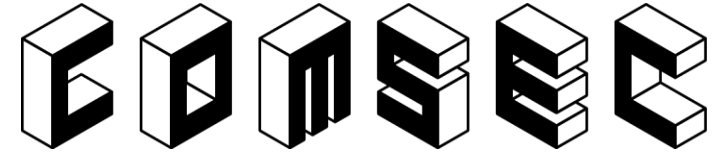


# Cascade



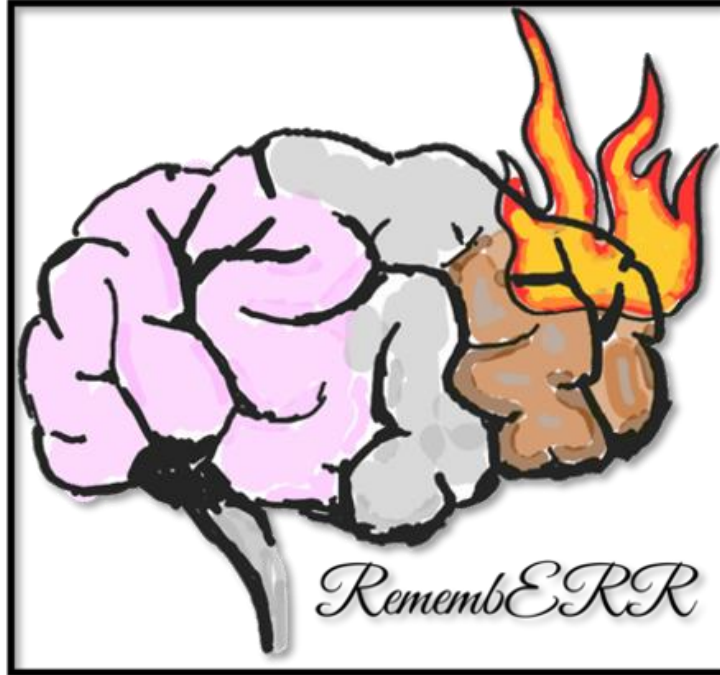
ETH zürich



## CPU Fuzzing via Intricate Program Generation

Flavien Solt, Katharina Ceesay-Seitz and Kaveh Razavi

ETH Zürich



[1]



CPUs are certainly still full of bugs, with potential security implications

# Cascade

More **new CVEs** than all previous  
CPU fuzzers combined

# Cascade

More **new CVEs** than all previous  
CPU fuzzers combined

Outperforms SoA **coverage**  
(despite being black-box)

# Cascade

More new CVEs than all previous CPU fuzzers combined

Outperforms SoA coverage (despite being black-box)



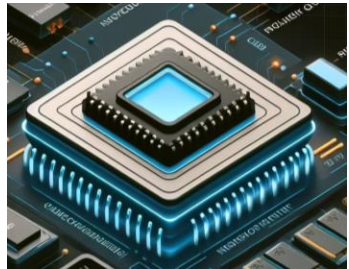
**Idea:** Explicitly generate long, complex and valid programs.



# Software vs. CPU fuzzing



Software  
under test



CPU  
under test

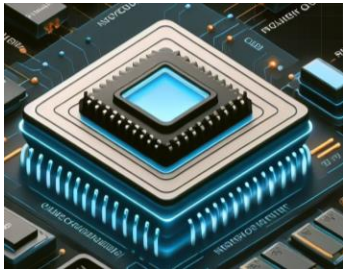
# Software vs. CPU fuzzing



Generic  
data



Software  
under test



CPU  
under test



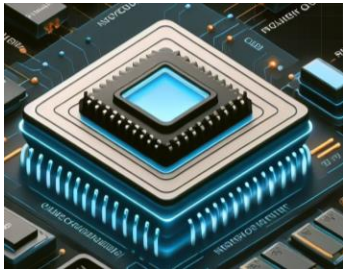
# Software vs. CPU fuzzing



Generic data



Software under test



Structured program



CPU under test



# Software vs. CPU fuzzing



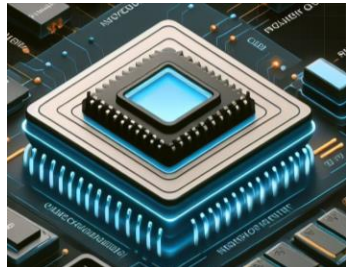
Generic data



Software under test



Crashes

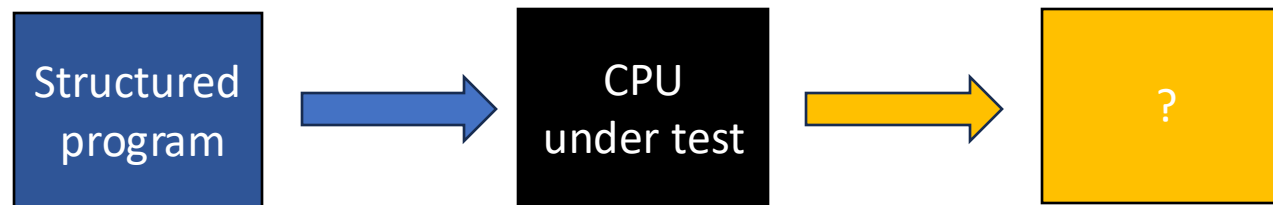
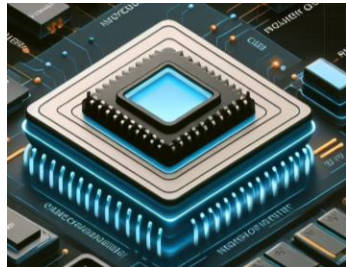
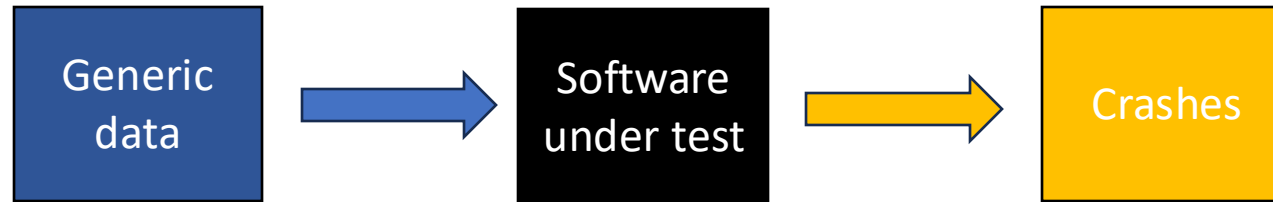


Structured program



CPU under test

# Software vs. CPU fuzzing



Differential fuzzing



ISA Simulator

CPU under test



Identical behavior?

# SoA CPU fuzzers

(DifuzzRTL family)

By definition, CPU inputs are programs.



J. Hur et al., "DifuzzRTL: Differential Fuzz Testing to Find CPU Bugs", S&P '21

# SoA CPU fuzzers

(DifuzzRTL family)



Initialization

Program

# SoA CPU fuzzers

(DifuzzRTL family)



J. Hur et al., "DifuzzRTL: Differential Fuzz Testing to Find CPU Bugs", S&P '21



Initialization

Program

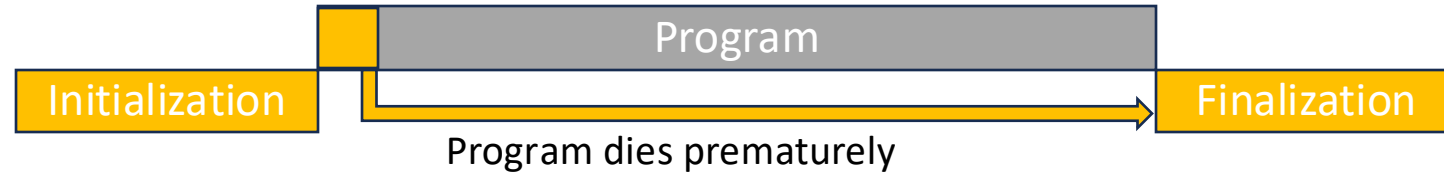
# SoA CPU fuzzers

(DifuzzRTL family)



# SoA CPU fuzzers

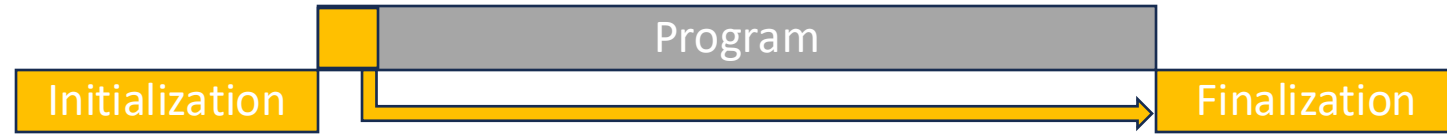
(DifuzzRTL family)



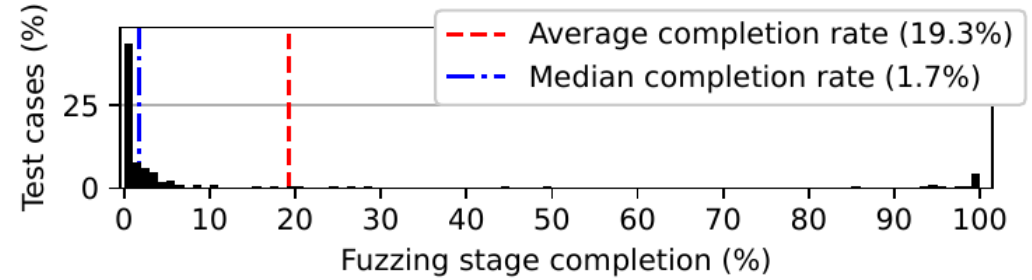


# SoA CPU fuzzers

(DifuzzRTL family)

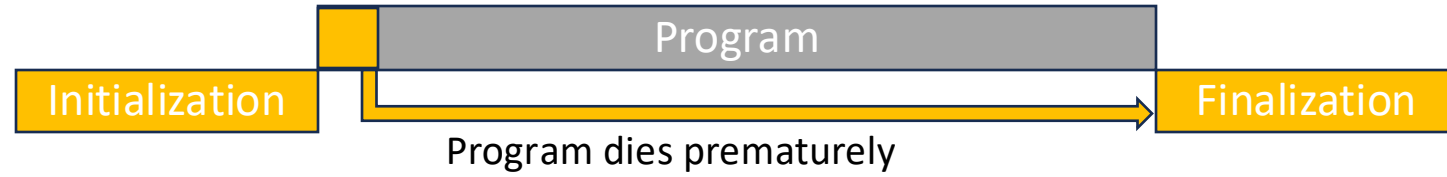


Test cases are almost always broken

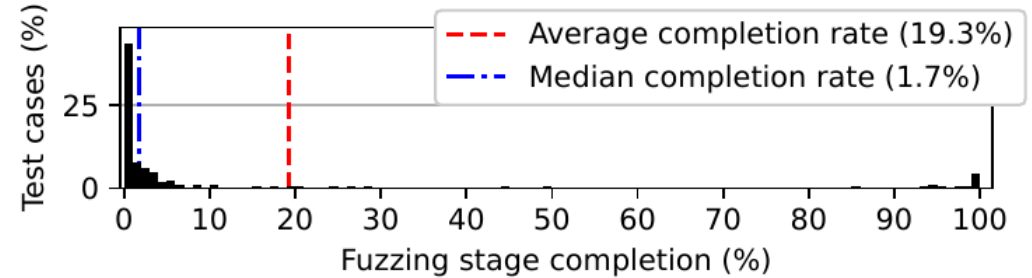


# SoA CPU fuzzers

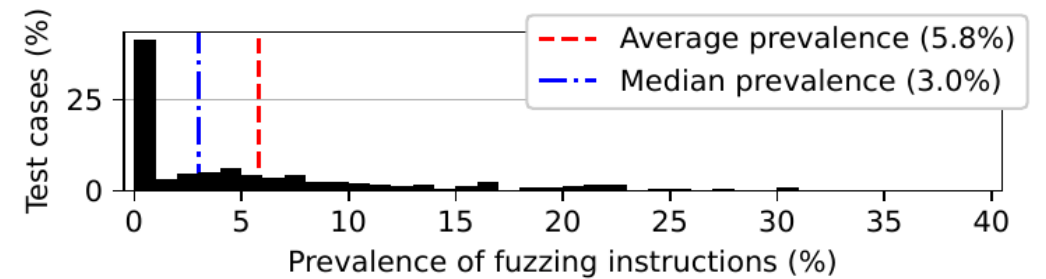
(DifuzzRTL family)



Test cases are almost always broken

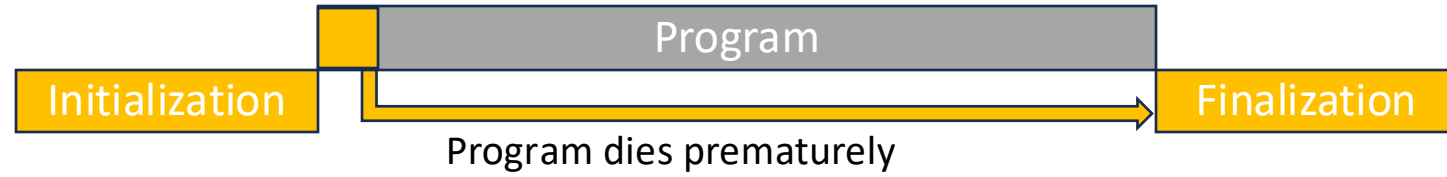


**Problem 1:** Overrepresentation of always the same **instruction snippets**

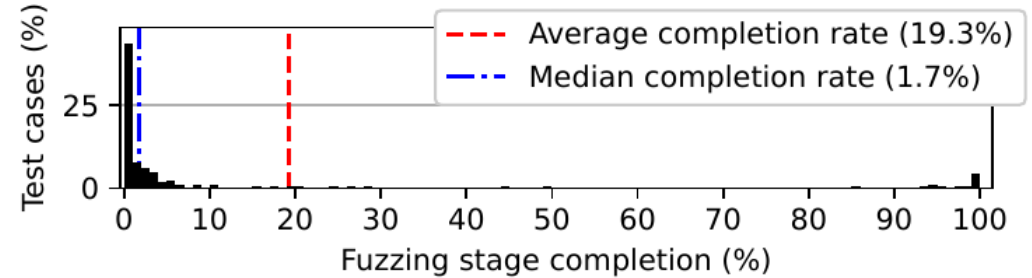


# SoA CPU fuzzers

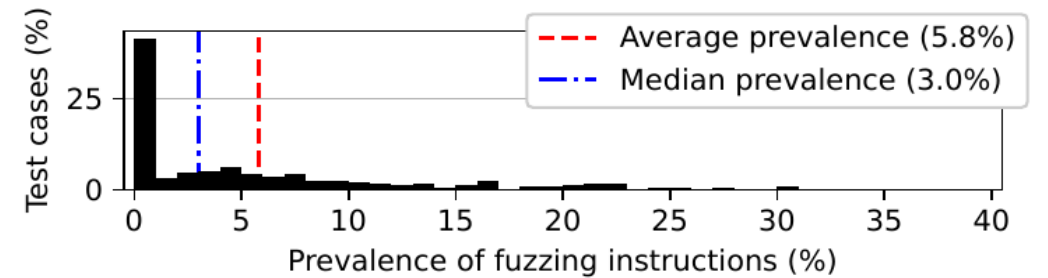
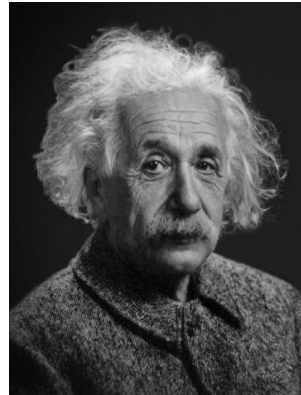
(DifuzzRTL family)



Test cases are almost always broken

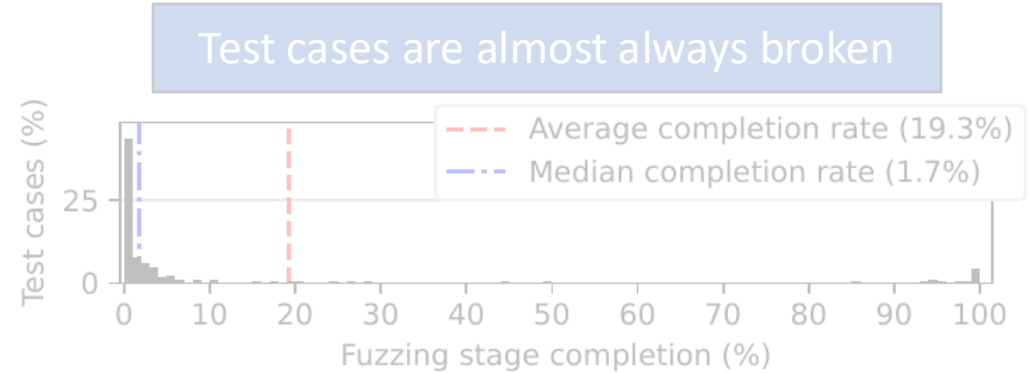
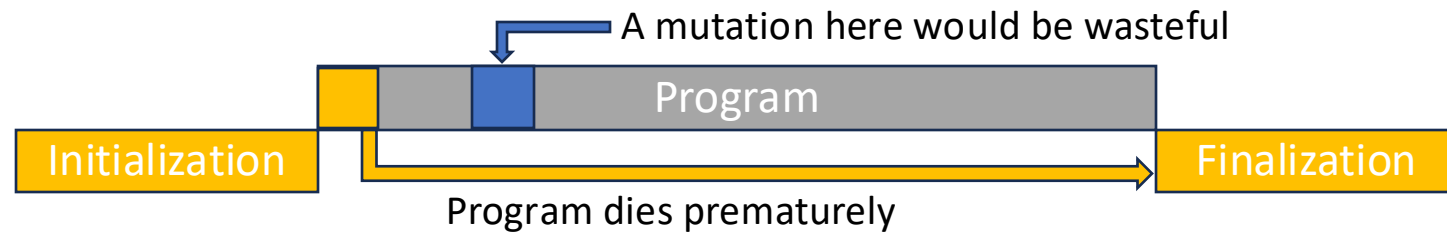


**Problem 1:** Overrepresentation of always the same **instruction snippets**

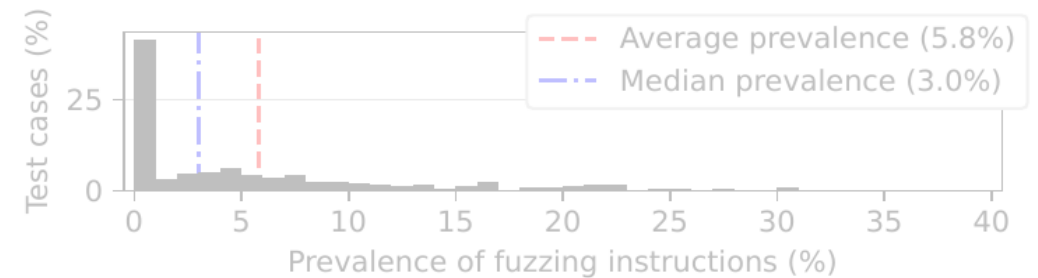


# SoA CPU fuzzers

(DifuzzRTL family)



**Problem 1:** Overrepresentation of always the same **instruction snippets**

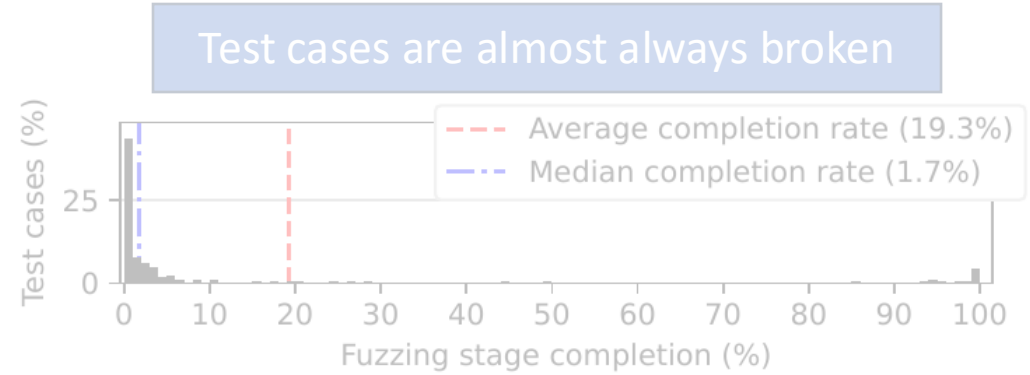
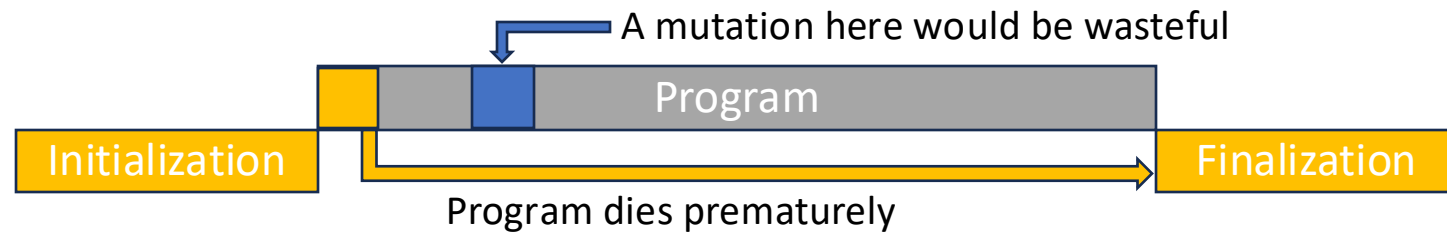


**Problem 2:** Uncertain effect of **mutations**

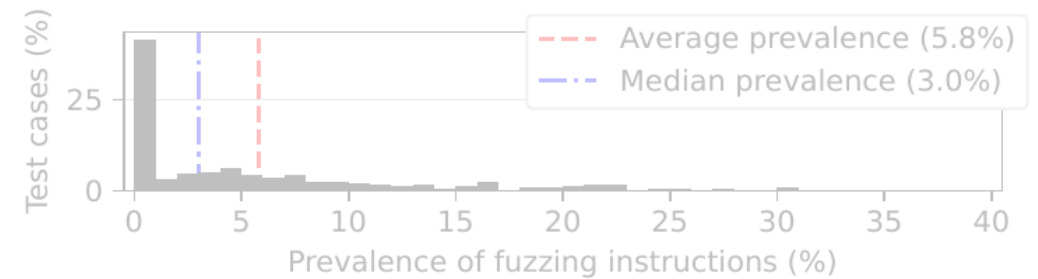


# SoA CPU fuzzers

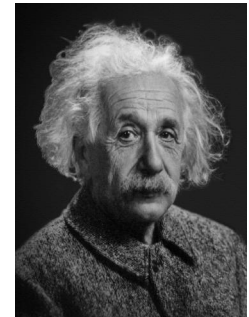
(DifuzzRTL family)



**Problem 1:** Overrepresentation of always the same **instruction snippets**

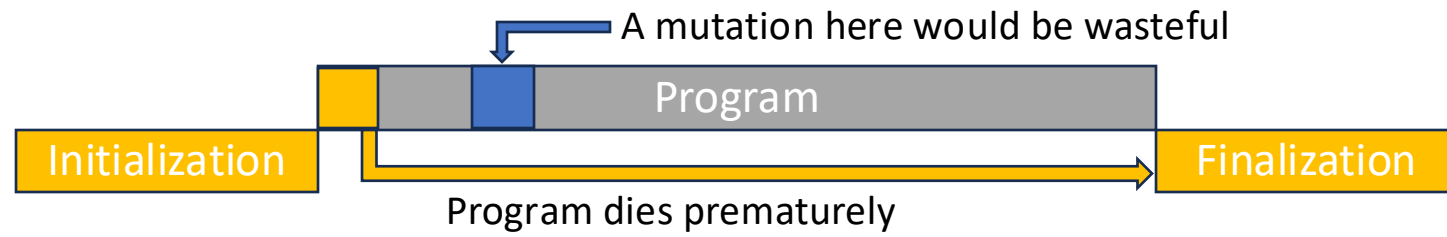


**Problem 2:** Uncertain effect of **mutations**

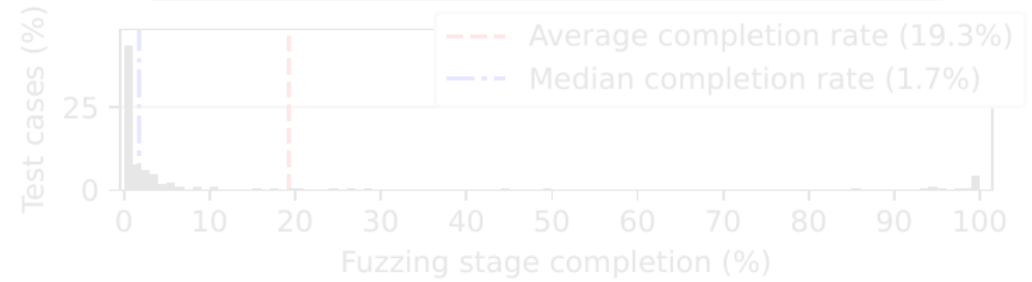


# SoA CPU fuzzers

(DifuzzRTL family)

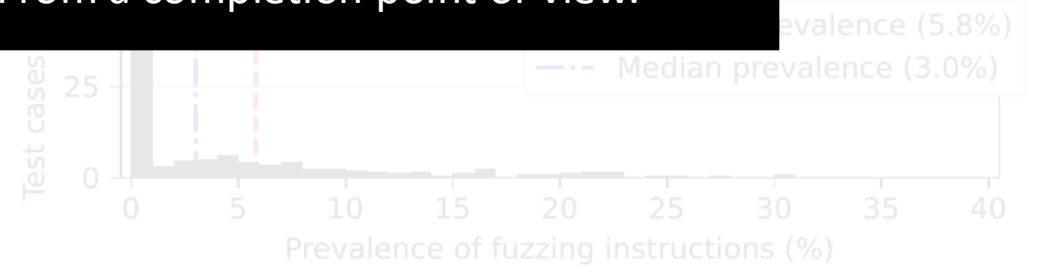


Test cases are almost always broken



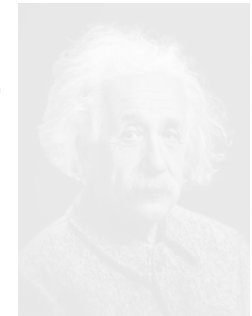
Problem 1: Overrepresentation of always the same

From a completion point of view.

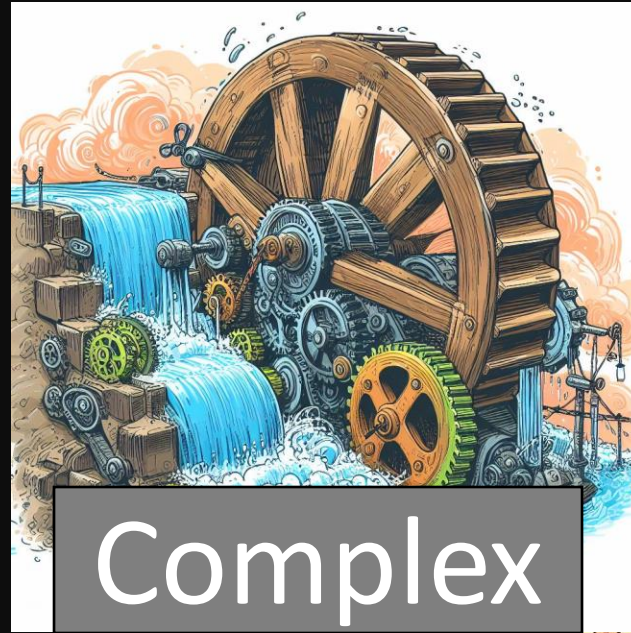


Problem 2: Uncertain effect of mutations

J. Hur et al., "DifuzzRTL: Differential Fuzz Testing to Find CPU Bugs", S&P '21



# Requirements



Complex



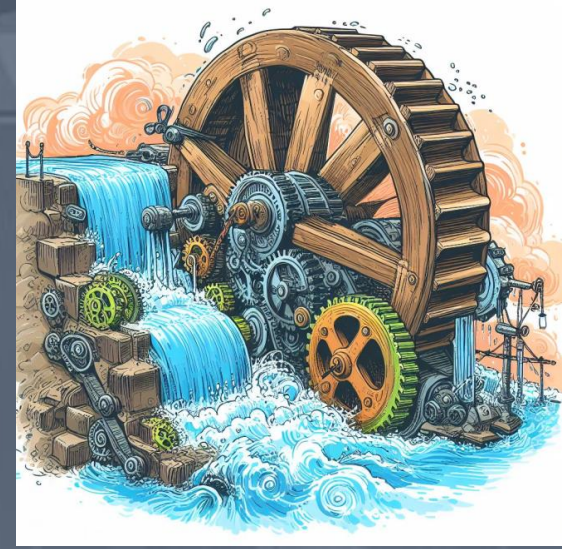
Valid



# Cascade design

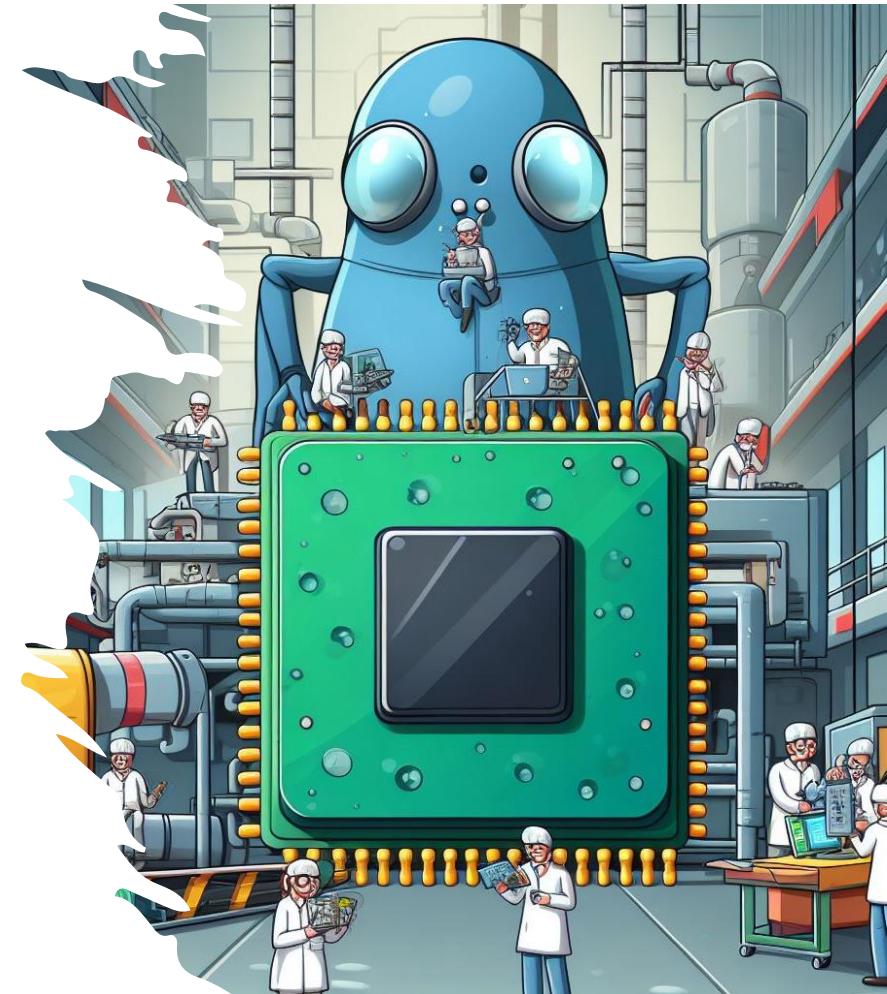
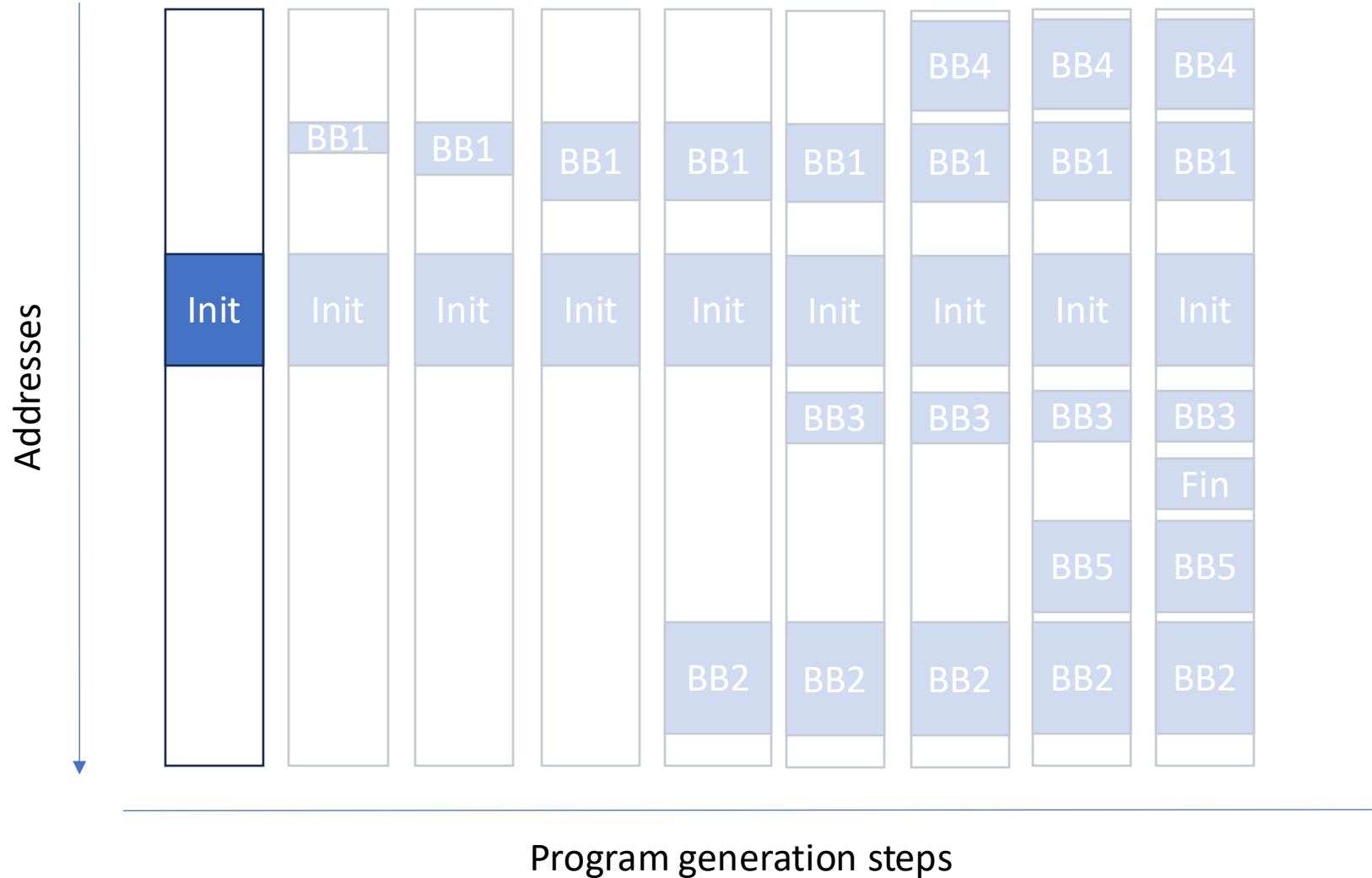


1. Program generation

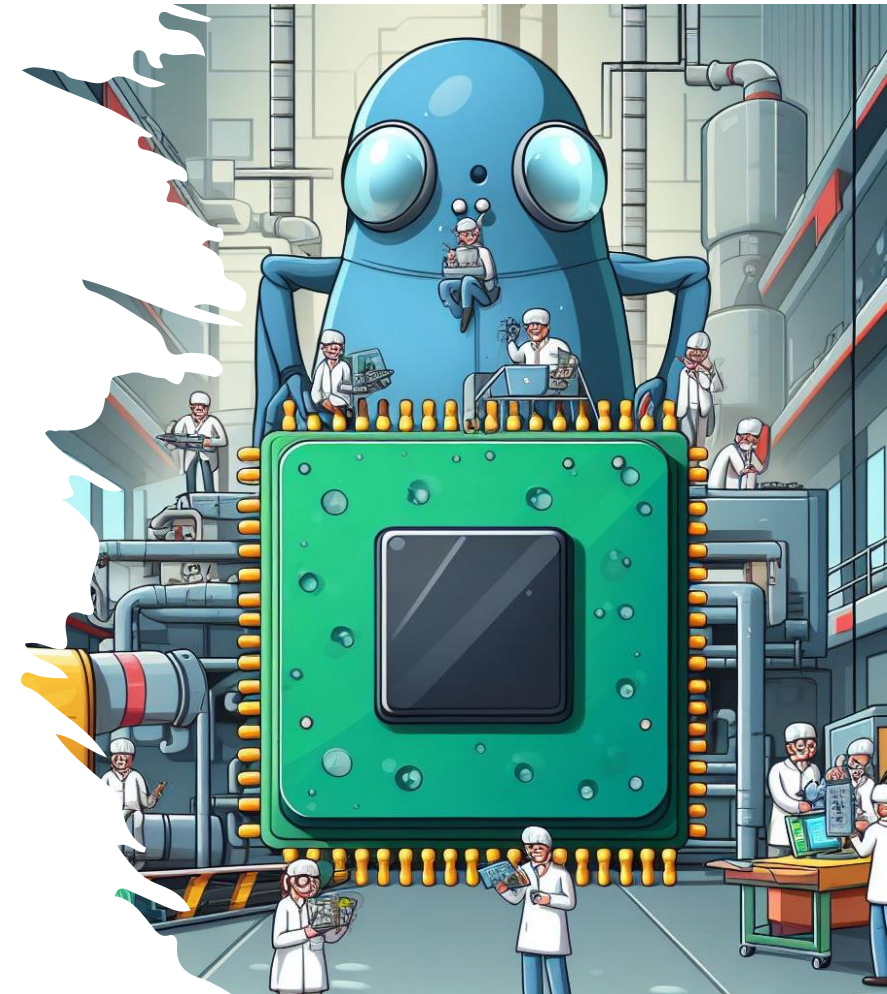
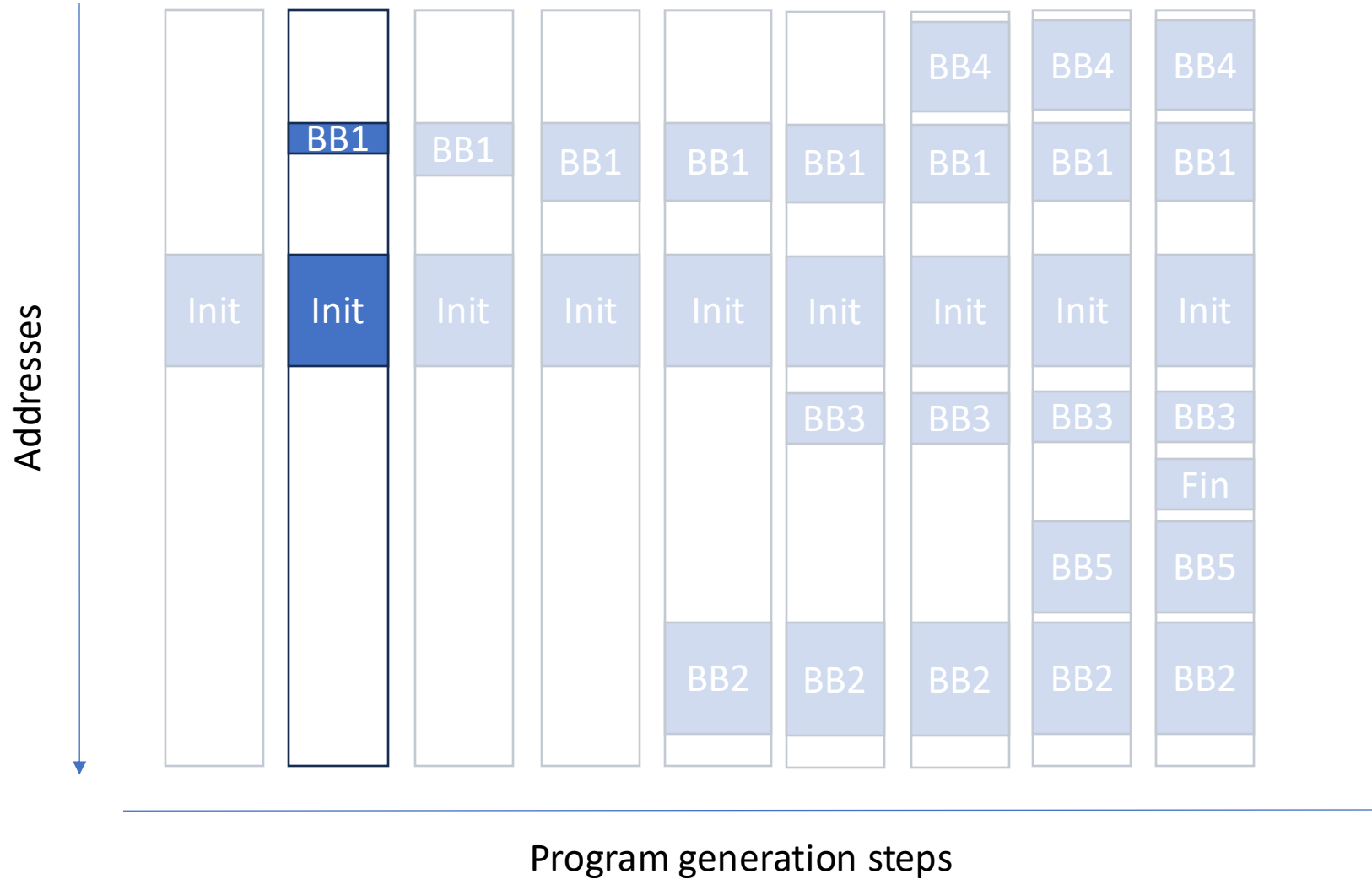


2. Entanglement

# Code generation in Cascade

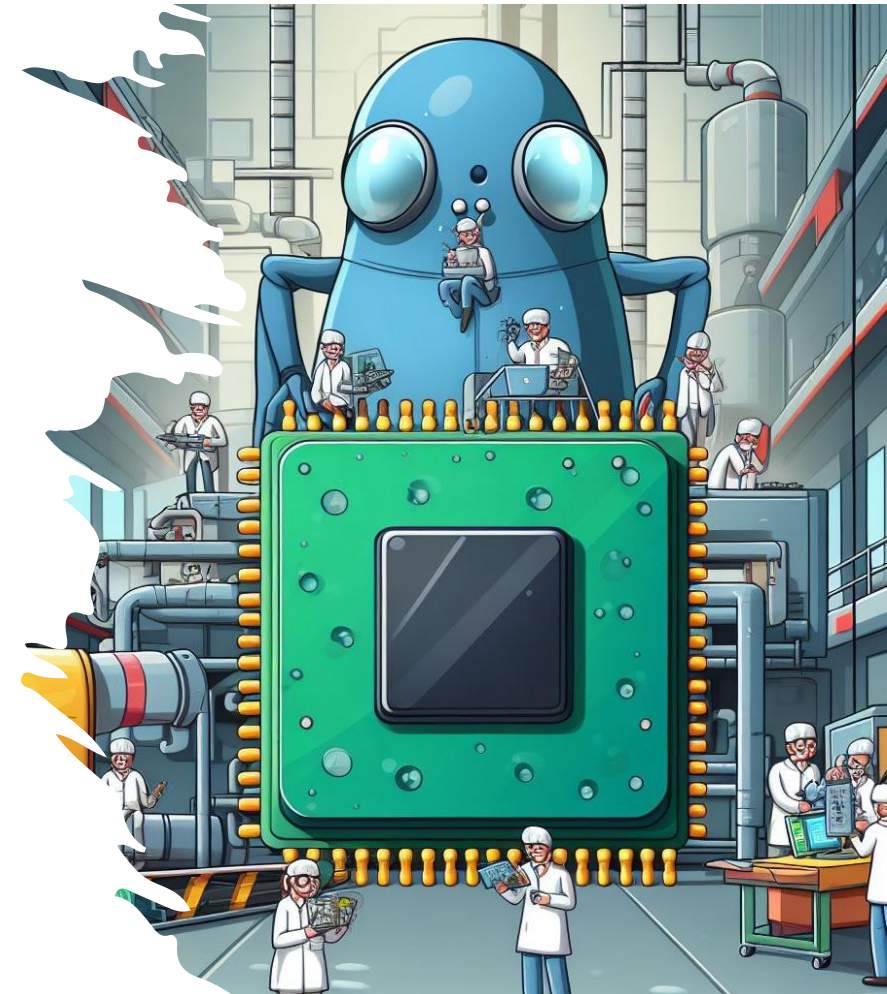
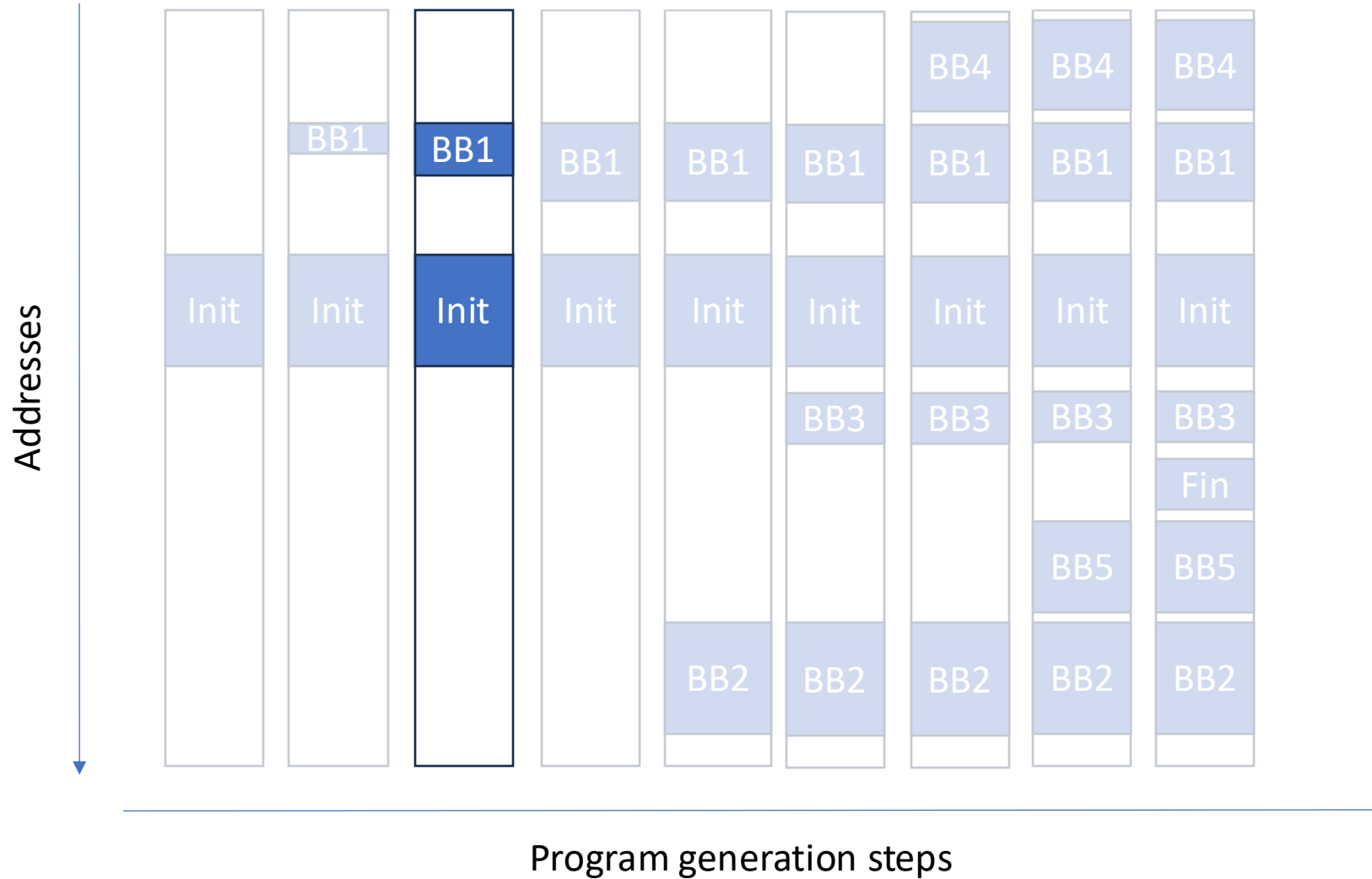


# Code generation in Cascade

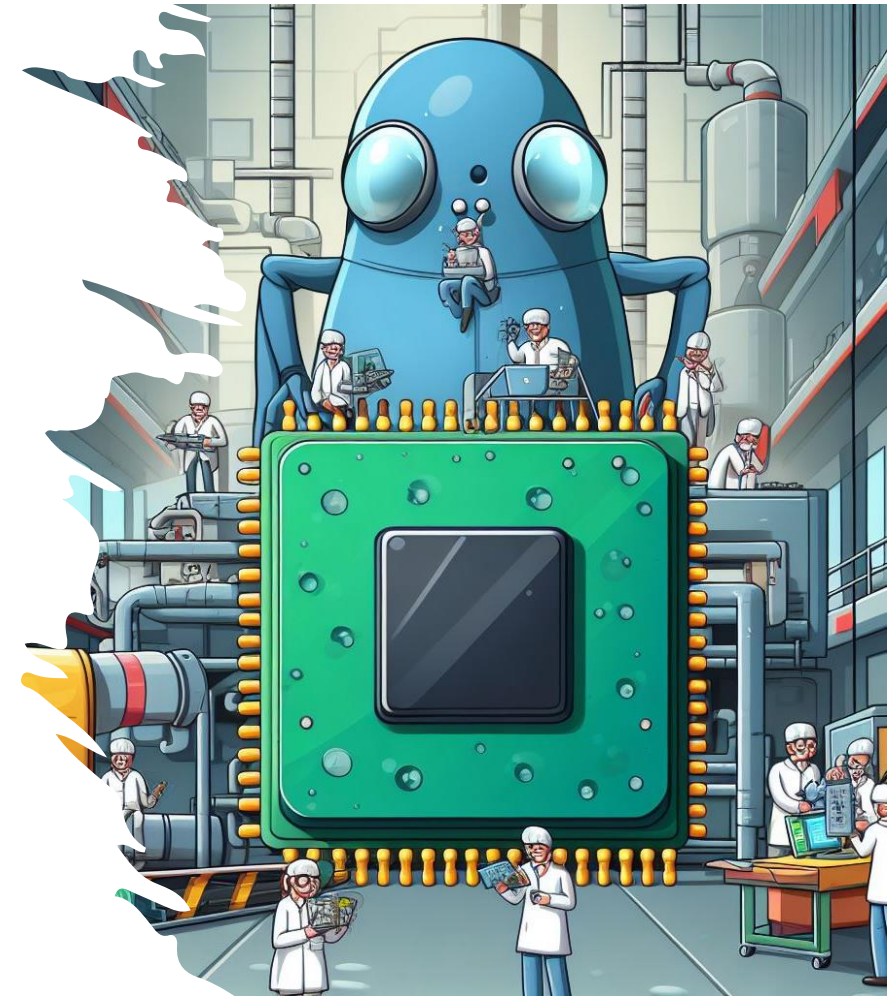
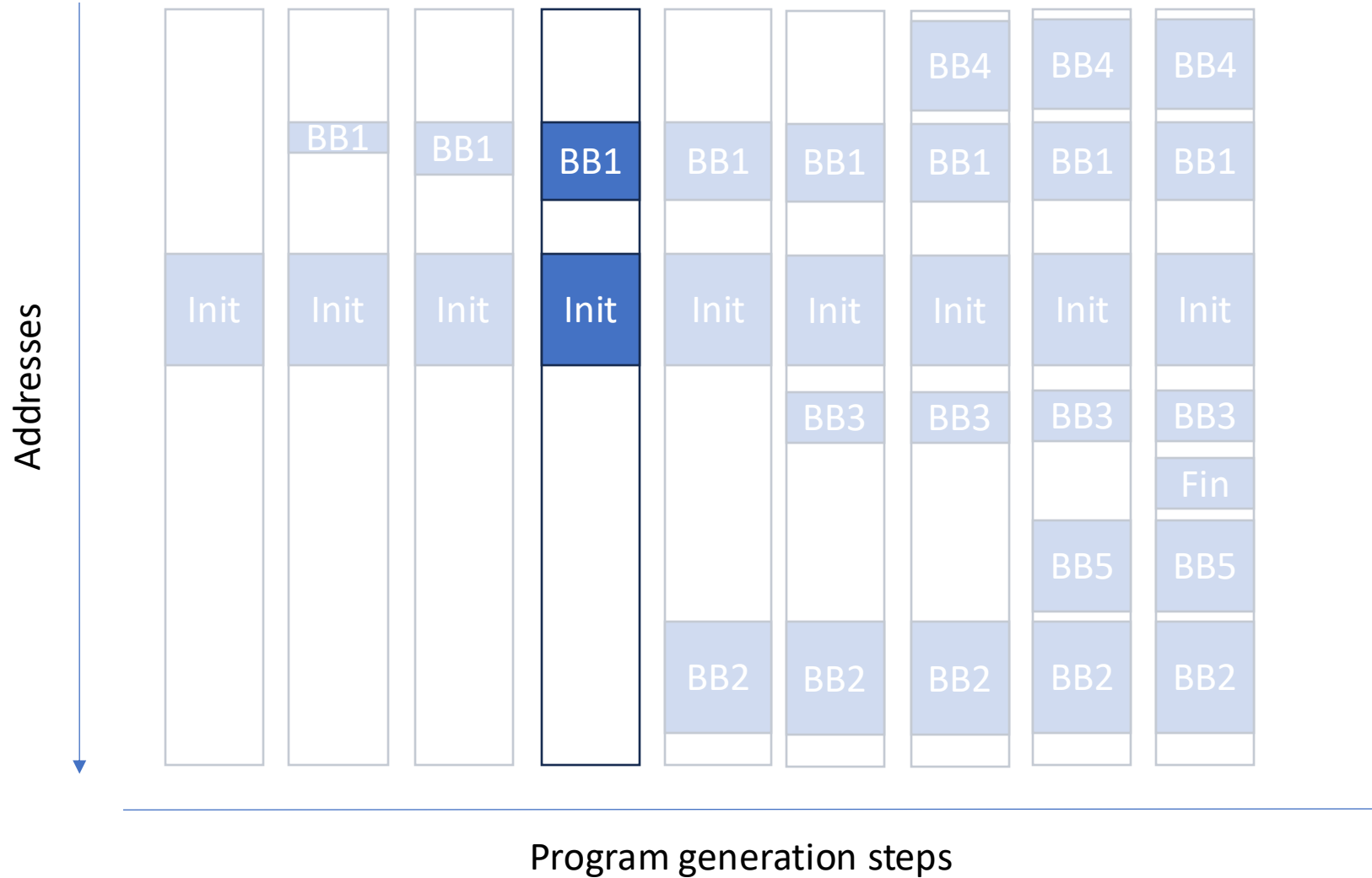




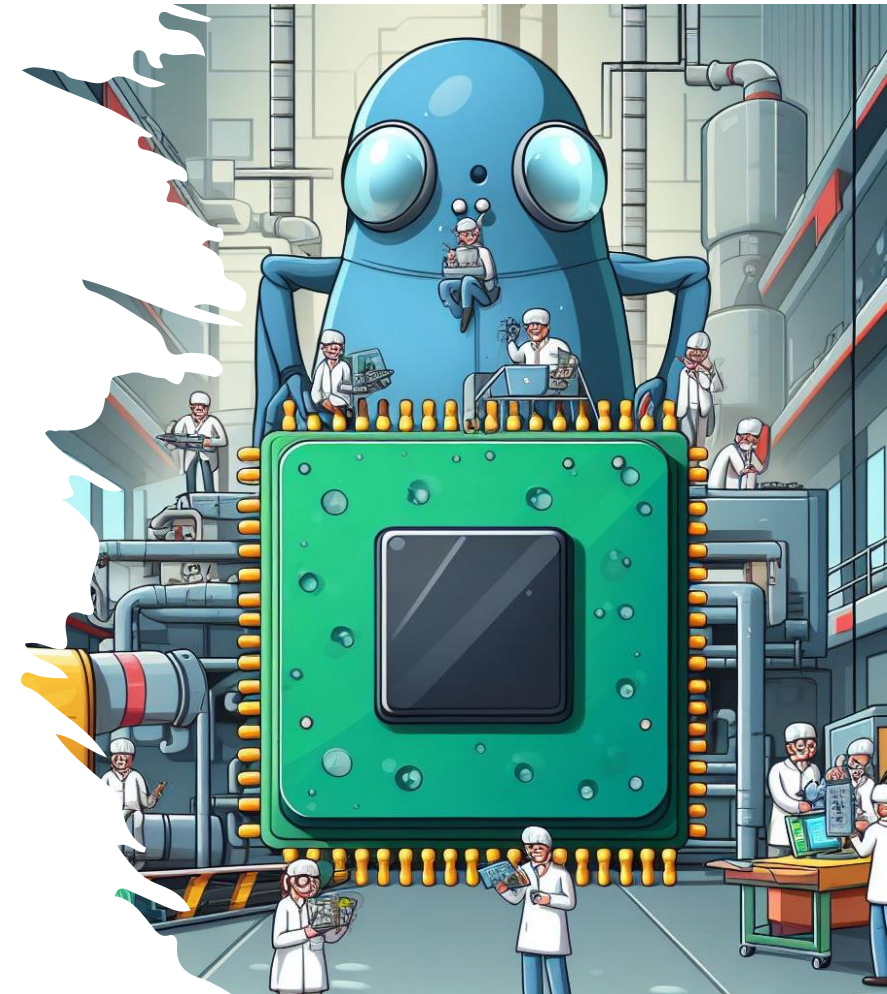
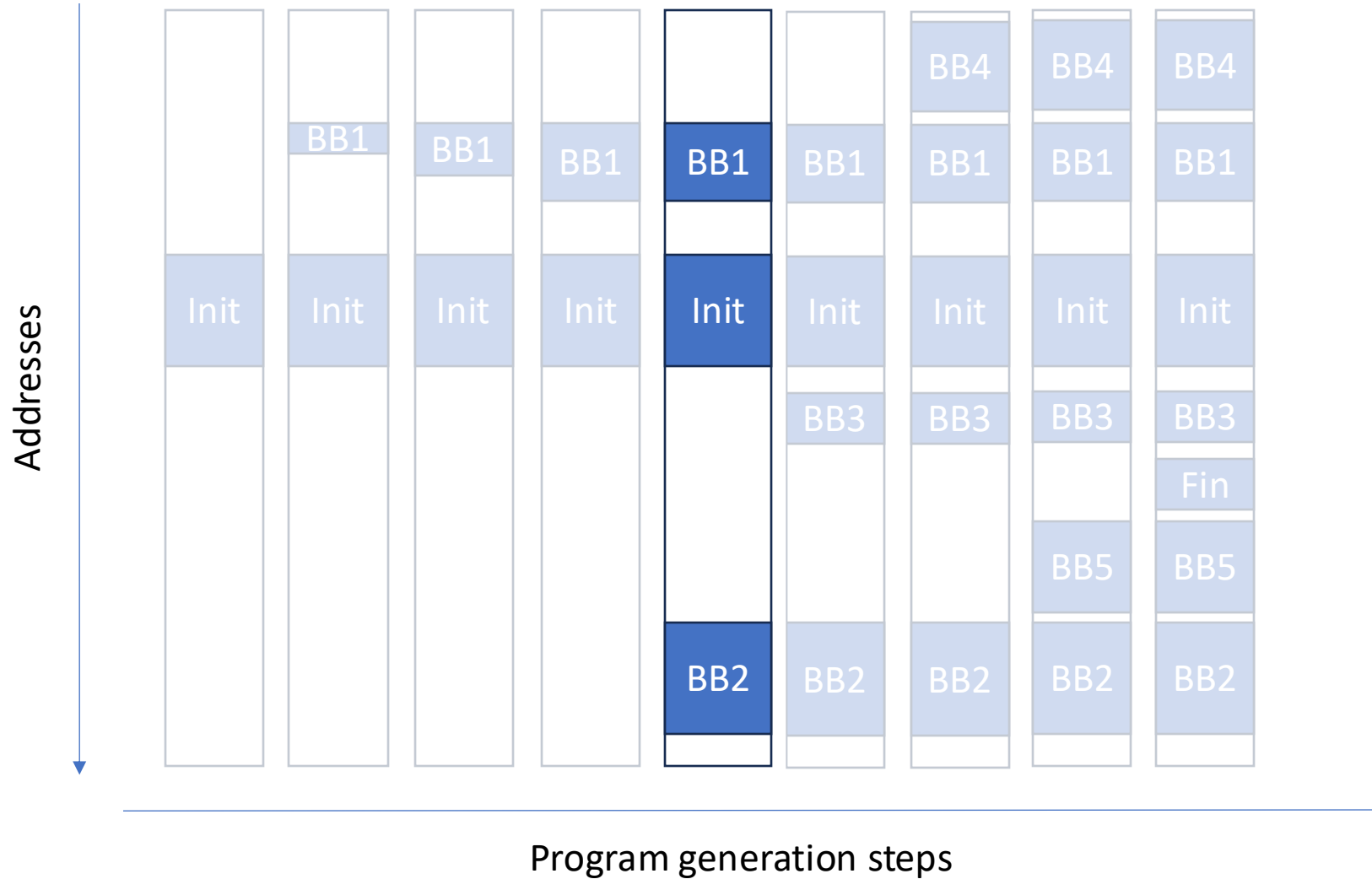
# Code generation in Cascade



# Code generation in Cascade

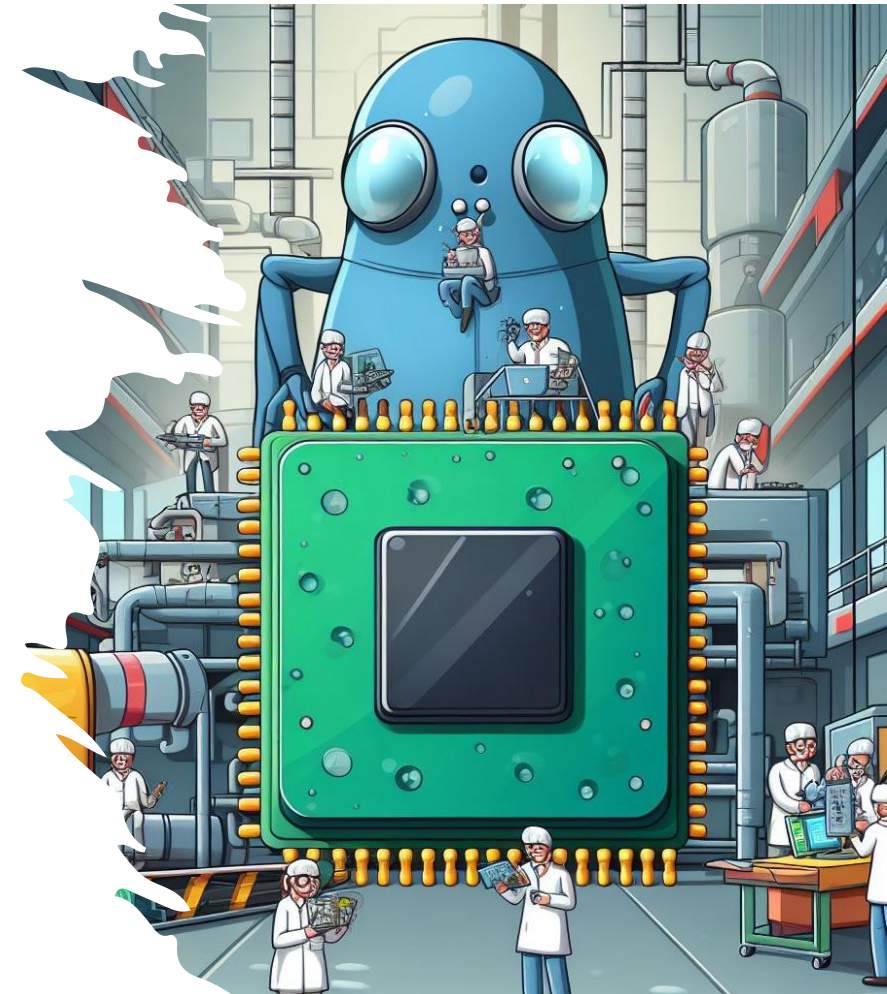
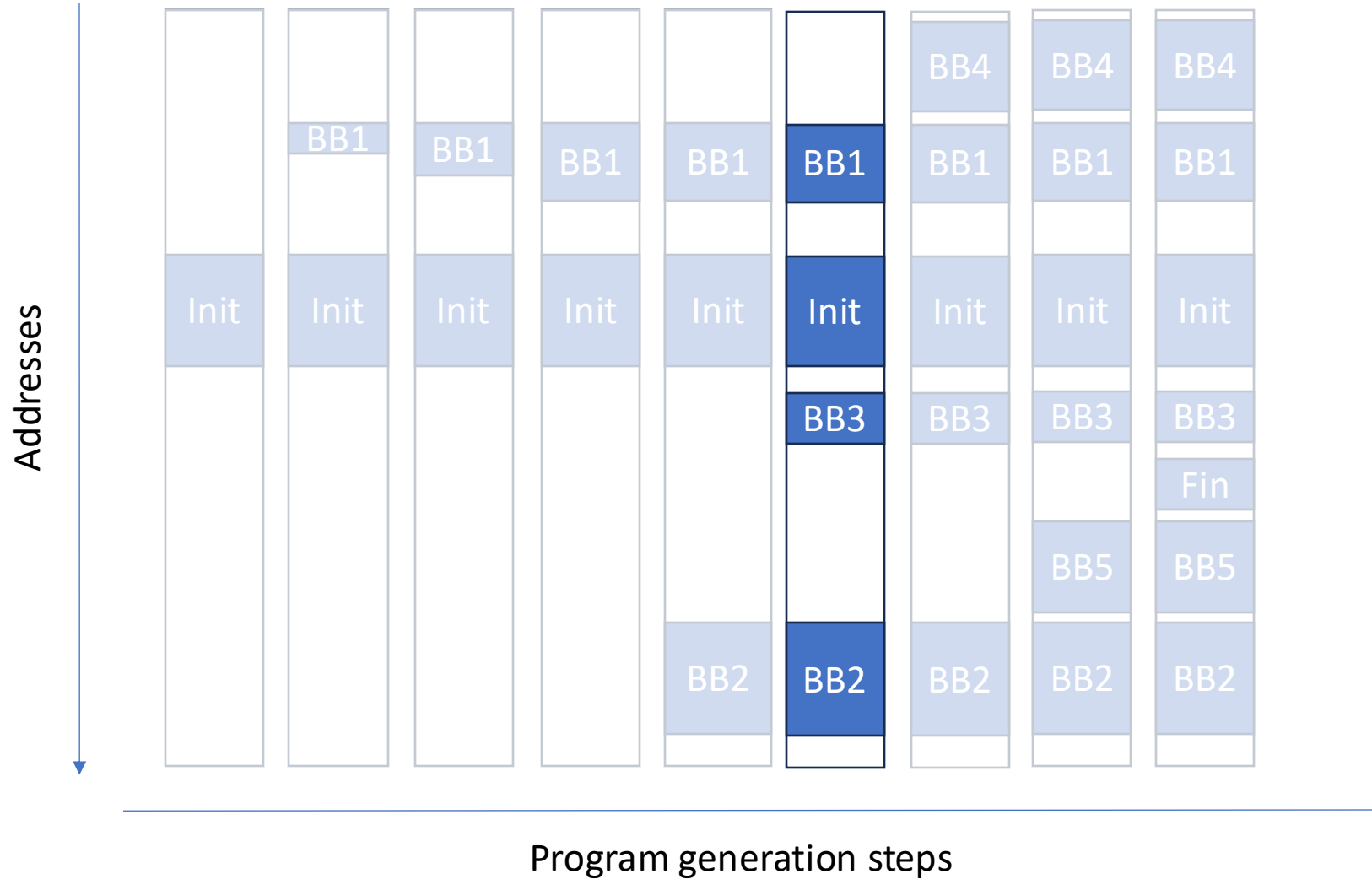


# Code generation in Cascade



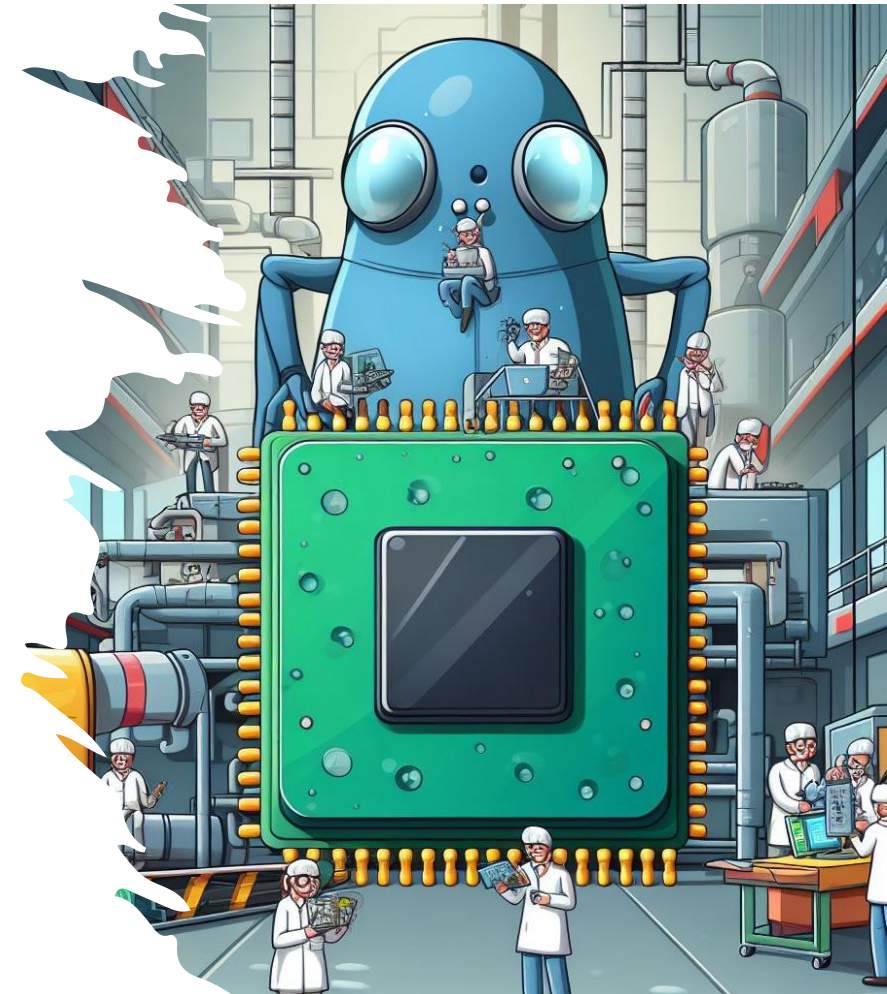
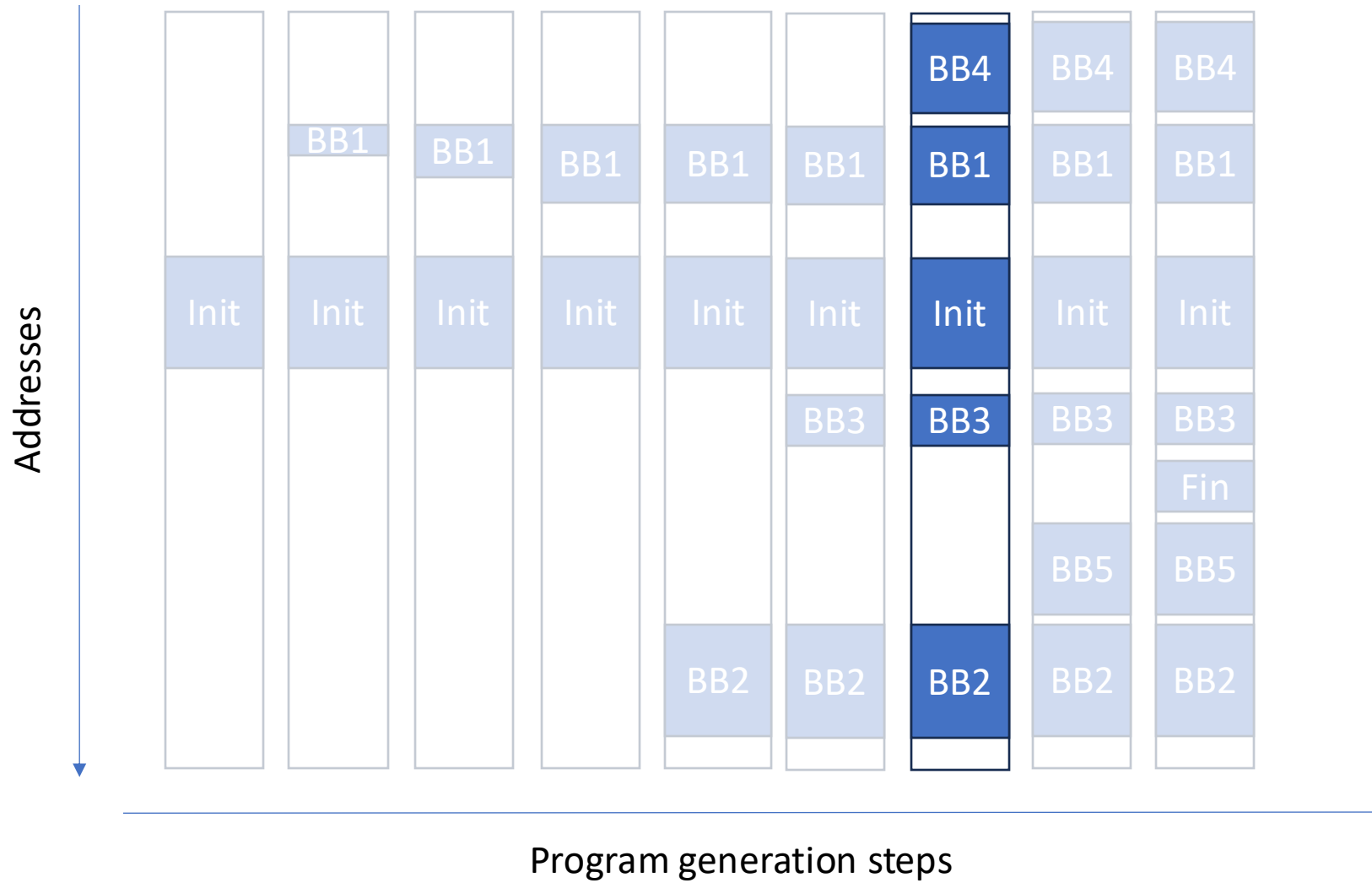


# Code generation in Cascade

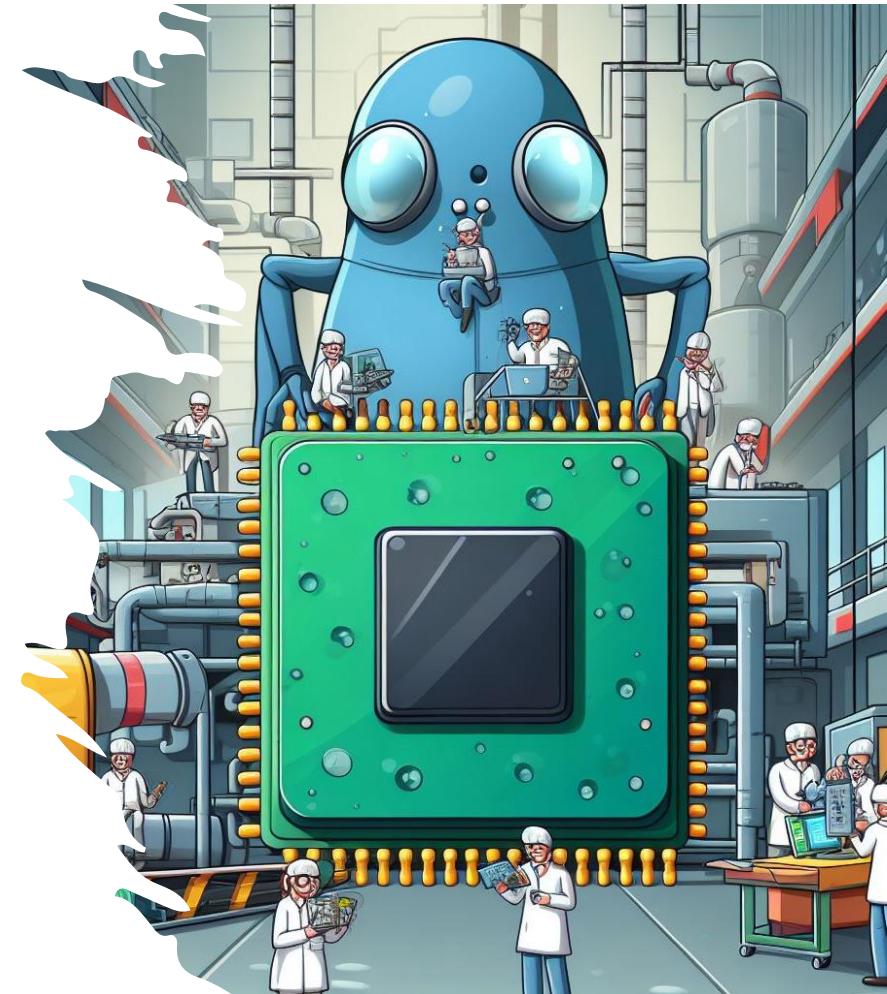
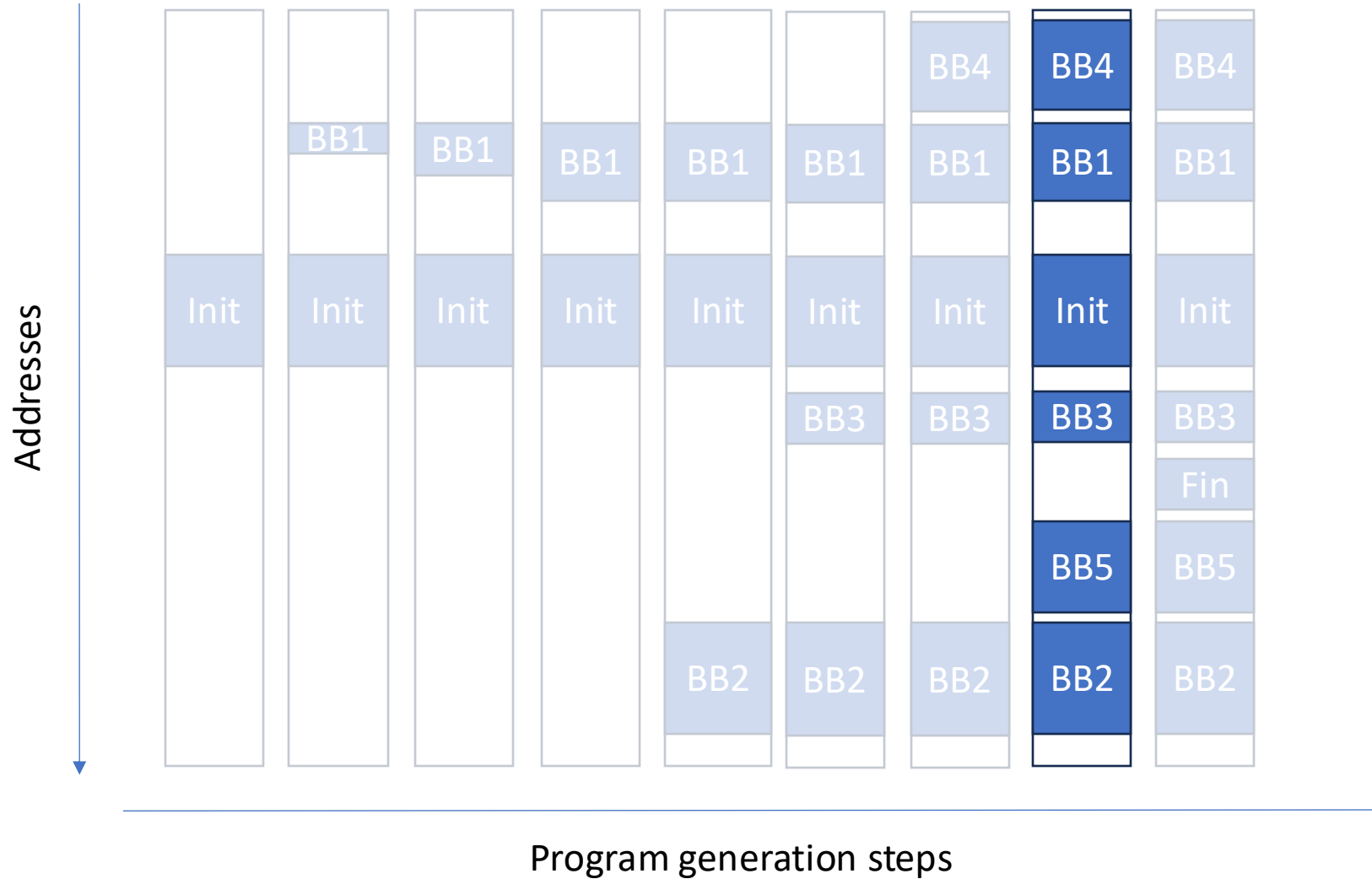




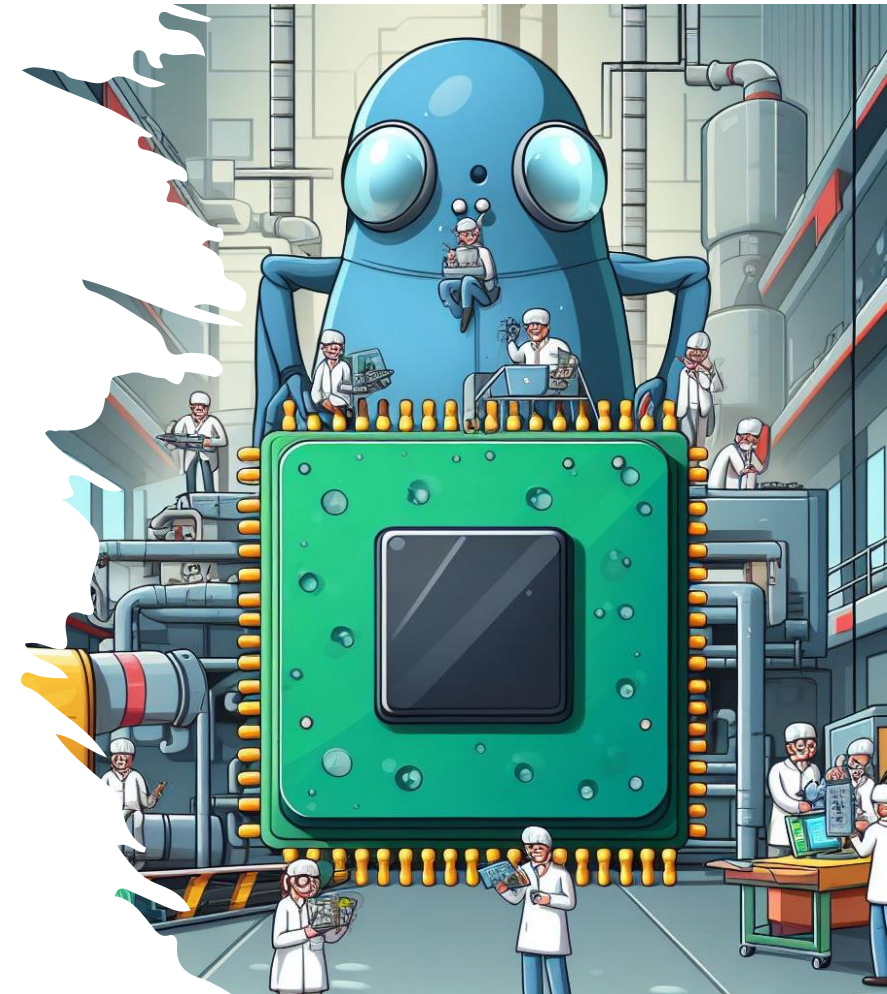
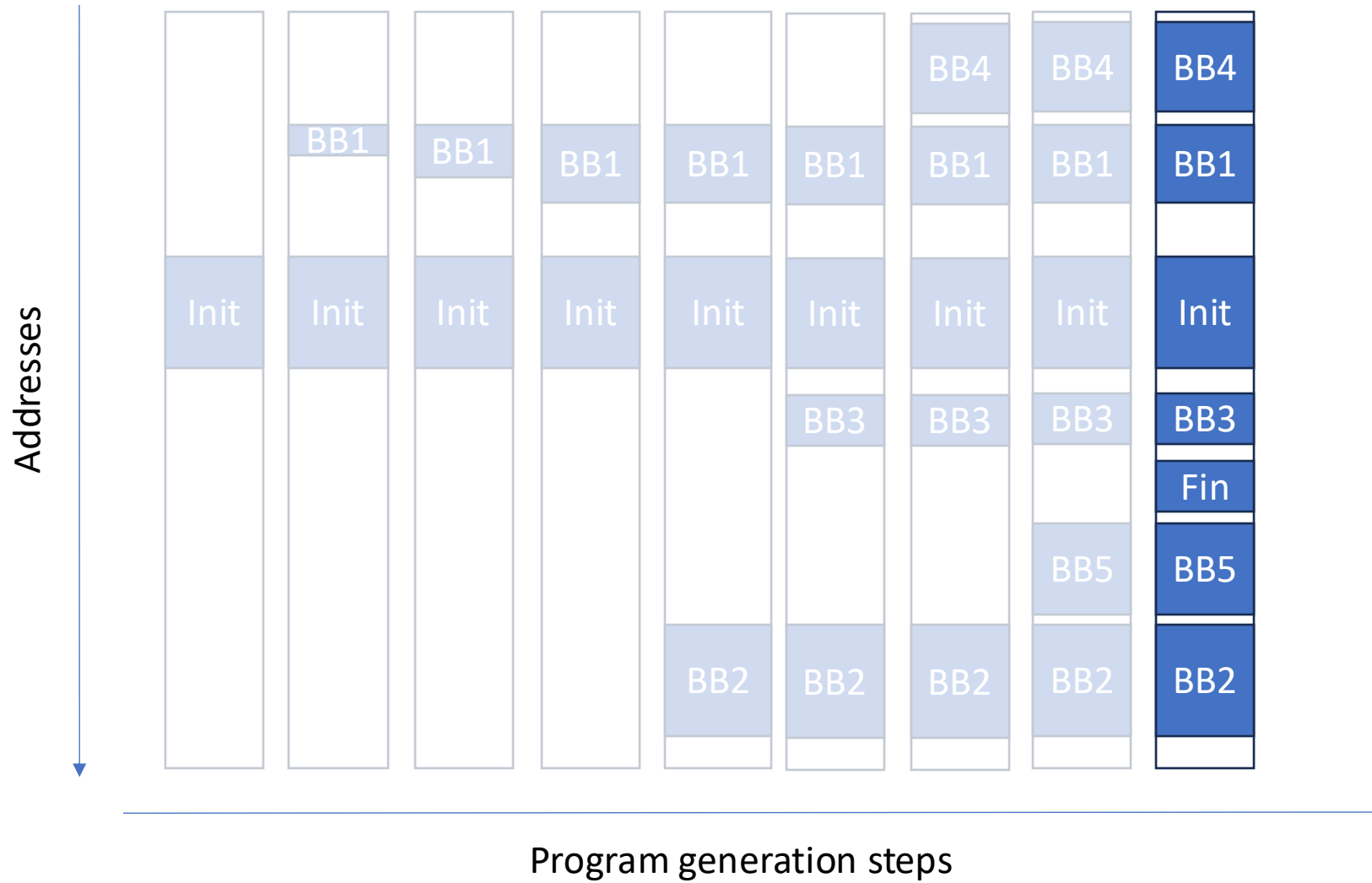
# Code generation in Cascade



# Code generation in Cascade



# Code generation in Cascade



# Code generation in Cascade

```
xor      x7,x4,x9
csrrwi   x9,mcause,15
beq      x9,x4,0x8000098e
fadd     f8,f9,f10
feq.s    x4,f9,f8
jalr     x9,(x7)
```

Intended basic block

# Code generation in Cascade

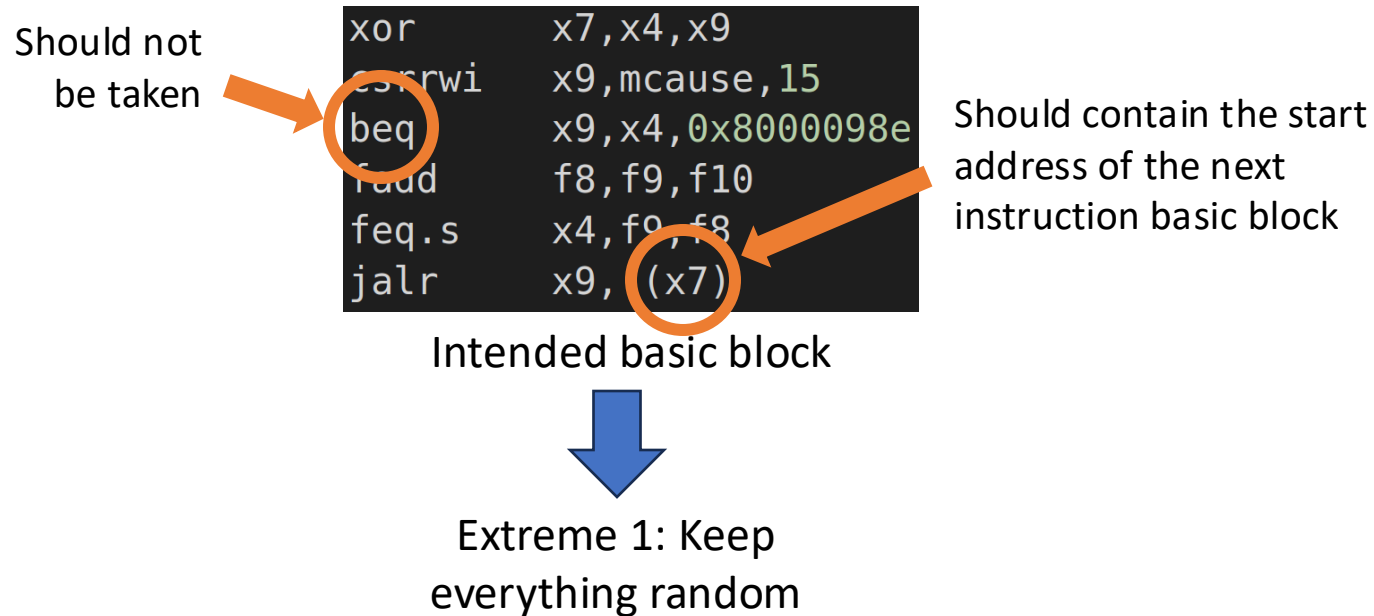
Should not  
be taken

```
xor    x7,x4,x9
csrrwi x9,mcause,15
beq    x9,x4,0x8000098e
fadd   f8,f9,f10
feq.s  x4,f9,f8
jalr   x9,(x7)
```

Should contain the start  
address of the next  
instruction basic block

Intended basic block

# Code generation in Cascade



# Code generation in Cascade

```
xor    x7,x4,x9
csrrwi x9,mcause,15
beq    x9,x4,0x8000098e
fadd   f8,f9,f10
feq.s  x4,f9,f8
jalr   x9,(x7)
```

Should not be taken

Should contain the start address of the next instruction basic block

Intended basic block



Extreme 1: Keep everything random





# Code generation in Cascade

Should not  
be taken

```
xor    x7,x4,x9
csrrwi x9,mcause,15
beq    x9,x4,0x8000098e
fadd   f8,f9,f10
feq.s  x4,f9,f8
jalr   x9,(x7)
```

Should contain the start  
address of the next  
instruction basic block

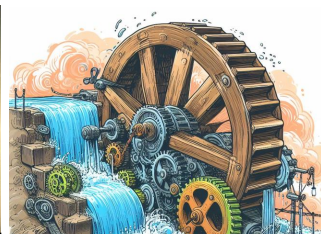
Intended basic block



Extreme 1: Keep  
everything random



Invalid



Complex

# Code generation in Cascade

Should not  
be taken

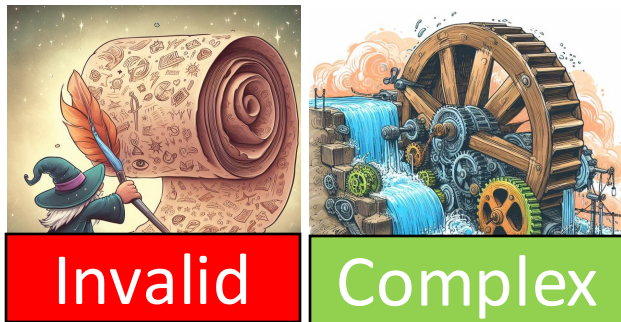
```
xor    x7, x4, x9
csrrwi x9, mcause, 15
beq    x9, x4, 0x8000098e
fadd   f8, f9, f10
feq.s  x4, f9, f8
jalr   x9, (x7)
```

Should contain the start  
address of the next  
instruction basic block

Intended basic block



Extreme 1: Keep  
everything random

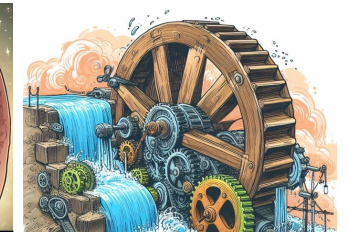


Extreme 2: Isolate control  
flow from data flow

```
xor    x7, x4, x9
csrrwi x9, mcause, 15
<nop>
fadd   f8, f9, f10
feq.s  x4, f9, f8
jalr   x9, (<"safe_reg">)
```



Valid



Simple

# Code generation in Cascade

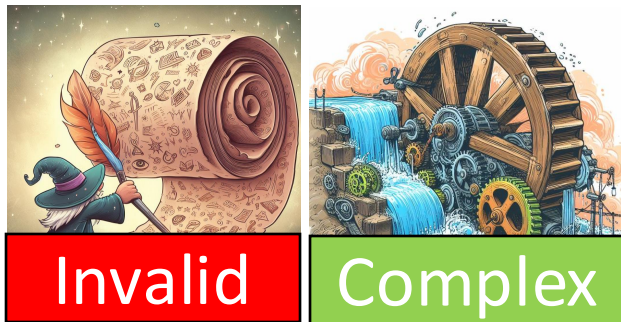
Should not be taken

```
xor    x7, x4, x9
csrrwi x9, mcause, 15
beq    x9, x4, 0x8000098e
fadd   f8, f9, f10
feq.s  x4, f9, f8
jalr   x9, (x7)
```

Should contain the start address of the next instruction basic block

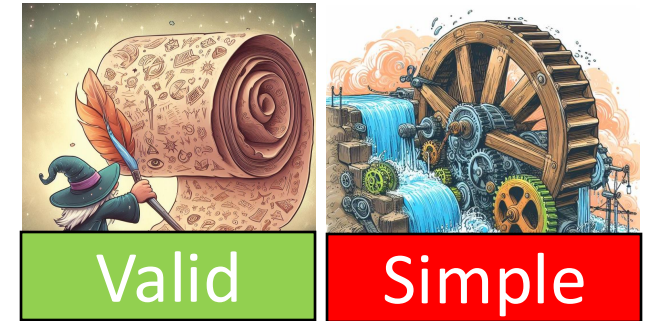
Intended basic block

Extreme 1: Keep everything random

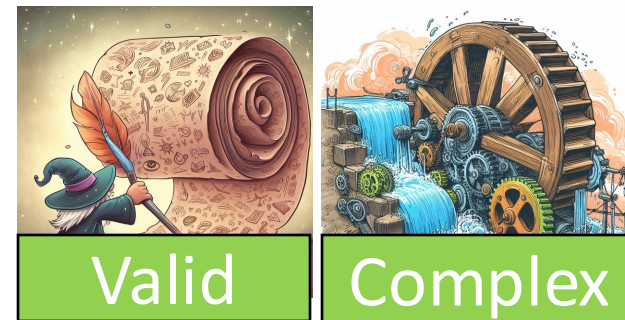


Extreme 2: Isolate control flow from data flow

```
xor    x7, x4, x9
csrrwi x9, mcause, 15
<nop>
fadd   f8, f9, f10
feq.s  x4, f9, f8
jalr   x9, (<"safe_reg">)
```



Flow entanglement



# Asymmetric ISA pre-simulation



Valid



Simple

```
xor    x7,x4,x9
csrrwi x9,mcause,15
<nop>
fadd   f8,f9,f10
feq.s  x4,f9,f8
jalr   x9, (<"safe_reg">)
```

Simplified test case



ISA Simulator  
(executed only once per program)

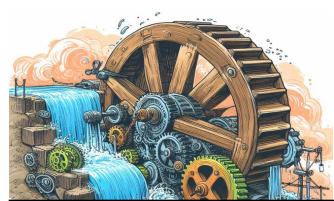




# Asymmetric ISA pre-simulation




Valid



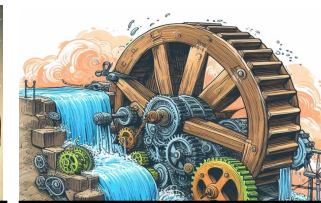
Simple

```
xor    x7,x4,x9
csrrwi x9,mcause,15
<nop>
fadd   f8,f9,f10
feq.s  x4,f9,f8
jalr   x9, (<"safe_reg">)
```

Simplified test case



Valid



Complex

```
xor    x7,x4,x9
csrrwi x9,mcause,15
bne    x9,x4,0x8000098e
fadd   f8,f9,f10
feq.s  x4,f9,f8
jalr   x9, (x7)
```

Entangled test case



ISA Simulator  
(executed only once per program)



Test case  
generation

Test case  
execution

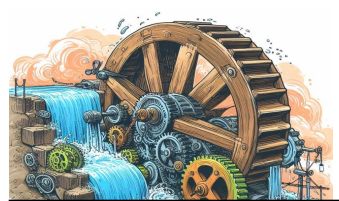




# Asymmetric ISA pre-simulation




**Valid**




**Simple**

```
xor    x7,x4,x9
csrrwi x9,mcause,15
<nop>
fadd   f8,f9,f10
feq.s  x4,f9,f8
jalr   x9, (<"safe_reg">)
```

Simplified test case



**Valid**



**Complex**

```
xor    x7,x4,x9
csrrwi x9,mcause,15
bne    x9,x4,0x8000098e
fadd   f8,f9,f10
feq.s  x4,f9,f8
jalr   x9, (x7)
```

Entangled test case



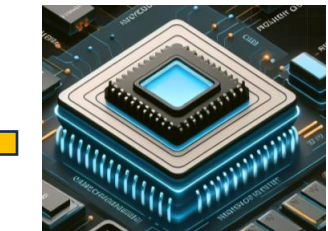
ISA Simulator  
(executed only once per program)



Test case generation

---

Test case execution



CPU under test

Does execution terminate?



# Asymmetric ISA pre-simulation



Valid




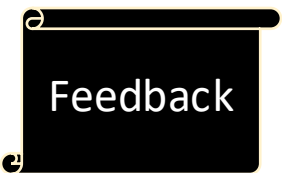
Simple

```
xor    x7,x4,x9
csrrwi x9,mcause,15
<nop>
fadd   f8,f9,f10
feq.s  x4,f9,f8
jalr   x9,(<"safe_reg">)
```

Simplified test case



ISA Simulator  
(executed only once per program)



Valid



Complex

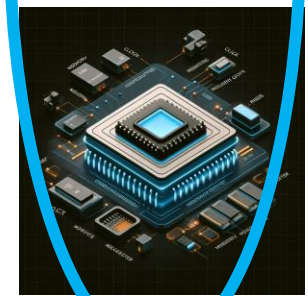
```
xor    x7,x4,x9
csrrwi x9,mcause,15
bne    x9,x4,0x8000098e
fadd   f8,f9,f10
feq.s  x4,f9,f8
jalr   x9,(x7)
```

Entangled test case

Test case generation

---

Test case execution



CPU under test

Does execution terminate?

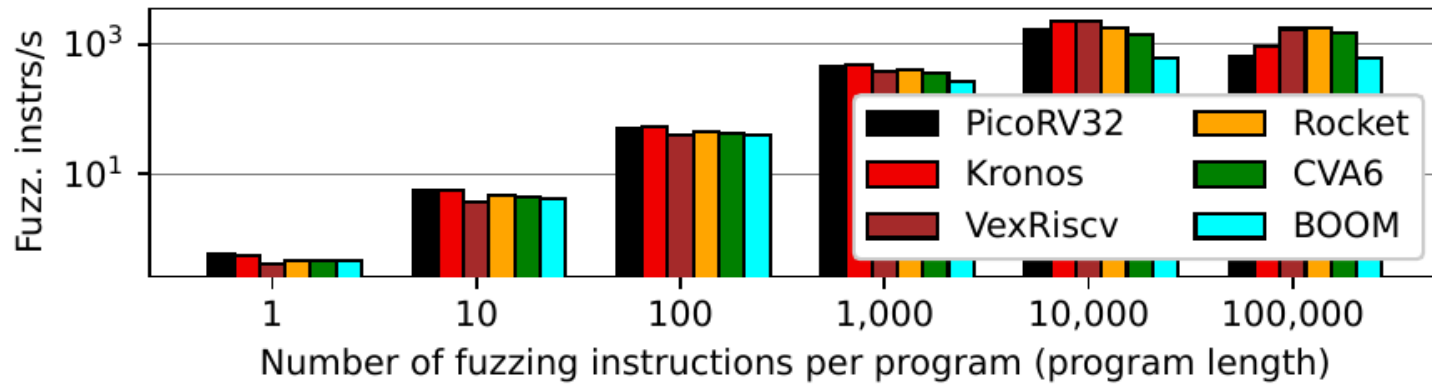




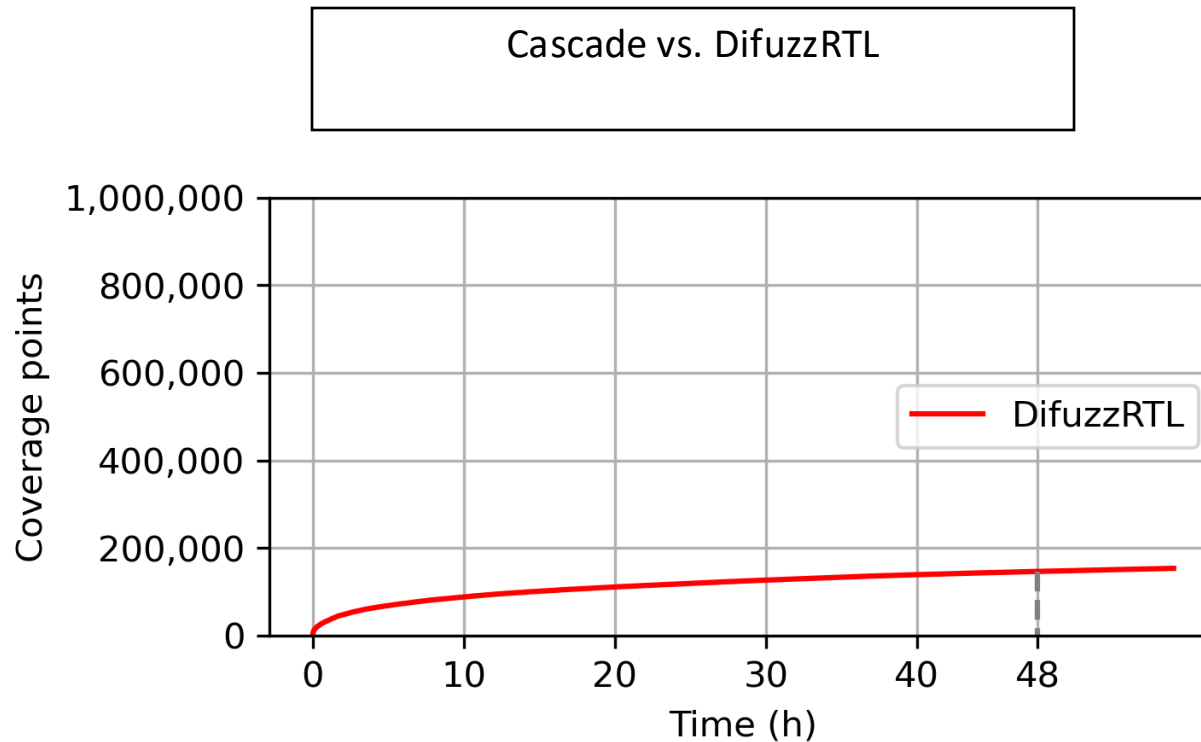
# Results



# Program length matters

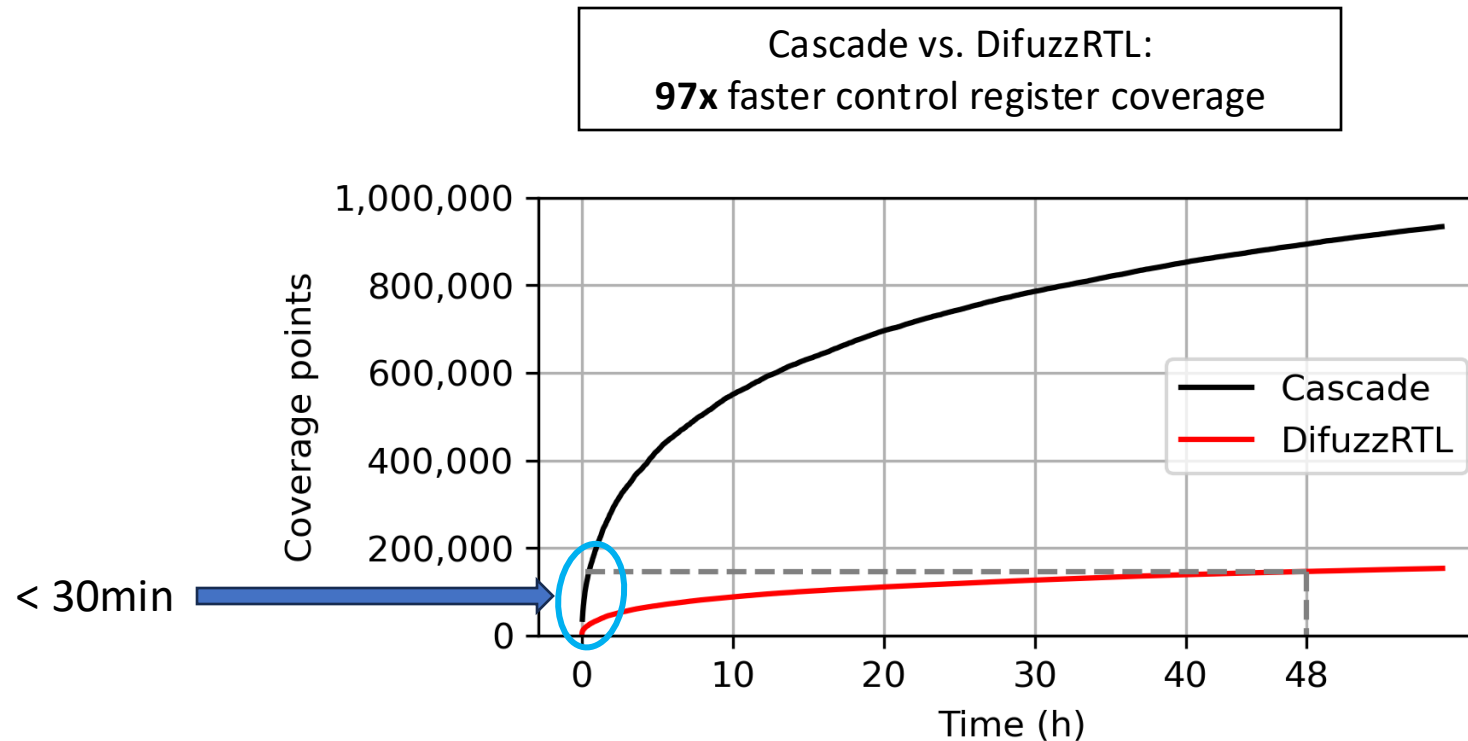


# Coverage of SoA CPU fuzzers





# Coverage of SoA CPU fuzzers



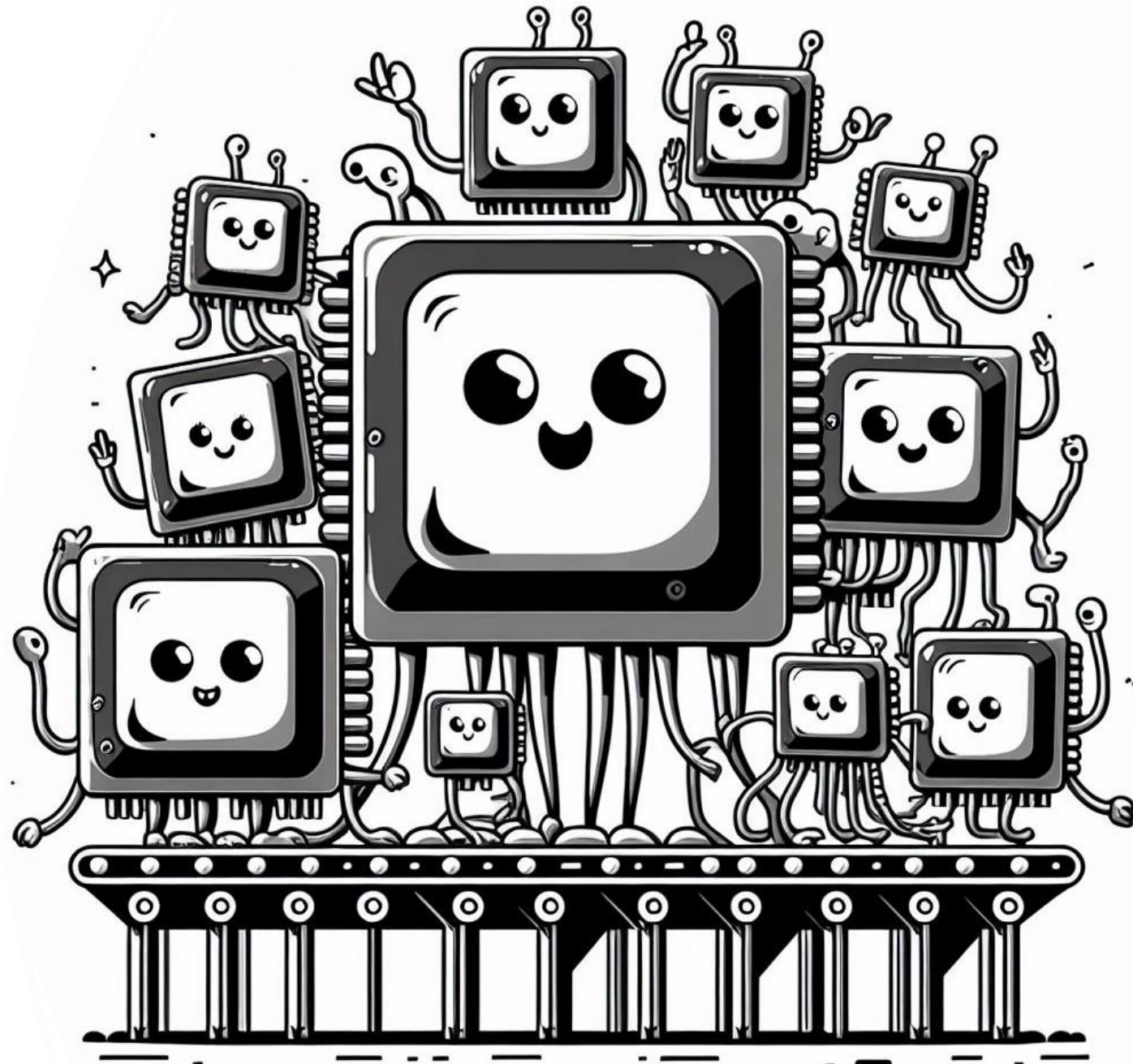


# Bugs

# RISC-V cores under test

---

- PicoRV32
- Kronos
- VexRiscv
- CVA6
- Rocket
- BOOM

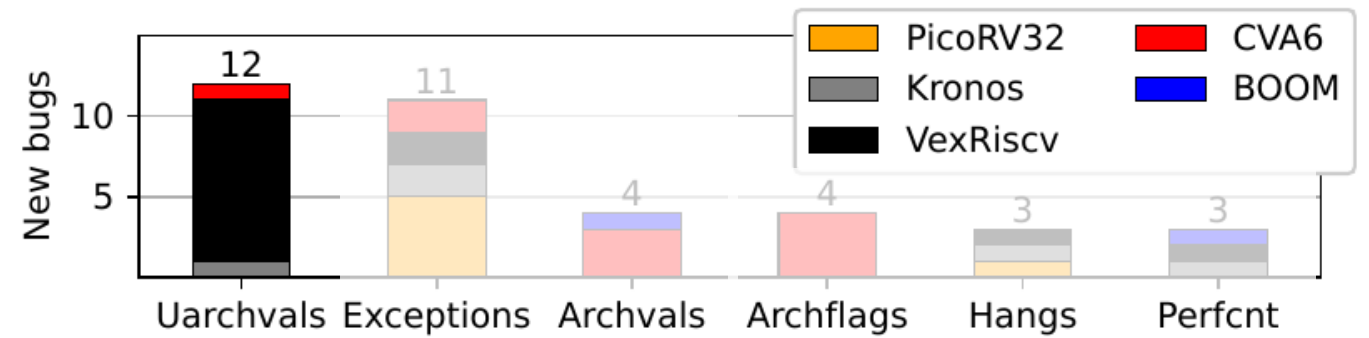




# Bugs

---

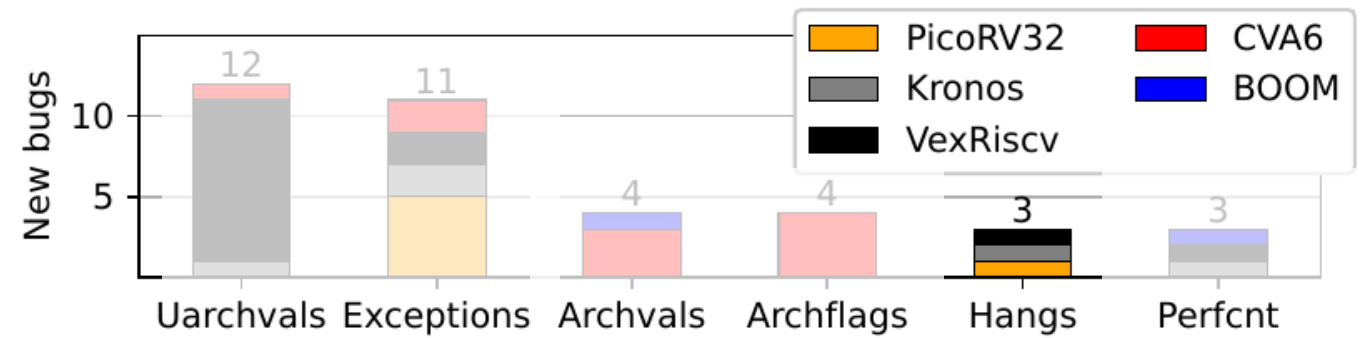
37 new CPU bugs (29 new CVEs) in 5 of the 6 cores



# Bugs

---

37 new CPU bugs (29 new CVEs) in 5 of the 6 cores

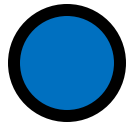




# An example bug (CVE-2023-34896)

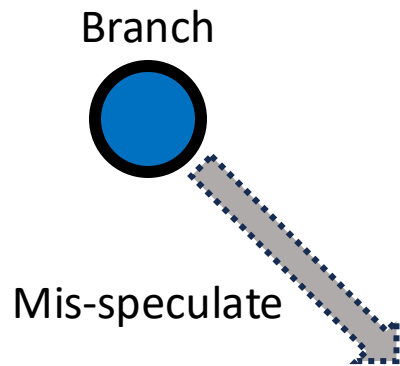
On VexRiscv without compressed instruction support

Branch



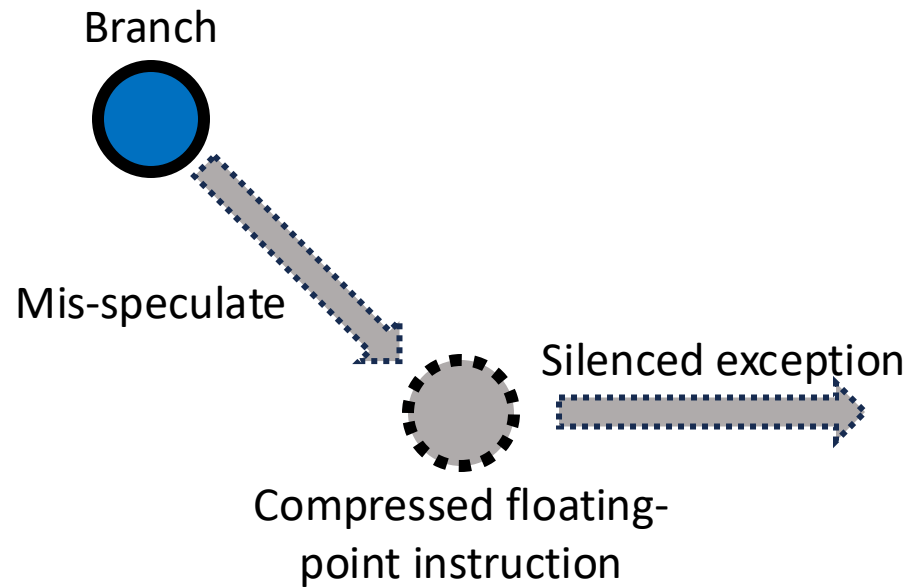
# An example bug (CVE-2023-34896)

On VexRiscv without compressed instruction support



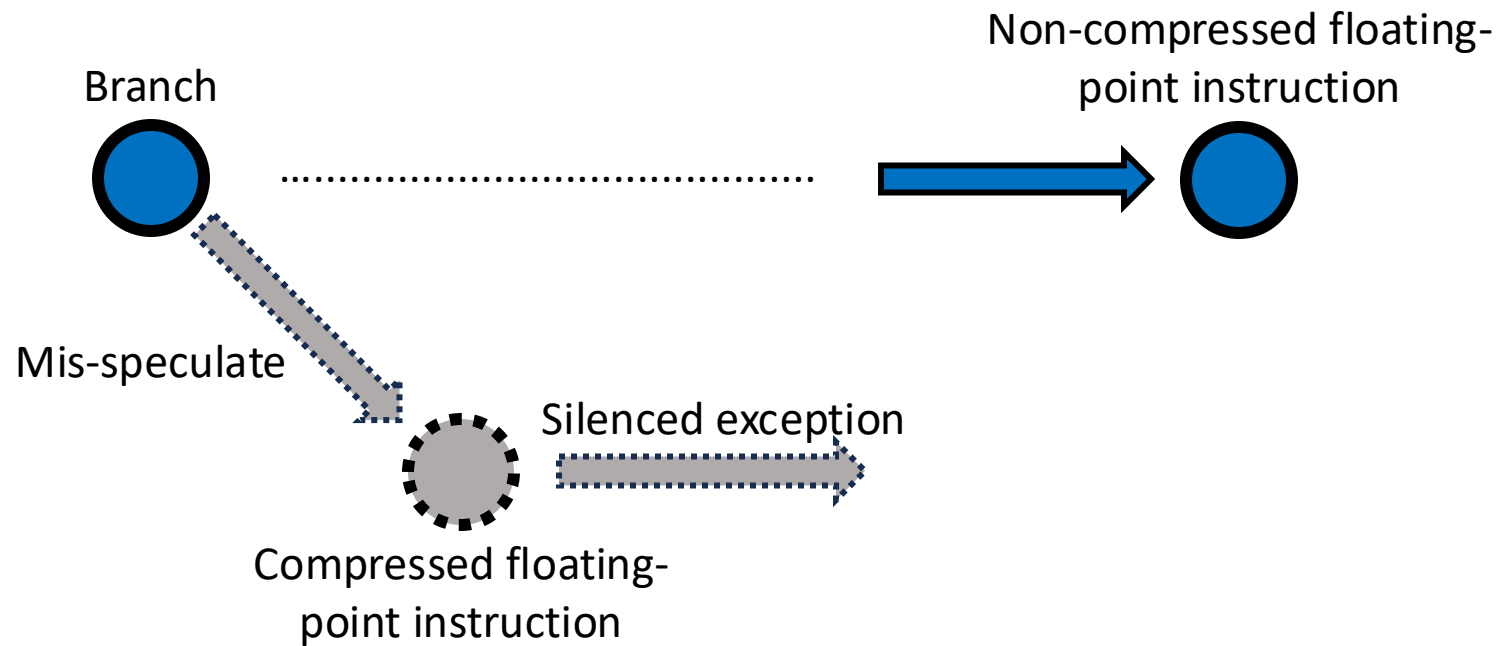
# An example bug (CVE-2023-34896)

On VexRiscv without compressed instruction support



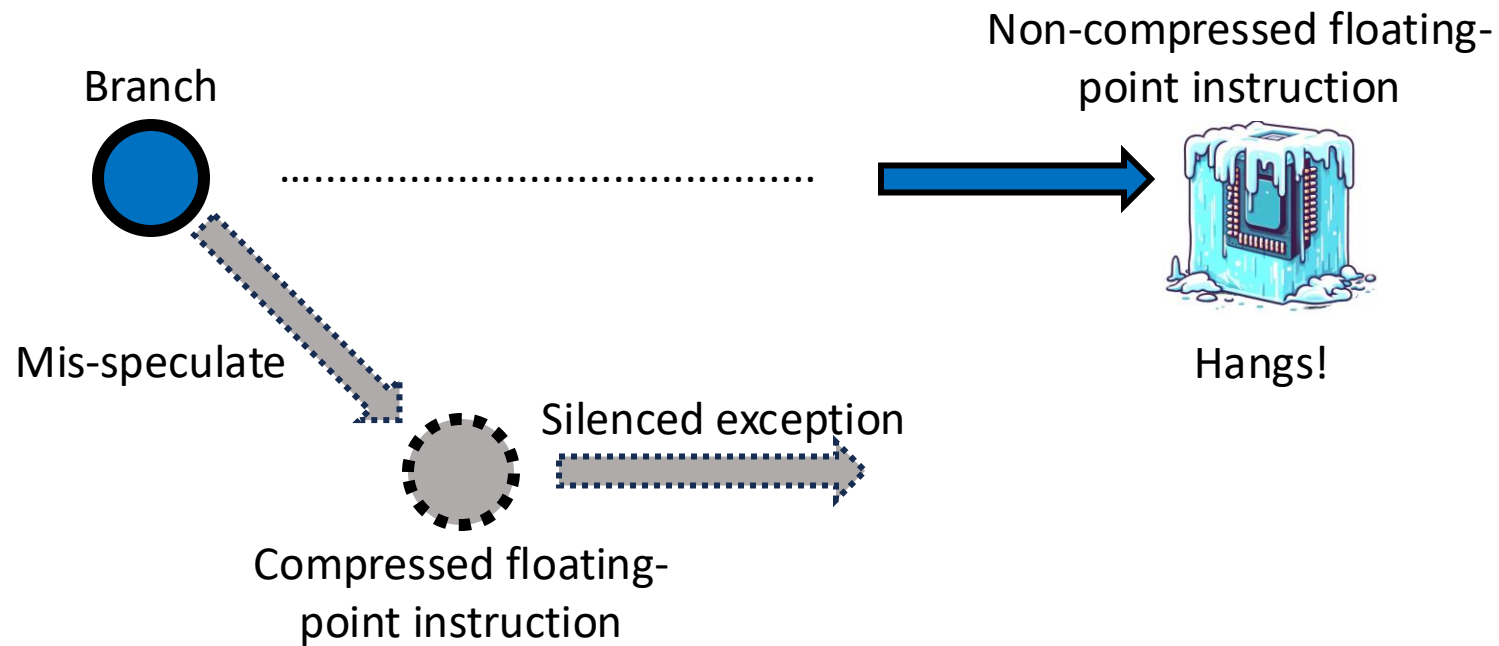
# An example bug (CVE-2023-34896)

On VexRiscv without compressed instruction support



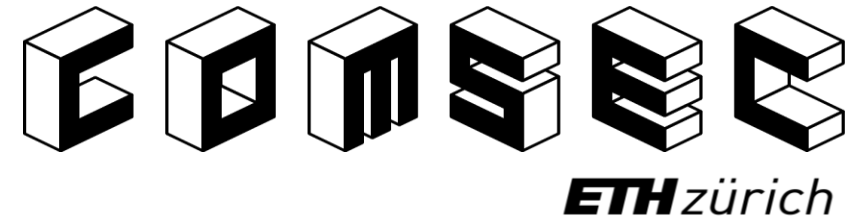
# An example bug (CVE-2023-34896)

On VexRiscv without compressed instruction support





# Conclusion



- **Cascade** is a RISC-V CPU fuzzer that generates **valid, long & complex** programs.
- Cascade introduces AIPS to entangle flows and use non-termination as a bug signal.
- Cascade outperforms state-of-the-art coverage-guided CPU fuzzers by 28-200x.
- Cascade found 37 new CPU bugs + 1 new synthesizer bug, 29 new CVEs.
- Cascade is readily open source: <https://github.com/comsec-group/cascade-artifacts>



f1solt@ethz.ch



comsec.ethz.ch



comsec-group/cascade-artifacts



Computer Security Group