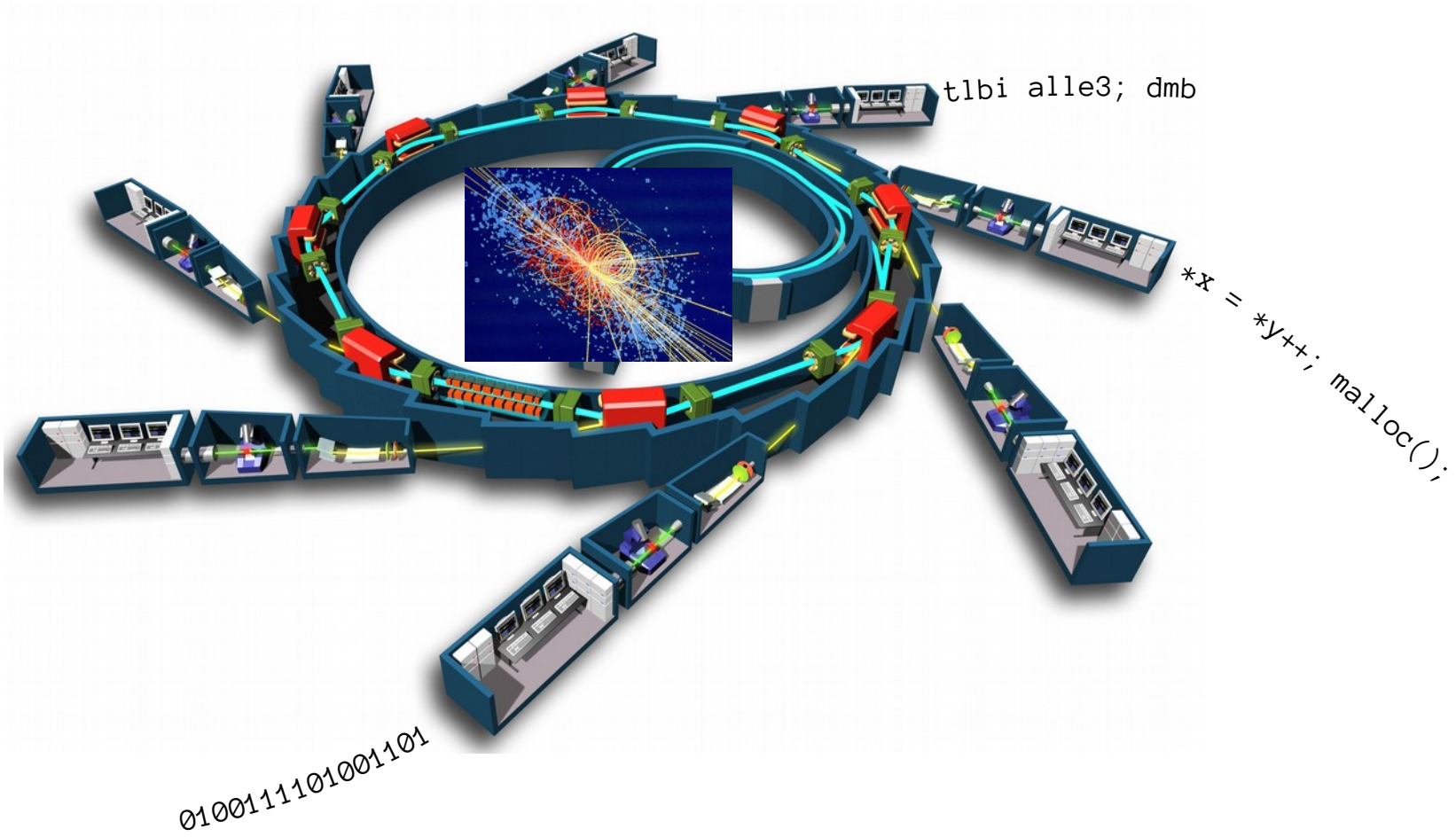
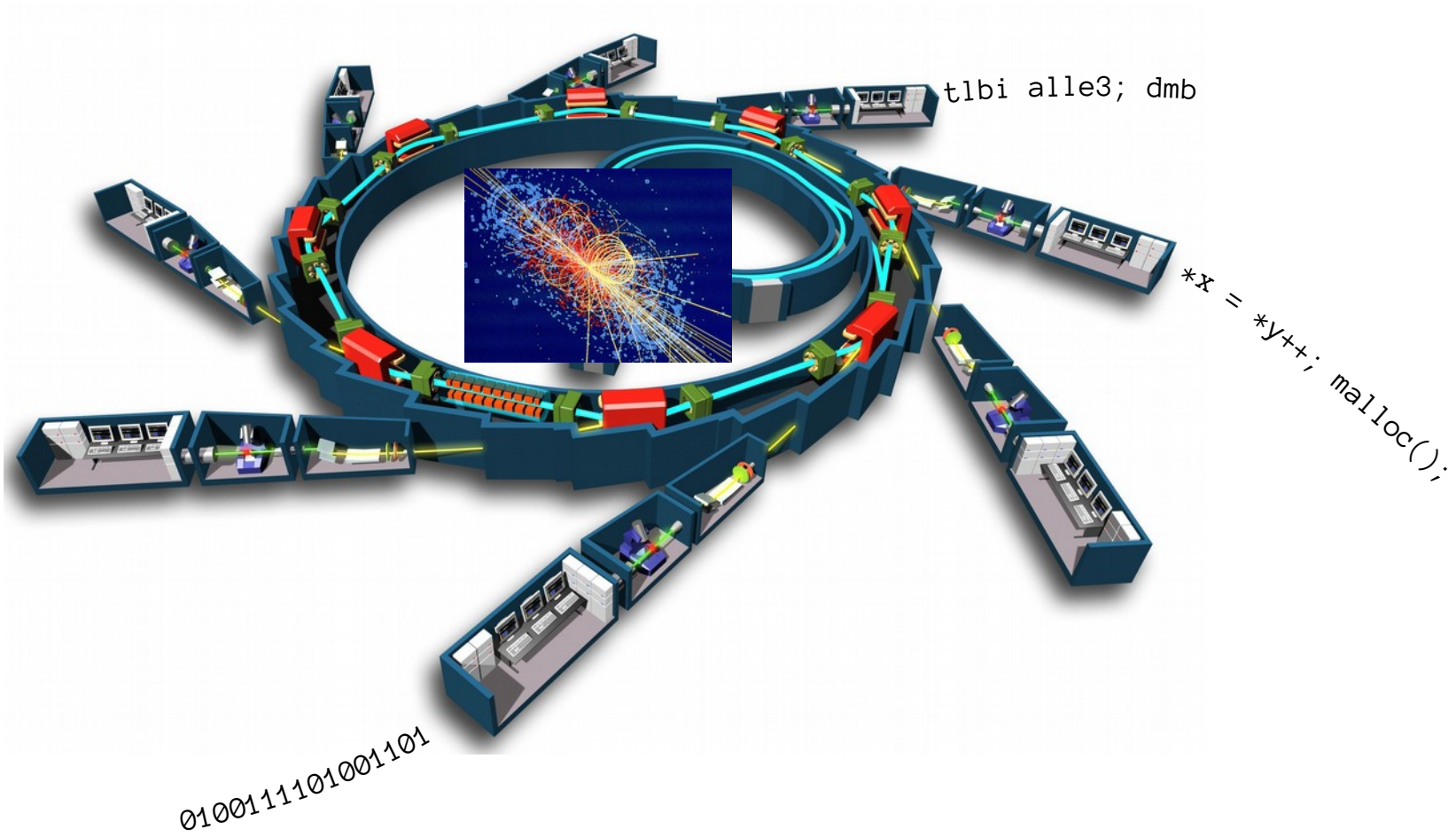
Runtime Verification

David Cock – david.cock@inf.ethz.ch

Systems@ETH Zürich



We're Building the *Large Program Collider*



Collide *instructions* at 0.99*c*, and observe the decay products.

Images: CERN; Chaix & Morel et associés

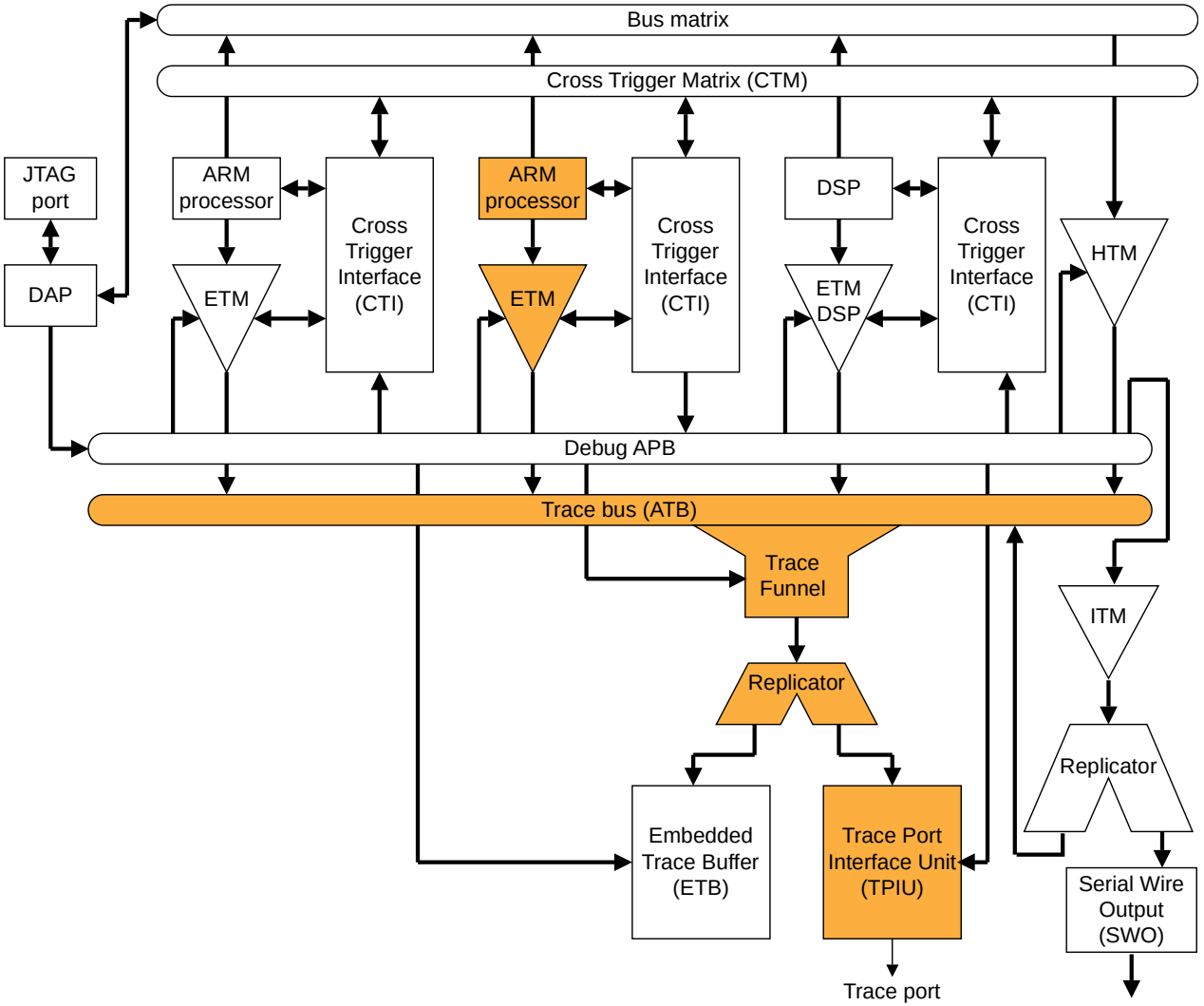
Why is This Useful?

- Formal verification relies on accurate models
- For systems-level HW, *these mostly don't exist*!
  - Testing lets us build confidence at these low levels

- **The hardware is *trying* to tell us what it's doing.**

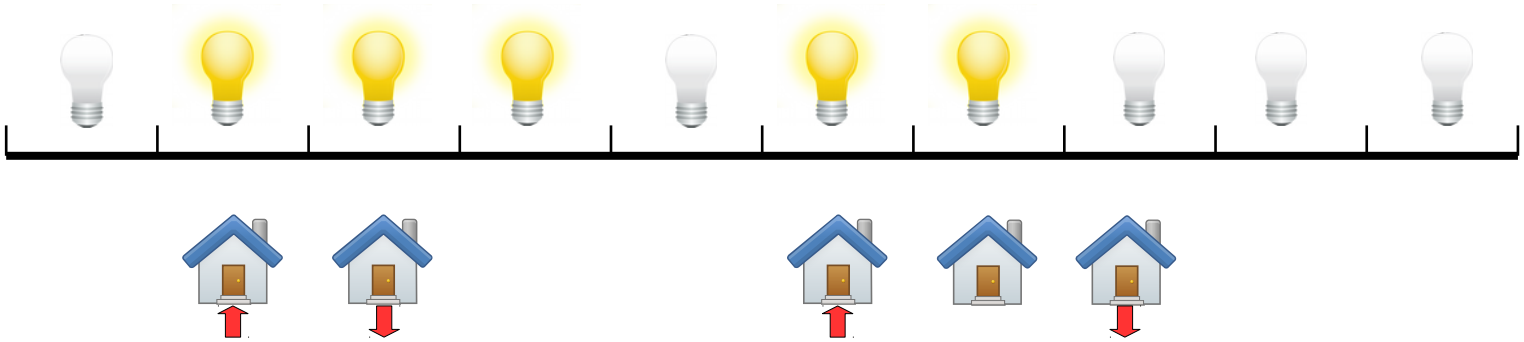
- Further applications:
  - Debugging rack-scale systems.
  - Monitoring control flow (security).

Program Trace



LTL and Automata

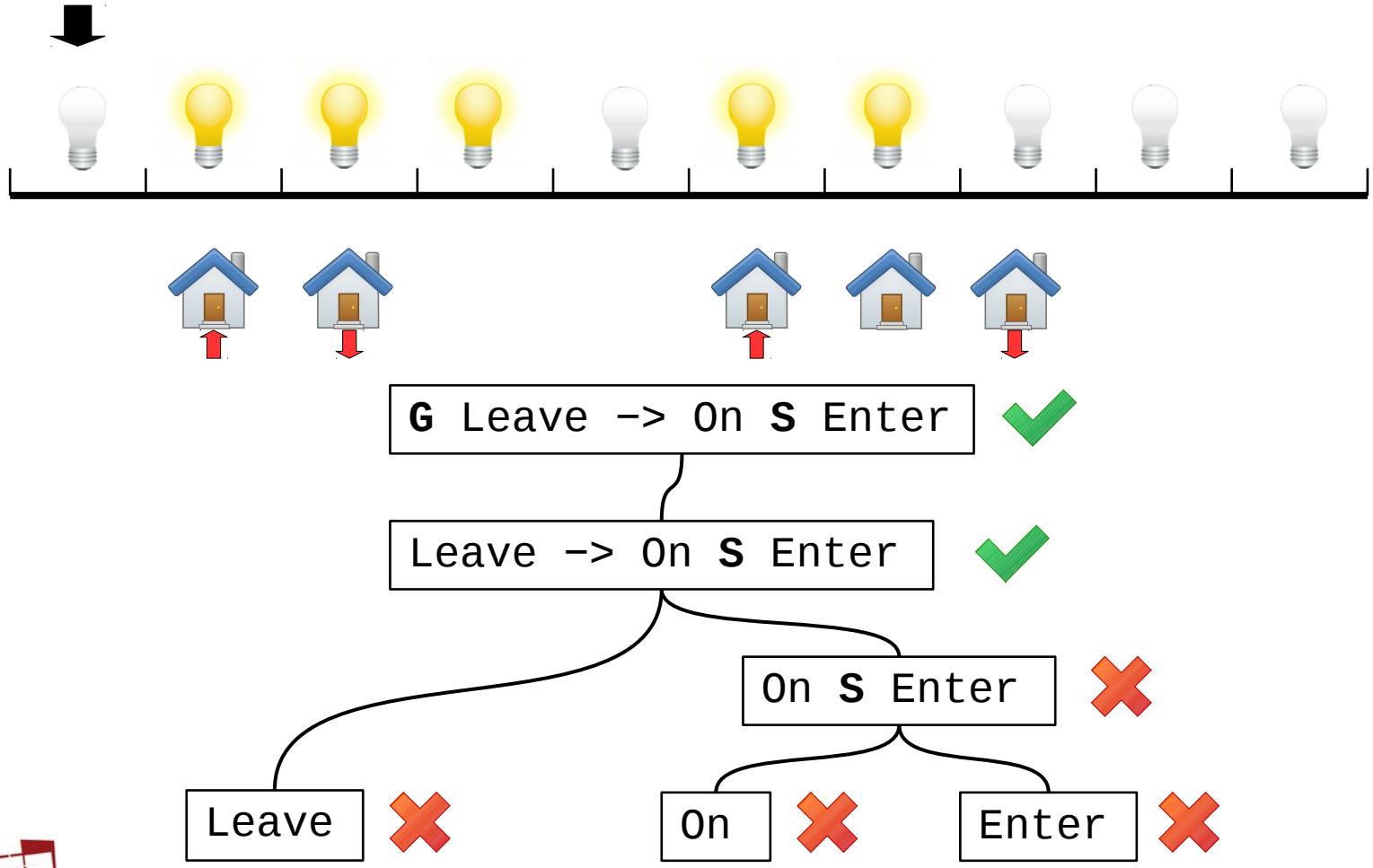
The light stays on until I leave the house.



```
G Leave -> On S Enter;
```

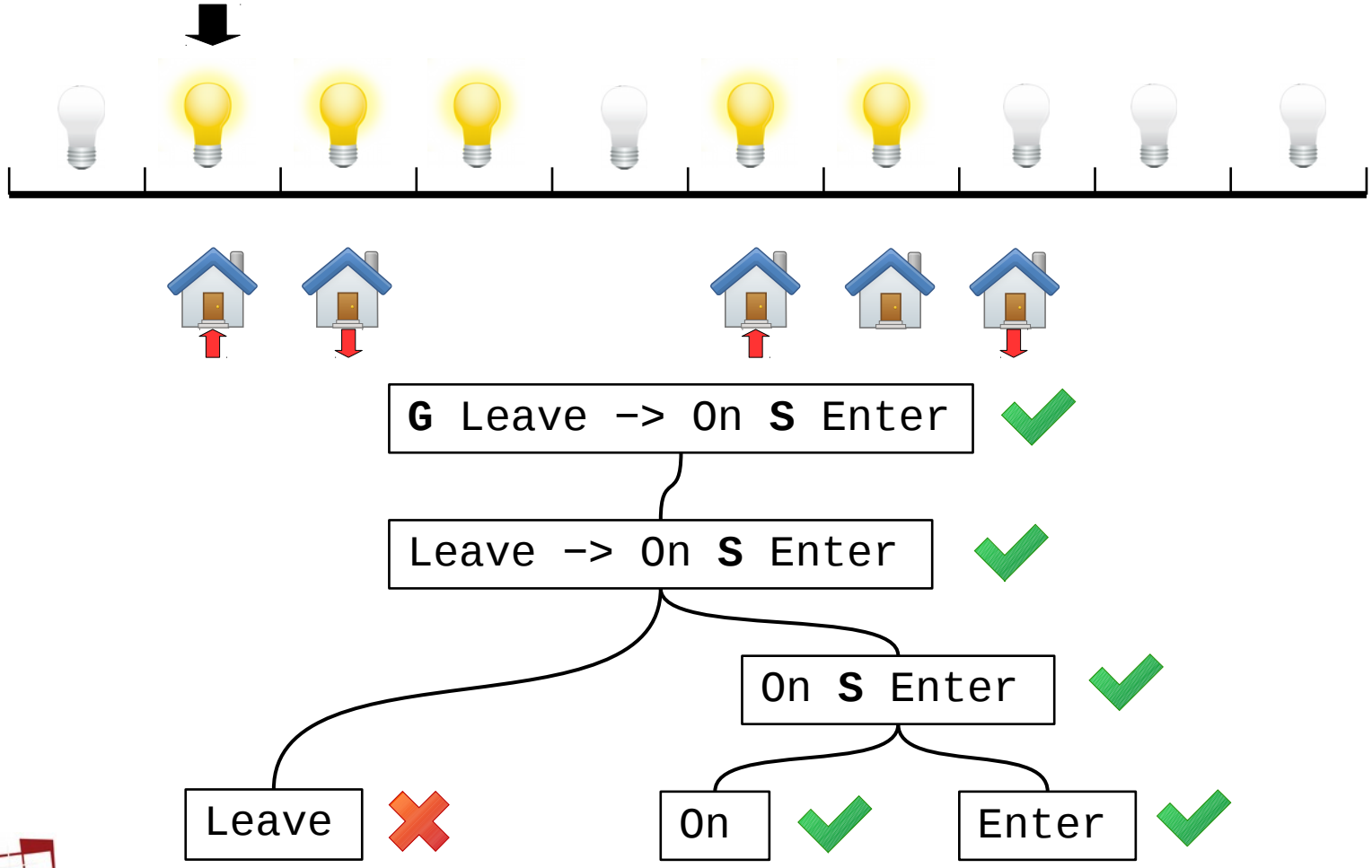
LTL and Automata

The light stays on until I leave the house.



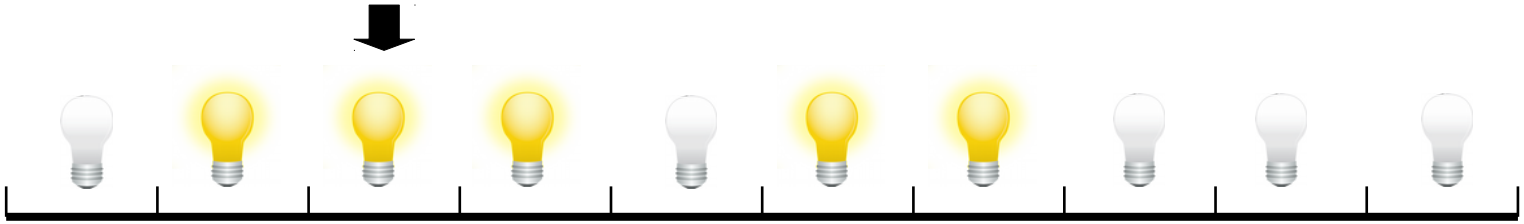
LTL and Automata

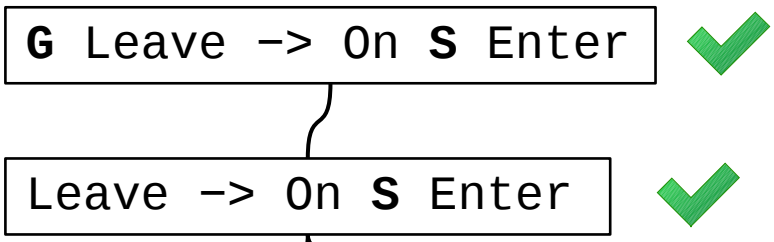
The light stays on until I leave the house.



LTL and Automata

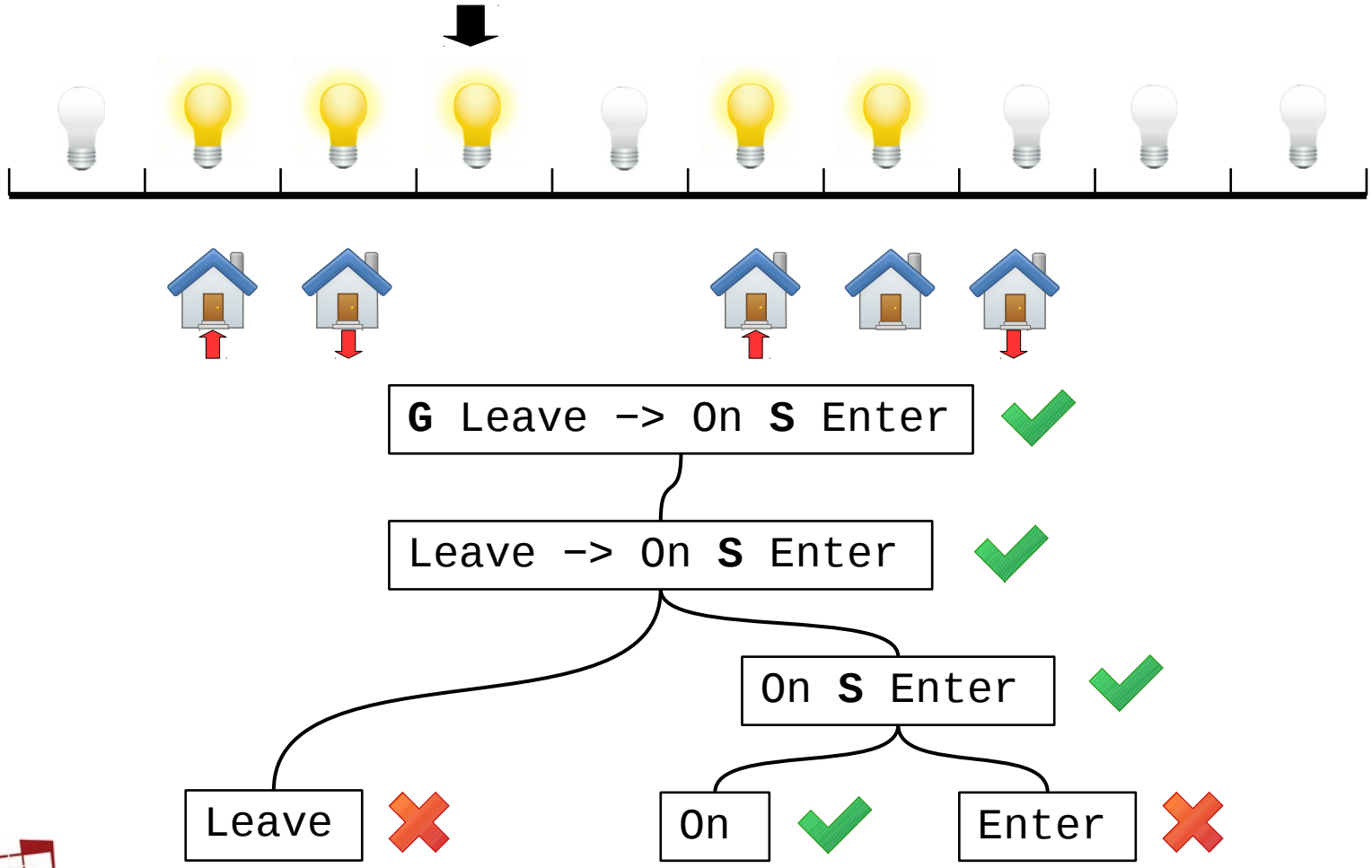
The light stays on until I leave the house.





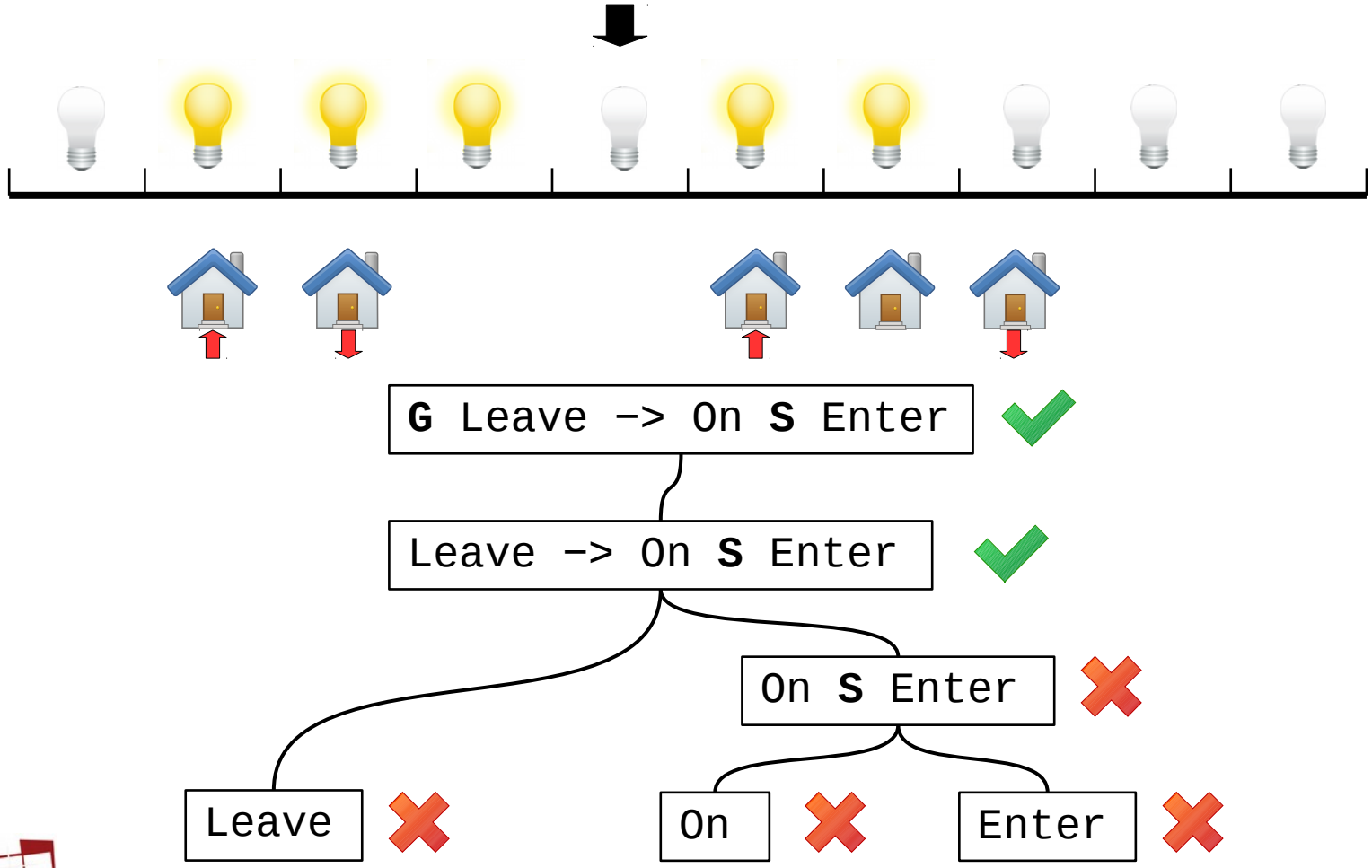
LTL and Automata

The light stays on until I leave the house.



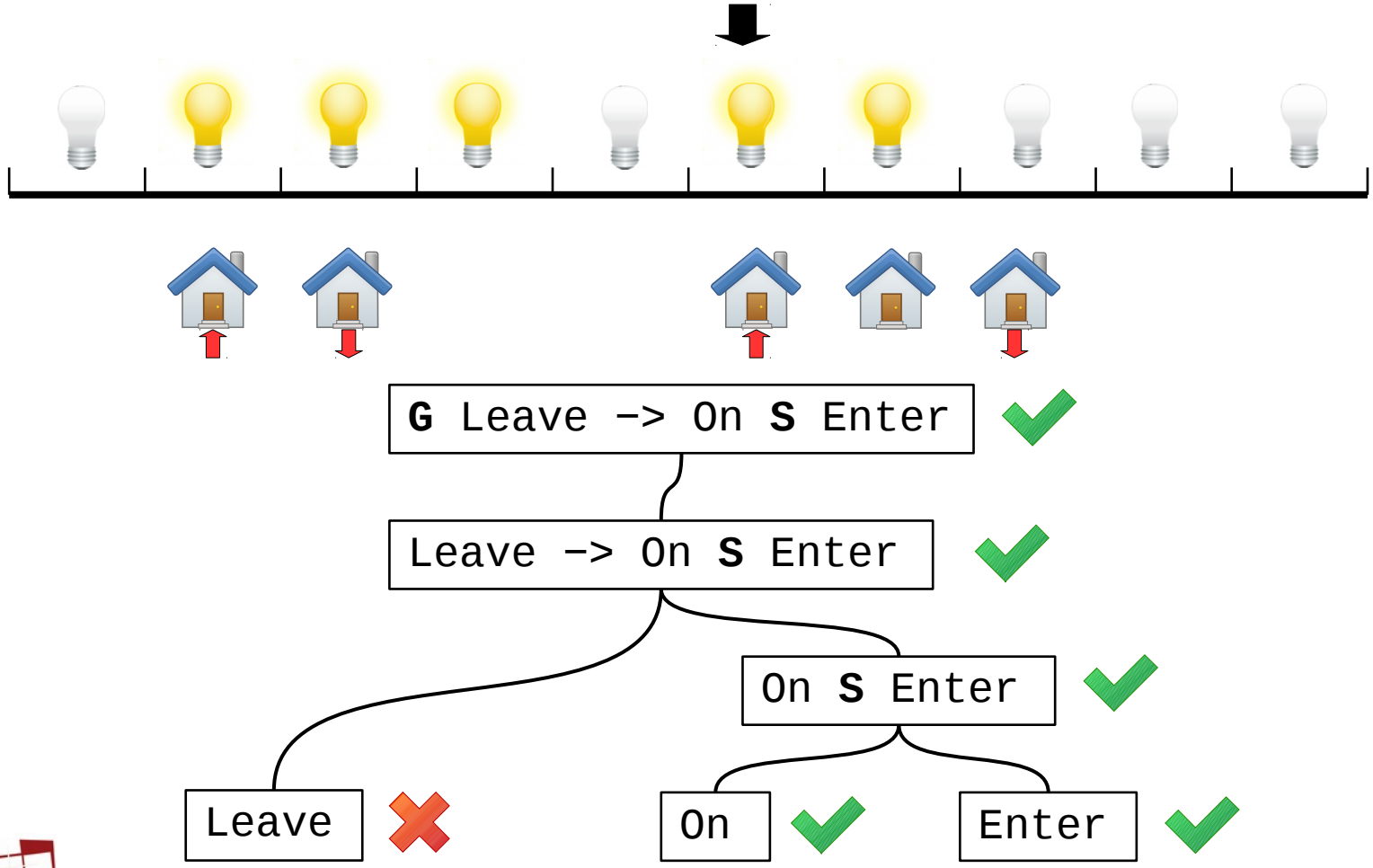
LTL and Automata

The light stays on until I leave the house.



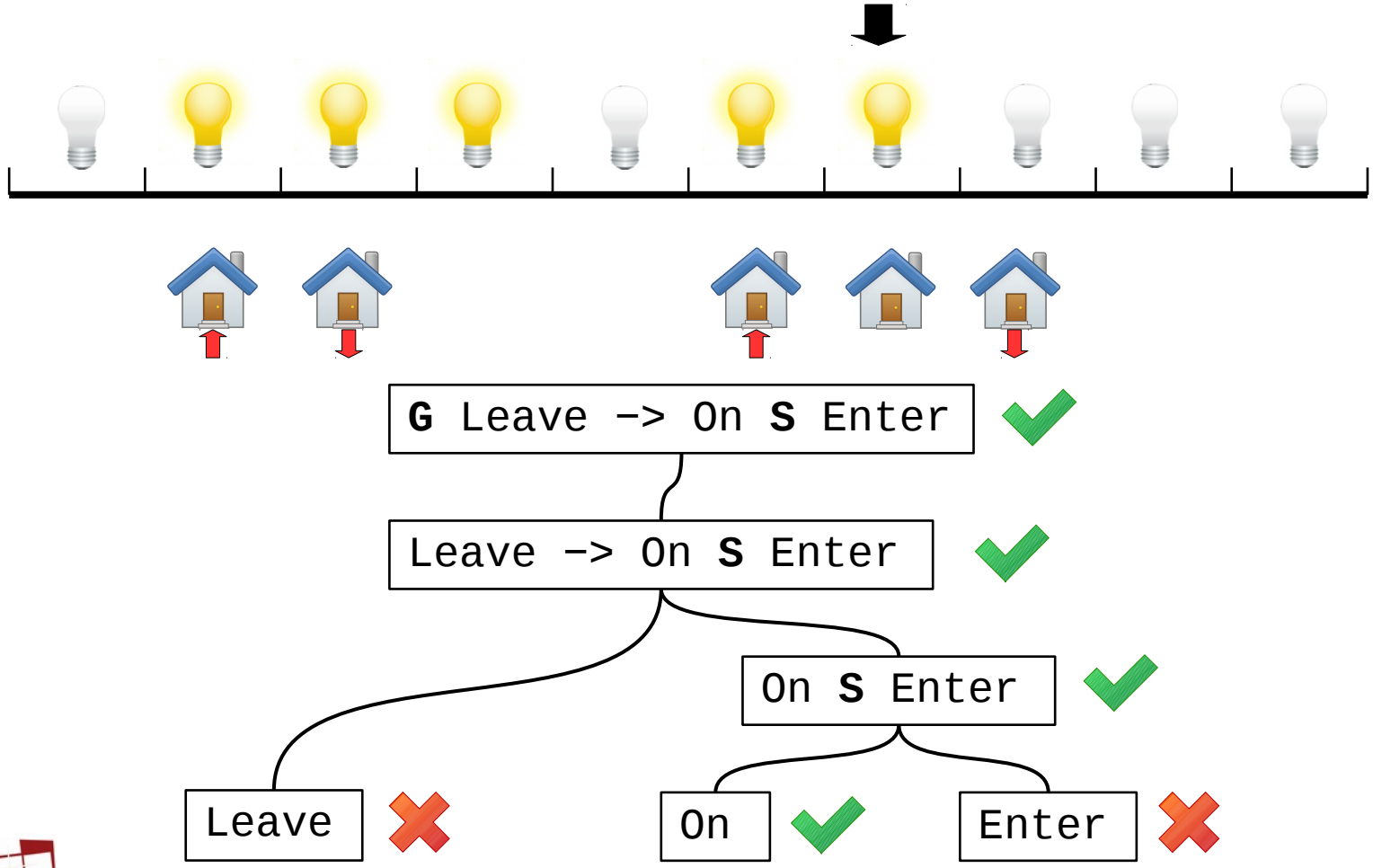
LTL and Automata

The light stays on until I leave the house.



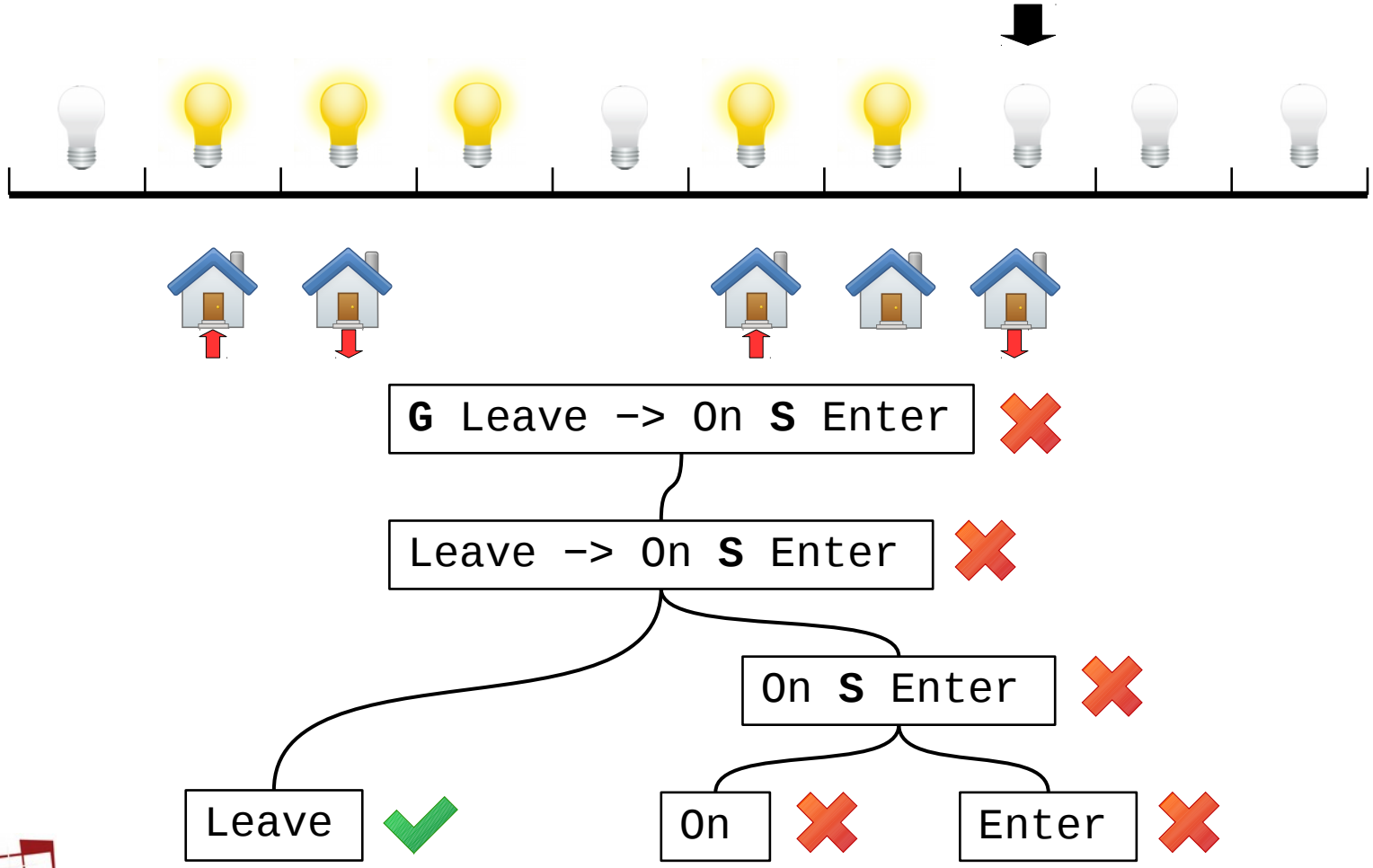
LTL and Automata

The light stays on until I leave the house.



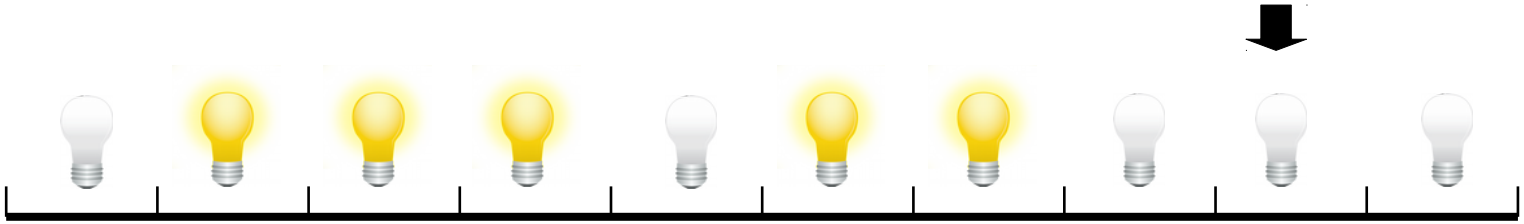
LTL and Automata

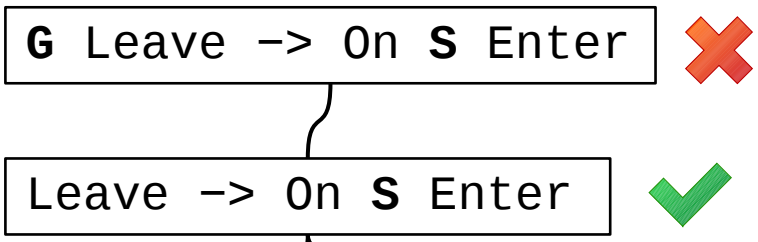
The light stays on until I leave the house.



LTL and Automata

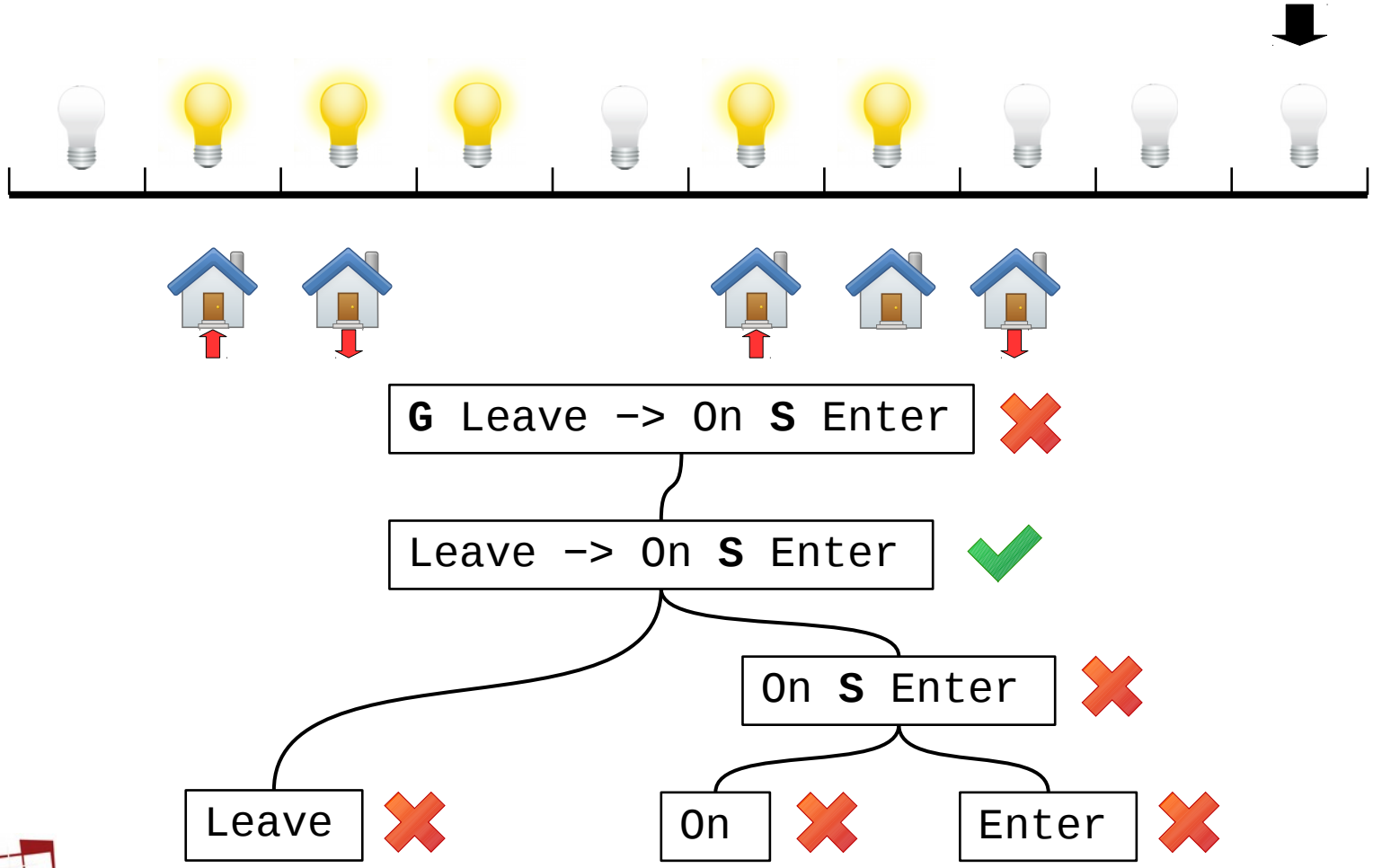
The light stays on until I leave the house.





LTL and Automata

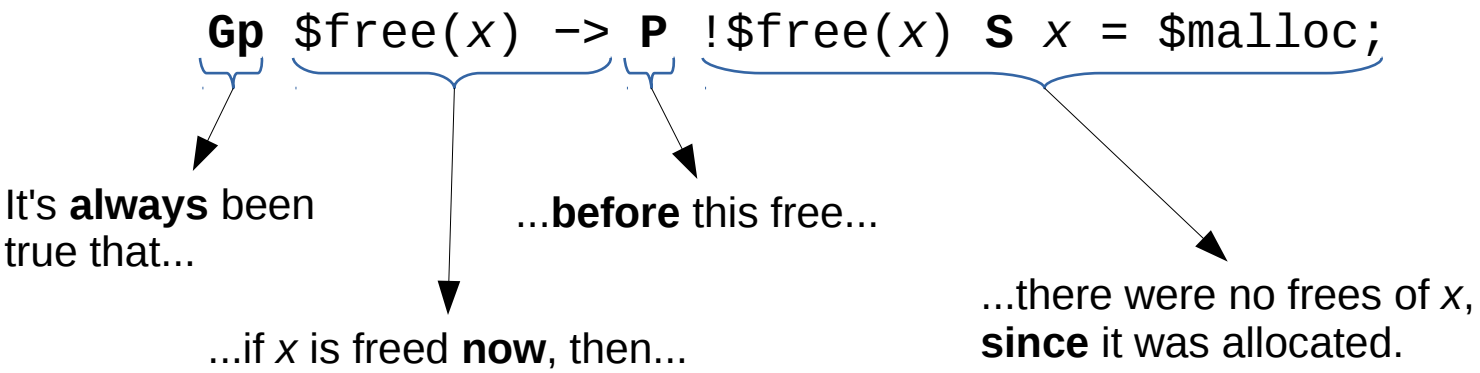
The light stays on until I leave the house.



Properties
No Double Frees in LTL

```
void *a = malloc();
...
{a is still allocated}
free(a);
```

- We can now check this:

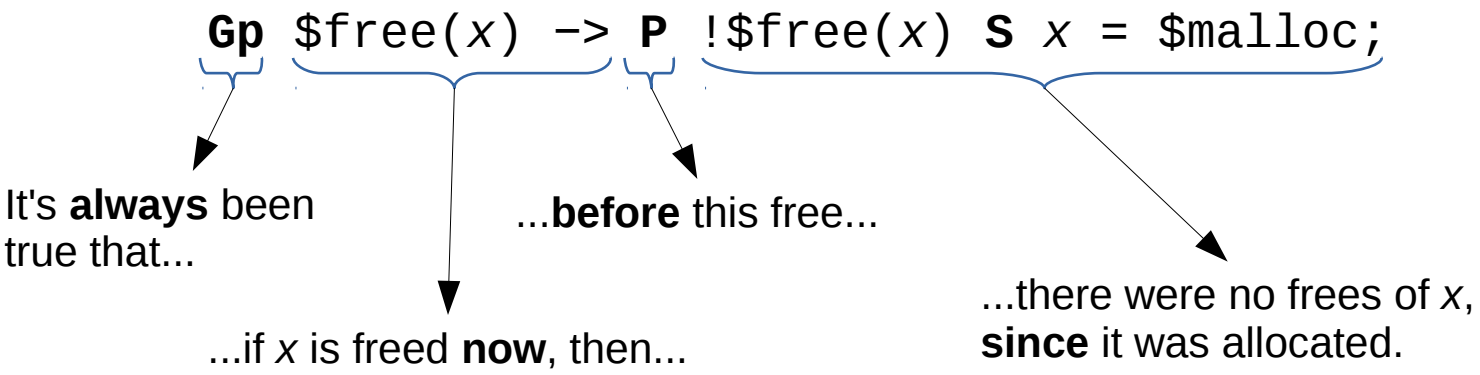


Thanks to my student Andrei Pârvu.

Properties
No Double Frees in LTL

```
void *a = malloc();
...
{a is still allocated}
free(a);
```

- We can now check this:

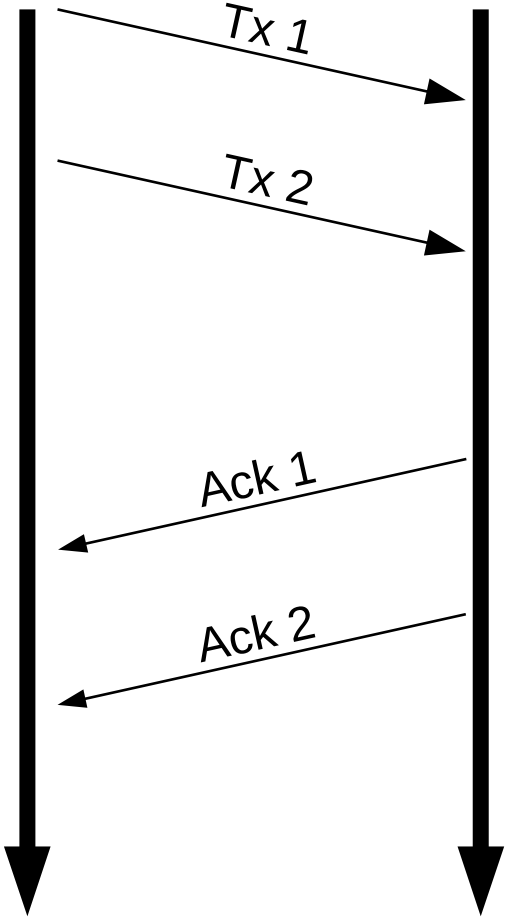


Valgrind, with ***zero*** overhead!

Thanks to my student Andrei Pârvu.

Protocol Debugging

No more than two packets in flight

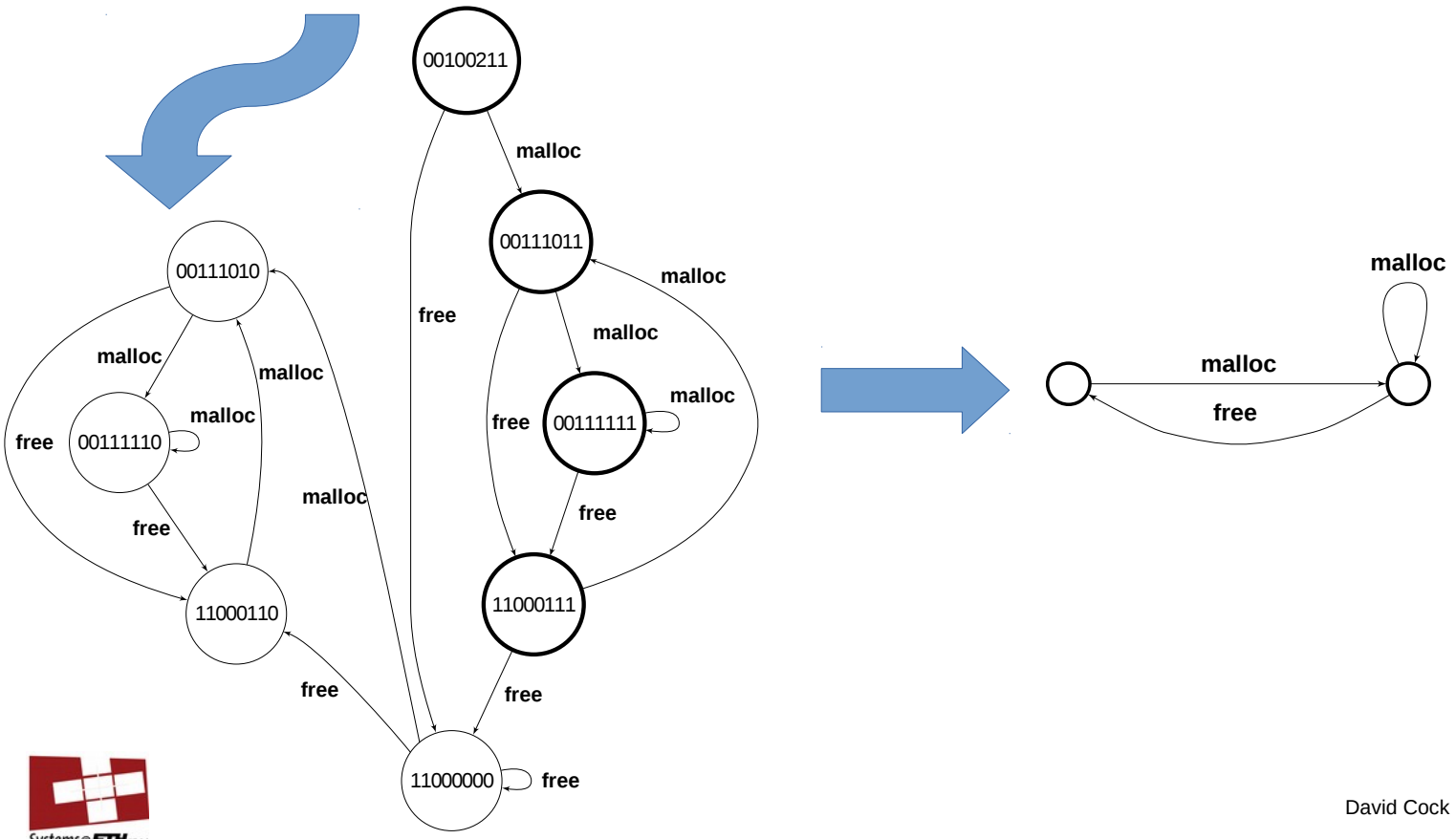



```
G Tx(x) -> !Tx(x+2) U Ack(x);
```

Processing: Checking LTL with Automata

This is a well-studied problem, and standard algorithms exist:

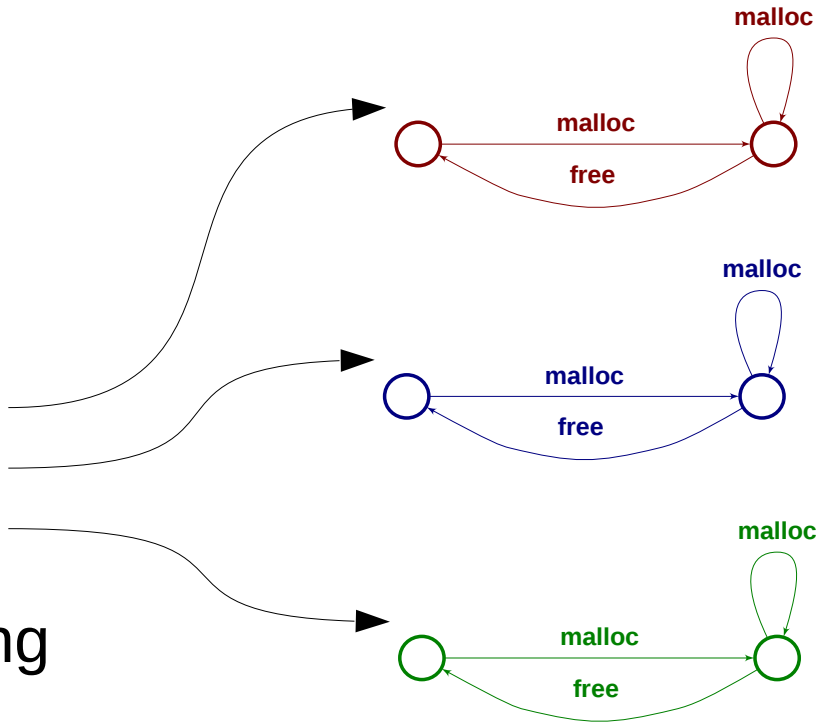
Gp $free(x) -> **P** !$free(x) **S** $x = $malloc;



Processing
Bound Variables and Multiple Automata

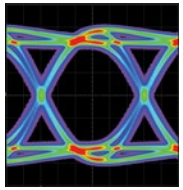
- What about multiple x-s?
- Every x needs an automaton instance.

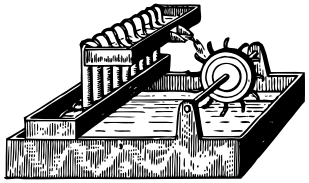
```
Gp $free(1) -> P !$free(1) S 1 = $malloc;
Gp $free(2) -> P !$free(2) S 2 = $malloc;
Gp $free(3) -> P !$free(3) S 3 = $malloc;
```



- Quantifier instantiation using *resolution*.

Our Streaming Verification Engine





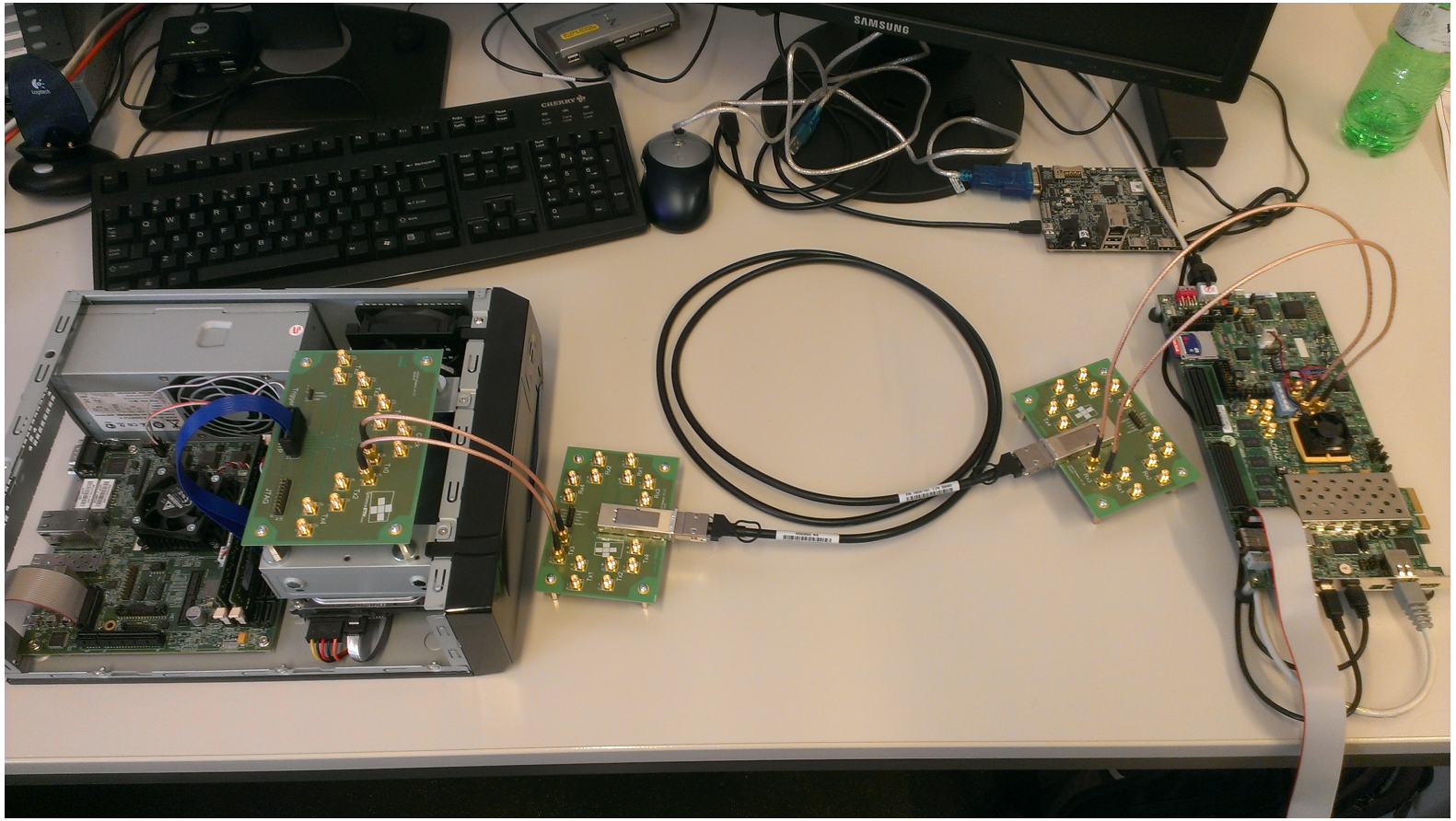
Sources	Capture	Processing	Properties
HSSTP	ETM Sequencer	LTL Automata	TESLA
Zynq TPIU	FPGA Capture	FPGA Offload	`malloc()` pairing
			Live code coverage

Observations

Requirements

Sources

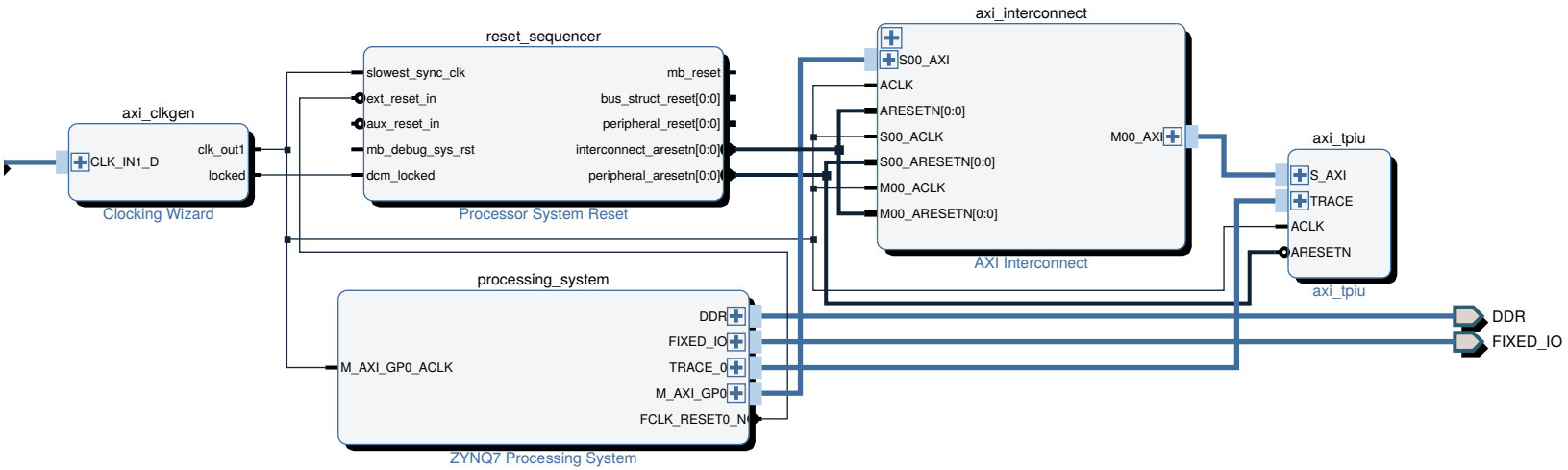
HSSTP Testbench: Zynq7000



Capture

Local Trace Capture on the Zynq7000

- 32b trace port to the FPGA fabric, 250MHz, 8Gb/s.
- Custom TPIU → AXI core, with Linux drivers:
  - Integrates with ARM OpenCSAL, interchangeable with ETB.
  - Full-speed capture and FIFO buffering (512kB).
  - Easy to use: `trace_launch <bin>; cat /dev/axi_tpiu`
  - Coming soon: PCIe & HSSTP output.



Directions

- Enzian will support tracing.
  - Using monitoring to validate the coherence protocol.
- Increasing the scope of FPGA offload (some parts are still software).
- Analysing live systems.
  - Cache operations correctness in Linux, seL4 & Barrelfish.
  - Input sanitisation.
- PSL as an input language.
- SoC integration.

I'd love to hear suggestions!

Questions?

Checking LTL with Automata

This is a well-studied problem, and standard algorithms exist:

```
Gp $free(x) -> P !$free(x) S x = $malloc;
```

Gp P, *at t-1* *„P was true until t-1“*	P, *at t* *„P is still true at t“*	**Gp** P, *at t* *„P has always been true“*
0	0	0
0	1	0
1	0	0
1	1	1