SiFive

Scheduling Model in LLVM

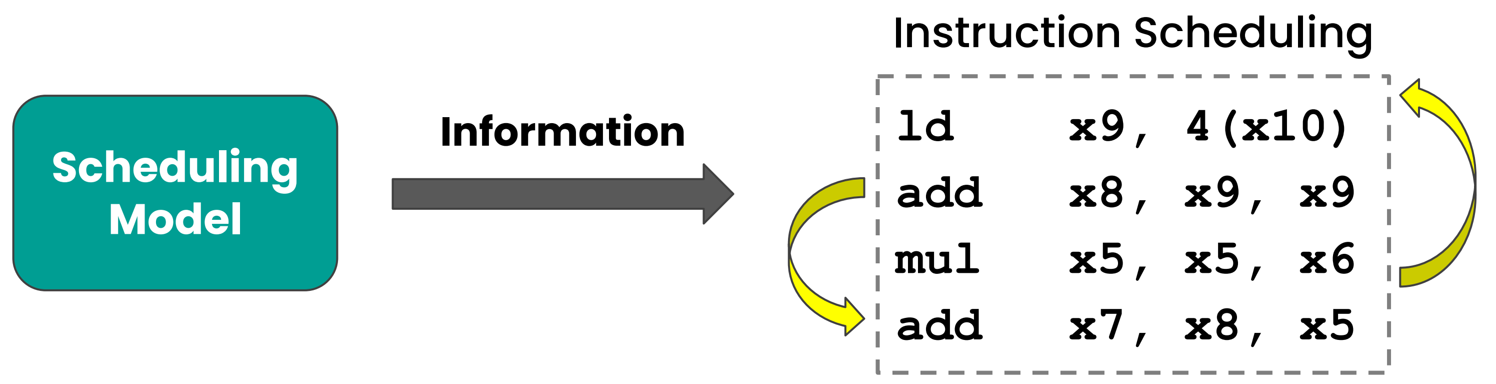
Past, Present, and Future

By Min Hsu @ LLVM Dev Meeting 2025

By the end of this tutorial…

- **What** is LLVM’s Scheduling Model
- **How** do *other* parts of LLVM use Scheduling Model
- **What** are Scheduling Model’s connections with hardware
- **How** can we improve this framework

Scheduling Model: The Inception

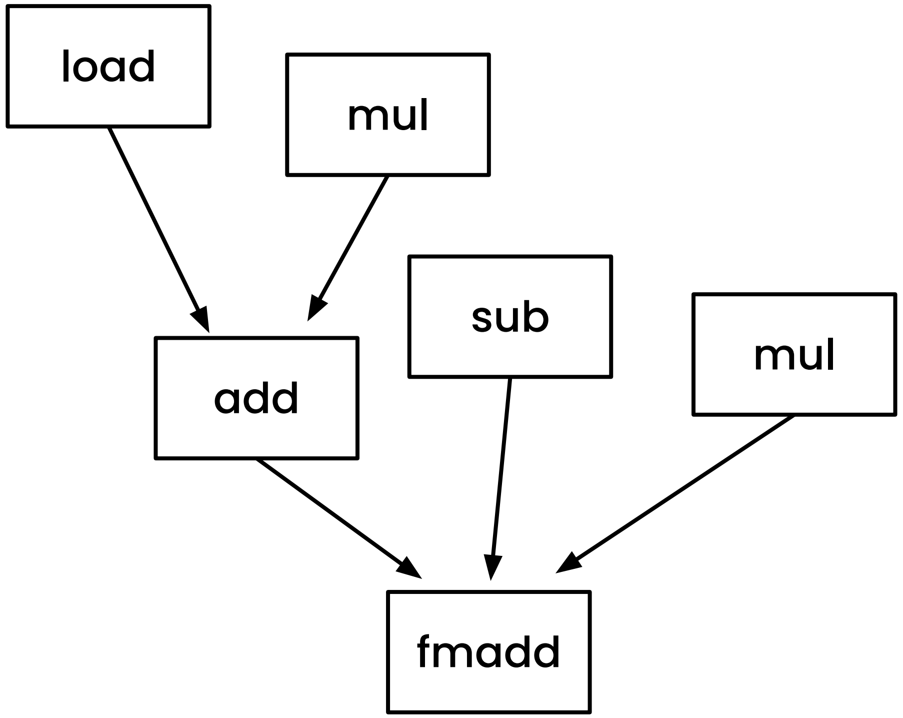


```
ld     x9, 4(x10)
add    x8, x9, x9
mul    x5, x5, x6
add    x7, x8, x5
```

- Increase instruction level parallelism
- Reduce the number of register spillings

Instruction Scheduling

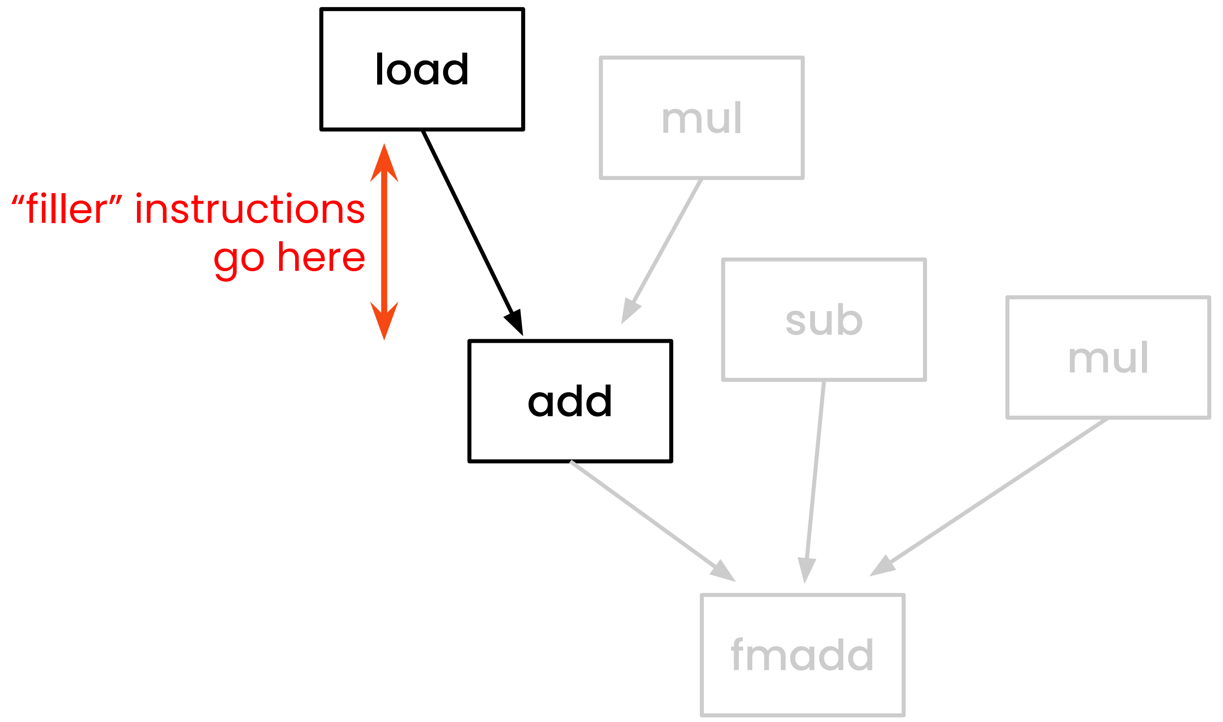
Increase Instruction Level Parallelism



Avoid *stalling* the processor pipeline

Instruction Scheduling

Increase Instruction Level Parallelism

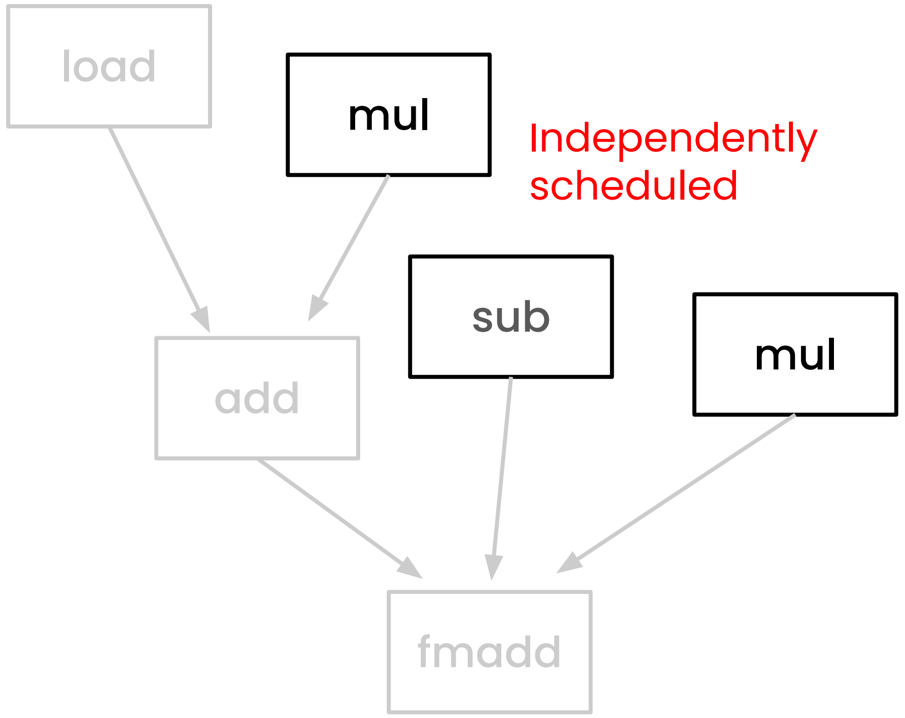


Avoid *stalling* the processor pipeline

- Make the pipeline **busy**

Instruction Scheduling

Increase Instruction Level Parallelism

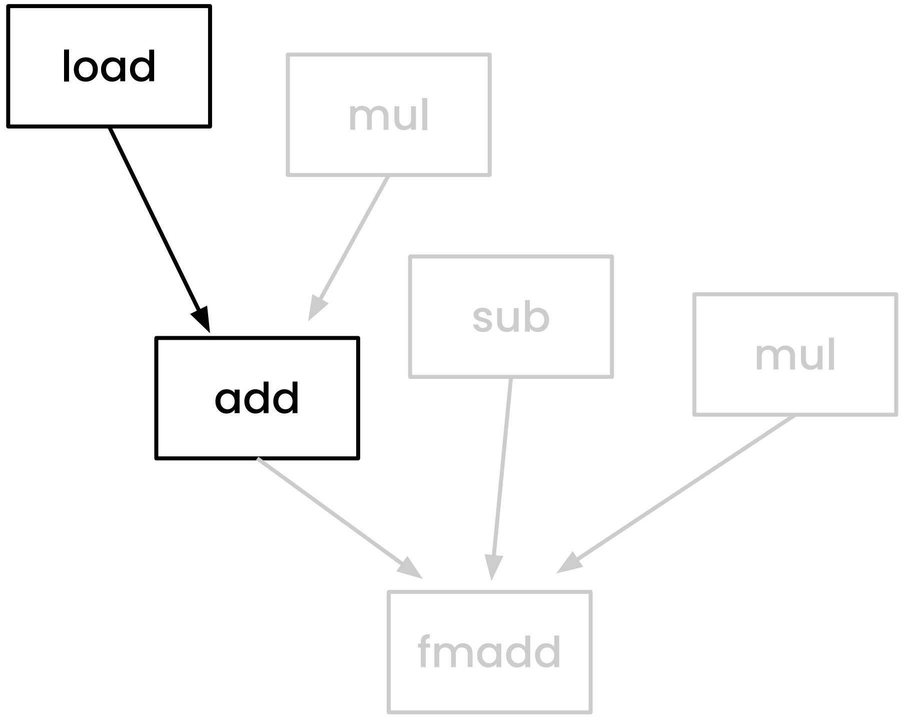


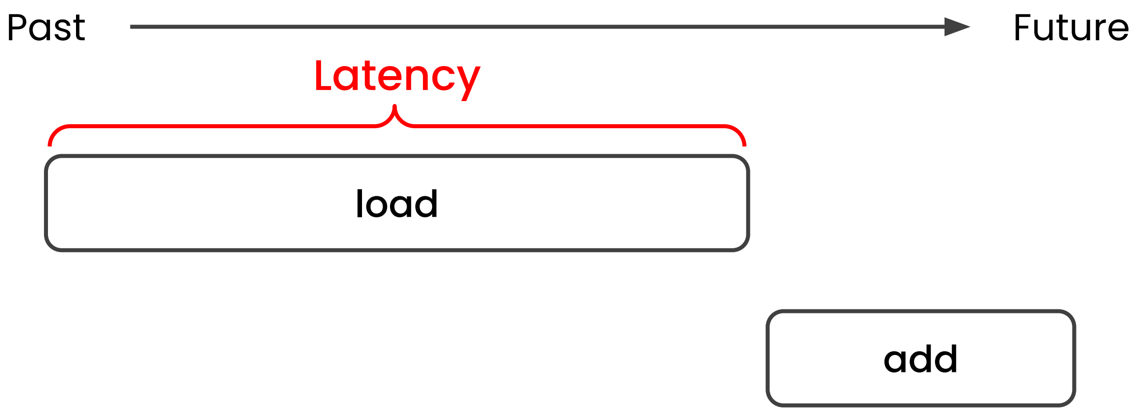
Avoid *stalling* the processor pipeline

- Make the pipeline **busy**
- Avoid multiple instructions competing for the same processor **resource**

Instruction Scheduling

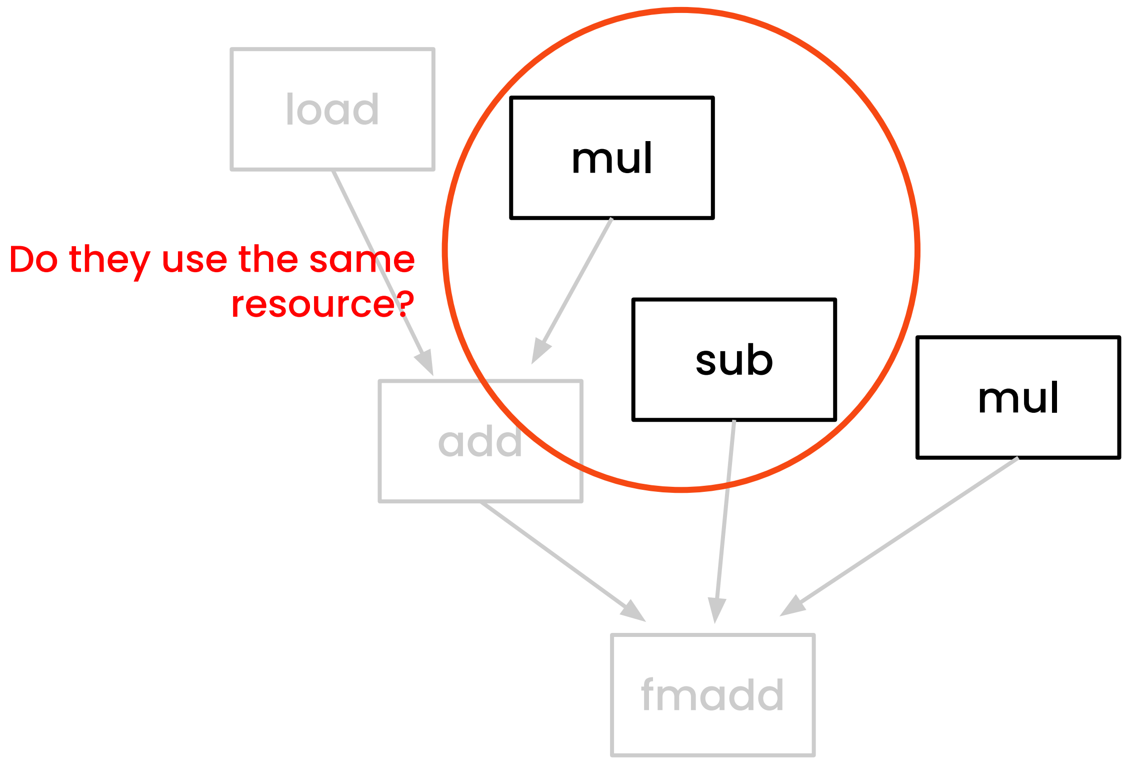
Increase Instruction Level Parallelism

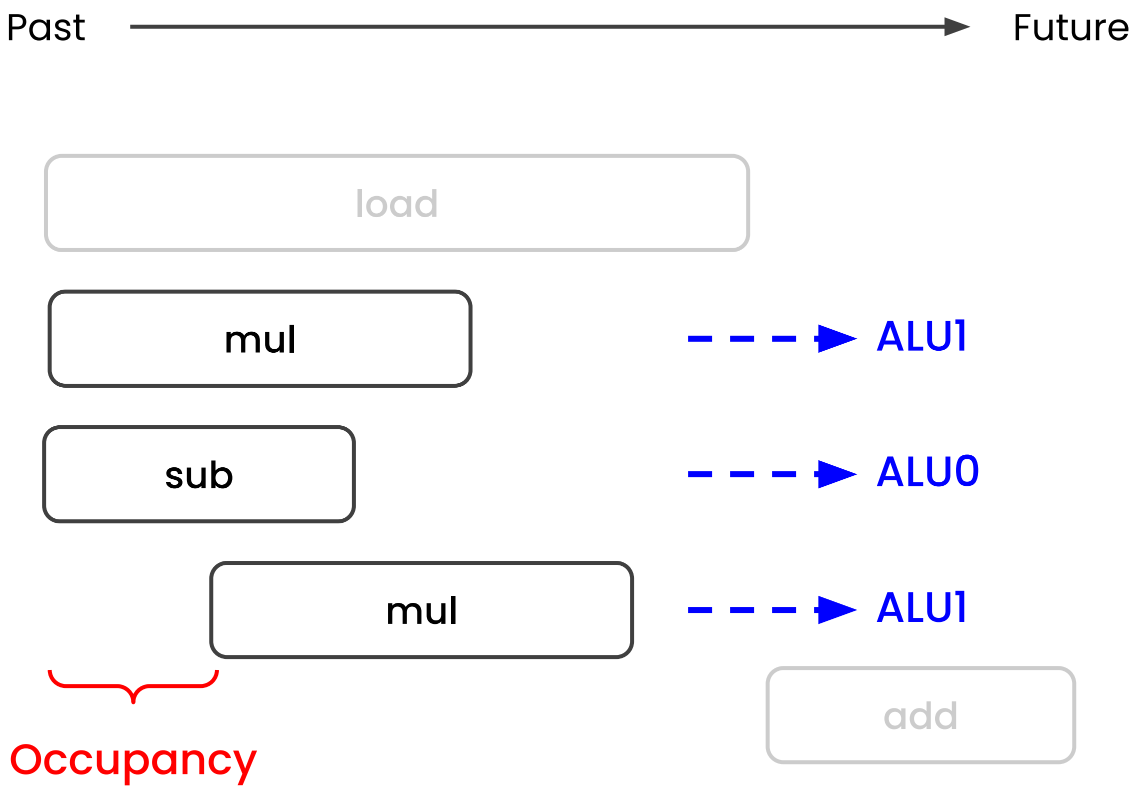




Instruction Scheduling

Increase Instruction Level Parallelism

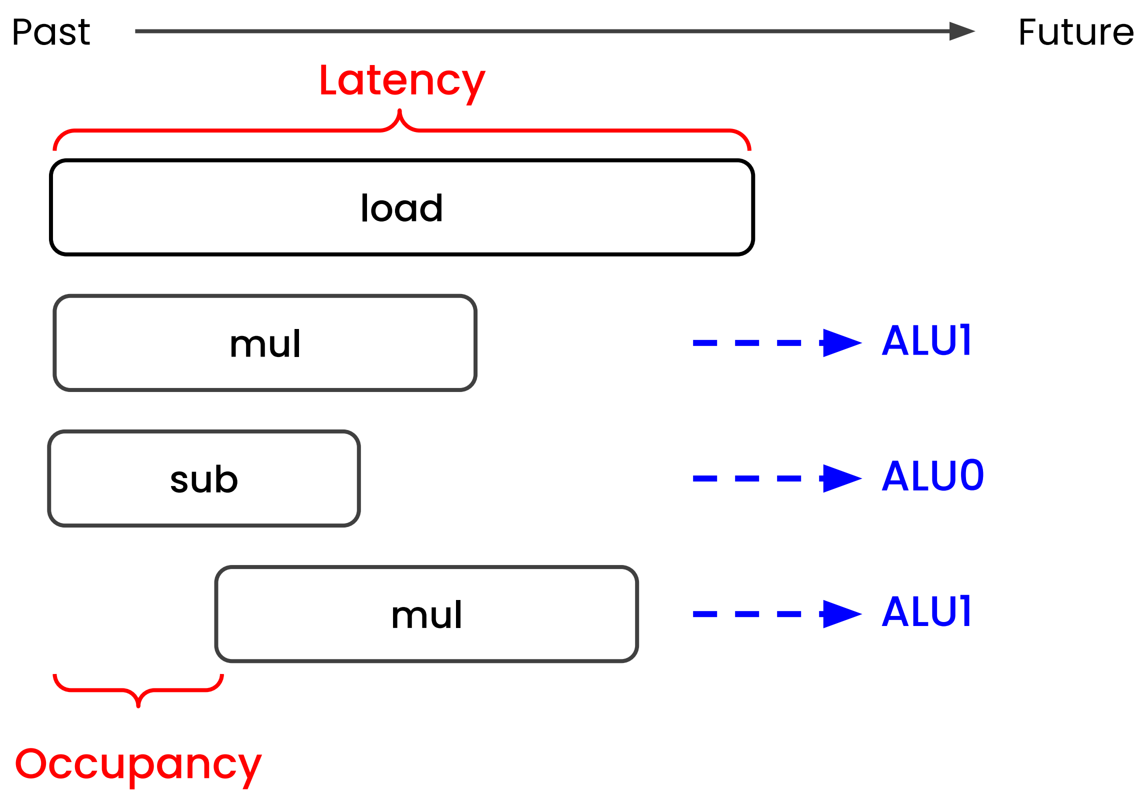




*Assuming we have infinite issue width

Latency, Resource, and Occupancy

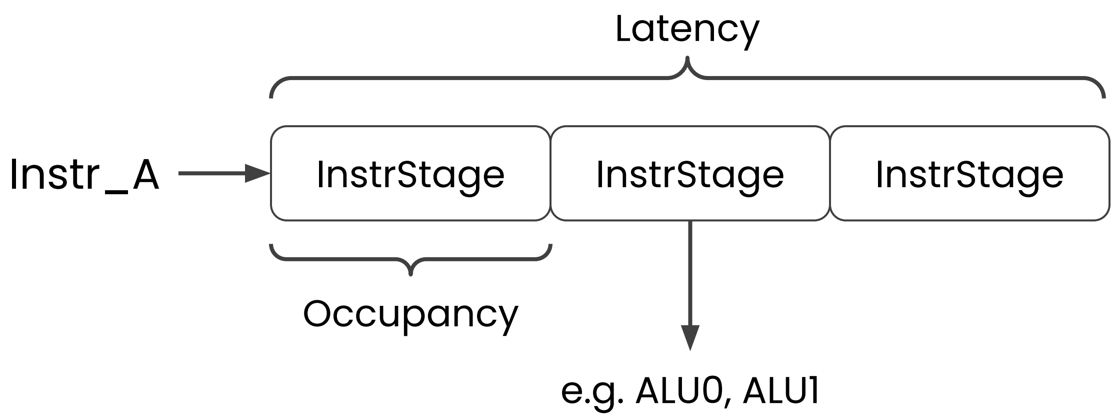
Instruction	Latency (cycles)	Resource	Occupancy (cycles)
load	10	LSU	2
mul	4	ALU1	2
sub	3	ALU0	1



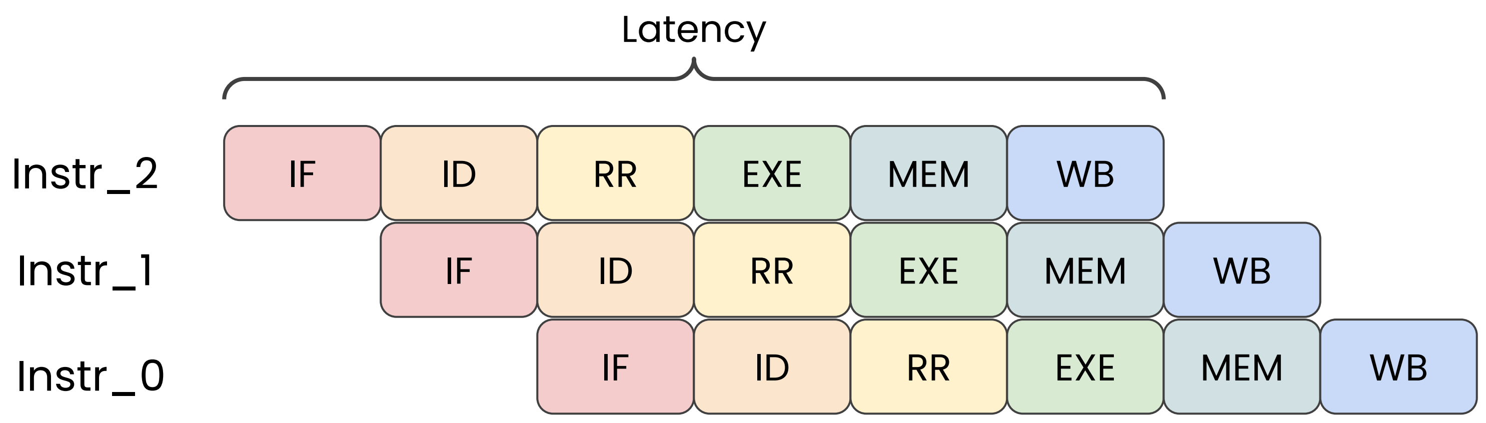
Itinerary Scheding Model

Legacy scheduling model framework

- Split instruction execution into *stages*
- Duration (i.e. occupancy) and resources used in each stage
- When will the result be available (i.e. latency)



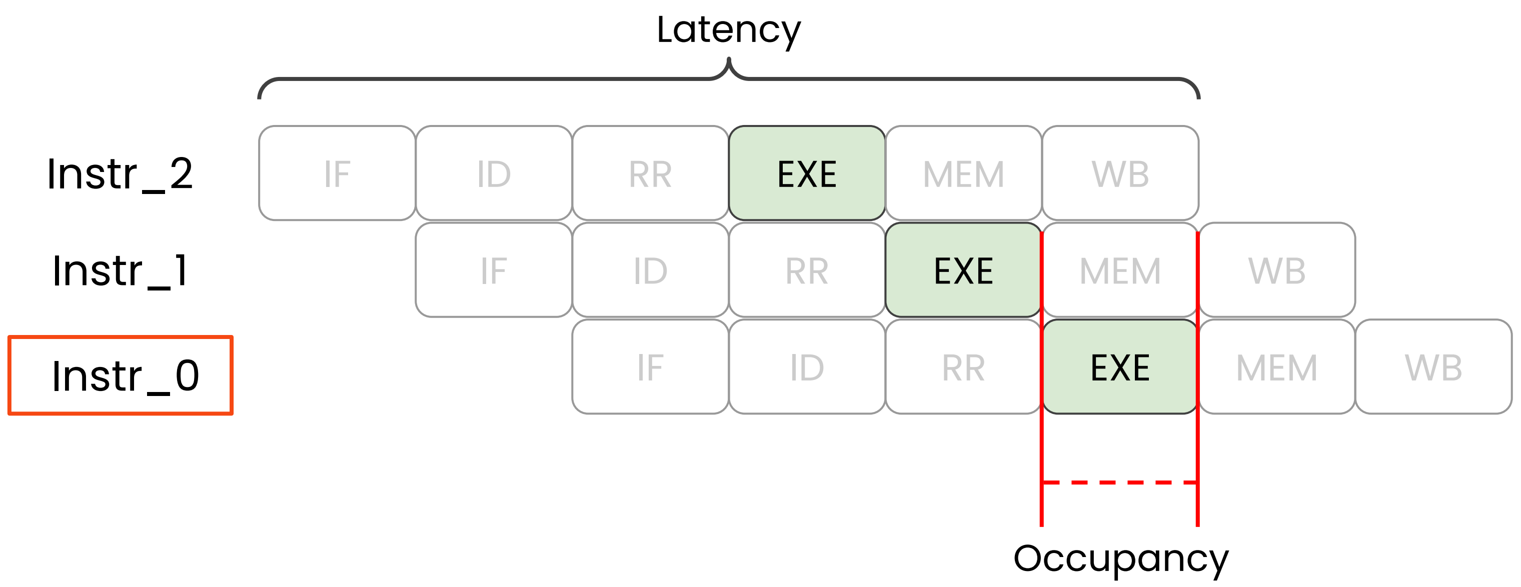
Instruction Stages: From a Hardware Perspective



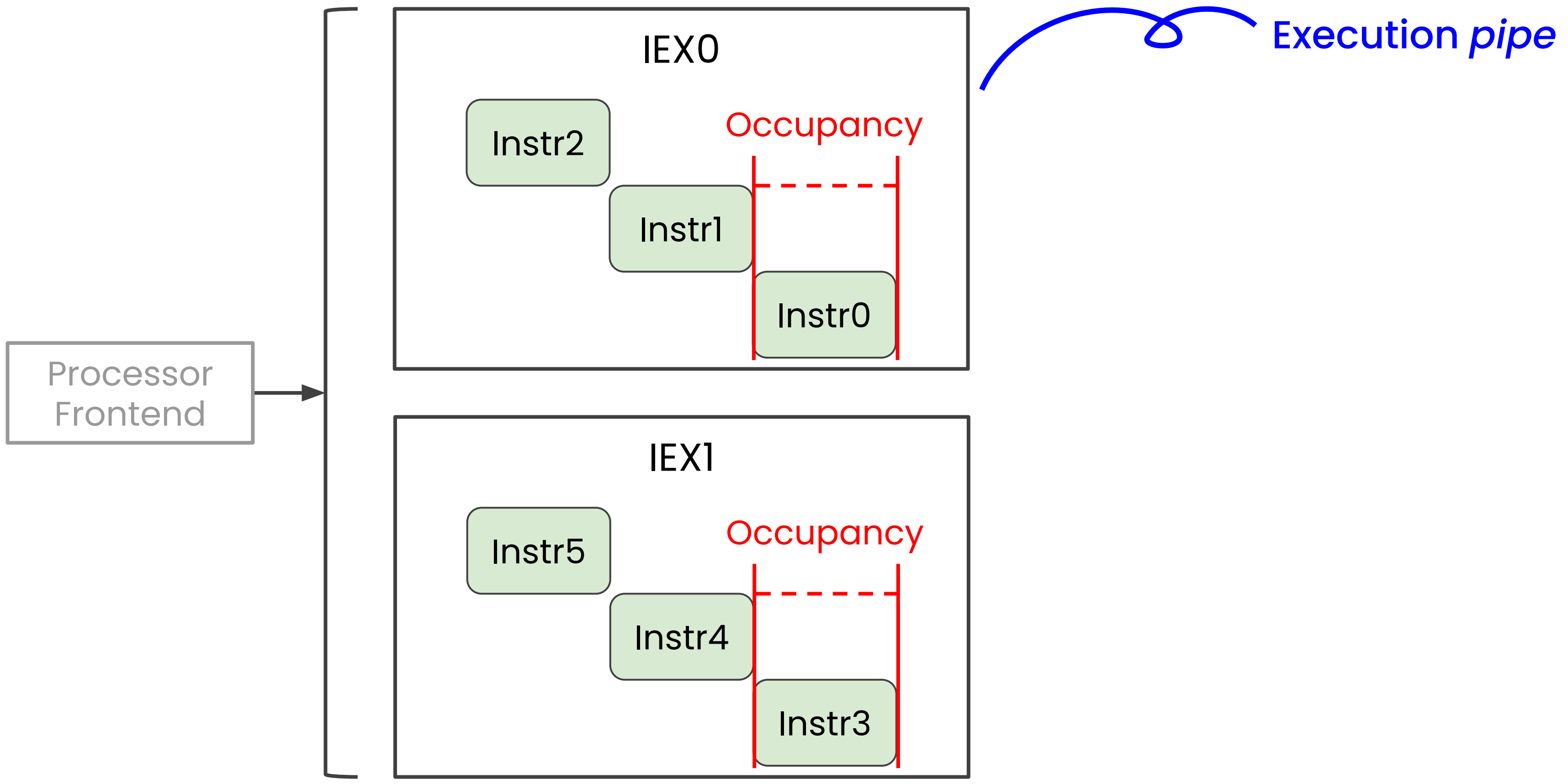
- IF: Instruction Fetch
- ID: Instruction Decode
- RR: Register Read
- EXE: Execution
- MEM: Memory
- WB: Write Back

Instruction Stages: From a Hardware Perspective

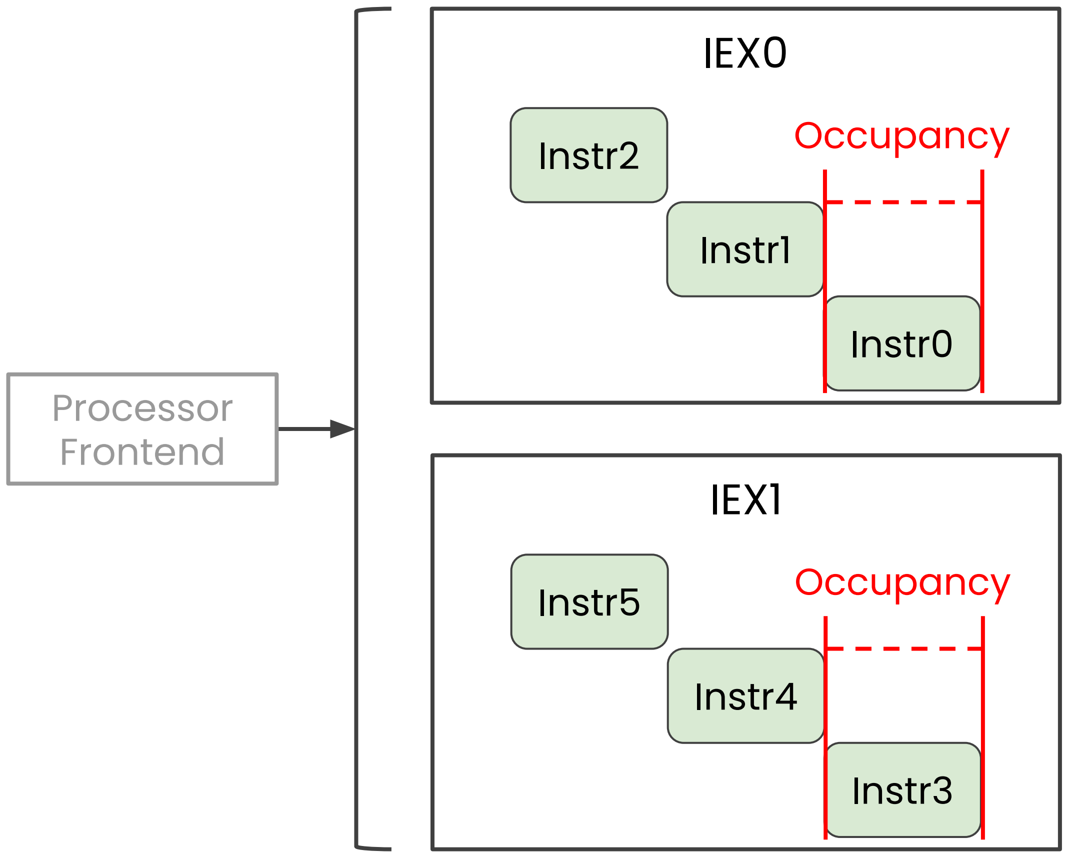
- IF: Instruction Fetch
- ID: Instruction Decode
- RR: Register Read
- EXE: Execution
- MEM: Memory
- WB: Write Back



Instruction Stages with Superscalar Processors



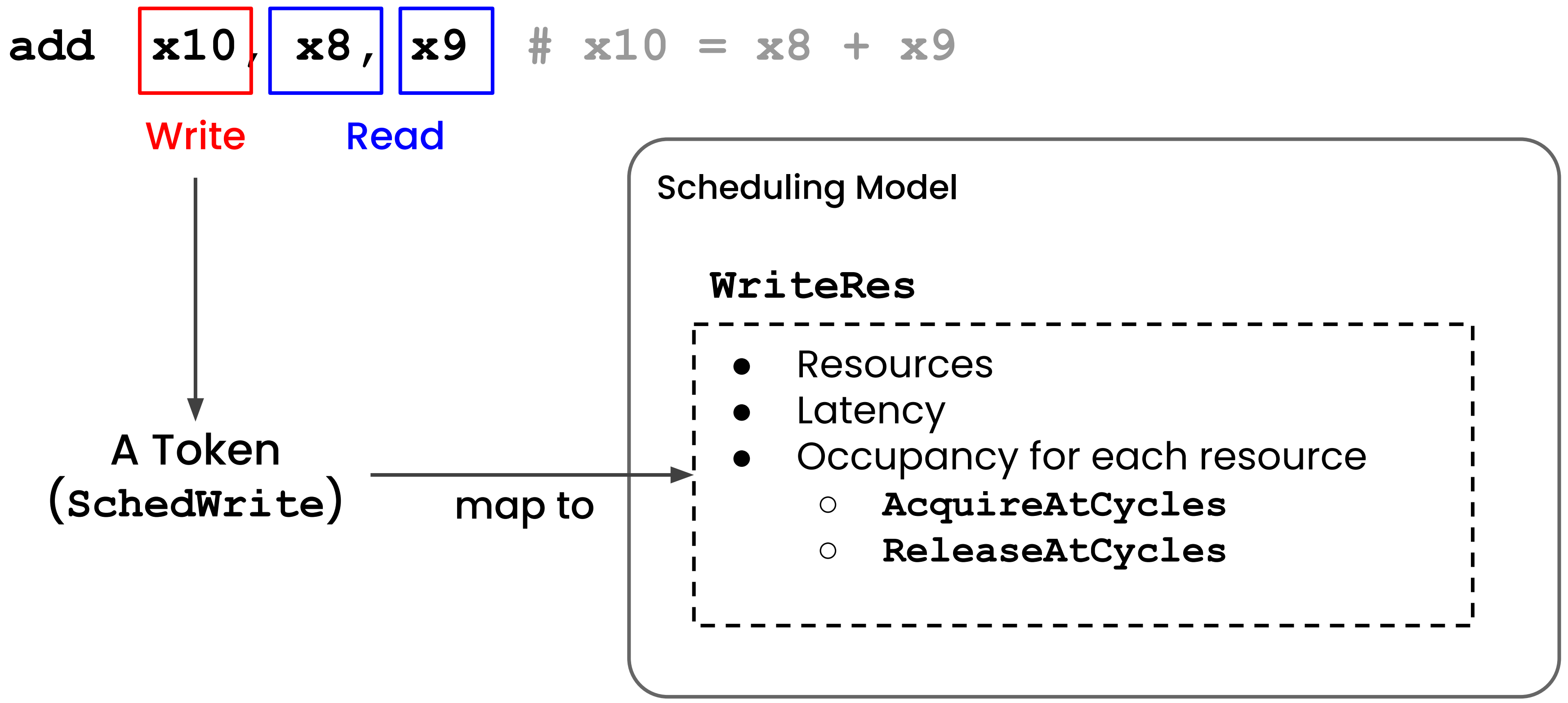
Instruction Stages with Superscalar Processors



	Resource	Occupancy
Instr0	IEX0	2
Instr1	IEX0	2
Instr2	IEX0, IEX1	1
Instr3	IEX0, IEX1	1
Instr4	IEX1	3
Instr5	IEX0, IEX1	1

Instruction Scheduling Model

Contemporary scheduling model framework



```
add  x10, x8, x9  # x10 = x8 + x9
```

- Resources
- Latency
- Occupancy for each resource
  - **`AcquireAtCycles`**
  - **`ReleaseAtCycles`**

Instruction Scheduling Model

An Example

```
// In RISCVInstrInfo.td
def ADD   : ALU_rr<0b0000000, 0b000, "add", Commutable=1>,
            Sched<[WriteIALU, ReadIALU, ReadIALU]>;


// In RISCVSchedSiFive7.td*
def : WriteRes<WriteIALU, [PipeAB]> {
  let Latency = 3;
  let AcquireAtCycles = [0];
  let ReleaseAtCycles = [1];
}
```

Resource

Occupancy

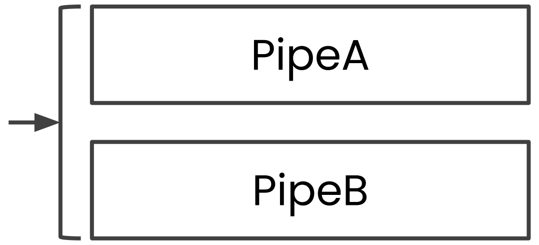
* The code here has been slightly modified s.t. related fields are *explicitly* spelled out

Instruction Scheduling Model

Processor resources

```
// In RISCVSchedSiFive7.td*
def PipeA  : ProcResource<1>;
def PipeB  : ProcResource<1>;
def PipeAB : ProcResGroup<[PipeA, PipeB]>;

def : WriteRes<WriteIALU, [PipeAB]> {
  let Latency = 3;
  let AcquireAtCycles = [0];
  let ReleaseAtCycles = [1];
}
```

PipeA *or* PipeB



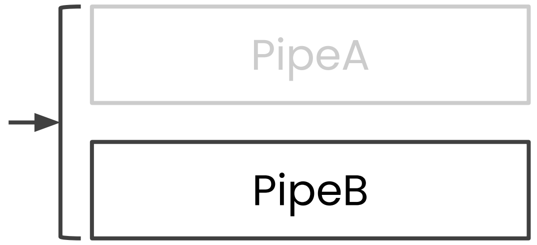
* The code here has been slightly modified

Instruction Scheduling Model

Processor resources

```
// In RISCVSchedSiFive7.td*
def PipeA  : ProcResource<1>;
def PipeB  : ProcResource<1>;
def PipeAB : ProcResGroup<[PipeA, PipeB]>;

def : WriteRes<WriteIALU, [PipeAB]>;
def : WriteRes<WriteIMul, [PipeB]>;
```

Can only run on PipeB

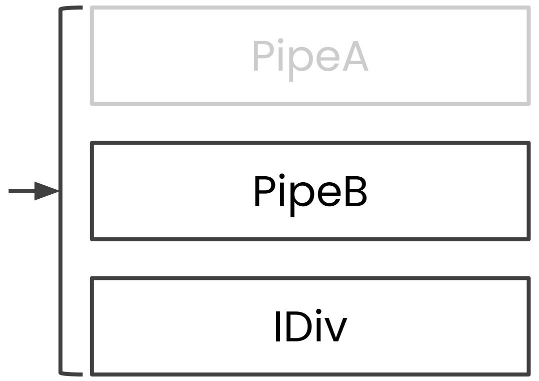


* The code here has been slightly modified

Instruction Scheduling Model

Processor resources

```
// In RISCVSchedSiFive7.td*
def PipeA  : ProcResource<1>;
def PipeB  : ProcResource<1>;
def PipeAB : ProcResGroup<[PipeA, PipeB]>;


def : WriteRes<WriteIALU, [PipeAB]>;
def : WriteRes<WriteIMul, [PipeB]>;
def : WriteRes<WriteIDiv, [PipeB, IDiv]>
```



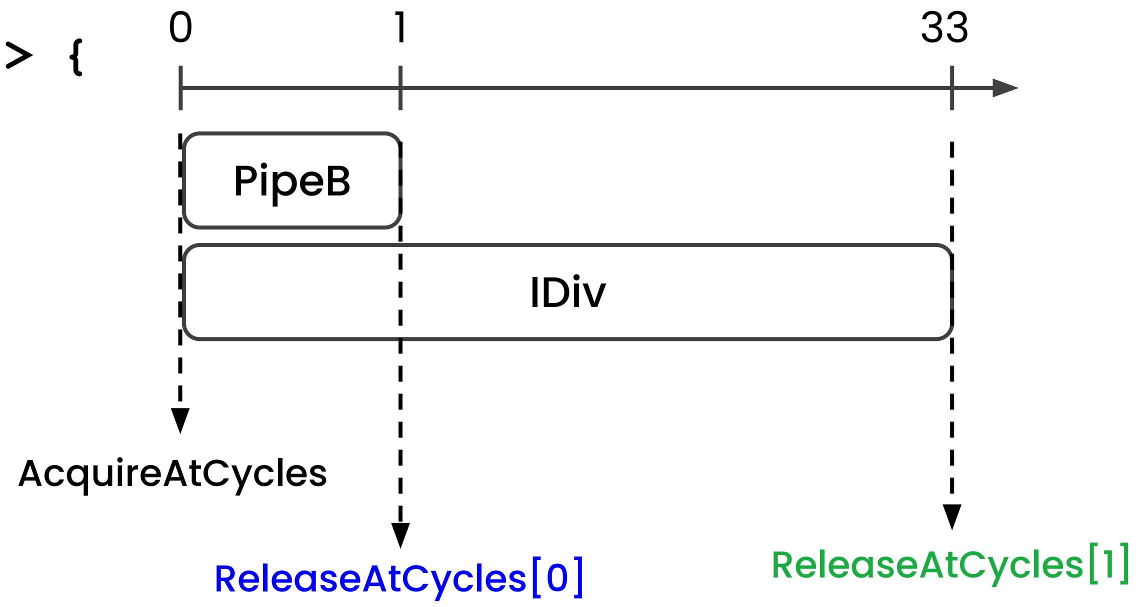
Need PipeB *and* IDiv

* The code here has been slightly modified

Instruction Scheduling Model

Occupancy: Acquire/ReleaseAtCycles

```
def : WriteRes<WriteIDiv, [PipeB, IDiv]> {
  let Latency = 34;
  let AcquireAtCycles = [0, 0];
  let ReleaseAtCycles = [1, 33];
}
```

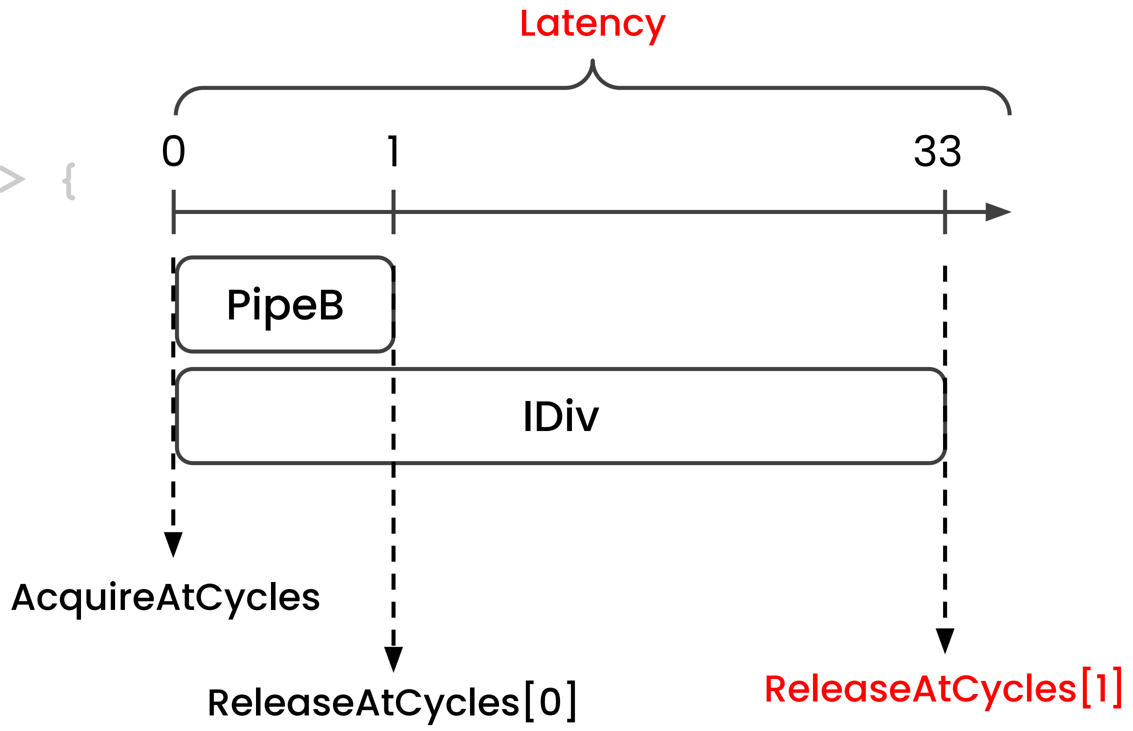


* The code here has been slightly modified

Instruction Scheduling Model

Unpipelined instructions

```
def : WriteRes<WriteIDiv, [PipeB, IDiv]> {
  let Latency = 34;
  let AcquireAtCycles = [0, 0];
  let ReleaseAtCycles = [1, 33];
}
```

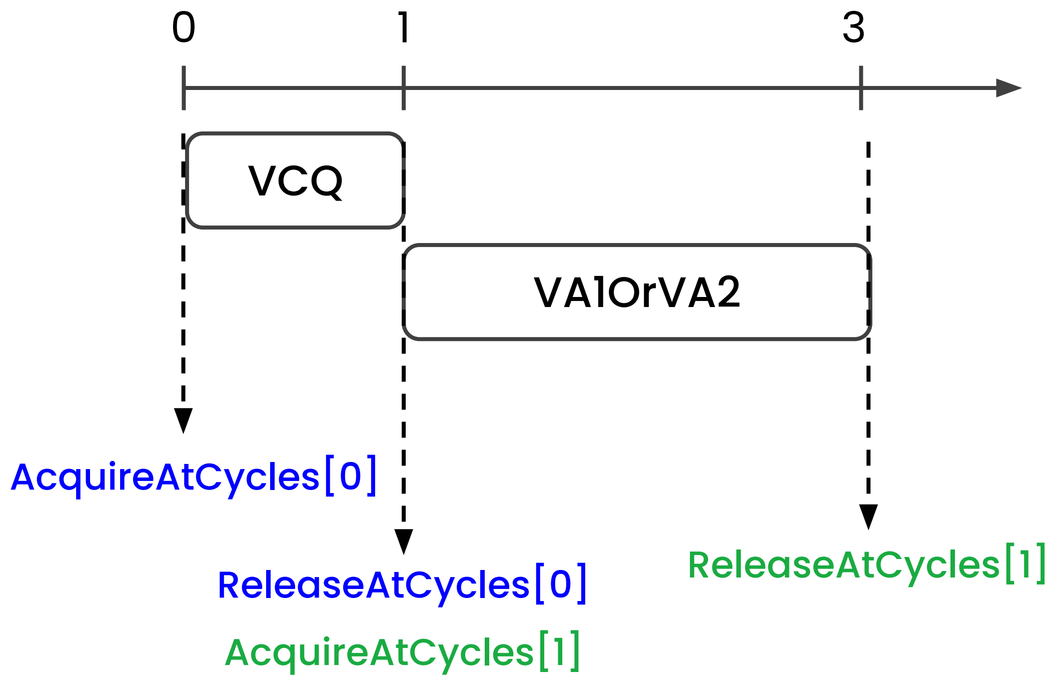


* The code here has been slightly modified

Instruction Scheduling Model

Occupancy: Acquire/ReleaseAtCycles

```
def : WriteRes<WriteVIALUV_M1,
               [VCQ, VA1OrVA2]> {
  let AcquireAtCycles = [0, 1];
  let ReleaseAtCycles = [1, 3];
}
```



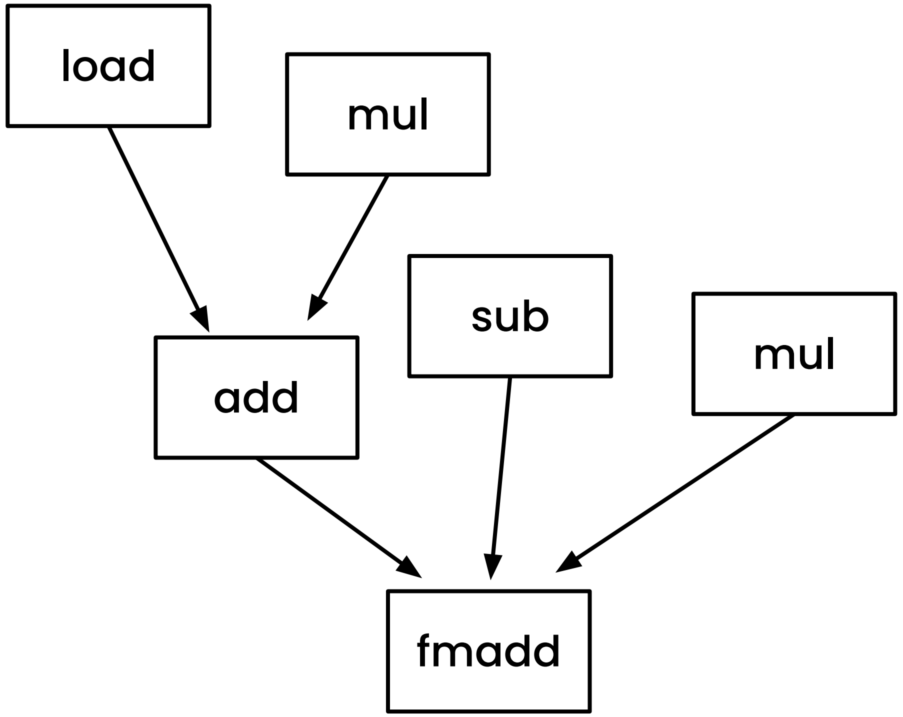
To Learn More:
https://www.youtube.com/watch?v=XWBVLcdzmFg

* The code here has been slightly modified

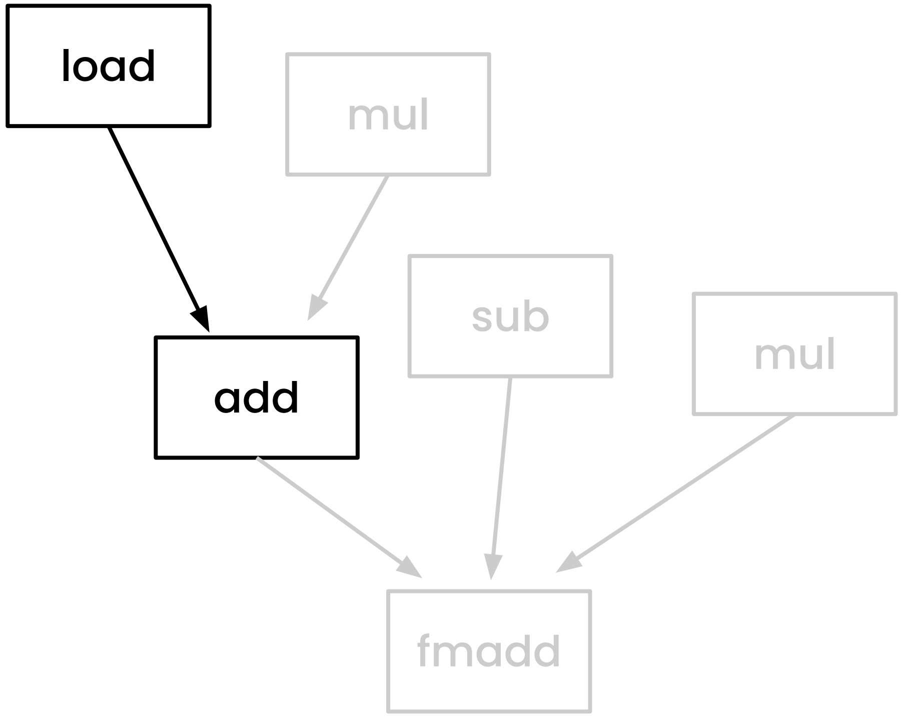
SiFive

Instruction	Latency	Resource	Occupancy
load	10	LSU	2
mul	4	ALU0	2
sub	3	ALU1	1

```
def : WriteRes<WriteIMul, [ALU0]> {
  let Latency = 4;
  let AcquireAtCycles = [0];
  let ReleaseAtCycles = [2];
}
```

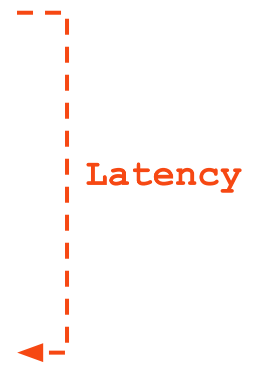


Instruction	Latency	Resource	Occupancy
load	10	LSU	2
mul	4	ALU0	2
sub	3	ALU1	1



```
def : WriteRes<WriteIMul, [ALU0]> {
  let Latency = 4;
  let AcquireAtCycles = [0];
  let ReleaseAtCycles = [2];
}
```

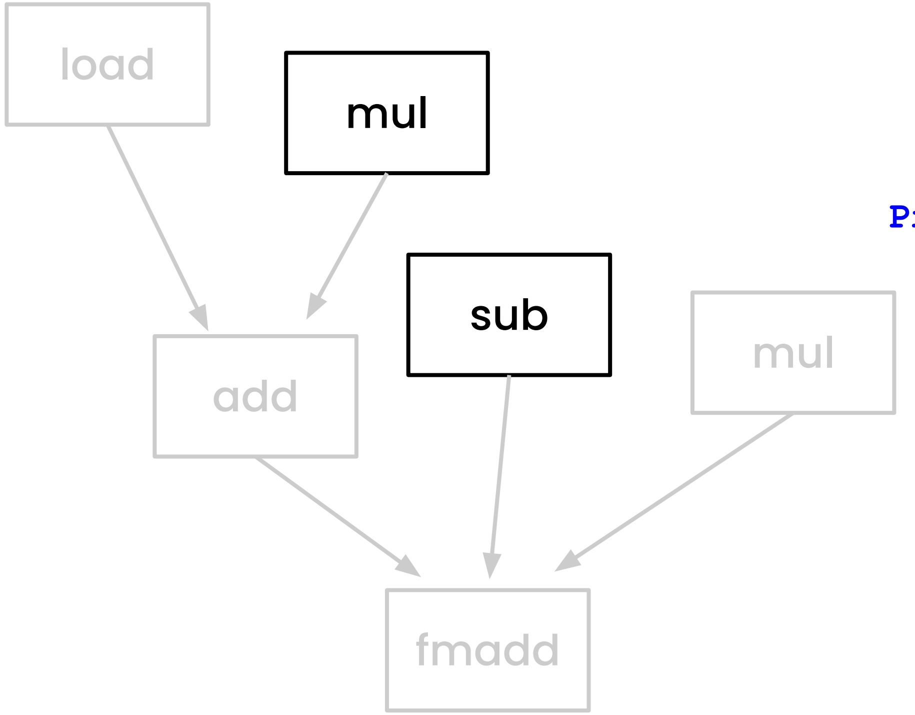
```
ld  x8, 0(x8)


add x12, x9, x8
```



Avoid **Data Hazard**

Instruction	Latency	Resource	Occupancy
load	10	LSU	2
mul	4	ALU0	2
sub	3	ALU1	1

```
def : WriteRes<WriteIMul, [ALU0]> {
  let Latency = 4;
  let AcquireAtCycles = [0];
  let ReleaseAtCycles = [2];
}
```

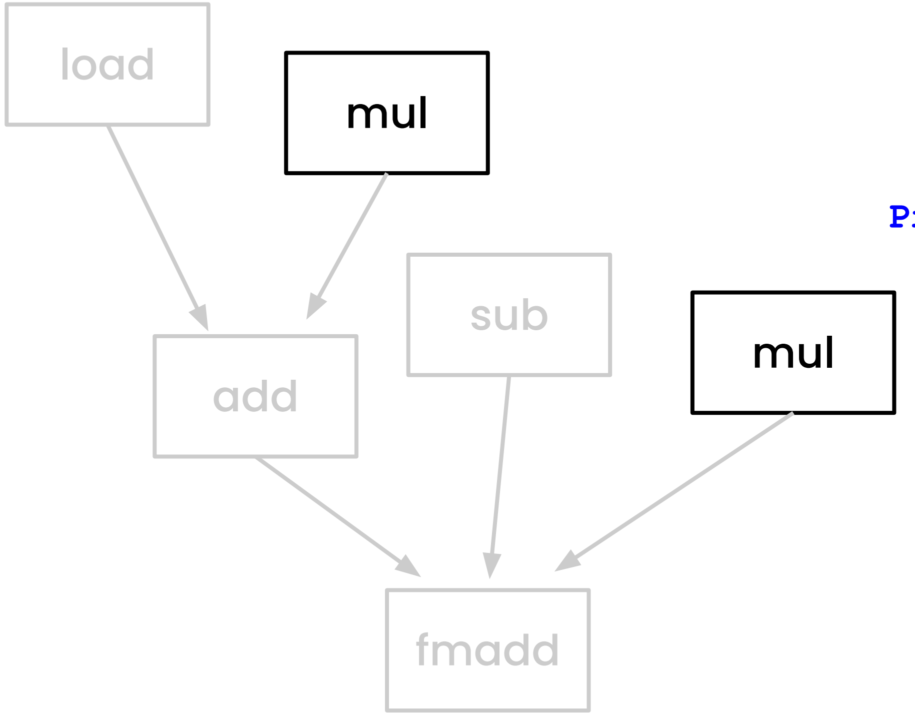


ProcResources

```
ld  x8, 0(x8)
mul x9, x9, x7
sub x10, x10, x7

add x12, x9, x8
```

Instruction	Latency	Resource	Occupancy
load	10	LSU	2
mul	4	ALU0	2
sub	3	ALU1	1

```
def : WriteRes<WriteIMul, [ALU0]> {
  let Latency = 4;
  let AcquireAtCycles = [0];
  let ReleaseAtCycles = [2];
}
```



```
ld  x8, 0(x8)
mul x9, x9, x7
sub x10, x10, x7
mul x11, x11, x7
add x12, x9, x8
```

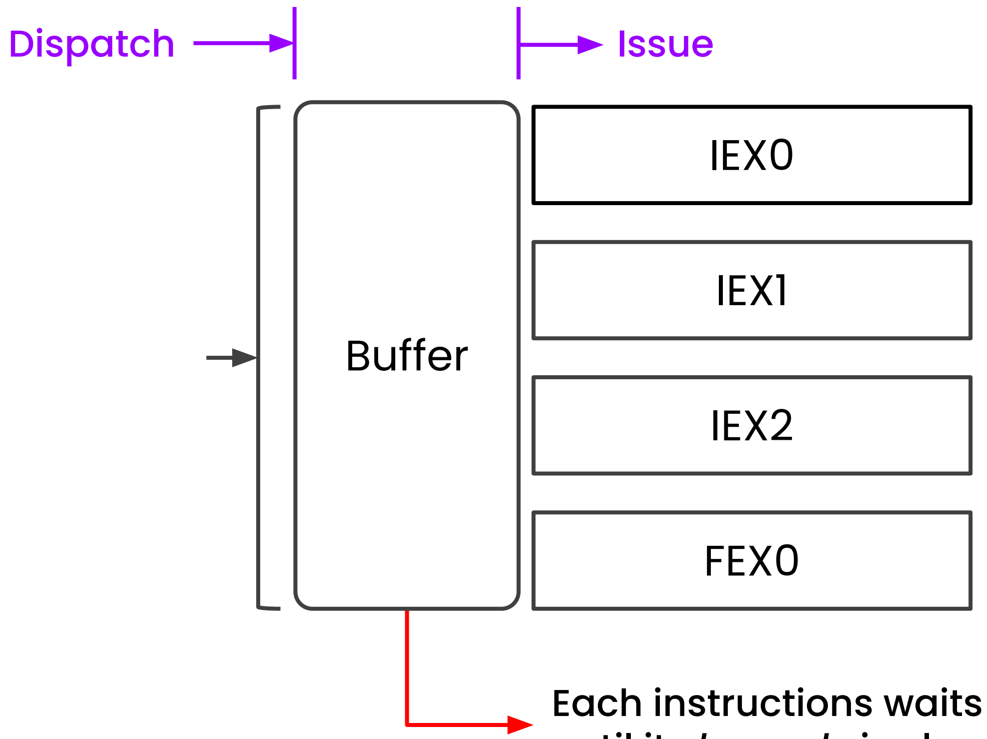
ProcResources

AcquireAtCycles
ReleaseAtCycles

Avoid **Structural Hazard**

Out-of-Order Execution

Hardware solution for avoiding hazards



Each instructions waits in the buffer until its *hazards* is cleared

Out-of-Order Execution

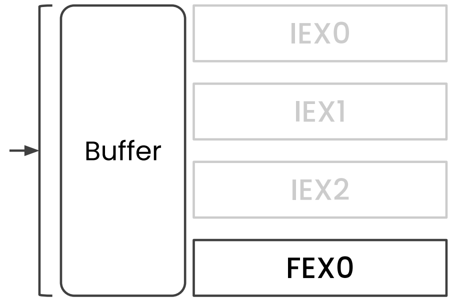
Case study with llvm-mca: *data* hazard

```
$ llvm-mca -mtriple=riscv64 -mcpu=sifive-p470 -timeline

Index     0123456789

[0,0]     DeeeeER   .    fmul.s   ft0, ft0, ft1
[0,1]     D====eeeeER    fmul.s   ft2, ft2, ft0
[0,2]     D=eeeeE---R    fmul.s   ft3, ft3, ft4
```

Cycles

Simulation timeline

- **D**: dispatch
- **=**: stall
- **e**: executing
- **E**: end execution
- **R**: retire

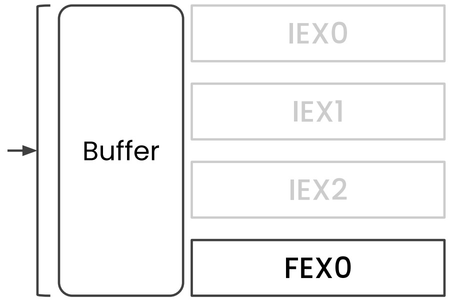


Instruction	Resource
fmul.s	FEX0

Out-of-Order Execution

Case study with llvm-mca: *data* hazard

```
$ llvm-mca -mtriple=riscv64 -mcpu=sifive-p470 -timeline

Index      0123456789

[0,0]      DeeeeER   .    fmul.s   ft0, ft0, ft1
[0,1]      D====eeeeER    fmul.s   ft2, ft2, ft0
[0,2]      D=eeeeE---R    fmul.s   ft3, ft3, ft4
```



Instruction	Resource
fmul.s	FEX0

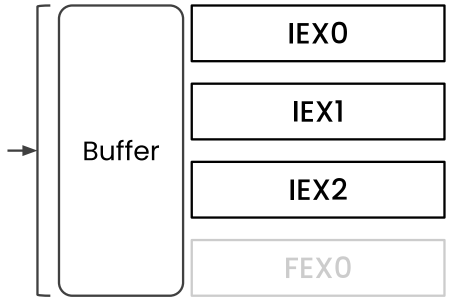
- **D**: dispatch
- **=**: stall
- **e**: executing
- **E**: end execution
- **R**: retire

SiFive

Out-of-Order Execution

Case study with llvm-mca: *structural* hazard

```
$ llvm-mca -mtriple=riscv64 -mcpu=sifive-p470 -timeline

Index     0123456

[0,0]     DeeER..   mul s0, s0, s1
[0,1]     D=eeER.   mul a0, a0, a1
[0,2]     D==eeER   mul a2, a2, a3
[0,3]     .DeE--R   add s9, s9, s10
[0,4]     .DeE--R   add s11, s11, t3
```



Instruction	Resource
add	IEX0, IEX1, IEX2
mul	IEX2

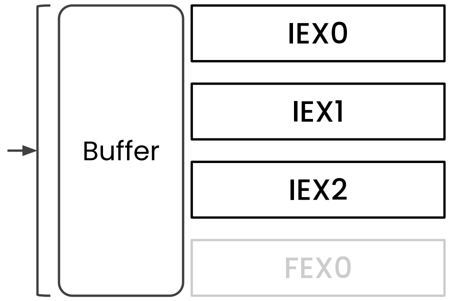
- **D**: dispatch
- **=**: stall
- **e**: executing
- **E**: end execution
- **R**: retire

Out-of-Order Execution

Case study with llvm-mca: *structural* hazard

```
$ llvm-mca -mtriple=riscv64 -mcpu=sifive-p470 -timeline

Index      0123456

[0,0]      DeeER..   mul s0, s0, s1
[0,1]      D=eeER.   mul a0, a0, a1
[0,2]      D==eeER   mul a2, a2, a3
[0,3]      .DeE--R   add s9, s9, s10
[0,4]      .DeE--R   add s11, s11, t3
```

- **D**: dispatch
- **=**: stall
- **e**: executing
- **E**: end execution
- **R**: retire

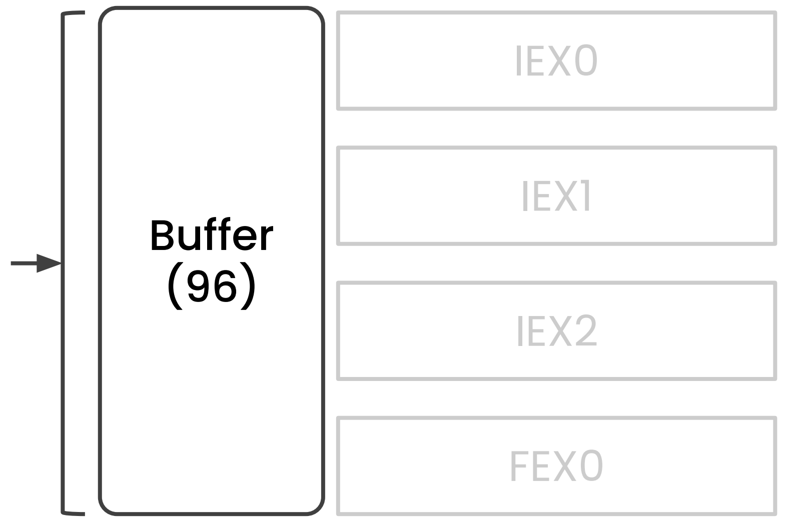


Instruction	Resource
add	IEX0, IEX1, IEX2
mul	IEX2

Scheduling Model: Buffer Sizes

Unified Reservation Station

```
// In RISCVSchedSiFiveP400.td*
def SiFiveP400Model : SchedMachineModel {
  let MicroOpBufferSize = 96;
}

let BufferSize = -1 in {
  def IEXQ0   : ProcResource<1>;
  def IEXQ1   : ProcResource<1>;
  def IEXQ2   : ProcResource<1>;
  def FEXQ0   : ProcResource<1>;
}
```

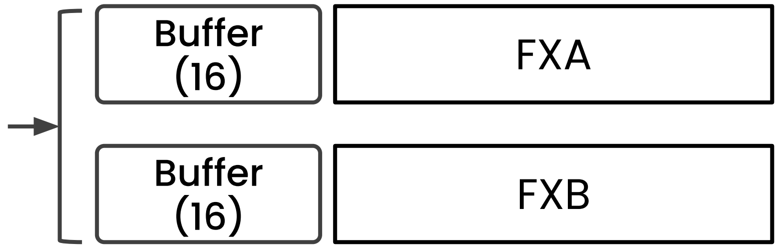


* The code here has been slightly modified

Scheduling Model: Buffer Sizes

Decoupled Reservation Station

```
// In RISCVSchedTTAscalonD8.td*
let BufferSize = 16 in {
  def AscalonFXA : ProcResource<1>;
  def AscalonFXB : ProcResource<1>;
  def AscalonFXC : ProcResource<2>;
  def AscalonFXD : ProcResource<2>;
  def AscalonFP  : ProcResource<2>;
}
```

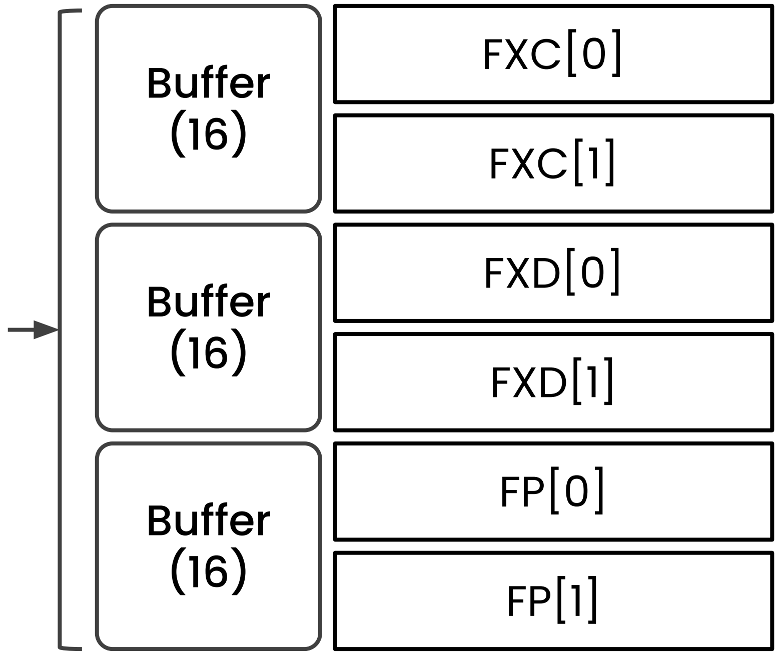


* The code here has been slightly modified

Scheduling Model: Buffer Sizes

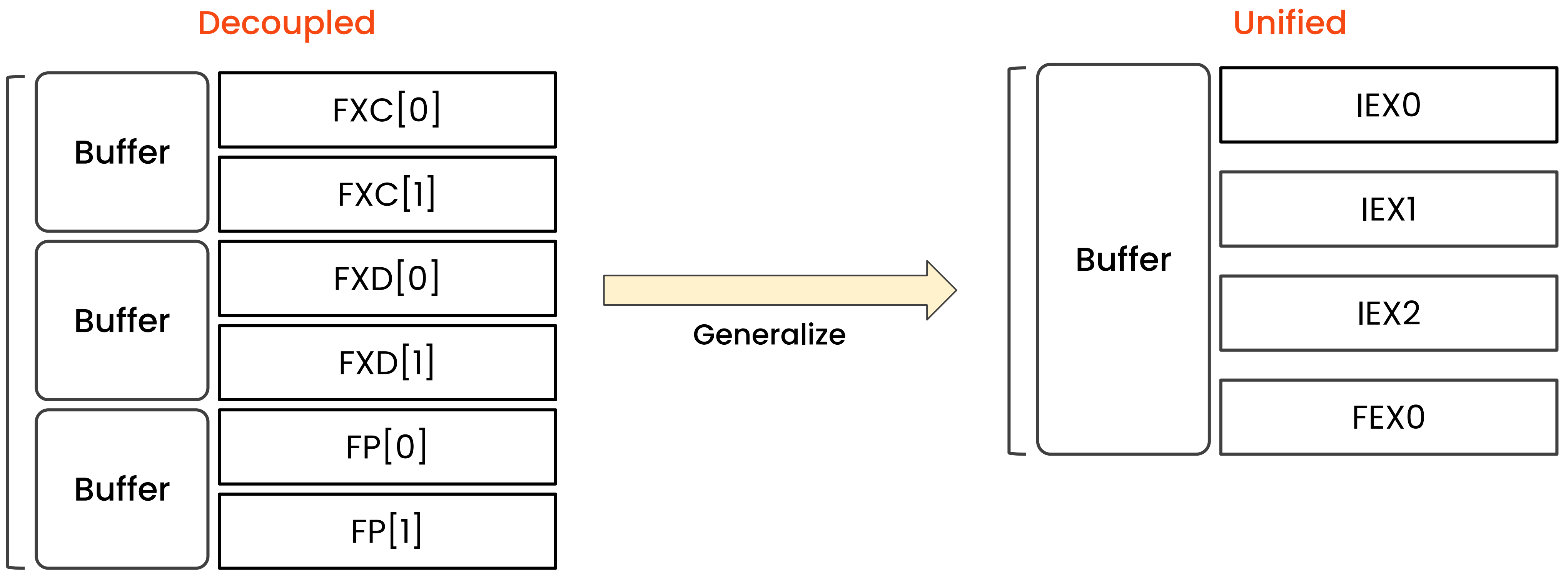
Decoupled Reservation Station

```
// In RISCVSchedTTAscalonD8.td*
let BufferSize = 16 in {
  def AscalonFXA : ProcResource<1>;
  def AscalonFXB : ProcResource<1>;
  def AscalonFXC : ProcResource<2>;
  def AscalonFXD : ProcResource<2>;
  def AscalonFP  : ProcResource<2>;
}
```



* The code here has been slightly modified

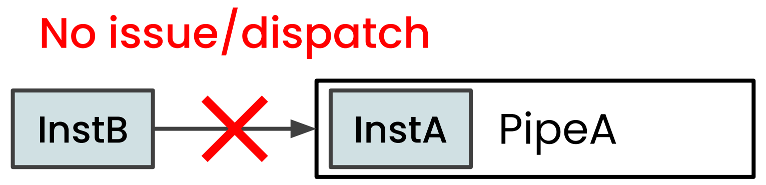
Scheduling Model: Buffer Sizes



Scheduling Model: Buffer Sizes

In-order cores

```
// In RISCVSchedSiFive7.td*
let BufferSize = 0 in {
  def PipeA  : ProcResource<1>;
  def PipeB  : ProcResource<1>;
}
def PipeAB : ProcResGroup<[PipeA, PipeB]>;
```

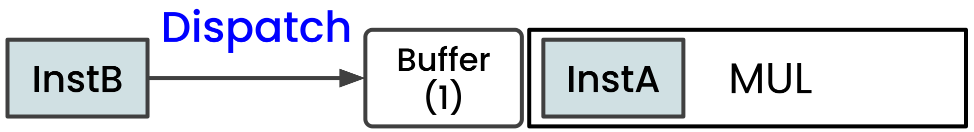


* The code here has been slightly modified

Scheduling Model: Buffer Sizes

In-order cores

```
// In RISCVSchedSiFive7.td*
let BufferSize = 0 in {
  def PipeA  : ProcResource<1>;
  def PipeB  : ProcResource<1>;
}
def PipeAB : ProcResGroup<[PipeA, PipeB]>;
```



```
// In RISCVSchedSyntacoreSCR7.td*
def ALU_MUL_IS : ProcResource<1> { let BufferSize = 8; }
def ALU_DIV_IS : ProcResource<1> { let BufferSize = 8; }

def MUL : ProcResource<1> { let BufferSize = 1; }
def DIV : ProcResource<1> { let BufferSize = 1; }
```

* The code here has been slightly modified

Scheduling Model: Buffer Sizes

In-order cores

BufferSize = 0

```
                    0123456789          0123456789          0123456789
Index     0123456789          0123456789          0123456789          01

[0,0]     DeeeeeeeeeeeeeeeeeeeER   .    .    .    .    .    .    .    ..   divw s0, s1, s1
[0,1]     .    .    .    .    DeeeeeeeeeeeeeeeeeeeER   .    .    .    ..   divw a0, a1, a1
[0,2]     .    .    .    .    .    .    .    .    DeeeeeeeeeeeeeeeeeeeER   divw a2, a3, a3
```

dispatch(D) after the first instruction *releases* the pipe

- **D**: dispatch
- **=**: stall
- **e**: executing
- **E**: end execution
- **R**: retire

Scheduling Model: Buffer Sizes

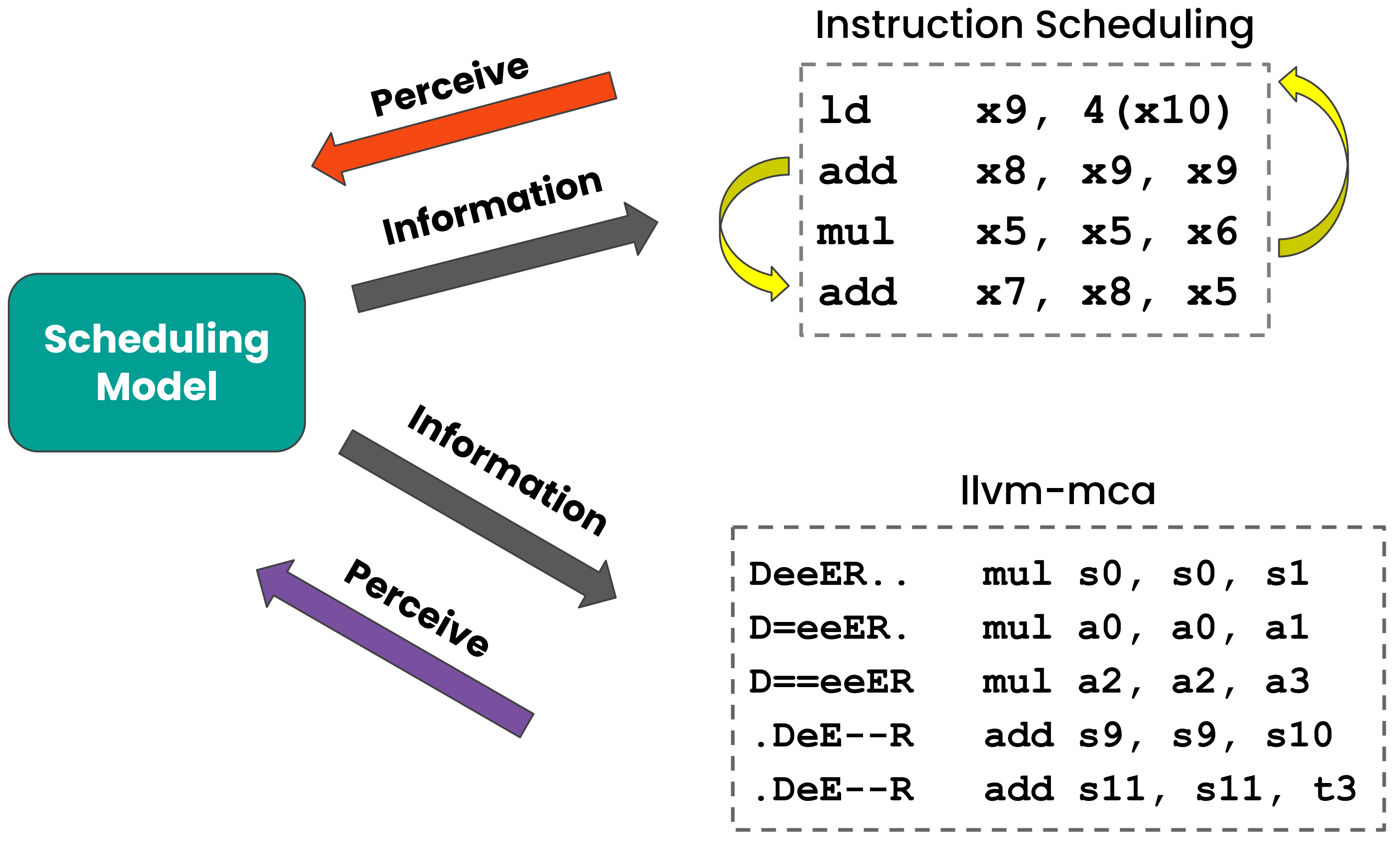
In-order cores

BufferSize = 1

```
$ llvm-mca -mtriple=riscv64 -mcpu=syntacore-scr7 -timeline

                    0123456789          0123456789          0123456789
Index     0123456789          0123456789          0123456789

[0,0]     DeeeeeeeeeeeeeeeeeeeER   .    .    .    .    .    .    .   .   divw   s0, s1, s1
[0,1]     .D==================eeeeeeeeeeeeeeeeeeeER    .    .    .   .   divw   a0, a1, a1
[0,2]     .    .    .    .    D==================eeeeeeeeeeeeeeeeeeeER   divw   a2, a3, a3
```

Dispatch right after the first instruction was *issued*, wait inside the buffer

- **D**: dispatch
- **=**: stall
- **e**: executing
- **E**: end execution
- **R**: retire

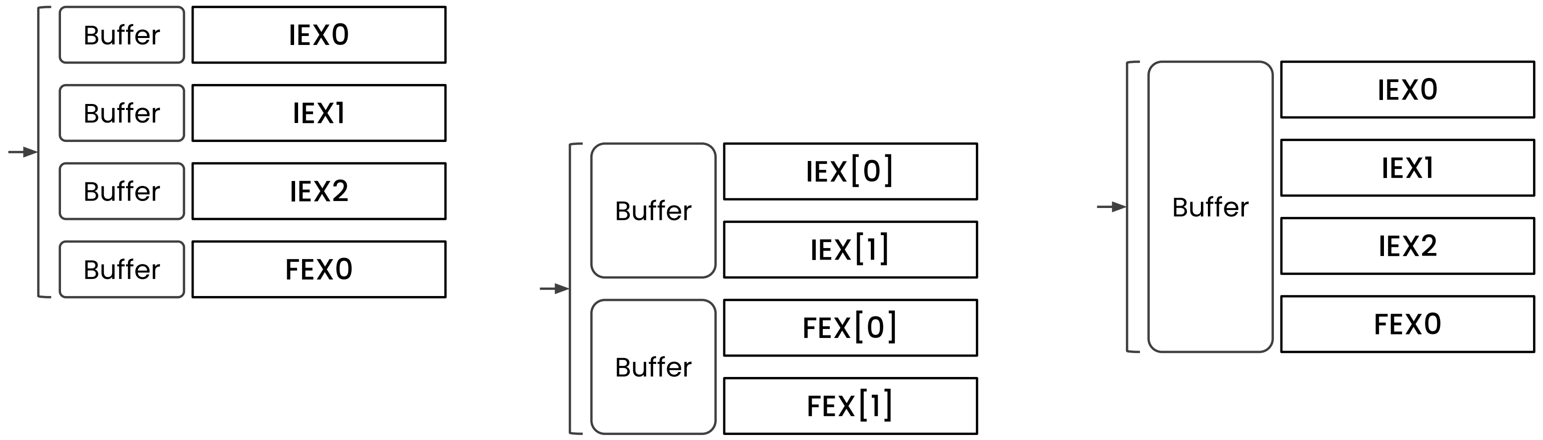


```
ld      x9, 4(x10)
add     x8, x9, x9
mul     x5, x5, x6
add     x7, x8, x5
```

```
DeeER..    mul s0, s0, s1
D=eeER.    mul a0, a0, a1
D==eeER    mul a2, a2, a3
.DeE--R    add s9, s9, s10
.DeE--R    add s11, s11, t3
```

Q: How do you leverage different OoO BufferSizes?

llvm-mca:



Machine Scheduler: *“That’s the neat part, WE DON’T”*

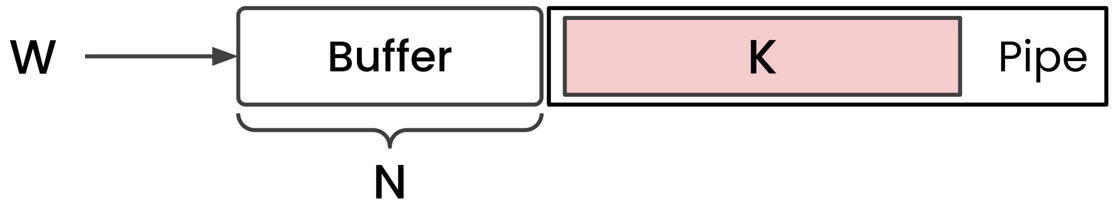
*OoO: Out-of-Order

Machine Scheduler: Out-of-Order Cores

- Does the *minimal* amount of efforts to predict hazards
    - Rationale: Nearly **impossible** to predict during compile-time

Machine Scheduler: Out-of-Order Cores

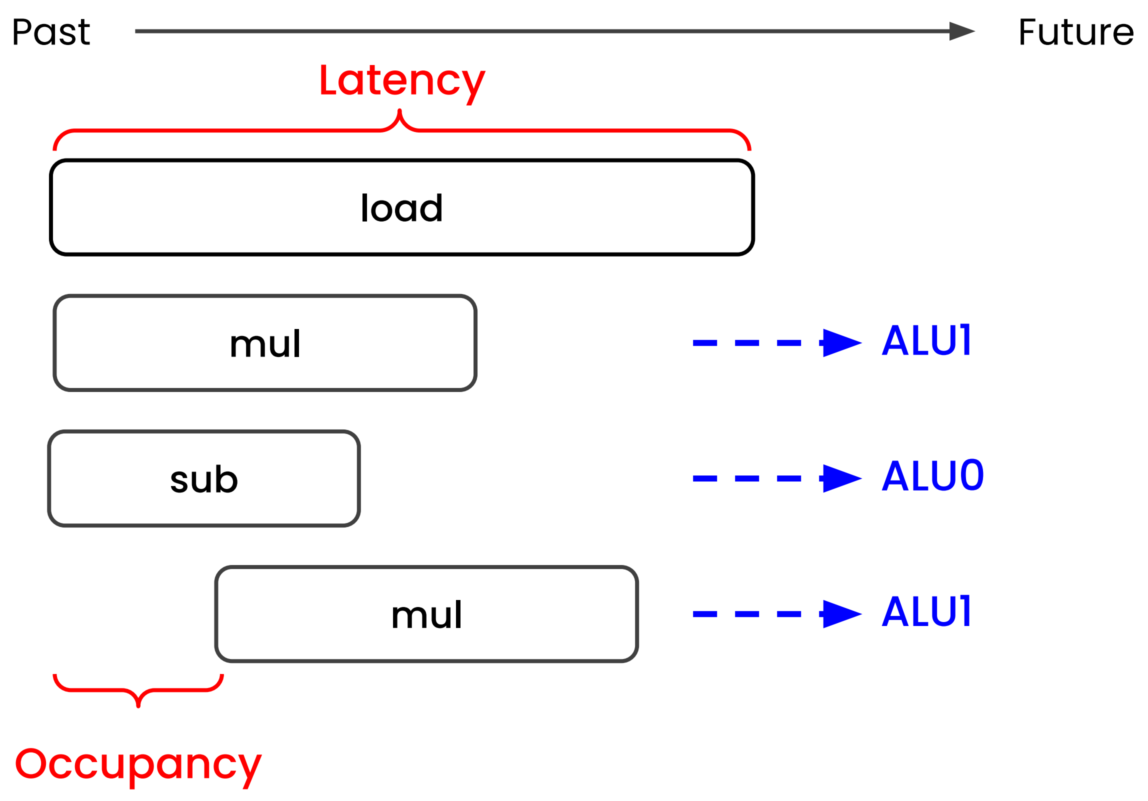
- Does the *minimal* amount of efforts to predict hazards
  - Rationale: Nearly **impossible** to predict during compile-time
- The condition where a buffer might be **full**: `N < W*K - 1`
  - `N`: buffer size
  - `W`: number of instructions that go into this buffer per cycle
  - `K`: *largest* occupancy that can execute in this pipe



Machine Scheduler: In-Order Cores

MicroOpBufferSize & BufferSize = 0

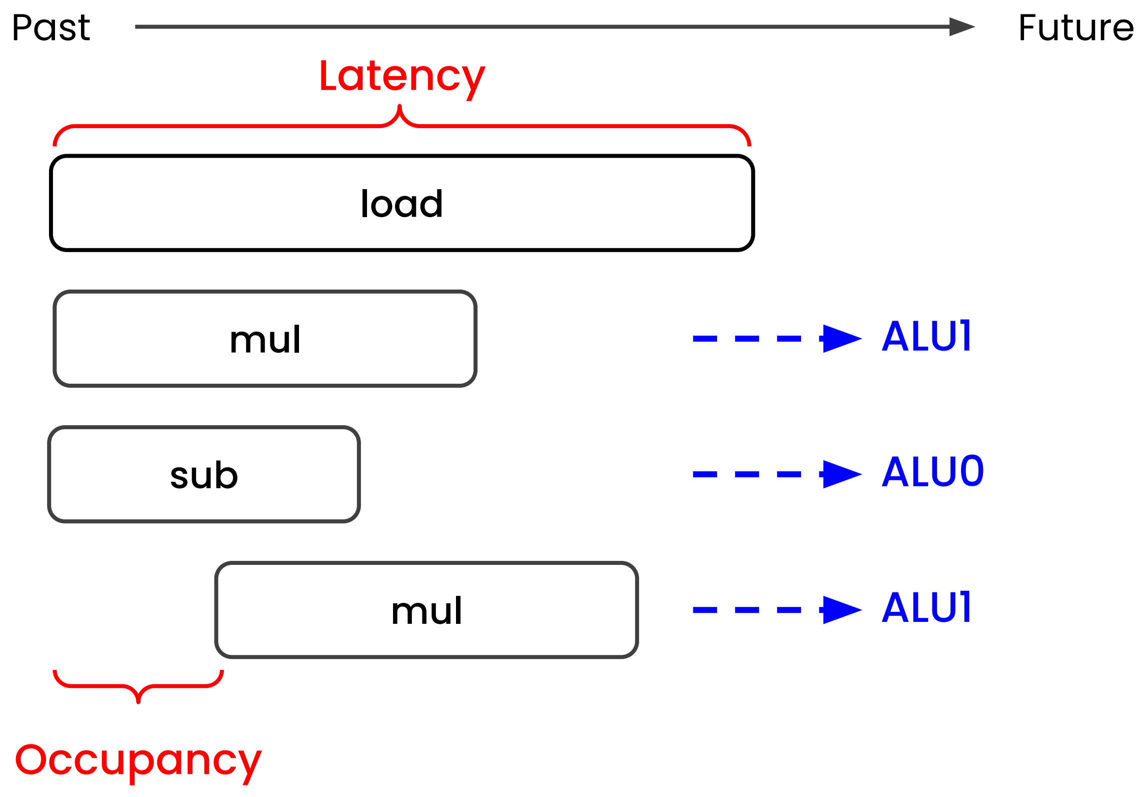
- We can predict stallings more **accurately** during *compile-time*



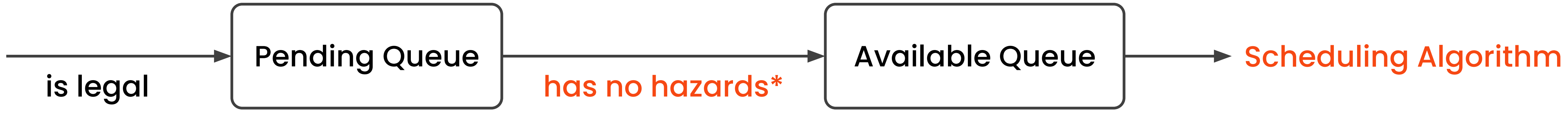
Machine Scheduler: In-Order Cores

MicroOpBufferSize & BufferSize = 0

- We can predict stallings more accurately during *compile-time*
- Machine Scheduler doesn't even *consider* an instruction if it might induce any kind of **hazards**

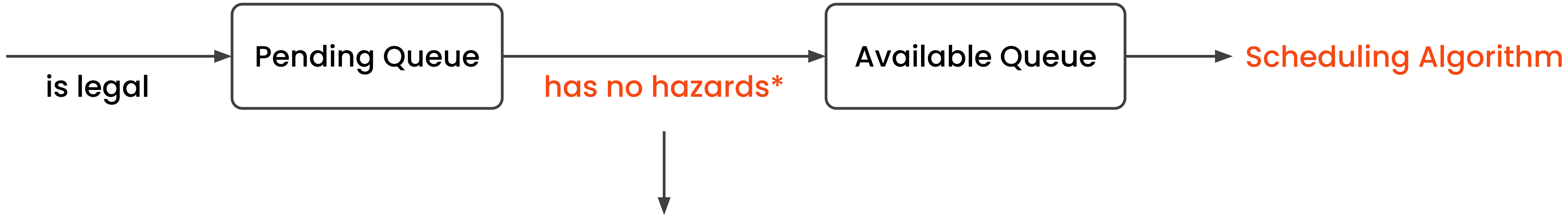


Machine Scheduler: Hazards



* Excluding custom hazard detector

Machine Scheduler: Hazards



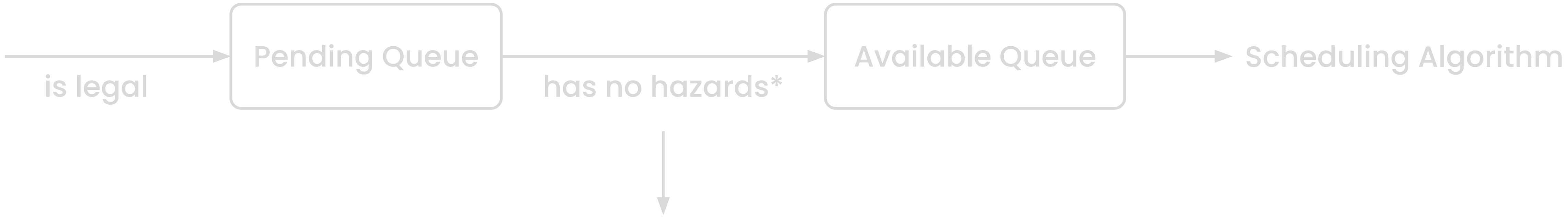
	BufferSize	Data hazard checks	Structural hazard checks	Issue width checks
In-Order	0	Y	Y	Y
Out-of-Order	-1 / larger than 1	N	N	Y

- More **optimistics**
- Scheduling algorithm generally has *more* choices

* Excluding custom hazard detector

Machine Scheduler: BufferSize = 1

In-Order core…scheduled *out-of-order*-ly?

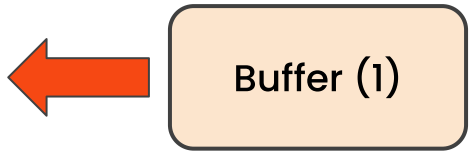


	BufferSize	Data hazard checks	Structural hazard checks	Issue width checks
In-Order	0	Y	Y	Y
	1	N	N	Y
Out-of-Order	-1 / larger than 1	N	N	Y

* Excluding custom hazard detector

Machine Scheduler: The *Duality* of BufferSize = 1

Too **optimistic** on a capability (i.e. out-of-order-ness) that doesn't even exist

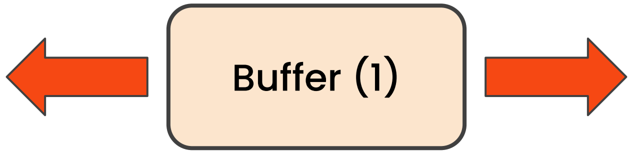


FWIW…

Machine Scheduler does consider *data hazards* when it's scheduling for BufferSize = 1. But not structural hazards

Machine Scheduler: The *Duality* of BufferSize = 1

Too **optimistic** on a capability (i.e. out-of-order-ness) that doesn't even exist



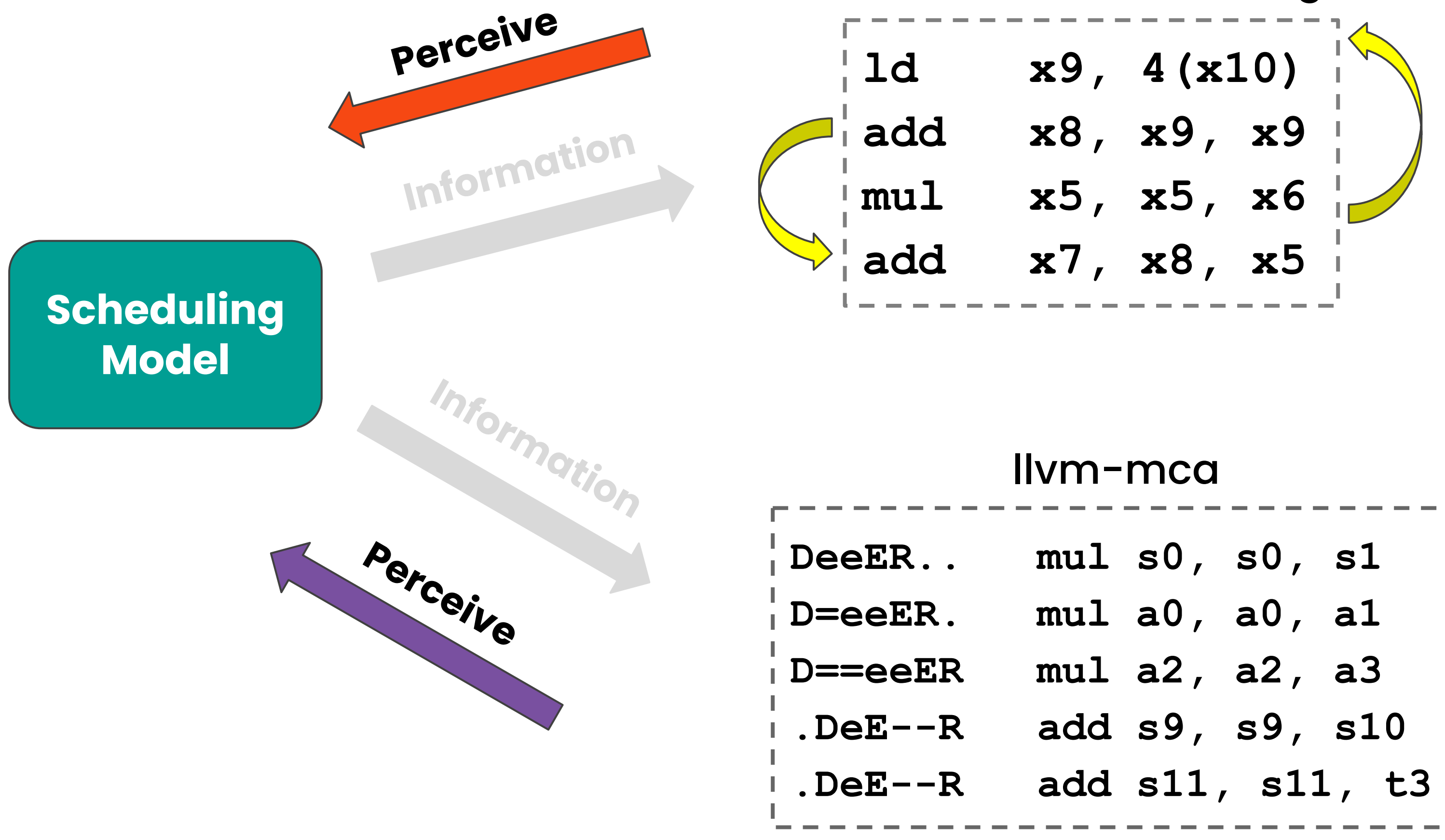
An escape hatch when **in-order** scheduling (i.e. BufferSize = 0) is too *strict* or *pessimistic*

FWIW...

Machine Scheduler does consider *data hazards* when it's scheduling for BufferSize = 1. But not structural hazards

Example:

Sometimes we want to schedule aggressively for **register pressure** even though there is a hazard



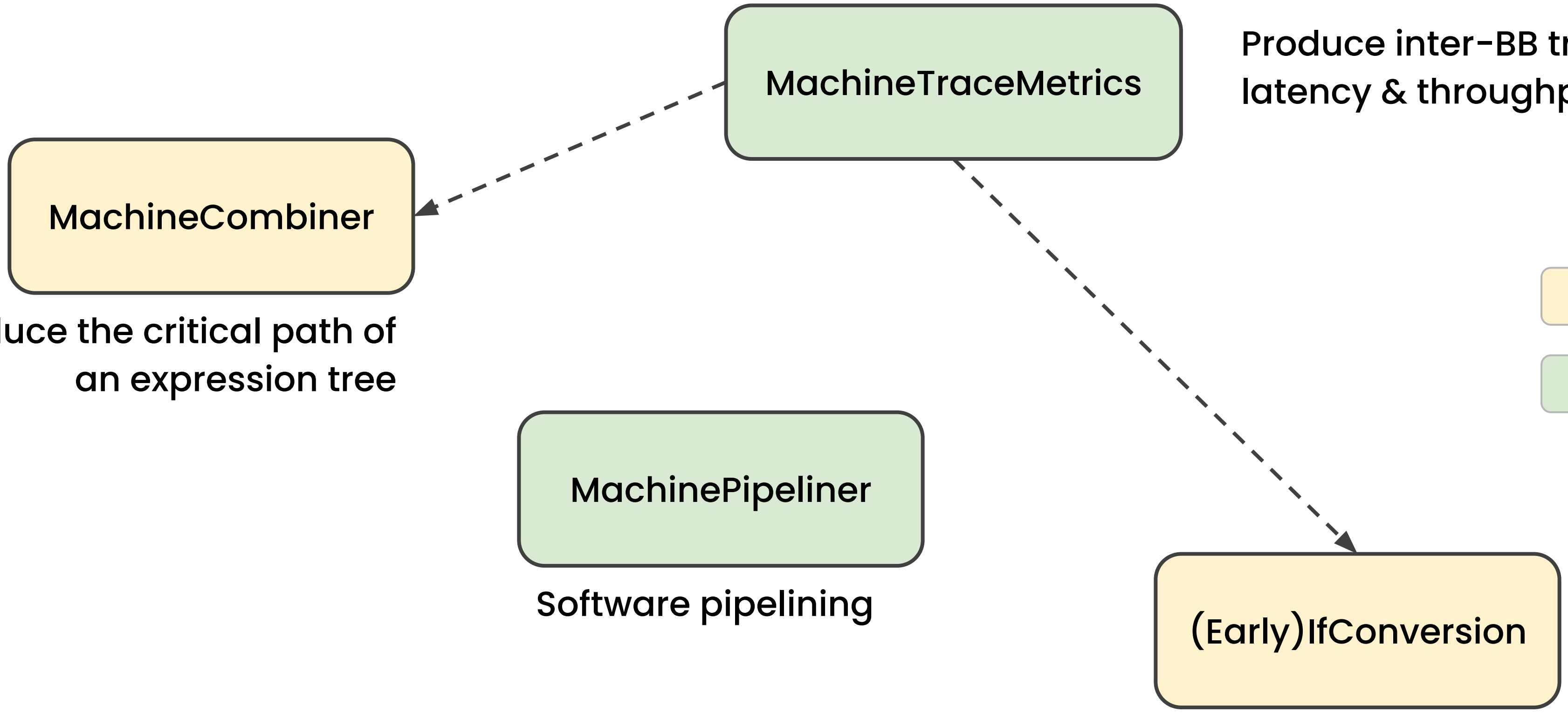
```
ld     x9, 4(x10)
add    x8, x9, x9
mul    x5, x5, x6
add    x7, x8, x5
```

```
DeeER..    mul s0, s0, s1
D=eeER.    mul a0, a0, a1
D==eeER    mul a2, a2, a3
.DeE--R    add s9, s9, s10
.DeE--R    add s11, s11, t3
```

SiFive

Can we use Scheduling Model in More Places?

Status Quo



Produce inter-BB traces with their latency & throughput information

Reduce the critical path of an expression tree

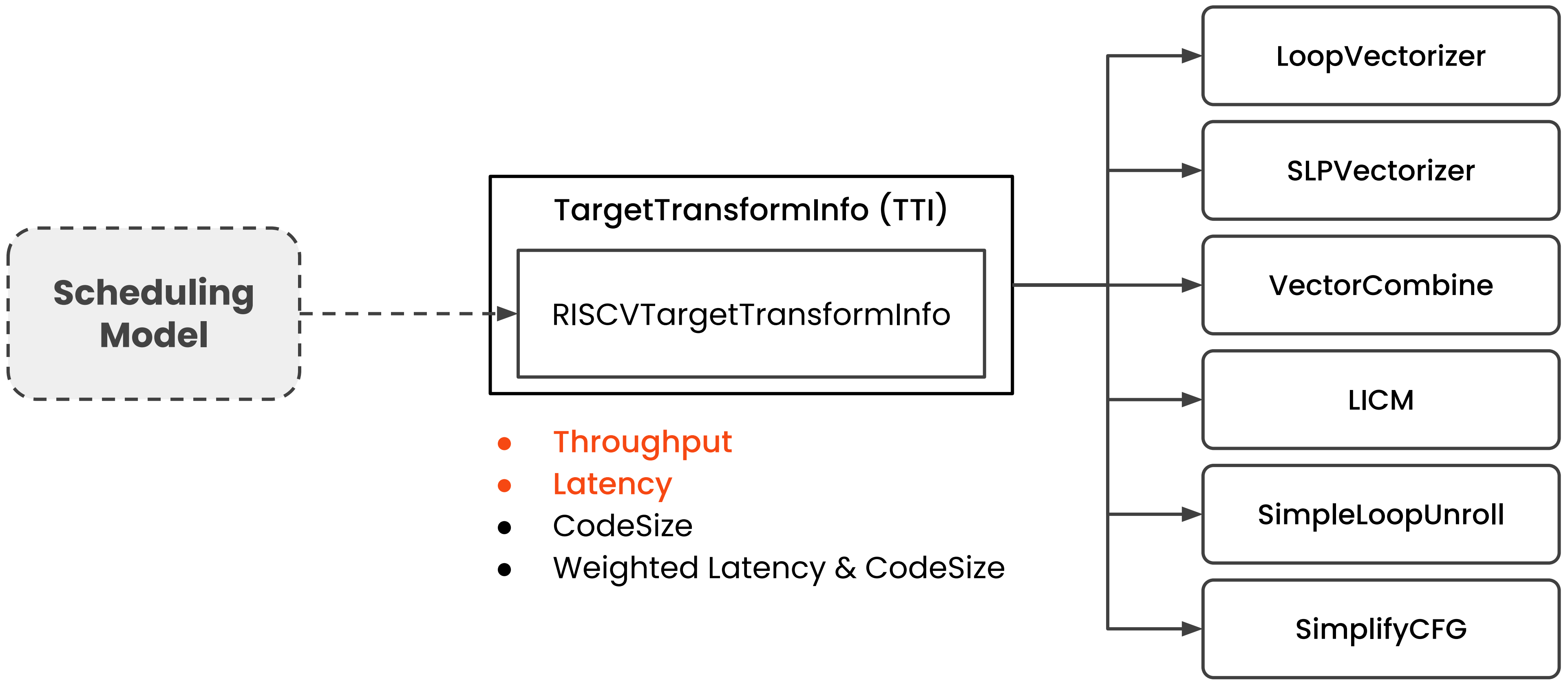
Software pipelining

Turn branches into *predicated* instructions

🟨 Mostly use latency info

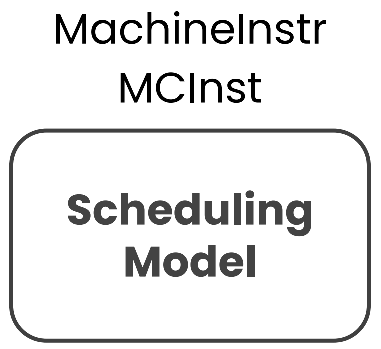
🟩 Latency and occupancy / throughput

Potential User: TTI Cost Model



- Throughput
- Latency
- CodeSize
- Weighted Latency & CodeSize

Potential User: TTI Cost Model

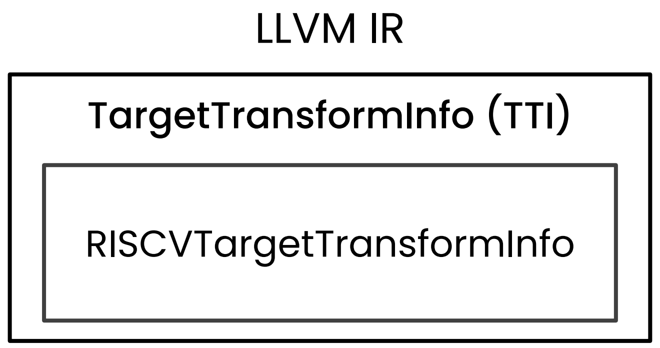


Pros

- Scheduling model can be the *centralized* place to provide such kinds of information

Cons

- Mapping LLVM IR instructions to low-level ones can be (really) challenging
- Necessity: how many *non-trivial* instructions will actually benefit from this?



- Throughput
- Latency
- CodeSize
- Weighted Latency & CodeSize

Other Potential Uses

- Software pipelining "lite": scheduling-model-guided *loop unrolling*
- *Validate* instruction scheduling with LLVM MCA
- Sort **ISel patterns** by latency

```
commit 3622f15491d4ae3b27f3399031933bd888e20cf0
Author: Chris Lattner <sabre@nondot.org>
Date:   Wed Sep 28 17:57:56 2005 +0000

    Prefer cheaper patterns to more expensive ones.
    Print the costs to the generated
    file

    llvm-svn: 23492
```

```
/// Compute the number of instructions for this pattern.
/// This is a temporary hack.  We should really include the instruction
/// latencies in this calculation.
static unsigned getResultPatternCost(const TreePatternNode &P,
                                     const CodeGenDAGPatterns &CGP)
```

Summary

What does LLVM’s Scheduling Model provide

Scheduling Model is all about how it’s **used** and **perceived**

Potential areas we can apply Scheduling Model to

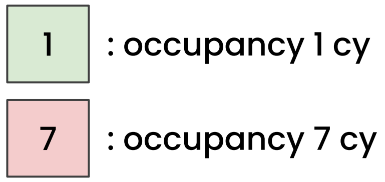
THANK YOU

SiFive

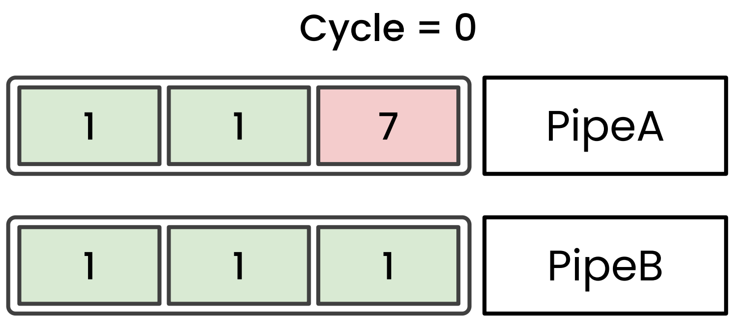
Appendix

Scheduling Model: Buffer Sizes

Buffer per pipe v.s. Pipes sharing the same buffer



```
let BufferSize = 3 in {
  def PipeA : ProcResource<1>;
  def PipeB : ProcResource<1>;
}
def PipeAB : ProcResGroup<[PipeA, PipeB]>;
def : WriteRes<WriteIALU, [PipeAB]>;
```



```
let BufferSize = 6 in {
  def PipeAB : ProcResource<2>;
}
def : WriteRes<WriteIALU, [PipeAB]>;
```

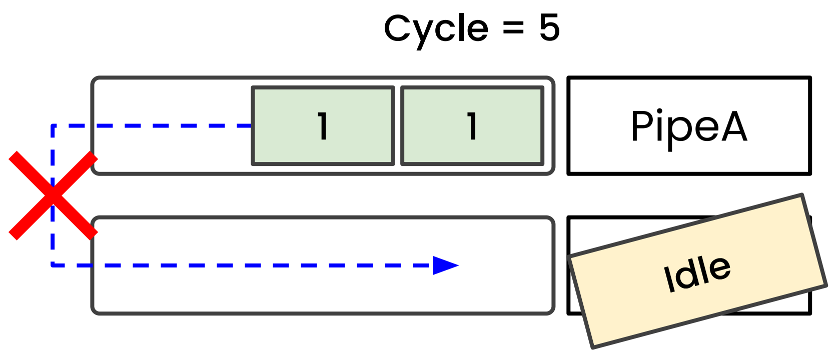
* The code here has been slightly modified

Scheduling Model: Buffer Sizes

Buffer per pipe v.s. Pipes sharing the same buffer

1	: occupancy 1 cy
7	: occupancy 7 cy

```
let BufferSize = 3 in {
  def PipeA : ProcResource<1>;
  def PipeB : ProcResource<1>;
}
def PipeAB : ProcResGroup<[PipeA, PipeB]>;
def : WriteRes<WriteIALU, [PipeAB]>;
```

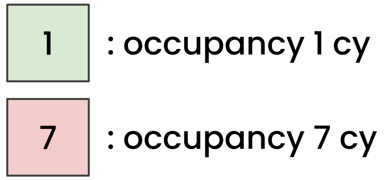


```
let BufferSize = 6 in {
  def PipeAB : ProcResource<2>;
}
def : WriteRes<WriteIALU, [PipeAB]>;
```

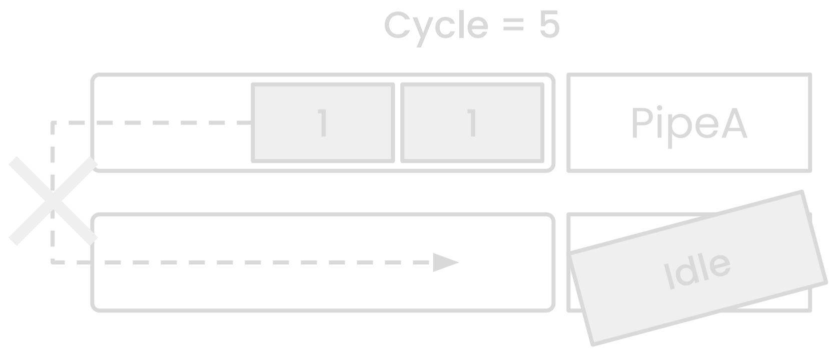
* The code here has been slightly modified

Scheduling Model: Buffer Sizes

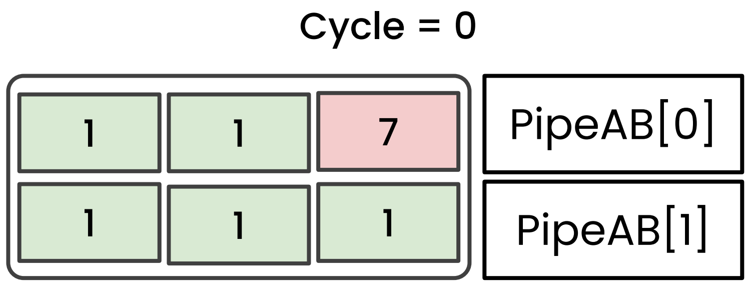
Buffer per pipe v.s. Pipes sharing the same buffer



```
let BufferSize = 3 in {
  def PipeA : ProcResource<1>;
  def PipeB : ProcResource<1>;
}
def PipeAB : ProcResGroup<[PipeA, PipeB]>;
def : WriteRes<WriteIALU, [PipeAB]>;
```



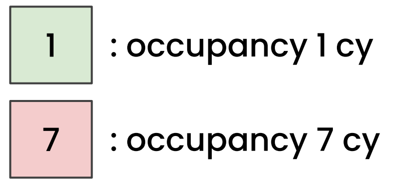
```
let BufferSize = 6 in {
  def PipeAB : ProcResource<2>;
}
def : WriteRes<WriteIALU, [PipeAB]>;
```



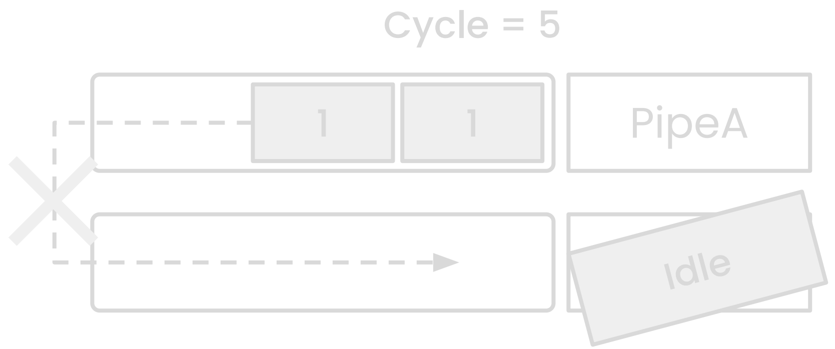
* The code here has been slightly modified

Scheduling Model: Buffer Sizes

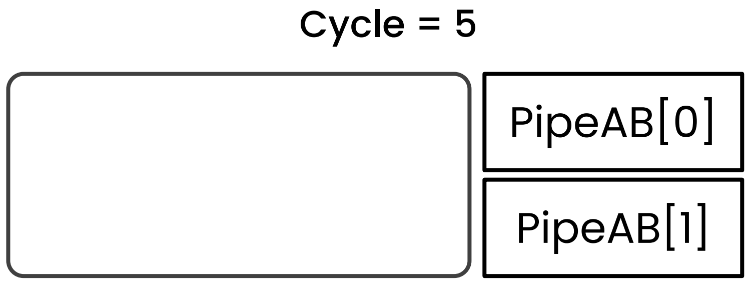
Buffer per pipe v.s. Pipes sharing the same buffer



```
let BufferSize = 3 in {
  def PipeA : ProcResource<1>;
  def PipeB : ProcResource<1>;
}
def PipeAB : ProcResGroup<[PipeA, PipeB]>;
def : WriteRes<WriteIALU, [PipeAB]>;
```



```
let BufferSize = 6 in {
  def PipeAB : ProcResource<2>;
}
def : WriteRes<WriteIALU, [PipeAB]>;
```



* The code here has been slightly modified