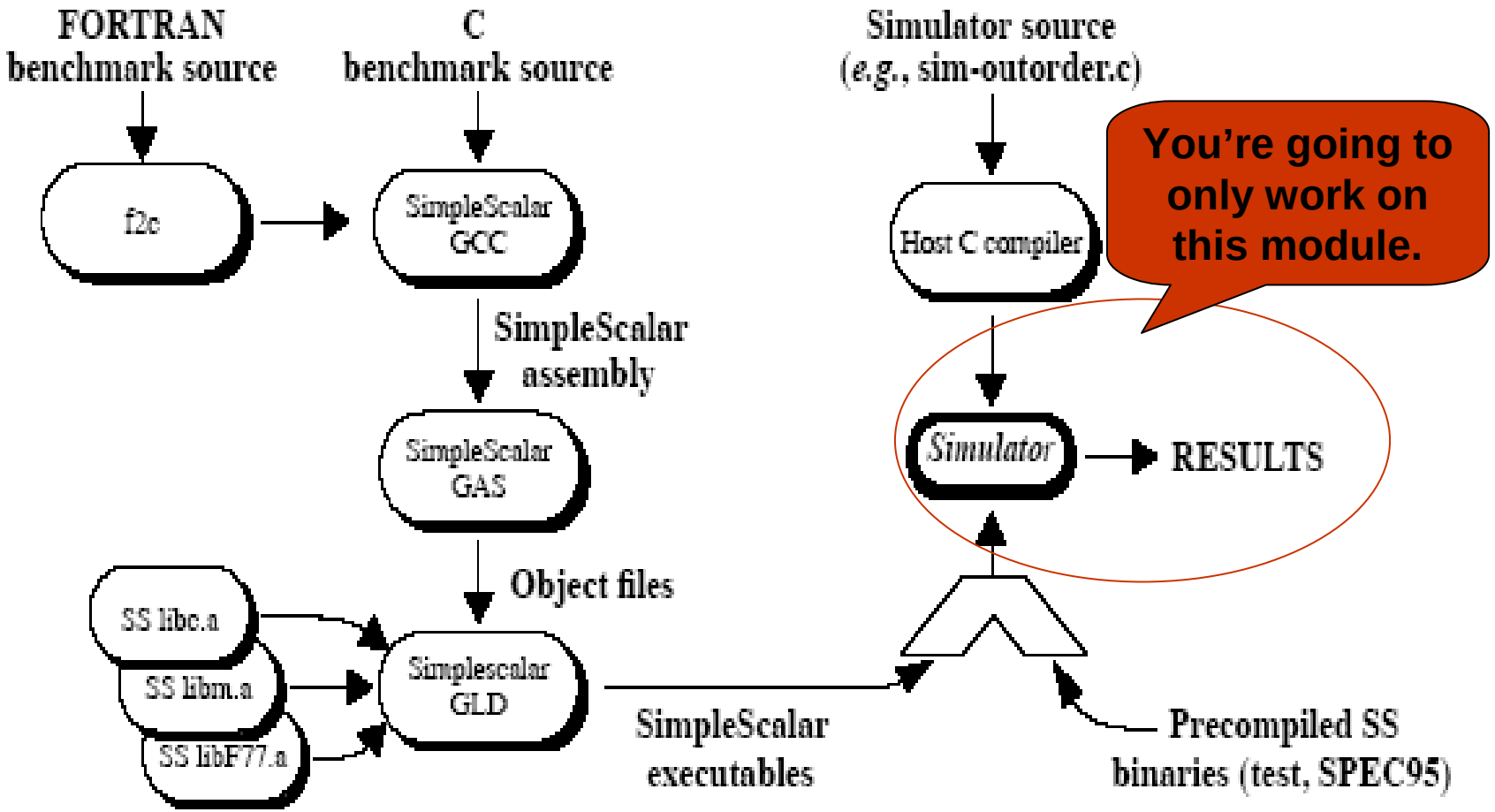
A Quick SimpleScalar Tutorial

[Prepared from SimpleScalar Tutorial v2, Todd Austin and Doug Burger.]

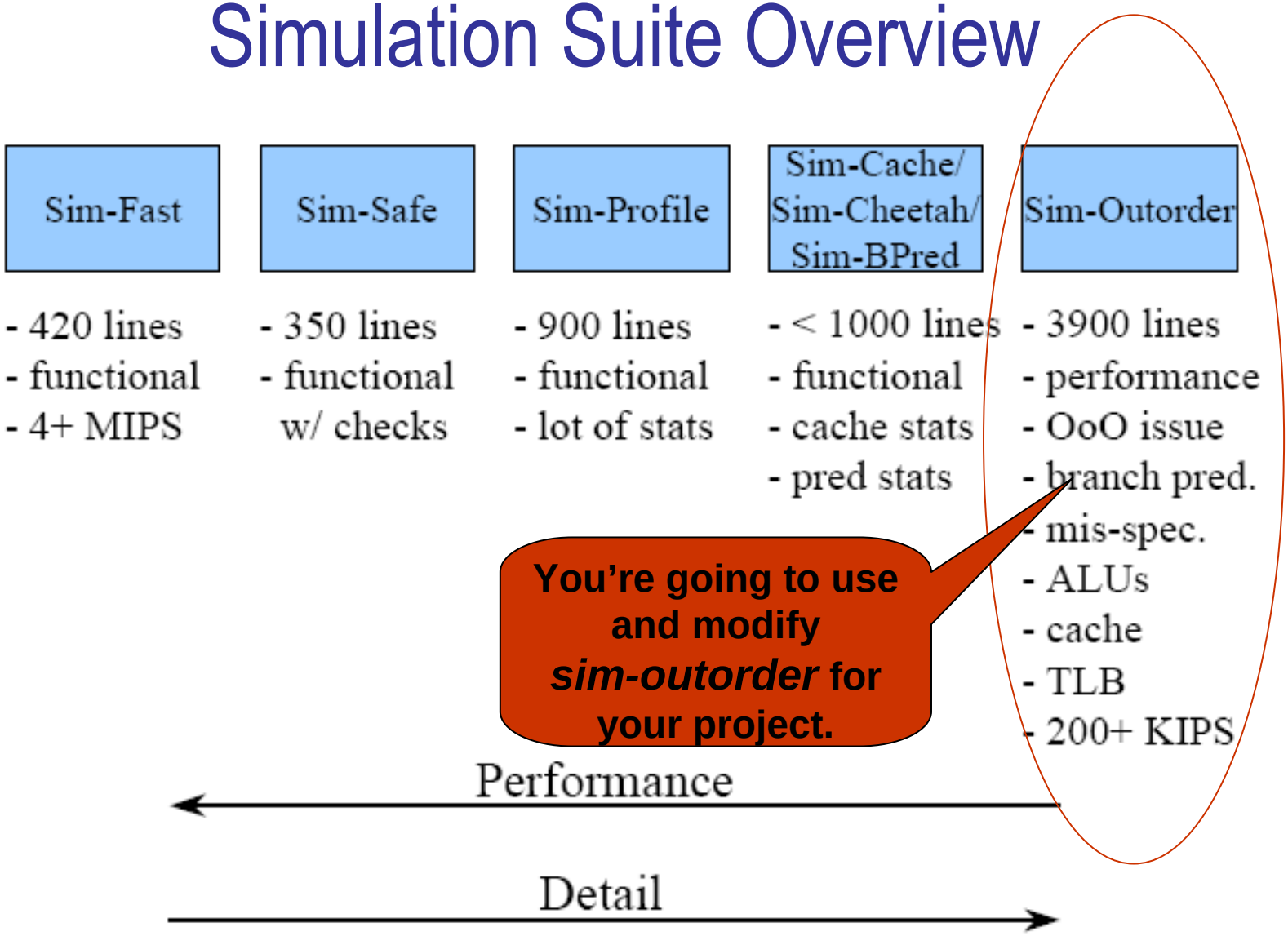
What is SimpleScalar?

- SimpleScalar is a tool set for high-performance simulation and research of modern microprocessors
  - SimpleScalar compiler (C, Fortran)
  - Functional/performance simulators
- Sim-outorder: full performance simulator
  - Execution-driven simulator
  - Simulates 6-stage out-of-order issue/execution in-order commit superscalar microprocessors

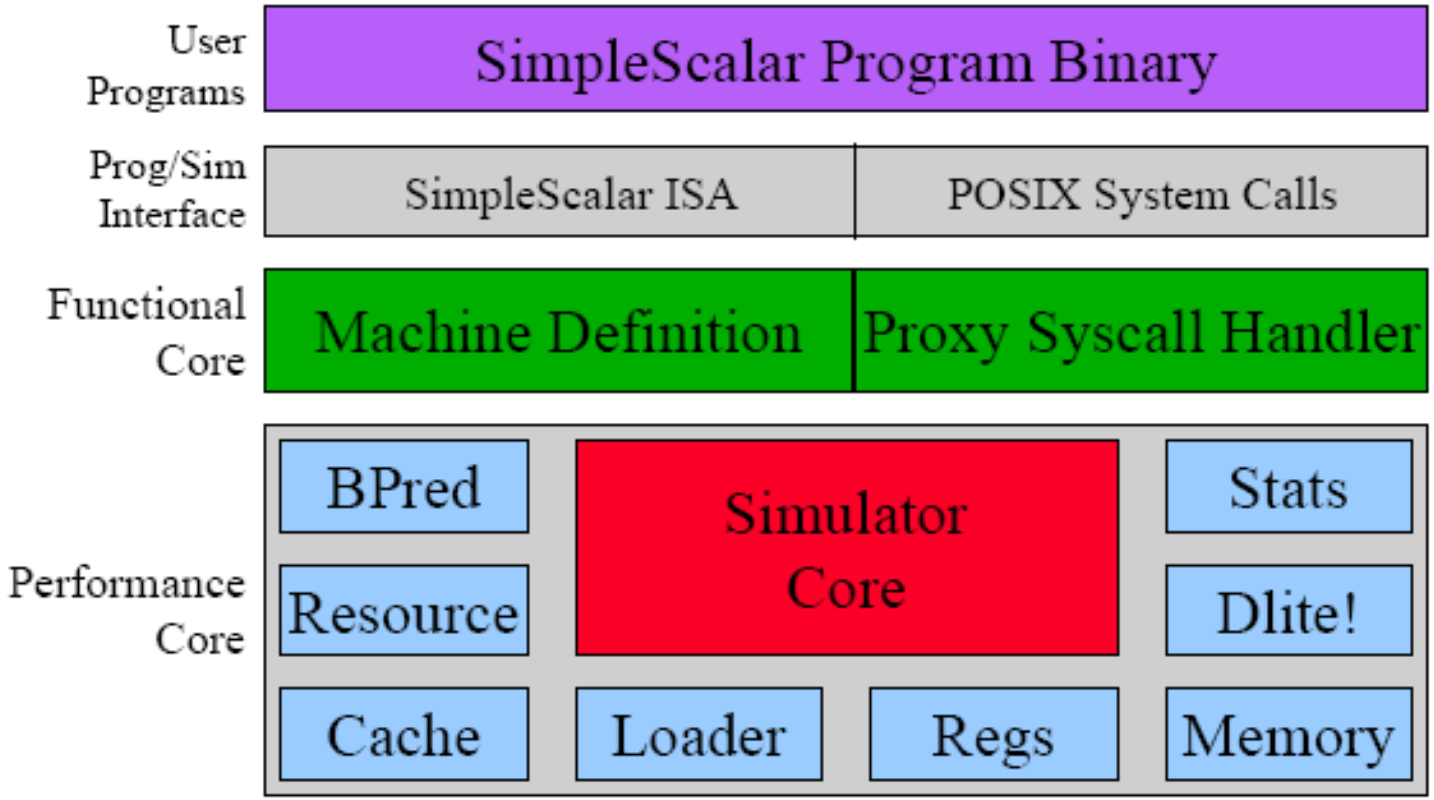
SimpleScalar Tool Set Overview



Simulation Suite Overview

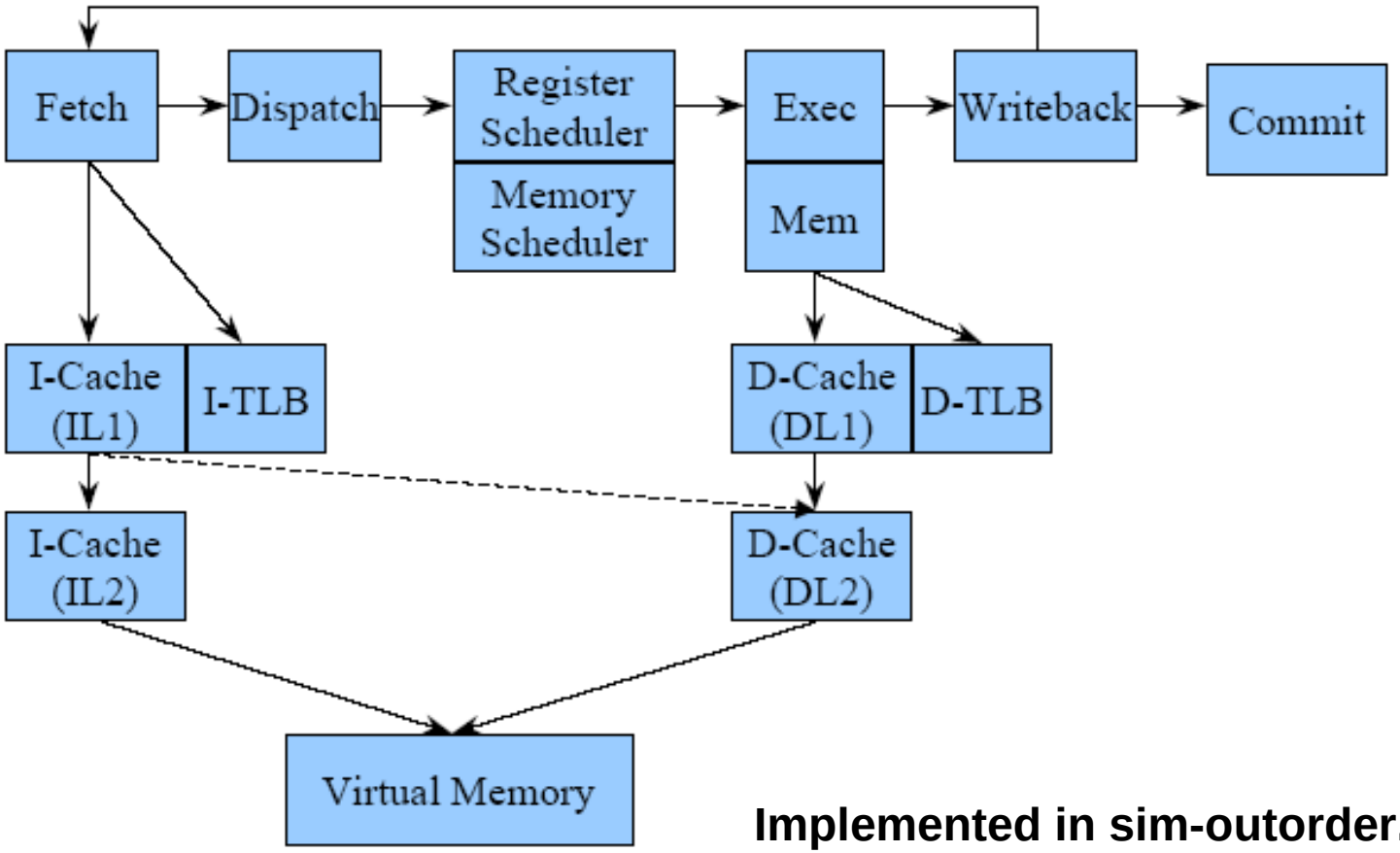


Simulator S/W Architecture



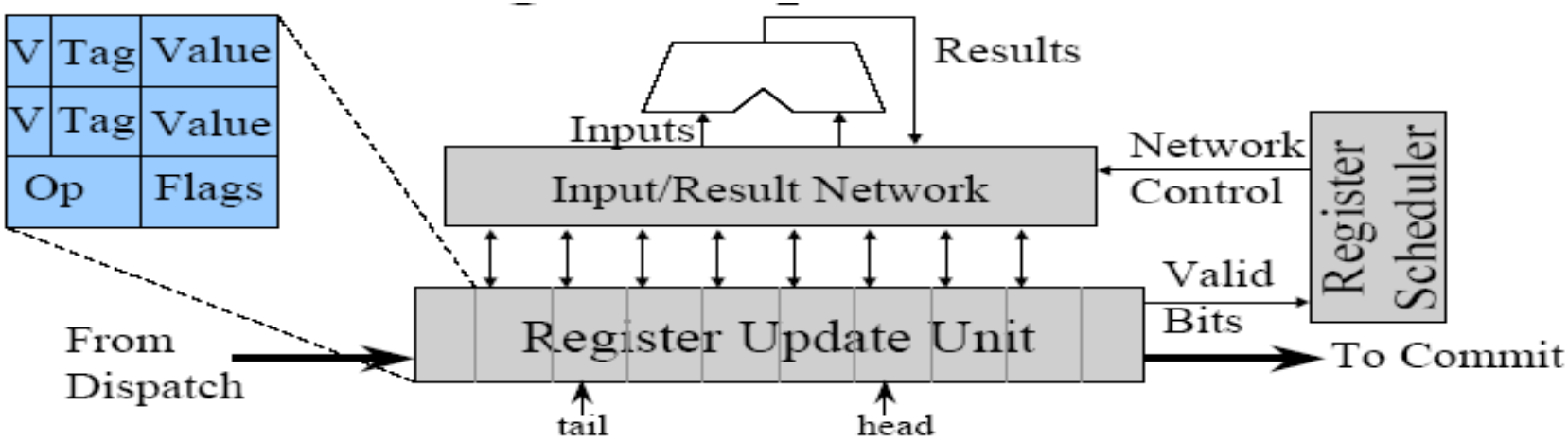
- most of performance core is optional
- most projects will enhance on the “simulator core”

Sim-outorder: H/W Architecture



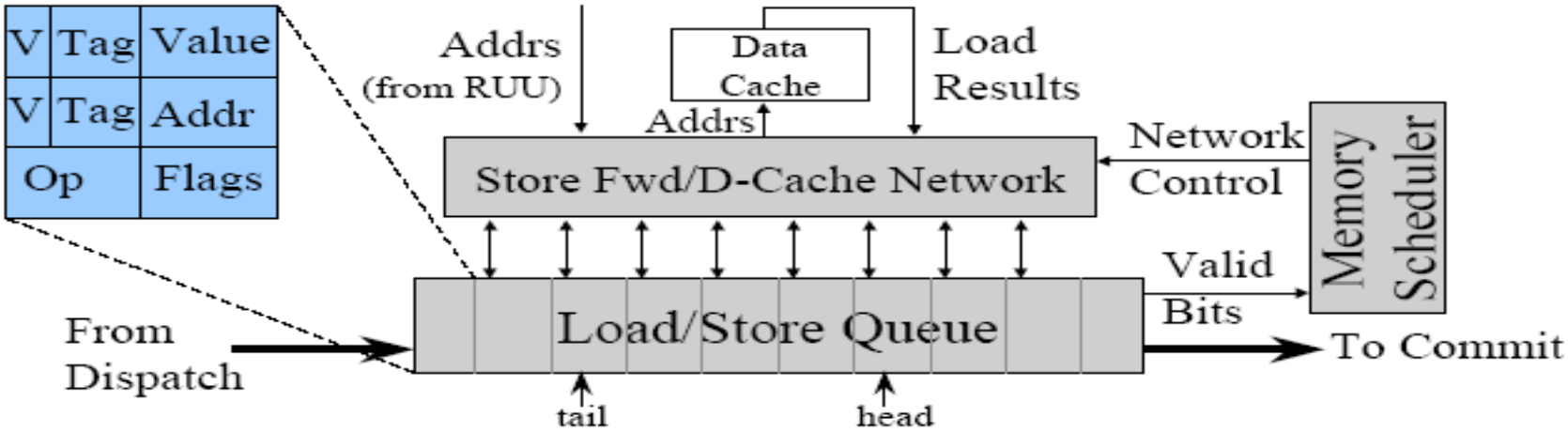
Implemented in sim-outorder.c

The Register Update Unit (RUU)



- RUU handles register synchronization/communication
  - unifies reorder buffer and reservation stations
    - managed as a circular queue
    - entries allocated at Dispatch, deallocated at Commit
  - out-of-order issue, when register and memory deps satisfied
    - memory dependencies resolved by load/store queue (LSQ)

The Load/Store Queue (LSQ)



- LSQ handles memory synchronization/communication
  - contains all loads and stores in program order
    - load/store primitives really, address calculation is separate op
    - effective address calculations reside in RUU (as ADD insts)
  - loads issue out-of-order, when memory deps known satisfied
    - load addr known, source data identified, no unknown store address

The Main Simulator Loop

```
for (;;) {
  ruu_commit();
  ruu_writeback();
  lsq_refresh();
  ruu_issue();
  ruu_dispatch();
  ruu_fetch();
}
```

- main simulator loop is implemented in `sim_main()`
- walks pipeline from Commit to Fetch
  - backward pipeline traversal eliminates relaxation problems, e.g., provides correct inter-stage latch synchronization
- loop is exited via a `longjmp()` to `main()` when simulated program executes an `exit()` system call

sim-outorder: a detailed performance simulator

- generates timing statistics for a detailed out-of-order issue processor core with two-level cache memory hierarchy and main memory
- extra options:

`-fetch:ifqsize <size>` - instruction fetch queue size (in insts)

`-fetch:mplat <cycles>` - extra branch mis-prediction latency (cycles)

`-bpred <type>` - specify the branch predictor

`-decode:width <insts>` - decoder bandwidth (insts/cycle)

`-issue:width <insts>` - RUU issue bandwidth (insts/cycle)

`-issue:inorder` - constrain instruction issue to program order

`-issue:wrongpath` - permit instruction issue after mis-speculation

`-ruu:size <insts>` - capacity of RUU (insts)

`-lsq:size <insts>` - capacity of load/store queue (insts)

`-cache:dl1 <config>` - level 1 data cache configuration

`-cache:dl1lat <cycles>` - level 1 data cache hit latency

sim-outorder: a detailed performance simulator

`-cache:dl2 <config>`	- level 2 data cache configuration
`-cache:dl2lat <cycles>`	- level 2 data cache hit latency
`-cache:il1 <config>`	- level 1 instruction cache configuration
`-cache:il1lat <cycles>`	- level 1 instruction cache hit latency
`-cache:il2 <config>`	- level 2 instruction cache configuration
`-cache:il2lat <cycles>`	- level 2 instruction cache hit latency
`-cache:flush`	- flush all caches on system calls
`-cache:icompress`	- remap 64-bit inst addresses to 32-bit equiv.
`-mem:lat <1st> <next>`	- specify memory access latency (first, rest)
`-mem:width`	- specify width of memory bus (in bytes)
`-tlb:itlb <config>`	- instruction TLB configuration
`-tlb:dtlb <config>`	- data TLB configuration
`-tlb:lat <cycles>`	- latency (in cycles) to service a TLB miss

sim-outorder: a detailed performance simulator

`-res:ialu`	- specify number of integer ALUs
`-res:imult` multiplier/dividers	- specify number of integer
`-res:memports`	- specify number of first-level cache ports
`-res:fpalu`	- specify number of FP ALUs
`-res:fpmult`	- specify number of FP multiplier/dividers
`-pcstat <stat>`	- record statistic `<stat>` by text address
`-ptrace <file> <range>`	- generate pipetrace

Specifying Cache Configurations

- all caches and TLB configurations specified with same format:

  `<name>:<nsets>:<bsize>:<assoc>:<repl>`

- where:

  `<name>` - cache name (make this unique)

  `<nsets>` - number of sets

  `<assoc>` - associativity (number of “ways”)

  `<repl>` - set replacement policy

  - `l` - for LRU
  - `f` - for FIFO
  - `r` - for RANDOM

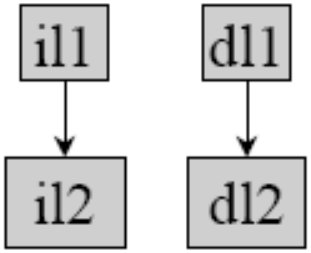
- examples:

  `il1:1024:32:2:l` 2-way set-assoc 64k-byte cache, LRU

  `dtlb:1:4096:64:r` 64-entry fully assoc TLB w/ 4k pages,random replacement

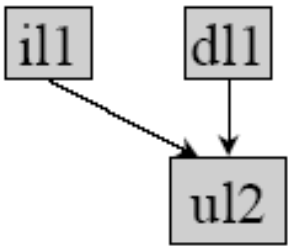
Specifying Cache Hierarchies

- specify all cache parameters in no unified levels exist, e.g.,




```
-cache:il1 il1:128:64:1:l -cache:il2
   il2:128:64:4:l
-cache:dl1 dl1:256:32:1:l -cache:dl2
   dl2:1024:64:2:l
```

- to unify any level of the hierarchy, “point” an I-cache level into the data cache hierarchy:




```
-cache:il1 il1:128:64:1:l -cache:il2 dl2
-cache:dl1 dl1:256:32:1:l -cache:dl2
   ul2:1024:64:2:l
```

Specifying the Branch Predictor

- specifying the branch predictor type:

  `-bpred <type>`

- the supported predictor types are:

  `nottaken` always predict not taken

  `taken` always predict taken

  `perfect` perfect predictor

  `bimod` bimodal predictor (BTB w/ 2 bit counters)

  `2lev` 2-level adaptive predictor

- configuring the bimodal predictor (only useful when "`-bpred bimod`" is specified):

  `-bpred:bimod <size>` size of direct-mapped BTB

Specifying the Branch Predictor

- configuring the 2-level adaptive predictor (only useful when "`-bpred 2lev`" is specified):

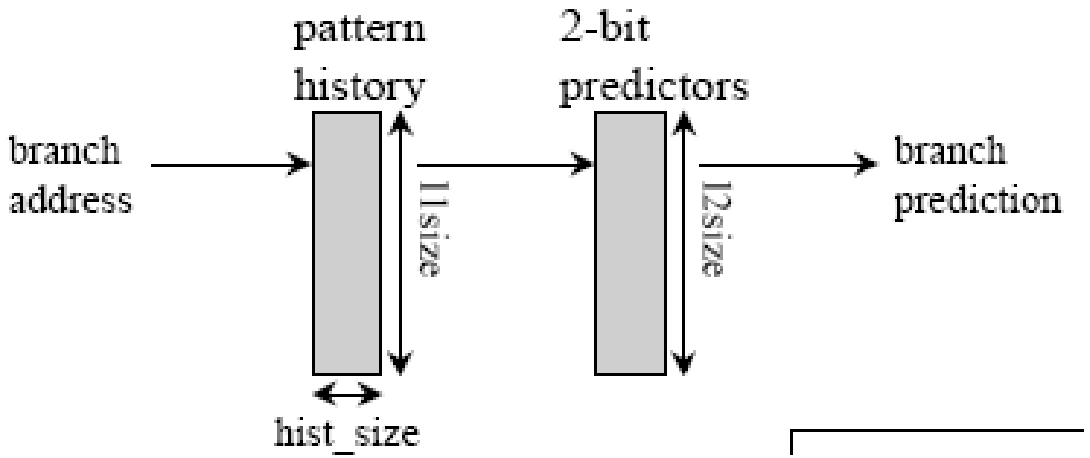
  `-bpred:2lev <l1size> <l2size> <hist_size>`

- where:

  `<l1size>` size of the first level table

  `<l2size>` size of the second level table

  `<hist_size>` history (pattern) width



Run Simulation

- Command line
  - `sim-outorder [simulator opts] benchmark_name [bench opts]`