Lecture R:

Instruction Scheduling *(Backend 2)*

CSE401/501m:

Introduction to Compiler Construction

Instructor: Gilbert Bernstein

Administrivia

- Final Project Reports — due Tuesday night
- HW4 — due Thursday night
- Lectures this week
    - Wednesday is *Gilbert's* last lecture (Reg. Allocation)
    - Friday is a guest lecture from Hal (Garbage Collection)
    - Both cover material that may be on the final
- Please nominate great TAs for the Bandes award!

Administrivia

- Past Finals on website
  - (Gilbert is behind; will get those up tonight)
- **Final Review — G-01 2:30-3:30pm next MONDAY**
- Closed Book exam (like midterm) — You can have ***two*** 5x8 index cards
- Sections this week will do last minute hw4 Q&A, some backend review and whatever your instructors think would be helpful
- **Exam — next TUESDAY (this room) 2:30pm**

Outline

Instruction Scheduling

Dependency Analysis

List Scheduling Algorithm

Beyond Basic Blocks

Outline

Instruction Scheduling

Dependency Analysis

List Scheduling Algorithm

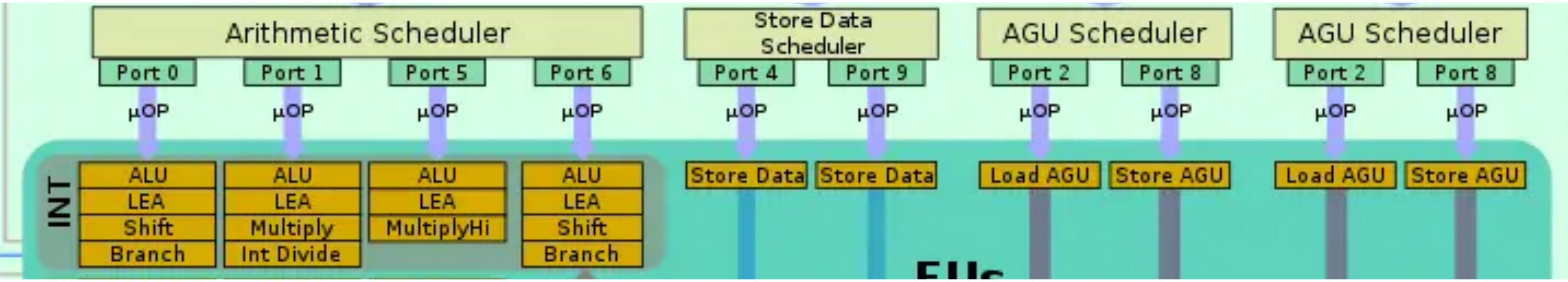
Beyond Basic Blocks

Instruction Scheduling

- Goal — within a basic block, reorder the sequence of instructions to improve…
    - execution time
        - by hiding load/store latencies & operation latencies
        - by maximizing parallelism
        - by keeping the CPU pipeline full of work
    - space usage (i.e. prevent register spilling)

Ice Lake (Sunny Cove) Instruction Latencies

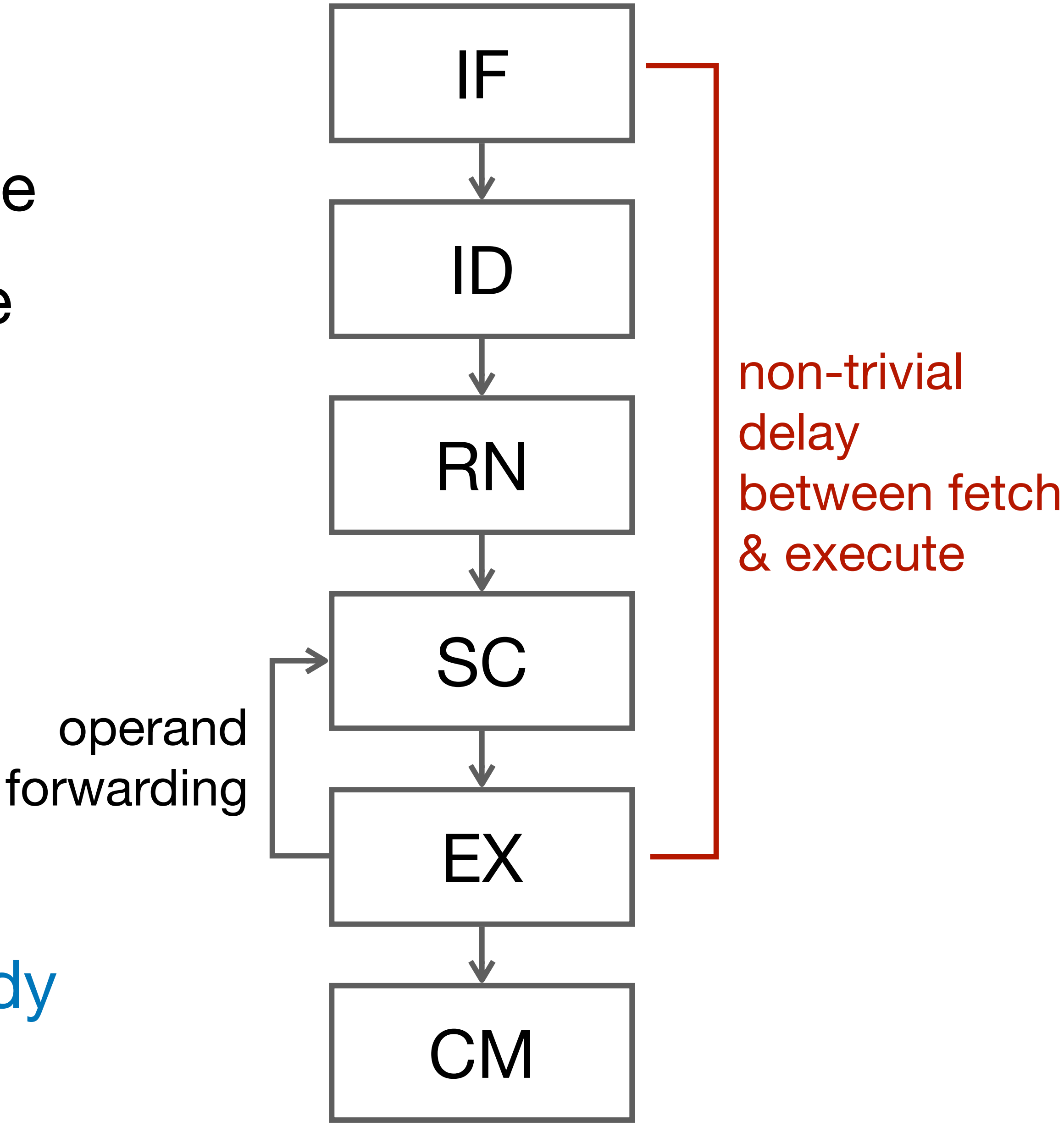
Instruction	port	latency	comments
MOV	0,1,5,6	1	reg move
MOV	2,3	3	load/store
ADD, SUB	0,1,5,6	1	
IMUL	1,5	3	
IDIV	0,1,5,6	15	
TEST, CMP	0,1,5,6	1	
LEA	0,1,5,6	1	
POP	2,3	3	
PUSH	4,9,7,8	3	



data reinterpreted & simplified, from Agner Fog
(I likely introduced errors from simplification)

A Fictitious Proc. Pipeline

- IF — Instruction Fetch
- ID — Instruction Decode
- RN — Register Rename
- SC — Schedule
- EX — Execute
- CM — Commit



Takeaway

Instruction latencies are optimistic, based on steady state execution

Branch Prediction

- Absolutely critical for keeping modern processors with deep pipelines busy
- Does not exert strong influence on static scheduling *within* a basic block
- Branch prediction helps to expose larger instruction windows to the *dynamic scheduler* on the processor
- Modern branch predictors are very good!
- (not going to discuss much more)

A Fictitious Processor

Instructions

LOAD – 3-? cycles
STORE – 3-? cycles
FADD – 1 cycle
FSUB – 1 cycle
FMUL – 1 cycle
FDIV – 11 cycles

Instruction queue
Depth – assume ∞

Instruction issue
rate – 2 per cycle

Port 0	**Port 1**	**Port 2**
FADD	FADD	LOAD
FSUB	FSUB	STORE
FMUL	FMUL	
FDIV		

Summary of Key Points

- Latencies on executing different operations
  - affects load balancing and packing
- How many and which functional units can run in parallel?
  - affects what kind of instruction mix you want
- What is the issue rate?
  - How quickly can instructions get fetched & decoded?
- How big is the instruction window?
  - How many instructions can be queued up?
    (and potentially executed out of order)

Outline

Instruction Scheduling

Dependency Analysis

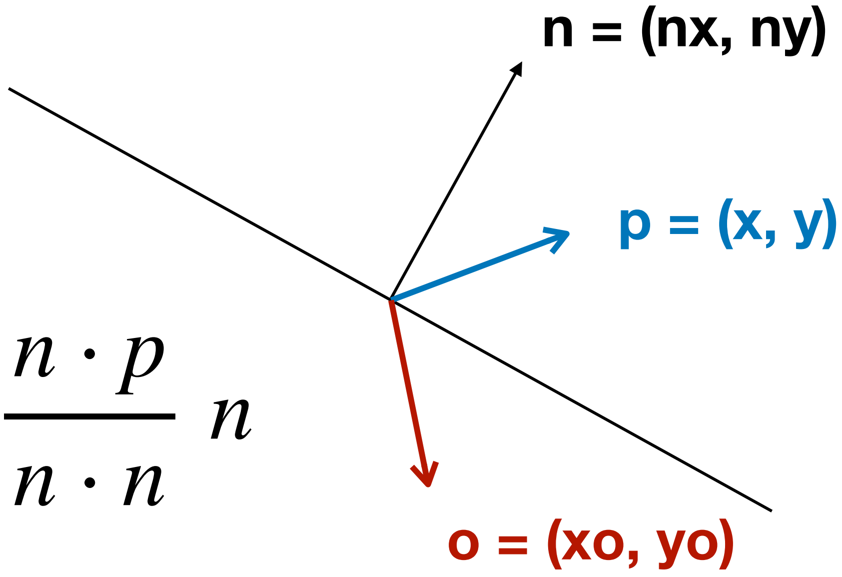
List Scheduling Algorithm

Beyond Basic Blocks

Example Program

```
LOAD   r1 ← x
LOAD   r2 ← y
LOAD   r3 ← nx
LOAD   r4 ← ny
FMUL   r5 ← r1 * r3
FMUL   r6 ← r2 * r4
FADD   r7 ← r5 + r6
FMUL   r8 ← r3 * r3
FMUL   r9 ← r4 * r4
FADD  r10 ← r8 + r9
FDIV  r11 ← 1.0 / r10
FMUL  r12 ← 2.0 * r7
FMUL  r13 ← r12 * r11
FMUL  r14 ← r3 * r13
FSUB  r15 ← r1 - r14
STORE  xo ← r15
FMUL  r16 ← r4 * r13
FSUB  r17 ← r2 - r16
STORE  yo ← r17
```

$$o = p - 2\,\frac{n \cdot p}{n \cdot n}\,n$$



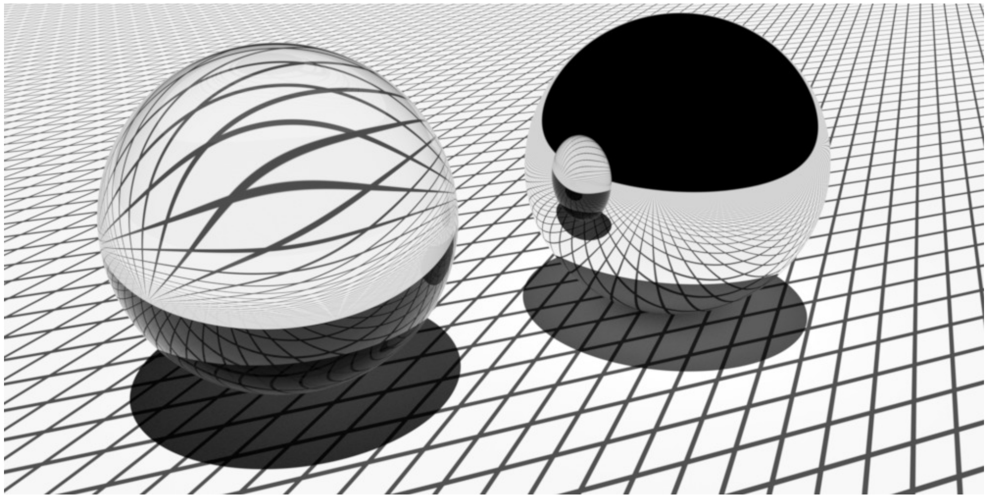


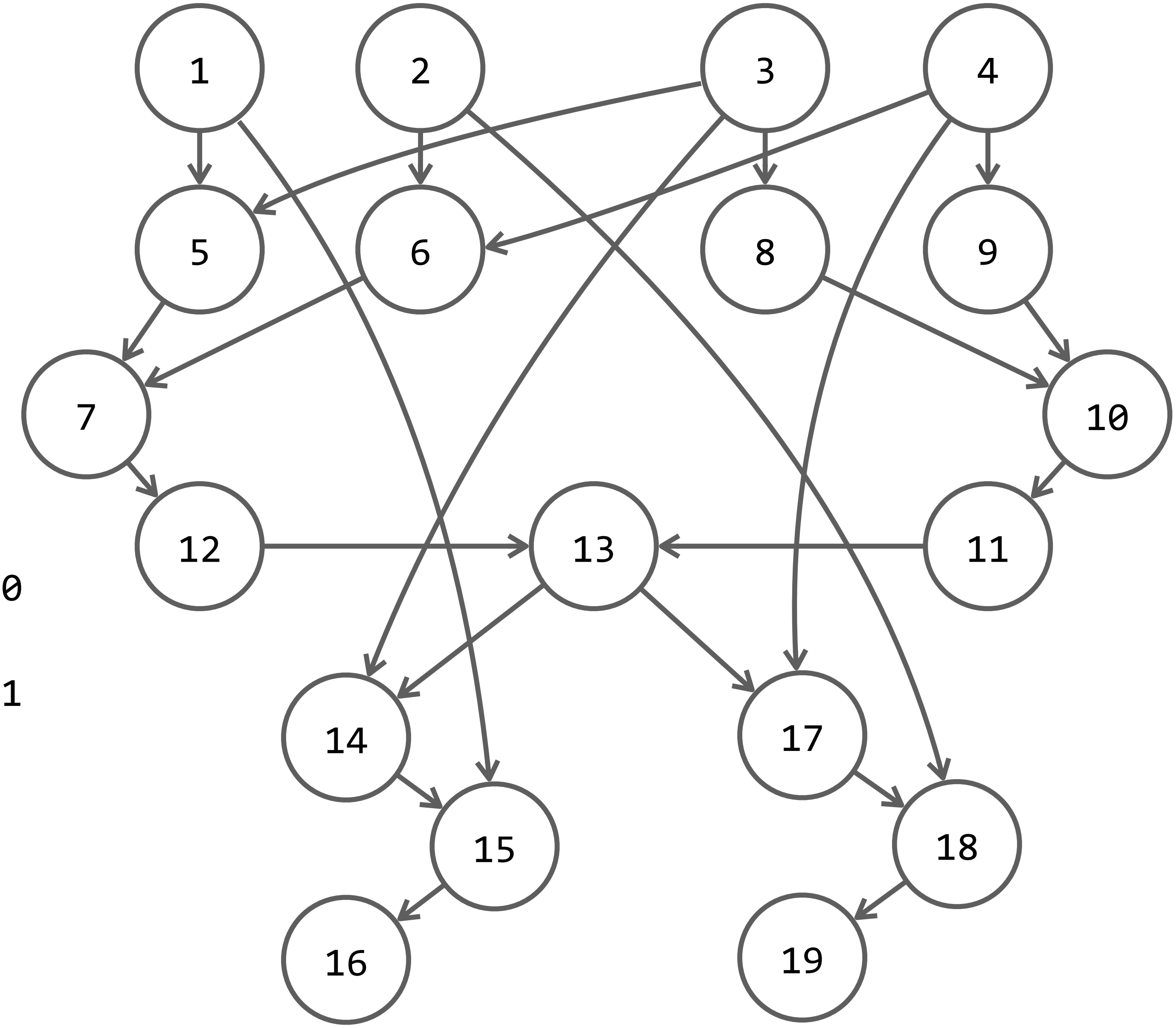
image from Physically Based Ray-Tracing book/site

Dependency Analysis

- Each operation **writes** to or **reads** from some number of memory resources (here, registers or memory)
- Imagine a single memory resource (one memory cell)
  - Each instruction is annotated with whether it reads, writes, or reads & writes the cell
- When is it ***not ok*** to reorder two instructions?
  - RAW — Read-after-Write
  - WAR — Write-after-Read
  - WAW — Write-after-Write

Example — Dependency Graph

```
 1 LOAD   r1 ← x
 2 LOAD   r2 ← y
 3 LOAD   r3 ← nx
 4 LOAD   r4 ← ny
 5 FMUL   r5 ← r1 * r3
 6 FMUL   r6 ← r2 * r4
 7 FADD   r7 ← r5 + r6
 8 FMUL   r8 ← r3 * r3
 9 FMUL   r9 ← r4 * r4
10 FADD  r10 ← r8 + r9
11 FDIV  r11 ← 1.0 / r10
12 FMUL  r12 ← 2.0 * r7
13 FMUL  r13 ← r12 * r11
14 FMUL  r14 ← r3 * r13
15 FSUB  r15 ← r1 - r14
16 STORE  xo ← r15
17 FMUL  r16 ← r4 * r13
18 FSUB  r17 ← r2 - r16
19 STORE  yo ← r17
```

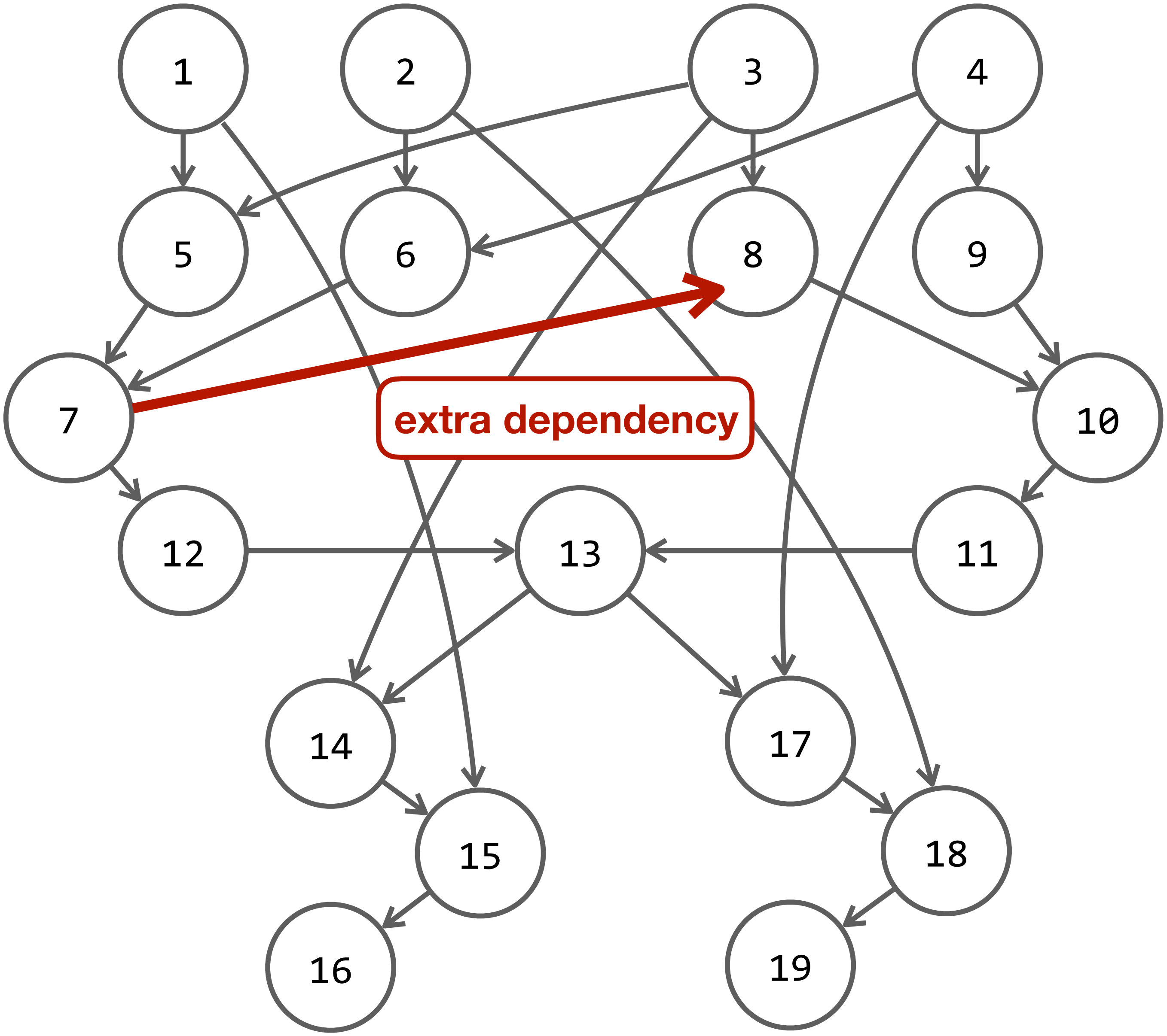


Example — Liveness Analysis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1 LOAD r1 ← x	x																
2 LOAD r2 ← y	|	x															
3 LOAD r3 ← nx	|	|	x														
4 LOAD r4 ← ny	|	|	|	x													
5 FMUL r5 ← r1 * r3	o	|	o	|	x												
6 FMUL r6 ← r2 * r4	|	o	|	o	|	x											
7 FADD r7 ← r5 + r6	|	|	|	|	x	x	x										
8 FMUL r8 ← r3 * r3	|	|	o	|			|	x									
9 FMUL r9 ← r4 * r4	|	|	|	o			|	|	x								
10 FADD r10 ← r8 + r9	|	|	|	|			|	x	x	x							
11 FDIV r11 ← 1.0 / r10	|	|	|	|			|			x	x						
12 FMUL r12 ← 2.0 * r7	|	|	|	|			x				|	x					
13 FMUL r13 ← r12 * r11	|	|	|	|							x	x	x				
14 FMUL r14 ← r3 * r13	|	|	x	|									o	x			
15 FSUB r15 ← r1 - r14	x	|		|									|	x	x		
16 STORE xo ← r15		|		|									|		x		
17 FMUL r16 ← r4 * r13		|		x									x			x	
18 FSUB r17 ← r2 - r16		x														x	x
19 STORE yo ← r17																	x

Example w/ Registers Renamed

```
 1 LOAD   r1 ← x
 2 LOAD   r2 ← y
 3 LOAD   r3 ← nx
 4 LOAD   r4 ← ny
 5 FMUL   r5 ← r1 * r3
 6 FMUL   r6 ← r2 * r4
 7 FADD   r5 ← r5 + r6
 8 FMUL   r6 ← r3 * r3
 9 FMUL   r7 ← r4 * r4
10 FADD   r6 ← r6 + r7
11 FDIV   r6 ← 1.0 / r6
12 FMUL   r5 ← 2.0 * r5
13 FMUL   r5 ← r5 * r6
14 FMUL   r3 ← r3 * r5
15 FSUB   r1 ← r1 - r3
16 STORE  xo ← r1
17 FMUL   r4 ← r4 * r5
18 FSUB   r2 ← r2 - r4
19 STORE  yo ← r2
```



Dependency — Legal Schedules

- Any reordering of instructions that is consistent with the dependency DAG is legal
- The scheduling problem is to pick the best legal schedule
- Post-register allocation scheduling is the same algorithm but has additional dependencies that have to be obeyed.
  - SSA form code *only* has RAW dependencies
  - Register allocation doesn't break RAW dependencies
- Simplest approach to memory ops — treat memory as a big register (no re-ordering of stores with loads or other stores)

Outline

Instruction Scheduling

Dependency Analysis

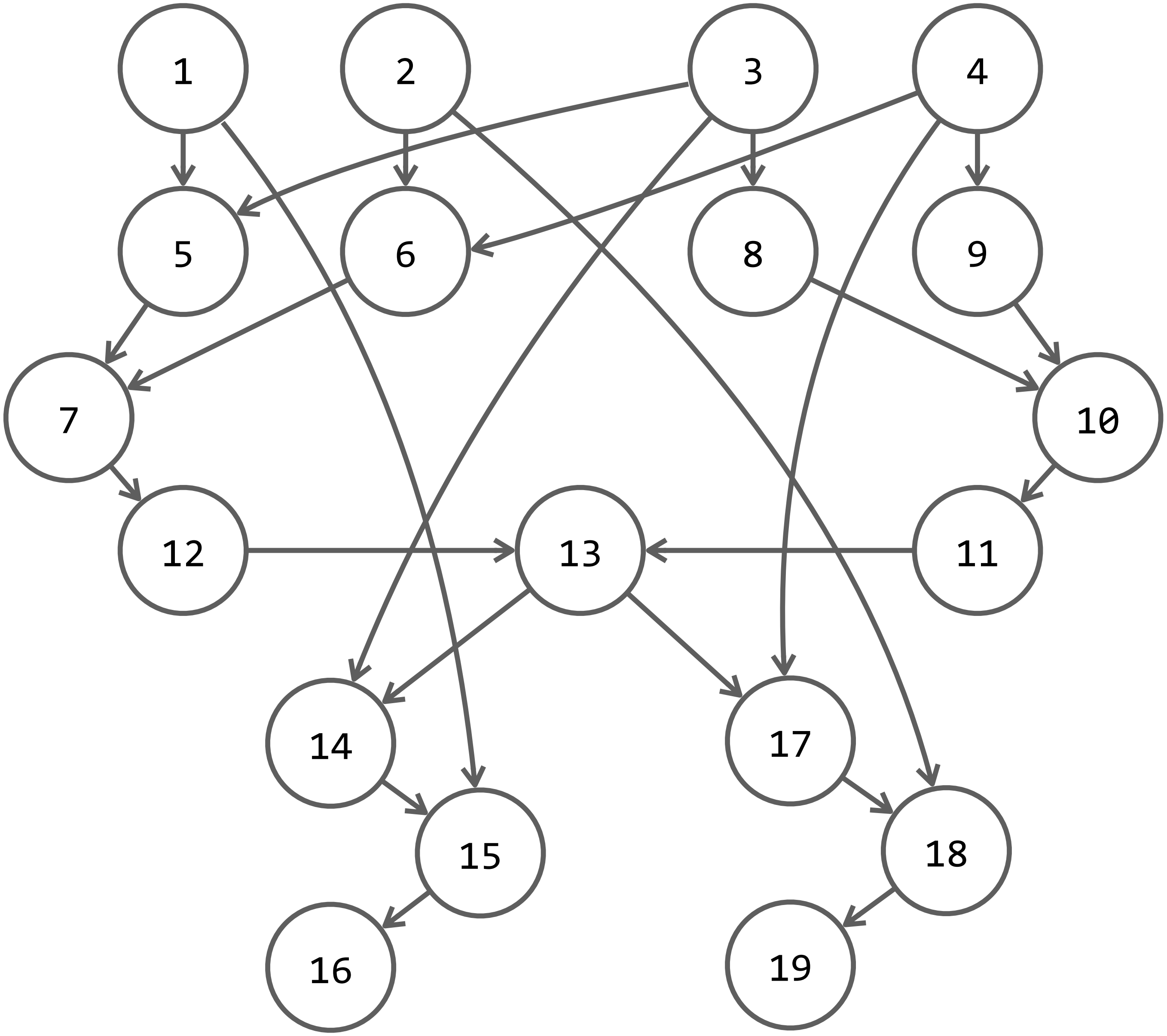
List Scheduling Algorithm

Beyond Basic Blocks

Idea — Traverse the DAG

A ***topological ordering*** is an enumeration of a DAG s.t. the tail of any edge is numbered prior to the head of any edge

Any topological ordering leads to a legal schedule



Greedy Algorithm — List Sched.

- Construct the schedule, one instruction at a time
- Keep a list of operations which are legal (**ready**) to list next
- At each step, pick the best next choice from the ready set
    - What is best? (critical path to minimize latency…)
- After scheduling another instruction, update the ready set
    - remove the scheduled instruction
    - for each successor, check if all predecessors have now been scheduled
- (can also run in reverse — i.e. backwards)

How to Pick — Parallelism

Basic Idea

When trying to pick the next instruction to schedule, we can “simulate” the processor.

The processor will undo this, but if our static schedule is good enough, the dynamic schedule will turn out better

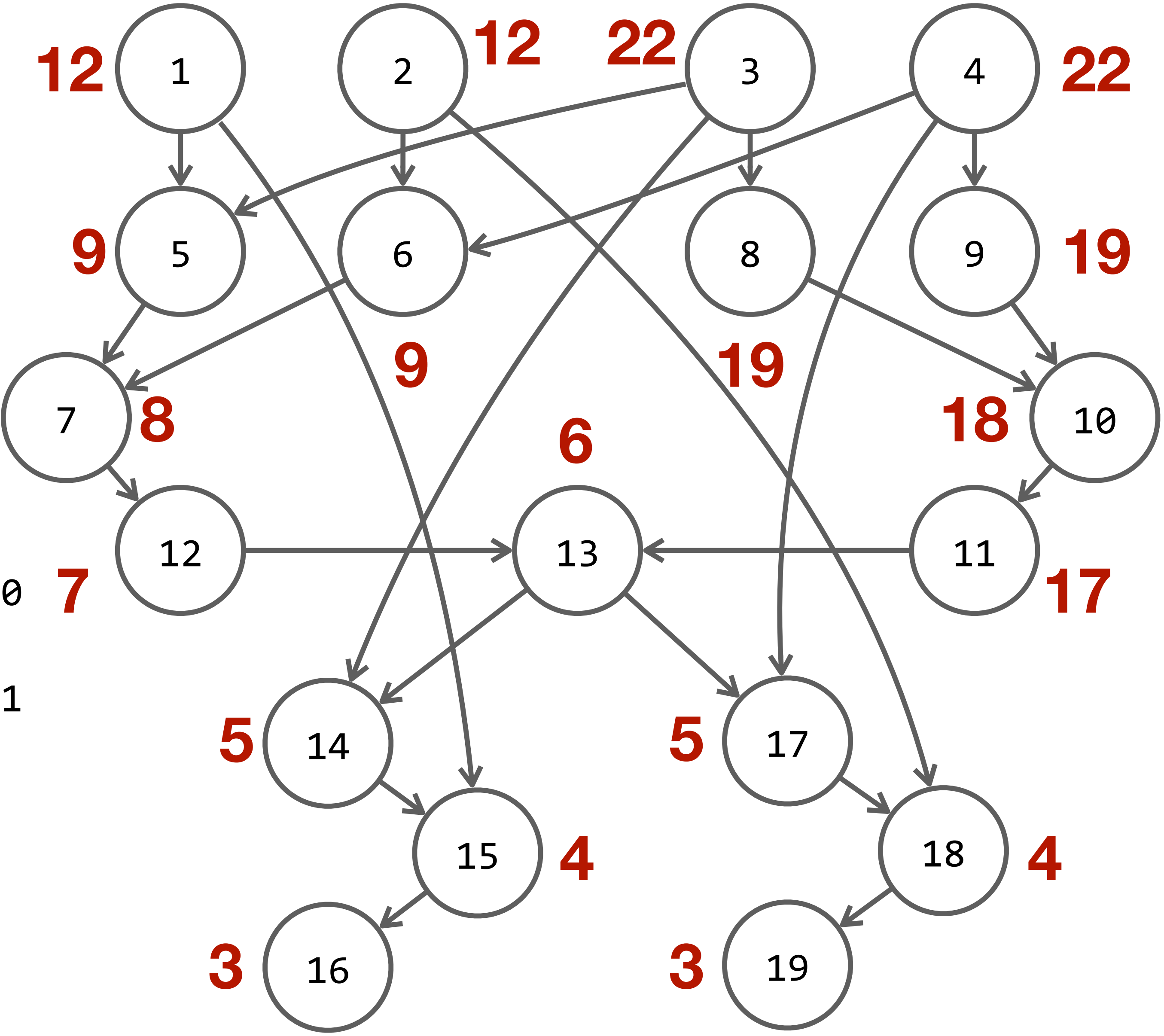
Port 0	**Port 1**	**Port 2**
F+ − ×, F÷	F+ − ×	LD/ST

How to Pick — Critical Path

- Identifying the critical path in the compute DAG requires finding the ***longest path*** in the DAG
    - Longest according to node weight — operation latency
- How hard is it to compute the longest path to the exit from each node?
- In a DAG (acyclic!) this is equivalent to the shortest path problem — an algorithm using topological sorting takes O(n) time
- If using critical path data, precompute the "critical path latency" for every node before starting list scheduling

Example — Critical Path Latency

```
12  1 LOAD   r1 ← x
12  2 LOAD   r2 ← y
22  3 LOAD   r3 ← nx
22  4 LOAD   r4 ← ny
 9  5 FMUL   r5 ← r1 * r3
 9  6 FMUL   r6 ← r2 * r4
 8  7 FADD   r7 ← r5 + r6
19  8 FMUL   r8 ← r3 * r3
19  9 FMUL   r9 ← r4 * r4
18 10 FADD  r10 ← r8 + r9
17 11 FDIV  r11 ← 1.0 / r10
 7 12 FMUL  r12 ← 2.0 * r7
 6 13 FMUL  r13 ← r12 * r11
 5 14 FMUL  r14 ← r3 * r13
 4 15 FSUB  r15 ← r1 - r14
 3 16 STORE  xo ← r15
 5 17 FMUL  r16 ← r4 * r13
 4 18 FSUB  r17 ← r2 - r16
 3 19 STORE  yo ← r17
```



List Scheduling Algorithm

```
exec_units = …;
cycle      = 0;
schedule   = empty list;
ready      = {nodes w/o predecessors};

while ready ≠ ∅
    ops = { op ∈ ready | op can exec at exec_unit[cycle] }
    if ops = ∅
        cycle++;  continue;
    else
        best_op = op ∈ ops with maximum critical path latency
        add op to exec_units[cycle]
        schedule.add(best_op)
        ready = ready - {best_op}
        ready = ready U { op ∈ succ(best_op) |
                                    all pred of op are scheduled  }
```

Example

```
12  1 LOAD   r1 ← x
12  2 LOAD   r2 ← y
22  3 LOAD   r3 ← nx
22  4 LOAD   r4 ← ny
 9  5 FMUL   r5 ← r1 * r3
 9  6 FMUL   r6 ← r2 * r4
 8  7 FADD   r7 ← r5 + r6
19  8 FMUL   r8 ← r3 * r3
19  9 FMUL   r9 ← r4 * r4
18 10 FADD  r10 ← r8 + r9
17 11 FDIV  r11 ← 1.0 / r10
 7 12 FMUL  r12 ← 2.0 * r7
 6 13 FMUL  r13 ← r12 * r11
 5 14 FMUL  r14 ← r3 * r13
 4 15 FSUB  r15 ← r1 - r14
 3 16 STORE  xo ← r15
 5 17 FMUL  r16 ← r4 * r13
 4 18 FSUB  r17 ← r2 - r16
 3 19 STORE  yo ← r17
```

Schedule

Port 0	Port 1	Port 2
F+ − × , F÷	F+ − ×	LD/ST

… 4 more rows …

Example

```
12  1 LOAD    r1 ← x
12  2 LOAD    r2 ← y
22  3 LOAD    r3 ← nx
22  4 LOAD    r4 ← ny
 9  5 FMUL    r5 ← r1 * r3
 9  6 FMUL    r6 ← r2 * r4
 8  7 FADD    r7 ← r5 + r6
19  8 FMUL    r8 ← r3 * r3
19  9 FMUL    r9 ← r4 * r4
18 10 FADD   r10 ← r8 + r9
17 11 FDIV   r11 ← 1.0 / r10
 7 12 FMUL   r12 ← 2.0 * r7
 6 13 FMUL   r13 ← r12 * r11
 5 14 FMUL   r14 ← r3 * r13
 4 15 FSUB   r15 ← r1 - r14
 3 16 STORE   xo ← r15
 5 17 FMUL   r16 ← r4 * r13
 4 18 FSUB   r17 ← r2 - r16
 3 19 STORE   yo ← r17
```

Schedule

3

Port 0	Port 1	Port 2
F+ − × , F÷	F+ − ×	LD/ST
		3
		-
		-

… 4 more rows …

Example

```
12   1 LOAD    r1 ← x
12   2 LOAD    r2 ← y
22   3 LOAD    r3 ← nx
22   4 LOAD    r4 ← ny
 9   5 FMUL    r5 ← r1 * r3
 9   6 FMUL    r6 ← r2 * r4
 8   7 FADD    r7 ← r5 + r6
19   8 FMUL    r8 ← r3 * r3
19   9 FMUL    r9 ← r4 * r4
18  10 FADD   r10 ← r8 + r9
17  11 FDIV   r11 ← 1.0 / r10
 7  12 FMUL   r12 ← 2.0 * r7
 6  13 FMUL   r13 ← r12 * r11
 5  14 FMUL   r14 ← r3 * r13
 4  15 FSUB   r15 ← r1 - r14
 3  16 STORE   xo ← r15
 5  17 FMUL   r16 ← r4 * r13
 4  18 FSUB   r17 ← r2 - r16
 3  19 STORE   yo ← r17
```

Schedule

3

4

Port 0	Port 1	Port 2
F+ − ×, F÷	F+ − ×	LD/ST
		3
		-
		-
		4
		-
		-

… 4 more rows …

Example

```
12  1 LOAD    r1 ← x
12  2 LOAD    r2 ← y
22  3 LOAD    r3 ← nx
22  4 LOAD    r4 ← ny
 9  5 FMUL    r5 ← r1 * r3
 9  6 FMUL    r6 ← r2 * r4
 8  7 FADD    r7 ← r5 + r6
19  8 FMUL    r8 ← r3 * r3
19  9 FMUL    r9 ← r4 * r4
18 10 FADD   r10 ← r8 + r9
17 11 FDIV   r11 ← 1.0 / r10
 7 12 FMUL   r12 ← 2.0 * r7
 6 13 FMUL   r13 ← r12 * r11
 5 14 FMUL   r14 ← r3 * r13
 4 15 FSUB   r15 ← r1 - r14
 3 16 STORE   xo ← r15
 5 17 FMUL   r16 ← r4 * r13
 4 18 FSUB   r17 ← r2 - r16
 3 19 STORE   yo ← r17
```

Schedule

3
4
8

Port 0 F+ − × , F÷	Port 1 F+ − ×	Port 2 LD/ST
		3
		-
		-
8		4
		-
		-

… 4 more rows …

Port 0	Port 1	Port 2

Example

```
12  1 LOAD   r1 ← x
12  2 LOAD   r2 ← y
22  3 LOAD   r3 ← nx
22  4 LOAD   r4 ← ny
 9  5 FMUL   r5 ← r1 * r3
 9  6 FMUL   r6 ← r2 * r4
 8  7 FADD   r7 ← r5 + r6
19  8 FMUL   r8 ← r3 * r3
19  9 FMUL   r9 ← r4 * r4
18 10 FADD  r10 ← r8 + r9
17 11 FDIV  r11 ← 1.0 / r10
 7 12 FMUL  r12 ← 2.0 * r7
 6 13 FMUL  r13 ← r12 * r11
 5 14 FMUL  r14 ← r3 * r13
 4 15 FSUB  r15 ← r1 - r14
 3 16 STORE  xo ← r15
 5 17 FMUL  r16 ← r4 * r13
 4 18 FSUB  r17 ← r2 - r16
 3 19 STORE  yo ← r17
```

Schedule

3
4
8
1

Port 0	Port 1	Port 2
F+ − ×, F÷	F+ − ×	LD/ST
		3
		-
		-
8		4
		-
		-
		1
		-
		-

… 4 more rows …

Example

```
12   1 LOAD    r1 ← x
12   2 LOAD    r2 ← y
22   3 LOAD    r3 ← nx
22   4 LOAD    r4 ← ny
 9   5 FMUL    r5 ← r1 * r3
 9   6 FMUL    r6 ← r2 * r4
 8   7 FADD    r7 ← r5 + r6
19   8 FMUL    r8 ← r3 * r3
19   9 FMUL    r9 ← r4 * r4
18  10 FADD   r10 ← r8 + r9
17  11 FDIV   r11 ← 1.0 / r10
 7  12 FMUL   r12 ← 2.0 * r7
 6  13 FMUL   r13 ← r12 * r11
 5  14 FMUL   r14 ← r3 * r13
 4  15 FSUB   r15 ← r1 - r14
 3  16 STORE   xo ← r15
 5  17 FMUL   r16 ← r4 * r13
 4  18 FSUB   r17 ← r2 - r16
 3  19 STORE   yo ← r17
```

Schedule

3
4
8
1
9

Port 0	Port 1	Port 2
F+ − ×, F÷	F+ − ×	LD/ST
		3
		-
		-
8		4
		-
		-
9		1
		-
		-

… 4 more rows …

Example

```
12   1 LOAD    r1 ← x
12   2 LOAD    r2 ← y
22   3 LOAD    r3 ← nx
22   4 LOAD    r4 ← ny
 9   5 FMUL    r5 ← r1 * r3
 9   6 FMUL    r6 ← r2 * r4
 8   7 FADD    r7 ← r5 + r6
19   8 FMUL    r8 ← r3 * r3
19   9 FMUL    r9 ← r4 * r4
18  10 FADD   r10 ← r8 + r9
17  11 FDIV   r11 ← 1.0 / r10
 7  12 FMUL   r12 ← 2.0 * r7
 6  13 FMUL   r13 ← r12 * r11
 5  14 FMUL   r14 ← r3 * r13
 4  15 FSUB   r15 ← r1 - r14
 3  16 STORE   xo ← r15
 5  17 FMUL   r16 ← r4 * r13
 4  18 FSUB   r17 ← r2 - r16
 3  19 STORE   yo ← r17
```

Schedule

3
4
8
1
9
10

Port 0	Port 1	Port 2
F+ − ×, F÷	F+ − ×	LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
		-

… 4 more rows …

Example

```
12   1 LOAD    r1 ← x
12   2 LOAD    r2 ← y
22   3 LOAD    r3 ← nx
22   4 LOAD    r4 ← ny
 9   5 FMUL    r5 ← r1 * r3
 9   6 FMUL    r6 ← r2 * r4
 8   7 FADD    r7 ← r5 + r6
19   8 FMUL    r8 ← r3 * r3
19   9 FMUL    r9 ← r4 * r4
18  10 FADD   r10 ← r8 + r9
17  11 FDIV   r11 ← 1.0 / r10
 7  12 FMUL   r12 ← 2.0 * r7
 6  13 FMUL   r13 ← r12 * r11
 5  14 FMUL   r14 ← r3 * r13
 4  15 FSUB   r15 ← r1 - r14
 3  16 STORE   xo ← r15
 5  17 FMUL   r16 ← r4 * r13
 4  18 FSUB   r17 ← r2 - r16
 3  19 STORE   yo ← r17
```

Schedule

3
4
8
1
9
10
11

Port 0 F+ − ×, F÷	Port 1 F+ − ×	Port 2 LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
11		-
-		
-		
-		
-		
-		
-		

… 4 more rows …

Example

```
12   1 LOAD    r1  ← x
12   2 LOAD    r2  ← y
22   3 LOAD    r3  ← nx
22   4 LOAD    r4  ← ny
 9   5 FMUL    r5  ← r1 * r3
 9   6 FMUL    r6  ← r2 * r4
 8   7 FADD    r7  ← r5 + r6
19   8 FMUL    r8  ← r3 * r3
19   9 FMUL    r9  ← r4 * r4
18  10 FADD   r10  ← r8 + r9
17  11 FDIV   r11  ← 1.0 / r10
 7  12 FMUL   r12  ← 2.0 * r7
 6  13 FMUL   r13  ← r12 * r11
 5  14 FMUL   r14  ← r3 * r13
 4  15 FSUB   r15  ← r1 - r14
 3  16 STORE   xo  ← r15
 5  17 FMUL   r16  ← r4 * r13
 4  18 FSUB   r17  ← r2 - r16
 3  19 STORE   yo  ← r17
```

Schedule

3
4
8
1
9
10
11
2

Port 0	**Port 1**	**Port 2**
F+ – ×, F÷	F+ – ×	LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
11		-
-		2
-		-
-		-
-		
-		
-		

… 4 more rows …

Example

```
12   1 LOAD    r1  ← x
12   2 LOAD    r2  ← y
22   3 LOAD    r3  ← nx
22   4 LOAD    r4  ← ny
 9   5 FMUL    r5  ← r1 * r3
 9   6 FMUL    r6  ← r2 * r4
 8   7 FADD    r7  ← r5 + r6
19   8 FMUL    r8  ← r3 * r3
19   9 FMUL    r9  ← r4 * r4
18  10 FADD   r10  ← r8 + r9
17  11 FDIV   r11  ← 1.0 / r10
 7  12 FMUL   r12  ← 2.0 * r7
 6  13 FMUL   r13  ← r12 * r11
 5  14 FMUL   r14  ← r3 * r13
 4  15 FSUB   r15  ← r1 - r14
 3  16 STORE   xo  ← r15
 5  17 FMUL   r16  ← r4 * r13
 4  18 FSUB   r17  ← r2 - r16
 3  19 STORE   yo  ← r17
```

Schedule

3
4
8
1
9
10
11
2
5

Port 0	**Port 1**	**Port 2**
F+ − ×, F÷	F+ − ×	LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
11		-
-	5	2
-		-
-		-
-		
-		
-		

… 4 more rows …

Example

```
12  1 LOAD   r1 ← x
12  2 LOAD   r2 ← y
22  3 LOAD   r3 ← nx
22  4 LOAD   r4 ← ny
 9  5 FMUL   r5 ← r1 * r3
 9  6 FMUL   r6 ← r2 * r4
 8  7 FADD   r7 ← r5 + r6
19  8 FMUL   r8 ← r3 * r3
19  9 FMUL   r9 ← r4 * r4
18 10 FADD  r10 ← r8 + r9
17 11 FDIV  r11 ← 1.0 / r10
 7 12 FMUL  r12 ← 2.0 * r7
 6 13 FMUL  r13 ← r12 * r11
 5 14 FMUL  r14 ← r3 * r13
 4 15 FSUB  r15 ← r1 - r14
 3 16 STORE  xo ← r15
 5 17 FMUL  r16 ← r4 * r13
 4 18 FSUB  r17 ← r2 - r16
 3 19 STORE  yo ← r17
```

Schedule

3
4
8
1
9
10
11
2
5
6

Port 0 F+ − × , F÷	**Port 1** F+ − ×	**Port 2** LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
11		-
-	5	2
-		-
-		-
-	6	
-		
-		

… 4 more rows …

Example

```
12  1 LOAD   r1 ← x
12  2 LOAD   r2 ← y
22  3 LOAD   r3 ← nx
22  4 LOAD   r4 ← ny
 9  5 FMUL   r5 ← r1 * r3
 9  6 FMUL   r6 ← r2 * r4
 8  7 FADD   r7 ← r5 + r6
19  8 FMUL   r8 ← r3 * r3
19  9 FMUL   r9 ← r4 * r4
18 10 FADD  r10 ← r8 + r9
17 11 FDIV  r11 ← 1.0 / r10
 7 12 FMUL  r12 ← 2.0 * r7
 6 13 FMUL  r13 ← r12 * r11
 5 14 FMUL  r14 ← r3 * r13
 4 15 FSUB  r15 ← r1 - r14
 3 16 STORE  xo ← r15
 5 17 FMUL  r16 ← r4 * r13
 4 18 FSUB  r17 ← r2 - r16
 3 19 STORE  yo ← r17
```

Schedule

3
4
8
1
9
10
11
2
5
6
7

Port 0	Port 1	Port 2
F+ − ×, F÷	F+ − ×	LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
11		-
-	5	2
-		-
-		-
-	6	
-	7	
-		

… 4 more rows …

Example

```
12   1 LOAD   r1  ← x
12   2 LOAD   r2  ← y
22   3 LOAD   r3  ← nx
22   4 LOAD   r4  ← ny
 9   5 FMUL   r5  ← r1 * r3
 9   6 FMUL   r6  ← r2 * r4
 8   7 FADD   r7  ← r5 + r6
19   8 FMUL   r8  ← r3 * r3
19   9 FMUL   r9  ← r4 * r4
18  10 FADD  r10  ← r8 + r9
17  11 FDIV  r11  ← 1.0 / r10
 7  12 FMUL  r12  ← 2.0 * r7
 6  13 FMUL  r13  ← r12 * r11
 5  14 FMUL  r14  ← r3 * r13
 4  15 FSUB  r15  ← r1 - r14
 3  16 STORE  xo  ← r15
 5  17 FMUL  r16  ← r4 * r13
 4  18 FSUB  r17  ← r2 - r16
 3  19 STORE  yo  ← r17
```

Schedule

3
4
8
1
9
10
11
2
5
6
7
12

Port 0	Port 1	Port 2
F+ – ×, F÷	F+ – ×	LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
11		-
-	5	2
-		-
-		-
-	6	
-	7	
-	12	

… 4 more rows …

Example

```
12   1 LOAD    r1  ← x
12   2 LOAD    r2  ← y
22   3 LOAD    r3  ← nx
22   4 LOAD    r4  ← ny
 9   5 FMUL    r5  ← r1 * r3
 9   6 FMUL    r6  ← r2 * r4
 8   7 FADD    r7  ← r5 + r6
19   8 FMUL    r8  ← r3 * r3
19   9 FMUL    r9  ← r4 * r4
18  10 FADD   r10  ← r8 + r9
17  11 FDIV   r11  ← 1.0 / r10
 7  12 FMUL   r12  ← 2.0 * r7
 6  13 FMUL   r13  ← r12 * r11
 5  14 FMUL   r14  ← r3 * r13
 4  15 FSUB   r15  ← r1 - r14
 3  16 STORE   xo  ← r15
 5  17 FMUL   r16  ← r4 * r13
 4  18 FSUB   r17  ← r2 - r16
 3  19 STORE   yo  ← r17
```

Schedule

3
4
8
1
9
10
11
2
5
6
7
12
13

Port 0	Port 1	Port 2
F+ − × , F÷	F+ − ×	LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
11		-
-	5	2
-		-
-		-
-	6	
-	7	
-	12	

… 4 more rows …

13		

Example

```
12  1 LOAD   r1  ← x
12  2 LOAD   r2  ← y
22  3 LOAD   r3  ← nx
22  4 LOAD   r4  ← ny
 9  5 FMUL   r5  ← r1 * r3
 9  6 FMUL   r6  ← r2 * r4
 8  7 FADD   r7  ← r5 + r6
19  8 FMUL   r8  ← r3 * r3
19  9 FMUL   r9  ← r4 * r4
18 10 FADD  r10  ← r8 + r9
17 11 FDIV  r11  ← 1.0 / r10
 7 12 FMUL  r12  ← 2.0 * r7
 6 13 FMUL  r13  ← r12 * r11
 5 14 FMUL  r14  ← r3 * r13
 4 15 FSUB  r15  ← r1 - r14
 3 16 STORE  xo  ← r15
 5 17 FMUL  r16  ← r4 * r13
 4 18 FSUB  r17  ← r2 - r16
 3 19 STORE  yo  ← r17
```

Schedule

3
4
8
1
9
10
11
2
5
6
7
12
13
14

Port 0 F+ − × , F÷	**Port 1** F+ − ×	**Port 2** LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
11		-
-	5	2
-		-
-		-
-	6	
-	7	
-	12	

… 4 more rows …

13		
14		

Example

```
12   1 LOAD   r1  ← x
12   2 LOAD   r2  ← y
22   3 LOAD   r3  ← nx
22   4 LOAD   r4  ← ny
 9   5 FMUL   r5  ← r1 * r3
 9   6 FMUL   r6  ← r2 * r4
 8   7 FADD   r7  ← r5 + r6
19   8 FMUL   r8  ← r3 * r3
19   9 FMUL   r9  ← r4 * r4
18  10 FADD  r10  ← r8 + r9
17  11 FDIV  r11  ← 1.0 / r10
 7  12 FMUL  r12  ← 2.0 * r7
 6  13 FMUL  r13  ← r12 * r11
 5  14 FMUL  r14  ← r3 * r13
 4  15 FSUB  r15  ← r1 - r14
 3  16 STORE  xo  ← r15
 5  17 FMUL  r16  ← r4 * r13
 4  18 FSUB  r17  ← r2 - r16
 3  19 STORE  yo  ← r17
```

Schedule

3
4
8
1
9
10
11
2
5
6
7
12
13
14
17

Port 0	Port 1	Port 2
F+ − ×, F÷	F+ − ×	LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
11		-
-	5	2
-		-
-		-
-	6	
-	7	
-	12	

… 4 more rows …

13		
14	17	

Example

```
12   1 LOAD    r1 ← x
12   2 LOAD    r2 ← y
22   3 LOAD    r3 ← nx
22   4 LOAD    r4 ← ny
 9   5 FMUL    r5 ← r1 * r3
 9   6 FMUL    r6 ← r2 * r4
 8   7 FADD    r7 ← r5 + r6
19   8 FMUL    r8 ← r3 * r3
19   9 FMUL    r9 ← r4 * r4
18  10 FADD   r10 ← r8 + r9
17  11 FDIV   r11 ← 1.0 / r10
 7  12 FMUL   r12 ← 2.0 * r7
 6  13 FMUL   r13 ← r12 * r11
 5  14 FMUL   r14 ← r3 * r13
 4  15 FSUB   r15 ← r1 - r14
 3  16 STORE   xo ← r15
 5  17 FMUL   r16 ← r4 * r13
 4  18 FSUB   r17 ← r2 - r16
 3  19 STORE   yo ← r17
```

Schedule

3
4
8
1
9
10
11
2
5
6
7
12
13
14
17
15

Port 0 F+ − × , F÷	**Port 1** F+ − ×	**Port 2** LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
11		-
-	5	2
-		-
-		-
-	6	
-	7	
-	12	

… 4 more rows …

13		
14	17	
15		

Example

```
12   1 LOAD    r1  ← x
12   2 LOAD    r2  ← y
22   3 LOAD    r3  ← nx
22   4 LOAD    r4  ← ny
 9   5 FMUL    r5  ← r1 * r3
 9   6 FMUL    r6  ← r2 * r4
 8   7 FADD    r7  ← r5 + r6
19   8 FMUL    r8  ← r3 * r3
19   9 FMUL    r9  ← r4 * r4
18  10 FADD   r10  ← r8 + r9
17  11 FDIV   r11  ← 1.0 / r10
 7  12 FMUL   r12  ← 2.0 * r7
 6  13 FMUL   r13  ← r12 * r11
 5  14 FMUL   r14  ← r3 * r13
 4  15 FSUB   r15  ← r1 - r14
 3  16 STORE   xo  ← r15
 5  17 FMUL   r16  ← r4 * r13
 4  18 FSUB   r17  ← r2 - r16
 3  19 STORE   yo  ← r17
```

Schedule

3
4
8
1
9
10
11
2
5
6
7
12
13
14
17
15
18

Port 0	**Port 1**	**Port 2**
F+ − × , F÷	F+ − ×	LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
11		-
-	5	2
-		-
-		-
-	6	
-	7	
-	12	

… 4 more rows …

13		
14	17	
15	18	

Example

```
12   1 LOAD    r1  ← x
12   2 LOAD    r2  ← y
22   3 LOAD    r3  ← nx
22   4 LOAD    r4  ← ny
 9   5 FMUL    r5  ← r1 * r3
 9   6 FMUL    r6  ← r2 * r4
 8   7 FADD    r7  ← r5 + r6
19   8 FMUL    r8  ← r3 * r3
19   9 FMUL    r9  ← r4 * r4
18  10 FADD   r10  ← r8 + r9
17  11 FDIV   r11  ← 1.0 / r10
 7  12 FMUL   r12  ← 2.0 * r7
 6  13 FMUL   r13  ← r12 * r11
 5  14 FMUL   r14  ← r3 * r13
 4  15 FSUB   r15  ← r1 - r14
 3  16 STORE   xo  ← r15
 5  17 FMUL   r16  ← r4 * r13
 4  18 FSUB   r17  ← r2 - r16
 3  19 STORE   yo  ← r17
```

Schedule

3
4
8
1
9
10
11
2
5
6
7
12
13
14
17
15
18
16

Port 0	Port 1	Port 2
F+ − ×, F÷	F+ − ×	LD/ST
		3
		-
		-
8		4
		-
		-
9		1
10		-
11		-
-	5	2
-		-
-		-
-	6	
-	7	
-	12	

… 4 more rows …

13		
14	17	
15	18	
		16
		-
		-

Example — Effect of Sched.

Original

```
 1 LOAD   r1 ← x
 2 LOAD   r2 ← y
 3 LOAD   r3 ← nx
 4 LOAD   r4 ← ny
 5 FMUL   r5 ← r1 * r3
 6 FMUL   r6 ← r2 * r4
 7 FADD   r7 ← r5 + r6
 8 FMUL   r8 ← r3 * r3
 9 FMUL   r9 ← r4 * r4
10 FADD  r10 ← r8 + r9
11 FDIV  r11 ← 1.0 / r10
12 FMUL  r12 ← 2.0 * r7
13 FMUL  r13 ← r12 * r11
14 FMUL  r14 ← r3 * r13
15 FSUB  r15 ← r1 - r14
16 STORE  xo ← r15
17 FMUL  r16 ← r4 * r13
18 FSUB  r17 ← r2 - r16
19 STORE  yo ← r17
```

Scheduled

```
 3 LOAD   r3 ← nx
 4 LOAD   r4 ← ny
 8 FMUL   r8 ← r3 * r3
 1 LOAD   r1 ← x
 9 FMUL   r9 ← r4 * r4
10 FADD  r10 ← r8 + r9
11 FDIV  r11 ← 1.0 / r10
 2 LOAD   r2 ← y
 5 FMUL   r5 ← r1 * r3
 6 FMUL   r6 ← r2 * r4
 7 FADD   r7 ← r5 + r6
12 FMUL  r12 ← 2.0 * r7
13 FMUL  r13 ← r12 * r11
14 FMUL  r14 ← r3 * r13
17 FMUL  r16 ← r4 * r13
15 FSUB  r15 ← r1 - r14
18 FSUB  r17 ← r2 - r16
16 STORE  xo ← r15
19 STORE  yo ← r17
```

Outline

Instruction Scheduling

Dependency Analysis

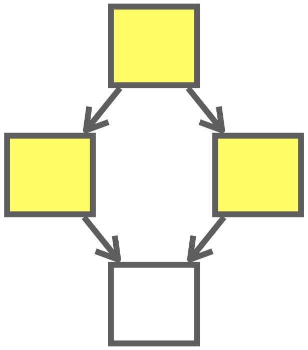
List Scheduling Algorithm

Beyond Basic Blocks

Loop Unrolling

- Where is most of the execution time of a program spent?
- The middle of the compiler can transform the code to create larger basic blocks
- Loop unrolling creates larger blocks of inner-loop code
    - More instructions to schedule, interleave, etc.
    - Creates more opportunities for overlapping compute and memory instructions
    - Reduces opportunities for branch mispredictions
- Other loop transforms (e.g. software pipelining, not discussed) can have similar benefits, but loop unrolling is very simple and easy to do

Extended Basic Blocks



- Run dominator analysis and extract a portion of the CFG that exactly matches the dominator tree (i.e. no other edges in)
- Perform scheduling on all the instructions in all of these basic blocks jointly
- New Problems!
  - What if we move an instruction out of a branch into a common ancestor?
- Correctness — need to ensure no side effects on such instructions
- Speculation — We might end up doing extra work. Is it worth it?

Trace Scheduling

- In JIT compilers, or *profile-guided compilation*, we have access to ***dynamic*** information about the program execution. (We don't have to guess which code is executed more frequently, or which branches are taken more often
- Often there is a ***hot path*** through a sequence of branches
- We can "trace" this path and extract it as a single basic block
  - Before causing side-effects, check whether all the branch conditions were met
  - If some branch condition fails, fall-back to a slower, but complete version of the code

Scheduling at Large

- Many other systems must tackle some kind of similar scheduling problem
- Operating Systems — Process Scheduling
  - ✦ Which CPU core should a process run on?
- GPUs
  - ✦ Which “SM” should a thread-block run on?
- Distributed Systems
  - ✦ Which node should a task run on?
- One of the great fundamental systems problems

Next Time…

- Almost Done!
- Register Allocation on Wednesday
  - AND Closing lecture from Gilbert
- FRIDAY — Guest lecture from Hal on Garbage Collection
  - Please come, GC material may be on the final!