Core Models

Po-An Tsai

Massachusetts Institute of Technology

Outline

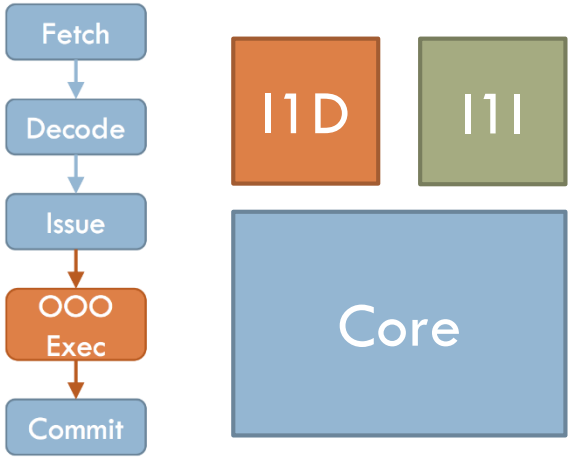
Outline

- ZSim core simulation techniques

Outline

- ZSim core simulation techniques

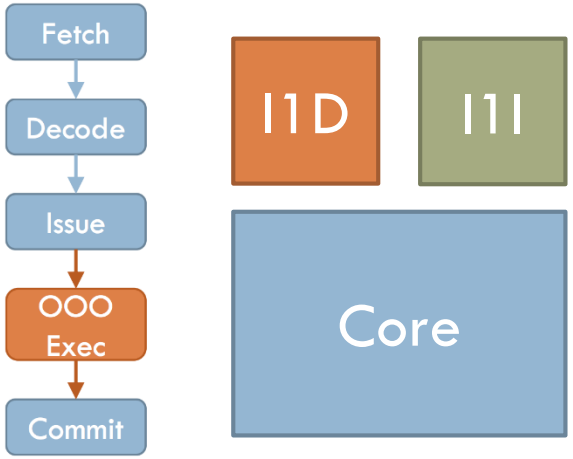
- ZSim core structure
  - Simple IPC 1 core
  - Timing core
  - OOO core



Outline

- ZSim core simulation techniques

- ZSim core structure
  - Simple IPC 1 core
  - Timing core
  - OOO core



- Coding examples with demo
  - Branch predictor
  - Westmere to Silvermont

```
class GShareBranchPredictor {
    private:
        bool lastSeen;
    public:
        GShareBranchPredictor(){ lastSeen = false;}
        inline bool predict(Address branchPc, bool taken){
            bool prediction = (taken == lastSeen);
            lastSeen = taken;
            return prediction;
        }
};
```

Core Simulation Techniques

Core Simulation Techniques

- ZSim simulates the system using Pin
  - Leverages dynamic binary translation

Core Simulation Techniques

- ZSim simulates the system using Pin
  - Leverages dynamic binary translation

- ZSim mainly uses 4 types of analysis routine
  - Basic block
  - Load and Store
  - Branch

  to cover the simulated program

Core Simulation Technique

Core Simulation Technique

- A basic block (BBL) from Pin

Core Simulation Technique

- A basic block (BBL) from Pin

```
mov (%rbp),%rcx
add %rax,%rbx
mov %rdx,(%rbp)
ja 40530a
```

Core Simulation Technique

- A basic block (BBL) from Pin

```
mov (%rbp),%rcx
add %rax,%rbx
mov %rdx,(%rbp)
ja 40530a
```

Core Simulation Technique

- A basic block (BBL) from Pin

```
mov (%rbp),%rcx
add %rax,%rbx
mov %rdx,(%rbp)
ja 40530a
```

- 1. Simulate core activities with a BBL descriptor that contains most of the static information

Core Simulation Technique

- A basic block (BBL) from Pin

```
mov (%rbp),%rcx
add %rax,%rbx
mov %rdx,(%rbp)
ja 40530a
```

- 1. Simulate core activities with a BBL descriptor that contains most of the static information

```
mov (%rbp),%rcx
add %rax,%rbx
mov %rdx,(%rbp)

ja 40530a
```

Core Simulation Technique

- A basic block (BBL) from Pin

```
mov (%rbp),%rcx
add %rax,%rbx
mov %rdx,(%rbp)
ja 40530a
```

- 1. Simulate core activities with a BBL descriptor that contains most of the static information

```
mov (%rbp),%rcx
add %rax,%rbx
mov %rdx,(%rbp)

ja 40530a
```

Decode BBL into BBL descriptor

Core Simulation Technique

- A basic block (BBL) from Pin

```
mov (%rbp),%rcx
add %rax,%rbx
mov %rdx,(%rbp)
ja 40530a
```

- 1. Simulate core activities with a BBL descriptor that contains most of the static information

```
mov (%rbp),%rcx
add %rax,%rbx
mov %rdx,(%rbp)

ja 40530a
```

Decode BBL into BBL descriptor

BblDescriptor:

numInstructions = 4

numBytes = 4

uop[]

Core Simulation Technique

- A basic block (BBL) from Pin

```
mov (%rbp),%rcx
add %rax,%rbx
mov %rdx,(%rbp)
ja 40530a
```

- 1. Simulate core activities with a BBL descriptor that contains most of the static information

```
mov (%rbp),%rcx
add %rax,%rbx
mov %rdx,(%rbp)
BasicBlock(BblDescriptor)
ja 40530a
```

Decode BBL into BBL descriptor

BblDescriptor:
numInstructions = 4
numBytes = 4
uop[]

Core Simulation Technique

- Decode x86 instructions into uops
  - With different latencies, src/dst pair, function unit ports

Type	Src1	Src2	Dst1	Dst2	Lat	PortMsk
Load	rbp		rcx			001000
Exec	rbp		rdx		3	110001
Exec	rax	rdx	rdx	rflgs	1	110001
StAddr	rbp		S0		1	000100
StData	rdx	S0				000010
Exec	rax	rip	rip	rflgs	1	000001

Core Simulation Technique

Core Simulation Technique

- 2. Simulate memory system operations with addresses

Core Simulation Technique

- 2. Simulate memory system operations with addresses

```
mov (%rbp),%rcx

add %rax,%rbx
mov %rdx,(%rbp)

BasicBlock(BblDescriptor)
ja 40530a
```

Core Simulation Technique

- 2. Simulate memory system operations with addresses

```
mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)
Store(%rbp)
BasicBlock(BblDescriptor)
ja 40530a
```

Core Simulation Technique

- 2. Simulate memory system operations with addresses

```
mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)
Store(%rbp)
BasicBlock(BblDescriptor)
ja 40530a
```

```
Load(Address addr) {
        L1D->load(addr);
}

Store(Address addr) {
        L1D->Store(addr);
}
```

Core Simulation Technique

Core Simulation Technique

- Instruction-driven core activity (basic block) simulation
  - Simulates multiple stages for single instruction at once
  - Each stage maintains a separate clock

Core Simulation Technique

- Instruction-driven core activity (basic block) simulation
  - Simulates multiple stages for single instruction at once
  - Each stage maintains a separate clock

```
BasicBlock(BblDescriptor) {
        foreach uop {
                simulateFetch(uop);
                simulateDecode(uop);
                simulateIssue(uop);
                simulateExecute(uop);
                simulateCommit(uop);
        }
}
```

Core Simulation Technique

- Instruction-driven core activity (basic block) simulation
  - Simulates multiple stages for single instruction at once
  - Each stage maintains a separate clock

```
BasicBlock(BblDescriptor) {
        foreach uop {
                simulateFetch(uop);
                simulateDecode(uop);
                simulateIssue(uop);
                simulateExecute(uop);
                simulateCommit(uop);
        }
}
```

```
simulateIssue(uop) {
    addUopToRob(curRobCycle, uop);
    if(rob.isFull()){
        nextRobAvailCycle = rob.advance();
    }
}
```

Core Simulation Technique

- Event-driven uncore activity simulation

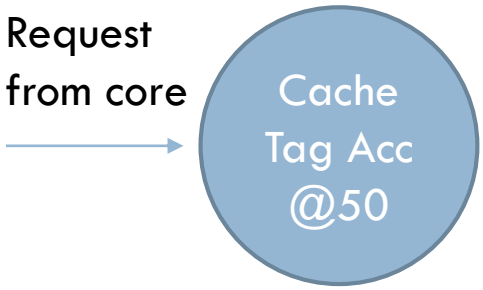
Core Simulation Technique

- Event-driven uncore activity simulation

Request
from core

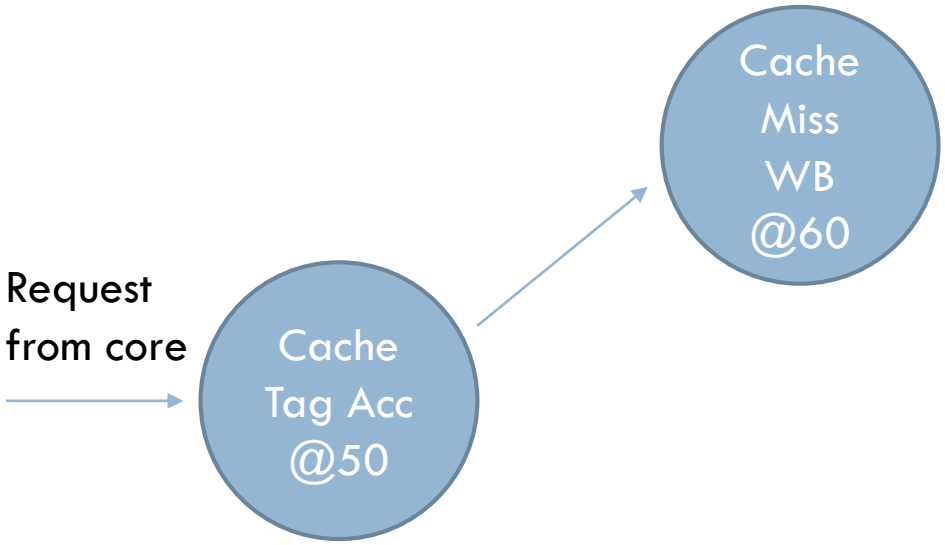
Core Simulation Technique

- Event-driven uncore activity simulation



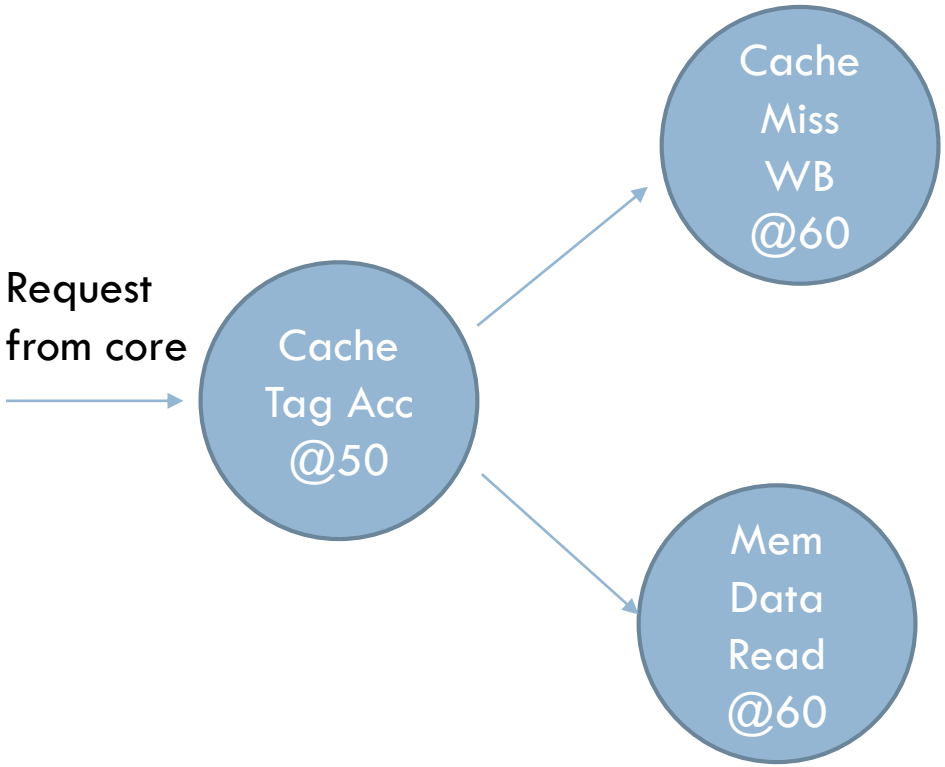
Core Simulation Technique

- Event-driven uncore activity simulation



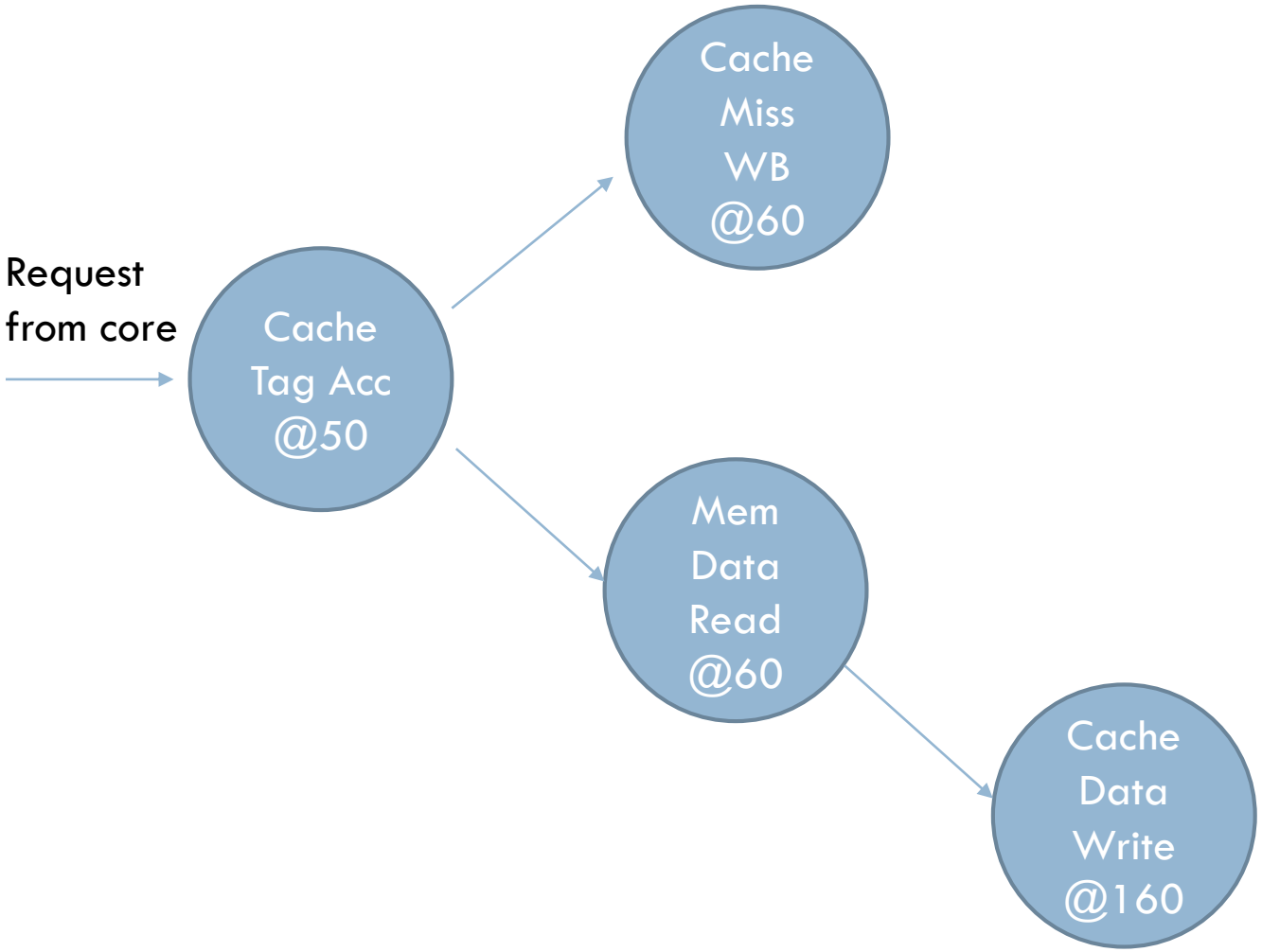
Core Simulation Technique

- Event-driven uncore activity simulation



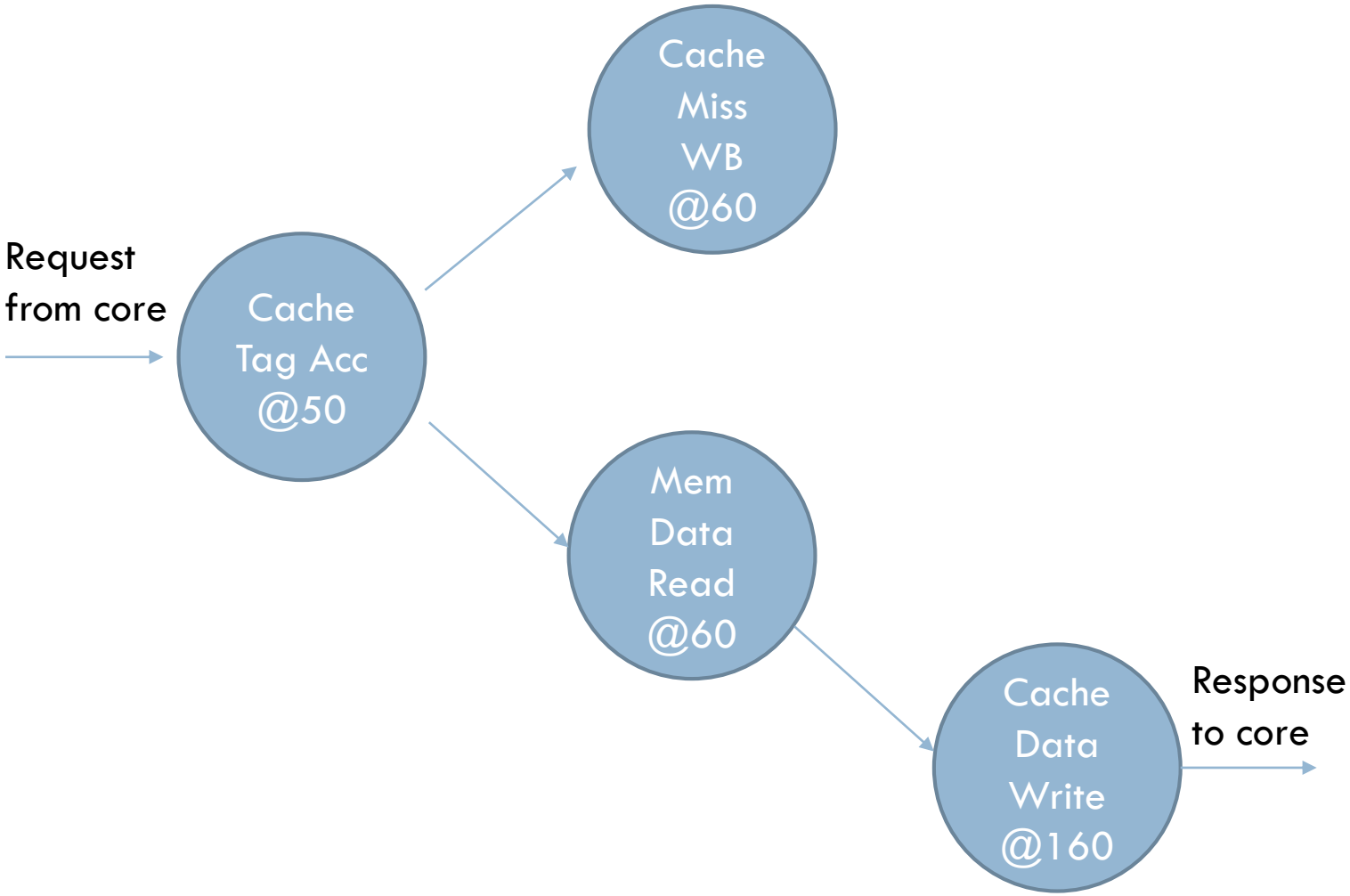
Core Simulation Technique

- Event-driven uncore activity simulation



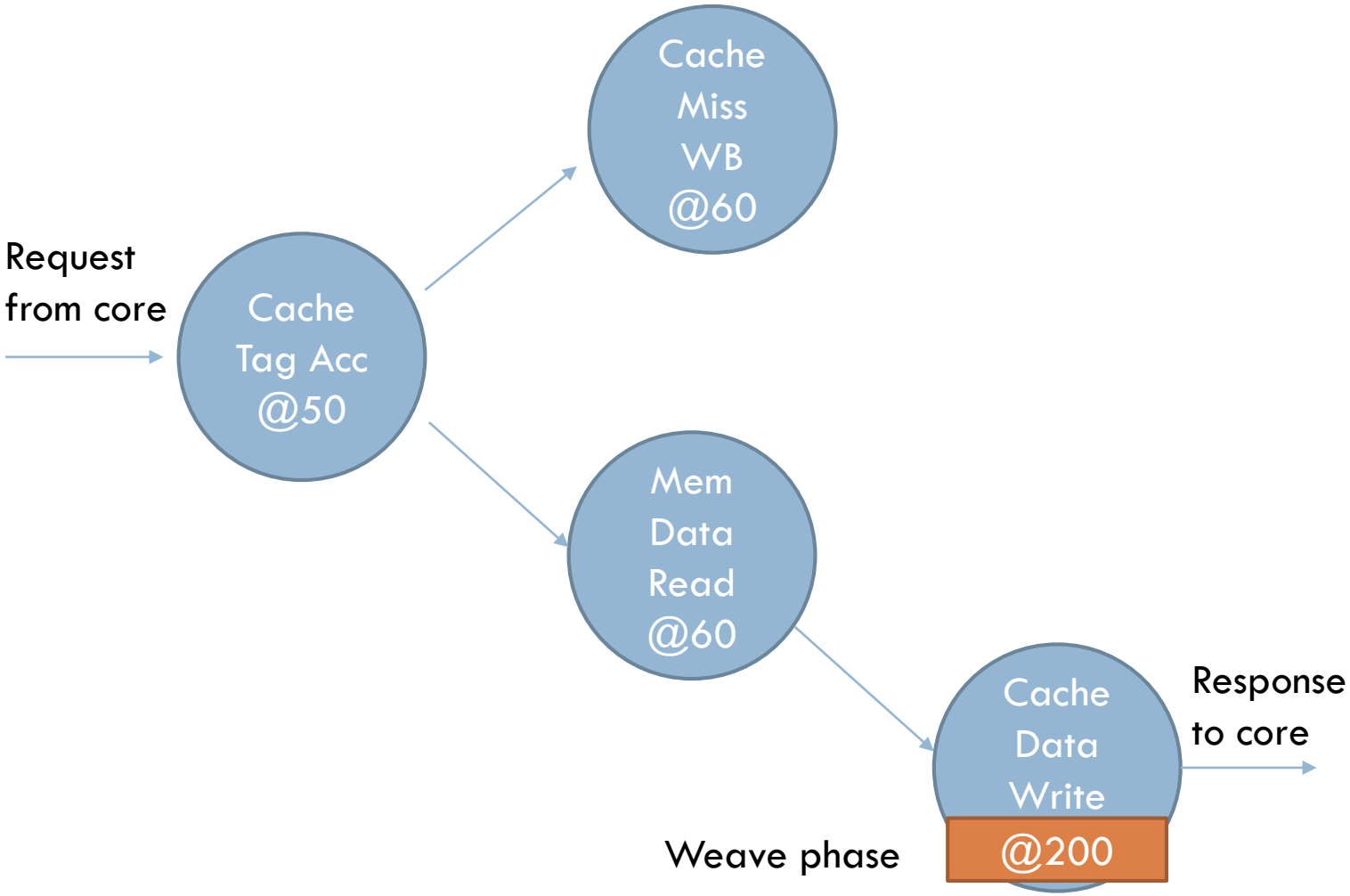
Core Simulation Technique

- Event-driven uncore activity simulation



Core Simulation Technique

- Event-driven uncore activity simulation



General Core Structure

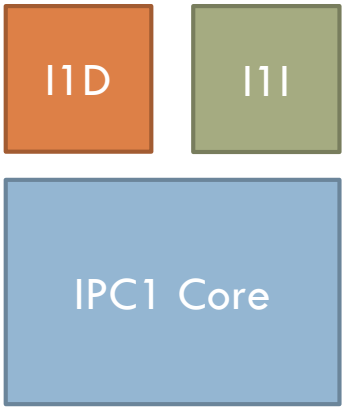
General Core Structure

- ZSim simulates a core with 4 functions using Pin's APIs
  - BblFunc
  - LoadFunc
  - StoreFunc
  - BranchFunc

General Core Structure

- ZSim simulates a core with 4 functions using Pin's APIs
  - BblFunc
  - LoadFunc
  - StoreFunc
  - BranchFunc
- Current supported core type
  - Simple IPC1 core
  - Timing core
  - OOO core (Westmere-like)

Simple IPC1 Core



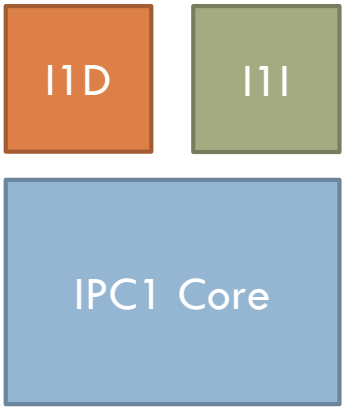
Current cycle = 0

mov (%rbp),%rcx

add %rax,%rbx
mov %rdx,(%rbp)

ja 40530a

Simple IPC1 Core

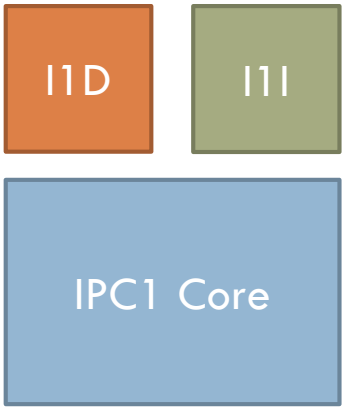


Current cycle = 0

mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)

ja 40530a

Simple IPC1 Core



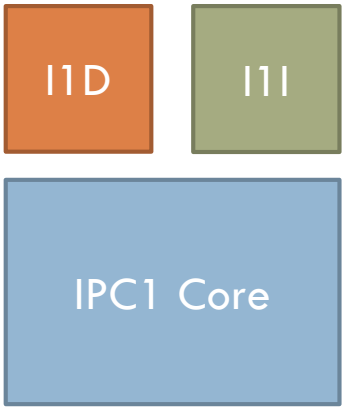
Current cycle = 0

mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)

ja 40530a

Current cycle = I1d->load(curCycle)

Simple IPC1 Core



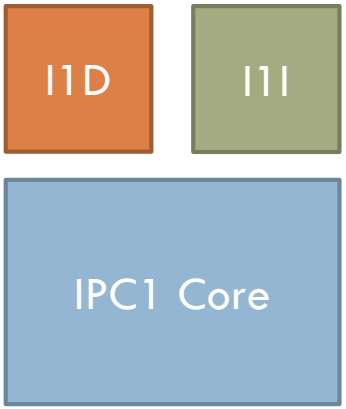
Current cycle = 0

mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)
Store(%rbp)

ja 40530a

Current cycle = I1d->load(curCycle)

Simple IPC1 Core



Current cycle = 0

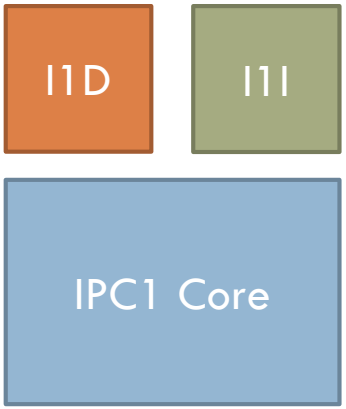
mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)
Store(%rbp)

ja 40530a

Current cycle = I1d->load(curCycle)

Current cycle = I1d->store(curCycle)

Simple IPC1 Core



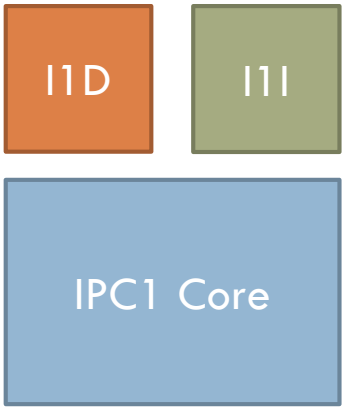
Current cycle = 0

mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)
Store(%rbp)
BasicBlock(BblDescriptor)
ja 40530a

Current cycle = I1d->load(curCycle)

Current cycle = I1d->store(curCycle)

Simple IPC1 Core



Current cycle = 0

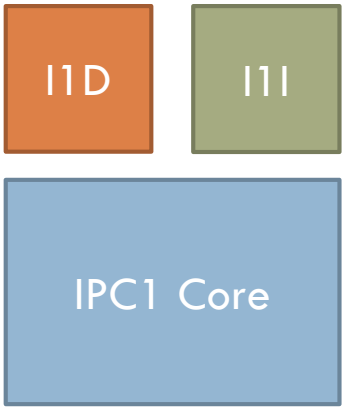
```
mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)
Store(%rbp)
BasicBlock(BblDescriptor)
ja 40530a
```

Current cycle = I1d->load(curCycle)

Current cycle = I1d->store(curCycle)

Current cycle += 4

Timing Core



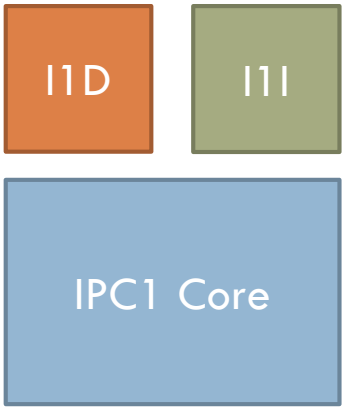
Current cycle = 0

mov (%rbp),%rcx

add %rax,%rbx
mov %rdx,(%rbp)

ja 40530a

Timing Core

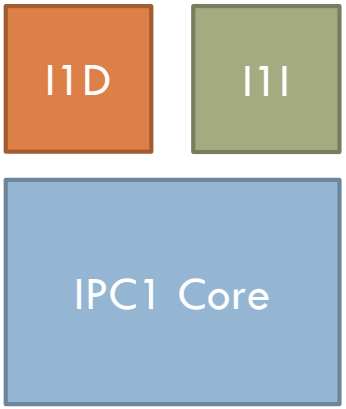


Current cycle = 0

mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)

ja 40530a

Timing Core



Current cycle = 0

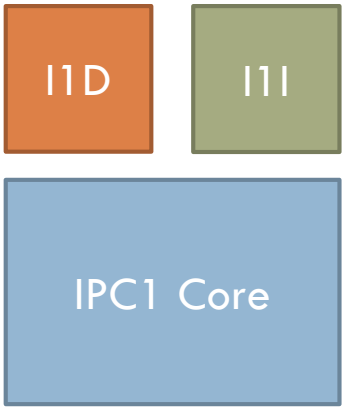
mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)

ja 40530a

Current cycle = I1d->load(curCycle)

Timing Core

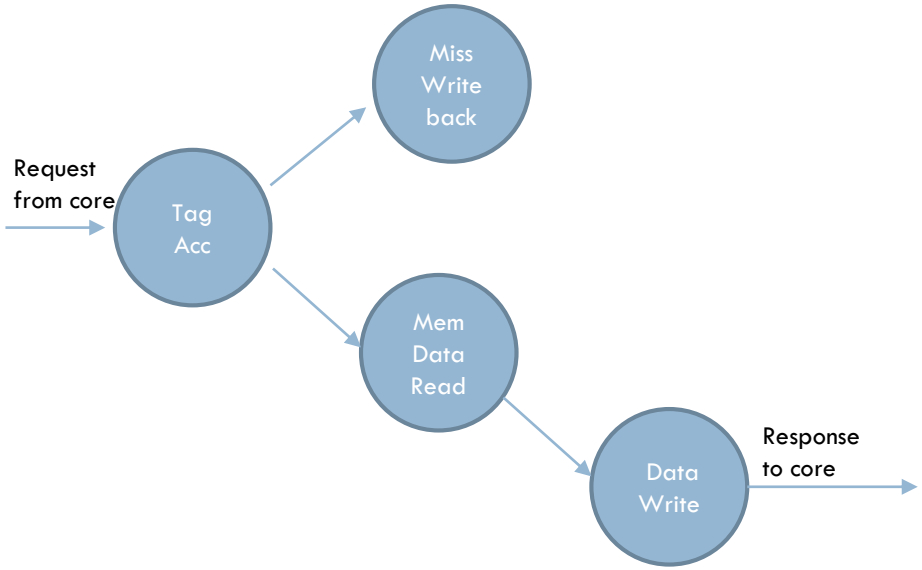
Current cycle = 0



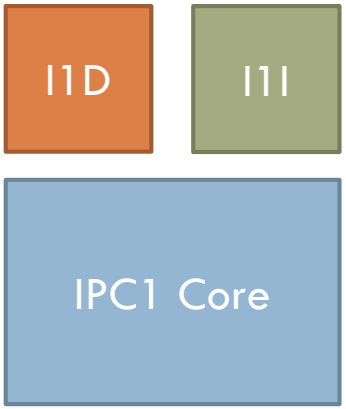
mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)

ja 40530a

Current cycle = I1d->load(curCycle)



Timing Core

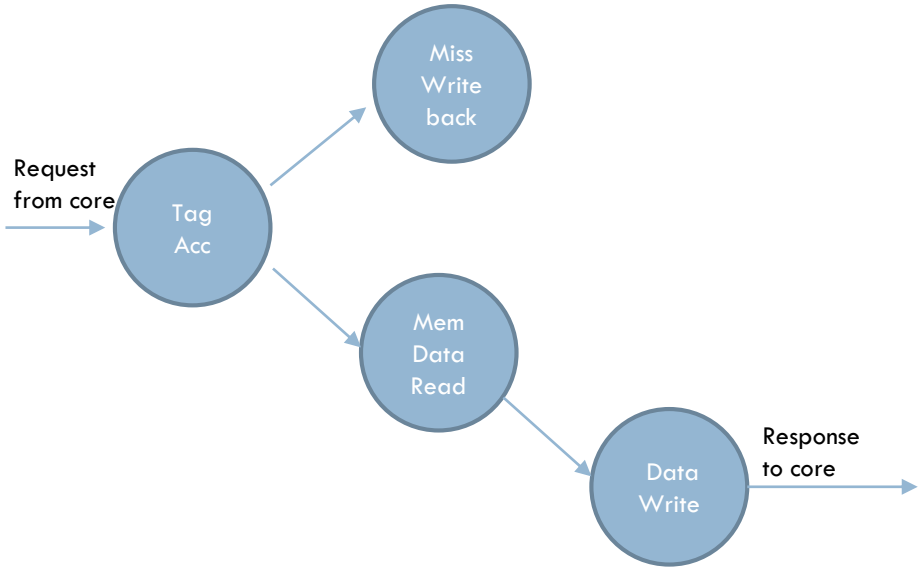


Current cycle = 0

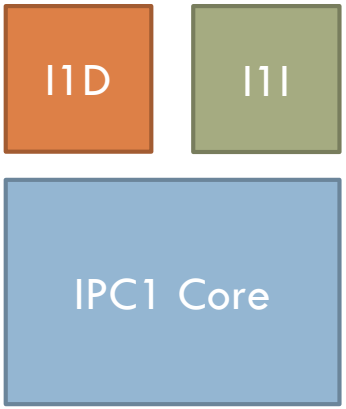
mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)
Store(%rbp)

ja 40530a

Current cycle = I1d->load(curCycle)



Timing Core



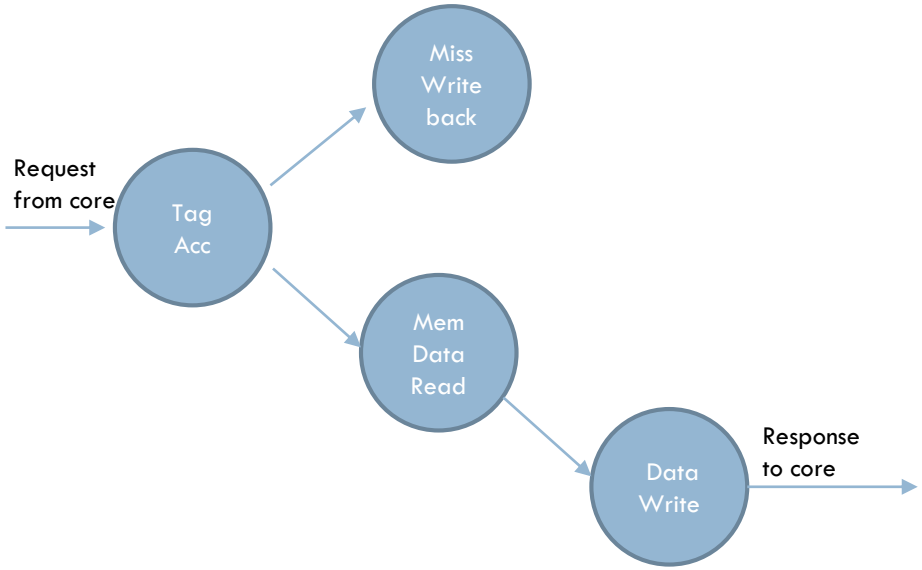
Current cycle = 0

mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)
Store(%rbp)

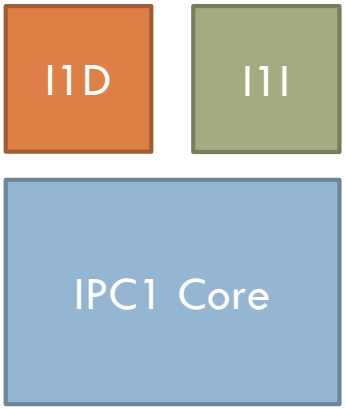
ja 40530a

Current cycle = l1d->load(curCycle)

Current cycle = l1d->store(curCycle)



Timing Core



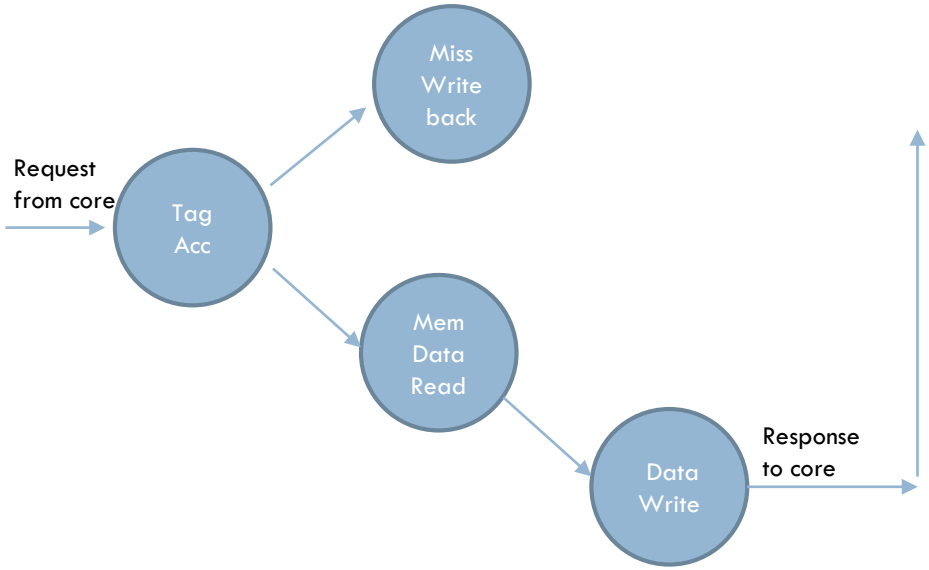
Current cycle = 0

mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)
Store(%rbp)

ja 40530a

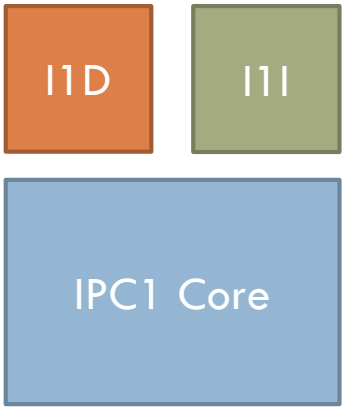
Current cycle = l1d->load(curCycle)

Current cycle = l1d->store(curCycle)



Timing Core

Current cycle = 0

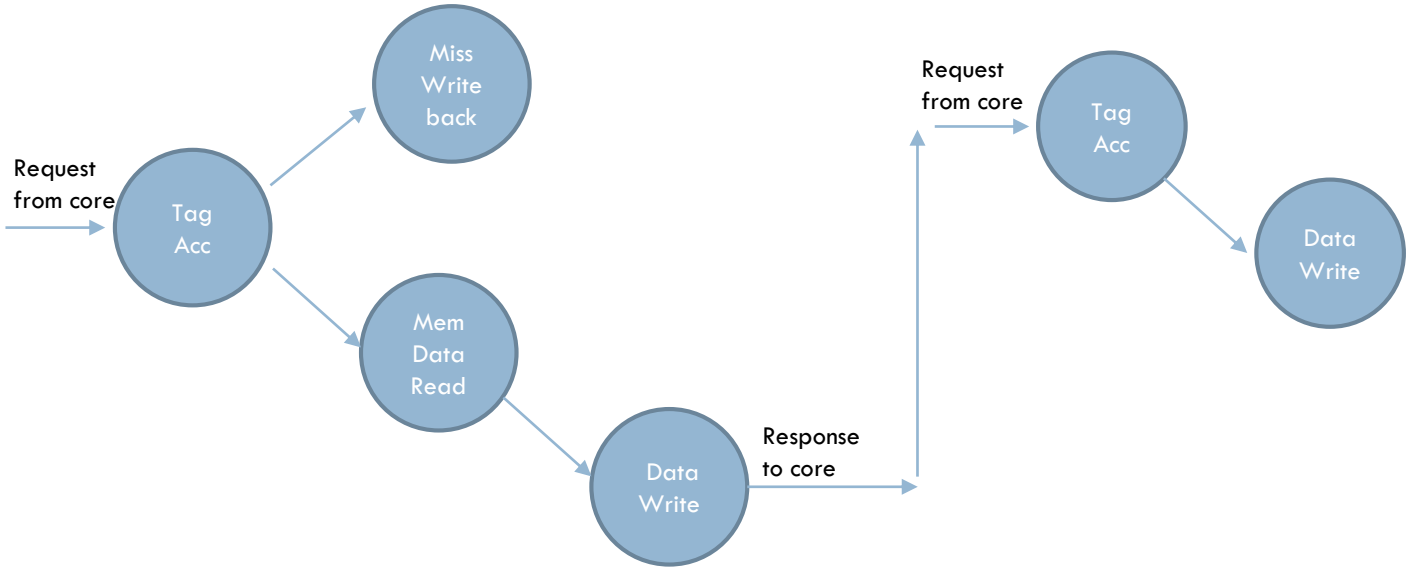


mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)
Store(%rbp)

ja 40530a

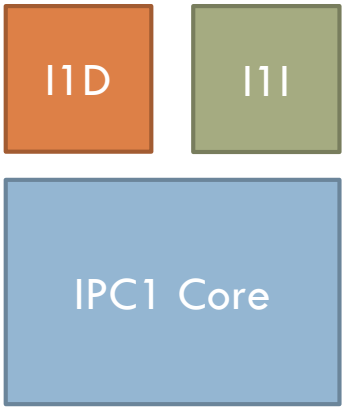
Current cycle = I1d->load(curCycle)

Current cycle = I1d->store(curCycle)



Timing Core

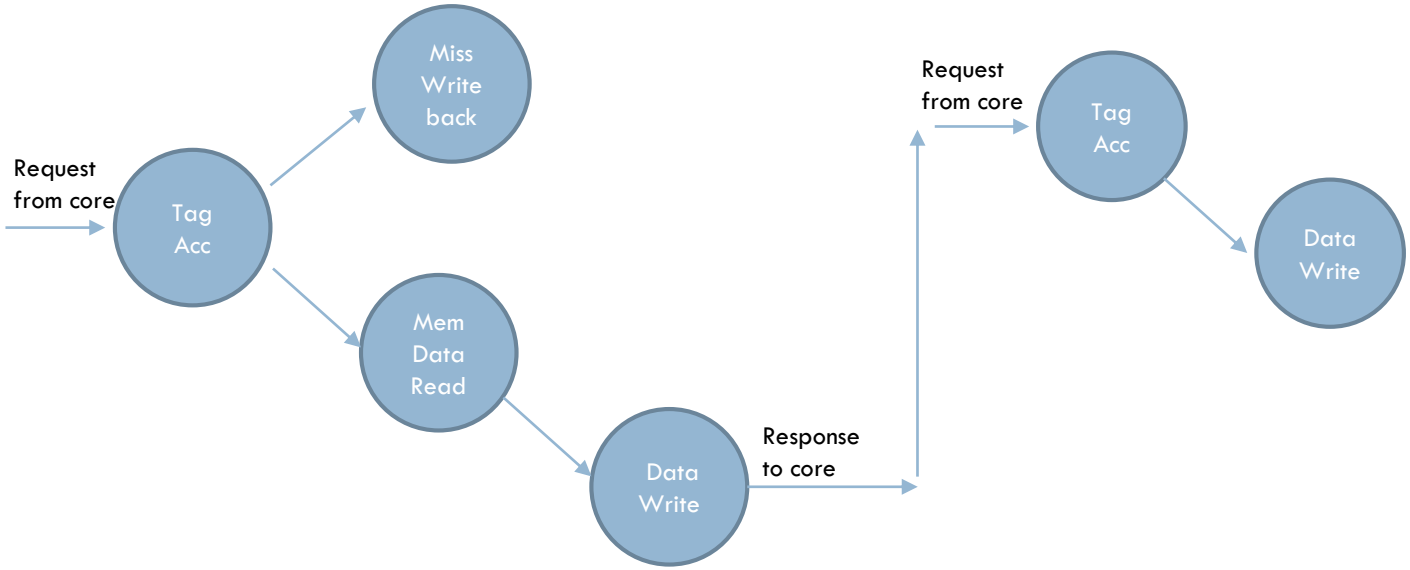
Current cycle = 0



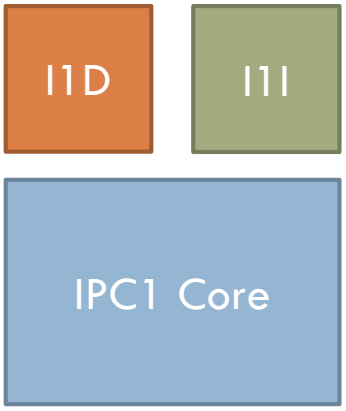
mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)
Store(%rbp)
BasicBlock(BblDescriptor)
ja 40530a

Current cycle = I1d->load(curCycle)

Current cycle = I1d->store(curCycle)



Timing Core



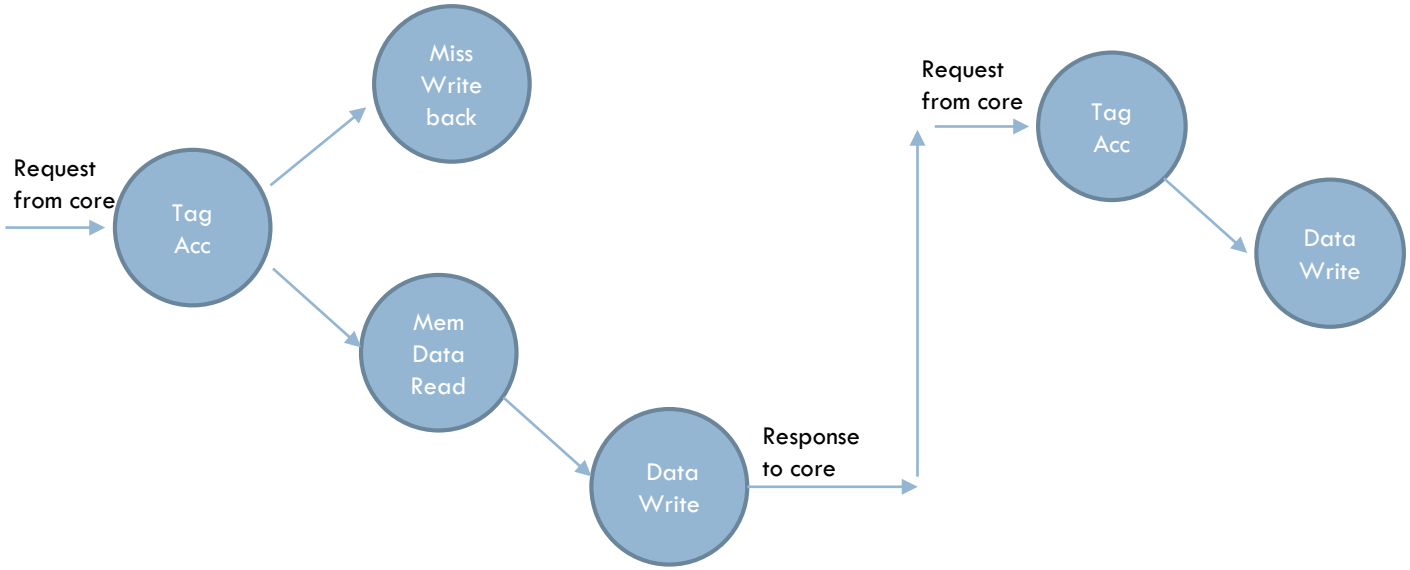
Current cycle = 0

mov (%rbp),%rcx
Load(%rbp)
add %rax,%rbx
mov %rdx,(%rbp)
Store(%rbp)
BasicBlock(BblDescriptor)
ja 40530a

Current cycle = I1d->load(curCycle)

Current cycle = I1d->store(curCycle)

Current cycle += 4



OOO Core - BBL

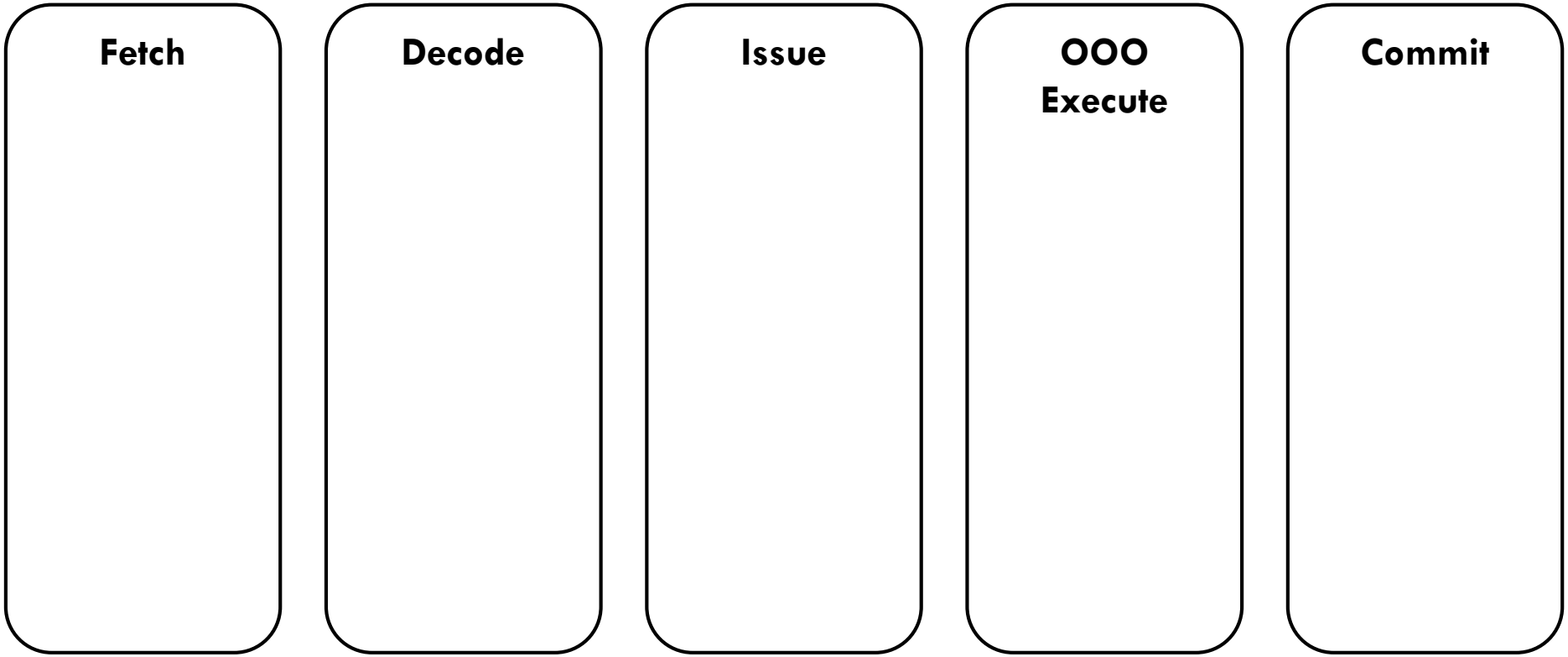
- Simulate all stages at once

  Load A
  Exec
  Store A
  Exec

OOO Core - BBL

- Simulate all stages at once

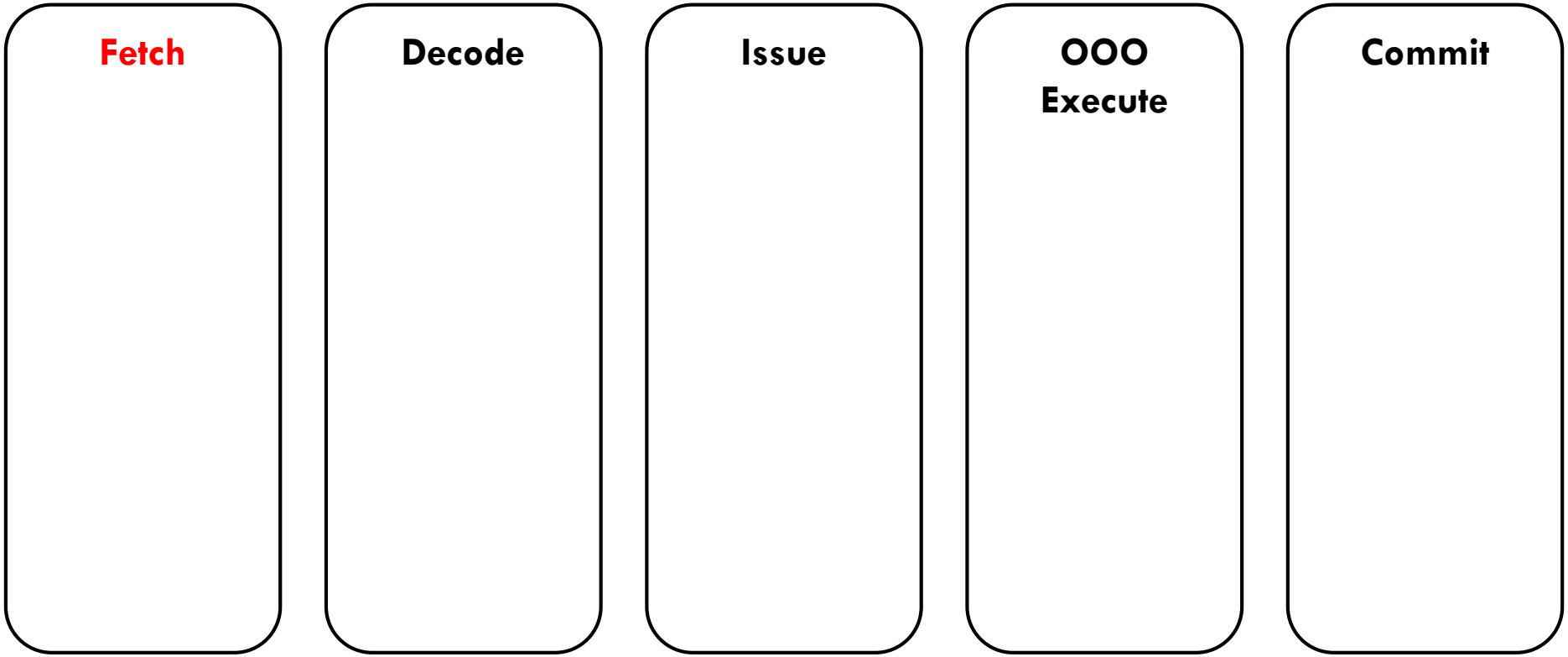
Load A
Exec
Store A
Exec



OOO Core - BBL

- Simulate all stages at once

Load A
Exec
Store A
Exec

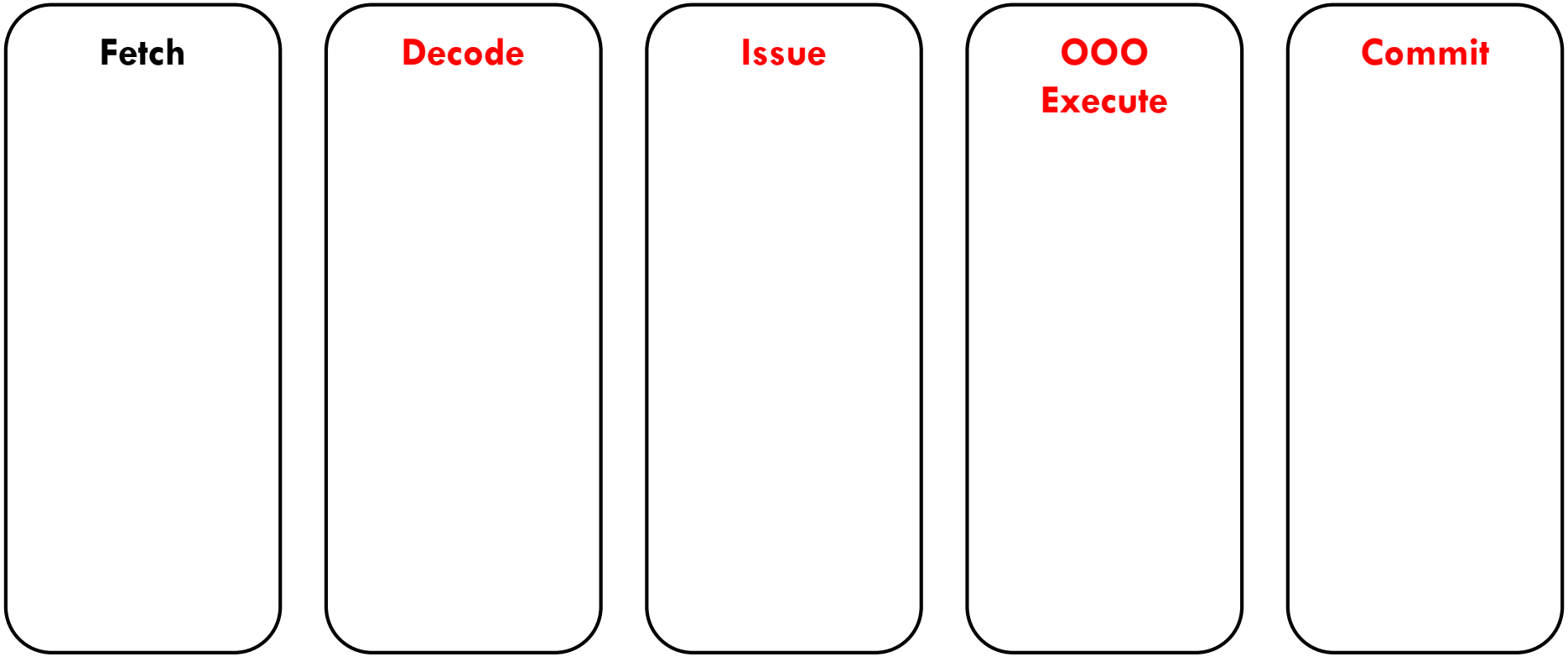


OOO Core - BBL

- Simulate all stages at once

Load A
Exec

Store A
Exec



OOO Core - BBL

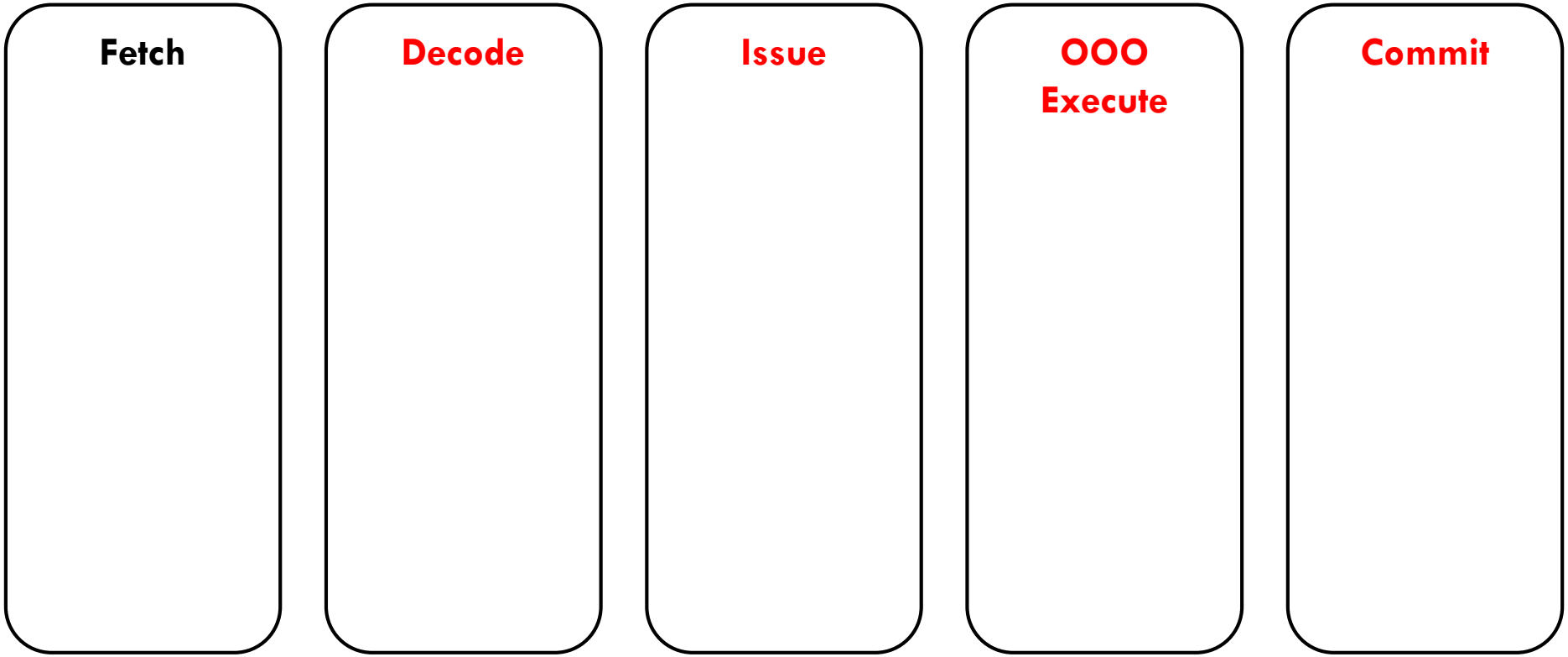
- Simulate all stages at once

Load A

Exec

Store A

Exec



OOO Core - BBL

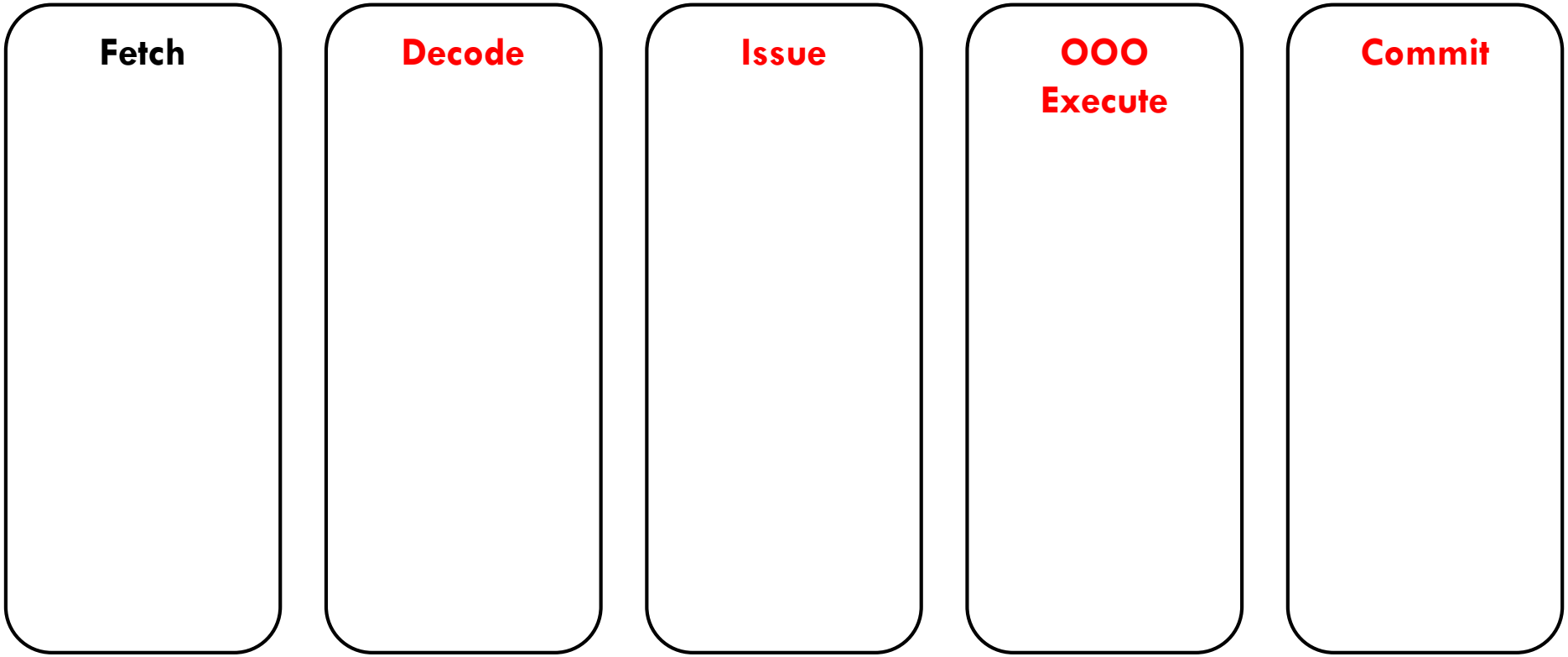
- Simulate all stages at once

Load A

Exec

Store A

Exec



OOO Core - BBL

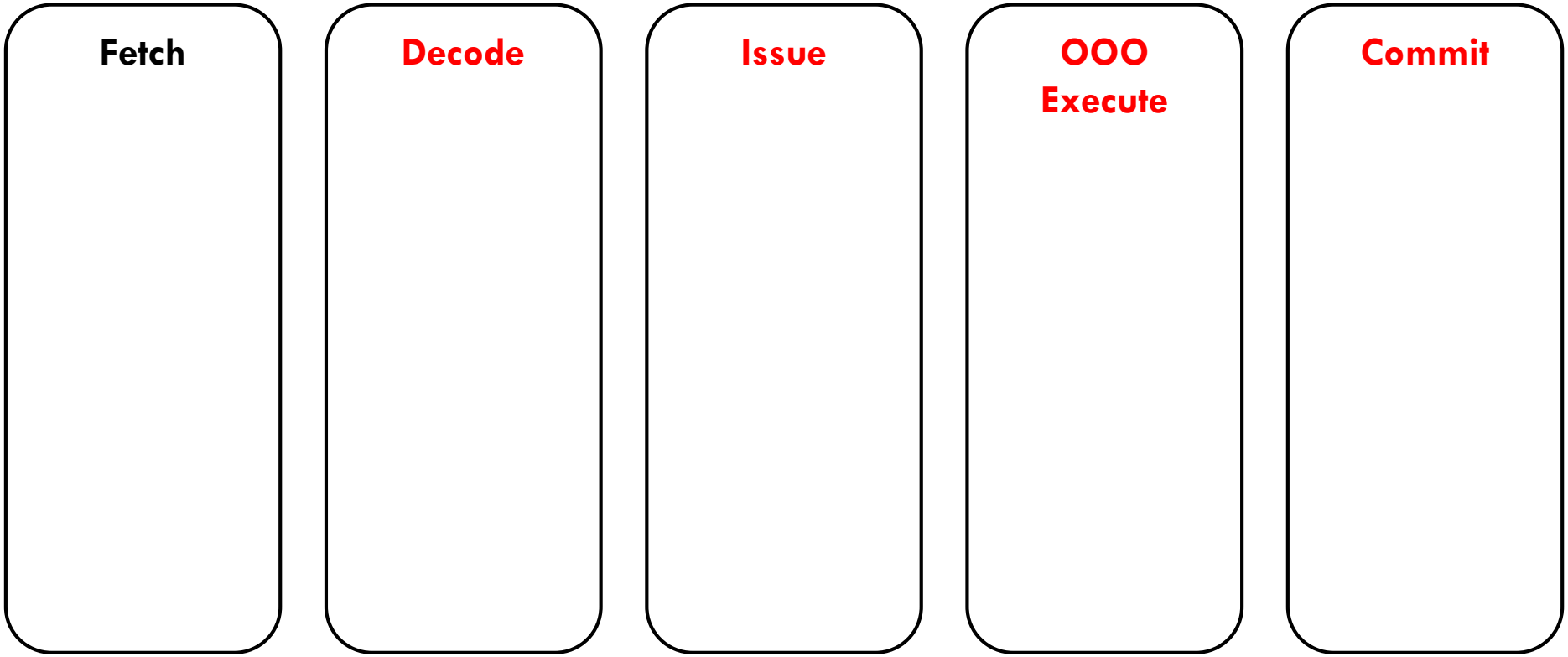
- Simulate all stages at once

Load A

Exec

Store A

Exec



OOO Core - BBL

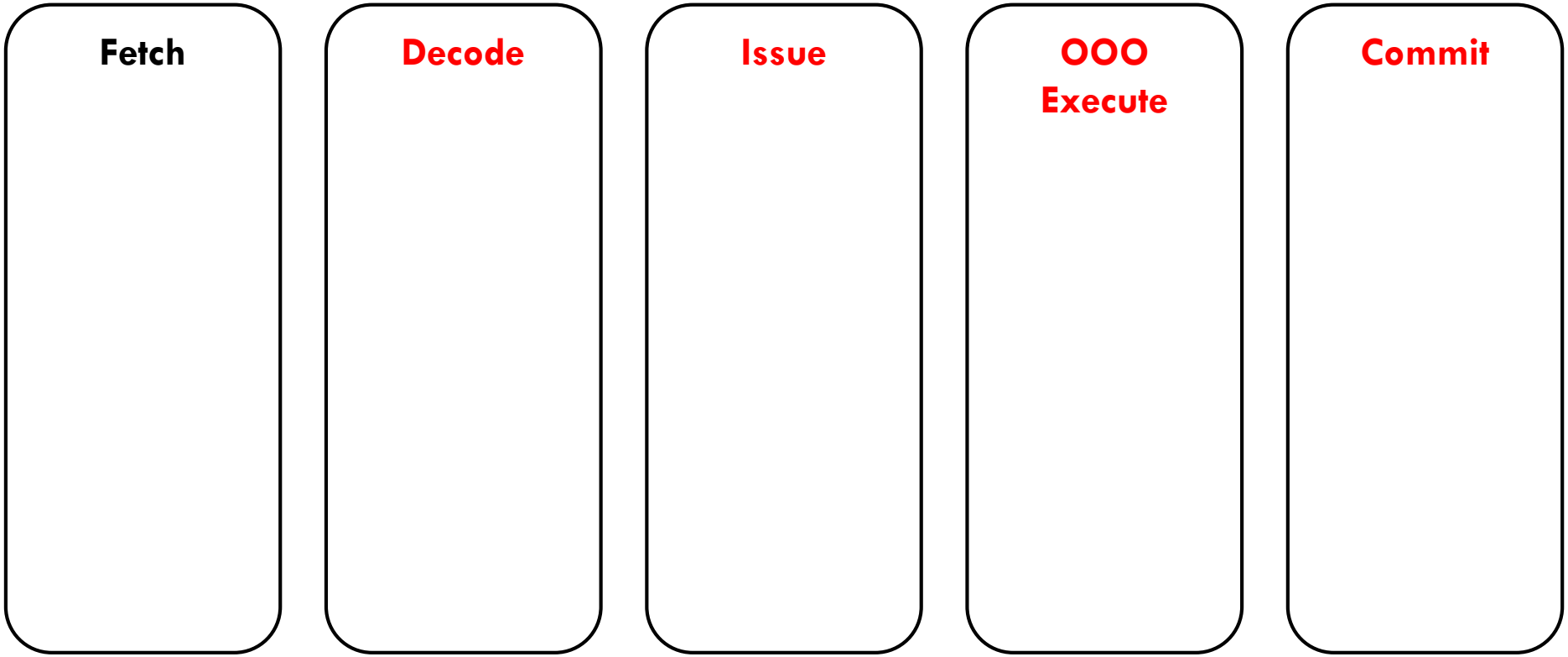
- Simulate all stages at once

Load A

Exec

Store A

Exec



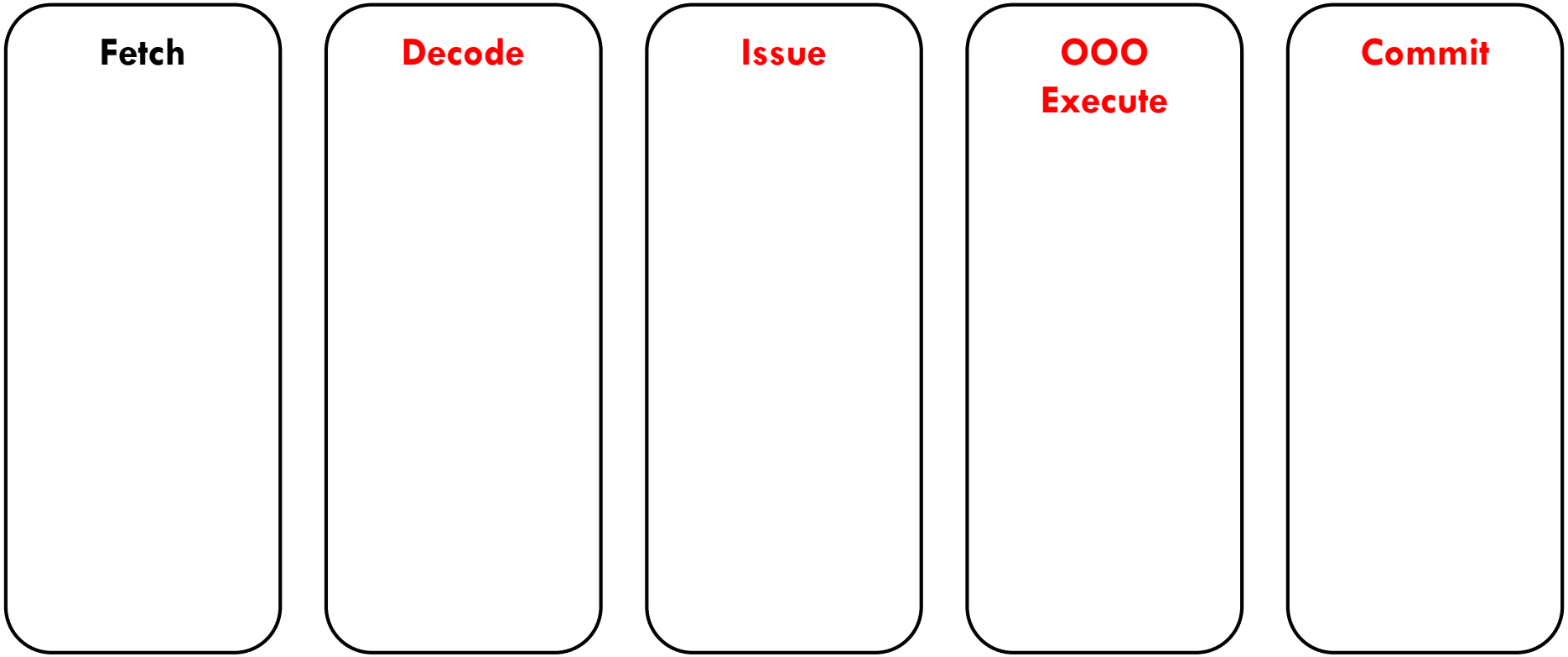
OOO Core - BBL

- Simulate all stages at once

Exec

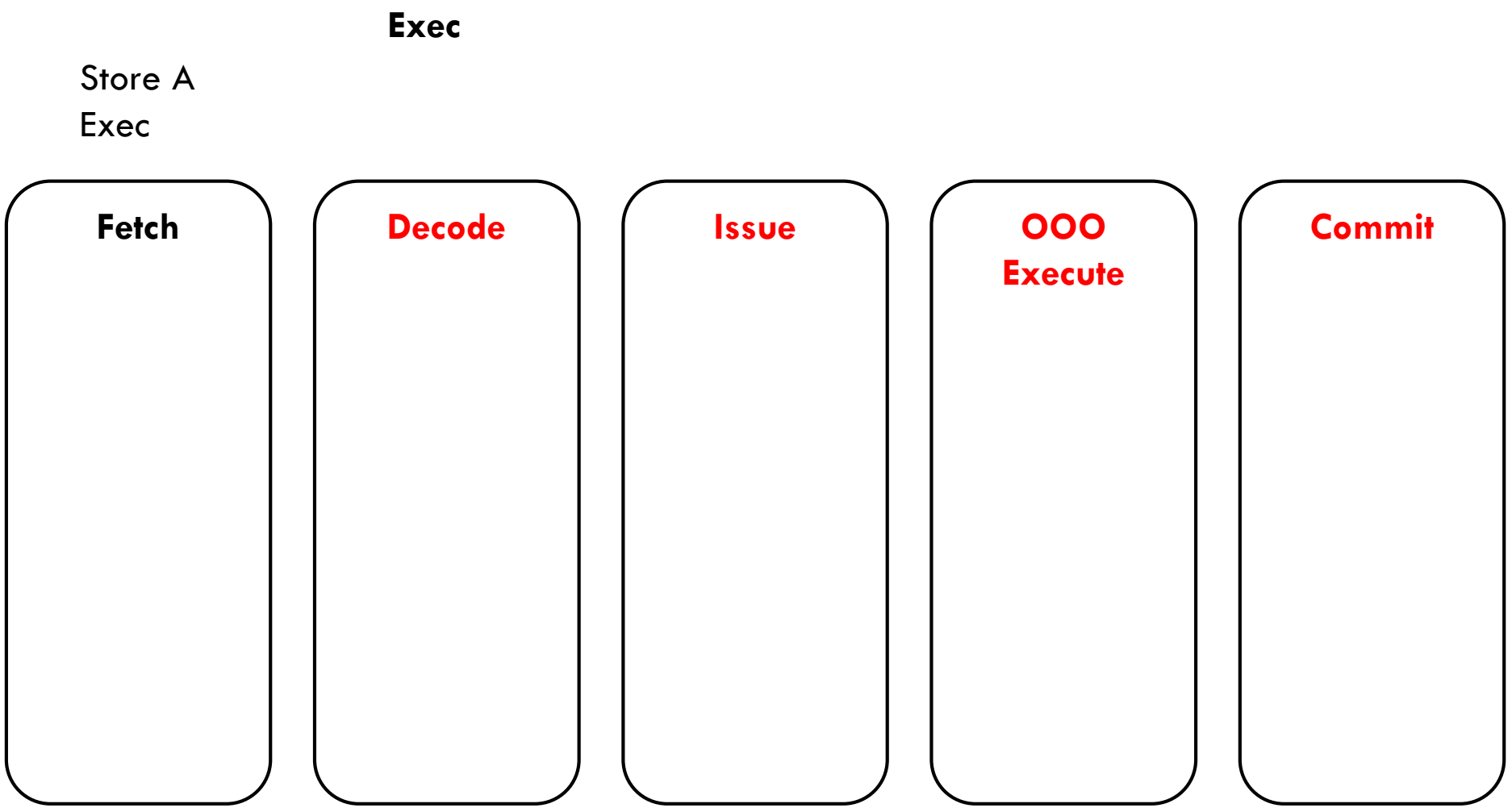
Store A

Exec



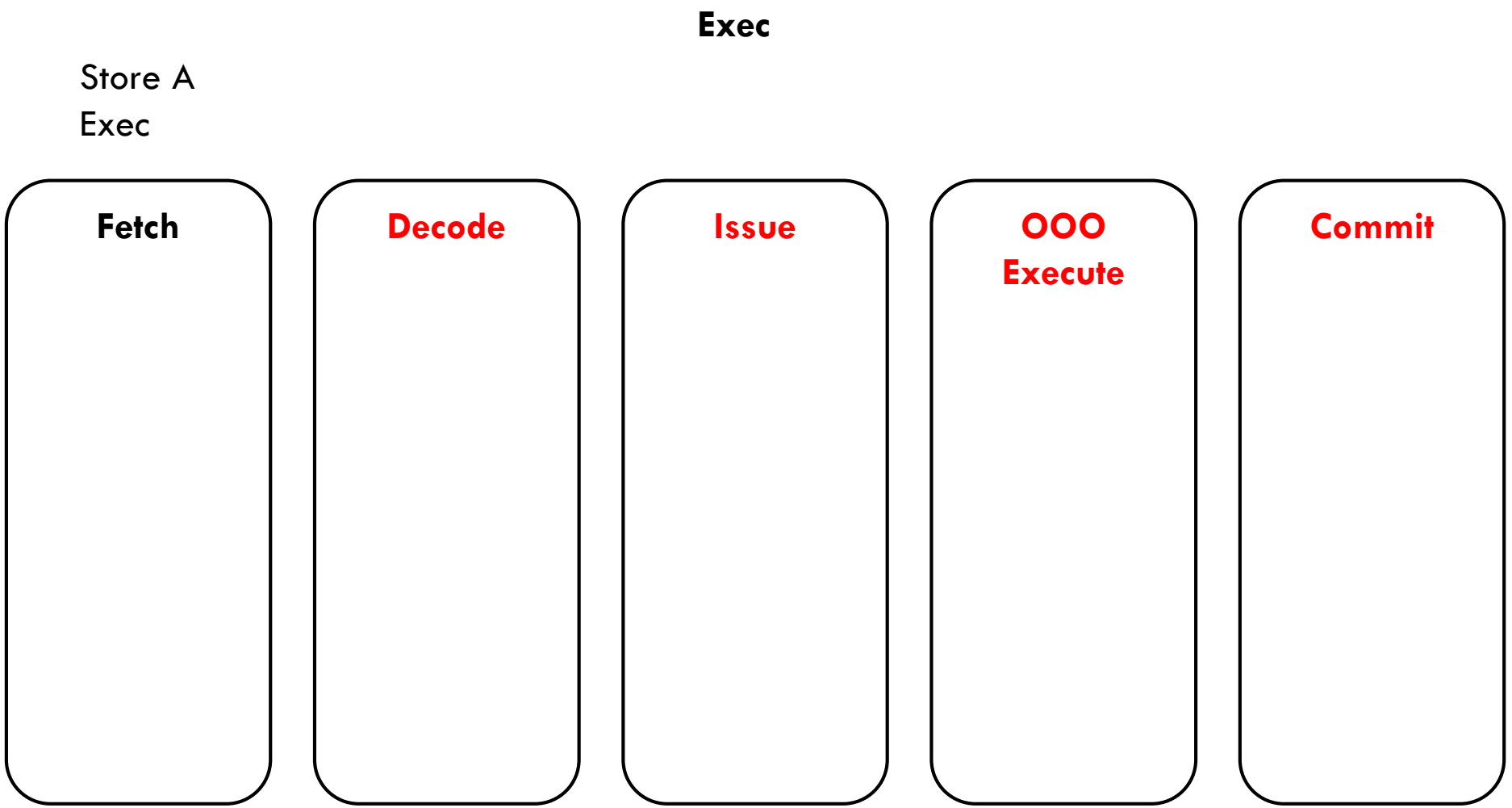
OOO Core - BBL

- Simulate all stages at once



OOO Core - BBL

- Simulate all stages at once

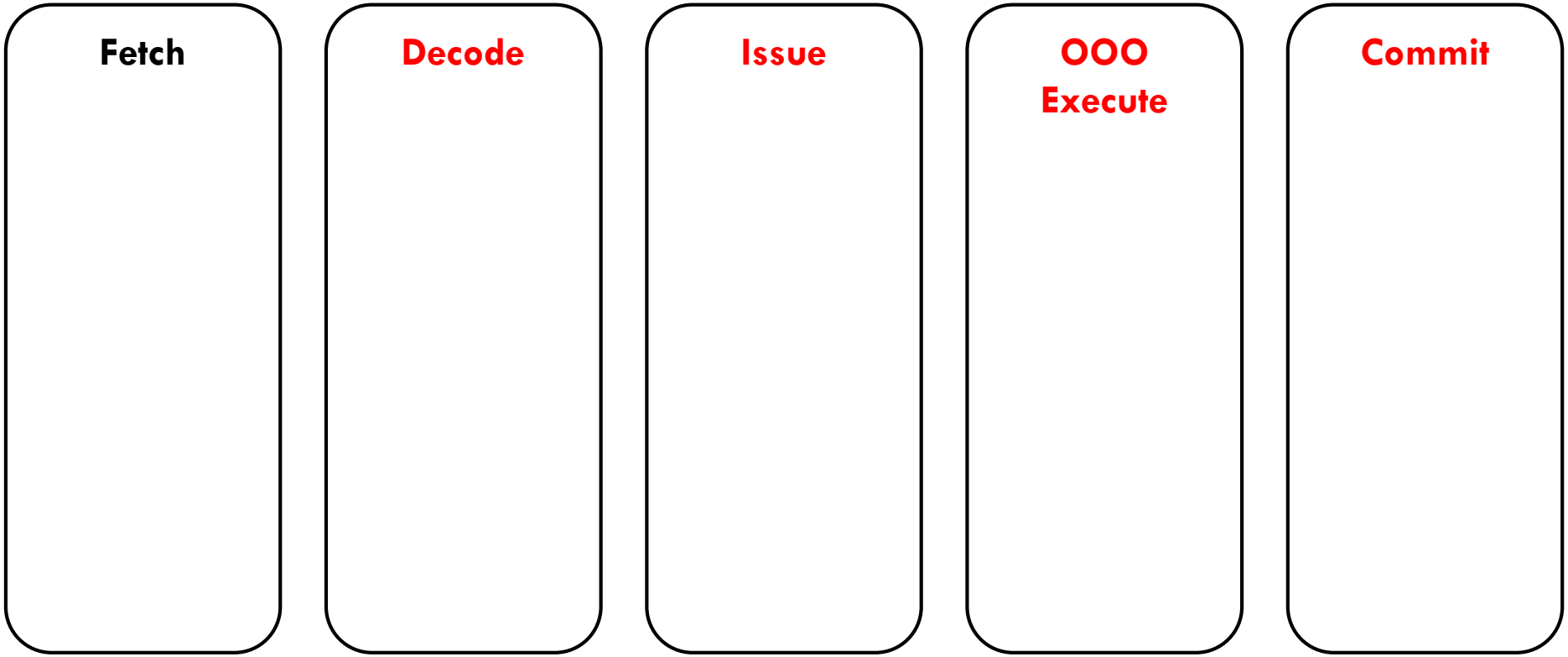


OOO Core - BBL

- Simulate all stages at once

Exec

Store A
Exec

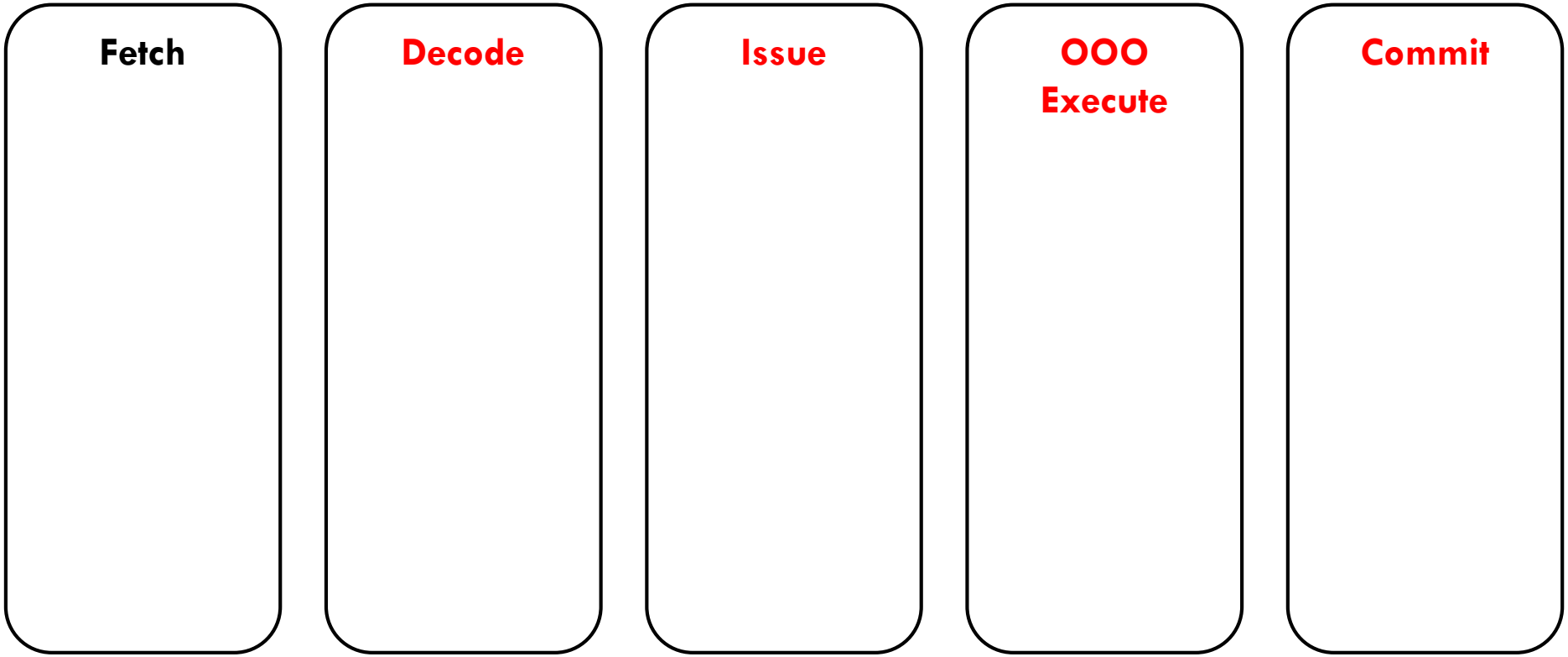


OOO Core - BBL

- Simulate all stages at once

Exec

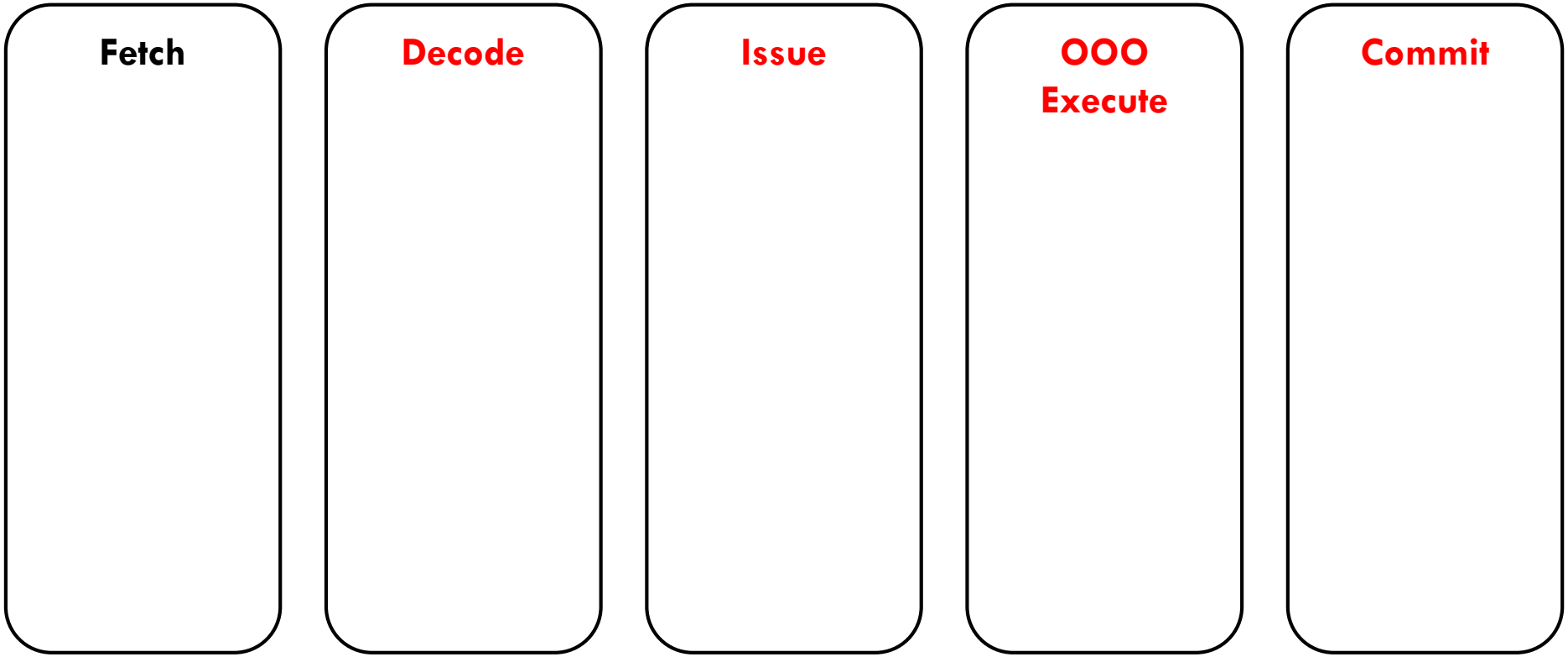
Store A
Exec



OOO Core - BBL

- Simulate all stages at once

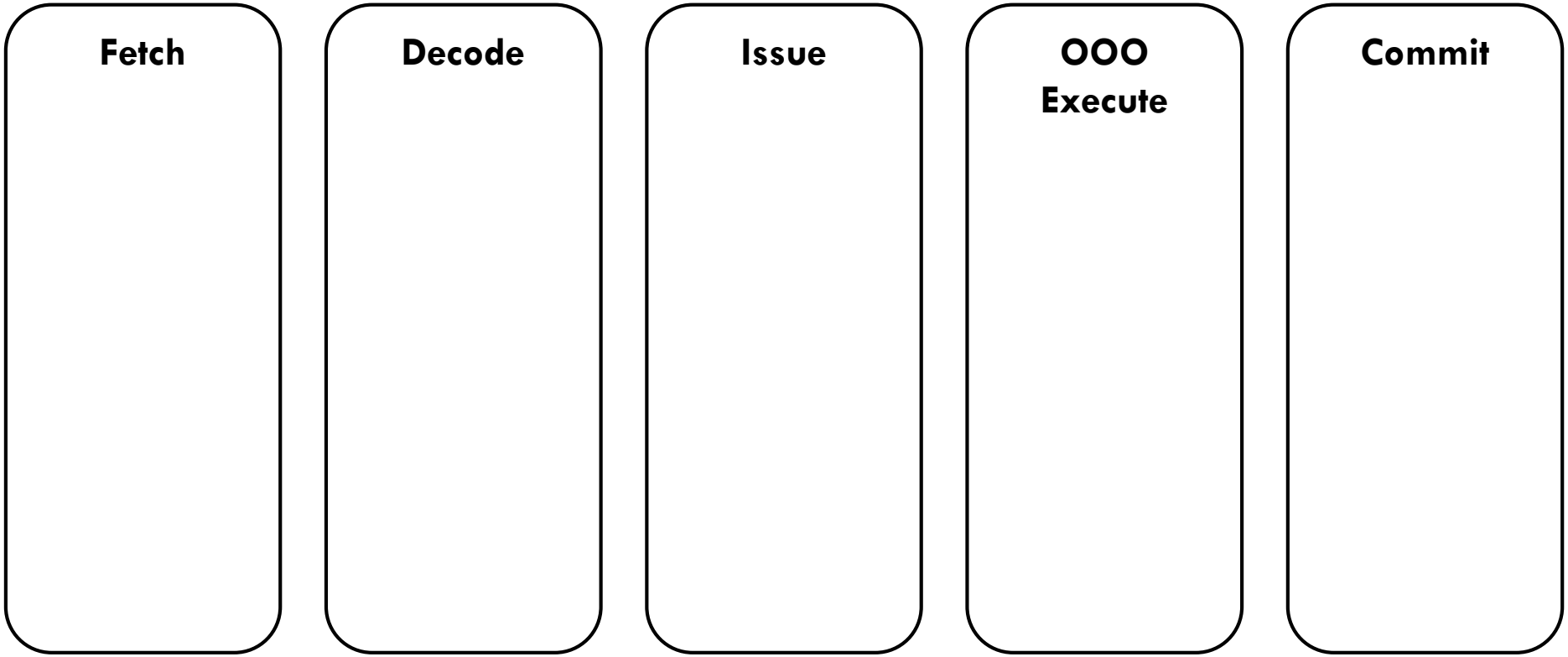
Store A
Exec



OOO Core - BBL

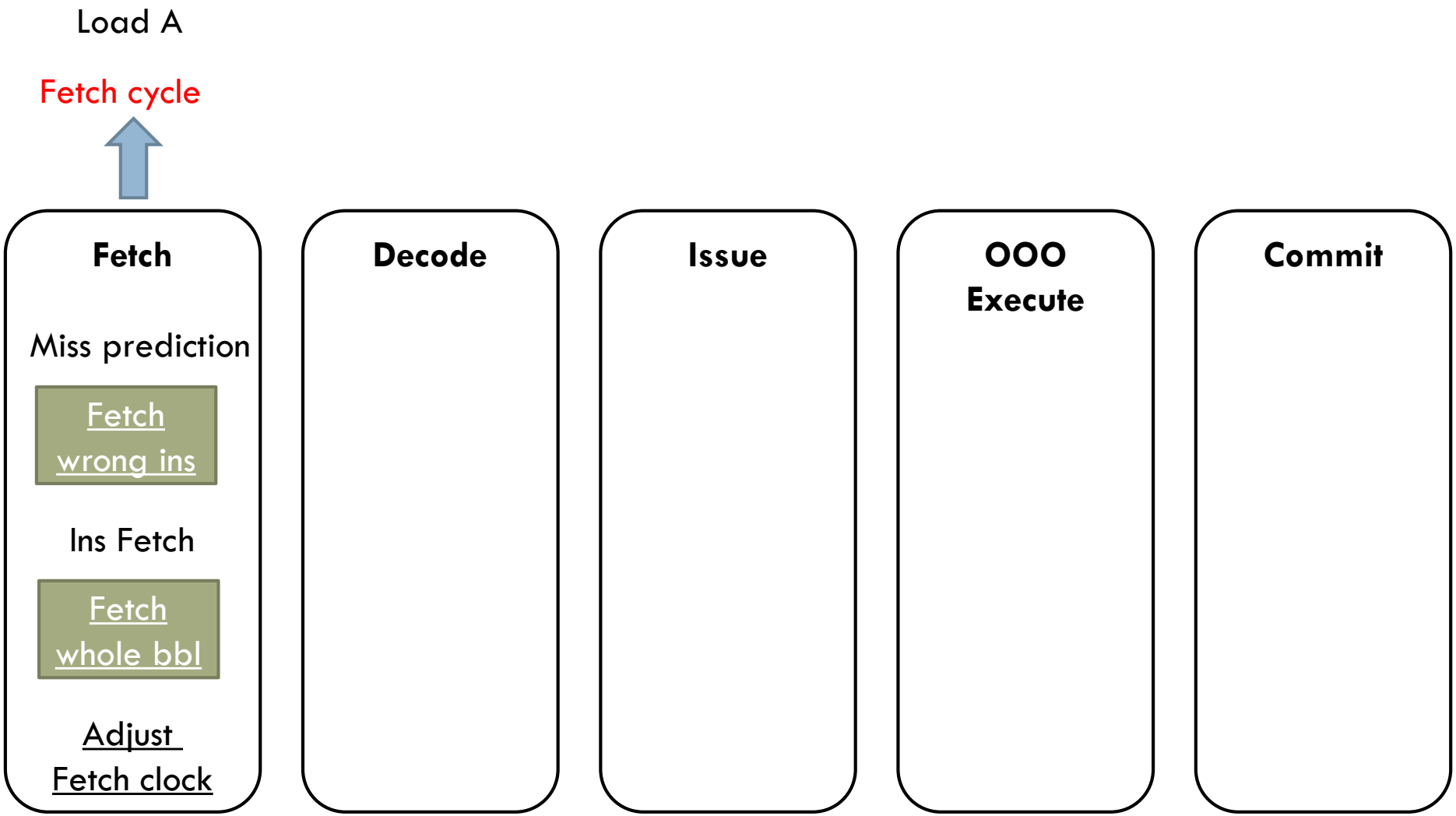
- Simulate all stages at once

  Load A



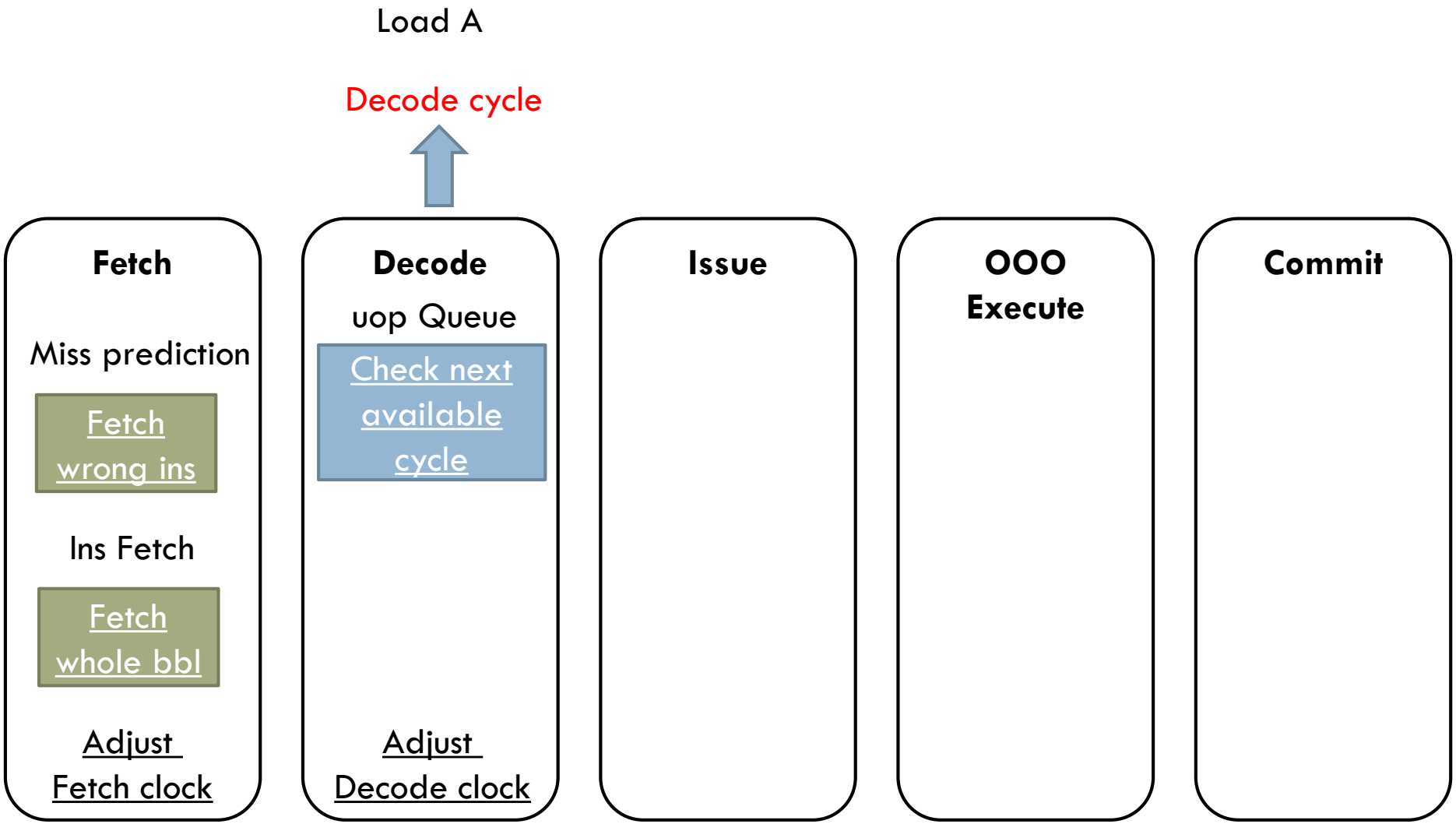
OOO Core - BBL

- Simulate all stages at once



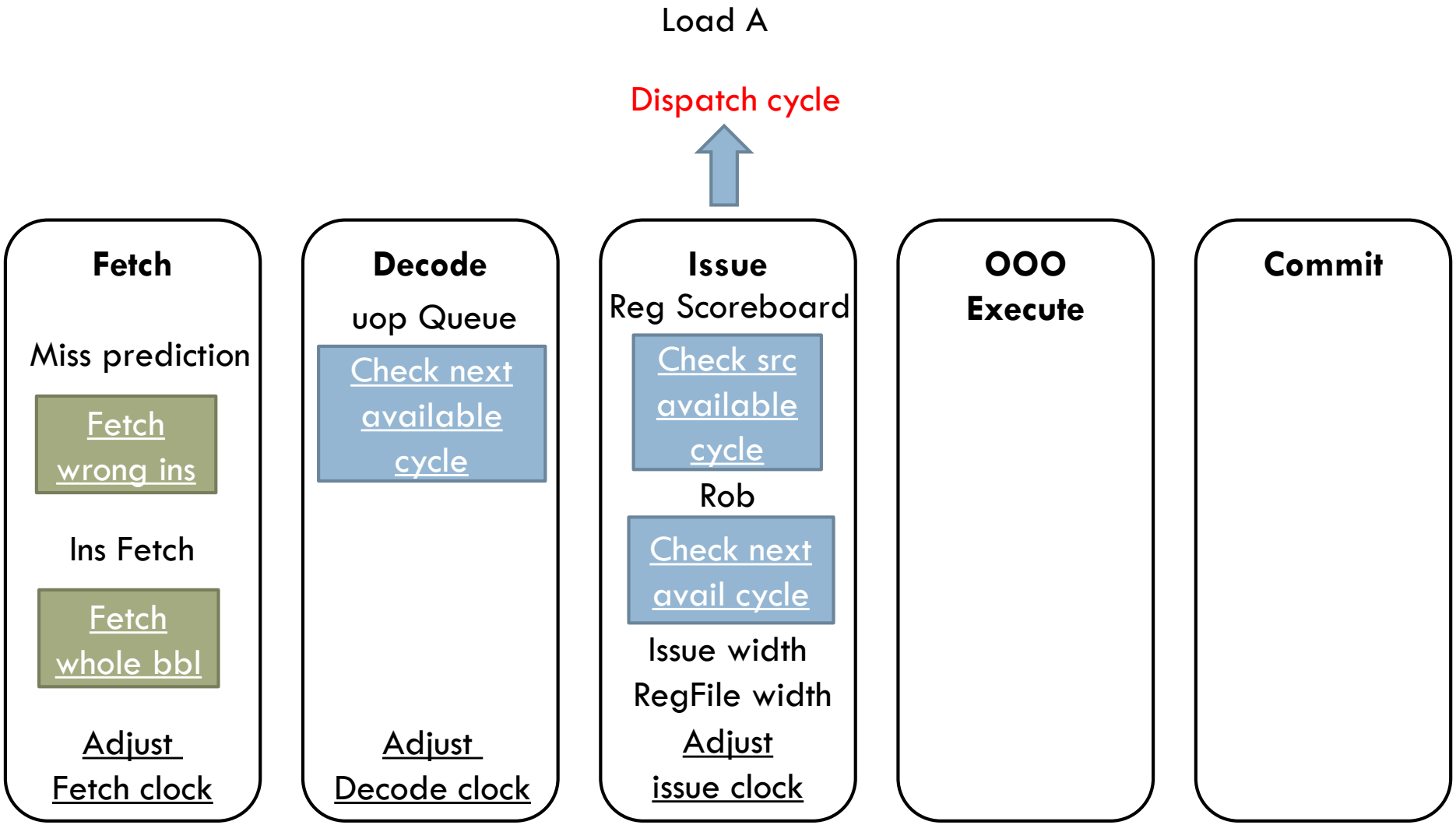
OOO Core - BBL

- Simulate all stages at once



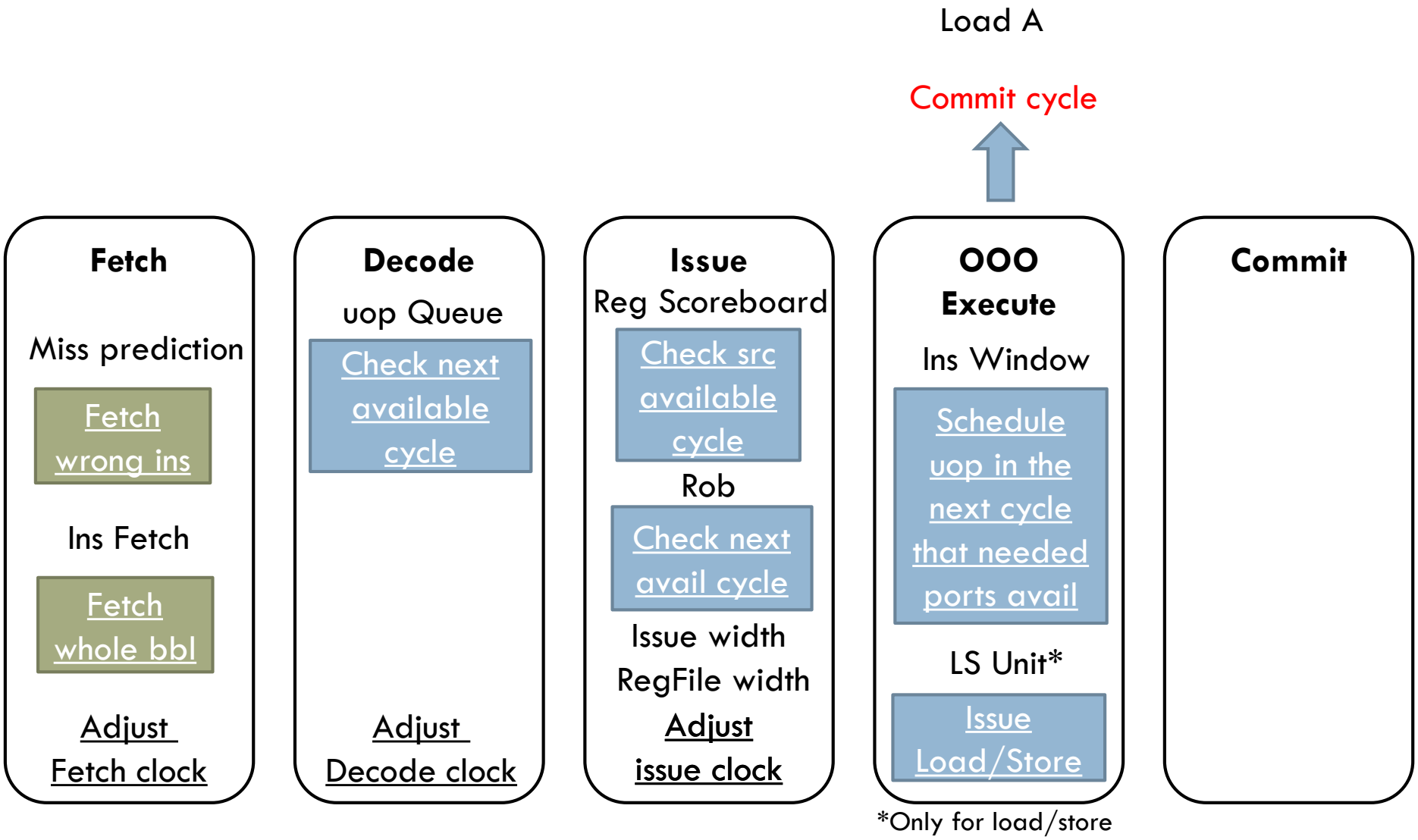
OOO Core - BBL

- Simulate all stages at once



OOO Core - BBL

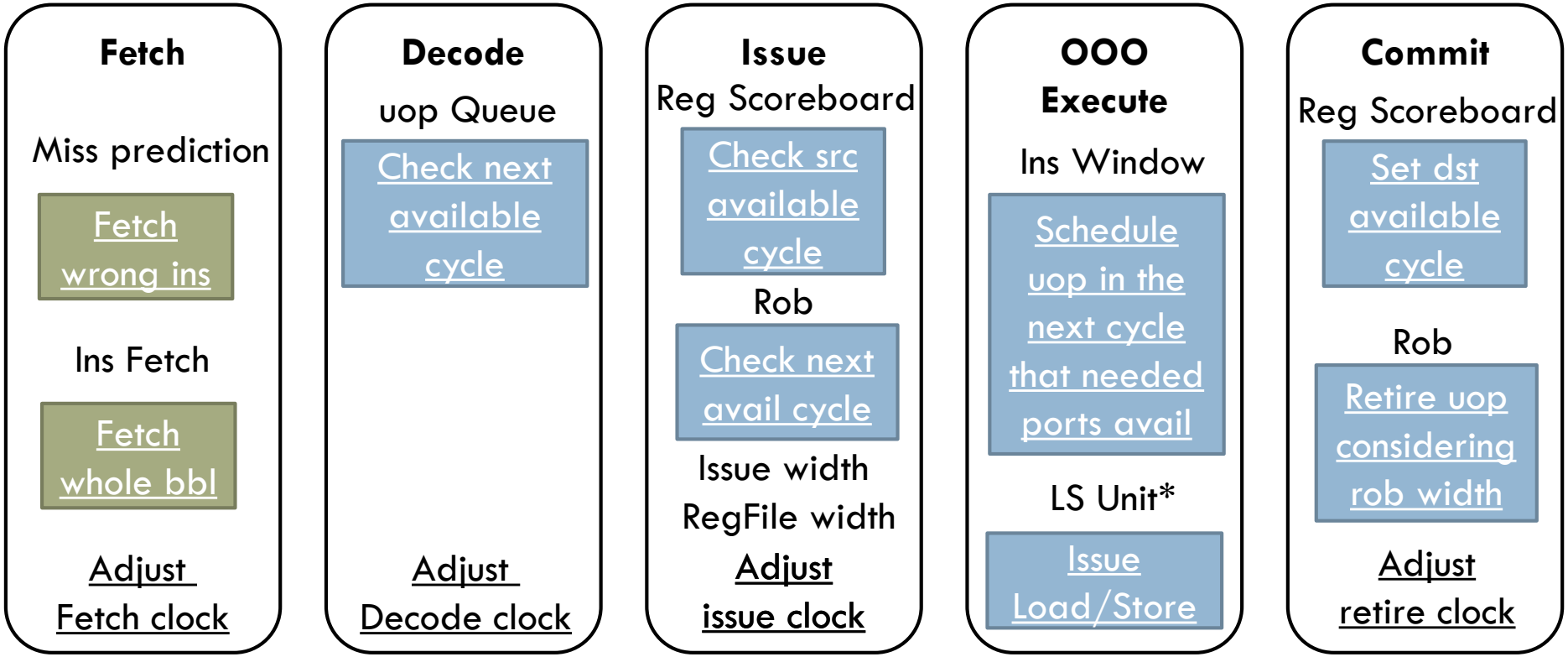
- Simulate all stages at once



OOO Core - BBL

- Simulate all stages at once

Load A



OOO Core – Load/Store

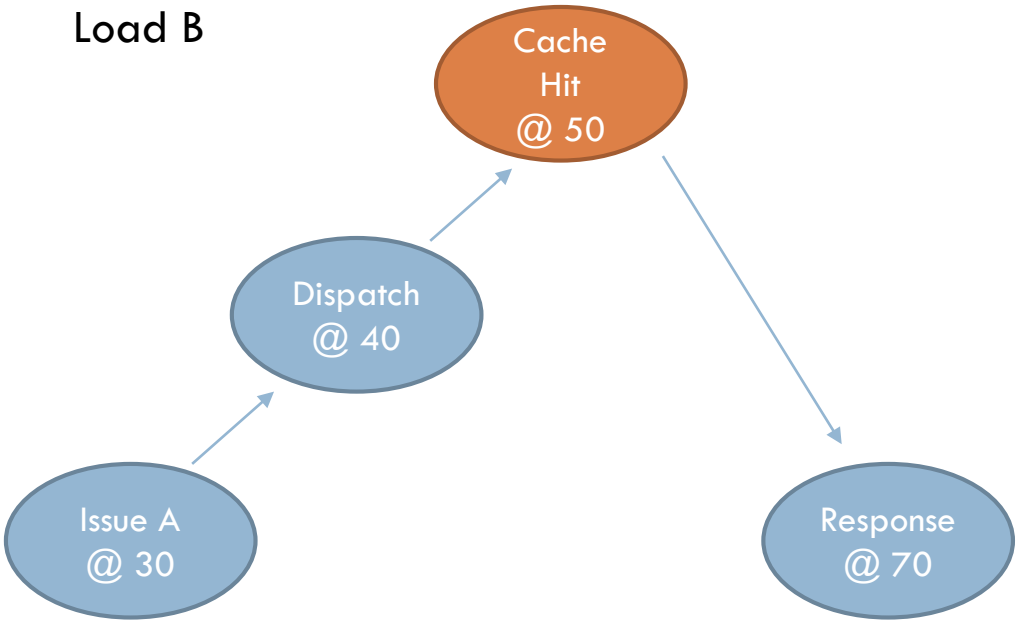
Simulate MLP

Load A
Load B

OOO Core – Load/Store

Simulate MLP

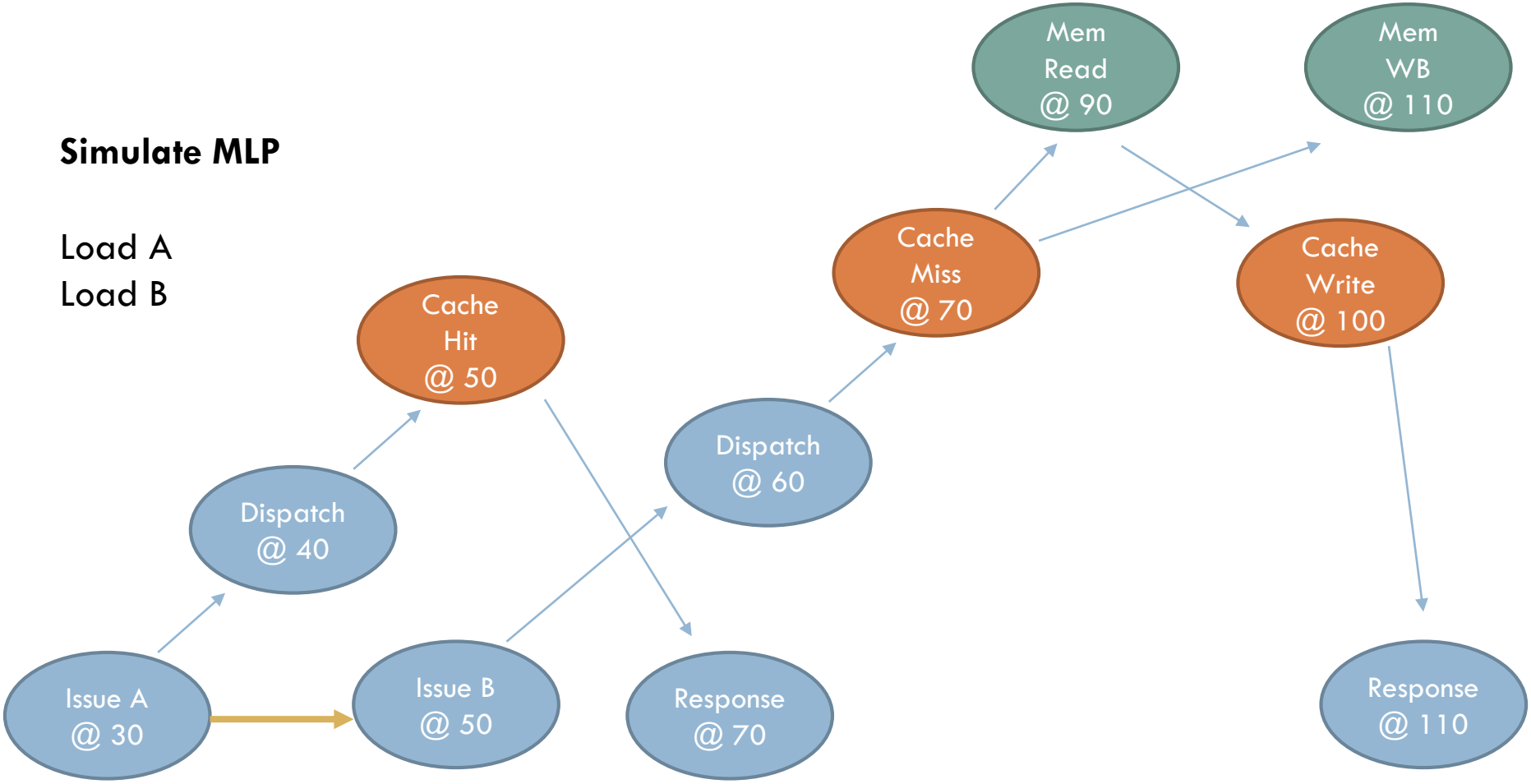
Load A
Load B



OOO Core – Load/Store

Simulate MLP

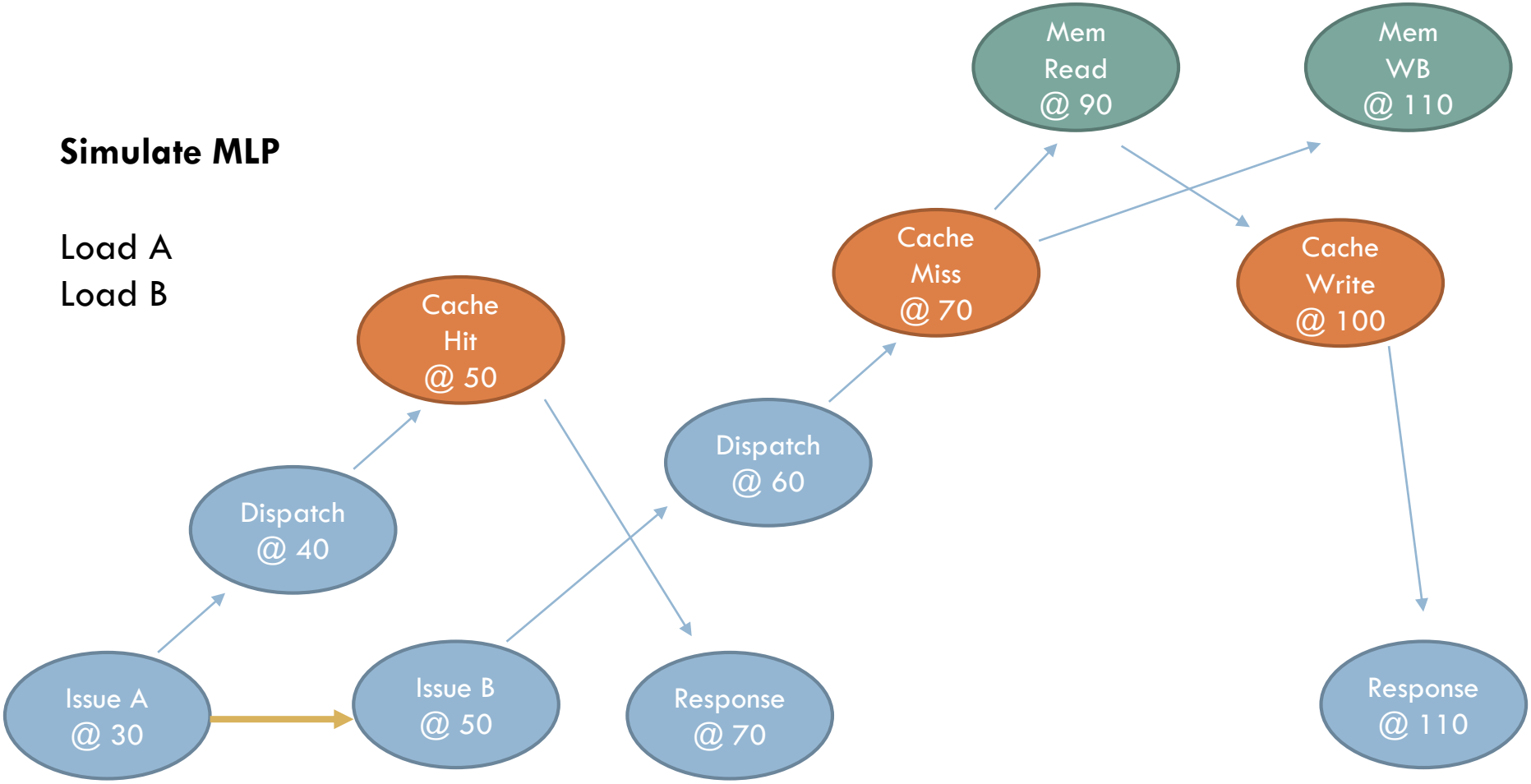
Load A
Load B



OOO Core – Load/Store

Simulate MLP

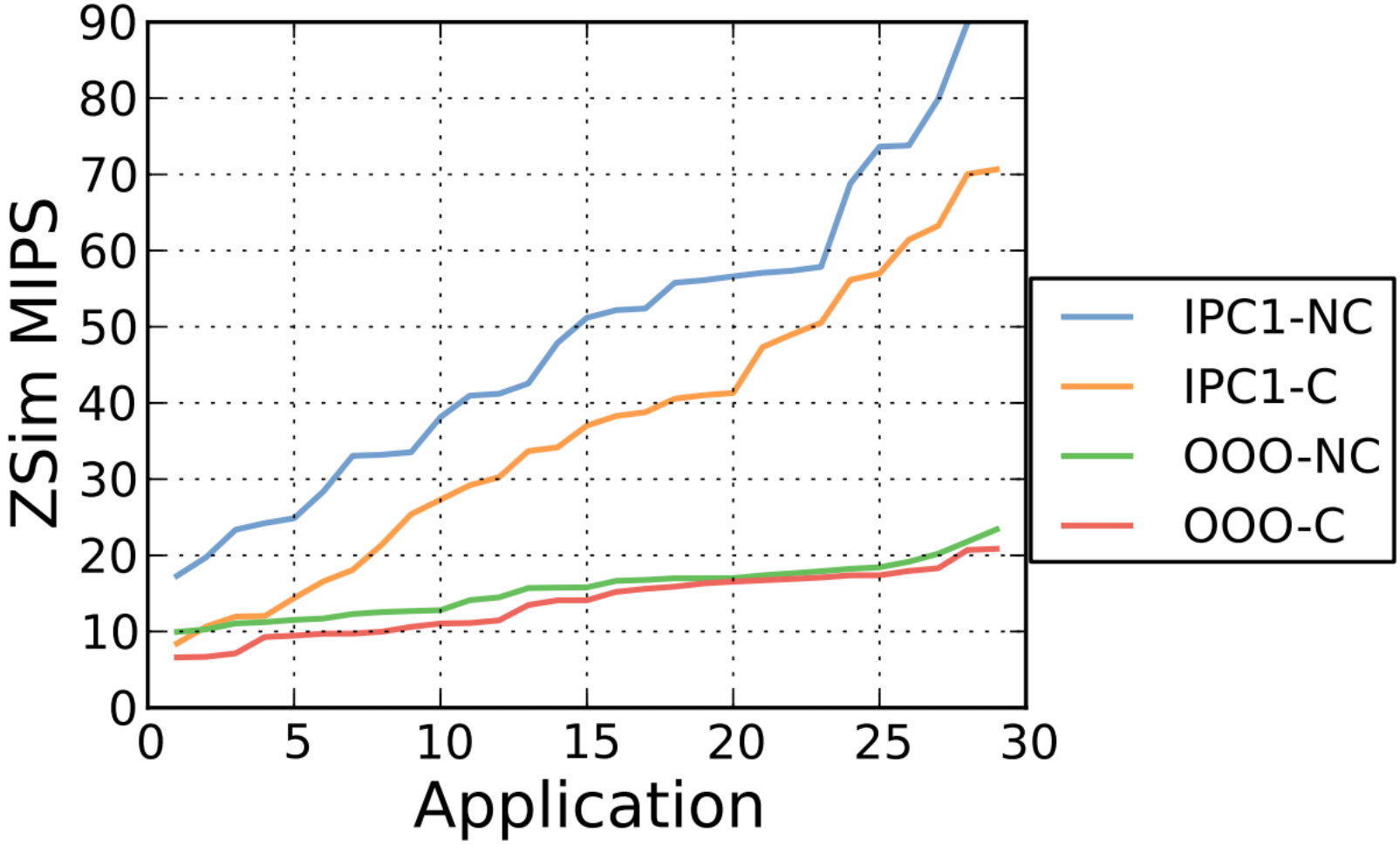
Load A
Load B



In weave phase, request B will not be delayed due to contentions for A

Simulation Speed for Different Core Type

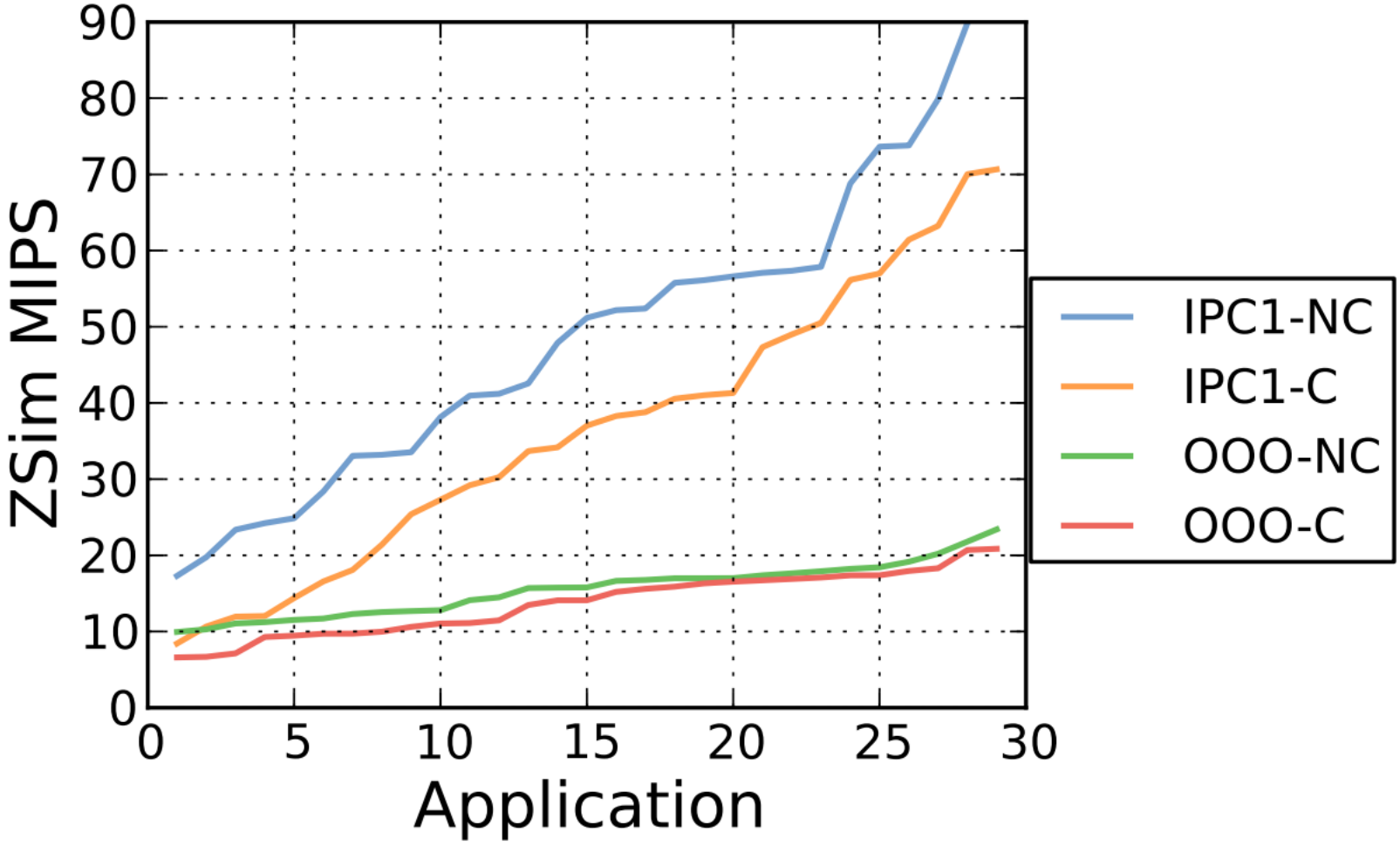
- SPECCPU 2006 suite



Simulation Speed for Different Core Type

- SPECCPU 2006 suite

~3X difference between IPC1 and OOO-C in Hmean



Not Modeled Core Behaviors

Not Modeled Core Behaviors

- Wrong path execution
  - Hard to simulate for Pin
  - Okay to skip for Westmere

Not Modeled Core Behaviors

- Wrong path execution
  - Hard to simulate for Pin
  - Okay to skip for Westmere
- Fine-grained message-passing
  - Need significant changes

Not Modeled Core Behaviors

- Wrong path execution
  - Hard to simulate for Pin
  - Okay to skip for Westmere
- Fine-grained message-passing
  - Need significant changes
- TLBs and SMT
  - Not supported yet

Coding Examples

Coding Examples

- Implement a branch predictor for OOO core

Coding Examples

- Implement a branch predictor for OOO core
- Change OOO core type
  - From Westmere to Silvermont

Implement Branch Predictors

Implement Branch Predictors

- Have a new branch predictor class

Implement Branch Predictors

- Have a new branch predictor class

```
class GShareBranchPredictor {
```

Implement Branch Predictors

- Have a new branch predictor class

```
class GShareBranchPredictor {
    private:
        bool lastSeen;
    ……
}
```

Implement Branch Predictors

- Have a new branch predictor class

```
class GShareBranchPredictor {
    private:
        bool lastSeen;
    ……
}
```

- Implement the predict method

Implement Branch Predictors

- Have a new branch predictor class

```
class GShareBranchPredictor {
    private:
        bool lastSeen;
    ……
}
```

- Implement the predict method

```
public:
    // Predicts and updates; returns false if mispredicted
    inline bool predict(Address branchPc, bool taken) {
```

Implement Branch Predictors

- Have a new branch predictor class

```
class GShareBranchPredictor {
    private:
        bool lastSeen;
    ……
}
```

- Implement the predict method

```
public:
    // Predicts and updates; returns false if mispredicted
    inline bool predict(Address branchPc, bool taken) {
        bool prediction = (taken == lastSeen);
```

Implement Branch Predictors

- Have a new branch predictor class

```
class GShareBranchPredictor {
    private:
        bool lastSeen;
    ……
}
```

- Implement the predict method

```
public:
    // Predicts and updates; returns false if mispredicted
    inline bool predict(Address branchPc, bool taken) {
        bool prediction = (taken == lastSeen);
        lastSeen = taken;
```

Implement Branch Predictors

- Have a new branch predictor class

```
class GShareBranchPredictor {
    private:
        bool lastSeen;
    ……
}
```

- Implement the predict method

```
public:
    // Predicts and updates; returns false if mispredicted
    inline bool predict(Address branchPc, bool taken) {
        bool prediction = (taken == lastSeen);
        lastSeen = taken;
        return prediction; // always predict taken
    }
```

Implement Branch Predictors

- Have a new branch predictor class

```
class GShareBranchPredictor {
    private:
        bool lastSeen;
    ……
}
```

- Implement the predict method

```
public:
    // Predicts and updates; returns false if mispredicted
    inline bool predict(Address branchPc, bool taken) {
        bool prediction = (taken == lastSeen);
        lastSeen = taken;
        return prediction; // always predict taken
    }
```

- Replace the branch predictor in ooo_core.h

Implement Branch Predictors

- Have a new branch predictor class

```
class GShareBranchPredictor {
    private:
        bool lastSeen;
    ……
}
```

- Implement the predict method

```
public:
    // Predicts and updates; returns false if mispredicted
    inline bool predict(Address branchPc, bool taken) {
        bool prediction = (taken == lastSeen);
        lastSeen = taken;
        return prediction; // always predict taken
    }
```

- Replace the branch predictor in ooo_core.h

```
    //BranchPredictorPAg<11, 18, 14> branchPred;
    GSharePredictor branchPred;
```

Demo

Different OOO Micro-architecture

Different OOO Micro-architecture

- The original zsim assumes Westmere OOO core, but what if I want to simulate a Silvermont/Haswell OOO core?

Different OOO Micro-architecture

- The original zsim assumes Westmere OOO core, but what if I want to simulate a Silvermont/Haswell OOO core?

- Step 1: obtain the important ooo core parameters

Different OOO Micro-architecture

- The original zsim assumes Westmere OOO core, but what if I want to simulate a Silvermont/Haswell OOO core?
- Step 1: obtain the important ooo core parameters
- Step 2: change the core parameters in ooo_core.h/cpp

Different OOO Micro-architecture

- The original zsim assumes Westmere OOO core, but what if I want to simulate a Silvermont/Haswell OOO core?
- Step 1: obtain the important ooo core parameters
- Step 2: change the core parameters in ooo_core.h/cpp
- Step 3: verify it against real system

Obtain Important Core Parameters

	Westmere[1]	Silvermont[2]
Issue width	4	2
F/D/I/E stages	1/4/7/13	1/3/5/8
Fetch width	16B	8B
RF read width	3	2
ROB size	128	32
Ins window	1K * 36	1K * 16
Issue queue	28	8

- [1] http://www.realworldtech.com/nehalem/
- [2] http://www.realworldtech.com/silvermont/

Change OOO Core Parameters

Change OOO Core Parameters

- Change sizes of hardware structures in ooo_core.h

Change OOO Core Parameters

- Change sizes of hardware structures in ooo_core.h
  - CycleQueue<28> uopQueue

    -> <8>
  - ReorderBuffer<128, 4> rob

    -> <32, 2>

Change OOO Core Parameters

- Change sizes of hardware structures in ooo_core.h
  - CycleQueue<28> uopQueue

    -> <8>
  - ReorderBuffer<128, 4> rob

    -> <32, 2>
- Change the ooo core parameter in ooo_core.cpp

Change OOO Core Parameters

- Change sizes of hardware structures in ooo_core.h
  - CycleQueue<28> uopQueue

    -> <8>
  - ReorderBuffer<128, 4> rob

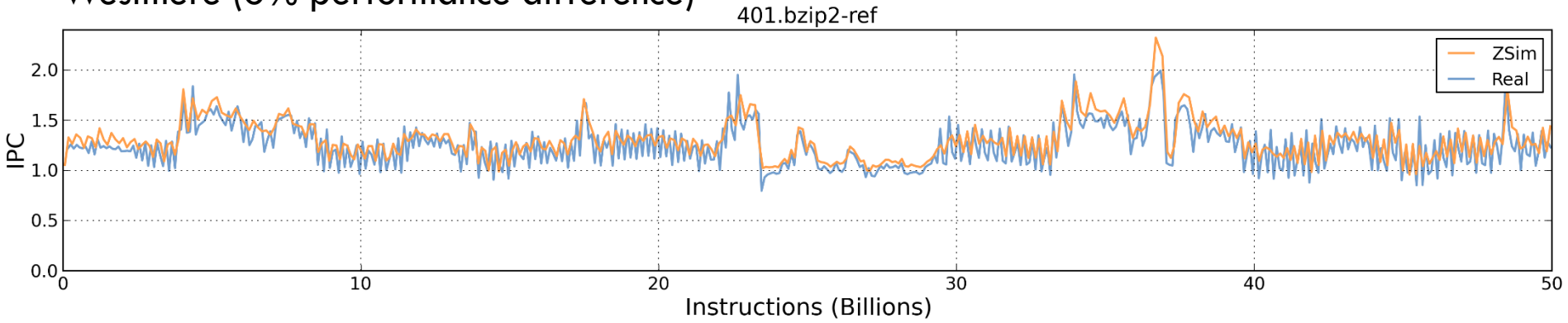
    -> <32, 2>
- Change the ooo core parameter in ooo_core.cpp
  - #define FETCH_STAGE 1 -> 1
  - #define DECODE_STAGE 4 -> 3
  - #define ISSUE_STAGE 7 -> 5
  - #define DISPATCH_STAGE 13 -> 8
  - #define FETCH_BYTES_PER_CYCLE 16 -> 8
  - #define ISSUES_PER_CYCLE 4 -> 2
  - #define RF_READS_PER_CYCLE 3 -> 2

Demo

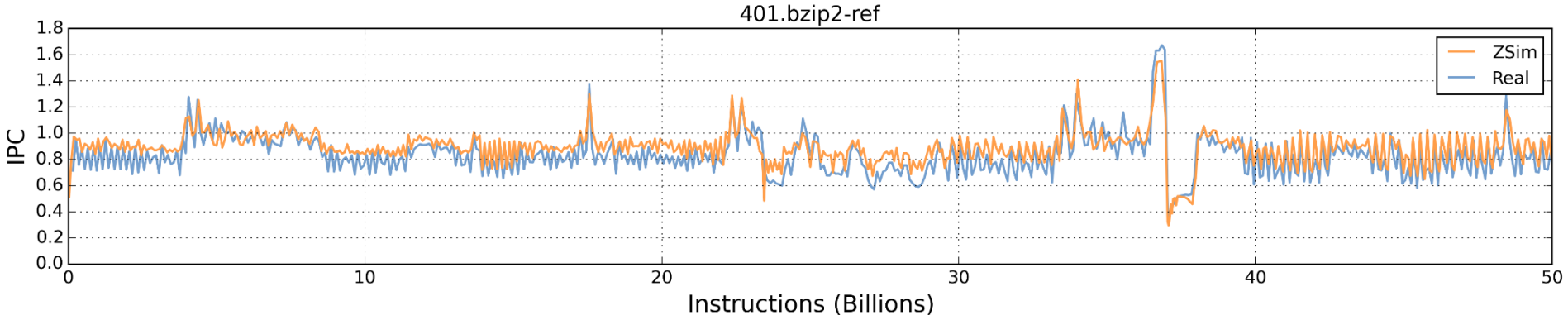
Verify It Against Real System

- IPC traces for Westmere and Silvermont

Westmere (6% performance difference)



Silvermont (9% performance difference)



Summary

Summary

- ZSim uses instruction-driven simulation for core activities and event-driven simulation for uncore activities

Summary

- ZSim uses instruction-driven simulation for core activities and event-driven simulation for uncore activities

- ZSim currently supports 3 types of core
  - Simple IPC1 core (simple_core.h)
  - Timing core (timing_core.h)
  - Westmere-like OOO core (ooo_core.h)

Summary

- ZSim uses instruction-driven simulation for core activities and event-driven simulation for uncore activities

- ZSim currently supports 3 types of core
  - Simple IPC1 core (simple_core.h)
  - Timing core (timing_core.h)
  - Westmere-like OOO core (ooo_core.h)

- Extending zsim core model is straightforward
  - Modify 4 basic analysis routines
  - Substitute the hardware structure with your implementation
  - Change the parameters in OOO

Hacking Advice

- As common Pin programming, functions in the core are very frequently called in zsim
  - You should be aware of performance when coding
  - It's the main reason why zsim statically allocates hardware structures and set ooo parameters

Thank you!

Any questions?

Break / Q&A

Try zsim now!

https://zsim.csail.mit.edu