NOREBA: A Compiler-Informed Non-speculative Out-of-Order Commit Processor

Ali Hajiabadi[✢], Andreas Diavastos[✦], Trevor E. Carlson[✢]

✢ *National University of Singapore*
✦ *Universitat Politècnica de Catalunya*

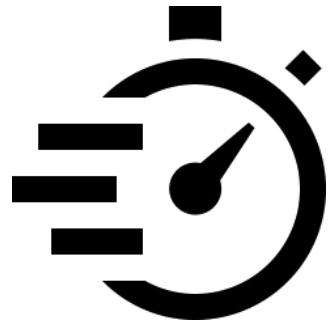
ASPLOS '21

NOREBA: Goal

- Current processors hold on to resources longer than necessary
- **NOREBA implements an intelligent resource management technique based on true branch dependencies**

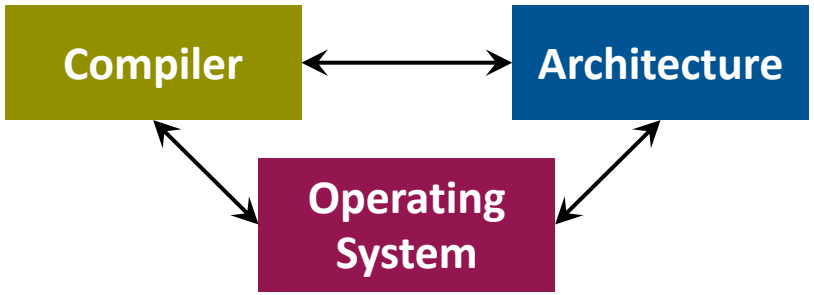
→Performance Improvement

→Low Power and Area Overheads



General-Purpose Out-of-Order Processors

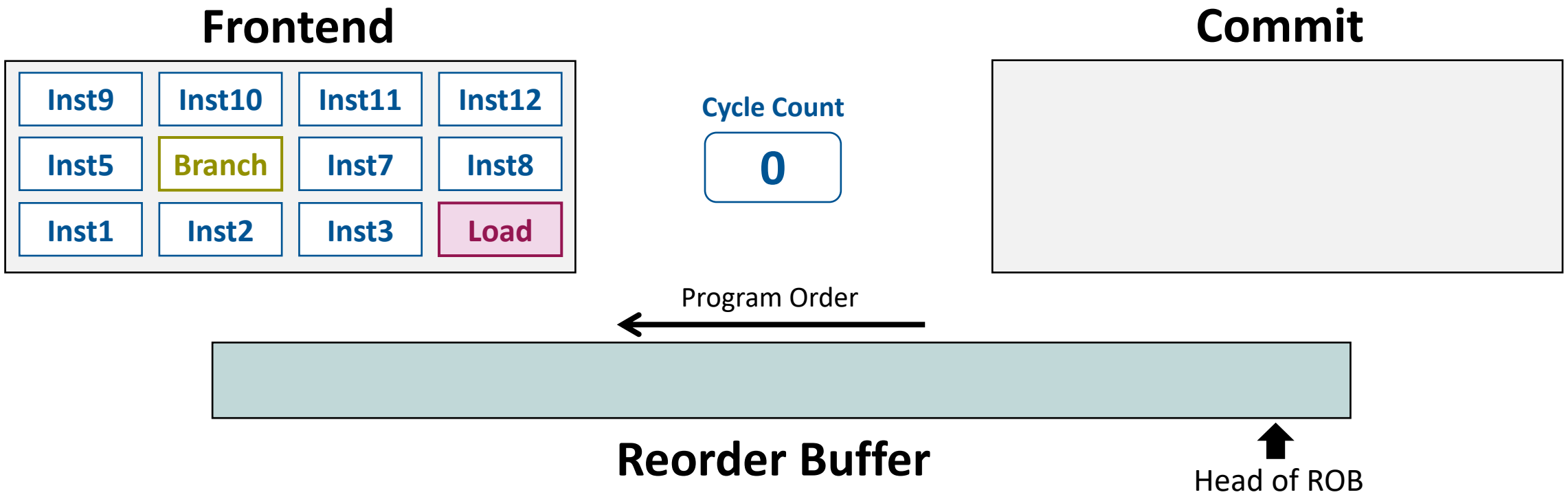
- End of Moore's law requires efficient computing
- However, general-purpose CPUs still have a significant impact on the overall performance; Hard-to-parallelize work is left for the CPU[1]
- Re-thinking the traditional design to unlock efficiency:
  - *Co-design the different layers of the system*



1. Arora, Manish, et al. "Redefining the Role of the CPU in the Era of CPU-GPU Integration." IEEE Micro 32.6 (2012): 4-16.

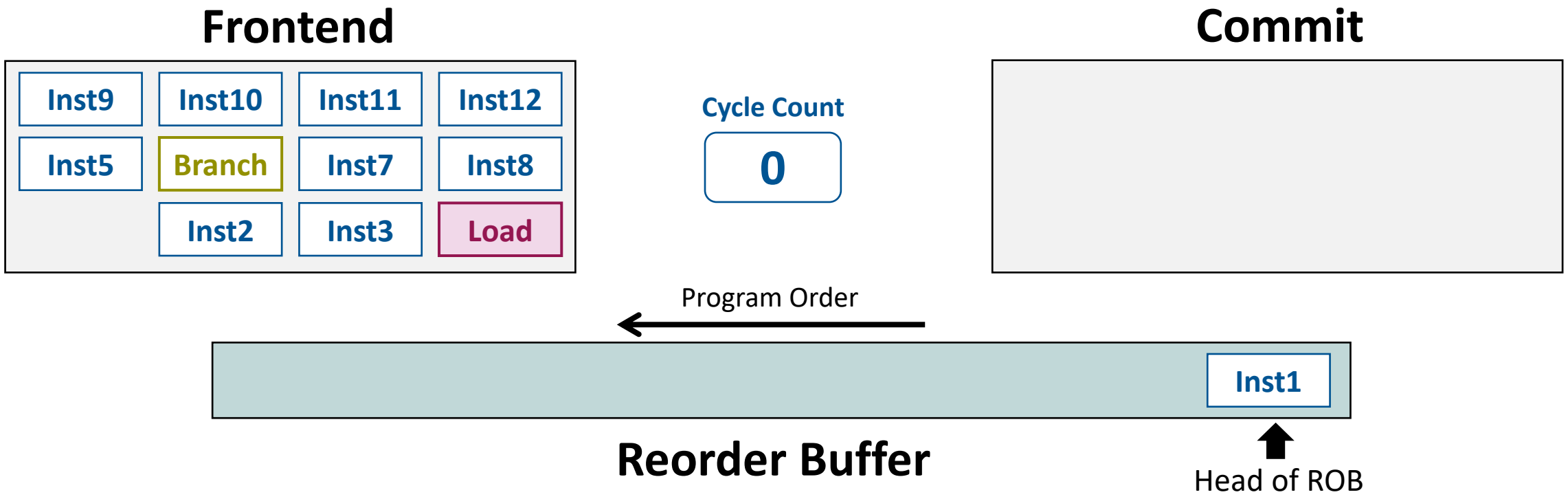
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



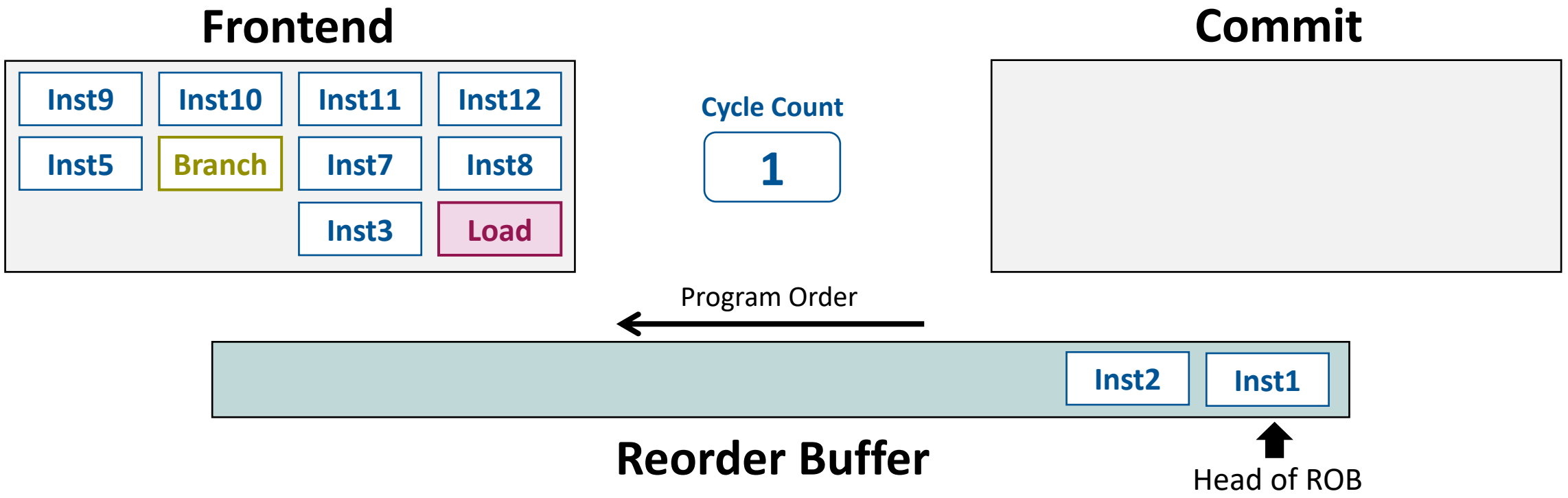
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



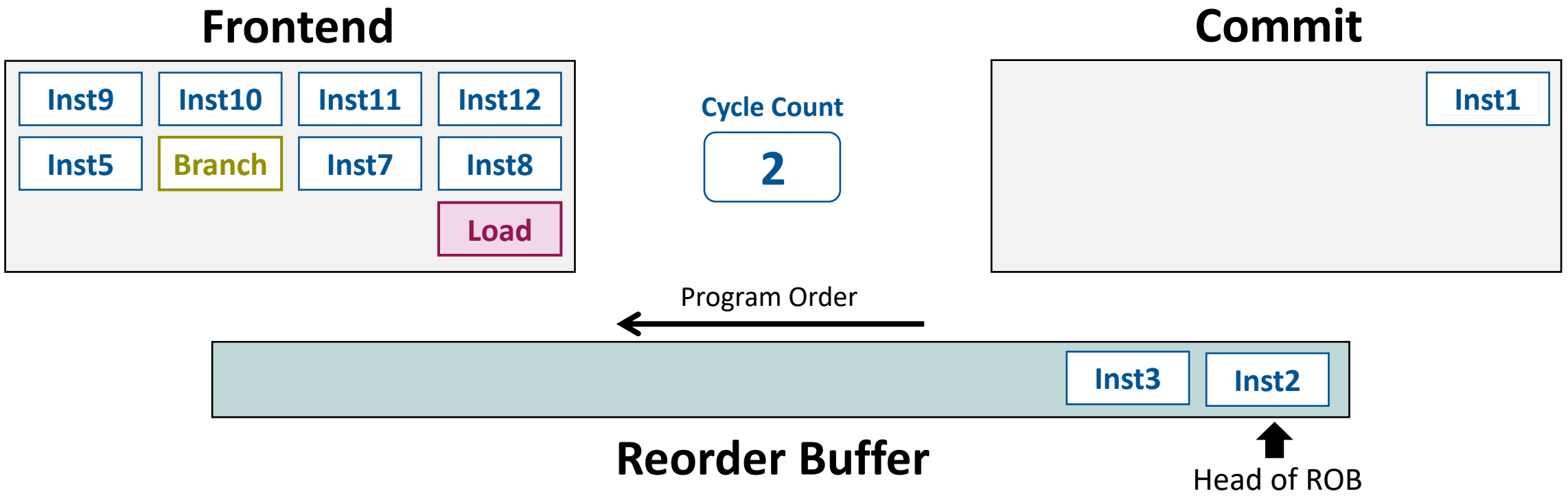
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



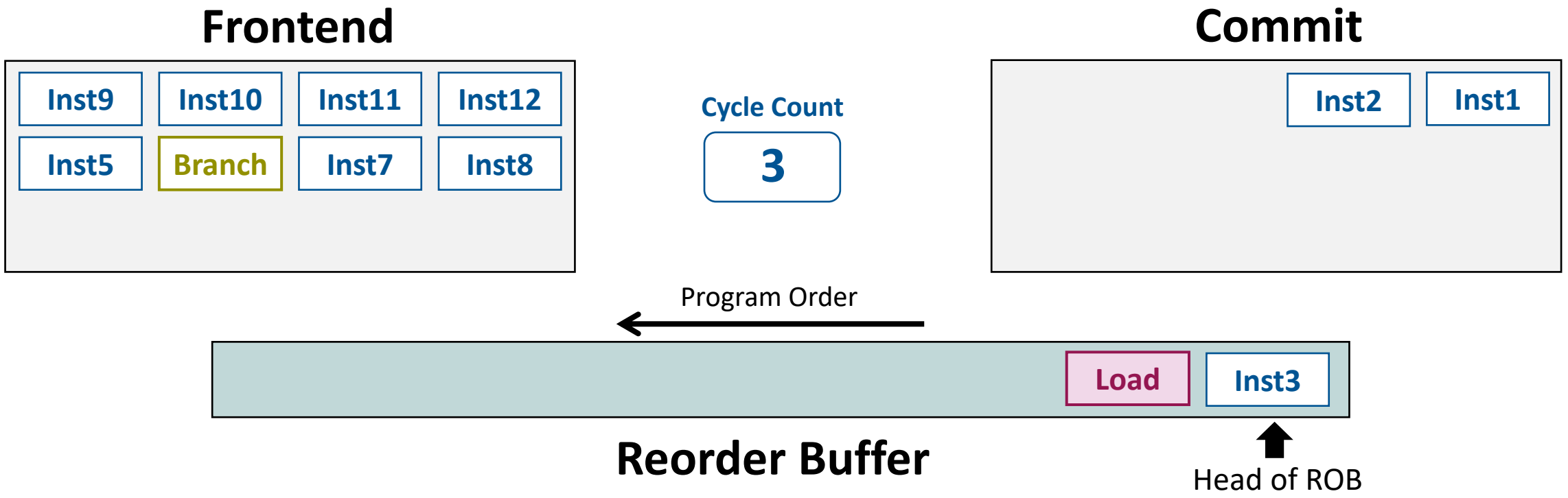
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



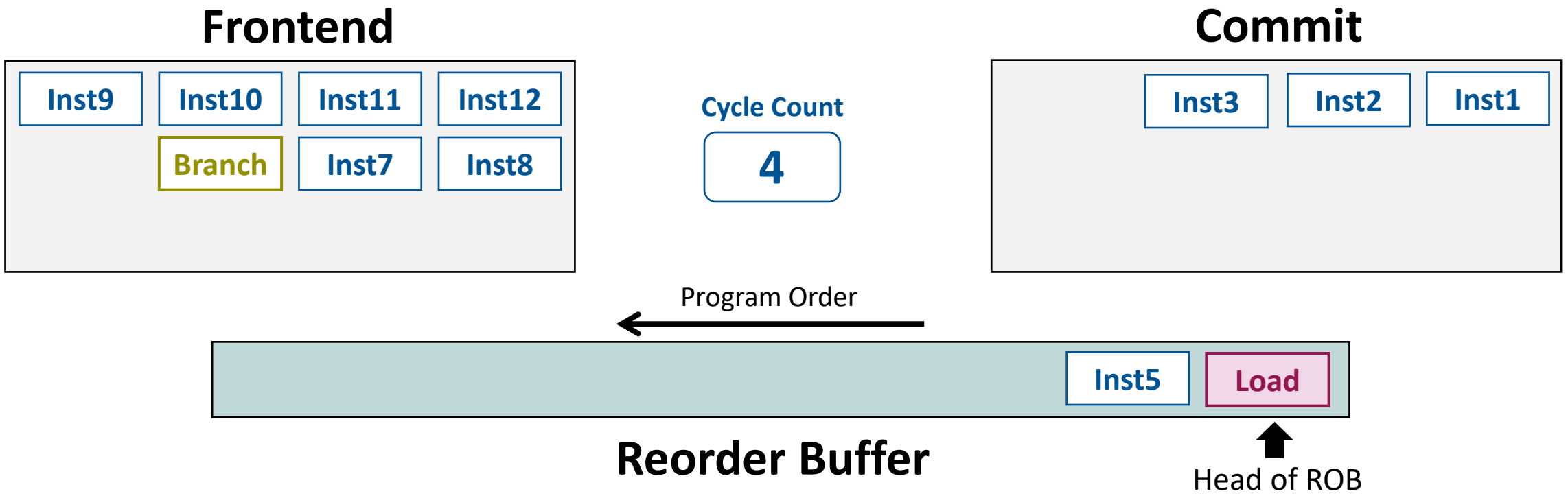
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



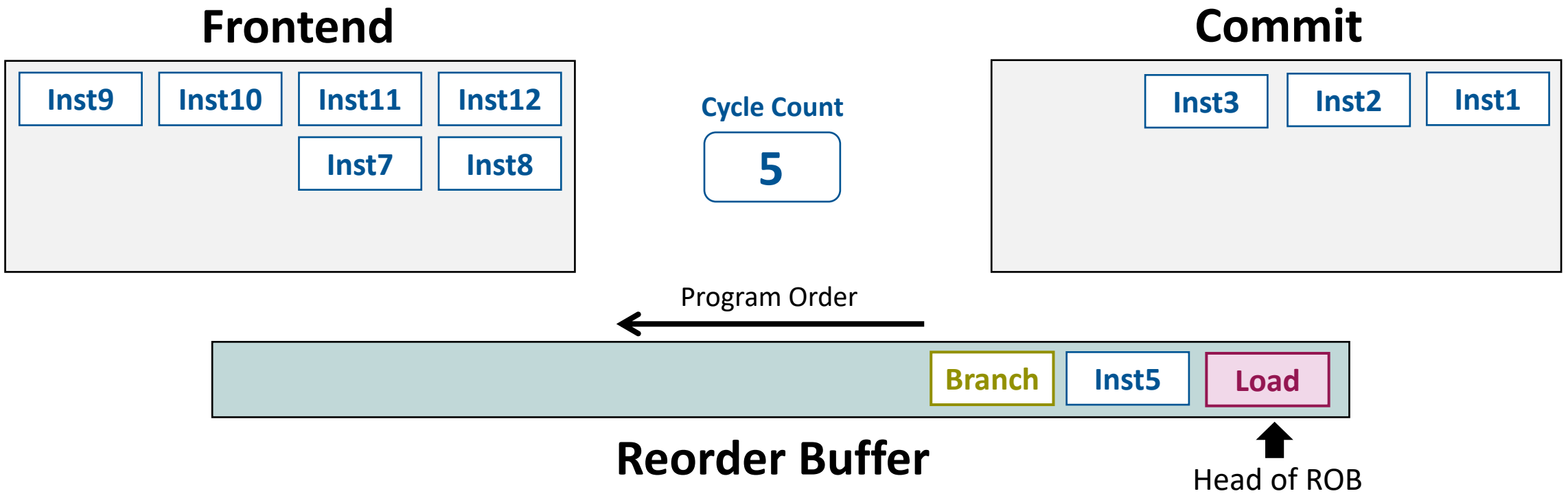
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



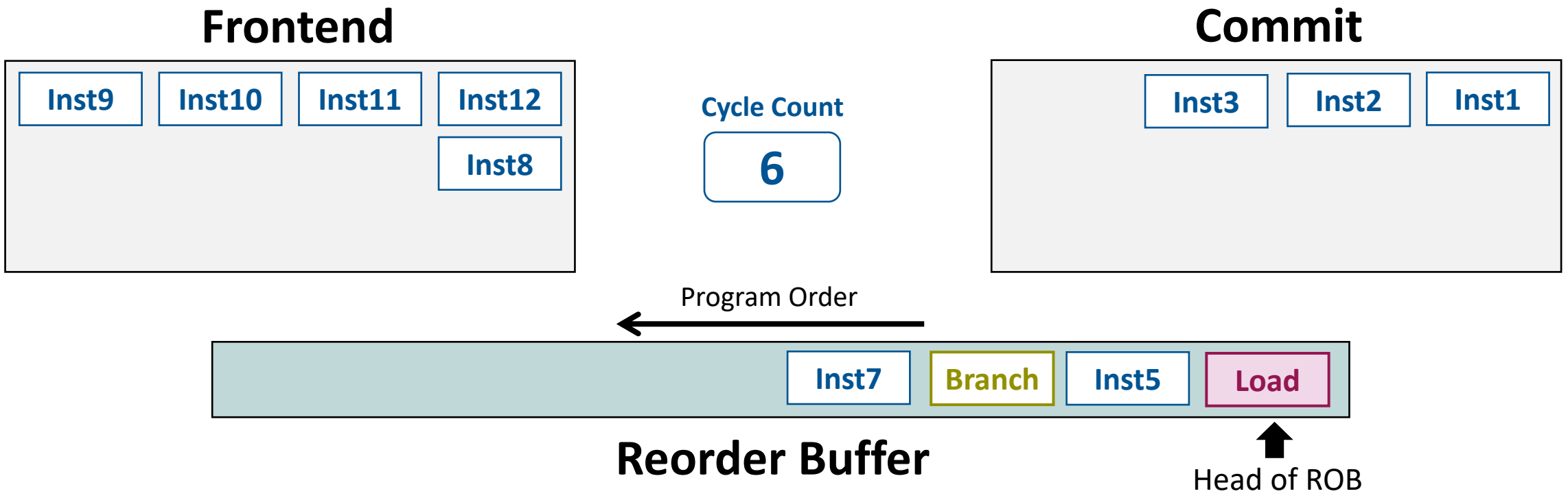
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



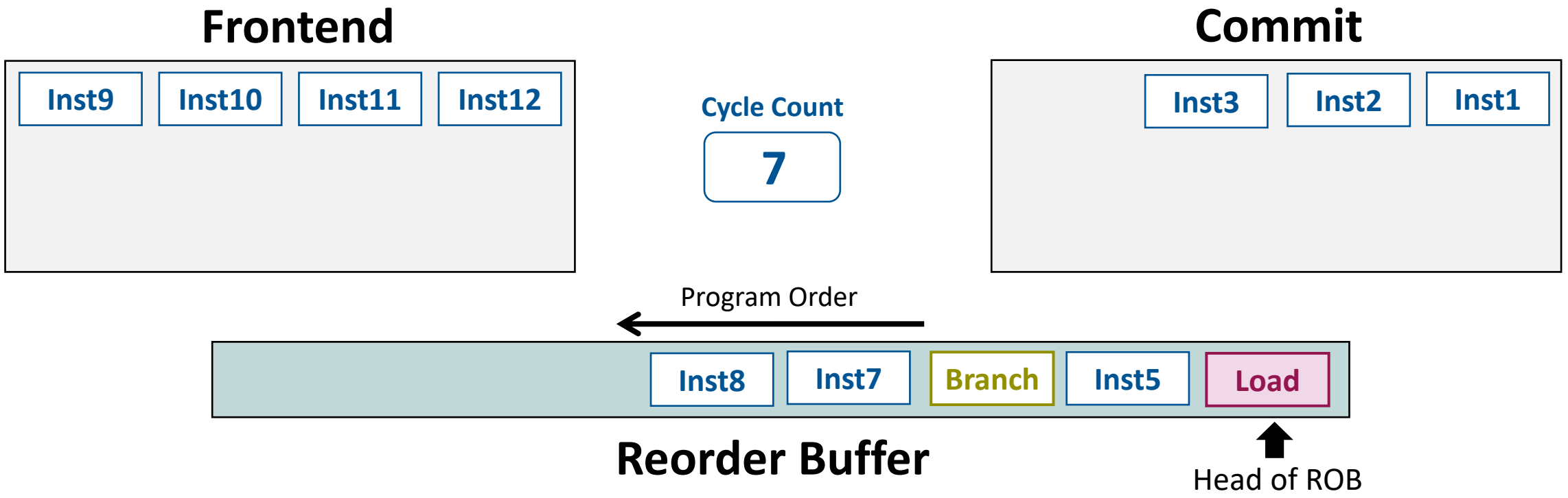
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



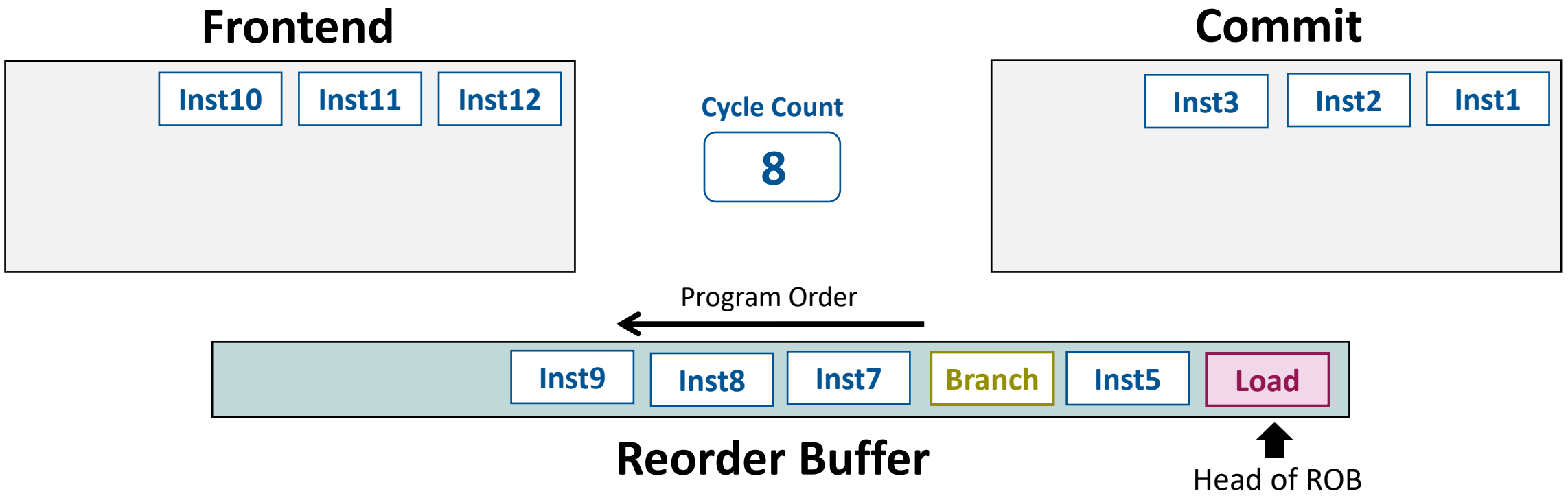
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



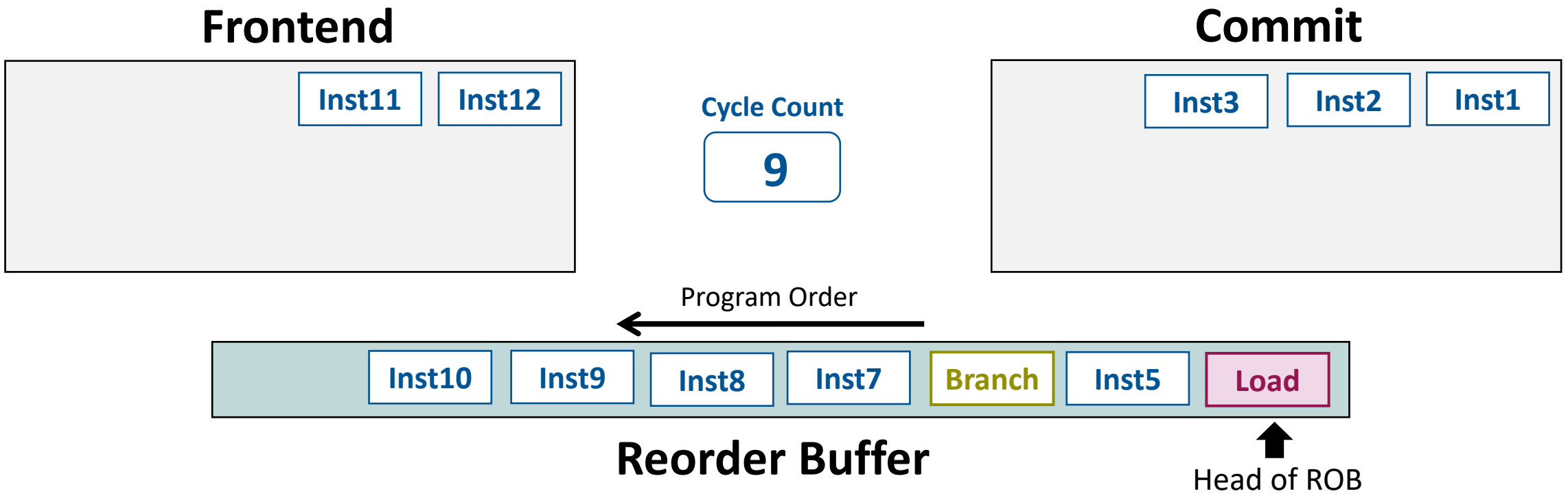
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



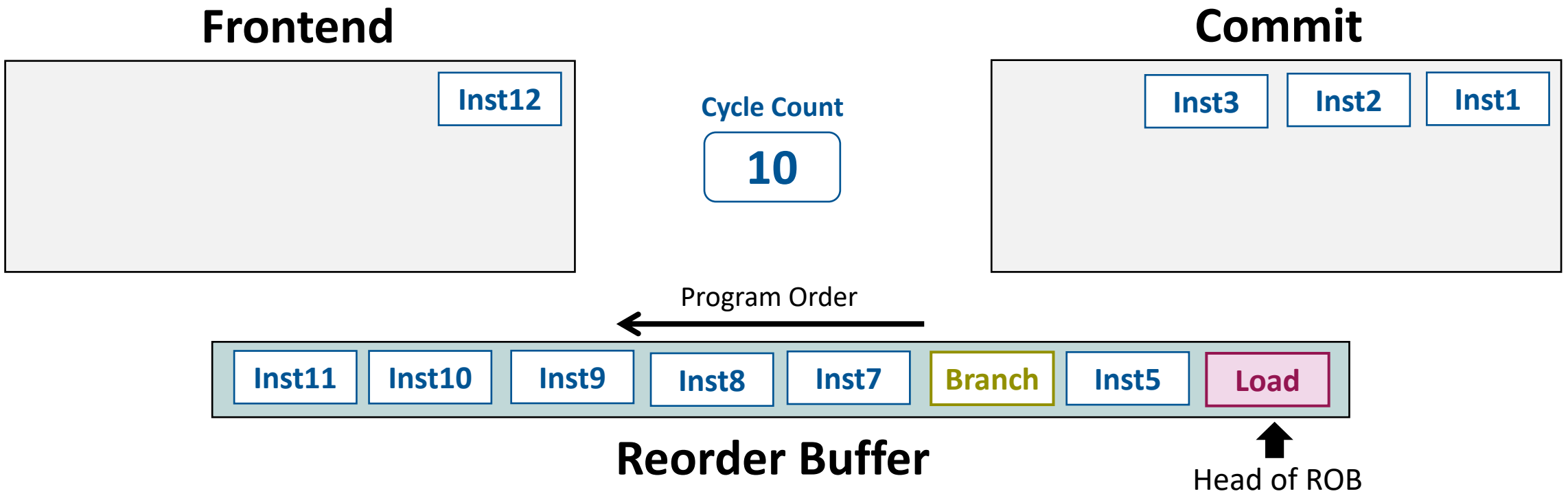
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



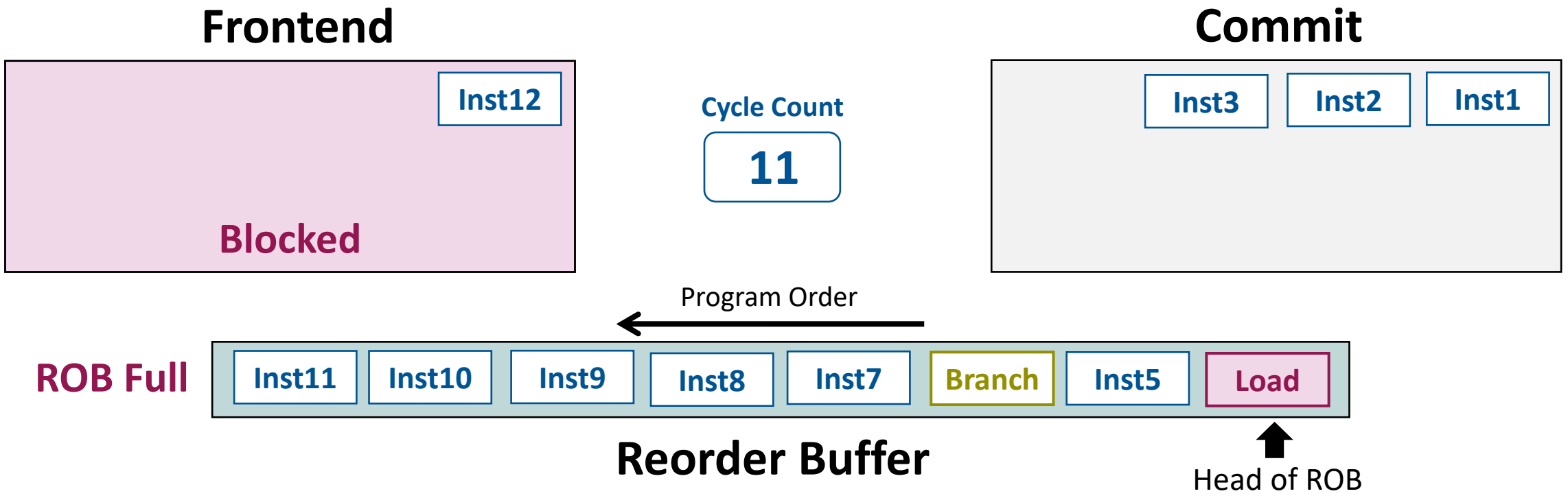
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



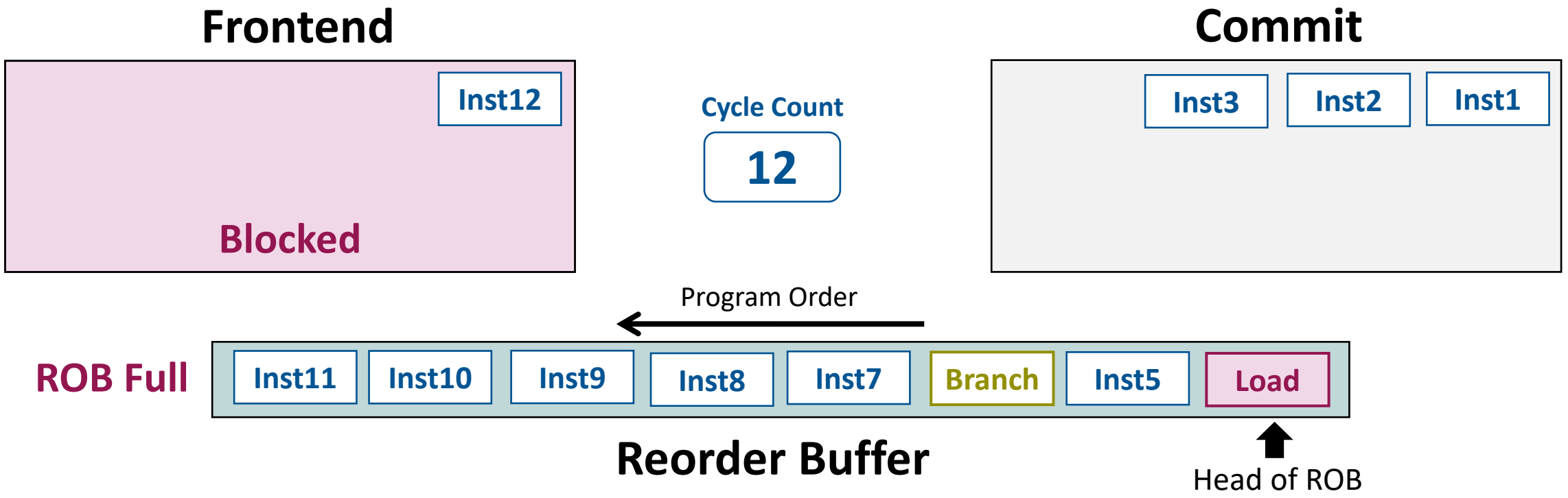
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



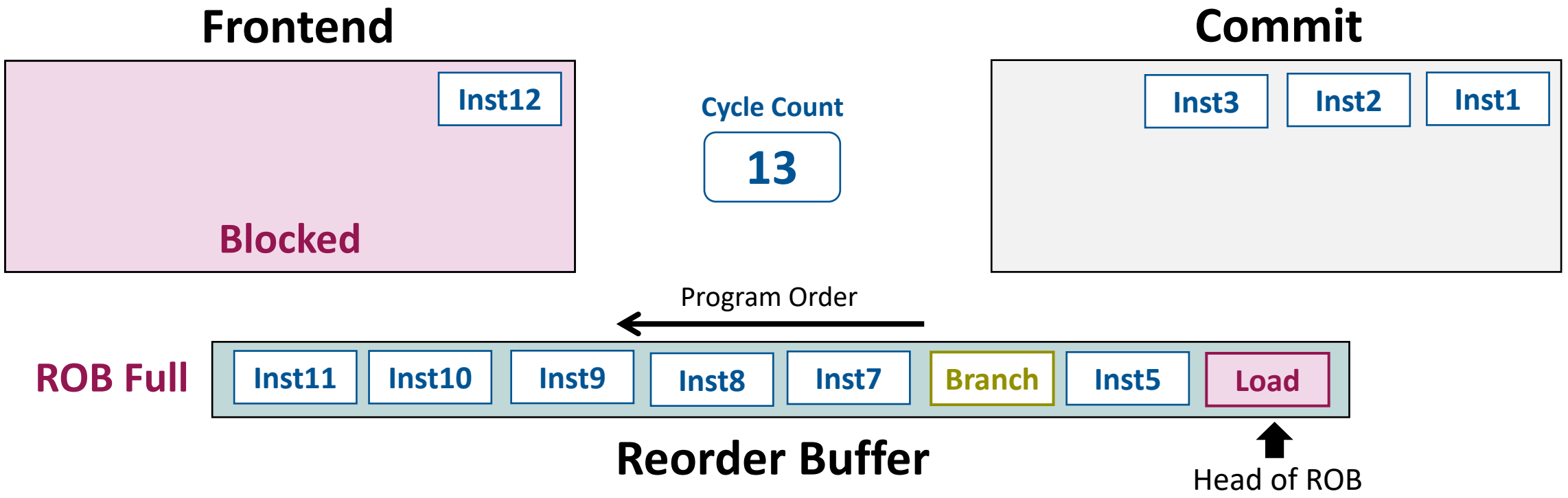
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



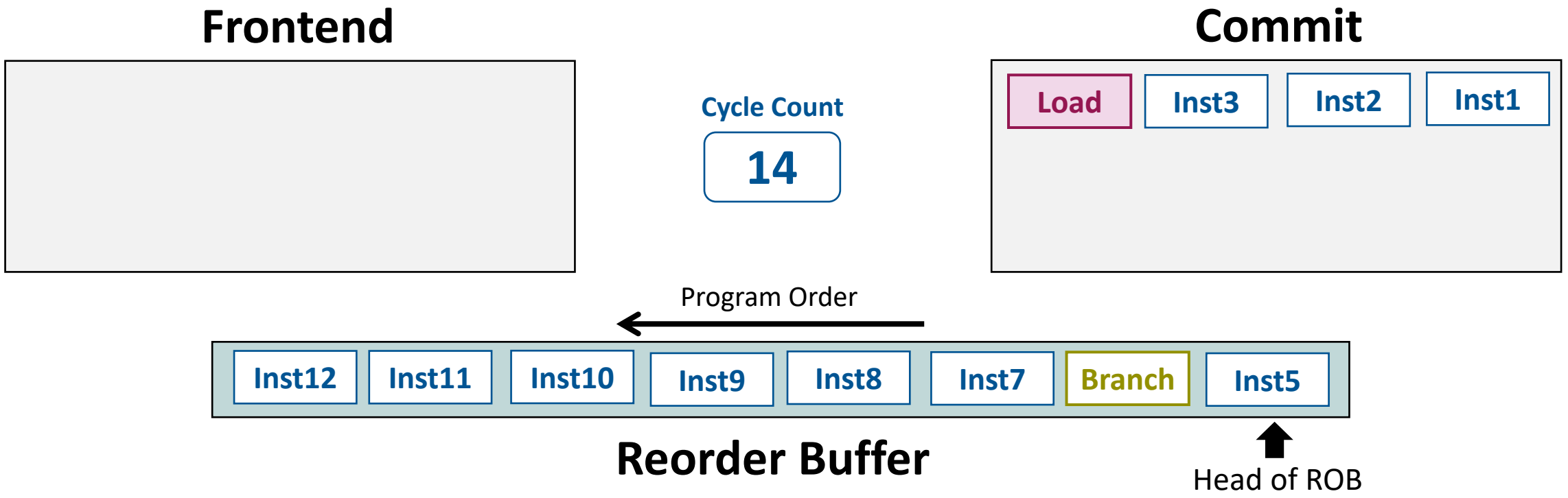
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



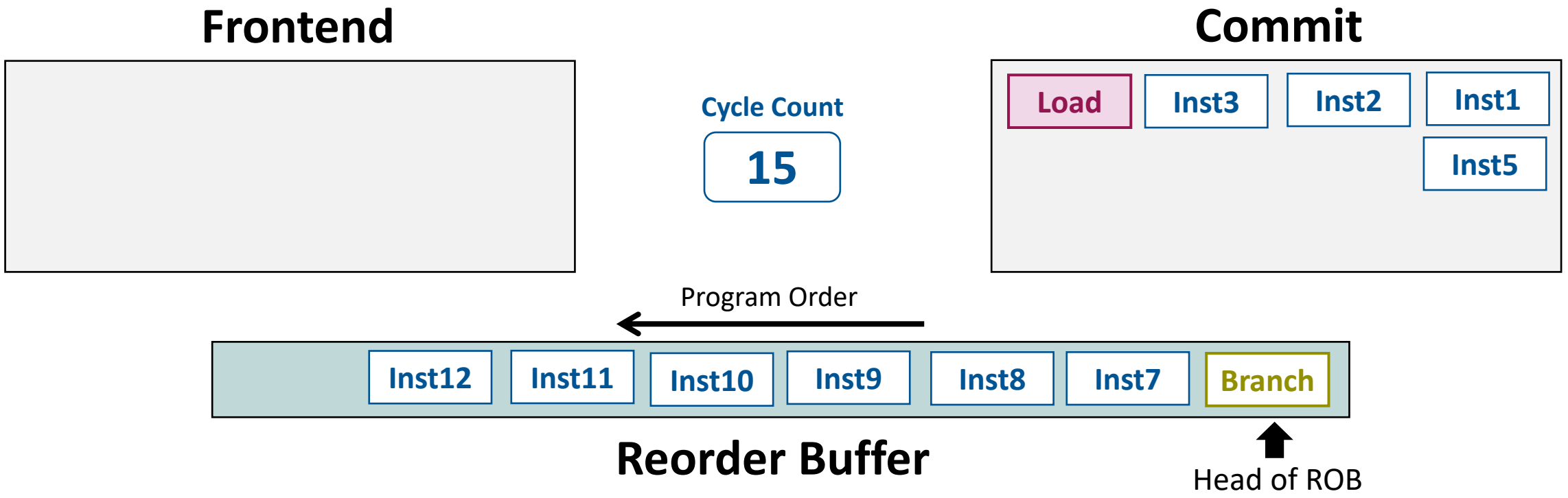
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



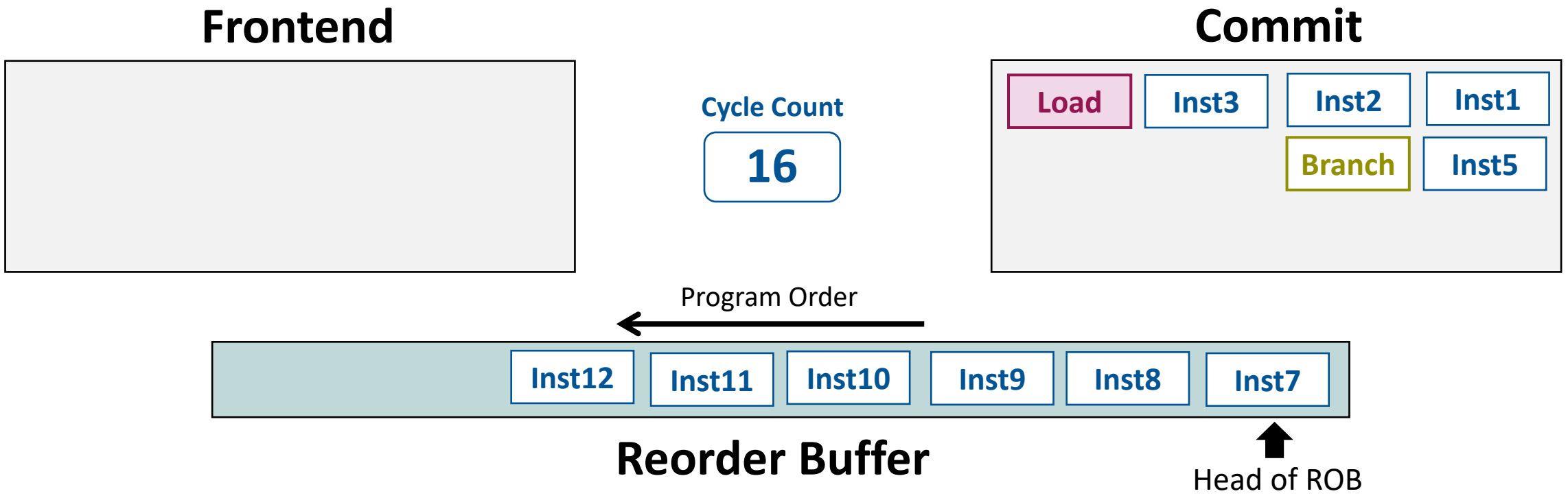
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



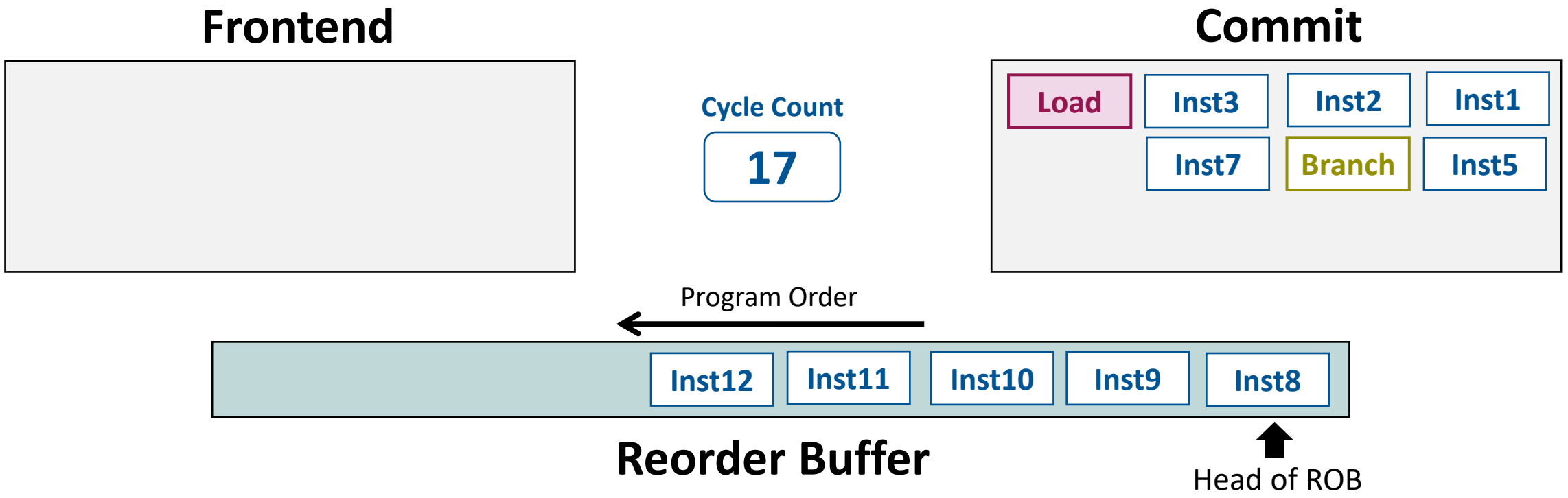
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



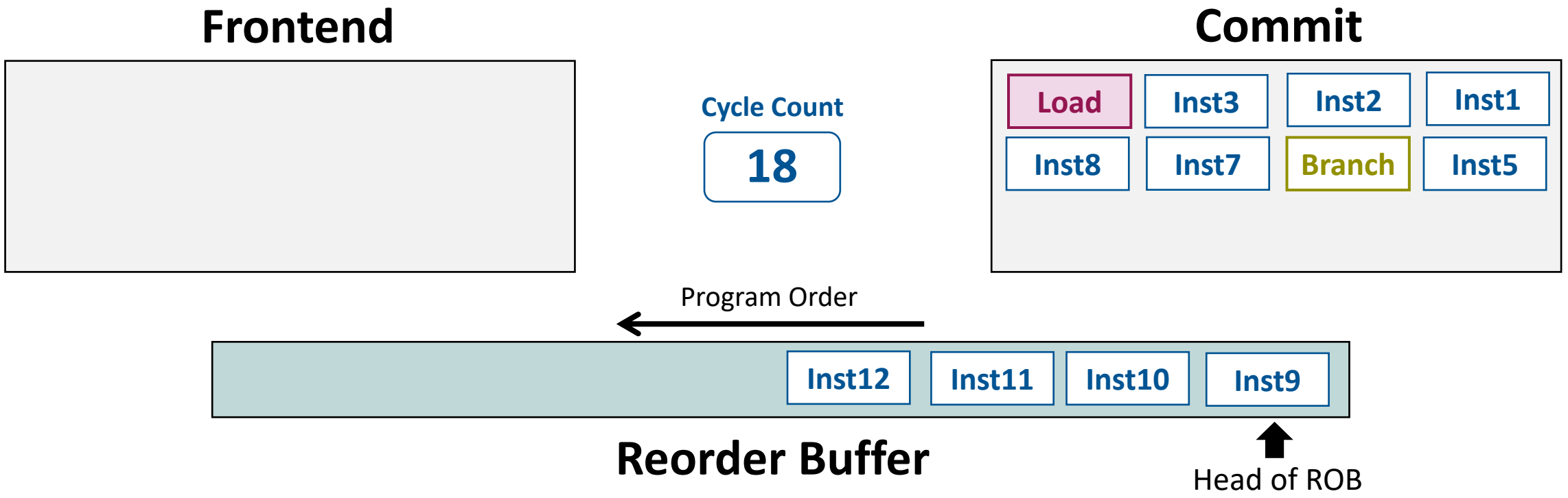
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



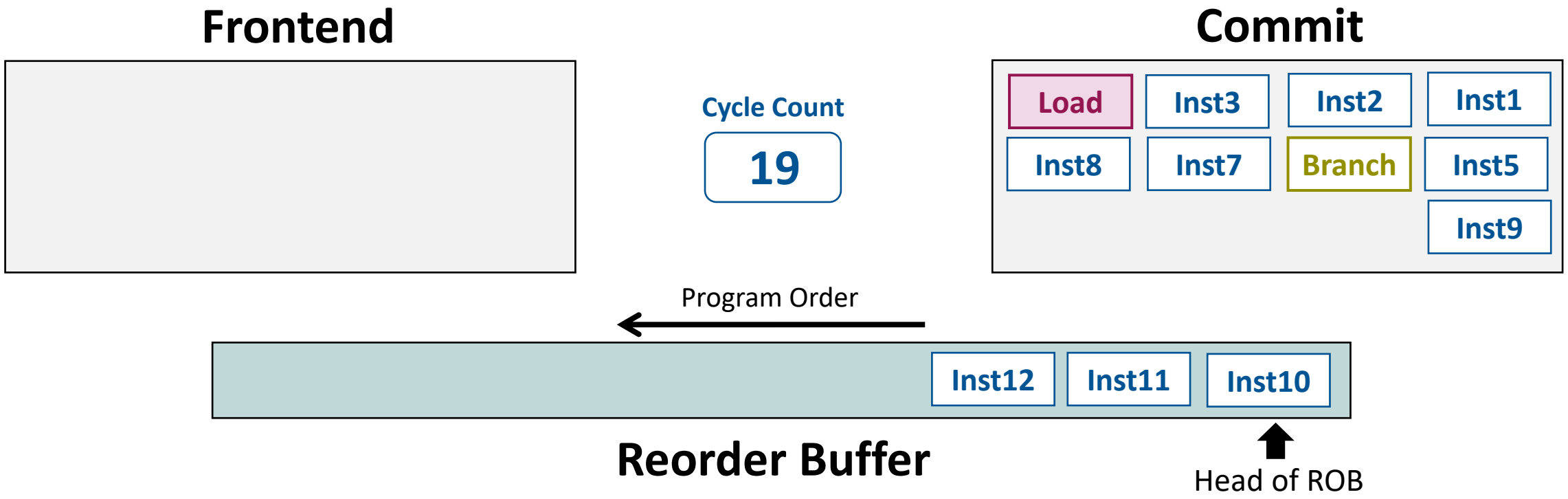
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



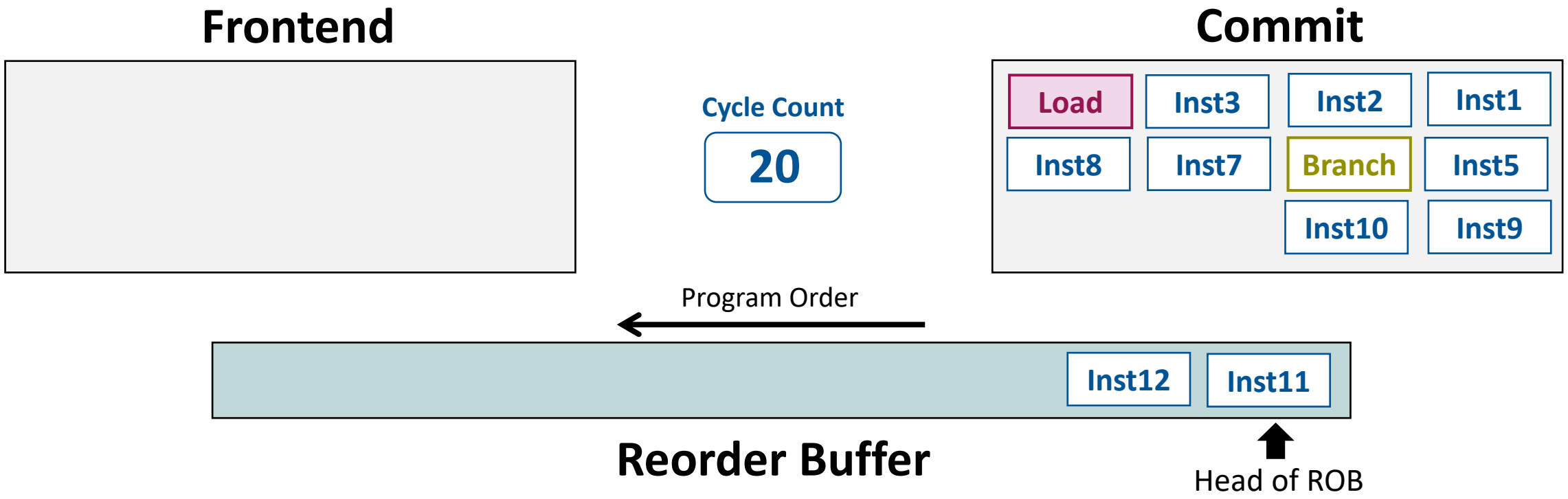
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



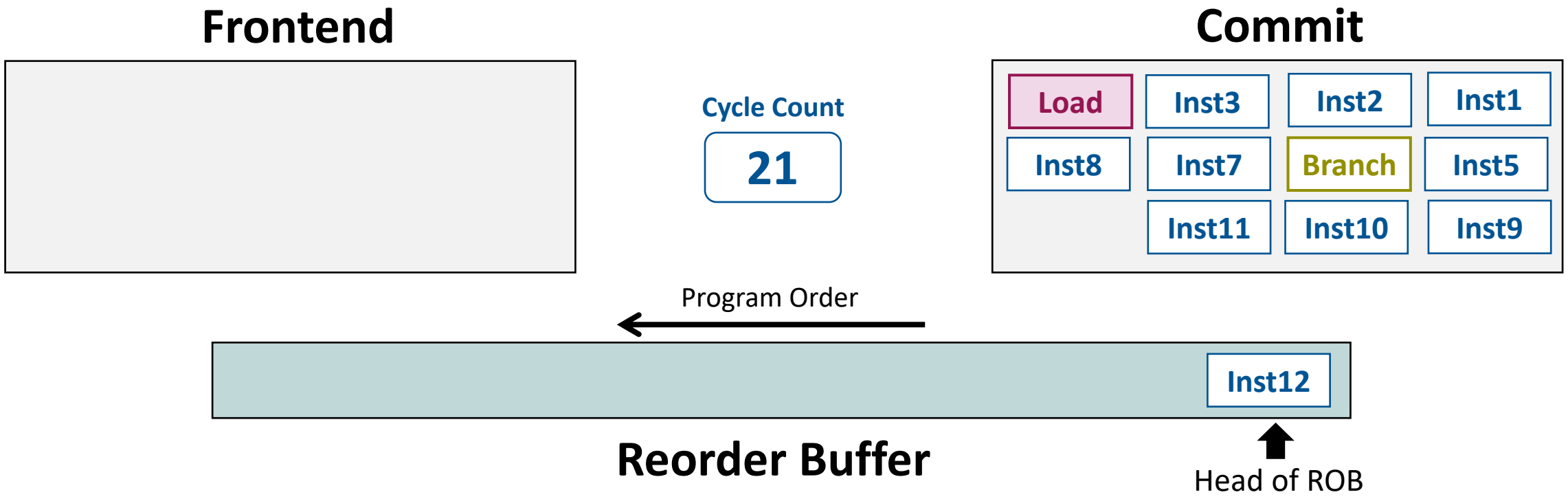
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



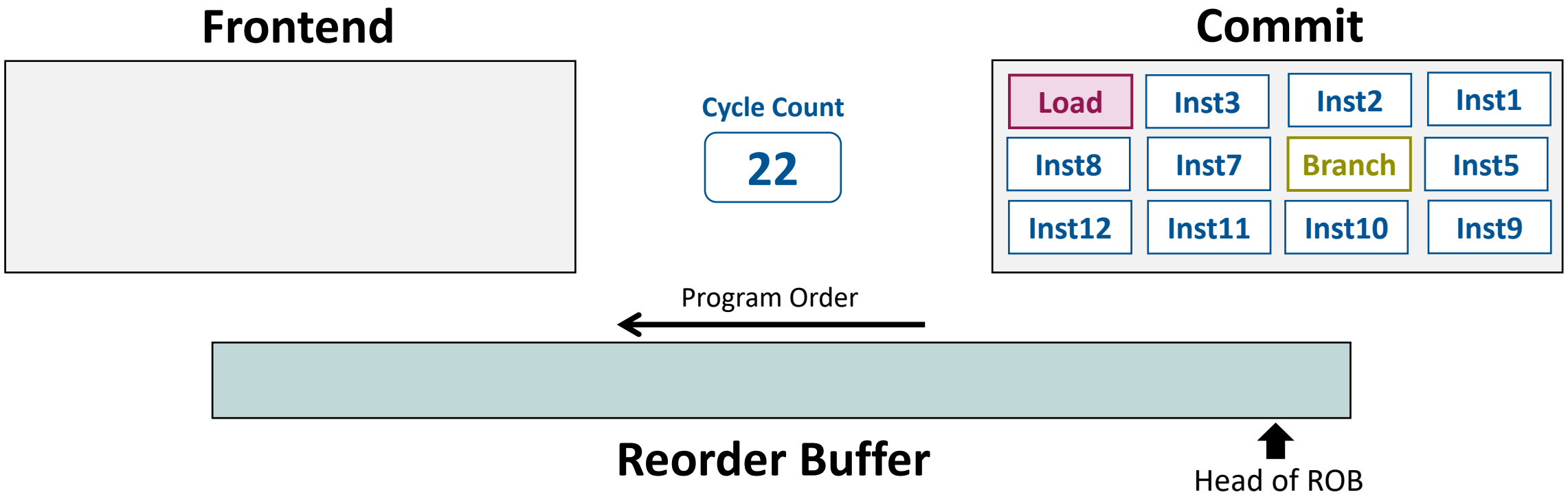
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



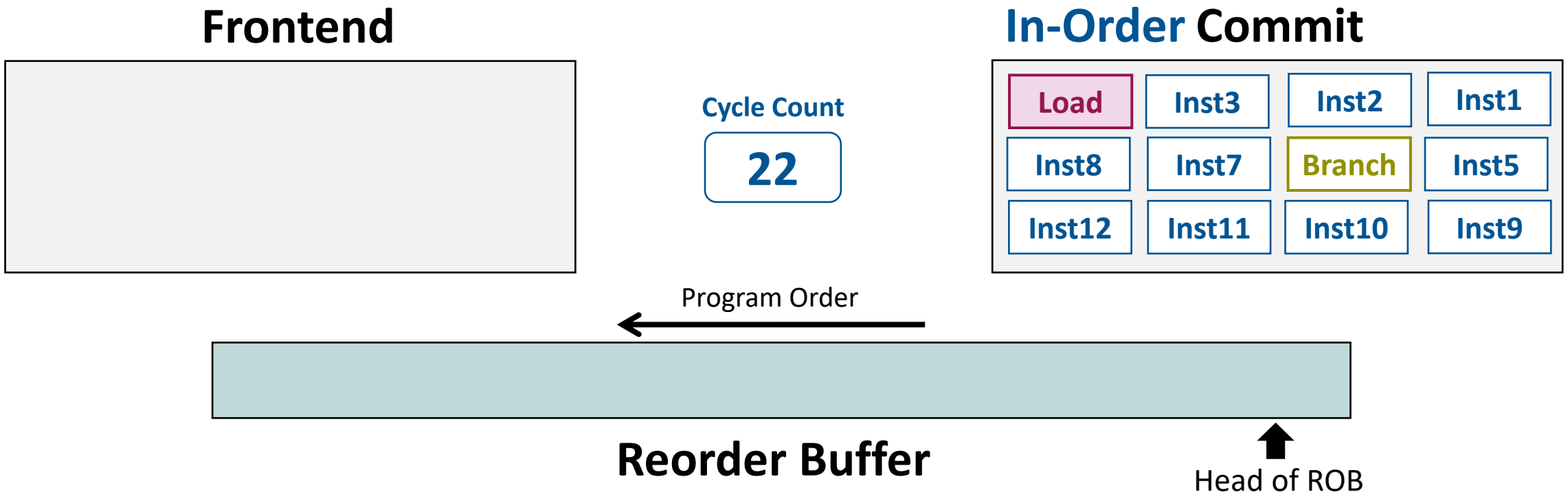
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



ROB in Traditional General-Purpose Processors

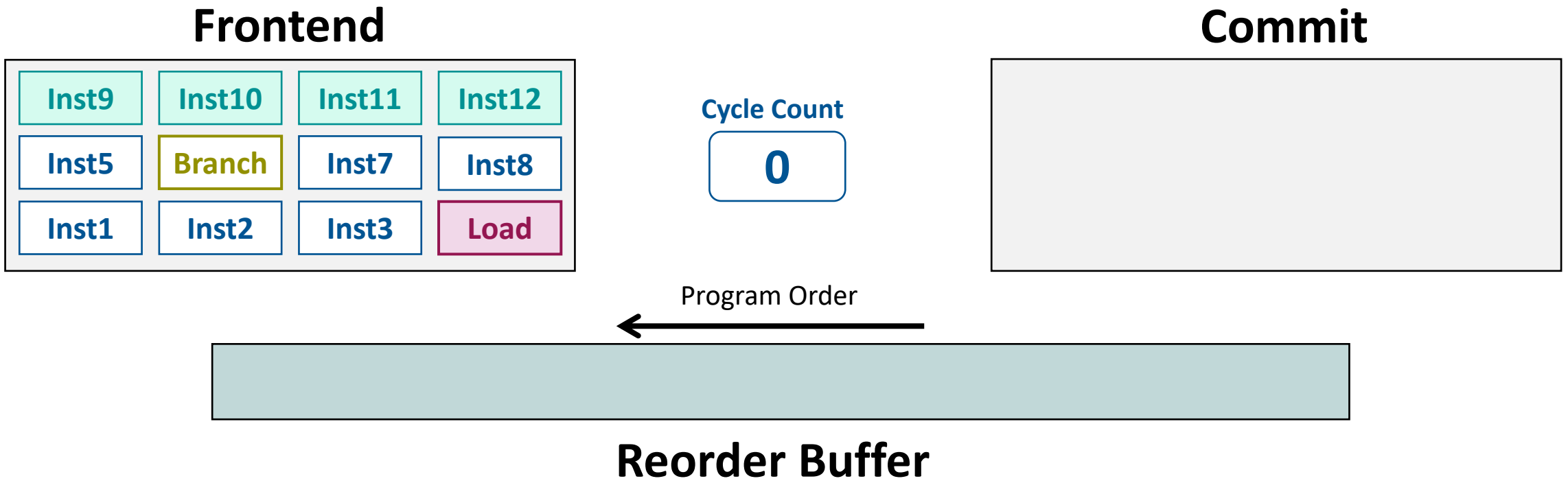
Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8



Alternative Approach

In-Order Commit is conservative

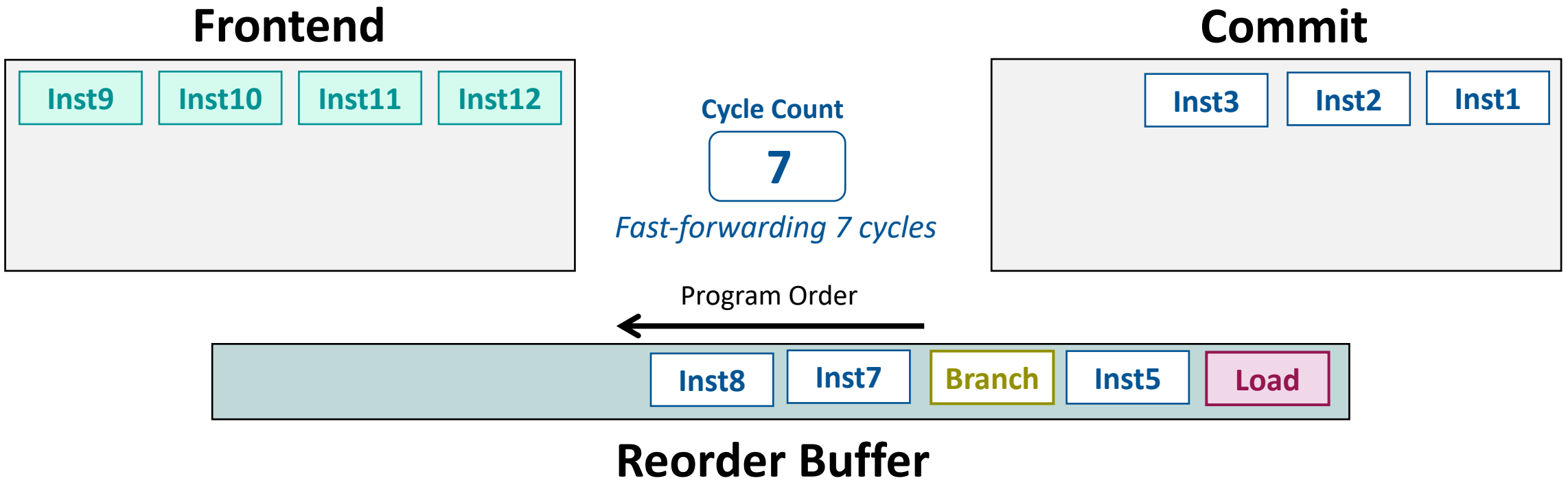
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

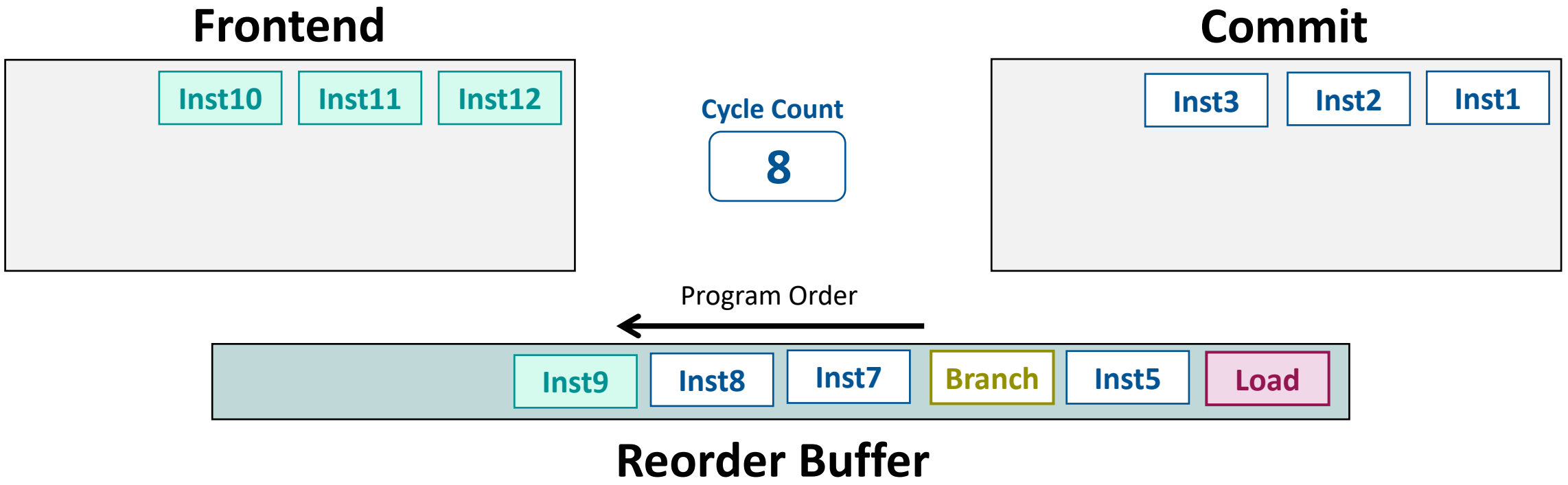
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

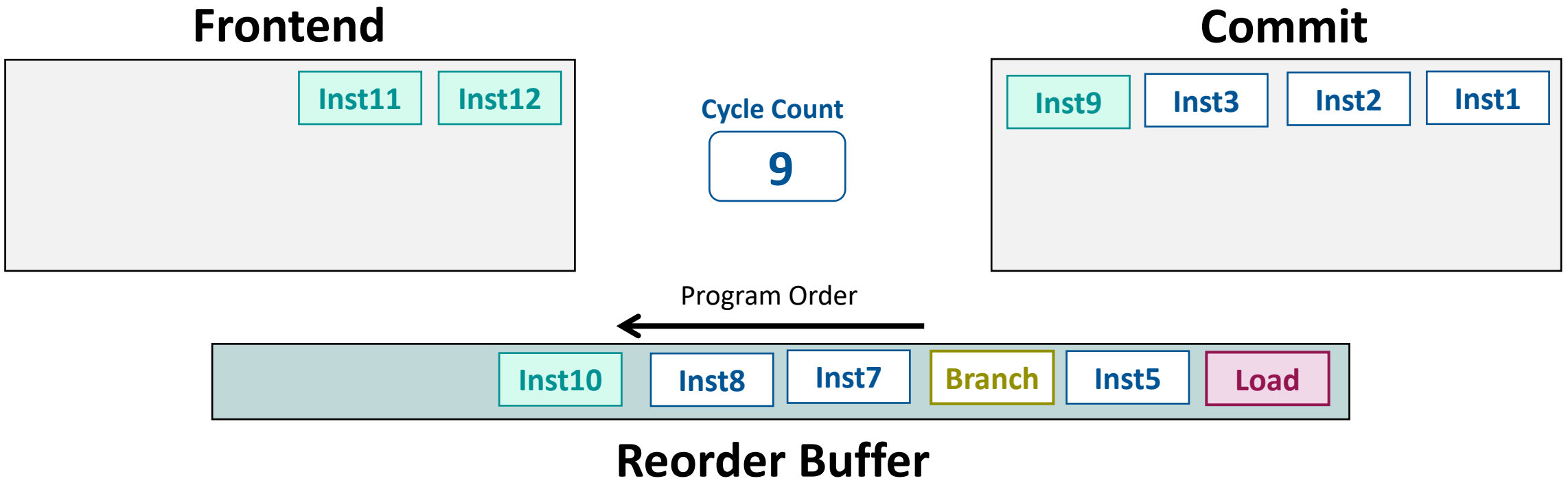
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

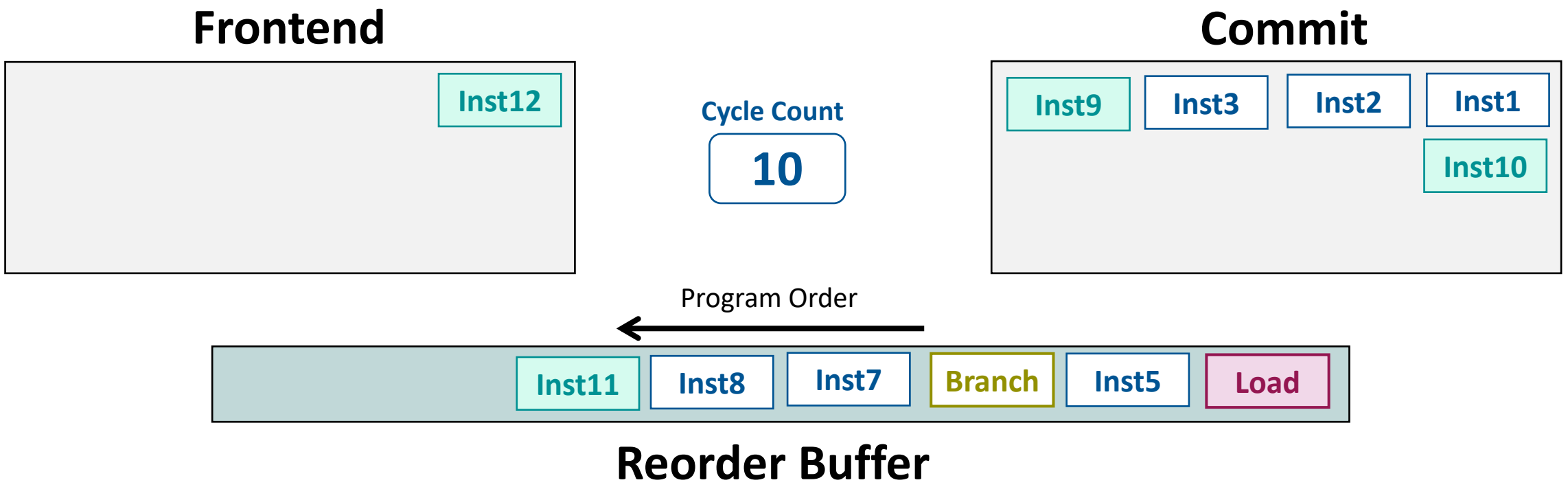
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

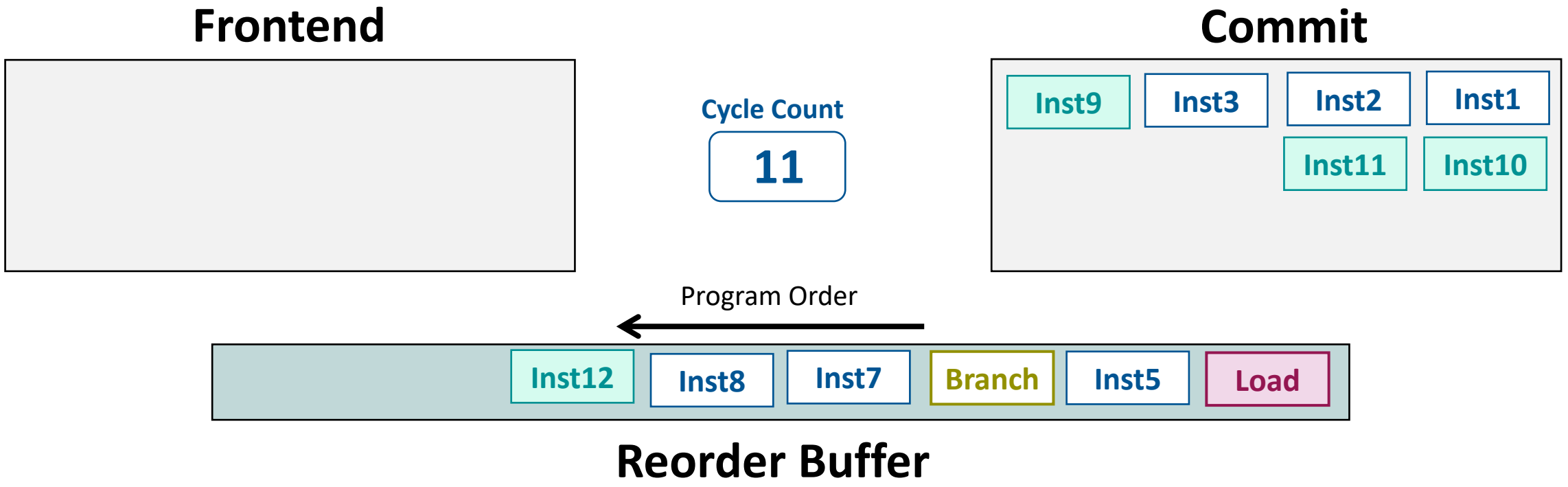
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

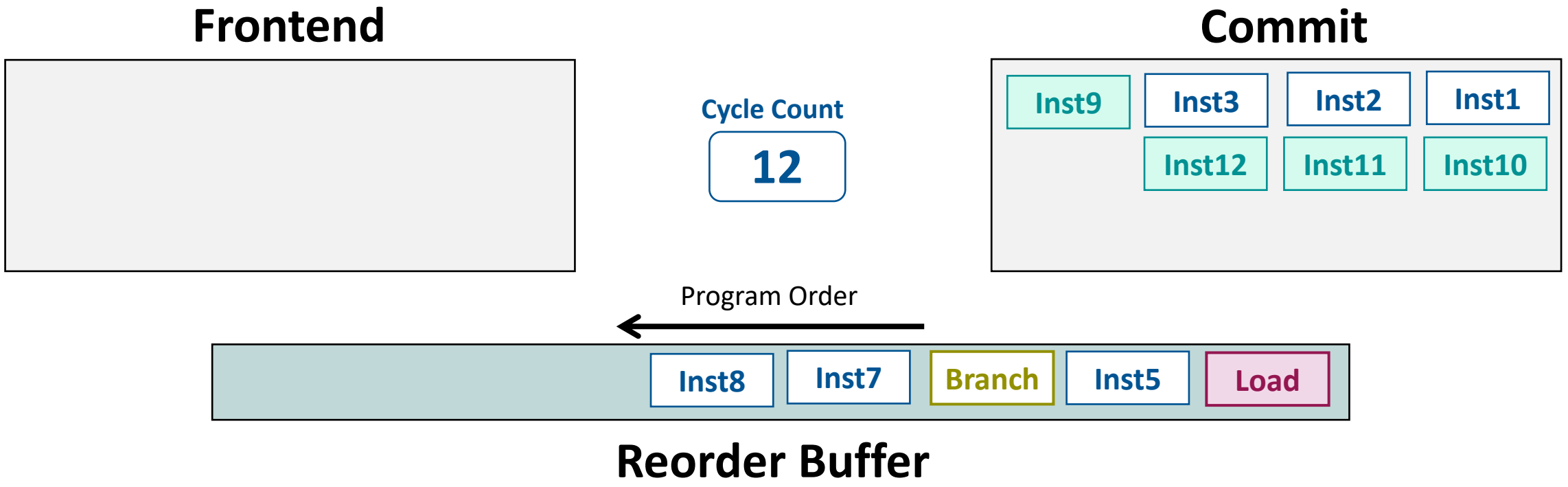
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

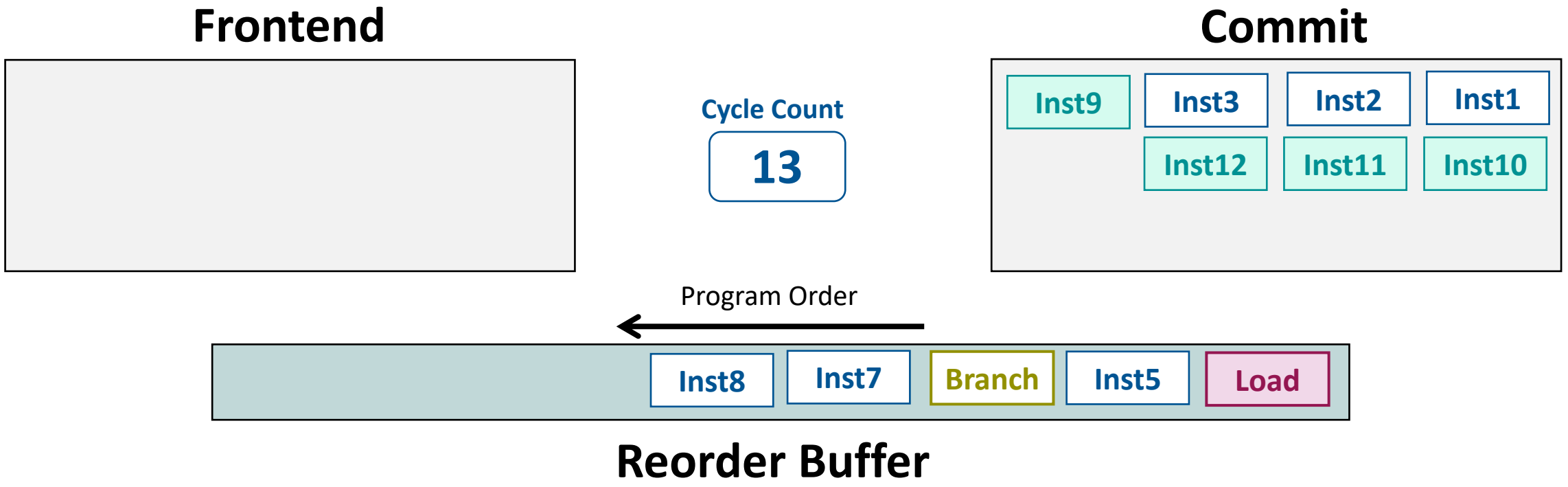
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

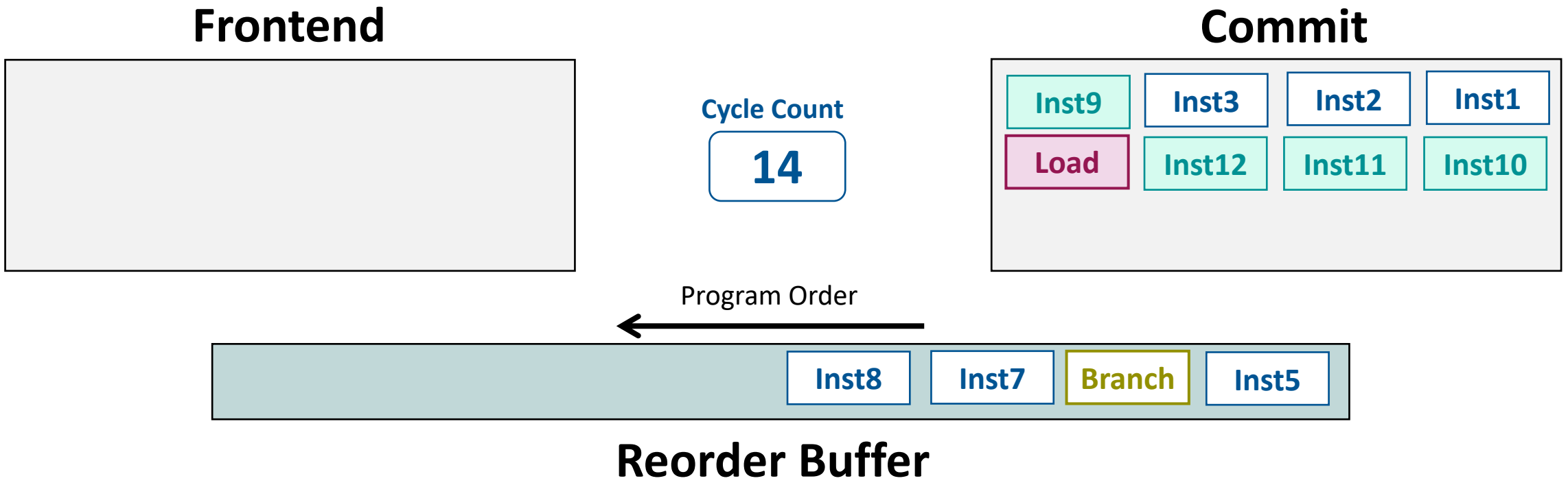
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

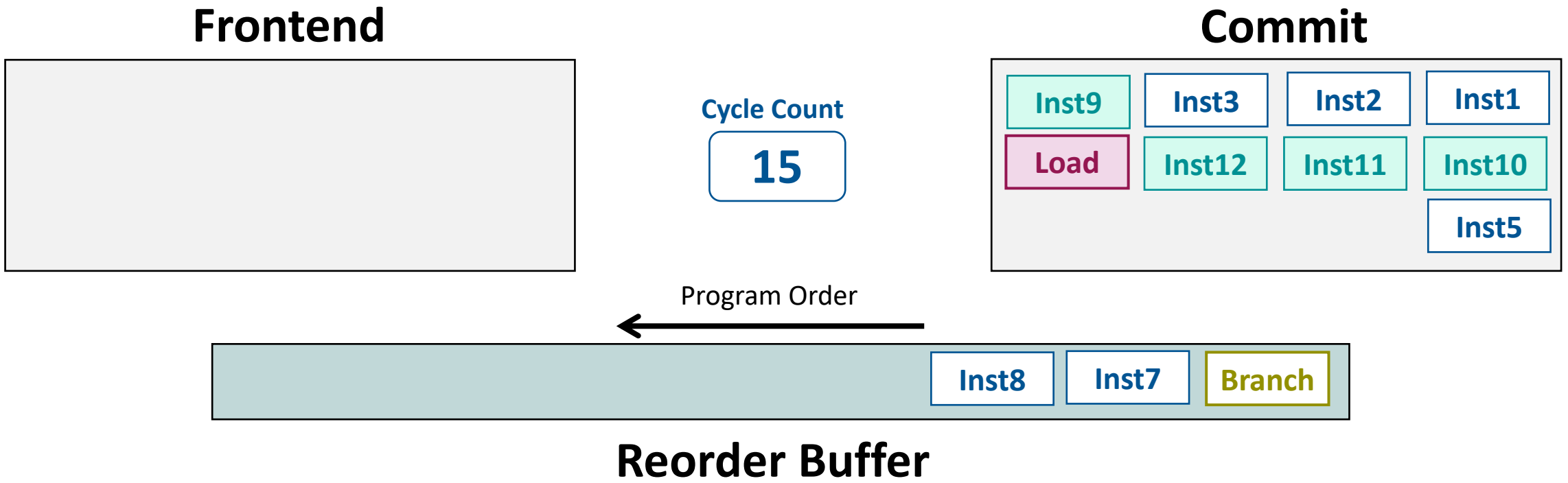
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

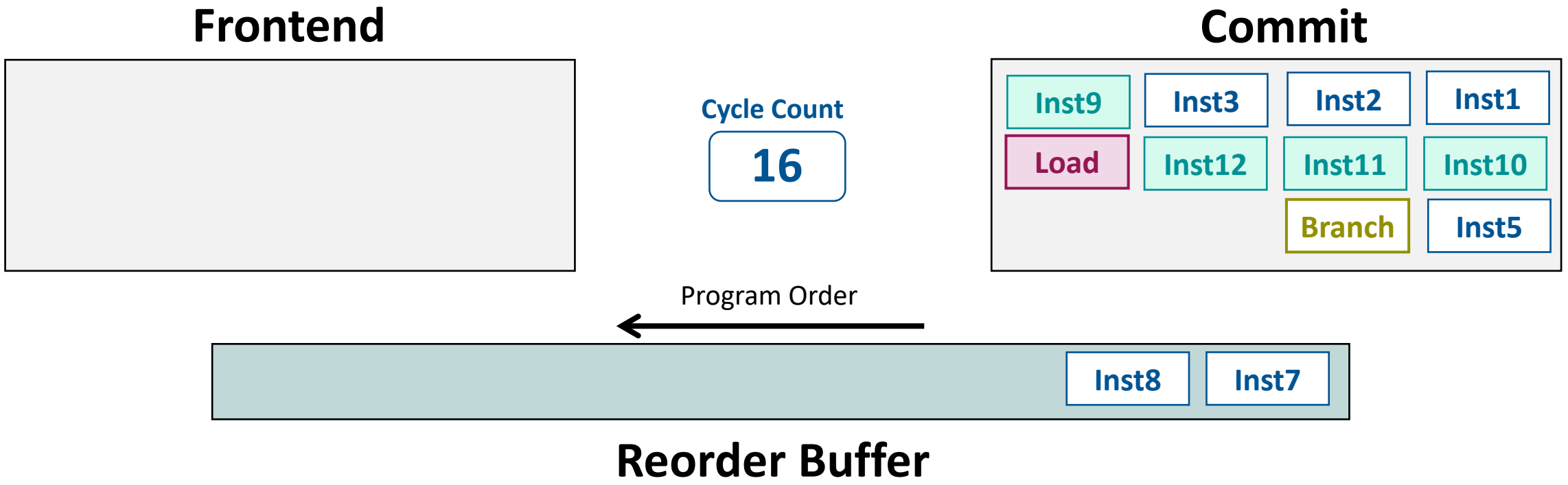
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

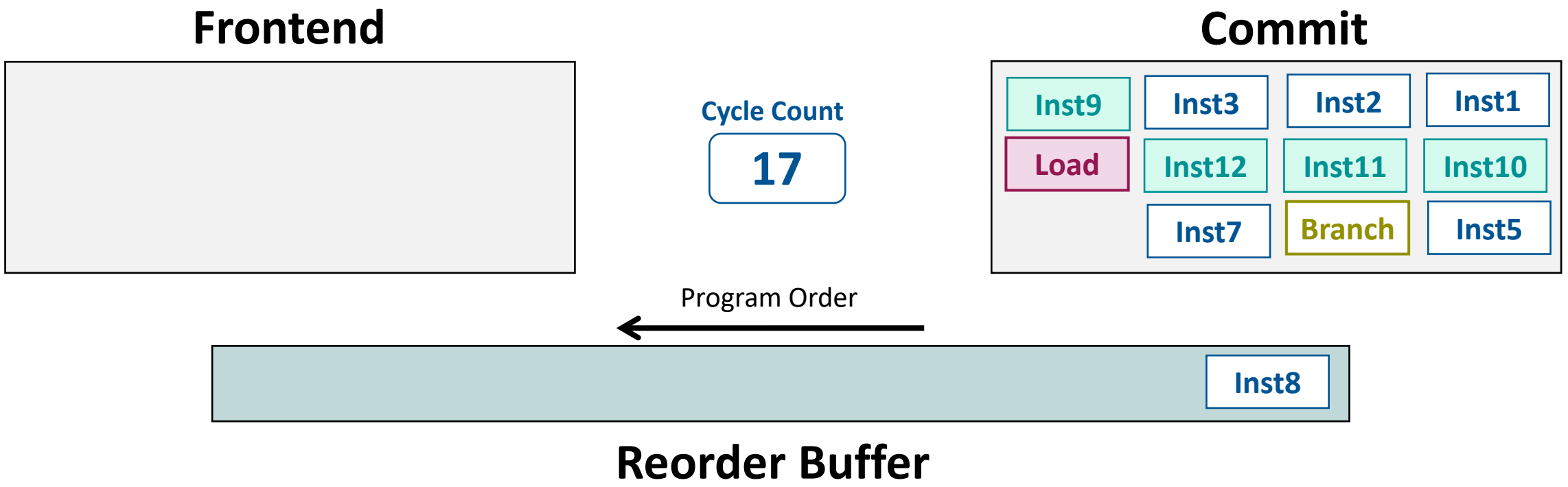
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

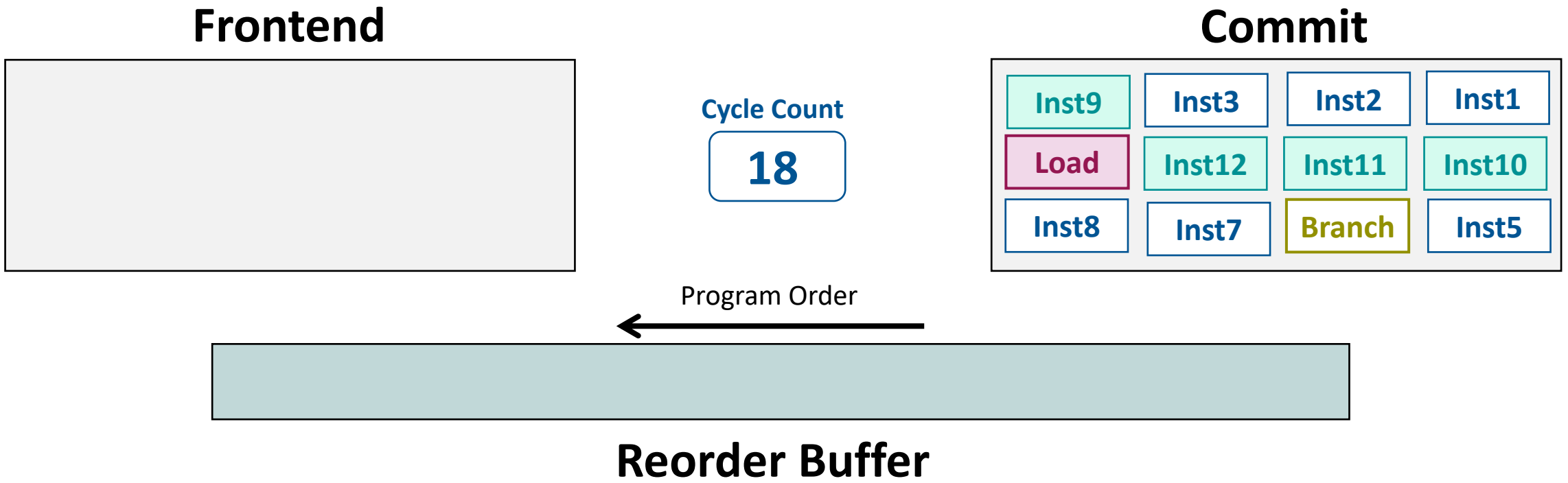
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

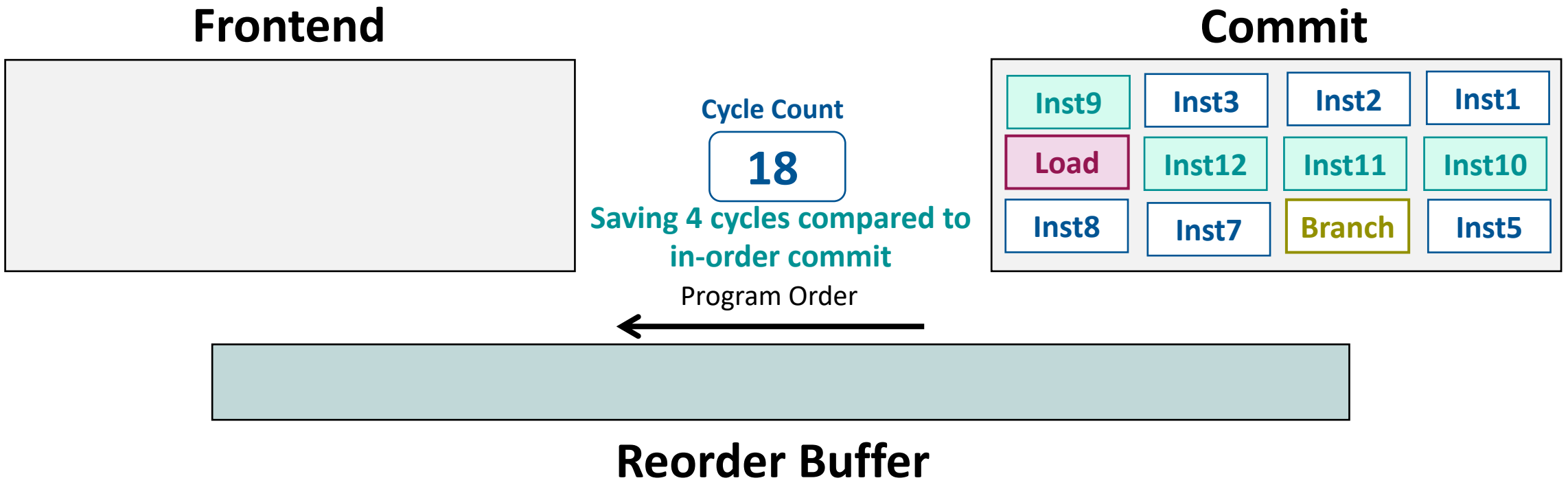
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

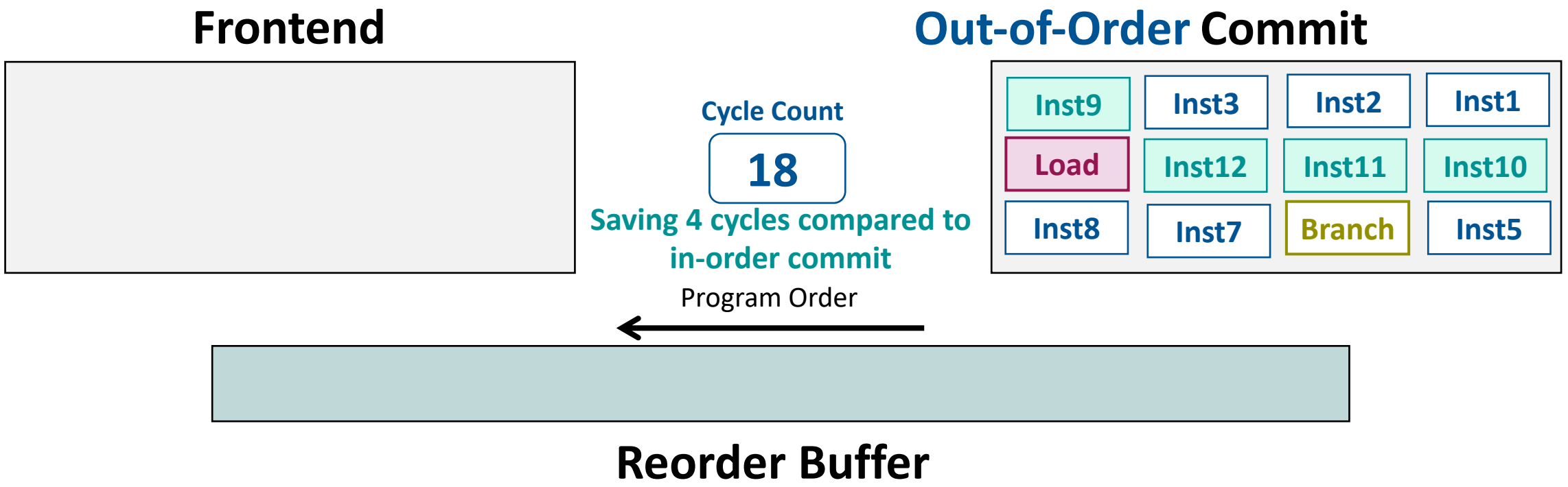
What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



Alternative Approach

In-Order Commit is conservative

What if **Inst9**, **Inst10**, **Inst11**, and **Inst12** are independent from **Branch**?



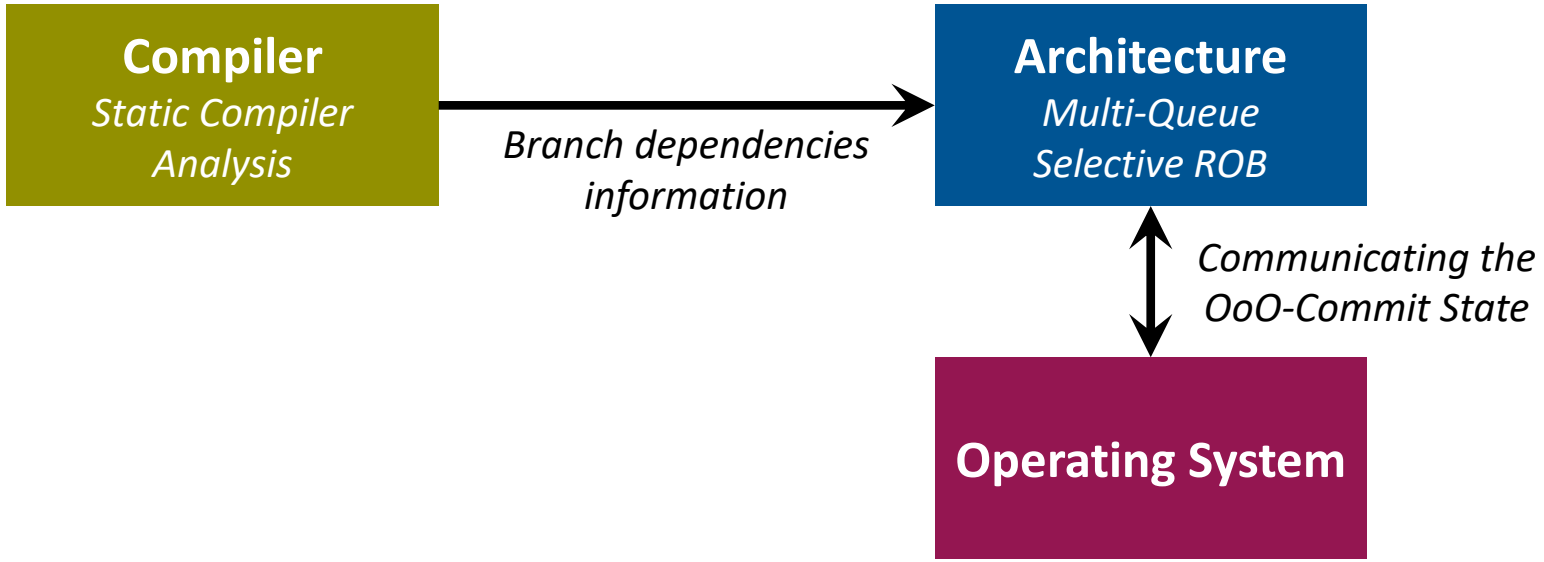
NOREBA: HW/SW Co-operative OoO-Commit

Questions

How to detect branch dependecies ***non–speculatively***?

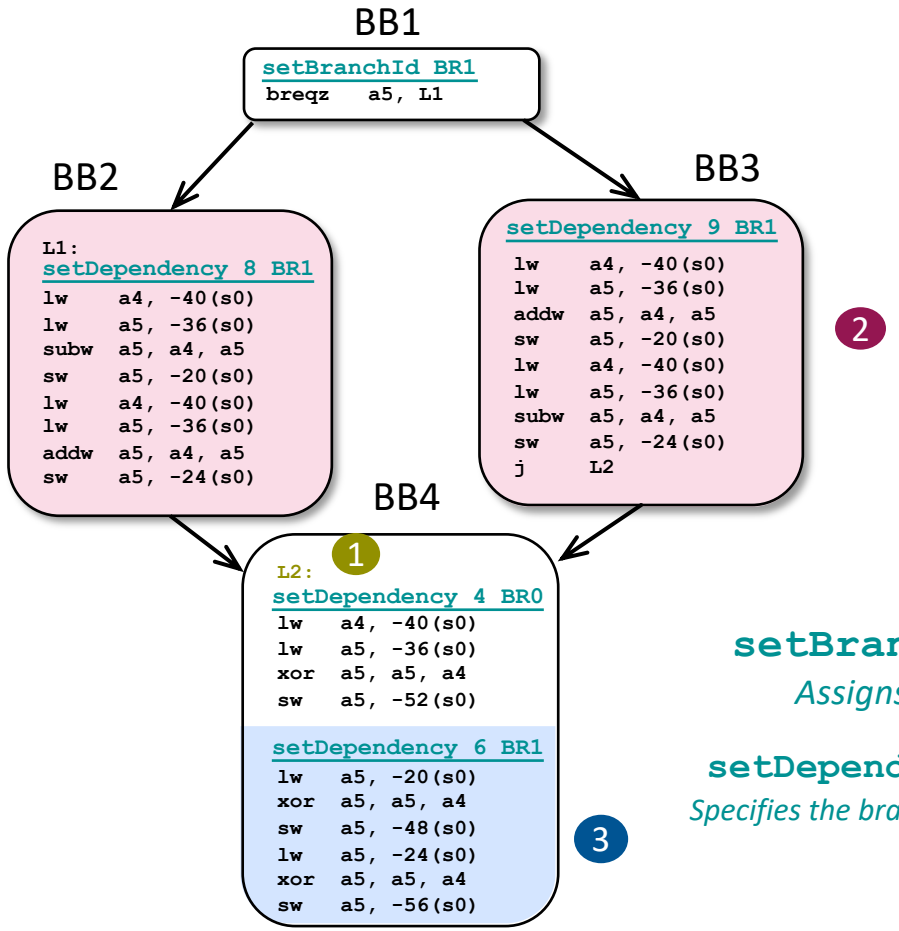
How to implement OoO–commit ***efficiently***?

How to handle ***exceptions*** and context switches?



NOREBA: Static Compiler Analysis

1. Detecting the branch reconvergence point
2. Detecting control dependent instructions
3. Detecting data dependent instructions
4. Marking branch dependent regions



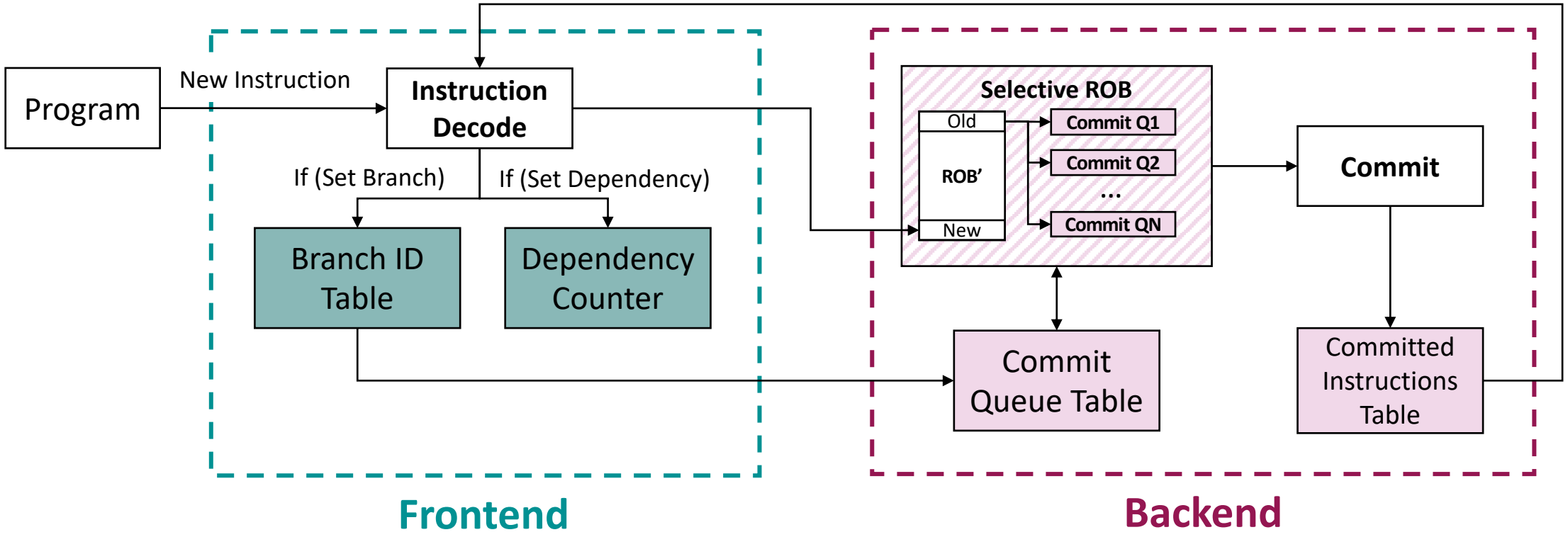
`setBranchId Branch_ID`

Assigns an ID to the branch

`setDependency #NUM Branch_ID`

Specifies the branch dependency of the next #NUM instructions

NOREBA: Microarchitecture

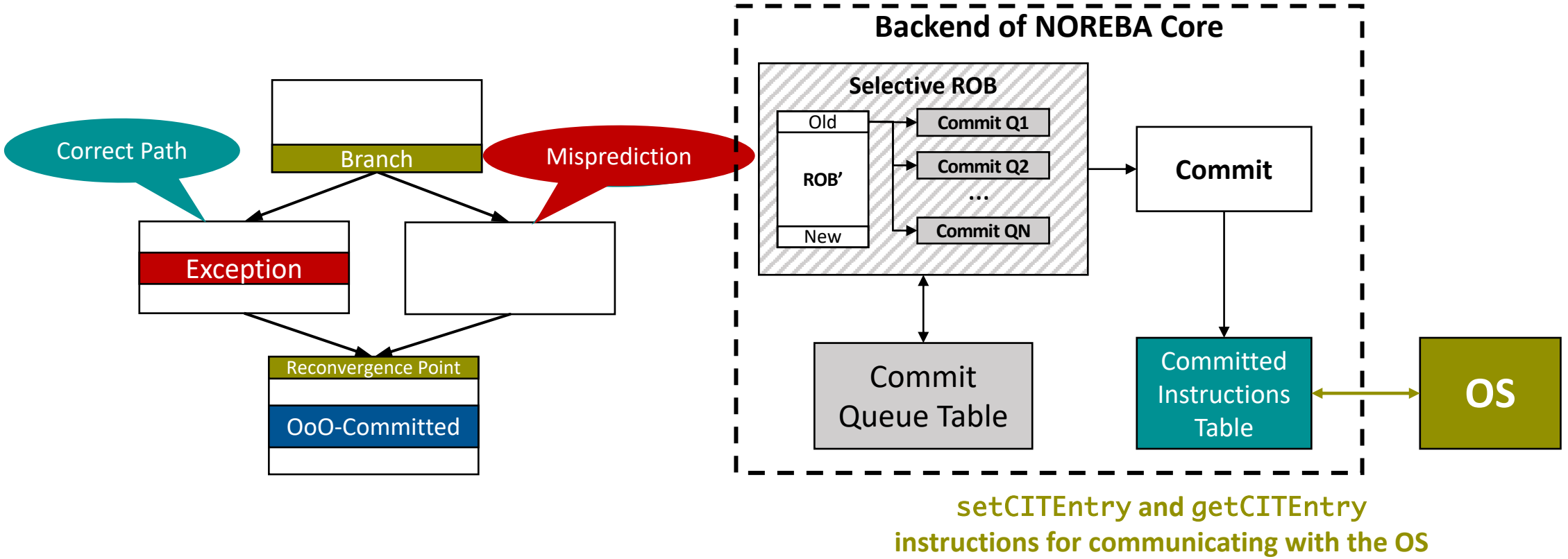


Tracking true branch dependencies informed by the compiler for each instruction

A lightweight implementation for OoO-commit that provides more opportunities to release resources

NOREBA: Challenge in Exception Handling

- Need to save and restore the state of the OoO-committed instructions

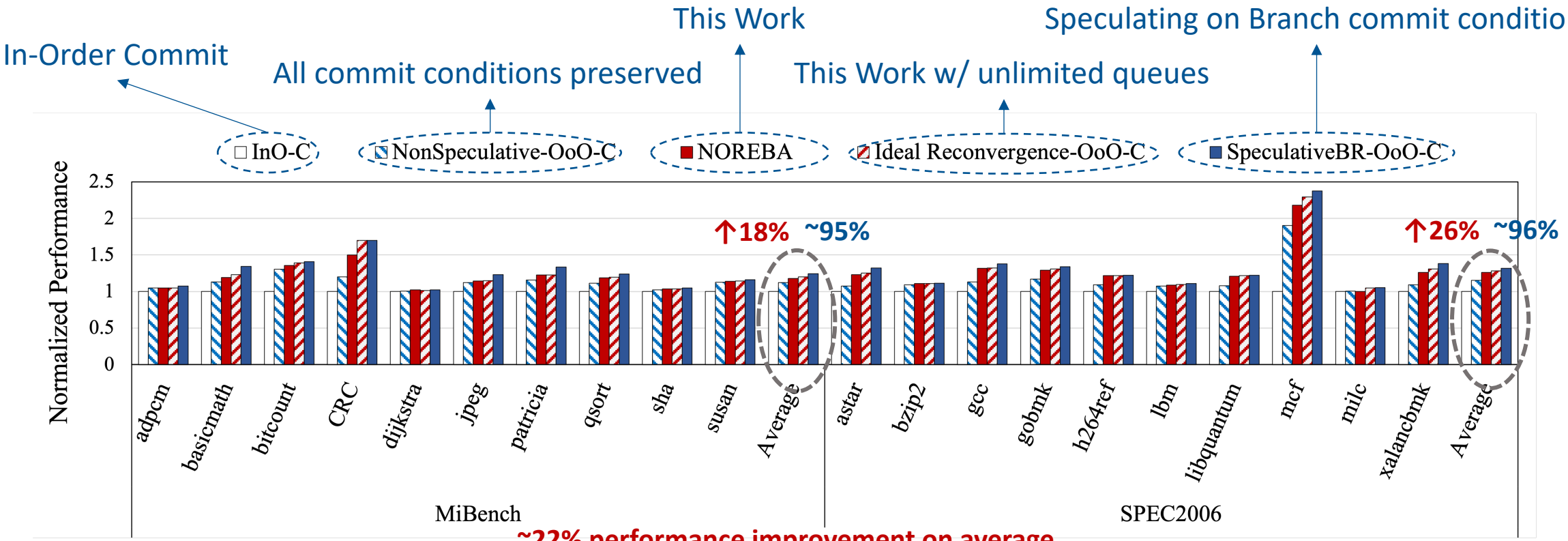


Evaluation: Setup

- Simulation: gem5
- Compiler: LLVM-10
- Benchmarks:
  - *SPEC CPU2006*: C/C++ programs, running single 1B instruction representatives (using SimPoint)
  - *MiBench*: entire program runs

L1d/i size	32*KB*, 4 *clk*
L2 size	256*KB*, 12 *clk*
L3 size	1*MB*, 36 *clk*
Fetch/dispatch/commit width	4/4/4
Branch Predictor	*TAGE-SC-L-8KB*
Prefetcher	*DCPT*
Selective ROB	
ROB'	224 *entries*
BR-CQ	2 × 8 *entries*
PR-CQ	8 *entries*
BIT/CQT size	8
CIT size	128
Baseline ROB	224 *entries*
IQ/LQ/SQ/RF	68/72/56/168

Evaluation: Performance

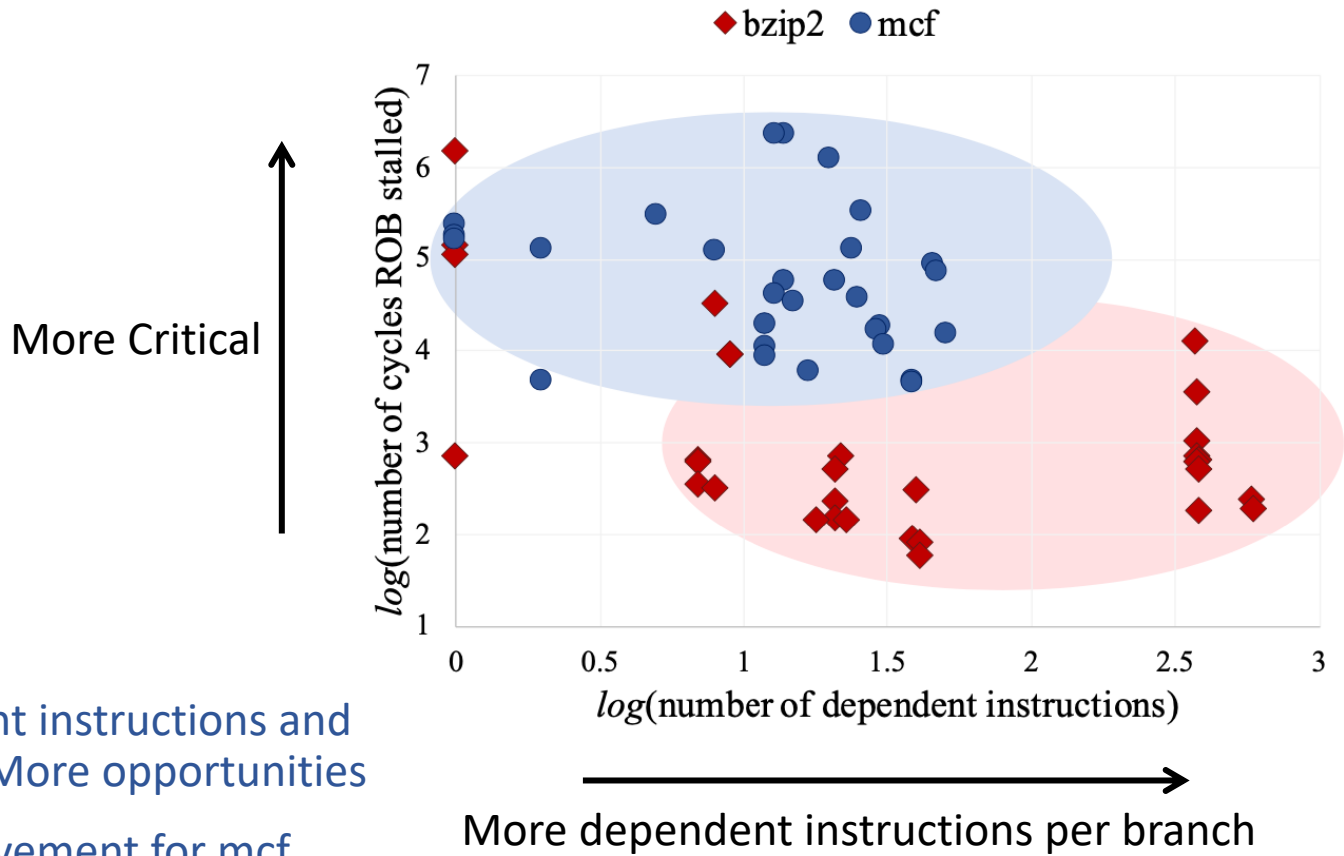


~22% performance improvement on average

Up to 230% performance improvement

Reaches ~95% of a fully branch speculative OoO-commit implementation

Evaluation: Critical Branches



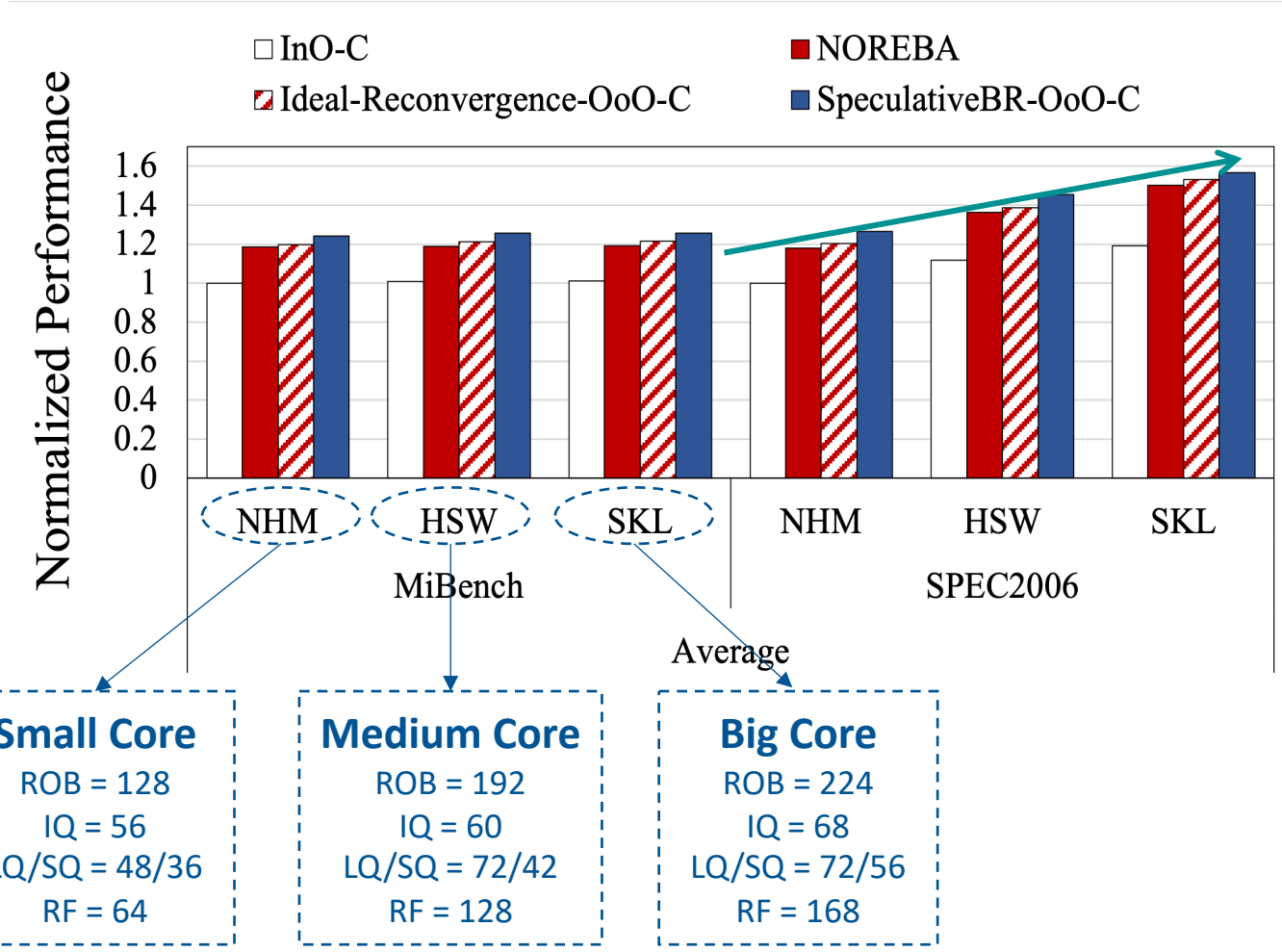
Fewer dependent instructions and more critical ➔ More opportunities

~ 2.3X improvement for mcf

More dependent instructions and less critical ➔ Fewer opportunities

~ 1.1X improvement for bzip2

Evaluation: Size of Resources

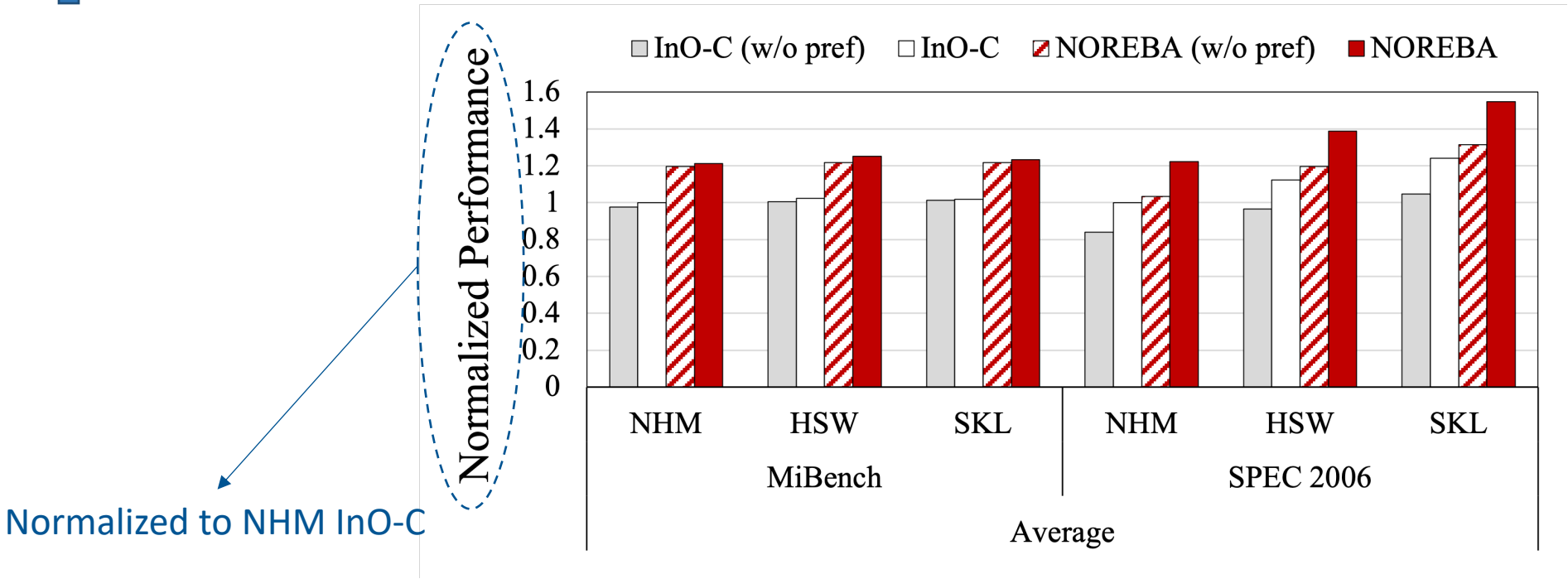


We are close to aggressive and branch speculative OoO-commit (~95% of SpeculativeBR OoO-C)

Higher performance for bigger cores with more resources

NOREBA continues to scale

Evaluation: NOREBA and Prefetchers

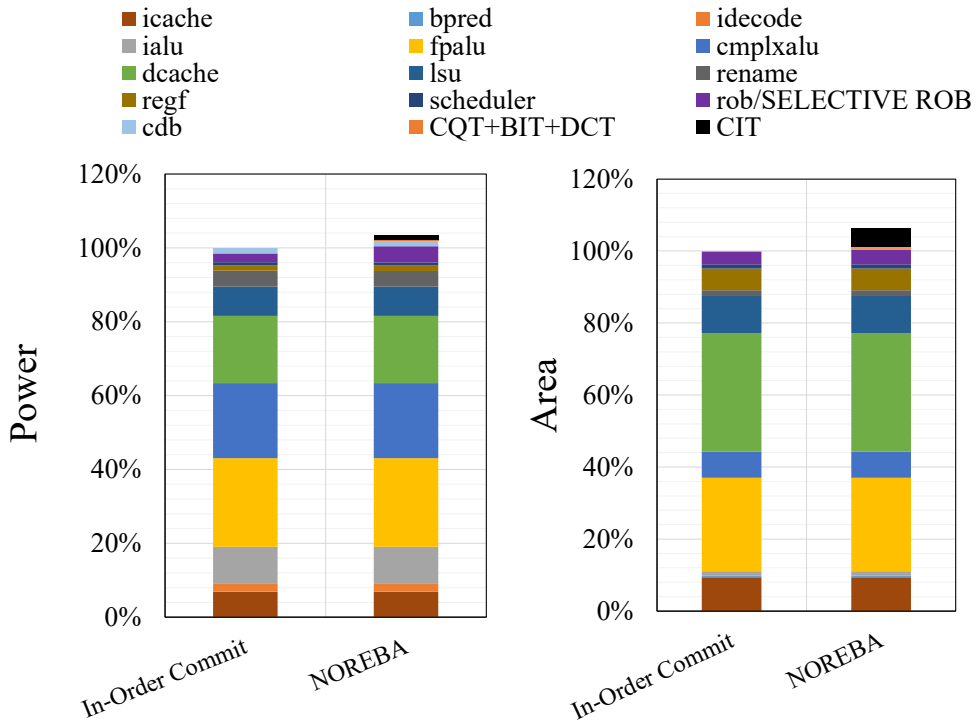


Additive effect of combining NOREBA and prefetchers

➔ Higher Performance using both

Prefetching allows continuing execution, but NOREBA allows continuing committing instructions

Evaluation: Power and Area Overhead

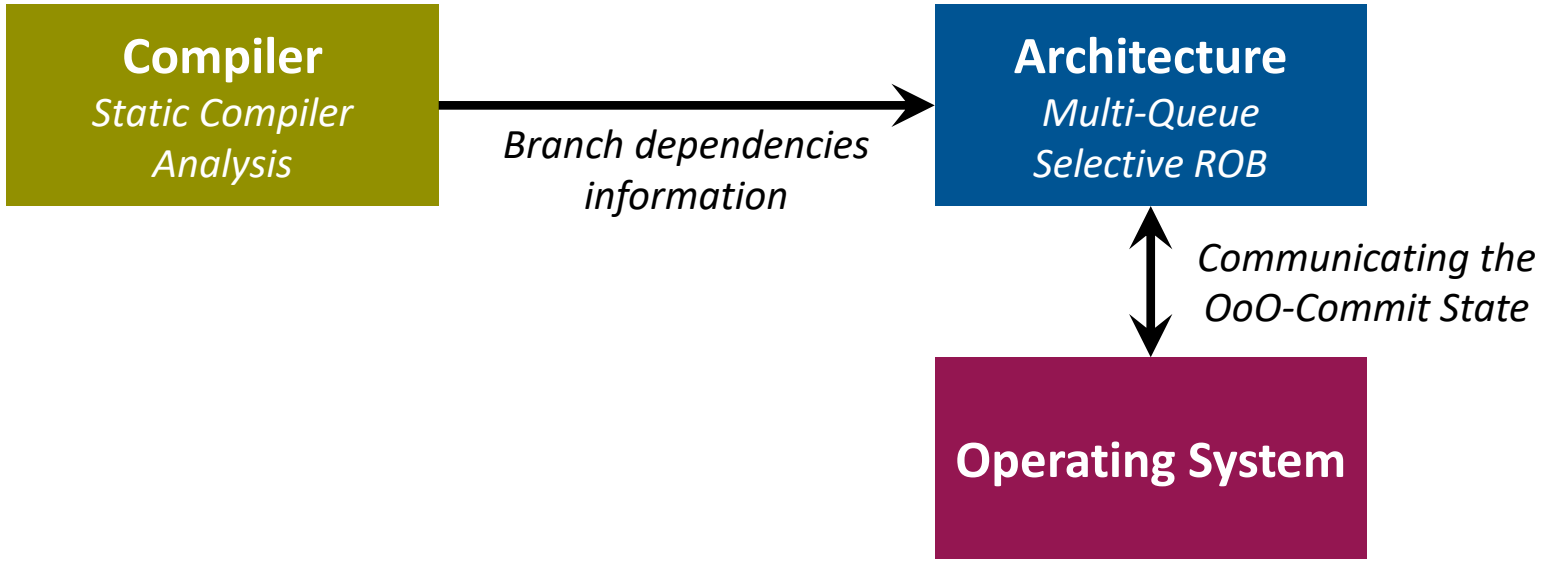


4% power overhead, 8% area overhead

Low overhead for the extra performance (~22% on average, up to 230%)

NOREBA: Overview of the Design

Implementation { Detecting branch dependencies ***non–speculatively***
Implementing OoO–commit ***efficiently***
Able to handle ***exceptions*** and context switches



Conclusion

- Efficient interaction between different layers of the system unlocks efficiency and performance for general-purpose processors

- NOREBA provides a HW/SW co-design solution that enables OoO-commit and better resource management
  - 22% performance improvement over the baseline and achieving 95% of the aggressive branch speculative OoO-commit implementation
  - Low power and area overheads (~4% power, and ~8% area overhead)

NOREBA: A Compiler-Informed Non-speculative Out-of-Order Commit Processor

Thanks for your attention

Ali Hajiabadi[✢], Andreas Diavastos[✦], Trevor E. Carlson[✢]

✢ *National University of Singapore*
✦ *Universitat Politècnica de Catalunya*

ASPLOS '21