Open CPU / SoC design, all the way up to Debian

Background / whoami

- Dolu1990 on github

- Active on open/free project

  - SpinalHDL / VexRiscv / NaxRiscv

- Software / Hardware background

  - Without computer science degree

  - With Industrial system / Electronic degree

Introduction

- This talk will be based on NaxRiscv

    - Opensource https://github.com/SpinalHDL/NaxRiscv
        - RISC-V softcore
        - out of order, multi-issue, multi core, memory coherent
- What the talk will do

    - Share experience / flow
    - Give tips / keywords

Hardware description
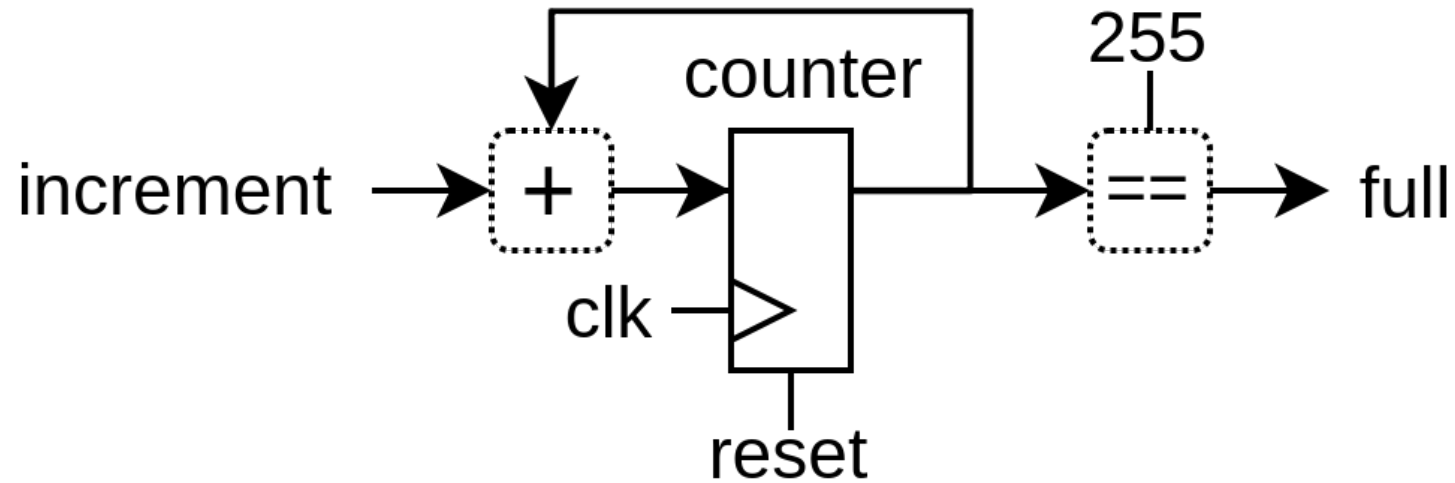
- "There is no alternative" (VHDL / Verilog)

  - Scala based : SpinalHDL, Chisel

  - Python based : Migen, Amaranth

  - …

It can be pretty / explicit
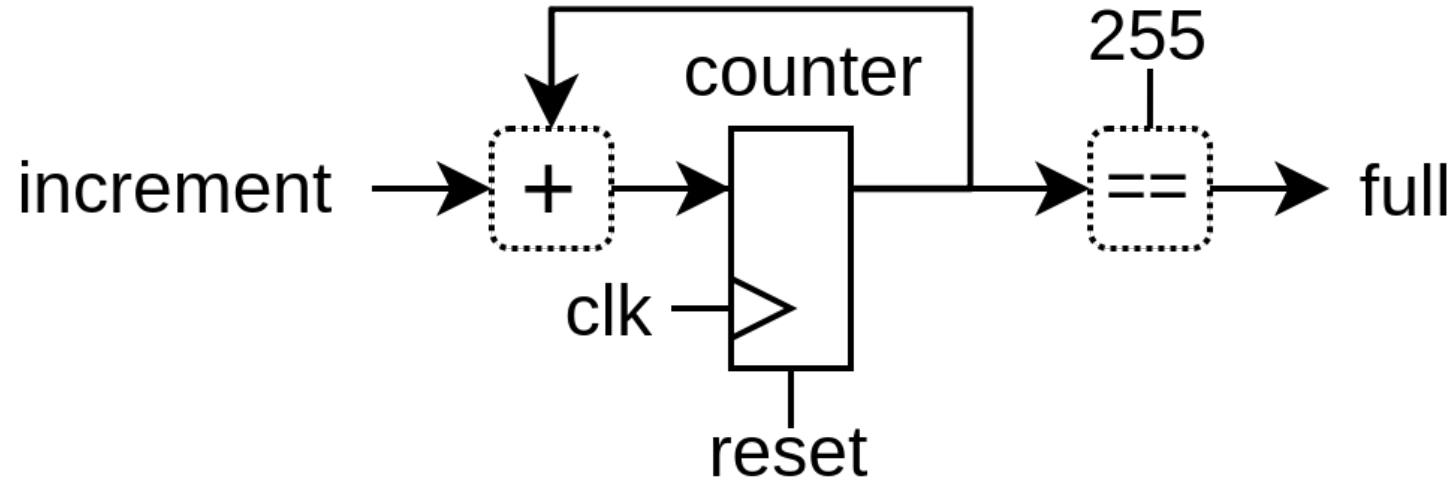

```
class Timer extends Component {
  val increment = in Bool()
  val counter = Reg(UInt(8 bits)) init(0)
  val full = counter === 255

  when(increment){
    counter := counter + 1
  }
}
```

Hardware description API
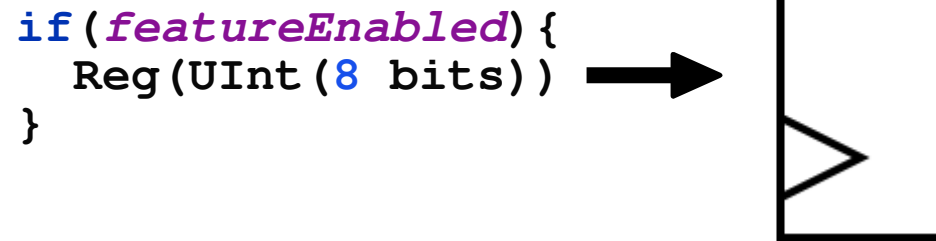
```
if(featureEnabled){
    Reg(UInt(8 bits))
}
```

- Control flow : **if** / for

- Data structures : dynamic array / hash map / hash set

- Lambda function : reduce / fold / map / filter / …

- OOP : class / software interface

- See https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/abstraction/index.html

Hardware description API

```
for(i <- 0 to 2){
    Reg(UInt(8 bits))
}
```

- Control flow : if / **for**

- Data structures : dynamic array / hash map / hash set

- Lambda function : reduce / fold / map / filter / …

- OOP : class / software interface

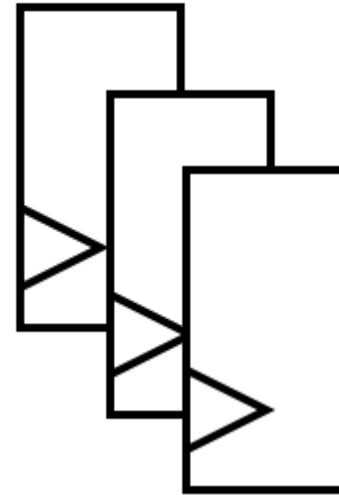- See https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/abstraction/index.html

Hardware description API

```scala
val flushes = ArrayBuffer[Flush]()
flushes += new Flush()
flushes += new Flush()
if(featureEnabled){
    flushes += new Flush()
}
```

Index	Value
0	hardware
1	hardware
2	hardware

- Data structures : **dynamic array** / hash map / hash set

- Lambda function : reduce / fold / map / filter / …

- OOP : class / software interface

- See https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/abstraction/index.html

```
val PC = Hardtype(UInt(32 bits))
val instruction = Hardtype(UInt(32 bits))
val fetch, decode, execute = HashMap[Hardtype, Hardware]()
...
fetch(PC) = U(42)
...
val target = execute(PC) + U(666)
...
autoPipeline(fetch, decode, execute)
```
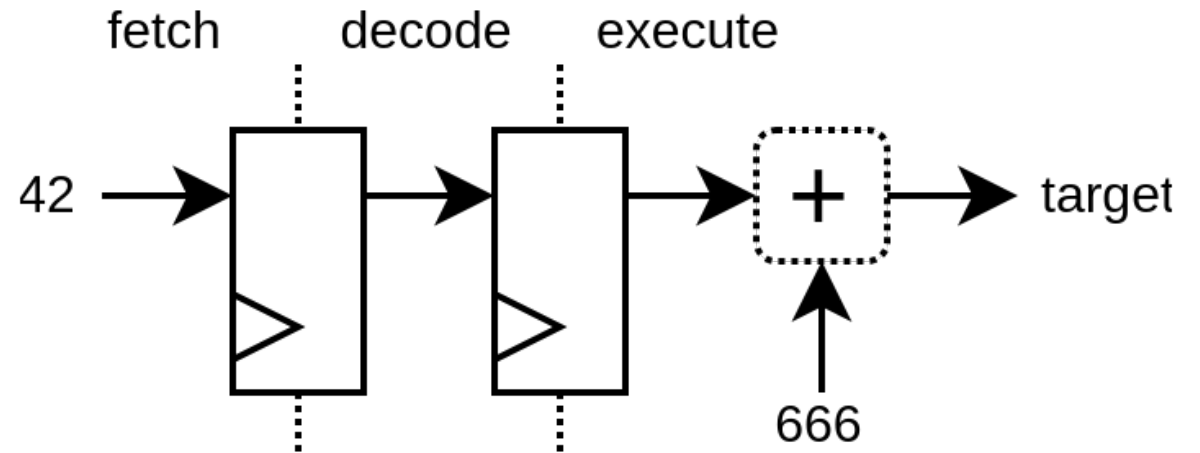

- Data structures : dynamic array / **hash map** / hash set

- Lambda function : reduce / fold / map / filter / ...

- OOP : class / software interface

- See https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/abstraction/index.html

ISA (instruction set architecture)

- A few open / free ISA (OpenRISC, RISC-V , ...)
- RISC-V
    - https://riscv.org/technical/specifications/
    - Secretly being embedded in ASICs
    - Mostly bloat free, from a FPGA perspective)
    - GCC / LLVM / Qemu
    - Linux / Debian port
    - ...

CPU design (NaxRiscv)

3 EU configuration

"Branch prediction"

Fetch[0]   PC

....   Fetch prediction : ? instruction ? data ?
                            - branch/jump presence prediction
Fetch[1]   - branch/jump target PC prediction
....   - branch taken prediction

Fetch[2]

....   Decode corrector : ? data ?
                            - branch/jump presence correction
Decode[0]   - branch/jump target PC correction

....

Dispatch[1]

....   Execute corrector :
                            - branch taken correction
Execute[0]   - jump target corrector
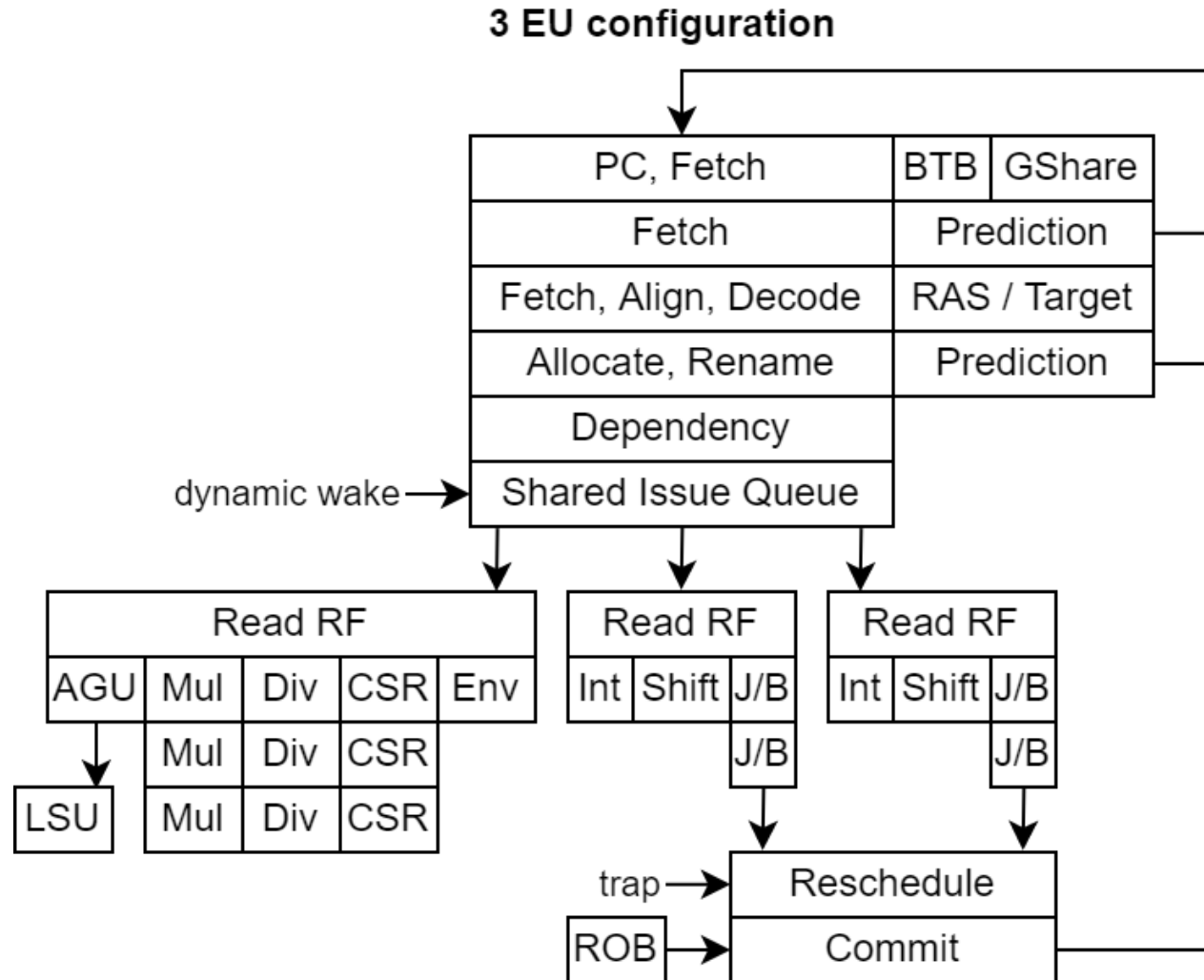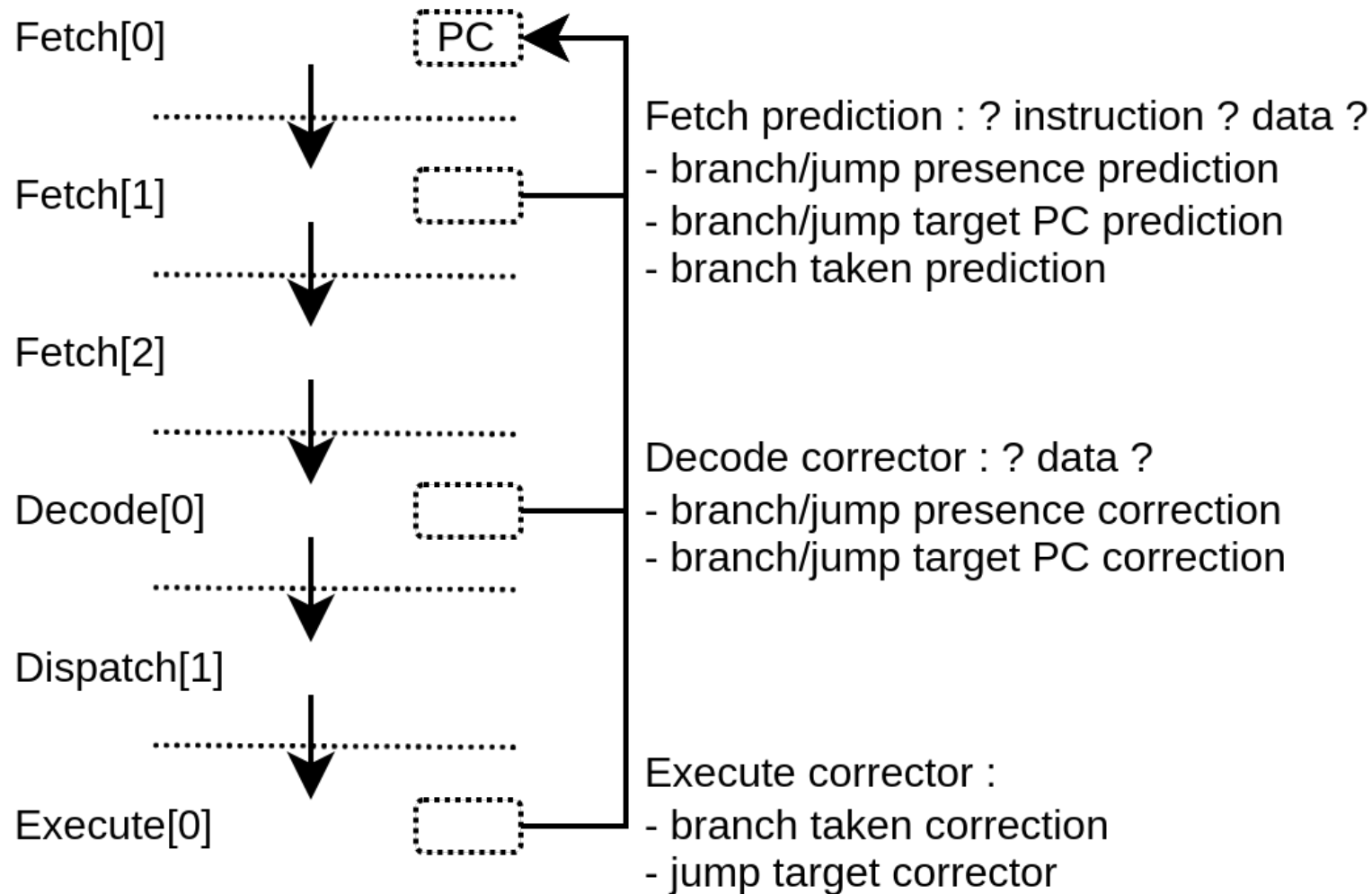
Secret leak (Konata trace)

Secret leak


```
jal      pc + 0x230        1
addi     sp, sp, 4088
sd       a3, 0(sp)
ld       a3, 0(sp)         2
sd       a3, 0(sp)
ld       a3, 0(sp)
beqz     a3, pc + 28       3
lb       t2, 0(a0)         4
srl      t2, t2, a1
andi     t2, t2, 1
slli     t2, t2, 6
or       t2, t2, a2
ld       t2, 0(t2)         5
addi     sp, sp, 8
ret
mv       a2, s1
nop
nop
nop
nop
nop
nop
nop
nop
addi     sp, sp, 8         6
ret
```

Generate long dependencies

Secret leak

Laaaaaaate branch miss prediction

Secret leak

```
jal     pc + 0x230
addi    sp, sp, 4088
sd      a3, 0(sp)
ld      a3, 0(sp)
sd      a3, 0(sp)
ld      a3, 0(sp)
beqz    a3, pc + 28
lb      t2, 0(a0)
srl     t2, t2, a1
andi    t2, t2, 1
slli    t2, t2, 6
or      t2, t2, a2
ld      t2, 0(t2)
addi    sp, sp, 8
ret
mv      a2, s1
nop
nop
nop
nop
nop
nop
nop
nop
addi    sp, sp, 8
ret
```

RAM[secret] -> (page fault)

Secret leak

probe + RAM[secret]

jal	pc + 0x230	1
addi	sp, sp, 4088	
sd	a3, 0(sp)	
ld	a3, 0(sp)	2
sd	a3, 0(sp)	
ld	a3, 0(sp)	
beqz	a3, pc + 28	3
lb	t2, 0(a0)	4
srl	t2, t2, a1	
andi	t2, t2, 1	
slli	t2, t2, 6	
or	t2, t2, a2	
ld	t2, 0(t2)	5
addi	sp, sp, 8	
ret		
mv	a2, s1	
nop		
nop		
nop		
nop		
nop		
nop		
nop		
nop		
addi	sp, sp, 8	6
ret		

Secret leak

RAM[probe + RAM[secret]]

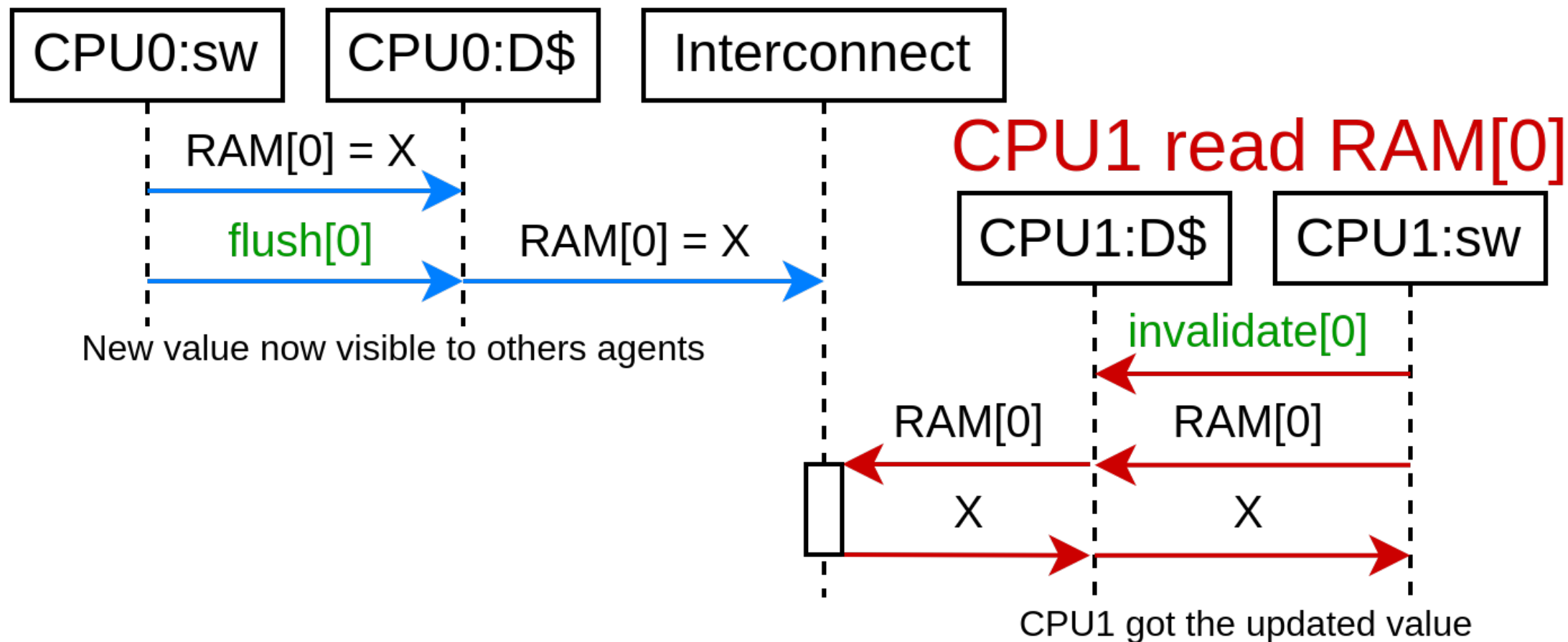# Multi core requirements (RISC-V linux)

- Hardware cache coherency

- Inter-CPU software interrupts
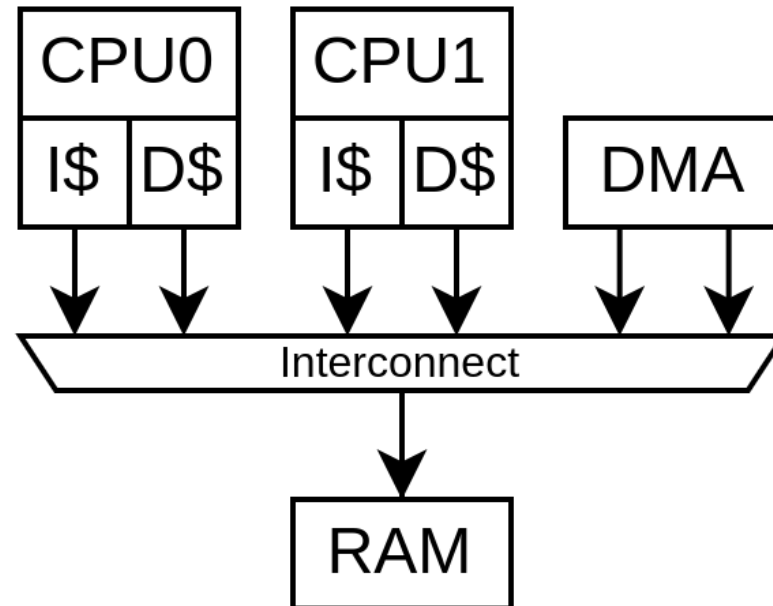
Software cache coherency

CPU0 write RAM[0]
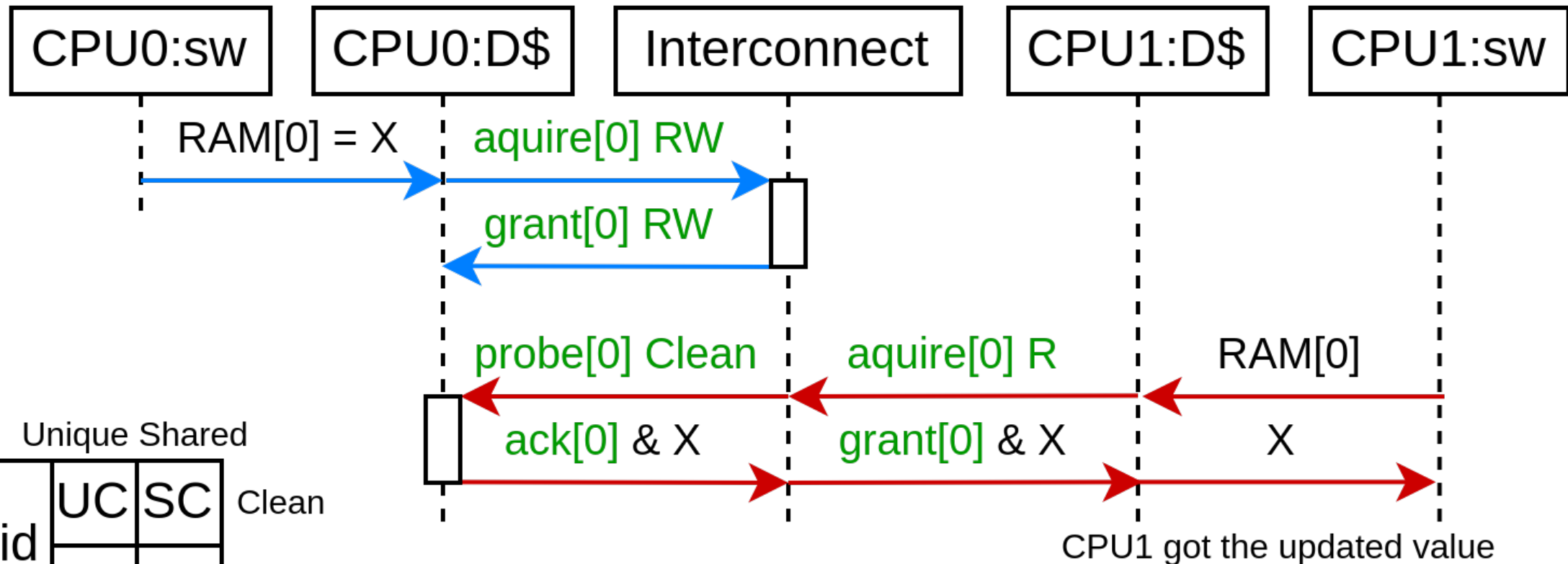

CPU1 read RAM[0]

Hardware cache coherency
with memory block copy and permissions

CPU0 write RAM[0]   CPU1 read RAM[0]

| CPU0:sw | CPU0:D$ | Interconnect | CPU1:D$ | CPU1:sw |

RAM[0] = X   aquire[0] RW

grant[0] RW

probe[0] Clean   aquire[0] R   RAM[0]

ack[0] & X   grant[0] & X   X

Unique Shared

| Invalid | UC | SC | Clean |
| | UD | SD | Dirty |

CPU1 got the updated value

Hardware cache coherency

And its side effects

```
volatile int array[2];

void thread_0() {
  while(1) {
    array[0] += 1;
  }
}

void thread_1() {
  while(1) {
    array[1] += 1;
  }
}
```

Same memory block (64 bytes): 18486 ps/iteration
Different memory blocks   :   902 ps/iteration

(on a 2 years old CPU)

Cache coherency

- Open specification : Tilelink

- Multiple open source implementations, ex :

  – https://github.com/SpinalHDL/SpinalHDL/tree/dev/lib/src/main/scala/spinal/lib/bus/tilelink

- Coherent L2 design

  – https://spinalhdl.github.io/SpinalDoc-RTD/master/SpinalHDL/Libraries/Bus/tilelink/tilelink_fabric.html

  – Exposes a lot of performance tricks

    - 1 pending transaction per 64 bytes block

   - 64 bytes per burst max

    - Keep bursts aligned

    - Alignment Aliasing

Memory / Peripherals / Deployment

- LiteX

  - Open-source SoC framework (in Python)

  - Integrate many peripherals

    - DDRx controller
    - SDCARD
    - Ethernet, USB host
    - SATA, PCIe, …

Memory / Peripherals / Deployment

- python3 -m litex_boards.targets.digilent_nexys_video --cpu-type=naxriscv --bus-standard axi-lite --with-video-framebuffer --with-coherent-dma --with-sdcard --with-ethernet --xlen=64 --scala-args='rvc=true,rvf=true,rvd=true,alu-count=2,decode-count=2' --with-jtag-tap --sys-clk-freq 100000000 --cpu-count 2 --build –load

- + stuff for USB / JTAG
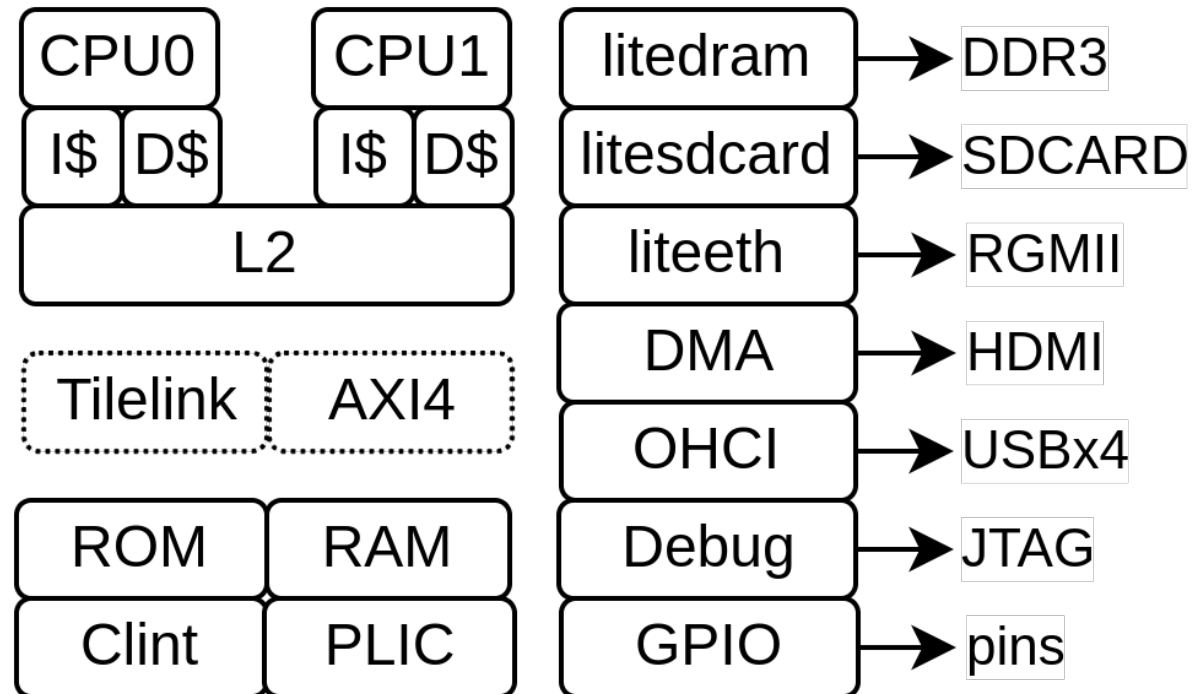
USB host support

- USB host support =>

  - Keyboard / mouse / audio / flash drive / sdcard / uart / ethernet / bluetooth

- Open specification :

  - Open Host Controller Interface Specification for USB (OHCI)

  - USB 12 Mb/s, up to 15 ports, 2 FPGA GPIO per port

- Open implementation :

  - https://spinalhdl.github.io/SpinalDoc-RTD/master/SpinalHDL/Libraries/Com/usb_ohci.html

  - https://github.com/SpinalHDL/SpinalHDL/blob/dev/lib/src/main/scala/spinal/lib/com/usb/ohci/UsbOhci.scala

  - https://github.com/litex-hub/pythondata-misc-usb_ohci/tree/master/pythondata_misc_usb_ohci/verilog

Synthesis / Place / Route

- Some open source tools for

  - Xilinx 7-Series

  - Lattice ice40 / ecp5

  - …

- Synthesis : Yosys

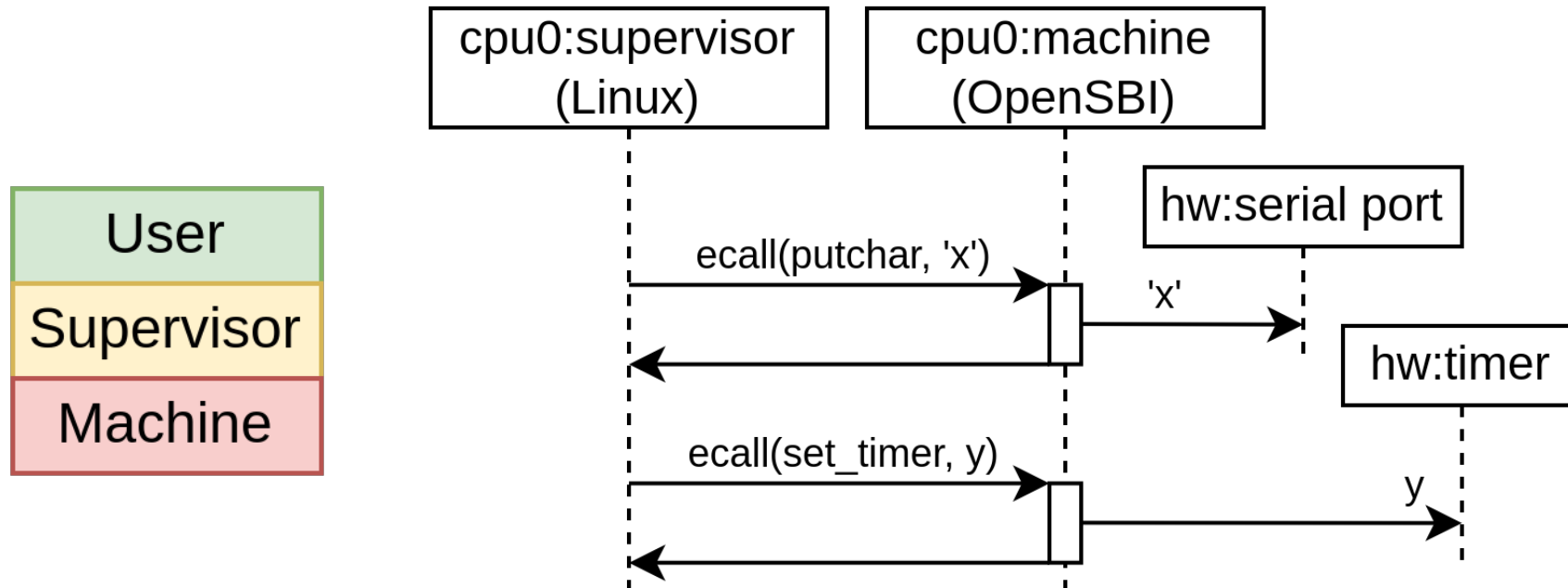- Place and route : Nextpnr

- …

Running Linux

- Requirements on RISC-V

  – Hardware cache coherency (between CPUs + DMA)

  – Machine/Supervisor/User modes (riscv-privileged spec)

  – SBI (Supervisor Binary Interface, kinda like BIOS)

  – DTB (Device Tree Binary)

https://twitter.com/i/status/1493996880593887235

SBI (Supervisor Binary Interface)

- https://github.com/riscv-non-isa/riscv-sbi-doc
- https://github.com/riscv-software-src/opensbi
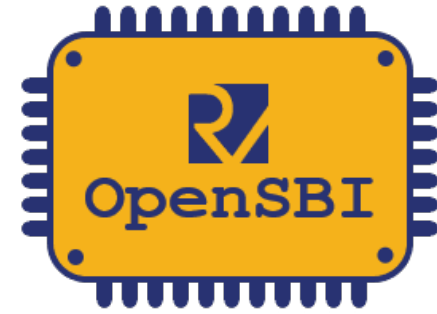
Running Debian

- Requirements on RISC-V

  – RV64IMAFDC

  – Linux kernel

  – Some storage (ex: SDCARD)

- Doc : https://github.com/SpinalHDL/NaxSoftware/tree/main/debian_litex

- Video : https://x.com/dolu1990/status/1712848731349889108

OpenTTD 12.2

OpenTTD 12.2

New Game	Load Game
Play	ghtmap
Scena	layer

Game
AI/Game S
Check O

VisualBo

LEVEL
9

HIGH

DOOM Shareware - Chocolate

PICKED UP 4 SHOTGUN SHELLS.

49 100% 2   0%   BULL 49 / 200
                           SHEL   4 / 50
AMMO HEALTH ARMS ARMOR RCKT   0 / 50
                           CELL   0 / 300

Verification / Debugging

- May create rituals

- May exhaust your sanity

- May never end

- May crush your soul

Hello darkness my old friend

Debugging – example (a real one)

- Dual core Debian freeze (RCU error) after ~1h (360'000'000'000 cycles)
  - Impossible to reproduced in simulation (~40 days of runtime @ 100 Khz)
- Hope you get clues
  - Core was still alive on JTAG => OpenOCD
  - PC was in linux "_raw_spin_lock"
  - Forcing it to unlock => Everything back on track
- What it was :
  - CPU D$ doing bad memory coherency
  - (releasing data permissions once again after being probed out)

Simulation

- A few open source simulation tools :
  - GHDL, IVerilog, Verilator, …
- Based on Verilator (~150-200 Khz on a 4 years old CPU)
- Lock-step checks against Spike / RVLS
- Konata traces

0xAA55

- I hope you had a good time

- If you are looking for open-source project funding :

    - NLnet Foundation (they helped me a lot)
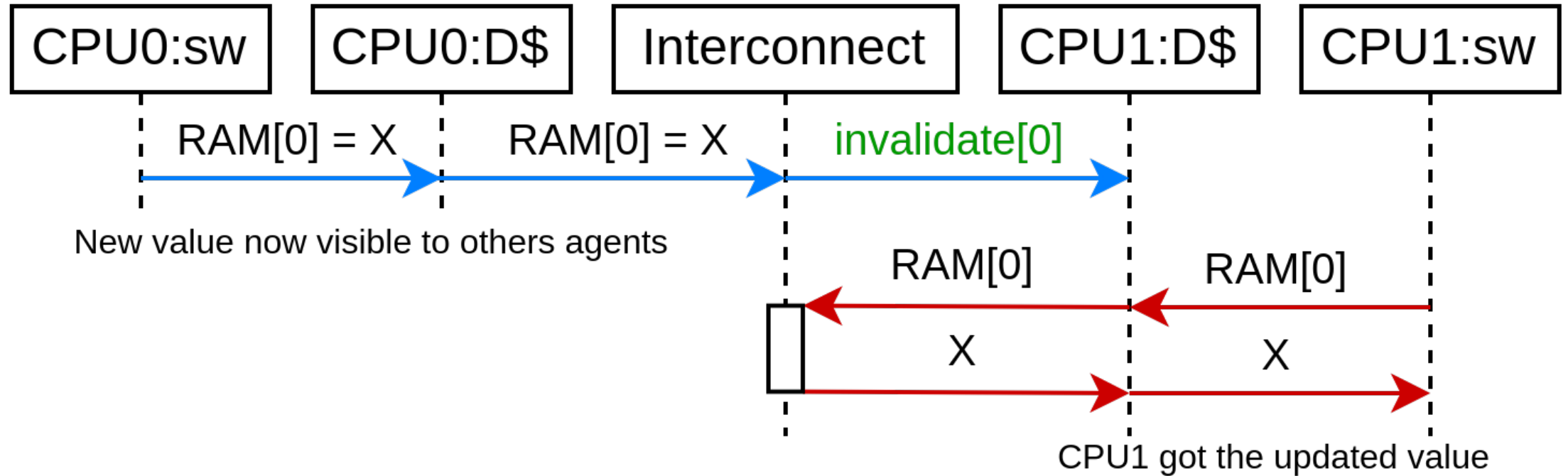
Speculative execution

- Execute as much as you can in advance

- Bad speculation → no side effect on the **ISA states**

  - x1, x2, .. , x31

  - CSR

  - Global memory (RAM)

- But what's about all the others **hardware states**

  - Branch predictors (BTB, RAS, GShare)

  - I$ D$ tags (is address X in the cache or not)

  - Those are easy to probe

Hardware cache coherency
without memory block ownership

Hardware cache coherency
with memory block ownership

- Caches have to acquire memory block permissions

  – read [write] permissions

  – 64 bytes per block (typically), aligned

- Cache line states

  – Valid / Invalid

  – Clean / Dirty

  – Unique / Shared

Boot sequence

- FPGA bitstream

- Litex bios

- OpenSBI

- [Uboot]

- Linux

- Buildroot / Debian / …

Using software to elaborate your hardware

```
val a,b,c = out UInt(8 bits)

val array = ArrayBuffer[UInt]()
array += a // By reference !
array += b
array += c

for(element <- array){
  element := 0
}
```

```
output     [7:0]    a,
output     [7:0]    b,
output     [7:0]    c

assign a = 8'h0;
assign b = 8'h0;
assign c = 8'h0;
```

- Data structures : dynamic array / hash map / hash set

- Lambda function : reduce / fold / map / filter / ...

- OOP : class / software interface

- See https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/abstraction/index.html