



L'interprète, le JIT et la licorne

Frédéric Recoules & Sébastien Bardin



En quelques mots

Contexte Une exécution symbolique au niveau binaire codée en OCaml

Problème L'interprétation des instructions qui devient *LE* goulot d'étranglement de l'analyse *dans certains cas*

Contribution De l'optimisation des blocs de base à l'aide compilation à la volée en OCaml

- Sûr et facile d'utilisation
- Efficace et autonome à l'exécution
- Au prix d'une expressivité restreinte

Le cas du *France CyberSecurity Challenge* 2022

```
4157 488d 05a7 5000 0041 5666 480
4155 660f 6cc0 4154 4c8d 25d1 4f0
4c89 e553 498d 5c24 4048 83ec 5866 0f6f
0d3b 0f00 0066 0fd4 c80f 290d 3050 0000
660f 6f0d 380f 0000 660f d4c8 0f29 0d2d
5000 0066 0f6f 0d35 0f00 0066 0fd4 c866
0fd4 0539 0f00 000f 290d 2250 0000 0f29
052b 5000 000f 1f00 31f6 4889 eabf 0200
0000 e859 ffff ff85 c00f 8590 0100 0048
8b7d 00b9 0700 0000 ba00 0400 0031 f648
0dc5 08e8 b8fe ffff 4839 dd75 cb4c 8d7c
2410 488d 4424 504c 897c 2408 488d 2d6d
4f00 004c 8d2d 4a0e 0000 4889 0424 4989
ee0f 1f80 0000 0000 4c89 fe4c 89ef 31c0
feff ff49 8b47 f849 83c6
3b3c 2475 dd41 bd1f 0000
0000 4989 ef4d 89e6 6690
bec7 0000 0049 83c6 0849
83c7 08e8 68fe ffff 4939 de75 e345 31f6
4c89 f749 83c6 01e8 e401 0000 4983 fe08
75ee 4d89 e741 be07 0000 000f
4489 f049 8b3f bec7 0000 0049
e007 4183 c601 488d 54c5 00e8
4939 df75 db41 83ed 0175 8d66
4e00 0066 0feb 05b5 4e00 0066
```

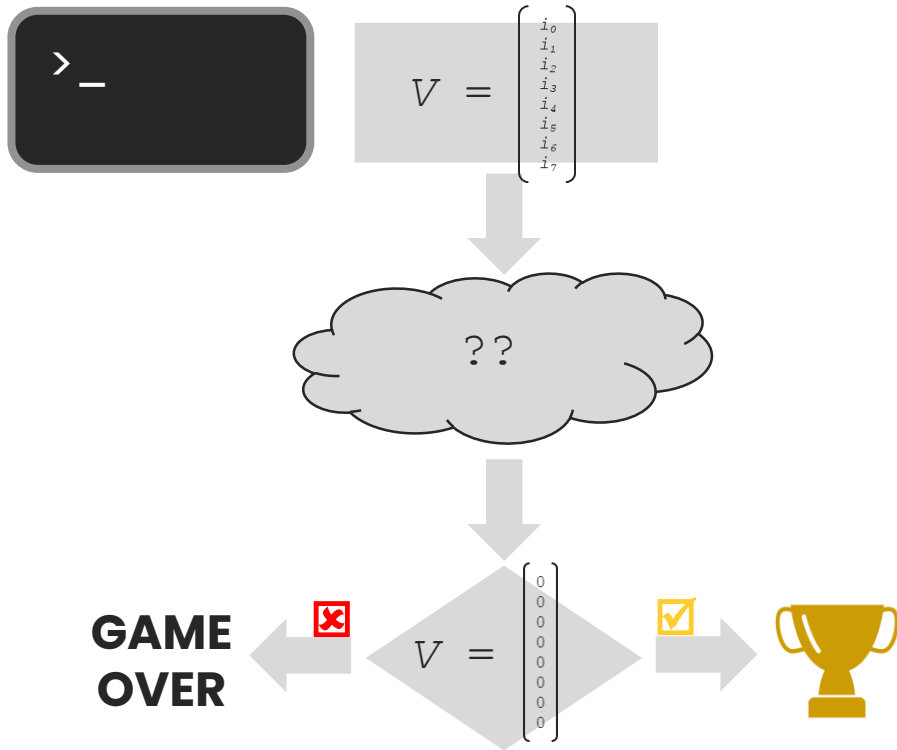
EXE



BINSEC



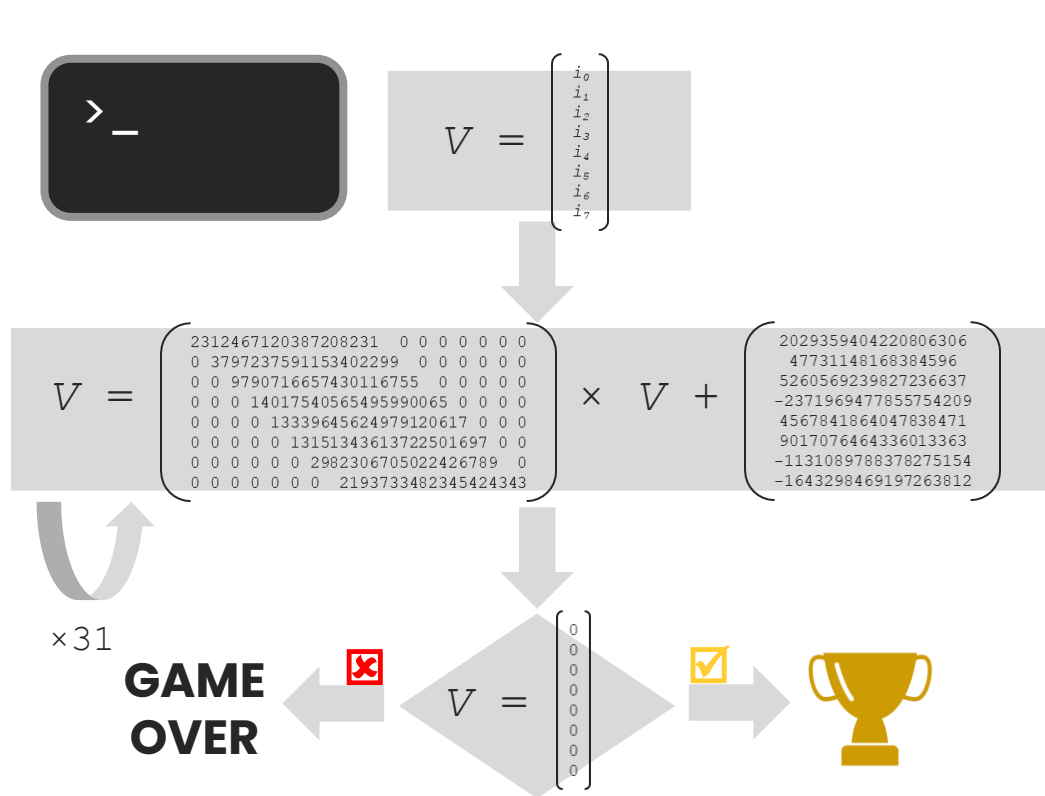
Le cas du *France CyberSecurity Challenge* 2022



```
4157 488d 05a7 5000 0041 5666 480
4155 660f 6cc0 4154 4c8d 25d1 4f0
4c89 e553 498d 5c24 4048 83ec 5866 0f6f
0d3b 0f00 0066 0fd4 c80f 290d 3050 0000
660f 6f0d 380f 0000 660f d4c8 0f29 0d2d
5000 0066 0f6f 0d35 0f00 0066 0fd4 c866
0fd4 0539 0f00 000f 290d 2250 0000 0f29
052b 5000 000f 1f00 31f6 4889 eabf 0200
0000 e859 ffff ff85 c00f 8590 0100 0048
8b7d 00b9 0700 0000 ba00 0400 0031 f648
0dc5 08e8 b8fe ffff 4839 dd75 cb4c 8d7c
2410 488d 4424 504c 897c 2408 488d 2d6d
4f00 004c 8d2d 4a0e 0000 4889 0424 4989
ee0f 1f80 0000 0000 4c89 fe4c 89ef 31c0
feff ff49 8b47 f849 83c6
3b3c 2475 dd41 bd1f 0000
0000 4989 ef4d 89e6 6690
bec7 0000 0049 83c6 0849
83c7 08e8 68fe ffff 4939 de75 e345 31f6
4c89 f749 83c6 01e8 e401 0000 4983 fe08
75ee 4d89 e741 be07 0000 000f
4489 f049 8b3f bec7 0000 0049
e007 4183 c601 488d 54c5 00e8
4939 df75 db41 83ed 0175 8d66
4e00 0066 0feb 05b5 4e00 0066
```

EXE

Le cas du *France CyberSecurity Challenge* 2022



```

4157 488d 05a7 5000 0041 5666 480
4155 660f 6cc0 4154 4c8d 25d1 4f0
4c89 e553 498d 5c24 4048 83ec 5866 0f6f
0d3b 0f00 0066 0fd4 c80f 290d 3050 0000
660f 6f0d 380f 0000 660f d4c8 0f29 0d2d
5000 0066 0f6f 0d35 0f00 0066 0fd4 c866
0fd4 0539 0f00 000f 290d 2250 0000
052b 5000 000f 1f00 31f6 4889 eabf 0200
0000 e859 ffff ff85 c00f 8590 0100 0000
8b7d 00b9 0700 0000 ba00 0400 0031
0dc5 08e8 b8fe ffff 4839 dd75 cb4c
2410 488d 4424 504c 897c 2408 488d
4f00 004c 8d2d 4a0e 0000 4889 0424 4
ee0f 1f80 0000 0000 4c89 fe4c 89ef
feff ff49 8b47 f815 8
3b3c 2475 dd41 bd1f
0000 4989 ef4d 89e6 8
bec7 0000 0049 83c6 0849
83c7 08e8 68fe ffff 4939 de75 e345 31f6
4c89 f749 83c6 01e8 e401 0000 4983 fe08
75ee 4d89 e741 be07 0000 000f
4489 f049 8b3f bec7 0000 0049
e007 4183 c601 488d 54c5 00e8
4939 df75 db41 83ed 0175 8d66
4e00 0066 0feb 05b5 4e00 0066
    
```

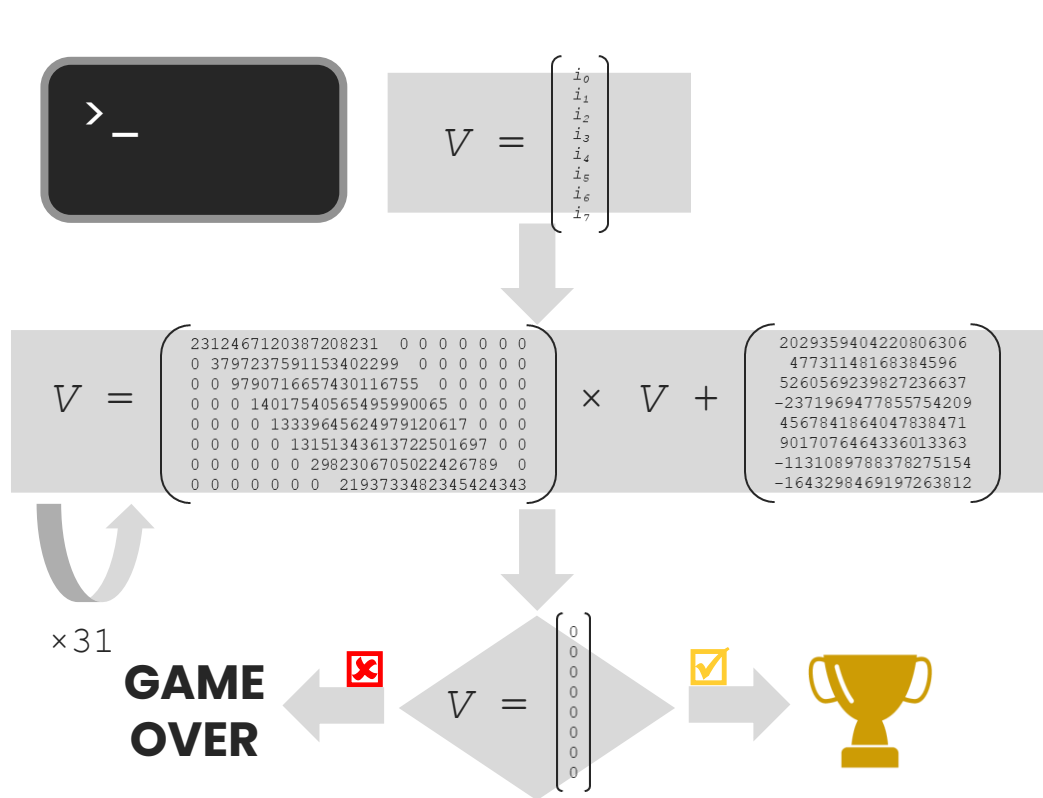
EXE



999 859 832 instructions

 BINSEC

Le cas du *France CyberSecurity Challenge* 2022



EXE

```

4157 488d 05a7 5000 0041 5666 480
4155 660f 6cc0 4154 4c8d 25d1 4f0
4c89 e553 498d 5c24 4048 83ec 5866 0f6f
0d3b 0f00 0066 0fd4 c80f 290d 3050 0000
660f 6f0d 380f 0000 660f d4c8 0f29 0d2d
5000 0066 0f6f 0d35 0f00 0066 0fd4 c866
0fd4 0539 0f00 000f 290d 2250 0000
052b 5000 000f 1f00 31f6 4889 eabf 0200
0000 e859 ffff ff85 c00f 8590 0100 0000
8b7d 00b9 0700 0000 ba00 0400 0031
0dc5 08e8 b8fe ffff 4839 dd75 cb4c
2410 488d 4424 504c 897c 2408 488d
4f00 004c 8d2d 4a0e 0000 4889 0424 4a0e
ee0f 1f80 0000 0000 4c89 fe4c 89ef
feff ff49 8b47 f819 8b47
3b3c 2475 dd41 bd1f
0000 4989 ef4d 89e6 89e6
bec7 0000 0049 83c6 0849
83c7 08e8 68fe ffff 4939 de75 e345 31f6
4c89 f749 83c6 01e8 e401 0000 4983 fe08
75ee 4d89 e741 be07 0000 000f
4489 f049 8b3f bec7 0000 0049
e007 4183 c601 488d 54c5 00e8
4939 df75 db41 83ed 0175 8d66
4e00 0066 0feb 05b5 4e00 0066
    
```

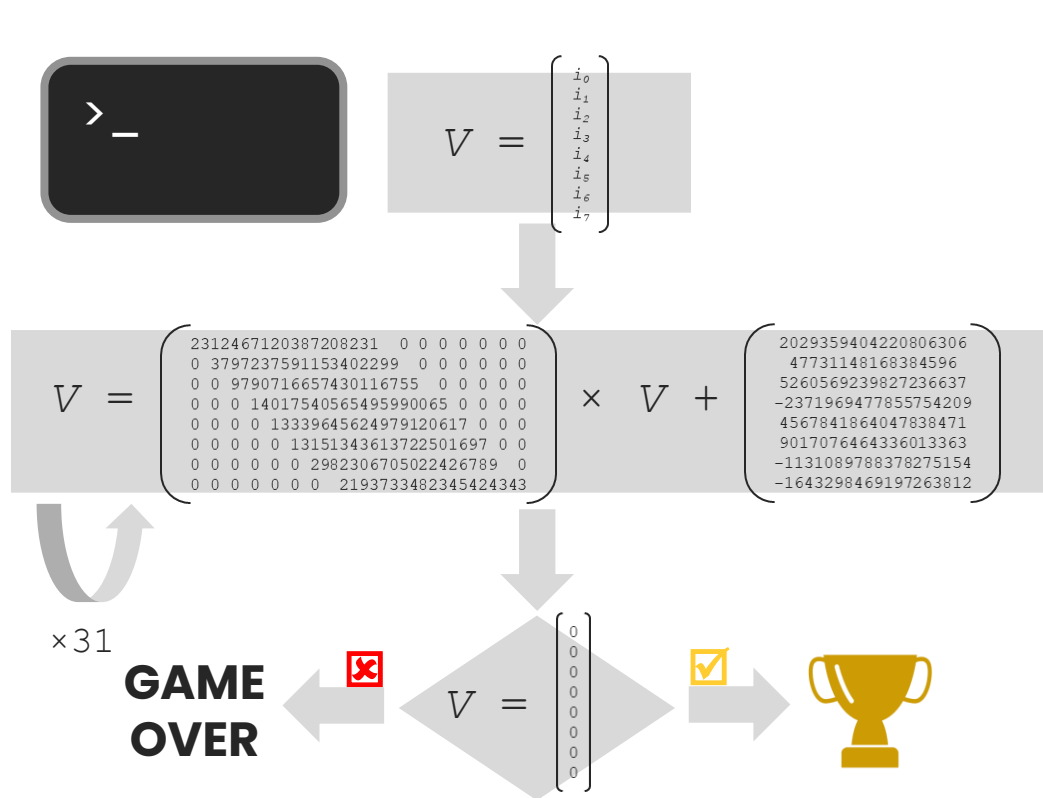
999 859 832 instructions **0.6**

RÉSOLUTION ✓

~3h

BINSEC

Le cas du *France CyberSecurity Challenge* 2022



```

4157 488d 05a7 5000 0041 5666 480
4155 660f 6cc0 4154 4c8d 25d1 4f0
4c89 e553 498d 5c24 4048 83ec 5866 0f6f
0d3b 0f00 0066 0fd4 c80f 290d 3050 0000
660f 6f0d 380f 0000 660f d4c8 0f29 0d2d
5000 0066 0f6f 0d35 0f00 0066 0fd4 c866
0fd4 0539 0f00 000f 290d 2250 0000
052b 5000 000f 1f00 31f6 4889 eabf 0200
0000 e859 ffff ff85 c00f 8590 0100 0000
8b7d 00b9 0700 0000 ba00 0400 0031
0dc5 08e8 b8fe ffff 4839 dd75 cb4c
2410 488d 4424 504c 897c 2408 488d
4f00 004c 8d2d 4a0e 0000 4889 0424 4
ee0f 1f80 0000 0000 4c89 fe4c 89ef
feff ff49 8b47 f815 8
3b3c 2475 dd41 bd1f
0000 4989 ef4d 89e6 8
bec7 0000 0049 83c6 0849
83c7 08e8 68fe ffff 4939 de75 e345 31f6
4c89 f749 83c6 01e8 e401 0000 4983 fe08
75ee 4d89 e741 be07 0000 000f
4489 f049 8b3f bec7 0000 0049
e007 4183 c601 488d 54c5 00e8
4939 df75 db41 83ed 0175 8d66
4e00 0066 0feb 05b5 4e00 0066
    
```

EXE

AMD 64

arm

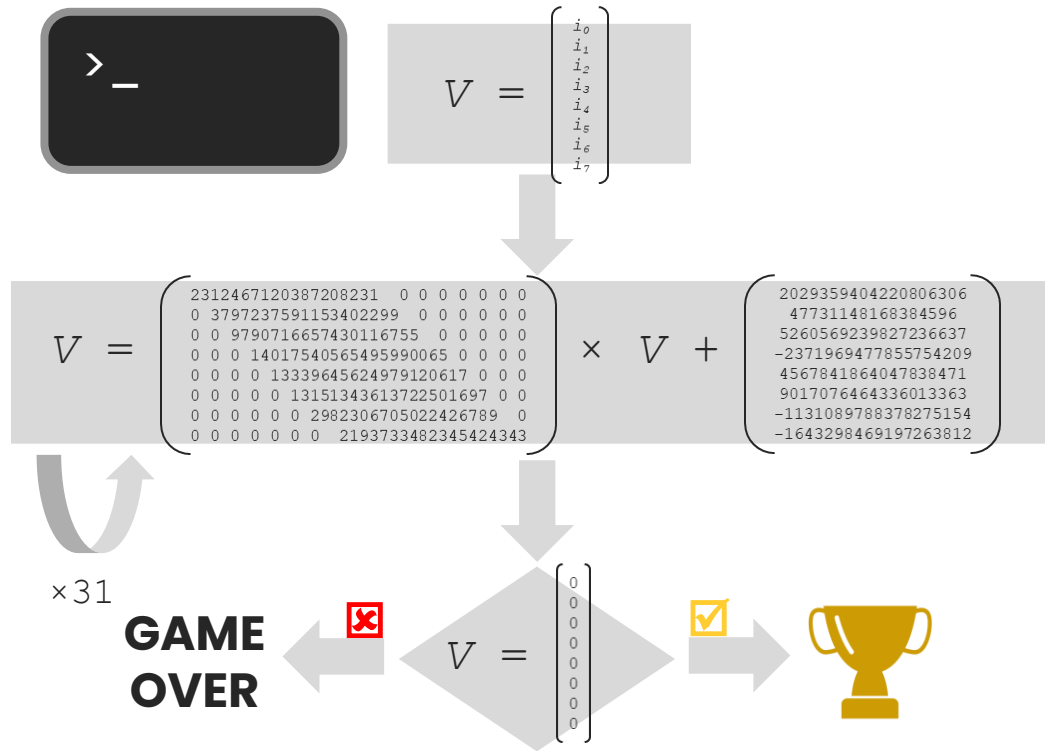
999 859 832 instructions **0.6** **0.8**

BINSEC

RÉSOLUTION ✓	RÉSOLUTION ✓
~3h	10m24



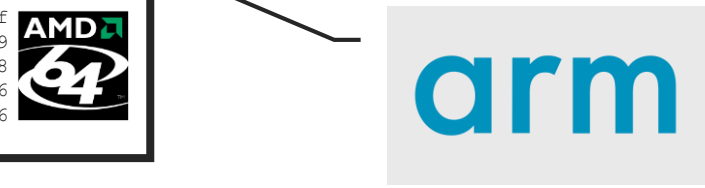
Le cas du *France CyberSecurity Challenge 2022*



```

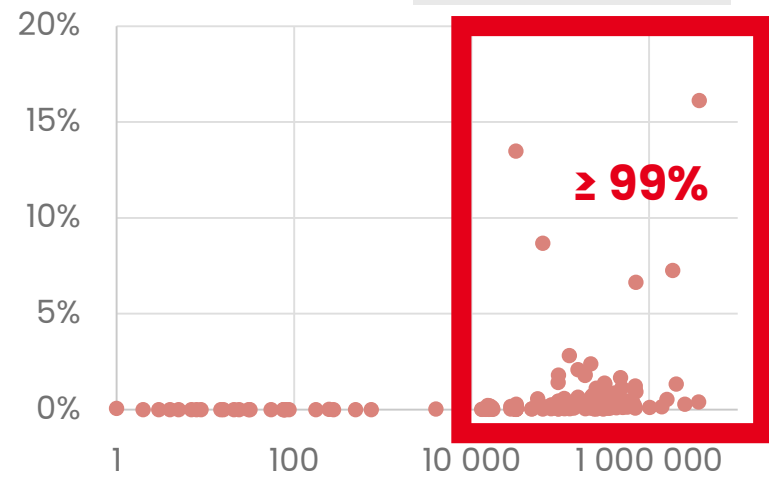
4157 488d 05a7 5000 0041 5666 480
4155 660f 6cc0 4154 4c8d 25d1 4f0
4c89 e553 498d 5c24 4048 83ec 5866 0f6f
0d3b 0f00 0066 0fd4 c80f 290d 3050 0000
660f 6f0d 380f 0000 660f d4c8 0f29 0d2d
5000 0066 0f6f 0d35 0f00 0066 0fd4 c866
0fd4 0539 0f00 000f 290d 2250 0000
052b 5000 000f 1f00 31f6 4889 eabf 0200
0000 e859 ffff ff85 c00f 8590 0100 0000
8b7d 00b9 0700 0000 ba00 0400 0031
0dc5 08e8 b8fe ffff 4839 dd75 cb4c
2410 488d 4424 504c 897c 2408 488d
4f00 004c 8d2d 4a0e 0000 4889 0424 4
ee0f 1f80 0000 0000 4c89 fe4c 89ef
feff ff49 8b47 f815 8
3b3c 2475 dd41 bd1f
0000 4989 ef4d 89e6 8
bec7 0000 0049 83c6 0849
83c7 08e8 68fe ffff 4939 de75 e345 31f6
4c89 f749 83c6 01e8 e401 0000 4983 fe08
75ee 4d89 e741 be07 0000 000f
4489 f049 8b3f bec7 0000 0049
e007 4183 c601 488d 54c5 00e8
4939 df75 db41 83ed 0175 8d66
4e00 0066 0feb 05b5 4e00 0066
    
```

EXE

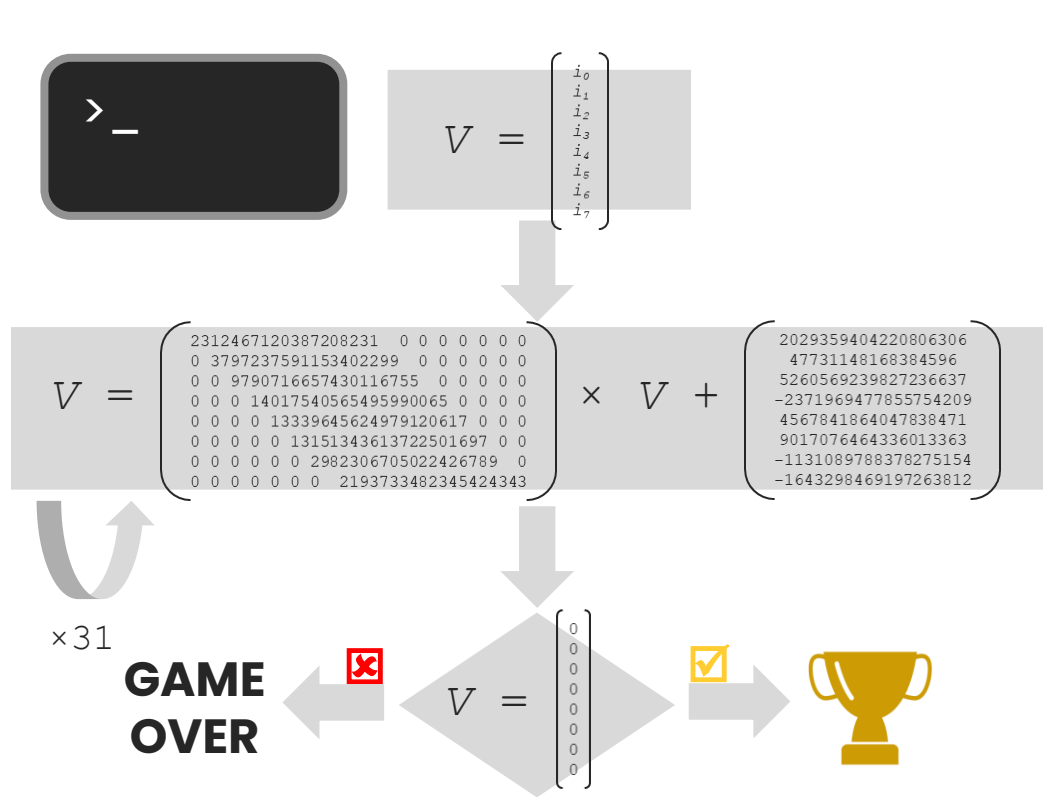


999 859 832 instructions **0.6** **0.8**
RÉSOLUTION ✓ **RÉSOLUTION ✓**
 ~3h 10m24

BINSEC



Le cas du *France CyberSecurity Challenge* 2022



```

4157 488d 05a7 5000 0041 5666 480
4155 660f 6cc0 4154 4c8d 25d1 4f0
4c89 e553 498d 5c24 4048 83ec 5866 0f6f
0d3b 0f00 0066 0fd4 c80f 290d 3050 0000
660f 6f0d 380f 0000 660f d4c8 0f29 0d2d
5000 0066 0f6f 0d35 0f00 0066 0fd4 c866
0fd4 0539 0f00 000f 290d 2250 0000
052b 5000 000f 1f00 31f6 4889 eabf 0200
0000 e859 ffff ff85 c00f 8590 0100 0000
8b7d 00b9 0700 0000 ba00 0400 0031
0dc5 08e8 b8fe ffff 4839 dd75 cb4c
2410 488d 4424 504c 897c 2408 488d
4f00 004c 8d2d 4a0e 0000 4889 0424 4
ee0f 1f80 0000 0000 4c89 fe4c 89ef
feff ff49 8b47 f815 8
3b3c 2475 dd41 bd1f
0000 4989 ef4d 89e6 8
bec7 0000 0049 83c6 0849
83c7 08e8 68fe ffff 4939 de75 e345 31f6
4c89 f749 83c6 01e8 e401 0000 4983 fe08
75ee 4d89 e741 be07 0000 000f
4489 f049 8b3f bec7 0000 0049
e007 4183 c601 488d 54c5 00e8
4939 df75 db41 83ed 0175 8d66
4e00 0066 0feb 05b5 4e00 0066
    
```

EXE




999 859 832 instructions

0.6

0.8

0.8+JIT

 **BINSEC**

RÉSOLUTION ✓

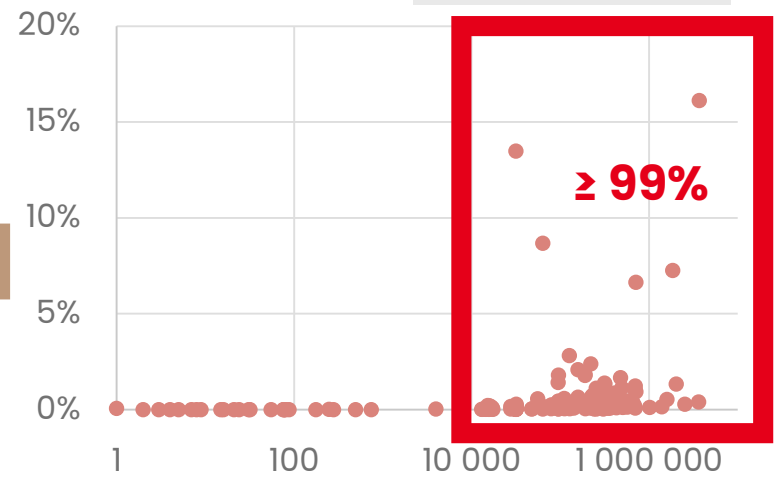
RÉSOLUTION ✓

RÉSOLUTION ✓

~3h 

10m24 

7m15 

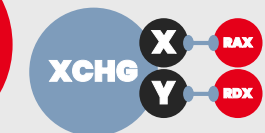


Spécialisation d'un interpréteur

RÉPARTITEUR

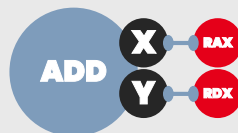
01

XCHG RAX, RDX



02

ADD RAX, RDX



LANGAGE

XCHG

XCHG X, Y



ADD

ADD X, Y



PRIMITIVES



Spécialisation d'un interpréteur

SPÉCIALISATION PARTIELLE

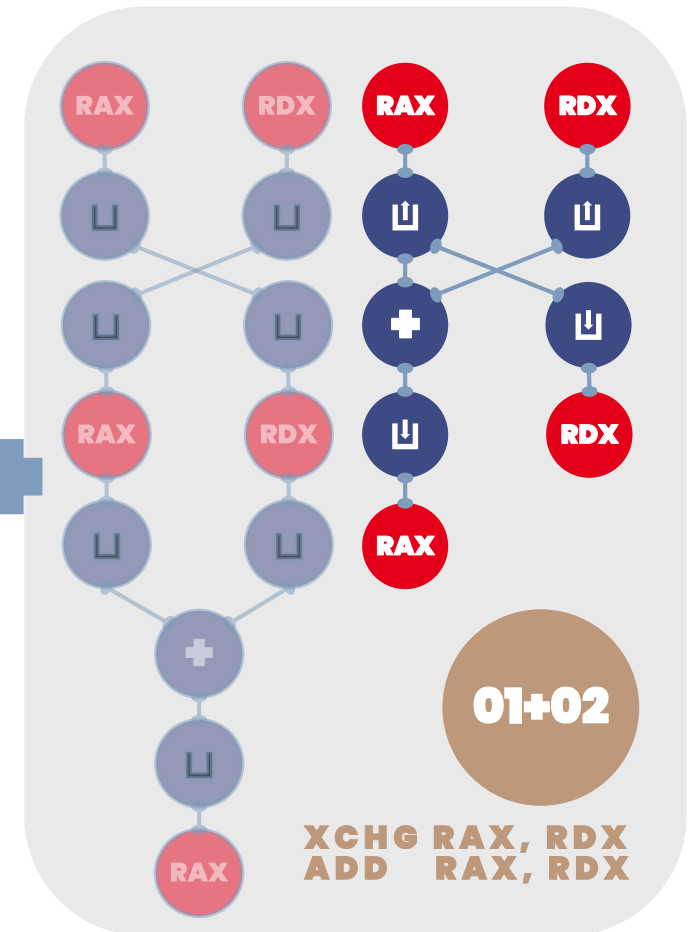
RÉPARTITEUR



LANGAGE



PRIMITIVES

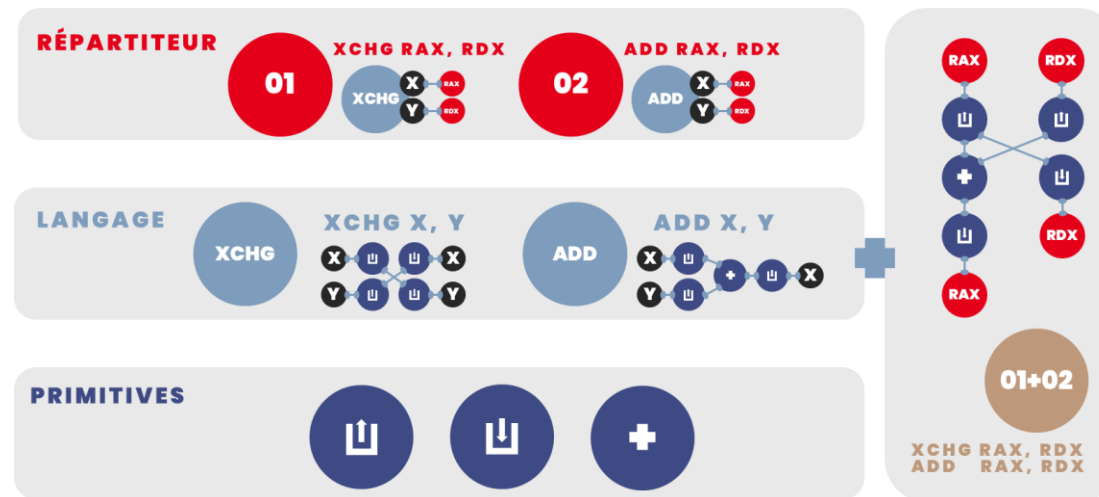


Spécialisation d'un interpréteur en



```
let xchg x y registers =  
  let vx = read_reg x registers in  
  let vy = read_reg y registers in  
  let registers = write_reg x vy registers in  
  let registers = write_reg y vx registers in  
  registers
```

```
let add x y registers =  
  let vx = read_reg x registers in  
  let vy = read_reg y registers in  
  let vp = make_plus vx vy in  
  let registers = write_reg x vp registers in  
  registers
```

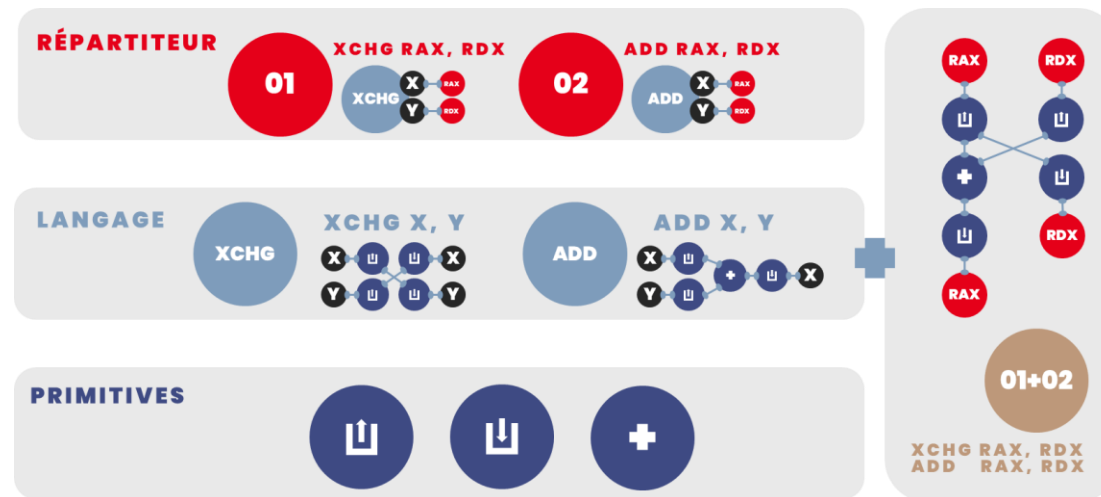


Spécialisation d'un interpréteur en



```
let xchg x y registers =  
  let vx = read_reg x registers in  
  let vy = read_reg y registers in  
  let registers = write_reg x vy registers in  
  let registers = write_reg y vx registers in  
  registers
```

```
let add x y registers =  
  let vx = read_reg x registers in  
  let vy = read_reg y registers in  
  let vp = make_plus vx vy in  
  let registers = write_reg x vp registers in  
  registers
```



```
let xchg_rax_rdx_add_rax_rdx registers =  
  let va = read_reg Rax registers in  
  let vd = read_reg Rdx registers in  
  let vp = make_plus va vd in  
  let registers = write_reg Rdx va registers in  
  let registers = write_reg Rax vp registers in  
  registers
```





“**Compiler *harmonieusement*,
directement depuis OCaml, une
séquence d’appels de fonctions**”

Votre serviteur




Solutions existantes ou ressemblantes



	 LLVM	MetaOCaml	OCaml-JIT
Compatibilité 	x	✓	✓
Autonomie à l'exécutions	N/A	x	x
Temps d'émission de code	N/A		
Multi-architectures	N/A	✓	x

Contributions de l'article



- **Conception** de la bibliothèque **JITPSI**
 - Compilation à la volée d'une séquence d'appel de fonctions
 - Simplicité, efficacité et compatibilité
- **Intégration** dans la plateforme  BINSEC
 - Accélération de la résolution du défis **licorne** (-30%)

- Code source disponible sur  **GitHub** 

JITPSI, une interface sobre mais effective



type value and lambda
type (+'a, 'b) t

val const : 'a -> ('a, value) t

val apply : ('a -> 'b, value) t -> ('a, value) t -> ('b, value) t

val lambda : (('a, value) t -> ('b, 'c) t) -> ('a -> 'b, lambda) t

val return : ('a, lambda) t -> 'a

JITPSI, une interface sobre mais effective



(int, value) t, (int -> string, value) t, etc.
Expression à compiler

type value and lambda
type (+'a, 'b) t

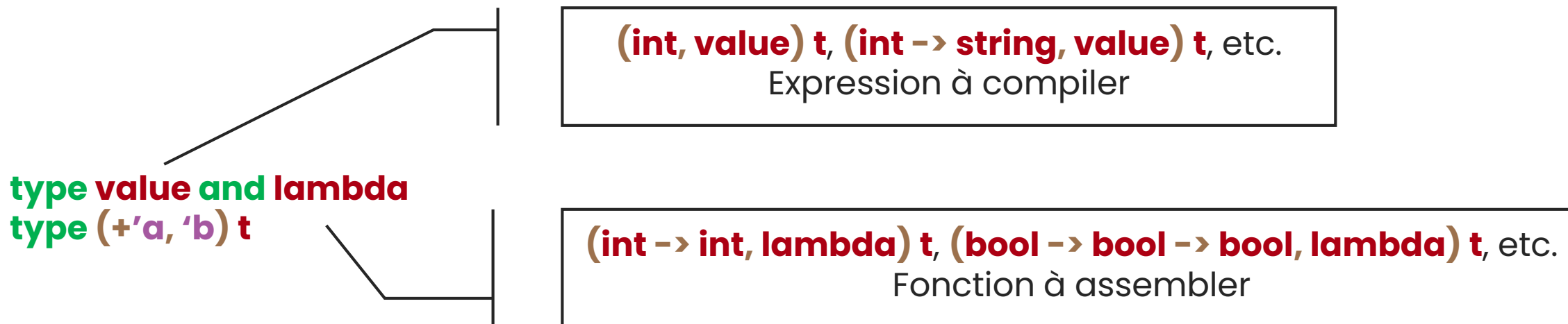
val const : 'a -> ('a, value) t

val apply : ('a -> 'b, value) t -> ('a, value) t -> ('b, value) t

val lambda : (('a, value) t -> ('b, 'c) t) -> ('a -> 'b, lambda) t

val return : ('a, lambda) t -> 'a

JITPSI, une interface sobre mais effective



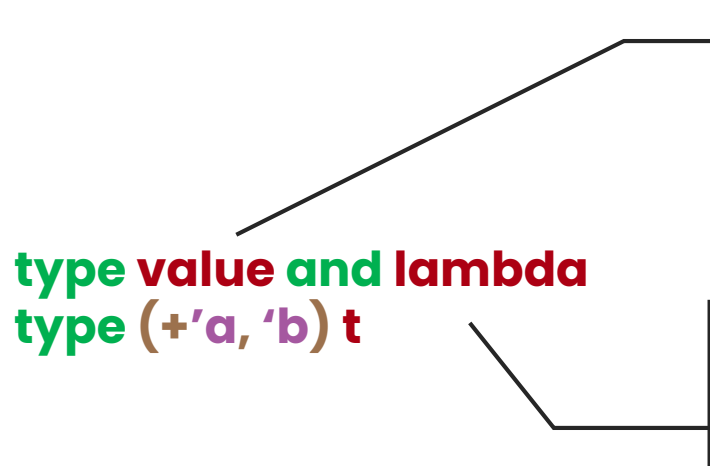
val const : 'a -> ('a, value) t

val apply : ('a -> 'b, value) t -> ('a, value) t -> ('b, value) t

val lambda : (('a, value) t -> ('b, 'c) t) -> ('a -> 'b, lambda) t

val return : ('a, lambda) t -> 'a

JITPSI, une interface sobre mais effective



(int, value) t, (int -> string, value) t, etc.
Expression à compiler

(int -> int, lambda) t, (bool -> bool -> bool, lambda) t, etc.
Fonction à assembler

val const : 'a -> ('a, value) t

val apply : ('a -> 'b, value) t -> ('a, value) t -> ('b, value) t **let apply2** f x y = **apply** (**apply** f x) y

val lambda : (('a, value) t -> ('b, 'c) t) -> ('a -> 'b, lambda) t **let lambda2** body =
lambda (**fun** x -> **lambda** (**body** x))

val return : ('a, lambda) t -> 'a

Pour l'exemple

```
let one : unit -> int = return  
  (lambda (fun _ -> const 1))
```

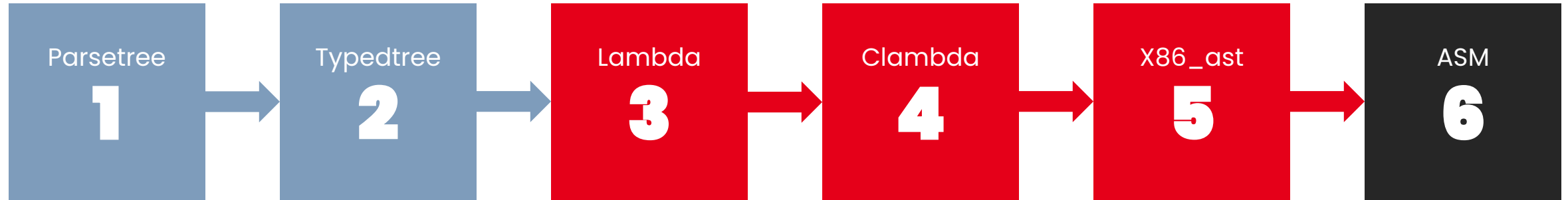
```
let identity : 'a -> 'a = return  
  (lambda (fun x -> x))
```

```
let curry (f : 'a -> 'b -> 'c) (x : 'a) : 'b -> 'c = return  
  (lambda (fun y -> apply2 (const f) (const x) y))
```

```
let flip (f : 'a -> 'b -> 'c) : 'b -> 'a -> 'c = return  
  (lambda2 (fun x y -> apply2 (const f) y x))
```

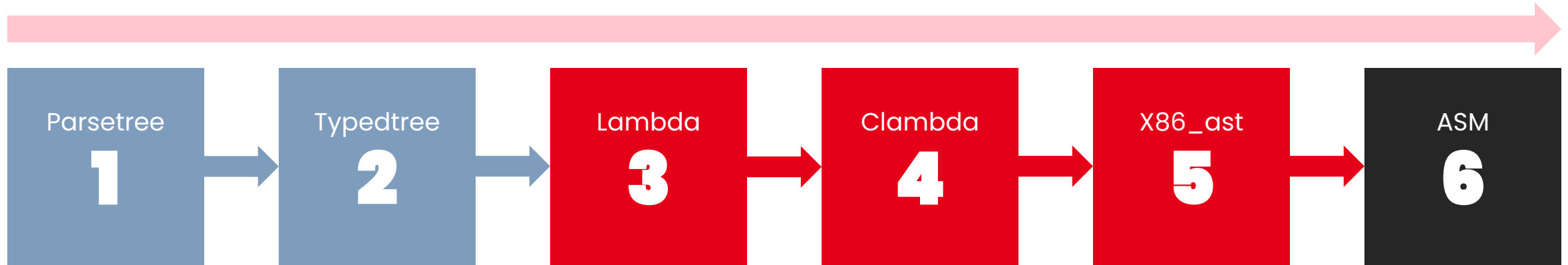


Positionnement de JITPSI



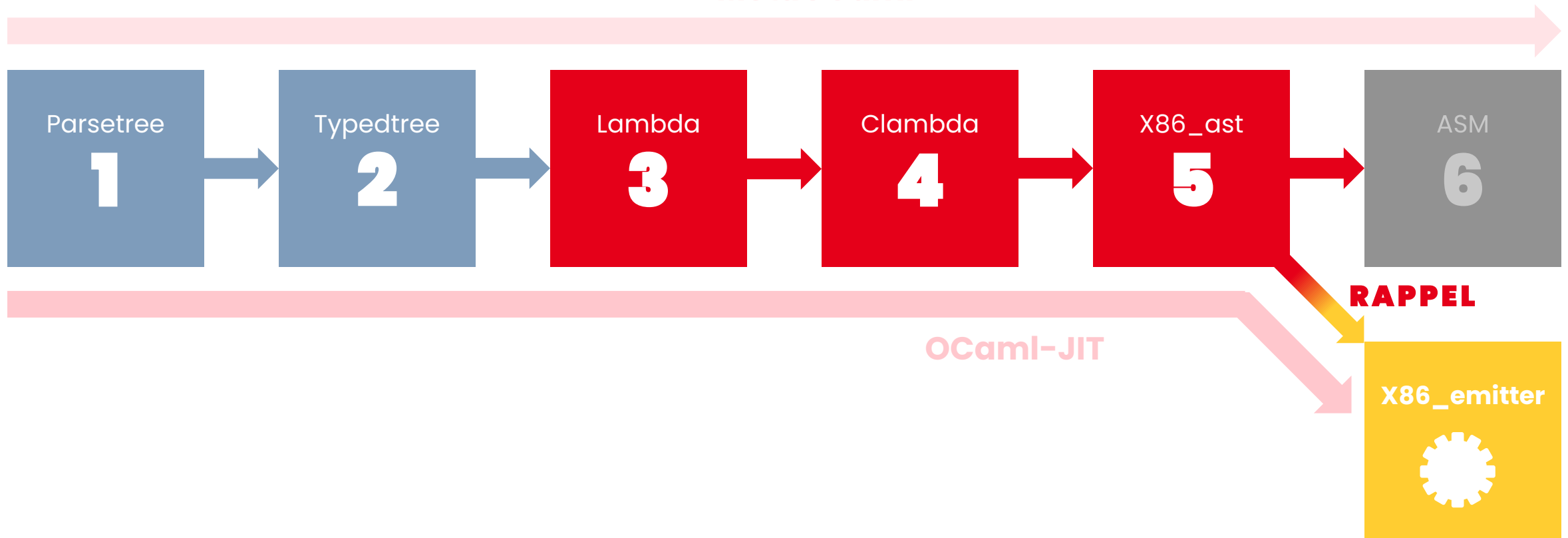
Positionnement de JITPSI

MetaOCaml



Positionnement de JITPSI

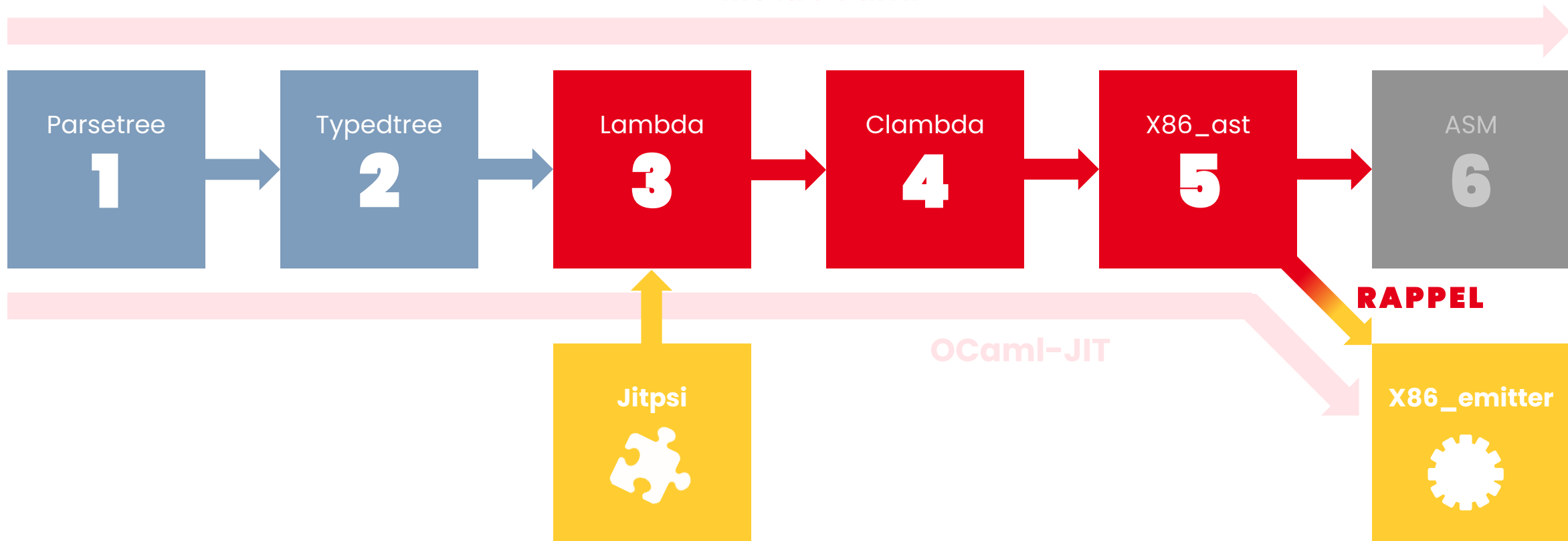
MetaOCaml



Positionnement de JITPSI

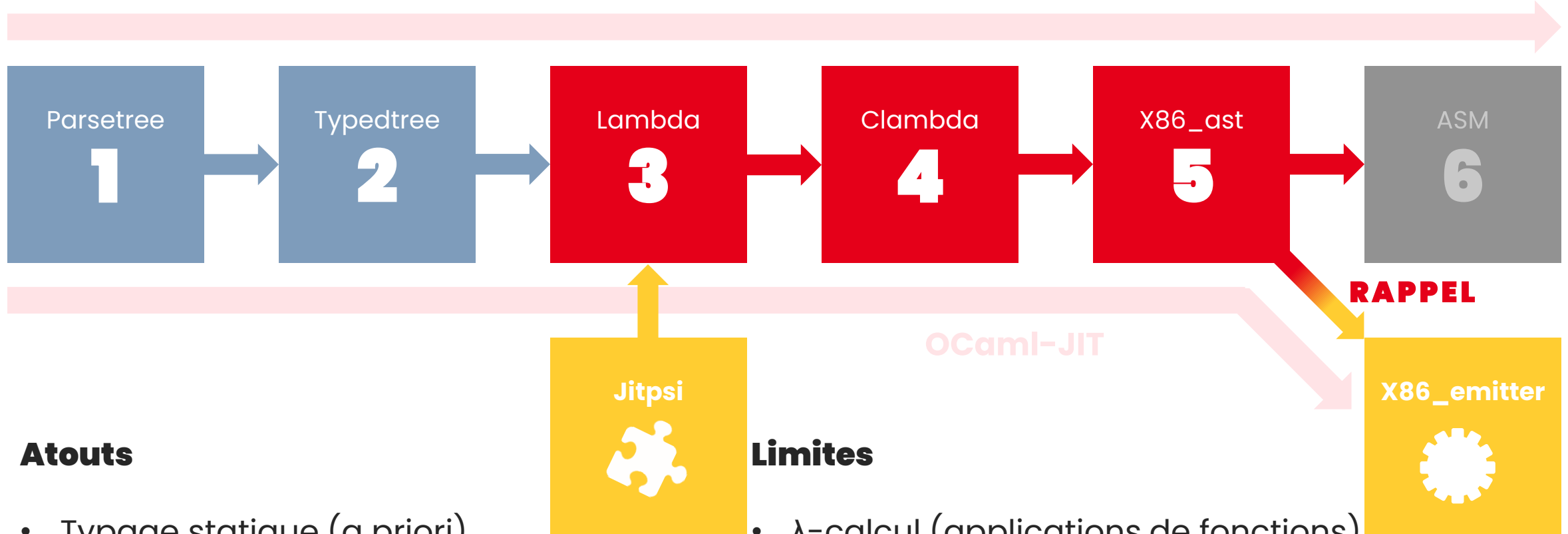


MetaOCaml



Positionnement de JITPSI

MetaOCaml



Atouts






- Typage statique (a priori)
- Indépendance à l'exécution
- Émission de code en mémoire
- *Collectable* par le GC

Limites

- λ -calcul (applications de fonctions)
 - Absence de structures de contrôle
 - Absence de primitives de bases

Récapitulatif final



	 LLVM	MetaOCaml	OCaml-JIT	JITPSI
Compatibilité	x	✓	✓	✓
Autonomie à l'exécutions	N/A	x	x	✓
Expressivité	N/A	✓	✓	x
Temps d'émission de code	N/A			
Multi-architectures	N/A	✓	x	x

Merci pour votre attention



Frédéric Recoules




Sébastien Bardin

- **Conception** de la bibliothèque **JITPSI**
 - Compilation à la volée d'une séquence d'appel de fonctions
 - Simplicité, efficacité et compatibilité 

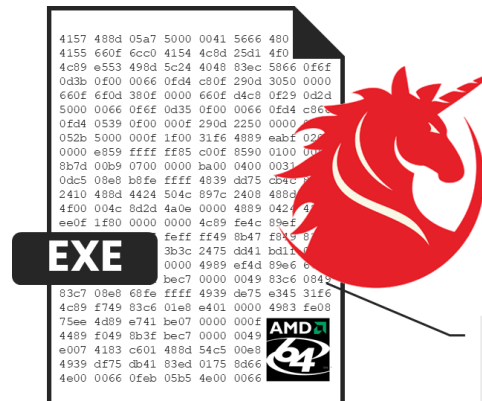
- **Optimisation** d'interpréteur « *au sens large* »

(séquence d'instructions répétées)

- Émulateurs, Machines virtuelles
- Exécution symbolique ( BINSEC)
- Interprétation abstraite ?
- Autres ?

- Code source disponible sur  **GitHub**

France CyberSecurity Challenge 2022



La main est à l'audience

Qui est qui, JITPSI ou code OCaml ?

```
sub    rsp, 18h
mov    [rsp+8], rbx
mov    rdi, [rbx+16]
mov    rbx, rax
mov    [rsp], rbx
mov    eax, 5
call   camlStdlib__Map__find_440
call   camlStdint__pred_4052
mov    rbx, rax
mov    rax, [rsp+8]
mov    rsi, [rax+32]
mov    eax, 5
mov    rdi, [rsp]
add    rsp, 18h
jmp    camlStdlib__Map__add_428
```

0:

```
sub    rsp, 18h
mov    rbx, rax
mov    [rsp], rbx
cmp    r15, [r14]
jbe    1f
lea    rax, camlLibexec
mov    rax, [rax]
mov    rdi, [rax+18h]
mov    [rsp+8], rdi
mov    rdi, [rax+d8h]
mov    eax, 5
call   caml_apply2
call   camlStdint__pred_4052
mov    rbx, rax
mov    eax, 5
mov    rdi, [rsp]
mov    rsi, [rsp+8]
add    rsp, 18h
jmp    caml_apply3
call   caml_call_gc
jmp    0b
```

1: