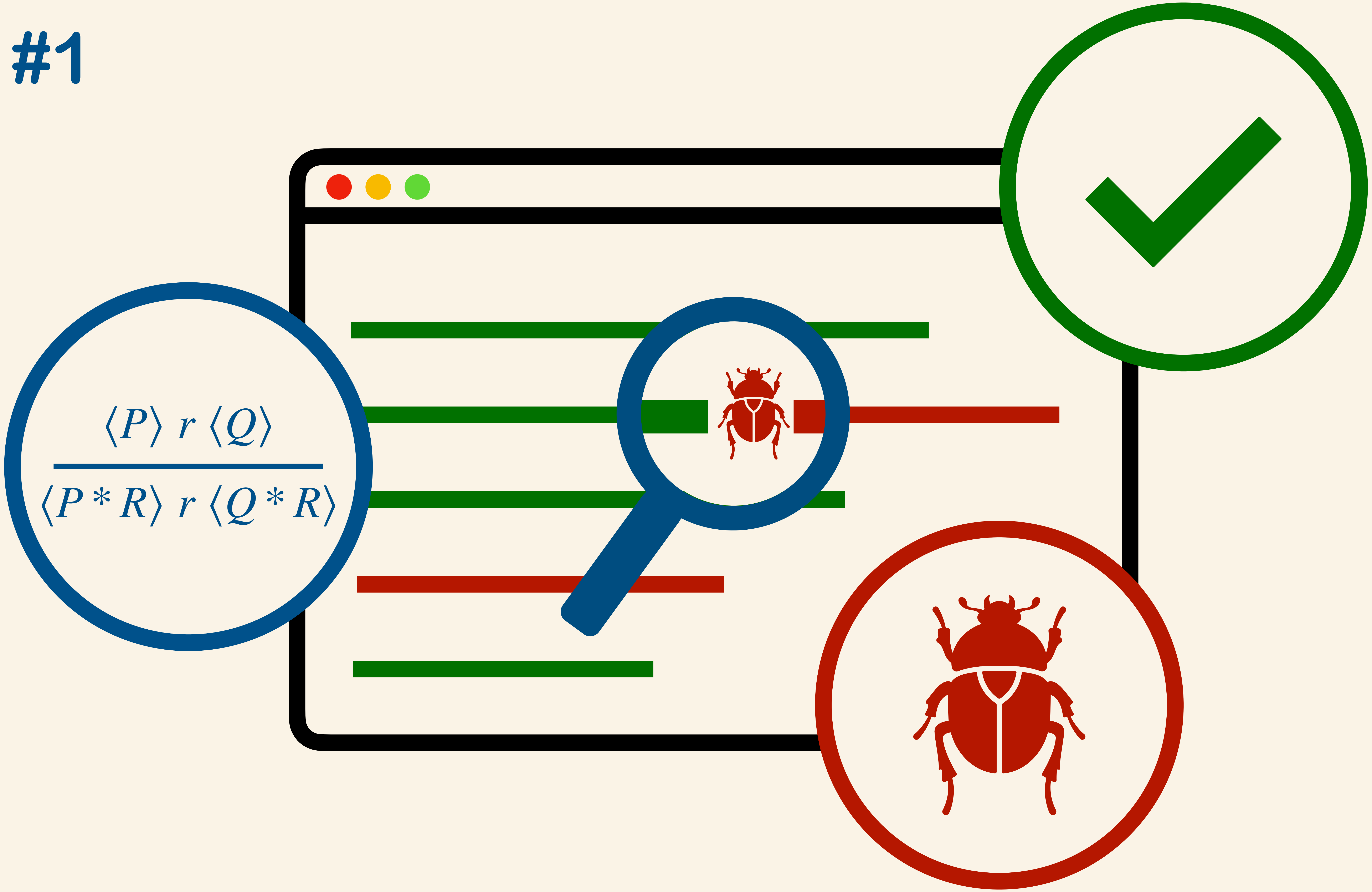
SCAN ME

Program Analysis

Lecture #1

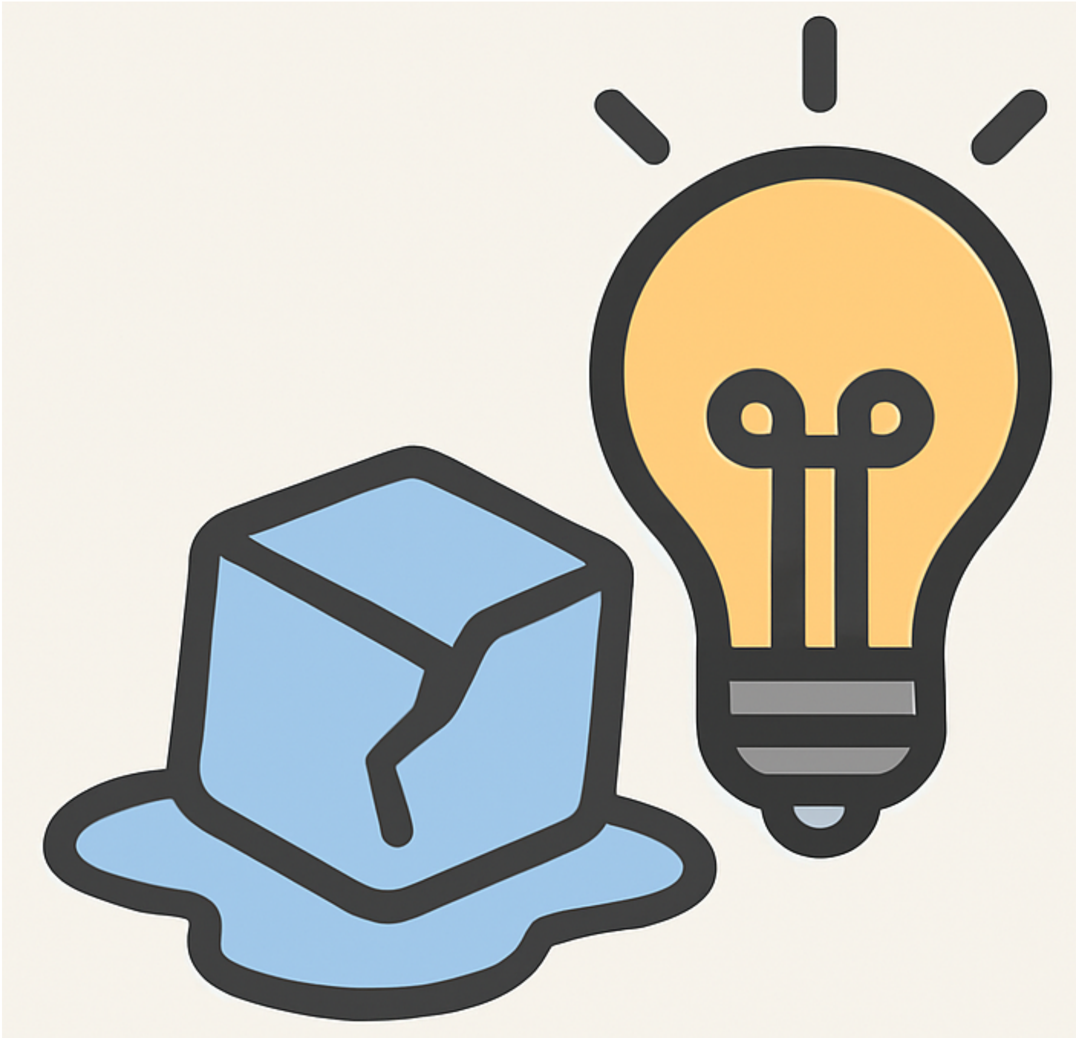
Roberto Bruni

Università di Pisa



PhD Course
June 30 - July 4, 2025

GRAN SASSO SCIENCE INSTITUTE
SCHOOL OF ADVANCED STUDIES
Scuola Universitaria Superiore



Icebreaker brain teasers

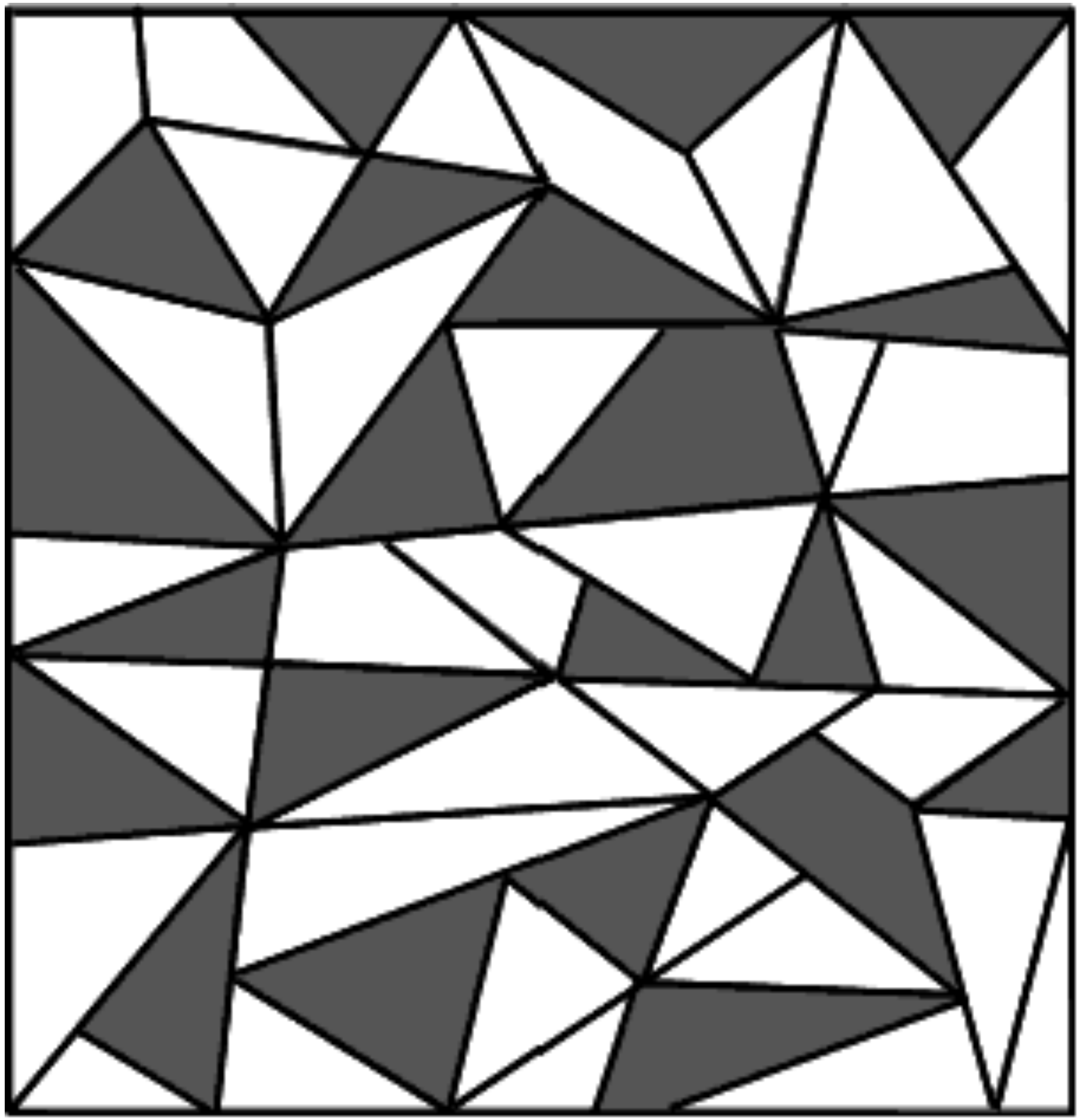

Where's Wally?
The first Where's Wally? book was published on September 21, 1987 and was created by English children's author and illustrator Martin Handford
Wally is a time traveller – he has been to Ancient Rome, the Stone Age and even the future. He has met pirates, knights, giants, dinosaurs and Robin Hood
More than 58m Where's Wally? books have been sold worldwide
Wally's Facebook fans:
4.2million
Wally app downloads:
6.8million
Wally isn't the only person to find in the pictures. There is his friend Wenda, his rival Odlaw (who is dressed in black and yellow. Odlaw is 'Waldo' spelt backwards), Wizard Whitebeard and me, Wally's dog, Woof
Where's Wally? has been translated into
30 languages...
... and published in
38 countries
Wally has many aliases depending on which country he is in, including...
Charlie (French)
Vallu (Finnish)
Gile (Serbian)
Waldo (American)
Hetti (Hindi)
Walter (German)
Holger (Danish)
Willy (Norway)
Valli (Icelandic)
Wolli (Korean)


HELP
HA HA HA

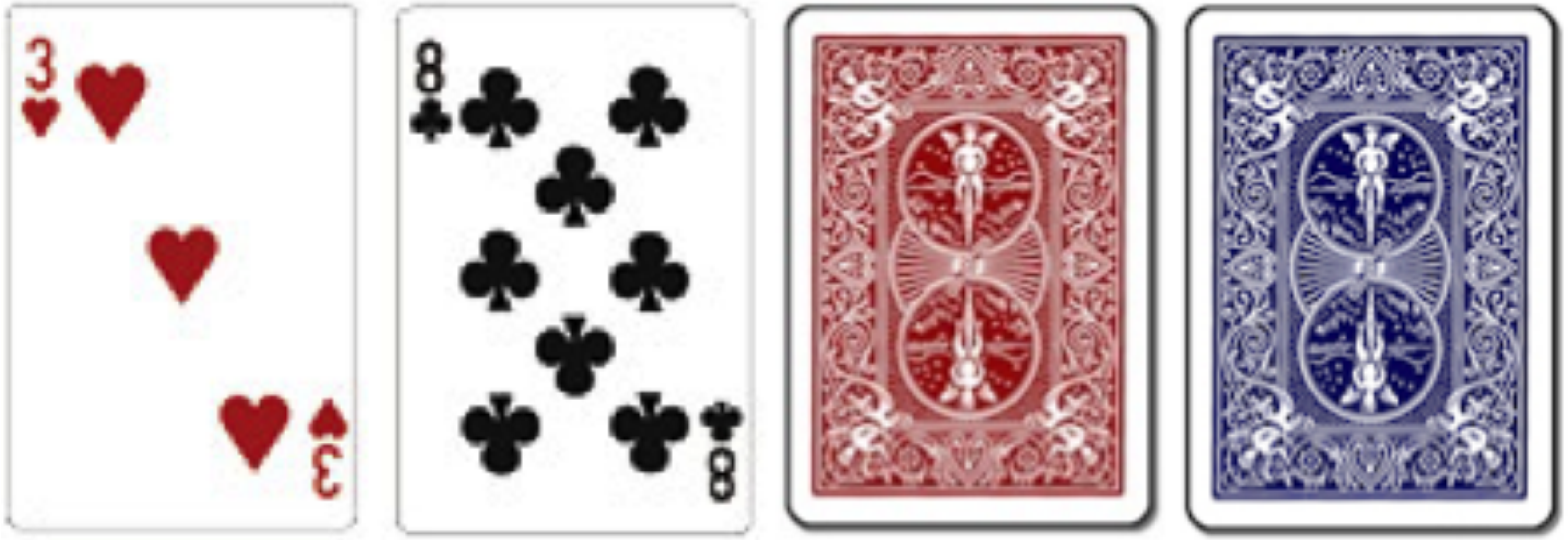
A metaphor for logical thinking

Where is the **regular five-pointed star**?
(There is one, really! No tricks!)

If you see it, raise your hand
or write '**found!**' on chat,
but don't point it out to your friends



Simple is not necessarily obvious



1 2 3 4

Which cards **must** be turned over to **make sure** the following claim is true?

“If the front face of a card bears an even number, then its back face is red”

Which implication is (always) valid?

$$(\exists x\,.\,\forall y\,.\,P) \implies (\forall y\,.\,\exists x\,.\,P)$$ **1**

$$(\forall y\,.\,\exists x\,.\,P) \implies (\exists x\,.\,\forall y\,.\,P)$$ **2**

All cats are the same colour

All cats are the same colour

base case $(n = 1)$**:** trivial

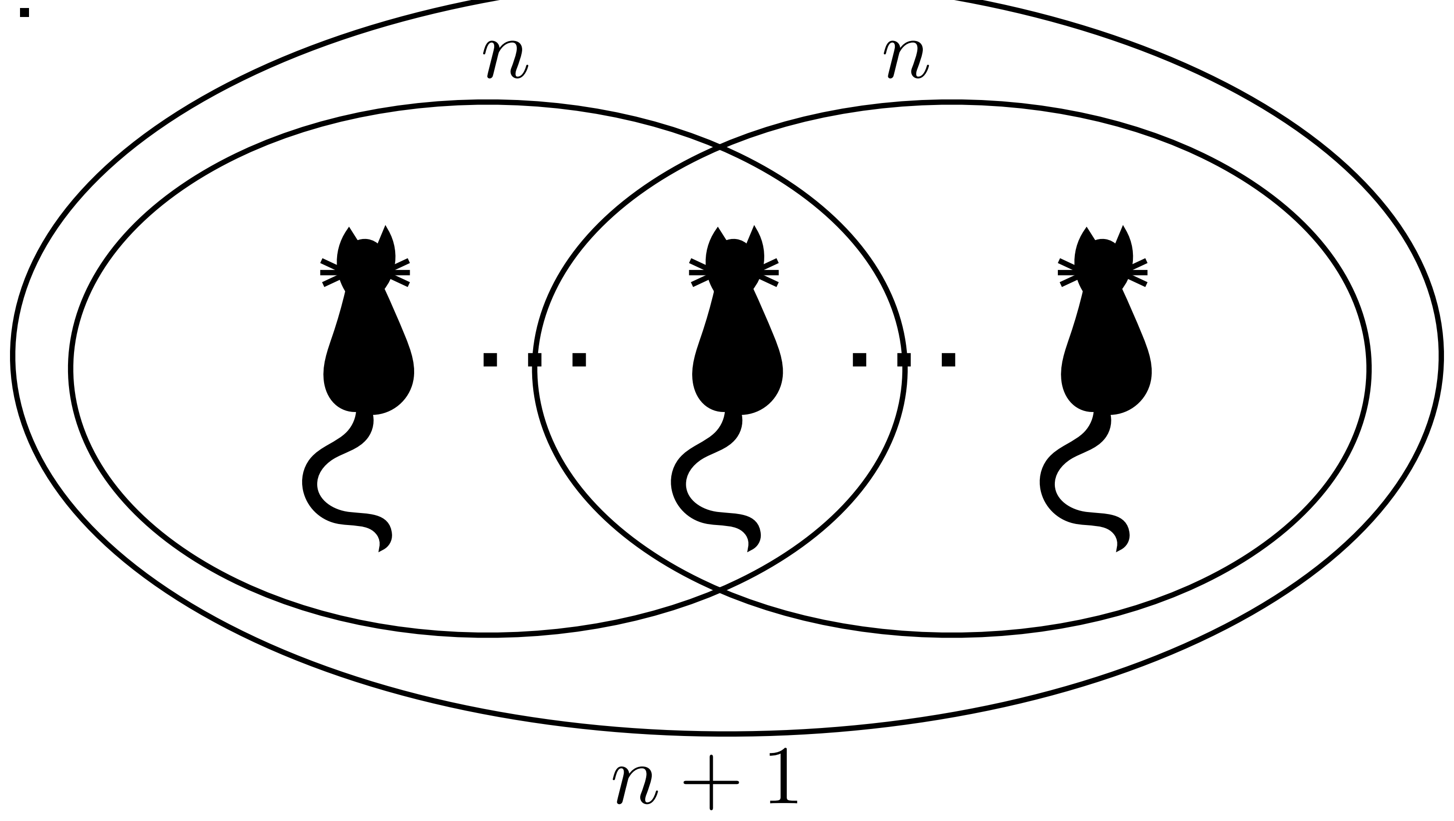
inductive case: taken a generic n, we assume the property holds for all groups with $k \leq n$ cats and prove it holds for any group with $n + 1$ cats as well.

Take $n + 1$ cats and place them along a line (this is the hardest part of the proof!).

By inductive hypothesis, the first n cats are the same colour.

By inductive hypothesis, the last n cats are the same colour.

Since the cats in the middle of the line belongs to both groups, by transitivity all $n + 1$ cats are the same colour.



What's wrong?

How would you rate your knowledge?

First order logic ☆☆☆

Denotational semantics ☆☆☆

Fixed points ☆☆☆

Hoare triples ☆☆☆

General info

Lectures plan

Monday	June 30	14:30-16:30
Tuesday	July 1	15:00-17:00
Wednesday	July 2	15:00-17:00
Thursday	July 3	14:00-16:00

Topics

Proving correctness: Hoare logic (HL)
Finding bugs: Incorrectness logic (IL)
Backward analysis: NC and SIL
Heap analysis: Separation logic(s)

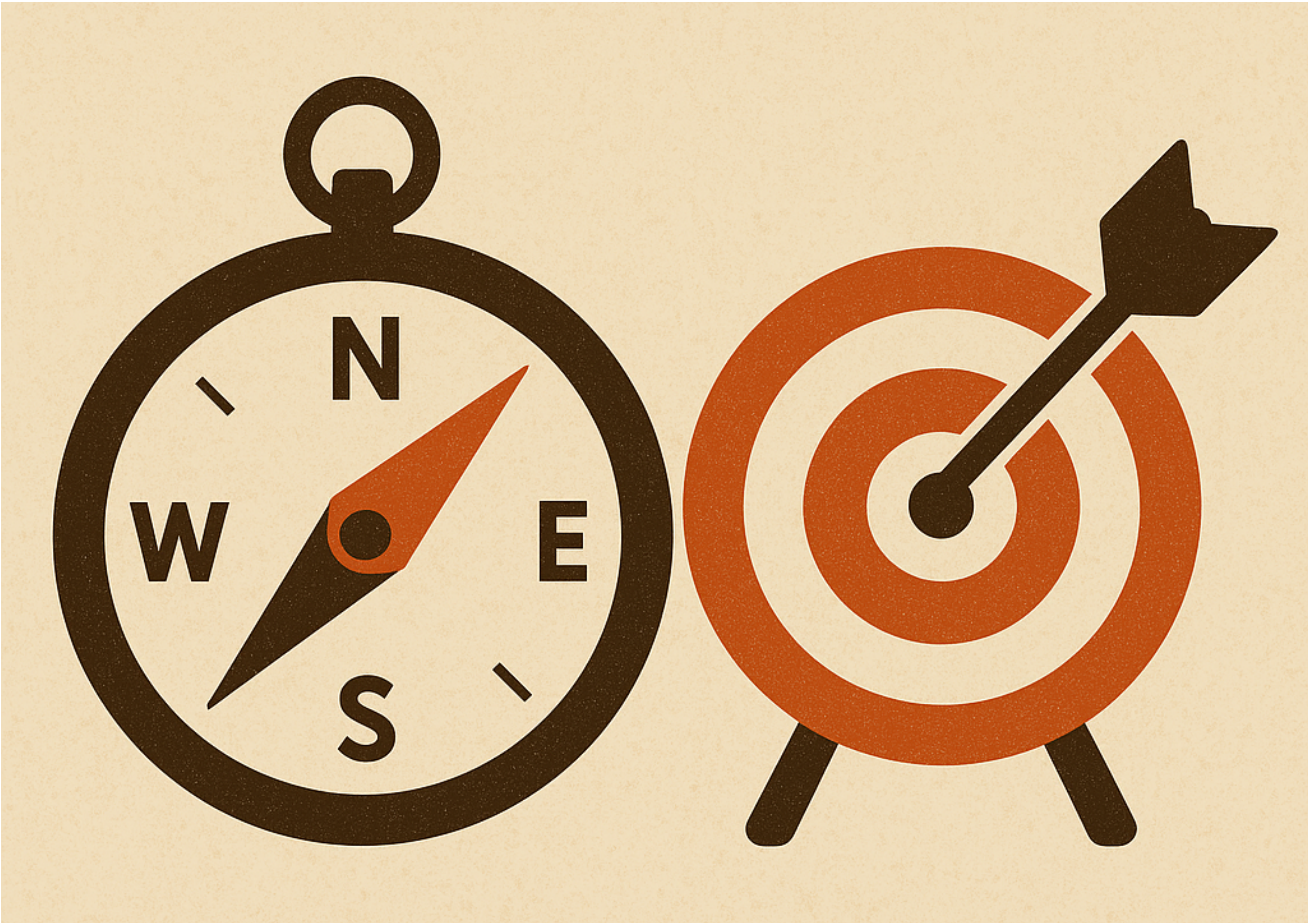
Exams?

Active participation during lectures?

Solving selected exercises?

Short oral Q&A exam session?

5' presentations (elevator pitch)?



Introduction and motivation

The need for verification

Friday, 24th June [1949]

Checking a large routine by Dr A. Turing.

How can one check a routine in the sense of making sure that it is right?

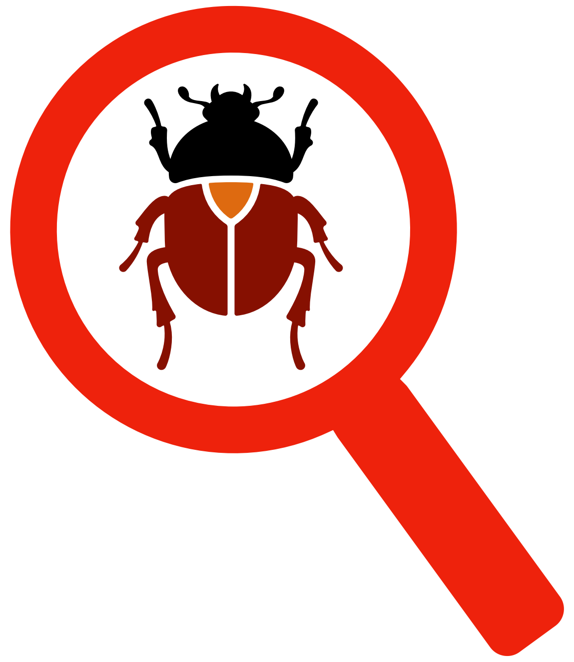
"Program correctness and incorrectness are two sides of the same coin"

Peter O'Hearn (2020)

Software Verification

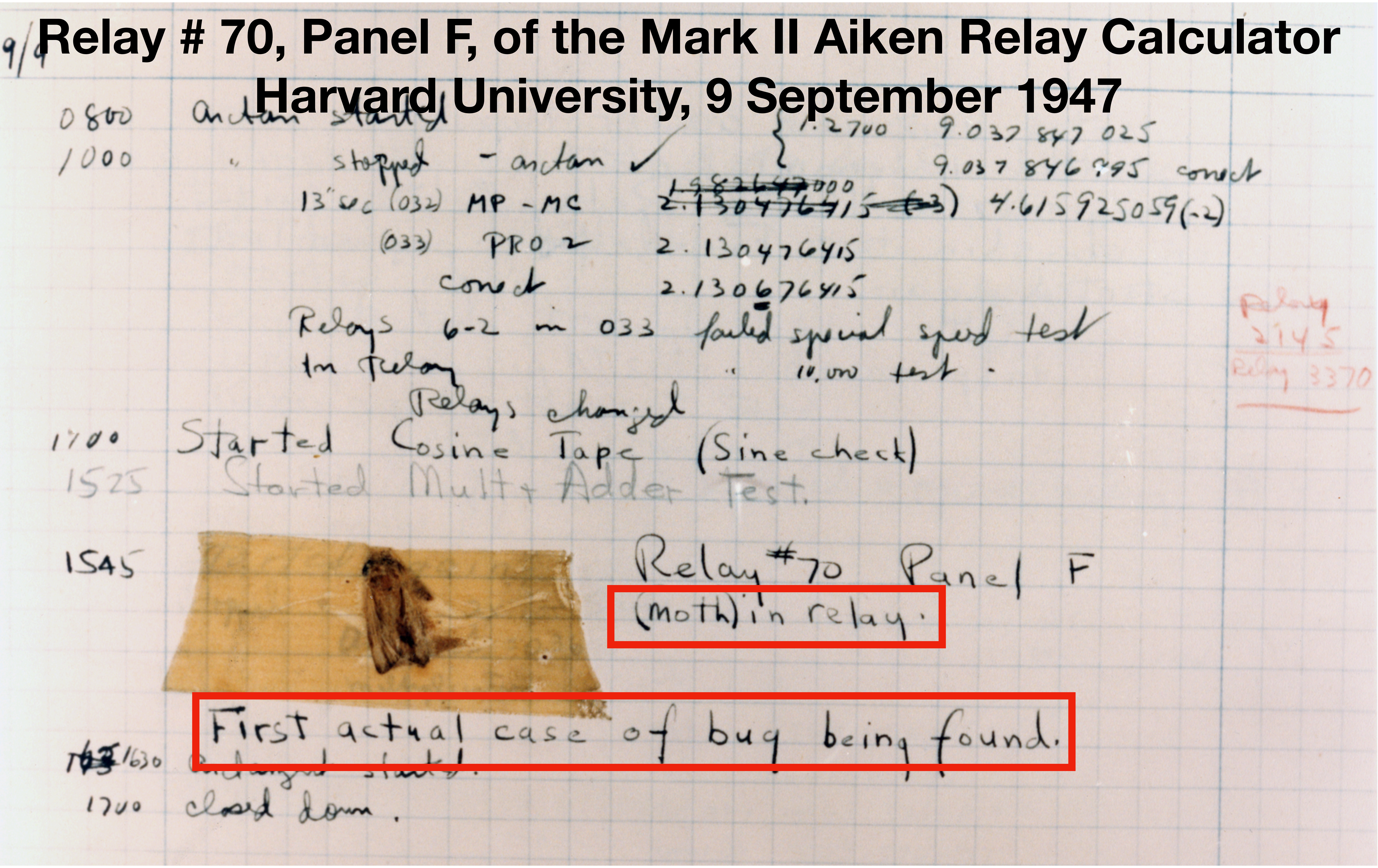
Correctness the aim is to prove the absence of bugs

Incorrectness the aim is to prove the presence of bugs



Have you seen this picture before?

Bugs



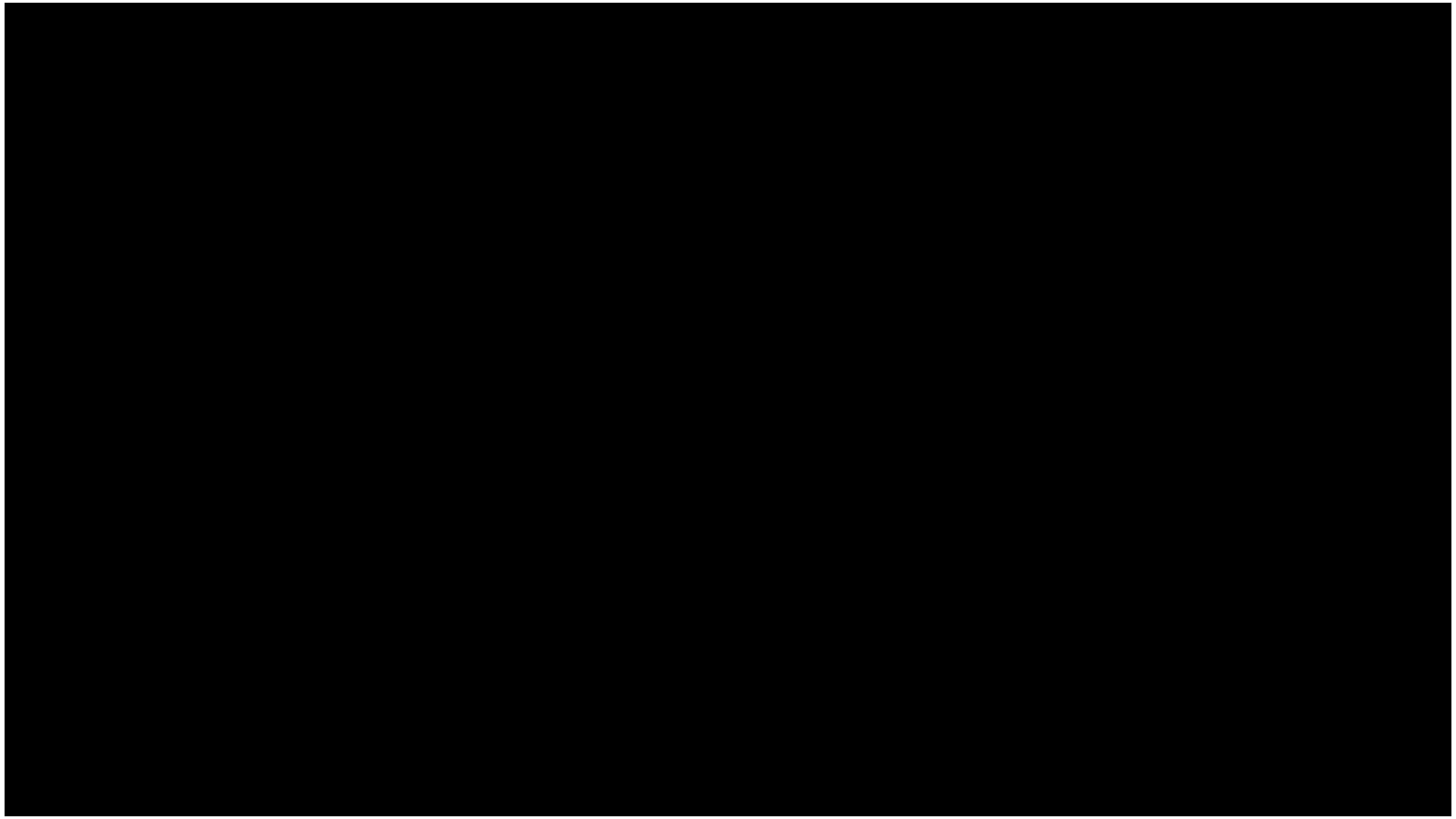
Relay # 70, Panel F, of the Mark II Aiken Relay Calculator
Harvard University, 9 September 1947

WIKIPEDIA
The Free Encyclopedia

A **software bug** is an error, flaw or fault in the design, development, or operation of computer software that causes it to produce an incorrect or unexpected result

Why do we need to verify our code?

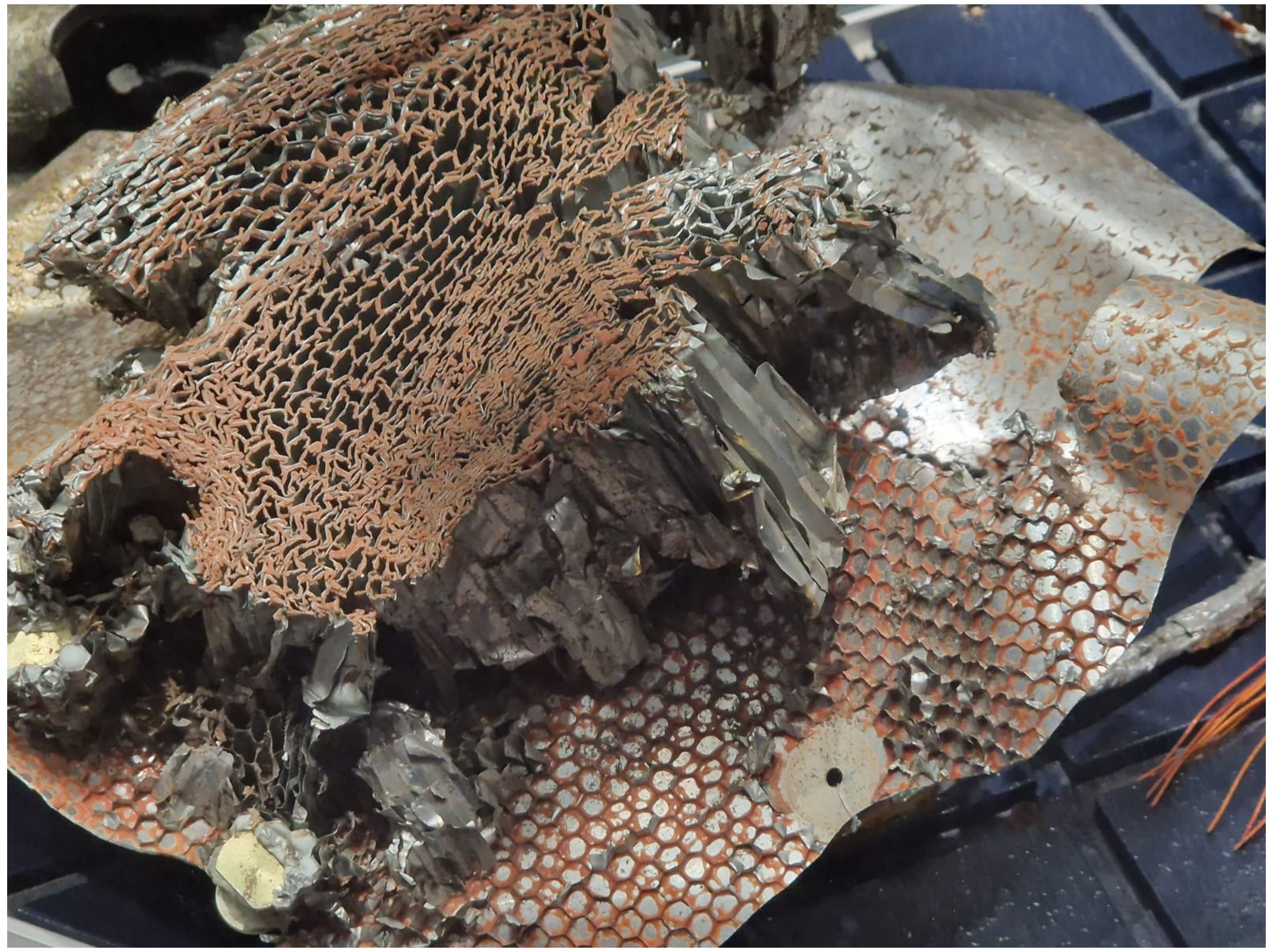
The code that exploded Ariane 5 rocket!
(video duration 5'45'')

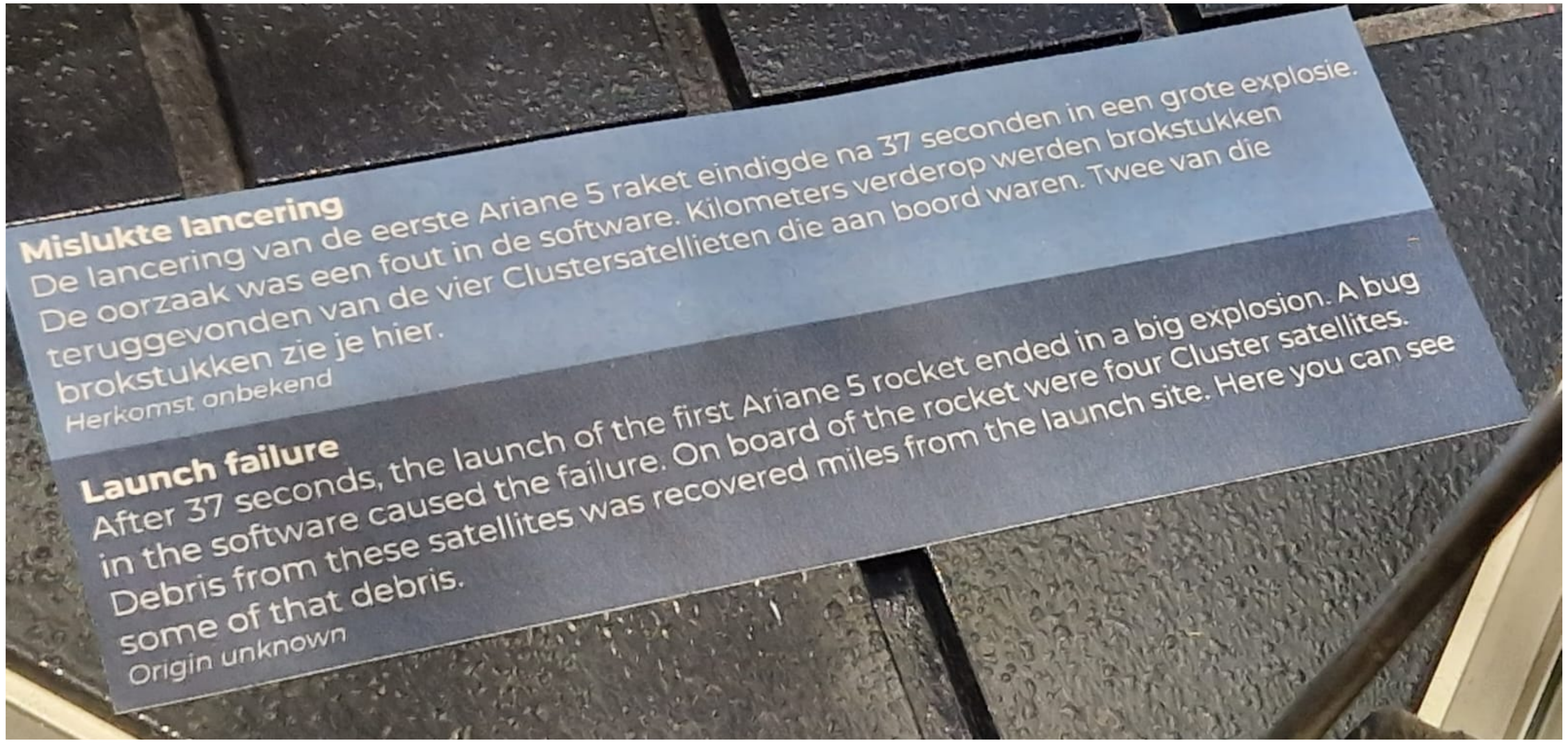


Ariane 5 Rocket Explosion (1996)

Attempt to fit 64-bit data into 16-bit data
(numeric overflow error): $100M for loss of mission

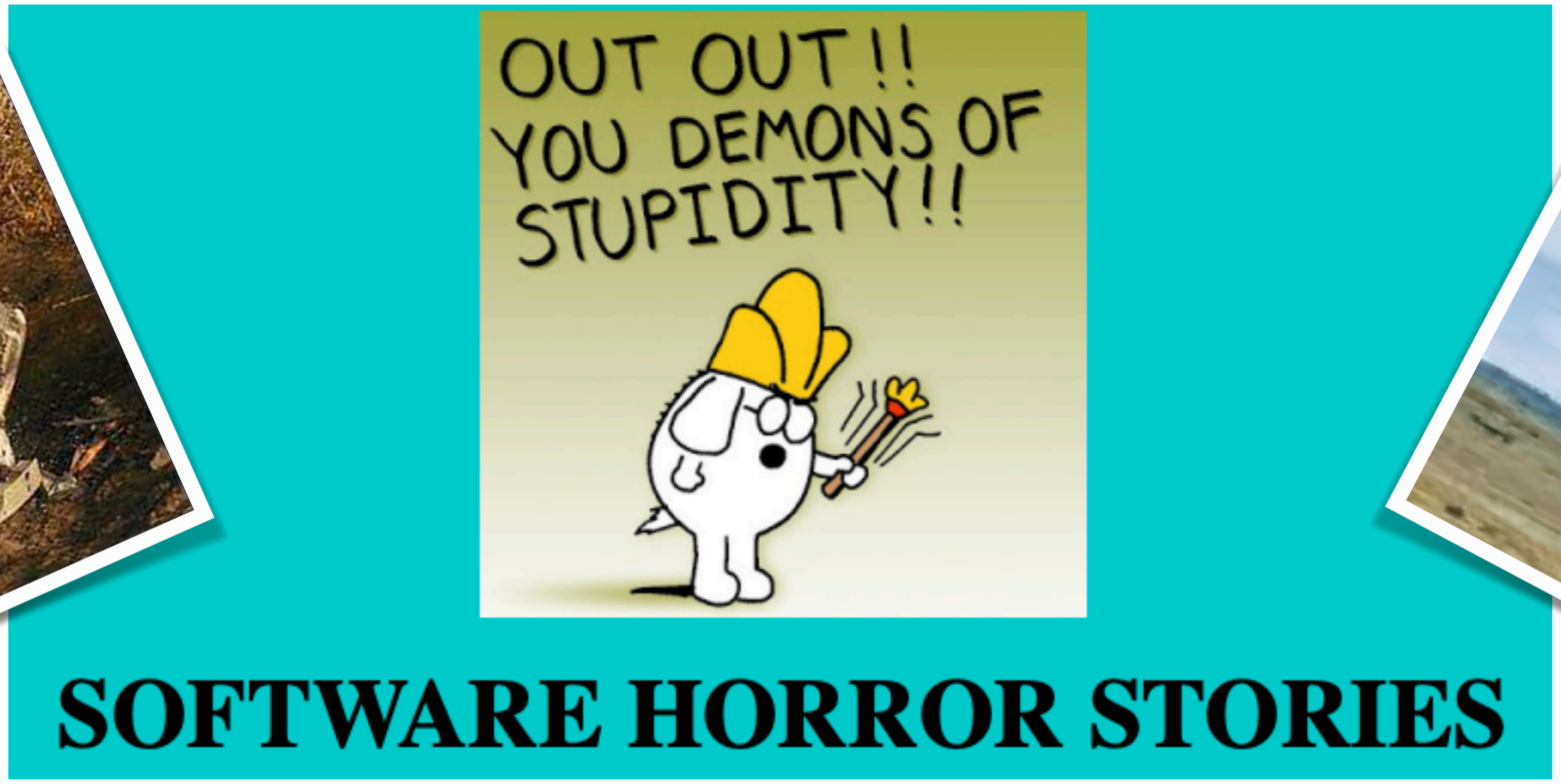
Read more at:
https://www.bugsnag.com/blog/bug-day-ariane-5-disaster/





Unfortunately

It was one of the most serious but not the only one….



https://www.cs.tau.ac.il/~nachumd/horror.html

Toyota unintended acceleration
4 people died

Boeing 747 Max Crashes
350 people died

Costs of SW bugs

Knight Capital Trading Glitch (2012)
$ 440 M

Nissan Airbag Malfunction (2014)
1 Million Vehicles Recalled

Software Fails Watch (Tricentis, 2017): SW bugs lead to $ 1.7 Trillion revenue lost.

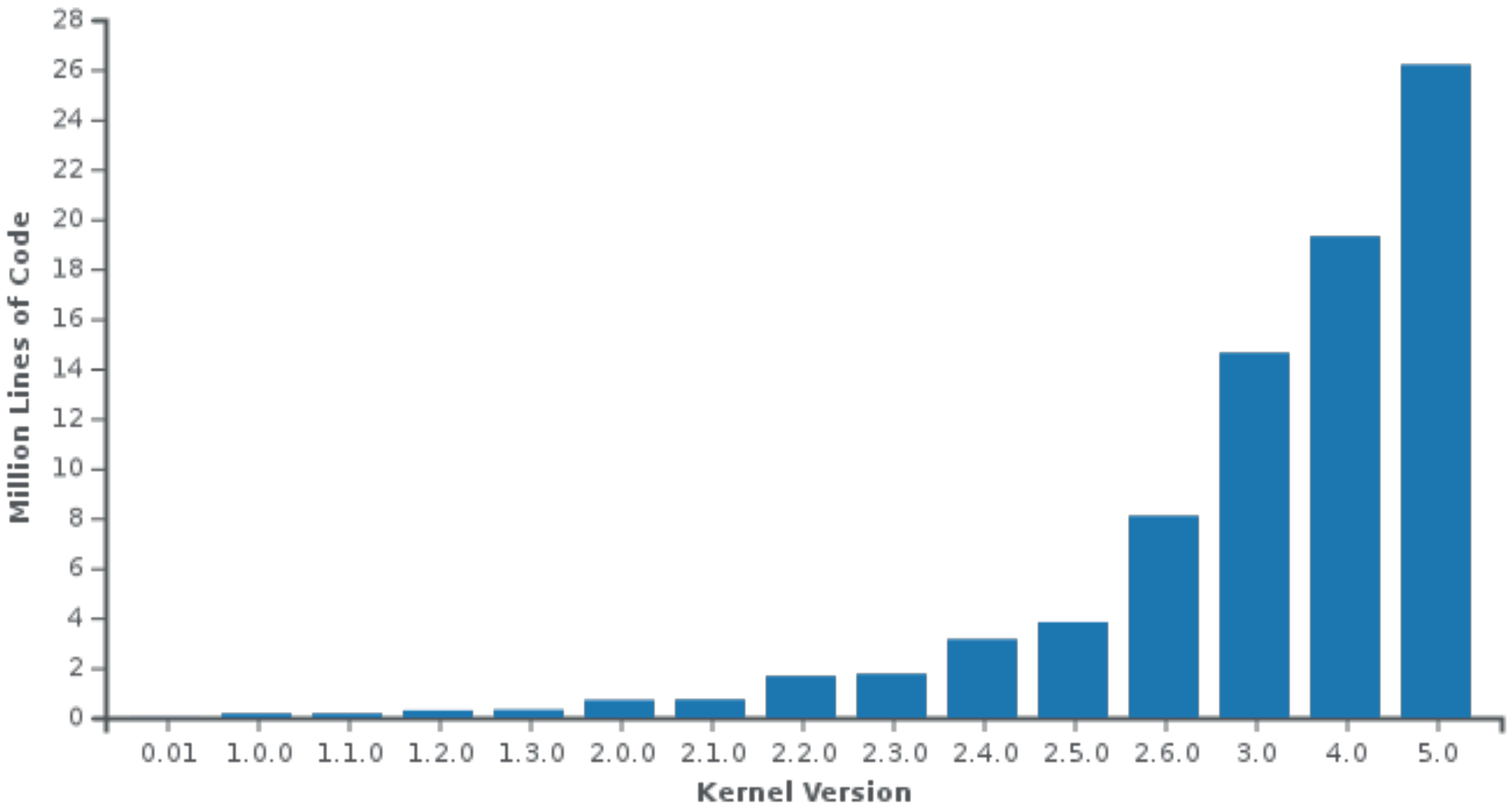
CISION PR Newswire (2020): SW bugs cost $ 61 Billion loss in productivity annually.

https://www.tricentis.com/news/tricentis-software-fail-watch-finds-3-6-billion-people-affected-and-1-7-trillion-revenue-lost-by-software-failures-last-year/

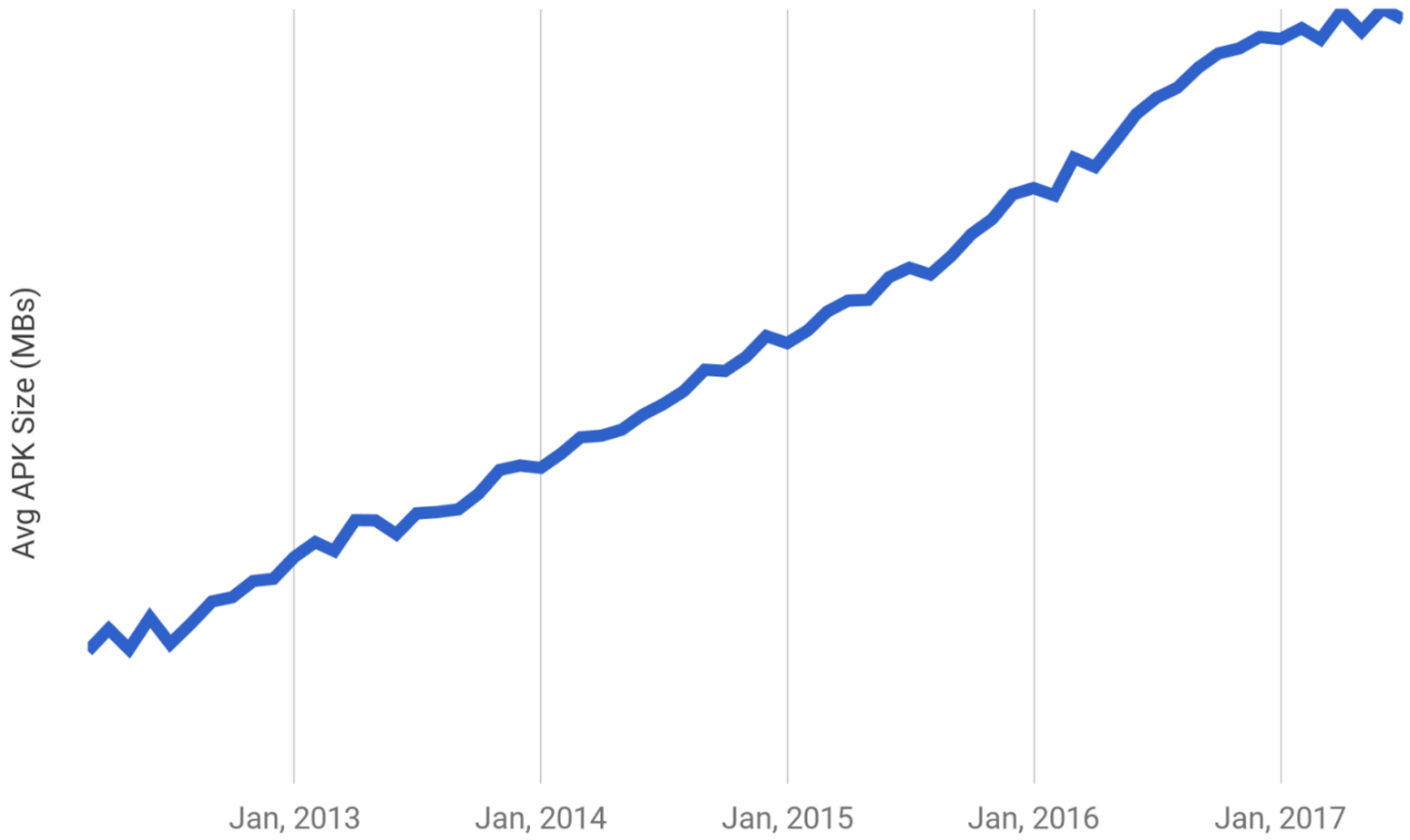
https://www.prnewswire.com/news-releases/study-software-failures-cost-the-enterprise-software-market-61b-annually-301066579.html

Complexity of programs

Size of Linux Kernel



Avg. Size of Android Apps



always increasing!

Is there any bug free program?

"There are two ways of constructing a sw design:
one way is to make it **so simple that there are obviously no deficiencies**,
and the other way is to make it **so complicated that there are no obvious deficiencies**"

Tony Hoare (1980 Turing award lecture)

Success stories

A long time before success

Computer-assisted verification is an old idea

- Turing, 1948
- Floyd-Hoare logic, 1969

Success in practice: only from the mid-1990s

- Importance of the *increase of performance of computers*

A first success story:

- Paris metro line 14, using *Atelier B* (1998, refinement approach)

Other Famous Success Stories

- Flight control software of A380: *Astree* verifies absence of run-time errors (2005, abstract interpretation)
  `http://www.astree.ens.fr/`
- Microsoft's hypervisor: using Microsoft's *VCC* and the *Z3* automated prover (2008, deductive verification)
  `http://research.microsoft.com/en-us/projects/vcc/`
  More recently: verification of PikeOS
- Certified C compiler, developed using the *Coq* proof assistant (2009, correct-by-construction code generated by a proof assistant)
  `http://compcert.inria.fr/`
- L4.verified micro-kernel, using tools on top of *Isabelle/HOL* proof assistant (2010, Haskell prototype, C code, proof assistant)
  `http://www.ertos.nicta.com.au/research/l4.verified/`

The main question

Will our program behave as we intended?

We need to analyse all executions of the program

The semantics of a program is a description of its run-time behaviors

Checking if a software will run as intended is equivalent to checking if the code satisfies a (semantic) property of interest

Formal methods

Semantics = assigning meaning to syntax

A program	Its meaning
c	$[\![c]\!]$
syntax (how the program is written)	semantics (its computed function)

Memory states

state

set of variables

$$\sigma : X \to \mathbb{Z}$$

set of integers

set of all states

$$\Sigma \triangleq \{\sigma : X \to \mathbb{Z}\}$$

Forward semantics (deterministic code)

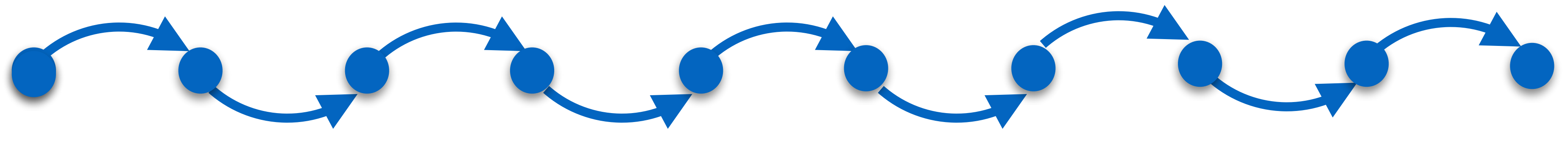
We start from input state σ and we want to characterise the reachable output states

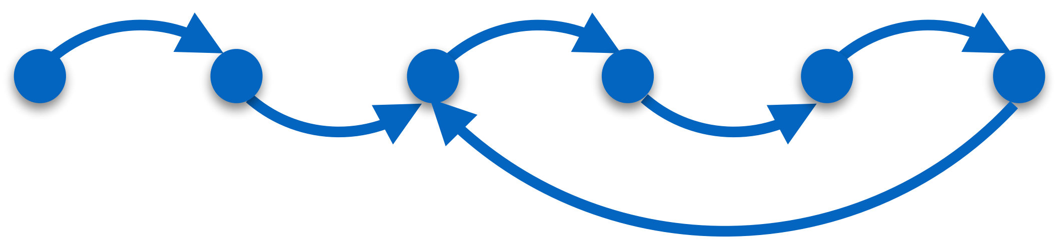
A program

Input store

c

Output store

σ  $[\![c]\!]\sigma$

σ  $[\![c]\!]\sigma = \bot$ Non terminating execution

Denotational semantics $[\![c]\!] : \Sigma \to \Sigma_\bot$ $\Sigma_\bot = \Sigma \uplus \{ \bot \}$

Example

$c \triangleq$

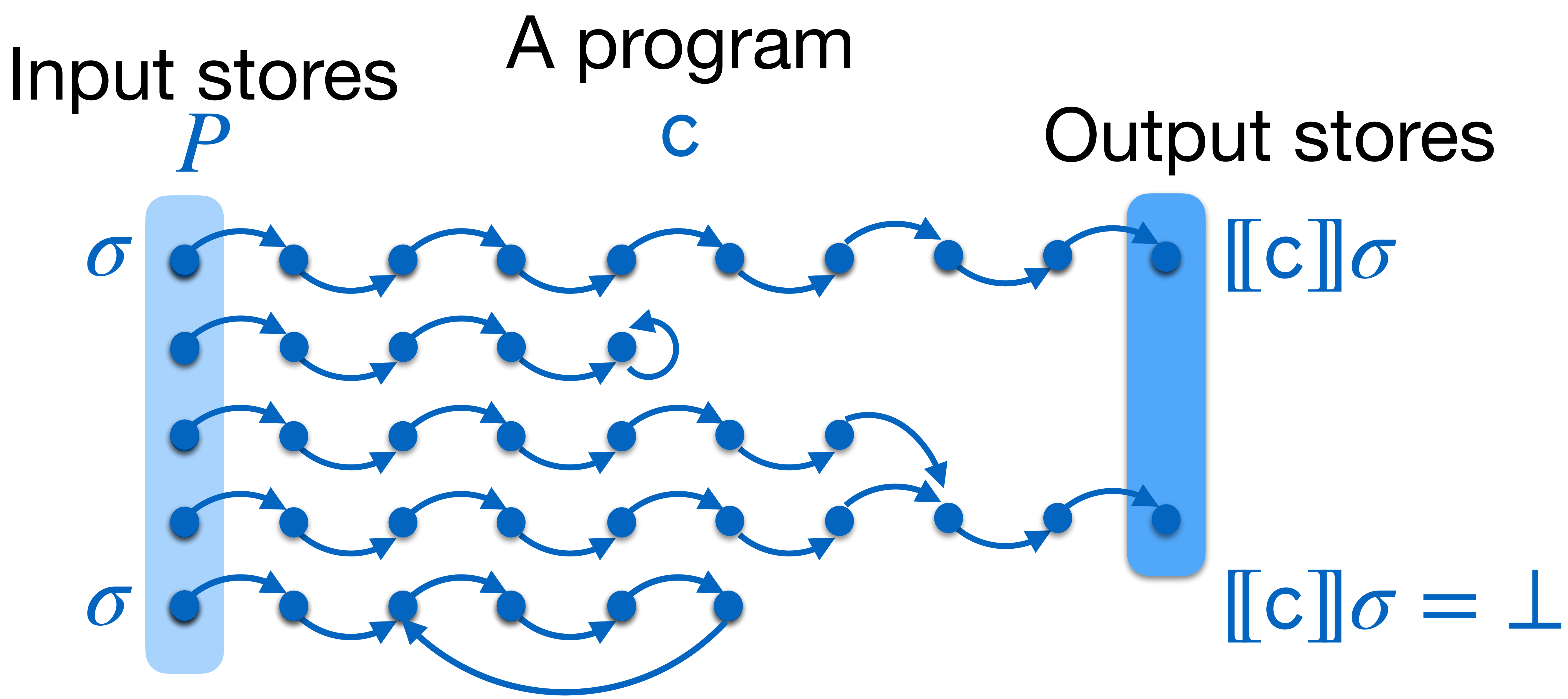
```
while (n>1) {
    n := n+1;
    x := 0;
}
x := n-1;
```

$$[\![c]\!][\mathtt{n} \mapsto 1] = [\mathtt{n} \mapsto 1, \mathtt{x} \mapsto 0]$$

0 is also the default value (left implicit)

$$[\![c]\!][\mathtt{n} \mapsto 2] = \bot$$

Collecting semantics (deterministic code)



$$[\![c]\!]P = \bigcup_{\sigma \in P} [\![c]\!]\sigma$$

Denotational semantics $[\![c]\!] : \Sigma \to \Sigma_\bot$

Collecting semantics $[\![c]\!] : \wp(\Sigma) \to \wp(\Sigma)$

Example

$c \triangleq$

```
while (n>1) {
    n := n+1;
    x := 0;
}
x := n-1;
```

$$[\![c]\!](\mathrm{n} > 1) = \emptyset$$

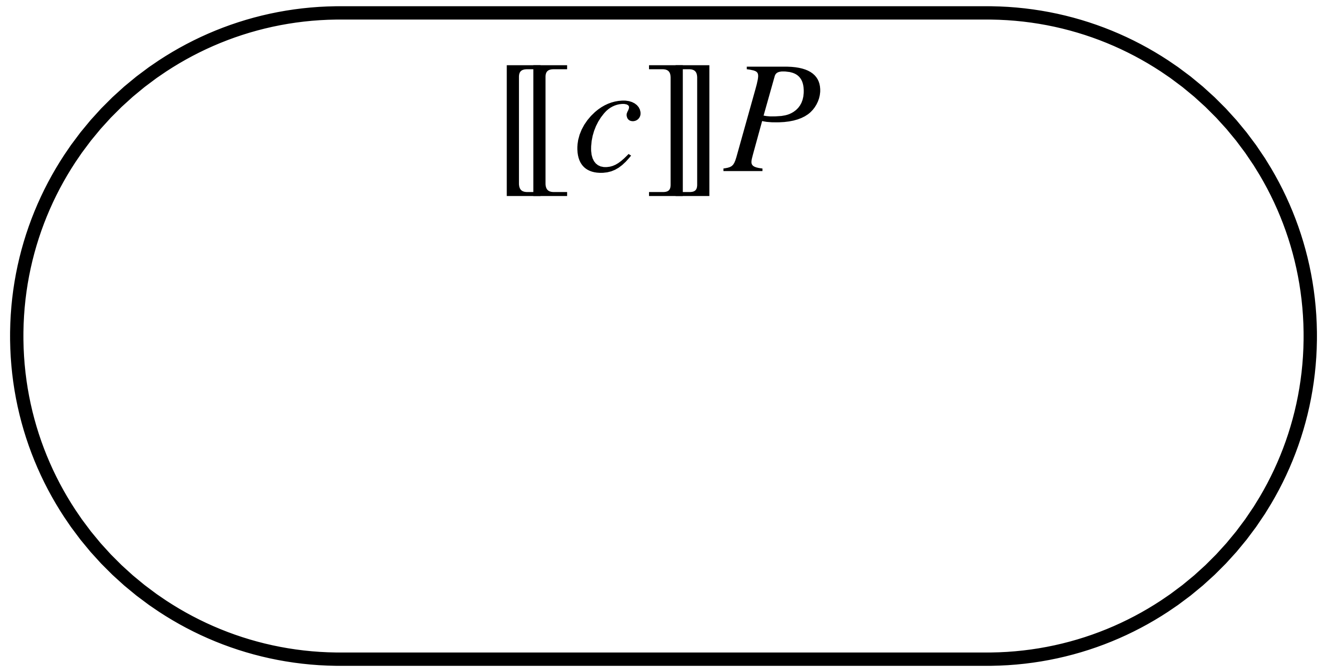
$$[\![c]\!](\mathrm{n} > 0) = \{[\mathrm{n} \mapsto 1, \mathrm{x} \mapsto 0]\}$$

$$[\![c]\!](\mathrm{n} \geq 0) = \{[\mathrm{n} \mapsto 1, \mathrm{x} \mapsto 0], [\mathrm{n} \mapsto 0, \mathrm{x} \mapsto -1]\}$$

$$[\![c]\!](\mathsf{true}) = (\mathrm{n} \leq 1, \mathrm{x} = \mathrm{n} - 1) \subseteq (\mathrm{n} \leq 1, \mathrm{x} \leq 0)$$

Exact analysis

$$[\![c]\!] : \wp(\Sigma) \to \wp(\Sigma)$$



🪲 $\in^{?} [\![c]\!]P$

it is a property about the computed function, not about how c is written

semantic property of a program: a property about $[\![c]\!]$

$$\mathscr{P}(c) \equiv \forall P \,.\, \forall \sigma \in [\![c]\!]P \,.\, \sigma(x) \neq 0$$

Undecidability in the way

non trivial property:

- there exists a program c_1 such that $\mathscr{P}(c_1)$ holds true
- and there exists also some program c_2 such that $\mathscr{P}(c_2)$ is false

Rice theorem.

Let $\mathscr{P}(c)$ be a **non trivial** semantic property of programs c.

There exists no **algorithm** such that, **for every program** c,

it returns true **if and only if** $\mathscr{P}(c)$ holds true

no analysis method that is automatic, universal, exact !

algorithmic | for any program | no false positive/negative

For some program…

$$\mathscr{P}(c) \equiv \forall P \neq \emptyset \,.\, \exists \sigma \in [\![c]\!]P \,.\, \sigma(x) \neq 0$$

$c \triangleq$

```
x := 1;
```

...and for some other program

$$\mathscr{P}(c) \equiv \forall P \neq \varnothing \,.\, \exists \sigma \in [\![c]\!]P \,.\, \sigma(x) \neq 0$$

$c \triangleq$

```
while (n>1) {
    n := n+1;
    x := 0;
}
x := n-1;
```

Collatz's conjecture

$$f(n) \triangleq \begin{cases} 1 & \text{if } n \leq 1 \\ f(n/2) & \text{else if } n\%2 = 0 \\ f(3n+1) & \text{otherwise} \end{cases}$$

$$\forall n \,.\, f(n) = 1$$

$$f(12) = f(6) = f(3) = f(10) = f(5) = f(16) = f(8) = f(4) = f(2) = f(1) = 1$$

The **Collatz conjecture**[a] is one of the most famous unsolved problems in mathematics. The conjecture asks whether repeating two simple arithmetic operations will eventually transform every positive integer into 1. It concerns sequences of integers in which each term is obtained from the previous term as follows: if a term is even, the next term is one half of it. If a term is odd, the next term is 3 times the previous term plus 1. The conjecture is that these sequences always reach 1, no matter which positive integer is chosen to start the sequence. The conjecture has been shown to hold for all positive integers up to 2.95×10^{20}, but no general proof has been found.

Unsolved problem in mathematics:

?

- *For even numbers, divide by 2;*
- *For odd numbers, multiply by 3 and add 1.*

With enough repetition, do all positive integers converge to 1?

(more unsolved problems in mathematics)

It is named after the mathematician Lothar Collatz, who introduced the idea in 1937, two years after receiving his doctorate.[4] The sequence of numbers involved is sometimes referred to as the **hailstone sequence**, **hailstone numbers** or **hailstone numerals** (because the values are usually subject to multiple descents and ascents like hailstones in a cloud),[5] or as **wondrous numbers**.[6]

And for Collatz's conjecture?

$$\mathscr{P}(c) \equiv \forall P \neq \emptyset \,.\, \exists \sigma \in [\![c]\!]P \,.\, \sigma(x) \neq 0$$

$c \triangleq$

```
while (x>1) {
    if (even(x)) { x := x/2; }
              else { x:= 3x+1; }
} % does it terminate for any value of x?
```

Limitations of the analysis

no analysis method that is automatic, universal, exact !

We need to give something up:

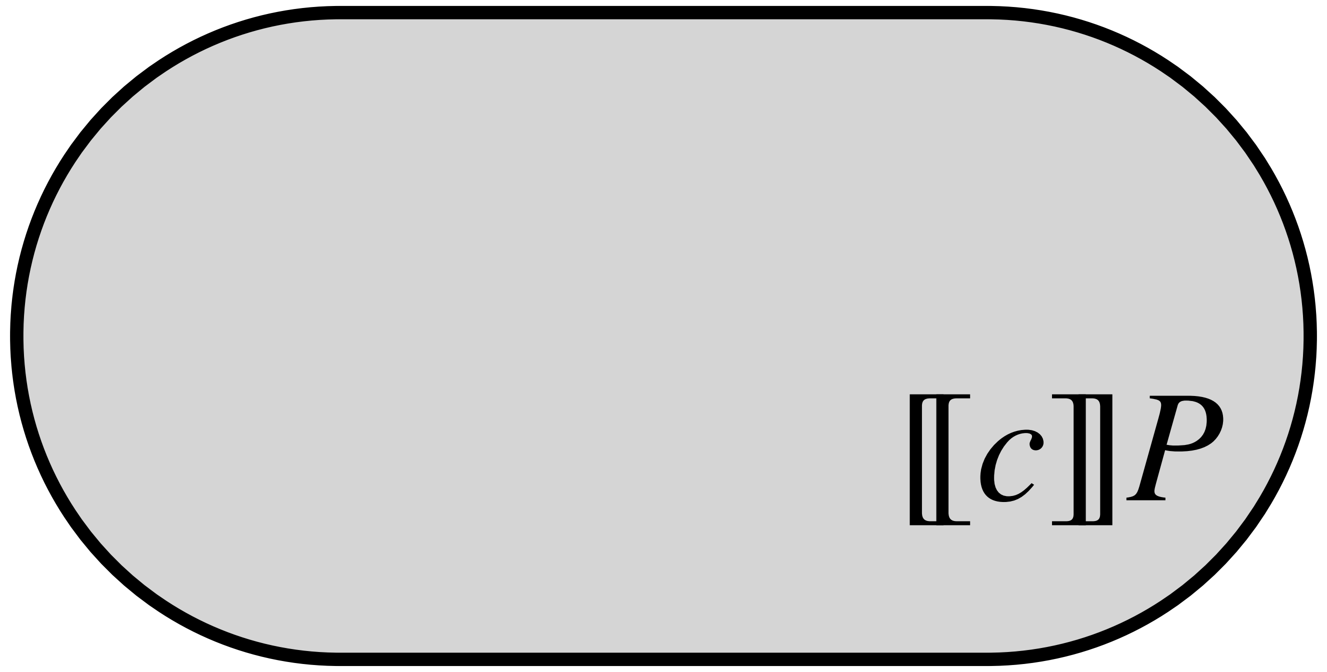
automation: machine-assisted techniques

the **universality** “for all programs”:
targeting only a restricted class of programs

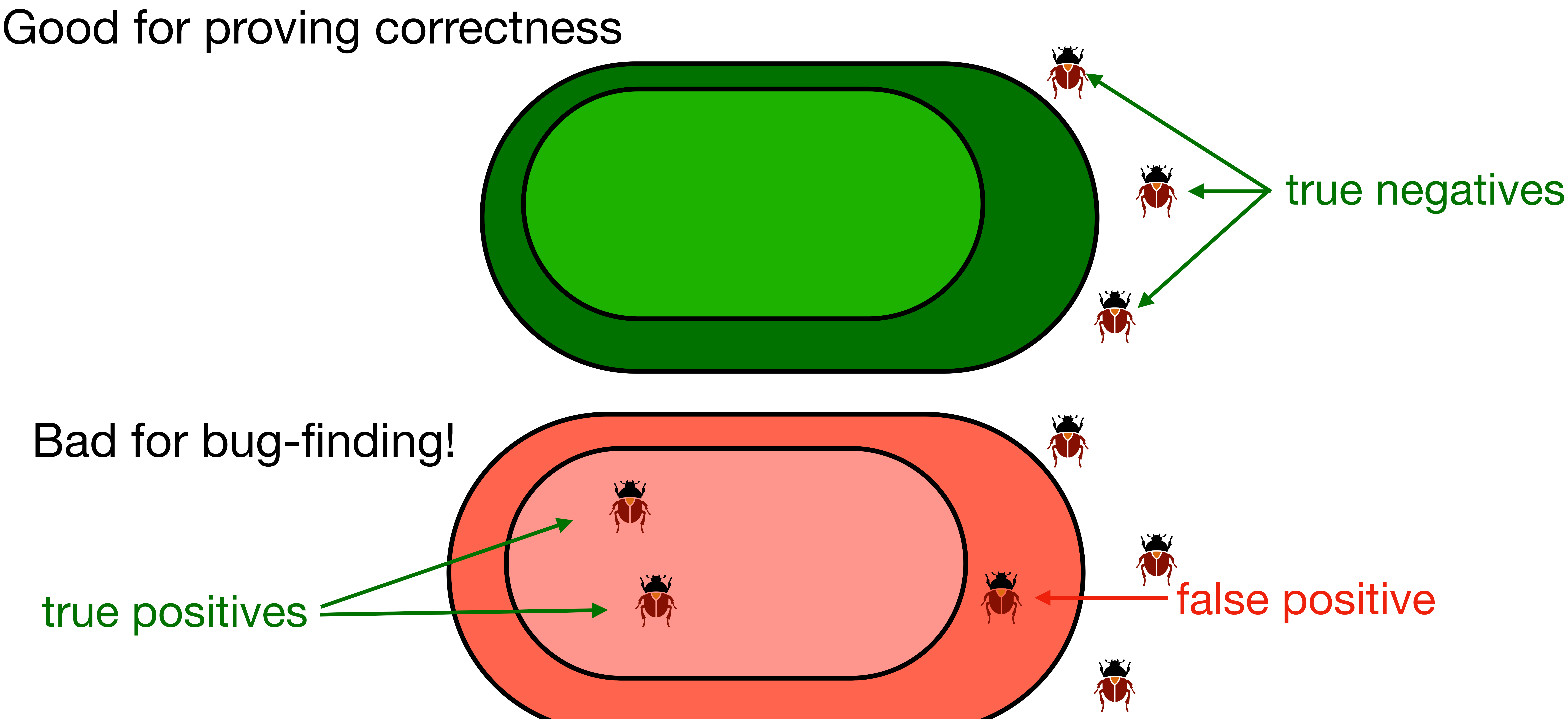
claim to find **exact** answers: **introduce approximations**


young
time
money
energy
adult
time
money
energy
old
time
money
energy
Me
time
money
energy

Over approximations



Over approximations



Example

$c \triangleq$

```
while (n>1) {
    n := n+1;
    x := 0;
}
x := n-1;
```

$$[\![c]\!](\mathtt{n} \geq 0) = \{[\mathtt{n} \mapsto 1, \mathtt{x} \mapsto 0], [\mathtt{n} \mapsto 0, \mathtt{x} \mapsto -1]\}$$

Correct

$$[\![c]\!]^{ov}(\mathtt{n} \geq 0) = \{\mathtt{n} \in \{0,1\}, \mathtt{x} \leq 0\}$$

```
y := 1/(x-2);
```

Undefined behaviour for 🐞 x= 2

$$🐞 \notin [\![c]\!]^{ov}(\mathtt{n} \geq 0) \implies 🐞 \notin [\![c]\!](\mathtt{n} \geq 0)$$

We can prove correctness!!

Example

$c \triangleq$

```
while (n>1) {
    n := n+1;
    x := 0;
}
x := n-1;
```

```
y := 1/(x+2);
```

Undefined behaviour for x=-2

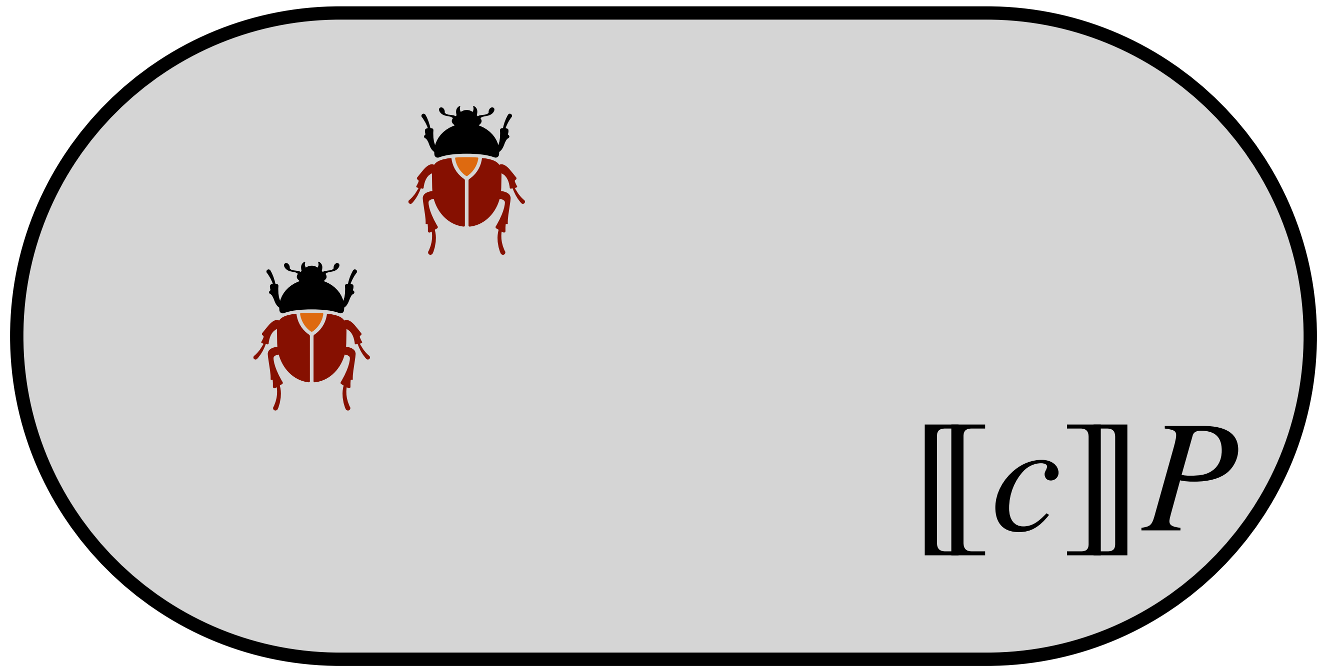
$$[\![c]\!](\mathtt{n} \geq 0) = \{[\mathtt{n} \mapsto 1, \mathtt{x} \mapsto 0], [\mathtt{n} \mapsto 0, \mathtt{x} \mapsto -1]\}$$

$$[\![c]\!]^{ov}(\mathtt{n} \geq 0) = \{\mathtt{n} \in \{0,1\}, \mathtt{x} \leq 0\}$$

$\in [\![c]\!]^{ov}(\mathtt{n} \geq 0)$ False Positive

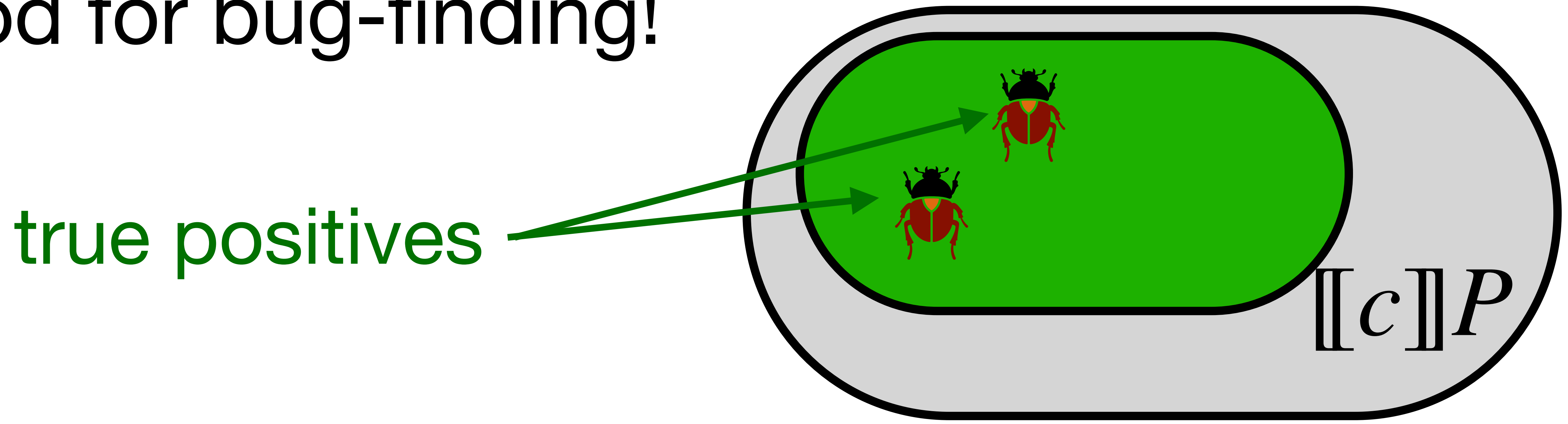
$\notin [\![c]\!](\mathtt{n} \geq 0)$

Under approximations

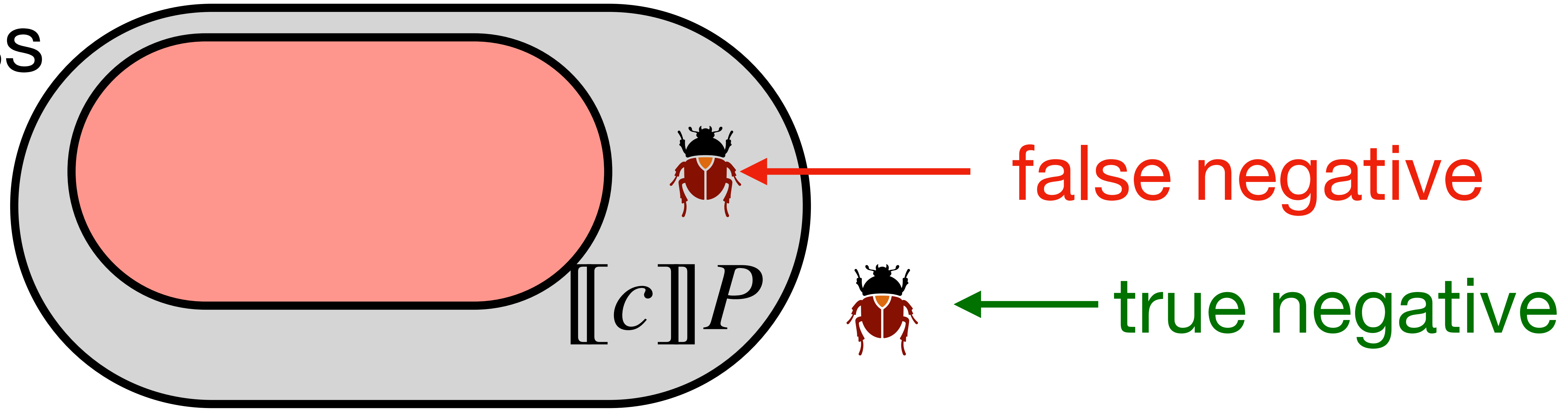


Under approximations

Good for bug-finding!



Bad for proving correctness



Example

$c \triangleq$

```
while (n>1) {
    n := n+1;
    x := 0;
}
x := n-1;
```

```
y := 1/(x);
```

Undefined behaviour for 🪲 x=0

$$[\![c]\!](\mathtt{n} \geq 0) = \{[\mathtt{n} \mapsto 1, \mathtt{x} \mapsto 0], [\mathtt{n} \mapsto 0, \mathtt{x} \mapsto -1]\}$$

$$[\![c]\!]^{un}(\mathtt{n} \geq 0) = \{[\mathtt{n} \mapsto 1, \mathtt{x} \mapsto 0]\}$$

$$🪲 \in [\![c]\!]^{un}(\mathtt{n} \geq 0) \implies 🪲 \in [\![c]\!](\mathtt{n} \geq 0)$$

We can prove there is an error !!

Example

$c \triangleq$

```
while (n>1) {
    n := n+1;
    x := 0;
}
x := n-1;
```

$$[\![c]\!](\mathtt{n} \geq 0) = \{[\mathtt{n} \mapsto 1, \mathtt{x} \mapsto 0], [\mathtt{n} \mapsto 0, \mathtt{x} \mapsto -1]\}$$

$$[\![c]\!]^{un}(\mathtt{n} \geq 0) = \{[\mathtt{n} \mapsto 1, \mathtt{x} \mapsto 0]\}$$

```
y := 1/(x+1);
```

Undefined behaviour for x=-1

$\in [\![c]\!](\mathtt{n} \geq 0)$

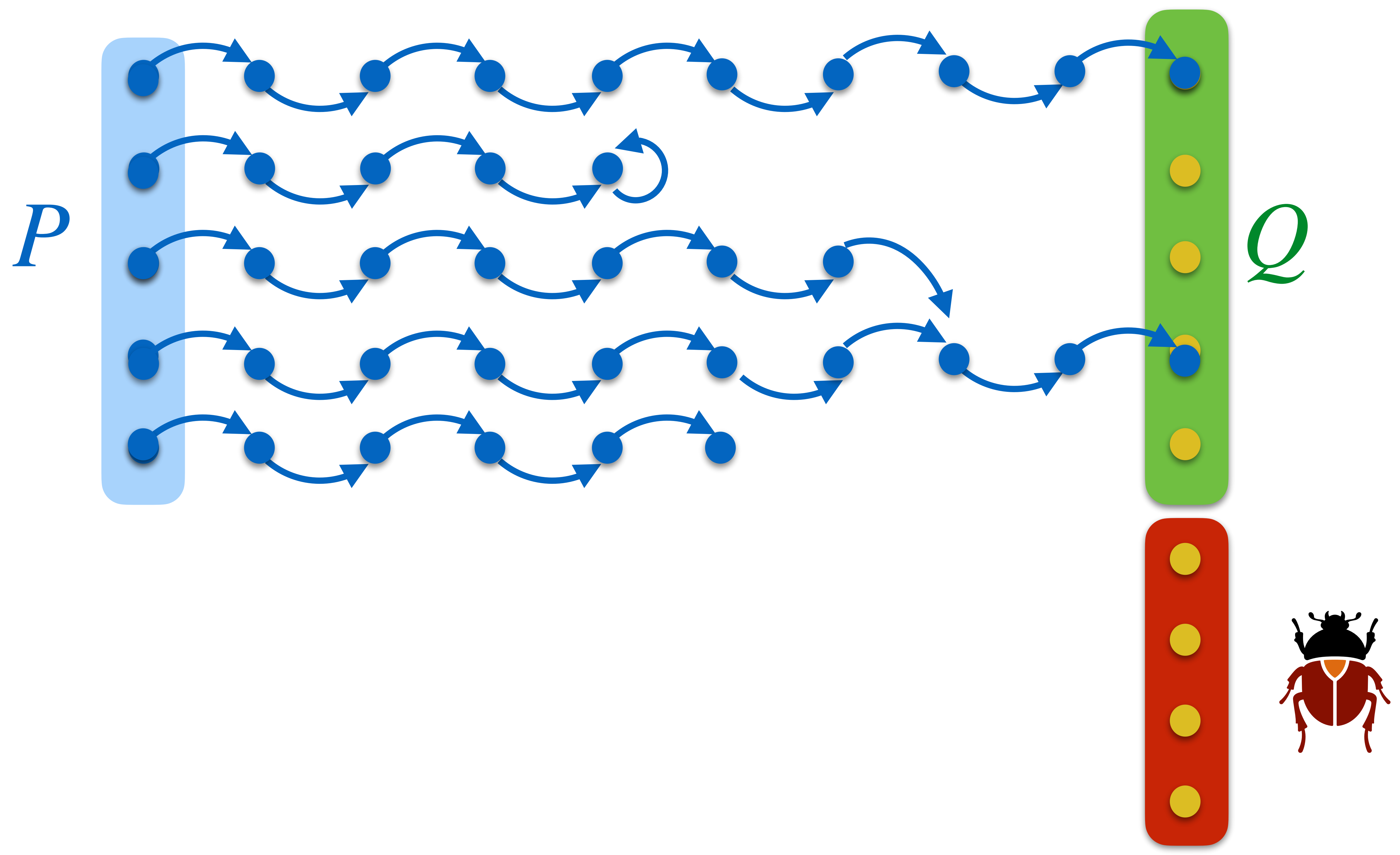
$\notin [\![c]\!]^{un}(\mathtt{n} \geq 0)$ False Negative

Proving Correctness: forward

A program

c

$[\![\text{c}]\!]P \subseteq Q$



$\forall \sigma \in P \,.\, [\![c]\!]\sigma$ either does not terminate or terminates in Q

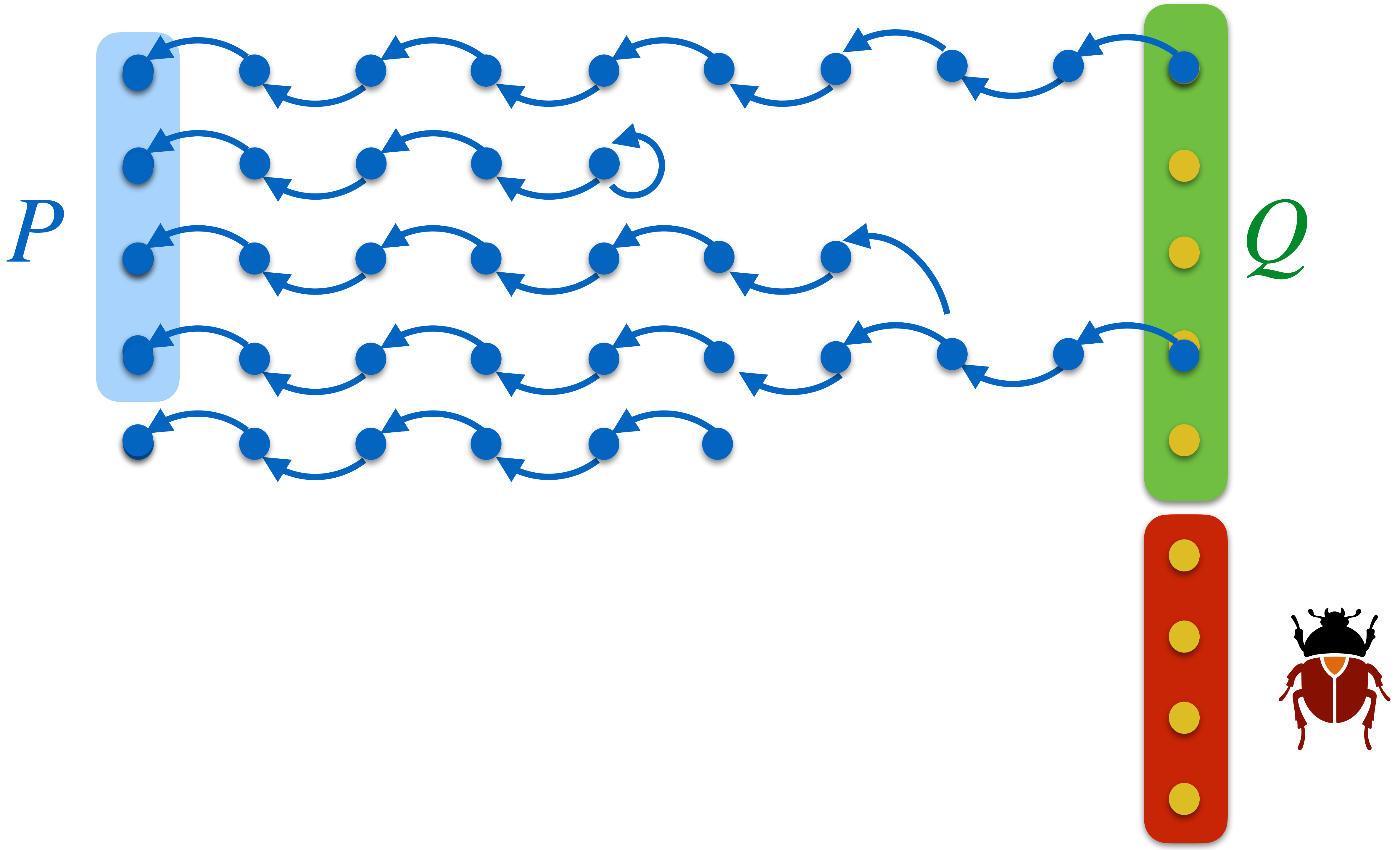
Proving Correctness: backward

$P \subseteq wlp(c, Q)$

$[\![c]\!]P \subseteq Q$

A program

c



Dijkstra's weakest liberal precondition

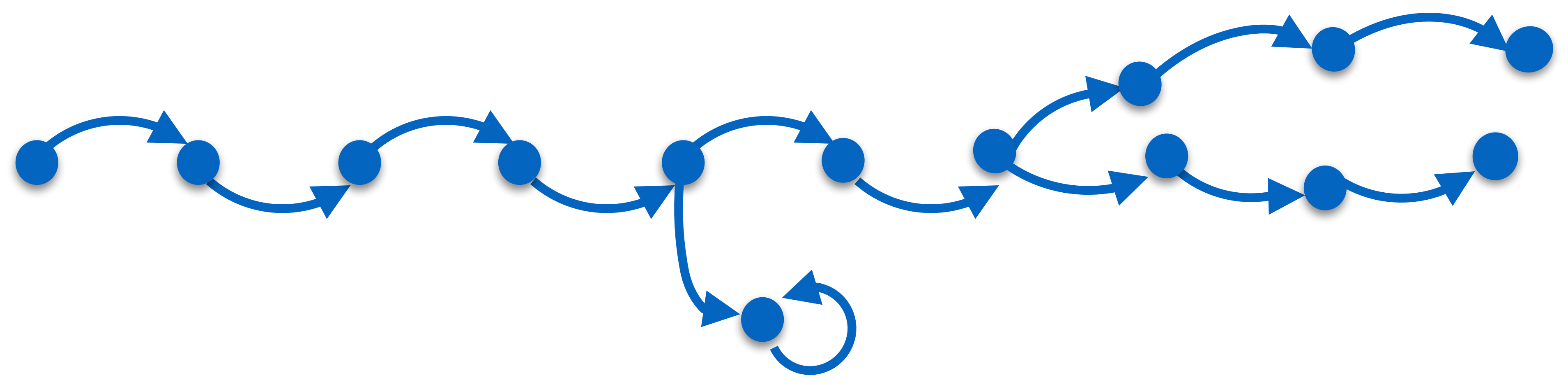
$$wlp(c, Q) = \{\sigma \mid [\![c]\!]\{\sigma\} \subseteq Q\}$$

Nondeterministic programs

Some programs may exhibit nondeterministic behaviour
(lack of information, approximation, programming constructs $c_1 + c_2$)

A program c

σ



$[\![c]\!] : \Sigma \to \wp(\Sigma)$

$[\![c]\!]P \subseteq Q$

$P \subseteq wlp(c, Q)$

all the outputs starting from $\sigma \in P$ (upon termination) are in Q

Example

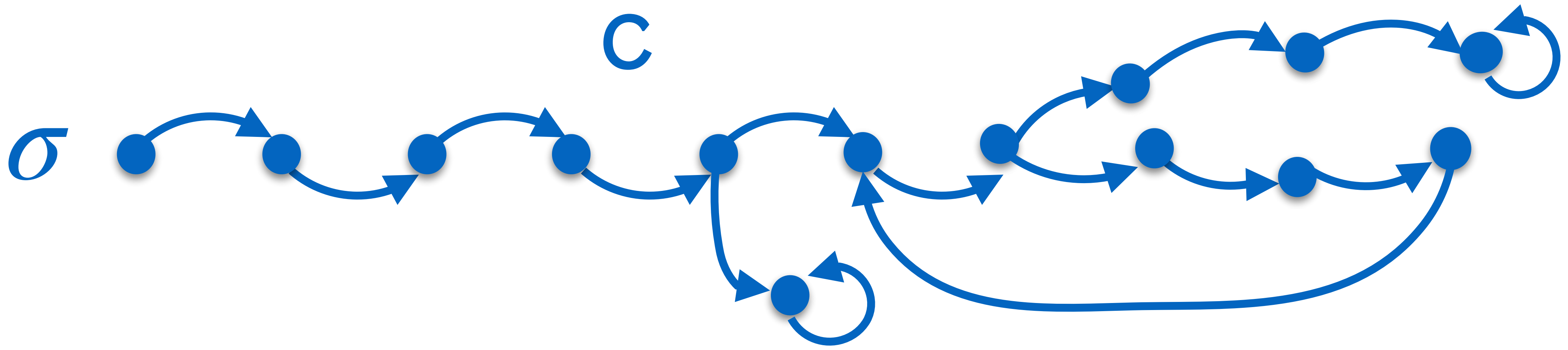
$c \triangleq$

```
Divisor_of(x) {
  …
  s := nondet[2..x/2];
  if (x%s=0)
    skip
  else
    while true do skip
}
```

$[\![c]\!][\mathrm{x} \mapsto 35] = (\mathrm{x} = 35, \mathrm{s} \in \{5{,}7\})$

An example: non-termination analysis

Given a program c and an input store σ does $[\![c]\!]\sigma = \varnothing$?



Non termination

Using over-approximation: we try to prove $[\![c]\!]^{ov}\sigma \subseteq \varnothing$

Termination

Using under-approximation: we try to prove $[\![c]\!]^{un}\sigma \supseteq Q$ for some $Q \neq \varnothing$

What we will see

	Forward	Backward	Over-approximation	Under-approximation
Hoare Logic (HL)	X		X	
Incorrectness Logic (IL)	X			X
Necessary Condition (NC)		X	X	
Sufficient Incorrectness Logic (SIL)		X		X
Separation logic (SL)	X		X	
Incorrectness Separation Logic (ISL)	X			X
Separation SIL		X		X