Introduction to Programming (C/C++)

09: Performance Fun!

Huanchen Zhang

清華大學
Tsinghua University

交叉信息研究院
Institute for Interdisciplinary Information Sciences

Measuring Time

Turn On Compiler Optimization

-O2/-O3

Command-Line Arguments

```
$ ./myProg 10 happy
```

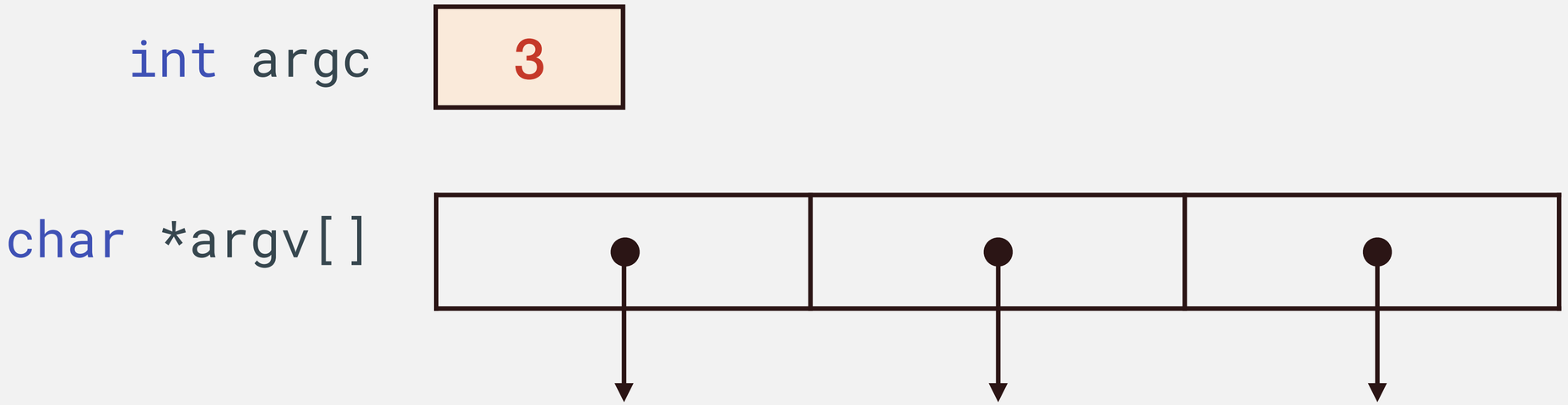
Command-Line Arguments

```
$ ./myProg 10 happy
```

```
int argc    3
```

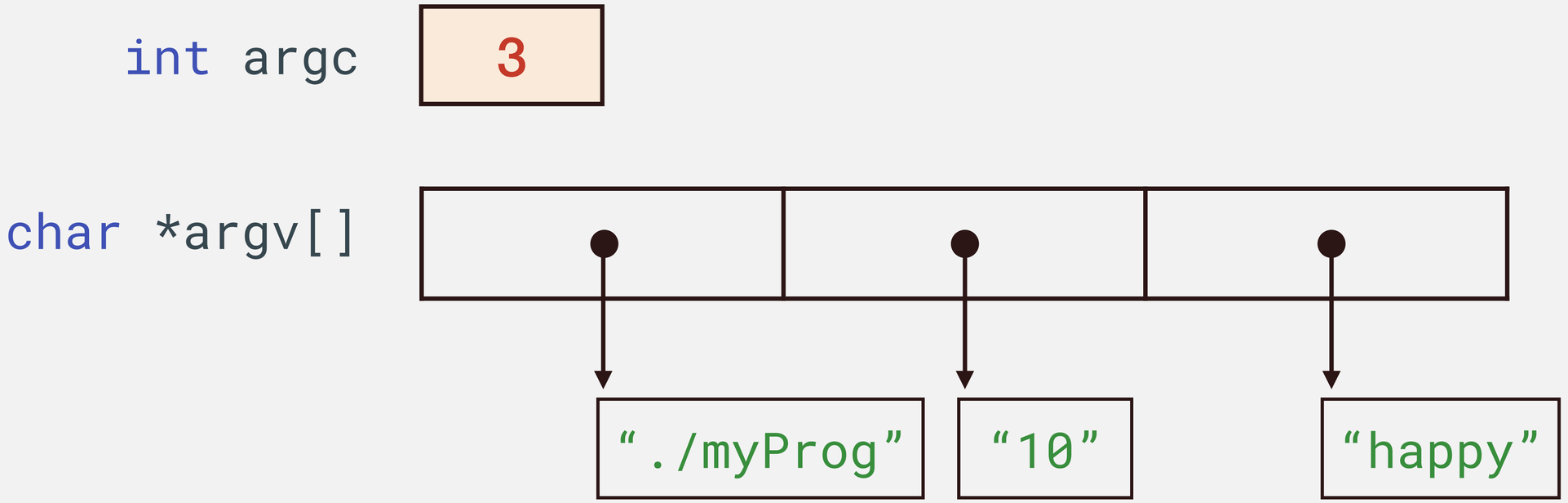
Command-Line Arguments

```
$ ./myProg 10 happy
```



Command-Line Arguments

```
$ ./myProg 10 happy
```



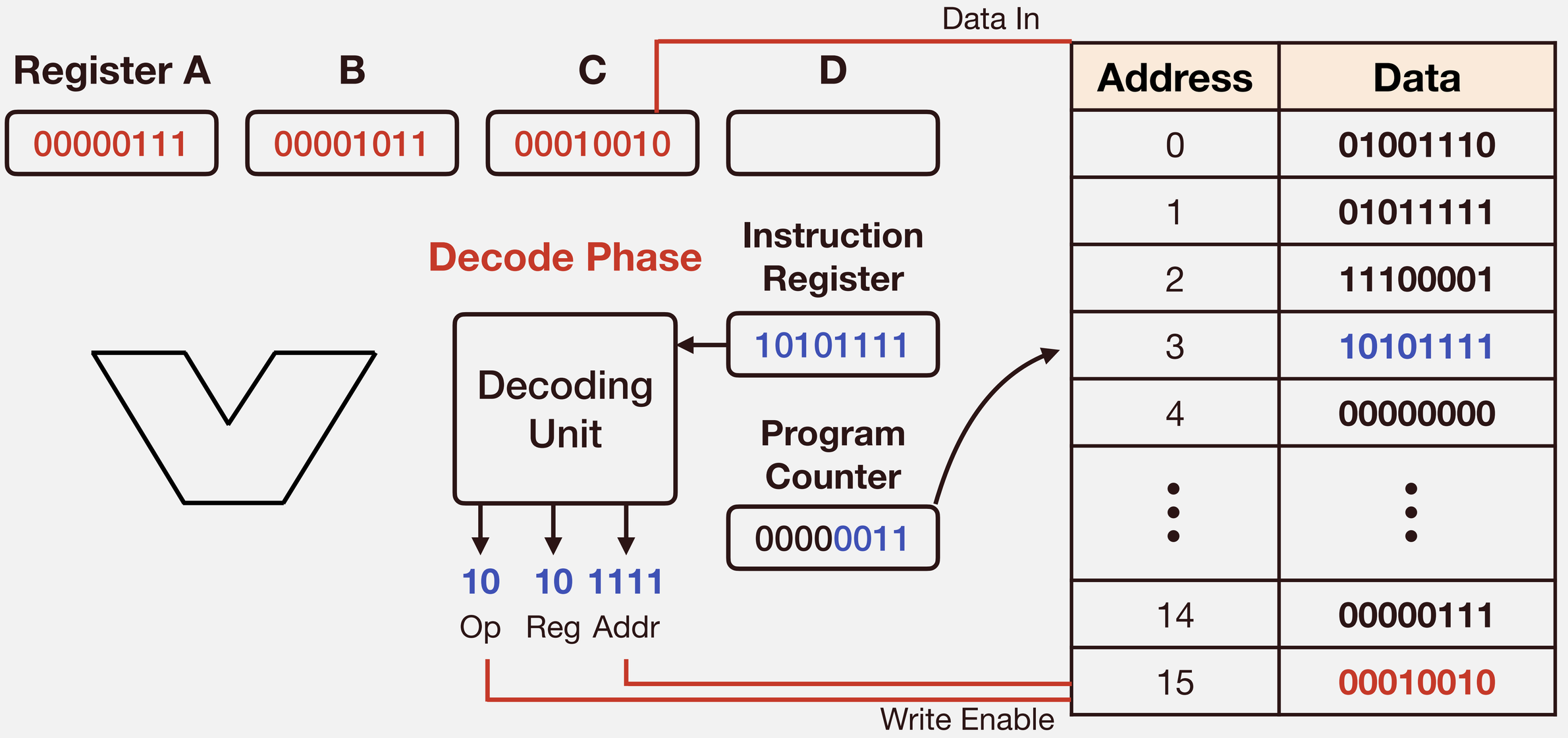
Algorithms Matter a Lot

Algorithms Matter a Lot

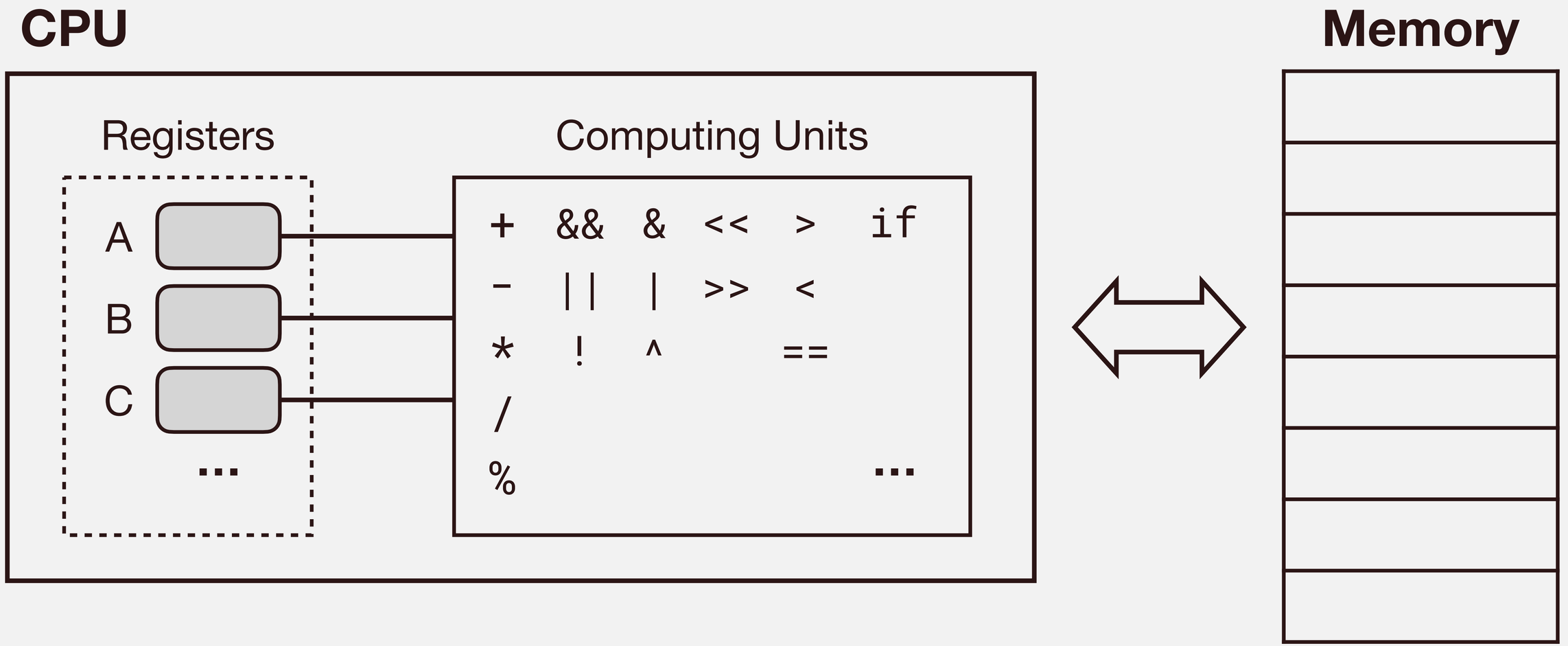
At Large Scale

Don’t Do More Work Than You Have To

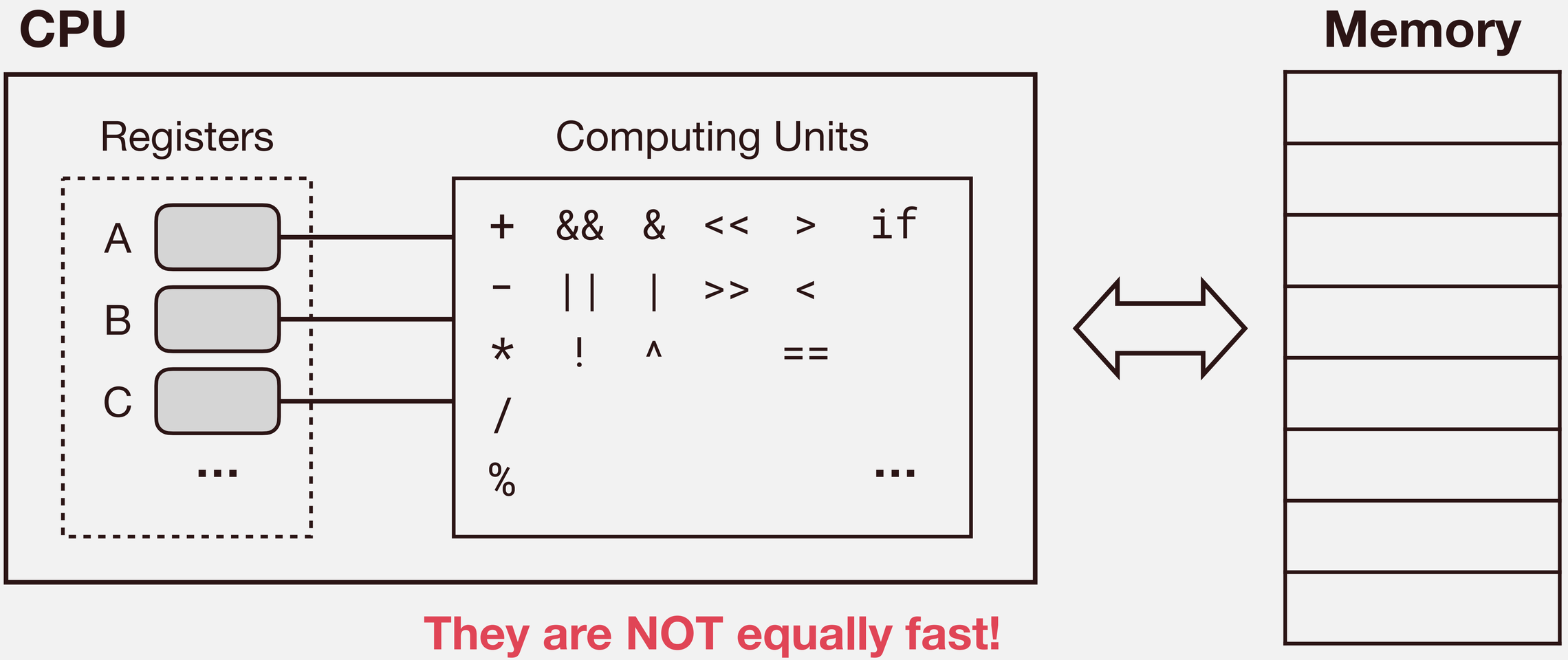
Building a Central Processing Unit (CPU)



Know Your Architecture

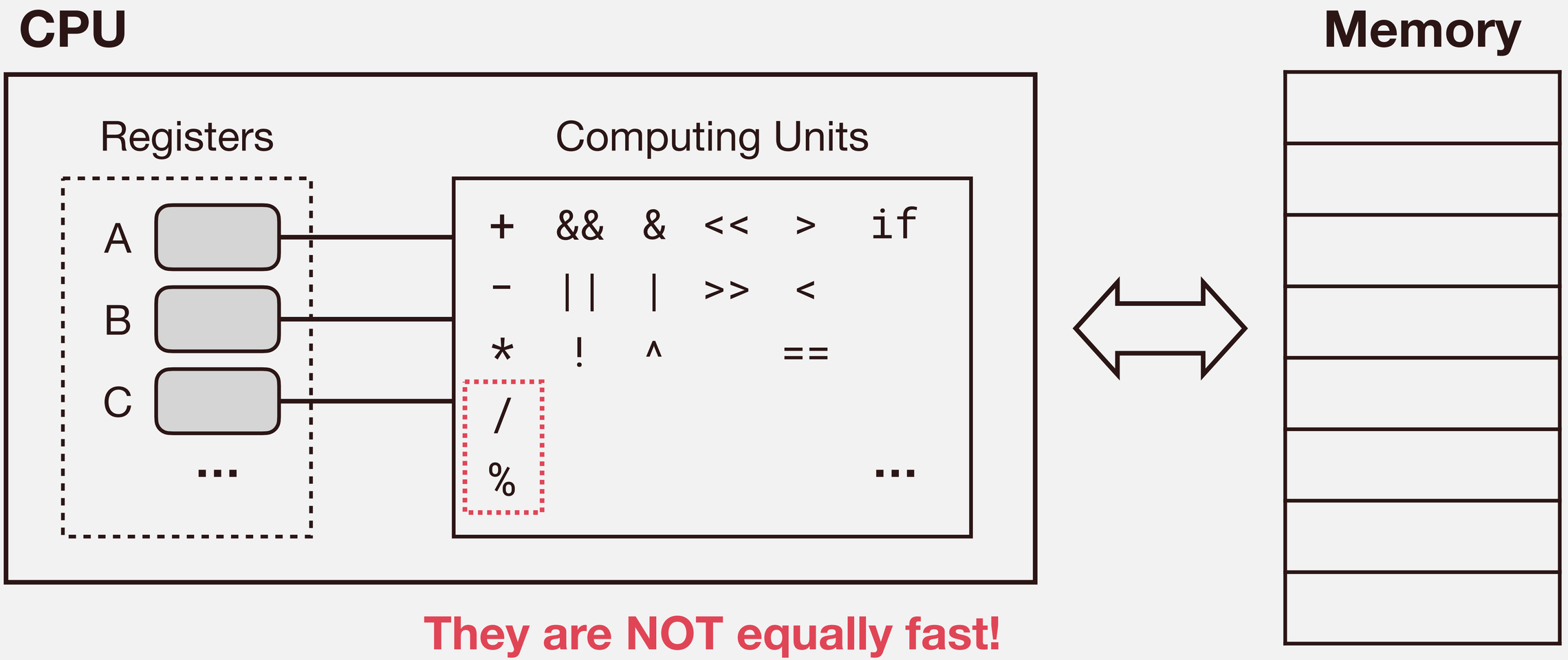


Know Your Architecture



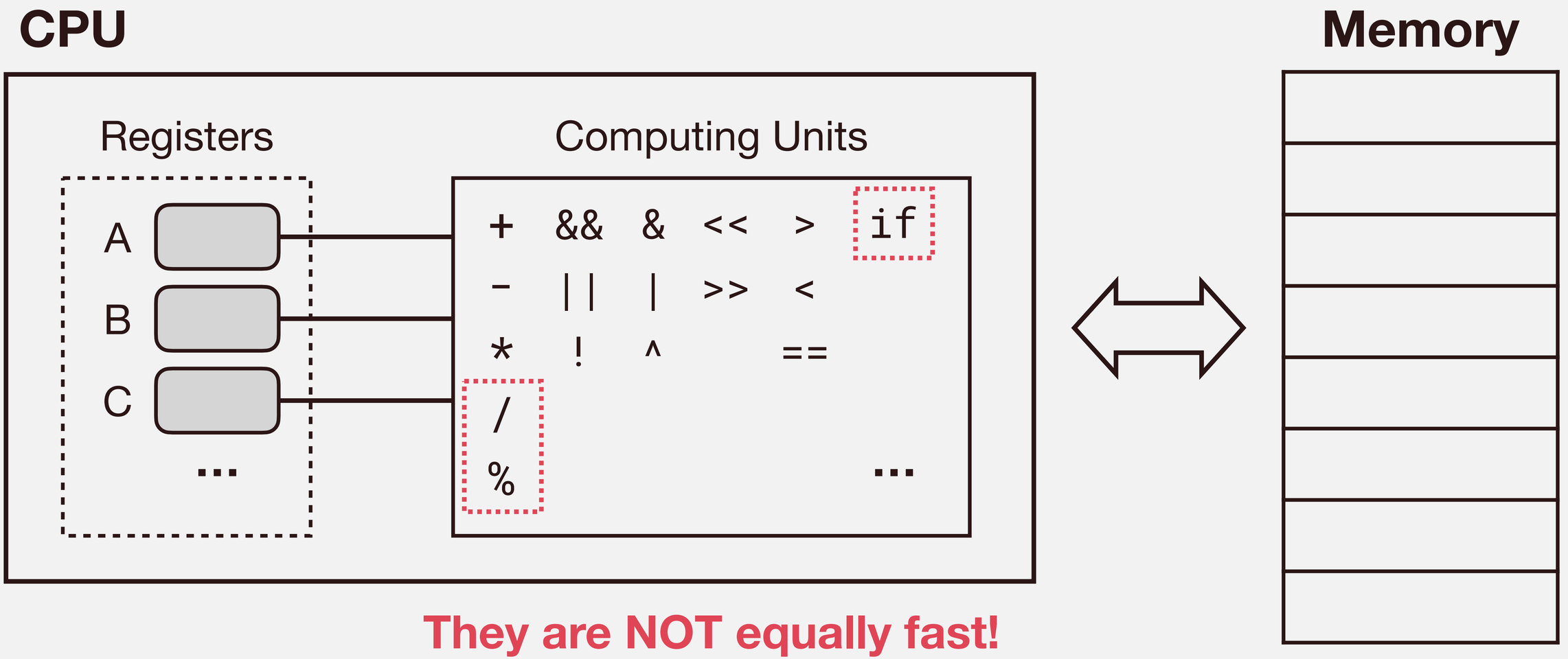
They are NOT equally fast!

Know Your Architecture



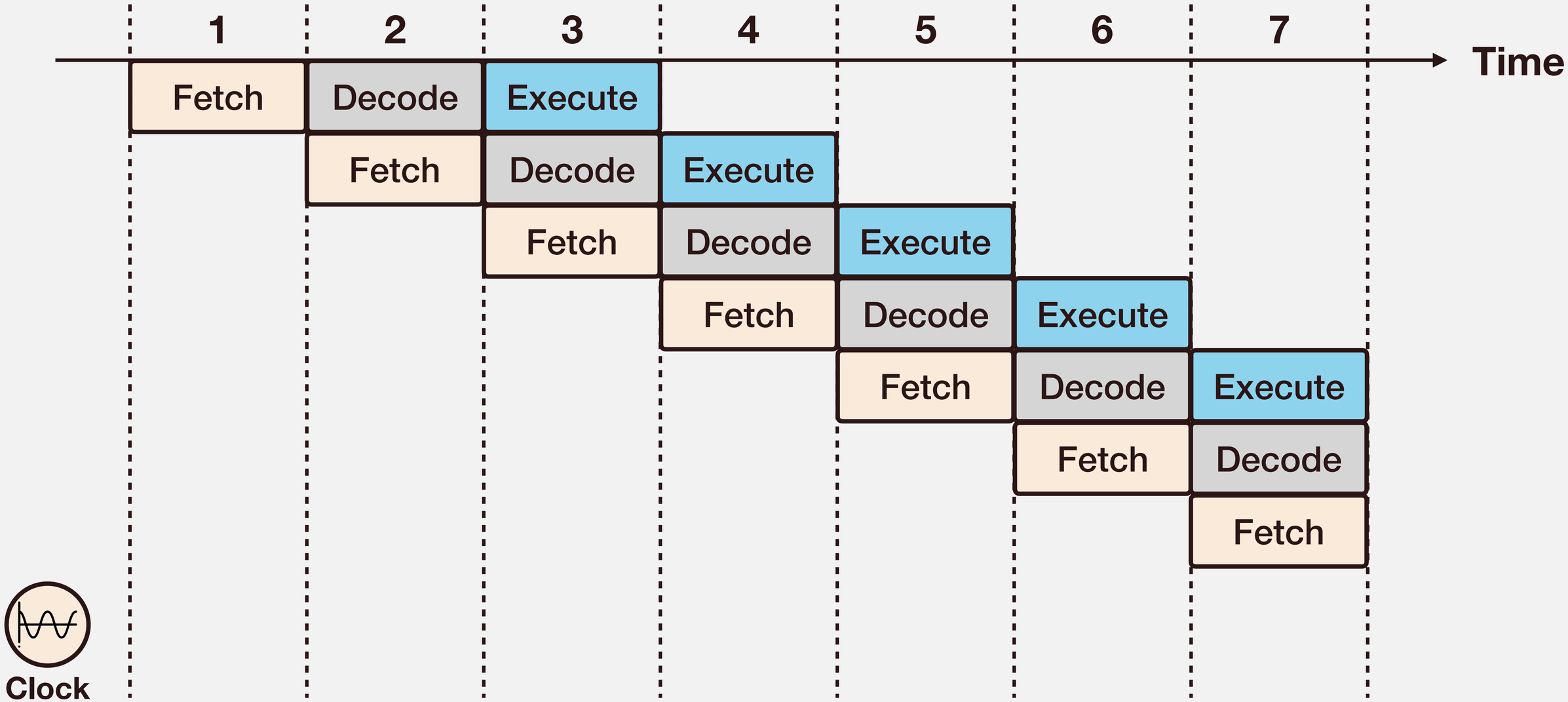
They are NOT equally fast!

Know Your Architecture



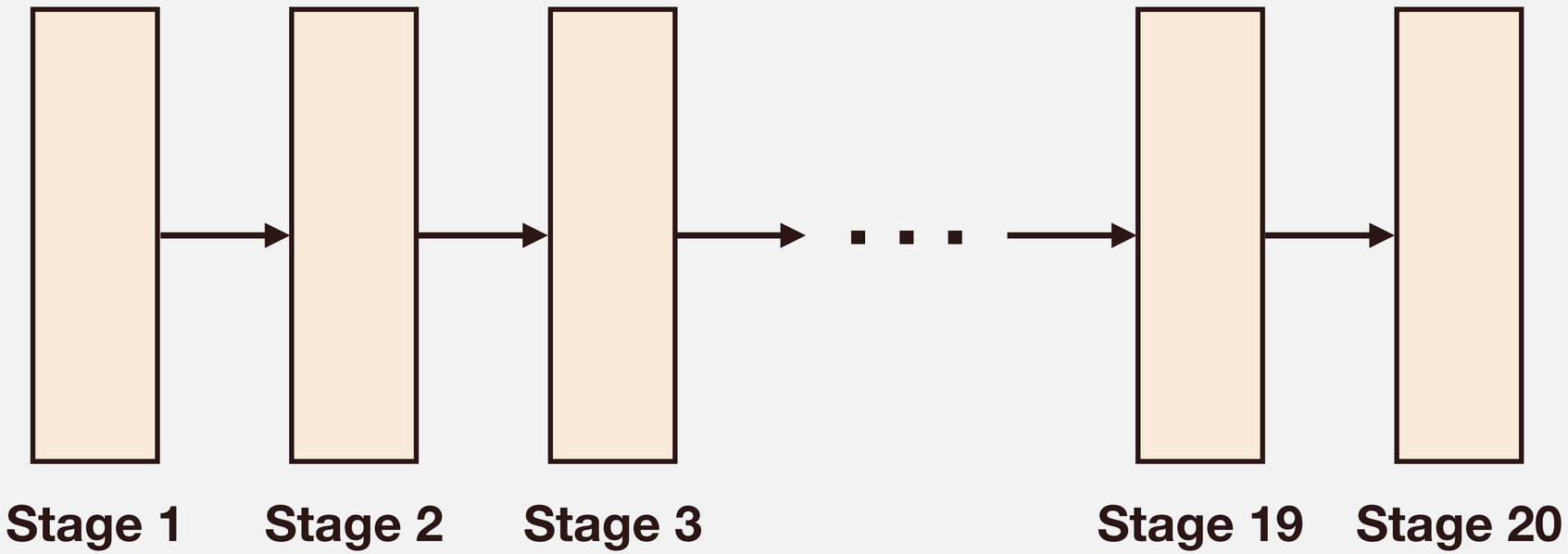
They are NOT equally fast!

CPU Pipeline



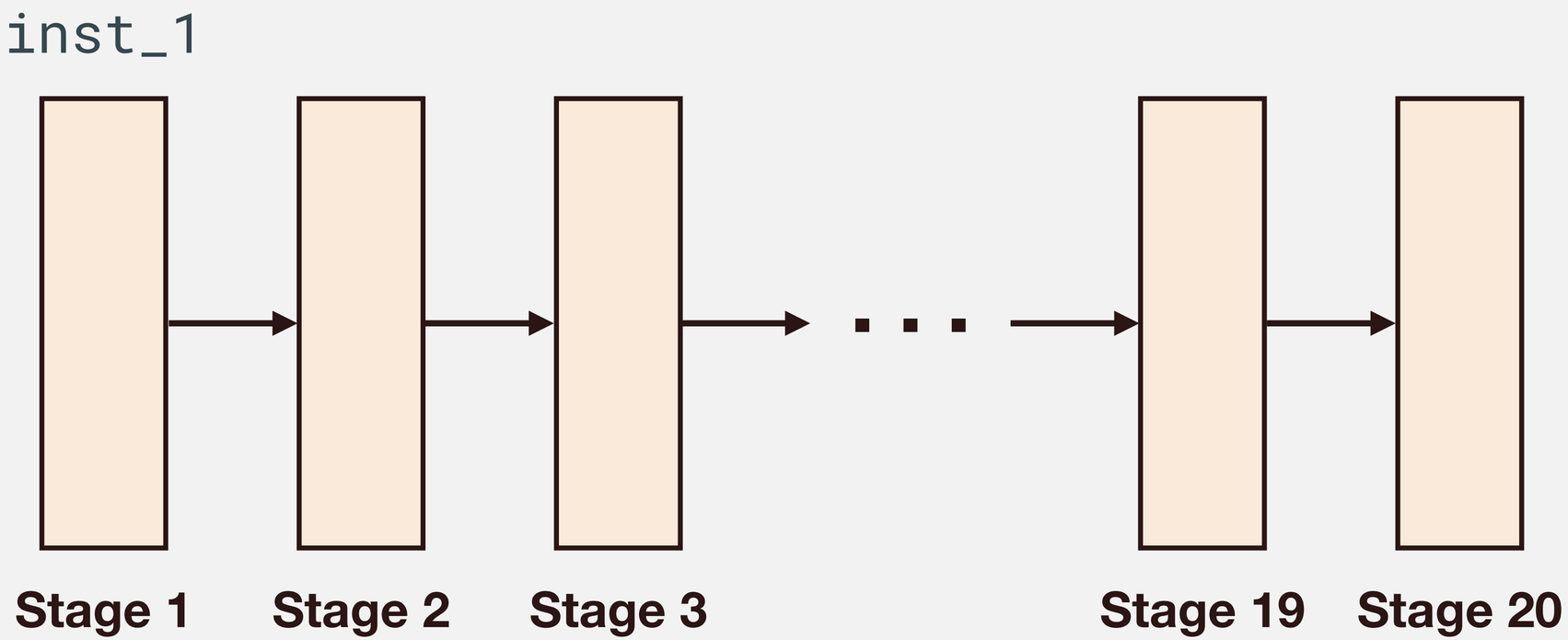
CPU Pipeline

```
inst_1
inst_2
inst_3
...
inst_N
```



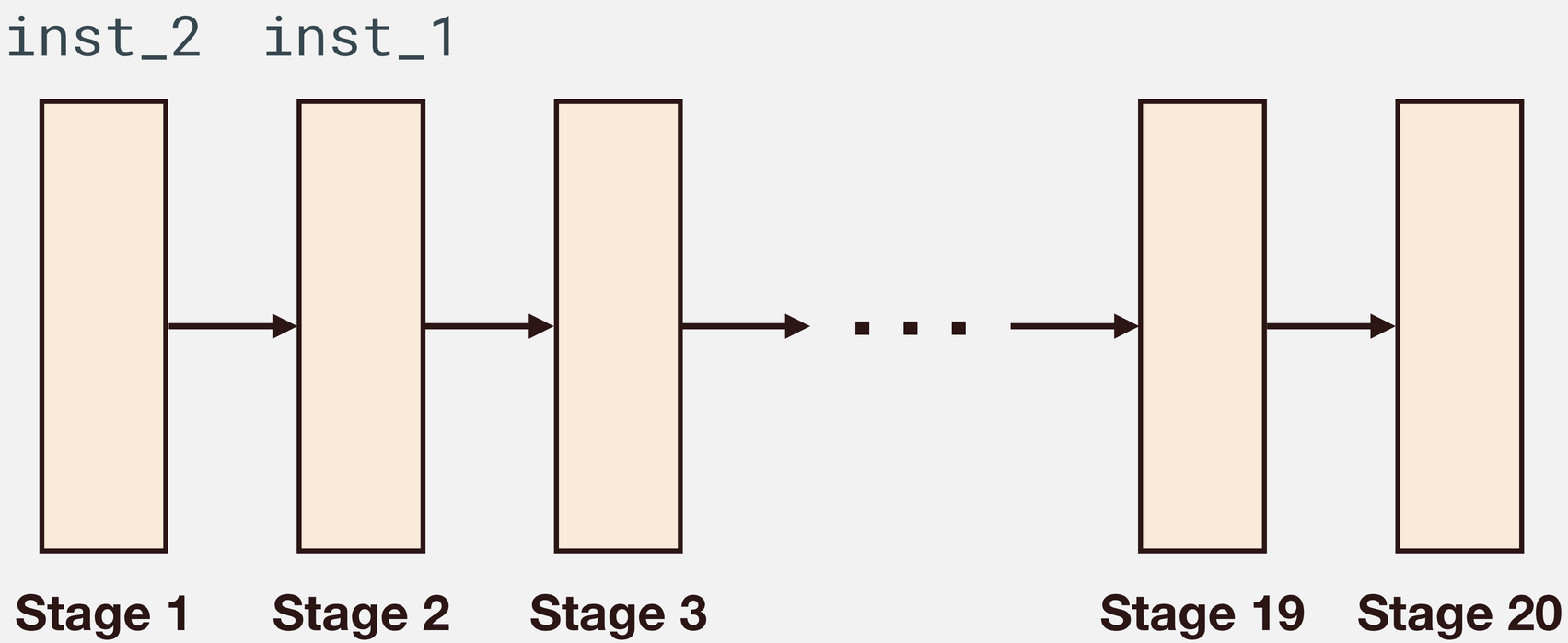
CPU Pipeline

```
inst_1
inst_2
inst_3
...
inst_N
```



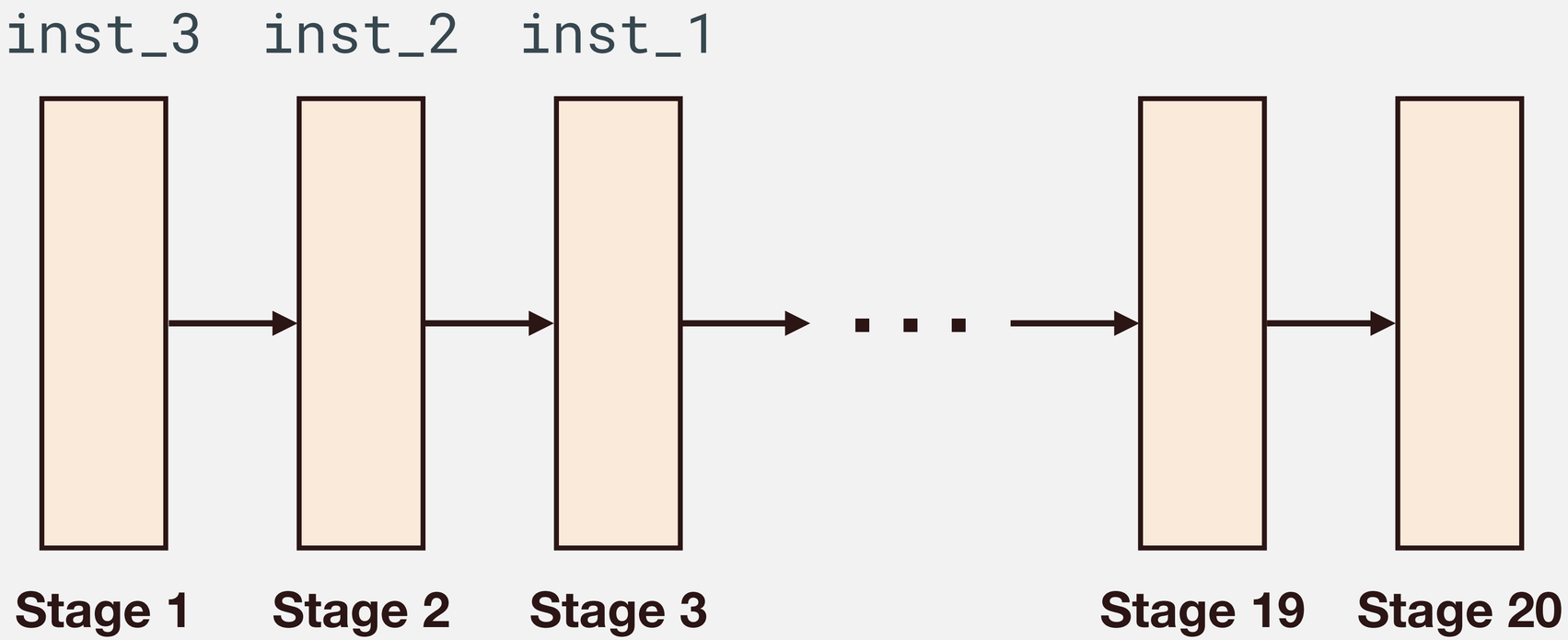
CPU Pipeline

```
inst_1
inst_2
inst_3
...
inst_N
```



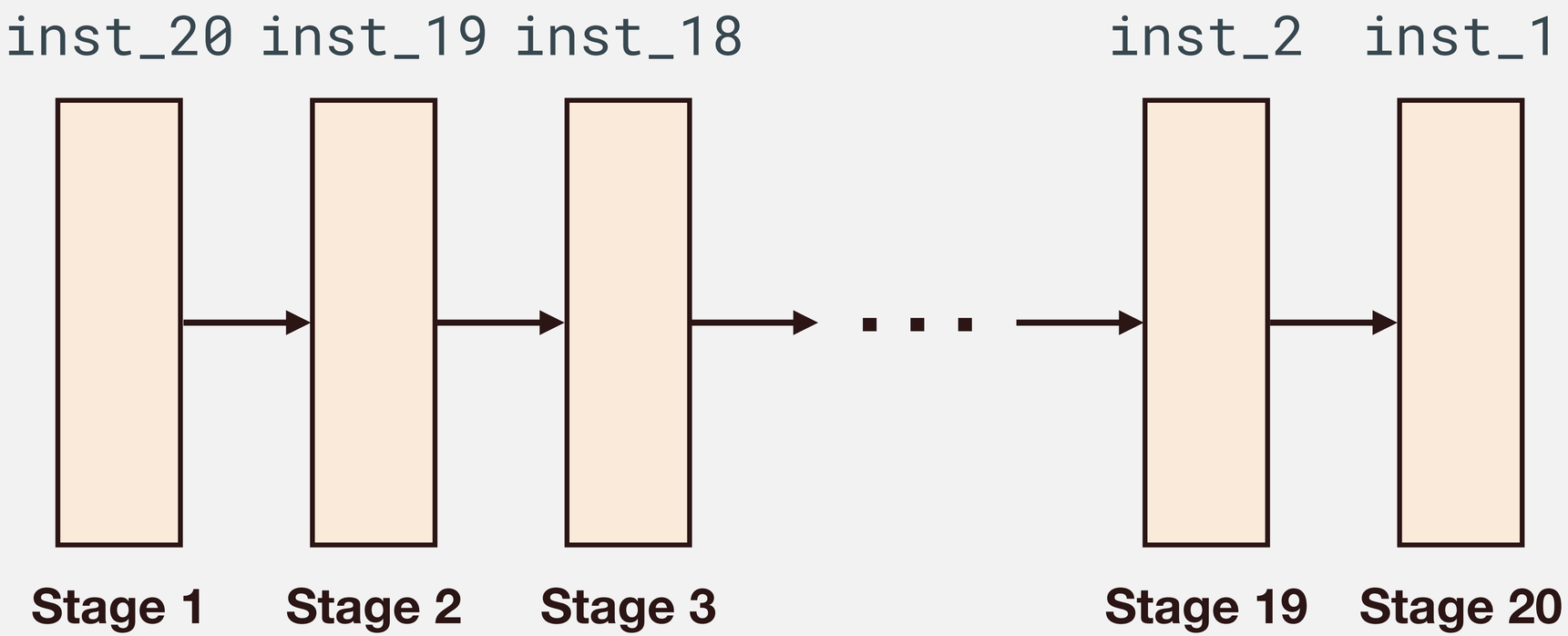
CPU Pipeline

```
inst_1
inst_2
inst_3
...
inst_N
```



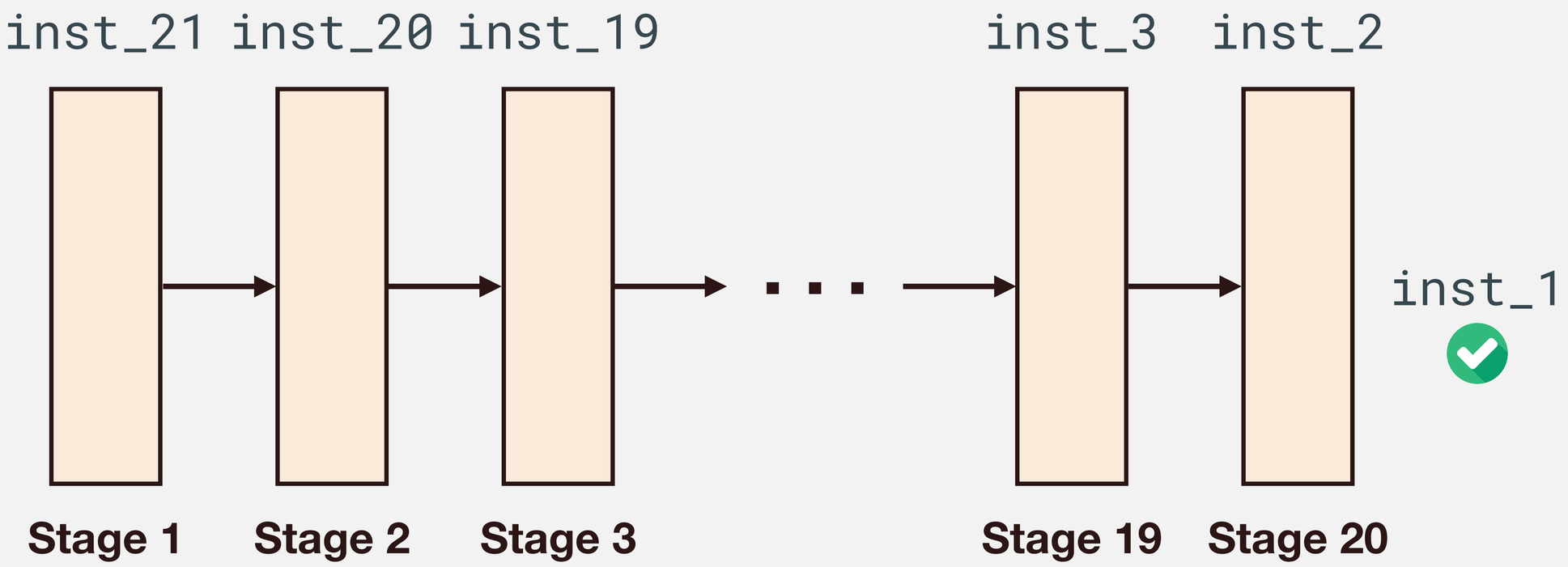
CPU Pipeline

```
inst_1
inst_2
inst_3
...
inst_N
```



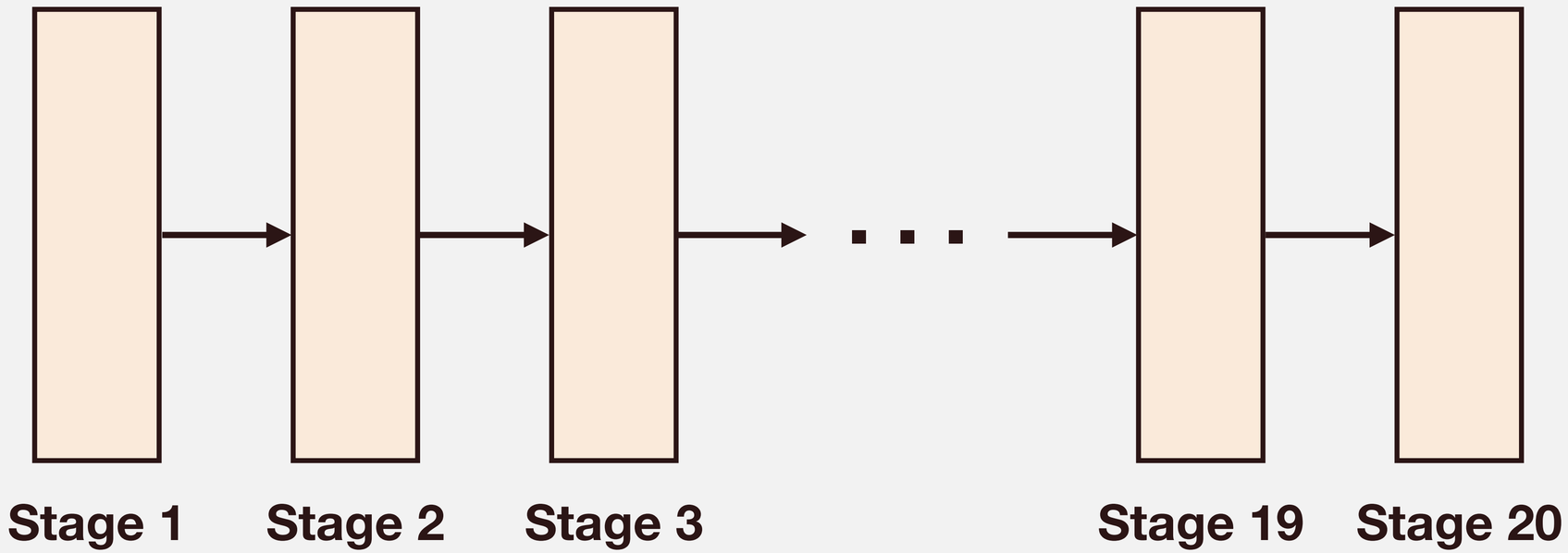
CPU Pipeline

inst_1
inst_2
inst_3
...
inst_N



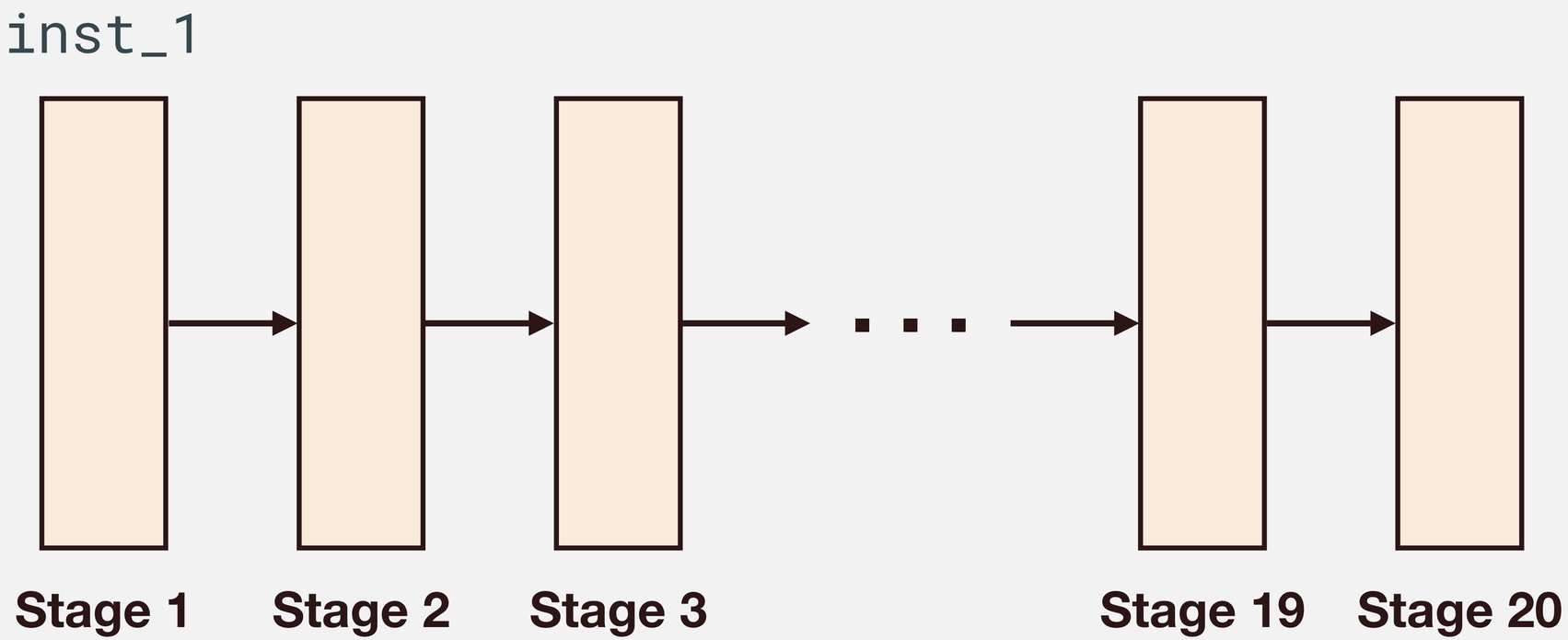
CPU Pipeline

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



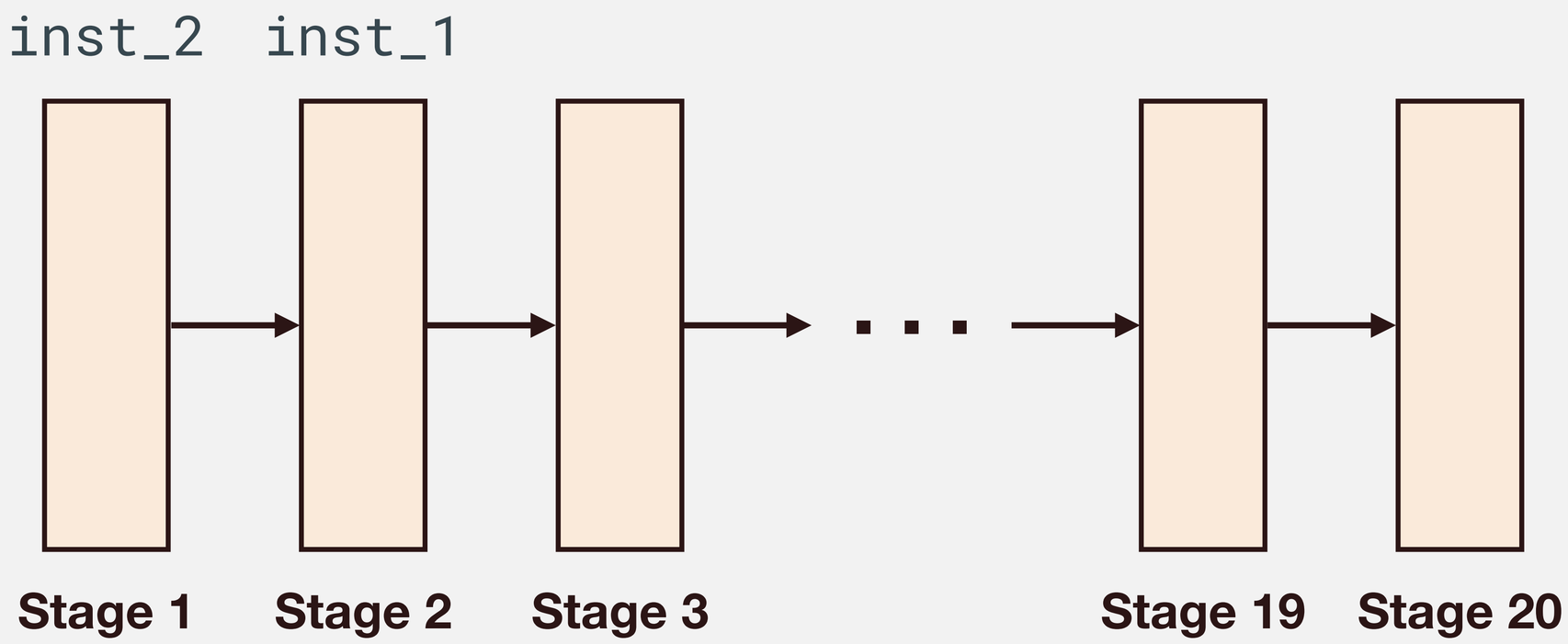
CPU Pipeline

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



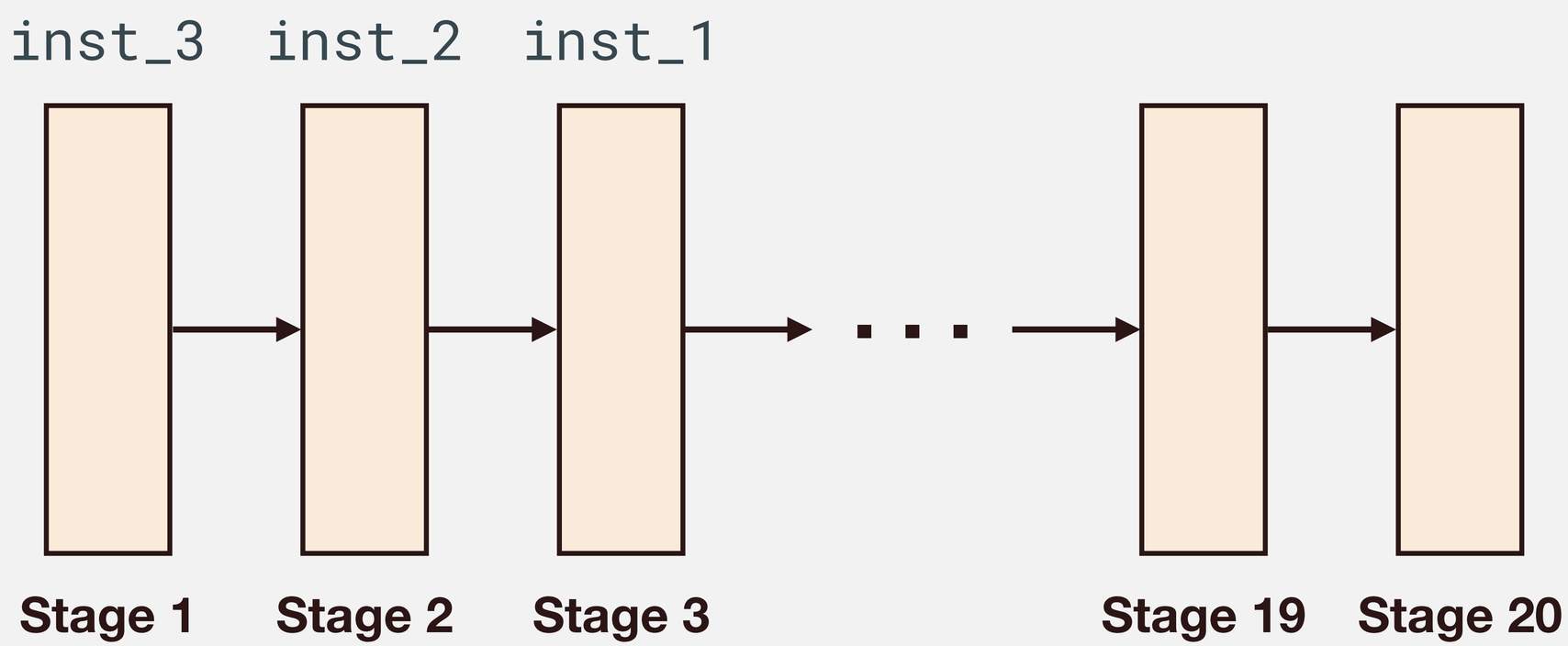
CPU Pipeline

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



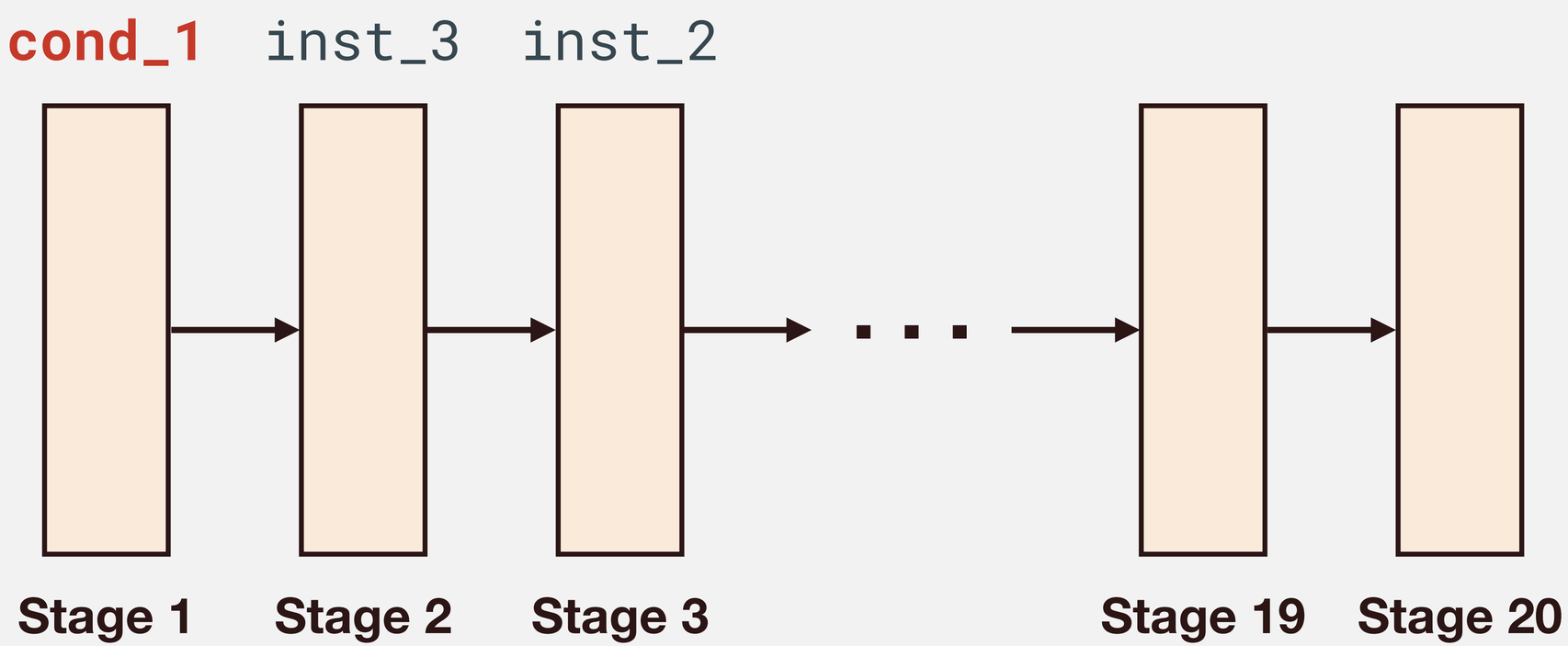
CPU Pipeline

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



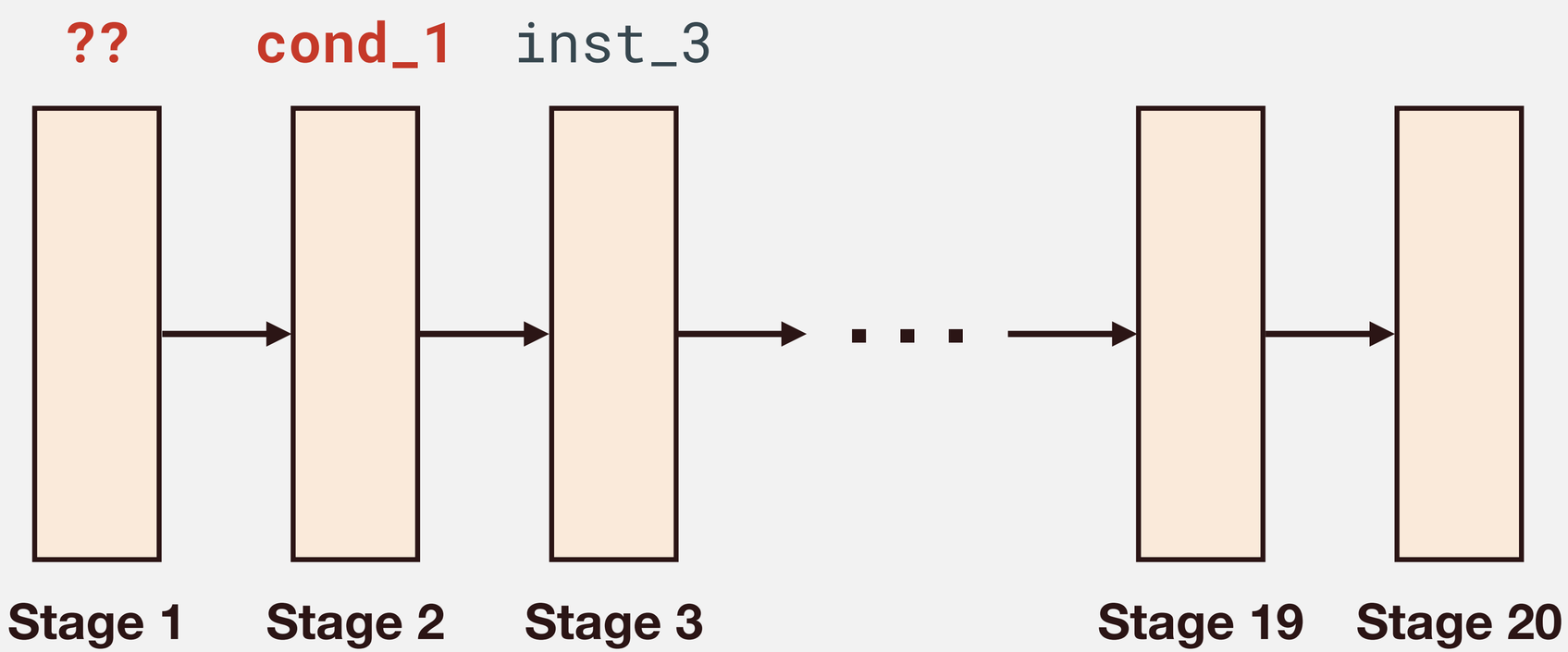
CPU Pipeline

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



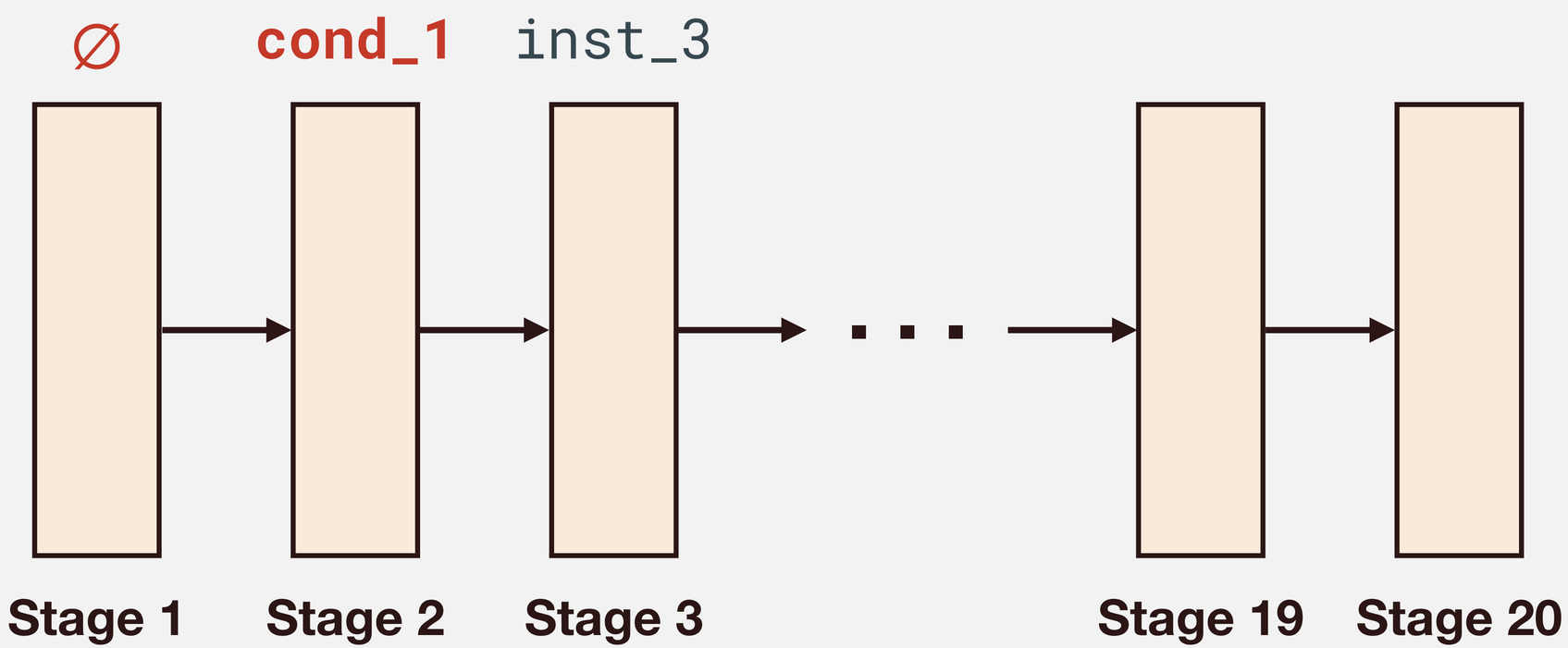
CPU Pipeline

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



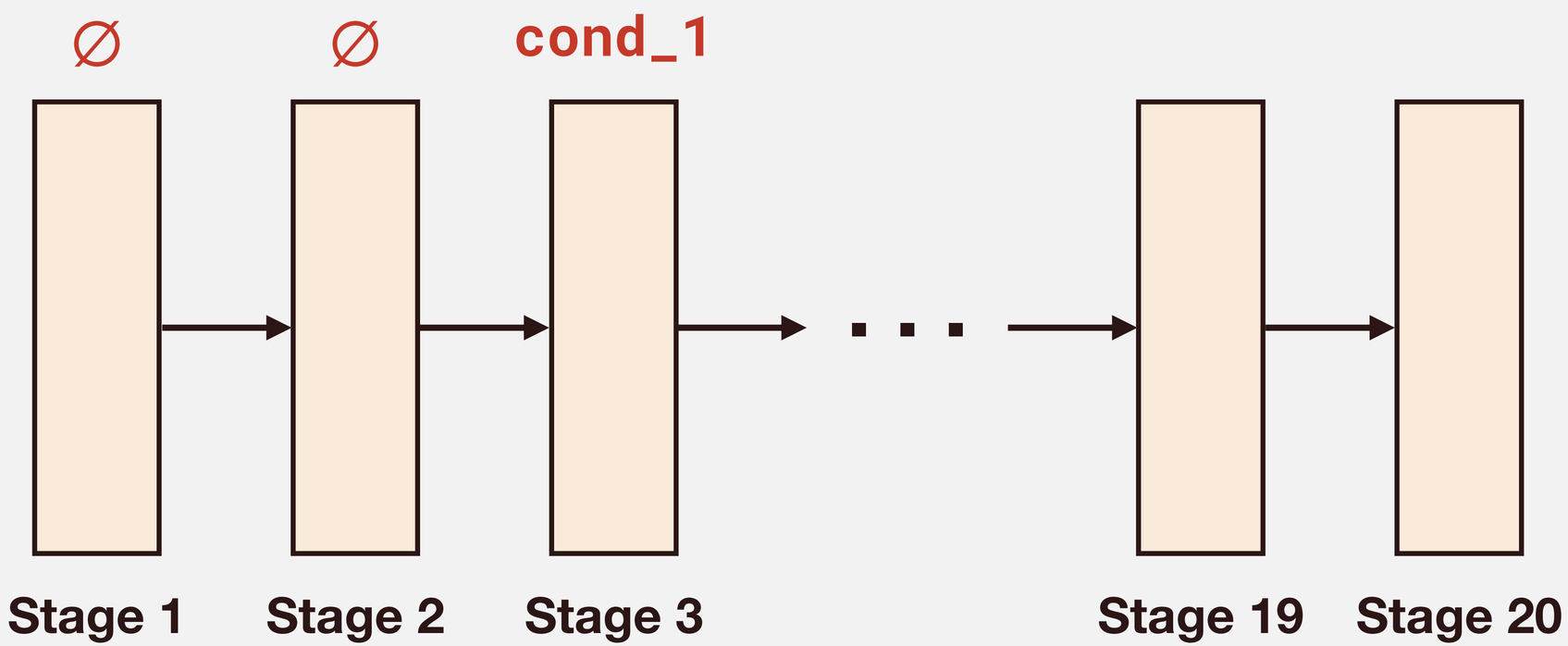
CPU Pipeline

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



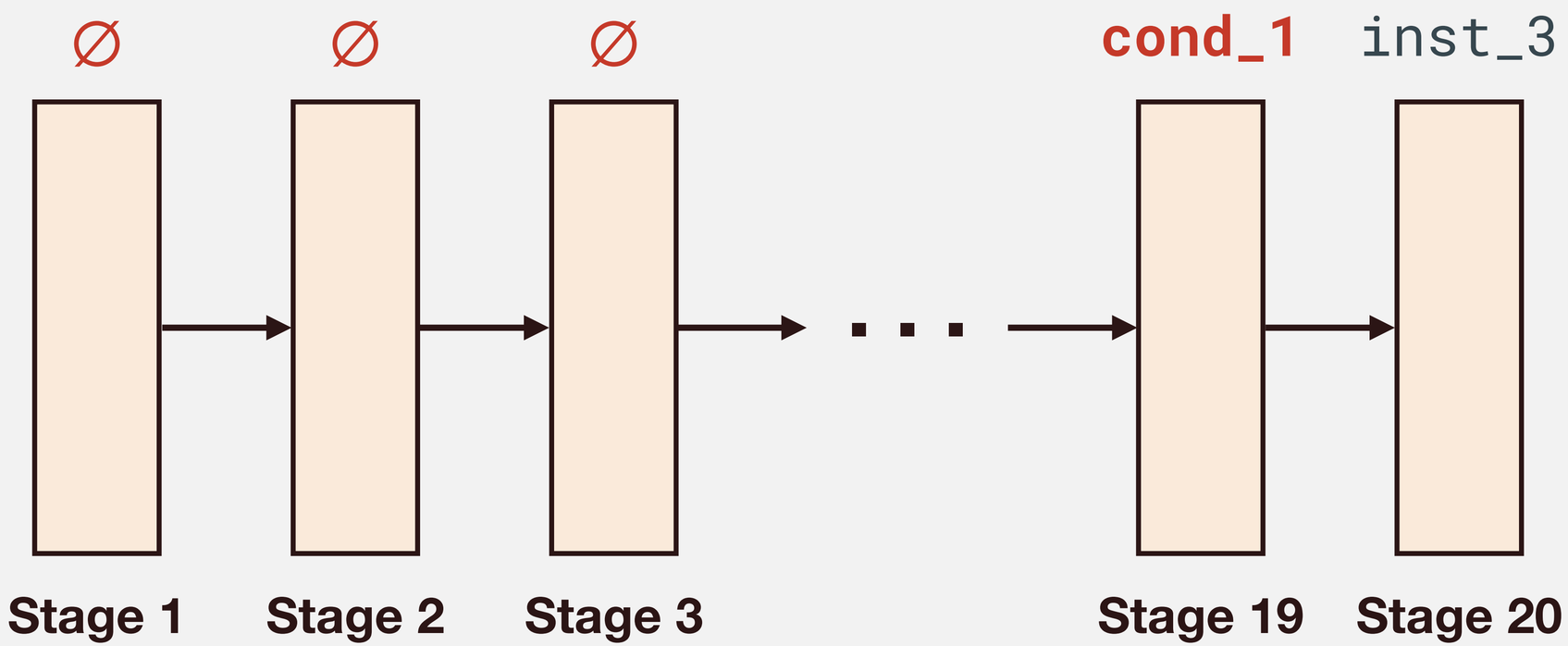
CPU Pipeline

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



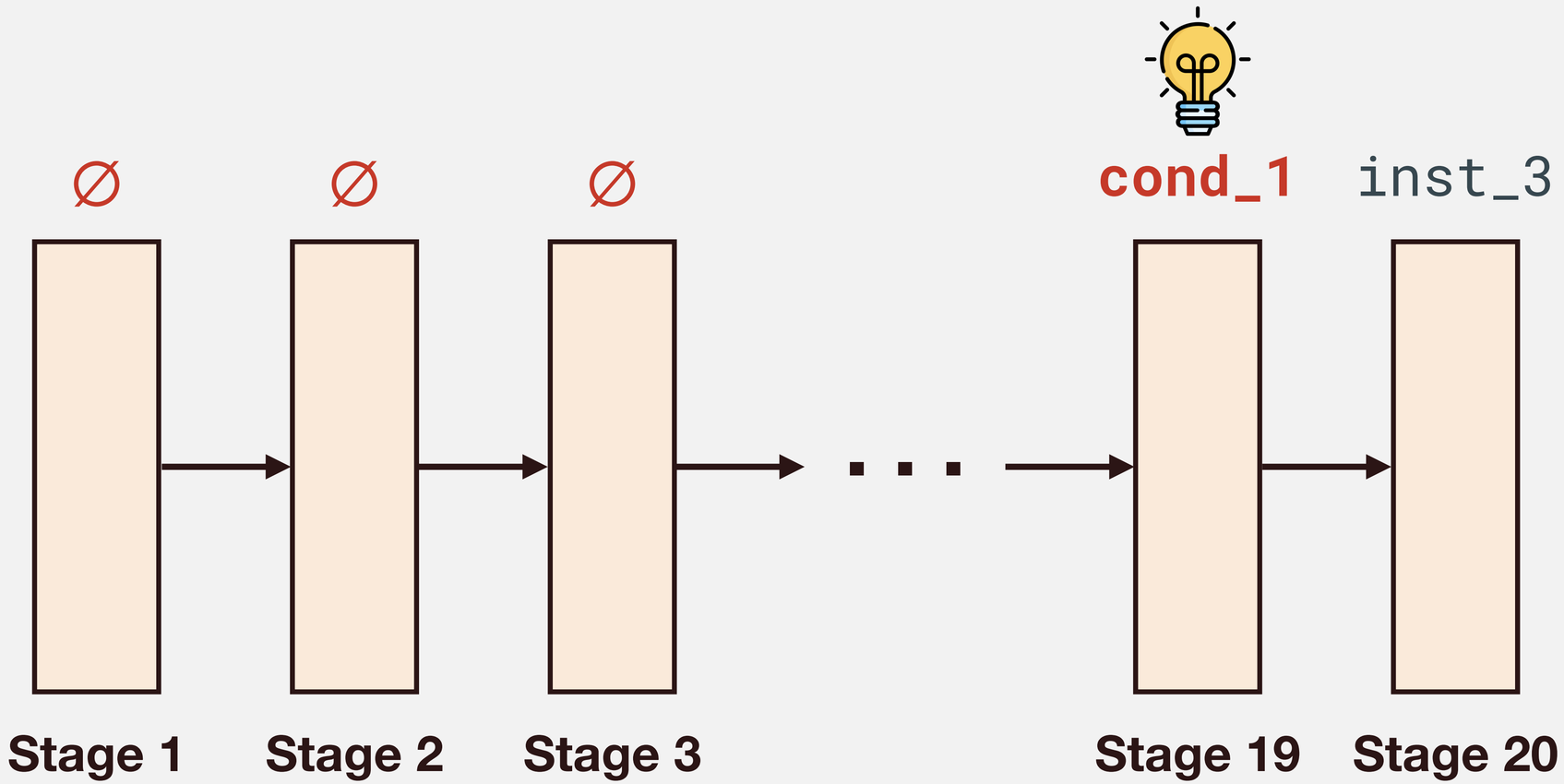
CPU Pipeline

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



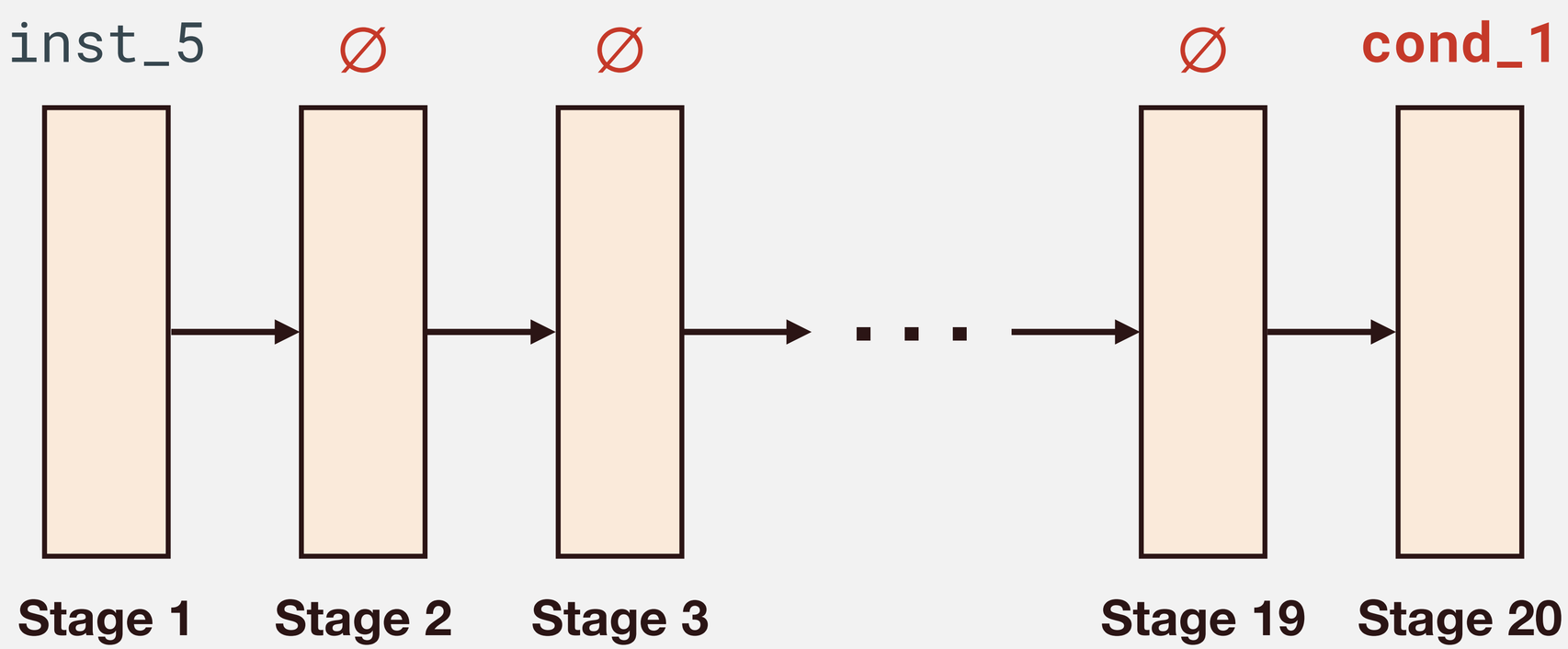
CPU Pipeline

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



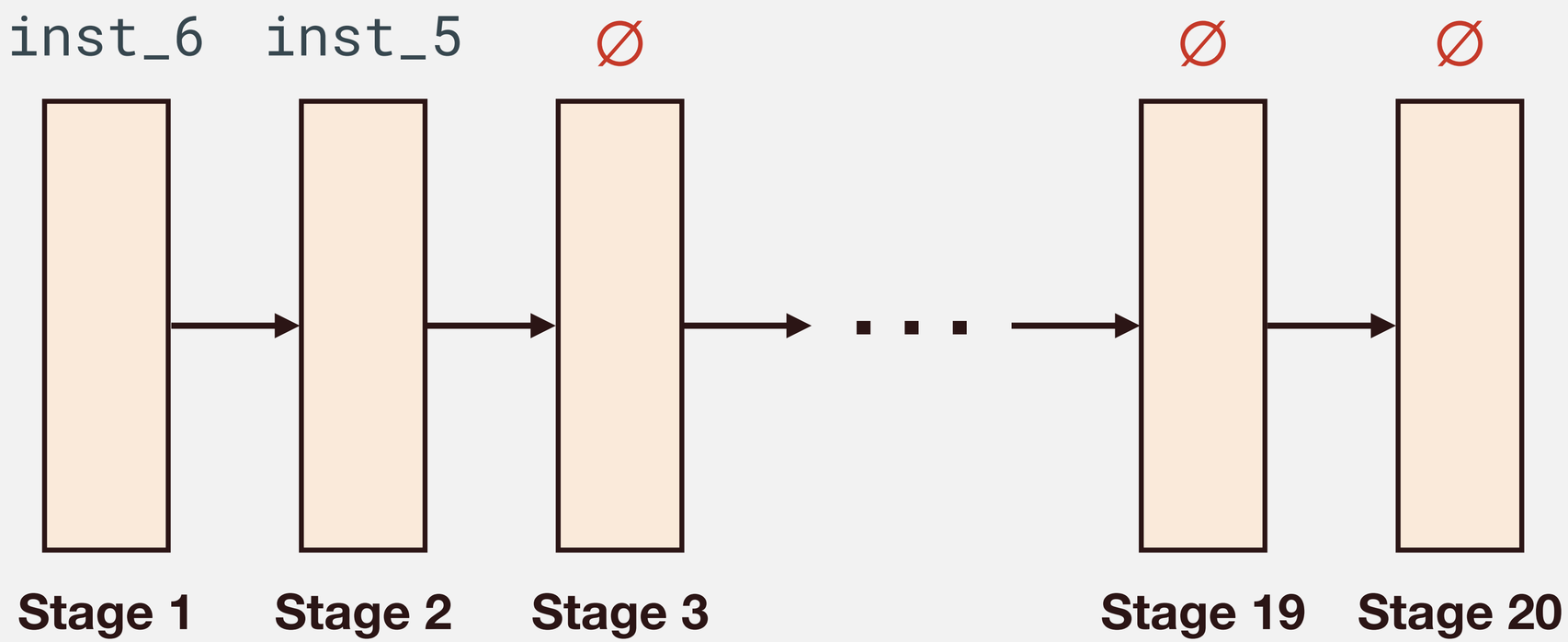
CPU Pipeline

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



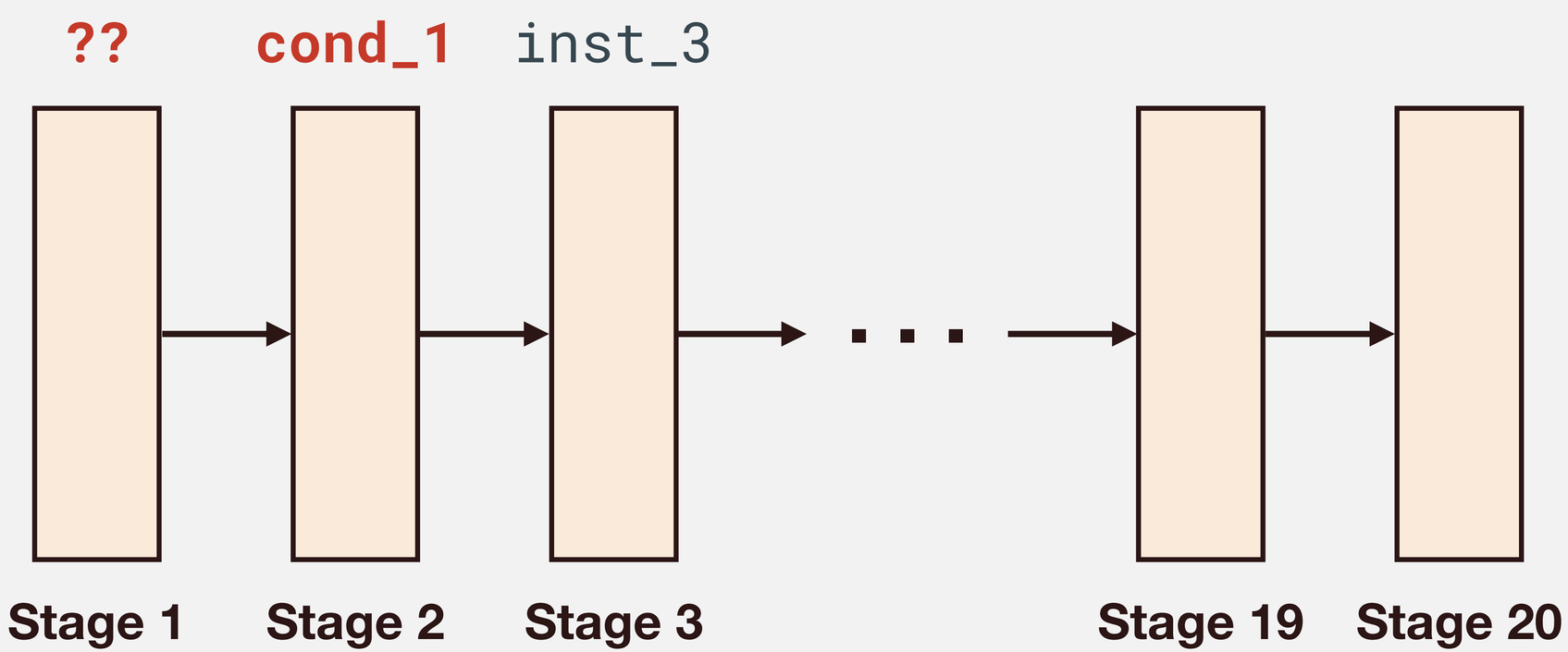
CPU Pipeline

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



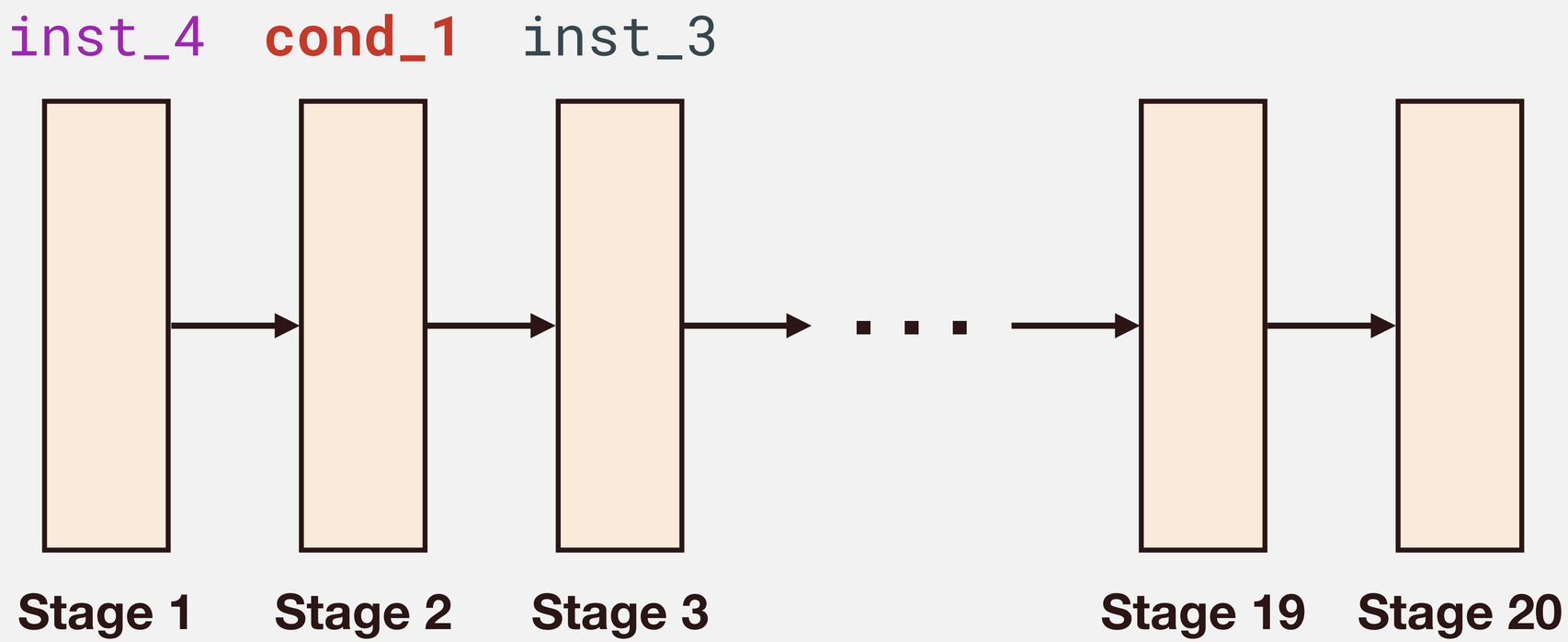
Branch Prediction

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



Branch Prediction

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```

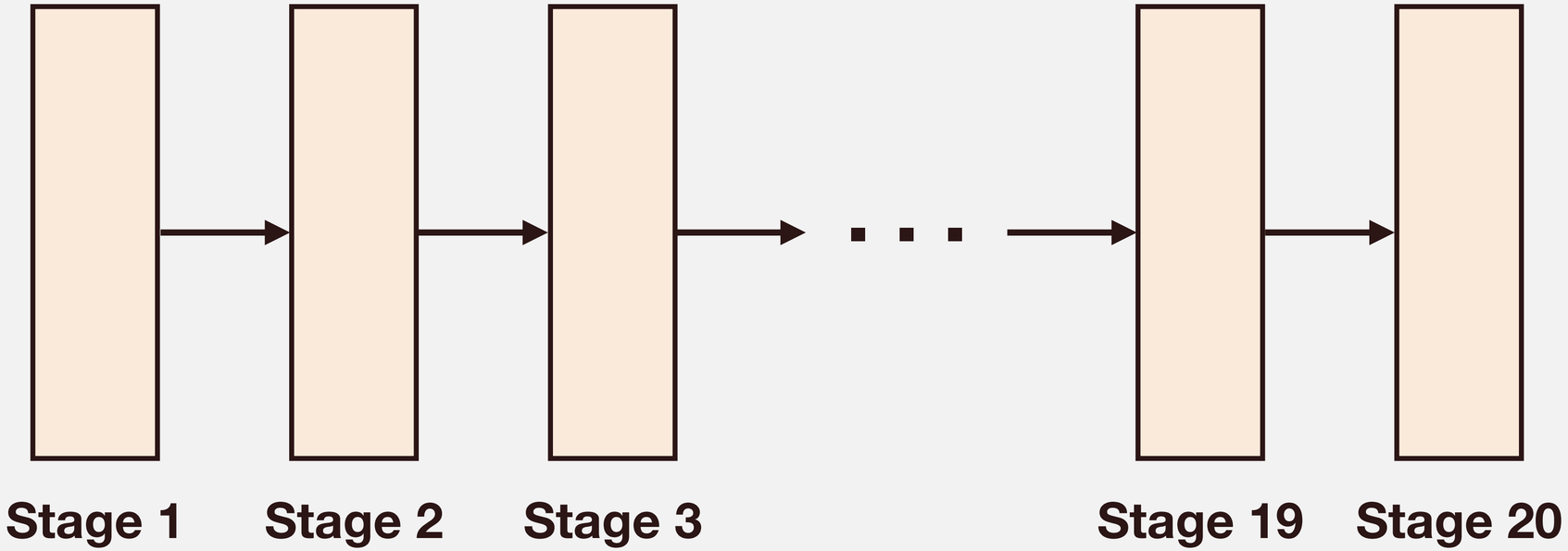


Branch Prediction

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```

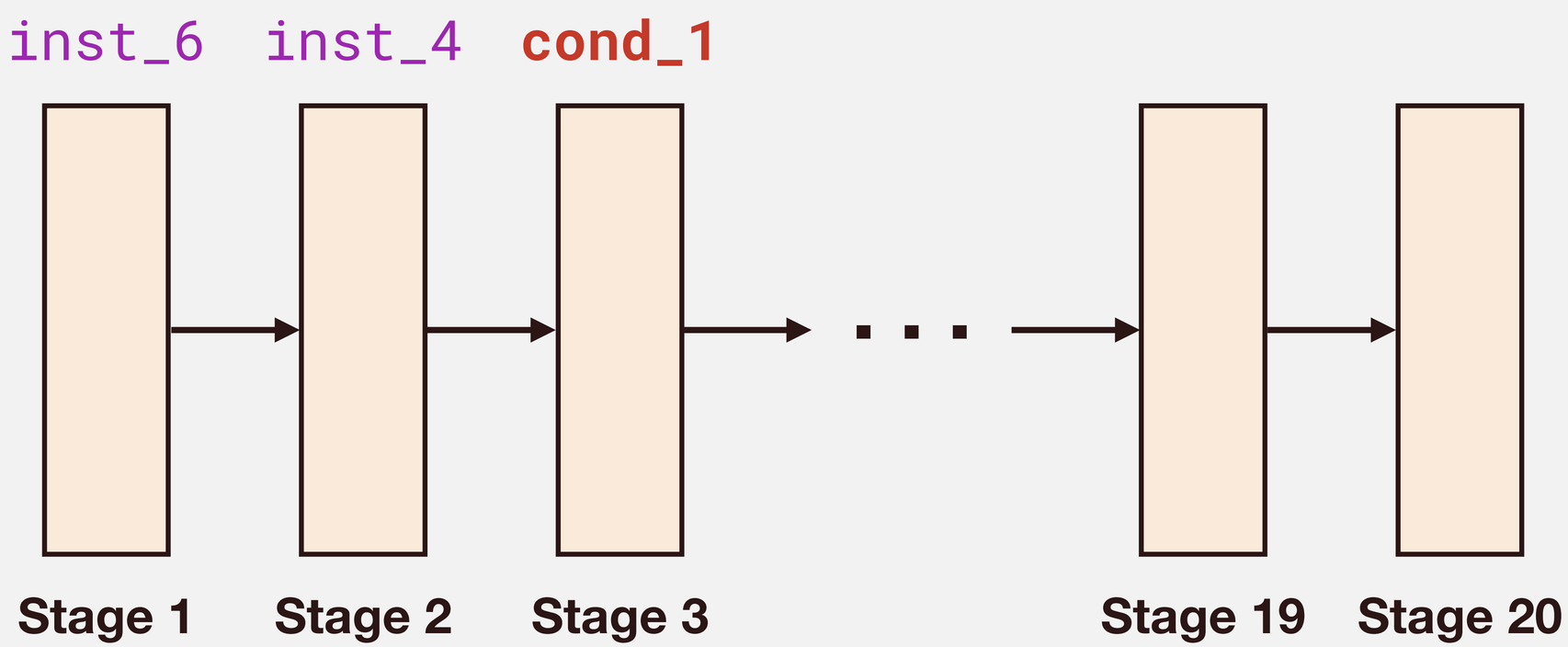
Speculative Execution

```
inst_4  cond_1  inst_3
```



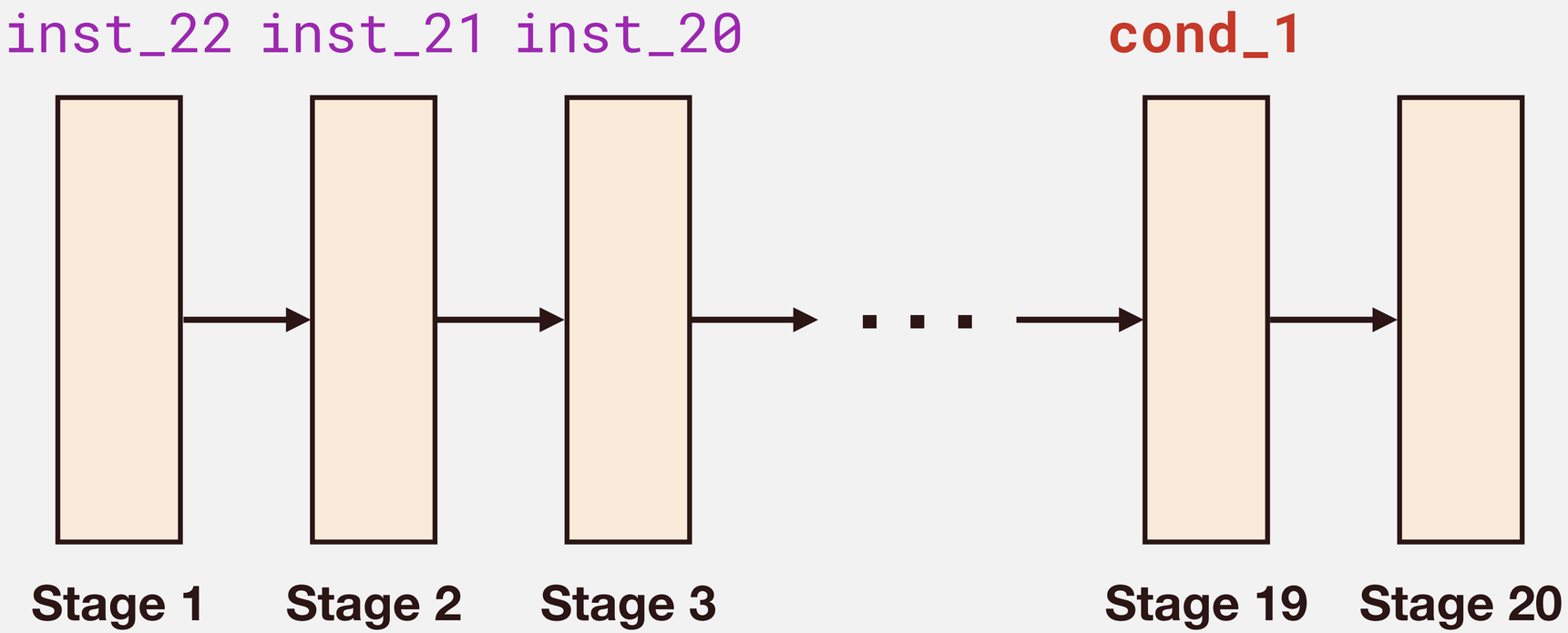
Branch Prediction

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



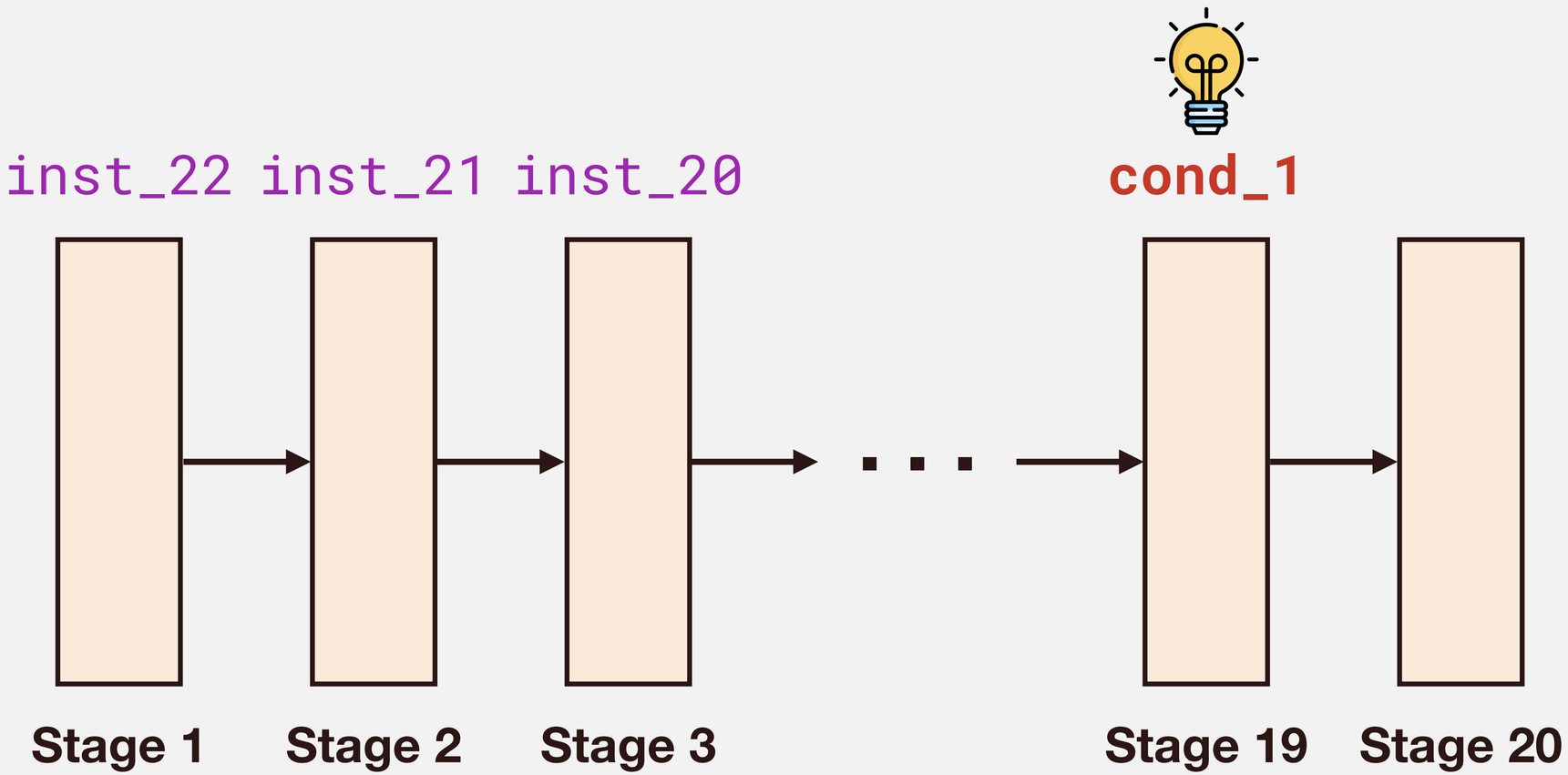
Branch Prediction

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



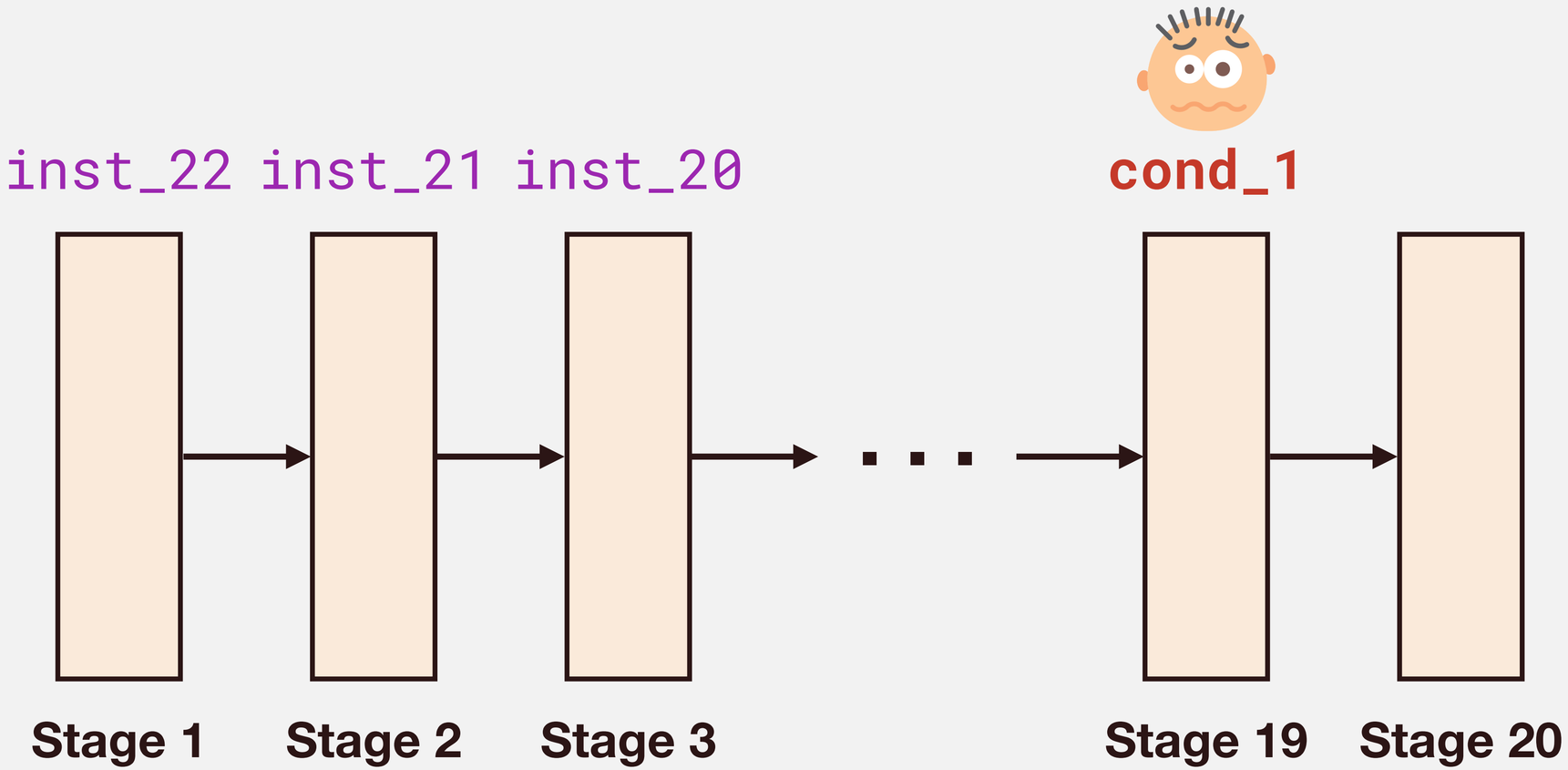
Branch Prediction

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



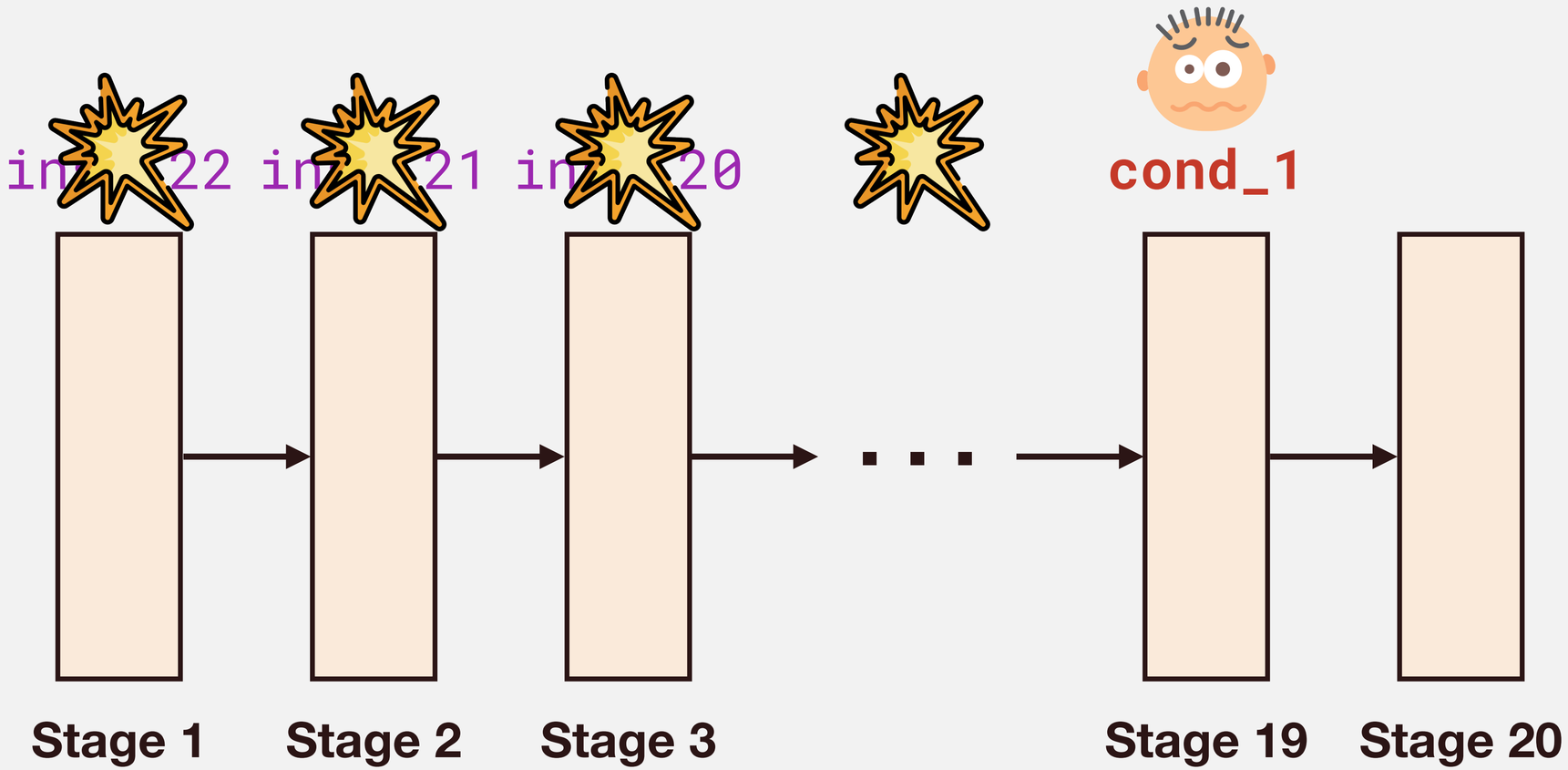
Branch Prediction

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



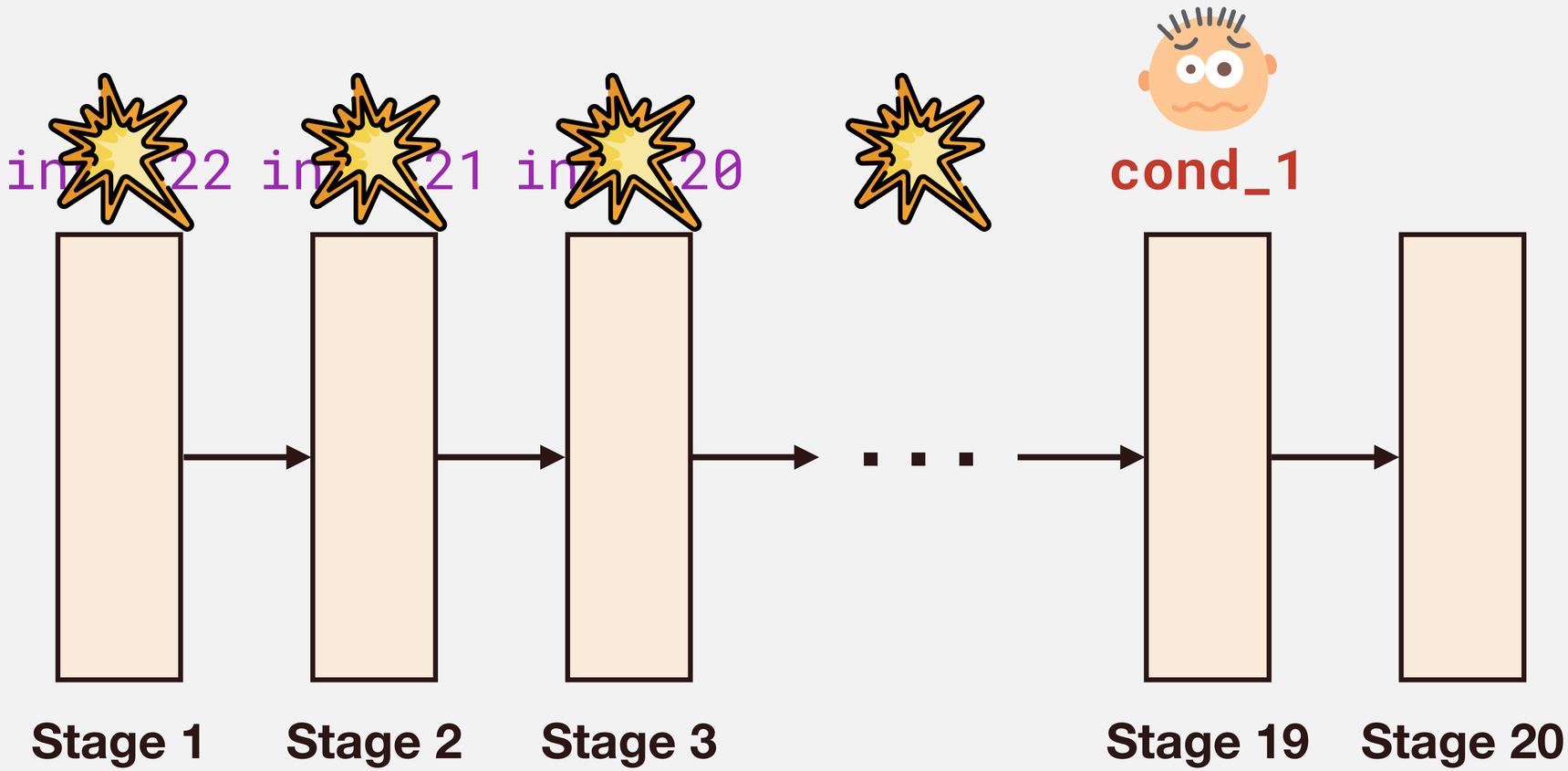
Branch Prediction

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```



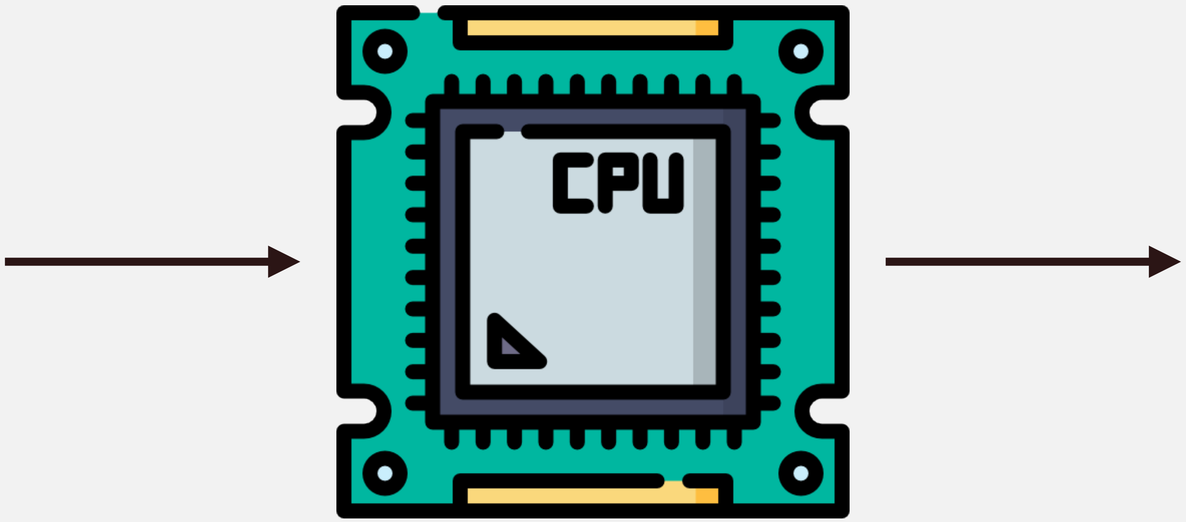
Branch Prediction

```
inst_1
inst_2
inst_3
if (cond_1)
  inst_4
else
  inst_5
inst_6
...
inst_N
```

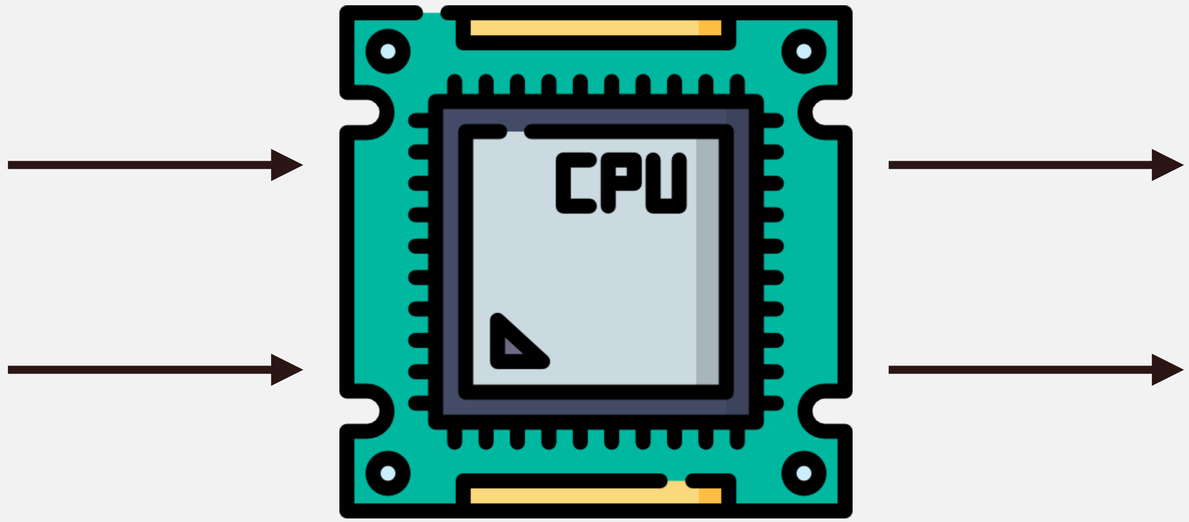


A Branch Misprediction costs ~10-20 CPU cycles

CPU Parallelism

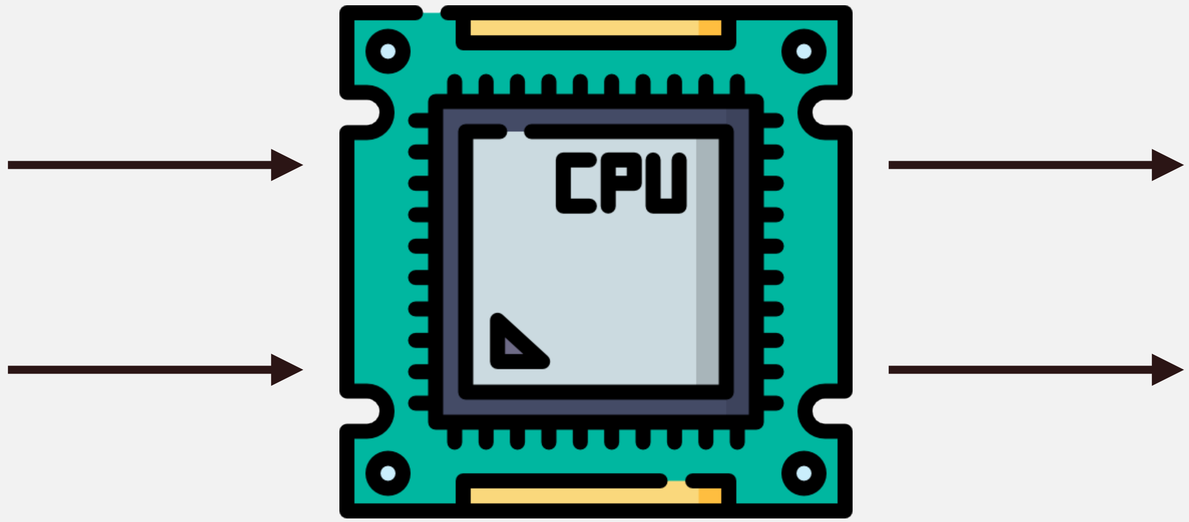


CPU Parallelism



Superscalar

CPU Parallelism

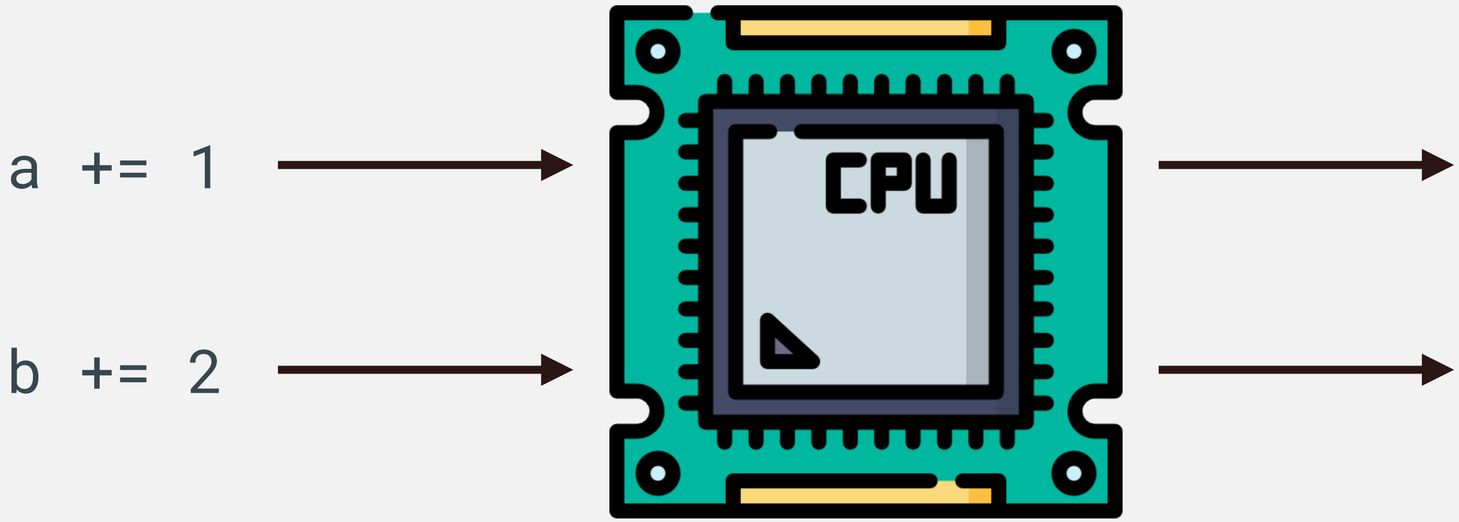


Superscalar

```
a += 1;
b += 2;
c += 3;
d += 4;
```

CPU Parallelism

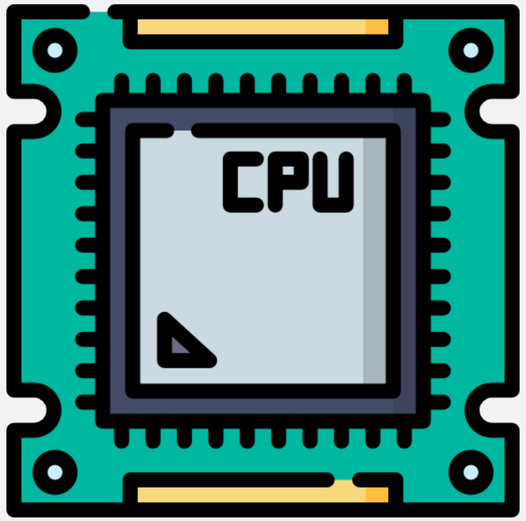
```
a += 1;
b += 2;
c += 3;
d += 4;
```



Superscalar

CPU Parallelism

```
c += 3    a += 1
d += 4    b += 2
```

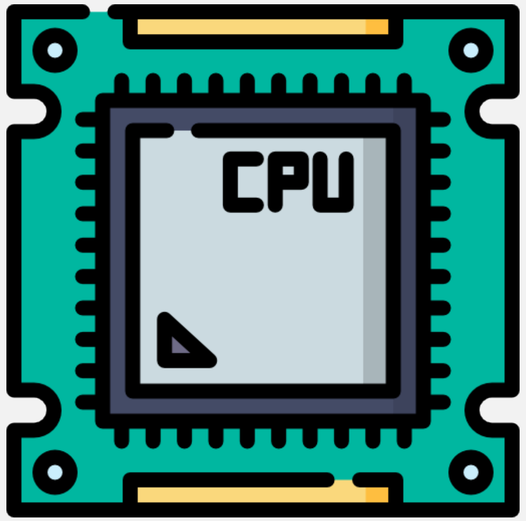



```
a += 1;
b += 2;
c += 3;
d += 4;
```

Superscalar

CPU Parallelism

```
d += 4    c += 3
a += 1    b += 2
```

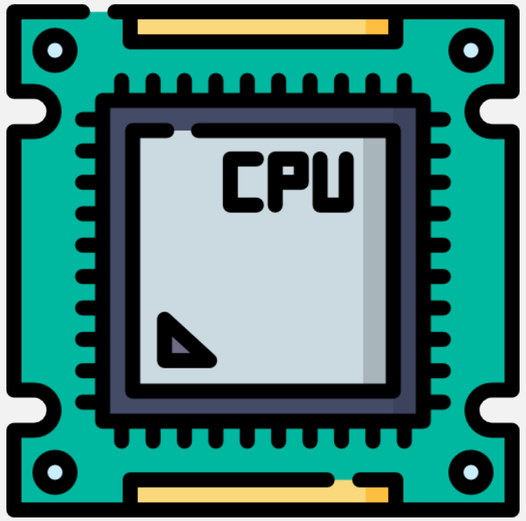



```
c += 3;
b += 2;
d += 4;
a += 1;
```

Superscalar

CPU Parallelism

```
d += 4    c += 3
a += 1    b += 2
```

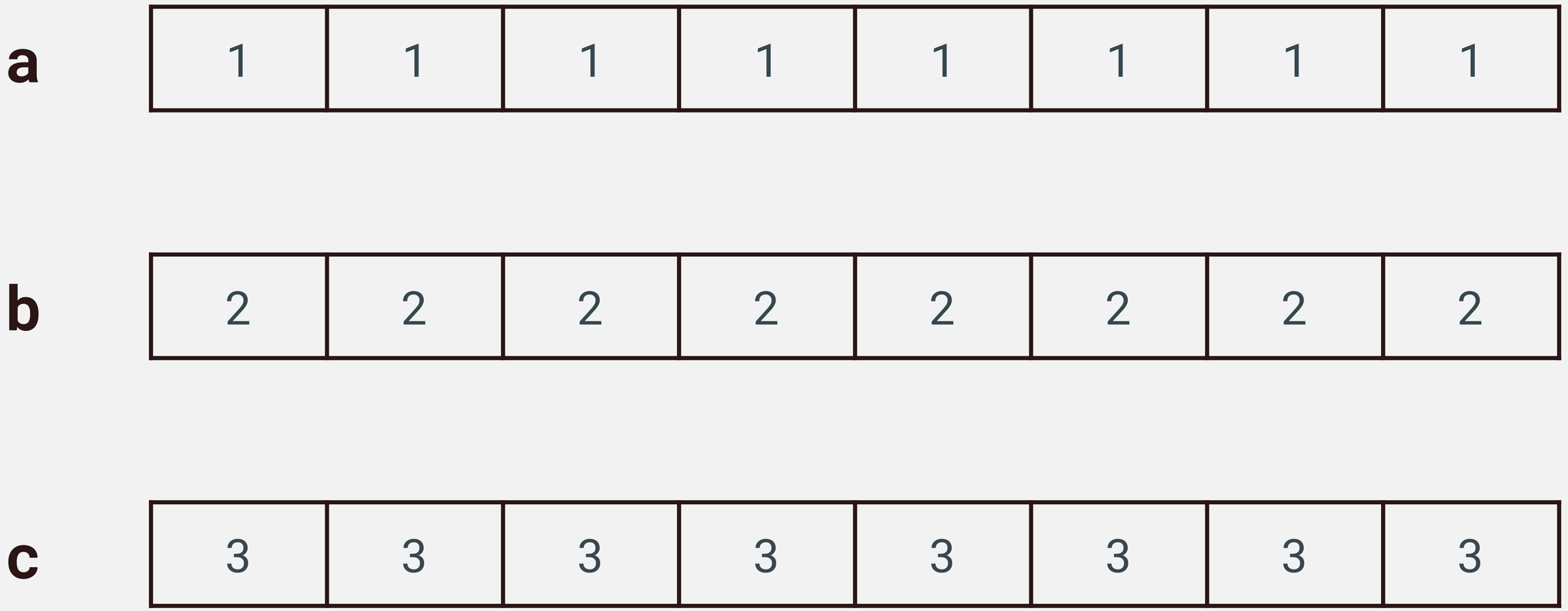



```
c += 3;
b += 2;
d += 4;
a += 1;
```

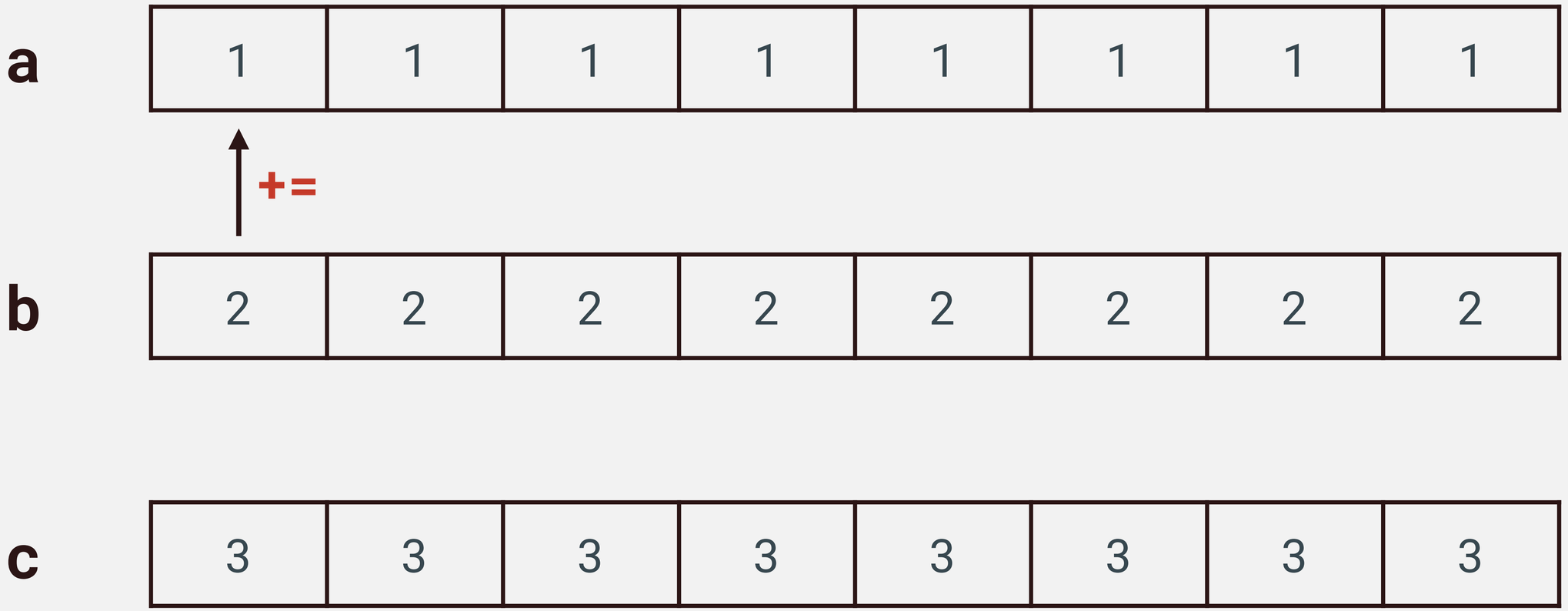
Superscalar

Out-of-Order

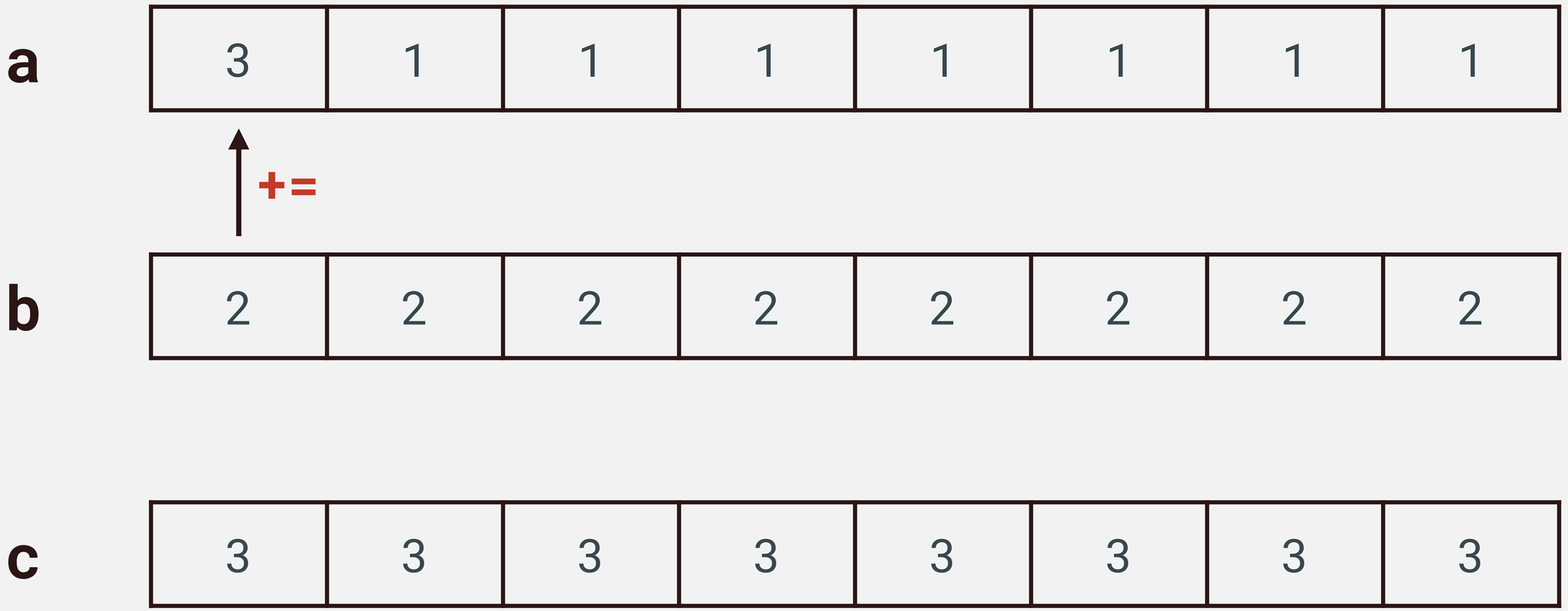
Data Dependency



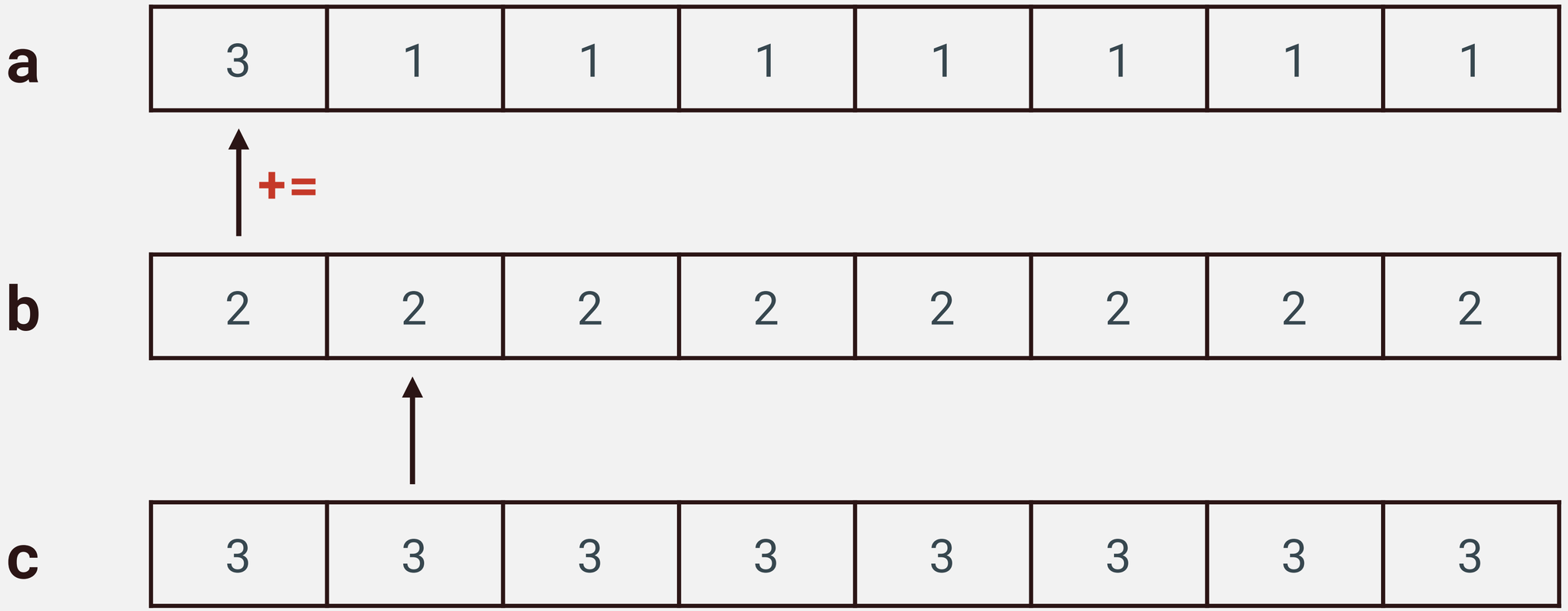
Data Dependency



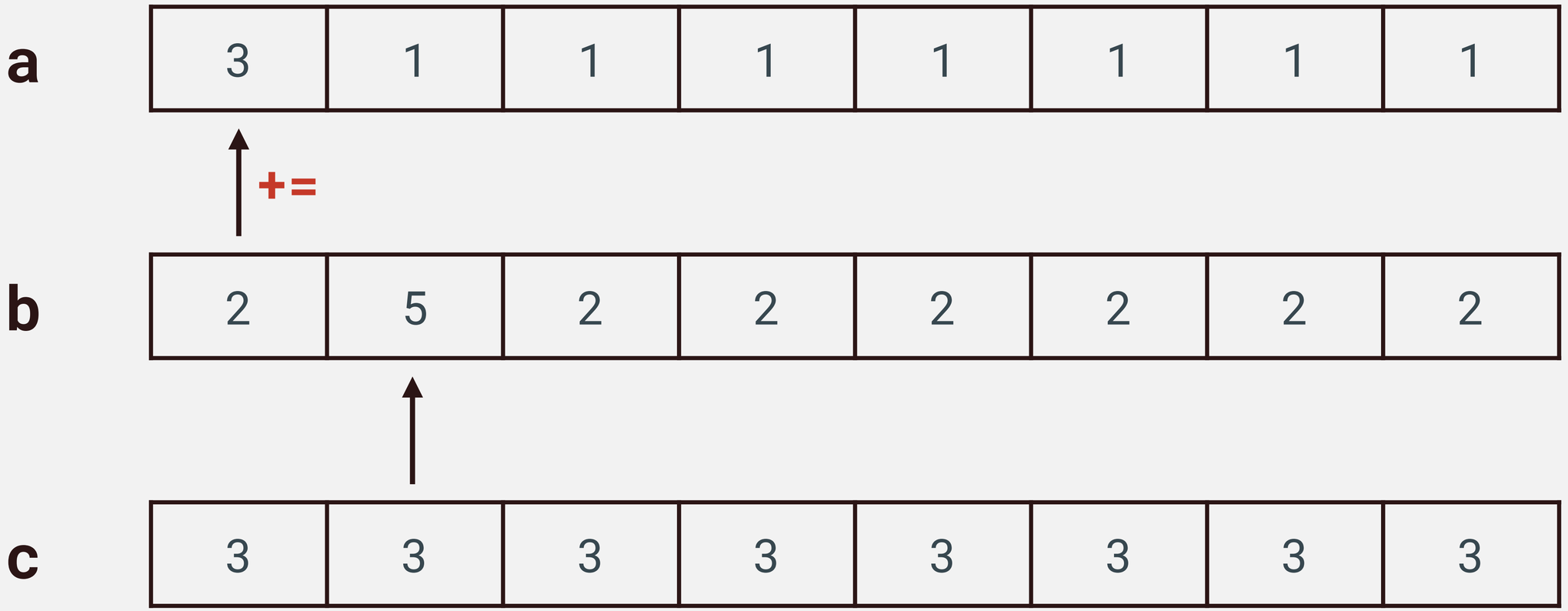
Data Dependency



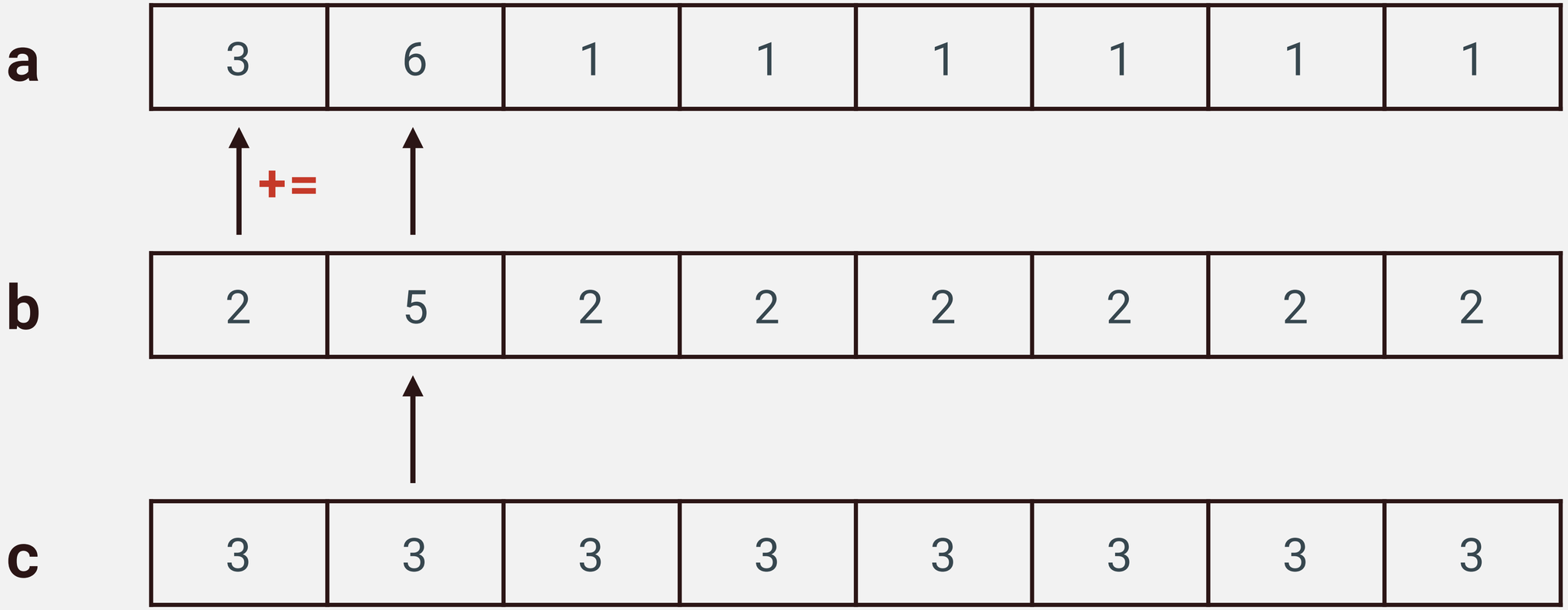
Data Dependency



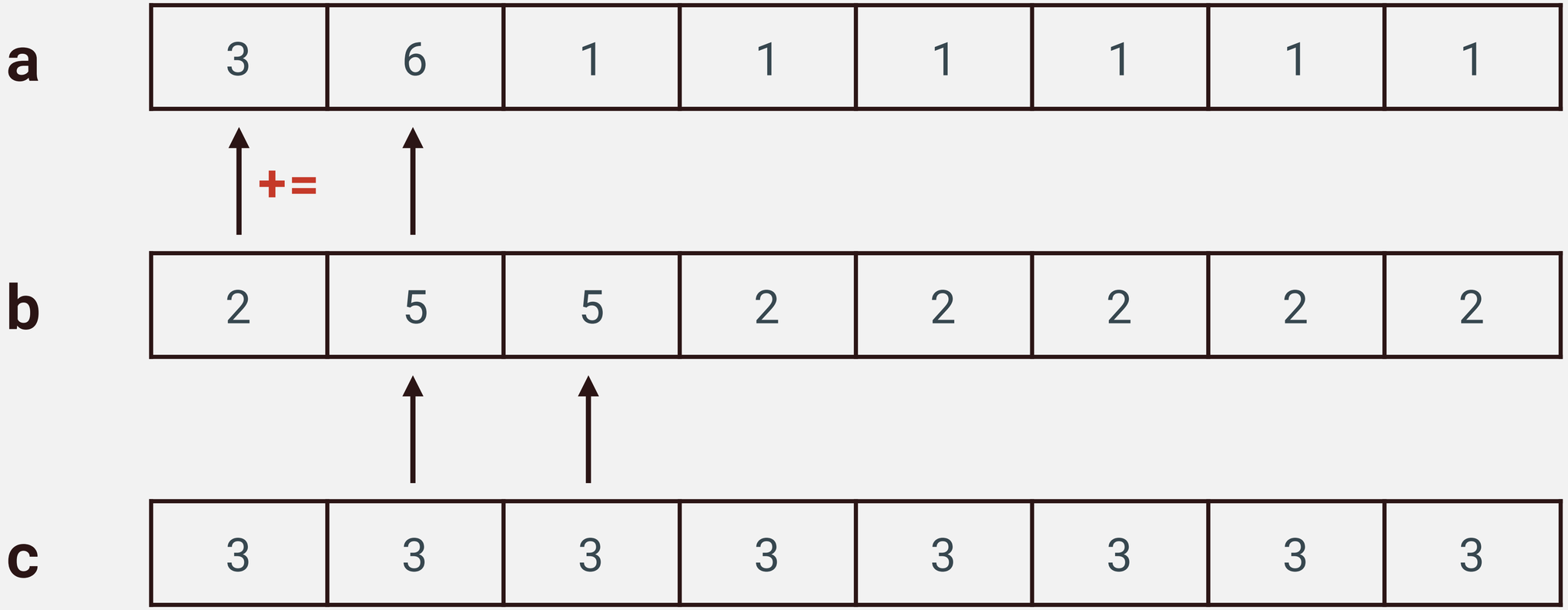
Data Dependency



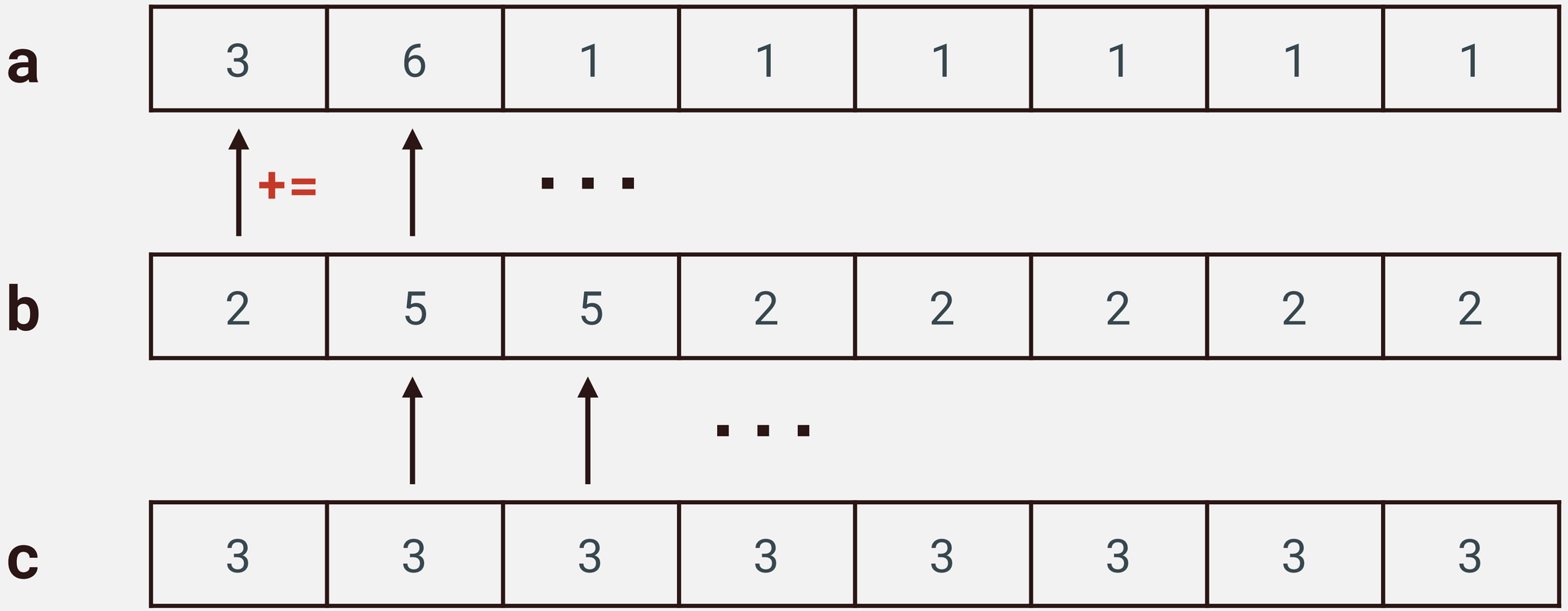
Data Dependency



Data Dependency



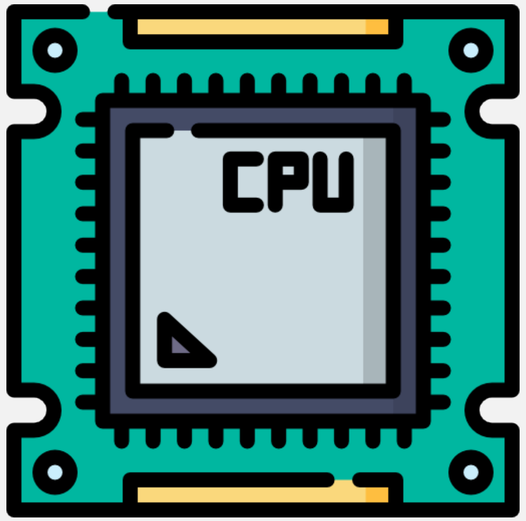
Data Dependency



CPU Parallelism

```
a += 3
b += 4
```

```
a += 1
b += 2
```




```
a += 1;
a += 3;
b += 2;
b += 4;
```

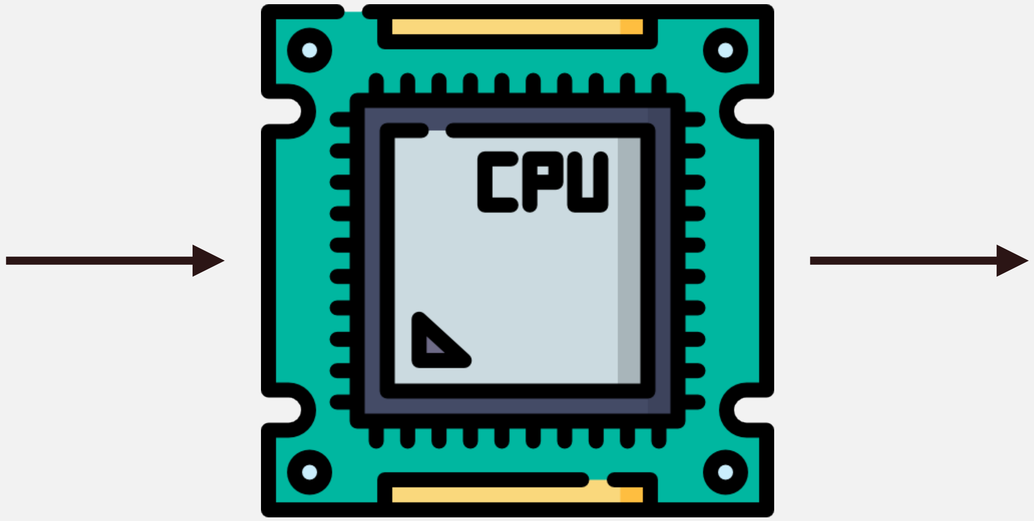
Superscalar

Out-of-Order

Loop Unrolling

- Reduces # of loop test
- Help compiler improve parallelism

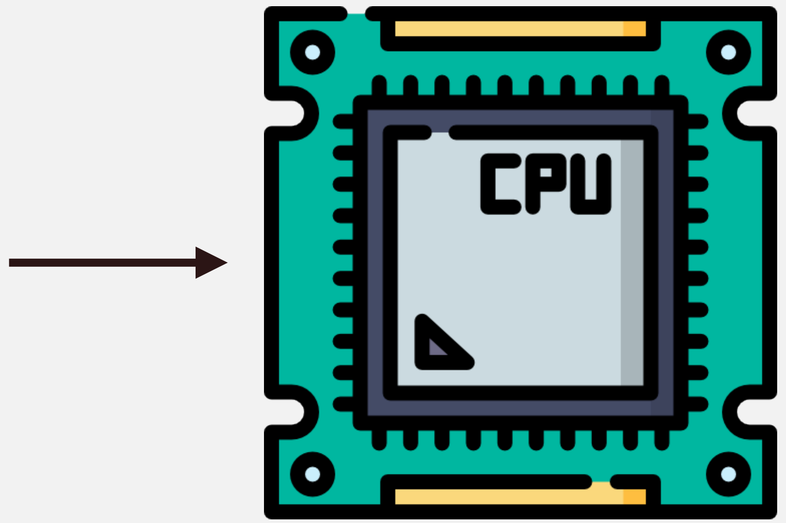
Data Parallelism (SIMD)




```
a += 1;
b += 2;
c += 3;
d += 4;
```

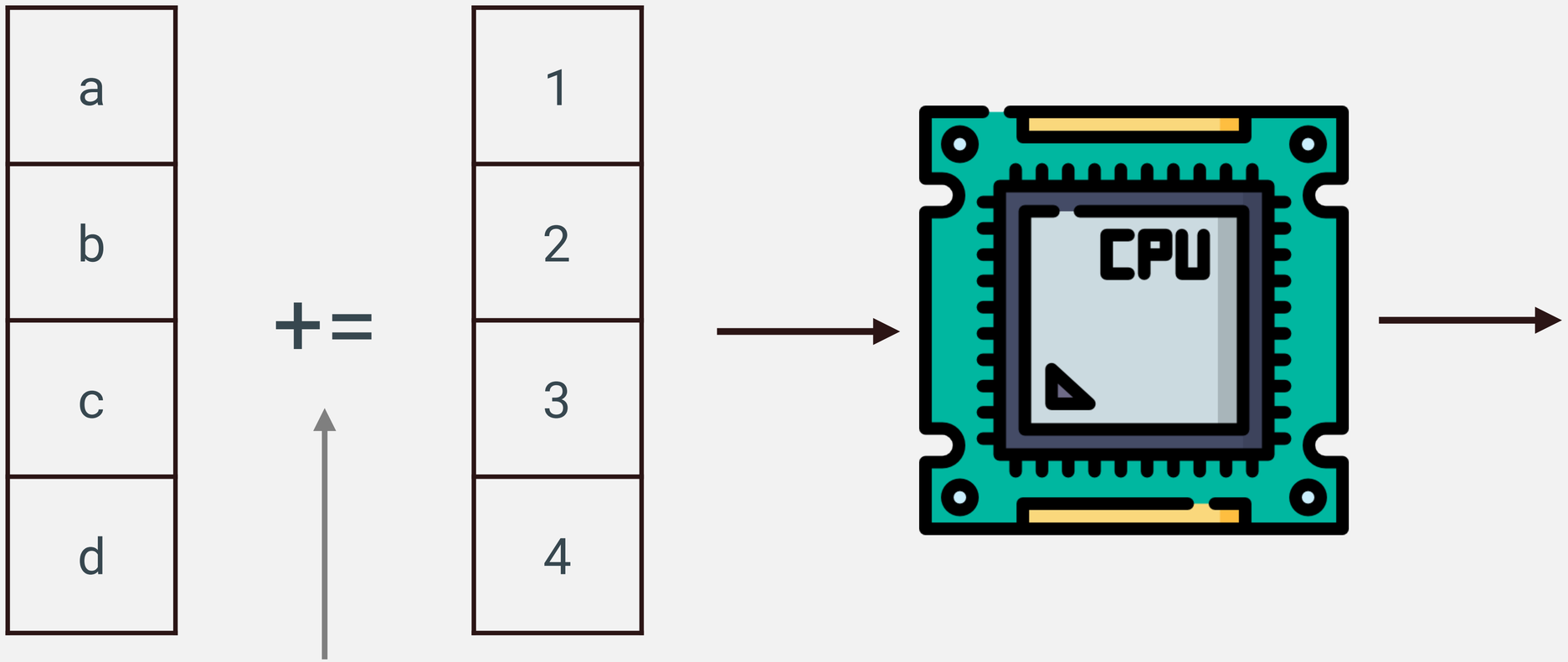
Data Parallelism (SIMD)

```
d+=4; c+=3; b+=2; a+=1;
```

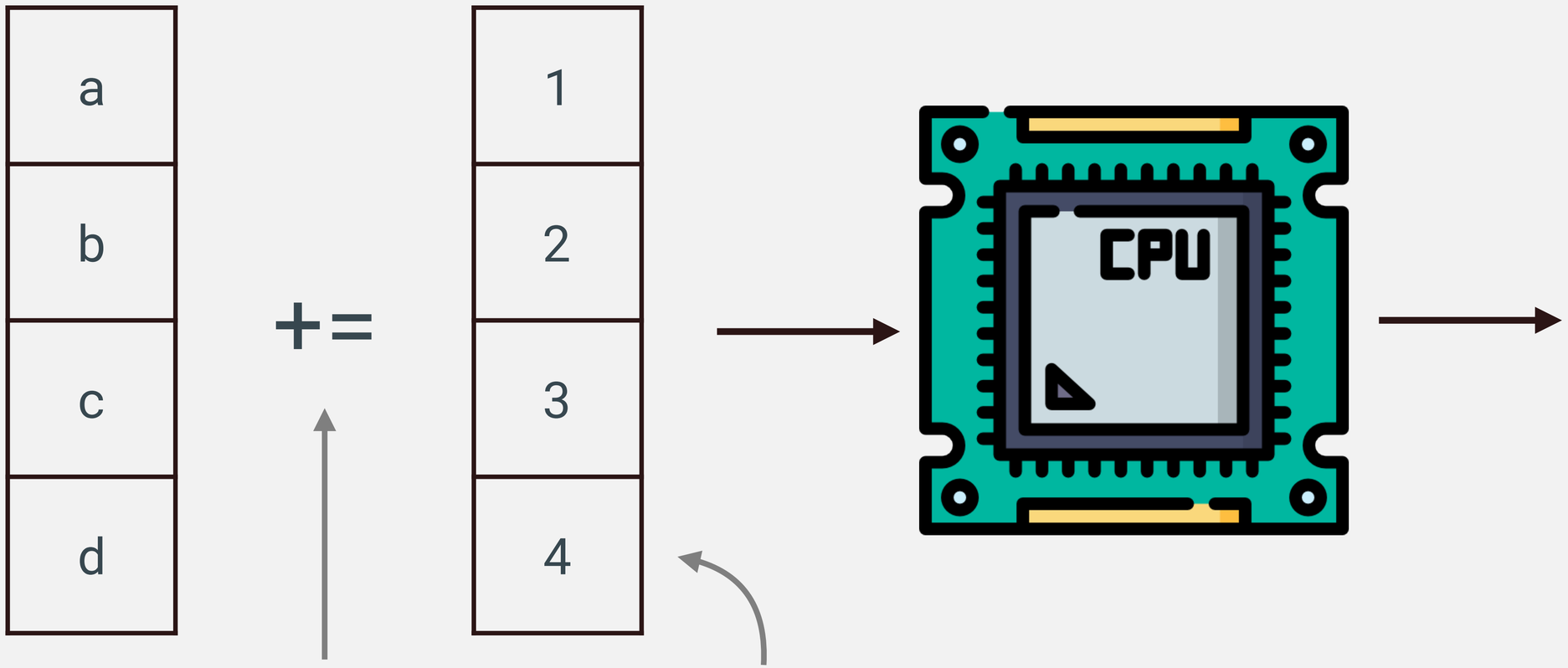



```
a += 1;
b += 2;
c += 3;
d += 4;
```

Data Parallelism (SIMD)



Data Parallelism (SIMD)

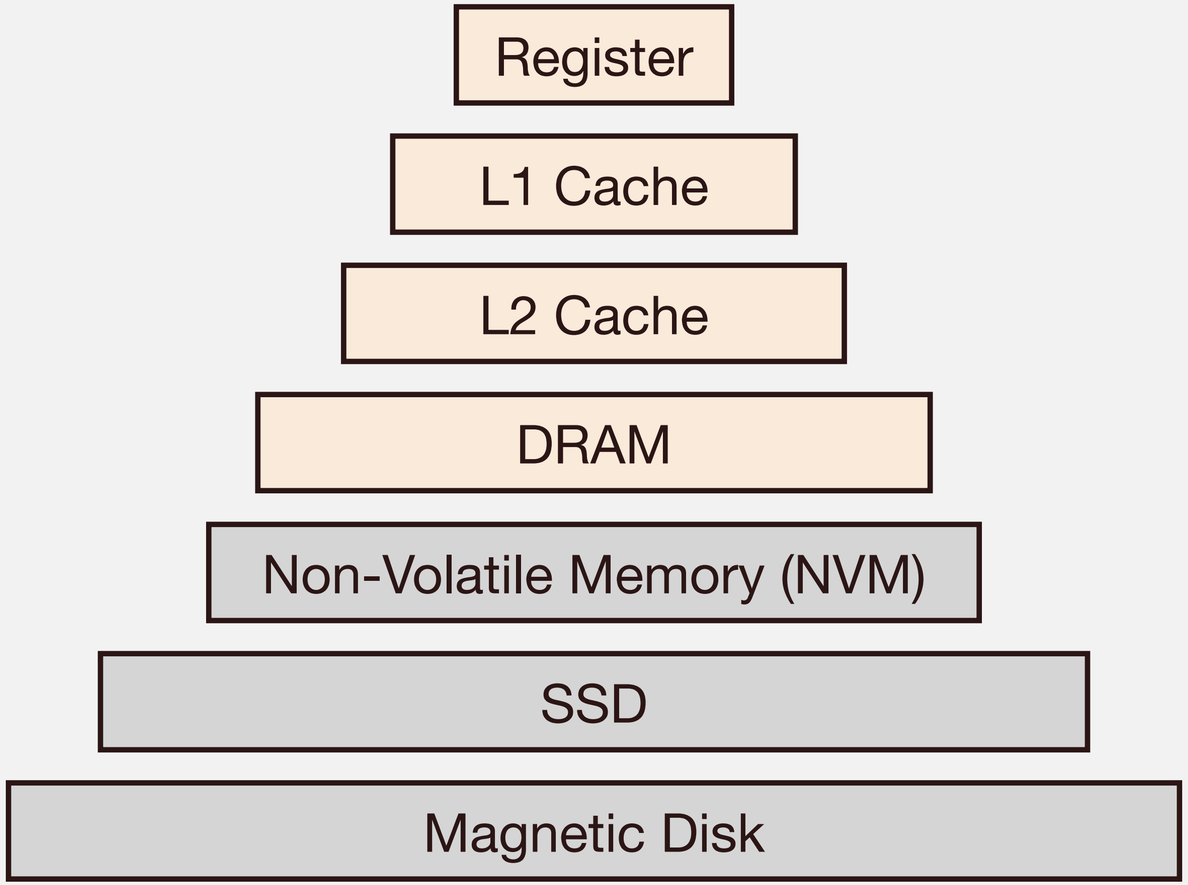


Low-Level Optimization Techniques

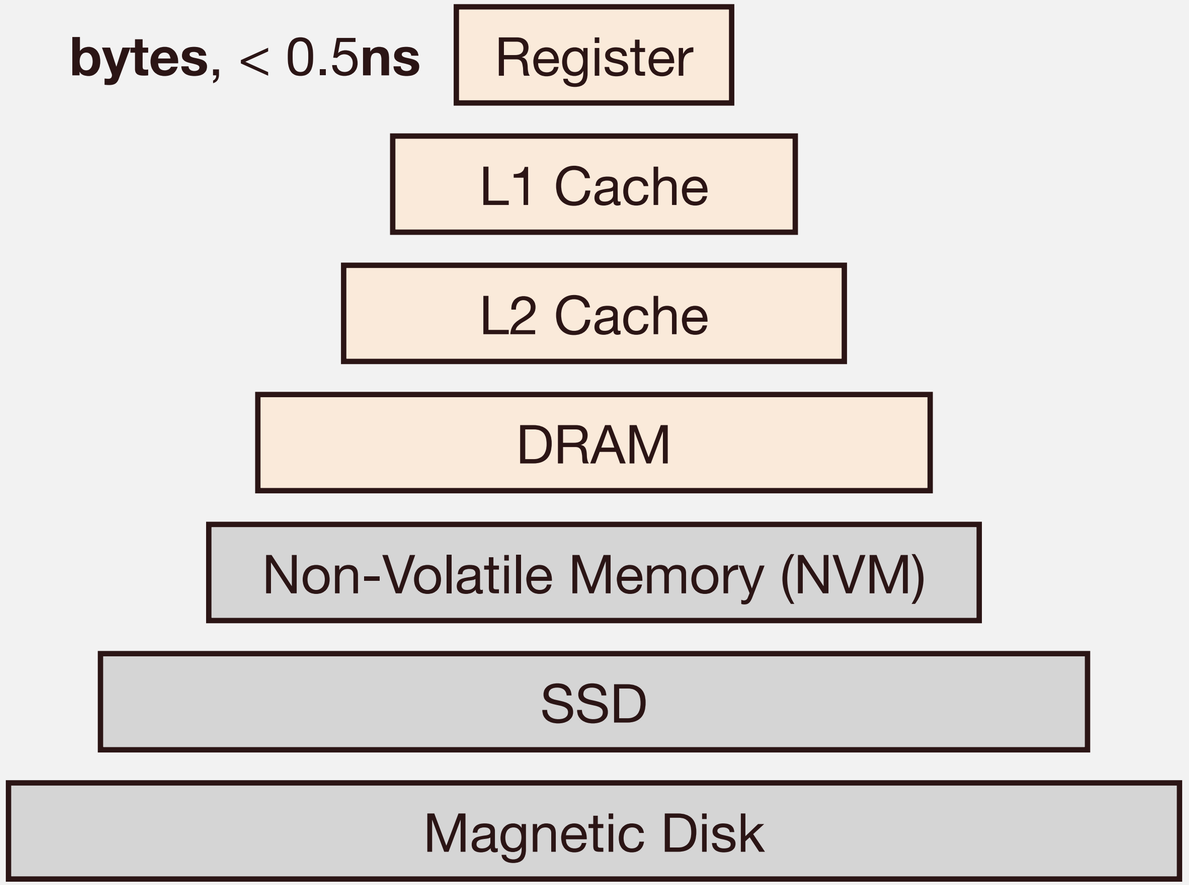
CPU

- ➜ Reduce Operator Strength
- ➜ Reduce Branch Misprediction
- ➜ Reduce Data Dependency
- ➜ SIMD

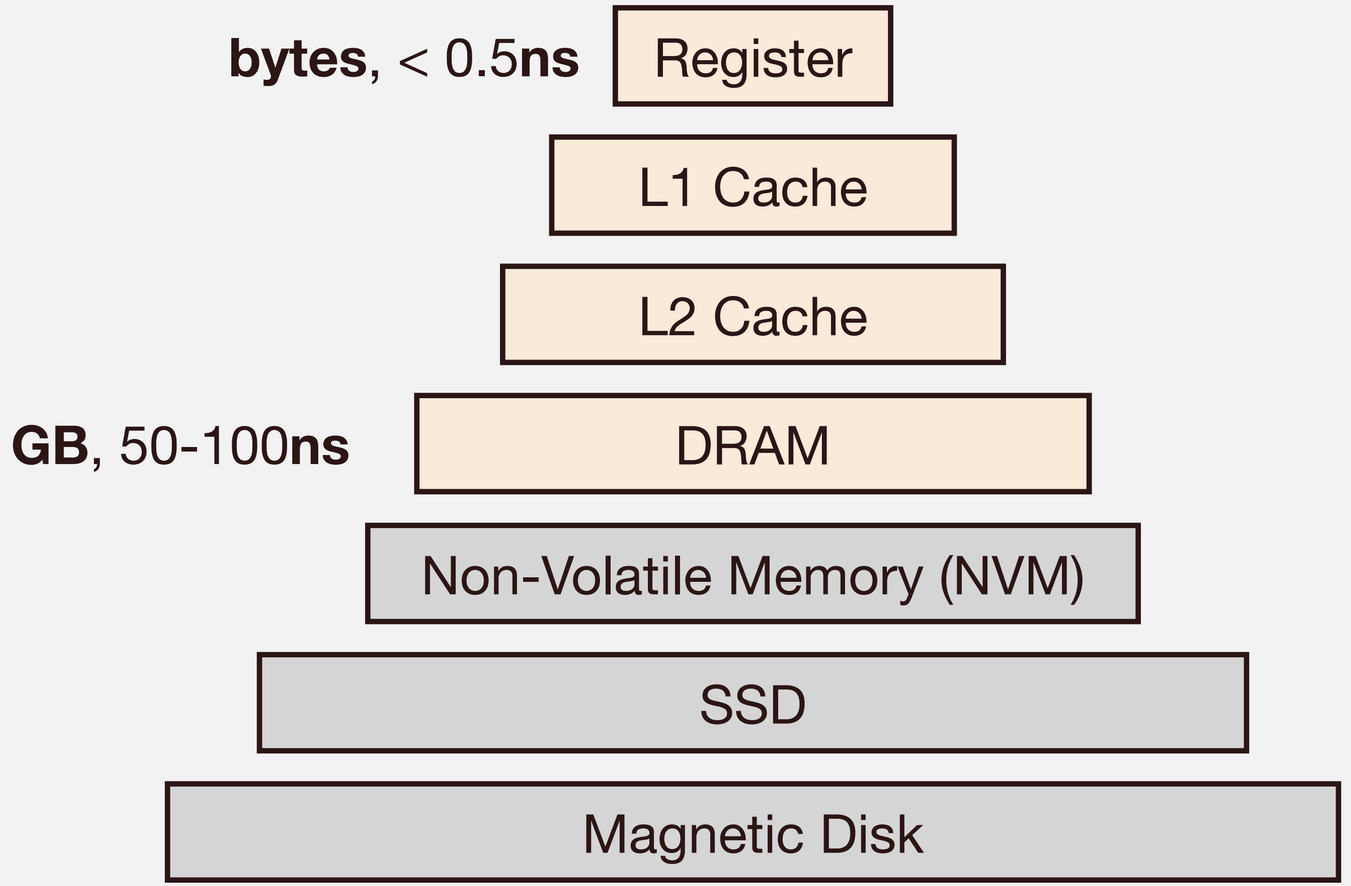
Storage Hierarchy



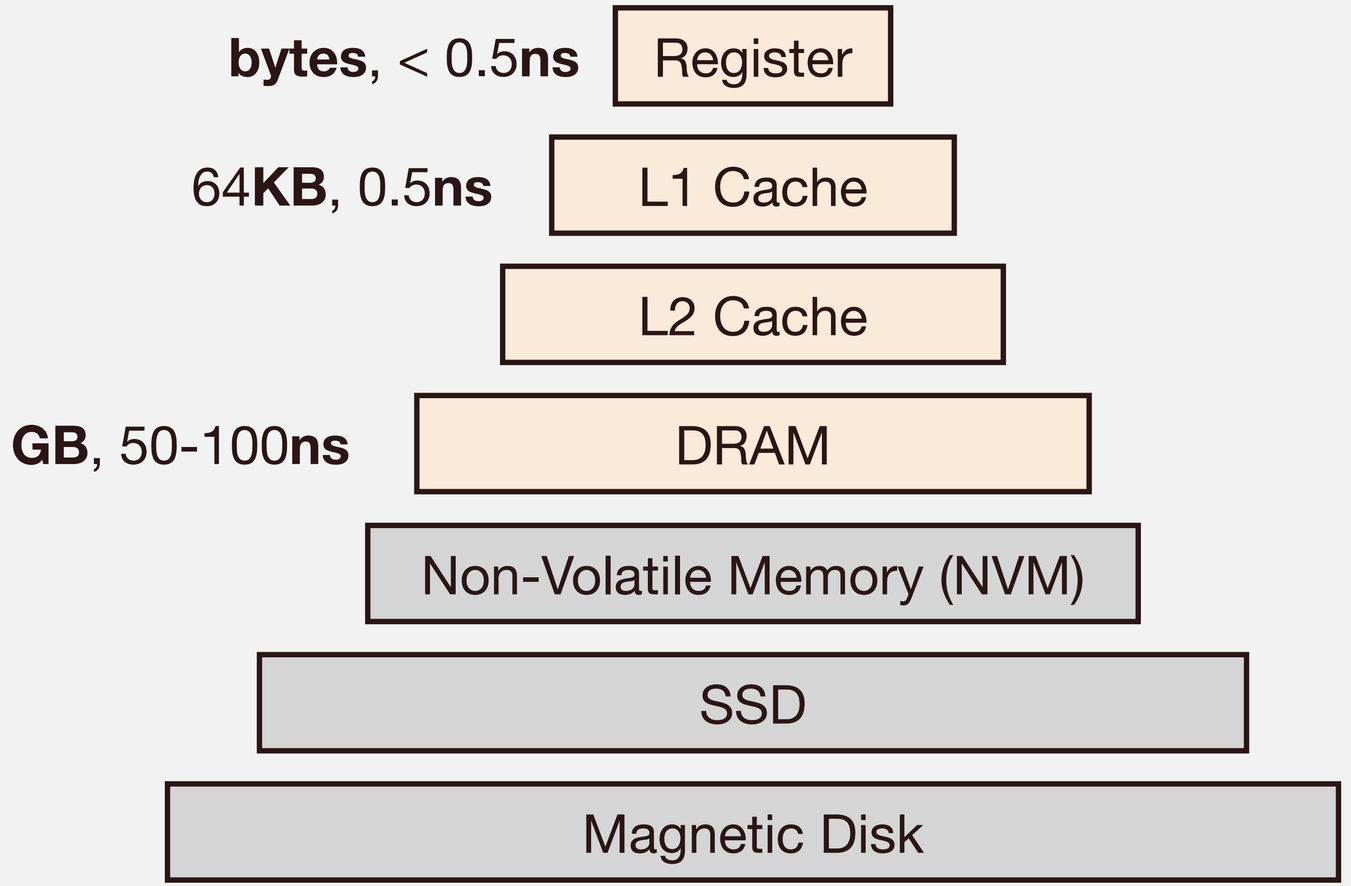
Storage Hierarchy



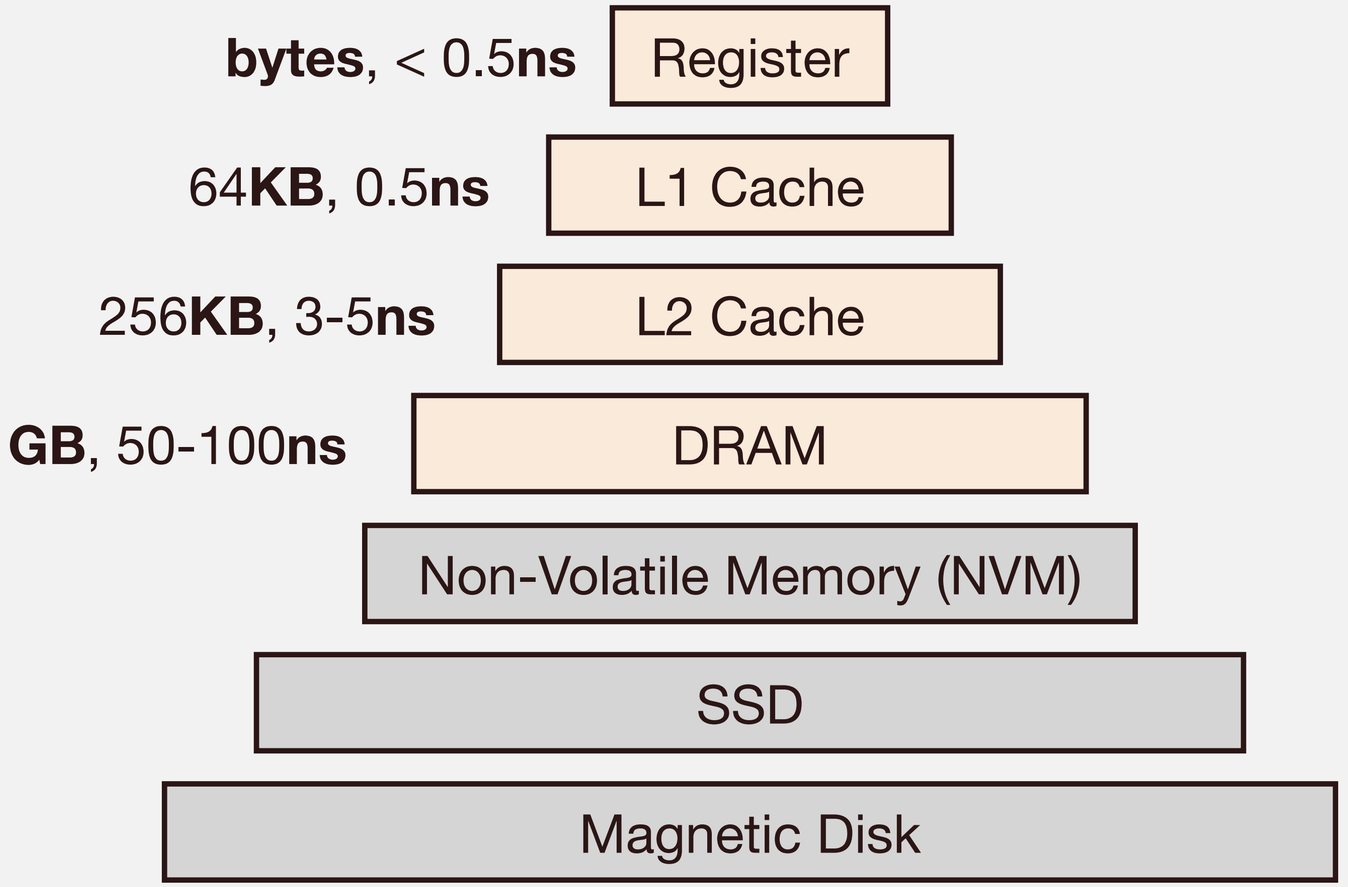
Storage Hierarchy



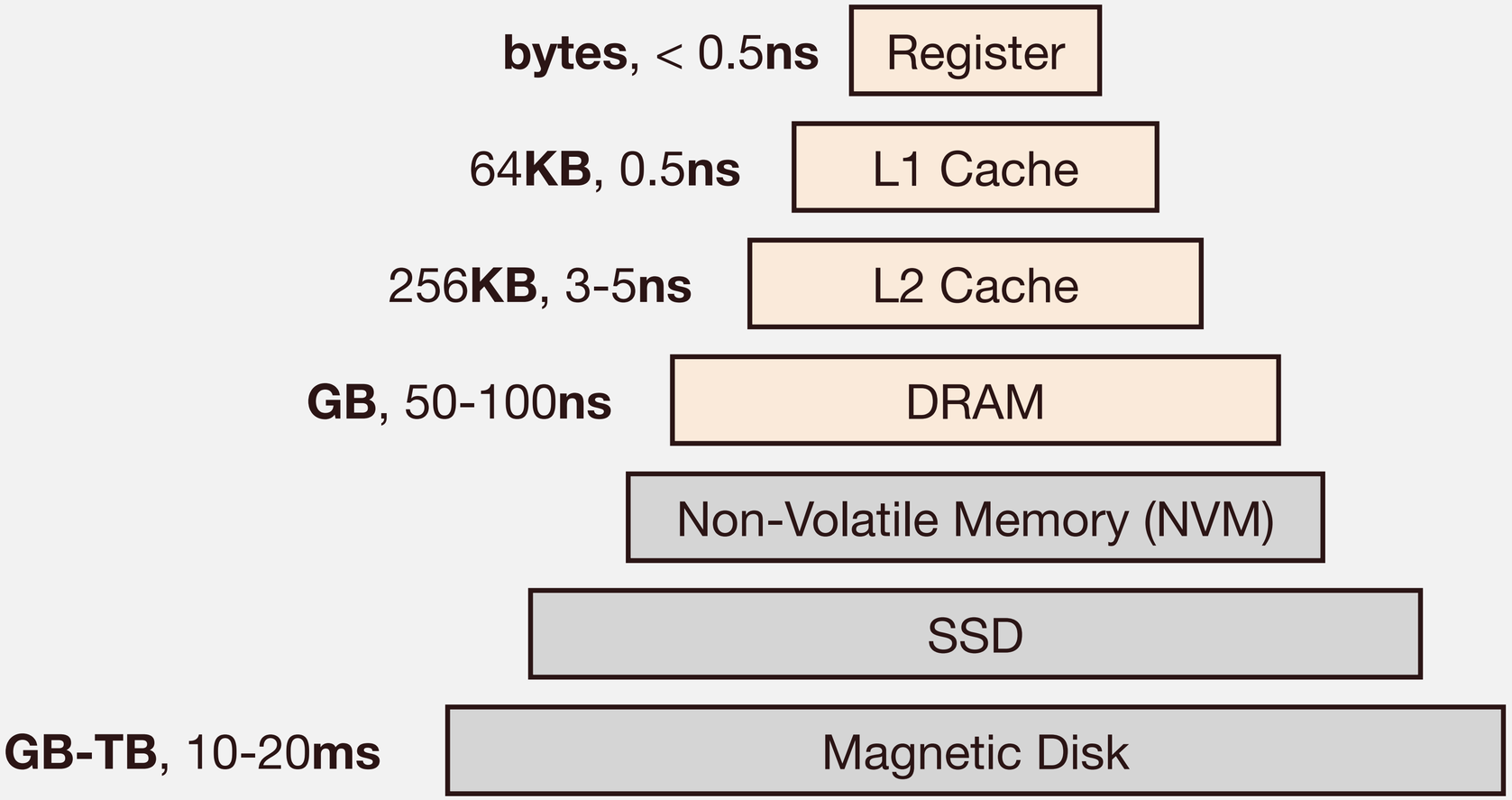
Storage Hierarchy



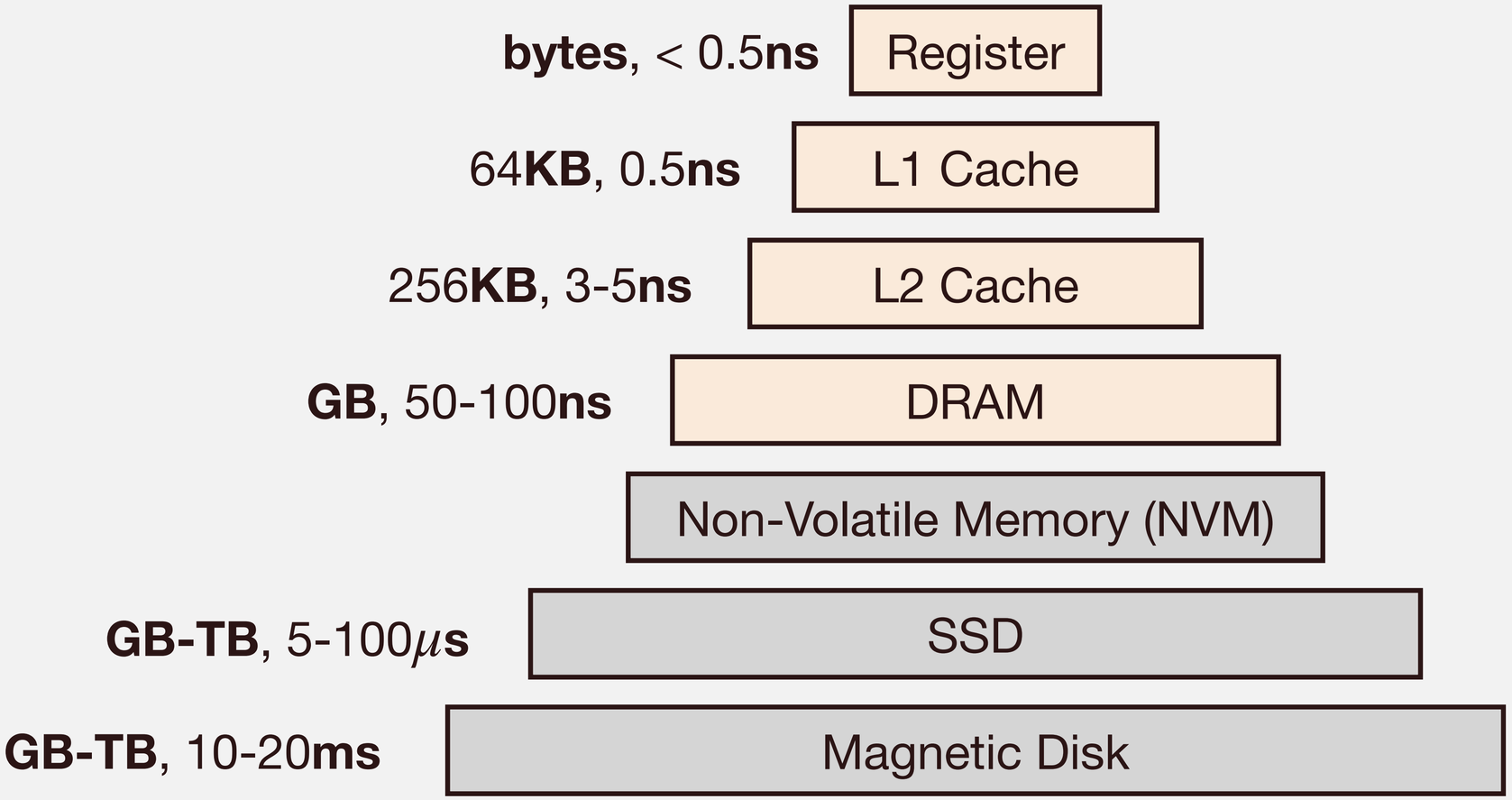
Storage Hierarchy



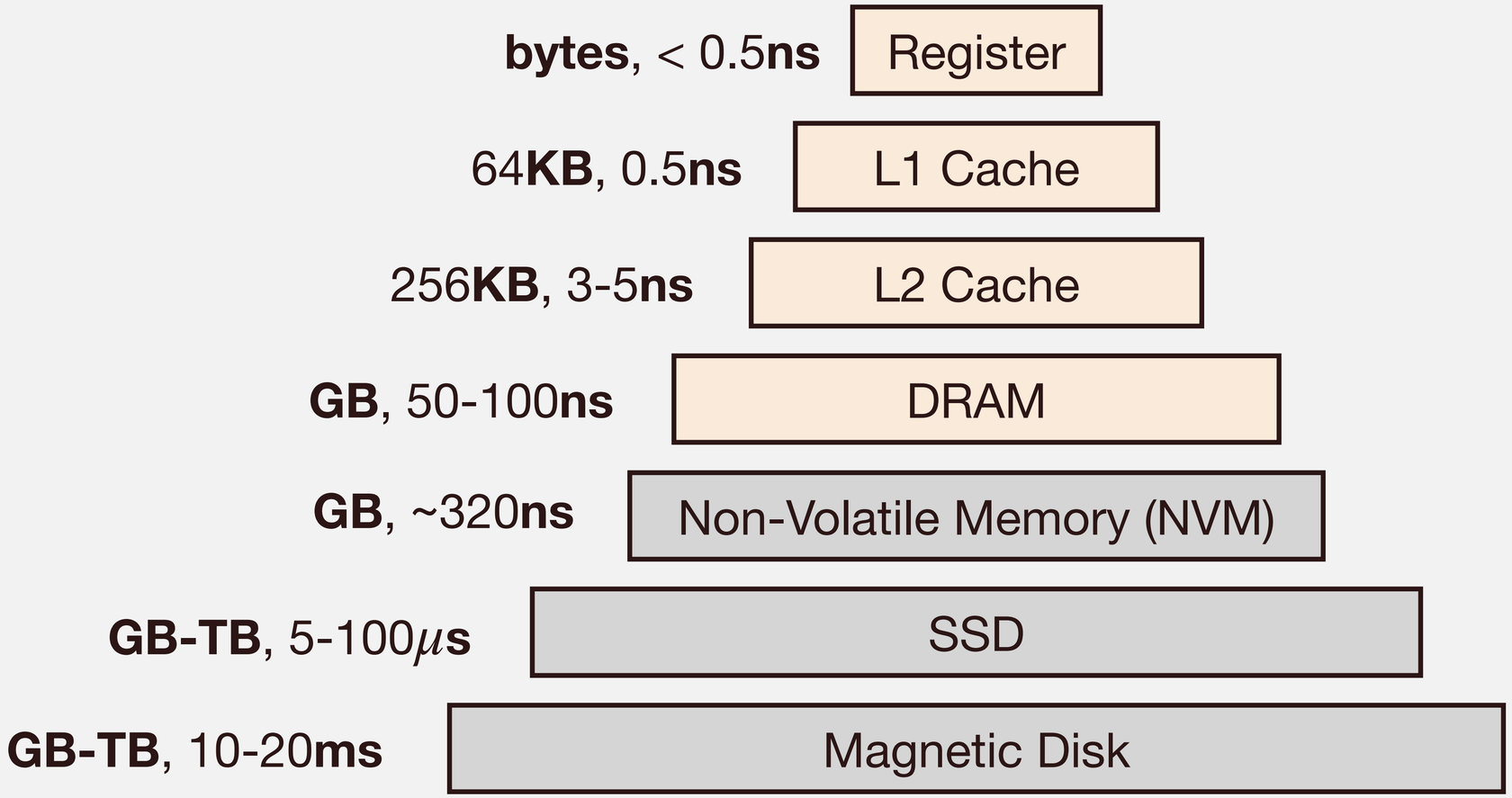
Storage Hierarchy



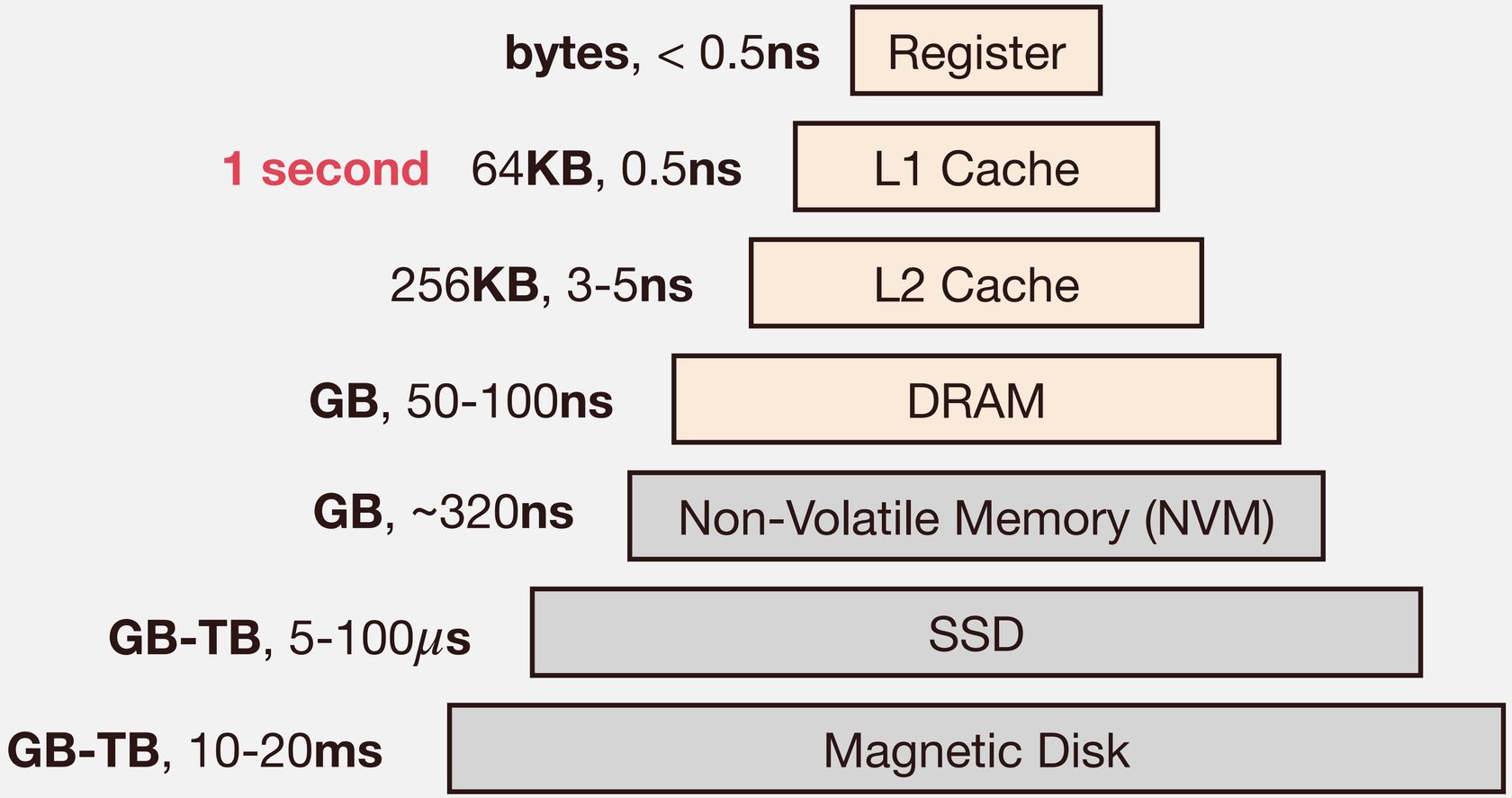
Storage Hierarchy



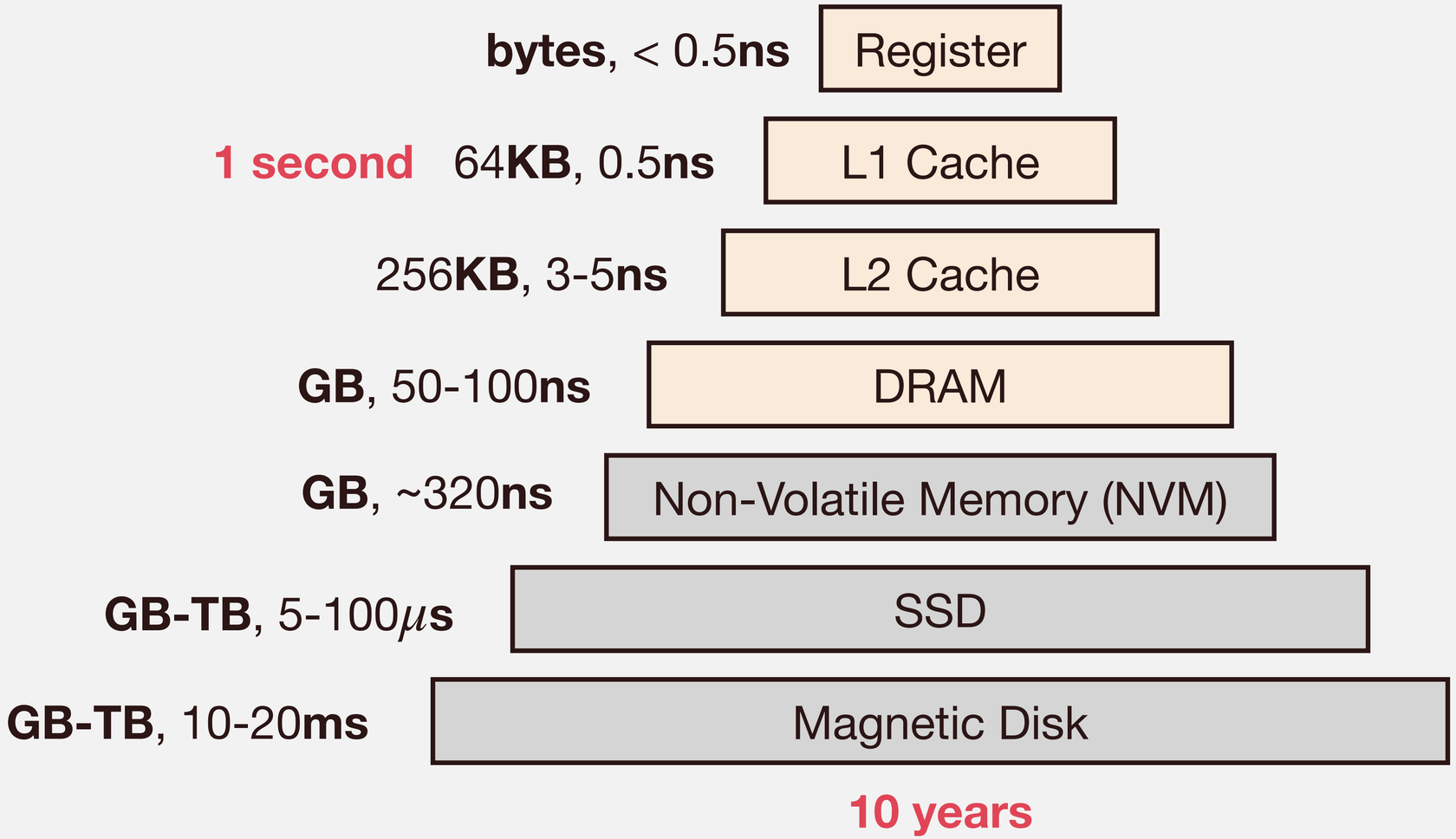
Storage Hierarchy



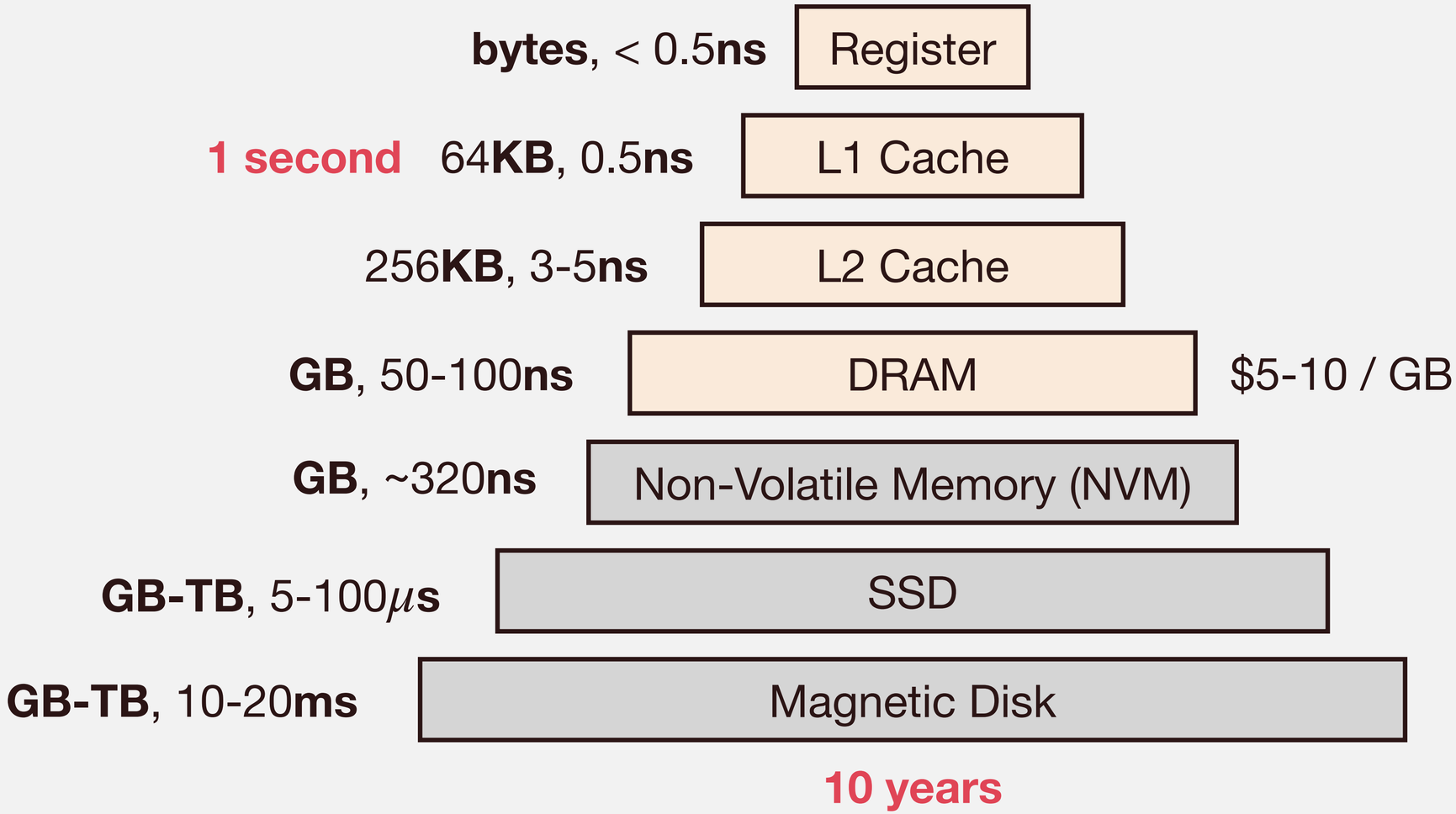
Storage Hierarchy



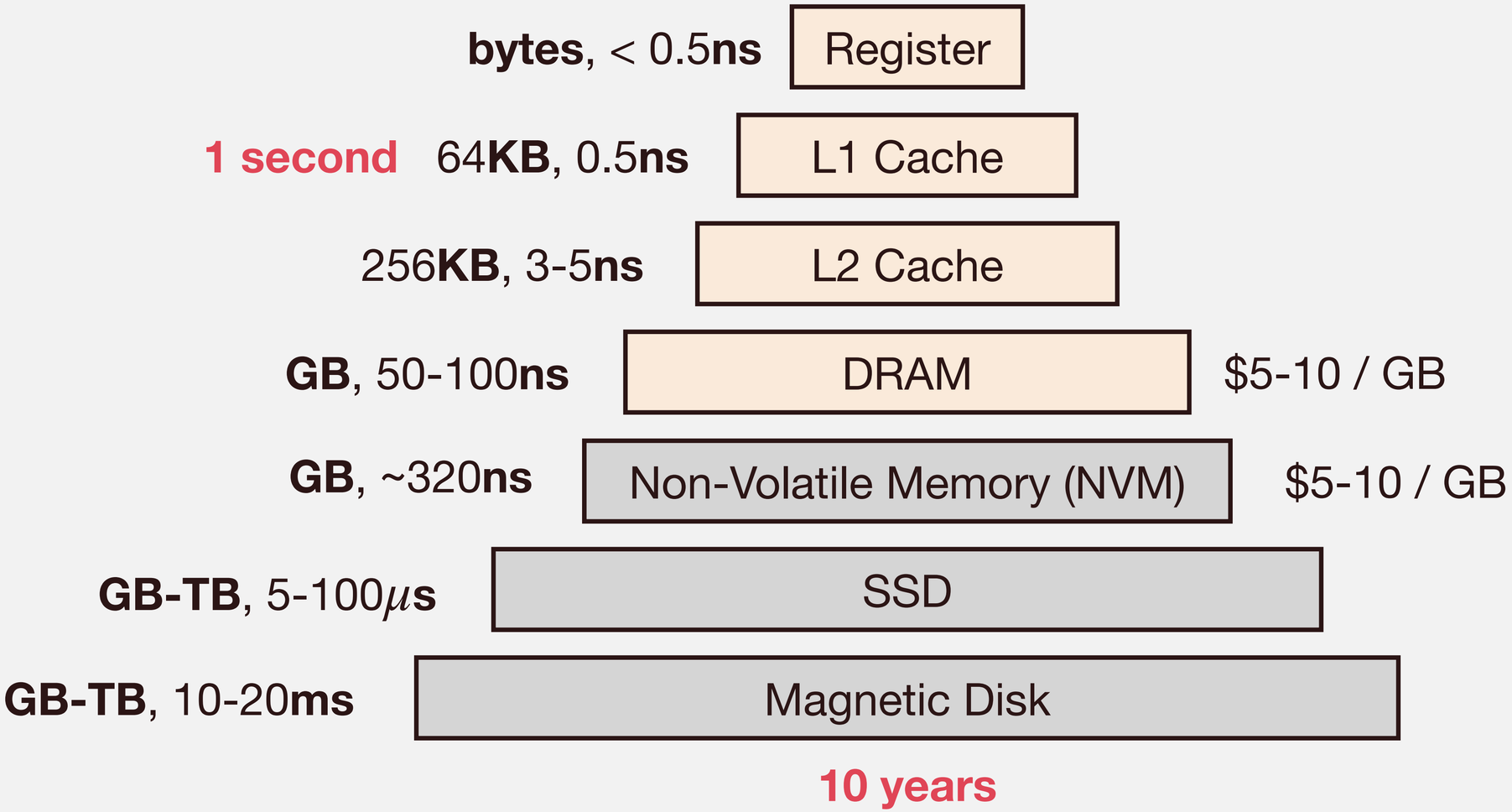
Storage Hierarchy



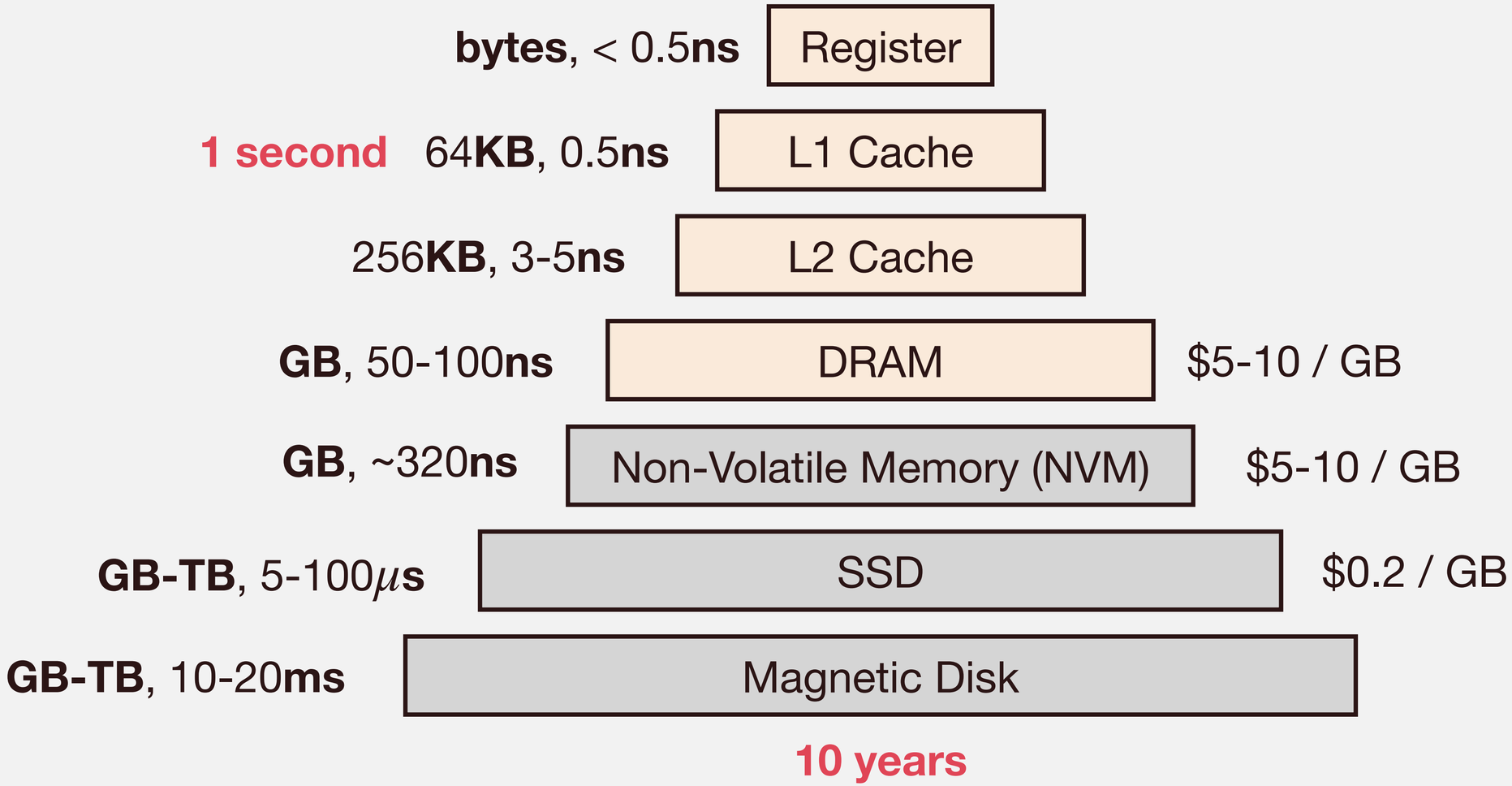
Storage Hierarchy



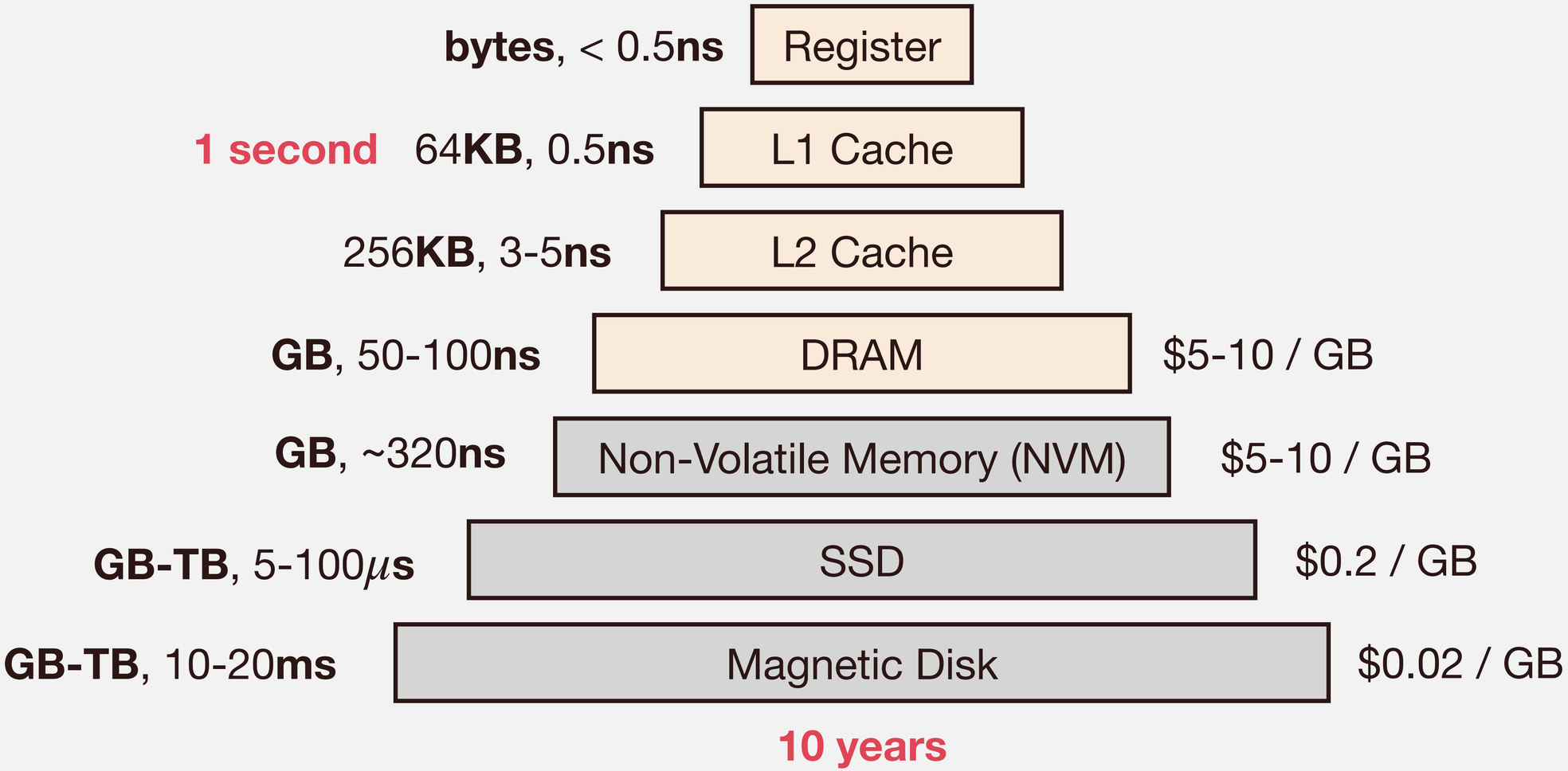
Storage Hierarchy



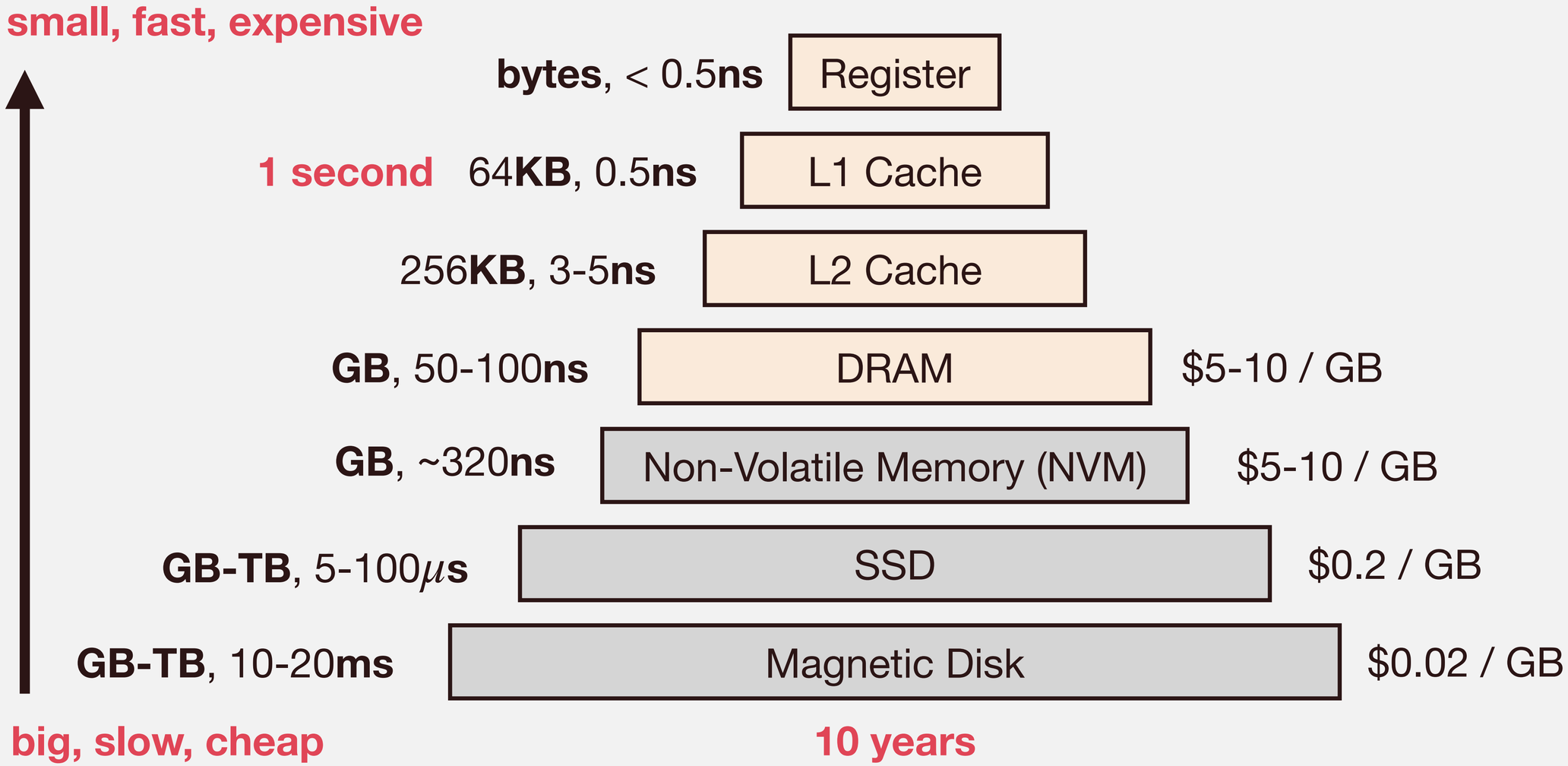
Storage Hierarchy



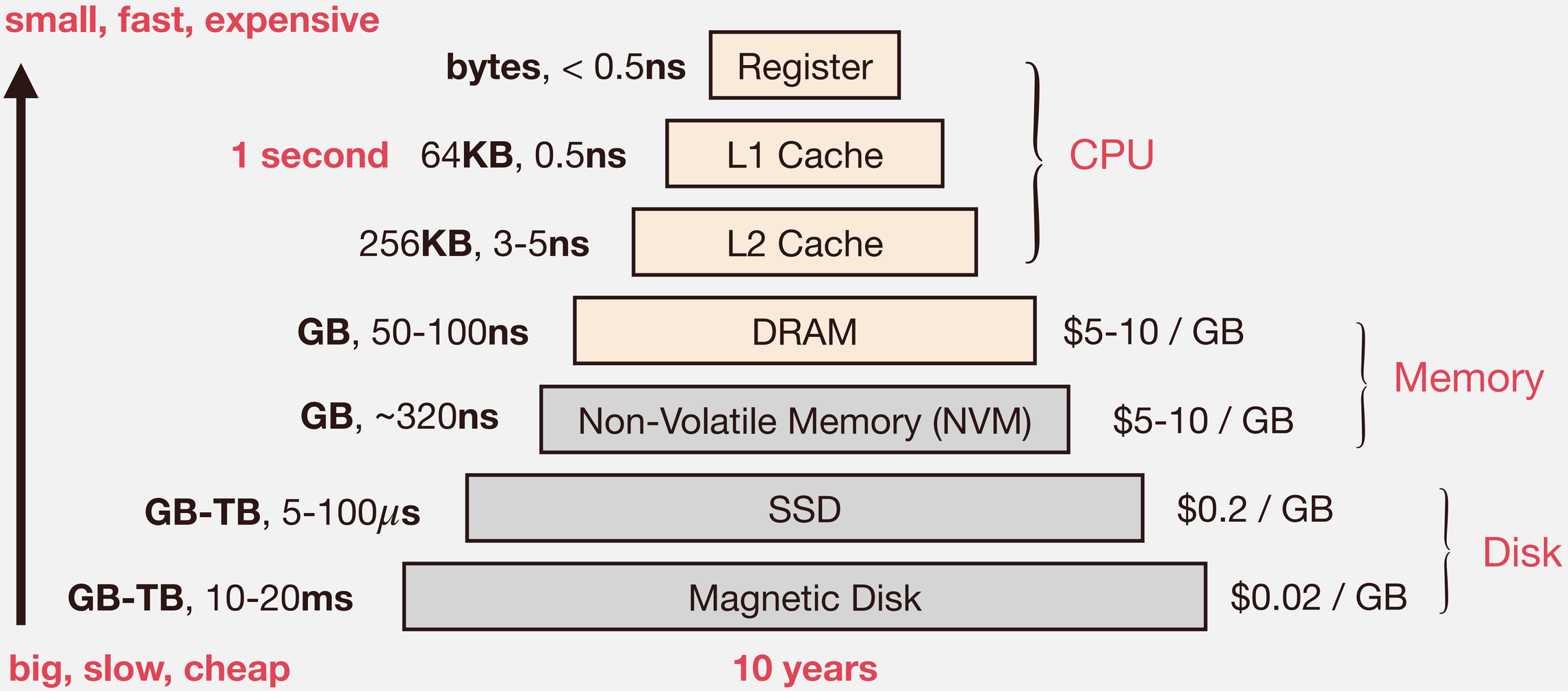
Storage Hierarchy



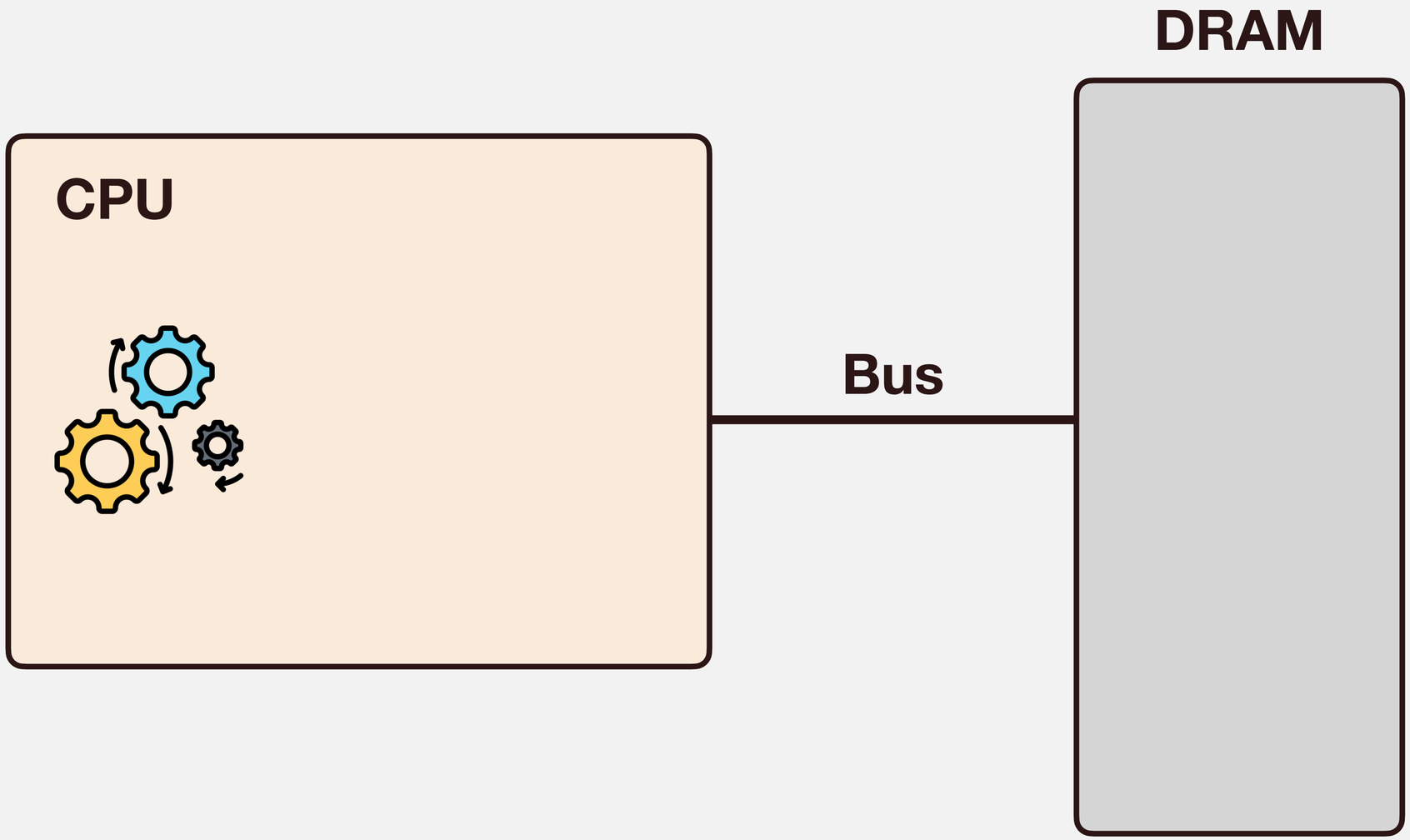
Storage Hierarchy



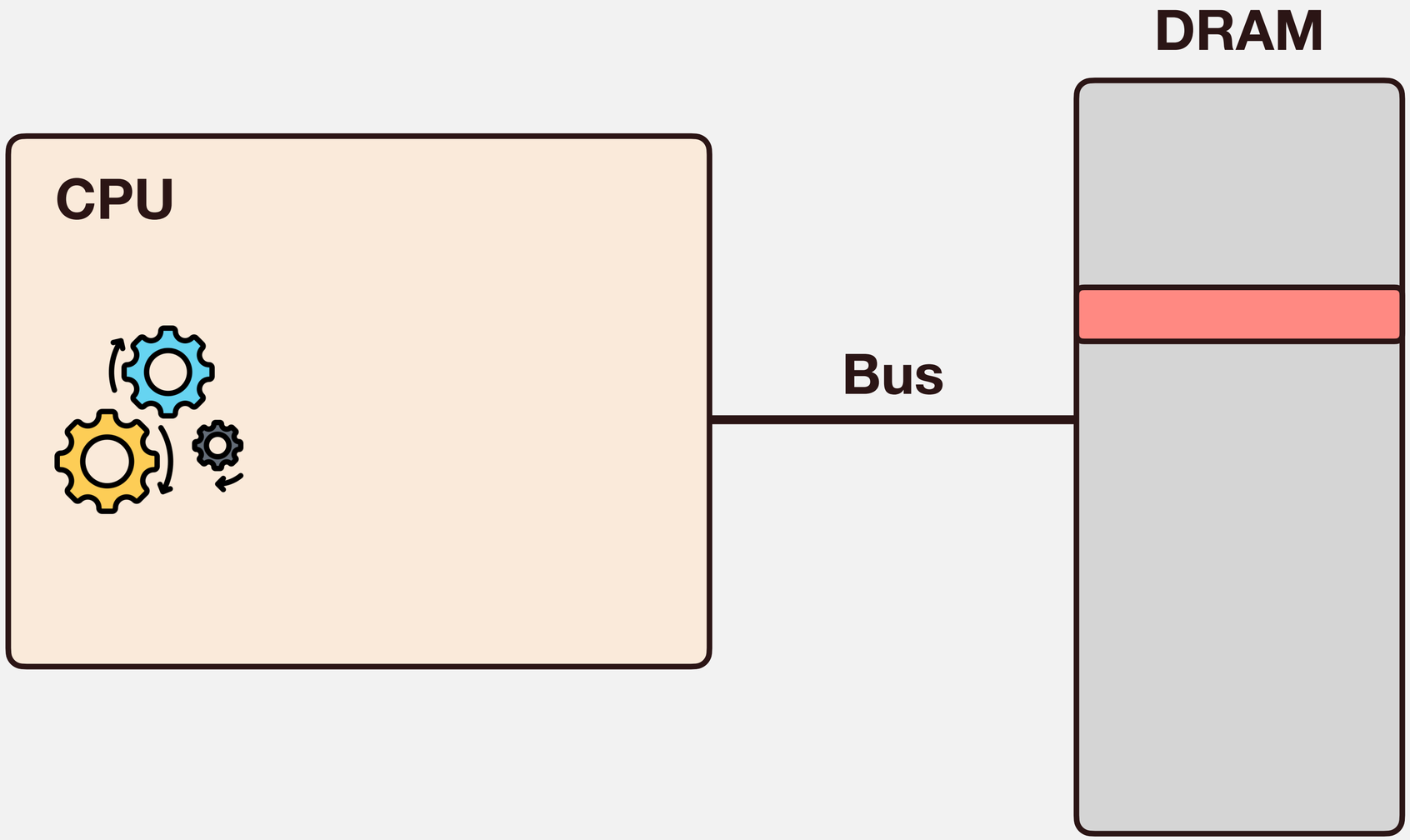
Storage Hierarchy



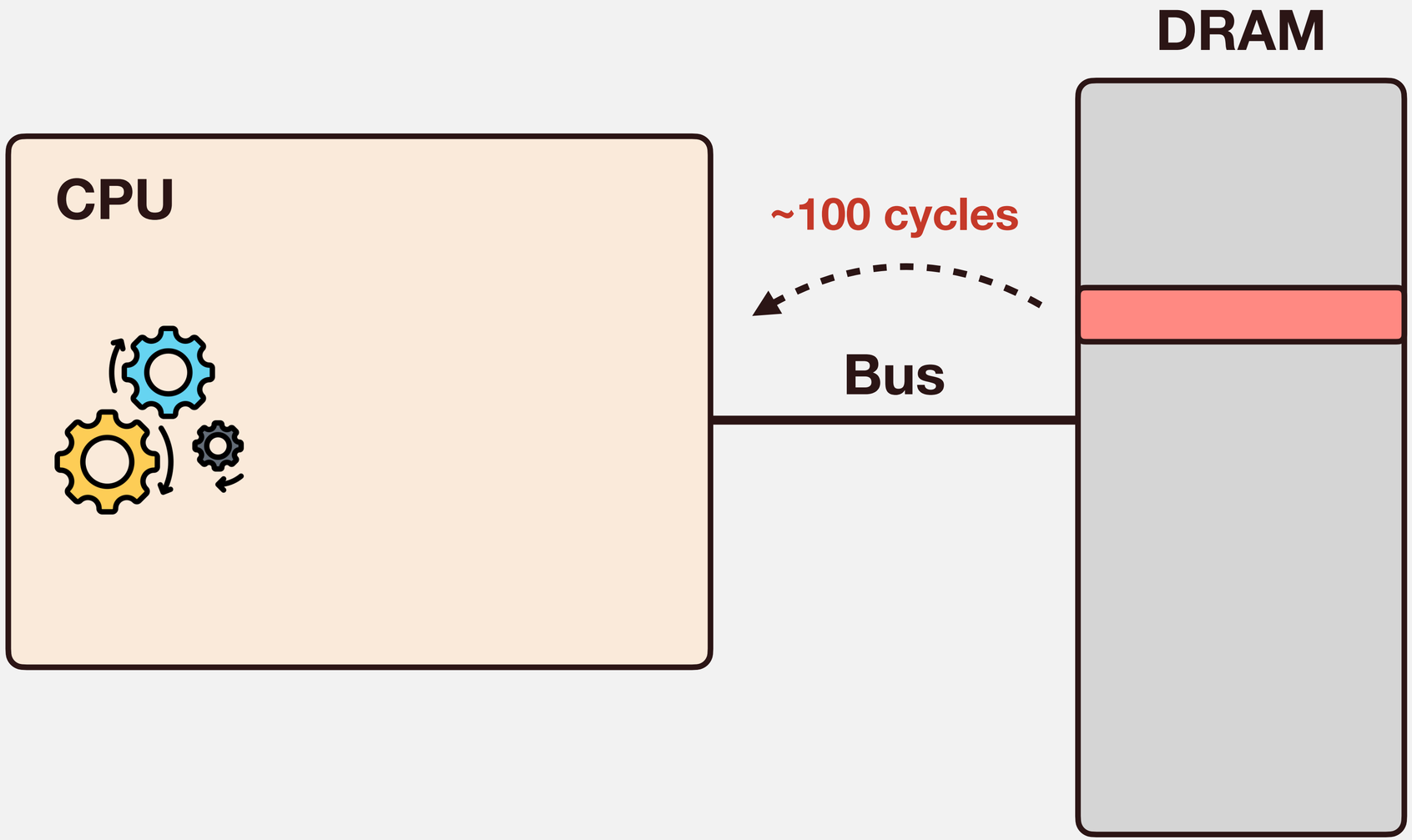
CPU Cache



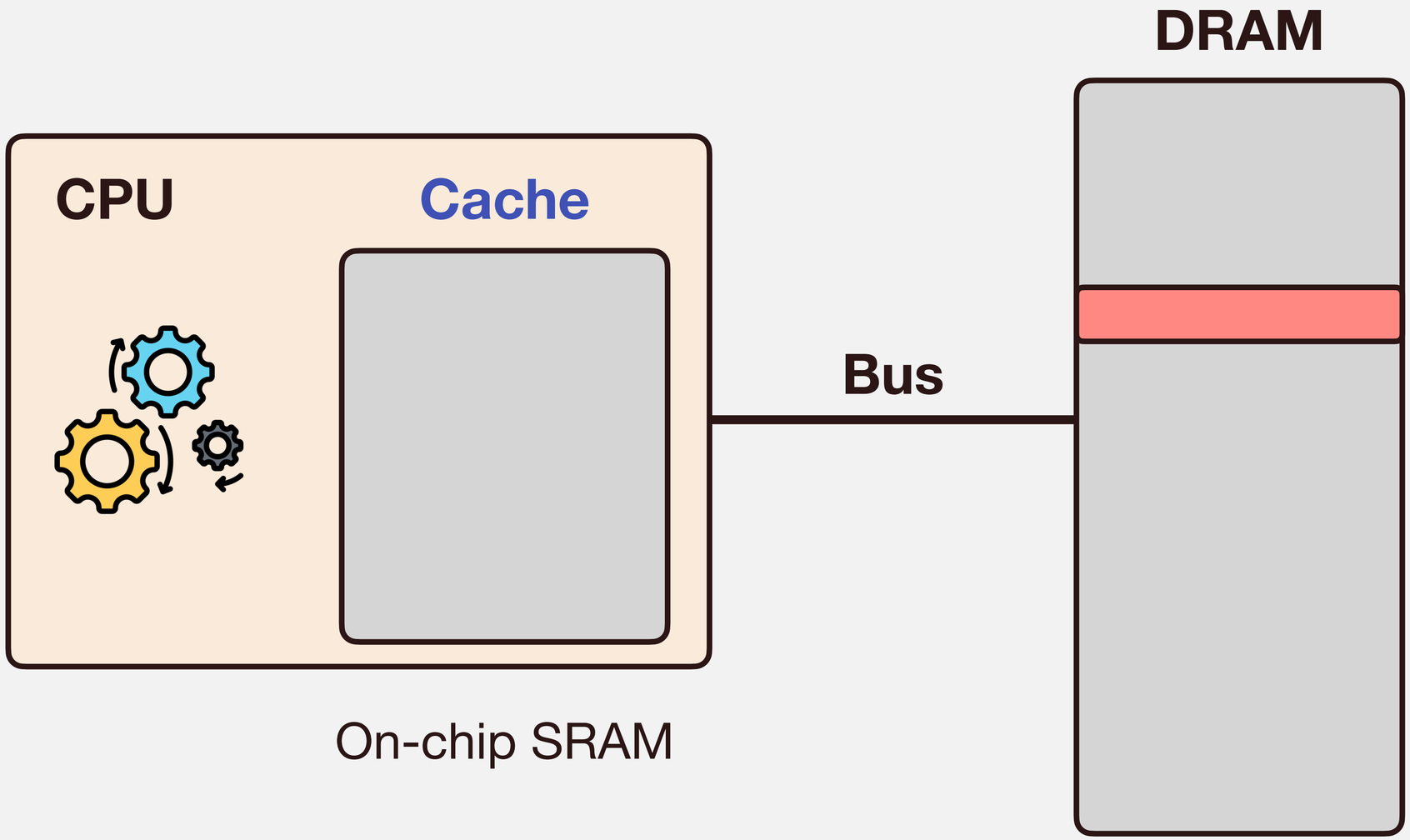
CPU Cache



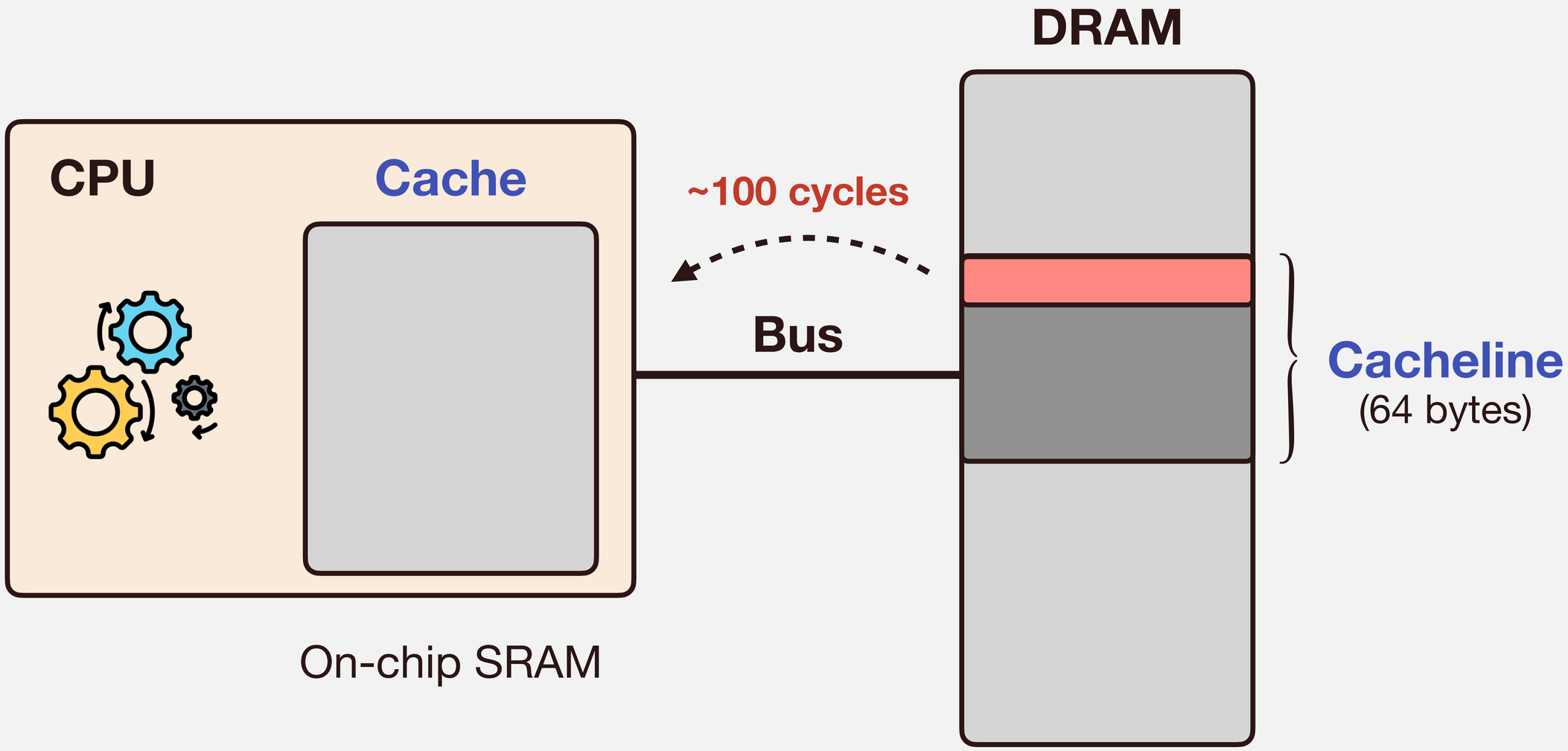
CPU Cache



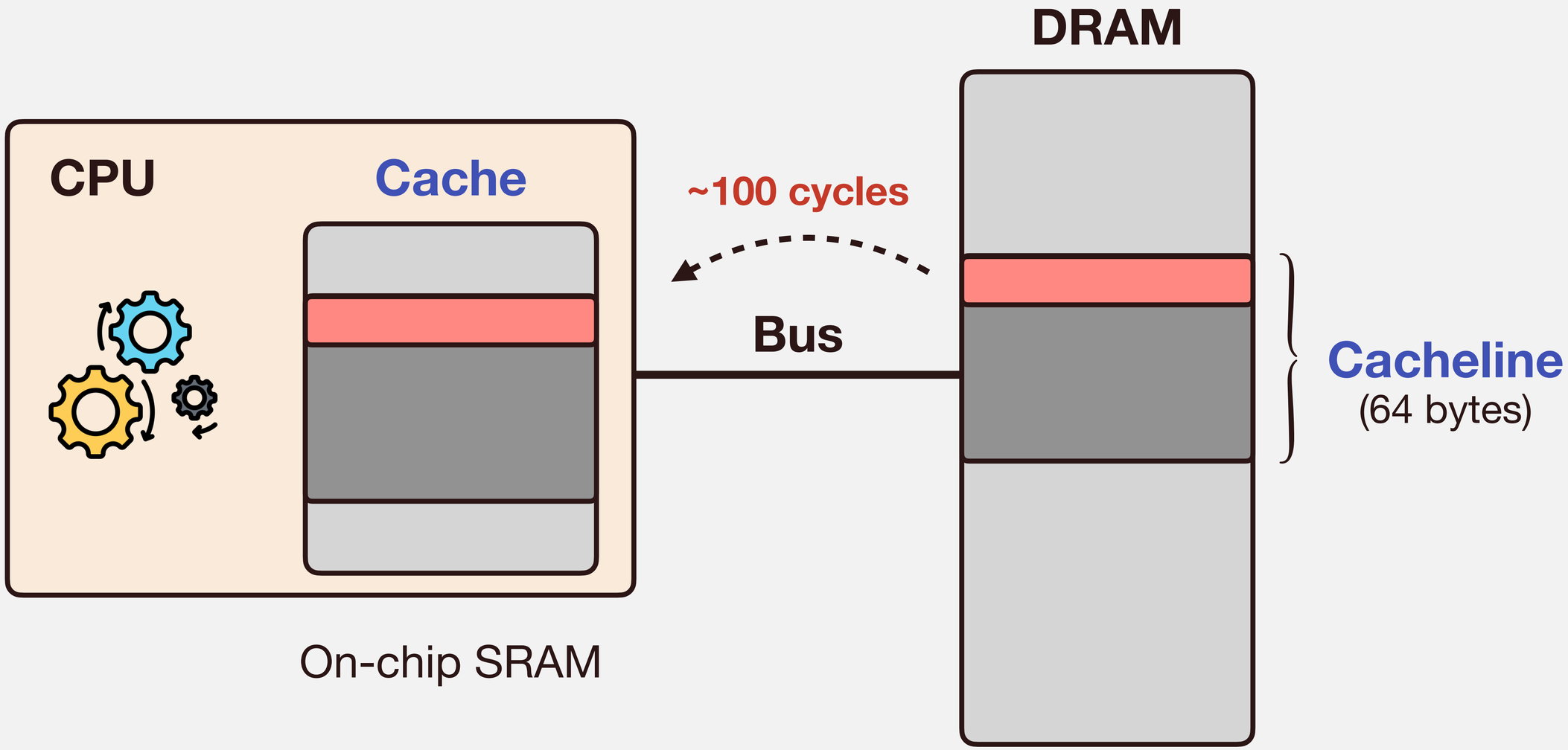
CPU Cache



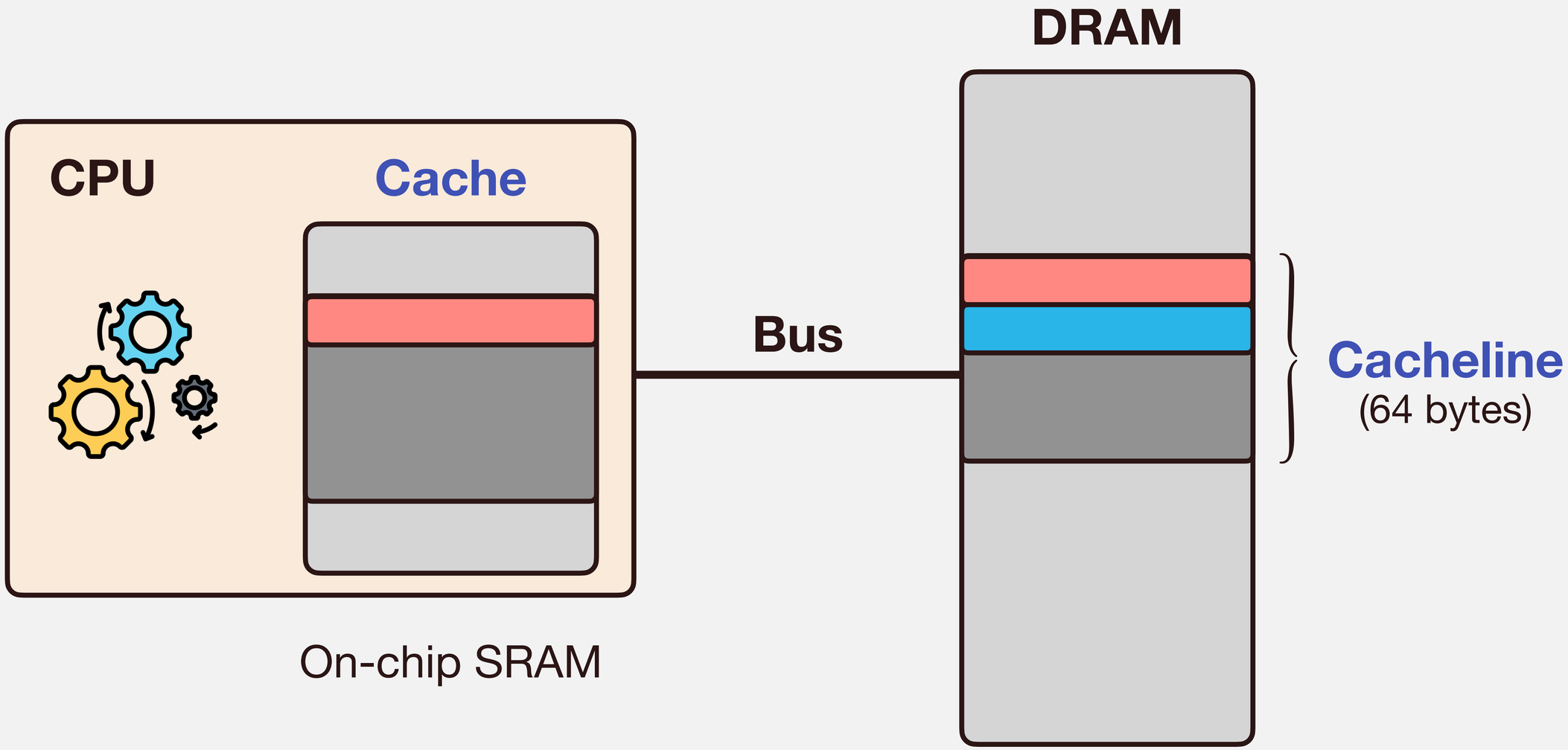
CPU Cache


DRAM
CPU
Cache
~100 cycles
Bus
Cacheline
(64 bytes)
On-chip SRAM

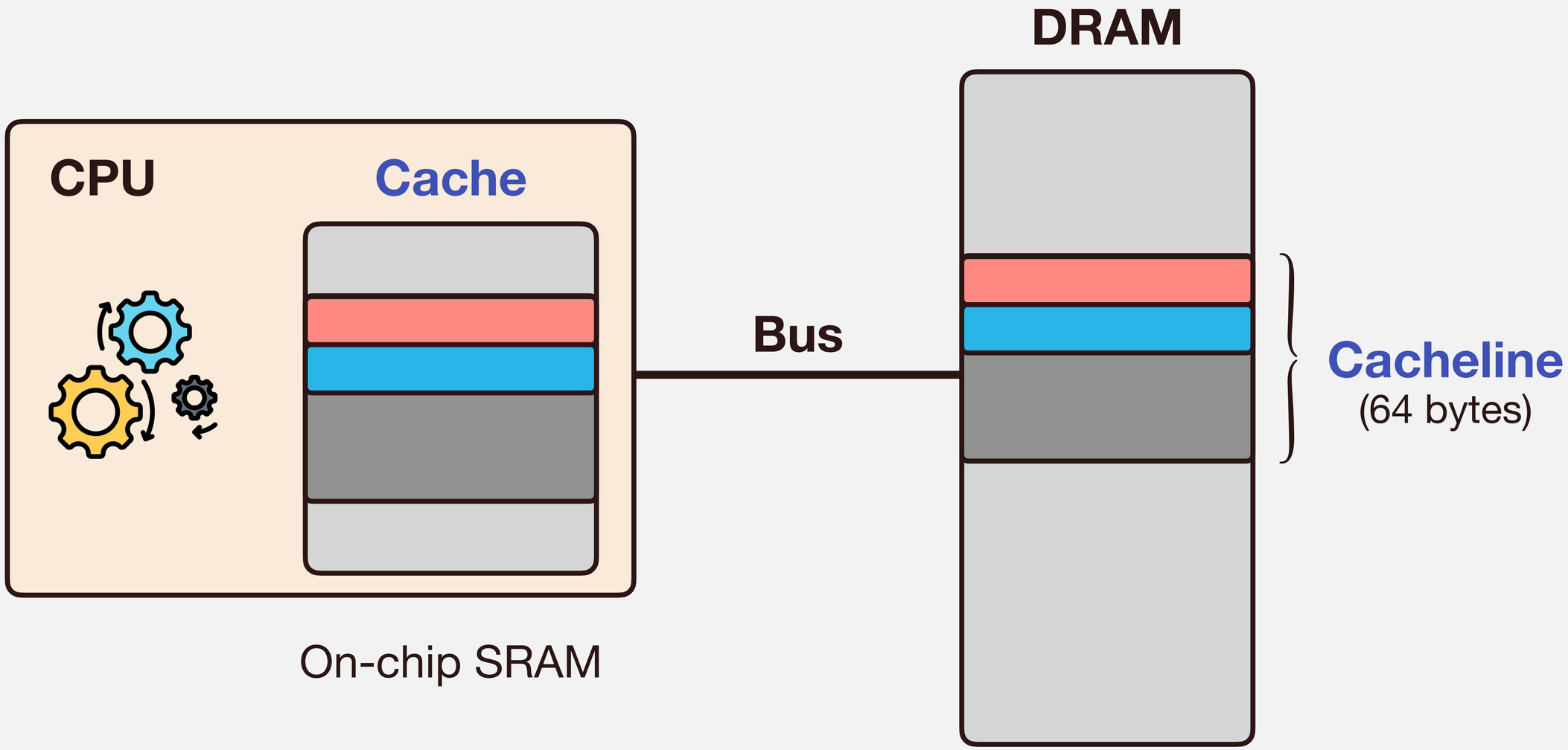
CPU Cache


DRAM
CPU
Cache
~100 cycles
Bus
Cacheline
(64 bytes)
On-chip SRAM

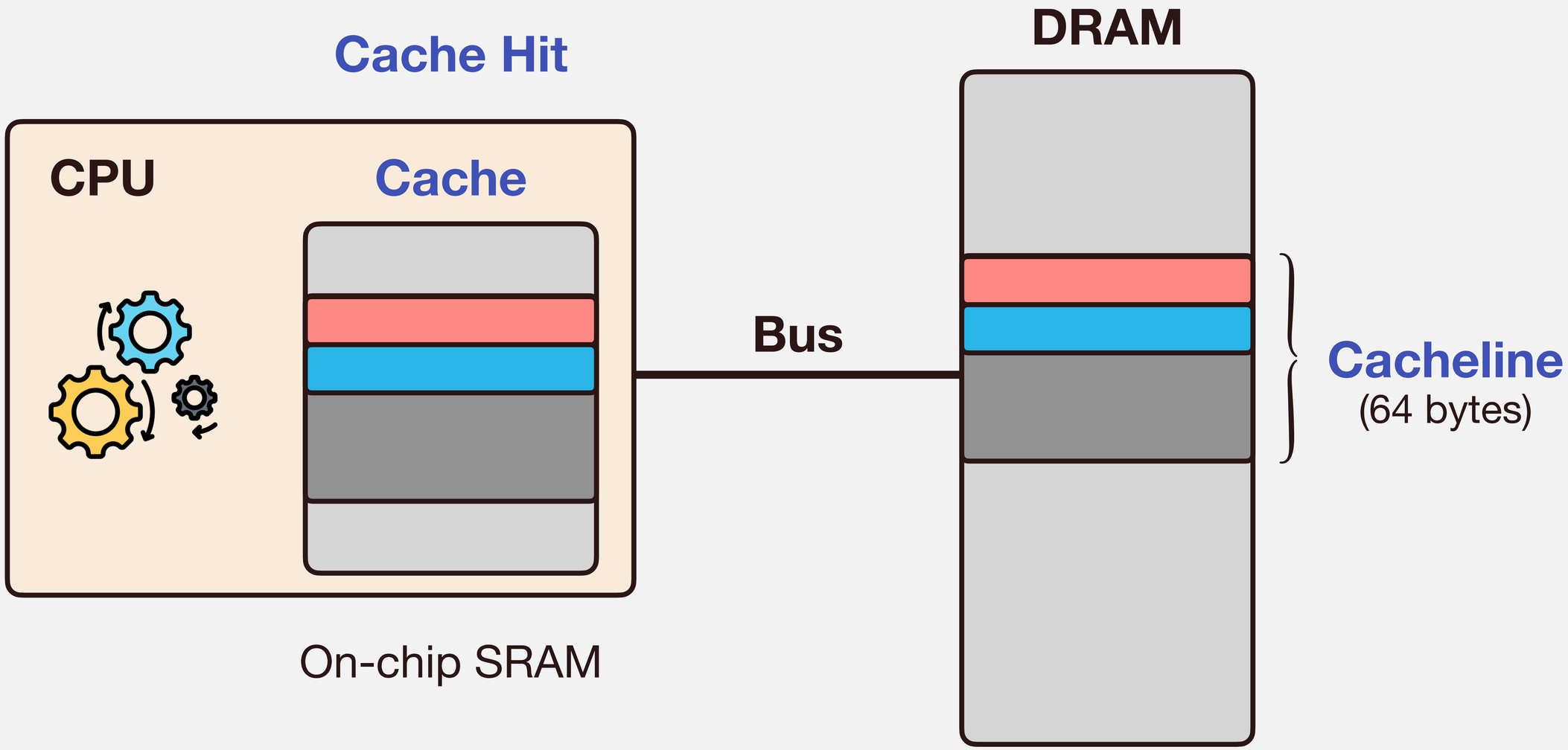
CPU Cache



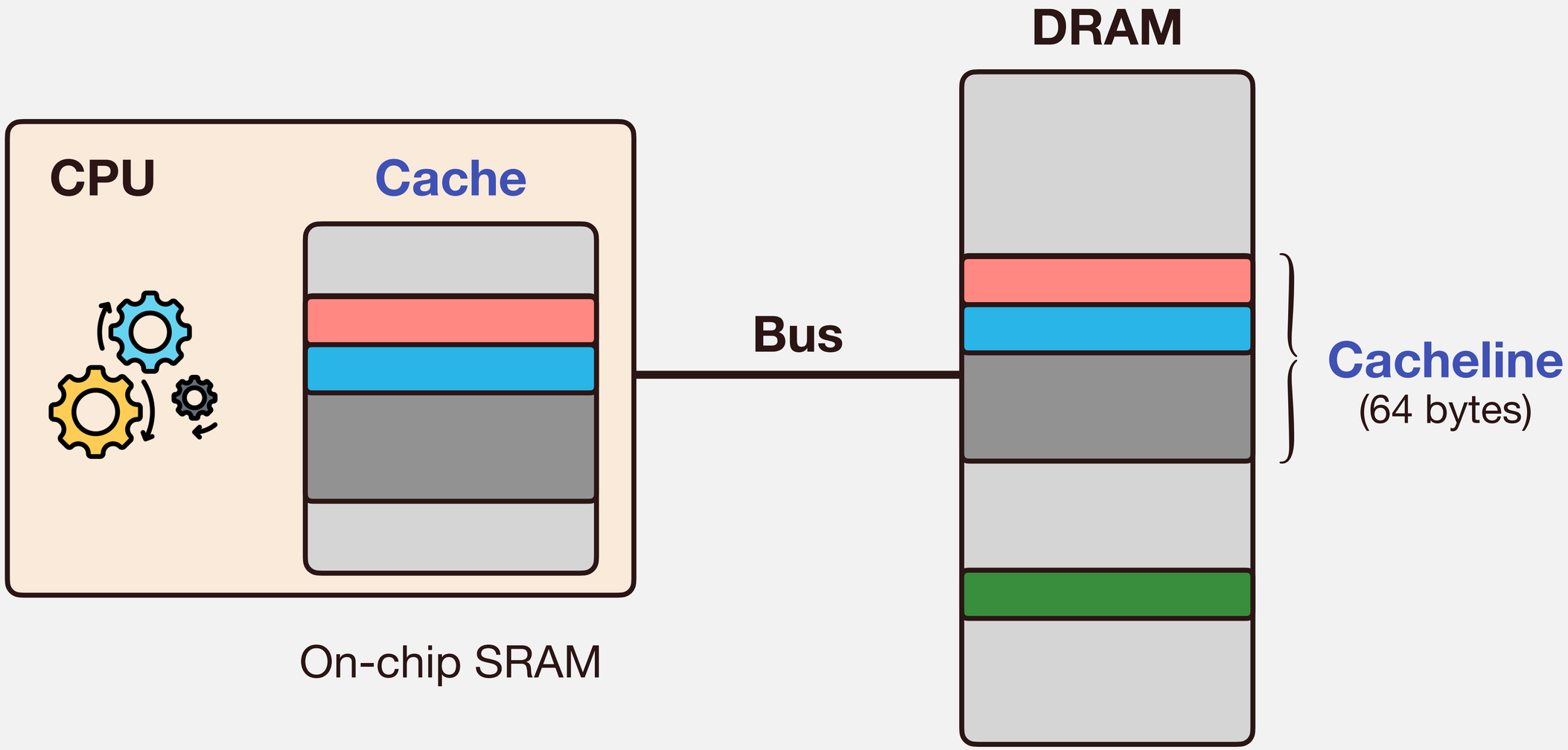
CPU Cache



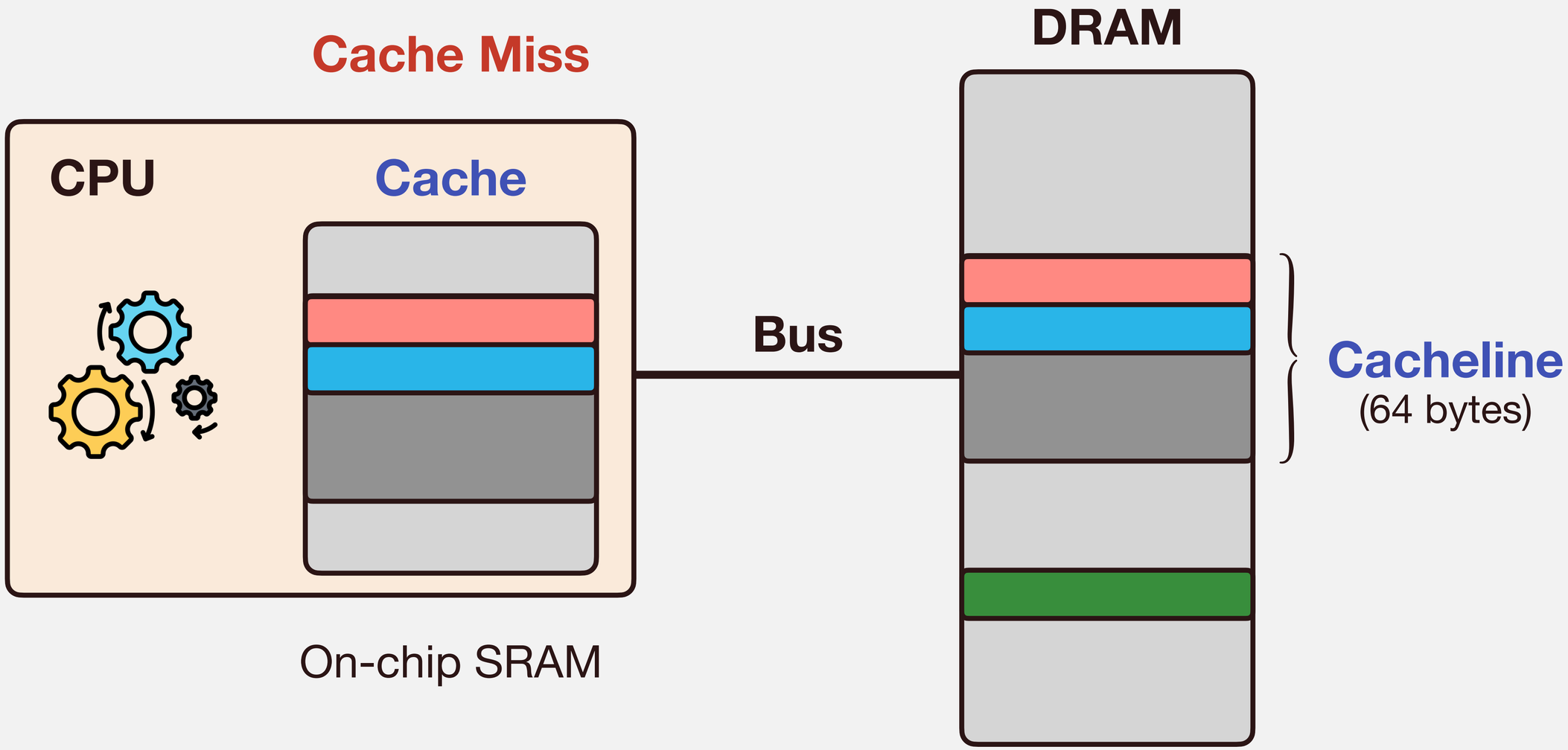
CPU Cache



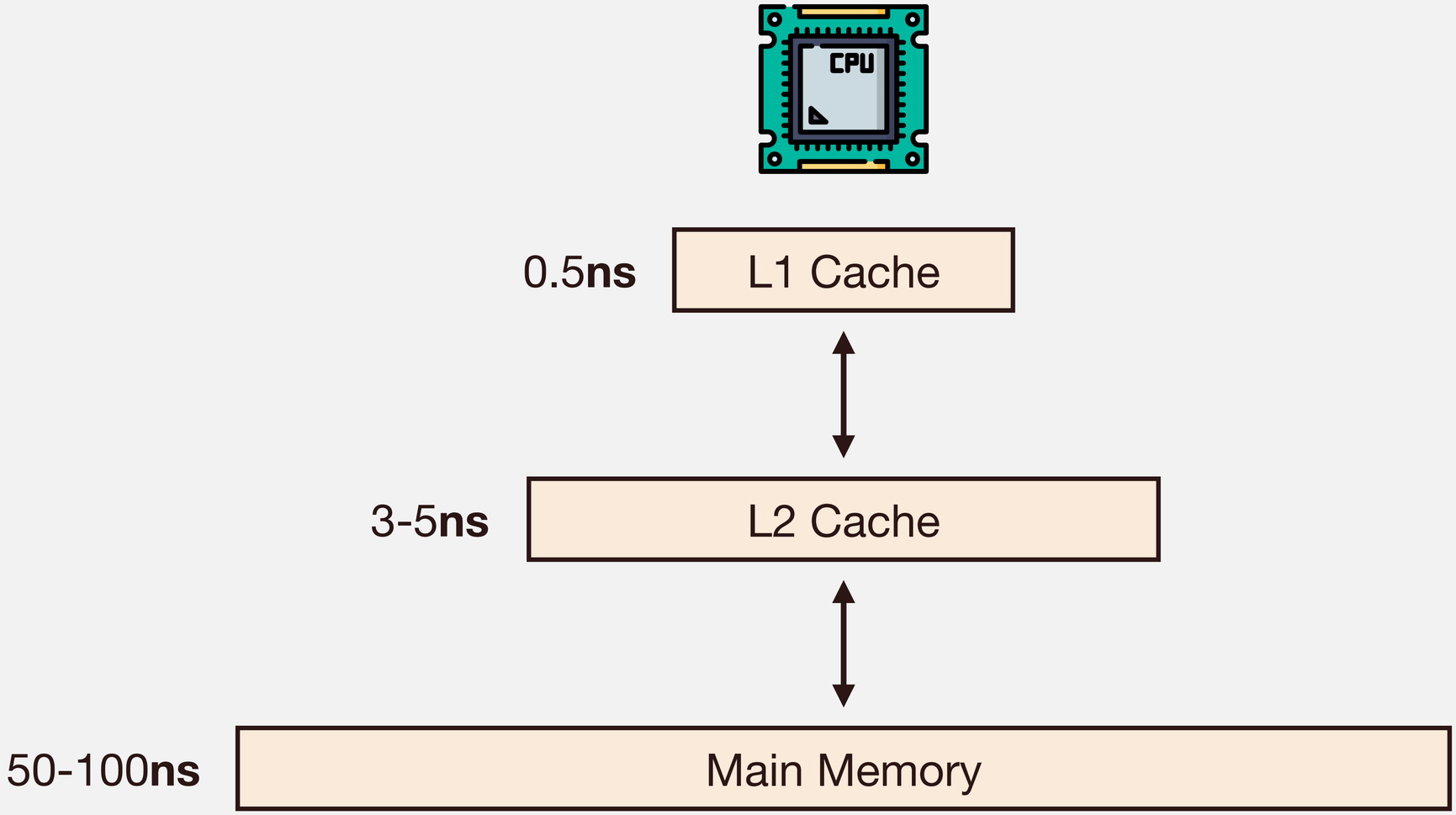
CPU Cache



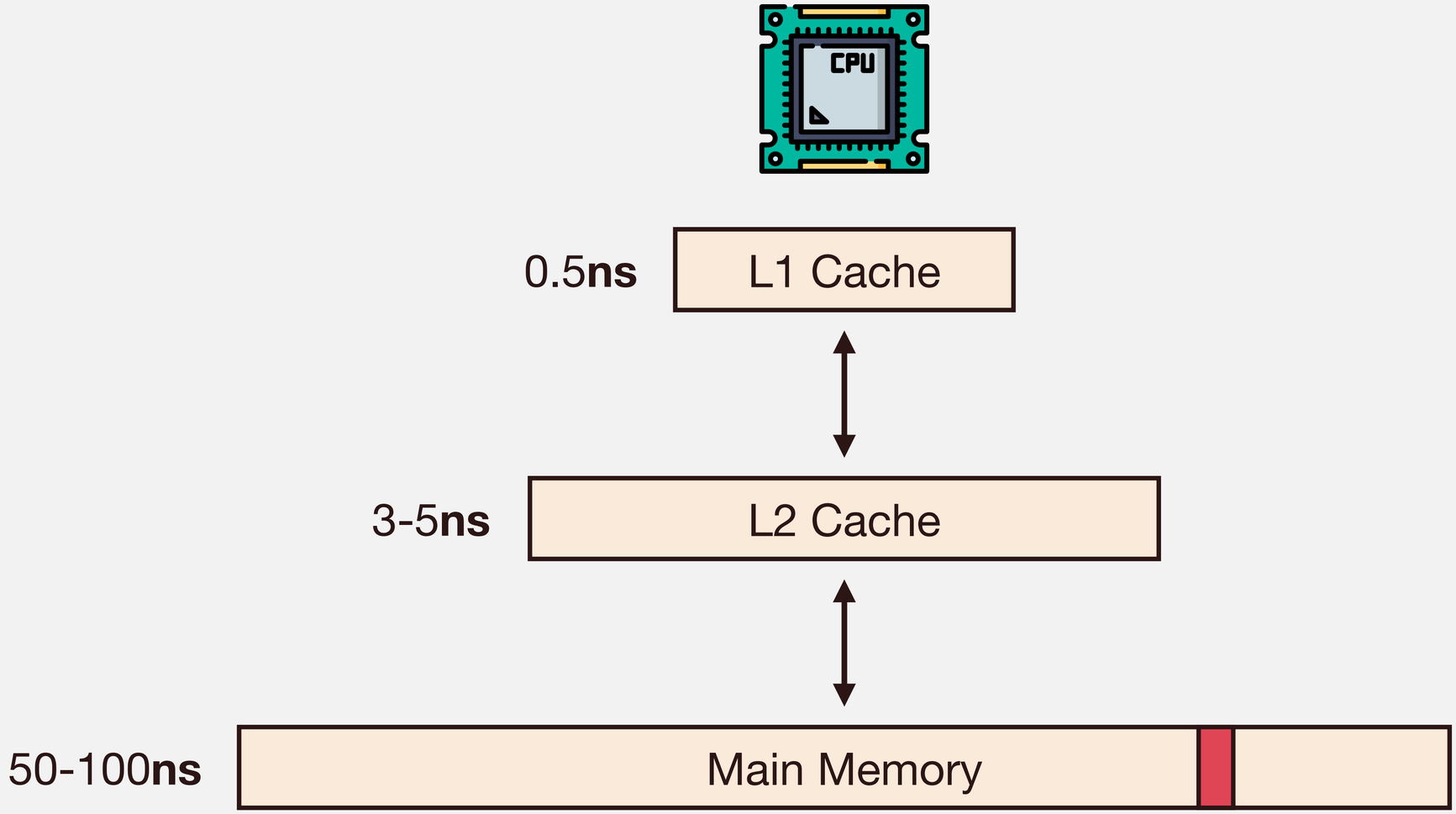
CPU Cache


Cache Miss
DRAM
CPU
Cache
Bus
Cacheline
(64 bytes)
On-chip SRAM

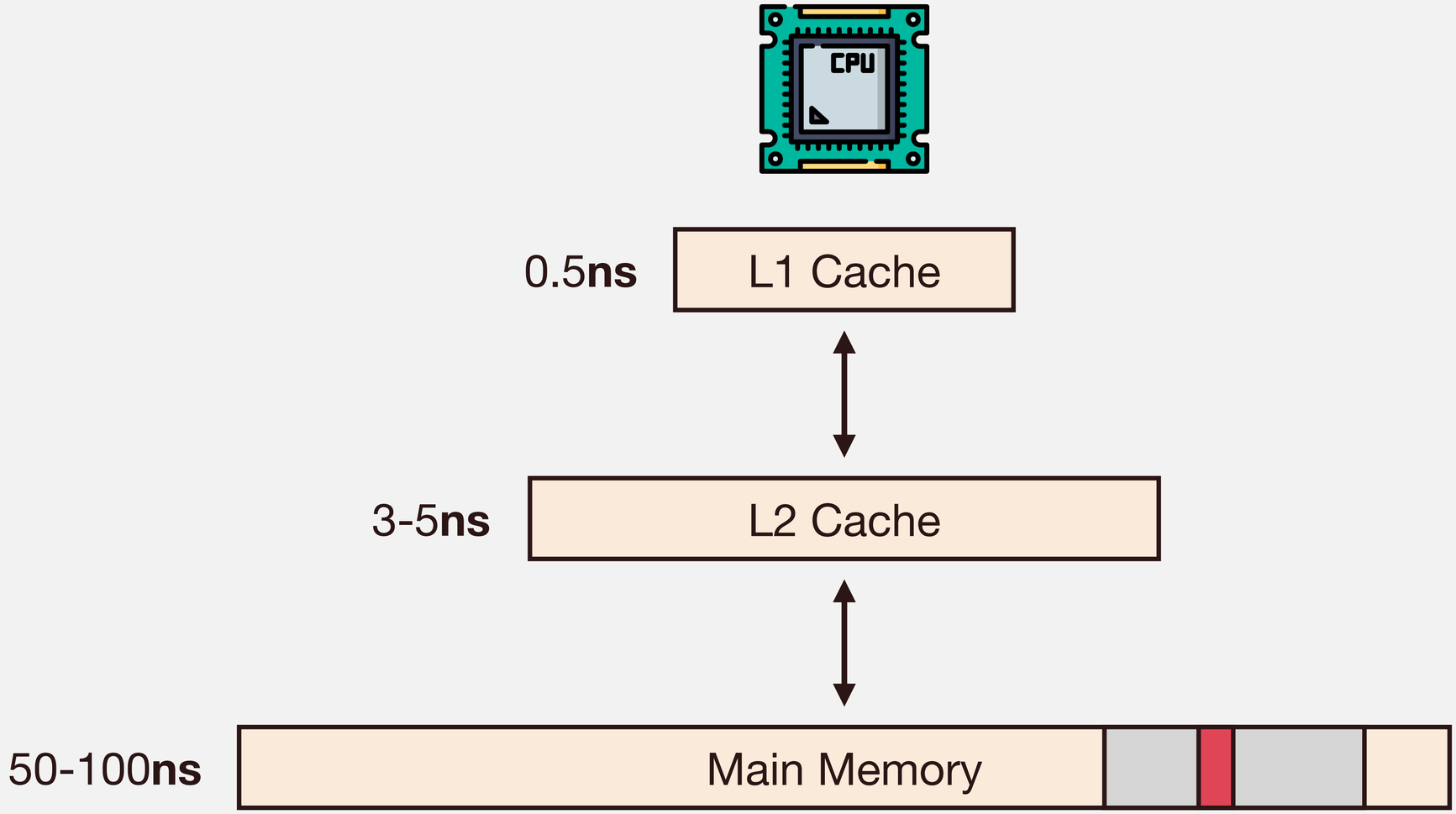
Cacheline



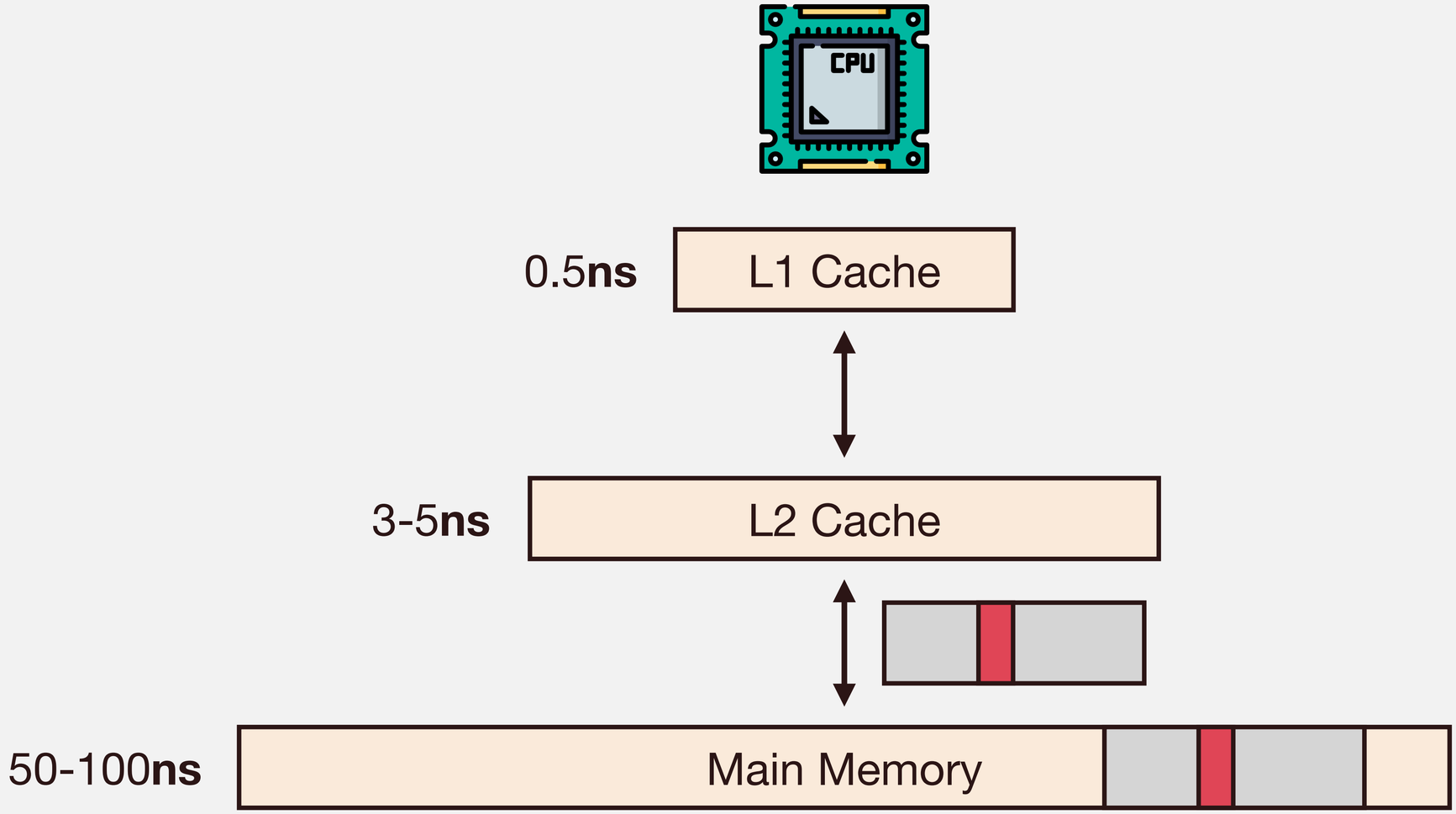
Cacheline



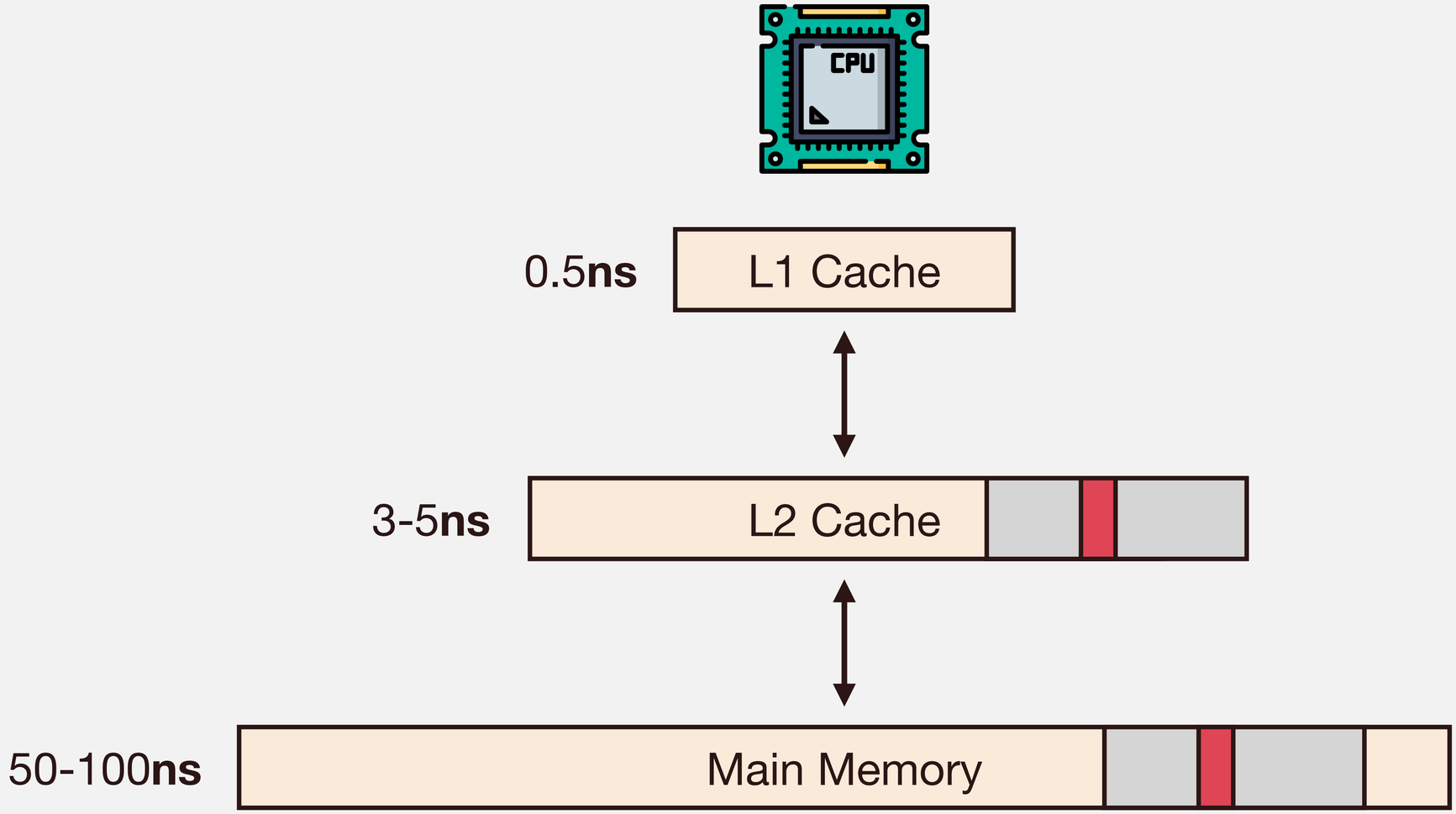
Cacheline



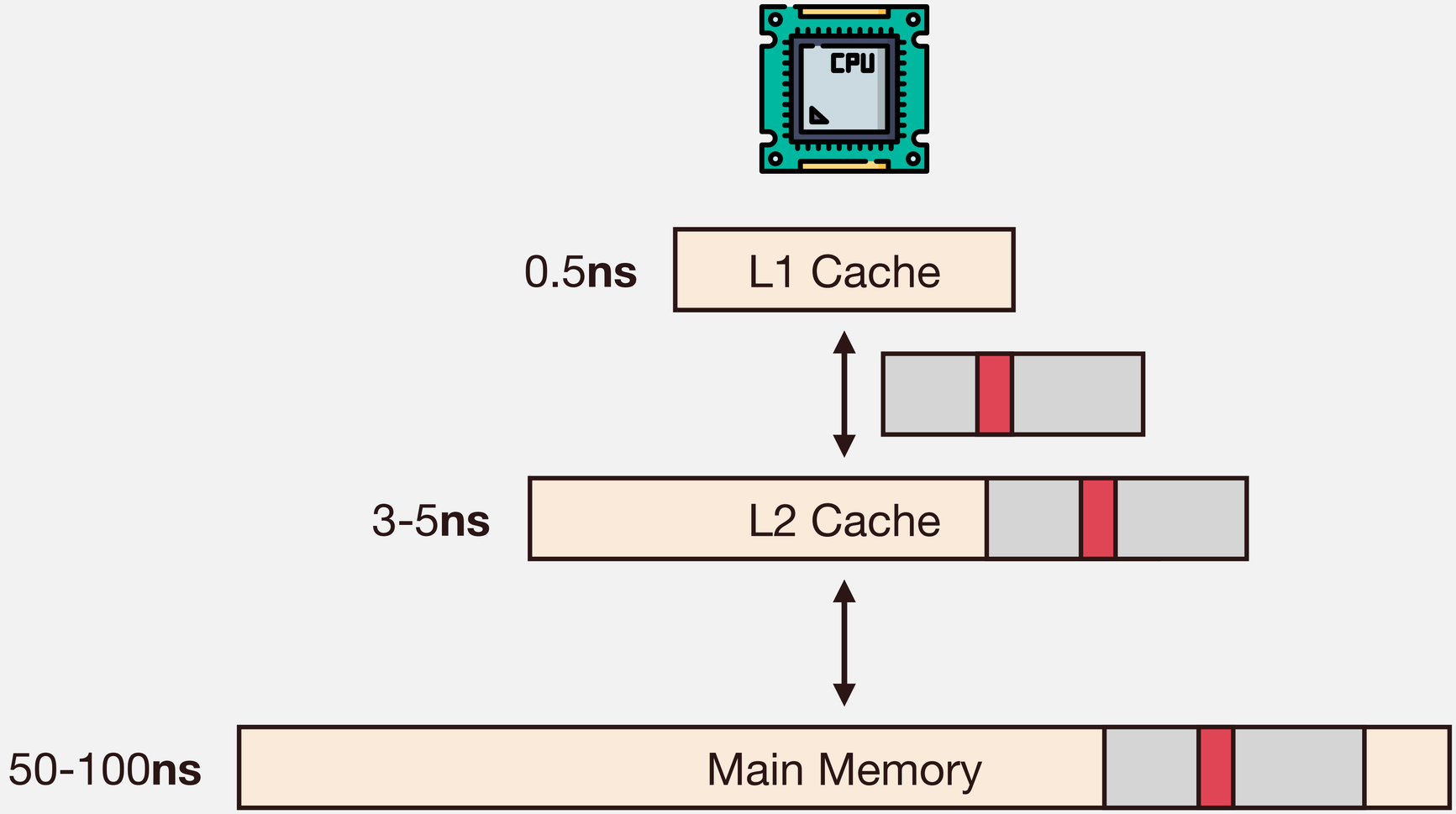
Cacheline



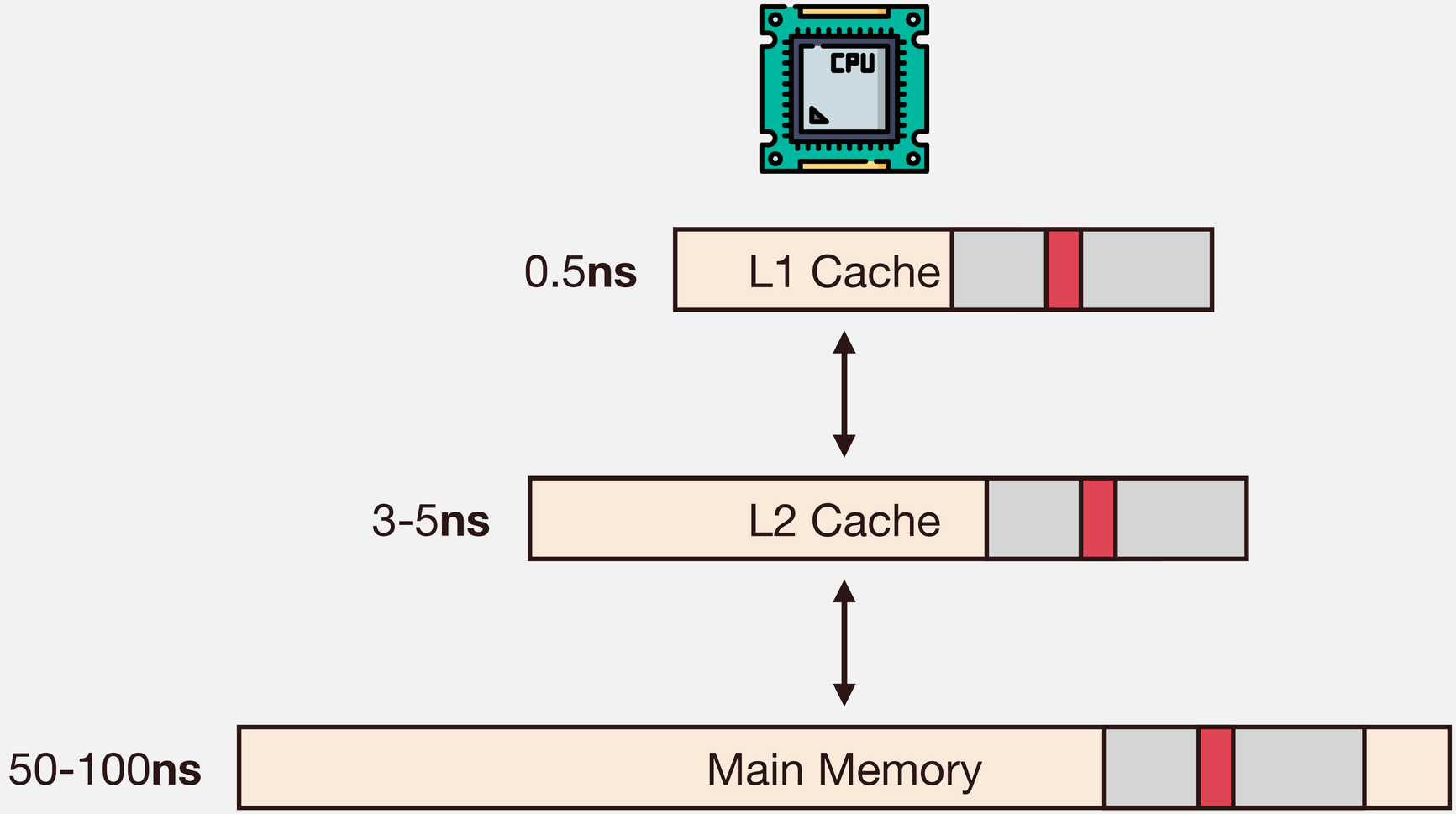
Cacheline



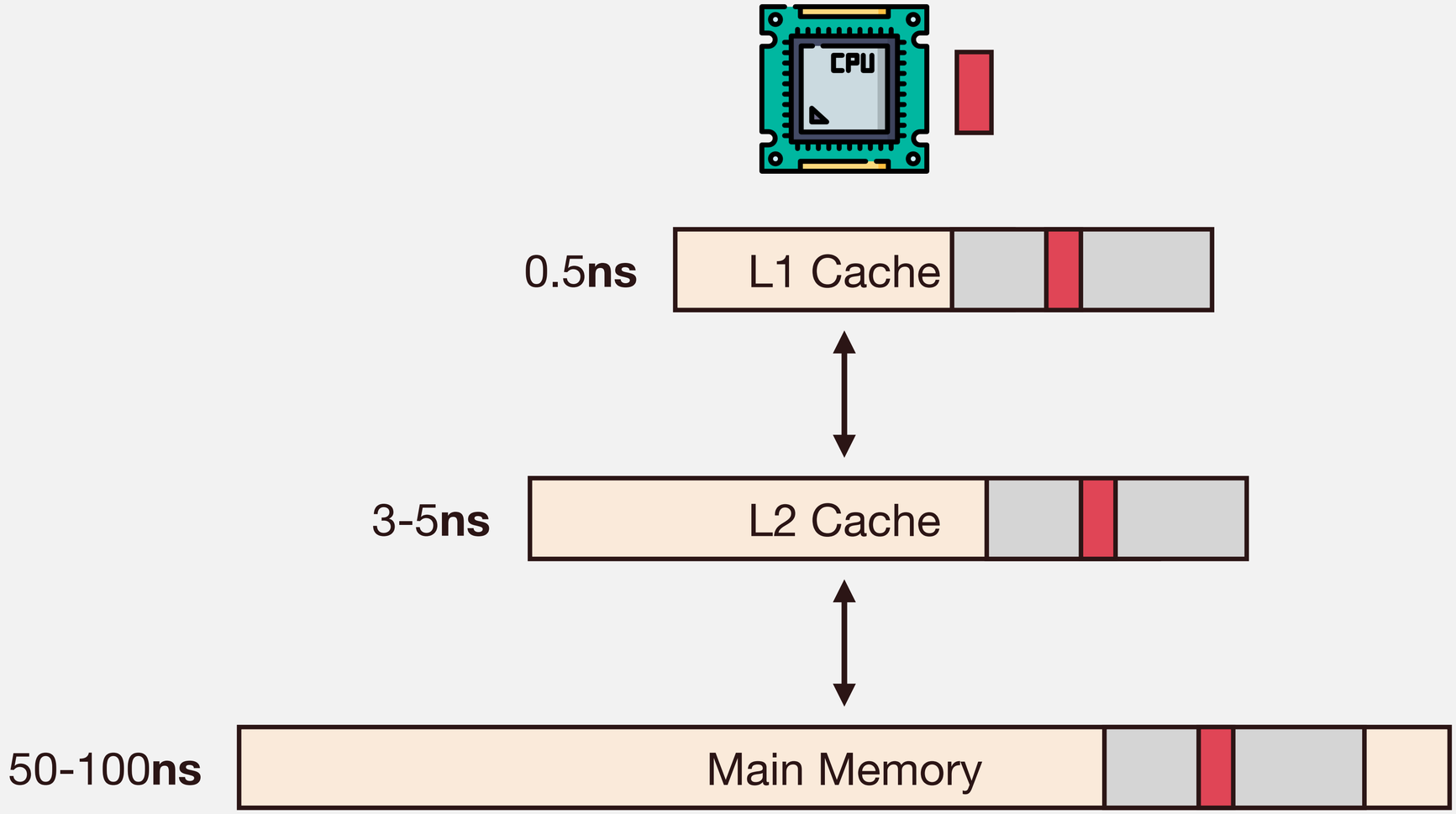
Cacheline



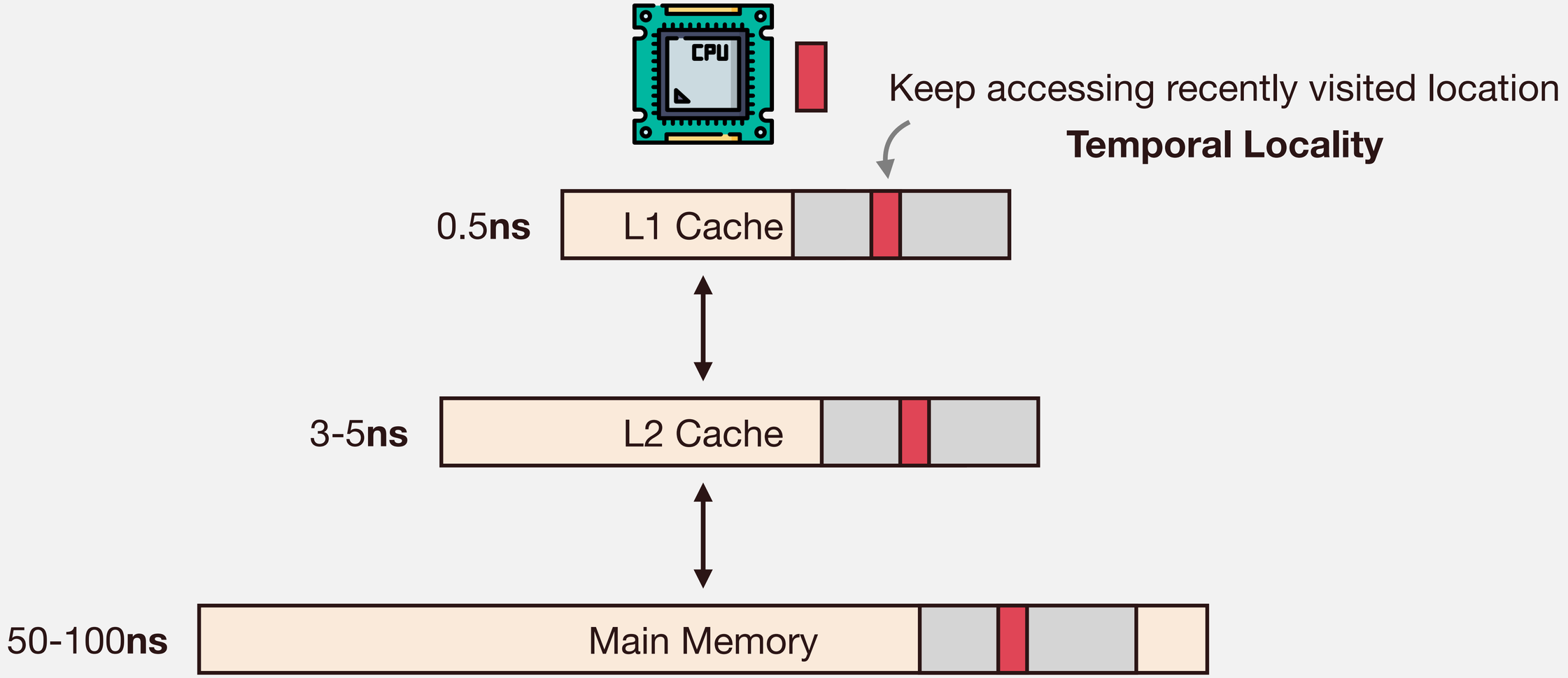
Cacheline



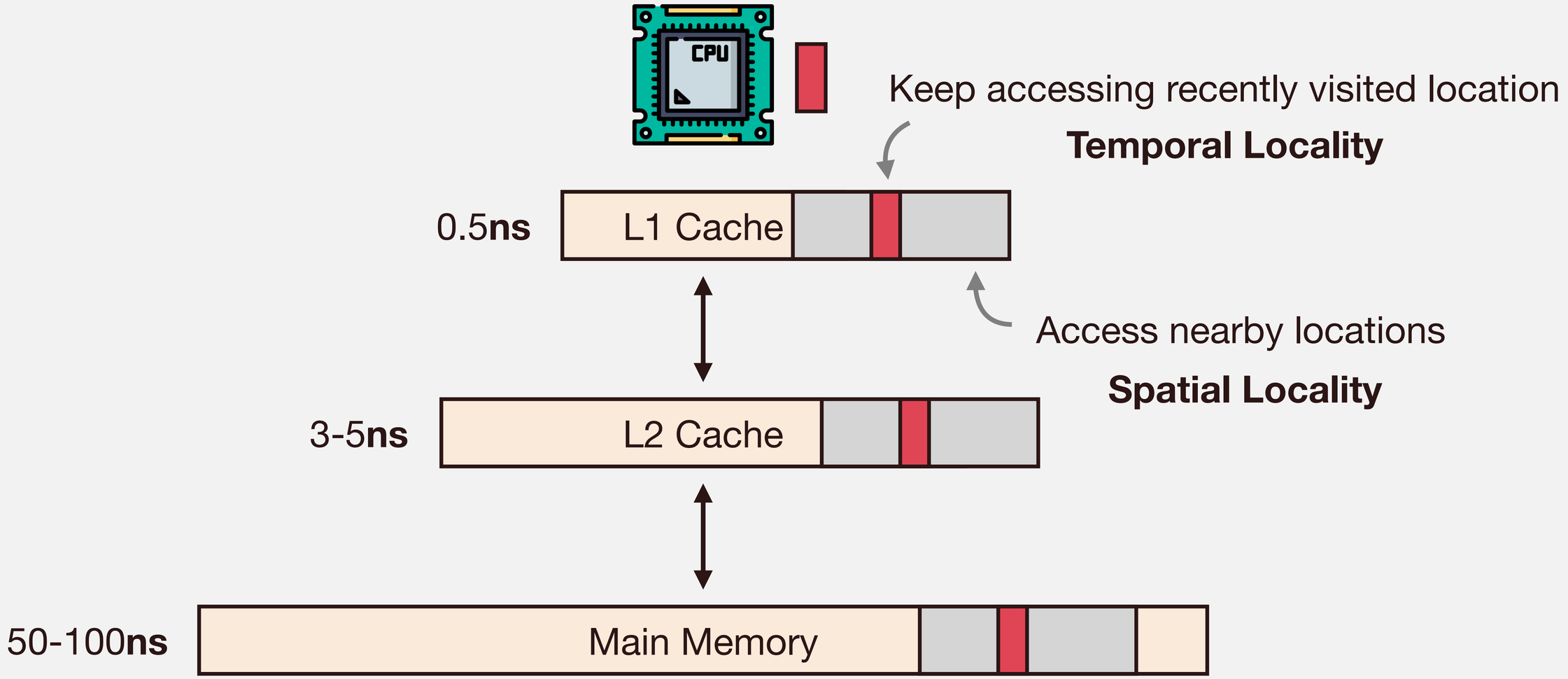
Cacheline



Cacheline


CPU
Keep accessing recently visited location
Temporal Locality
0.5ns
L1 Cache
3-5ns
L2 Cache
50-100ns
Main Memory

Cacheline


CPU
Keep accessing recently visited location
Temporal Locality
0.5ns
L1 Cache
Access nearby locations
Spatial Locality
3-5ns
L2 Cache
50-100ns
Main Memory

Low-Level Optimization Techniques

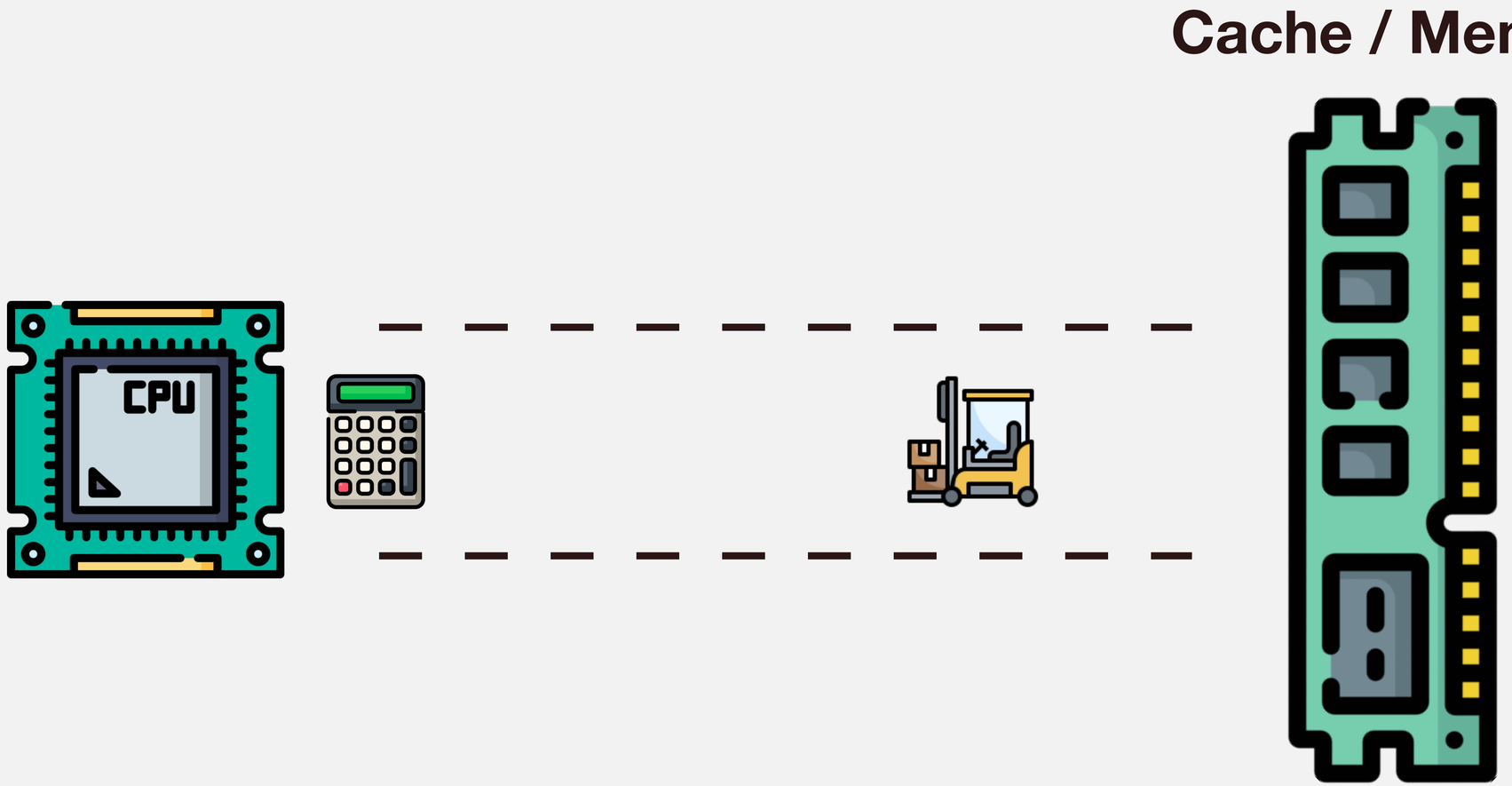
CPU

- ➜ Reduce Operator Strength
- ➜ Reduce Branch Misprediction
- ➜ Reduce Data Dependency
- ➜ SIMD

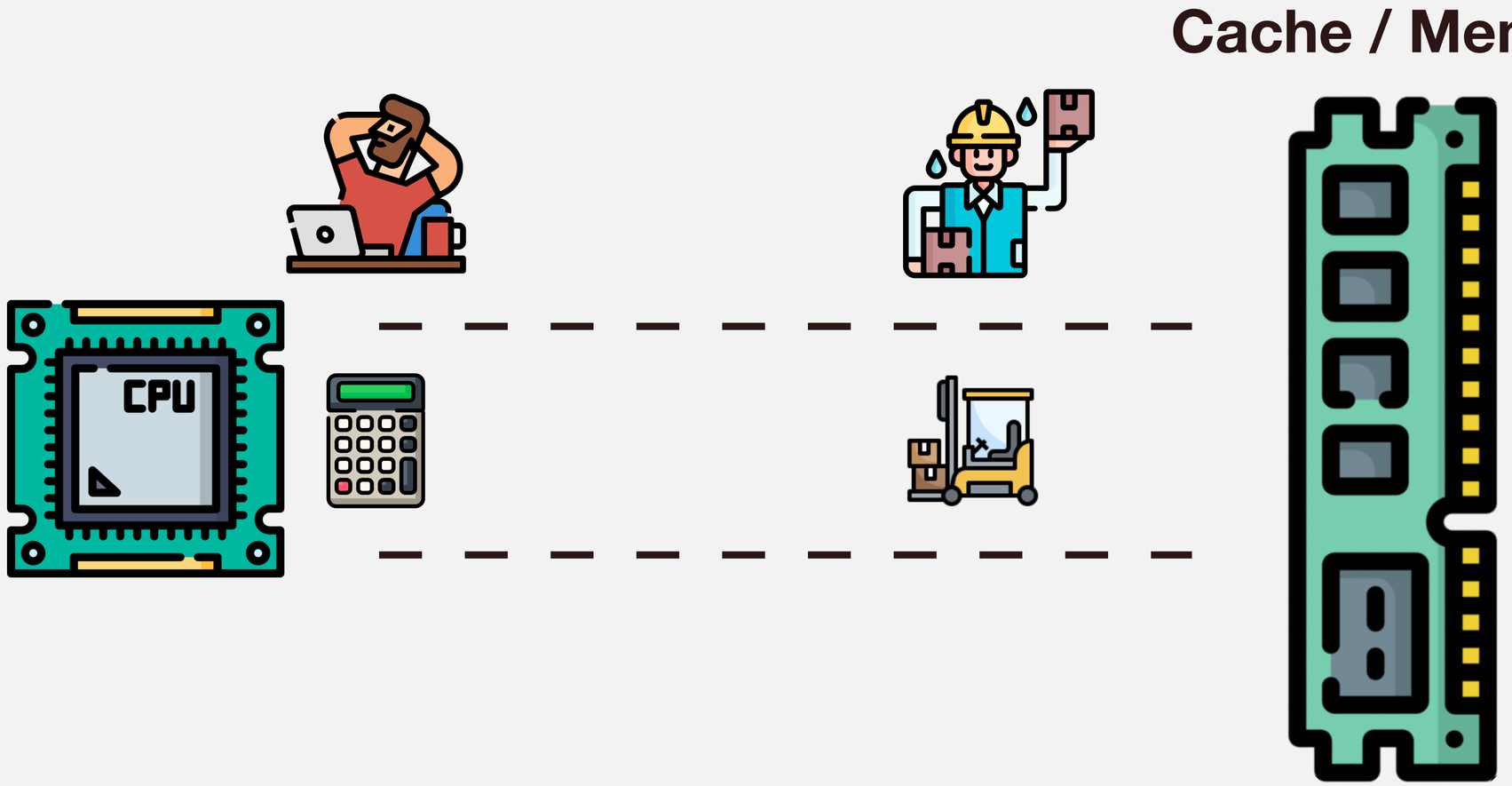
Cache

- ➜ Temporal Locality
- ➜ Spacial Locality

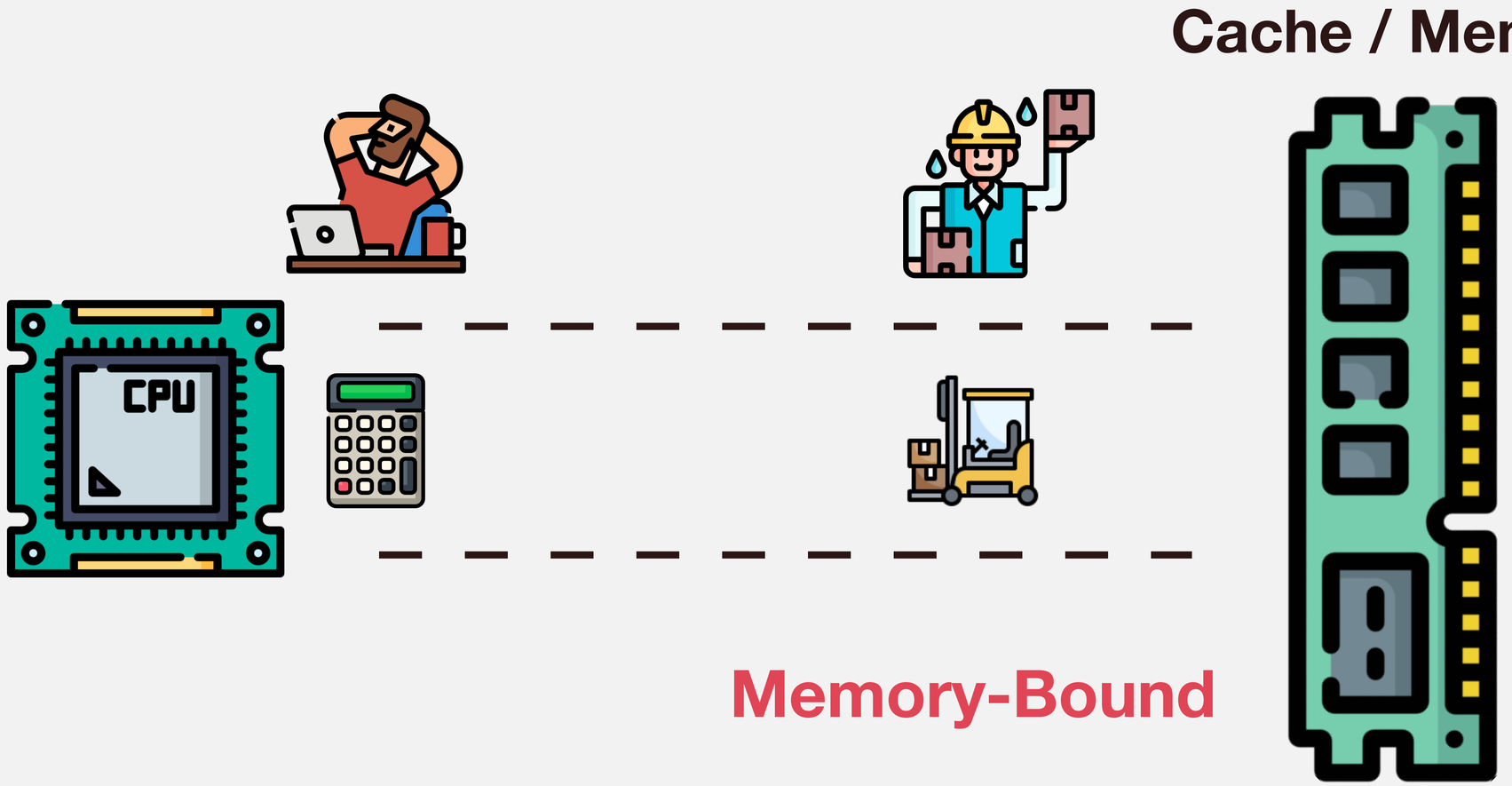
Who’s the Bottleneck?



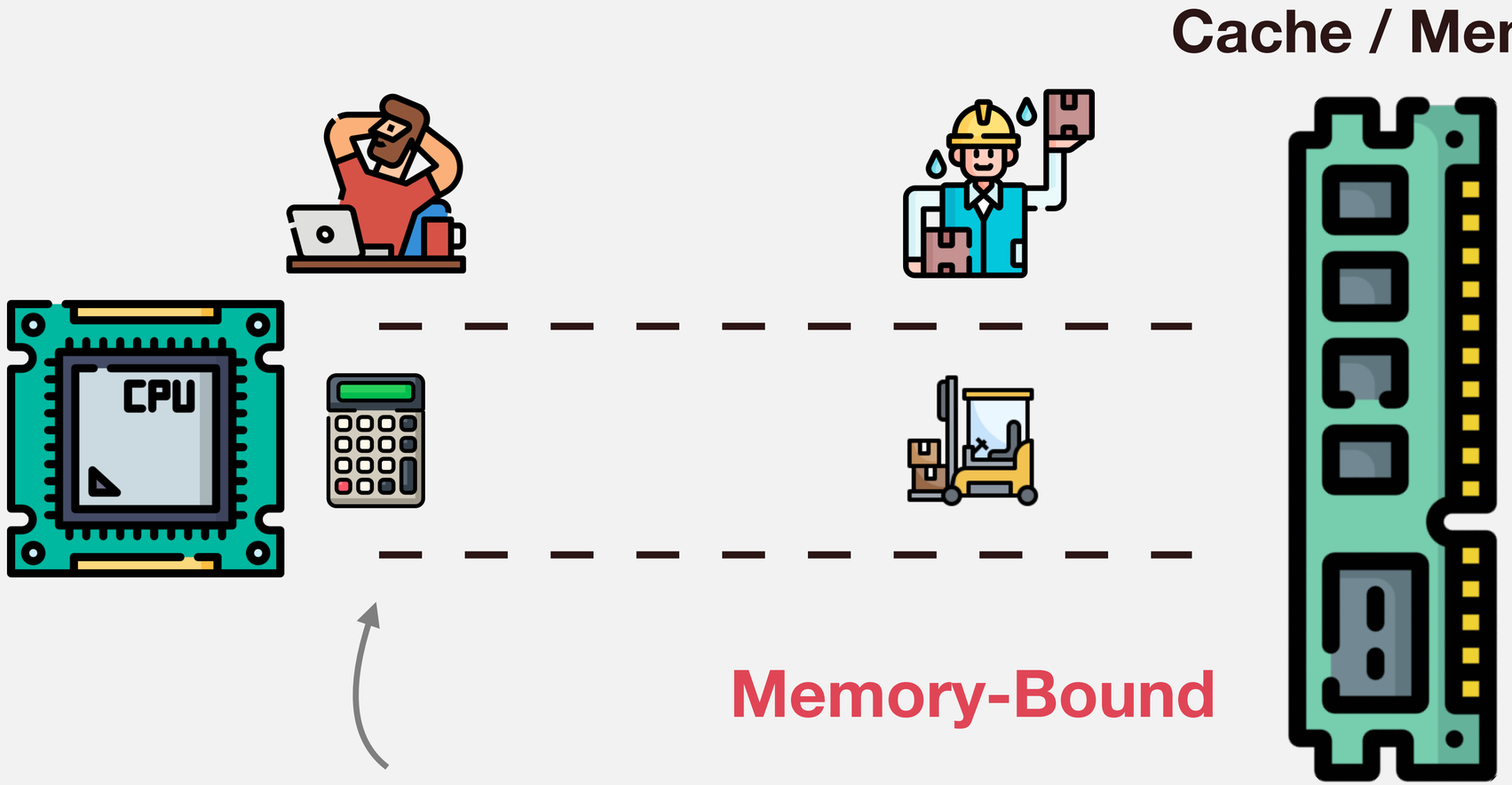
Who’s the Bottleneck?



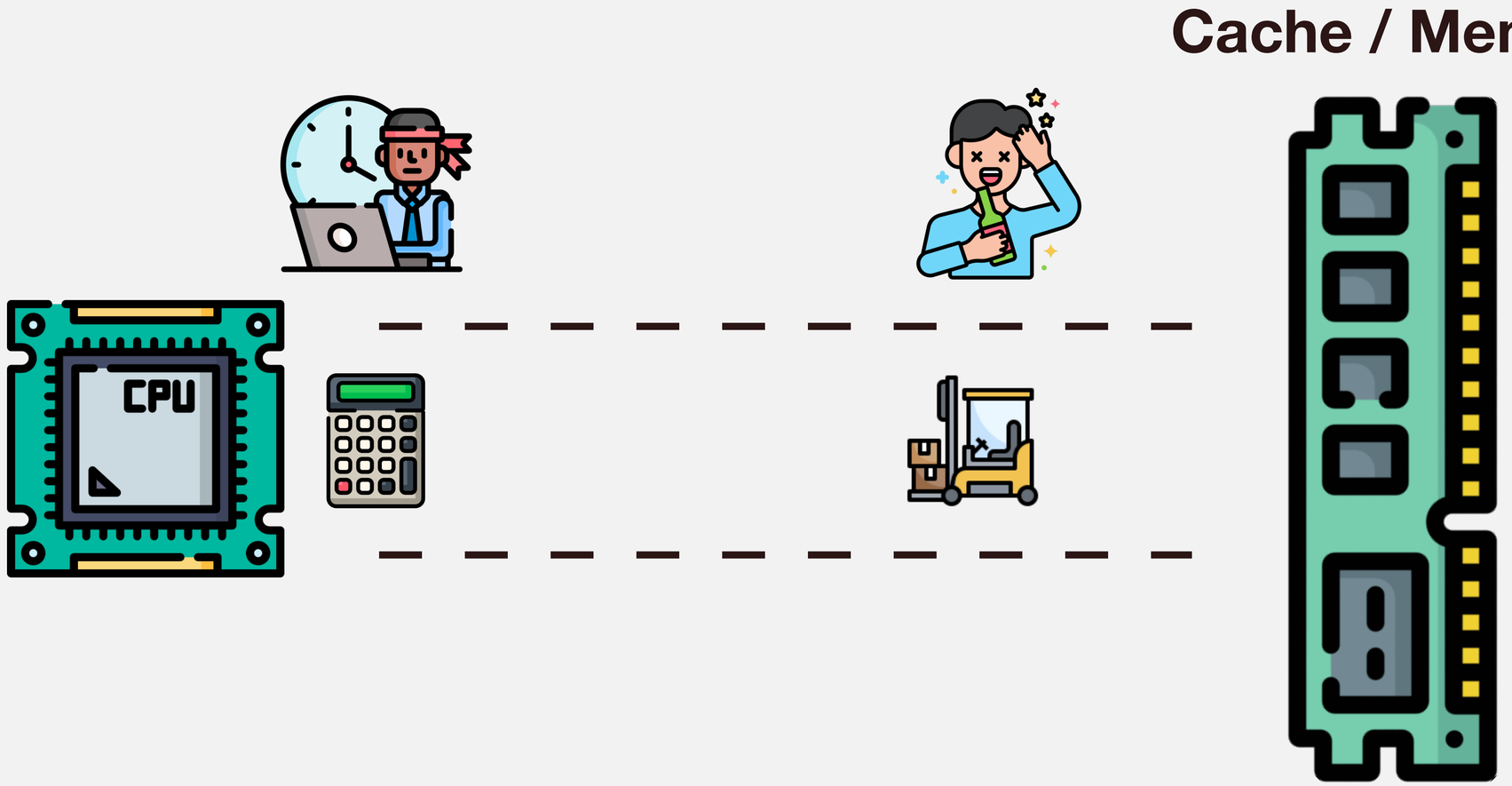
Who’s the Bottleneck?



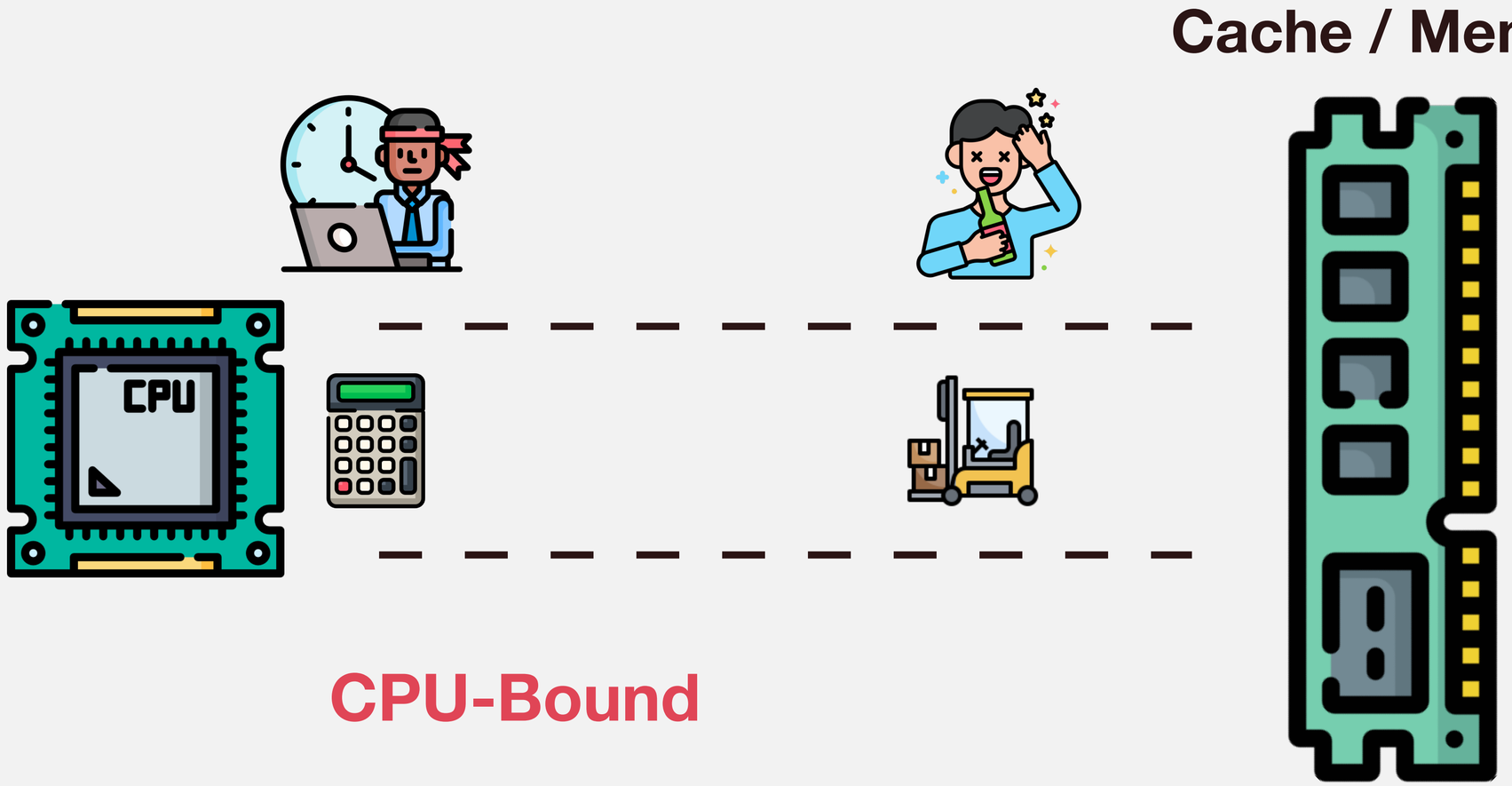
Who’s the Bottleneck?



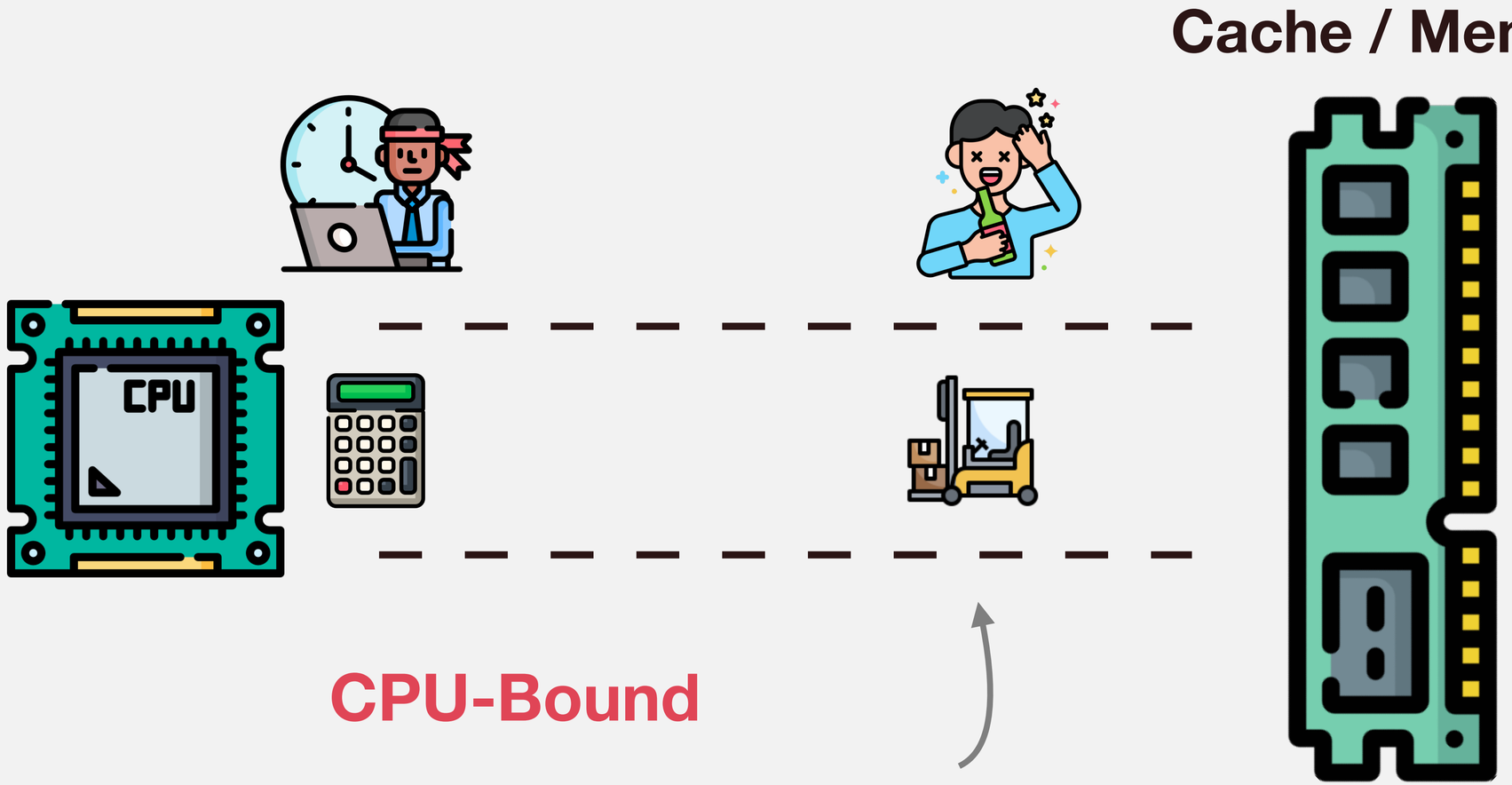
Who’s the Bottleneck?



Who’s the Bottleneck?

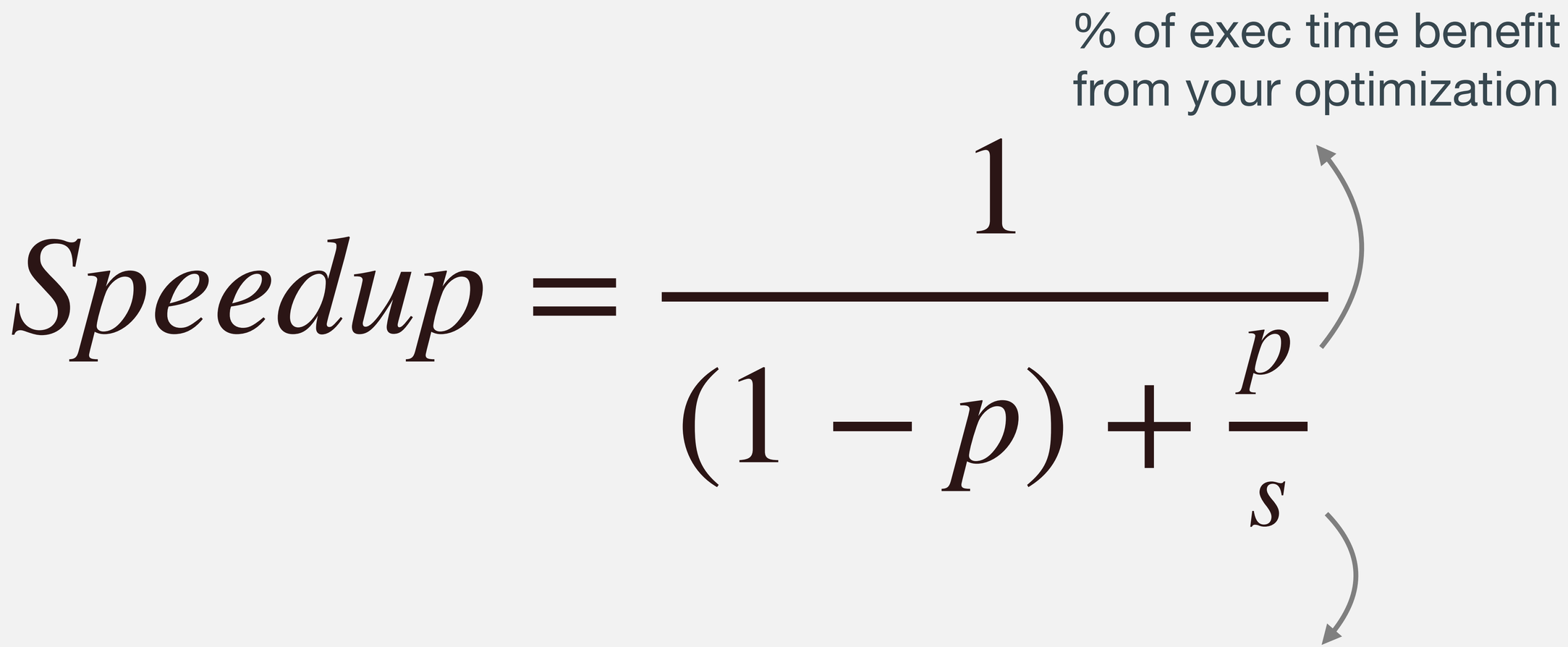


Who’s the Bottleneck?



Optimize cache performance has no gain

Don’t Forget Amdahl’s Law


% of exec time benefit
from your optimization
$$Speedup = \frac{1}{(1-p) + \frac{p}{s}}$$
speedup of that
portion of code

Optimizing Code Performance

- ➜ Design efficient algorithms

Optimizing Code Performance

- ➜ Design efficient algorithms
- ➜ Eliminate unnecessary work

Optimizing Code Performance

- ➜ Design efficient algorithms
- ➜ Eliminate unnecessary work
- ➜ Leverage compiler optimization

Optimizing Code Performance

- ➜ Design efficient algorithms
- ➜ Eliminate unnecessary work
- ➜ Leverage compiler optimization

- ➜ Detect hotspots / bottleneck

Optimizing Code Performance

- ➜ Design efficient algorithms
- ➜ Eliminate unnecessary work
- ➜ Leverage compiler optimization

- ➜ Detect hotspots / bottleneck
- ➜ Computation optimization techniques

Optimizing Code Performance

- ➜ Design efficient algorithms
- ➜ Eliminate unnecessary work
- ➜ Leverage compiler optimization

- ➜ Detect hotspots / bottleneck
- ➜ Computation optimization techniques
- ➜ Cache optimization techniques

Optimizing Code Performance

- ➜ Design efficient algorithms
- ➜ Eliminate unnecessary work
- ➜ Leverage compiler optimization

- ➜ Detect hotspots / bottleneck
- ➜ Computation optimization techniques
- ➜ Cache optimization techniques