Micro-Policies

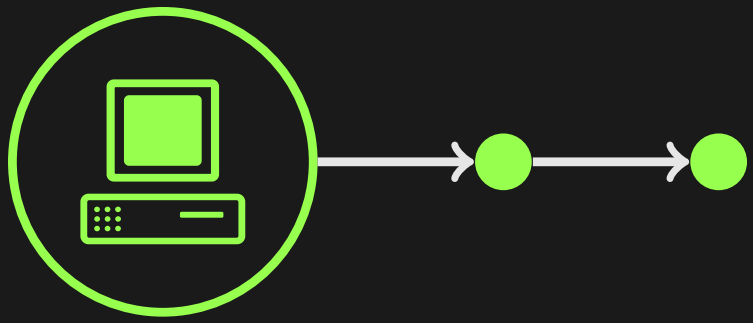
Formally Verified, Tag-Based Security Monitors

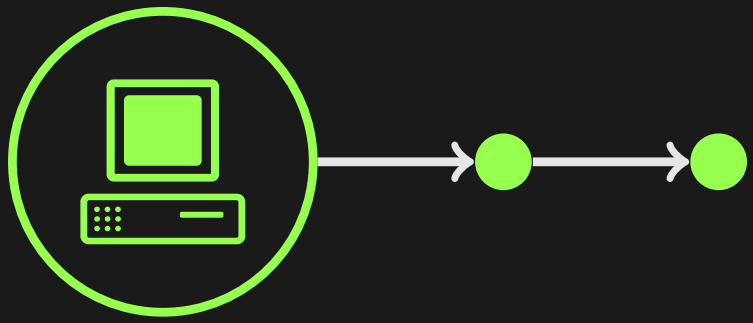
Arthur Azevedo de Amorim Maxime Dénès
Nick Giannarakis Cătălin Hriţcu Benjamin C. Pierce
Antal Spector-Zabusky Andrew Tolmach

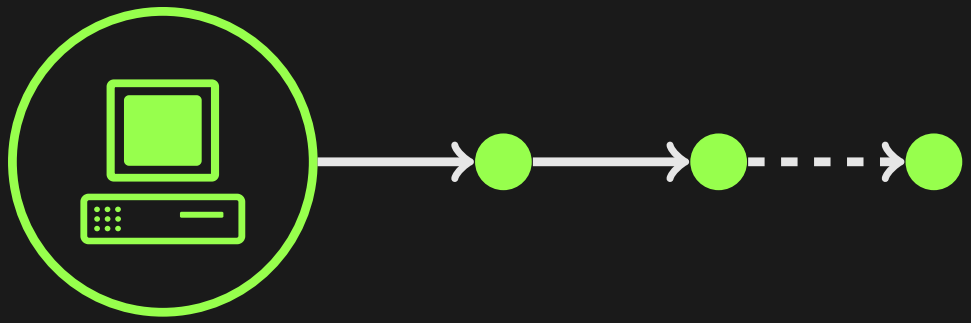
May 20, 2015

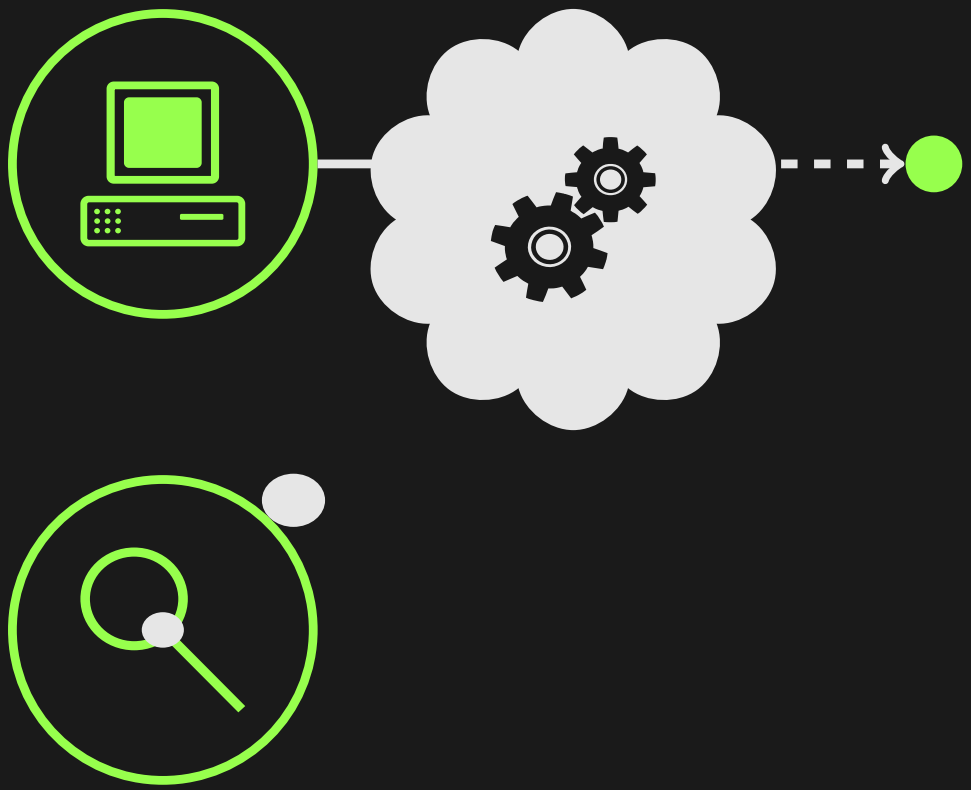
How can we design secure systems?

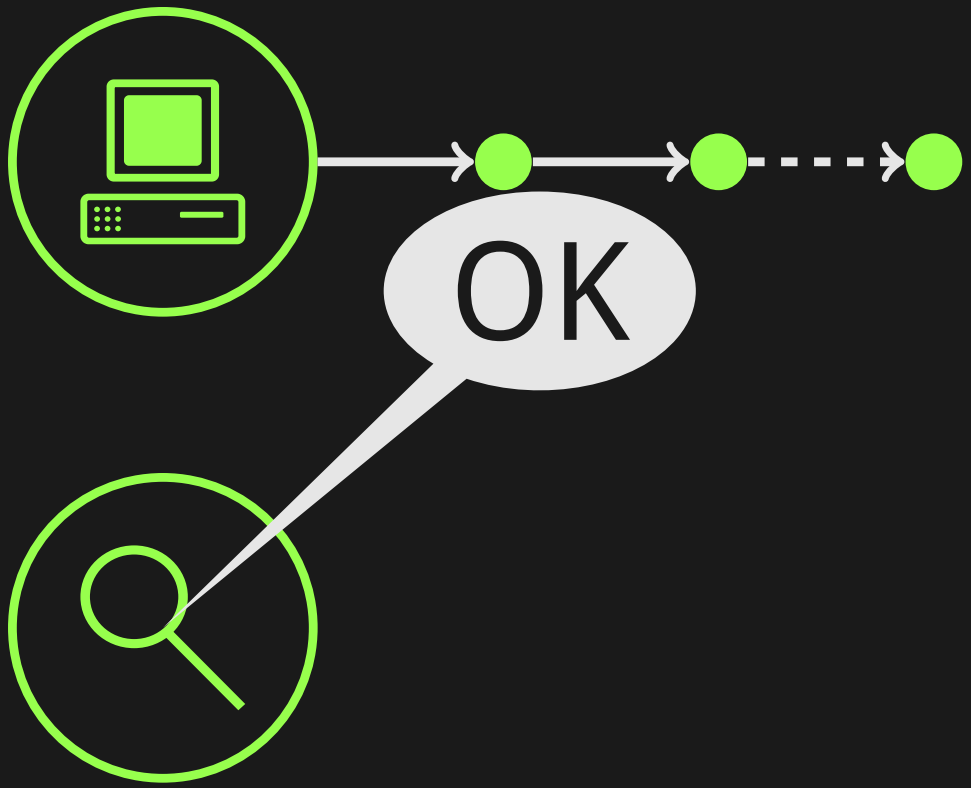
One approach: reference monitors

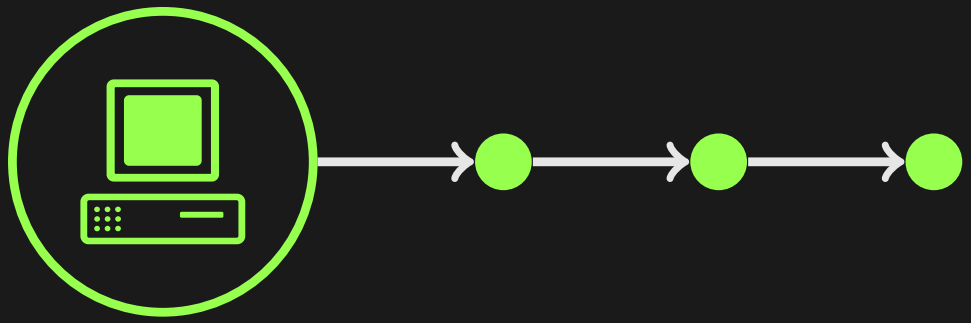


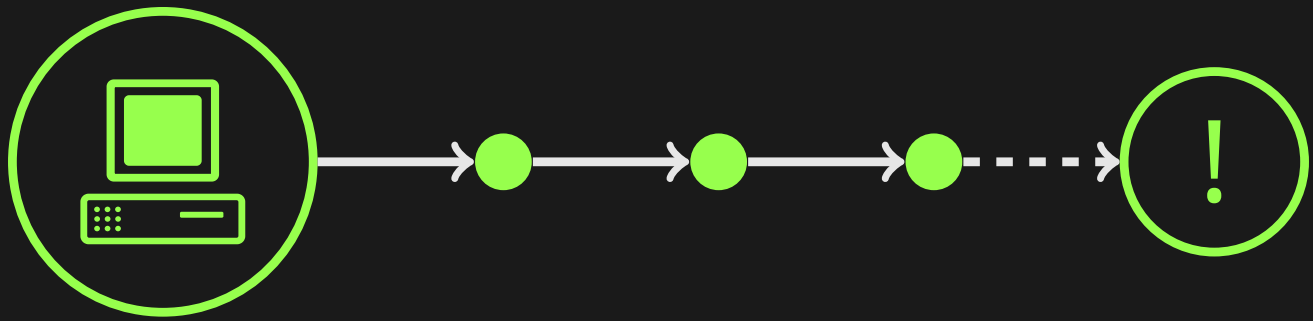


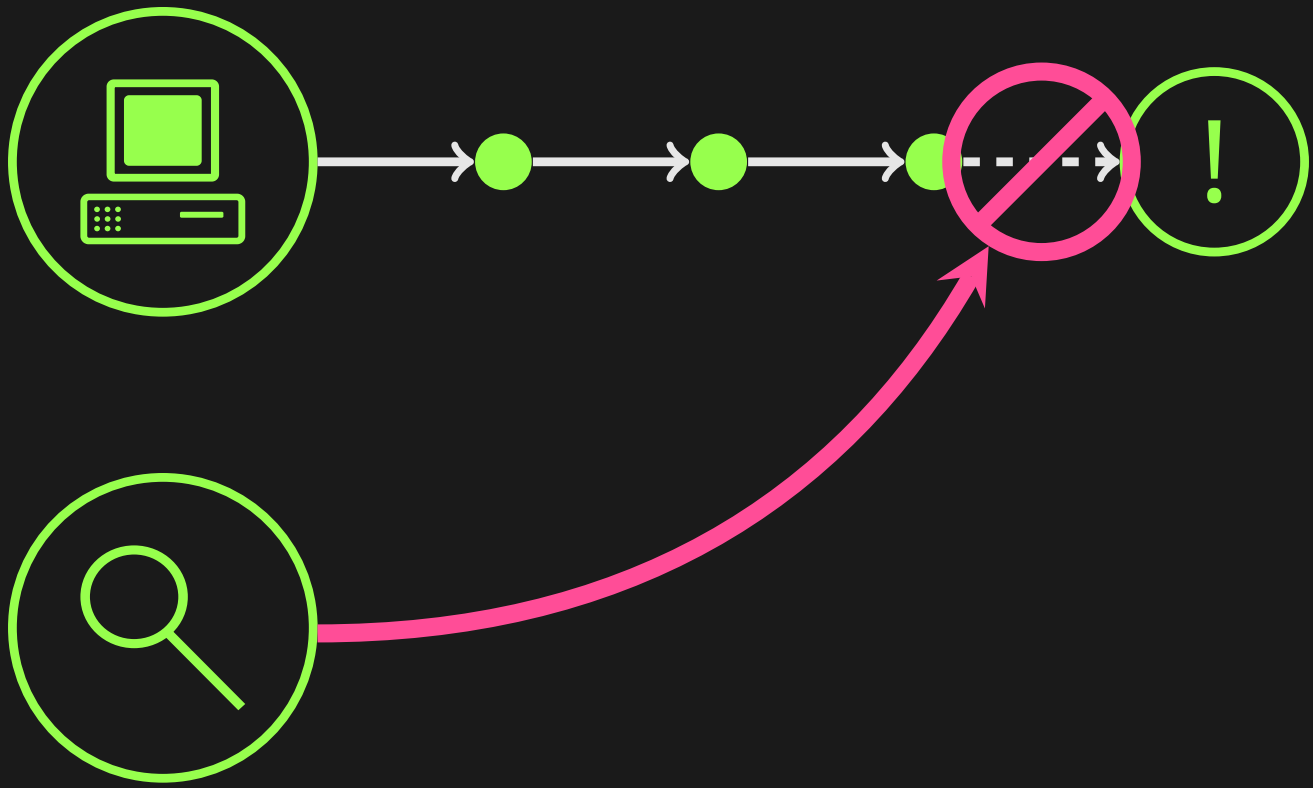




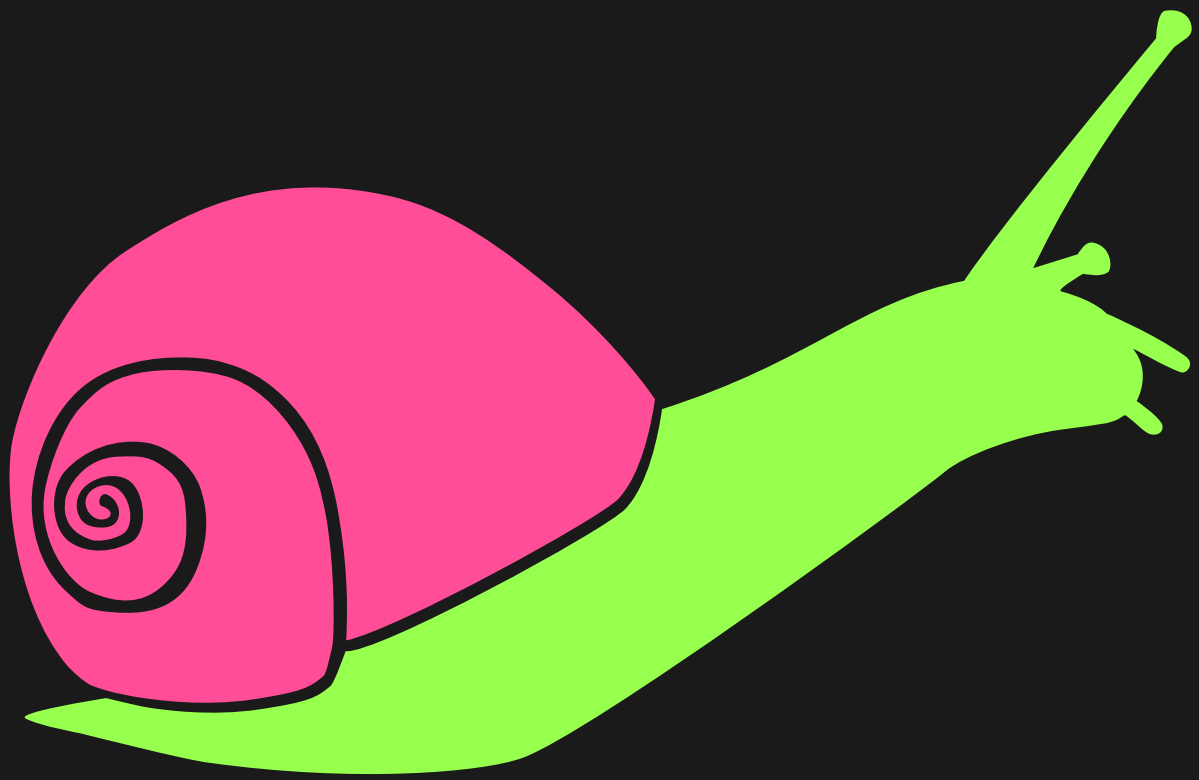

OK





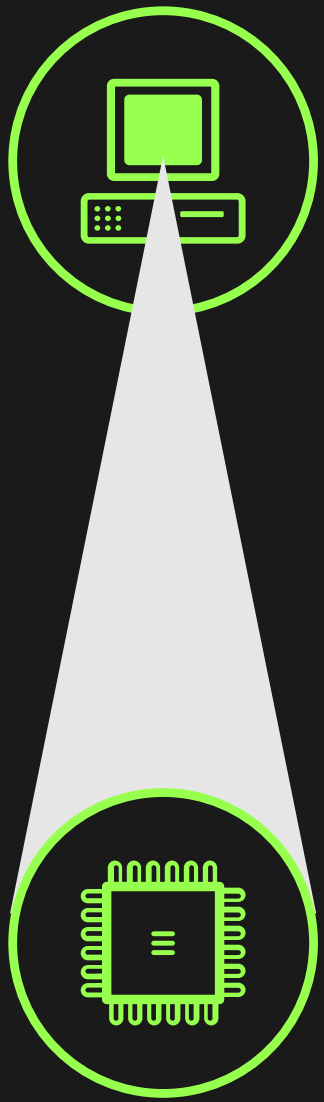


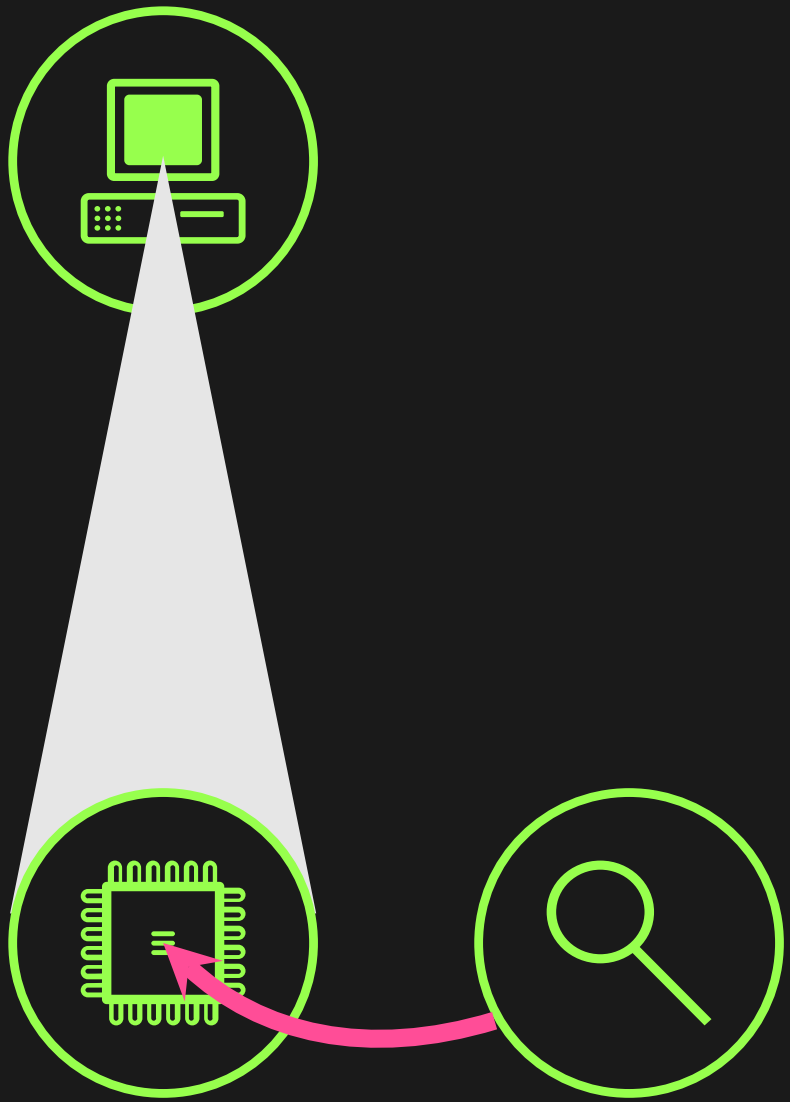
But there is a problem...

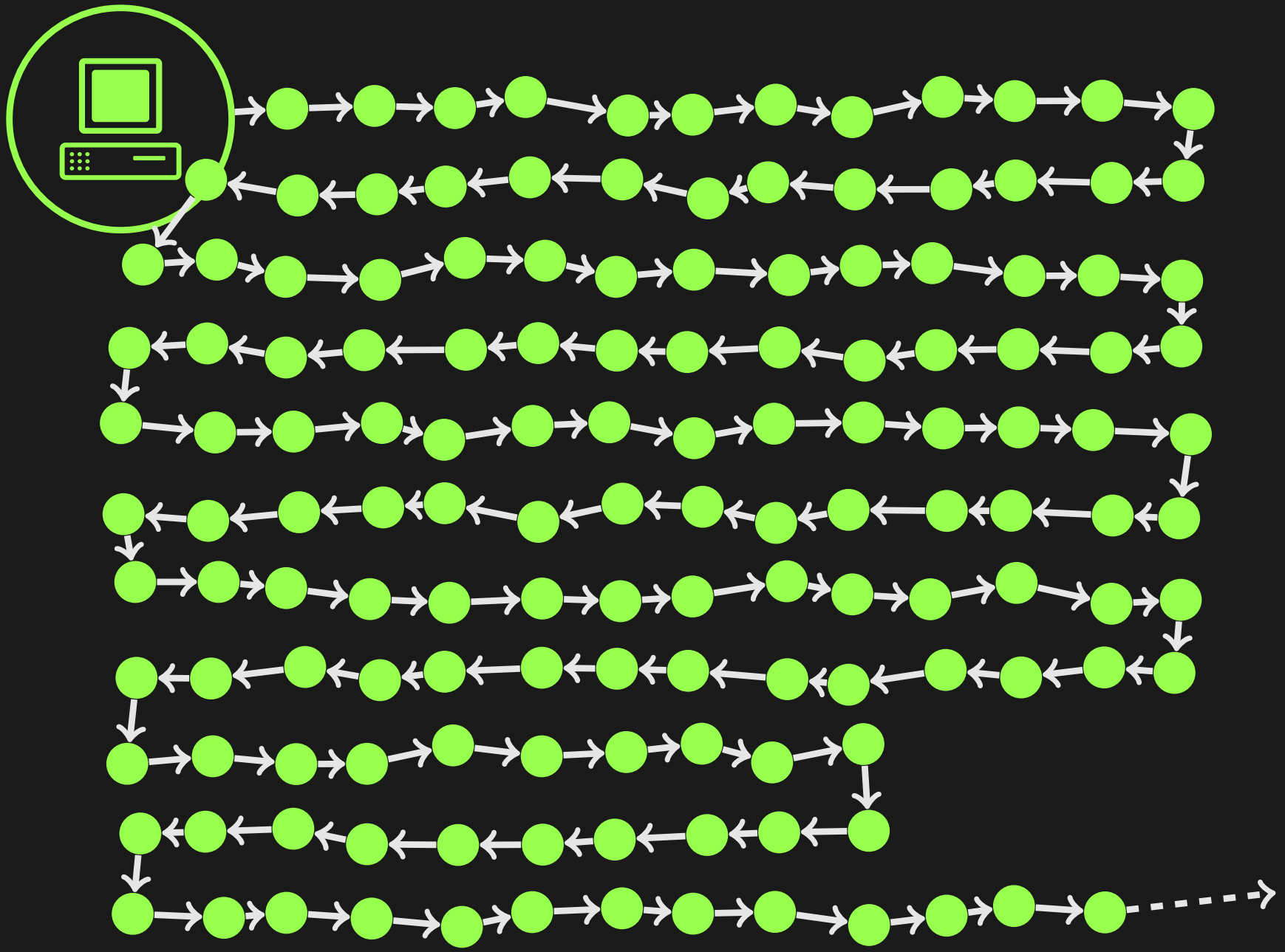


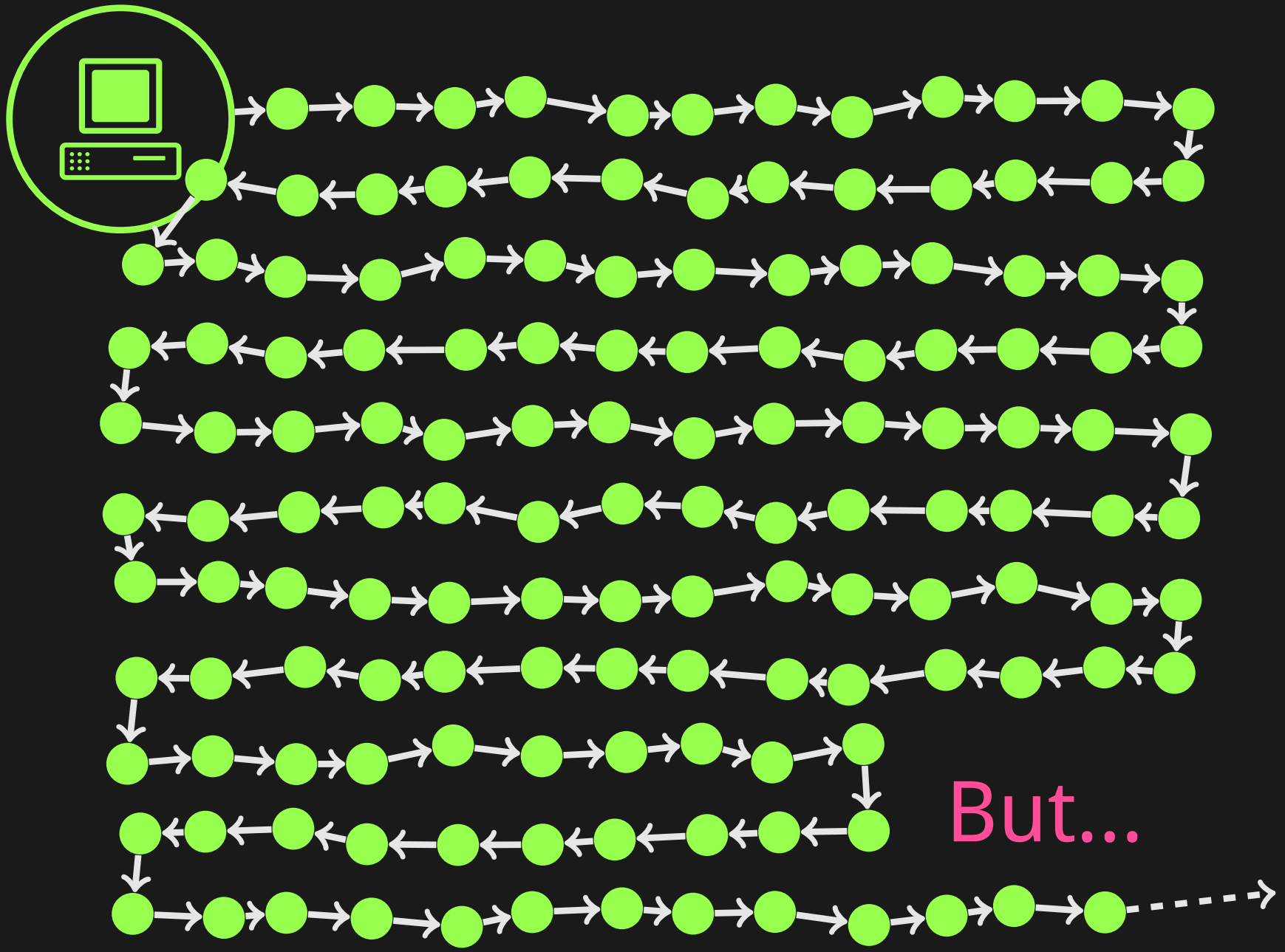
...they are slow

Idea: hardware support for reference monitors





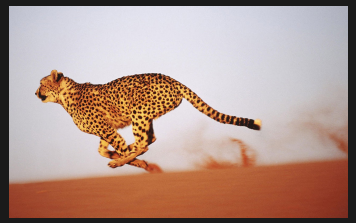



But...

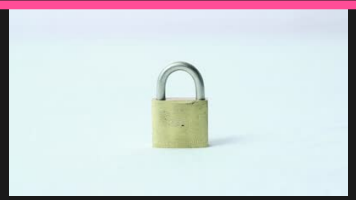
What if a new threat appears?

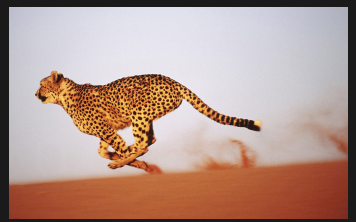
Micro-Policies

Micro-policy
programming model

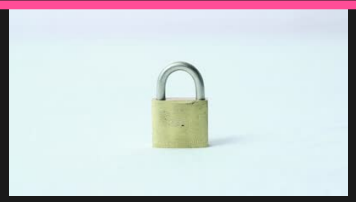


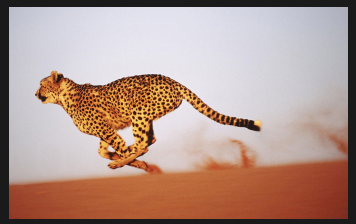
Micro-policy
programming model





Micro-policy
programming model





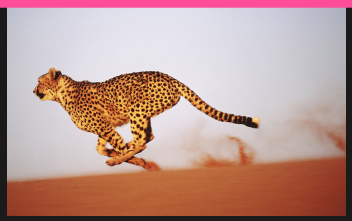
CFI

Compartments

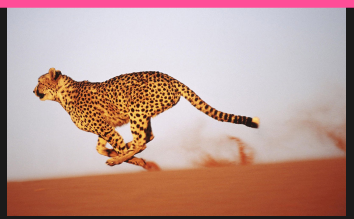
Micro-policy programming model

Sealing

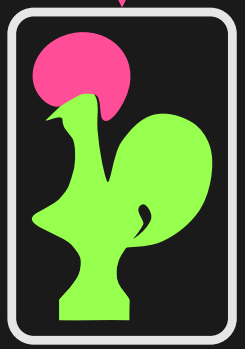
Memory safety

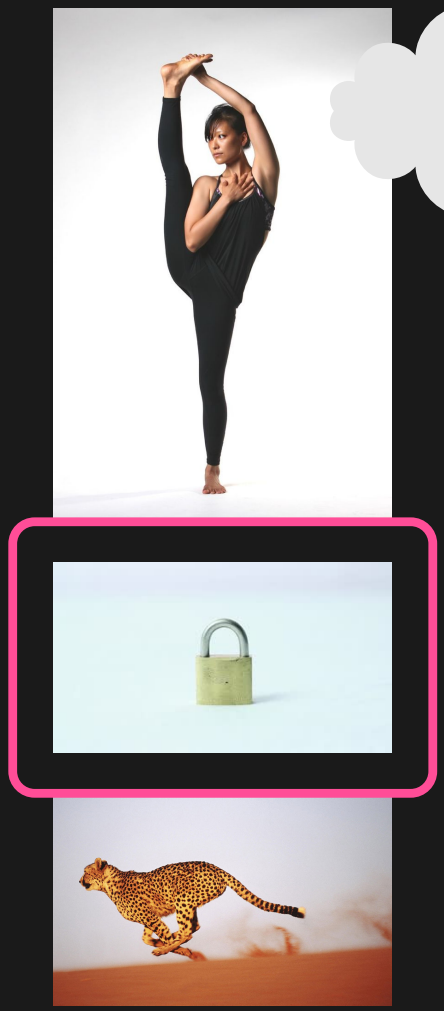


Micro-policy
programming model



Micro-policy
programming model

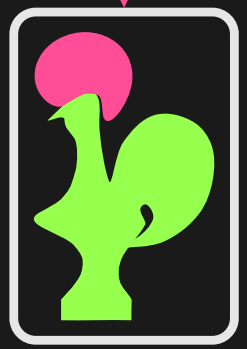


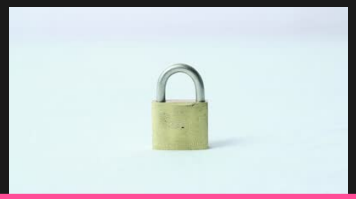


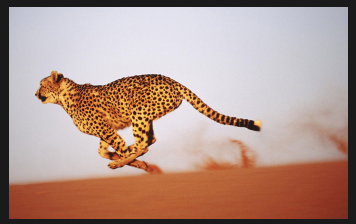
Micro-policy specification

machine-checked proof

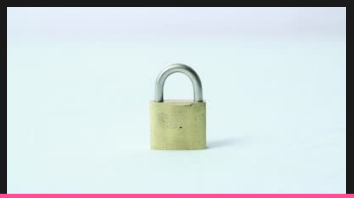
Micro-policy
programming model

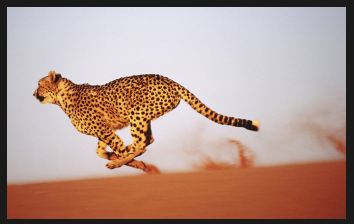






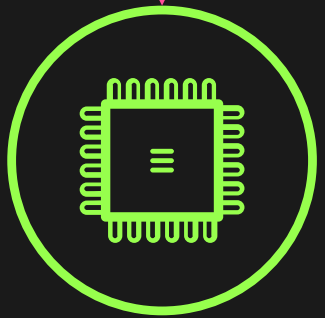
Micro-policy
programming model



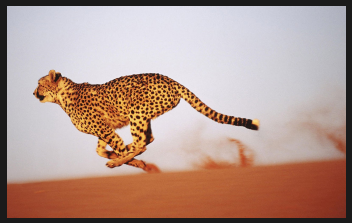


Micro-policy programming model

supported by



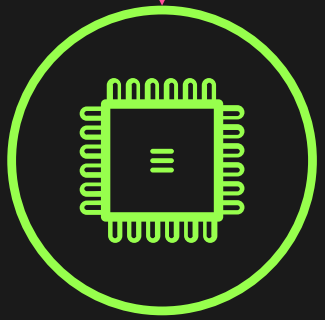
PUMP
(Programmable Unit for Metadata Processing)



typically < 10% runtime overhead
(ASPLOS '15)

Micro-policy
programming model

supported by



PUMP
(Programmable Unit for
Metadata Processing)

Programming model

General monitors

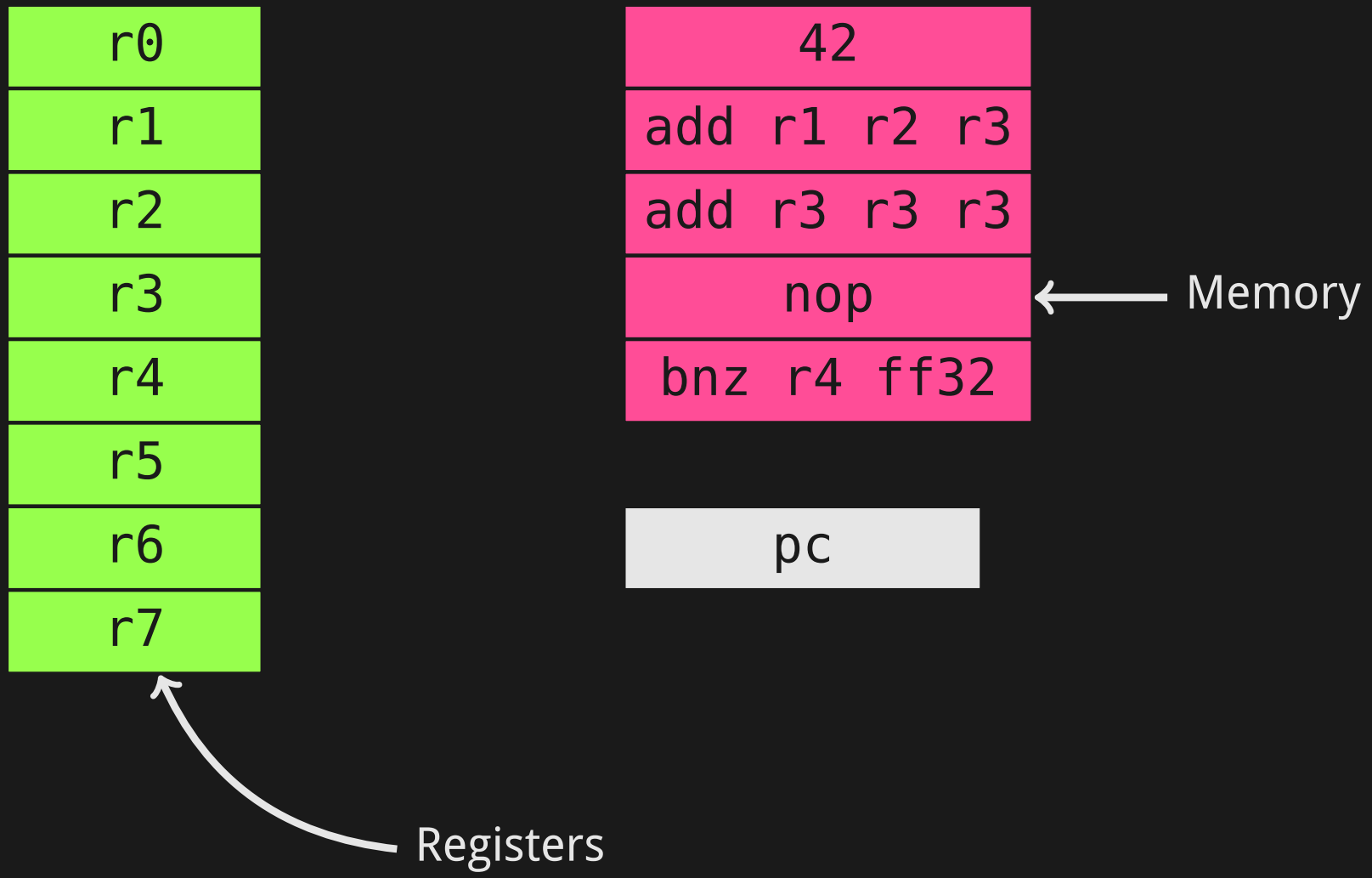
Inspect program state arbitrarily

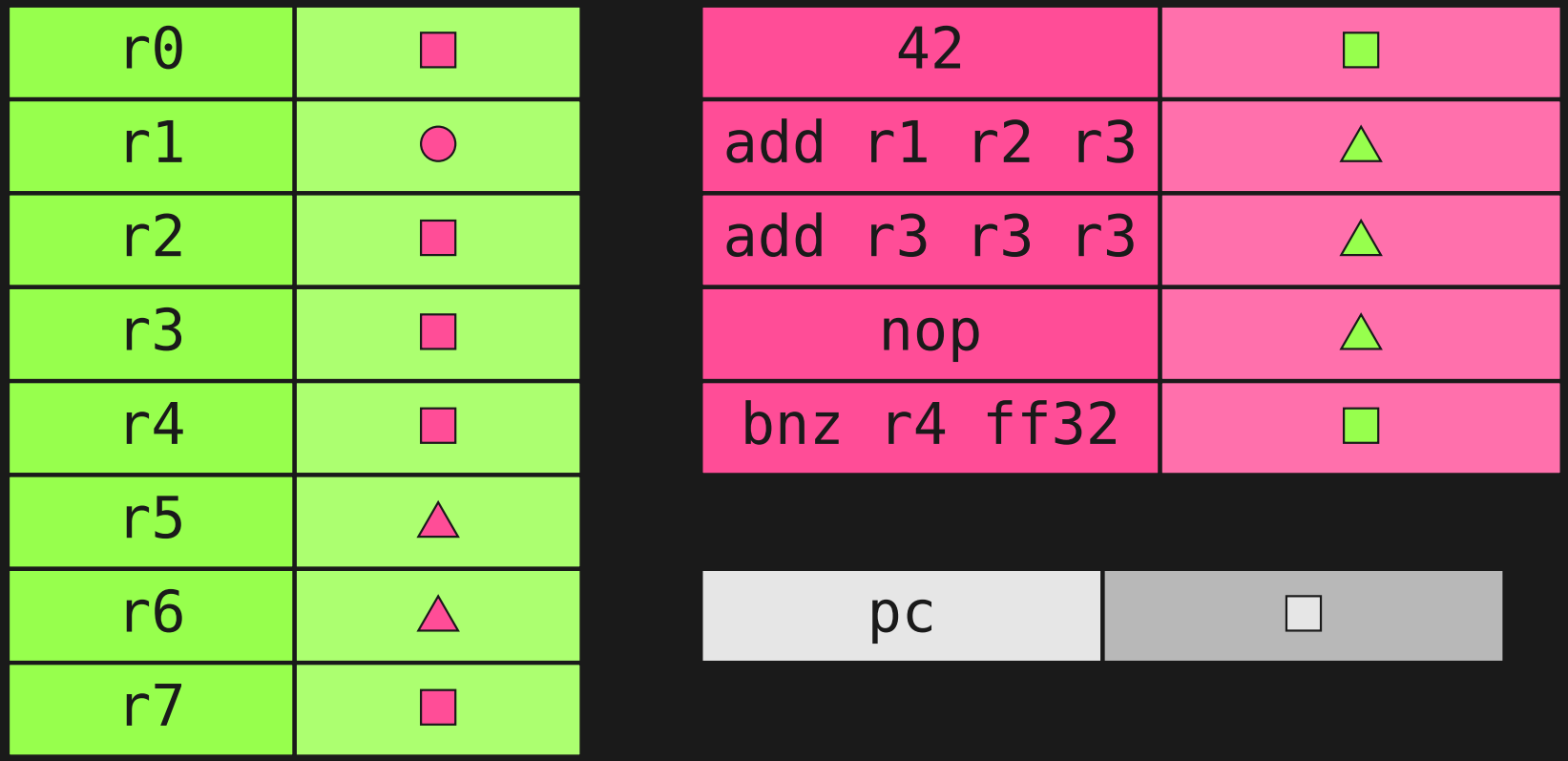
General monitors

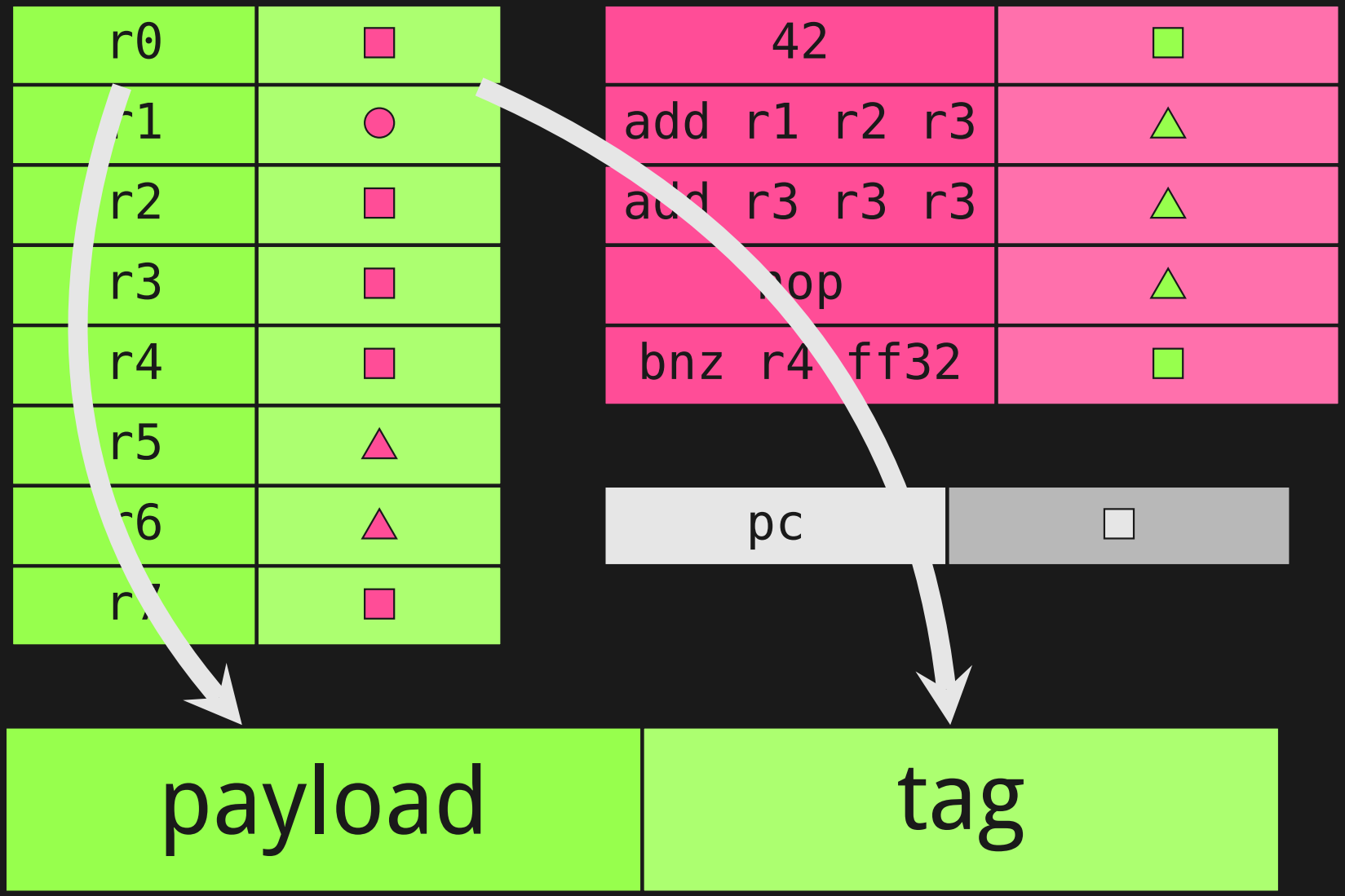
Inspect program state arbitrarily

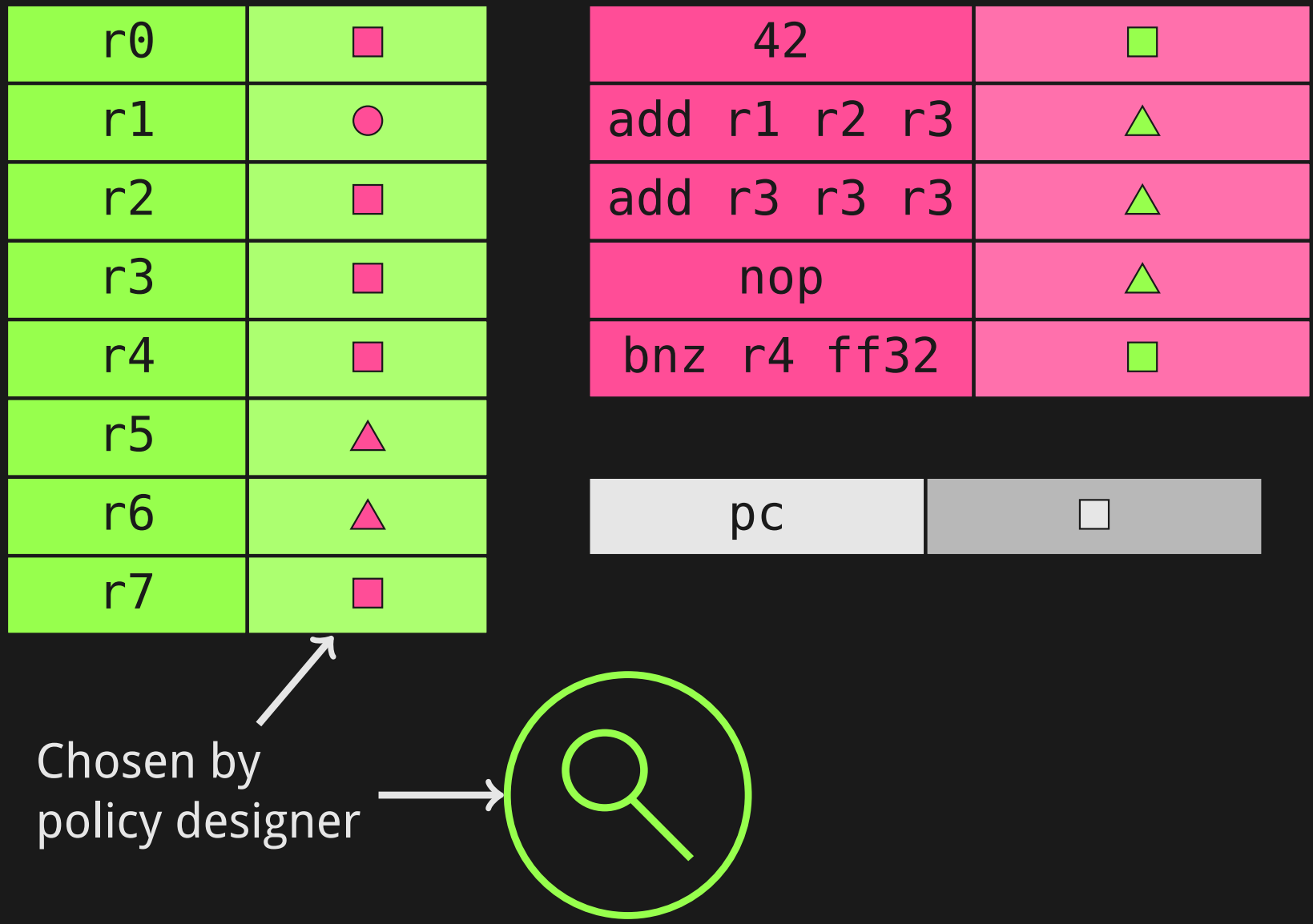
...but too powerful for efficient support

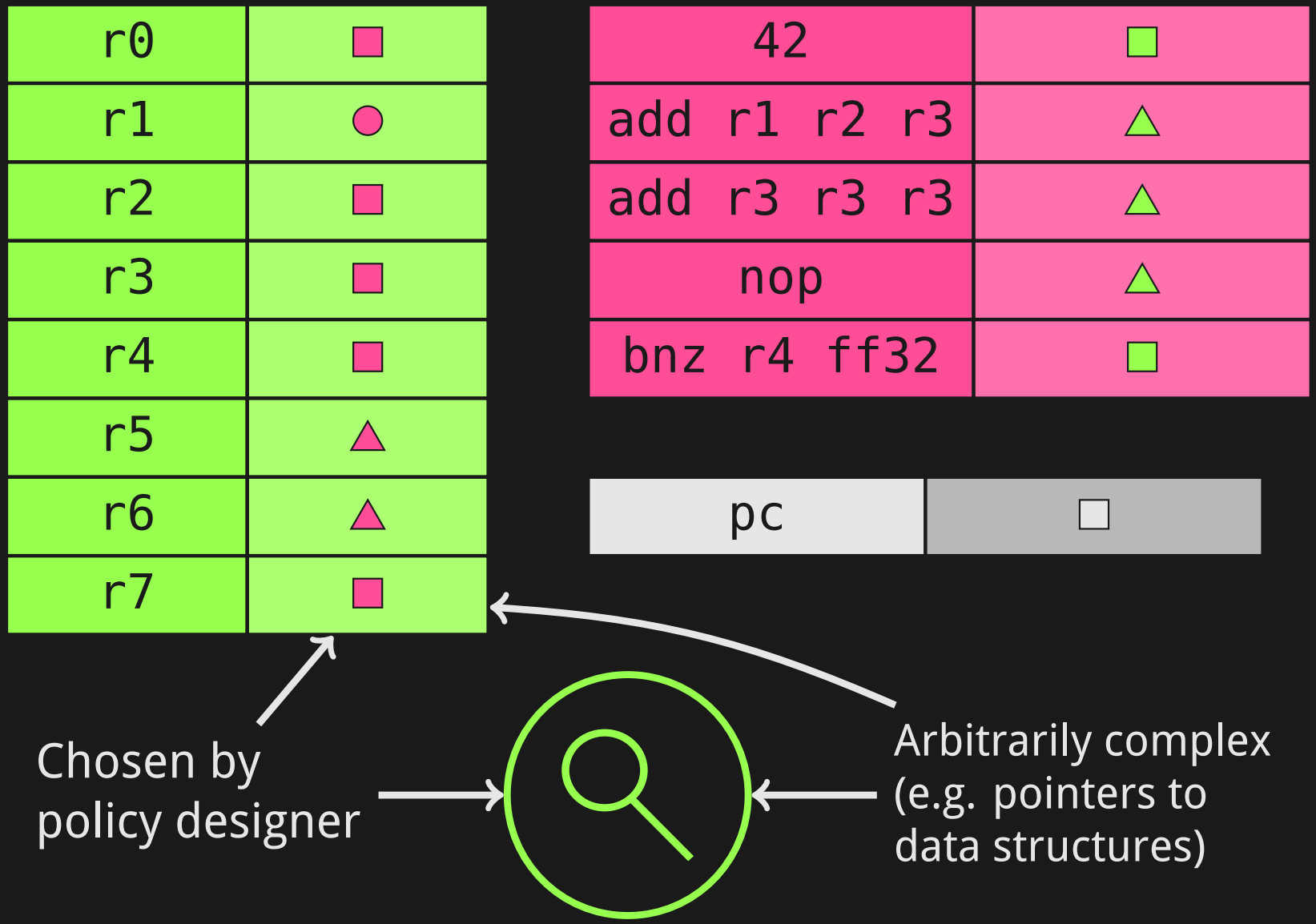
Insight: monitors as computation on metadata

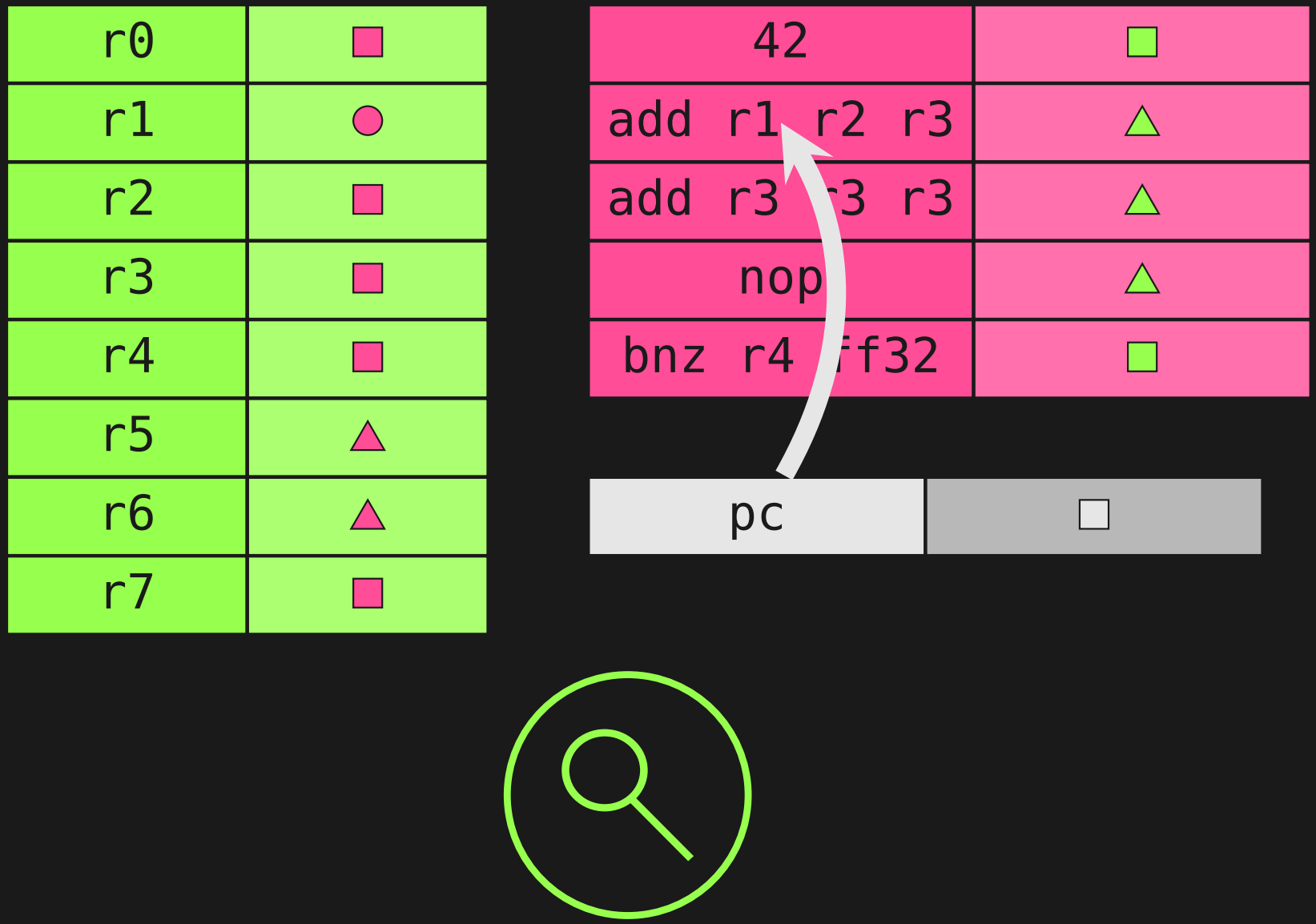

r0
r1
r2
r3
r4
r5
r6
r7
42
add r1 r2 r3
add r3 r3 r3
nop
bnz r4 ff32
Memory
pc
Registers

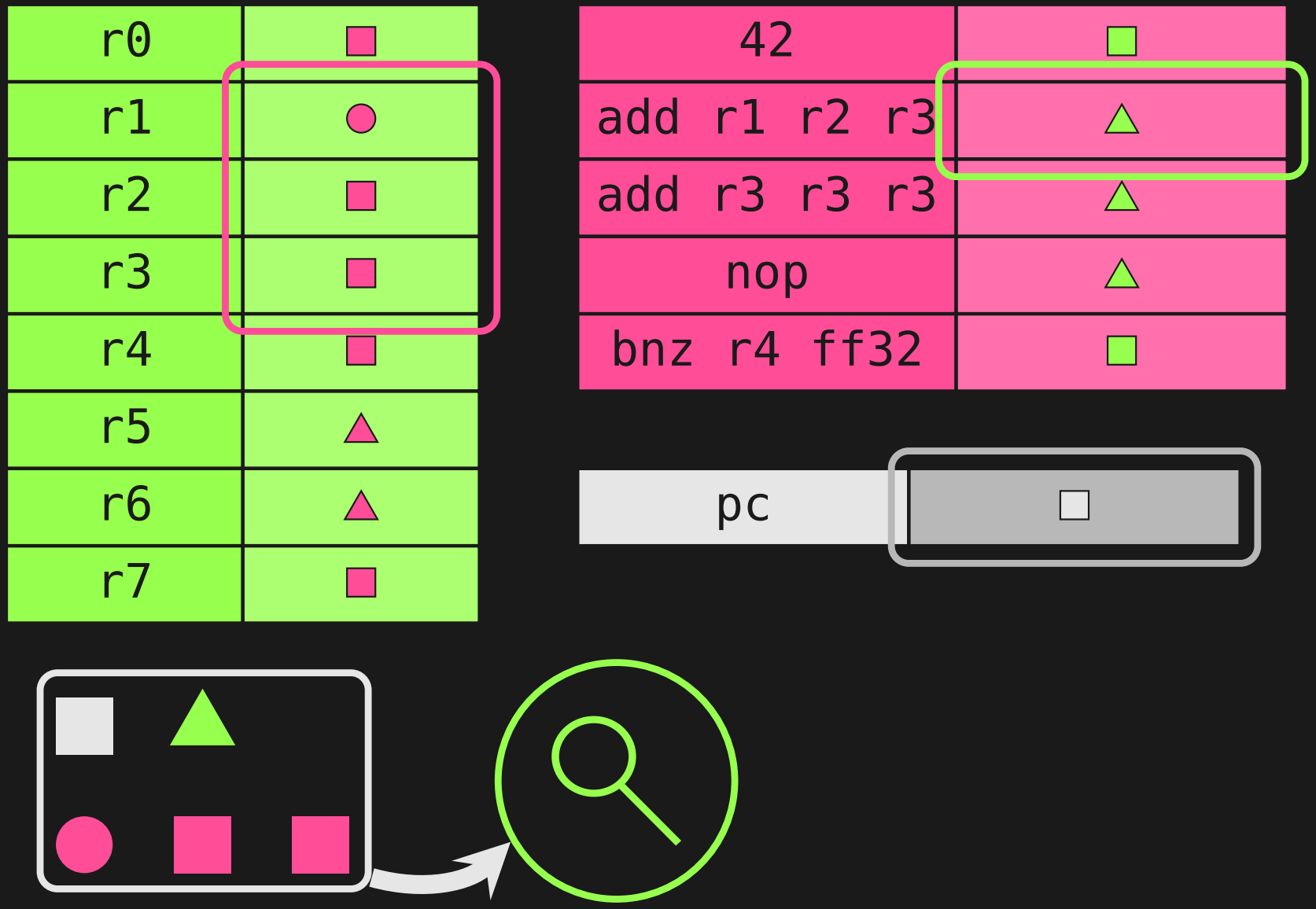

r0
r1
r2
r3
r4
r5
r6
r7
42
add r1 r2 r3
add r3 r3 r3
nop
bnz r4 ff32
pc


r0
r1
r2
r3
r4
r5
r6
42
add r1 r2 r3
add r3 r3 r3
nop
bnz r4 ff32
pc
payload
tag


r0
r1
r2
r3
r4
r5
r6
r7
42
add r1 r2 r3
add r3 r3 r3
nop
bnz r4 ff32
pc
Chosen by
policy designer


r0
r1
r2
r3
r4
r5
r6
r7
42
add r1 r2 r3
add r3 r3 r3
nop
bnz r4 ff32
pc
Chosen by
policy designer
Arbitrarily complex
(e.g. pointers to
data structures)

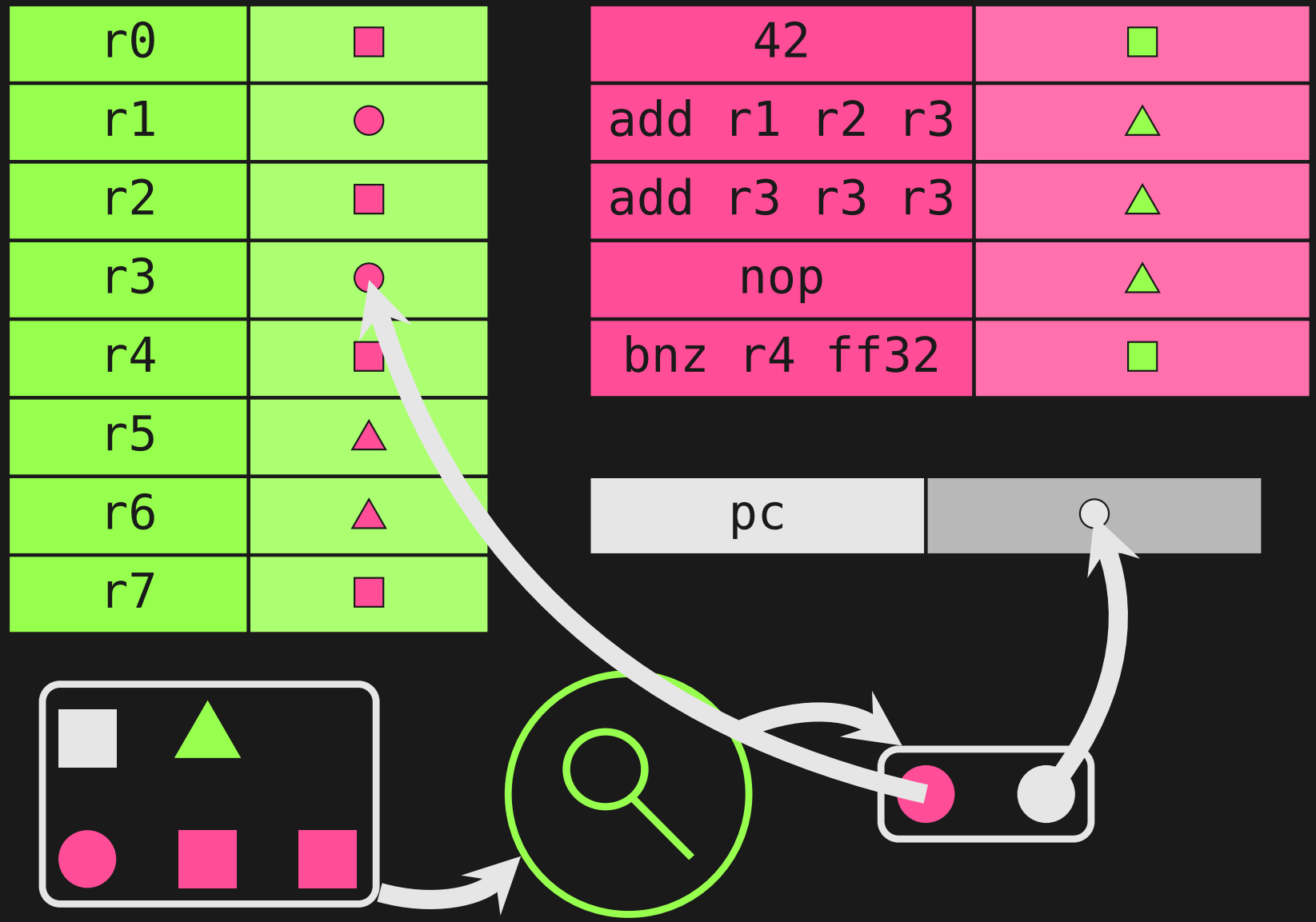

r0
r1
r2
r3
r4
r5
r6
r7
42
add r1 r2 r3
add r3 r3 r3
nop
bnz r4 ff32
pc

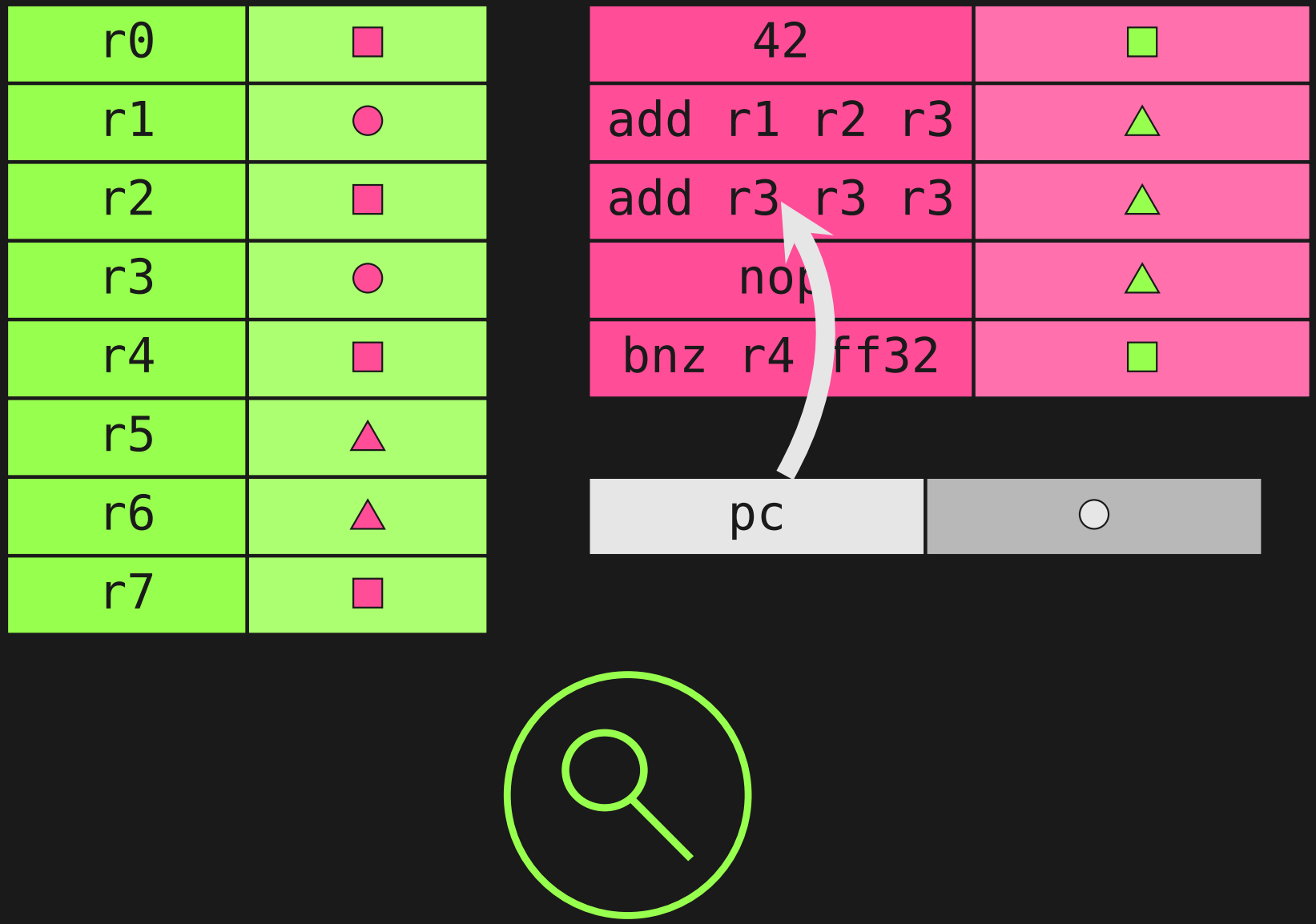

r0
r1
r2
r3
r4
r5
r6
r7
42
add r1 r2 r3
add r3 r3 r3
nop
bnz r4 ff32
pc

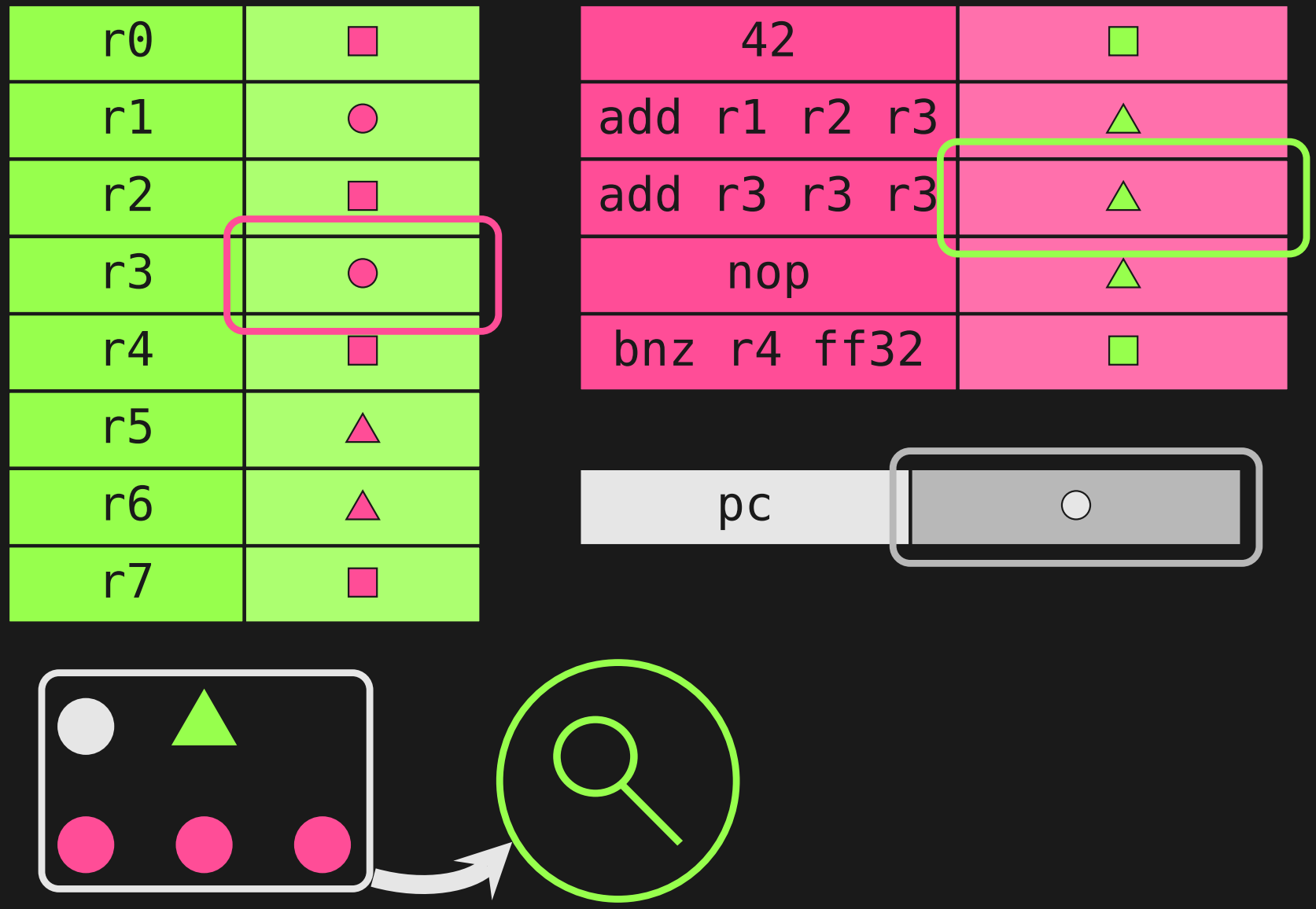
r0	■
r1	●
r2	■
r3	●
r4	■
r5	▲
r6	▲
r7	■

42	■
add r1 r2 r3	▲
add r3 r3 r3	▲
nop	▲
bnz r4 ff32	■

pc	○


r0
r1
r2
r3
r4
r5
r6
r7
42
add r1 r2 r3
add r3 r3 r3
nop
bnz r4 ff32
pc

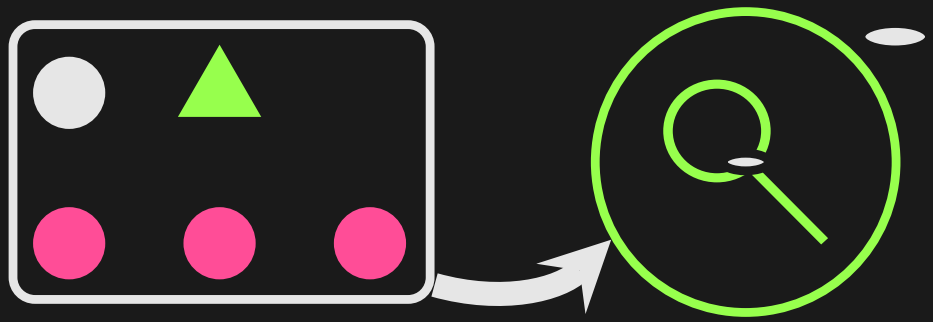

r0
r1
r2
r3
r4
r5
r6
r7
42
add r1 r2 r3
add r3 r3 r3
nop
bnz r4 ff32
pc

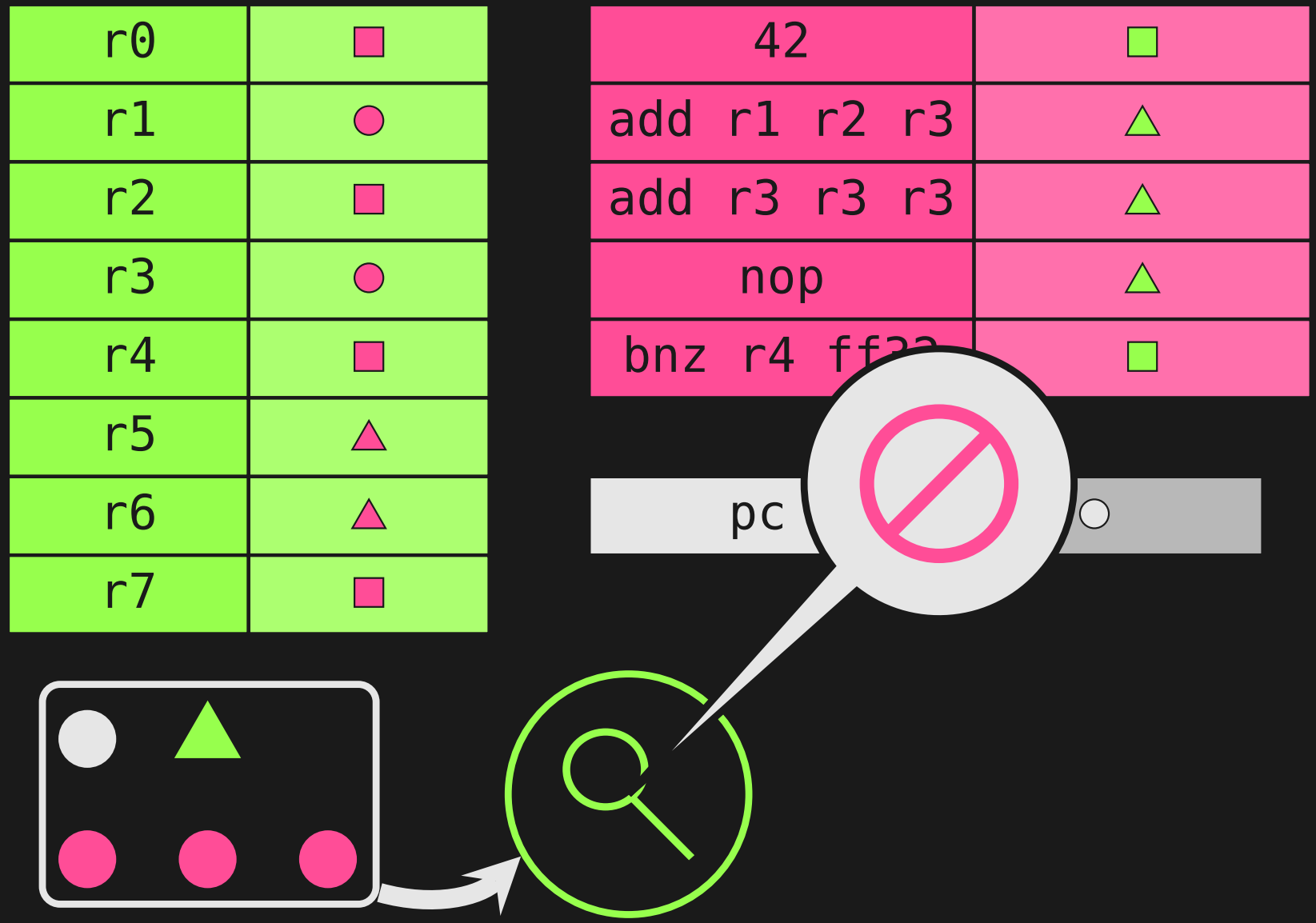

r0
r1
r2
r3
r4
r5
r6
r7
42
add r1 r2 r3
add r3 r3 r3
nop
bnz r4 ff32
pc

r0	■
r1	●
r2	■
r3	●
r4	■
r5	▲
r6	▲
r7	■

42	■
add r1 r2 r3	▲
add r3 r3 r3	▲
nop	▲
bnz r4 ff32	■

pc ???




r0
r1
r2
r3
r4
r5
r6
r7
42
add r1 r2 r3
add r3 r3 r3
nop
pc

Is it flexible?

Control-flow integrity

Compartmentalization

(à la Wahbe et al.'s SFI)

Heap memory safety

Dynamic sealing

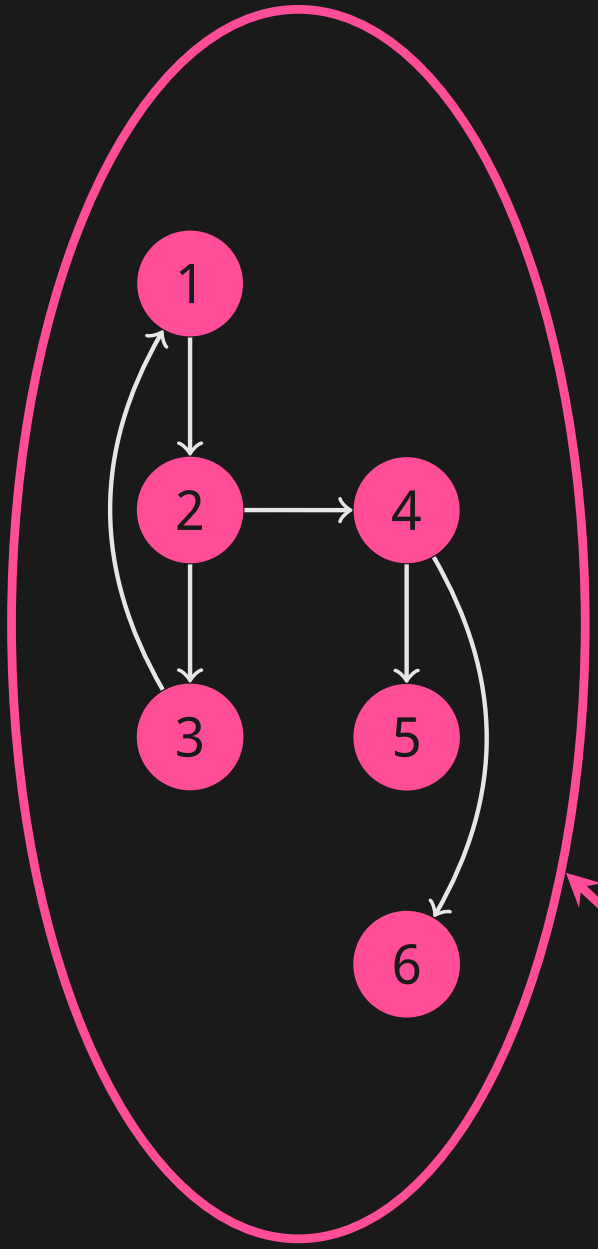
Example: CFI

1729	Data
add r1 r2 r3	Code
bnz r3 8	Code
jump r4	Code
bnz r5 8	Code
sub r1 r2 r1	Code
add r3 r4 r4	Code

1729	Data
add r1 r2 r3	Code
bnz r3 8	Code
jump r4	Code
bnz r5 8	Code
sub r1 r2 r1	Code
add r3 r4 r4	Code

$$\{\text{InstTag} = \texttt{Data}\} \rightarrow \oslash$$

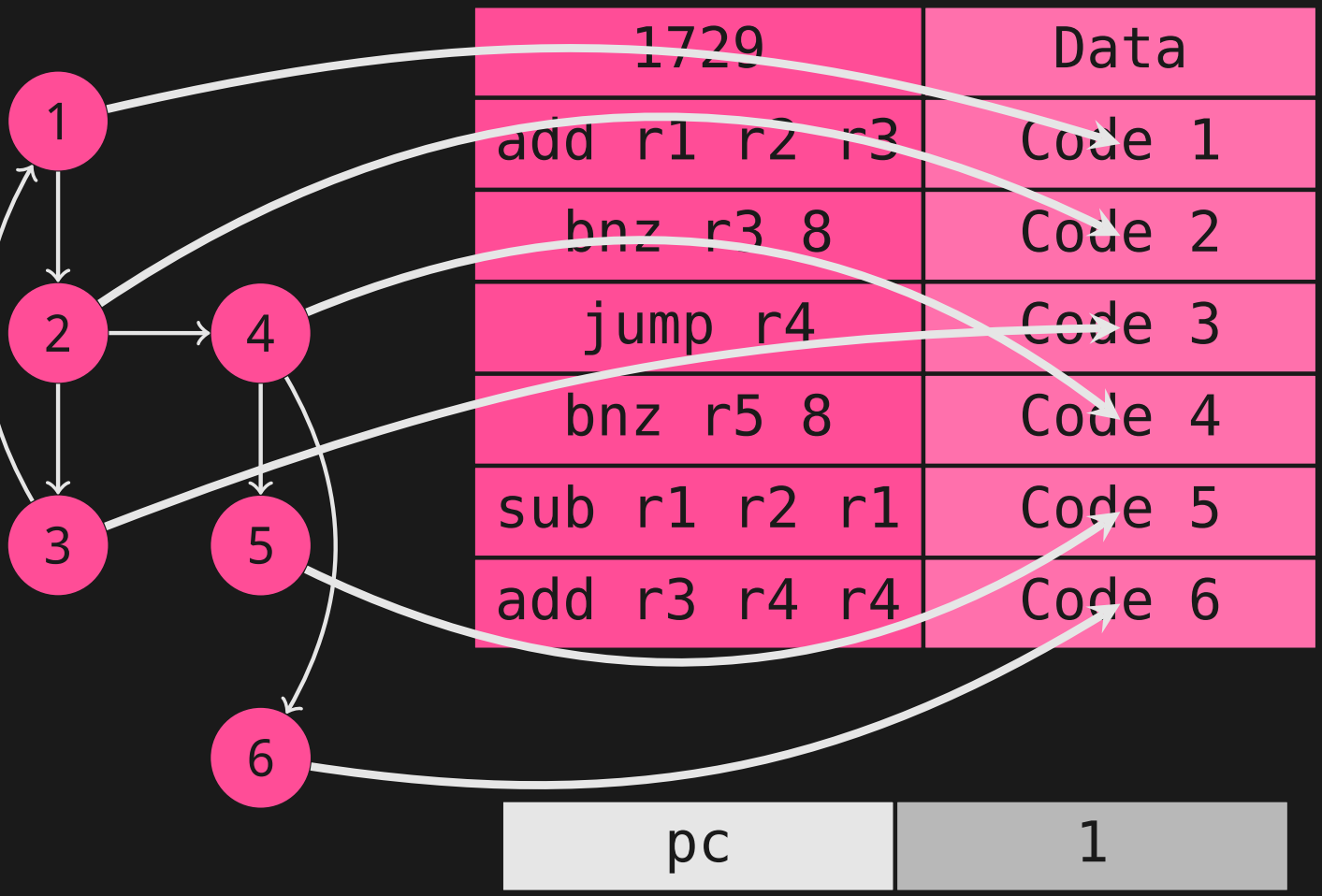
$$\{\text{Inst} = \texttt{Store}, \text{Mem} = \texttt{Code}\} \rightarrow \oslash$$

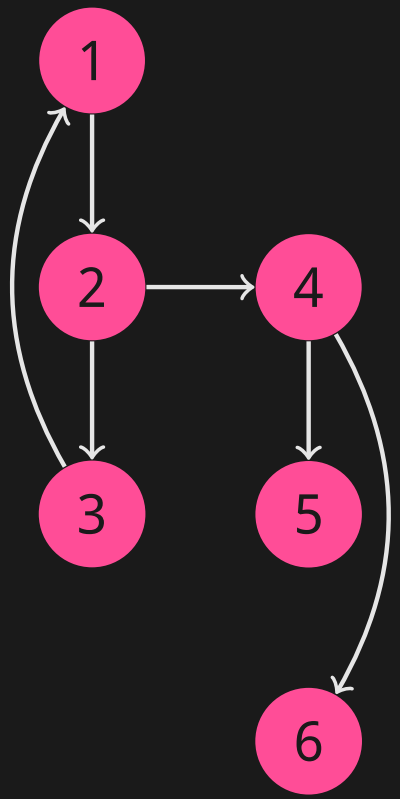


1729	Data
add r1 r2 r3	Code
bnz r3 8	Code
jump r4	Code
bnz r5 8	Code
sub r1 r2 r1	Code
add r3 r4 r4	Code

CFG

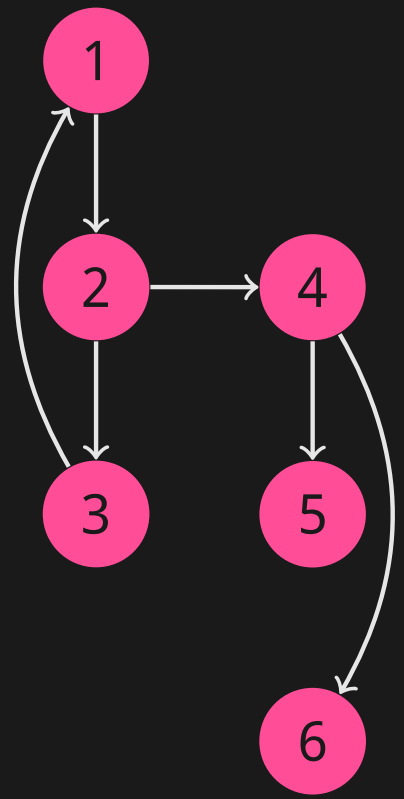
pc	1


1729
Data
add r1 r2 r3
Code 1
bnz r3 8
Code 2
jump r4
Code 3
bnz r5 8
Code 4
sub r1 r2 r1
Code 5
add r3 r4 r4
Code 6
1
2
3
4
5
6
pc
1



1729	Data
add r1 r2 r3	Code 1
bnz r3 8	Code 2
jump r4	Code 3
bnz r5 8	Code 4
sub r1 r2 r1	Code 5
add r3 r4 r4	Code 6

Previous instruction id



$\{Pc = 4, InstTag = Code\ \ 5\} \rightarrow$ OK

$\{Pc = 1, InstTag = Code\ \ 5\} \rightarrow \oslash$

pc	1

Is it secure?

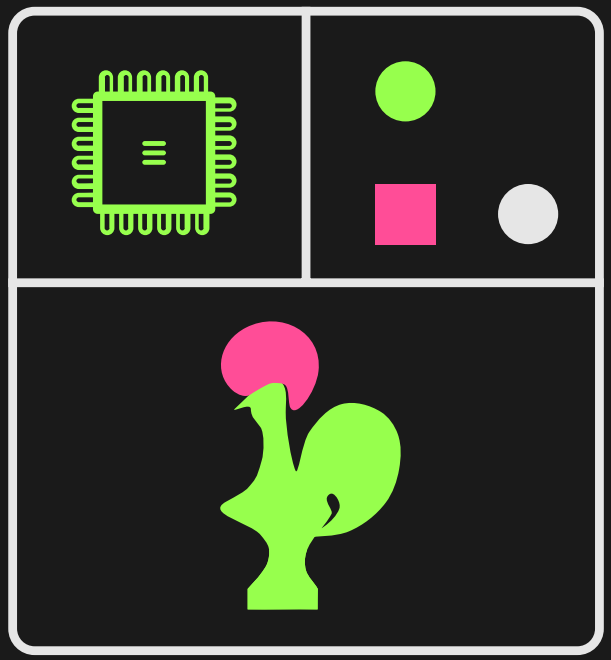
```
Inductive nat :=
| O : nat
| S : nat → nat.

Fixpoint add n m :=
  match n with
  | O ⇒ m
  | S n ⇒ S (add n m)
  end.
```

```
Lemma addn0 : ∀ n, add n O = n.
Proof. (* ... *) Qed.
```

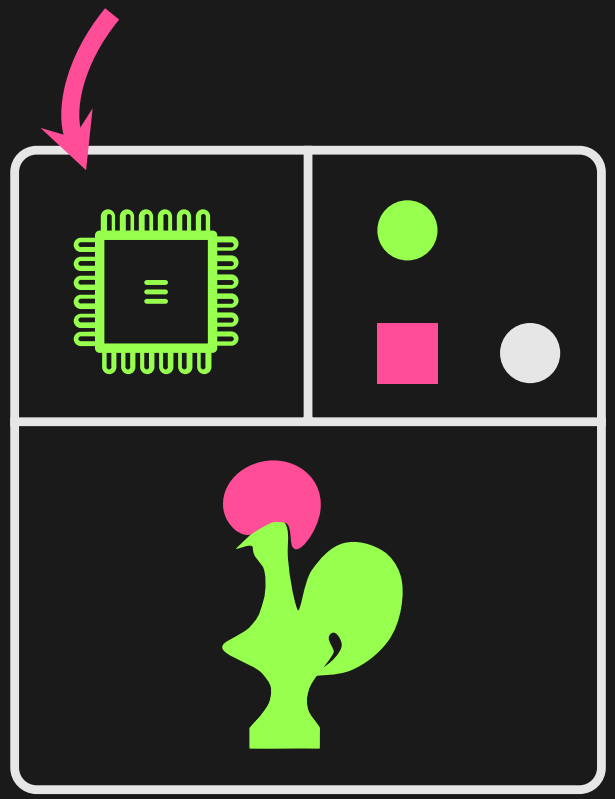
Mathematical definitions...

...and proofs about them

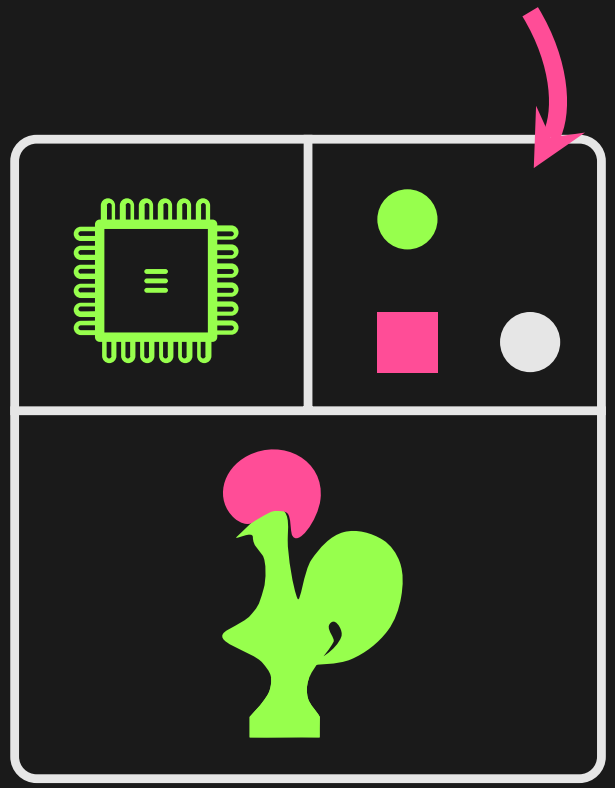
Micro-policy programming model



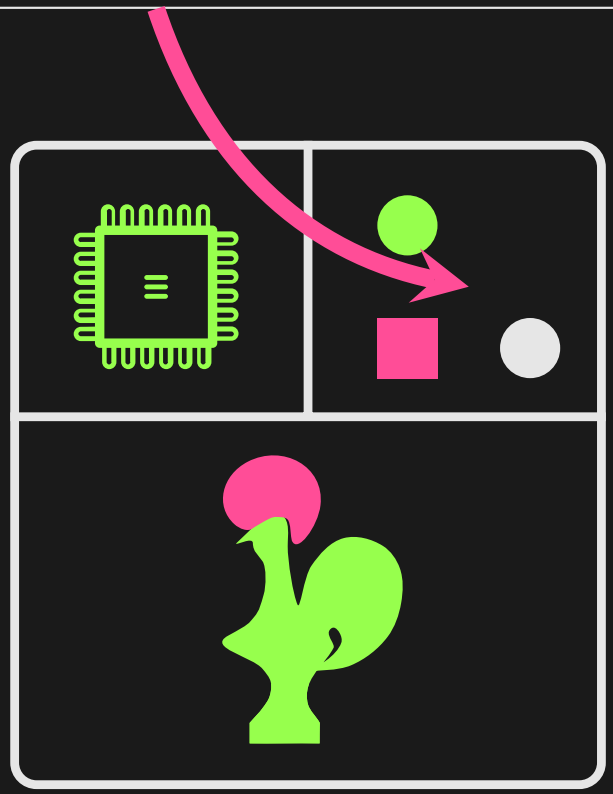
Model of simplified
RISC processor



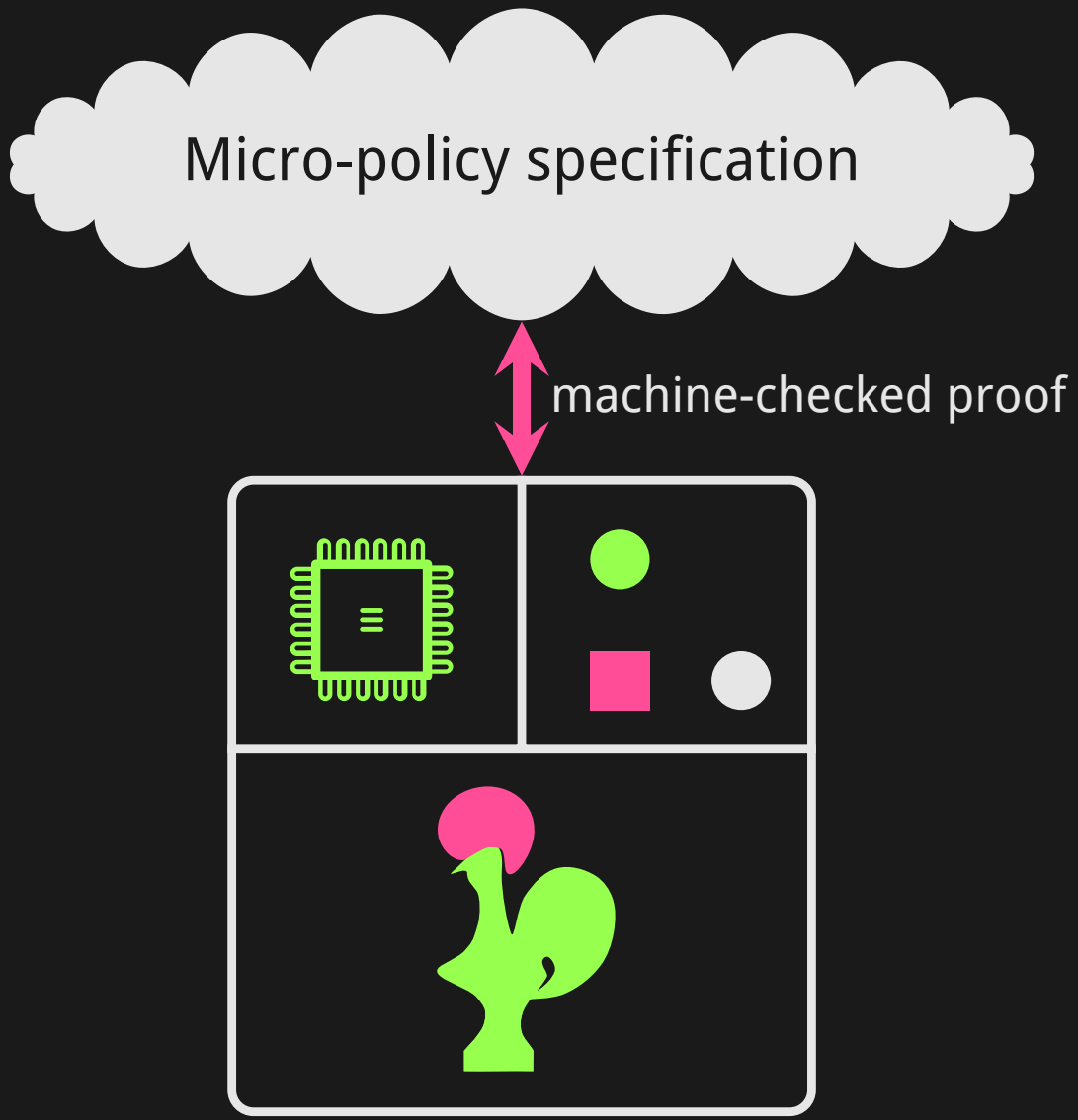
...parameterized by
user-defined tags
and policy

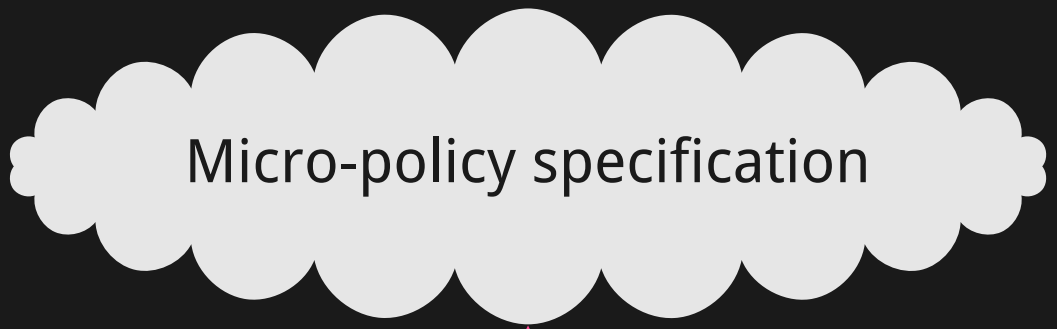


```
Inductive cfi_tag :=
| Data : cfi_tag
| Code : id → cfi_tag.

Variable cfg : id → id → bool.

Definition cfi_monitor tags :=
  match pc_tag tags, ci_tag tags with
  | n, Code m ⇒ if cfg n m then Some m
                else None
  | _, _ ⇒ None
  end.
```

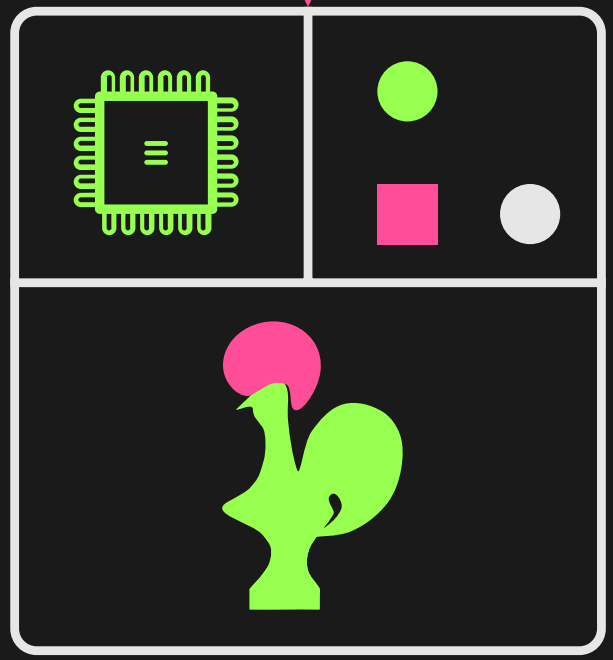


High-level description, abstract away low-level details


Micro-policy specification
machine-checked proof



machine-checked proof

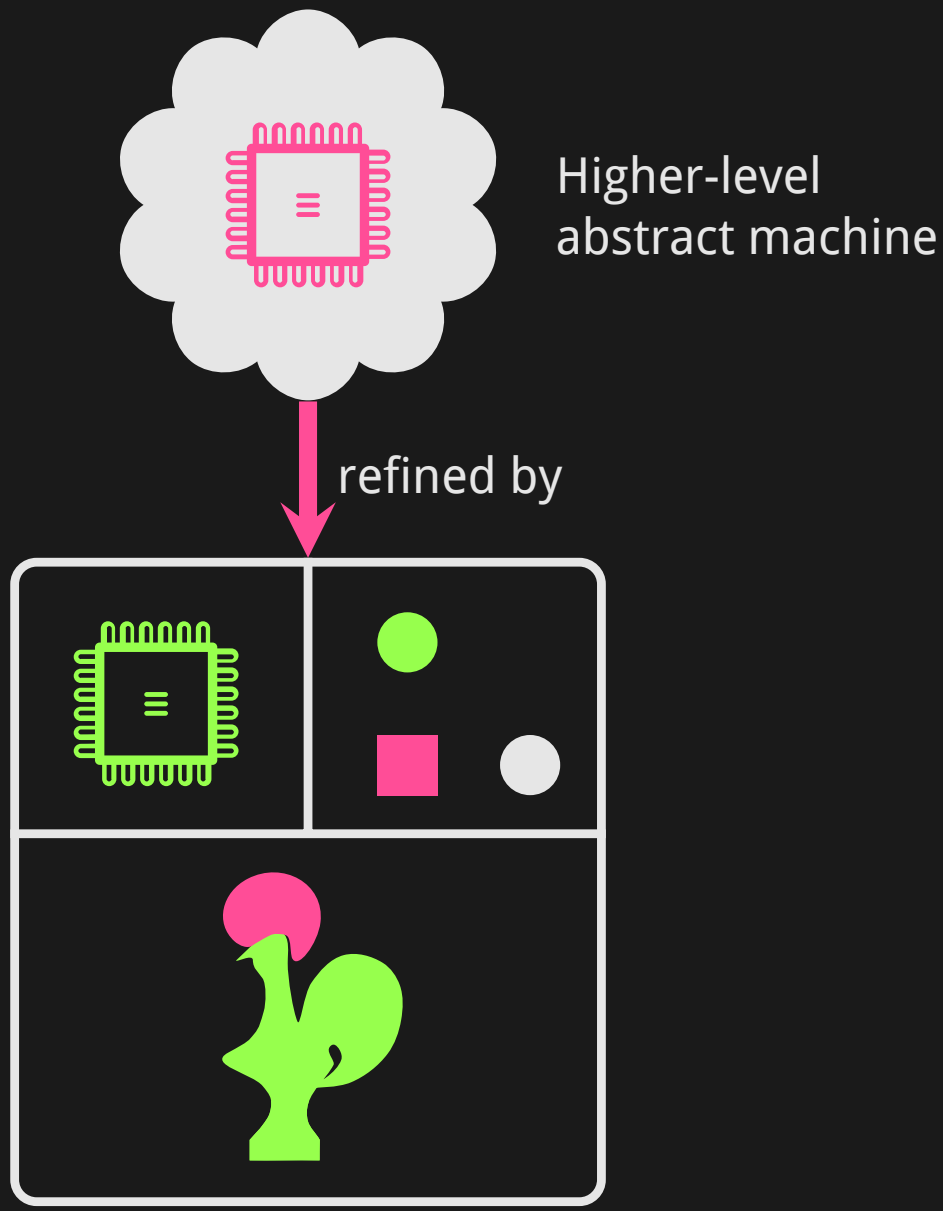


Threat model

Attacker controls input, but has no physical access

Not modeled

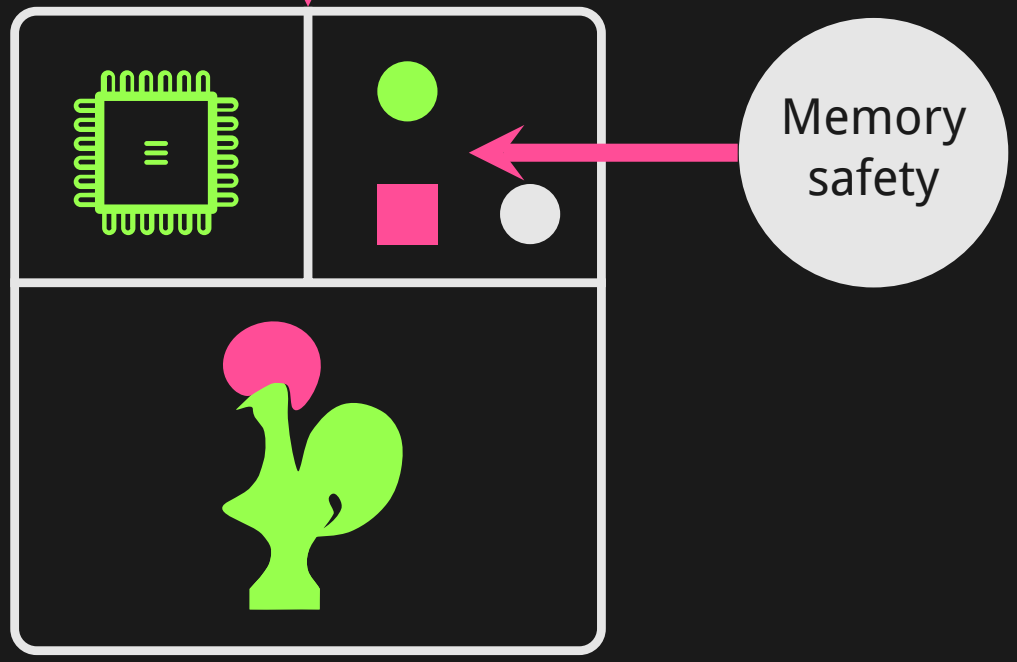
DMA, virtual memory, timing, ...

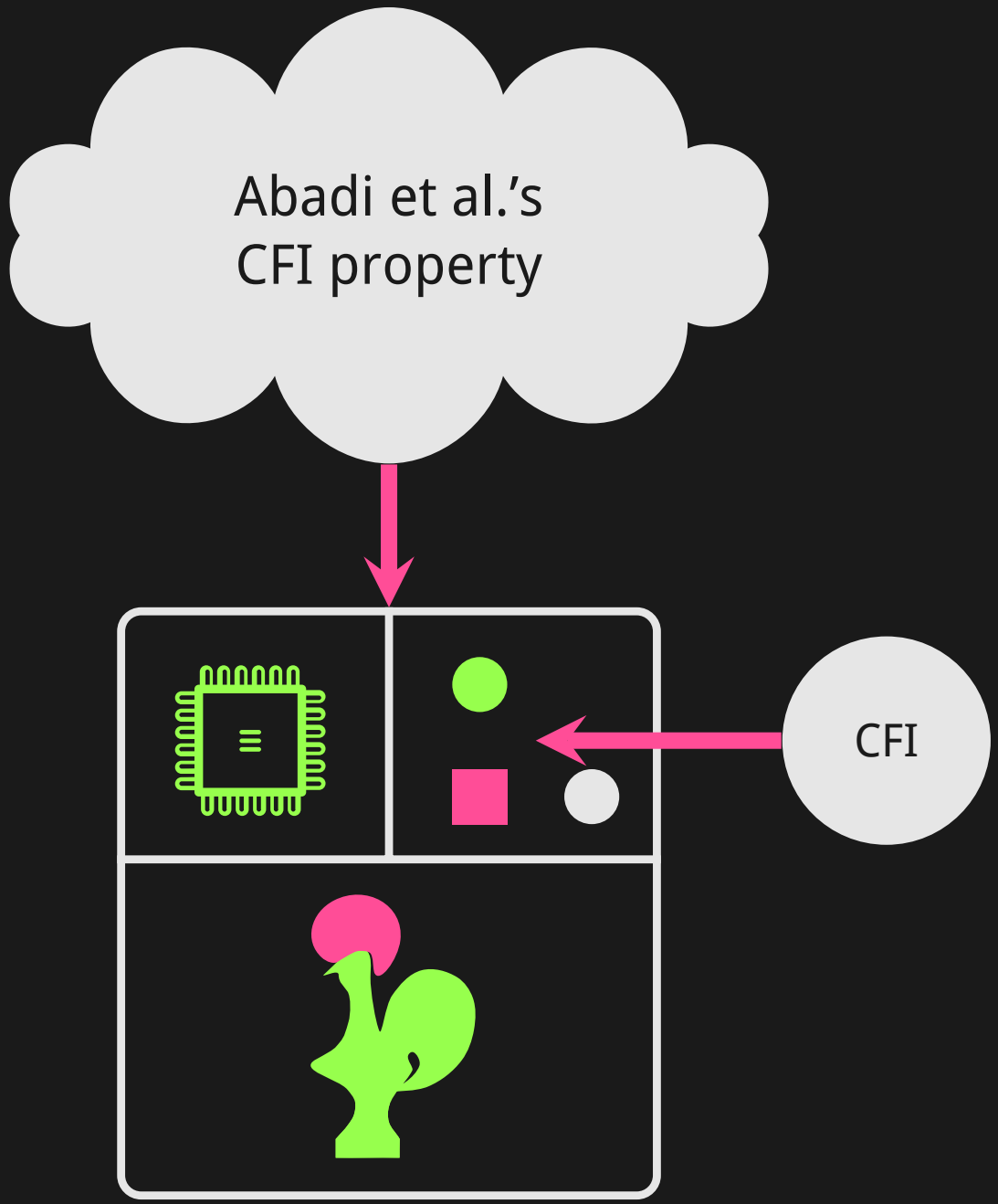

Higher-level
abstract machine
refined by

```
Inductive value :=
| Int : int → value
| Ptr : region → int → value.

Definition add v1 v2 :=
  match v1, v2 with
  | Int n, Int m ⇒ Some (Int (n + m))
  | Ptr r off, Int n
  | Int n, Ptr r off ⇒ Some (Ptr r (off + n))
  | _, _ ⇒ None
  end.
```

Memory-safe
abstract machine

refined by




Abadi et al.’s
CFI property
CFI

Takeaway

Expressive, high-level policies for security of low-level code

Takeaway

Expressive, high-level policies for security of low-level code

(with reasonable overheads)

What Else?

In the paper

- Detailed description of micro-policies and programming model
- Discussion of formal proofs
- Model of the PUMP hardware extension, where policy is implemented by machine code and cache
- Monitor self protection

`http://github.com/micro-policies/micro-policies-coq`

What's Next?

More policies

Stack protection

Operating system

Full abstraction

Composition

Combine policies
and their guarantees

Improve guarantees

Scale up to real ISA

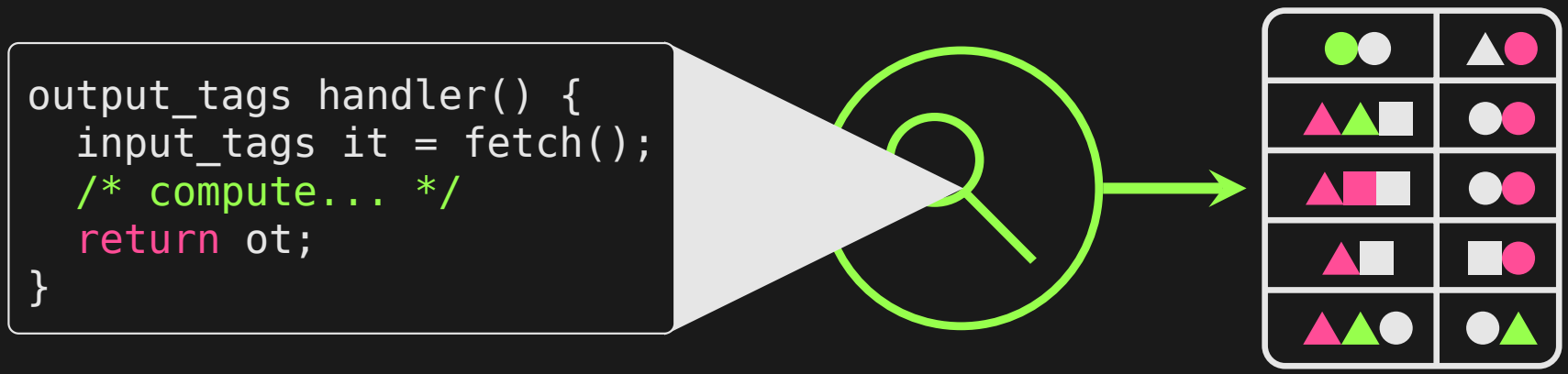
Verify implementations

Thank you!

Is it fast?

Meet the PUMP

(Programmable Unit for Metadata Processing)



Software implementation, hardware cache

Credits

Images borrowed from `thenounproject.com`

- Snail by Irene
- Chip by Arthur Shlain