Fast Out-of-Order Processor Simulation Using Memoization

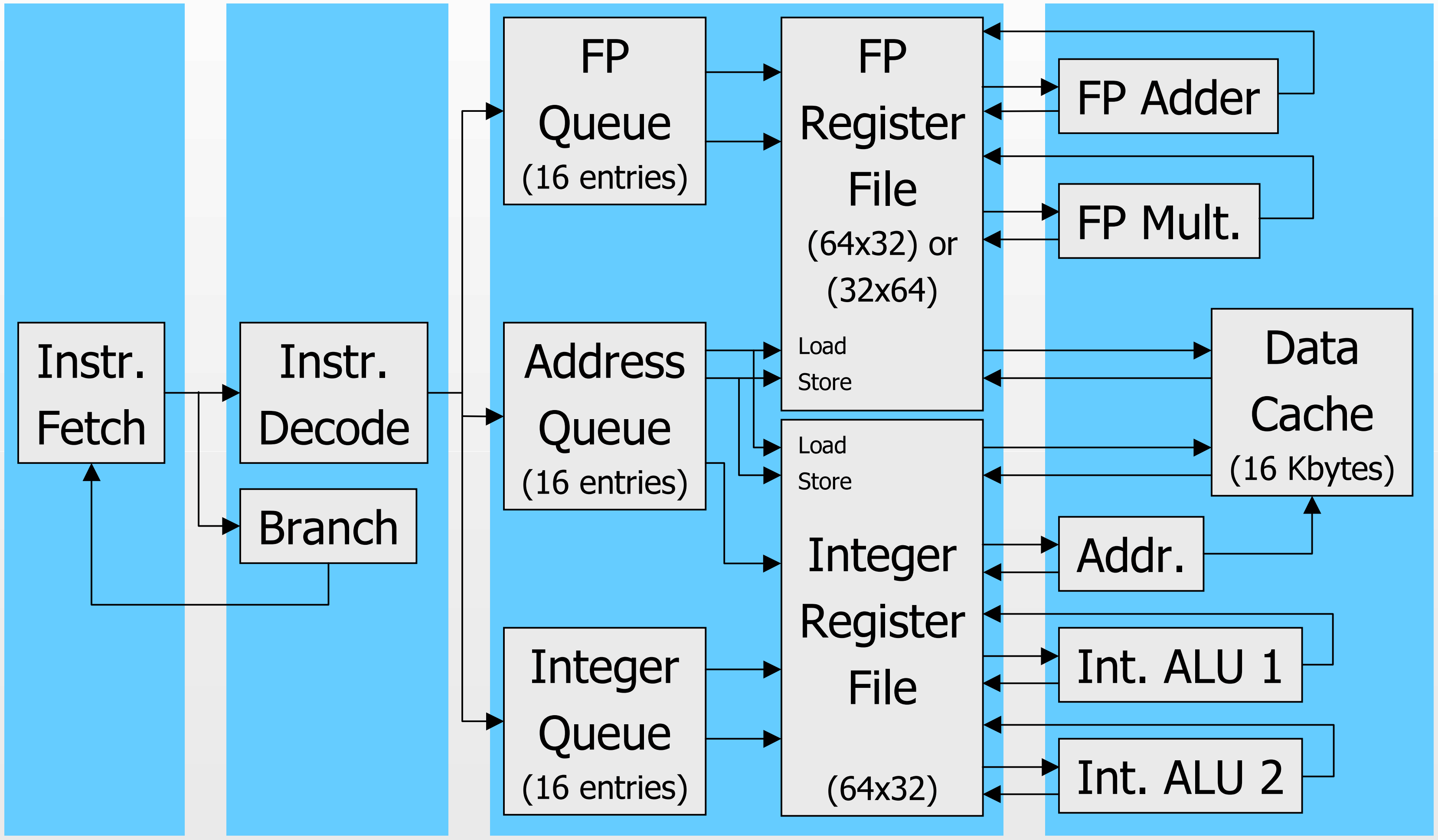
by Eric Schnarr and James R. Larus

University of Wisconsin-Madison
Wisconsin Wind Tunnel Project

Out-of-Order Simulators Are Slow

- SimpleScalar — 4,000 times slowdown
- RSIM — 10,000-15,000 times slowdown
- MXS — “several thousand times slowdown”

Out-of-Order Processors

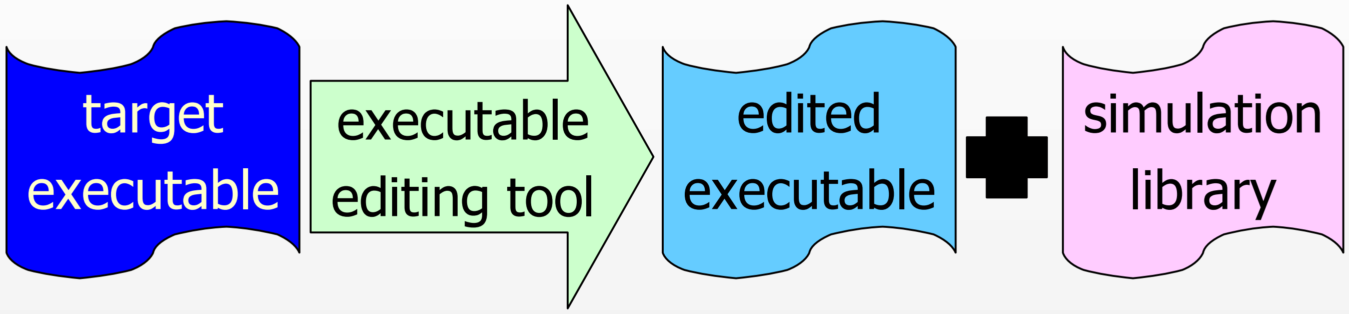


New Simulator: “FastSim”

- Direct execution improves instruction emulation
- Memoization speeds pipeline simulation

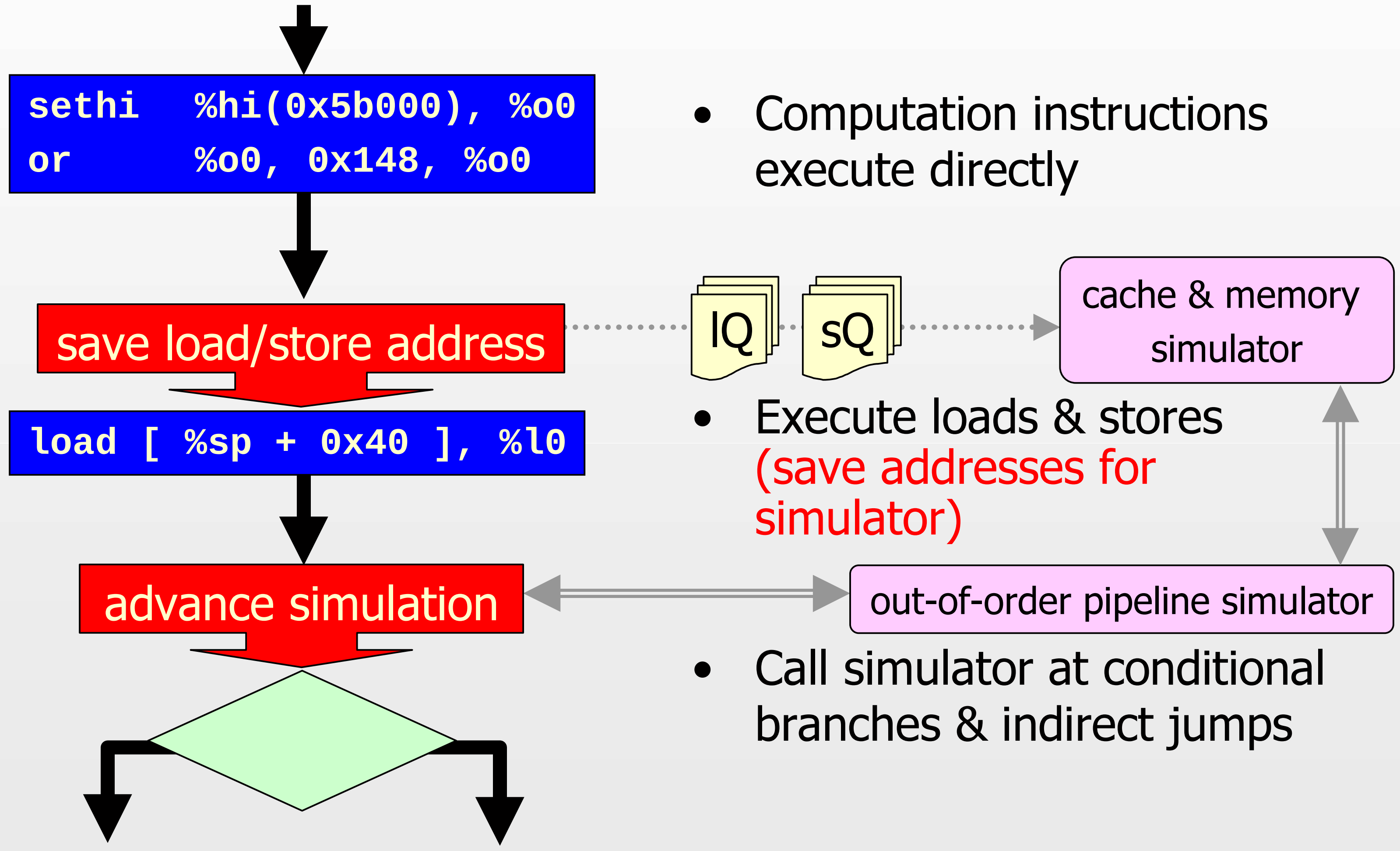
- *8-15 times faster than SimpleScalar*
- *No loss of accuracy!*

Direct Execution



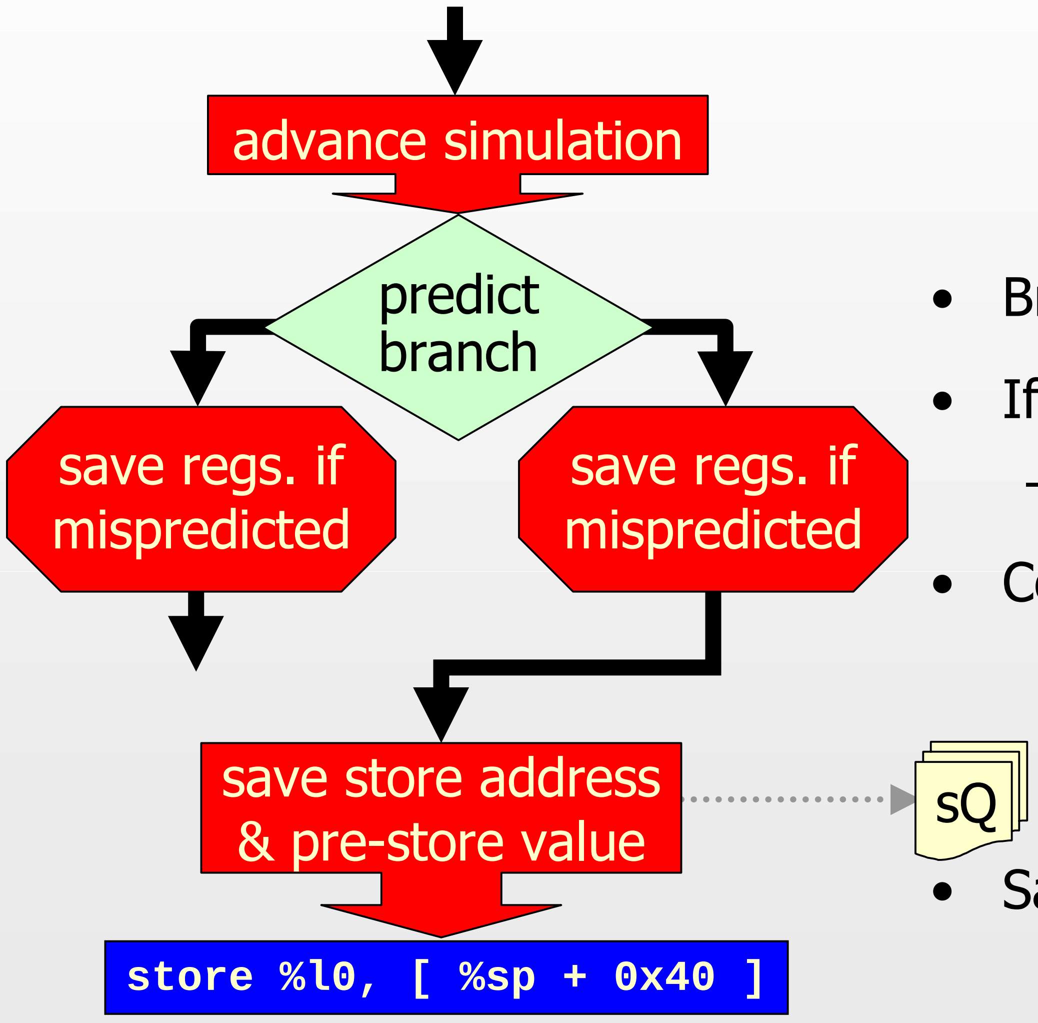
- Two problems
  - Out-of-Order simulation with static instrumentation
  - Speculative execution

Direct Execution vs. Out-of-Order



- Computation instructions execute directly
- Execute loads & stores (save addresses for simulator)
- Call simulator at conditional branches & indirect jumps

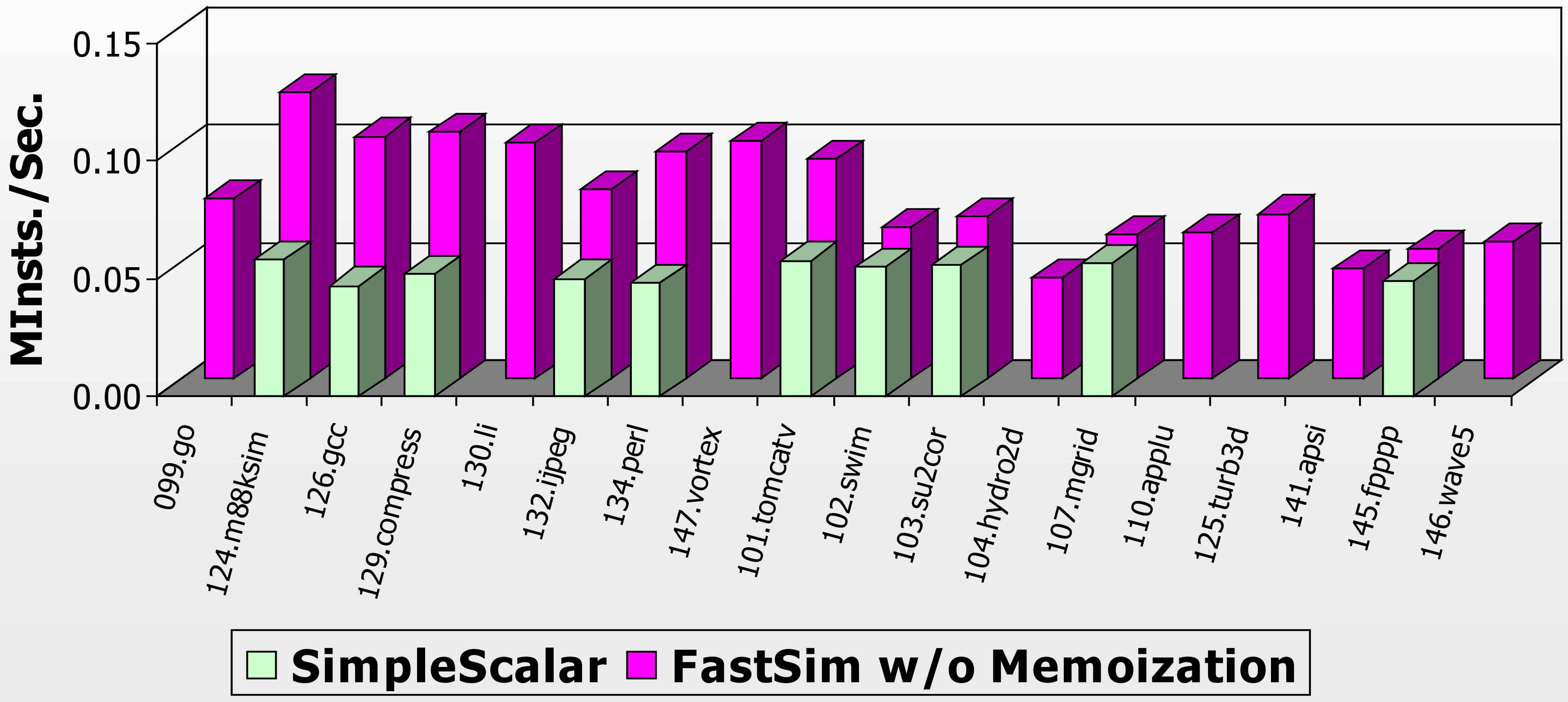
Speculative Direct Execution


advance simulation
predict branch
save regs. if mispredicted
save regs. if mispredicted
save store address & pre-store value
sQ

```
store %l0, [ %sp + 0x40 ]
```

- Branch based on prediction
- If mispredicted, save regs
  - if correct, do nothing
- Continue direct execution
- Save pre-store values

Performance vs. SimpleScalar



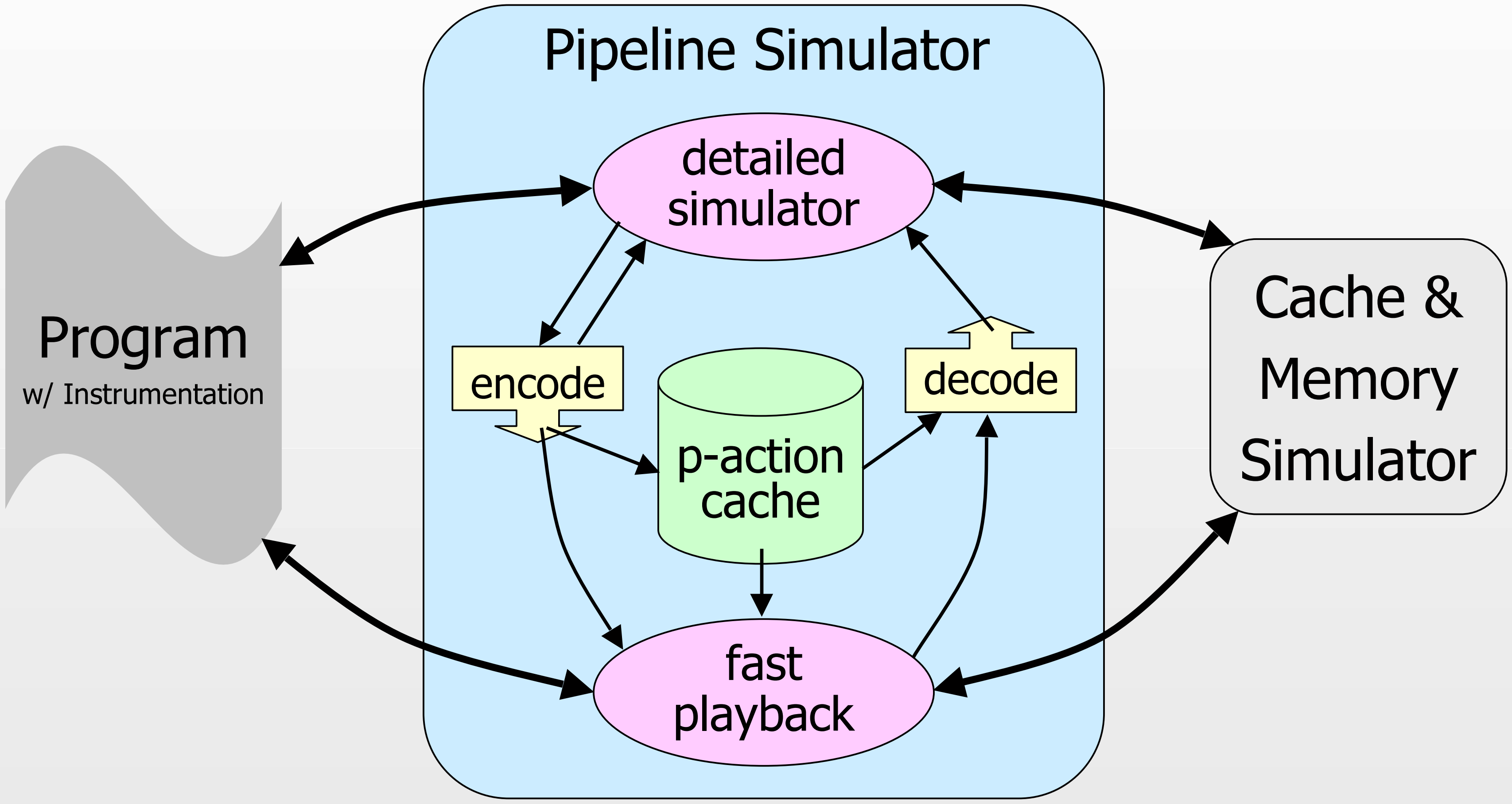
Limitations

- Target ISA same as host ISA
- Single target processor
  - Shared memory values are timing dependent
- Coarse-grained speculation
  - Value prediction may be difficult
- Small performance improvement
  - Most time spent simulating out-of-order pipeline

What is Memoization?

- Functional programming language optimization
- Cache each function’s (input, output) pairs
  - Use saved value instead of recomputing function
- FastSim’s Fast-Forwarding:
  - Inputs: pipeline state + branch & cache behavior
  - Outputs: processor state changes + timing info.

FastSim Simulator Components



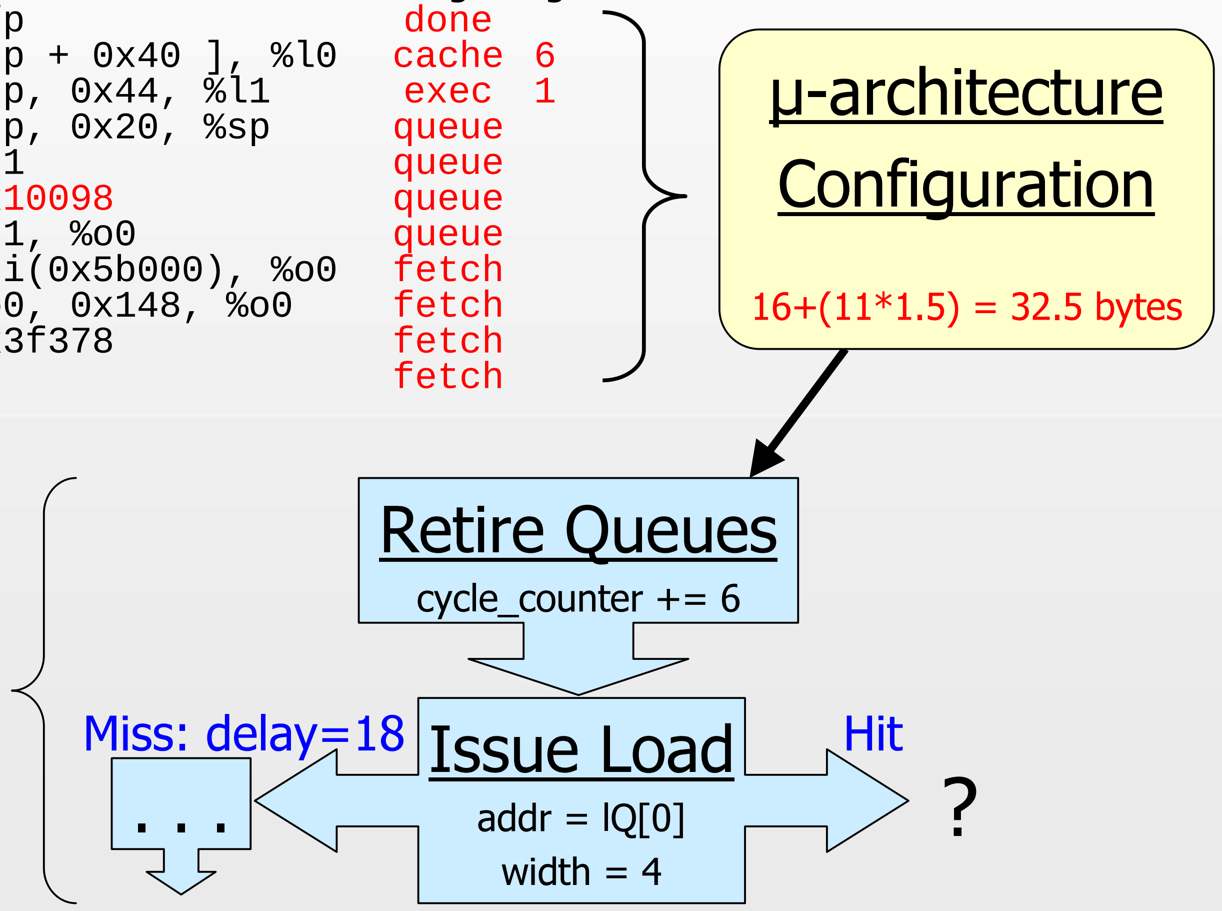
Simulator Instruction Queue (iQ)

Addr.	Instruction	Tag1	Tag2
0x10074	clr %fp	done	
0x10078	ld [%sp + 0x40], %l0	cache	6
0x1007c	add %sp, 0x44, %l1	exec	1
0x10080	sub %sp, 0x20, %sp	queue	
0x10084	tst %g1	queue	
0x10088	be 0x10098	queue	
0x1008c	mov %g1, %o0	queue	
0x10098	sethi %hi(0x5b000), %o0	fetch	
0x1009c	or %o0, 0x148, %o0	fetch	
0x100a0	call 0x3f378	fetch	
0x100a4	nop	fetch	

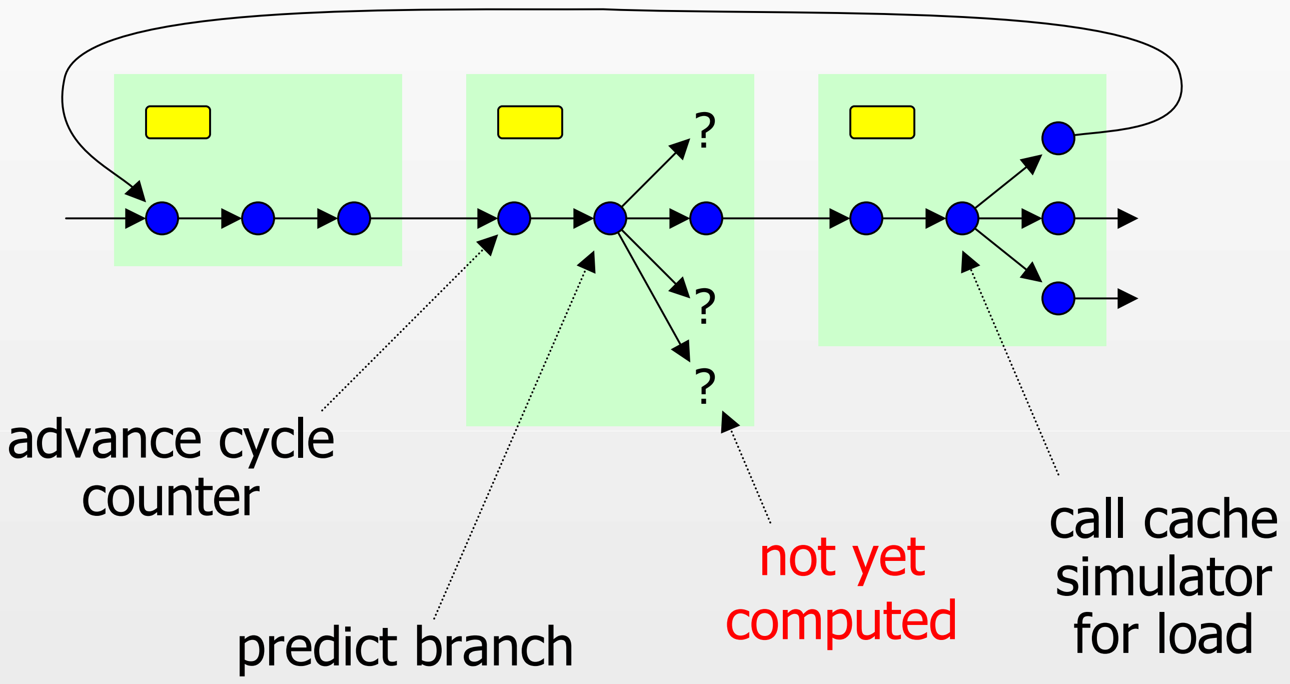
Configurations & Actions

Simulator Instruction Queue (iQ)

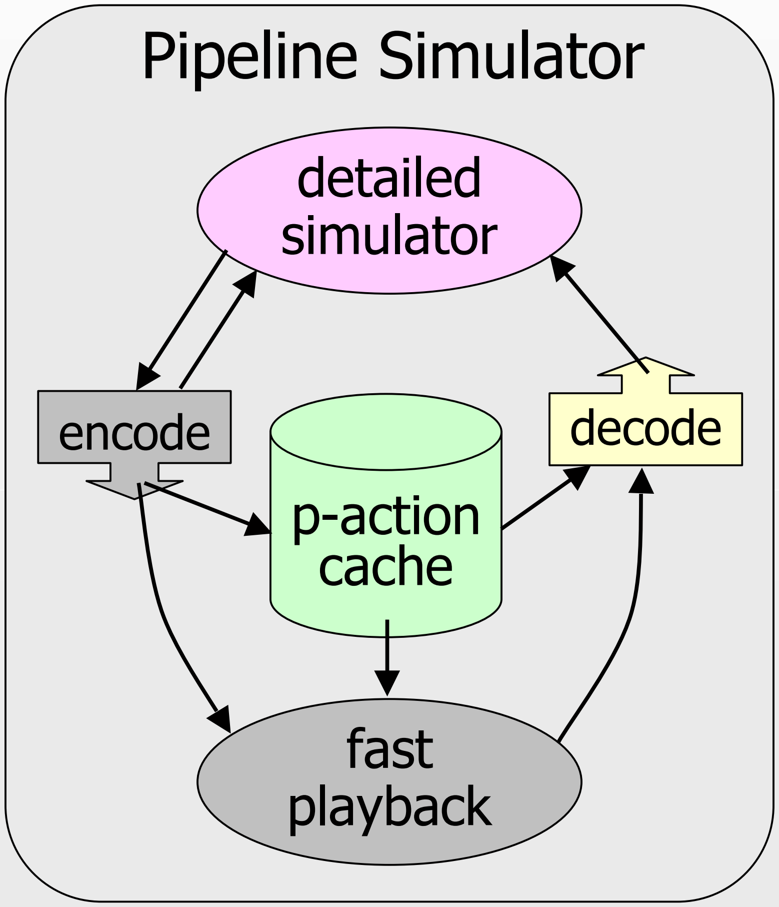
```
Addr.    Instruction                     Tag1  Tag2
0x10074  clr   %fp                        done
0x10078  ld    [ %sp + 0x40 ], %l0       cache  6
0x1007c  add   %sp, 0x44, %l1             exec  1
0x10080  sub   %sp, 0x20, %sp            queue
0x10084  tst   %g1                       queue
0x10088  be    0x10098                   queue
0x1008c  mov   %g1, %o0                  queue
0x10098  sethi %hi(0x5b000), %o0         fetch
0x1009c  or    %o0, 0x148, %o0           fetch
0x100a0  call  0x3f378                   fetch
0x100a4  nop                             fetch
```



Structure of the P-Action Cache

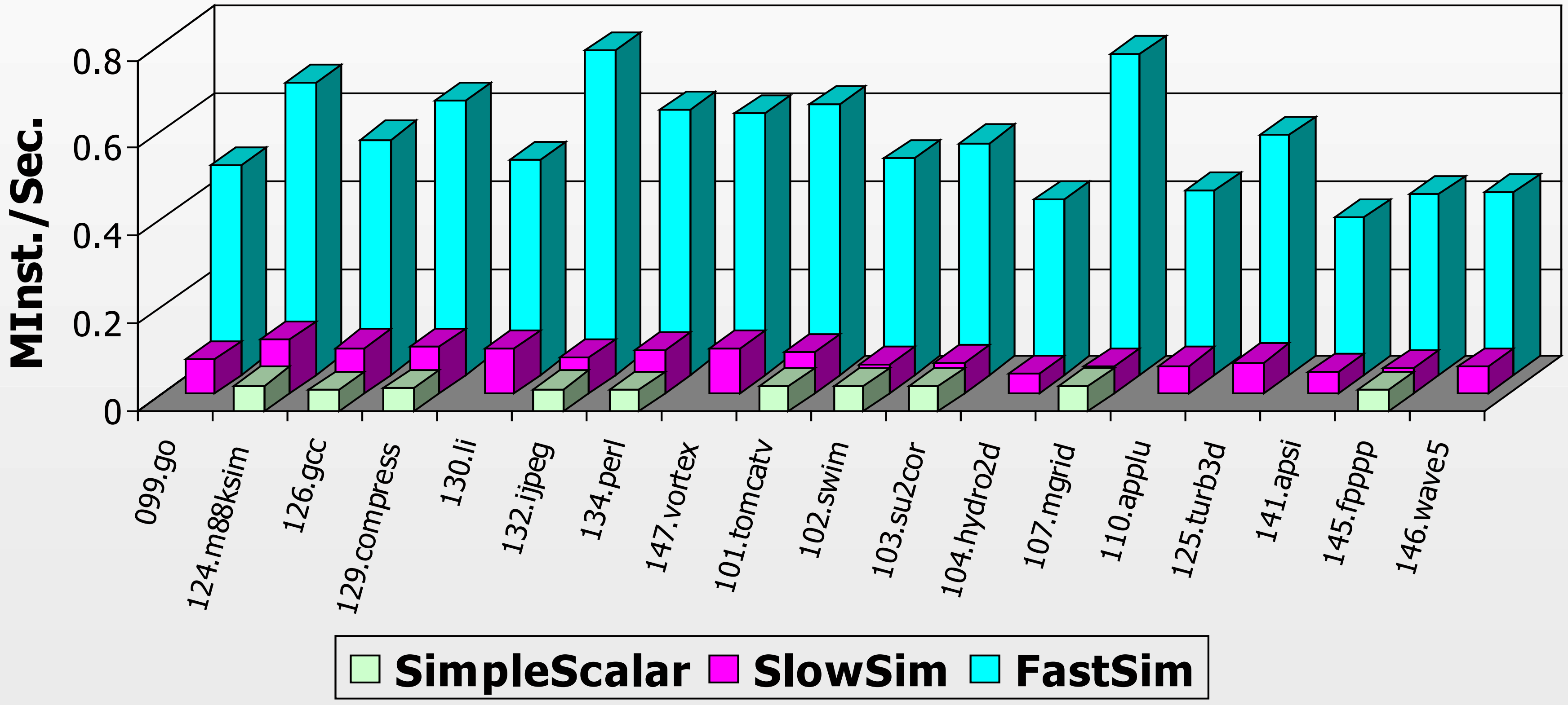


Return To Detailed Simulation

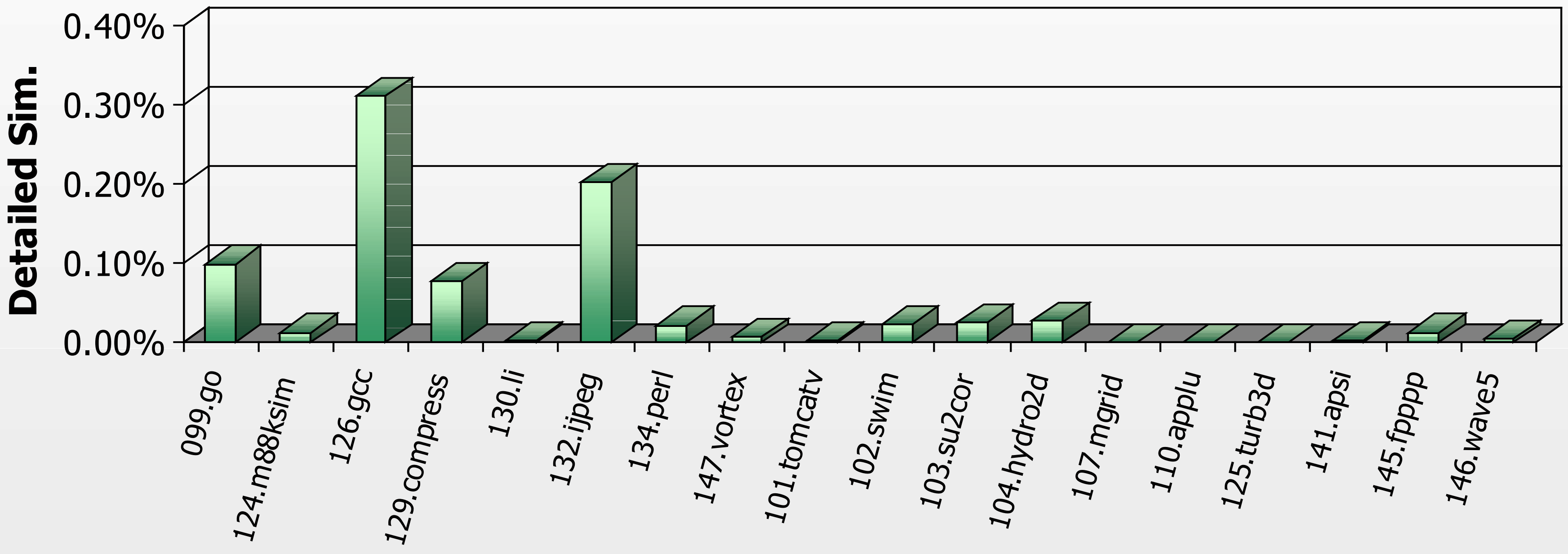


- Decode μ-architecture state from the p-action cache
- Restore simulator iQ
- Restart detailed simulation
- Grow the p-action cache

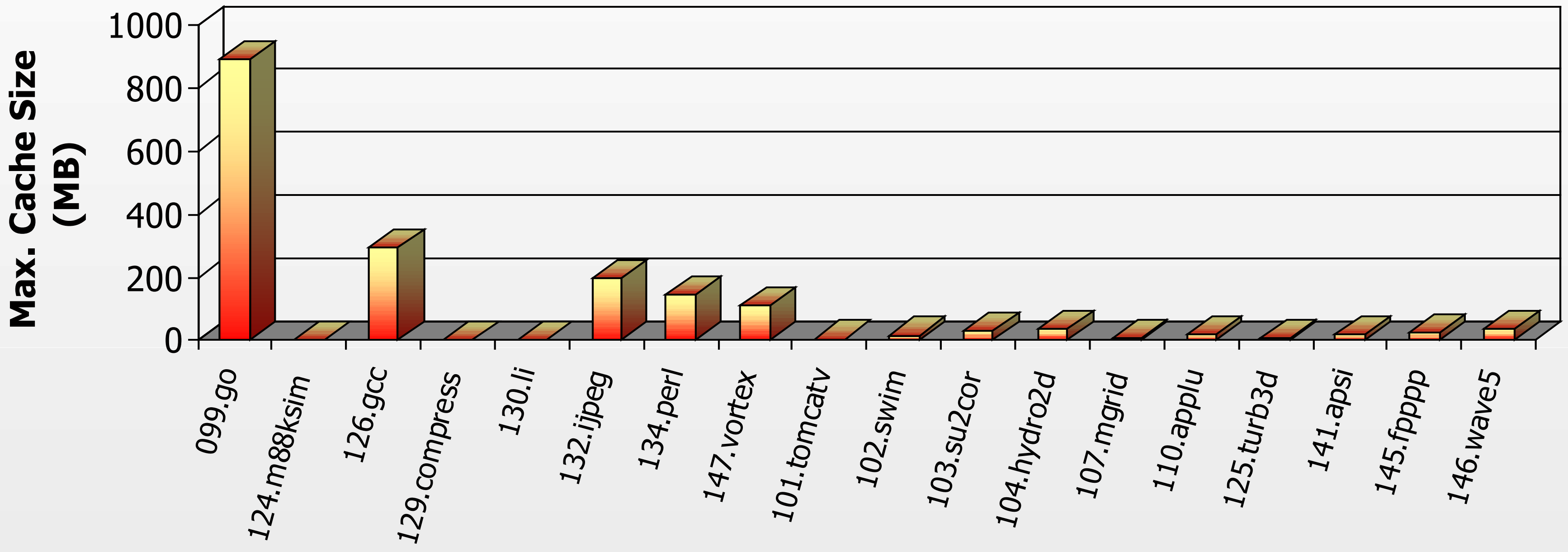
Performance vs. SimpleScalar



P-Action Cache Miss Rate



P-Action Cache Size



Limiting Size of P-Action Cache

- Replacement policy
  - Avoid fragmentation
  - Maintain pointers between actions
- Configurations can be deleted freely
  - If used again, they will be regenerated

Replacement Policies

- Cache-flush replacement policy
  - Easy to implement
  - Negligible overhead
  - 10x cache size reduction with little loss of performance
- Copying garbage collector
  - More difficult to implement
  - Added overhead from copying
  - Performance no better than cache-flush

Conclusion

- Can simulate out-of-order processors efficiently
  - 170-360 times slowdown
- Direct execution possible, but insufficient
- Memoization extremely effective
  - No loss of accuracy!
  - Cache size reduced by simple replacement policy