Design Tools for Application Specific Embedded Processors

Sharad Malik

Princeton University

EMSOFT '02

Grenoble

10/9/02

The Back End!

The Back End!

The Sixth no sooner had begun

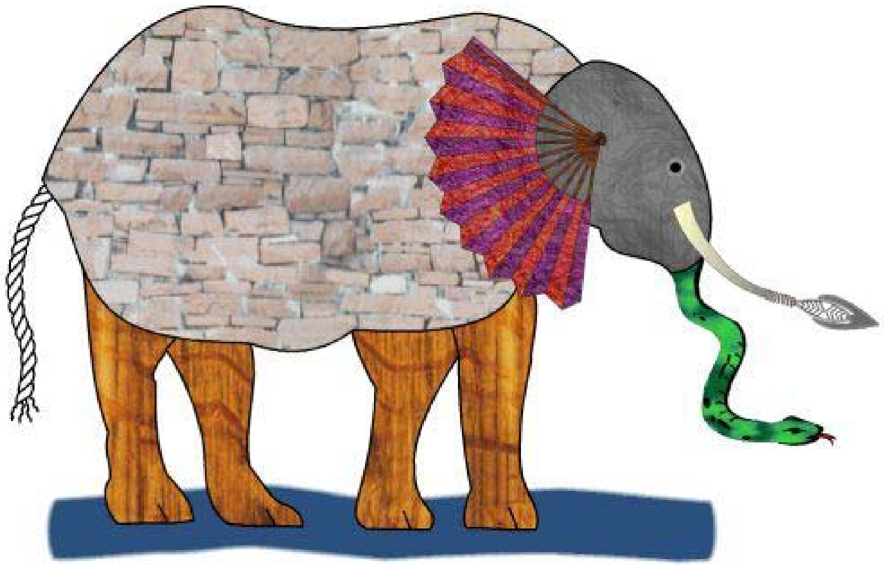
About the beast to grope

Than, seizing on the swinging tail

That fell within his scope,

"I see," quoth he, "the Elephant

Is very like a rope!"



MESCAL Project

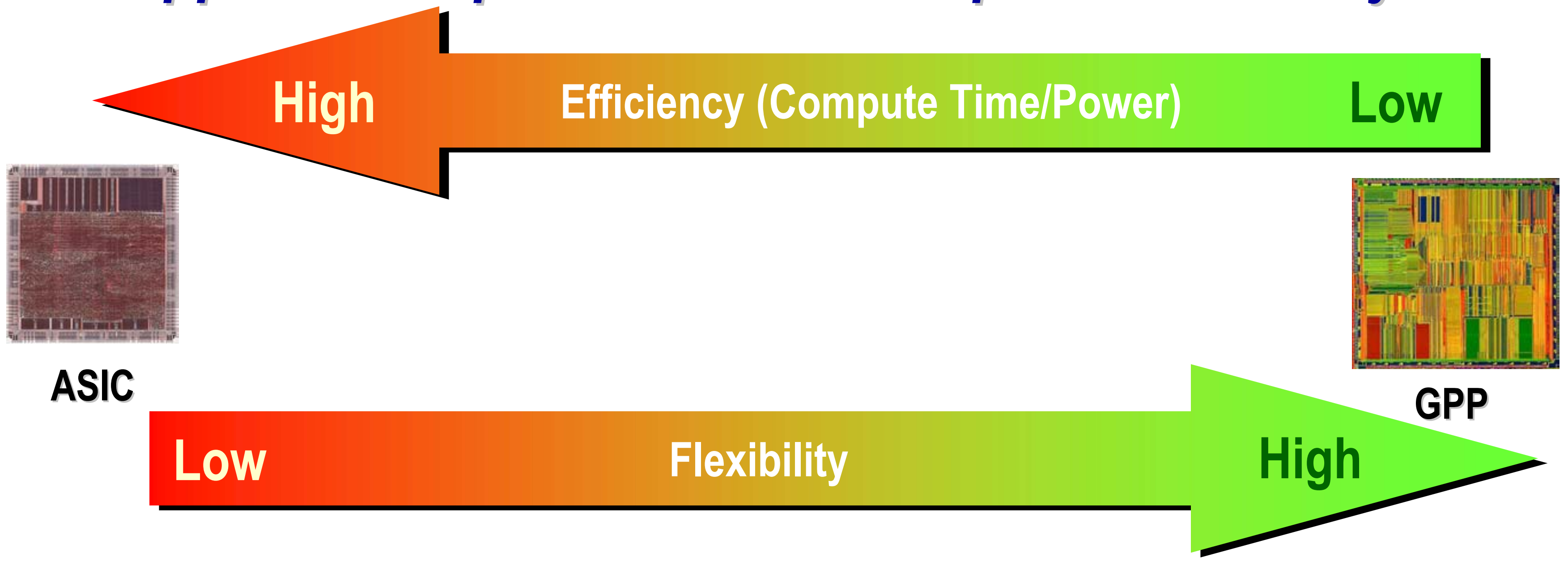
Modern Embedded Systems, Compilers Architectures and Languages

Part of the Gigascale Silicon Research Center (GSRC)

Funded by DARPA and MARCO (Micrelectronic Advanced Research Consortium)

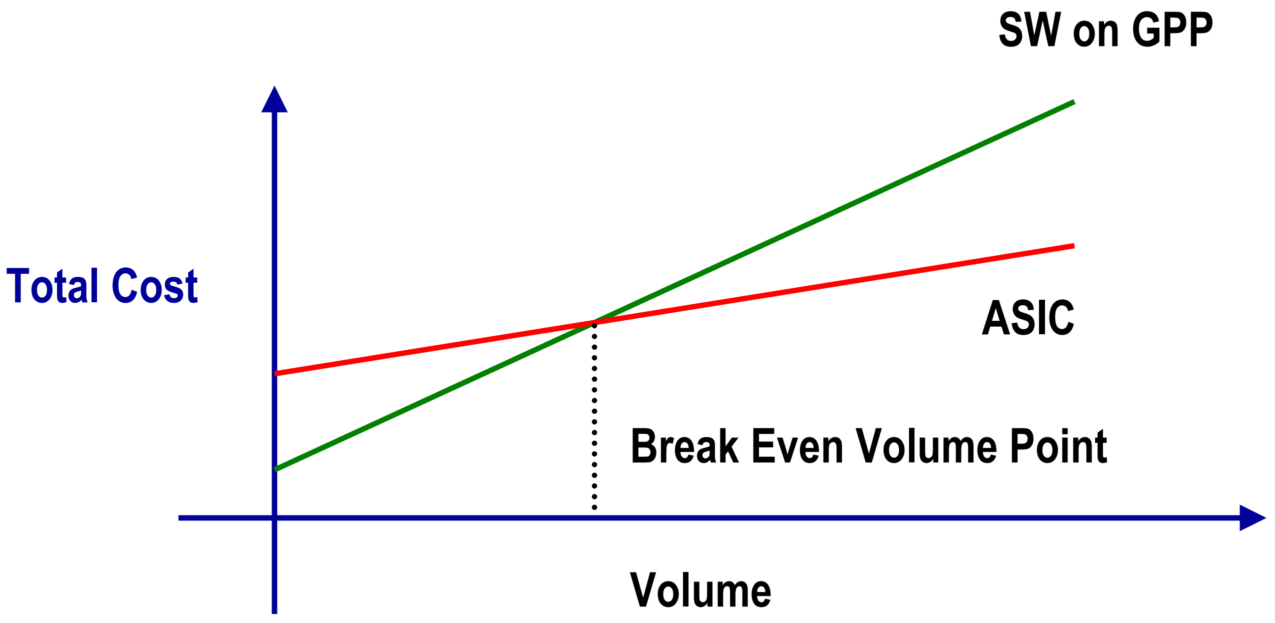
- Berkeley
  - Faculty: K. Keutzer, R. Newton
  - Post-Docs: C. Kulkarni, M. Gries
  - Grad Students: Y. Jin, A. Mihal, M. Moskewicz, W. Plishker, K. Ravindran, N. Shah, M. Tsai (graduating), S. Weber,
- Princeton
  - Faculty: D. August, S. Malik, Li-Shiuan Peh
  - Students: Z. Huang, W. Qin, S. Rajagopalan, S. Triantafyllis, M. Vachharajani, H. Wang, S. Wang, X. Zhu

The Application Specific/General Purpose Dichotomy



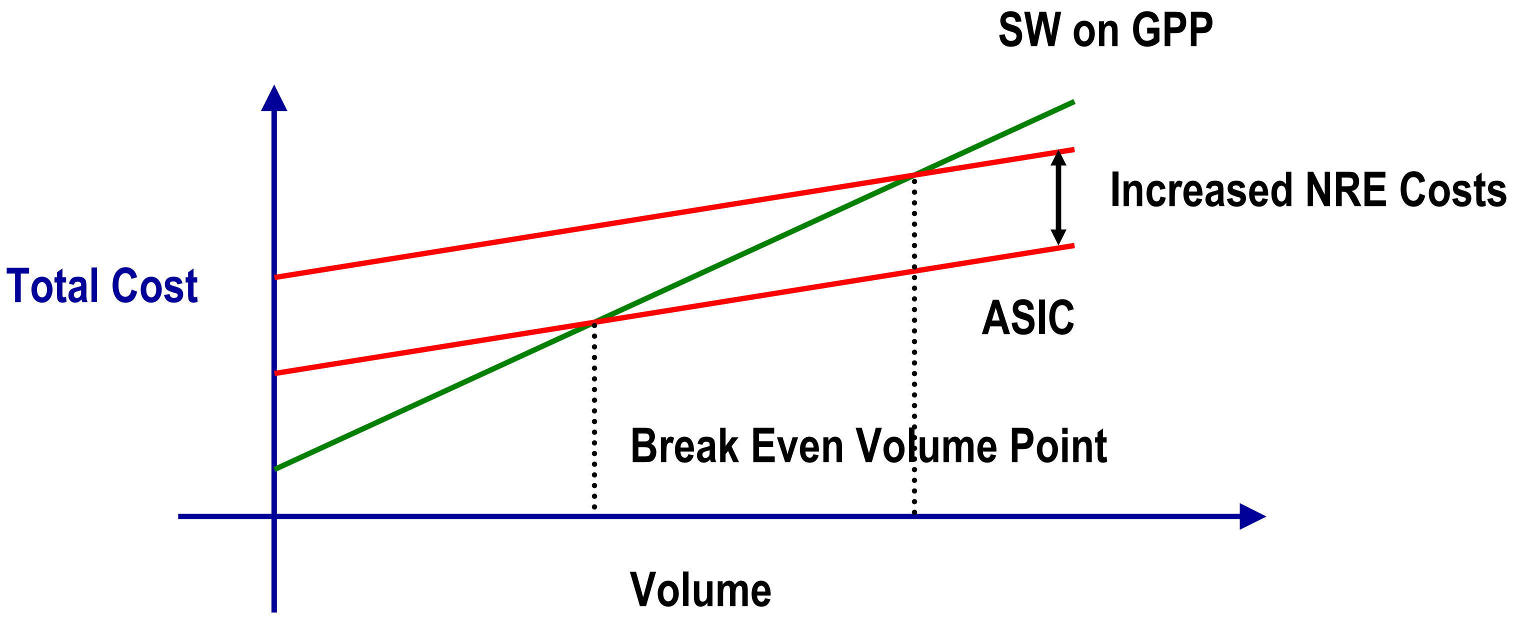
- Application Specific Integrated Circuits (ASICs)
  - Efficient, non-flexible, implementation of specific applications
  - Only choice for high-performance/low power systems
- General Purpose Processors (GPPs)
  - Flexible, inefficient, platforms for software implementations
  - Only choice for complex (dynamic data structures, recursion) or dynamically changing algorithms

The Middle Ground



- **Costs**
  - ASICs
    - Non-recurring costs (design, manufacturing set-up) higher
    - Per-unit manufacturing costs lower
  - SW on GPP
    - Non-recurring costs lower
    - Per-unit cost higher

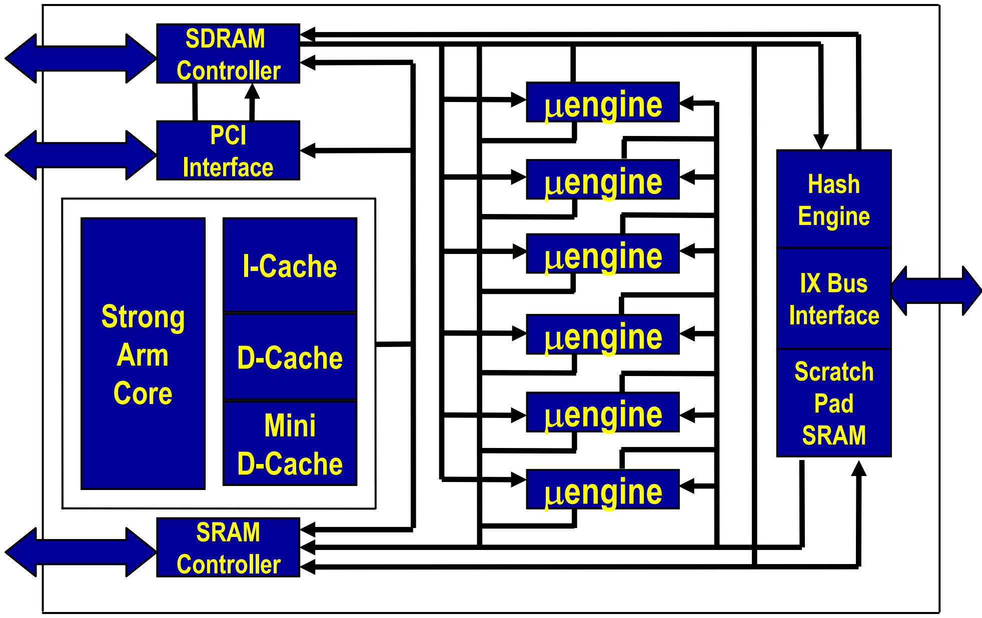
Rising ASIC Non-Recurring Costs



- Increase in ASIC Non-recurring Costs
  - Design getting harder
    - Higher engineering costs
  - More expensive design tools
  - Increasing mask costs
- Traditional ASICs moving to programmable processors
  - High efficiency requirements require significant application specific processor specialization

From ASICs to ASIPs: The Next Design Discontinuity

- ◆ Develop platforms that allow for amortization of design costs over multiple generations
- ◆ Make platforms *programmable* so that they have maximum flexibility with minimum overhead
- ◆ Already happening in some domains – network and communication processors
- ◆ Current design practices ad-hoc, with little tool support.



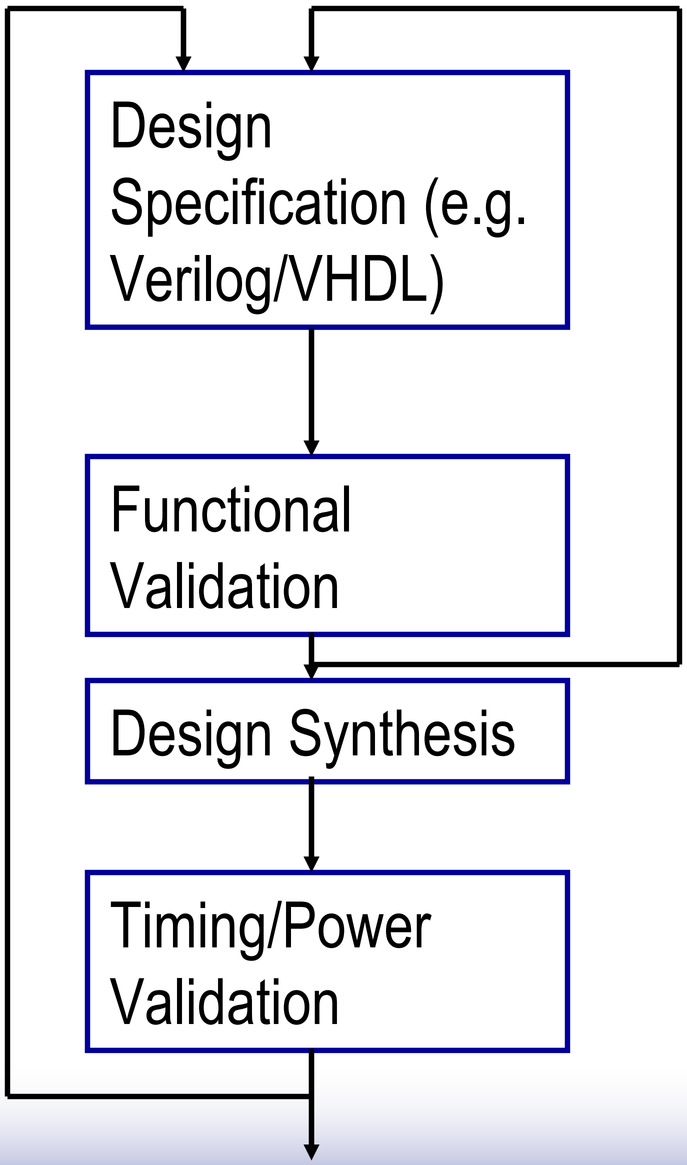
Intel IXP 1200

The MESCAL Project Mission:

To bring a disciplined methodology, and a supporting tool set, to the development of application-specific programmable platforms.

ASIC → ASIP: Tool Requirements

- **Key aspects:**
  - **Ability to specify the design in a formal specification.**
  - **Ability to validate the design for functionality/timing/power *using this specification*.**
- **ASIP requirements:**
  - **Ability to formally specify the ASIP**
  - **Ability to validate this design for functionality/timing/power using this specification**



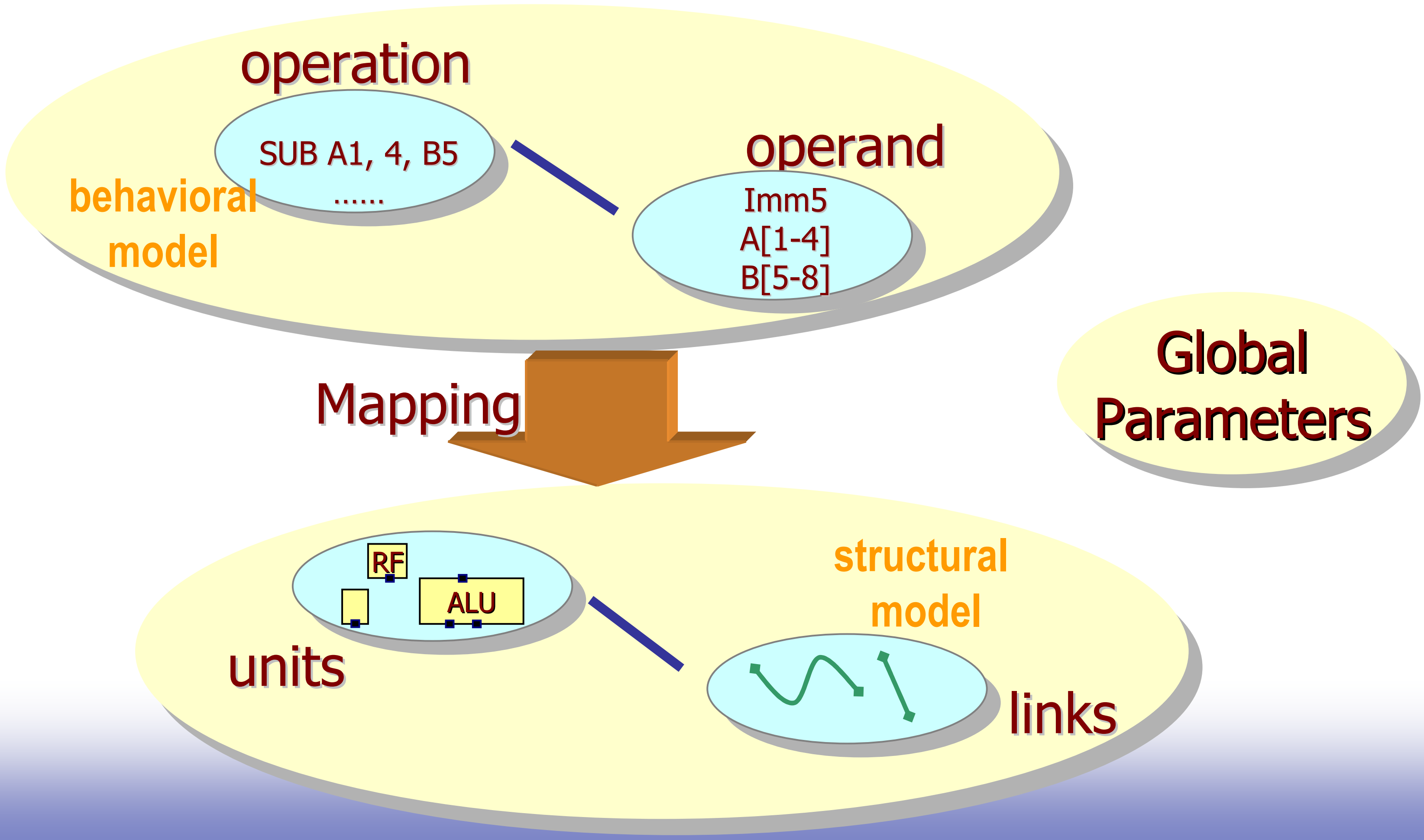
Specification Space

5 Axes of the Architectural Design Space

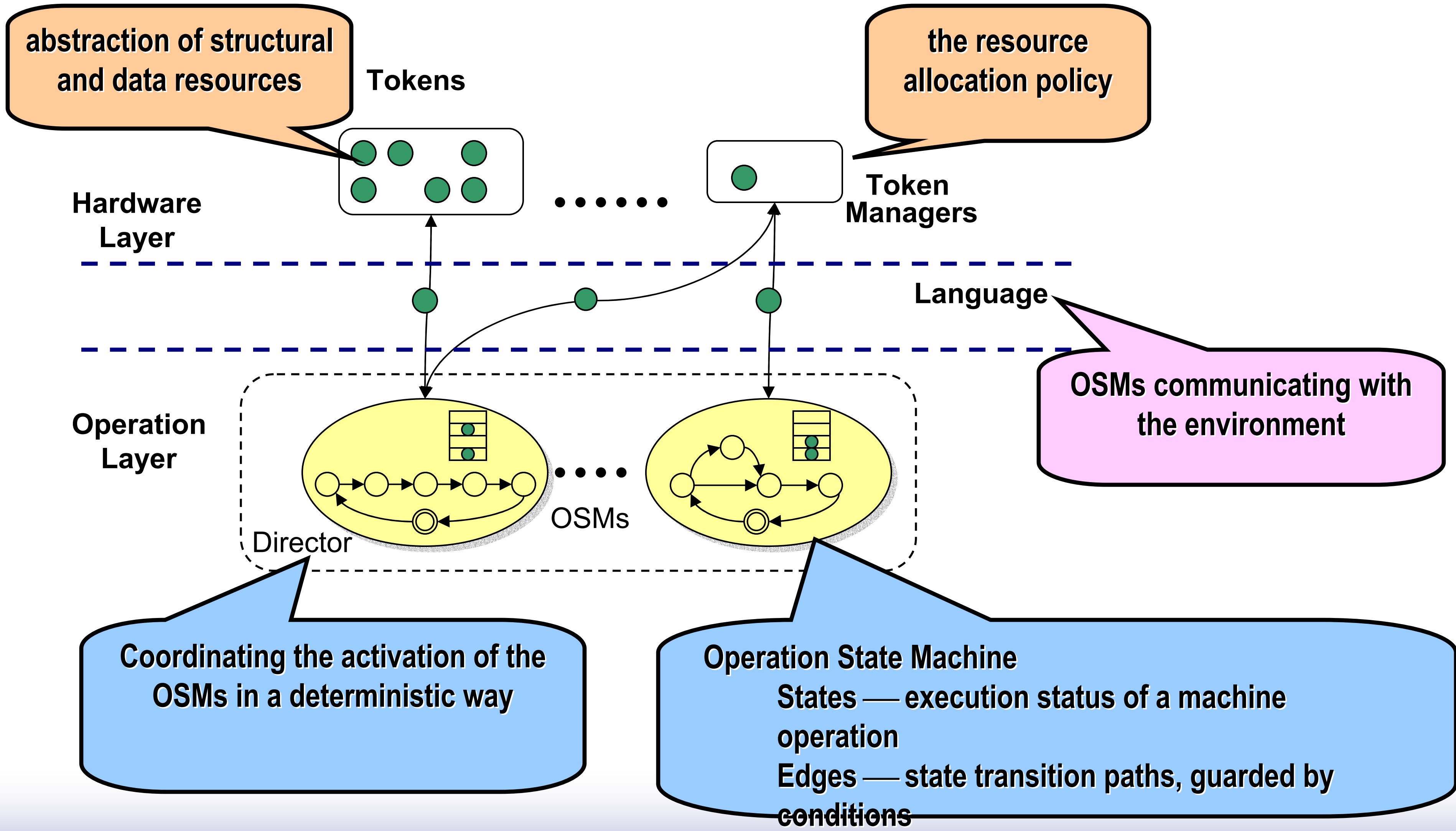
- Approaches to Parallel Processing
  - Processing Element (PE) level – threads and processes
  - Instruction-level
  - Bit-level
- Elements of Special Purpose Hardware
  - e.g. Hash engine
- Structure of Memory Architectures
  - Distributed special purpose memories
  - Memory hierarchies
- Types of On-Chip Communication Mechanisms
  - Buses
  - Networks
- Use of Peripherals
  - Peripheral behavior and interaction with computation

MESCAL Architecture Description (MAD)

- Formal specification of instructions and their use of micro-architectural resources.
  - Computation instructions
  - Communication instructions



Operation State Machine (OSM)

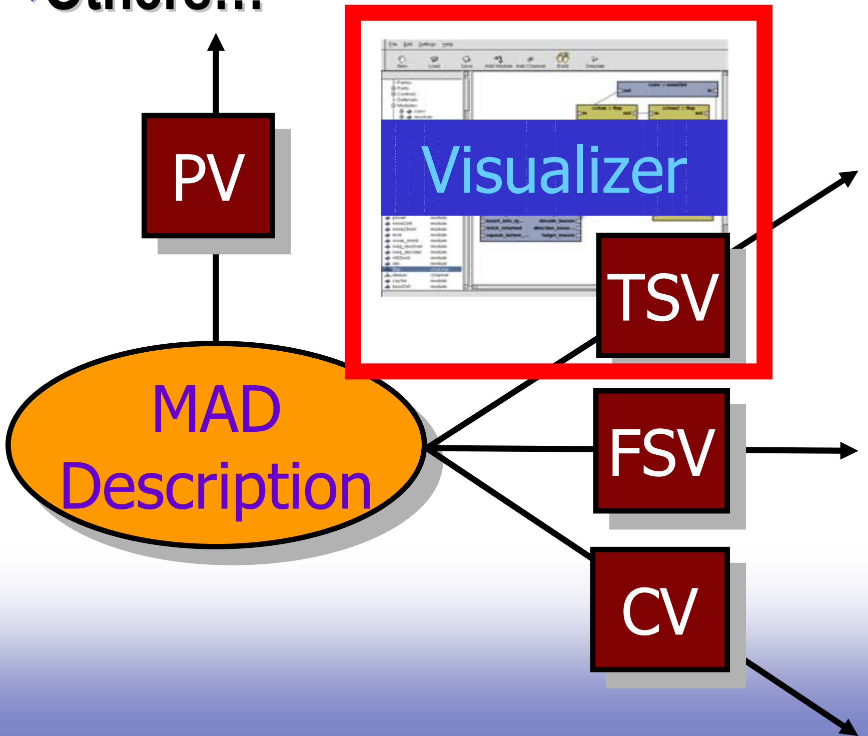

abstraction of structural and data resources
Tokens
the resource allocation policy
Hardware Layer
Token Managers
Language
OSMs communicating with the environment
Operation Layer
OSMs
Director
Coordinating the activation of the OSMs in a deterministic way
Operation State Machine
States — execution status of a machine operation
Edges — state transition paths, guarded by conditions

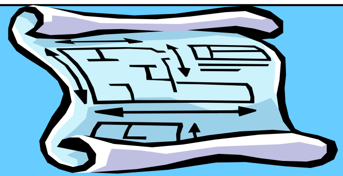
OSM (cont.)

- Cleanly separation of micro-processors into two layers
  - Operation layer — Instruction semantics, resource consumption and timing
  - Hardware layer — Pipeline control, external interface
- Easy modeling of common pipeline behaviors
  - Distributes control policies into token managers
  - Supports superscalar, VLIW, Multi-threaded architecture modeling
- Formal model
  - Easy to synthesize simulators.
  - Easy to analyze and extract model properties for verification purposes and for compiler optimization.

MAD Views

- However, tools have different **views** of the architecture
- **View Generators** create views from MAD
  - Compiler View
  - Timing Simulator View
  - Functional Simulator View
  - Others…





Document View

➡ Emulation
➡ Code generator
➡ Instruction Selection
➡ Register classes

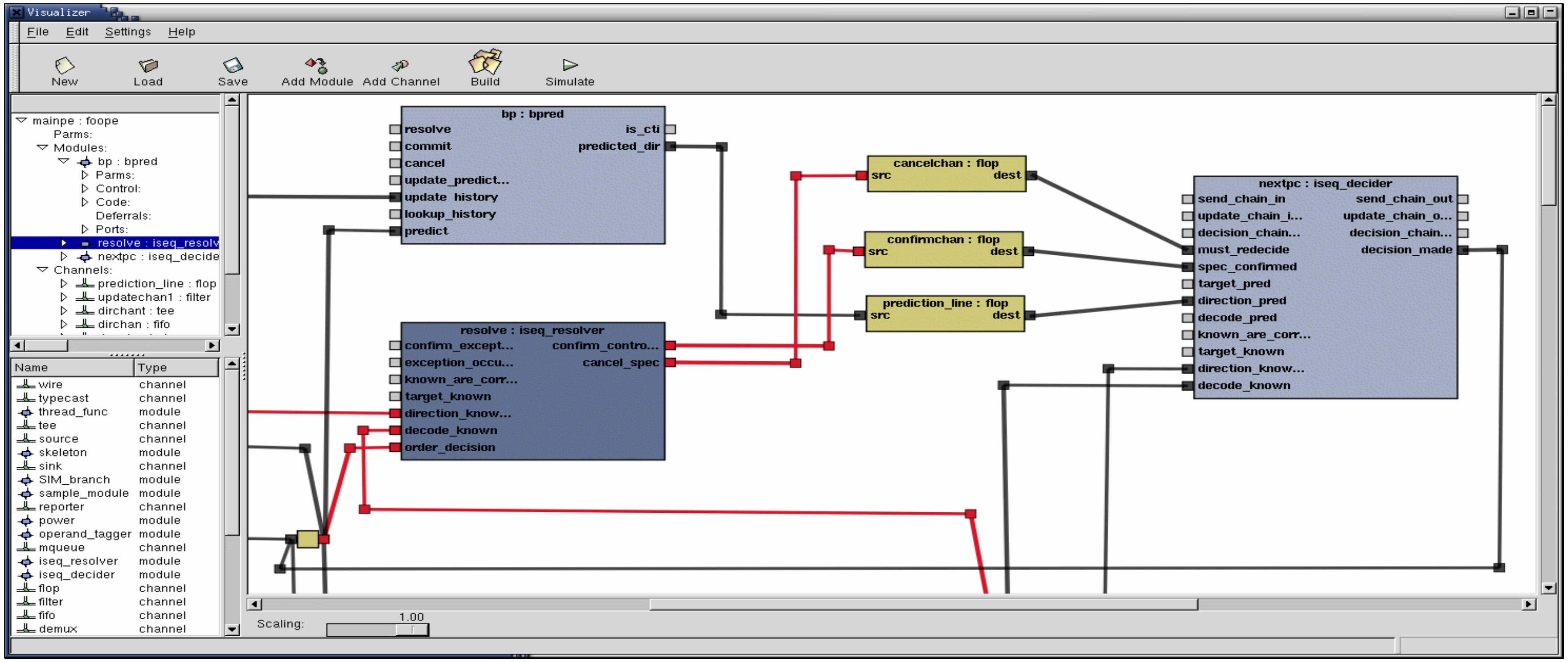
Semantics View

ADD:

Resource	0	1	2	3
RegFile	X			X
ALU		X	X	

Scheduling View

Liberty Timing Simulator View



Name	Type
wire	channel
typecast	channel
thread_func	module
tee	channel
source	channel
skeleton	module
sink	channel
SIM_branch	module
sample_module	module
reporter	channel
power	module
operand_tagger	module
mqueue	channel
iseq_resolver	module
iseq_decider	module
flop	channel
filter	channel
fifo	channel
demux	channel

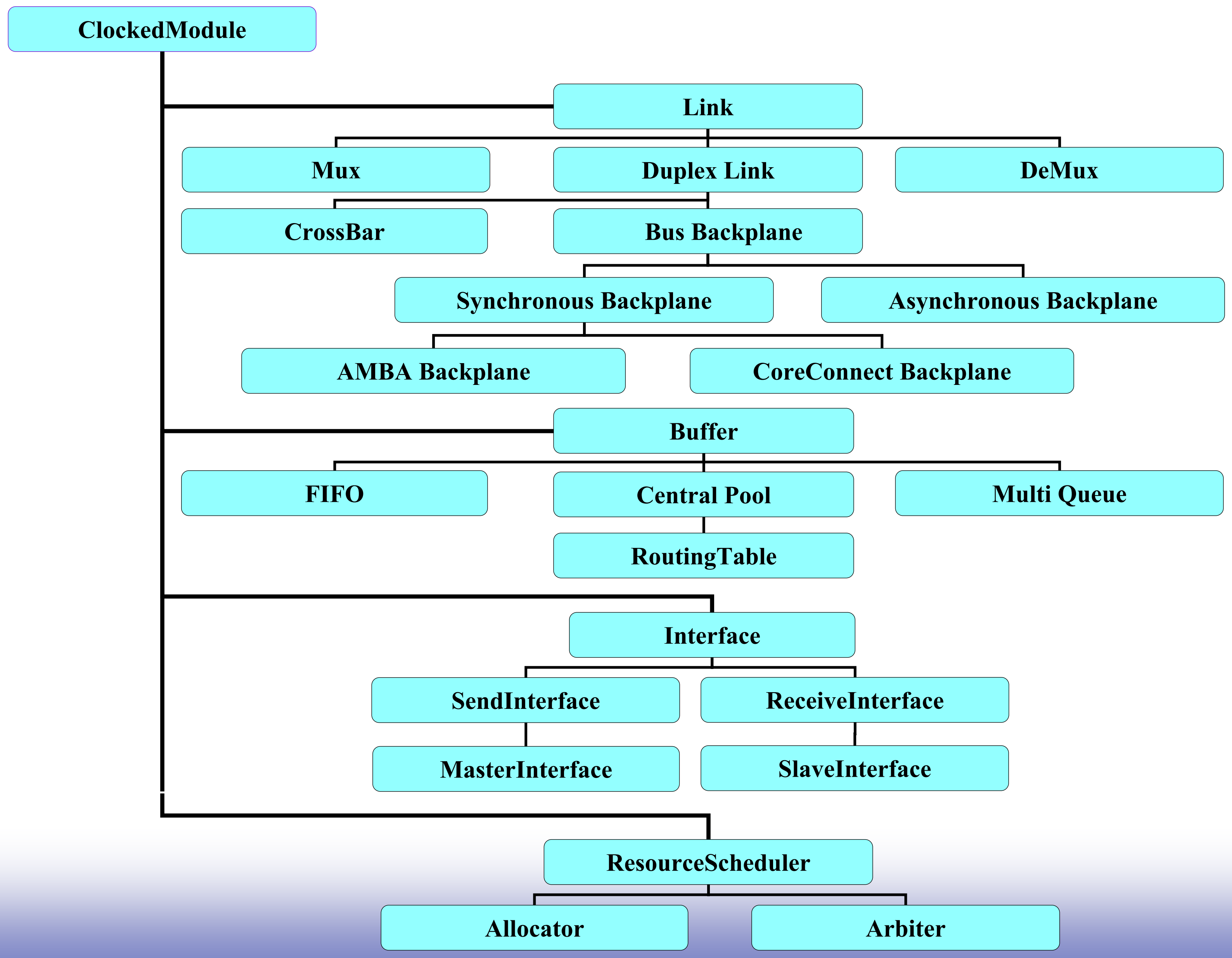
- Architectural and micro-architectural specification
  - Modular, extensible natural specification
    - Netlist of micro-architectural modules
  - Formal concurrency semantics (Synchronous Reactive)
  - Retargetable simulator synthesis

More at: http://liberty.princeton.edu

On-chip Communication Architectures

- Rich diversity
  - From buses to packet-switching networks
  - Need to consider these in a single framework
- No established micro-architecture primitives
  - Identify a small sufficient set of primitives
    - Links
    - Buffers
    - Resource Scheduler
    - Interface
  - Using object-oriented analysis
    - Commonality analysis
    - Differentiation
    - Classification by inheritance
  - Build micro-architectural models with significant reuse

Class Inheritance Hierarchy of OCAs

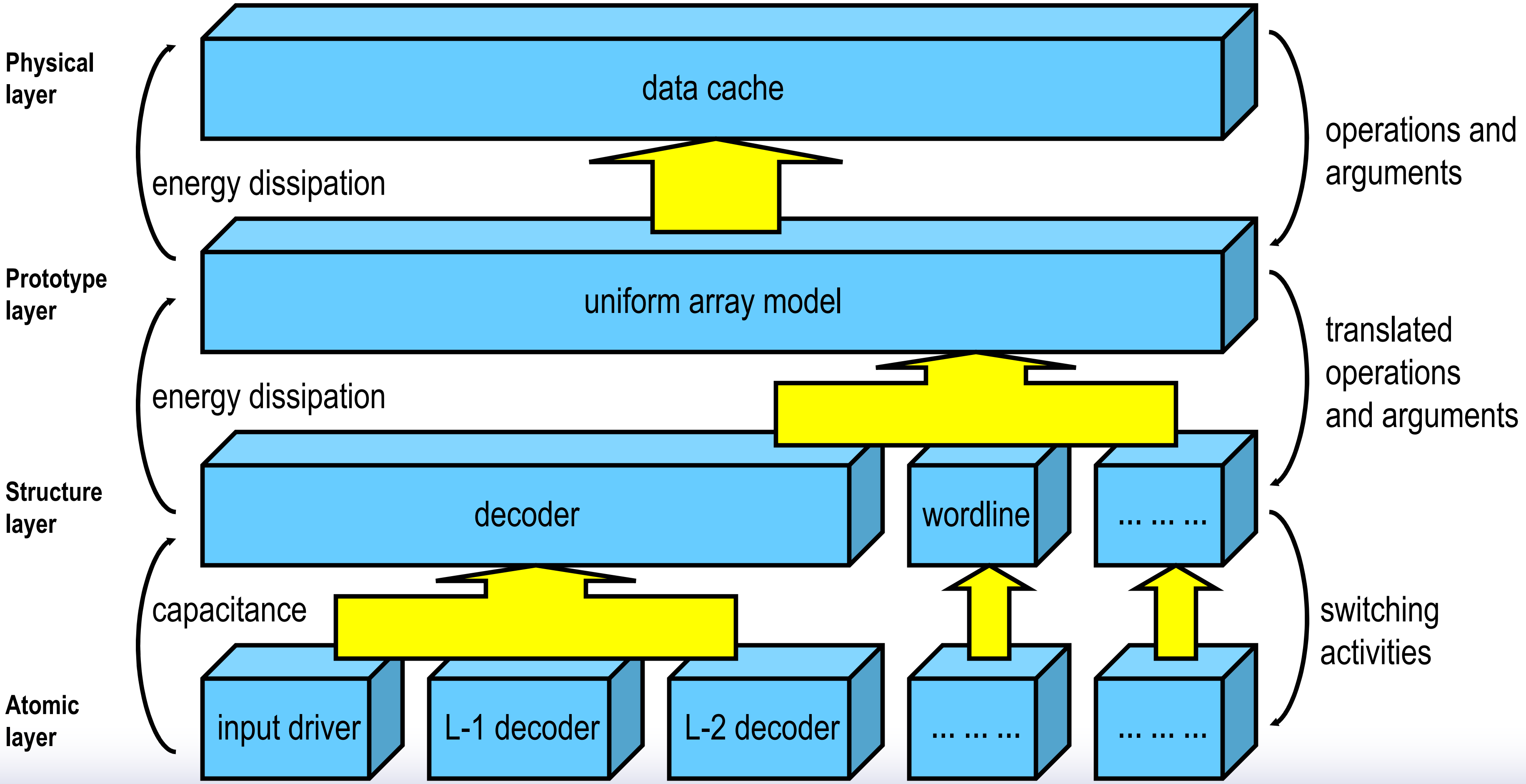


Issues in Power Modeling and Simulation

- **Simulator needs to be architecture/microarchitecture retargetable**
- **Models need to be technology retargetable**
- **Model needs to reflect design reuse**
- **Hierarchical modeling methodology to enable this:**
  - atomic layer: collection of components that switch together
    - Have same switching characteristics
    - Physical parameters are technology dependent
  - structure layer: circuit building blocks, e.g. decoder, tri-state buffer, etc.
  - prototype layer: an abstract function unit with structure but no functionality, e.g. general memory array
  - physical layer: a concrete function unit, e.g. data cache

An Example

◆ **Data cache power model hierarchy and its data flow**



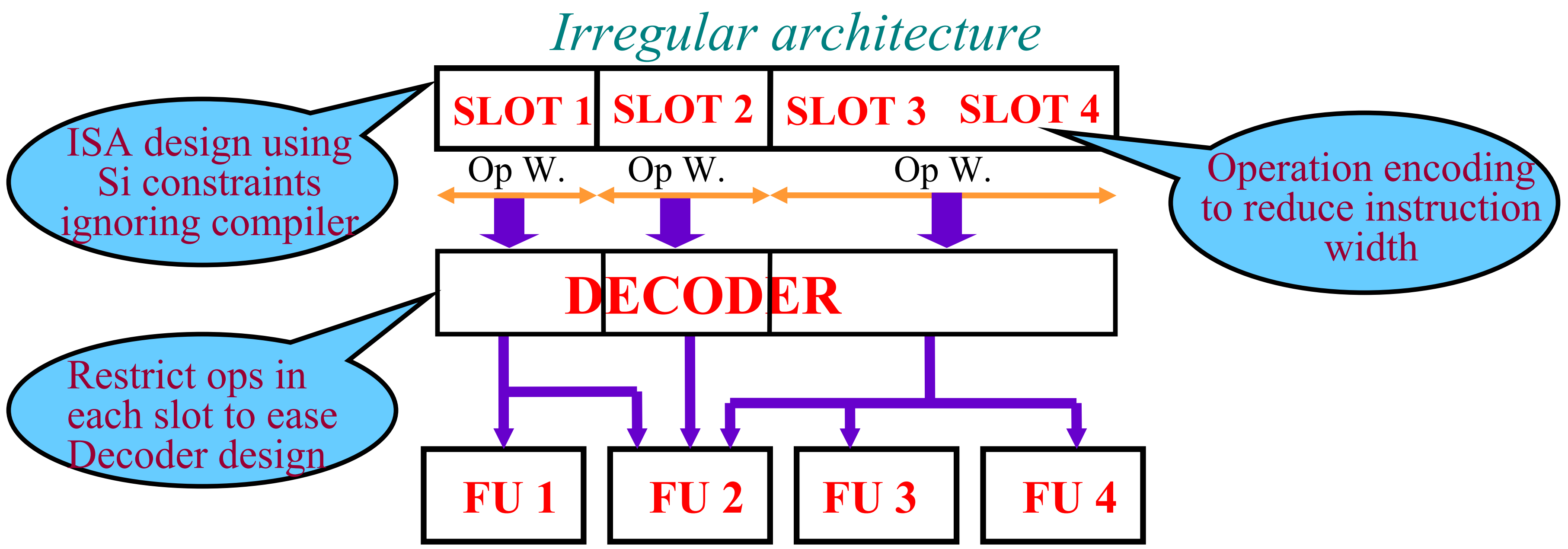
Advantages of Modeling Hierarchy

- Separation of microarchitecture dependency and technology dependency
- Fine-grained modeling granularity
  - Arbitrary tradeoff between accuracy and efficiency
- Reusability
  - Structure layer — building blocks
  - Prototype layer — templates
- Easy to maintain and extend

Compiler Issues

- Individual processors have significant specialization
  - Irregular instruction level parallelism
  - Specialized functional units and memories
- Multiprocessor compilation needs to synthesize communication and synchronization code
  - Use architectural specification of the on-chip communication architectures for direct translation

Handling Irregular ILP


Irregular architecture
ISA design using Si constraints ignoring compiler
SLOT 1
SLOT 2
SLOT 3
SLOT 4
Op W.
Op W.
Op W.
Operation encoding to reduce instruction width
DECODER
Restrict ops in each slot to ease Decoder design
FU 1
FU 2
FU 3
FU 4

- **Problem Statement**
  - **Input : The set of all Instructions in the ISA where each instruction represents a set of operations that can be issued in parallel**
  - **Output: Assign a set of artificial resources such that**
    - **operations that can be issued in parallel do not share an artificial resource**
    - **operations that cannot be issued in parallel share an artificial resource**
  - **Use a graph coloring algorithm to construct artificial resources**
- **Irregular ILP is converted into regular ILP for use in conventional VLIW resource based schedulers**

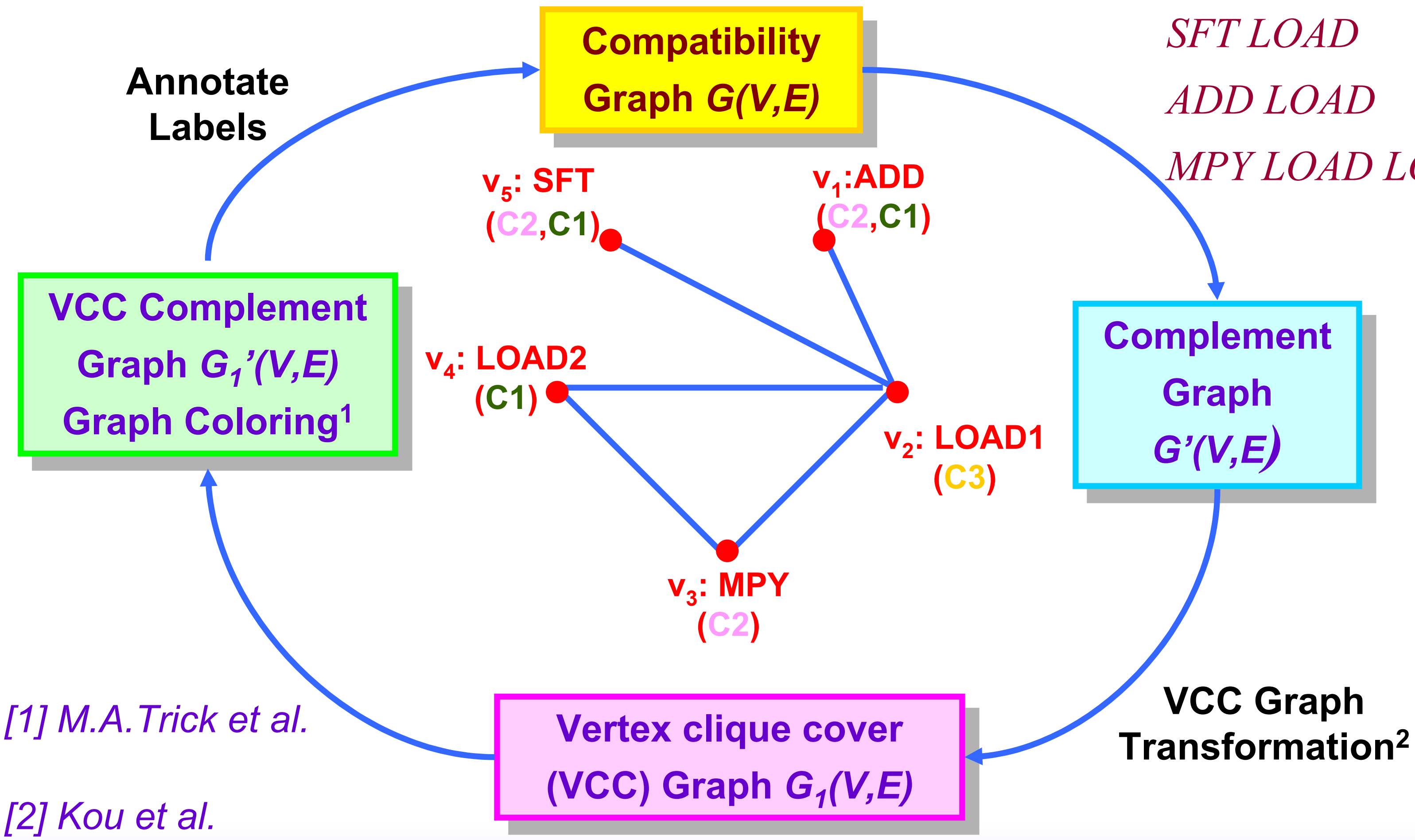
Artificial Resource Assignment Algorithm using Graph Coloring

Ops that can be issued in parallel

SFT LOAD

ADD LOAD

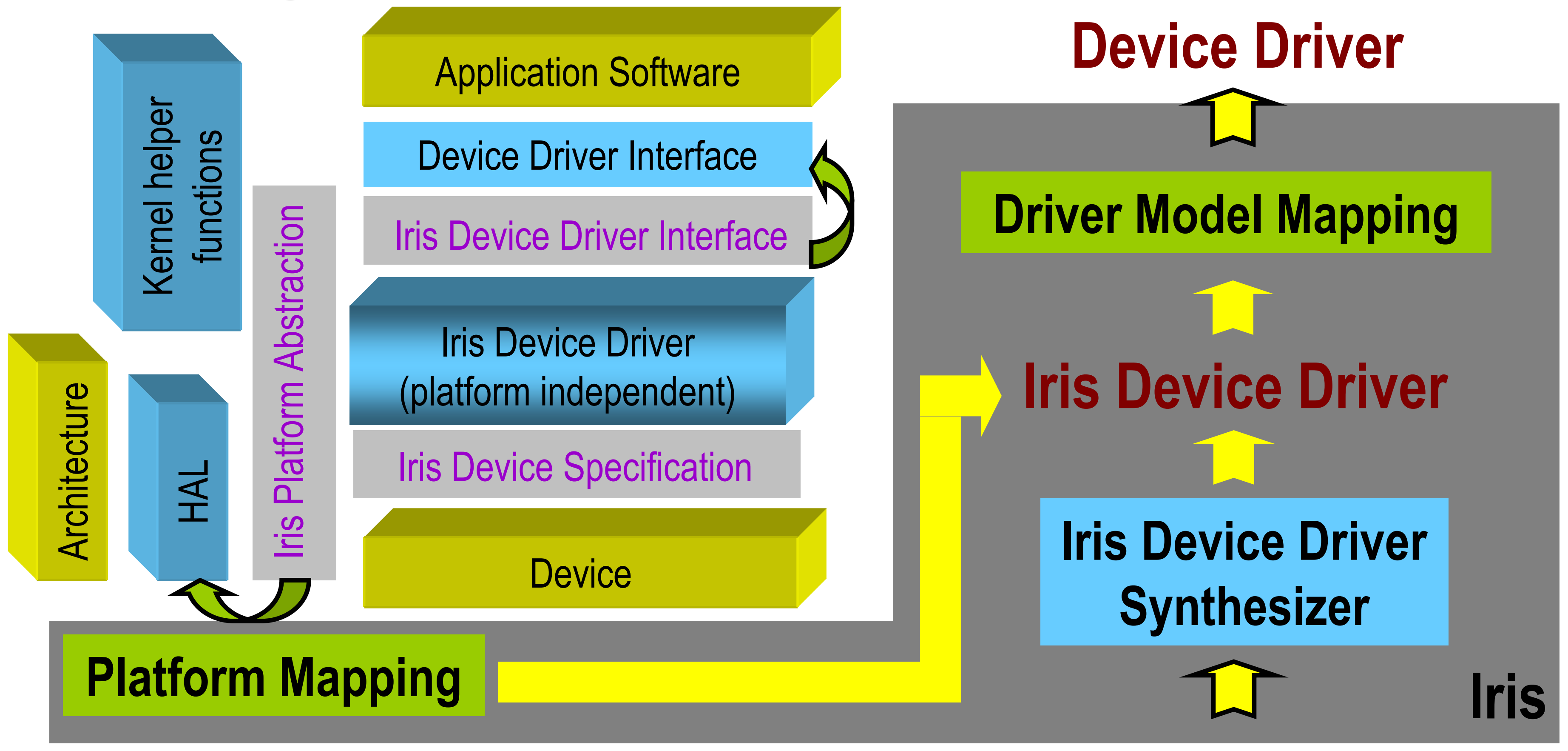
MPY LOAD LOAD


Compatibility Graph $G(V,E)$
Complement Graph $G'(V,E)$
VCC Graph Transformation[2]
Vertex clique cover (VCC) Graph $G_1(V,E)$
VCC Complement Graph $G_1'(V,E)$ Graph Coloring[1]
Annotate Labels
v_5: SFT (C2,C1)
v_1:ADD (C2,C1)
v_4: LOAD2 (C1)
v_2: LOAD1 (C3)
v_3: MPY (C2)

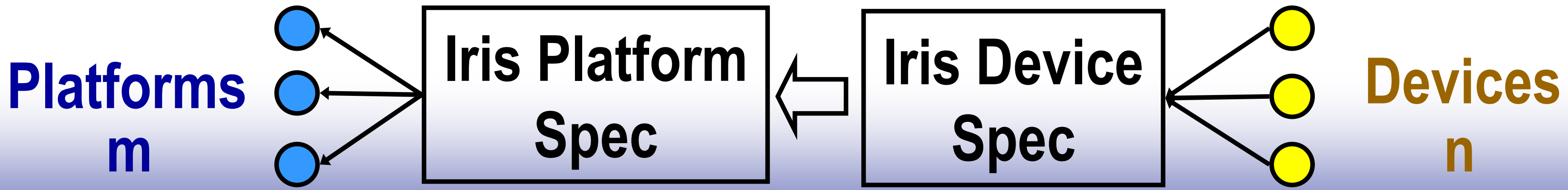
[1] M.A.Trick et al.

[2] Kou et al.

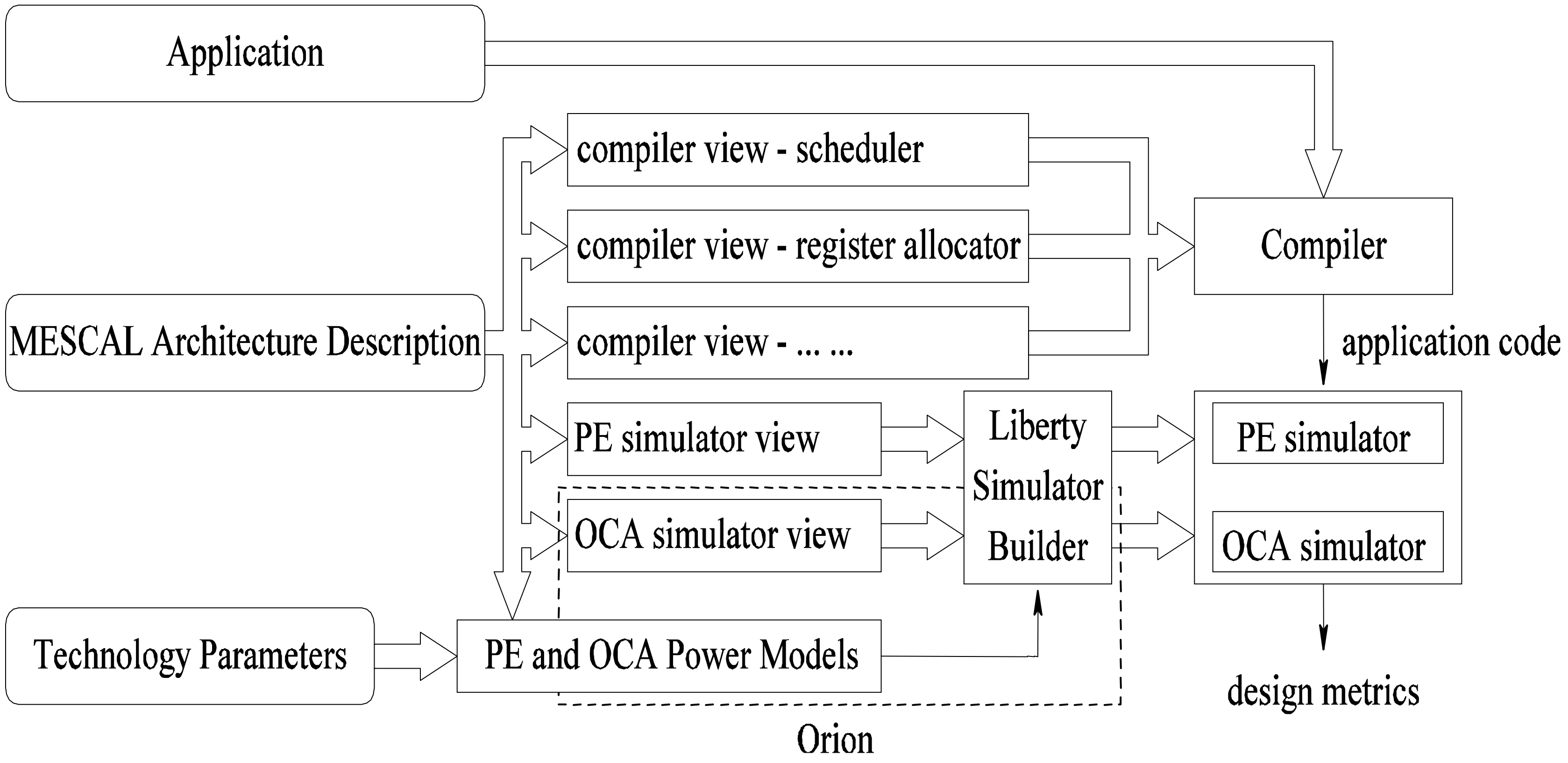
Iris: Integration of Peripheral Devices



m+n vs. m*n



Toolset Summary



Summary

- Rising non-recurring costs for ASICs leads to:
  - Lower ASIC design starts
  - Increase in ASIPs
- ASIP design methodology with
  - Formal specification of ASIPs
  - Development of software evaluation environments from this specification
    - Simulators
      - Timing and power
      - Computation and communication architectures
    - Compilers
      - Architecture and microarchitecture retargetable
      - Computation and communication
    - Peripherals
      - Synthesis of drivers from device behavior specifications