A Large, Fast Instruction Window for Tolerating Cache Misses

Alvin R. Lebeck

Tong Li

Jaidev Patwardhan

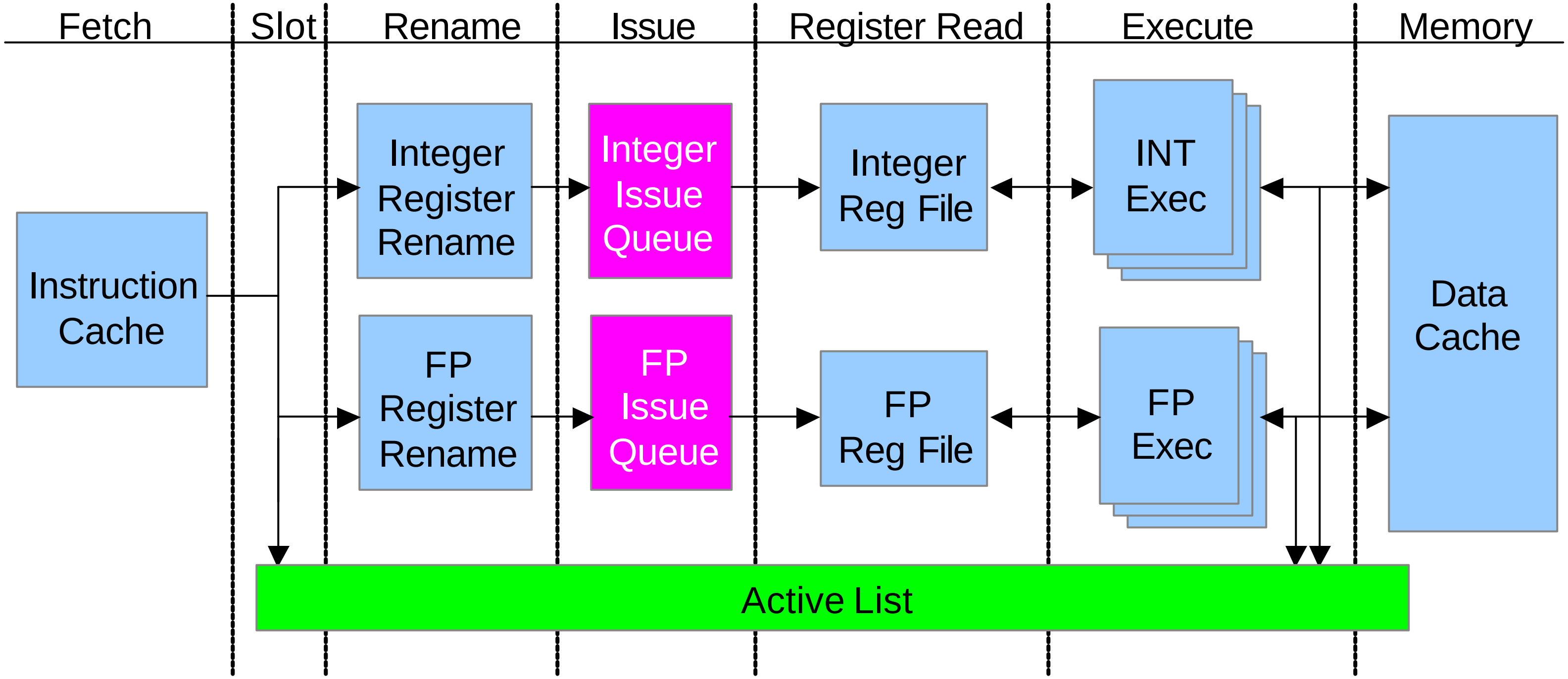
Department of Computer Science
Duke University
{alvy,tongli,jaidev}@cs.duke.edu

Eric Rotenberg

Jinson Koppanalil

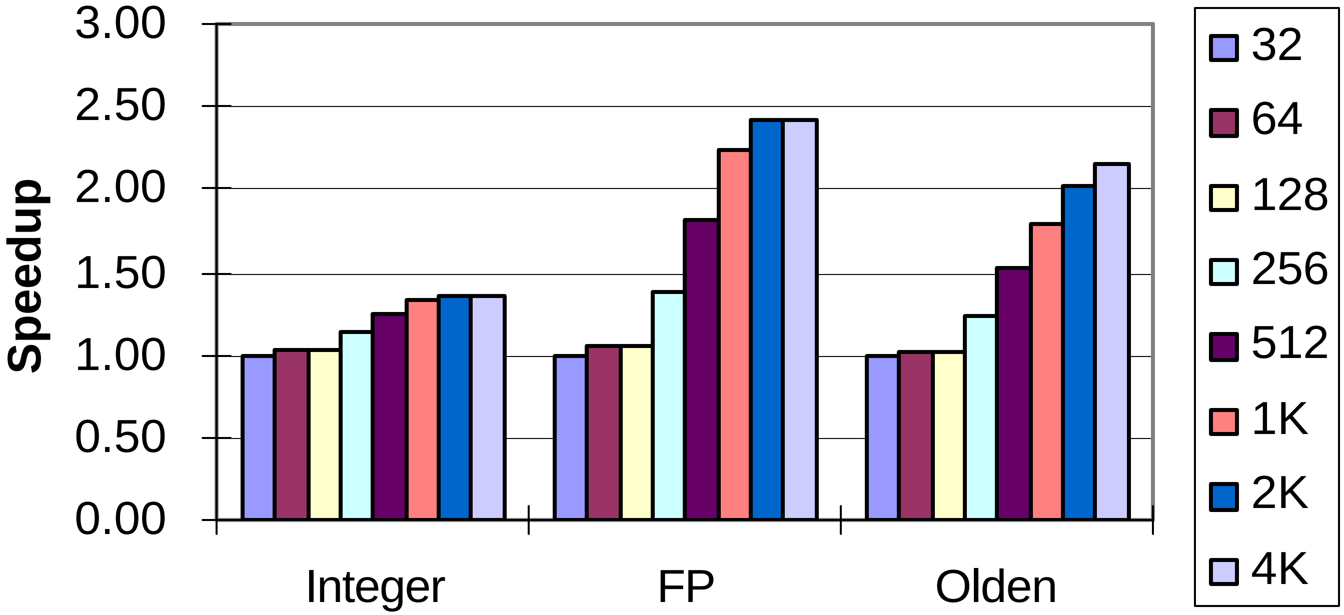
Department of Electrical and Computer Engineering
North Carolina State University
{jjkoppan,ericro}@ece.ncsu.edu

A Dynamically Scheduled Processor (Alpha 21264)



- **Active list:** keeps in-flight instructions in program order (the retire unit in the 21264)
- **Issue queues**: keep waiting instructions and issues ready instructions
- **Instruction window** = **Active list** + **Issue queues**
- ***We want thousands of instructions in-flight to exploit ILP***

Instruction Window Size Effects



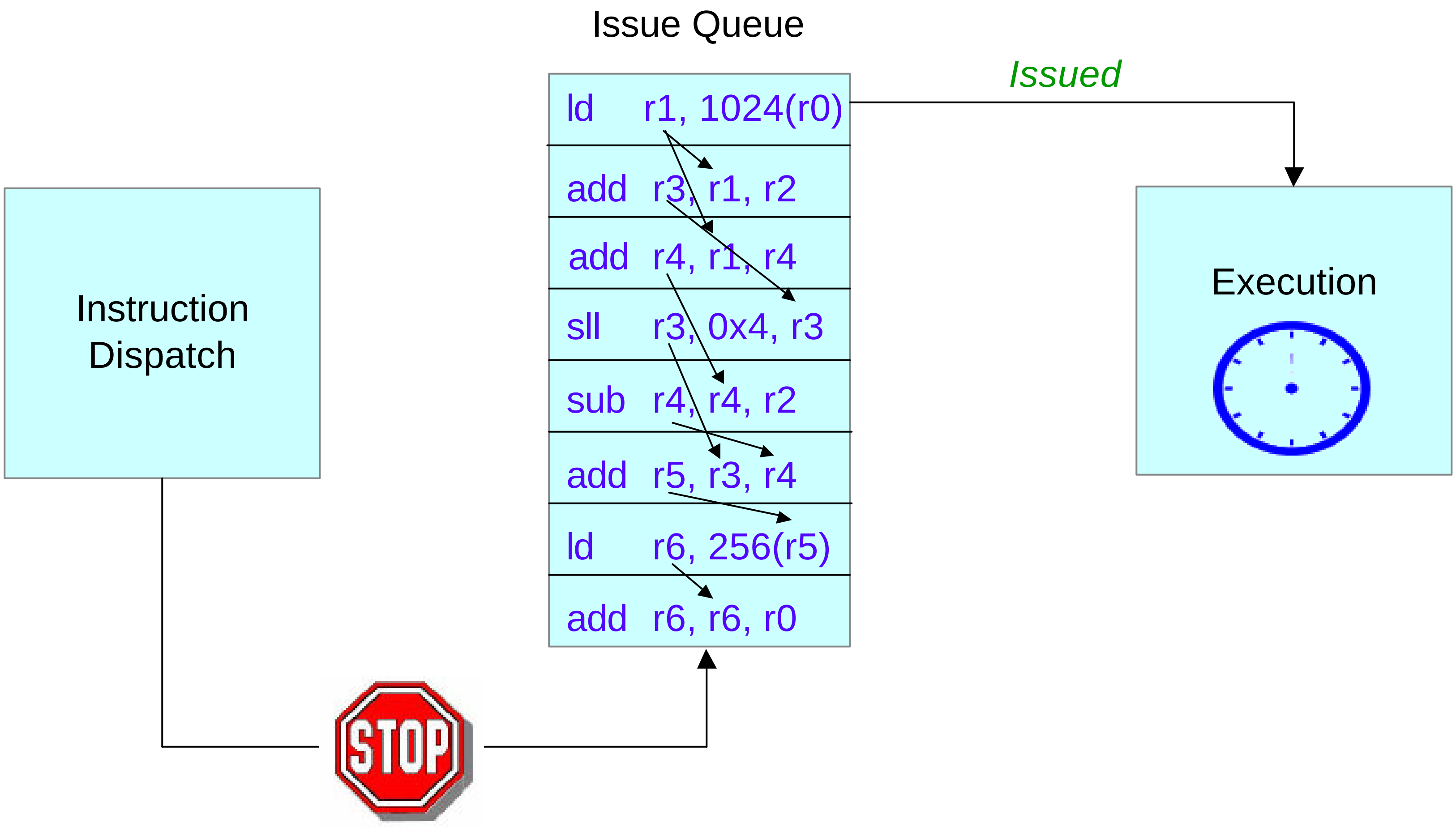
- 32, 64, and 128 have 128-entry active list, all others have active list size same as instruction window (issue queue) size
- Larger window, higher IPC, most plateau beyond 2K
- 2K vs. 32 (base): Integer 35%, FP 140%, Olden 103%

Towards a Large Instruction Window

- Larger instruction window, more ILP exposed
- Larger instruction window, slower clock (Palacharla et al. ISCA’97, Agarwal et al. ISCA’00)
  - Active list can be large because it’s not on critical path
  - Issue queue doesn’t scale due to complex wakeup/select

Goal: Support large instruction window without affecting clock cycle time

Problem of Conventional Design



```
ld   r1, 1024(r0)
add  r3, r1, r2
add  r4, r1, r4
sll  r3, 0x4, r3
sub  r4, r4, r2
add  r5, r3, r4
ld   r6, 256(r5)
add  r6, r6, r0
```

The Big Idea

- Problem:
  - Instructions dependent on a long latency operation (e.g., cache miss) *waste* issue queue entries!
  - No new instructions are able to come into the window.

- Our Solution:
  - Move instructions dependent on a long latency operation to a *waiting instruction buffer* (WIB)
  - Reinsert *all* dependent instructions from WIB back to issue queue when the long latency operation finishes

Outline

- Motivation
- The Waiting Instruction Buffer (WIB)
  - Design and implementation
- Results
  - Average speedups:
    - SPEC2000 Integer 20%
    - SPEC2000 FP 84%
    - Olden 50%
- Conclusion

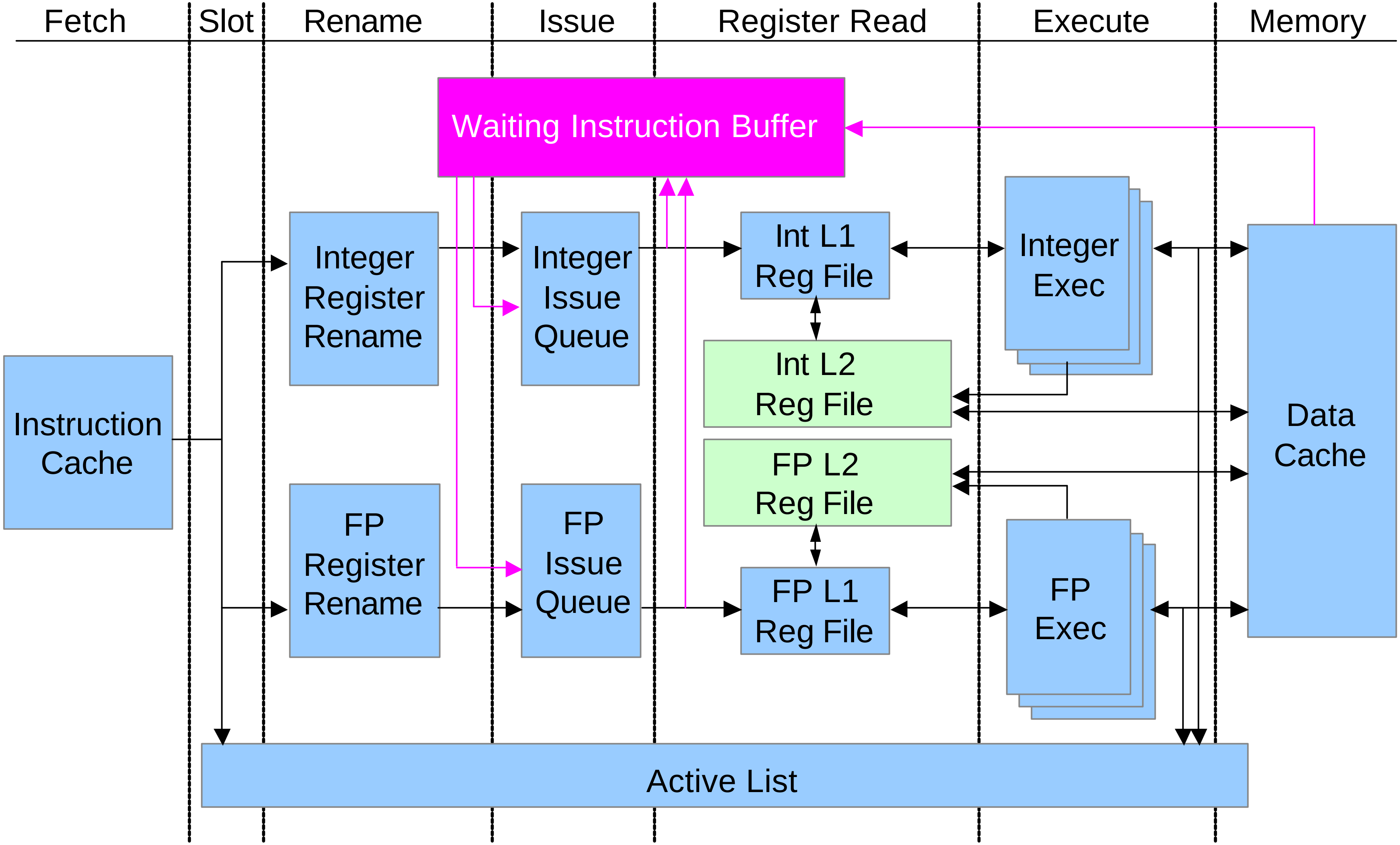
The Waiting Instruction Buffer

- Move instructions dependent on a long latency operation to a *waiting instruction buffer* (WIB)
- Reinsert *all* dependent instructions from WIB back to issue queue when the long latency operation finishes
- WIB has *simple* wakeup/select

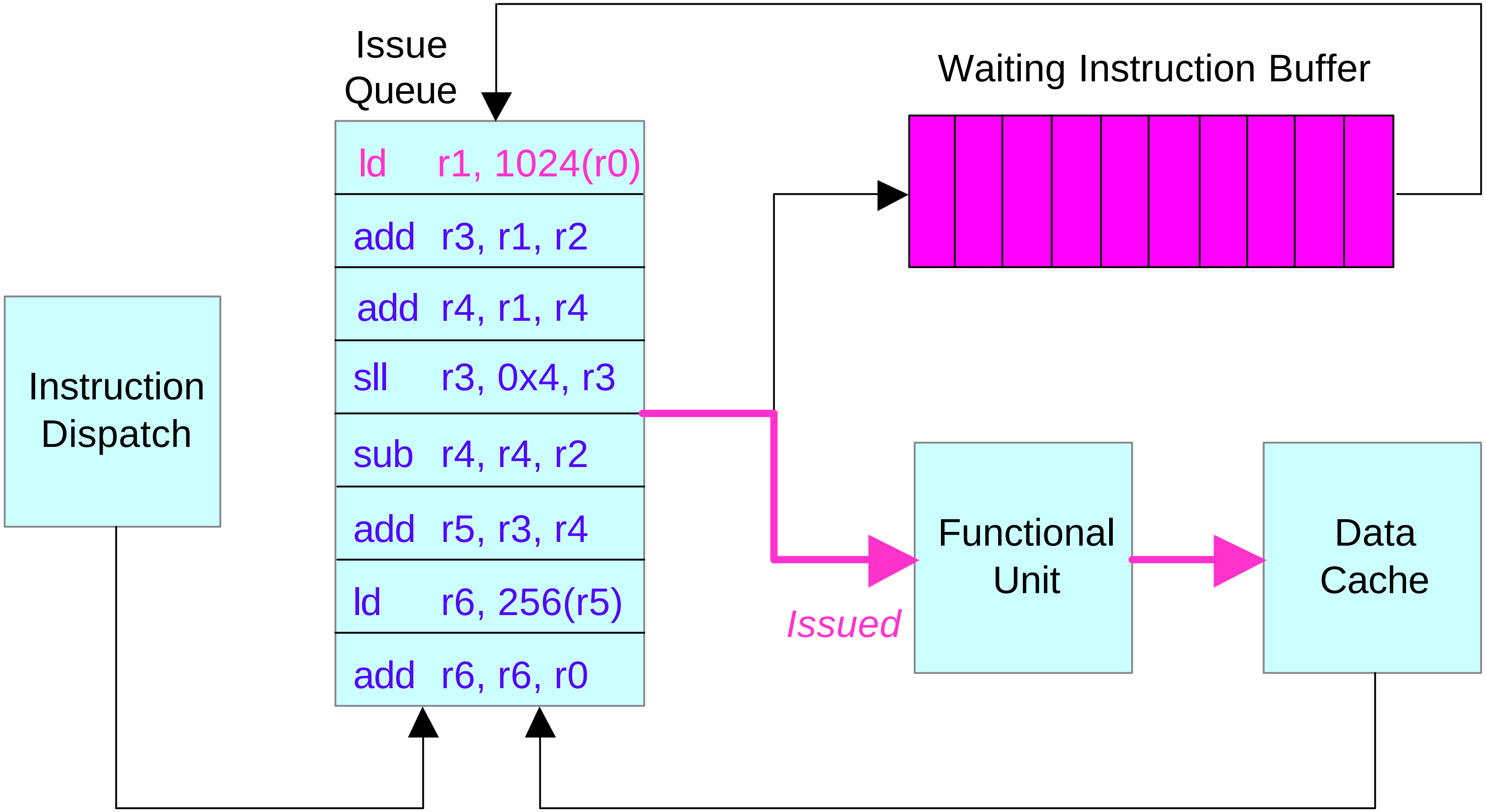
Key:

- No full dependence checking in the WIB
- WIB only tracks which long latency operation instructions depend on

WIB-based Architecture

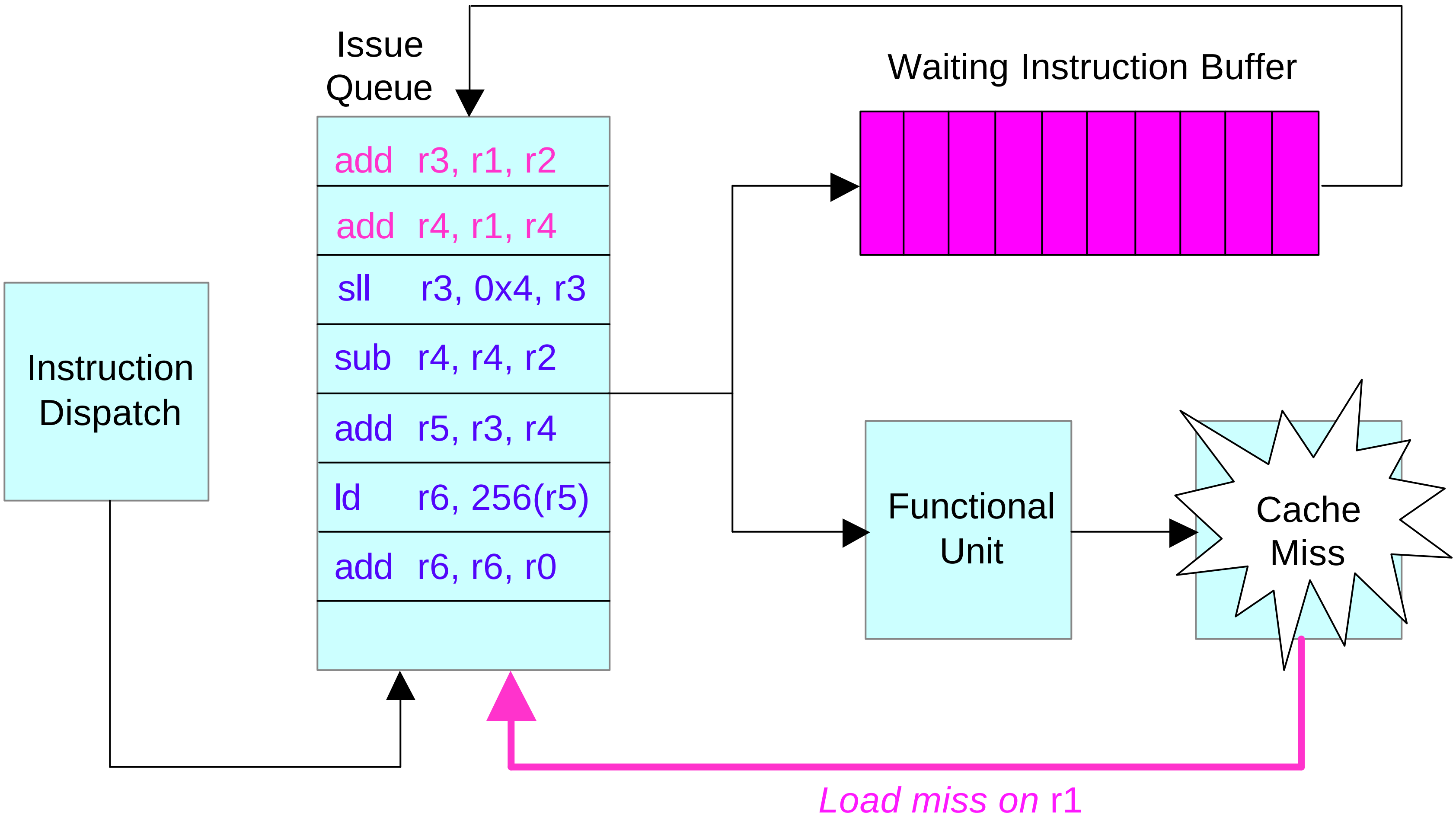


How the WIB Works

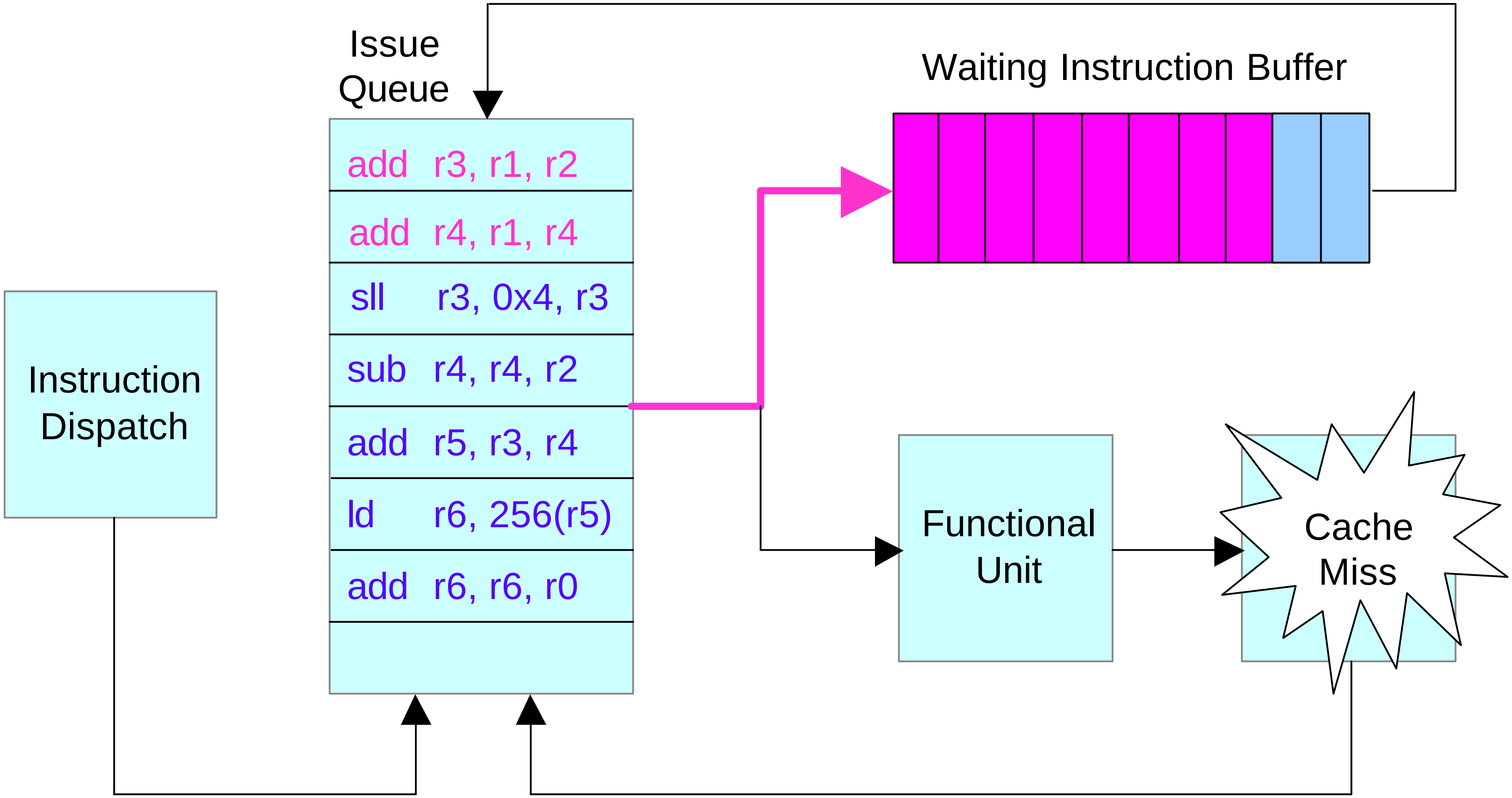


```
ld    r1, 1024(r0)
add   r3, r1, r2
add   r4, r1, r4
sll   r3, 0x4, r3
sub   r4, r4, r2
add   r5, r3, r4
ld    r6, 256(r5)
add   r6, r6, r0
```

How the WIB Works



How the WIB Works



Issue Queue

```
add  r3, r1, r2
add  r4, r1, r4
sll  r3, 0x4, r3
sub  r4, r4, r2
add  r5, r3, r4
ld   r6, 256(r5)
add  r6, r6, r0
```

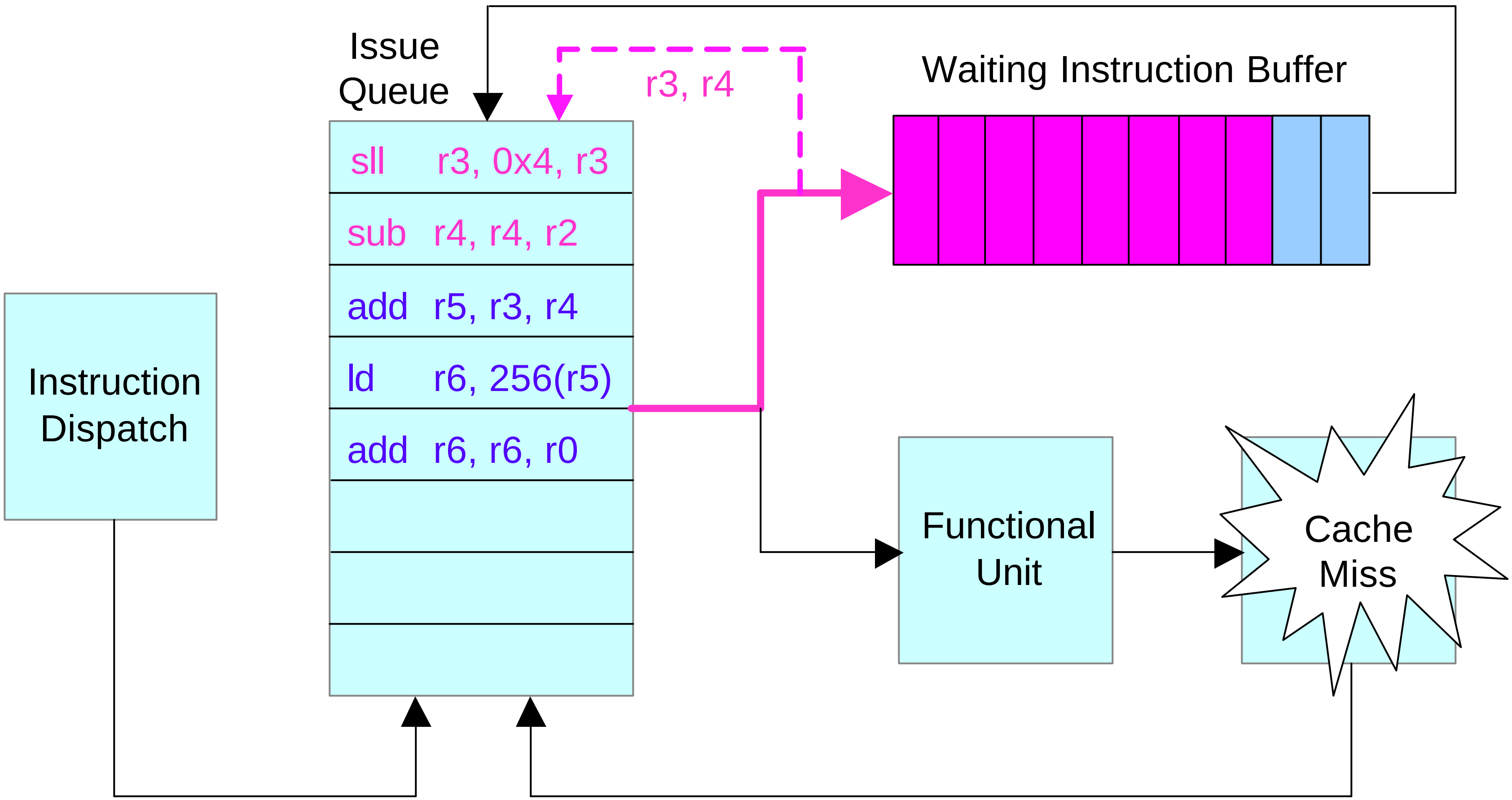
Waiting Instruction Buffer

Instruction Dispatch

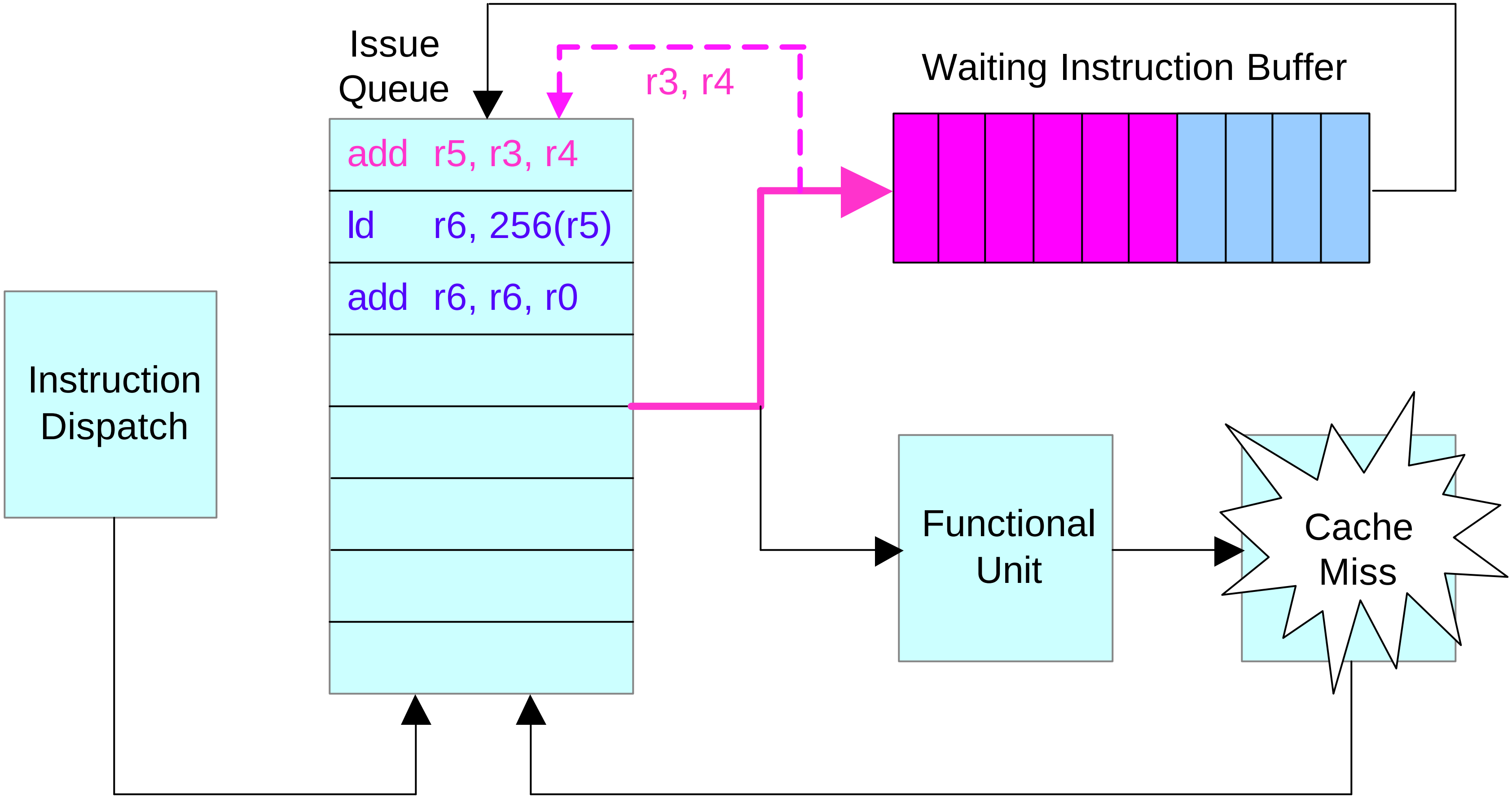
Functional Unit

Cache Miss

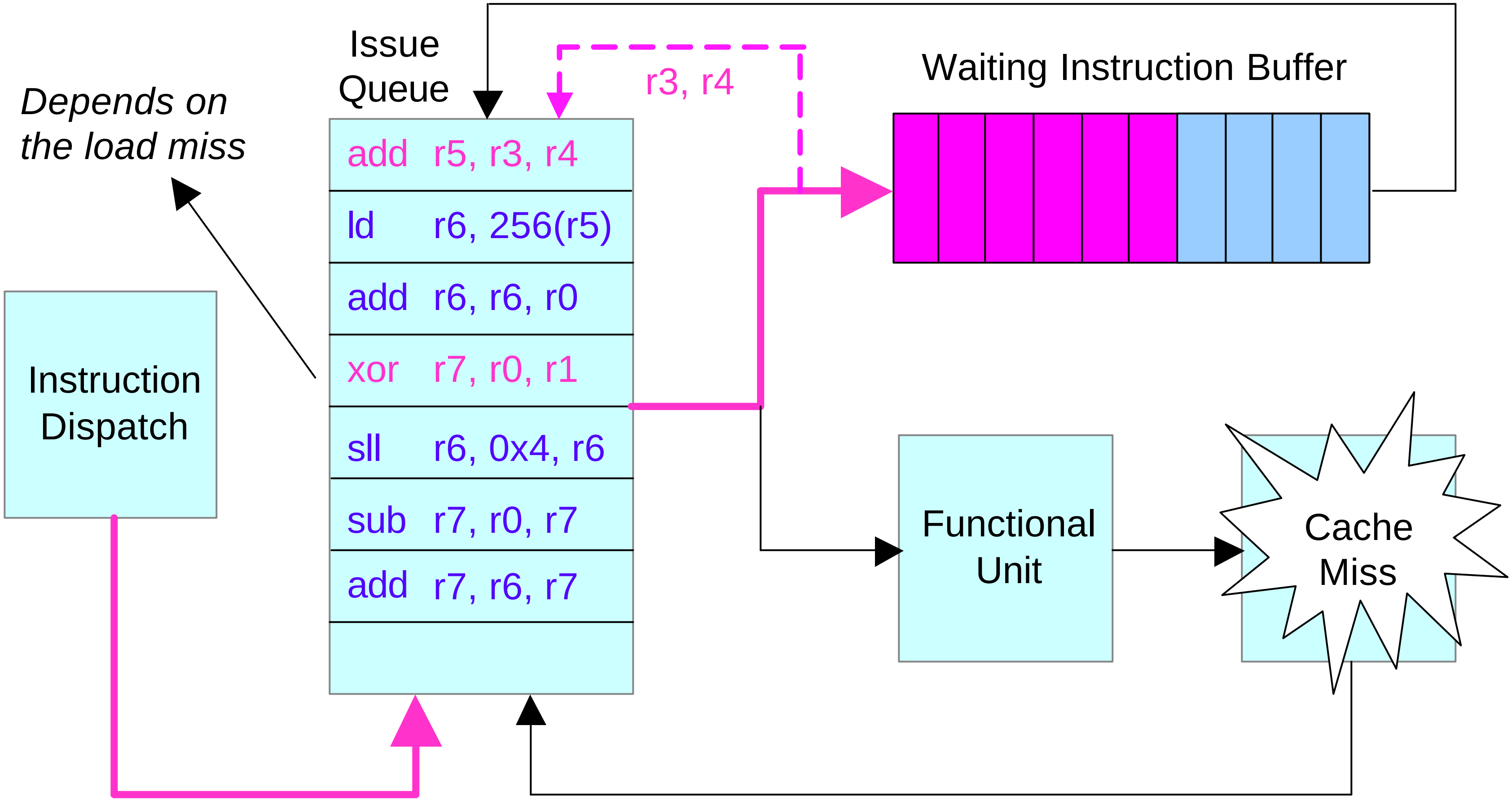
How the WIB Works



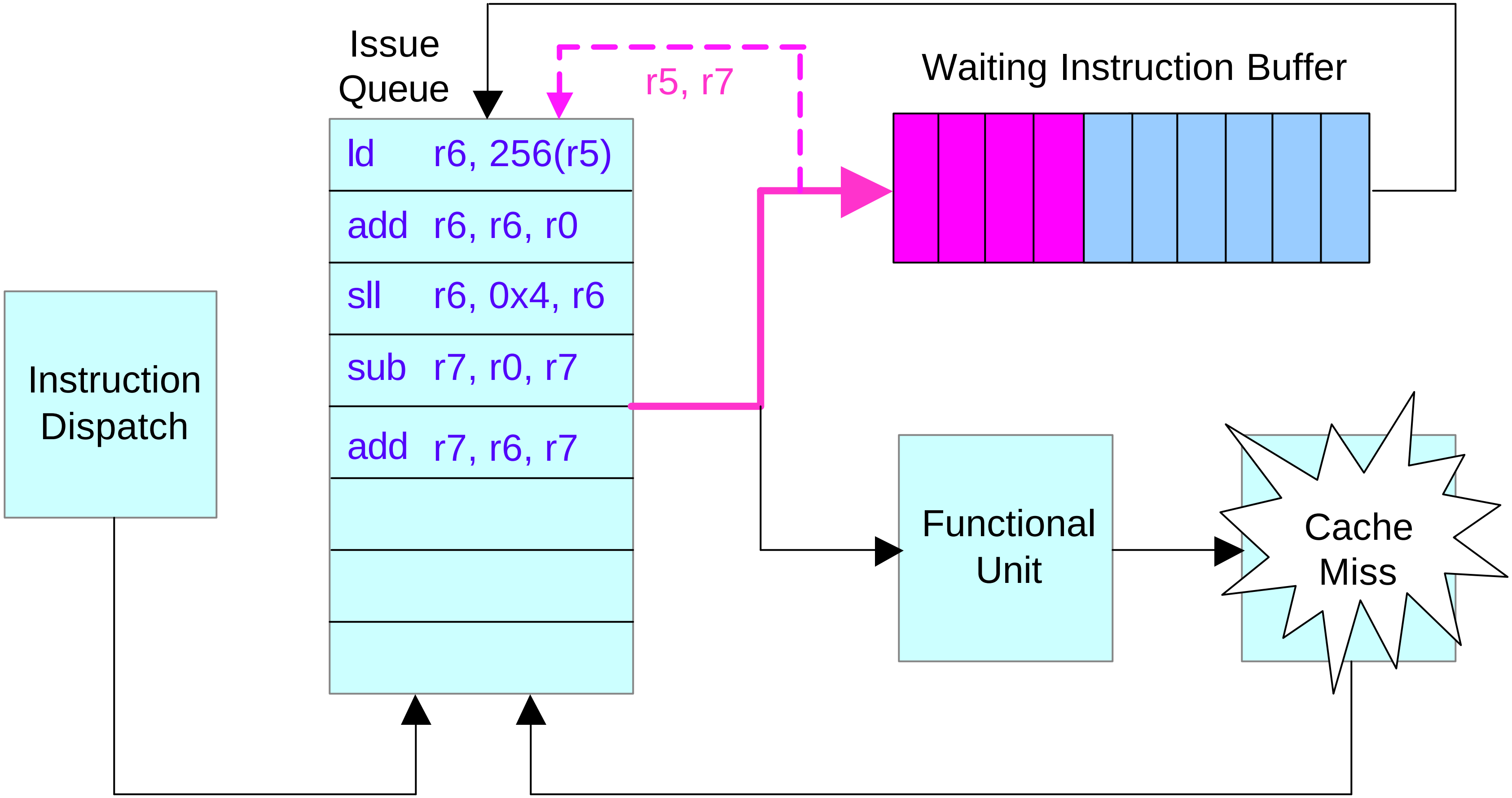
How the WIB Works



How the WIB Works

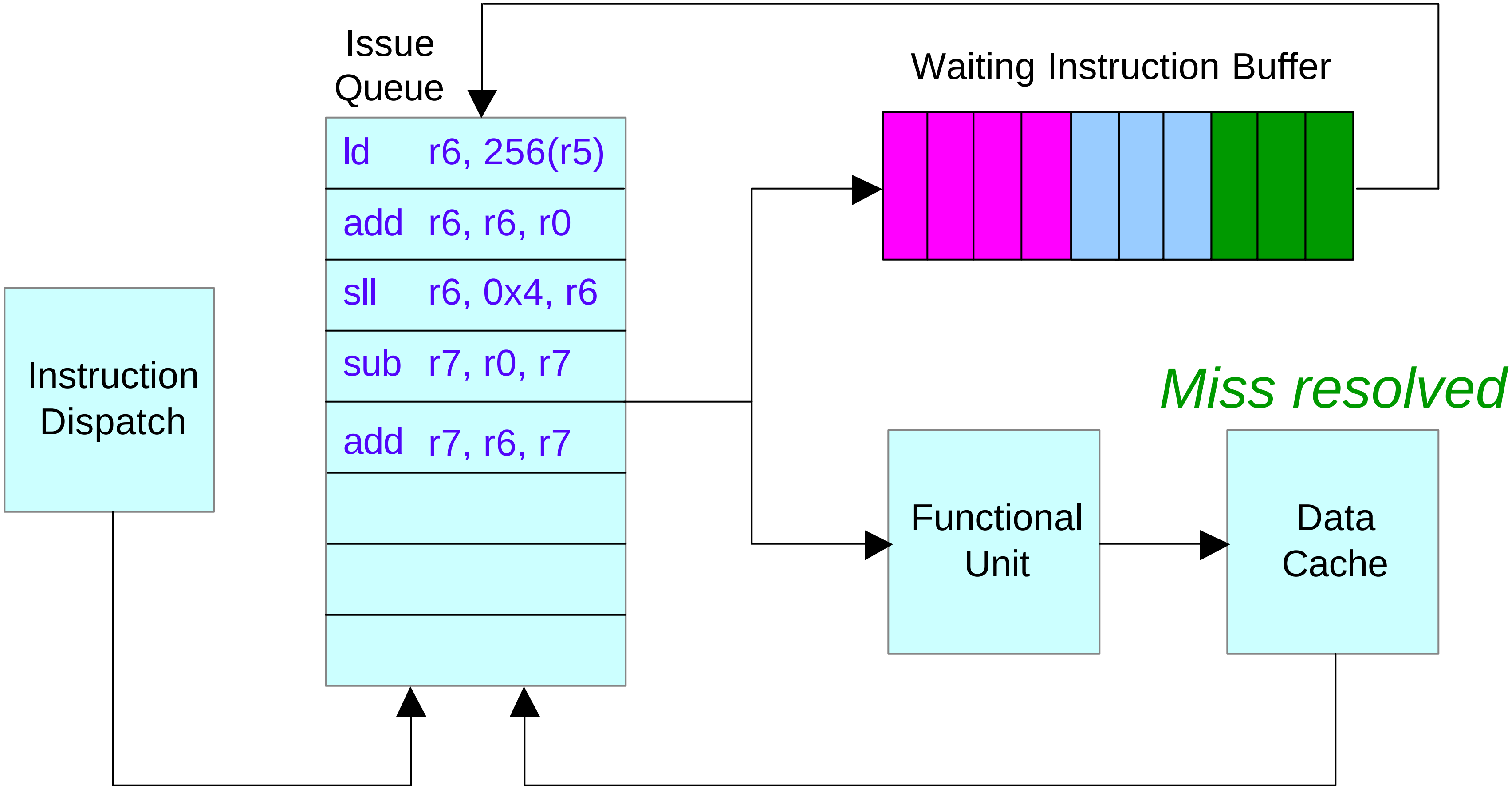


How the WIB Works



```
ld   r6, 256(r5)
add  r6, r6, r0
sll  r6, 0x4, r6
sub  r7, r0, r7
add  r7, r6, r7
```

How the WIB Works

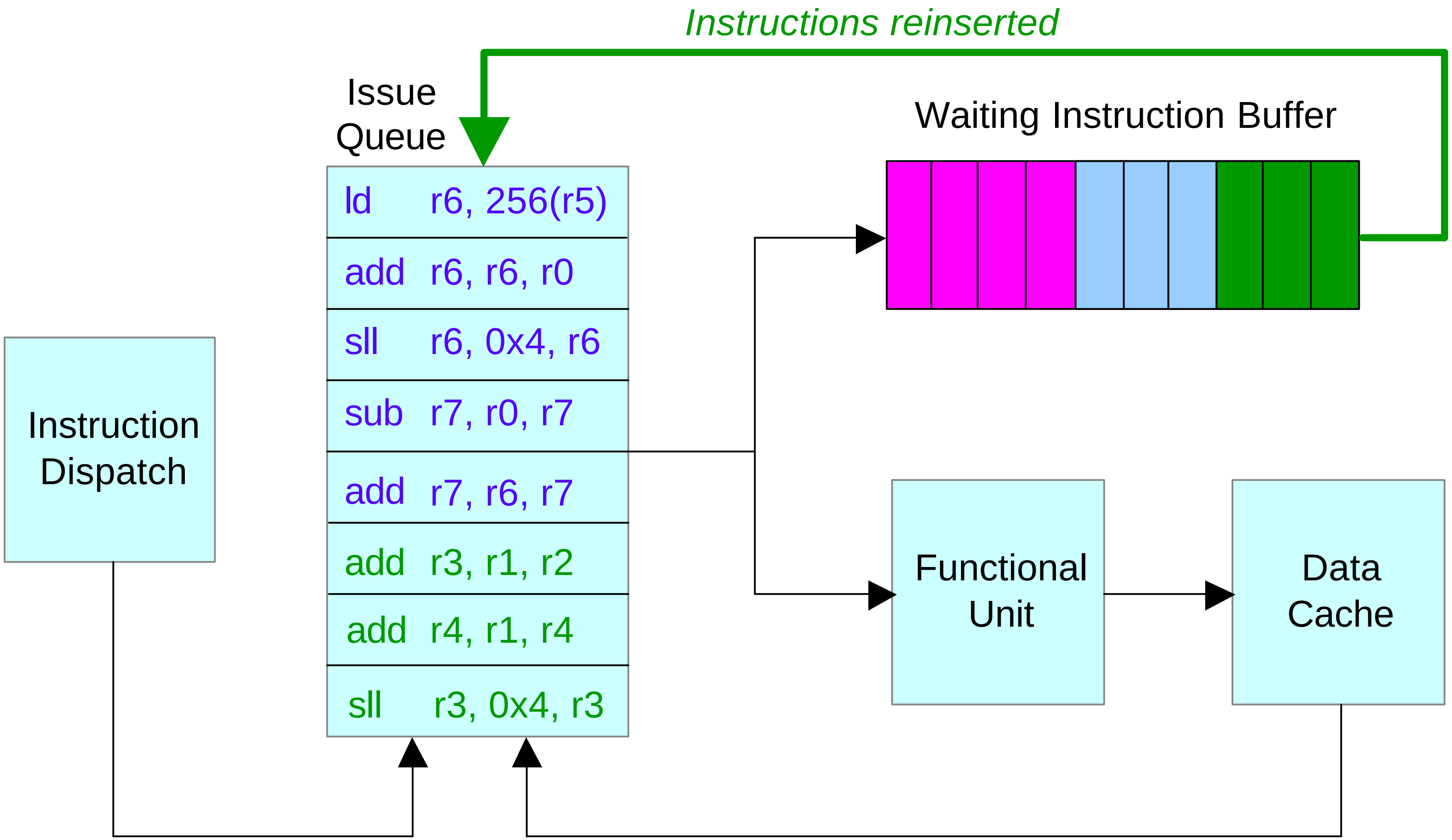


Issue Queue

```
ld	r6, 256(r5)
add	r6, r6, r0
sll	r6, 0x4, r6
sub	r7, r0, r7
add	r7, r6, r7
```

Miss resolved

How the WIB Works



WIB Design Issues

- WIB organization
- Allocating instructions to the WIB
- Reinserting instructions from WIB to issue queue
- Branch mispredicts & exceptions

Design choice:

- To handle recovery, WIB keeps instructions in program order.
- WIB is organized around the active list (same size as the active list).

Moving Instructions to WIB

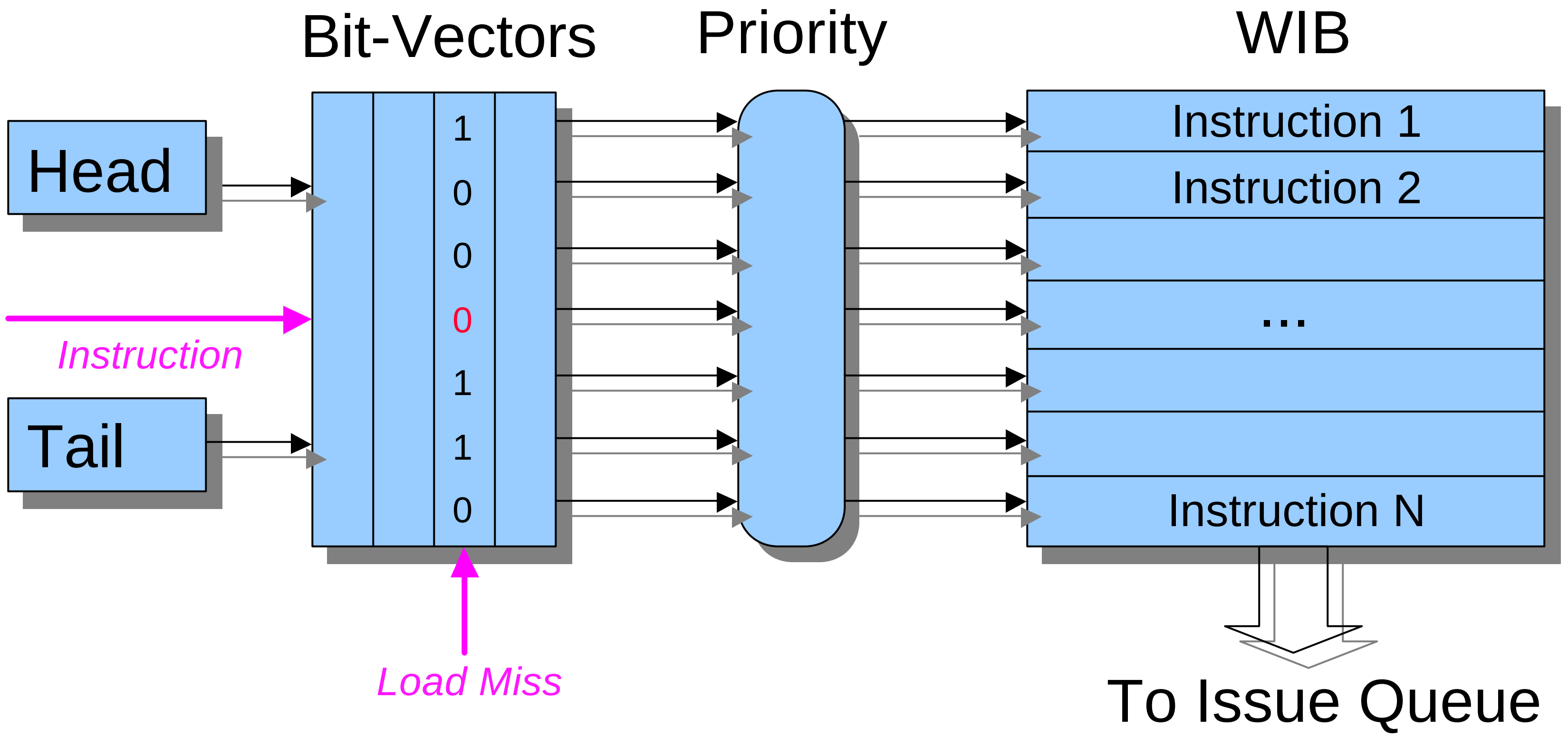
- Use a "*pretend ready*" bit alongside the conventional ready bit
- Cache miss raises a "*pretend ready*" signal
- Utilize existing issue queue wakeup/select logic
- Propagate pretend ready bit to dependent instructions that are
  - Already in flight
  - Not fetched yet

p0	busy
p1	ready
p2	pretend ready
⋮	⋮

Register Status Table

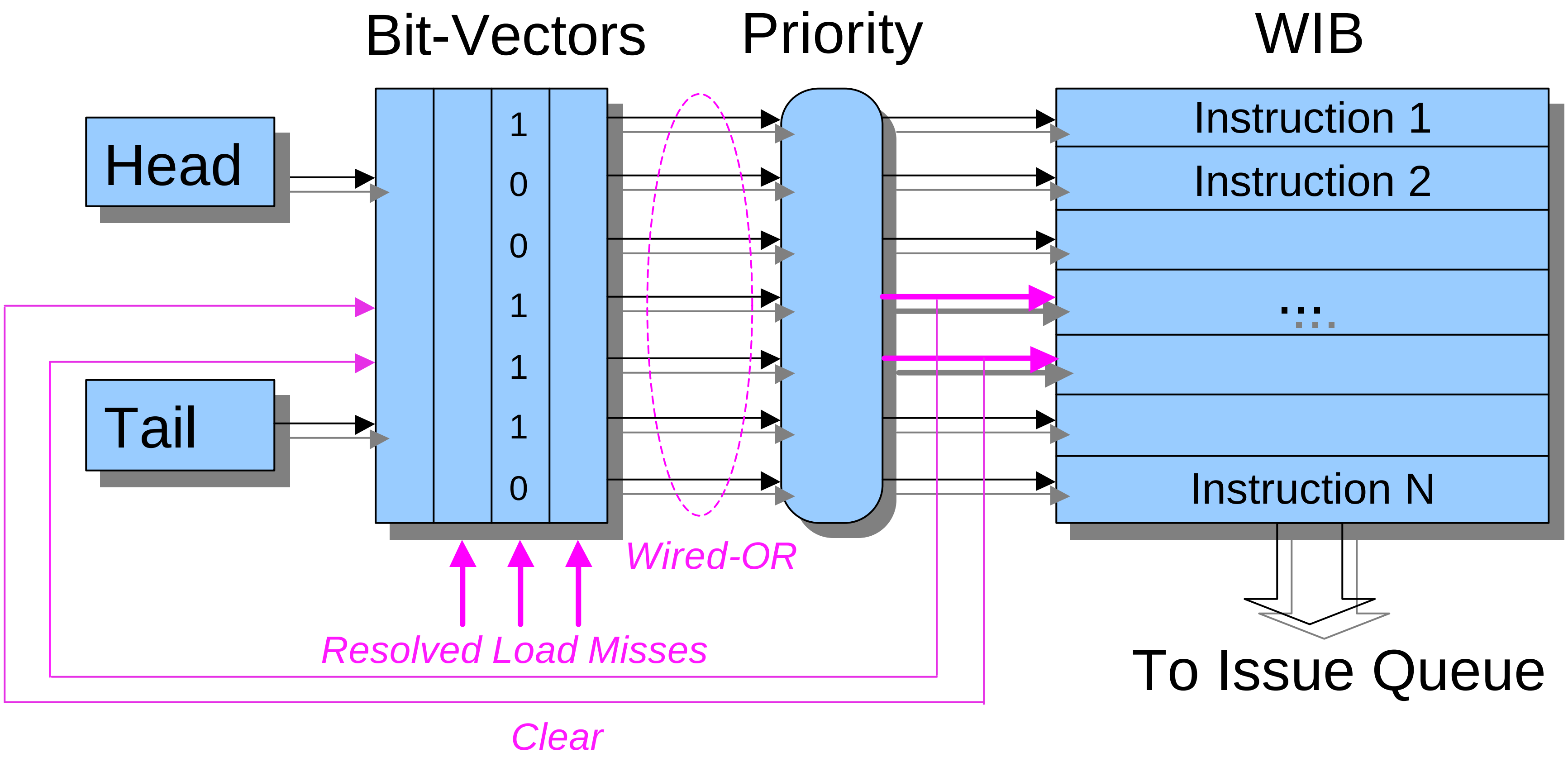
Moving Instructions to WIB

- ***Bit-vectors*** help ease WIB management
- Set instruction's corresponding bit to 1 to indicate it's moved to the WIB

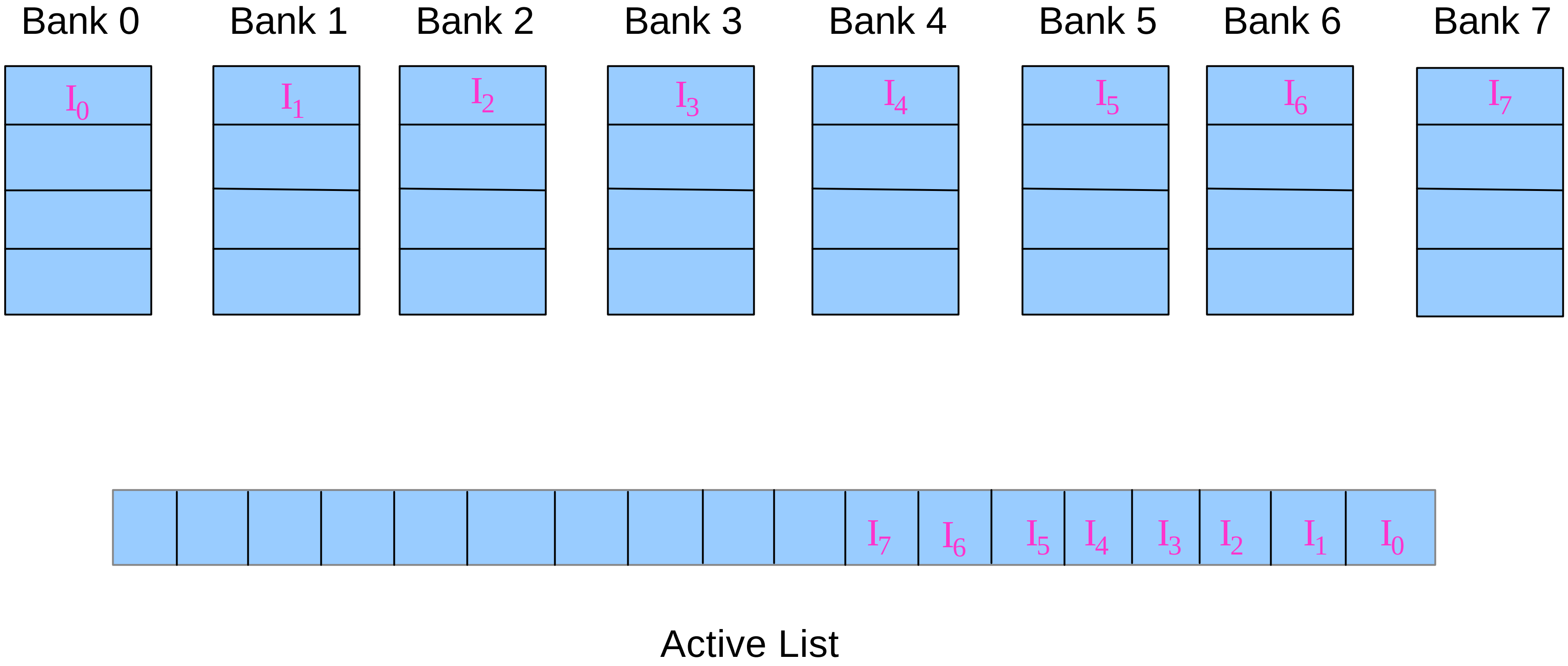


Reinserting Instructions to IQ

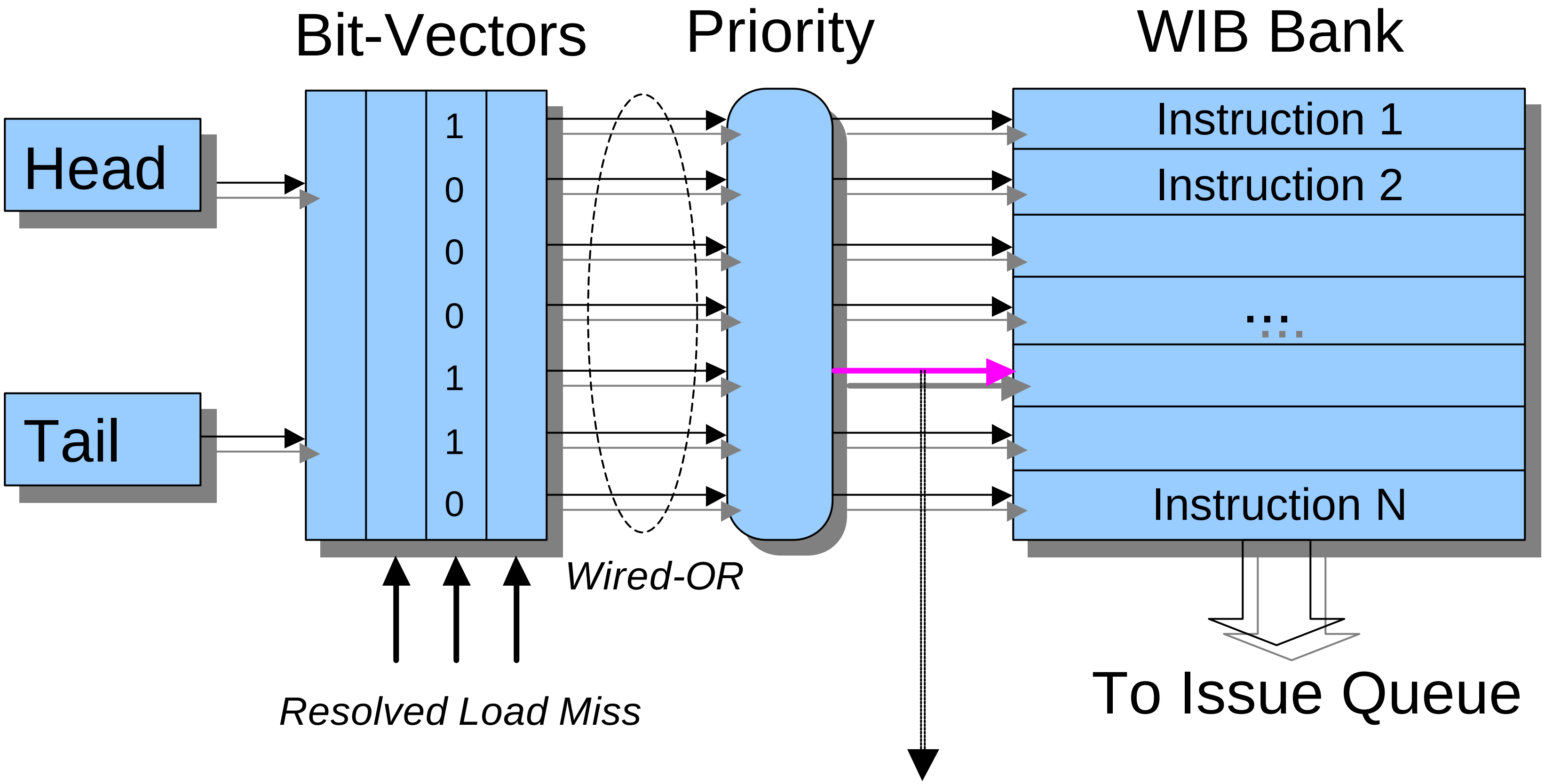
- *All* dependent instructions are eligible to be reinserted
- Priority circuit selects instructions based on selection policy



A Multi-banked WIB Organization



A Single Bank View



Select a single instruction out of the WIB

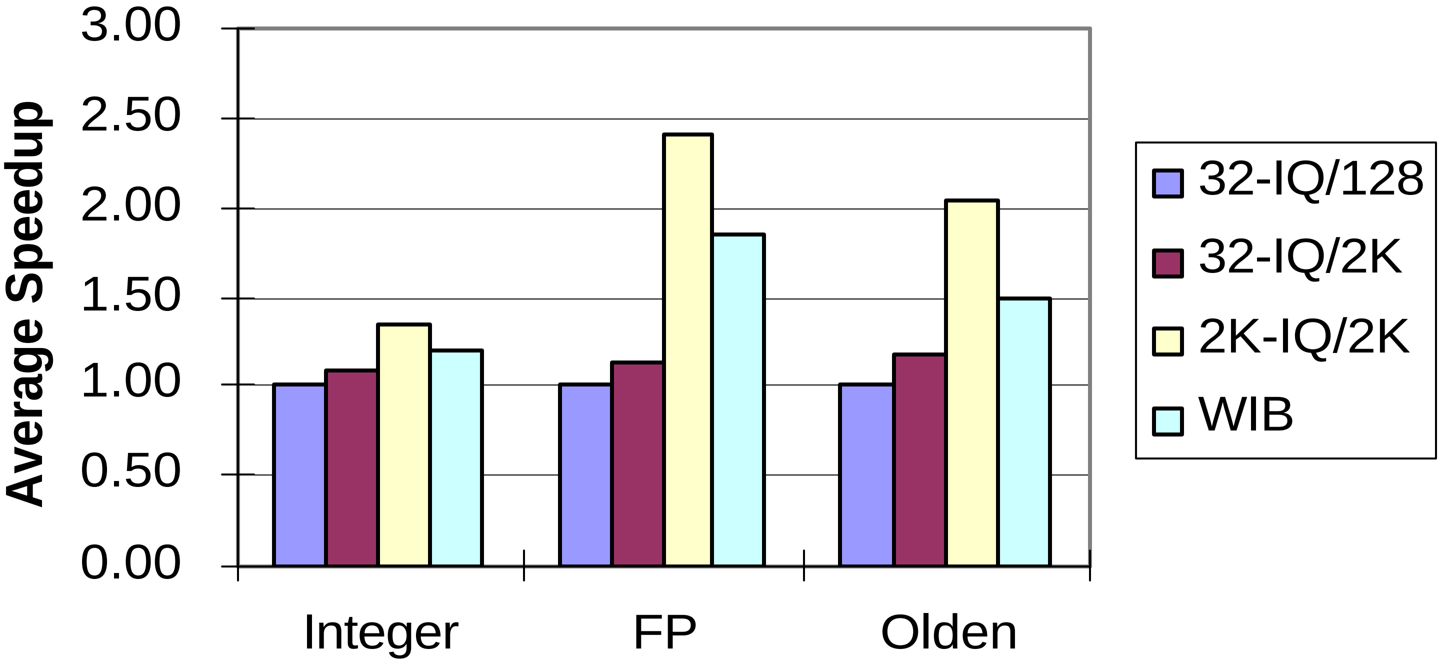
Outline

- Motivation
- The Waiting Instruction Buffer (WIB)
- WIB design issues
- Results
  - Average speedups:
    - SPEC2000 Integer 20%
    - SPEC2000 FP 84%
    - Olden 50%
- Conclusion

Methodology

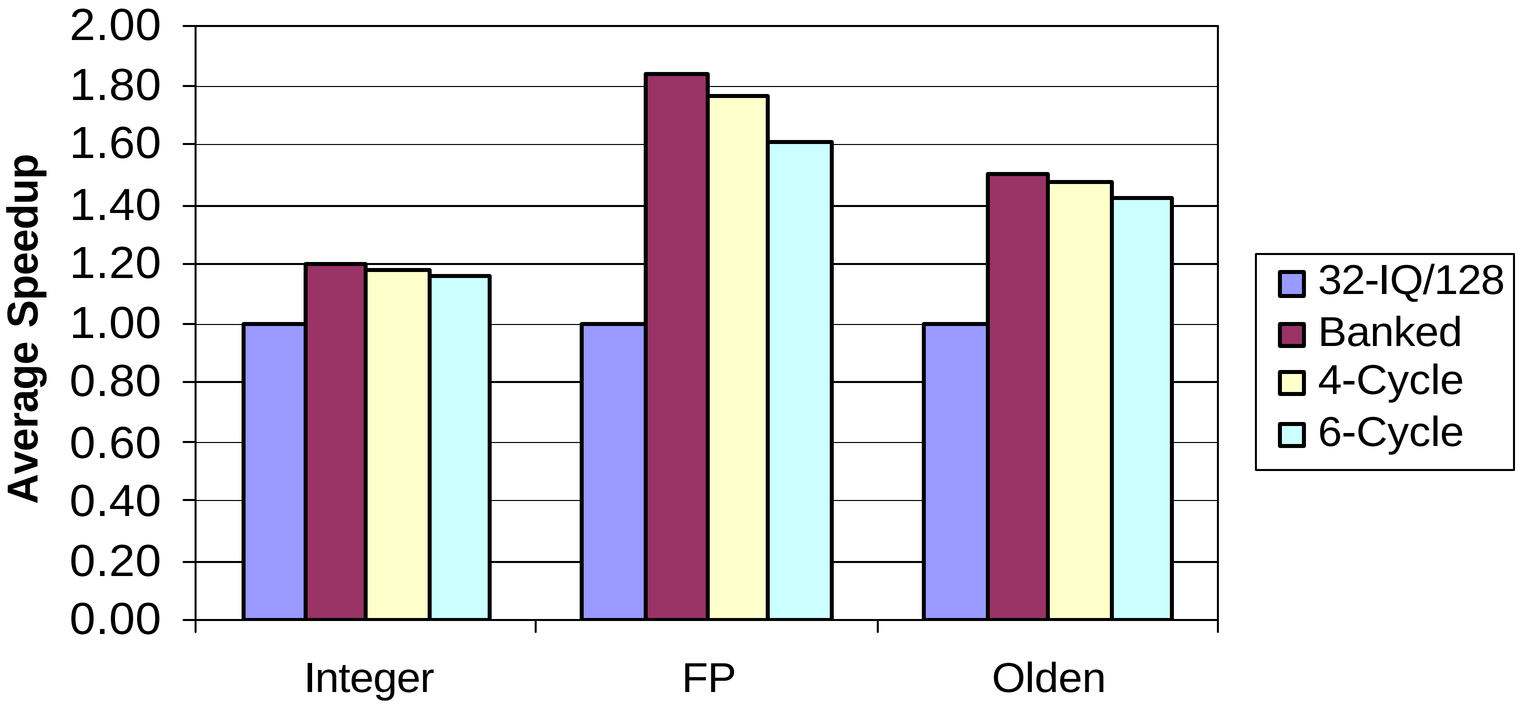
- Heavily modified SimpleScalar v3.0b
- 8-way fetch, decode, and commit
- 12-way issue (8 integer, 4 FP)
- 32 KB 4-way 2-cycle L1, 256 KB 4-way 10-cycle L2 data caches, 250-cycle memory latency
  - Shorter memory latency and larger L2 data cache do not affect WIB performance qualitatively
- Benchmarks:
  - SPEC2000 Integer and FP: skip first 400M, execute next 100M instructions
  - Olden: execute first 400M or until completion

WIB Performance



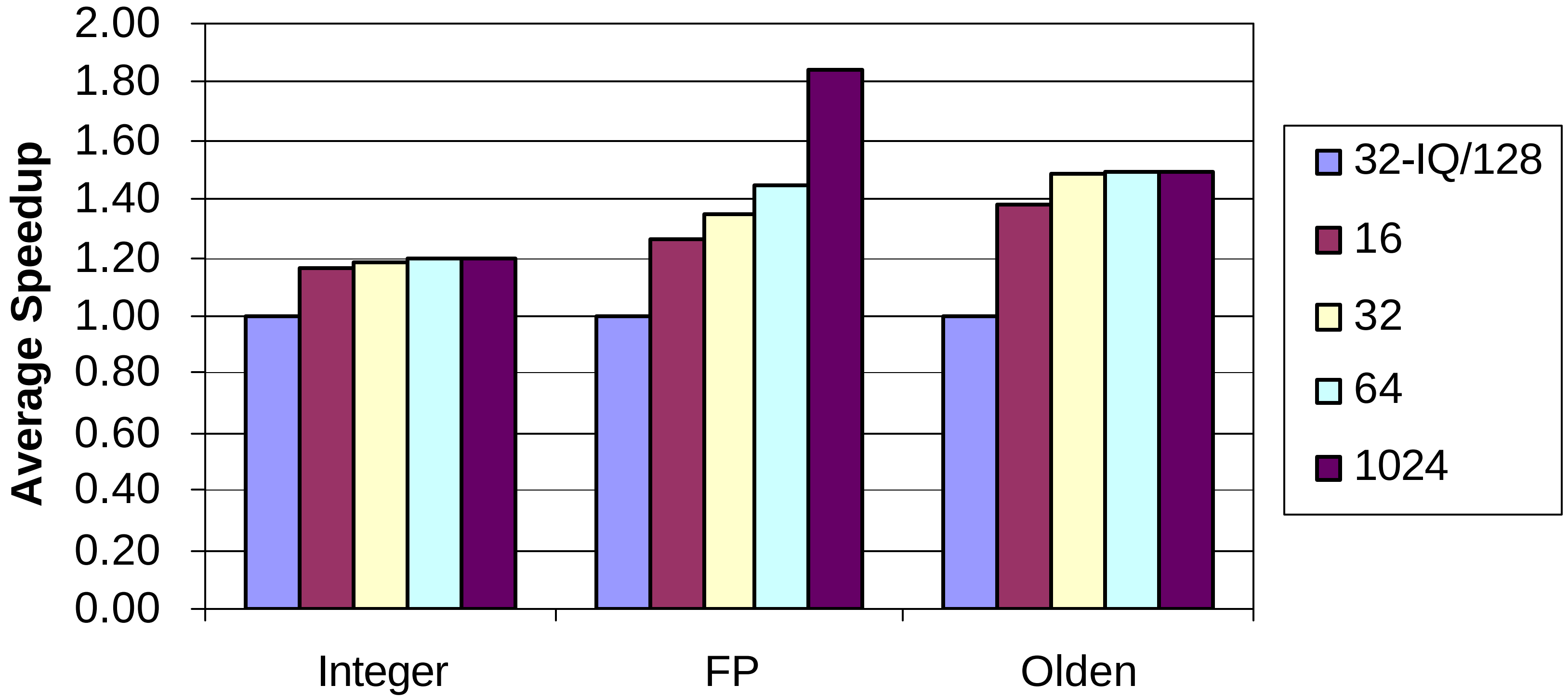
- WIB: 2K-entry active list, 32-entry IQ, 16-banked, 1K bit-vectors
- Average: SPEC INT 20%, SPEC FP 84%, Olden 50%
- Maximum: SPEC INT 76%, SPEC FP 290%, Olden 161%

Non-Banked Multicycle WIB



- 4-cycle and 6-cycle: non-banked WIB with instruction extraction in full program order
- WIB latency only affects performance slightly

Limited Bit-Vectors



- With 64 bit-vectors, SPEC INT 19%, SPEC FP 45%, Olden 50%
- With 16 bit-vectors, SPEC INT 16%, SPEC FP 26%, Olden 38%
- FP benchmarks are affected most because they have more memory level parallelism

Conclusion

- Motivation:
  - Larger instruction window exposes higher ILP
  - Conventional designs do not scale
- Observation:
  - In conventional designs, instructions dependent on long latency operations waste issue queue slots
- Waiting instruction buffer:
  - Enlarges effective window size without affecting clock cycle time
  - Implements a simplified form of wakeup-select
  - Insensitive to access latency