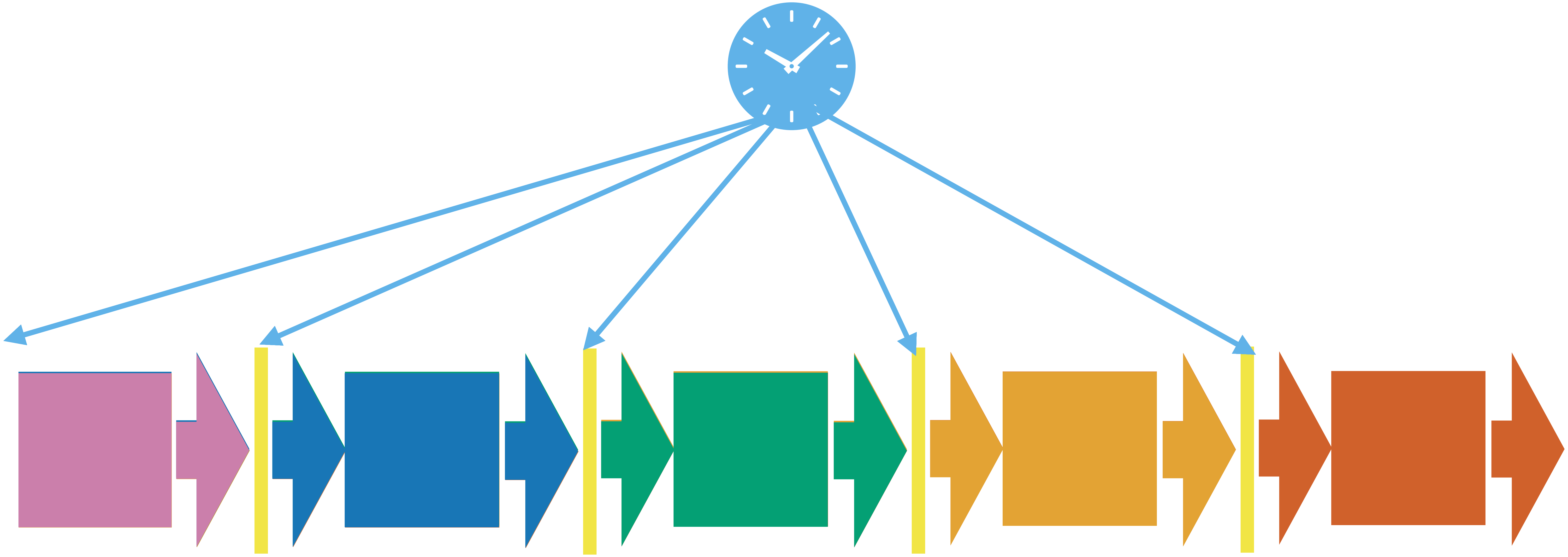
Data Hazards & Dynamic Instruction Scheduling

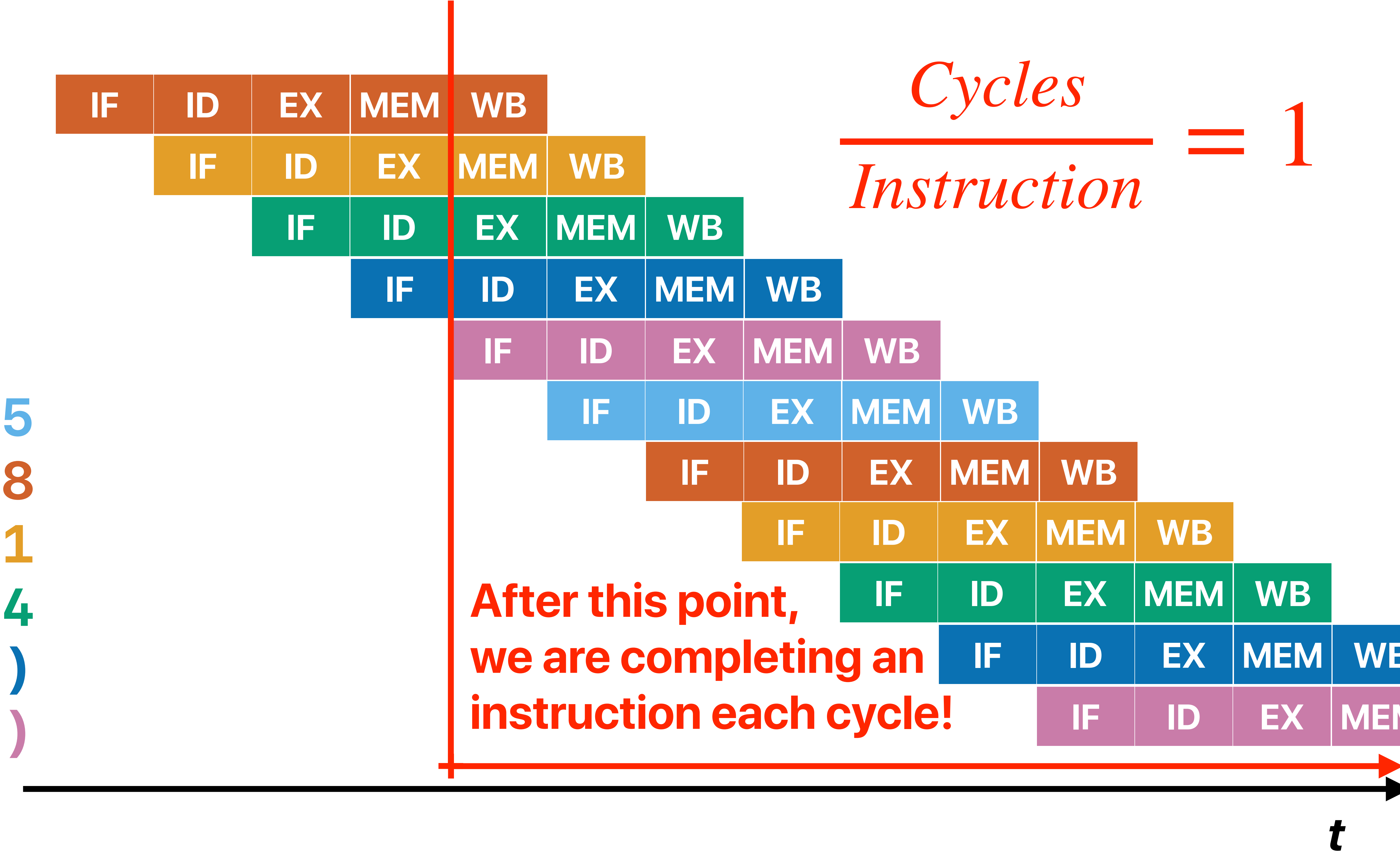
Hung-Wei Tseng

Recap: Pipelining



Recap: Pipelining

```
add x1, x2, x3
ld  x4, 0(x5)
sub x6, x7, x8
sub x9,x10,x11
sd  x1, 0(x12)
xor x13,x14,x15
and x16,x17,x18
add x19,x20,x21
sub x22,x23,x24
ld  x25, 4(x26)
sd  x27, 0(x28)
```



$$\frac{\mathit{Cycles}}{\mathit{Instruction}} = 1$$

After this point, we are completing an instruction each cycle!

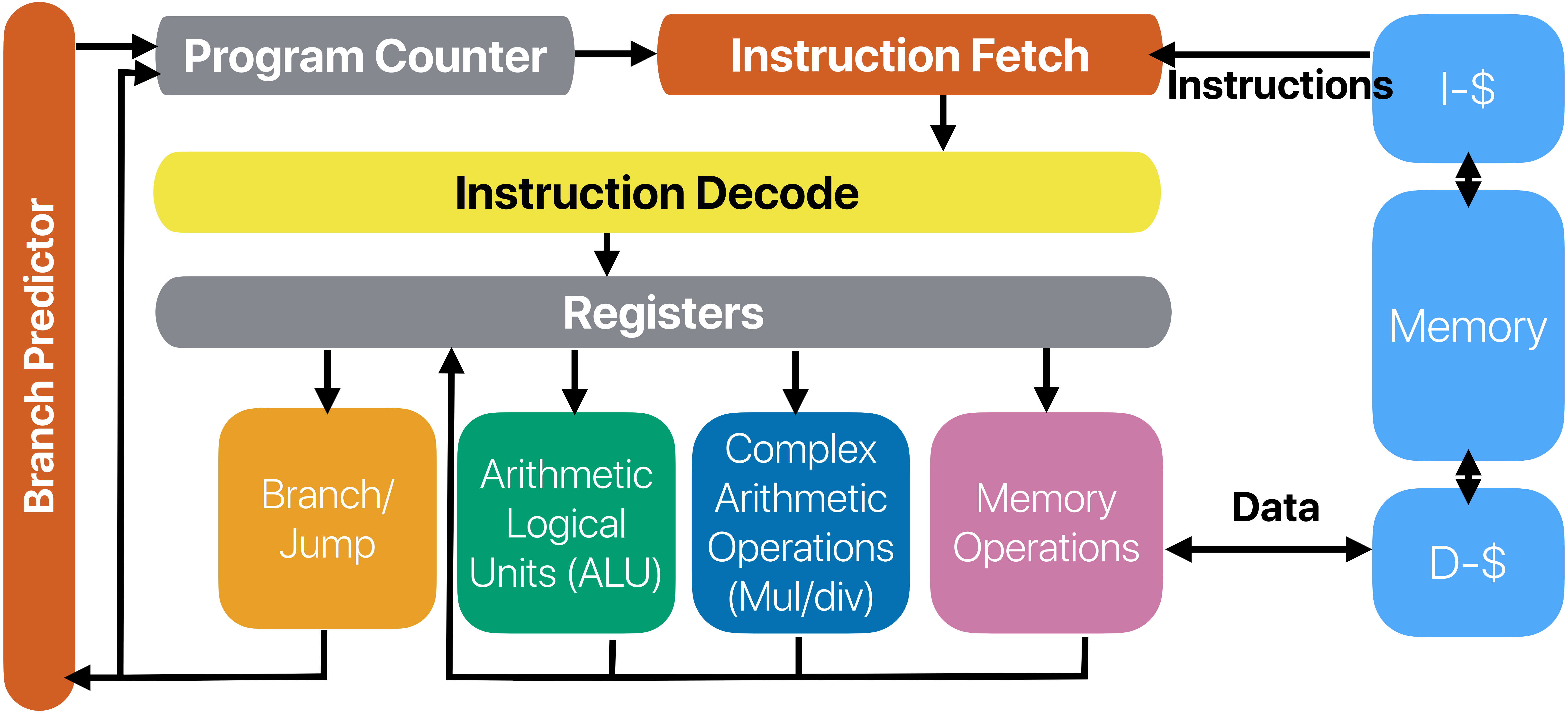
Recap: Three pipeline hazards

- Structural hazards — resource conflicts cannot support simultaneous execution of instructions in the pipeline
- Control hazards — the PC can be changed by an instruction in the pipeline
- Data hazards — an instruction depending on a the result that's not yet generated or propagated when the instruction needs that

Recap: addressing hazards

- Structural hazards
  - Stall
  - Modify hardware design
- Control hazards
  - Stall
  - Static prediction
  - Dynamic prediction — all “high-performance” processors nowadays have pretty decent branch predictors
    - Local bimodal
    - Global 2-level
    - Perceptron

The “current” pipeline



3:00

What will you do if you’re waiting for someone else’s response or outcome?

Ideas?

Outline

- Data hazards
  - Data forwarding
- SuperScalar
- Out-of-order, Dynamic instruction scheduling

Branch and programming

Demo revisited

```
if(option)
    std::sort(data, data + arraySize);

for (unsigned i = 0; i < 100000; ++i) {
    int threshold = std::rand();
    for (unsigned i = 0; i < arraySize; ++i) {
        if (data[i] >= threshold)
            sum ++;
    }
}
```

option = 1 is faster!!!

```
SELECT count(*) FROM TABLE WHERE val < A and val >= B;
```

https://www.pollev.com/hungweitseng close in 1:30

Demo revisited

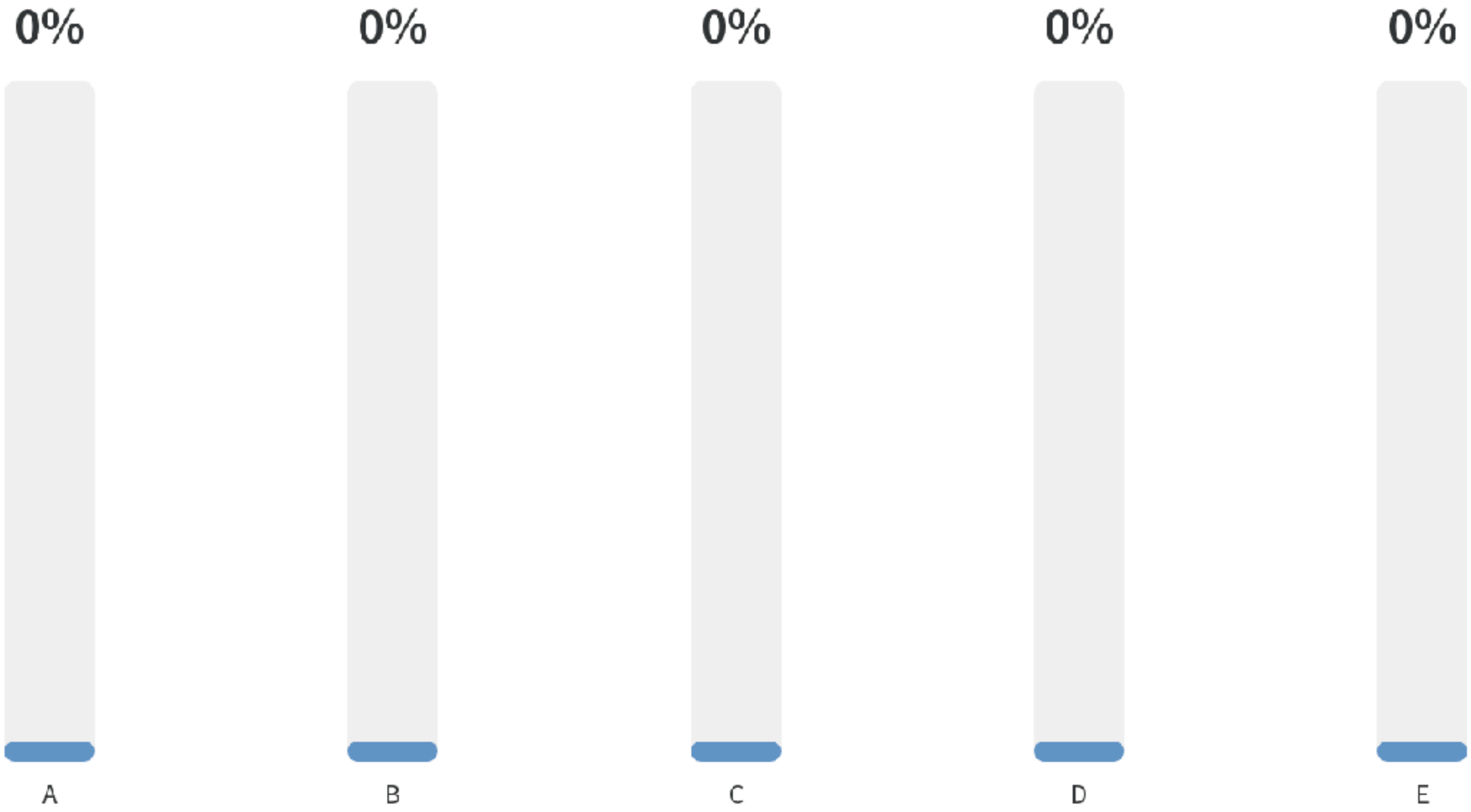
- Why the performance is better when option is not “0”
  ① The amount of dynamic instructions needs to execute is a lot smaller
  ② The amount of branch instructions to execute is smaller
  ③ The amount of branch mis-predictions is smaller
  ④ The amount of data accesses is smaller

A. 0
B. 1
C. 2
D. 3
E. 4

```
if(option)
    std::sort(data, data + arraySize);

for (unsigned i = 0; i < 100000; ++i) {
    int threshold = std::rand();
    for (unsigned i = 0; i < arraySize; ++i) {
        if (data[i] >= threshold)
            sum ++;
    }
}
```

✓ 0



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

https://www.pollev.com/hungweitseng close in 1:30

Demo revisited

- Why the performance is better when option is not “0”
  ① The amount of dynamic instructions needs to execute is a lot smaller
  ② The amount of branch instructions to execute is smaller
  ③ The amount of branch mis-predictions is smaller
  ④ The amount of data accesses is smaller

A. 0

B. 1

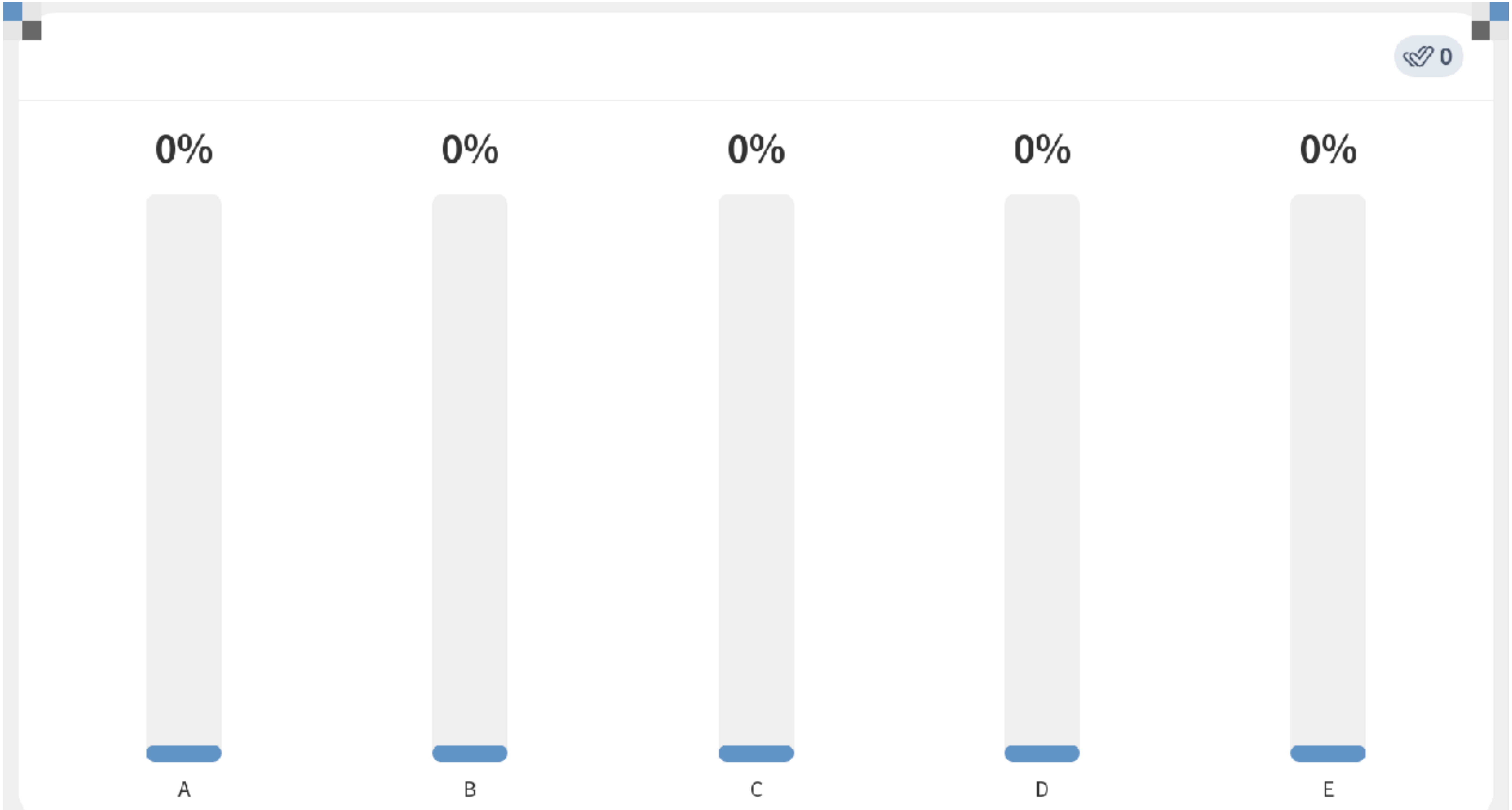
C. 2

D. 3

E. 4

```
if(option)
    std::sort(data, data + arraySize);

for (unsigned i = 0; i < 100000; ++i) {
    int threshold = std::rand();
    for (unsigned i = 0; i < arraySize; ++i) {
        if (data[i] >= threshold)
            sum ++;
    }
}
```



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Demo revisited

- Why the performance is better when option is not "0"
  - ① The amount of dynamic instructions needs to execute is a lot smaller
  - ② The amount of branch instructions to execute is smaller
  - [✓ ③ The amount of branch mis-predictions is smaller]
  - ④ The amount of data accesses is smaller

A. 0

[✓ B. 1]

C. 2

D. 3

E. 4

```
if(option)
    std::sort(data, data + arraySize);

for (unsigned i = 0; i < 100000; ++i) {
    int threshold = std::rand();
    for (unsigned i = 0; i < arraySize;
        if (data[i] >= threshold)
            sum ++;
    }
}
```

branch X

	Without sorting	With sorting
The prediction accuracy of X before threshold	50%	100%
The prediction accuracy of X after threshold	50%	100%

Demo revisited: evaluating the cost of mis-predicted branches

- Compare the number of mis-predictions
- Calculate the difference of cycles
- We can get the “average CPI” of a mis-prediction!

34 cycles!!!

Data hazards

Data hazards

- An instruction currently in the pipeline cannot receive the “logically” correct value for execution
- Data dependencies
  - The output of an instruction is the input of a later instruction
  - May result in data hazard if the later instruction that consumes the result is still in the pipeline

https://www.pollev.com/hungweitseng close in 1:30

How many data dependencies do we have?

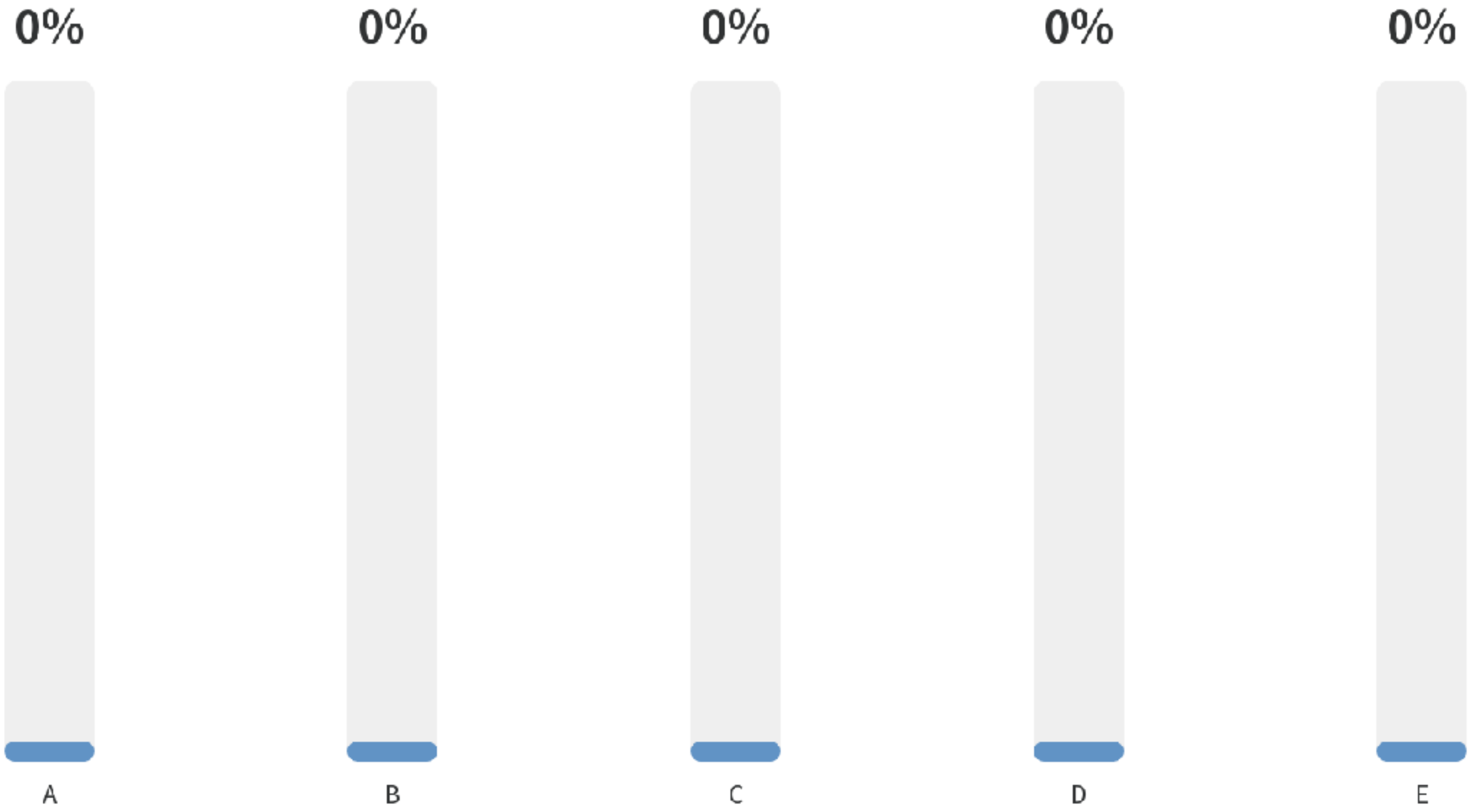
- How many pairs of data dependences are there in the following x86 instructions?

```
movl     (%rdi), %eax
movl     (%rsi), %edx
movl     %edx, (%rdi)
movl     %eax, (%rsi)
```

```
int temp = *a;
*a = *b;
*b = temp;
```

A. 1
B. 2
C. 3
D. 4
E. 5

0



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

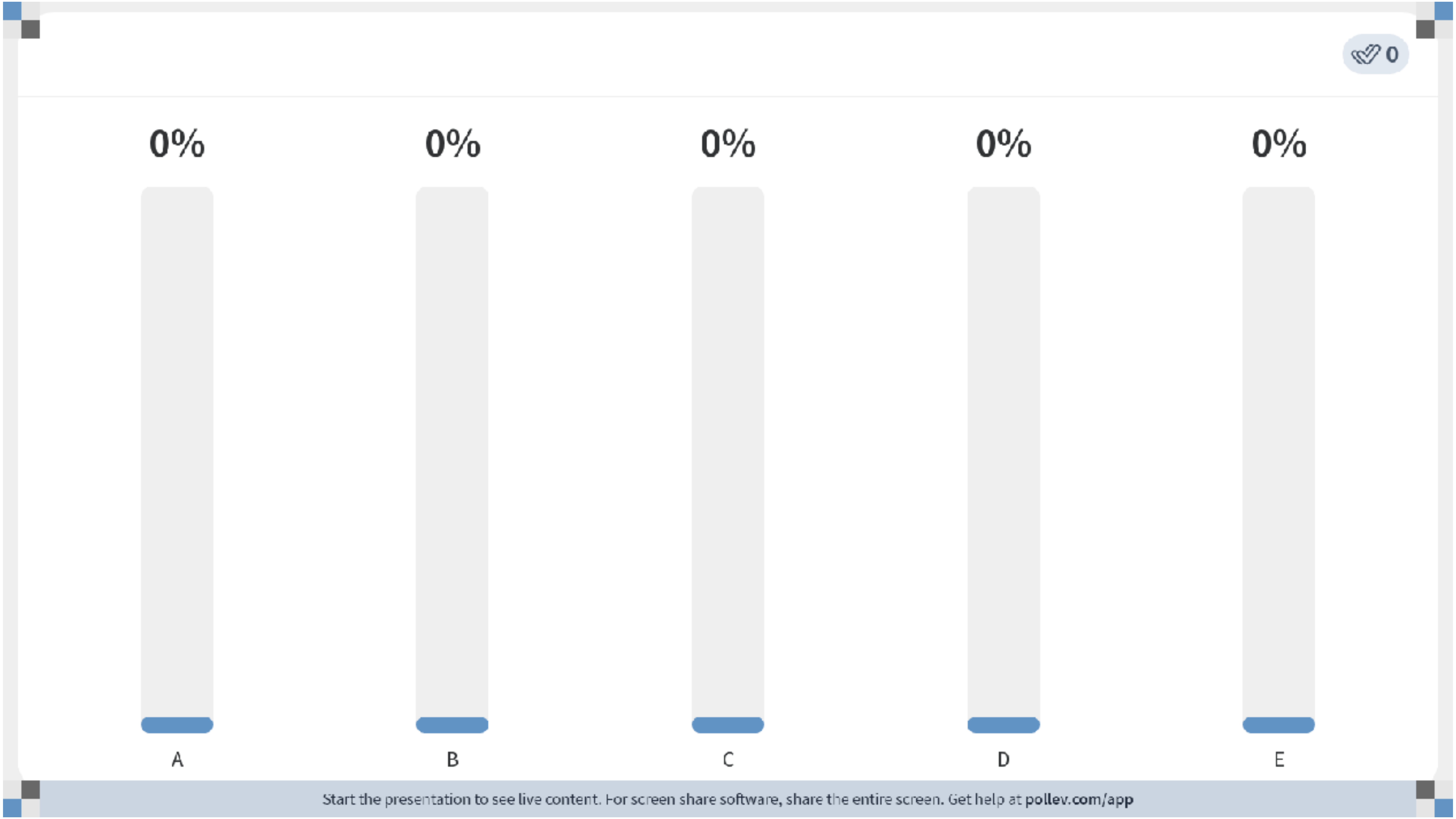
https://www.pollev.com/hungweitseng close in 1:30

How many data dependencies do we have?

- How many pairs of data dependences are there in the following x86 instructions?

```
movl     (%rdi), %eax
movl     (%rsi), %edx
movl    %edx, (%rdi)
movl    %eax, (%rsi)
```

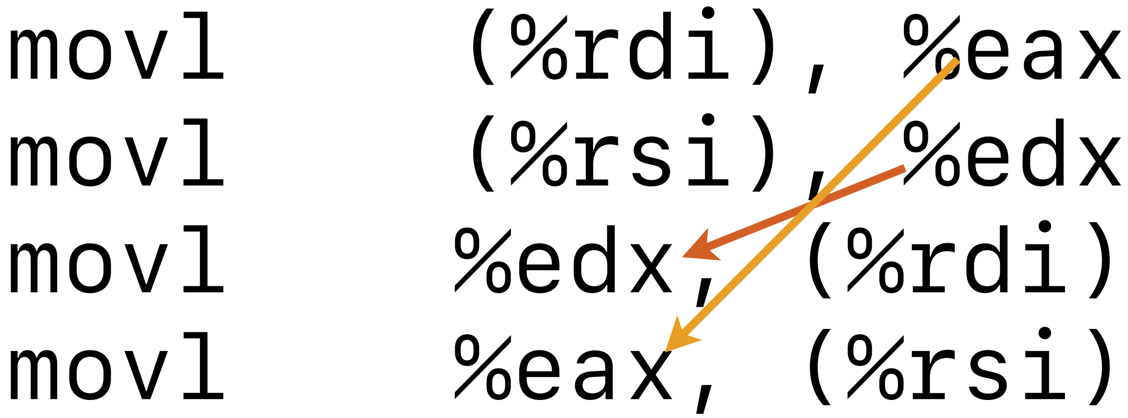
A. 1
B. 2
C. 3
D. 4
E. 5



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

How many dependencies do we have?

- How many pairs of data dependences are there in the following x86 instructions?



```
movl    (%rdi), %eax
movl    (%rsi), %edx
movl    %edx, (%rdi)
movl    %eax, (%rsi)
```

```c
int temp = *a;
*a = *b;
*b = temp;
```

A. 1

[✓ B. 2]

C. 3

D. 4

E. 5

https://www.pollev.com/hungweitseng close in 1:30

How many data dependencies do we have?

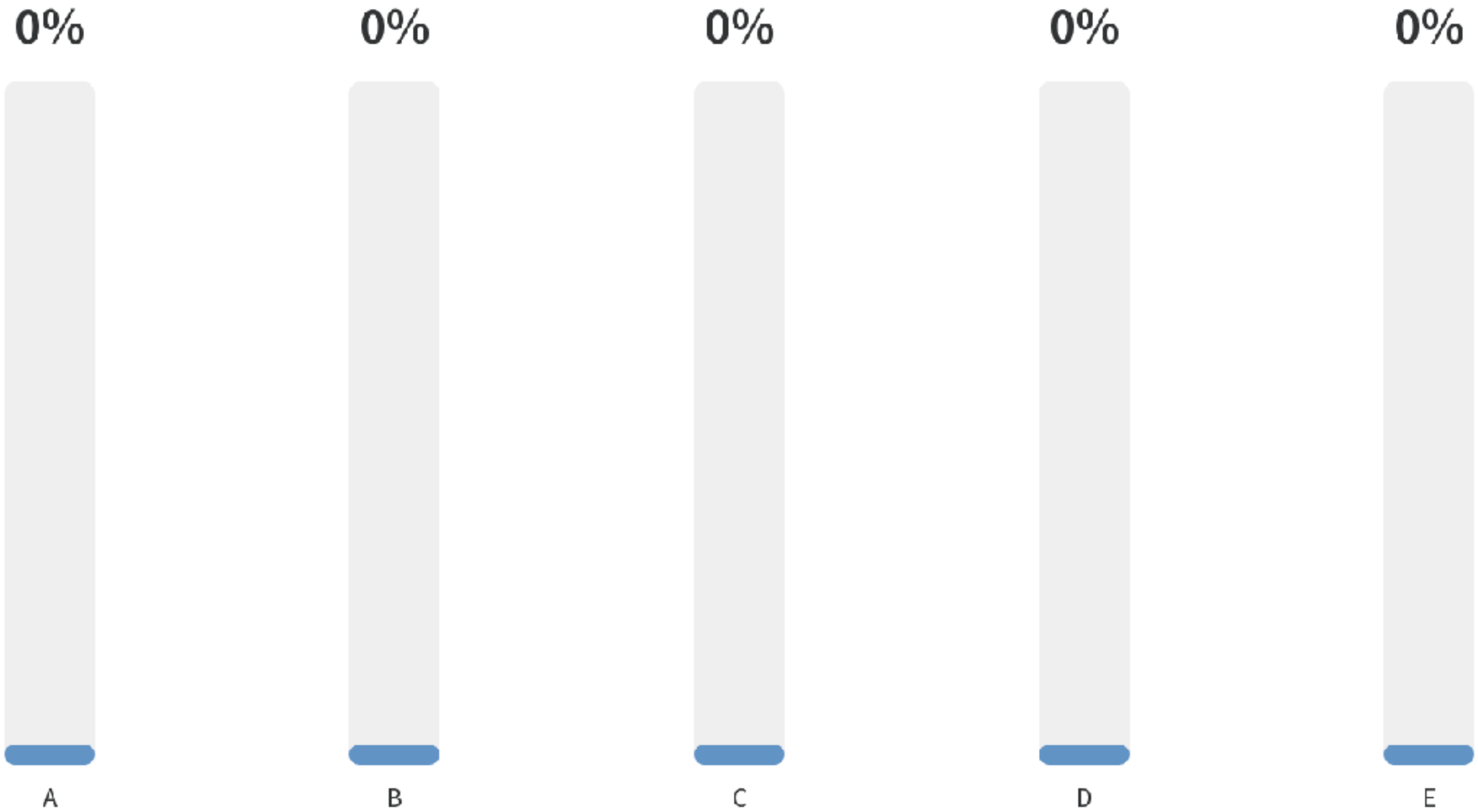
- How many pairs of data dependences are there in the following x86 instructions?

```
movl    (%rdi), %eax
xorl    (%rsi), %eax
movl    %eax, (%rdi)
xorl    (%rsi), %eax
movl    %eax, (%rsi)
xorl    %eax, (%rdi)
```

```
*a ^= *b;
*b ^= *a;
*a ^= *b;
```

A. 1
B. 2
C. 3
D. 4
E. 5

✓ 0



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

https://www.pollev.com/hungweitseng close in 1:30

How many data dependencies do we have?

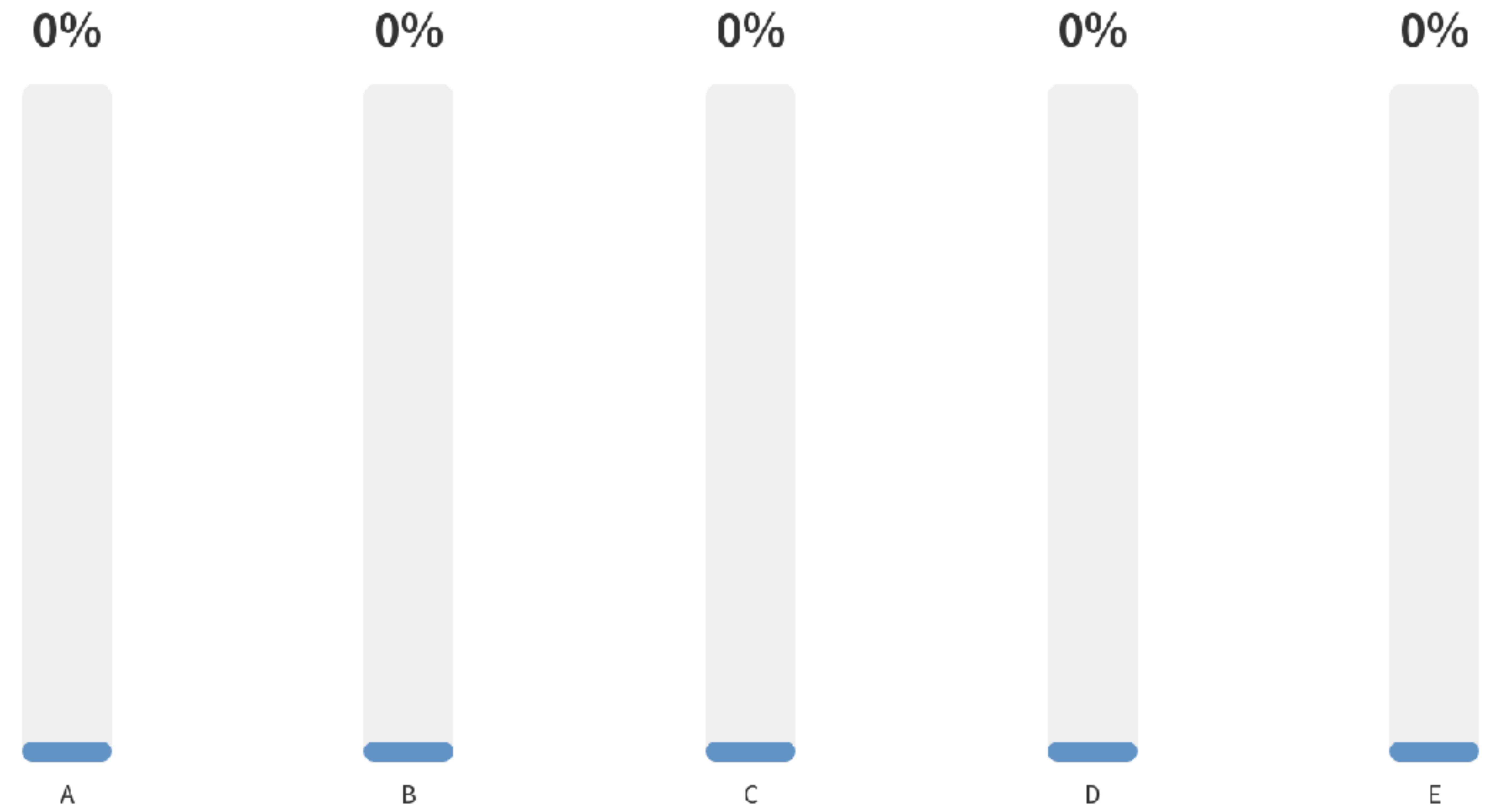
- How many pairs of data dependences are there in the following x86 instructions?

```
movl    (%rdi), %eax
xorl    (%rsi), %eax
movl    %eax, (%rdi)
xorl    (%rsi), %eax
movl    %eax, (%rsi)
xorl    %eax, (%rdi)
```

```
*a ^= *b;
*b ^= *a;
*a ^= *b;
```

A. 1
B. 2
C. 3
D. 4
E. 5

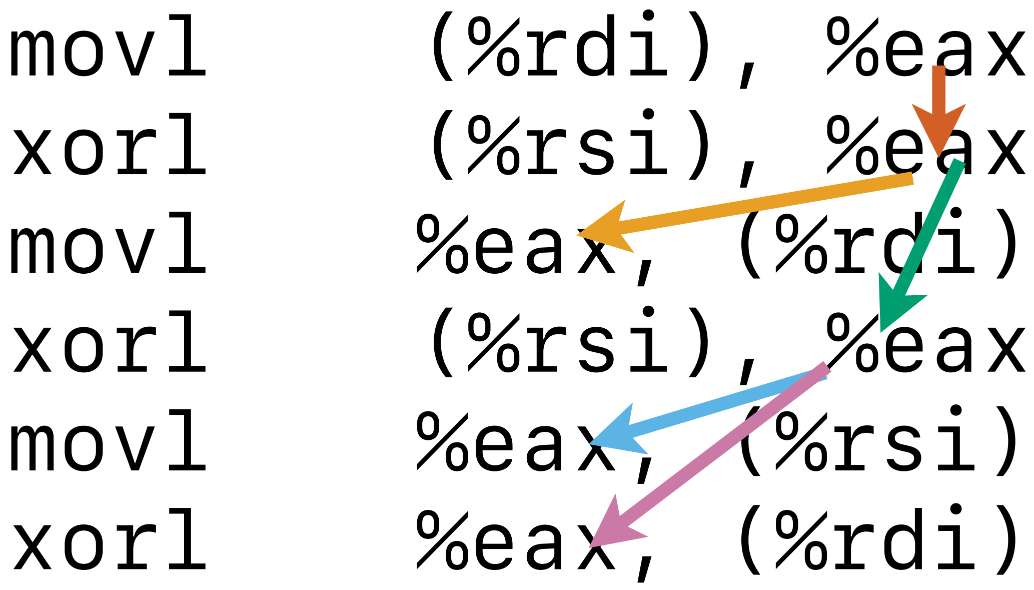
0



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

How many dependencies do we have?

- How many pairs of data dependences are there in the following x86 instructions?



```
movl    (%rdi), %eax
xorl    (%rsi), %eax
movl    %eax, (%rdi)
xorl    (%rsi), %eax
movl    %eax, (%rsi)
xorl    %eax, (%rdi)
```

```
*a ^= *b;
*b ^= *a;
*a ^= *b;
```

A. 1
B. 2
C. 3
D. 4
[✓ E. 5]

https://www.pollev.com/hungweitseng close in 1:30

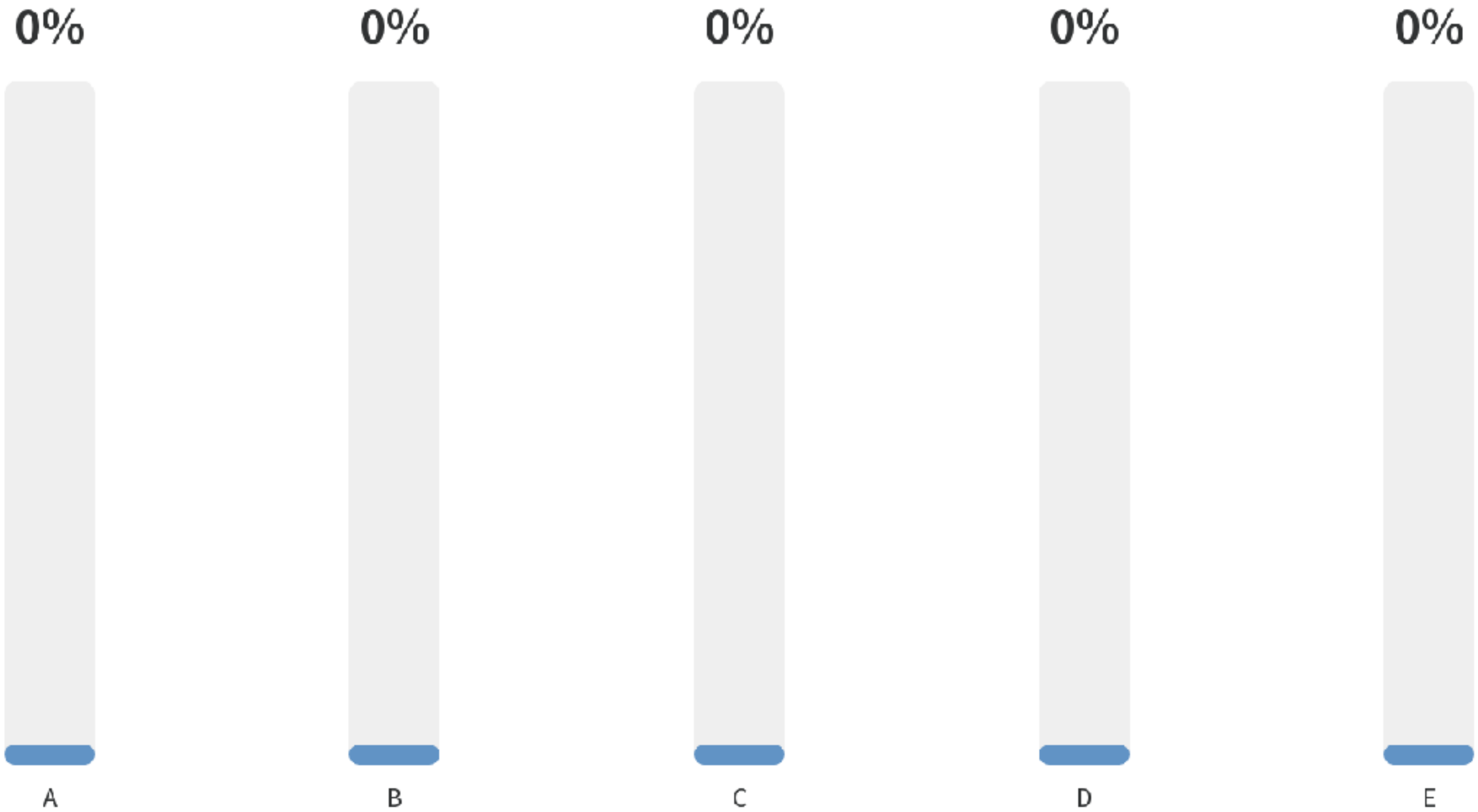
Data hazards?

- How many pairs of data dependences in the following x86 instructions will result in data hazards if a memory operation (assume 100% cache hit rate) takes 4 cycles?

```
movl    (%rdi), %eax
movl    (%rsi), %edx
movl   %edx, (%rdi)
movl   %eax, (%rsi)
```

A. 0
B. 1
C. 2
D. 3
E. 4

0



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

https://www.pollev.com/hungweitseng close in 1:30

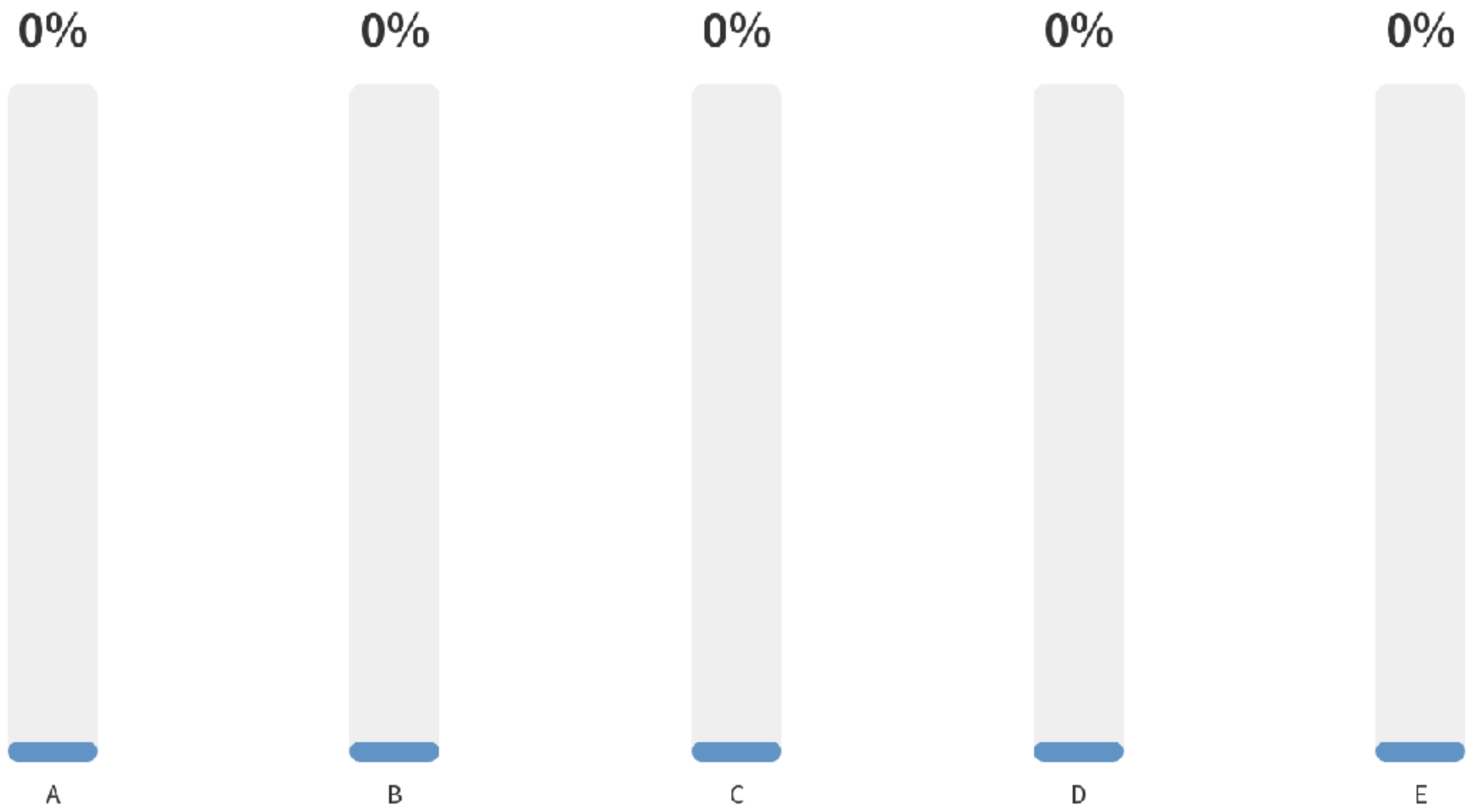
Data hazards?

- How many pairs of data dependences in the following x86 instructions will result in data hazards if a memory operation (assume 100% cache hit rate) takes 4 cycles?

```
movl    (%rdi), %eax
movl    (%rsi), %edx
movl   %edx, (%rdi)
movl   %eax, (%rsi)
```

A. 0
B. 1
C. 2
D. 3
E. 4

0

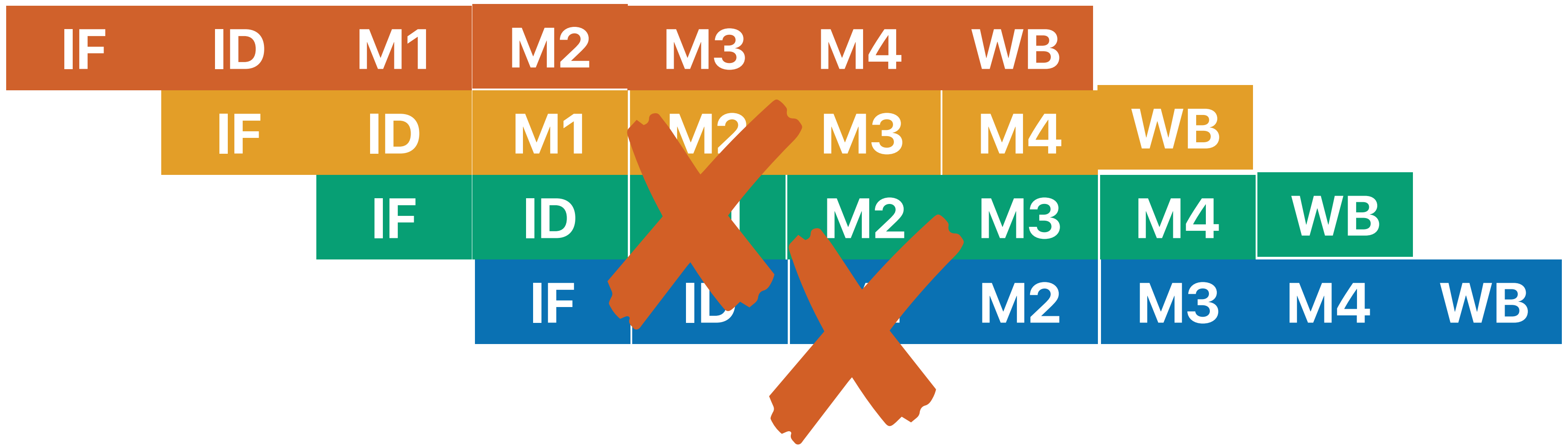


Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Data hazards?

- How many pairs of data dependences in the following x86 instructions will result in data hazards if a memory operation (assume 100% cache hit rate) takes 4 cycles?

```
movl     (%rdi), %eax
movl     (%rsi), %edx
movl    %edx, (%rdi)
movl    %eax, (%rsi)
```



A. 0

B. 1

C. 2

D. 3

E. 4

Data hazards

```
① movl    (%rdi), %eax
② movl    (%rsi), %edx
③ movl    %edx, (%rdi)
④ movl    %eax, (%rsi)
```

	IF	ID	ALU/BR/M1	M2	M3	M4	WB
1	(1)						
2	(2)	(1)					
3	(3)	(2)	(1)				
4	(4)	(3)	(2)	(1)			
5		(4)	(3)	(2)	(1)		
6			(4)	(3)	(2)	(1)	
7				(4)	(3)	(2)	(1)
8					(4)	(3)	(2)
9						(4)	(3)
10							(4)
11							
12							
13							
14							

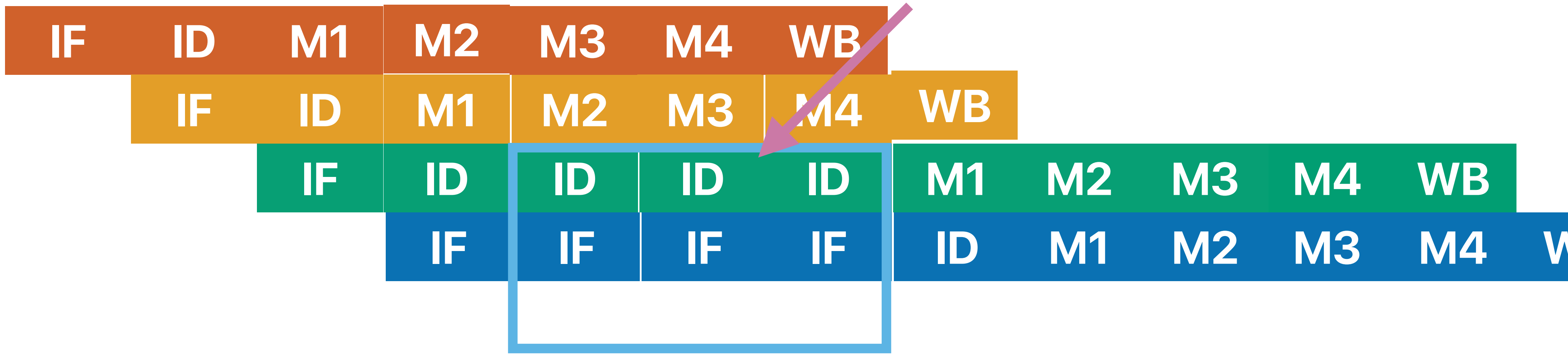
%eax does not have our desired value

%edx does not have our desired value

Solution 1: Let's try "stall" again

- Whenever the input is not ready when the consumer is decoding, just stall — the consumer stays at ID.

```
movl    (%rdi), %eax
movl    (%rsi), %edx
movl    %edx, (%rdi)
movl    %eax, (%rsi)
```



we have the value for %edx already! Why another cycle?

movl (%rdi), %eax	IF	ID	M1	M2	M3	M4	WB						
movl (%rsi), %edx		IF	ID	M1	M2	M3	M4	WB					
movl %edx, (%rdi)			IF	ID	ID	ID	ID	M1	M2	M3	M4	WB	
movl %eax, (%rsi)				IF	IF	IF	IF	ID	M1	M2	M3	M4	W

3 additional cycles

Solution 1: Let's try "stall" again

- Whenever the input is not ready when the consumer is decoding, just stall — the consumer stays at ID.

```
① movl    (%rdi), %eax
② movl    (%rsi), %edx
③ movl    %edx, (%rdi)
④ movl    %eax, (%rsi)
```

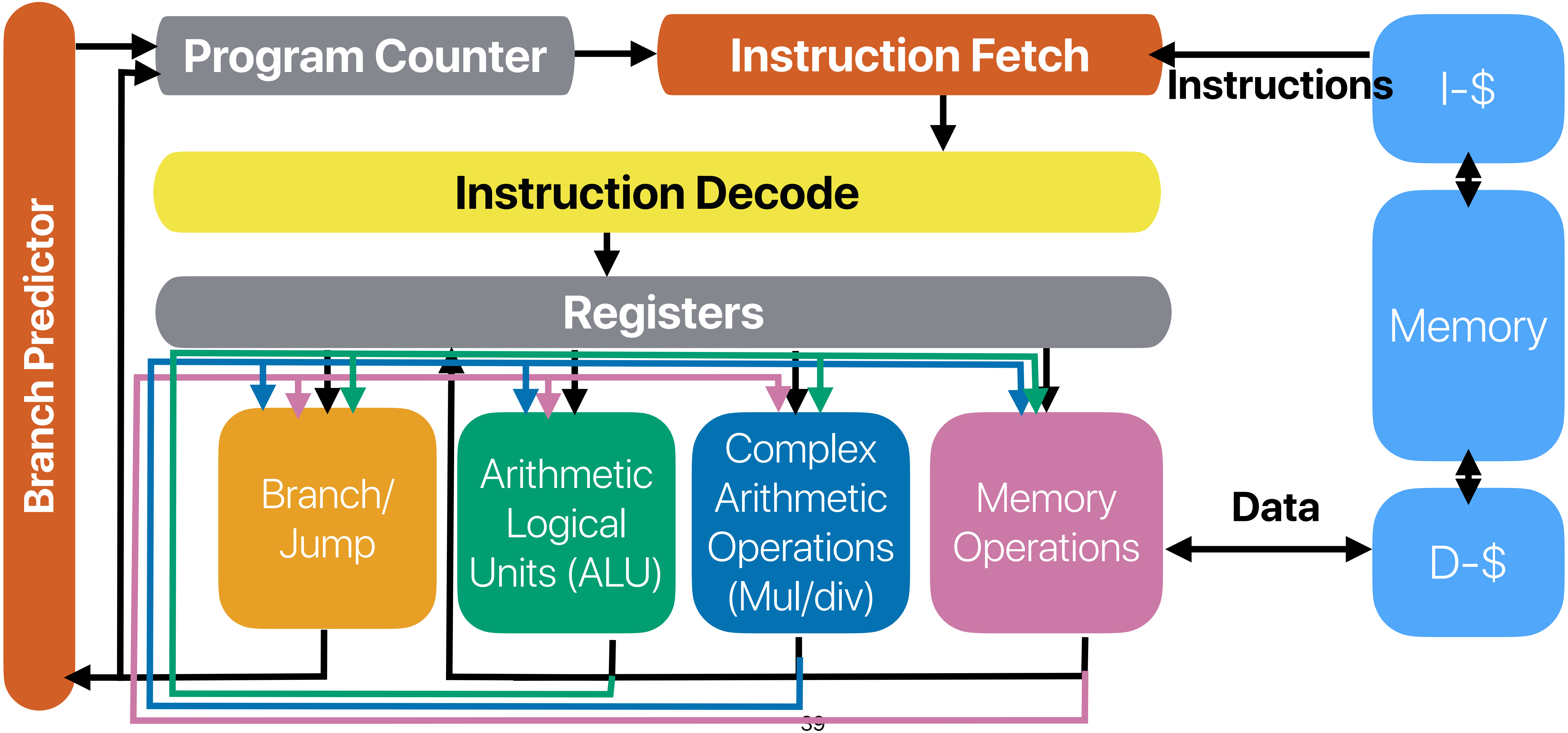
	IF	ID	ALU/BR/M1	M2	M3	M4	WB
1	(1)						
2	(2)	(1)					
3	(3)	(2)	(1)				
4	(4)	(3)	(2)	(1)			
5	(4)	(3)		(2)	(1)		
6	(4)	(3)			(2)	(1)	
7	(4)	(3)				(2)	(1)
8	(4)	(3)					(2)
9		(4)	(3)				
10			(4)	(3)			
11				(4)	(3)		
12					(4)	(3)	
13						(4)	(3)
14							(4)

we have the value for %edx already!
Why another cycle?

Solution 2: Data forwarding

- Add logics/wires to forward the desired values to the demanding instructions

Data “forwarding”



https://www.pollev.com/hungweitseng close in 1:30

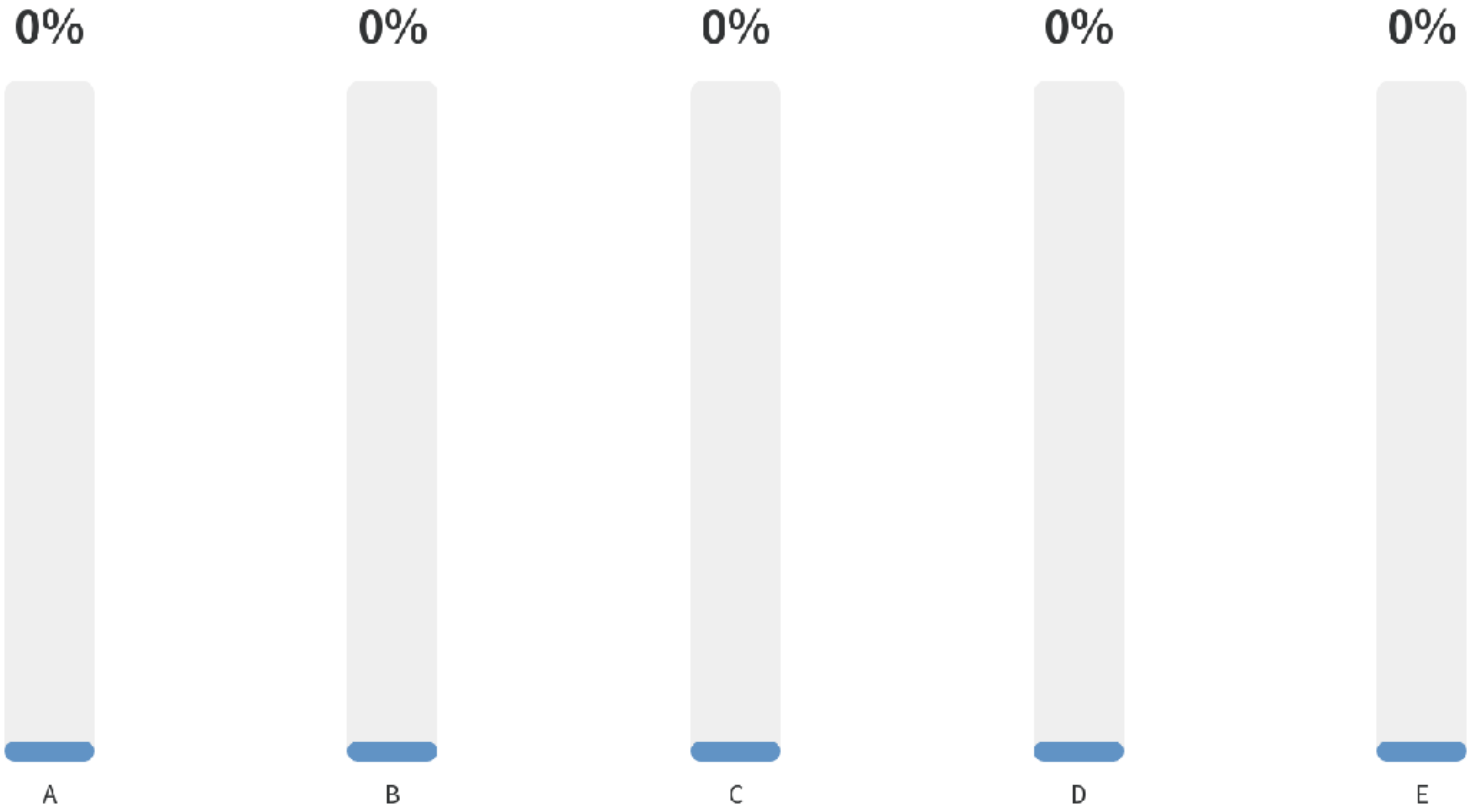
How many data dependencies are still problematic?

- How many pairs of data dependences in the following x86 instructions are still problematic with data forwarding if a memory operation (assume 100% cache hit rate) takes 4 cycles?

```
movl     (%rdi), %eax
movl     (%rsi), %edx
movl    %edx, (%rdi)
movl    %eax, (%rsi)
```

A. 0
B. 1
C. 2
D. 3
E. 4

0



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

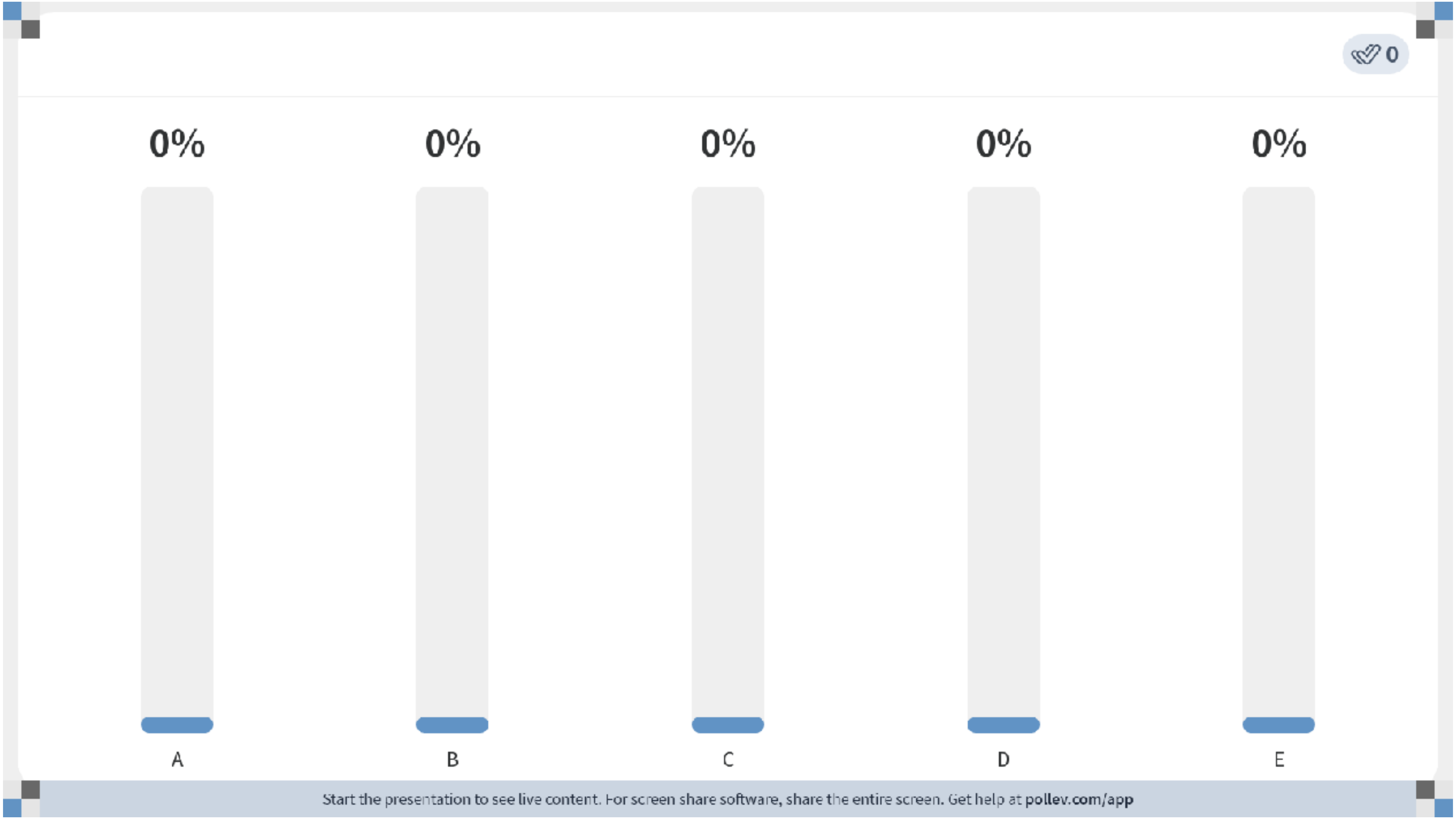
https://www.pollev.com/hungweitseng close in 1:30

How many data dependencies are still problematic?

- How many pairs of data dependences in the following x86 instructions are still problematic with data forwarding if a memory operation (assume 100% cache hit rate) takes 4 cycles?

```
movl    (%rdi), %eax
movl    (%rsi), %edx
movl    %edx, (%rdi)
movl    %eax, (%rsi)
```

A. 0
B. 1
C. 2
D. 3
E. 4

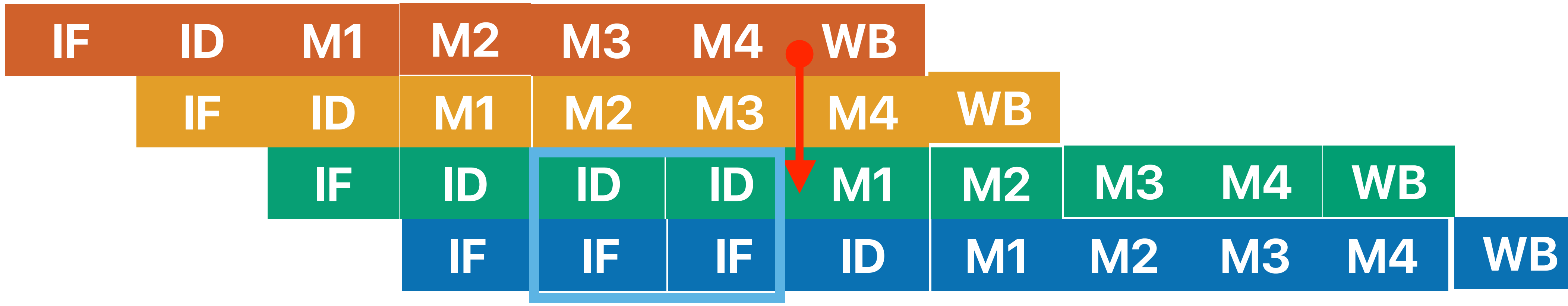


Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

How many data dependencies are still problematic?

- How many pairs of data dependences in the following x86 instructions are still problematic with data forwarding if a memory operation (assume 100% cache hit rate) takes 4 cycles?

```
movl     (%rdi), %eax
movl     (%rsi), %edx
movl    %edx,  (%rdi)
movl    %eax,  (%rsi)
```



2 additional cycles

A. 0

[✓ B. 1]

C. 2

D. 3

E. 4

```
int temp = *a;
*a = *b;
*b = temp;
```

Solution 2: Data forwarding

- Whenever the input is not ready when the consumer is decoding, just stall — the consumer stays at ID.

```
① movl     (%rdi), %eax
② movl     (%rsi), %edx
③ movl     %edx, (%rdi)
④ movl     %eax, (%rsi)
```

	IF	ID	ALU/BR/M1	M2	M3	M4	WB
1	(1)						
2	(2)	(1)					
3	(3)	(2)	(1)				
4	(4)	(3)	(2)	(1)			
5	(4)	(3)		(2)	(1)		
6	(4)	(3)			(2)	(1)	
7	(4)	(3)				(2)	(1)
8		(4)	(3)				(2)
9			(4)	(3)			
10				(4)	(3)		
11					(4)	(3)	
12						(4)	(3)
13							(4)
14							

data forwarding

https://www.pollev.com/hungweitseng close in 1:30

How many of data hazards w/ Data Forwarding?

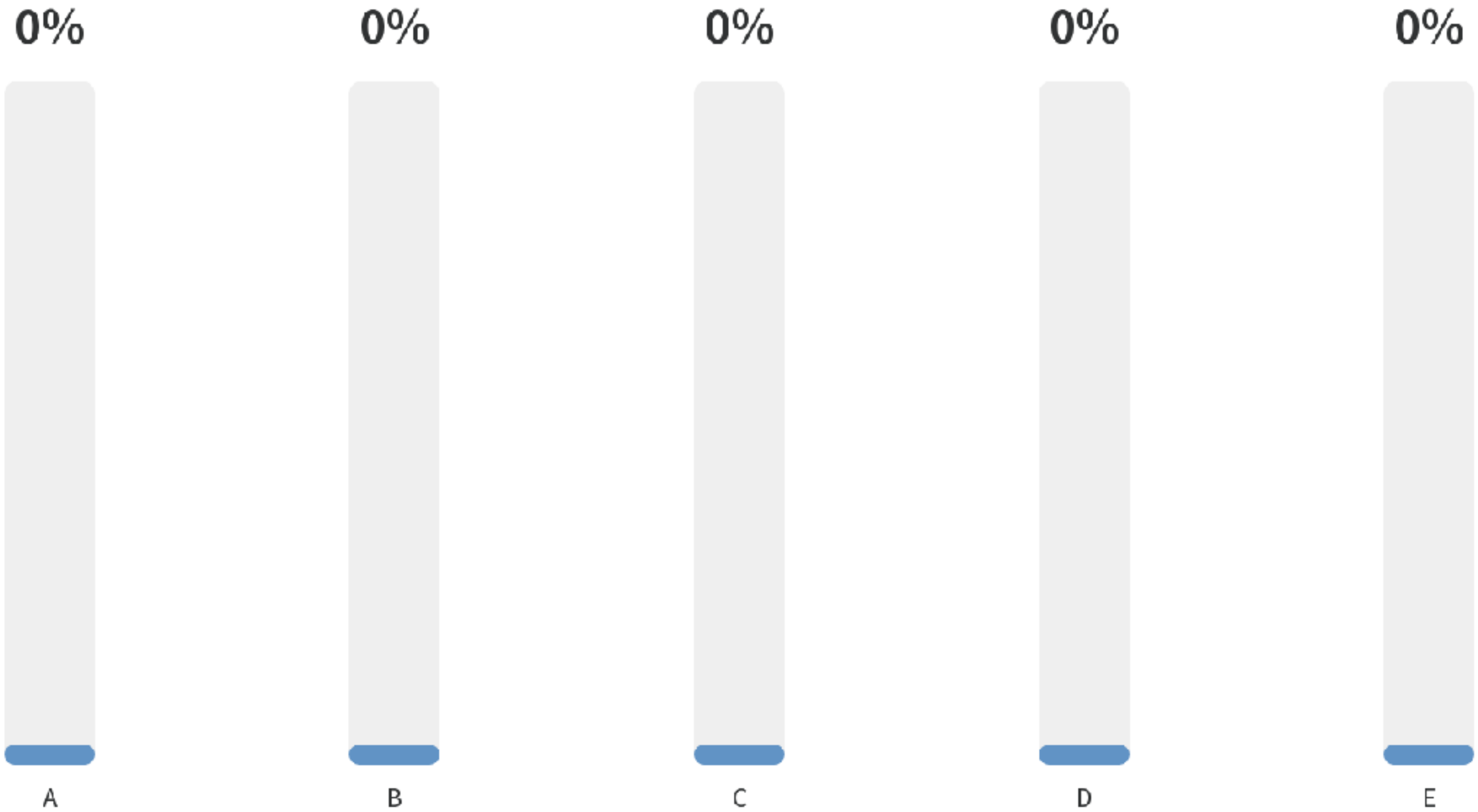
- How many pairs of data dependences in the following x86 instructions are still problematic with data forwarding if both a memory operation and an xorl take 4 cycles?

```
① movl    (%rdi), %eax
② xorl    (%rsi), %eax
③ movl    %eax, (%rdi)
④ xorl    (%rsi), %eax
⑤ movl    %eax, (%rsi)
⑥ xorl    %eax, (%rdi)
```

```
*a ^= *b;
*b ^= *a;
*a ^= *b;
```

A. 0
B. 1
C. 2
D. 3
E. 4

✓ 0



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

https://www.pollev.com/hungweitseng close in 1:30

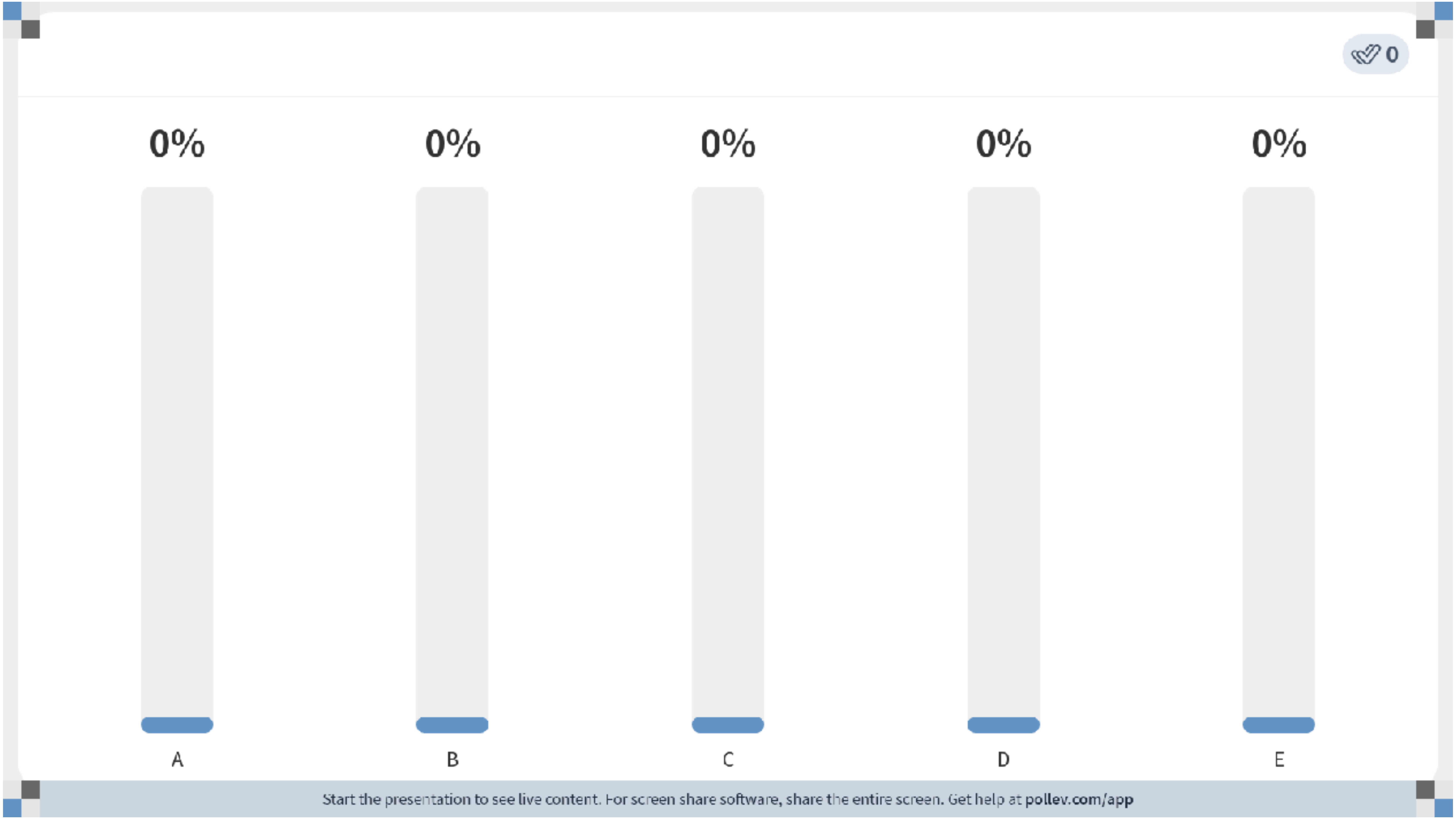
How many of data hazards w/ Data Forwarding?

- How many pairs of data dependences in the following x86 instructions are still problematic with data forwarding if both a memory operation and an xorl take 4 cycles?

```
*a ^= *b;
*b ^= *a;
*a ^= *b;
```

```
① movl    (%rdi), %eax
② xorl    (%rsi), %eax
③ movl    %eax, (%rdi)
④ xorl    (%rsi), %eax
⑤ movl    %eax, (%rsi)
⑥ xorl    %eax, (%rdi)
```

A. 0
B. 1
C. 2
D. 3
E. 4



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

How many of data hazards w/ Data Forwarding?

- How many pairs of data dependences in the following x86 instructions are still problematic with data forwarding if both a memory operation and an xorl take 4 cycles?

```
① movl    (%rdi), %eax
② xorl    (%rsi), %eax
③ movl    %eax, (%rdi)
④ xorl    (%rsi), %eax
⑤ movl    %eax, (%rsi)
⑥ xorl    %eax, (%rdi)
```

A. 0

B. 1

C. 2

[✓ D. 3]

E. 4

	IF	ID	ALU/BR/M1	M2	M3	M4/XORL	WB
1	(1)						
2	(2)	(1)					
3	(3)	(2)	(1)				
4	(3)	(2)		(1)			
5	(3)	(2)			(1)		
6	(3)	(2)				(1)	
7	(4)	(3)	(2)				(1)
8	(4)	(3)		(2)			
9	(4)	(3)			(2)		
10	(4)	(3)				(2)	
11	(5)	(4)	(3)				(2)
12	(6)	(5)	(4)	(3)			
13	(6)	(5)		(4)	(3)		
14	(6)	(5)			(4)	(3)	
15	(6)	(5)				(4)	(3)
16		(6)	(5)				(4)
17			(6)	(5)			
18				(6)	(5)		
19					(6)	(5)	
20						(6)	(5)
21							(6)
22							

Another code example

```
for(i = 0; i < count; i++) {
    s += a[i];
}
```

```
.L3:
①       movl    (%rdi), %ecx
②       addl    %ecx, %eax
③       addq    $4, %rdi
④       cmpq    %rdx, %rdi
⑤       jne     .L3
⑥       ret
```

	IF	ID	ALU/BR/M1	M2	M3	M4/XORL	WB
1	(1)						
2	(2)	(1)					
3	(3)	(2)	(1)				
4	(3)	(2)		(1)			
5	(3)	(2)			(1)		
6	(3)	(2)				(1)	
7	(4)	(3)	(2)				(1)
8	(5)	(4)	(3)	(2)			
9		(5)	(4)	(3)	(2)		
10			(5)	(4)	(3)	(2)	
11				(5)	(4)	(3)	(2)
12					(5)	(4)	(3)
13						(5)	(4)

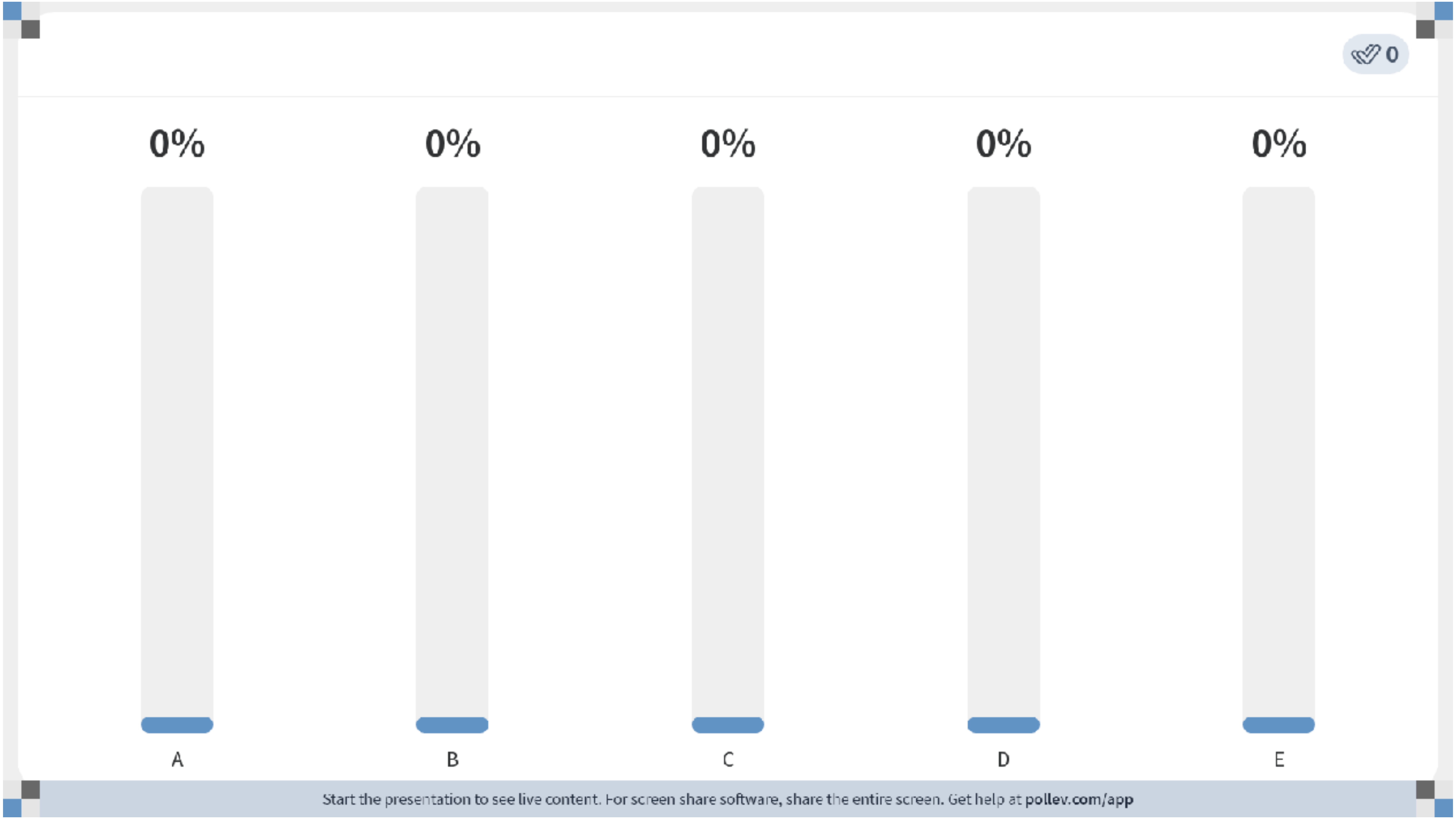
https://www.pollev.com/hungweitseng close in 1:30

The effect of code optimization

- By reordering which pair of the following instruction stream can we eliminate all stalls without affecting the correctness of the code?

```
① movl    (%rdi), %ecx
② addl    %ecx, %eax
③ addq    $4, %rdi
④ cmpq    %rdx, %rdi
⑤ jne     .L3
⑥ ret
```

A. (1) & (2)
B. (2) & (3)
C. (3) & (4)
D. (4) & (5)
E. None of the pairs can be reordered



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

https://www.pollev.com/hungweitseng close in 1:30

The effect of code optimization

- By reordering which pair of the following instruction stream can we eliminate all stalls without affecting the correctness of the code?

```
① movl    (%rdi), %ecx
② addl    %ecx, %eax
③ addq    $4, %rdi
④ cmpq    %rdx, %rdi
⑤ jne     .L3
⑥ ret
```

A. (1) & (2)

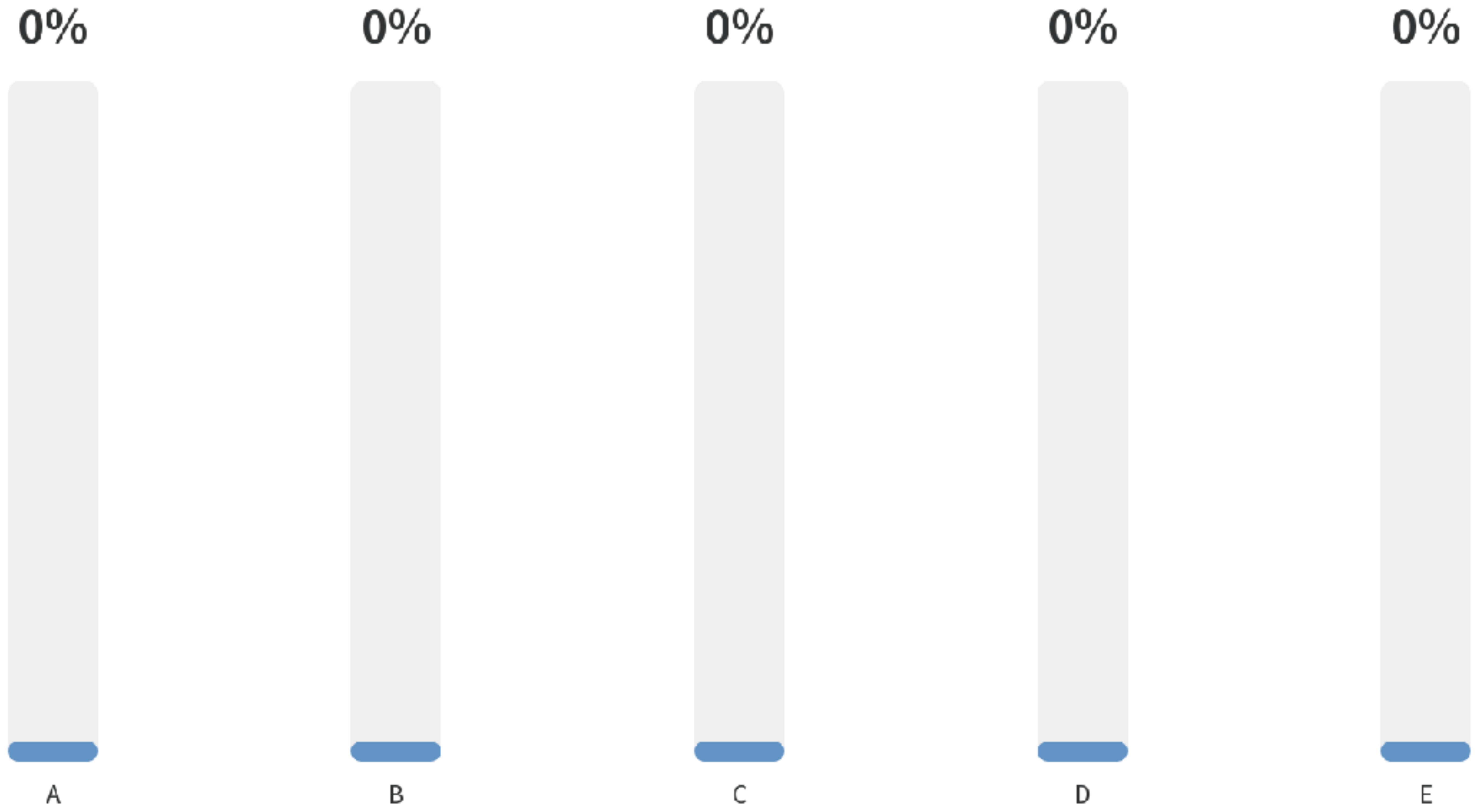
B. (2) & (3)

C. (3) & (4)

D. (4) & (5)

E. None of the pairs can be reordered

✓ 0



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

The effect of code optimization

- By reordering which pair of the following instruction stream can we eliminate all stalls without affecting the correctness of the code?

```
① movl    (%rdi), %ecx
② addl    %ecx, %eax
③ addq    $4, %rdi
④ cmpq    %rdx, %rdi
⑤ jne     .L3
⑥ ret
```

A. (1) & (2)

[✓ B. (2) & (3)]

C. (3) & (4)

D. (4) & (5)

E. None of the pairs can be reordered

Compiler optimization

```
    for(i = 0; i < count; i++) {
        s += a[i];
    }
.L3:
①       movl    (%rdi), %ecx
②       addq    $4, %rdi
③       addl    %ecx, %eax
④       cmpq    %rdx, %rdi
⑤       jne     .L3
⑥       ret
```

	IF	ID	ALU/BR/M1	M2	M3	M4/XORL	WB
1	(1)						
2	(2)	(1)					
3	(3)	(2)	(1)				
4	(3)	(2)	(2)	(1)			
5	(3)	(2)		(2)	(1)		
6	(4)	(2)			(2)	(1)	
7	(5)	(4)	(3)			(2)	(1)
8		(5)	(4)	(3)			(2)
9			(5)	(4)	(3)		
10				(5)	(4)	(3)	
11					(5)	(4)	(3)
12						(5)	(4)
13							(5)

Compiler optimization

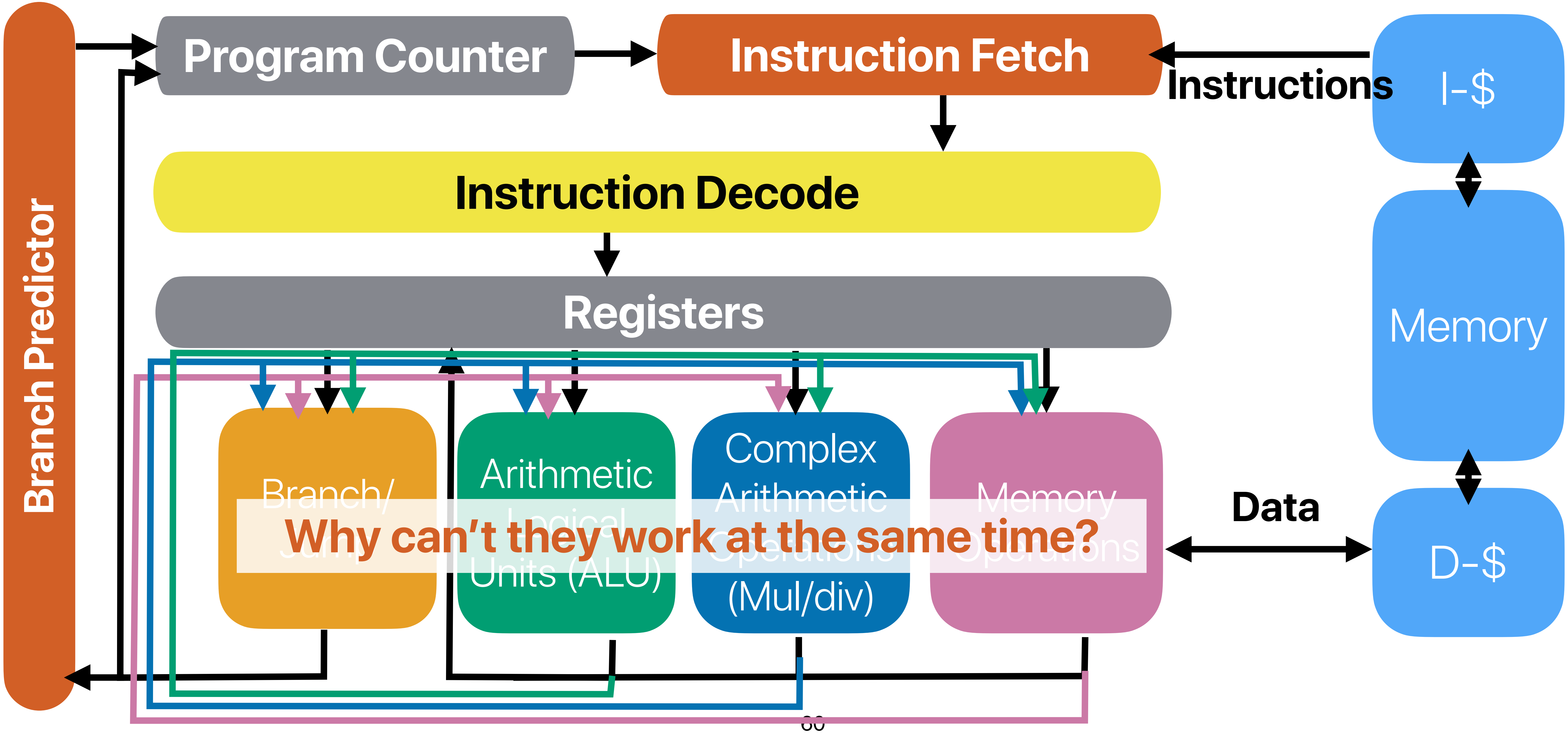
```
    for(i = 0; i < count; i++) {
        s += a[i];
    }
.L3:
①       movl    (%rdi), %ecx
②       addq    $4, %rdi
③       addl    %ecx, %eax
④       cmpq    %rdx, %rdi
⑤       jne     .L3
⑥       ret
```

	IF	ID	ALU/BR/M1	M2	M3	M4/XORL	WB
1	(1)						
2	(2)	(1)					
3	(3)	(2)	(1)				
4	(3)	(2)	(2)	(1)			
5	(3)	(2)		(2)	(1)		
6	(4)	(2)			(2)	(1)	
7	(5)	(4)	(3)			(2)	(1)
8		(5)	(4)	(3)			(2)
9			(5)	(4)	(3)		
10				(5)	(4)	(3)	
11					(5)	(4)	(3)
12						(5)	(4)
13							(5)

addq is not depending on movl and ALU is free! can we execute them together?

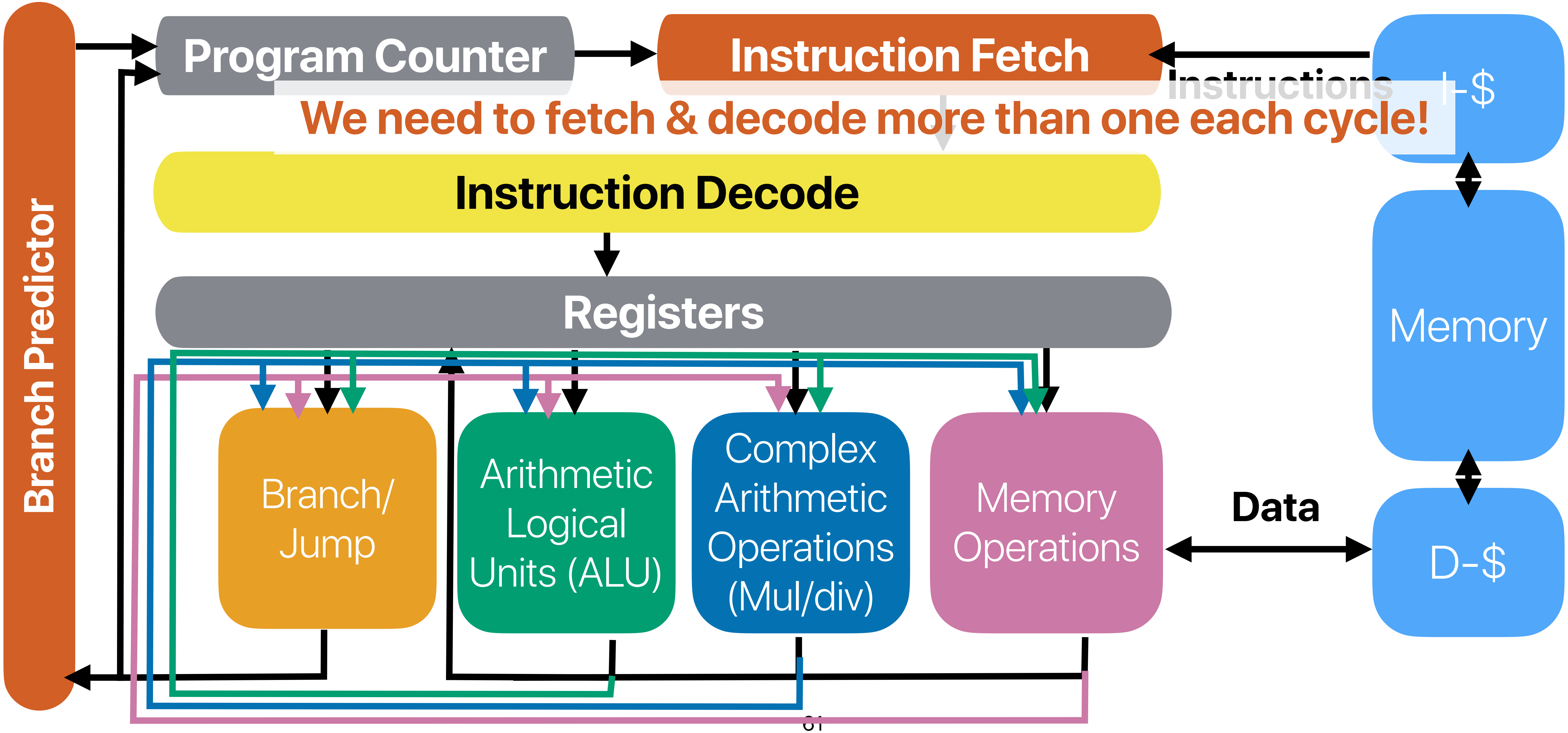
If CPI==1 the limitation?

Data "forwarding"



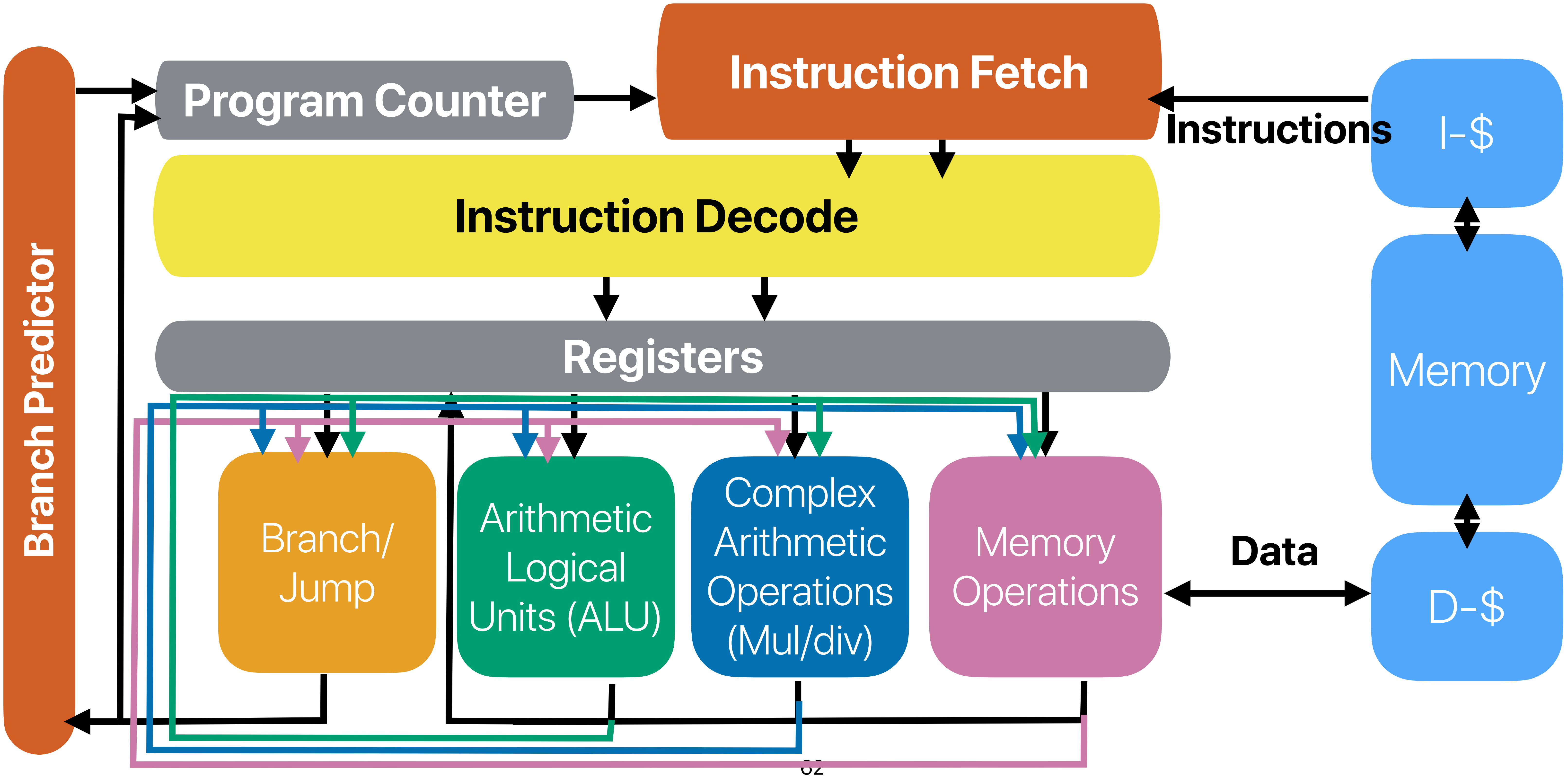
Why can't they work at the same time?

Data “forwarding”



We need to fetch & decode more than one each cycle!

Super Scalar



Super Scalar

Superscalar

- Since we have many functional units now, we should fetch/decode more instructions each cycle so that we can have more instructions to issue!
- Super-scalar: fetch/decode/issue more than one instruction each cycle
  - **Fetch width:** how many instructions can the processor fetch/decode each cycle
  - **Issue width**: how many instructions can the processor issue each cycle
- The theoretical CPI should now be

$$\frac{1}{min(issue\ width, fetch\ width, decode\ width)}$$

Superscalar: fetch/issue width == 2, theoretical CPI = 0.5

```
    for(i = 0; i < count; i++) {
        s += a[i];
    }
.L3:
①       movl    (%rdi), %ecx
②       addq    $4, %rdi
③       addl    %ecx, %eax
④       cmpq    %rdx, %rdi
⑤       jne     .L3
⑥       ret
```

	IF	ID	M1/ALU/BR	M2	M3	M4	WB
1	(1) (2)						
2	(3)(4)	(1) (2)					
3	(5)	(3)(4)	(1)(2)				
4	(5)	(3)(4)		(1)(2)			
5	(5)	(3)(4)			(1)(2)		
6	(5)	(3)(4)				(1)(2)	
7		(5)	(3)(4)				(1)(2)
8			(5)	(3)(4)			
9				(5)	(3)(4)		
10					(5)	(3)(4)	
11						(5)	(3)(4)
12							(5)

Everything we ne
for (4) is ready he
Why can't we
execute it?

If we loop many times (assume perfect predictor)

```
①  movl    (%rdi), %ecx
②  addq    $4, %rdi
③  addl    %ecx, %eax
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx
⑦  addq    $4, %rdi
⑧  addl    %ecx, %eax
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx
⑫  addq    $4, %rdi
⑬  addl    %ecx, %eax
⑭  cmpq    %rdx, %rdi
⑮  jne     .L3
```

	IF	ID	M1/ALU/BR	M2	M3	M4	WB
1	(1) (2)						
2	(3)(4)	(1) (2)					
3	(5)(6)	(3)(4)	(1)(2)				
4	(5)(6)	(3)(4)		(1)(2)			
5	(5)(6)	(3)(4)			(1)(2)		
6	(5)(6)	(3)(4)				(1)(2)	
7	(7)(8)	(5)(6)	(3)(4)				(1)(2)
8	(9)(10)	(7)(8)	(5)(6)	(3)(4)			
9	(9)(10)	(8)	(7)	(5)(6)	(3)(4)		
10	(9)(10)	(8)		(7)	(5)(6)	(3)(4)	
11	(9)(10)	(8)			(7)	(5)(6)	(3)(4)
12	(11)(12)	(9)(10)	(8)			(7)	(5)(6)
	(11)(12)	(10)	(9)	(8)			(7)
		(11)(12)	(10)	(9)	(8)		
			(11)(12)	(10)	(9)	(8)	
				(11)(12)	(10)	(9)	(8)
					(11)	(10)	(9)

Everything we need for (4) is ready here! Why can't we execute it?

Why can't I start loading (6) & (11)?

Limitations of Compiler Optimizations

- If the hardware (e.g., pipeline changes), the same compiler optimization may not be that helpful
- The compiler can only optimize on static instructions, but cannot optimize dynamic instruction
  - Compiler cannot predict branches
  - Compiler does not know if cache has the data/instructions

What do you need to execution an instruction?

- Whenever the instruction is decoded — put decoded instruction somewhere
- Whenever the inputs are ready — **all data dependencies are resolved**
- Whenever the target functional unit is available

Dynamic instruction scheduling/ Out-of-order (OoO) execution

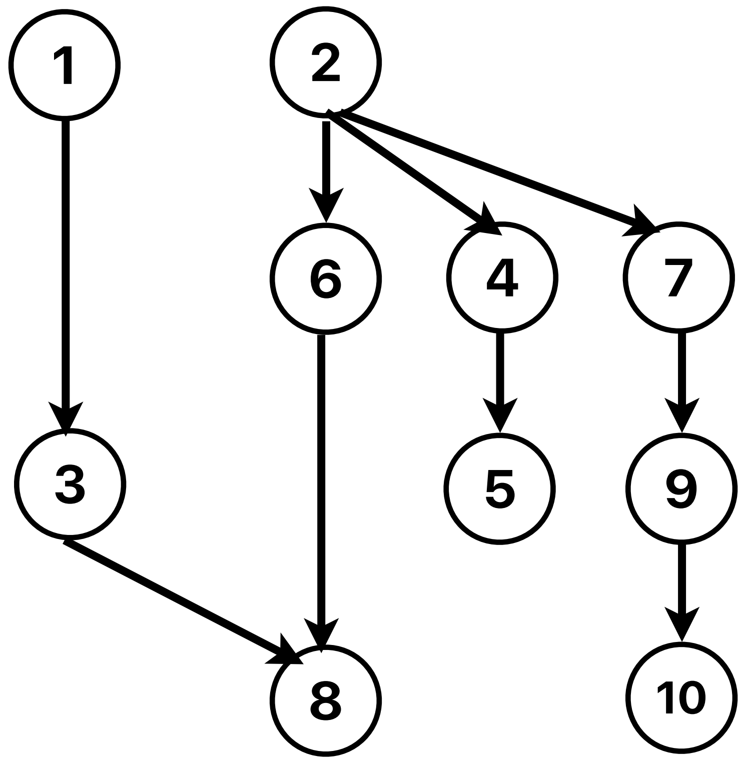
What do you need to execution an instruction?

- Whenever the instruction is decoded — put decoded instruction somewhere
- Whenever the inputs are ready — **all data dependencies are resolved**
- Whenever the target functional unit is available

Scheduling instructions: based on data dependencies

- Draw the data dependency graph, put an arrow if an instruction depends on the other.

```
①  movl    (%rdi), %ecx
②  addq    $4, %rdi
③  addl    %ecx, %eax
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx
⑦  addq    $4, %rdi
⑧  addl    %ecx, %eax
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
```



- **In theory**, instructions without dependencies can be executed in parallel or out-of-order
- Instructions with dependencies can never be reordered

https://www.pollev.com/hungweitseng close in 1:30

If we can predict the future …

- Consider the following dynamic instructions:

```
① movl    (%rdi), %ecx
② addq    $4, %rdi
③ addl    %ecx, %eax
④ cmpq    %rdx, %rdi
⑤ jne     .L3
⑥ movl    (%rdi), %ecx
⑦ addq    $4, %rdi
⑧ addl    %ecx, %eax
⑨ cmpq    %rdx, %rdi
⑩ jne     .L3
```

Which of the following pair can we reorder without affecting the correctness if the **branch prediction is perfect**?

A. (1) and (2)

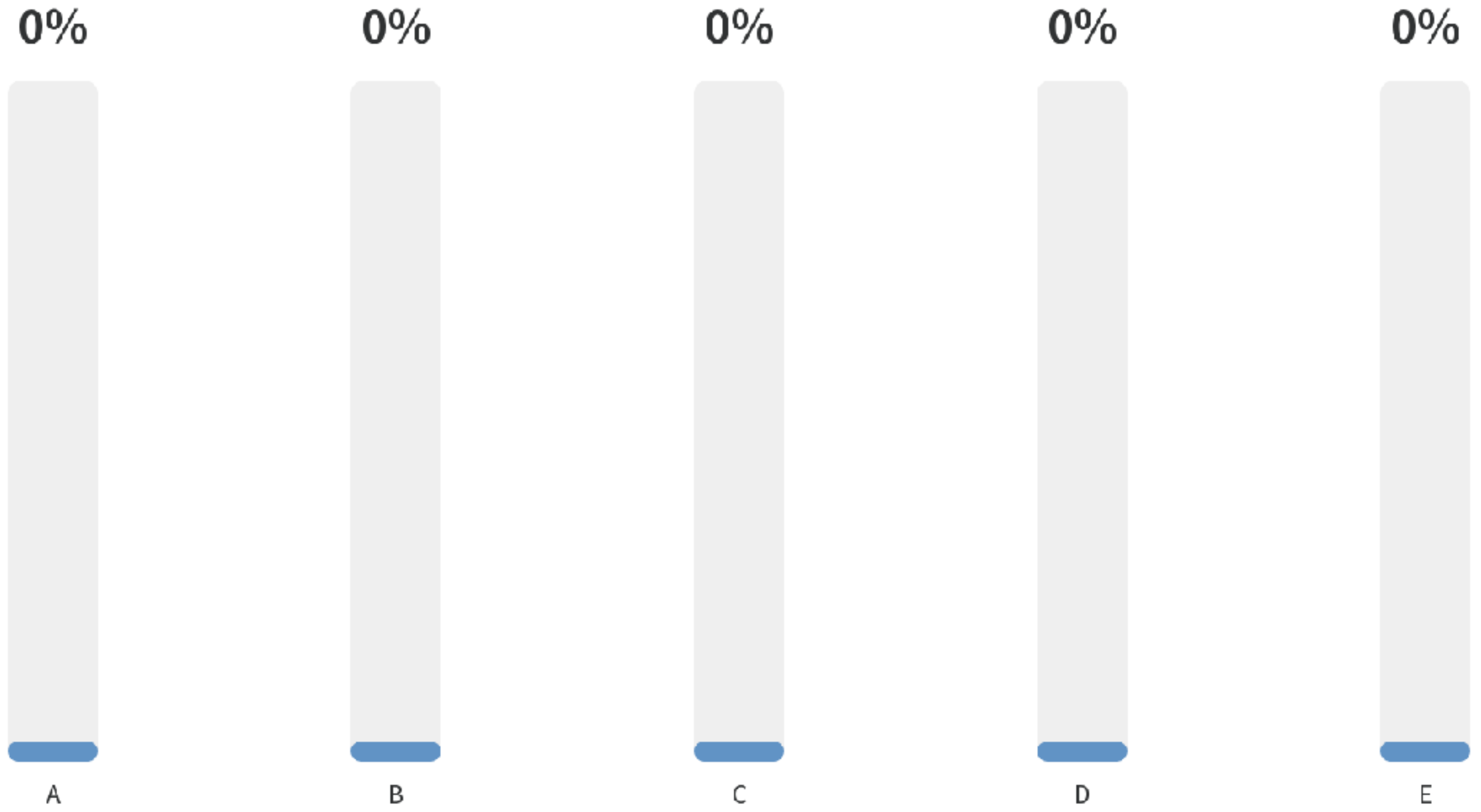
B. (3) and (4)

C. (3) and (6)

D. (4) and (7)

E. (6) and (7)

✓ 0



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

https://www.pollev.com/hungweitseng close in 1:30

If we can predict the future …

- Consider the following dynamic instructions:

```
① movl    (%rdi), %ecx
② addq    $4, %rdi
③ addl    %ecx, %eax
④ cmpq    %rdx, %rdi
⑤ jne     .L3
⑥ movl    (%rdi), %ecx
⑦ addq    $4, %rdi
⑧ addl    %ecx, %eax
⑨ cmpq    %rdx, %rdi
⑩ jne     .L3
```

Which of the following pair can we reorder without affecting the correctness if the **branch prediction is perfect**?

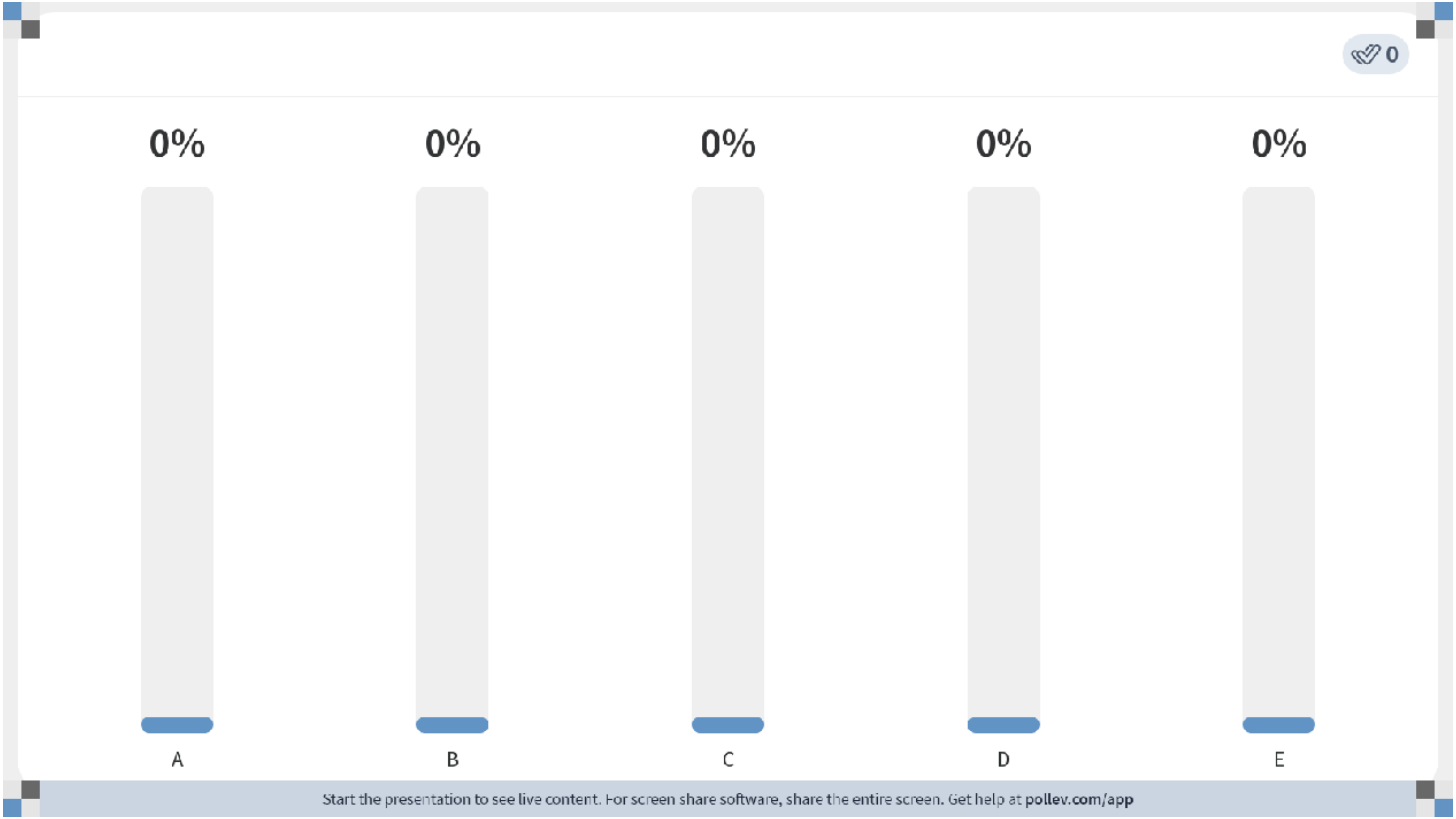
A. (1) and (2)

B. (3) and (4)

C. (3) and (6)

D. (4) and (7)

E. (6) and (7)



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

If we can predict the future …

- Consider the following dynamic instructions:

```
① movl    (%rdi), %ecx
② addq    $4, %rdi
③ addl    %ecx, %eax
④ cmpq    %rdx, %rdi
⑤ jne     .L3
⑥ movl    (%rdi), %ecx
⑦ addq    $4, %rdi
⑧ addl    %ecx, %eax
⑨ cmpq    %rdx, %rdi
⑩ jne     .L3
```

Can we use "branch prediction" to predict the future and reorder instructions across the branch?

Which of the following pair can we reorder without affecting the correctness if the **branch prediction is perfect**?

A. (1) and (2)

[✓ B. (3) and (4)]

C. (3) and (6)

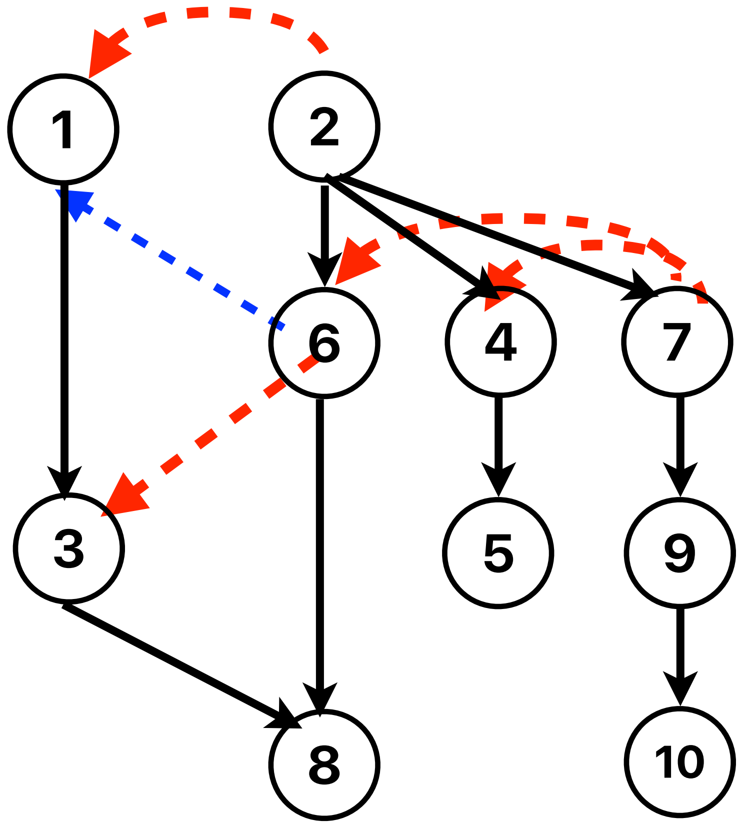
D. (4) and (7)

E. (6) and (7)

False dependencies

- We are still limited by **false dependencies**
- They are not “true” dependencies because they don’t have an arrow in data dependency graph
  - WAR (Write After Read): a later instruction overwrites the source of an earlier one
    - 2 and 1, 6 and 3, 7 and 4, 7 and 6
  - WAW (Write After Write): a later instruction overwrites the output of an earlier one
    - 6 and 1

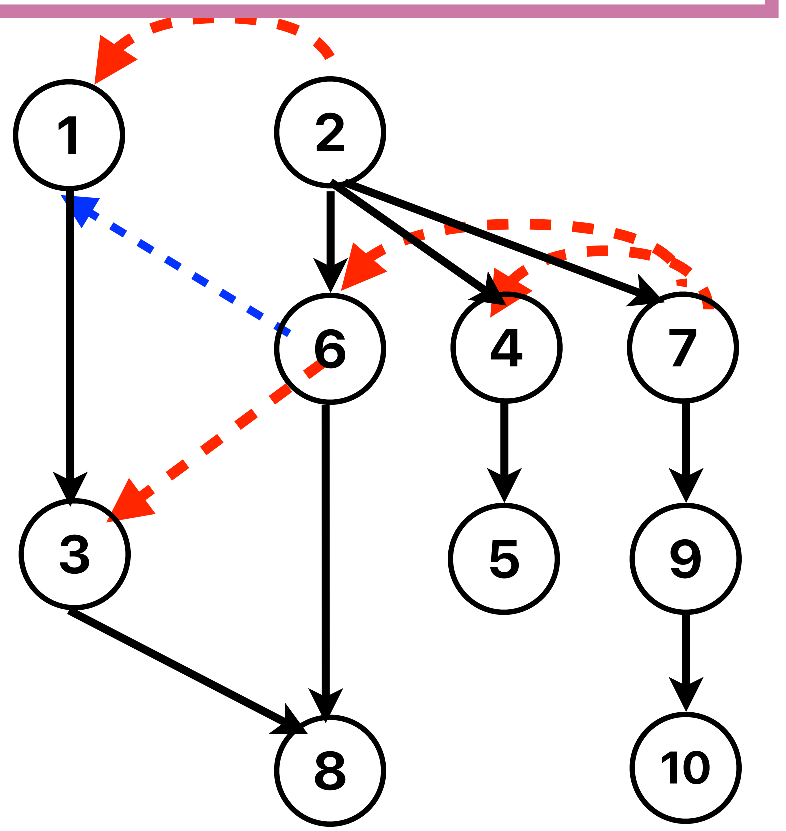
```
①   movl    (%rdi), %ecx
②   addq    $4, %rdi
③   addl    %ecx, %eax
④   cmpq    %rdx, %rdi
⑤   jne     .L3
⑥   movl    (%rdi), %ecx
⑦   addq    $4, %rdi
⑧   addl    %ecx, %eax
⑨   cmpq    %rdx, %rdi
⑩   jne     .L3
```



False dependencies

- We are still limited by **false dependencies**
- They are not "true" dependencies because they don't have an arrow in data dependency graph
  - WAR (Write After Read): a later instruction overwrites the source of an earlier one
    - 2 and 1, 6 and 3, 7 and 4, 7 and 6
  - WAW (Write After Write): a later instruction overwrites the output of an earlier one
    - 6 and 1

```
①  movl    (%rdi), %ecx
②  addq    $4, %rdi
③  addl    %ecx, %eax
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx
⑦  addq    $4, %rdi
⑧  addl    %ecx, %eax
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
```



Limitations of Compiler Optimizations

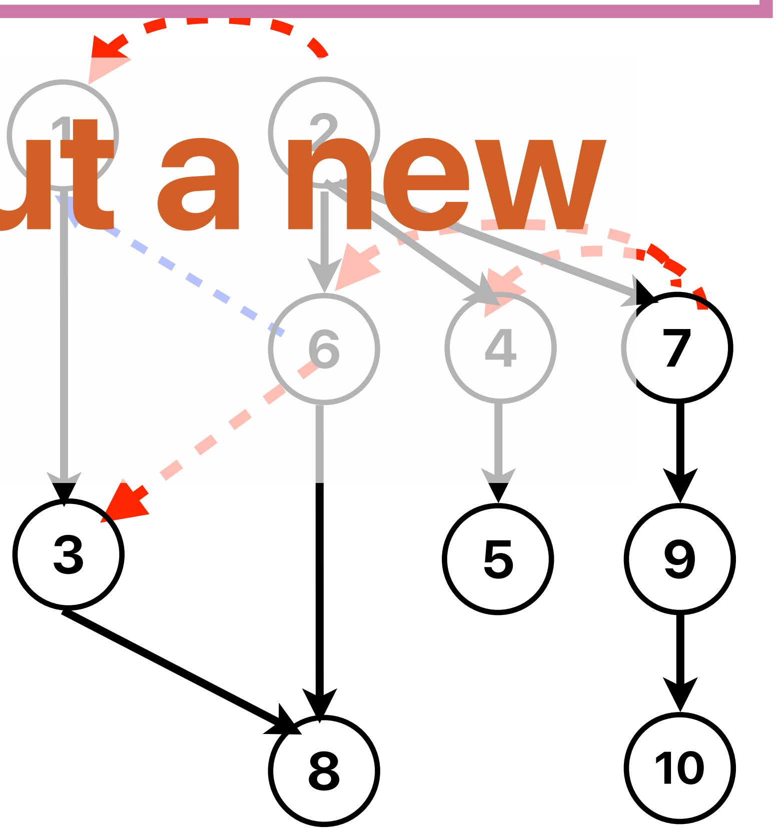
- If the hardware (e.g., pipeline changes), the same compiler optimization may not be that helpful
- The compiler can only optimize on static instructions, but cannot optimize dynamic instructions
- Compilers are limited by the registers an ISA provides

Recap: False dependencies

- We are still limited by **false dependencies**
- They are not "true" dependencies because they don't have an arrow in data dependency graph
  - WAR (Write After Read): a later instruction overwrites the source of an earlier one
    - 2 and 1, 6 and 3, 7 and 4, 7 and 6
  - WAW (Write After Write): a later instruction overwrites the output of an earlier one
    - 6 and 1

We need to give each output a new register!!!

```
①  movl    (%rdi), %ecx
②  addq    $4, %rdi
③  addl    %ecx, %eax
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx
⑦  addq    $4, %rdi
⑧  addl    %ecx, %eax
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
```



What if we can use more registers...

```
① movl    (%rdi), %ecx
② addq    $4, %rdi
③ addl    %ecx, %eax
④ cmpq    %rdx, %rdi
⑤ jne     .L3
⑥ movl    (%rdi), %ecx
⑦ addq    $4, %rdi
⑧ addl    %ecx, %eax
⑨ cmpq    %rdx, %rdi
⑩ jne     .L3
```

```
① movl    (%rdi), %ecx
② addq    $4, %rdi, %t0
③ addl    %ecx, %eax, %t1
④ cmpq    %rdx, %t0
⑤ jne     .L3
⑥ movl    (%t0), %t2
⑦ addq    $4, %t0, %t3
⑧ addl    %t1, %t2, %t4
⑨ cmpq    %rdx, %t3
⑩ jne     .L3
```

All false dependencies are gone!!!

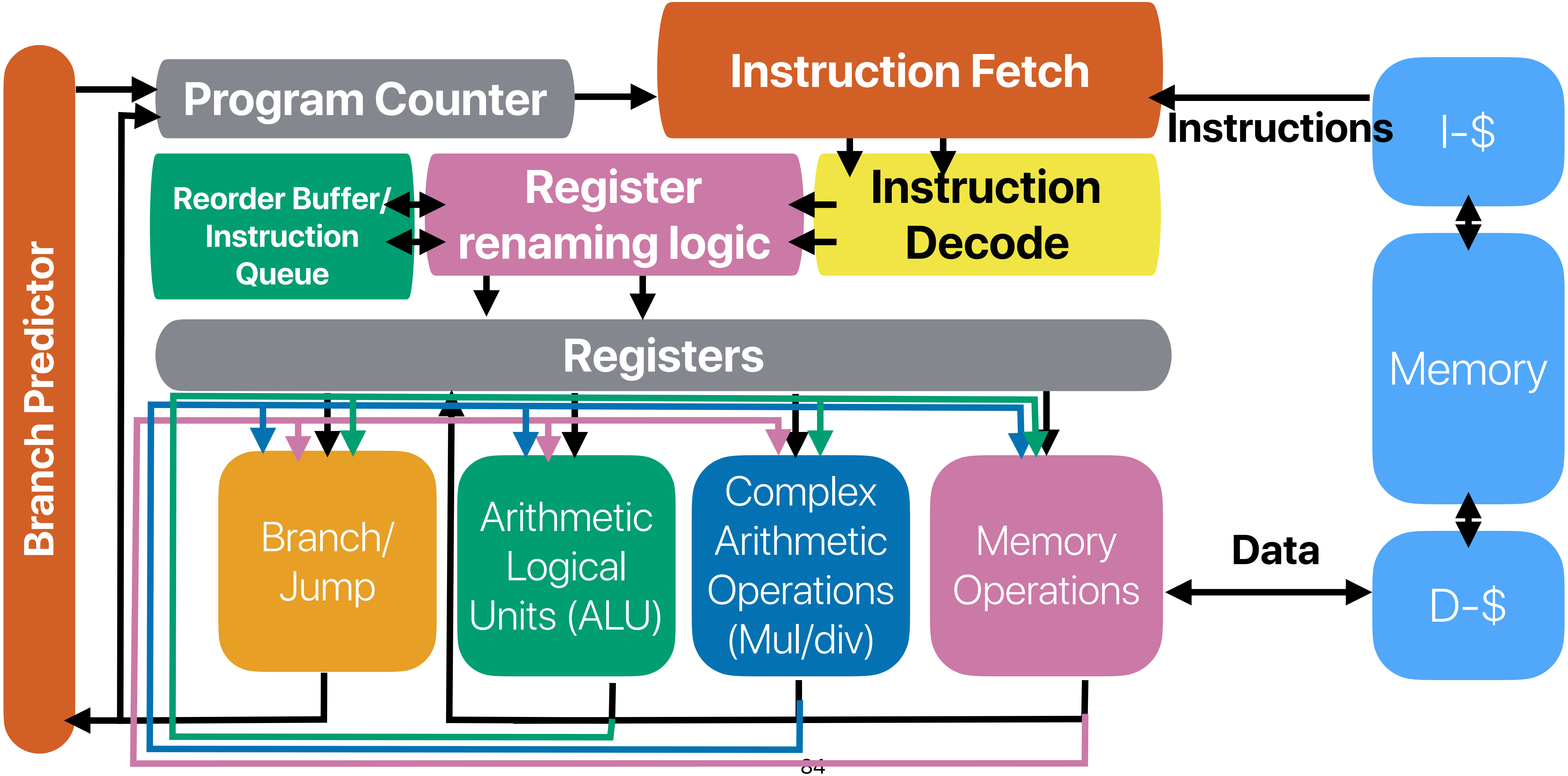
Register renaming + speculative execution

- K. C. Yeager, “The Mips R10000 superscalar microprocessor,” in IEEE Micro, vol. 16, no. 2, pp. 28-41, April 1996.

Speculative Execution

- Exceptions (e.g. divided by 0, page fault) may occur anytime
  - A later instruction cannot write back its own result otherwise the architectural states won't be correct
- Hardware can schedule instruction across branch instructions with the help of branch prediction
  - Fetch instructions according to the branch prediction
  - However, branch predictor can never be perfect
- Execute instructions across branches
  - Speculative execution: execute an instruction before the processor know if we need to execute or not
  - Execute an instruction all operands are ready (the values of depending physical registers are generated)
  - Store results in **reorder buffer** before the processor knows if the instruction is going to be executed or not.

Register renaming



Register renaming

```
①  movl    (%rdi), %ecx
②  addq    $4, %rdi
③  addl    %ecx, %eax
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx
⑦  addq    $4, %rdi
⑧  addl    %ecx, %eax
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx
⑫  addq    $4, %rdi
⑬  addl    %ecx, %eax
⑭  cmpq    %rdx, %rdi
⑮  jne     .L3
```

2-issue: Only 2 of them can have instructions at the same cycle

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4											
5											
6											
7											
8											
9											
10											
11											

	Physical Register
eax	
ecx	
rdi	
rdx	

	Valid	Value	In use		Valid	Value	In use
P1				P6			
P2				P7			
P3				P8			
P4				P9			
P5				P10			

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl    (%rdi), %ecx → P1
②  addq    $4, %rdi → P2
③  addl    %ecx, %eax
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx
⑦  addq    $4, %rdi
⑧  addl    %ecx, %eax
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx
⑫  addq    $4, %rdi
⑬  addl    %ecx, %eax
⑭  cmpq    %rdx, %rdi
⑮  jne     .L3
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4		(5)(6)	(3)(4)	(1)				(2)			
5											
6											
7											
8											
9											
10											
11											

	Physical Register
eax	
ecx	P1
rdi	P2
rdx	

	Valid	Value	In use		Valid	Value	In use
P1	0		1	P6			
P2	0		1	P7			
P3				P8			
P4				P9			
P5				P10			

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl    (%rdi), %ecx → P1
②  addq    $4, %rdi→ P2
③  addl    %ecx, %eax→ P3
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx → P4
⑦  addq    $4, %rdi
⑧  addl    %ecx, %eax
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx
⑫  addq    $4, %rdi
⑬  addl    %ecx, %eax
⑭  cmpq    %rdx, %rdi
⑮  jne     .L3
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4	(7)(8)	(5)(6)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6											
7											
8											
9											
10											
11											

(4) is now executing before (3)!

	Physical Register
eax	
ecx	P1
rdi	P2
rdx	P4

	Valid	Value	In use		Valid	Value	In use
P1	0		1	P6			
P2	1		1	P7			
P3	0		1	P8			
P4	0		1	P9			
P5				P10			

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl    (%rdi), %ecx → P1
②  addq    $4, %rdi→ P2
③  addl    %ecx, %eax→ P3
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx → P4
⑦  addq    $4, %rdi → P5
⑧  addl    %ecx, %eax → P6
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx
⑫  addq    $4, %rdi
⑬  addl    %ecx, %eax
⑭  cmpq    %rdx, %rdi
⑮  jne     .L3
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4	(7)(8)	(5)(6)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7											
8											
9											
10											
11											

	Physical Register
eax	P6
ecx	P1
rdi	P5
rdx	P4

	Valid	Value	In use		Valid	Value	In use
P1	0		1	P6	0		1
P2	1		1	P7			
P3	0		1	P8			
P4	0		1	P9			
P5	0		1	P10			

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl    (%rdi), %ecx → P1
②  addq    $4, %rdi→ P2
③  addl    %ecx, %eax→ P3
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx → P4
⑦  addq    $4, %rdi → P5
⑧  addl    %ecx, %eax  → P6
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx
⑫  addq    $4, %rdi
⑬  addl    %ecx, %eax
⑭  cmpq    %rdx, %rdi
⑮  jne     .L3
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4	(7)(8)	(5)(6)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9) (10)		(6)		(1)	(7)			(2)(4)(5)
8											
9											
10											
11											

	Physical Register
eax	P6
ecx	P1
rdi	P5
rdx	P4

	Valid	Value	In use		Valid	Value	In use
P1	0		1	P6	0		1
P2	1		1	P7			
P3	0		1	P8			
P4	0		1	P9			
P5	0		1	P10			

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl   (%rdi), %ecx → P1
②  addq   $4, %rdi → P2
③  addl   %ecx, %eax → P3
④  cmpq   %rdx, %rdi
⑤  jne    .L3
⑥  movl   (%rdi), %ecx → P4
⑦  addq   $4, %rdi → P5
⑧  addl   %ecx, %eax → P6
⑨  cmpq   %rdx, %rdi
⑩  jne    .L3
⑪  movl   (%rdi), %ecx → P7
⑫  addq   $4, %rdi → P8
⑬  addl   %ecx, %eax
⑭  cmpq   %rdx, %rdi
⑮  jne    .L3
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4	(7)(8)	(5)(6)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9)(10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10)(11)(12)			(6)		(3)			(1)(2)(4)(5)(7)
9											
10											
11											

	Physical Register
eax	P6
ecx	P7
rdi	P8
rdx	P4

	Valid	Value	In use		Valid	Value	In use
P1	1		1	P6	0		1
P2	1		1	P7	0		1
P3	0		1	P8	0		1
P4	0		1	P9			
P5	1		1	P10			

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl   (%rdi), %ecx → P1
②  addq   $4, %rdi→ P2
③  addl   %ecx, %eax→ P3
④  cmpq   %rdx, %rdi
⑤  jne    .L3
⑥  movl   (%rdi), %ecx → P4
⑦  addq   $4, %rdi → P5
⑧  addl   %ecx, %eax → P6
⑨  cmpq   %rdx, %rdi
⑩  jne    .L3
⑪  movl   (%rdi), %ecx → P7
⑫  addq   $4, %rdi → P8
⑬  addl   %ecx, %eax→ P9
⑭  cmpq   %rdx, %rdi
⑮  jne    .L3
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4	(7)(8)	(5)(6)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9) (10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10) (11)(12)			(6)		(3)			~~(1)(2)~~(4)(5) (7)
9	(17)(18)	(15)(16)	(8)(10)(12) (13)(14)	(11)			(6)	(9)			(3)(4)(5)(7)
10											
11											

	Physical Register
eax	P9
ecx	P7
rdi	P8
rdx	P4

	Valid	Value	In use		Valid	Value	In use
P1	1		0	P6	0		1
P2	1		0	P7	0		1
P3	1		1	P8	0		1
P4	0		1	P9	0		1
P5	1		1	P10			

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl    (%rdi), %ecx → P1
②  addq    $4, %rdi→ P2
③  addl    %ecx, %eax→ P3
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx → P4
⑦  addq    $4, %rdi → P5
⑧  addl    %ecx, %eax → P6
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx → P7
⑫  addq    $4, %rdi → P8
⑬  addl    %ecx, %eax→ P9
⑭  cmpq    %rdx, %rdi
⑮  jne     .L3
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4	(7)(8)	(5)(6)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9) (10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10) (11)(12)			(6)		(3)			~~(1)(2)~~(4)(5) (7)
9	(17)(18)	(15)(16)	(8)(10)(12) (13)(14)	(11)			(6)	(9)			~~(3)(4)(5)~~(7)
10	(19)(20)	(17)(18)	(12)(13)(14) (15)(16)		(11)			(8)		(10)	(6)(7)(9)
11											

	Physical Register
eax	P9
ecx	P7
rdi	P8
rdx	P4

	Valid	Value	In use		Valid	Value	In use
P1	1		0	P6	0		1
P2	1		0	P7	0		1
P3	1		0	P8	0		1
P4	0		1	P9	0		1
P5	1		1	P10	0		1

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl    (%rdi), %ecx → P1
②  addq    $4, %rdi→ P2
③  addl    %ecx, %eax→ P3
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx → P4
⑦  addq    $4, %rdi → P5
⑧  addl    %ecx, %eax → P6
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx → P7
⑫  addq    $4, %rdi → P8
⑬  addl    %ecx, %eax→ P9
⑭  cmpq    %rdx, %rdi
⑮  jne     .L3
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4	(7)(8)	(5)(6)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9) (10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10) (11)(12)			(6)		(3)			~~(1)(2)~~(4)(5) (7)
9	(17)(18)	(15)(16)	(8)(10)(12) (13)(14)	(11)			(6)	(9)			~~(3)(4)(5)(7)~~
10	(19)(20)	(17)(18)	(12)(13)(14) (15)(16)		(11)			(8)		(10)	~~(6)(7)~~(9)
11		(19)(20)	(13)(14)(15) (16)(17)(18)			(11)		(12)			(8)(9)(10)

	Physical Register
eax	P6
ecx	P1
rdi	P5
rdx	P4

	Valid	Value	In use		Valid	Value	In use
P1	1		0	P6	1		1
P2	1		0	P7	0		1
P3	1		0	P8	0		1
P4	1		0	P9	0		1
P5	1		1	P10	0		1

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl   (%rdi), %ecx → P1
②  addq   $4, %rdi→ P2
③  addl   %ecx, %eax→ P3
④  cmpq   %rdx, %rdi
⑤  jne    .L3
⑥  movl   (%rdi), %ecx → P4
⑦  addq   $4, %rdi → P5
⑧  addl   %ecx, %eax → P6
⑨  cmpq   %rdx, %rdi
⑩  jne    .L3
⑪  movl   (%rdi), %ecx → P7
⑫  addq   $4, %rdi → P8
⑬  addl   %ecx, %eax→ P9
⑭  cmpq   %rdx, %rdi
⑮  jne    .L3
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4	(7)(8)	(5)(6)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9) (10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10) (11)(12)			(6)		(3)			~~(1)(2)~~(4)(5) (7)
9	(17)(18)	(15)(16)	(8)(10)(12) (13)(14)	(11)			(6)	(9)			~~(3)(4)(5)(7)~~
10	(19)(20)	(17)(18)	(12)(13)(14) (15)(16)		(11)			(8)		(10)	~~(6)(7)~~(9)
11		(19)(20)	(13)(14)(15) (16)(17)(18)			(11)		(12)			~~(8)(9)(10)~~
12			(13)(15)(17)(18) (19)(20)	(16)			(11)	(14)			(12)
13											
14											
15											

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl   (%rdi), %ecx → P1
②  addq   $4, %rdi→ P2
③  addl   %ecx, %eax→ P3
④  cmpq   %rdx, %rdi
⑤  jne    .L3
⑥  movl   (%rdi), %ecx → P4
⑦  addq   $4, %rdi → P5
⑧  addl   %ecx, %eax → P6
⑨  cmpq   %rdx, %rdi
⑩  jne    .L3
⑪  movl   (%rdi), %ecx → P7
⑫  addq   $4, %rdi → P8
⑬  addl   %ecx, %eax→ P9
⑭  cmpq   %rdx, %rdi
⑮  jne    .L3
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4	(7)(8)	(5)(6)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9) (10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10) (11)(12)			(6)		(3)			~~(1)(2)~~(4)(5) (7)
9	(17)(18)	(15)(16)	(8)(10)(12) (13)(14)	(11)			(6)	(9)			~~(3)(4)(5)(7)~~
10	(19)(20)	(17)(18)	(12)(13)(14) (15)(16)		(11)			(8)		(10)	~~(6)(7)~~(9)
11		(19)(20)	(13)(14)(15) (16)(17)(18)			(11)		(12)			~~(8)(9)(10)~~
12			(13)(15)(17)(18) (19)(20)	(16)			(11)	(14)			(12)
13			(17)(18) (19)(20)		(16)			(13)		(15)	(11)(12)(14)
14											
15											

Register renaming

2-issue: Only 2 of them can have instructions at the same cycle

```
①  movl    (%rdi), %ecx → P1
②  addq    $4, %rdi → P2
③  addl    %ecx, %eax → P3
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx → P4
⑦  addq    $4, %rdi → P5
⑧  addl    %ecx, %eax → P6
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx → P7
⑫  addq    $4, %rdi → P8
⑬  addl    %ecx, %eax → P9
⑭  cmpq    %rdx, %rdi
⑮  jne     .L3
```

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4	(7)(8)	(5)(6)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9)(10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10)(11)(12)			(6)		(3)			~~(1)(2)~~(4)(5)(7)
9	(17)(18)	(15)(16)	(8)(10)(12)(13)(14)	(11)			(6)	(9)			~~(3)(4)(5)(7)~~
10	(19)(20)	(17)(18)	(12)(13)(14)(15)(16)		(11)			(8)		(10)	~~(6)(7)~~(9)
11		(19)(20)	(13)(14)(15)(16)(17)(18)			(11)		(12)			~~(8)(9)(10)~~
12			(13)(15)(17)(18)(19)(20)	(16)			(11)	(14)			(12)
13			(17)(18)(19)(20)		(16)			(13)		(15)	~~(11)(12)~~(14)
14						(16)		(17)			(13)(14)(15)
15											

Register renaming

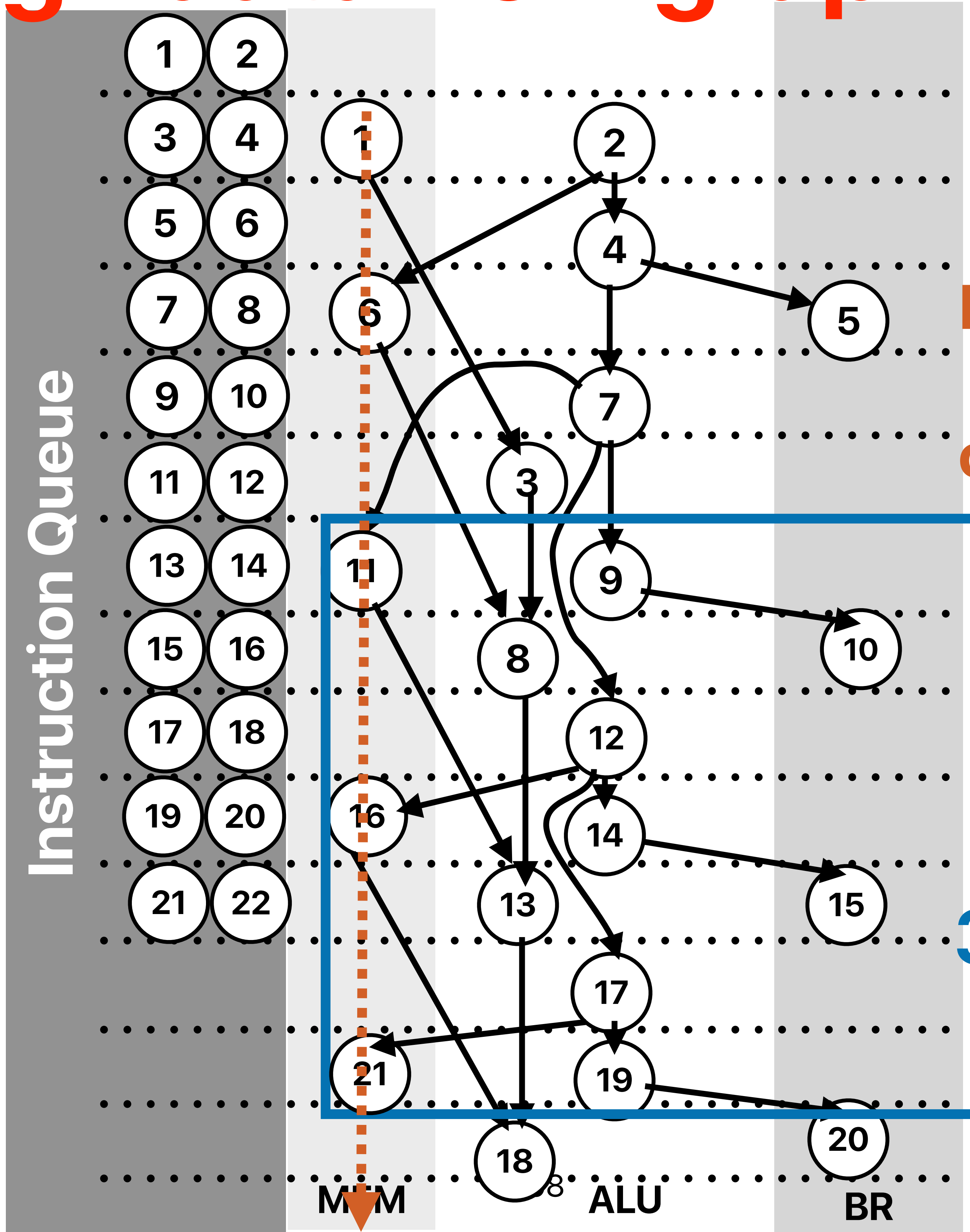
```
①  movl    (%rdi), %ecx → P1
②  addq    $4, %rdi→ P2
③  addl    %ecx, %eax→ P3
④  cmpq    %rdx, %rdi
⑤  jne     .L3
⑥  movl    (%rdi), %ecx → P4
⑦  addq    $4, %rdi → P5
⑧  addl    %ecx, %eax → P6
⑨  cmpq    %rdx, %rdi
⑩  jne     .L3
⑪  movl    (%rdi), %ecx → P7
⑫  addq    $4, %rdi → P8
⑬  addl    %ecx, %eax→ P9
⑭  cmpq    %rdx, %rdi
⑮  jne     .L3
```

2-issue: Only 2 of them can have instructions at the same cycle

	IF	ID	REN	M1	M2	M3	M4	ALU	MUL	BR	ROB
1	(1) (2)										
2	(3)(4)	(1) (2)									
3	(5)(6)	(3)(4)	(1) (2)								
4	(7)(8)	(5)(6)	(3)(4)	(1)				(2)			
5	(9)(10)	(7)(8)	(3)(5)(6)		(1)			(4)			(2)
6	(11)(12)	(9)(10)	(3)(7)(8)	(6)		(1)				(5)	(2)(4)
7	(13)(14)	(11)(12)	(3)(8)(9) (10)		(6)		(1)	(7)			(2)(4)(5)
8	(15)(16)	(13)(14)	(8)(9)(10) (11)(12)			(6)		(3)			~~(1)(2)~~(4)(5) (7)
9	(17)(18)	(15)(16)	(8)(10)(12) (13)(14)	(11)			(6)	(9)			~~(3)(4)(5)(7)~~
10	(19)(20)	(17)(18)	(12)(13)(14) (15)(16)		(11)			(8)		(10)	~~(6)(7)~~(9)
11		(19)(20)	(13)(14)(15) (16)(17)(18)			(11)		(12)			~~(8)(9)(10)~~
12			(13)(15)(17)(18) (19)(20)	(16)			(11)	(14)			(12)
13			(17)(18) (19)(20)		(16)			(13)		(15)	~~(11)(12)~~(14)
14						(16)		(17)			~~(13)(14)(15)~~
15							(16)	(19)			(17)

Through data flow graph analysis

```
① movl (%rdi), %ecx
② addq $4, %rdi
③ addl %ecx, %eax
④ cmpq %rdx, %rdi
⑤ jne  .L3
⑥ movl (%rdi), %ecx
⑦ addq $4, %rdi
⑧ addl %ecx, %eax
⑨ cmpq %rdx, %rdi
⑩ jne  .L3
⑪ movl (%rdi), %ecx
⑫ addq $4, %rdi
⑬ addl %ecx, %eax
⑭ cmpq %rdx, %rdi
⑮ jne  .L3
⑯ movl (%rdi), %ecx
⑰ addq $4, %rdi
⑱ addl %ecx, %eax
⑲ cmpq %rdx, %rdi
⑳ jne  .L3
㉑ movl (%rdi), %ecx
```



Execution time is determined by the "critical path" composed by 1, 6, 11, ..., 1+5n

3 cycles every iteration

$$\text{CPI} = \frac{3}{5} = 0.6!$$

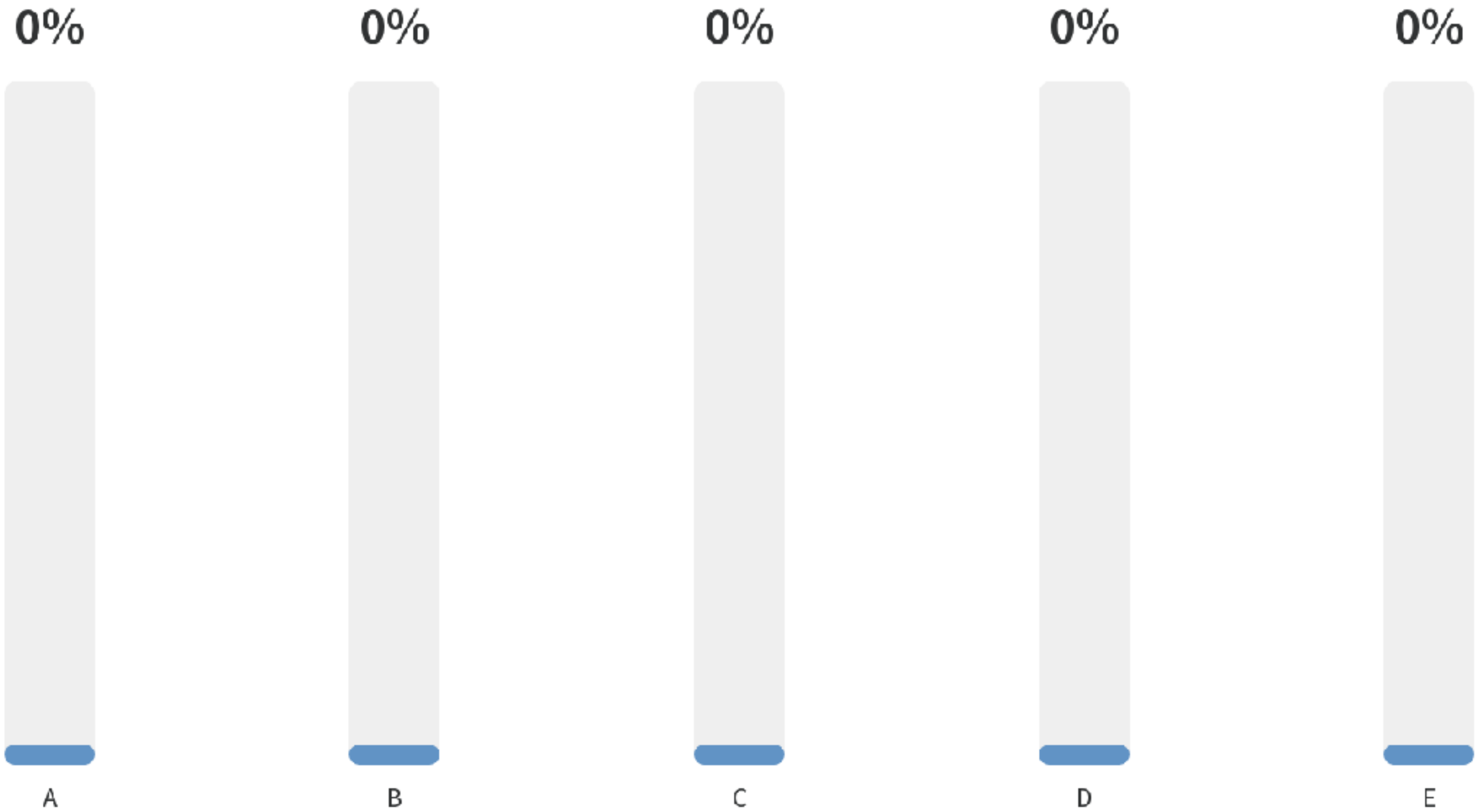
https://www.pollev.com/hungweitseng close in 1:30

What about "linked list"

- Assume the current PC is already at instruction (1) and this linked list has only three nodes. This processor can fetch and issue 2 instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.
  Which of the following C state of the code snippet determines the performance?

```
A.do {
B.     number_of_nodes++;
C.     current = current->next;
D.} while ( current != NULL );
```

0



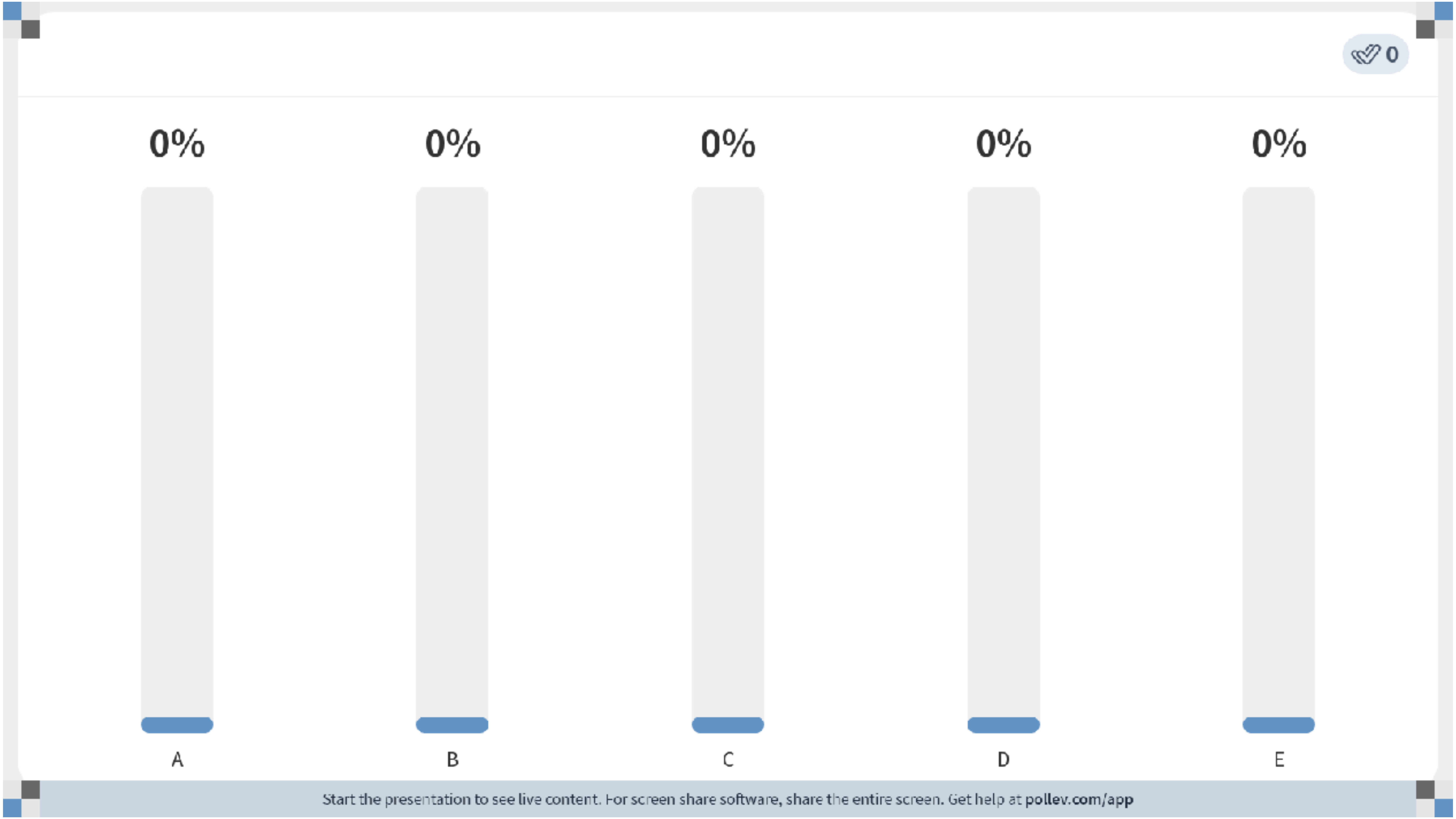
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

https://www.pollev.com/hungweitseng close in 1:30

What about "linked list"

- Assume the current PC is already at instruction (1) and this linked list has only three nodes. This processor can fetch and issue 2 instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.
  Which of the following C state of the code snippet determines the performance?

```
A.do {
B.    number_of_nodes++;
C.    current = current->next;
D.} while ( current != NULL );
```

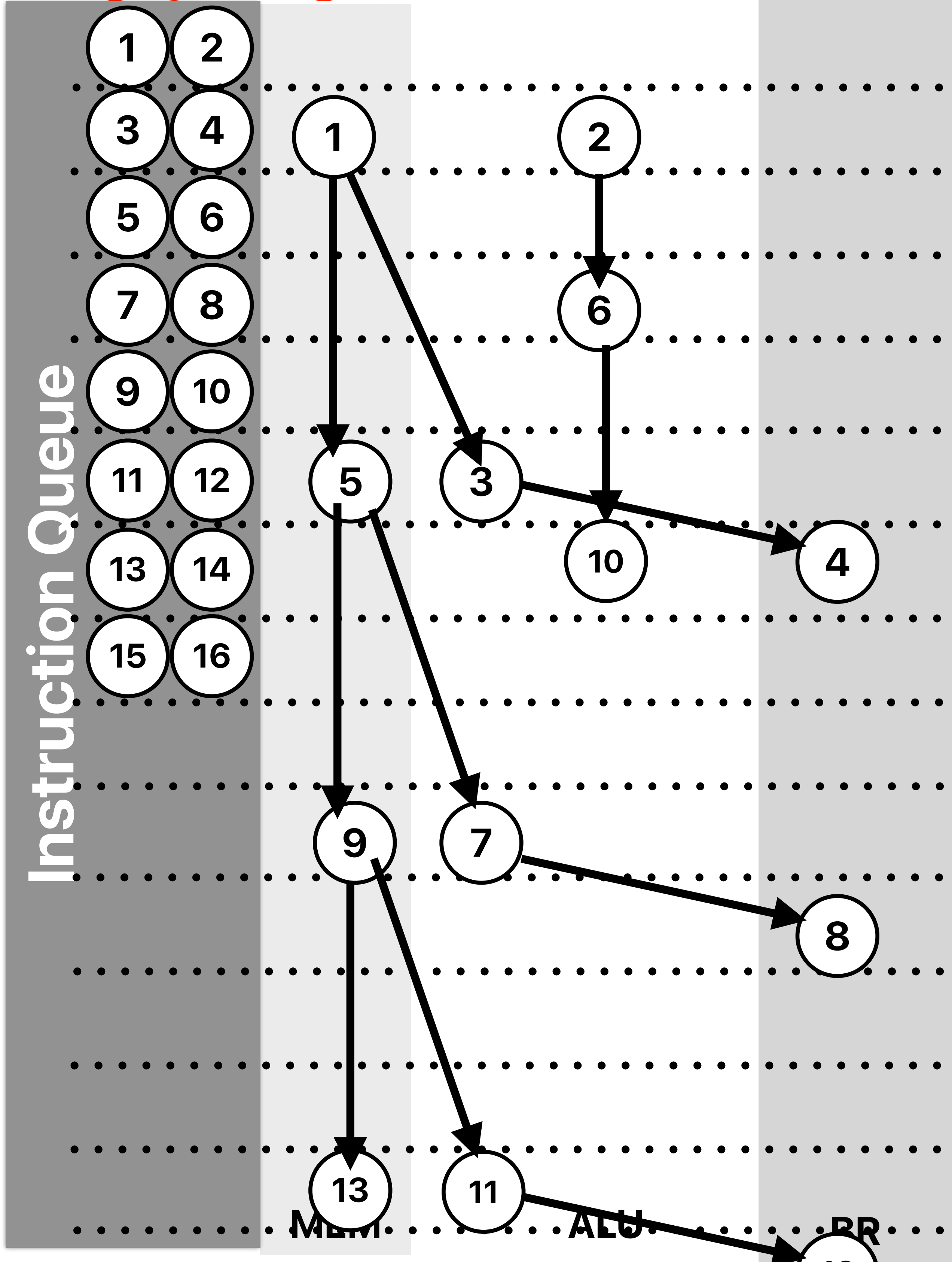


Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

What about "linked list"

Dynamic instructions

```
①  .L3:    movq    8(%rdi), %rdi
②          addl    $1, %eax
③          testq   %rdi, %rdi
④          jne     .L3
⑤  .L3:    movq    8(%rdi), %rdi
⑥          addl    $1, %eax
⑦          testq   %rdi, %rdi
⑧          jne     .L3
⑨  .L3:    movq    8(%rdi), %rdi
⑩          addl    $1, %eax
⑪          testq   %rdi, %rdi
⑫          jne     .L3
⑬  .L3:    movq    8(%rdi), %rdi
⑭          addl    $1, %eax
⑮          testq   %rdi, %rdi
⑯          jne     .L3
```



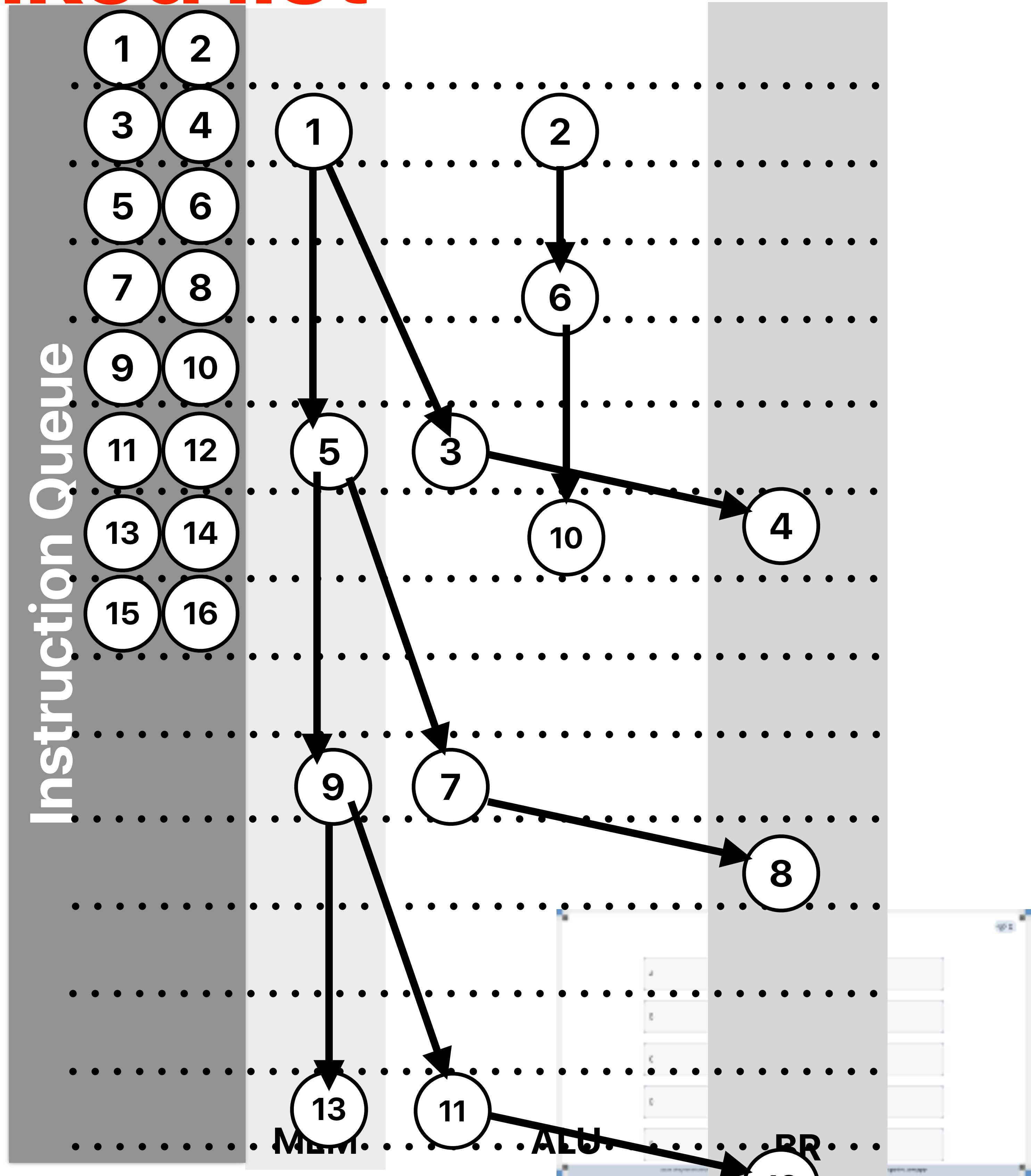
https://www.pollev.com/hungweitseng close in

What about "linked list"

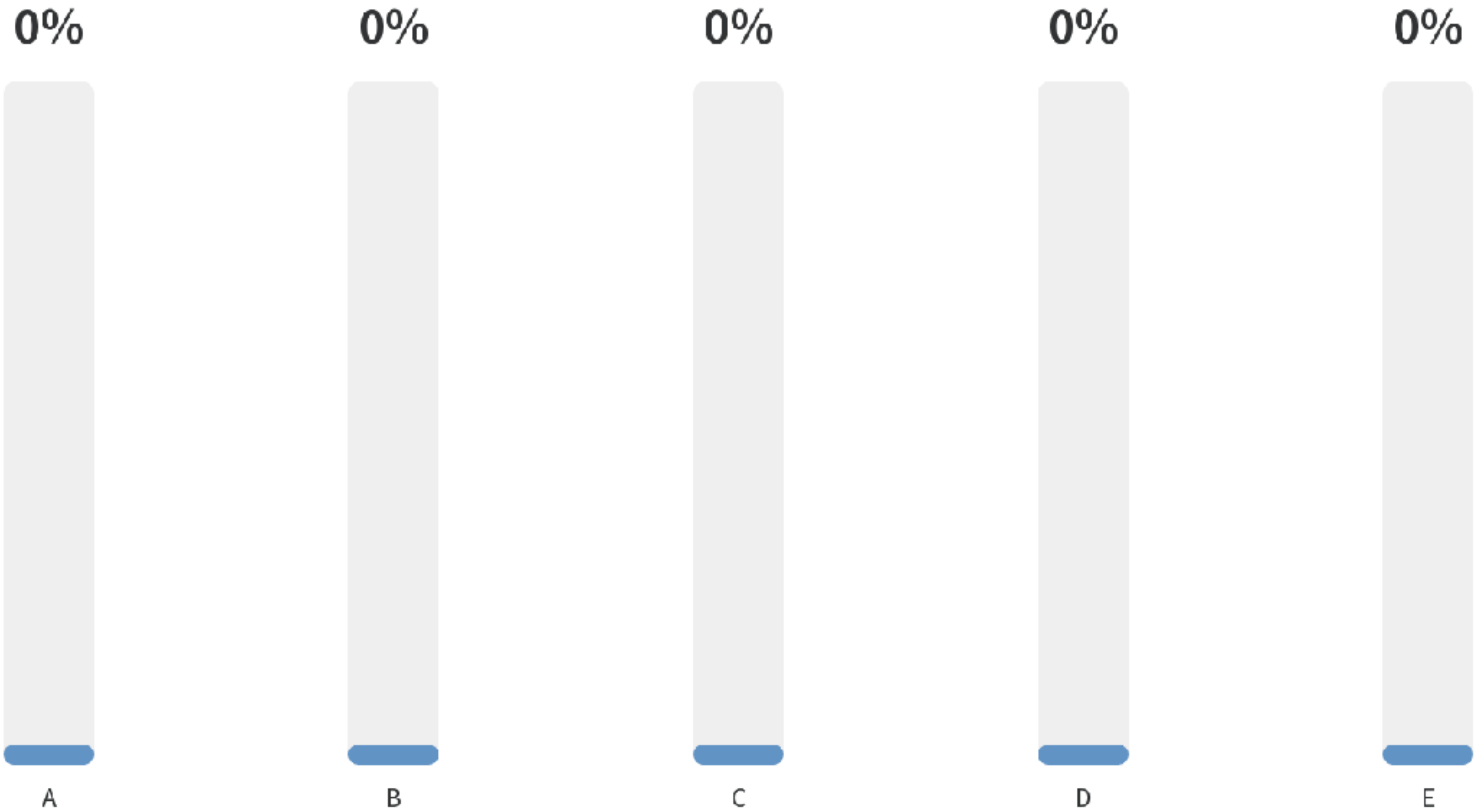
- For the following C code and it's translation in x86, **what's average CPI?** Assume the current PC is already at instruction (1) and this linked list has thousands of nodes. This processor can fetch and issue **2** instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.

```
do {
     number_of_nodes++;
     current = current->next;
} while ( current != NULL )
```

A. 0.5
B. 0.8
C. 1.0
D. 1.2
E. 1.5



0



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

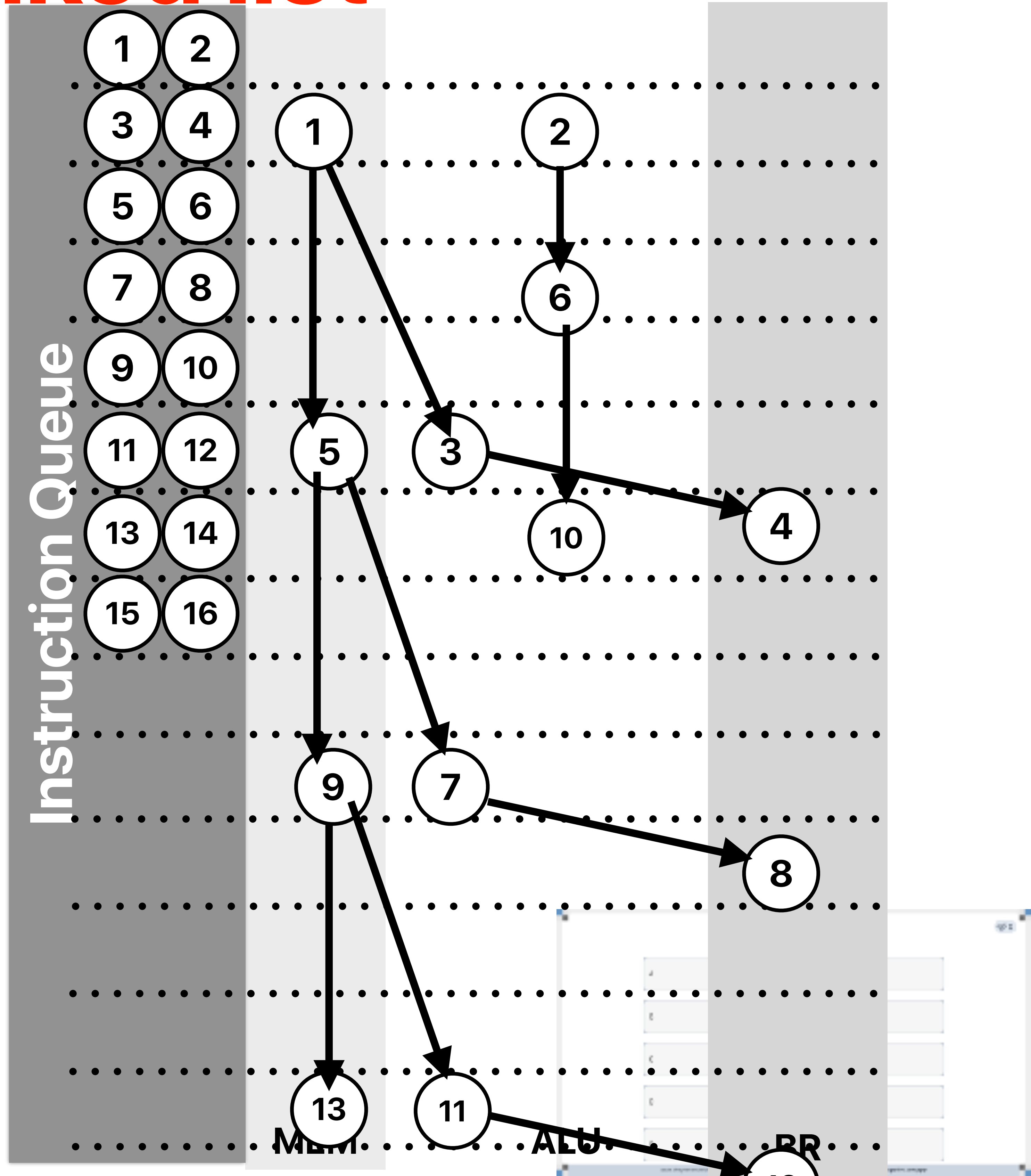
https://www.pollev.com/hungweitseng close in

What about "linked list"

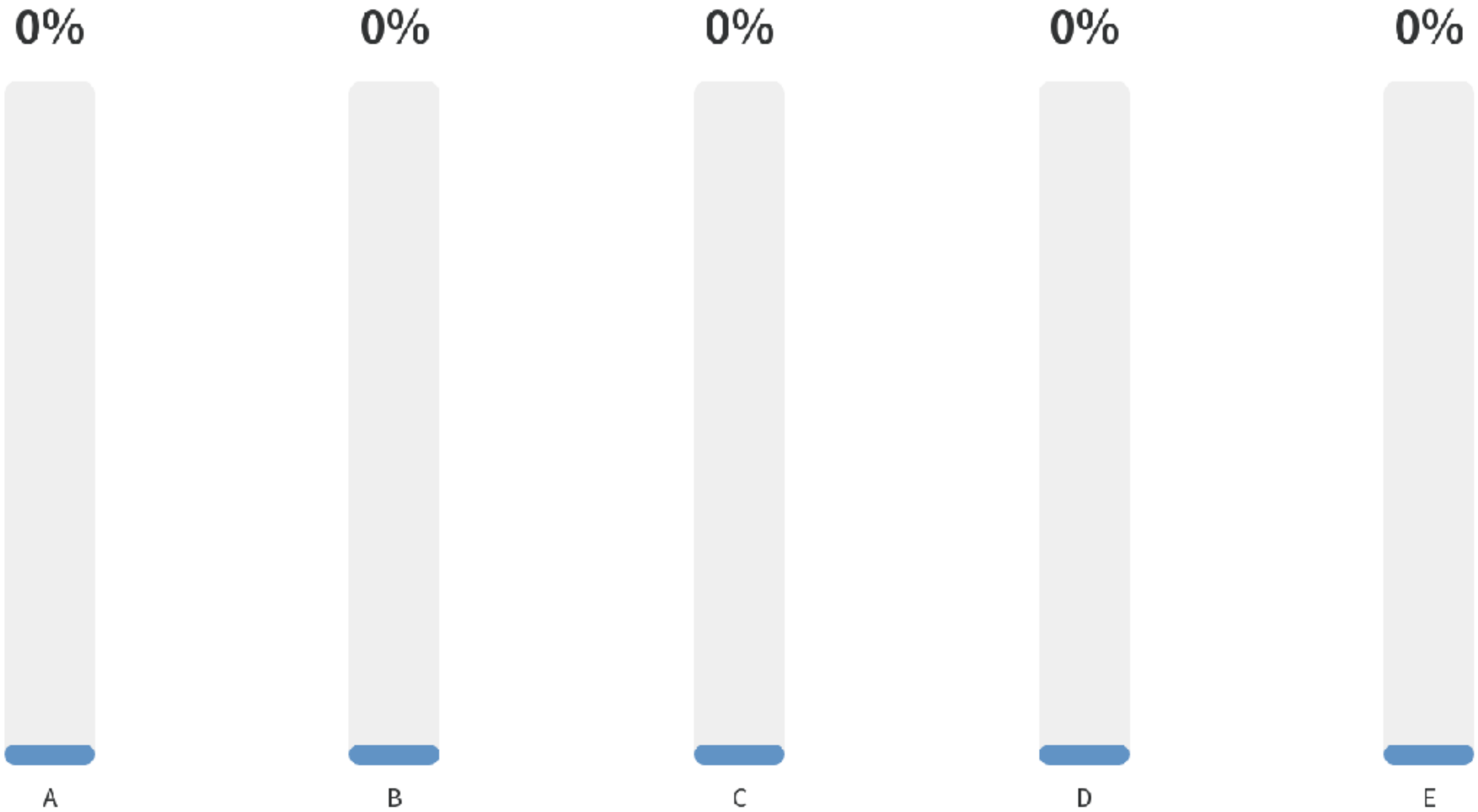
- For the following C code and it's translation in x86, **what's average CPI?** Assume the current PC is already at instruction (1) and this linked list has thousands of nodes. This processor can fetch and issue **2** instructions per cycle, with exactly the same register renaming hardware and pipeline as we showed previously.

```
do {
     number_of_nodes++;
     current = current->next;
} while ( current != NULL )
```

A. 0.5
B. 0.8
C. 1.0
D. 1.2
E. 1.5



0



Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

What about “linked list”

Performance determined by the critical path

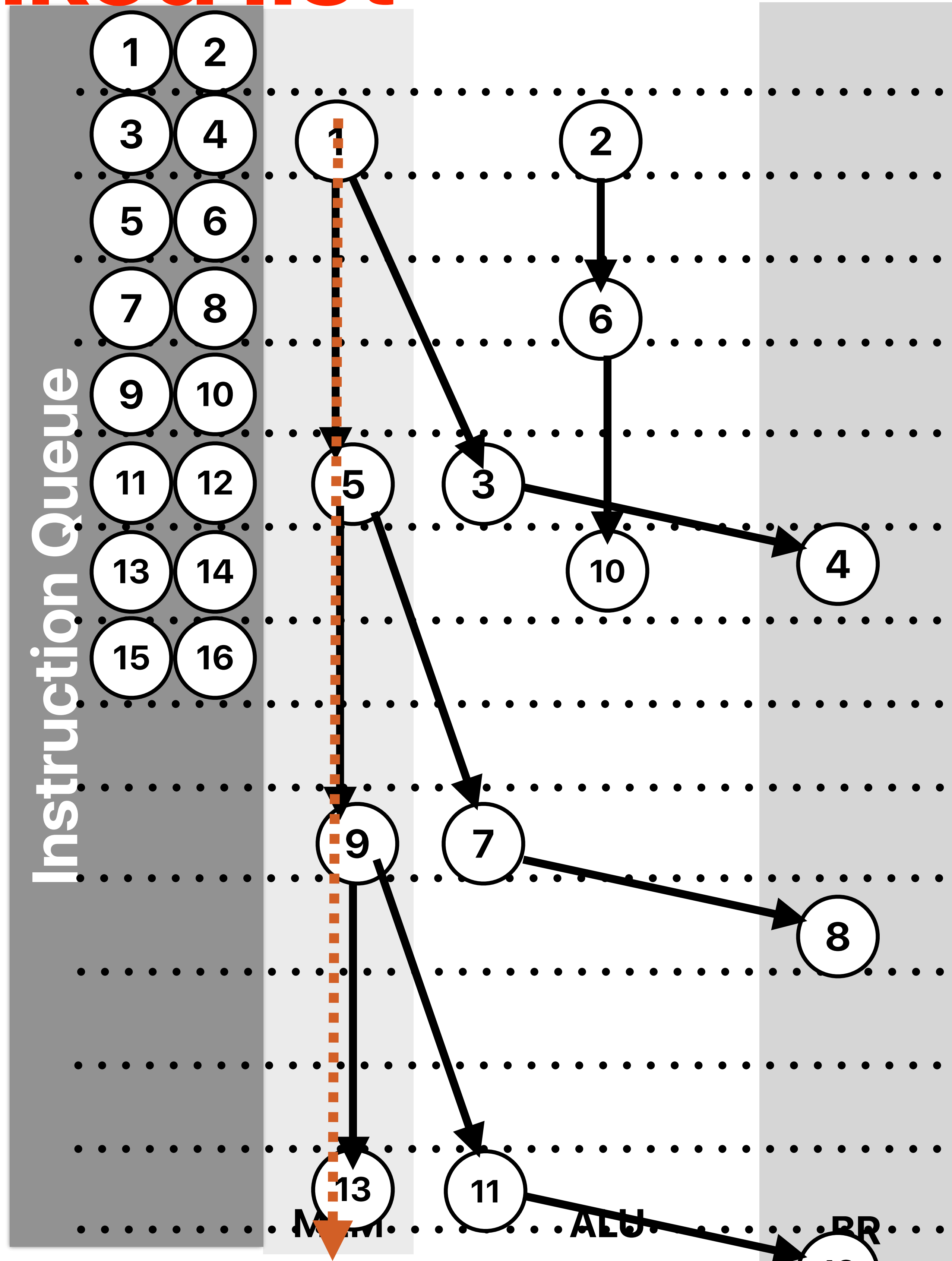
4 cycles each iteration

4 instructions per iteration

$$CPI = \frac{4}{4} = 1$$

```
do {
    number_of_nodes++;
    current = current->next;
} while ( current != NULL );
```

```
① .L3:    movq    8(%rdi), %rdi
②         addl    $1, %eax
③         testq   %rdi, %rdi
④         jne     .L3
```



The pipelines of Modern Processors

Intel Skylake

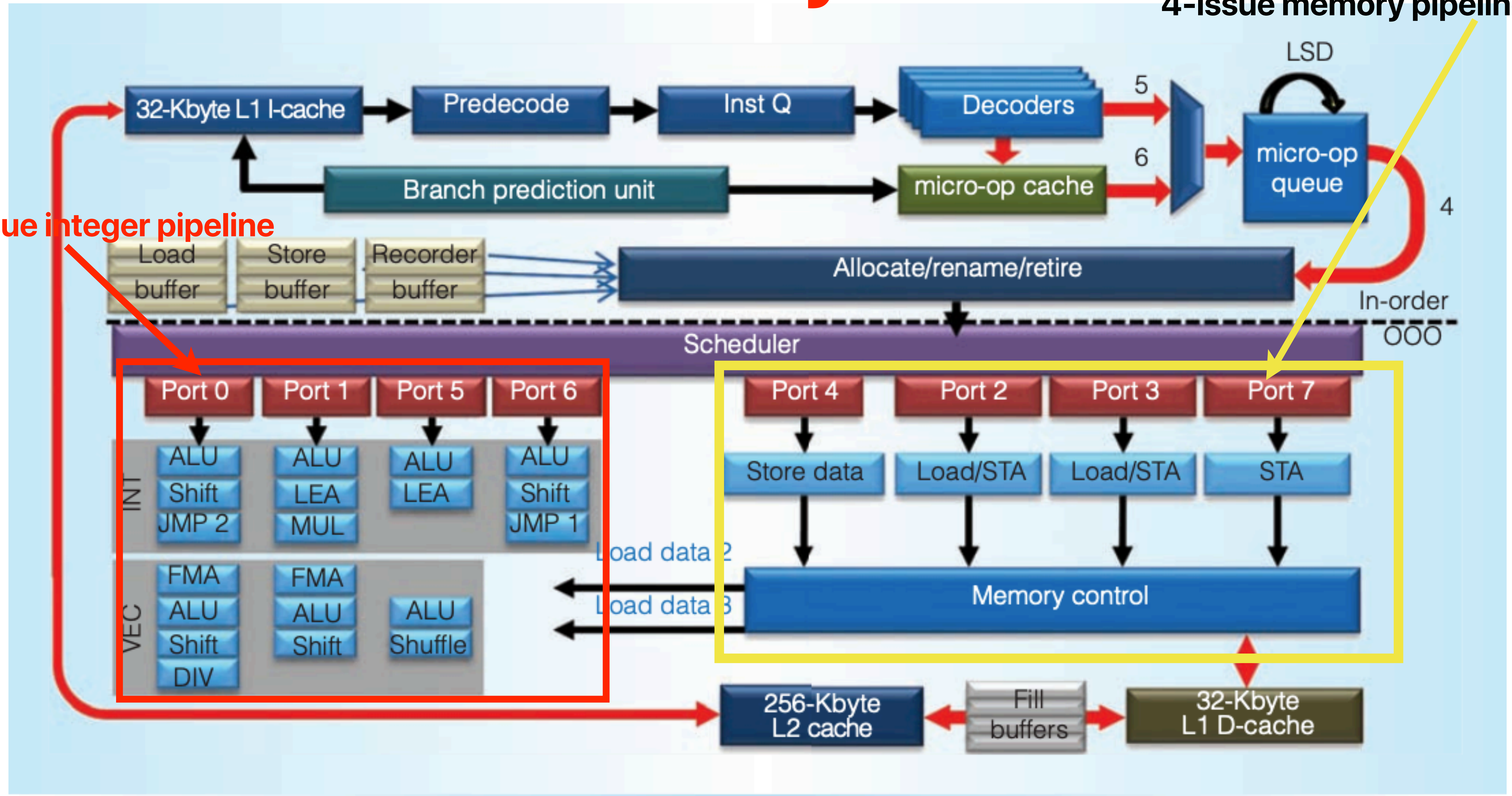


Figure 4. Skylake core block diagram.

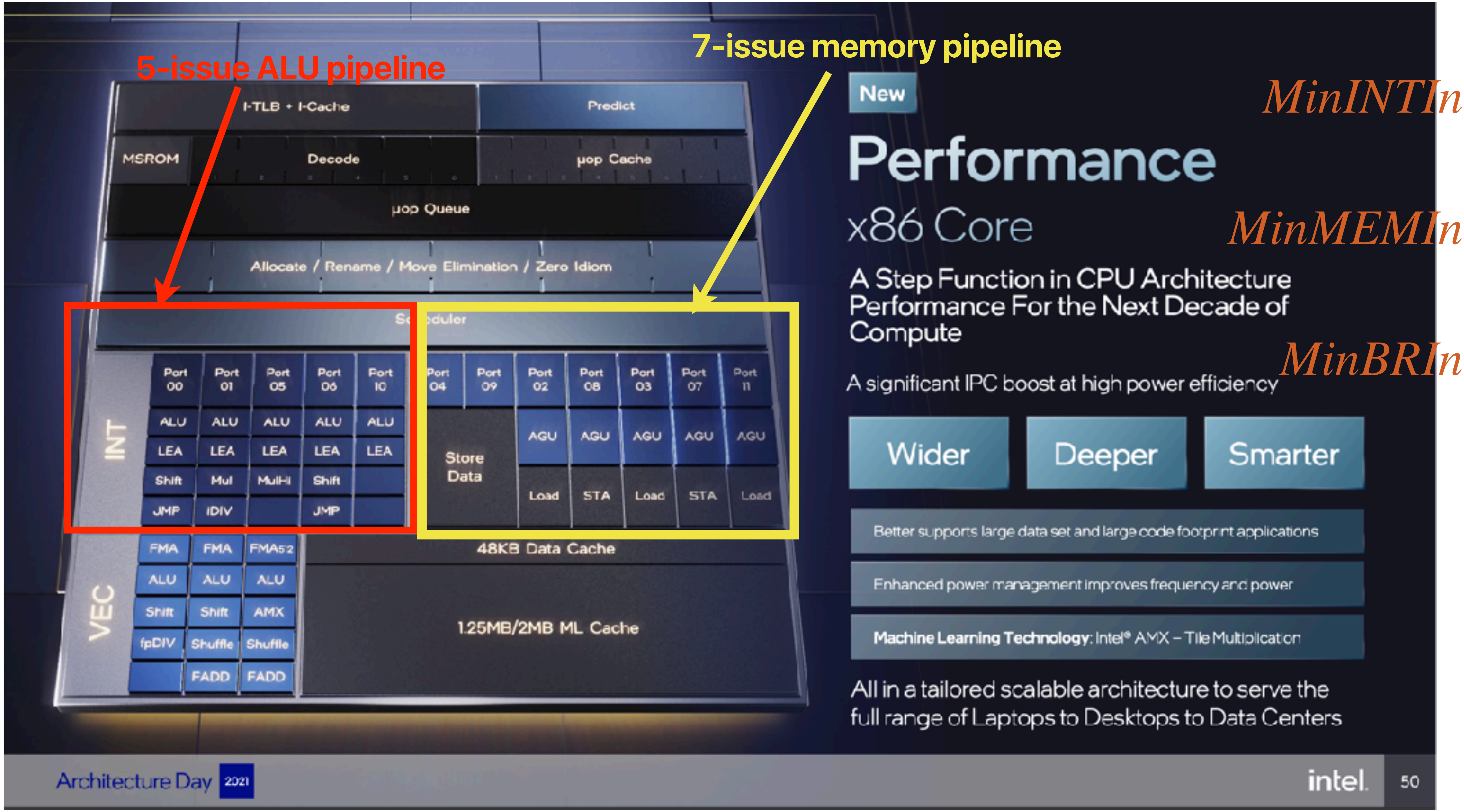
Intel Alder Lake

$$MinCPI = \frac{1}{12}$$

$$MinINTInst.CPI = \frac{1}{5}$$

$$MinMEMInst.CPI = \frac{1}{7}$$

$$MinBRInst.CPI = \frac{1}{2}$$



New

Performance

x86 Core

A Step Function in CPU Architecture Performance For the Next Decade of Compute

A significant IPC boost at high power efficiency

Wider

Deeper

Smarter

Better supports large data set and large code footprint applications

Enhanced power management improves frequency and power

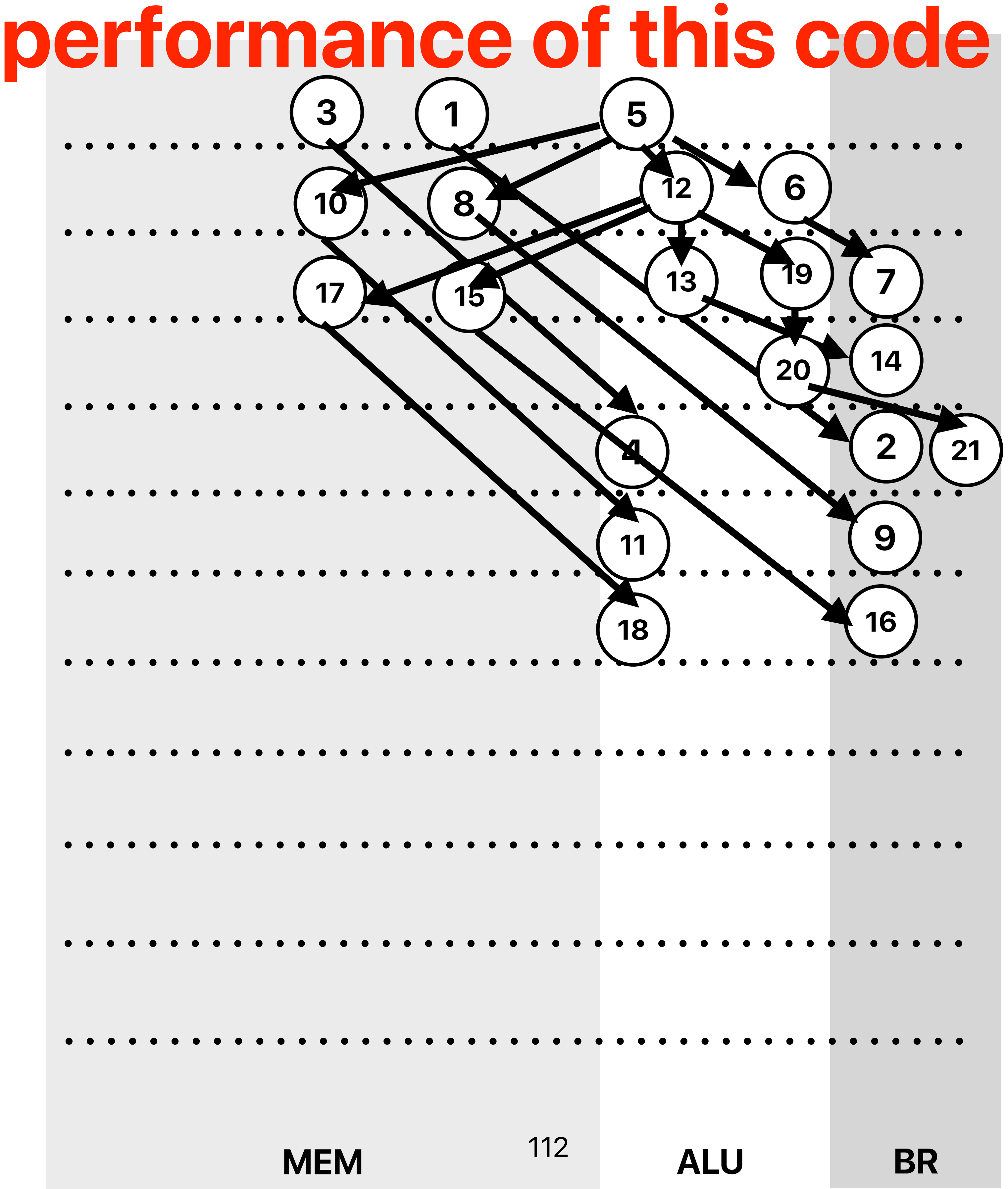
Machine Learning Technology: Intel® AMX – Tile Multiplication

All in a tailored scalable architecture to serve the full range of Laptops to Desktops to Data Centers

https://download.intel.com/newsroom/2021/client-computing/intel-architecture-day-2021-presentation.pdf

Project the performance of this code on Alder Lake

```
①  .L10:
    cmpq   $0, 8(%rdi)
②  je  .L9
③  movslq (%rdi), %rdx
④  addq   %rdx, %rax
⑤  .L9:
   addq   $16, %rdi
⑥  cmpq   %rdi, %rcx
⑦  jne .L10
⑧  .L10:
    cmpq   $0, 8(%rdi)
⑨  je  .L9
⑩  movslq (%rdi), %rdx
⑪  addq   %rdx, %rax
⑫  .L9:
   addq   $16, %rdi
⑬  cmpq   %rdi, %rcx
⑭  jne .L10
⑮  .L10:
    cmpq   $0, 8(%rdi)
⑯  je  .L9
⑰  movslq (%rdi), %rdx
⑱  addq   %rdx, %rax
⑲  .L9:
   addq   $16, %rdi
⑳  cmpq   %rdi, %rcx
㉑  jne .L10
```



Summary: Characteristics of modern processor architectures

- Multiple-issue pipelines with multiple functional units available
  - Multiple ALUs
  - Multiple Load/store units
  - Dynamic OoO scheduling to reorder instructions whenever possible
- Cache — very high hit rate if your code has good locality
  - Very matured data/instruction prefetcher
- Branch predictors — very high accuracy if your code is predictable
  - Perceptron
  - Variable history predictors

Announcements

- Tips for answering examine question (or more importantly, technical interviews)
  - Be precise — pulling everything you know simply shows you don't really understanding what the question is asking
  - Important things go first — interviewers and graders lose patience very quick, you need to give a summary or conclusion before going into detail
  - We cannot read your mind. Explaining your thoughts after the examine (e.g., regrading requests) or interview (e.g., through follow-up) **does not** help win anything back
  - Show your work — simply tell the interviewer we can use DP to solve this problem won't give you positive review results
- **Last reading quiz** due this Wednesday before the lecture

Computer
Science &
Engineering

142

つづく