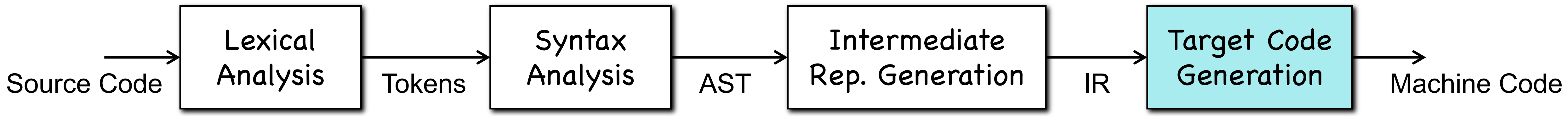
Made by Qingkai Shi
qingkaishi@nju.edu.cn

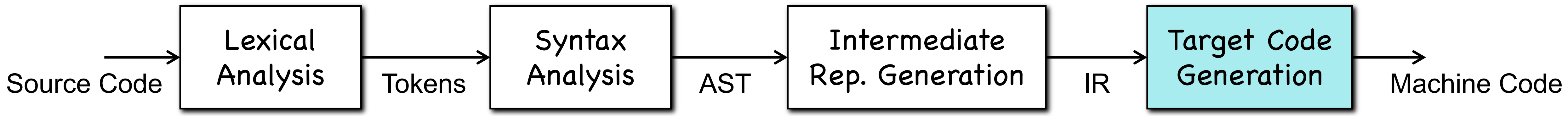
Chapter 10
Instruction Scheduling

Instruction Scheduling



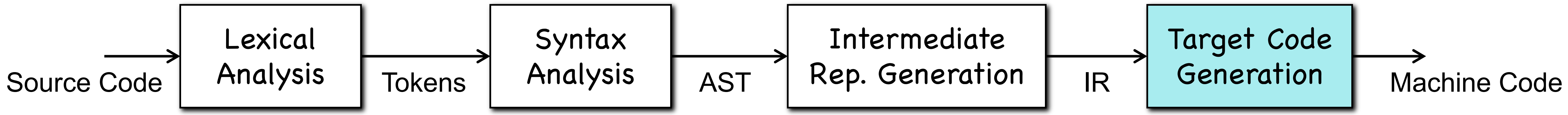
Can we generate code to leverage modern CPUs for better instruction-level parallelization?

Instruction Scheduling



- Every modern high-performance processor can execute several operations in a single clock cycle.
- **Billion-Dollar Question**: how fast can a program be run on a processor with instruction-level parallelism?

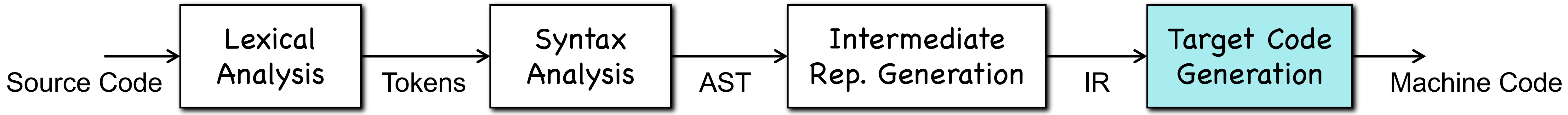
Instruction Scheduling



- Every modern high-performance processor can execute several operations in a single clock cycle.
- **Billion-Dollar Question**: how fast can a program be run on a processor with instruction-level parallelism?

The potential parallelism in the program	The available parallelism on the processor
Our ability to extract parallelism from the code	Our ability to find the best scheduling

Instruction Scheduling

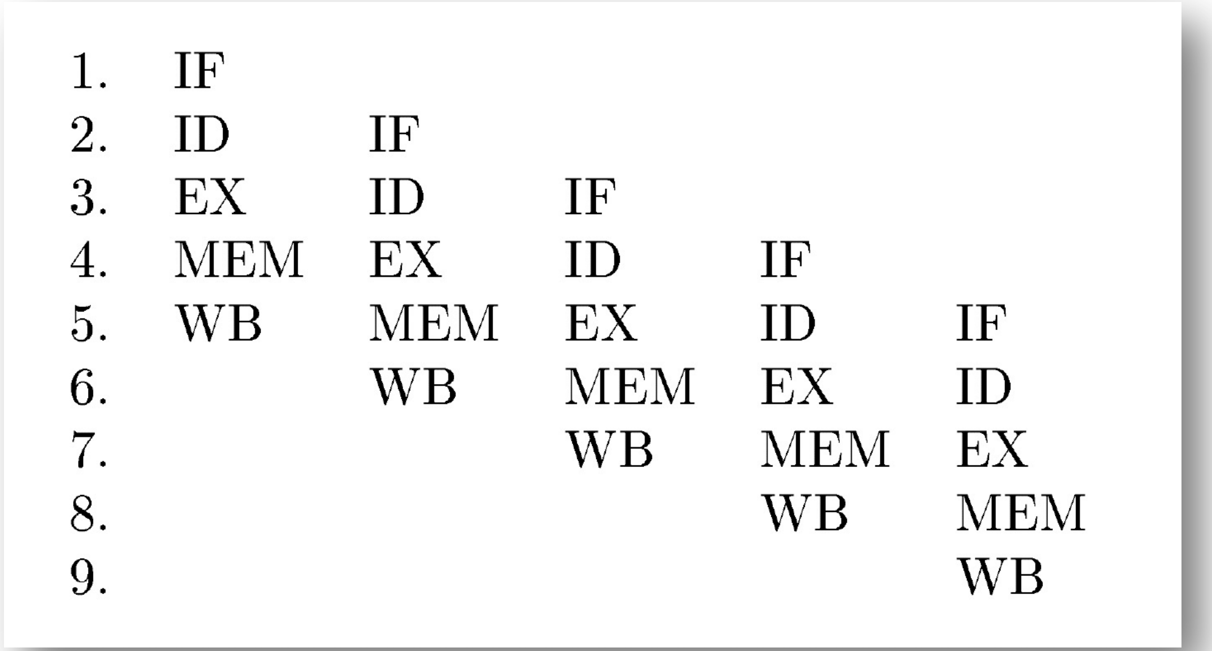


- Every modern high-performance processor can execute several operations in a single clock cycle.
- **Billion-Dollar Question**: how fast can a program be run on a processor with instruction-level parallelism?

The potential parallelism in the program	The available parallelism on the processor
Our ability to extract parallelism from the code	Our ability to find the best scheduling

Instruction Scheduling

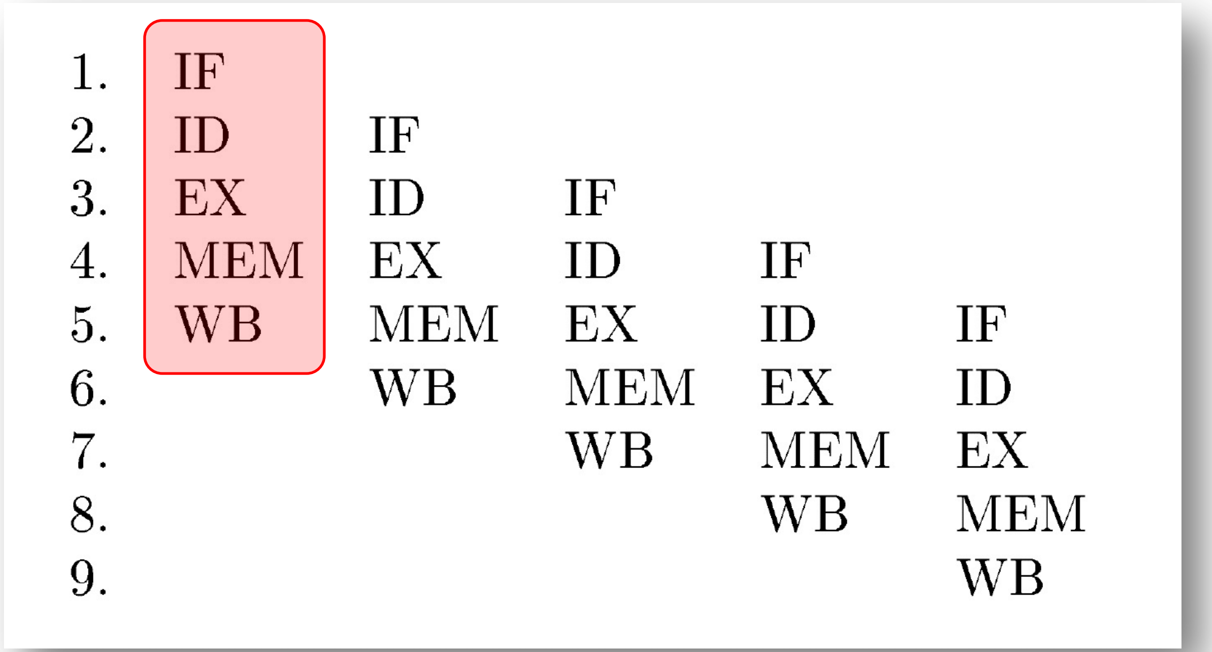
- Instruction pipelines



1.	IF				
2.	ID	IF			
3.	EX	ID	IF		
4.	MEM	EX	ID	IF	
5.	WB	MEM	EX	ID	IF
6.		WB	MEM	EX	ID
7.			WB	MEM	EX
8.				WB	MEM
9.					WB

Instruction Scheduling

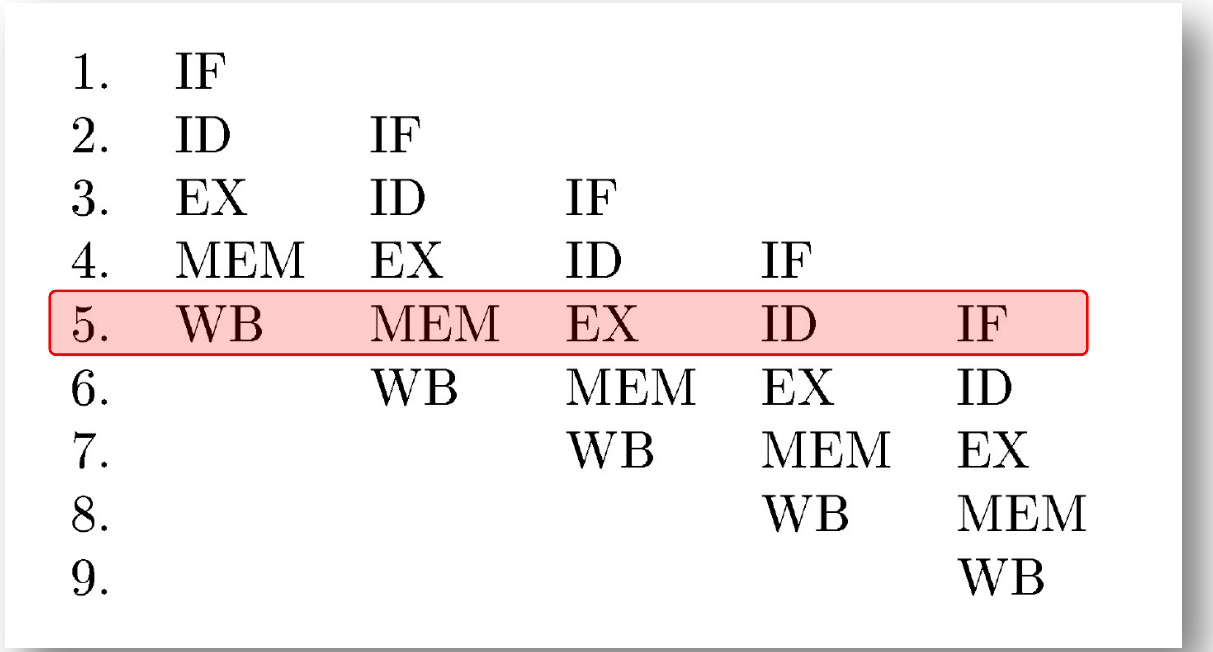
- Instruction pipelines



1.	IF				
2.	ID	IF			
3.	EX	ID	IF		
4.	MEM	EX	ID	IF	
5.	WB	MEM	EX	ID	IF
6.		WB	MEM	EX	ID
7.			WB	MEM	EX
8.				WB	MEM
9.					WB

Instruction Scheduling

- Instruction pipelines

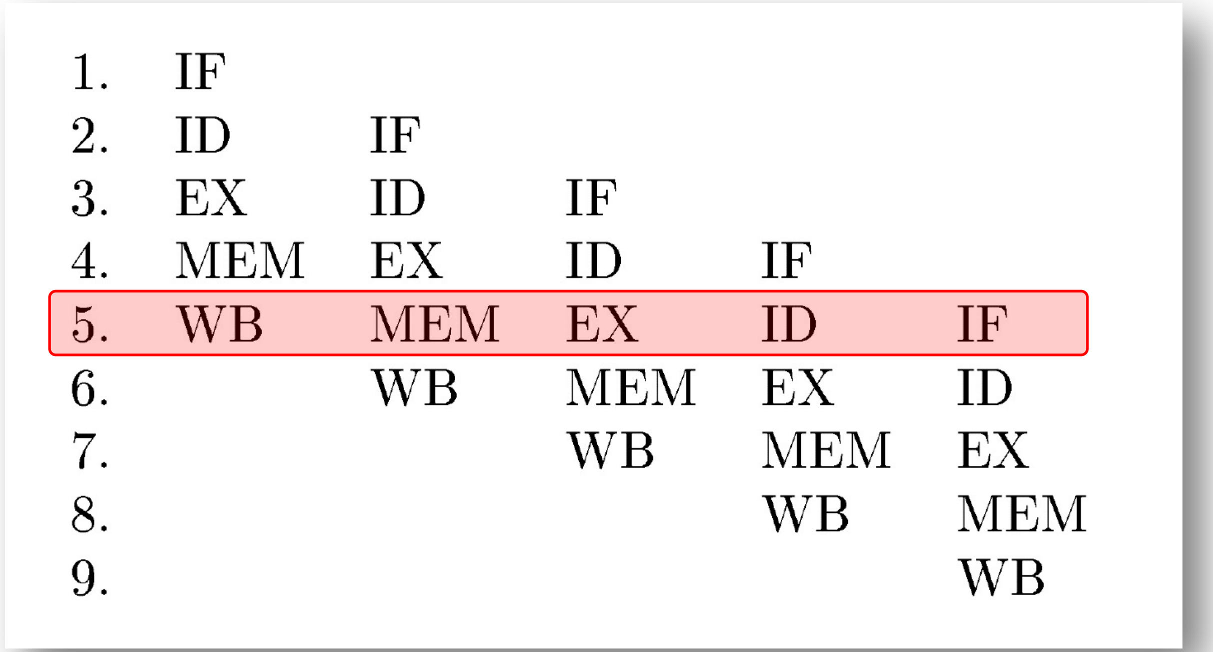


1.	IF				
2.	ID	IF			
3.	EX	ID	IF		
4.	MEM	EX	ID	IF	
5.	WB	MEM	EX	ID	IF
6.		WB	MEM	EX	ID
7.			WB	MEM	EX
8.				WB	MEM
9.					WB

Instruction Scheduling

- Instruction pipelines

Note: an instruction may not contain all five phases



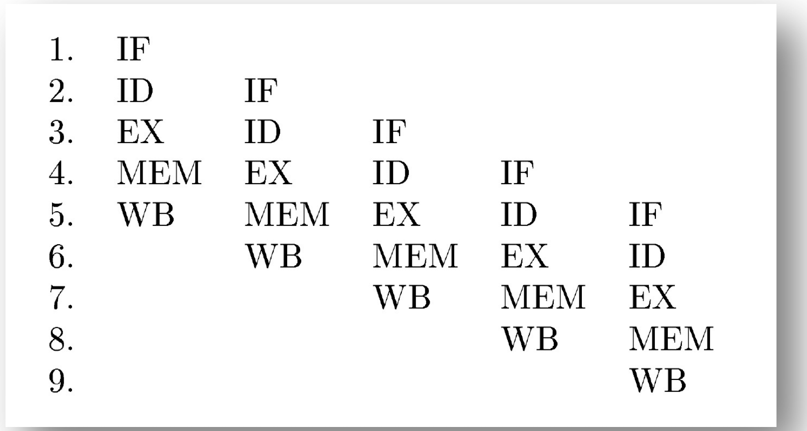
1.	IF				
2.	ID	IF			
3.	EX	ID	IF		
4.	MEM	EX	ID	IF	
5.	WB	MEM	EX	ID	IF
6.		WB	MEM	EX	ID
7.			WB	MEM	EX
8.				WB	MEM
9.					WB

Instruction Scheduling

- Machine types
  - **VLIW (Very long instruction machine) machine**
    - Instruction words are longer, encode the operations to be issued in a single clock
    - Compilers decide which operations are scheduled in parallel by encoding such info into the instructions

Instruction Scheduling

- Machine types
  - VLIW (Very long instruction machine) machine
  - **Superscalar machine**
    - Regular instruction set with an ordinary sequential-execution semantics
    - Automatically detect dependencies and issue them in parallel



1.	IF				
2.	ID	IF			
3.	EX	ID	IF		
4.	MEM	EX	ID	IF	
5.	WB	MEM	EX	ID	IF
6.		WB	MEM	EX	ID
7.			WB	MEM	EX
8.				WB	MEM
9.					WB

Instruction Scheduling

- Machine types
  - VLIW (Very long instruction machine) machine
  - Superscalar machine
  - **“Out-of-order” machine**
    - Automatically issue as many instructions as possible
    - Stop them until all operands are ready

Instruction Scheduling

- Machine types
  - VLIW (Very long instruction machine) machine
  - **Superscalar machine**
  - **“Out-of-order” machine**

- Why compilers matter?

南京大学编译技术
及软件安全研究组

Instruction Scheduling

- Machine types
  - VLIW (Very long instruction machine) machine
  - Superscalar machine
  - “Out-of-order” machine

- Why compilers matter?
  - Hardware only can execute instructions that have been fetched

Instruction Scheduling

- Machine types
  - VLIW (Very long instruction machine) machine
  - Superscalar machine
  - “Out-of-order” machine

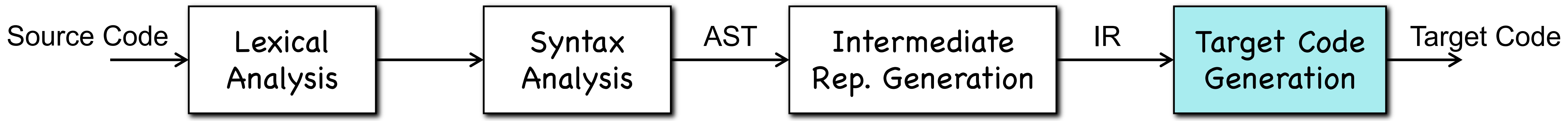
- Why compilers matter?
  - Hardware only can execute instructions that have been fetched
  - Hardware has limited space to buffer operations that must be stalled

Instruction Scheduling

- Machine types
  - VLIW (Very long instruction machine) machine
  - Superscalar machine
  - “Out-of-order” machine

- Why compilers matter?
  - Hardware only can execute instructions that have been fetched
  - Hardware has limited space to buffer operations that must be stalled
  - Compilers can place independent operations close together to allow better hardware utilization

Instruction Scheduling



- Every modern high-performance processor can execute several operations in a single clock cycle.
- **Billion-Dollar Question**: how fast can a program be run on a processor with instruction-level parallelism?

The potential parallelism in the program

The available parallelism on the processor

Our ability to extract parallelism from the code

Our ability to find the best scheduling

Instruction Scheduling

- **Example: Superscalar machine (1 functional unit)**
  - LD/ST 3 cycles; MUL 2 cycles; ADD 1 cycle

```
LD   R1,  a
ADD  R1,  R1,  R1
LD   R2,  b
MUL  R1,  R1,  R2
LD   R2,  c
MUL  R1,  R1,  R2
ST   a,   R1
```

a = 2 * a * b * c

Instruction Scheduling

- **Example: Superscalar machine (1 functional unit)**
  - LD/ST 3 cycles; MUL 2 cycles; ADD 1 cycle

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

Instruction Scheduling

- **Example: Superscalar machine (1 functional unit)**
  - LD/ST 3 cycles; MUL 2 cycles; ADD 1 cycle

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

Instruction Scheduling

- **Example: Superscalar machine (1 functional unit)**
  - LD/ST 3 cycles; MUL 2 cycles; ADD 1 cycle

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

Instruction Scheduling

- **Example: Superscalar machine (1 functional unit)**
  - LD/ST 3 cycles; MUL 2 cycles; ADD 1 cycle

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

Instruction Scheduling

- **Example: Superscalar machine (1 functional unit)**
  - LD/ST 3 cycles; MUL 2 cycles; ADD 1 cycle

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

$\xrightarrow{\textit{schedule}}$

1	LD R1, a	……
2		……
3		R1 ready
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		

Instruction Schec

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

schedule

1	LD R1, a	……
2		……
3		**R1 ready**
4	ADD R1, R1, R1	**R1 ready**
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		

南京大学编译技术
及软件安全研究组

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

schedule →

1	LD R1, a	……
2		……
3		**R1 ready**
4	ADD R1, R1, R1	**R1 ready**
5	LD R2, b	**R1 ready**
6		**R1 ready**
7		**R1/R2 ready**
8		
9		
10		
11		
12		
13		
14		
15		
16		

Instruction Schec

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

schedule

a = 2 * a * b * c

1	LD R1, a	……
2		……
3		**R1 ready**
4	ADD R1, R1, R1	**R1 ready**
5	LD R2, b	**R1 ready**
6		**R1 ready**
7		**R1/R2 ready**
8	MUL R1, R1, R2	**R2 ready**
9		
10		
11		
12		
13		
14		
15		
16		

南京大学编译技术
及软件安全研究组

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	**R2,**	**c**	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

schedule →

1	LD R1, a	……
2		……
3		**R1 ready**
4	ADD R1, R1, R1	**R1 ready**
5	LD R2, b	**R1 ready**
6		**R1 ready**
7		**R1/R2 ready**
8	MUL R1, R1, R2	**R2 ready**
9		**R1/R2 ready**
10		
11		
12		
13		
14		
15		
16		

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	**R2,**	**c**	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

schedule

1	LD R1, a	……
2		……
3		**R1 ready**
4	ADD R1, R1, R1	**R1 ready**
5	LD R2, b	**R1 ready**
6		**R1 ready**
7		**R1/R2 ready**
8	MUL R1, R1, R2	**R2 ready**
9		**R1/R2 ready**
10		
11		
12		
13		
14		
15		
16		

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	**R2,**	**c**	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

schedule

1	LD R1, a	……
2		……
3		**R1 ready**
4	ADD R1, R1, R1	**R1 ready**
5	LD R2, b	**R1 ready**
6		**R1 ready**
7		**R1/R2 ready**
8	MUL R1, R1, R2	**R2 ready**
9		**R1/R2 ready**
10		
11		
12		
13		
14		
15		
16		

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD R1, a
ADD R1, R1, R1
LD R2, b
MUL R1, R1, R2
LD R2, c
MUL R1, R1, R2
ST a, R1

a = 2 * a * b * c

$\xrightarrow{\textit{schedule}}$

1	LD R1, a	……
2		……
3		**R1 ready**
4	ADD R1, R1, R1	**R1 ready**
5	LD R2, b	**R1 ready**
6		**R1 ready**
7		**R1/R2 ready**
8	MUL R1, R1, R2	**R2 ready**
9	LD R2, c	**R1 ready**
10		**R1 ready**
11		**R1/R2 ready**
12		
13		
14		
15		
16		

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

$\xrightarrow{\text{schedule}}$

1	LD R1, a	……
2		……
3		**R1 ready**
4	ADD R1, R1, R1	**R1 ready**
5	LD R2, b	**R1 ready**
6		**R1 ready**
7		**R1/R2 ready**
8	MUL R1, R1, R2	**R2 ready**
9	LD R2, c	**R1 ready**
10		**R1 ready**
11		**R1/R2 ready**
12	MUL R1, R1, R2	**R2 ready**
13		**R1/R2 ready**
14		
15		
16		

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

schedule $\Rightarrow$

1	LD R1, a	……
2		……
3		**R1 ready**
4	ADD R1, R1, R1	**R1 ready**
5	LD R2, b	**R1 ready**
6		**R1 ready**
7		**R1/R2 ready**
8	MUL R1, R1, R2	**R2 ready**
9	LD R2, c	**R1 ready**
10		**R1 ready**
11		**R1/R2 ready**
12	MUL R1, R1, R2	**R2 ready**
13		**R1/R2 ready**
14	ST a, R1	……
15		……
16		Done!

Instruction Schee

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

$\xrightarrow{\text{schedule}}$

1	LD R1, a	……
2		……
3		**R1 ready**
4	ADD R1, R1, R1	**R1 ready**
5	LD R2, b	**R1 ready**
6		**R1 ready**
7		**R1/R2 ready**
8	MUL R1, R1, R2	**R2 ready**
9	LD R2, c	**R1 ready**
10		**R1 ready**
11		**R1/R2 ready**
12	MUL R1, R1, R2	**R2 ready**
13		**R1/R2 ready**
14	ST a, R1	……
15		……
16		Done!

Instruction Scheduling

- Can a compiler generate better code?

Instruction Scheduling

- **Example: Superscalar machine (1 functional unit)**
  - LD/ST 3 cycles; MUL 2 cycles; ADD 1 cycle

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

Instruction Scheduling

- **Example: Superscalar machine (1 functional unit)**
  - LD/ST 3 cycles; MUL 2 cycles; ADD 1 cycle

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

LD	R1,	a	
LD	R2,	b	
LD	**R3**,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	**R3**
ST	a,	R1	

a = 2 * a * b * c

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	R3
ST	a,	R1	

a = 2 * a * b * c

$\xrightarrow{\textit{schedule}}$

1	LD R1, a	……
2		……
3		**R1 ready**
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		

Instruction Sche

- **Example: Superscalar machi**
    - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	R3
ST	a,	R1	

a = 2 * a * b * c

schedule

1	LD R1, a	……
2		……
3		**R1 ready**
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		

南京大学编译技术
及软件安全研究组

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	R3
ST	a,	R1	

a = 2 * a * b * c

schedule

1	LD R1, a	……
2	LD R2, b	……
3		R1 ready
4		R1/R2 ready
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		

南京大学编译技术
及软件安全研究组

Instruction Schec

- **Example: Superscalar machir**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	R3
ST	a,	R1	

a = 2 * a * b * c

$\xrightarrow{\text{schedule}}$

1	LD R1, a	……
2	LD R2, b	……
3	LD R3, c	**R1 ready**
4		**R1/R2 ready**
5		**R1/R2/R3 ready**
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	R3
ST	a,	R1	

a = 2 * a * b * c

schedule

1	LD R1, a	……
2	LD R2, b	……
3	LD R3, c	**R1 ready**
4	ADD R1, R1, R1	**R1/R2 ready**
5		**R1/R2/R3 ready**
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		

Instruction Schee

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	R3
ST	a,	R1	

a = 2 * a * b * c

schedule

1	LD R1, a	……
2	LD R2, b	……
3	LD R3, c	**R1 ready**
4	ADD R1, R1, R1	**R1/R2 ready**
5	MUL R1, R1, R2	**R2/R3 ready**
6		**R1/R2/R3 ready**
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	R3
ST	a,	R1	

a = 2 * a * b * c

$\xrightarrow{schedule}$

1	LD R1, a	......
2	LD R2, b	......
3	LD R3, c	**R1 ready**
4	ADD R1, R1, R1	**R1/R2 ready**
5	MUL R1, R1, R2	**R2/R3 ready**
6		**R1/R2/R3 ready**
7	MUL R1, R1, R3	**R2/R3 ready**
8		**R1/R2/R3 ready**
9		
10		
11		
12		
13		
14		
15		
16		

南京大学编译技术
及软件安全研究组

Instruction Schec

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	R3
ST	a,	R1	

a = 2 * a * b * c

schedule →

1	LD R1, a	……
2	LD R2, b	……
3	LD R3, c	**R1 ready**
4	ADD R1, R1, R1	**R1/R2 ready**
5	MUL R1, R1, R2	**R2/R3 ready**
6		**R1/R2/R3 ready**
7	MUL R1, R1, R3	**R2/R3 ready**
8		**R1/R2/R3 ready**
9	ST a, R1	……
10		……
11		Done!
12		
13		
14		
15		
16		

Instruction Sche

- **Example: Superscalar machi**
  - LD/ST 3 cycles; MUL 2 cycles; AD

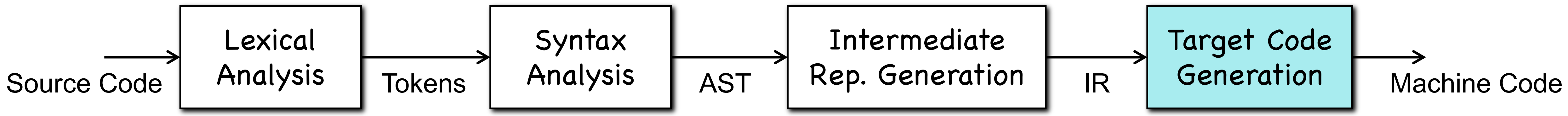
LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	R3
ST	a,	R1	

a = 2 * a * b * c

schedule

1	LD R1, a	……
2	LD R2, b	……
3	LD R3, c	**R1 ready**
4	ADD R1, R1, R1	**R1/R2 ready**
5	MUL R1, R1, R2	**R2/R3 ready**
6		**R1/R2/R3 ready**
7	MUL R1, R1, R3	**R2/R3 ready**
8		**R1/R2/R3 ready**
9	ST a, R1	……
10		……
11		Done!
12		
13		
14		
15		
16		

Instruction Scheduling

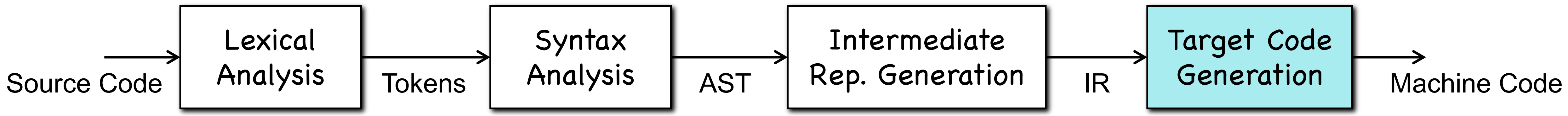


Can we generate code to leverage modern CPUs for better instruction-level parallelization? **Yes! Possible!**

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

a = 2 * a * b * c

Instruction Scheduling



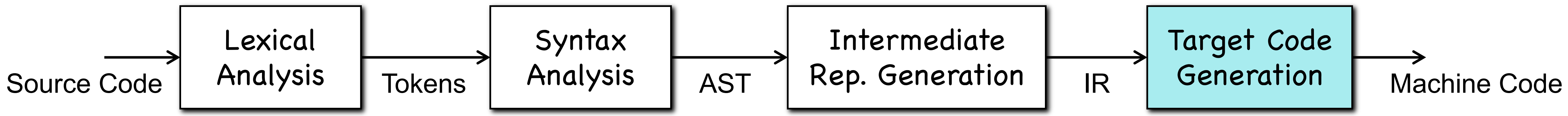
Can we generate code to leverage modern CPUs for better instruction-level parallelization? **Yes! Possible!**

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	R3
ST	a,	R1	

a = 2 * a * b * c

Instruction Scheduling



Can we generate code to leverage modern CPUs for better instruction-level parallelization? **Yes! Possible!**

- 1. Scheduling constraints
- 2. Basic block scheduling → 3. Global scheduling
- 4. Software pipelining (optional)

PART I: Scheduling Constraints

Performance is Order Dependent

- a = 2 * a * b * c

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

Performance is Order Dependent

- a = 2 * a * b * c

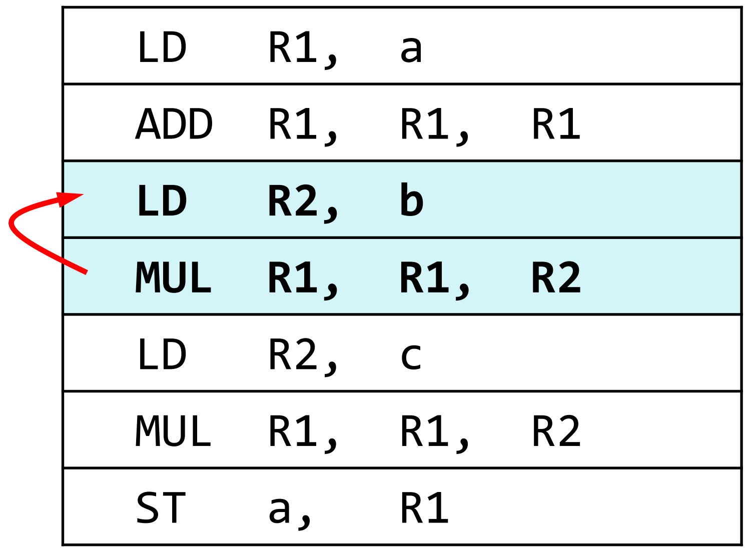
LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	R3
ST	a,	R1	

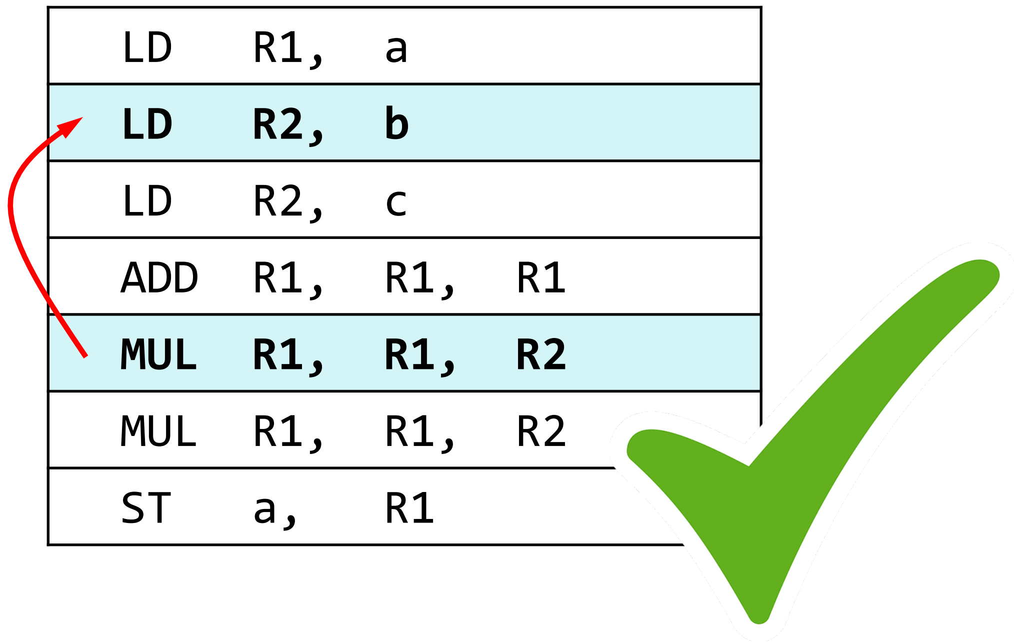
✓

Scheduling Constraints

- True Dependence (Read-After-Write Dependence)



```
LD   R1, a
ADD  R1, R1, R1
LD   R2, b
MUL  R1, R1, R2
LD   R2, c
MUL  R1, R1, R2
ST   a,  R1
```



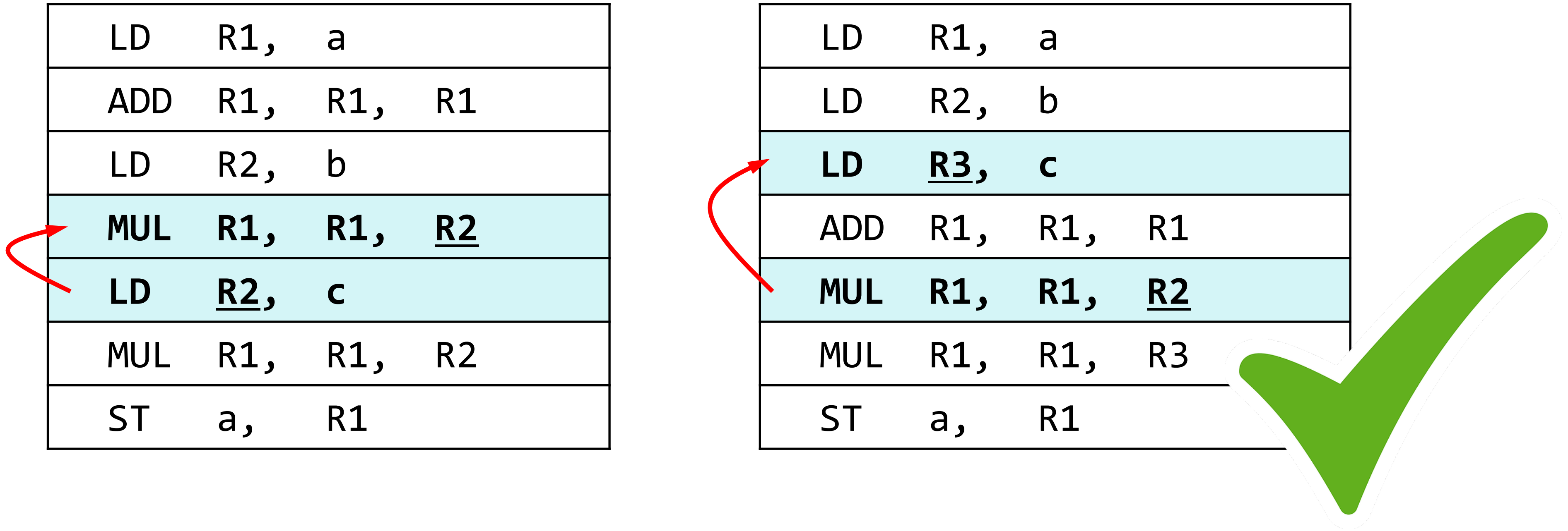
```
LD   R1, a
LD   R2, b
LD   R2, c
ADD  R1, R1, R1
MUL  R1, R1, R2
MUL  R1, R1, R2
ST   a,  R1
```

Scheduling Constraints

- Storage (Fake) Dependence --- Rename to Remove
  - Antidependence & Output dependence

Scheduling Constraints

- Storage (Fake) Dependence --- Rename to Remove
  - **Antidependence (Write-After-Read)** & Output dependence

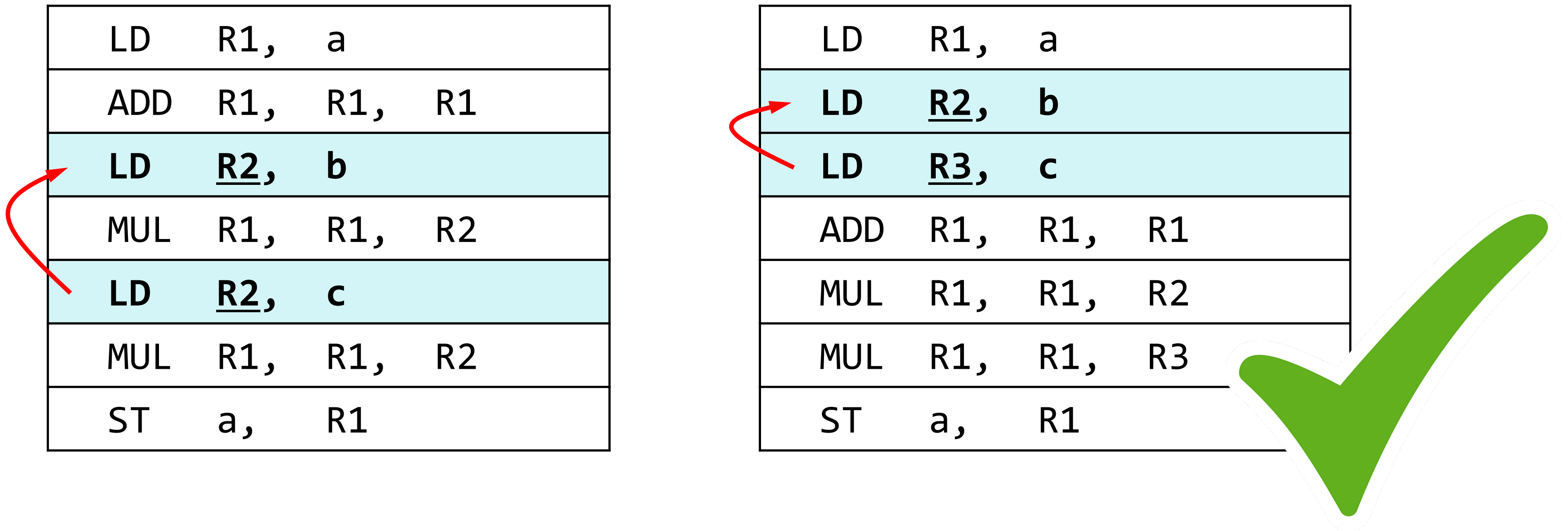


LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	**R1,**	**R1,**	**R2**
LD	**R2,**	**c**	
MUL	R1,	R1,	R2
ST	a,	R1	

LD	R1,	a	
LD	R2,	b	
LD	**R3,**	**c**	
ADD	R1,	R1,	R1
MUL	**R1,**	**R1,**	**R2**
MUL	R1,	R1,	R3
ST	a,	R1	

Scheduling Constraints

- Storage (Fake) Dependence --- Rename to Remove
    - Antidependence & **Output dependence (Write-After-Write)**



```
LD   R1,  a
ADD  R1,  R1,  R1
LD   R2,  b
MUL  R1,  R1,  R2
LD   R2,  c
MUL  R1,  R1,  R2
ST   a,   R1
```

```
LD   R1,  a
LD   R2,  b
LD   R3,  c
ADD  R1,  R1,  R1
MUL  R1,  R1,  R2
MUL  R1,  R1,  R3
ST   a,   R1
```

Register Allocation vs. Scheduling

- Independent instructions can be scheduled in parallel

南京大学编译技术
及软件安全研究组

Register Allocation vs. Scheduling

- Independent instructions can be scheduled in parallel

- We can remove fake dependence by renaming registers

Register Allocation vs. Scheduling

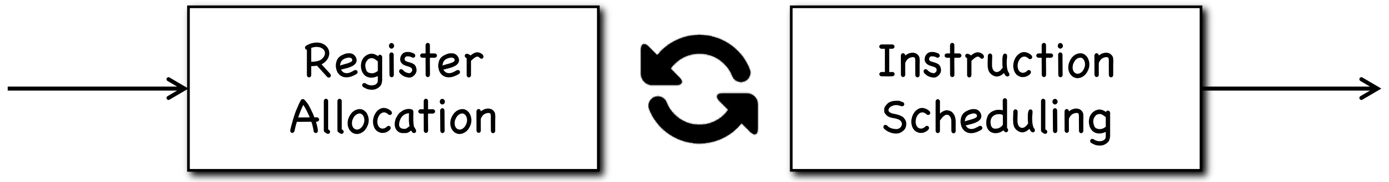
- Independent instructions can be scheduled in parallel
- We can remove fake dependence by renaming registers
- However, …

Register Allocation vs. Scheduling

- Independent instructions can be scheduled in parallel
- We can remove fake dependence by renaming registers
- However, # registers is limited! (Recall register allocation)

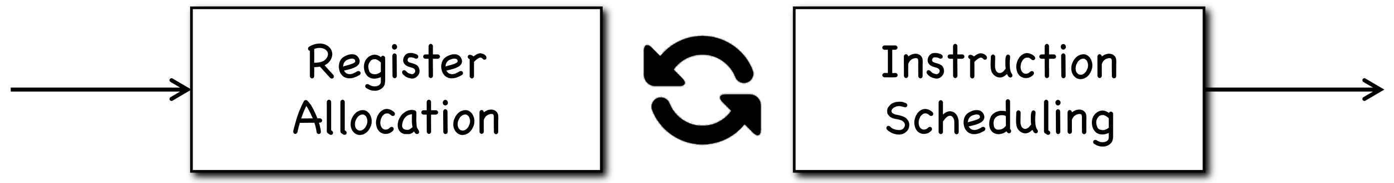
Register Allocation vs. Scheduling

- Which should we do first?
- **Register Allocation**: Designed to use fewer registers
- **Instruction Scheduling**: Prefer more registers



Register Allocation vs. Scheduling

- Which should we do first?
- **Register Allocation**: Designed to use fewer registers
- **Instruction Scheduling**: Prefer more registers



- It's a tough question! It depends on applications
- Multi-objective optimization https://en.wikipedia.org/wiki/Multi-objective_optimization

PART II: Basic Block Scheduling

Dependence Graph

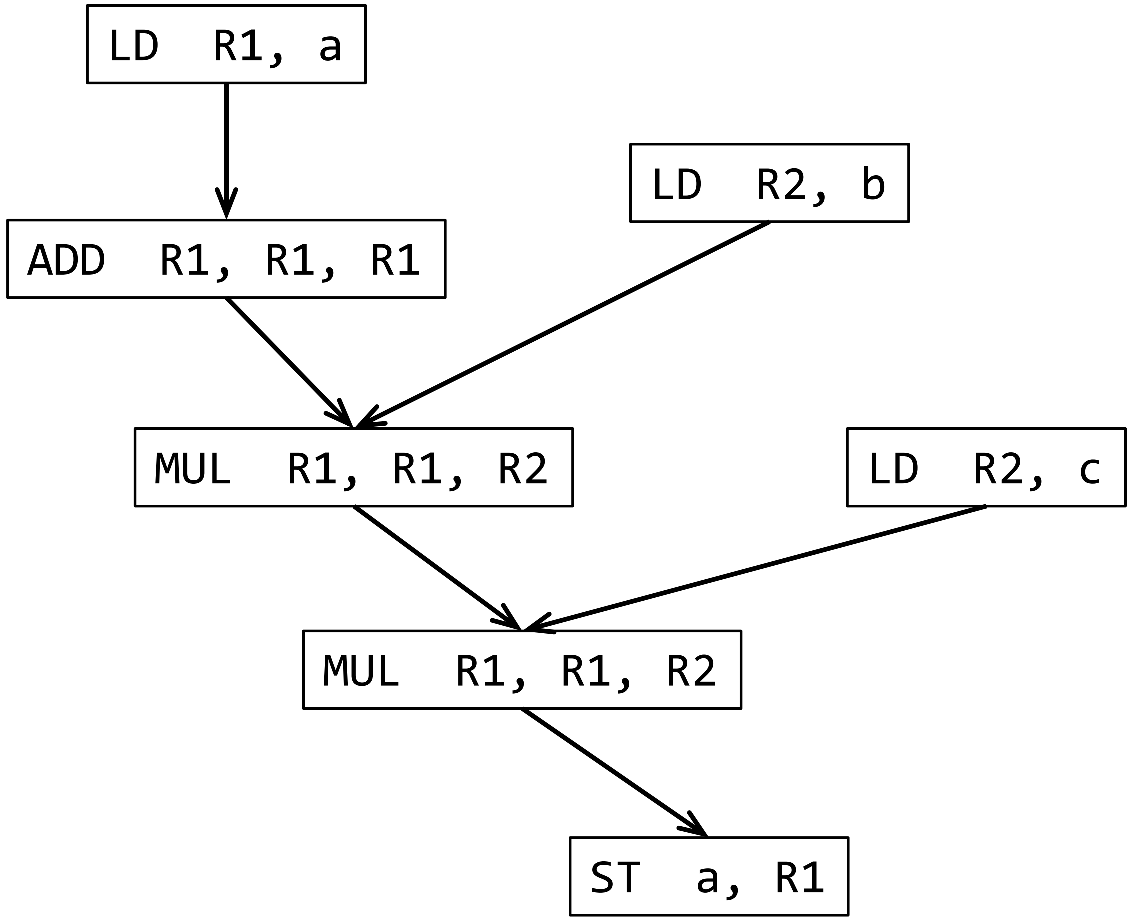
- Node: Instructions
- Edge: True Dependence **(What is a true dependence?)**

Dependence Graph

- Node: Instructions
- Edge: True Dependence **(Read-After-Write Dependence)**

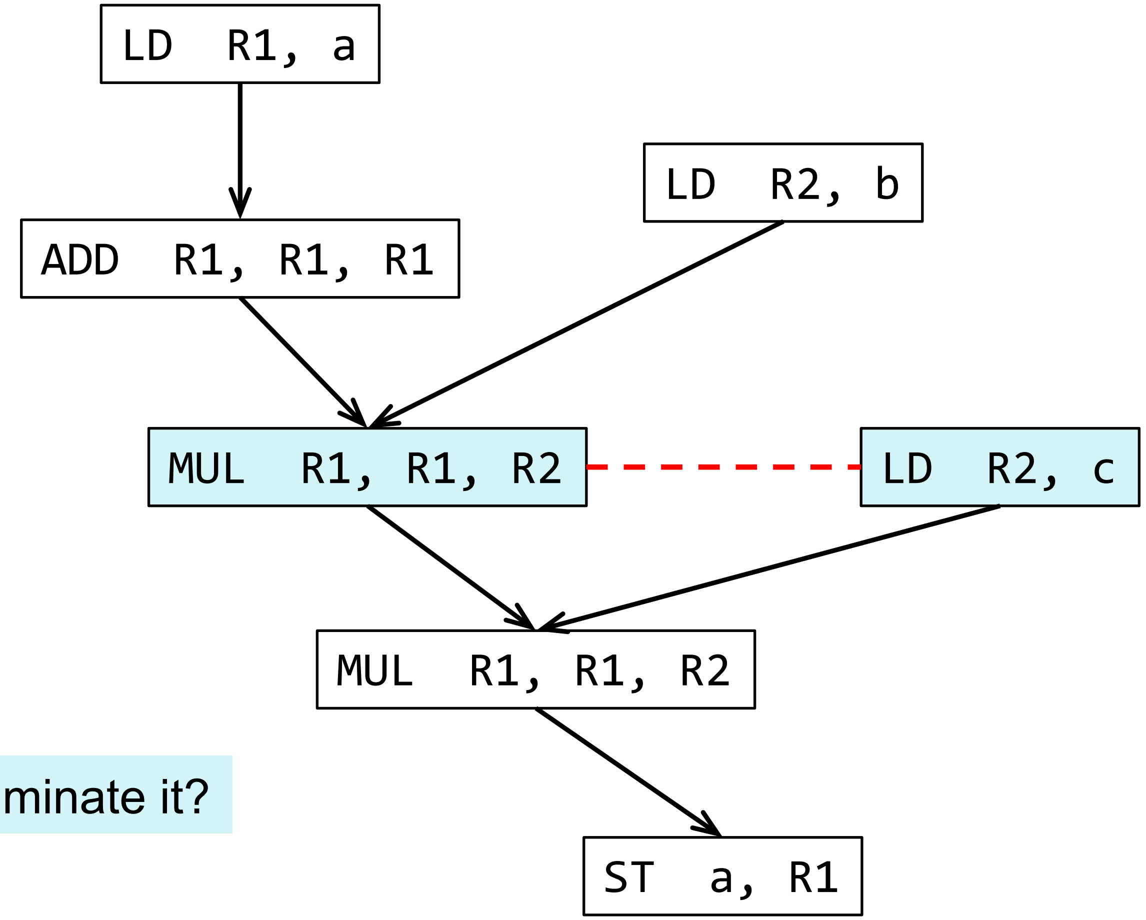
Dependence Graph

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	



Dependence Graph

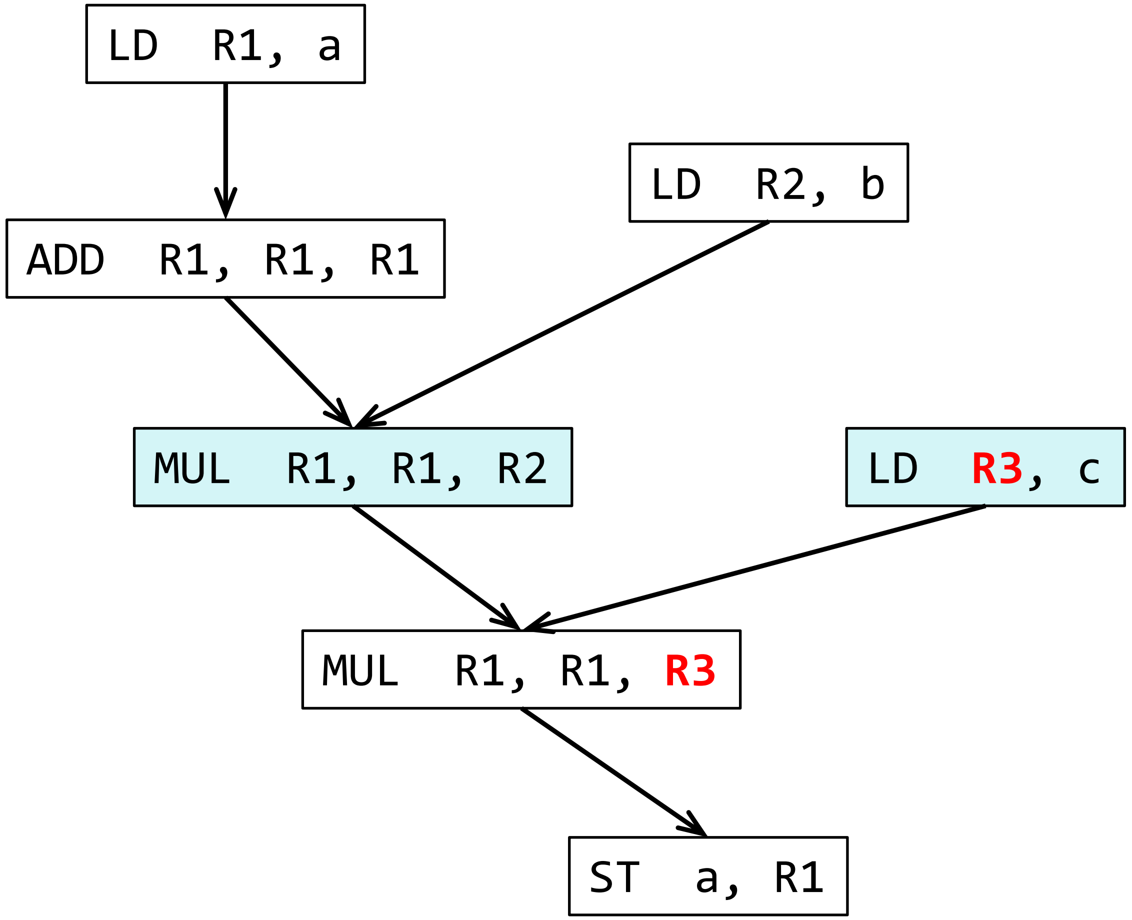
LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	



Recap: Fake dependence. How can we eliminate it?

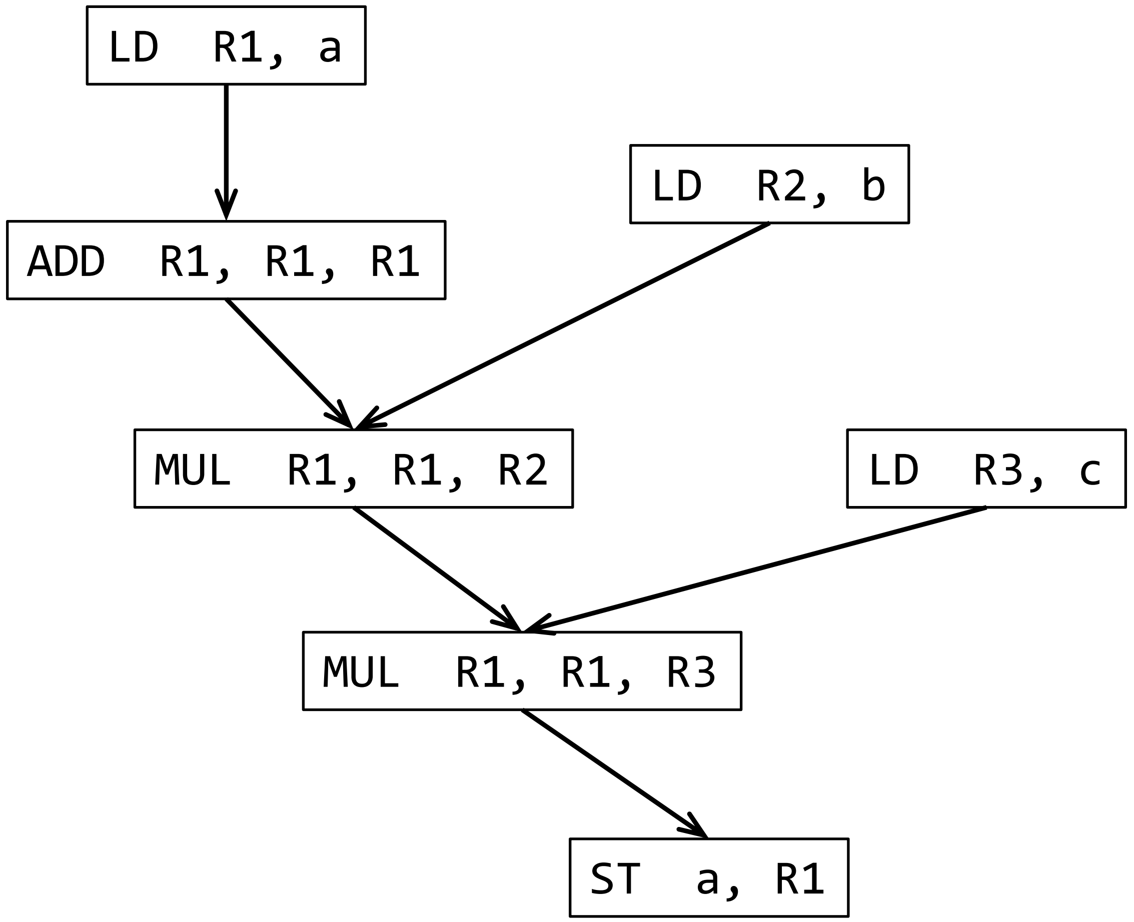
Dependence Graph

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	**R3**,	c	
MUL	R1,	R1,	**R3**
ST	a,	R1	



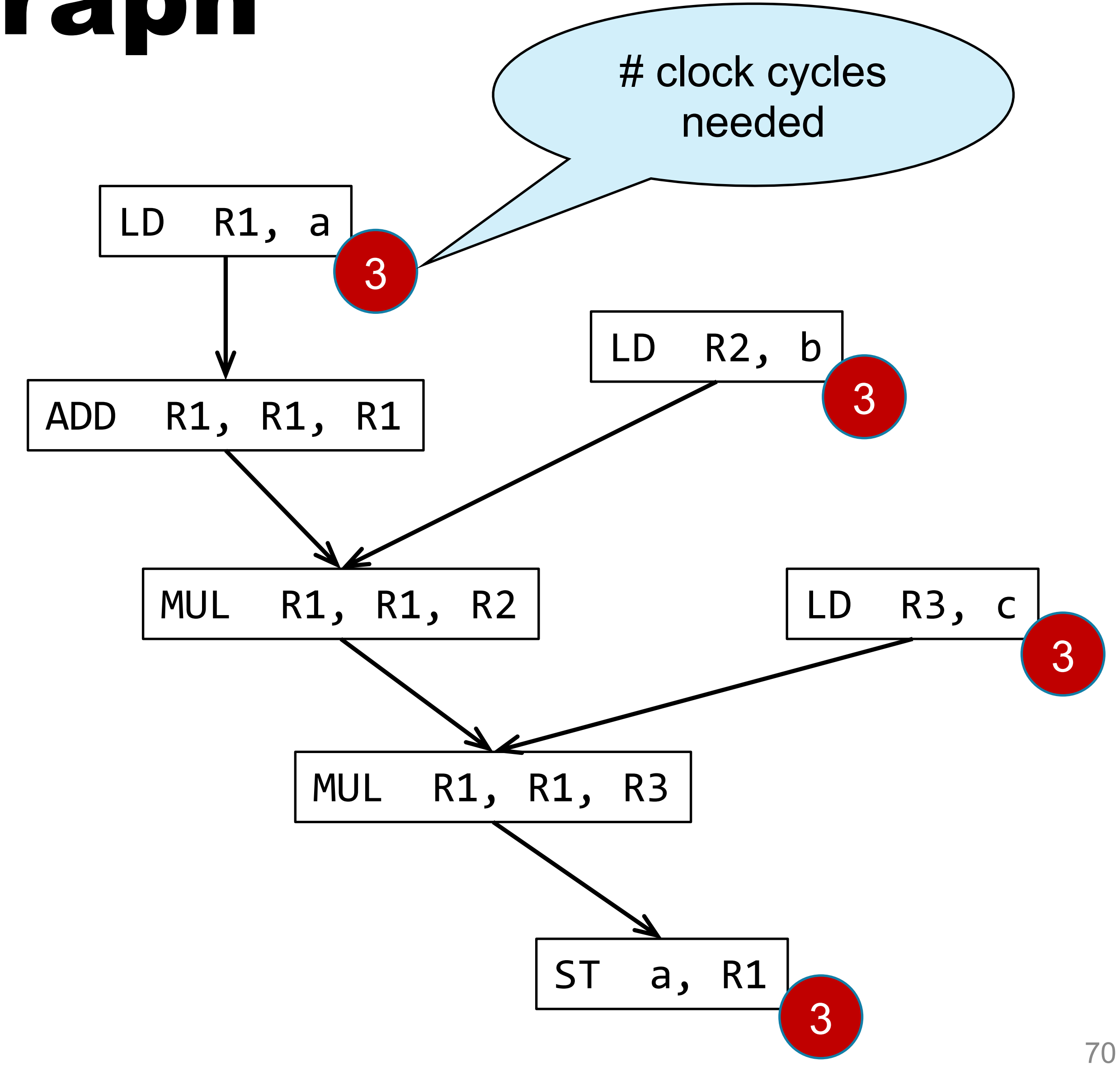
Dependence Graph

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R3,	c	
MUL	R1,	R1,	R3
ST	a,	R1	



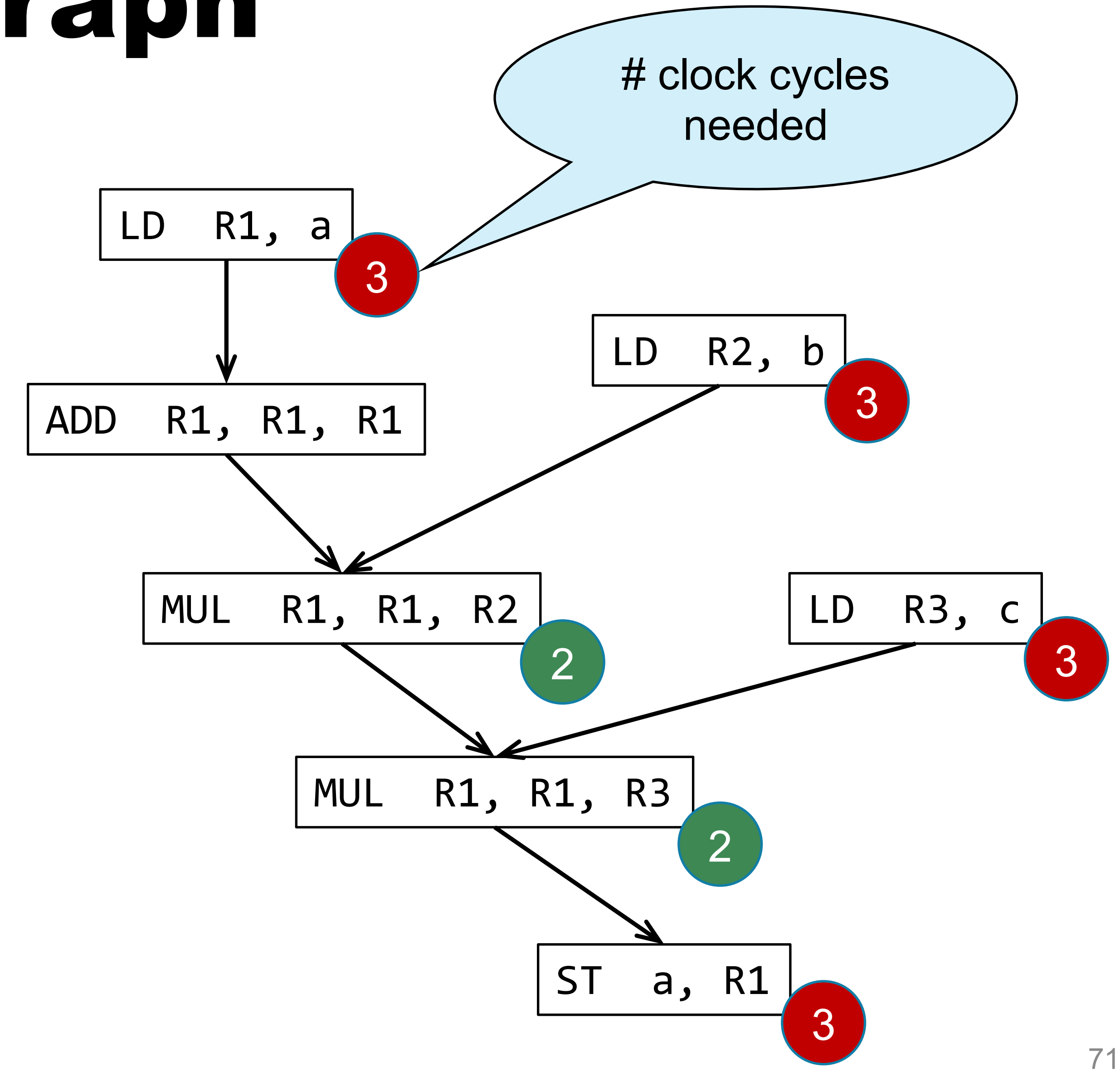
Dependence Graph

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R3,	c	
MUL	R1,	R1,	R3
ST	a,	R1	



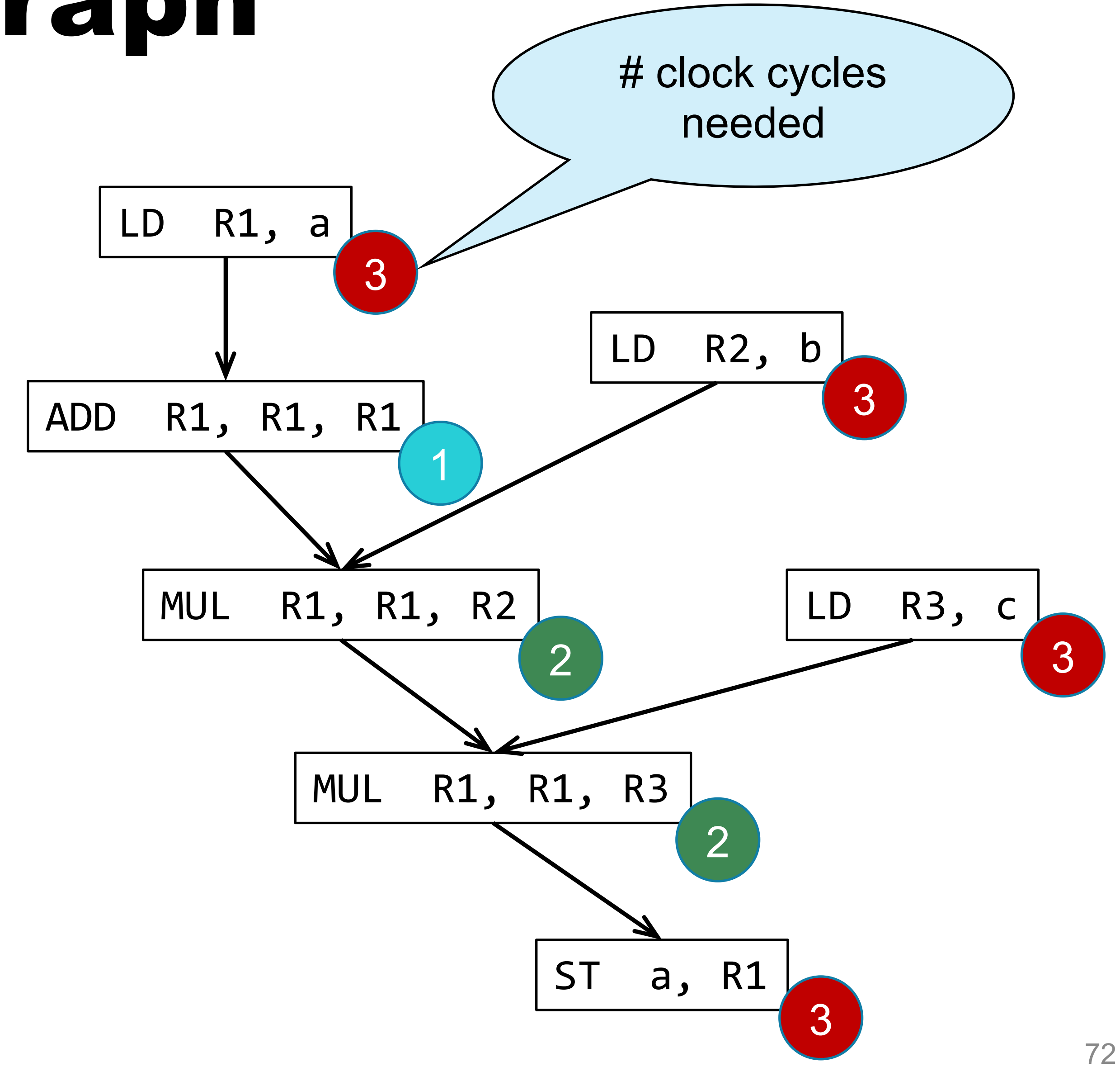
Dependence Graph

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R3,	c	
MUL	R1,	R1,	R3
ST	a,	R1	



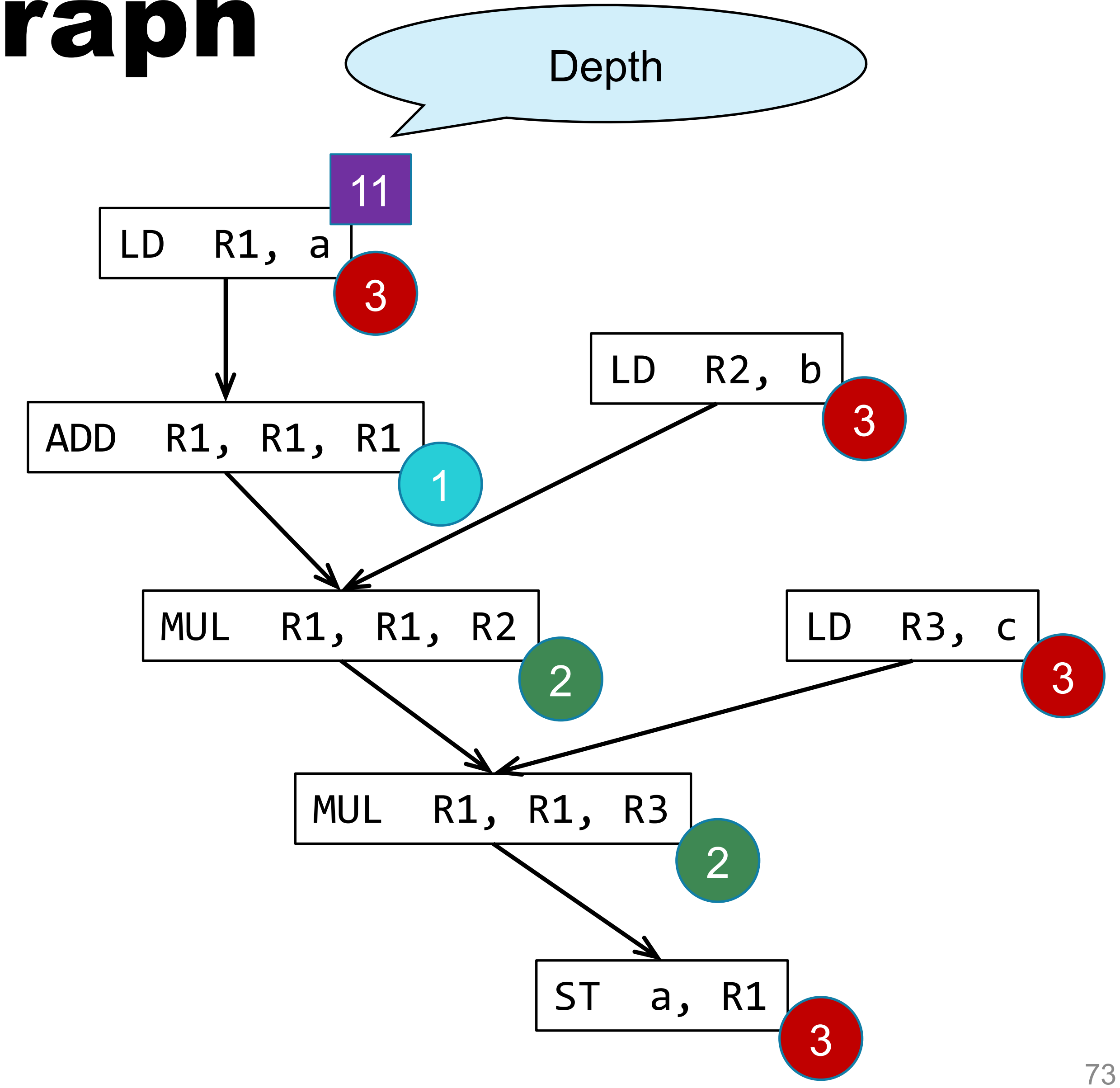
Dependence Graph

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R3,	c	
MUL	R1,	R1,	R3
ST	a,	R1	



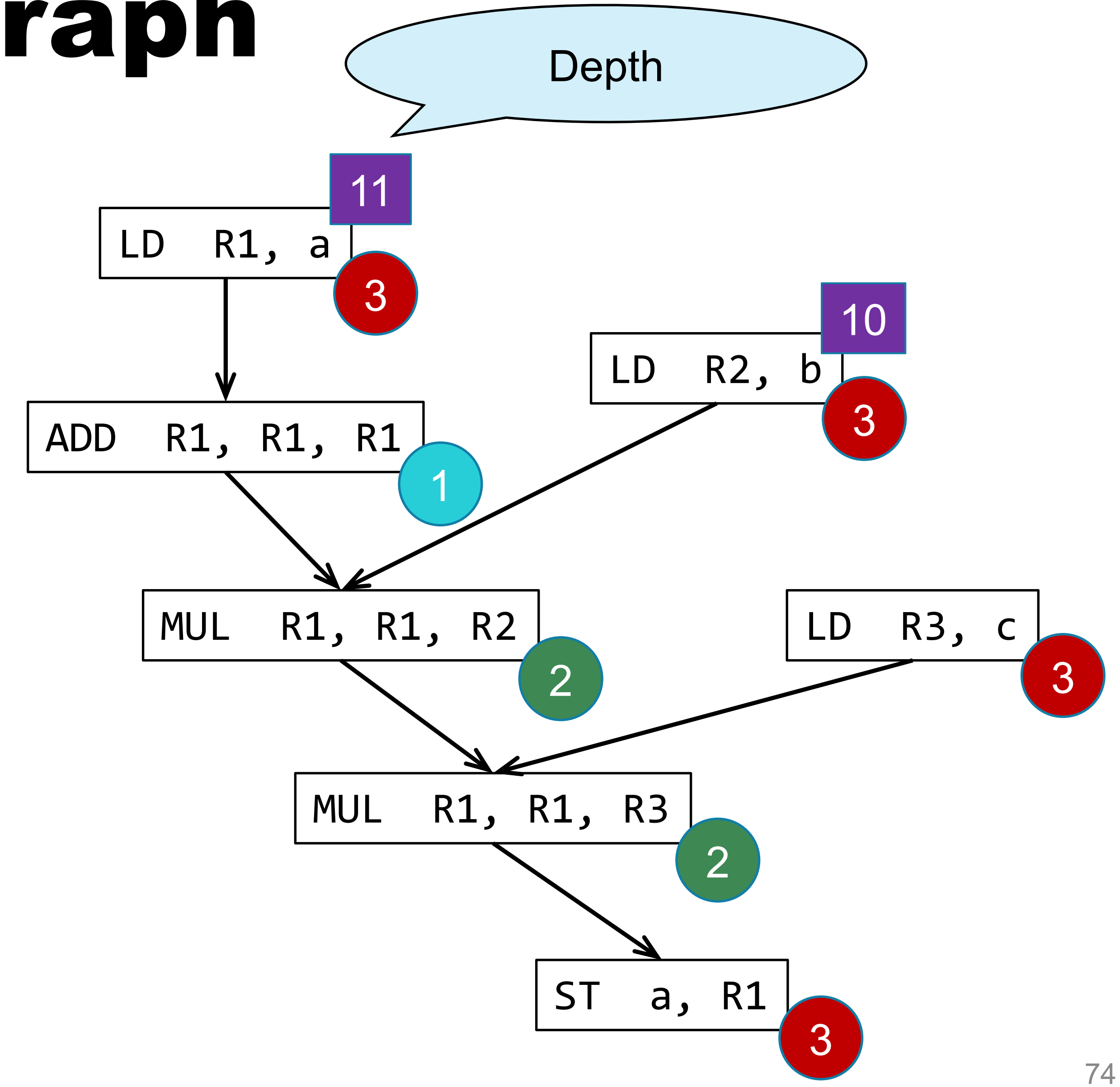
Dependence Graph

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R3,	c	
MUL	R1,	R1,	R3
ST	a,	R1	



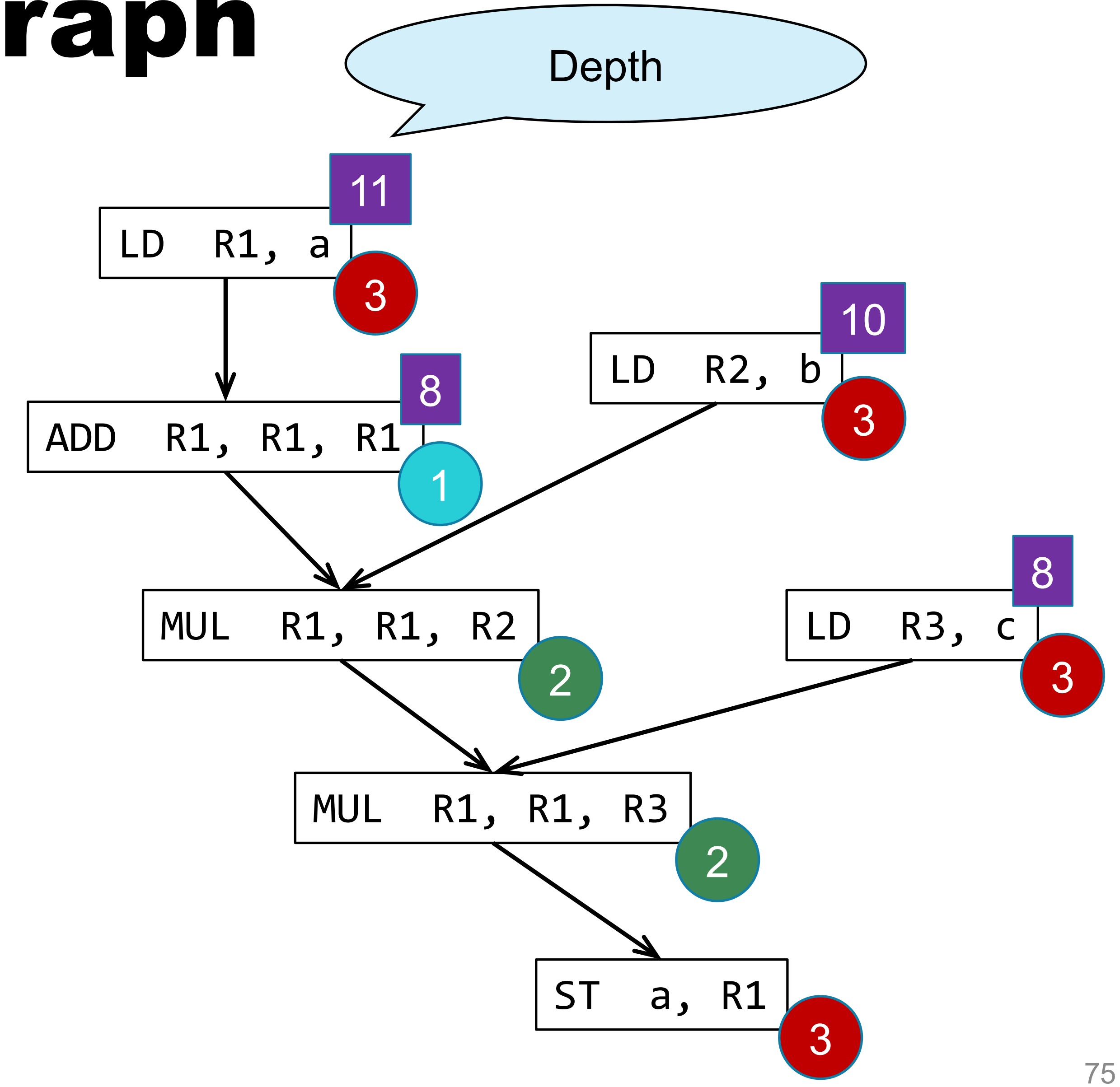
Dependence Graph

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R3,	c	
MUL	R1,	R1,	R3
ST	a,	R1	



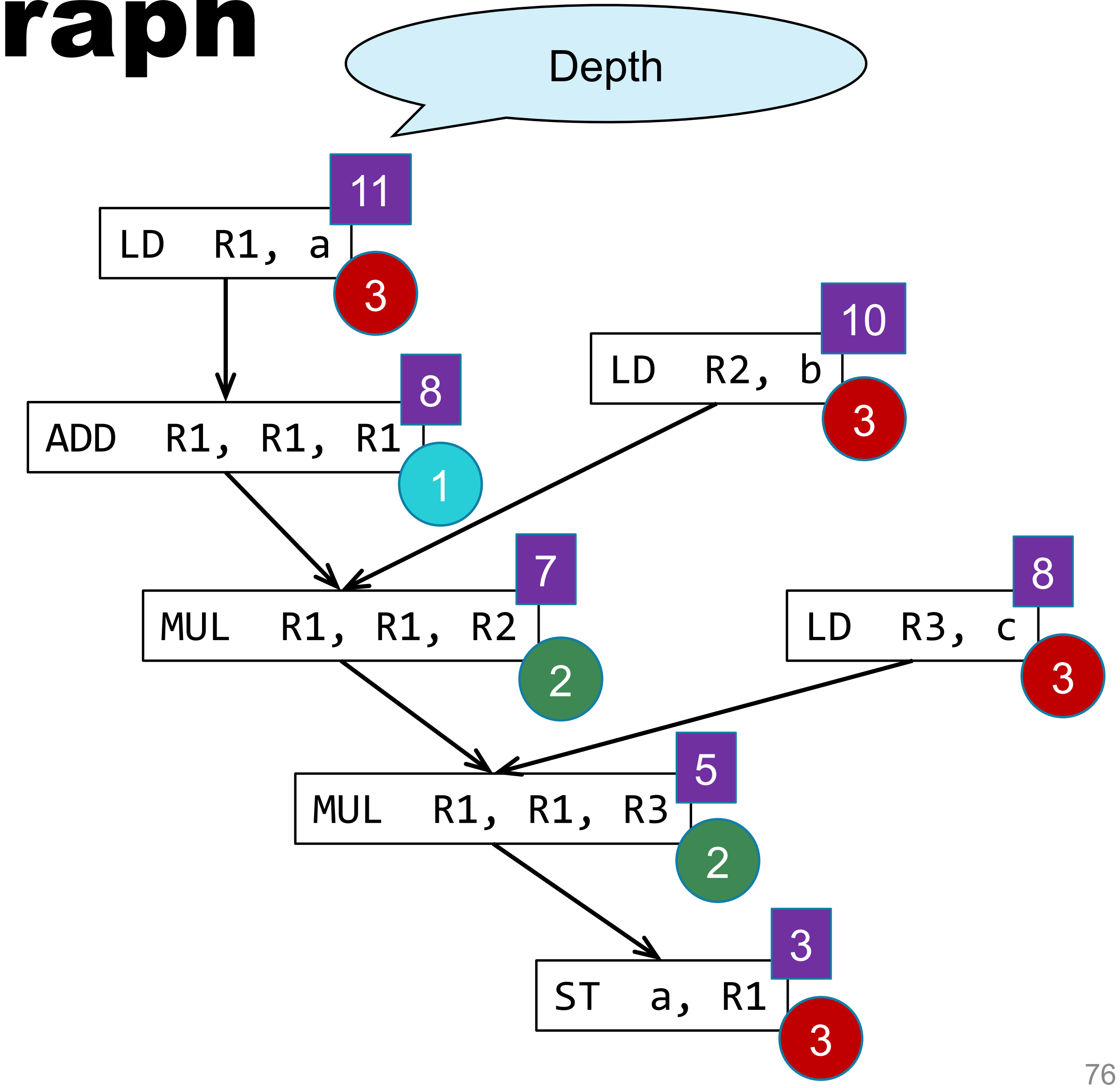
Dependence Graph

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R3,	c	
MUL	R1,	R1,	R3
ST	a,	R1	



Dependence Graph

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R3,	c	
MUL	R1,	R1,	R3
ST	a,	R1	



List Scheduling

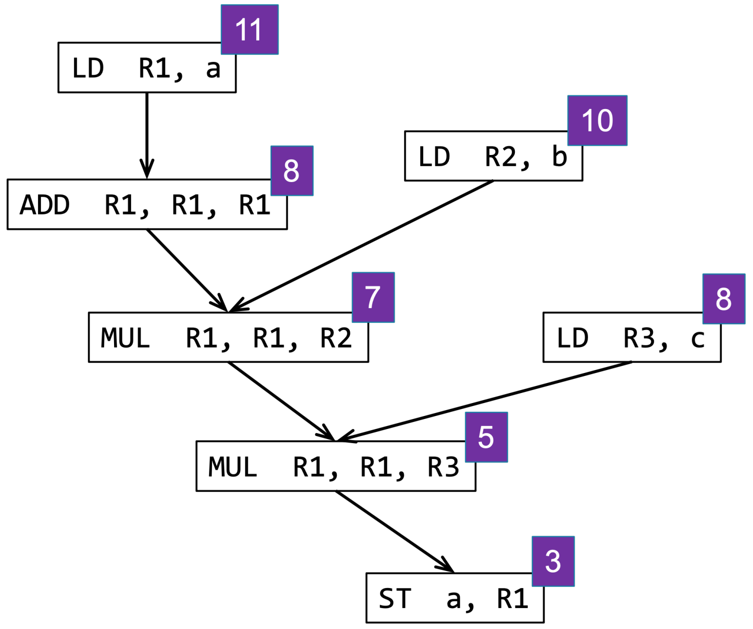
- Check each clock cycle
- Schedule instructions when they are ready

List Scheduling

- Check each clock cycle
- Schedule instructions when they are ready
- When multi instructions can be scheduled, check their **priority**

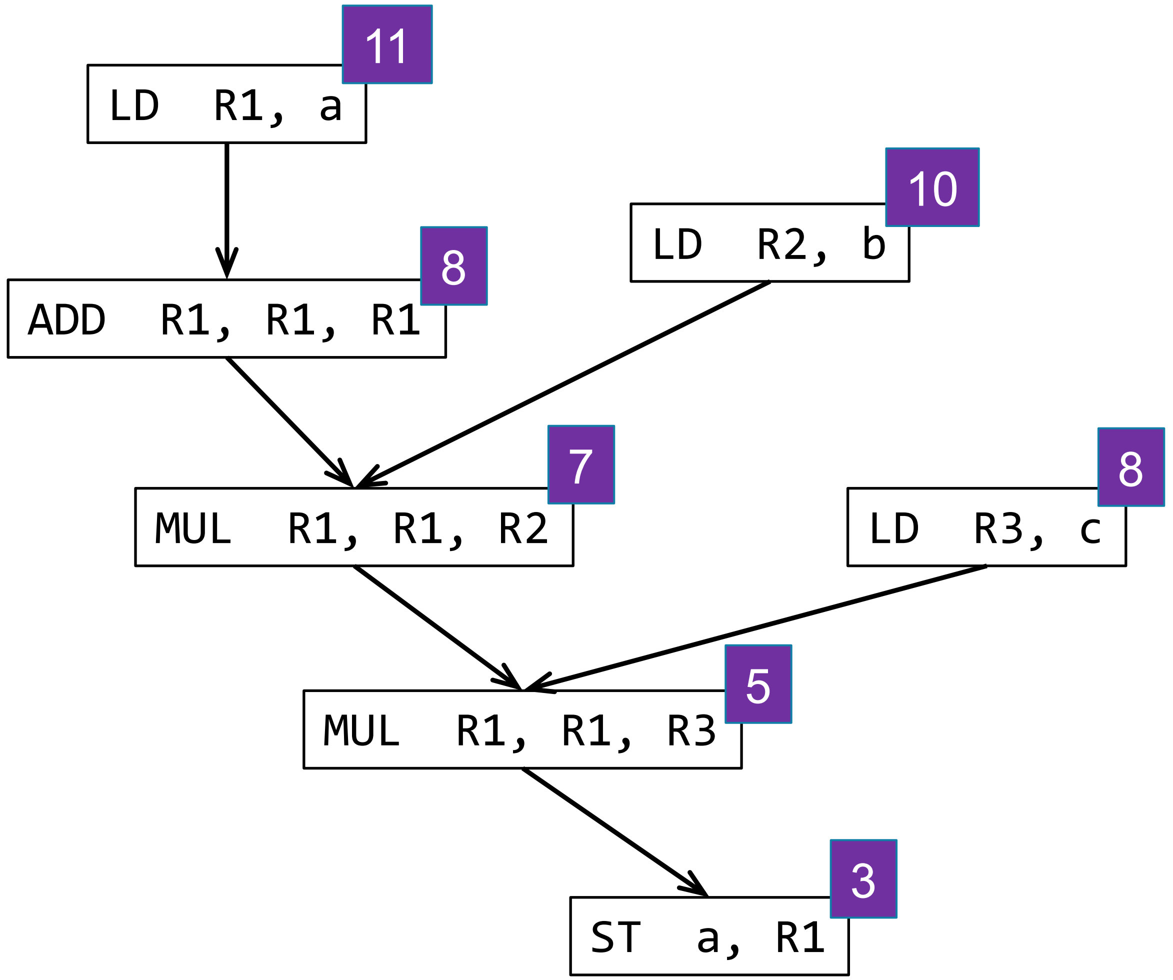
List Scheduling

- Check each clock cycle
- Schedule instructions when they are ready
- When multi instructions can be scheduled, check their **priority**
  - The longest latency path (max depth)
  - Using the most resources
  - …



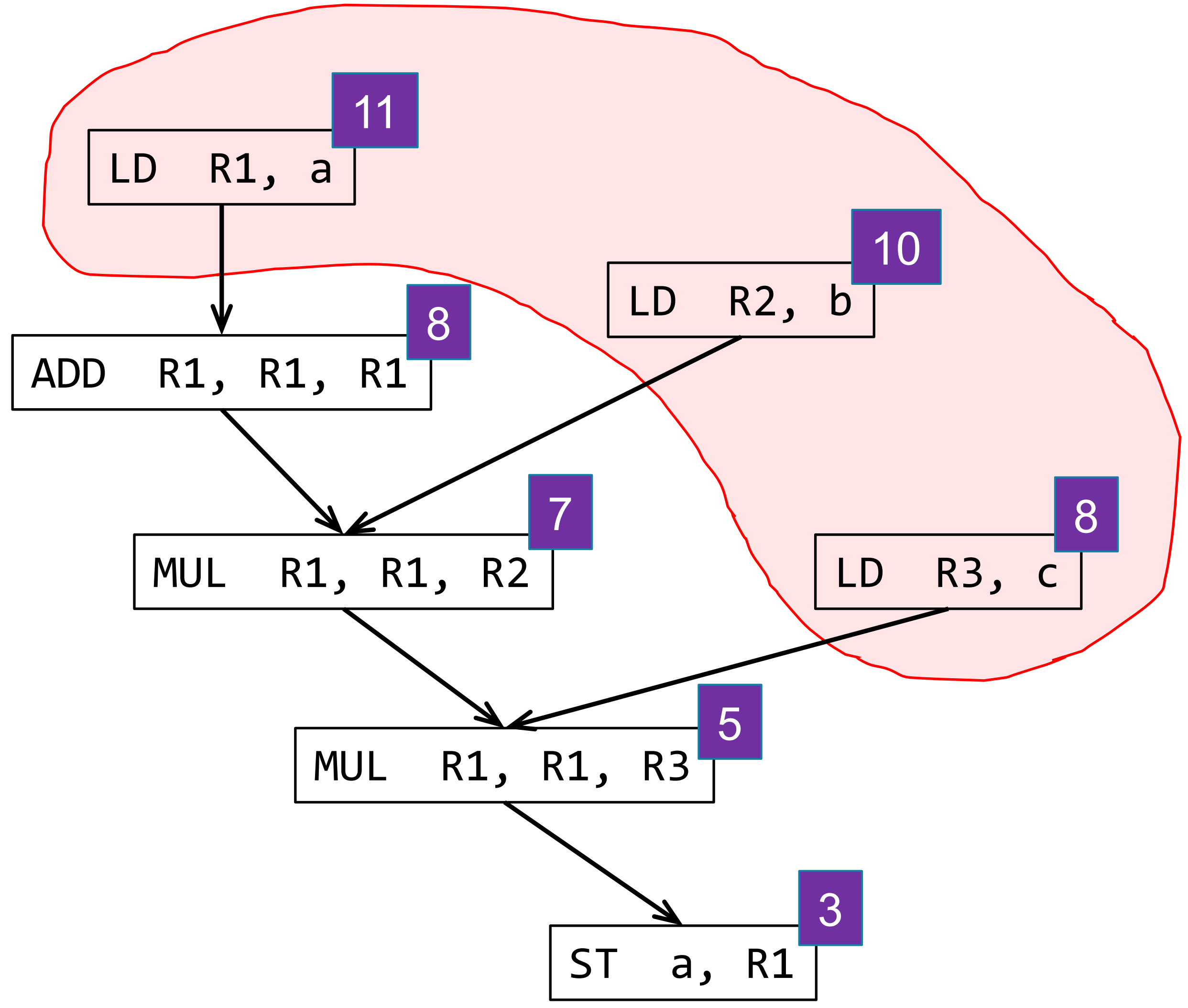
List Scheduling

- Cycle = 1



List Scheduling

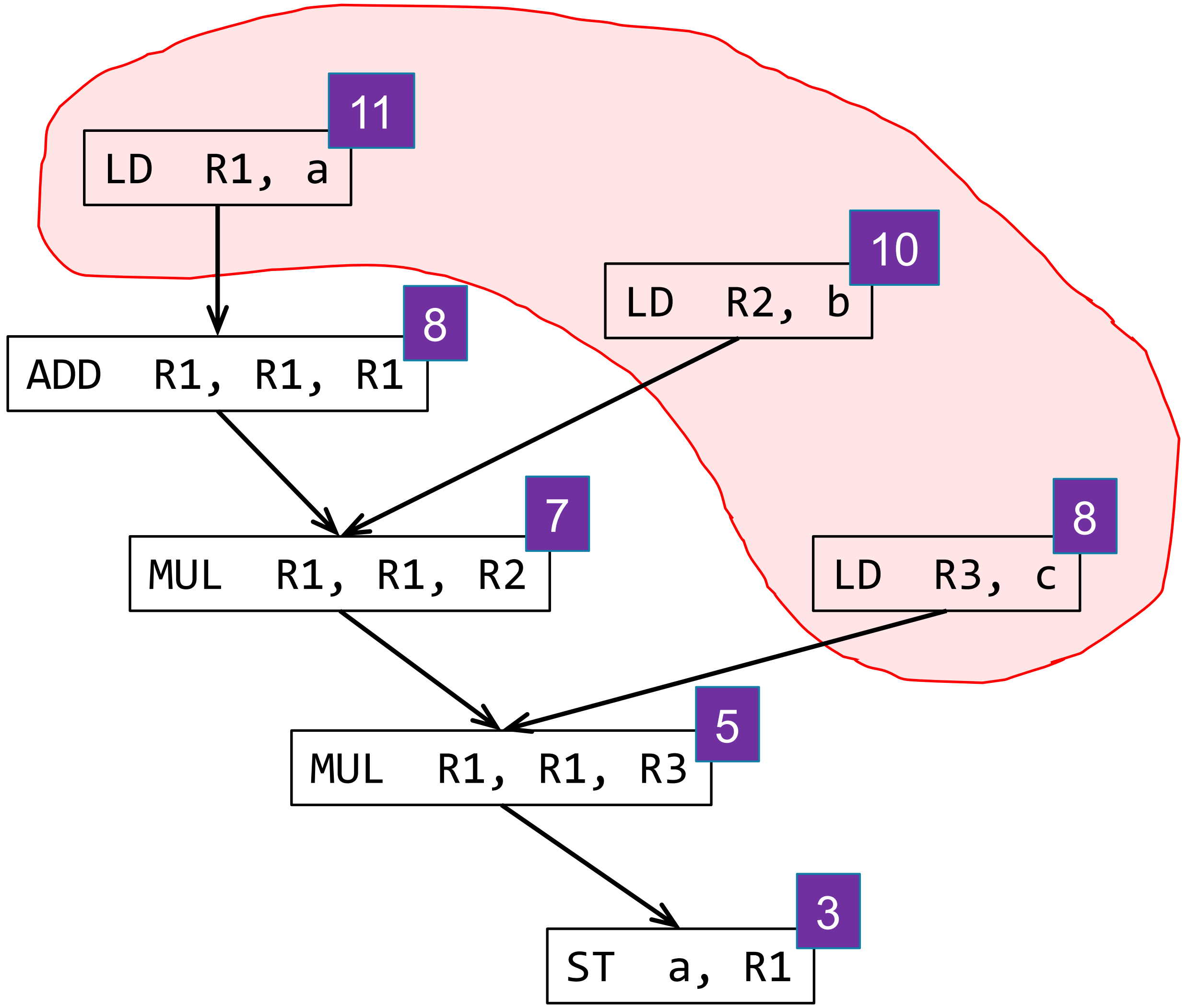
- Cycle = 1



List Scheduling

- Cycle = 1

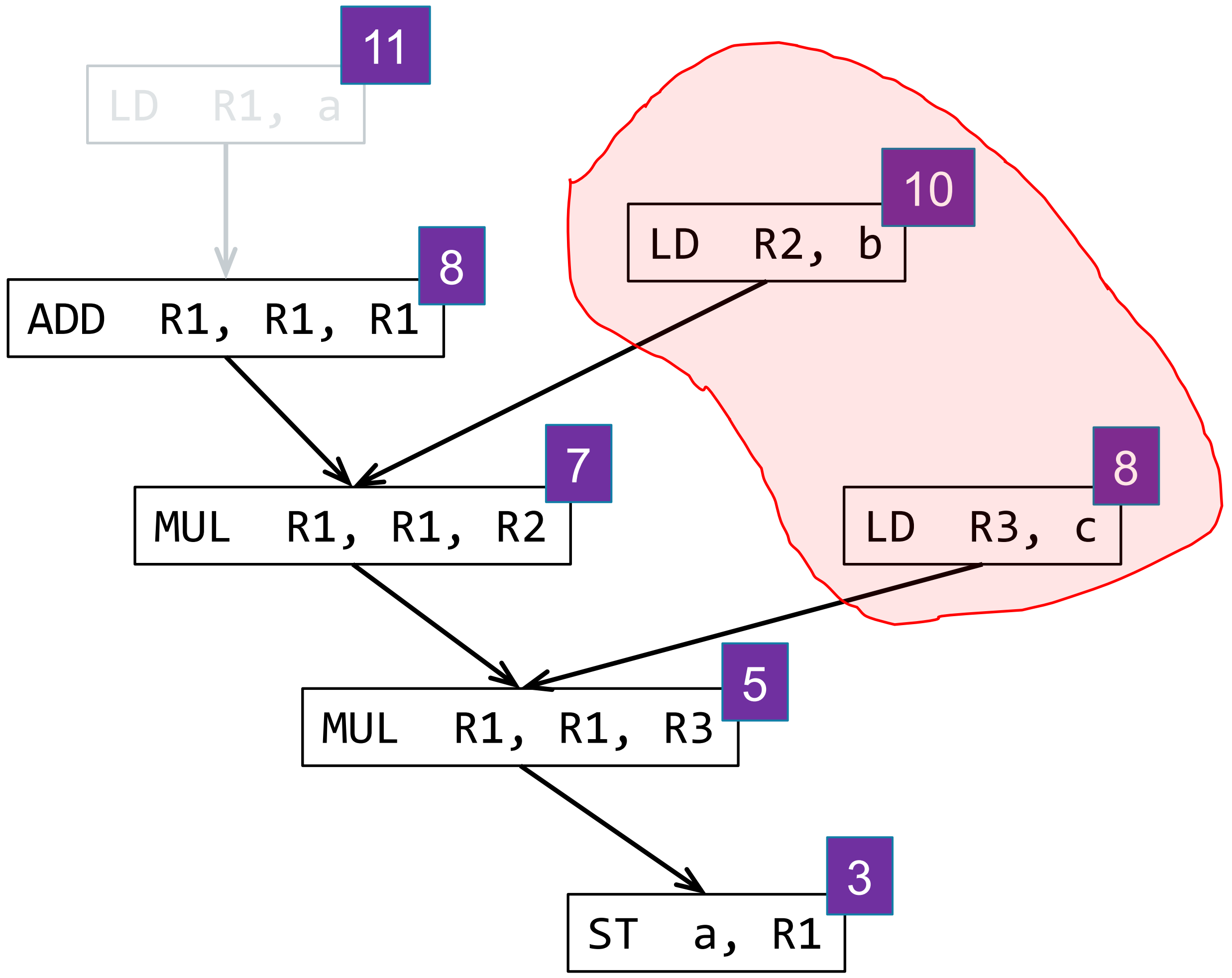
LD R1, a



List Scheduling

- Cycle = 2

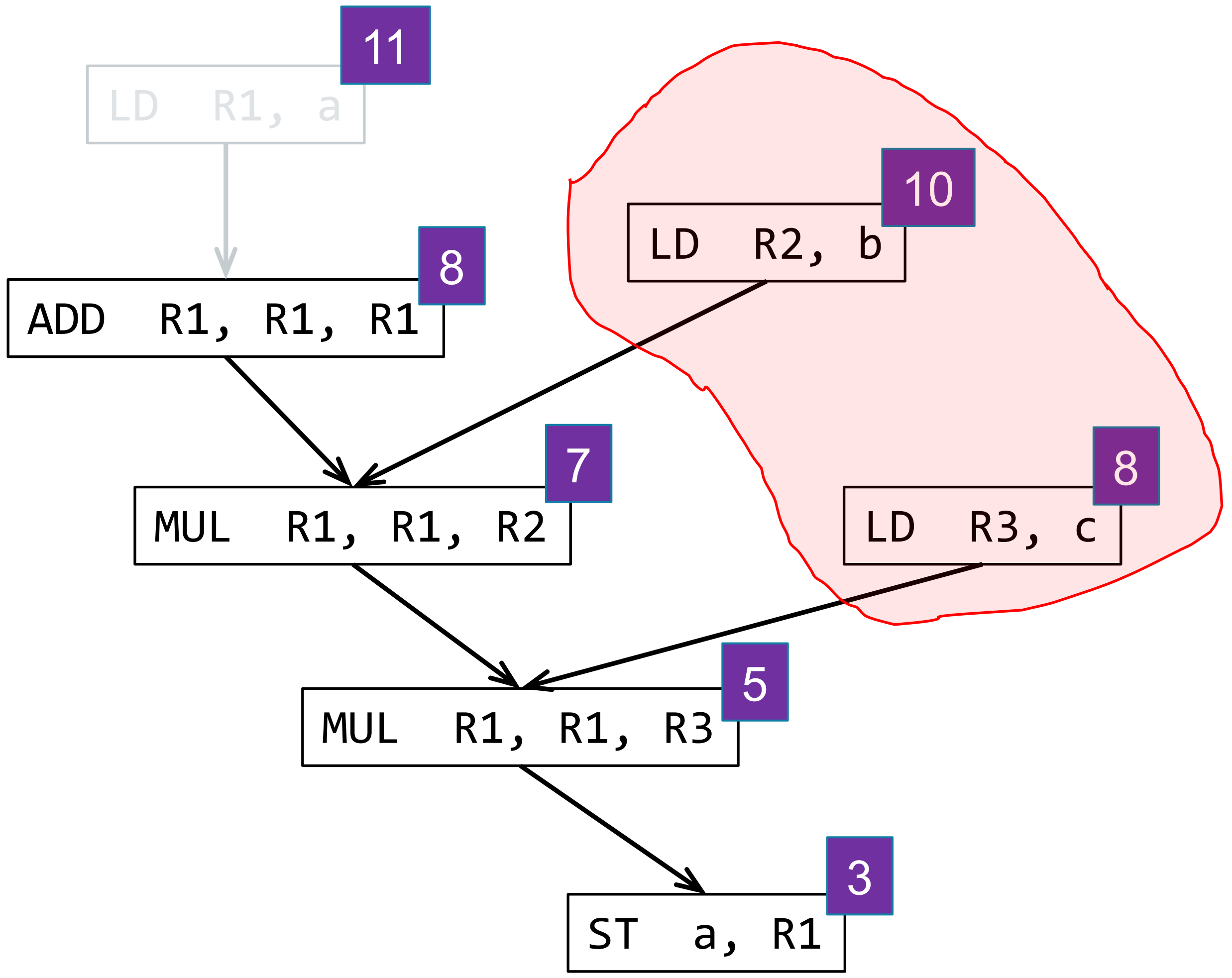
LD R1, a



List Scheduling

- Cycle = 2

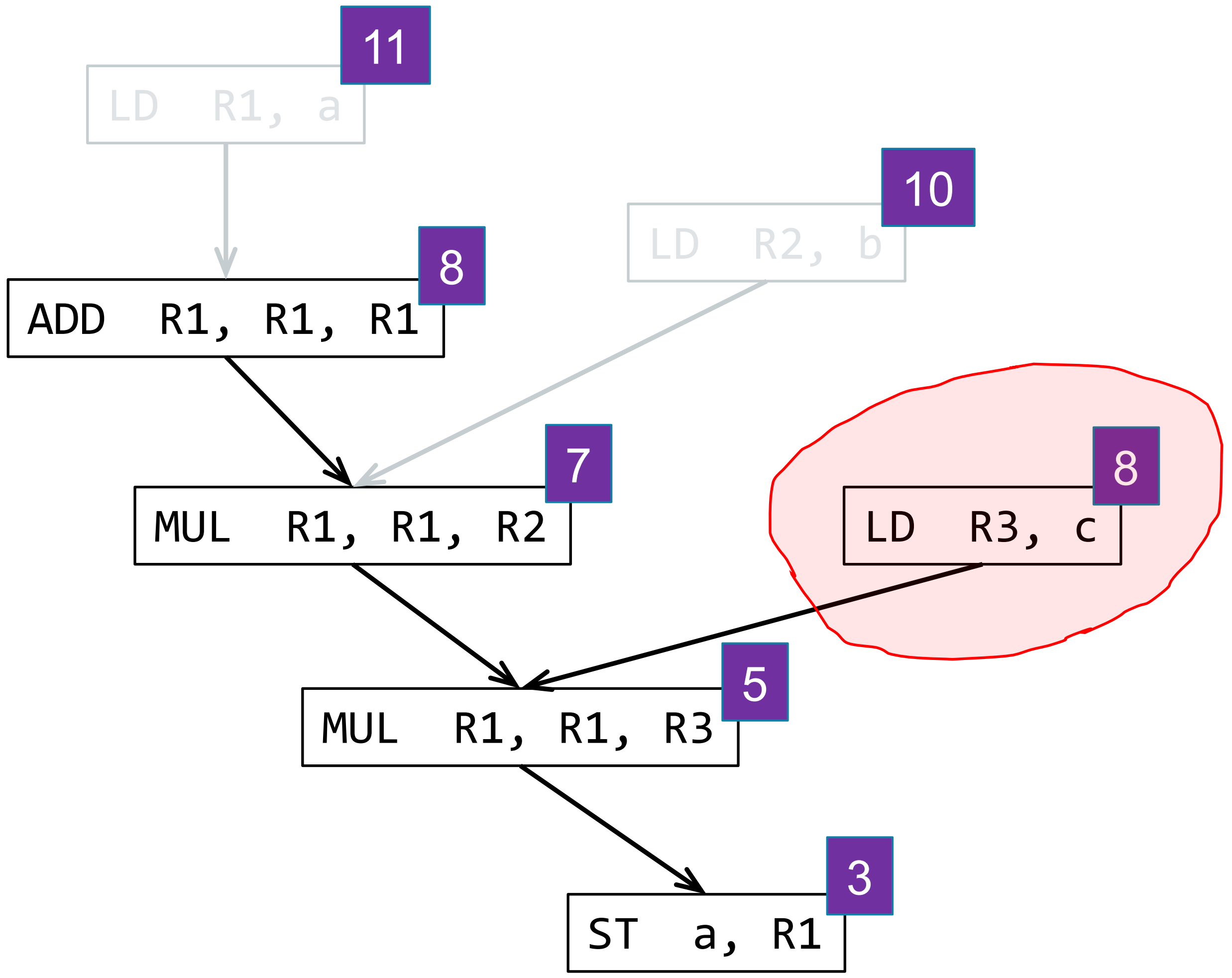
LD R1, a
LD R2, b



List Scheduling

- Cycle = 3

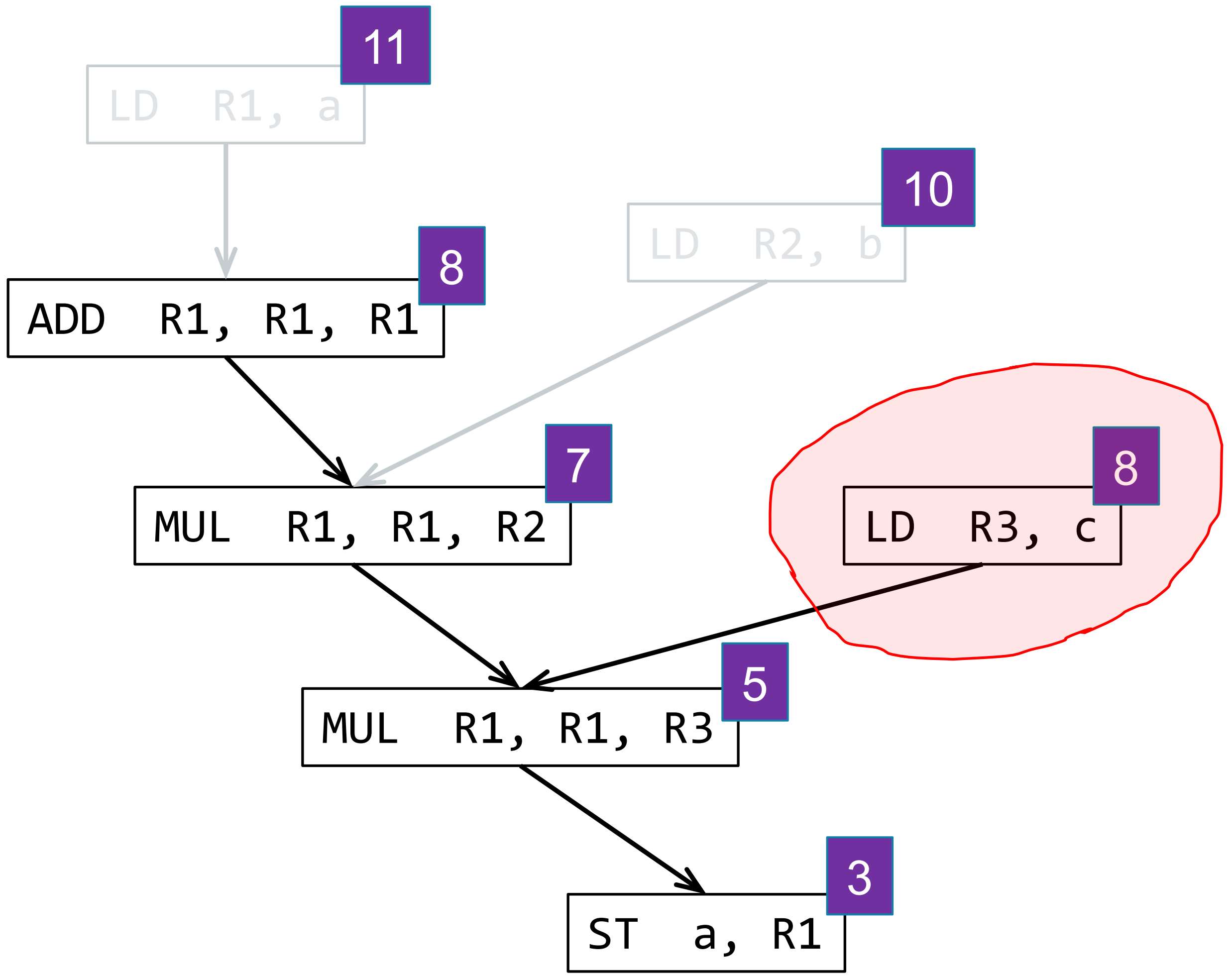
LD R1, a
LD R2, b



List Scheduling

- Cycle = 3

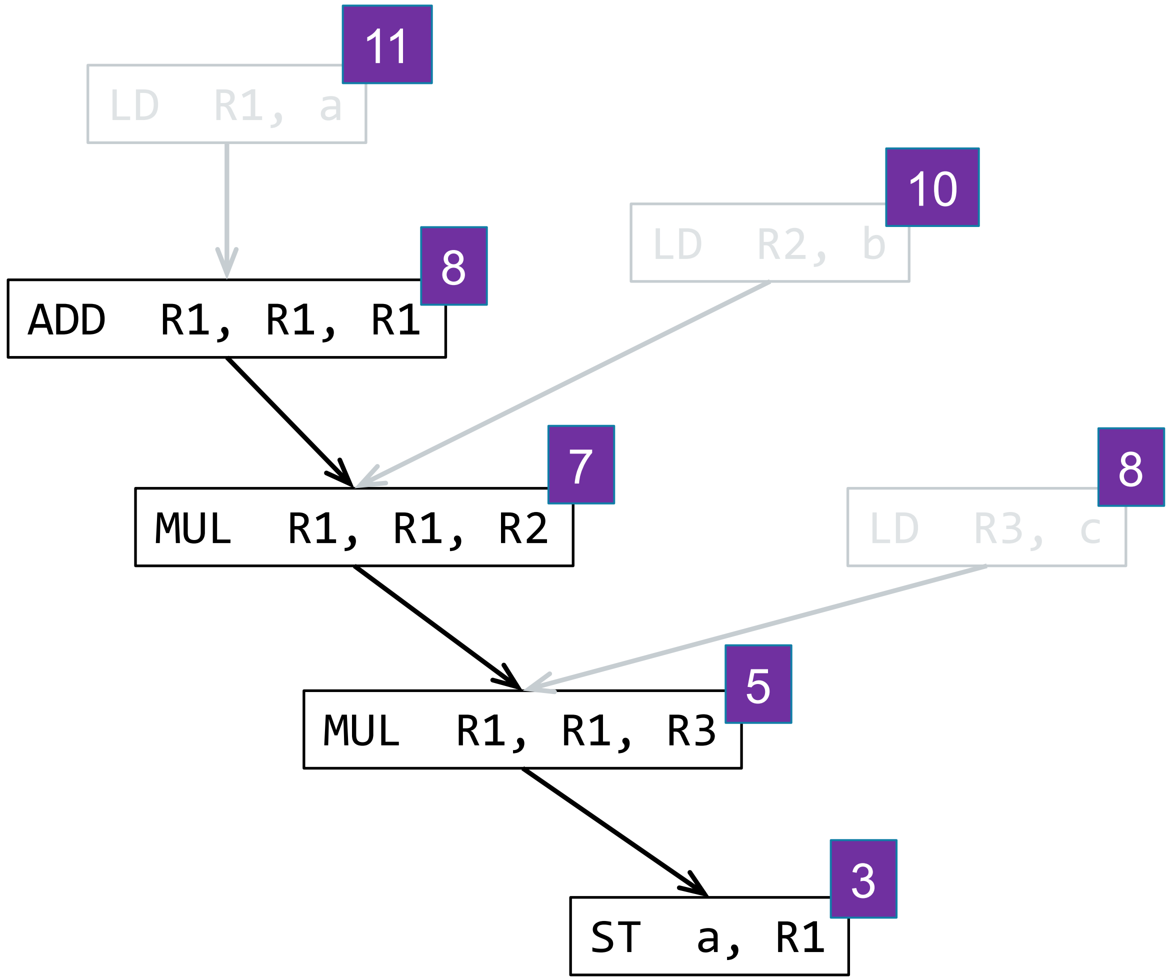
LD	R1,	a
LD	R2,	b
LD	R3,	c



List Scheduling

- Cycle = 4

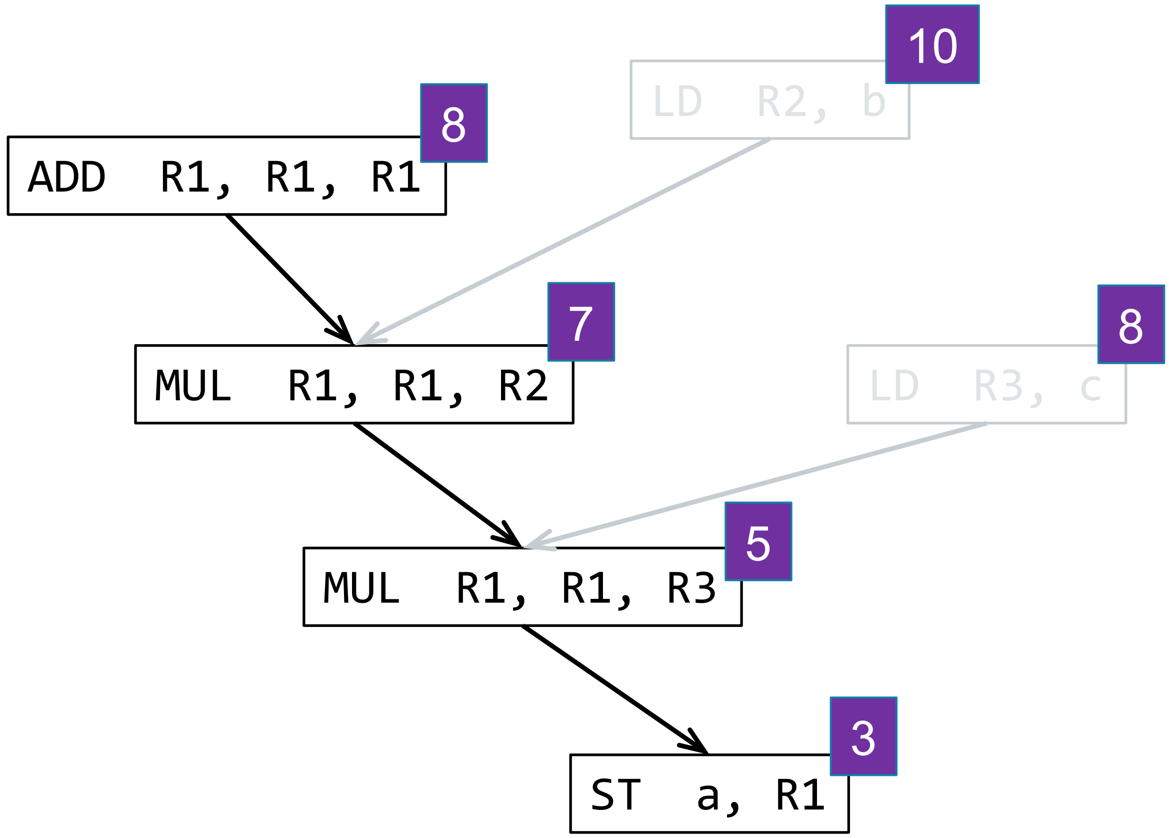
LD	R1,	a
LD	R2,	b
LD	R3,	c



List Scheduling

- Cycle = 4

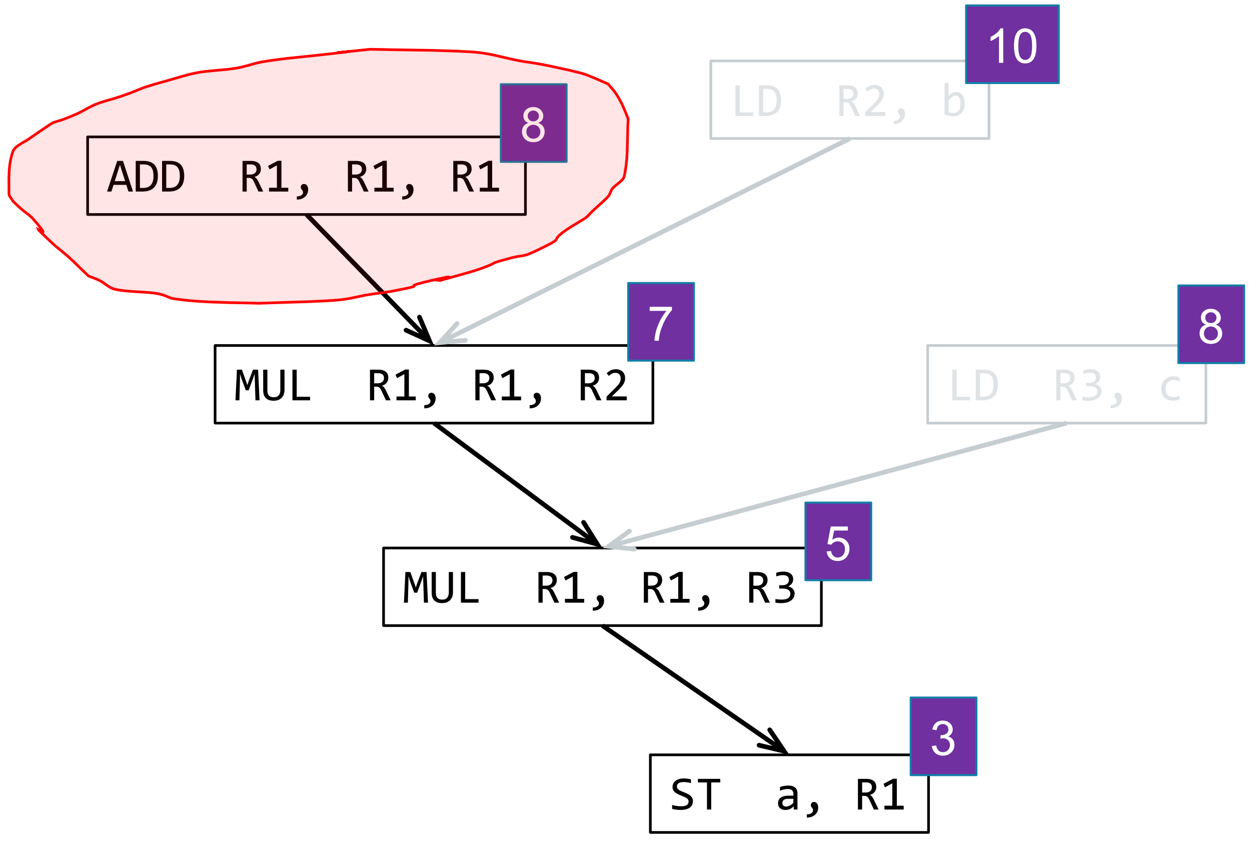
LD R1, a
LD R2, b
LD R3, c



List Scheduling

- Cycle = 4

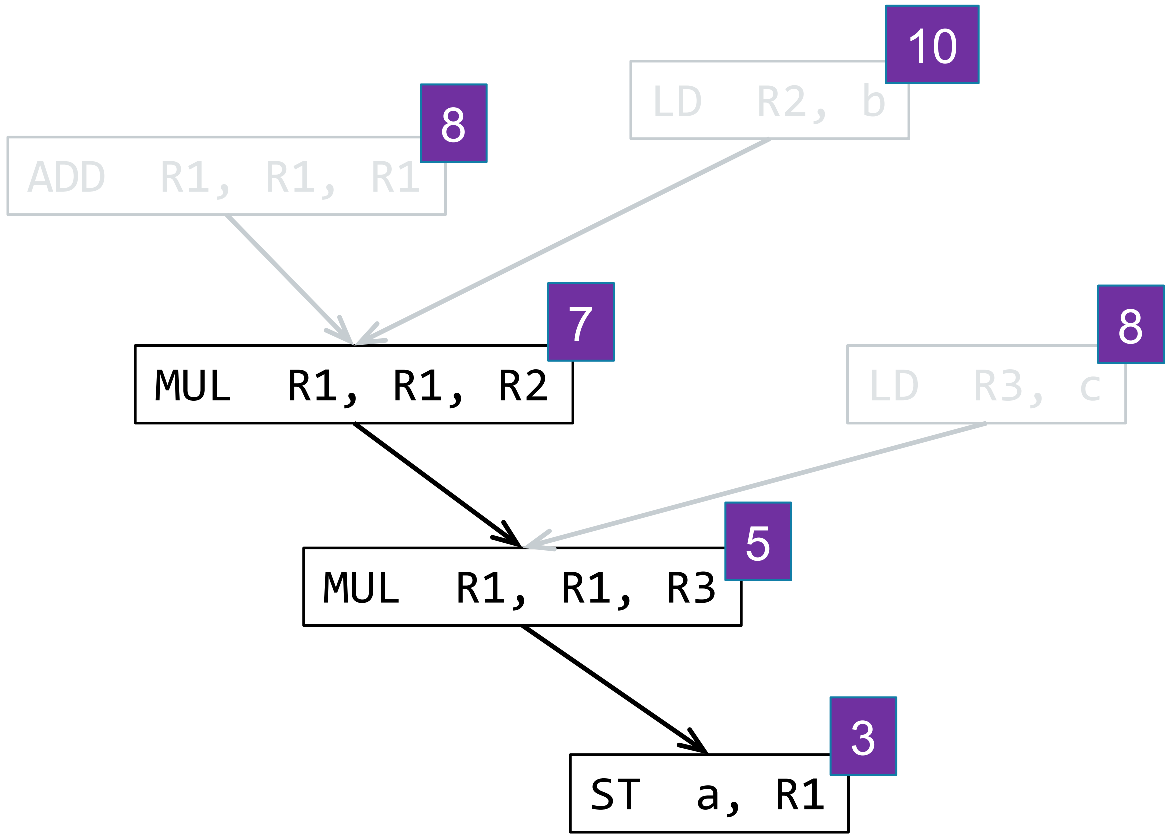
LD R1, a
LD R2, b
LD R3, c



List Scheduling

- Cycle = 4

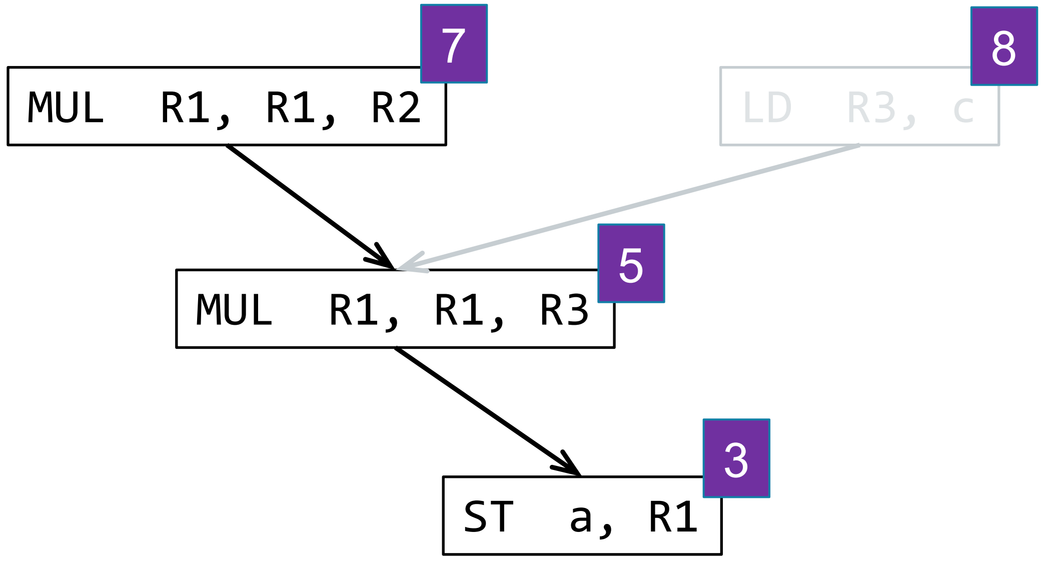
LD R1, a
LD R2, b
LD R3, c
ADD R1, R1, R1



List Scheduling

- Cycle = 5

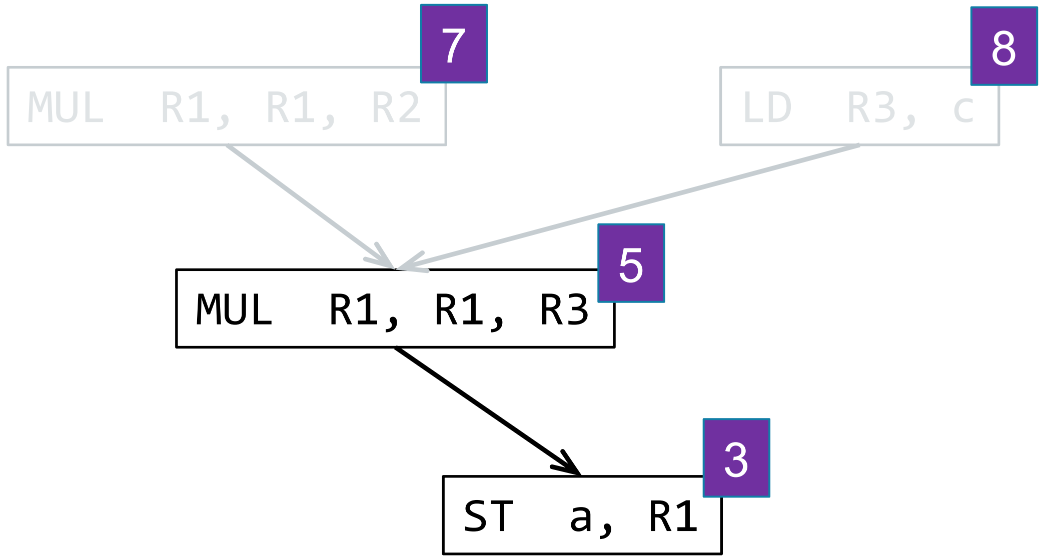
LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1



List Scheduling

- Cycle = 5

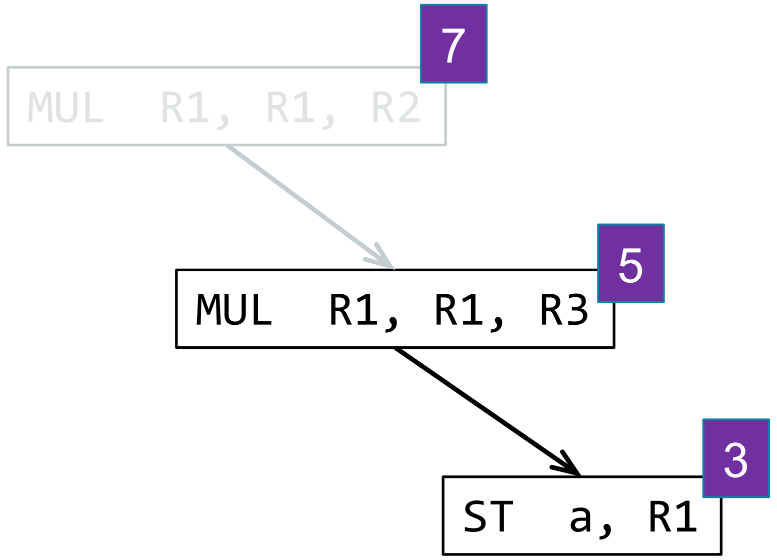
LD R1, a
LD R2, b
LD R3, c
ADD R1, R1, R1
MUL R1, R1, R2



List Scheduling

- Cycle = 6

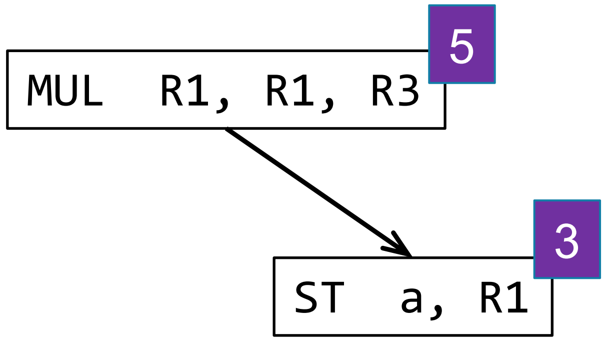
LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
NOP (no operation)			



List Scheduling

- Cycle = 7

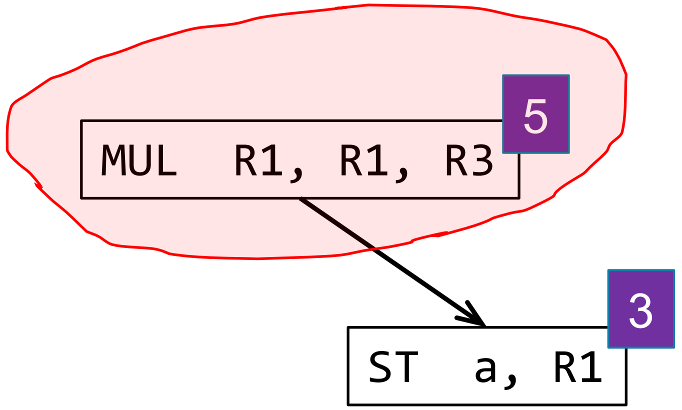
LD R1, a
LD R2, b
LD R3, c
ADD R1, R1, R1
MUL R1, R1, R2
NOP (no operation)



List Scheduling

- Cycle = 7

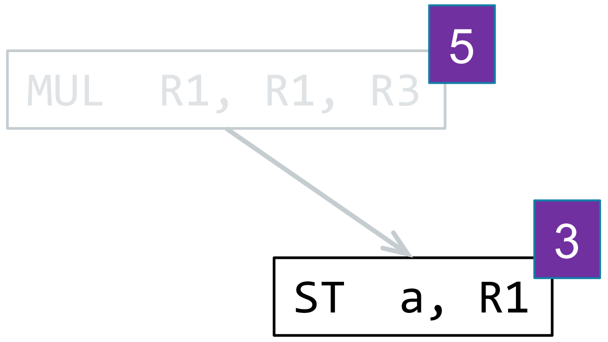
LD R1, a
LD R2, b
LD R3, c
ADD R1, R1, R1
MUL R1, R1, R2
NOP (no operation)



List Scheduling

- Cycle = 7, 8

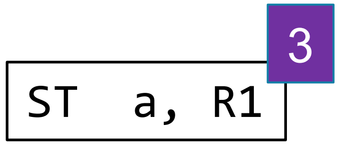
LD R1, a
LD R2, b
LD R3, c
ADD R1, R1, R1
MUL R1, R1, R2
NOP (no operation)
MUL R1, R1, R3
NOP (no operation)



List Scheduling

- Cycle = 9, 10, 11

LD R1, a
LD R2, b
LD R3, c
ADD R1, R1, R1
MUL R1, R1, R2
NOP (no operation)
MUL R1, R1, R3
NOP (no operation)
ST a, R1



List Scheduling

- a = 2 * a * b * c

$\xrightarrow{\text{List Scheduling}}$

LD	R1,	a	
ADD	R1,	R1,	R1
LD	R2,	b	
MUL	R1,	R1,	R2
LD	R2,	c	
MUL	R1,	R1,	R2
ST	a,	R1	

LD	R1,	a	
LD	R2,	b	
LD	R3,	c	
ADD	R1,	R1,	R1
MUL	R1,	R1,	R2
MUL	R1,	R1,	R3
ST	a,	R1	

Summary of List Scheduling

- Check each clock cycle
- Schedule instructions when they are ready
- When multiple instructions can be scheduled, check their **priority**
  - The longest latency path (max depth)
  - Using the most resources
  - …

PART III: Global Code Scheduling

Recap: Dominance

- A **dominates** B if and only if …?
- A **strictly dominates** B if and only if …?

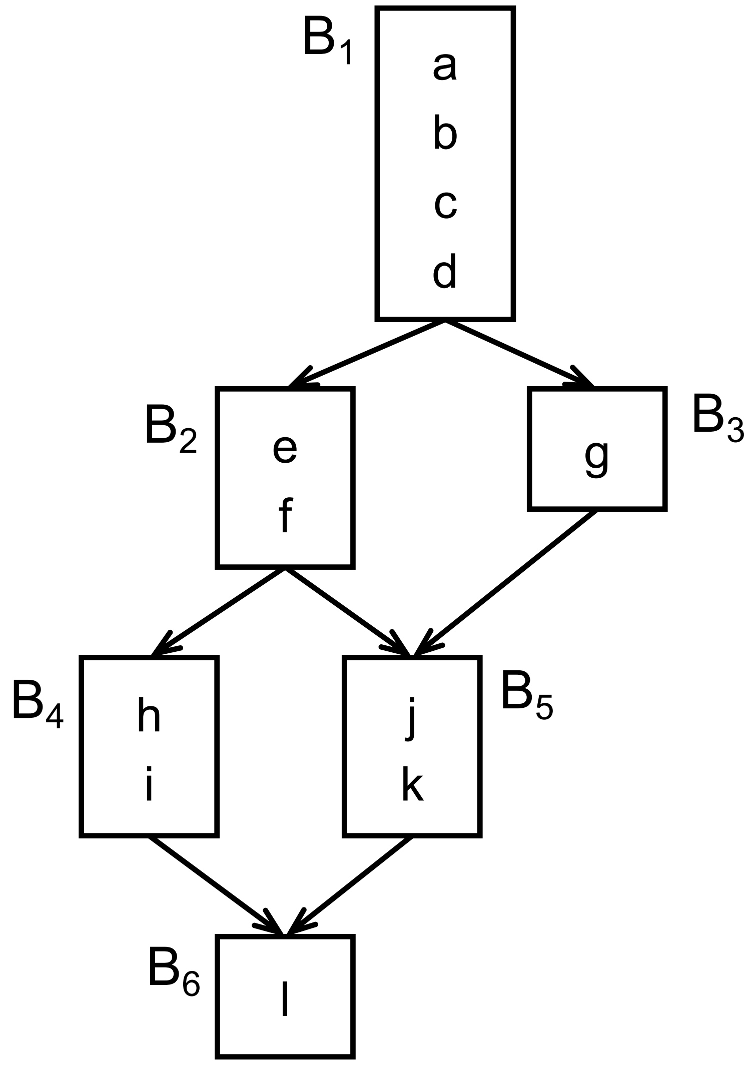
Recap: Dominance

- A **dominates** B if and only if …?
- A **strictly dominates** B if and only if …?

- A **post-dominates** B if and only if …?
- A **strictly post-dominates** B if and only if …?

Extended Basic Block (EBB)

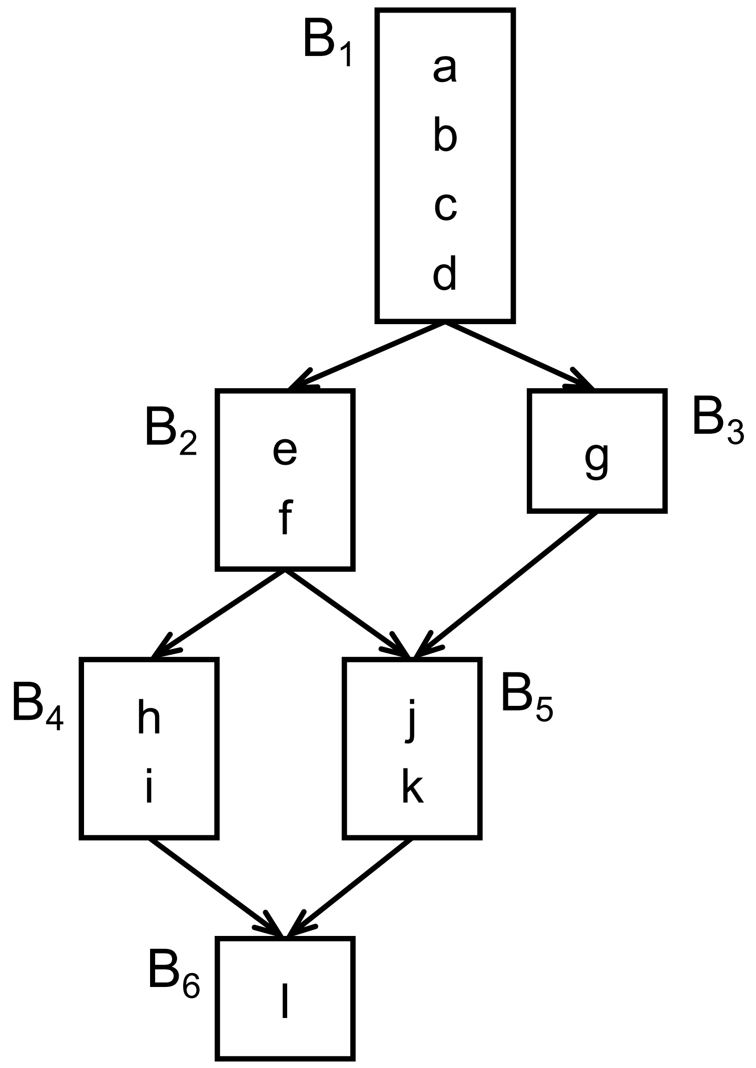
- EBB is a maximal set of blocks such that
- (1) It has a single entry;
- (2) All blocks except for the entry has only one predecessor.



Extended Basic Block (EBB)

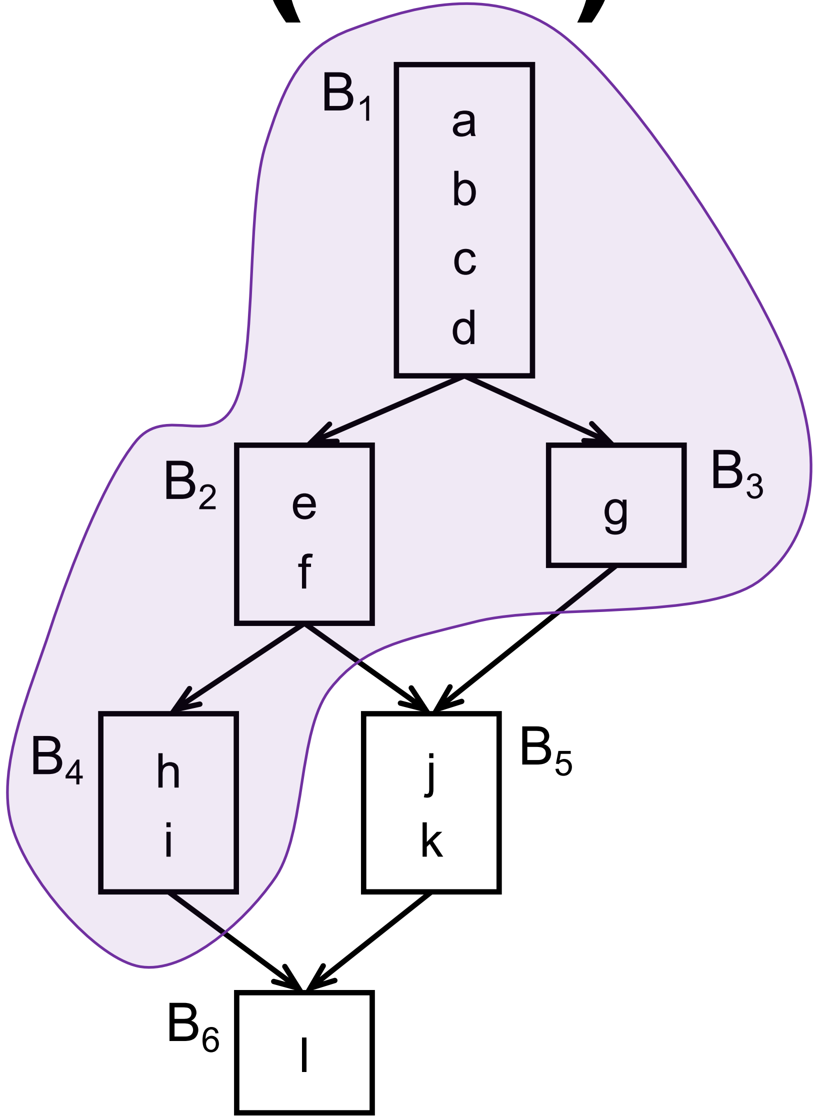
- EBB is a maximal set of blocks such that
- (1) It has a single entry;
- (2) All blocks except for the entry has only one predecessor.

Try!



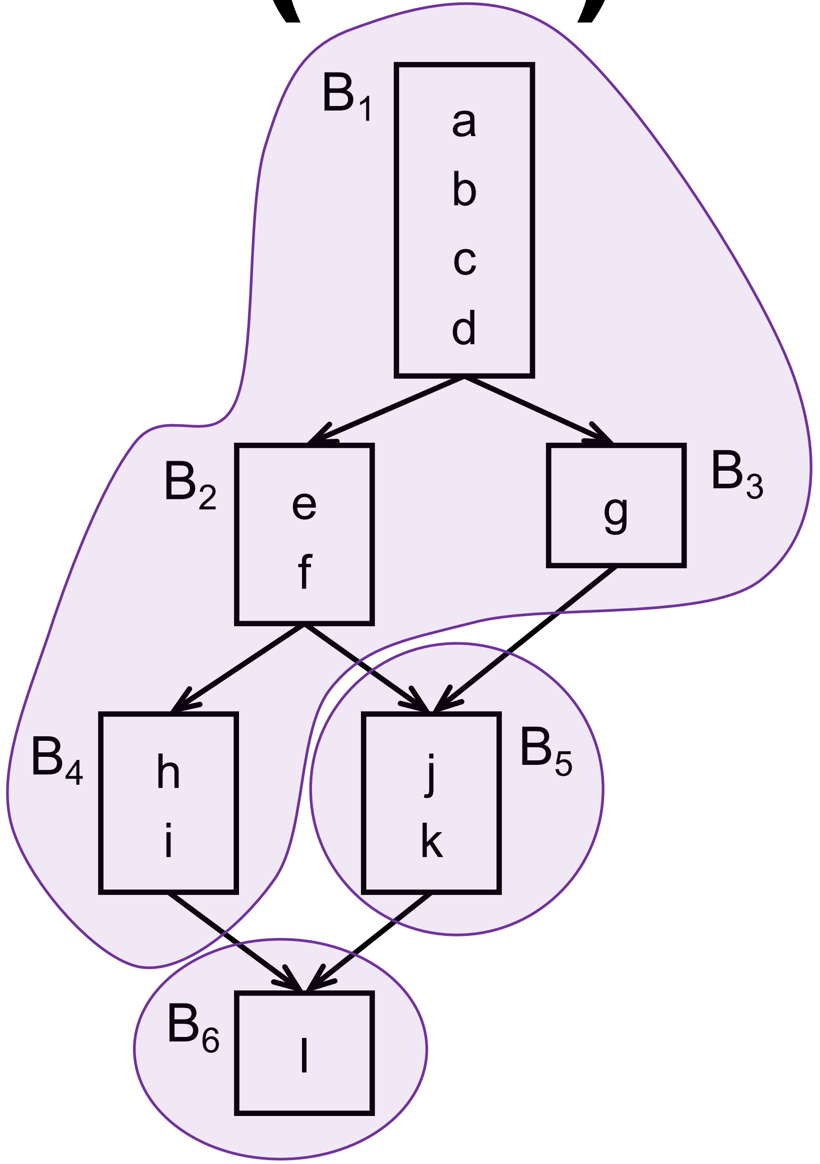
Extended Basic Block (EBB)

- EBB is a maximal set of blocks such that
- (1) It has a single entry;
- (2) All blocks except for the entry has only one predecessor.



Extended Basic Block (EBB)

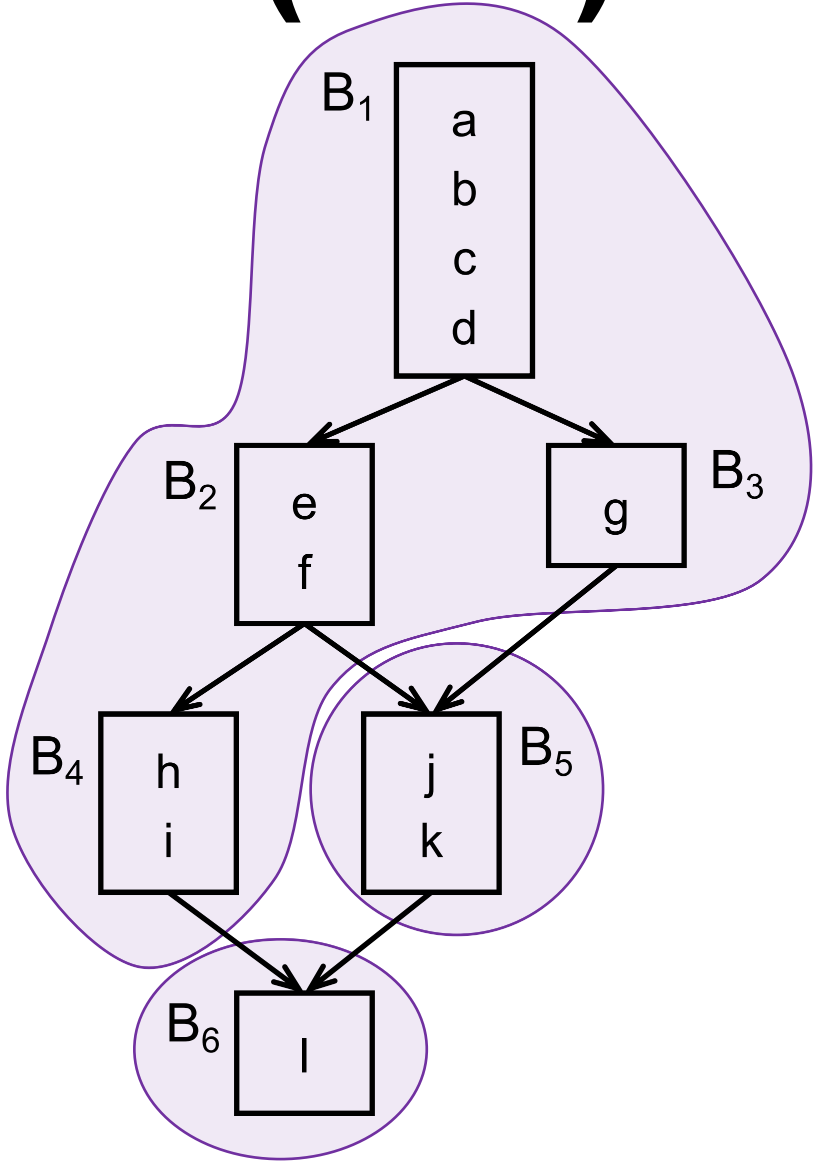
- EBB is a maximal set of blocks such that
- (1) It has a single entry;
- (2) All blocks except for the entry has only one predecessor.



Extended Basic Block (EBB)

- EBB is a maximal set of blocks such that
- (1) It has a single entry;
- (2) All blocks except for the entry has only one predecessor.

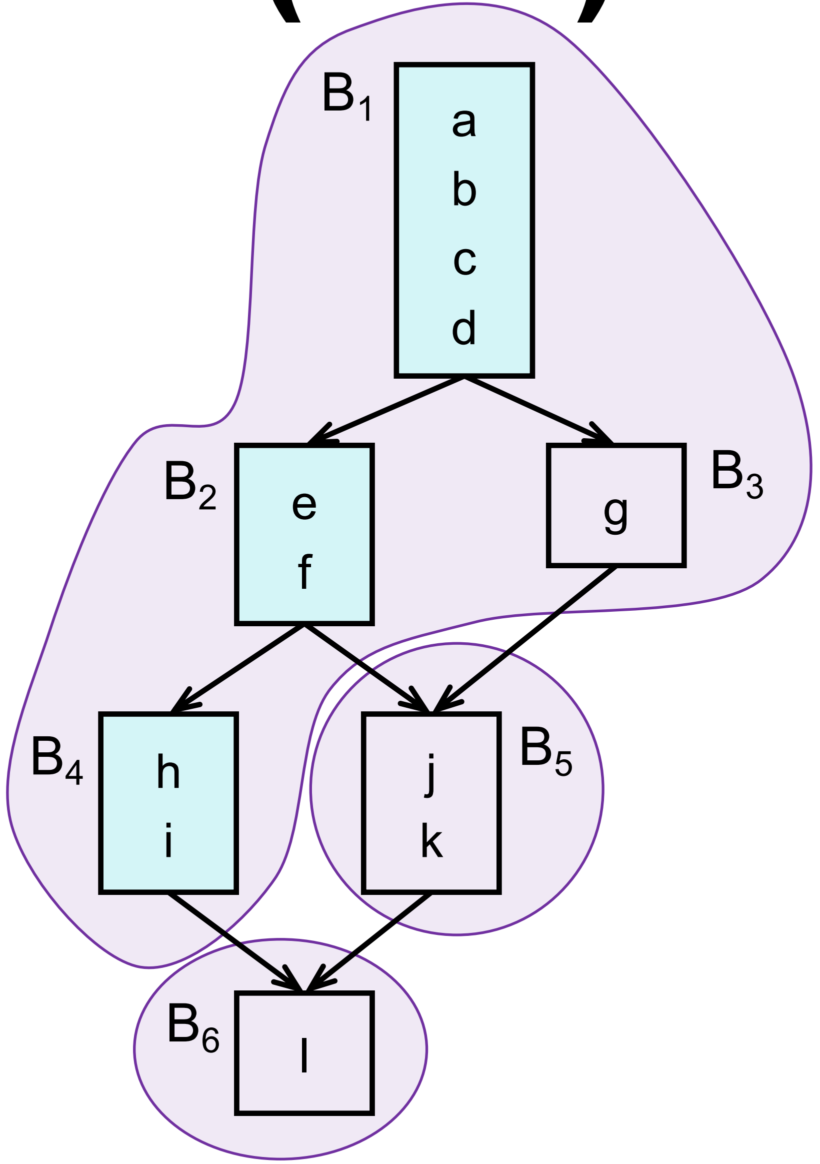
- Three EBBs in the example
- **Four EBB paths**



Extended Basic Block (EBB)

- EBB is a maximal set of blocks such that
- (1) It has a single entry;
- (2) All blocks except for the entry has only one predecessor.

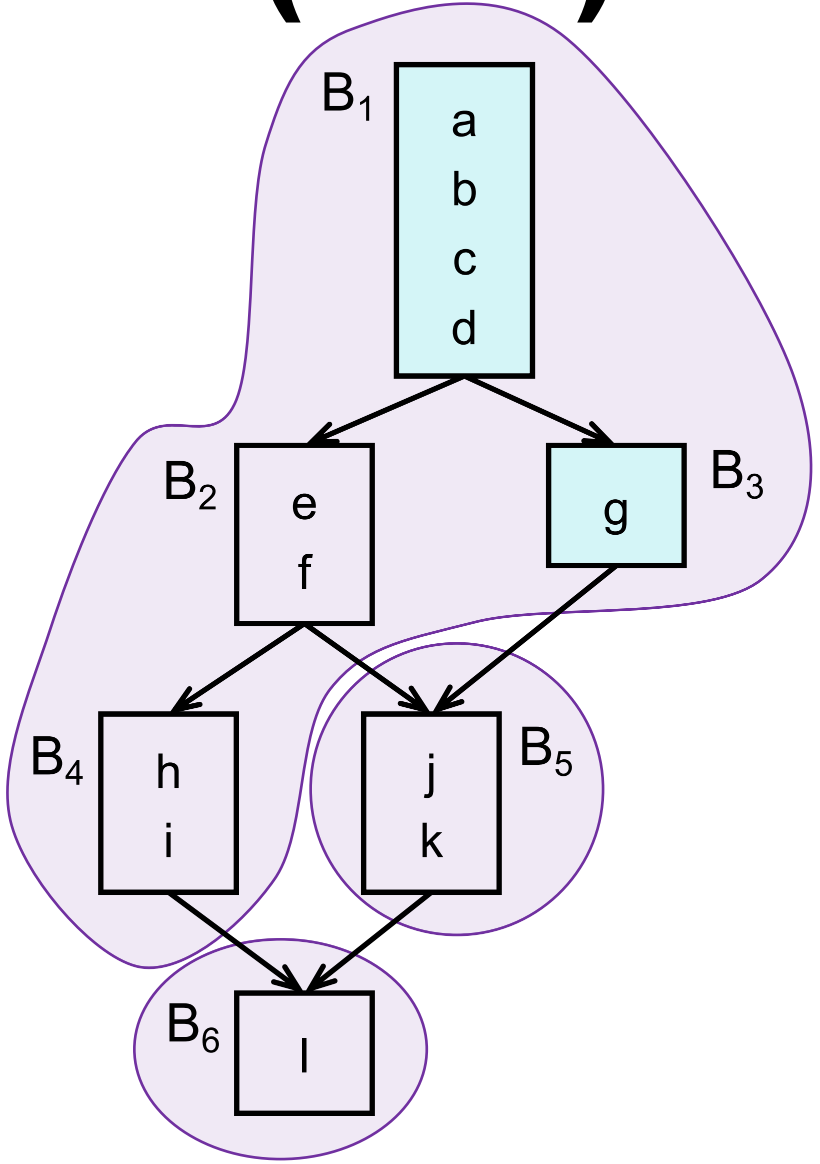
- Three EBBs in the example
- **Four EBB paths**



Extended Basic Block (EBB)

- EBB is a maximal set of blocks such that
- (1) It has a single entry;
- (2) All blocks except for the entry has only one predecessor.

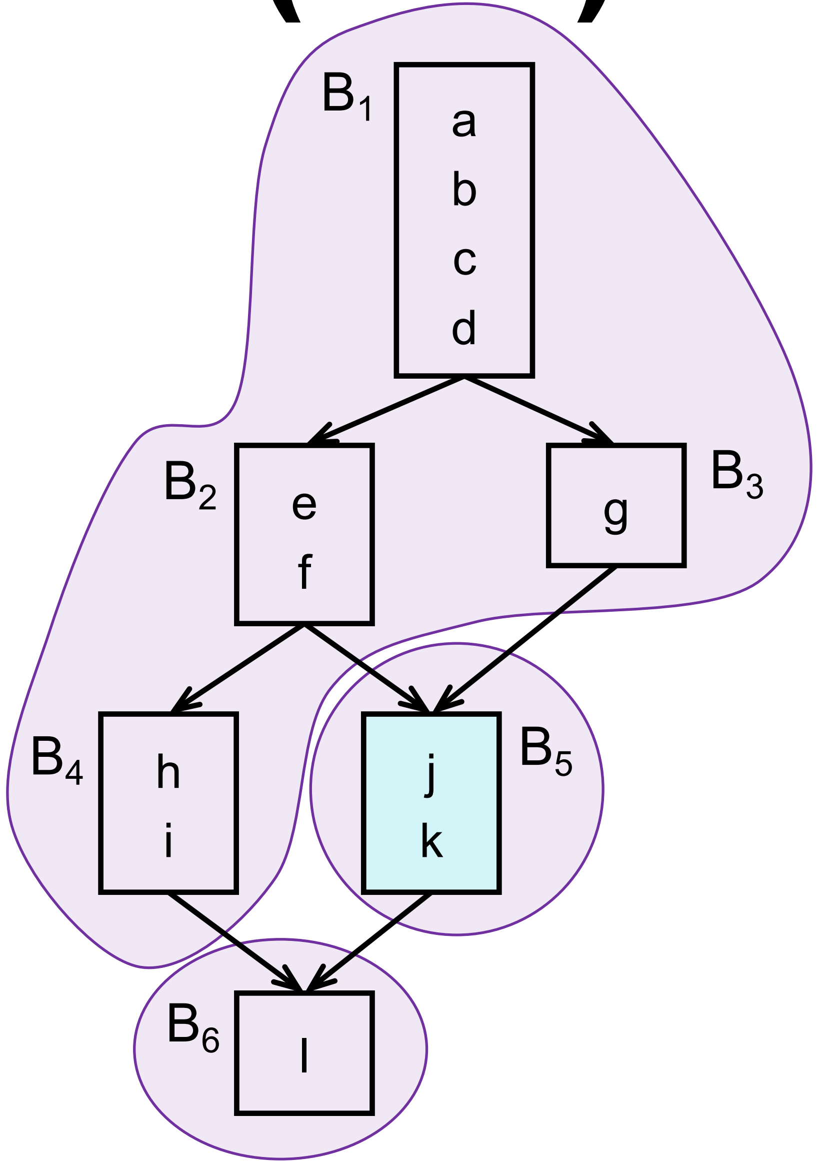
- Three EBBs in the example
- **Four EBB paths**



Extended Basic Block (EBB)

- EBB is a maximal set of blocks such that
- (1) It has a single entry;
- (2) All blocks except for the entry has only one predecessor.

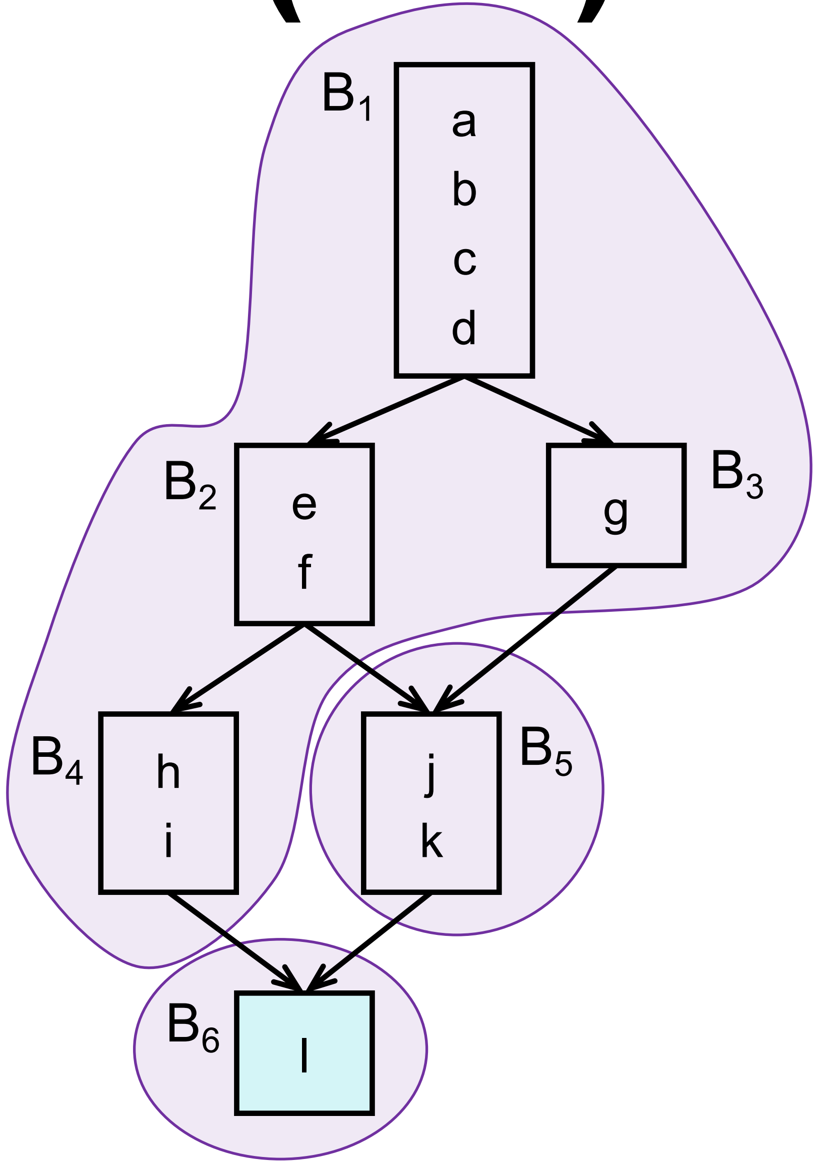
- Three EBBs in the example
- **Four EBB paths**



Extended Basic Block (EBB)

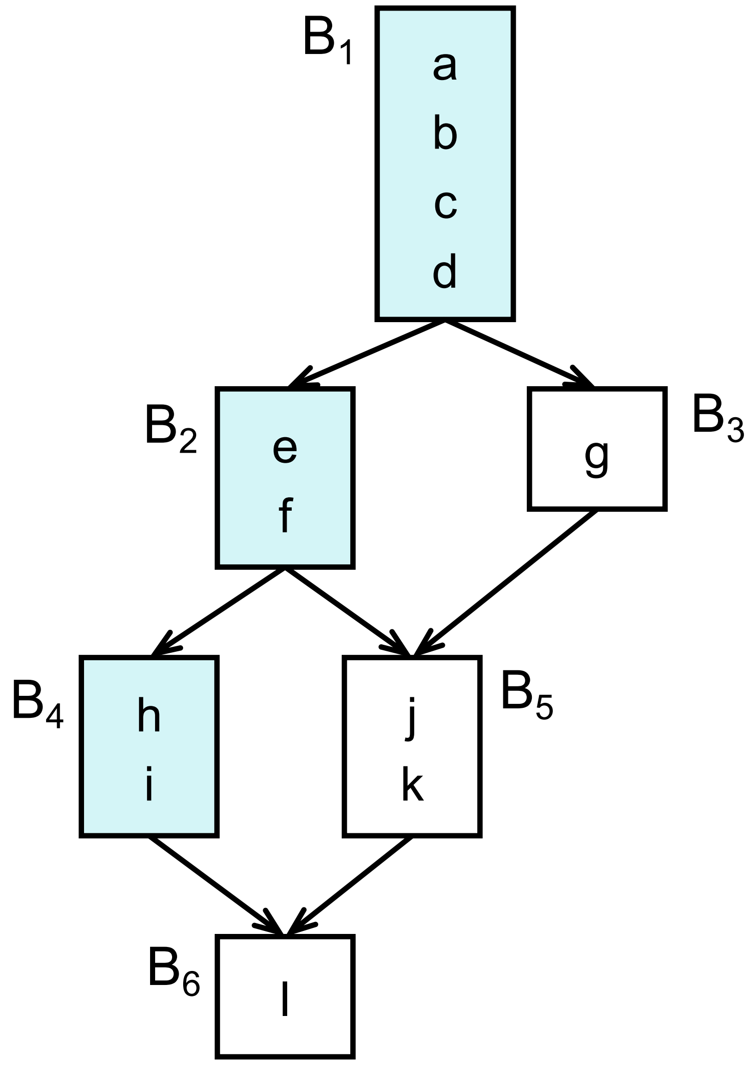
- EBB is a maximal set of blocks such that
- (1) It has a single entry;
- (2) All blocks except for the entry has only one predecessor.

- Three EBBs in the example
- **Four EBB paths**



Global Scheduling

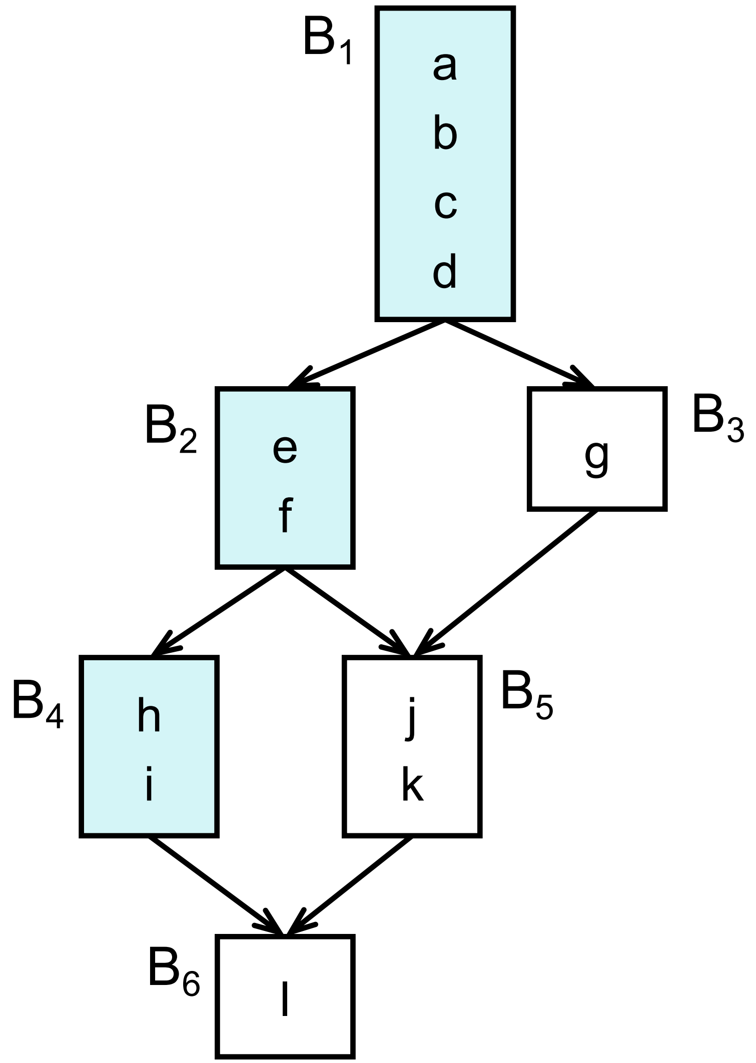
- Schedule each EBB path like a basic block, e.g., via list scheduling



Global Scheduling

- Schedule each EBB path like a basic block, e.g., via list scheduling

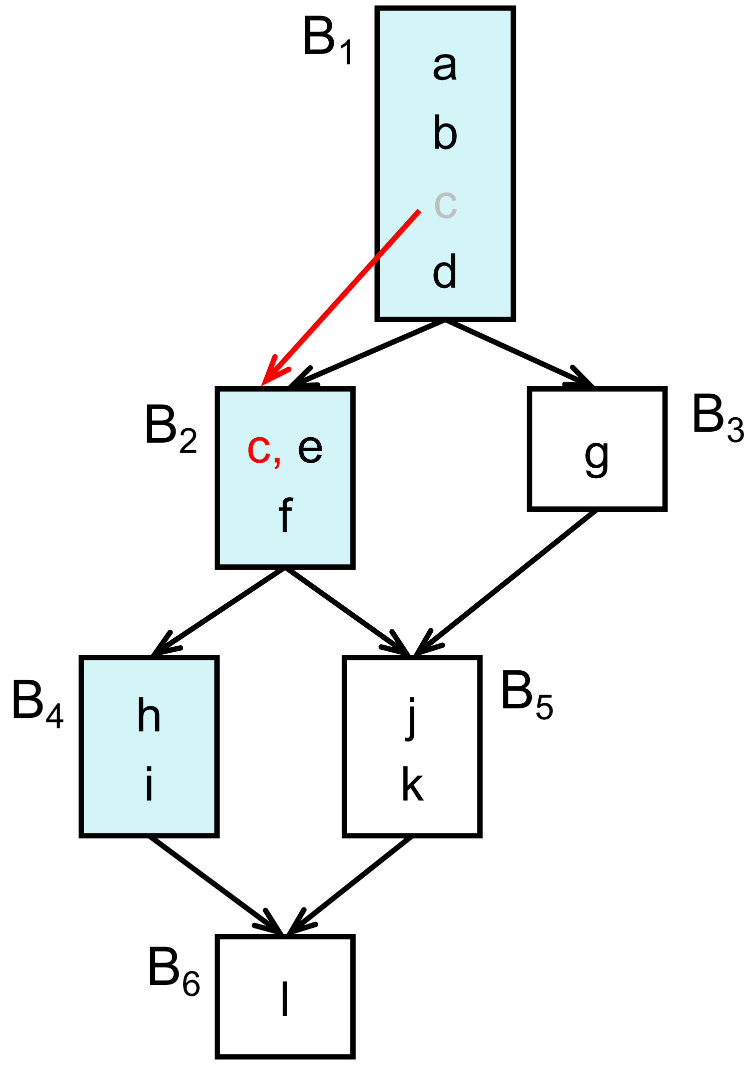
- Scheduling may change the order of instructions



Global Scheduling

- Schedule each EBB path like a basic block, e.g., via list scheduling

- Scheduling may change the order of instructions
  - **Downward code motion**
  - **Upward code motion**

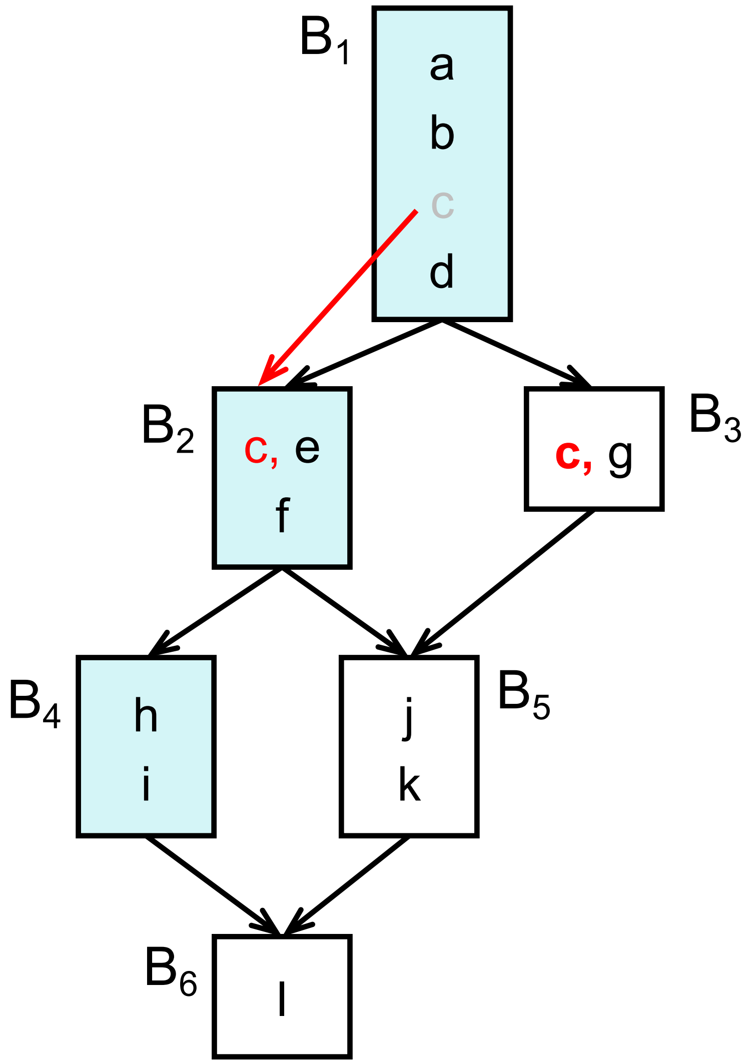


Downward Code Motion

- Schedule each EBB path like a basic block, e.g., via list scheduling

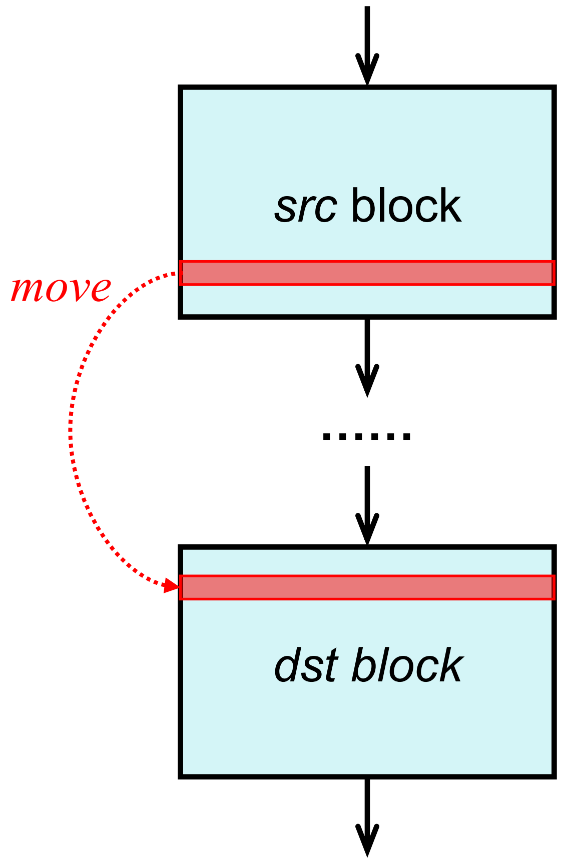
- Scheduling may change the order of instructions

- Extra code may need to be inserted



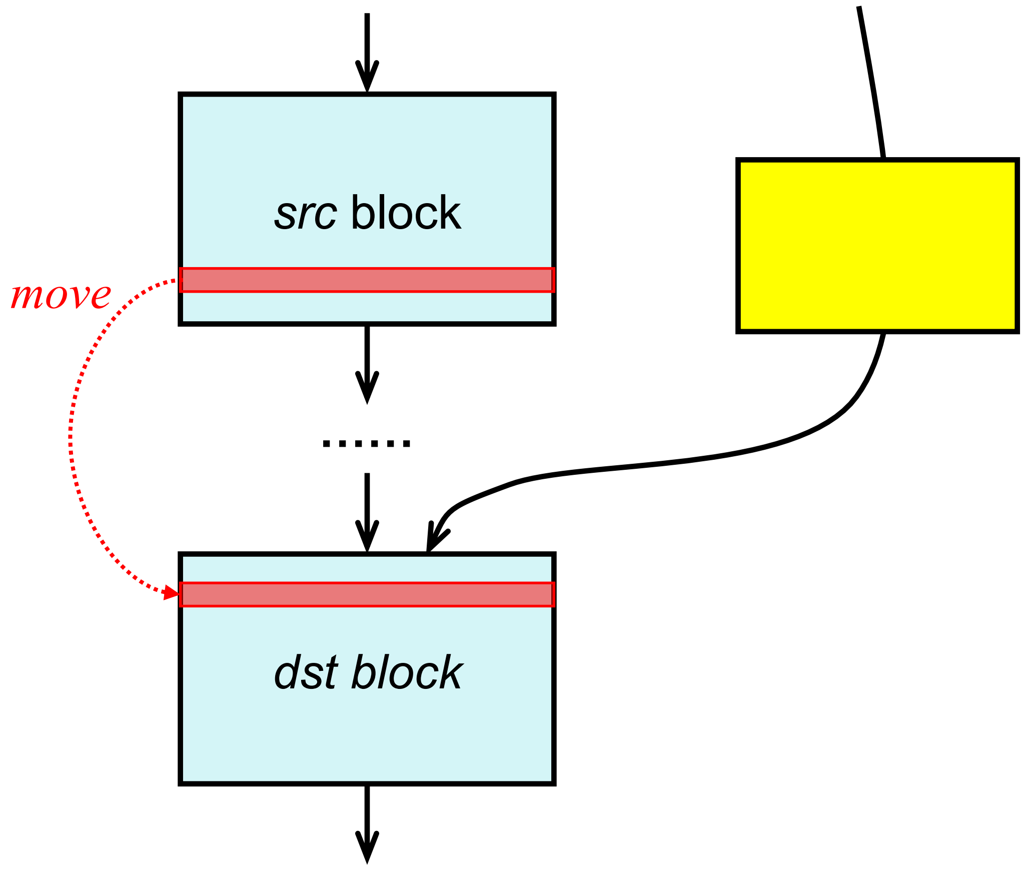
Downward Code Motion

- I. *src* does not dominate *dst*



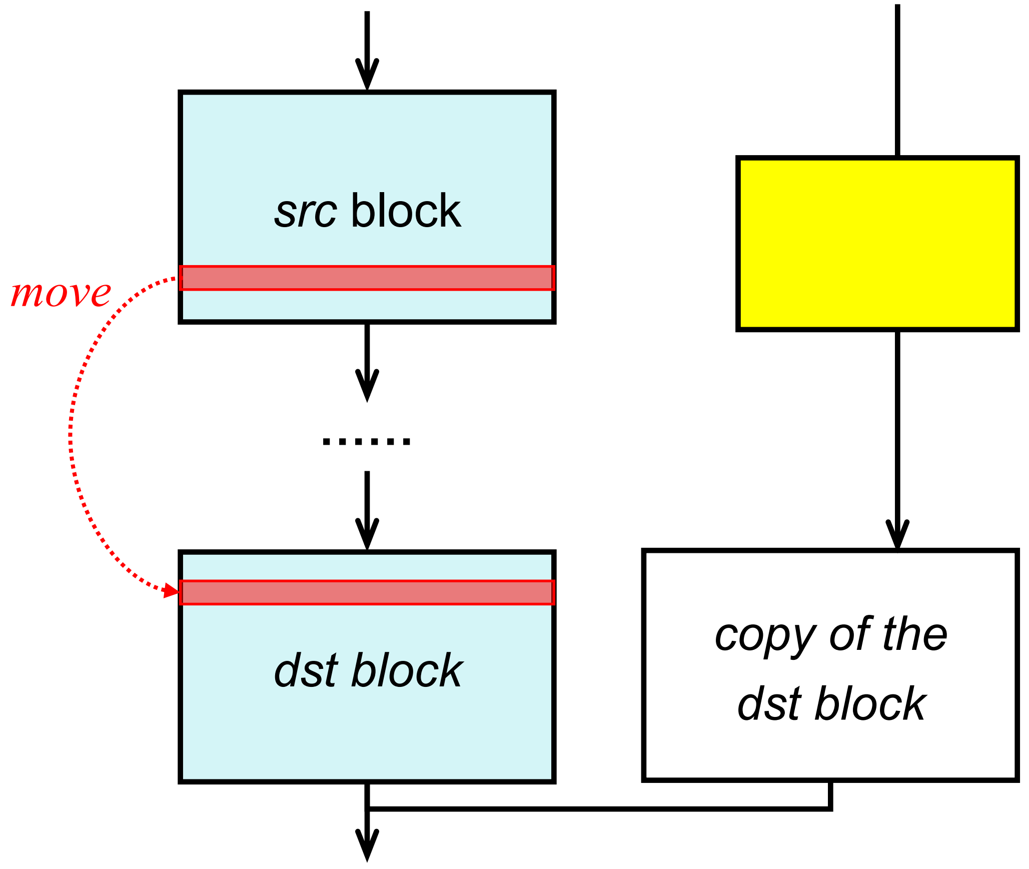
Downward Code Motion

- I. *src* does not dominate *dst*



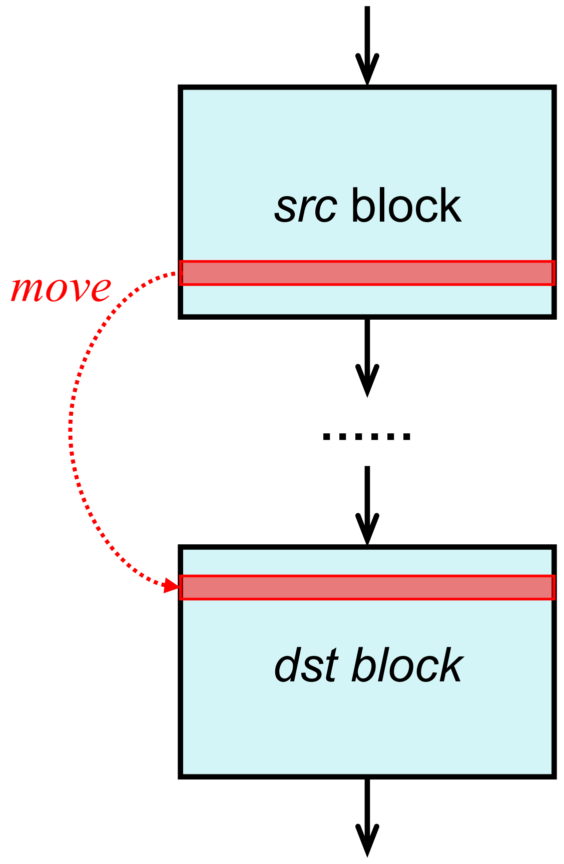
Downward Code Motion

- I. *src* does not dominate *dst*



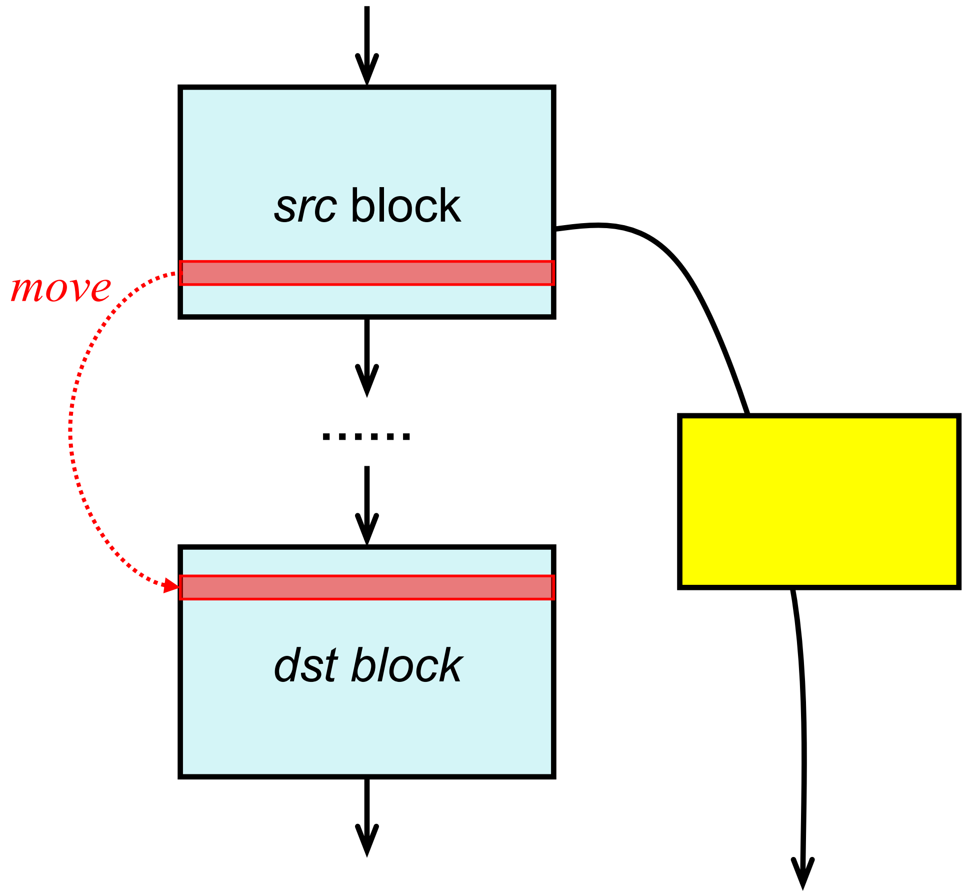
Downward Code Motion

- II. *dst* does not post-dominate *src*



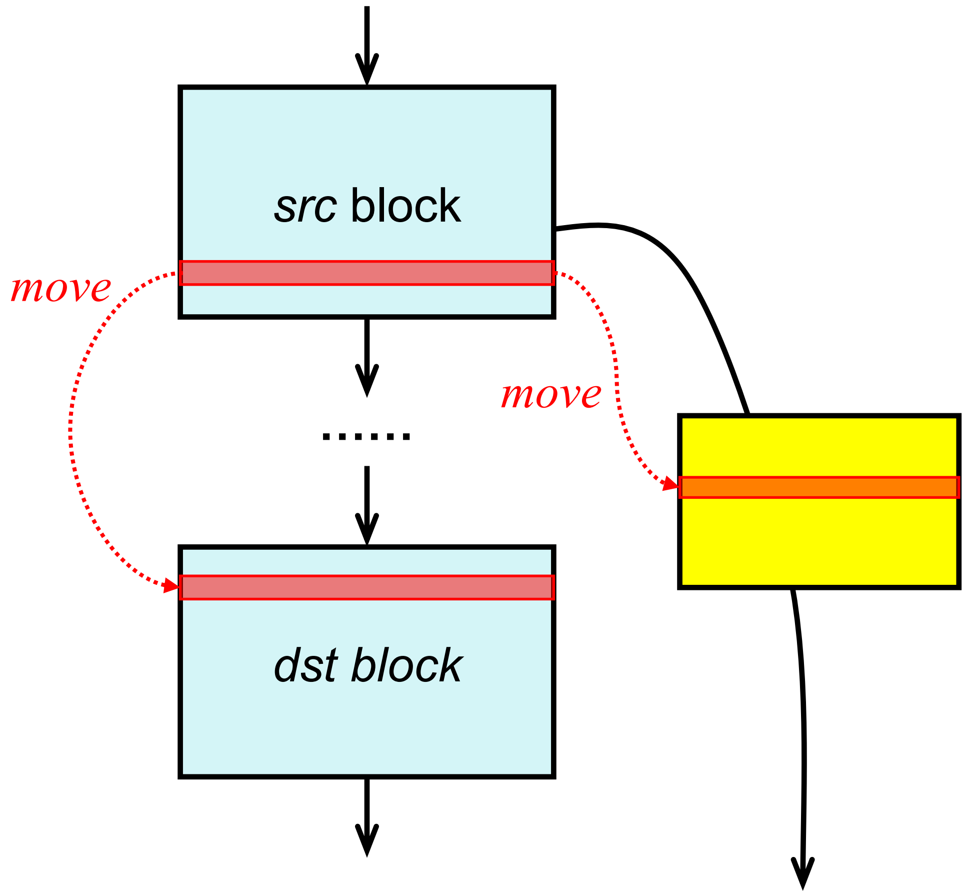
Downward Code Motion

- II. *dst* does not post-dominate *src*



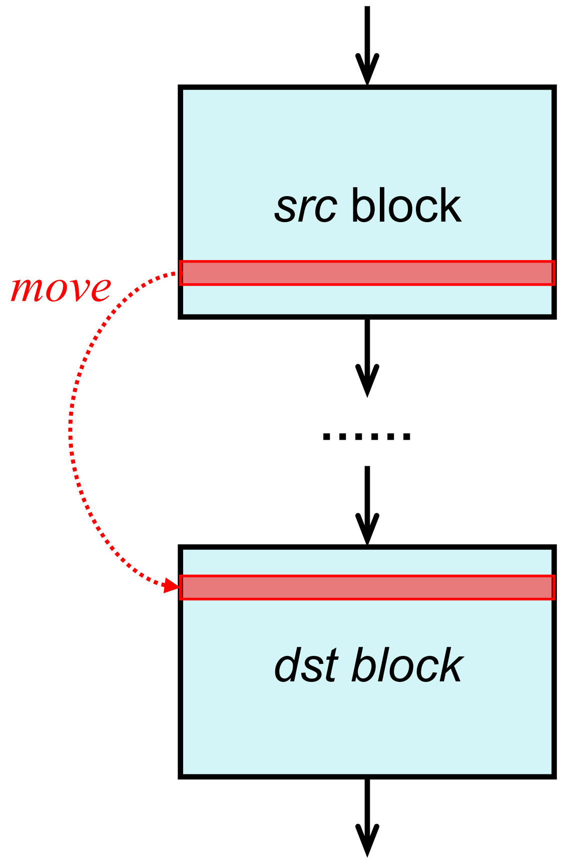
Downward Code Motion

- II. dst does not post-dominate src



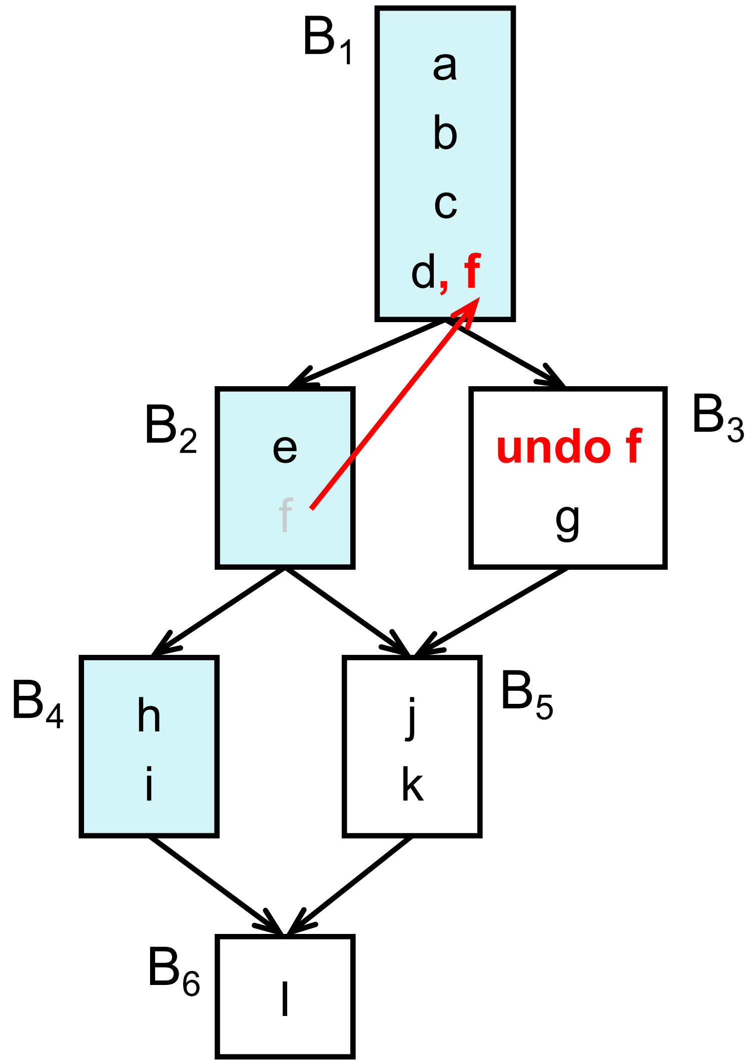
Downward Code Motion

- III. *src* **dom** *dst* & *dst* **post-dom** *src*



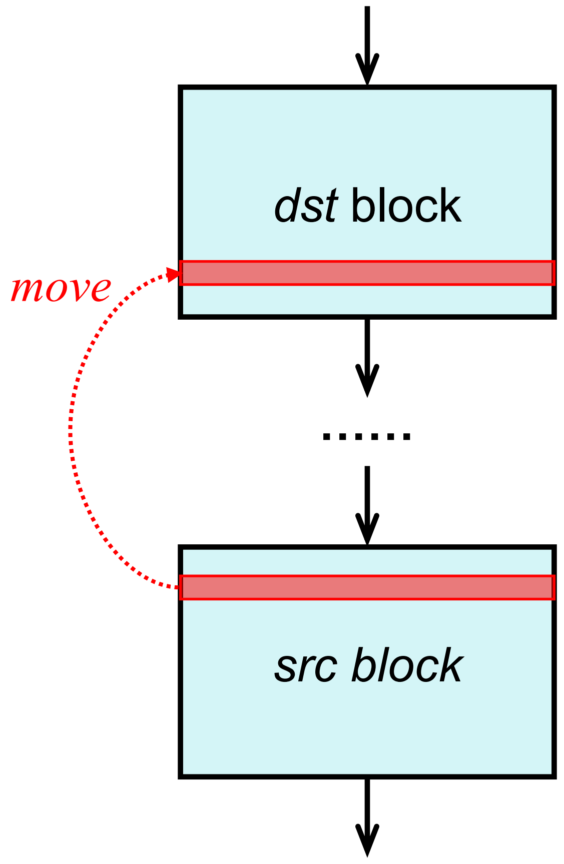
Upward Code Motion

- Schedule an EBB path like a basic block, e.g., via list scheduling.
- Scheduling may change the order of instructions
- Extra code may need to be inserted



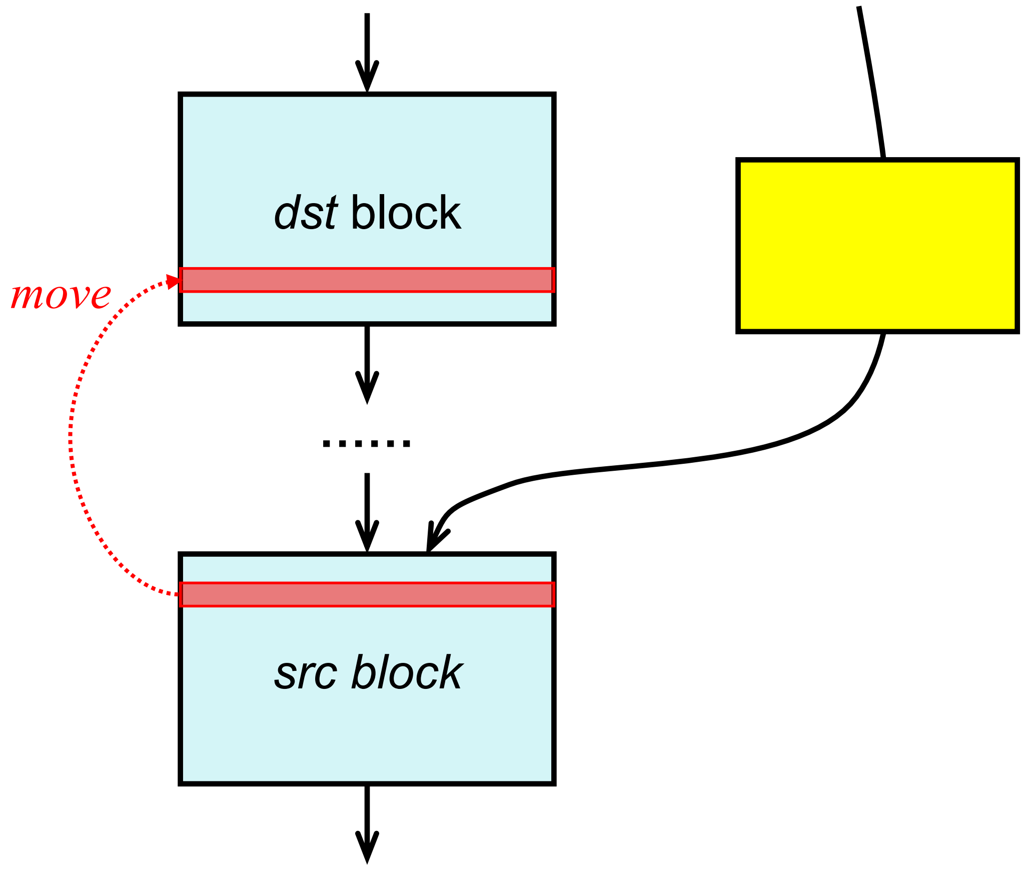
Upward Code Motion

- I. *dst* does not dominate *src*



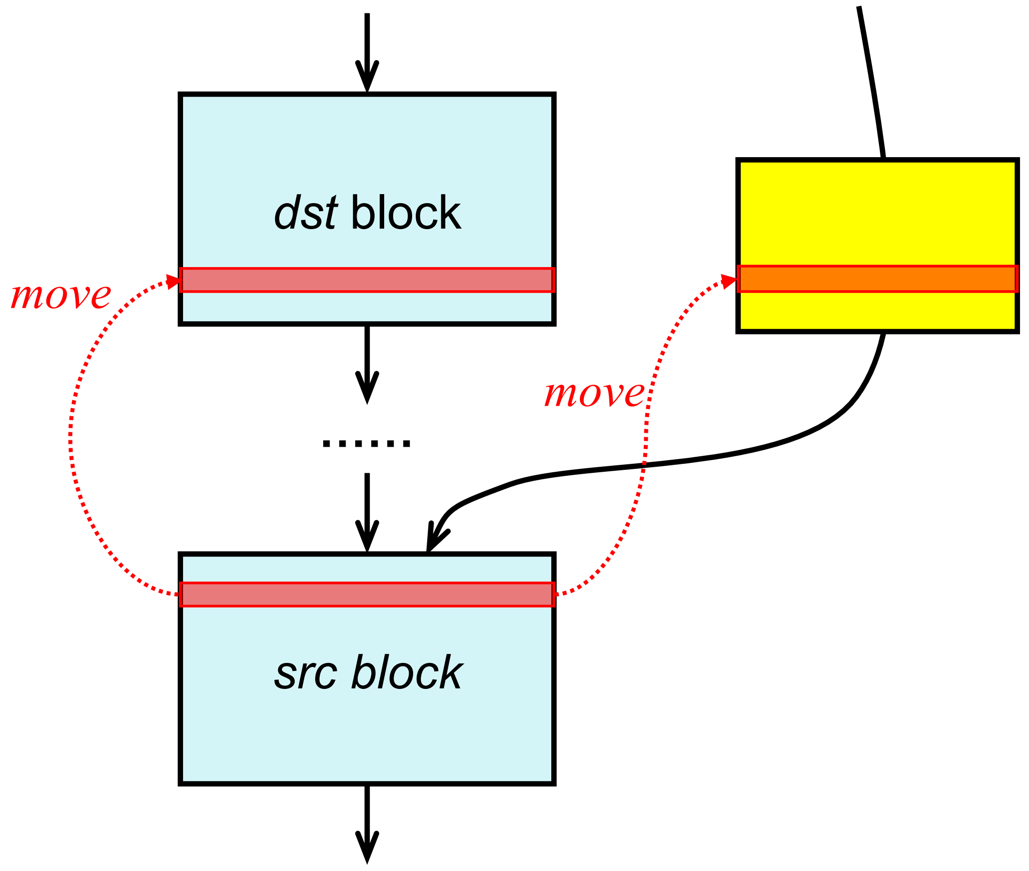
Upward Code Motion

- I. *dst* does not dominate *src*



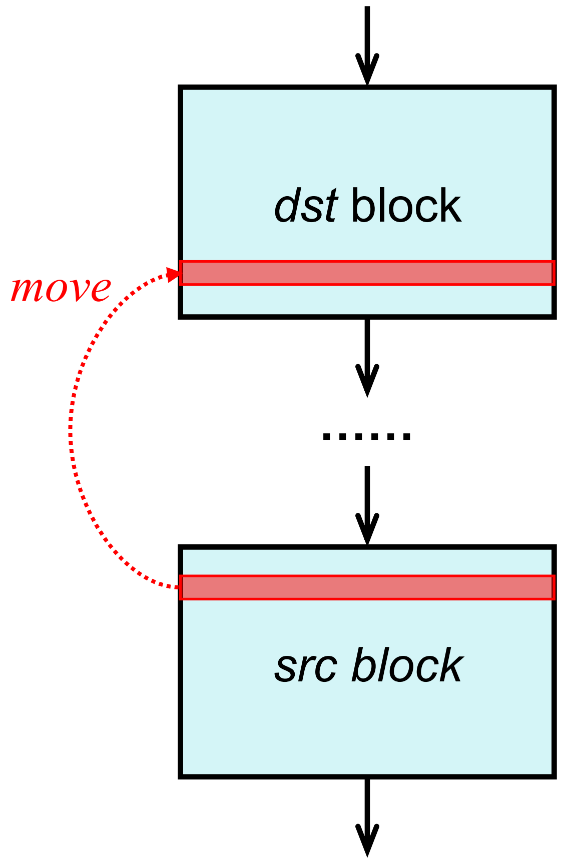
Upward Code Motion

- I. *dst* does not dominate *src*



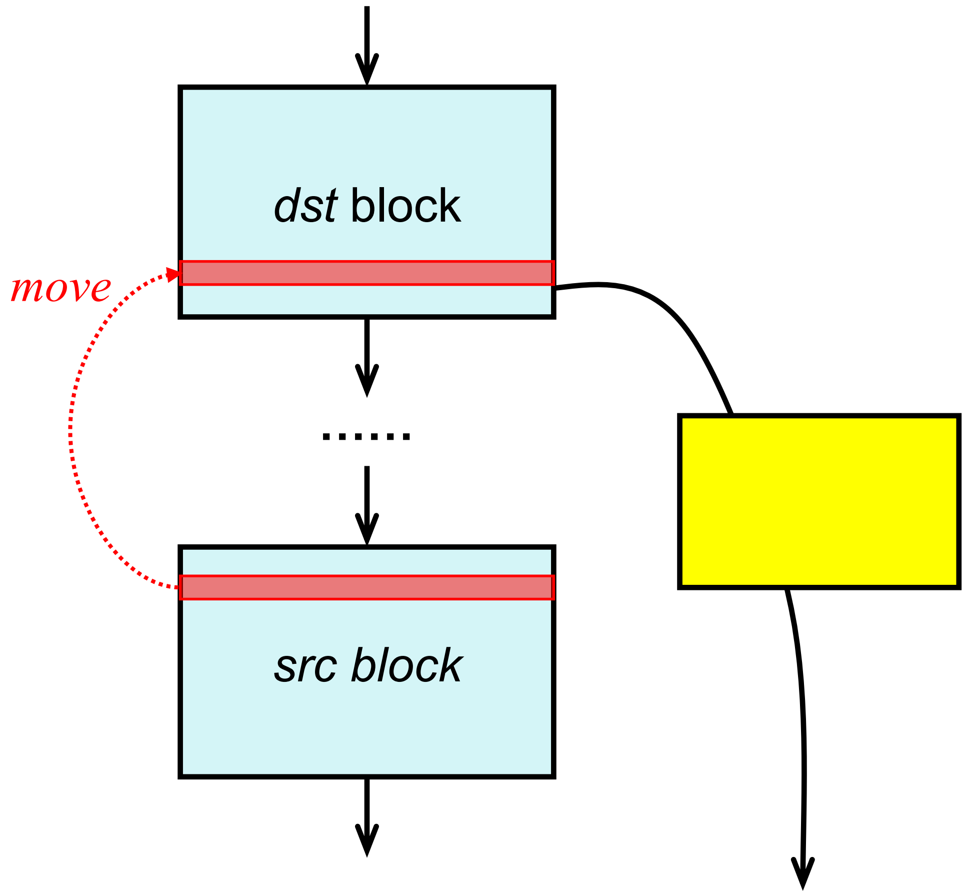
Upward Code Motion

- II. *src* does not post-dominate *dst*



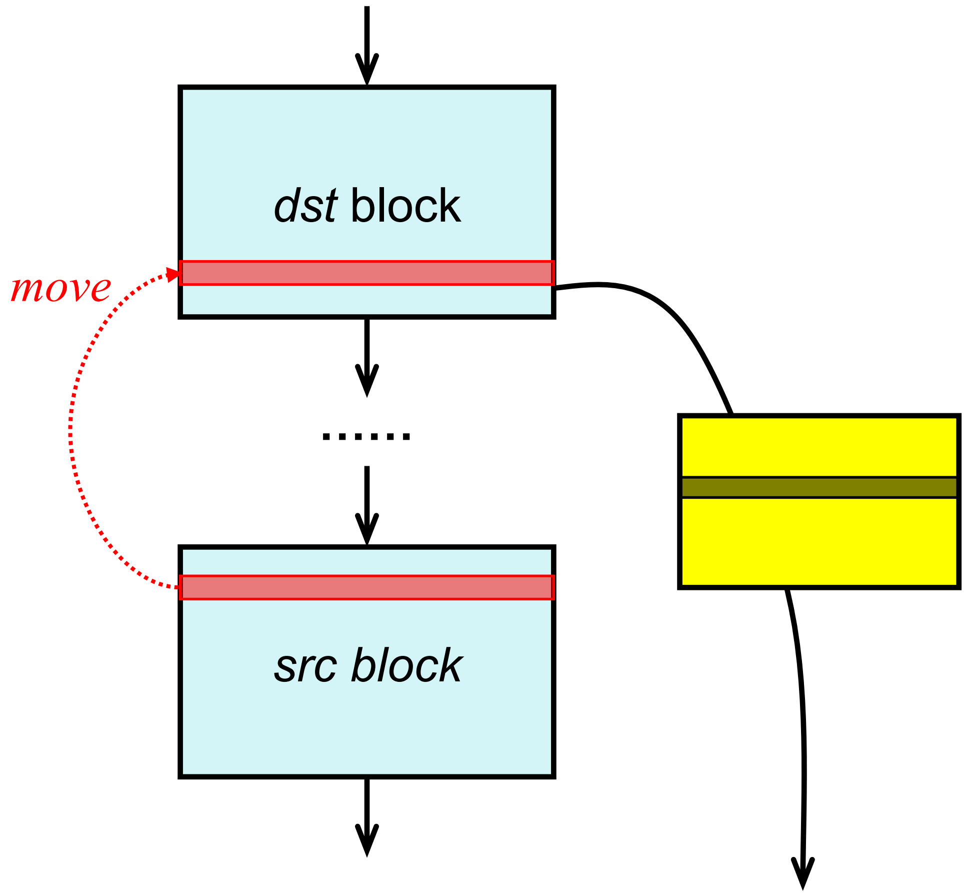
Upward Code Motion

- II. *src* does not post-dominate *dst*



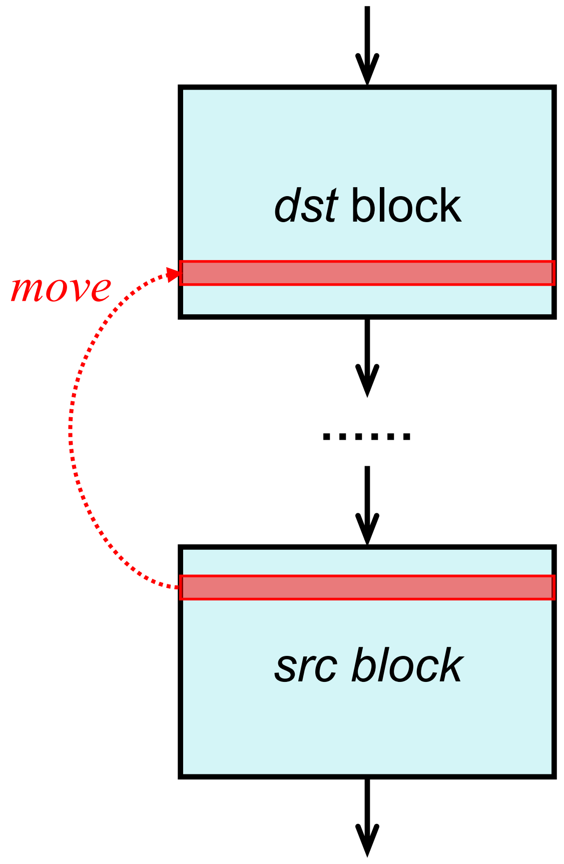
Upward Code Motion

- II. *src* does not post-dominate *dst*



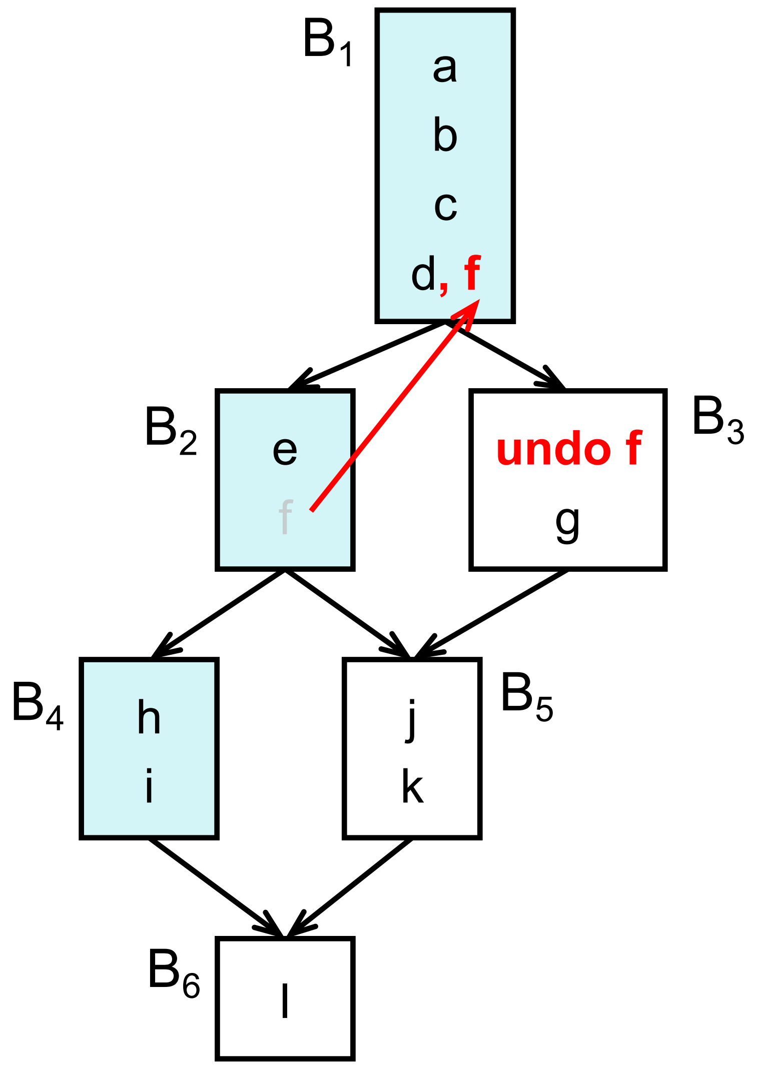
Upward Code Motion

- III. *dst* **dom** *src* & *src* **post-dom** *dst*



Global Scheduling

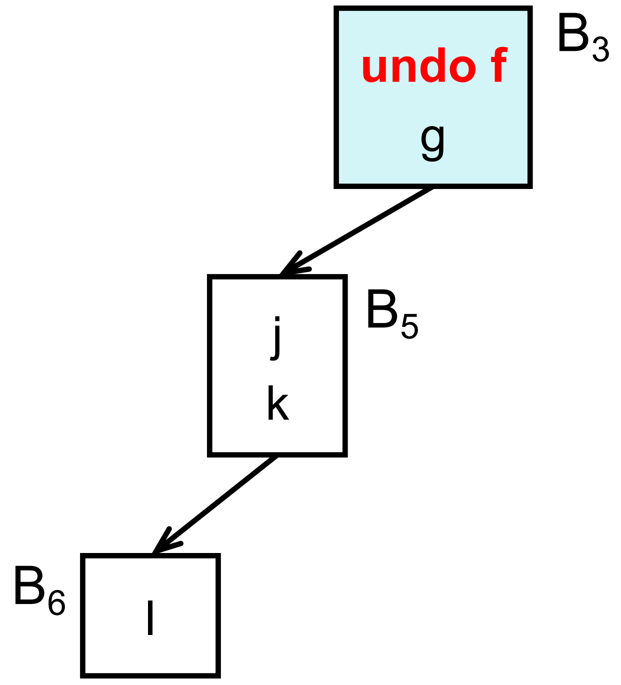
- After scheduling an EBB path, remove it from the graph
- Then schedule the next



Global Scheduling

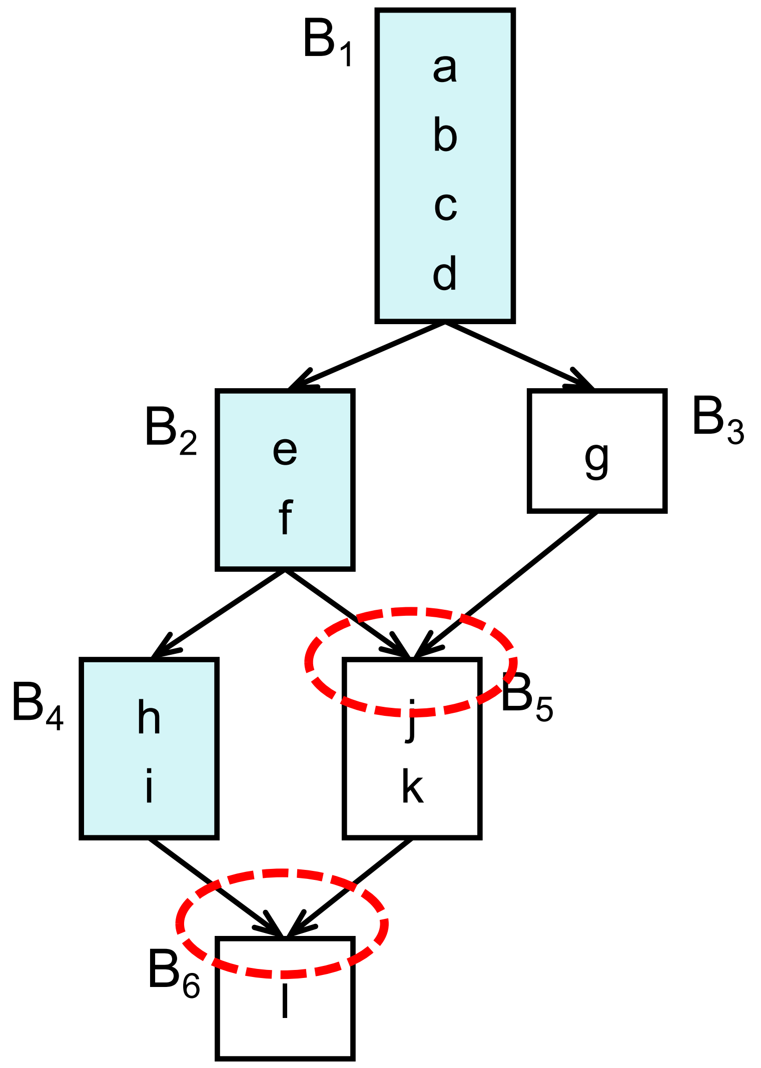
- After scheduling an EBB path, remove it from the graph

- Then schedule the next, until all EBB paths are scheduled



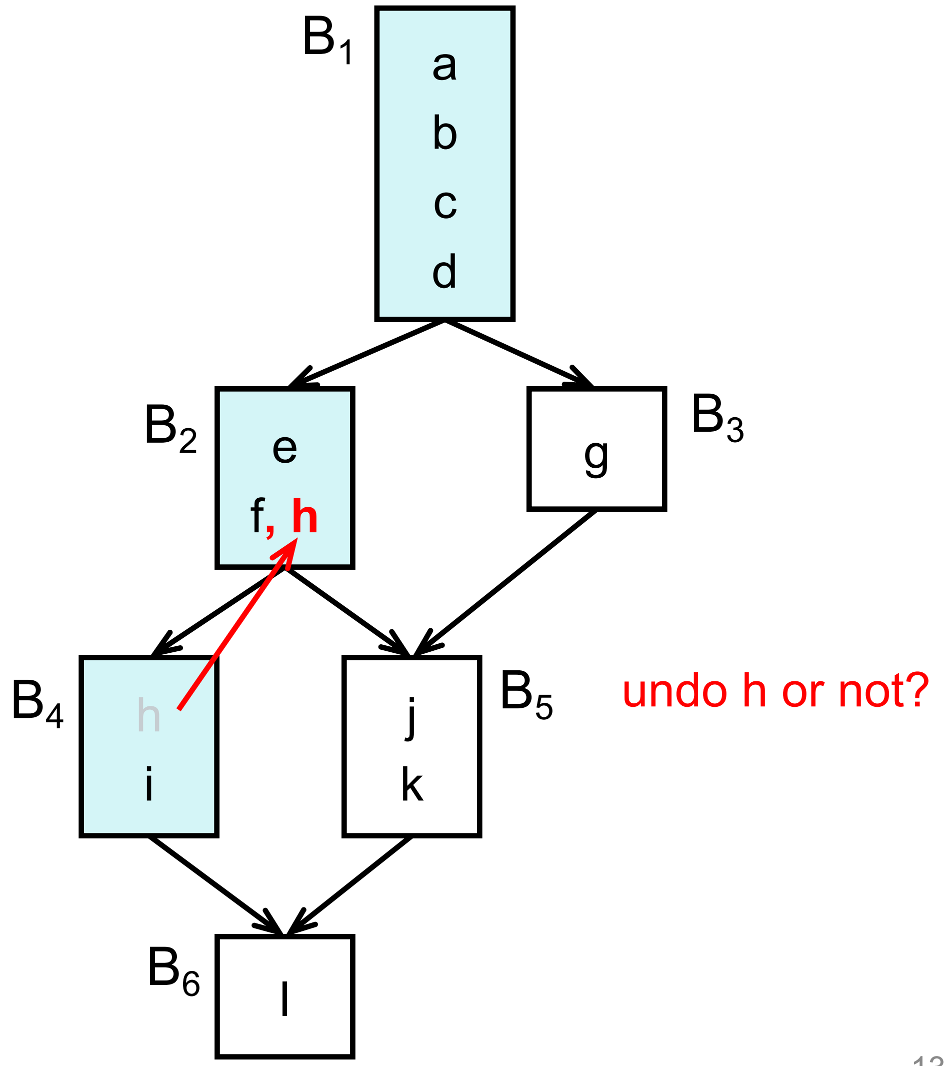
Context Problem at Join Points

- Join points



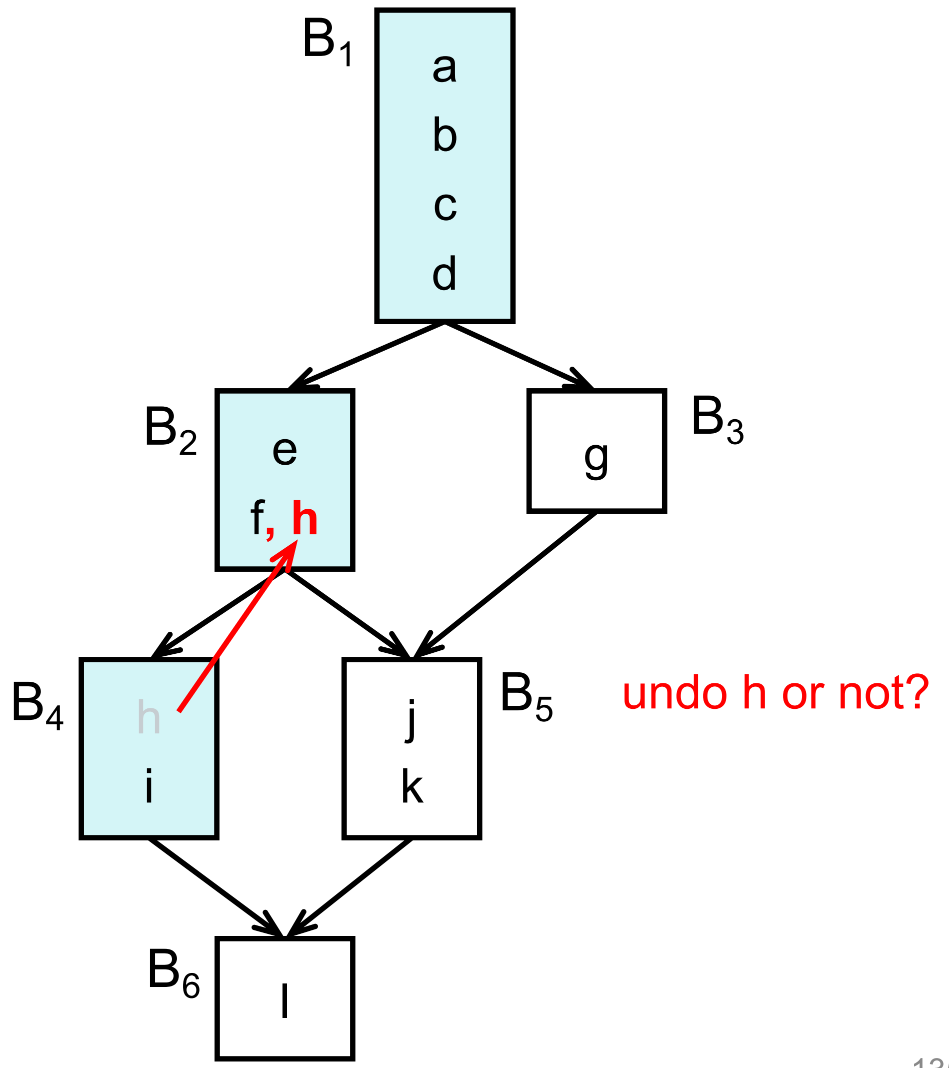
Context Problem at Join Points

- Join points
- Context Problem:
    - From B2 to B5, Undo h
    - From B3 to B5, Undo h changes the code semantics



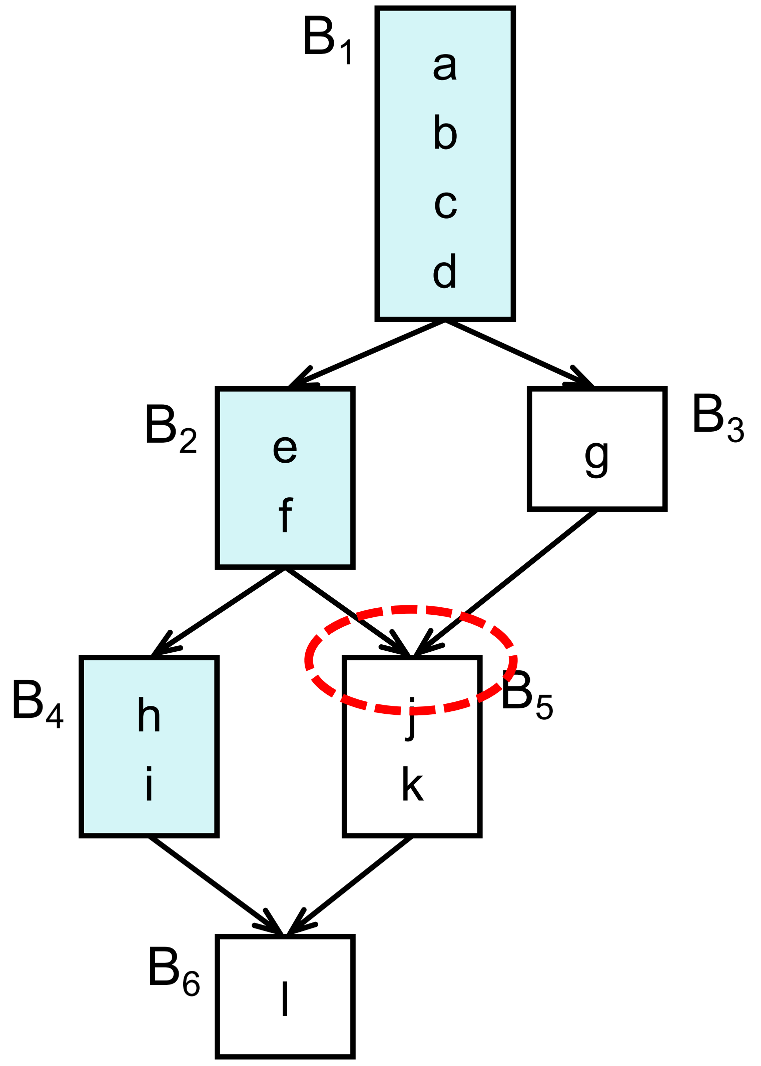
Context Problem at Join Points

- Join points
- Context Problem:
  - From B2 to B5, Undo h
  - From B3 to B5, Undo h
    changes the code semantics
- **Clone blocks to address the context problem**



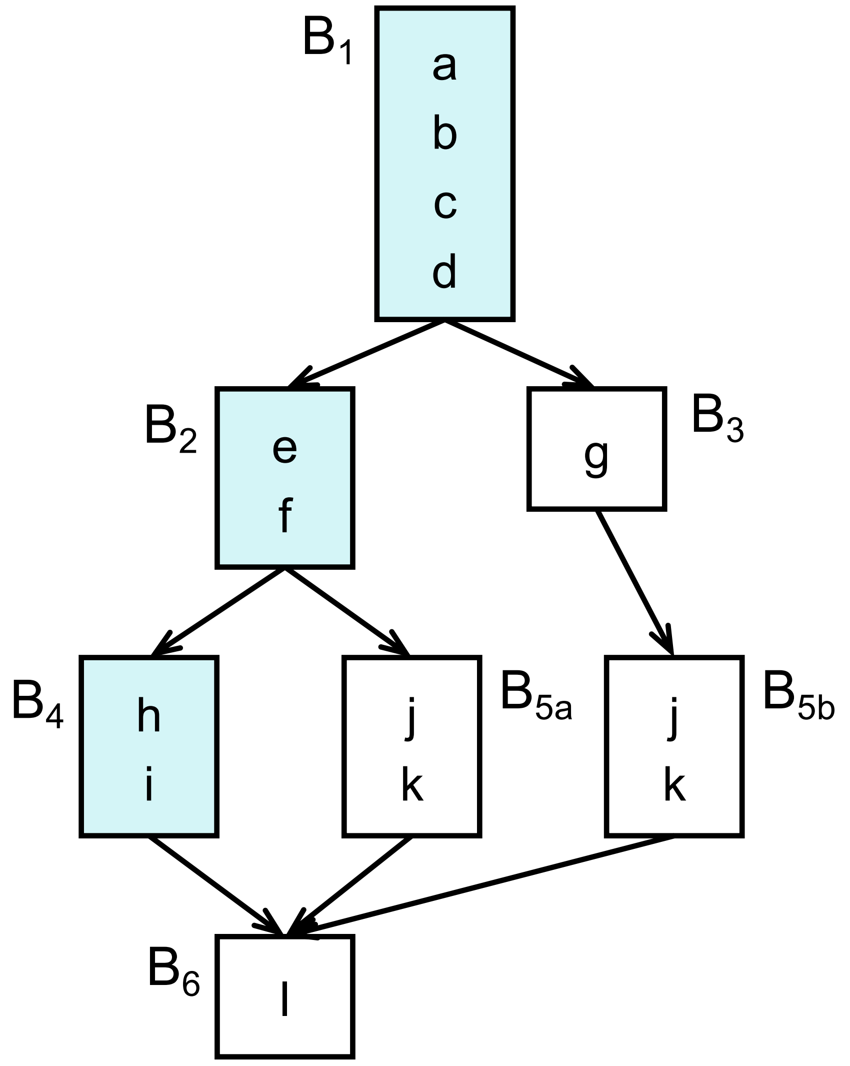
Context Problem at Join Points

- Join points
- Context Problem:
  - From B2 to B5, Undo h
  - From B3 to B5, Undo h
    changes the code semantics
- **Clone blocks to address the context problem**



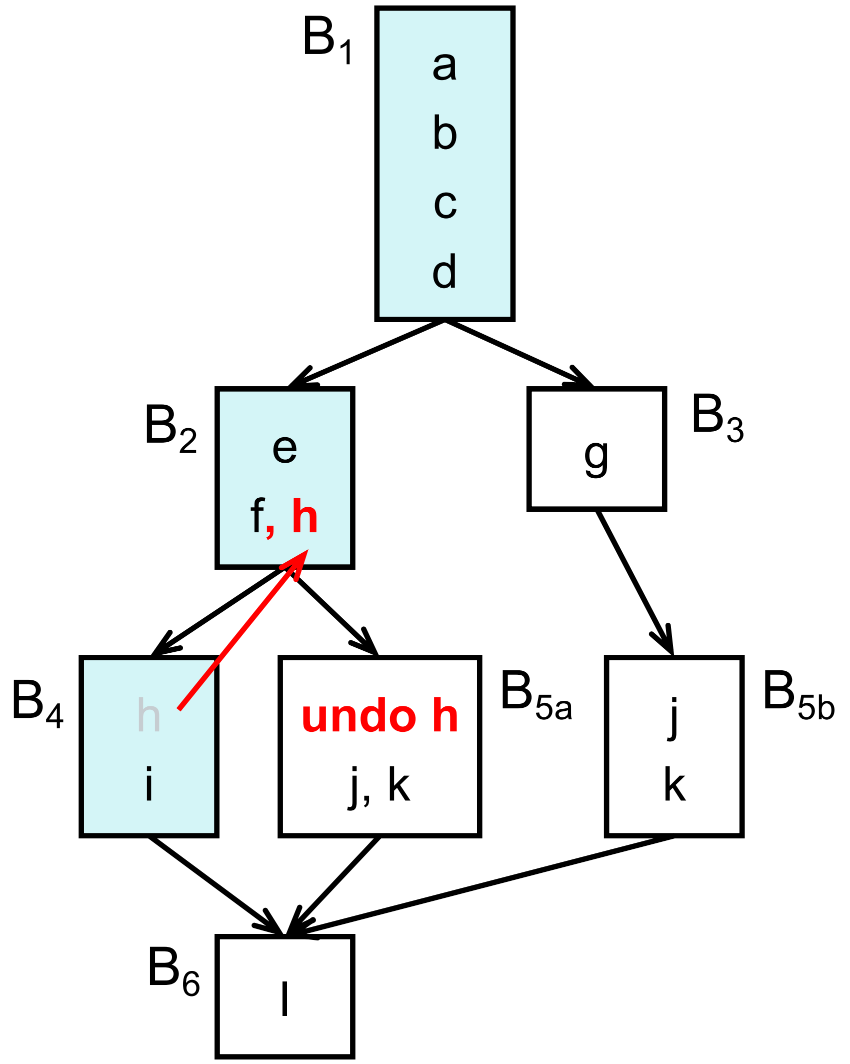
Context Problem at Join Points

- Join points
- Context Problem:
    - From B2 to B5, Undo h
    - From B3 to B5, Undo h
      changes the code semantics
- **Clone blocks to address the context problem**



Context Problem at Join Points

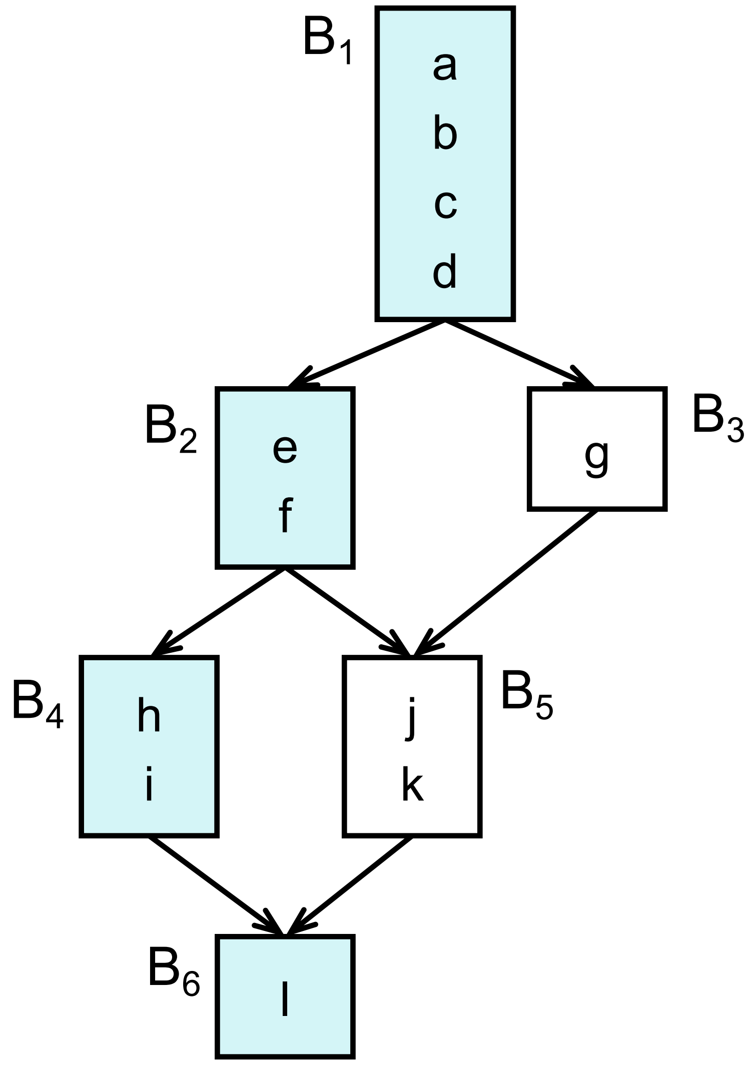
- Join points
- Context Problem:
  - From B2 to B5, Undo h
  - From B3 to B5, Undo h
    changes the code semantics
- **Clone blocks to address the context problem**



Trace Scheduling

Trace Scheduling

- Profile and schedule the hot path

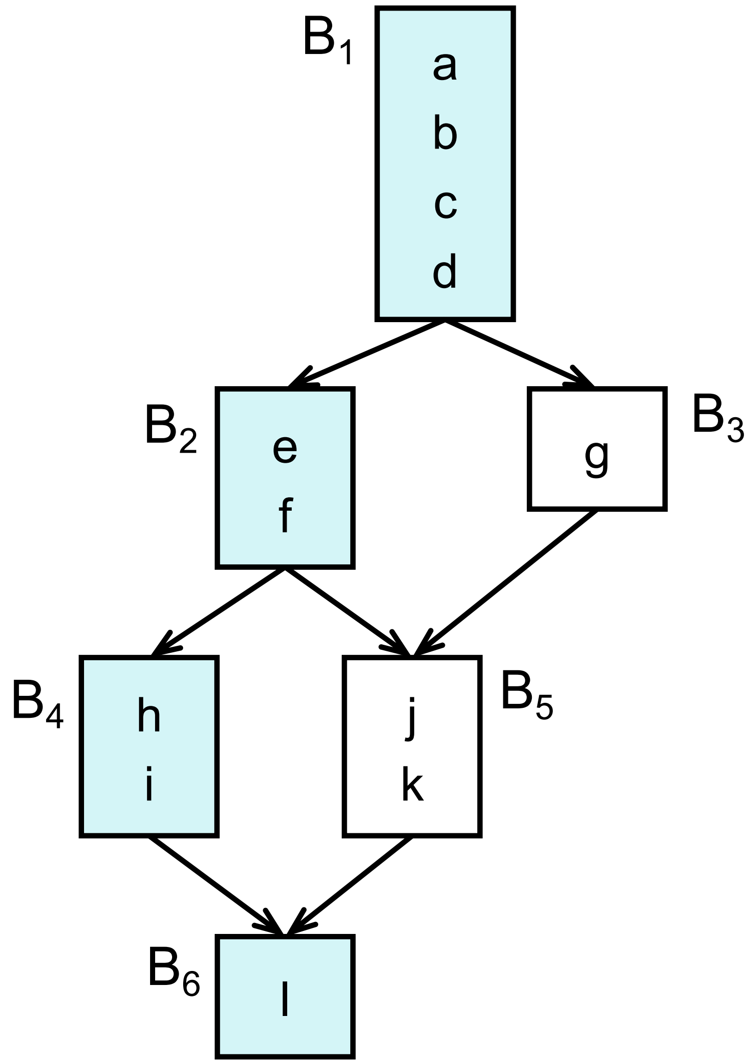


Trace Scheduling

- Profile and schedule the hot path

- Profiling-guided optimization
- Just-in-time compiler (that in JVM)

- What is JIT?



PART IV: Software Pipelining

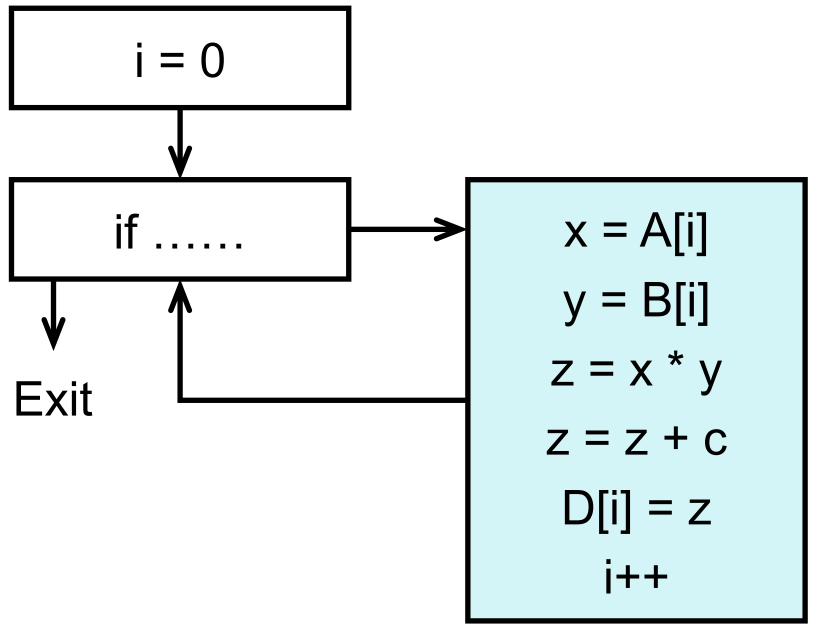
How about Loops?

- We can schedule the loop body as a single block

How about Loops?

- We can schedule the loop body as a single block

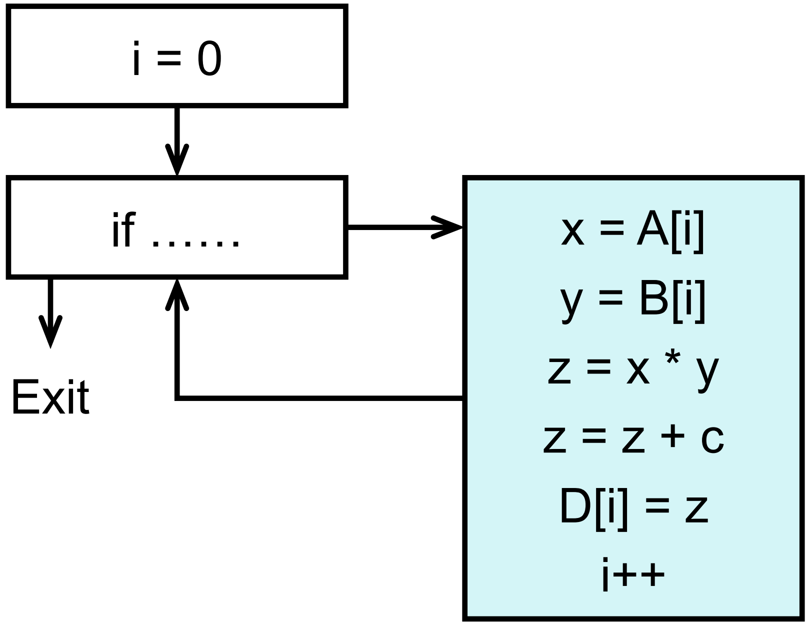
```
for (i = 0; ……; i++)
    D[i] = A[i] * B[i] + c
```



How about Loops?

- We can schedule the loop body as a single block
- Miss cross-iteration parallelism

```
for (i = 0; ……; i++)
    D[i] = A[i] * B[i] + c
```

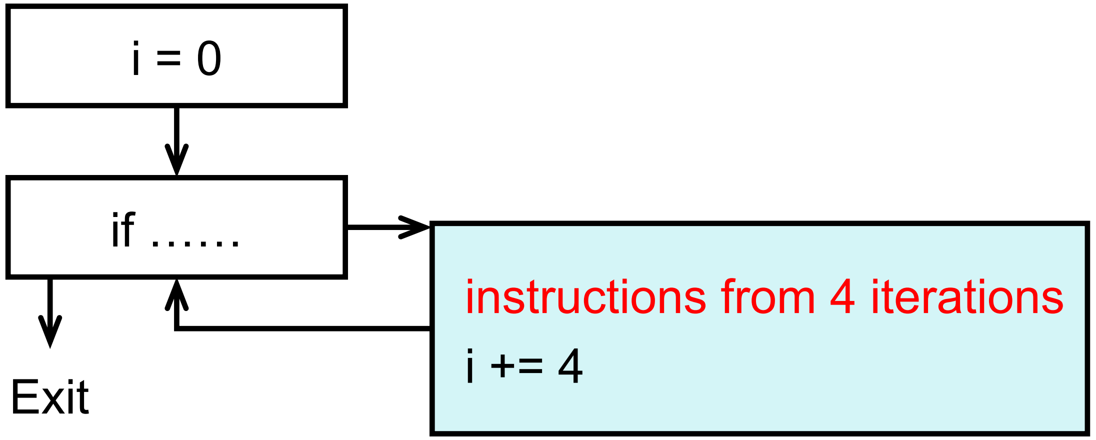


Loop Unrolling Helps!

```
for (i = 0; ……; i += 4)
     D[i]   = A[i]   * B[i]   + c
     D[i+1] = A[i+1] * B[i+1] + c
     D[i+2] = A[i+2] * B[i+2] + c
     D[i+3] = A[i+3] * B[i+3] + c
```

Loop Unrolling Helps!

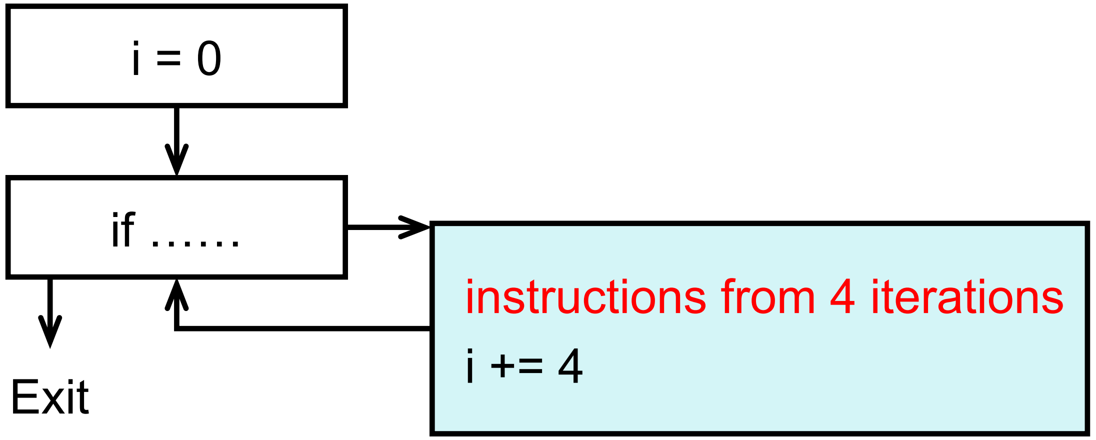
```
for (i = 0; ……; i += 4)
    D[i]   = A[i]   * B[i]   + c
    D[i+1] = A[i+1] * B[i+1] + c
    D[i+2] = A[i+2] * B[i+2] + c
    D[i+3] = A[i+3] * B[i+3] + c
```



Loop Unrolling Helps!

- We still schedule the loop body as a single block

```
for (i = 0; ……; i += 4)
    D[i]   = A[i]   * B[i]   + c
    D[i+1] = A[i+1] * B[i+1] + c
    D[i+2] = A[i+2] * B[i+2] + c
    D[i+3] = A[i+3] * B[i+3] + c
```

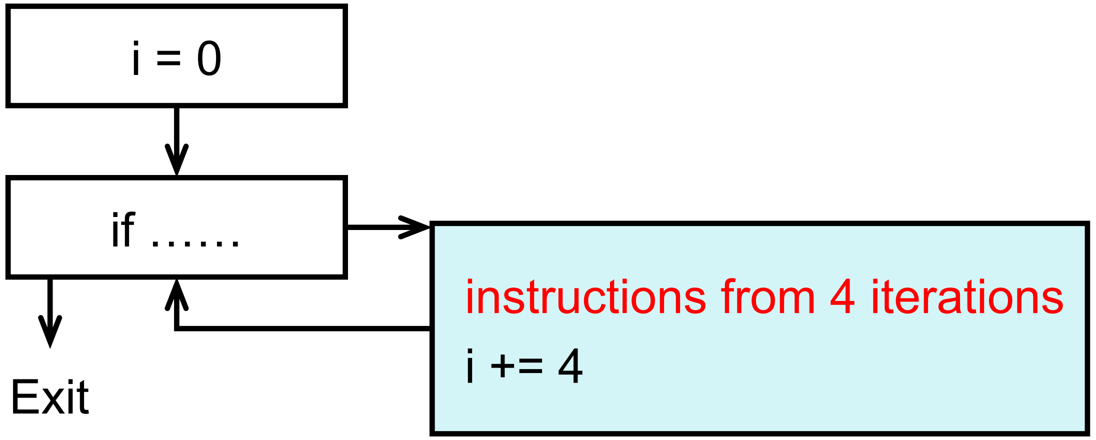


Loop Unrolling Helps!

- We still schedule the loop body as a single block

- Let us schedule instructions across multiple loop iterations

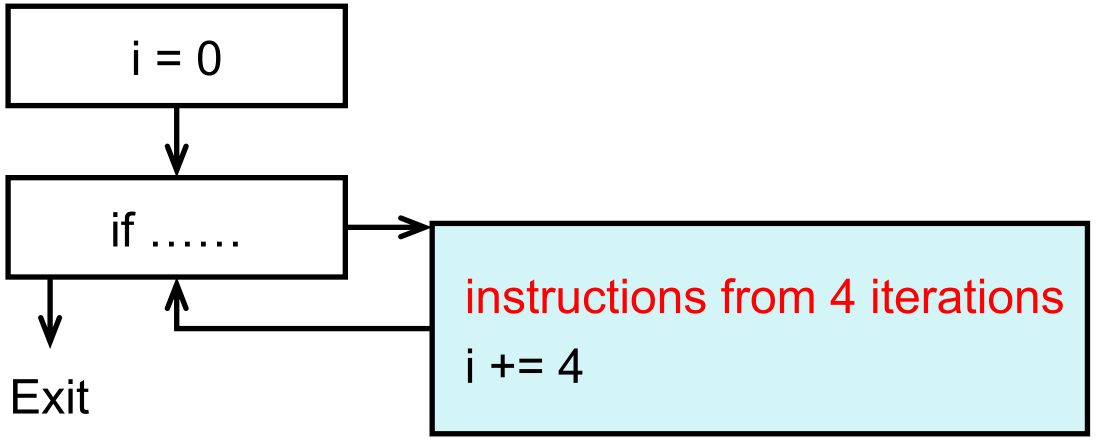
```
  for (i = 0; ……; i += 4)
       D[i]   = A[i]   * B[i]   + c
       D[i+1] = A[i+1] * B[i+1] + c
       D[i+2] = A[i+2] * B[i+2] + c
       D[i+3] = A[i+3] * B[i+3] + c
```



Problems of Loop Unrolling

- We cannot unroll too many times, which
- I. makes code size larger
  - cache pressure
  - register pressure
  - (slows down the code)
- II. and does not break the boundary of loop iteration

```
for (i = 0; ……; i += 4)
      D[i]   = A[i]   * B[i]   + c
      D[i+1] = A[i+1] * B[i+1] + c
      D[i+2] = A[i+2] * B[i+2] + c
      D[i+3] = A[i+3] * B[i+3] + c
```



Software Pipelining

- Break the boundary of loop iterations
- Regard a loop as a whole, not separate blocks
- More chances of parallelism without bloating the code

Software Pipelining

- Break the boundary of loop iterations
- Regard a loop as a whole, not separate blocks
- More chances of parallelism without bloating the code

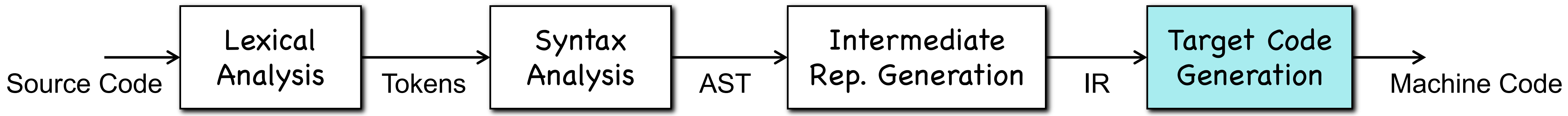
- ***Refer to Chapter 10, the Dragon book***

Software Pipelining

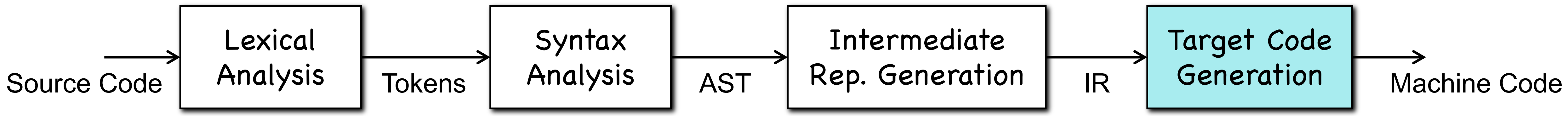
- Break the boundary of loop iterations
- Regard a loop as a whole, not separate blocks
- More chances of parallelism without bloating the code

- ***Refer to Chapter 10, the Dragon book***
- ***No Silver Bullet!***

Summary

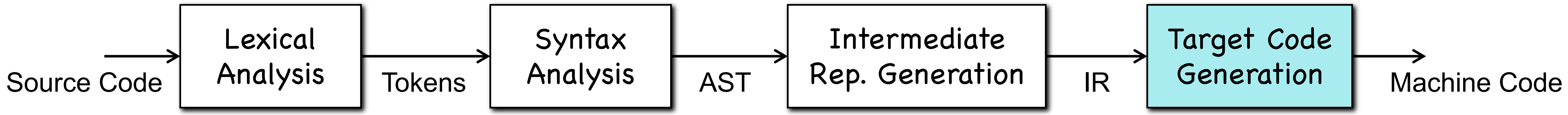

Source Code
Lexical Analysis
Tokens
Syntax Analysis
AST
Intermediate Rep. Generation
IR
Target Code Generation
Machine Code

Summary



Can we generate code to leverage modern CPUs for better instruction-level parallelization? **Yes! Possible!**

Summary



Can we generate code to leverage modern CPUs for better instruction-level parallelization? **Yes! Possible!**

- Scheduling a basic block
- Scheduling extended basic blocks
- Scheduling loops (optional)

THANKS!