technische universität
dortmund

Towards versatile Models for Contemporary Hardware Platforms

12th Workshop on Operating Systems Platforms for Embedded Real-Time Applications
Toulouse, France 2016

Hendrik Borghorst, Karen Bieling and Olaf Spinczyk

hendrik.borghorst@udo.edu
https://ess.cs.tu-dortmund.de/~hb

Embedded System Software Group
Computer Science 12, TU Dortmund

Motivation

- Operating system design for cyber-physical systems
- Special hardware is expensive

Goals:

- Predictable execution platform
- Use cheap multi-core hardware
  - **ARM**
  - (X86)
- Hardware not predictable without special care
  - Sophisticated hardware knowledge necessary

Measures for predictability

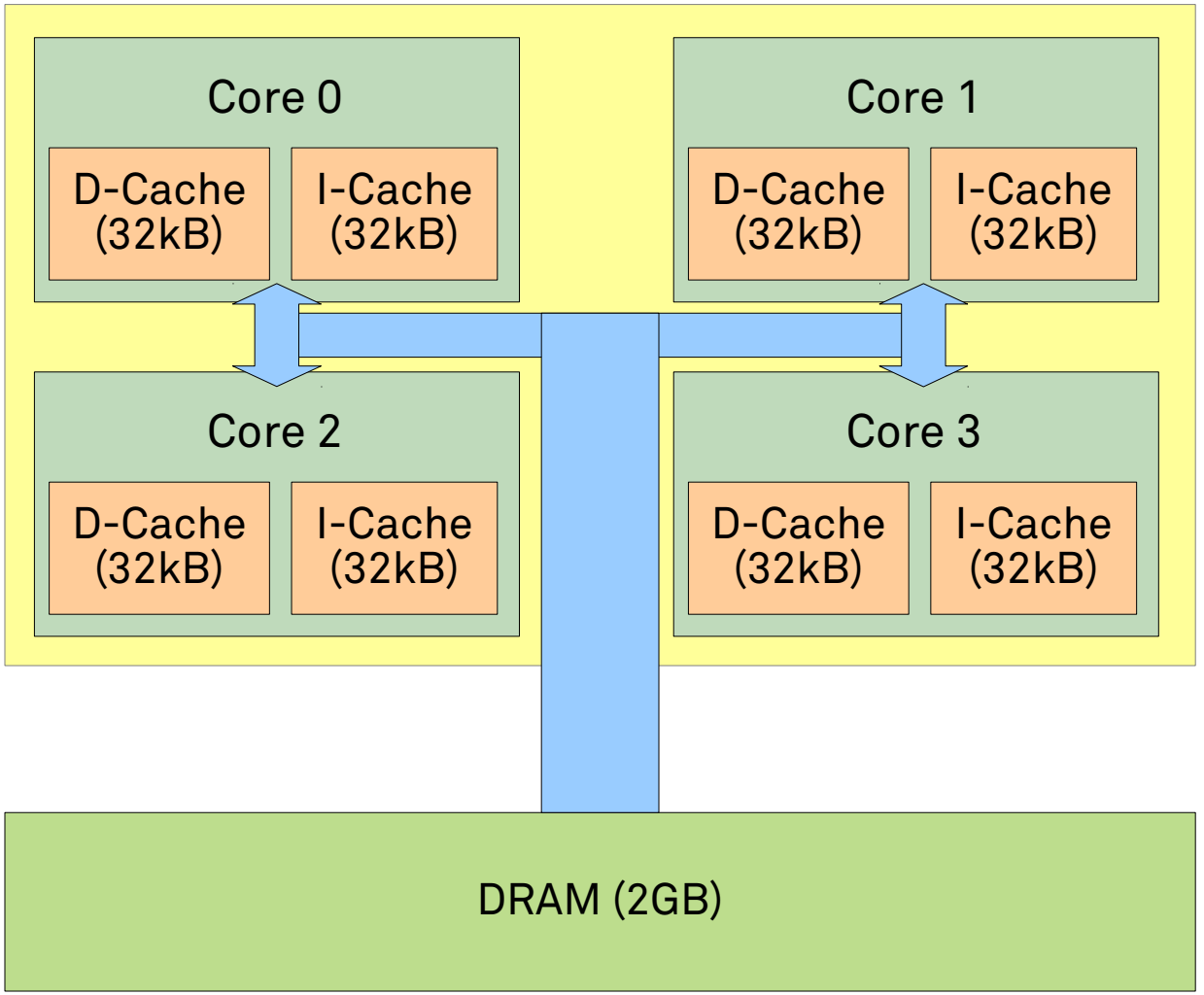
- Memory access rescheduling
- Alignment to cache-lines / cache-ways
- Cache partitioning
  - Hardware-based with cache controller support
  - Software-based methods
- ...

Measures for predictability

- Memory access rescheduling
- Alignment to cache-lines / cache-ways
- Cache partitioning
  - Hardware-based with cache controller support
  - Software-based methods
- ...

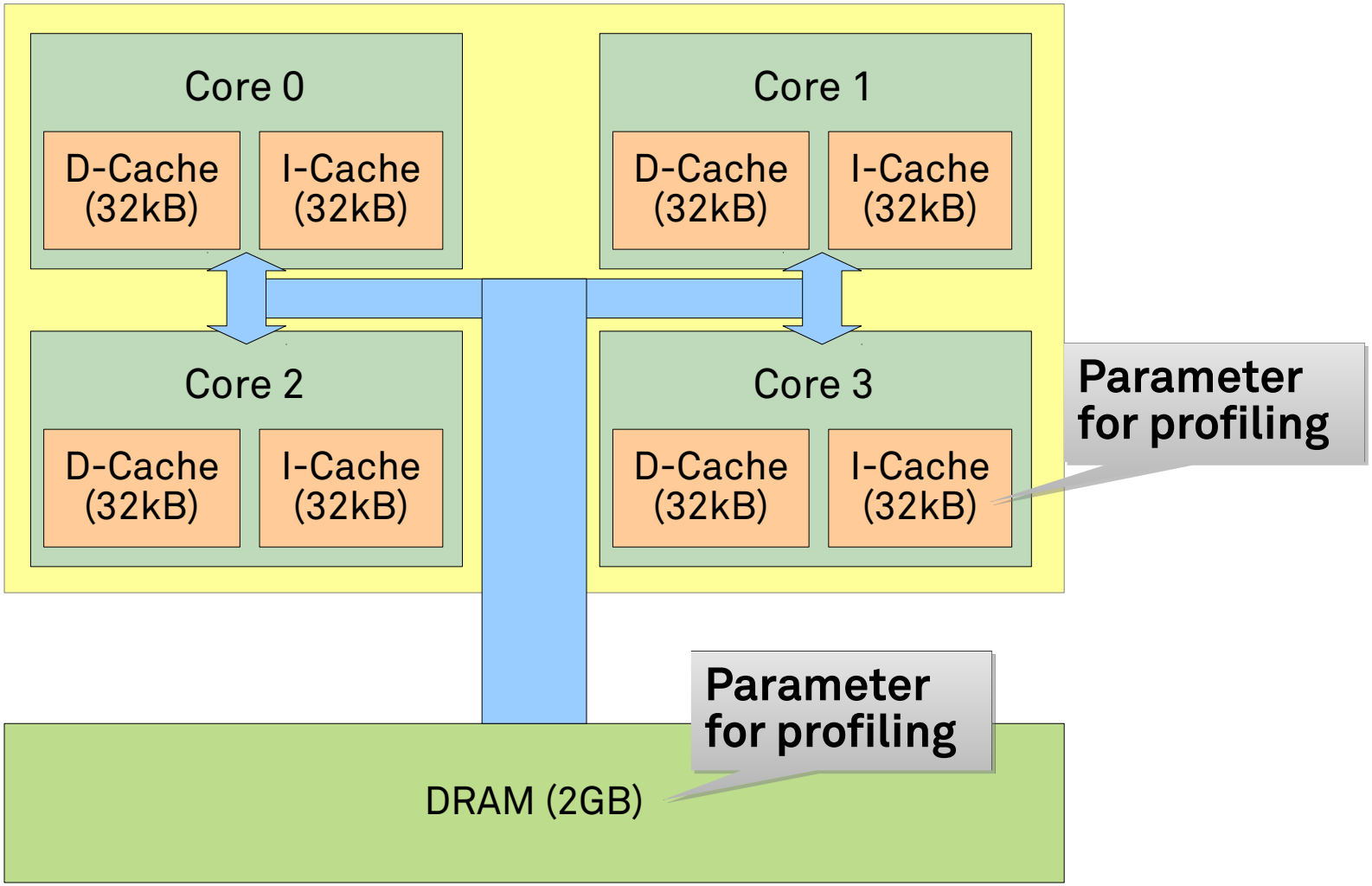
Dependency on hardware-specific parameters & knowledge

Use case: Memory profiling



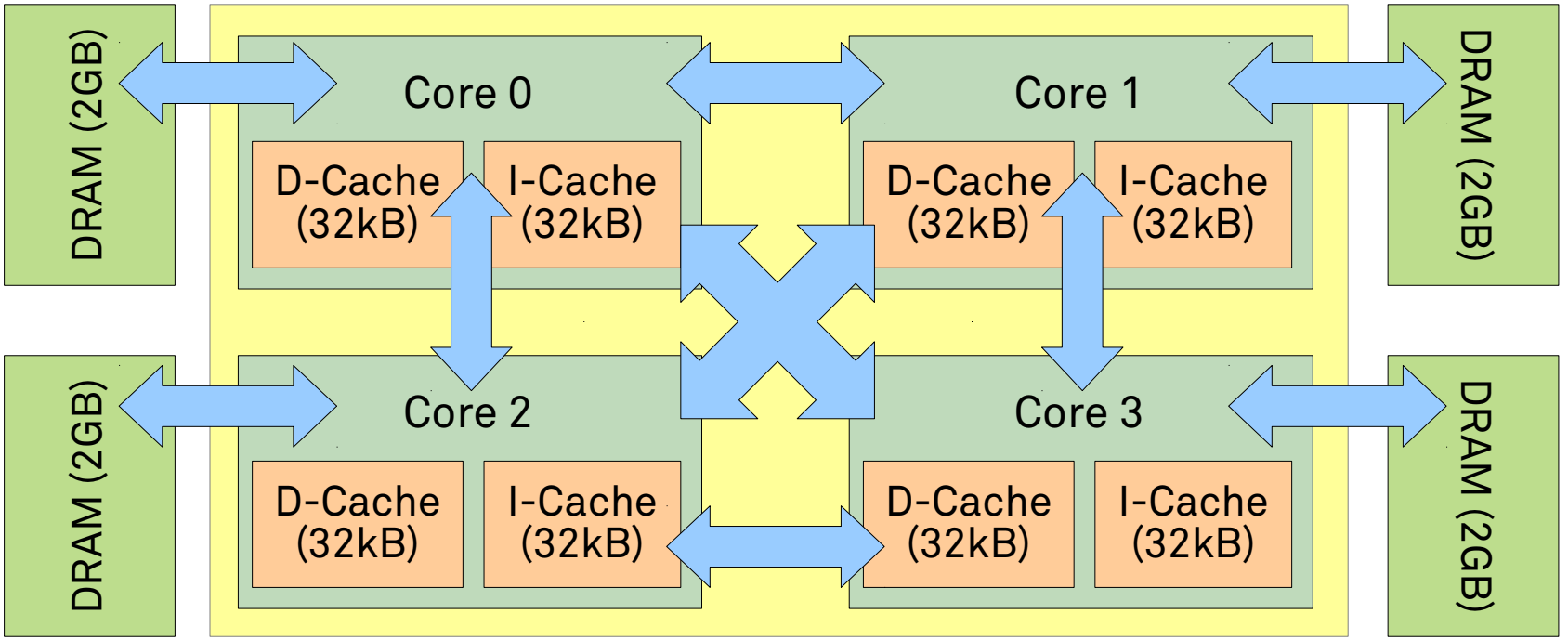
Example: Typical ARM SoC with DRAM

Use case: Memory profiling



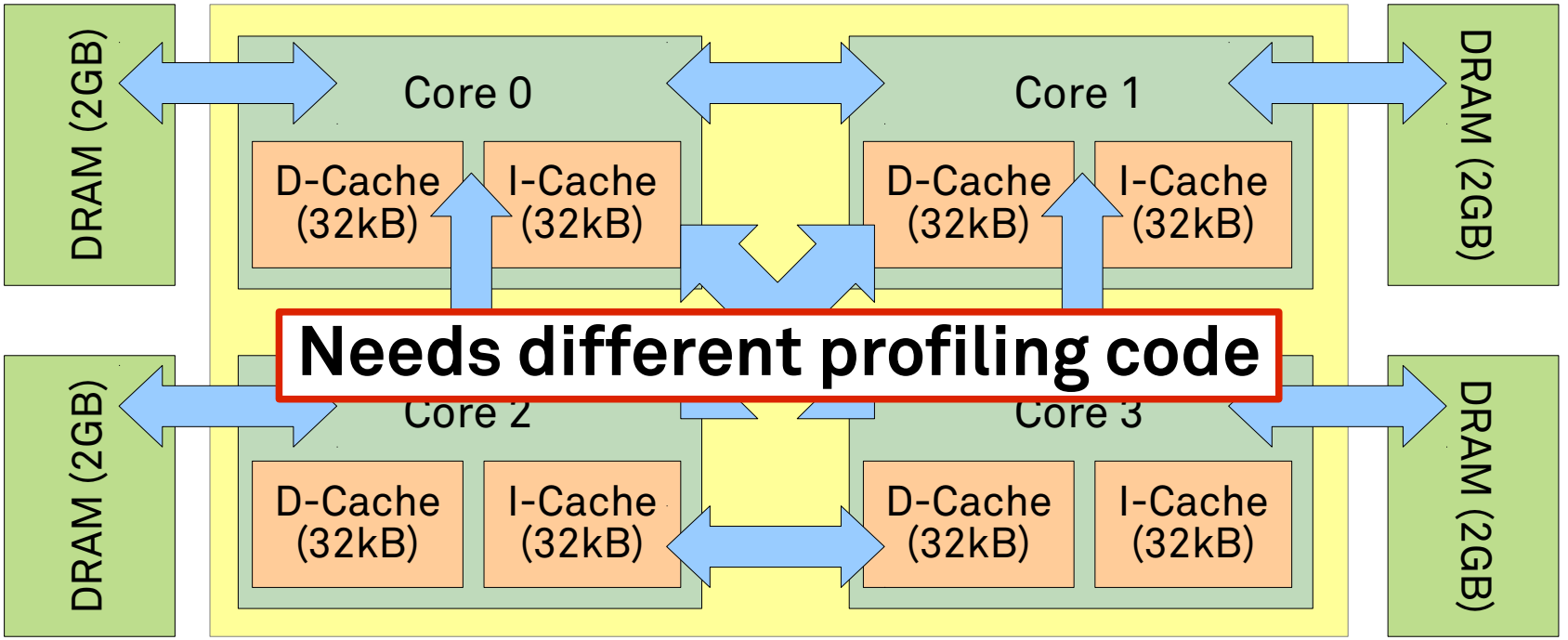
Example: Typical ARM SoC with DRAM

Use case: Memory profiling



Example: NUMA-Architecture

Use case: Memory profiling



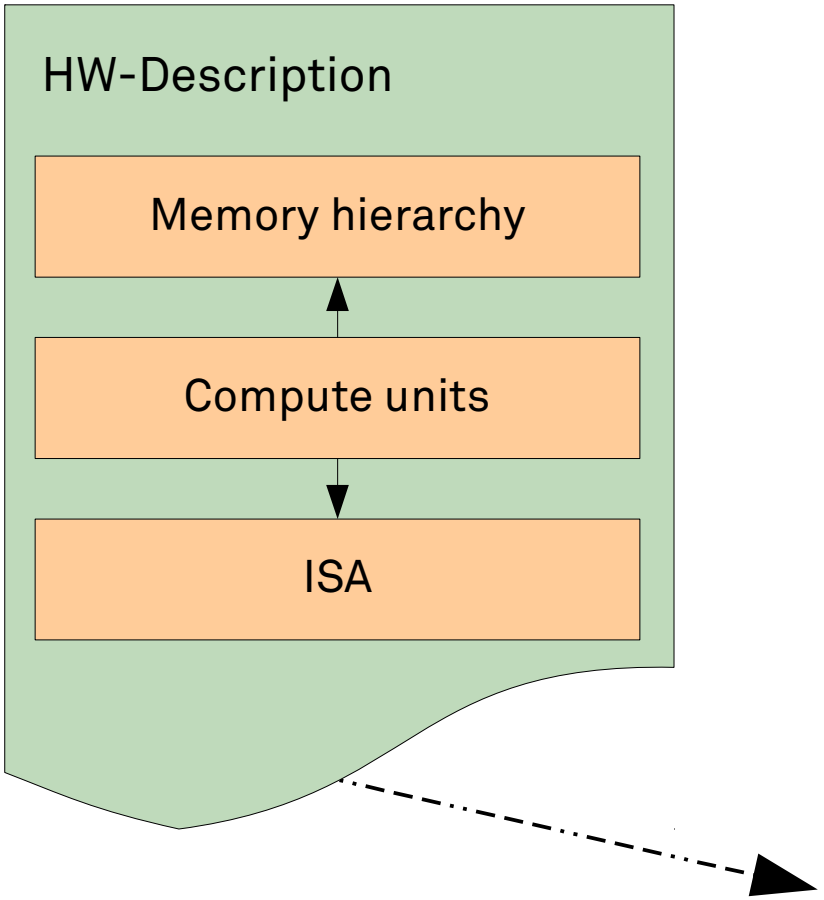
Example: NUMA-Architecture

Proposed Solution

- Use **Domain-Specific Languages**

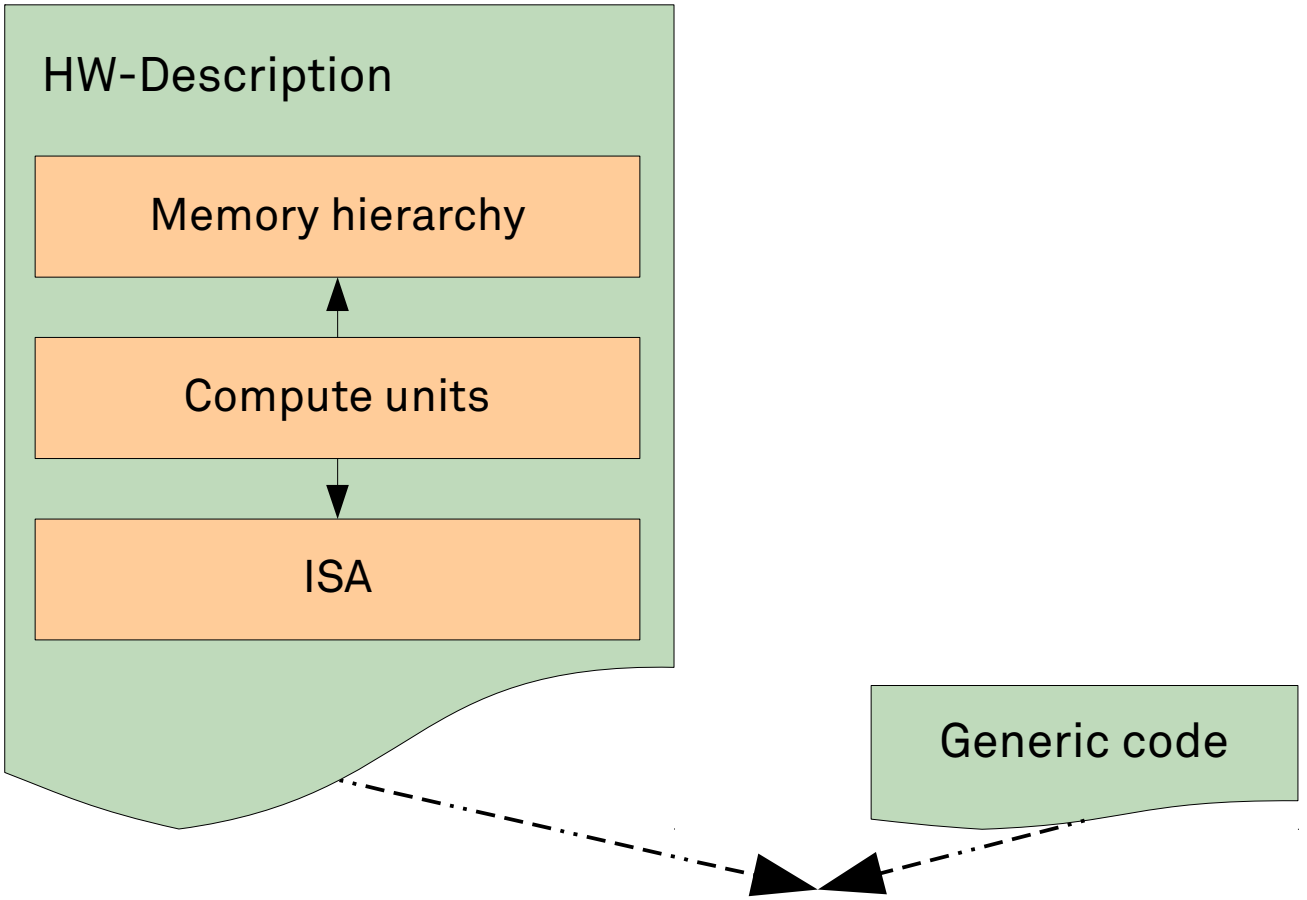
Proposed Solution

- Use **Domain-Specific Languages**



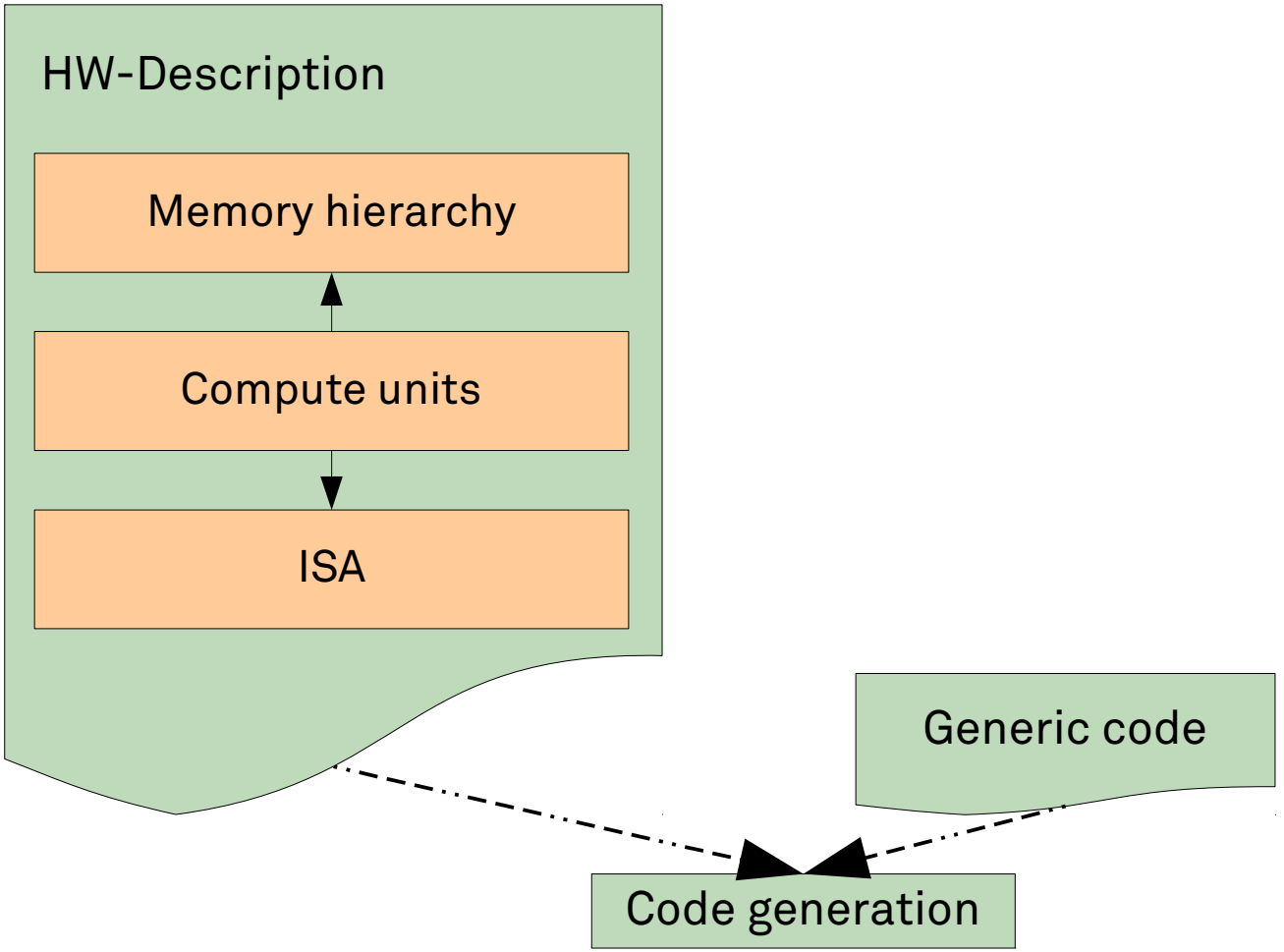
Proposed Solution

- Use **Domain-Specific Languages**



Proposed Solution

- Use **Domain-Specific Languages**



Architecture Description

```
architecture ExampleArch {
 wordLength: 4
 Memory RAM {}
 Memory L2Cache : RAM {}
 Memory Cache0 : L2Cache {}
 Memory Cache1 : L2Cache {}
 Processor CPU0 : Cache0 {}
 Processor CPU1 : Cache1 {}

 ISA {
  registers { R%[0..15] }
  instructions {
     add_const ADD: dest, arg, #arg const
     add_reg ADD: dest, arg, arg
   }
  }
}
```

Memory components

Compute units

Instruction set architecture (referenced by compute units)

Memory hierarchy description

```
Memory Cache1 : L2 {
        minAccessSize: 16 // Cache-line size (bytes)
        size: 64kB
}
```

```
Memory  L2 : RAM {
        minAccessSize: 32 // Cache-line size (bytes)
        size: 1M
}
```

```
Memory RAM {
        startAddress: 0x40000000
        size: 2G
}
```

memory connection

Abstract assembly code

```
ram_benchmark {
  move(dest reg:0,arg %[bmStart_<BM>])
  move(dest reg:1,arg %[bmEnd_<BM>])

  jmp_mark(arg "loop_begin:")
  measure_start()
  load(dest reg:3,src *reg:0)
  measure_end()

  add_const(dest reg:1,arg reg:1,
  arg <wordLength>)
  cmp(arg reg:0,arg reg:1)
  cond_jump_lt(arg "loop_begin")
}
```

Time measurement functions

Replaced with HW-specific values

Abstract assembly code

```
ram_benchmark {
  move(dest reg:0,arg %[bmStart_<BM>])
  move(dest reg:1,arg %[bmEnd_<BM>])

  jmp_mark(arg "loop_begin:")
```

Time measurement functions

Can be used to benchmark all memory components of an architecture

```
  add_const(dest reg:1,arg reg:1,
  arg <wordLength>)
  cmp(arg reg:0,arg reg:1)
  cond_jump_lt(arg "loop_begin")
}
```

Replaced with HW-specific values

Prototype implementation

- Implemented with Eclipse **EMF** & **Xtext**
- **Fast prototyping** and **adaption** to requirements of OS-design

- Test case:
  - Memory system profiling (RAM & Cache access times)
  - Code generation for memory hierarchy profiling
  - Samsung Exynos 4412 (Odroid U3 Development-Board)

eclipse Xtext https://eclipse.org/Xtext/

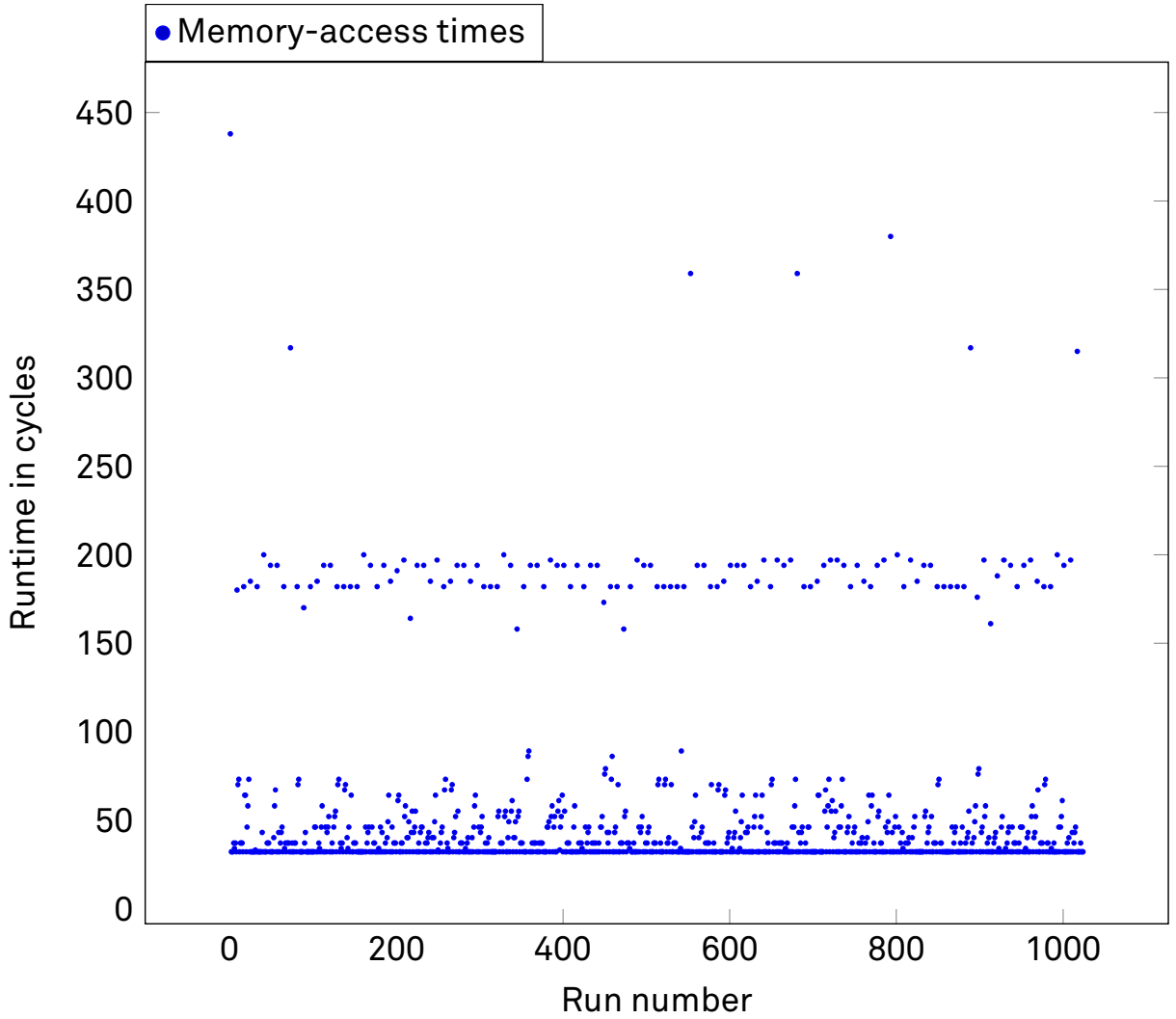
Code generation process

- Provide architecture description
  - Processor cores (4xARMv7 – Cortex A9)
  - Private caches per core
  - Shared L2-cache connected to private caches
  - 2GB of memory connected to L2-cache
- Abstract benchmark file
  - Iterates over memory range and loads words

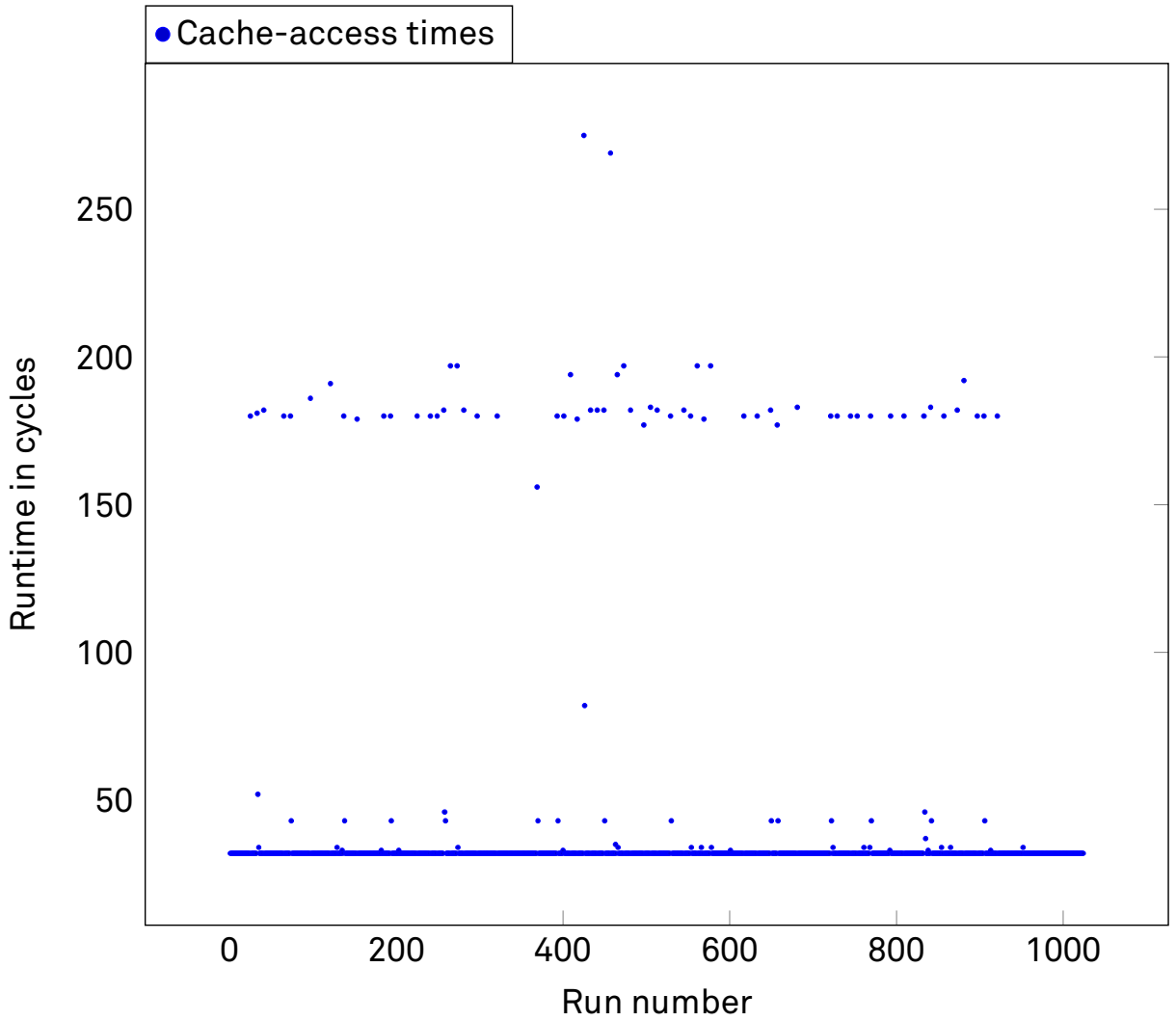
→ Assembly code is generated and integrated with a bit of glue code

→ Runs on bare-metal OS without interference

Prototype results (DRAM-access times)



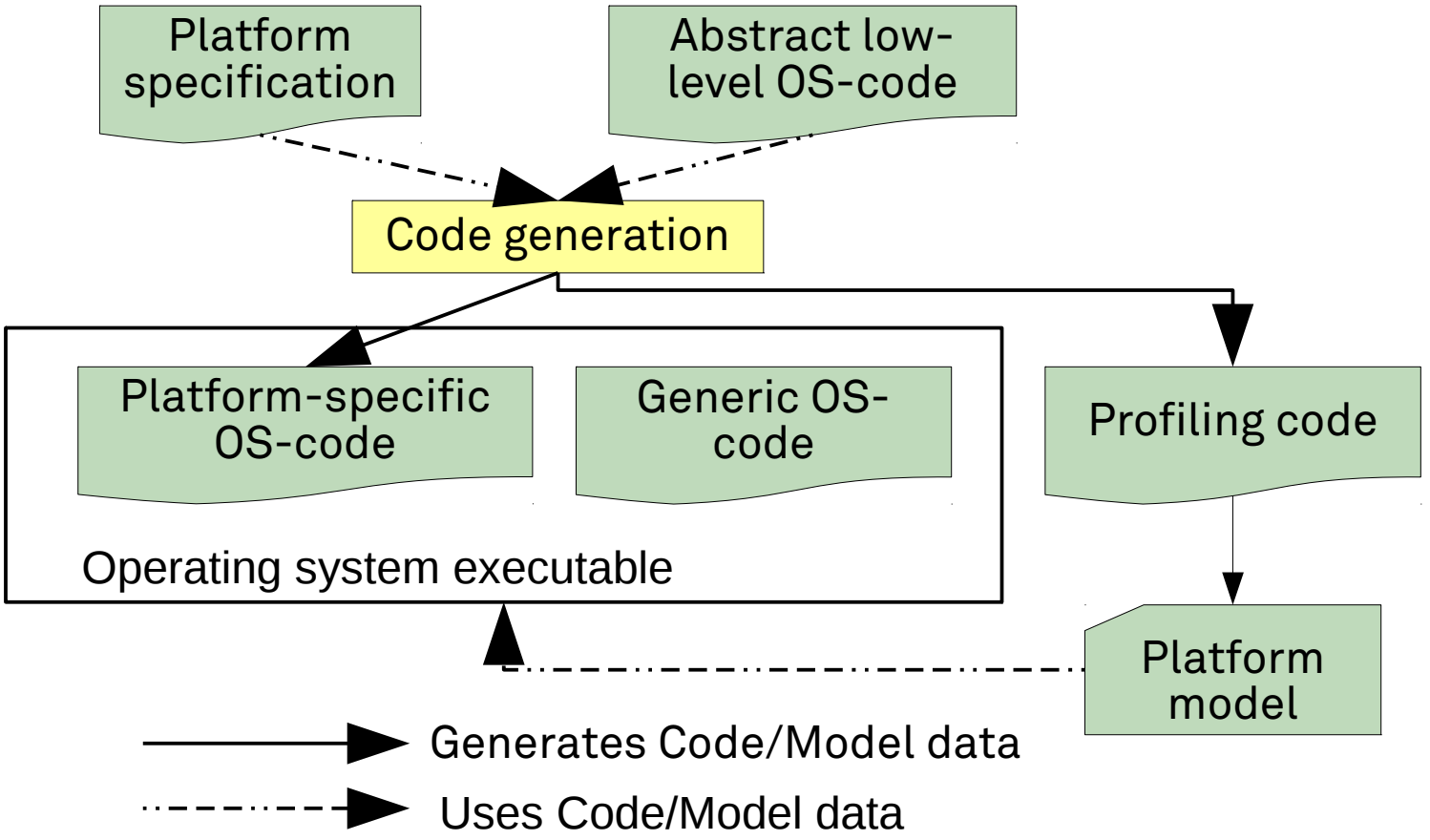
Prototype results (Cache-access times)



Future work

- Use generic profiling code to create comprehensive hardware models to use in OS-design
- Optimize code generation process
  - Automatic register allocation
  - Increase flexibility of abstract assembly
- Test with real world hw-specific code (Linux ?)
- Create open hardware model database

Goal


Platform specification
Abstract low-level OS-code
Code generation
Platform-specific OS-code
Generic OS-code
Profiling code
Operating system executable
Platform model
Generates Code/Model data
Uses Code/Model data

Conclusion

- Hardware-specific code generation based on:
  - Hardware-architecture description
  - Abstract code
- Multiple use-cases:
  - Hardware profiling
  - Hardware-specific operating system code
  - Application optimizations
- Hardware-models useful for predictable OS-design

Thank you for your attention

References

- J. M. Calandrino and J. H. Anderson, “Cache-aware real-time scheduling on multicore platforms: Heuristics and a case study,” in 20th Euromicro Conf. on Real-Time Sys. (ECRTS ’08), Jul. 2008, pp. 299–308.
- D. Dasari, B. Akesson, V. Nelis, M. Awan, and S. Petters, “Identifying the sources of unpredictability in COTS-based multicore systems,” in 08th IEEE Int. Symp. on Industrial Embedded Systems (SIES 2013), Jun. 2013, pp. 39–48.
- S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, “Memory access scheduling,” in 27th Int. Symp. on Comp. Arch. (ISCA ’00). New York, NY, USA: ACM, 2000, pp. 128–138.
- H. Borghorst and O. Spinczyk, “Increasing the predictability of modern COTS hardware through cache-aware OS-design,” in 11th W’shop on OS-Platf. for Emb. Real-Time App. (OSPERT ’15), Jul. 2015.
- D. Molka, D. Hackenberg, R. Schone, and M. S. Muller, “Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system,” in Parallel Architectures and Compilation Techniques, 2009. PACT ’09. 18th International Conference on, Sep. 2009, pp. 261–270.