The RISC-V Berkeley Out-of-Order Machine:

An update on BOOM and the wider ecosystem
(Chisel, FIRRTL, & Rocket-chip)

Christopher Celio

ORCONF

2016 October

celio@eecs.berkeley.edu

http://ucb-bar.github.io/riscv-boom

ASPIRE
UC Berkeley

Berkeley
Architecture
Research

RISC-V

An Update on the Berkeley Architecture Research Infrastructure

(Oh, and BOOM is cool too.)

Christopher Celio

celio@eecs.berkeley.edu

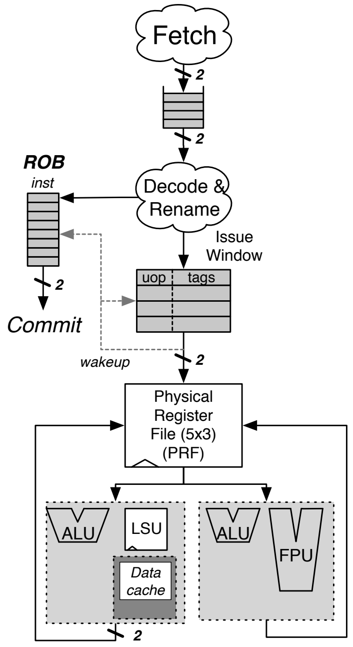
ORCONF

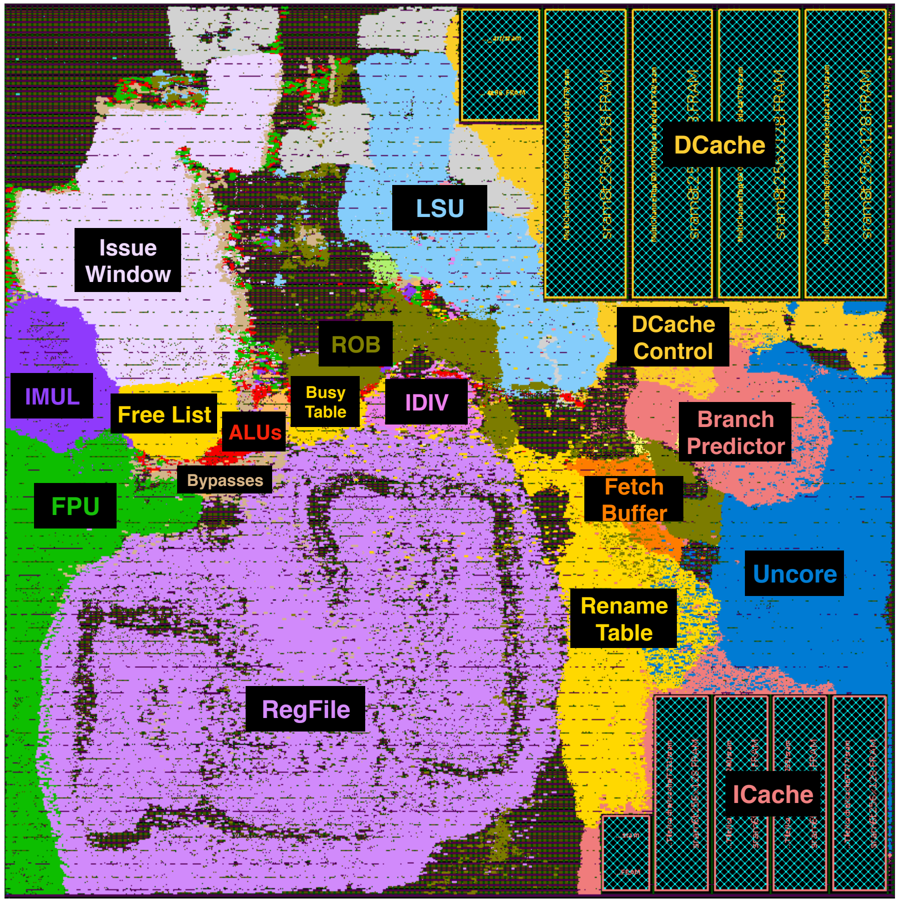
2016 October

http://ucb-bar.github.io/riscv-boom

What is BOOM?

- superscalar, out-of-order RISC-V processor written in Berkeley's Chisel hardware construction language (HCL)
- It is synthesizable
- It is parameterizable
- it is open-source





2-wide BOOM (16kB/16kB) 1.2mm^2 @ 45nm

How do you make an OoO processor?

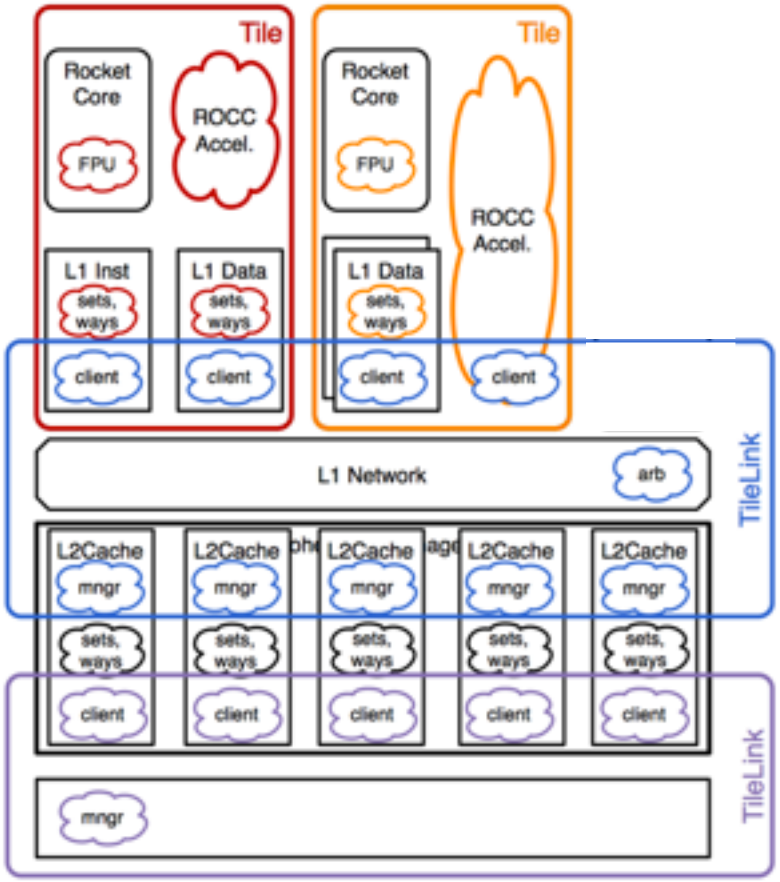
- Start with a new hardware construction language
  - Verilog is awful and will just get in the way.
- Start with a working processor.
  - Way easier than writing everything from scratch!
  - PTWs, FPUs, uncore, devices, off-chip IOs are unglamorous.

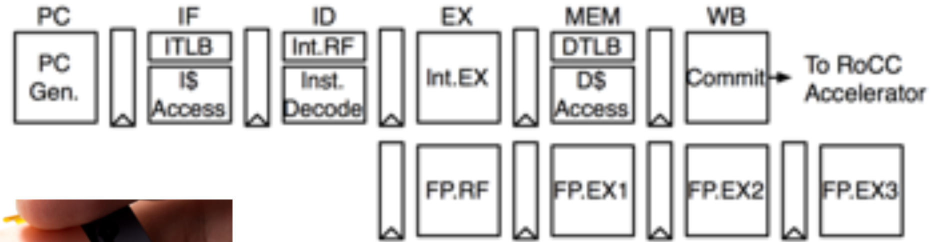
It takes a village.

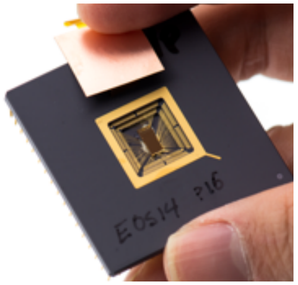
- RISC-V ISA
  - very out-of-order friendly!
- Chisel hardware construction language
  - object-oriented, functional programming
- FIRRTL (**brand new**!)
  - exposed RTL intermediate representation (IR)
- Rocket-chip
  - A full working SoC platform built around the Rocket in-order core
- Thanks to:
  - Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Palmer Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, Jack Koenig, Jim Lawson, Yunsup Lee, Richard Lin, Eric Love, Martin Maas, Chick Markley, Albert Magyar, Howard Mao, Miquel Moreto, Quan Nguyen, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Wenyu Tang, Stephen Twigg, Huy Vo, Andrew Waterman, Angie Wang, and more...

The Rocket-Chip SoC Generator Ecosystem

- BOOM is a piece of the Rocket-chip ecosystem
- Started in 2011
- taped out ~~10~~ (12?) times by Berkeley
- runs at **1.65 GHz** in IBM 45nm
- MMU supports page-based virtual memory
- IEEE 754-2008 compliant FPU
  - supports SP, DP FMA with hw support for subnormals
- cache coherent, non-blocking L1 data cache, L2 cache, and more





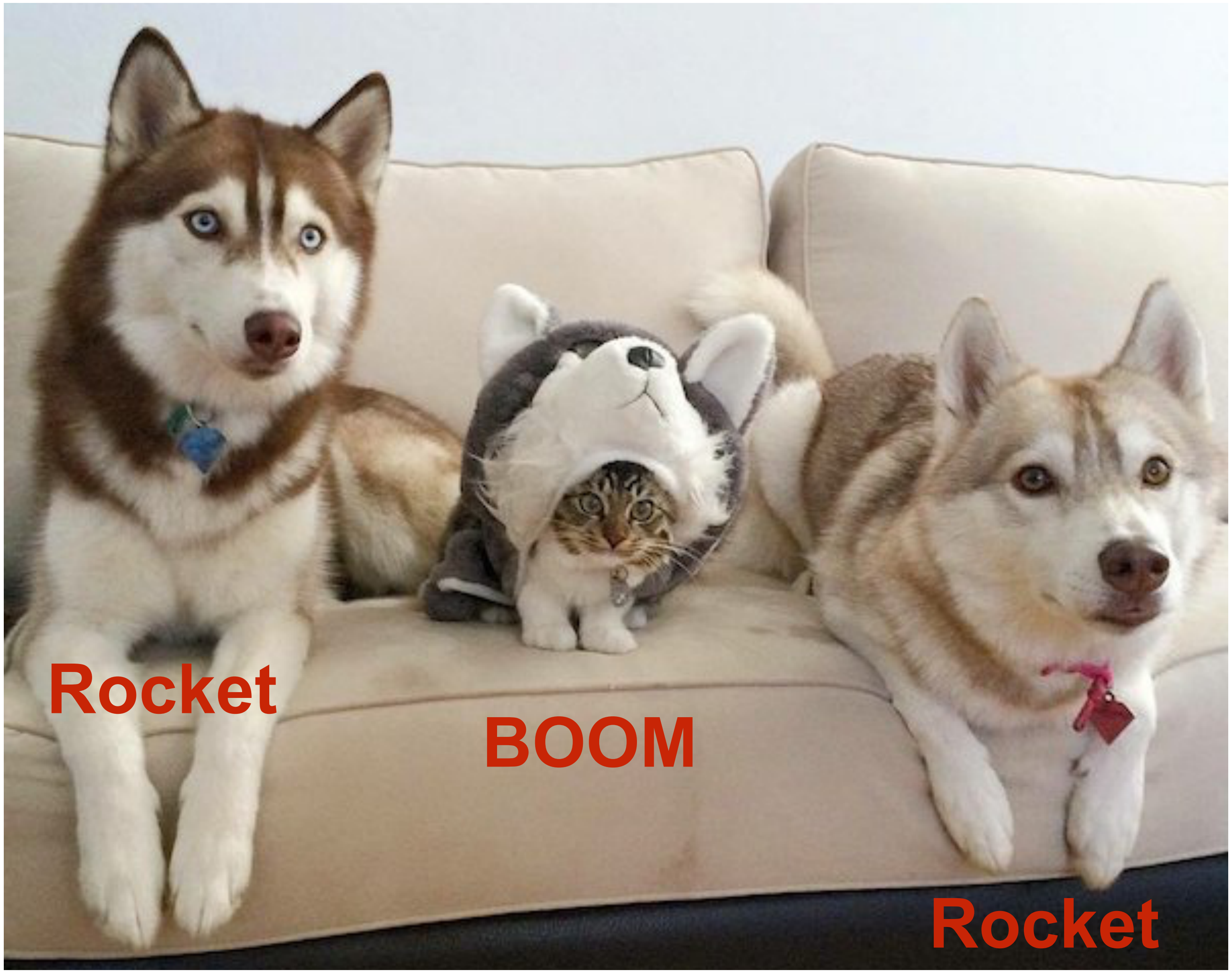


Rocket 5-stage pipeline

https://github.com/ucb-bar/rocket-chip

https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

BOOM fits into Rocket-chip SoC

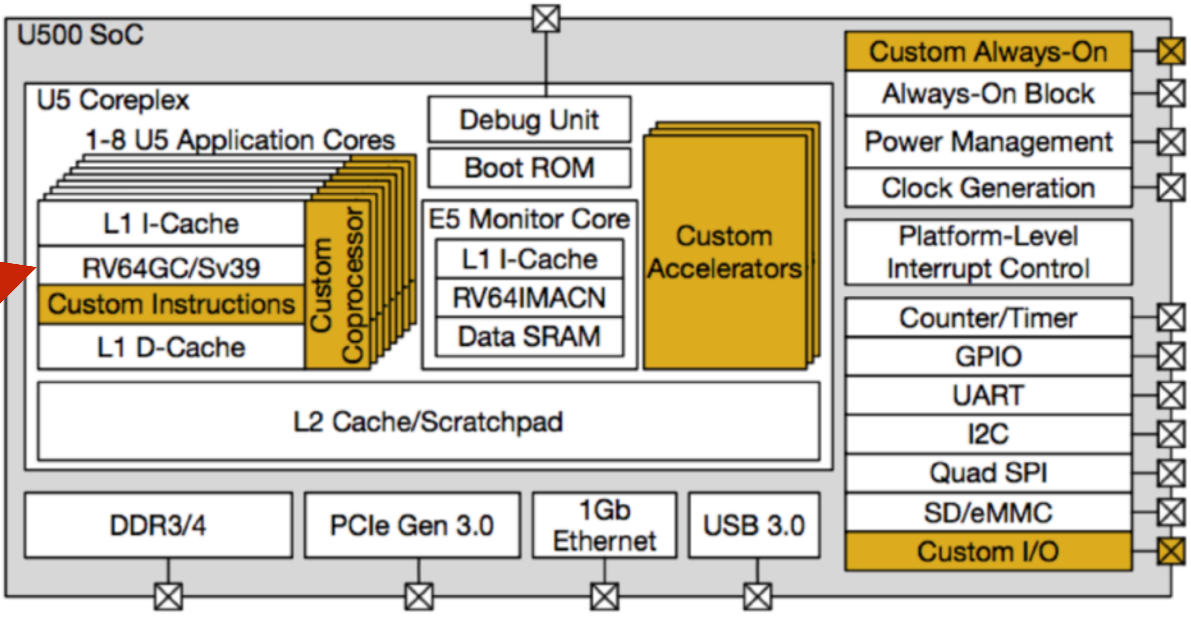


Rocket-chip Updates

- The devs graduated!
  - started SiFive start-up to support RISC-V, rocket-chip and design custom chips around them
- work in progress on a new Rocket-chip Foundation
- new generation of Berkeley student devs!
  - Berkeley is committed to open-source Rocket-chip
  - (our research depends on it!)
- 37 contributors
  - SiFive, Berkeley, LowRISC, Boston U., and more

the vision...

Rocket-chip



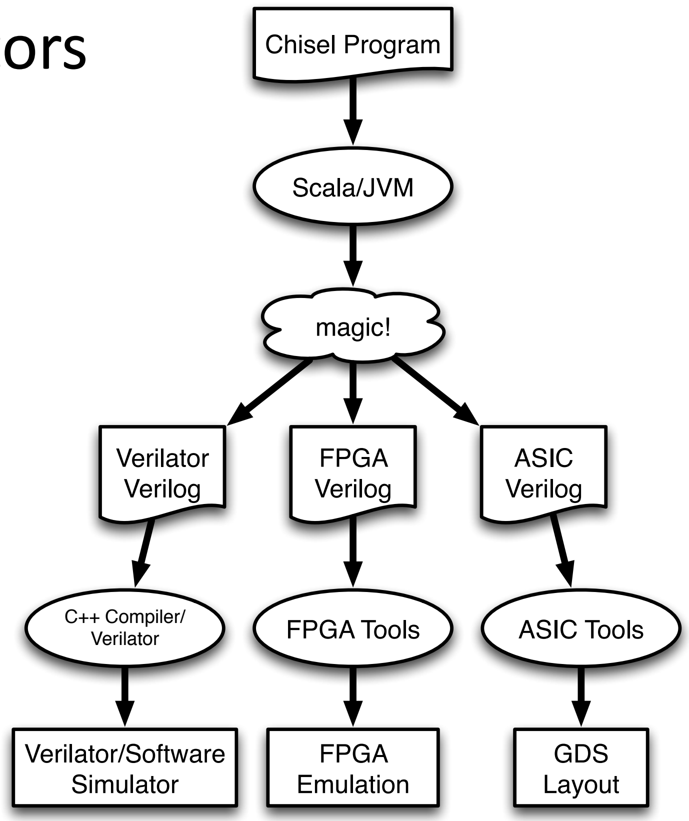
https://dev.sifive.com/documentation/freedom-u500-platform-guide/

Rocket-chip Updates

- committed to open-source since Oct 2011
  - 3588 commits!!!
  - 310 commits last four weeks!
- Better documentation
  - https://dev.sifive.com/documentation/u5-coreplex-series-manual/
- removed git submodules
  - speed up development (reduce merge headaches)
- RISC-V Privileged Spec v1.9
- uncached loads/stores (memory-mapped IO)
- RISC-V External Debug
  - breakpoints
  - stand-alone boot!
  - non-standard HTIF (host/target interface) has been removed!
- More configurations
  - RV32 support + M/A/F as options
  - RISC-V Compressed ISA support
  - blocking data cache (MSHRs = 0)
- Updated to Chisel3
  - No more C++ backend, only Verilog is emitted.
  - we use Verilator in the build system for free and fast simulation.
- And a lot more...
  - tilelink updates, multi-clock domains, I/Os, devices, ...
  - I'm having trouble keeping up!

Chisel

- Hardware Construction Language embedded in **Scala**
- ***not*** a high-level synthesis language
- hardware module is a data structure in Scala
- Full power of Scala for writing generators
  - object-oriented programming
    - factory objects, traits, overloading
  - functional programming
    - high-order funs, anonymous funcs, currying



Chisel3 (frontend) and FIRRTL (backend)

- Goal
  - turn research-ware into a quality compiler platform
  - open up the IR for hardware designers to write their own IR transform passes
- Chisel3
  - embedded in Scala
  - generates FIRRTL RTL code
- FIRRTL (Flexible IR for RTL)
  - serves as an IR for hardware (i.e., LLVM for hw!)
  - generates Verilog
- Success!
  - can add your own transformations
  - you can throw away Chisel and write your own front-end
  - https://github.com/ucb-bar/chisel3 (alpha version)
  - https://github.com/ucb-bar/firrtl (alpha version)
    - **Spec:** https://github.com/ucb-bar/firrtl/tree/master/spec

FIRRTL IR Passes

- Code coverage
  - how much of my circuit is being exercised?
- Scan chain insertion
- SRAM
  - FPGAs vs ASIC memories
  - re-sizing (transforming skinny/tall to rectangular)
- early statistic gathering (e.g., gate count)
  - synthesis tools take a long time...
- Decoupling target time from host time
  - e.g., adding a "stop-the-world" button
- assertion support (TBA)
  - turning simulator assertions into a real HW assertion

New & Upcoming Chisel Features

- parameterized Verilog blackbox support
- Analog types
  - represents outside wires that are "not digital"
  - e.g., connecting **inout** I/Os to blackboxes
- multi-clock domain support
  - fixed-ratio has first-class citizen support
  - you provide your own clock domain crossings*
    - e.g., async FIFOs
- *async reset
  - TBD, but with an eye towards multi-clock support
- DSP support
  - FixedPoint type
  - can provide value ranges, not bit widths
- Annotations
  - allow FIRRTL back-end to get additional information from Chisel

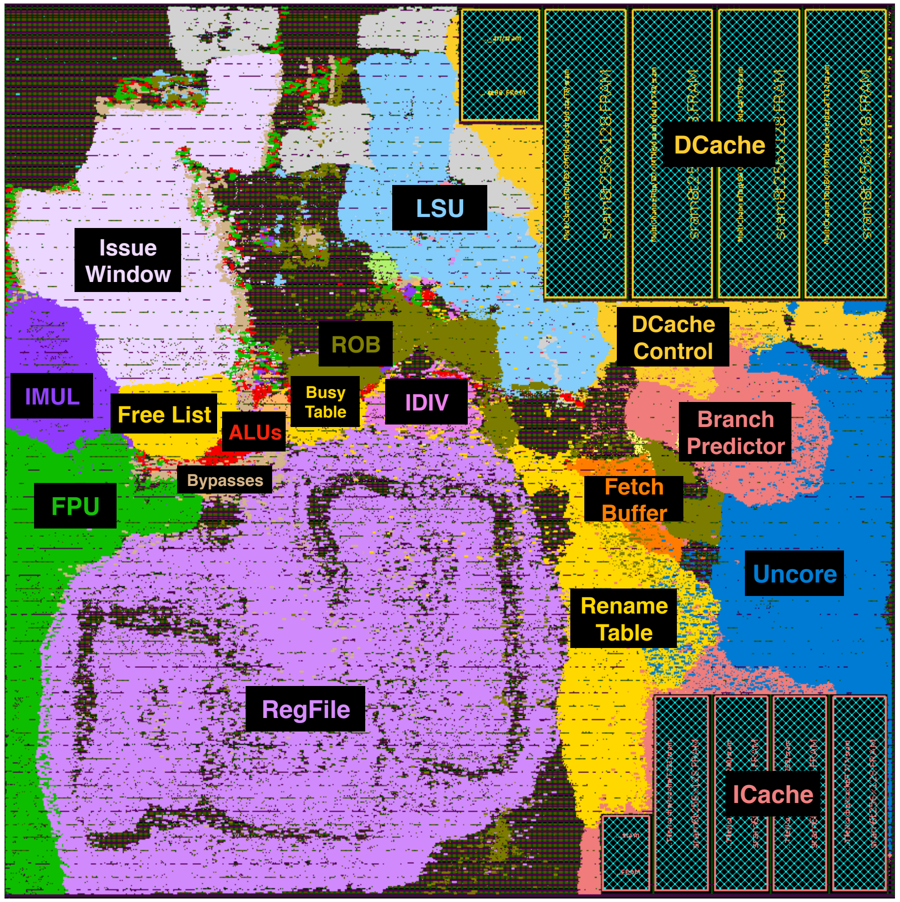
Chisel2

- provides compatibility warnings to help migration to Chisel3
- migration documentation is available
- End of life...?
- if you use Chisel2 and depend on it let us know!

Questions on BAR?

What is BOOM?

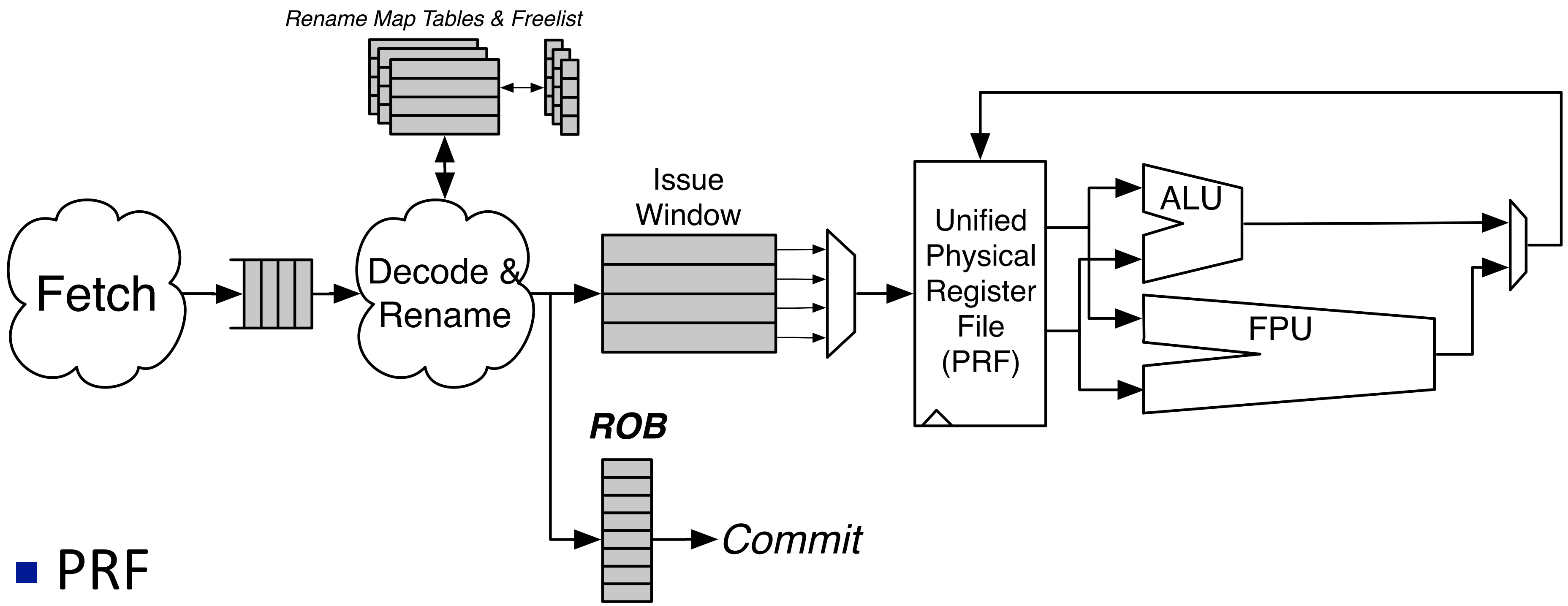
- superscalar, out-of-order RISC-V processor written in Berkeley's Chisel RTL
- It is synthesizable
- It is parameterizable
- it is open-source
- started in **2012**
- **10k** lines of code
- implements **RV64G**



2-wide BOOM (16kB/16kB) 1.2mm^2 @ 45nm

http://ucb-bar.github.io/riscv-boom

BOOM


Rename Map Tables & Freelist
Fetch
Decode & Rename
Issue Window
ROB
Commit
Unified Physical Register File (PRF)
ALU
FPU

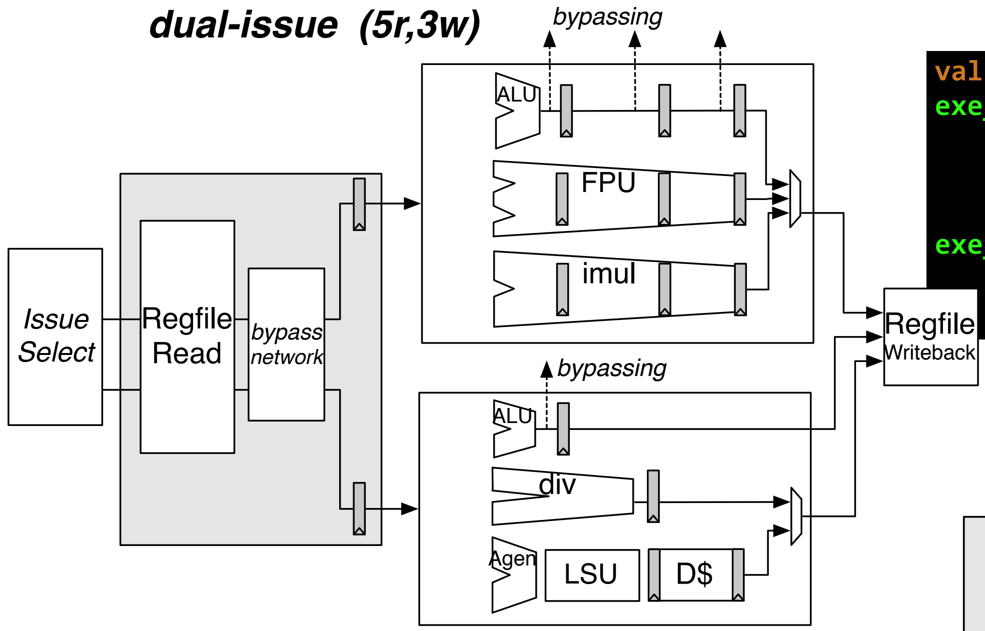
- PRF
  - explicit renaming
  - holds speculative and committed data
  - holds both x-regs, f-regs
- Unified Issue Window
  - holds all instructions
- split ROB/issue window design

Benefits of using Chisel & Rocket-chip

- **~10,000 loc** in BOOM github repo
- + additional ~**12,000** loc instantiated from other libraries
  - **~5,000 loc** from Rocket core repository
    - 90 (integer ALU)
    - 150 (unpipelined mul/div)
    - 550 (floating point units)
    - 1,000 (non-blocking datacache)
    - 300 (icache)
    - 300 (next line predictor/BTB/RAS)
    - 200 (decoder minimization logic)
    - 200 (page-table walker)
    - 200 (TLB)
    - 400 (control/status register file)
    - 300 (instruction definitions + constants)
  - **~4,500 loc** from uncore
    - coherence hubs, L2 caches, networks
  - **~2000 loc** from hardfloat
    - floating point hard units

Parameterized Superscalar

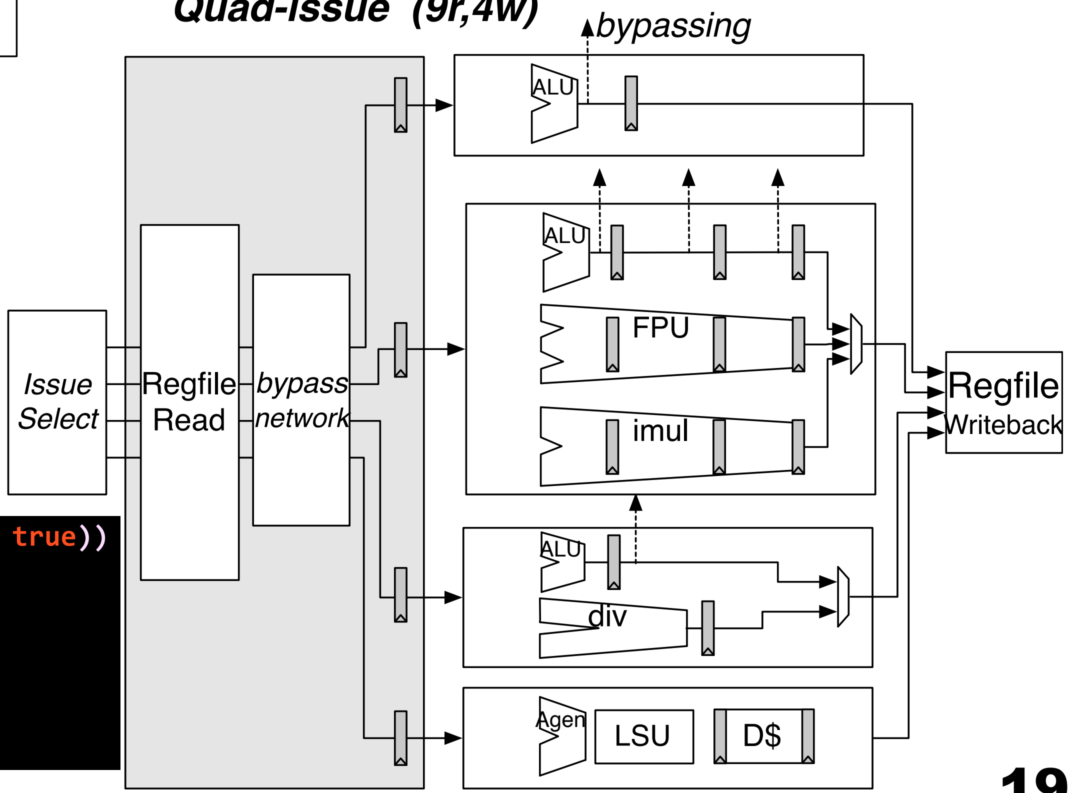
dual-issue (5r,3w)



```
val exe_units = ArrayBuffer[ExecutionUnit]()
exe_units += Module(new ALUExeUnit(is_branch_unit    = true
                                  , has_fpu          = true
                                  , has_mul          = true
                                  ))
exe_units += Module(new ALUMemExeUnit(fp_mem_support = true
                                  , has_div          = true
                                  ))
```

OR

Quad-issue (9r,4w)



```
exe_units += Module(new ALUExeUnit(is_branch_unit = true))
exe_units += Module(new ALUExeUnit(has_fpu = true
                                  , has_mul = true
                                  ))
exe_units += Module(new ALUExeUnit(has_div = true))
exe_units += Module(new MemExeUnit())
```

ARM Cortex-A9 vs. RISC-V BOOM

Category	ARM Cortex-A9	RISC-V BOOM-2w
ISA	32-bit ARM v7	64-bit RISC-V v2 (RV64G)
Architecture	2 wide, 3+1 issue Out-of-Order 8-stage	2 wide, 3 issue Out-of-Order 6-stage
Performance	**3.59** CoreMarks/MHz	**4.61** CoreMarks/MHz
Process	TSMC 40GPLUS	TSMC 40GPLUS
Area with 32K caches	2.5 mm^2	1.00 mm^2
Area efficiency	1.4 CoreMarks/MHz/mm^2	4.6 CoreMarks/MHz/mm^2
Frequency	1.4 GHz	1.5 GHz

Caveats: A9 includes NEON;
BOOM is 64-bit, has IEEE-2008 fused mul-add

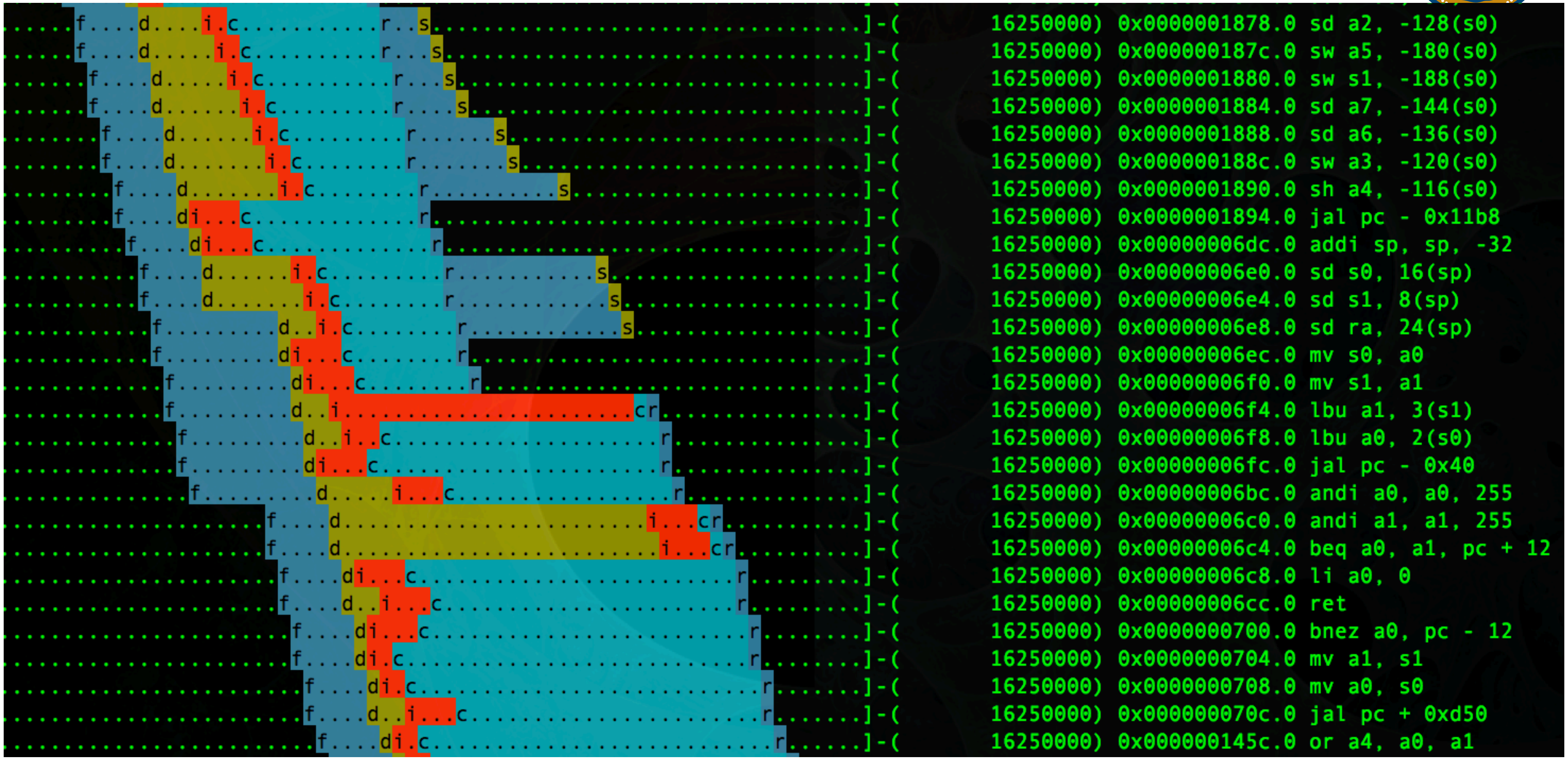
BOOM Updates

- open sourced in Jan 2016
  - http://ucb-bar.github.io/riscv-boom/
- ~60 page BOOM Design Specification
  - https://ccelio.github.io/riscv-boom-doc/
- used for case study in ISCA 2016 paper (strober.org)
- first external contribution (visualizer)
- ported to Chisel3/FIRRTL
- supports uncached loads, stores (allows for memory-mapped IO)
- updated to Privilege Spec v1.9
- supports RISC-V External Debug Spec
- added High Performance Monitor counters (HPM)
- branch predictor improvements
- beginning tape-out

Zynq FPGA Repository

- BOOM runs on a Xilinx Zynq zc706
- https://github.com/ucb-bar/fpga-zynq
- Updated to handle latest Privileged Spec v1.9, External Debug spec, self-booting
- was "tethered" via the Debug Transport Module, but...
- ... just finished up a "Tether Serial Interface" update

Visualization



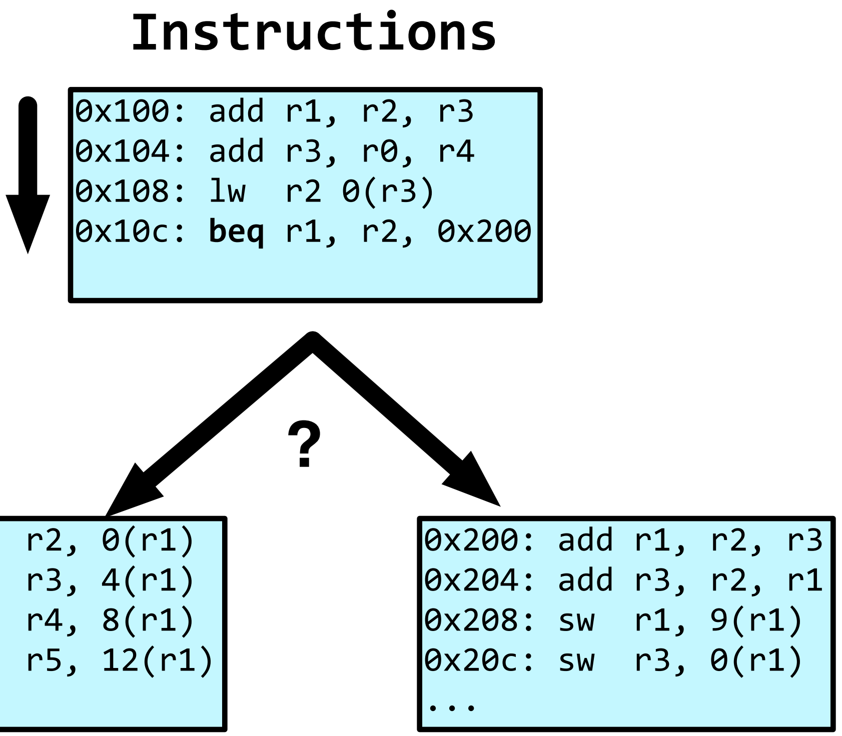
```
......f....d....i.c...........r..s.................................]-(       16250000) 0x0000001878.0 sd a2, -128(s0)
......f....d.....i.c..........r...s................................]-(       16250000) 0x000000187c.0 sw a5, -180(s0)
.......f....d.....i.c..........r...s...............................]-(       16250000) 0x0000001880.0 sw s1, -188(s0)
.......f....d......i.c.........r....s..............................]-(       16250000) 0x0000001884.0 sd a7, -144(s0)
........f....d......i.c.........r......s...........................]-(       16250000) 0x0000001888.0 sd a6, -136(s0)
........f....d.......i.c........r.......s..........................]-(       16250000) 0x000000188c.0 sw a3, -120(s0)
.........f....d.......i.c........r..........s......................]-(       16250000) 0x0000001890.0 sh a4, -116(s0)
.........f....di...c.............r.................................]-(       16250000) 0x0000001894.0 jal pc - 0x11b8
..........f....di...c.............r................................]-(       16250000) 0x00000006dc.0 addi sp, sp, -32
...........f....d......i.c.........r...........s...................]-(       16250000) 0x00000006e0.0 sd s0, 16(sp)
...........f....d.......i.c........r............s..................]-(       16250000) 0x00000006e4.0 sd s1, 8(sp)
............f.........d..i.c........r............s.................]-(       16250000) 0x00000006e8.0 sd ra, 24(sp)
............f.........di...c........r..............................]-(       16250000) 0x00000006ec.0 mv s0, a0
.............f.........di...c........r.............................]-(       16250000) 0x00000006f0.0 mv s1, a1
.............f.........d..i.......................cr...............]-(       16250000) 0x00000006f4.0 lbu a1, 3(s1)
..............f.........d..i..c.....................r..............]-(       16250000) 0x00000006f8.0 lbu a0, 2(s0)
..............f.........di...c......................r..............]-(       16250000) 0x00000006fc.0 jal pc - 0x40
...............f.........d.....i...c.................r.............]-(       16250000) 0x00000006bc.0 andi a0, a0, 255
.....................f....d........................i...cr..........]-(       16250000) 0x00000006c0.0 andi a1, a1, 255
.....................f....d.........................i...cr.........]-(       16250000) 0x00000006c4.0 beq a0, a1, pc + 12
......................f....di...c.........................r........]-(       16250000) 0x00000006c8.0 li a0, 0
......................f....d..i...c.......................r........]-(       16250000) 0x00000006cc.0 ret
.......................f....di...c.........................r.......]-(       16250000) 0x0000000700.0 bnez a0, pc - 12
.......................f....di.c...........................r.......]-(       16250000) 0x0000000704.0 mv a1, s1
........................f....di.c...........................r......]-(       16250000) 0x0000000708.0 mv a0, s0
........................f....d..i...c.......................r......]-(       16250000) 0x000000070c.0 jal pc + 0xd50
.........................f....di.c...........................r.....]-(       16250000) 0x000000145c.0 or a4, a0, a1
```

- Above shows:
  - SD-to-LBU hazard in dhrystone
  - instructions dependent on LBU wait for LBU to execute
  - instructions not-dependent issue out-of-order, before LBU executes

I accept contributions!

- Visualizer is BOOM's first external contribution
- Happy to accept more!
  - code that crashes
  - fixes to code that crashes
  - performance analysis tools
  - debugging or visualization tools
  - performance improvements
  - new features

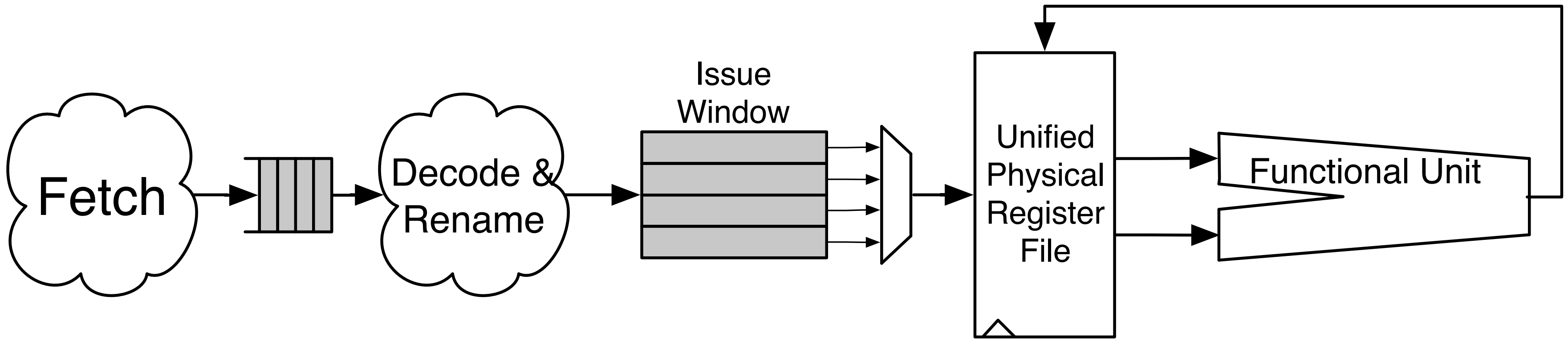
Branch Predictors



```
0x100: add r1, r2, r3
0x104: add r3, r0, r4
0x108: lw  r2 0(r3)
0x10c: beq r1, r2, 0x200
```

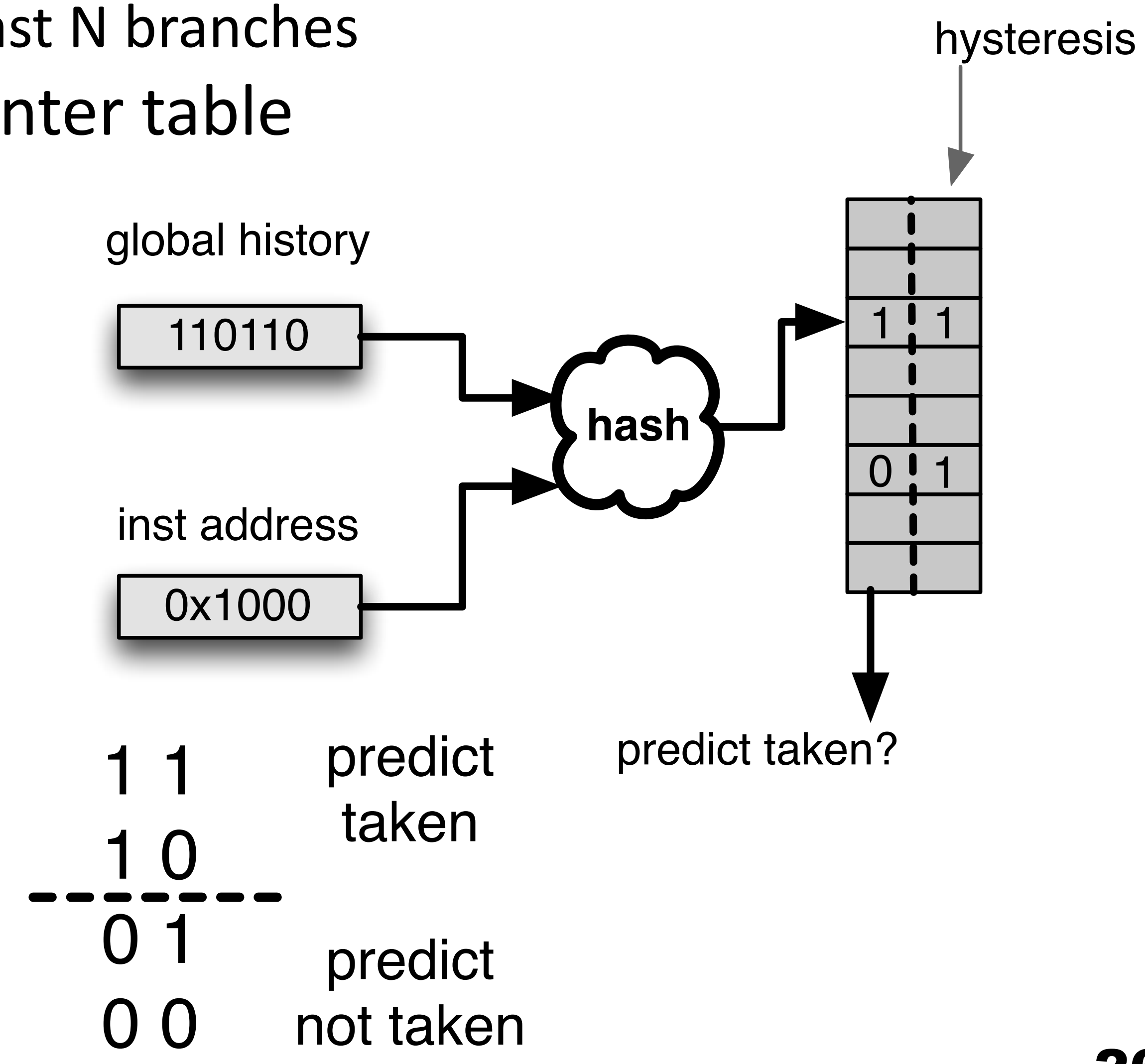
```
0x110: ld  r2, 0(r1)
0x114: ld  r3, 4(r1)
0x118: ld  r4, 8(r1)
0x11c: ld  r5, 12(r1)
0x120: ...
```

```
0x200: add r1, r2, r3
0x204: add r3, r2, r1
0x208: sw  r1, 9(r1)
0x20c: sw  r3, 0(r1)
...
```



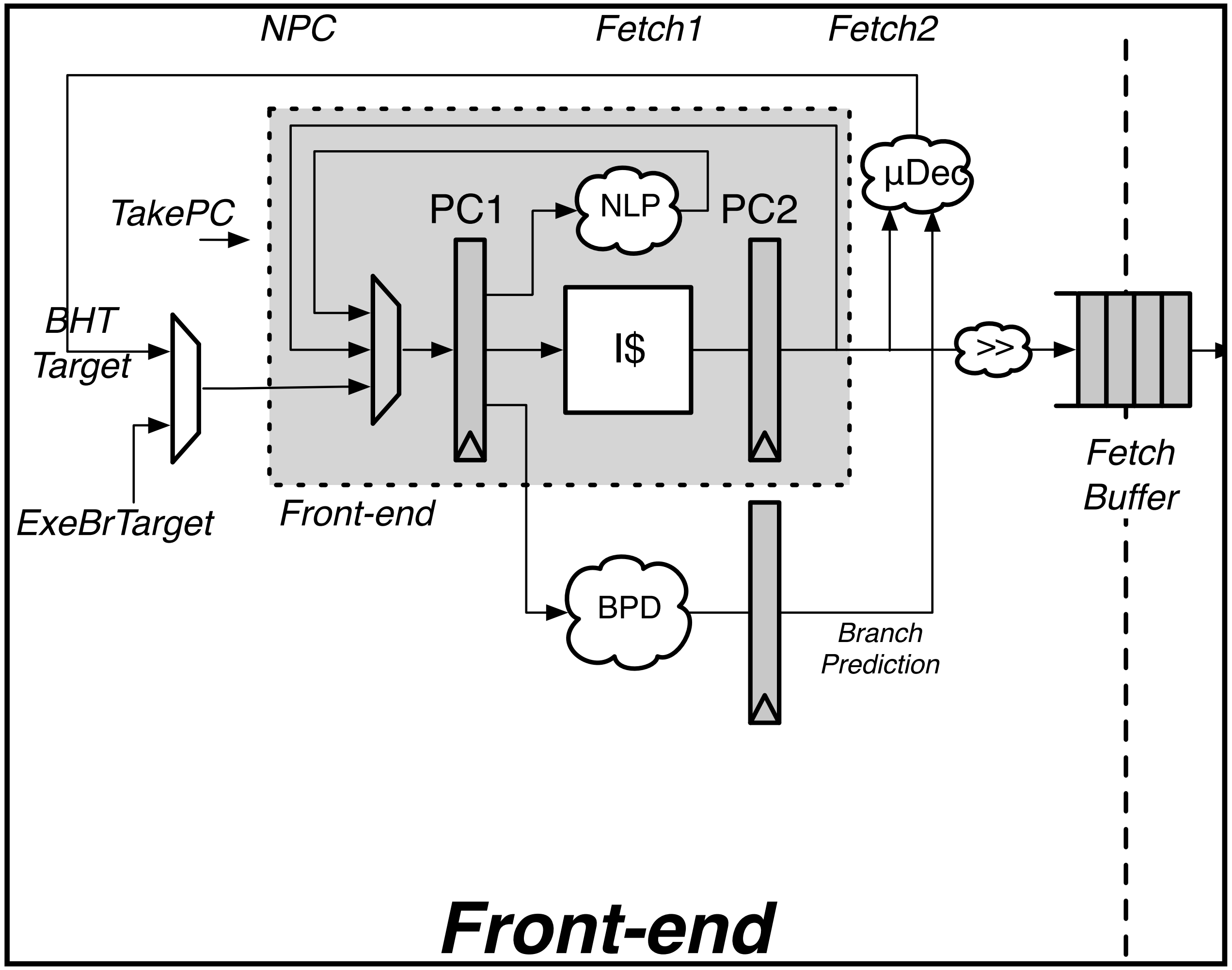
GShare Predictor

- global history
  - track outcome of last N branches
- 2-bit saturating counter table



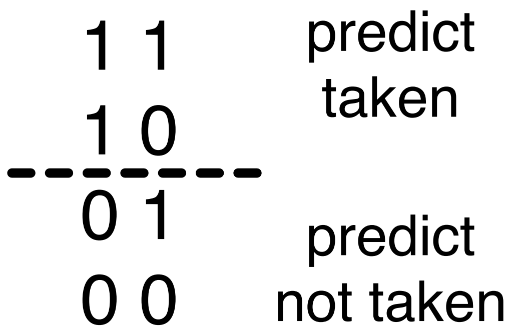
BOOM's Branch Prediction

- next-line predictor (NLP)
  - BTB, BHT, RAS
  - combinational
- backing predictor (BPD)
  - global history predictor
  - SRAM (1 r/w port)



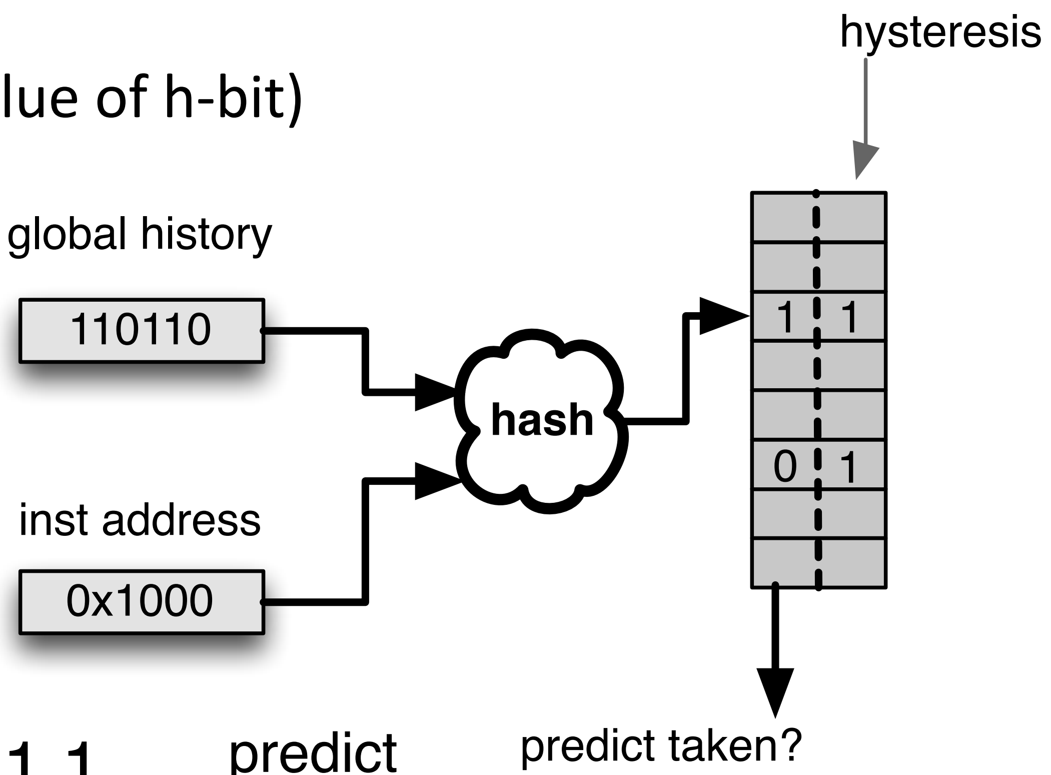
GShare Predictor - Fitting it in SRAM

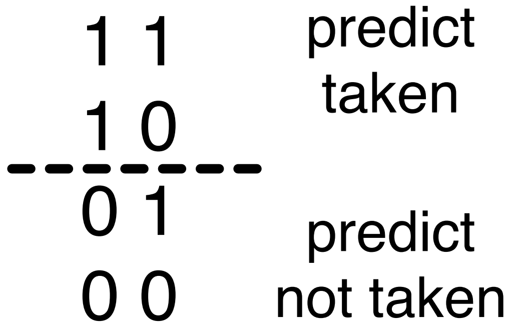
- change "2-bit counter" state machine
  - on misprediction, jump from weak to strong
    - 00 -> 01 -> 11
  - this will allow us to reduce the read/write requirements
  - described in Hennessy & Patterson computer architecture textbook



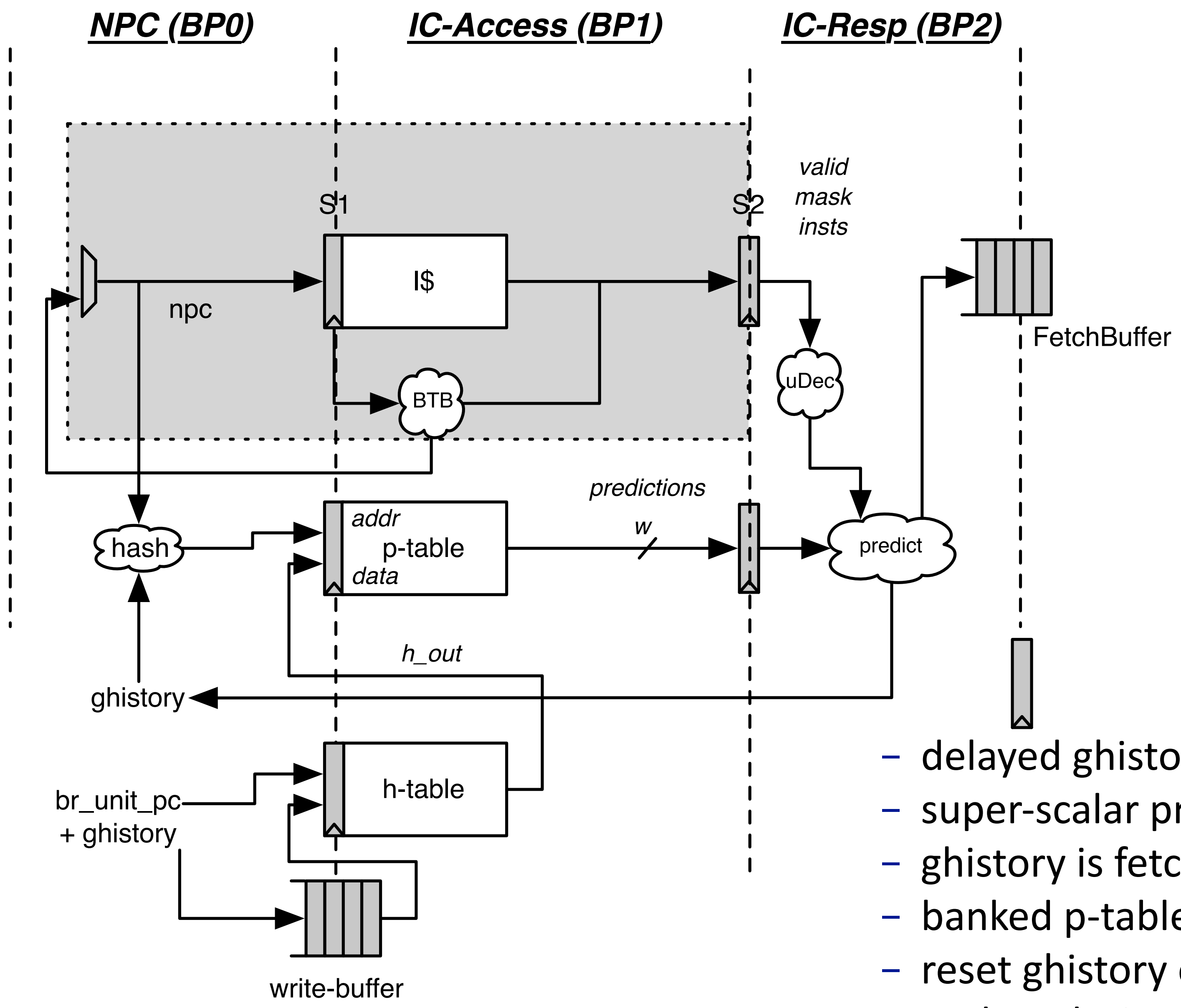
GShare Predictor - Fitting it in SRAM

- p-bit (prediction)
  - read every cycle
  - write on mispredict (value of h-bit)
- h-bit (hysteresis)
  - write every branch
  - read on mispredict



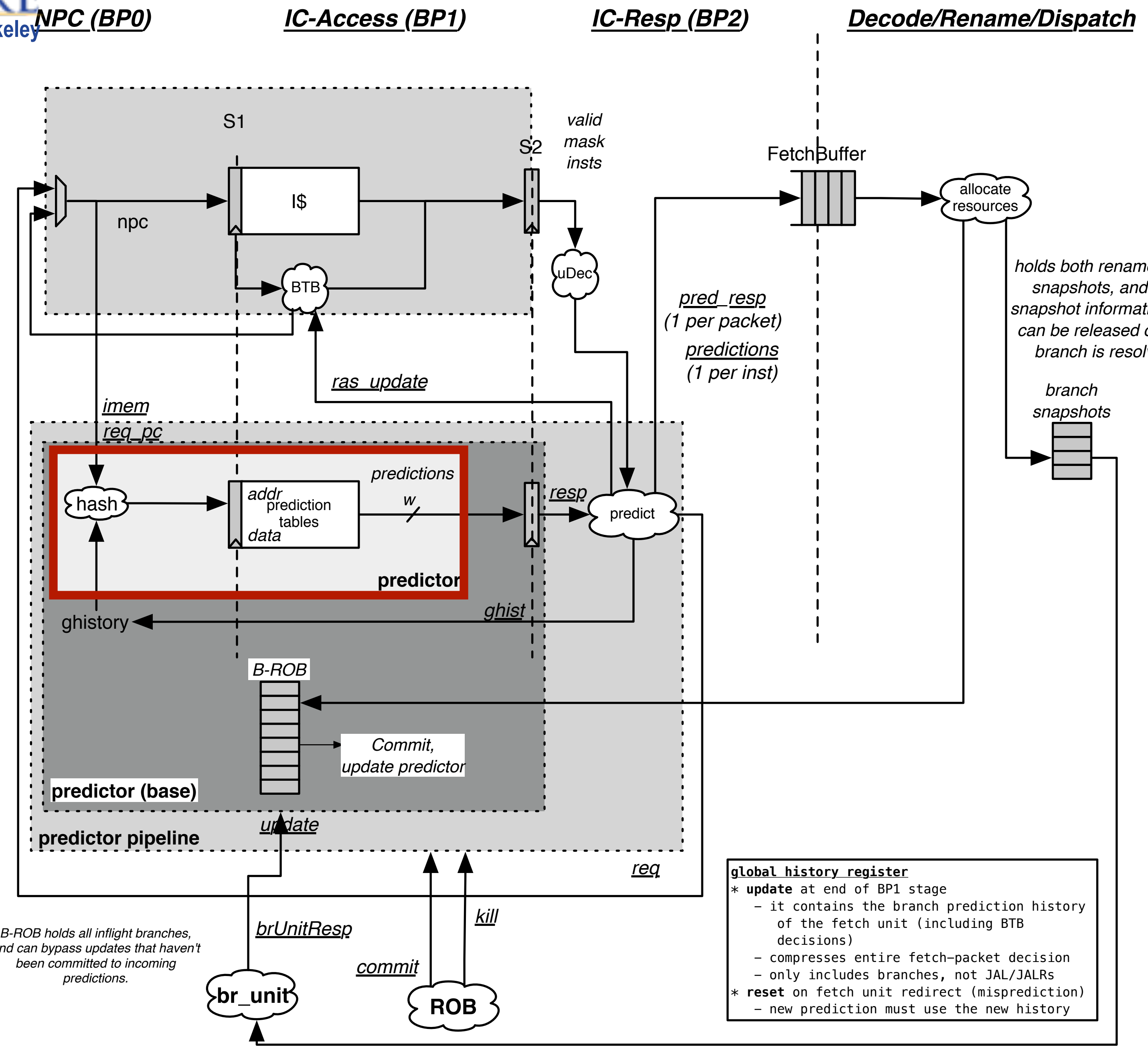


GShare in single-ported SRAM



- delayed ghistory update
- super-scalar predictions
- ghistory is fetch packet granularity
- banked p-table
- reset ghistory on misspeculations
- update during commit

Abstract Branch Predictors



```
global history register
* update at end of BP1 stage
   - it contains the branch prediction history
     of the fetch unit (including BTB
     decisions)
   - compresses entire fetch-packet decision
   - only includes branches, not JAL/JALRs
* reset on fetch unit redirect (misprediction)
   - new prediction must use the new history
```

BOOM's Branch Predictors

- Null
  - predicts not-taken
- Random
  - serves as the baseline worst-case predictor
  - useful for testing the pipeline
- Simple Gshare
  - demonstrates how to interface with the branch predictor framework
  - not synthesizable
- GShare
  - targeting 1r/1w SRAM (dualported)...
  - ... or 1rw SRAM (banked)
- 2bc-GSkew
  - based on the EV8 (Alpha 21464) predictor
  - 1rw (or 1r/1w) SRAM
  - took 12 hours to implement
- TAGE
  - a super awesome predictor
  - still in prototyping; WIP to make it synthesizable and scalable

Thank you!

- Source
  - https://ucb-bar.github.io/riscv-boom
- Documentation
  - https://ccelio.github.io/riscv-boom-doc
- Google group
  - https://groups.google.com/forum/#!forum/riscv-boom
- Tech Report
  - https://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html
- Twitter
  - https://twitter.com/boom_cpu

Funding Acknowledgements

- *Research partially funded by DARPA Award Number HR0011-12-2-0016, the Center for Future Architecture Research, a member of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and ASPIRE Lab industrial sponsors and affiliates Intel, Google, Huawei, Nokia, NVIDIA, Oracle, and Samsung.*
- *Approved for public release; distribution is unlimited. The content of this presentation does not necessarily reflect the position or the policy of the US government and no official endorsement should be inferred.*
- *Any opinions, findings, conclusions, or recommendations in this paper are solely those of the authors and does not necessarily reflect the position or the policy of the sponsors.*