Advanced Computer Architectures
(High Performance Processors and Systems)

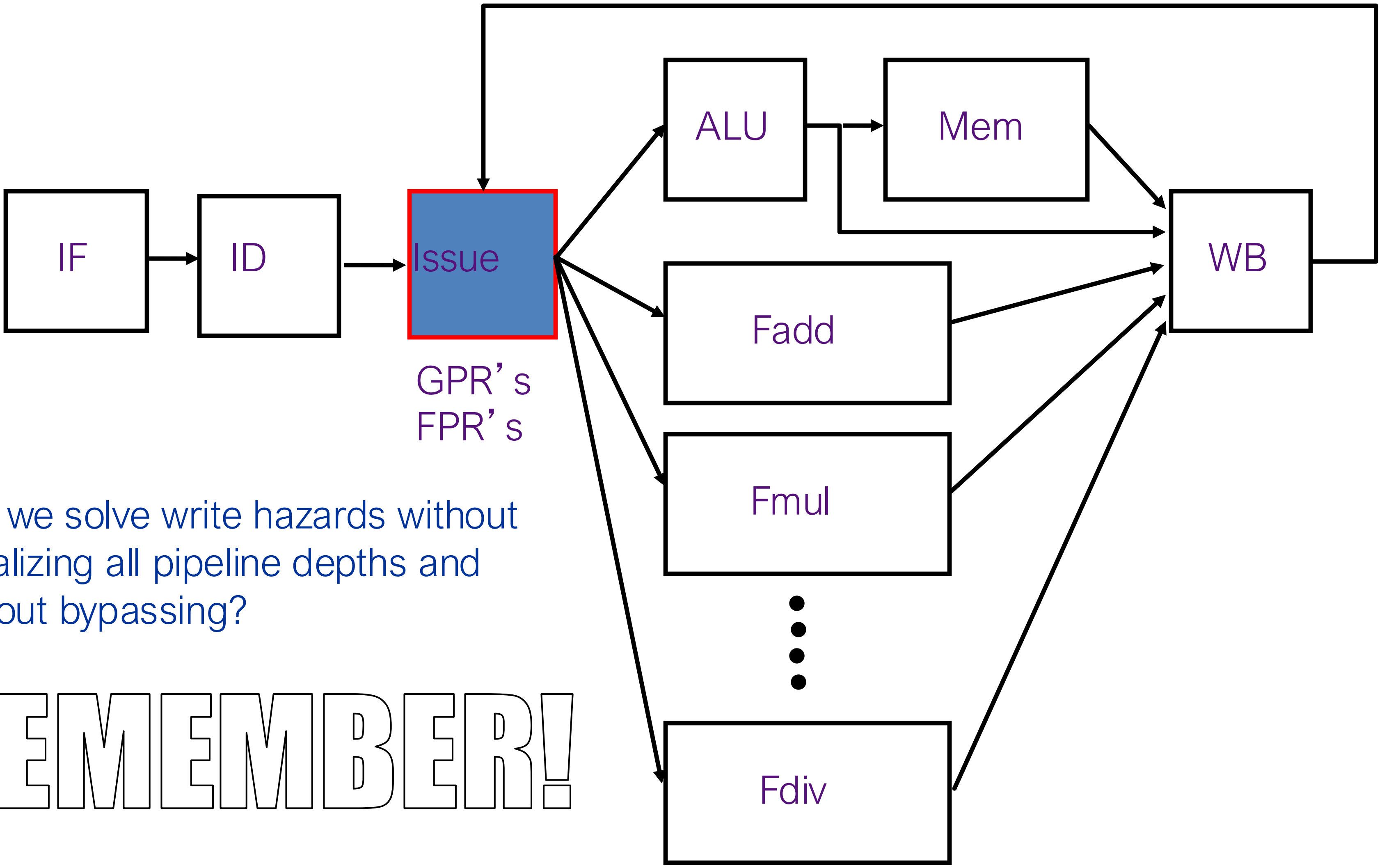
Static Scheduling and VLIW Architectures

Politecnico di Milano
v1

Alessandro Verosimile < alessandro.verosimile@polimi.it>
Marco D. Santambrogio <marco.santambrogio@polimi.it>

https://flic.kr/p/8puKvo

Complex Pipeline



Can we solve write hazards without equalizing all pipeline depths and without bypassing?

REMEMBER!

Instruction Level Parallelism

- Two strategies to support ILP:
  - Dynamic Scheduling: Depend on the hardware to locate parallelism
  - Static Scheduling: Rely on software for identifying potential parallelism
- Hardware intensive approaches dominate desktop and server markets

https://flic.kr/p/8puKvo

Outline

- Introduction to ILP (a short reminder)
- VLIW architecture
- Static Scheduling
  - Basic Blocks
  - Trace scheduling

Beyond CPI = 1

- Initial goal to achieve CPI = 1
- Can we improve beyond this?

- Two approaches
  - Superscalar and VLIW

Beyond CPI = 1

- Initial goal to achieve CPI = 1
- Can we improve beyond this?

- Two approaches
  - Superscalar and VLIW

Beyond CPI = 1

- (Very) Long Instruction Words (V)LIW:
  - fixed number of instructions (4-16)
  - scheduled by the compiler; put ops into wide templates
  - Currently found more success in DSP, Multimedia applications
  - Joint HP/Intel agreement in 1999/2000
  - Intel Architecture-64 (Merced/A-64) 64-bit address
  - Style: “Explicitly Parallel Instruction Computer (EPIC)”

- ILP with a CPI > 1, where to start from…

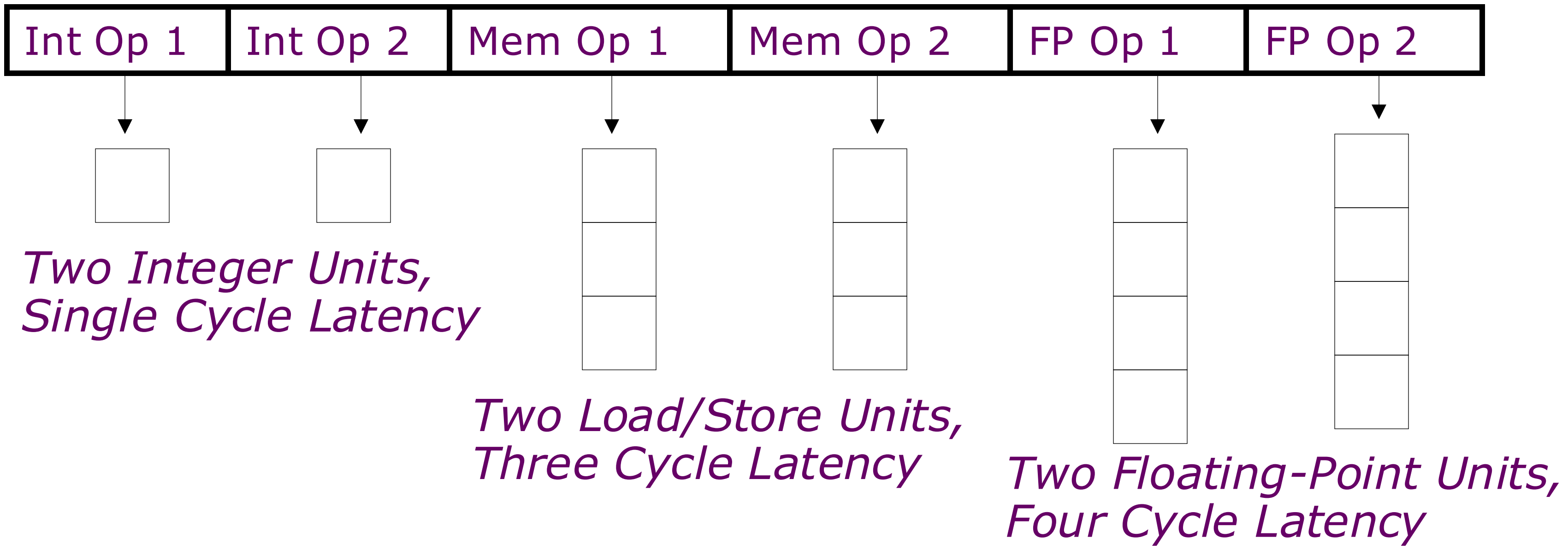
Very Long Instruction Word Architectures

- Processor can initiate multiple operations per cycle
  - Specified completely by the compiler (!like superscalar)
- Low hardware complexity (no scheduling hardware, reduced support of variable latency instructions)
  - No instruction reordering performed by the hardware
- Explicit parallelism
- Single control flow

VLIW processors

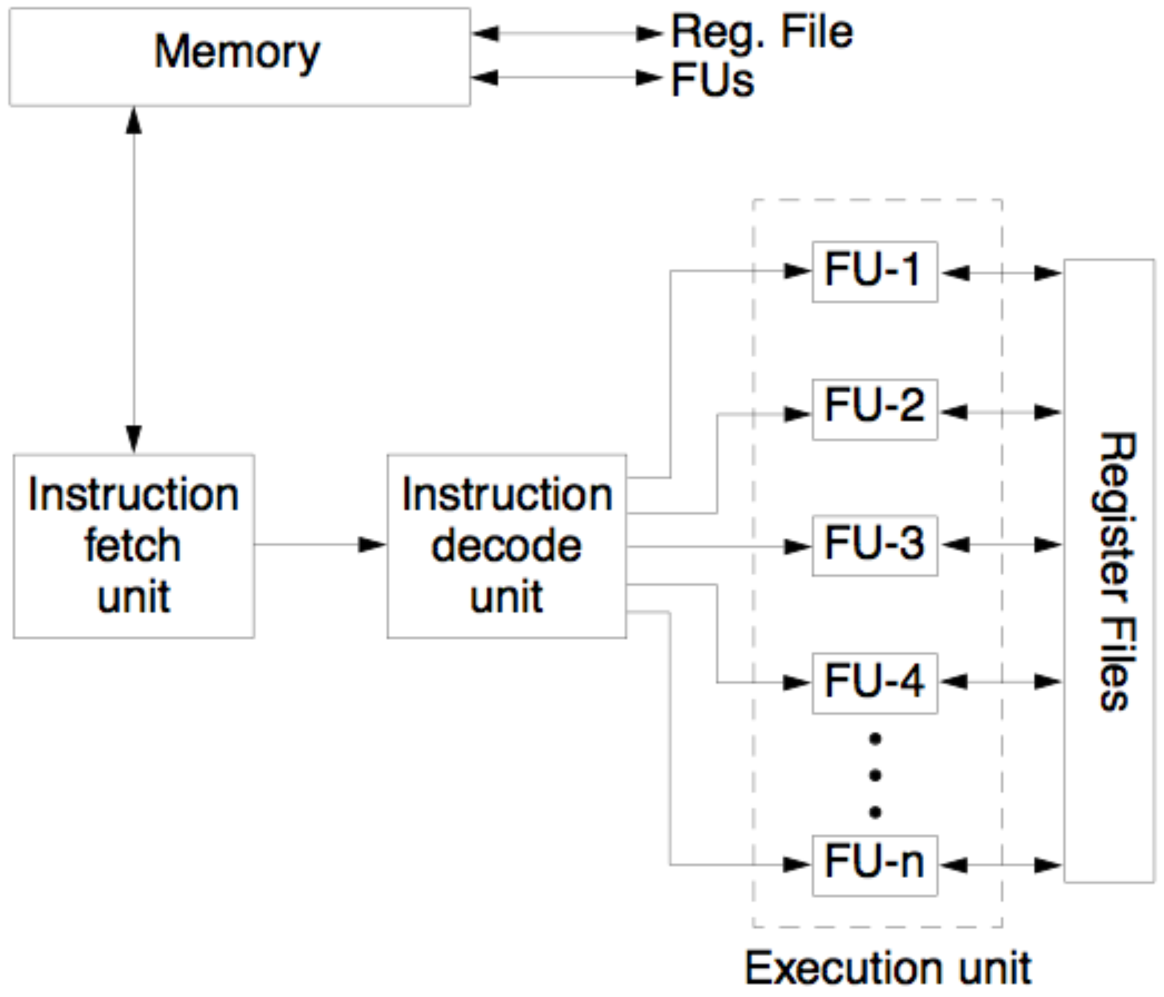
- Operation vs instruction
  - Operation: is a unit of computation (add, load, branch = instruction in sequential ar.)
  - Instruction: set of operations that are intended to be issued simultaneously
- Compiler decides which operation to go to each instruction (scheduling)
- All operations that are supposed to begin at the same time are packaged into a single VLIW instruction

VLIW: Very Long Instruction Word


Int Op 1
Int Op 2
Mem Op 1
Mem Op 2
FP Op 1
FP Op 2
Two Integer Units, Single Cycle Latency
Two Load/Store Units, Three Cycle Latency
Two Floating-Point Units, Four Cycle Latency

- Multiple operations packed into one instruction
- Each operation slot is for a fixed function
- Constant operation latencies are specified
- Architecture requires guarantee of:
  - Parallelism within an instruction => no x-operation RAW check
  - No data use before data ready => no data interlocks

A VLIW Machine Configuration



VLIW Compiler Responsibilities

The compiler:

- Schedules to maximize parallel execution
  - Exploit ILP and LLP (Loop Level Parallelism)
  - It is necessary to map the instructions over the machine functional units
  - This mapping must account for time constraints and dependencies among the tasks
- Guarantees intra-instruction parallelism
- Schedules to avoid data hazards (no interlocks)
  - Typically separates operations with explicit NOPs
- The goal is to minimize the total execution time for the program

VLIW Compiler Responsibilities

The compiler:

- Schedules to maximize parallel execution
  - Exploit ILP and LLP (Loop Level Parallelism)
  - It is necessary to map the instructions over the machine functional units
  - This mapping must account for time constraints and dependencies among the tasks
- Guarantees intra-instruction parallelism
- Schedules to avoid data hazards (no interlocks)
  - Typically separates operations with explicit NOPs
- The goal is to minimize the total execution time for the program

Instruction Level Parallelism

- Two strategies to support ILP:
  - Dynamic Scheduling: Depend on the hardware to locate parallelism
  - Static Scheduling: Rely on software for identifying potential parallelism
- Hardware intensive approaches dominate desktop and server markets

Instruction Level Parallelism

- Two strategies to support ILP:
  - Dynamic Scheduling: Depend on the hardware to locate parallelism
  - Static Scheduling: Rely on software for identifying potential parallelism
- Hardware intensive approaches dominate desktop and server markets

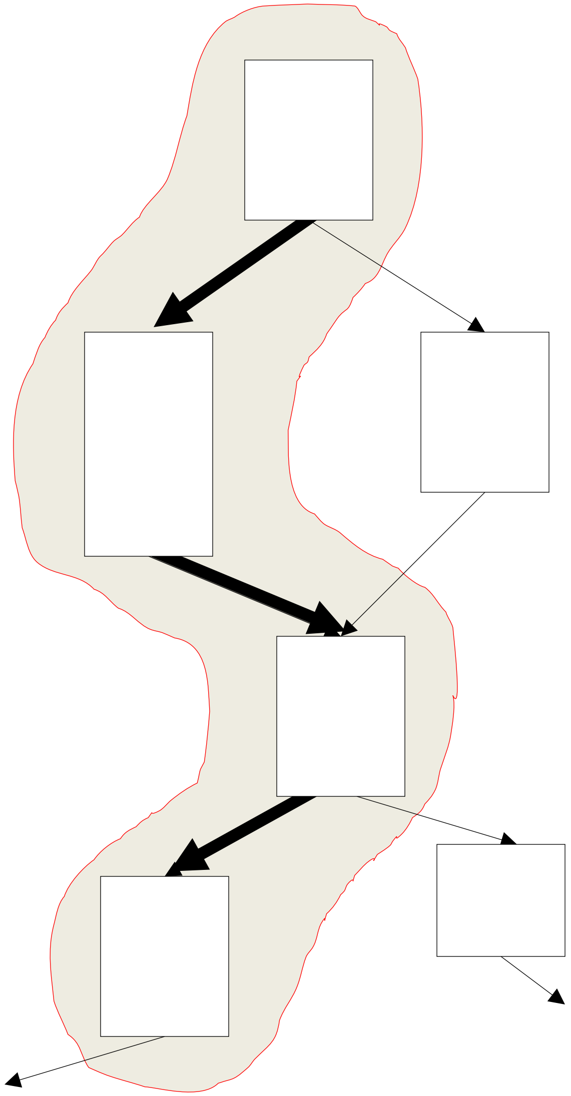
Static Scheduling: the idea

- Try to keep pipeline full (in single issue pipelines) or utilize all FUs in each cycle (in VLIW) as much as possible to reach better ILP and therefore higher parallel speedups.

Static Scheduling: general context

- Compilers can use sophisticated algorithms for code scheduling to exploit ILP (Instruction Level Parallelism).

Trace Scheduling *[Fisher,Ellis]*



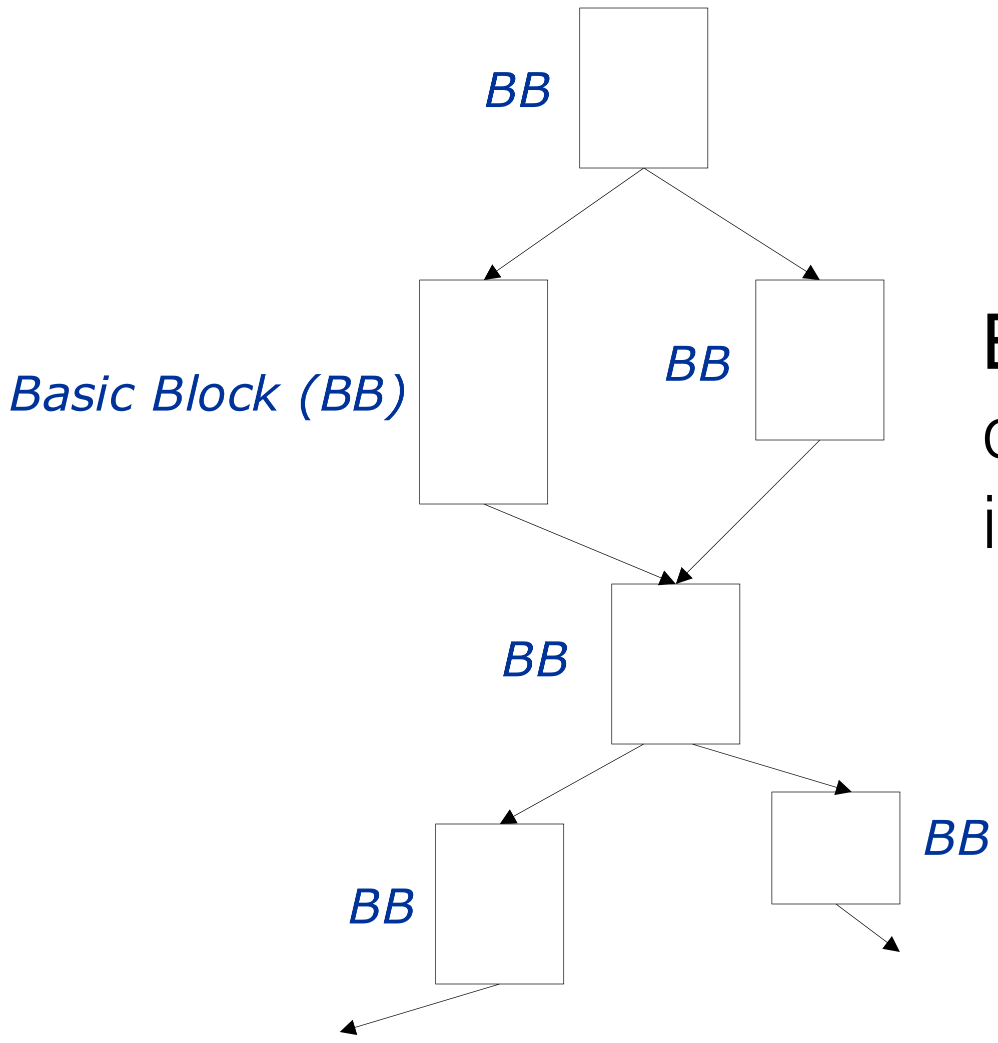
- Ellis JR (1985) Bulldog: a compiler for VLIW architectures. PhD thesis, Yale University
  - https://www.dropbox.com/s/kylqrvcqi137qfe/tr364.pdf?dl=0
- Fisher JA (1993) Global code generation for instruction-level parallelism: trace scheduling-2. Technical Report HPL-93-43. Hewlett-Packard Laboratories
  - https://www.dropbox.com/s/0c6go3udnppq3ov/10.1.1.474.6658.pdf?dl=0
- Fisher JA (1981) Trace scheduling: a technique for global microcode compaction, IEEE Trans Comput, July 1981, 30(7):478–490
  - https://www.dropbox.com/s/m5j0lmy47qxghe4/TraceScheduling.pdf?dl=0
- We are going to see more, not only about Trace Scheduling, next in this class

Static Scheduling: general context

- Compilers can use sophisticated algorithms for code scheduling to exploit ILP (Instruction Level Parallelism).

- The amount of parallelism available within a basic block is quite small.
- Data dependence can further limit the amount of ILP we can exploit within a basic block to much less than the average basic block size.
- To obtain substantial performance enhancements, we must exploit ILP across multiple basic blocks (i.e. across branches).

Basic Block Definition



Basic block: a sequence of straight non-branch instructions

Detection and resolution of dependences Static Scheduling

- Static detection and resolution of dependences (⇨ static scheduling): accomplished by the compiler ⇨ dependences are avoided by code reordering.
  - Output of the compiler: reordered into dependency-free code.
- Typical example: VLIW (Very Long Instruction Word) processors expect dependency-free code.

VLIW: Pros and Cons

- Pros
  - Simple HW
    - Easy to extend the #FUs
  - Good compilers can effectively detect parallelism

- Cons
  - Huge number of registers to keep active the FUs
    - Needed to store operands and results
  - Large data transport capacity between
    - FUs and register files
    - Register files and Memory
  - High bandwidth between i-cache and fetch unit
  - Large code size
- Knowing branch probabilities
  - Profiling requires a significant extra step in build process
- Scheduling for statically unpredictable branches
  - optimal schedule varies with branch path

Static Scheduling: methods

- Simple code motion
- Loop unrolling & loop peeling
- Software pipeline
- Global code scheduling (across basic block)
  - Trace scheduling
  - Superblock scheduling
  - Hyperblock scheduling
  - Speculative Trace scheduling

Loop Execution

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Loop Execution

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

$\downarrow$ *Compile*

```
loop:   ld f1, 0(r1)
        add r1, 8
        fadd f2, f0, f1
        sd f2, 0(r2)
        add r2, 8
        bne r1, r3, loop
```

Loop Execution

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

$$\Big\downarrow \text{ \textit{Compile}}$$

```
loop:   ld f1, 0(r1)
        add r1, 8
        fadd f2, f0, f1
        sd f2, 0(r2)
        add r2, 8
        bne r1, r3, loop
```

ld, sd : 3cc

add, bne : 1cc

fadd: 4cc

Loop Execution

ld, sd : 3cc
add, bne : 1cc
fadd: 4cc

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

↓ *Compile*

```
loop:   ld f1, 0(r1)
        add r1, 8
        fadd f2, f0, f1
        sd f2, 0(r2)
        add r2, 8
        bne r1, r3, loop
```

Schedule →

	Int1	Int 2	M1	M2	FP+	FPx
	1cc	1cc	3cc	3cc	4cc	4cc
loop:						

Loop Execution

ld, sd : 3cc
add, bne : 1cc
fadd: 4cc

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Compile

```
loop:   ld f1, 0(r1)
        add r1, 8
        fadd f2, f0, f1
        sd f2, 0(r2)
        add r2, 8
        bne r1, r3, loop
```

Schedule

	Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
loop:			ld			

Loop Execution

ld, sd : 3cc
add, bne : 1cc
fadd: 4cc

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Compile

```
loop:   ld f1, 0(r1)
        add r1, 8
        fadd f2, f0, f1
        sd f2, 0(r2)
        add r2, 8
        bne r1, r3, loop
```

Schedule

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
add r1		ld			

Loop Execution

ld, sd : 3cc
add, bne : 1cc
fadd: 4cc

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Compile

```
loop:   ld f1, 0(r1)
        add r1, 8
        fadd f2, f0, f1
        sd f2, 0(r2)
        add r2, 8
        bne r1, r3, loop
```

Schedule

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
add r1		ld		1cc	
				2cc	
				3cc	

Loop Execution

ld, sd : 3cc
add, bne : 1cc
fadd: 4cc

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Compile

```
loop:  ld f1, 0(r1)
       add r1, 8
       fadd f2, f0, f1
       sd f2, 0(r2)
       add r2, 8
       bne r1, r3, loop
```

Schedule

	Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
loop:	add r1		ld		1cc	
					2cc	
					3cc	
					fadd	

Loop Execution

ld, sd : 3cc
add, bne : 1cc
fadd: 4cc

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Compile

```
loop:   ld f1, 0(r1)
        add r1, 8
        fadd f2, f0, f1
        sd f2, 0(r2)
        add r2, 8
        bne r1, r3, loop
```

Schedule

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
add r1		ld			
		1cc		fadd	
		2cc			
		3cc			
		4cc			

Loop Execution

ld, sd : 3cc
add, bne : 1cc
fadd: 4cc

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

↓ *Compile*

```
loop:   ld f1, 0(r1)
        add r1, 8
        fadd f2, f0, f1
        sd f2, 0(r2)
        add r2, 8
        bne r1, r3, loop
```

Schedule →

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
add r1		ld			
		1cc		fadd	
		2cc			
		3cc			
		4cc			
		sd			

Loop Execution

ld, sd : 3cc
add, bne : 1cc
fadd: 4cc

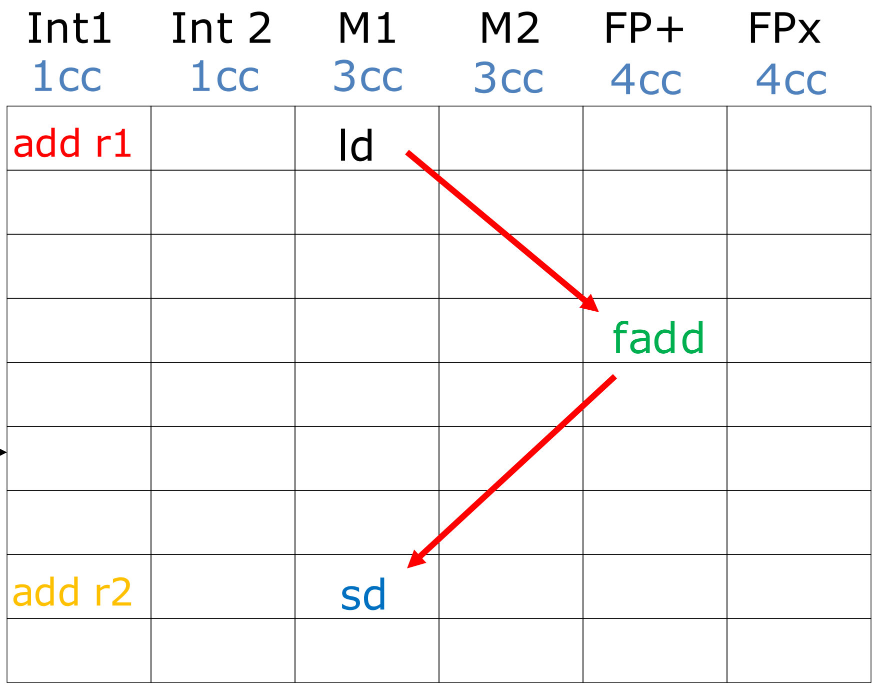
```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Compile

```
loop:   ld f1, 0(r1)
        add r1, 8
        fadd f2, f0, f1
        sd f2, 0(r2)
        add r2, 8
        bne r1, r3, loop
```

Schedule

loop:



Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
add r1		ld			
				fadd	
add r2		sd			

Loop Execution

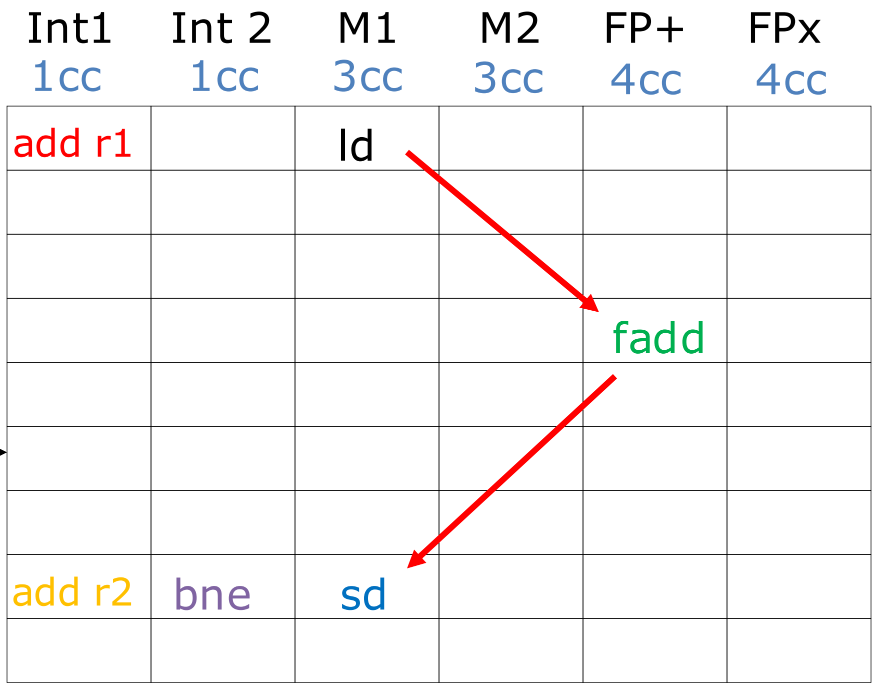
ld, sd : 3cc
add, bne : 1cc
fadd: 4cc

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Compile

```
loop:  ld f1, 0(r1)
       add r1, 8
       fadd f2, f0, f1
       sd f2, 0(r2)
       add r2, 8
       bne r1, r3, loop
```

Schedule



	Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
loop:	add r1		ld			
					fadd	
	add r2	bne	sd			

Loop Execution

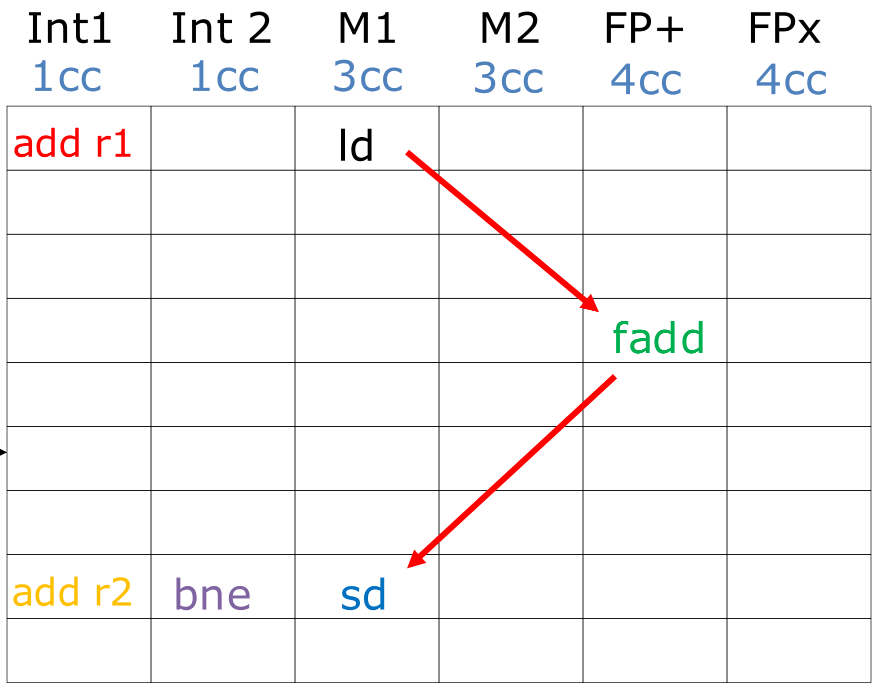
ld, sd : 3cc
add, bne : 1cc
fadd: 4cc

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Compile

```
loop:   ld f1, 0(r1)
        add r1, 8
        fadd f2, f0, f1
        sd f2, 0(r2)
        add r2, 8
        bne r1, r3, loop
```

Schedule



loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
add r1		ld			
				fadd	
add r2	bne	sd			

How many FP ops/cycle?

Loop Execution

ld, sd : 3cc
add, bne : 1cc
fadd: 4cc

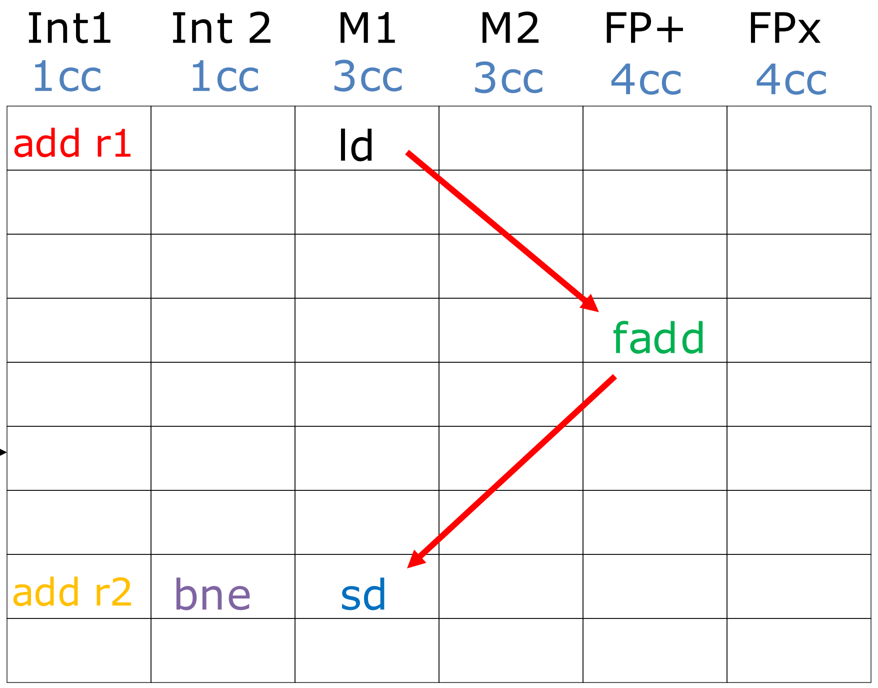
```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Compile

```
loop:   ld f1, 0(r1)
        add r1, 8
        fadd f2, f0, f1
        sd f2, 0(r2)
        add r2, 8
        bne r1, r3, loop
```

Schedule

loop:



Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
add r1		ld			
				fadd	
add r2	bne	sd			

How many FP ops/cycle?

1 fadd / 8 cycles = 0.125

Loop Unrolling

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Loop Unrolling

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Unroll inner loop to perform 4 iterations at once

Loop Unrolling

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

$\downarrow$ Unroll inner loop to perform 4 iterations at once

```
for (i=0; i<N; i+=4){
    B[i]     = A[i] + C;
    B[i+1] = A[i+1] + C;
    B[i+2] = A[i+2] + C;
    B[i+3] = A[i+3] + C;
}
```

Need to handle values of N that are not multiples of unrolling factor with final cleanup loop

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule $\longrightarrow$

	Int1	Int 2	M1	M2	FP+	FPx
	1cc	1cc	3cc	3cc	4cc	4cc
loop:						

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:  ld f1, 0(r1)
       ld f2, 8(r1)
       ld f3, 16(r1)
       ld f4, 24(r1)
       add r1, 32
       fadd f5, f0, f1
       fadd f6, f0, f2
       fadd f7, f0, f3
       fadd f8, f0, f4
       sd f5, 0(r2)
       sd f6, 8(r2)
       sd f7, 16(r2)
       sd f8, 24(r2)
       add r2, 32
       bne r1, r3, loop
```

Schedule →

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:  ld f1, 0(r1)
       ld f2, 8(r1)
       ld f3, 16(r1)
       ld f4, 24(r1)
       add r1, 32
       fadd f5, f0, f1
       fadd f6, f0, f2
       fadd f7, f0, f3
       fadd f8, f0, f4
       sd f5, 0(r2)
       sd f6, 8(r2)
       sd f7, 16(r2)
       sd f8, 24(r2)
       add r2, 32
       bne r1, r3, loop
```

Schedule $\longrightarrow$

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1		1cc	
				2cc	
				3cc	

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule →

loop:

Int1	Int 2	M1	M2	FP+	FPx
1cc	1cc	3cc	3cc	4cc	4cc
		ld f1		1cc	
				2cc	
				3cc	
				fadd f5	

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:  ld f1, 0(r1)
       ld f2, 8(r1)
       ld f3, 16(r1)
       ld f4, 24(r1)
       add r1, 32
       fadd f5, f0, f1
       fadd f6, f0, f2
       fadd f7, f0, f3
       fadd f8, f0, f4
       sd f5, 0(r2)
       sd f6, 8(r2)
       sd f7, 16(r2)
       sd f8, 24(r2)
       add r2, 32
       bne r1, r3, loop
```

Schedule →

	Int1	Int 2	M1	M2	FP+	FPx
	1cc	1cc	3cc	3cc	4cc	4cc
loop:			ld f1			
			1cc		fadd f5	
			2cc			
			3cc			
			4cc			

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule $\longrightarrow$

loop:

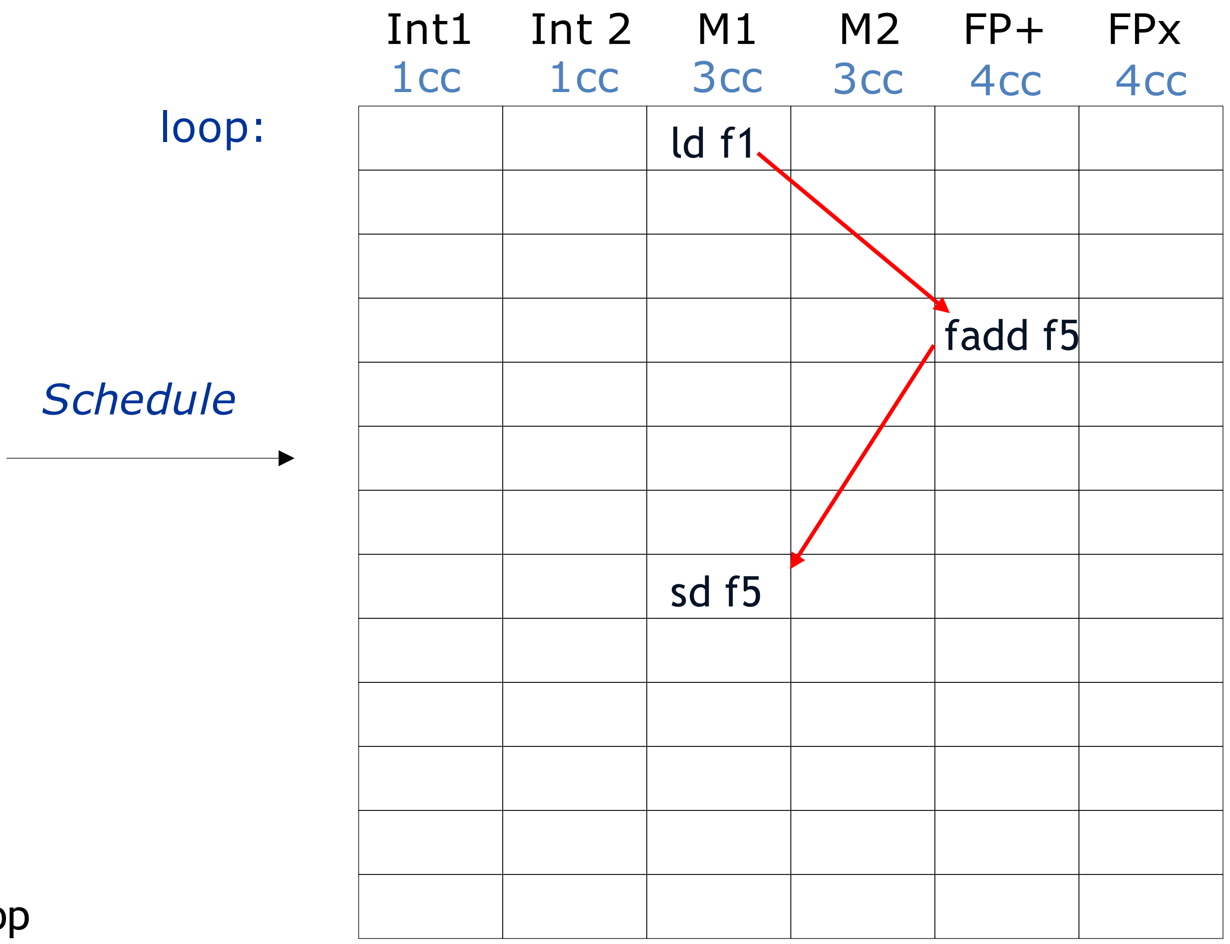
Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		1cc		fadd f5	
		2cc			
		3cc			
		4cc			
		sd f5			

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:  ld f1, 0(r1)
       ld f2, 8(r1)
       ld f3, 16(r1)
       ld f4, 24(r1)
       add r1, 32
       fadd f5, f0, f1
       fadd f6, f0, f2
       fadd f7, f0, f3
       fadd f8, f0, f4
       sd f5, 0(r2)
       sd f6, 8(r2)
       sd f7, 16(r2)
       sd f8, 24(r2)
       add r2, 32
       bne r1, r3, loop
```

Schedule

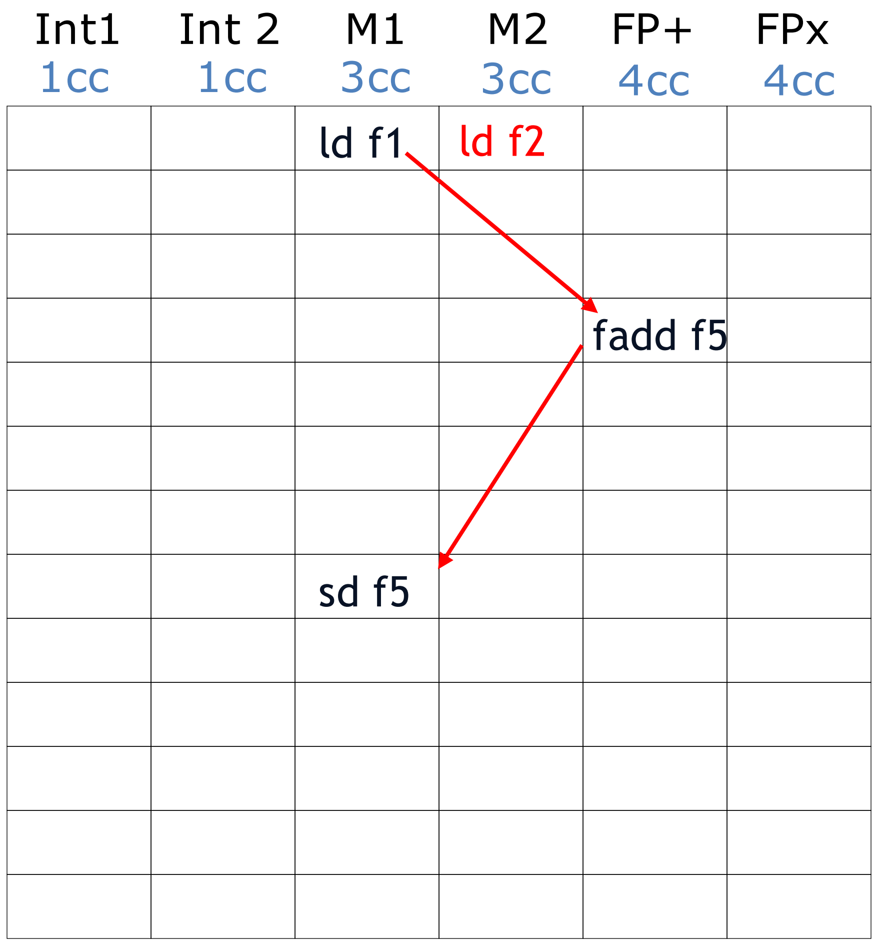


Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule $\longrightarrow$



loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1	ld f2		
				fadd f5	
		sd f5			

Scheduling Loop Unrolled Code

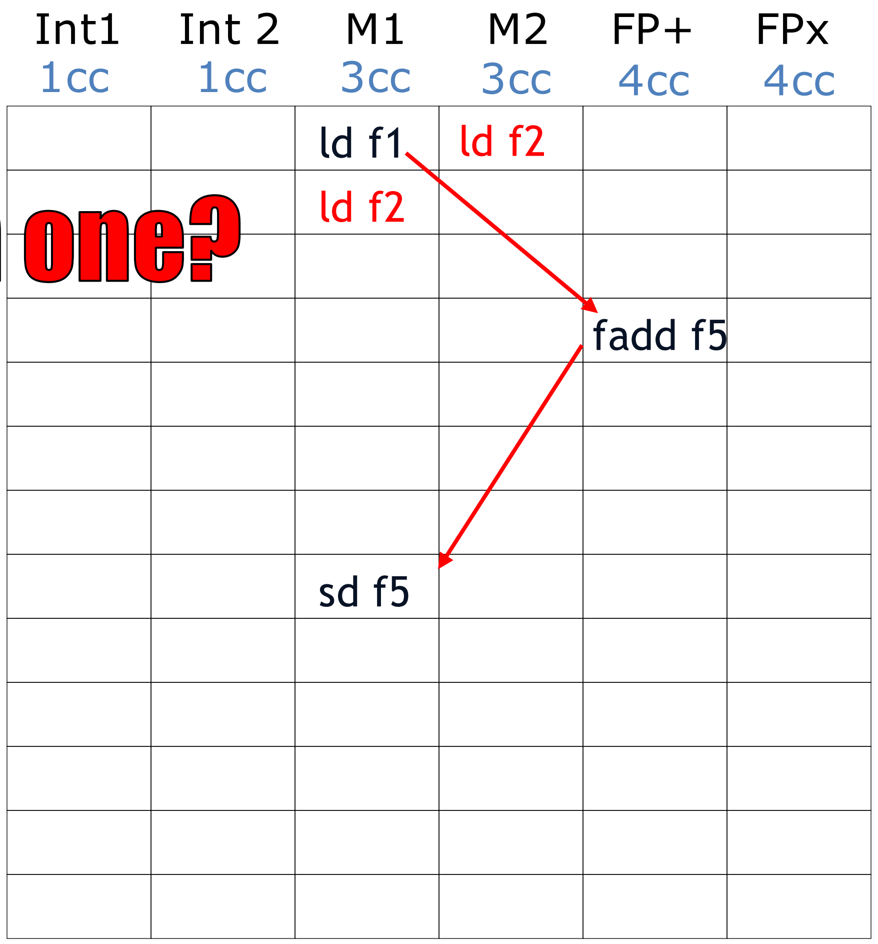
Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule $\longrightarrow$

Which one?

loop:



Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1	ld f2		
		ld f2			
				fadd f5	
		sd f5			

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule $\longrightarrow$

loop:

Which one?

Int1	Int 2	M1	M2	FP+	FPx
1cc	1cc	3cc	3cc	4cc	4cc
		ld f1	ld f2		
		ld f2			
				fadd f5	
				fadd f6	
		sd f5			

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule $\longrightarrow$

THIS one!!!

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1	🟥		
		ld f2			
				fadd f5	
				fadd f6	
		sd f5			

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule →

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
				fadd f5	
				fadd f6	
		sd f5			
		sd f6			

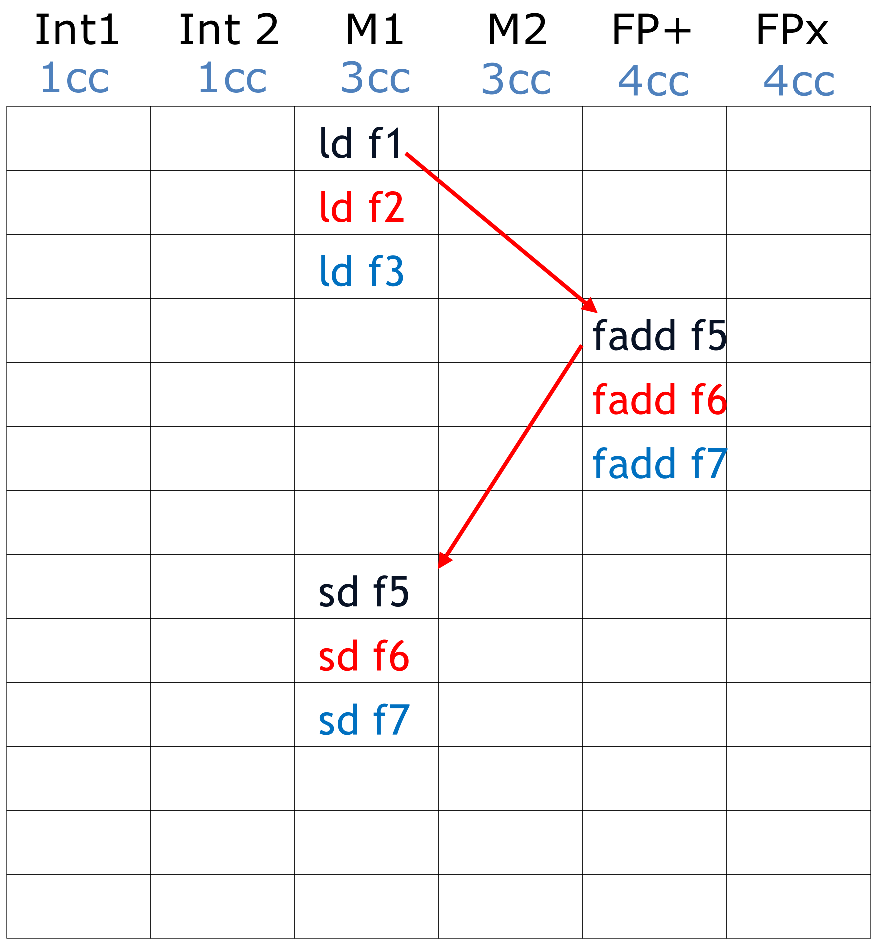
Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule →

loop:



Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
				fadd f5	
				fadd f6	
				fadd f7	
		sd f5			
		sd f6			
		sd f7			

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule $\longrightarrow$

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
		ld f4		fadd f5	
				fadd f6	
				fadd f7	
				fadd f8	
		sd f5			
		sd f6			
		sd f7			
		sd f8			

Scheduling Loop Unrolled Code

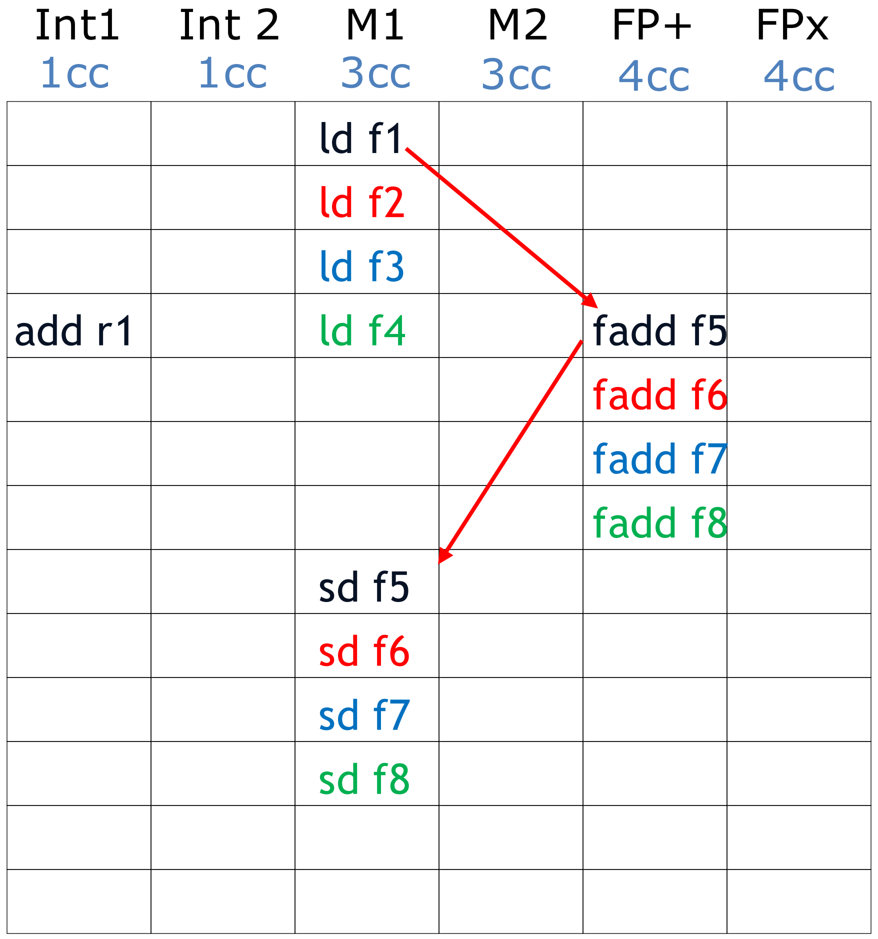
Unroll 4 ways

```
loop:  ld f1, 0(r1)
       ld f2, 8(r1)
       ld f3, 16(r1)
       ld f4, 24(r1)
       add r1, 32
       fadd f5, f0, f1
       fadd f6, f0, f2
       fadd f7, f0, f3
       fadd f8, f0, f4
       sd f5, 0(r2)
       sd f6, 8(r2)
       sd f7, 16(r2)
       sd f8, 24(r2)
       add r2, 32
       bne r1, r3, loop
```

ASAP

Schedule

loop:



Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4		fadd f5	
				fadd f6	
				fadd f7	
				fadd f8	
		sd f5			
		sd f6			
		sd f7			
		sd f8			

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule $\longrightarrow$

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4		fadd f5	
				fadd f6	
				fadd f7	
				fadd f8	
		sd f5			
		sd f6			
		sd f7			
add r2		sd f8			

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule →

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4		fadd f5	
				fadd f6	
				fadd f7	
				fadd f8	
		sd f5			
		sd f6			
		sd f7			
add r2	bne	sd f8			

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:  ld f1, 0(r1)
       ld f2, 8(r1)
       ld f3, 16(r1)
       ld f4, 24(r1)
       add r1, 32
       fadd f5, f0, f1
       fadd f6, f0, f2
       fadd f7, f0, f3
       fadd f8, f0, f4
       sd f5, 0(r2)
       sd f6, 8(r2)
       sd f7, 16(r2)
       sd f8, 24(r2)
       add r2, 32
       bne r1, r3, loop
```

Schedule $\longrightarrow$

	Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
loop:			ld f1			
			ld f2			
			ld f3			
	add r1		ld f4		fadd f5	
					fadd f6	
					fadd f7	
					fadd f8	
			sd f5			
			sd f6			
			sd f7			
	add r2	bne	sd f8			

How many FLOPS/cycle?

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        sd f8, 24(r2)
        add r2, 32
        bne r1, r3, loop
```

Schedule $\longrightarrow$

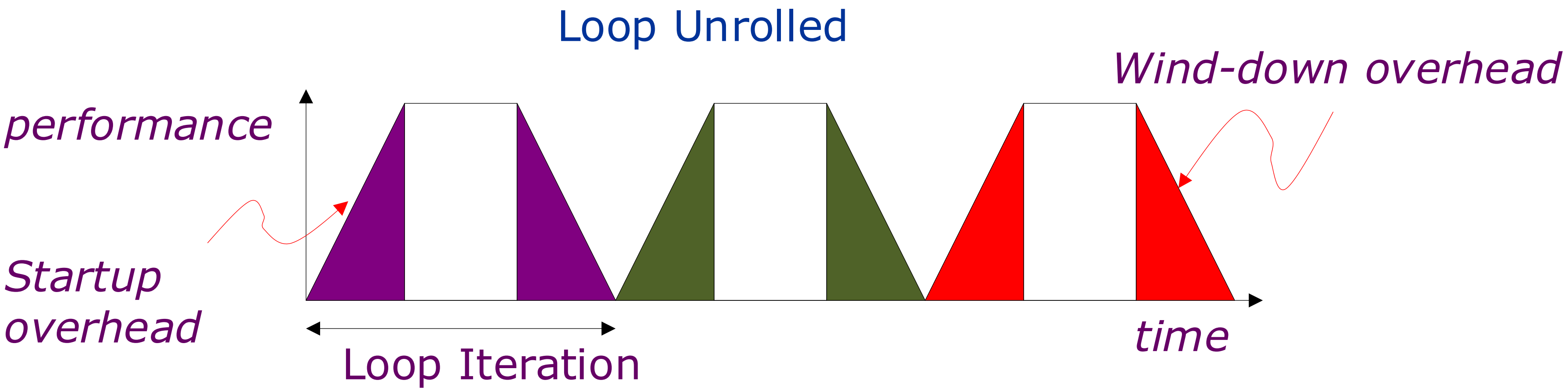
loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4		fadd f5	
				fadd f6	
				fadd f7	
				fadd f8	
		sd f5			
		sd f6			
		sd f7			
add r2	bne	sd f8			

How many FLOPS/cycle?

4 fadds / 11 cycles = 0.36

Loop Unrolling



Software Pipelining

Unroll 4 ways first

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        add r2, 32
        sd f8, -8(r2)
        bne r1, r3, loop
```

Schedule $\longrightarrow$

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc

Software Pipelining

Unroll 4 ways first

```
loop:  ld f1, 0(r1)
       ld f2, 8(r1)
       ld f3, 16(r1)
       ld f4, 24(r1)
       add r1, 32
       fadd f5, f0, f1
       fadd f6, f0, f2
       fadd f7, f0, f3
       fadd f8, f0, f4
       sd f5, 0(r2)
       sd f6, 8(r2)
       sd f7, 16(r2)
       add r2, 32
       sd f8, -8(r2)
       bne r1, r3, loop
```

Schedule →

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4			
				fadd f5	
				fadd f6	
				fadd f7	
				fadd f8	
			sd f5		
			sd f6		
			sd f7		
	bne		sd f8		

Software Pipelining

Unroll 4 ways first

```
loop:  ld f1, 0(r1)
       ld f2, 8(r1)
       ld f3, 16(r1)
       ld f4, 24(r1)
       add r1, 32
       fadd f5, f0, f1
       fadd f6, f0, f2
       fadd f7, f0, f3
       fadd f8, f0, f4
       sd f5, 0(r2)
       sd f6, 8(r2)
       sd f7, 16(r2)
       add r2, 32
       sd f8, -8(r2)
       bne r1, r3, loop
```

Schedule →

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4			
				fadd f5	
				fadd f6	
				fadd f7	
				fadd f8	
			sd f5		
			sd f6		
			sd f7		
	bne		sd f8		

Software Pipelining

Unroll 4 ways first

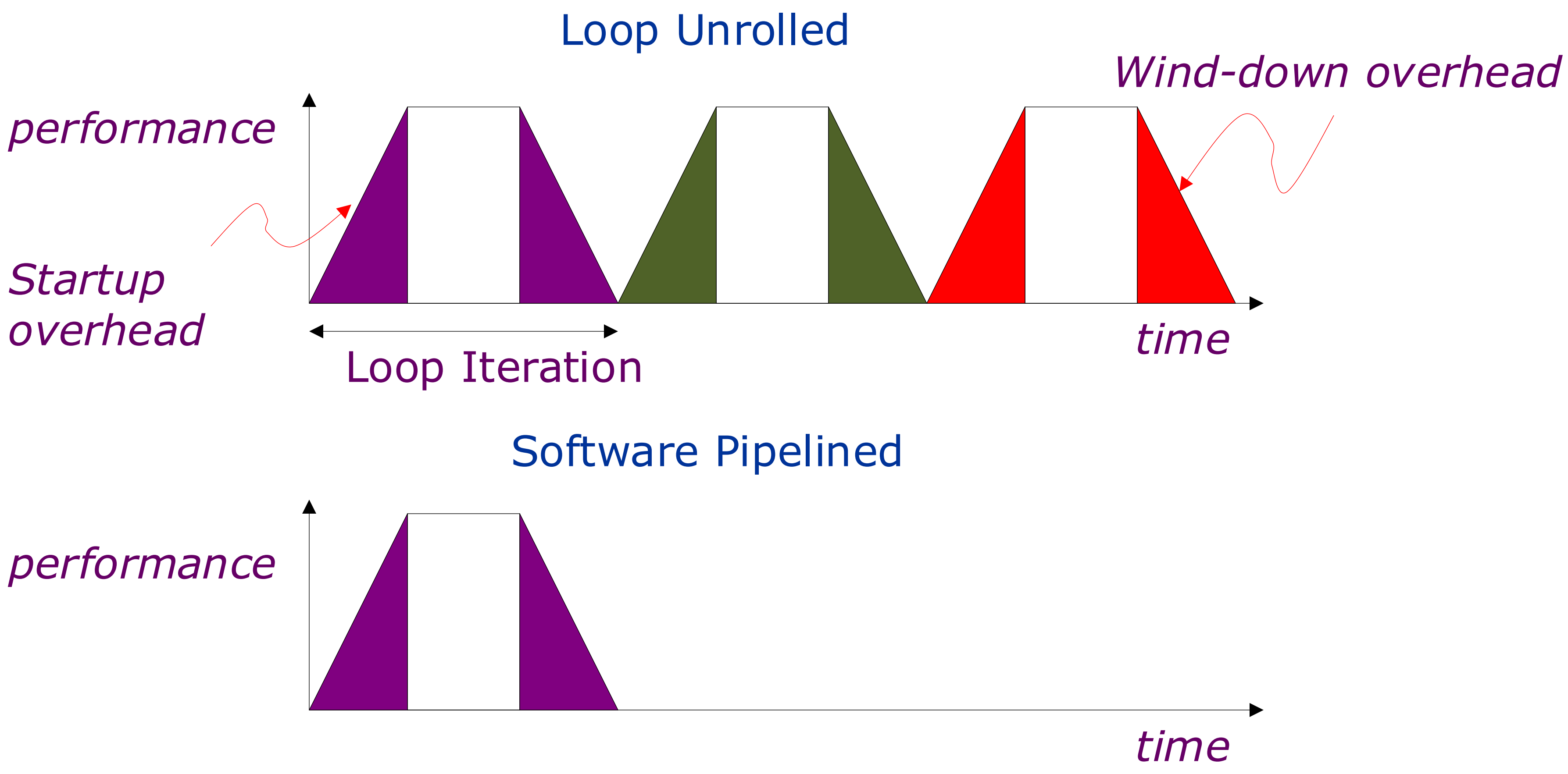
```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        add r2, 32
        sd f8, -8(r2)
        bne r1, r3, loop
```

Schedule

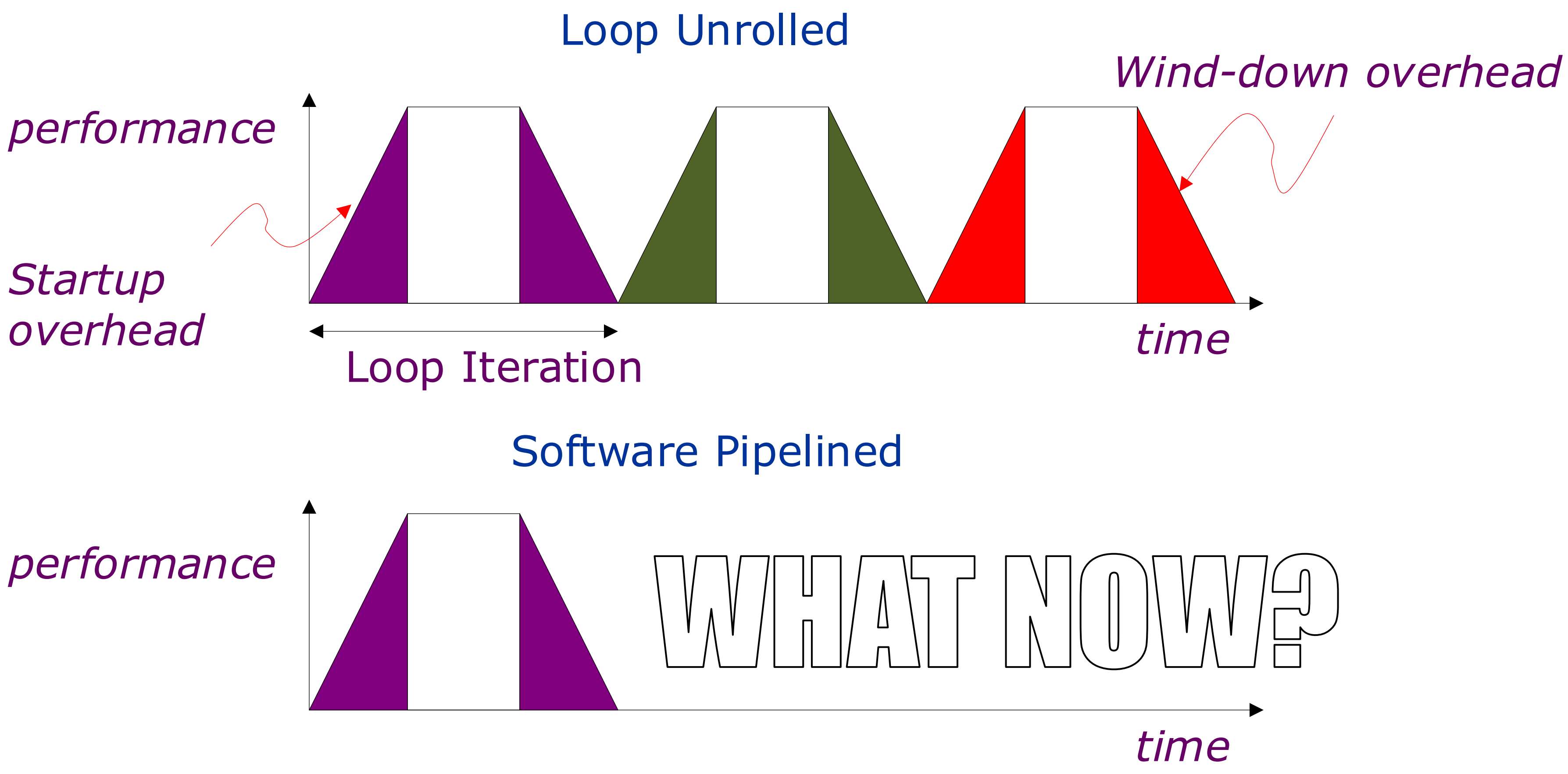
loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4			
				fadd f5	
				fadd f6	
				fadd f7	
				fadd f8	
			sd f5		
			sd f6		
	add r2		sd f7		
	bne		sd f8		

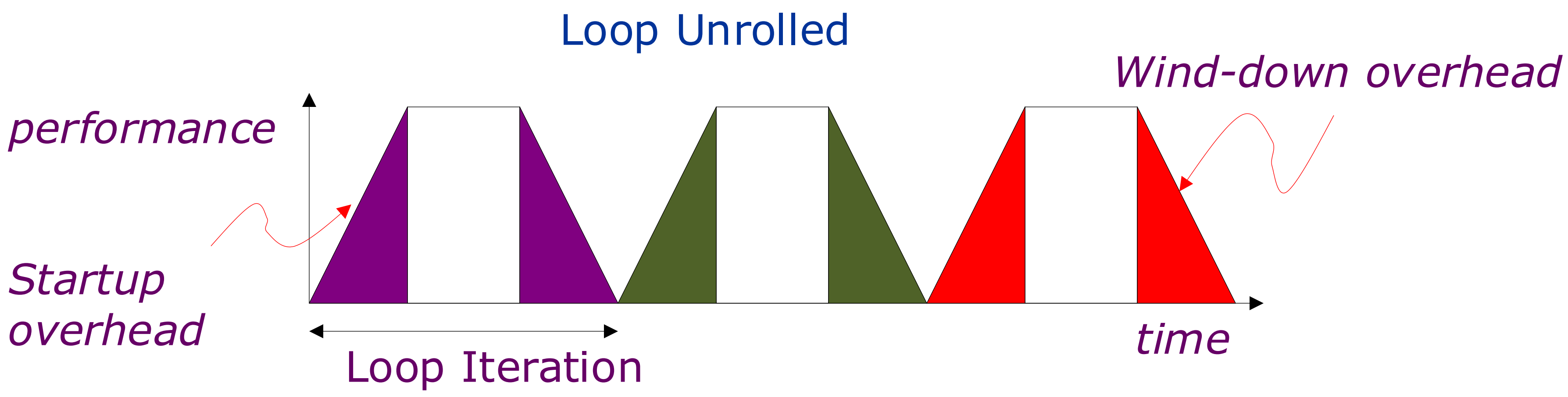
Loop Unrolling vs. Software Pipelining

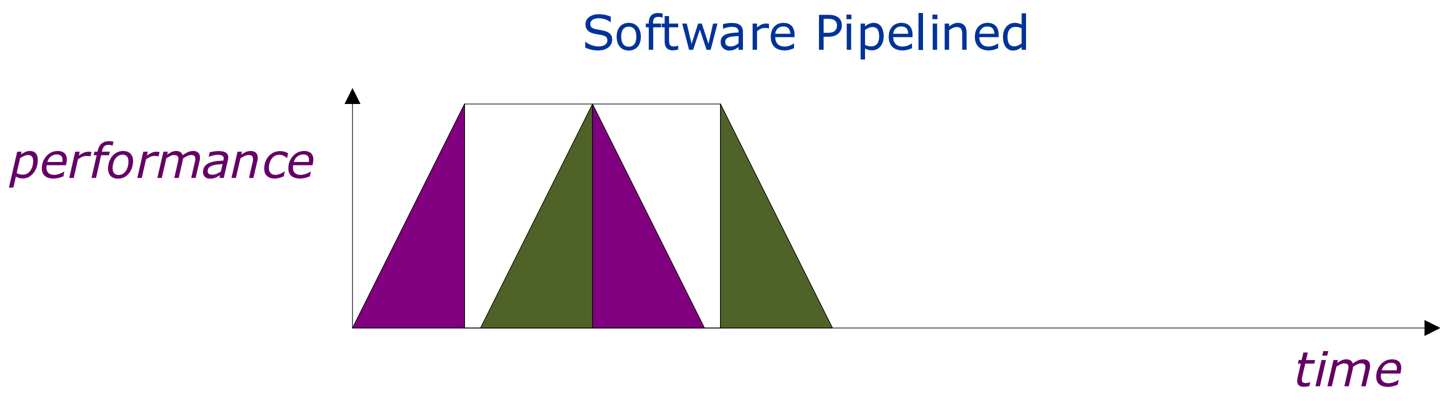


Loop Unrolling vs. Software Pipelining



Loop Unrolling vs. Software Pipelining





Software Pipelining

Unroll 4 ways first

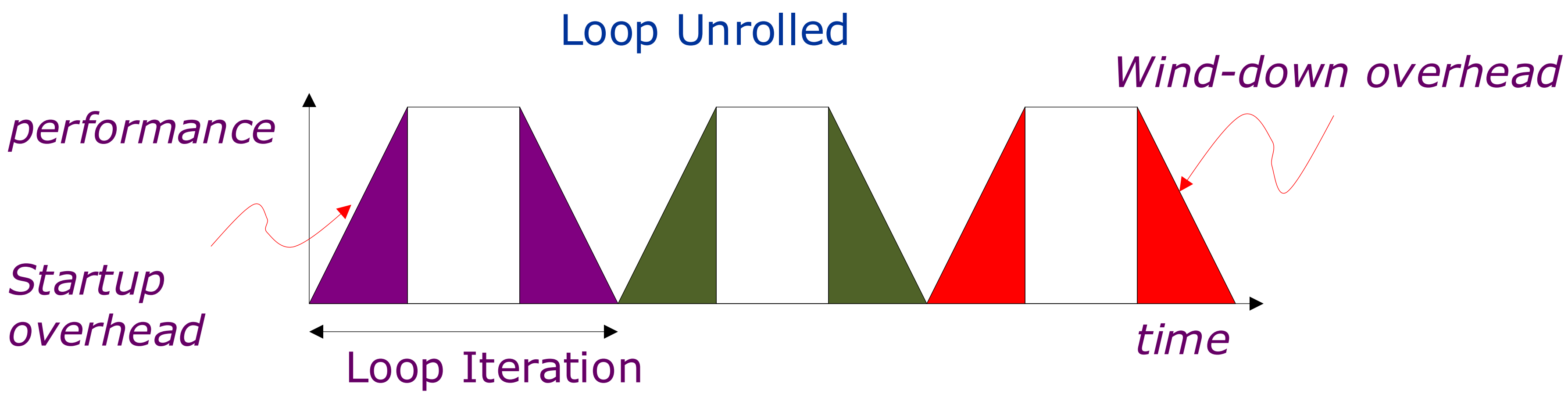
```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        add r2, 32
        sd f8, -8(r2)
        bne r1, r3, loop
```

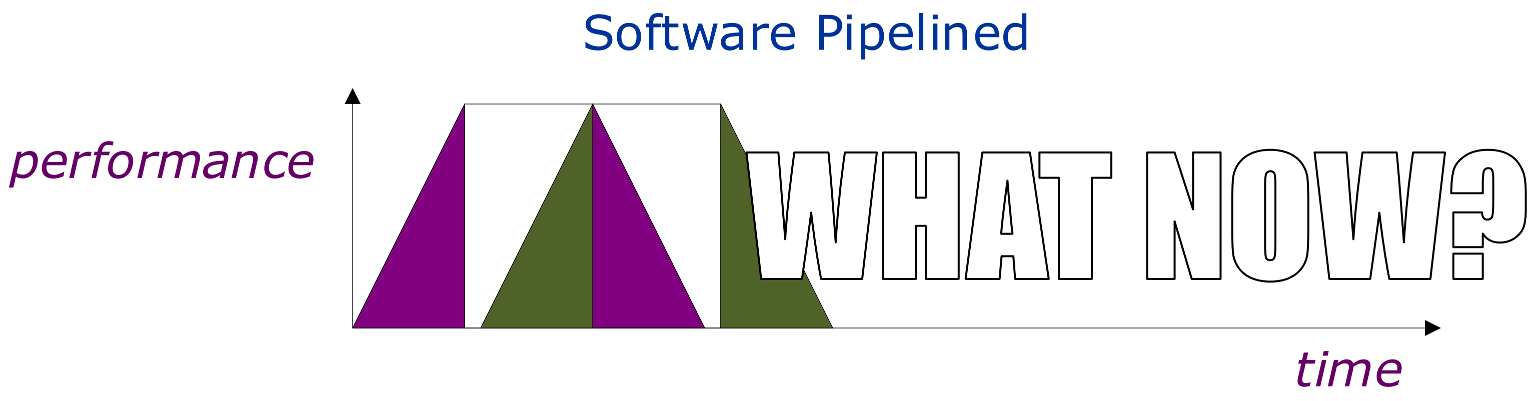
Schedule →

loop:

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4			
		ld f1		fadd f5	
		ld f2		fadd f6	
		ld f3		fadd f7	
add r1		ld f4		fadd f8	
			sd f5	fadd f5	
			sd f6	fadd f6	
	add r2		sd f7	fadd f7	
	bne		sd f8	fadd f8	
			sd f5		
			sd f6		
	add r2		sd f7		
	bne		sd f8		

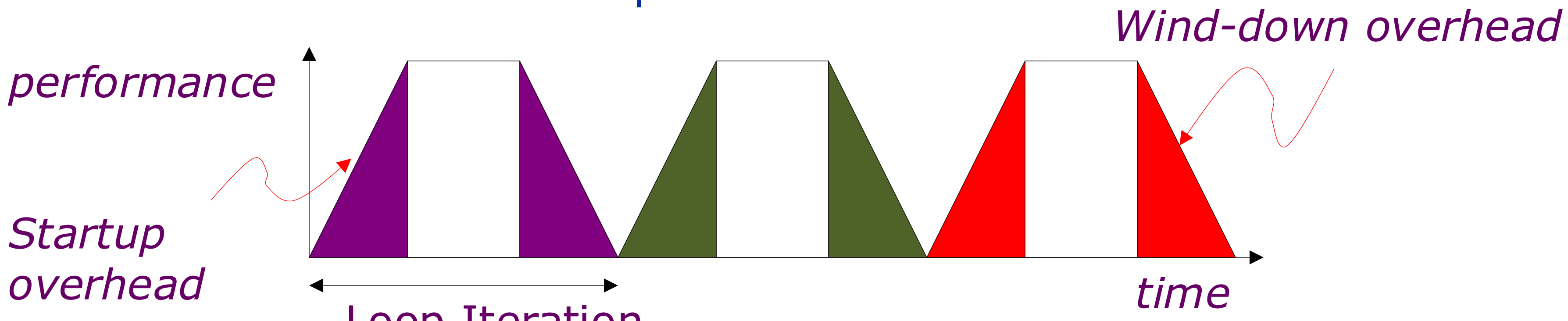
Loop Unrolling vs. Software Pipelining





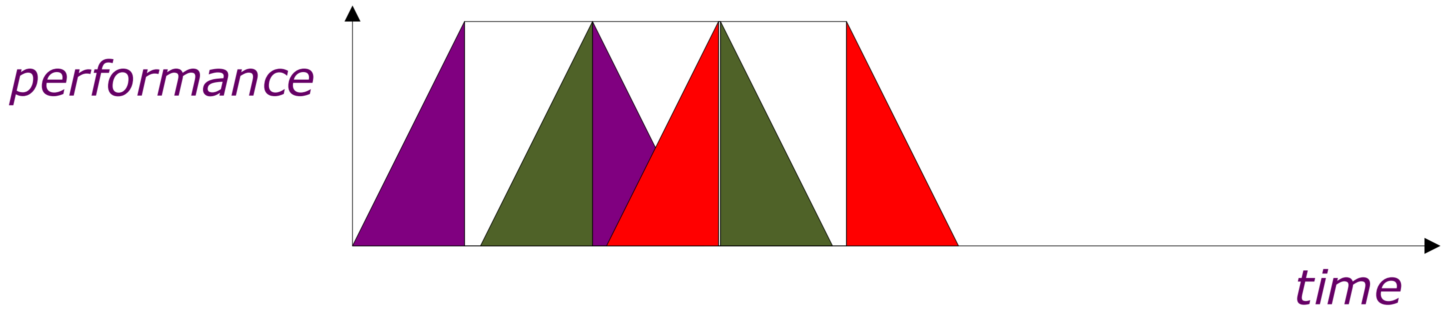
Loop Unrolling vs. Software Pipelining

Loop Unrolled



Loop Iteration

Software Pipelined



Software Pipelining

Unroll 4 ways first

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        add r2, 32
        sd f8, -8(r2)
        bne r1, r3, loop
```

Schedule $\longrightarrow$

loop:

	Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
loop:			ld f1			
			ld f2			
			ld f3			
	add r1		ld f4			
			ld f1		fadd f5	
			ld f2		fadd f6	
			ld f3		fadd f7	
	add r1		ld f4		fadd f8	
			ld f1	sd f5	fadd f5	
			ld f2	sd f6	fadd f6	
		add r2	ld f3	sd f7	fadd f7	
	add r1	bne	ld f4	sd f8	fadd f8	
				sd f5	fadd f5	
				sd f6	fadd f6	
		add r2		sd f7	fadd f7	
		bne		sd f8	fadd f8	
				sd f5		

Software Pipelining

Unroll 4 ways first

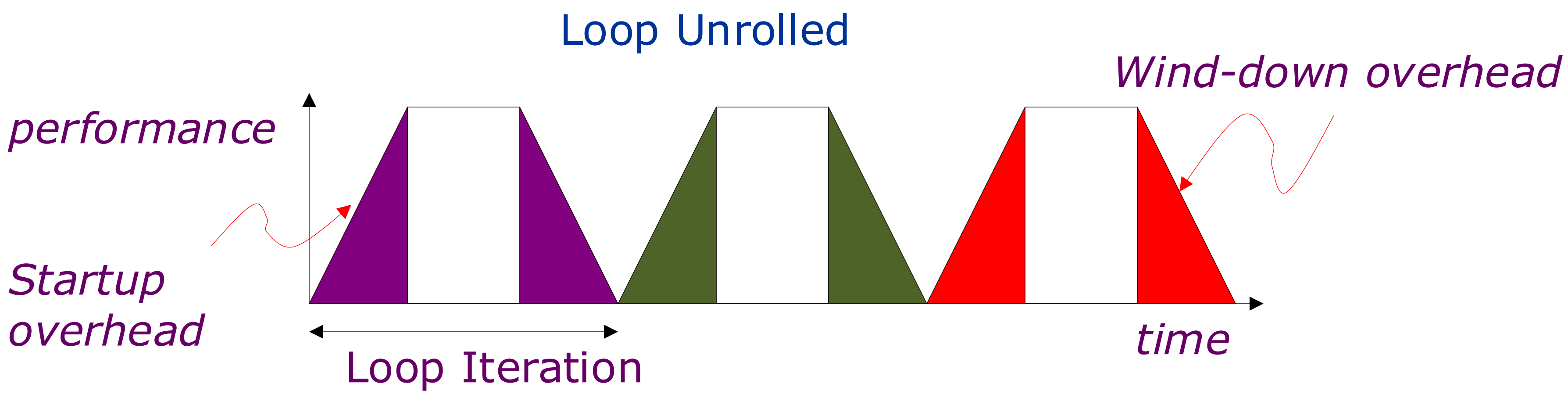
```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        add r2, 32
        sd f8, -8(r2)
        bne r1, r3, loop
```

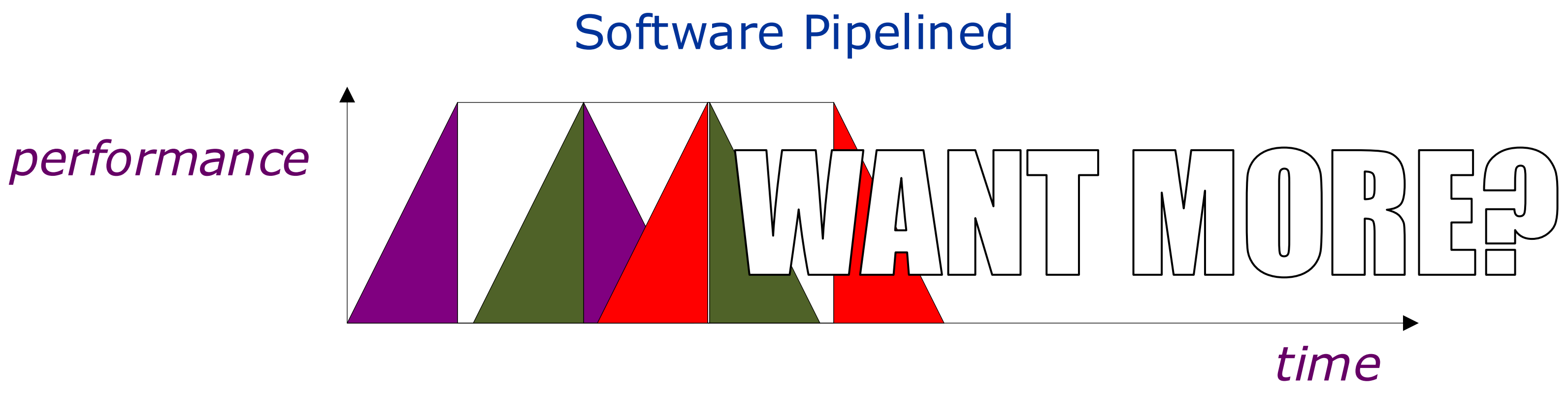
Schedule

loop:

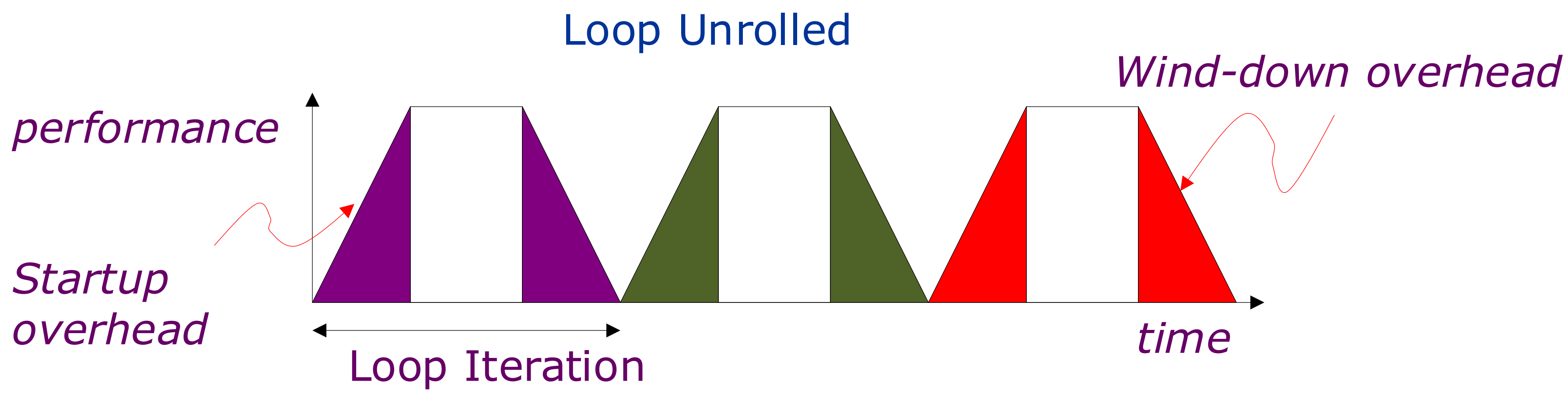
Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4			
		ld f1		fadd f5	
		ld f2		fadd f6	
		ld f3		fadd f7	
add r1		ld f4		fadd f8	
		ld f1	sd f5	fadd f5	
		ld f2	sd f6	fadd f6	
	add r2	ld f3	sd f7	fadd f7	
add r1	bne	ld f4	sd f8	fadd f8	
			sd f5	fadd f5	
			sd f6	fadd f6	
	add r2		sd f7	fadd f7	
	bne		sd f8	fadd f8	
			sd f5		

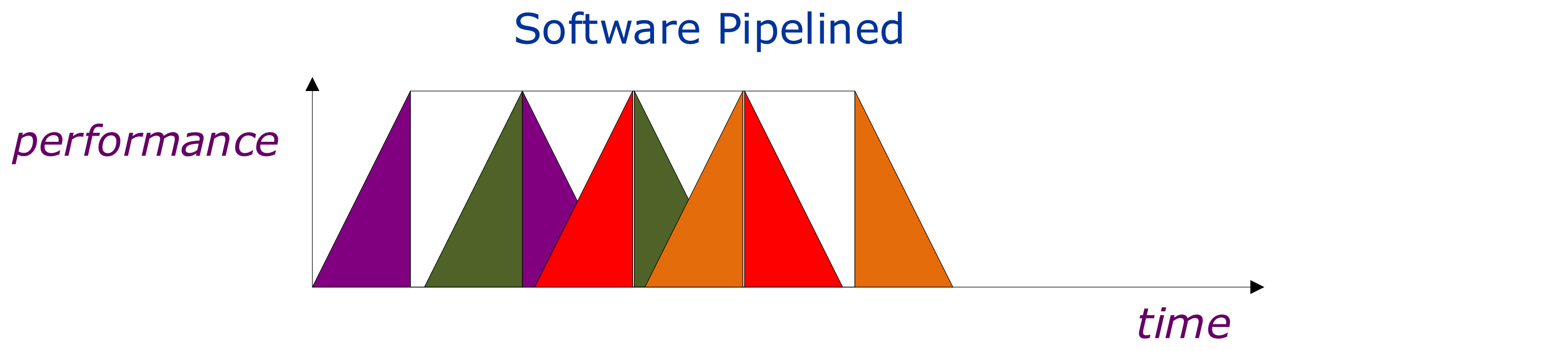
Loop Unrolling vs. Software Pipelining



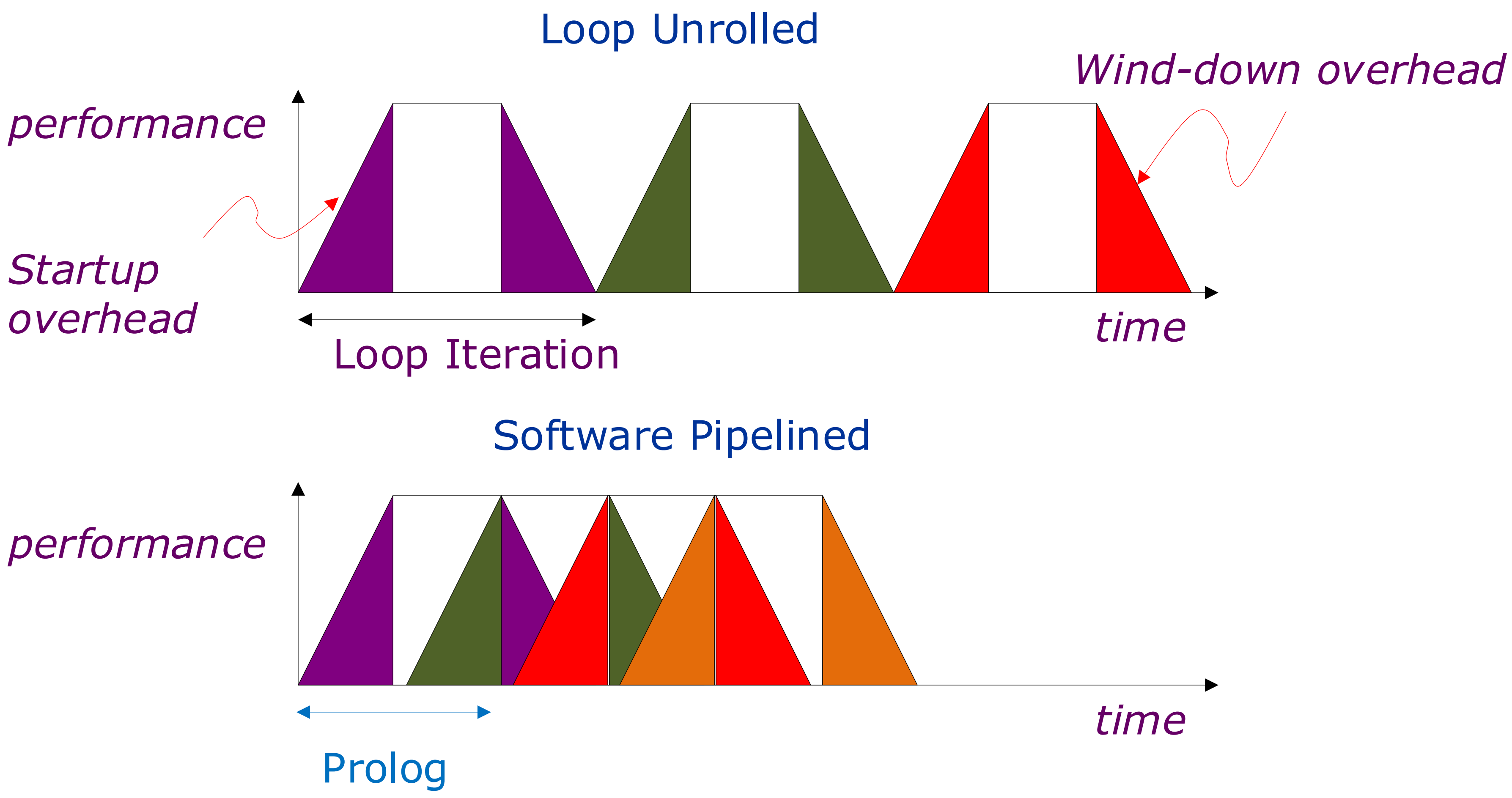


Loop Unrolling vs. Software Pipelining





Loop Unrolling vs. Software Pipelining



Software Pipelining

Unroll 4 ways first

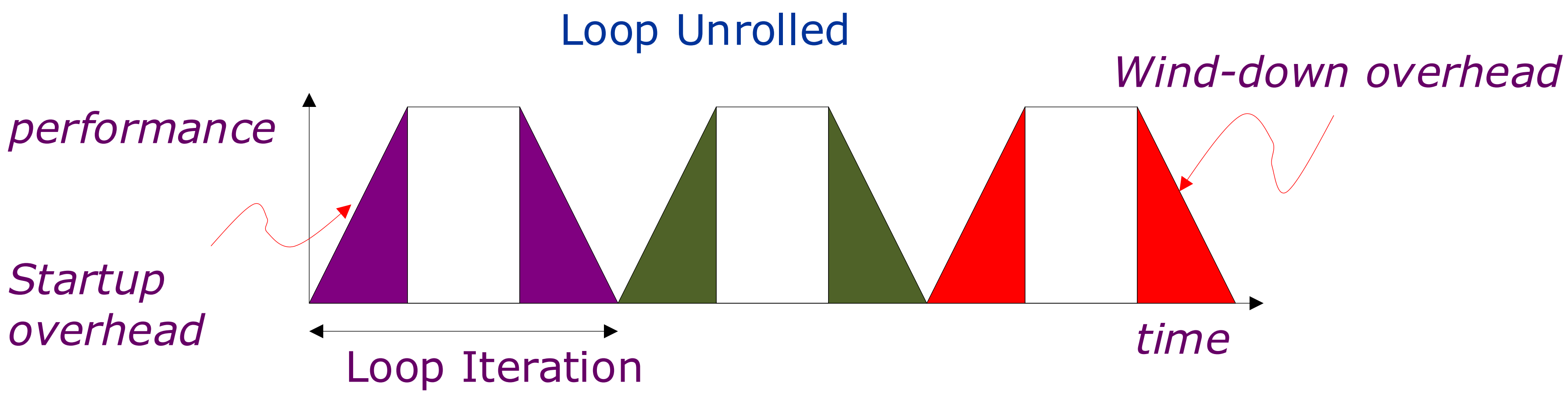
```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        add r2, 32
        sd f8, -8(r2)
        bne r1, r3, loop
```

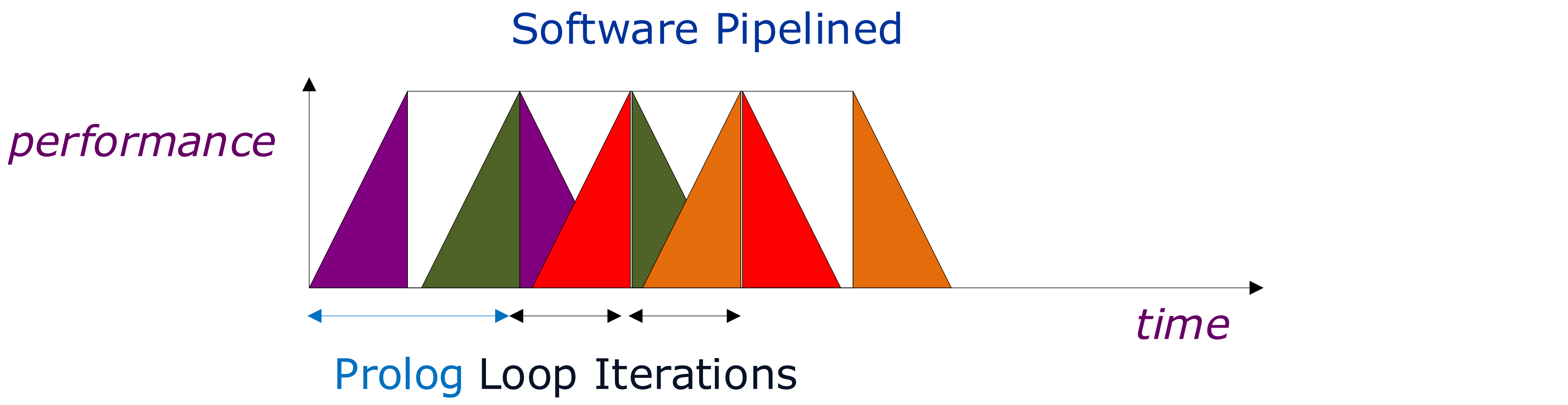
Prolog

Schedule

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4			
		ld f1		fadd f5	
		ld f2		fadd f6	
		ld f3		fadd f7	
add r1		ld f4		fadd f8	
		ld f1	sd f5	fadd f5	
		ld f2	sd f6	fadd f6	
	add r2	ld f3	sd f7	fadd f7	
add r1	bne	ld f4	sd f8	fadd f8	
			sd f5	fadd f5	
			sd f6	fadd f6	
	add r2		sd f7	fadd f7	
	bne		sd f8	fadd f8	
			sd f5		

Loop Unrolling vs. Software Pipelining





Software Pipelining

Unroll 4 ways first

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        add r2, 32
        sd f8, -8(r2)
        bne r1, r3, loop
```

Schedule

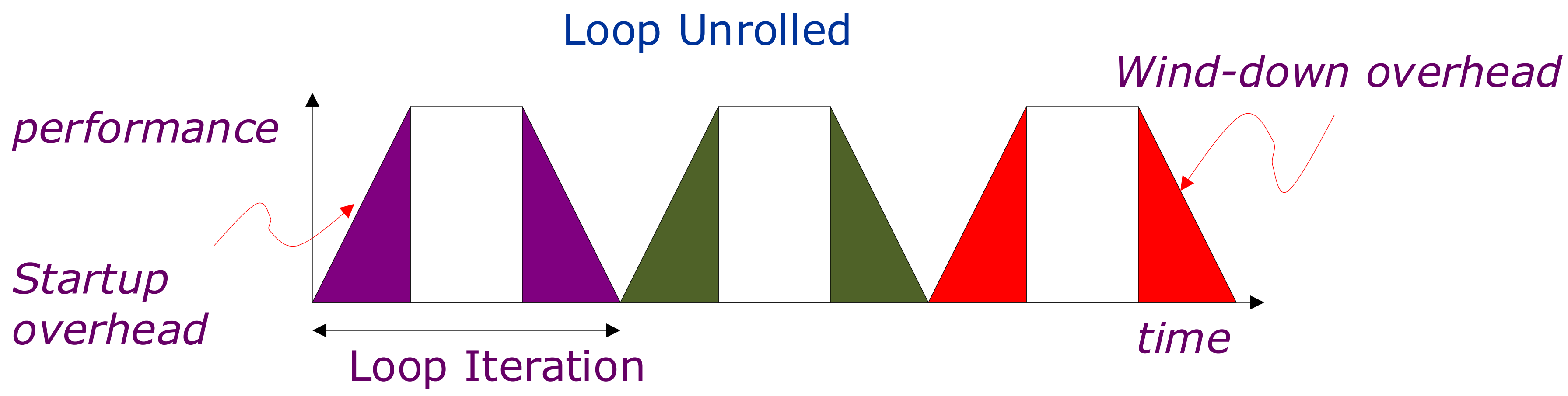
Prolog

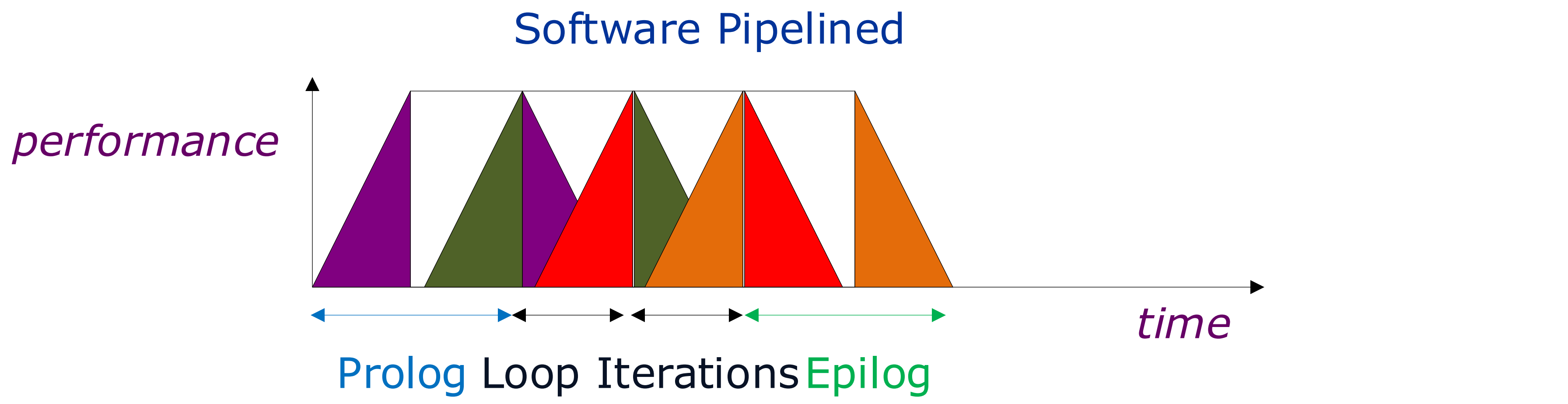
loop:

iterate

	Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
Prolog			ld f1			
			ld f2			
			ld f3			
	add r1		ld f4			
			ld f1		fadd f5	
			ld f2		fadd f6	
			ld f3		fadd f7	
	add r1		ld f4		fadd f8	
loop: iterate			ld f1	sd f5	fadd f5	
			ld f2	sd f6	fadd f6	
		add r2	ld f3	sd f7	fadd f7	
	add r1	bne	ld f4	sd f8	fadd f8	
				sd f5	fadd f5	
				sd f6	fadd f6	
		add r2		sd f7	fadd f7	
		bne		sd f8	fadd f8	
				sd f5		

Loop Unrolling vs. Software Pipelining





Software Pipelining

Unroll 4 ways first

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        add r2, 32
        sd f8, -8(r2)
        bne r1, r3, loop
```

Prolog

Schedule

loop:

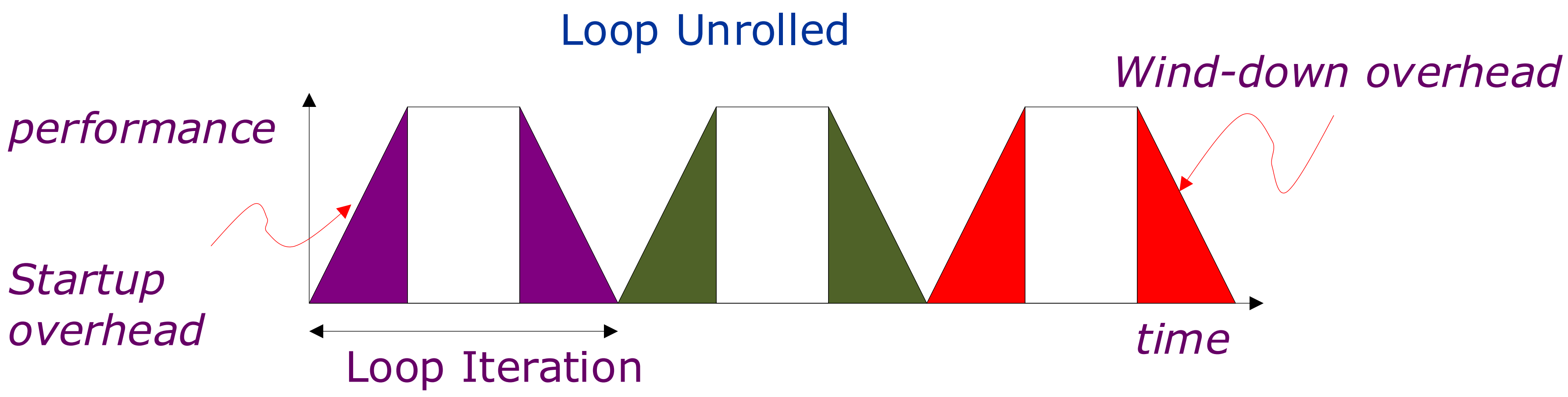
iterate

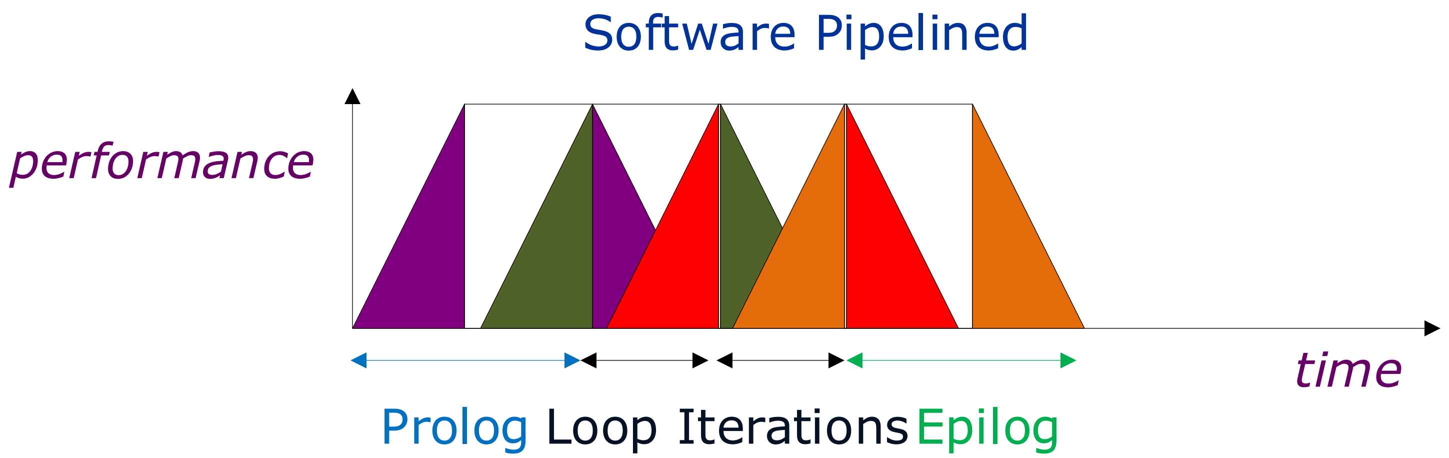
Epilog

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4			
		ld f1		fadd f5	
		ld f2		fadd f6	
		ld f3		fadd f7	
add r1		ld f4		fadd f8	
		ld f1	sd f5	fadd f5	
		ld f2	sd f6	fadd f6	
	add r2	ld f3	sd f7	fadd f7	
add r1	bne	ld f4	sd f8	fadd f8	
			sd f5	fadd f5	
			sd f6	fadd f6	
	add r2		sd f7	fadd f7	
	bne		sd f8	fadd f8	

sd f5

Loop Unrolling vs. Software Pipelining





Software pipelining pays startup/wind-down costs only once per loop, not once per iteration

Software Pipelining

Unroll 4 ways first

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        add r2, 32
        sd f8, -8(r2)
        bne r1, r3, loop
```

Prolog

Schedule

loop:

iterate

Epilog

Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4			
		ld f1		fadd f5	
		ld f2		fadd f6	
		ld f3		fadd f7	
add r1		ld f4		fadd f8	
		ld f1	sd f5	fadd f5	
		ld f2	sd f6	fadd f6	
	add r2	ld f3	sd f7	fadd f7	
add r1	bne	ld f4	sd f8	fadd f8	
			sd f5	fadd f5	
			sd f6	fadd f6	
	add r2		sd f7	fadd f7	
	bne		sd f8	fadd f8	
			sd f5		

How many FLOPS/cycle?

Software Pipelining

Unroll 4 ways first

```
loop:   ld f1, 0(r1)
        ld f2, 8(r1)
        ld f3, 16(r1)
        ld f4, 24(r1)
        add r1, 32
        fadd f5, f0, f1
        fadd f6, f0, f2
        fadd f7, f0, f3
        fadd f8, f0, f4
        sd f5, 0(r2)
        sd f6, 8(r2)
        sd f7, 16(r2)
        add r2, 32
        sd f8, -8(r2)
        bne r1, r3, loop
```

Prolog

Schedule

loop:

iterate

Epilog

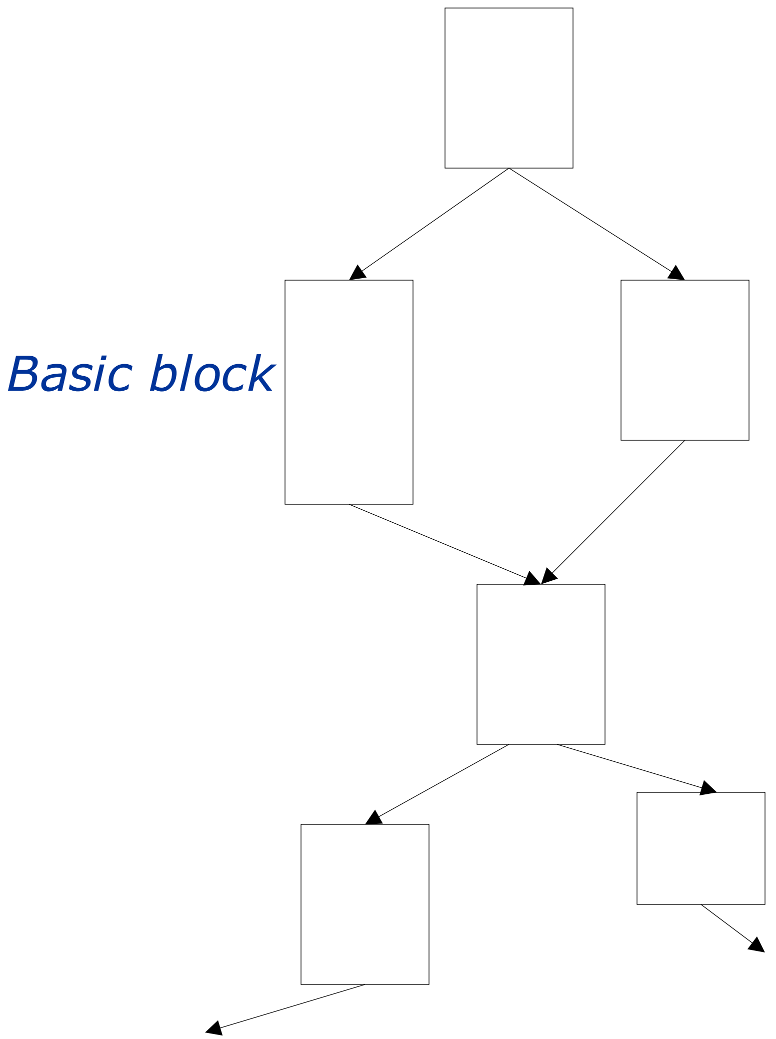
Int1 1cc	Int 2 1cc	M1 3cc	M2 3cc	FP+ 4cc	FPx 4cc
		ld f1			
		ld f2			
		ld f3			
add r1		ld f4			
		ld f1		fadd f5	
		ld f2		fadd f6	
		ld f3		fadd f7	
add r1		ld f4		fadd f8	
		ld f1	sd f5	fadd f5	
		ld f2	sd f6	fadd f6	
	add r2	ld f3	sd f7	fadd f7	
add r1	bne	ld f4	sd f8	fadd f8	
			sd f5	fadd f5	
			sd f6	fadd f6	
	add r2		sd f7	fadd f7	
	bne		sd f8	fadd f8	

sd f5

How many FLOPS/cycle?

4 fadds / 4 cycles = 1

What if there are no loops?



- Branches limit basic block size in control-flow intensive irregular code
- Difficult to find ILP in individual basic blocks

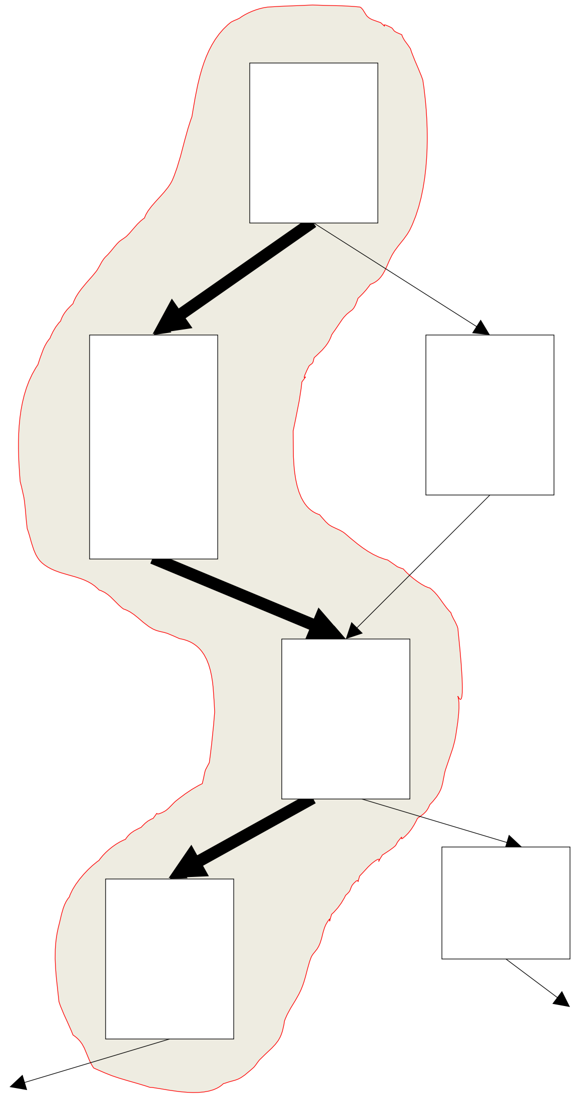
Basic block: a sequence of straight non-branch instructions

Trace scheduling: basic idea

- Trace scheduling focuses on traces
  - A trace is a loop-free sequence of basic blocks embedded in the control flow graph (Fisher)
  - It is an execution path which can be taken for some set of inputs
  - The chances that a trace is actually executed depends on the input set that allows its execution

- Some traces are executed much more frequently than others

Trace Scheduling *[Fisher,Ellis]*



- Pick string of basic blocks, a trace, that represents most frequent branch path
- Use <u>profiling feedback</u> or compiler heuristics to find common branch paths
- Schedule whole “trace” at once
- Add fixup code to cope with branches jumping out of trace

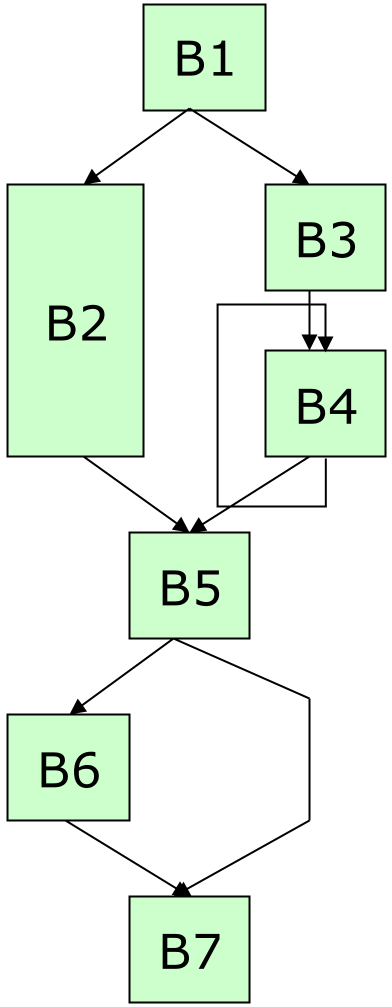
Trace scheduling and loops

- Trace scheduling and loops
  - Trace scheduling cannot proceed beyond a loop barrier
  - Techniques used to overcome this limitation are based on loop unrolling
- Negative effects on unrolling
  - Unrolling produces much extra code
  - It also looses performance, because of the costs of starting and closing the iterations
- Traces scheduling schedules traces in order of decreasing probability of being executed
  - So, most frequently executed traces get better schedules
  - Traces are scheduled as if they were basic blocks (no special considerations for branches)

Trace Scheduling: in deep

- Trace: a sequence of instructions which may Include branches but not including loops

- For example some traces in the control flow graph:
  - B1, B3
  - B4
  - B5, B7
  - B1, B2
  - B1, B2, B5, B6, B7

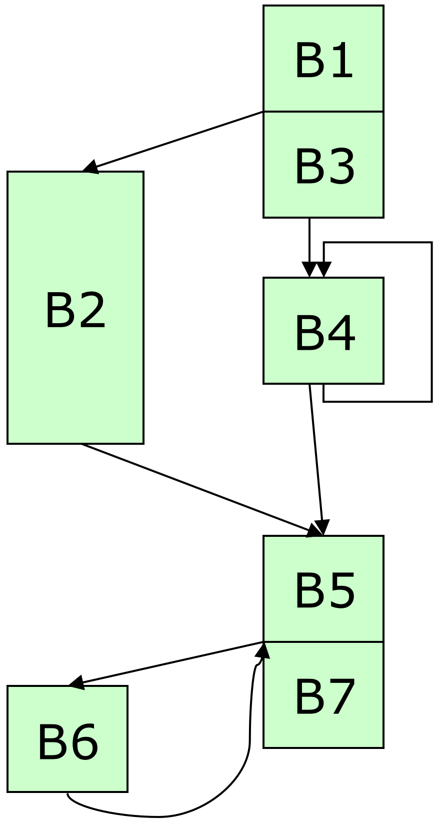


Trace Scheduling Cont’d

- Trace Scheduling: finding common path and scheduling traces in that path independently
- Scheduling in a trace rely on basic code motion but now has a global taste across more that one basic block by appropriate use of renaming
- Compensation codes are needed
  - for side entry points: i.e. points except beginning
  - and slide exit points: i.e. points except ending
- Blocks on non common path may now have added overhead, so there must be a high probability of taking common path according to profile (may not be clear for some programs)
- Problems: compensation codes are difficult to generate specially for entry points

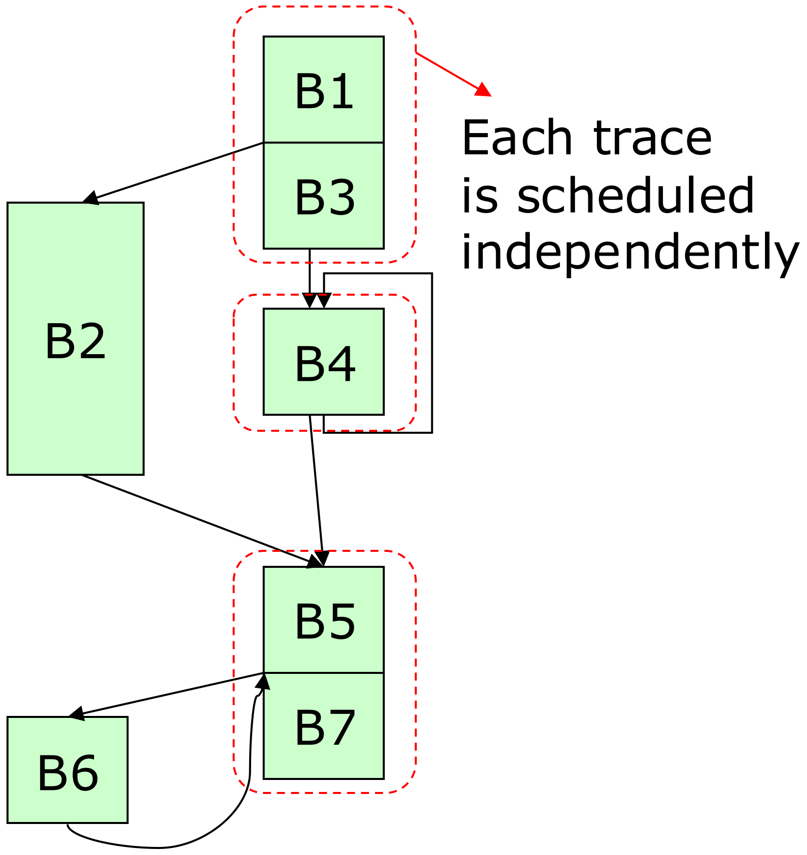
Trace Scheduling Example

- For example suppose that B1,B3,B4,B5,B7 is the most frequently executed path
- Therefore traces are
  - B1,B3
  - B4
  - B5,B7



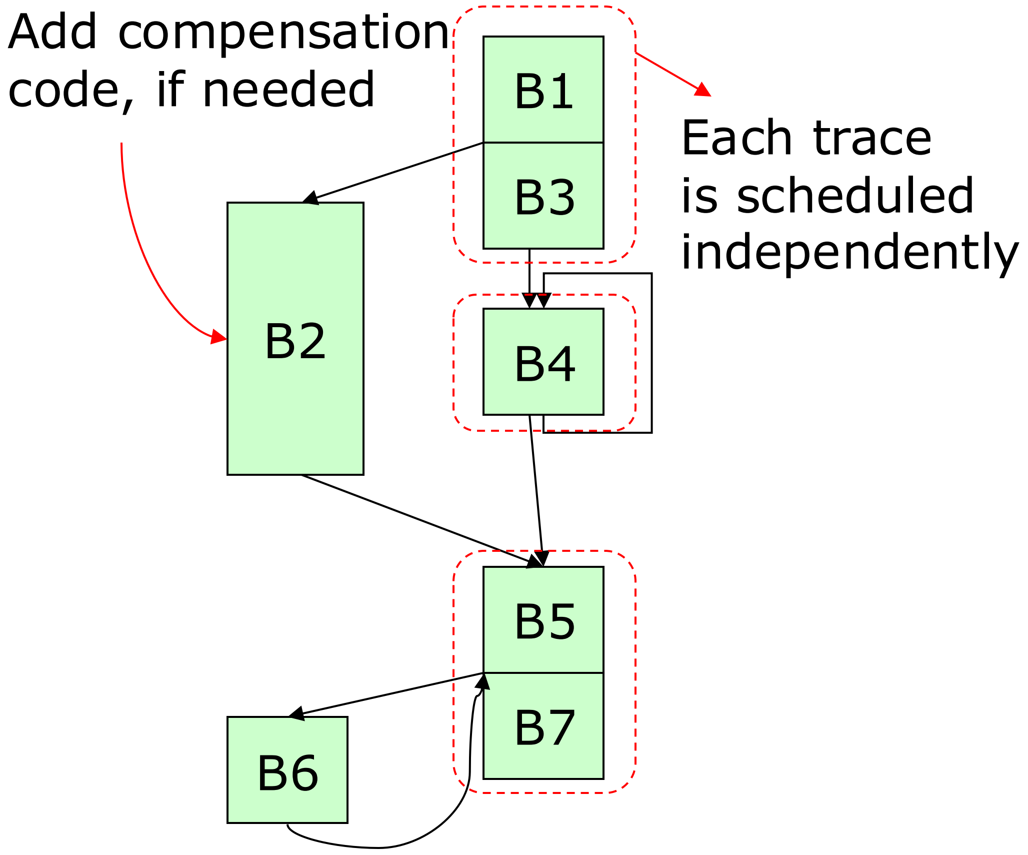
Trace Scheduling Example

- For example suppose that B1,B3,B4,B5,B7 is the most frequently executed path
- Therefore traces are
  - B1,B3
  - B4
  - B5,B7



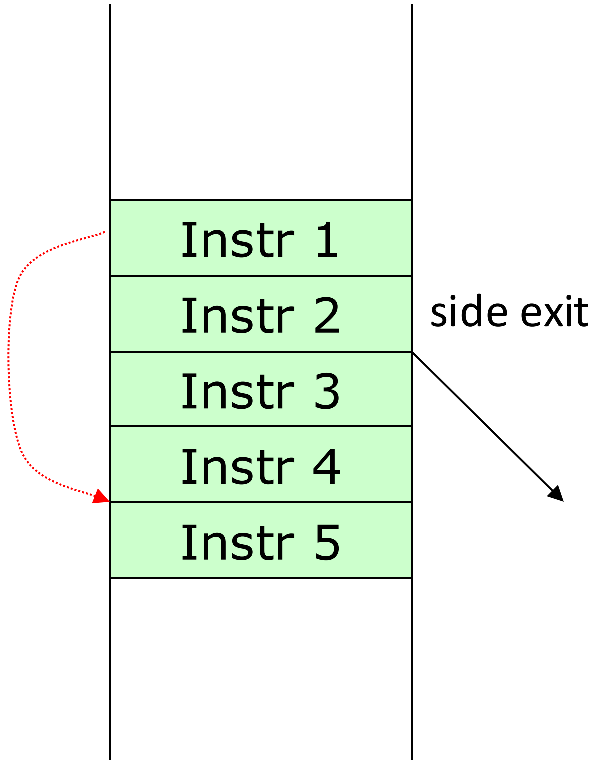
Trace Scheduling Example

- For example suppose that B1,B3,B4,B5,B7 is the most frequently executed path
- Therefore traces are
  - B1,B3
  - B4
  - B5,B7


Add compensation code, if needed
Each trace is scheduled independently
B1
B3
B2
B4
B5
B7
B6

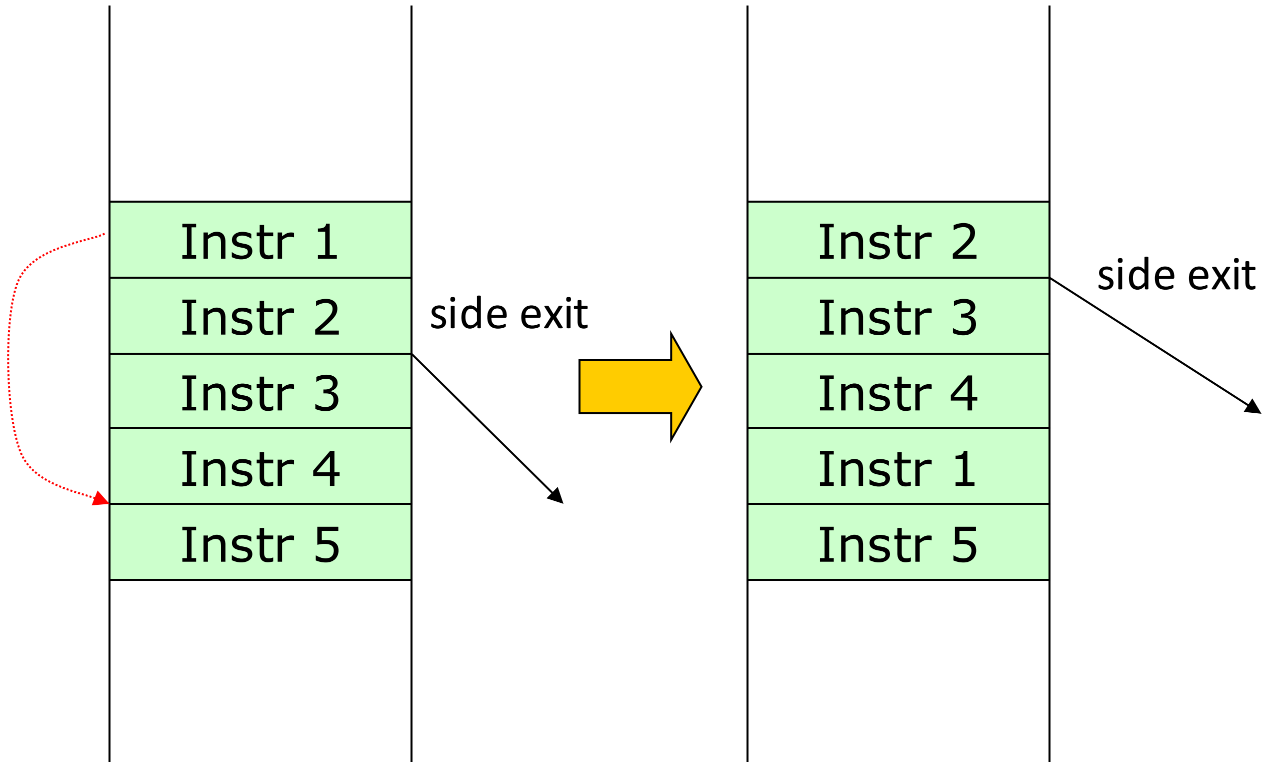
Trace Scheduling Compensation 1

Moving an instruction below a side exit (simple)



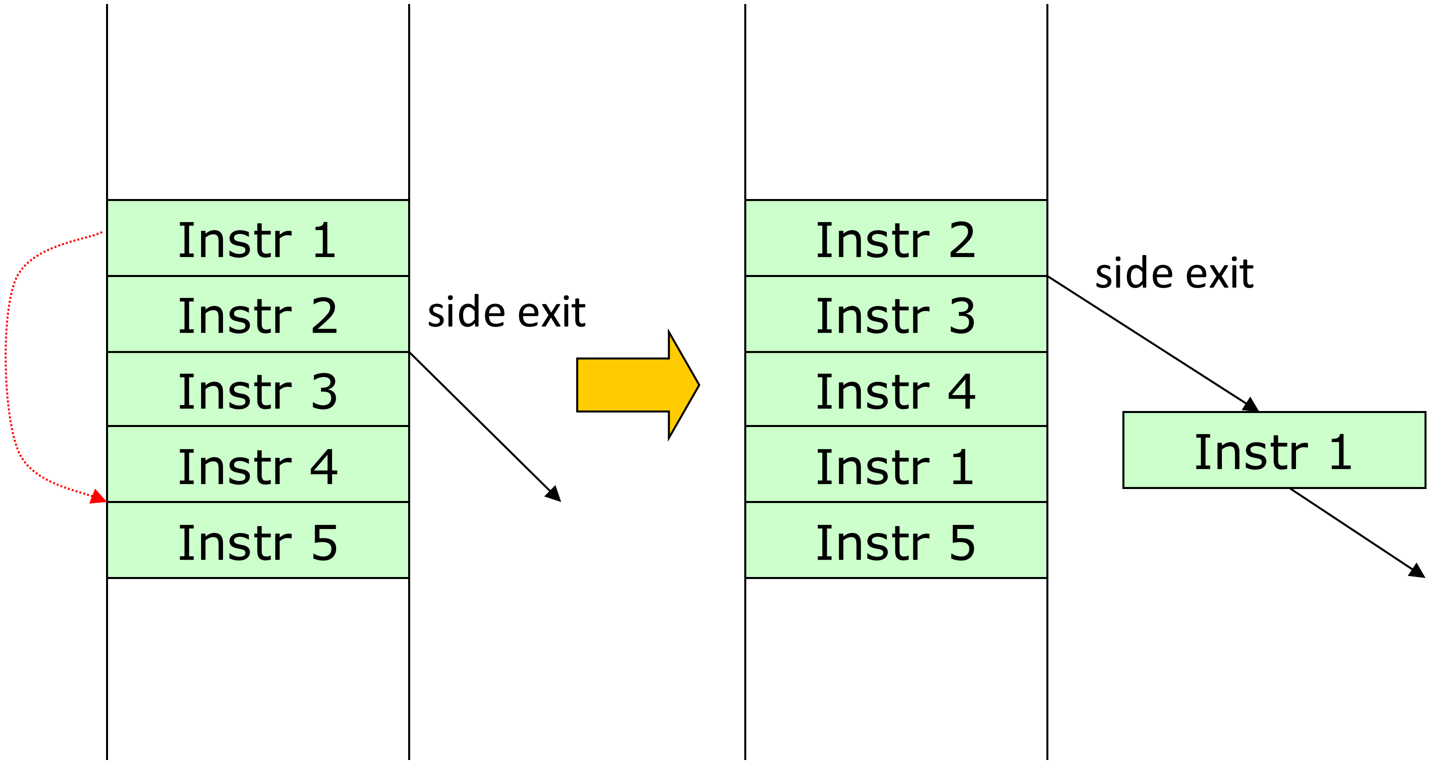
Trace Scheduling Compensation 1

Moving an instruction below a side exit (simple)



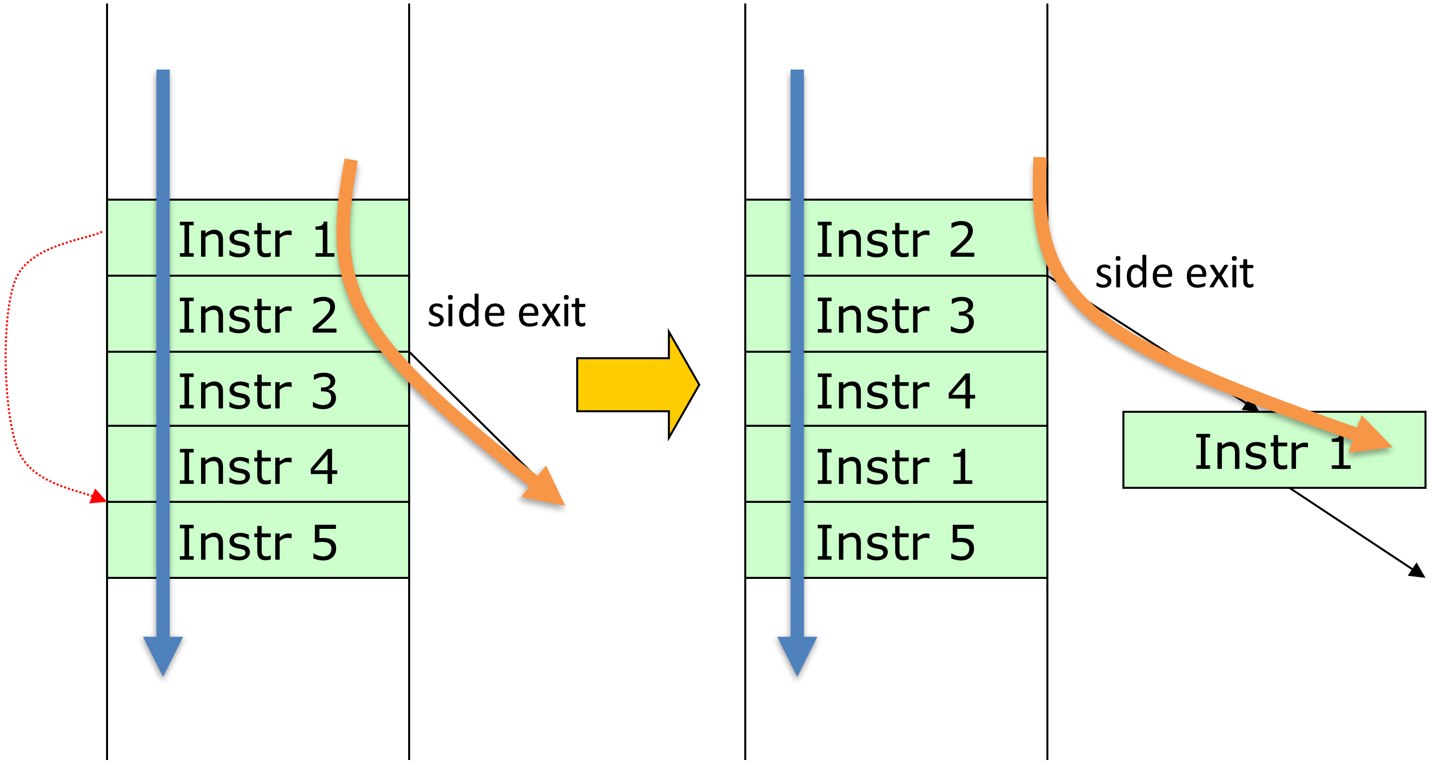
Trace Scheduling Compensation 1

Moving an instruction below a side exit (simple)



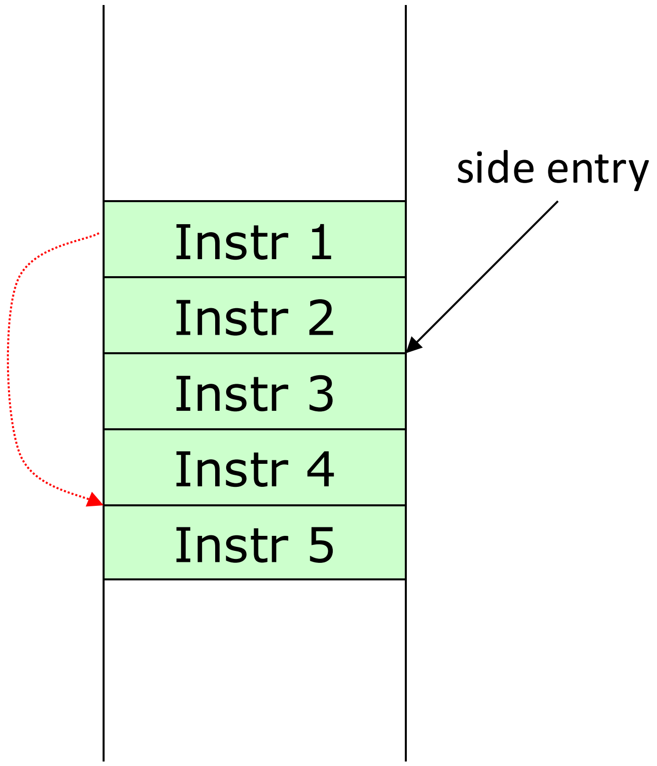
Trace Scheduling Compensation 1

Moving an instruction below a side exit (simple)



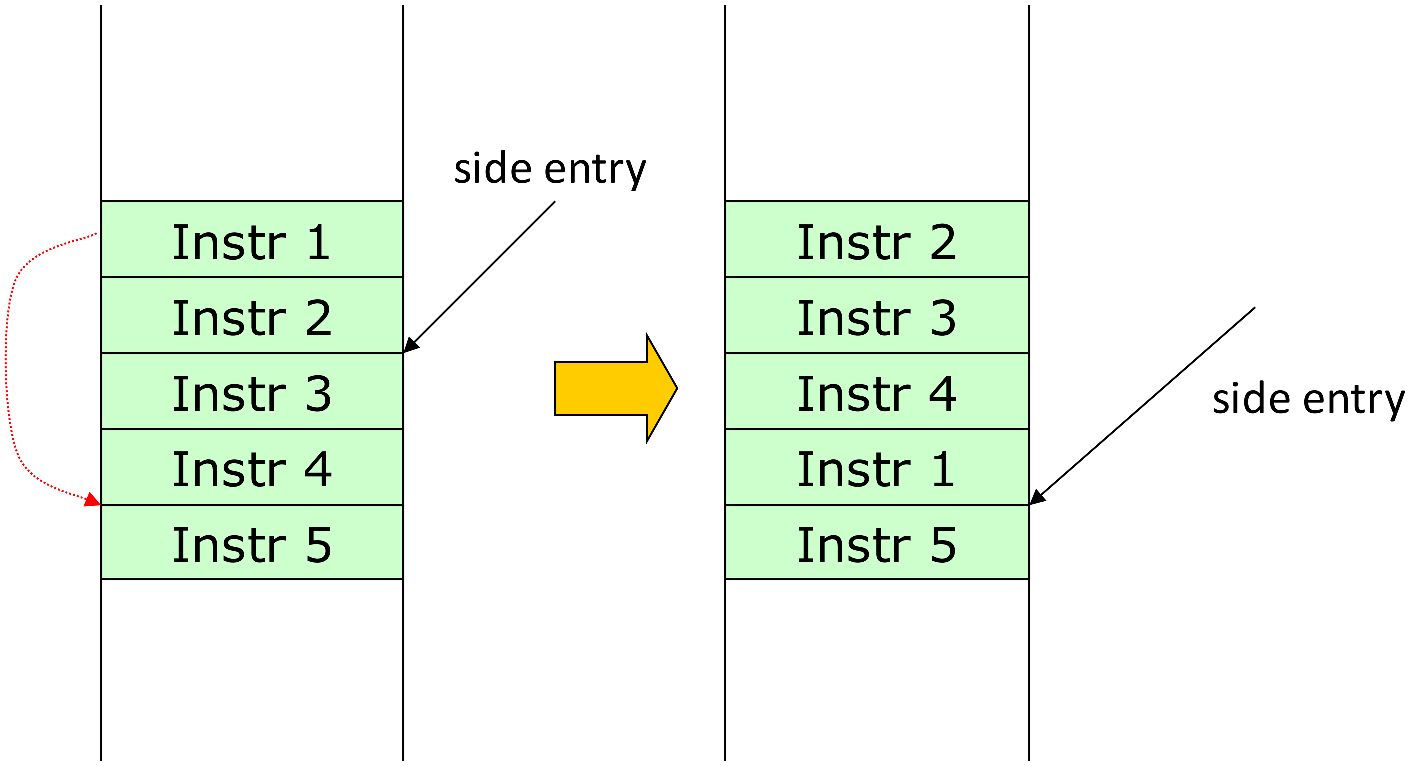
Trace Scheduling Compensation 2

Moving an instruction below a side entry (complex)



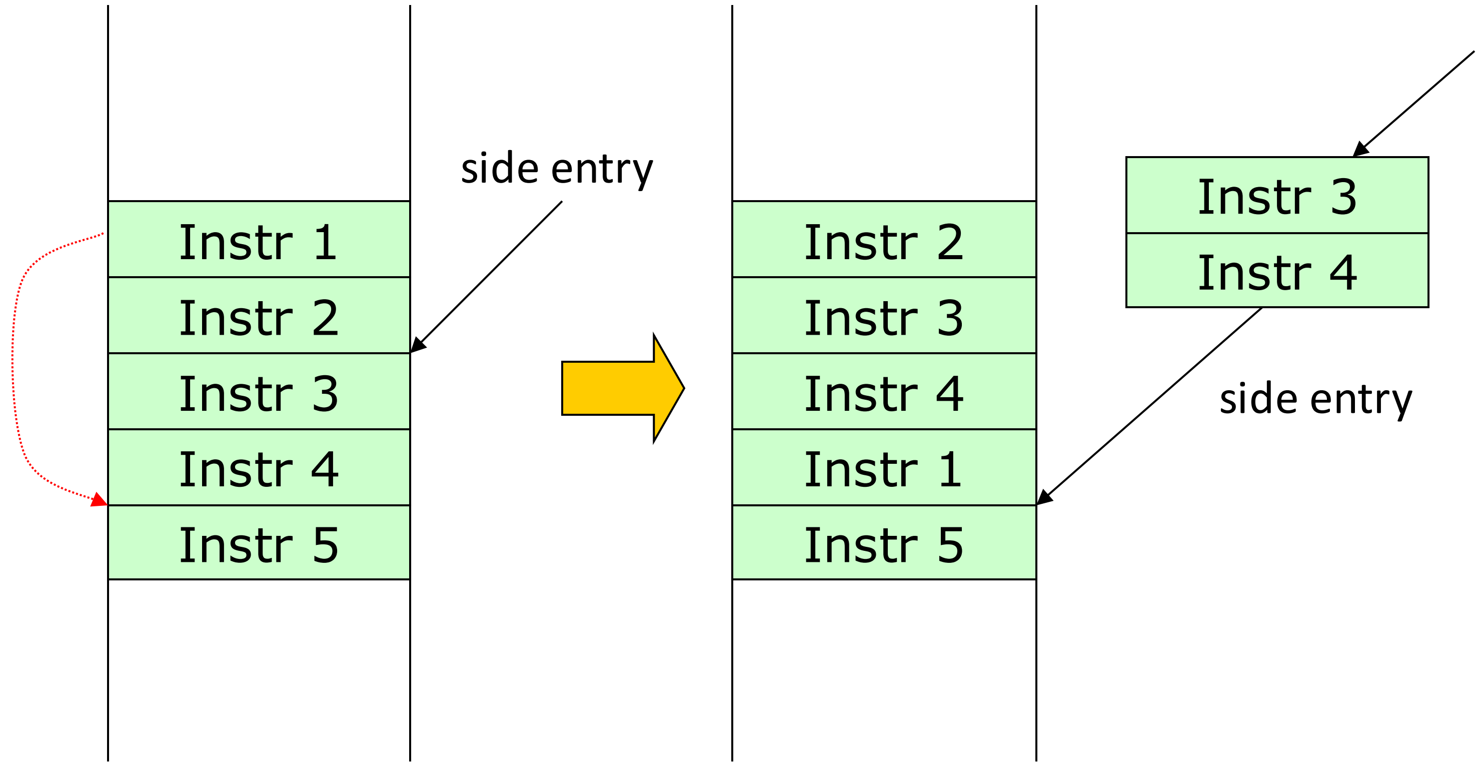
Trace Scheduling Compensation 2

Moving an instruction below a side entry (complex)



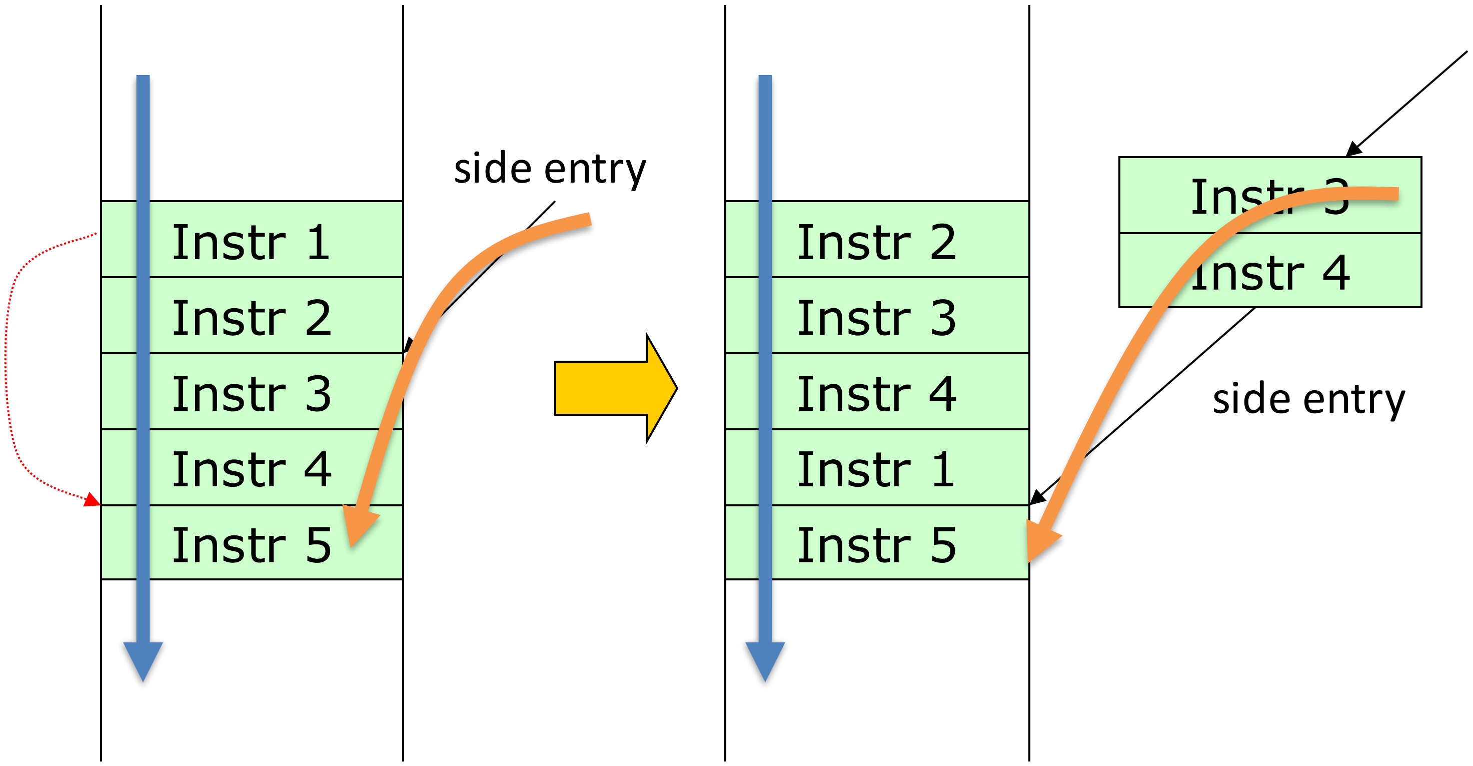
Trace Scheduling Compensation 2

Moving an instruction below a side entry (complex)



Trace Scheduling Compensation 2

Moving an instruction below a side entry (complex)



Code Motion in Trace Scheduling

- In addition to need of compensation codes there are restrictions on movement of a code in a trace:
  - The dataflow of the program must not change
  - The exception behavior must be preserved
- Dataflow can be guaranteed to be correct by maintaining two dependencies:
  - Data dependency
  - Control dependency
- There are two solutions to eliminate control dependency:
  - By use of predicate instructions (Hyperblock scheduling) and removing the branch.
  - By use of speculative instructions (Speculative Scheduling) and speculatively move an instruction before the branch.

Code Motion in Trace Scheduling

- In addition to need of compensation codes there are restrictions on movement of a code in a trace:
  - The dataflow of the program must not change
  - The exception behavior must be preserved
- Dataflow can be guaranteed to be correct by maintaining two dependencies:
  - Data dependency
  - Control dependency
- There are two solutions to eliminate control dependency:
  - By use of predicate instructions (Hyperblock scheduling) and removing the branch.
  - By use of speculative instructions (Speculative Scheduling) and speculatively move an instruction before the branch.

Code Motion in Trace Scheduling

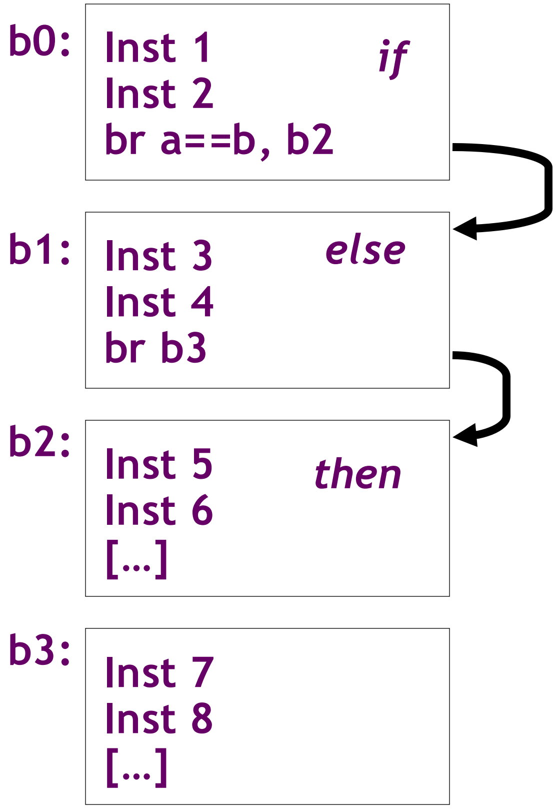
- In addition to need of compensation codes there are restrictions on movement of a code in a trace:
  - The dataflow of the program must not change
  - The exception behavior must be preserved
- Dataflow can be guaranteed to be correct by maintaining two dependencies:
  - Data dependency
  - Control dependency
- There are two solutions to eliminate control dependency:
  - By use of predicate instructions (Hyperblock scheduling) and removing the branch.
  - By use of speculative instructions (Speculative Scheduling) and speculatively move an instruction before the branch.

Predicated Execution

Problem: Mispredicted branches limit ILP

Solution: Eliminate hard to predict branches with predicated execution

- Almost all IA-64 instructions can be executed conditionally under predicate
- Instruction becomes NOP if predicate register false



```
b0: Inst 1
    Inst 2
    br a==b, b2

b1: Inst 3
    Inst 4
    br b3

b2: Inst 5
    Inst 6
    […]

b3: Inst 7
    Inst 8
    […]
```

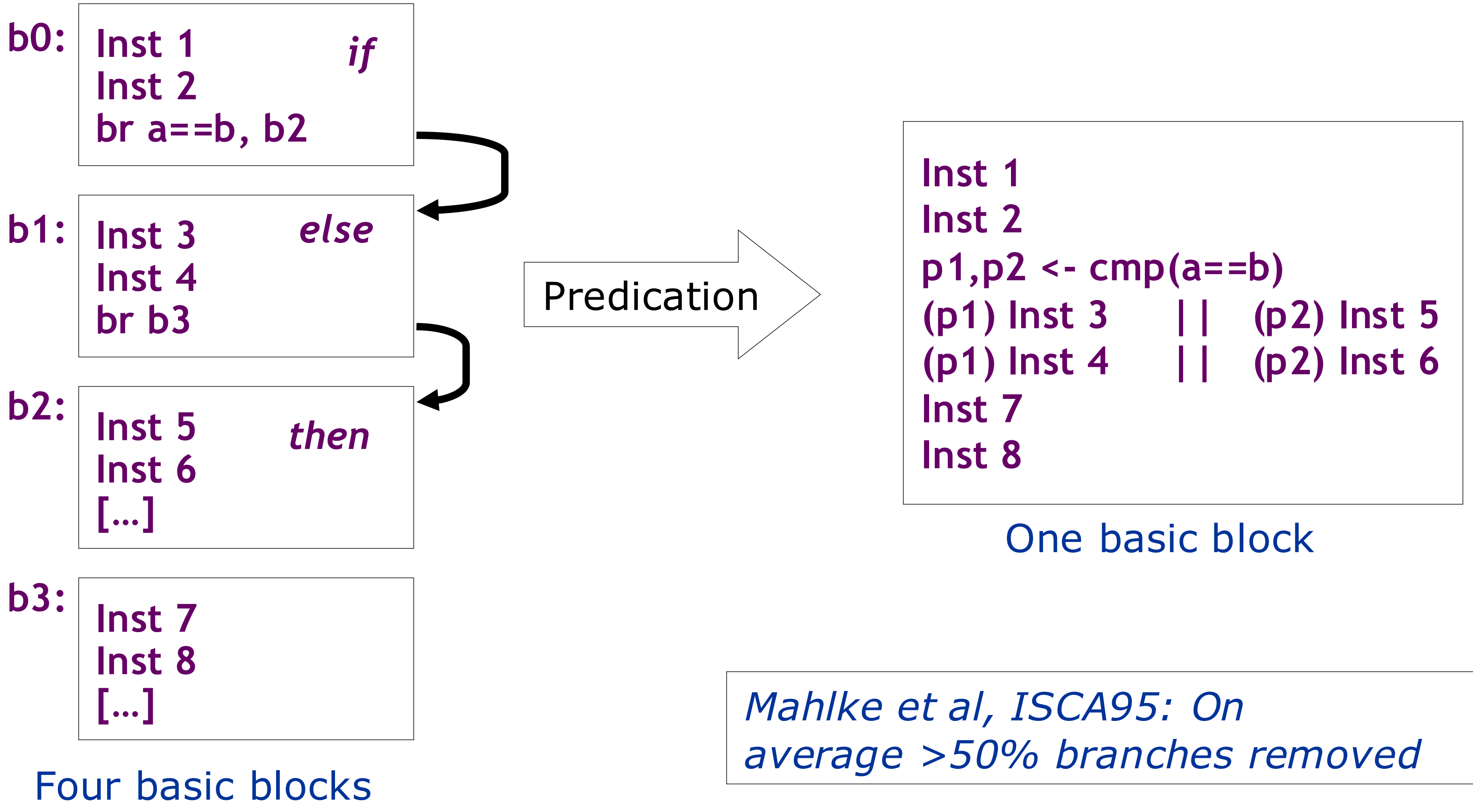
Four basic blocks

Predicated Execution

Problem: Mispredicted branches limit ILP

Solution: Eliminate hard to predict branches with predicated execution

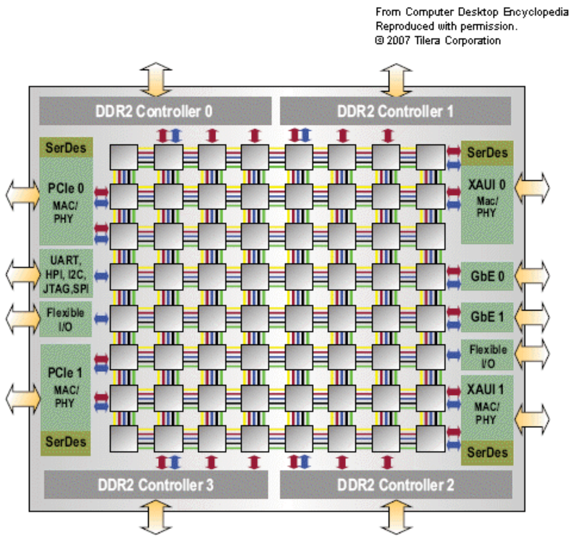
- Almost all IA-64 instructions can be executed conditionally under predicate
- Instruction becomes NOP if predicate register false



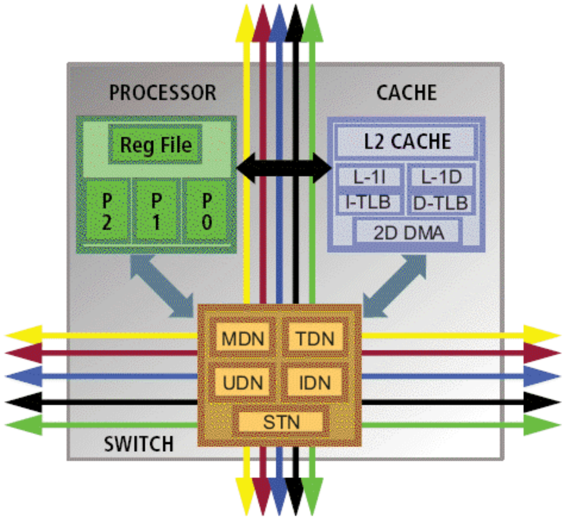
Mahlke et al, ISCA95: On average >50% branches removed

END...

mit raw


From Computer Desktop Encyclopedia
Reproduced with permission.
© 2007 Tilera Corporation
DDR2 Controller 0
DDR2 Controller 1
DDR2 Controller 3
DDR2 Controller 2
SerDes
PCIe 0 MAC/ PHY
UART, HPI, I2C, JTAG,SPI
Flexible I/O
PCIe 1 MAC/ PHY
XAUI 0 Mac/ PHY
GbE 0
GbE 1
XAUI 1 MAC/ PHY

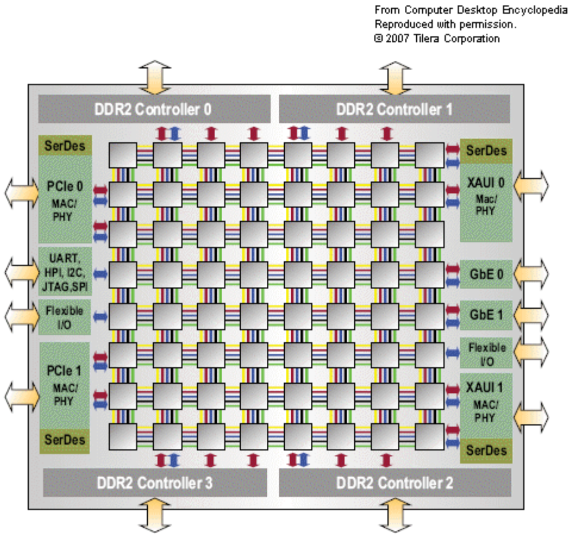
TILERA

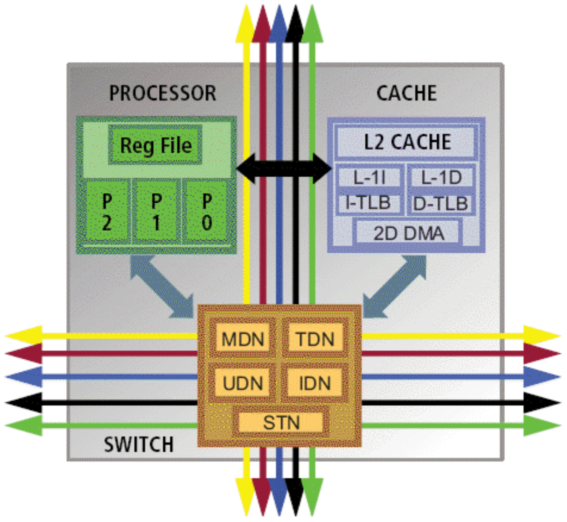

PROCESSOR
CACHE
Reg File
P 2
P 1
P 0
L2 CACHE
L-1I
L-1D
I-TLB
D-TLB
2D DMA
MDN
TDN
UDN
IDN
STN
SWITCH

TILERA®

END...

T-SHIRT ;)





TILERA®