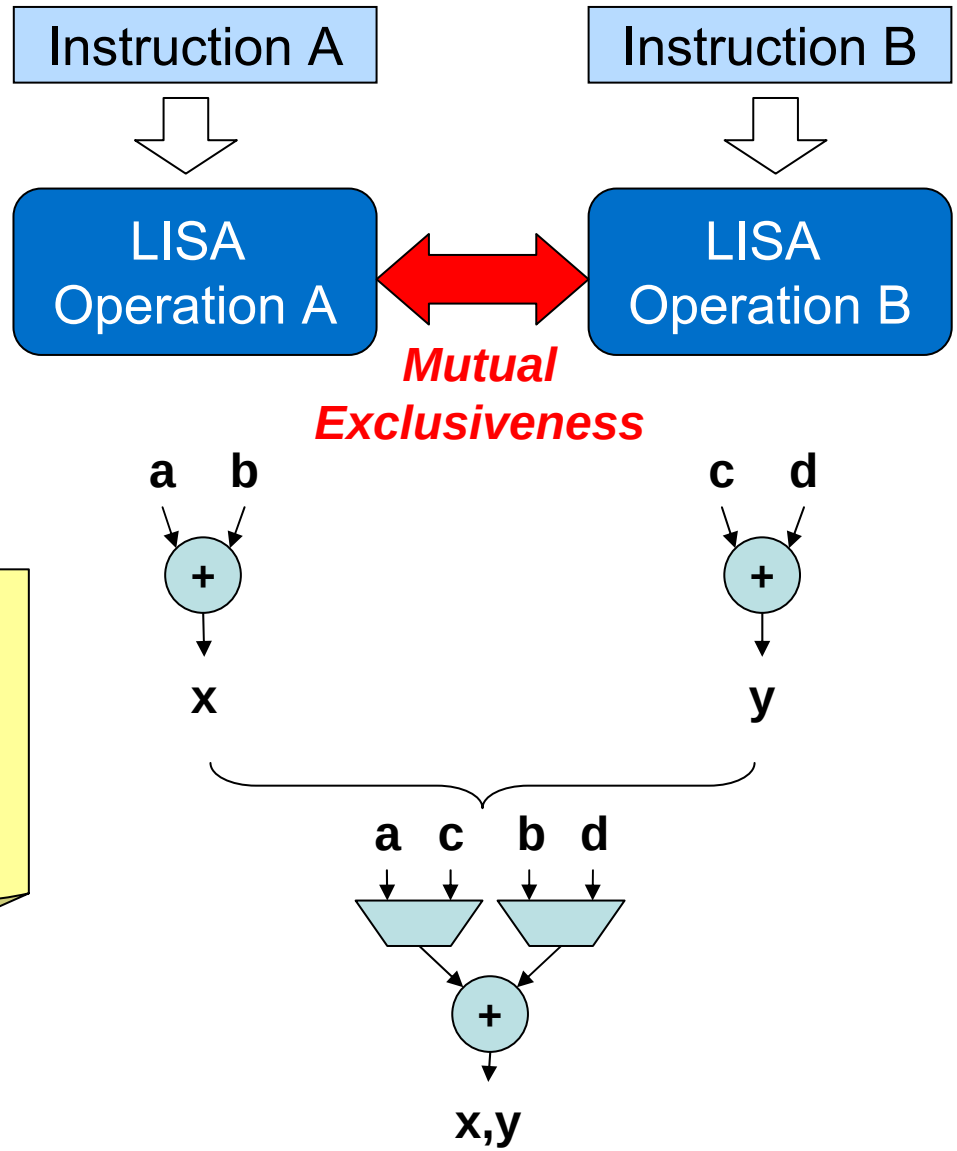
Gate–Level Synthesis
(e.g. SYNOPSYS design compiler)



Optimization strategies

LISA: separate descriptions for separate instructions

Goal: share hardware for separate instructions



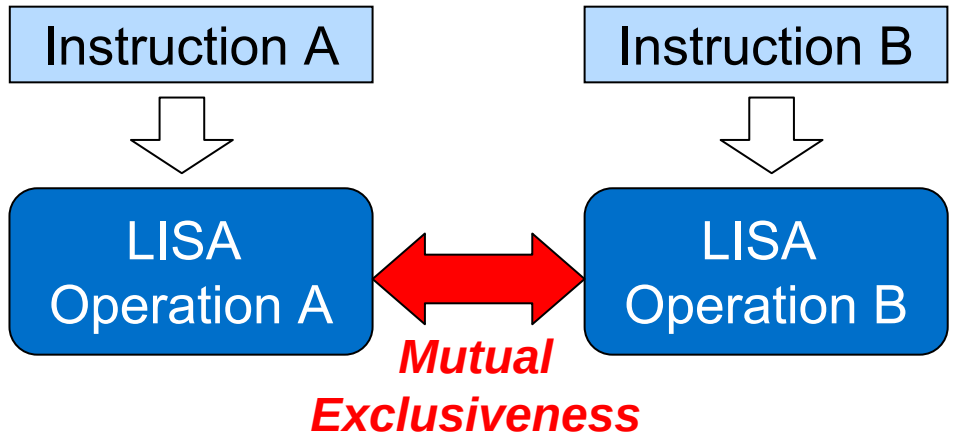
Possible Optimizations

- ALU Sharing

Optimization strategies

LISA: separate descriptions for separate instructions

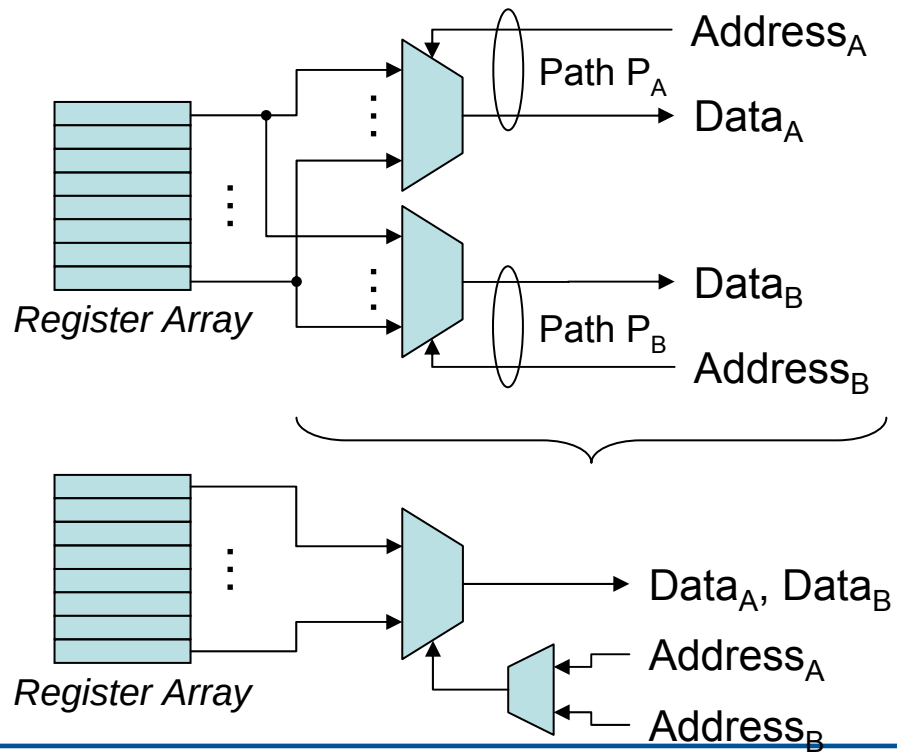
Goal: same hardware for separate instructions



Possible Optimizations

- ALU Sharing
- Path Sharing
- ...

} *Resource Sharing*



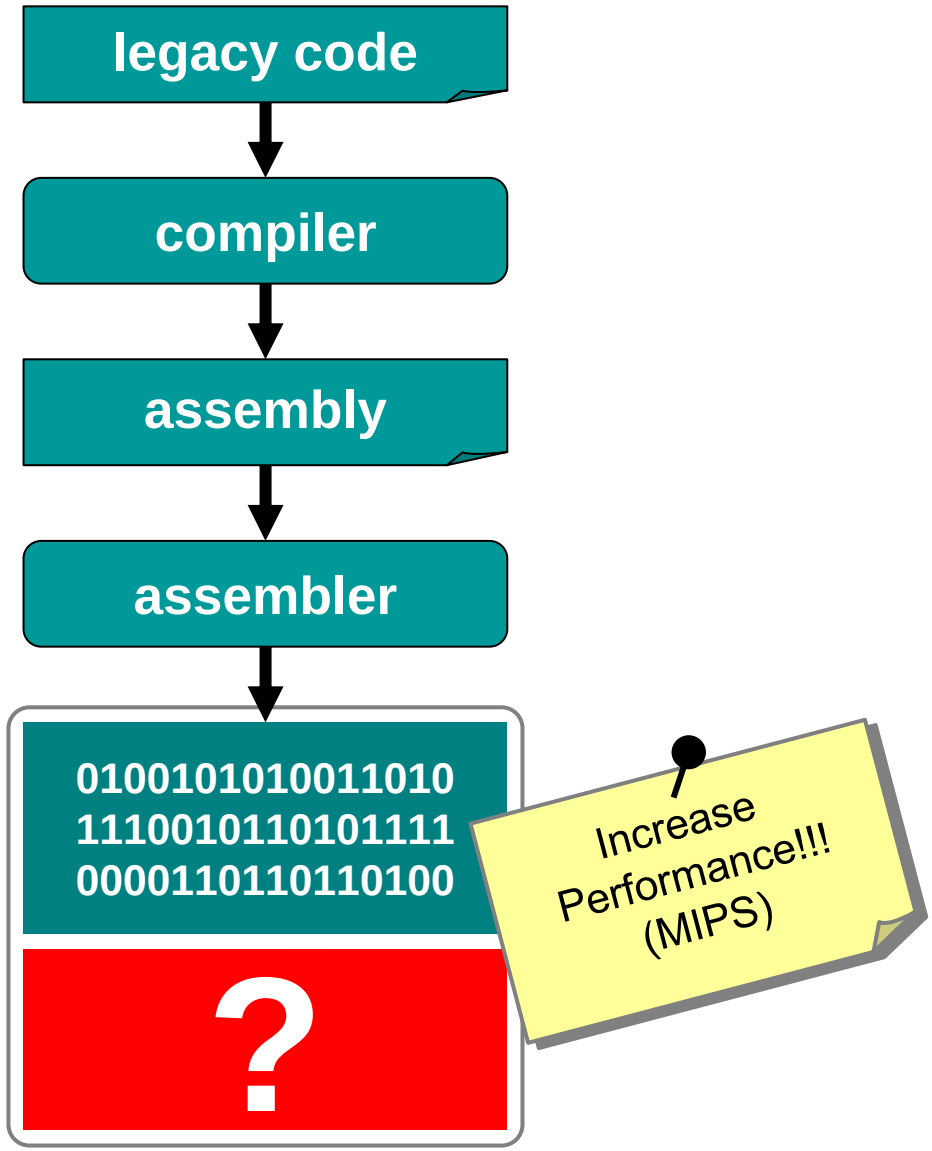
5. Case study

Project Goals:

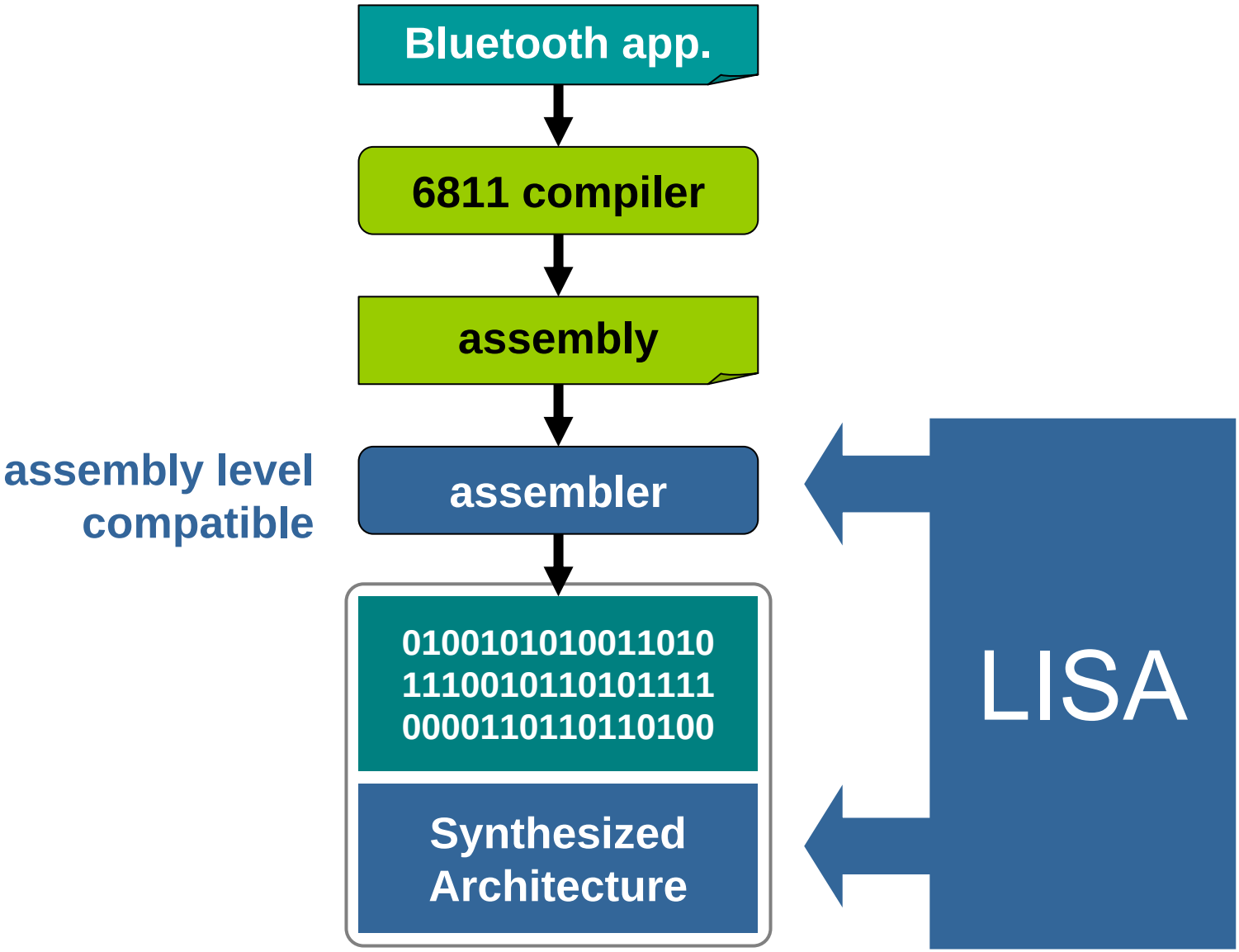
- Performance (MIPS) must be increased
- Compatibility on the assembly level
  for reuse of legacy code
  (Integration into existing tool flow)
- Royalty free design

➔ compatible architecture developed with LISA
  using RTL processor synthesis

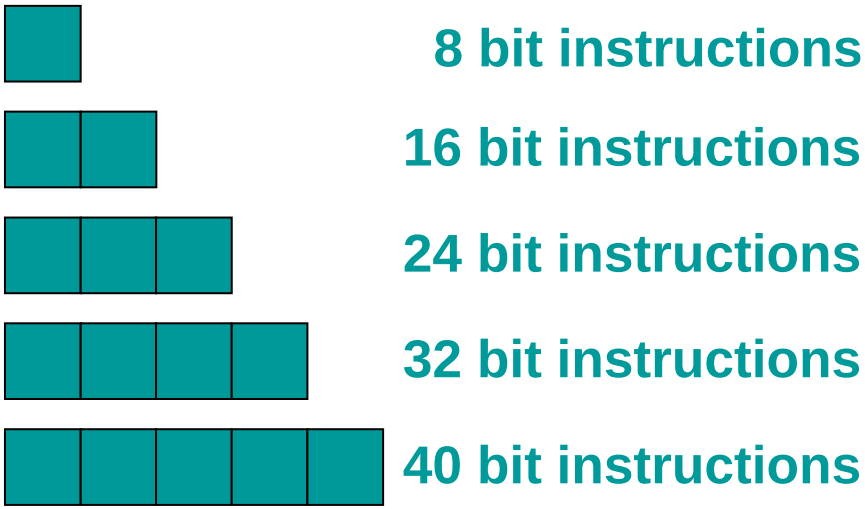
Motorola 6811



Motorola 6811



original 6811 Processor



8 bit instructions

16 bit instructions

24 bit instructions

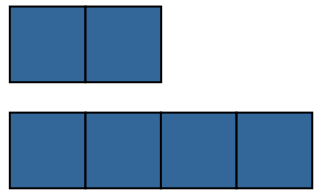
32 bit instructions

40 bit instructions

Instruction is fetched by 8 bit blocks:

➔ up to 5 cycles for fetching!

LISA 6811 Processor



16 bit instructions

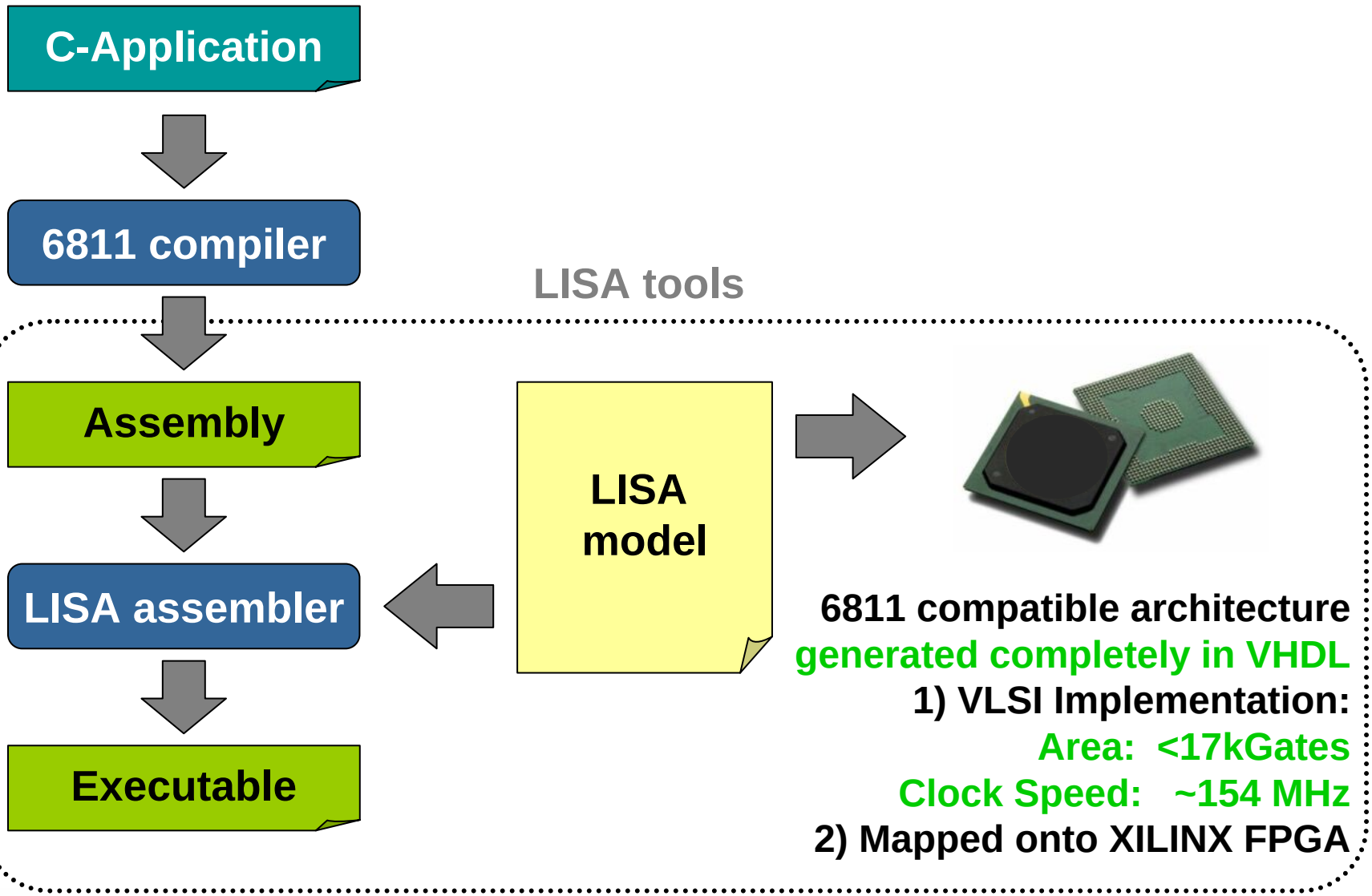
32 bit instructions

16 bit are fetched simultaneously:

➔ max 2 cycles for fetching!

\+ pipelined architecture
\+ possibility for special instructions

Tools Flow and RTL Processor Synthesis



References

- R. Leupers: *Code Optimization Techniques for Embedded Processors - Methods, Algorithms, and Tools*, Kluwer, 2000
- R. Leupers, P. Marwedel: *Retargetable Compiler Technology for Embedded Systems - Tools and Applications*, Kluwer, 2001
- A. Hoffmann, H. Meyr, R. Leupers: *Architecture Exploration for Embedded Processors with LISA*, Kluwer, 2002
- C. Rowen, S. Leibson: *Engineering the Complex SoC: Fast, Flexible Design with Configurable Processors*, Prentice Hall, 2004
- M. Gries, K. Keutzer, et al.: *Building ASIPs: The Mescal Methodology*, Springer, 2005
- P. Ienne, R. Leupers (eds.): *Customizable and Configurable Embedded Processor Cores*, Morgan Kaufmann, to appear 2006