Software Diversity vs Side Channels

Stefan Brunthaler

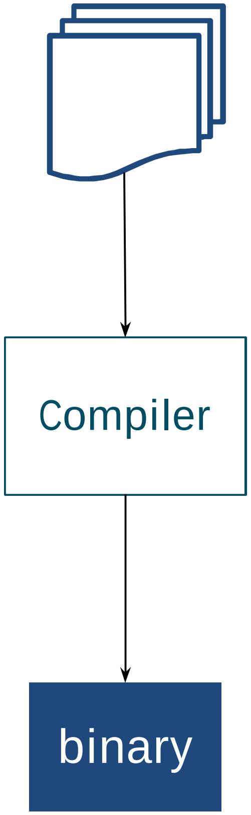
SPECTRE • CODE • BUM

Security, privacy &
Performance
Enhancing
Compilation
Techniques
Research lab

National Cyber
Defense
Research
Institute

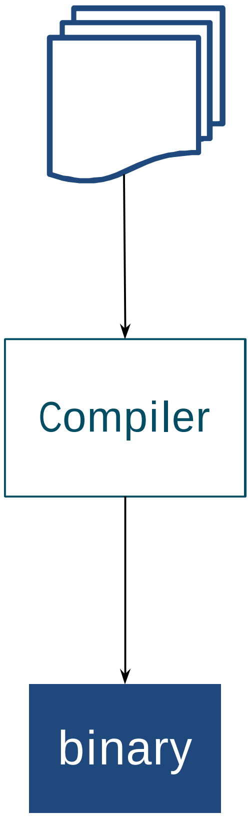
Bundeswehr
University
Munich

Software Monoculture



All users run the same binary
(incl. attackers)

SOFTWARE MONOCULTURE

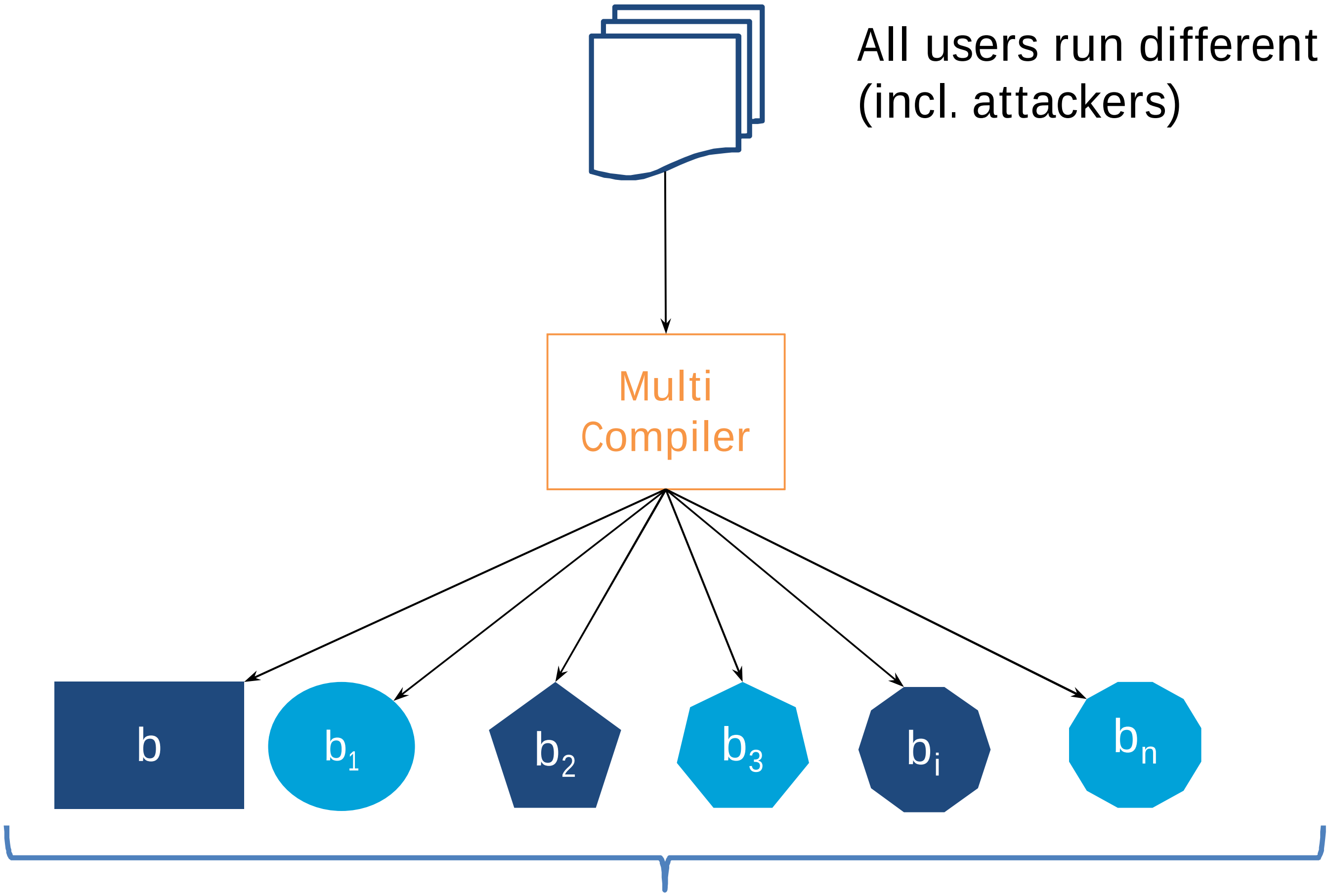


All users run the same binary
(incl. attackers)

Fundamental unfair advantage
& huge economies of scale

What is Software Diversity?

All users run different binaries
(incl. attackers)

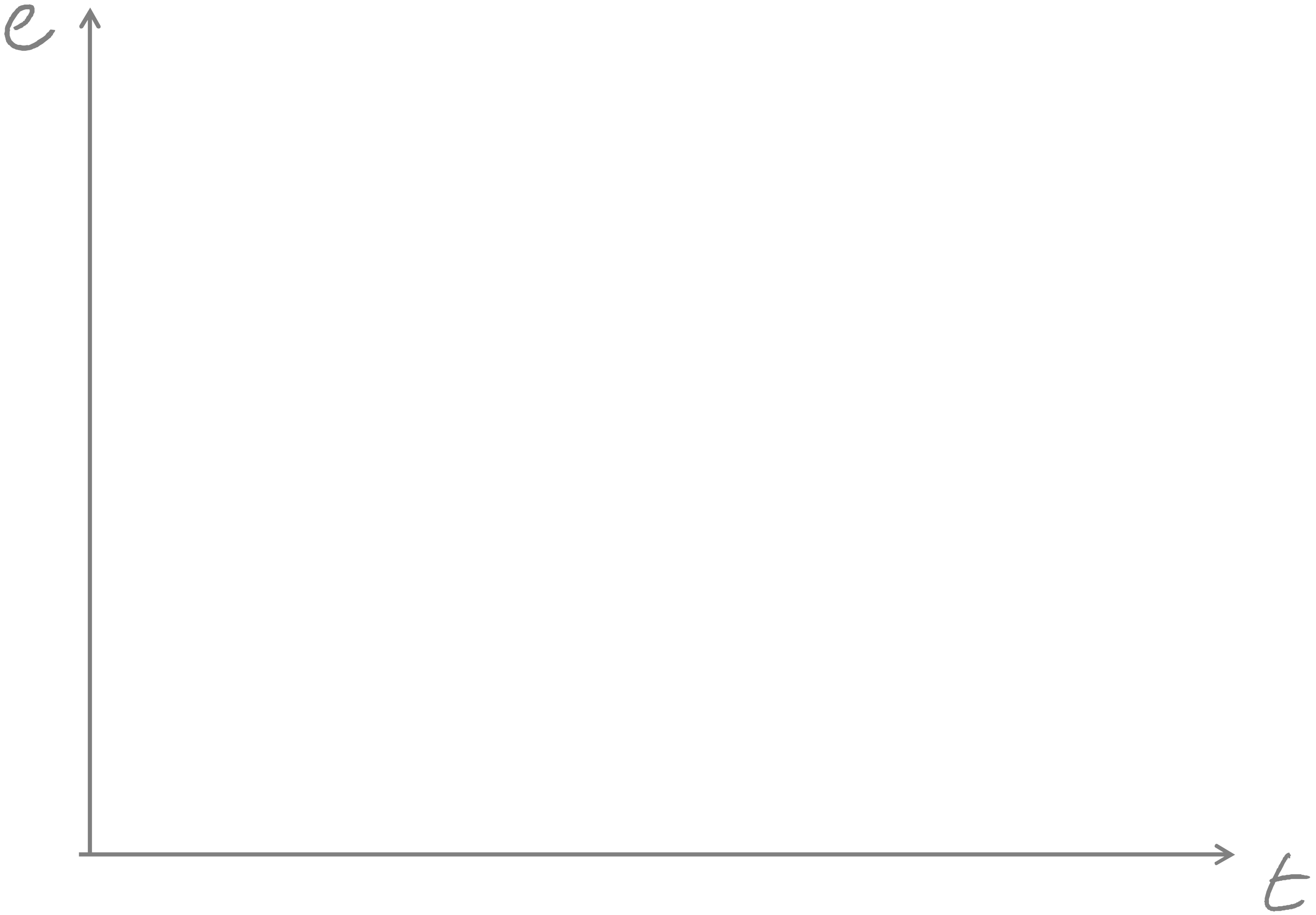


Multi
Compiler

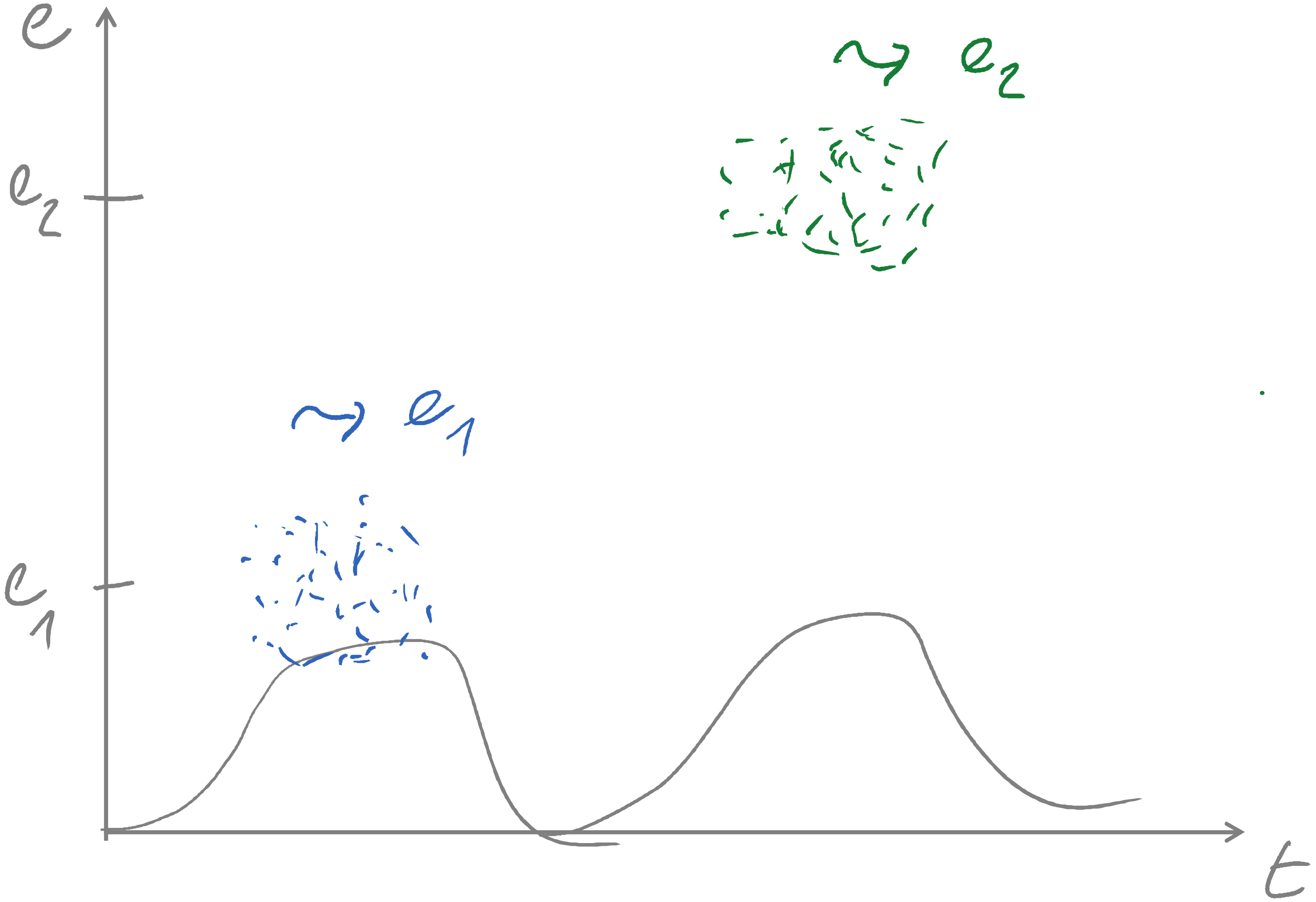
b b_1 b_2 b_3 b_i b_n

diversified code
same semantics

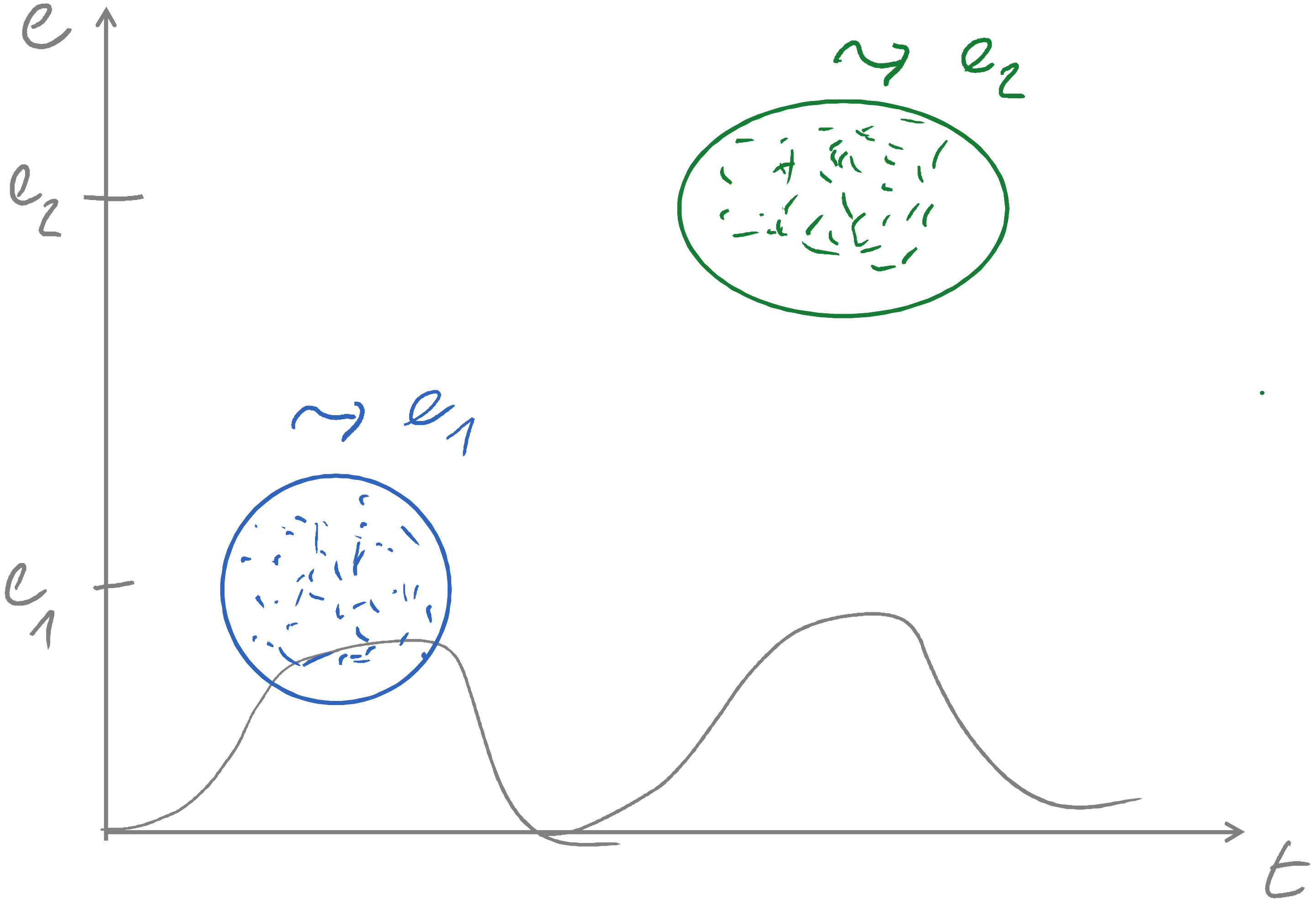
SIDE CHANNELS 101



Side Channels 101



Side Channels 101

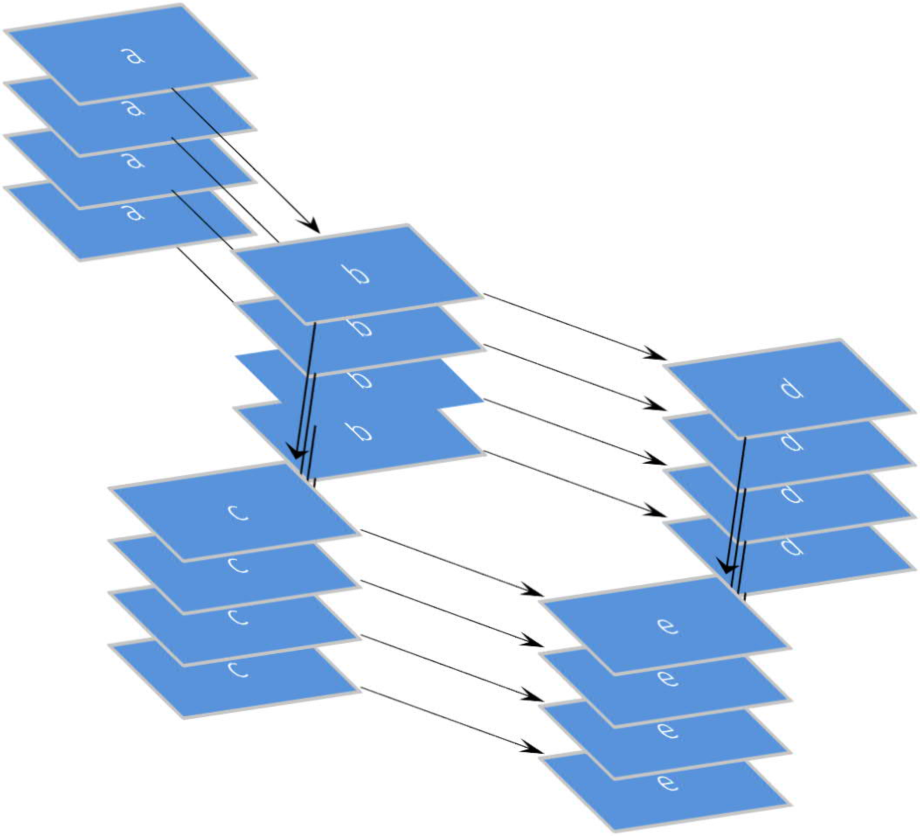


DYNAMIC DIVERSITY

IDEA:

1. replicate program n times

DYNAMIC DIVERSITY – REPLICATE CODE

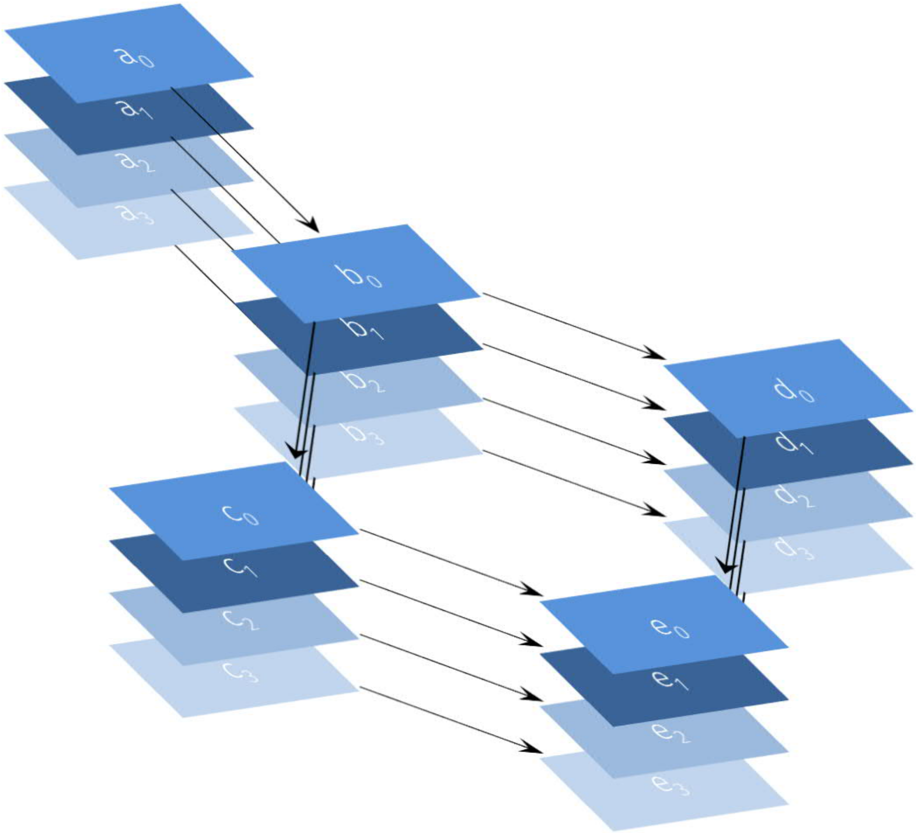


DYNAMIC DIVERSITY

IDEA:

1. replicate program n times
2. diversify code blocks
   - basic blocks
   - functions

DYNAMIC DIVERSITY – DIVERSIFY CODE

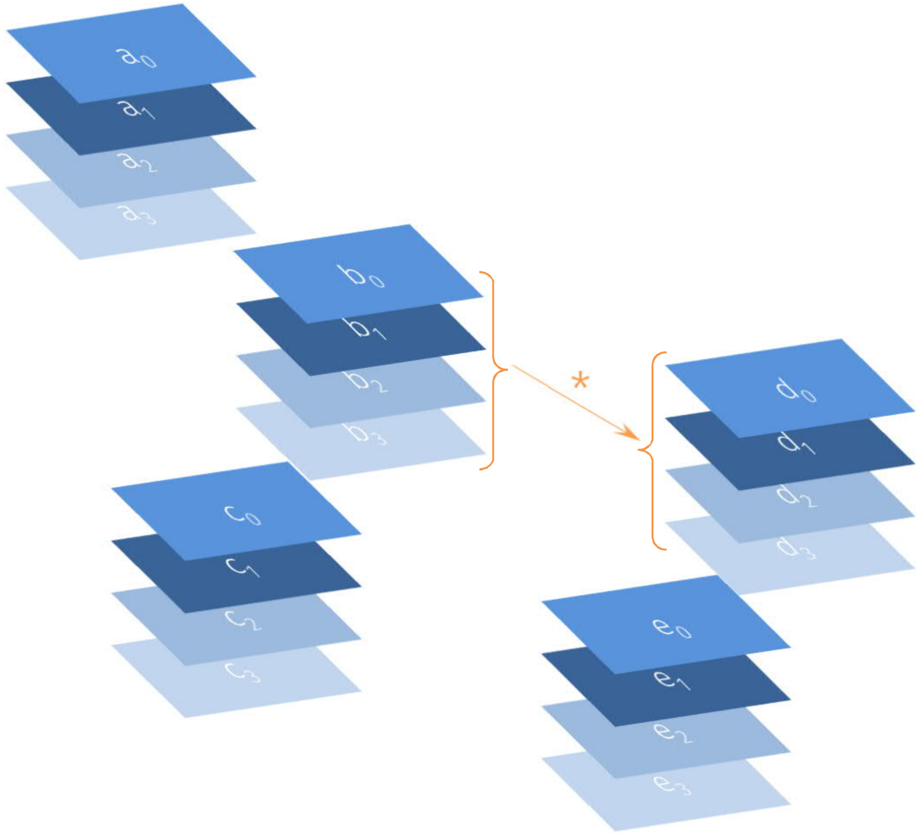


DYNAMIC DIVERSITY

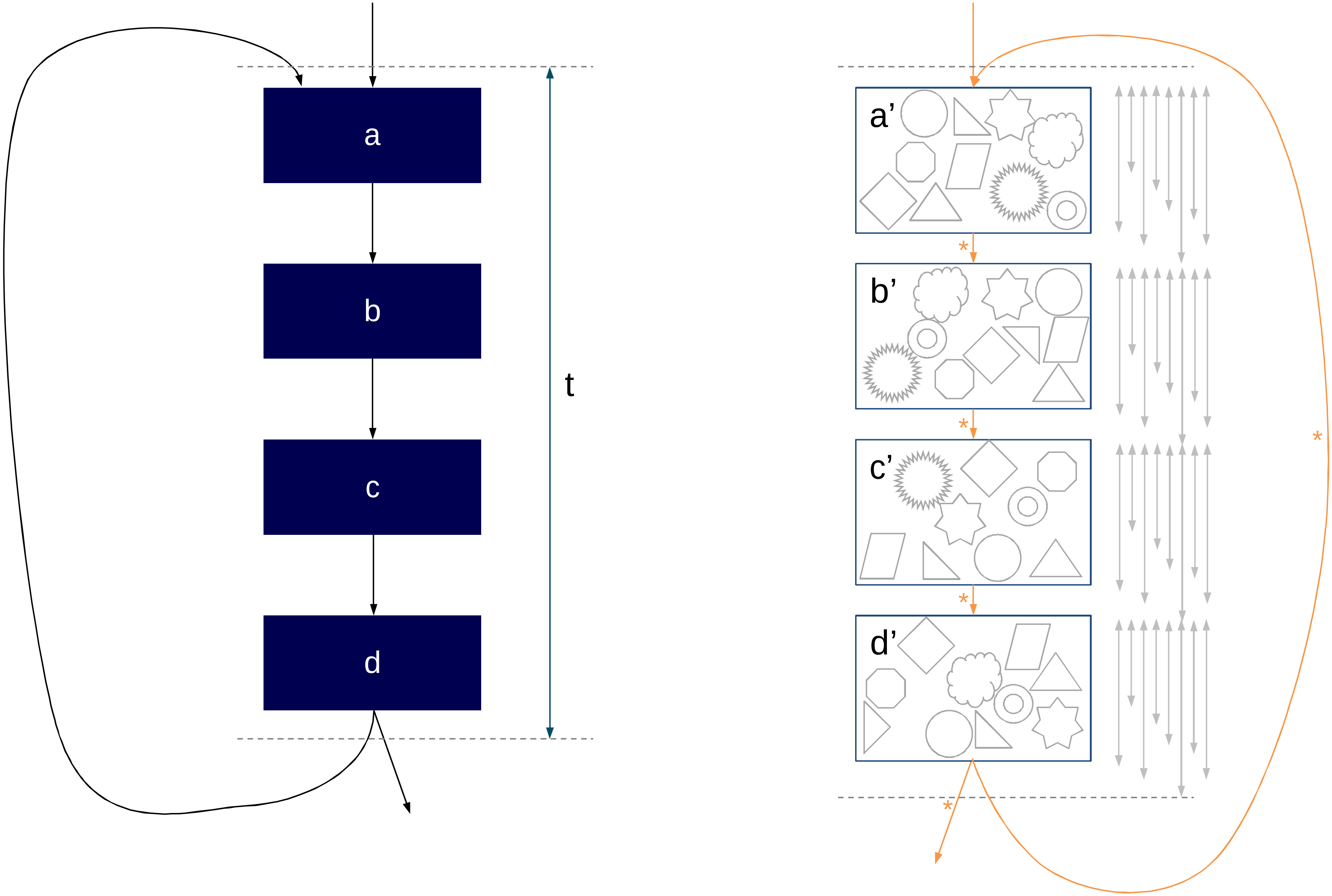
IDEA:

1. replicate program n times
2. diversify code blocks
   - basic blocks
   - functions
3. randomize control-flow

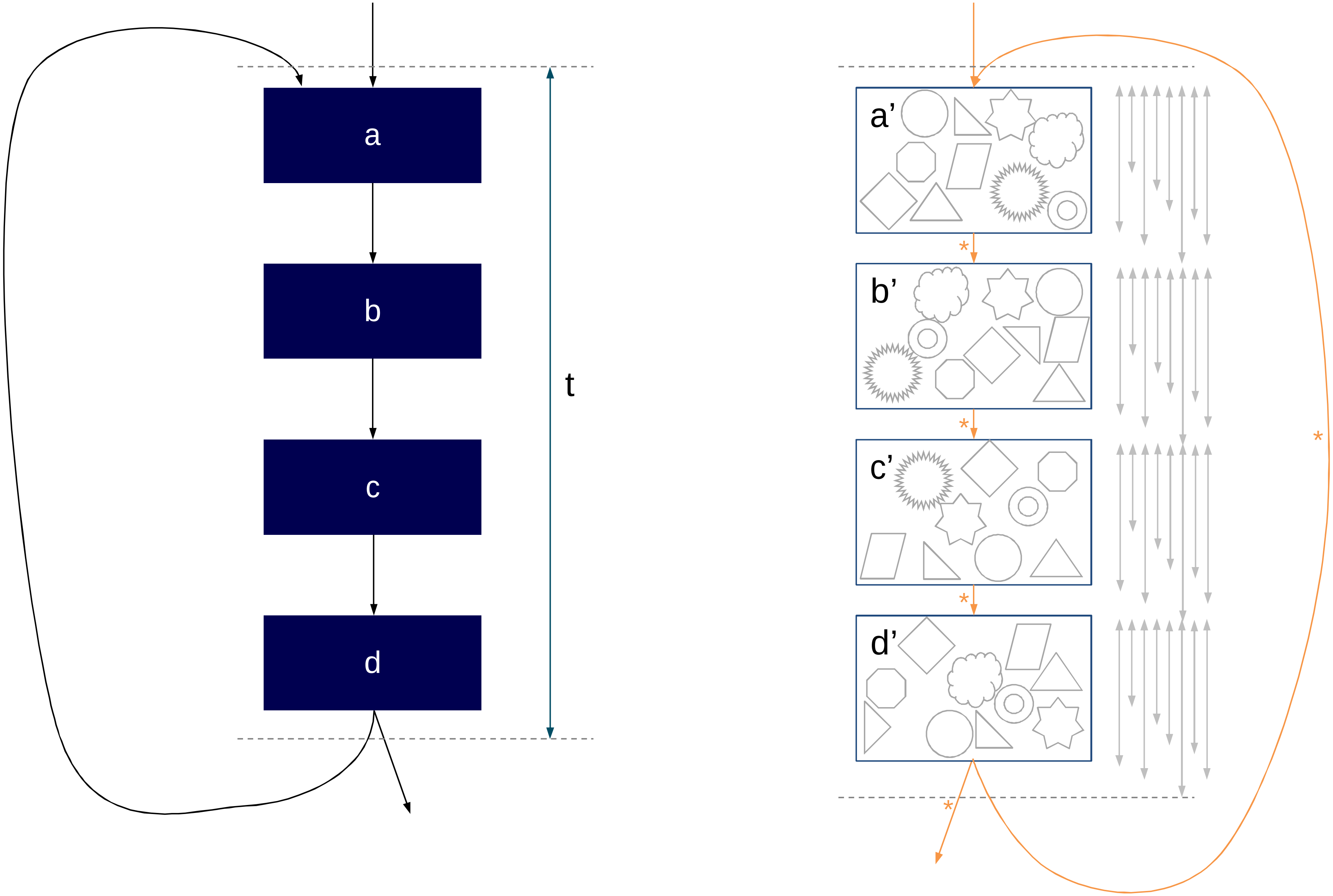
Dynamic Diversity – Randomize CF



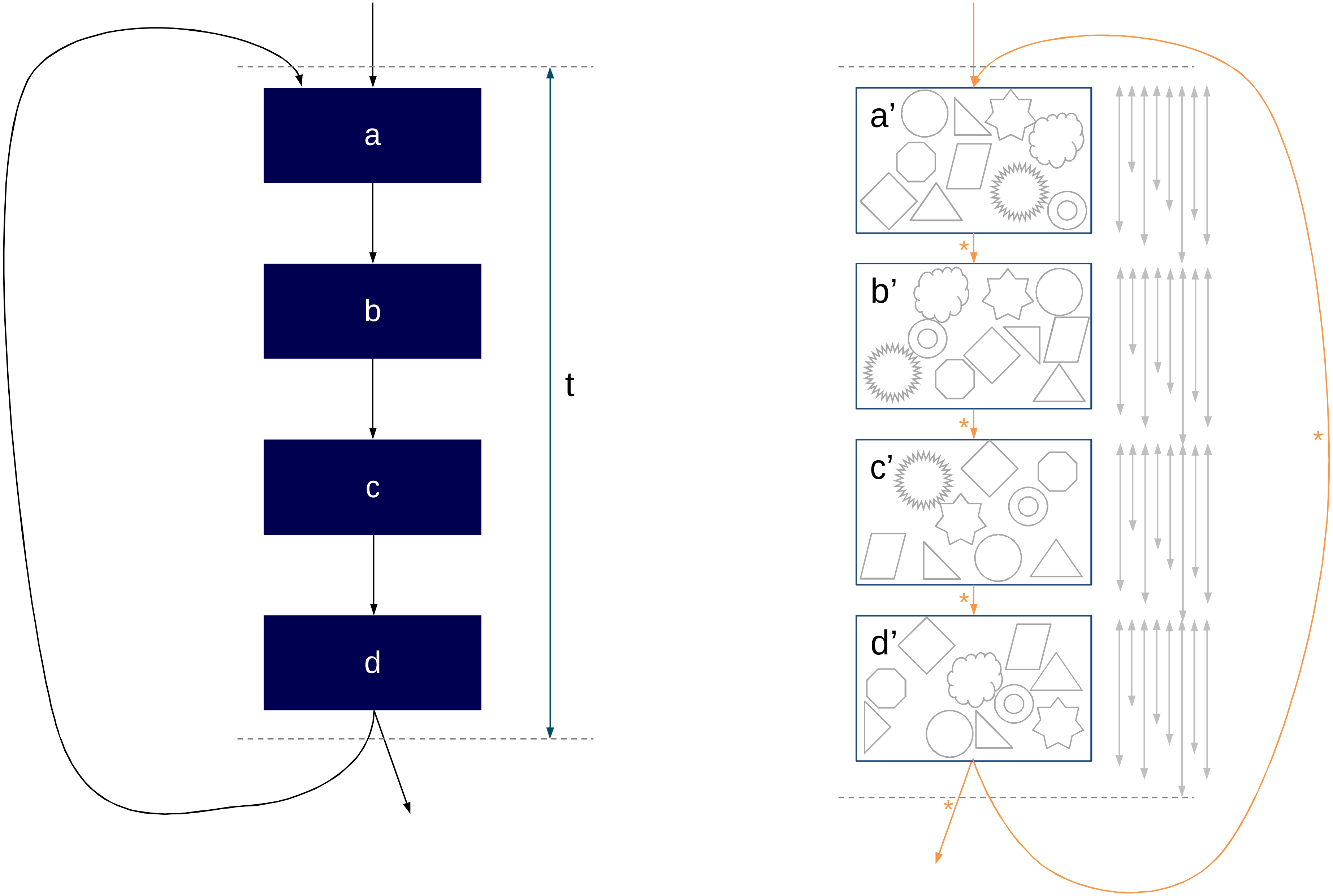
Dynamic Diversity effect on Timing



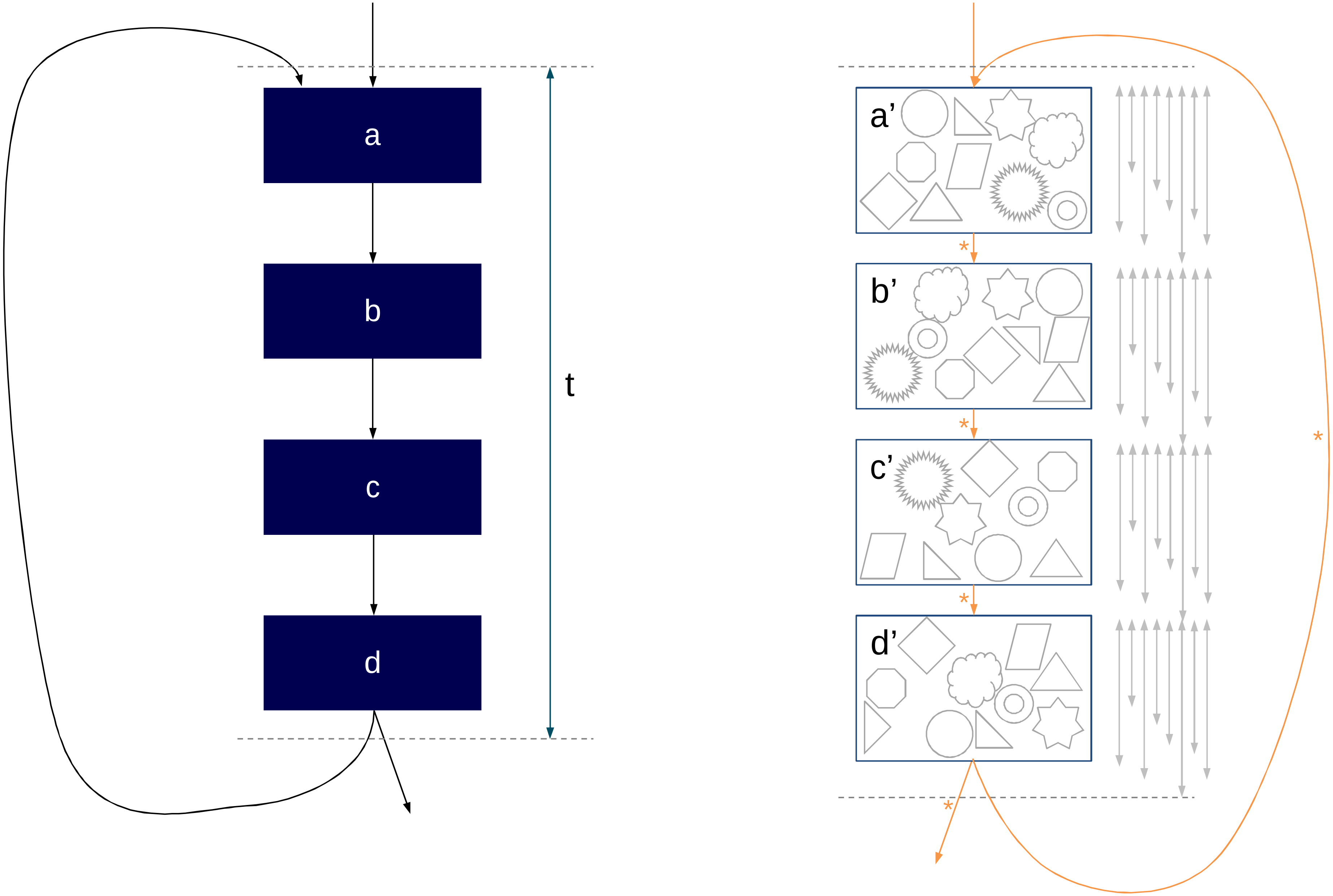
Dynamic Diversity effect on Timing



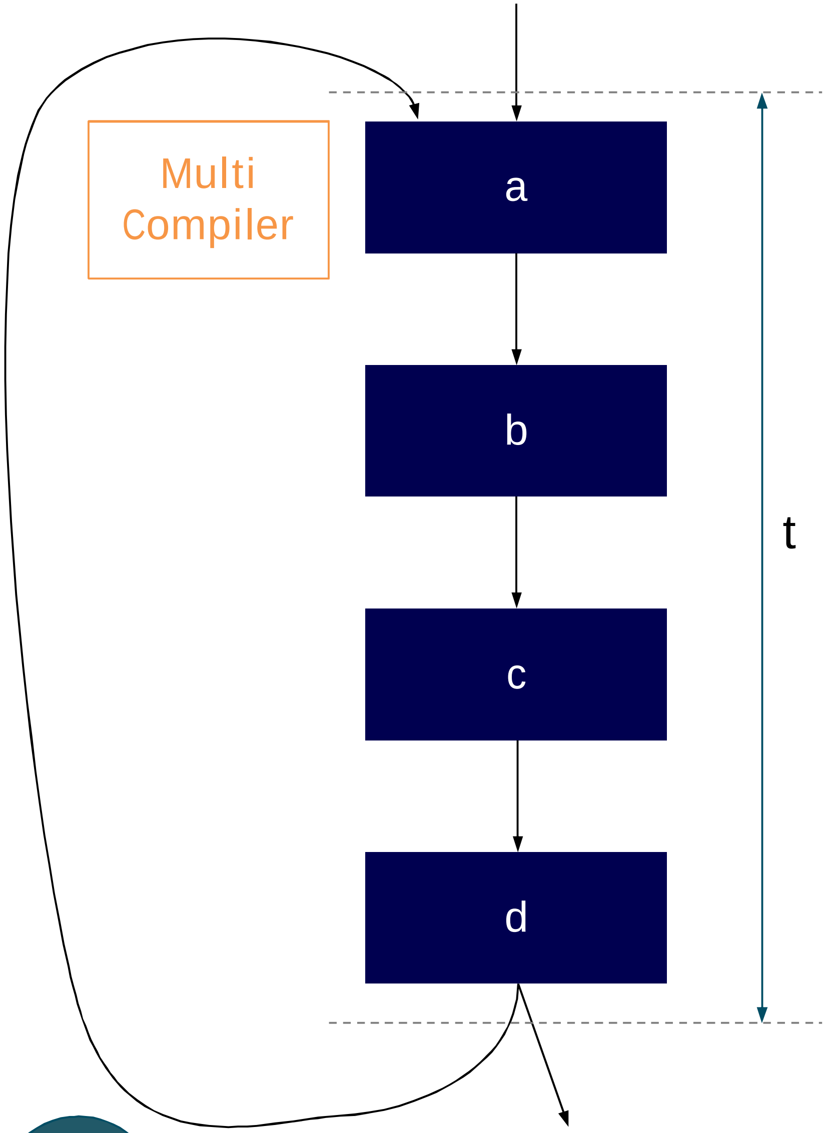
Dynamic Diversity effect on Timing

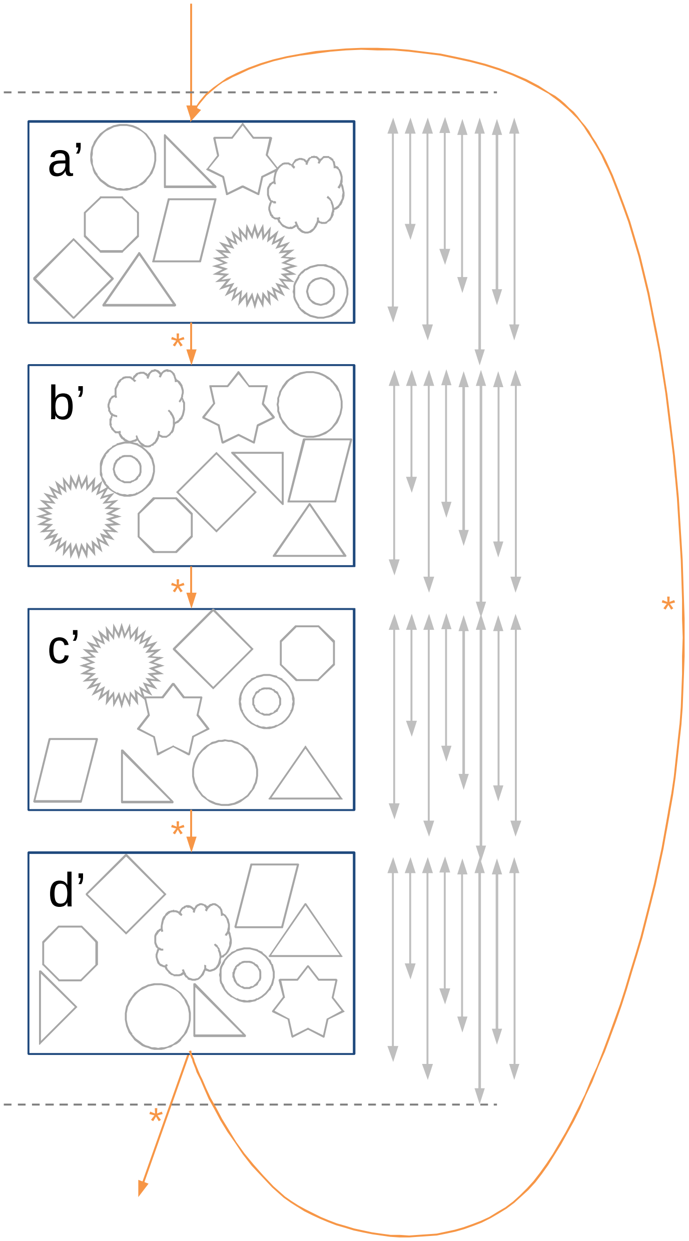


Dynamic Diversity effect on Timing

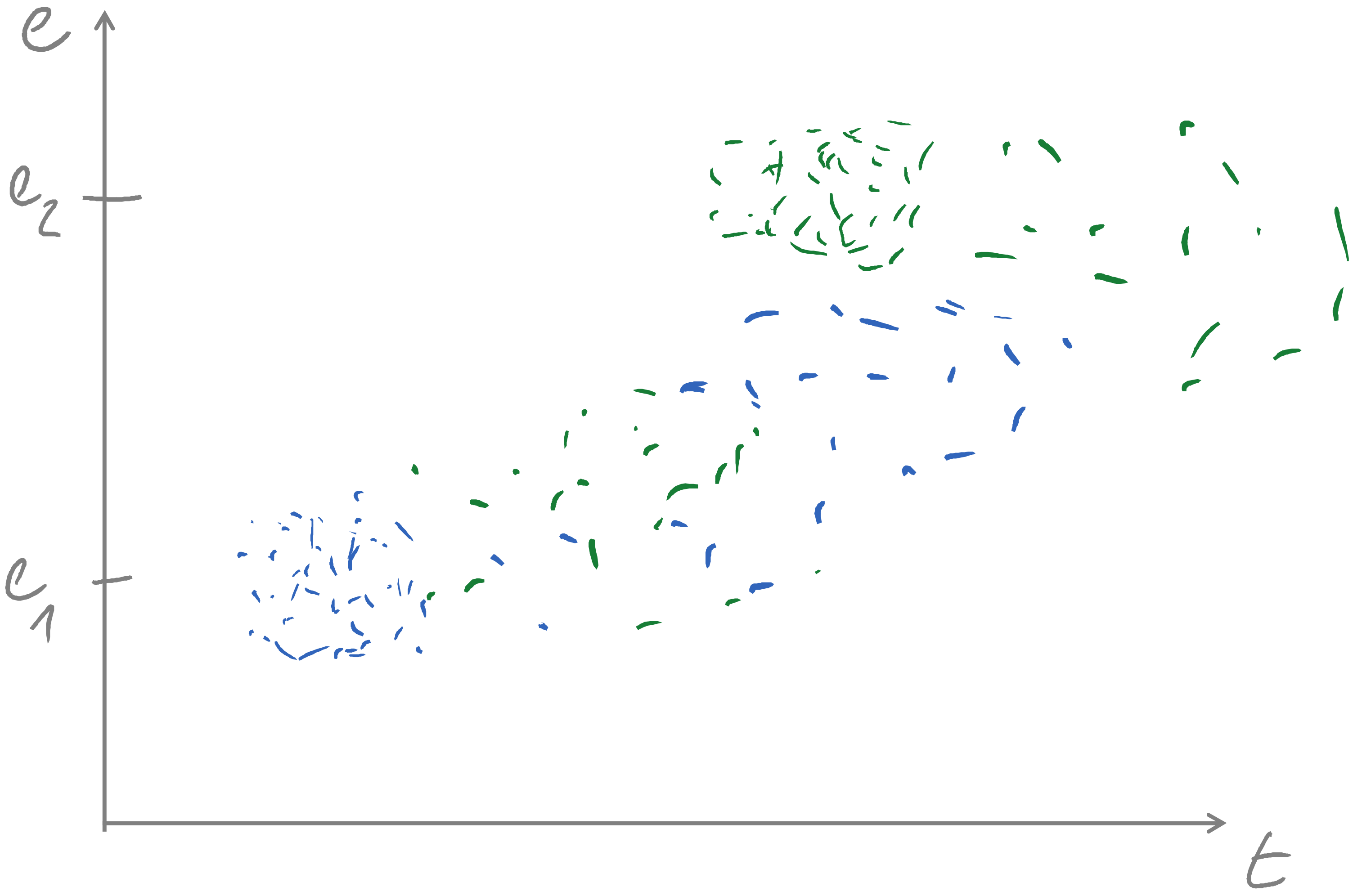


Dynamic Diversity effect on Timing

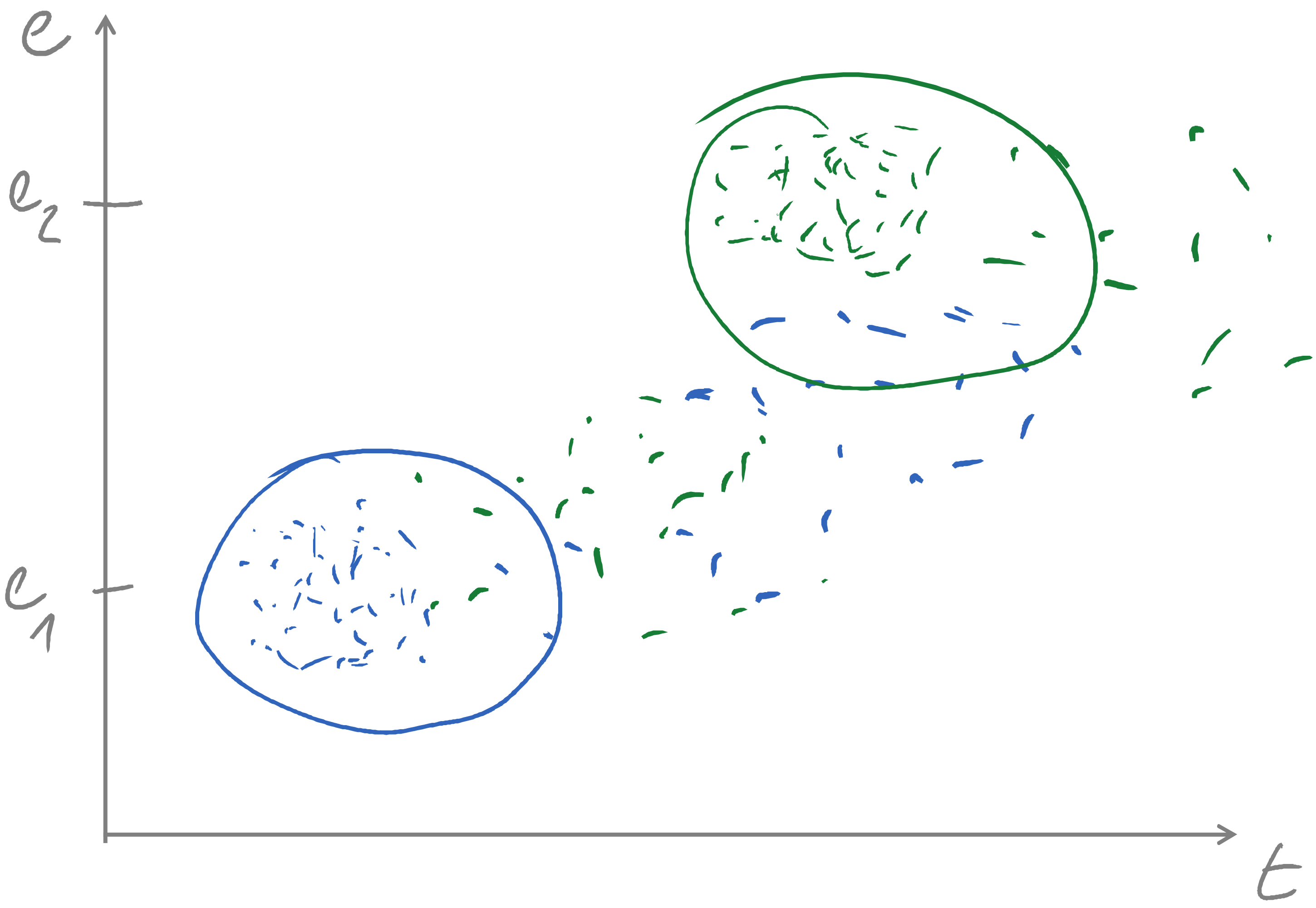




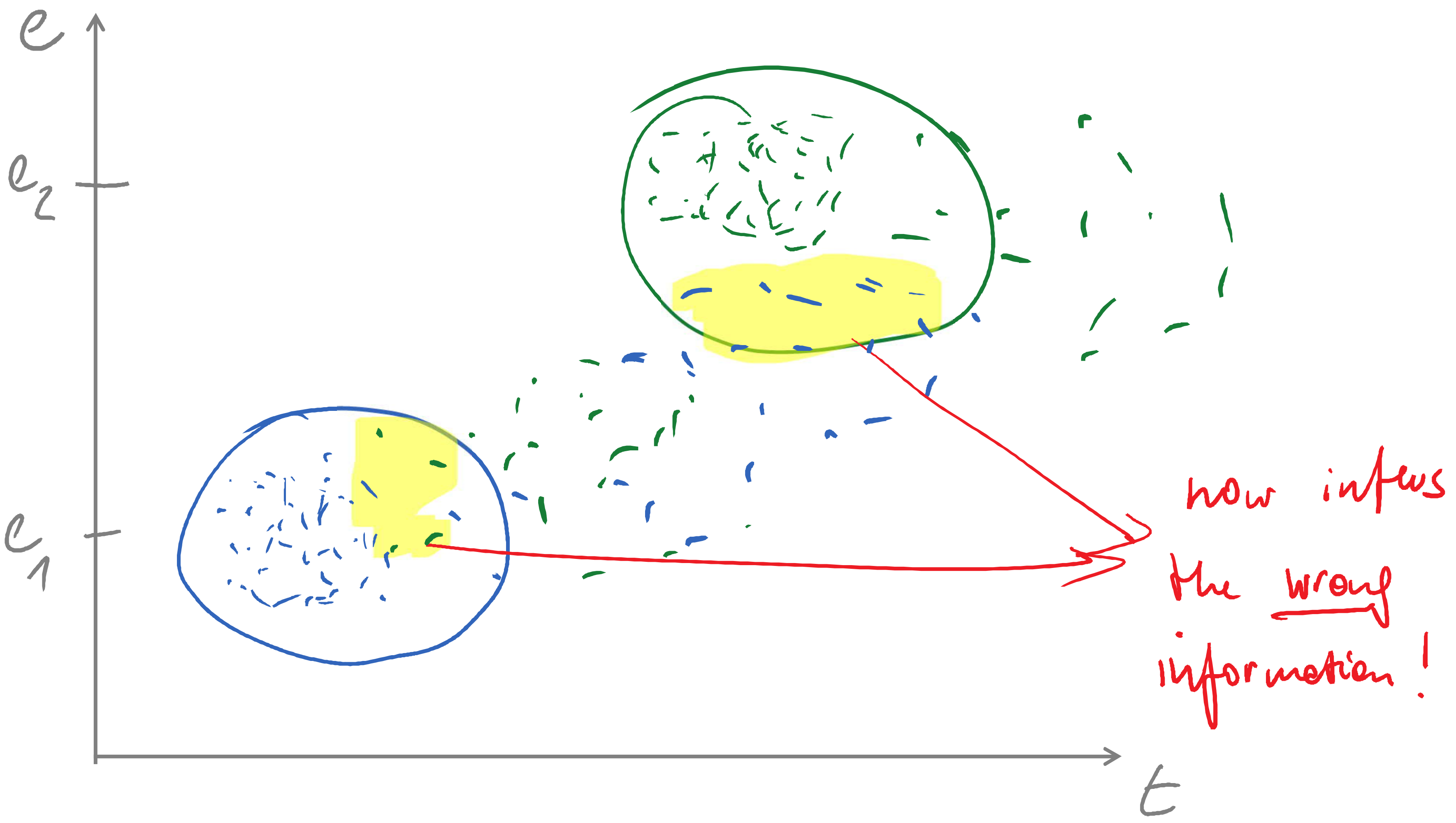
SIDE CHANNELS 101



Side Channels 101



SIDE CHANNELS 101



SW Diversity vs Traditional Side Channels

- general approach
- supports many side channels
  - acoustic
  - thermal
  - power

…Spectre, anyone?!?!

SPECTRE: CONDITIONAL BRANCH MISPREDICTION

a_1 … array N_1

a_2 … array N_2

⋮

```
if ($i < N_1$)
    $y = a_2[a_1[i] * 256]$
```

SPECTRE: CONDITIONAL BRANCH MISPREDICTION

a_1 ... array N_1

a_2 ... array N_2

⋮

```
if (i < N_1)
    y = a_2[a_1[i] * 256]
```

not "executed during speculative execution"

$\Rightarrow$ i can be out-of-bounds (i.e., $i \geq N_1$)

SPECTRE: CONDITIONAL BRANCH MISPREDICTION

a_1 ... array N_1

a_2 ... array N_2

```
⋮
if ($i < N_1$)
    $y = a_2[a_1[i] * 256]$
```

not "executed during speculative execution"

=> i can be out-of-bounds (i.e., $i \geq N_1$)

Q: Can we "force" execution of the bounds check?

SPECTRE: CONDITIONAL BRANCH MISPREDICTION

a_1 ... array N_1

a_2 ... array N_2

⋮

$\text{if } (i < N_1)$

$\quad y = a_2[a_1[i] * 256]$

OBSERVATION RE SPECULATION:

- all instructions will be executed
- branches temporarily "skipped" over

⇒ bounds check needs to be a <u>non-conditional instruction</u>

SPECTRE: CONDITIONAL BRANCH MISPREDICTION

a_1 ... array N_1

a_2 ... array N_2

$\vdots$

if $(i < N_1)$

$i = i \% N_1$

$y = a_2[\underbrace{a_1[i]}_{} * 256]$

=> cannot read out-of-bounds anymore

SPECTRE: CONDITIONAL BRANCH MISPREDICTION

a_1 ... array N_1

a_2 ... array N_2

⋮

```
if ($i < N_1$)
    $i = i \% N_1$
    $y = a_2[\underbrace{a_1[i]} * 256]$
```

=> cannot read out-of-bounds any more

HINTS / QUESTIONS:

- may not work for all C programs in general
- easy to incorporate into a compiler

SPECTRE: CONDITIONAL BRANCH MISPREDICTION

- If % is expensive (cf. example from Chris)

```
for (int i=0; i<10; i++) {
    x = a[i % 10]
    ⋮
}
```

(i) round to nearest power of 2

⇒ (ii) do "postfix" loop unrolling

```
for (i=0; i<8; i++) {
    x = a[i & 8];
    ...
}
x = a[8]; ...
x = a[9]; ...
```

SPECTRE: BRANCH TARGET POISONING

benign case: interpreter instruction dispatch with threaded code

```
opcode_table[256];

LOAD_FAST:
    x = locals[arg];
    PUSH(x);
    goto *opcode_table[ip++];
     ↓
   movq  rax, opcode_table[ip++]
   jmpq  *rax
```

can now jump to any of the 256 addresses stored in opcode_table

Spectre: Branch Target Poisoning

jmpq *rax .... rip instr. pointer register

in Branch Target Table there is a record of potential target addresses ℓ_i indexed per rip location / value.

$$BTT[rip] = \{\ell_0, \dots, \ell_n\}$$

Spectre: Branch Target Poisoning

jmpq *rax .... rip instr. pointer register

in Branch Target Table there is a record of potential target addresses ℓ_i indexed per rip location / value.

$$BTT[rip] = \{\ell_0, \dots, \ell_n\}$$

cannot hold all rip values and not all locations ℓ_i

⤳ similar indexing "optimization" as with cache-associative sets (last 12 bits as index)

SPECTRE: BRANCH TARGET POISONING

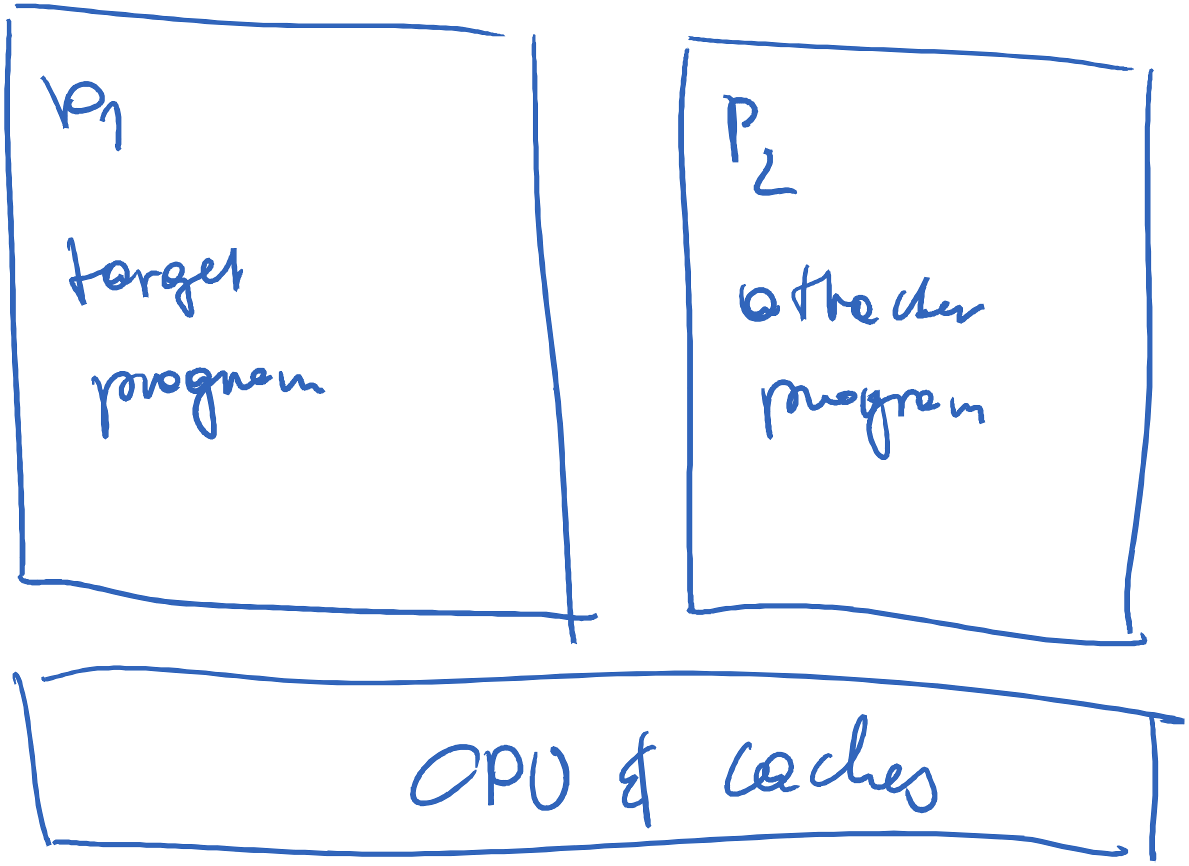
jmpq *rax .... rip instr. pointer register

$BTT[rip] = \{ l_0, \ldots, l_n \}$

KEY IDEA: overwrite all BTT entries with a target address l_a that an attacker chooses and divert execution to that location

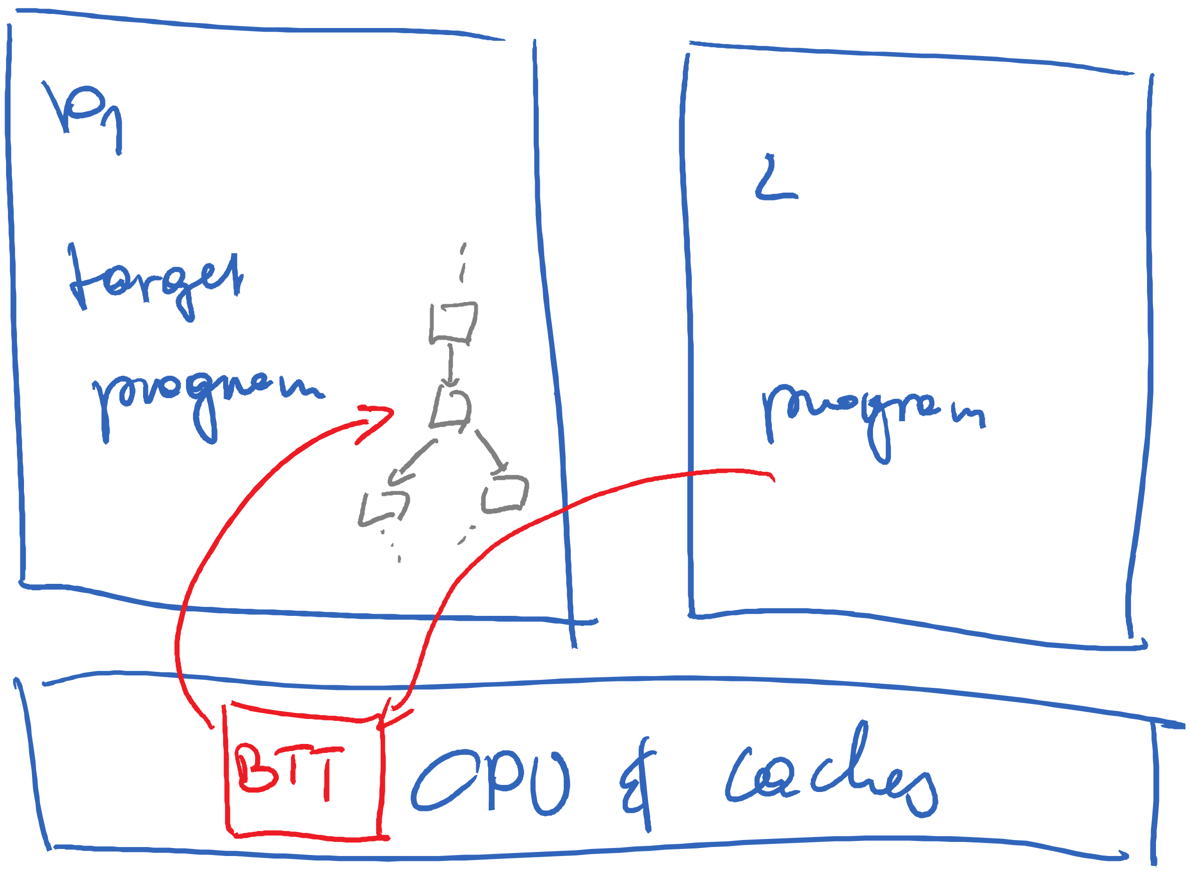
SPECTRE: BRANCH TARGET POISONING

specific attack scenario:



SPECTRE: BRANCH TARGET POISONING

attack scenario:



SPECTRE: BRANCH TARGET POISONING

jmpq *rax .... rip instr. pointer register

$\mathrm{BTT}[\mathrm{rip}] = \{ l_0, \ldots, l_n \}$

1. create program with an indirect branch instruction that "masks" the target ind. branch instruction's rip (i.e., lower 12 bits are equal)

2. repeatedly branch to maliciously chosen location l_a (i.e., $\mathrm{BTT}[\mathrm{rip}] = \{ l_a, \ldots, l_a \}$)

3. wait

Spectre: Branch Target Poisoning

jmpq *rax .... rip instr. pointer register

$\text{BTT}[\text{rip}] = \{ l_0, \dots, l_n \}$

1. create program with an indirect branch instruction that "masks" the target ind. branch instruction's rip (i.e., lower 12 bits are equal)
2. repeatedly branch to maliciously chosen location l_a (i.e., $\text{BTT}[\text{rip}] = \{ l_a, \dots, l_a \}$)
3. wait

Q: Why does this attack work?

Spectre: Branch Target Poisoning

jmpq *rax ..... rip instr. pointer register

$$BTT[rip] = \{l_0, \dots, l_n\}$$

1. create program with an indirect branch instruction that "masks" the target ind. branch instruction's rip (i.e., lower 12 bits are equal)

2. repeatedly branch to maliciously chosen location l_a (i.e., $BTT[rip] = \{l_a, \dots, l_a\}$)

3. wait

Q: Why does this attack work?

Spectre: Branch Target Poisoning

jmpq *rax .... rip instr. pointer register

$$BTT[rip] = \{ l_0, \ldots, l_n \}$$

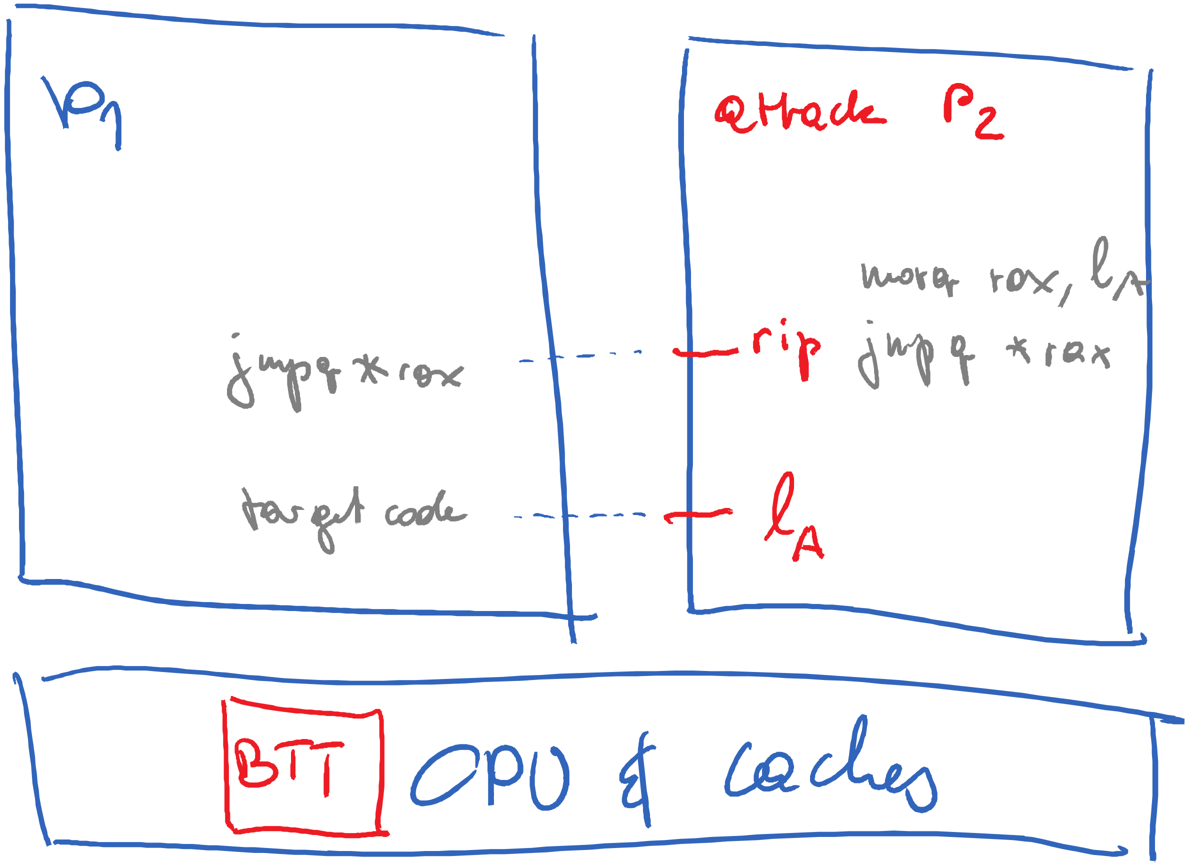
1. create program with an indirect branch instruction that "masks" the target ind. branch instruction's rip (i.e., lower 12 bits are equal)

⇒ attacker needs to have precise a priori knowledge of rip value, surgical attack

Q: Why does this attack work?

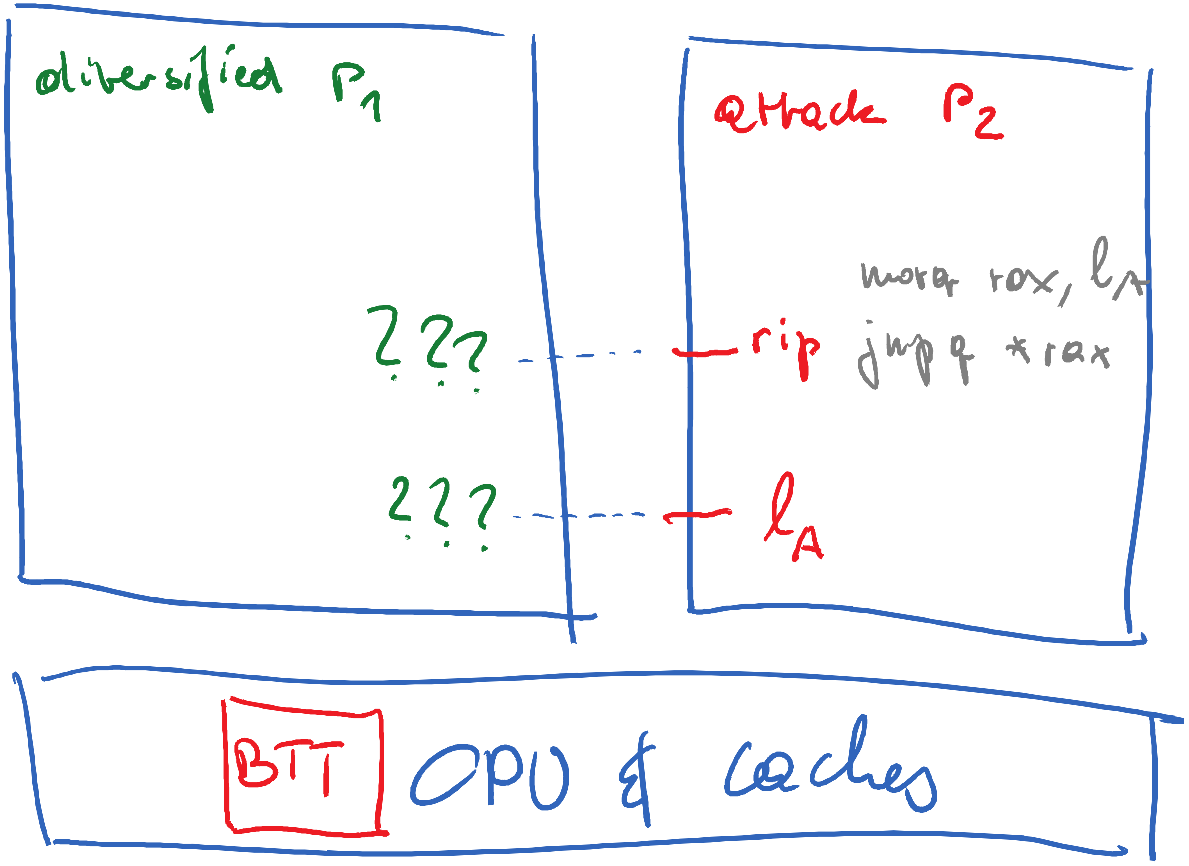
SPECTRE: BRANCH TARGET POISONING

attack scenario:



Spectre: Branch Target Poisoning

attack scenario:



SUMMARY

- probabilistic protection
- not exclusive
  - mutually beneficial with other defenses
- free & immediate protection
- compatible with basically all software techniques (incl. JIT compilers)
- scales to complex real-world software (browsers)
- does not require precise static analysis information
- no formal guarantees, yet
  - need certification in safety-critical contexts