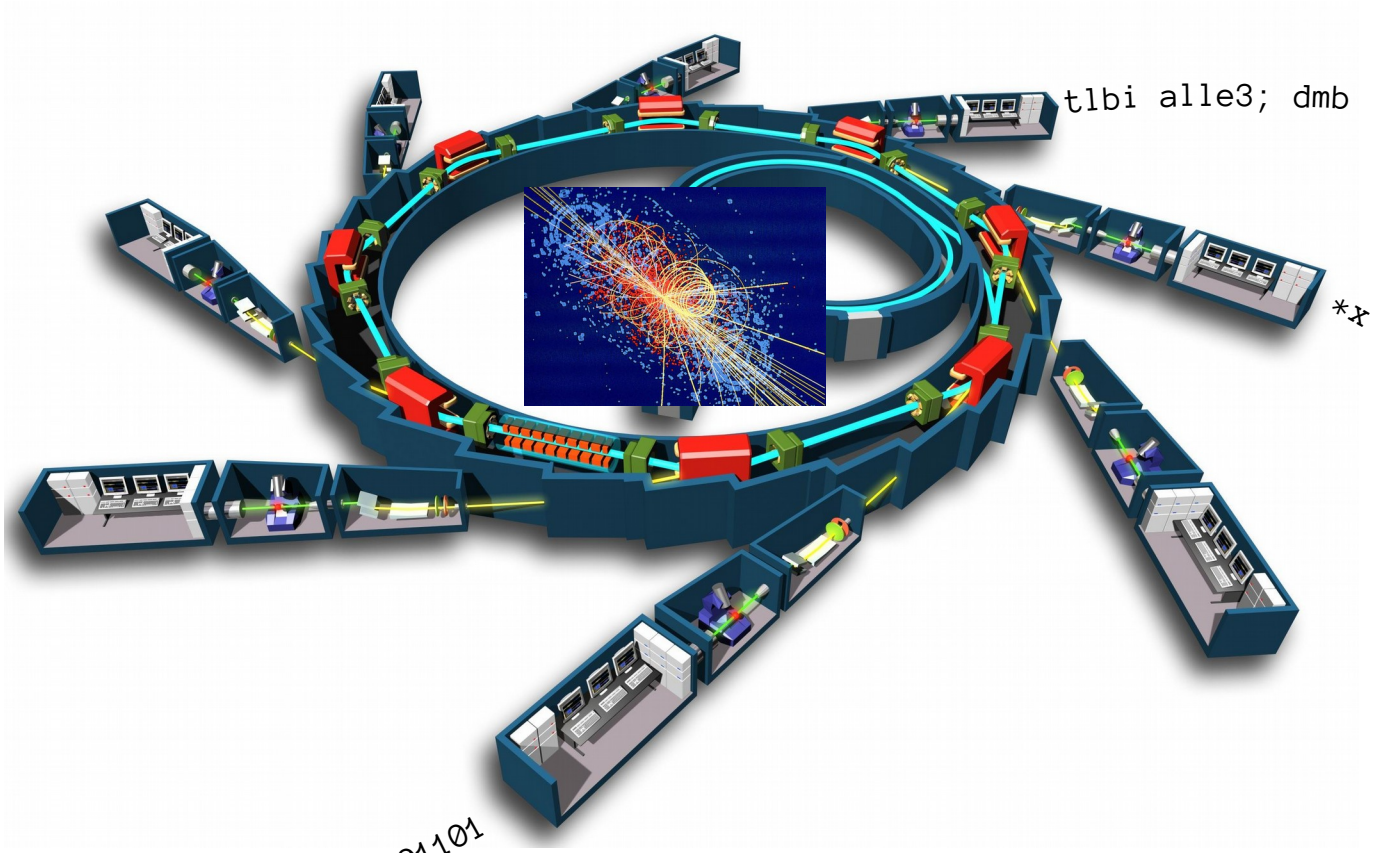
Runtime Verification

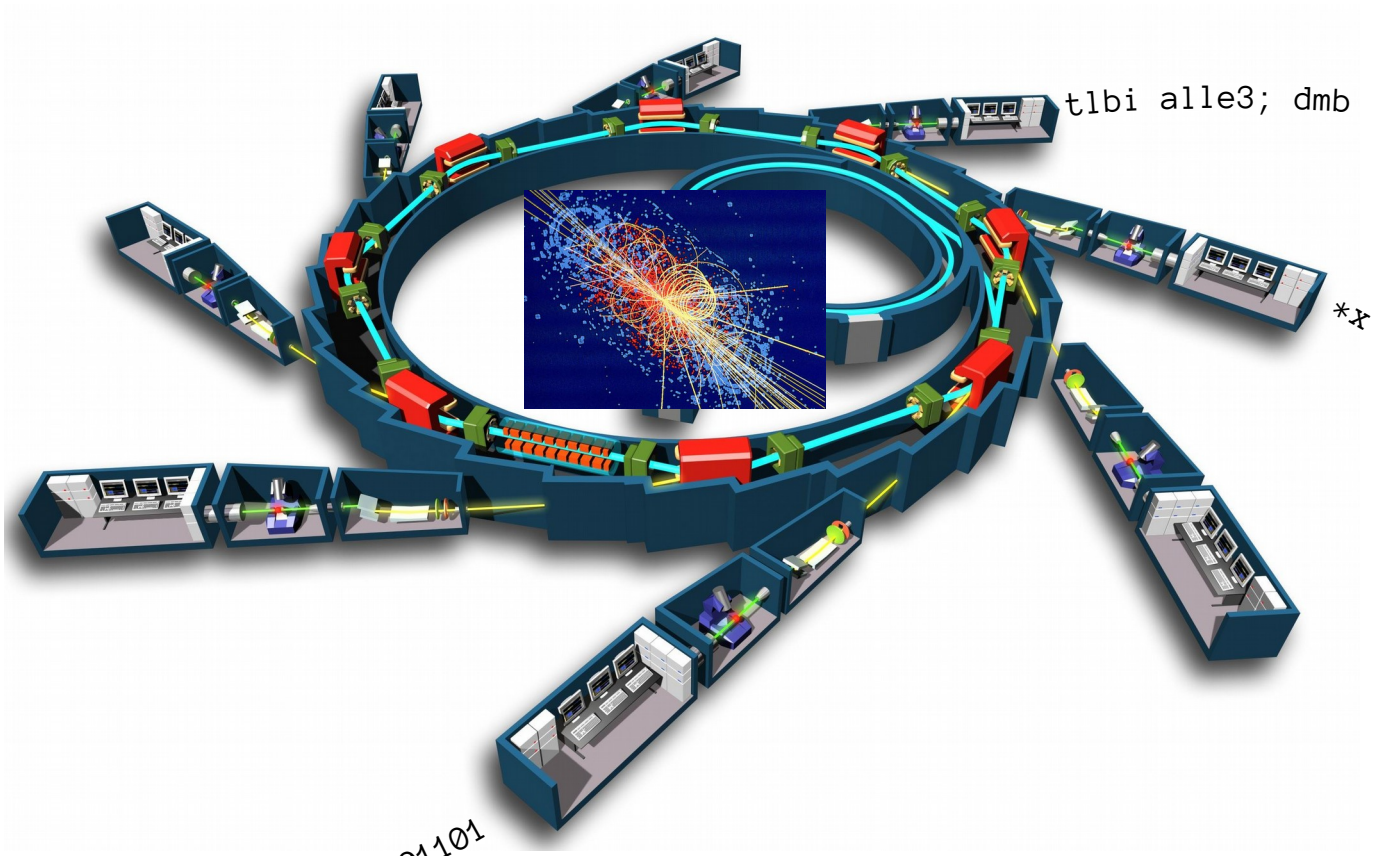
David Cock – david.cock@inf.ethz.ch

Systems@ETH Zürich

ad



We're Building the *Large Program Collider*



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

Images: CERN; Chaix & Morel et associés

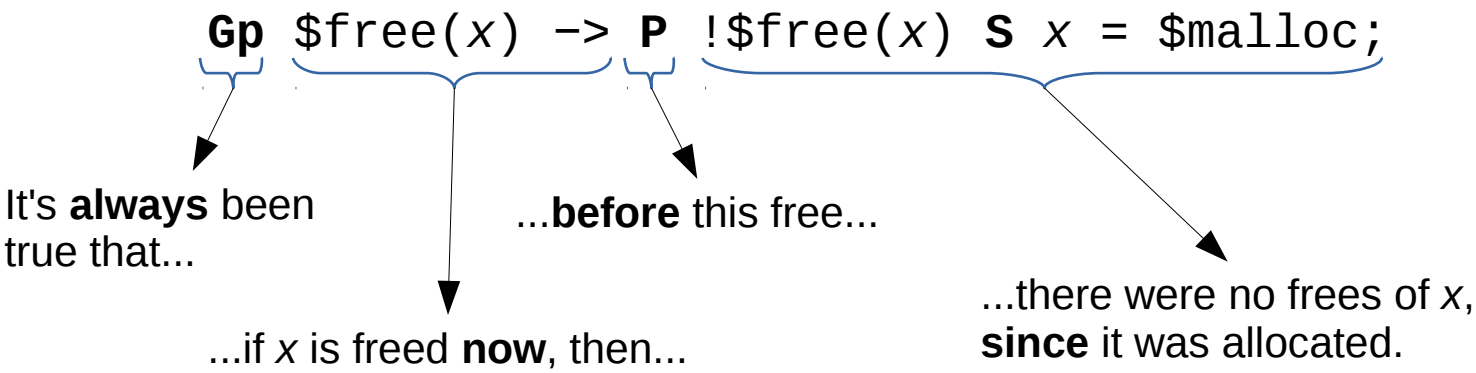
Why is This Useful?

- 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).

Properties
No Double Frees in LTL

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

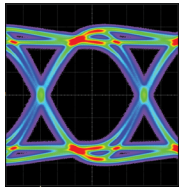
- We can now check this:

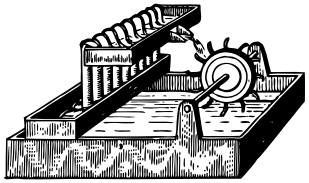


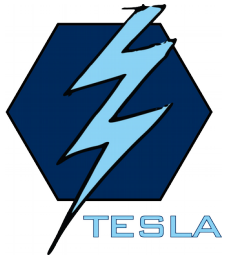
Valgrind, with ***zero*** overhead!

Thanks to my student Andrei Pârvu.

Our Streaming Verification Engine







Sources

HSSTP

Zynq TPIU

Capture

ETM Sequencer

FPGA Capture

Processing

LTL Automata

FPGA Offload

Properties

TESLA

`malloc()` pairing

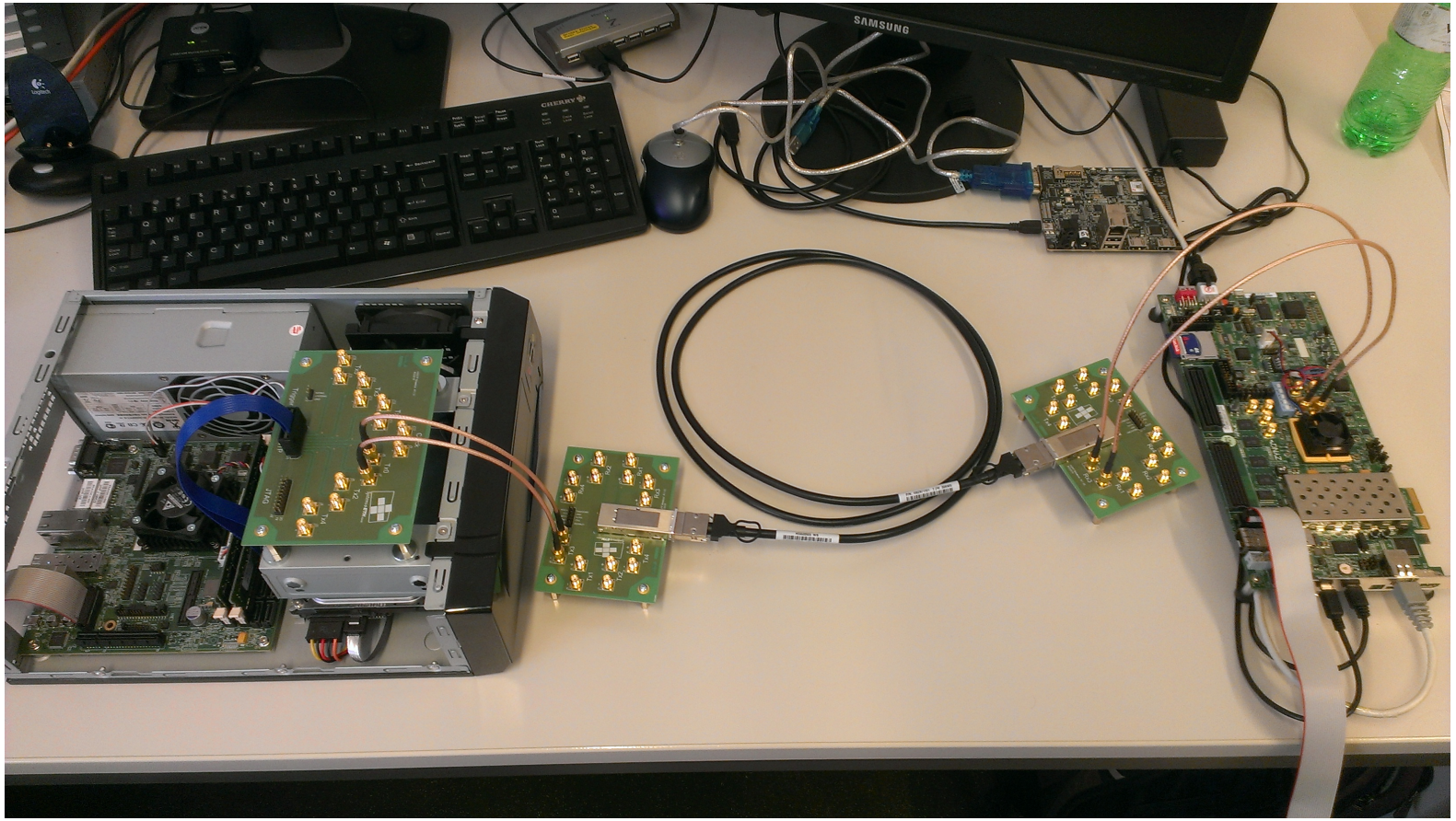
Live code coverage

Observations

Requirements

Sources

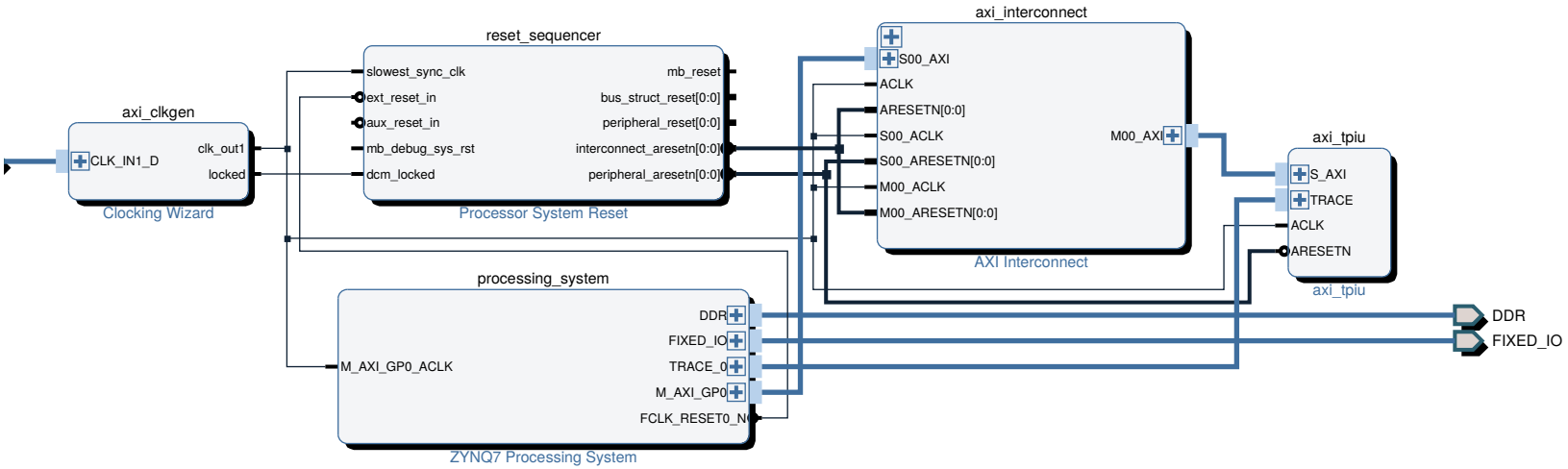
HSSTP Testbench: XGene 1, Zynq7000, Custom HW



Capture

Trace Capture on the Zynq7000

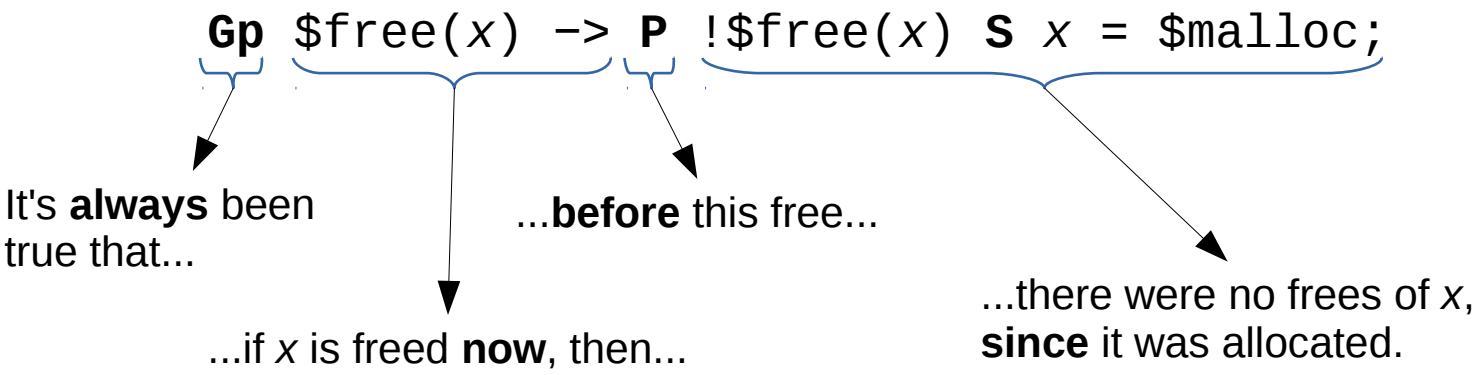
- 32b trace port to the FPGA fabric, 250MHz, 8Gb/s.
- Custom TPIU → AXI core, with Linux drivers:
  - Integrates with ARM OpenCSAL, interchangeable with ETB.
  - Full-speed capture and FIFO buffering (512kB).
  - Easy to use: `trace_launch <bin>; cat /dev/axi_tpiu`
  - Coming soon: PCIe & HSSTP output.



Properties
No Double Frees in LTL

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

- We can now check this:

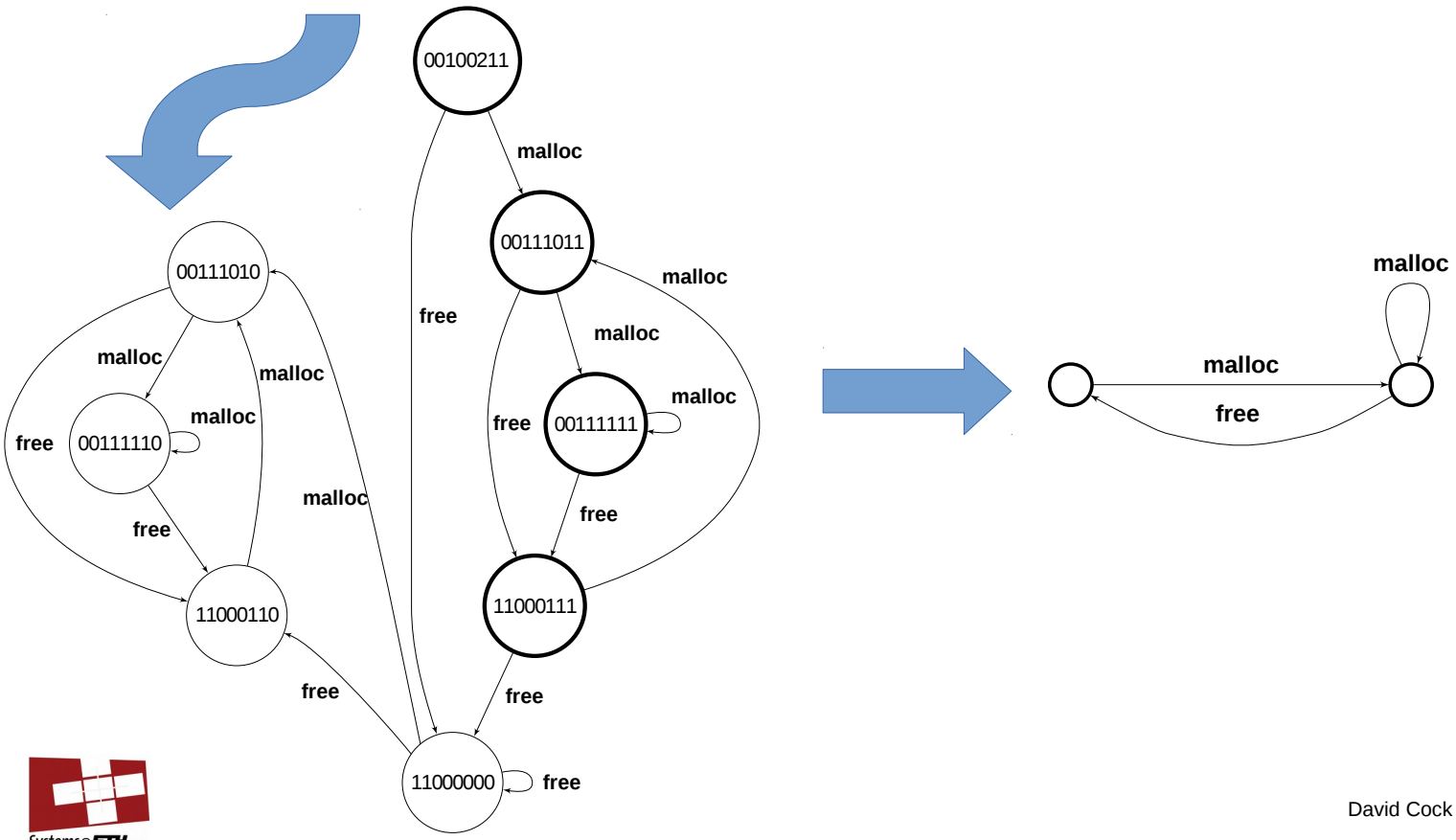


Thanks to my student Andrei Pârvu.

Processing: Checking LTL with Automata

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

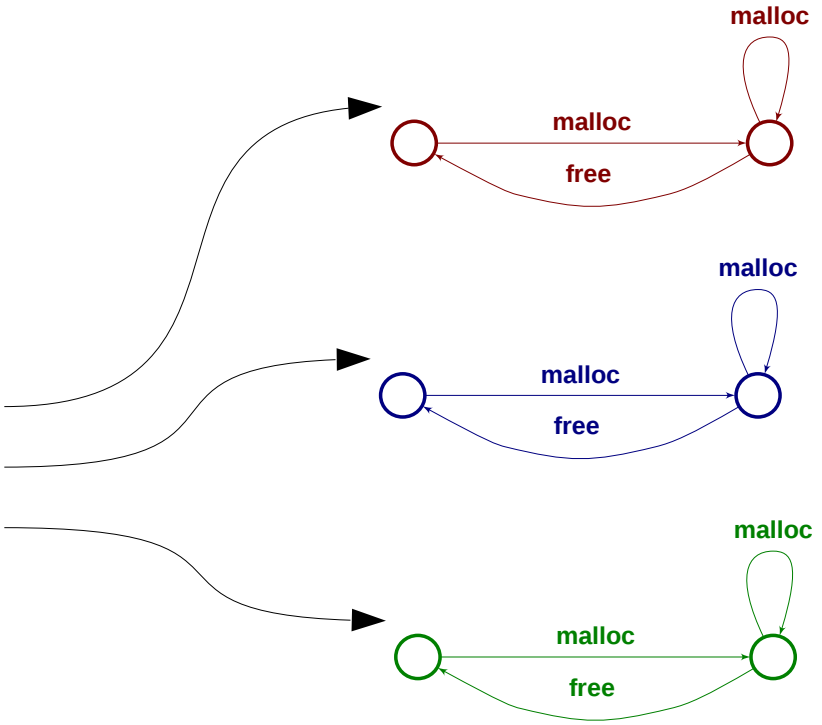
Gp `$free(`*x*`)` `->` **P** `!$free(`*x*`)` **S** *x* `= $malloc;`



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;
```



- We keep a tree of partially-instantiated automata.
- This is *resolution*.

Processing
Streaming Pipeline

Chained processing stages:

- Capture
  - Using the TPIU → AXI core.
  - Using HSSTP: On the Zynq soon, other hardware is buggy.
- Decode
  - We have a custom PTM decoder.
  - Also use Linaro OpenCSD – works, but still buggy.
- Checking
  - Run the automaton.
  - Live coverage analysis – ***demo***.

Directions

- BRISC will support tracing.
- We're moving PTM parsing into the FPGA.
- We need to integrate and test the pipeline.
  - The pieces are all there, they need to work together.
- Analysing live systems.
  - We have an ongoing survey of cache operations in Linux, seL4 & Barrelfish. We should be able to test these.
- More interesting CPU platforms (see. BRISC).

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