Gator: What Now?

a talk from your pal, Andrew!

05 27 2024

What's the most stressful thing you've done?

What's the most stressful thing Andrew's done?

Several ideas:

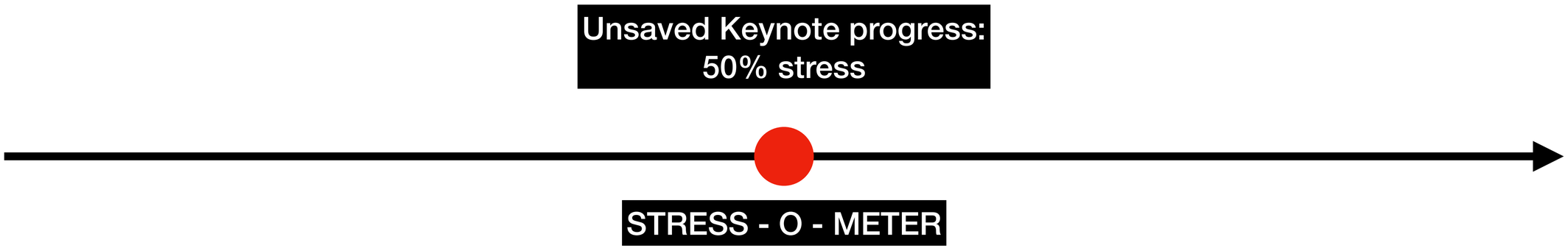
STRESS - O - METER

What's the most stressful thing Andrew's done?

Several ideas:

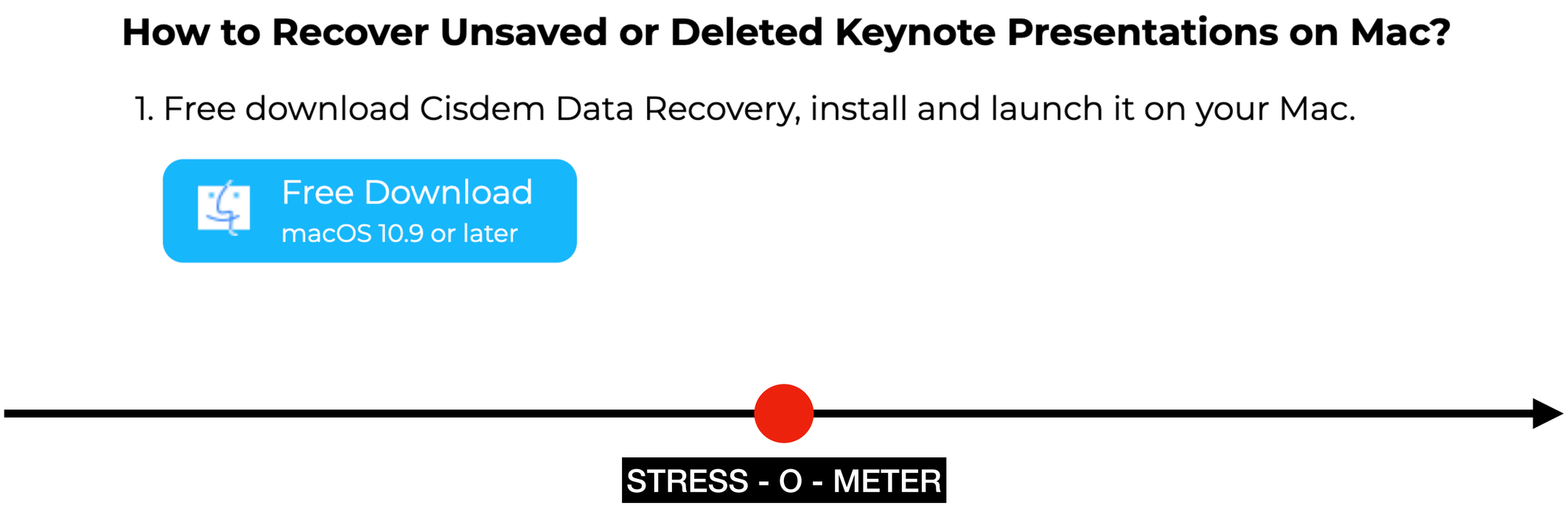
What's the most stressful thing Andrew's done?

Several ideas:



What's the most stressful thing Andrew's done?

Several ideas:



What's the most stressful thing Andrew's done?

Several ideas:

What's the most stressful thing Andrew's done?

Several ideas:

Working as a
Formal Verification Hardware Intern at:

STRESS - O - METER

What's the most stressful thing Andrew's done?

Several ideas:

Working as a
Formal Verification Hardware Intern at:

(This stress was caused
by my own stupidity, and not at all
by the talented awesome people
I worked with!)

STRESS - O - METER

At Intel, I formally verified the correctness for an accelerator.

At Intel, I formally verified the correctness for (part of) an accelerator.

Hardware Verification

How does it work?

Hardware Verification

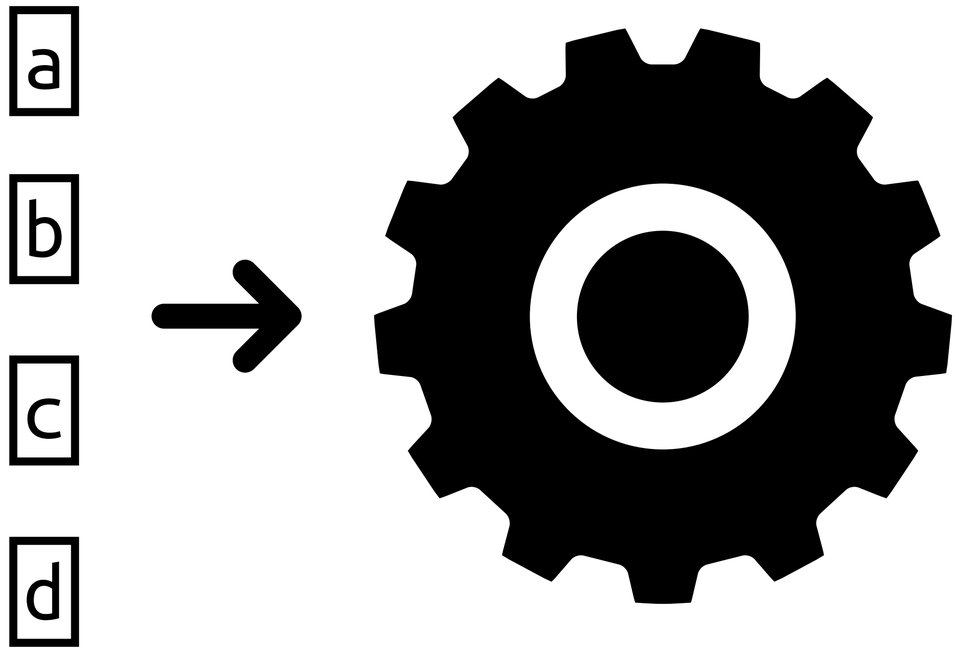
How does it work?

(ab + cd)

Specification

Hardware Verification

How does it work?



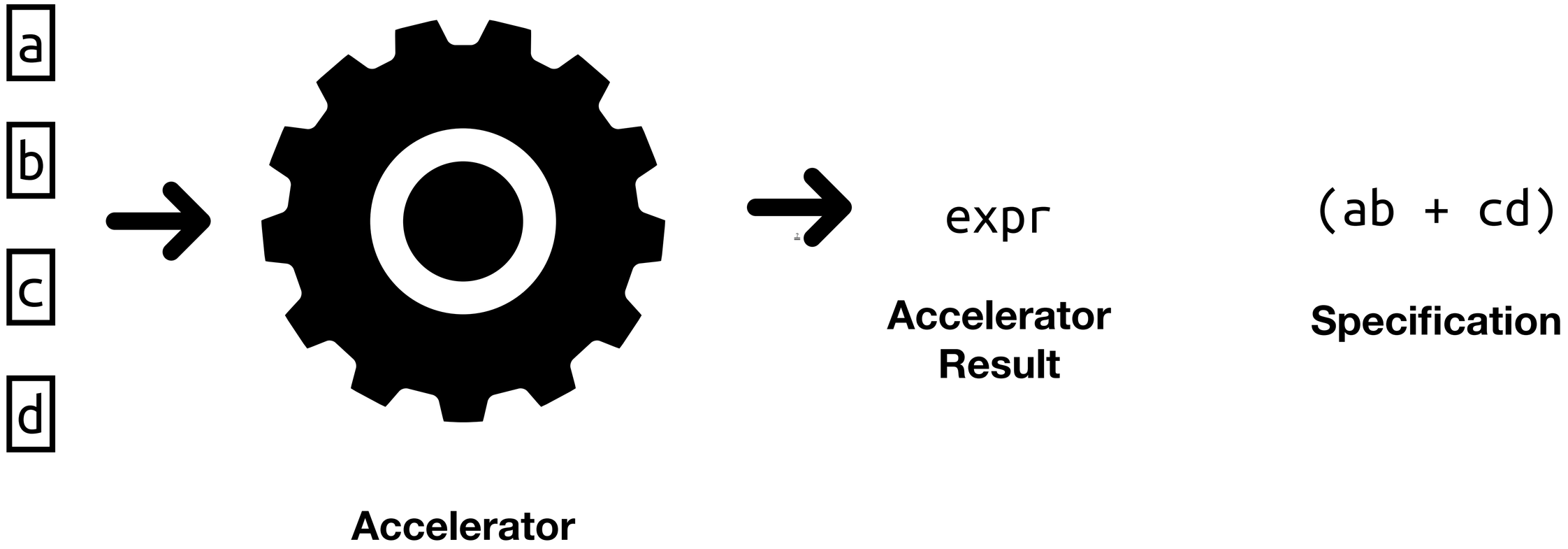
Accelerator

```
(ab + cd)
```

Specification

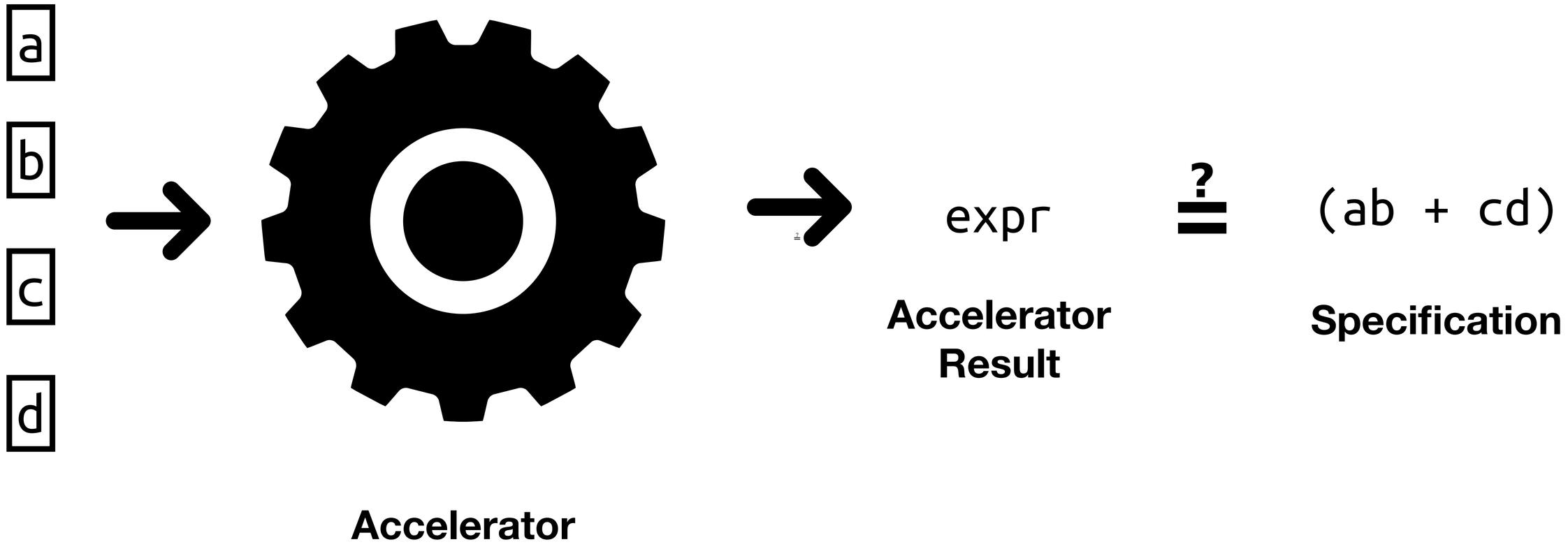
Hardware Verification

How does it work?



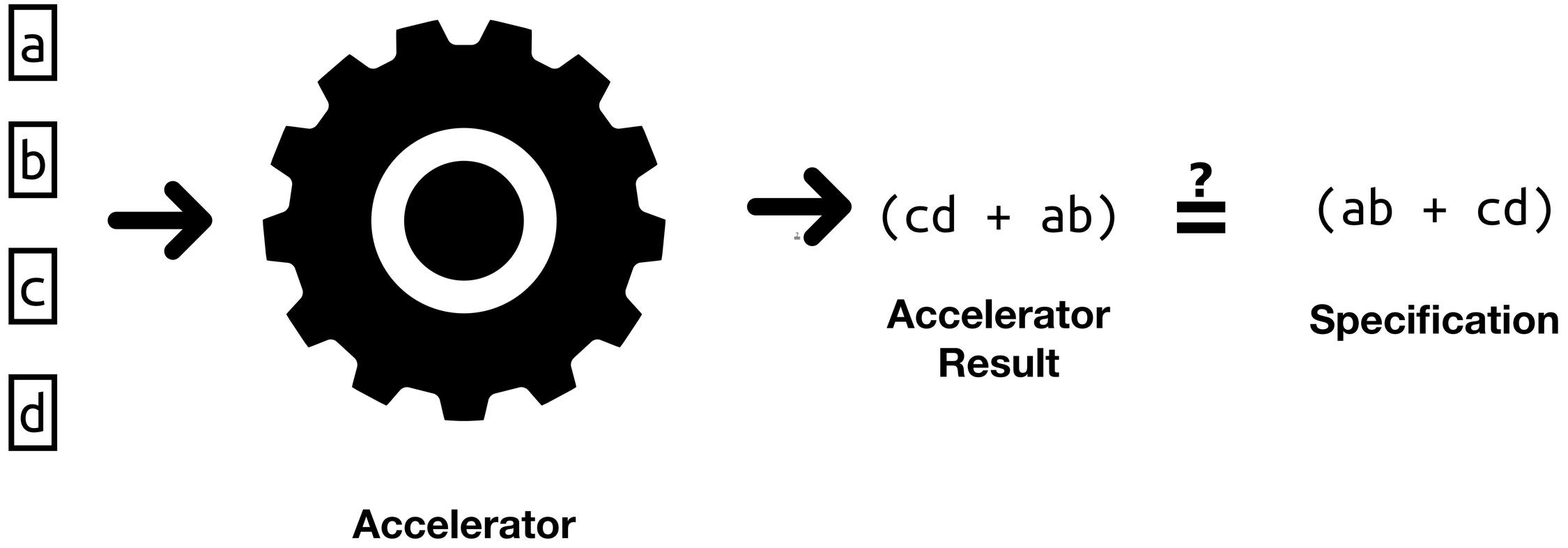
Hardware Verification

How does it work?



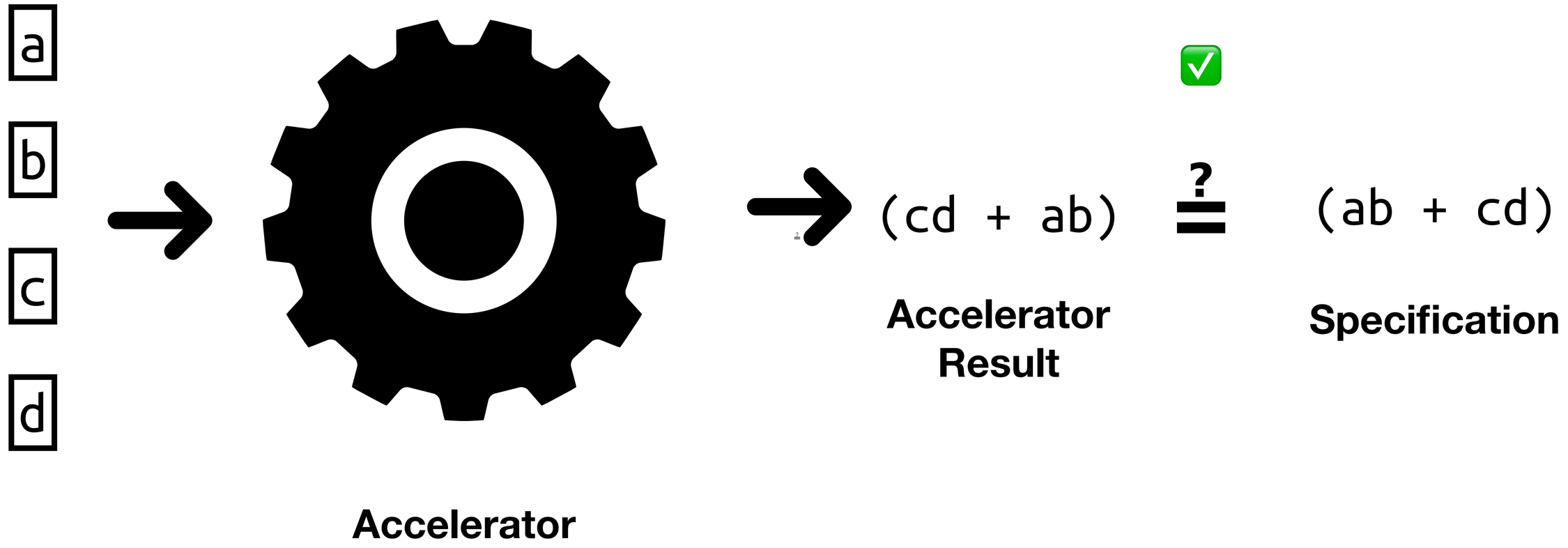
Hardware Verification

How does it work?



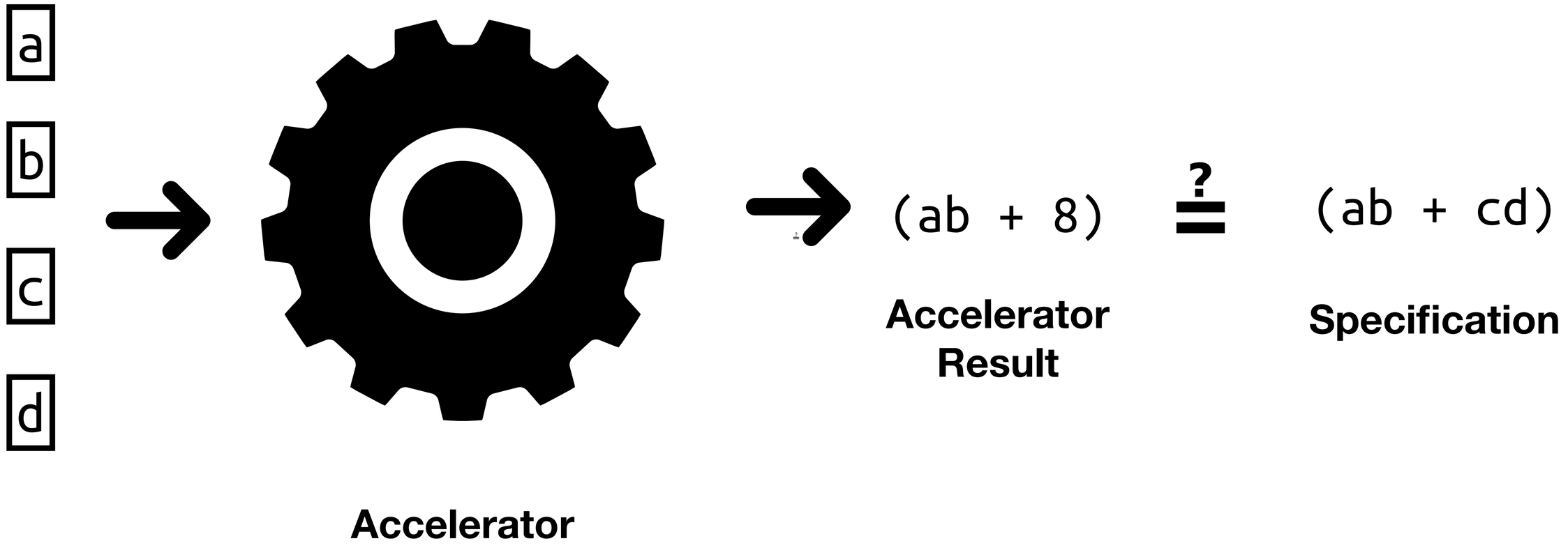
Hardware Verification

How does it work?



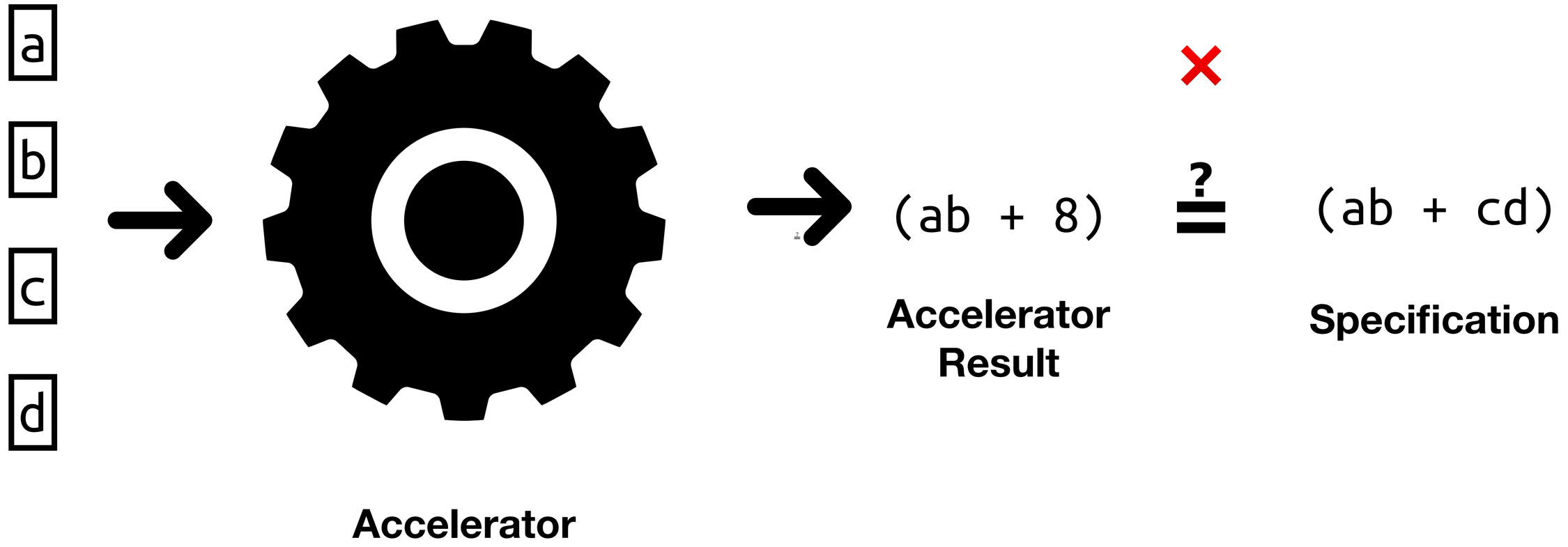
Hardware Verification

How does it work?



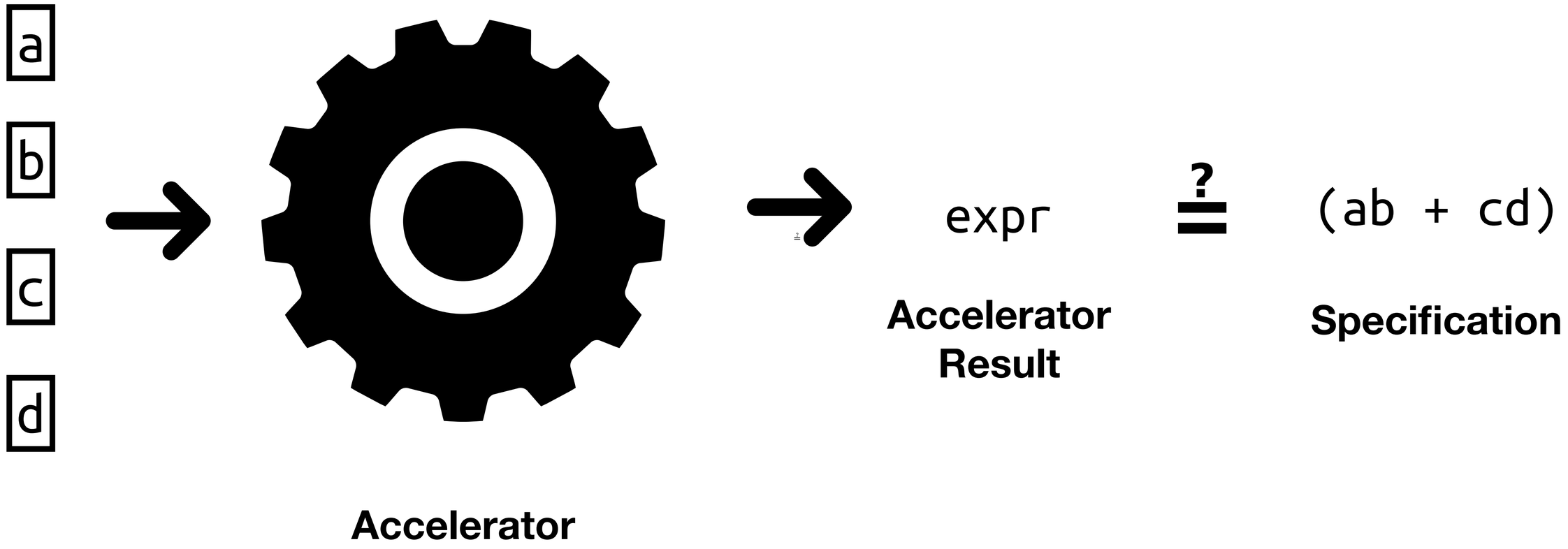
Hardware Verification

How does it work?



Hardware Verification

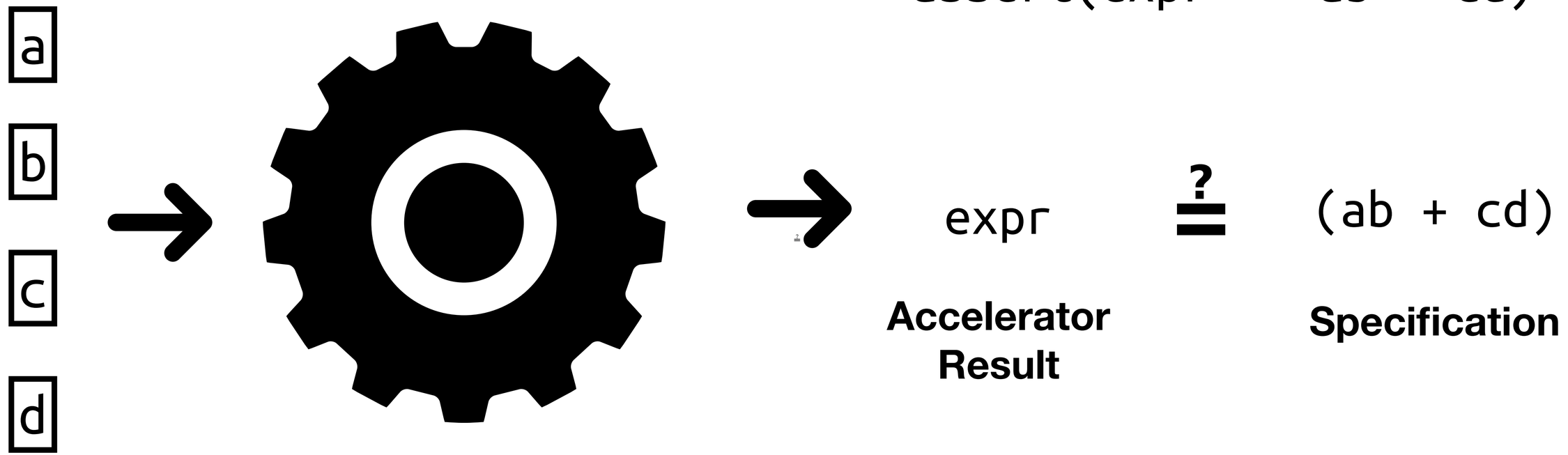
How does it work?



Hardware Verification

How does it work?

```
assert(expr == ab + cd)
```

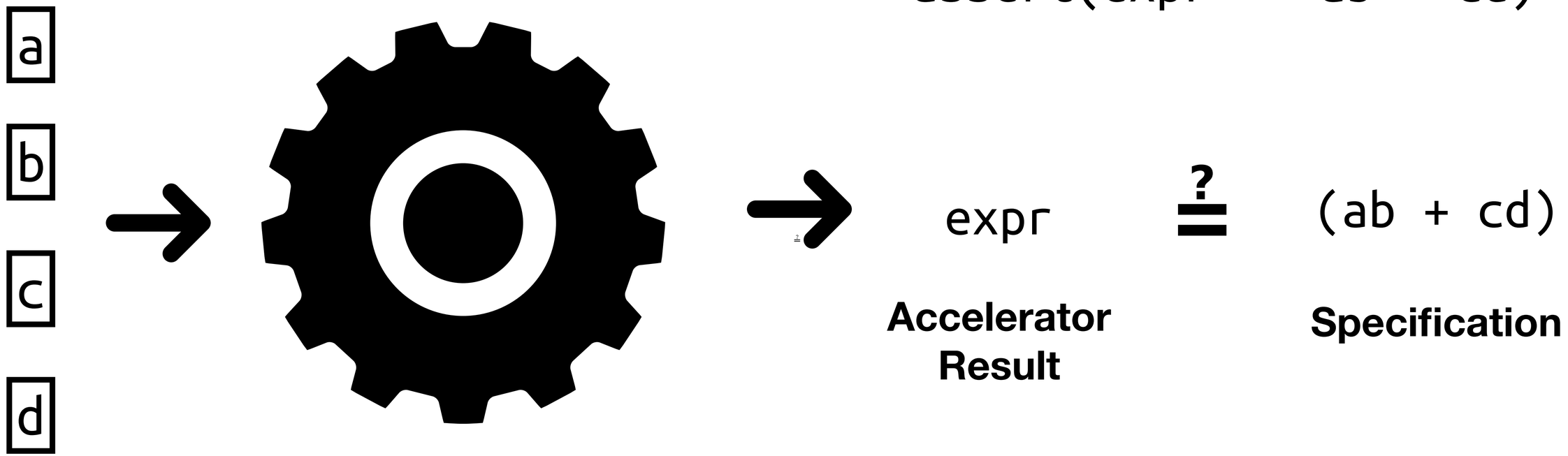


Hardware Verification

How does it work?

What's hard about this?

```
assert(expr == ab + cd)
```



Industrial-grade verification systems require *pure, functional specifications.*

On Writing Pure Function Specs

What's hard about it?

```
def model(inputs: List[Packet], stall_count: int):
    if stall_count > 0:
        return [DEFAULT_PACKET] * 4
    outputs = [None] * 4
    for input_packet in inputs:
        outputs[input_packet.out_port] = input_packet
    return outputs
```

On Writing Pure Function Specs

What's hard about it?

```
def pure_model(inputs: List[Packet], stall_count: int) -> List[Packet]:
    return map(
        lambda out_port: (
            DEFAULT_PACKET
            if stall_count > 0
            else foldl(
                lambda packet, accumulator: (
                    packet if packet.out_port == out_port else accumulator
                ),
                inputs,
                None,
            )
        ),
        range(4),
    )
```

On Writing Pure Function Specs

What's hard about it?

```python
def model(inputs: List[Packet], stall_count: int):
    if stall_count > 0:
        return [DEFAULT_PACKET] * 4
    outputs = [None] * 4
    for input_packet in inputs:
        outputs[input_packet.out_port] = input_packet
    return outputs
```

```python
def pure_model(inputs: List[Packet], stall_count: int) -> List[Packet]:
    return map(
        lambda out_port: (
            [DEFAULT_PACKET] * 4
            if stall_count > 0
            else foldl(
                lambda packet, accumulator: (
                    packet if packet.out_port == out_port else accumulator
                ),
                inputs,
                None,
            )
        ),
        range(4),
    )
```

On Writing Pure Function Specs

What's hard about it?

```python
def model(inputs: List[Packet], stall_count: int):
    if stall_count > 0:
        return [DEFAULT_PACKET] * 4
    outputs = [None] * 4
    for input_packet in inputs:
        outputs[input_packet.out_port] = input_packet
    return outputs
```

calculating the answer

```python
def pure_model(inputs: List[Packet], stall_count: int) -> List[Packet]:
    return map(
        lambda out_port: (
            [DEFAULT_PACKET] * 4
            if stall_count > 0
            else foldl(
                lambda packet, accumulator: (
                    packet if packet.out_port == out_port else accumulator
                ),
                inputs,
                None,
            )
        ),
        range(4),
    )
```

On Writing Pure Function Specs

What's hard about it?

```python
def model(inputs: List[Packet], stall_count: int):
    if stall_count > 0:
        return [DEFAULT_PACKET] * 4
    outputs = [None] * 4
    for input_packet in inputs:
        outputs[input_packet.out_port] = input_packet
    return outputs
```

calculating the answer

```python
def pure_model(inputs: List[Packet], stall_count: int) -> List[Packet]:
    return map(
        lambda out_port: (
            [DEFAULT_PACKET] * 4
            if stall_count > 0
            else foldl(
                lambda packet, accumulator: (
                    packet if packet.out_port == out_port else accumulator
                ),
                inputs,
                None,
            )
        ),
        range(4),
    )
```

calculating all possible answers

On Writing Pure Function Specs

It's definitely not impossible...

On Writing Pure Function Specs

It's definitely not impossible...

On Writing Pure Function Specs

It's definitely not impossible...

On Writing Pure Function Specs

It's definitely not impossible...

On Writing Pure Function Specs

It's definitely not impossible...

On Writing Pure Function Specs

It's definitely not impossible...

On Writing Pure Function Specs

It's definitely not impossible...

To Recap:

- **Post-synthesis verification**: making a model and verifying equivalence *after* the hardware's been implemented.
- Post-synthesis verification is not fun:
  - It involves writing programs in an unnatural way
  - Even small changes in the hardware mean dramatic changes to a model

Can we do better?

Can we do better?

Correct-by-Construction Design of Custom Accelerator Microarchitectures

Jin Yang, Zhenkun Yang, Jeremy Casas, and Sandip Ray, *Senior Member*, IEEE

Can we do better?

FPGA Technology Mapping Using Sketch-Guided Program Synthesis

Gus Henry Smith
University of Washington
Seattle, USA
gussmith@cs.washington.edu

Ben Kushigian
University of Washington
Seattle, USA
benku@cs.washington.edu

Vishal Canumalla
University of Washington
Seattle, USA
vishalc@cs.washington.edu

Andrew Cheung
University of Washington
Seattle, USA
acheung8@cs.washington.edu

Steven Lyubomirsky
OctoAI
Seattle, USA
slyubomirsky@octo.ai

Sorawee Porncharoenwase
University of Washington
Seattle, USA
sorawee@cs.washington.edu

René Just
University of Washington
Seattle, USA
rjust@cs.washington.edu

Gilbert Louis Bernstein
University of Washington
Seattle, USA
gilbo@cs.washington.edu

Zachary Tatlock
University of Washington
Seattle, USA
ztatlock@cs.washington.edu

Can we do better?

A Survey of Hardware Compilers

Lakeroad

Lutsig

Vivado

Extensible

Correct

Lakeroad uses *program synthesis* to compile hardware designs correct for the first k cycles.

 uses *program synthesis* to compile hardware designs correct for the first k cycles.

 uses *program synthesis* to compile hardware designs correct for the first ∞ cycles.

🐊 uses *p* :ompile

hardware (e first ∞

Program Synthesis

```
sketch = 2 + ??
spec = 5

synthesize(assert(spec == sketch))
```

Program Synthesis

A solver will fill in holes...

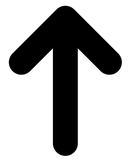
```
sketch = 2 + ??
spec = 5

synthesize(assert(spec == sketch))
```

Program Synthesis

```
sketch = 2 + 3
spec = 5

synthesize(assert(spec == sketch))
```



...that satisfy the query!

Lakeroad's Synthesis Query

```
synthesize(
    assert_that(
        simulate(model, 0) == simulate(sketch, 0)
        and simulate(model, 1) == simulate(sketch, 1)
        and simulate(model, 2) == simulate(sketch, 2)
    )
)
```

Gator's Synthesis Query

```
synthesize(
    assume(t >= 0),
    assert_that(
        simulate(model, t) == simulate(sketch, t)
    )
)
```

Gator's Synthesis Query

```
synthesize(
    assume(t > 0),
    # Base case
    assert_that(
        simulate(model, 0) == simulate(sketch, 0)
    ),
    # Inductive hypothesis
    assume(
            simulate(model, t) == simulate(sketch, t)
    ),
    # Inductive step
    assert_that(
        simulate(model, t + 1) == simulate(sketch, t + 1)
    )
)
```

Three Questions:

1: How do you simulate hardware over symbolic time?

2. How do you encode inductive hypotheses?

3. How do you make step 2 less painful?

Gator

What is it?

```
synthesize(
    assume(t > 0),
    # Base case
    assert_that(
        simulate(model, 0) == simulate(sketch, 0)
    ),
    # Inductive hypothesis
    assume(
            simulate(model, t) == simulate(sketch, t)
    ),
    # Inductive step
    assert_that(
        simulate(model, t + 1) == simulate(sketch, t + 1)
    )
)
```

Gator

What is it?

```
synthesize(
    assume(t > 0),
    # Base case
    assert_that(
        simulate(model, 0) == simulate(sketch, 0)
    ),
    # Inductive hypothesis
    assume(
            simulate(model, t) == simulate(sketch, t)
    ),
    # Inductive step
    assert_that(
        simulate(model, t + 1) == simulate(sketch, t + 1)
    )
)
```

The solver will actually try and run this!

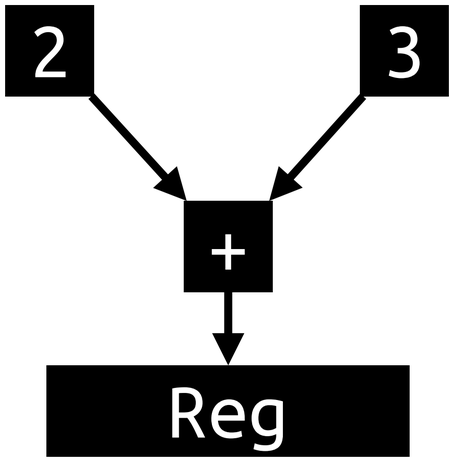
Gator

What is it?

```
def simulate(circuit, t):
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator

What is it?

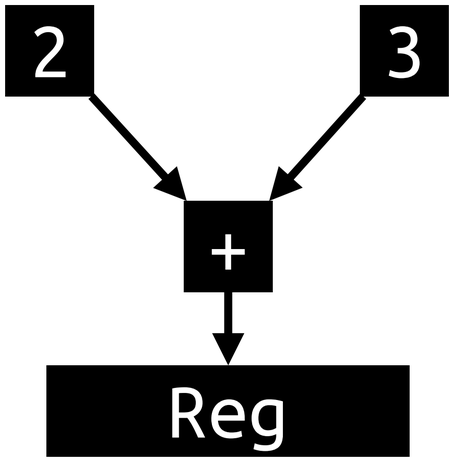



```
simulate(Register(0, Add(Num(2), Num(3))), t)
```

```
def simulate(circuit, t):
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator

What is it?

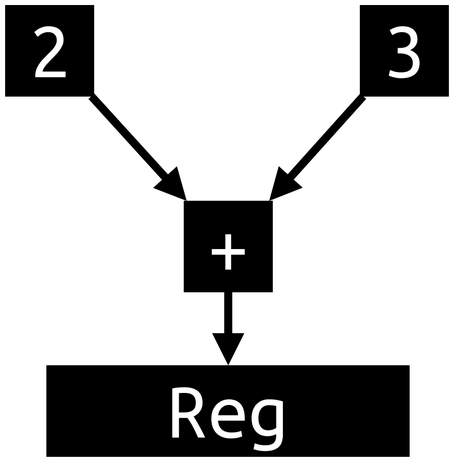



```
simulate(Add(Num(2), Num(3)), t - 1)
```

```
def simulate(circuit, t):
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator

What is it?

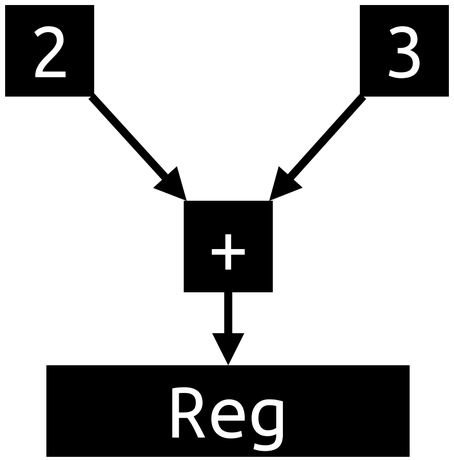



```
simulate(Num(2), t - 1) + simulate(Num(3), t - 1)
```

```
def simulate(circuit, t):
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator

What is it?

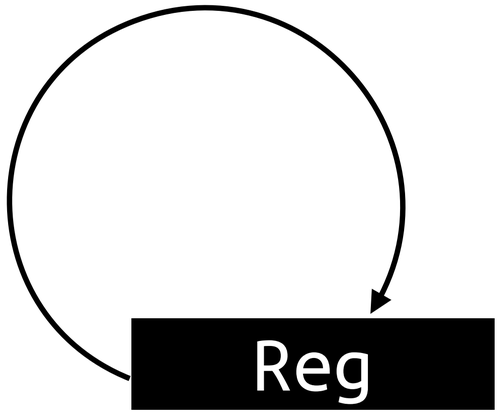


2 + 3

```
def simulate(circuit, t):
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator

What is it?

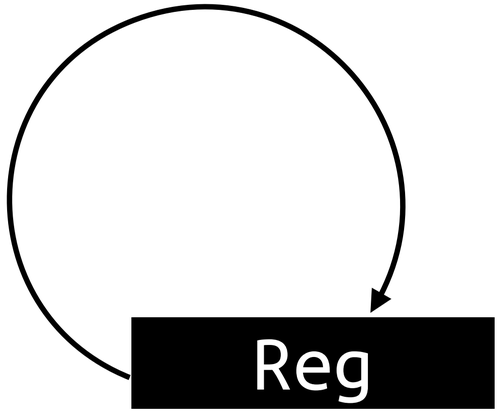



```
simulate(Register(0, itself), t)
```

```
def simulate(circuit, t):
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator

What is it?

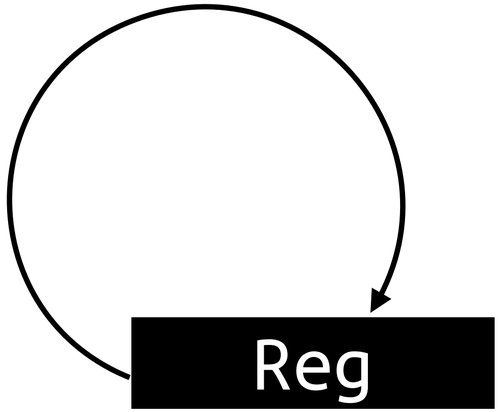



```
simulate(Register(0, itself), t - 1)
```

```
def simulate(circuit, t):
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator

What is it?

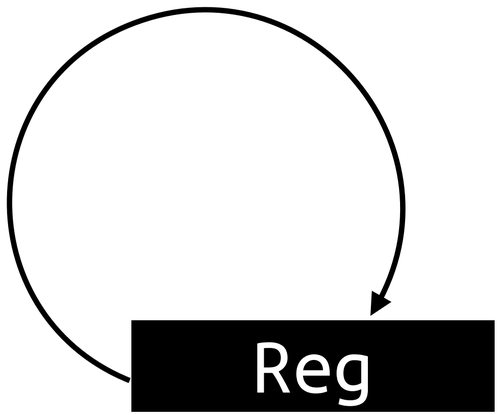



```
simulate(Register(0, itself), t - 2)
```

```
def simulate(circuit, t):
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator

What is it?




```
simulate(Register(0, itself), t - 3)
```

```
def simulate(circuit, t):
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator

What is it?


```
simulate(Register(0, itself), t - 3)
```

```
def simulate(circuit, t):
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

When dealing with cyclical hardware programs over a symbolic time, we need to "stop the simulation" somehow!

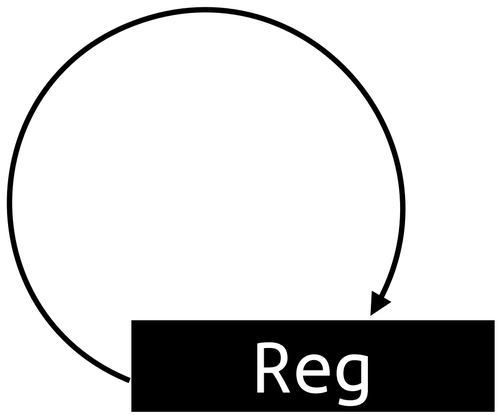
Gator's Solution

Add a cache!

```
def simulate(circuit, t, cache):
    if (circuit, t) in cache:
        return cache[(circuit, t)]
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator's Solution

Add a cache!

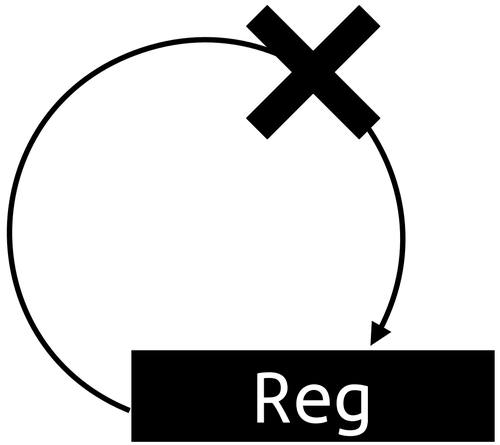



```
simulate(Register(0, itself), t)
```

```
def simulate(circuit, t, cache):
    if (circuit, t) in cache:
        return cache[(circuit, t)]
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator's Solution

Add a cache!

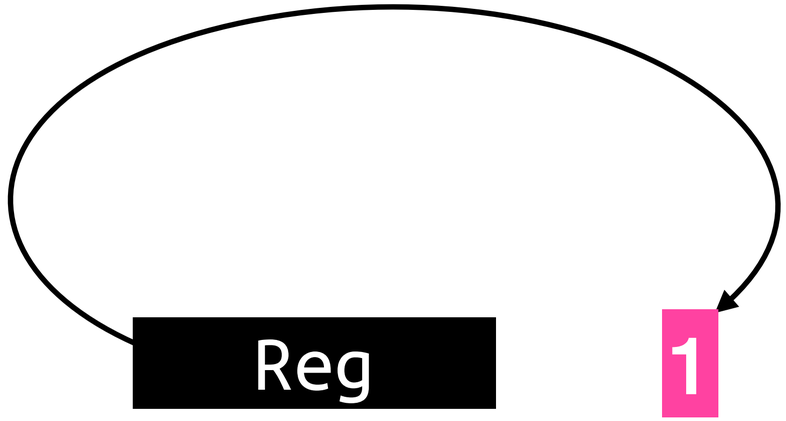



```
simulate(Register(0, itself), t)
```

```
def simulate(circuit, t, cache):
    if (circuit, t) in cache:
        return cache[(circuit, t)]
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator's Solution

Add a cache!

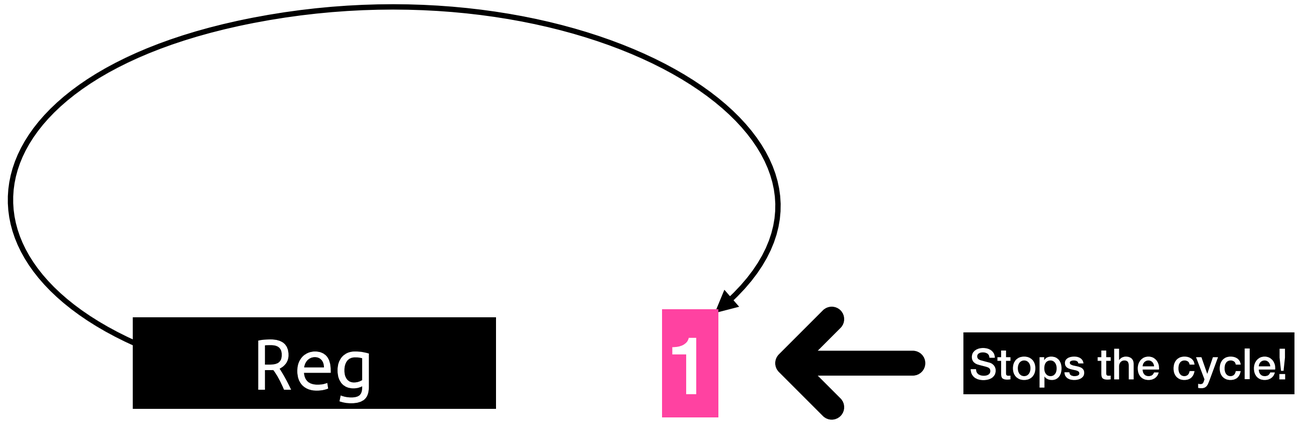



```
simulate(Register(0, itself), t, {(Register(0, itself), t - 1): 1})
```

```
def simulate(circuit, t, cache):
    if (circuit, t) in cache:
        return cache[(circuit, t)]
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

Gator's Solution

Add a cache!




```
simulate(Register(0, itself), t, {(Register(0, itself), t - 1): 1})
```

```
def simulate(circuit, t, cache):
    if (circuit, t) in cache:
        return cache[(circuit, t)]
    match circuit:
        case Num(n): return n
        case Add(a, b): return simulate(a, t) + simulate(b, t)
        case Register(init, data):
            return init if t == 0 else simulate(data, t - 1)
```

By making use of a cache, we can reason about cyclical programs!

Three Problems:

1: How do you write the problem down? ✅

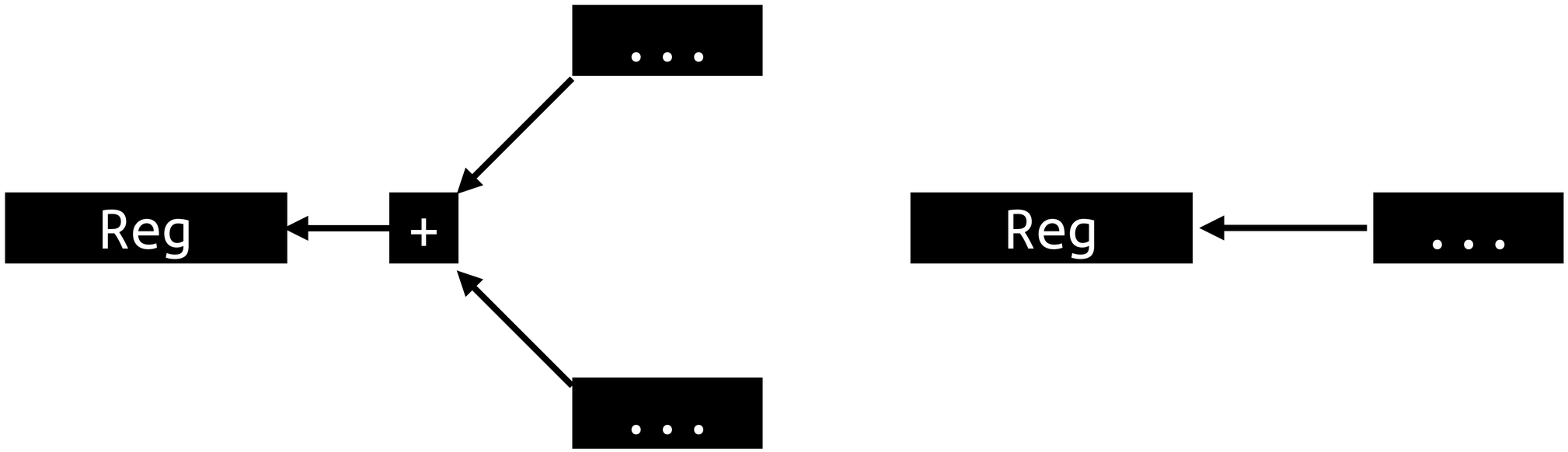
2. How do you encode inductive hypotheses?

3. How do you make step 2 less painful?

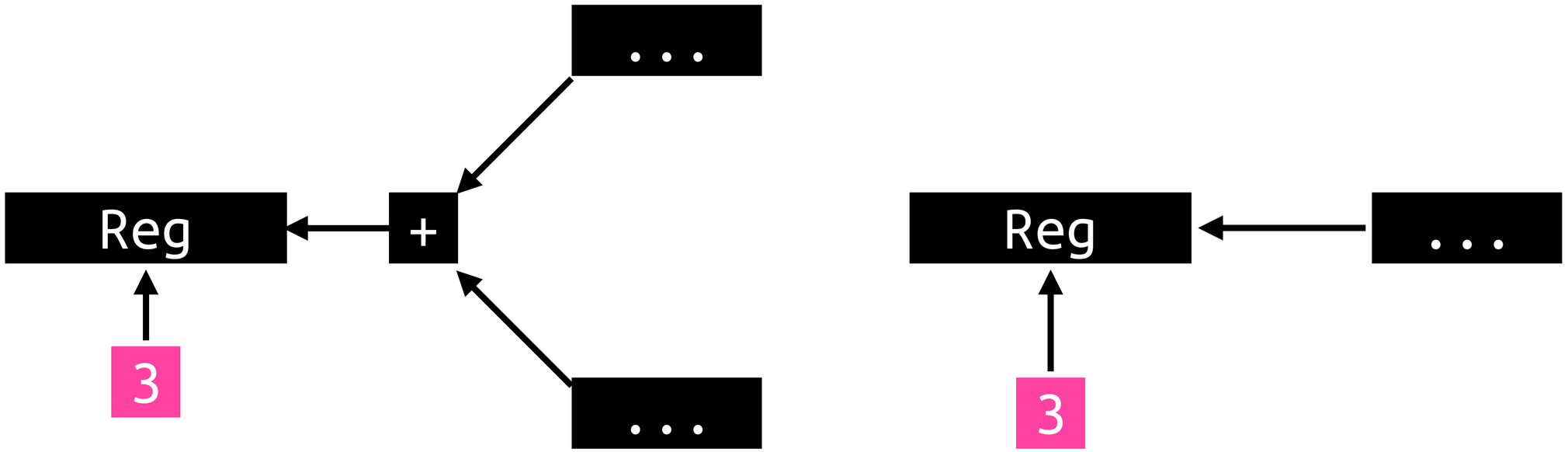
Gator's Synthesis Query

```
synthesize(
    assume(t > 0),
    # Base case
    assert_that(
        simulate(model, 0) == simulate(sketch, 0)
    ),
    # Inductive hypothesis
    assume(
            simulate(model, t) == simulate(sketch, t)
    ),
    # Inductive step
    assert_that(
        simulate(model, t + 1) == simulate(sketch, t + 1)
    )
)
```

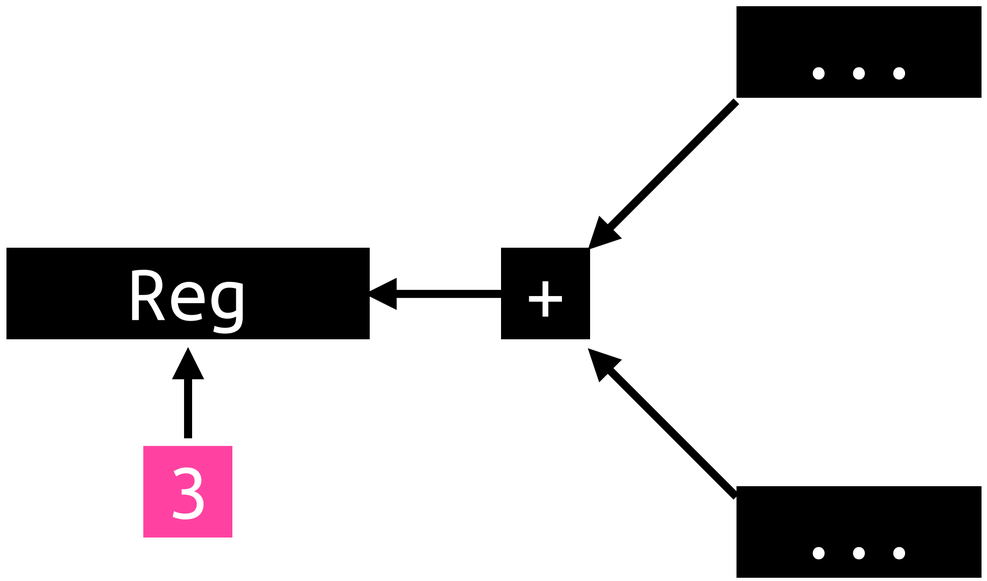
Gator's Solution:

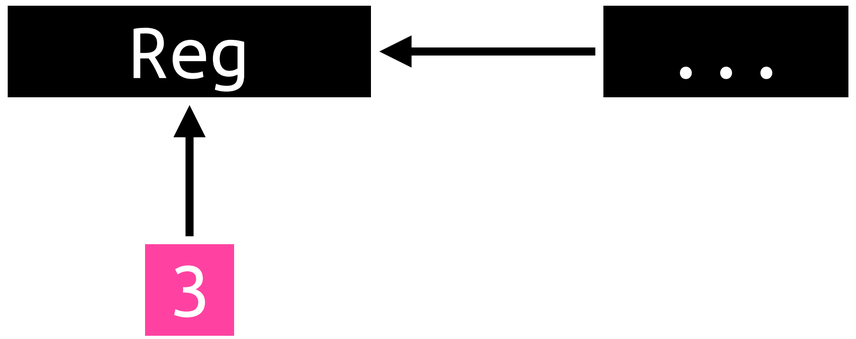


Gator's Solution:



Gator's Solution:






```
simulate(model, t) == 3
```

```
simulate(sketch, t) == 3
```

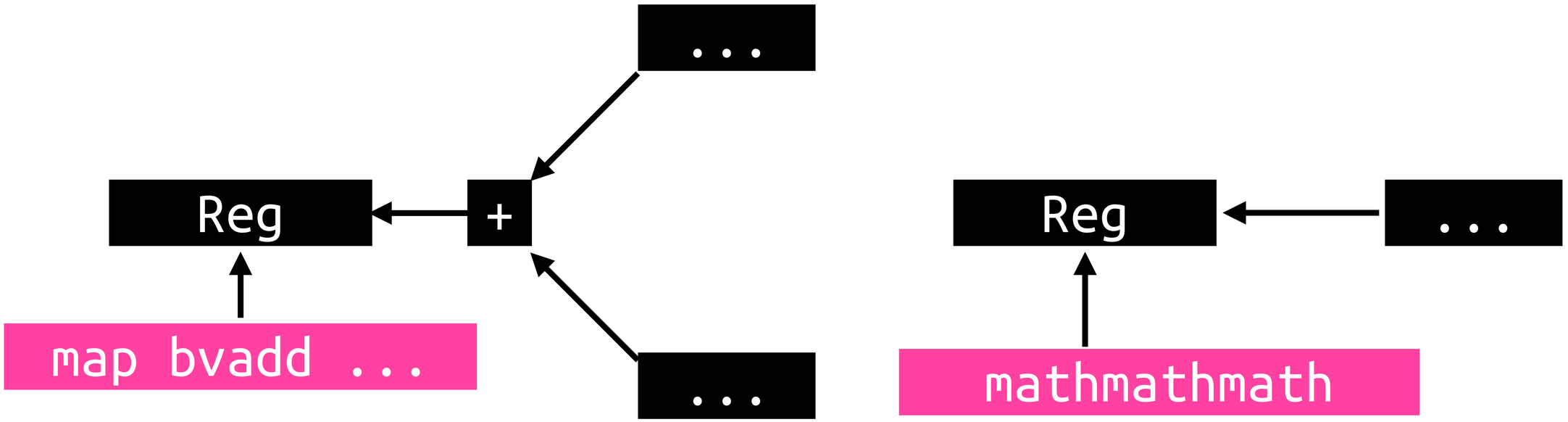
Gator's Solution:


```
simulate(model, t) == 3
```

```
simulate(sketch, t) == 3
```

This problem is out of scope; Gator only compiles hardware which outputs 3 at time t. QED

Gator's Solution:



The user has to put sound values into the cache.

The cache is good, all users have to do is put in sound values into the cache.

and, it works!

Three Problems:

1: How do you write the problem down? ✅

2. How do you encode inductive hypotheses? ✅

3. How do you make step 2 less painful?

Sound values are pure, functional expressions of what the hardware will do at some symbolic t.

wait

```python
def pure_model(inputs: List[Packet], stall_count: int) -> List[Packet]:
    return map(
        lambda out_port: (
            [DEFAULT_PACKET] * 4
            if stall_count > 0
            else foldl(
                lambda packet, accumulator: (
                    packet if packet.out_port == out_port else accumulator
                ),
                inputs,
                None,
            )
        ),
        range(4),
    )
```

```
def pure_model(inputs: List[Packet], stall_count: int) -> List[Packet]:
    return map(
        lar
            
umulator
        ),
        range(4),
    )
```

We didn't make *no* progress!

compile --> math --> verify
math --> compile

The Future

Three Problems:

1: How do you write the problem down? ✅

2. How do you encode inductive hypotheses? ✅

3. How do you make step 2 less painful?

Presenting: GatorGold

The power of the gator's bite, but without the pain ...

Presenting: GatorGold

The power of the gator's bite, but without the pain of manual cache population.

Reimagining Gator
What now?

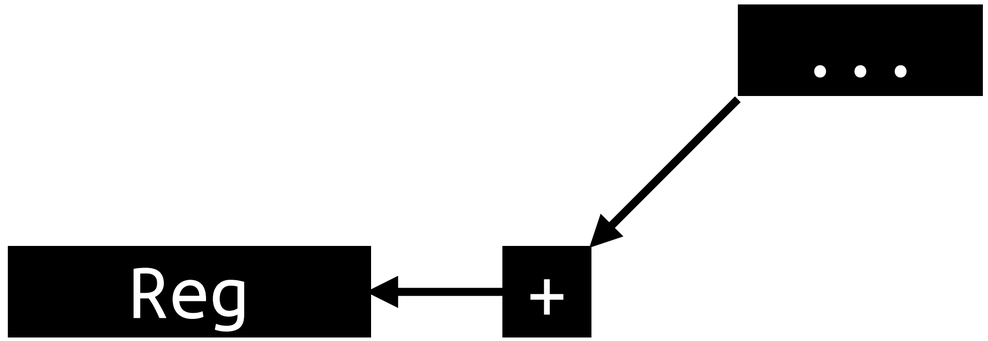
- Previously, we've represented equivalence via simulation.
- Is there a way of representing hardware without simulating it?

Reimagining Gator
What now?

- Previously, we've represented equivalence via simulation.
- Is there a way of representing hardware without simulating it?
  - Yes: streams!

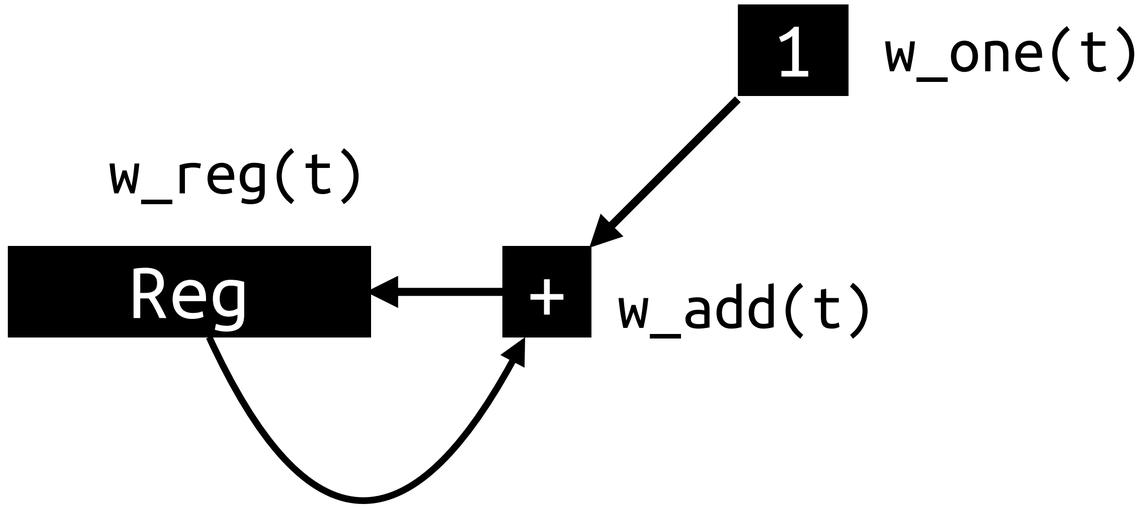
Reimagining Gator

Step 1: Represent each node as an uninterpreted function.



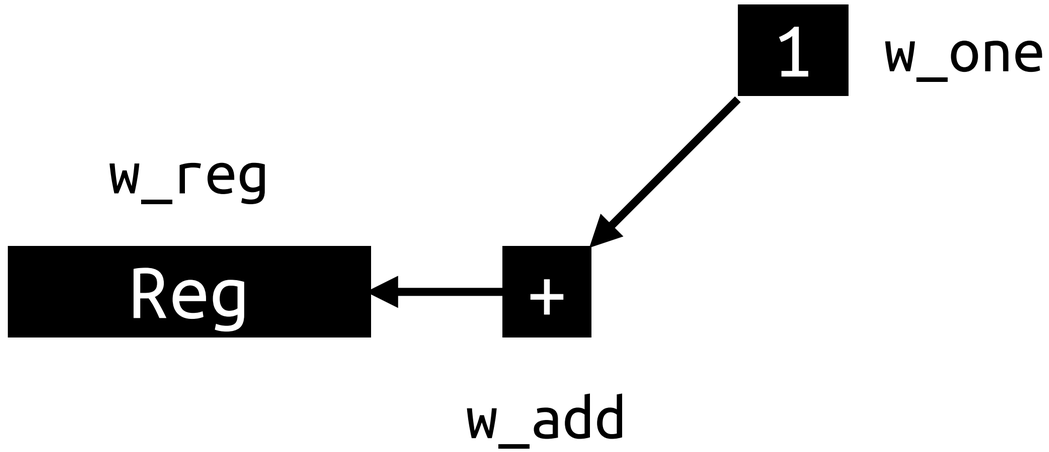
Reimagining Gator

Step 1: Represent each node as an uninterpreted function.



Reimagining Gator

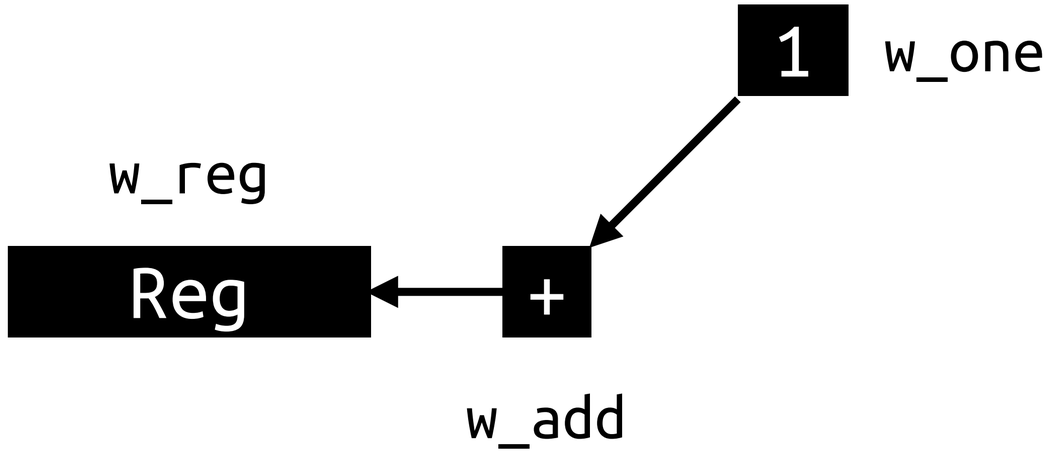
Step 2: Write a recursive expression for each function.




```
w_one(t) = 1
```

Reimagining Gator

Step 2: Write a recursive expression for each function.

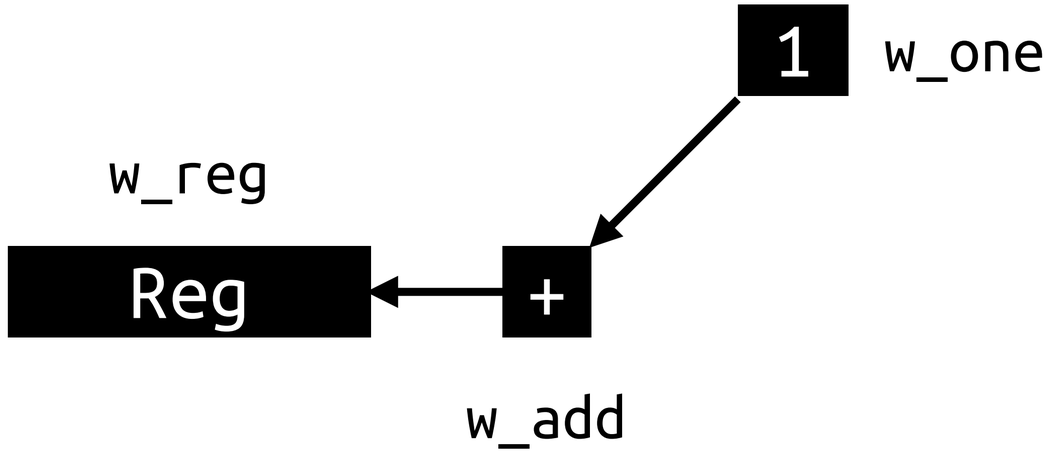



```
w_one(t) = 1

w_add(t) = w_one(t) + w_reg(t)
```

Reimagining Gator

Step 2: Write a recursive expression for each function.



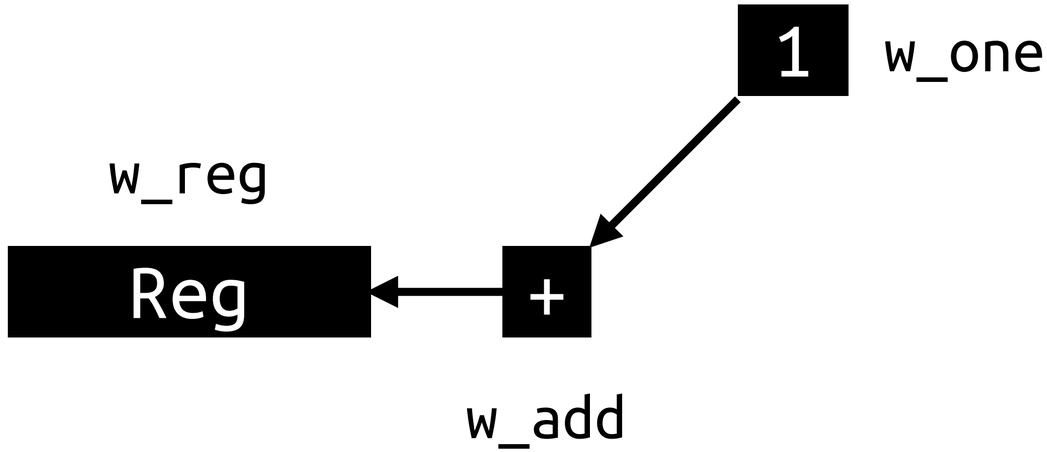

```
w_one(t) = 1

w_add(t) = w_one(t) + w_reg(t)

w_reg(t) =
   if t = 0 then 0 else
       w_add(t - 1)
```

Reimagining Gator

Step 2: Write a recursive expression for each function.



The solver won't try to run this!

```
w_one(t) = 1

w_add(t) = w_one(t) + w_reg(t)

w_reg(t) =
    if t = 0 then 0 else
        w_add(t - 1)
```

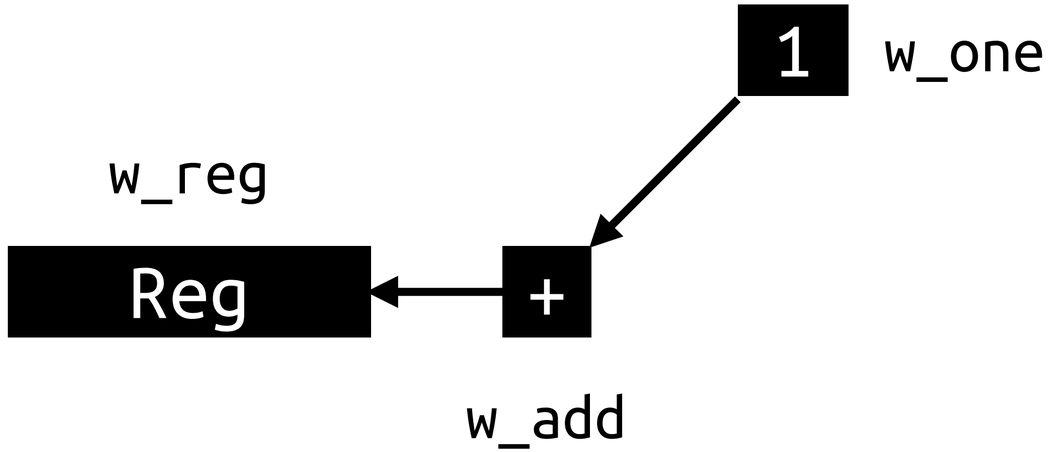
Reimagining Gator

Step 3: Profit!

```
w_one(t) = 1

w_add(t) = w_one(t) + w_reg(t)

w_reg(t) =
    if t = 0 then 0 else
        w_add(t - 1)
```




```
assume(w_reg(t) == w_sketch(t))

assert(w_reg(t + 1) == w_sketch(t + 1))
```

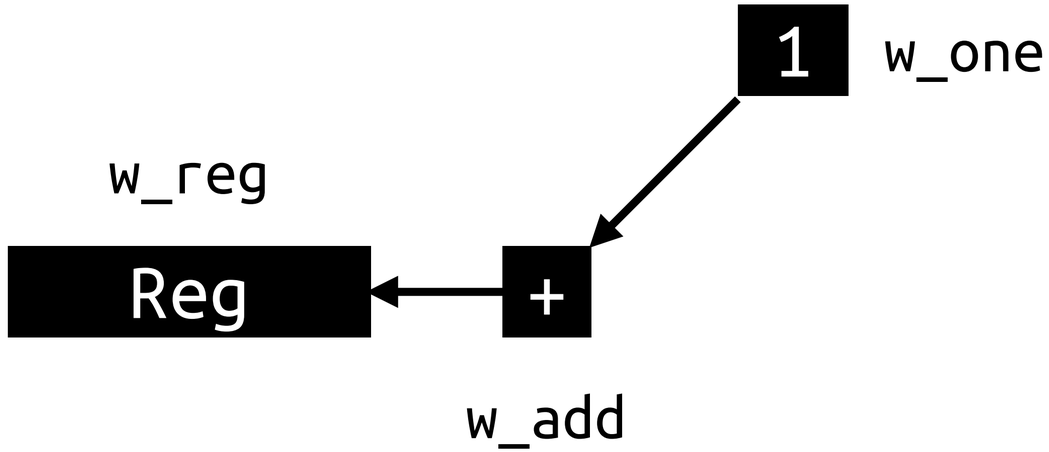
Reimagining Gator

Step 3: Profit!

```
w_one(t) = 1

w_add(t) = w_one(t) + w_reg(t)

w_reg(t) =
    if t = 0 then 0 else
        w_add(t - 1)
```



No math required!

```
assume(w_reg(t) == w_sketch(t))

assert(w_reg(t + 1) == w_sketch(t + 1))
```

This actually works, sometimes!

Presenting: GatorGold

This actually works, sometimes!

Three Problems:

1: How do you write the problem down? ✅

2. How do you encode inductive hypotheses? ✅

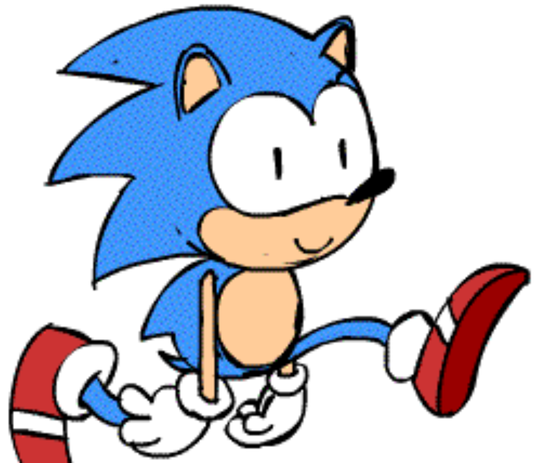
3. How do you make step 2 less painful? 🏃

Three Problems:

1: How do you write the problem down? ✅

2. How do you encode inductive hypotheses? ✅

3. How do you make step 2 less painful?



The Future

What did we learn?

- It's possible to do in-synthesis verification!
- It involves using a cache to avoid infinite cycles.
- And, it involves using a cache to encode inductive hypotheses.
- And, it is painful.
- ...but maybe not if you use streams.
- And, we built a tool to do this!