BinCAT

Purrfecting binary static analysis

June 16th 2017 - REcon

Philippe Biondi, Raphaël Rigo, Sarah Zennou, Xavier Mehrenberger

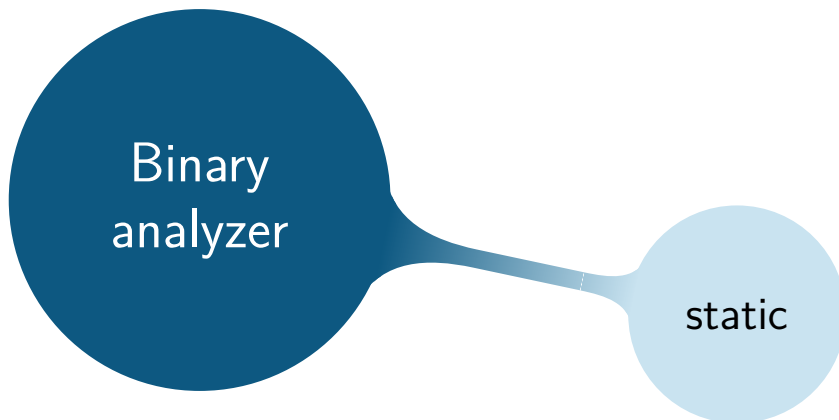
AIRBUS

Plan

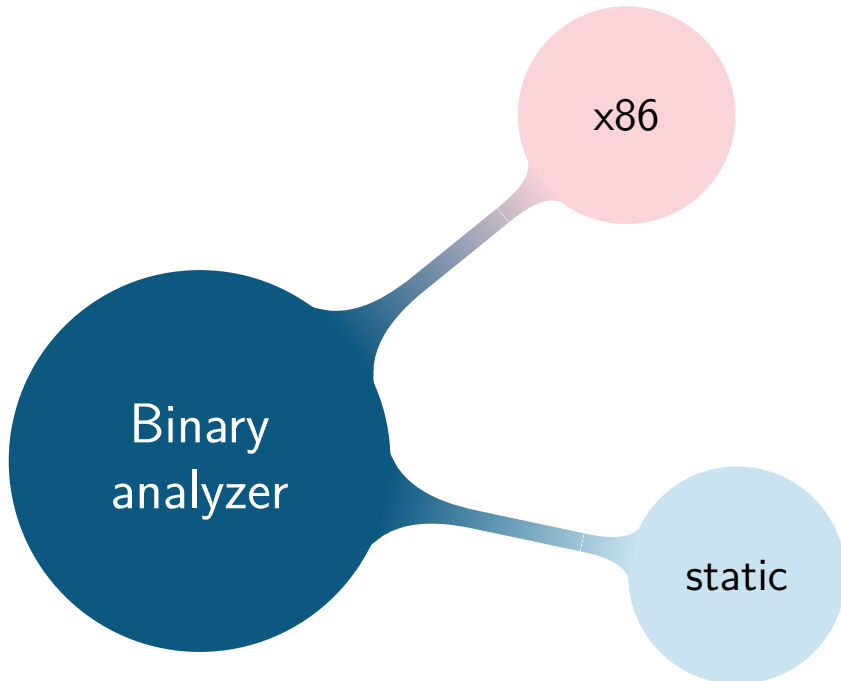
Plan

BinCAT (*Binary Code Analysis Toolkit*)

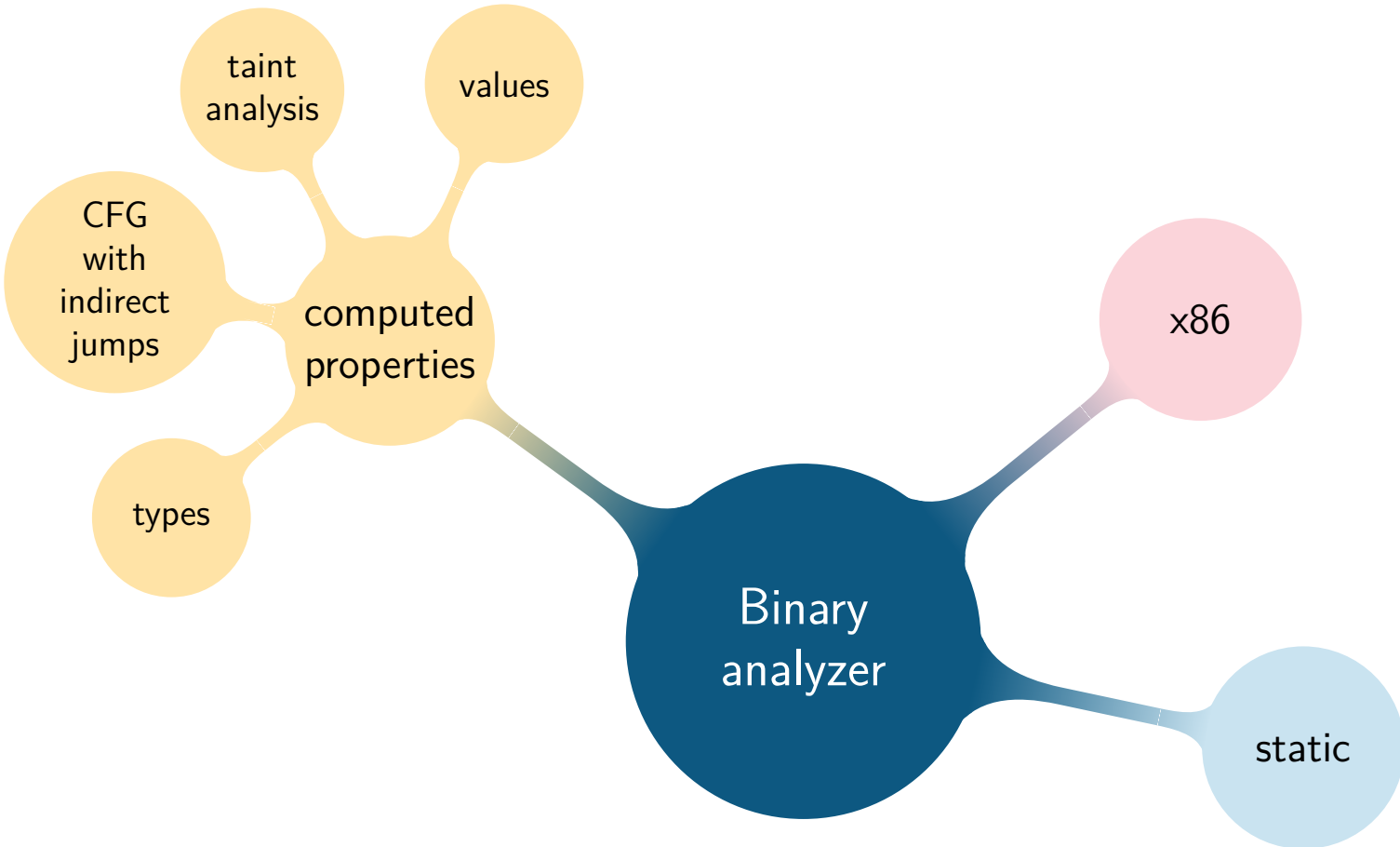
BinCAT (*Binary Code Analysis Toolkit*)



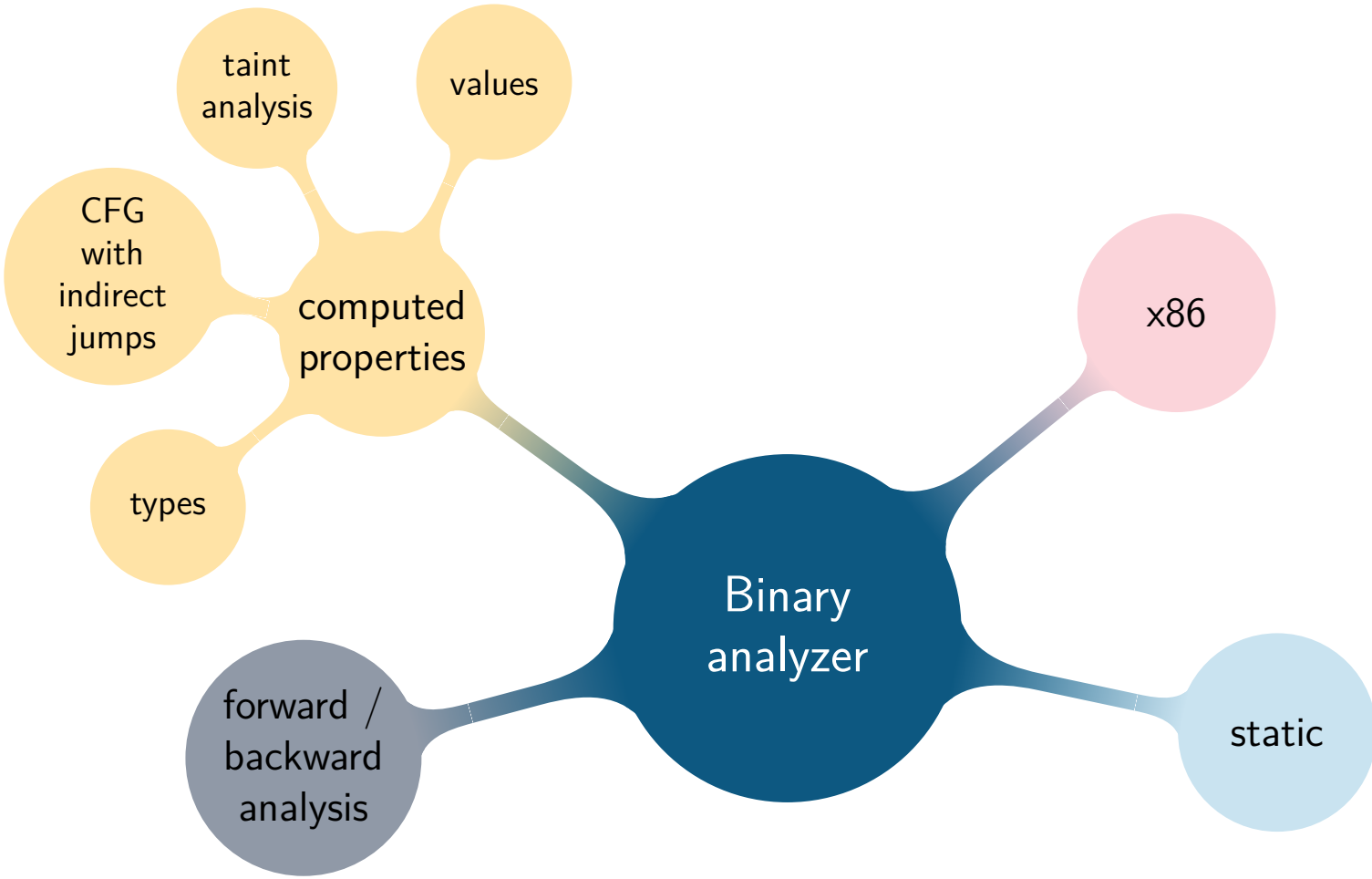
BinCAT (*Binary Code Analysis Toolkit*)



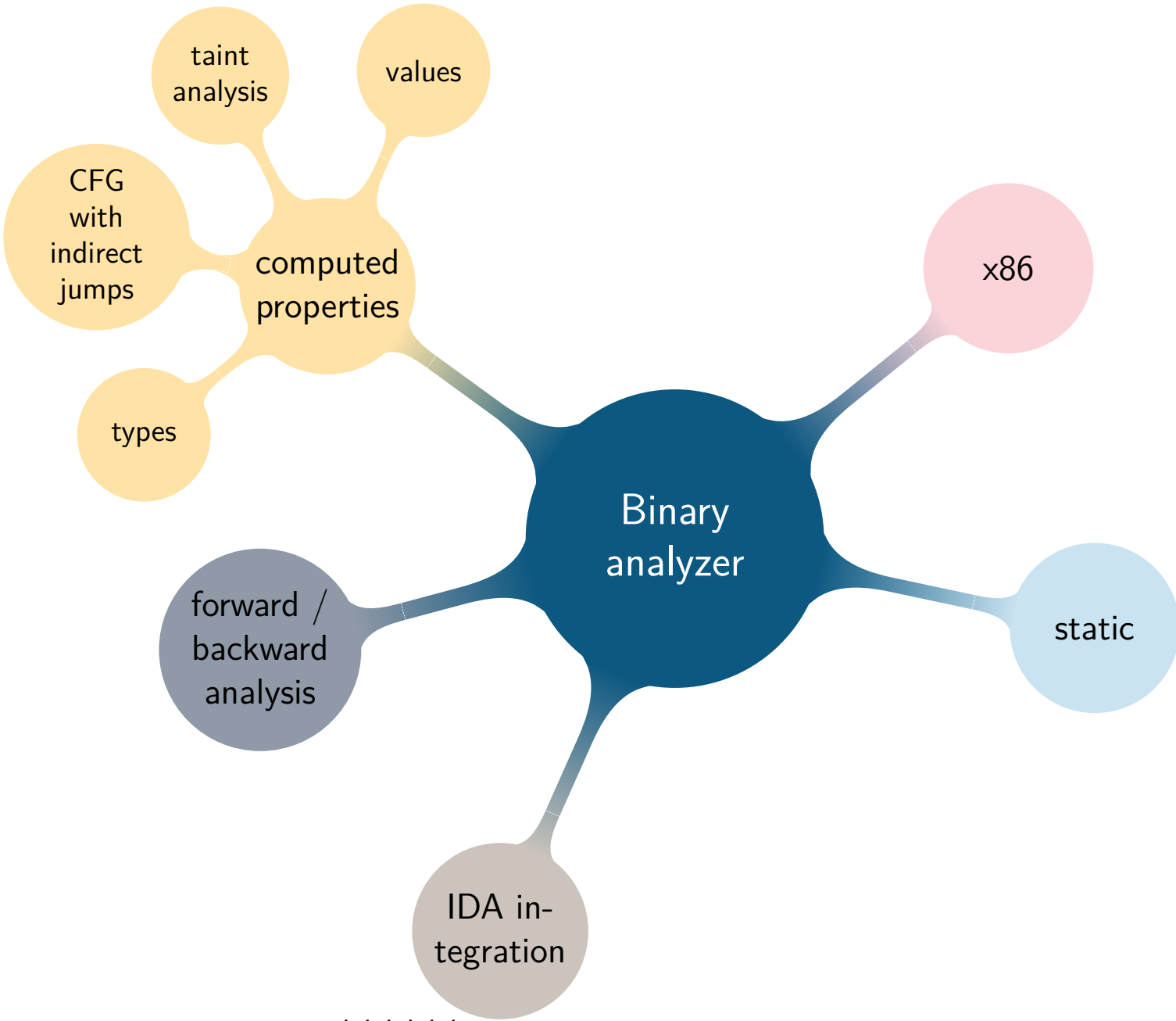
BinCAT (*Binary Code Analysis Toolkit*)



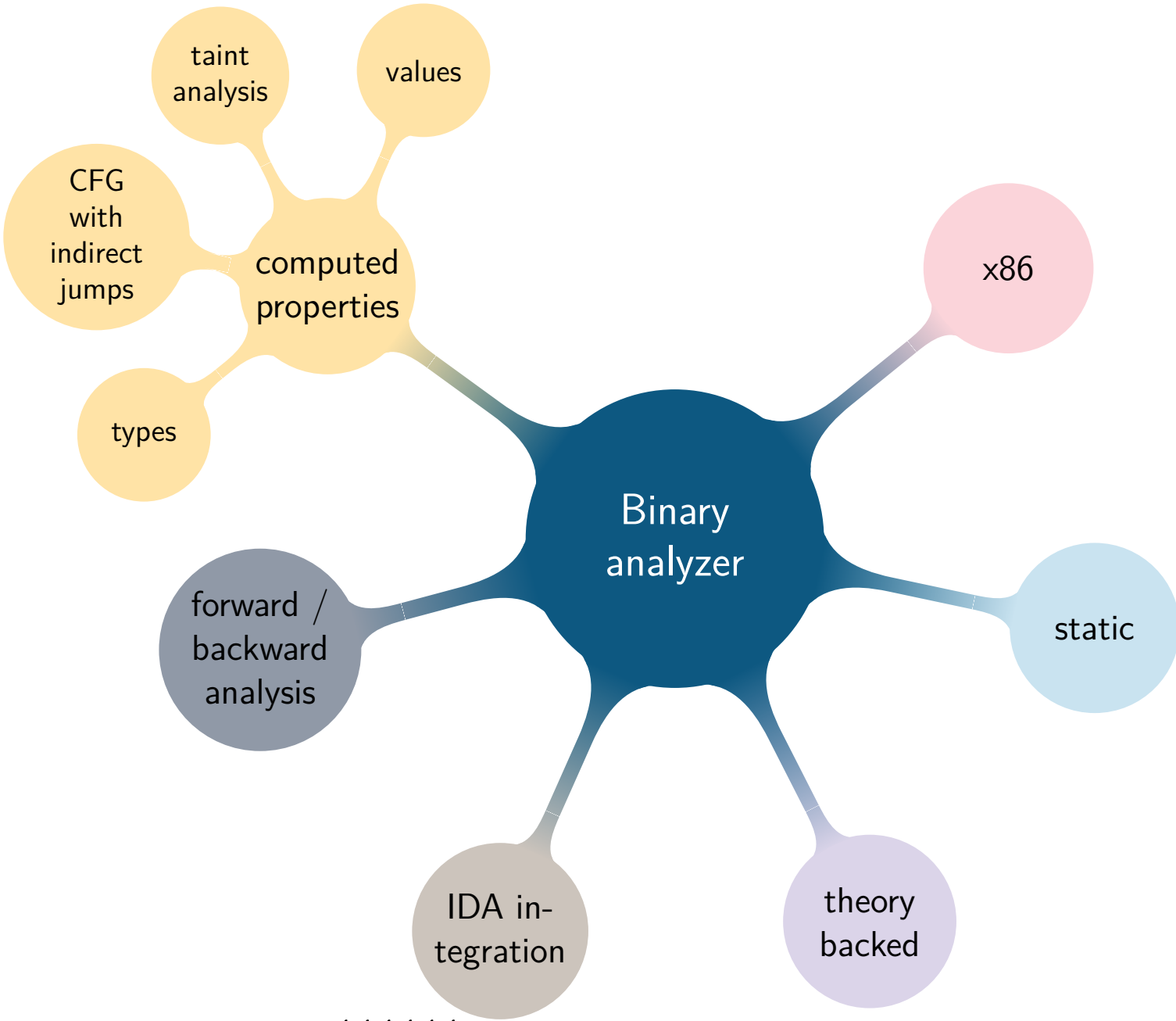
BinCAT (*Binary Code Analysis Toolkit*)



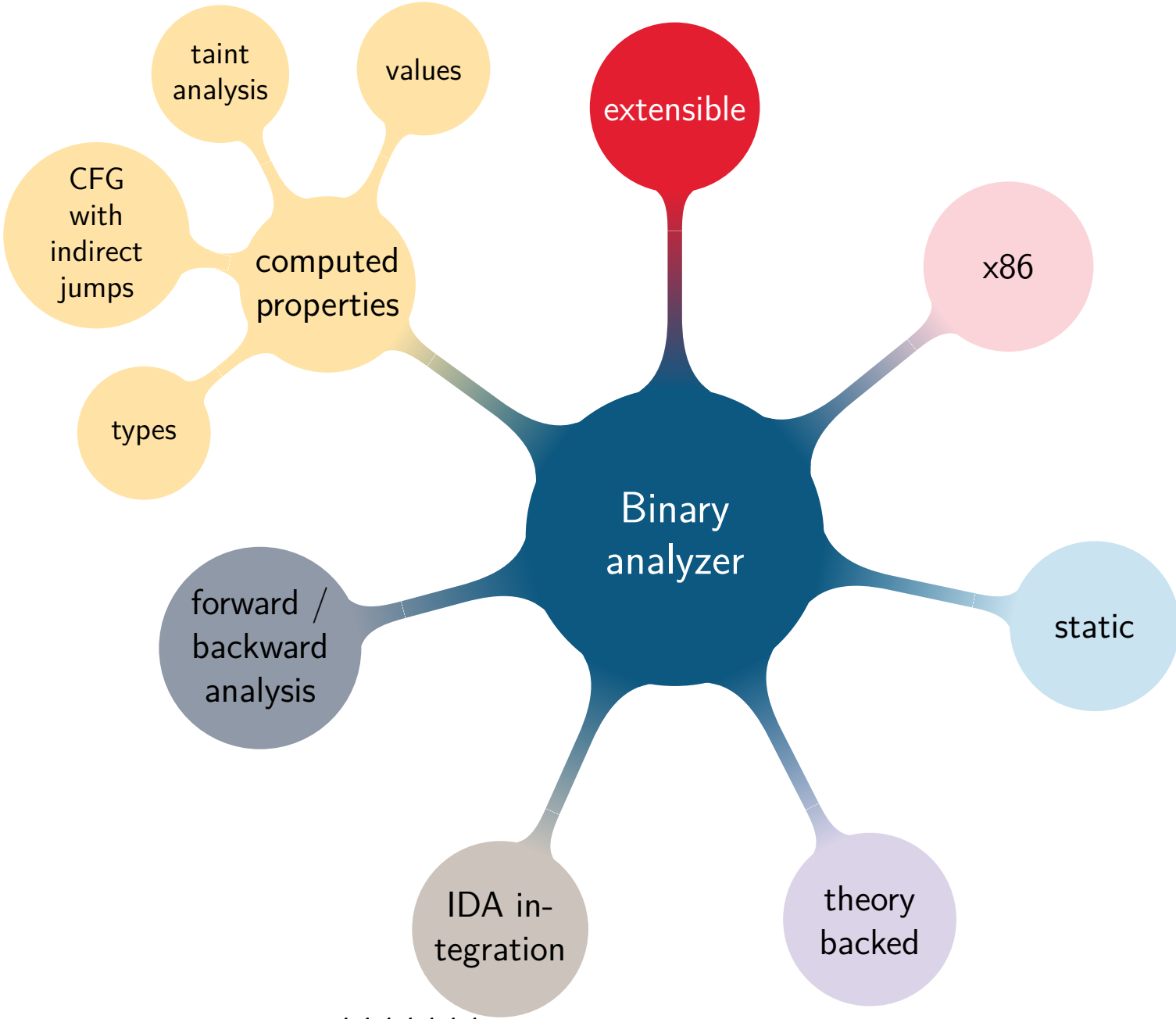
BinCAT (*Binary Code Analysis Toolkit*)



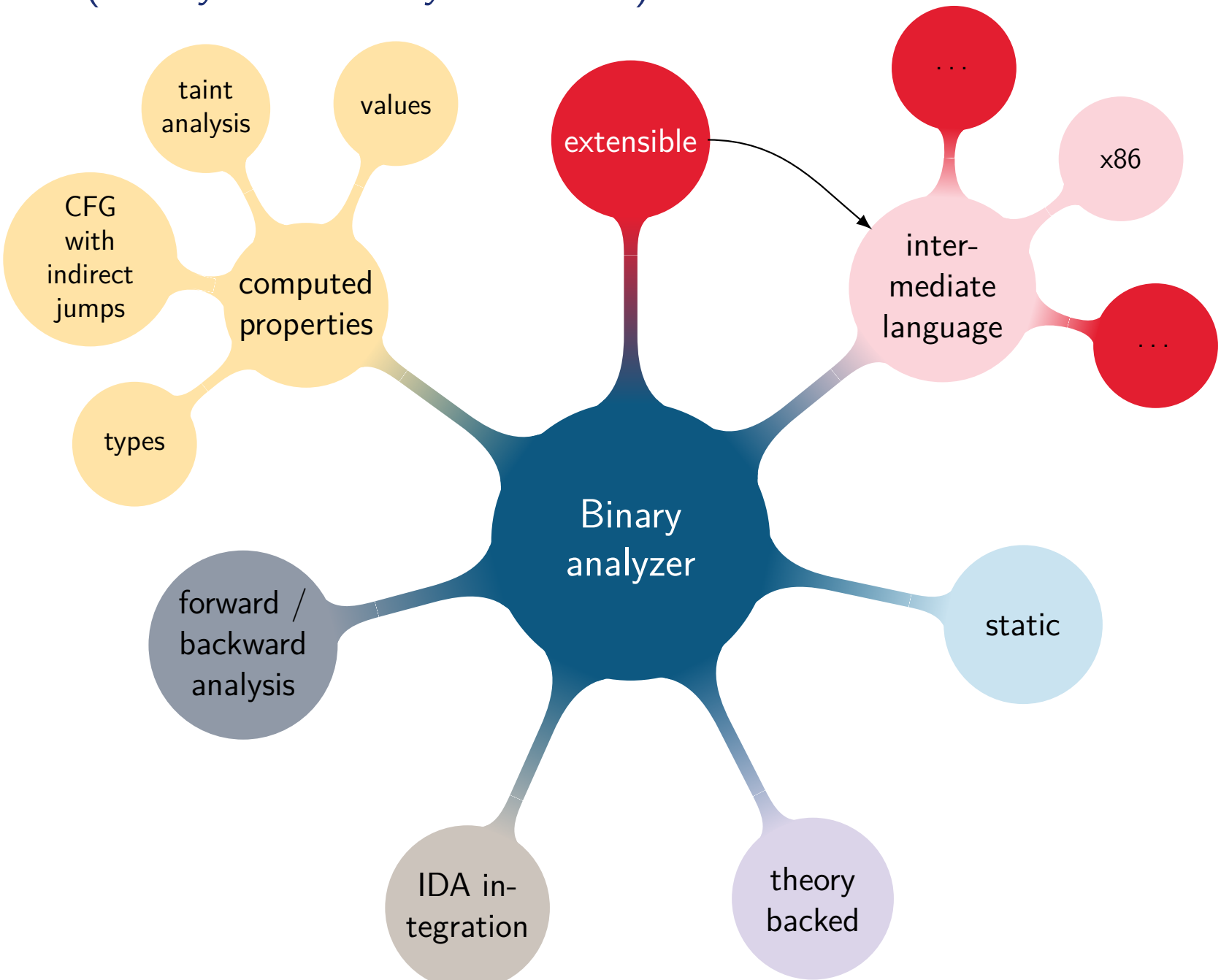
BinCAT (*Binary Code Analysis Toolkit*)



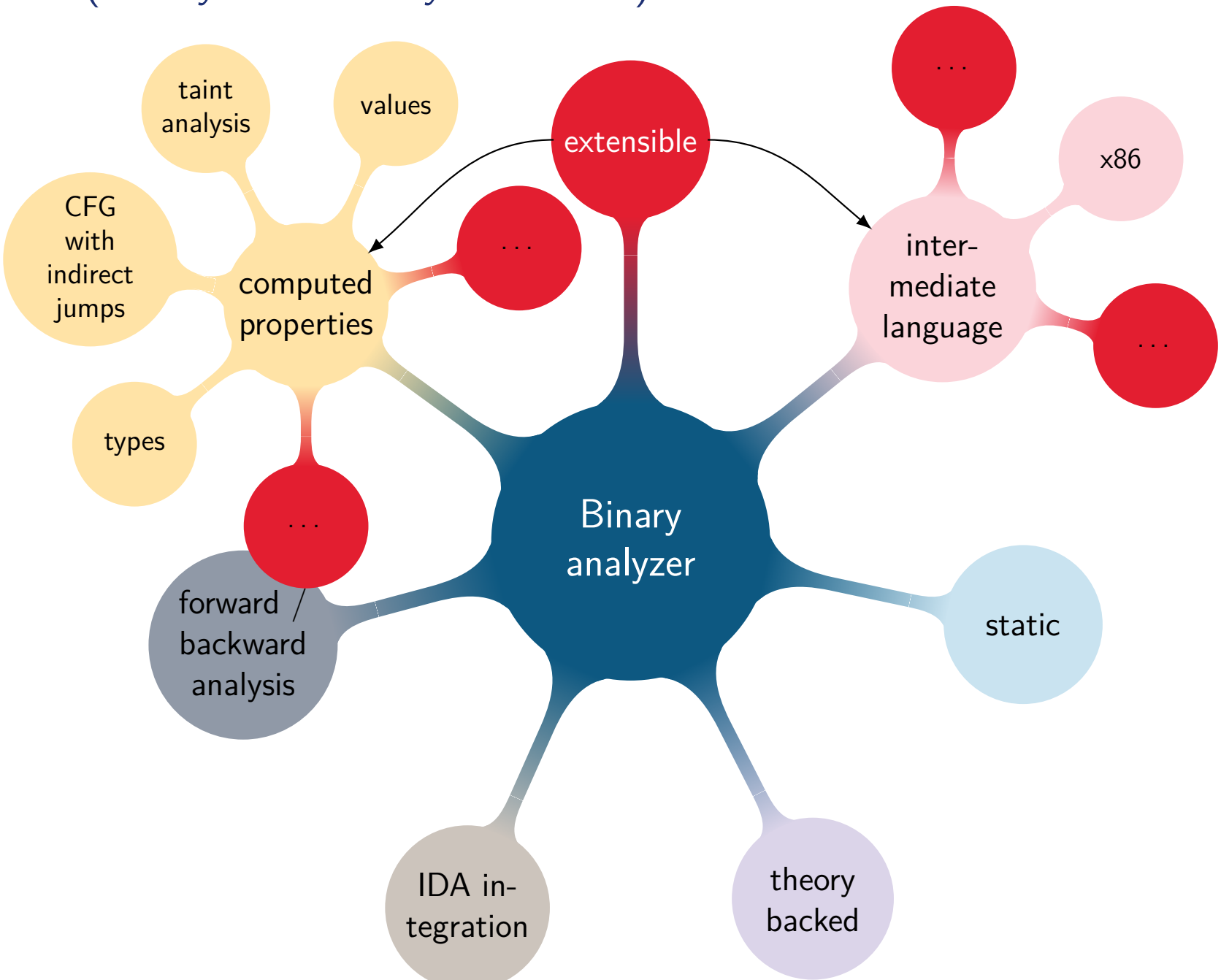
BinCAT (*Binary Code Analysis Toolkit*)



BinCAT (*Binary Code Analysis Toolkit*)



BinCAT (*Binary Code Analysis Toolkit*)



Plan

Example: keygenme

```
$ ./get_key
Usage: ./get_key company department name licence
```

Example: keygenme

```
$ ./get_key
Usage: ./get_key company department name licence
```

```
$ ./get_key company department name wrong_serial
```

Example: keygenme

```
$ ./get_key
Usage: ./get_key company department name licence
```

```
$ ./get_key company department name wrong_serial
Licence=>[025E60CB08F00A1A23F236CC78FC819CE6590DD7]
Invalid serial wrong_serial
```

Example: keygenme

```
$ ./get_key
Usage: ./get_key company department name licence
```

```
$ ./get_key company department name wrong_serial
Licence=>[025E60CB08F00A1A23F236CC78FC819CE6590DD7]
Invalid serial wrong_serial
```

```
$ ./get_key company department name 025E60CB0[...]
```

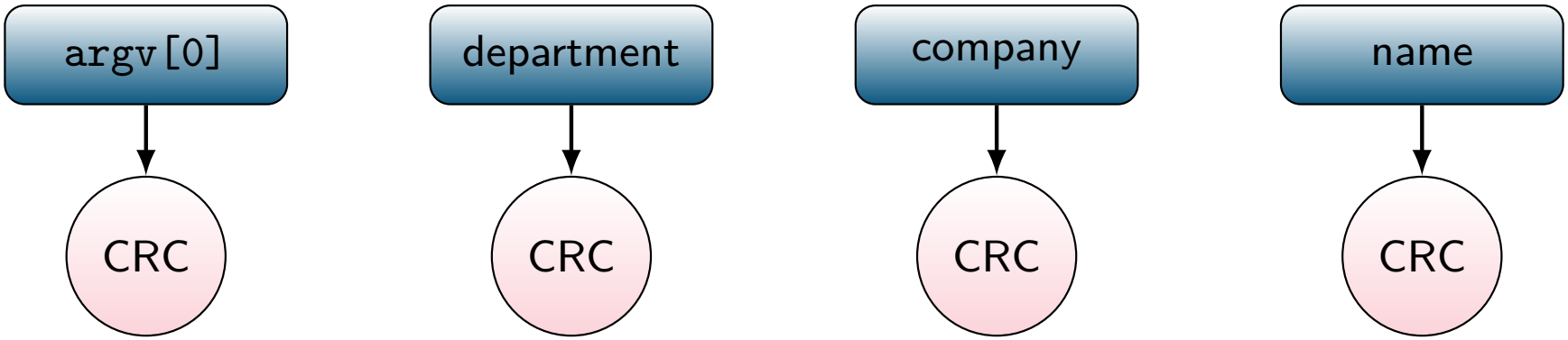
Example: keygenme

```
$ ./get_key
Usage: ./get_key company department name licence
```

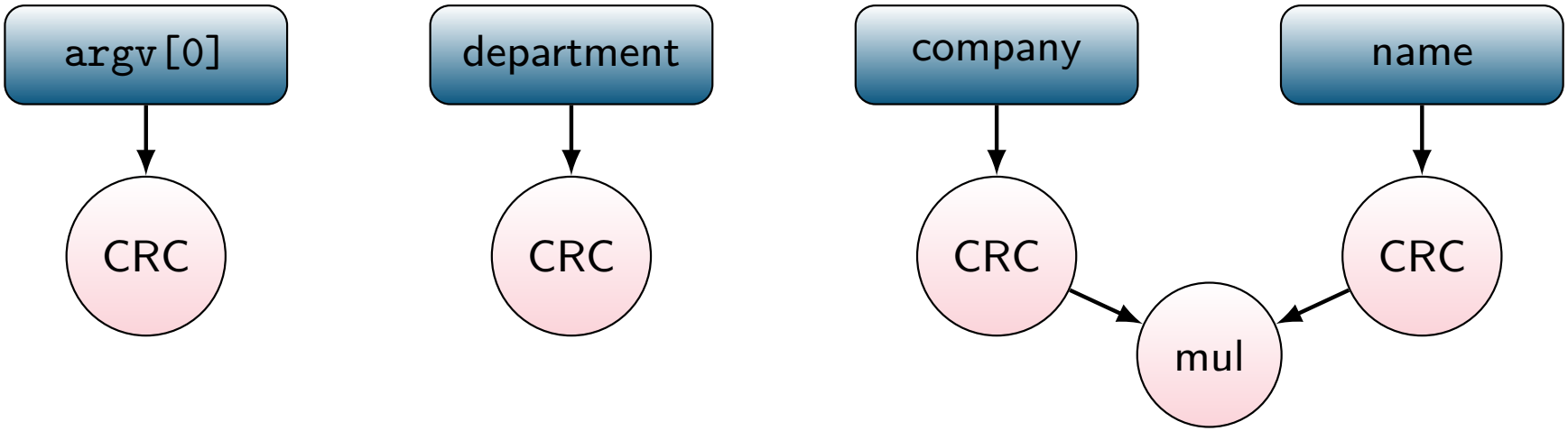
```
$ ./get_key company department name wrong_serial
Licence=>[025E60CB08F00A1A23F236CC78FC819CE6590DD7]
Invalid serial wrong_serial
```

```
$ ./get_key company department name 025E60CB0[...]
Licence=>[025E60CB08F00A1A23F236CC78FC819CE6590DD7]
Thank you for registering !
```

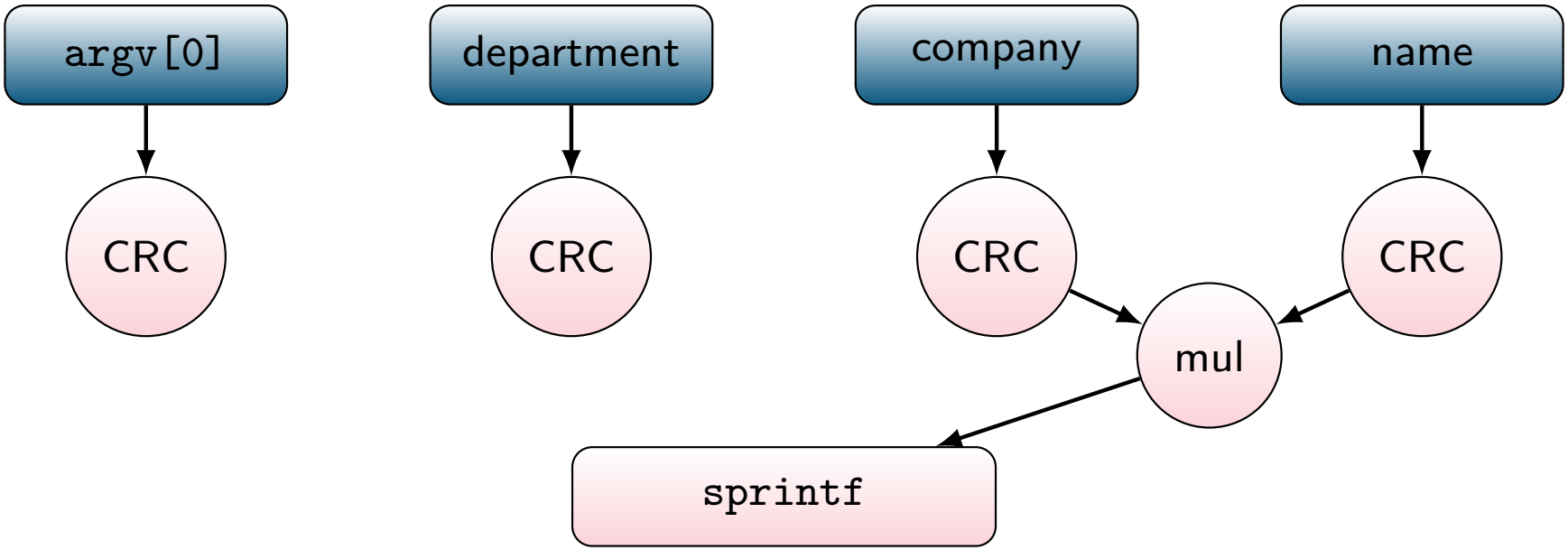
Keygenme: data flow



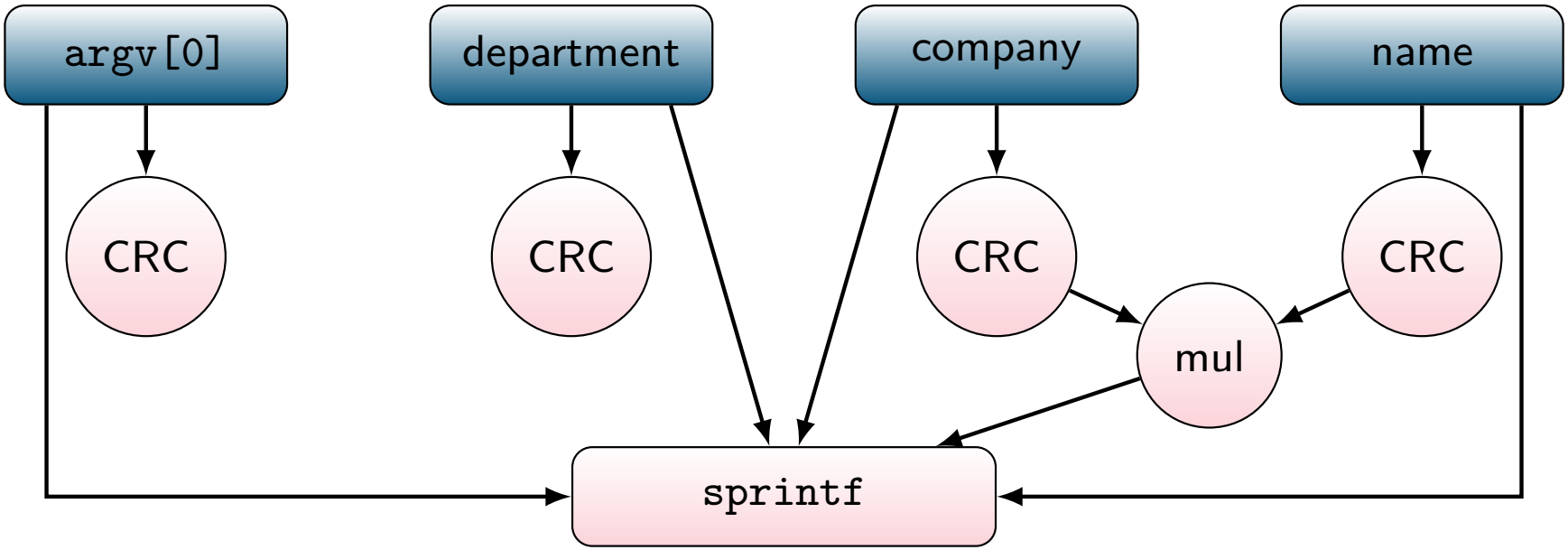
Keygenme: data flow



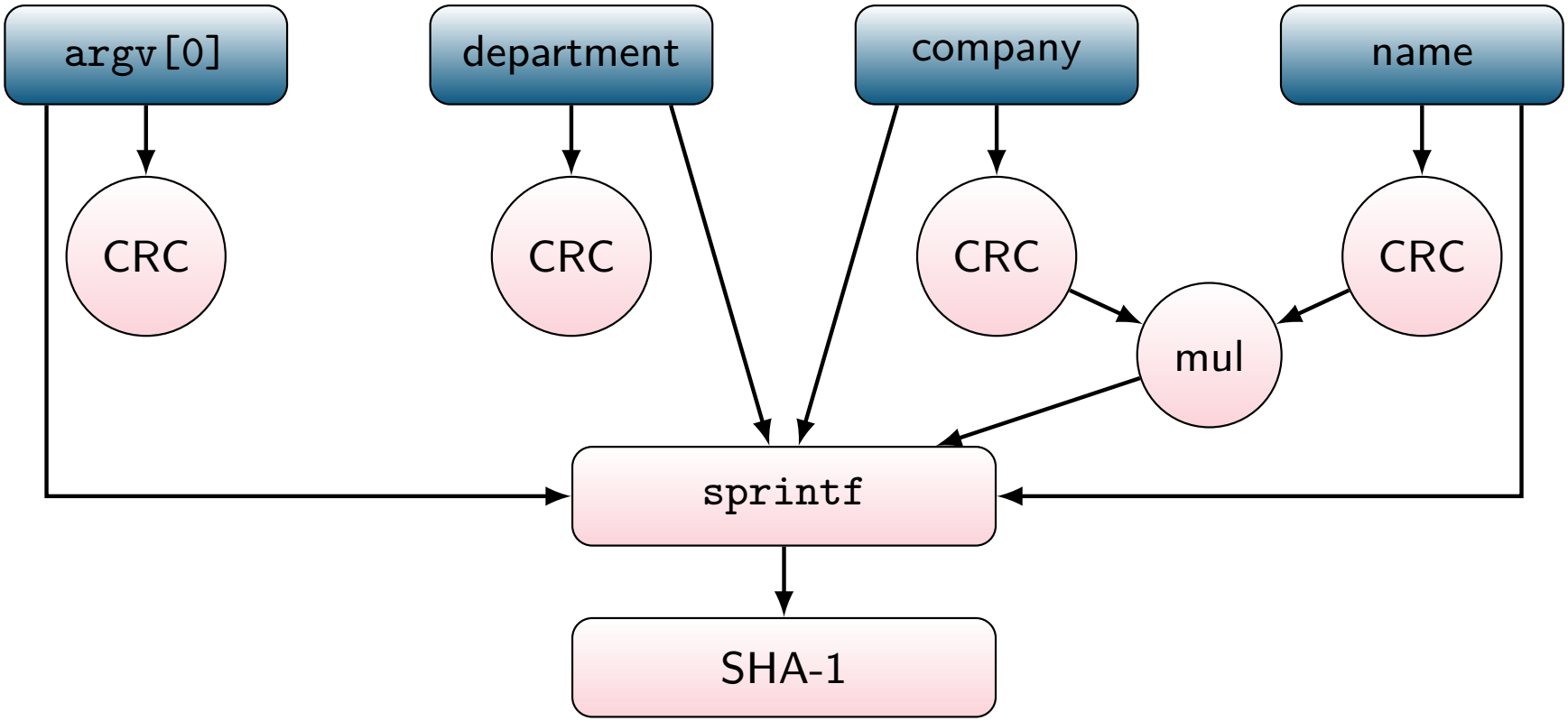
Keygenme: data flow



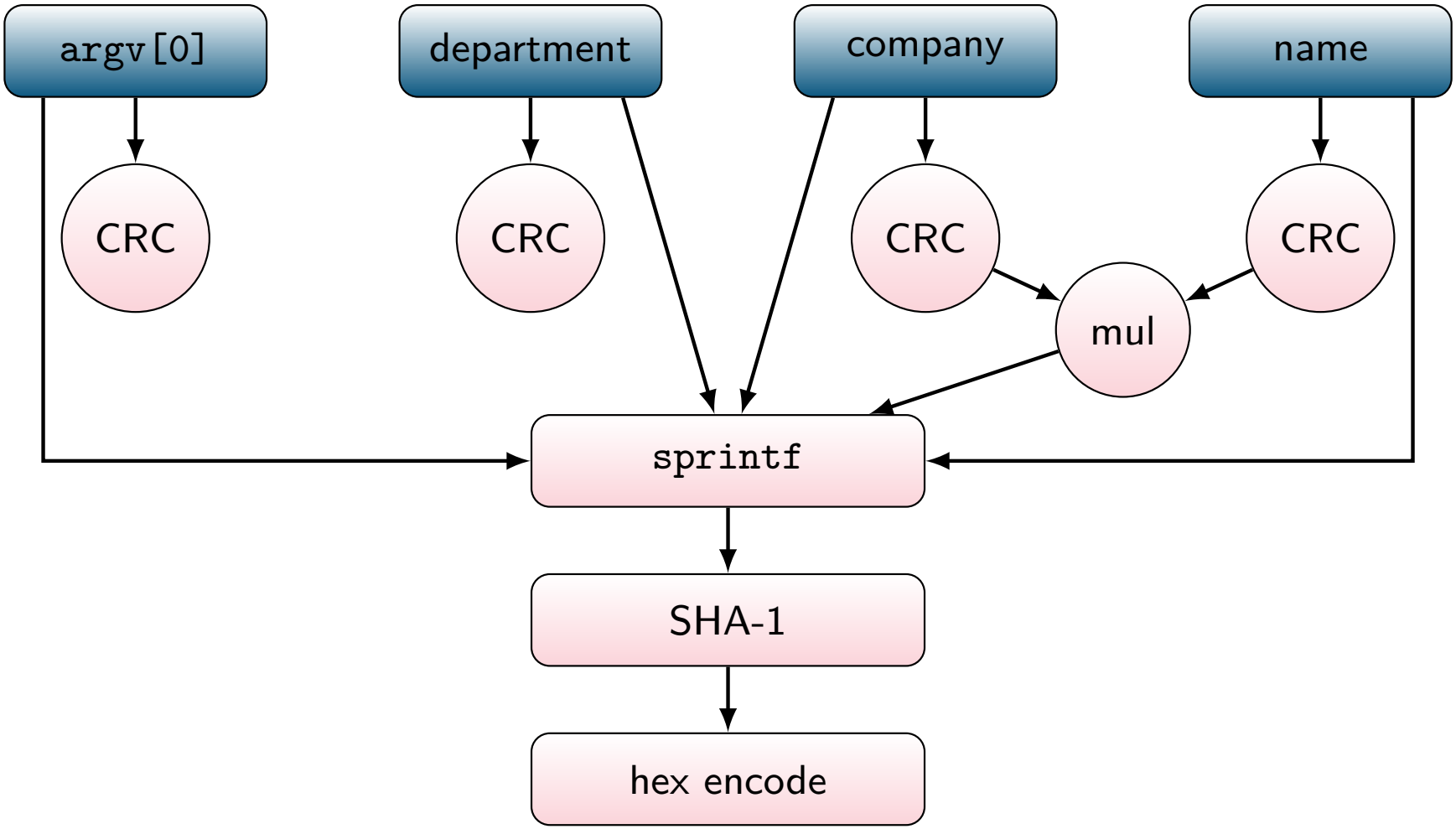
Keygenme: data flow



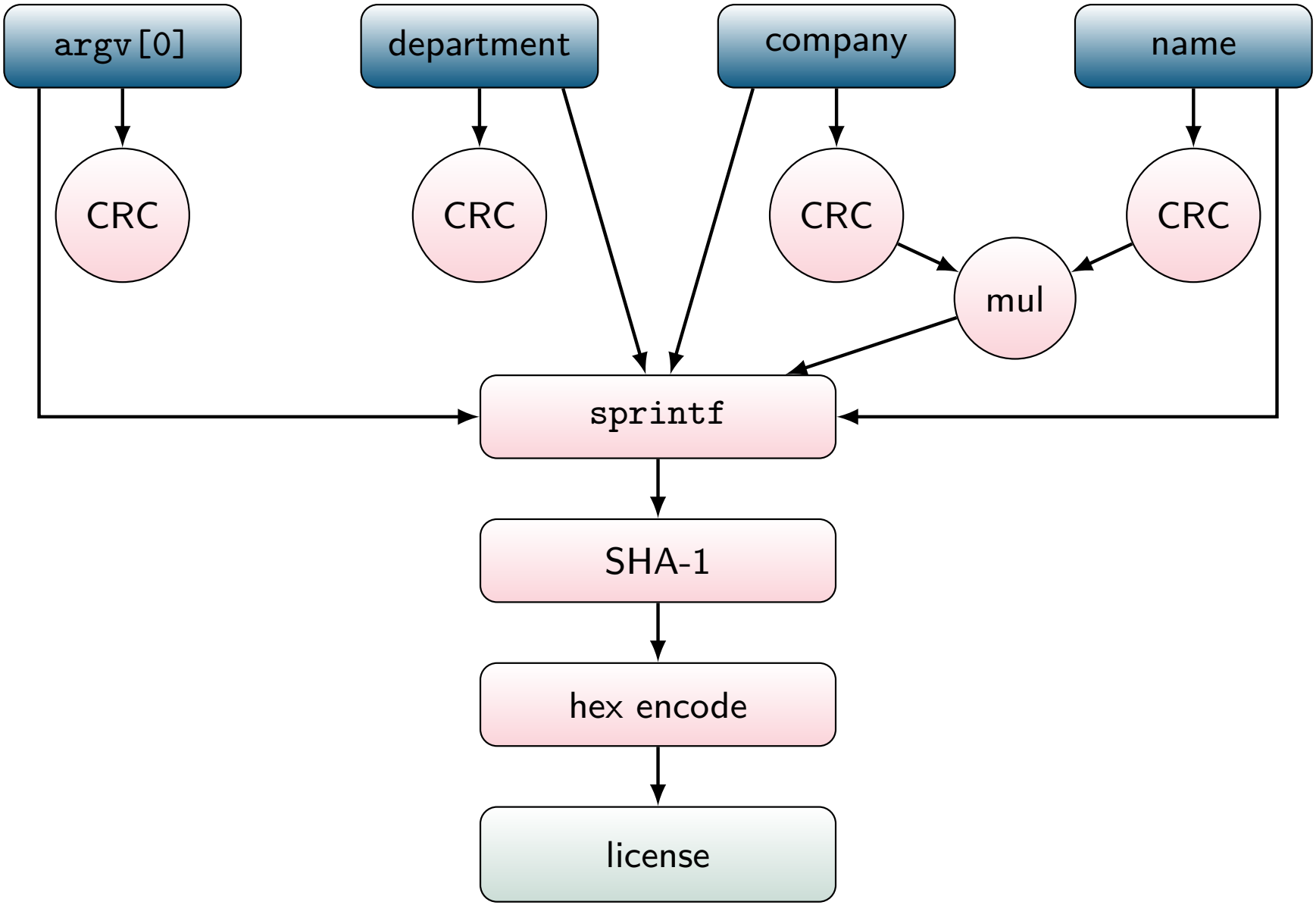
Keygenme: data flow



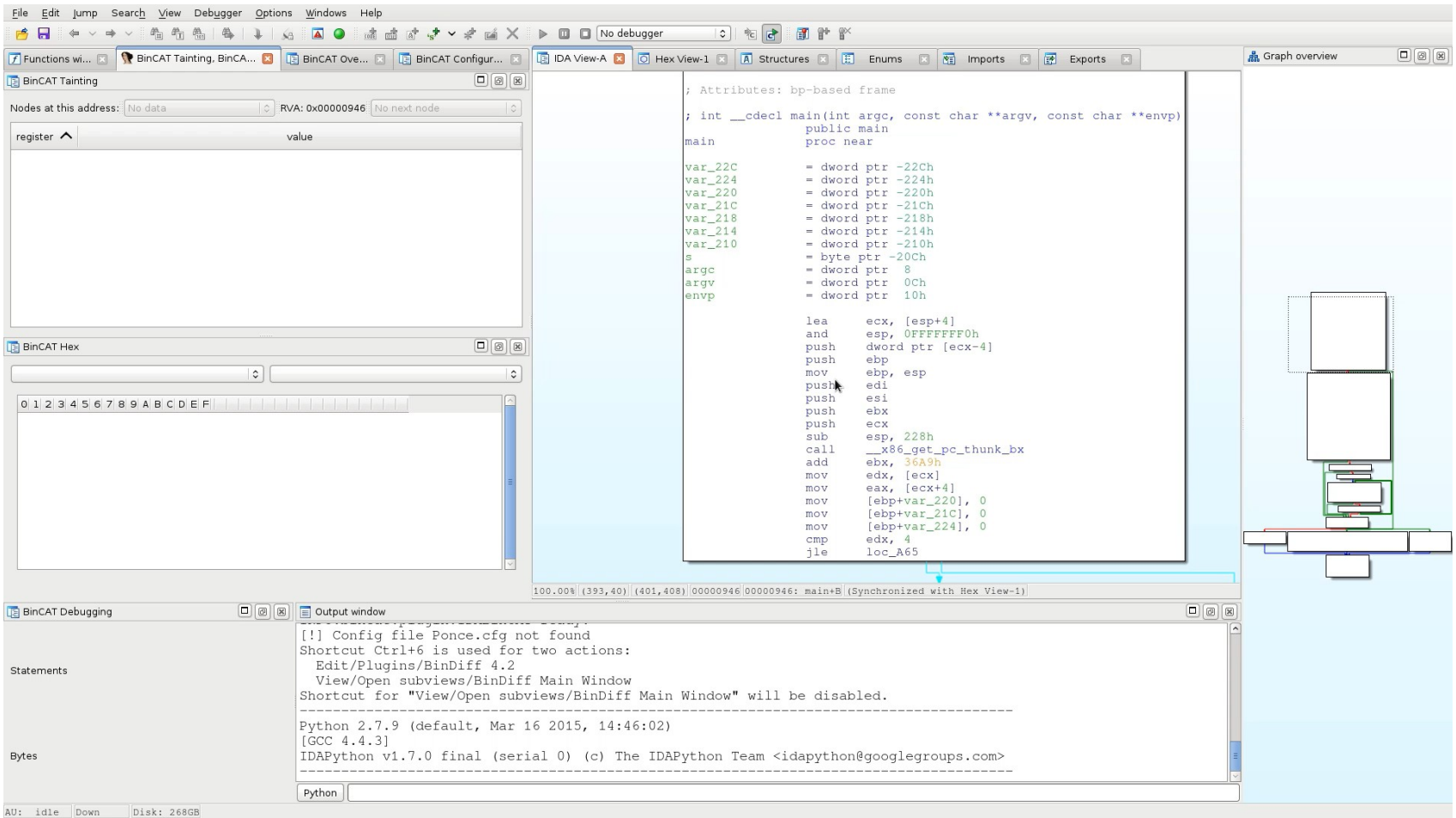
Keygenme: data flow



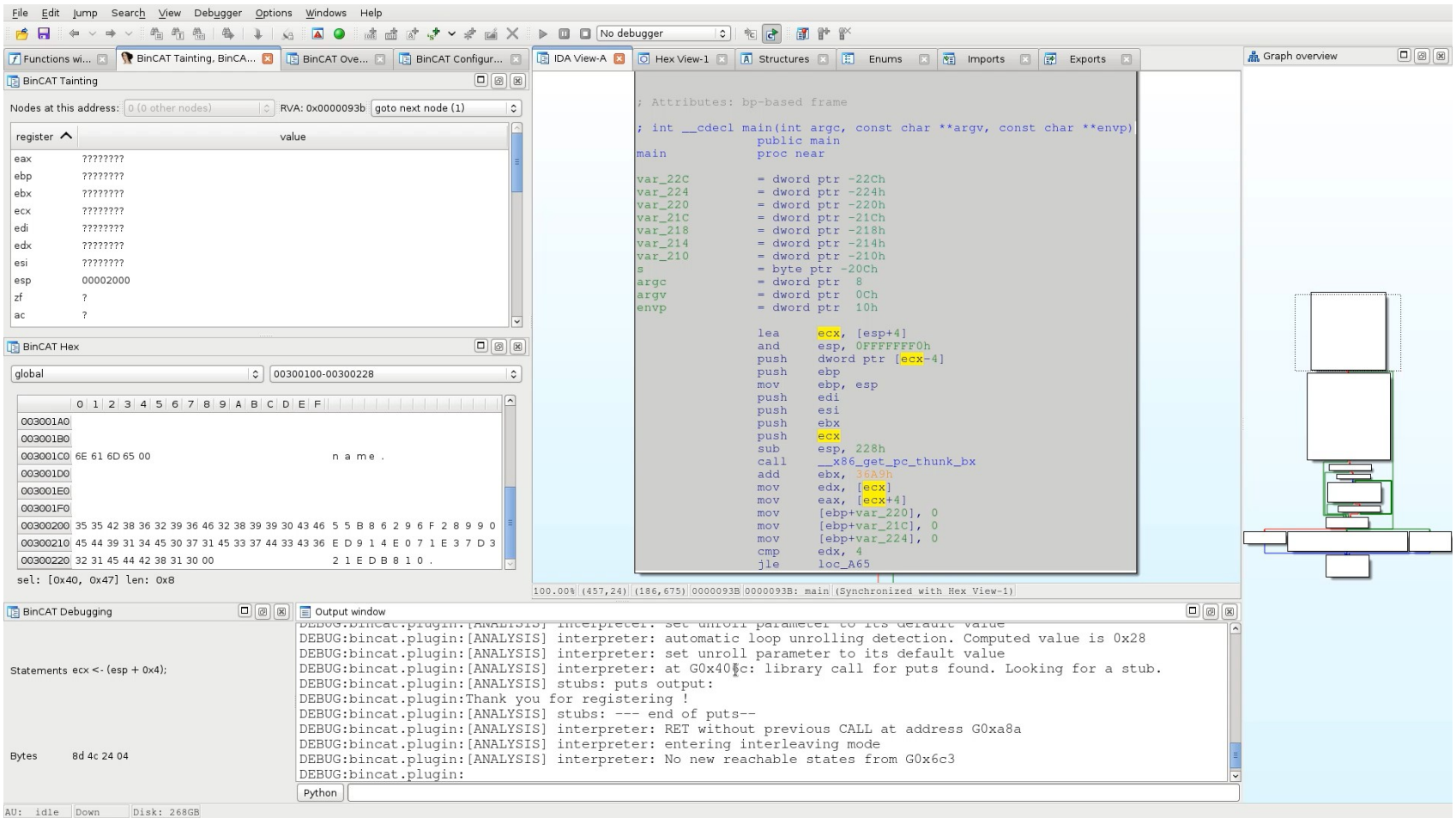
Keygenme: data flow



Demo 1: BinCAT usage

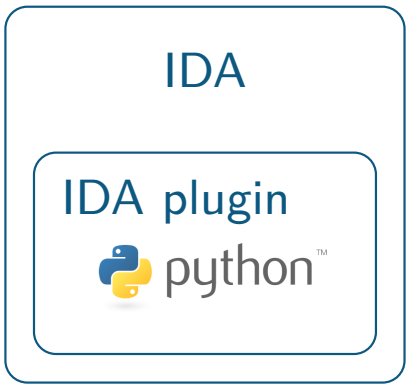


Demo 2: Tainting

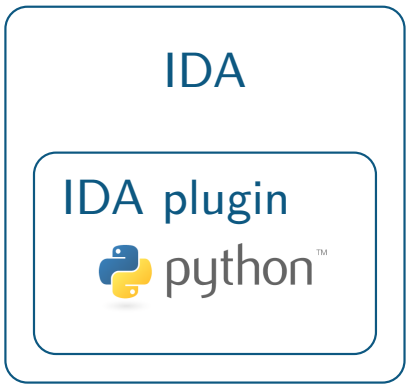


Plan

Architecture

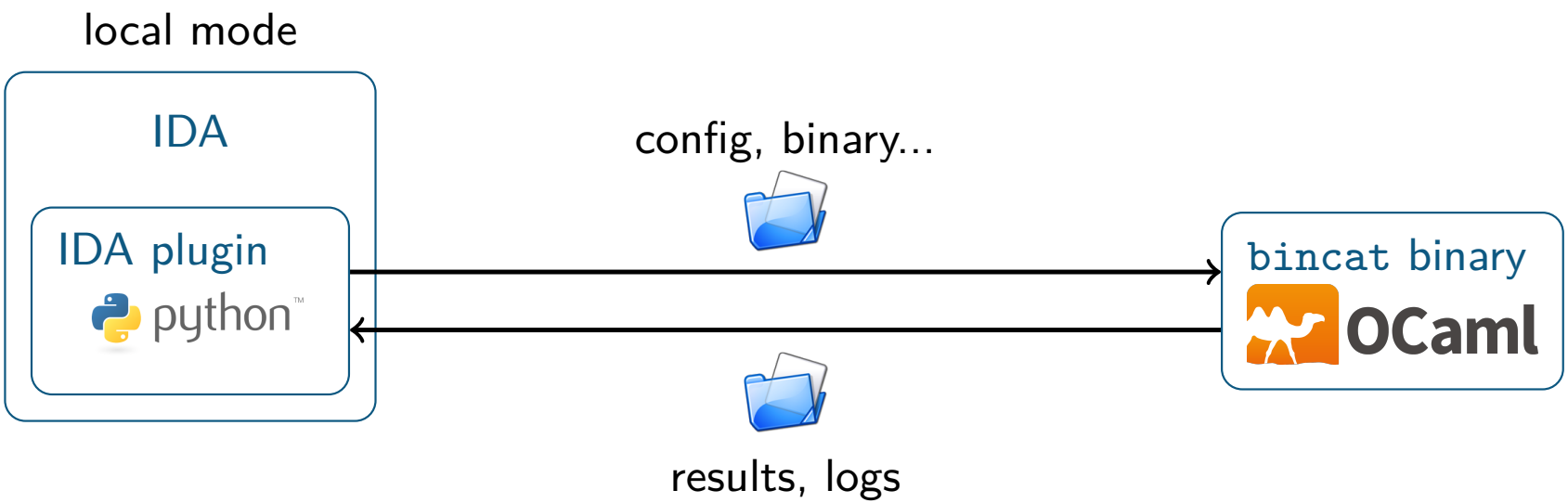


Architecture

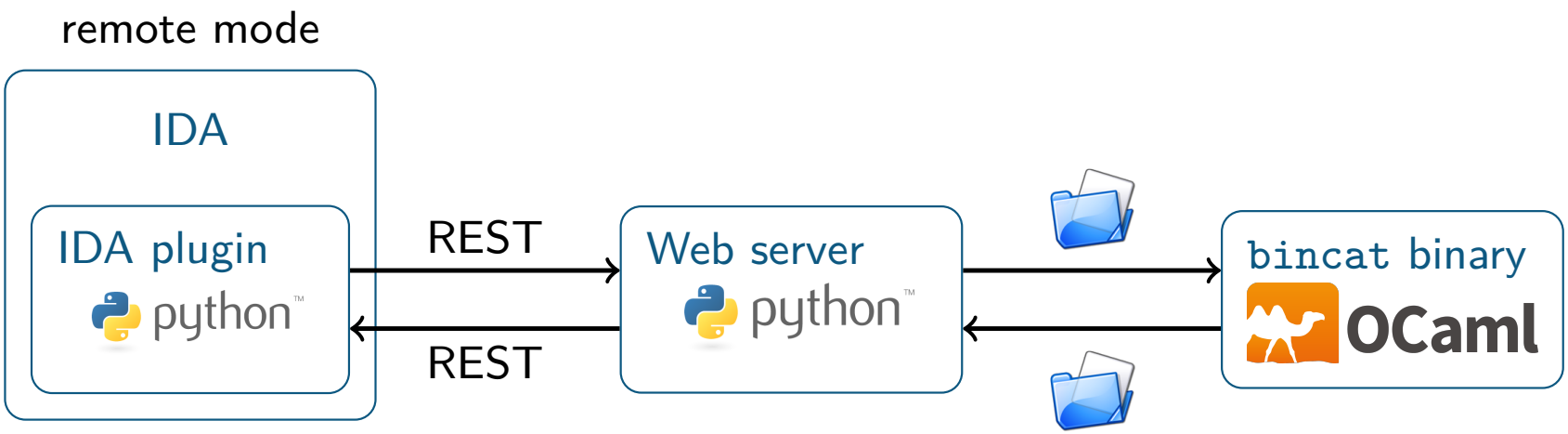


Architecture

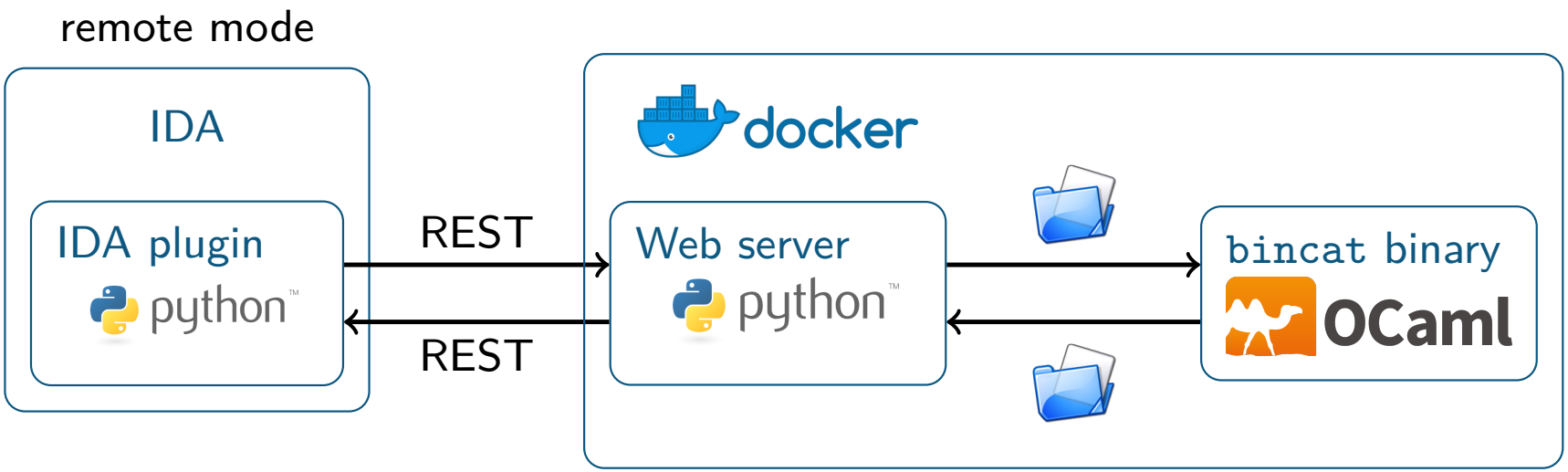
Architecture



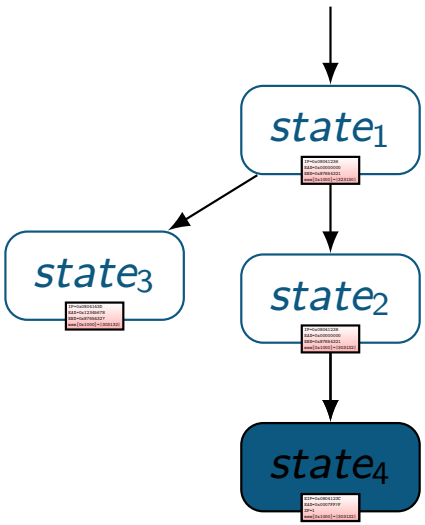
Architecture



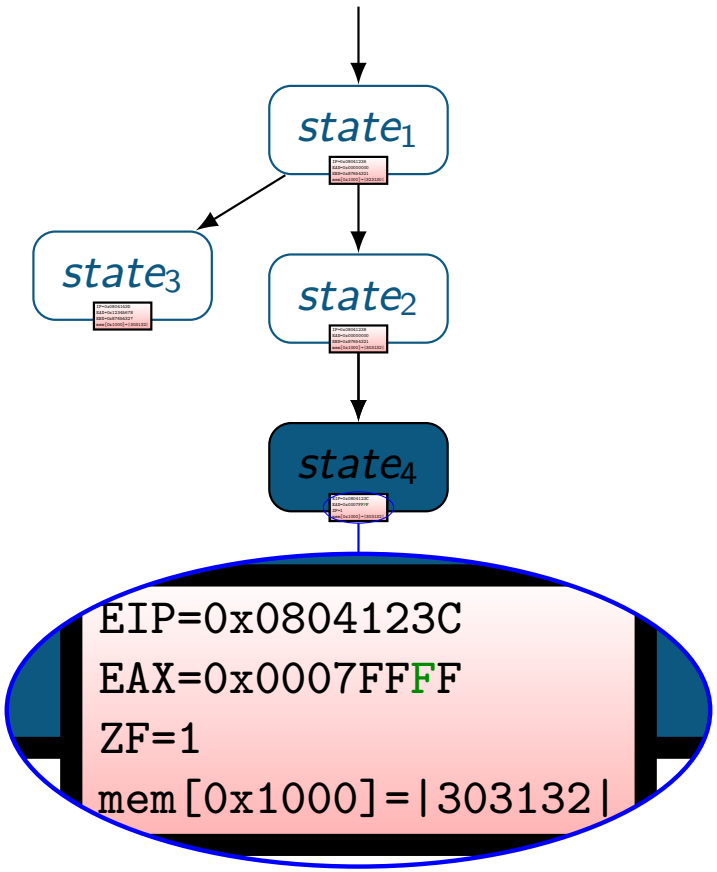
Architecture



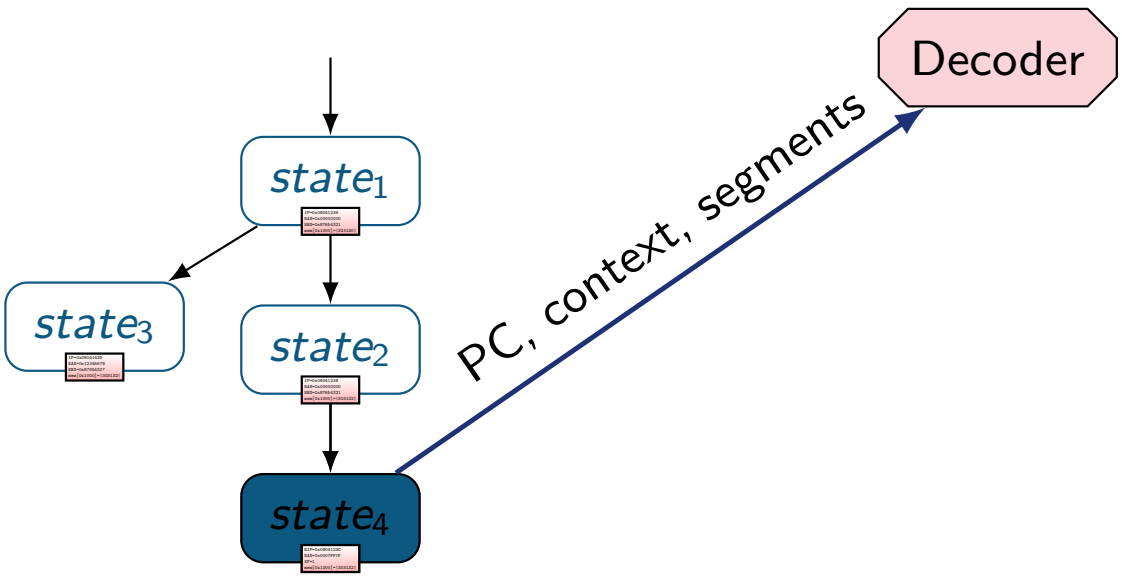
Control flow graph reconstruction



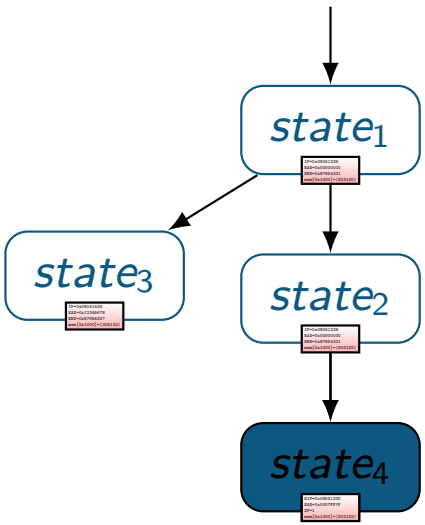
Control flow graph reconstruction

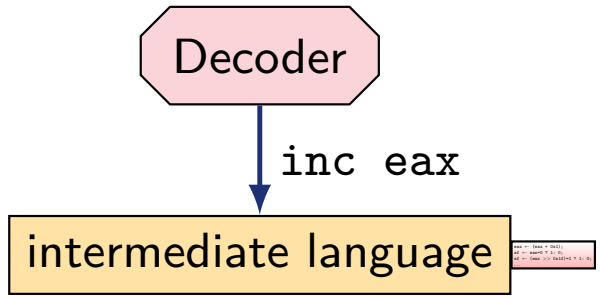


Control flow graph reconstruction

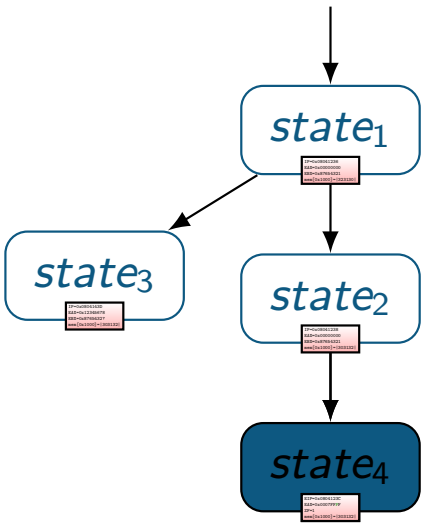


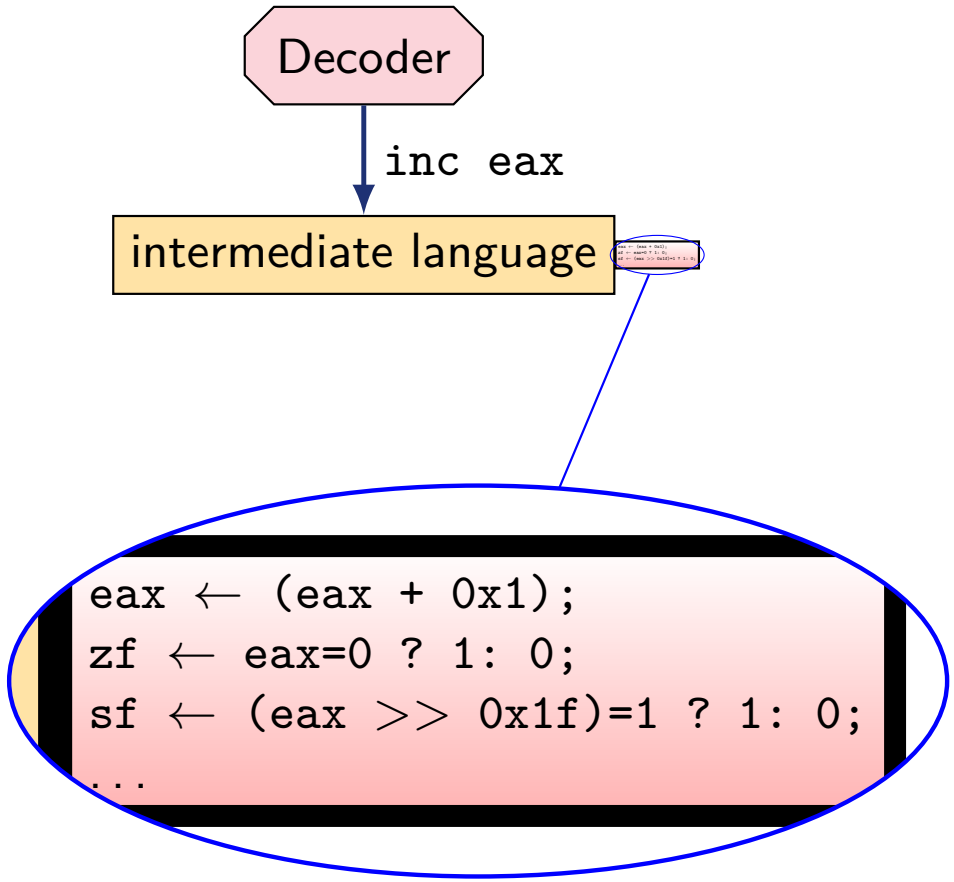
Control flow graph reconstruction



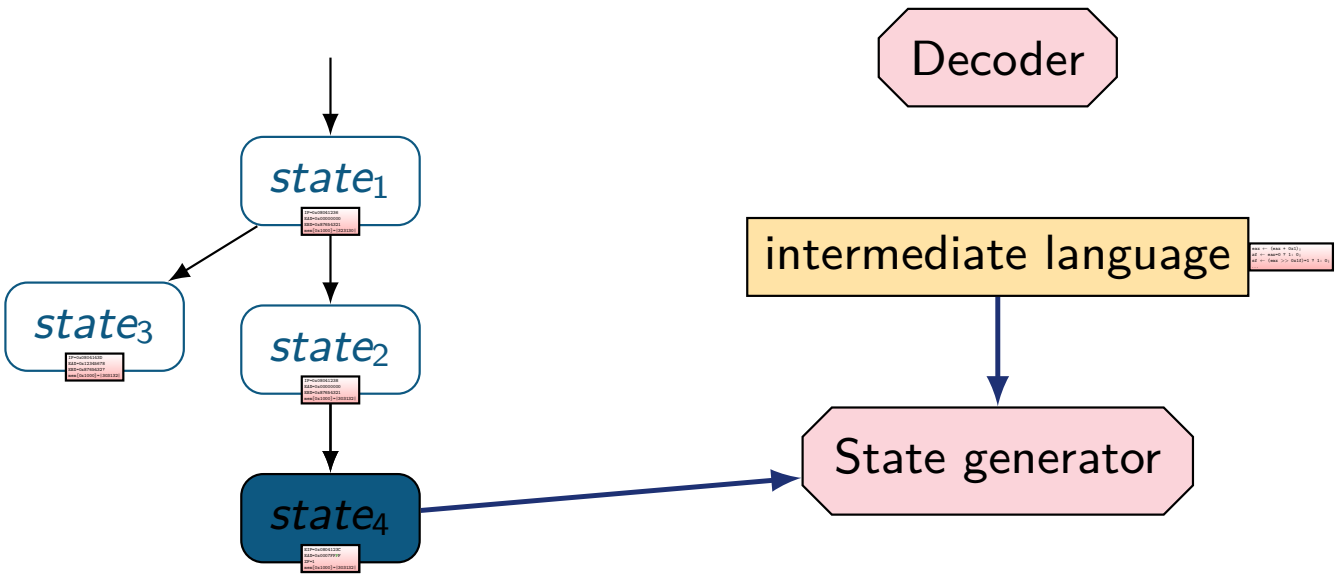


Control flow graph reconstruction

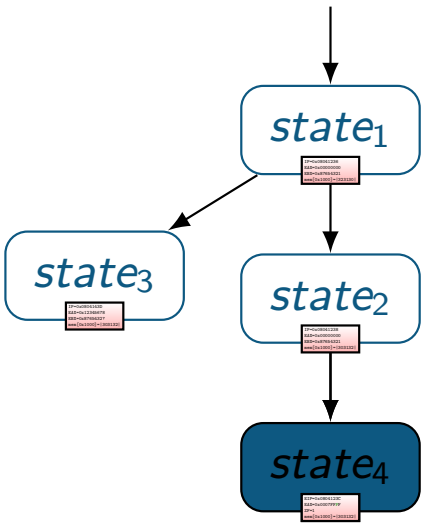


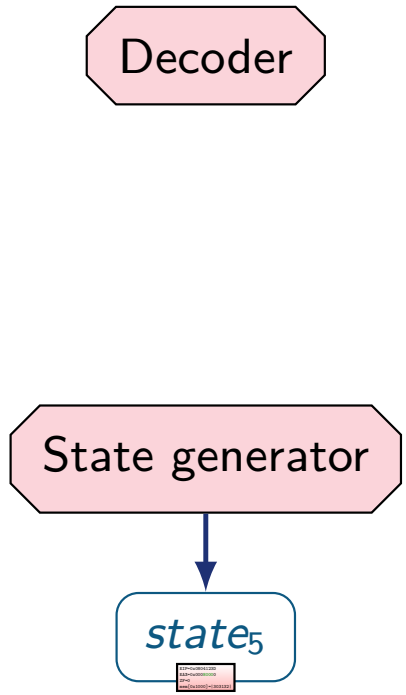


Control flow graph reconstruction

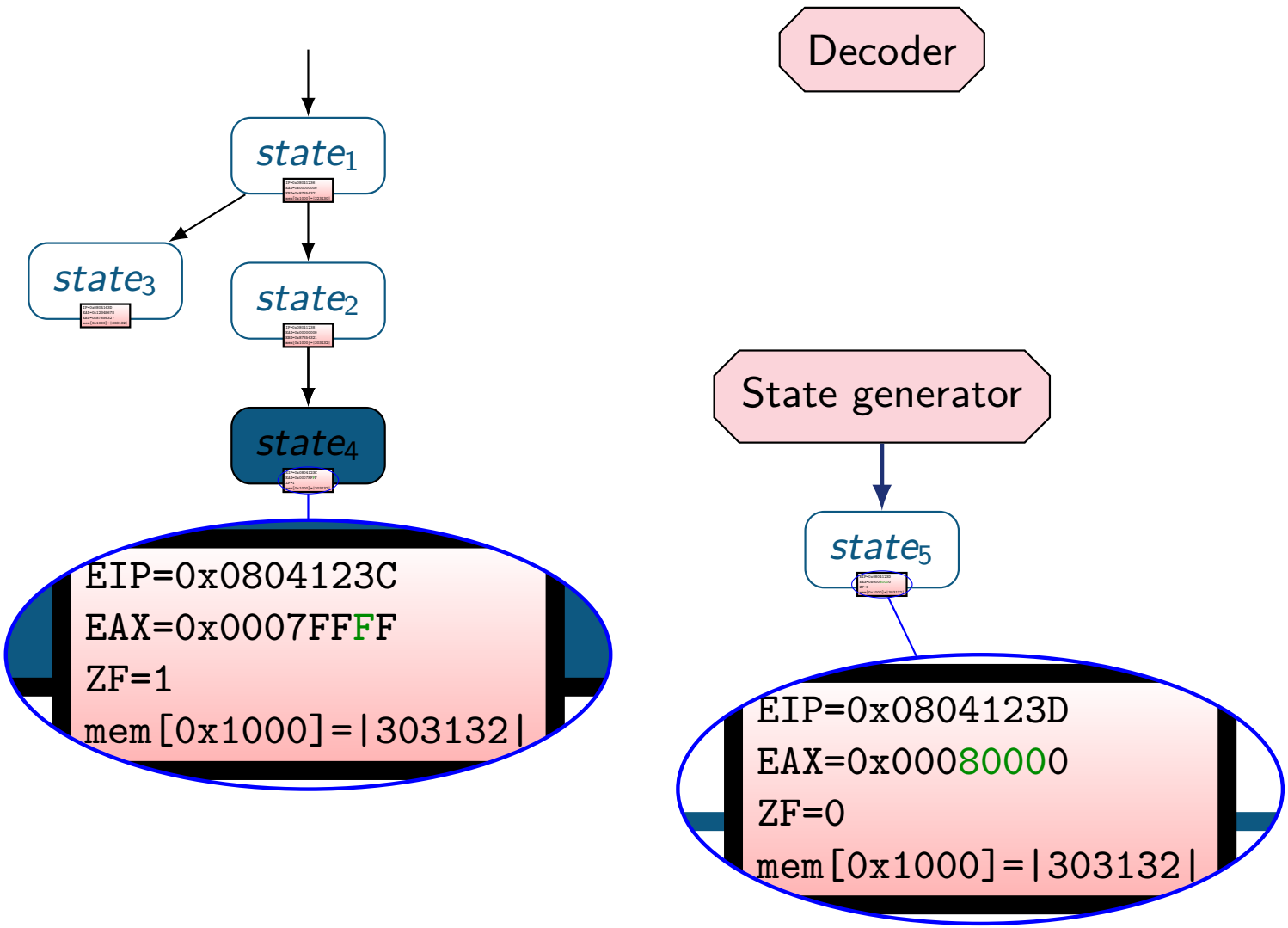


Control flow graph reconstruction

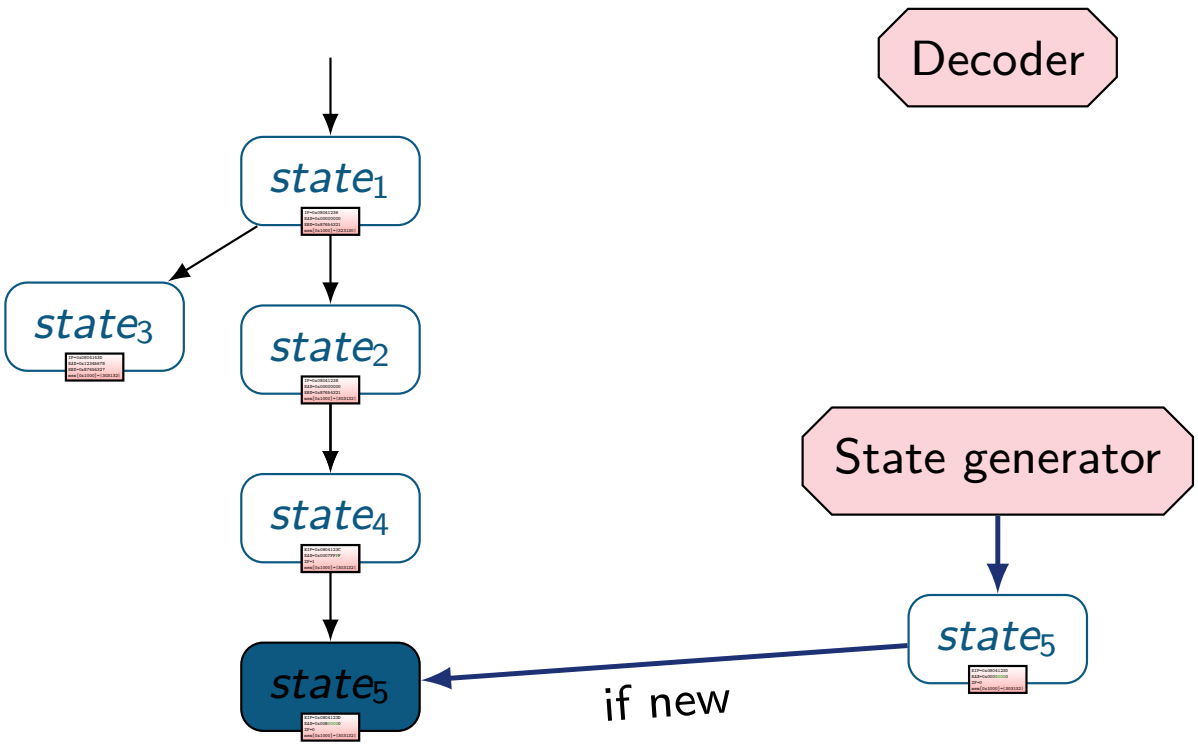




Control flow graph reconstruction



Control flow graph reconstruction

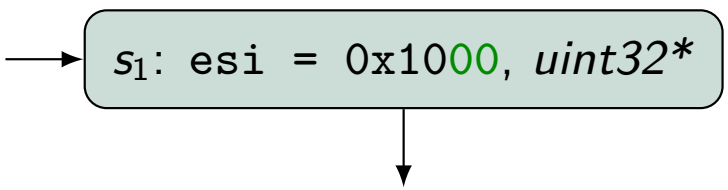


Formal correctness: static analysis by abstract interpretation

- operations on values/taint/types are done on *abstract* objects which represent sets of values/taint/types
  ex: $0 \equiv \{0\}$, $? \equiv \{\text{integers}\}$, $\text{Struct} \equiv \{\text{C structs}\}$
- abstract computations are always an *overapproximation* of actual ones
- approximation example: loop widening (∇)

Formal correctness: static analysis by abstract interpretation

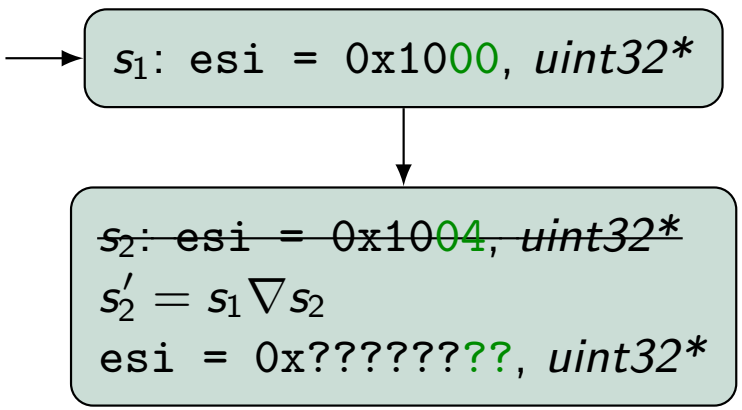
- operations on values/taint/types are done on *abstract* objects which represent sets of values/taint/types
  ex: $0 \equiv \{0\}$, $? \equiv \{\text{integers}\}$, $\text{Struct} \equiv \{\text{C structs}\}$
- abstract computations are always an *overapproximation* of actual ones
- approximation example: loop widening (∇)



while esi $<$ 0x8000
  • esi $=$ esi $+$ 4

Formal correctness: static analysis by abstract interpretation

- operations on values/taint/types are done on *abstract* objects which represent sets of values/taint/types
  ex: $0 \equiv \{0\}$, $? \equiv \{\text{integers}\}$, $\text{Struct} \equiv \{\text{C structs}\}$
- abstract computations are always an *overapproximation* of actual ones
- approximation example: loop widening (∇)

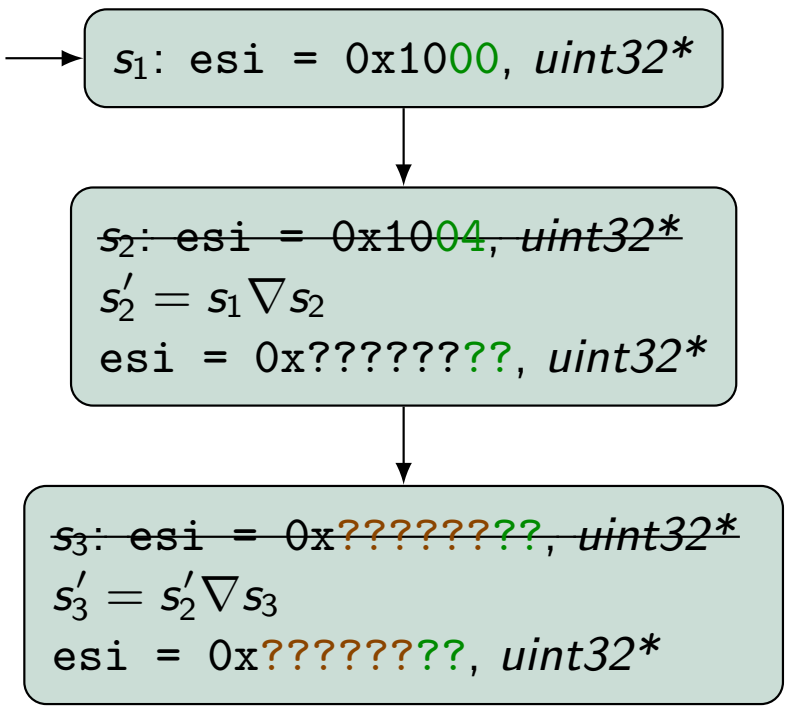


while esi $<$ 0x8000
  • esi $=$ esi $+$ 4

- Idea:
  - what is stable is kept
    Ex: type
  - what changes is overapproximated
    Ex: value

Formal correctness: static analysis by abstract interpretation

- operations on values/taint/types are done on *abstract* objects which represent sets of values/taint/types
  ex: $0 \equiv \{0\}$, $? \equiv \{\text{integers}\}$, $\text{Struct} \equiv \{\text{C structs}\}$
- abstract computations are always an *overapproximation* of actual ones
- approximation example: loop widening (∇)

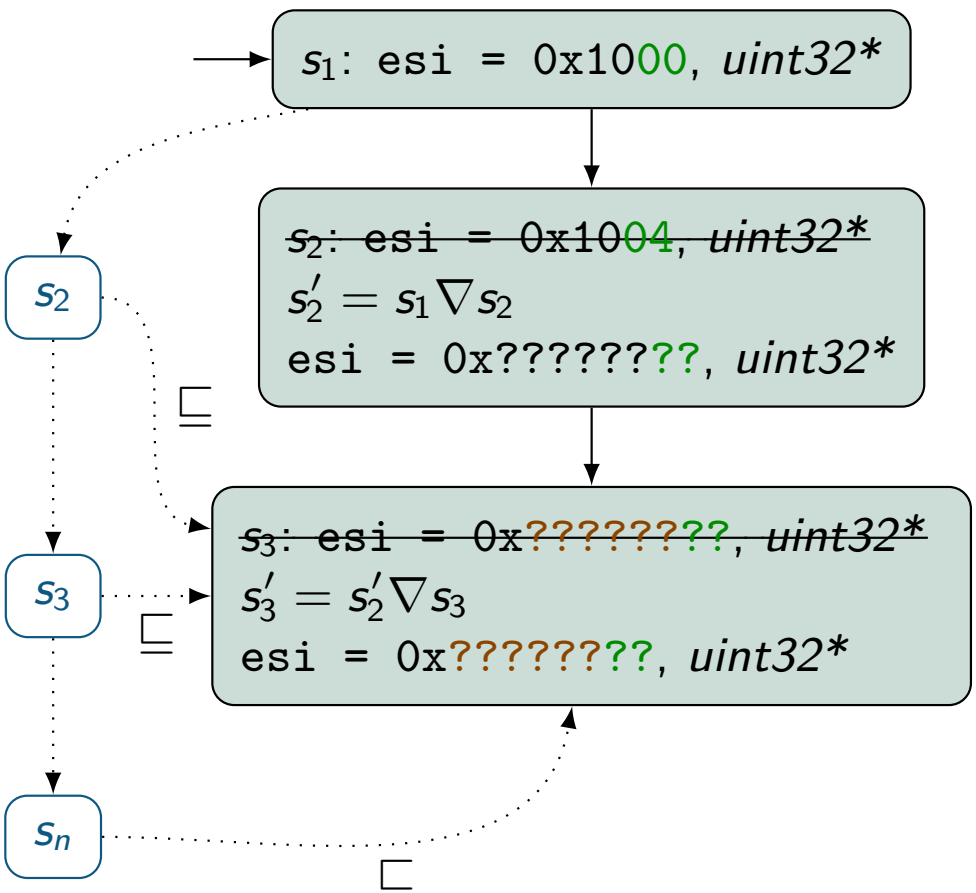


while esi $<$ 0x8000
  • esi $=$ esi $+$ 4

- Idea:
  - what is stable is kept
    Ex: type
  - what changes is overapproximated
    Ex: value

Formal correctness: static analysis by abstract interpretation

- operations on values/taint/types are done on *abstract* objects which represent sets of values/taint/types
  ex: $0 \equiv \{0\}$, $? \equiv \{\text{integers}\}$, $\text{Struct} \equiv \{\text{C structs}\}$
- abstract computations are always an *overapproximation* of actual ones
- approximation example: loop widening (∇)

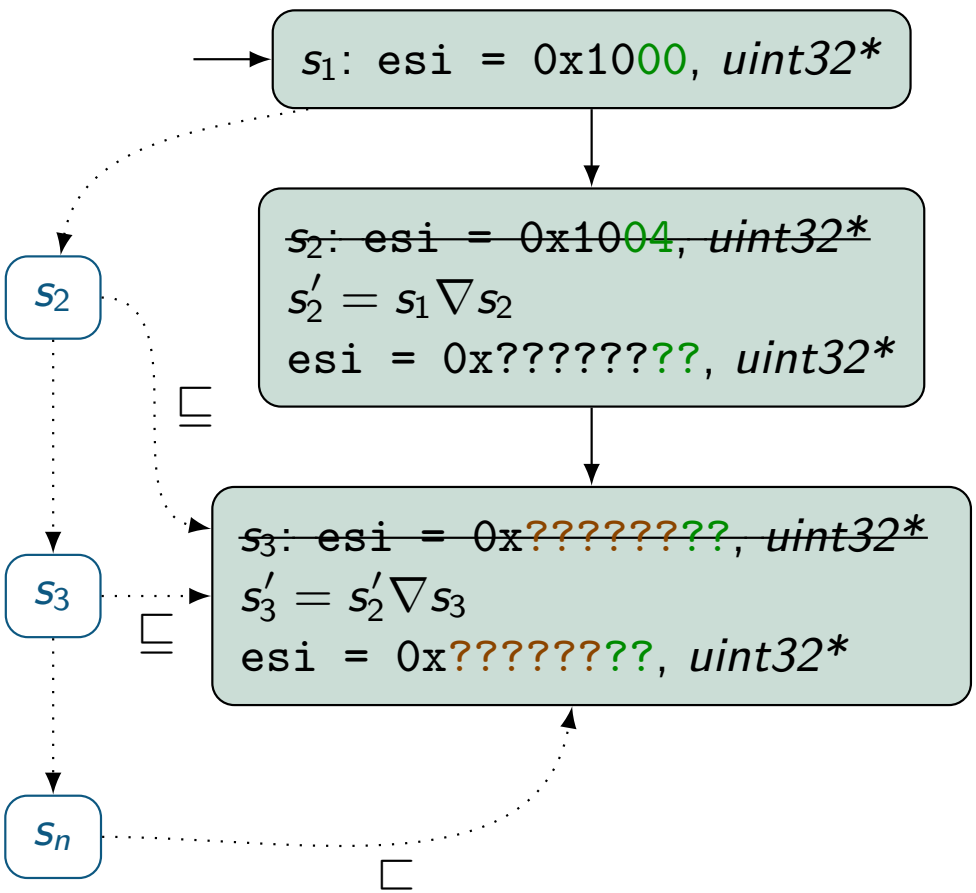


while esi $<$ 0x8000
  • esi $=$ esi $+$ 4

- Idea:
  - what is stable is kept
    Ex: type
  - what changes is overapproximated
    Ex: value
- Theorem 1: (s_i') sequence is ultimately stationary
- Theorem 2: fixpoint s_f' is an overapproximation of the real execution trace

Formal correctness: static analysis by abstract interpretation

- operations on values/taint/types are done on *abstract* objects which represent sets of values/taint/types
  ex: $0 \equiv \{0\}$, $? \equiv \{\text{integers}\}$, $\text{Struct} \equiv \{\text{C structs}\}$
- abstract computations are always an *overapproximation* of actual ones
- approximation example: loop widening (∇)



while esi $<$ 0x8000
  • esi $=$ esi $+$ 4

- Idea:
  - what is stable is kept
    Ex: type
  - what changes is overapproximated
    Ex: value
- Theorem 1: (s_i') sequence is ultimately stationary
- Theorem 2: fixpoint s_f' is an overapproximation of the real execution trace
- some techniques allow for precision recovery

Empirical testing

- Theory is correct.

Empirical testing

- Theory is correct.
- *In theory*, the implementation too.

Empirical testing

- Theory is correct.
- *In theory*, the implementation too.
- In practice, a lot of things are complex and bug prone: the decoder, abstract operations, etc.

Empirical testing

- Theory is correct.
- *In theory*, the implementation too.
- In practice, a lot of things are complex and bug prone: the decoder, abstract operations, etc.

$\Longrightarrow$ lots of unit tests

Empirical testing

- Theory is correct.
- *In theory*, the implementation too.
- In practice, a lot of things are complex and bug prone: the decoder, abstract operations, etc.

$\Longrightarrow$ lots of unit tests

  - BinCAT vs CPU: $>$ 67.000 tests for $\simeq$ 55 instructions

Empirical testing

- Theory is correct.
- *In theory*, the implementation too.
- In practice, a lot of things are complex and bug prone: the decoder, abstract operations, etc.

$\Longrightarrow$ lots of unit tests

  - BinCAT vs CPU: $>$ 67.000 tests for $\simeq$ 55 instructions
  - BinCAT vs QEMU `test-i386`: 87% test coverage over $\simeq$ 105 instructions

Empirical testing

- Theory is correct.
- *In theory*, the implementation too.
- In practice, a lot of things are complex and bug prone: the decoder, abstract operations, etc.

$\Longrightarrow$ lots of unit tests

- BinCAT vs CPU: $>$ 67.000 tests for $\simeq$ 55 instructions
- BinCAT vs QEMU `test-i386`: 87% test coverage over $\simeq$ 105 instructions

Analyzer's performance

Example: keygenme

- 6407 instructions analyzed
- RAM usage: 90 MiB
- running time: 6s
- average: $\simeq$ 1060 insn/s

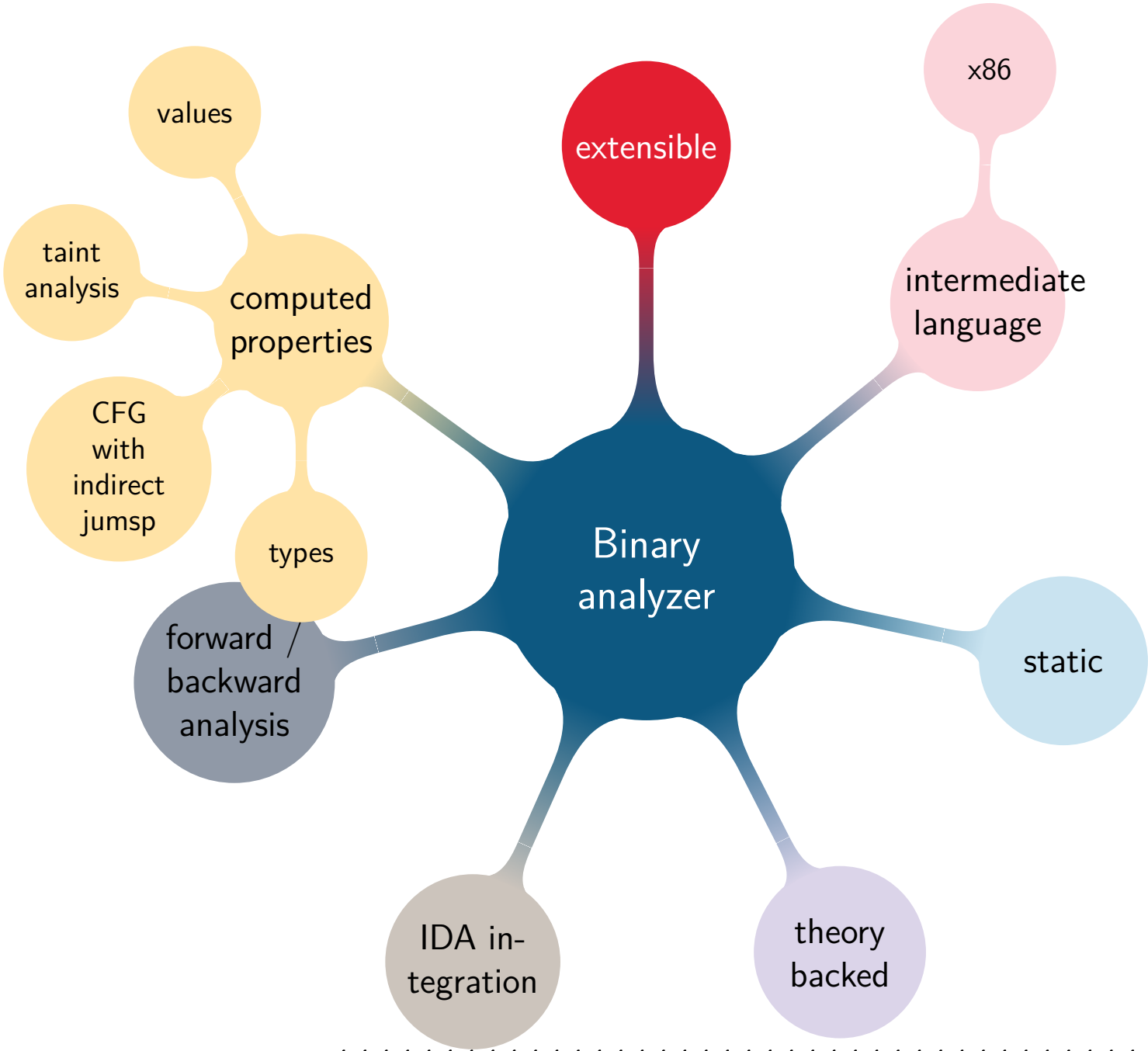
QEMU tests:

- 209 120 instructions analyzed
- RAM usage: 2.3 GiB
- running time: 23 min 30 s
- average: $\simeq$ 150 insn/s

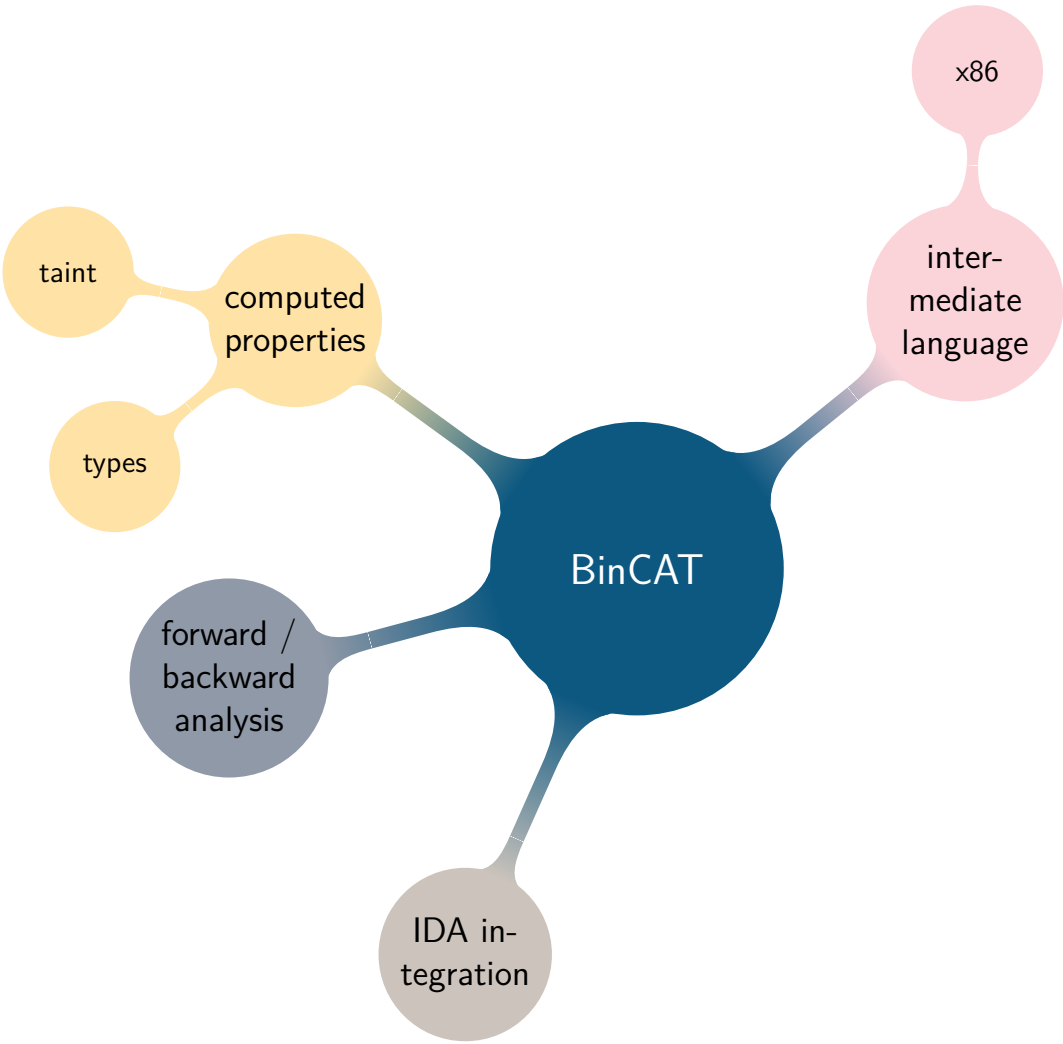
Intel Core i7-6700K CPU @ 4,00GHz

Plan

Conclusion

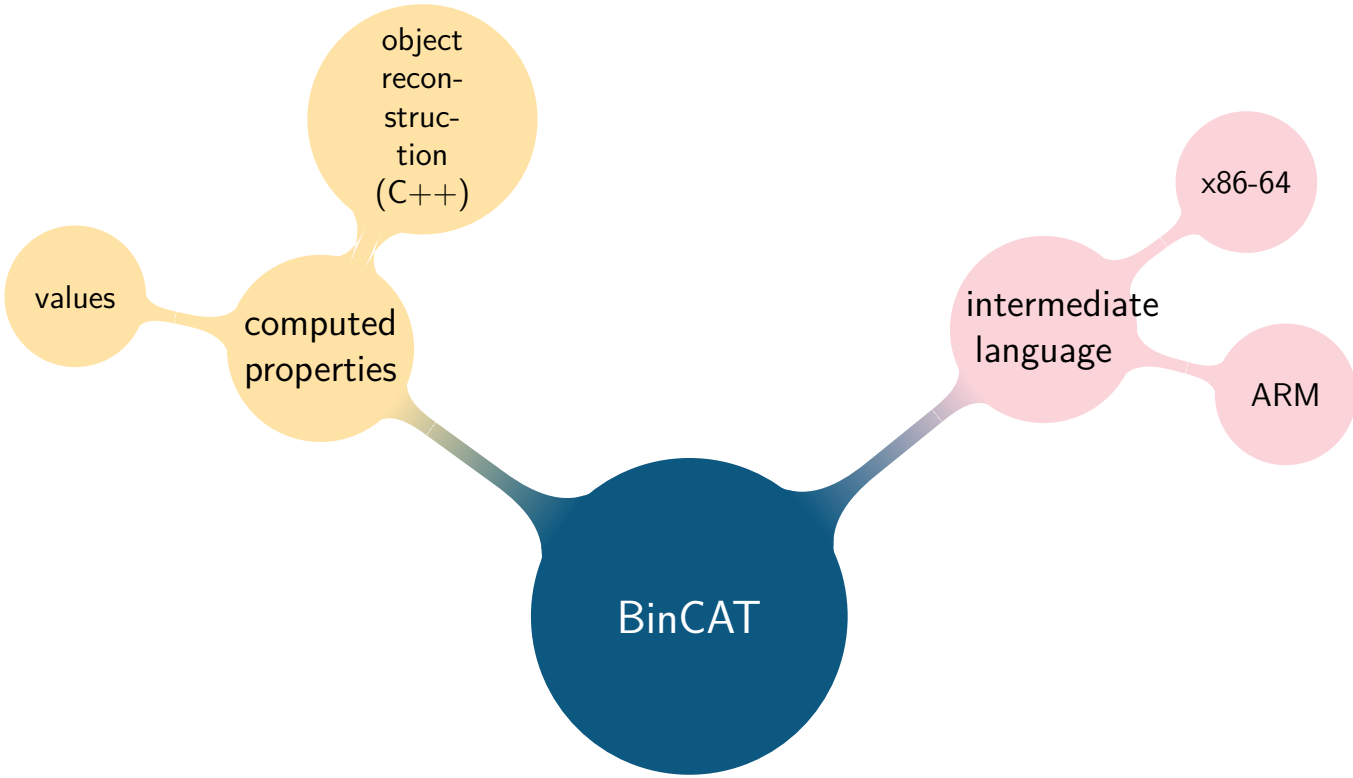

values
taint analysis
computed properties
CFG with indirect jumsp
types
forward / backward analysis
extensible
x86
intermediate language
Binary analyzer
static
IDA integration
theory backed

Current features improvements (planned)



- better type reconstruction
  - new types from heuristics. Ex: structures detection on stack
- several distinct taint sources
- more precise computations in backward analysis
- more standard library functions models
- type and value override in IDA
- memory definition directly in IDA

Future features



- finer approximations in values computation by using intervals
- complex objects reconstruction (C++)
- x86-64 and ARM decoders

Thanks!

Full paper (link in README):

`https://www.sstic.org/media/SSTIC2017/SSTIC-actes/bincat_purrfecting_binary_static_analysis/SSTIC2017-Article-bincat_purrfecting_binary_static_analysis-biondi_rigo_zennou_mehrenberger.pdf`

- project was partially financed by DGA-MI

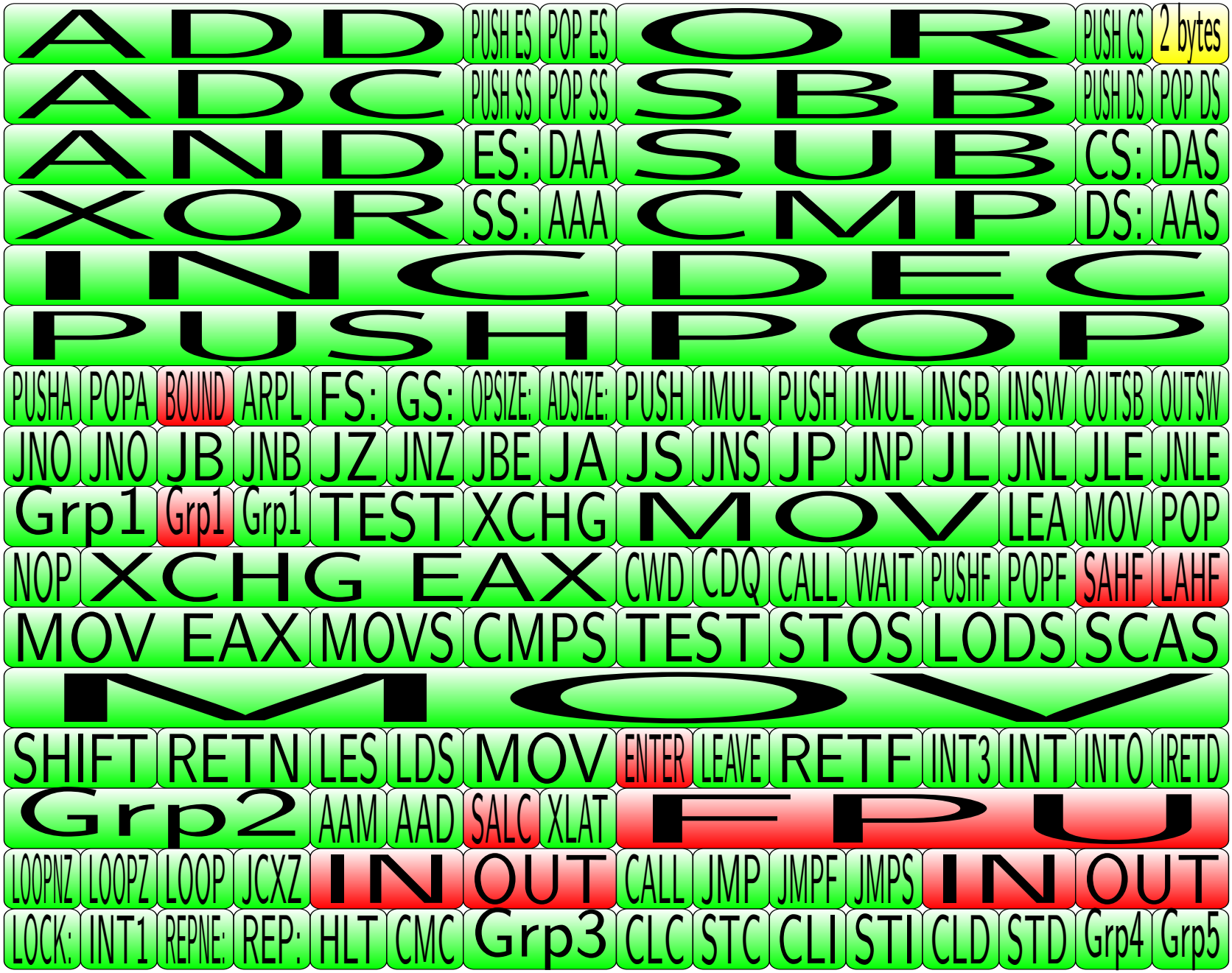
- Get it! (AGPL licence)

`https://github.com/airbus-seclab/bincat`

```
docker run -p 5000:5000 airbusseclab/bincat
```

tutorial in `doc/tutorial.md`

x86 coverage



x86 coverage - Second table

Currently implemented lattices