An intro to SAT and SMT: a user's perspective

Elizabeth Polgreen

By the end of these lectures you will know ...

- What the satisfiability problem is
- How to encode problems into SAT/SMT
- Which SAT/SMT solvers are available and how to use them, and some useful tools to generate SAT/SMT queries.
- How SAT/SMT solvers are deployed in the wild

You will not know ...

- The detail of how SAT/SMT solvers work

By the end of these lectures you will know ...

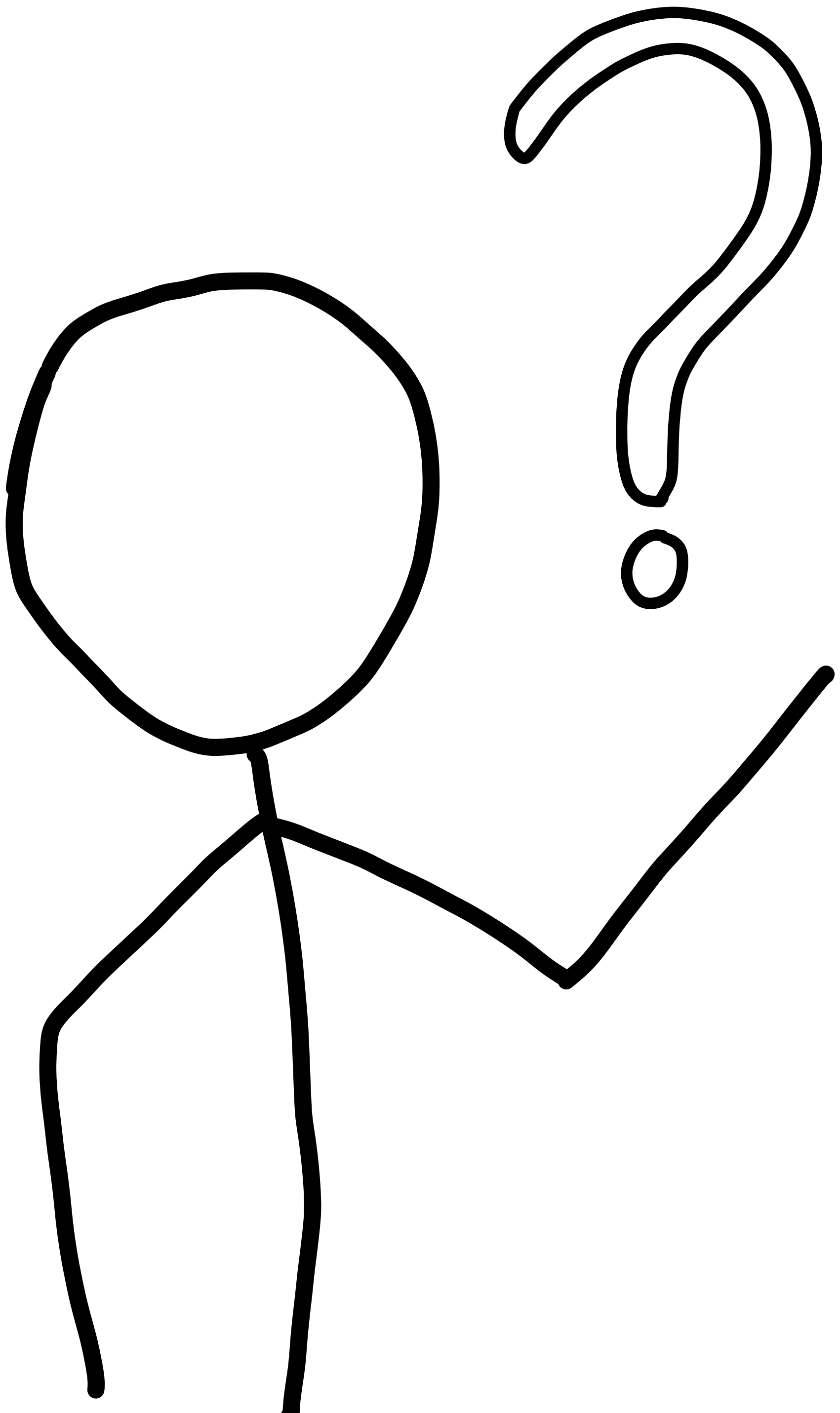
- What the satisfiability problem is
- How to encode problems into SAT/SMT
- Which SAT/SMT solvers are available and how to use them
- How SAT/SMT solvers are deployed in the wild

Bonus: how to synthesise programs, using SMT solvers

You will not know ...

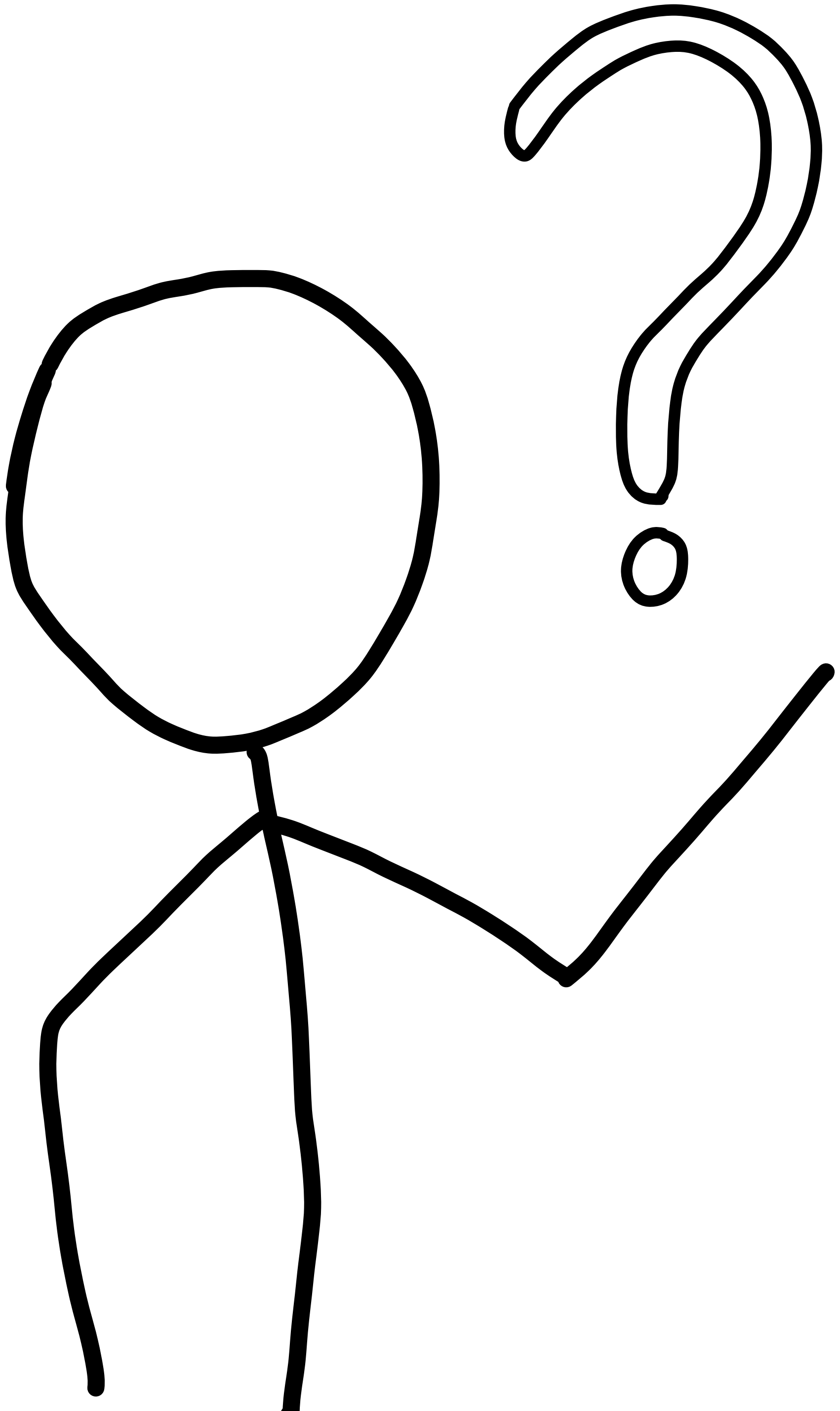
- The detail of how SAT/SMT solvers work

Who am I?



- Lecturer in LFCS at Edinburgh
- A big user of SAT and SMT solving
- Developer of synthesis algorithms

Who am I?



UCLID5: Modelling, verification and synthesis

CBMC: bounded model checking for C programs

SAT

A : **Boolean**

B : **Boolean**

$$\exists A, B$$
$$A \wedge \neg B$$

A : **true**

B : **false**

SAT

A : Boolean

B : Boolean

$\exists A, B$

$A \land \neg B$

A : true

B : false

SMT

A : Integer

B : Integer

$\exists A, B$

$A > 0 \land B < 0$

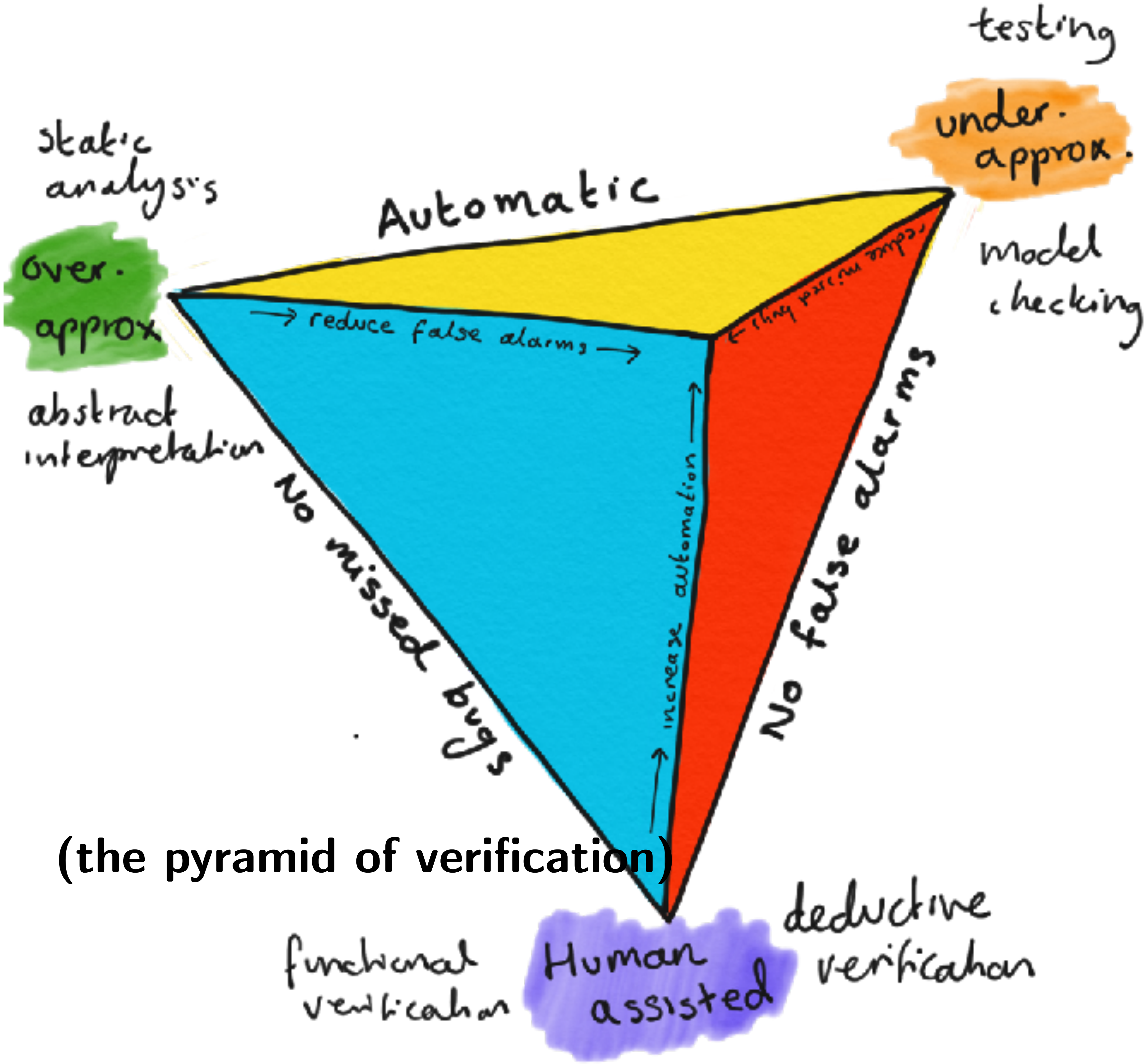
A : 10

B : -3

Why are SAT/SMT important?

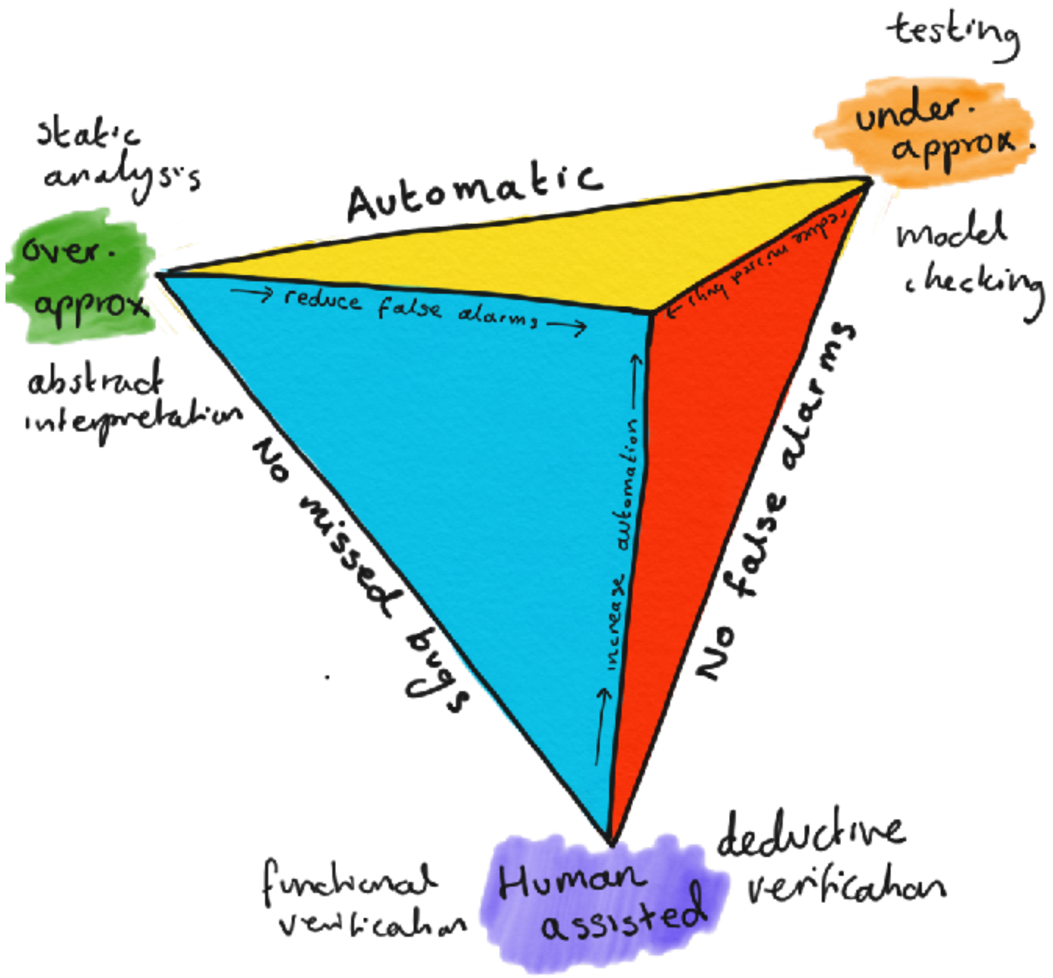
They are used in so many different verification tools!

Why are SAT/SMT important?



(the pyramid of verification)

(The pyramid of verification)

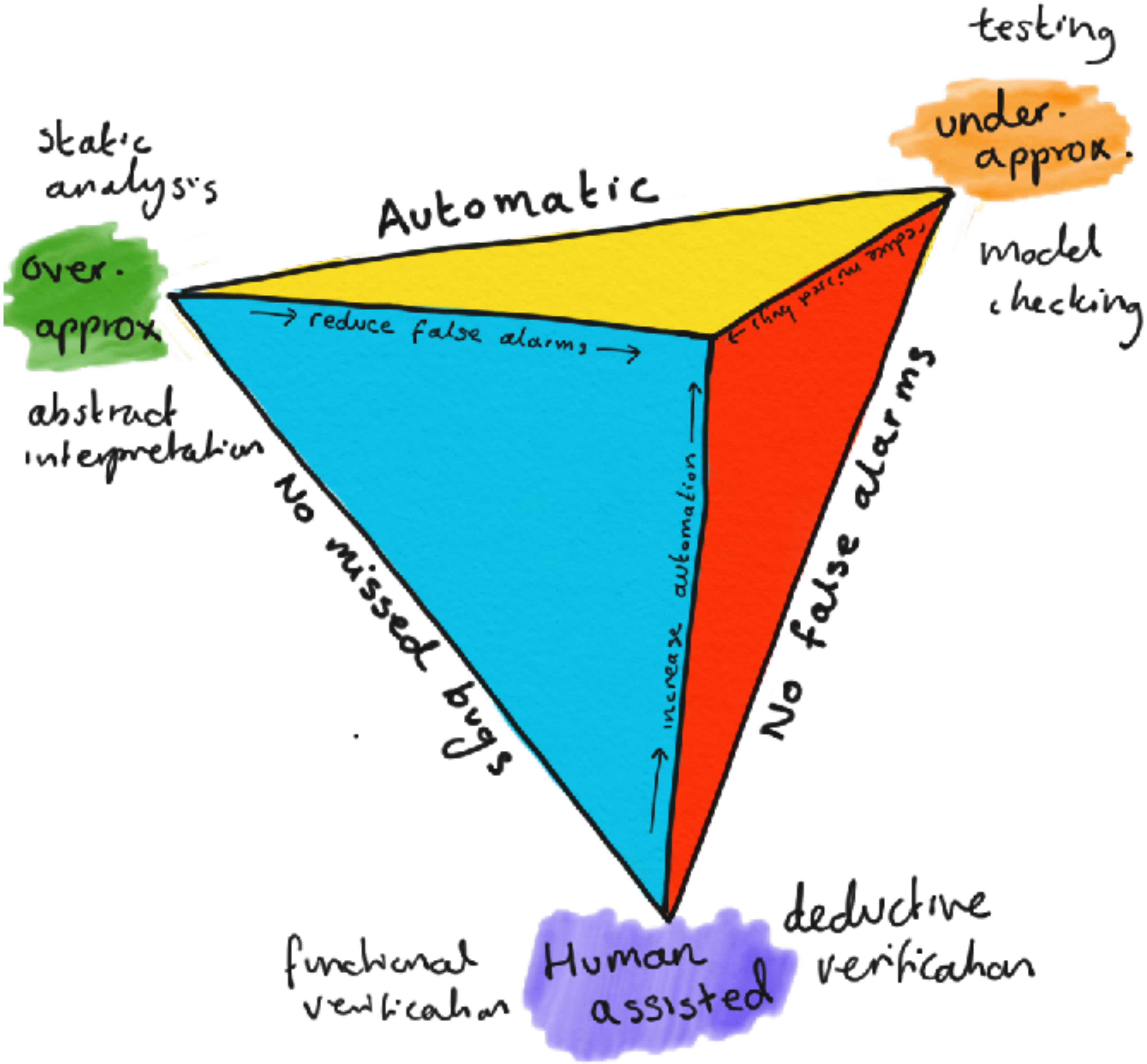


The perfect tool:

- Never misses a bug,
- Never gives a false alarm,
- And is fully automated for all specifications and all programs

This is impossible!

Why are SAT/SMT important?



Propositional SAT

A propositional formula is composed from Boolean variables and the operators:

- $\neg$ (negation, ‘not’, sometimes written ‘!’)
- $\vee$ (disjunction, ‘or’, sometimes written ‘||’)
- $\wedge$ (conjunction, ‘and’, sometimes written ‘&&’)
- $\rightarrow$ (implication, $p \rightarrow q \equiv \neg p \vee q$)

Propositional SAT

A formula is:

- Satisfiable if there exist values to the variables such that the formula evaluates to true
- Unsatisfiable if the formula evaluates to false **for all** assignments to the variables
- Valid if the formula evaluates to true **for all** assignments to the variables

Propositional SAT

Given a propositional formula $F(x_1, x_2, x_3, \ldots x_n)$

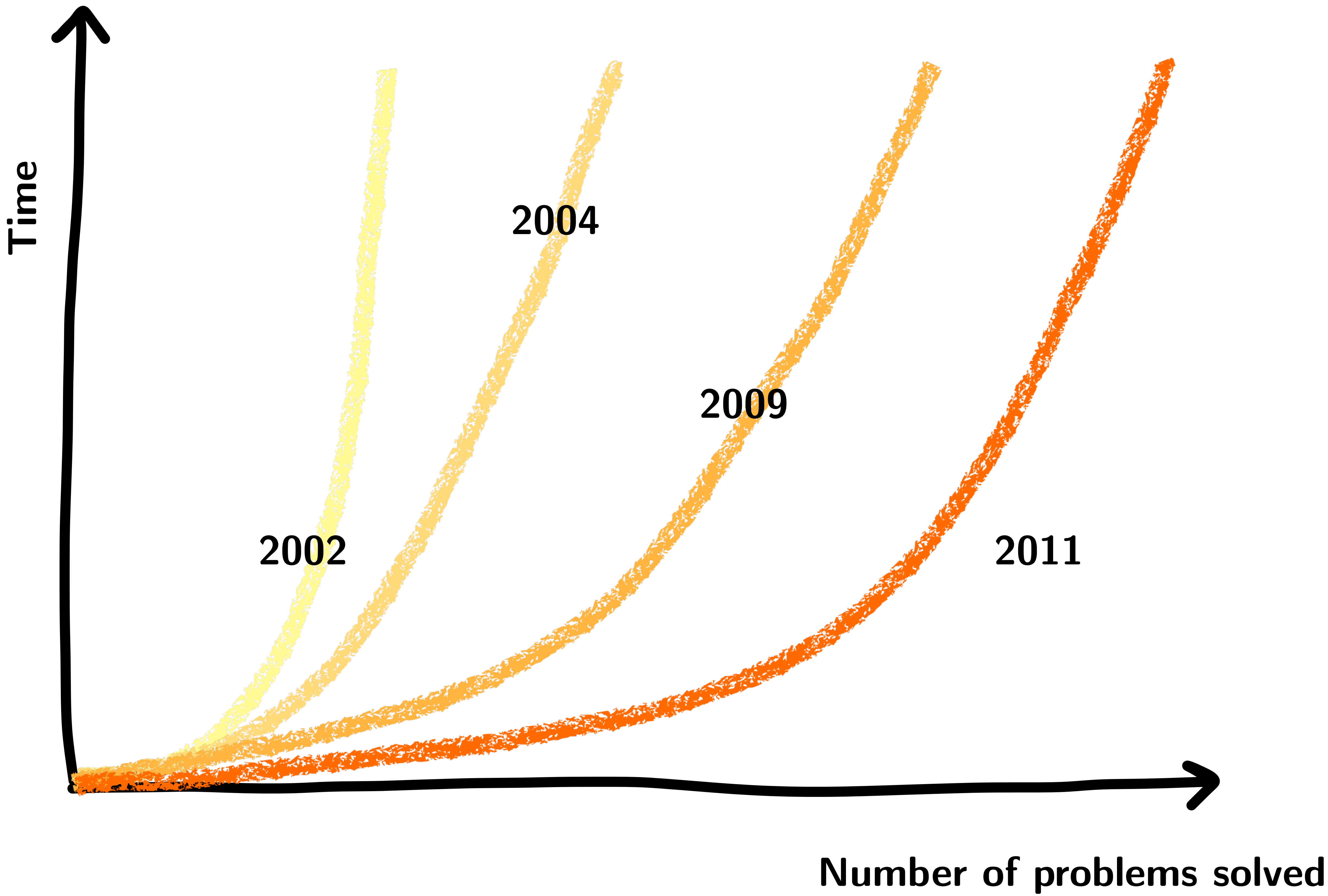
Is F satisfiable?

If yes, return the values to $x_1 \ldots x_n$ that make F true

Complexity

- SAT is the canonical NP-complete problem (no polynomial time algorithm)
- If you can reduce a problem to SAT, the problem is in NP
- Including Mario..

Progress of SAT solvers in SAT competition



Example 1

```
if(!a && !b) h();
else
    if(!a) g();
    else f();
```

```
if(a) f();
else
    if(b) g();
    else h();
```

Are these two code fragments the same?

Example 1

```
if(!a && !b) h();
else
    if(!a) g();
    else f();
```

```
if(a) f();
else
    if(b) g();
    else h();
```

Are these two code fragments the same?

```
if $\neg a \land \neg b$ then $h$
else
    if $\neg a$ then $g$
    else $f$
```

```
if $a$ then $f$
else
    if $b$ then $g$
    else $h$
```

Example 1

```
if(!a && !b) h();
else
    if(!a) g();
    else f();
```

```
if(a) f();
else
    if(b) g();
    else h();
```

Are these two code fragments the same?

```
if $\neg a \wedge \neg b$ then $h$
else
   if $\neg a$ then $g$
   else $f$
```

```
if $a$ then $f$
else
   if $b$ then $g$
   else $h$
```

Is this formula valid:

$$\boxed{\begin{aligned} & (\neg a \wedge \neg b) \wedge h \ \vee \ \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \ \vee \ a \wedge f) \\ \Longleftrightarrow \quad & a \wedge f \ \vee \ \neg a \wedge (b \wedge g \ \vee \ \neg b \wedge h) \ . \end{aligned}}$$

$$\boxed{\begin{array}{ll} & (\neg a \wedge \neg b) \wedge h \;\vee\; \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \;\vee\; a \wedge f) \\ \iff & a \wedge f \;\vee\; \neg a \wedge (b \wedge g \;\vee\; \neg b \wedge h)\;. \end{array}}$$

```
if(!a && !b) h();
else
    if(!a) g();
    else f();
```

```
if(a) f();
else
    if(b) g();
    else h();
```

Look for a counterexample?

```
if $\neg a \wedge \neg b$ then $h$
else
    if $\neg a$ then $g$
    else $f$
```

```
if $a$ then $f$
else
    if $b$ then $g$
    else $h$
```

$$\boxed{\begin{aligned} & (\neg a \wedge \neg b) \wedge h \;\vee\; \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \;\vee\; a \wedge f) \\ \iff \quad & a \wedge f \;\vee\; \neg a \wedge (b \wedge g \;\vee\; \neg b \wedge h)\,. \end{aligned}}$$

```
if(!a && !b) h();
else
    if(!a) g();
    else f();
```

```
if(a) f();
else
    if(b) g();
    else h();
```

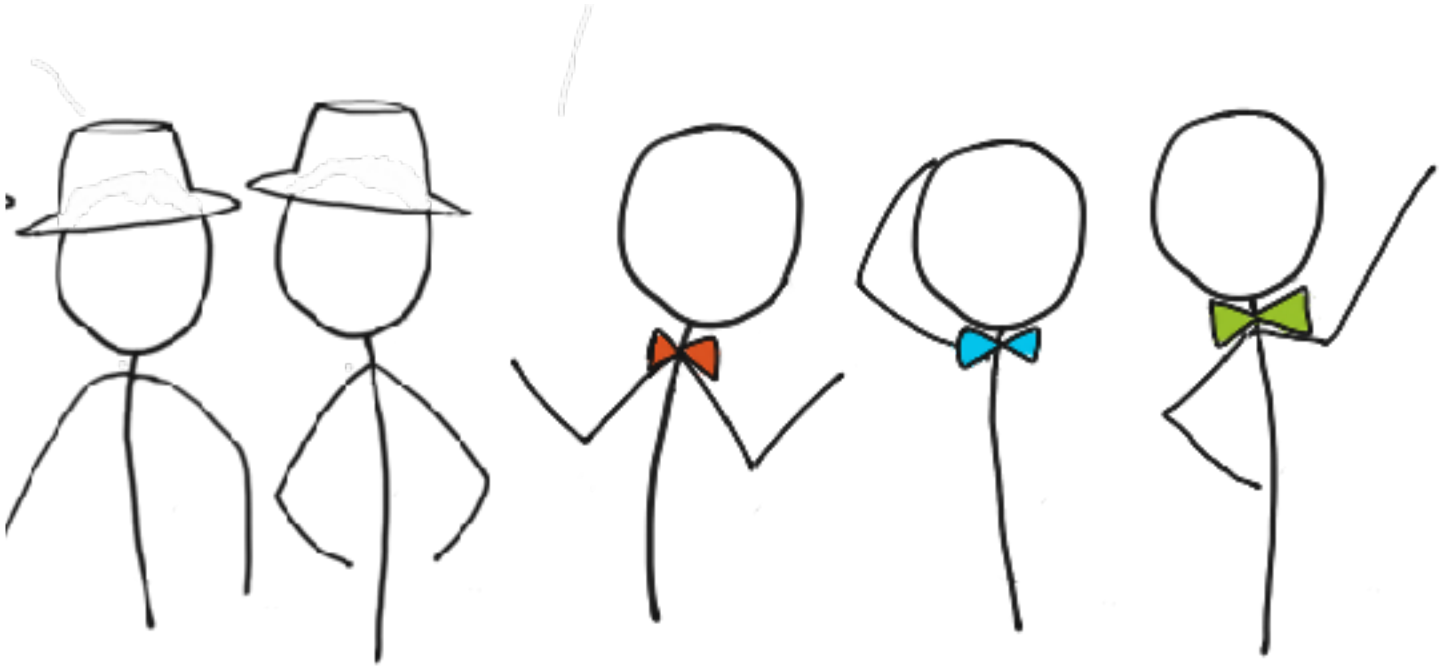
Look for a counterexample?

```
if $\neg a \wedge \neg b$ then $h$
else
    if $\neg a$ then $g$
    else $f$
```

```
if $a$ then $f$
else
    if $b$ then $g$
    else $h$
```

$$\boxed{\begin{aligned} & (\neg a \wedge \neg b) \wedge h \;\vee\; \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \;\vee\; a \wedge f) \\ \oplus \quad & a \wedge f \;\vee\; \neg a \wedge (b \wedge g \;\vee\; \neg b \wedge h)\,. \end{aligned}}$$

Example 2



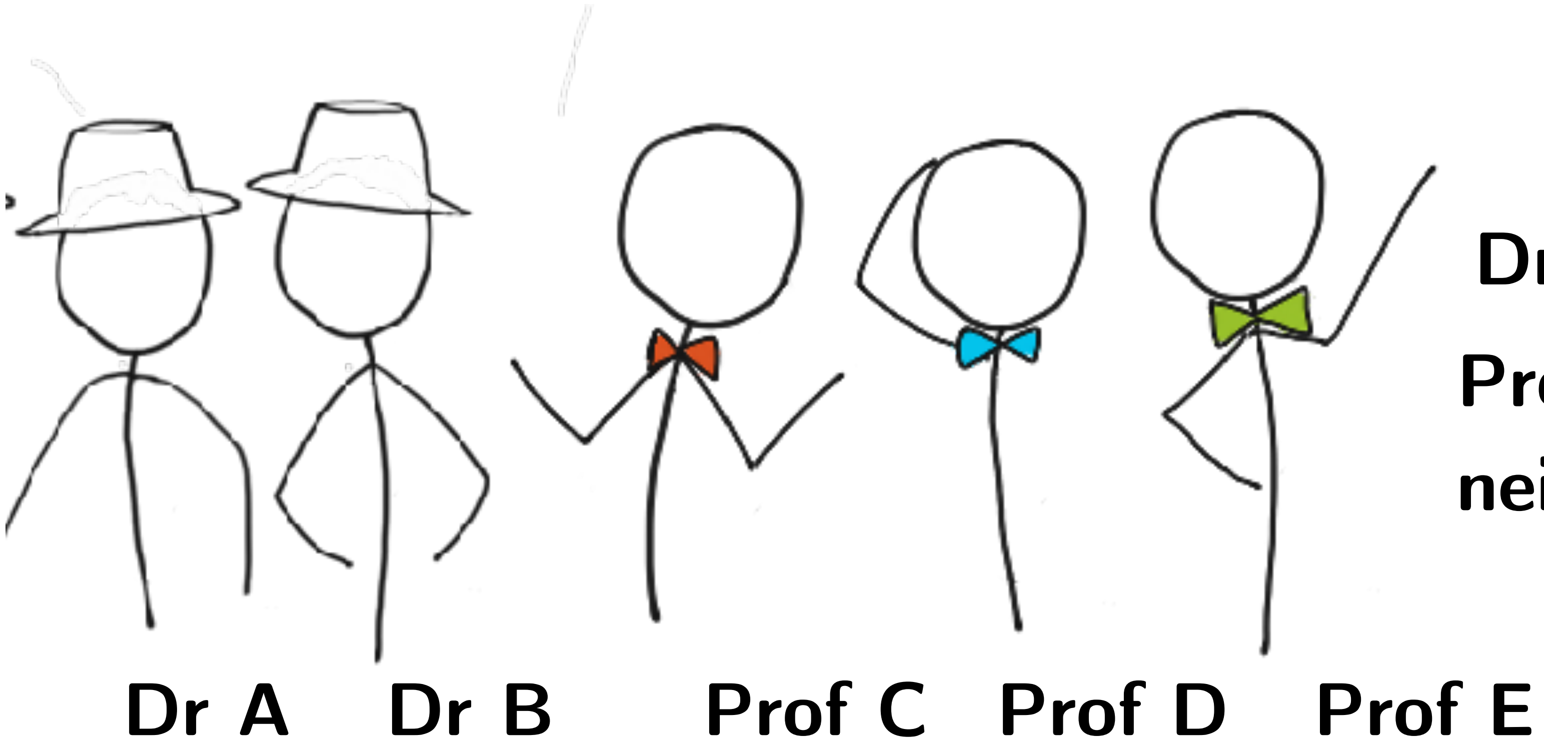
Dr A Dr B Prof C Prof D Prof E

Dr A hates everyone except Dr B

Professor D only wants to be neighbours with other professors

R1 R2 R3 R4 R5

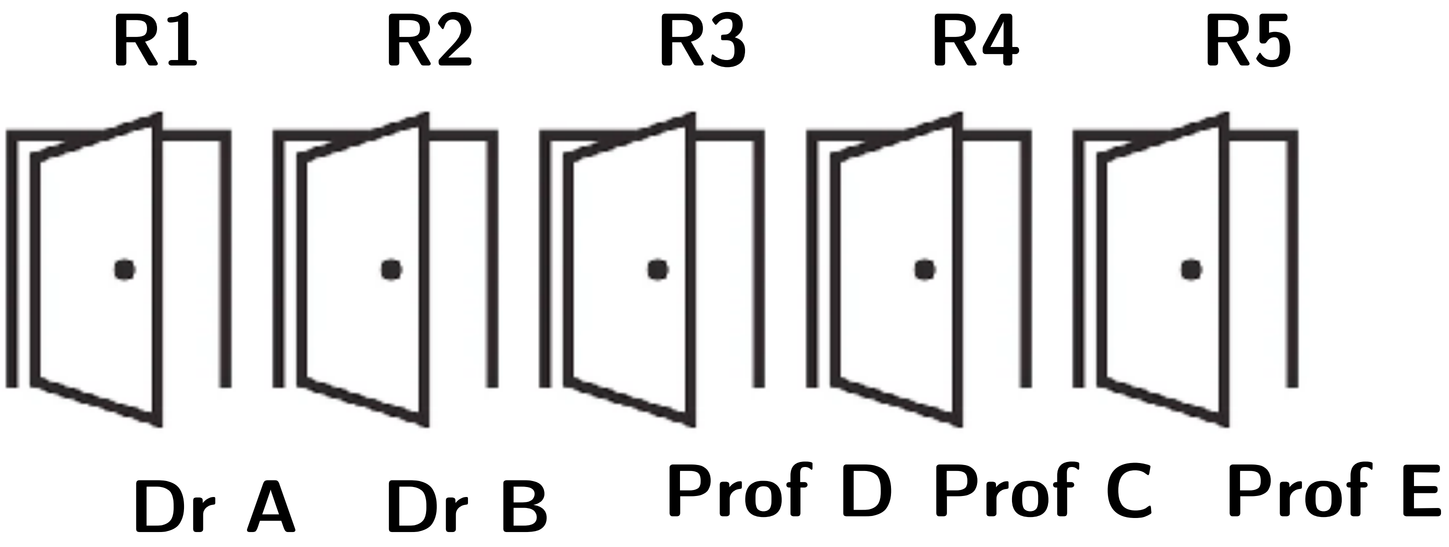
Example 2



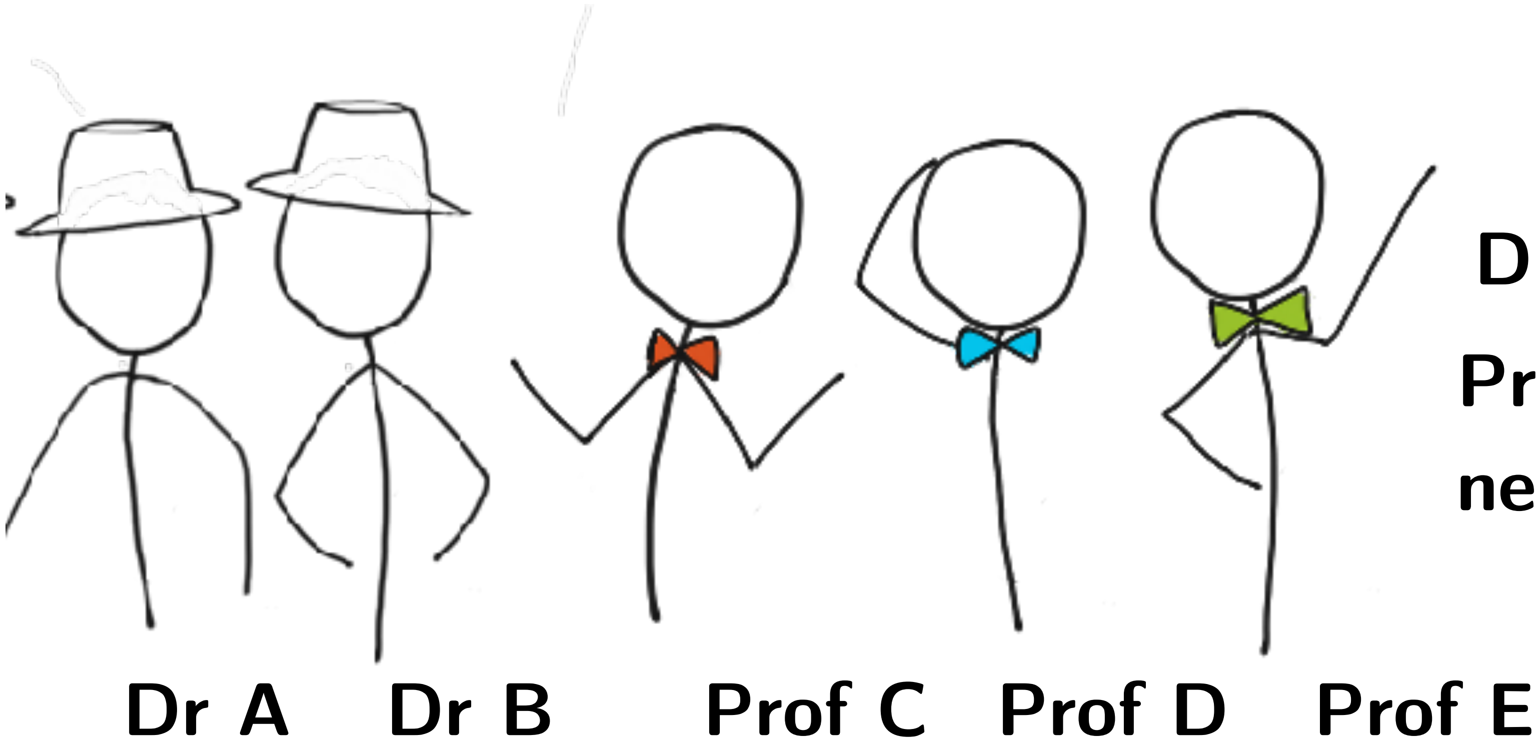
Dr A hates everyone except Dr B

Professor D only wants to be neighbours with other professors

SAT



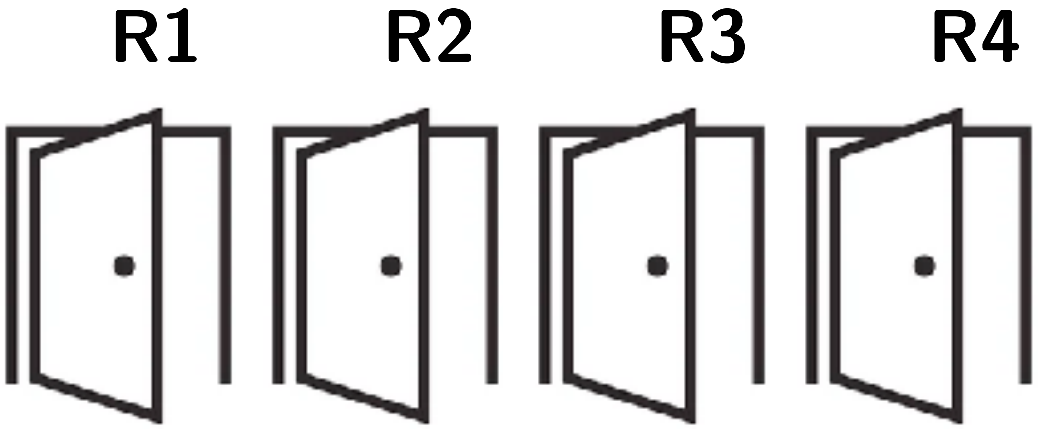
Example 2



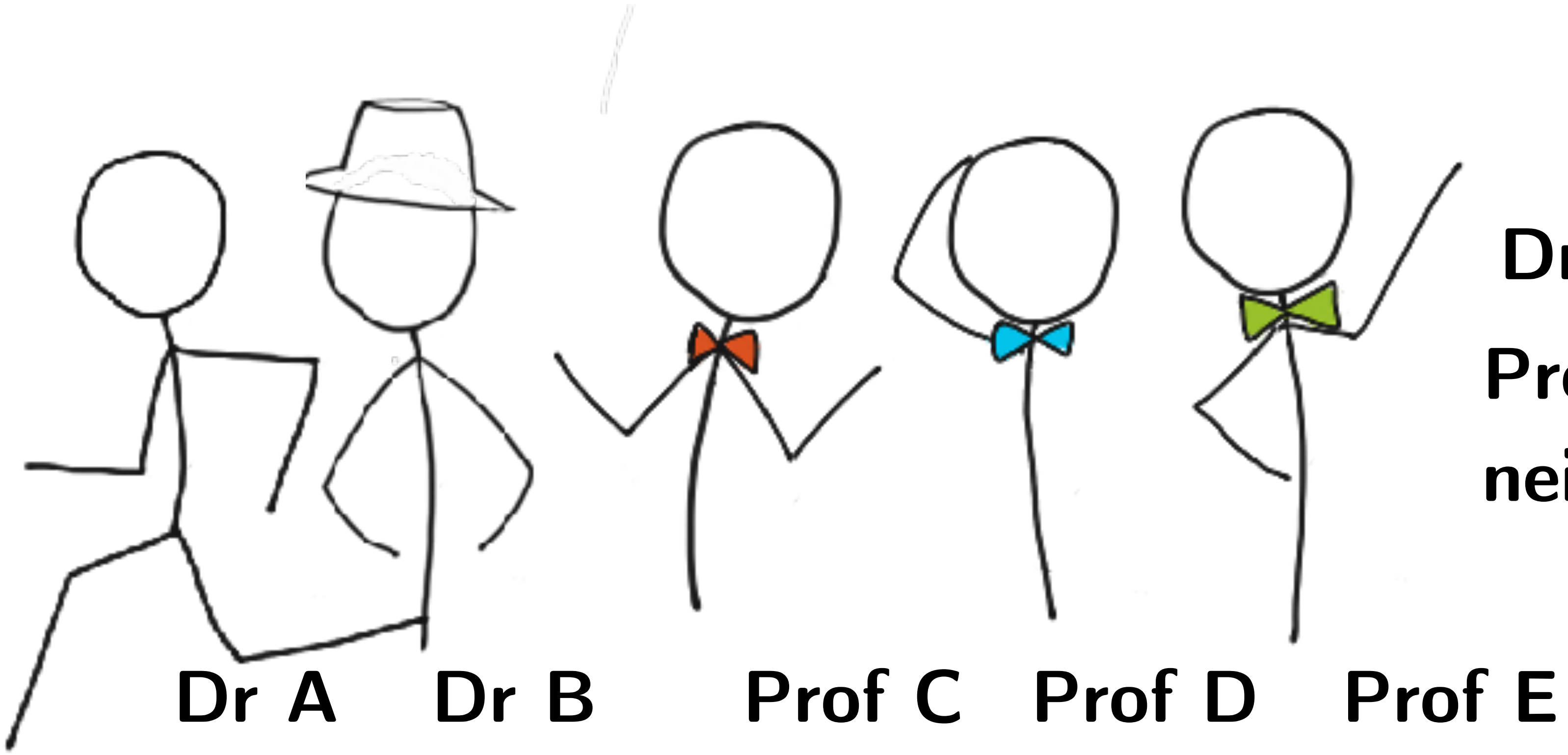
Dr A hates everyone except Dr B

Professor D only wants to be neighbours with other professors

UNSAT



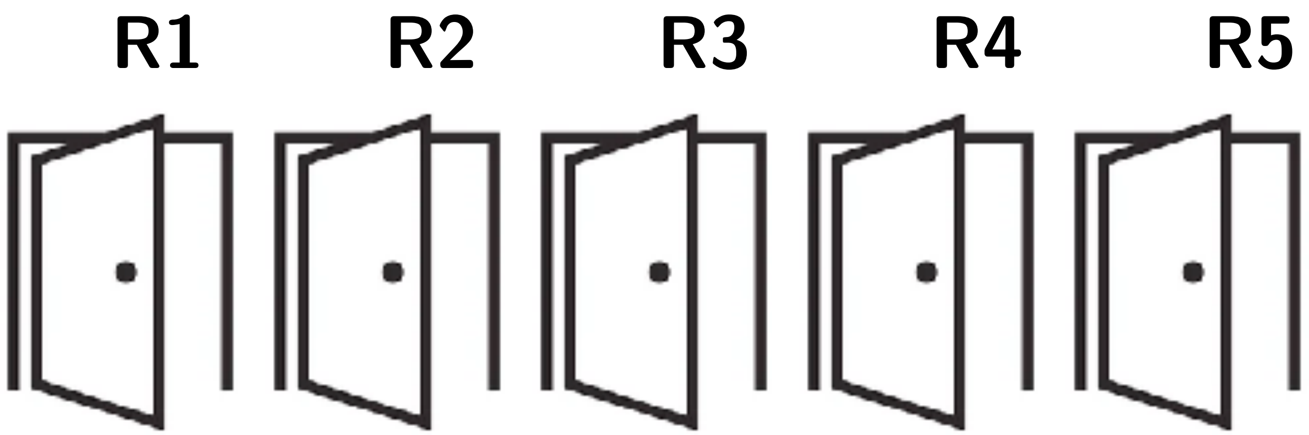
Example 2



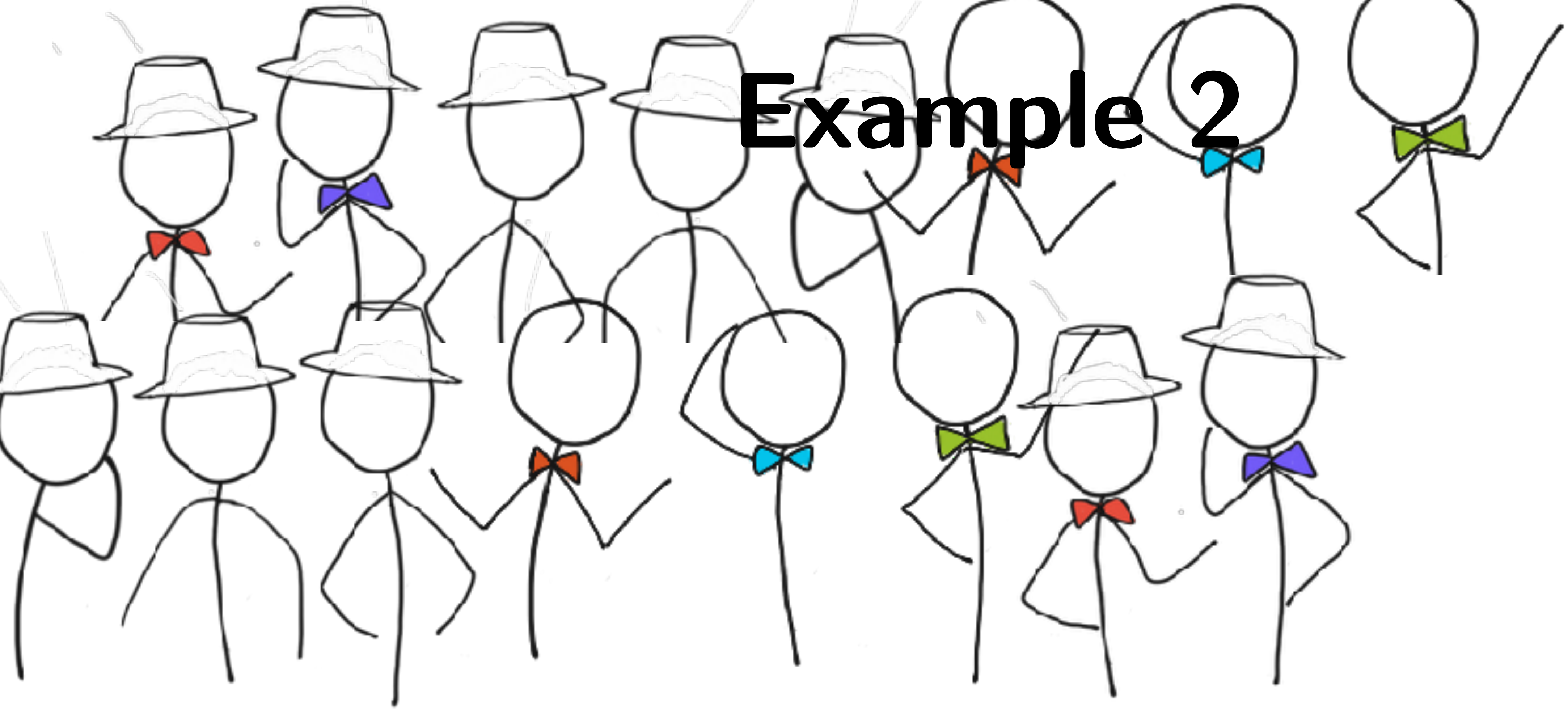
Dr A hates everyone ~~except Dr B~~

Professor D only wants to be neighbours with other professors

UNSAT



Example 2



Dr A hates ev

Dr B will only

Professor C on

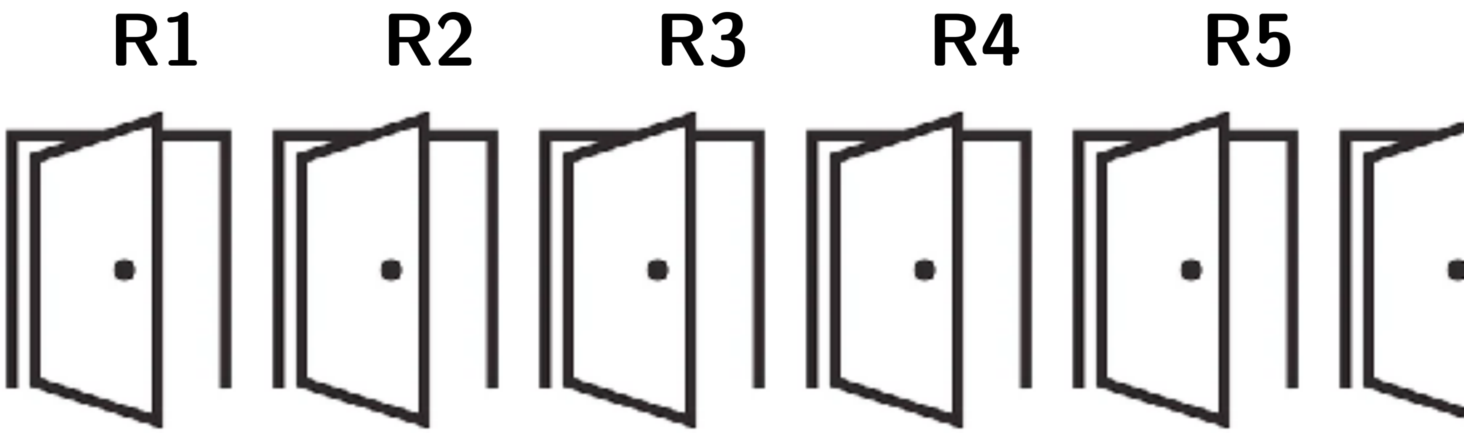
neighbours wit

....

Dr X hates ev

Dr Y only wan

?



Example 2

$A = \{a_1, \ldots a_n\}$ is a set of academics.

$R = \{r_1 \ldots r_k\}$ is a set of offices.

E is a set of pairs of academics who refuse to be office neighbours with each other.

Find an allocation of offices so that all the academics are happy.

Example 2

Define $X = \{x_{ij} \mid i \in \{1,\ldots n\}, j \in \{1,\ldots k\}\}$**.**

x_{ij} **is true iff academic** a_i **is allocated to office** r_j**.**

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{k} x_{ij}$$

every academic is allocated at least one office

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{k-1} (x_{ij} \implies \bigwedge_{j<t\leq k} \neg x_{it})$$

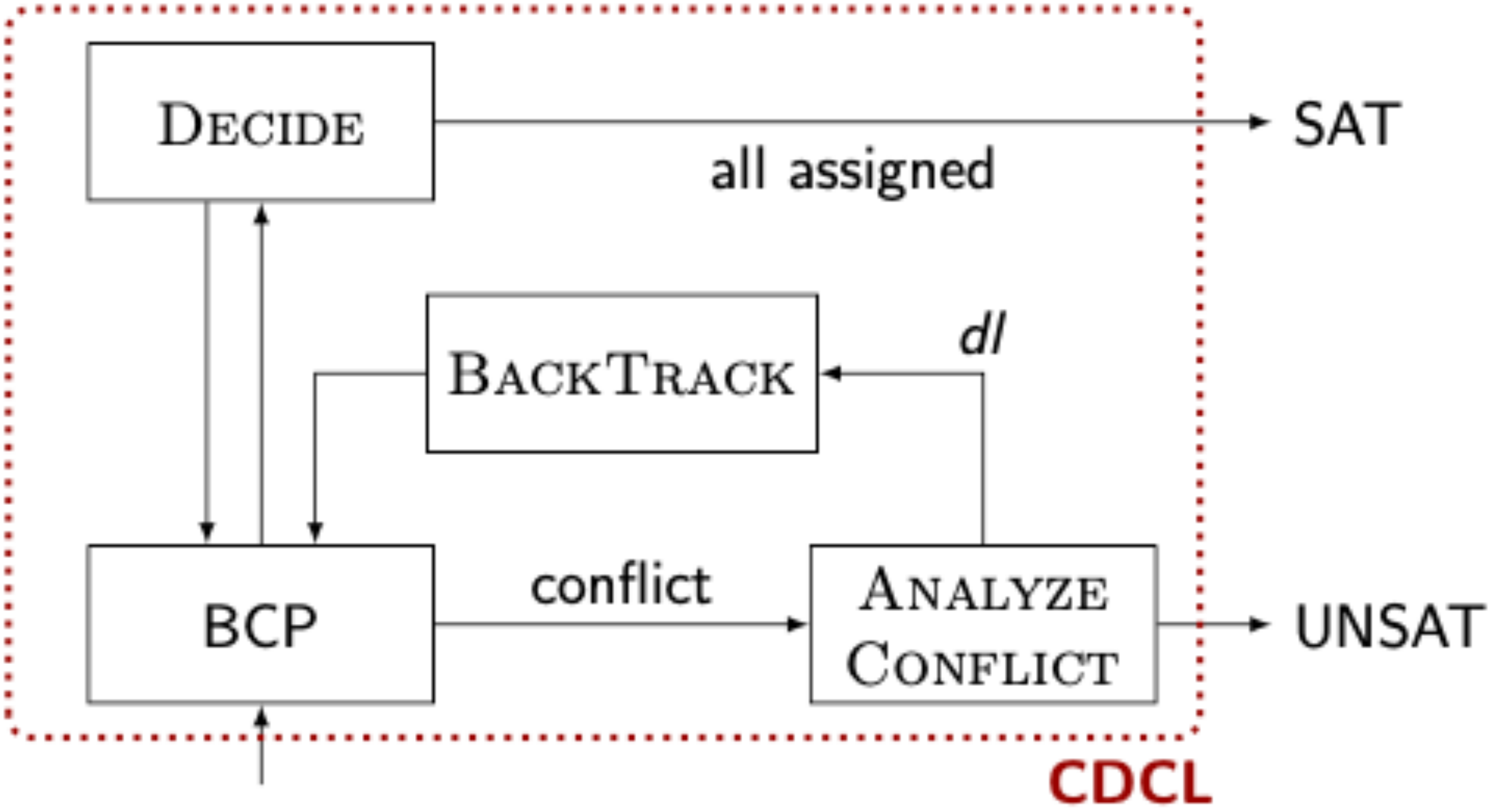
but no more than one!

For each $(i,j) \in E$, $\bigwedge_{t=1}^{n} (x_{it} \implies (\neg x_{jt+1} \wedge \neg x_{jt-1}))$

And even professor D's preferences are taken into account

CDCL (high-level)

SAT solvers use Conflict Driven Clause Learning.
We won't cover this in detail here.



How do we use a SAT solver?

- Formula needs to be in CNF
- Formula is translated to DIMACS

CNF

SAT solvers need formula in Conjunctive Normal Form (CNF), i.e., a disjunction of literals

Terminology:

- An atom p is a propositional symbol
- A literal l is an atom p or its negation $\neg p$
- A clause C is a disjunction of literals $l_1 \vee \ldots \vee l_n$
- A CNF formula is a conjunction of clauses $C_1 \wedge \ldots \wedge C_m$

Tseitsin Transformation

Translates a formula into an equisatisfiable CNF formula in linear time by :

- introducing a fresh variable for every non-atomic sub-formula

- Add a constraint that gives equivalence of new variable with subformula

Tseitsin Transformation

Translates a formula into an equisatisfiable CNF formula in linear time by :

- introducing a fresh variable for every non-atomic sub-formula
- Add a constraint that gives equivalence of new variable with subformula

Transformation rules for three basic operators

formula	$p \leftrightarrow$ formula	rewritten in CNF
$\neg A$	$(\neg A \rightarrow p) \wedge (p \rightarrow \neg A)$	$(A \vee p) \wedge (\neg A \vee \neg p)$
$A \wedge B$	$(A \wedge B \rightarrow p) \wedge (p \rightarrow A \wedge B)$	$(\neg A \vee \neg B \vee p) \wedge (A \vee \neg p) \wedge (B \vee \neg p)$
$A \vee B$	$(p \rightarrow A \vee B) \wedge (A \vee B \rightarrow p)$	$(A \vee B \vee \neg p) \wedge (\neg A \vee p) \wedge (\neg B \vee p)$

Tseitsin Transformation

Translates a formula into an equisatisfiable CNF formula in linear time by :

- introducing a fresh variable for every non-atomic sub-formula

- Add a constraint that gives equivalence of new variable with subformula

Transformation rules for three basic operators

formula	$p \leftrightarrow$ formula	rewritten in CNF
$\neg A$	$(\neg A \to p) \wedge (p \to \neg A)$	$(A \vee p) \wedge (\neg A \vee \neg p)$
$A \wedge B$	$(A \wedge B \to p) \wedge (p \to A \wedge B)$	$(\neg A \vee \neg B \vee p) \wedge (A \vee \neg p) \wedge (B \vee \neg p)$
$A \vee B$	$(p \to A \vee B) \wedge (A \vee B \to p)$	$(A \vee B \vee \neg p) \wedge (\neg A \vee p) \wedge (\neg B \vee p)$

Dimacs

- header line: p cnf <variables> <clauses>,
  where <variables> <clauses> are decimal numbers for the number of variables and clauses in the formula respectively

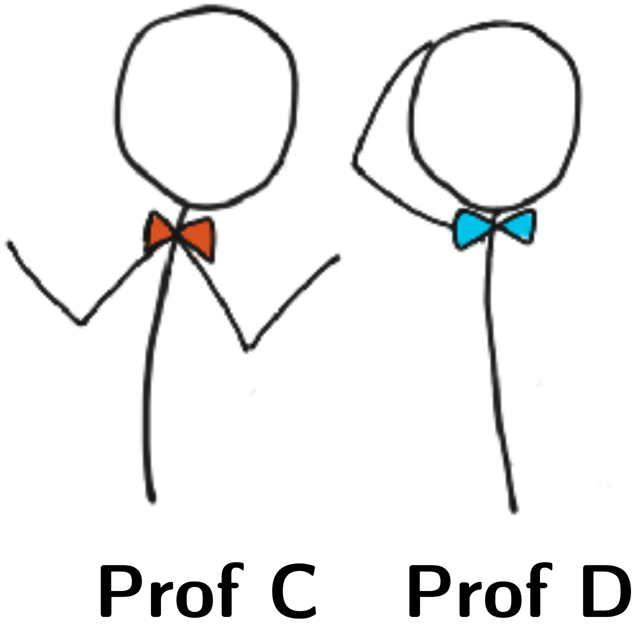
- one clause per line of file with a 0 at the end:

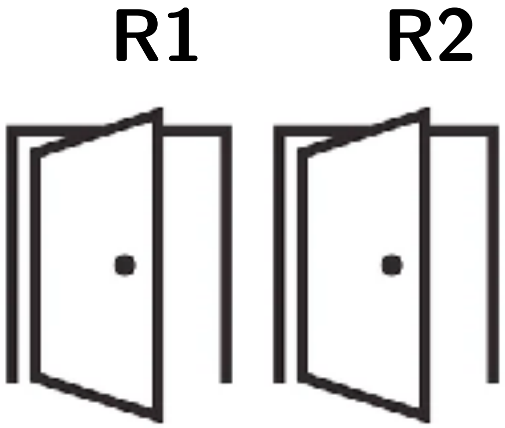
- each variable has an decimal number, and – indicates the negation of that variable.

$(x \vee y \vee \neg z) \wedge (\neg y \vee z)$

```
p cnf 3 2
1 2 -3 0
-2 3 0
```

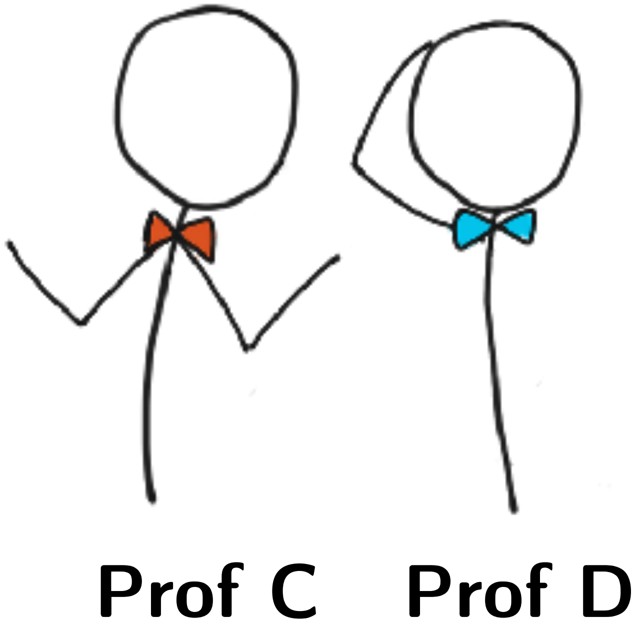
Example 2



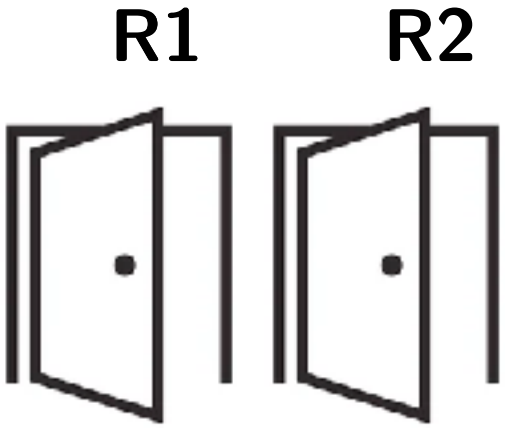


$$(x_{11} \vee x_{12}) \wedge (x_{21} \vee x_{22}) \wedge$$

$$(x_{11} \rightarrow \neg x_{12}) \wedge (x_{12} \rightarrow \neg x_{11}) \wedge (x_{21} \rightarrow \neg x_{22}) \wedge (x_{22} \rightarrow \neg x_{21})$$



Example 2



$$(x_{11} \lor x_{12}) \land (x_{21} \lor x_{22}) \land$$
$$(x_{11} \to \neg x_{12}) \land (x_{12} \to \neg x_{11}) \land (x_{21} \to \neg x_{22}) \land (x_{22} \to \neg x_{21})$$

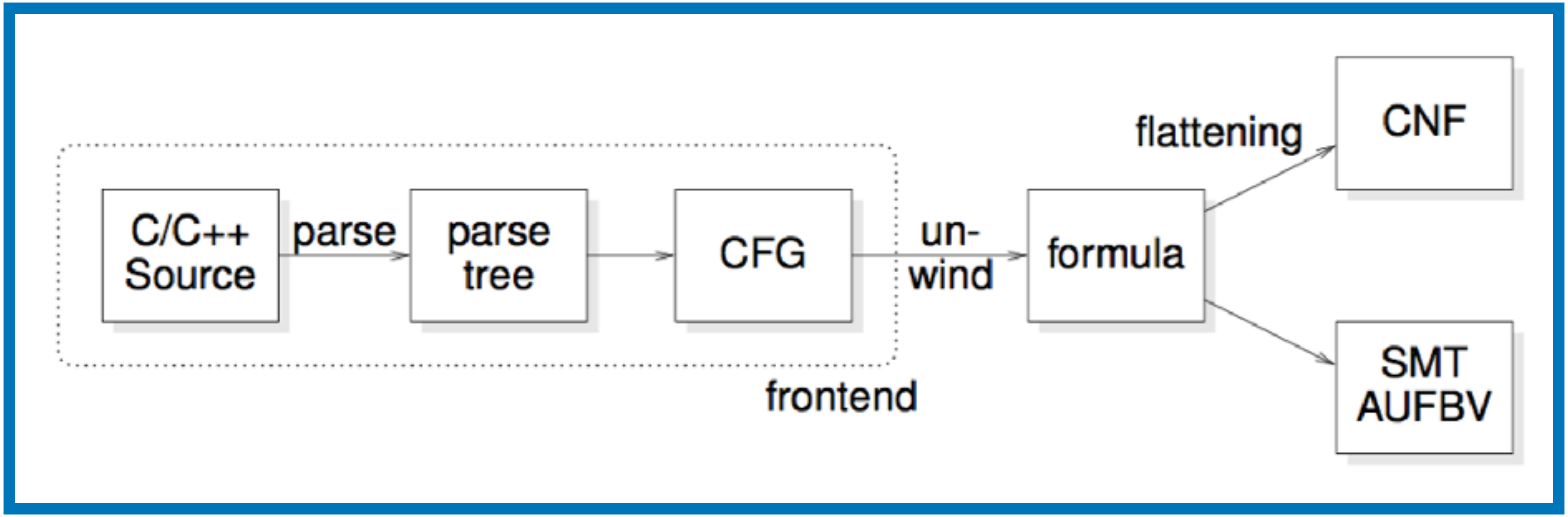
$$(x_{11} \lor x_{12}) \land (x_{21} \lor x_{22}) \land$$
$$(\neg x_{11} \lor \neg x_{12}) \land (\neg x_{12} \lor \neg x_{11}) \land (\neg x_{21} \lor \neg x_{22}) \land (\neg x_{22} \lor \neg x_{21})$$

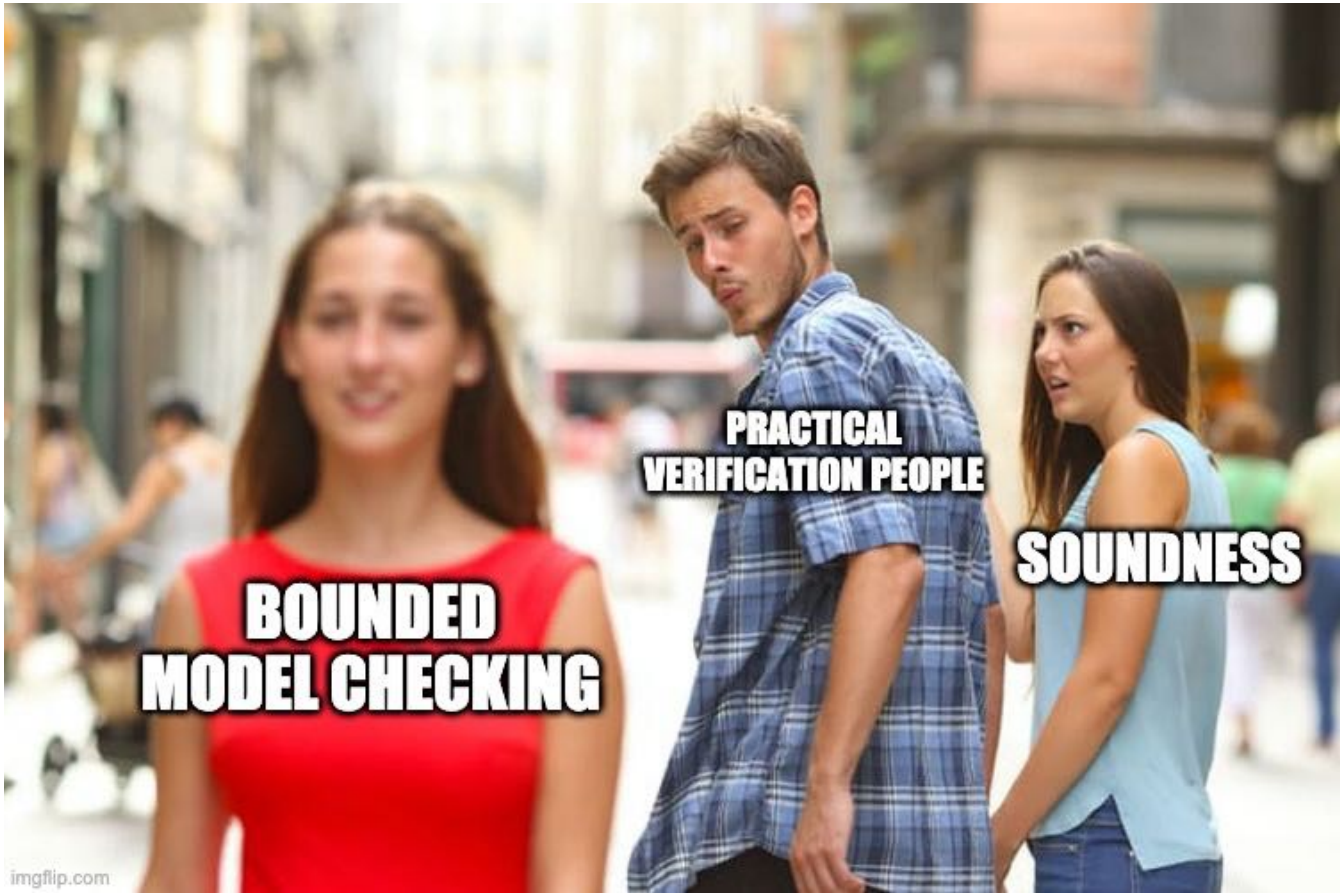
But realistically, no-one does these transformations and writes these dimacs files by hand...

CBMC

- Bounded Model Checking tool for C programs
- Based on producing a SAT formula for all possible paths through a program (with loops unwound to a bound N), and then asking a SAT solver if there is a path that violates an assertion
- Industrial users: Toyota, AWS

CBMC


C/C++ Source
parse
parse tree
CFG
un-wind
formula
flattening
CNF
SMT AUFBV
frontend


BOUNDED MODEL CHECKING
PRACTICAL VERIFICATION PEOPLE
SOUNDNESS
imgflip.com

CBMC - example 3

```
bool x;
char y=8, z=0, w=0;

if(x)
  z = y-1;
else
  w = y+1;

assert(z==7 || w==9);
```

CBMC - example 3

```
bool x;
char y=8, z=0, w=0;

if(x)
  z = y-1;
else
  w = y+1;

assert(z==7 || w==9);
```

$$(y = 8) \land (w = 0) \land (z = 0) \quad \land$$
$$(z = x \,?\, y - 1 : 0) \quad \land$$
$$(w = x \,?\, 0 : y + 1) \quad \land$$
$$(z \neq 7) \quad \land$$
$$(w \neq 9)$$

Example 1 - in CBMC

```
if(!a && !b) h();
else
    if(!a) g();
    else f();
```

```
if(a) f();
else
    if(b) g();
    else h();
```

CBMC - example 4

- That was a simple example.. there are much harder ones

Using model checking to triage the severity of security bugs in the Xen hypervisor.

Should we wake the developer up?

Byron Cook[1,2], Björn Döbel[1], Daniel Kroening[1,3], Norbert Manthey[1], Martin Pohlack[1], Elizabeth Polgreen[5,6], Michael Tautschnig[1,4], Pawel Wieczorkiewicz[1]

[1] Amazon Web Services

[2] University College London

[3] University of Oxford

[4] Queen Mary University of London

[5] UC Berkeley

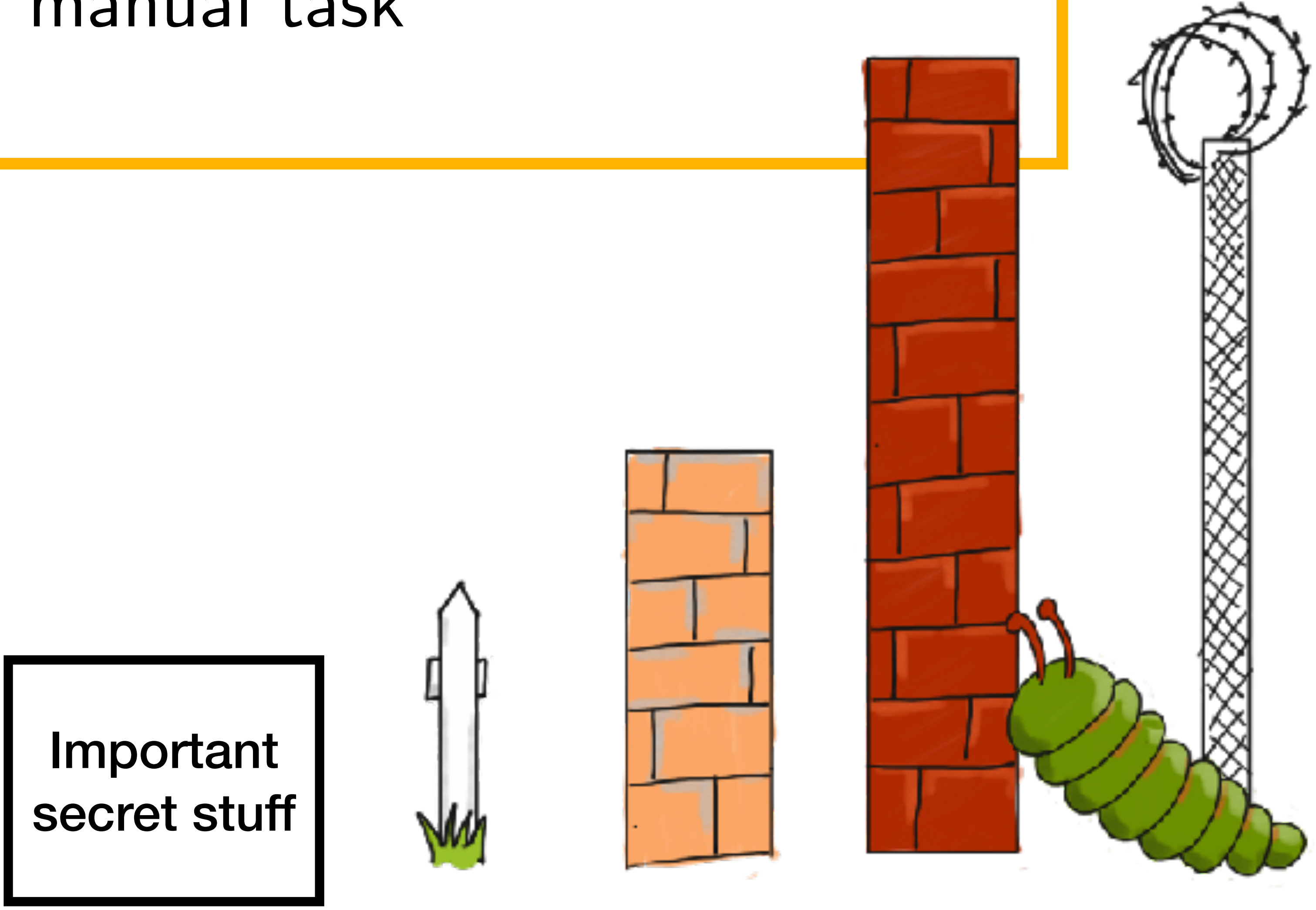
[6] Edinburgh University

Problem:

- Most systems have layers of security
- Most bugs are not critical security issues
- BUT determining which ones are is a difficult, manual task

Problem:

- Most systems have layers of security
- Most bugs are not critical security issues
- BUT determining which ones are is a difficult, manual task



Problem:

- Most systems have layers of security
- Most bugs are not critical security issues
- BUT determining which ones are is a difficult, manual task

Solution:

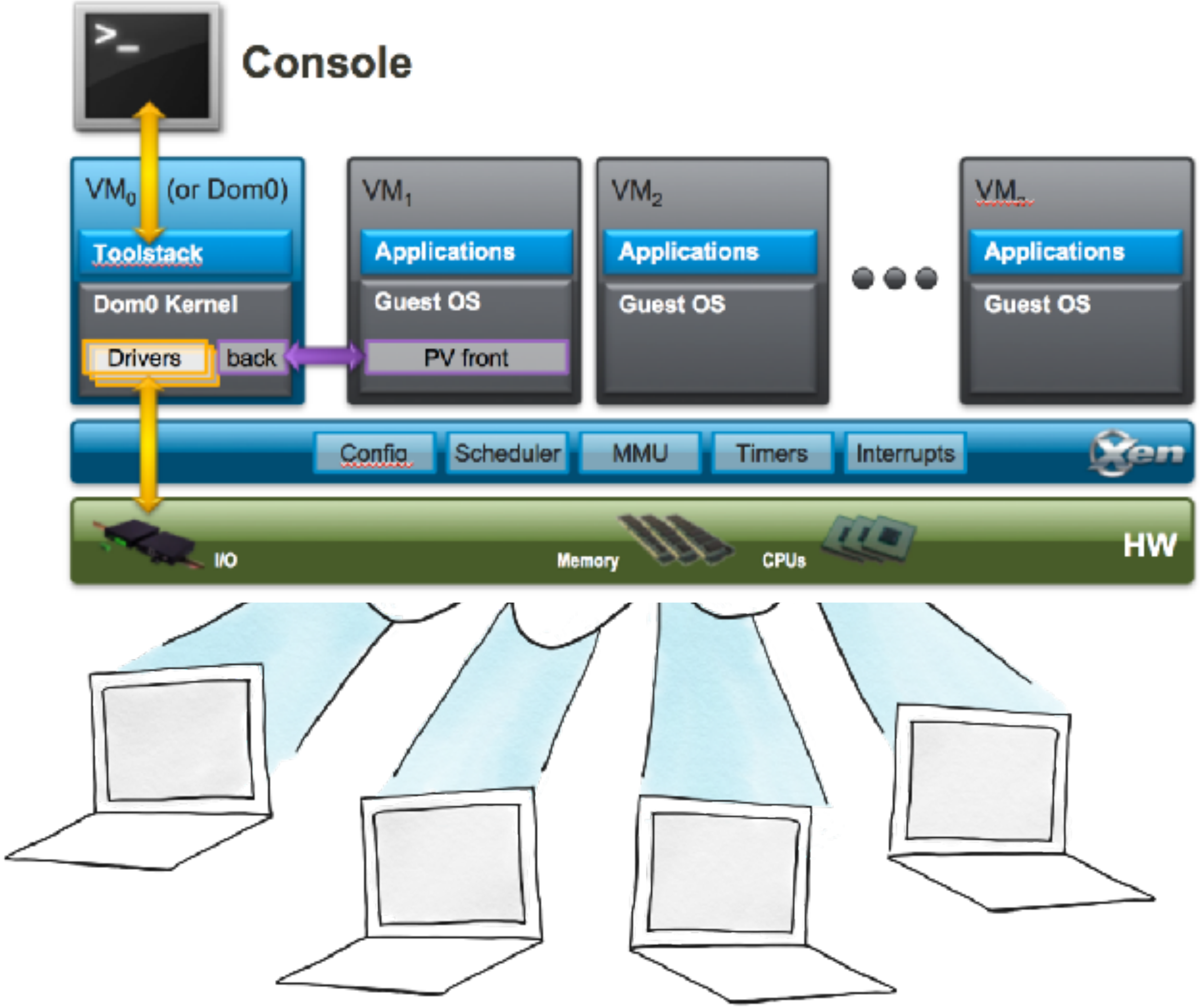
- We show how to use model checking to triage the severity of security bugs
- We make adaptations to CBMC, a bounded model checker for C programs, so that it scales to big code bases
- Case study: Xen

What is Xen?

Hypervisor: creates and runs virtual machines

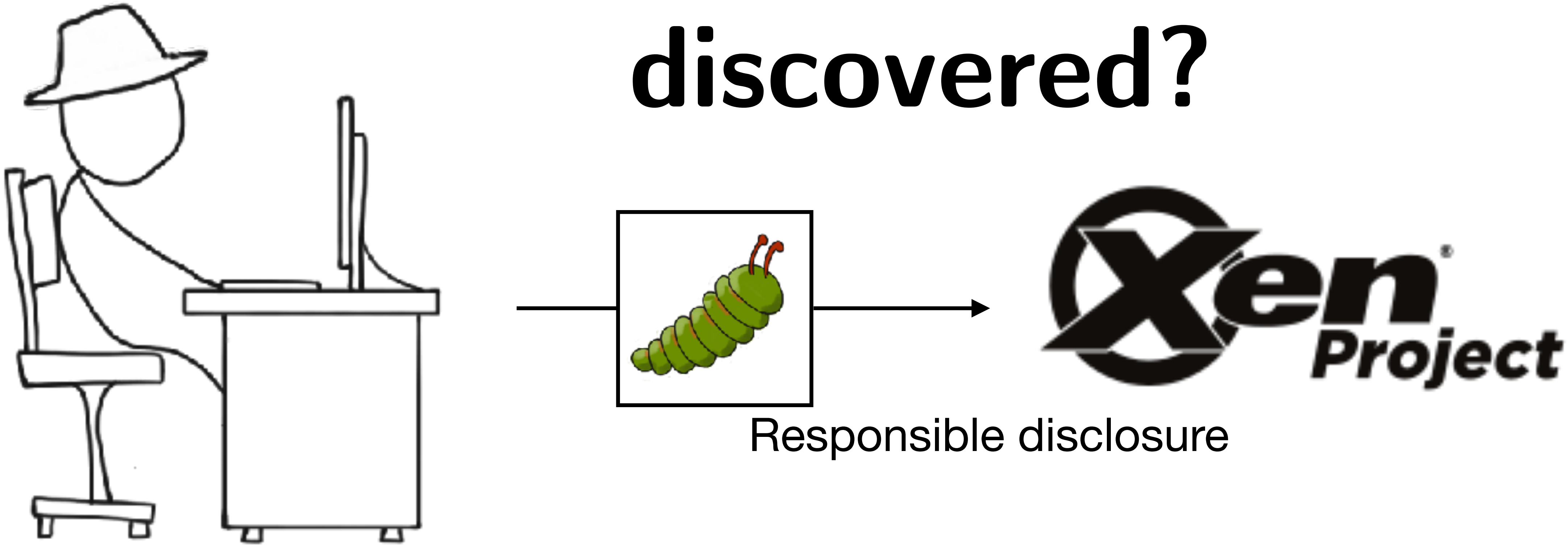
Amazon use a custom version of Xen on some EC2 servers

What is Xen?


Console
VM$_0$ (or Dom0)
Toolstack
Dom0 Kernel
Drivers
back
VM$_1$
Applications
Guest OS
PV front
VM$_2$
Applications
Guest OS
VM$_n$
Applications
Guest OS
Config
Scheduler
MMU
Timers
Interrupts
Xen
I/O
Memory
CPUs
HW

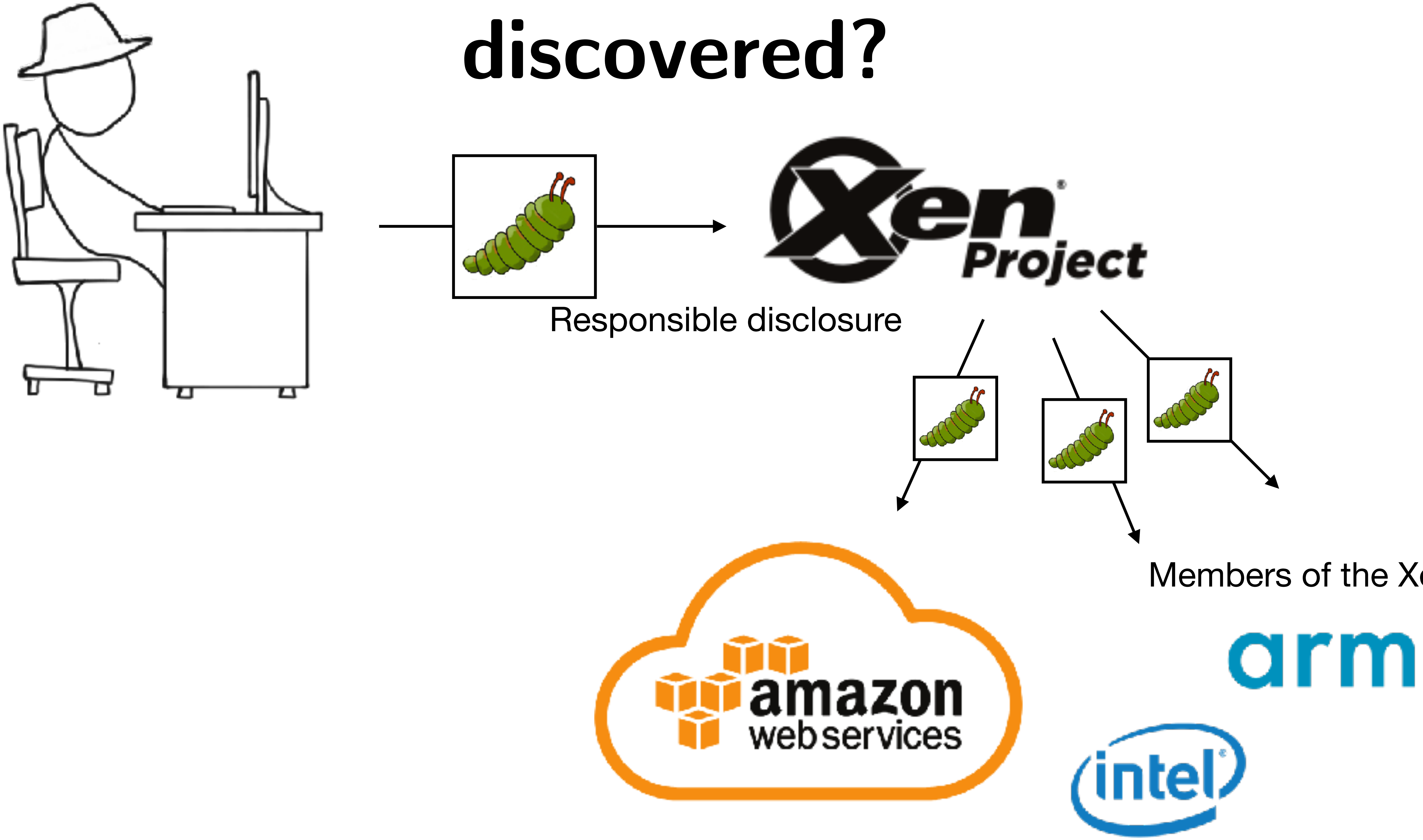
What happens when a bug is discovered?

What happens when a bug is discovered?



Responsible disclosure

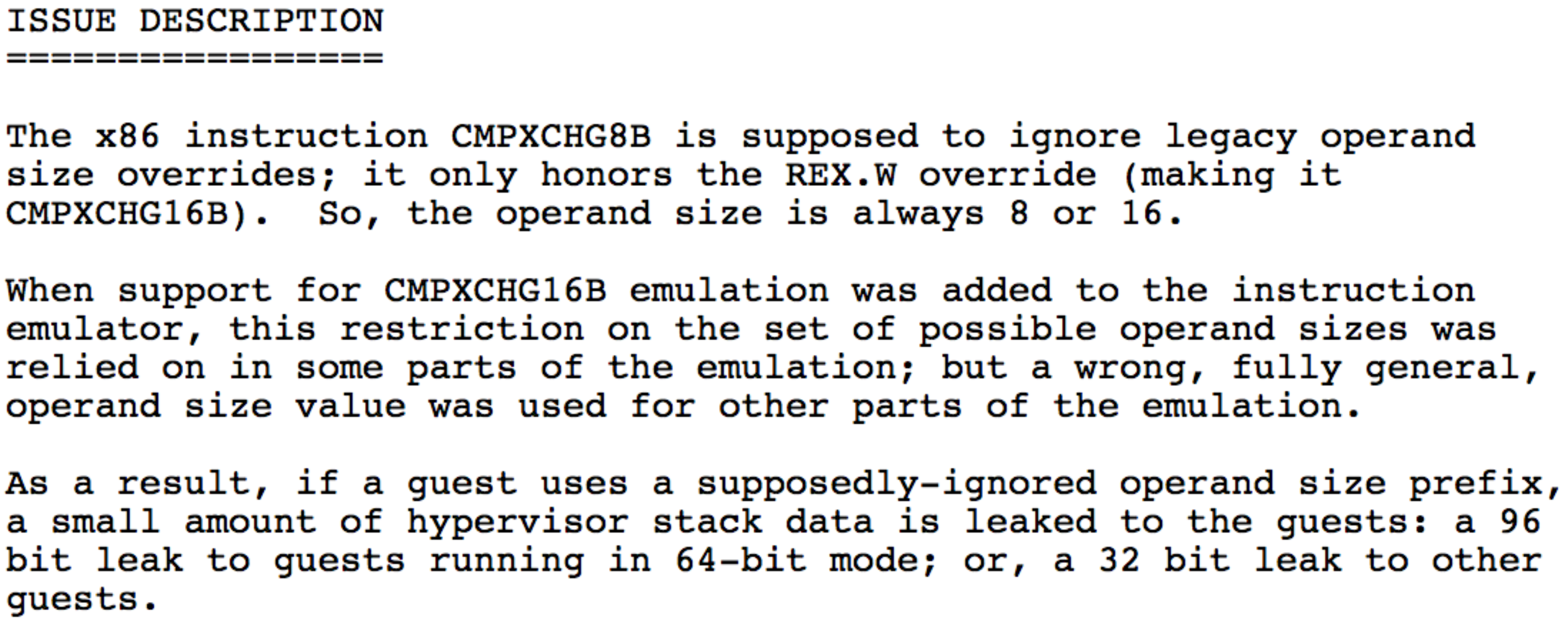
What happens when a bug is discovered?



Responsible disclosure

Members of the X

XSA: Xen Security Announcement

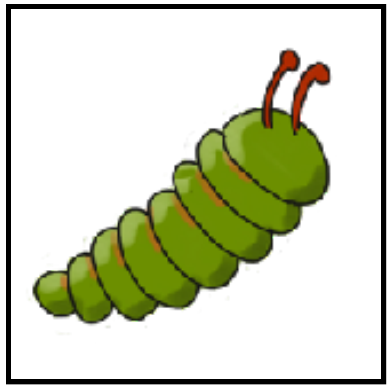


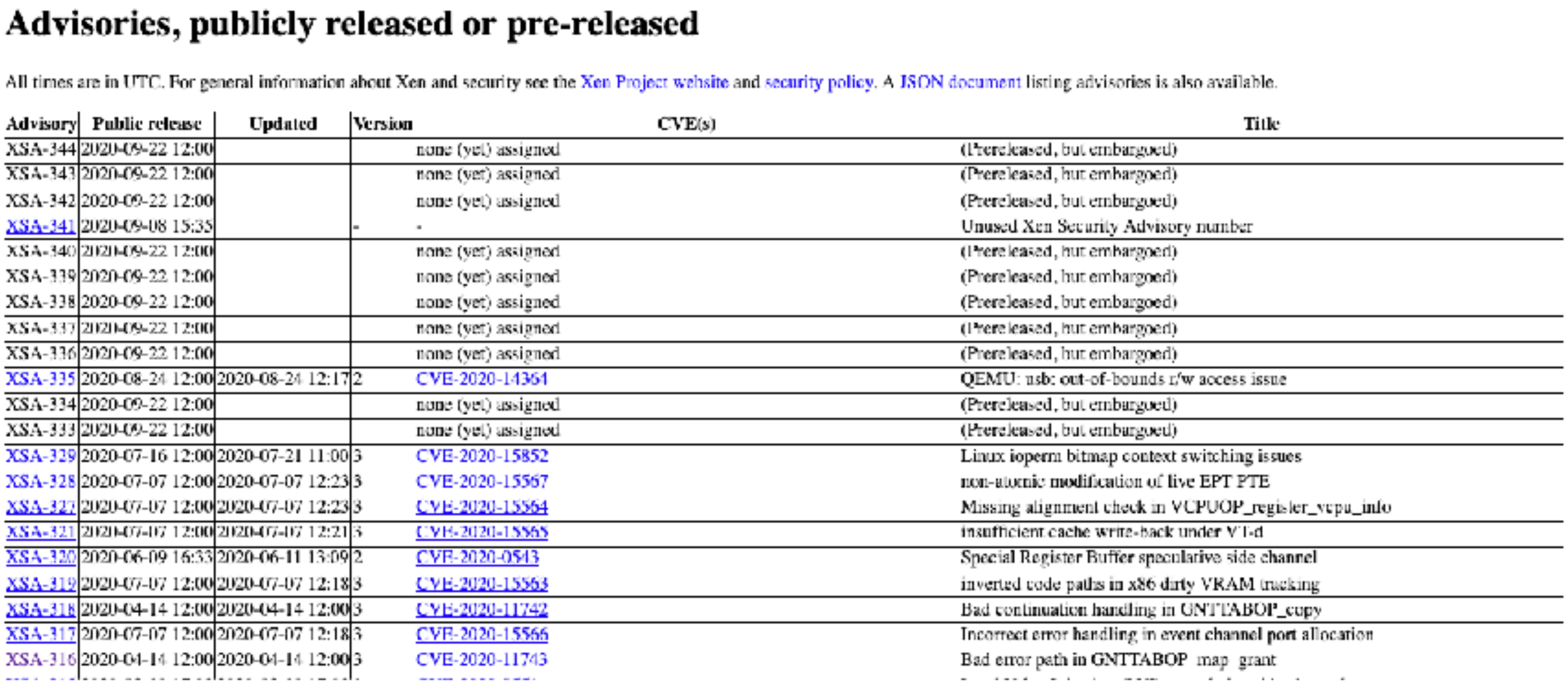
```
ISSUE DESCRIPTION
=================

The x86 instruction CMPXCHG8B is supposed to ignore legacy operand
size overrides; it only honors the REX.W override (making it
CMPXCHG16B).  So, the operand size is always 8 or 16.

When support for CMPXCHG16B emulation was added to the instruction
emulator, this restriction on the set of possible operand sizes was
relied on in some parts of the emulation; but a wrong, fully general,
operand size value was used for other parts of the emulation.

As a result, if a guest uses a supposedly-ignored operand size prefix,
a small amount of hypervisor stack data is leaked to the guests: a 96
bit leak to guests running in 64-bit mode; or, a 32 bit leak to other
guests.
```

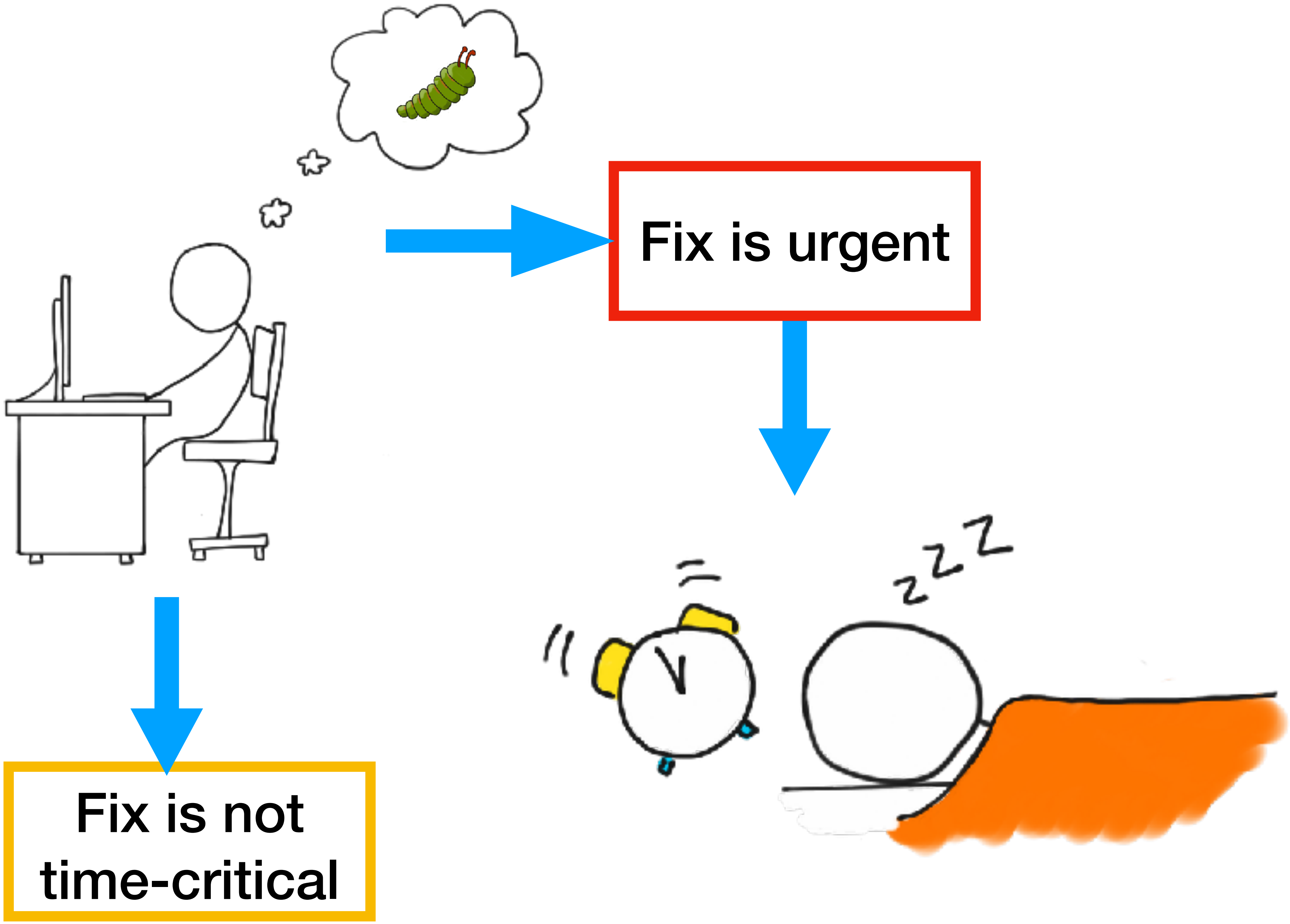




Advisories, publicly released or pre-released

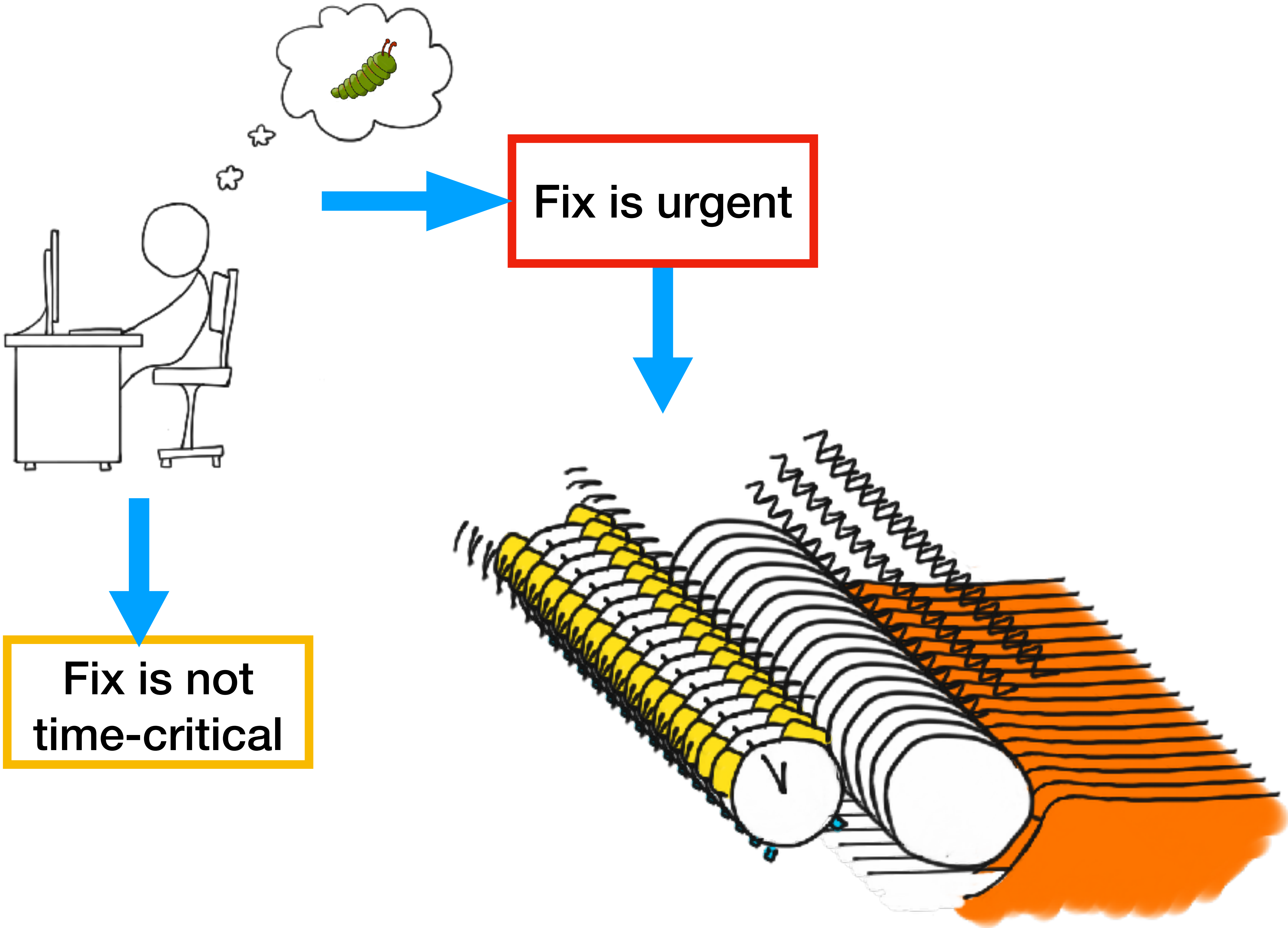
All times are in UTC. For general information about Xen and security see the Xen Project website and security policy. A JSON document listing advisories is also available.

Advisory	Public release	Updated	Version	CVE(s)	Title
XSA-344	2020-09-22 12:00			none (yet) assigned	(Prereleased, but embargoed)
XSA-343	2020-09-22 12:00			none (yet) assigned	(Prereleased, but embargoed)
XSA-342	2020-09-22 12:00			none (yet) assigned	(Prereleased, but embargoed)
XSA-341	2020-09-08 15:35		-	-	Unused Xen Security Advisory number
XSA-340	2020-09-22 12:00			none (yet) assigned	(Prereleased, but embargoed)
XSA-339	2020-09-22 12:00			none (yet) assigned	(Prereleased, but embargoed)
XSA-338	2020-09-22 12:00			none (yet) assigned	(Prereleased, but embargoed)
XSA-337	2020-09-22 12:00			none (yet) assigned	(Prereleased, but embargoed)
XSA-336	2020-09-22 12:00			none (yet) assigned	(Prereleased, but embargoed)
XSA-335	2020-08-24 12:00	2020-08-24 12:17	2	CVE-2020-14364	QEMU: usb: out-of-bounds r/w access issue
XSA-334	2020-09-22 12:00			none (yet) assigned	(Prereleased, but embargoed)
XSA-333	2020-09-22 12:00			none (yet) assigned	(Prereleased, but embargoed)
XSA-329	2020-07-16 12:00	2020-07-21 11:00	3	CVE-2020-15852	Linux ioperm bitmap context switching issues
XSA-328	2020-07-07 12:00	2020-07-07 12:23	3	CVE-2020-15567	non-atomic modification of live EPT PTE
XSA-327	2020-07-07 12:00	2020-07-07 12:23	3	CVE-2020-15564	Missing alignment check in VCPUOP_register_vcpu_info
XSA-321	2020-07-07 12:00	2020-07-07 12:21	3	CVE-2020-15565	insufficient cache write-back under VT-d
XSA-320	2020-06-09 16:33	2020-06-11 13:09	2	CVE-2020-0543	Special Register Buffer speculative side channel
XSA-319	2020-07-07 12:00	2020-07-07 12:18	3	CVE-2020-15563	inverted code paths in x86 dirty VRAM tracking
XSA-318	2020-04-14 12:00	2020-04-14 12:00	3	CVE-2020-11742	Bad continuation handling in GNTTABOP_copy
XSA-317	2020-07-07 12:00	2020-07-07 12:18	3	CVE-2020-15566	Incorrect error handling in event channel port allocation
XSA-316	2020-04-14 12:00	2020-04-14 12:00	3	CVE-2020-11743	Bad error path in GNTTABOP_map_grant



Fix is urgent

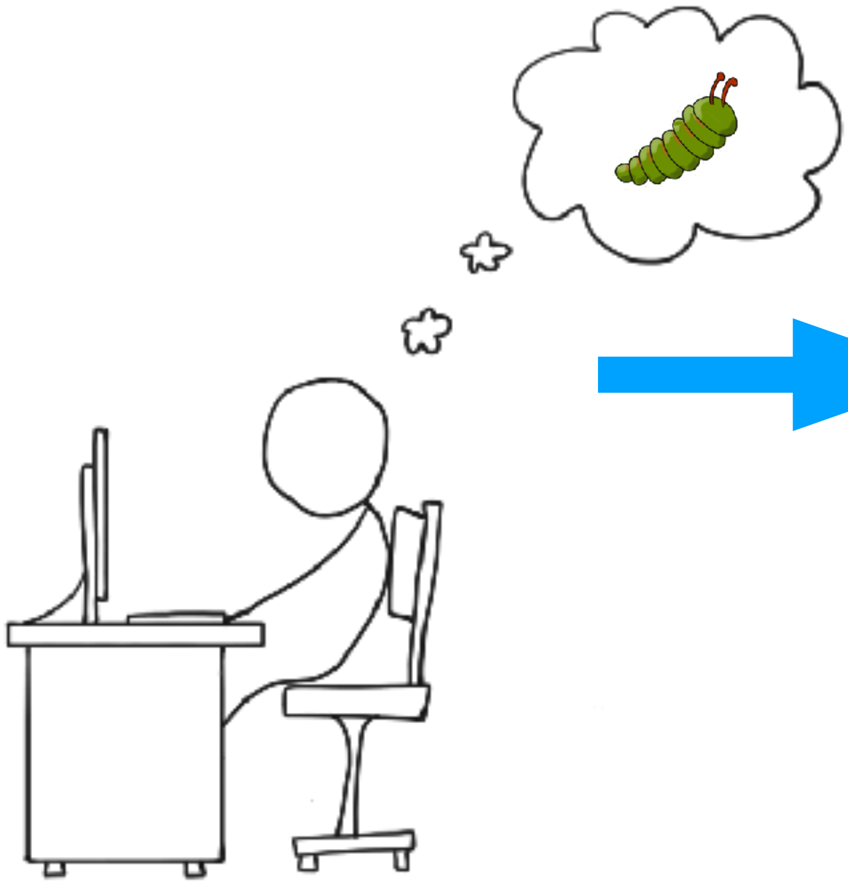
Fix is not time-critical



Fix is urgent

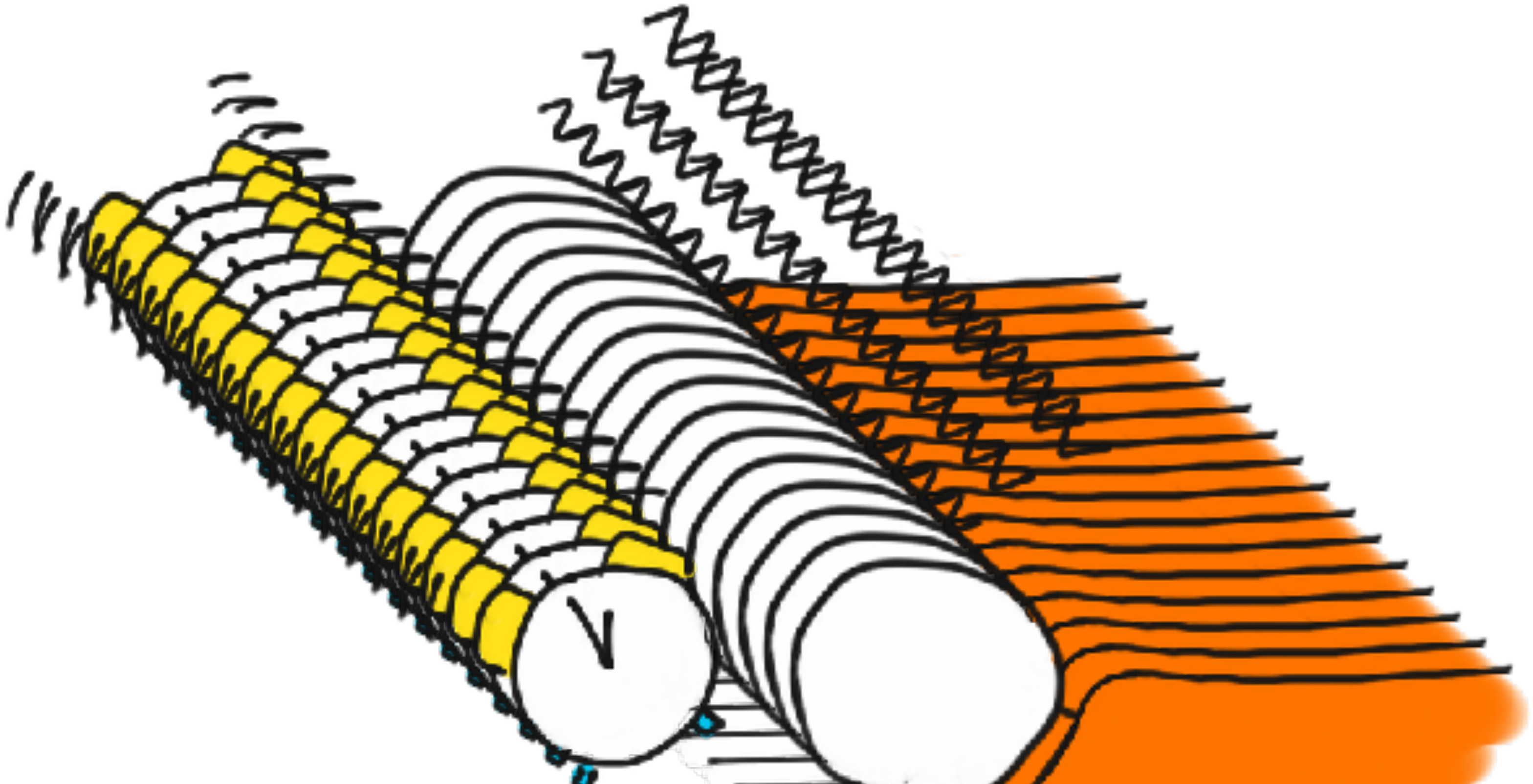
Fix is not time-critical

When is a fix urgent?

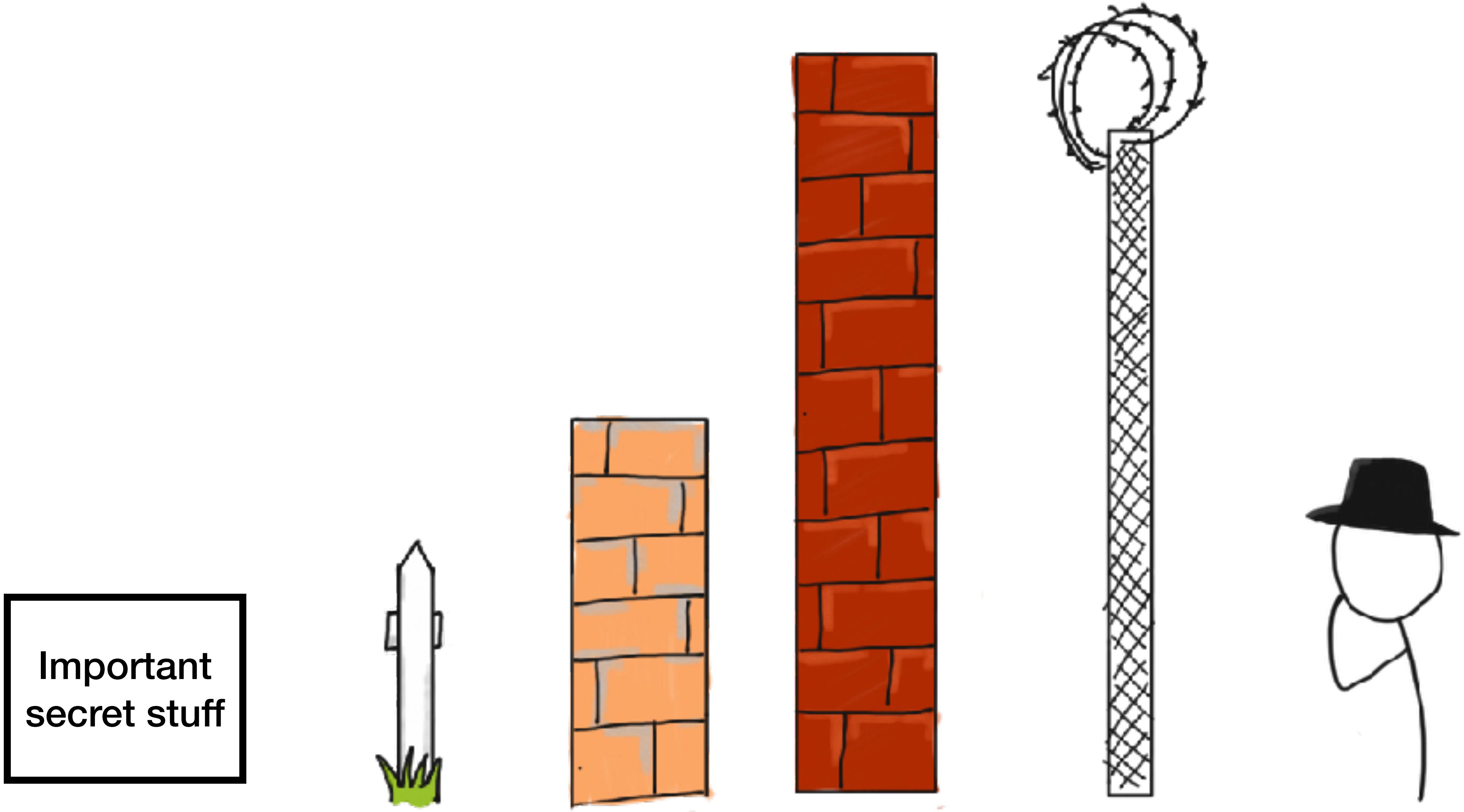


Fix is urgent

Fix is not time-critical

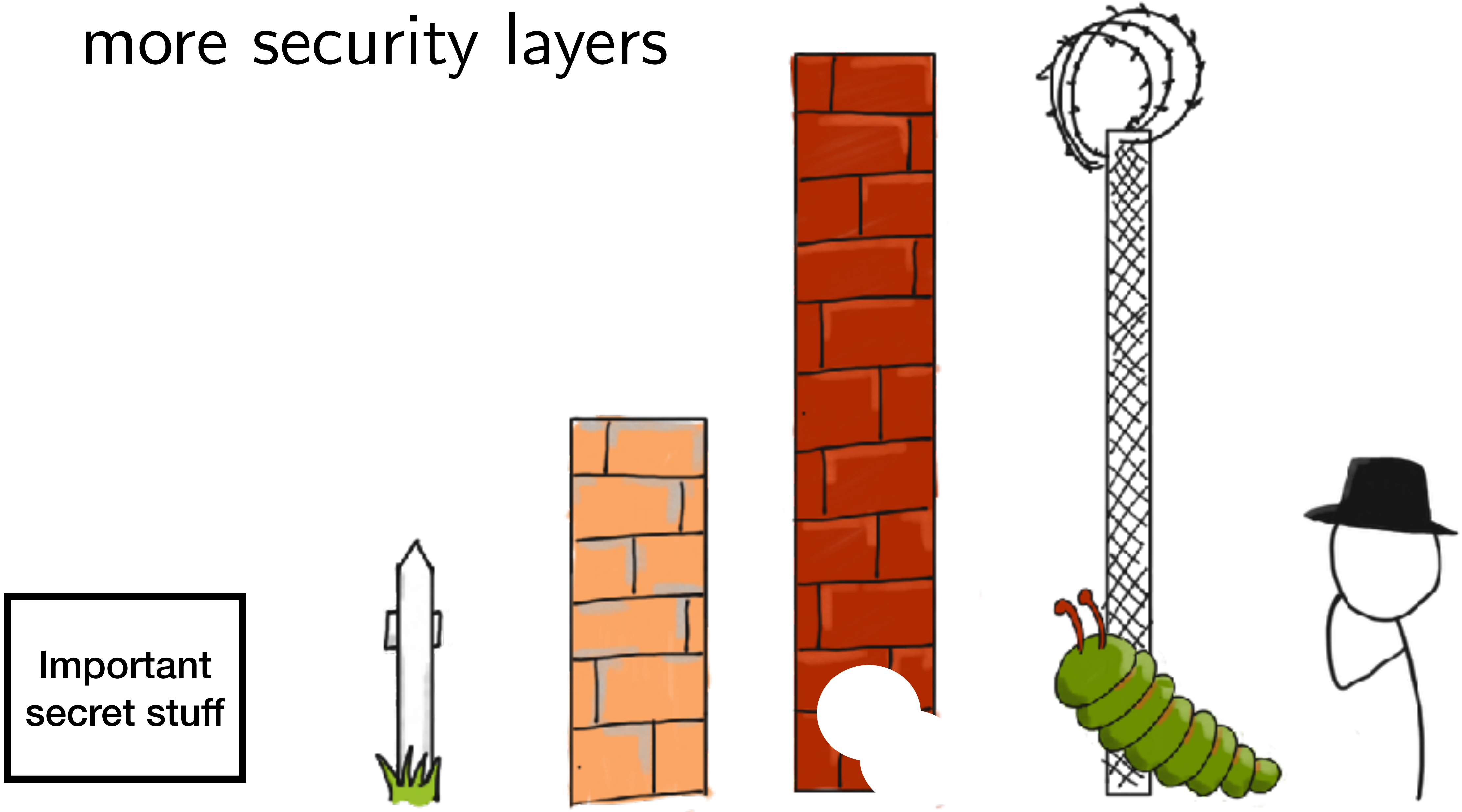


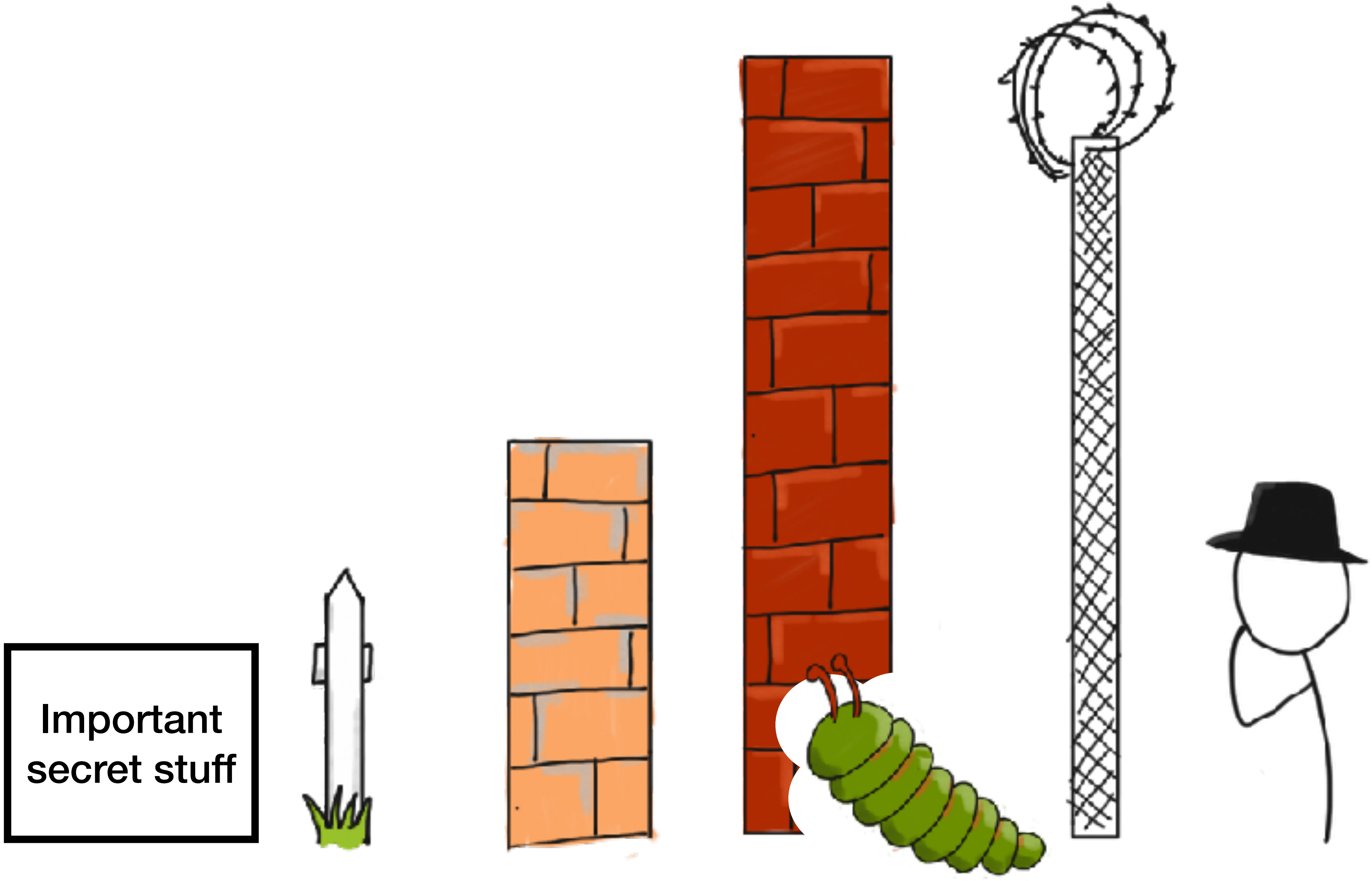
- Well-engineered systems are built with defence in depth



Important secret stuff

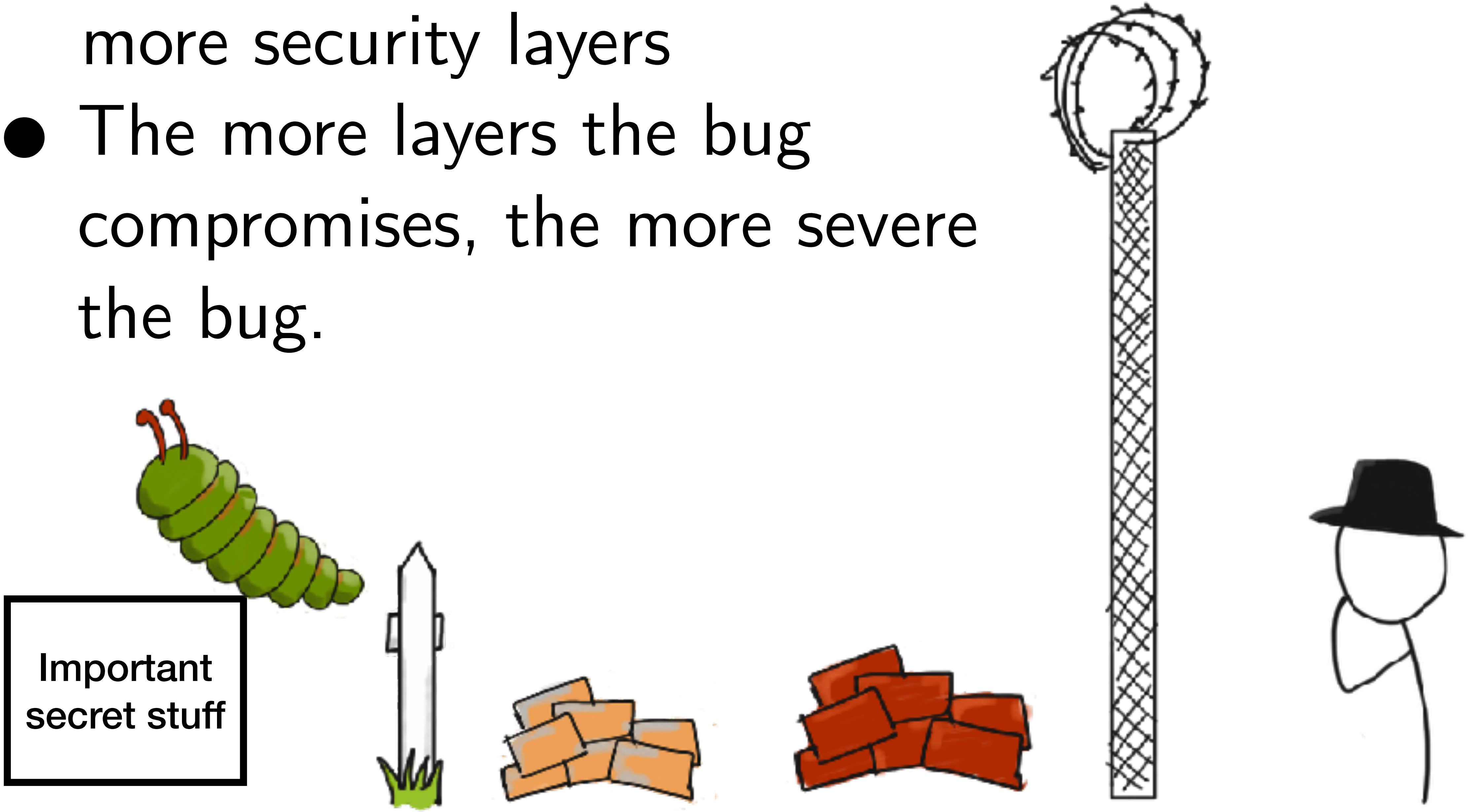
- Well-engineered systems are built with defence in depth
- Bugs may compromise one or more security layers



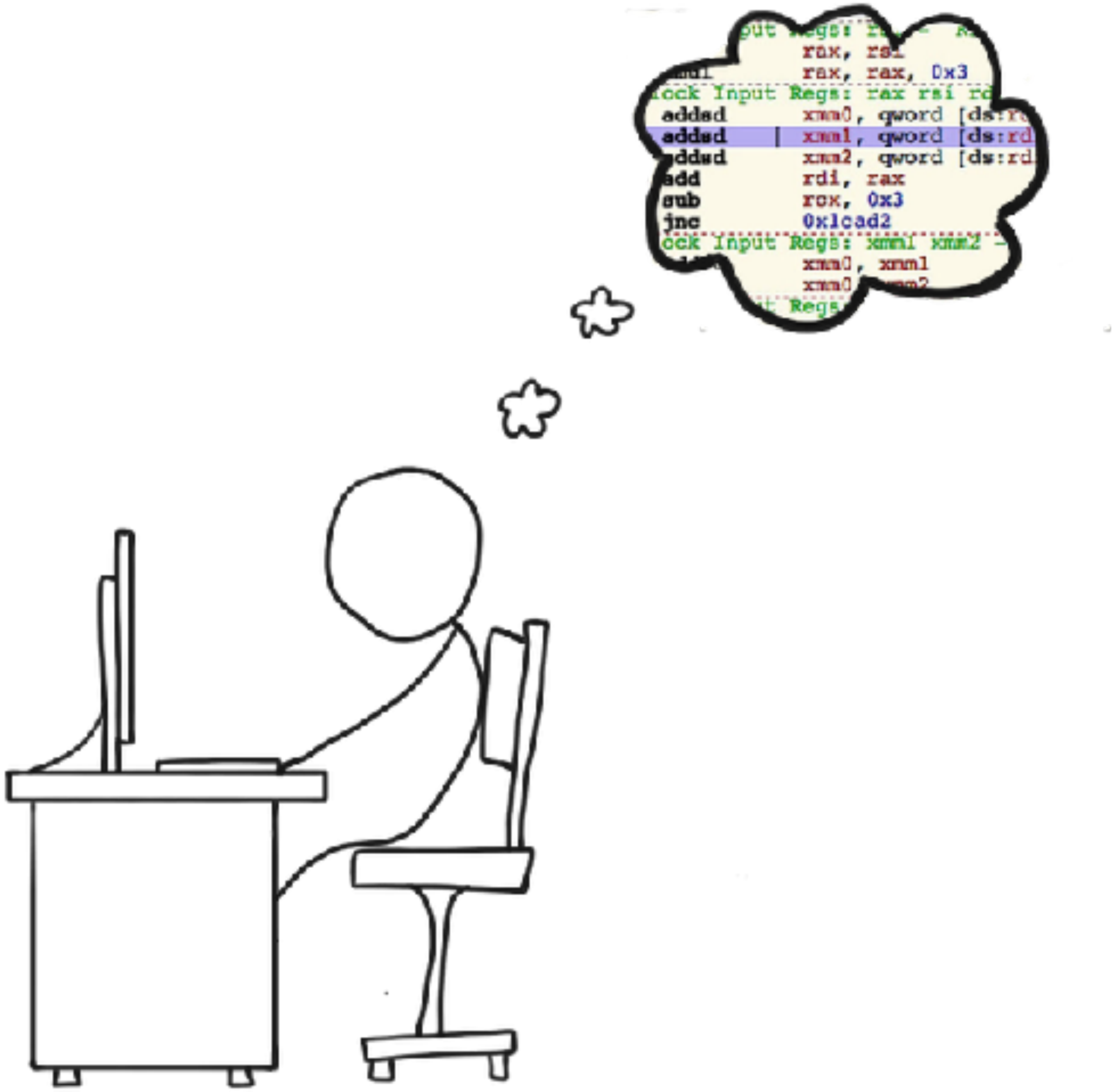


Important secret stuff

- Well-engineered systems are built with defence in depth
- Bugs may compromise one or more security layers
- The more layers the bug compromises, the more severe the bug.



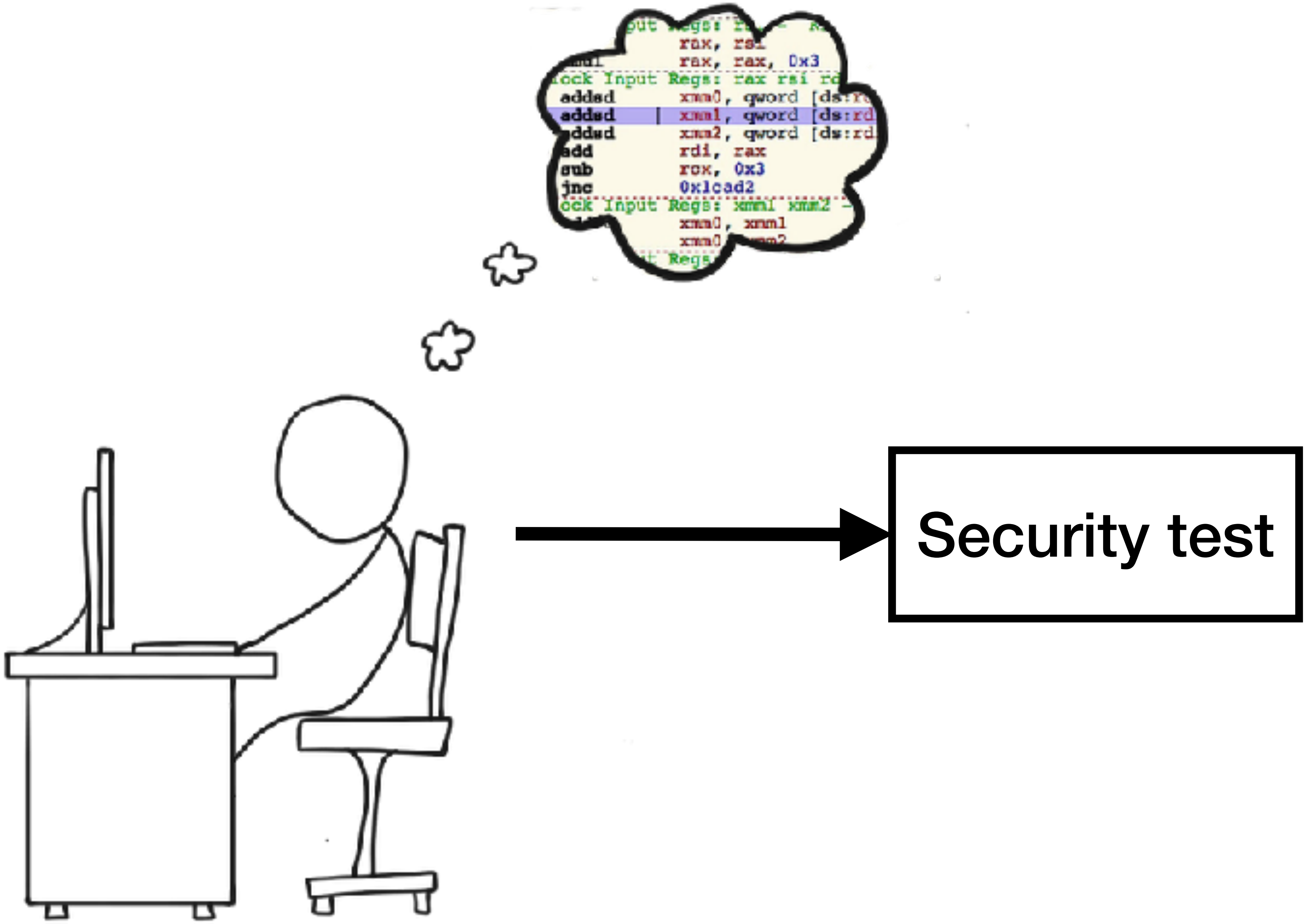
How do we determine if a fix is urgent?



```
put Regs: r
          rax, rsi
          rax, rax, 0x3
ock Input Regs: rax rsi rd
addsd     xmm0, qword [ds:r
addsd   | xmm1, qword [ds:rd
addsd     xmm2, qword [ds:rd
add       rdi, rax
sub       rcx, 0x3
jne       0x1cad2
ock Input Regs: xmm1 xmm2 -
          xmm0, xmm1
          xmm0,
t Regs
```

]

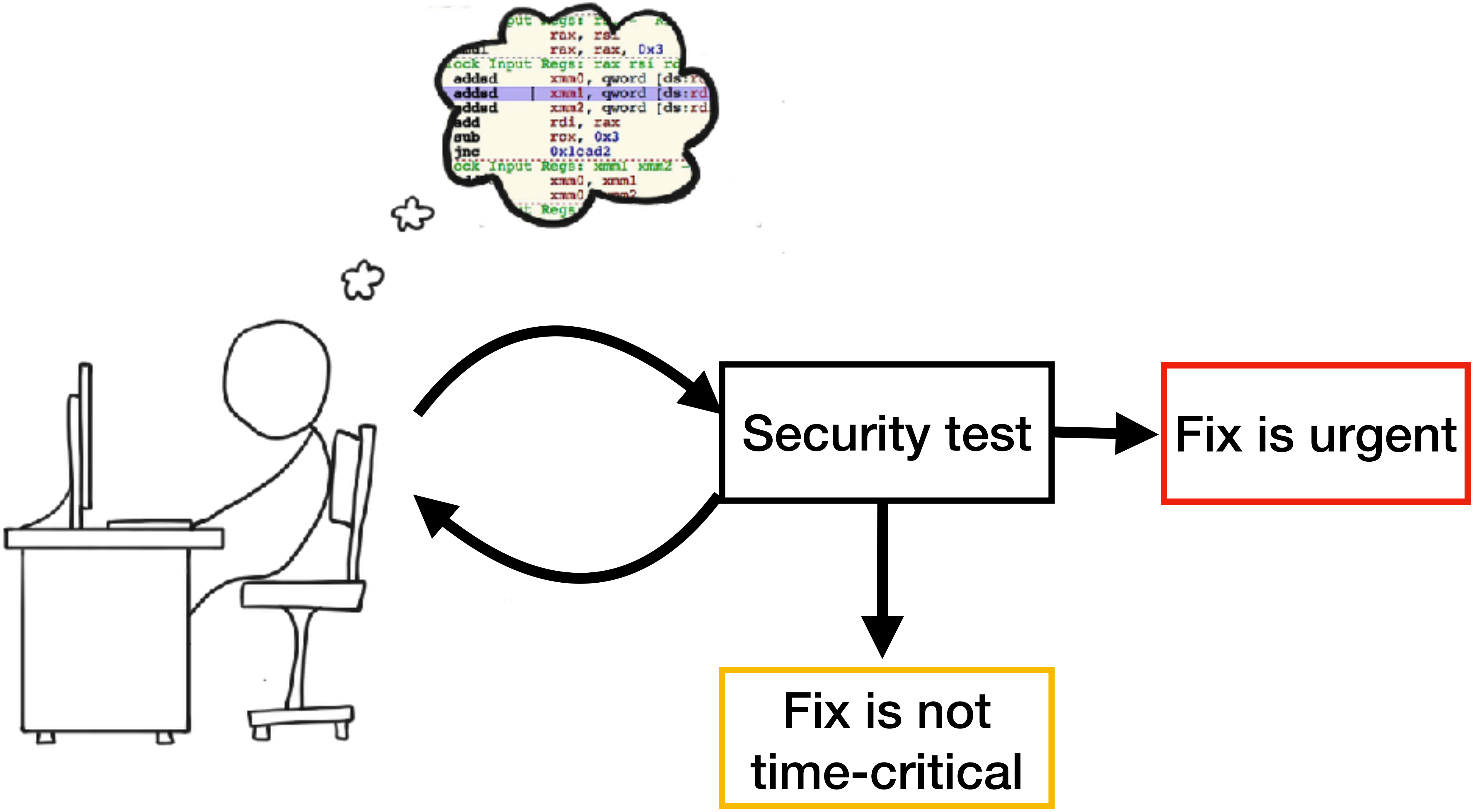
How do we determine if a fix is urgent?



```
put Regs: r
            rax, rsi
            rax, rax, 0x3
ock Input Regs: rax rsi rd
addsd       xmm0, qword [ds:r
addsd     | xmm1, qword [ds:rd
addsd       xmm2, qword [ds:rd
add         rdi, rax
sub         rcx, 0x3
jne         0x1cad2
ock Input Regs: xmm1 xmm2 -
            xmm0, xmm1
            xmm0
t Regs
```

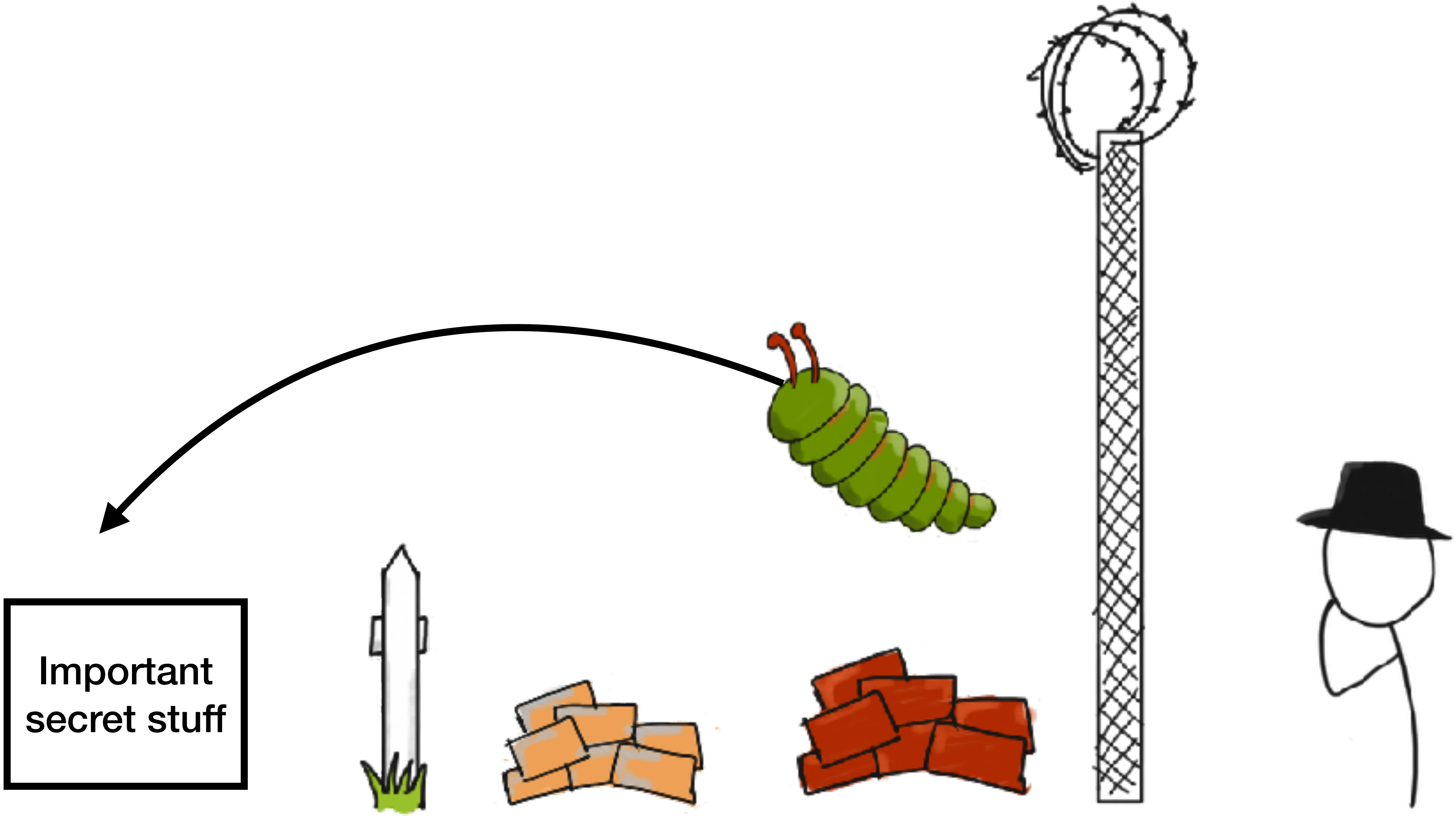
Security test

How do we determine if a fix is urgent?



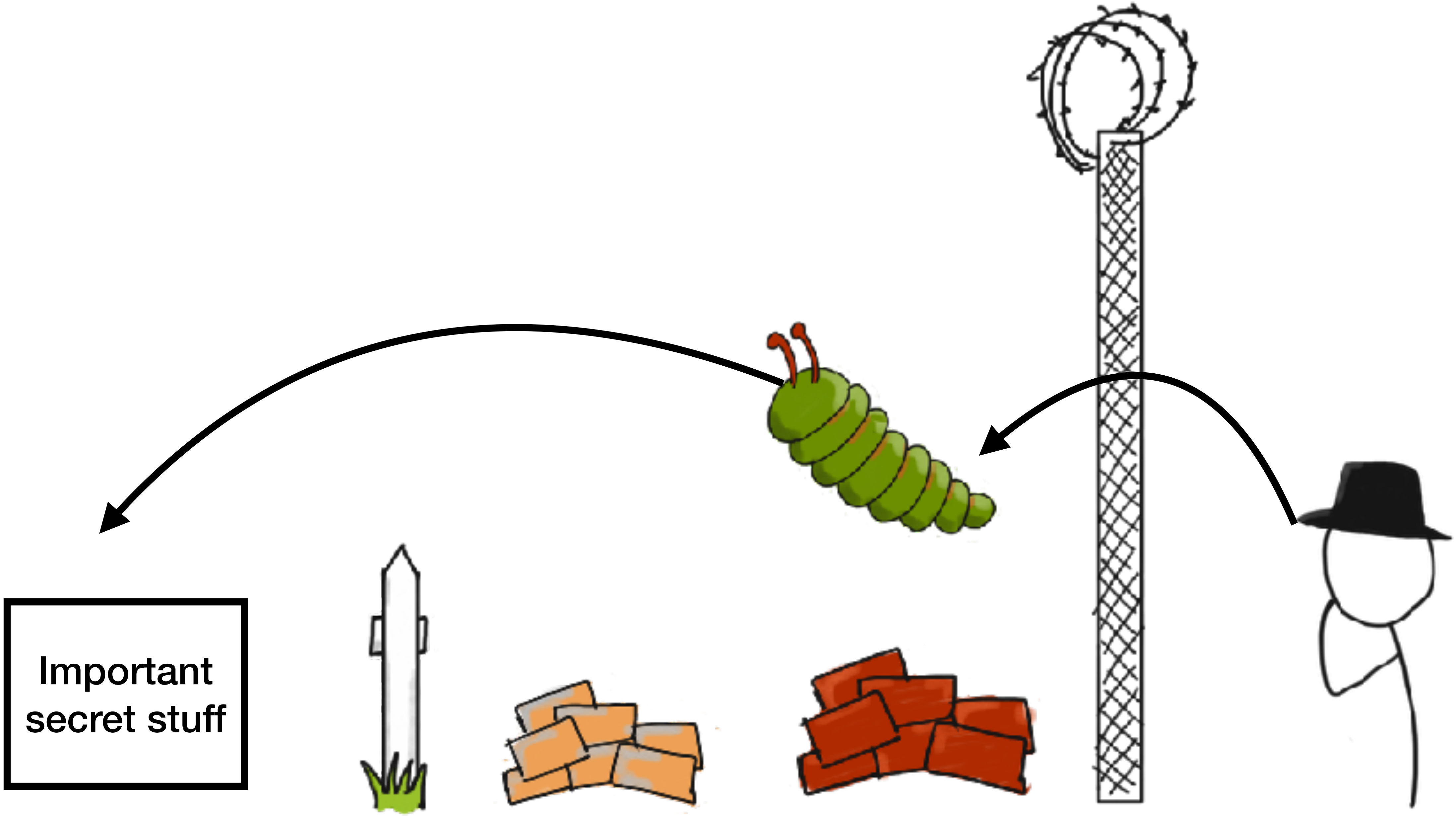
```
              rax, rsi
              rax, rax, 0x3
ock Input Regs: rax rsi rd
addsd         xmm0, qword [ds:r
addsd       | xmm1, qword [ds:rd
addsd         xmm2, qword [ds:rd
add           rdi, rax
sub           rcx, 0x3
jnc           0x1cad2
ock Input Regs: xmm1 xmm2 -
              xmm0, xmm1
              xmm0
```

Using SAT-based model checking



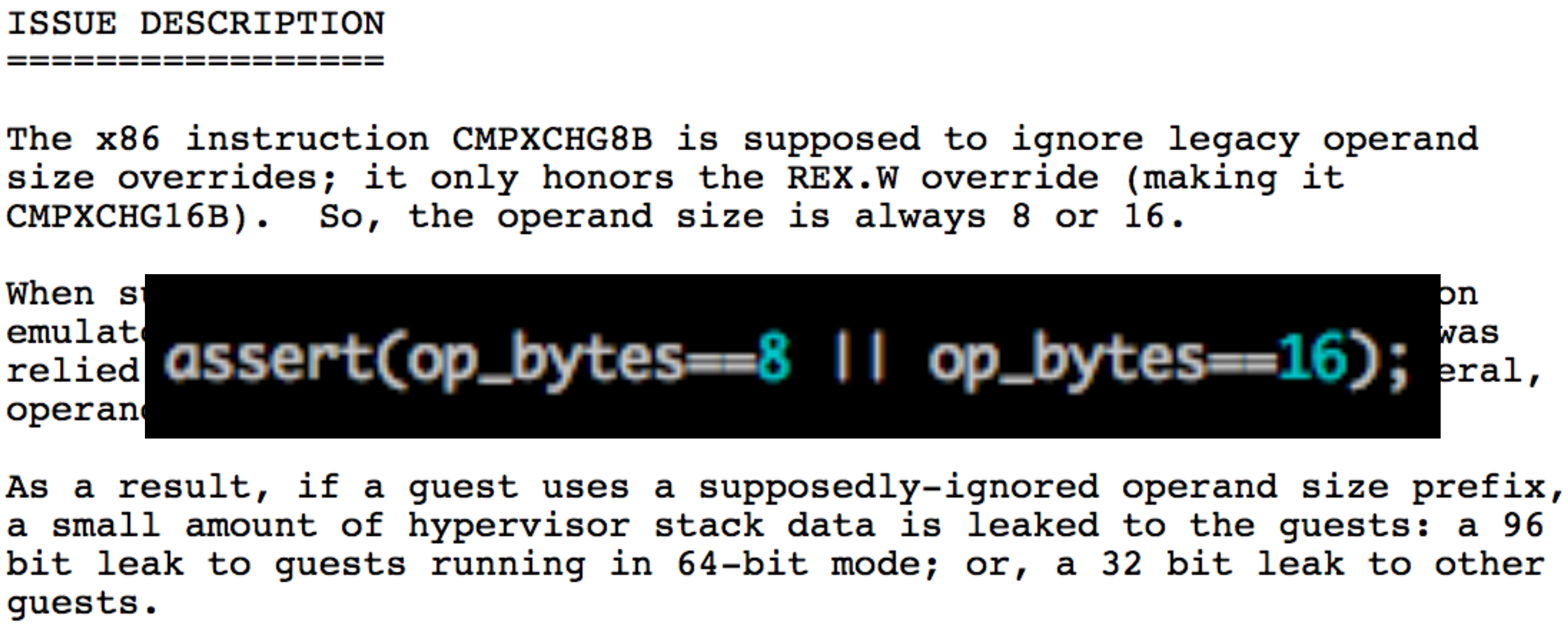
Important secret stuff

Security tests establish reachability of the bug



Important secret stuff

Reachability assertion



ISSUE DESCRIPTION
=================

The x86 instruction CMPXCHG8B is supposed to ignore legacy operand size overrides; it only honors the REX.W override (making it CMPXCHG16B). So, the operand size is always 8 or 16.

When s… on
emulat… was
relied… eral,
operan…

```
assert(op_bytes==8 || op_bytes==16);
```

As a result, if a guest uses a supposedly-ignored operand size prefix, a small amount of hypervisor stack data is leaked to the guests: a 96 bit leak to guests running in 64-bit mode; or, a 32 bit leak to other guests.

SAT take-aways

- SAT solvers are surprisingly good at NP
- If you have an NP problem, don't solve it yourself, translate it into SAT!
- Don't do the translation yourself! Use a tool.

SAT

A : Boolean

B : Boolean

$$\exists A, B$$

$$A \land \neg B$$

A : true

B : false

SMT

A : Integer

B : Integer

$$\exists A, B$$

$$A > 0 \land B < 0$$

A : 10

B : -3

Satisfiability Modulo Theories

- SMT solvers solve formula in some quantifier-free fragment of a first-order theory T

- Formulas use propositional connectives and a set Σ of additional function and predicate symbols that uniquely define the theory T

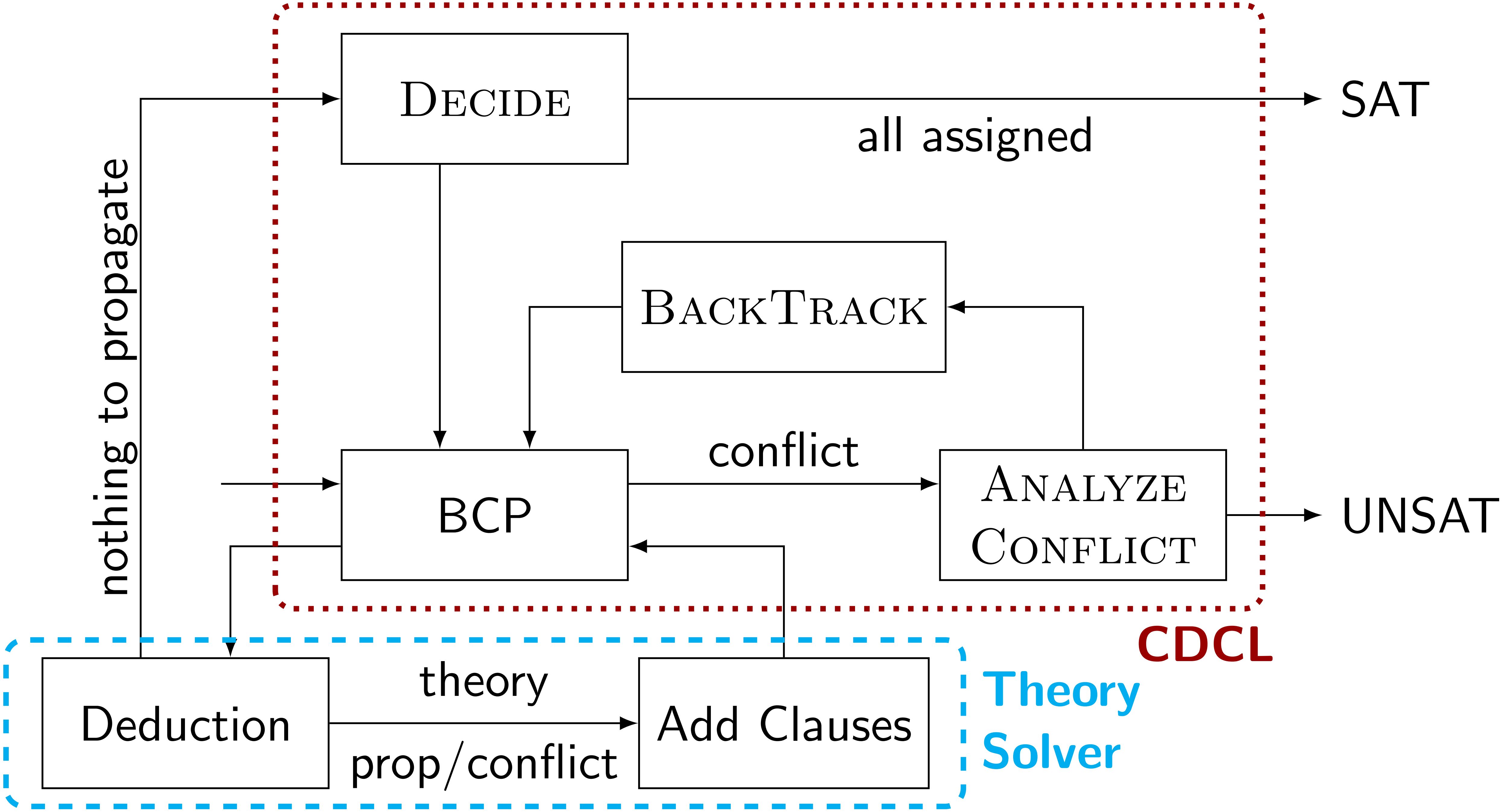
- Σ is called the signature of T

- SMT solvers determine whether the formula is T-valid, T-satisfiable or T-unsatisfiable

What theories?

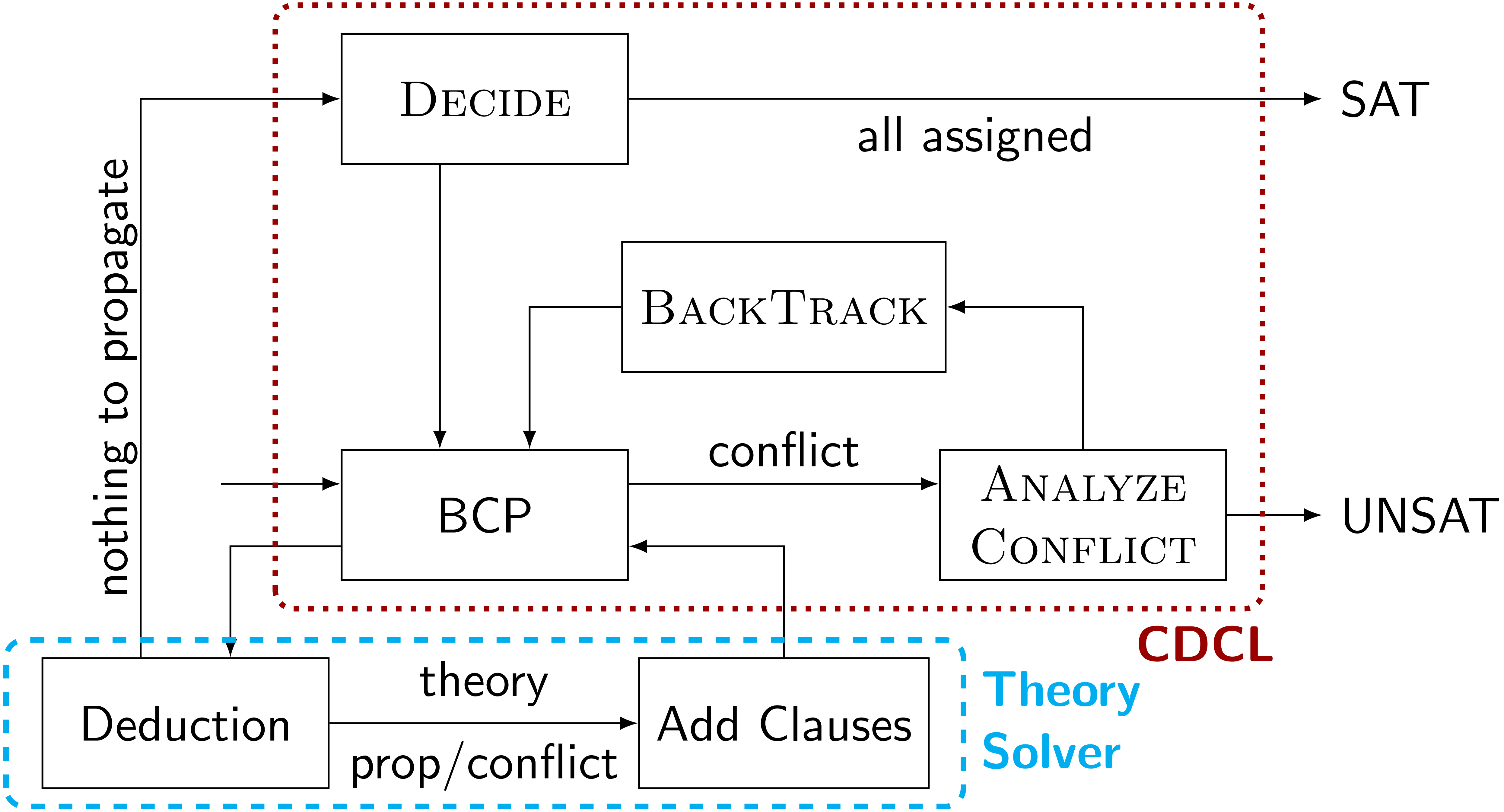
- Arrays
- BitVectors
- Floating Point
- Integers
- Reals
- String

CDCL(T)



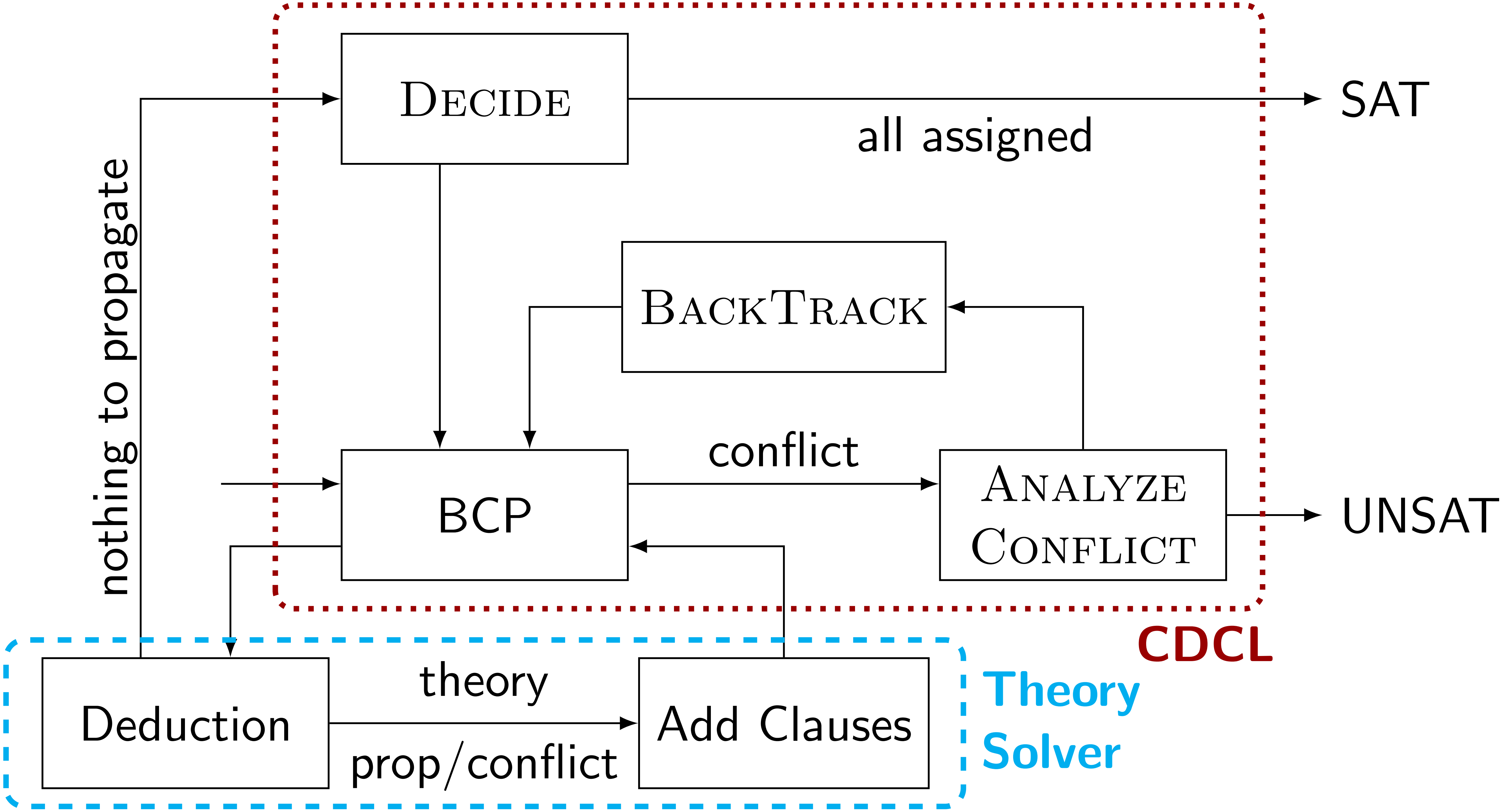
Step 1: make a boolean skeleton of your formula

CDCL(T)



Step 2: give the boolean skeleton to the SAT solver

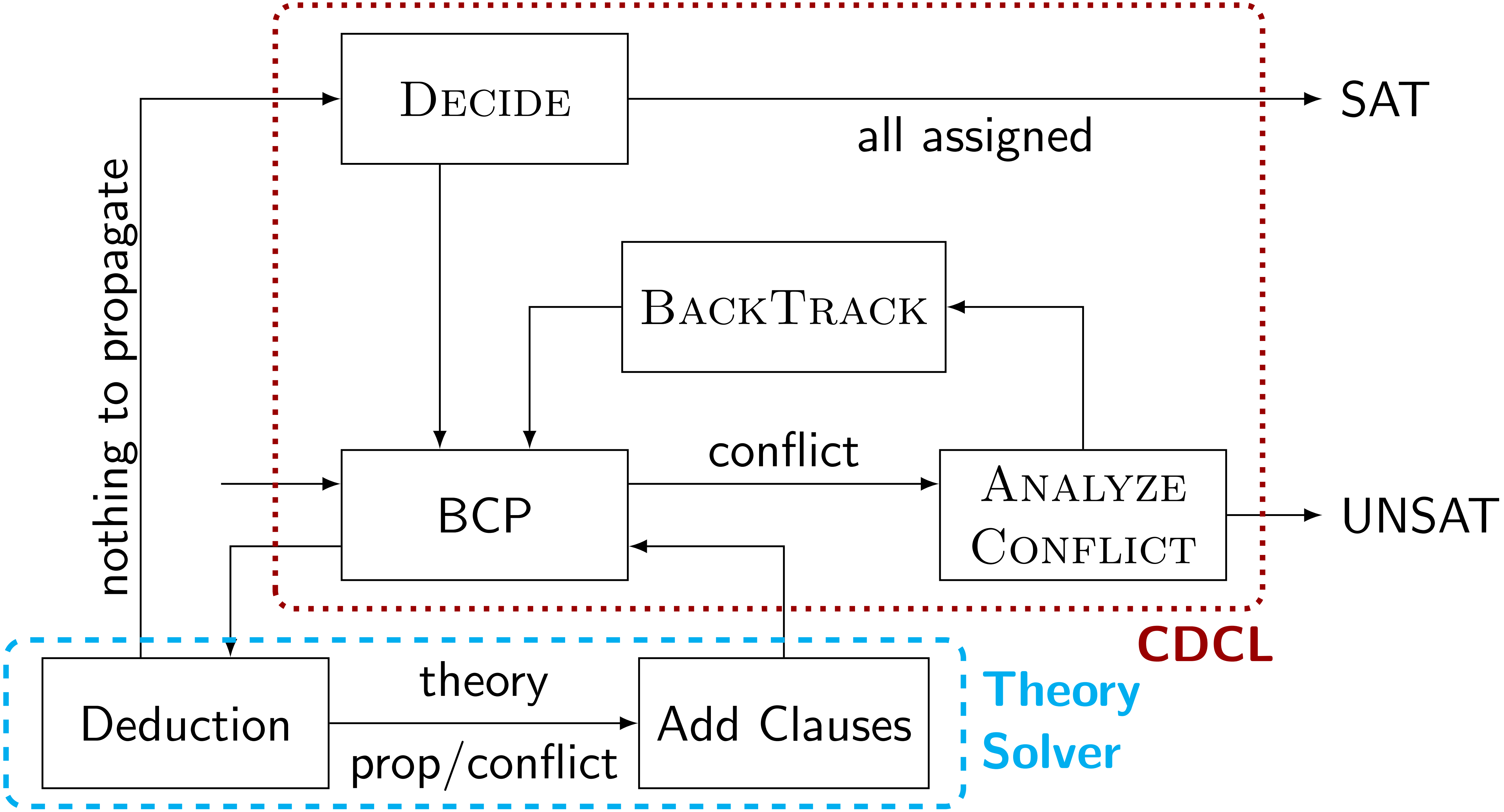
CDCL(T)



Step 2: give the boolean skeleton to the SAT solver

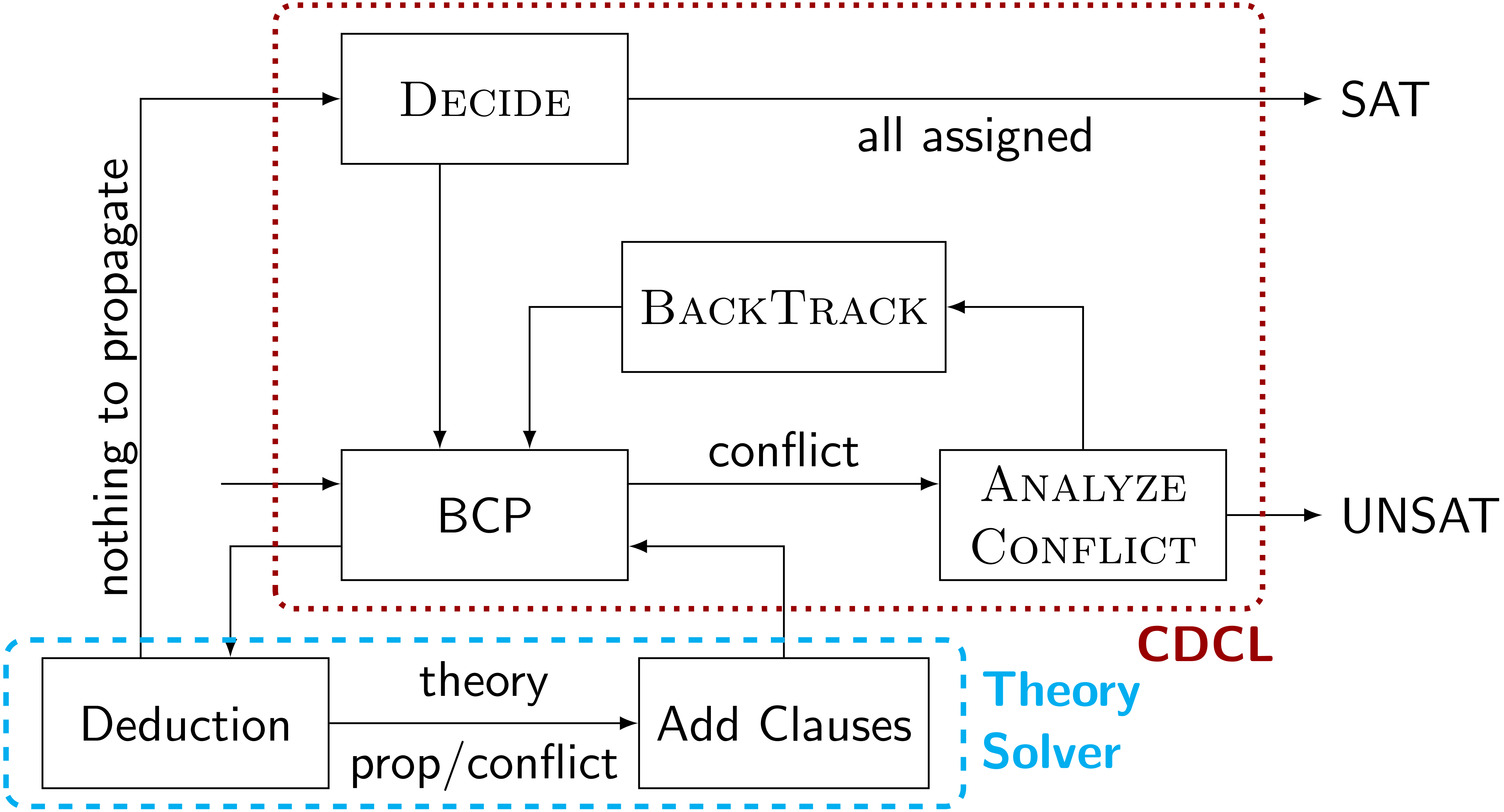
If it's UNSAT, we're done!

CDCL(T)



Step 3: If it's SAT, check with the theory solver

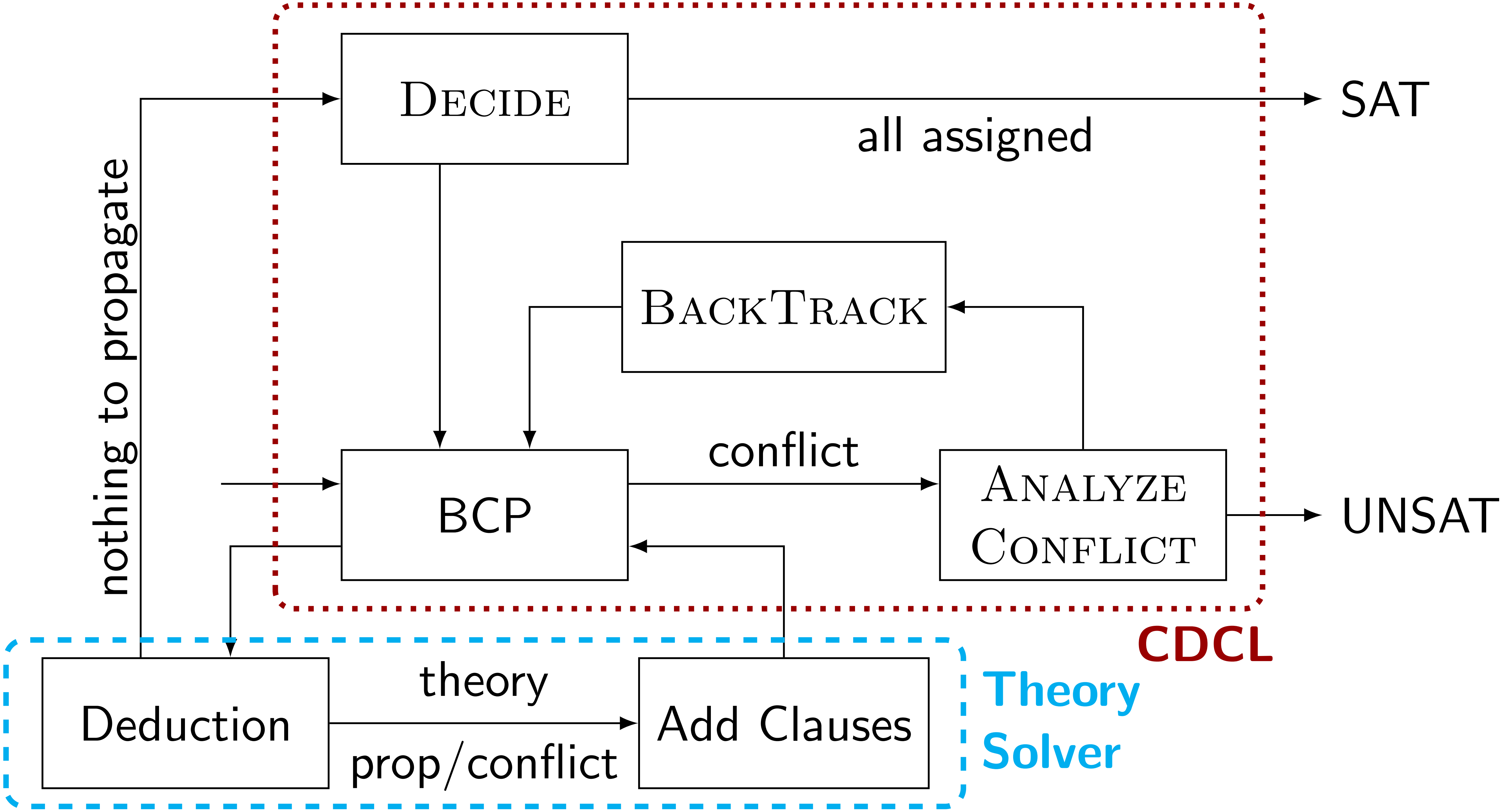
CDCL(T)



Step 3: If it's SAT, check with the theory solver

If theory solver agrees: return assignment

CDCL(T)



Step 3: If it's SAT, check with the theory solver

Otherwise, return clause to block assignment

Notes

- SMT solvers combine theories with Nelsen-Oppen

- Combinations of theories may be undecidable, even if the individual theories are decidable.

- Not covering details: read “Decision Procedures” by Strichman and Kroening.

SMT-LIB

- SMT-LIB is the standard input format (Python API's exist for specific solvers)

- An SMT-LIB file must include:

  - Set-logic

  - Function declarations (variables are 0-ary functions)

  - Assertions

  - Check-sat command (and optionally get-model)

SMT-LIB

```
(set-logic LIA)
(declare-fun a () Int)
(declare-fun f (Int Bool) Int)
(assert (> a 10))
(assert (< (f a true) 100))
(check-sat)
(get-model)
```

SMT-LIB - example 5

```
(set-logic LIA)
(declare-fun a () Int)
(declare-fun f (Int Bool) Int)
(assert (> a 10))
(assert (< (f a true) 100))
(check-sat)
(get-model)
```

```
sat
(
  (define-fun a () Int
    11)
  (define-fun f ((x!0 Int) (x!1 Bool)) Int
    0)
)
```

Examples

- Solving Sudokus
- Verifying Code
- Verifying access policies for S2 buckets at Amazon
- Synthesising code!

Example 6 - Sudoku

		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

Example 6 - Sudoku

		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

All rows and columns contain the numbers 1 to 9, only once each

Each box contains the numbers 1 to 9, only once each

SMT-LIB - uninterpreted functions

- Functions have no side effects and are total (defined on all input values). No exceptions!

```
(declare-fun A (Int Int) Int)
```

SMT-LIB - arrays

- Functions have no side effects and are total (defined on all input values). No exceptions!

```
(declare-fun A (Int Int) Int)
```

Could also use nested arrays instead:

```
(declare-fun A () (Array Int (Array Int Int)))
```

Arrays are not like arrays in C! More like functions!

SMT-LIB - constants

- Functions have no side effects and are total (defined on all input values). No exceptions!

```
(declare-fun A (Int Int) Int)
```

“declare-const” is syntax sugar for declaring a nullary symbol:

```
(declare-const A (Array Int (Array Int Int)))
```

Example 6 - Sudoku

		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

All rows and columns contain the numbers 1 to 9, only once each

Each box contains the numbers 1 to 9, only once each

```
(declare-fun A (Int Int) Int)
```

Example 6 - Sudoku

		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

All rows and columns contain the numbers 1 to 9, only once each

Each box contains the numbers 1 to 9, only once each

```
(declare-fun A (Int Int) Int)
(assert (and (<= 1 (A 1 1))(>= 9 (A 1 1))))
```

Example 6 - Sudoku

		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

All rows and columns contain the numbers 1 to 9, only once each

Each box contains the numbers 1 to 9, only once each

```
(declare-fun A (Int Int) Int)
(assert (and (<= 1 (A 1 1))(>= 9 (A 1 1))))
(assert (and (<= 1 (A 1 2))(>= 9 (A 1 2))))
```

Example 6 - Sudoku

		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

All rows and columns contain the numbers 1 to 9, only once each

Each box contains the numbers 1 to 9, only once each

```
(declare-fun A (Int Int) Int)
(assert (and (<= 1 (A 1 1))(>= 9 (A 1 1))))
(assert (and (<= 1 (A 1 2))(>= 9 (A 1 2))))
```

Example 6 - Sudoku

		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

All rows and columns contain the numbers 1 to 9, only once each

Each box contains the numbers 1 to 9, only once each

```
(declare-fun A (Int Int) Int)
(assert (and (<= 1 (A 1 1))(>= 9 (A 1 1))))
(assert (and (<= 1 (A 1 2))(>= 9 (A 1 2))))
…
(assert (distinct (A 1 1)(A 1 2)(A 1 3)(A 1
4)(A 1 5)(A 1 6)(A 1 7)(A 1 8)(A 1 9)))
```

Example 6 - Sudoku

		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

All rows and columns contain the numbers 1 to 9, only once each

Each box contains the numbers 1 to 9, only once each

```
(declare-fun A (Int Int) Int)
(assert (and (<= 1 (A 1 1))(>= 9 (A 1 1))))
(assert (and (<= 1 (A 1 2))(>= 9 (A 1 2))))
…
(assert (distinct (A 1 1)(A 1 2)(A 1 3)(A 1
4)(A 1 5)(A 1 6)(A 1 7)(A 1 8)(A 1 9)))
…
(assert (= 9 (A 1 3)))
```

Example 6 - Sudoku

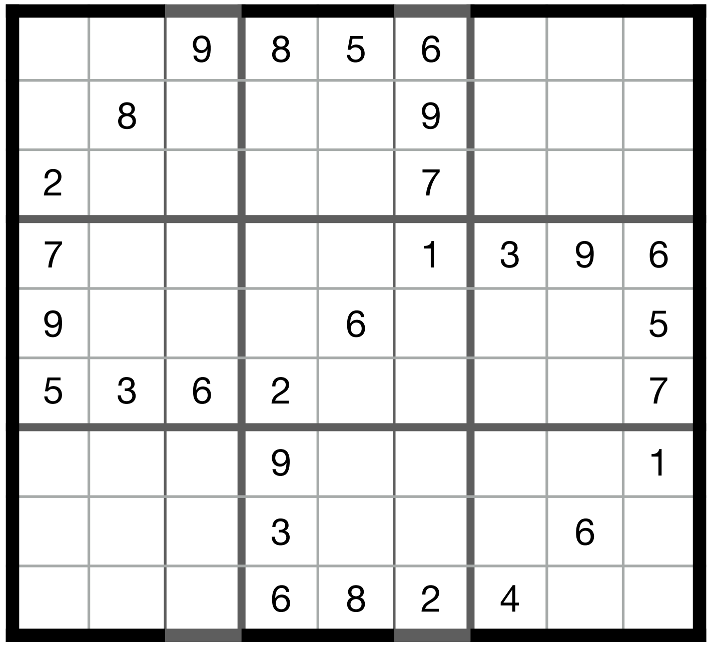
		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

All rows and columns contain the numbers 1 to 9, only once each

Each box contains the numbers 1 to 9, only once each

```
(declare-fun A (Int Int) Int)
(assert (and (<= 1 (A 1 1))(>= 9 (A 1 1))))
(assert (and (<= 1 (A 1 2))(>= 9 (A 1 2))))
…
(assert (distinct (A 1 1)(A 1 2)(A 1 3)(A 1
4)(A 1 5)(A 1 6)(A 1 7)(A 1 8)(A 1 9)))
…
(assert (= 9 (A 1 3)))
(check-sat)(get-model)
```

Example 6 - Sudoku



		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

All rows and columns contain the numbers 1 to 9, only once each

Each box contains the numbers 1 to 9, only once each

Solves the conjuction of all assertions!

```
(declare-fun A (Int Int) Int)
(assert (and (<= 1 (A 1 1))(>= 9 (A 1 1))))
(assert (and (<= 1 (A 1 2))(>= 9 (A 1 2))))
…
(assert (distinct (A 1 1)(A 1 2)(A 1 3)(A 1
4)(A 1 5)(A 1 6)(A 1 7)(A 1 8)(A 1 9)))
…
(assert (= 9 (A 1 3)))
(check-sat)(get-model)
```

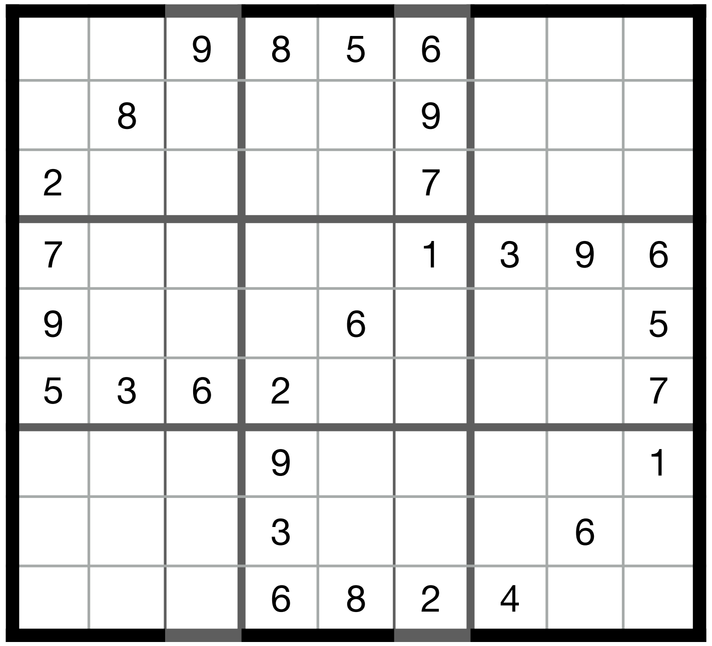
SMT-LIB - push/pop

```
(declare-const x Int)
(declare-const y Int)
(declare-const z Int)
(push)
(assert (= (+ x y) 10))
(assert (= (+ x (* 2 y)) 20))
(check-sat)
(pop) ; remove the two assertions
(push)
(assert (= (+ (* 3 x) y) 10))
(assert (= (+ (* 2 x) (* 2 y)) 21))
(check-sat)
```

SMT-LIB - push/pop

```
(declare-const x Int)
(declare-const y Int)
(declare-const z Int)
(push)
(assert (= (+ x y) 10))
(assert (= (+ x (* 2 y)) 20))
(check-sat)
(pop) ; remove the two assertions
(push)
(assert (= (+ (* 3 x) y) 10))
(assert (= (+ (* 2 x) (* 2 y)) 21))
(check-sat)
(declare-const p Bool)
(pop)
(assert p) ; error
```

SMT-LIB - bitvectors



		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

- No notion of signed-ness
- Overflow wraps around
- Divide by zero gives FFFFF

```
(declare-fun A ((_ BitVec 3)(_ BitVec 3))(_ BitVec 3))
```

SMT-LIB - bitvectors

		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

- No notion of signed-ness
- Overflow wraps around
- Divide by zero gives FFFFF

```
(declare-fun A ((_ BitVec 3)(_ BitVec 3))(_ BitVec 3))

(assert (bvuge (_ bv9 3)(A (_ bv1 3) (_ bv1 3)))
```

CBMC - example 3

```
bool x;
char y=8, z=0, w=0;

if(x)
  z = y-1;
else
  w = y+1;

assert(z==7 || w==9);
```

$$(y = 8) \wedge (w = 0) \wedge (z = 0) \quad \wedge$$
$$(z = x \,?\, y - 1 : 0) \quad \wedge$$
$$(w = x \,?\, 0 : y + 1) \quad \wedge$$
$$(z \neq 7) \quad \wedge$$
$$(w \neq 9)$$

How do we represent char in SMT?

SMT-LIB - quantifiers

		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

- Compact
- SMT solvers support first-order quantification
- Careful not to write things that are trivially false!

```
(declare-fun A (Int Int) Int)

(assert (forall ((i Int)(j Int)) (=> (and (<= i 9)(>=
i 0)(<= j 9)(>= j 0)) (and (<= (A i j) 9)(>= (A i j)
1)))))
```

SMT-LIB - define-fun

		9	8	5	6			
	8				9			
2					7			
7					1	3	9	6
9				6				5
5	3	6	2					7
			9					1
			3				6	
			6	8	2	4		

- Functions can be defined, useful to make things more compact

```
(define-fun inRange ((x Int)) Int
 (and (<= 1 x)(>= 9 x)))

(declare-fun A (Int Int) Int)
(assert (and (inRange (A 1 1)) (inRange (A 1
2) …))
```

Other ways to build SMT files

- Python API's available for specific solvers

  https://ericpony.github.io/z3py-tutorial/guide-examples.htm

- Using other tools, e.g., UCLID5, CBMC

What next?

- SMT for verifying permissions at Amazon
- SMT for synthesis
- UCLID5: useful modeling tool to generate SMT- and synthesis queries