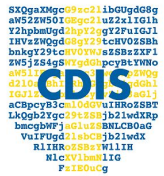


CENTER FOR
CYBER DEFENCE AND
INFORMATION SECURITY

KTH
VETENSKAP
OCH KONST

Provable Security

Roberto Guanciale, KTH

CDIS Spring Conference 2023
KTH Royal Institute of Technology
Thursday, May 25, 2023

Side channel attacks

- Power consumption, Electromagnetic radiation, Sound, Temperature, Timing

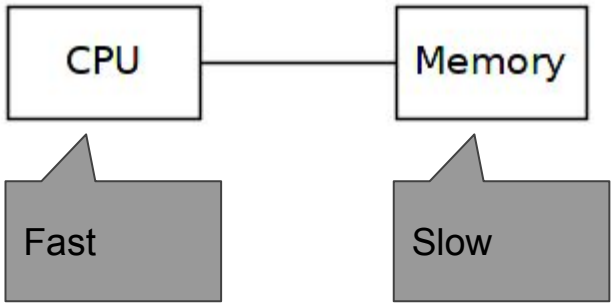
Timing side channel

```
check-pin(pin):
    for i in 0..3
        if pin[i] != pwd[i]:
            return False
    return True
```

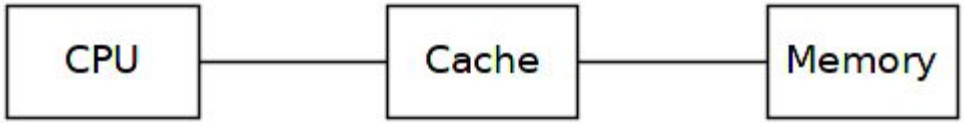
- 10K combinations
  - Avg 5K combinations
  - Probability guess in three attempts 3/10000
- Timing attack

```
T1=time(check-pin(pin1))
pin2 = pin1[0]=x
T2=time(check-pin(pin2))
if T1=T2 pin1[0] and pin[1] wrong
if T1>T2 pin1[0] correct
if T1<T2 pin2[0] correct
```

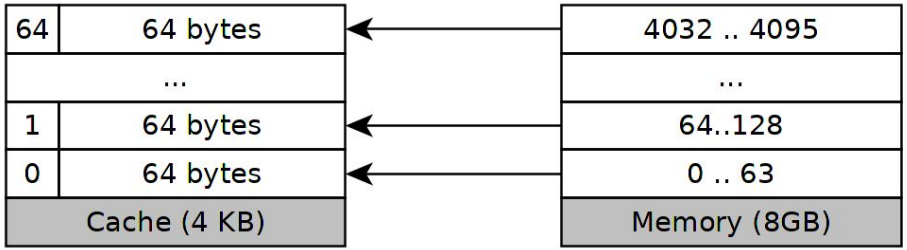
CPU architecture



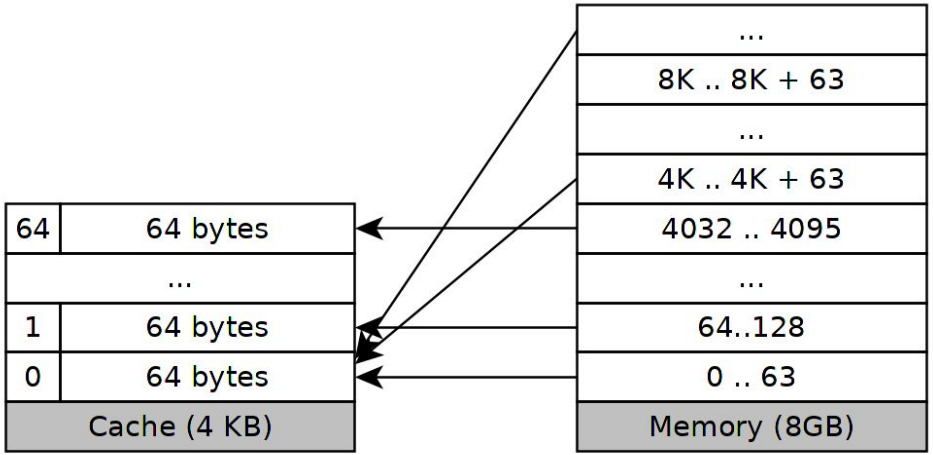
CPU architecture



Cache

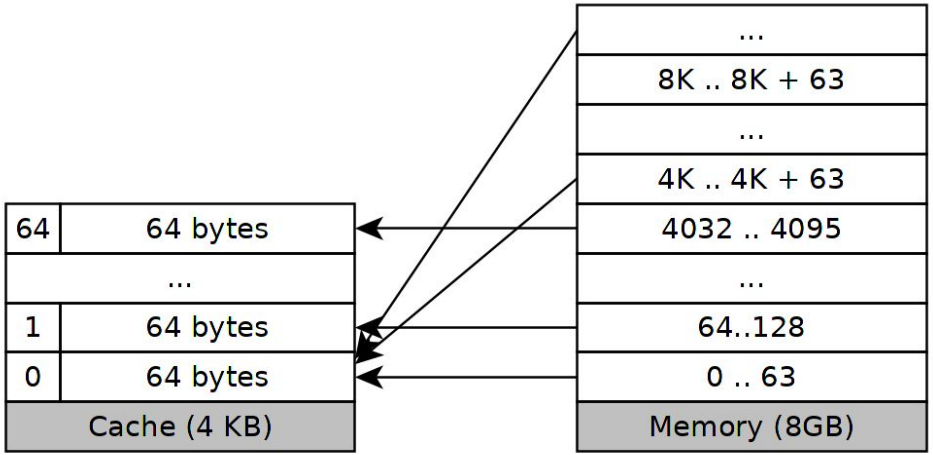

64
64 bytes
...
1
64 bytes
0
64 bytes
Cache (4 KB)
4032 .. 4095
...
64..128
0 .. 63
Memory (8GB)

Cache


...
8K .. 8K + 63
...
4K .. 4K + 63
4032 .. 4095
...
64..128
0 .. 63
Memory (8GB)
64
64 bytes
...
1
64 bytes
0
64 bytes
Cache (4 KB)

Cache

- When we access address 0, CPU
  - fetches 64 bytes into line 0
- When we load address 4KB (cache miss), CPU
  - evicts line 0
  - if dirty, writes back the content
- Multiway, VT/PT index, write through, …
- Cache are transparent
- Cache are shared
- There are collisions
- Cache misses require more time



Side channel attacks

- Power consumption, Electromagnetic radiation, Sound, Temperature, Timing (caches)
- Difficult to audit
- Security condition:
  - Execution from two states that should be indistinguishable for the attacker should have the same footprint on the channel

Timing side channel

```
check-pin(pin):
    for i in 0..3
        if pin[i] != pwd[i]:
            return False
    return True
```

Here two states should have the same pin, but different pwd

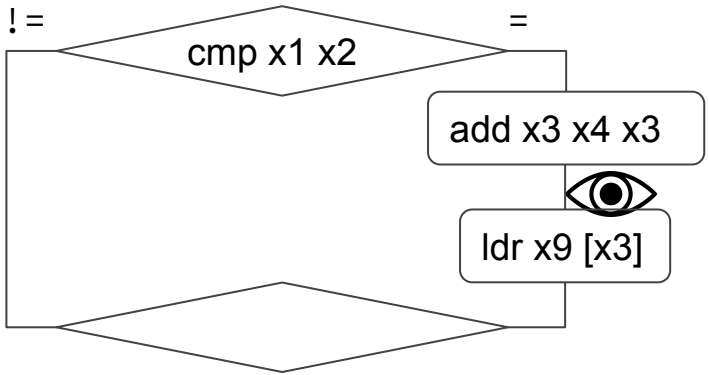
Side channel attacks

- Power consumption, Electromagnetic radiation, Sound, Temperature, Timing (caches)
- Difficult to audit
- Security condition:
  - Execution from two states that should be indistinguishable for the attacker should have the same footprint on the channel

Unfeasible to have precise deterministic models of these channels.

- Too complex
- Many undocumented features (e.g. cache replacement policies)
- Different processors / same ISA / different channels

Observational models



Verify absence of side-channel leakage

- Via constant-time (observation) programming policy (J.B. Almeida et al.)

Require abstract attacker observations

- I.e. a model of what an attacker may see (over approximation)

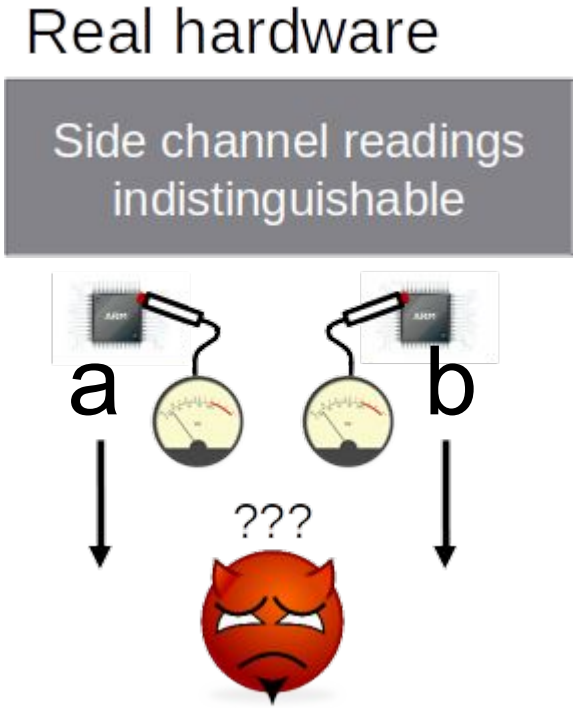
Side channel attacks

- Power consumption, Electromagnetic radiation, Sound, Temperature, Timing (caches)
- Difficult to audit
- Security condition:
  - Execution from two states that should be indistinguishable for the attacker should have the same footprint on the channel
- Security condition’:
  - Execution from two states that should be indistinguishable for the attacker should have the same observations

Side Channel Abstract Model Validation

$P(a) \sim_{\text{eye}} P(b) \quad ==>$

- Are the existing models sound?
- Program may be wrongly considered secure



Are the existing models sound? No, Spectre! (P. Kocher et al.)

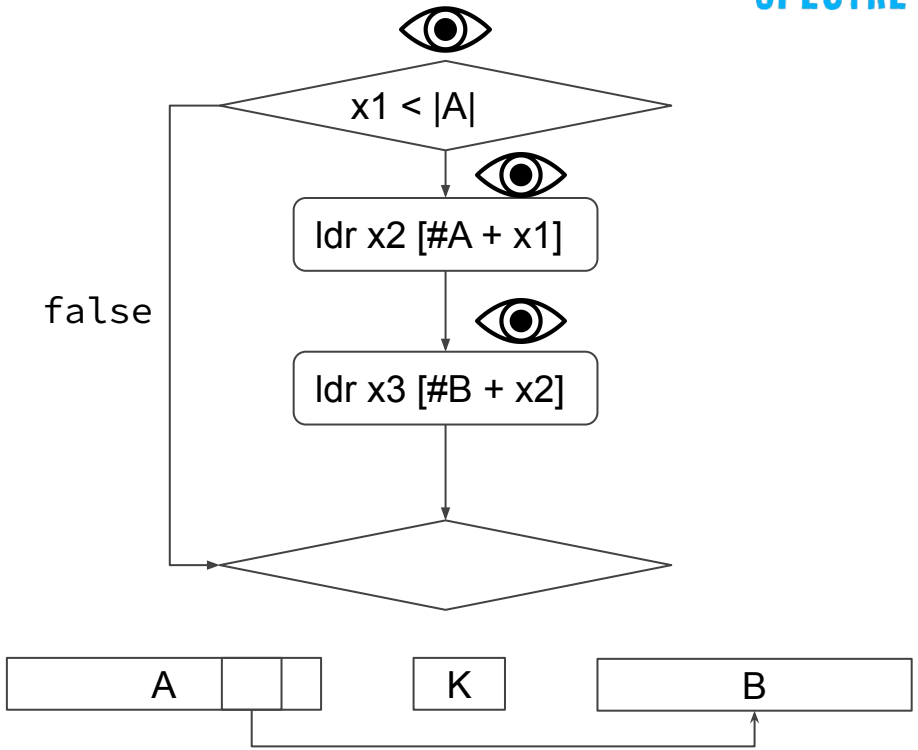
Assume for every every i

- 0 <= A[i] < |B|

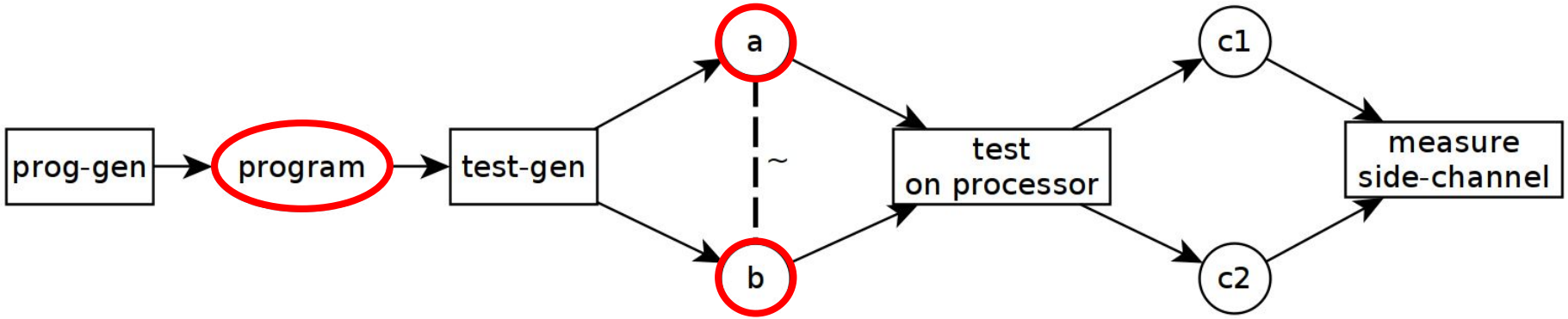
Assumptions and condition ensure no out-of-bound memory read.

Observations depend on

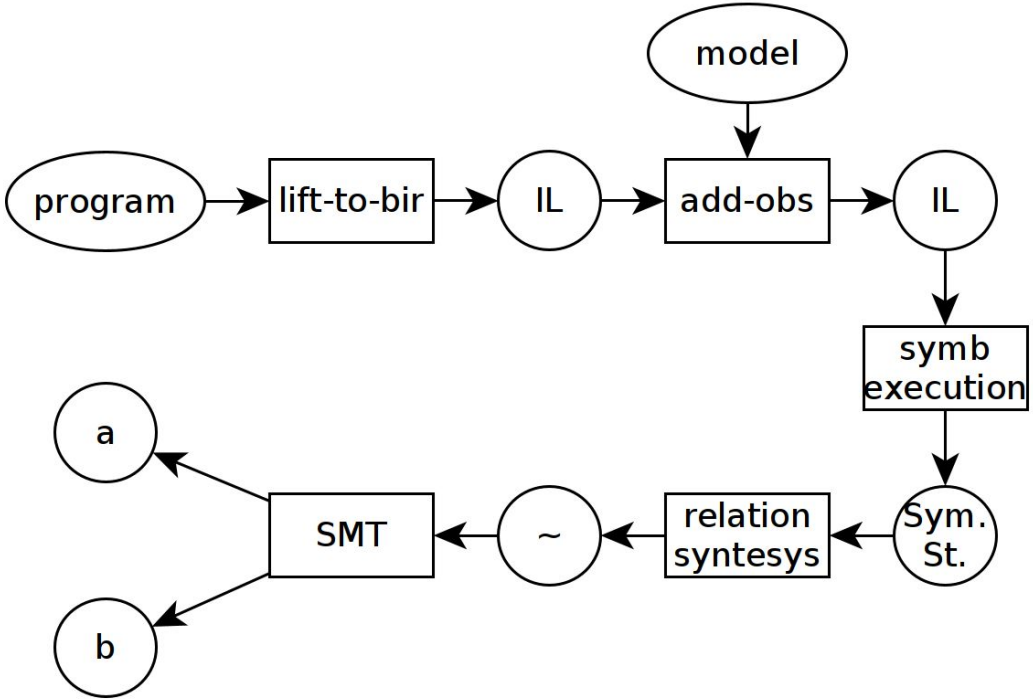
- Position of A and B
- Size of A and B
- Content of A and B
- Input x1



SCAM-V: Validation via testing

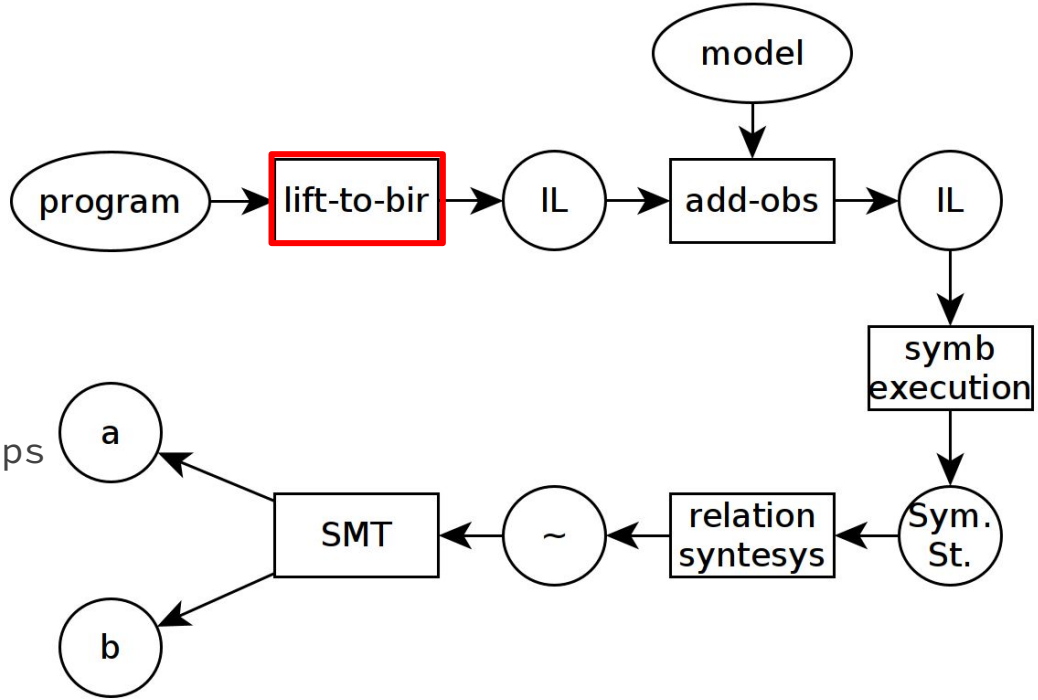


test-gen


model
program
lift-to-bir
IL
add-obs
IL
symb
execution
a
SMT
~
relation
syntesys
Sym.
St.
b

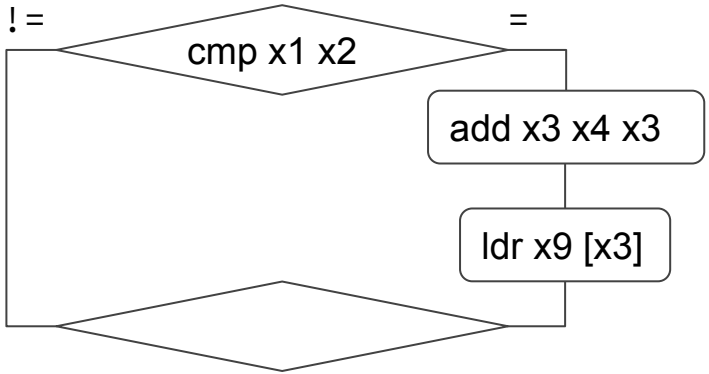
BIR

- Abstract Assembly Language
- Infinite number of register variables
- Assignments, jumps, cond. jumps



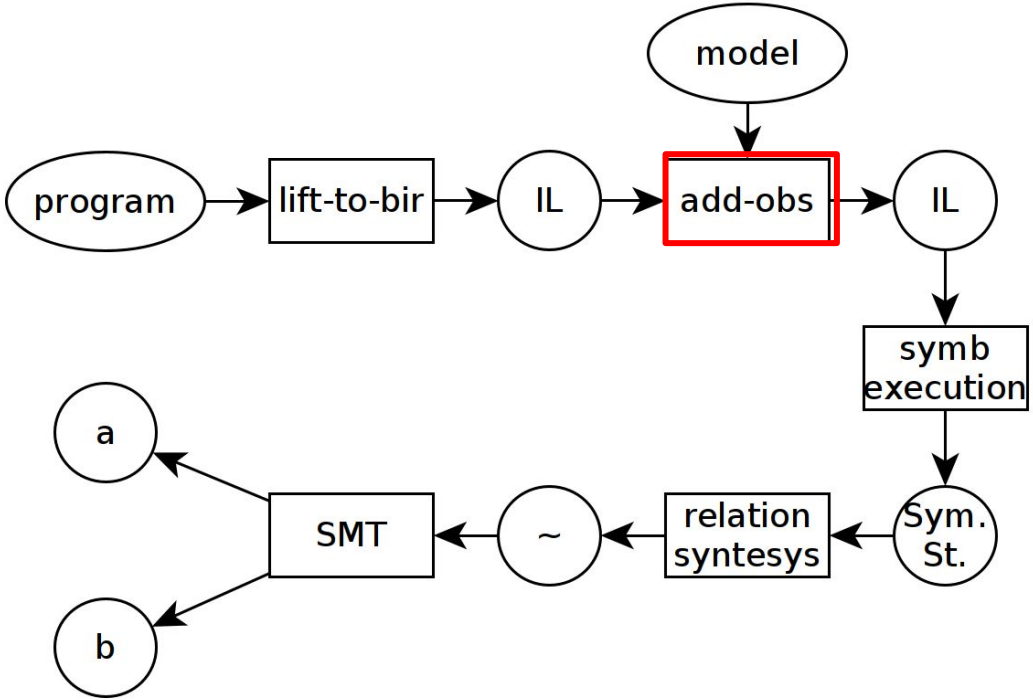
BIR

```
0 : CJMP x1=x2 4 12

4: x3 = x4 + x3; JMP 8

8 : x9 = MEM[x3]; JMP 12

12 : HALT
```



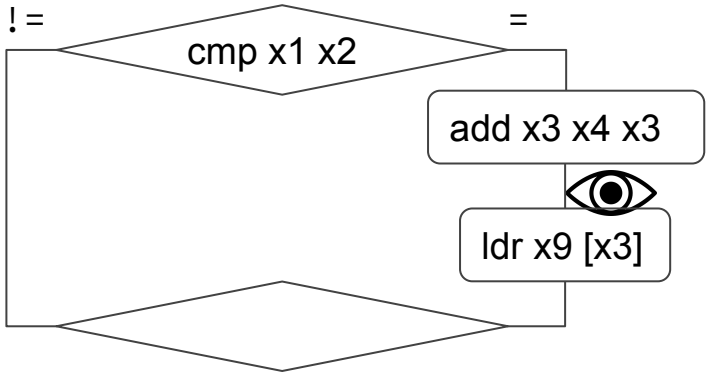
Add observations

- Program transformation that inlines observations
- Different transformations for different models



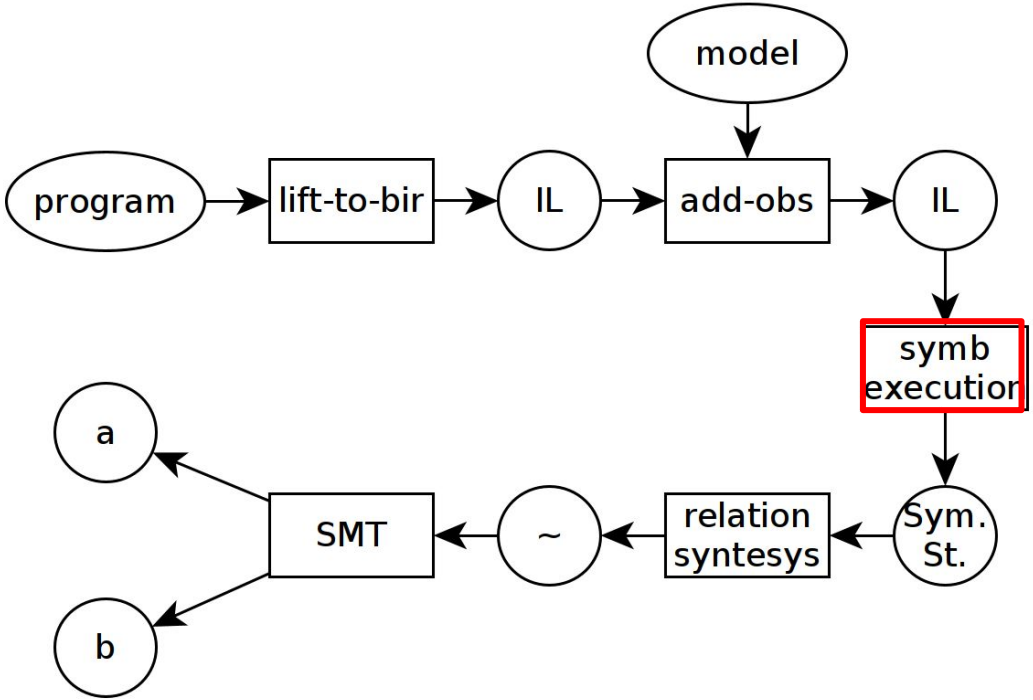
BIR

```
0 : CJMP x1=x2 4 12

4: x3 = x4 + x3; JMP 8

8 : Obs(x3); x9 = MEM[x3]; JMP 12

12 : HALT
```

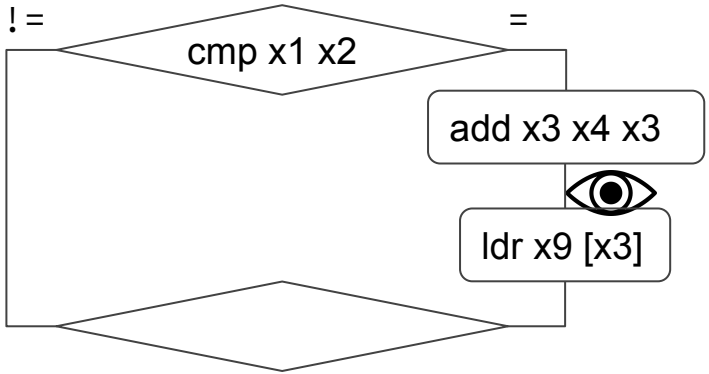


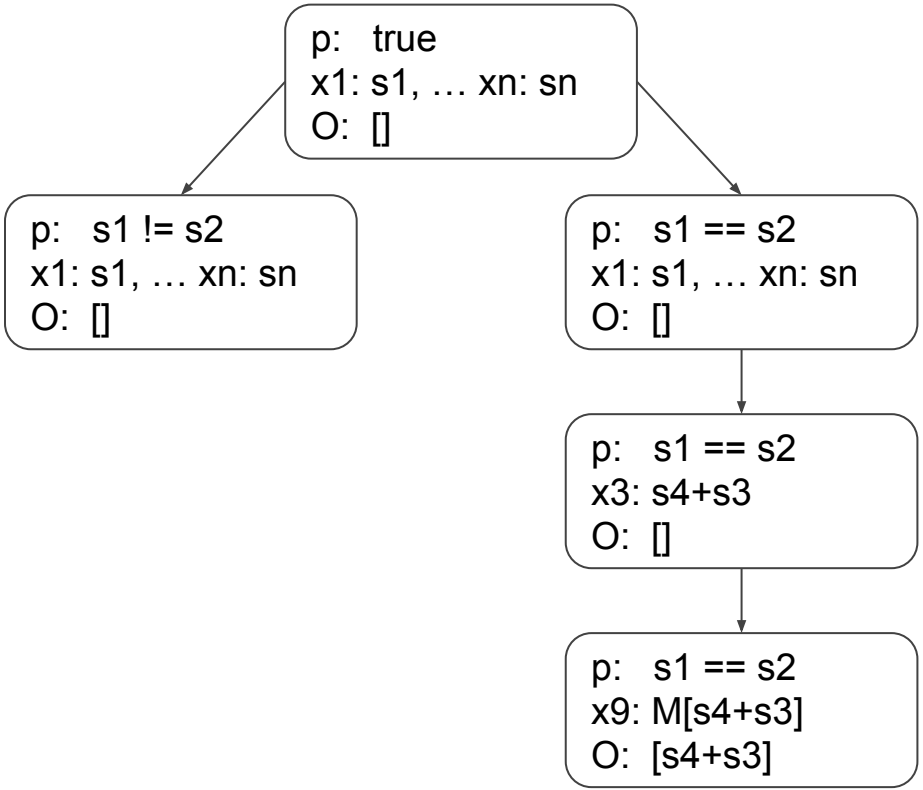
Symbolic execution

- States have symbolic path and symbolic assignments
- Plus a symbolic observation list



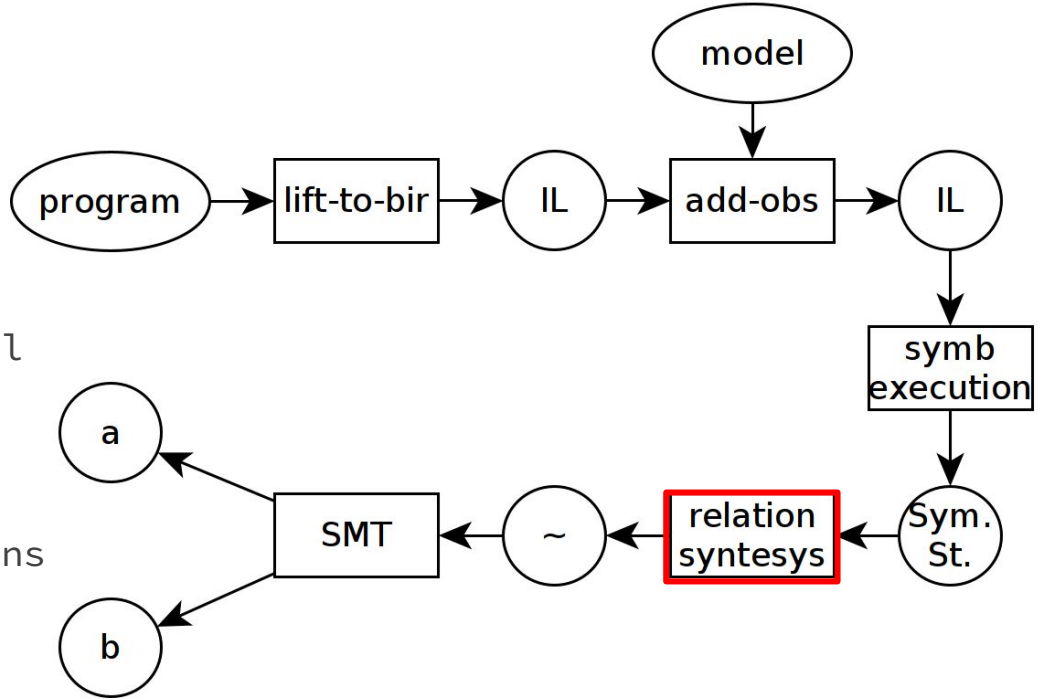
Symbolic execution





Relation synthesis

- Self composition
- Cartesian product of the final symbolic states (i.e. all possible pair of execution paths)
- Impose equality of observations



Relation synthesis

p: s1 == s2
...
O: [s3+s4]

p: s1 != s2
...
O: []

p: s1 == s2
...
O: [s3+s4]

(a.x1 == a.x2)/\(b.x1 == b.x2)
==>
[a.x3+a.x4] == [b.x3+b.x4]

(a.x1 == a.x2) /\ (b.x1 != b.x2)
==>
[a.x3+a.x4] == []

p: s1 != s2
...
O: []

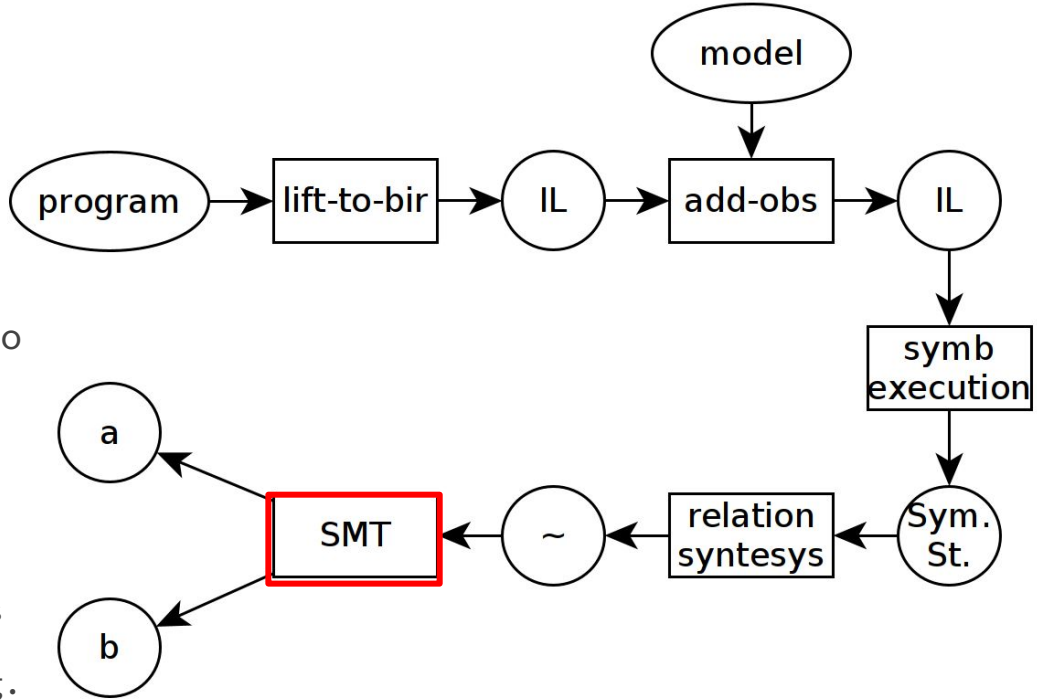
...

(a.x1 != a.x2) /\ (b.x1 != b.x2)
==>
[] == []

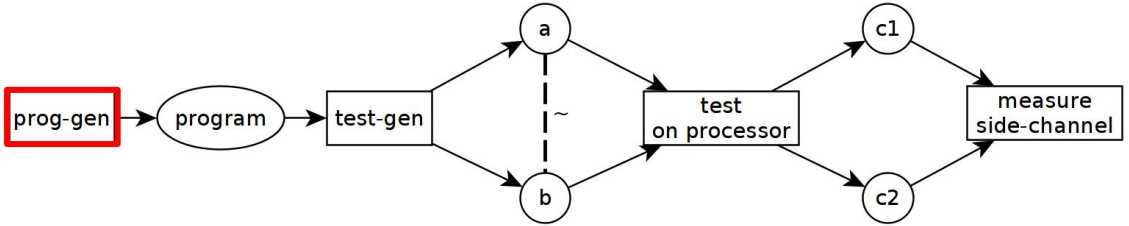
SMT Solver

- Z3 results in assignments to relevant variables for the two states

- Some additional constraints enforced by the lifter
  - E.g. only a part of memory is accessible for tests
  - Result in BIR assertions (e.g. for each memory load/store)
  - Stricter path conditions that lead to only executable test cases

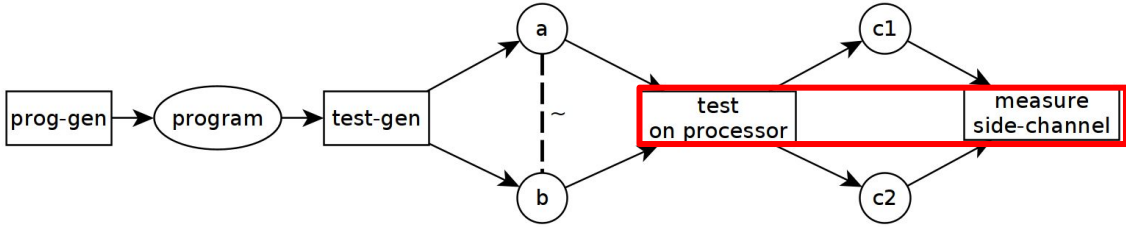


Program generation



- Generators can be composed
- Generators are randomly instantiated

Test and measure



- ARMv8 (Raspberry Pi 3)
- Execution coordinated by a server
- Bootloader
  - load program
  - sets-up environment (e.g. page tables)
  - sets-up state a
  - exec from state a
  - inspect cache
  - clean state
  - repeat for state b
- Implemented in TrustZone
  - execution bare to metal => no noise
  - instructions for cache inspection

How about spectre?

———

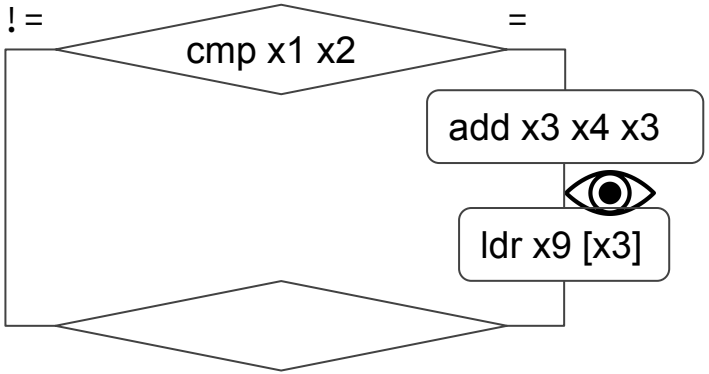
Raspberry Pi3 is claimed to be immune.

NO!

It seems to be immune to the original Spectre, but it is affected by other speculative leakage (i.e. the first load in the mispredicted branch can leak data)

16000 tests, 600 programs

- Without refinement 2 counterexamples
- With refinement 3971 counterexamples



Concluding remarks

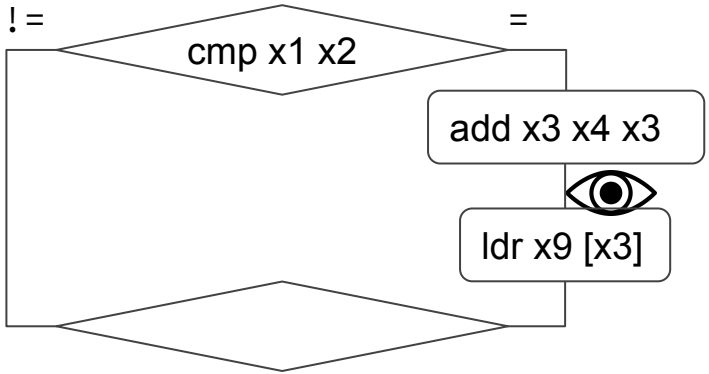
Summary:

- Existing observational model often overlook some information flows
- Without sound models we cannot reason about SW security
- Observation equivalence provides a good strategy to drive tests

New challenges:

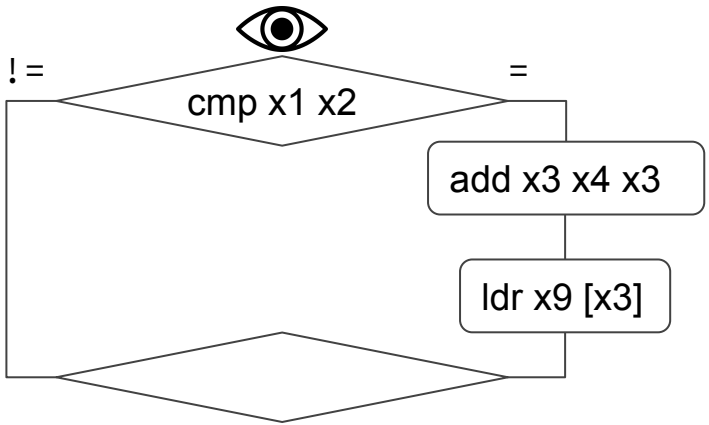
- Automatic model repair/improvement
- Hardware coverage

Observation equivalence classes



x1 != x2	x1 == x2 /\ x4+x3 = 0
	x1 == x2 /\ x4+x3 = 1
	...
	x1 == x2 /\ x4+x3 = 2^64

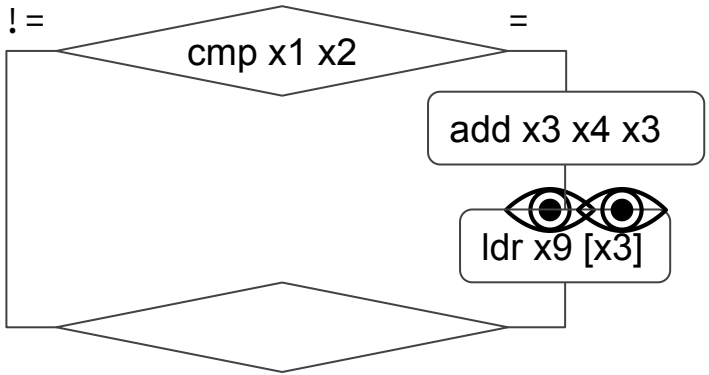
Observation equivalence classes



x1 != x2	x1 == x2 /\ x4+x3 = 0
	x1 == x2 /\ x4+x3 = 1
	...
	x1 == x2 /\ x4+x3 = 2^64

x1 != x2	x1 == x2

Observation equivalence classes



x1 != x2	M[0]=0	M[0]=1	...
	...		

x1 != x2	x1 == x2 /\ x4+x3 = 0
	x1 == x2 /\ x4+x3 = 1
	...
	x1 == x2 /\ x4+x3 = 2^64

x1 != x2	x1 == x2

Model Lattice