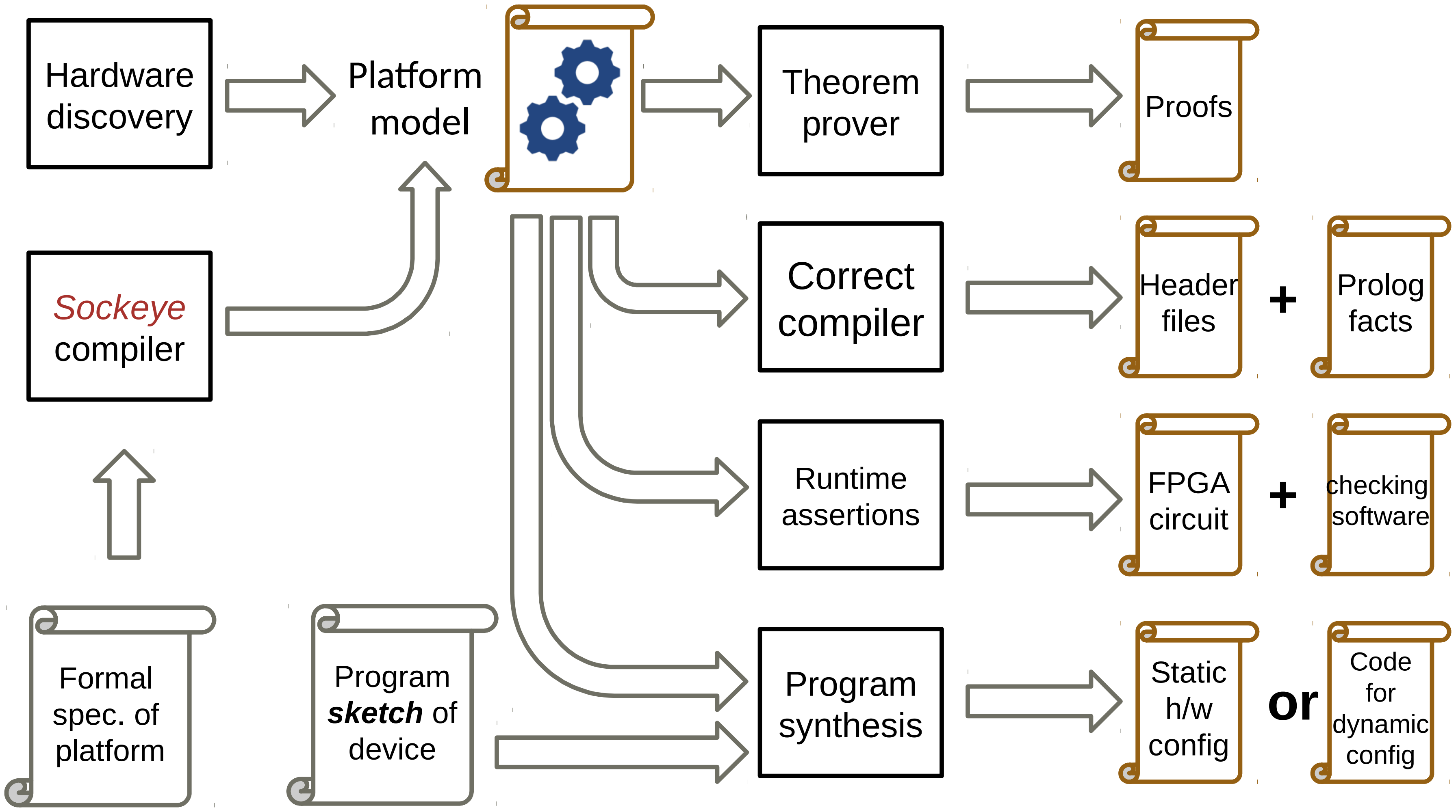
# Modeling the OS/Hardware Interface with Sockeye

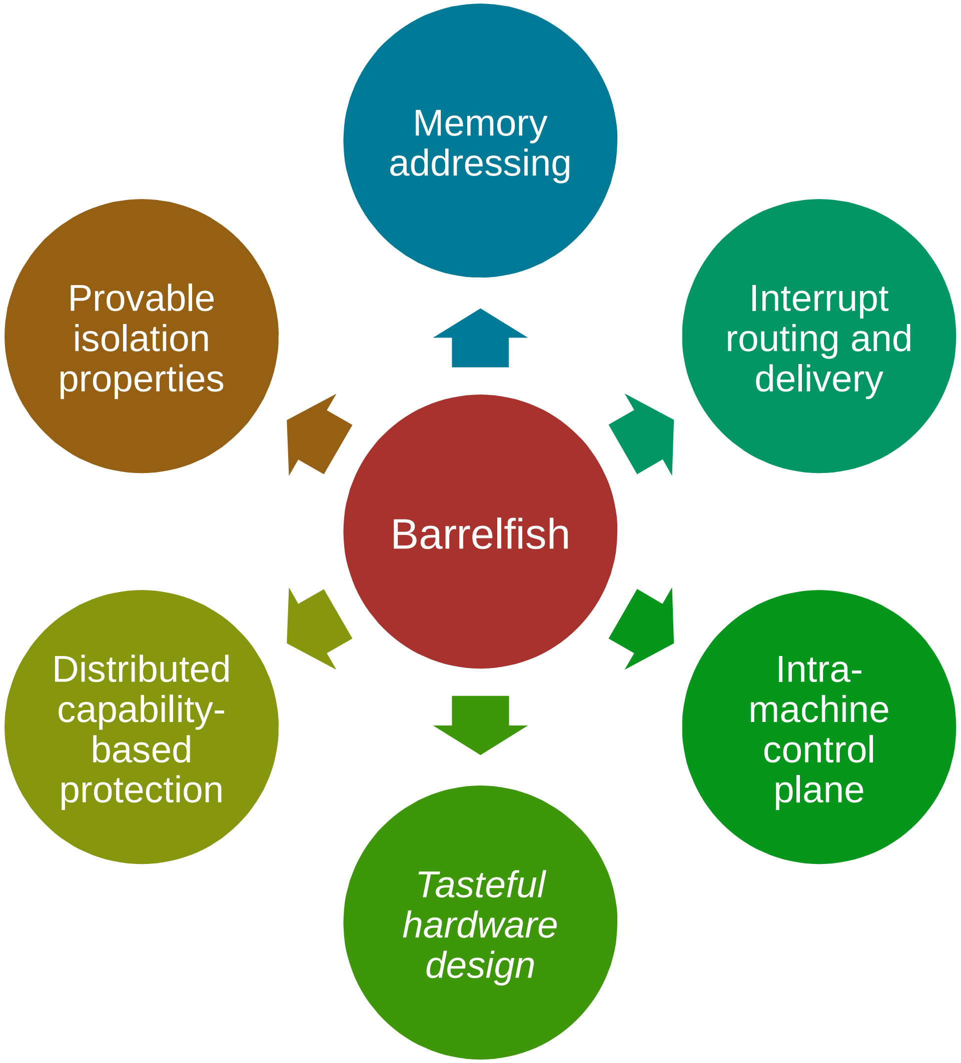
David Cock, ETH Zürich
David.Cock@inf.ethz.ch

# Specifying HW for Systems and Verification



## Overview

- Specification gap between ISA and HDL
- Hardware is ***weird***.
- OS coding & verification have similar needs:
  - What does the hardware do?
  - How do I control it?
  - What do I do when it changes?
- Work in progress


Barrelfish
Memory addressing
Interrupt routing and delivery
Intra-machine control plane
*Tasteful hardware design*
Distributed capability-based protection
Provable isolation properties

David Cock

Reto Achermann

Lukas Humbel

Roni Haecki

Daniel Schwyn

Simon Gerber

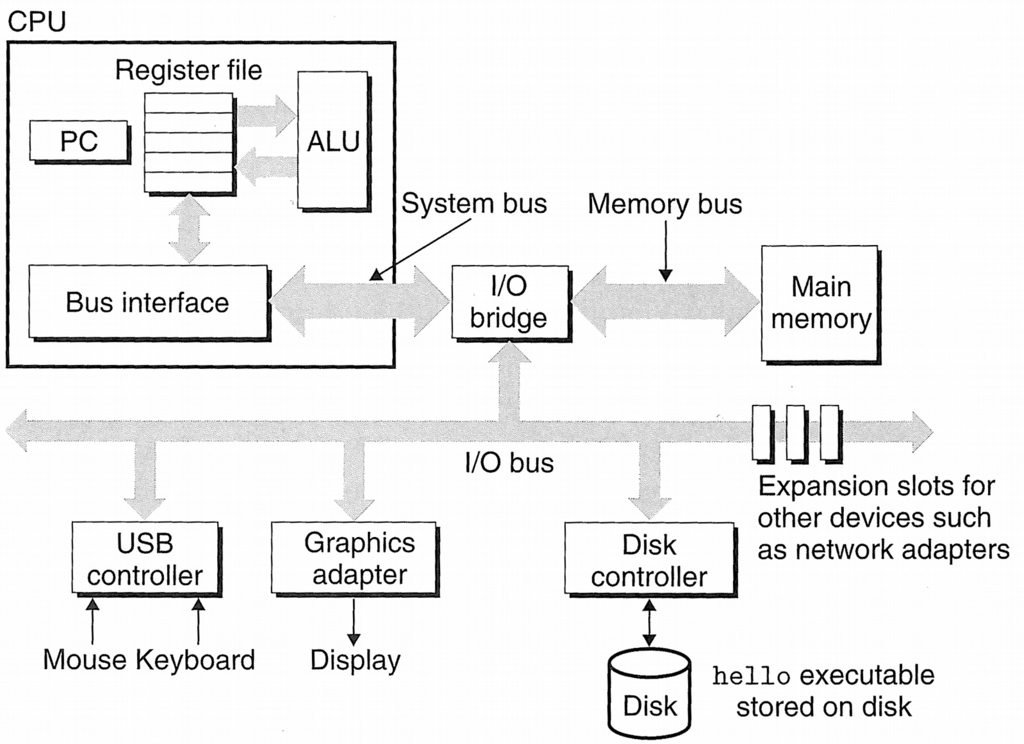


Figure 1.4
**Hardware organization of a typical system.** CPU: Central Processing Unit, ALU: Arithmetic/Logic Unit, PC: Program counter, USB: Universal Serial Bus.

systems, but all systems have a similar look and feel. Don't worry about the complexity of this figure just now. We will get to its various details in stages throughout the course of the book.

**Computer Systems, A Programmer's Perspective, Bryant & O'Hallaron, 2011**

## Why do we need a formal model for memory accesses and interrupts?

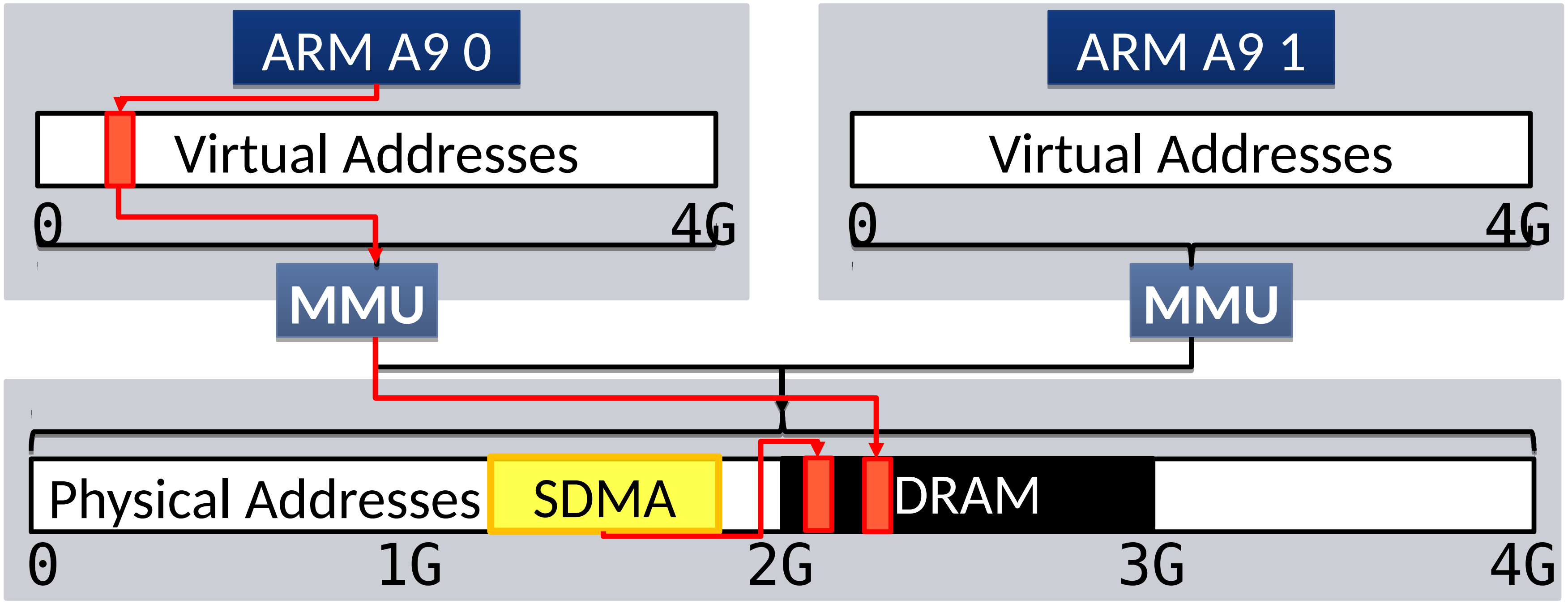
- Abstractions make **incorrect** assumptions

- System software **verification** requires a sound system **hardware model**

- Hardware can be **automatically** configured, given a good enough model.

David Cock
Systems Group, D-INFK, ETH Zürich

# How I Picture a Computer



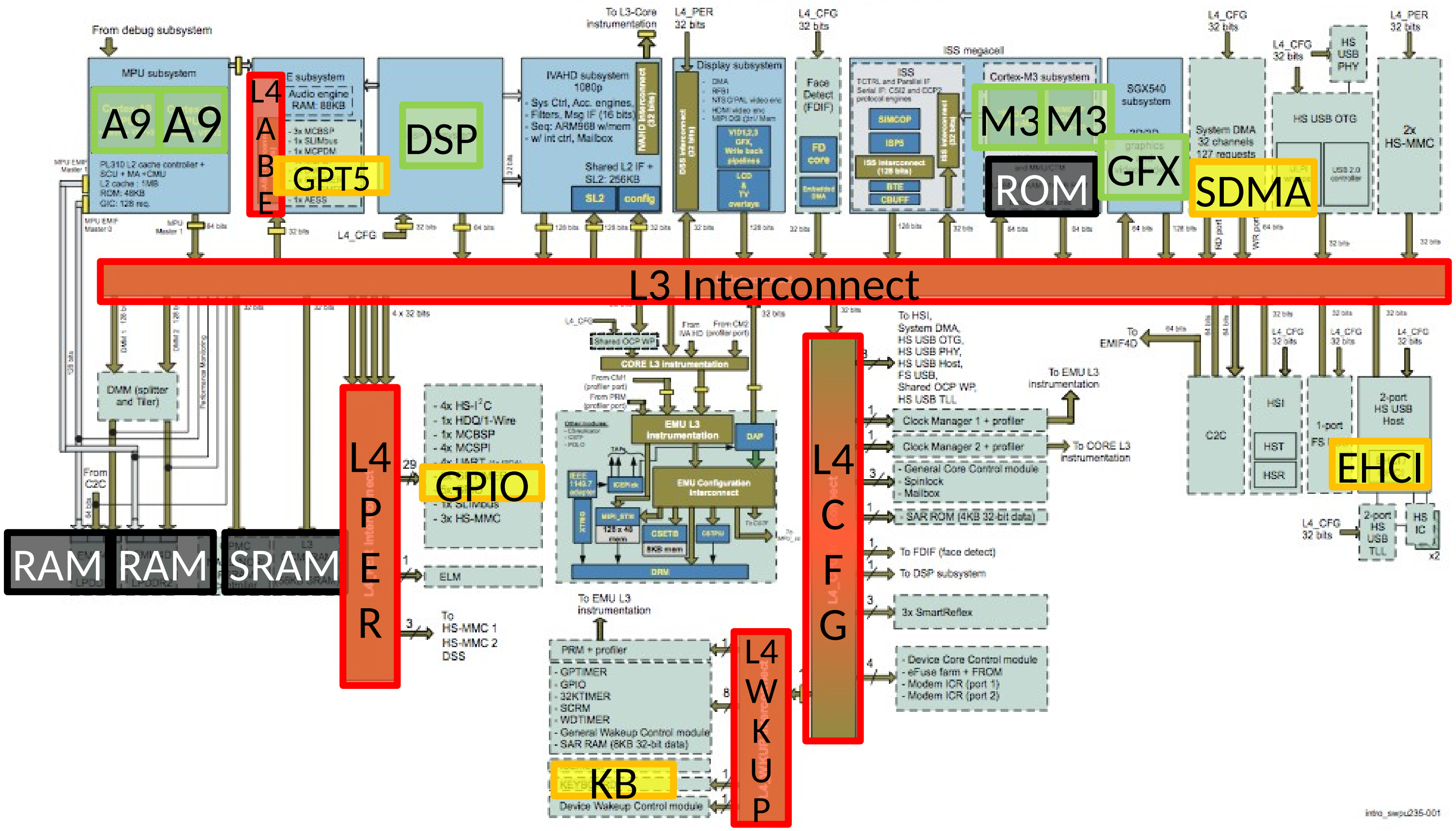
Ti OMAP 4460 SoC

David Cock
Systems Group, D-INFK, ETH Zürich

ETHzürich

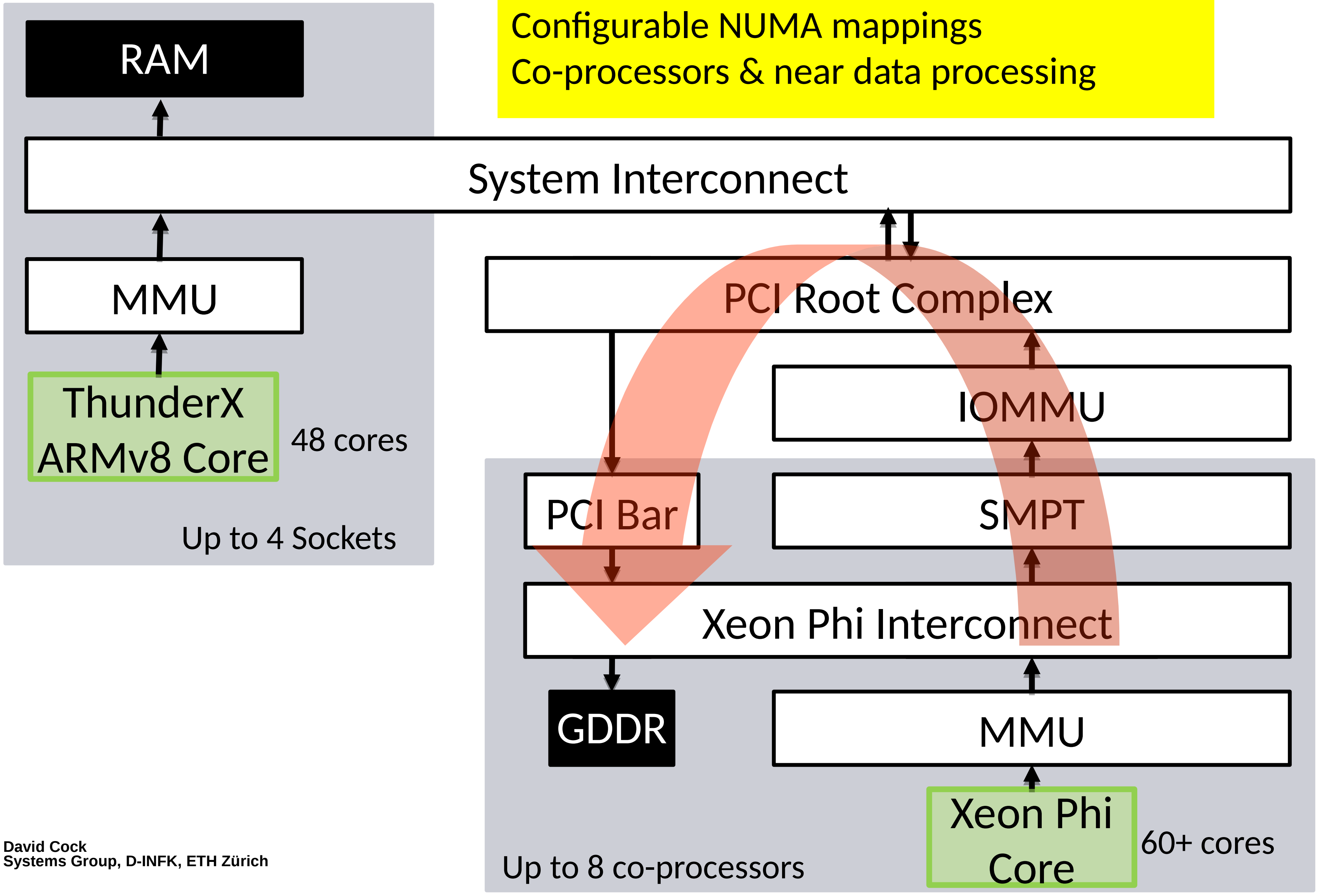
## How the Computer *Actually* Looks

*Your mobile phone… 5-10 years ago!*



ETH zürich

Multiple NUMA nodes & PCI root complexes
Configurable NUMA mappings
Co-processors & near data processing



## What Do We Need in a Model?

1. Faithfulness
   - Don’t hide gory details → abstract *later*.
2. Usability
   - Recover a model that fits in a human brain.
3. Retargetability
   - New hardware, easily.
4. **Surprises!**
   - Don’t hardcode verification conditions.

## What Models Exist?

**Formal**

- ISA Models
  - ARM/HOL, ...
- Weak Memory Models
  - TSO, ARM, Power, …
- …

**Engineering**

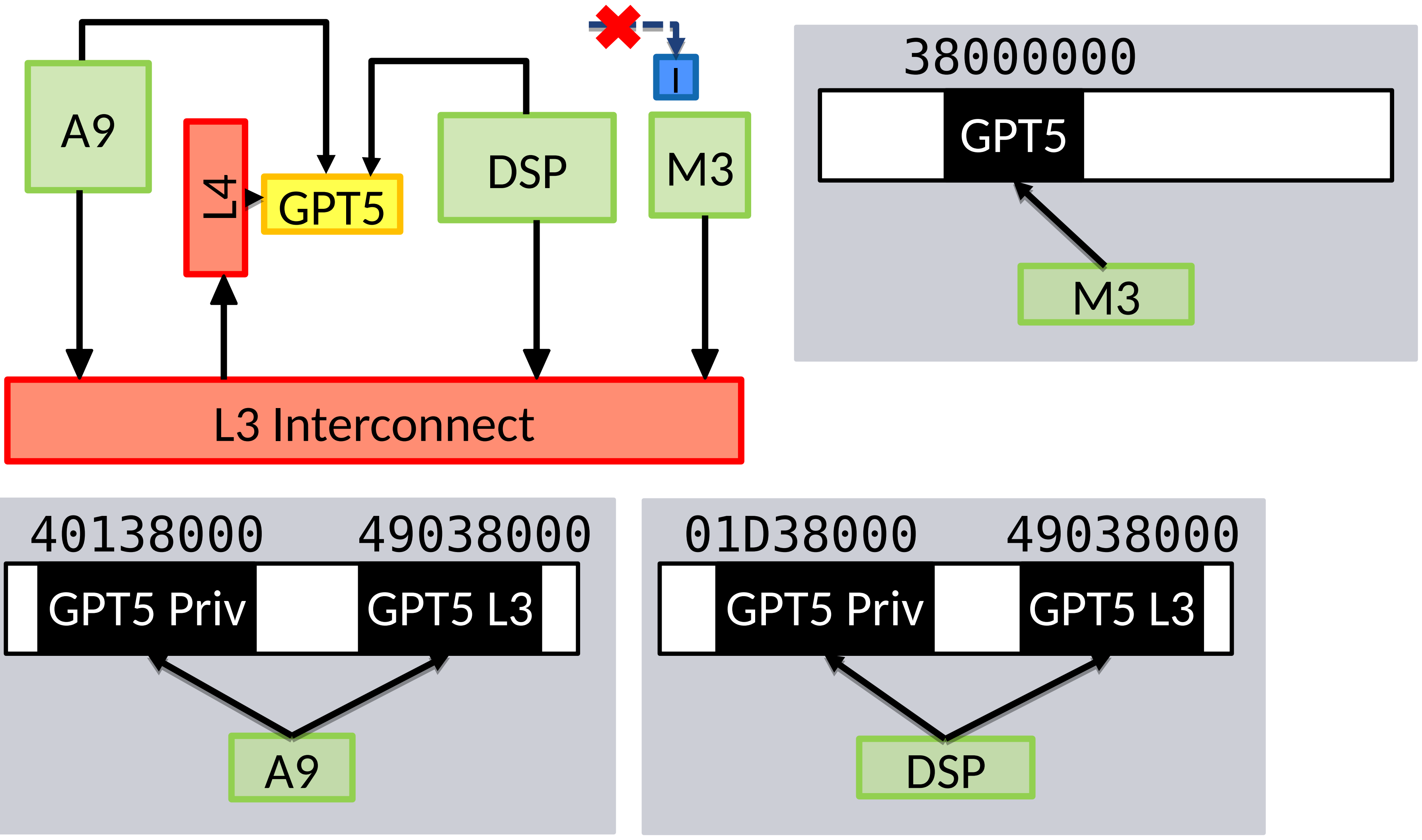
- Device Trees
  - No defined semantics
  - Impose tree structure
  - Single address space
- …

## What About Device Trees?

From arch/arm/boot/dts/omap4.dtsi (Linux 4.14):

```
/*
 * XXX: Use a flat representation of the OMAP4 interconnect.
 * The real OMAP interconnect network is quite complex.
 * Since it will not bring real advantage to represent that in DT for
 * the moment, just use a fake OCP bus entry to represent the whole bus
 * hierarchy.
 */
ocp {
compatible = "ti,omap4-l3-noc", "simple-bus";
#address-cells = <1>;
#size-cells = <1>;
ranges;
```

## There is no Physical Address Space



David Cock
Systems Group, D-INFK, ETH Zürich

ETH zürich

## What About Device Trees?

From arch/arm/boot/dts/omap4.dtsi (Linux 4.14):

```
timer5: timer@40138000 {
        compatible = "ti,omap4430-timer";
        reg = <0x40138000 0x80>,
              <0x49038000 0x80>;
        interrupts = <GIC_SPI 41 IRQ_TYPE_LEVEL_HIGH>;
        ti,hwmods = "timer5";
        ti,timer-dsp;
};
```

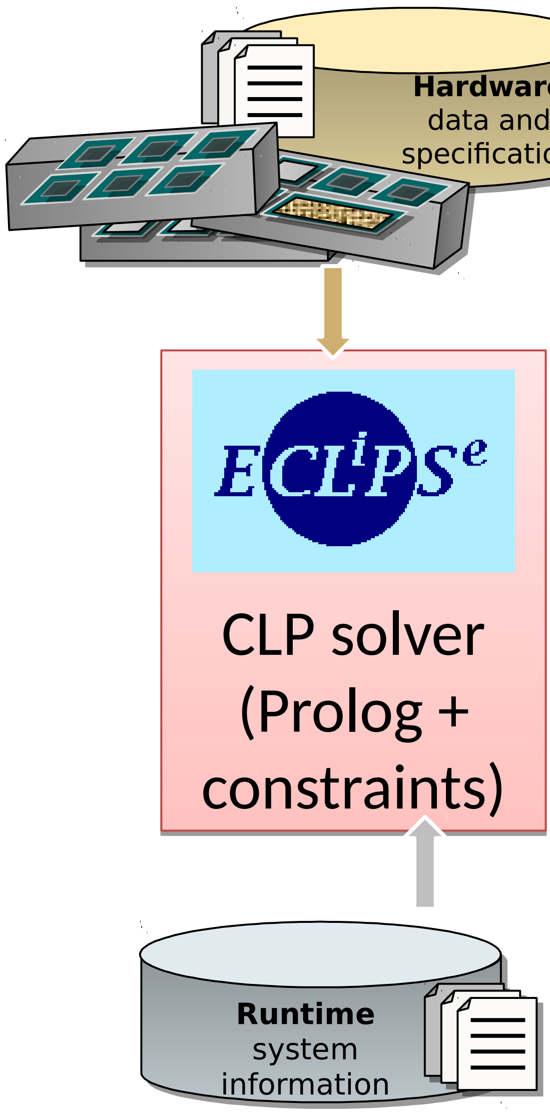
## Application: Barrelfish and the SKB

- seL4-related research OS
- Modern HW (esp. multicore)
- **Automatic configuration**
- DSLs
- Info-/Exo-kernel influence

ETH zürich

## Application: Barrelfish and the SKB

- System Knowledge Base
  - Hardware information
  - Runtime state
- Rich semantic model
  - Represent the hardware
  - Reason about it
  - Separate policy from implementation, and from representation


Hardware data and specification
ECLiPSe
CLP solver (Prolog + constraints)
Runtime system information

## What goes in?

- Hardware resource discovery
  - E.g. PCI enumeration, ACPI, CPUID…
- Online hardware profiling
  - Inter-core all-pairs latency, cache measurements…
- Operating system state
  - Locks, process placement, etc.
- “Things we just know”
  - SoC specs, assertions from data sheets, etc.

CLP solver (Prolog + constraints)

## Current SKB applications

- General name server / service registry
- Coordination service / lock manager
- Device management
  - Driver startup / hotplug
- PCIe bridge configuration
  - A surprisingly hard CSAT problem!
- Intra-machine routing
  - Efficient multicast tree construction
- Cache-aware thread placement
  - Used by e.g. databases for query planning

***And now:***

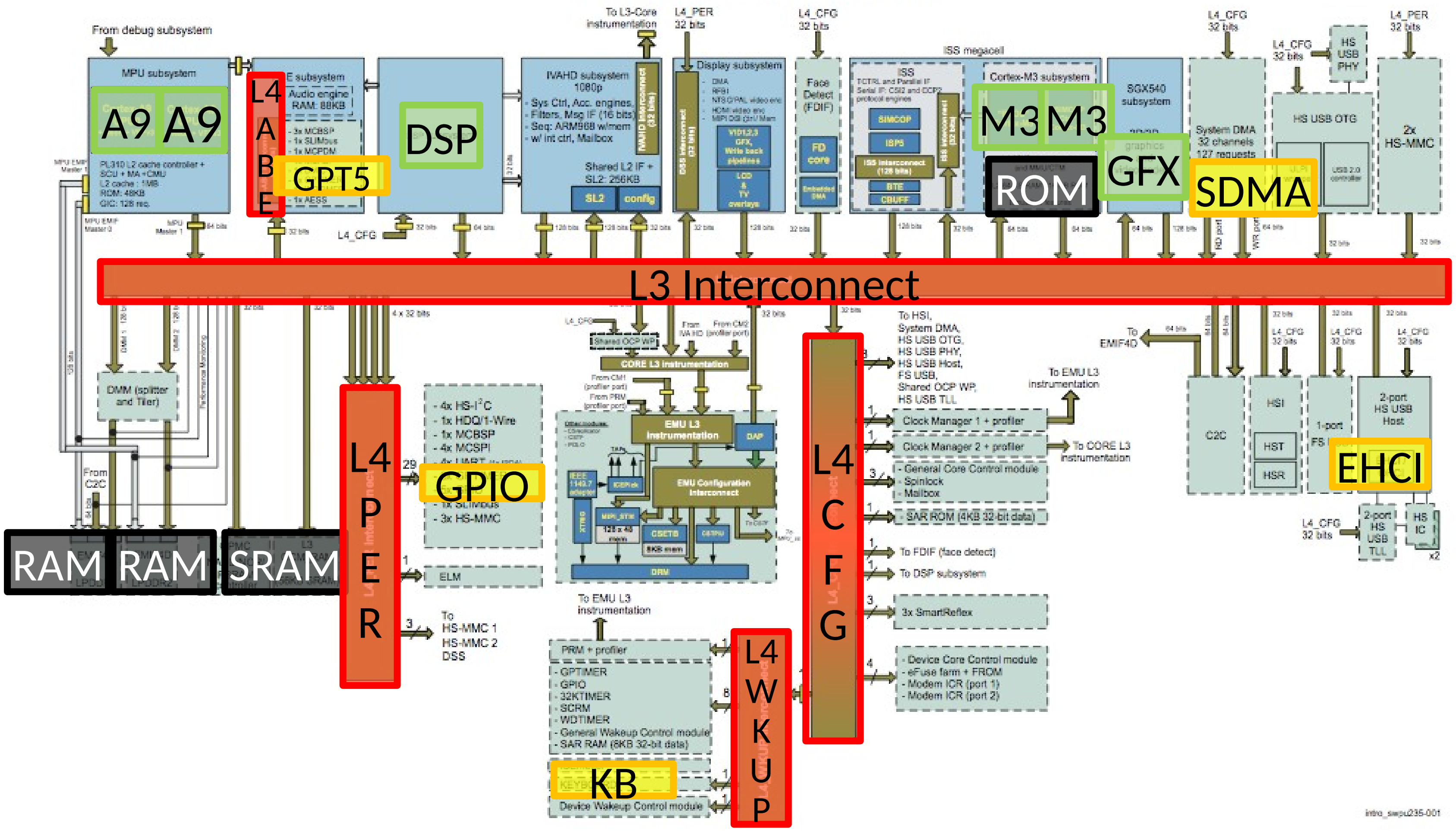
- Teach the SKB about architecture!

CLP solver (Prolog + constraints)

## Decoding Nets

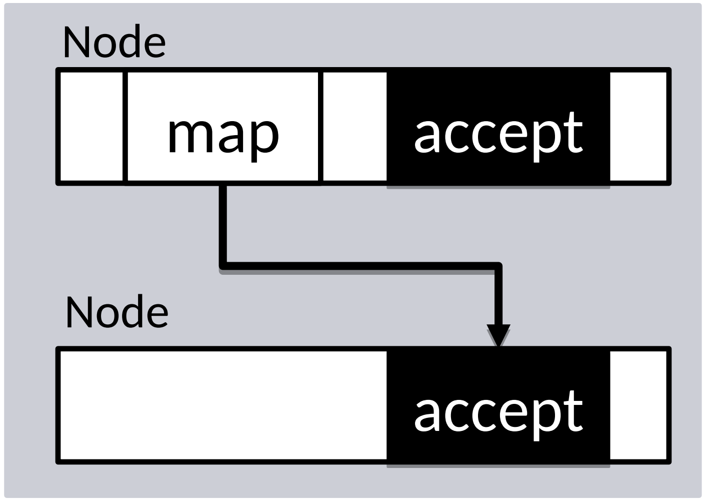
## How the Computer *Actually* Looks

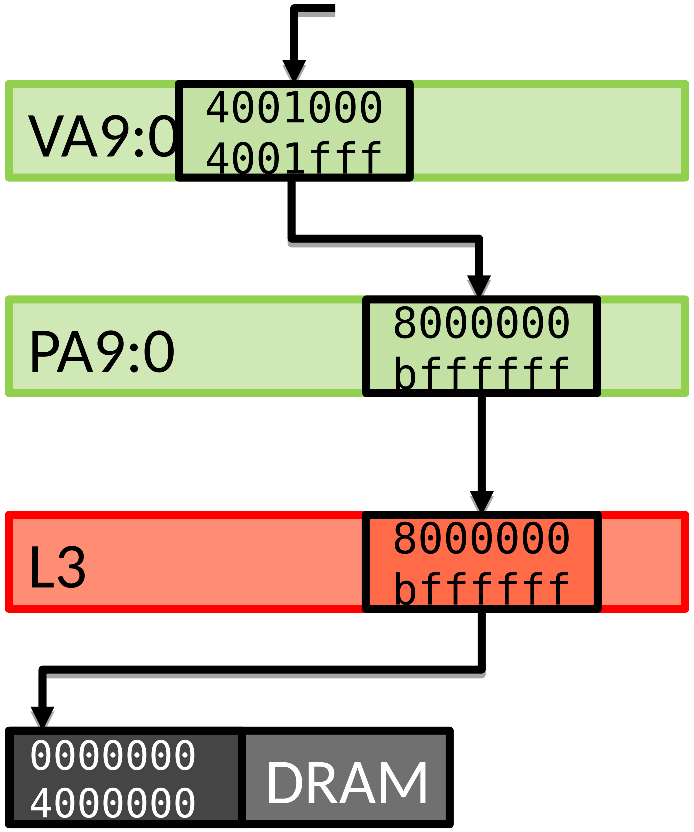
*Your mobile phone... 5-10 years ago!*



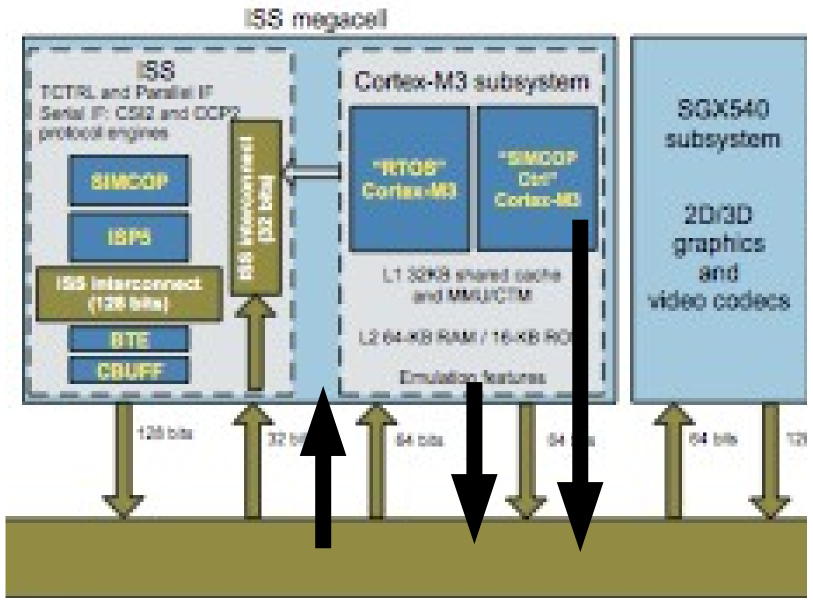
ETHzürich

## Decoding Nets





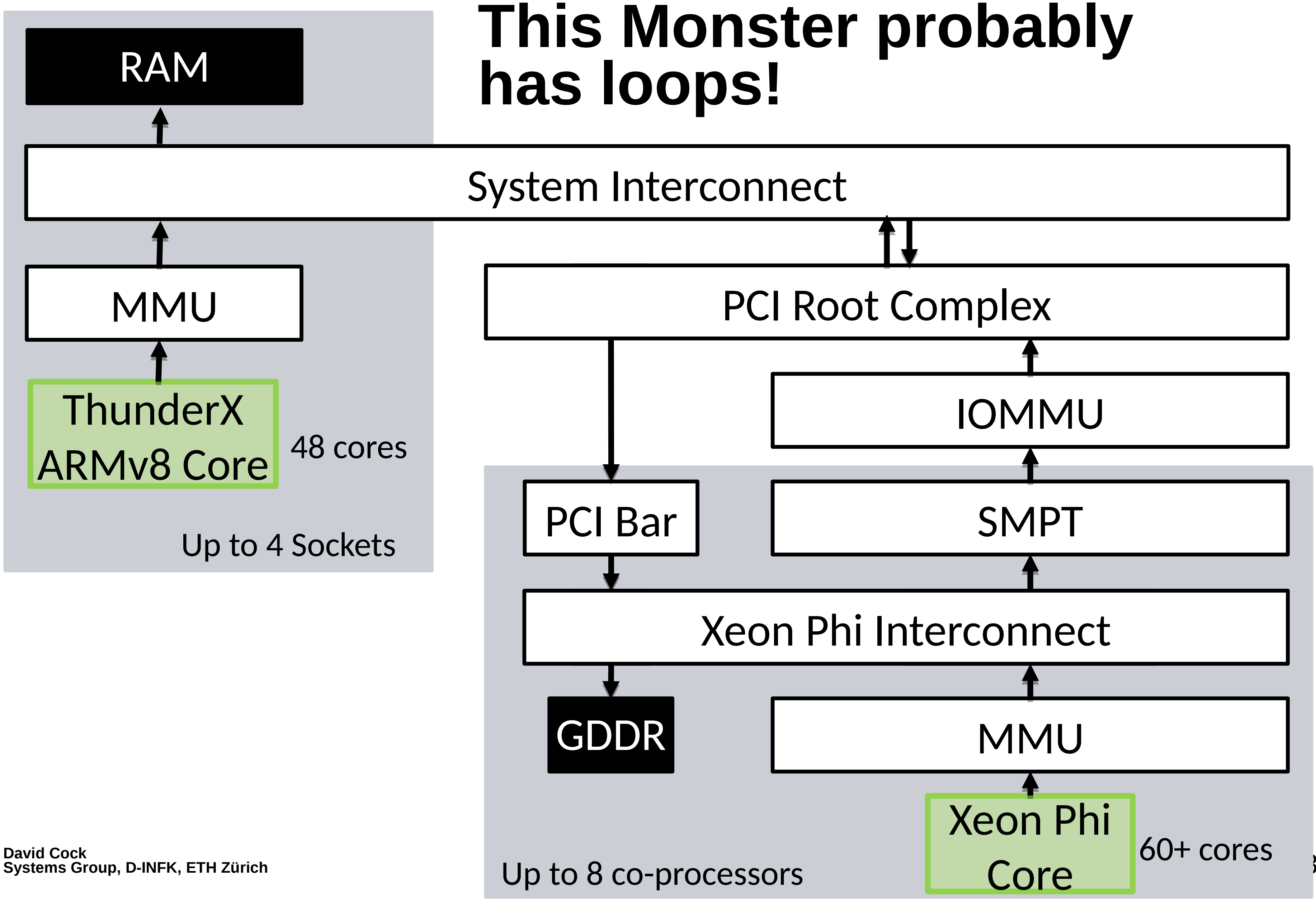
## Loops



- We can hit the L3 interconnect twice!
- Haven‘t (yet) constructed a loop, but we‘re trying.

ETH zürich

# This Monster probably has loops!

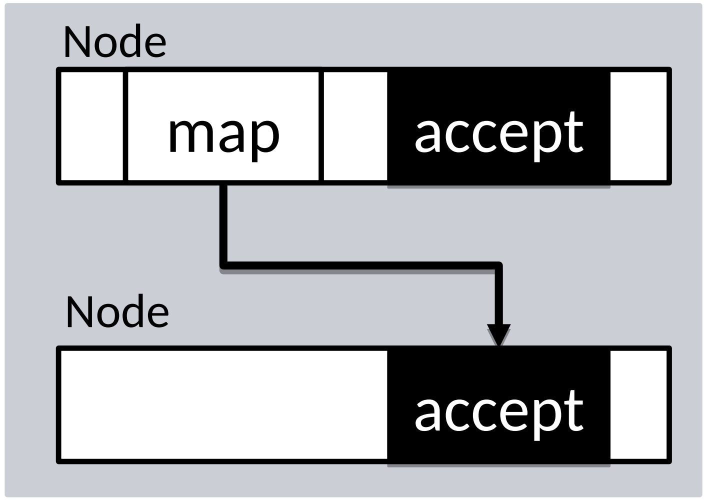


## Modelling memory accesses and interrupts as a decoding net

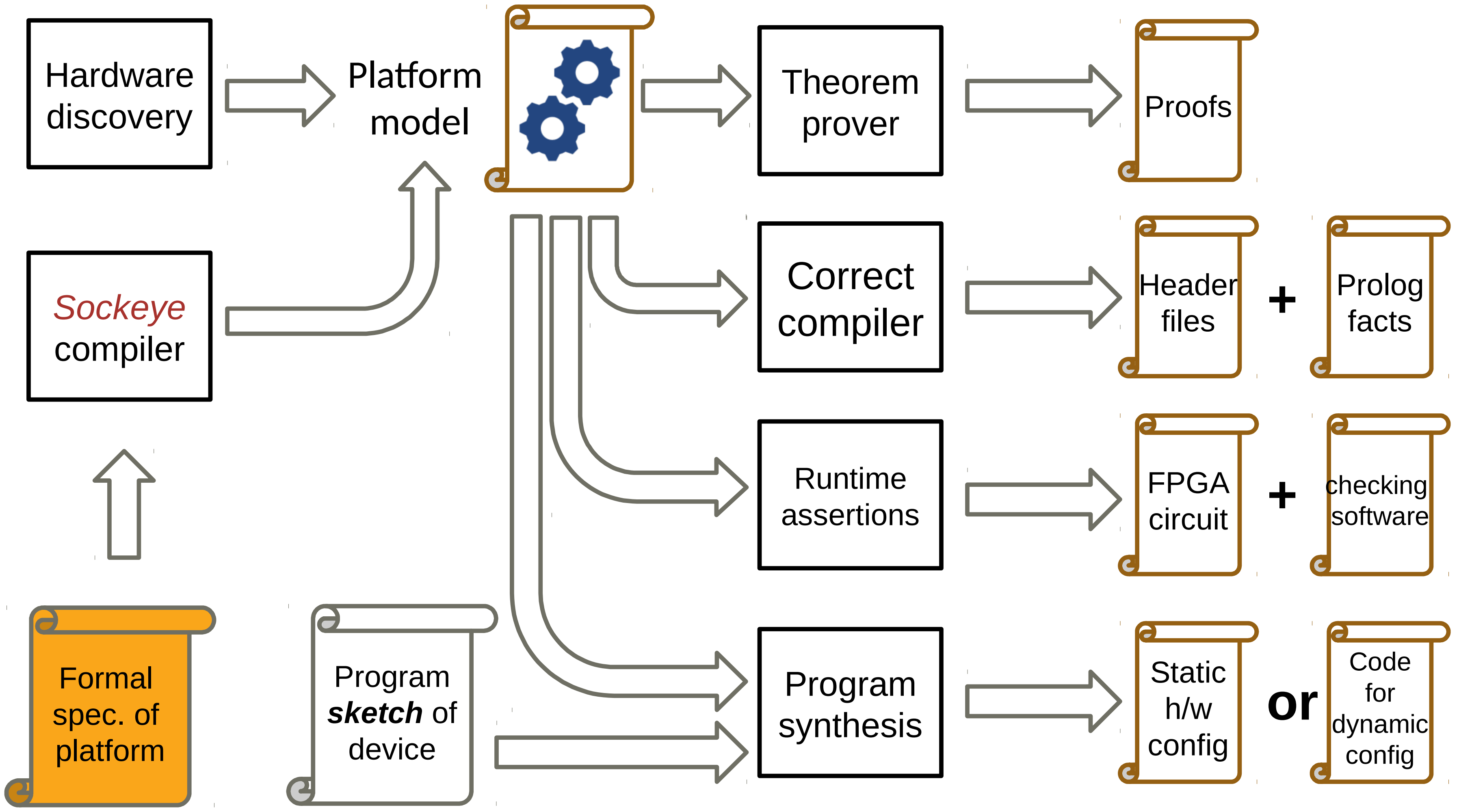
- The model is a **decoding net**, a directed graph

- Address spaces are **nodes**

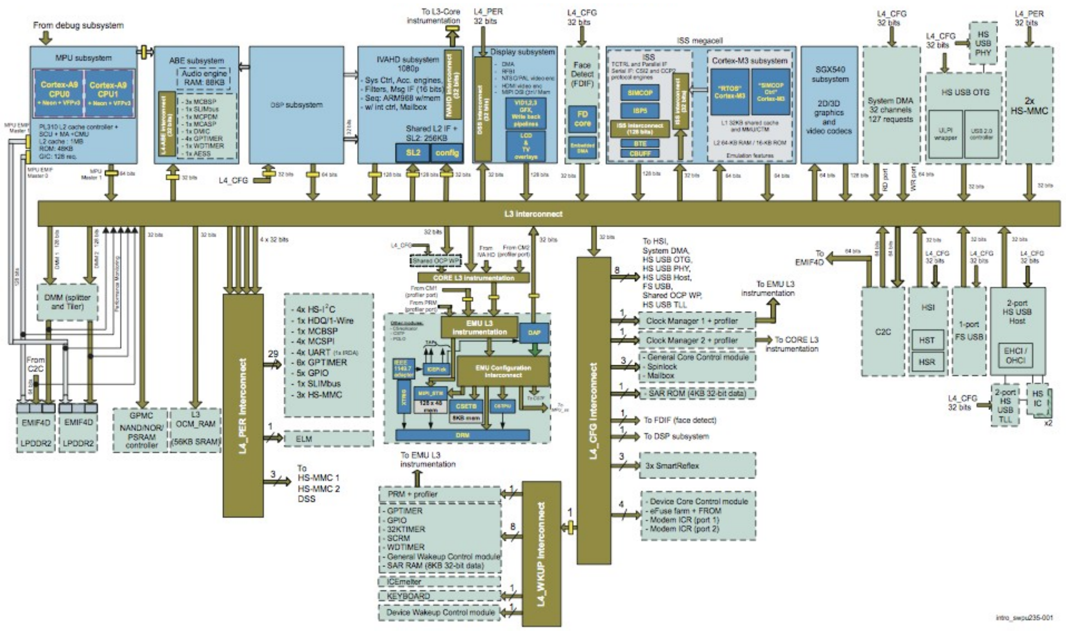
  Address spaces, translation units, devices, interrupt controllers, interconnects, …



David Cock
Systems Group, D-INFK, ETH Zürich


Hardware discovery
Platform model
Theorem prover
Proofs
Sockeye compiler
Correct compiler
Header files
+
Prolog facts
Runtime assertions
FPGA circuit
+
checking software
Formal spec. of platform
Program *sketch* of device
Program synthesis
Static h/w config
or
Code for dynamic config

# The OMAP4460 Decoding Net



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

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

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

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

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

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

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

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

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

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

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

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

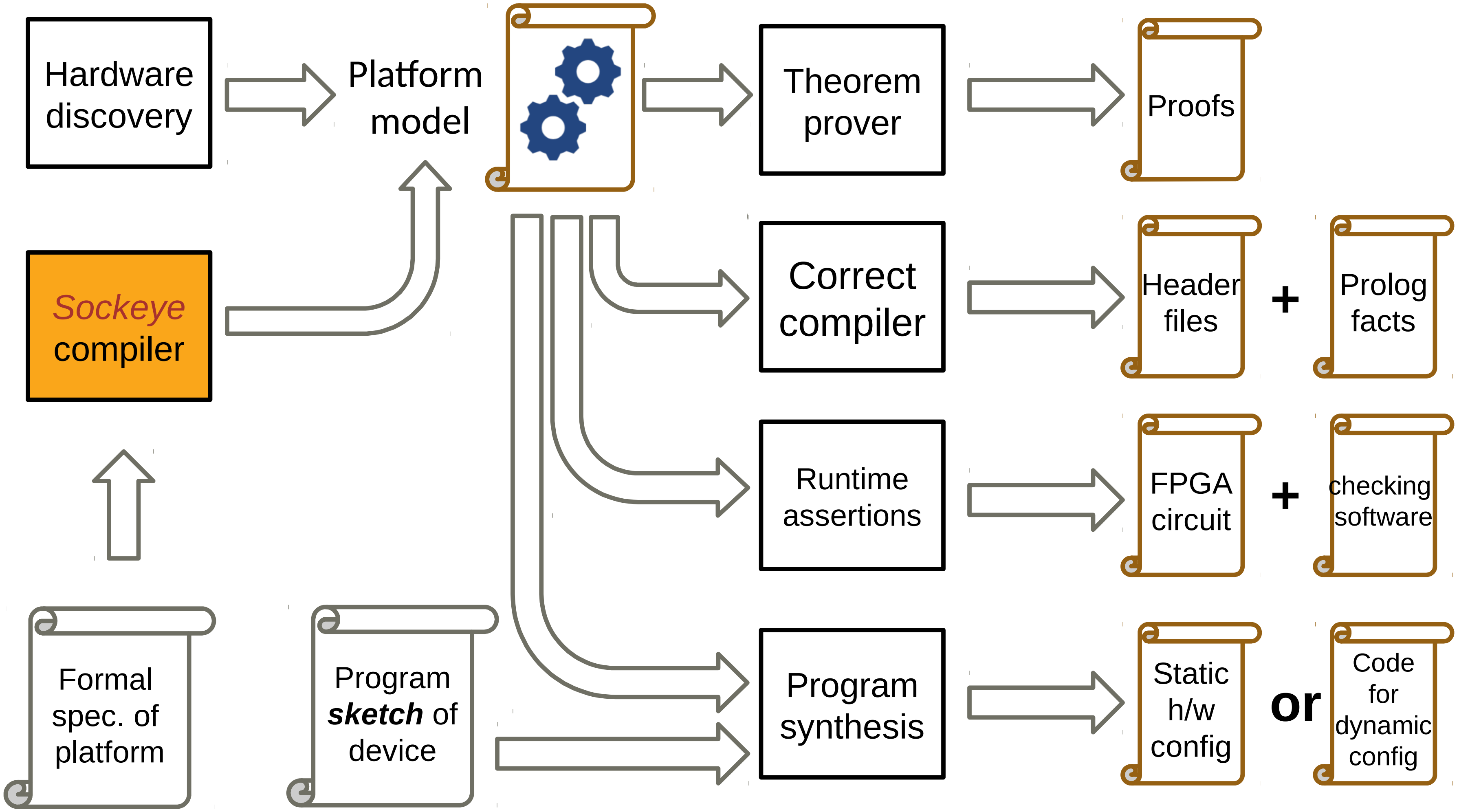
$MIF$ **is map** $[0 - \texttt{5ffffff}$ **to** $L2_{M3}, 55000_3/14$ **to** $RAM_{M3}, 55020_3/16$ **to** $ROM_{M3}]$

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

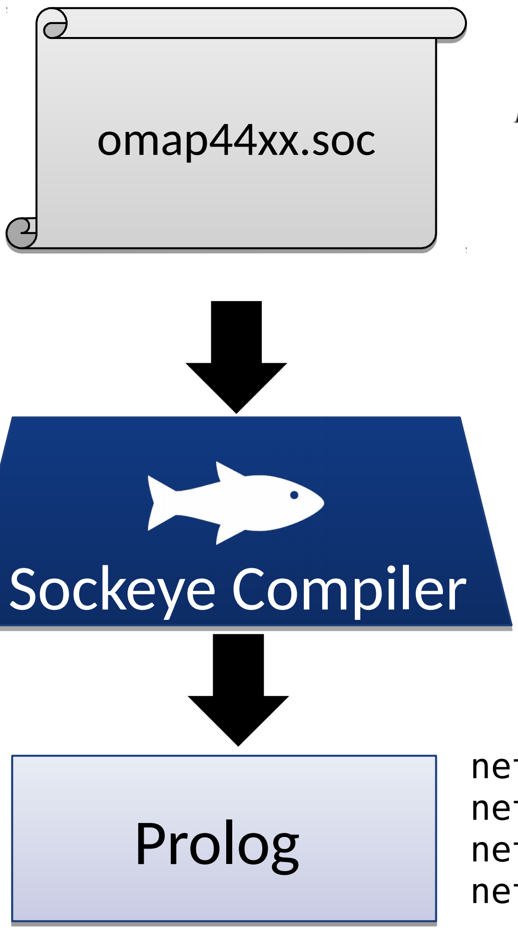
## Applications

## Using the model

- Static Configuration:
  - We now generate the kernel page tables directly from the formal spec.
- Dynamic Discovery and Reconfiguration:
  - The SKB can be populated at runtime – extend the model as hardware is discovered.


Hardware discovery
Platform model
Theorem prover
Proofs
Sockeye compiler
Correct compiler
Header files
+
Prolog facts
Runtime assertions
FPGA circuit
+
checking software
Formal spec. of platform
Program *sketch* of device
Program synthesis
Static h/w config
or
Code for dynamic config

## Sockeye Compiler



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

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

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

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

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

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

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

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

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

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

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

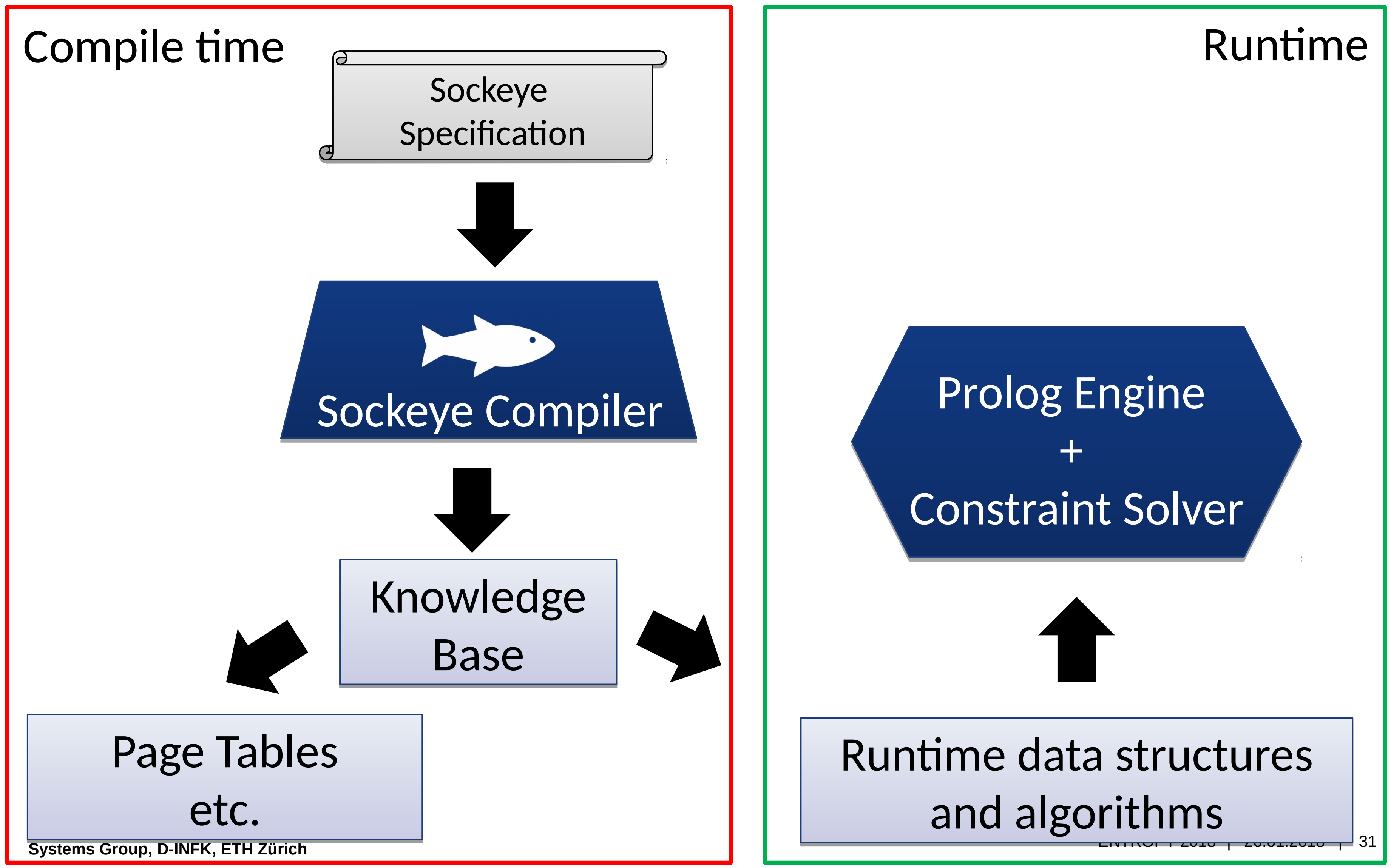
$GPT$ **is accept** $[0/12]$

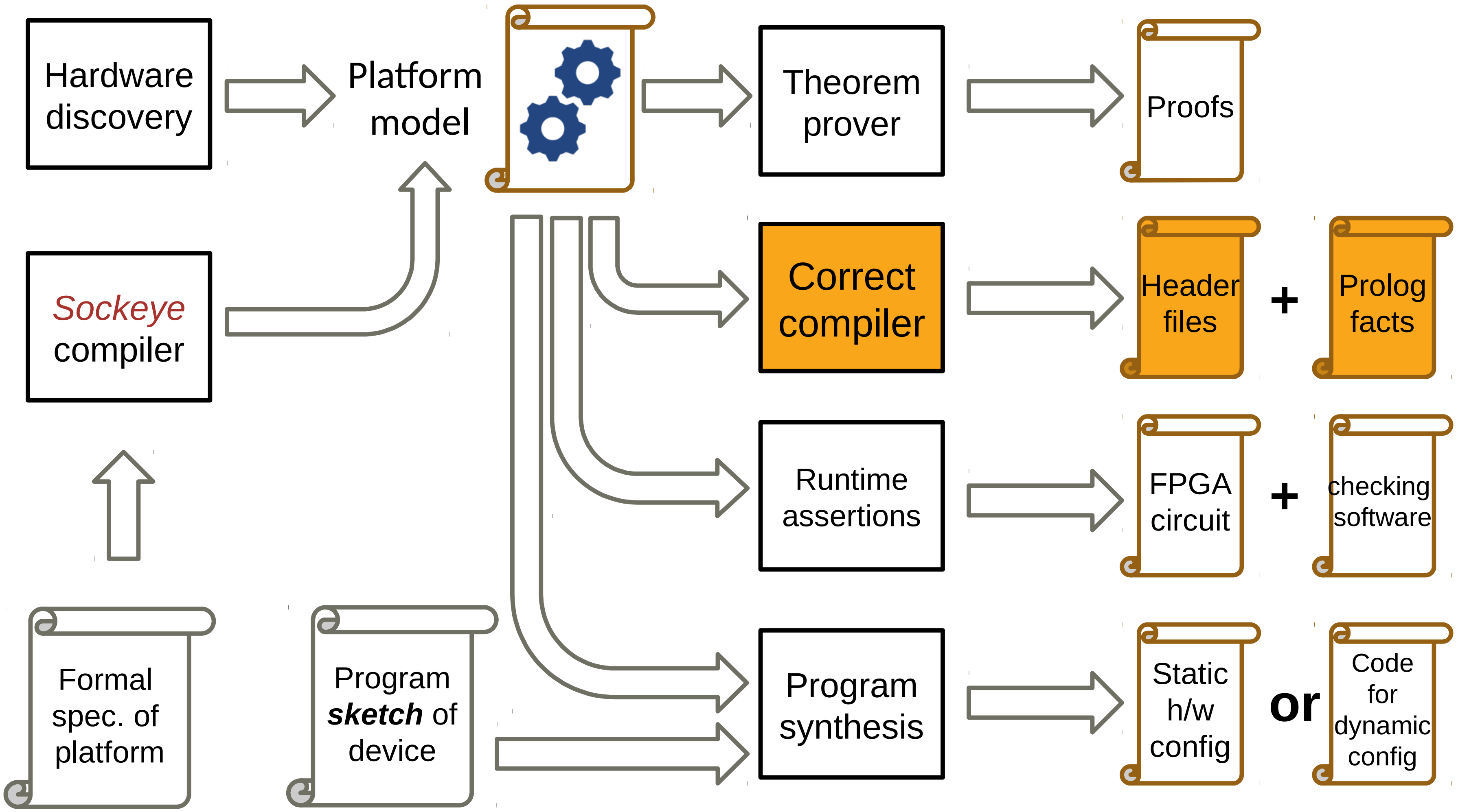
$MIF$ **is map** $[0 - \mathtt{5ffffff}$ **to** $L2_{M3}, 55000_3/14$ **to** $RAM_{M3}, 55020_3/16$ **to** $ROM_{M3}]$

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

```
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 Workflow




Hardware discovery
Platform model
Theorem prover
Proofs
Sockeye compiler
Correct compiler
Header files
+
Prolog facts
Runtime assertions
FPGA circuit
+
checking software
Formal spec. of platform
Program sketch of device
Program synthesis
Static h/w config
or
Code for dynamic config

ETH zürich

## Page Table Generation

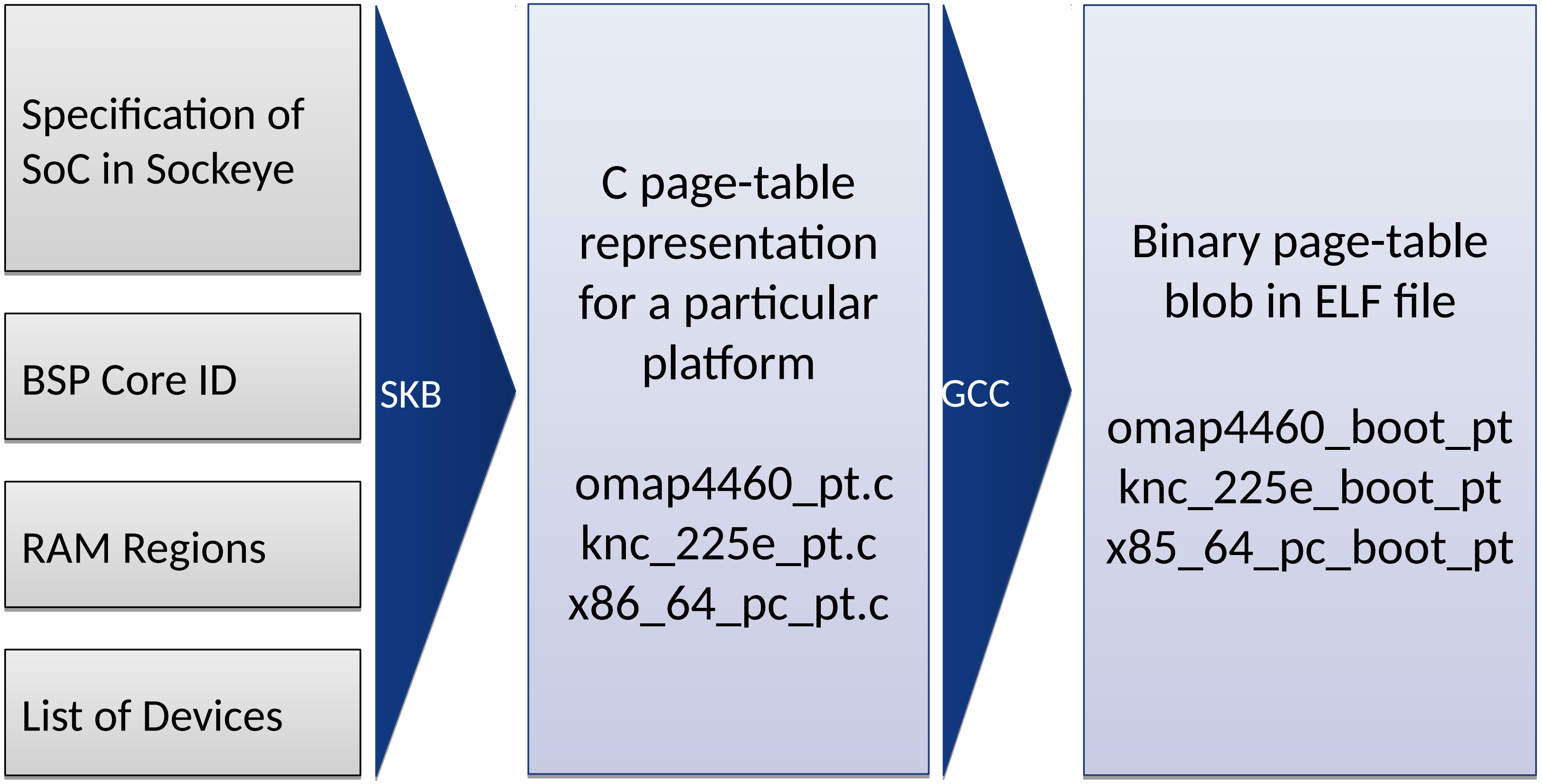
```
 bootDriver {
     target = "omap44xx",
     architectures = [ "armv7" ],
     cFiles = [ … ]
     kernelPageTable = pageTableWith
{
         cpu = "CORTEXA9",
         mainMemory = "SDRAM",
         devices = [
           "UART3",
           "SCU",
            …]
     },
},
```

Specify the observer

Memory

Required devices

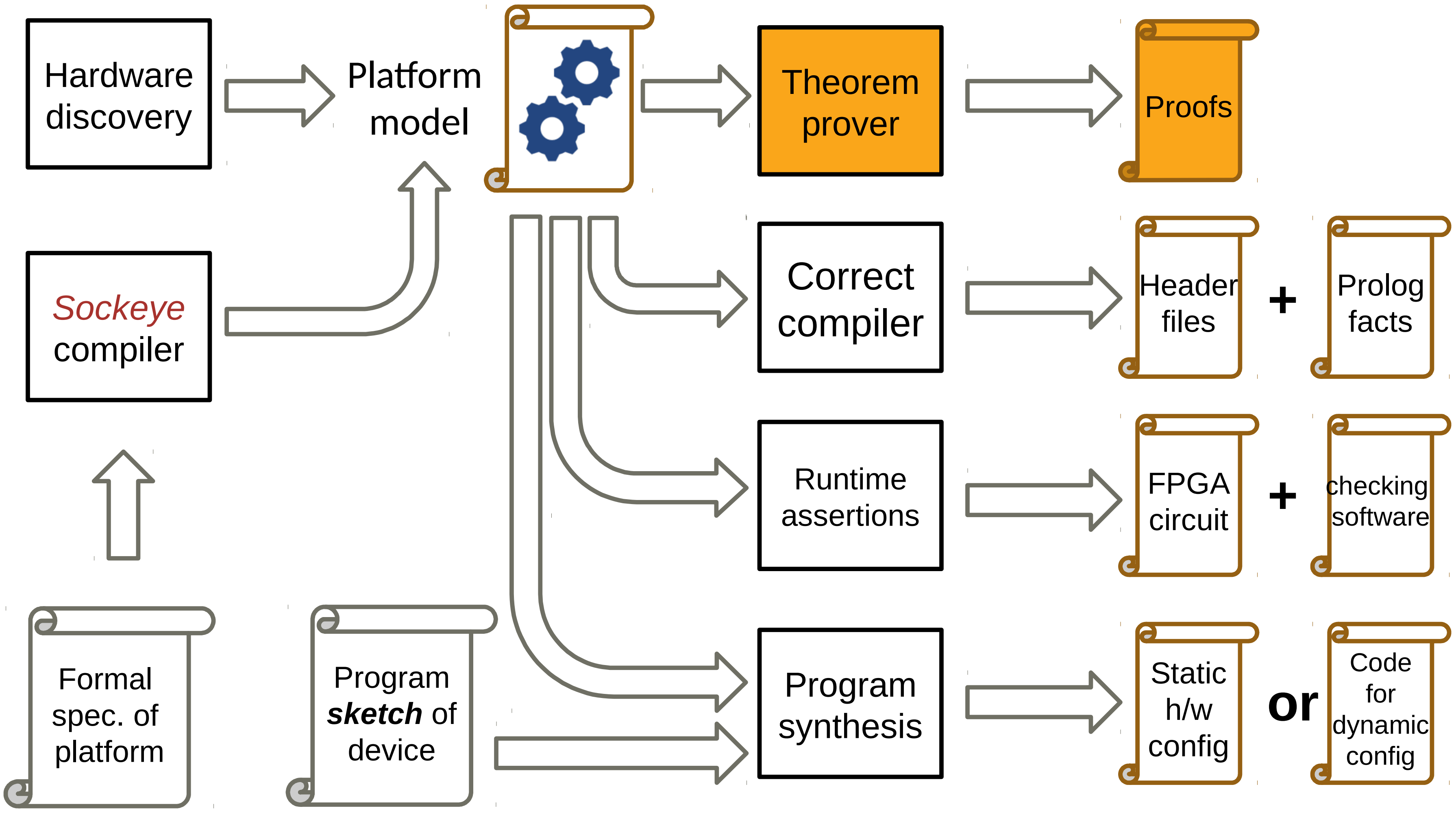
## Application: Generation of Kernel Page Tables



## A Generated Page Table

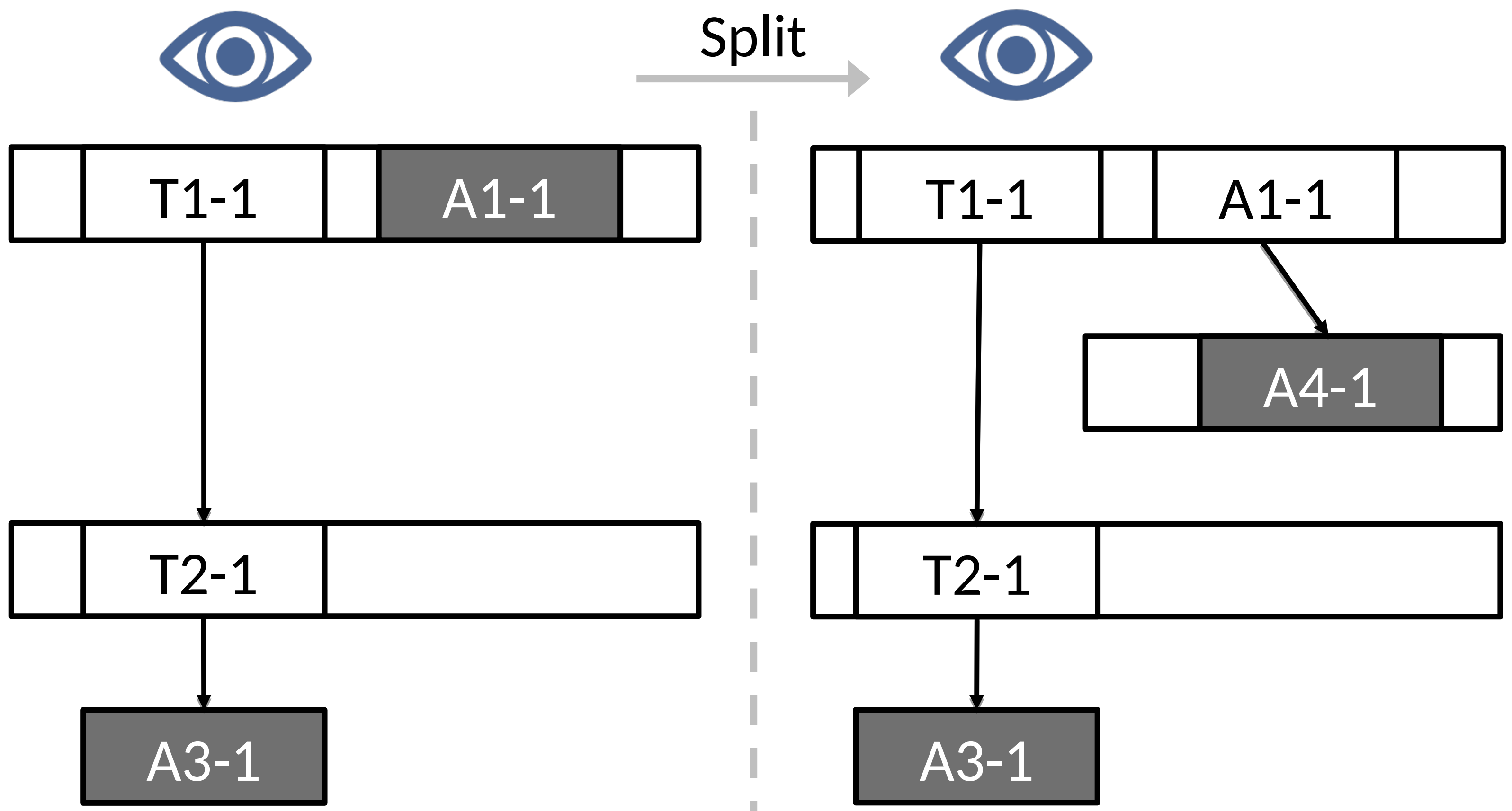
```
union arm_l1_entry l1_table[ARM_L1_MAX_ENTRIES]
    __attribute__((aligned(ARM_L1_ALIGN),
section(".boot.tables"))) =
{
    [L1_TABLE_INDEX(0x7FE00000)] = L1_DEVICE_ENTRY(0x7FE00000),
    [L1_TABLE_INDEX(0x7FF00000)] = L1_DEVICE_ENTRY(0x7FF00000),
….
    [L1_TABLE_INDEX(0x80000000)] = L1_MEMORY_ENTRY(0x80000000),
    [L1_TABLE_INDEX(0x80100000)] = L1_MEMORY_ENTRY(0x80100000),
    [L1_TABLE_INDEX(0x80200000)] = L1_MEMORY_ENTRY(0x80200000),
    [L1_TABLE_INDEX(0x80300000)] = L1_MEMORY_ENTRY(0x80300000),
…
}
```

## Formal Results


Hardware discovery
Platform model
Theorem prover
Proofs
*Sockeye* compiler
Correct compiler
Header files
+
Prolog facts
Runtime assertions
FPGA circuit
+
checking software
Formal spec. of platform
Program ***sketch*** of device
Program synthesis
Static h/w config
or
Code for dynamic config

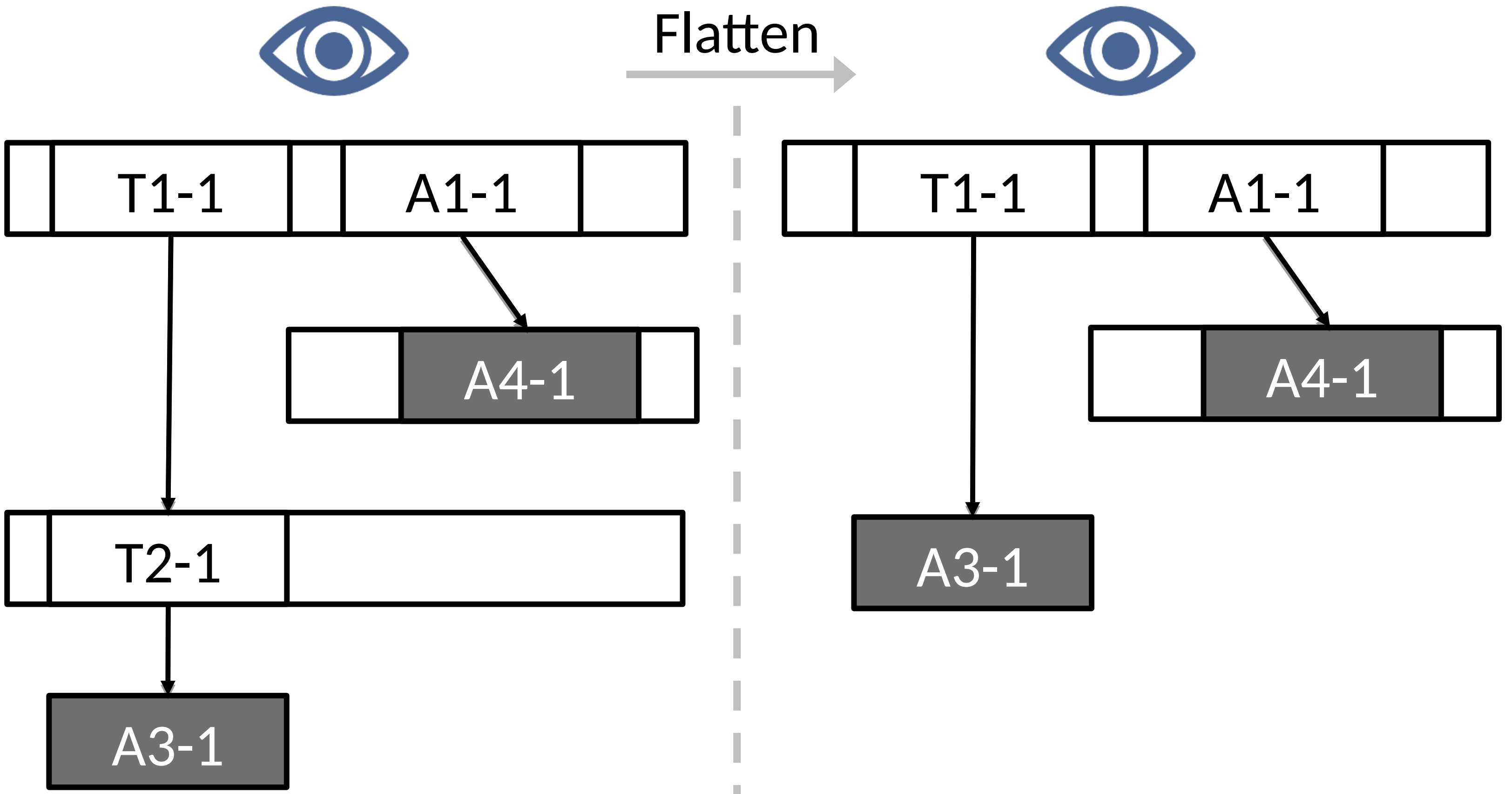
ETH zürich

## Flattening – Step 1: Split



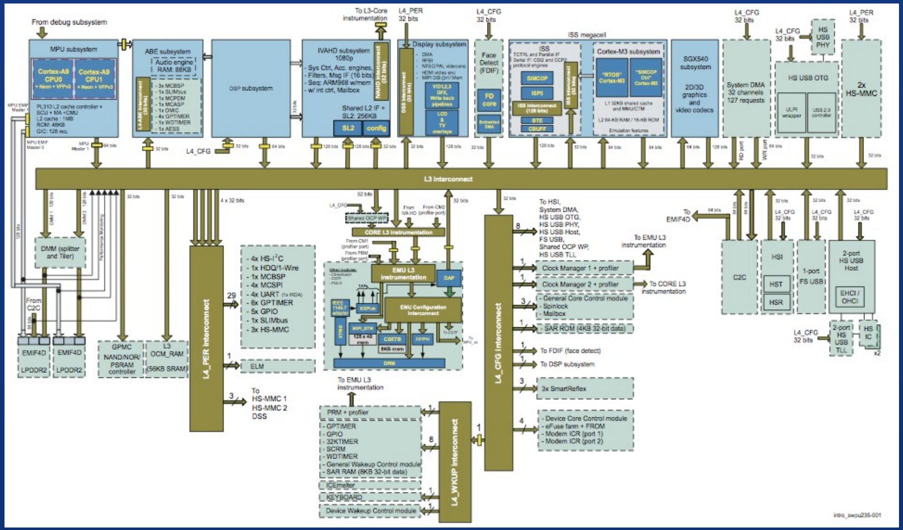
ETH zürich

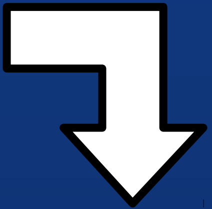
## Flattening – Step 2: Flatten



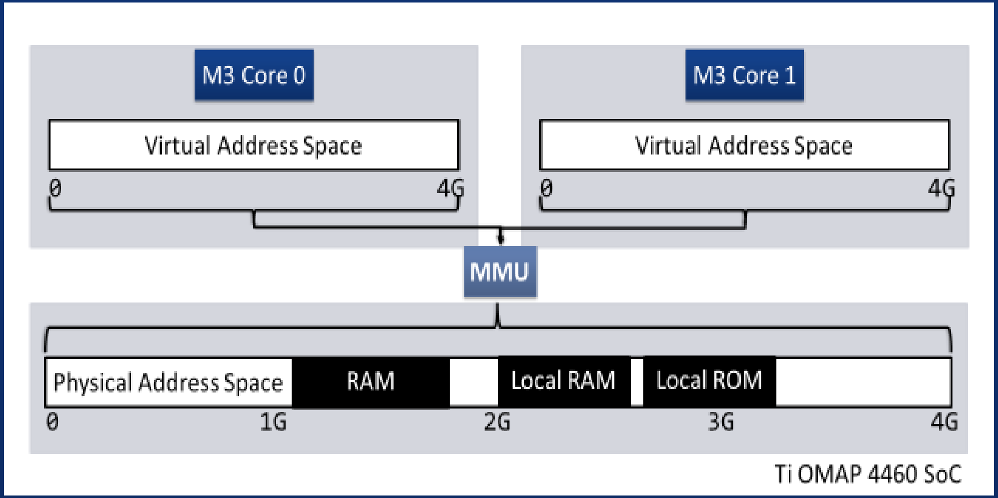
ETH zürich

## Recovering the Simple View





Any given observer has an equivalent flat view.



## Where Are We Going with This?

- Barrelfish
  - All HW description and configuration is migrating to Sockeye.
  - More generated code, less written.
- Models
  - Validation with runtime verification.
  - Verified compilation.
  - Power and Clock networks.