Security research

Tamara Rezk

C3
Monday, September 26th

Previous lesson:

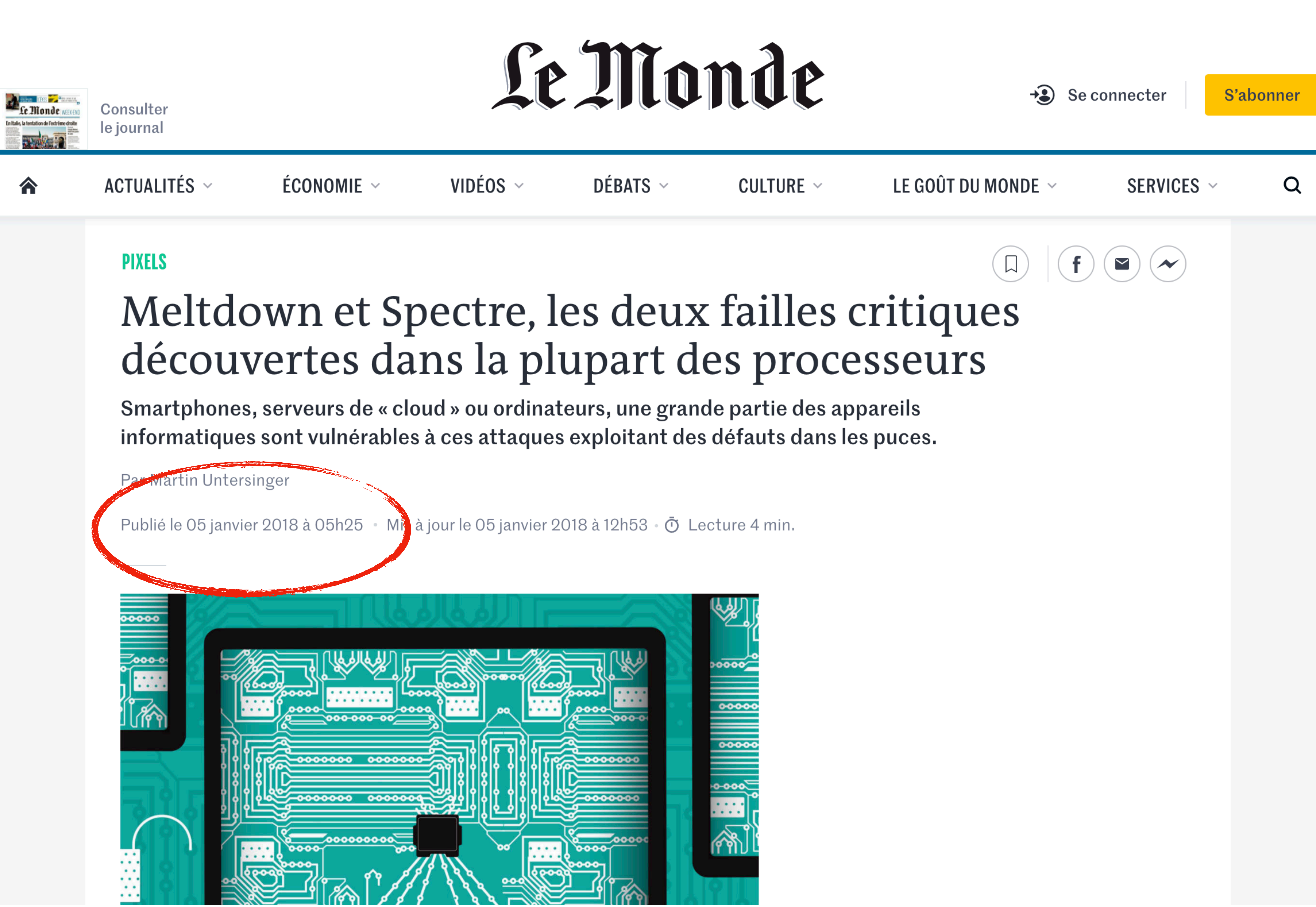
Vulnerability Finding training (Capture the Flag)

goal : get basis on tools and manual vulnerability discovery

Today:

Study of a class of attacks disclosed in early 2018

Transient Execution Attacks



Consulter le journal

Le Monde

Se connecter

S'abonner

ACTUALITÉS ÉCONOMIE VIDÉOS DÉBATS CULTURE LE GOÛT DU MONDE SERVICES

PIXELS

Meltdown et Spectre, les deux failles critiques découvertes dans la plupart des processeurs

Smartphones, serveurs de « cloud » ou ordinateurs, une grande partie des appareils informatiques sont vulnérables à ces attaques exploitant des défauts dans les puces.

Par Martin Untersinger

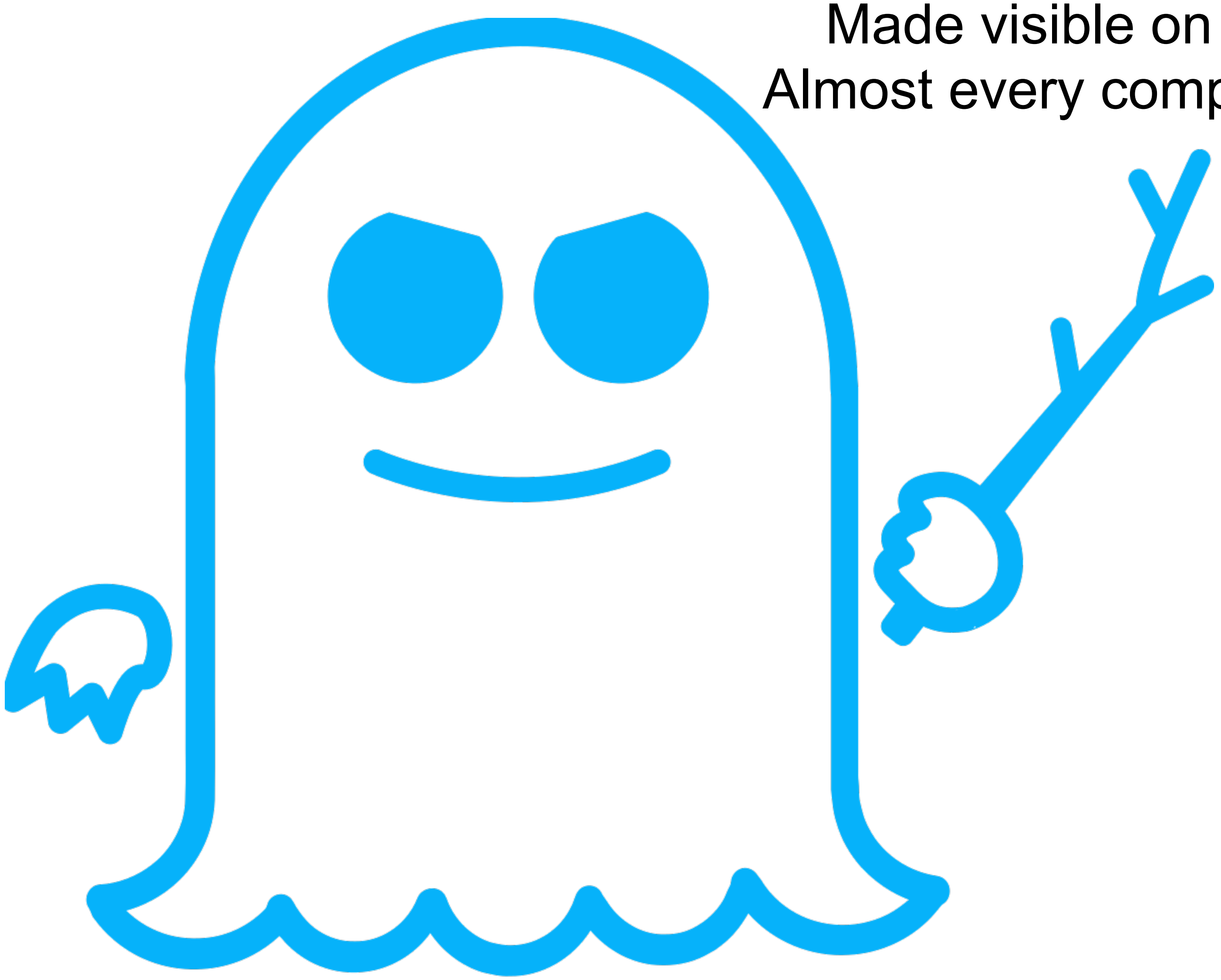
Publié le 05 janvier 2018 à 05h25 • Mis à jour le 05 janvier 2018 à 12h53 • Lecture 4 min.

```
int
crypto_secretbox_xsalsa20poly1305( unsigned char *c,
                                 const unsigned char *m,
                                 unsigned long long men,
                                 const unsigned char *n,
                                 const unsigned char *k) {
          int i;

          if (mlen < 32) {
              return -1;
          }
          crypto_stream_xsalsa20_xor(c, m, mlen, n, k);
          crypto_onetimeauth_poly1305( c + 16, c + 32, mlen -
          32, c);
          for (i = 0; i < 16; ++i) {
                c[i] = 0;
          }
          return 0;
 }
```

```
int
crypto_secretbox_xsalsa20_poly1305( unsigned char *c,
                                 const unsigned char *m,
                                 unsigned long long men,
                                 const unsigned char *n,
                                 const unsigned char *k) {
          int i;

          if (mlen < 32) {
              return -1;
          }
          crypto_stream_xsalsa20_xor(c, m, mlen, n, k);
          crypto_onetimeauth_poly1305( c + 16, c + 32, mlen -
          32, c);
          for (i = 0; i < 16; ++i) {
              c[i] = 0;
          }
          return 0;
 }
```

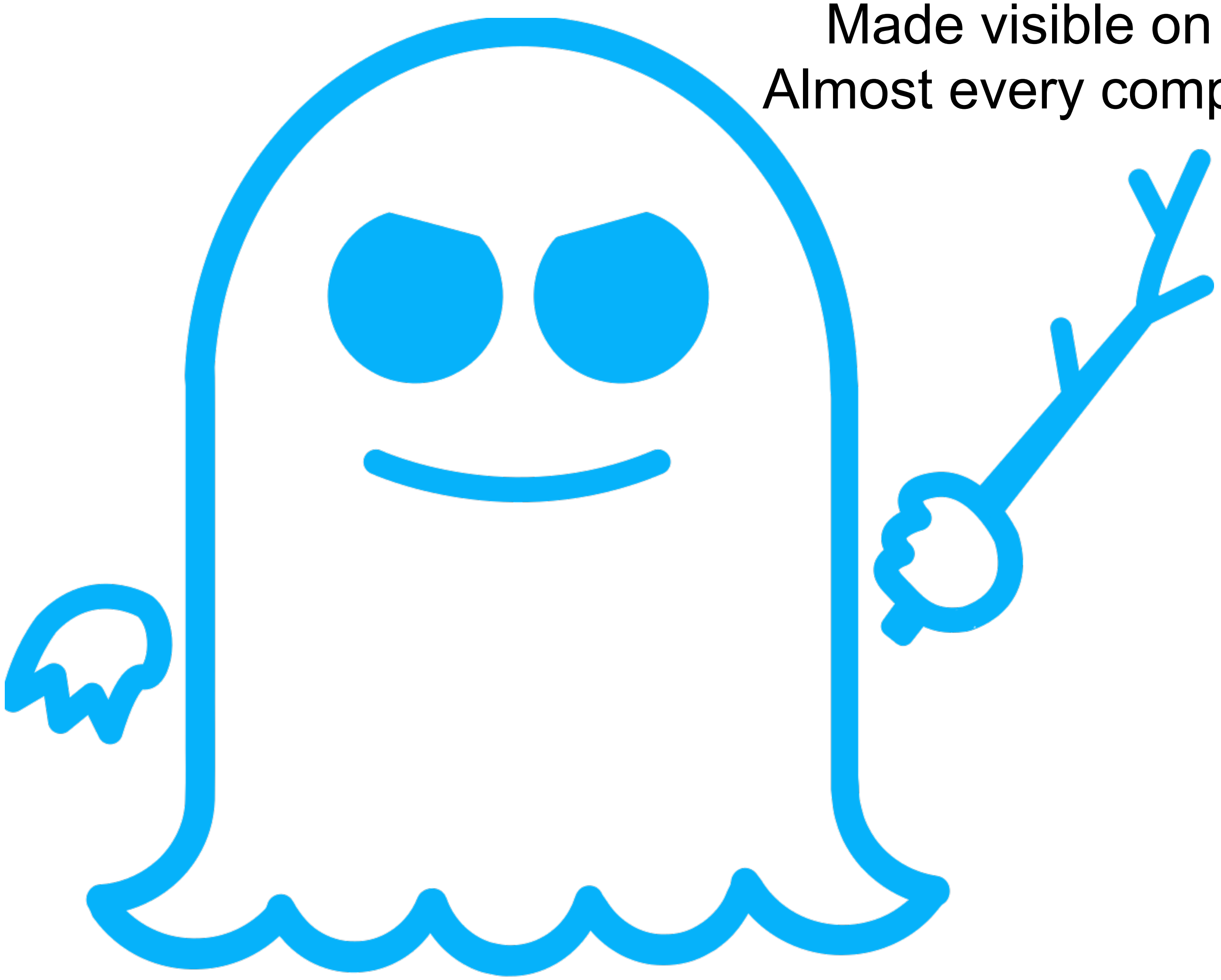
Spectre vulnerabilities

Made visible on January, 3rd 2018
Almost every computer system affected



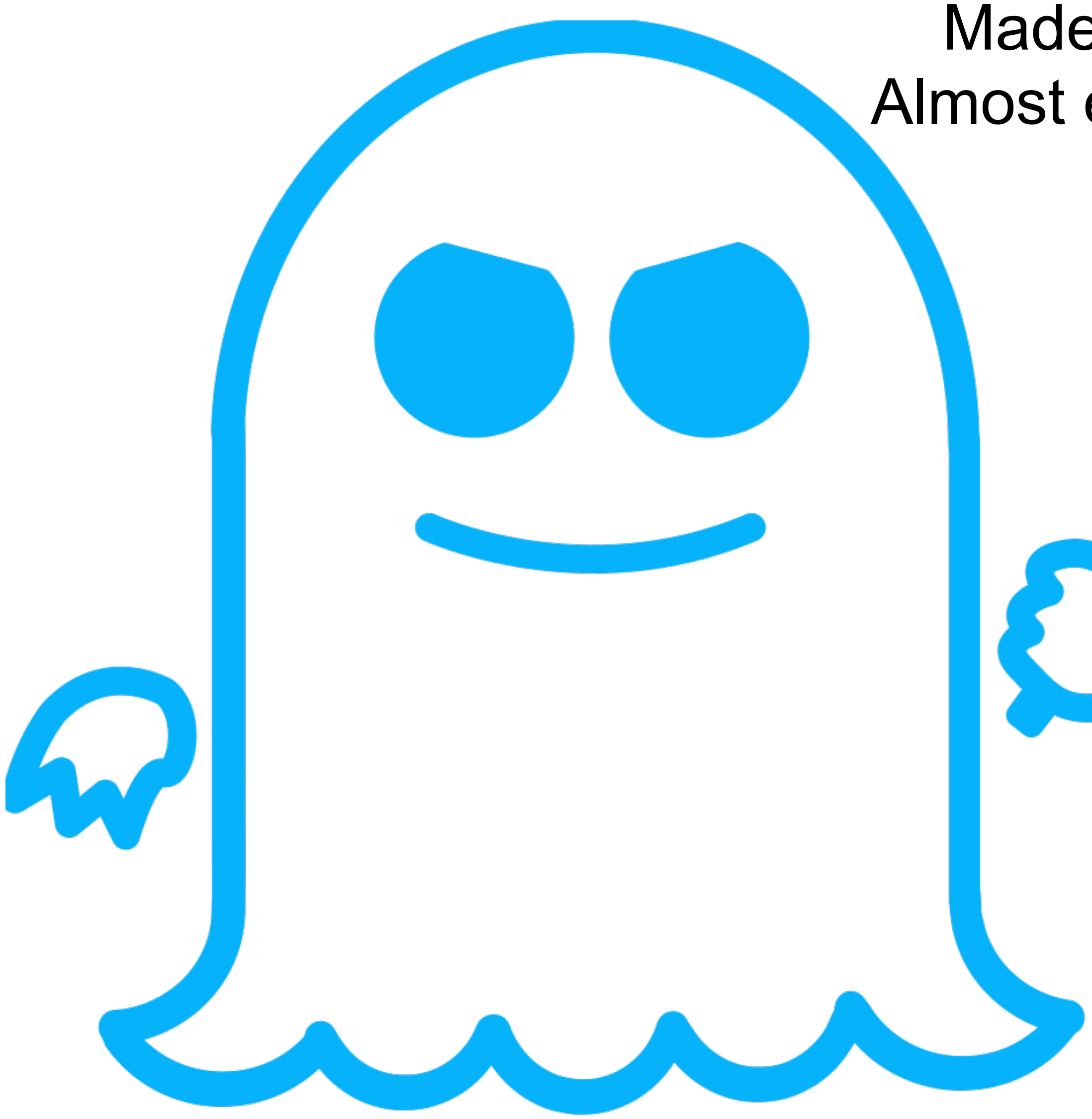
Spectre vulnerabilities

Made visible on January, 3rd 2018
Almost every computer system affected



Spectre vulnerabilities

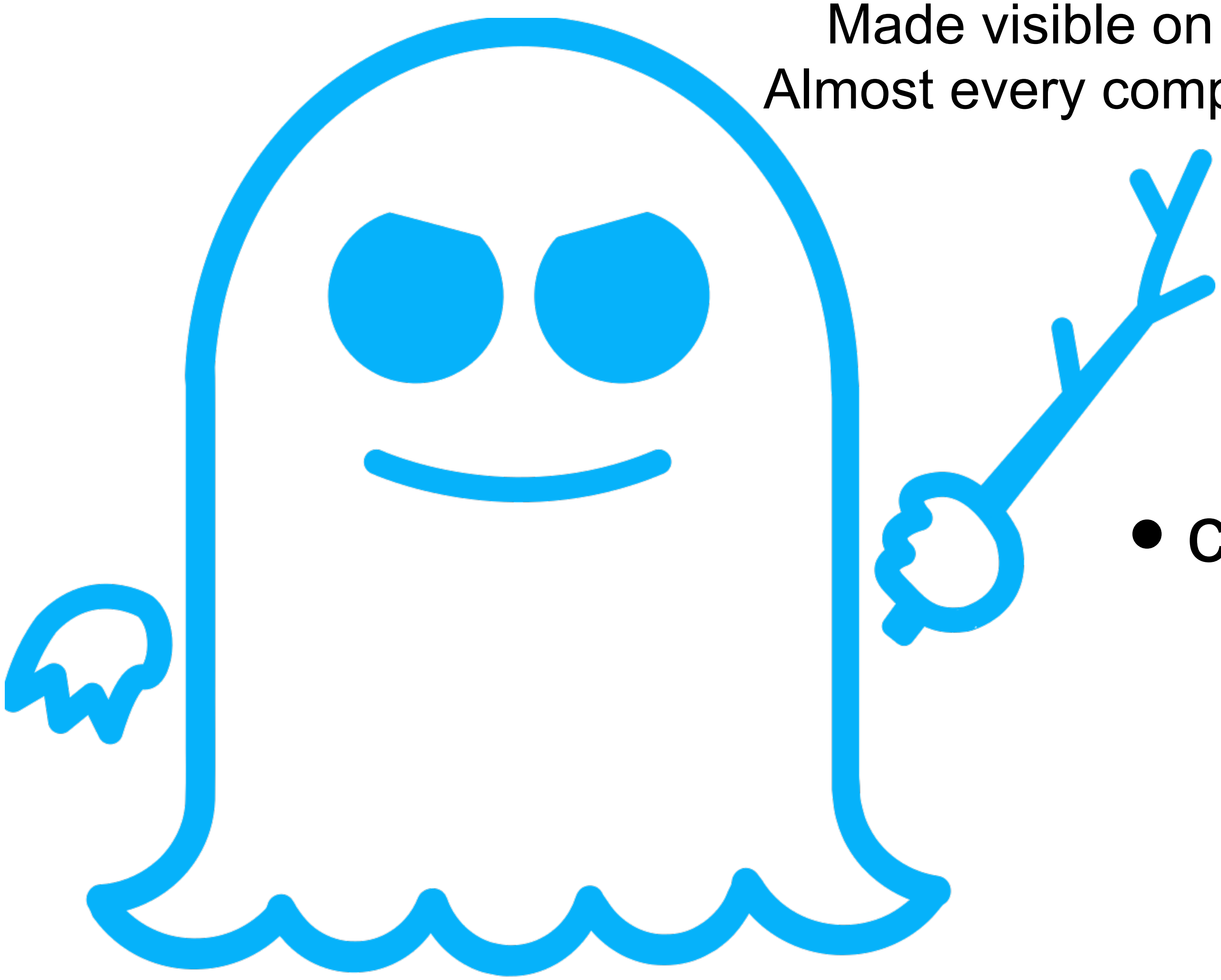
Made visible on January, 3rd 2018
Almost every computer system affected



- constant-time

Spectre vulnerabilities

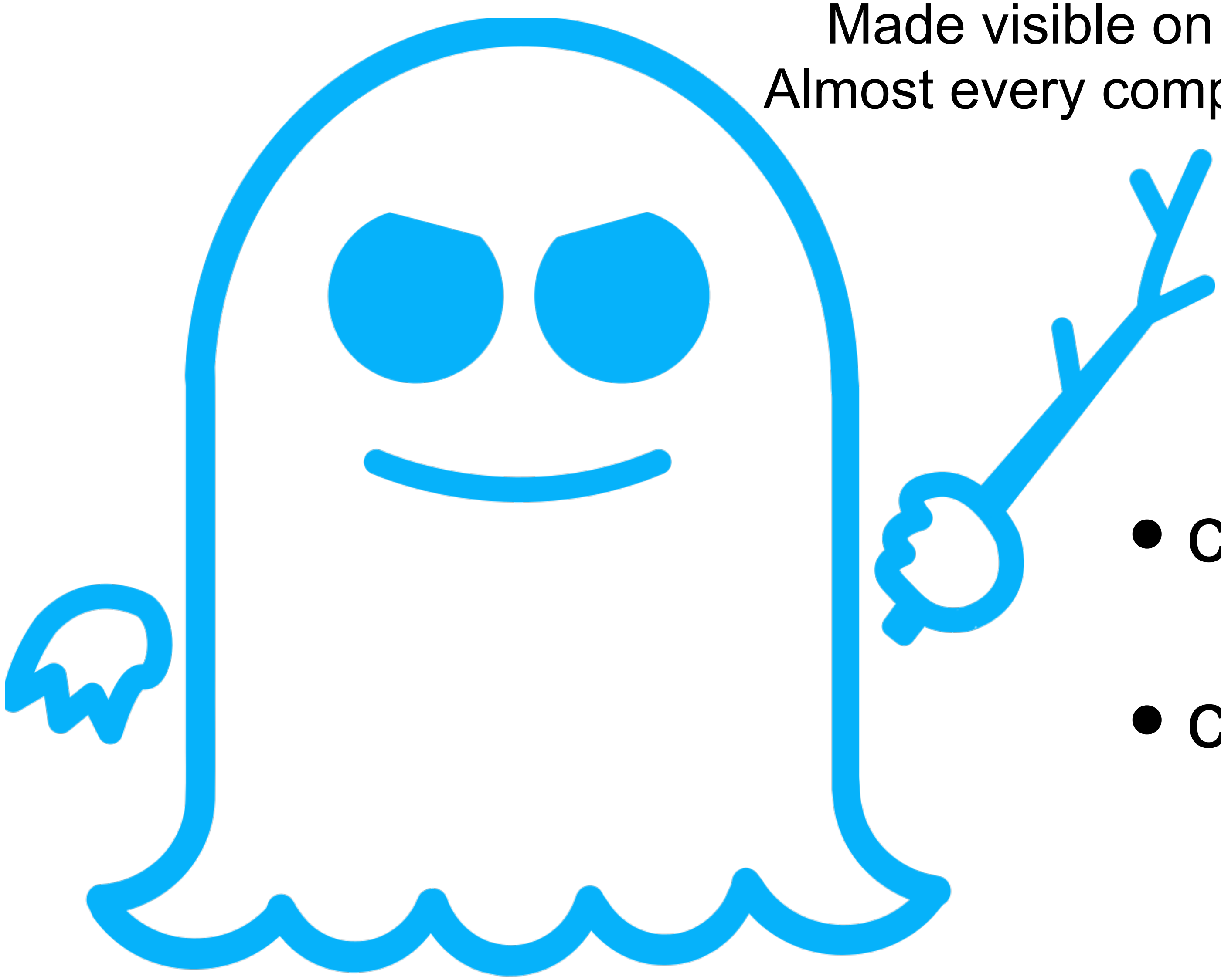
Made visible on January, 3rd 2018
Almost every computer system affected



- constant-time

Spectre vulnerabilities

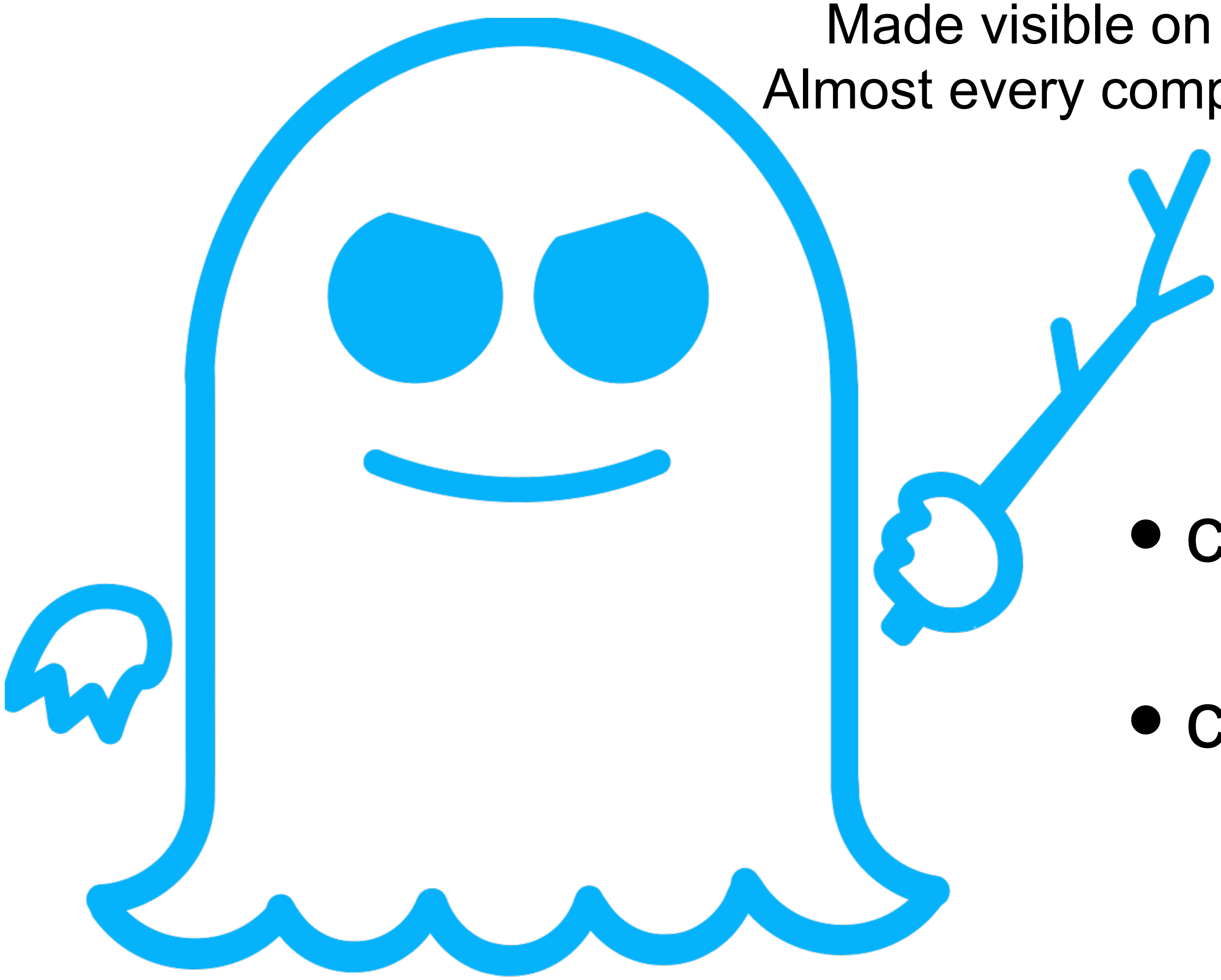
Made visible on January, 3rd 2018
Almost every computer system affected



- constant-time
- cache side-channels attacks

Spectre vulnerabilities

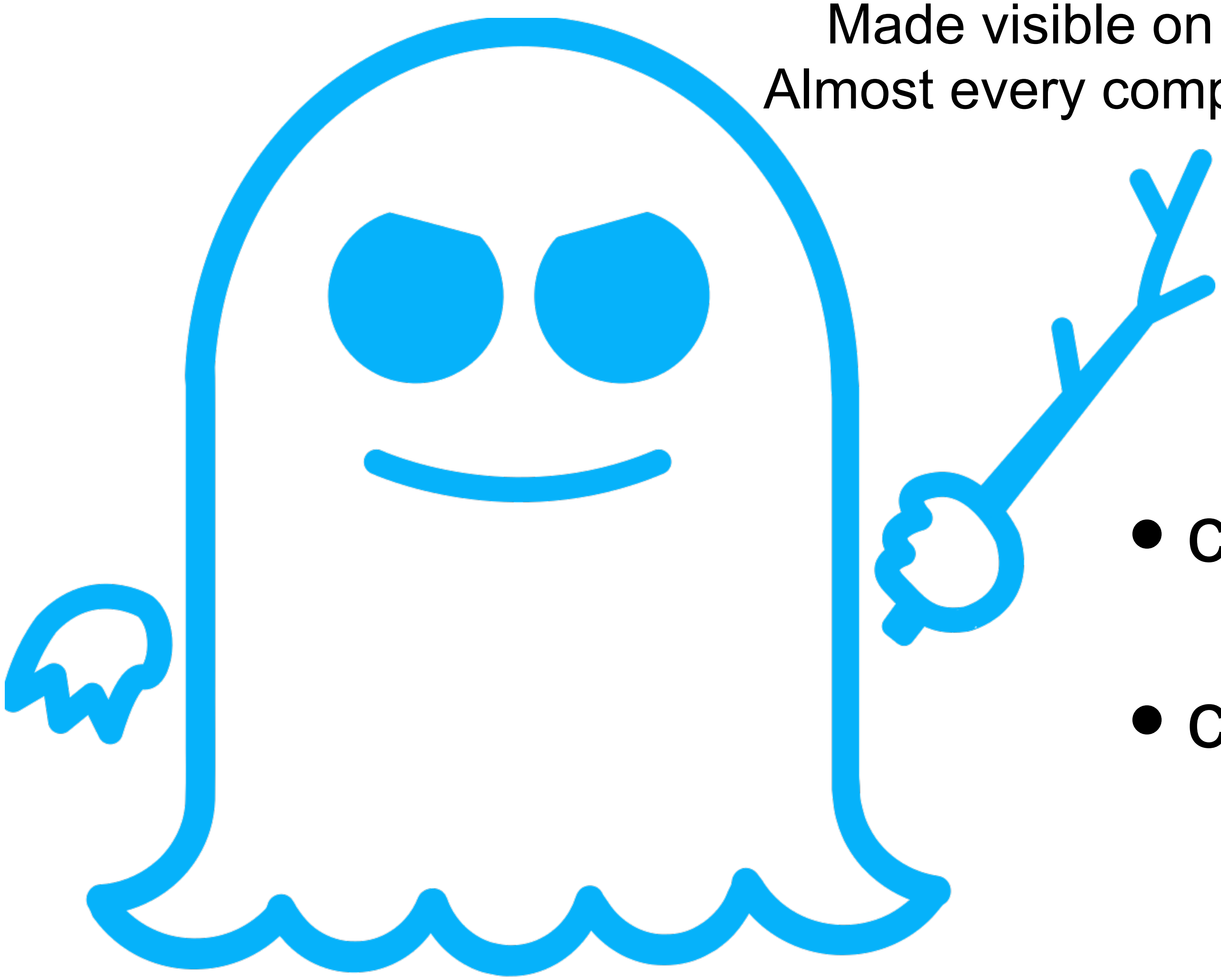
Made visible on January, 3rd 2018
Almost every computer system affected



- constant-time
- cache side-channels attacks

Spectre vulnerabilities

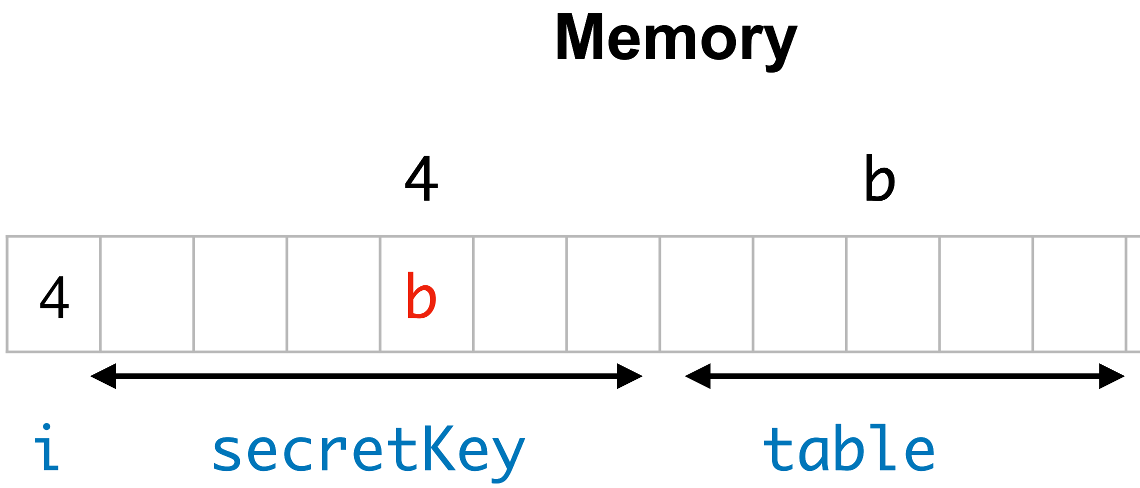
Made visible on January, 3rd 2018
Almost every computer system affected



- constant-time
- cache side-channels attacks

Cache side-channels

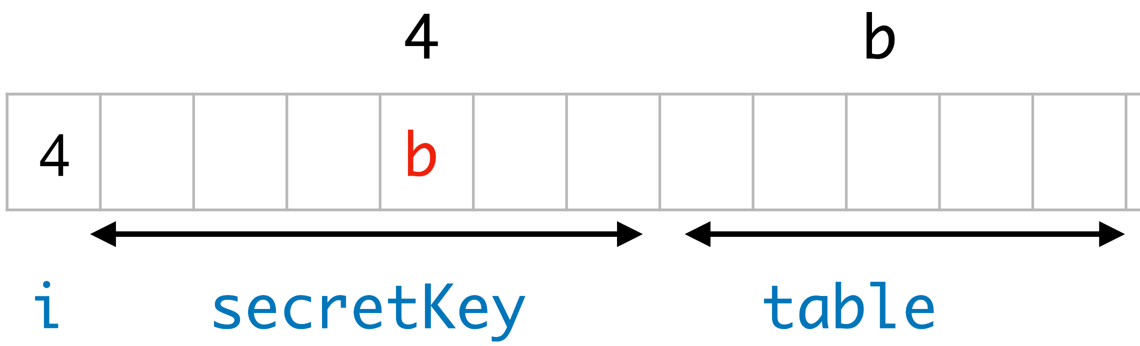
```
b = secretKey[i];

s = table[b];

//. . .
```



Cache side-channels

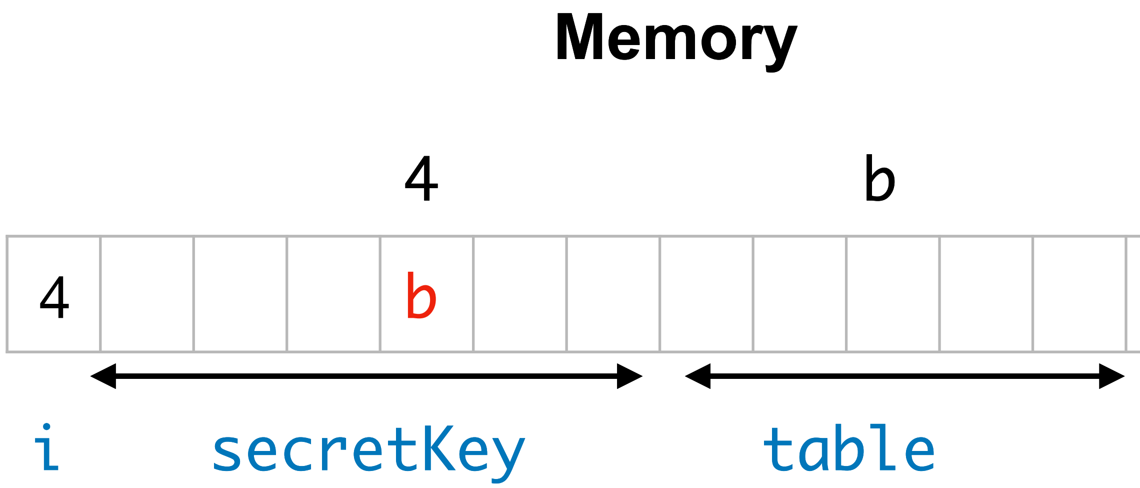
```
b = secretKey[i];

s = table[b];

//. . .
```

Memory



Cache side-channels

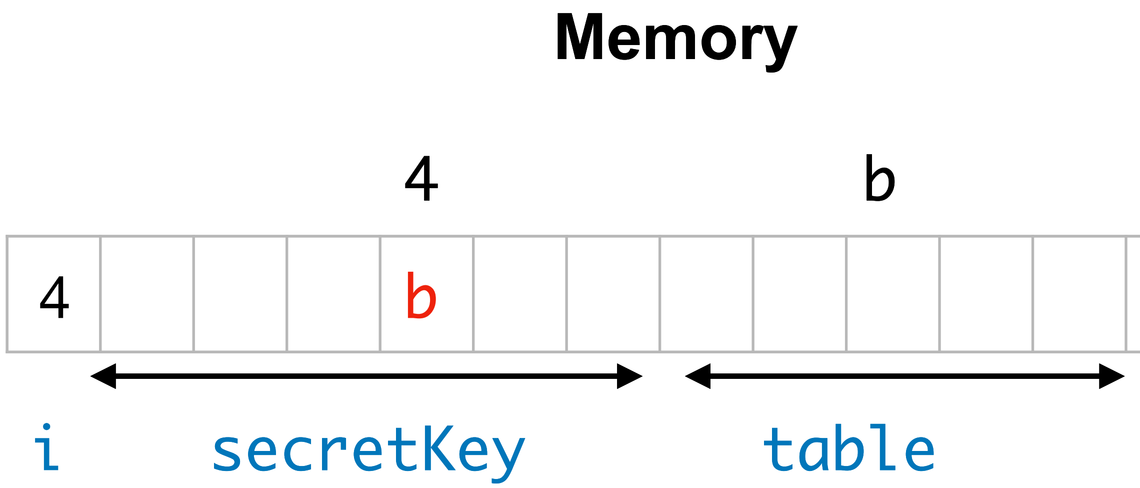
```
b = secretKey[i];

s = table[b];

//. . .
```



Cache side-channels

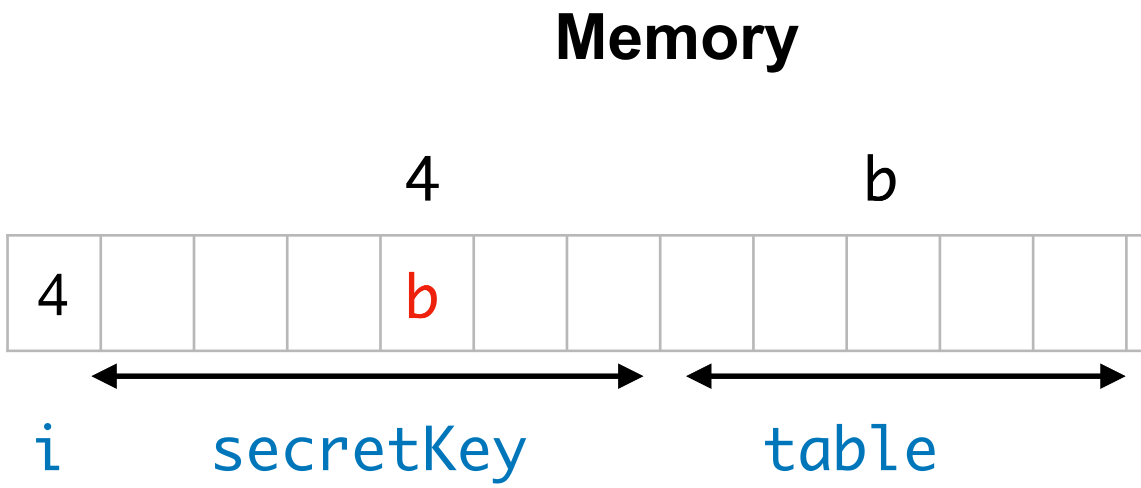
```
b = secretKey[i];

s = table[b];

//. . .
```

Leaks secretKey!



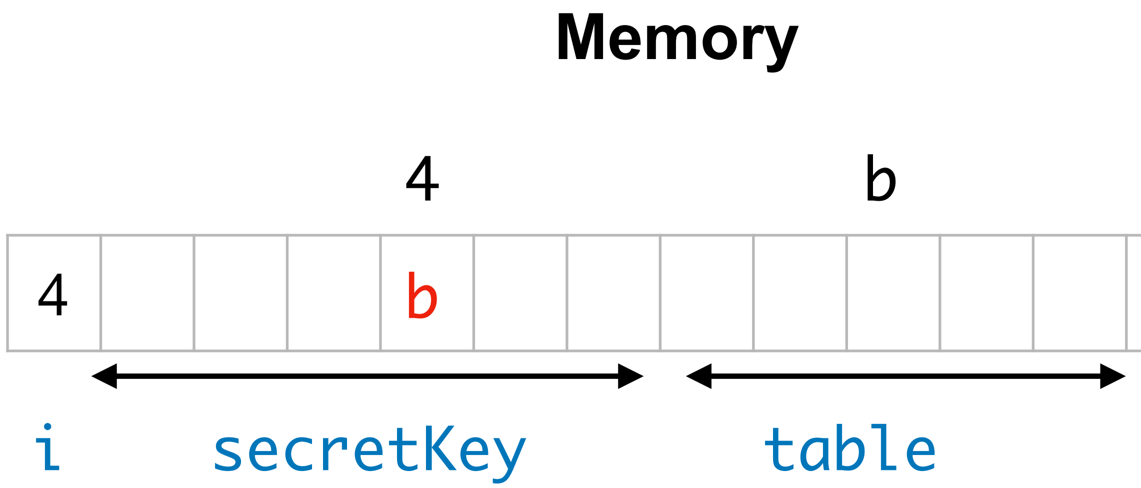
Cache side-channels

```
b = secretKey[i];

s = table[b];

//. . .
```



Cache side-channels

```
b = secretKey[i];

s = table[b];

//. . .
```



Memory

4

b

4

b

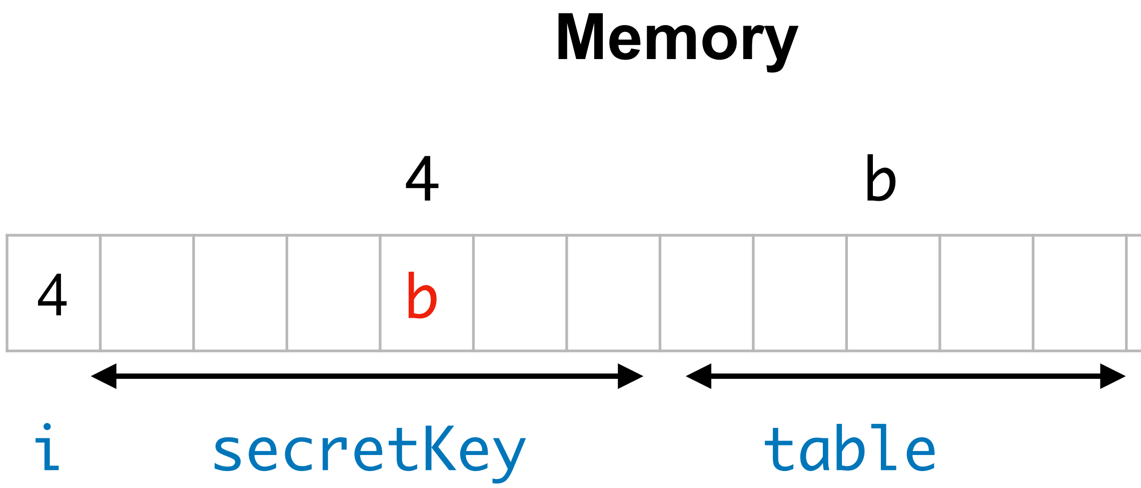
i

secretKey

table

Cache side-channels

```
b = secretKey[i];

s = table[b];

//. . .
```



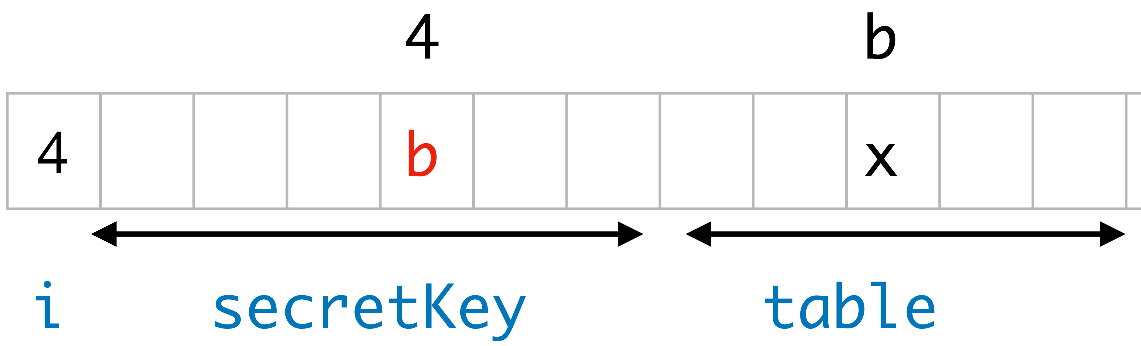
Cache side-channels

```
b = secretKey[i];

s = table[b];

//. . .
```

Cache

addr	data
…	
…	…
…	

Memory



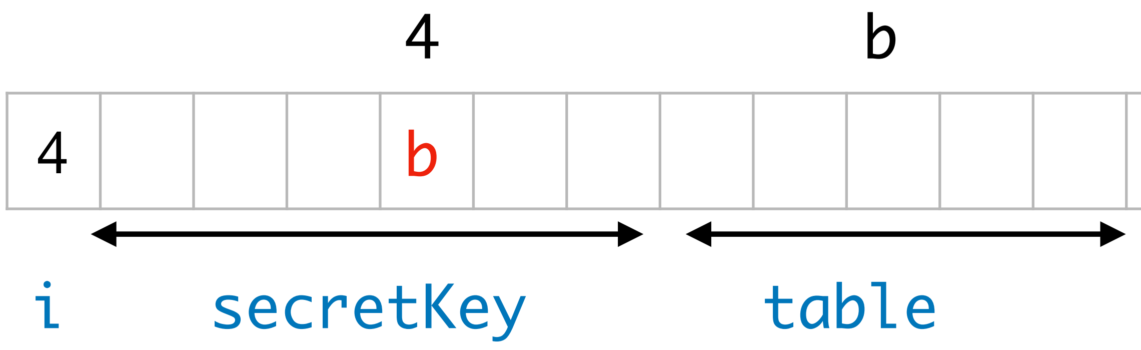
Cache side-channels

```
b = secretKey[i];

s = table[b];

//. . .
```

Cache

addr	data
…	
table+b	x
…	…
…	

Memory



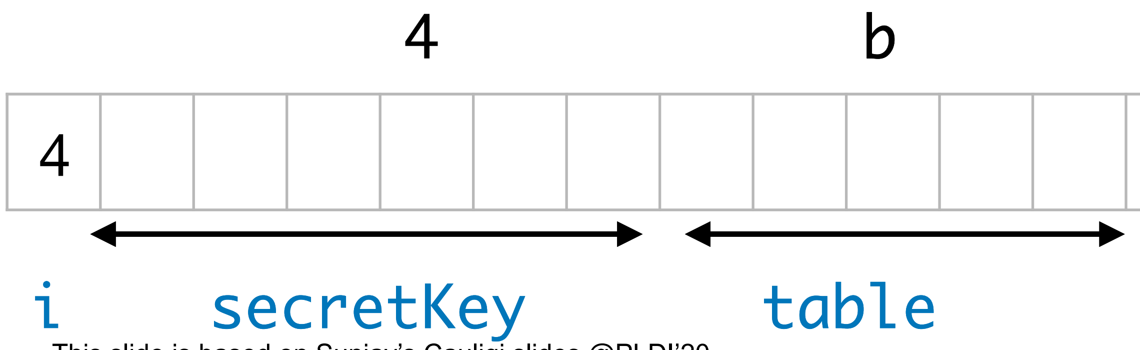
Cache side-channels

```
b = secretKey[i];

s = table[b];

//. . .
```

Cache

addr	data
...	
table+b	x
...	
...	

Memory



This slide is based on Sunjay's Cauligi slides @PLDI'20.

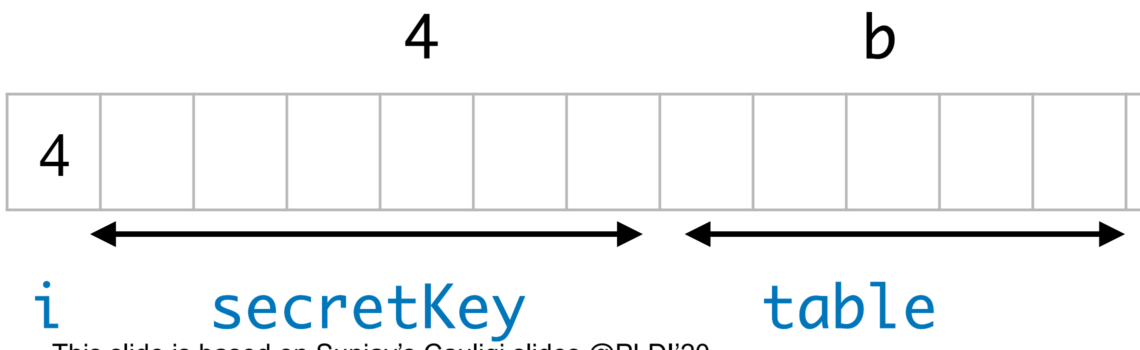
Cache side-channels

```
b = secretKey[i];

s = table[b];

//. . .
```

Cache

addr	data
…	
table+b	x
…	
…	

Memory



This slide is based on Sunjay's Cauligi slides @PLDI'20.

Cache side-channels

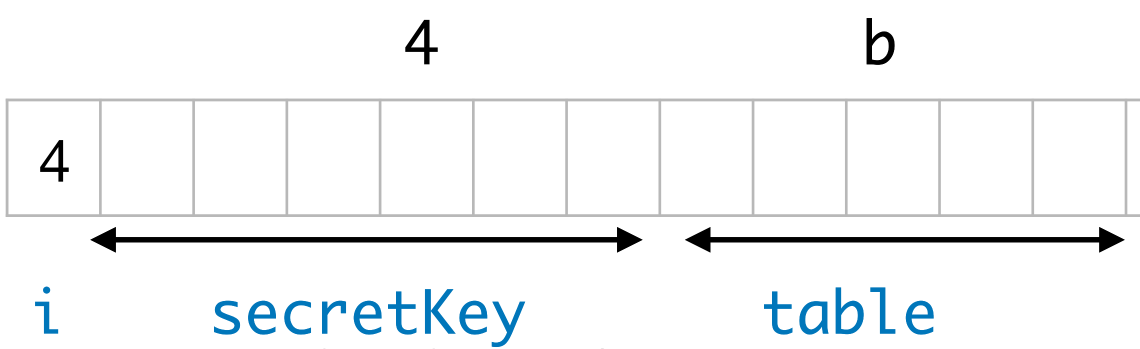
```
b = secretKey[i];

s = table[b];

//. . .
```

table[n]	access time
0	slow
…	slow
b	**fast**
…	

Cache

addr	data
…	
table+b	x
…	
…	

Memory



This slide is based on Sunjay's Cauligi slides @PLDI'20.

Cache side-channels

```
b = secretKey[i];

s = table[b];

//. . .
```

table[n]	access time
0	slow
…	slow
b	**fast**
…	

Cache

addr	data
…	
table+b	x
…	
…	

This slide is based on Sunjay's Cauligi slides @PLDI'20.

Cache side-channels

```
b = secretKey[i];

s = table[b];

//. . .
```

table[n]	access time
0	slow
…	slow
b	**fast**
…	

recently accessed

Cache

addr	data
…	
table+b	x
…	
…	

This slide is based on Sunjay’s Cauligi slides @PLDI’20.

Cache side-channels

```
b = secretKey[i];

s = table[b];

//. . .
```

table[n]	access time
0	slow
…	slow
b	fast
…	

recently accessed

Attacker gets value b

Cache

addr	data
…	
table+b	x
…	
…	

This slide is based on Sunjay's Cauligi slides @PLDI'20.

Cache side-channels

```
b = secretKey[i];

s = table[b];

//. . .
```

table[n]	access time
0	slow
…	slow
b	**fast**
…	

recently accessed

Attacker gets value b

[Lucky thirteen: Breaking the TLS and DTLS record protocols. IEEE S&P 2013]

Constant-time programming

Prevention against cache side-channel attacks

Constant-time programming

Prevention against cache side-channel attacks

Secrets must not influence …

Constant-time programming

Prevention against cache side-channel attacks

Secrets must not influence …

- Control flow

Constant-time programming

Prevention against cache side-channel attacks

Secrets must not influence …

- Control flow

Constant-time programming

Prevention against cache side-channel attacks

Secrets must not influence …

- Control flow
- Memory accesses

Constant-time programming

Prevention against cache side-channel attacks

Secrets must not influence …

- Control flow
- **Memory accesses** → Array indices

table[n]	access time
0	slow
1	slow
2	fast
…	

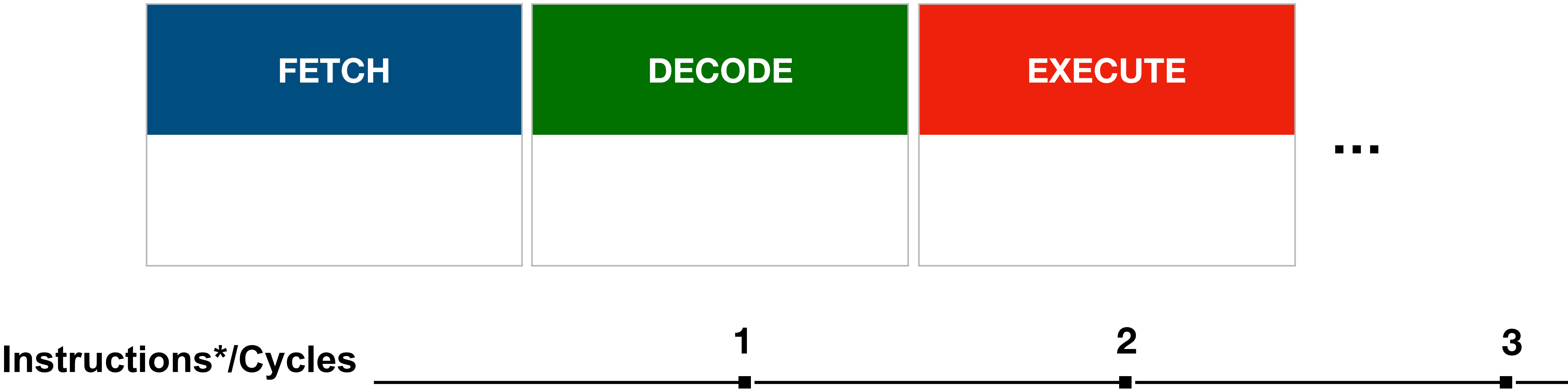
No secrets in cache!

⚠

These constant-time programming rules fail to account on how modern processors process instructions!!

Processing instructions

Stages



FETCH

DECODE

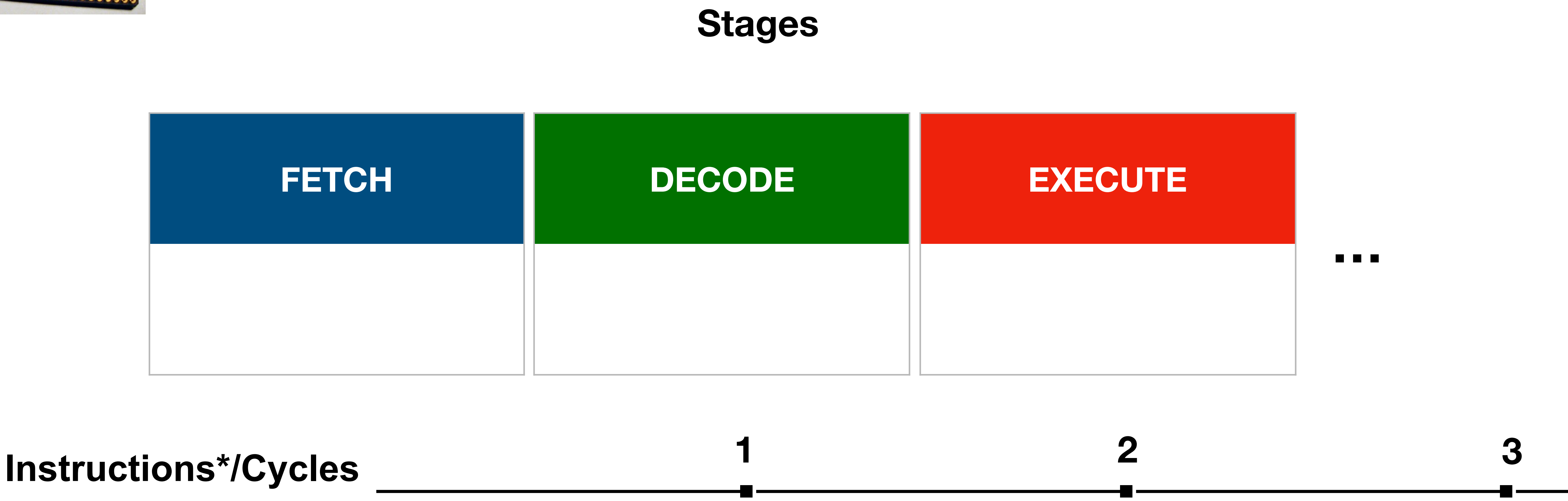
EXECUTE

...

Instructions*/Cycles

```
1.rb =load [40+ra]
```

Processing instructions



Stages

FETCH

DECODE

EXECUTE

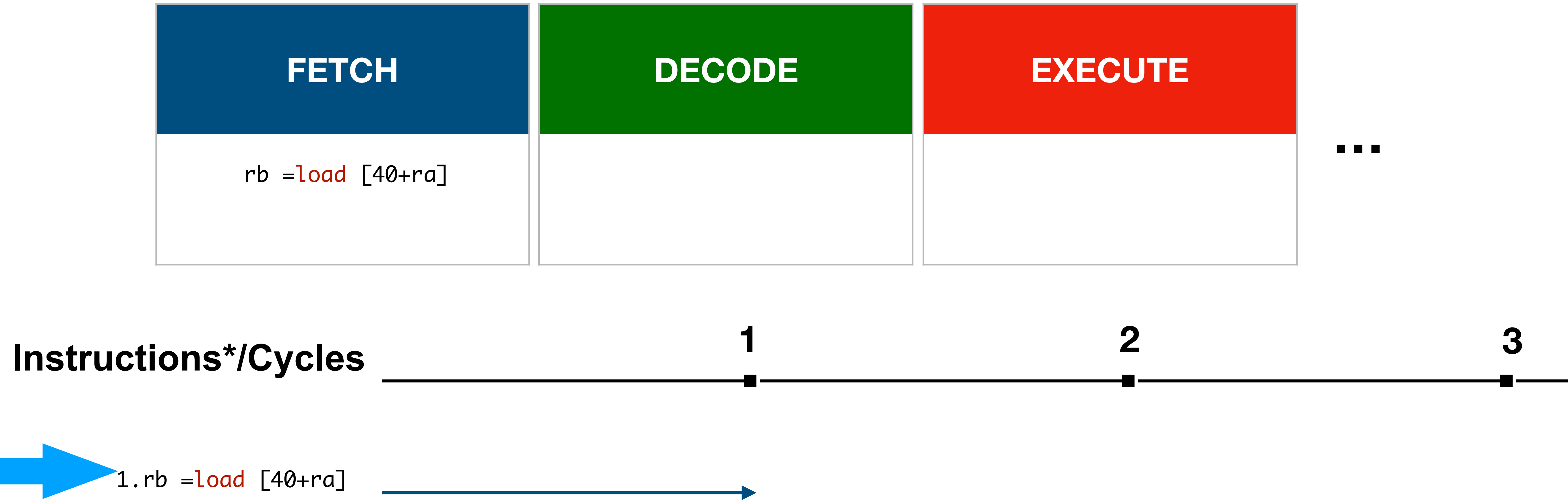
...

Instructions*/Cycles

```
1.rb =load [40+ra]
```

Processing instructions

Stages



FETCH

DECODE

EXECUTE

`rb =load [40+ra]`

...

Instructions*/Cycles

1

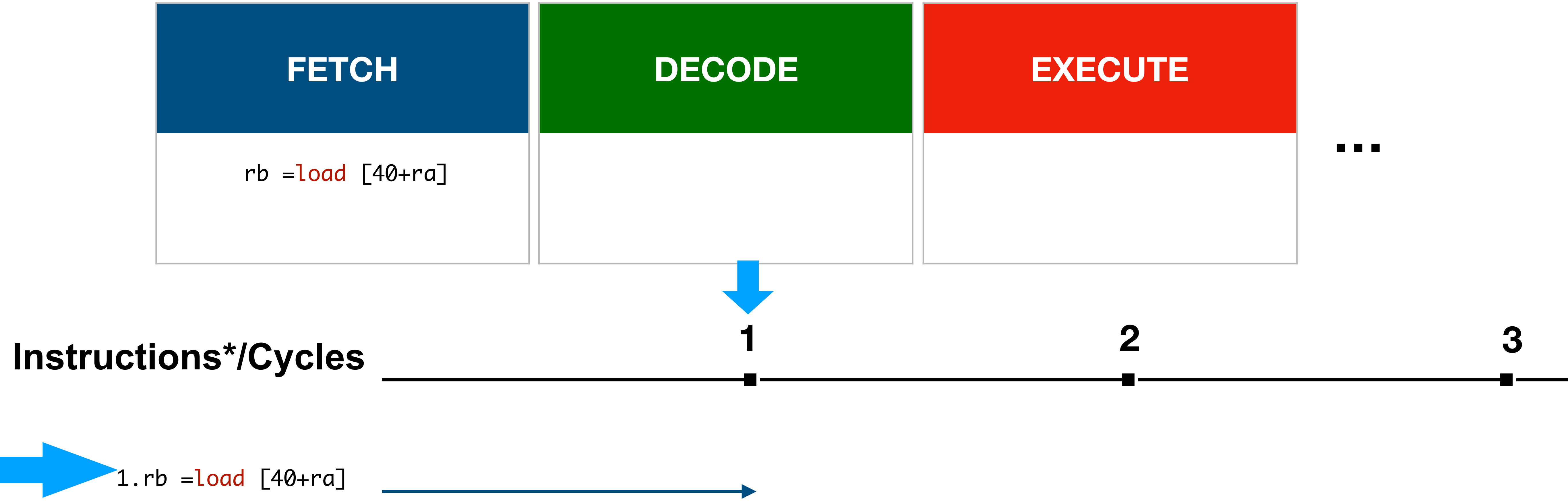
2

3

`1.rb =load [40+ra]`

Processing instructions

Stages



FETCH

DECODE

EXECUTE

`rb =load [40+ra]`

...

Instructions*/Cycles

1

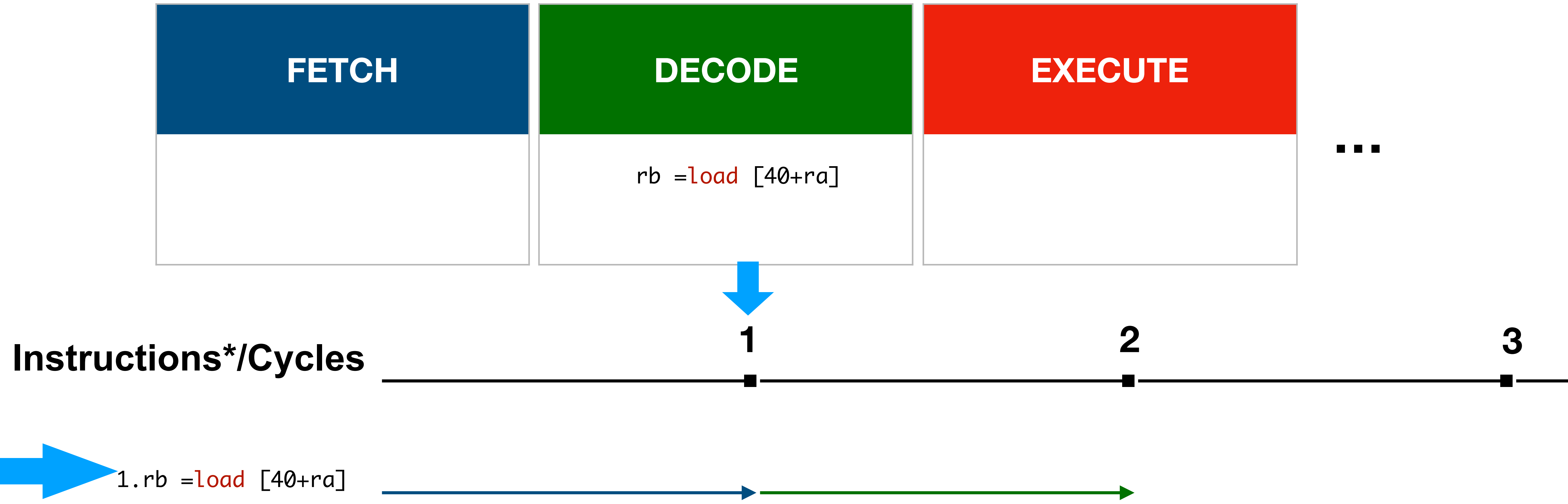
2

3

`1.rb =load [40+ra]`

Processing instructions

Stages

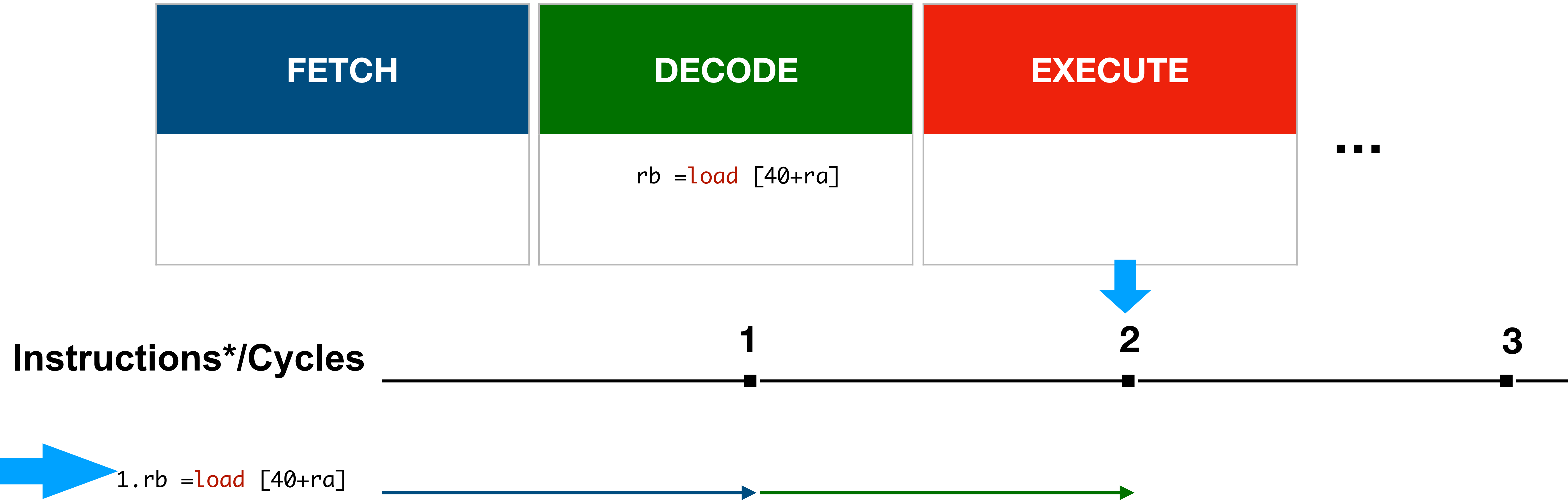


`rb =load [40+ra]`

`1.rb =load [40+ra]`

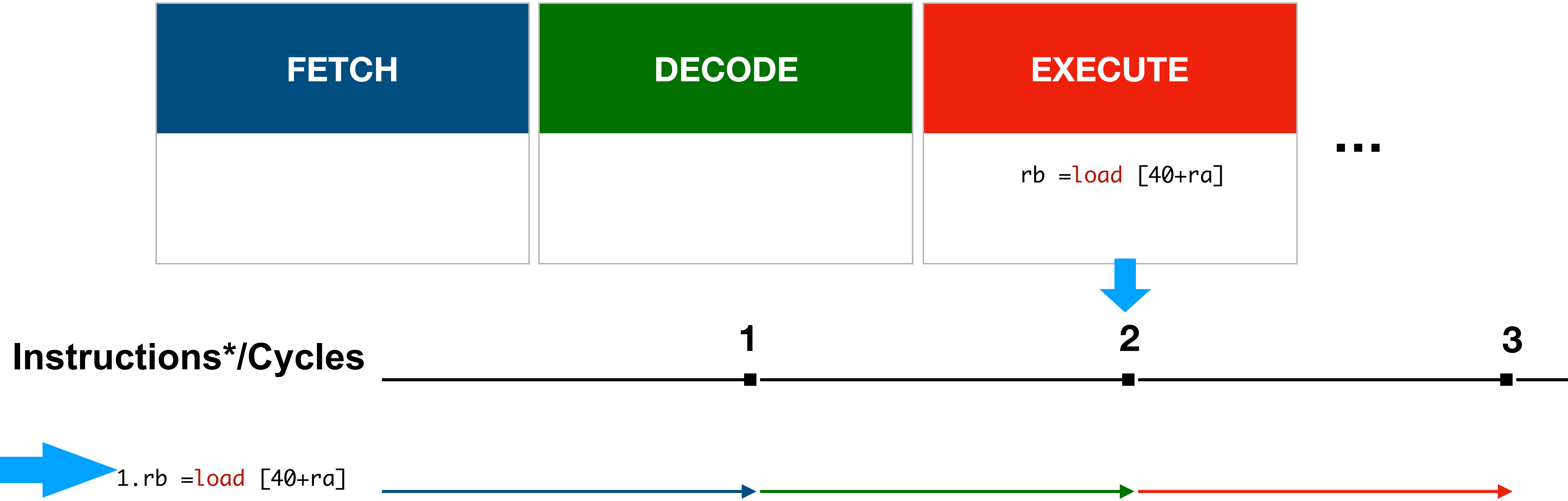
Processing instructions

Stages



Processing instructions

Stages



FETCH

DECODE

EXECUTE

```
rb =load [40+ra]
```

...

Instructions*/Cycles

1

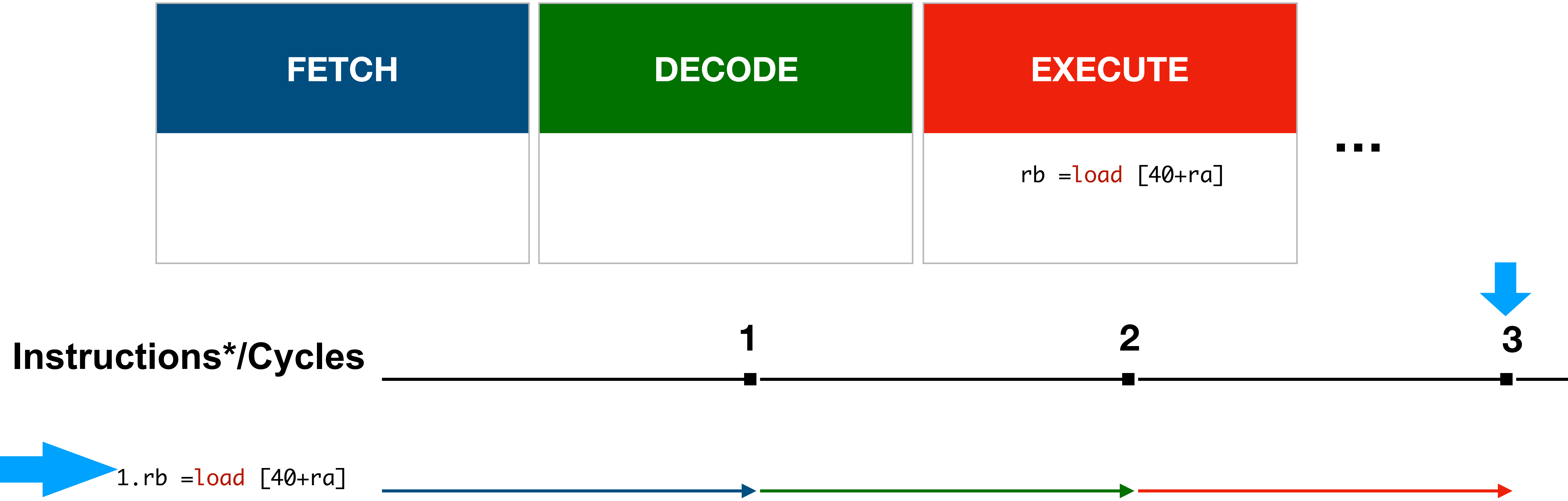
2

3

```
1.rb =load [40+ra]
```

Processing instructions

Stages



FETCH

DECODE

EXECUTE

```
rb =load [40+ra]
```

...

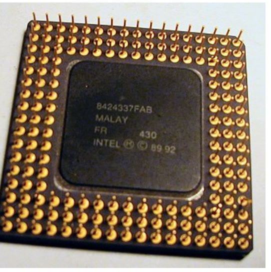
Instructions*/Cycles

1

2

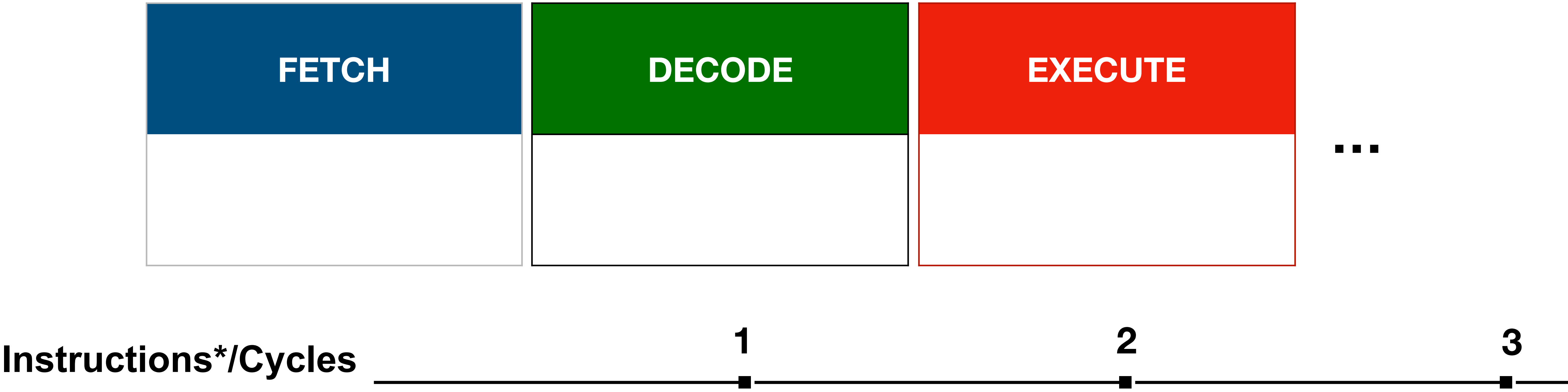
3

```
1.rb =load [40+ra]
```



Processing instructions

Pipelining stages



FETCH

DECODE

EXECUTE

...

Instructions*/Cycles

1

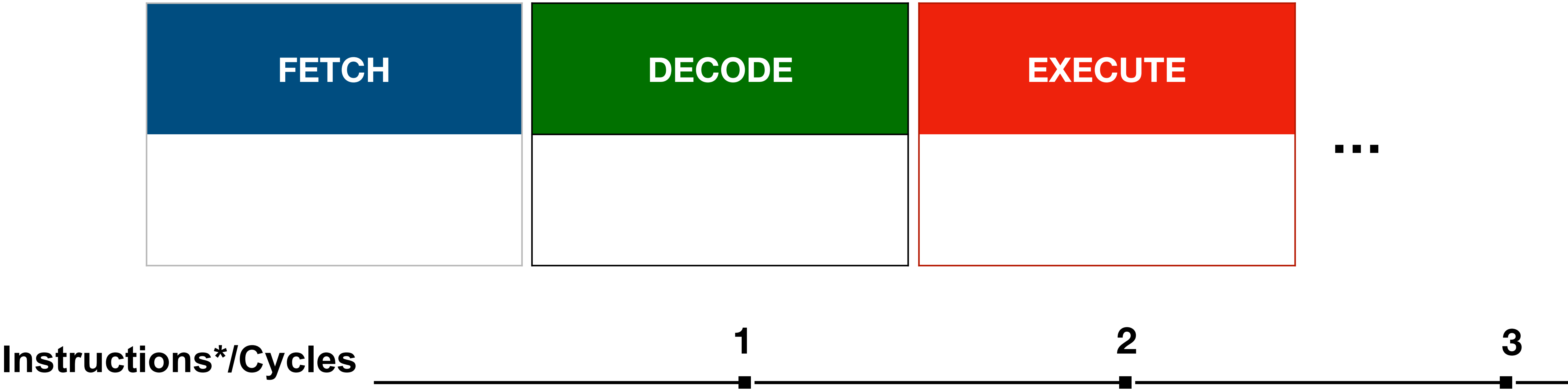
2

3

```
1.rb =load [40+ra]

2.rc= load [44+rb]

3.rd= load [41+rc]
```

Processing instructions

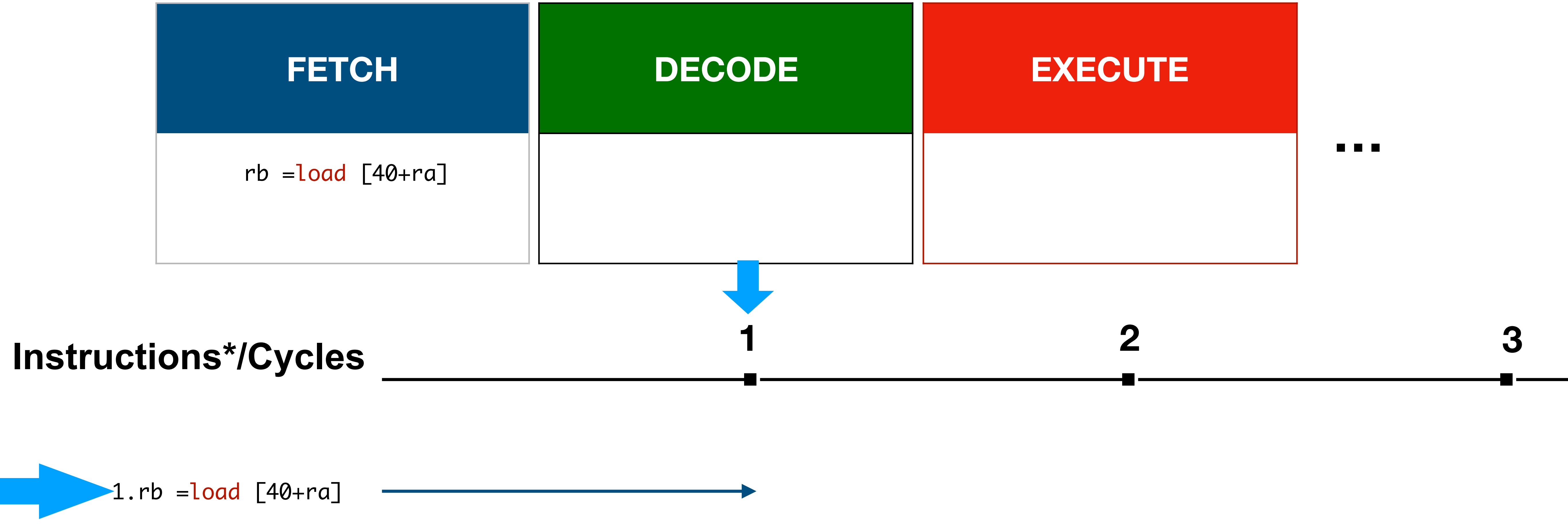
Pipelining stages



```
1.rb =load [40+ra]

2.rc= load [44+rb]

3.rd= load [41+rc]
```

Processing instructions

Pipelining stages



FETCH

DECODE

EXECUTE

rb =load [40+ra]

...

Instructions*/Cycles

1

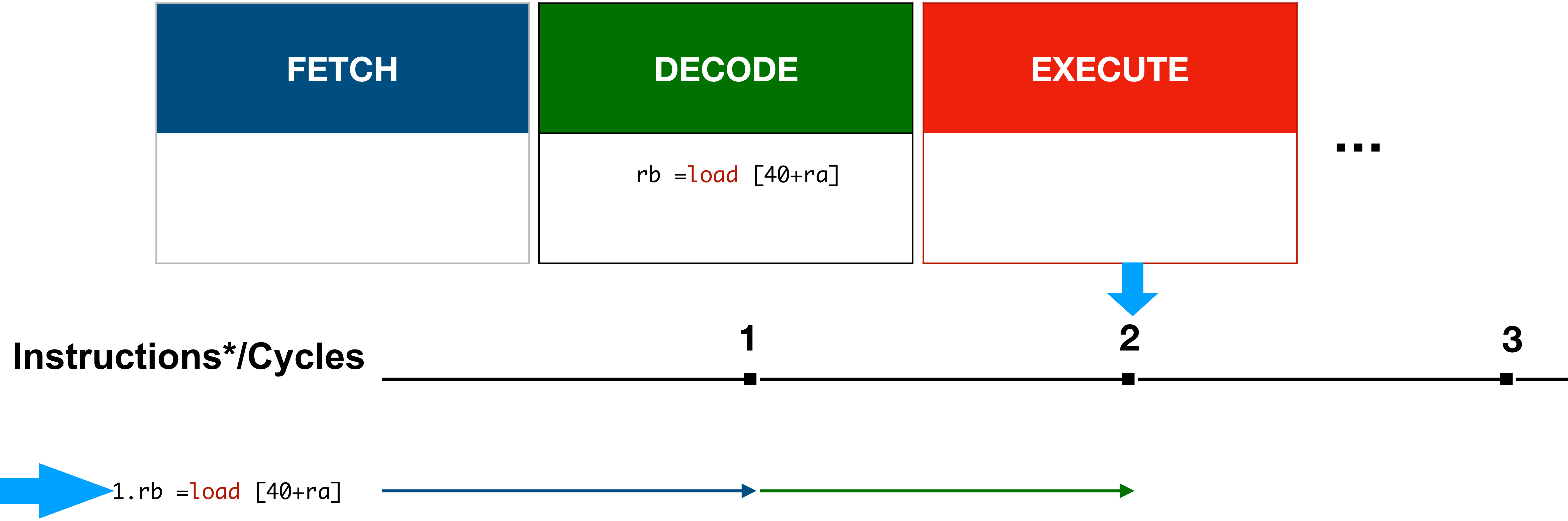
2

3

```
1.rb =load [40+ra]

2.rc= load [44+rb]

3.rd= load [41+rc]
```

Processing instructions

Pipelining stages



FETCH

DECODE

EXECUTE

```
rb =load [40+ra]
```

...

Instructions*/Cycles

1

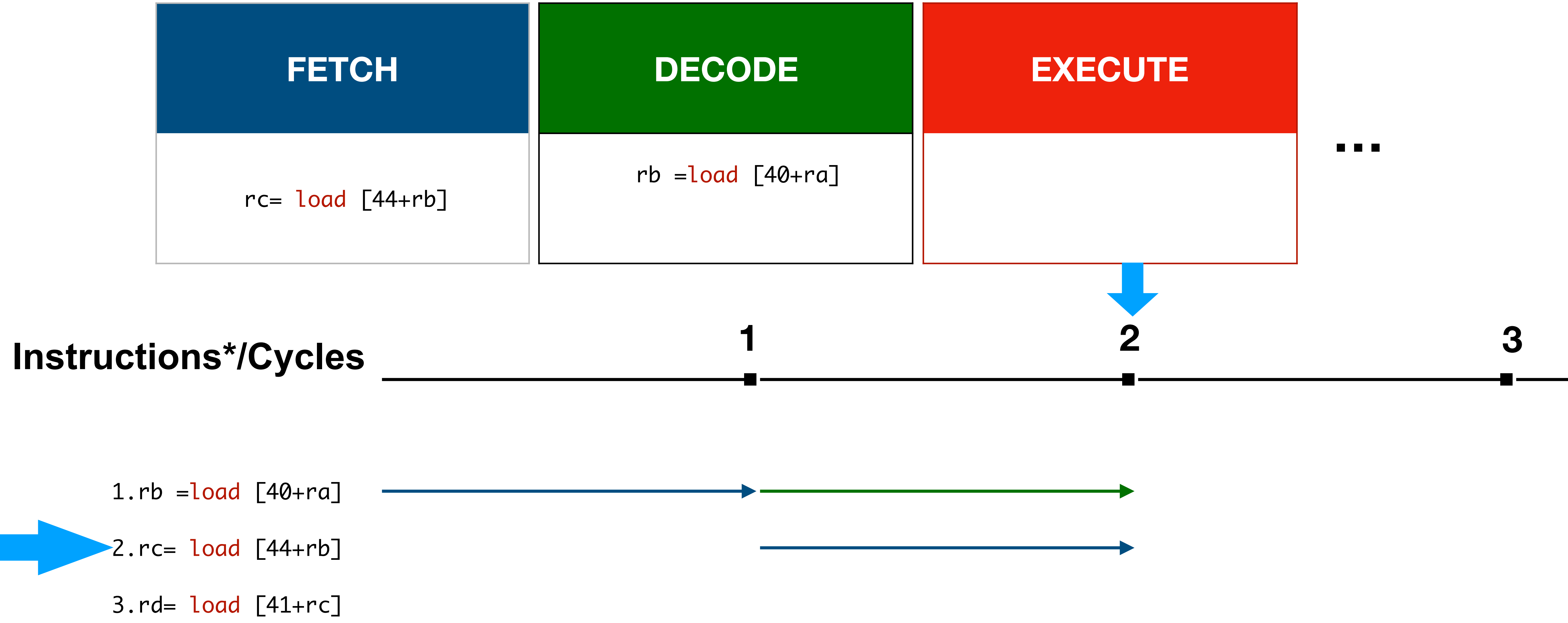
2

3

```
1.rb =load [40+ra]

2.rc= load [44+rb]

3.rd= load [41+rc]
```

Processing instructions

Pipelining stages



FETCH

rc= load [44+rb]

DECODE

rb =load [40+ra]

EXECUTE

...

Instructions*/Cycles

1

2

3

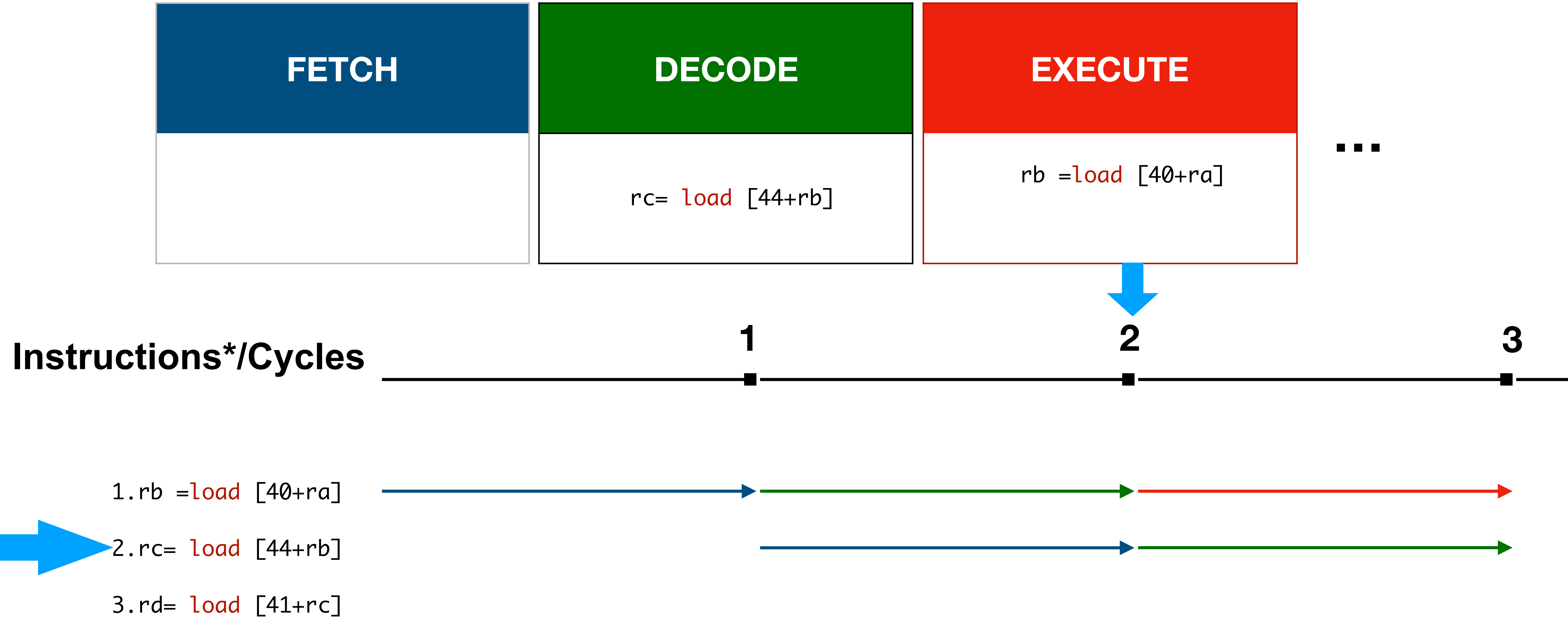
1.rb =load [40+ra]

2.rc= load [44+rb]

3.rd= load [41+rc]

Processing instructions

Pipelining stages



FETCH

DECODE

EXECUTE

...

rc= load [44+rb]

rb =load [40+ra]

Instructions*/Cycles

1

2

3

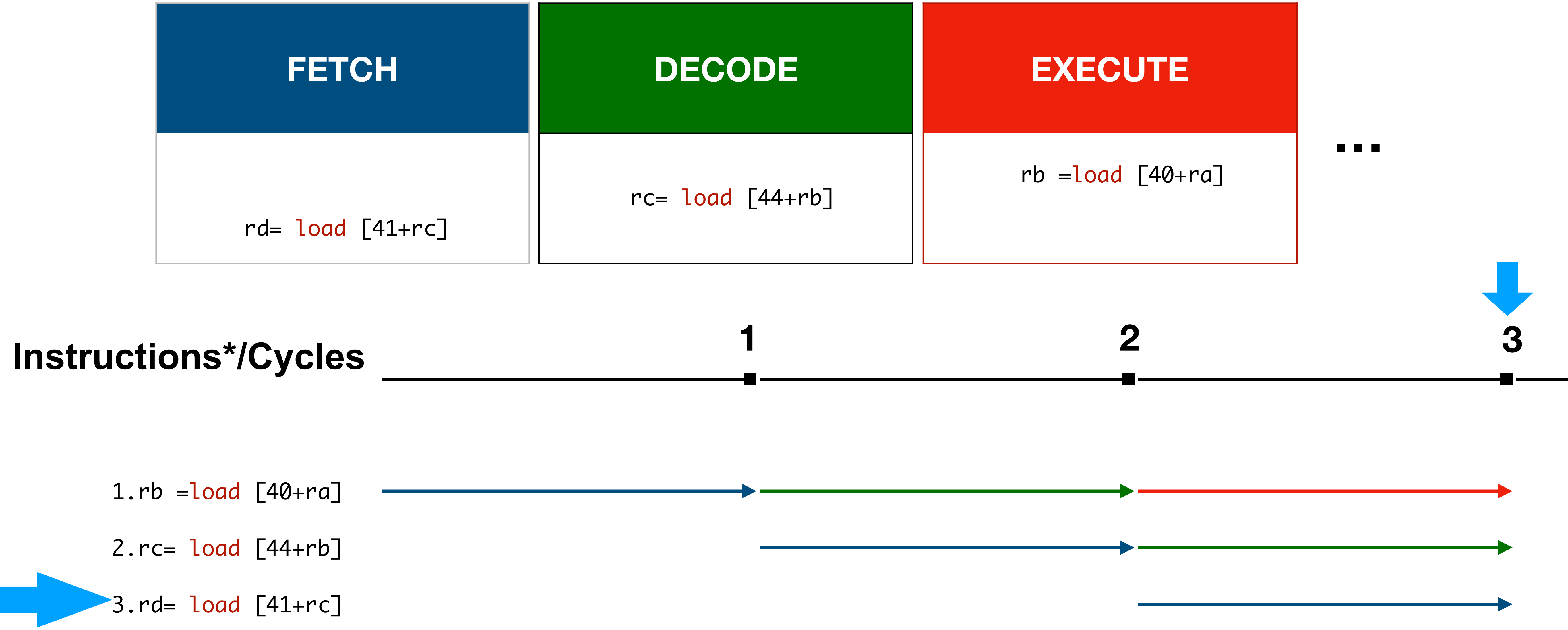
1.rb =load [40+ra]

2.rc= load [44+rb]

3.rd= load [41+rc]

Processing instructions

Pipelining stages



FETCH

rd= load [41+rc]

DECODE

rc= load [44+rb]

EXECUTE

rb =load [40+ra]

...

Instructions*/Cycles

1 2 3

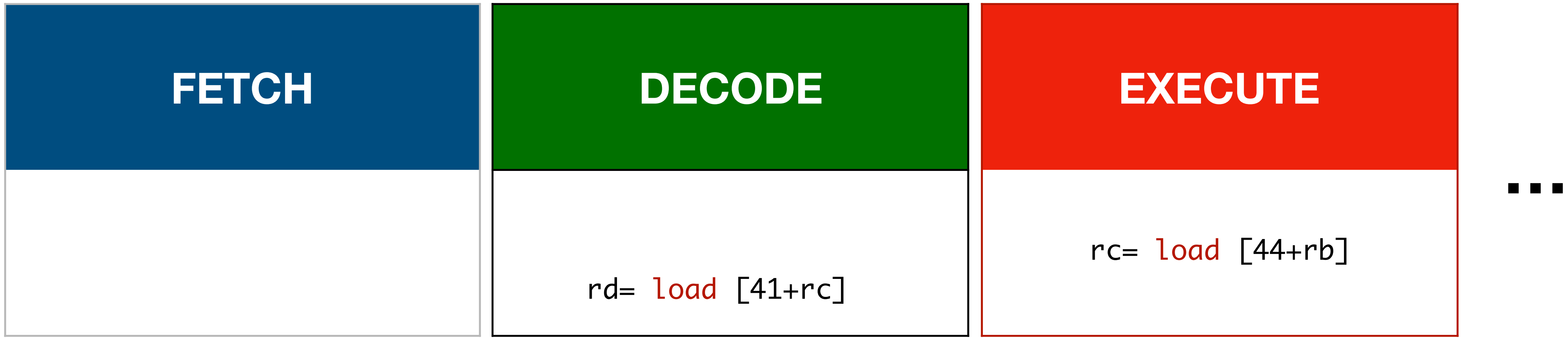
1.rb =load [40+ra]

2.rc= load [44+rb]

3.rd= load [41+rc]

Processing instructions

Pipelining stages



FETCH

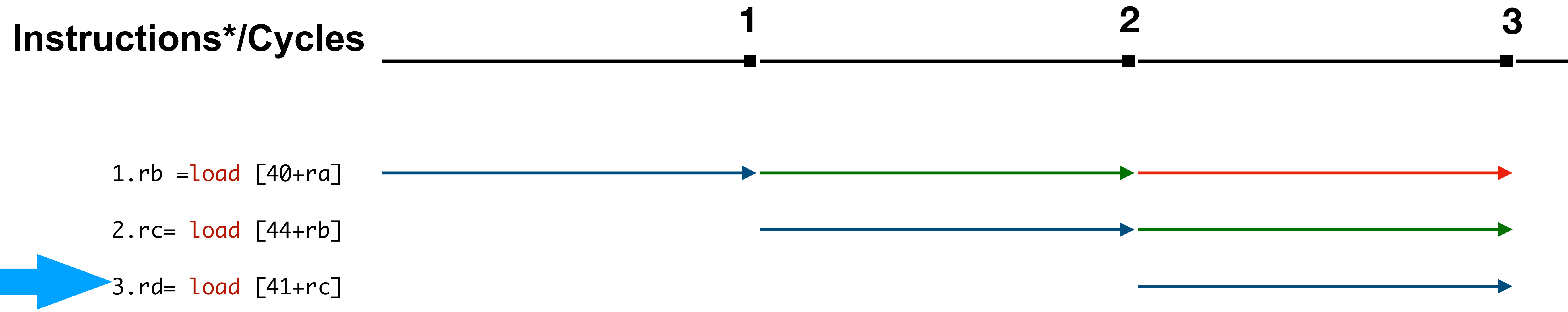
DECODE

EXECUTE

rd= load [41+rc]

rc= load [44+rb]

...



Instructions*/Cycles

1.rb =load [40+ra]

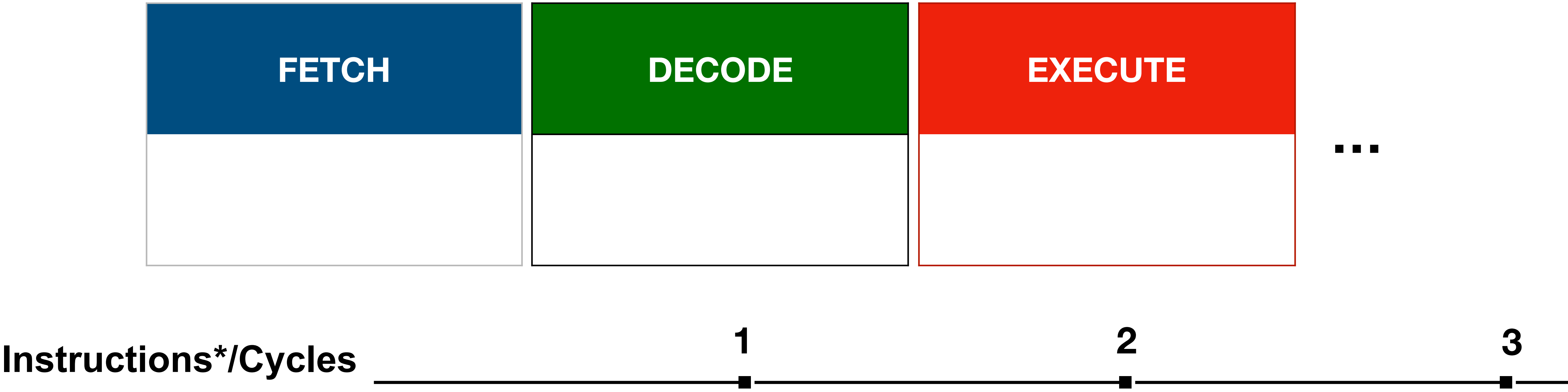
2.rc= load [44+rb]

3.rd= load [41+rc]

What happens with the pipeline when there is a branch instruction?

Processing instructions

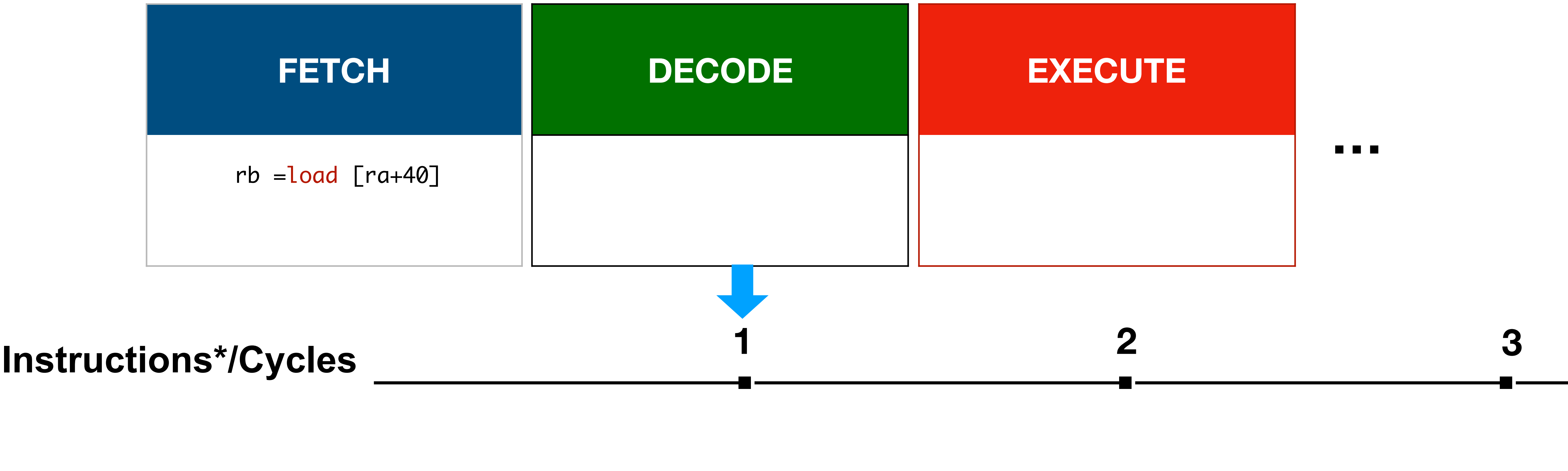
Pipelining stall on branch



```
1.rb =load [ra+40]

2.br (rb<4) 3 5

3.rd= load [rc+41]

4. …
```

Processing instructions

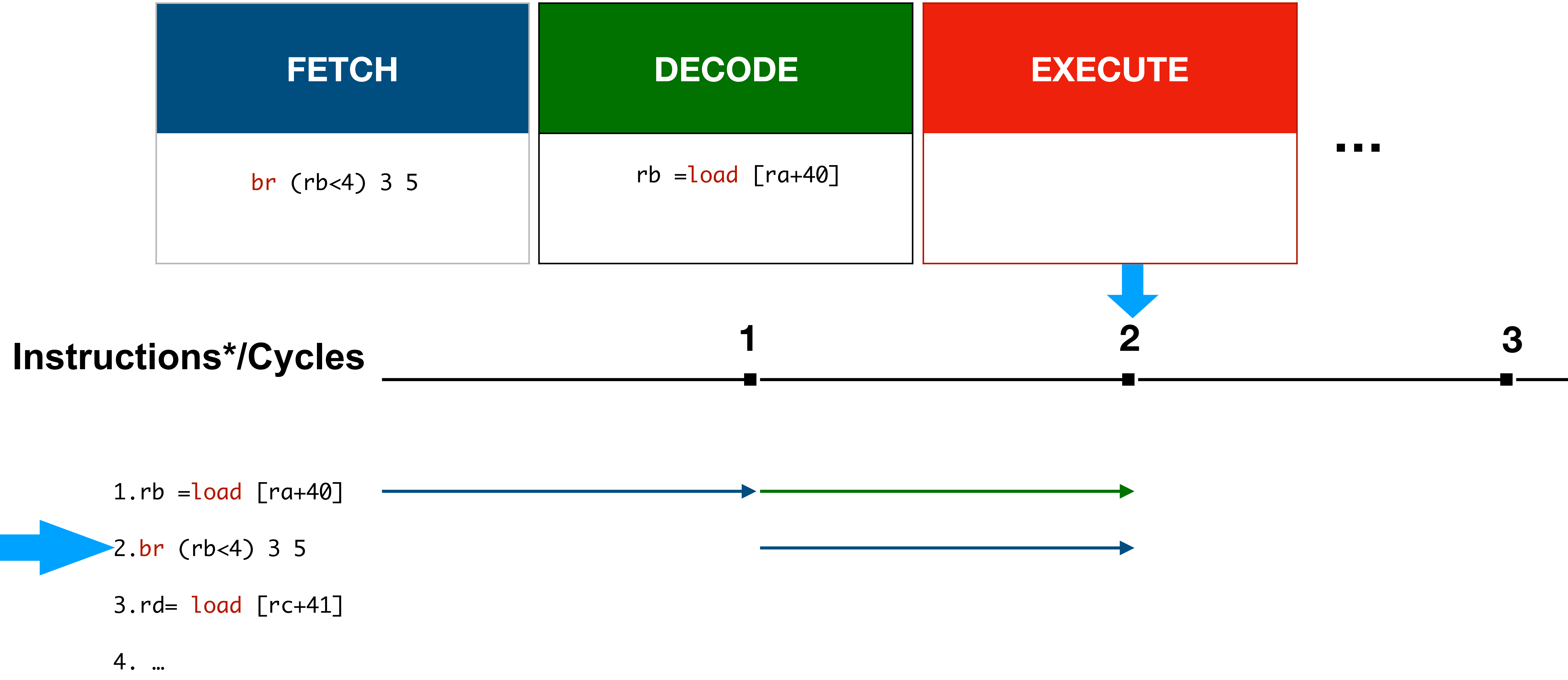
Pipelining stall on branch



```
1.rb =load [ra+40]

2.br (rb<4) 3 5

3.rd= load [rc+41]

4. …
```

Processing instructions

Pipelining stall on branch



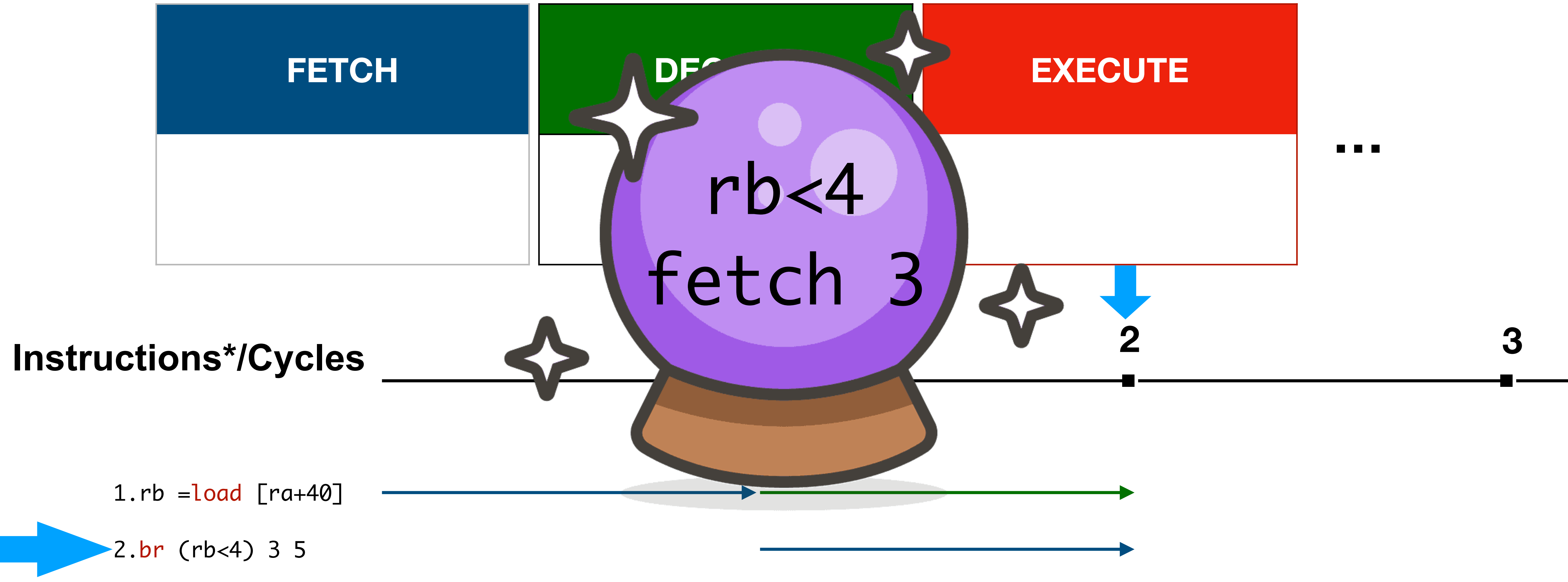
FETCH: br (rb<4) 3 5

DECODE: rb =load [ra+40]

```
1.rb =load [ra+40]
2.br (rb<4) 3 5
3.rd= load [rc+41]
4. …
```

Processing instructions

Branch predictor

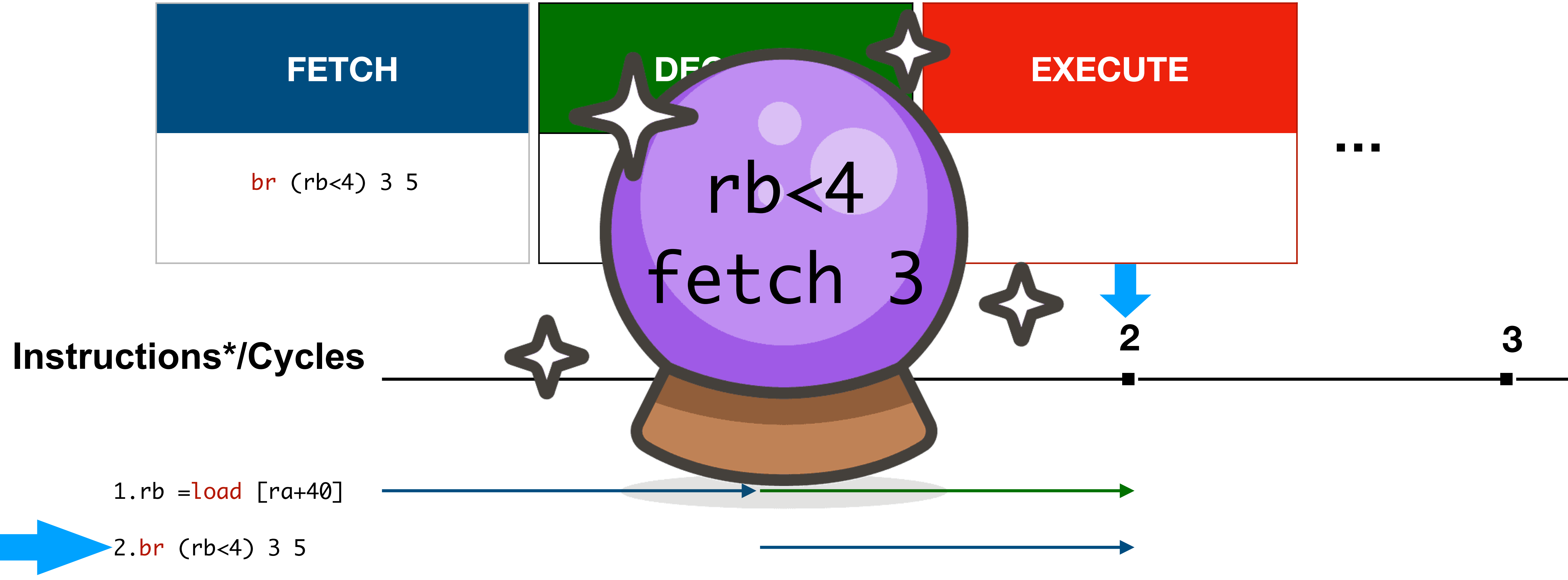


```
1.rb =load [ra+40]
2.br (rb<4) 3 5
3.rd= load [rc+41]
4. …
```

Predictor image by Vincent Le Moign CC BY4.0

Processing instructions

Branch predictor

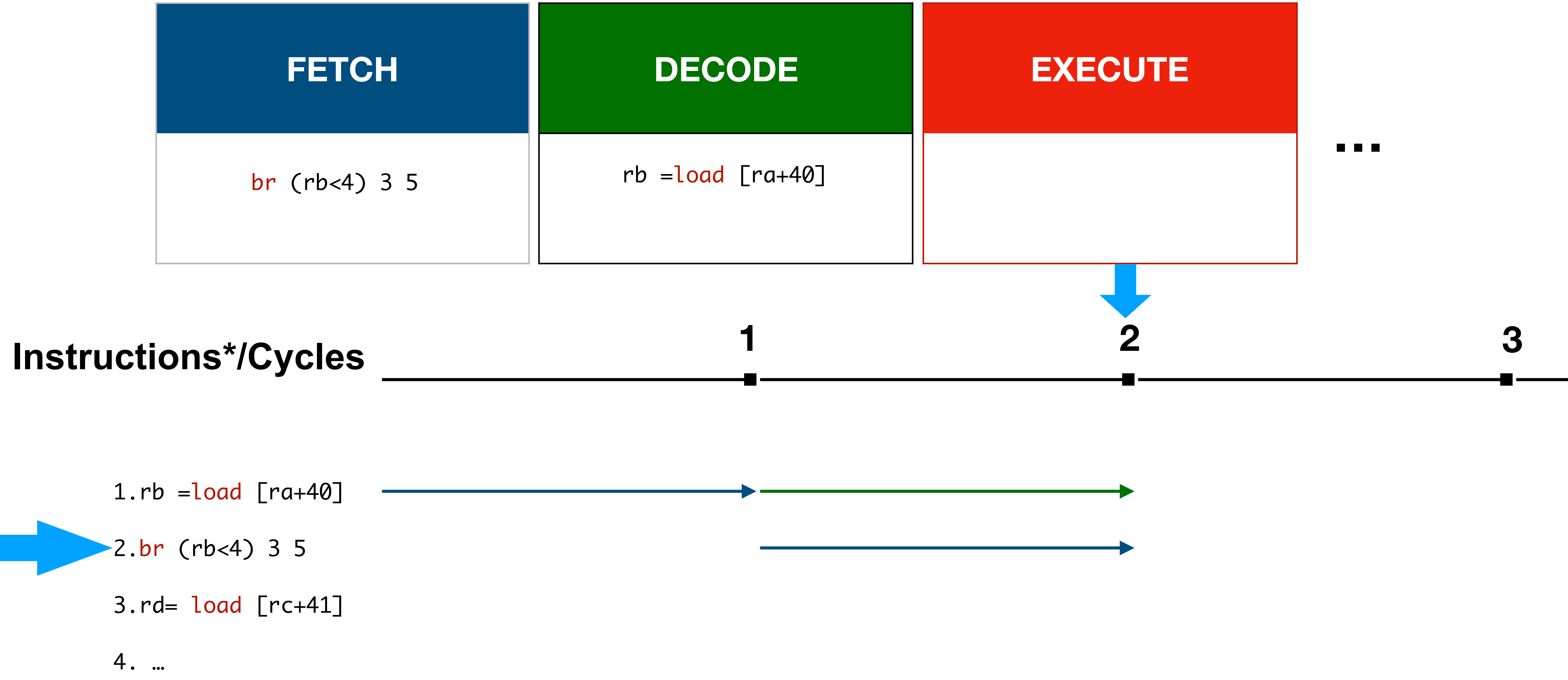


```
1.rb =load [ra+40]
2.br (rb<4) 3 5
3.rd= load [rc+41]
4. …
```

Predictor image by Vincent Le Moign CC BY4.0

Processing instructions

Branch predictor and speculative execution



FETCH

DECODE

EXECUTE

…

br (rb<4) 3 5

rb =load [ra+40]

Instructions*/Cycles

1

2

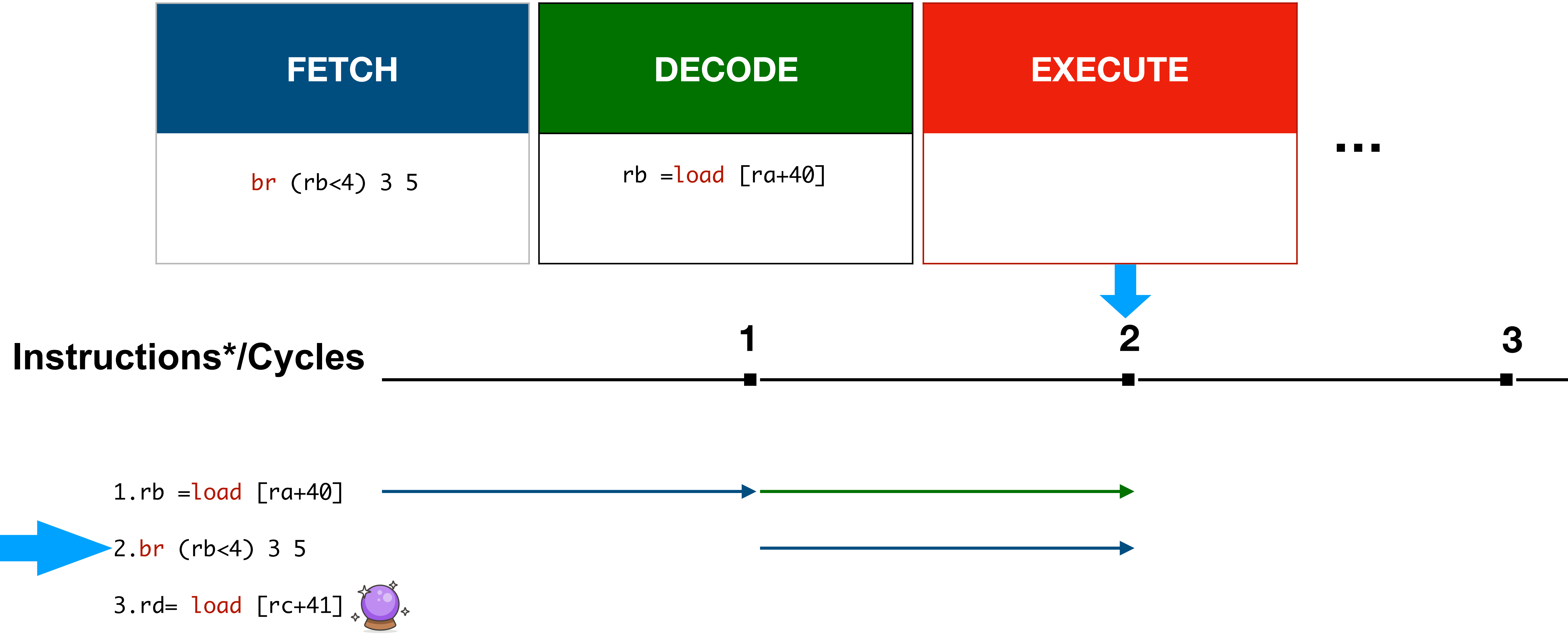
3

```
1.rb =load [ra+40]
2.br (rb<4) 3 5
3.rd= load [rc+41]
4. …
```

Predictor image by Vincent Le Moign CC BY4.0

Processing instructions

Branch predictor and speculative execution



FETCH

br (rb<4) 3 5

DECODE

rb =load [ra+40]

EXECUTE

...

Instructions*/Cycles

1

2

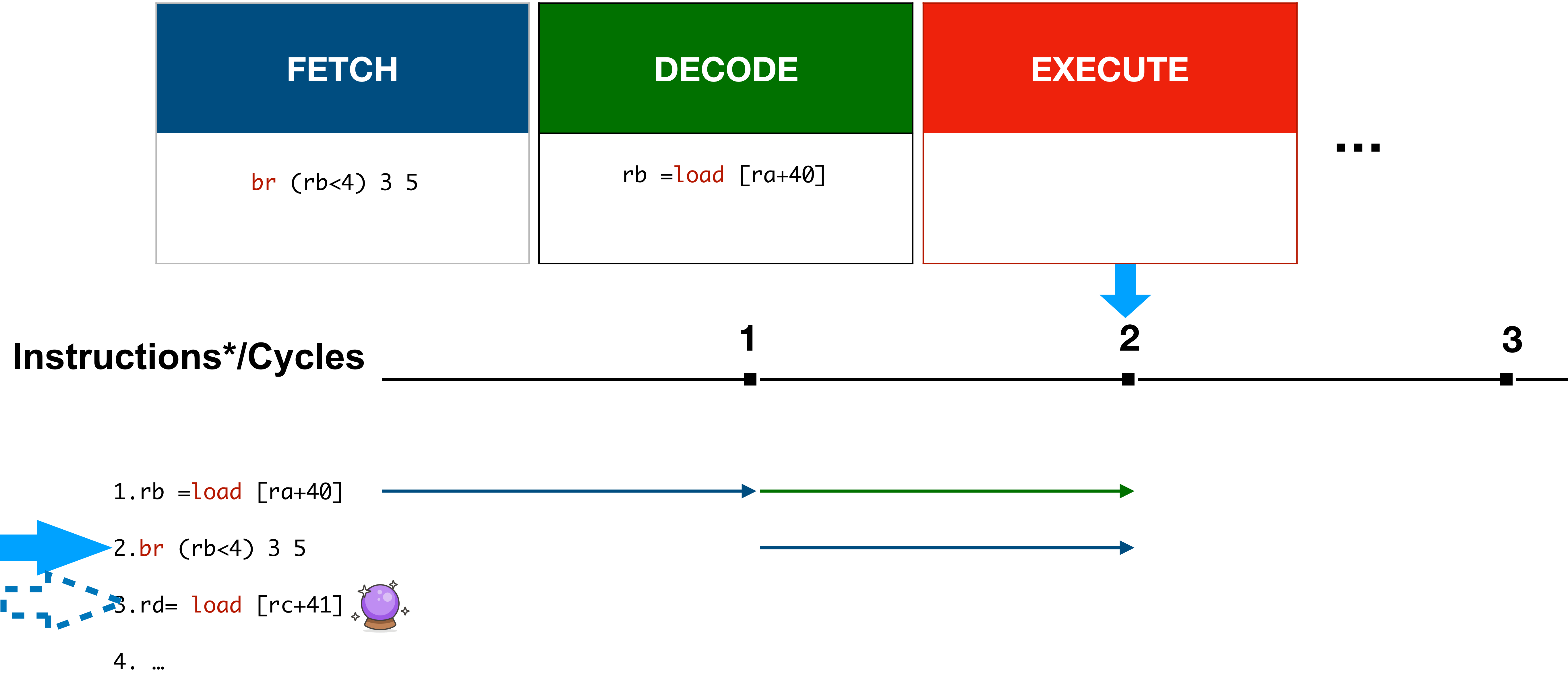
3

```
1.rb =load [ra+40]
2.br (rb<4) 3 5
3.rd= load [rc+41]
4. …
```

Predictor image by Vincent Le Moign CC BY4.0

Processing instructions

Branch predictor and speculative execution

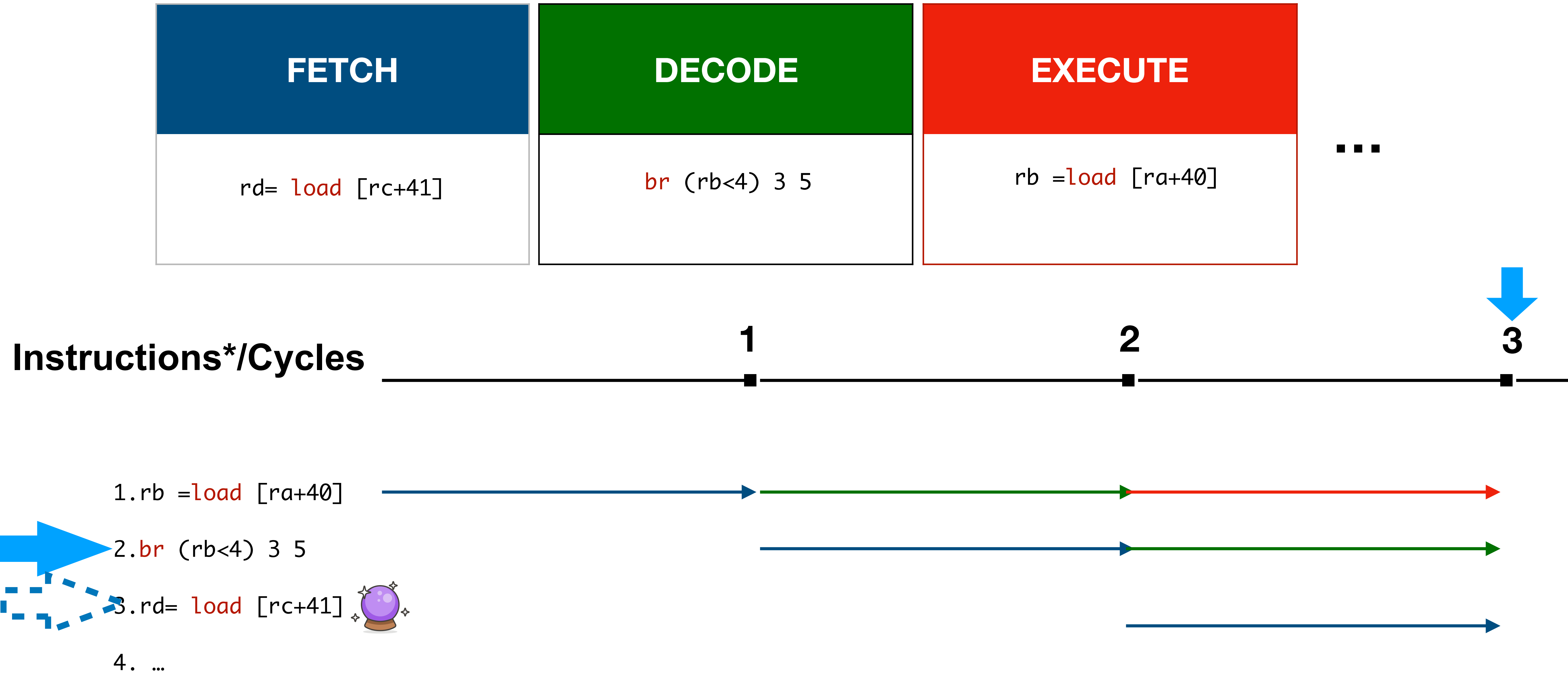


```
1.rb =load [ra+40]
2.br (rb<4) 3 5
3.rd= load [rc+41]
4. …
```

Predictor image by Vincent Le Moign CC BY4.0

Processing instructions

Branch predictor and speculative execution



FETCH

rd= load [rc+41]

DECODE

br (rb<4) 3 5

EXECUTE

rb =load [ra+40]

...

Instructions*/Cycles

1

2

3

1.rb =load [ra+40]

2.br (rb<4) 3 5

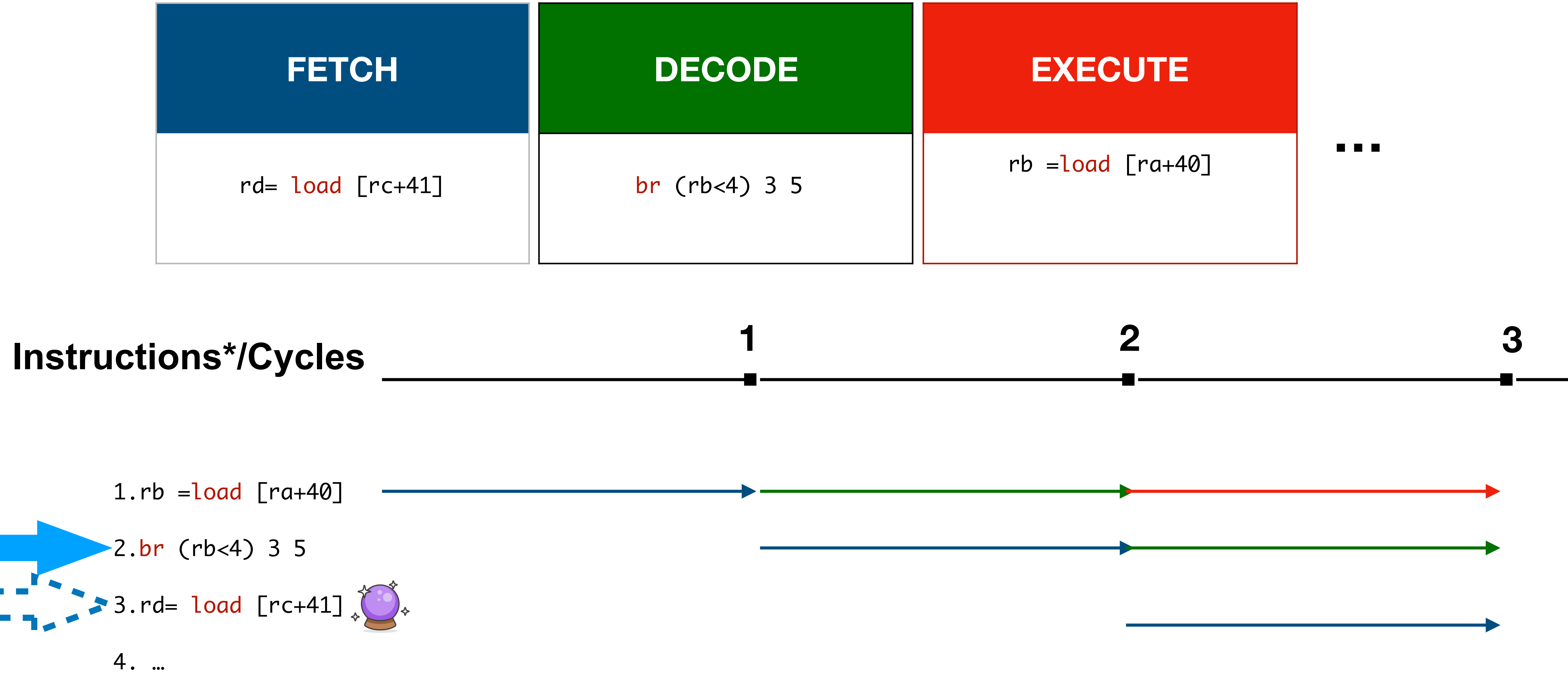
3.rd= load [rc+41]

4. …

Predictor image by Vincent Le Moign CC BY4.0

Processing instructions

Transient speculative execution and rollback



FETCH

rd= load [rc+41]

DECODE

br (rb<4) 3 5

EXECUTE

rb =load [ra+40]

…

Instructions*/Cycles

1 2 3

1.rb =load [ra+40]

2.br (rb<4) 3 5

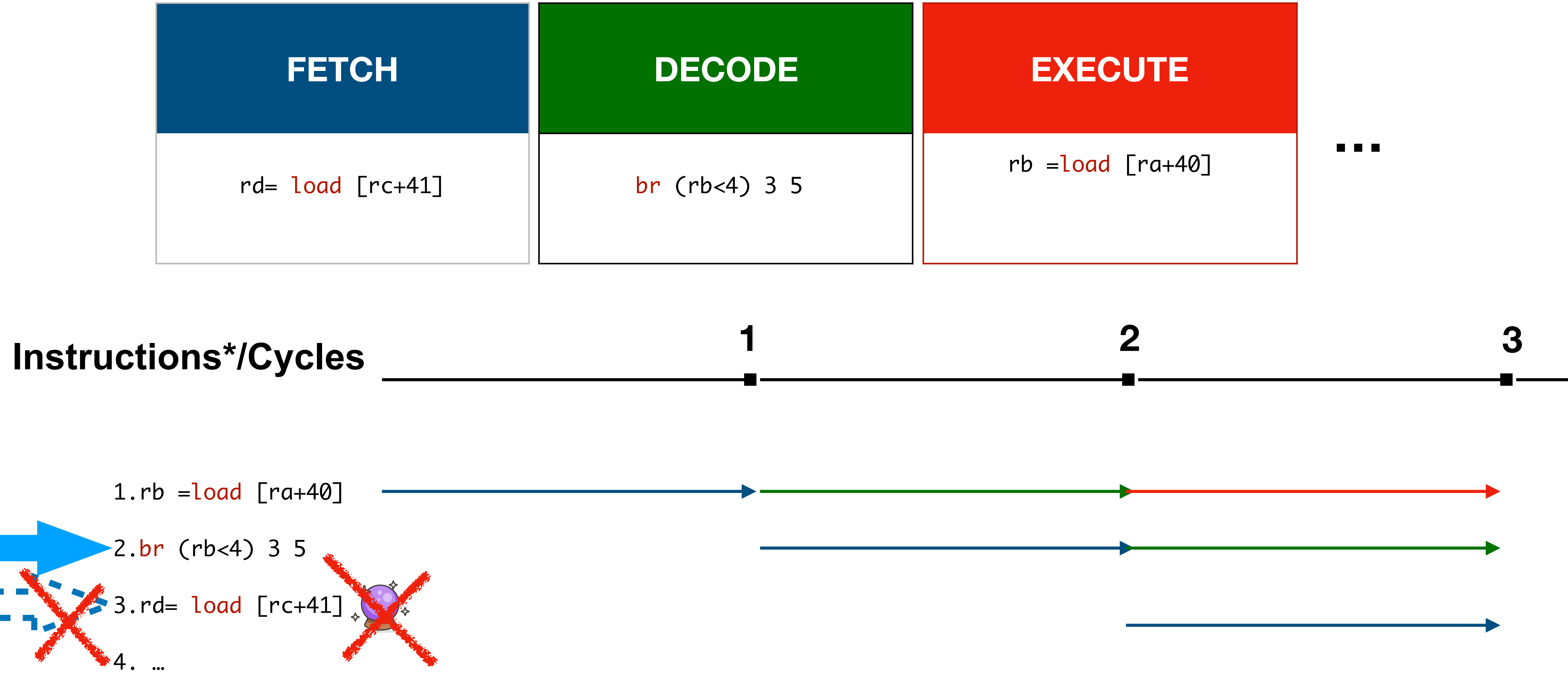
3.rd= load [rc+41]

4. …

Predictor image by Vincent Le Moign CC BY4.0

Processing instructions

Transient speculative execution and rollback



FETCH

rd= load [rc+41]

DECODE

br (rb<4) 3 5

EXECUTE

rb =load [ra+40]

...

Instructions*/Cycles

1

2

3

1.rb =load [ra+40]

2.br (rb<4) 3 5

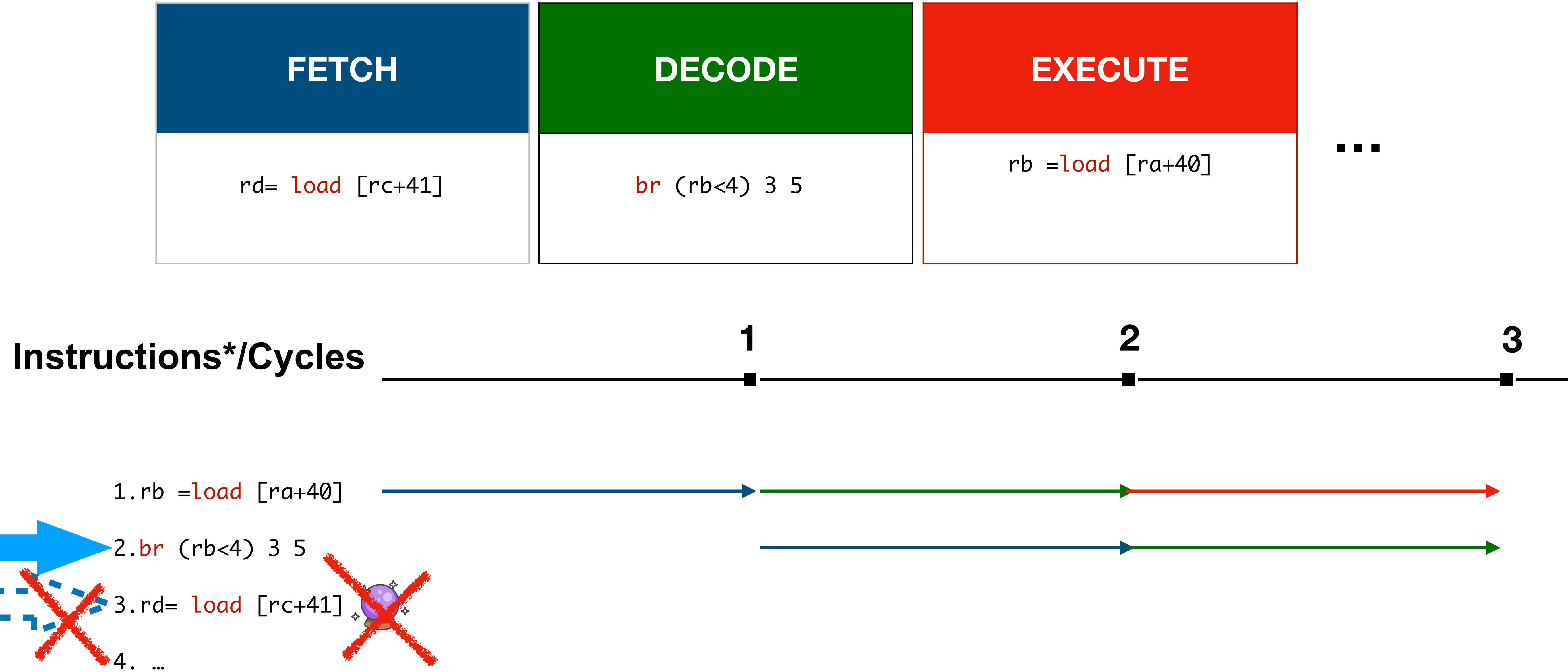
3.rd= load [rc+41]

4. …

Predictor image by Vincent Le Moign CC BY4.0

Processing instructions

Transient speculative execution and rollback



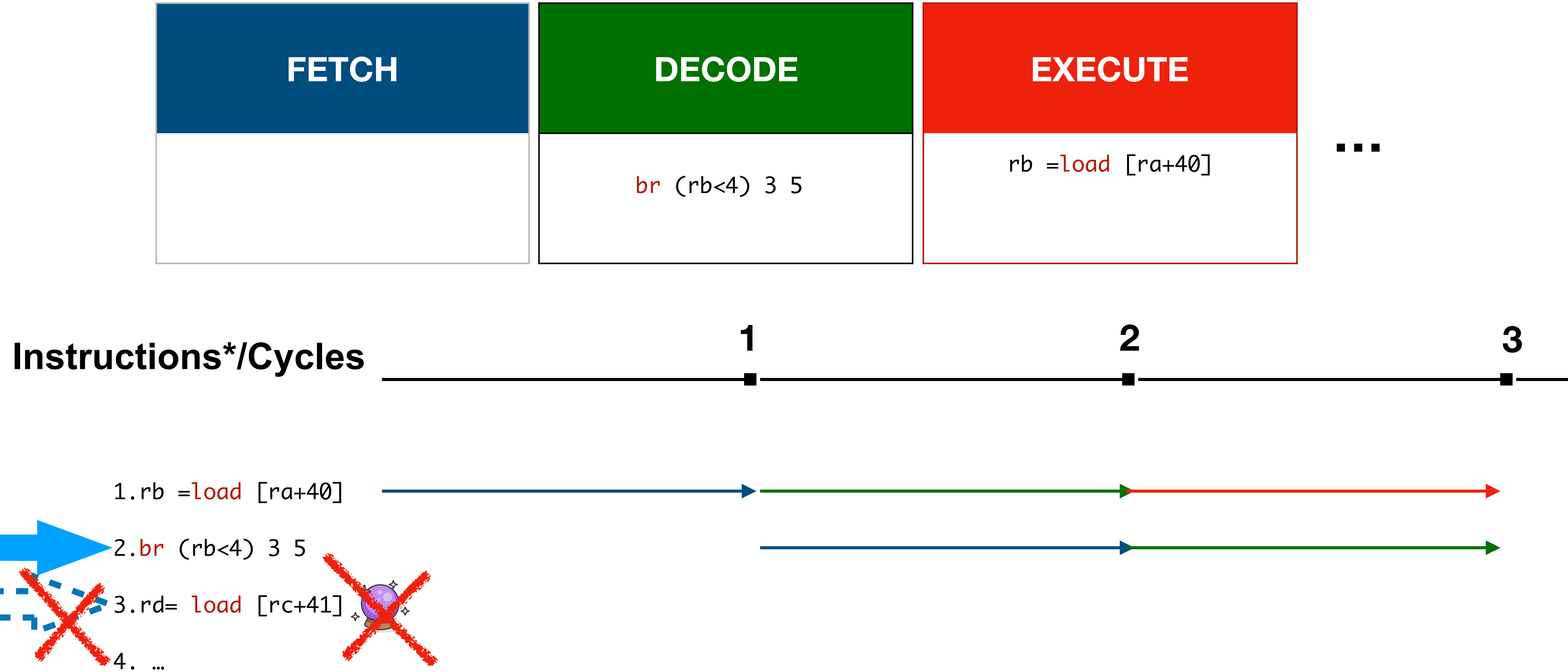
FETCH	DECODE	EXECUTE
rd= load [rc+41]	br (rb<4) 3 5	rb =load [ra+40]

```
1.rb =load [ra+40]
2.br (rb<4) 3 5
3.rd= load [rc+41]
4. …
```

Predictor image by Vincent Le Moign CC BY4.0

Processing instructions

Transient speculative execution and rollback



FETCH

DECODE

`br (rb<4) 3 5`

EXECUTE

`rb =load [ra+40]`

…

Instructions*/Cycles

1 **2** **3**

```
1.rb =load [ra+40]
2.br (rb<4) 3 5
3.rd= load [rc+41]
4. …
```

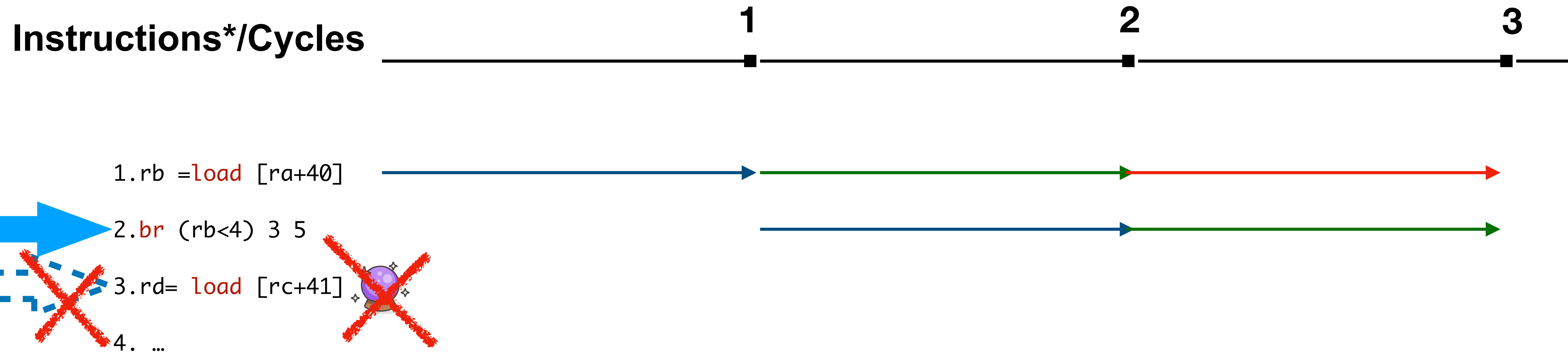
Predictor image by Vincent Le Moign CC BY4.0

Processing instructions

Transient speculative execution and rollback

FETCH | **DECODE** | **EXECUTE**

After rollback, speculative execution is reverted: all wrong computation is thrown away

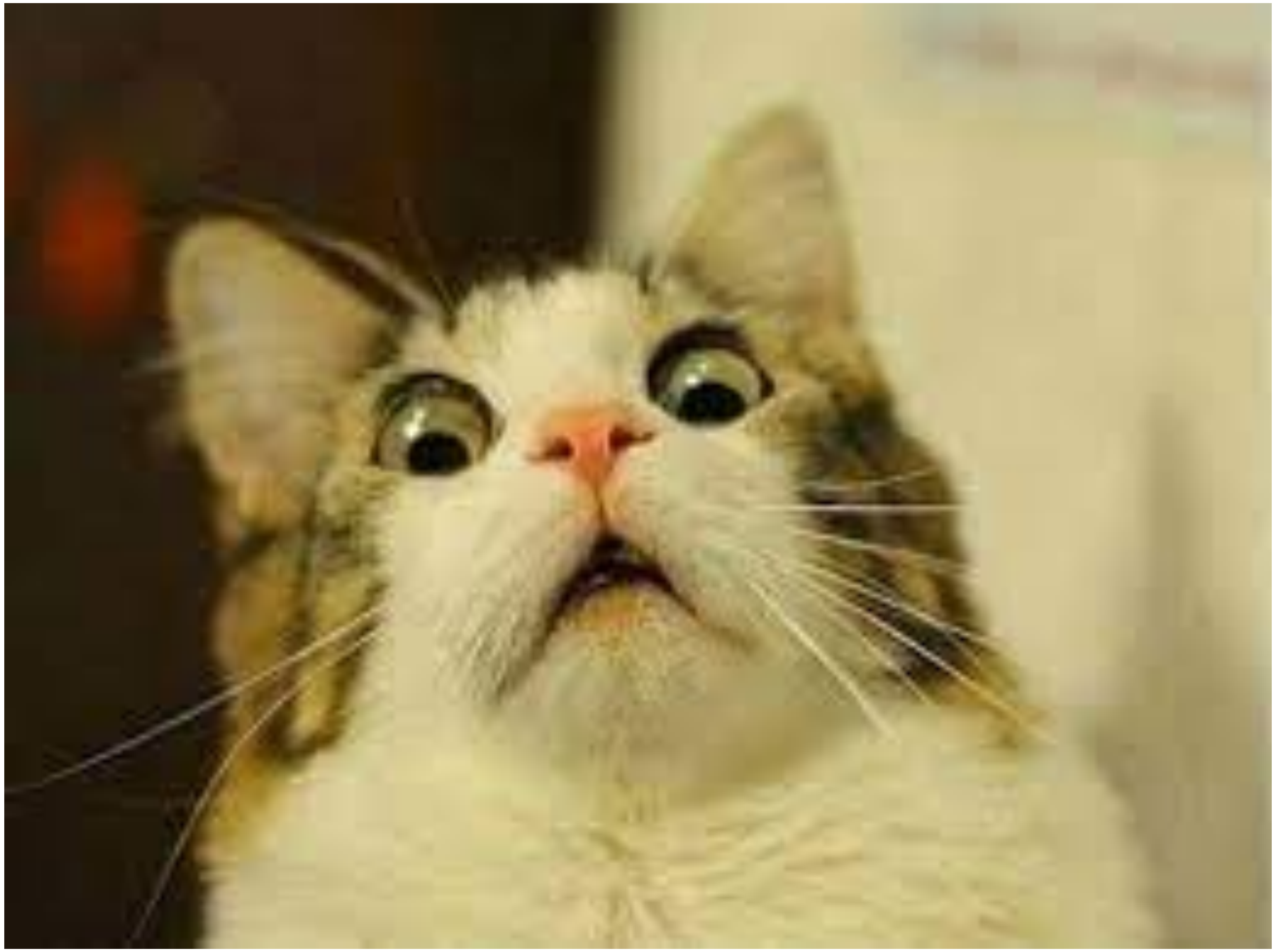


Instructions*/Cycles

```
1.rb =load [ra+40]
2.br (rb<4) 3 5
3.rd= load [rc+41]
4. …
```

Predictor image by Vincent Le Moign CC BY4.0

Small detail:
Cache state is not reset!!



Spectre v1 vulnerability

Cache

addr	data

Registers

ra = 9

Memory

Instructio

```
1.br (ra<4) 2 4

2.rb= load [40+ra]

3.rc= load [44+rb]

. . . .
```

```
1.br (
2.rb=
3.rc=
4. . . .
5. . . .
```

Spectre v1 vulnerability

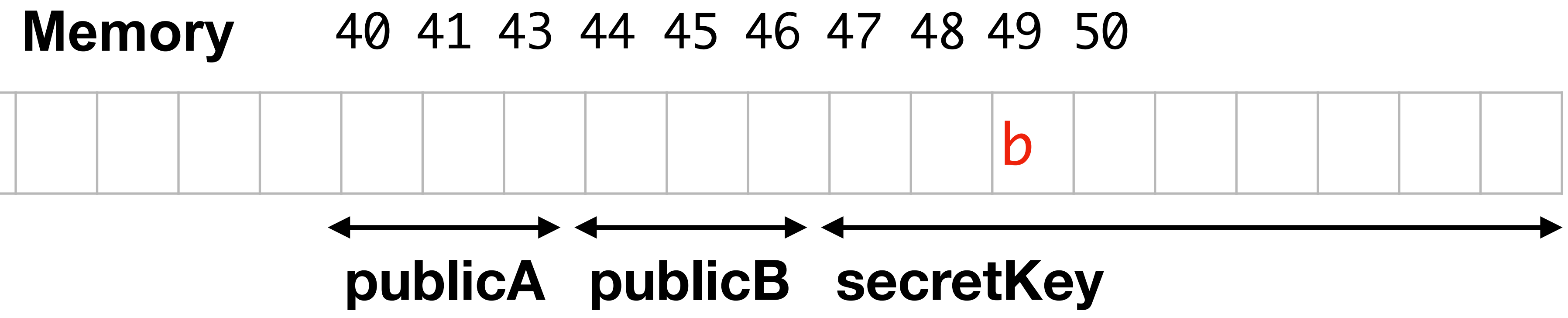
Registers

ra = 9

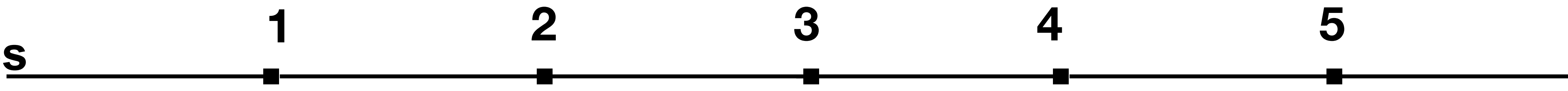
Cache

addr	data
…	
…	…
…	

Memory

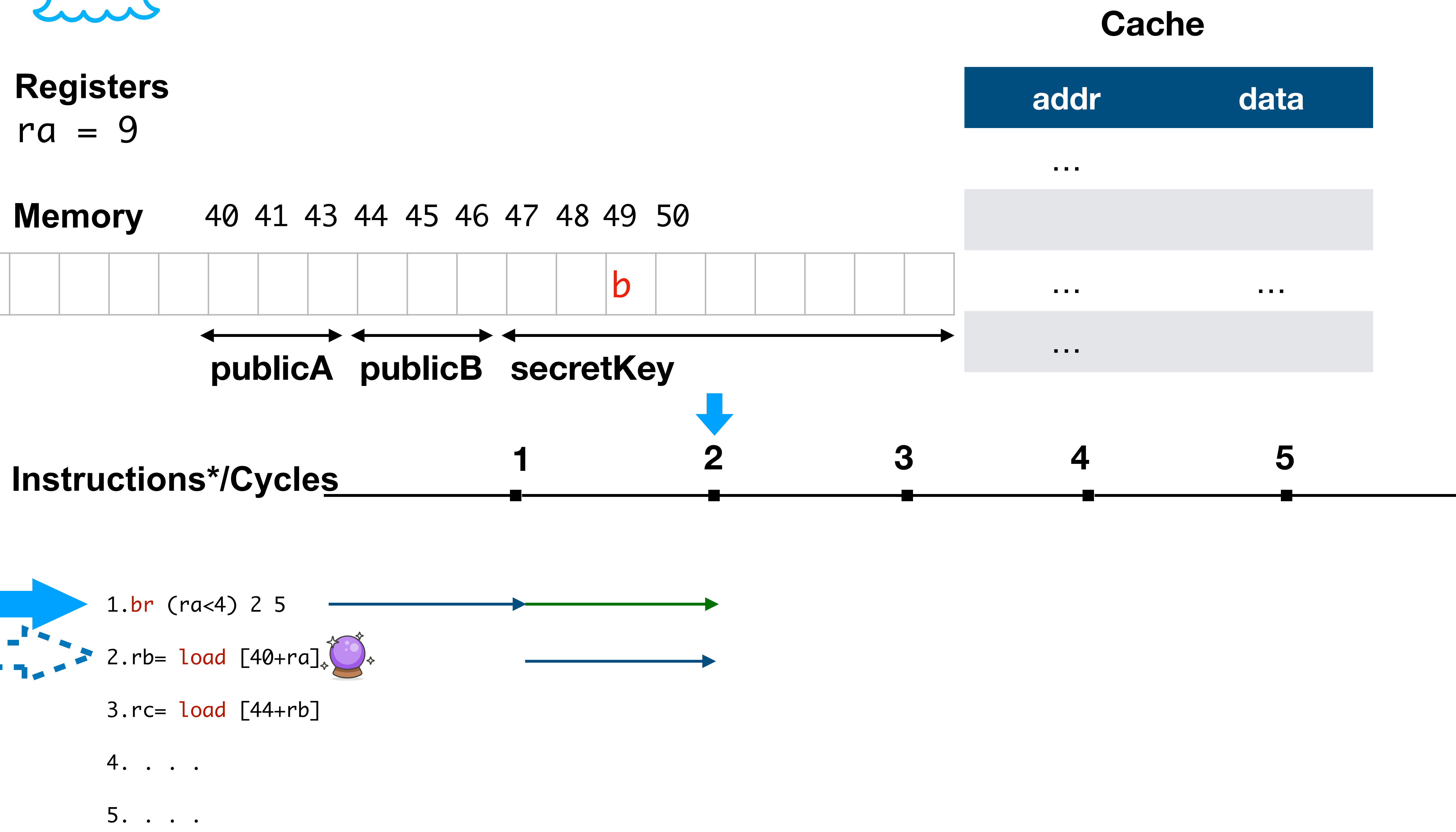


Instructions*/Cycles



```
1.br (ra<4) 2 5

2.rb= load [40+ra]

3.rc= load [44+rb]

4. . . .

5. . . .
```

Spectre v1 vulnerability



Registers

`ra = 9`

Cache

addr	data
…	
…	…
…	

Memory

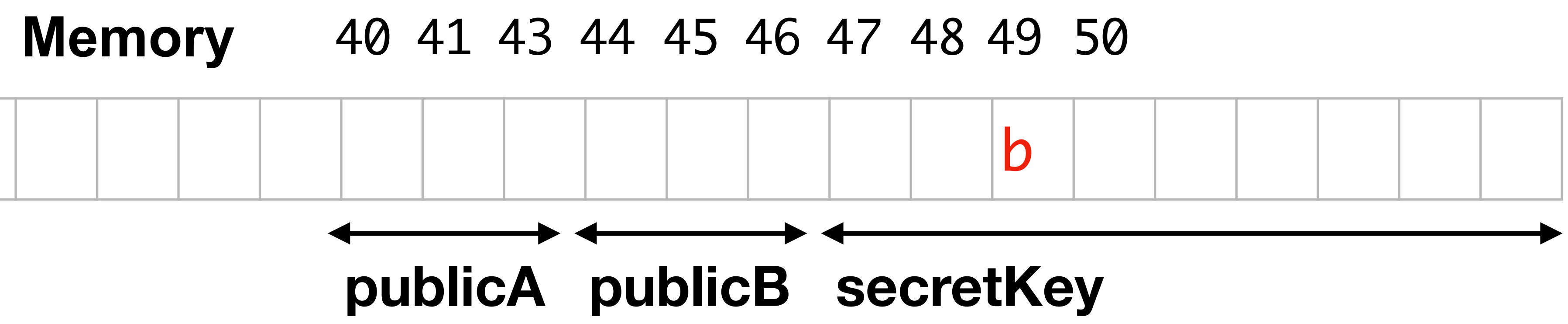
Instructions*/Cycles

```
1.br (ra<4) 2 5
2.rb= load [40+ra]
3.rc= load [44+rb]
4. . . .
5. . . .
```

Spectre v1 vulnerability

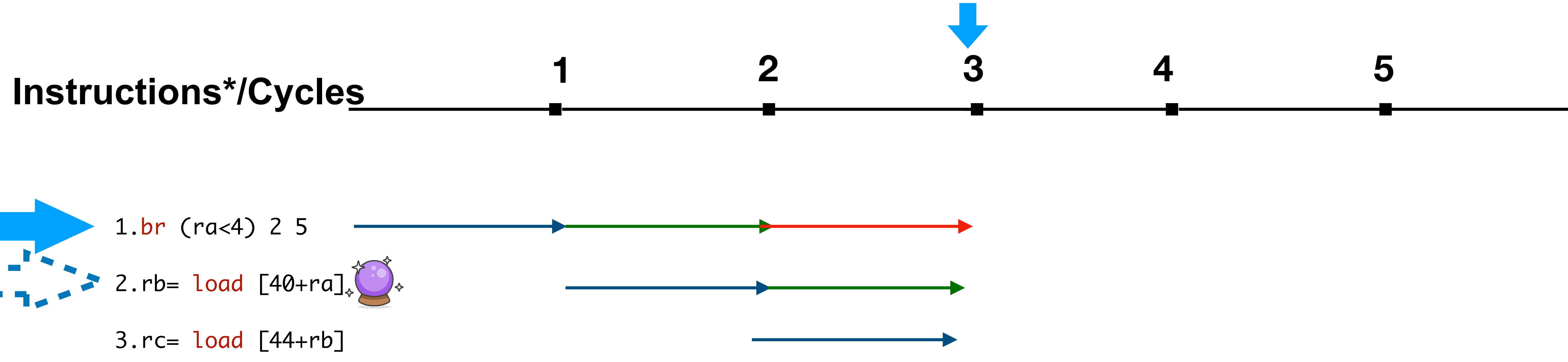
Registers

ra = 9



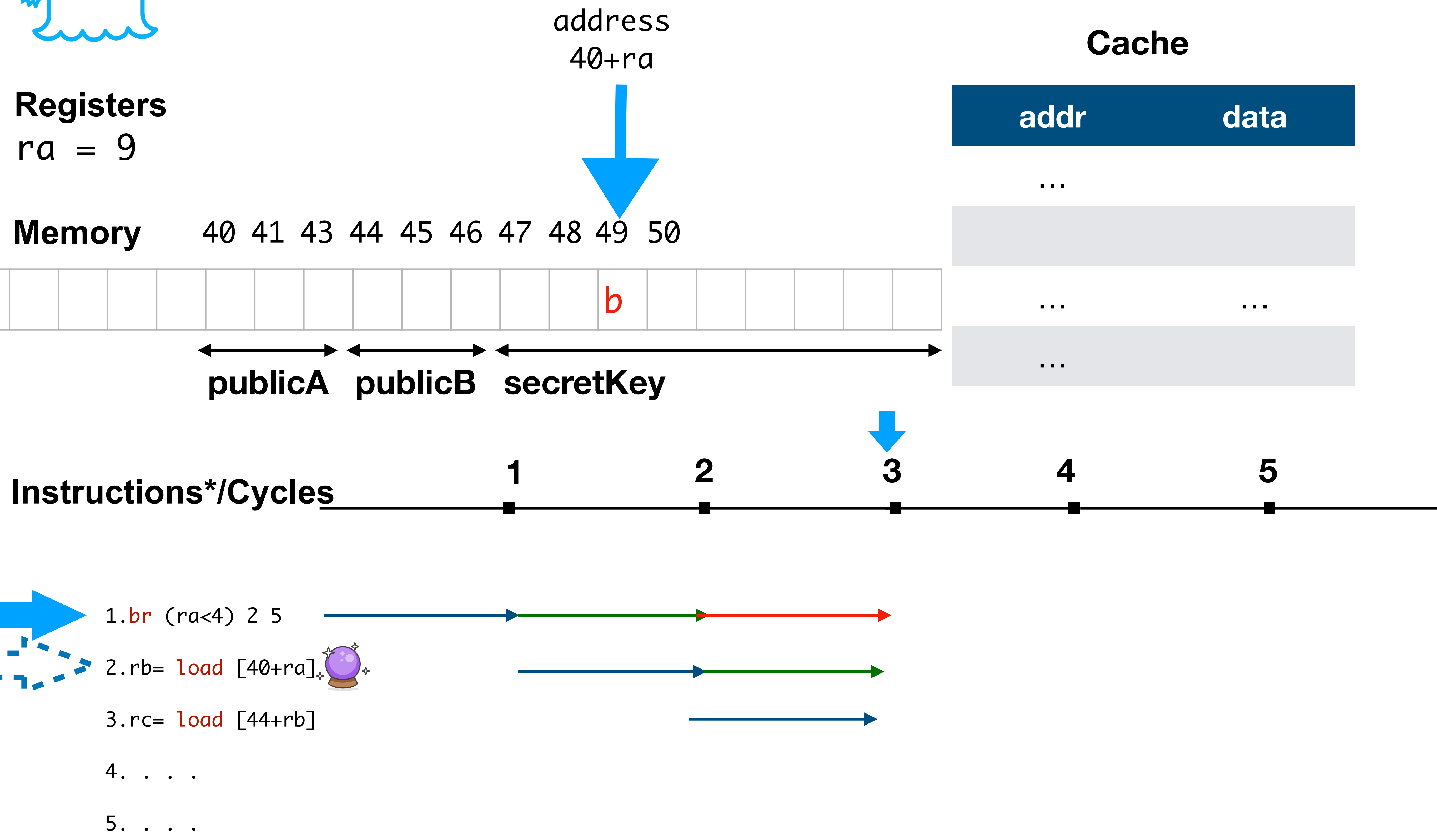
Cache

addr	data
…	
…	…
…	



```
1.br (ra<4) 2 5
2.rb= load [40+ra]
3.rc= load [44+rb]
4. . . .
5. . . .
```

Spectre v1 vulnerability



address
40+ra

Registers
ra = 9

Memory

40 41 43 44 45 46 47 48 49 50

b

publicA **publicB** **secretKey**

Cache

addr	data
…	
…	…
…	

Instructions*/Cycles 1 2 3 4 5

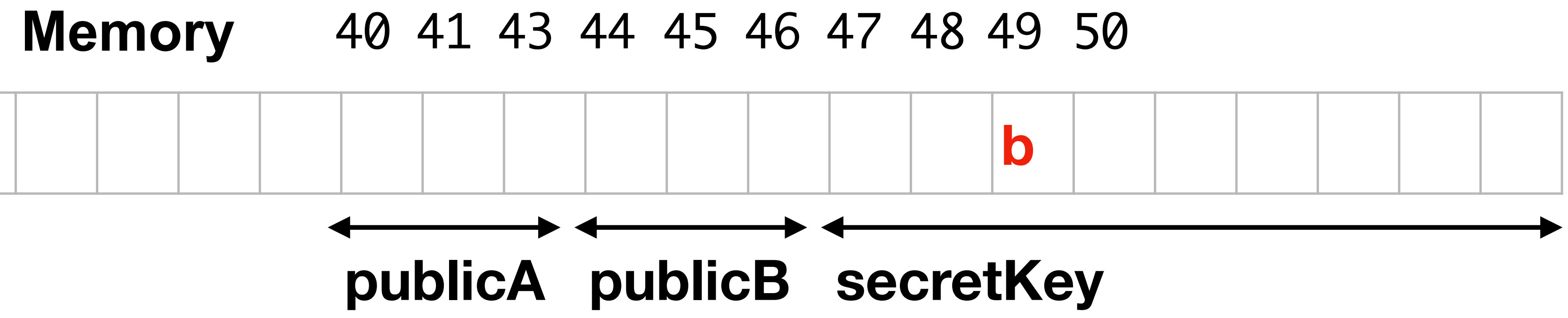
```
1.br (ra<4) 2 5
2.rb= load [40+ra]
3.rc= load [44+rb]
4. . . .
5. . . .
```

Spectre v1 vulnerability

Registers

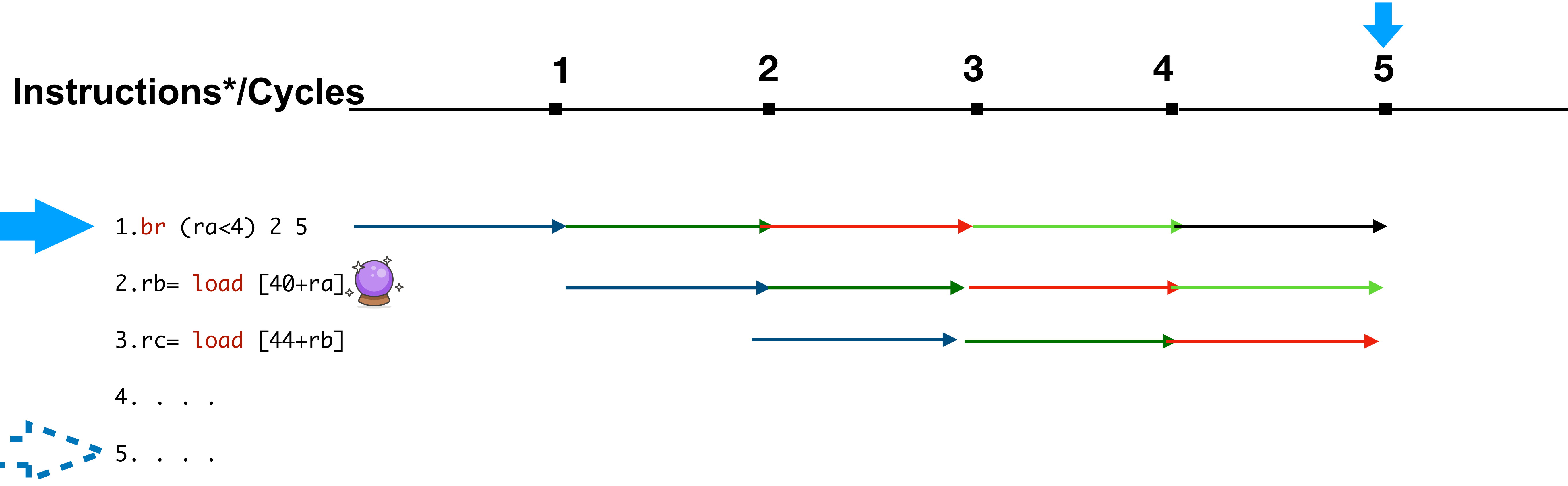
```
ra = 9
rb = b
```

Memory



Cache

addr	data
…	
40+9	b
…	…
…	

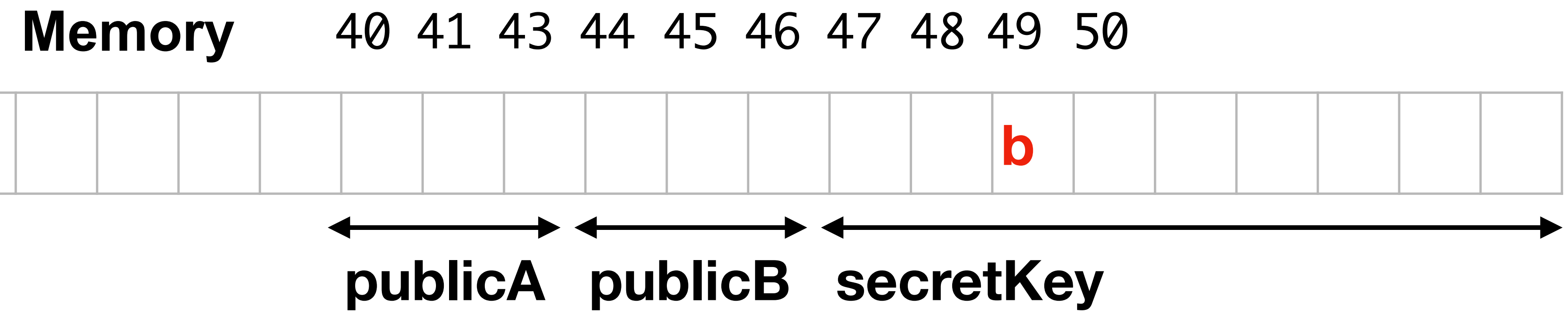


```
1.br (ra<4) 2 5
2.rb= load [40+ra]
3.rc= load [44+rb]
4. . . .
5. . . .
```

Spectre v1 vulnerability

Registers

```
ra = 9
rb = b
```



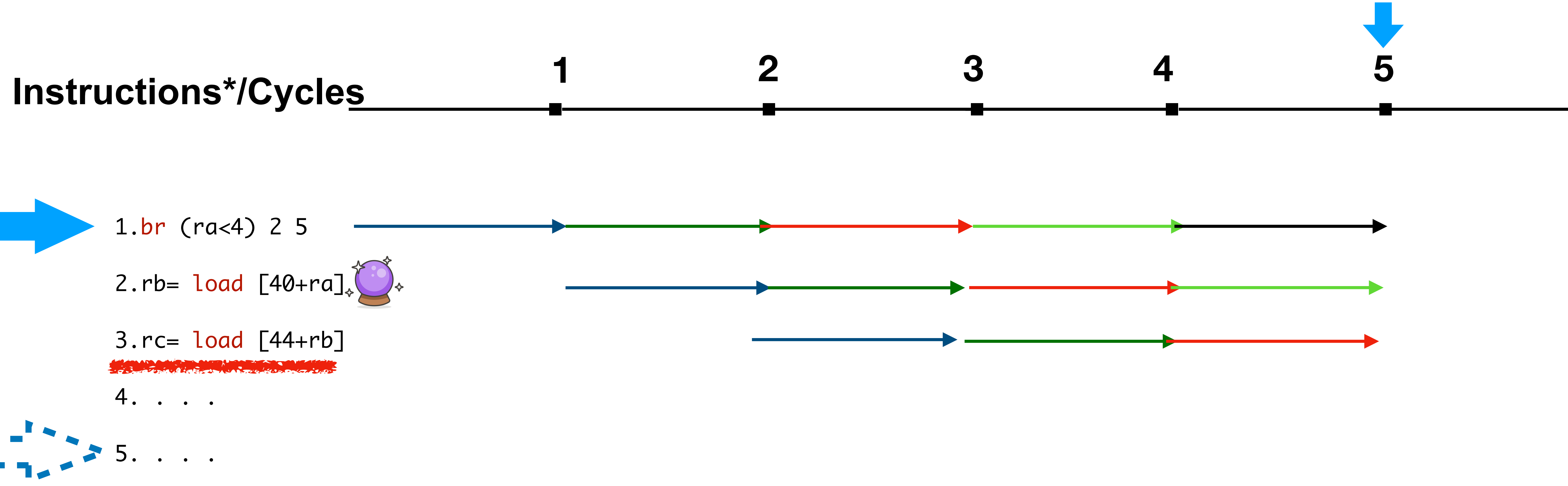
Memory 40 41 43 44 45 46 47 48 49 50

b

publicA **publicB** **secretKey**

Cache

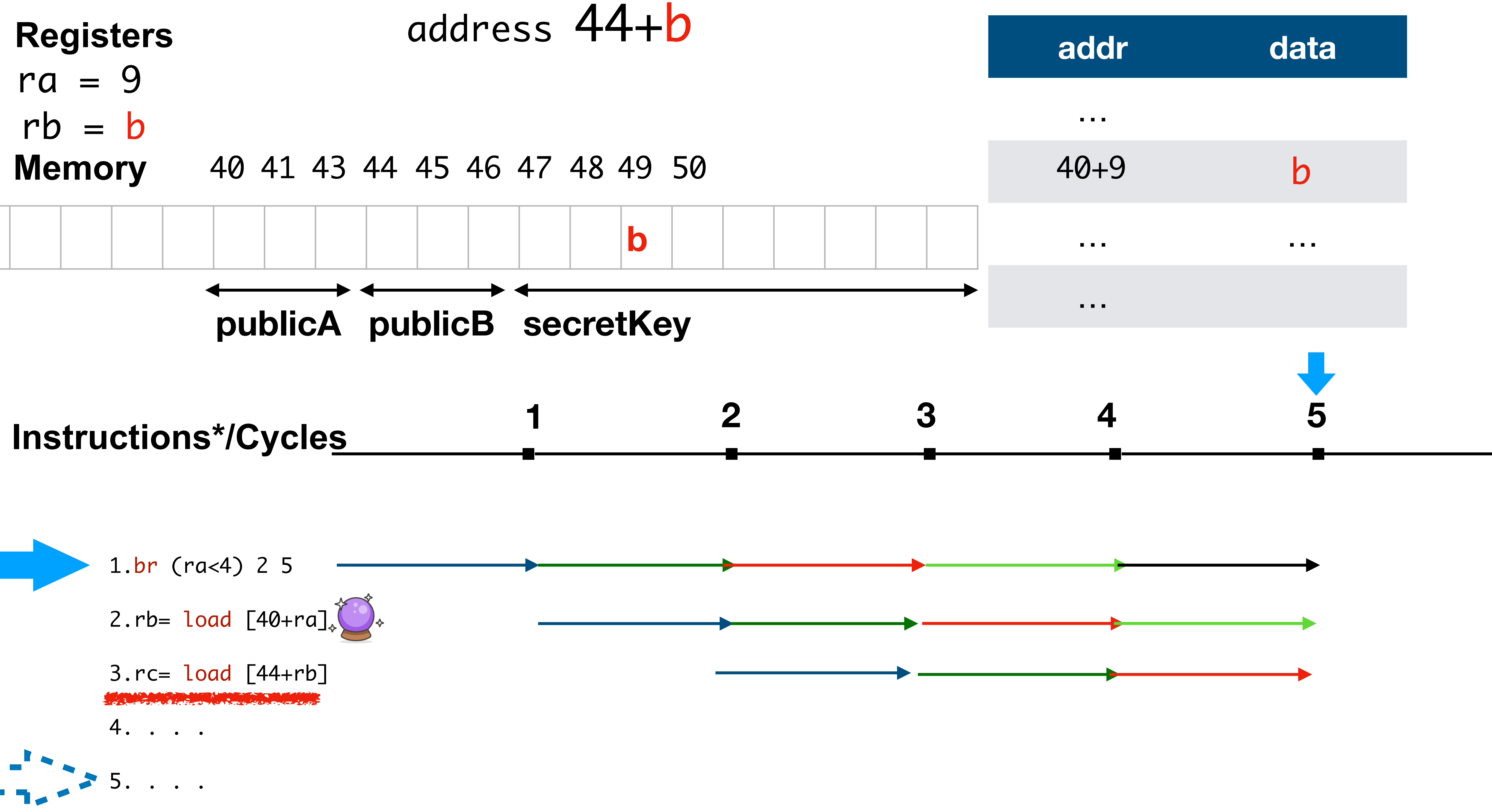
addr	data
…	
40+9	b
…	…
…	



Instructions*/Cycles 1 2 3 4 5

```
1.br (ra<4) 2 5
2.rb= load [40+ra]
3.rc= load [44+rb]
4. . . .
5. . . .
```

Spectre v1 vulnerability



Registers

ra = 9

rb = b

address 44+b

Memory 40 41 43 44 45 46 47 48 49 50

b

publicA **publicB** **secretKey**

Cache

addr	data
…	
40+9	b
…	…
…	

Instructions*/Cycles 1 2 3 4 5

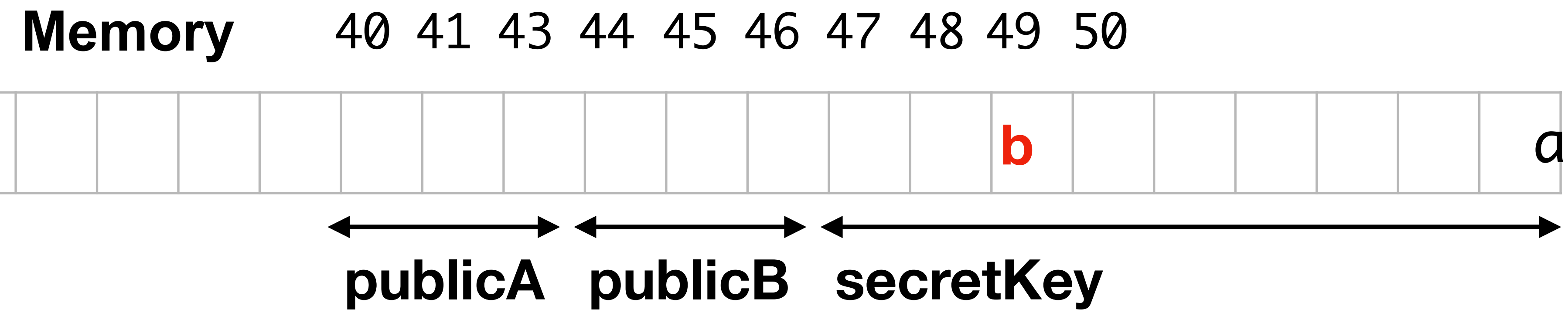
```
1.br (ra<4) 2 5
2.rb= load [40+ra]
3.rc= load [44+rb]
4. . . .
5. . . .
```

Spectre v1 vulnerability

Registers

```
ra = 9
rb = b
```

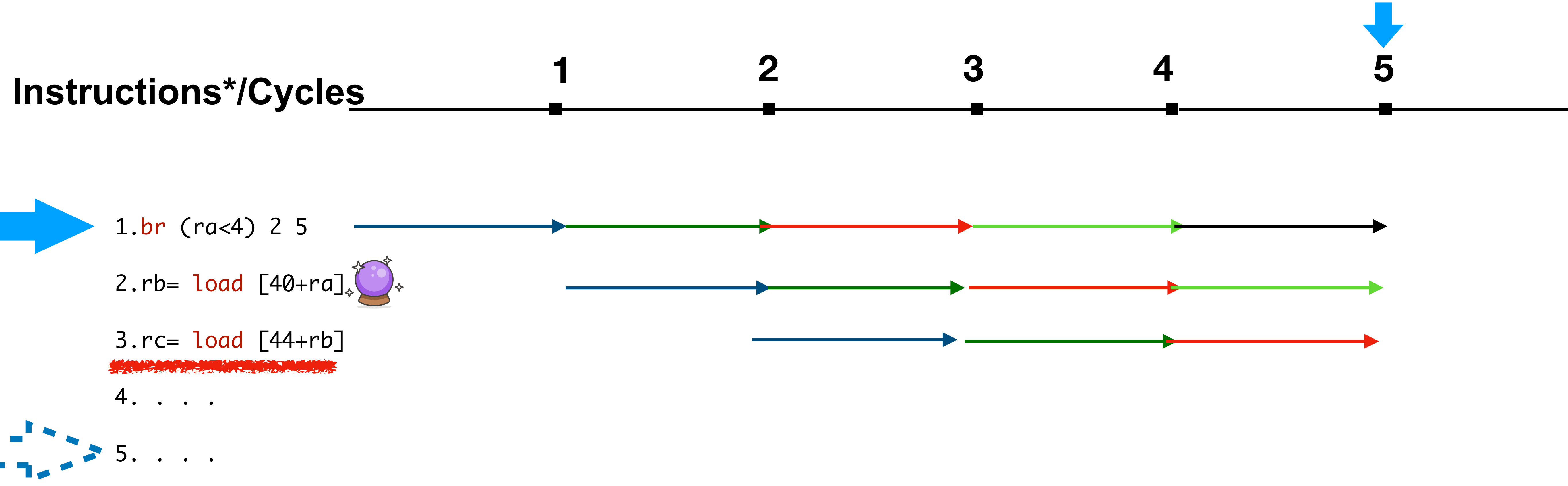
Memory



Cache

addr	data
…	
40+9	b
address 44+b	…
…	

Instructions*/Cycles



```
1.br (ra<4) 2 5
2.rb= load [40+ra]
3.rc= load [44+rb]
4. . . .
5. . . .
```

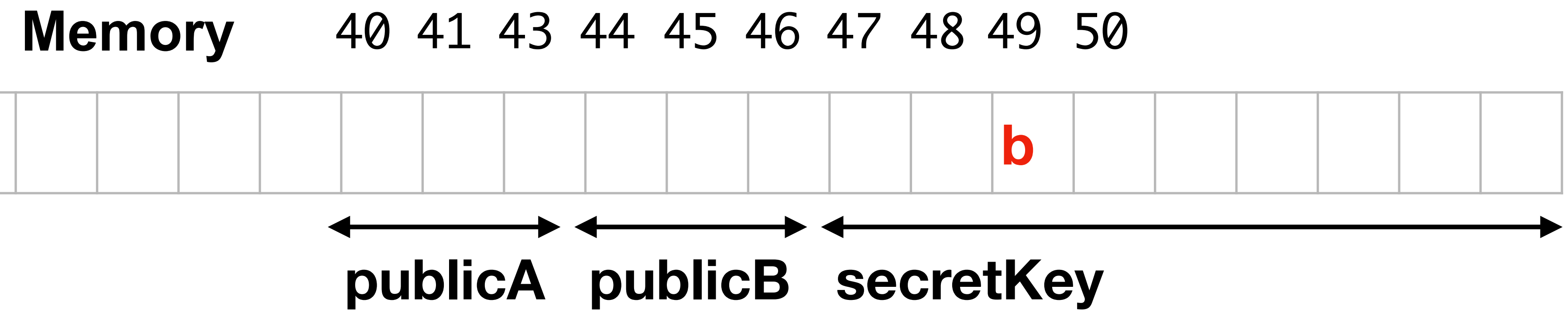
Spectre v1 vulnerability

Registers

ra = 9

rb = b

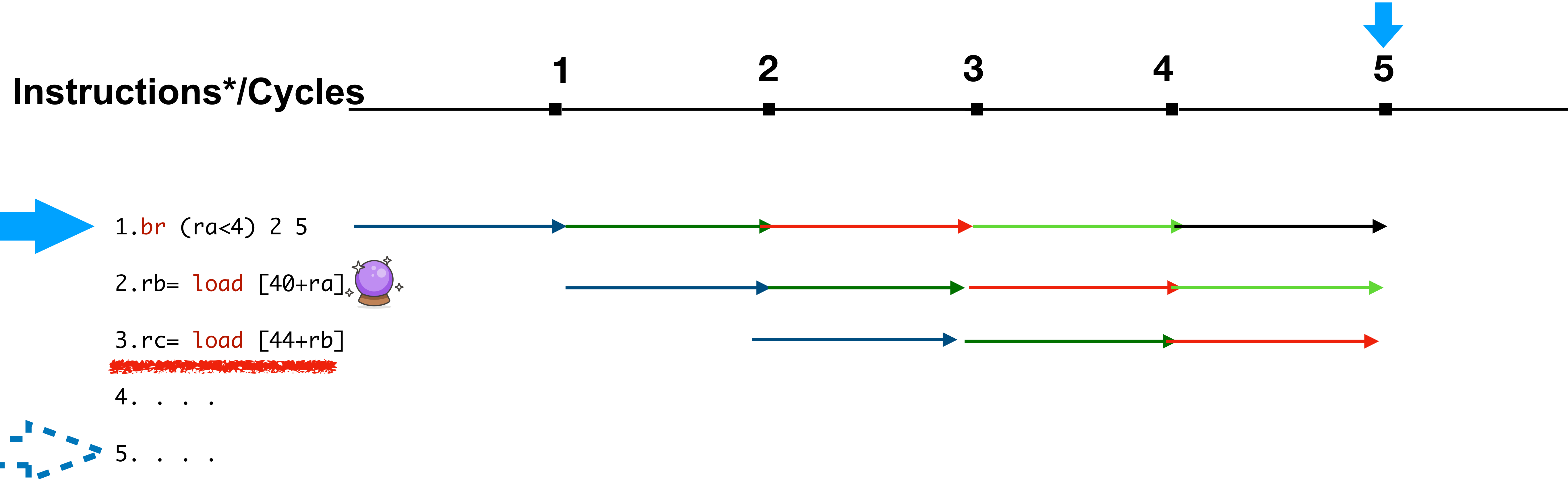
Memory



Cache

addr	data
…	
40+9	b
44+b	…
…	

Instructions*/Cycles



```
1.br (ra<4) 2 5
2.rb= load [40+ra]
3.rc= load [44+rb]
4. . . .
5. . . .
```

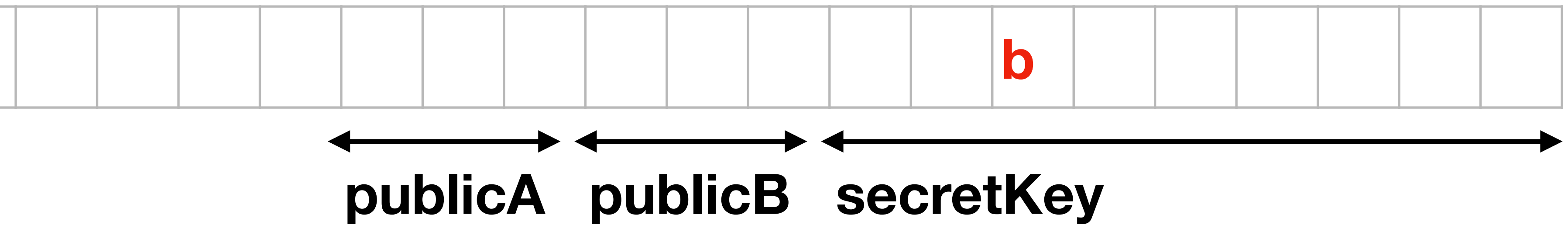
Spectre v1 vulnerability

Registers

ra = 9

rb = b

Memory 40 41 43 44 45 46 47 48 49 50

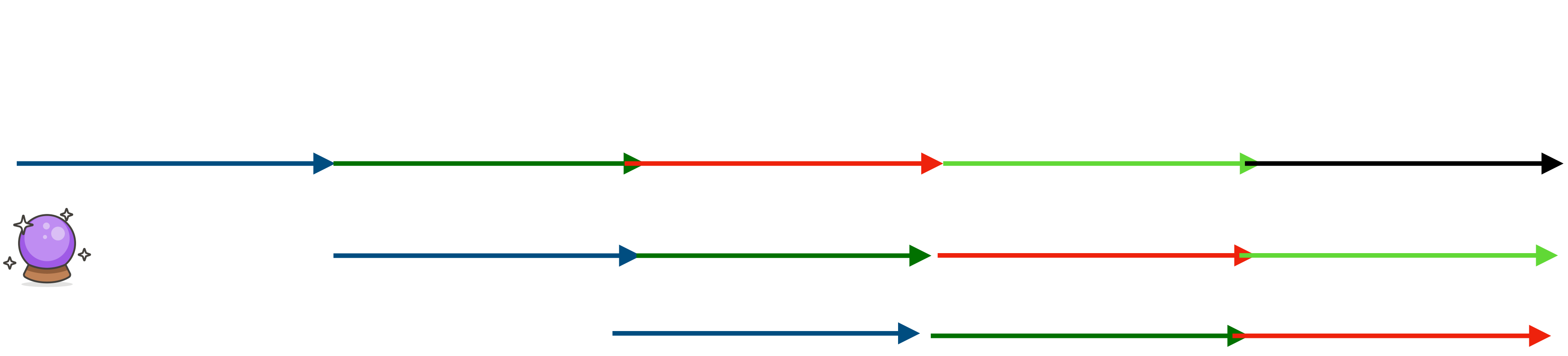


Cache

addr	data
…	
40+9	
44+b	
…	

Secret b in cache addresses: attacker can recover it

Instructions*/Cycles 1 2 3 4



```
1.br (ra<4) 2 5
2.rb= load [40+ra]
3.rc= load [44+rb]
4. . . .
5. . . .
```

Overview

Transient execution attacks: bring you up to speed with Spectre v1

2022: Different variants and their defenses

Open challenges in the area

Different variants

Spectre-family

Meltdown-family

LVI-family

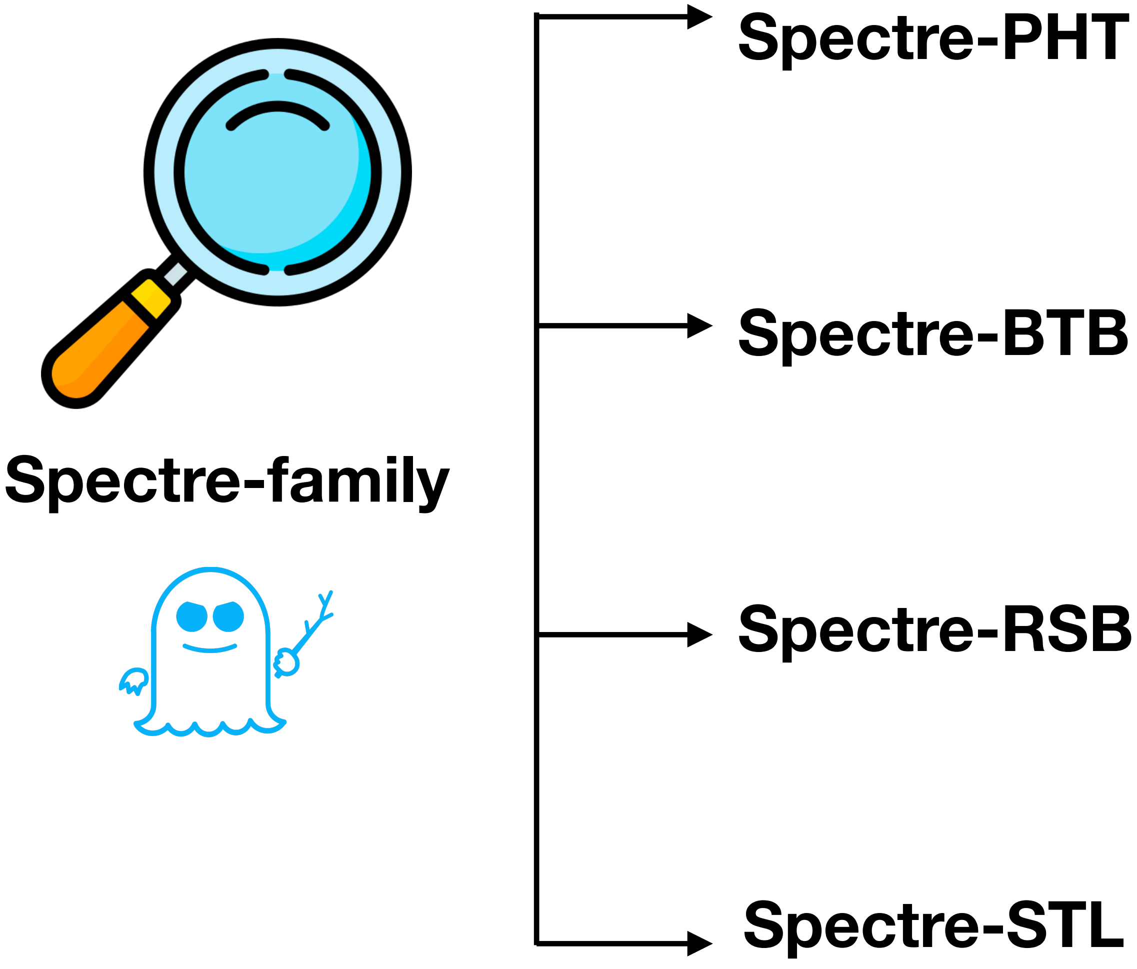
LIV machine clears-family

A Systematic Evaluation of Transient Execution Attacks and Defenses, Canella et al., Usenix Security 2019
https://transient.fail/

LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection, Van-Bulck et al., S&P 2020

Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks, Ragab et al., Usenix Security 2021

Different variants



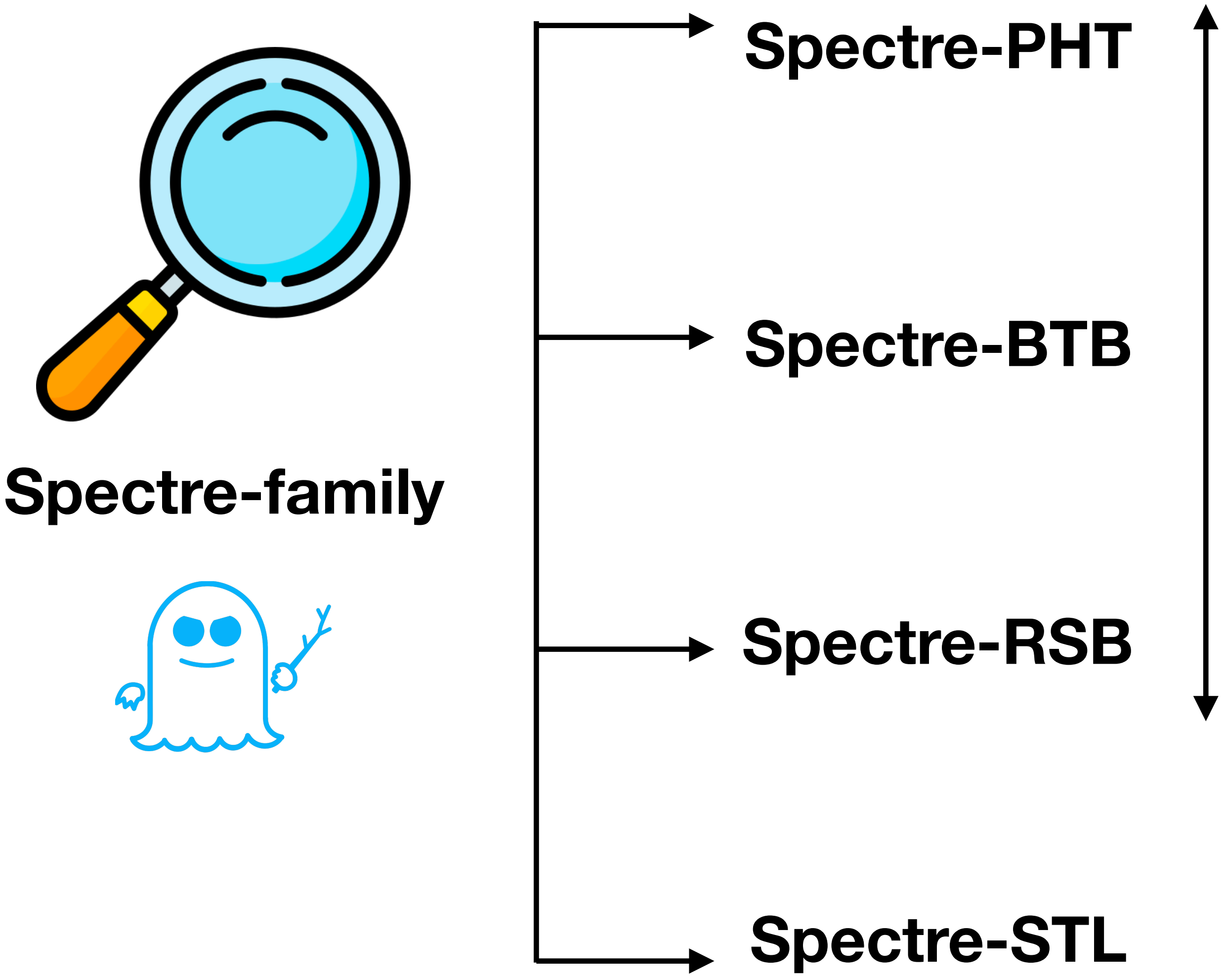
Spectre-family

- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

A Systematic Evaluation of Transient Execution Attacks and Defenses, Canella et al., Usenix Security 2019

https://transient.fail/

Threat models: mistraining strategies



Spectre-family

Spectre-PHT

Spectre-BTB

Spectre-RSB

Spectre-STL

poisoning **intra-process**

poisoning **cross-process**

A Systematic Evaluation of Transient Execution Attacks and Defenses, Canella et al., Usenix Security 2019
https://transient.fail/

2022: Selected widely-used defenses

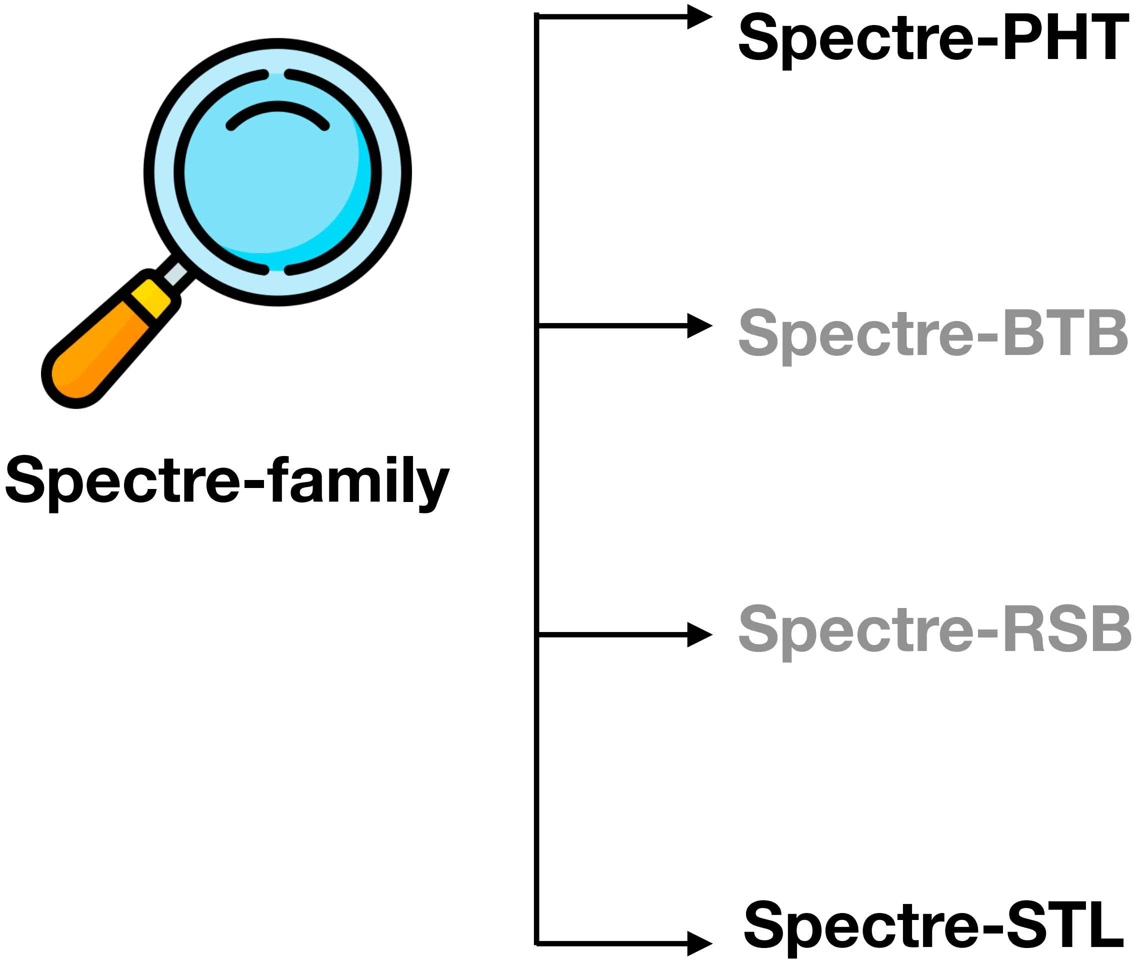
Spectre-family

Spectre-PHT	index-masking (software), SLH (software)
Spectre-BTB	retpolines (software), eIBRS, csv2 (hardware)
Spectre-RSB	RSB stuffing (hardware+OS)
Spectre-STL	SSBD/SSBB (hardware)

+ serialization (lfence)
+ site isolation (Chrome)
+ timer mitigations

2022: Do these defenses work?

2022: Do these defenses work?



Spectre-family

Spectre-PHT

Spectre-BTB

Spectre-RSB

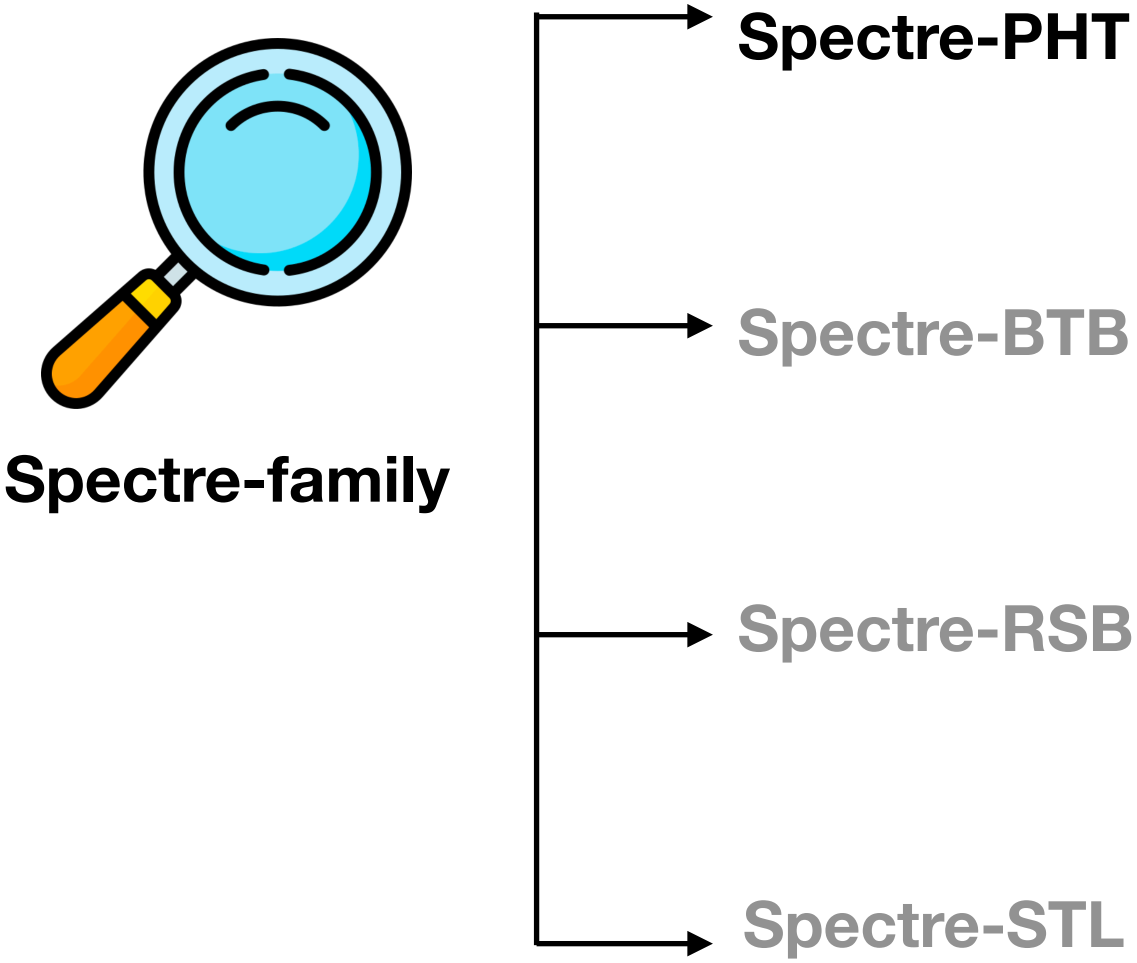
Spectre-STL

index-masking (Webkit), SLH (compilers)

Result:
index-masking (gcc) may
introduce
Spectre-STL

Hunting the haunter - efficient relational symbolic execution for Spectre with Haunted Relse
L. Daniel, S. Bardin, and T. Rezk
NDSS 2021

2022: Do these defenses work?


Spectre-family

Spectre-PHT

Spectre-BTB

Spectre-RSB

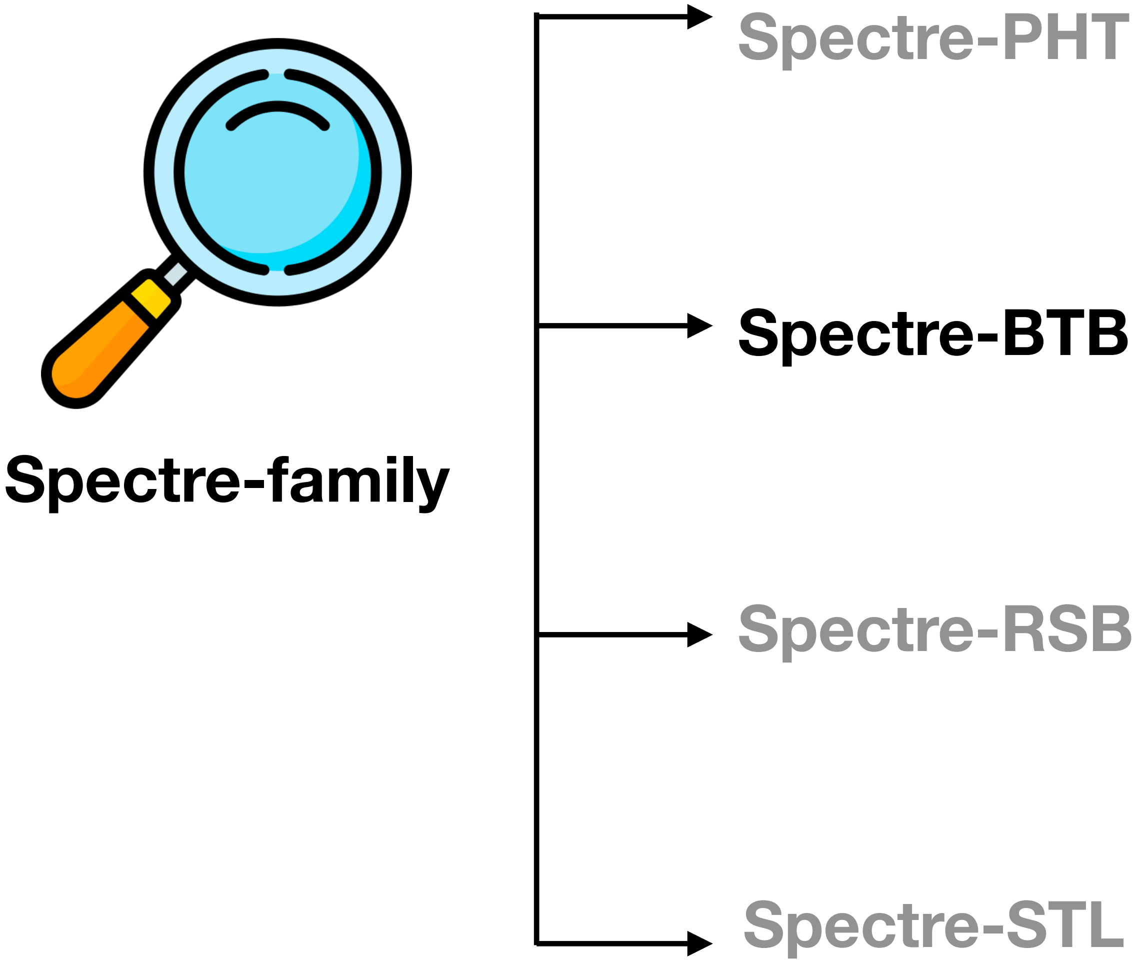
Spectre-STL

index-masking (Webkit), SLH (compilers)

Result:
Spectre-PHT to recover
AES key in spite of SLH

Spectre Declassified: Reading from the Right Place at the Wrong Time
Shivakumar, Barnes, Barthe, Cauligi, Chuengsatieansup, Genkin, O'Connell, Schwabe, Sim, Yarom
eprint 2022

2022: Do these defenses work?



Spectre-family

Spectre-PHT

Spectre-BTB

Spectre-RSB

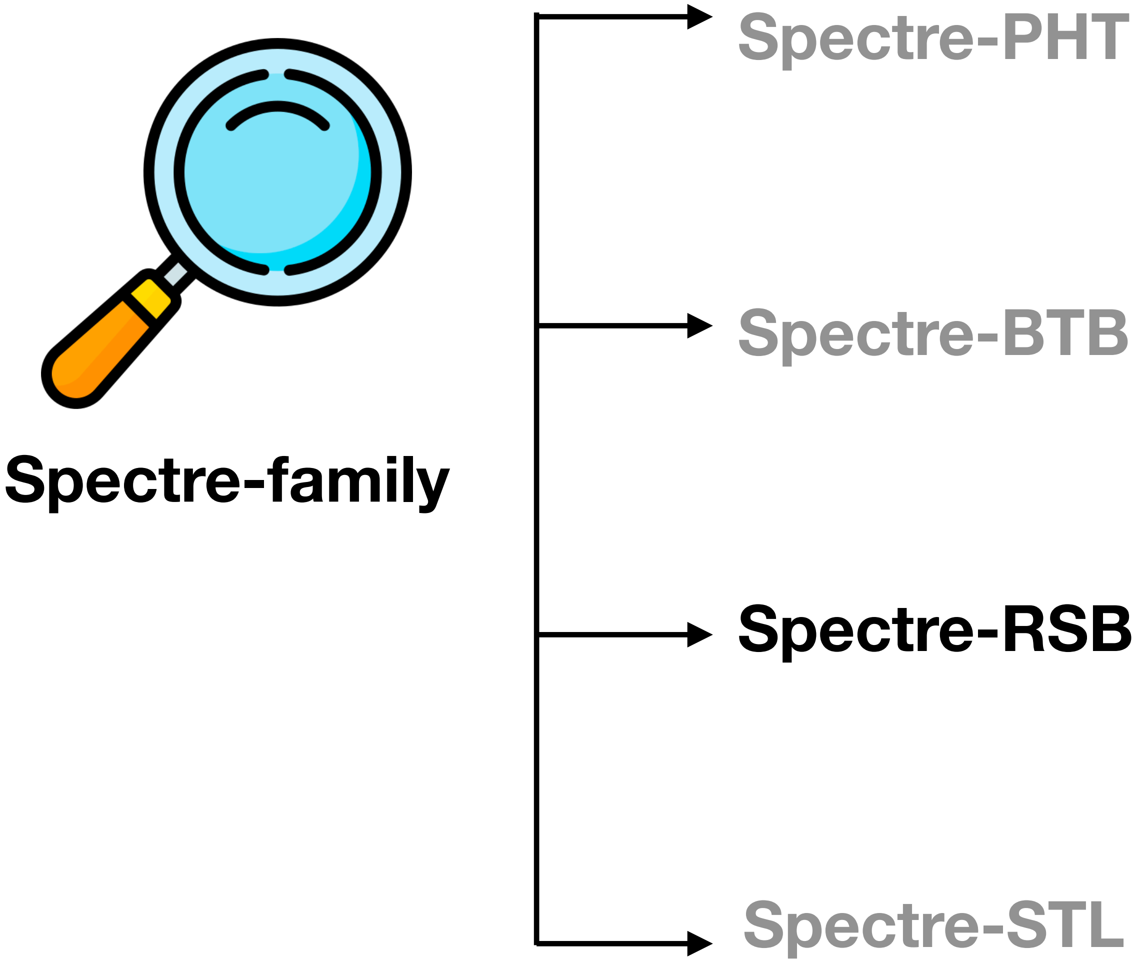
Spectre-STL

retpolines (software), eIBRS, csv2 (hardware)

Result:
eIBPR and csv2
ineffective.
Demo: leaking kernel
memory at 160bytes/s.

Branch History Injection: On the Effectiveness of Hardware Mitigations Against Cross-Privilege Spectre-v2 Attacks
E.Barberis, P.Frigo, M.Muench, H.Bos, C.Giuffrida
Usenix Security 2022

2022: Do these defenses work?



Spectre-family

Spectre-PHT

Spectre-BTB

Spectre-RSB

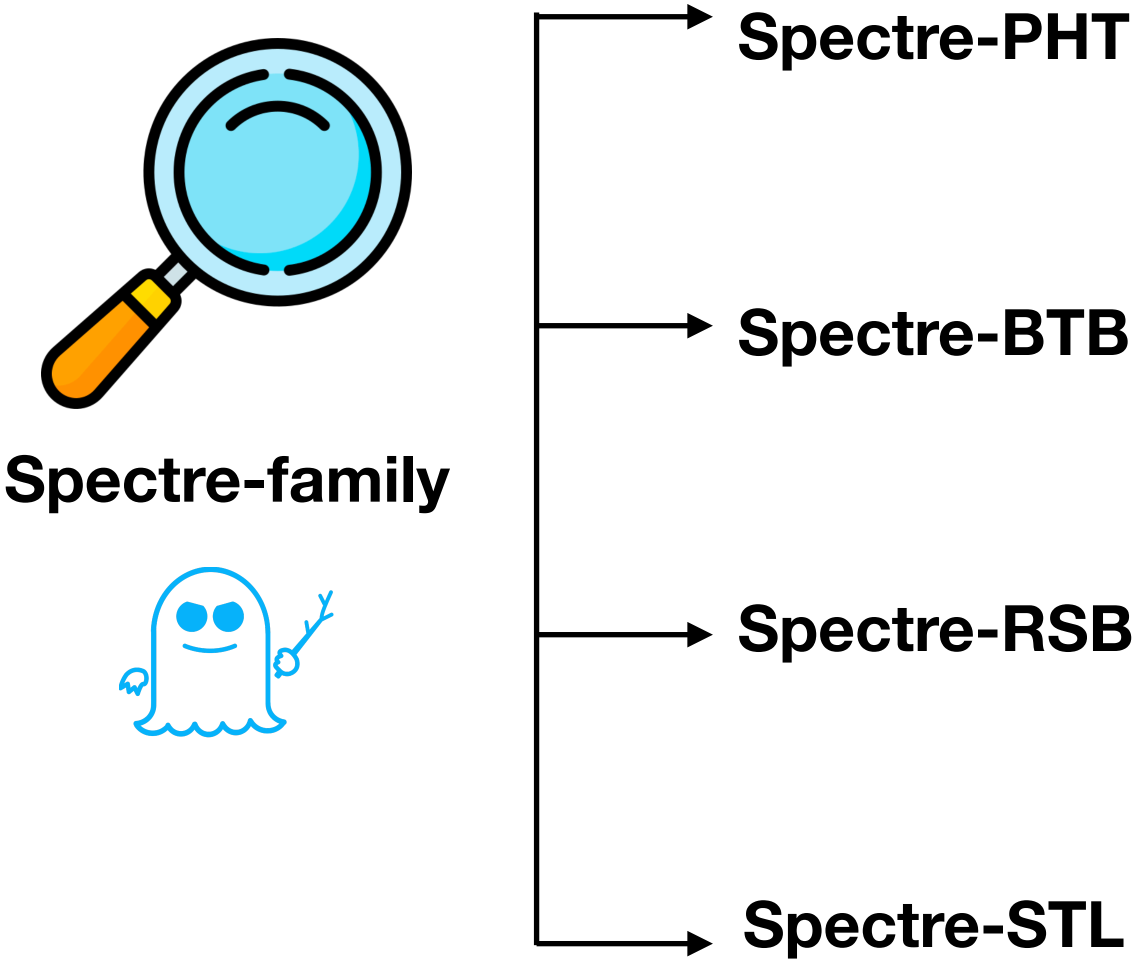
Spectre-STL

Result:
enabling Spectre-RSB attacks via RSB underflows in Firefox.

RSB stuffing + Firefox timer mitigation

Spring: Spectre Returning in the Browser with Speculative Load Queuing and Deep Stacks
J.Wikner, C.Giuffrida, H.Bos, K.Razavi
WOOT 2022

2022: Selected widely-used defenses



Spectre-PHT: index-masking (software), SLH (software)

Spectre-BTB: retpolines (software), eIBRS, csv2 (hardware)

Spectre-RSB: RSB stuffing (hardware+OS)

Spectre-STL: **SSBD/SSBB (hardware)**

+ serialization (lfence)
+ site isolation (Chrome)
+ timer mitigations

Open challenges

1. Attacks: LIV machine clears-based
2. Defenses: More comprehensive formal threat models
3. Hardware: New microarchitectures and new contracts

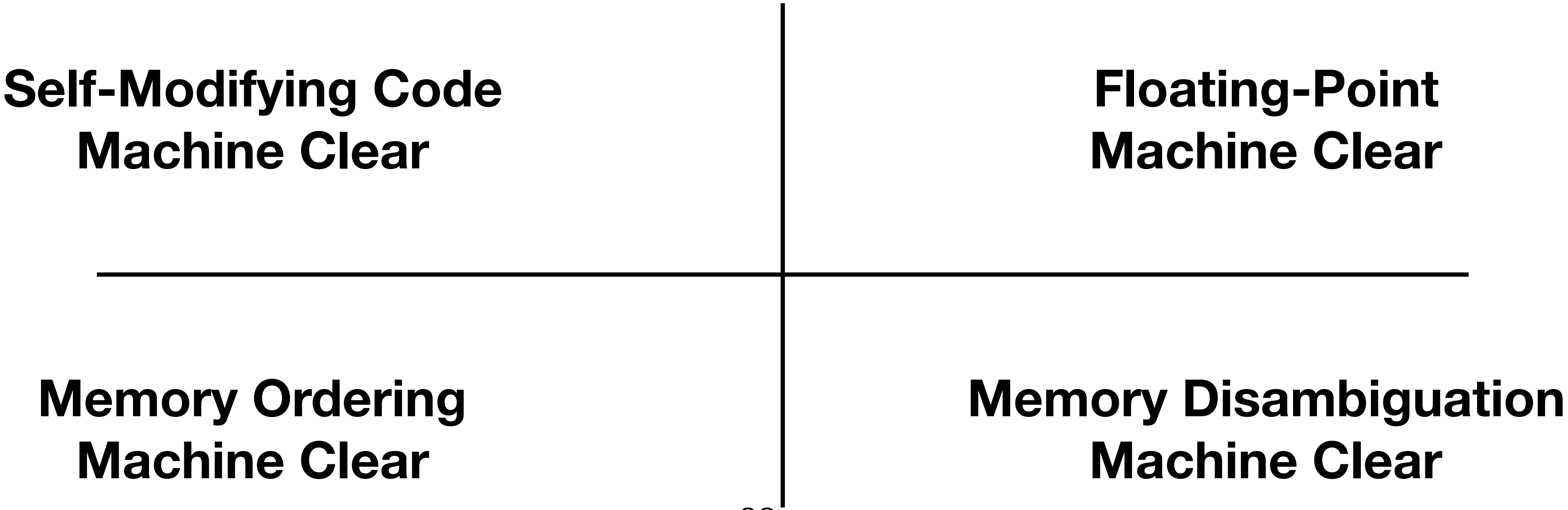
Challenge 1

Attacks: LIV machine clears-based

Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks

H.Ragab, E.Barberis, H.Bos, C.Giuffrida

Usenix Security 2021



Self-Modifying Code Machine Clear

Floating-Point Machine Clear

Memory Ordering Machine Clear

Memory Disambiguation Machine Clear

Challenge 1

Attacks: LIV machine clears-based

Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks

H.Ragab, E.Barberis, H.Bos, C.Giuffrida
Usenix Security 2021

Which new attacks based on LIV machine clears are there?

Self-Modifying Code Machine Clear	**Floating-Point Machine Clear**
Memory Ordering Machine Clear	**Memory Disambiguation Machine Clear**

Challenge 2

Defenses: More comprehensive formal threat models

Understanding microarchitectural vulnerabilities and countermeasures

Frank Piessens - Keynote IEEE EuroS&P 2021

Constant-Time Foundations for the New Spectre Era

S.Cauligi, C. Disselkoen, K. Gleissenthall, D. Tullsen, D. Stefan, T. Rezk, G. Barthe

PLDI 2020

Hardware-Software Contracts for Secure Speculation

M.Guarnieri, B.Köpf, J.Reineke, P.Vila

IEEE S&P 2021

SoK: Practical Foundations for Spectre Defenses

S.Cauligi, C. Disselkoen, D.Moghimi, G. Barthe, D.Stefan

IEEE S&P 2022

Cats vs. Spectre: An Axiomatic Approach to Modeling Speculative Execution Attacks

H. Ponce-de-León, Johannes Kinder

IEEE S&P 2022

Challenge 2

Defenses: More comprehensive formal threat models

Understanding microarchitectural vulnerabilities and countermeasures

Frank Piessens - Keynote IEEE EuroS&P 2021

Constant-Time Foundations for the New Spectre Era

S.Cauligi, C. Disselkoen, K. Gleissenthall, D. Tullsen, D. Stefan, T. Rezk, G. Barthe

PLDI 2020

Hardware-Software Contracts for Secure Speculation

Which semantics to capture more transient execution attacks families?
which security properties for those models?

S.Cauligi, C. Disselkoen, D.Moghimi, G. Barthe, D.Stefan

IEEE S&P 2022

Cats vs. Spectre: An Axiomatic Approach to Modeling Speculative Execution Attacks

H. Ponce-de-León, Johannes Kinder

IEEE S&P 2022

Challenge 3

Hardware: New microarchitectures

"Future processors could potentially track whether data was fetched as the result of a speculative operation and, if so, prevent that data from being used in subsequent operations that might leak it"

Section VII, **Spectre Attacks: Exploiting Speculative Execution**

Kocher, Horn, Fogh, Genkin, Gruss, Haas, Hamburg,Lipp, Mangard, Prescher, Schwarz, Yarom

ConTExT: A Generic Approach for Mitigating Spectre
Schwarz, Lipp, Canella, Schilling, Kargl, Gruss
NDSS 2020

Speculative Privacy Tracking (SPT): Leaking Information from Speculative Execution without Compromising Privacy
Choudhary, Yu, Fletcher, Morrison
MICRO 2021

Challenge 3

Hardware: New microarchitectures

"Future processors could potentially track whether data was fetched as the result of a speculative operation and, if so, prevent that data from being used in subsequent operations that might leak it"

Section VII, **Spectre Attacks: Exploiting Speculative Execution**

Kocher, Horn, Fogh, Genkin, Gruss, Haas, Hamburg,Lipp, Mangard, Prescher, Schwarz, Yarom

ConTExT: A Generic Approach for Mitigating Spectre
Schwarz, Lipp, Canella, Schilling, Kargl, Gruss
NDSS 2020

Speculative Privacy Tracking (SPT): Leaking Information from Speculative Execution without Compromising Privacy
Choudhary, Yu, Fletcher, Morrison
MICRO 2021

Does taint-tracking based defenses extend to other families?
Which new microarchitectures do we need?

Conclusion

Have Transient Execution Attacks Been Fully Solved?

Hot Topics

Challenge 1 Attacks: LIV machine clears based

Which new attacks based on LIV machine clears are there?

Challenge 2 Defenses: More comprehensive formal threat models

Which semantics to capture more transient execution attacks families?
which security properties for those models?

Challenge 3 Hardware: New microarchitectures and new contracts

Does taint-tracking based defenses extend to other families?
Which new microarchitectures do we need?

Exercises 26/9

Exercise 1 Attacks: https://transient.fail/

See the PoCs of the different attacks. Leak the secret with at least two different attacks. Understand how the different attacks work.

Exercise 2 Defense: defend against Spectre PHT (a.k.a. v1)

Read the original Spectre paper https://spectreattack.com/spectre.pdf and implement a defense for Spectre PHT for the provided PoC.

Exercise 3 (difficult) Hardware: New microarchitecture

Read the paper ConTExT: A Generic Approach for Mitigating Spectre and figure out which Spectre attacks this hardware prevents and if it can also cover Load Value Injection attacks.
https://www.ndss-symposium.org/ndss-paper/context-a-generic-approach-for-mitigating-spectre/

Next lessons:

03/10 (**virtual** 9-10h, in-person INRIA on the week):
4 groups, 3 persons MAX, paper choice around this topic
Tuesday 4/10 starting at 14h, Thursday 6/10 (30 minutes)

afterwards (in-person): symbolic execution, formal methods for security, research project.