Model based system configuration and tasteful hardware

Reto Achermann, Lukas Humbel, Daniel Schwyn, David Cock and Timothy Roscoe

Systems Group, Department of Computer Science, ETH Zurich

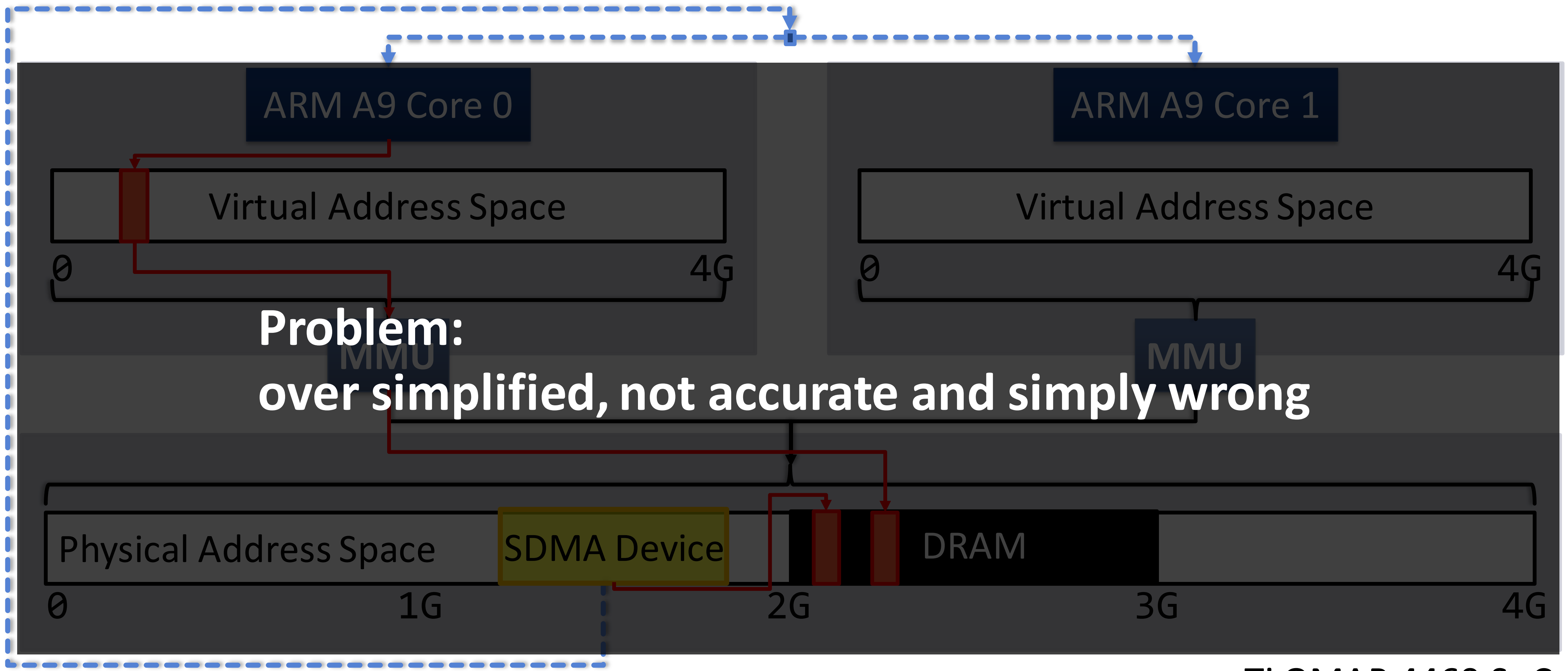
Systems@ETH zürich

Tasteful hardware – or the pursuit of happiness for a systems programmer

- Operating systems **wrongly** assume a single, uniform address space
- DeviceTrees fail to express important **details** of the hardware configuration
- Programming devices imposes a lot of **constraints** on resource management (memory regions, interrupts, …)

Ideally, we like to accurately **express** hardware configuration and then **generate** a configuration for it.

Tasteful hardware – or the textbook abstraction



Problem:
over simplified, not accurate and simply wrong

Ti OMAP 4460 SoC

ETH zürich

Reality: The devil is in the details

Your mobile phone… 5-10 years ago!

6+ heterogeneous cores

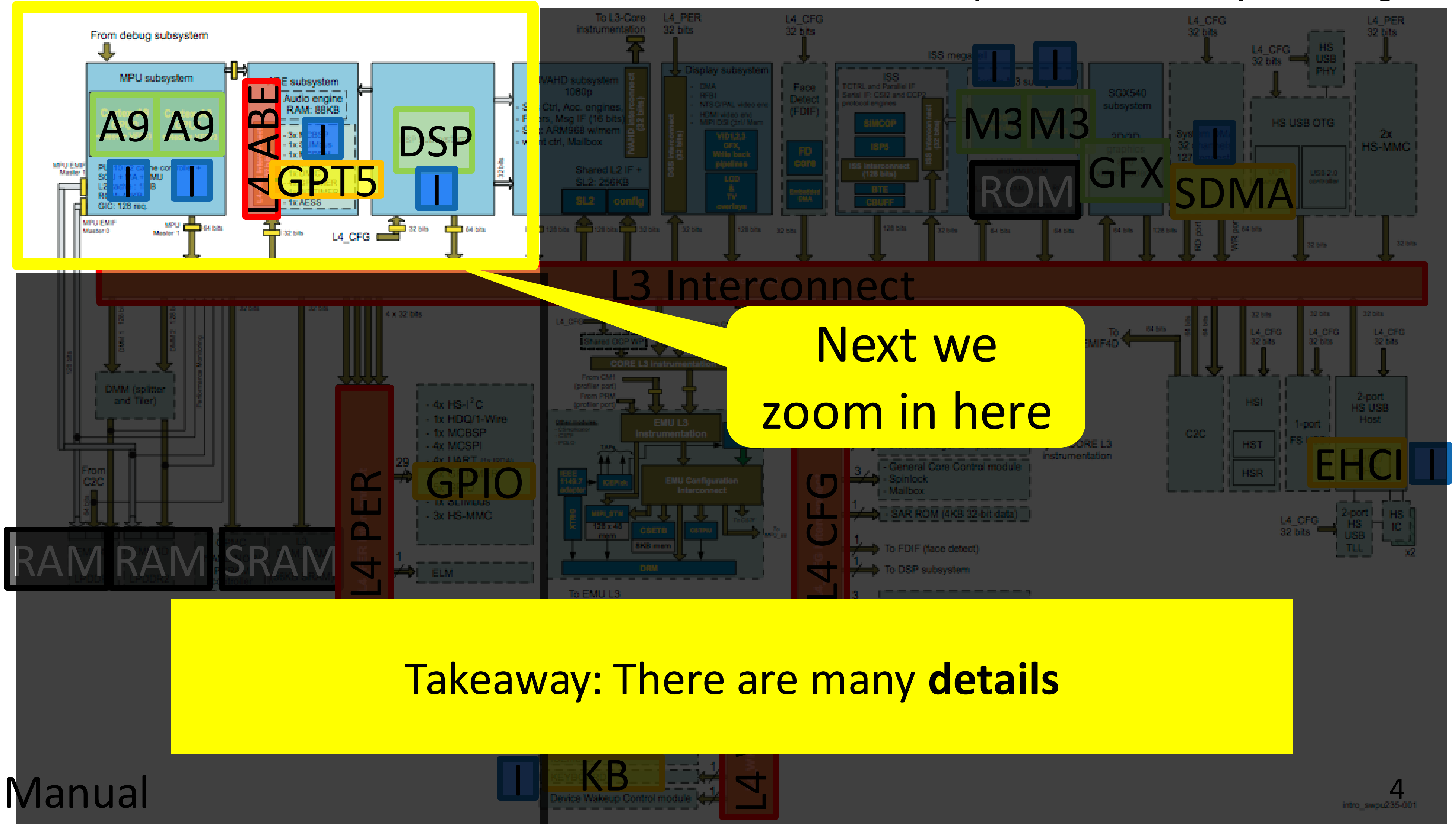
Private and shared memory

5+ Interconnects

Devices attached to different interconnects

Complex interrupt subsystem

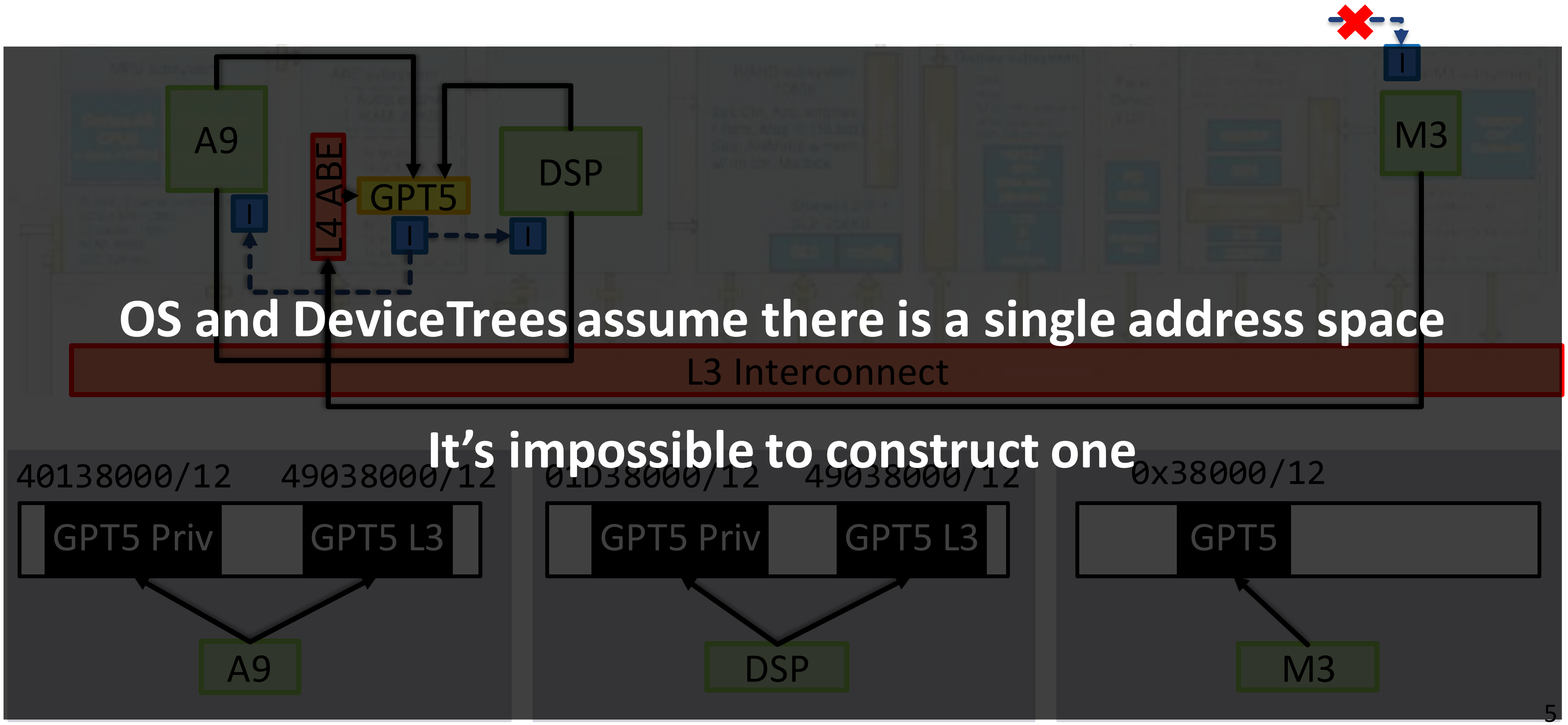
OMAP 4460 SoC, Technical Reference Manual



Takeaway: There are many **details**

There is NO uniform view of the system

OMAP 4460 SoC



OS and DeviceTrees assume there is a single address space

It's impossible to construct one

ETH zürich

We need an accurate representation of hardware

- Experience from the Barrelfish operating system: dealing with this complexity every day.
  - *PCI programming, devices and device drivers,*
  - *Heterogeneity, new architectures, interconnects and platforms*
  - *Resource management: memory allocation, interrupt vectors, …*
- We don’t want to make the same **wrong/imprecise** assumptions
- Provide a sound **hardware description** to facilitate system software **verification**

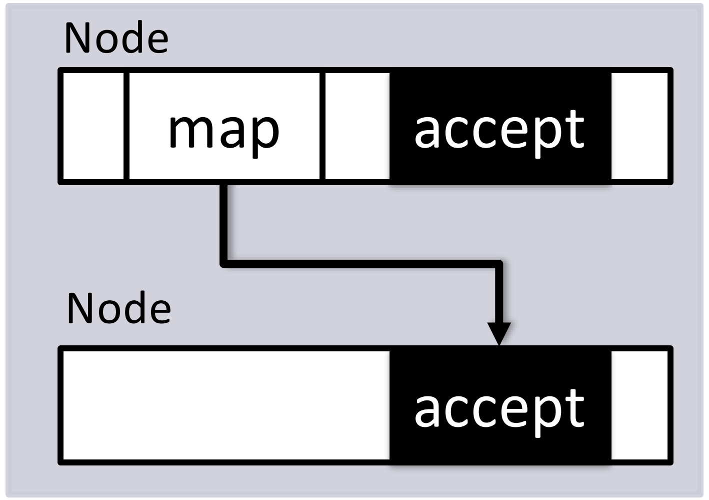
Barrelfish.org

System programmer’s wishes

- Write **correct** systems code, and do this **fast**
  Memory management, device drivers, support for new platforms and architectures
- Make **accurate statements** about hardware platforms
  At which address can a core/device access memory, what are the access constraints
- **Generate** system configuration and synthesize configuration algorithms
  Page tables, DMA bus addresses

Outline

1. **Formal model** to accurately represent memory and interrupts subsystems
2. Generate **configuration** based on the model specification
3. Model **refinement** example using MIPS R4600 TLB



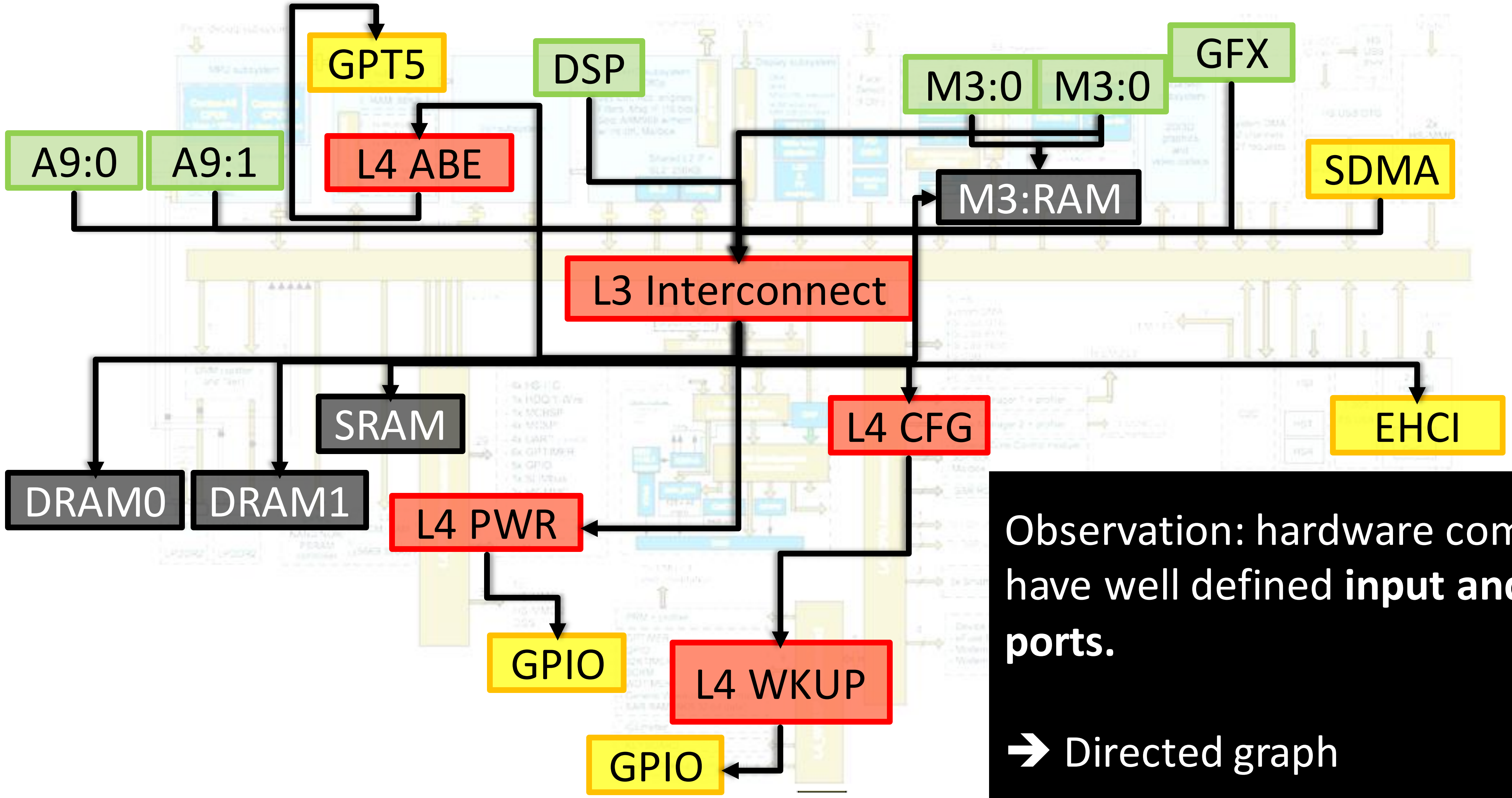
1. A formal model for memory accesses and interrupts

Design goals of the formal model

- Close to hardware: capture as much **details** as possible
- Represent the **interactions** between the components
  signal path of a memory access / interrupt
- Enable different **views/observers** of the system
  e.g. cores / devices see the system differently

ETH zürich

Interaction of hardware components on the OMAP4460



Observation: hardware components have well defined **input and output ports.**

➔ Directed graph

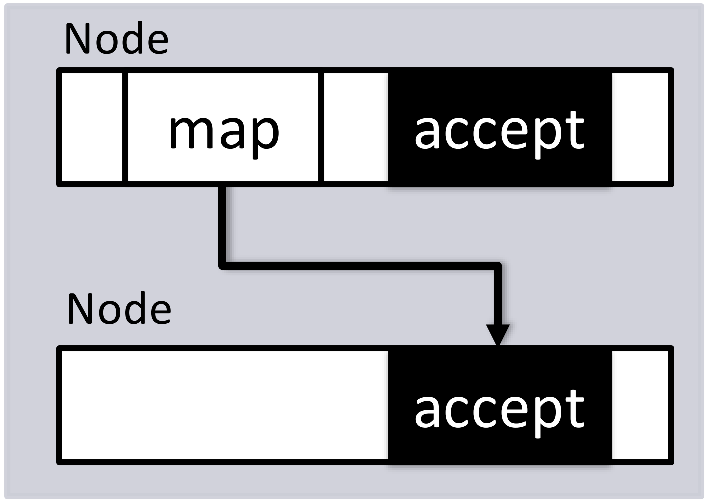
ETH zürich

Modelling memory accesses and interrupts as a decoding net

- The model is a **decoding net,** a directed graph
- Hardware components ⬄ **nodes**

  Address spaces, translation units, devices, interrupt controllers, interconnects, …
- Formally: assignment of identifiers to nodes

$$net: \mathbb{N} \rightarrow node$$



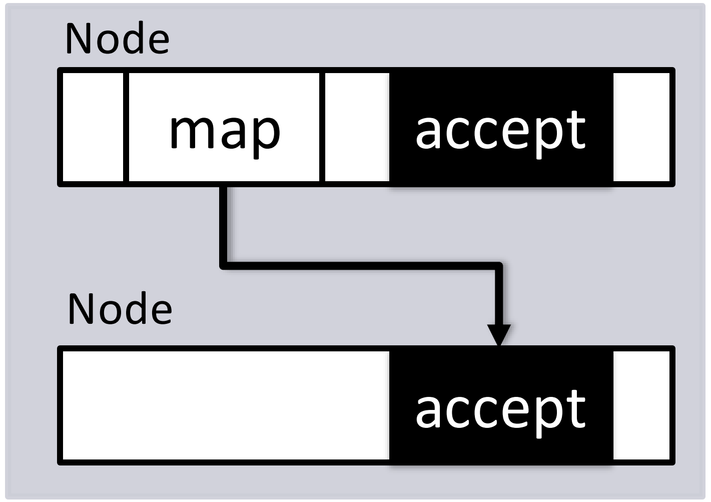
Modelling memory accesses and interrupts as a decoding net

- Nodes have **two properties**:
  - **Accept**: set of addresses this node responds to
  - **Translate**: map input address at node to a name

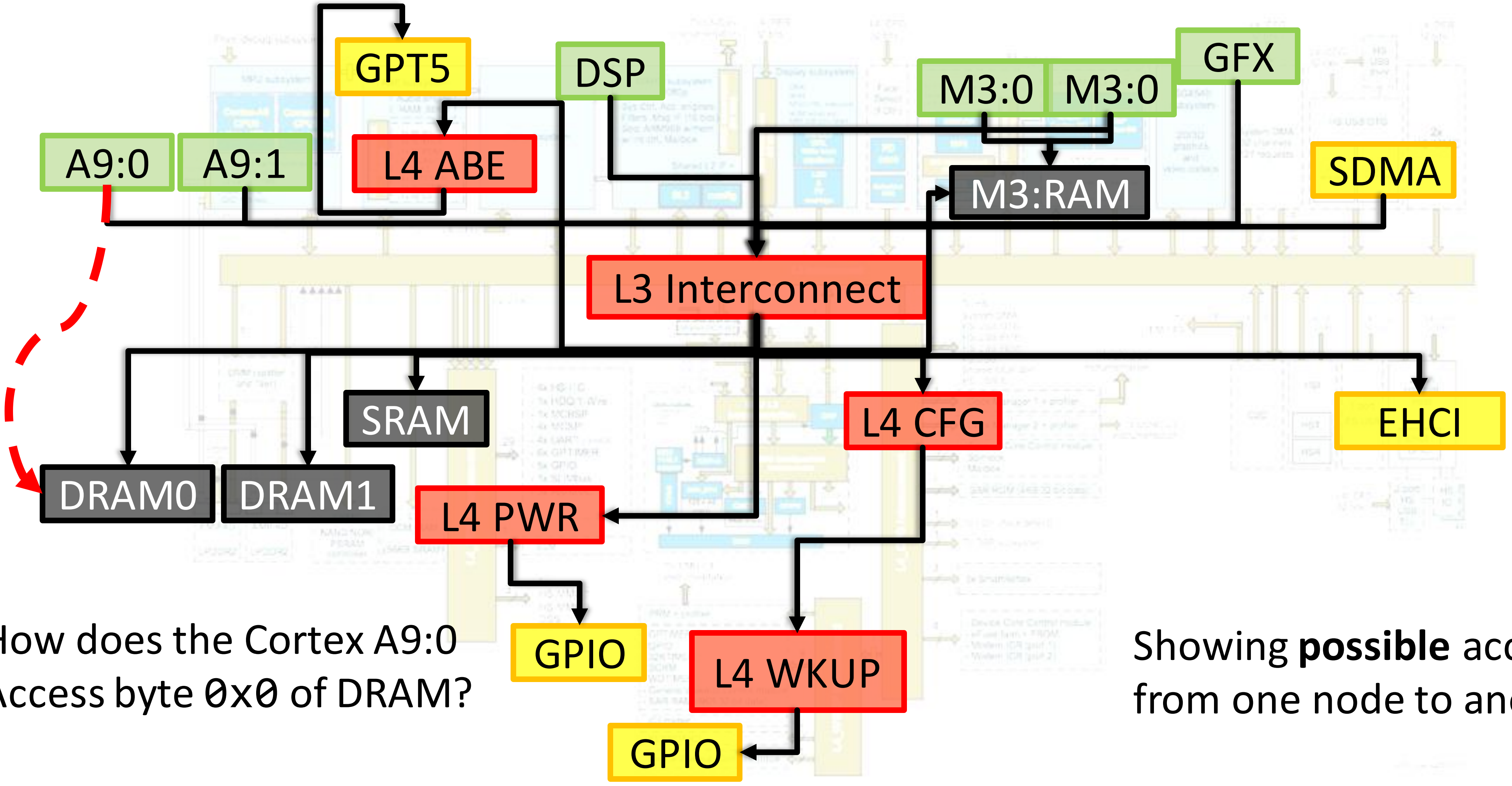
node:

$$accept :: node \rightarrow \{\mathbb{N}\}$$
$$translate :: node \rightarrow \mathbb{N} \rightarrow \{name\}$$

- Name is a **qualified address** at a node: $name = (\mathbb{N}, \mathbb{N})$



A possible decoding net for the OMAP 4460

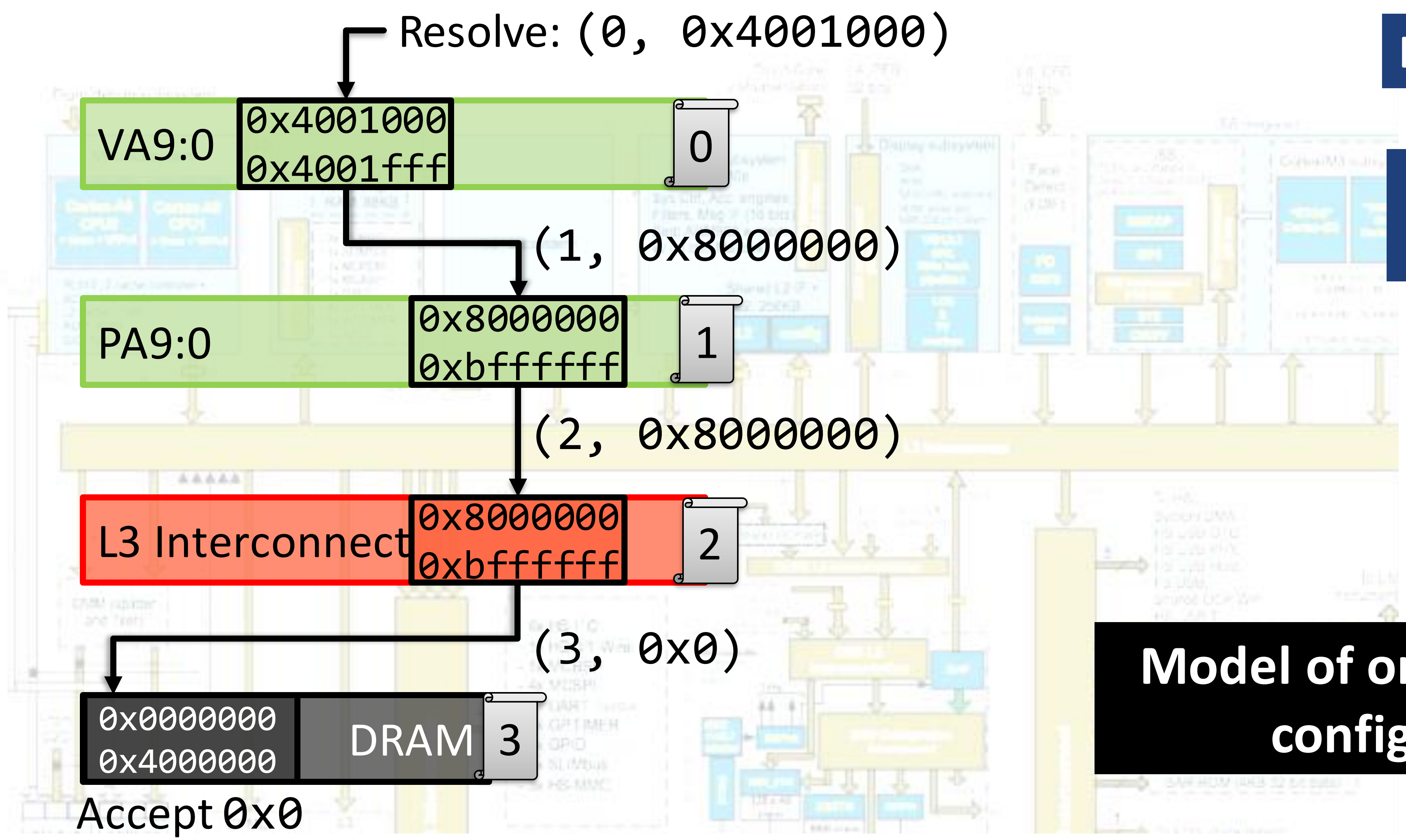


How does the Cortex A9:0 Access byte `0x0` of DRAM?

Showing **possible** accesses from one node to another

ETH zürich

Modelling the access to byte 0x0 of DRAM from an A9 core



Each node has a label

Resolve a **name** (node, address)

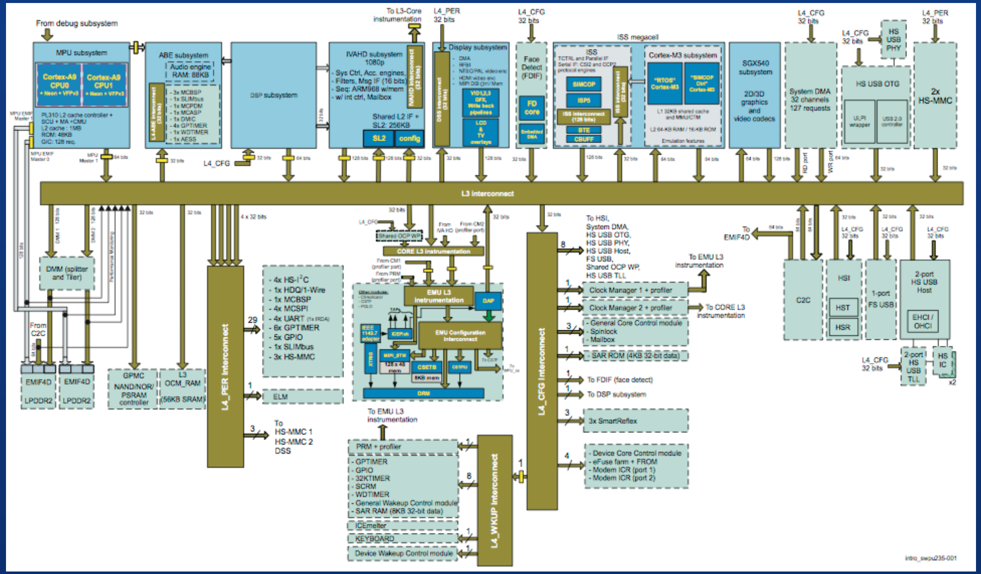
Model of one particular, static configuration state

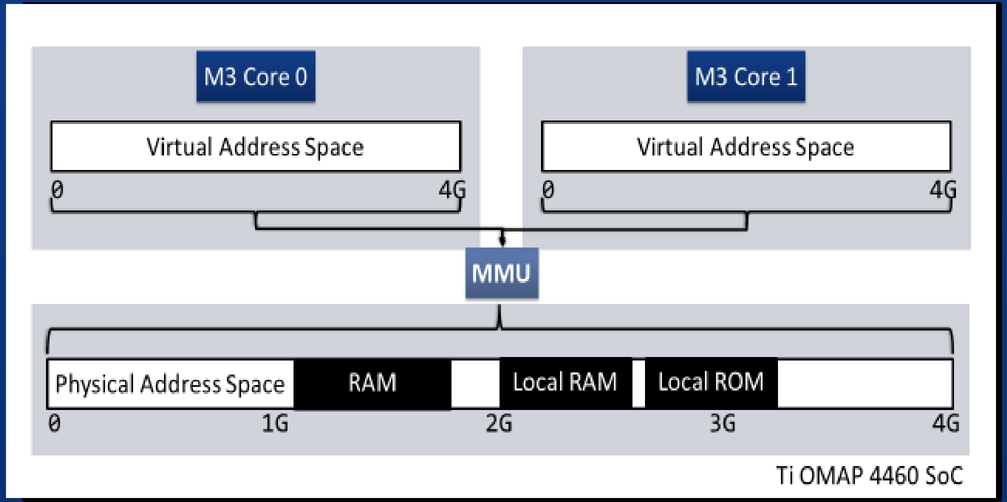
ETH zürich

View Equivalence

Resolve: (0, 0x4001000)

Resolve: (0', 0x4001000)





For ONE observer the flattened representation is equivalent to the textbook abstraction

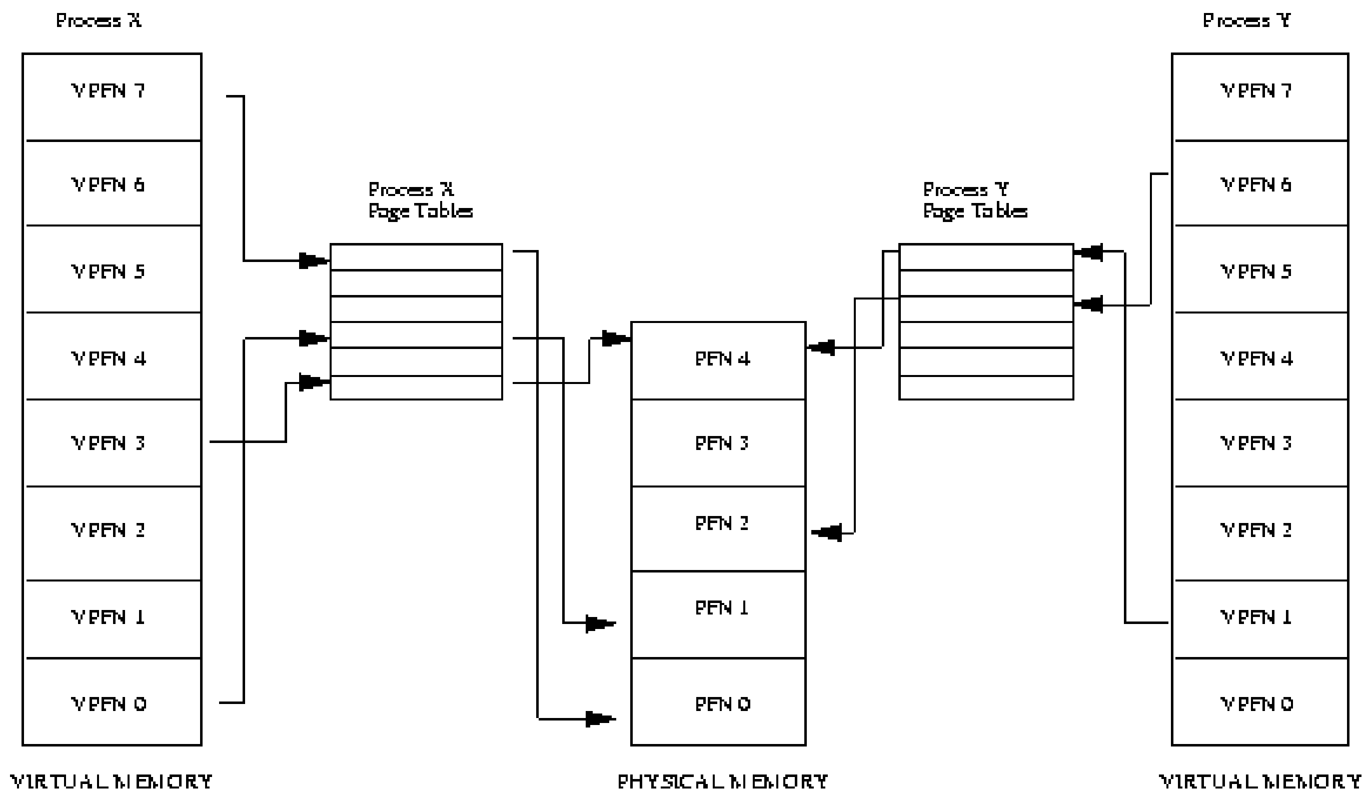
0x4000000

Accept 0x0

Summary

- Express the hardware configuration as a **decoding net model**
- View equivalence preserving transformations for a **fixed observer**
- Formally proven using Isabelle/HOL
- Memory access – Interrupt duality:
  The model is also applicable to the Interrupt subsystem

ETH zürich



2. Sockeye: Generate Hardware Configurations

Daniel Schwyn, Master's Thesis

Application of the model in the context of SoC and hardware discovery

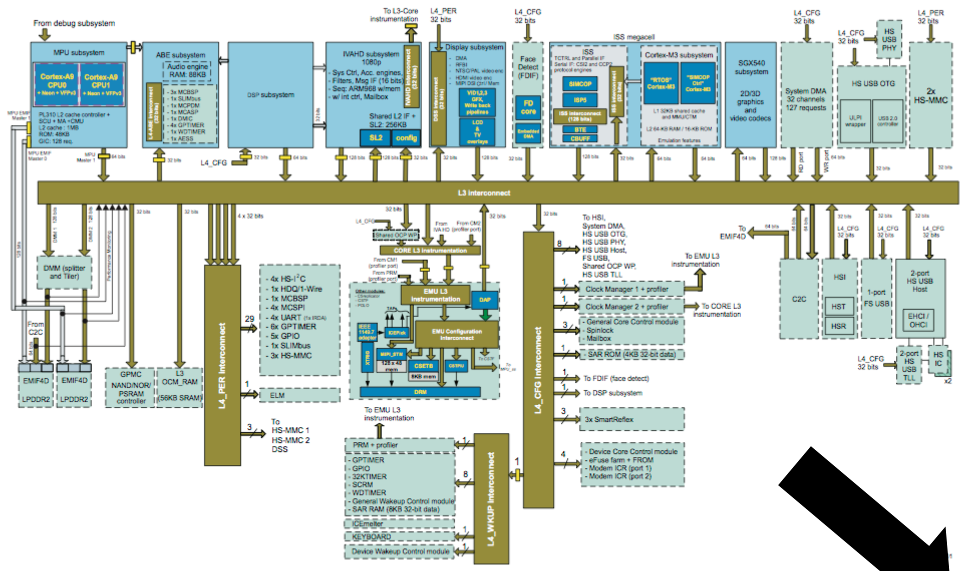
- At boot, the kernel needs to know about memory, devices, cores etc.
  ➔ **Devices trees** assumes a **uniform view**

- Our formal model accurately **describes hardware**
  ➔ generate hardware configuration based on the model (*e.g. page tables*)
  ➔ use description for runtime decisions (*e.g. memory allocation*)

- **Sockeye: a domain specific language to describe the hardware configuration**

Sockeye: a DSL for describing hardware

Models for x86 desktop & servers, Xeon Phi, InfiniBand Cluster, Intel’s Single Chip Cloud Computer



Sockeye: DSL + prolog queries

$V_{A9:0}\ \textbf{is map}\ [20000_3/12\ \textbf{to}\ P_{A9:0}\ \textbf{at}\ 80000_3]$

$P_{A9:0}, P_{A9:1}\ \textbf{are map}\ [40138_3/12\ \textbf{to}\ GPT\ \textbf{at}\ 0]\ \textbf{over}\ L3$

$P_{DSP}\ \textbf{is map}\ [\texttt{1d3e}_3/12\ \textbf{to}\ GPT\ \textbf{at}\ 0]\ \textbf{over}\ L3$

$V_{M3}, V_{M3}\ \textbf{are over}\ L1_{M3}$

$RAM_{M3}\ \textbf{is accept}\ [55020_3/16]$

$ROM_{M3}\ \textbf{is accept}\ [55000_3/14]$

$V_{A9:1}\ \textbf{is map}\ [20000_3/12\ \textbf{to}\ P_{A9:1}\ \textbf{at}\ 80000_3]$

$V_{DSP}\ \textbf{is over}\ P_{DSP}$

$L2_{M3}\ \textbf{is map}\ [0_{30}\ \textbf{to}\ L3\ \textbf{at}\ 80000_3]$

$L1_{M3}\ \textbf{is map}\ [0_{28}\ \textbf{to}\ MIF]$

$L4\ \textbf{is map}\ [49038_3/12\ \textbf{to}\ GPT\ \textbf{at}\ 0]$

$GPT\ \textbf{is accept}\ [0/12]$

$MIF\ \textbf{is map}\ [0-\texttt{5ffffff}\ \textbf{to}\ L2_{M3}, 55000_3/14\ \textbf{to}\ RAM_{M3}, 55020_3/16\ \textbf{to}\ ROM_{M3}]$

$L3\ \textbf{is map}\ [49000_3/24\ \textbf{to}\ L4\ \textbf{at}\ 40100_3, 55000_3/12\ \textbf{to}\ MIF]\ \textbf{accept}\ [80000_3/30]$

Sockeye Code

Background: Barrelfish's System Knowledge Base

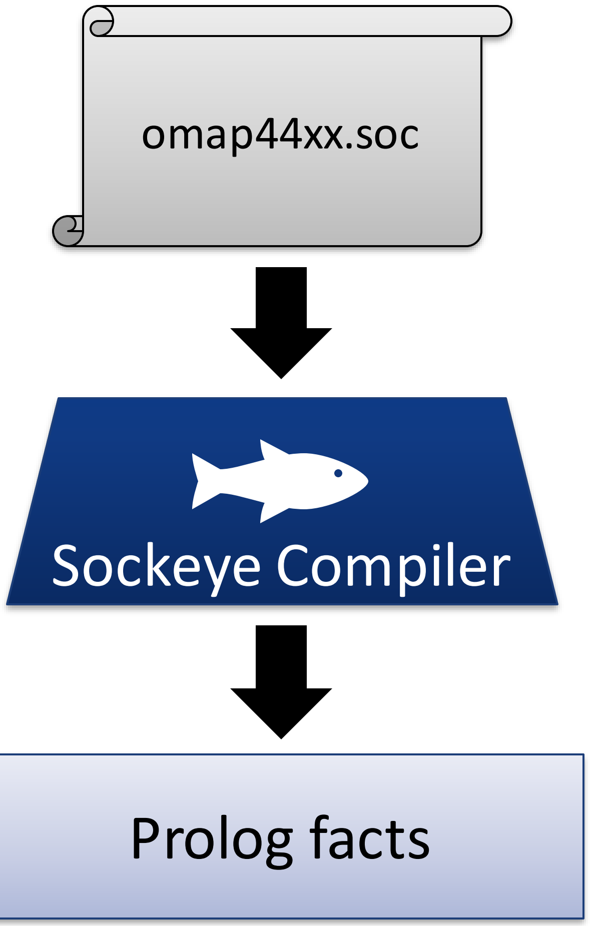
- Observation: resource allocation and hardware configuration can be expressed as a constrained satisfaction problem.
  e.g. PCI programming, DMA, …

- Barrelfish's approach: **SKB = Prolog engine + constraint solver**

- Information about the system is stored as **Prolog facts** in the SKB

- **Prolog queries** express allocation policies and configuration constraints

Sockeye Compiler: Generate the decoding net model



$V_{A9:0}$ **is map** $[20000_3/12$ **to** $P_{A9:0}$ **at** $80000_3]$

$P_{A9:0}, P_{A9:1}$ **are map** $[40138_3/12$ **to** GPT **at** $0]$ **over** $L3$

P_{DSP} **is map** $[\texttt{1d3e}_3/12$ **to** GPT **at** $0]$ **over** $L3$

V_{M3}, V_{M3} **are over** $L1_{M3}$

RAM_{M3} **is accept** $[55020_3/16]$

ROM_{M3} **is accept** $[55000_3/14]$

$V_{A9:1}$ **is map** $[20000_3/12$ **to** $P_{A9:1}$ **at** $80000_3]$

V_{DSP} **is over** P_{DSP}

$L2_{M3}$ **is map** $[0_{30}$ **to** $L3$ **at** $80000_3]$

$L1_{M3}$ **is map** $[0_{28}$ **to** $MIF]$

$L4$ **is map** $[49038_3/12$ **to** GPT **at** $0]$

GPT **is accept** $[0/12]$

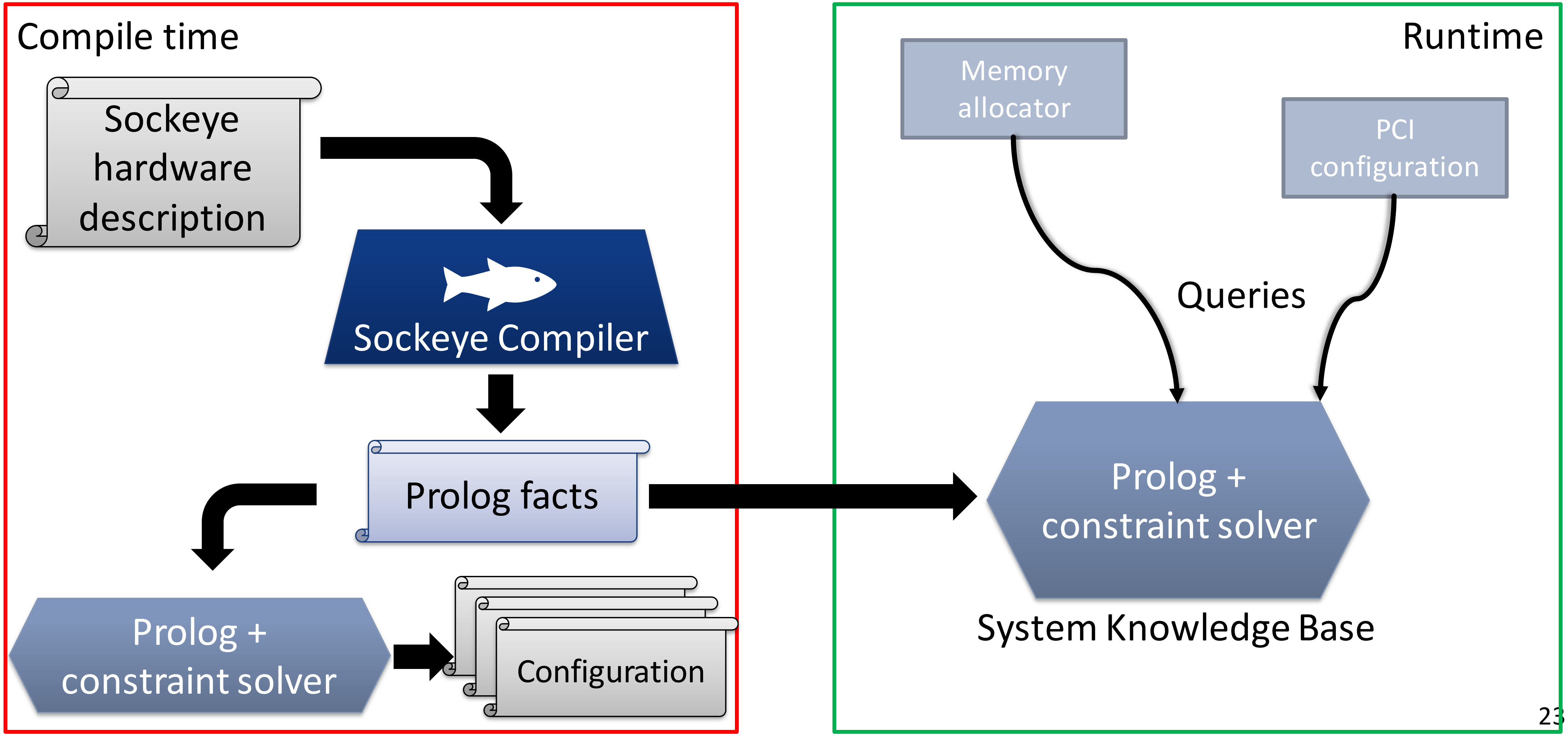
MIF **is map** $[0 - \texttt{5ffffff}$ **to** $L2_{M3}, 55000_3/14$ **to** $RAM_{M3}, 55020_3/16$ **to** $ROM_{M3}]$

$L3$ **is map** $[49000_3/24$ **to** $L4$ **at** $40100_3, 55000_3/12$ **to** $MIF]$ **accept** $[80000_3/30]$

Sockeye compiler converts the description into the **decoding net model** expressed as Prolog facts

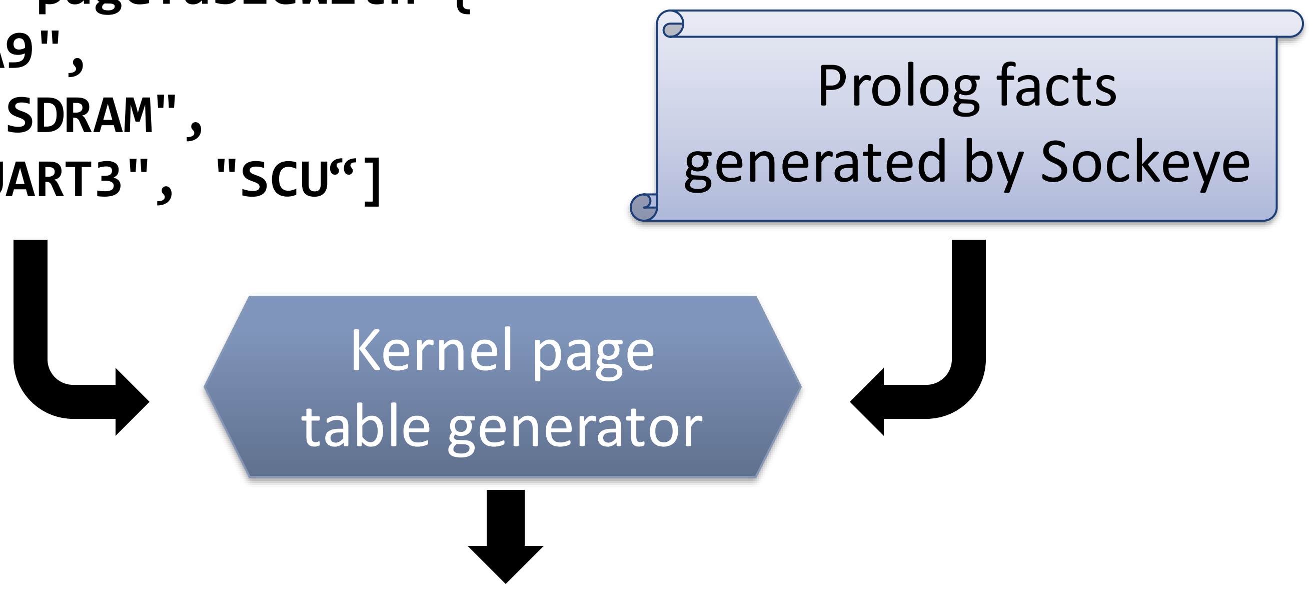
```
net('SRAM',node(memory,[block(16'0,16'3ffffff)],[],'@none')).
net('BOOT_ROM',node(memory,[block(16'0,16'bfff)],[],'@none')).
net('L3_OCM_RAM',node(memory,[block(16'0,16'dfff)],[],'@none')).
net('SDRAM',node(memory,[block(16'0,16'3ffffff)],[],'@none')).
```

Sockeye Runtime Environment



Application: Generation of kernel page tables

```
kernelPageTable = pageTableWith {
   cpu = "CORTEXA9",
   mainMemory = "SDRAM",
   devices = [ "UART3", "SCU“]
},
```


Prolog facts generated by Sockeye
Kernel page table generator

```
union arm_l1_entry l1_table[ARM_L1_MAX_ENTRIES]
    __attribute__((aligned(ARM_L1_ALIGN), section(".boot.tables"))) =
{
       [L1_TABLE_INDEX(0x7FE00000)] = L1_DEVICE_ENTRY(0x7FE00000),
       [L1_TABLE_INDEX(0x7FF00000)] = L1_DEVICE_ENTRY(0x7FF00000), …
       [L1_TABLE_INDEX(0x80000000)] = L1_MEMORY_ENTRY(0x80000000),
       [L1_TABLE_INDEX(0x80100000)] = L1_MEMORY_ENTRY(0x80100000), …
}
```

ETH zürich

Application: Memory allocation

DMA bus address allocation is required to satisfy various **constraints**

A good fit for the SKB:

- The decoding net model can resolve all **reachable memory regions** from the DMA device

- Express the constraints as a prolog query to get **usable memory regions**

Summary: Configuration generation

- Describe hardware configuration in the Sockeye DSL
- Obtain a Prolog representation of the decoding net model
- Generate hardware configuration
- Using Barrelfish’s SKB to perform DMA bus address allocation

3. Case Study with the MIPS R4600 TLB

ETH zürich

MIPS R4600 TLB Case study

- Software loaded TLB: explicit management of the TLB entries
  → we don’t need to worry about hardware page table walker

- We want to express the MIPS R4600 TLB in the decoding net model

- **Main Question:** Can we refine the abstract decoding net node to a **dynamic** TLB node?

MIPS R4600 TLB entry format at a glance

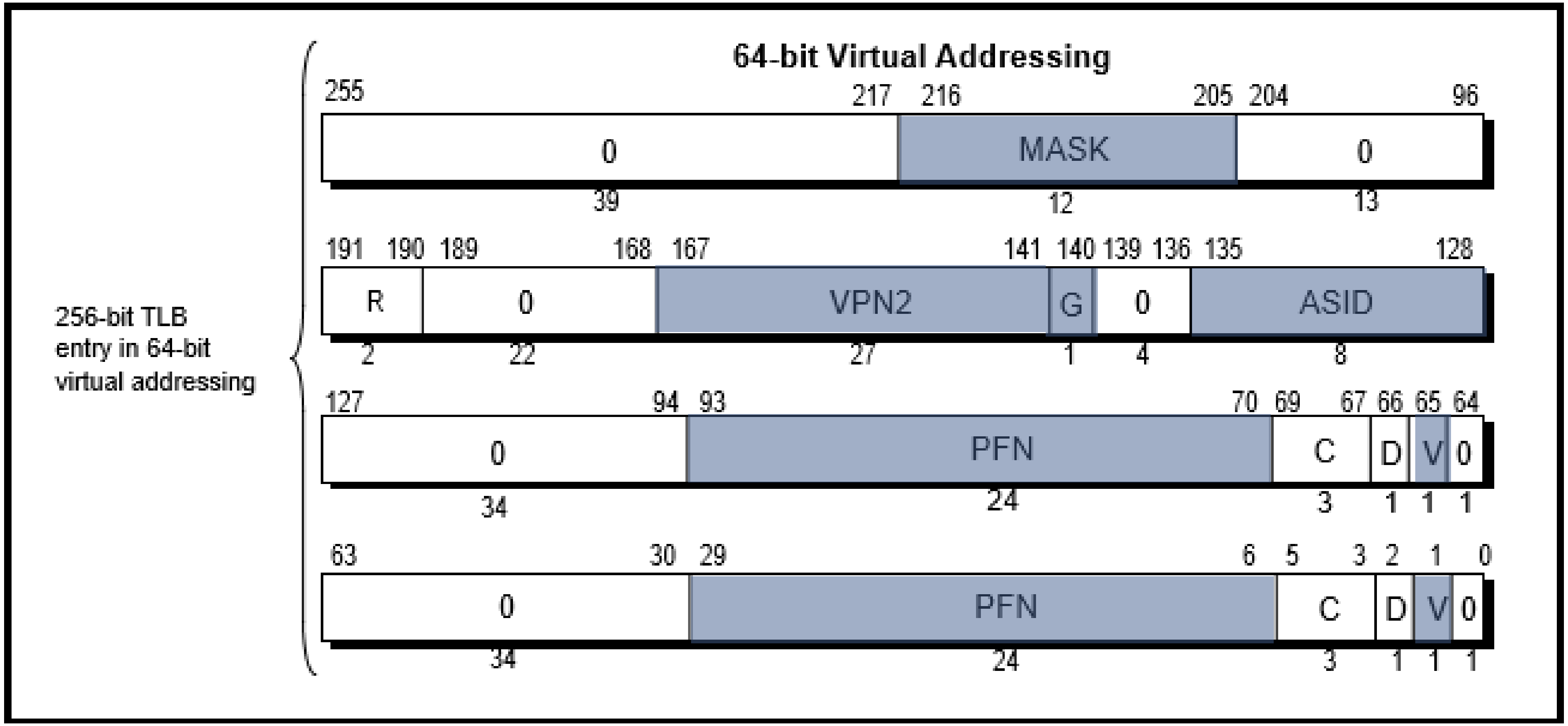
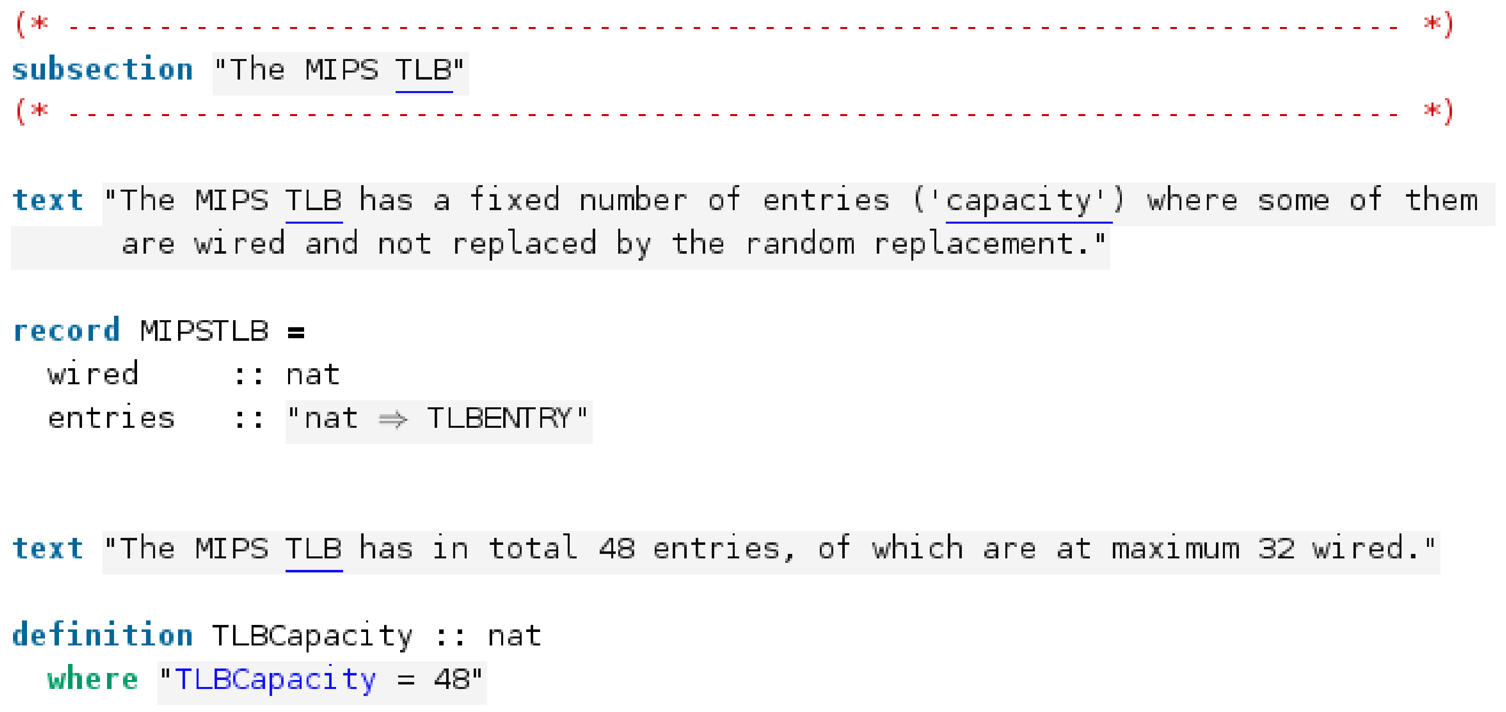


Figure 4.8 Format of a TLB Entry

ETH zürich

Modelling the MIPS R4600 TLB in Isabelle/HOL



```
(* ------------------------------------------------------------------------ *)
subsection "The MIPS TLB"
(* ------------------------------------------------------------------------ *)

text "The MIPS TLB has a fixed number of entries ('capacity') where some of them
      are wired and not replaced by the random replacement."

record MIPSTLB =
  wired     :: nat
  entries   :: "nat ⇒ TLBENTRY"


text "The MIPS TLB has in total 48 entries, of which are at maximum 32 wired."

definition TLBCapacity :: nat
  where "TLBCapacity = 48"
```

- Formal specification of the four TLB operations with pre and post conditions
- Proofs on operations *e.g. commutativity of updates, …*

Refinement of the decoding net

- Defined an lifting function from MIPS TLB to decoding net node.

- Formal proof in Isabelle/HOL that an entry update of a well-formed MIPS TLB is reflected at the map function of the decoding net node
  e.g. write entry e to index i:

$$\begin{tikzcd}
\boxed{\text{node}} \arrow[r, "\texttt{node'= update\_map olde e node}"] & \boxed{\text{node'}} \\
\boxed{\text{TLB}} \arrow[u, "\texttt{convert}"'] \arrow[r, "\texttt{tlb' = tlbwr i e tlb}"'] & \boxed{\text{TLB'}} \arrow[u, "\texttt{convert}"']
\end{tikzcd}$$

Invariant for the MIPS R4600 TLB

"The R4600/R4700 does not provide any detection or shutdown mechanism for multiple matches in the TLB. There is no damage possible from this condition. ***The result is undefined for this condition. Software is expected never to allow this to occur"***

-- MIPS R4600 Manual

An address must never match two entries at the same time.

We say a TLB is valid, if all it's entries are well formed and there is at most one entry that matches a particular VPN

The Invariant cannot be satisfied!

*"The contents of all registers in the CPU are **undefined** when this [reset] exception occurs, except for the following register fields: […]"*

-- MIPS R4600 Manual

- Therefore: at reset, the TLB entries are undefined and considered random.
- TLB is enabled at boot time, cannot be enabled

- Theorem $\exists t. (TLB\_in\_reset\ t) \wedge \neg(TLBValid\ t)$
  A proof that there exists a TLB in reset which is invalid at the same time

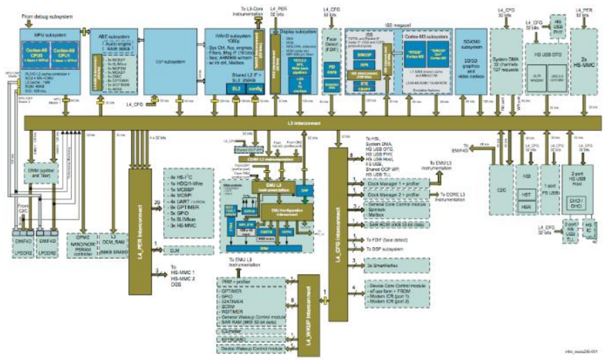
Summary: MIPS TLB Case study

- Formal model of the MIPS R4600 TLB in Isabelle/HOL
- Refinement: from the decoding net node to detailed real hardware
- Invariant cannot be satisfied:
  At reset, the TLB is in an undefined (random) state and may contain two matching entries

github.com/BarrelfishOS
(soon)

Summary

Barrelfish.org



$V_{A9:0}$ **is map** $[20000_3/12$ **to** $P_{A9:0}$ **at** $80000_3]$

$P_{A9:0}, P_{A9:1}$ **are map** $[40138_3/12$ **to** GPT **at** $0]$ **over** $L3$

P_{DSP} **is map** $[\texttt{1d3e}_3/12$ **to** GPT **at** $0]$ **over** $L3$

V_{M3}, V_{M3} **are over** $L1_{M3}$

RAM_{M3} **is accept** $[55020_3/16]$

ROM_{M3} **is accept** $[55000_3/14]$

$V_{A9:1}$ **is map** $[20000_3/12$ **to** $P_{A9:1}$ **at** $80000_3]$

V_{DSP} **is over** P_{DSP}

$L2_{M3}$ **is map** $[0_{30}$ **to** $L3$ **at** $80000_3]$

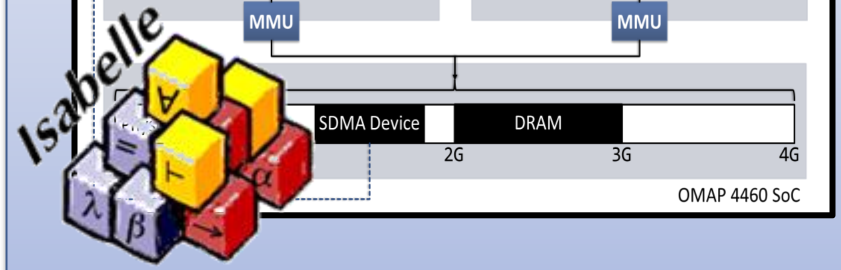
$L1_{M3}$ **is map** $[0_{28}$ **to** $MIF]$

$L4$ **is map** $[49038_3/12$ **to** GPT **at** $0]$

GPT **is accept** $[0/12]$

MIF **is map** $[0 - \texttt{5fffffff}$ **to** $L2_{M3}, 55000_3/14$ **to** $RAM_{M3}, 55020_3/16$ **to** $ROM_{M3}]$

$L3$ **is map** $[49000_3/24$ **to** $L4$ **at** $40100_3, 55000_3/12$ **to** $MIF]$ **accept** $[80000_3/30]$

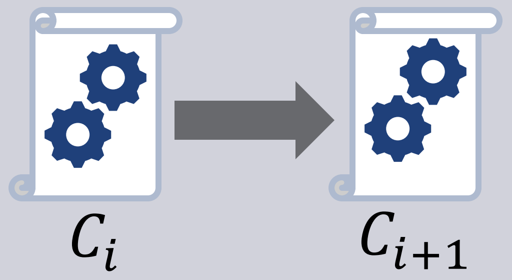


Sockeye Compiler

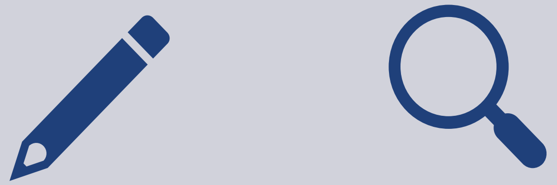
Configuration

Refinement at the example of a software loaded TLB

Future work: towards verified configuration



System (re-)configuration



Distinction of Read/Write accesses

- **Synthesize** configuration procedures with transitions between configurations without violation of **invariants**
- **Modularization** of Sockeye
- Distinguish read and write accesses as they do not have the same semantics, read-only