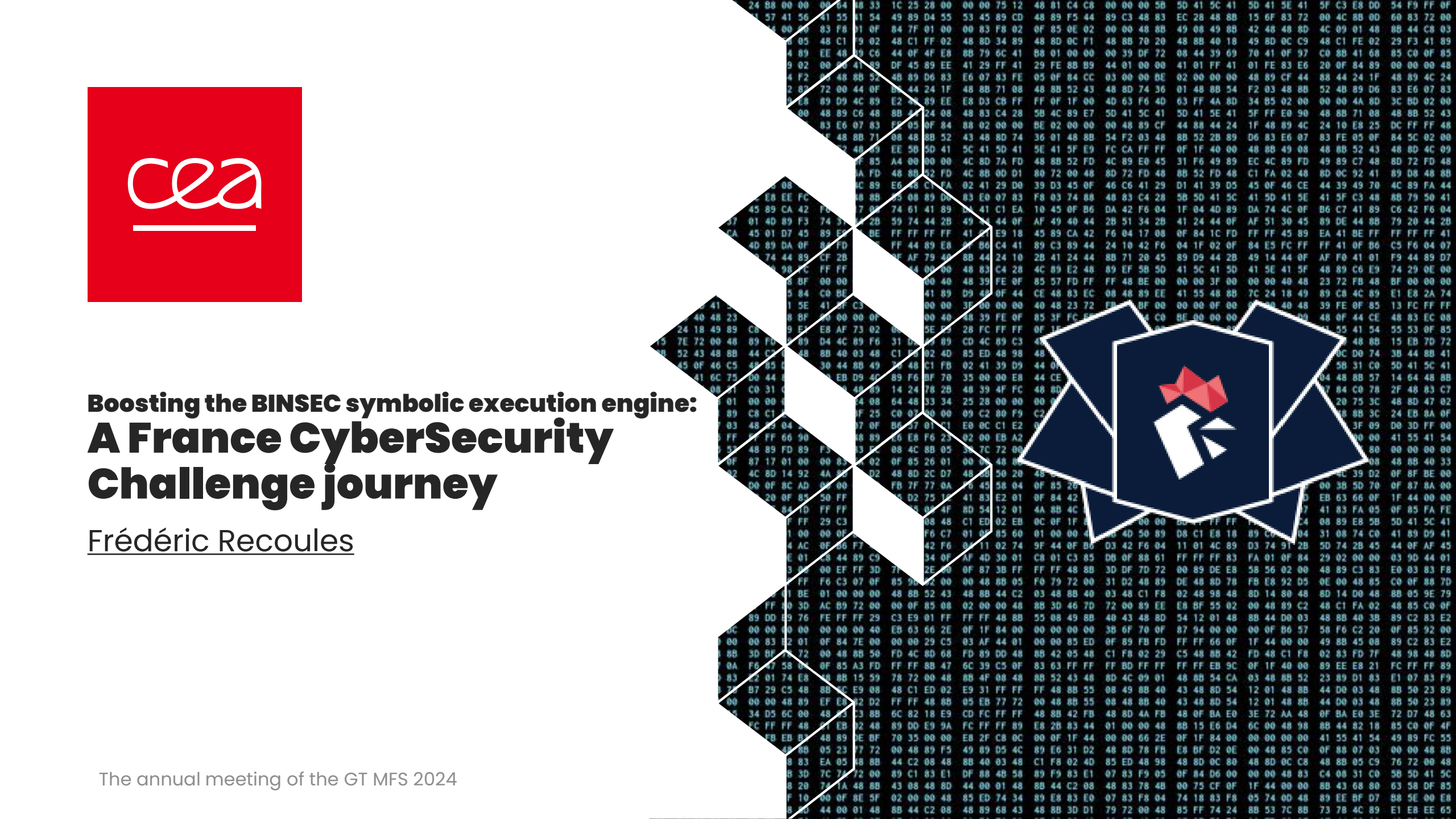


Boosting the BINSEC symbolic execution engine:

A France CyberSecurity Challenge journey

Frédéric Recoules

The annual meeting of the GT MFS 2024

Beginning of the journey

Former intern

(Binsec/Rel2)

Beginning of the journey

Former intern

(Binsec/Rel2)

Former PhD student

(Binsec/Rel)

Beginning of the journey

Former intern (Binsec/Rel2)

Former PhD student (Binsec/Rel)

Z80

Beginning of the journey

Former intern (Binsec/Rel2)

Former PhD student (Binsec/Rel)

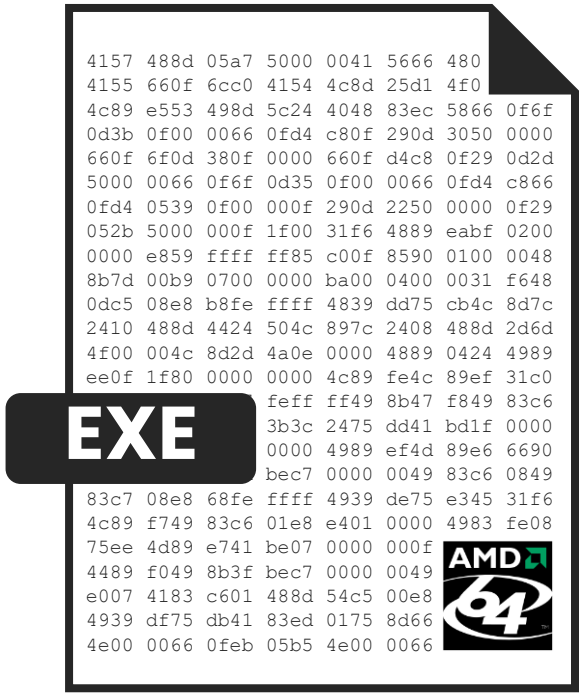
Z80

France Cybersecurity Challenge 2022

France Cybersecurity Challenge

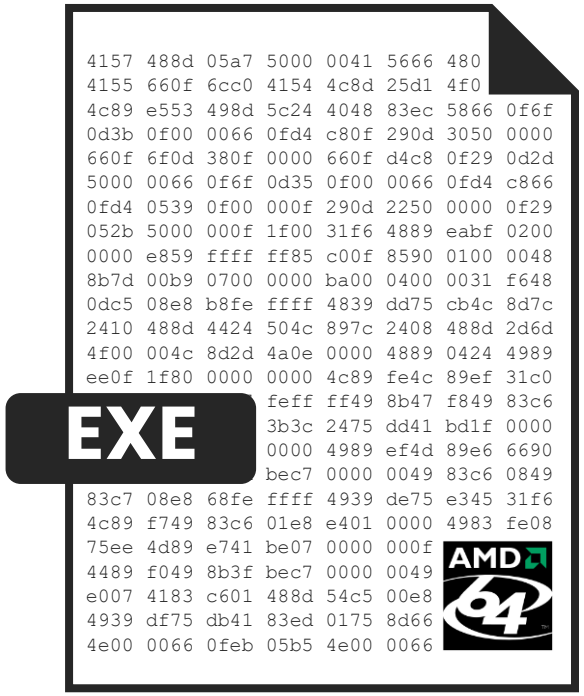
- Preselection for the European Cybersecurity Challenge
- crypto, **reverse**, pwn, web, forensics, hardware, side channel or fault attack, etc.
- From **5 April 2024, 14h** to 14 April

An example of reverse engineering challenge



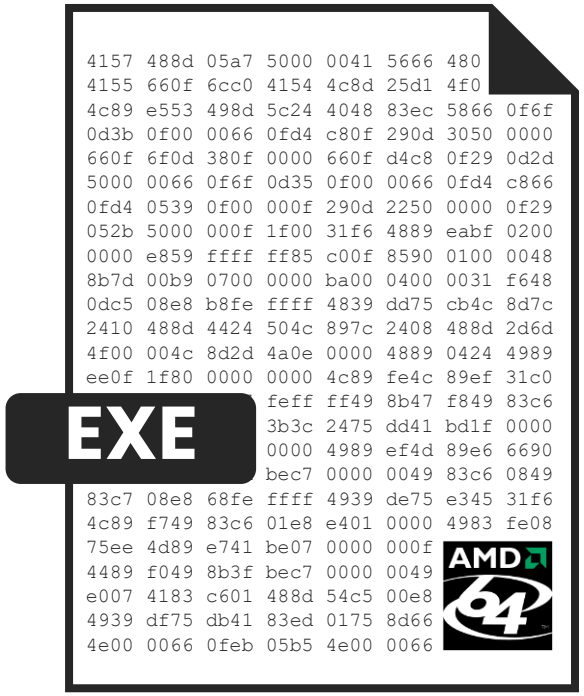
An example of reverse engineering challenge

```
> =-=-=-= Very secure vault =-=-=-=
> Please enter you very secure password:
> _
```



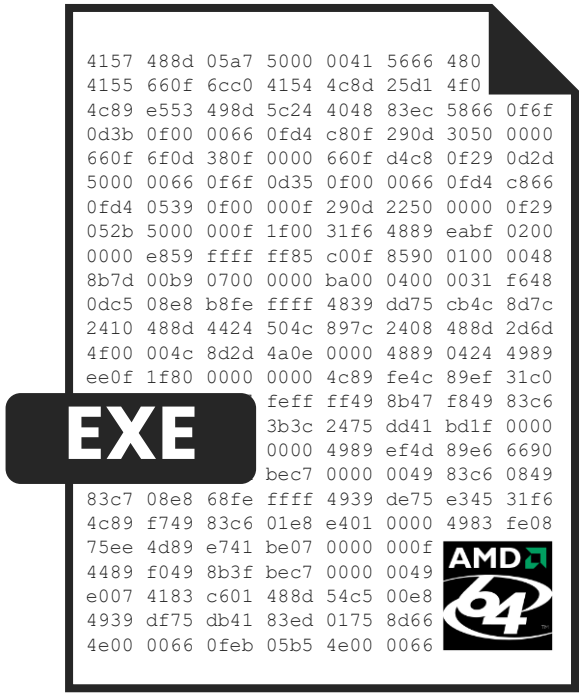
An example of reverse engineering challenge

```
> =-=-=-= Very secure vault =-=-=-=
> Please enter you very secure password:
> aaaaaaa
```



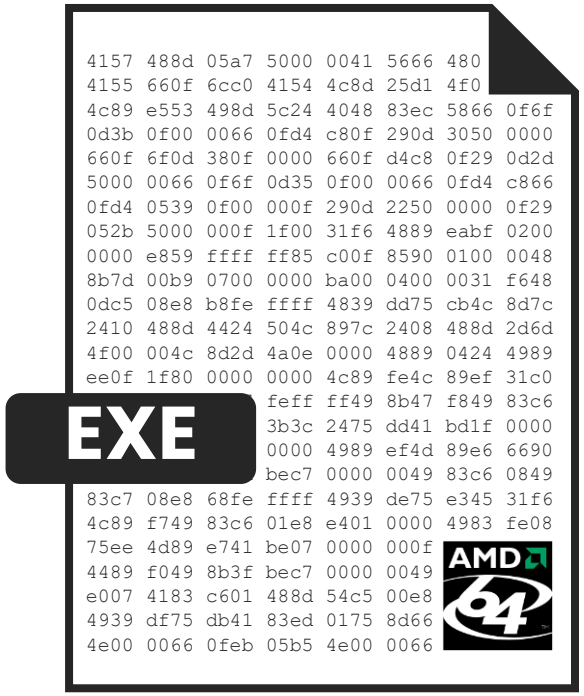
An example of reverse engineering challenge

```
> =-=-=-= Very secure vault =-=-=-=
> Please enter you very secure password:
> aaaaaaa
> Wrong password: authorities have been alerted!
```



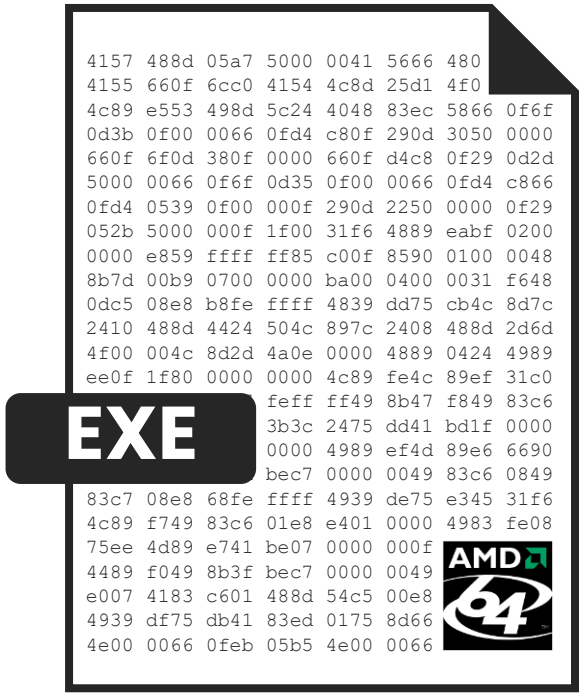
An example of reverse engineering challenge

```
> =-=-=-= Very secure vault =-=-=-=
> Please enter you very secure password:
> _
```



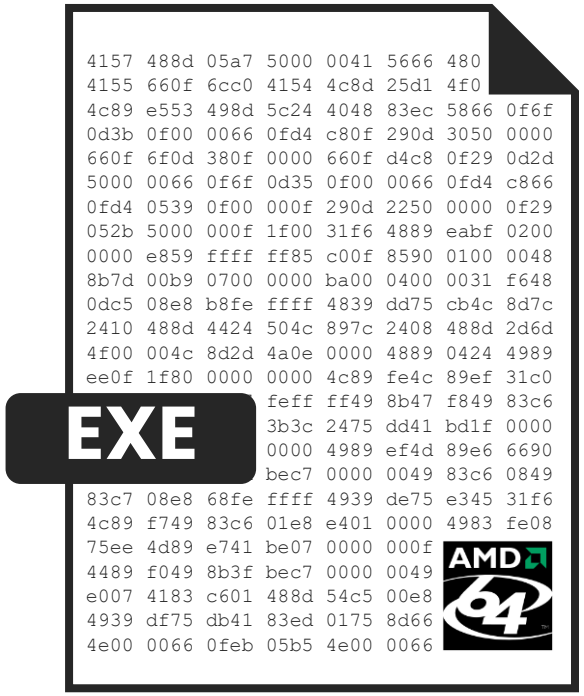
An example of reverse engineering challenge

```
> =-=-=-= Very secure vault =-=-=-=
> Please enter you very secure password:
> 346bc605be4ed8361a68a3d9748fc9b87de397e1
```



An example of reverse engineering challenge

```
> =-=-=-= Very secure vault =-=-=-=
> Please enter you very secure password:
> 346bc605be4ed8361a68a3d9748fc9b87de397e1
>\o/ Access granted! \o/
>
> Here is your flag:
> ECSC{346bc605be4ed8361a68a3d9748fc9b87de397e1}
```



An example of reverse engineering challenge

```
> =-=-=-= Very secure vault =-=-=-=
> Please enter you very secure password:
> 346bc605be4ed8361a68a3d9748fc9b87de397e1
>\o/ Access granted! \o/
>
> Here is your flag:
> ECSC{346bc605be4ed8361a68a3d9748fc9b87de397e1}
```

```
int check_char(int pos, char c) {
  return password[(pos + 10) % 40] == c;
}

int main(int argc, char **argv) {
  char c;
  int pos = 0;
  int res = 1;
  int sum = 0;

  while((c = getchar()) != EOF) {
      if (c == '\n') break;
      res &= check_char(pos, c);
      if (res) sum += c;
      pos += 1;
  }
  if (res == 1 && sum == 2827) success();
  else failure();
  return 0;
}
```

This talk in a nutshell

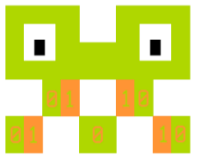
Goal

Introduce some recent improvements of the BINSEC symbolic execution engine through the prism of CTF challenges

Highlights

- Symbolic execution and its limitations
- Improvements
  - Efficient use of SMT solvers
  - Straightforward path merging
  - JIT specialization of the interpreter

BINSEC in a nutshell (since 2012)



Bitwuzla

binary lifting,
IR, CFG, call graph,
symbolic execution,
static analysis, ..

Solvers

Decoders

Symbolic engine

BINSEC

Generic IR

OCaml

56k LOC

Vulnerability Assessment

Security critical components

- Fault injection
- Side channel attack
- Attacker model

Bug finding

Supply chain

- Advanced fuzzing
- Test case generation

Reverse Engineering

Malware comprehension

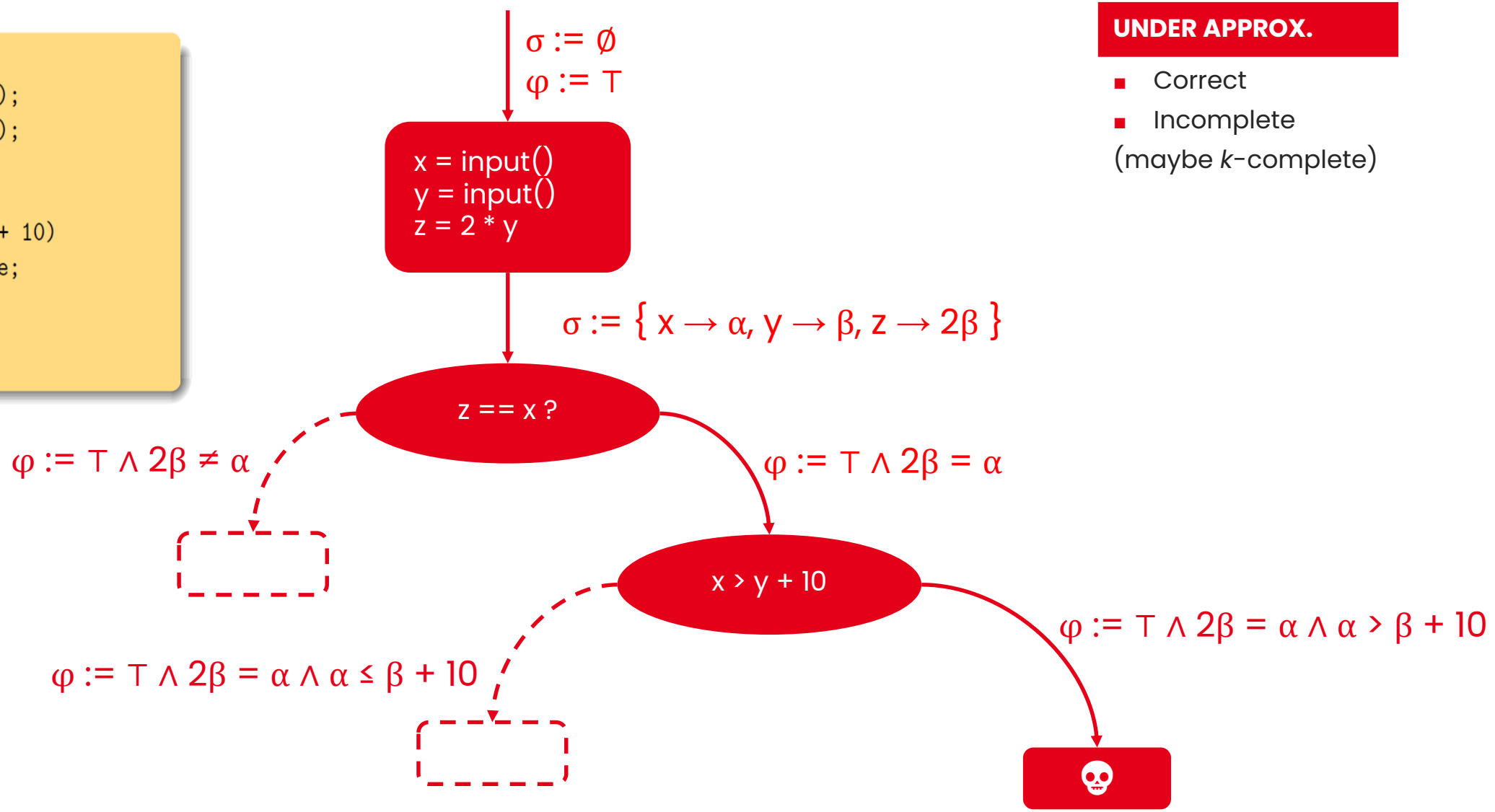
- Capture The Flag
- Deobuscation
- Decompilation

arm

UNISIM Virtual Platforms .org

Symbolic execution in a nutshell

```
int main () {
    int x = input();
    int y = input();
    int z = 2 * y;
    if (z == x) {
        if (x > y + 10)
            failure;
    }
    success;
}
```

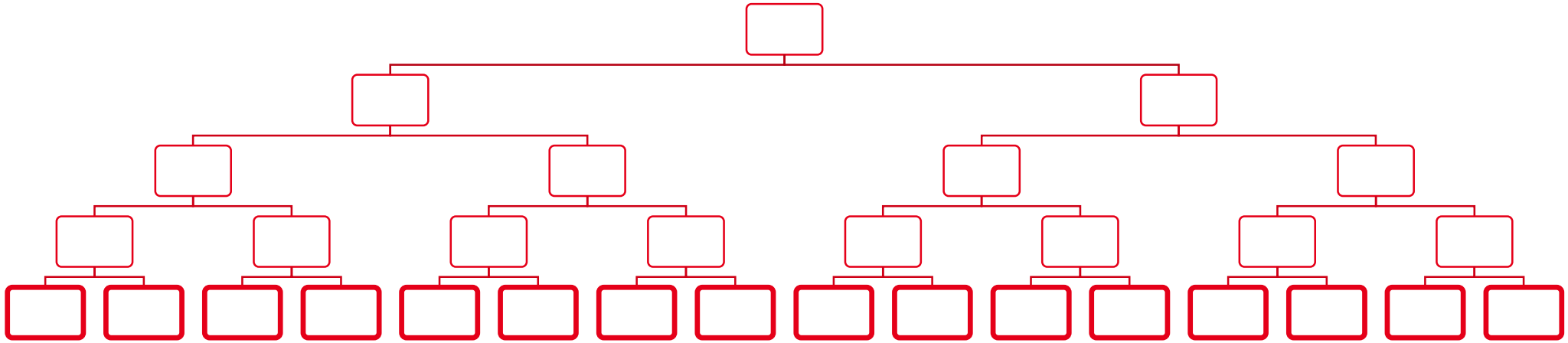


UNDER APPROX.

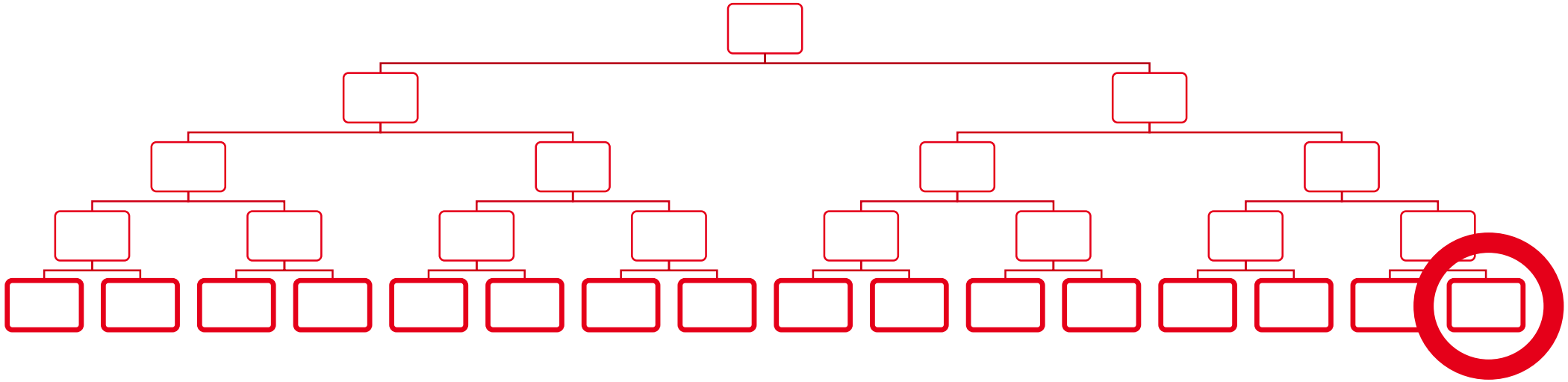
- Correct
- Incomplete

(maybe *k*-complete)

Theoretical and practical limits



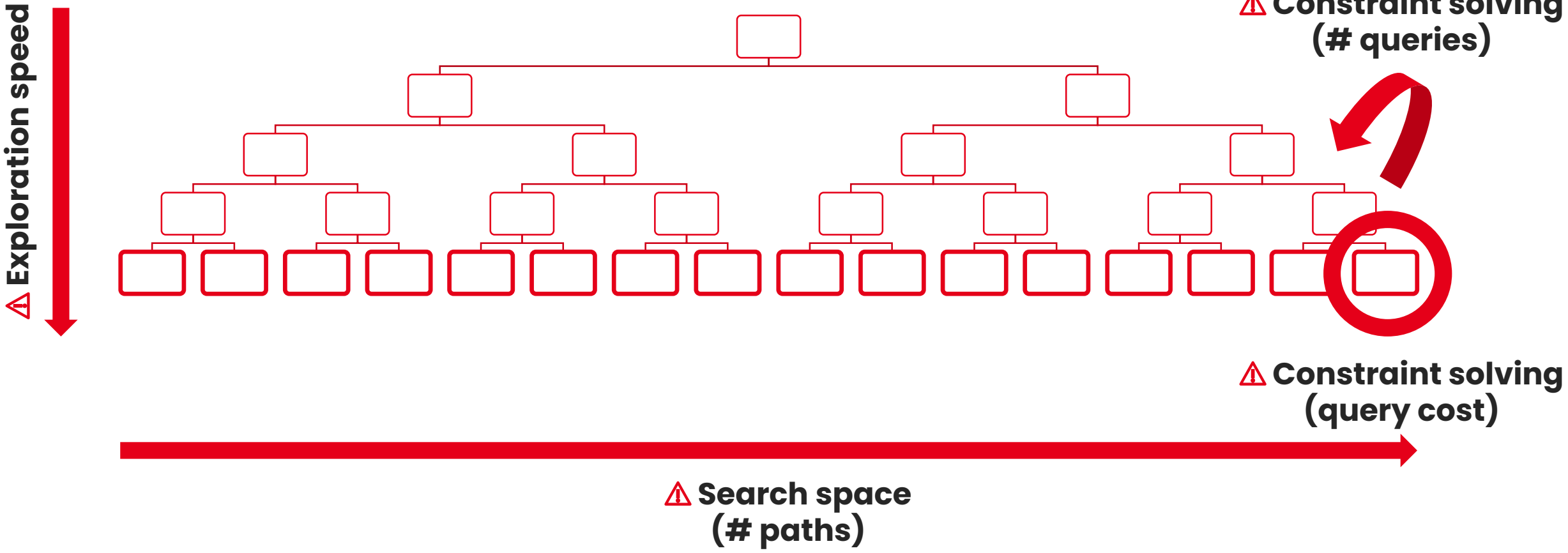
Theoretical and practical limits



⚠ Constraint solving (query cost)

⚠ Search space (# paths)

Theoretical and practical limits



How to help competitors

Applicability
1

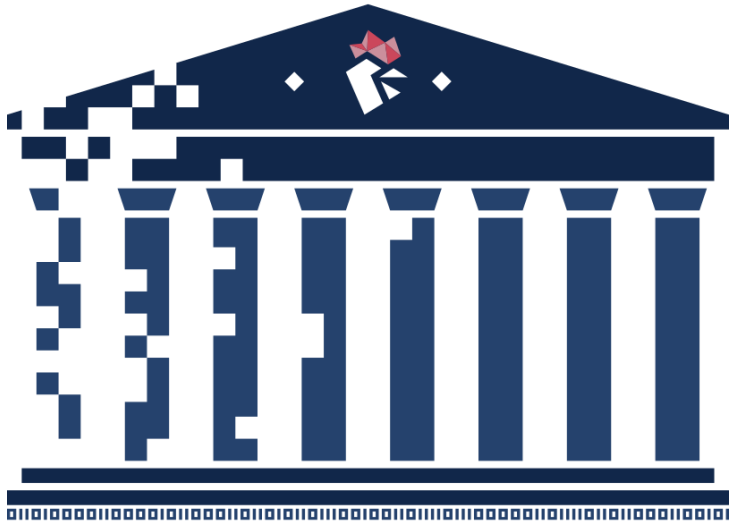
32 / 64 bits architectures, static / shared / self-modifying code, etc.

Usability
2

easy to install, easy to configure, meaningful result, etc.

Speed
3

trial-and-error, should be faster than doing it by hand!



Sneak peak over usability progresses

From salt

```
[kernel]
entrypoint = main

[disasm]
decode-replacement =                                                    \
    <fgets> ->                                                          \
      0: nondet_assume ({asize<64>},                                    \
                        ((0<64> < asize<64>) && (asize<64> <= rsi<64>))); \
              goto 1                                                     \
      1: i<64> := 0<64>; goto 2                                         \
      2: if (i<64> <=s (asize<64> - 1<64>)) goto 3 else goto 6          \
      3: @[rdi<64> + i<64>, 1] := stdin[stdin_p<64>, 1]; goto 4         \
      4: stdin_p<64> := stdin_p<64> + 1<64>; goto 5                     \
      5: i<64> := i<64> + 1<64>; goto 2                                 \
      6: rax<64> := rdi<64>; goto 7                                     \
      7: rsp<64> := rsp<64> + 8<64>; goto 8                             \
      8: goto @[rsp<64> - 8<64>, 8] // ret                              \
    <__stack_chk_fail> ->                                               \
      0: assert (0<1>); goto 0

[sse]
enabled = true
directives = <puts> reach; <main + 144> cut
```

To sugar

```
starting from core

stdin_p<64> := 0
replace <fgets> (ptr, size, _) by
  asize<64> := nondet
  assume 0 < asize <= size
  for i<64> in 0 to asize - 1 do
    @[ptr + i] := stdin[stdin_p]
    stdin_p := stdin_p + 1
  end
  return ptr
end

reach <puts> then print c string stdin

halt at @[rsp, 8]
abort at <__stack_chk_fail>
```

Sneak peak over usability progresses

From salt

```
[kernel]
entrypoint = main

[disasm]
decode-replacement =                                                    \
    <fgets> ->                                                          \
      0: nondet_assume ({asize<64>},                                    \
                        ((0<64> < asize<64>) && (asize<64> <= rsi<64>))); \
              goto 1                                                    \
      1: i<64> := 0<64>; goto 2                                         \
      2: if (i<64> <=s (asize<64> - 1<64>)) goto 3 else goto 6          \
      3: @[rdi<64> + i<64>, 1] := stdin[stdin_p<64>, 1]; goto 4         \
      4: stdin_p<64> := stdin_p<64> + 1<64>; goto 5                     \
      5: i<64> := i<64> + 1<64>; goto 2                                 \
      6: rax<64> := rdi<64>; goto 7                                     \
      7: rsp<64> := rsp<64> + 8<64>; goto 8                             \
      8: goto @[rsp<64> - 8<64>, 8] // ret                              \
    <__stack_chk_fail> ->                                               \
      0: assert (0<1>); goto 0

[sse]
enabled = true
directives = <puts> reach; <main + 144> cut
```

To sugar

```
starting from core

stdin_p<64> := 0
replace <fgets> (ptr, size, _) by
  asize<64> := nondet
  assume 0 < asize <= size
  for i<64> in 0 to asize - 1 do
    @[ptr + i] := stdin[stdin_p]
    stdin_p := stdin_p + 1
  end
  return ptr
end

reach <puts> then print c string stdin

halt at @[rsp, 8]
abort at <__stack_chk_fail>
```

01

CORE DUMP

Dynamic shared libraries handling
Complex initialization

Sneak peak over usability progresses

From salt

```
[kernel]
entrypoint = main

[disasm]
decode-replacement =                                                    \
    <fgets> ->                                                          \
      0: nondet_assume ({asize<64>},                                    \
                        ((0<64> < asize<64>) && (asize<64> <= rsi<64>))); \
              goto 1                                                    \
      1: i<64> := 0<64>; goto 2                                         \
      2: if (i<64> <=s (asize<64> - 1<64>)) goto 3 else goto 6          \
      3: @[rdi<64> + i<64>, 1] := stdin[stdin_p<64>, 1]; goto 4         \
      4: stdin_p<64> := stdin_p<64> + 1<64>; goto 5                     \
      5: i<64> := i<64> + 1<64>; goto 2                                 \
      6: rax<64> := rdi<64>; goto 7                                     \
      7: rsp<64> := rsp<64> + 8<64>; goto 8                             \
      8: goto @[rsp<64> - 8<64>, 8] // ret                              \
    <__stack_chk_fail> ->                                               \
      0: assert (0<1>); goto 0

[sse]
enabled = true
directives = <puts> reach; <main + 144> cut
```

To sugar

```
starting from core

stdin_p<64> := 0
replace <fgets> (ptr, size, _) by
  asize<64> := nondet
  assume 0 < asize <= size
  for i<64> in 0 to asize - 1 do
    @[ptr + i] := stdin[stdin_p]
    stdin_p := stdin_p + 1
  end
  return ptr
end

reach <puts> then print c string stdin

halt at @[rsp, 8]
abort at <__stack_chk_fail>
```

01 **CORE DUMP**
Dynamic shared libraries handling
Complex initialization

02 **GOAL & ACTION**
Format values as needed

Sneak peak over usability progresses

From salt

```
[kernel]
entrypoint = main

[disasm]
decode-replacement =                                                          \
    <fgets> ->                                                                \
      0: nondet_assume ({asize<64>},                                          \
                        ((0<64> < asize<64>) && (asize<64> <= rsi<64>)));     \
              goto 1                                                          \
      1: i<64> := 0<64>; goto 2                                               \
      2: if (i<64> <=s (asize<64> - 1<64>)) goto 3 else goto 6                \
      3: @[rdi<64> + i<64>, 1] := stdin[stdin_p<64>, 1]; goto 4               \
      4: stdin_p<64> := stdin_p<64> + 1<64>; goto 5                           \
      5: i<64> := i<64> + 1<64>; goto 2                                       \
      6: rax<64> := rdi<64>; goto 7                                           \
      7: rsp<64> := rsp<64> + 8<64>; goto 8                                   \
      8: goto @[rsp<64> - 8<64>, 8] // ret                                    \
    <__stack_chk_fail> ->                                                     \
      0: assert (0<1>); goto 0

[sse]
enabled = true
directives = <puts> reach; <main + 144> cut
```

To sugar

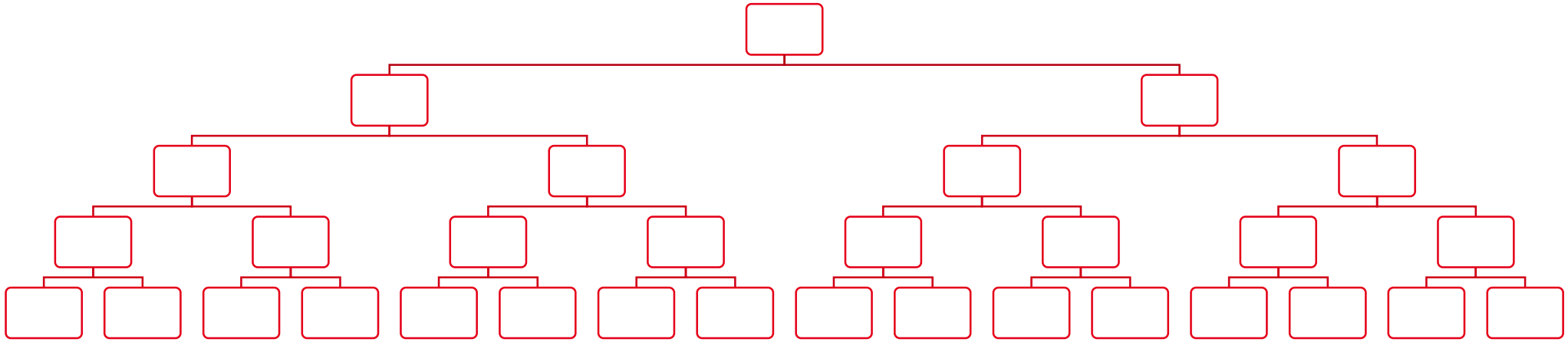
```
starting from core

stdin_p<64> := 0
replace <fgets> (ptr, size, _) by
  asize<64> := nondet
  assume 0 < asize <= size
  for i<64> in 0 to asize - 1 do
    @[ptr + i] := stdin[stdin_p]
    stdin_p := stdin_p + 1
  end
  return ptr
end

reach <puts> then print c string stdin

halt at @[rsp, 8]
abort at <__stack_chk_fail>
```

01 **CORE DUMP**
Dynamic shared libraries handling
Complex initialization

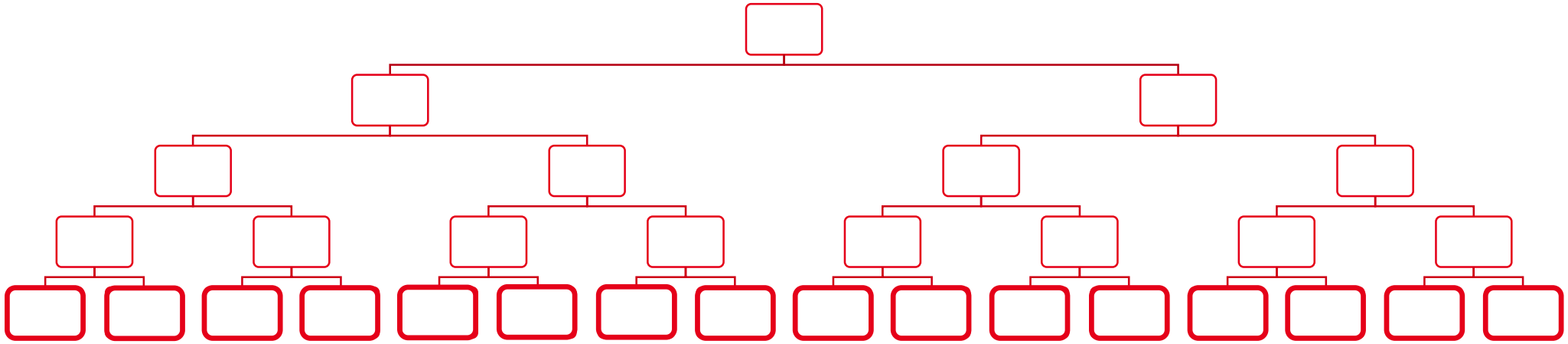
03 **SYNTACTIC SUGAR**
Optional size
Syntactic bloc (if, for, while)
Automatic argument & return

02 **GOAL & ACTION**
Format values as needed

Avoid calling the SMT solver



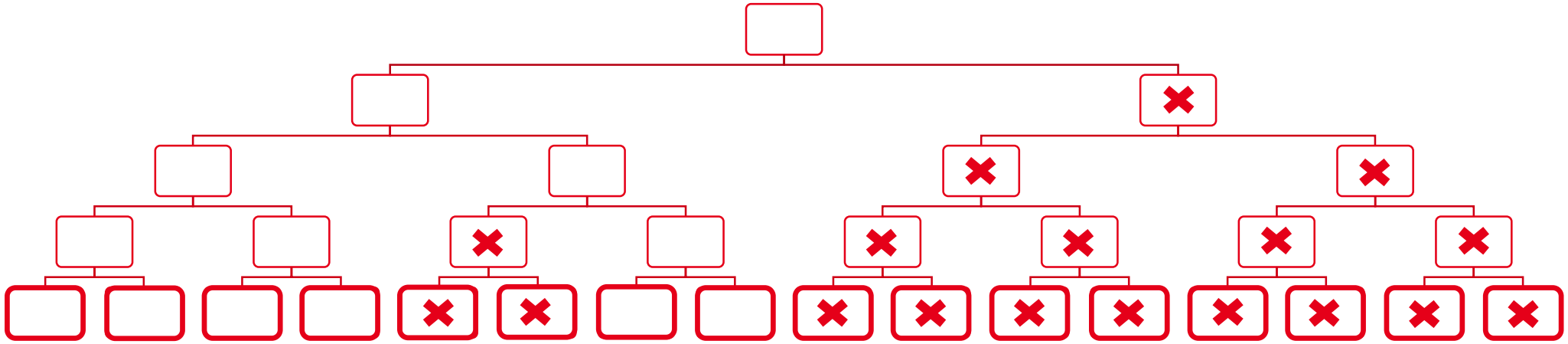
Avoid calling the SMT solver



01

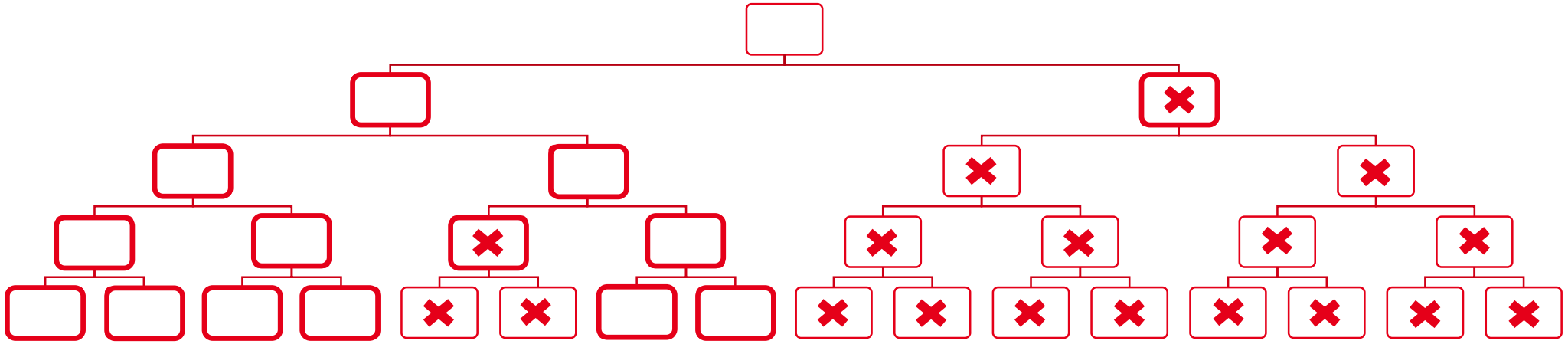
SMT SOLVER
Generate a new model

Avoid calling the SMT solver



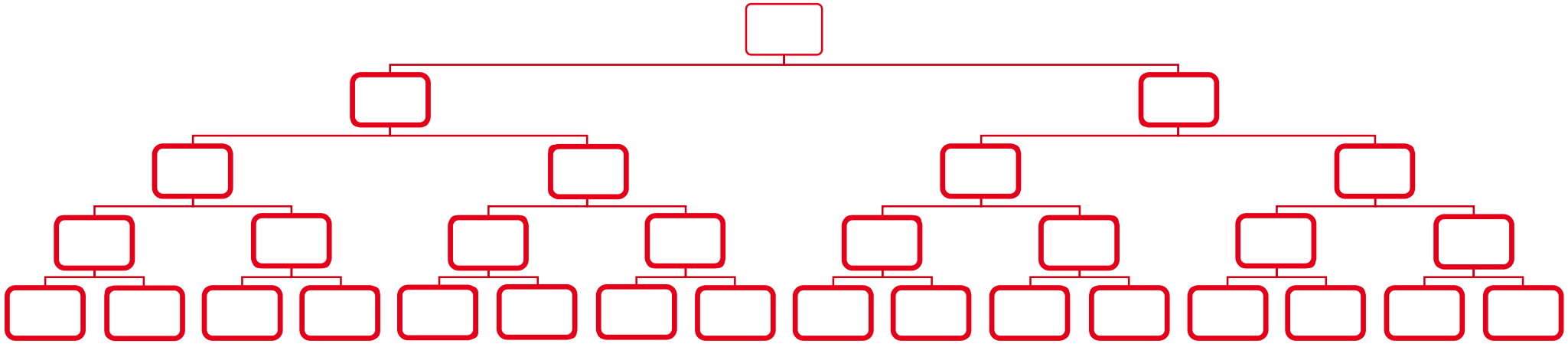
01 **SMT SOLVER**
Generate a new model

Avoid calling the SMT solver



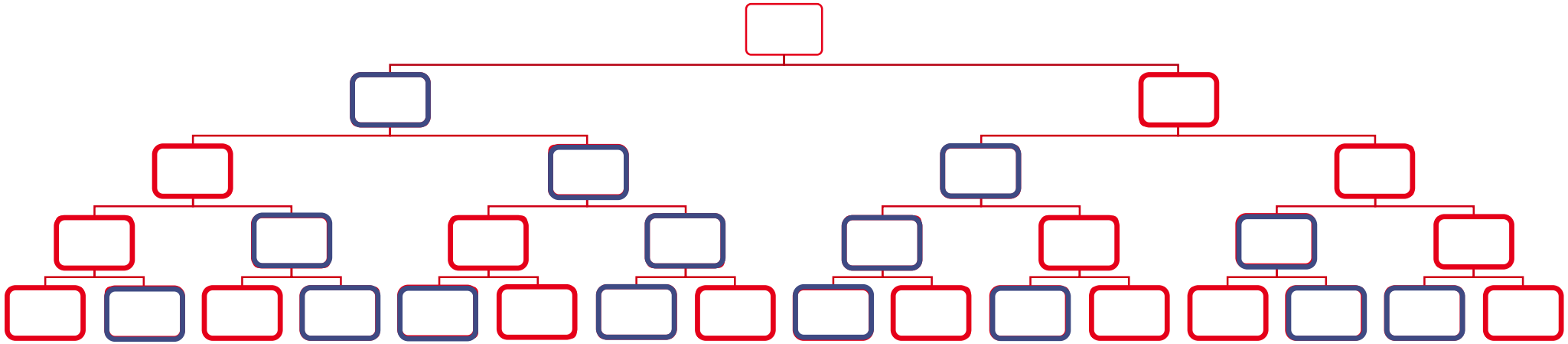
01 **SMT SOLVER**
Generate a new model

Avoid calling the SMT solver



01 **SMT SOLVER**
Generate a new model

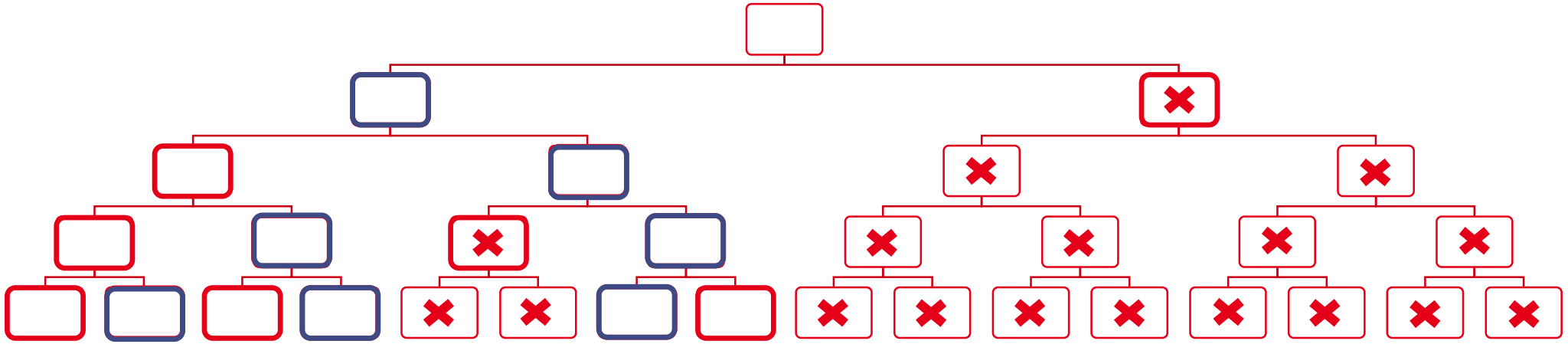
Avoid calling the SMT solver



01 **SMT SOLVER**
Generate a new model

02 **UNDER APPROX.**
Along a path
Propagate the previous model
Halve the solver queries

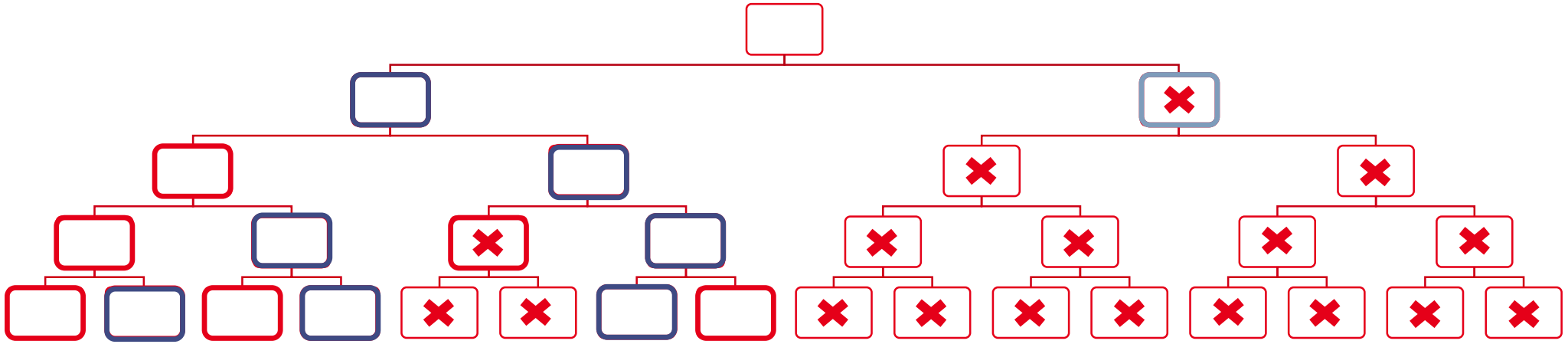
Avoid calling the SMT solver



01 **SMT SOLVER**
Generate a new model

02 **UNDER APPROX.**
Along a path
Propagate the previous model
Halve the solver queries

Avoid calling the SMT solver

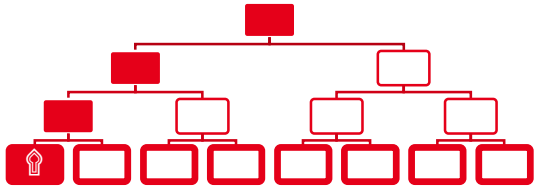


01 **SMT SOLVER**
Generate a new model

02 **UNDER APPROX.**
Along a path
Propagate the previous model
Halve the solver queries

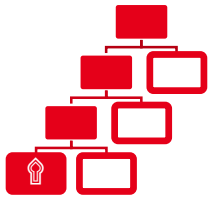
03 **OVER APPROX.**
Along a path
Discard invalid constraints

Souk : the diamond loop



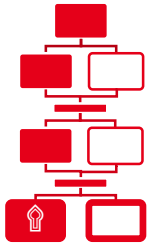
2ⁿ

EXPLOSION
All combination are explored even if they can not lead to the goal



n+1

EARLY EXITS
Paths end as soon as they can no longer lead to the goal



2

MERGING
Paths are merged when possible – reasoning is harder but paths are under control

Quick merging from timeout ($> 2^{71}$ paths) to 30s

Licorne : a difficulty that is not virtual

QEMU performs just in time compilation
⚠ **write in memory can change code !**

01
Shared library

Binary is not self-contained ?
💾 **allow starting from a process snapshot !**
(coredump)

02
Self-modifying code

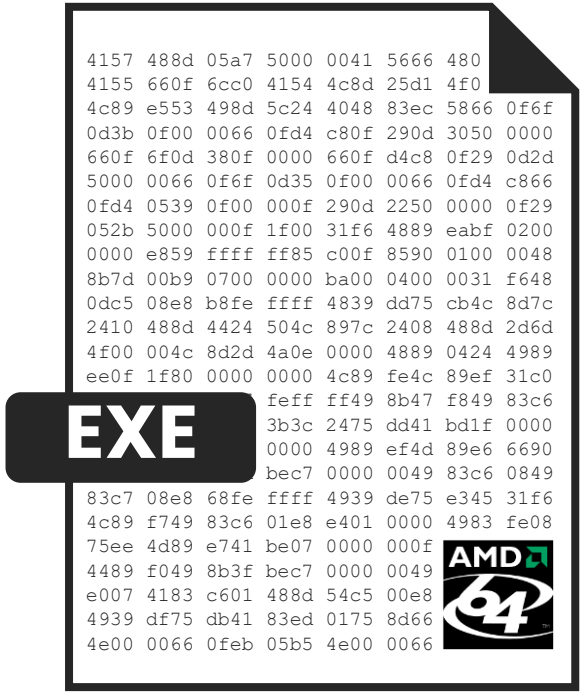
03
Billion of instructions

Emulate an emulator ?
⚡ **need to be twice as performant !**

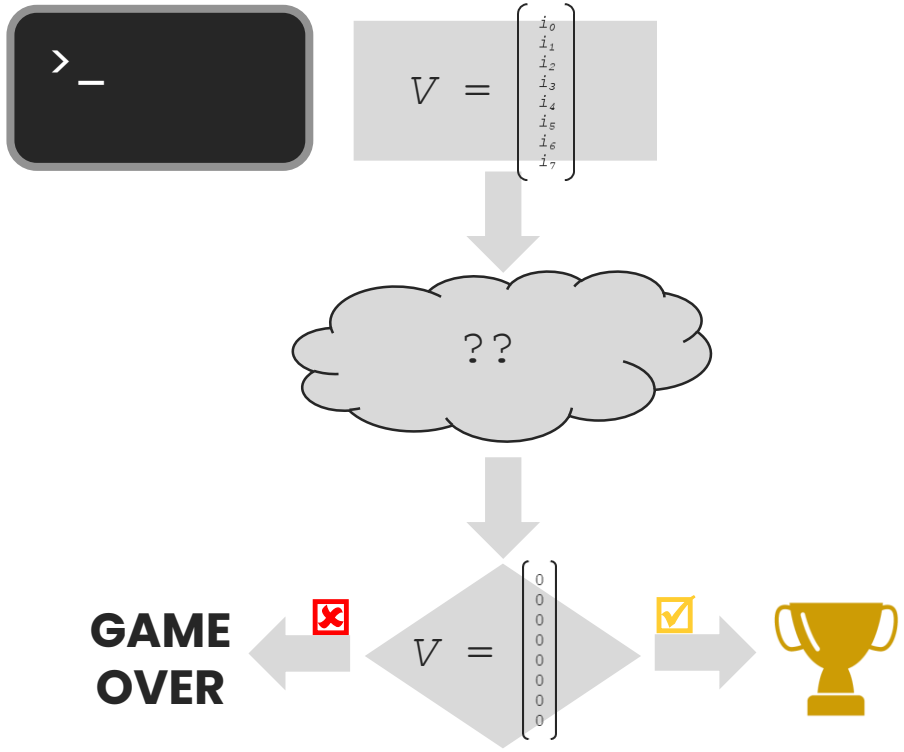
UNICORN

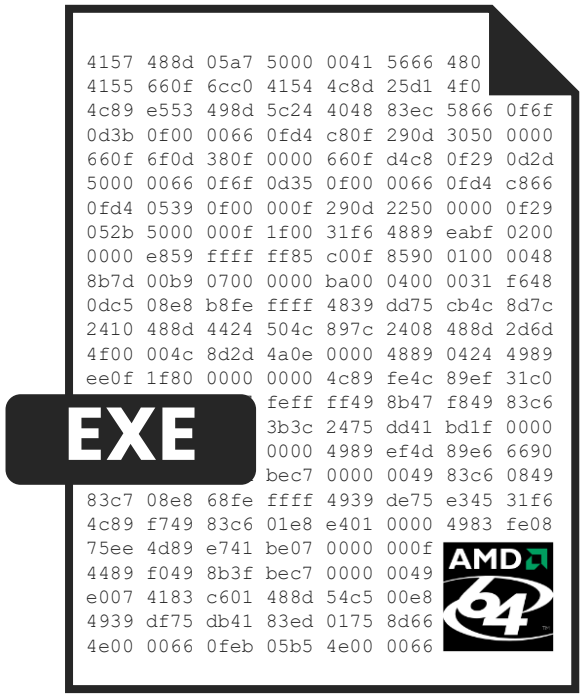
multi-architecture CPU emulator framework (based on QEMU)

Licorne in a nutshell

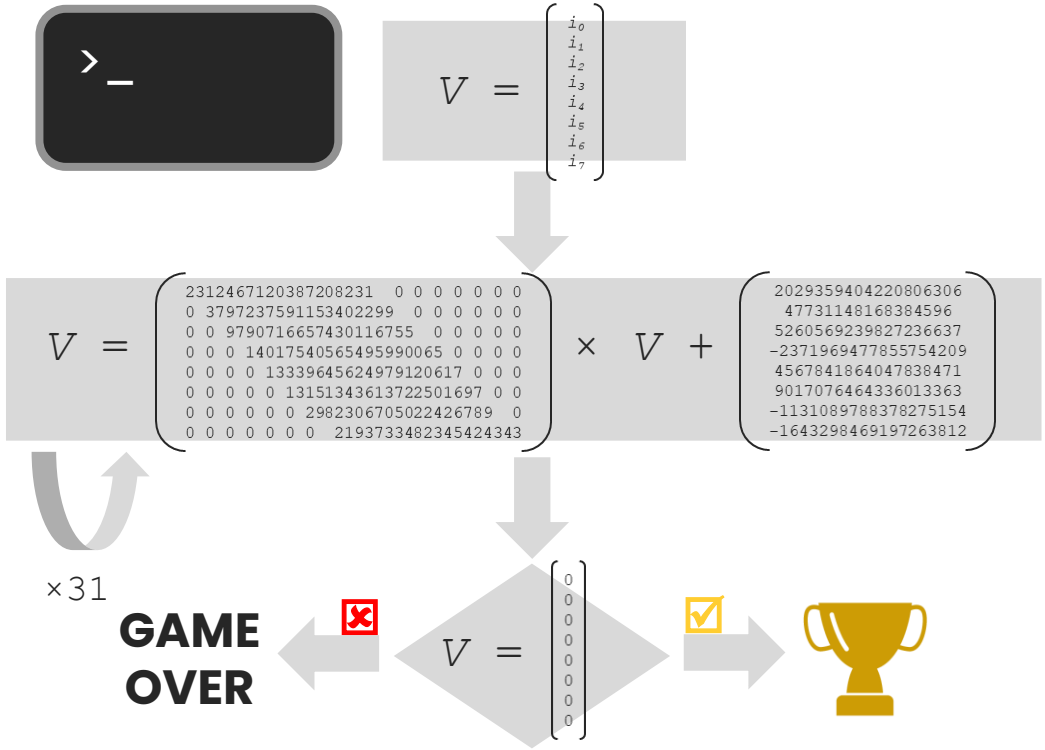


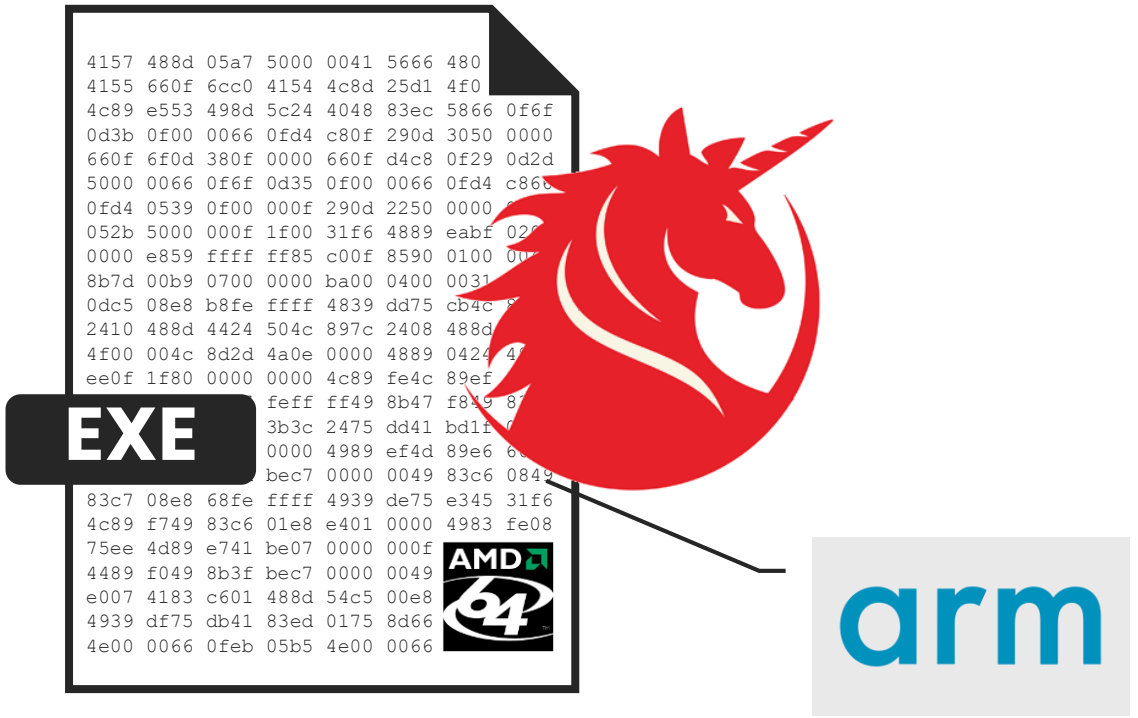
Licorne in a nutshell





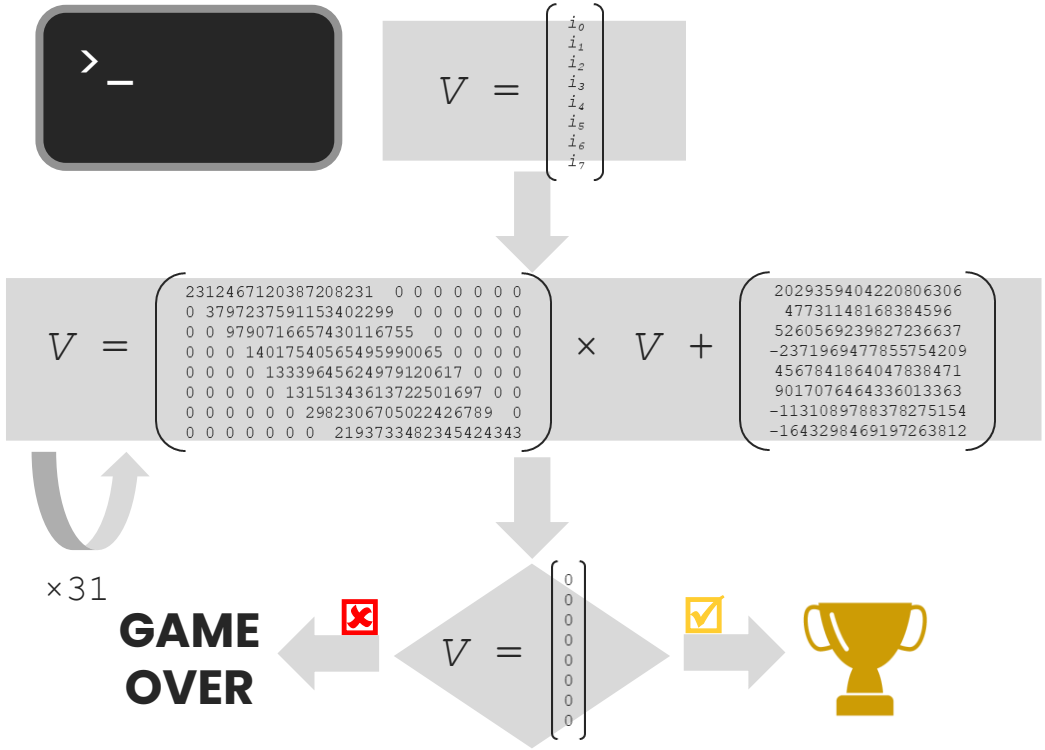
Licorne in a nutshell

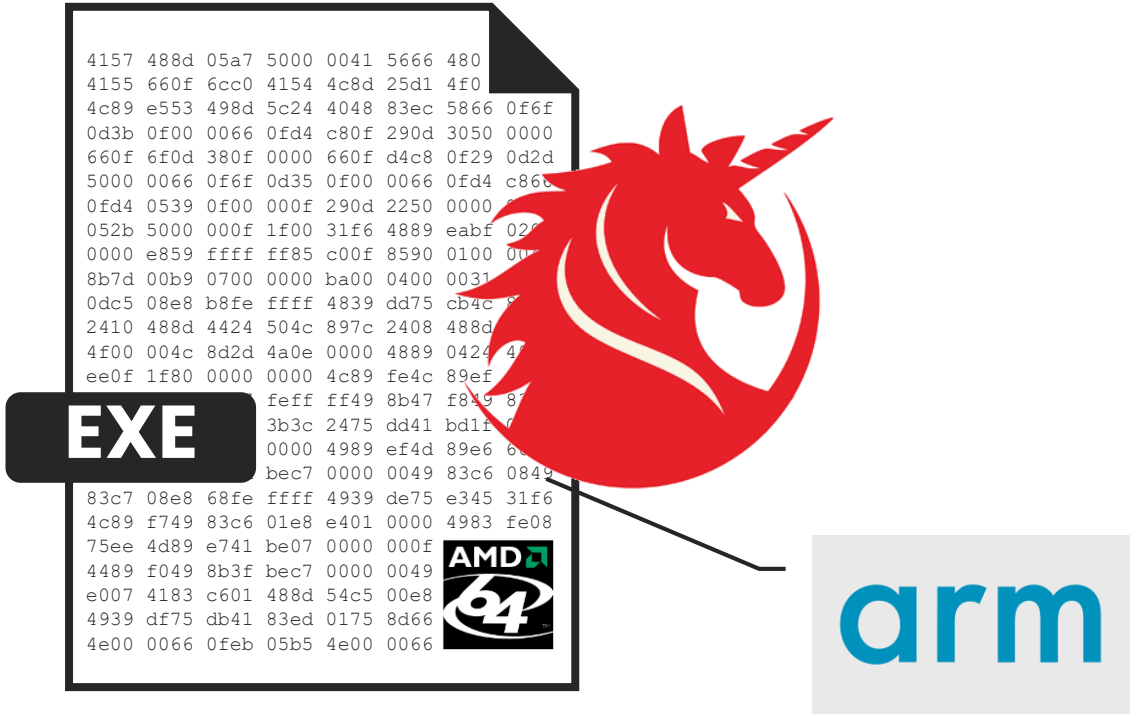




999 859 832 instructions

Licorne in a nutshell



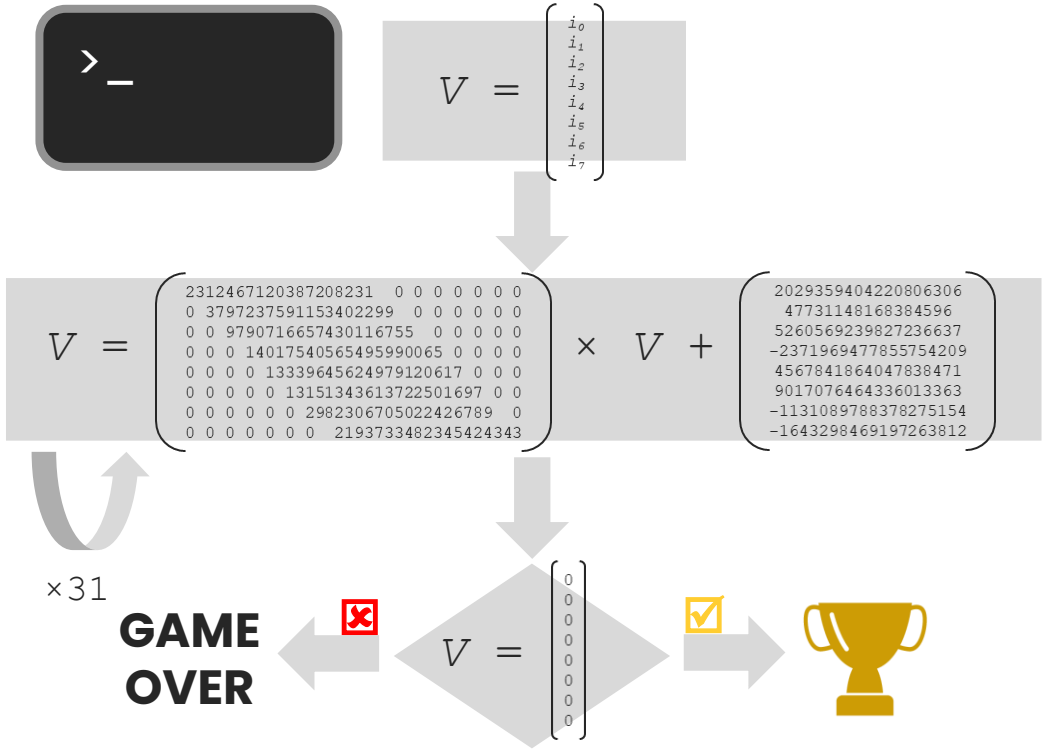


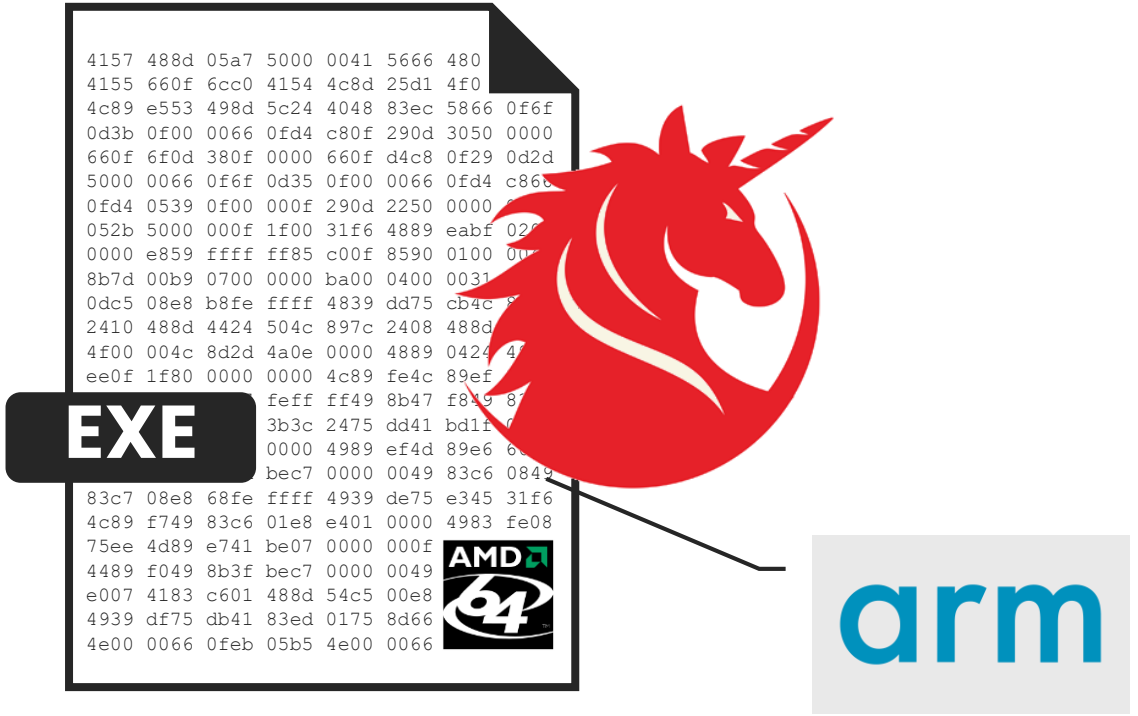
999 859 832 instructions **0.6**

RESOLUTION ✓

~3h

Licorne in a nutshell

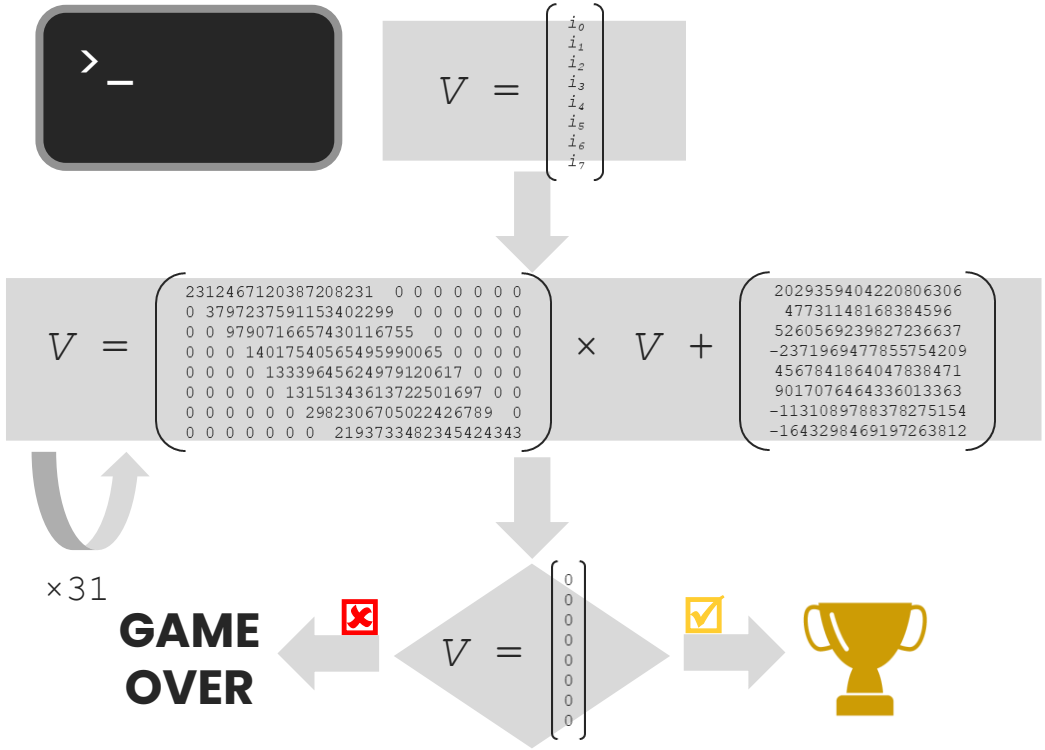


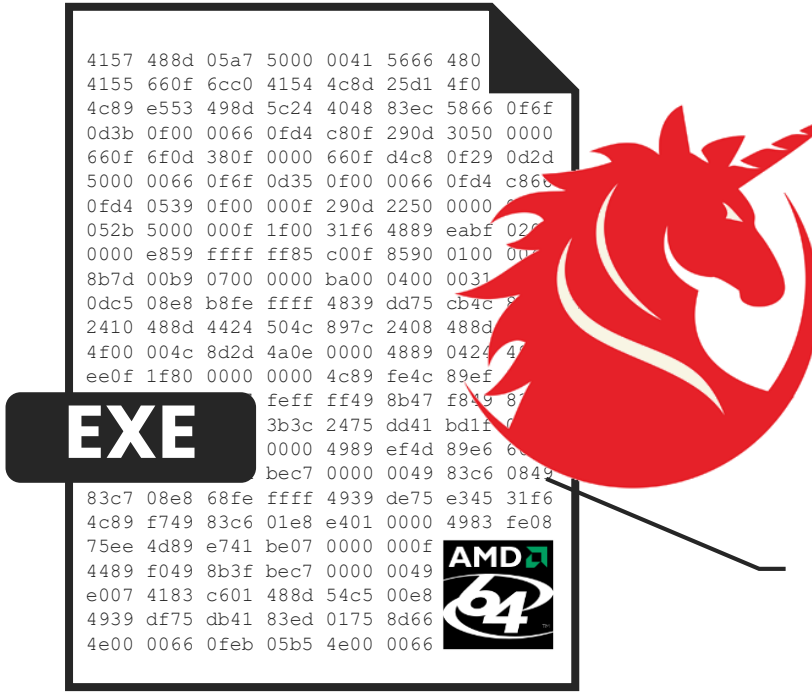


999 859 832 instructions

0.6	**0.8**
RESOLUTION ✓	RESOLUTION ✓
~3h	10m24

Licorne in a nutshell

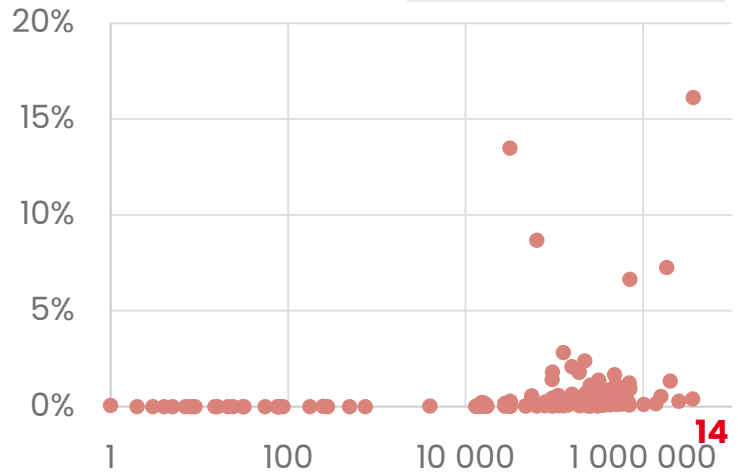




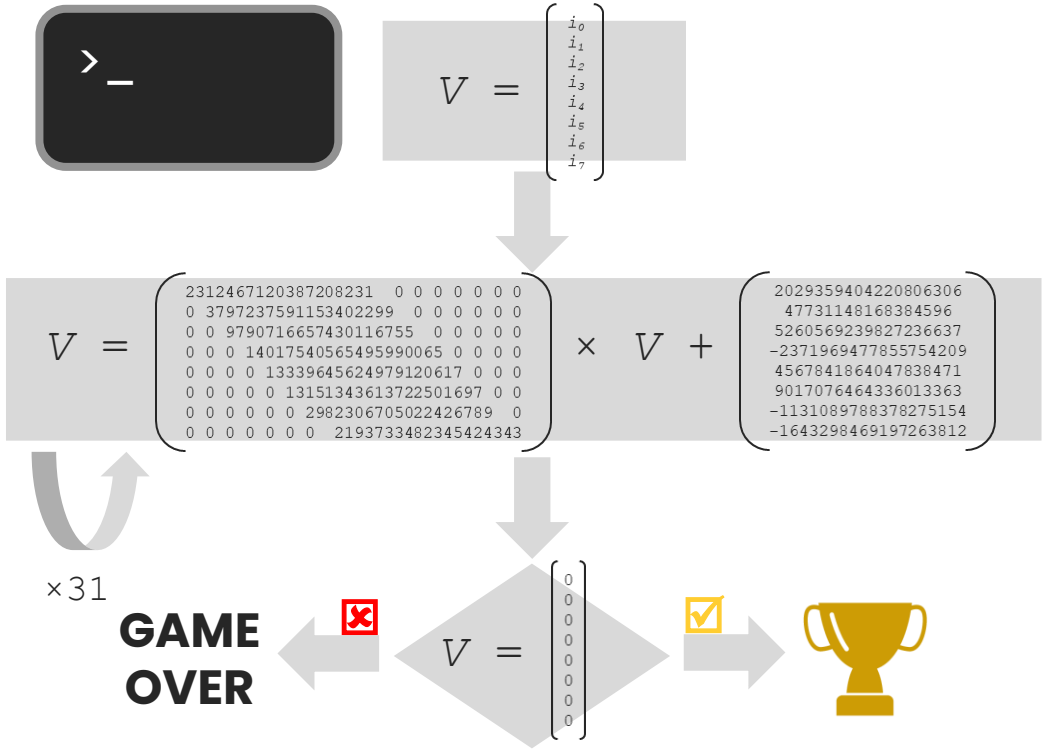
arm

999 859 832 instructions

0.6	**0.8**
RESOLUTION ✓	RESOLUTION ✓
~3h	10m24



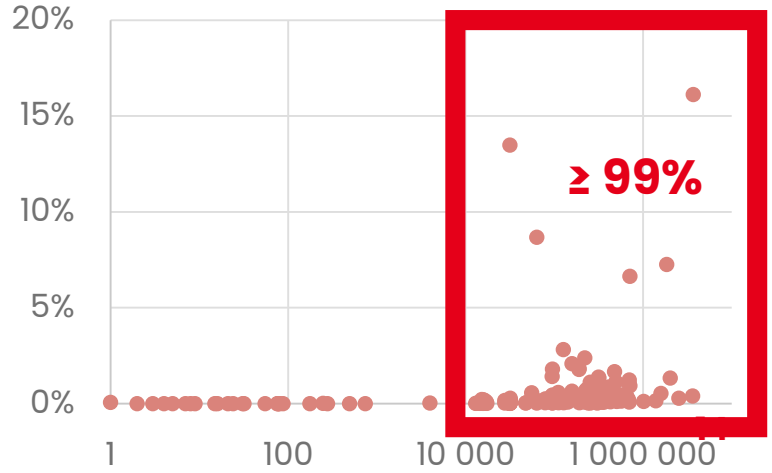
Licorne in a nutshell



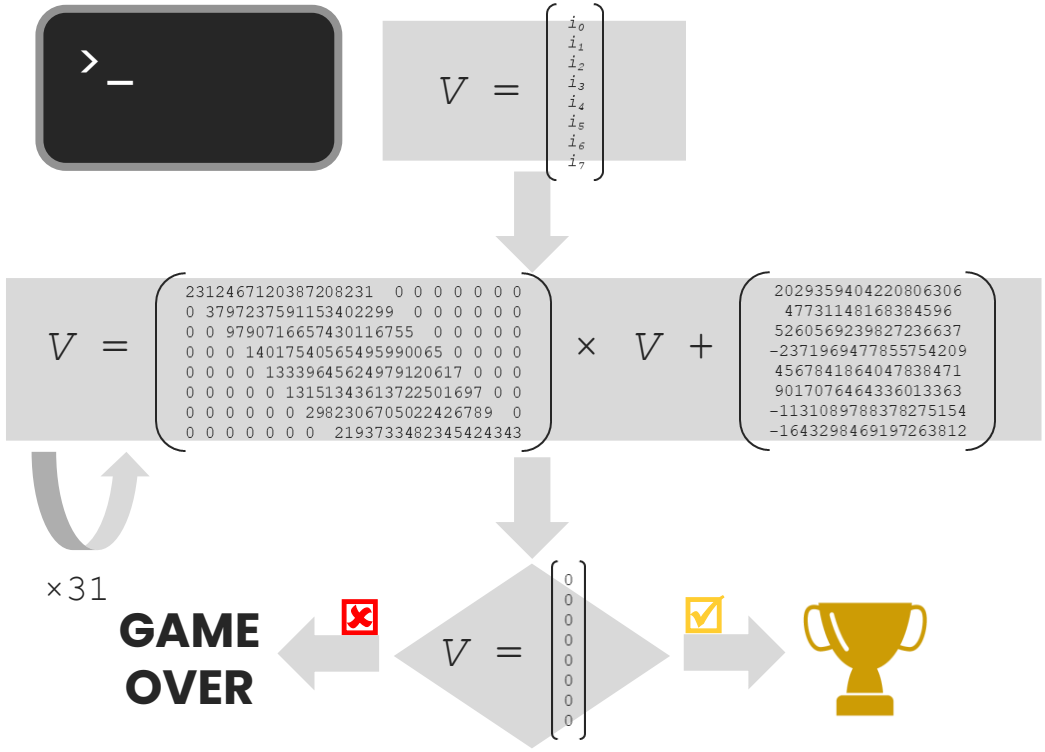
arm

999 859 832 instructions

0.6	**0.8**
RESOLUTION ✓	RESOLUTION ✓
~3h	10m24



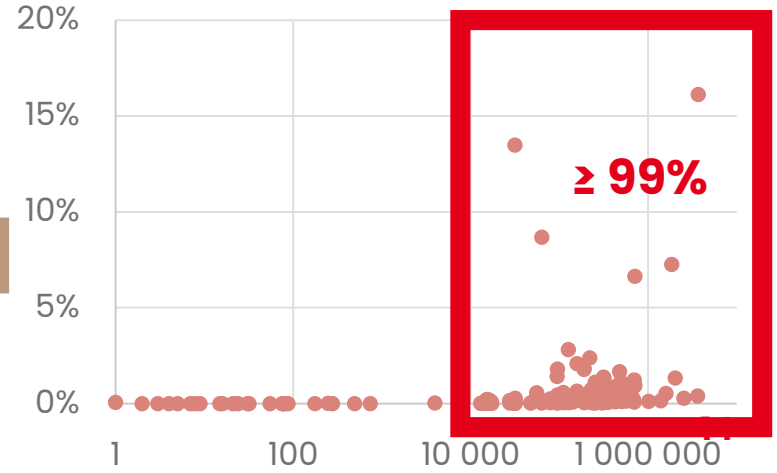
Licorne in a nutshell



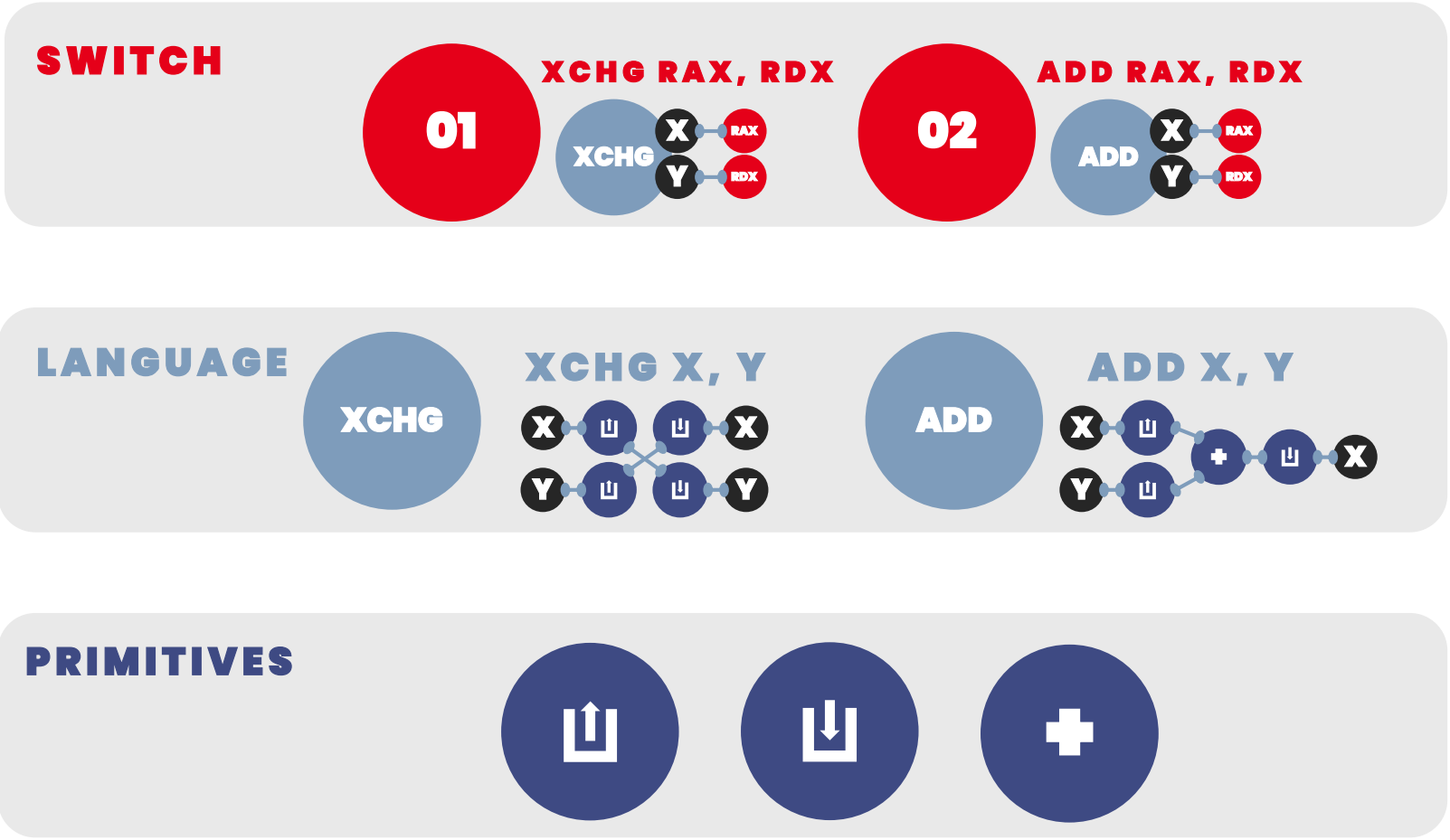
arm

999 859 832 instructions

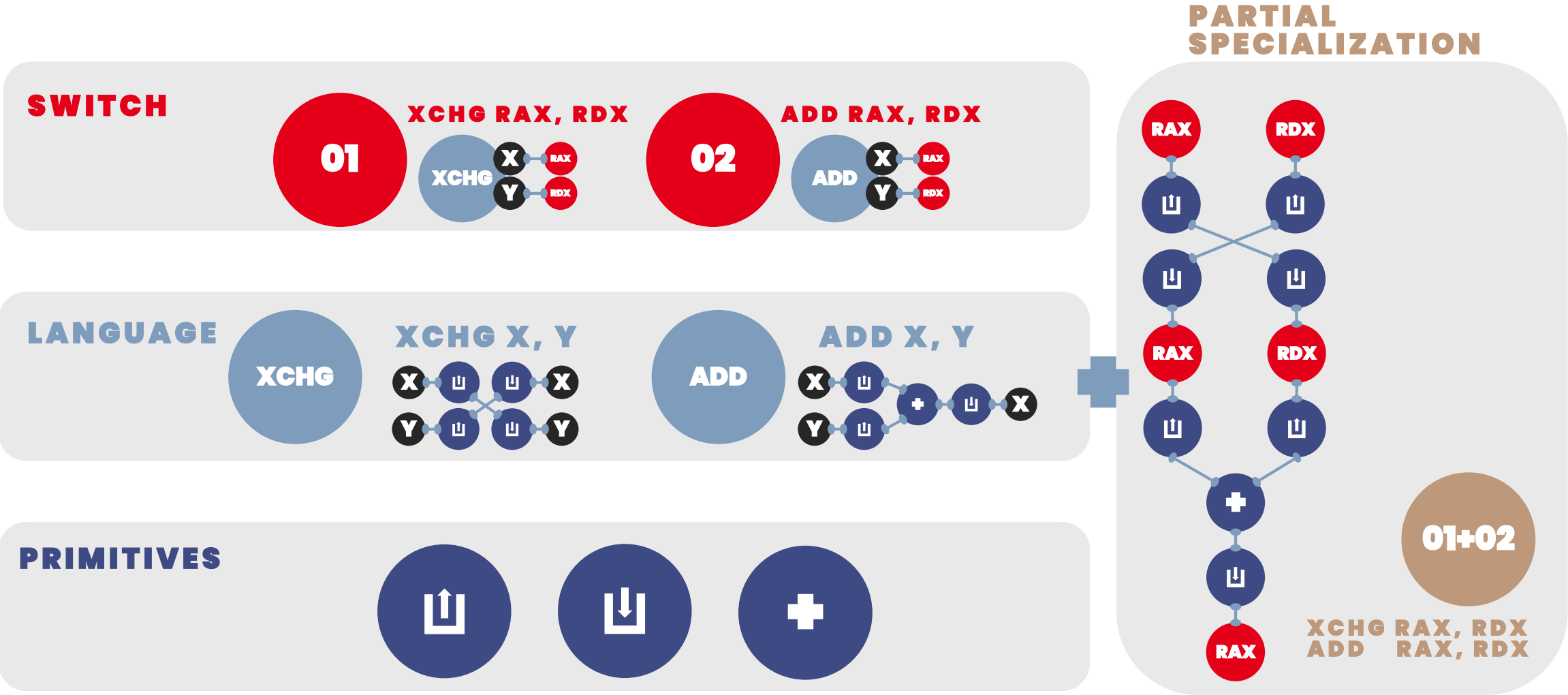
0.6	0.8	0.8+JIT
RESOLUTION ✓	RESOLUTION ✓	RESOLUTION ✓
~3h	10m24	7m15



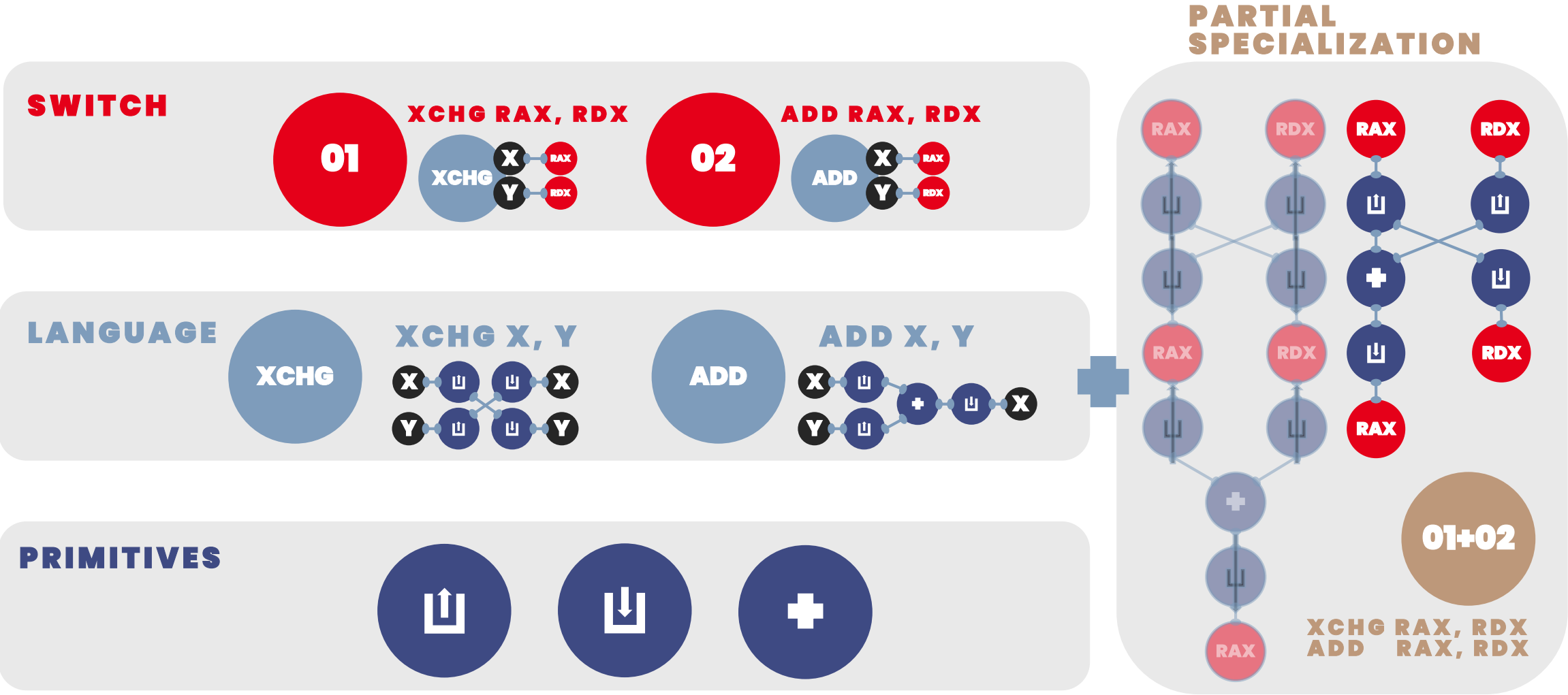
Specialization of an interpreter



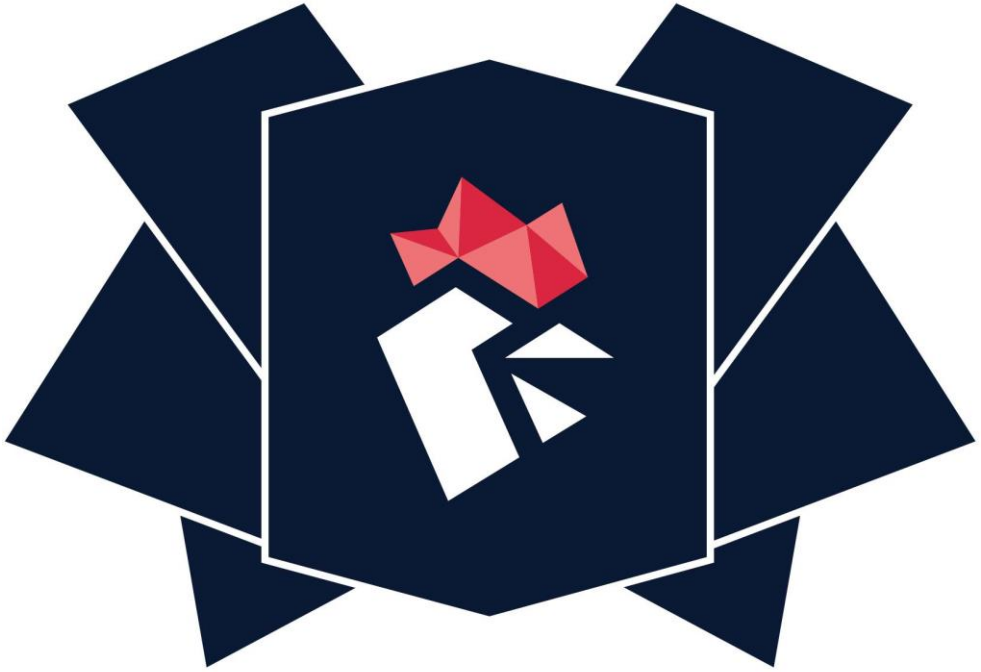
Specialization of an interpreter



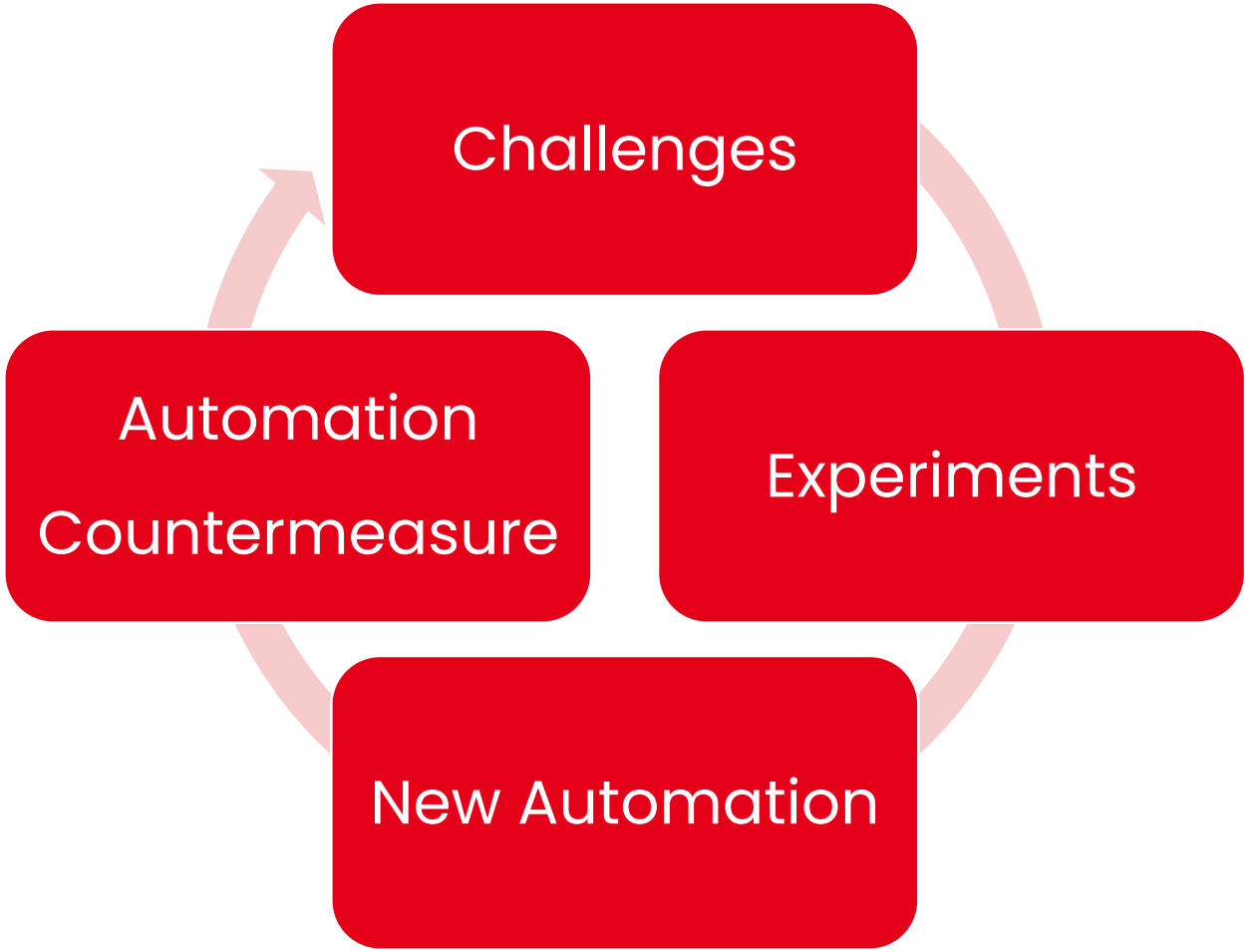
Specialization of an interpreter



Capture the Flag (CTF) challenges evolve



FRANCE CYBERSECURITY
CHALLENGE



Conclusion

01

MOUTHON RANKED HIGH

Mainly because it is pretty good at CTF challenges, but I like to believe it is also a little bit because of **BINSEC** ¯_(ツ)_/¯

Conclusion

01

MOUTHON RANKED HIGH

Mainly because it is pretty good at CTF challenges, but I like to believe it is also a little bit because of **BINSEC** ¯_(ツ)_/¯

02

BINSEC GETS BETTER

New development and refactoring end up benefitting other parts of the platform (e.g. CCS 2023)

Benchmark	Binsec/Rel	Binsec/Rel2	Speedup
AES-CBC-bearssl (BS)	16.77	0.31	x 54
AES-GCM-bearssl (BS)	53.32	0.48	x 111
PolyChacha-bearssl (CT)	9.72	0.18	x 53
PolyChacha-mbedtls	18.62	0.49	x 38
PolyChacha-openssl (EVP)	⏳	21.55	x 163+
Chacha20-openssl	0.77	0.09	x 8

Conclusion

01

MOUTHON RANKED HIGH

Mainly because it is pretty good at CTF challenges, but I like to believe it is also a little bit because of **BINSEC** ¯_(ツ)_/¯

02

BINSEC GETS BETTER

New development and refactoring end up benefitting other parts of the platform (e.g. CCS 2023)

JOBS

WE ARE HIRING

Benchmark	Binsec/Rel	Binsec/Rel2	Speedup
AES-CBC-bearssl (BS)	16.77	0.31	x 54
AES-GCM-bearssl (BS)	53.32	0.48	x 111
PolyChacha-bearssl (CT)	9.72	0.18	x 53
PolyChacha-mbedtls	18.62	0.49	x 38
PolyChacha-openssl (EVP)	⌛	21.55	x 163+
Chacha20-openssl	0.77	0.09	x 8