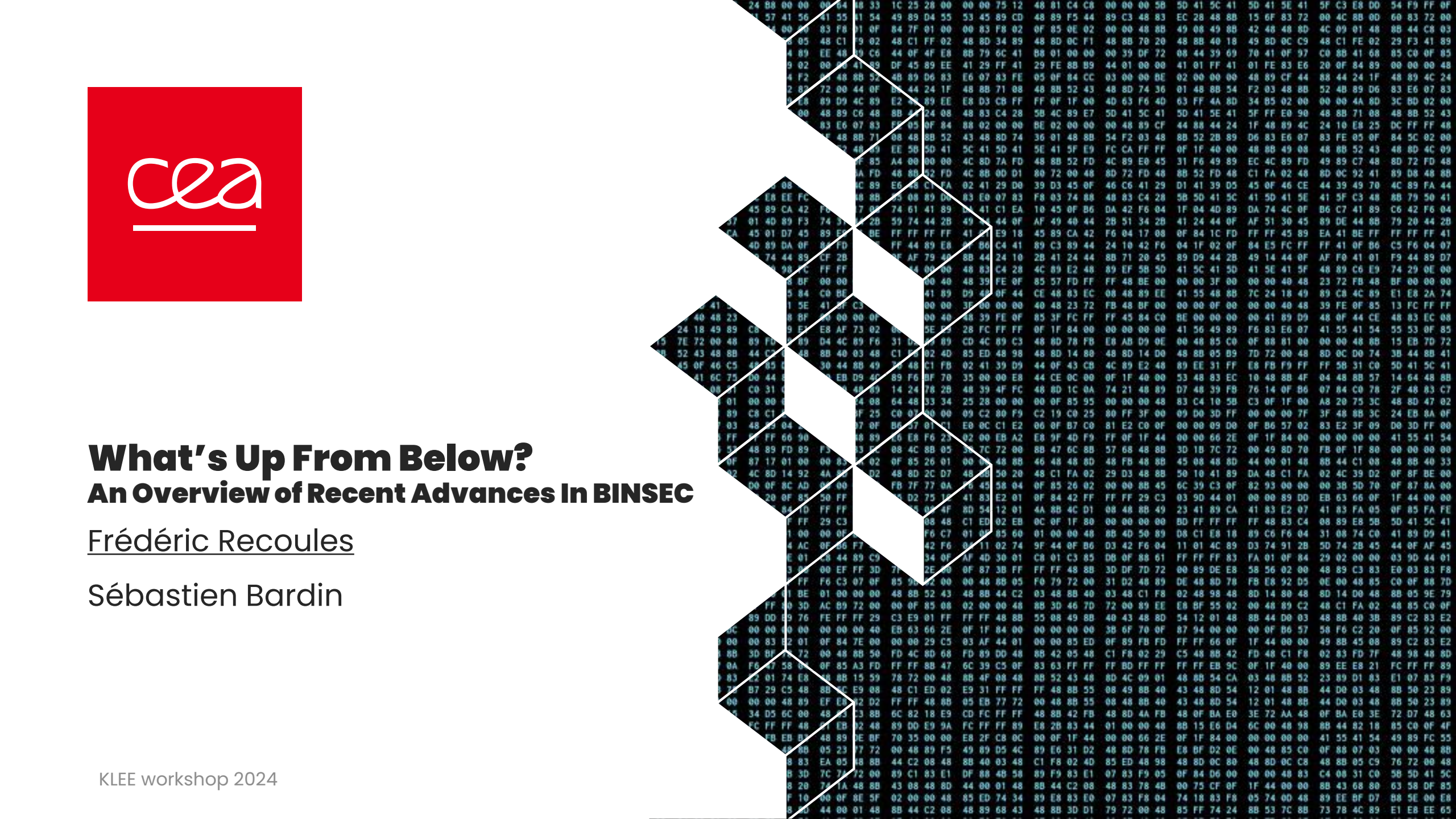


What's Up From Below?

An Overview of Recent Advances In BINSEC

Frédéric Recoules

Sébastien Bardin

KLEE workshop 2024

This talk in a nutshell

Goal

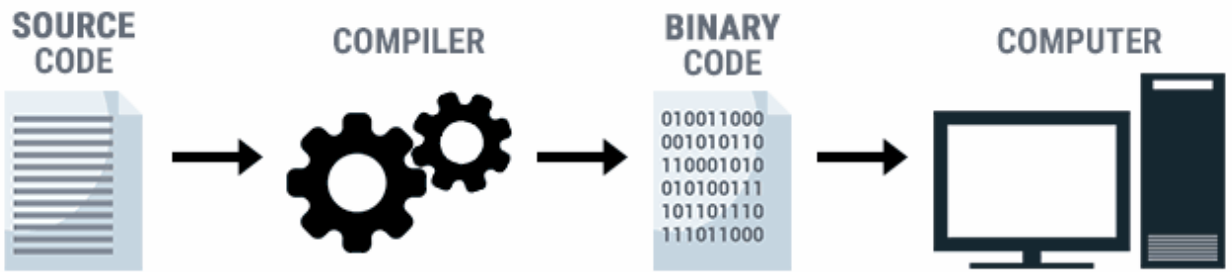
Introduce and discuss some design choices and recent improvements of the BINSEC SE

- Hope it can trigger discussions
- Beware: strongly biased toward binary code analysis

Highlights

- Introduction to BINSEC platform
- Under the Hood
  - Path predicate & Memory model
  - Efficient use of SMT solvers
  - JIT specialization of the interpreter
- Plugin extensibility

A need for binary level analysis



No source code

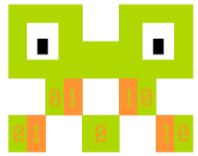
Malware

What You See Is Not What You Execute

BINSEC in a nutshell (since 2012)

Z3

Yices2



Bitwuzla

Solvers

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

Symbolic engine

BINSEC

Generic IR

Decoders

RISC-V

arm

ARM 64

PowerPC

AMD 64

intel

UNISIM Virtual Platforms .org

OCaml

56k LOC

Vulnerability Assessment

Security critical components

- Fault injection
- Side channel attack
- Attacker model

Bug finding

Supply chain

- Advanced fuzzing
- Test case generation

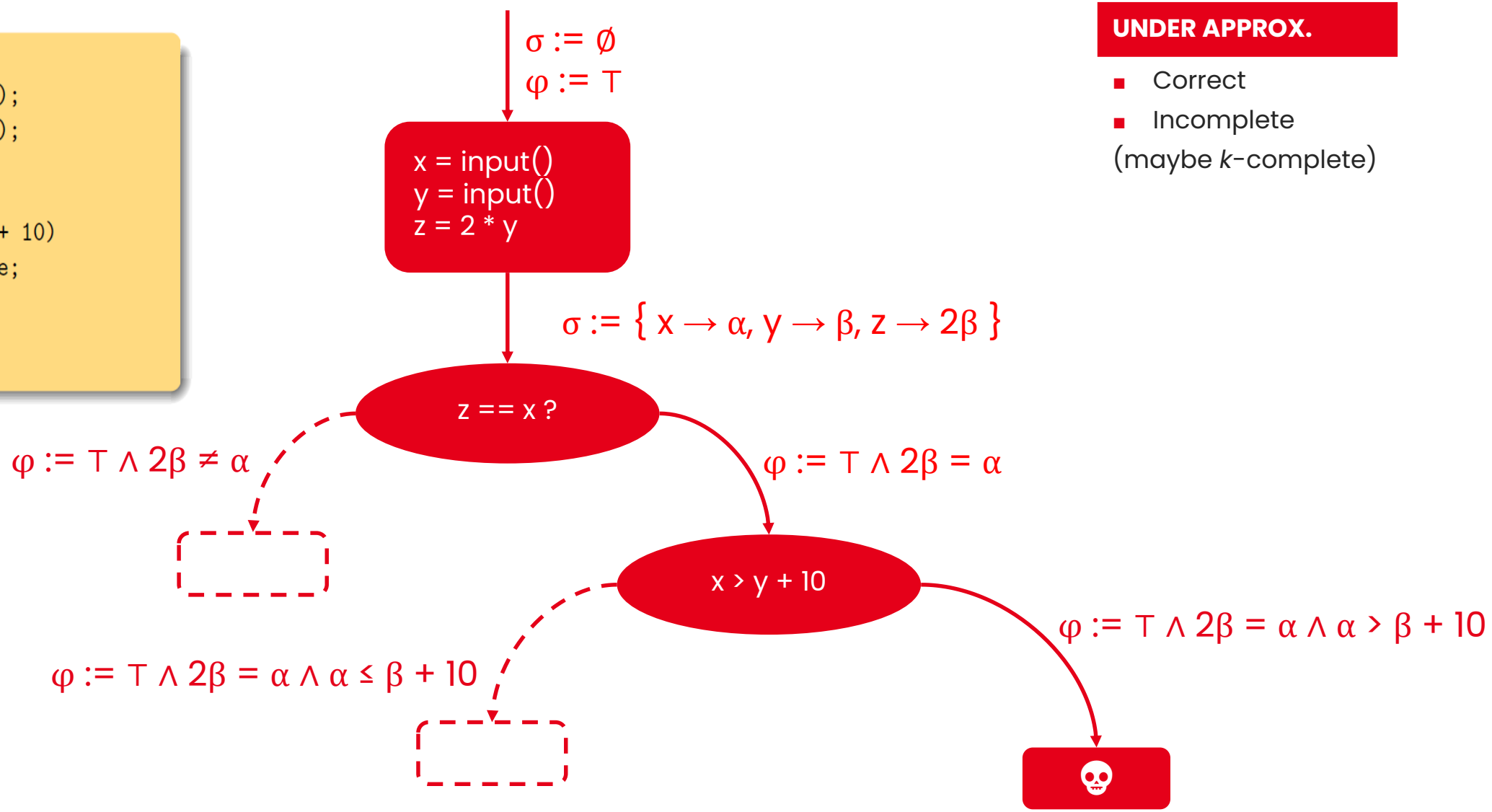
Reverse Engineering

Malware comprehension

- Capture The Flag
- Deobuscation
- Decompilation

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;
}
```

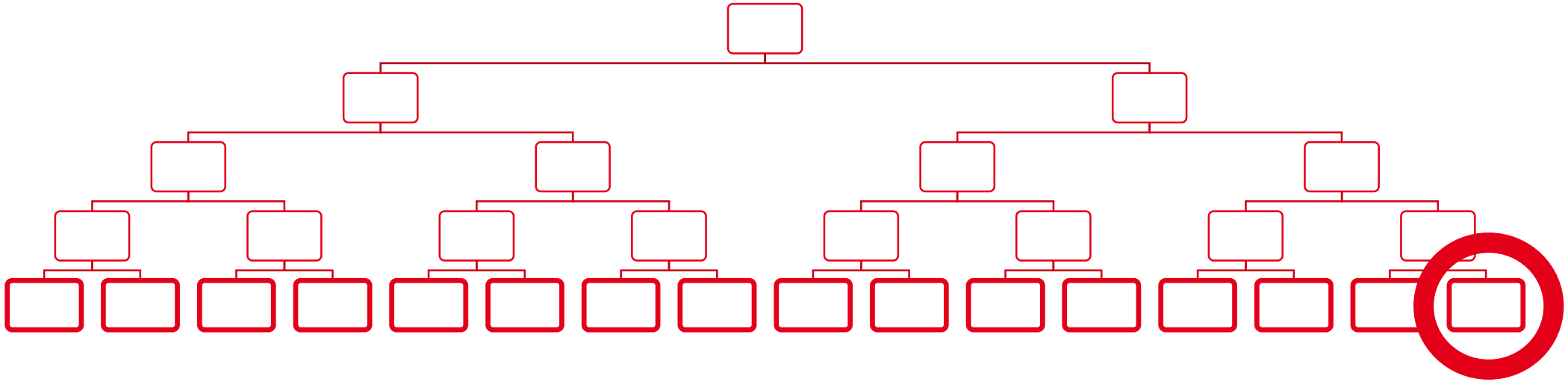


UNDER APPROX.

- Correct
- Incomplete

(maybe *k*-complete)

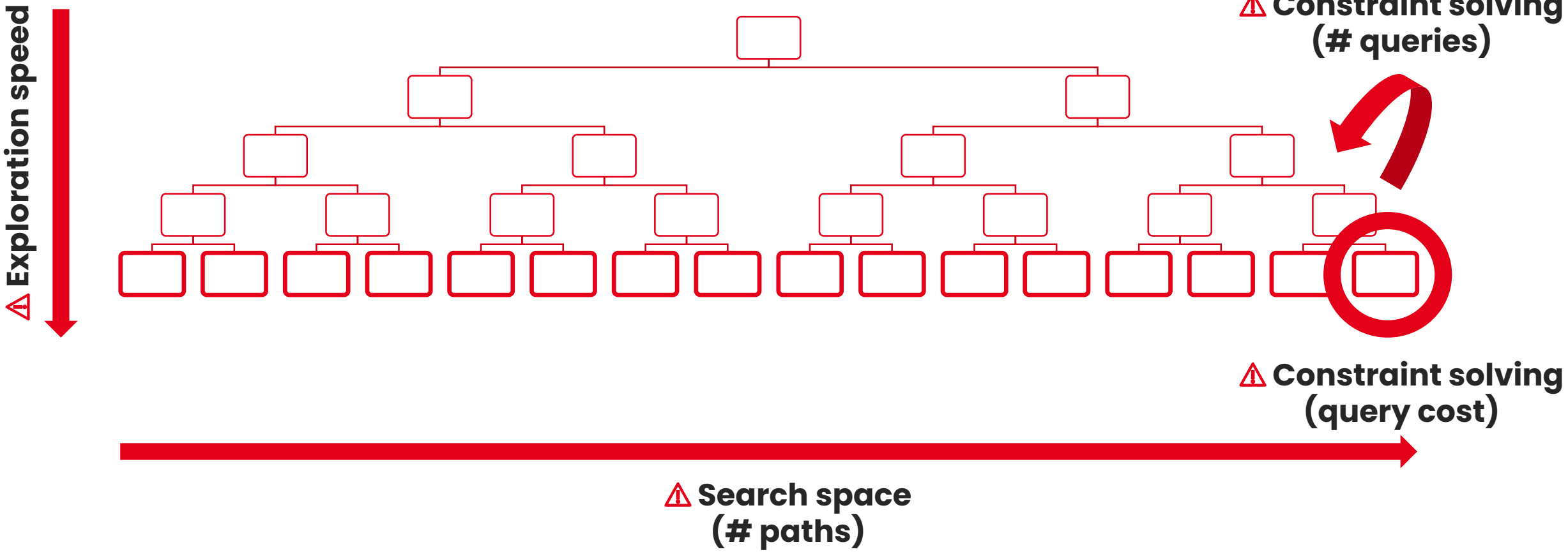
Theoretical and practical limits



⚠ Constraint solving (query cost)

⚠ Search space (# paths)

Theoretical and practical limits



Binary code difficulties

01

CFG RECONSTRUCTION

- CFG is hardly known in advance
- dynamic jump can go everywhere
- self-modifying code

02

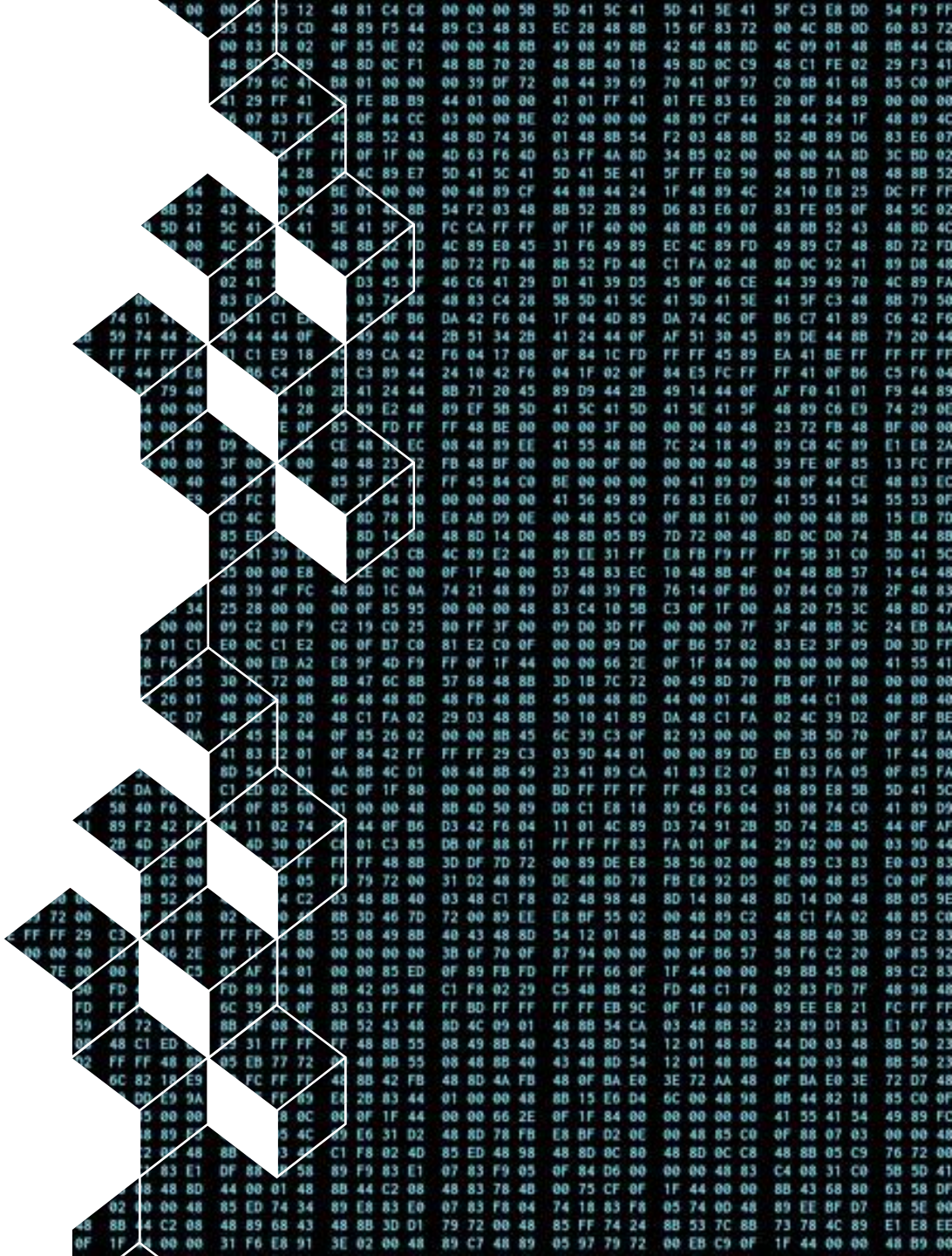
NO TYPE

- no object boundary
- any value can be used as an address
- memory is a single flat array of bytes

03

UBIQUITOUS MEMORY

- no variable scope
- unconstrained store can clobber everything



1. The BINSEC designs

- Introduction to BINSEC platform
- Under the Hood
  - **Path predicate & Memory model**
  - Efficient use of SMT solvers
  - JIT specialization of the interpreter
- Plugin extensibility

```
mov     edx, 0x1
shl     edx, cl
or      eax, edx
add     ecx, 0x1
cmp     ecx, 0x8
je      .label
```

```
(define-fun  edx0 () (_ BitVec 32) #x00000001)
(declare-fun ecx0 () (_ BitVec 32))
(define-fun  edx1 () (_ BitVec 32) (bvshl edx0 ecx0))
(define-fun   zf0 () Bool (= edx1 #x00000000))
(declare-fun eax0 () (_ BitVec 32))
(define-fun  eax1 () (_ BitVec 32) (bvor eax0 edx1))
(define-fun   zf1 () Bool (= eax1 #x00000000))
(define-fun  ecx1 () (_ BitVec 32) (bvadd ecx0 #x00000001))
(define-fun   zf2 () Bool (= ecx1 #x00000000))
(define-fun   zf3 () Bool (= ecx1 #x00000008))
(assert zf3)
(check-sat)
```

Revisiting path predicate construction

The good old one

- One definition per assignment
- Powerful preprocessing
  - Local rewriting rules
  - Single-use value propagation / inlining
    - Avoid formula size explosion
  - Pruning
  - Memory simplification

✚ −

GENERIC

Well defined optimization passes

SCALING

Too many definitions
Intermediate variables disable rewriting

Reworked symbolic store

✚ 

Do less, get more !

TAKING ADVANTAGE OF A FUNCTIONAL LANGUAGE

Natural sharing (all values are reference)
Automatic garbage collector

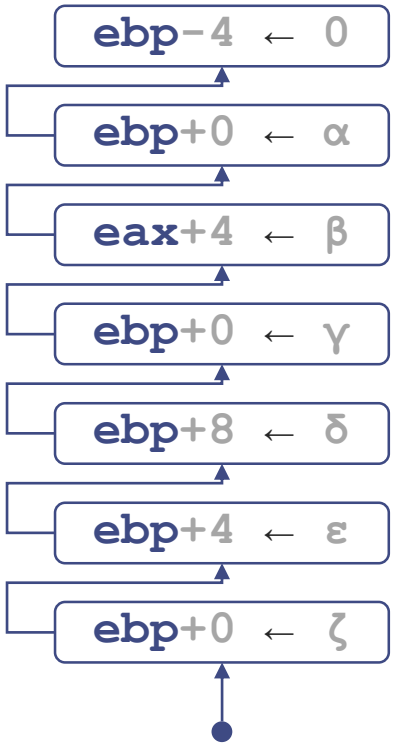
- Greedy inlining (DAG)
  - Keep only the latest definition
  - Fully enable rewriting rules
  - Pruning for free (GC)
- Lazy hash consing
  - Structural comparison melds equal sub-terms (union-find'like)
  - Introduce only essential definitions during export (SMT)

Revisiting the memory (Farinier et al. 18)

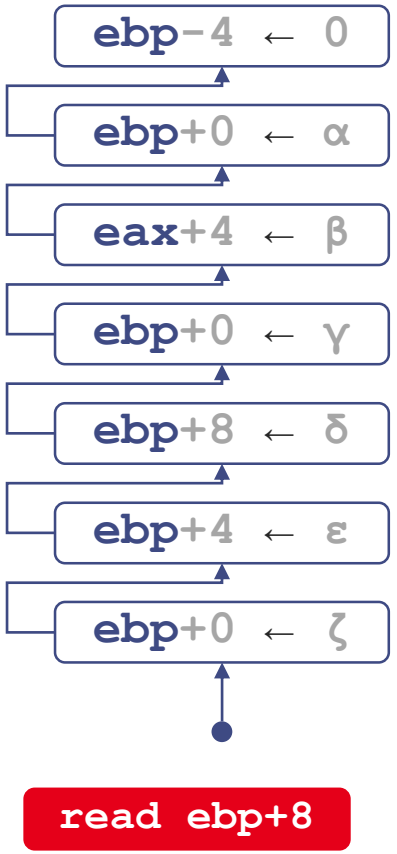
1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution



Revisiting the memory (Farinier et al. 18)



1

MEMORY SIMPLIFICATIONS

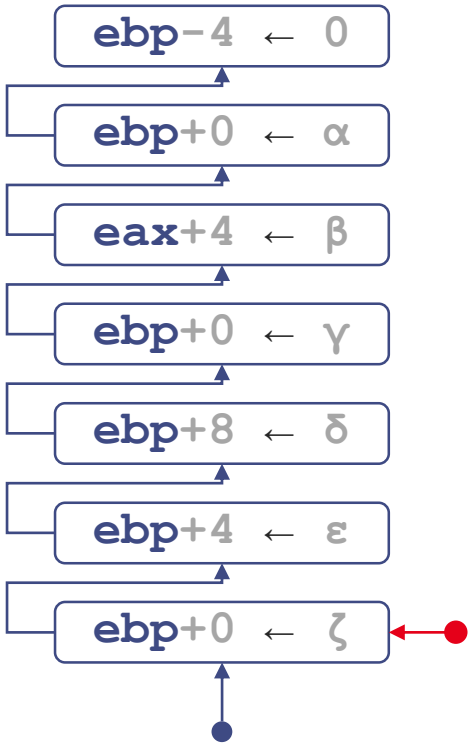
Read-over-Write, Write-over-Write
Astract domain disequality resolution

Revisiting the memory (Farinier et al. 18)

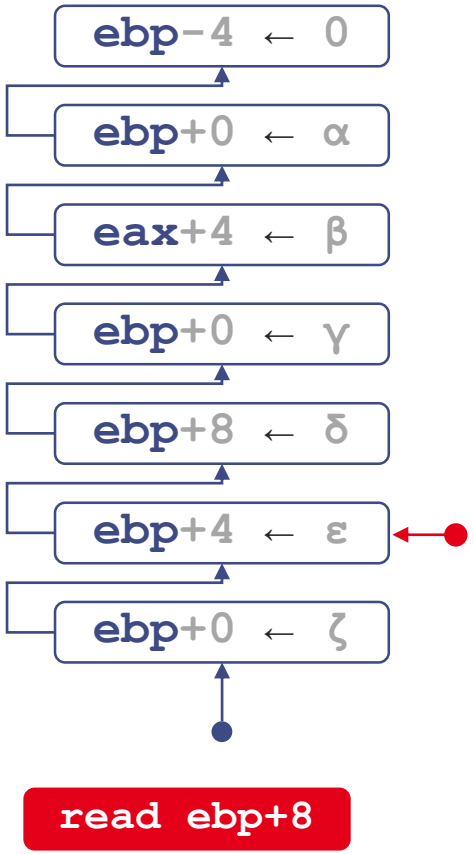
1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution



Revisiting the memory (Farinier et al. 18)



1

MEMORY SIMPLIFICATIONS

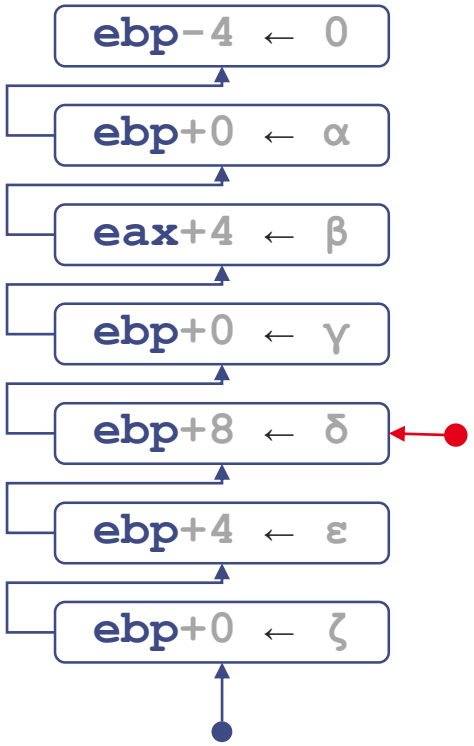
Read-over-Write, Write-over-Write
Astract domain disequality resolution

Revisiting the memory (Farinier et al. 18)

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution



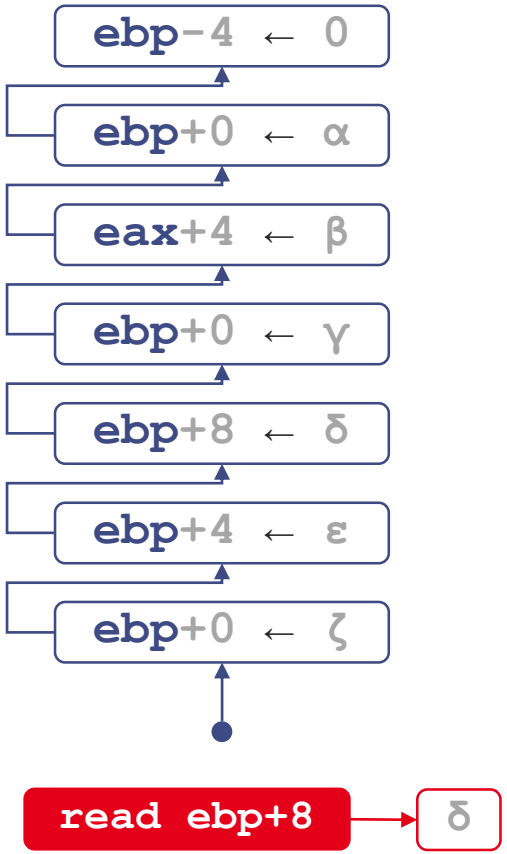
`read ebp+8`

Revisiting the memory (Farinier et al. 18)

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

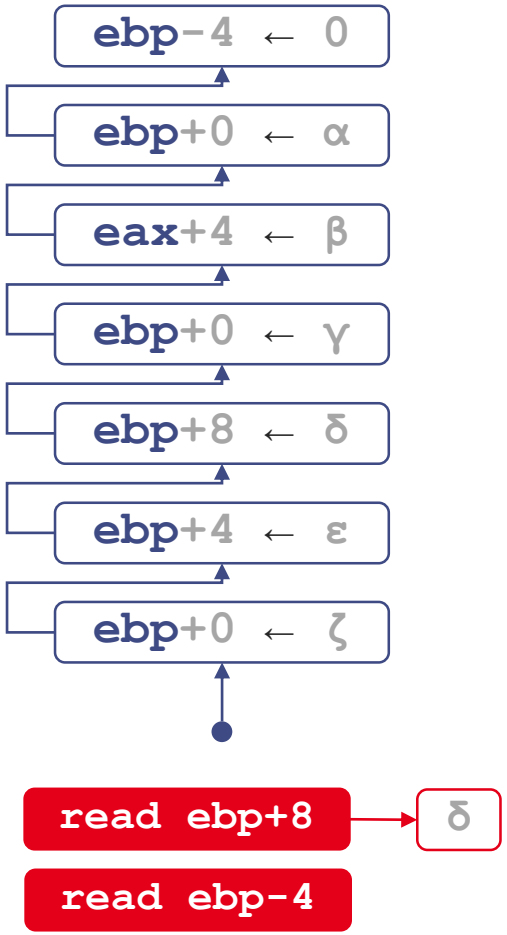


Revisiting the memory (Farinier et al. 18)

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

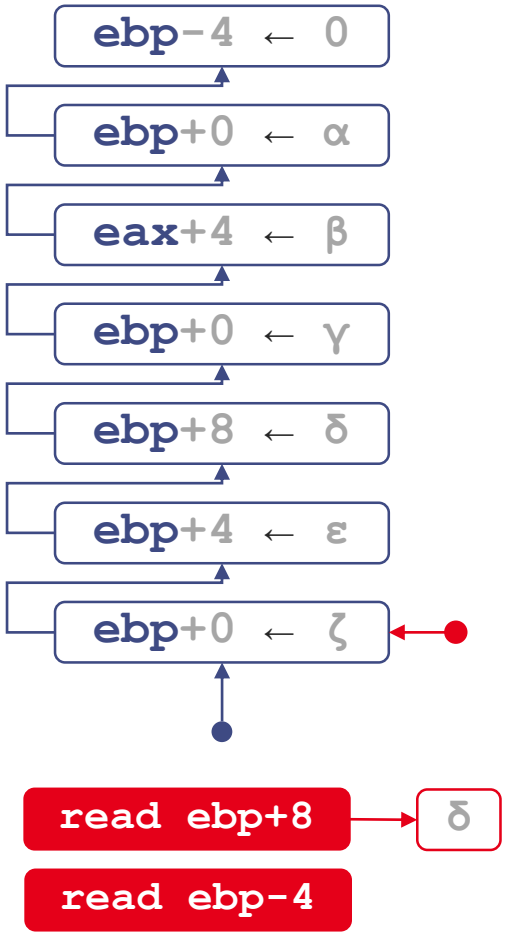


Revisiting the memory (Farinier et al. 18)

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

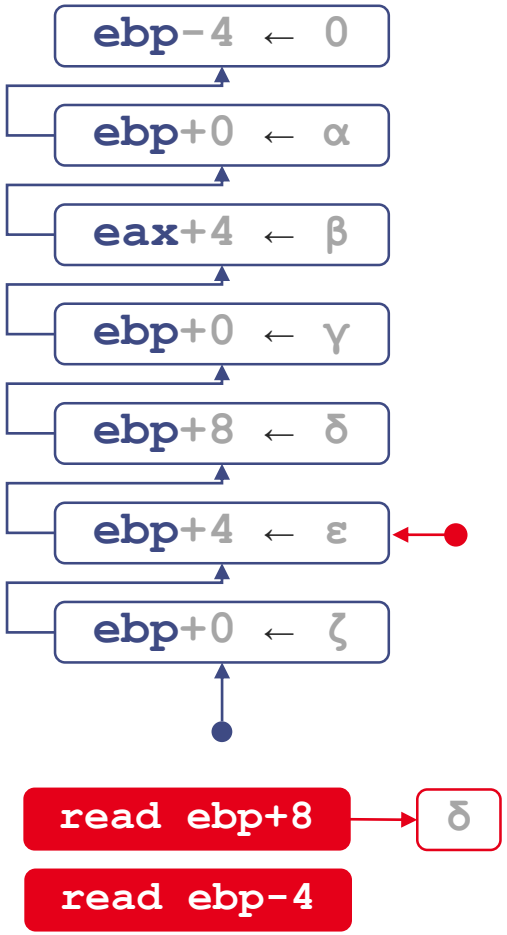


Revisiting the memory (Farinier et al. 18)

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

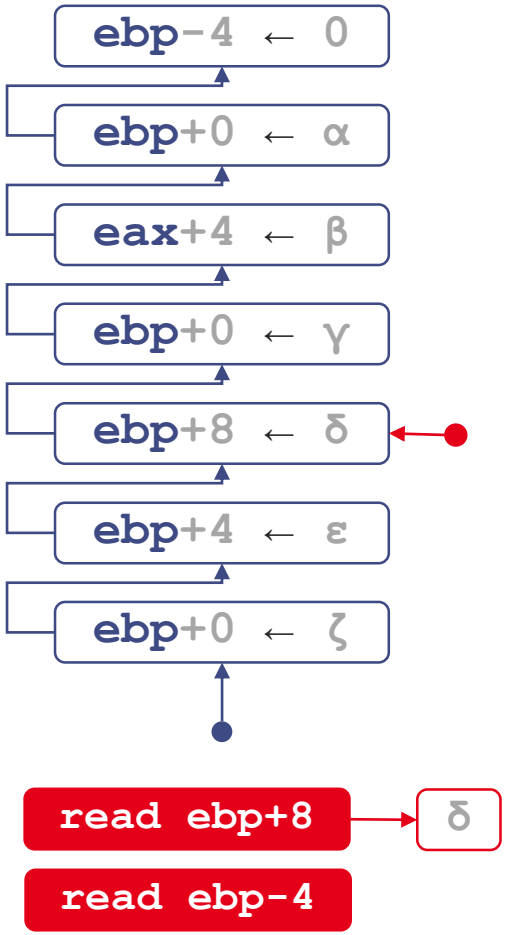


Revisiting the memory (Farinier et al. 18)

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

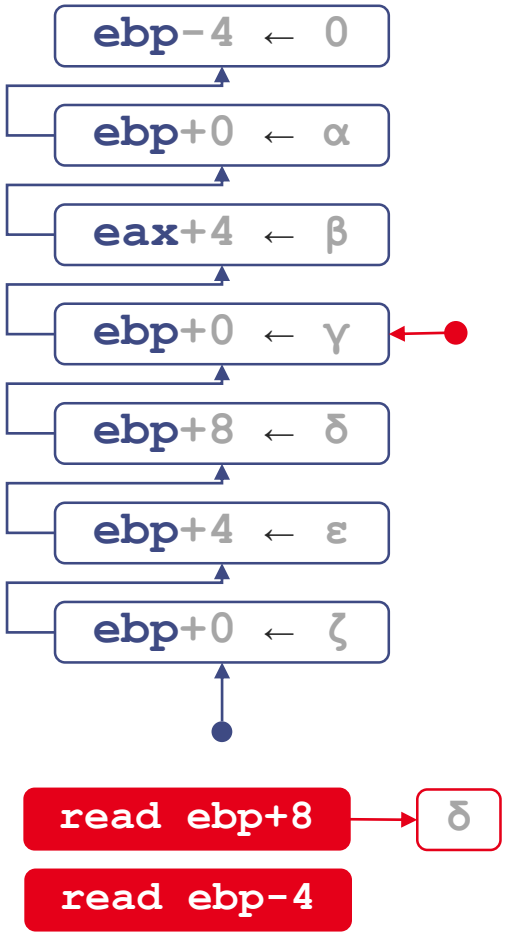


Revisiting the memory (Farinier et al. 18)

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

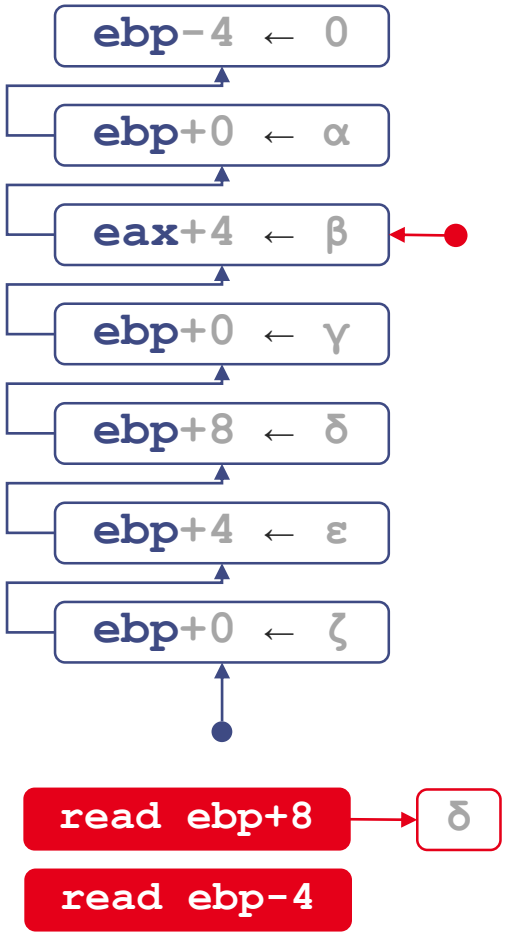


Revisiting the memory (Farinier et al. 18)

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

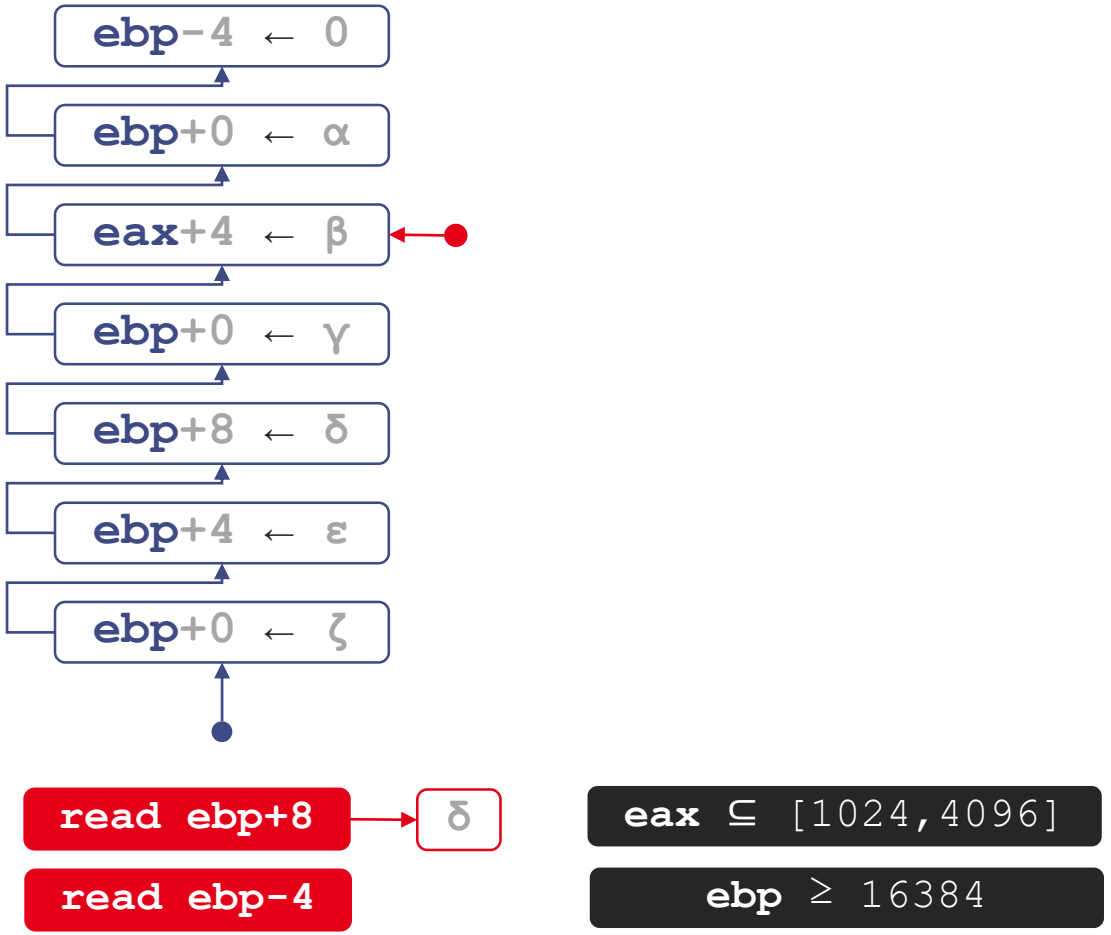


Revisiting the memory (Farinier et al. 18)

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

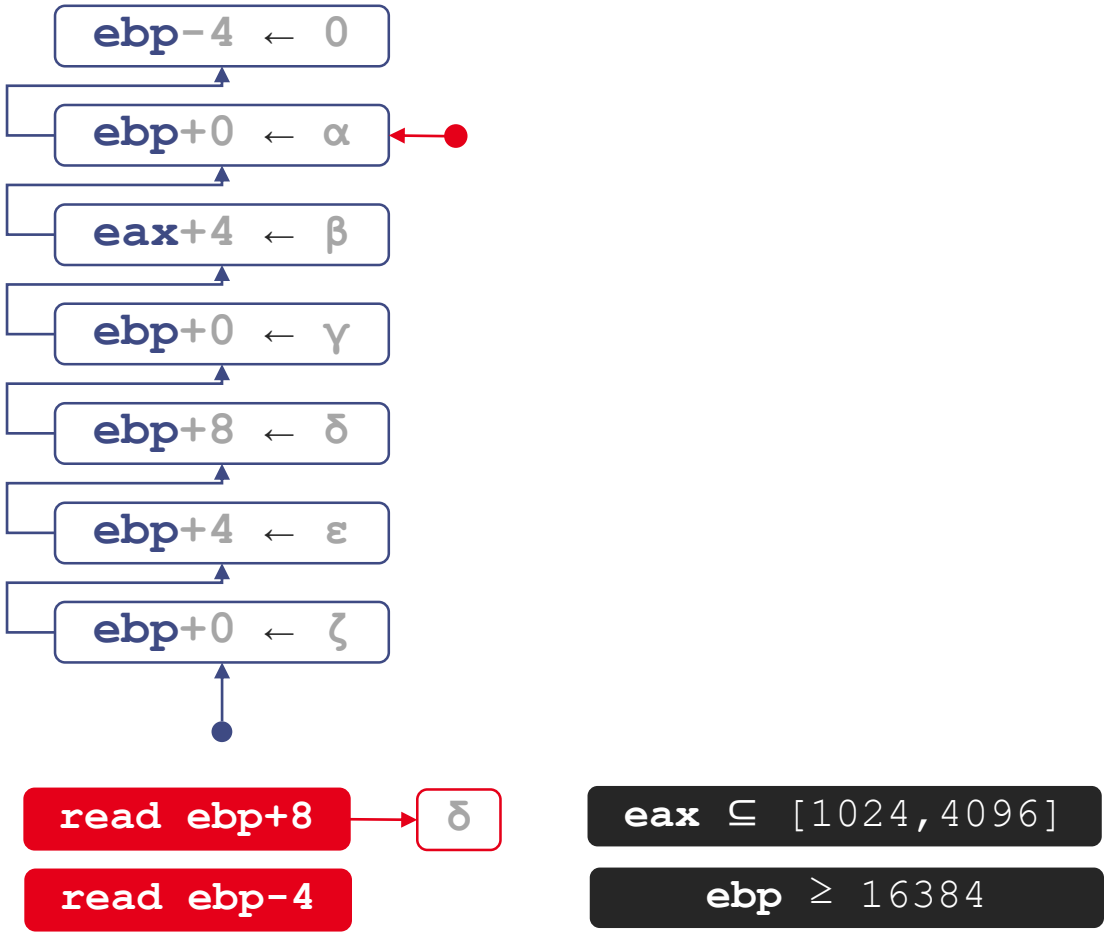


Revisiting the memory (Farinier et al. 18)

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

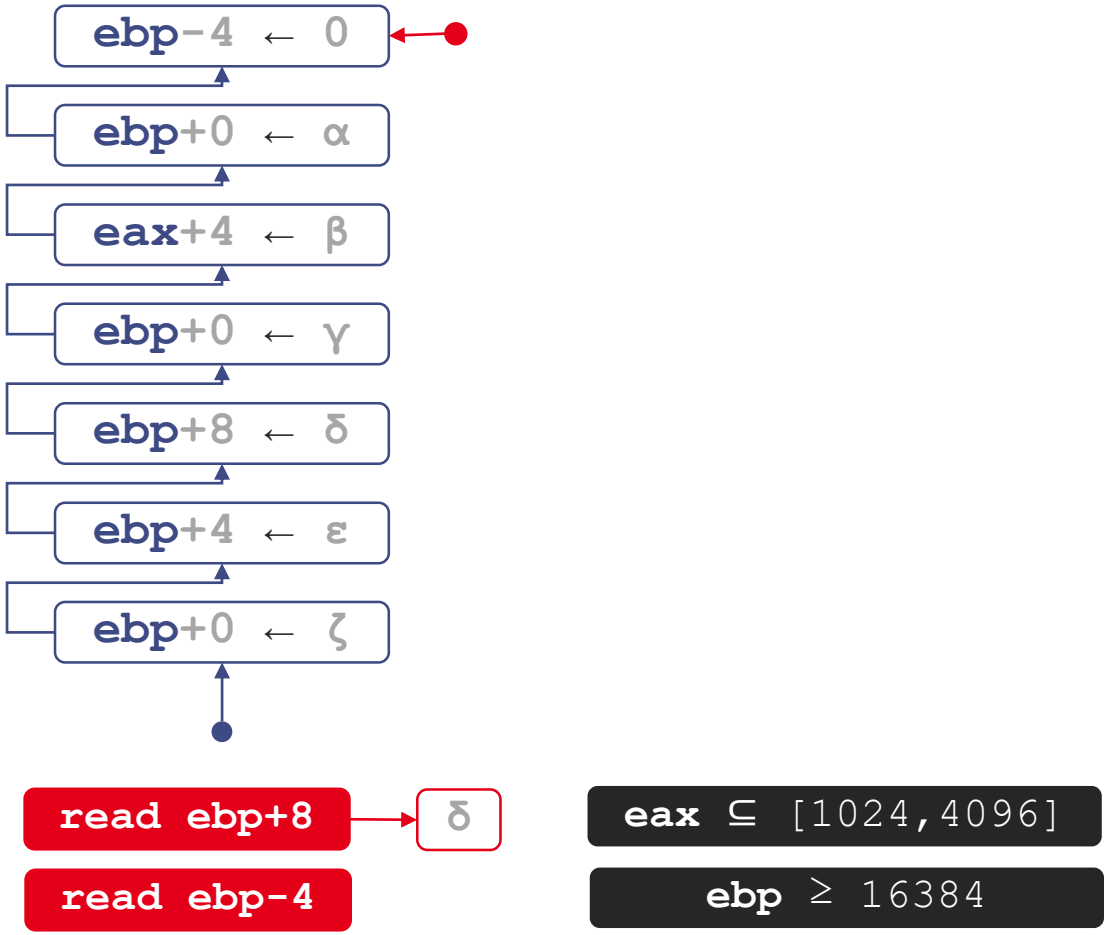


Revisiting the memory (Farinier et al. 18)

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

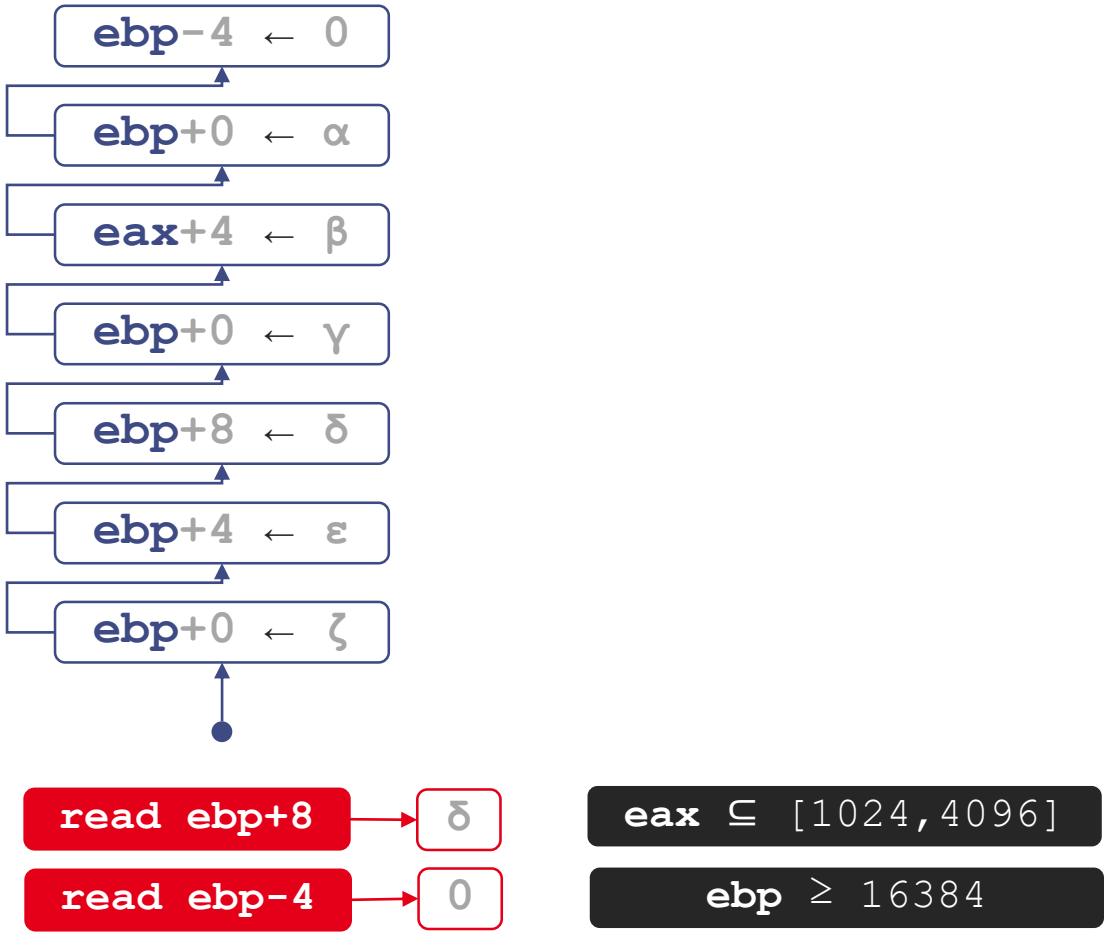


Revisiting the memory (Farinier et al. 18)

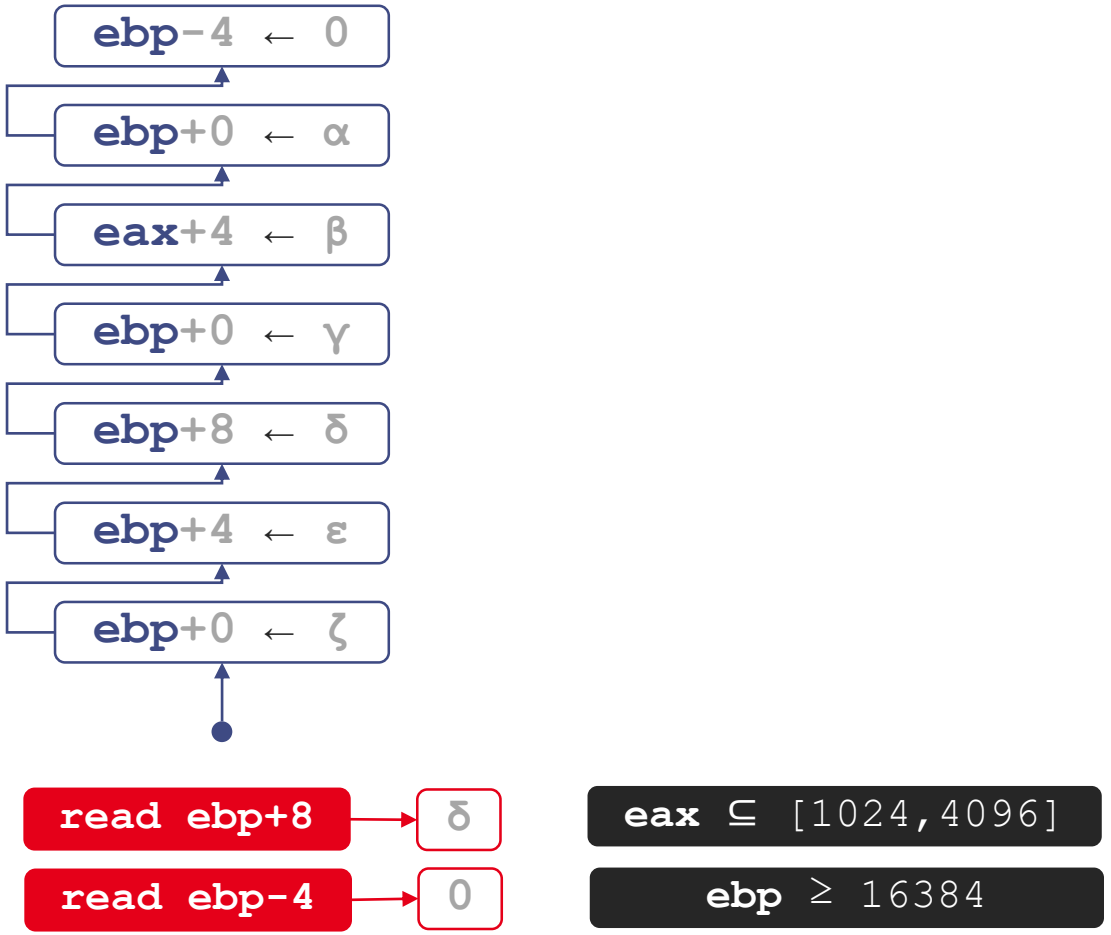
1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution



Revisiting the memory (Farinier et al. 18)



1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

2

MEMORY LAYER

Base address with Offset-value map

3

WRITE HISTORY

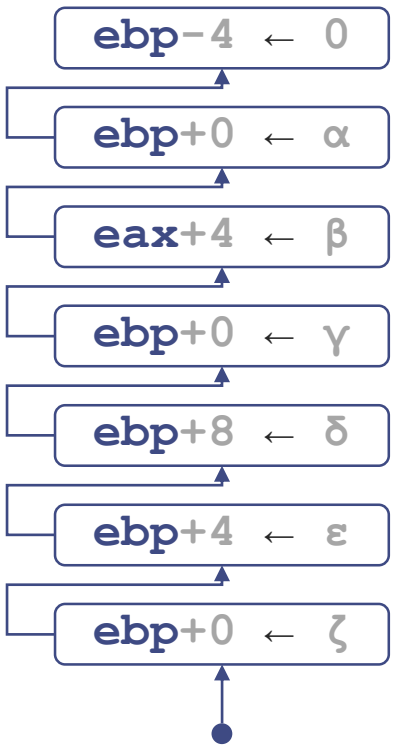
Sequential list of layers of non-comparable base addresses

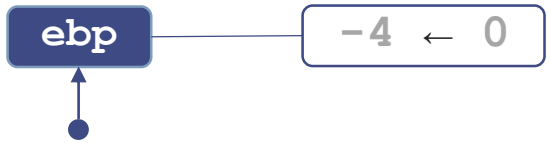
4

LAZY INITIALIZATION

File backed chunk of initial memory
Iterative memory refinement lemma

Revisiting the memory (Farinier et al. 18)





eax ⊆ [1024,4096]

ebp ≥ 16384

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

2

MEMORY LAYER

Base address with Offset-value map

3

WRITE HISTORY

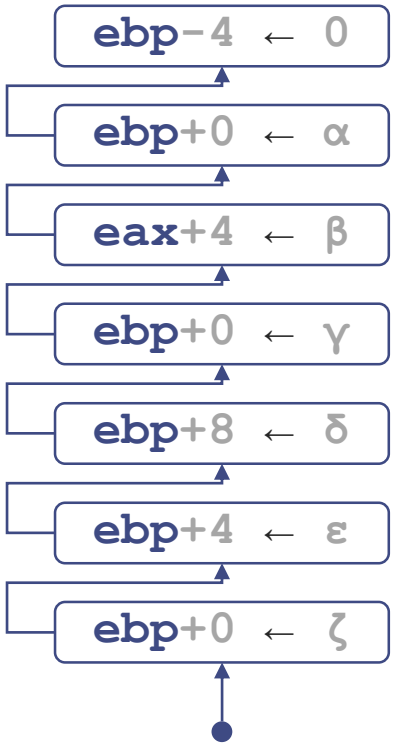
Sequential list of layers of non-comparable base addresses

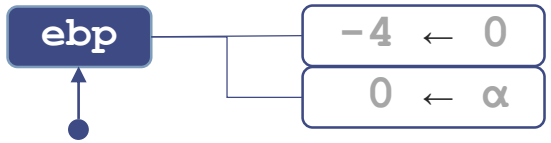
4

LAZY INITIALIZATION

File backed chunk of initial memory
Iterative memory refinement lemma

Revisiting the memory (Farinier et al. 18)





`eax ⊆ [1024,4096]`

`ebp ≥ 16384`

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

2

MEMORY LAYER

Base address with Offset-value map

3

WRITE HISTORY

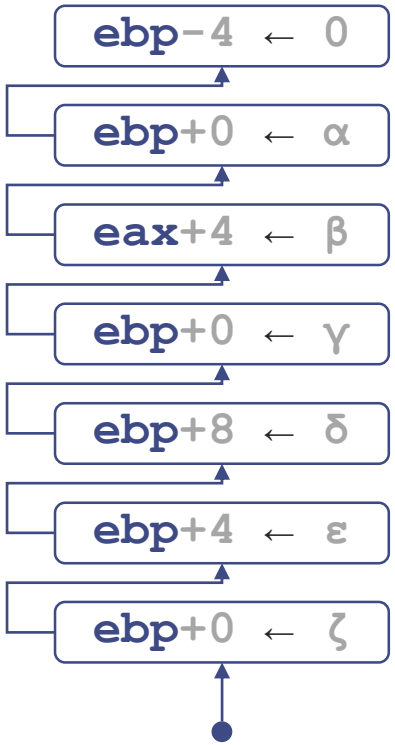
Sequential list of layers of non-comparable base addresses

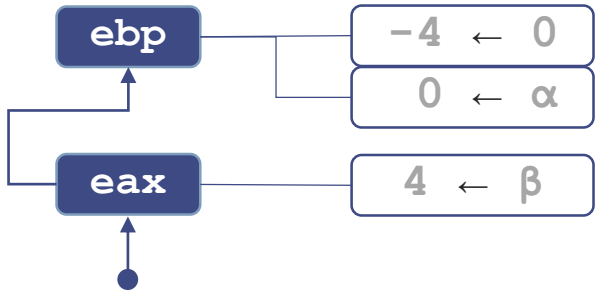
4

LAZY INITIALIZATION

File backed chunk of initial memory
Iterative memory refinement lemma

Revisiting the memory (Farinier et al. 18)





`read ebp+8` → δ

`read ebp-4` → 0

`eax ⊆ [1024,4096]`

`ebp ≥ 16384`

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

2

MEMORY LAYER

Base address with Offset-value map

3

WRITE HISTORY

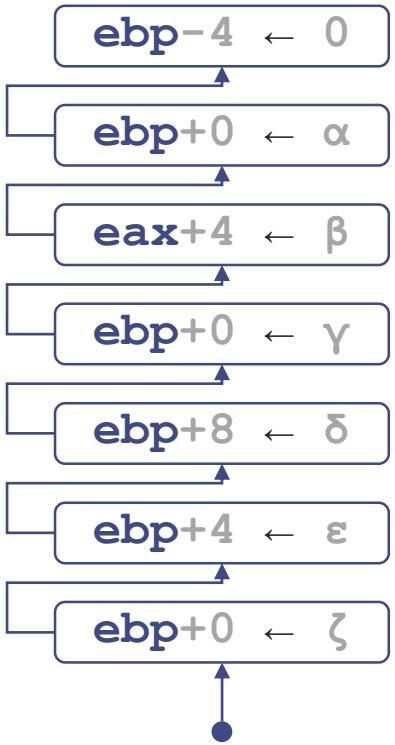
Sequential list of layers of non-comparable base addresses

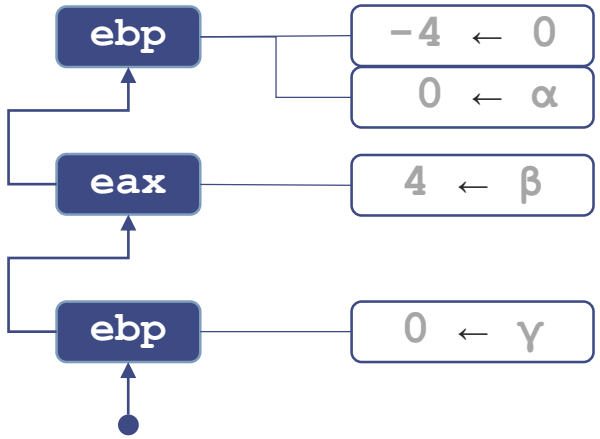
4

LAZY INITIALIZATION

File backed chunk of initial memory
Iterative memory refinement lemma

Revisiting the memory (Farinier et al. 18)





`read ebp+8` → δ

`read ebp-4` → 0

`eax ⊆ [1024,4096]`

`ebp ≥ 16384`

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

2

MEMORY LAYER

Base address with Offset-value map

3

WRITE HISTORY

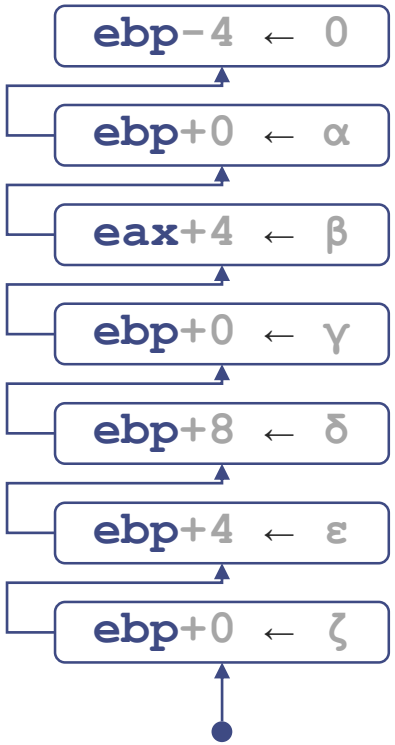
Sequential list of layers of non-comparable base addresses

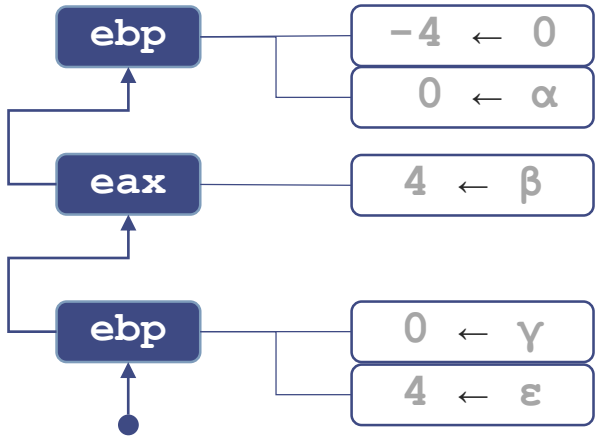
4

LAZY INITIALIZATION

File backed chunk of initial memory
Iterative memory refinement lemma

Revisiting the memory (Farinier et al. 18)





`read ebp+8` → δ

`read ebp-4` → 0

`eax ⊆ [1024,4096]`

`ebp ≥ 16384`

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

2

MEMORY LAYER

Base address with Offset-value map

3

WRITE HISTORY

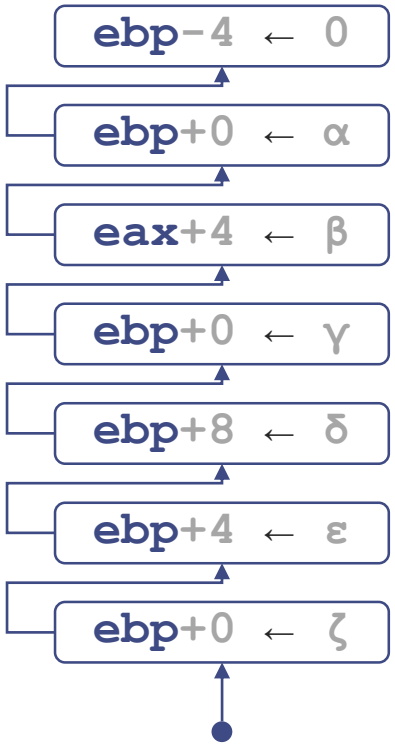
Sequential list of layers of non-comparable base addresses

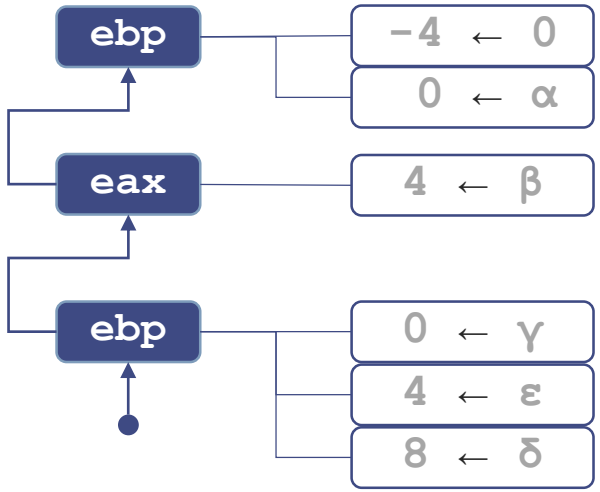
4

LAZY INITIALIZATION

File backed chunk of initial memory
Iterative memory refinement lemma

Revisiting the memory (Farinier et al. 18)





1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

2

MEMORY LAYER

Base address with Offset-value map

3

WRITE HISTORY

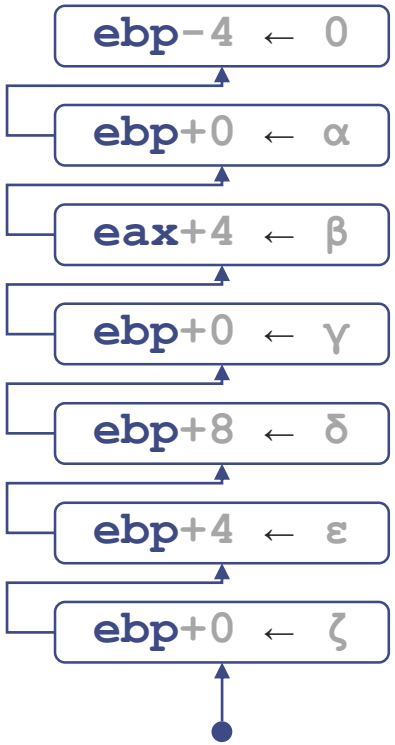
Sequential list of layers of non-comparable base addresses

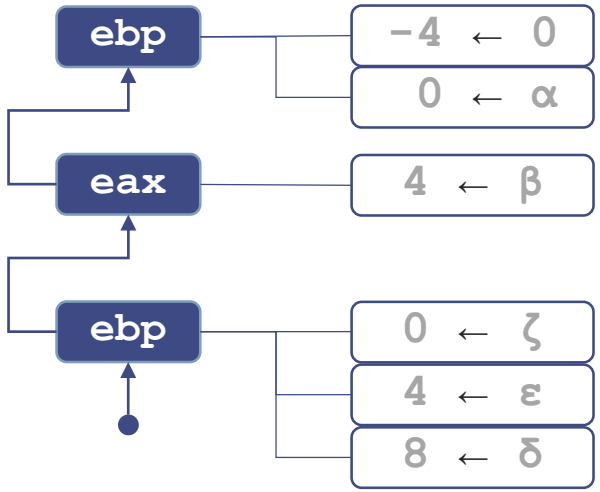
4

LAZY INITIALIZATION

File backed chunk of initial memory
Iterative memory refinement lemma

Revisiting the memory (Farinier et al. 18)





`read ebp+8` → δ

`read ebp-4` → 0

`eax ⊆ [1024,4096]`

`ebp ≥ 16384`

1

MEMORY SIMPLIFICATIONS

Read-over-Write, Write-over-Write
Astract domain disequality resolution

2

MEMORY LAYER

Base address with Offset-value map

3

WRITE HISTORY

Sequential list of layers of non-comparable base addresses

4

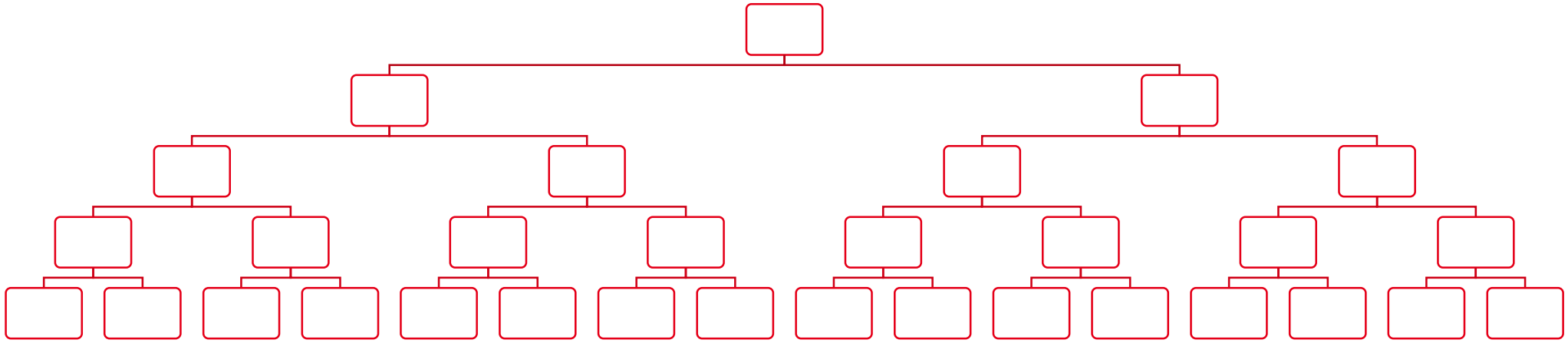
LAZY INITIALIZATION

File backed chunk of initial memory
Iterative memory refinement lemma

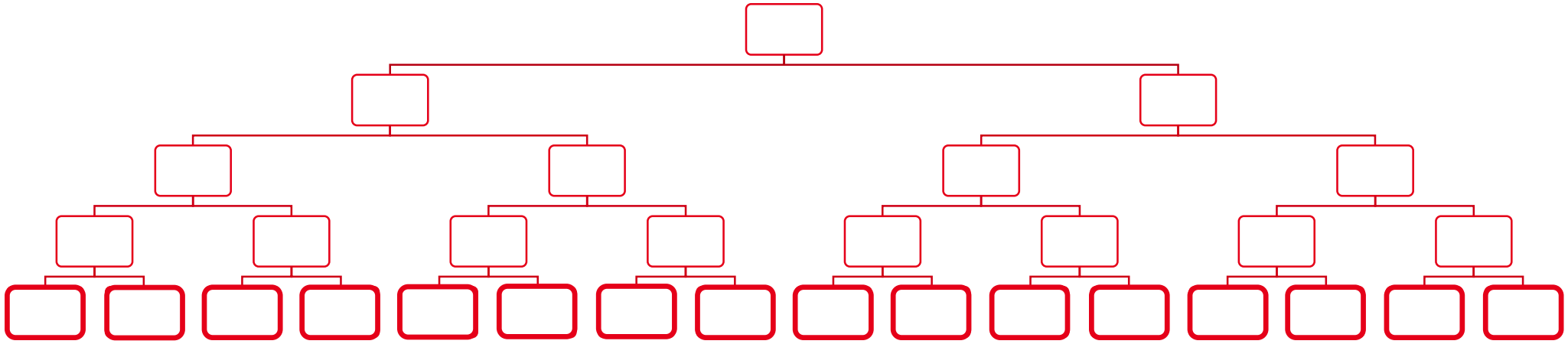
1. The BINSEC designs

- Introduction to BINSEC platform
- Under the Hood
  - Path predicate & Memory model
  - **Efficient use of SMT solvers**
  - JIT specialization of the interpreter
- Plugin extensibility

Avoid calling the SMT solver

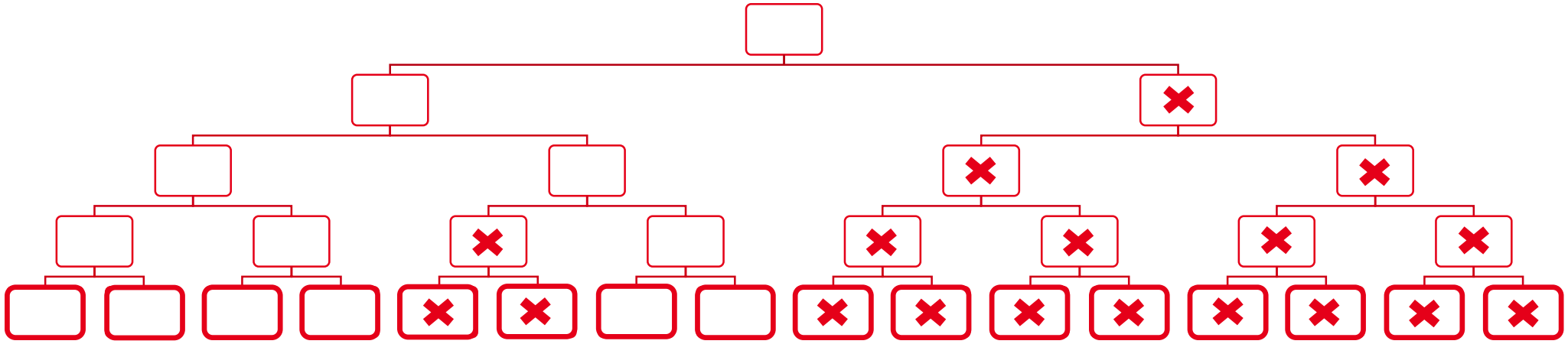


Avoid calling the SMT solver



01 **SMT SOLVER**
Generate a new model

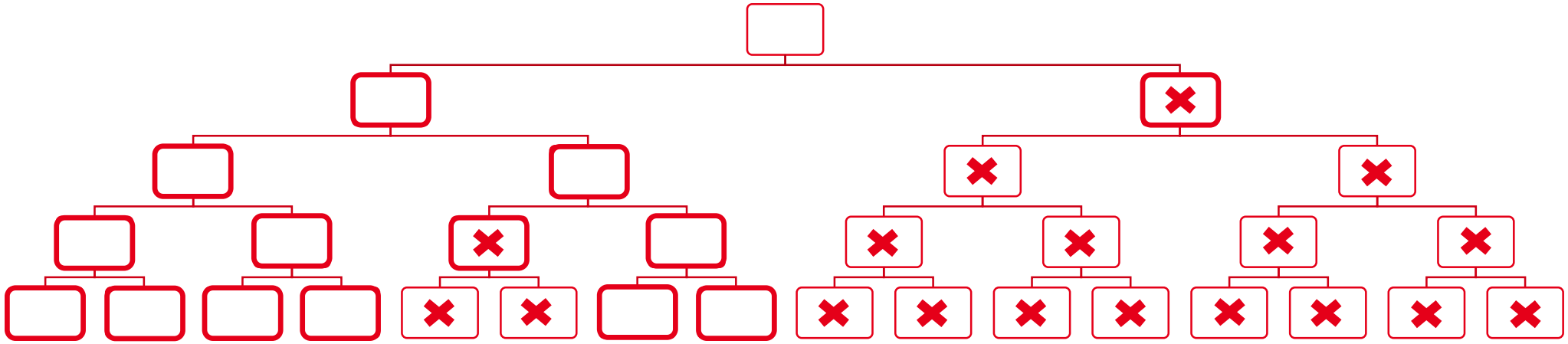
Avoid calling the SMT solver



01

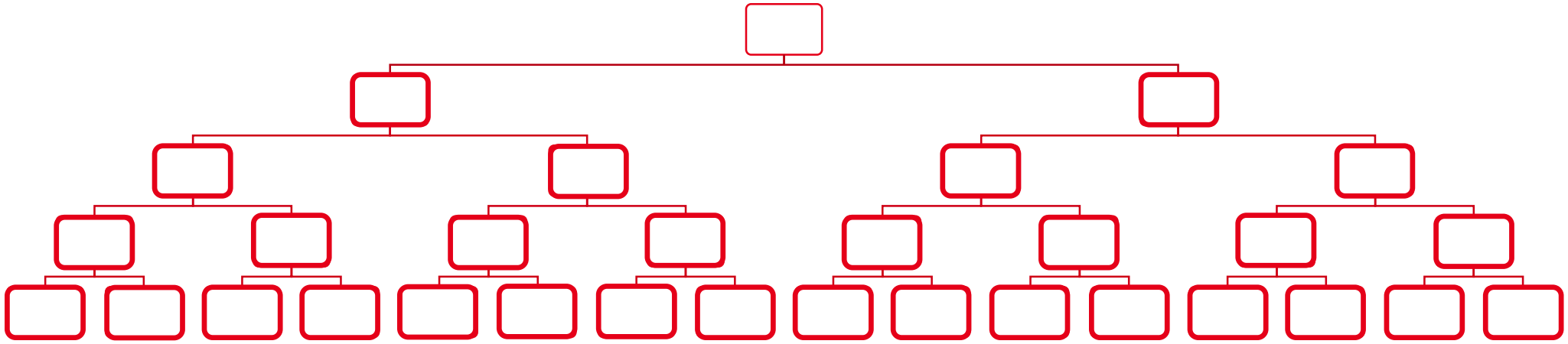
SMT SOLVER
Generate a new model

Avoid calling the SMT solver



01 **SMT SOLVER**
Generate a new model

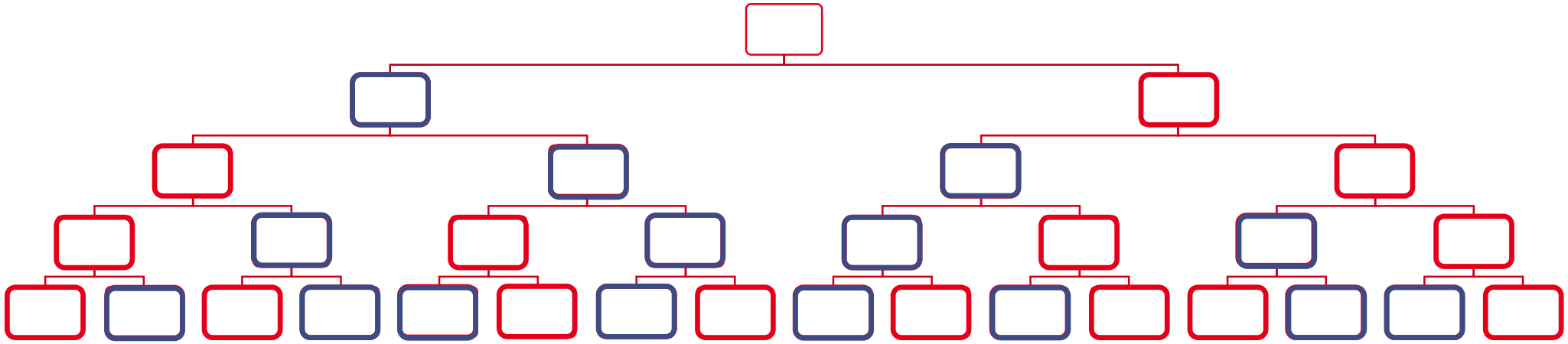
Avoid calling the SMT solver



01

SMT SOLVER
Generate a new model

Avoid calling the SMT solver

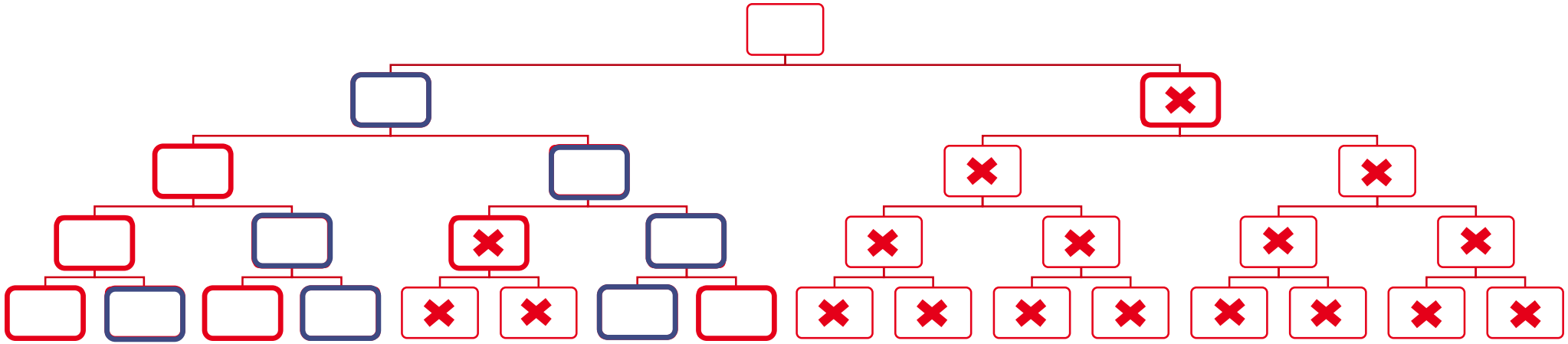


01 **SMT SOLVER**
Generate a new model

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

Kapus et al., Pending Constraints in Symbolic Execution for Better Exploration and Seeding, ASE 2020
Bardin et al., Structural Testing of Executables, ICST 2008
Williams et al., On-the-Fly Generation of K-Path Tests for C Functions, ASE 2004

Avoid calling the SMT solver

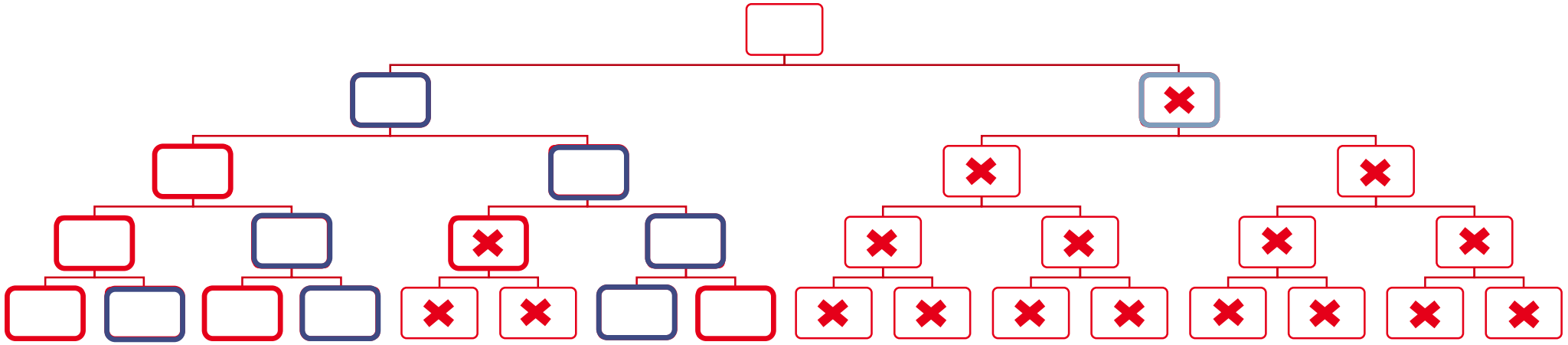


01 **SMT SOLVER**
Generate a new model

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

Kapus et al., Pending Constraints in Symbolic Execution for Better Exploration and Seeding, ASE 2020
Bardin et al., Structural Testing of Executables, ICST 2008
Williams et al., On-the-Fly Generation of K-Path Tests for C Functions, ASE 2004

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

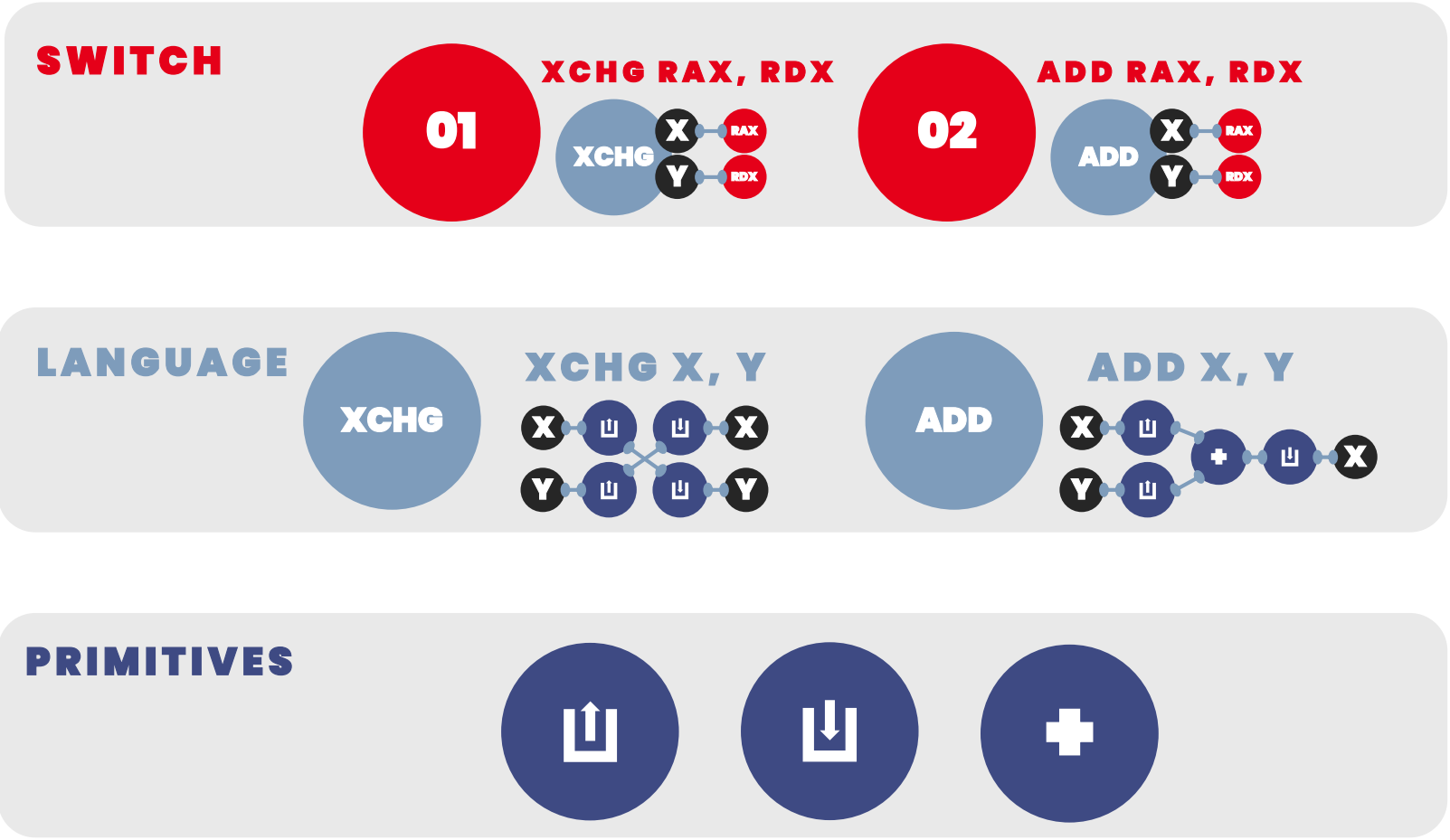
03 **OVER APPROX.**
Along a path
Discard invalid constraints

Kapus et al., Pending Constraints in Symbolic Execution for Better Exploration and Seeding, ASE 2020
Bardin et al., Structural Testing of Executables, ICST 2008
Williams et al., On-the-Fly Generation of K-Path Tests for C Functions, ASE 2004

1. The BINSEC designs

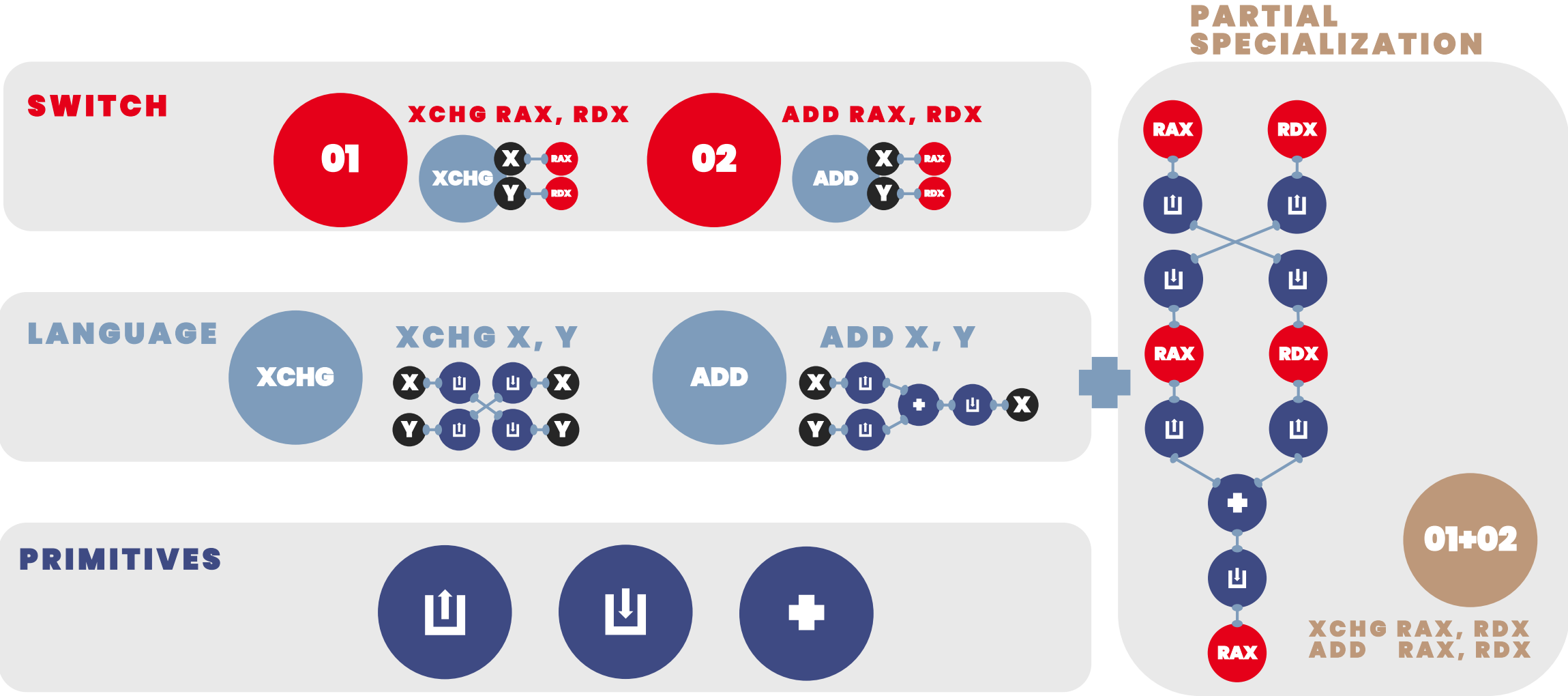
- Introduction to BINSEC platform
- Under the Hood
  - Path predicate & Memory model
  - Efficient use of SMT solvers
  - **JIT specialization of the interpreter**
- Plugin extensibility

Specialize the symbolic interpreter



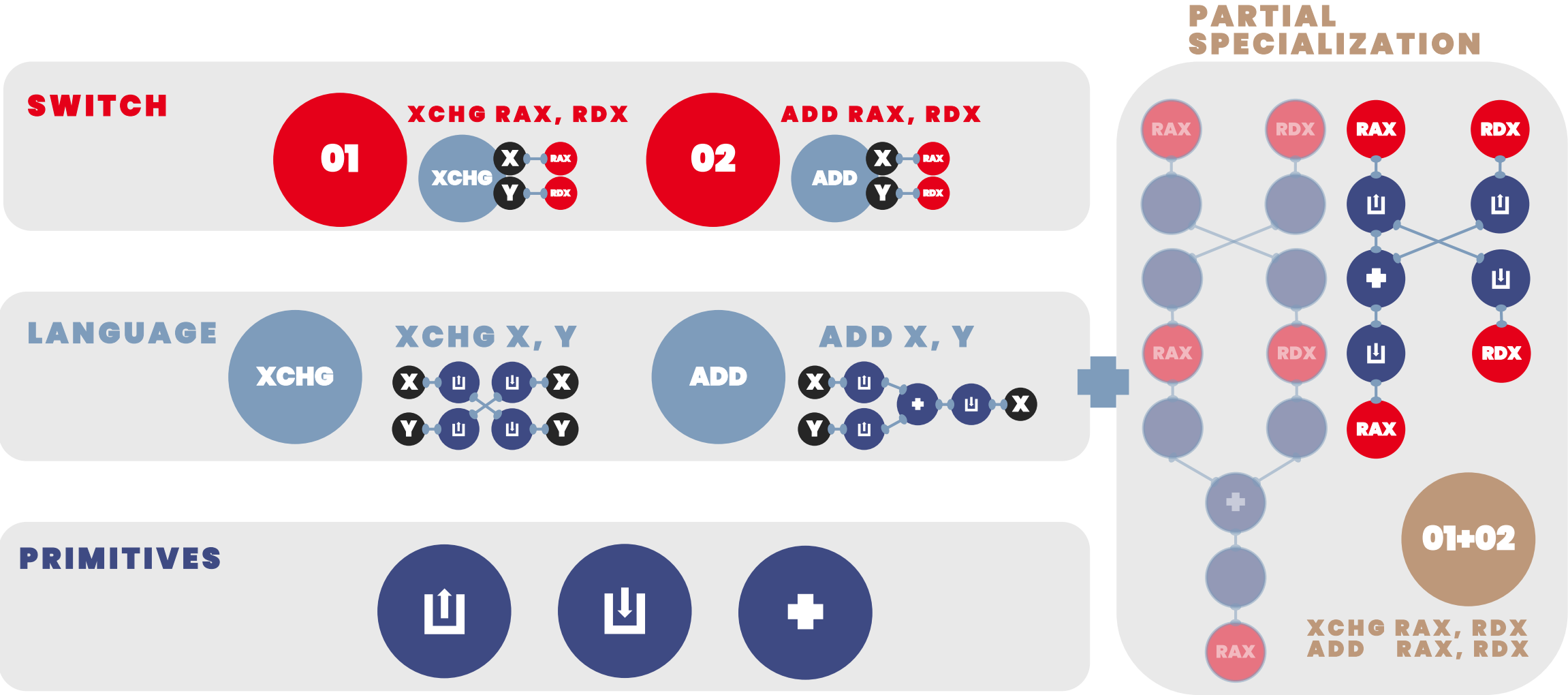
Recoules et al., L'interprète, le JIT et la licorne, JFLA 2024

Specialize the symbolic interpreter



Recoules et al., L'interprète, le JIT et la licorne, JFLA 2024

Specialize the symbolic interpreter



Recoules et al., L'interprète, le JIT et la licorne, JFLA 2024

Challenge 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 !**
(core dump)

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)

Challenge licorne : a difficulty that is not virtual

FRANCE CYBERSECURITY CHALLENGE 2022

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

01 **Shared library**

02 **Self-modifying code**

03 **Billion of instructions**

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

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

UNICORN

multi-architecture CPU emulator framework (based on QEMU)

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

2. The BINSEC news

- Introduction to BINSEC platform
- Under the Hood
  - Path predicate & Memory model
  - Efficient use of SMT solvers
  - JIT specialization of the interpreter
- **Plugin extensibility**

SE instrumentation with plugins

01

SCRIPT SYNTAX

Extensible parser copes for brand new initialization commands, instructions or syntactic sugar

02

BUILTIN

Extension mechanism makes advanced Builtin function written in host language OCaml possible

03

INSTRUMENTATION ROUTINE

Disassembly provides newly discovered code fragment inspection and mark-up procedures

04

EVENT CALLBACK

Hook registration monitors path related event like forking or ending

SE instrumentation with plugins

01

SCRIPT SYNTAX

Extensible parser copes for brand new initialization commands, instructions or syntactic sugar

02

BUILTIN

Extension mechanism makes advanced Builtin function written in host language OCaml possible

03

INSTRUMENTATION ROUTINE

Disassembly provides newly discovered code fragment inspection and mark-up procedures

04

EVENT CALLBACK

Hook registration monitors path related event like forking or ending

checkct

SEMI-RELATIONAL ENGINE REWORK

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

Xmas edition

- **X86_64**
- ARMv7
- **ARMv8**
- **BBSE**

0.4

12/2021

0.5

- **New symbolic engine**
- Write-ups & tutorials
- CTF examples

04/2022

Easter hunt

0.6

- **RISC-V 64**
- x86 AVX extension
- Exploration board

09/2022

Back to school

Twin souls

0.7

- **Constant time**
- Z80
- Custom array

02/2023

Uprising

0.8

- **Plugins**
- PowerPC 64

07/2023

Labor easing

0.9

- **Incremental solver**
- **JIT specialization**

05/2024

JOBS

WE ARE HIRING