

# NaxRiscv CPU : introduction and extension demonstration

lw	ra, 12(sp)	F	1	2	Rn	1	Ds	1	Is	1	2	3	Cm	1	2			
andi	a5, a2, 255	F	1	2	Rn	1	Ds	1	2	3	Is	Cm	1	2	3			
sw	a5, 20(sp)	F	1	2	Rn	1	Ds	1	Is	1	2	3	4	Cm	1			
bltz	ra, pc + 912	F	1	2	Rn	1	Ds	1	2	3	4	5	Is	1	Cm			
beqz	s0, pc + 2064	F	1	2	Rn	1	Ds	1	Is	1	Cm	1	2	3	4			
mv	t1, s0	F	1	2	Rn	1	Ds	1	2	Is	Cm	1	2	3	4			
j	pc + 0xc	F	1	2	Rn	1	Ds	1	Is	1	Cm	1	2	3	4			
lw	t2, 4(t1)	F	1	2	Rn	1	Ds	1	2	Is	1	2	3	Cm	1			
lw	s3, 12(sp)	F	1	2	Rn	1	Ds	1	Is	1	2	3	Cm	1	2			
lh	s2, 2(t2)	F	1	2	Rn	1	Ds	1	2	3	4	Is	1	2	3	Cm		
bne	s2, s3, pc + 4076	F	1	2	Rn	1	Ds	1	2	3	4	5	6	7	Is	1	Cm	
lw	t1, 0(t1)	F	1	2	Rn	1	Ds	1	Is	1	2	3	Cm	1	2	3	4	
beqz	t1, pc + 20	F	1	2	Rn	1	Ds	1	2	3	4	Is	1	Cm	1	2	3	
lw	t2, 4(t1)	F	1	2	Rn	1	Ds	1	2	3	Is	1	2	3	Cm	1	2	
lw	s3, 12(sp)	F	1	2	Rn	1	Ds	1	Is	1	2	3	Cm	1	2	3	4	
lh	s2, 2(t2)	F	1	2	Rn	1	Ds	1	2	3	4	5	Is	1	2	3	Cm	
bne	s2, s3, pc + 4076	F	1	2	Rn	1	Ds	1	2	3	4	5	6	7	8	Is	1	Cm
lw	t1, 0(t1)	F	1	2	Rn	1	Ds	1	2	Is	1	2	3	Cm	1	2	3	4
beqz	t1, pc + 20	F	1	2	Rn	1	Ds	1	2	3	4	5	Is	1	Cm	1	2	3
lw	t2, 4(t1)	F	1	2	Rn	1	Ds	1	2	3	4	Is	1	2	3	Cm	1	2
lw	s3, 12(sp)	F	1	2	Rn	1	Ds	1	2	Is	1	2	3	Cm	1	2	3	4
lh	s2, 2(t2)	F	1	2	Rn	1	Ds	1	2	3	4	5	6	Is	1	2	3	

# Background / whoami

- Dolu1990 on github
- Active on
  - SpinalHDL
  - VexRiscv
  - NaxRiscv
- Software / Hardware background

# NaxRiscv in short

- RV32/RV64 IMAFDCCSU (Linux/Debian ready)
- Out of order / superscalar
- Free / Open source (MIT)
- Many paradigm experimentation with alternative HDL
- Target FPGA with distributed RAM
- ~75 FPS Chocolate-doom (@100Mhz / linux)

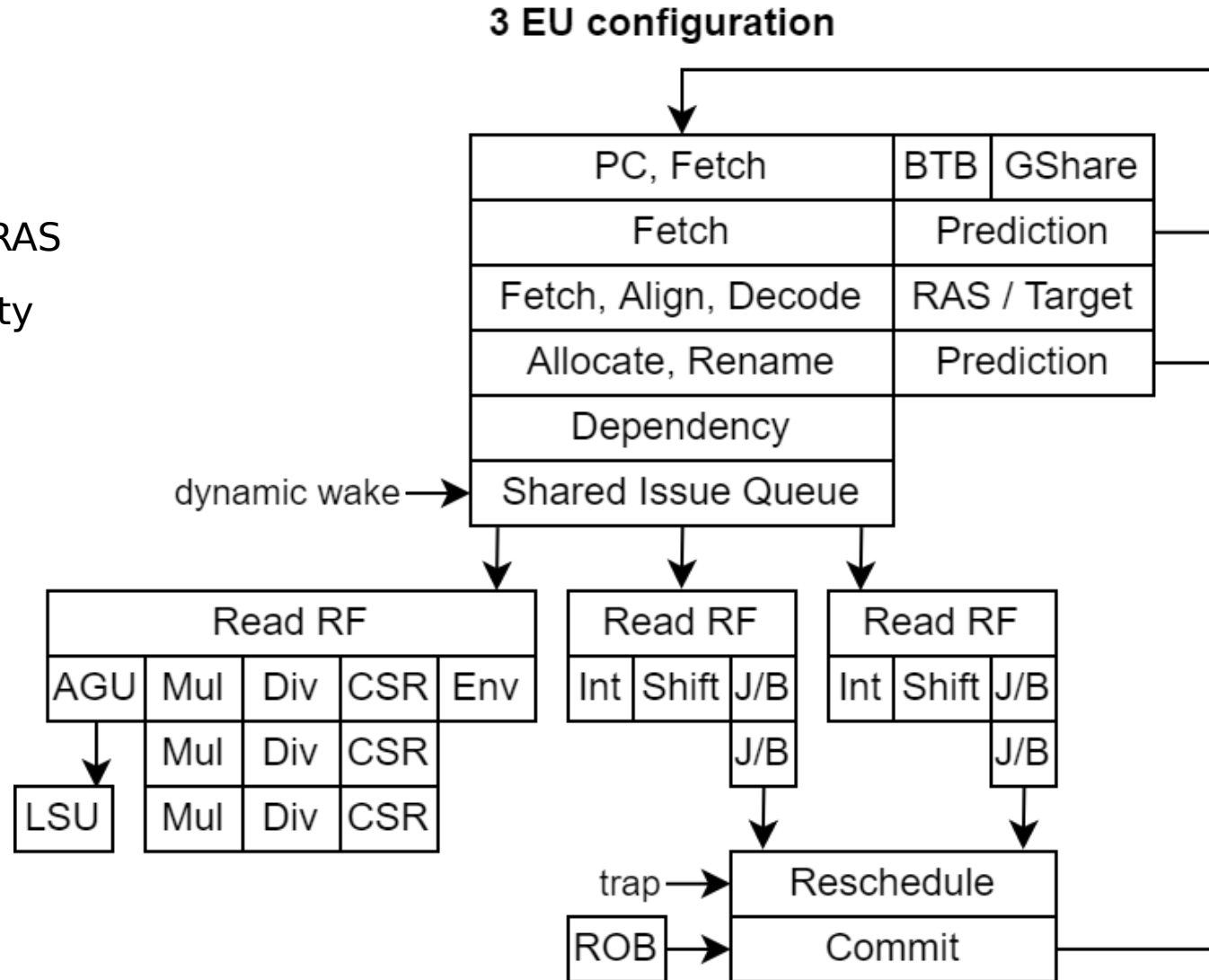


# Performances / Area

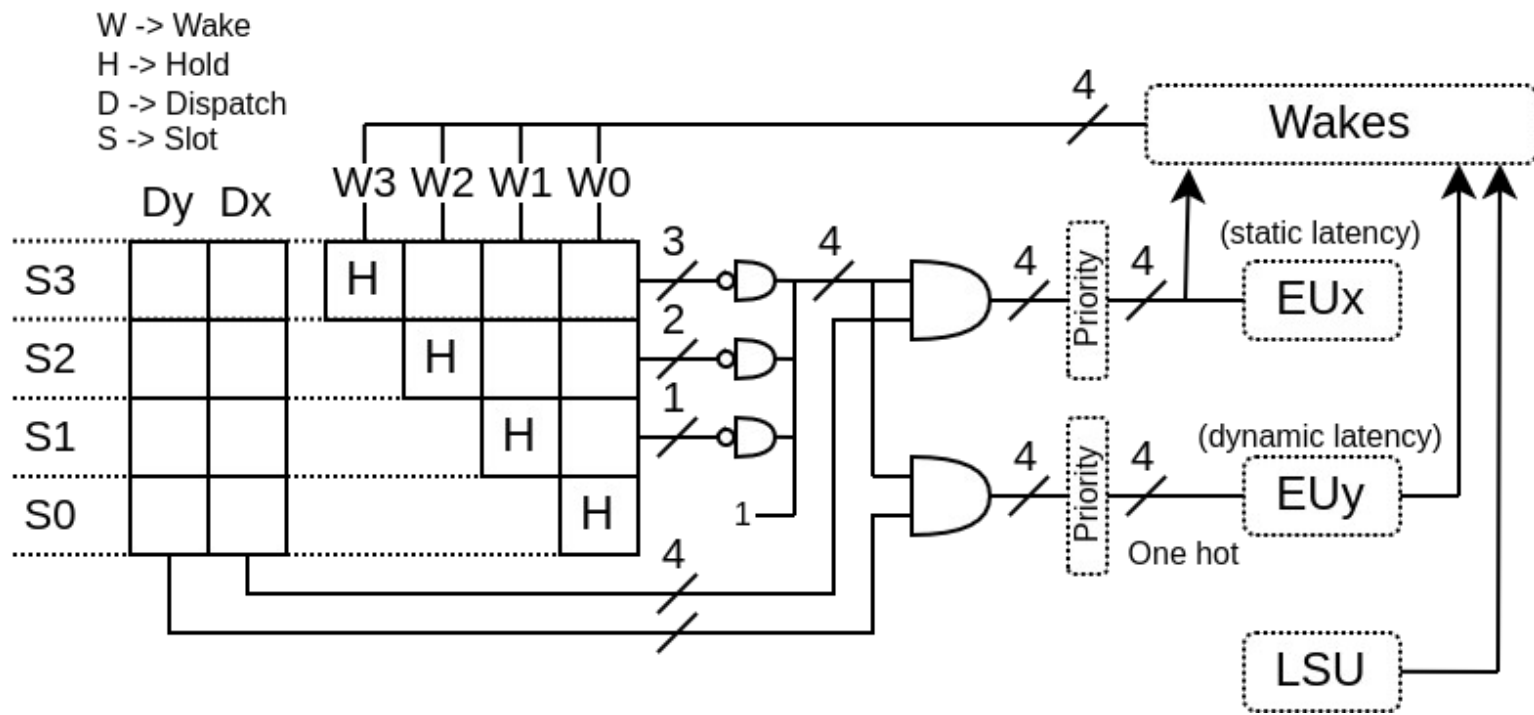
- RV32IMASU, 2 decode, 3 issue (Linux ready)
  - Dhrystone : 2.93 DMIPS/Mhz 1.65 IPC (-O3 -fno-common -fno-inline)
  - Coremark : 5.00 Coremark/Mhz 1.28 IPC
  - Embench-iot : 1.68 baseline 1.42 IPC (baseline = Cortex M4)
- On Artix 7 speed grade 3 :
  - 145 Mhz (360 Mhz on Kintex ultrascale+ grade 3)
  - 14.2 KLUT, 9.7 KFF, 11.5 BRAM, 4 DSP
- RV64IMASU, same FPGA/config
  - 137 Mhz
  - 18.6 KLUT, 11.8 KFF, 11.5 BRAM, 16 DSP

# Architecture (from last slide)

- 2 decode, 3 issue (2 ALU, 1 shared EU)
- 64 physical register, 64 entry ROB
- 4KB GShare + 4KB BTB (both 1 way), RAS
- 10 cycles miss predicted branch penalty
- Non-blocking D\$
- I\$ D\$ each have 16 KB (4 ways)
- I\$ D\$ each have 192 TLB (2+4 ways)

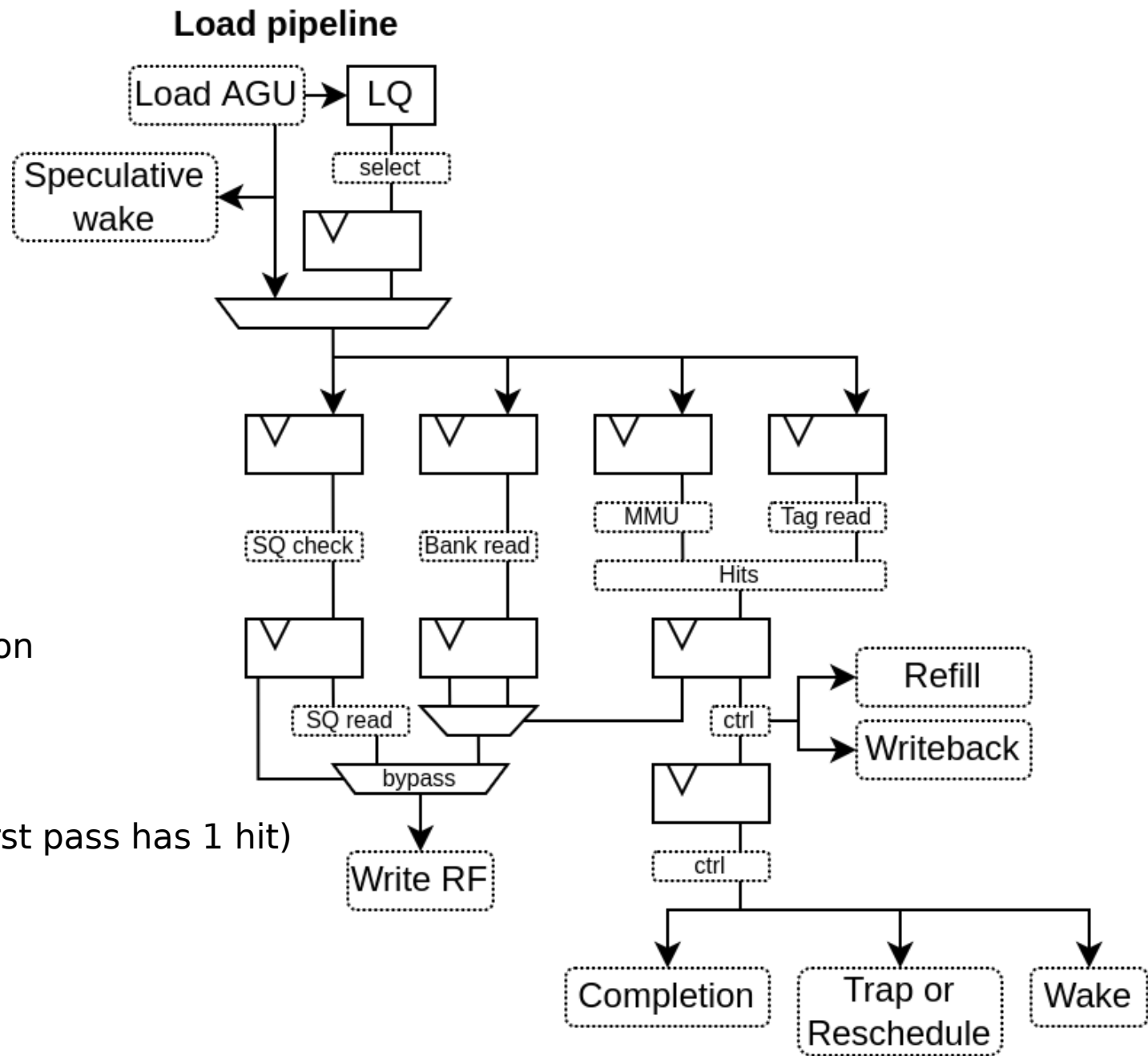


- Shared between execution units
- 32 entries, grouped by “wave”, (ex: 2x16)
- Older first, no compression
- Matrix design, removal on completion



# LSU (load)

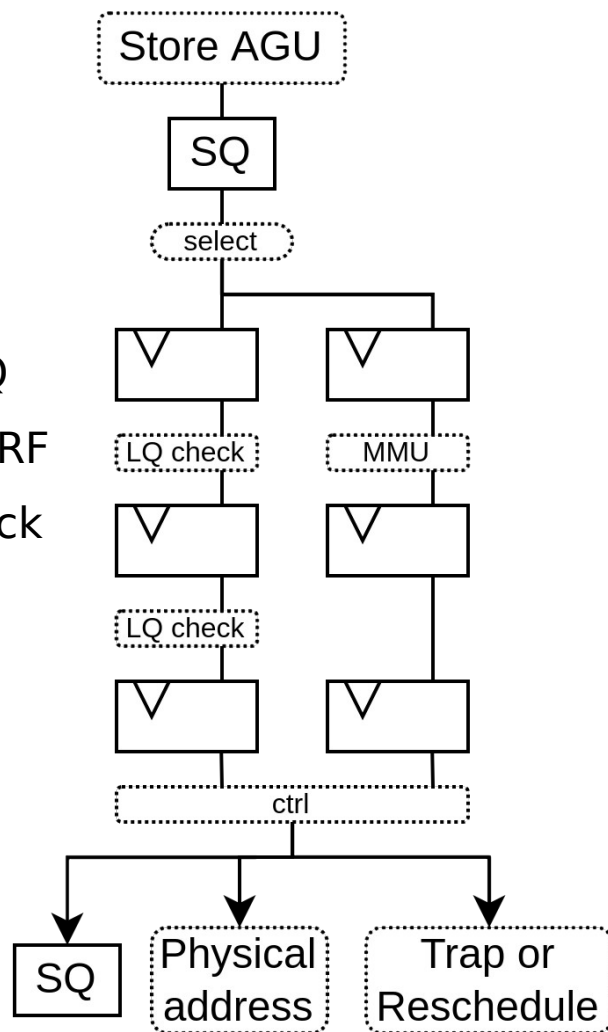
- Speculative wake (hit prediction)
- load to use latency :
  - 3 cycle (hit predicted OK)
  - 6 cycle (miss predicted)
  - Full flush (hit predicted KO)
- AGU to D\$ bypass
- RF written 1 cycle before completion
- SQ address hazard check
  - 4KB virtual (first pass)
  - Full physical (second pass if first pass has 1 hit)
- 16 entries



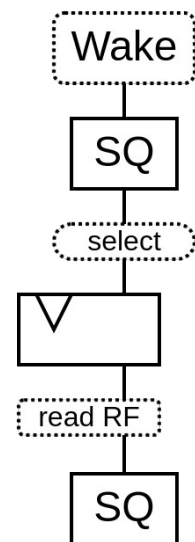
# LSU (store)

- 4 pipelines
- Dependencies
  - Address through IQ
  - Data directly read RF
- LQ address hazard check
  - 4KB virtual
- 16 entries

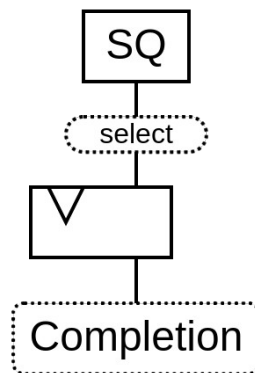
Store pipeline (address)



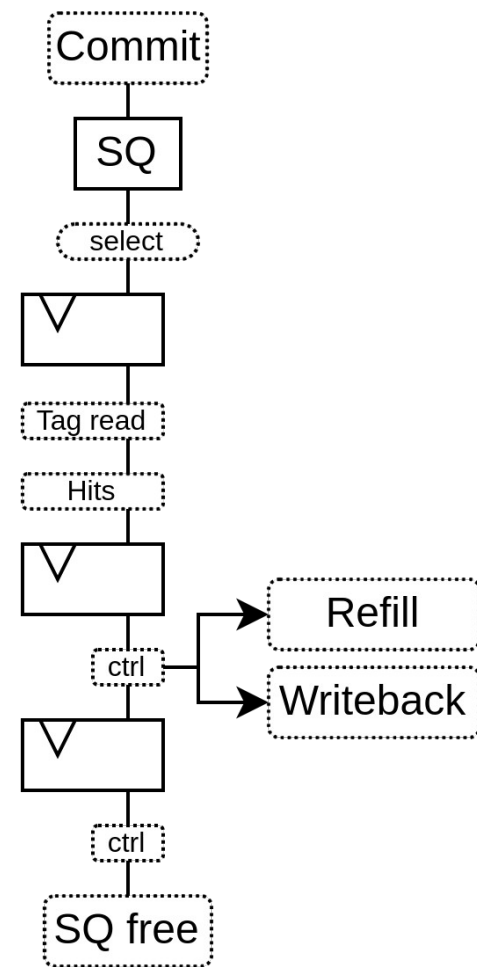
Store pipeline (data)



Store pipeline (completion)



Writeback pipeline





# Simulation

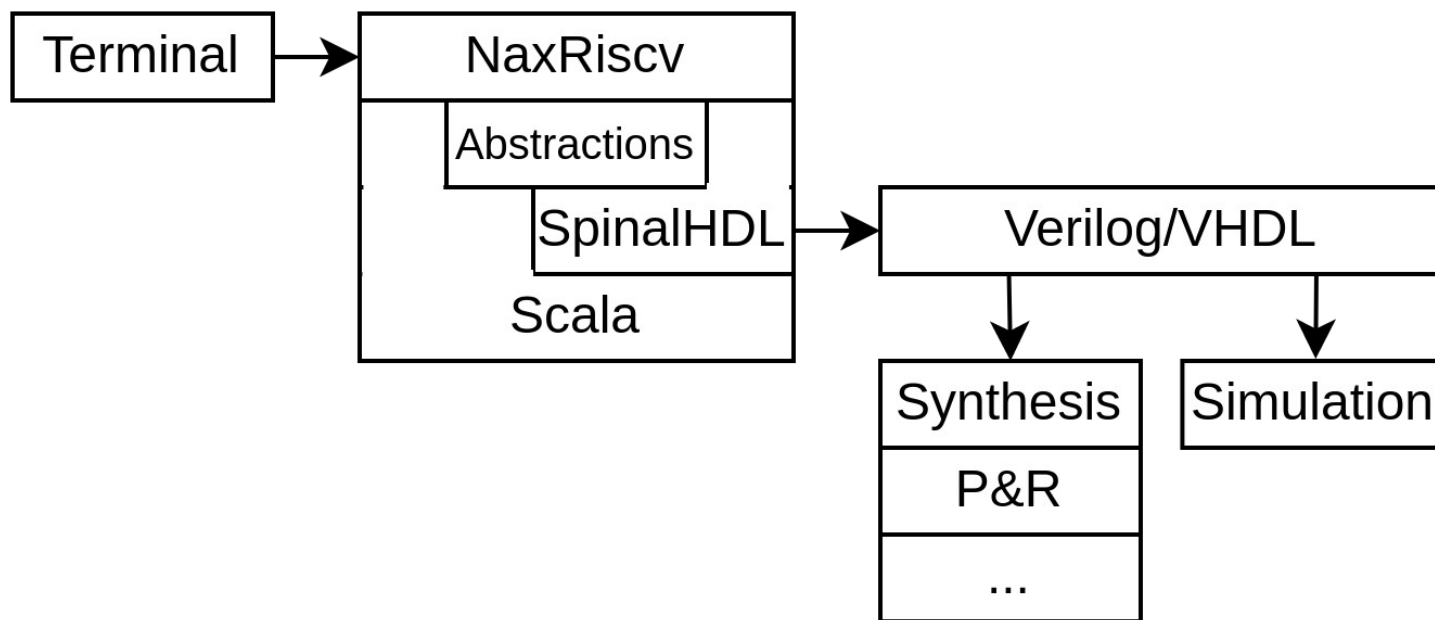
- Based on Verilator (~150-200 Khz on Ryzen 3900)
- Lock-step checks against Spike (life saver)
- Can emit Gem5-O3 / Konata traces

```
0x80006858: slli    a2, a4, 2
0x8000685c: addi    a4, a2, 64
0x80006860: addi    a2, sp, 16
0x80006864: add     a4, a4, a2
0x80006868: lw      a2, 4032(a4)
0x8000686c: addiw   a2, a2, 1
0x80006870: sw      a2, -64(a4)
0x80006874: bnez    a6, pc + 4004
0x80006878: lbu     a4, 1(a7)
0x8000687c: addi    a2, a7, 1
0x80006880: beq     a6, t5, pc + 1148
0x80006884: addiw   t1, a6, 4048
0x80006888: andi    t1, t1, 255
0x8000688c: addiw   t4, t4, 1
0x80006890: bgeu    t3, t1, pc + 628
0x80006894: beqz    a4, pc + 784
0x80006898: beq     a4, t5, pc + 800
0x8000689c: mv      a6, a4
0x800068a0: addi    a2, a2, 1
0x800068a4: lbu     a4, 0(a2)
0x800068a8: beq     a6, a3, pc + 44
0x800068ac: addiw   a6, a6, 4048
0x800068b0: andi    a7, a6, 255
```



# NaxRiscv elaboration

sbt "runMain naxriscv.Gen"



# Scala in one slides

```
object Main extends App{  
  println("hello world")  
}
```

```
object Main extends App{  
  val verilog = new PrintWriter(new File("toplevel.v"))  
  verilog.write("module miaou{\n")  
  verilog.write("  input  clk,\n")  
  //...  
  verilog.close()  
}
```

# Scala in one slides

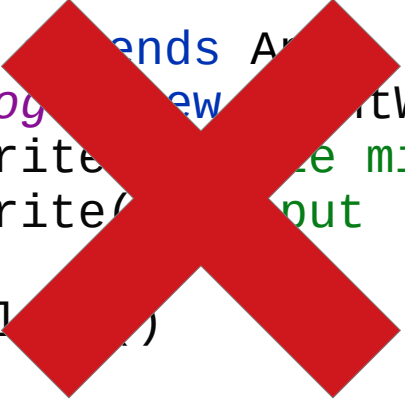
```
object Main extends App{  
  println("hello world")  
}
```

```
object Main extends App{  
  val verilog = new PrintWriter(new File("toplevel.v"))  
  verilog.write("module miaou{\n")  
  verilog.write("  input  clk,\n")  
  //...  
  verilog.close()  
}
```

# Scala in one slides

```
object Main extends App{  
  println("hello world")  
}
```

```
object Main extends App{  
  val verilog = new PrintWriter(new File("toplevel.v"))  
  verilog.write("miaoou\n")  
  verilog.write("output clk,\n")  
  //...  
  verilog.close()  
}
```



# SpinalHDL in two slides

```
import spinal.core._
```

```
object Main extends App{  
  SpinalVerilog(  
    new Module{  
      val a,b = in UInt(8 bits)  
      val result = out(a + b)  
    }  
  )  
}
```



```
module unnamed (  
  input      [7:0]  a,  
  input      [7:0]  b,  
  output     [7:0]  result  
);  
  assign result = (a + b);  
endmodule
```

# SpinalHDL in two slides

```
import spinal.core._
```

```
object Main extends App{  
  SpinalVerilog(  
    new Module{  
      val a,b = in UInt(8 bits)  
      val result = out(a + b)  
    }  
  )  
}
```



```
module unnamed (  
  input      [7:0]    a,  
  input      [7:0]    b,  
  output     [7:0]    result  
);  
  assign result = (a + b);  
endmodule
```

# SpinalHDL – Scala interaction

```
new Module {  
  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  
  }  
}
```



```
module unnamed (  
  output [7:0] a,  
  output [7:0] b,  
  output [7:0] c  
);  
  assign a = 8'h0;  
  assign b = 8'h0;  
  assign c = 8'h0;  
endmodule
```



# SpinalHDL – Scala interaction

```
new Module {  
  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  
  }  
}
```



```
module unnamed (  
  output [7:0] a,  
  output [7:0] b,  
  output [7:0] c  
);  
  assign a = 8'h0;  
  assign b = 8'h0;  
  assign c = 8'h0;  
endmodule
```

# SpinalHDL – Scala interaction

```
new Module {  
  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  
  }  
}
```



```
module unnamed (  
  output [7:0] a,  
  output [7:0] b,  
  output [7:0] c  
);  
  assign a = 8'h0;  
  assign b = 8'h0;  
  assign c = 8'h0;  
endmodule
```

# SpinalHDL – API side effects

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

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

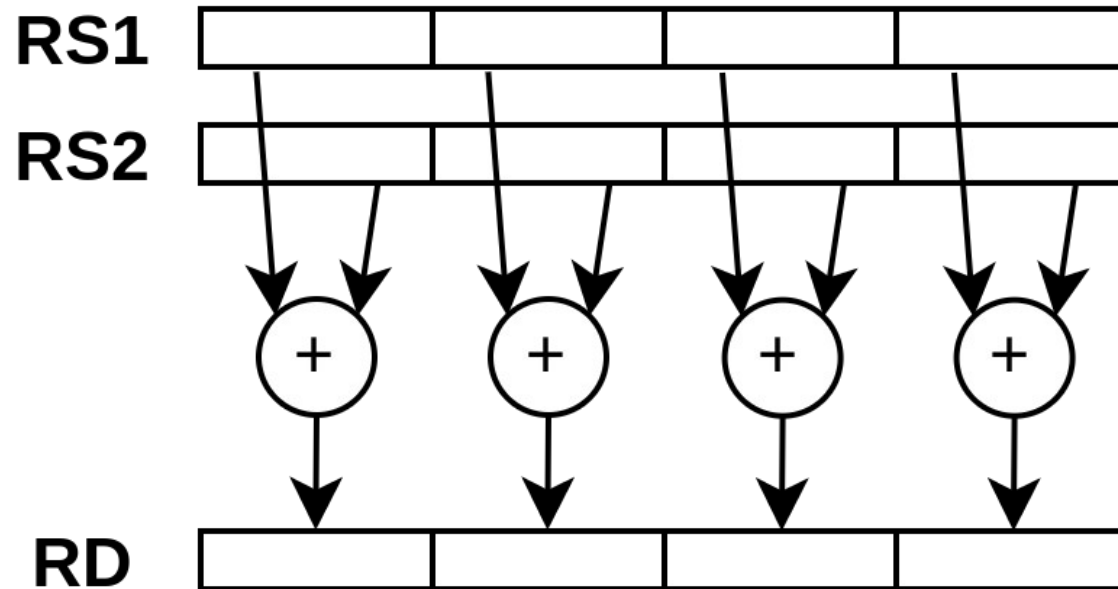
```
  for(element <- array){  
    element := 0  
  }  
}
```



```
module unnamed (  
  output [7:0] a,  
  output [7:0] b,  
  output [7:0] c  
);  
  assign a = 8'h0;  
  assign b = 8'h0;  
  assign c = 8'h0;  
endmodule
```

# Demonstration – Custom instruction

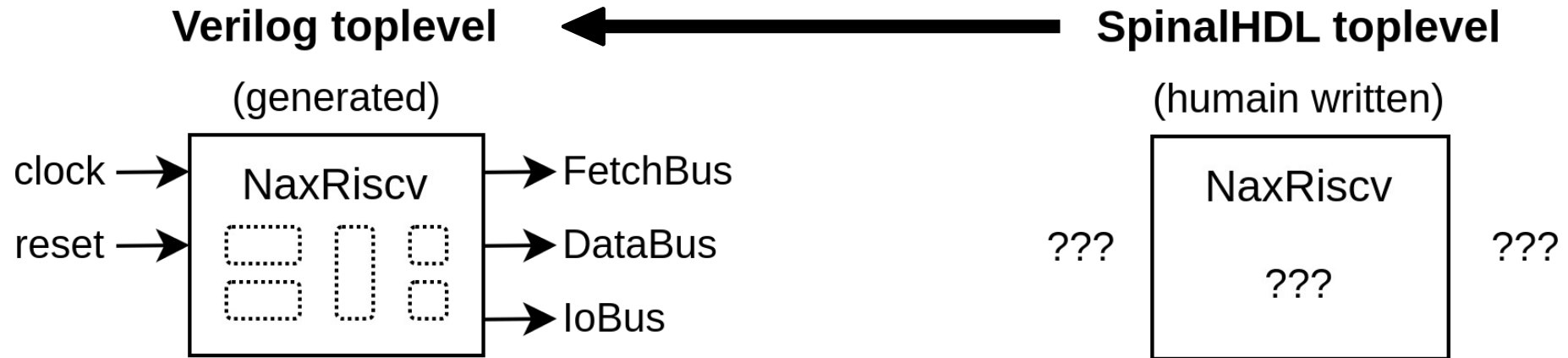
- A few steps
  - Implementation
  - Integration
  - Generation
  - Baremetal
  - Simulation



# Status / Future

- JTAG / OpenOCD / GDB supported (RISCV External Debug Support 0.13.2)
- Planning :
  - Debian support
  - Multicore / Memory coherency
- GIT : <https://github.com/SpinalHDL/NaxRiscv>
- Another talk : <https://peertube.f-si.org/videos/watch/b2c577b1-977a-48b9-b395-244ebf3d1ef6>
- Integrated in Litex for FPGA deployments
  - `python3 -m litex_boards.targets.digilent_nexys_video --cpu-type=naxriscv --with-video-framebuffer --with-spi-sdcard --with-ethernet --build -load`
- Thanks Nlnet for the funding

Extra slides not included in the talk



```
class NaxRiscv(plugins : Seq[Plugin]) extends Component{  
  val database = new DataBase  
  val framework = NaxScope(database) on new Framework(plugins)  
}
```

```

class SimdAddPlugin(val euld : String) extends ExecutionUnitElementSimple(euld, staticLatency = true) {
  override def euWritebackAt = 0
  override val setup = create early new Setup{
    add(SimdAddPlugin.ADD4)
  }

  override val logic = create late new Logic{
    val process = new ExecuteArea(stageld = 0) {
      val rs1 = stage(eu(IntRegFile, RS1)).asUInt
      val rs2 = stage(eu(IntRegFile, RS2)).asUInt

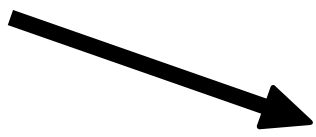
      val rd = UInt(32 bits)
      rd( 7 downto 0) := rs1( 7 downto 0) + rs2( 7 downto 0)
      rd(16 downto 8) := rs1(16 downto 8) + rs2(16 downto 8)
      rd(23 downto 16) := rs1(23 downto 16) + rs2(23 downto 16)
      rd(31 downto 24) := rs1(31 downto 24) + rs2(31 downto 24)

      wb.payload := rd.asBits
    }
  }
}


```



```
class CommitPlugin extends Plugin {  
  val setup = create early new Area {  
    val jump = getService[JumpService].newJumpPort(10)  
  }  
  val logic = create late new Area {  
    setup.jump.valid := ???  
    setup.jump.payload := ???  
  }  
}
```



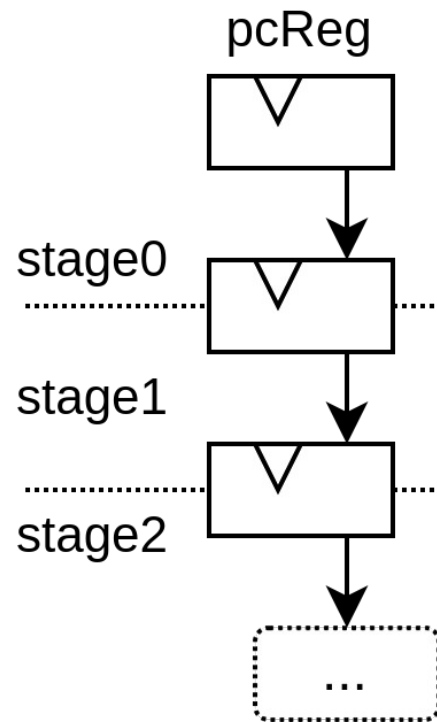
```
trait JumpService extends Service {  
  def newJumpPort(priority : Int) : Flow[UInt]  
}
```



```
class PcPlugin extends Plugin with JumpService {  
  override def newJumpPort(priority: Int) = {...}  
  val logic = create late new Area {  
    val pc = Reg(UInt(32 bits))  
    ...  
  }  
}
```

```
class PcPlugin extends Plugin {  
  val logic = ...{  
    val pcReg = Reg(UInt(32 bits))  
    getService[FetchPlugin].pipeline.stages(0)(PC) := pcReg  
  }  
}  
  
class FetchCachePlugin extends Plugin{  
  val logic = ...{  
    getService[FetchPlugin].pipeline.stages(2)(PC) === wayTag  
  }  
}
```

object PC extends Stageable(UInt(32 bits))



```

class FetchCachePlugin extends Plugin{
  val logic = ...{
    getService[FetchPlugin].pipeline.stages(2)(PC) === wayTag
  }
}

```



```

class FetchPlugin() extends Plugin {
  val pipeline = create early new Pipeline {
    val stages = Array.fill(3)(newStage())
    ...
  }

  val lock = Lock()
  val builder = create late new Area{
    lock.await()
    pipeline.build()
  }
}

```

