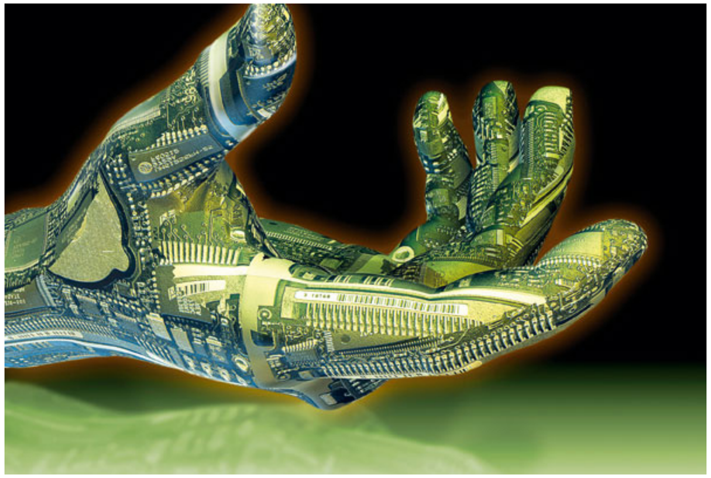
Workload-basierte Verlässlichkeitsanalyse für eingebettete Prozessoren

Oliver Bringmann, Stefan Stattelmann, Björn Sander

Universität Tübingen /
FZI Forschungszentrum Informatik

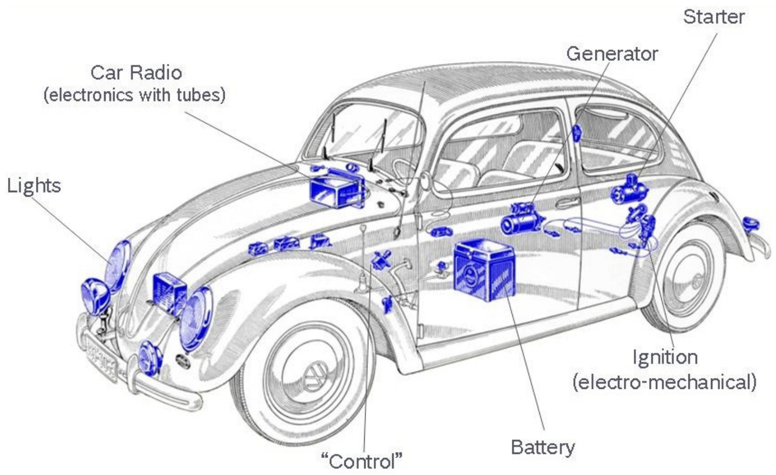


Outline

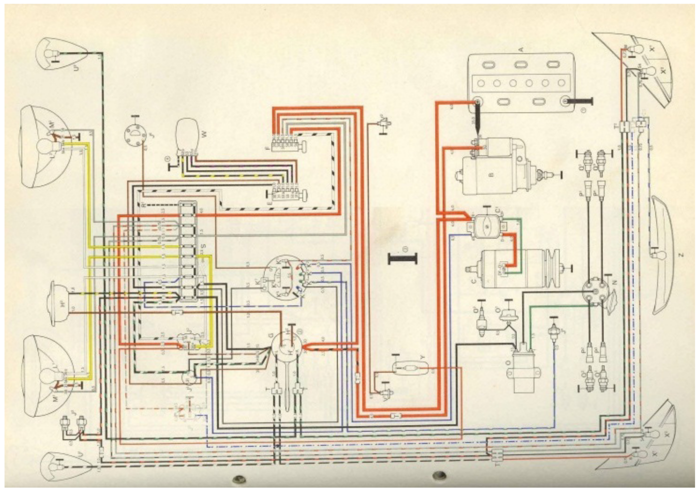
- Embedded Software: Power & Temperature
- System-Level Simulation of Non-Functional Properties by Host-Compiled Execution
- Application-Dependent Power and Temperature Simulation Framework
- Areas of Application

Challenges: Automotive Electronics

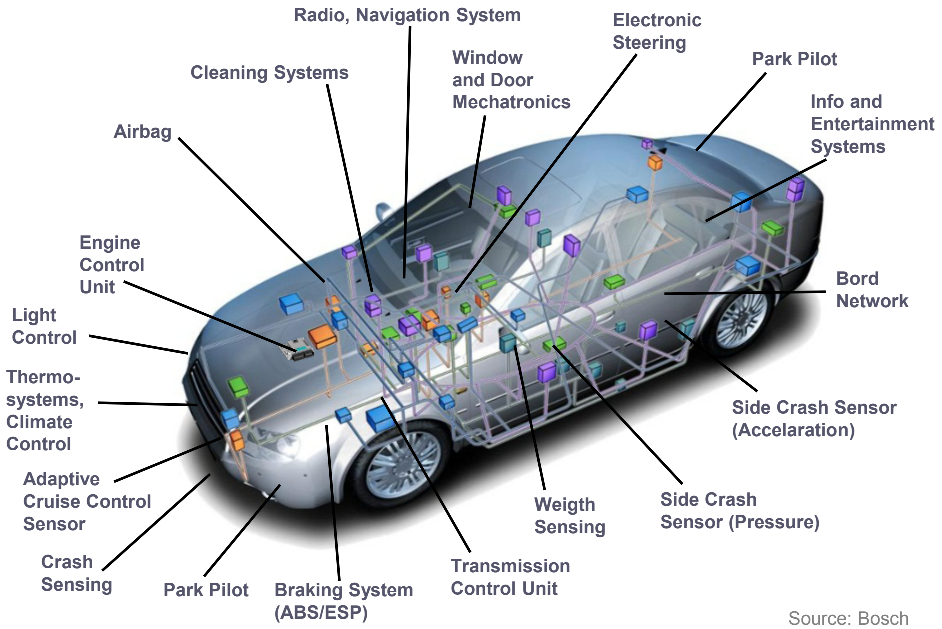
Yesterday …



Starter
Generator
Car Radio (electronics with tubes)
Lights
Ignition (electro-mechanical)
Battery
"Control"



… and today



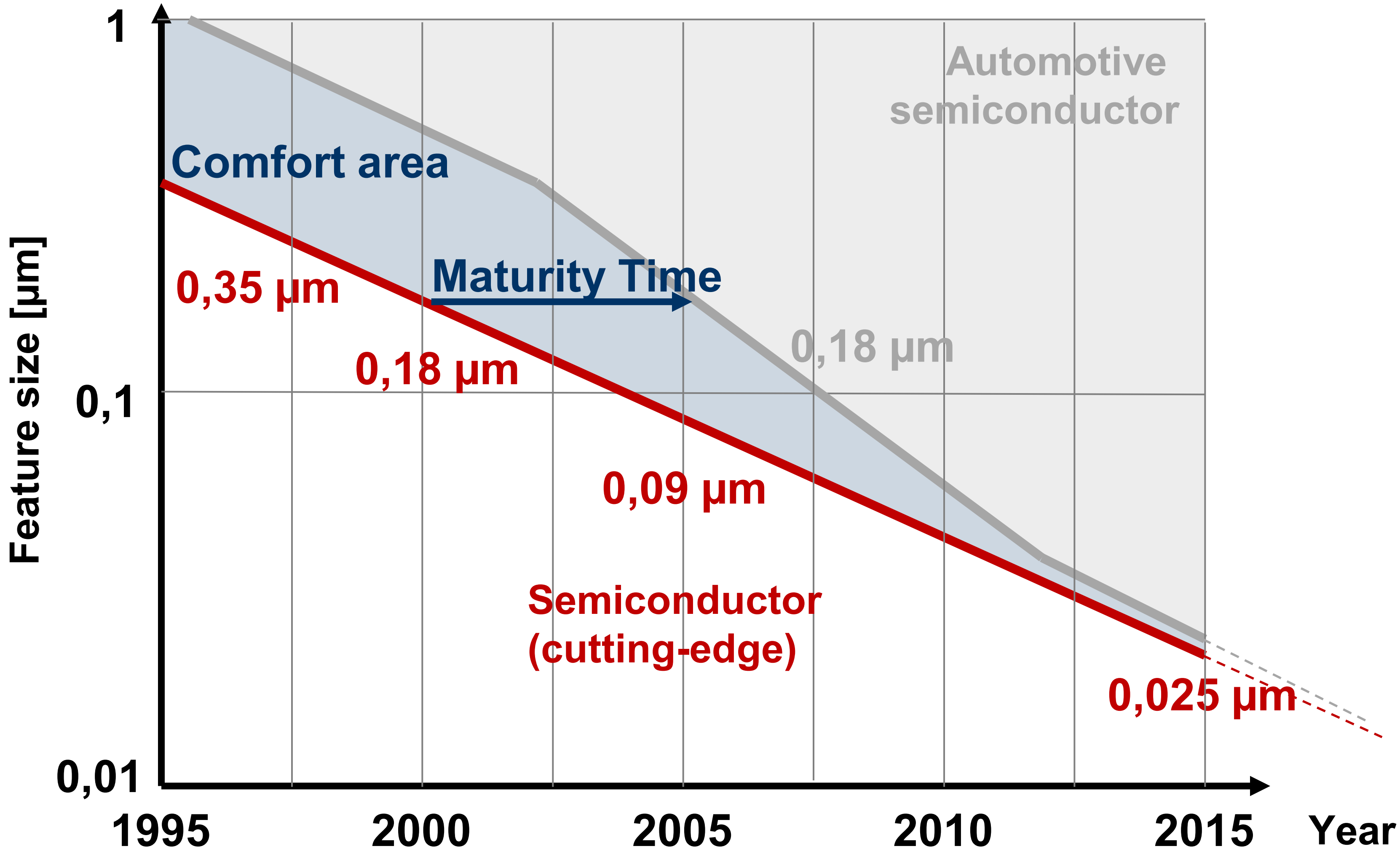
Radio, Navigation System
Electronic Steering
Cleaning Systems
Window and Door Mechatronics
Park Pilot
Info and Entertainment Systems
Airbag
Engine Control Unit
Light Control
Bord Network
Thermo-systems, Climate Control
Side Crash Sensor (Accelaration)
Adaptive Cruise Control Sensor
Weigth Sensing
Side Crash Sensor (Pressure)
Crash Sensing
Park Pilot
Braking System (ABS/ESP)
Transmission Control Unit

Source: Bosch

	A8	A4	A8 new
Number of ECUs	68	65	85
Number of Busses	6	6	7
Software (Mbyte)	60	90	>230

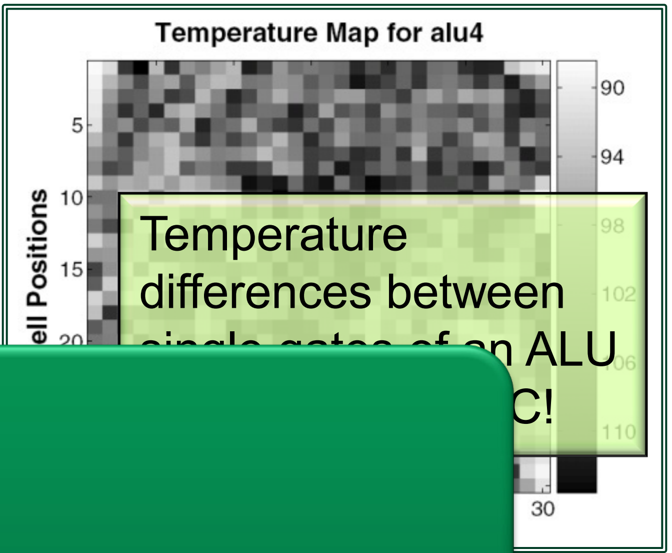
Source: Audi

Automotive Electronics – Reducing Comfort Area



Temperatures: Dominated by Local Power Effects

- "…thermal modeling at finer granularity level i.e., transistor level or logic gate level is required for more accurate estimation of local hot spots."
  Bansal et al., ASP-DAC 2006
- "…hot spots tens of micrometers in

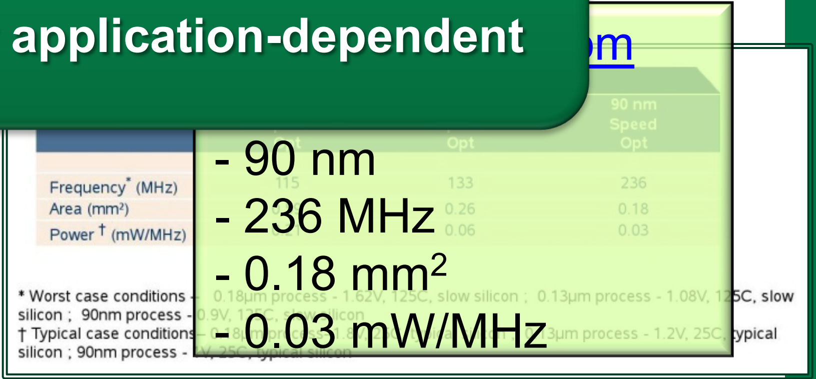


Temperature differences between single gates of an ALU … C!

Conclusion: **Non-Functional Properties**

- Performance
- Energy and Power Consumption
- Temperature and Power Densities

have to be considered and are strongly application-dependent



…m

			90 nm Speed Opt
	Opt	Opt	Opt
Frequency* (MHz)	115	133	236
Area (mm²)	[illegible]	0.26	0.18
Power † (mW/MHz)	0.21	0.06	0.03

* Worst case conditions - 0.18µm process - 1.62V, 125C, slow silicon ; 0.13µm process - 1.08V, 125C, slow silicon ; 90nm process - 0.9V, 125C, slow silicon
† Typical case conditions - 0.18µm process - 1.8V, 25C, typical silicon ; 0.13µm process - 1.2V, 25C, typical silicon ; 90nm process - 1V, 25C, typical silicon

- 90 nm
- 236 MHz
- 0.18 mm²
- 0.03 mW/MHz

- Example: ARM7TDMI
  - Power RUN state: 1.3 mW => Power density: 3.9 W/cm²

Assuming an equal distribution is probably not accurate

Example: Power Simulation at Different Levels of Abstraction

- ARM7TDMI, 90 nm, 236 MHz, 0.18 mm^2

Accuracy insufficient. System model disregards the number of branches in the application.

Functional Representation	Power Model	gcd		ellip	
		Simulation Factor (Simulation Time /Simulated Time)	Average Power - Error (comp. to MS+PC)	Simulation Factor	Average Power - Error
Task Graph, CDG, CFG	Average	<< 1	4.17 uW - 81 %	<< 1	4.17 uW -541 %
Instrum. Source-Code	Instruction-Dependent	0.5	5.8 uW - 31 %	0.1	0.39 uW - 66 %
Static BinaryTransl., Dynamic Binary Transl.	Data-Dependent	75	9.69 uW - 28 %	10	0.70 uW - 7 %
Netlist: ModelSim (MS) + Power Compiler (PC)	PC Internal	> 10^{10}	7.57 uW	> 10^{10}	0.65 uW

Simulation of one second would last several hundred years!!

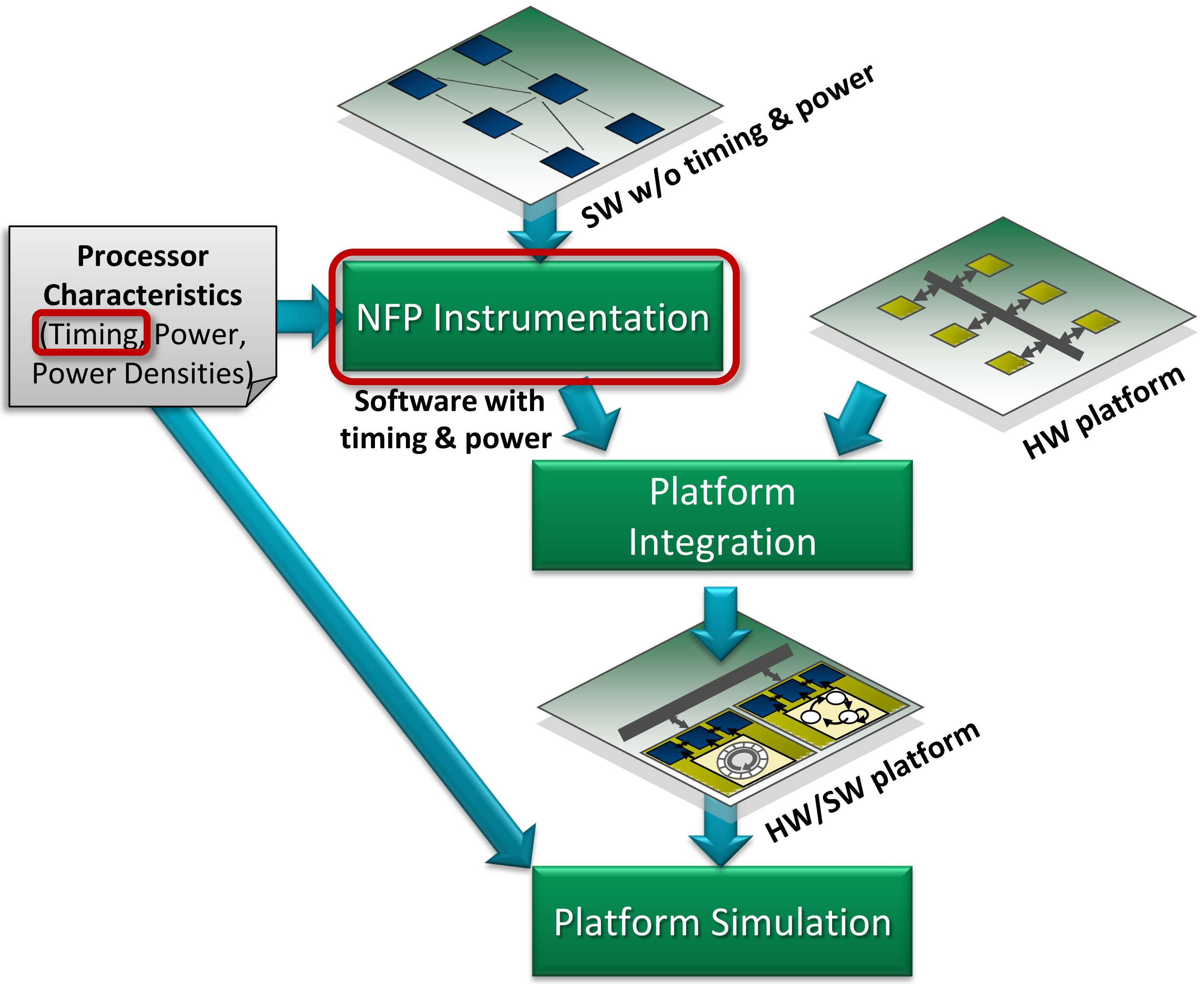
Data-dependent power model leads to a small error. But usually some kind of RTL simulation is necessary to calculate input stimuli of the components.

Simulation speed and appearing error are acceptable. Efficient link with information from binary level could be very promising.

Sander, Bringmann [CODES-ISSS 2009]

SYSTEM-LEVEL SIMULATION OF NON-FUNCTIONAL PROPERTIES BY HOST-COMPILED EXECUTION

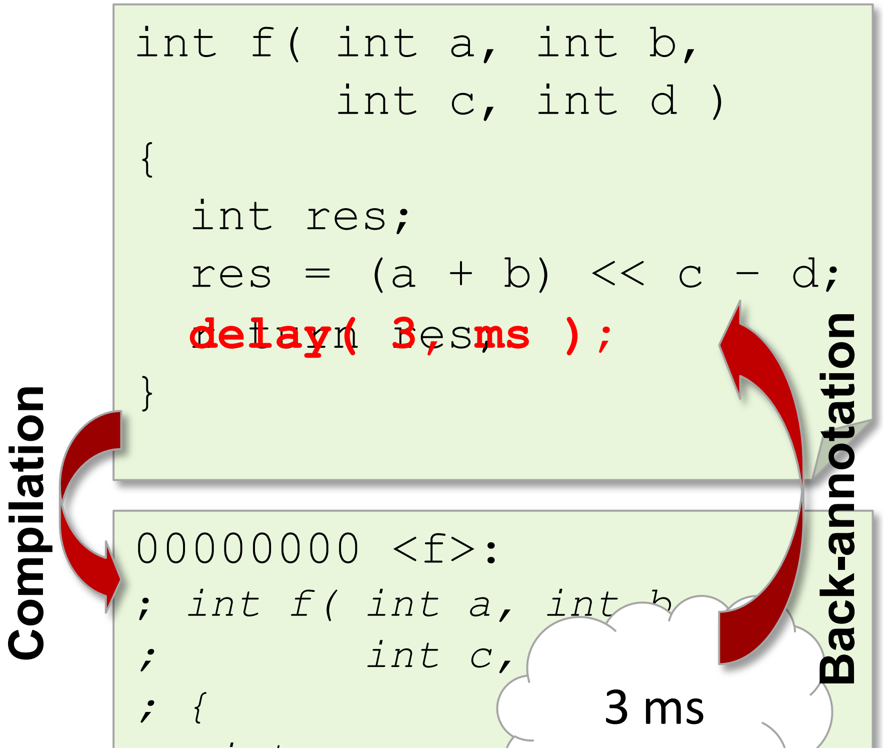
Simulation of Functional and Non-Functional Behavior – Basic Idea



Basic Idea: Source-Level Timing Instrumentation

Proposed Hybrid Approach

- **Compilation into binary code**
- **Static execution time analysis** w.r.t. architectural details
  Back-annotation of analyzed timing information into the source code
- **Simulation** by host-compiled execution



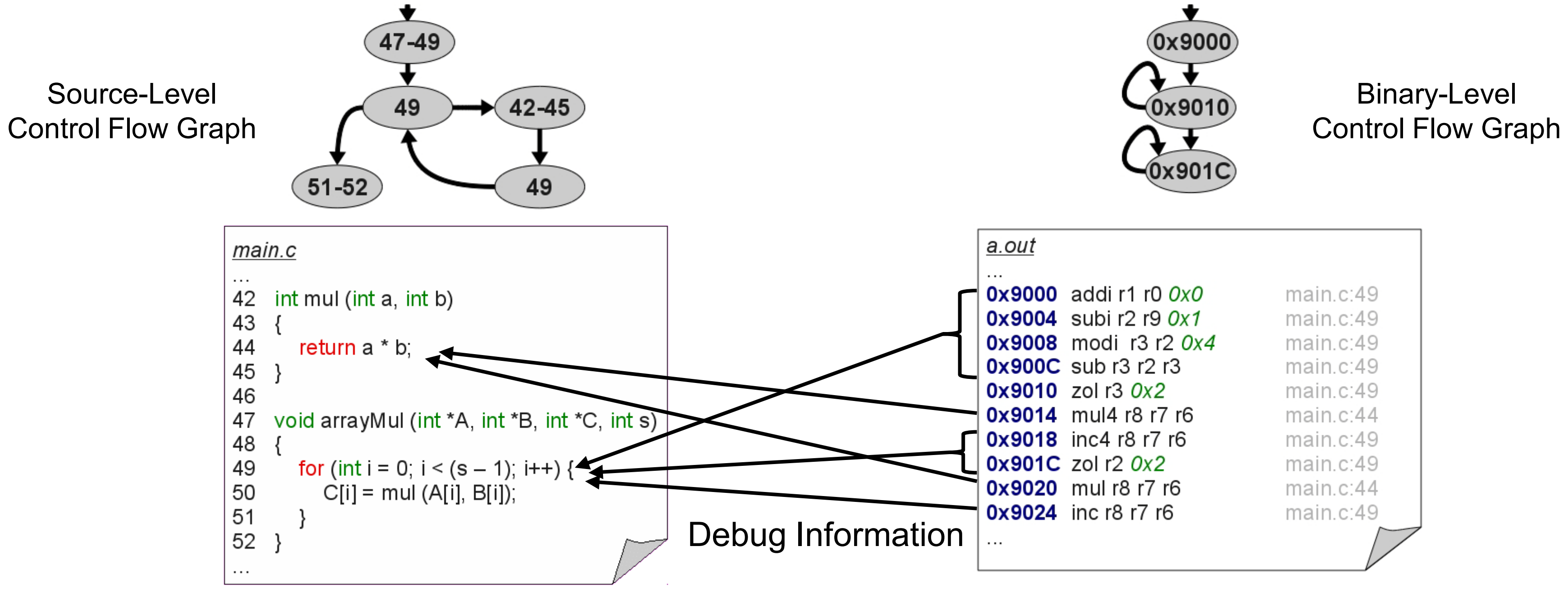
```
int f( int a, int b,
       int c, int d )
{
  int res;
  res = (a + b) << c - d;
  delay( 3, ms );
  return res;
}
```

```
00000000 <f>:
; int f( int a, int b
;         int c,
; {
```

Important

- Requires accurate relation between source code and binary code
- Run-time models for branch prediction and caching have to be incorporated

Compiler Optimizations

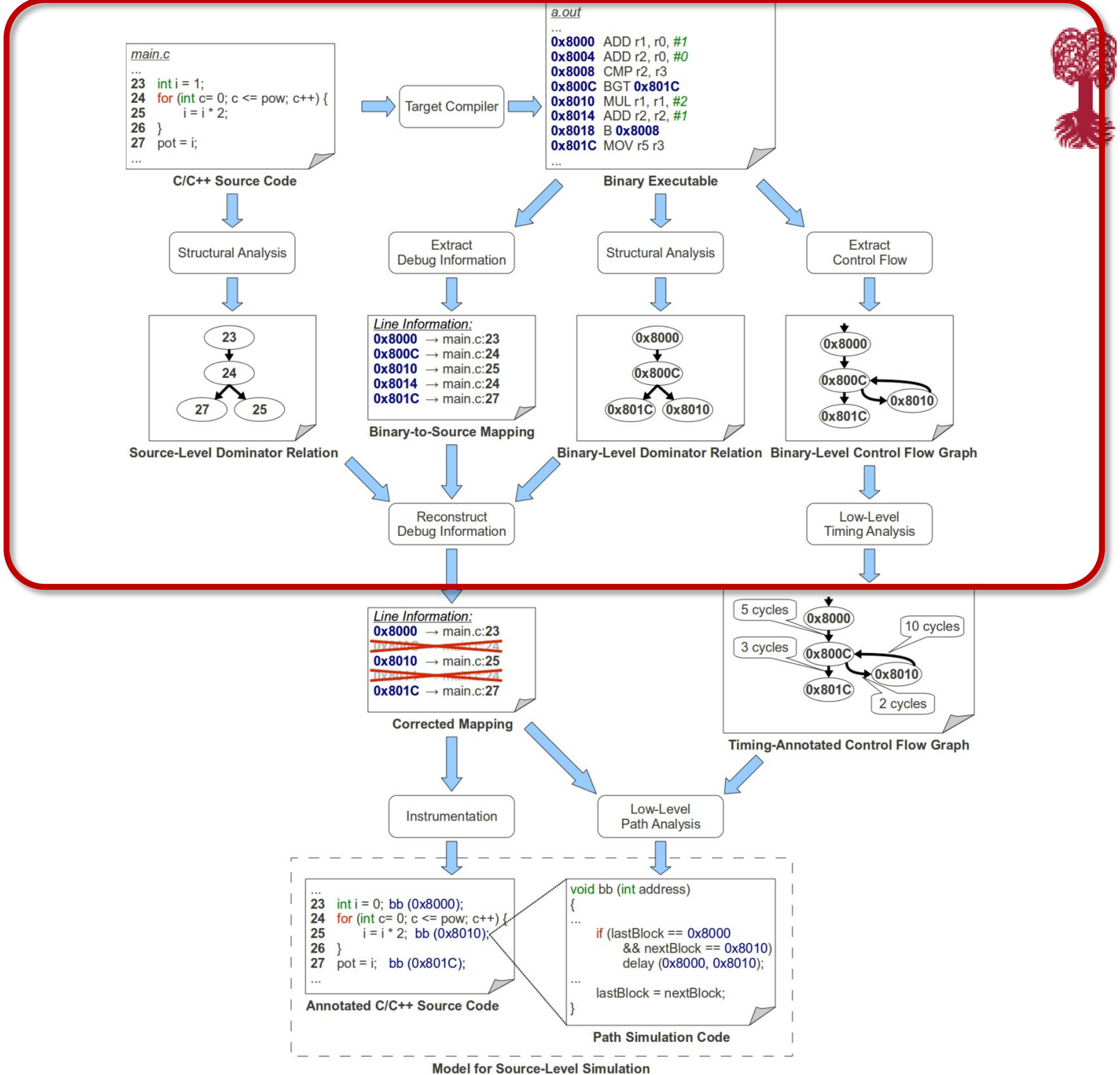

Source-Level Control Flow Graph
47-49
49
42-45
51-52
49
Binary-Level Control Flow Graph
0x9000
0x9010
0x901C
Debug Information

```
main.c
...
42   int mul (int a, int b)
43   {
44       return a * b;
45   }
46
47   void arrayMul (int *A, int *B, int *C, int s)
48   {
49       for (int i = 0; i < (s – 1); i++) {
50           C[i] = mul (A[i], B[i]);
51       }
52   }
...
```

```
a.out
...
0x9000  addi r1 r0 0x0     main.c:49
0x9004  subi r2 r9 0x1     main.c:49
0x9008  modi  r3 r2 0x4    main.c:49
0x900C  sub r3 r2 r3       main.c:49
0x9010  zol r3 0x2         main.c:49
0x9014  mul4 r8 r7 r6      main.c:44
0x9018  inc4 r8 r7 r6      main.c:49
0x901C  zol r2 0x2         main.c:49
0x9020  mul r8 r7 r6       main.c:44
0x9024  inc r8 r7 r6       main.c:49
...
```

- **Structure of source code and binary code can be completely different**
  - **Function Inlining adds basic blocks**
  - **Loop Unrolling modifies execution count of basic blocks**
  - **…**
- **Compilers don't generate accurate debug information for optimized code**
  - ➔ **No 1:1 relation between source-level and binary-level basic blocks**
  - ➔ **Simply annotating delay attributes for source-level timing simulation does not work**

How to match structure of source code and machine instructions?



Target Compiler
C/C++ Source Code
Binary Executable
Structural Analysis
Extract Debug Information
Structural Analysis
Extract Control Flow
Source-Level Dominator Relation
Binary-to-Source Mapping
Binary-Level Dominator Relation
Binary-Level Control Flow Graph
Reconstruct Debug Information
Low-Level Timing Analysis
Corrected Mapping
Timing-Annotated Control Flow Graph
5 cycles
10 cycles
3 cycles
2 cycles
Instrumentation
Low-Level Path Analysis
Annotated C/C++ Source Code
Path Simulation Code
Model for Source-Level Simulation

main.c
```
...
23  int i = 1;
24  for (int c= 0; c <= pow; c++) {
25        i = i * 2;
26  }
27  pot = i;
...
```

a.out
```
...
0x8000  ADD r1, r0, #1
0x8004  ADD r2, r0, #0
0x8008  CMP r2, r3
0x800C  BGT 0x801C
0x8010  MUL r1, r1, #2
0x8014  ADD r2, r2, #1
0x8018  B 0x8008
0x801C  MOV r5 r3
...
```

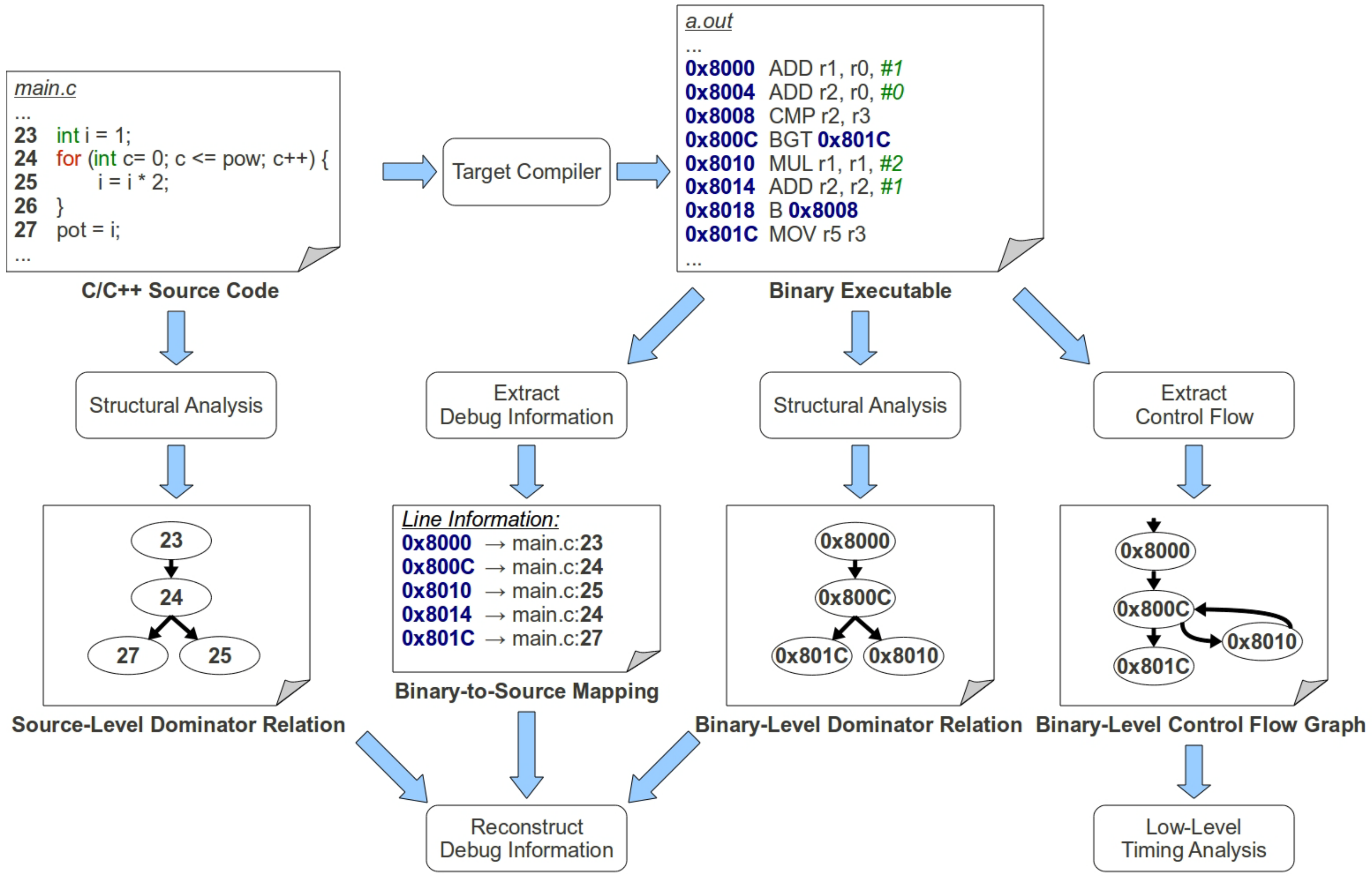
Line Information:
```
0x8000  → main.c:23
0x800C  → main.c:24
0x8010  → main.c:25
0x8014  → main.c:24
0x801C  → main.c:27
```

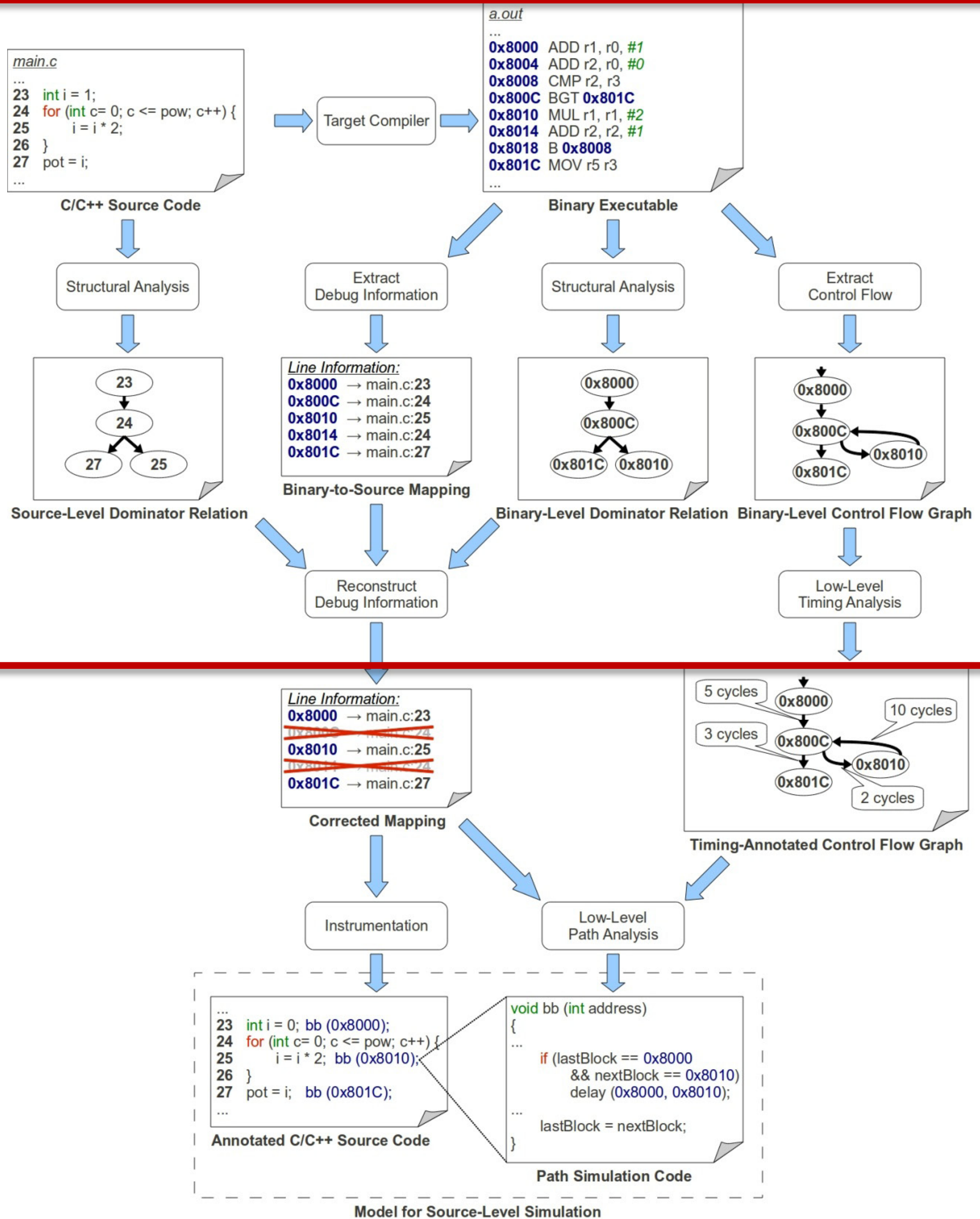
Line Information:
```
0x8000  → main.c:23
~~0x800C  → main.c:24~~
0x8010  → main.c:25
~~0x8014  → main.c:24~~
0x801C  → main.c:27
```

```
...
23  int i = 0;  bb (0x8000);
24  for (int c= 0; c <= pow; c++) {
25        i = i * 2;  bb (0x8010);
26  }
27  pot = i;   bb (0x801C);
...
```

```
void bb (int address)
{
...
    if (lastBlock == 0x8000
          && nextBlock == 0x8010)
          delay (0x8000, 0x8010);
...
    lastBlock = nextBlock;
}
```

Structural Analysis and Code Matching





main.c

```
...
23  int i = 1;
24  for (int c= 0; c <= pow; c++) {
25        i = i * 2;
26  }
27  pot = i;
...
```

a.out

```
...
0x8000  ADD r1, r0, #1
0x8004  ADD r2, r0, #0
0x8008  CMP r2, r3
0x800C  BGT 0x801C
0x8010  MUL r1, r1, #2
0x8014  ADD r2, r2, #1
0x8018  B 0x8008
0x801C  MOV r5 r3
...
```

Line Information:

```
0x8000  → main.c:23
0x800C  → main.c:24
0x8010  → main.c:25
0x8014  → main.c:24
0x801C  → main.c:27
```

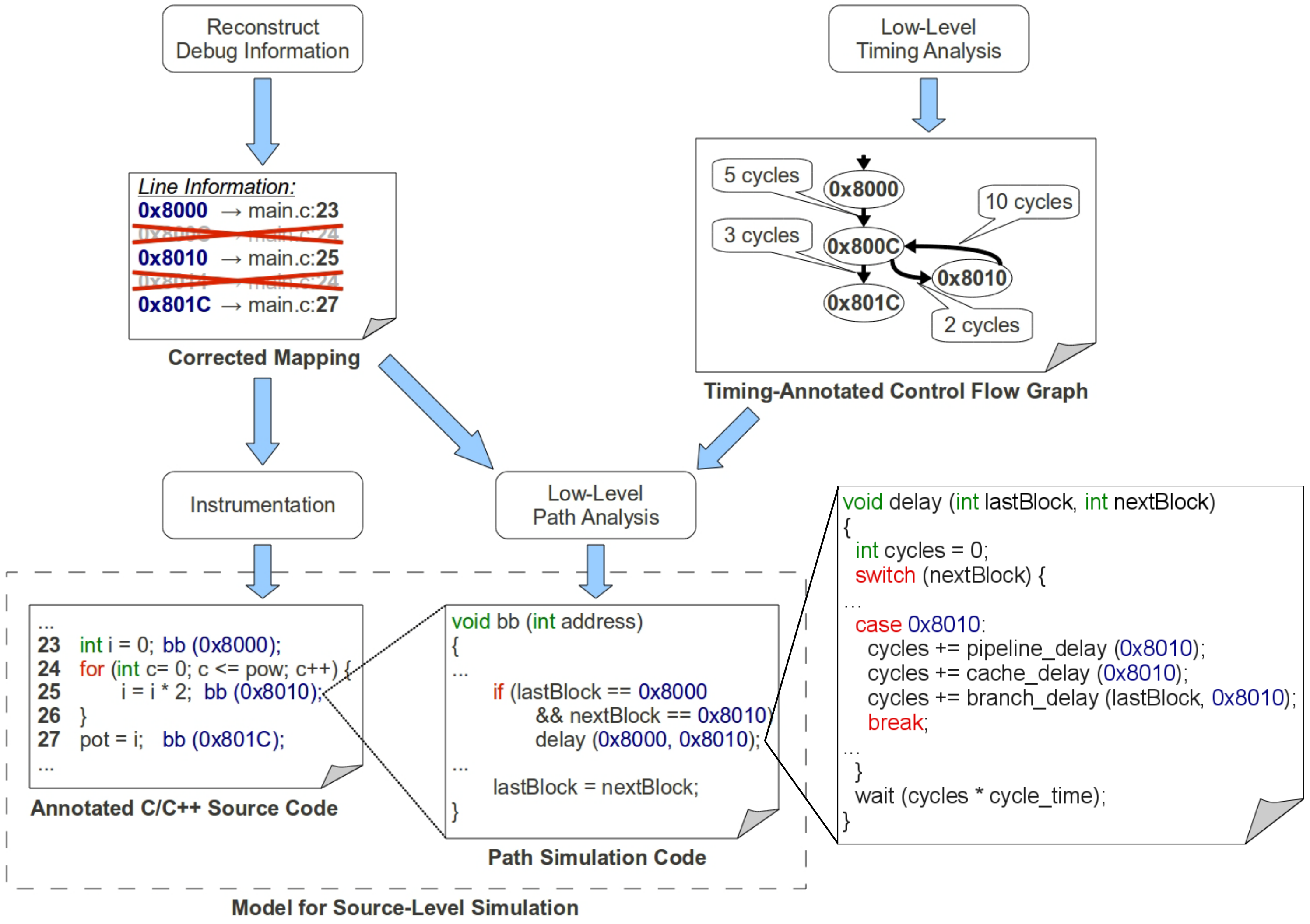
Line Information:

```
0x8000  → main.c:23
~~0x800C  → main.c:24~~
0x8010  → main.c:25
~~0x8014  → main.c:24~~
0x801C  → main.c:27
```

```
...
23  int i = 0;  bb (0x8000);
24  for (int c= 0; c <= pow; c++) {
25        i = i * 2;  bb (0x8010);
26  }
27  pot = i;   bb (0x801C);
...
```

```
void bb (int address)
{
...
    if (lastBlock == 0x8000
        && nextBlock == 0x8010)
        delay (0x8000, 0x8010);
...
    lastBlock = nextBlock;
}
```

Annotation and Path Simulation Code Generation


Reconstruct Debug Information
Low-Level Timing Analysis
5 cycles
0x8000
10 cycles
3 cycles
0x800C
0x8010
0x801C
2 cycles
Timing-Annotated Control Flow Graph
Instrumentation
Low-Level Path Analysis
Model for Source-Level Simulation

Line Information:
0x8000 → main.c:**23**
~~0x8008 → main.c:24~~
0x8010 → main.c:**25**
~~0x8014 → main.c:24~~
0x801C → main.c:**27**

Corrected Mapping

```
...
23   int i = 0;  bb (0x8000);
24   for (int c= 0; c <= pow; c++) {
25          i = i * 2;  bb (0x8010);
26   }
27   pot = i;   bb (0x801C);
...
```

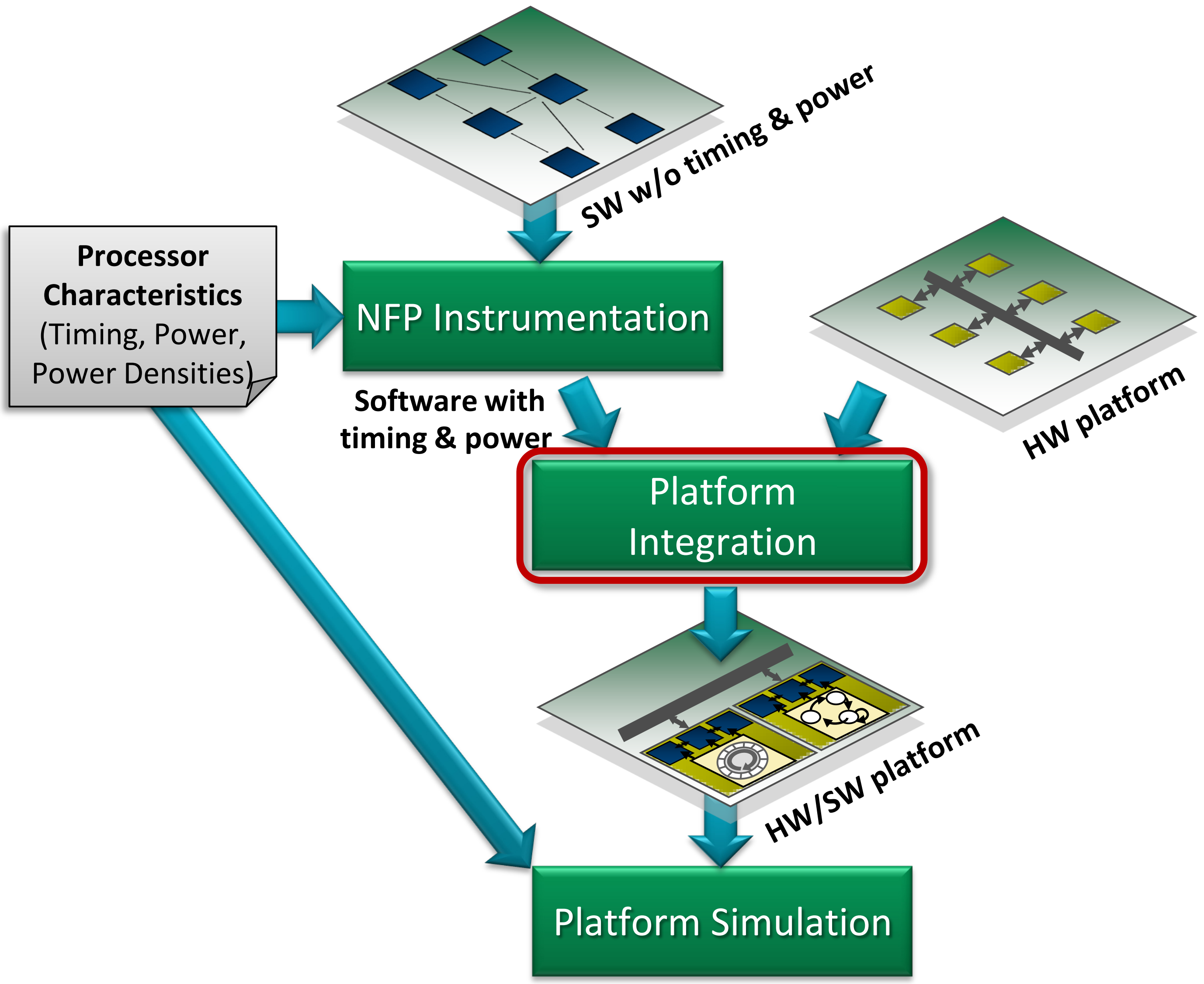
Annotated C/C++ Source Code

```
void bb (int address)
{
...
     if (lastBlock == 0x8000
          && nextBlock == 0x8010)
          delay (0x8000, 0x8010);
...
     lastBlock = nextBlock;
}
```

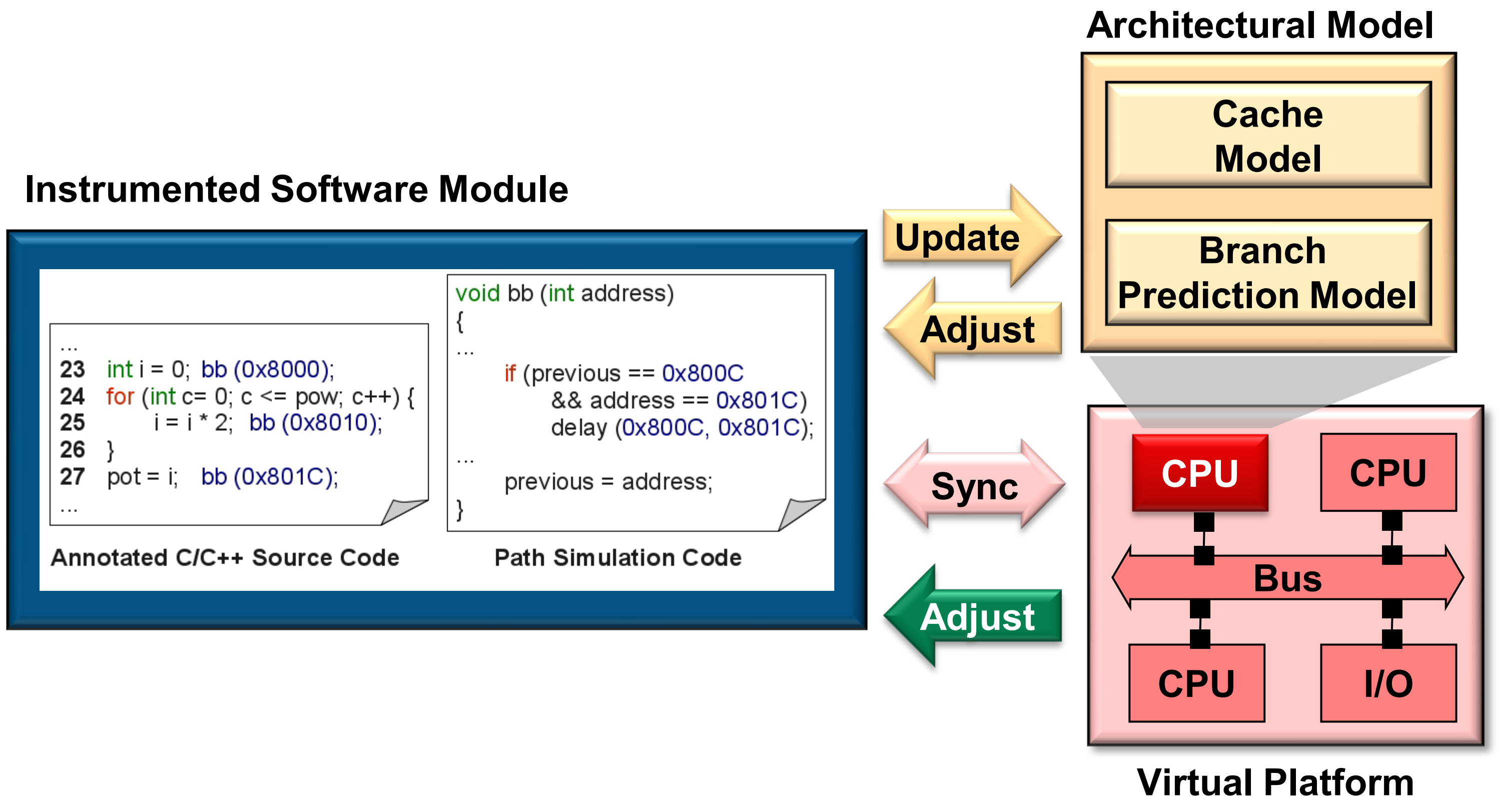
Path Simulation Code

```
void delay (int lastBlock, int nextBlock)
{
  int cycles = 0;
  switch (nextBlock) {
...
  case 0x8010:
    cycles += pipeline_delay (0x8010);
    cycles += cache_delay (0x8010);
    cycles += branch_delay (lastBlock, 0x8010);
    break;
...
  }
  wait (cycles * cycle_time);
}
```

Simulation of Functional and Non-Functional Behavior – Basic Idea



Platform Model Integration



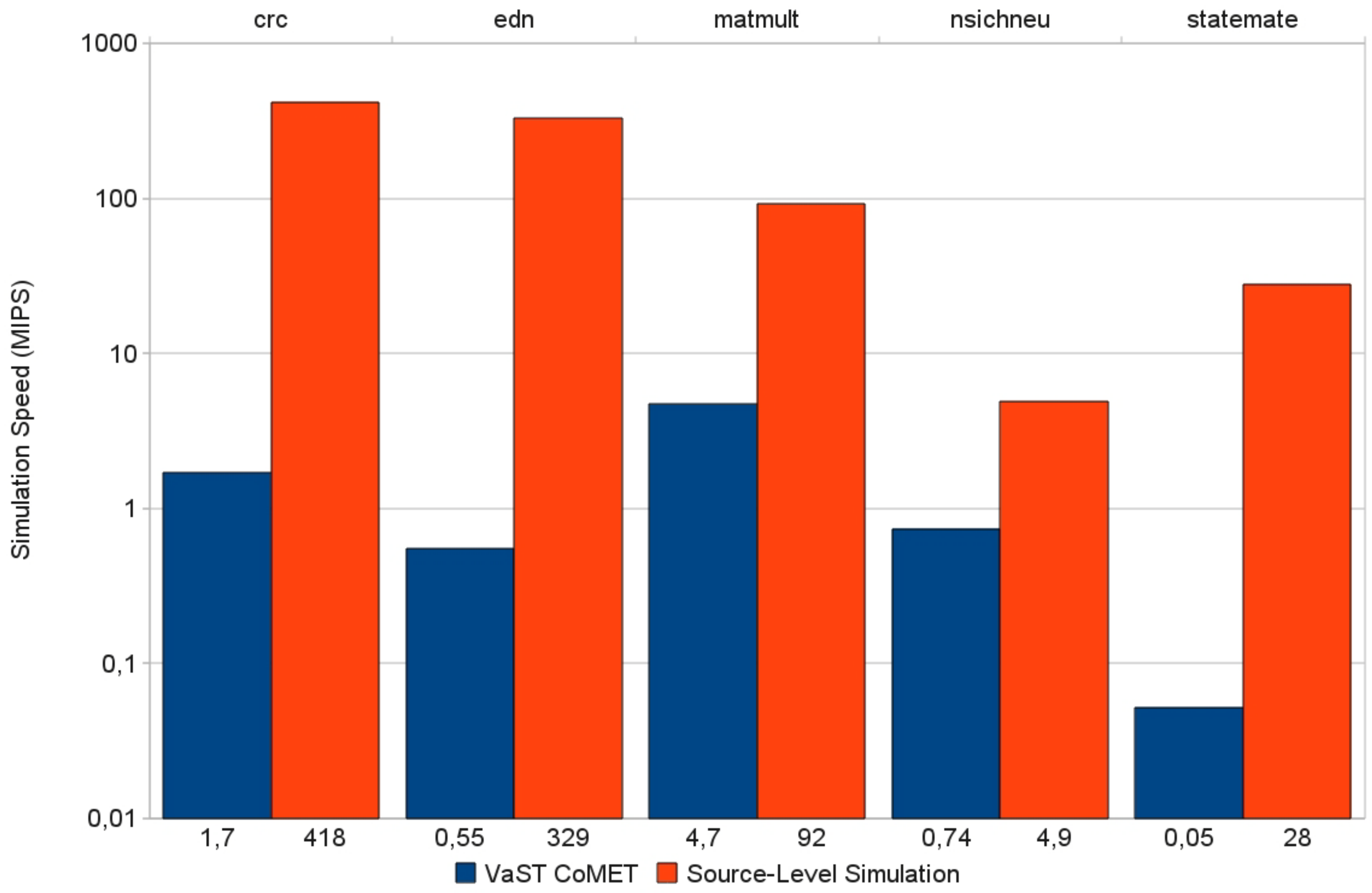
```
...
int i = 0;  bb (0x8000);
for (int c= 0; c <= pow; c++) {
        i = i * 2;  bb (0x8010);
}
pot = i;   bb (0x801C);
...
```

```
void bb (int address)
{
...
    if (previous == 0x800C
        && address == 0x801C)
        delay (0x800C, 0x801C);
...
    previous = address;
}
```

TLM-2.0 Loosely Timed Simulation

S. Stattelmann, O. Bringmann et al. [DATE 2011]

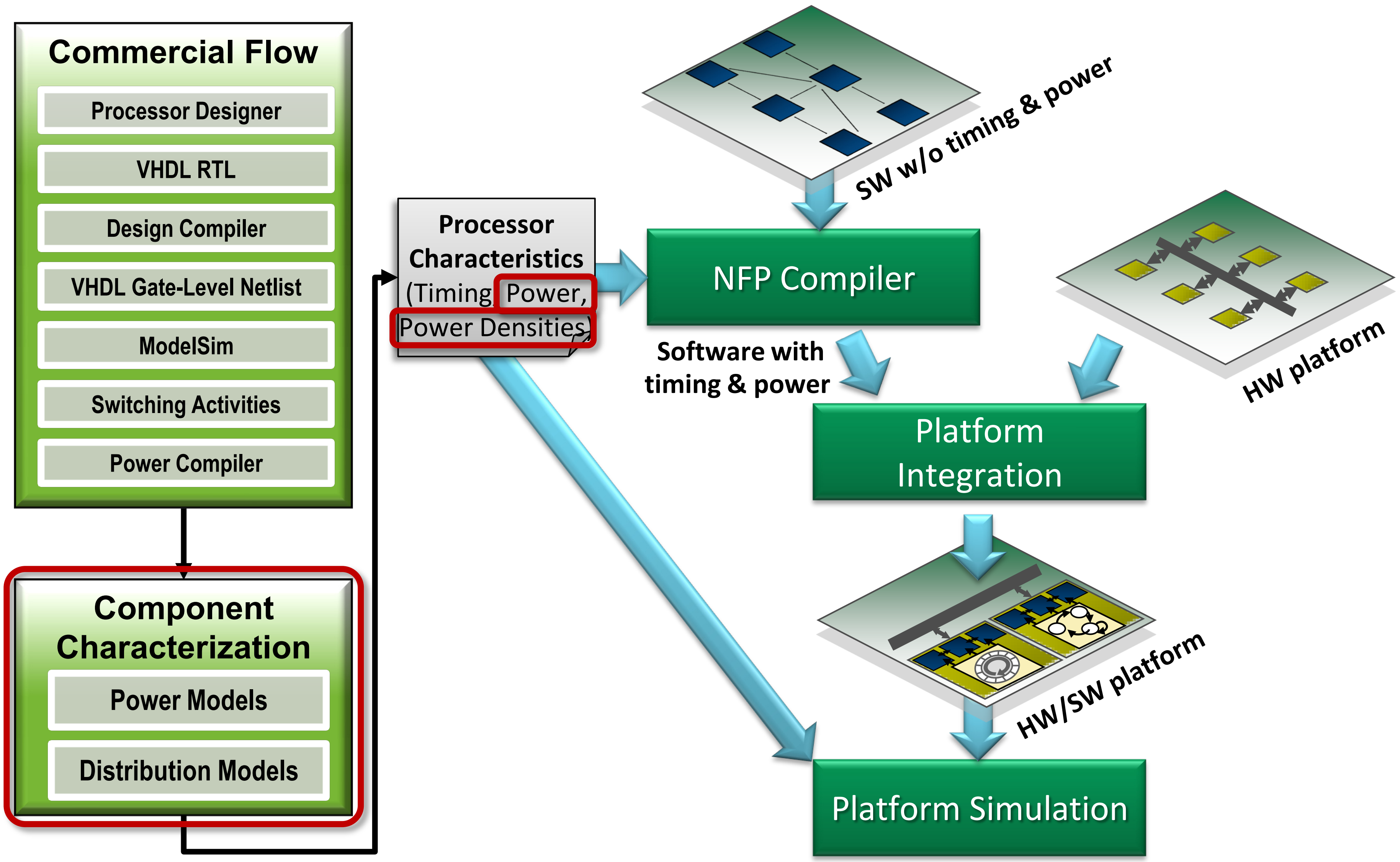
Experimental Results



- Experiments conducted for an ARM processor
- Improved simulation performance compared to high-performance ISS based on just-in-time compilation

APPLICATION-DEPENDENT POWER AND TEMPERATURE SIMULATION FRAMEWORK

Power and Power Density Characterization



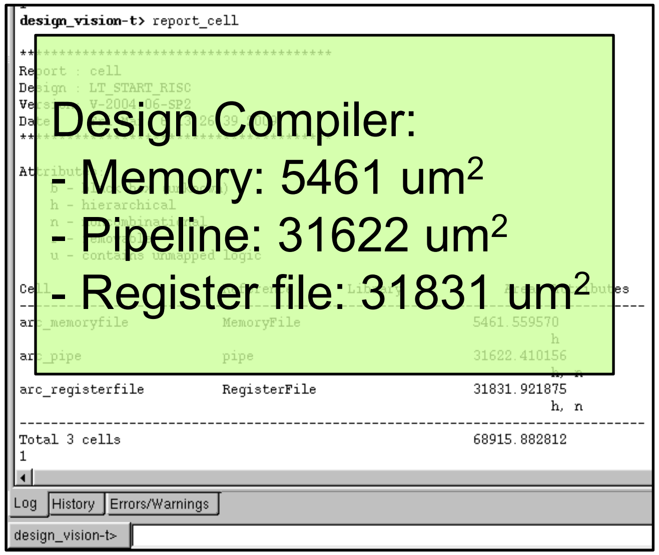
Characterization: Power Distribution Models

Starting Point:
- Commercial Tool Chain
- ARM-like processor design

➔ Used to create accurate power measurements for processor components

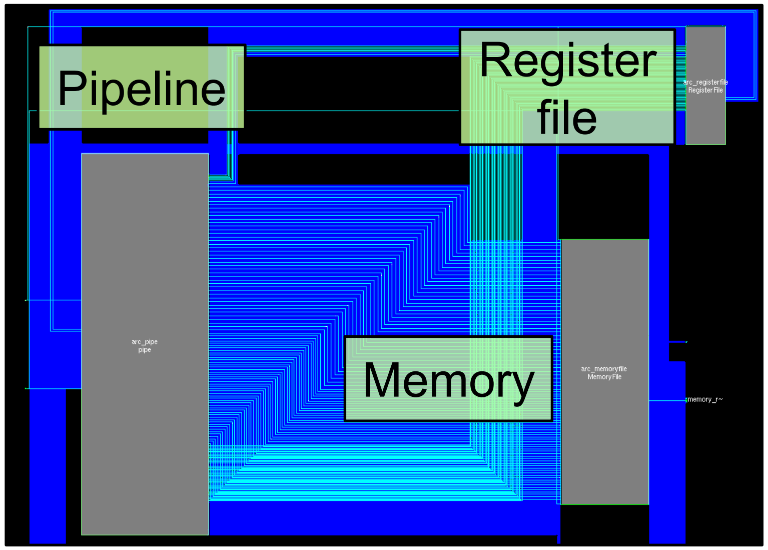
Commercial Flow

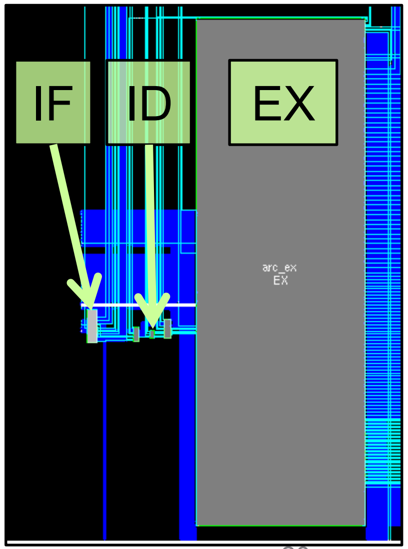
- Processor Designer
- VHDL RTL
- Design Compiler
- VHDL Gate-Level Netlist
- ModelSim
- Switching Activities
- Power Compiler



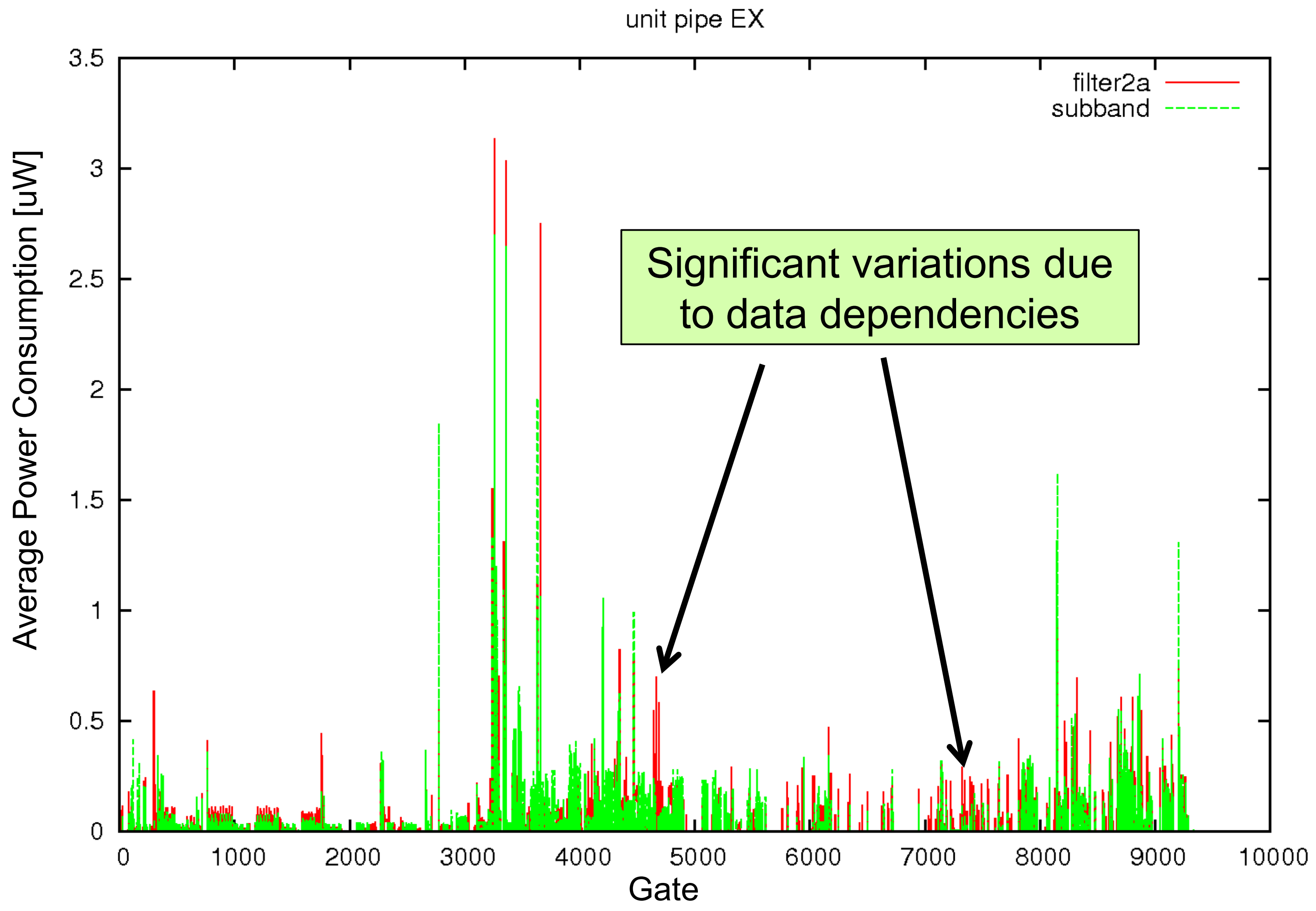
```
design_vision-t> report_cell

****************************************
Report : cell
Design : LT_START_RISC
Version: V-2004.06-SP2
Date   : [illegible] 26:39 2009
****************************************

Attributes:
    b - black box (unknown)
    h - hierarchical
    n - noncombinational
    r - removable
    u - contains unmapped logic

Cell                    Reference       Library         Area  Attributes
--------------------------------------------------------------------------
arc_memoryfile          MemoryFile                 5461.559570
                                                              h
arc_pipe                pipe                      31622.410156
                                                              h, n
arc_registerfile        RegisterFile              31831.921875
                                                              h, n
--------------------------------------------------------------------------
Total 3 cells                                     68915.882812
1
```

Design Compiler:
- Memory: 5461 um^2
- Pipeline: 31622 um^2
- Register file: 31831 um^2



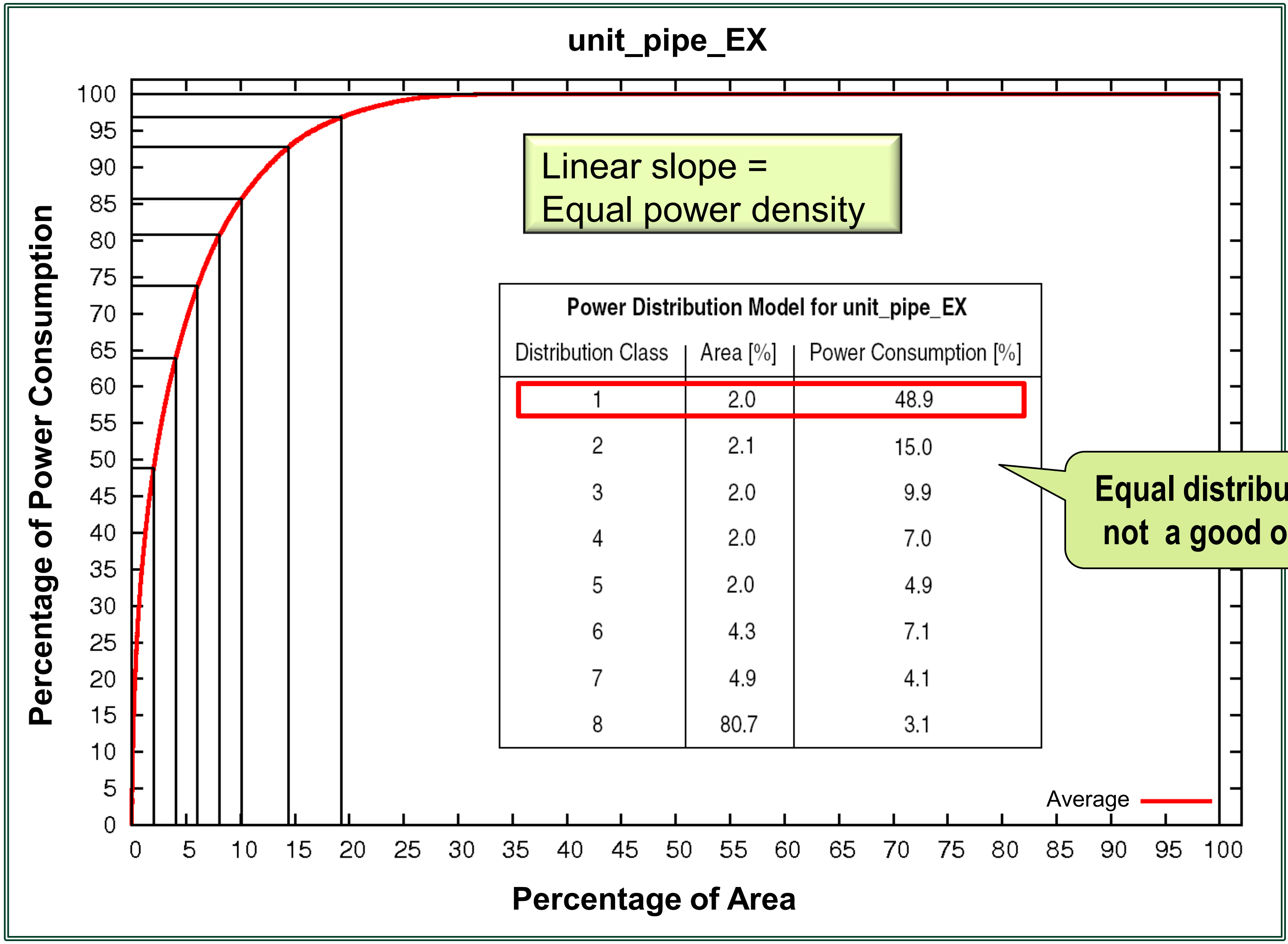


Gate Average Power Consumption



Significant variations due to data dependencies

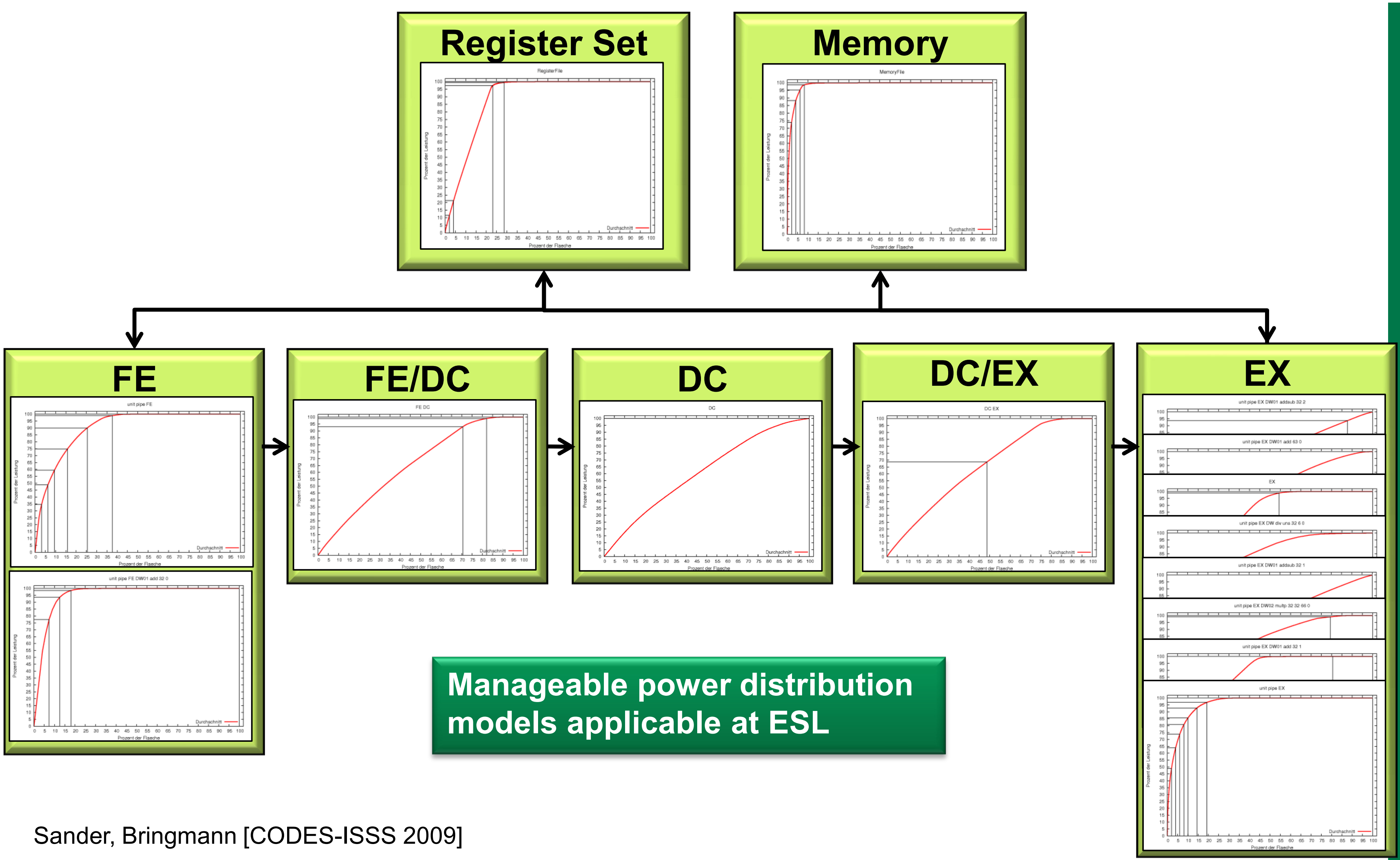
Characterization: Power Distribution Models



Power Distribution Model for unit_pipe_EX

Distribution Class	Area [%]	Power Consumption [%]
1	2.0	48.9
2	2.1	15.0
3	2.0	9.9
4	2.0	7.0
5	2.0	4.9
6	4.3	7.1
7	4.9	4.1
8	80.7	3.1

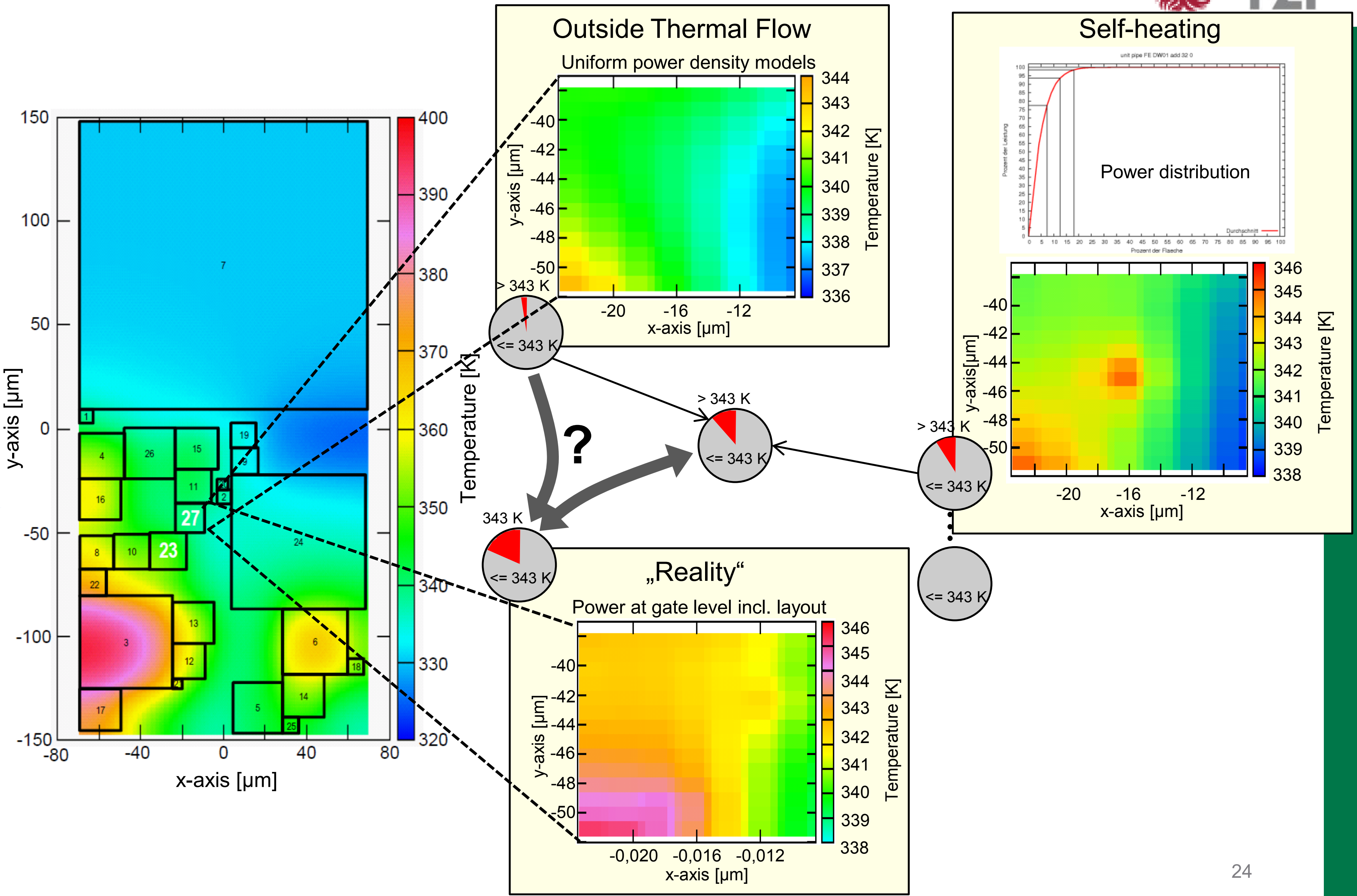
Characterization: Power Distribution Models



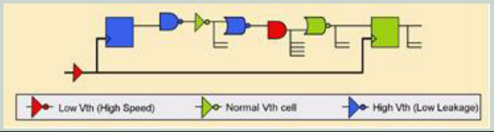
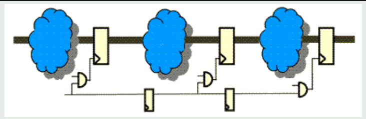
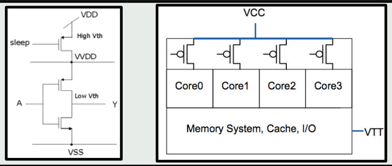
Manageable power distribution models applicable at ESL

Sander, Bringmann [CODES-ISSS 2009]

Power Density Models for Temperature Analysis



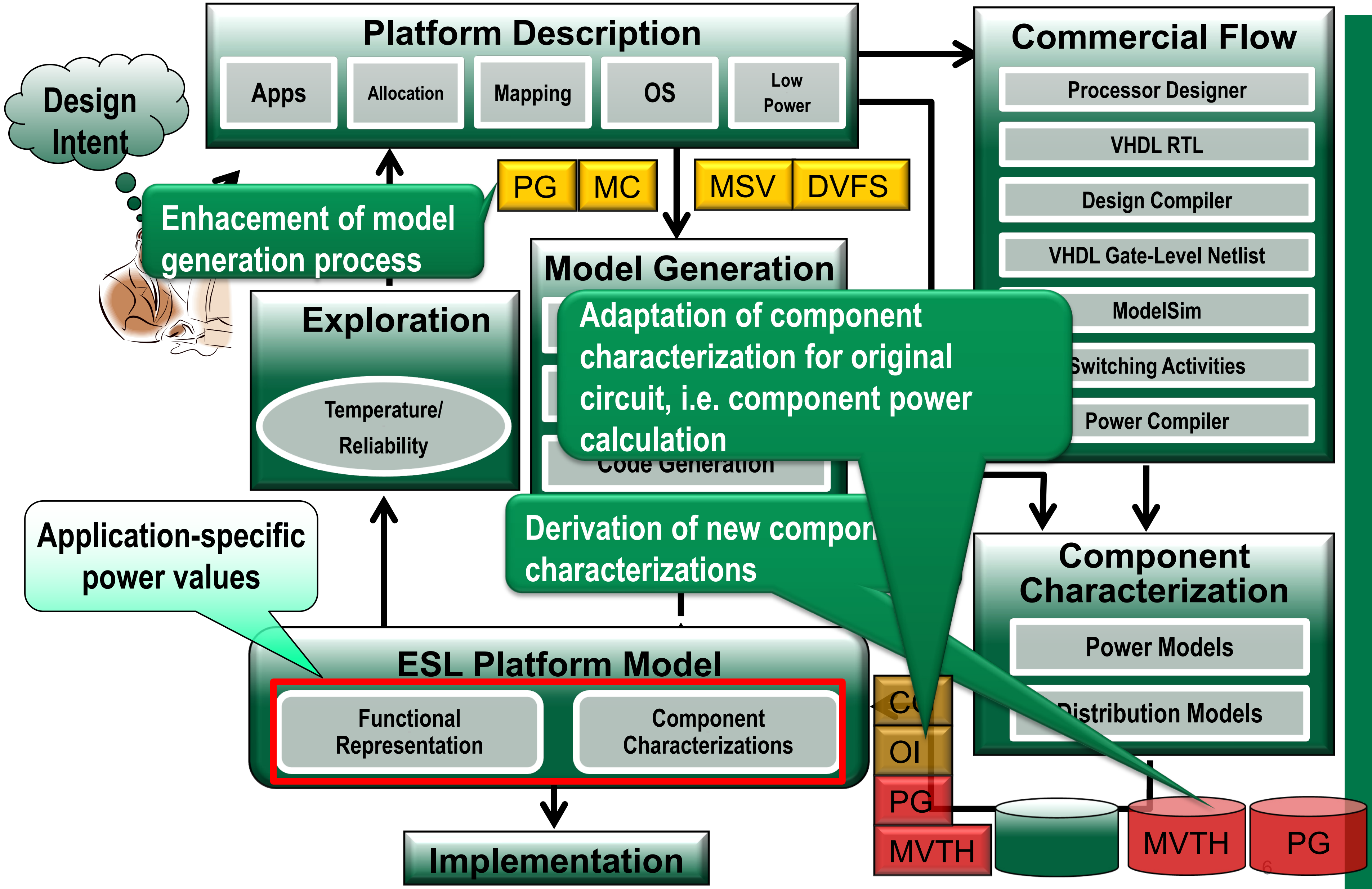
Low Power Design: What can be done…

Low Power Technique	Description	
Multi-Voltage Threshold (MVTH) MVTH	Individual logic gates use transistors with low threshold voltages	 Low Vth (High Speed) Normal Vth cell High Vth (Low Leakage)
Clock Gating (CG) CG	Disable registers	
Operand Isolation (OI) OI	Datapath blocks are prevented from swichting (keep in constant)	
Multiple Supply Voltages (MSV) MSV	Blocks operate with dif supply voltages	
Dynamic Voltage and Frequency Scaling (DVFS) DVFS	Adjust voltage and fre the fly depending on th	
Power Gating (PG) PG PG	Torn of supply voltage when not in use	 VDD sleep High Vth VVDD Low Vth A Y VSS VCC Core0 Core1 Core2 Core3 Memory System, Cache, I/O VTT
Multi-Core (MC) MC	Distribute functionality to more than a single processor	Functionality on 2 processors => f/2, 0,7*V => power cut in half

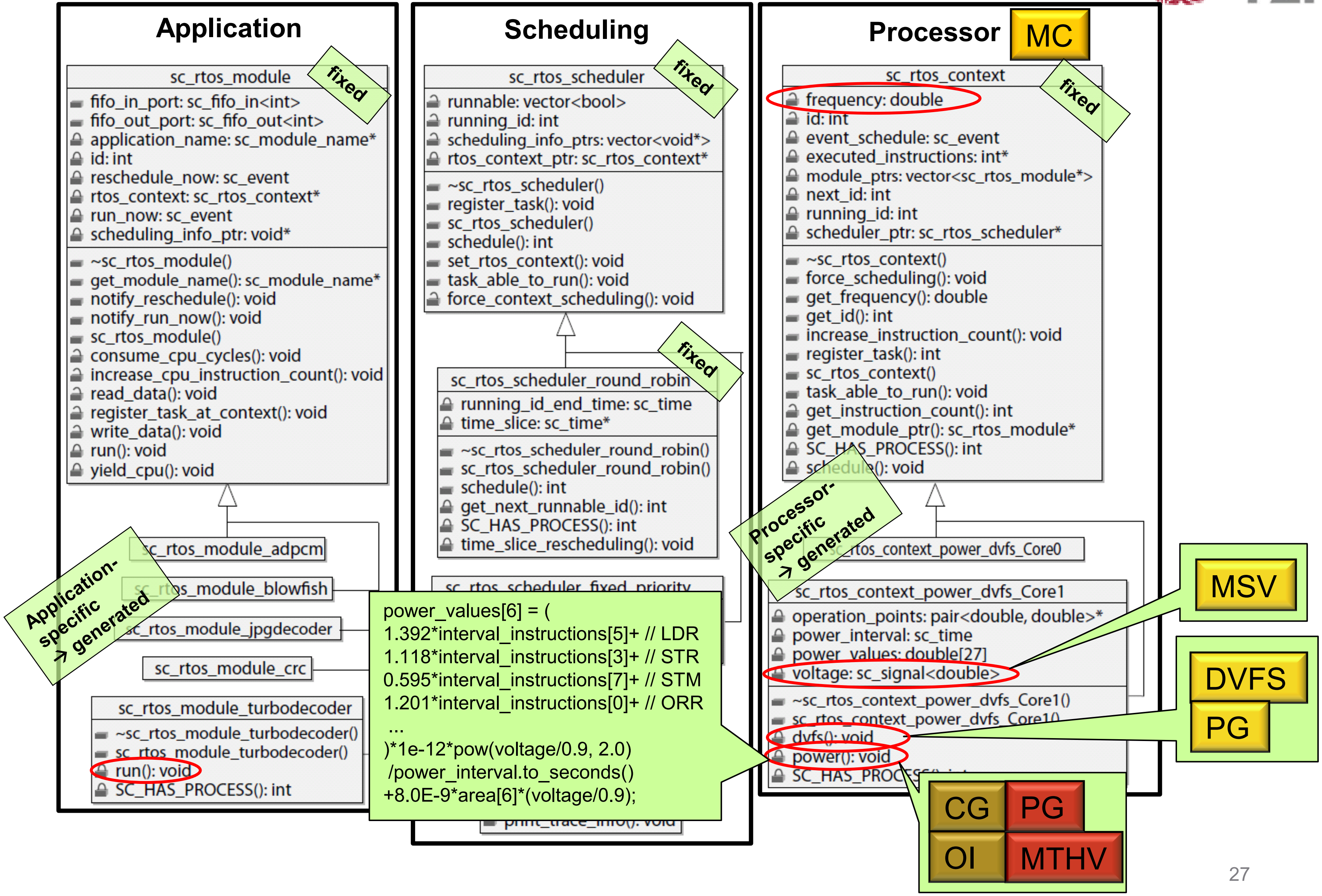
For system model integration 3 different classes of low power techniques can be identified:

🟥	**„Original circuit “ is changed**
🟫	**Additional hardware is used**
🟨	**Architectural measures**

Incorporation of Low Power Techniques



Parameterization of the System Model

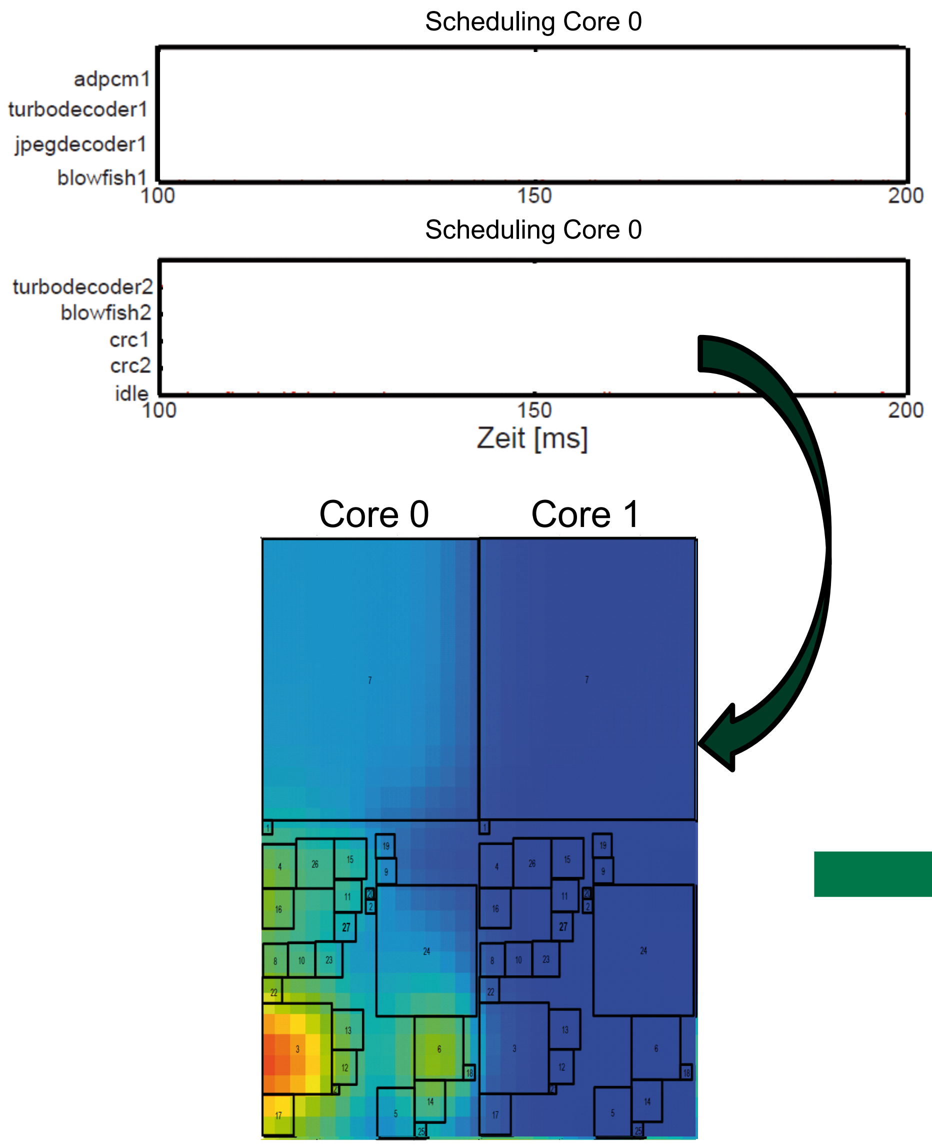


Application
sc_rtos_module
fixed
fifo_in_port: sc_fifo_in<int>
fifo_out_port: sc_fifo_out<int>
application_name: sc_module_name*
id: int
reschedule_now: sc_event
rtos_context: sc_rtos_context*
run_now: sc_event
scheduling_info_ptr: void*
~sc_rtos_module()
get_module_name(): sc_module_name*
notify_reschedule(): void
notify_run_now(): void
sc_rtos_module()
consume_cpu_cycles(): void
increase_cpu_instruction_count(): void
read_data(): void
register_task_at_context(): void
write_data(): void
run(): void
yield_cpu(): void
sc_rtos_module_adpcm
sc_rtos_module_blowfish
sc_rtos_module_jpgdecoder
sc_rtos_module_crc
sc_rtos_module_turbodecoder
~sc_rtos_module_turbodecoder()
sc_rtos_module_turbodecoder()
run(): void
SC_HAS_PROCESS(): int
Application-specific → generated
Scheduling
sc_rtos_scheduler
fixed
runnable: vector<bool>
running_id: int
scheduling_info_ptrs: vector<void*>
rtos_context_ptr: sc_rtos_context*
~sc_rtos_scheduler()
register_task(): void
sc_rtos_scheduler()
schedule(): int
set_rtos_context(): void
task_able_to_run(): void
force_context_scheduling(): void
sc_rtos_scheduler_round_robin
fixed
running_id_end_time: sc_time
time_slice: sc_time*
~sc_rtos_scheduler_round_robin()
sc_rtos_scheduler_round_robin()
schedule(): int
get_next_runnable_id(): int
SC_HAS_PROCESS(): int
time_slice_rescheduling(): void
sc_rtos_scheduler_fixed_priority
print_trace_info(): void
Processor
MC
sc_rtos_context
fixed
frequency: double
id: int
event_schedule: sc_event
executed_instructions: int*
module_ptrs: vector<sc_rtos_module*>
next_id: int
running_id: int
scheduler_ptr: sc_rtos_scheduler*
~sc_rtos_context()
force_scheduling(): void
get_frequency(): double
get_id(): int
increase_instruction_count(): void
register_task(): int
sc_rtos_context()
task_able_to_run(): void
get_instruction_count(): int
get_module_ptr(): sc_rtos_module*
SC_HAS_PROCESS(): int
schedule(): void
Processor-specific → generated
sc_rtos_context_power_dvfs_Core0
sc_rtos_context_power_dvfs_Core1
operation_points: pair<double, double>*
power_interval: sc_time
power_values: double[27]
voltage: sc_signal<double>
~sc_rtos_context_power_dvfs_Core1()
sc_rtos_context_power_dvfs_Core1()
dvfs(): void
power(): void
SC_HAS_PROCESS(): int
MSV
DVFS
PG
CG
PG
OI
MTHV

```
power_values[6] = (
1.392*interval_instructions[5]+ // LDR
1.118*interval_instructions[3]+ // STR
0.595*interval_instructions[7]+ // STM
1.201*interval_instructions[0]+ // ORR
 ...
)*1e-12*pow(voltage/0.9, 2.0)
 /power_interval.to_seconds()
+8.0E-9*area[6]*(voltage/0.9);
```

AREAS OF APPLICATION

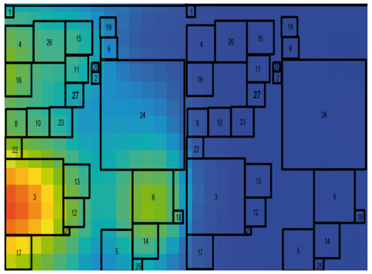
Thermal Simulation

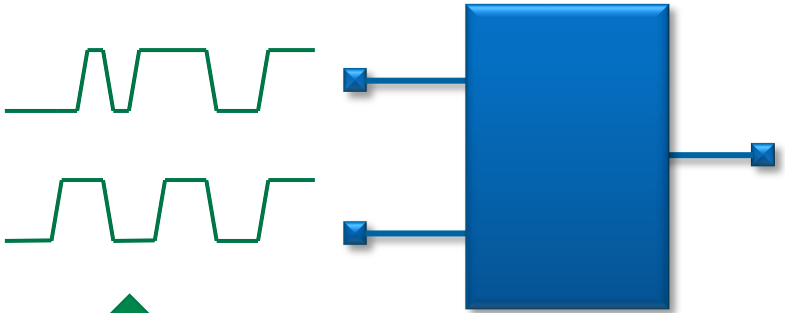


Trigger temperature-dependent robustness and reliability analyses

Triggering of RT / Gate Level Aging Models

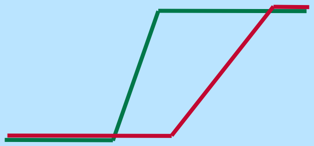
Operating Conditions



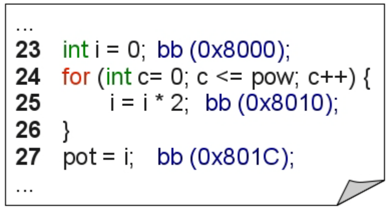


Modeled Effects

- NBTI
- HCI

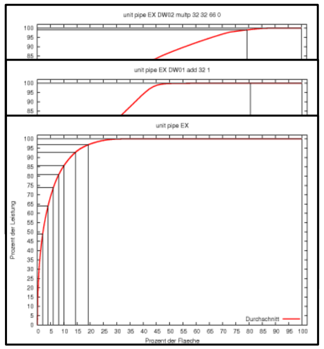


Workload



```
...
int i = 0;  bb (0x8000);
for (int c= 0; c <= pow; c++) {
      i = i * 2;  bb (0x8010);
}
pot = i;    bb (0x801C);
...
```

Annotated C/C++ Source Code

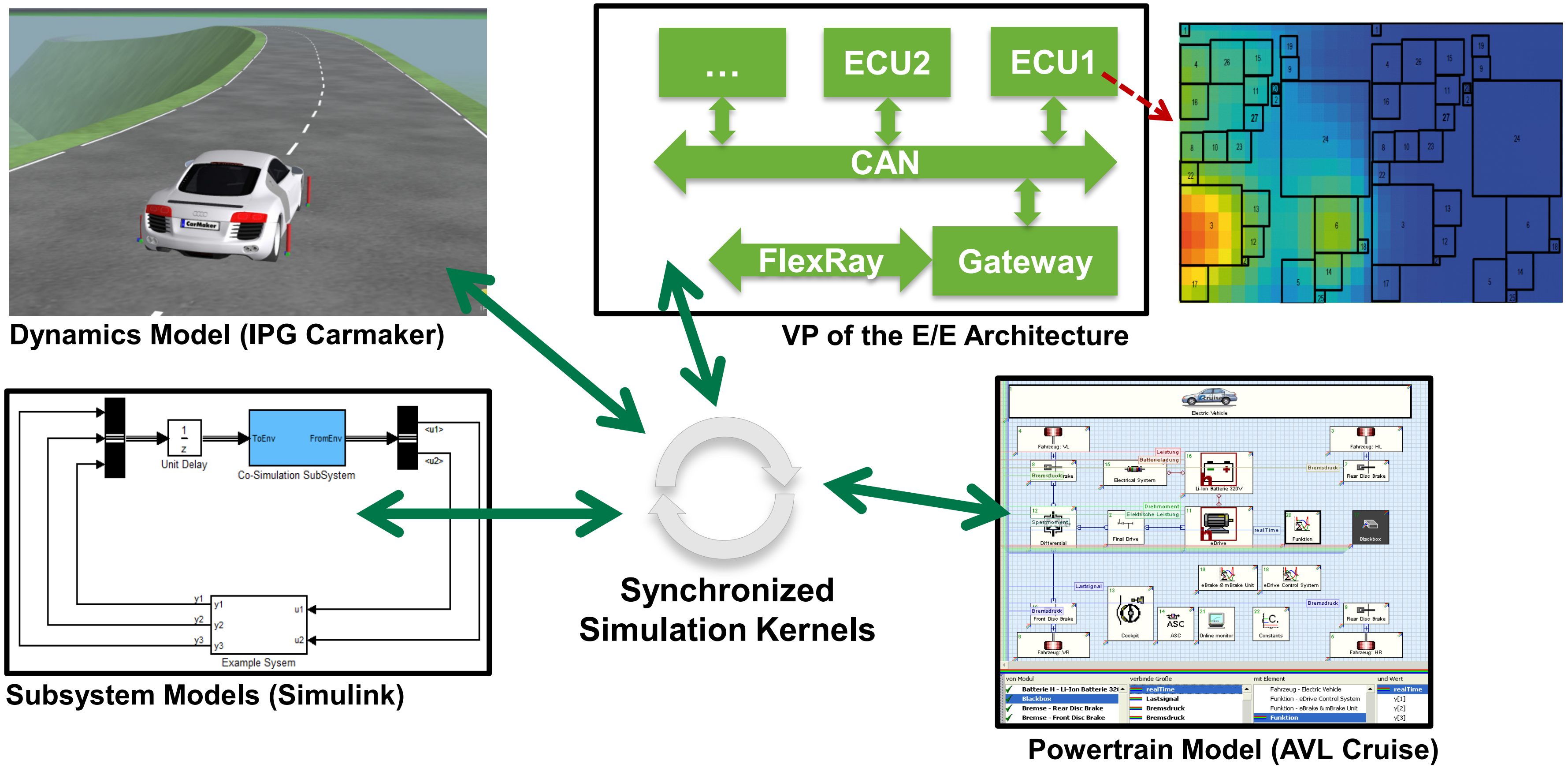


Aged Component Model

- gate delay
- output slope

→ aged critical path

Platform for Fast Holistic Vehicle Analysis



…
ECU2
ECU1
CAN
FlexRay
Gateway
VP of the E/E Architecture
Dynamics Model (IPG Carmaker)
Unit Delay
ToEnv
FromEnv
Co-Simulation SubSystem
<u1>
<u2>
y1
y2
y3
u1
u2
Example Sysem
Subsystem Models (Simulink)
Synchronized Simulation Kernels
Electric Vehicle
Powertrain Model (AVL Cruise)

- Fast co-simulation to cover application and environment context
- Virtual prototype (VP) reflects timing/power/temperature behavior
- Foundation for application/environment-driven robustness analysis

Conclusion

- Non-functional properties important issue in embedded SW design
- Real-time simulation possible using source code instrumentation
  - Timing and power characteristics are annotated
  - Impact of data-dependencies and access to shared resources are solved by dynamic execution
  - Highly scalable in terms of the number of processors/processor cores
- Applicable to other non-functional properties like determination of thermal distributions and application-dependent reliability analysis
- Foundation for efficient simulation of heterogeneous HW/SW systems including sensors and actors

Thank you very much for your attention!

Questions?

Oliver Bringmann
FZI Forschungszentrum Informatik
Intelligent Systems and Production Engineering (ISPE)
Email: bringmann@fzi.de