Glenside

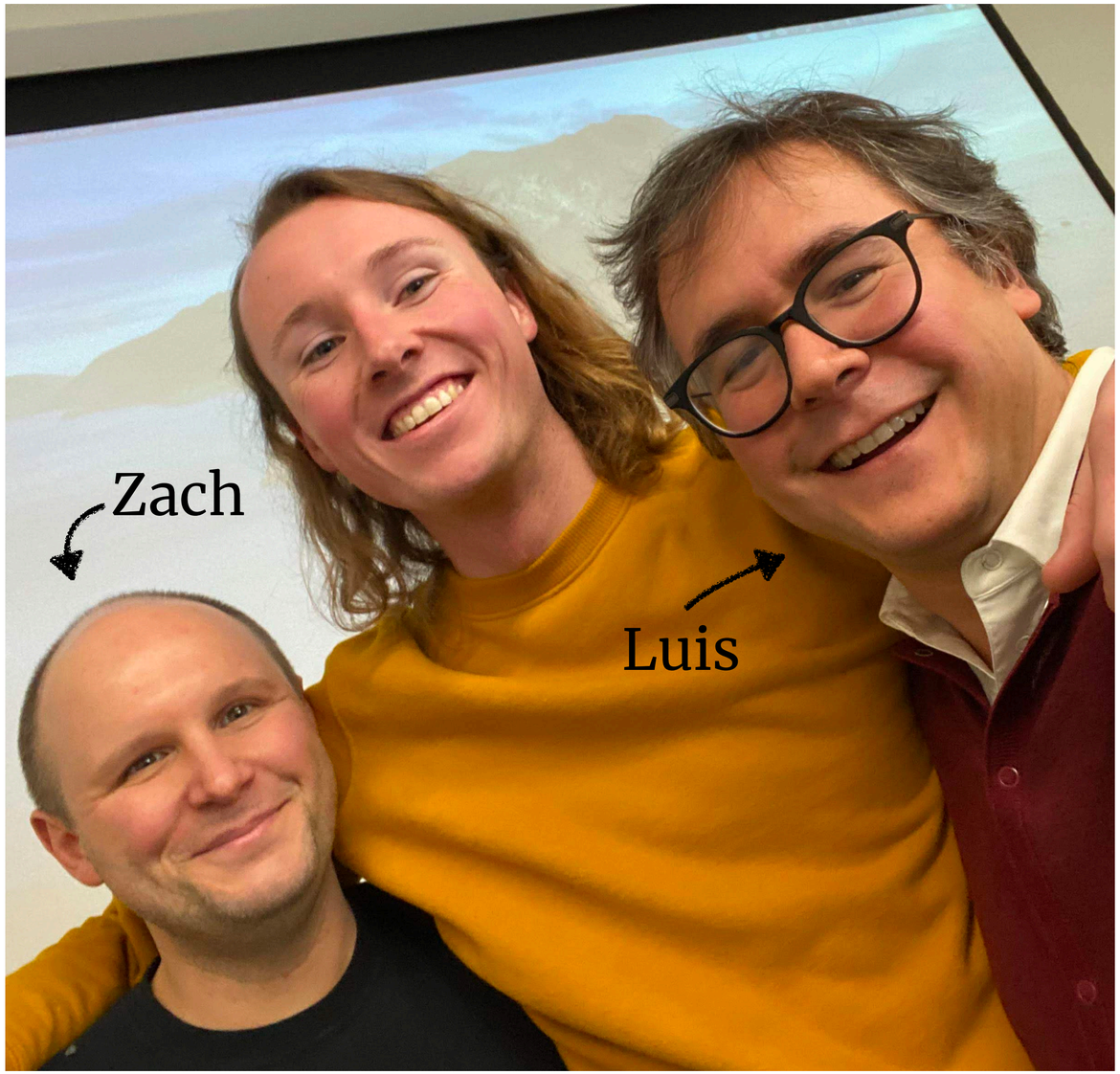
Partitioning Deep Learning Hardware and Software

Gus Smith, University of Washington

PL/Arch External Talk Series @ CAPRA, May 27th, 2020

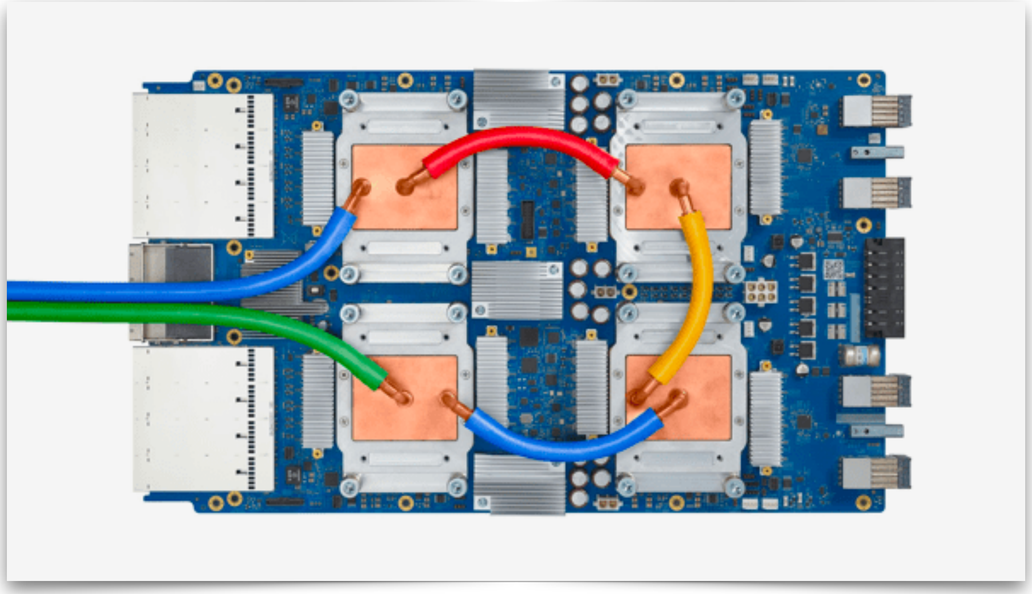
Who?

- 2nd year at UW working in PL/ Arch with Zach Tatlock/Luis Ceze
- Interested in how we can advance architecture w/ PL techniques

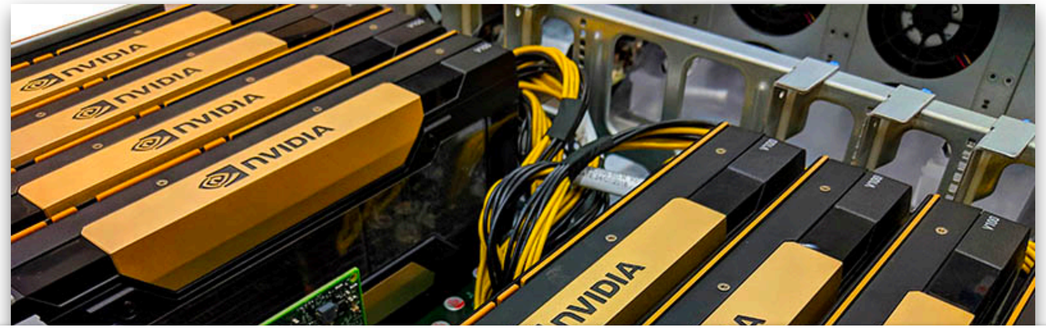


⚠ WIP ⚠

Architects accelerate deep learning with custom hardware.



Architects accelerate deep learning with custom hardware.



Share

SHARE

TWEET

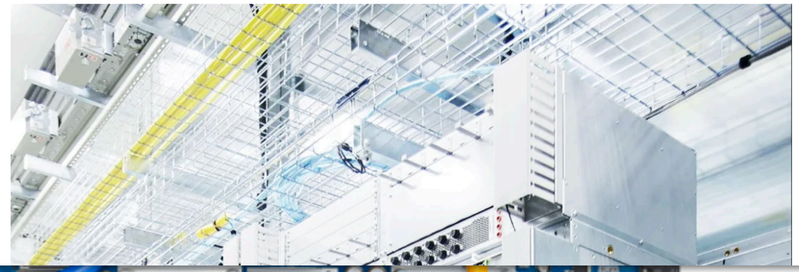
COMMENT

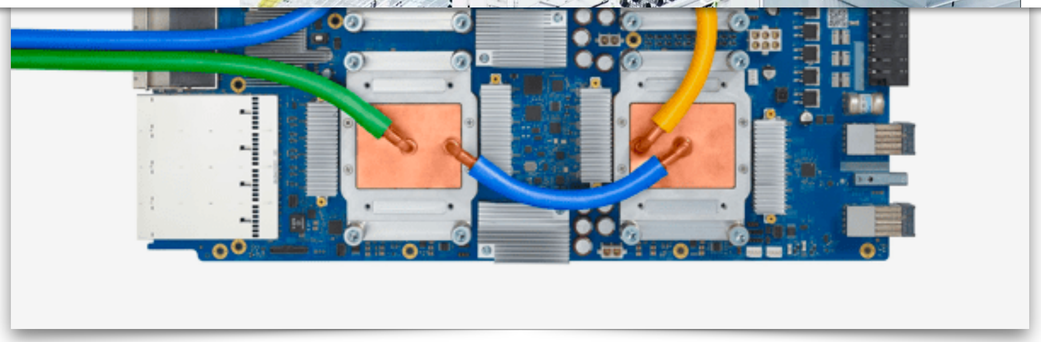
EMAIL

CADE METZ 04.05.17 12:03 PM

WIRED

Building an AI Chip Saved Google From Building a Dozen New Data Centers





...but designing deep learning stacks is complex!

In-Datacenter Performance Analysis of a Tensor Processing Unit™

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon

Google, Inc., Mountain View, CA USA

Email: {jouppi, cliffy, nishantpatil, davidpatterson}@google.com

To appear at the 44th International Symposium on Computer Architecture (ISCA), Toronto, Canada, June 26, 2017.

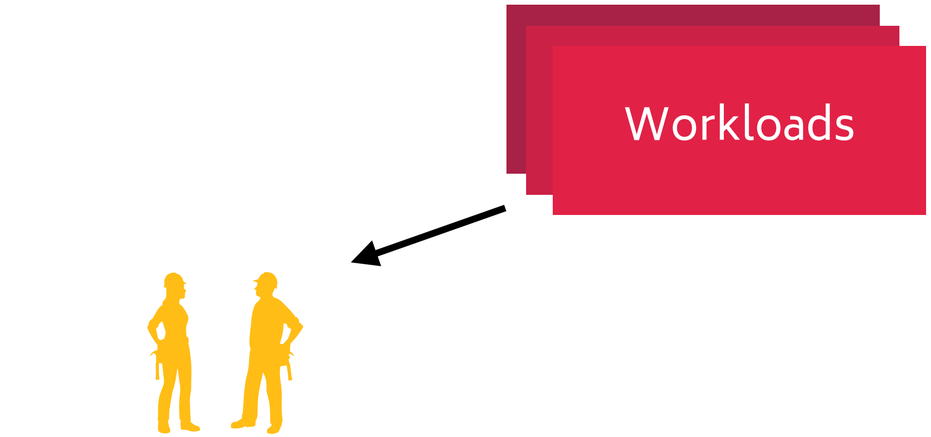
TensorFlow:
Large-Scale Machine Learning on Heterogeneous Distributed Systems

(Preliminary White Paper, November 9, 2015)

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro,
Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow,
Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser,
Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray,
Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar,
Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals,
Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng
Google Research*

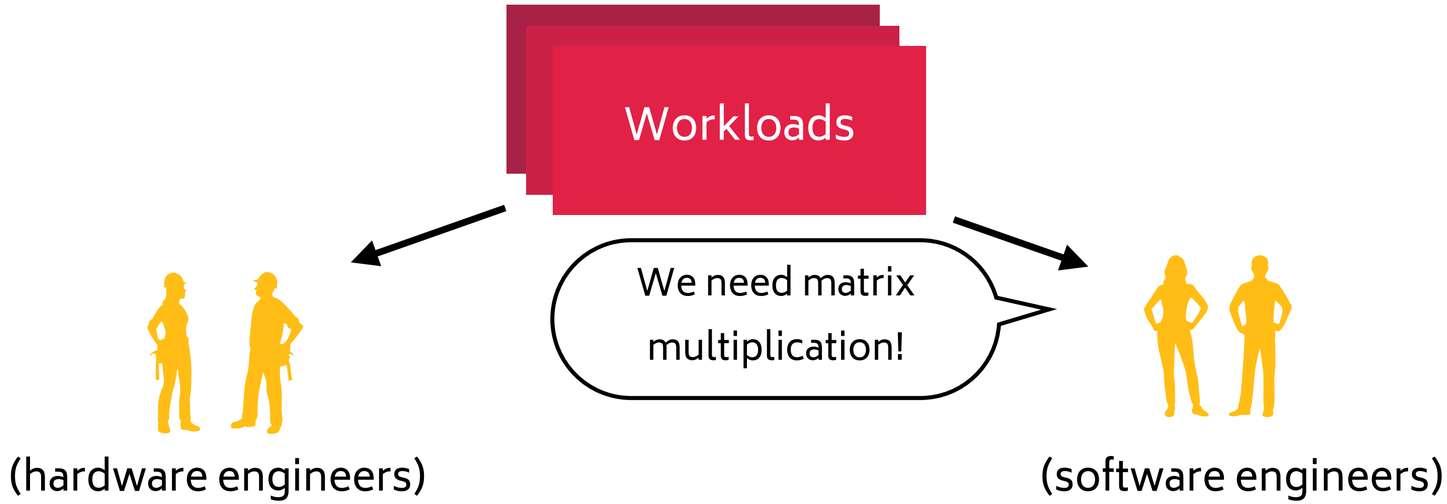
An Oversimplified View of HW–SW stack design

An Oversimplified View of HW–SW stack design

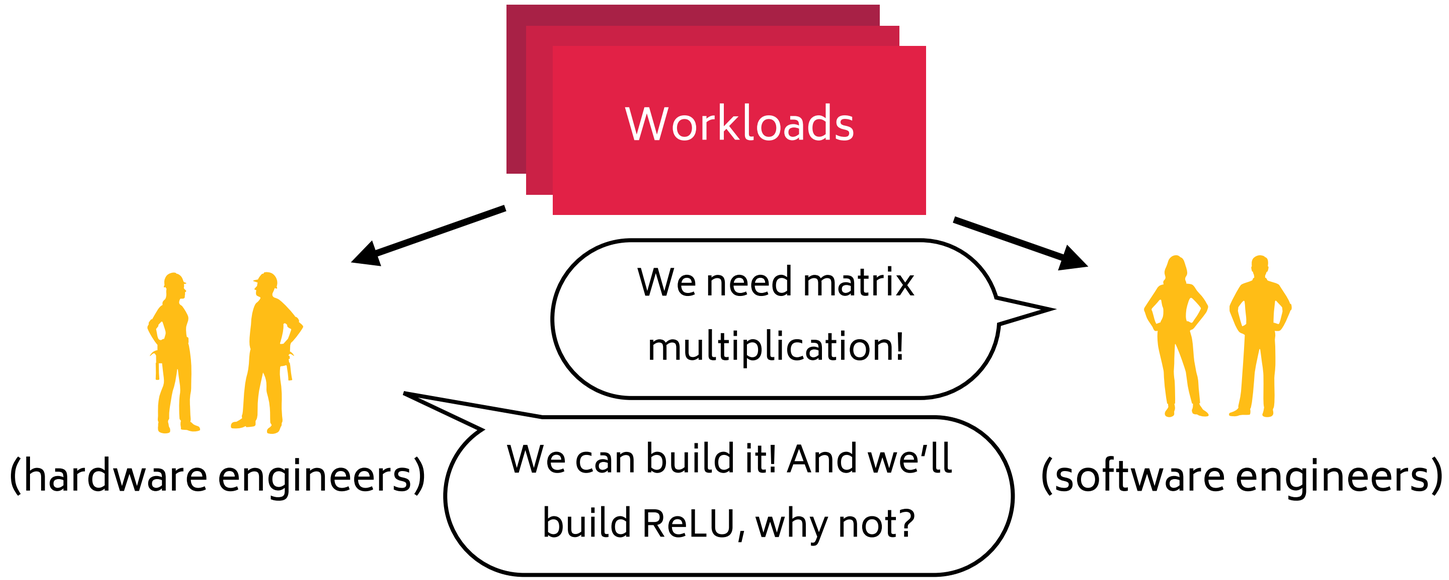


An Oversimplified View of HW–SW stack design

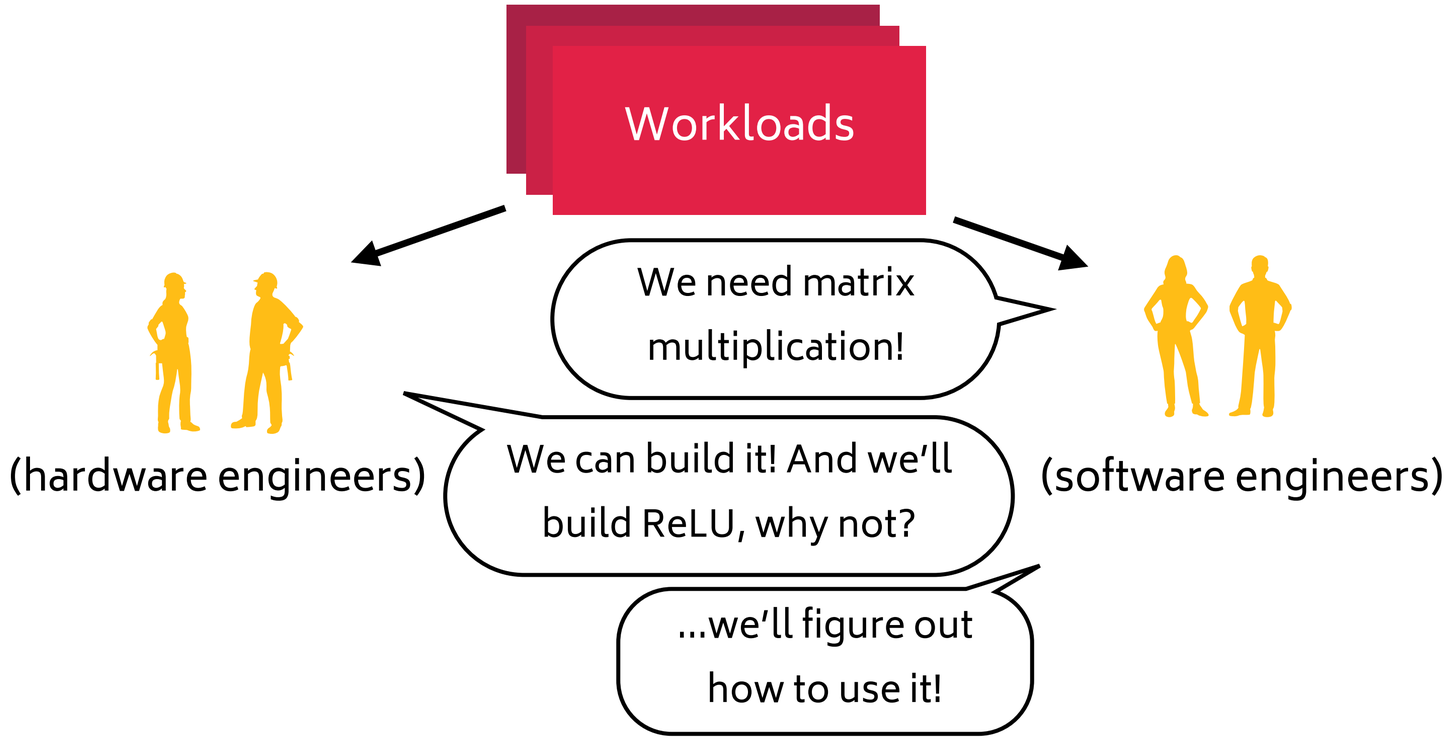
An Oversimplified View of HW–SW stack design



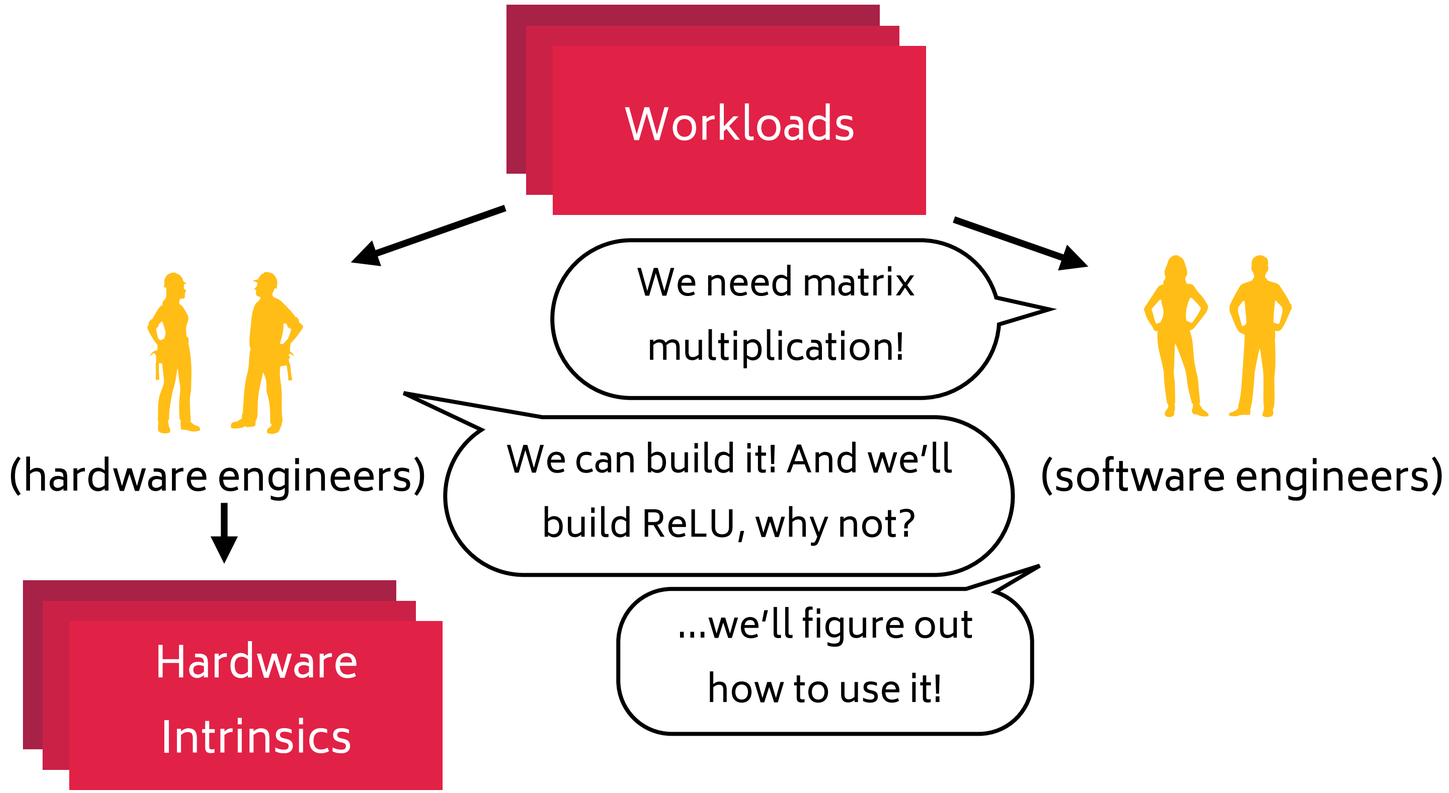
An Oversimplified View of HW–SW stack design



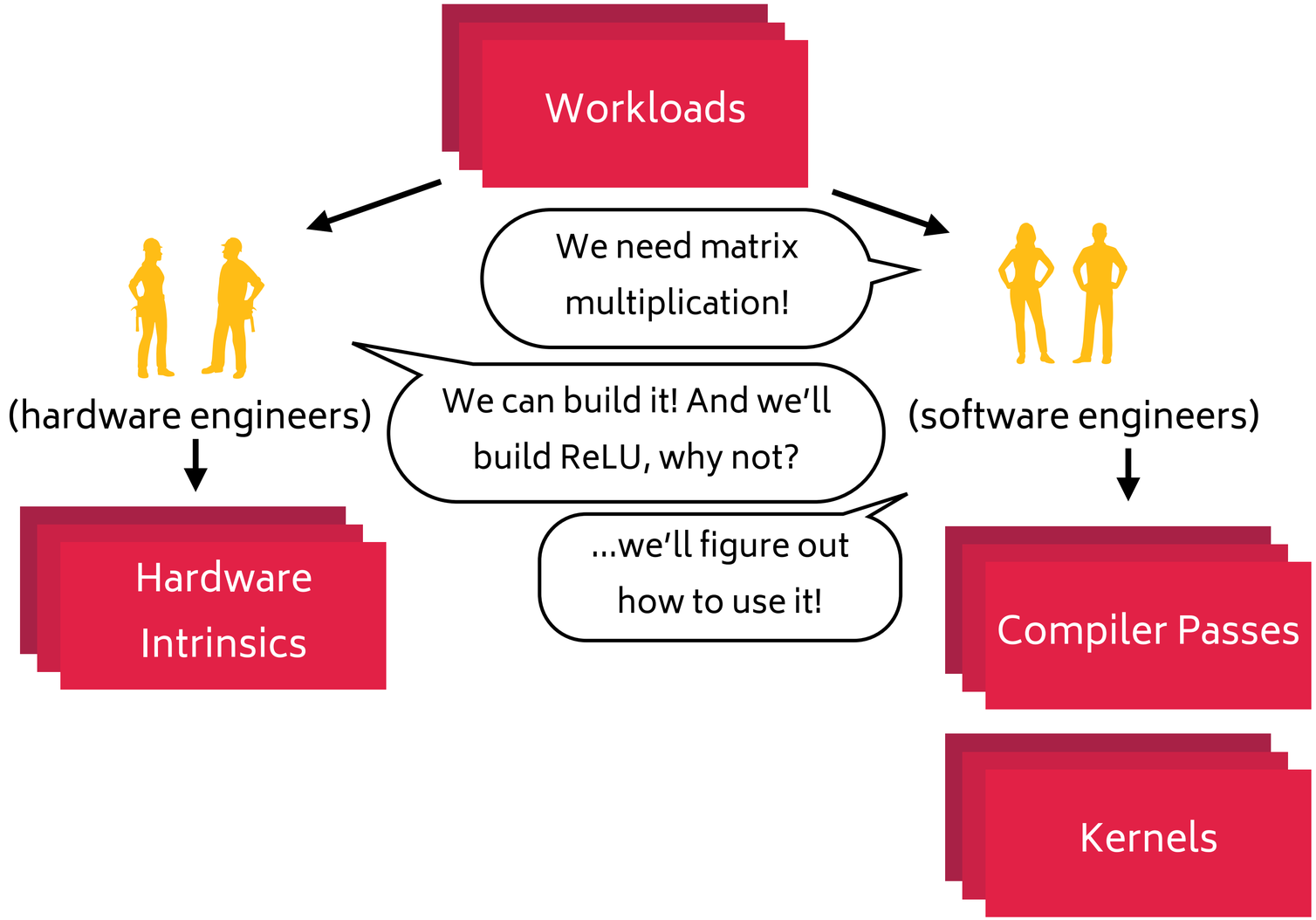
An Oversimplified View of HW–SW stack design



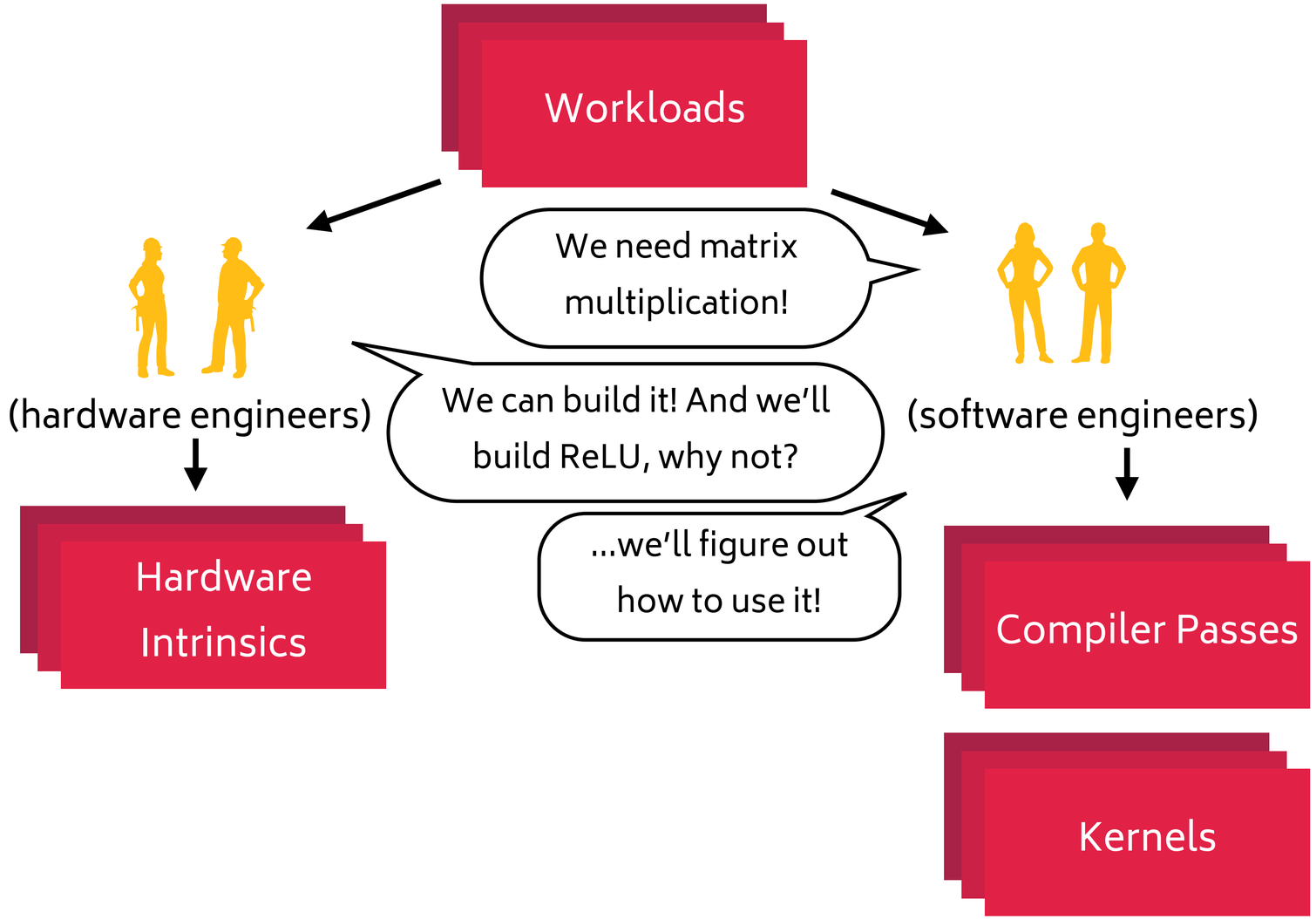
An Oversimplified View of HW–SW stack design



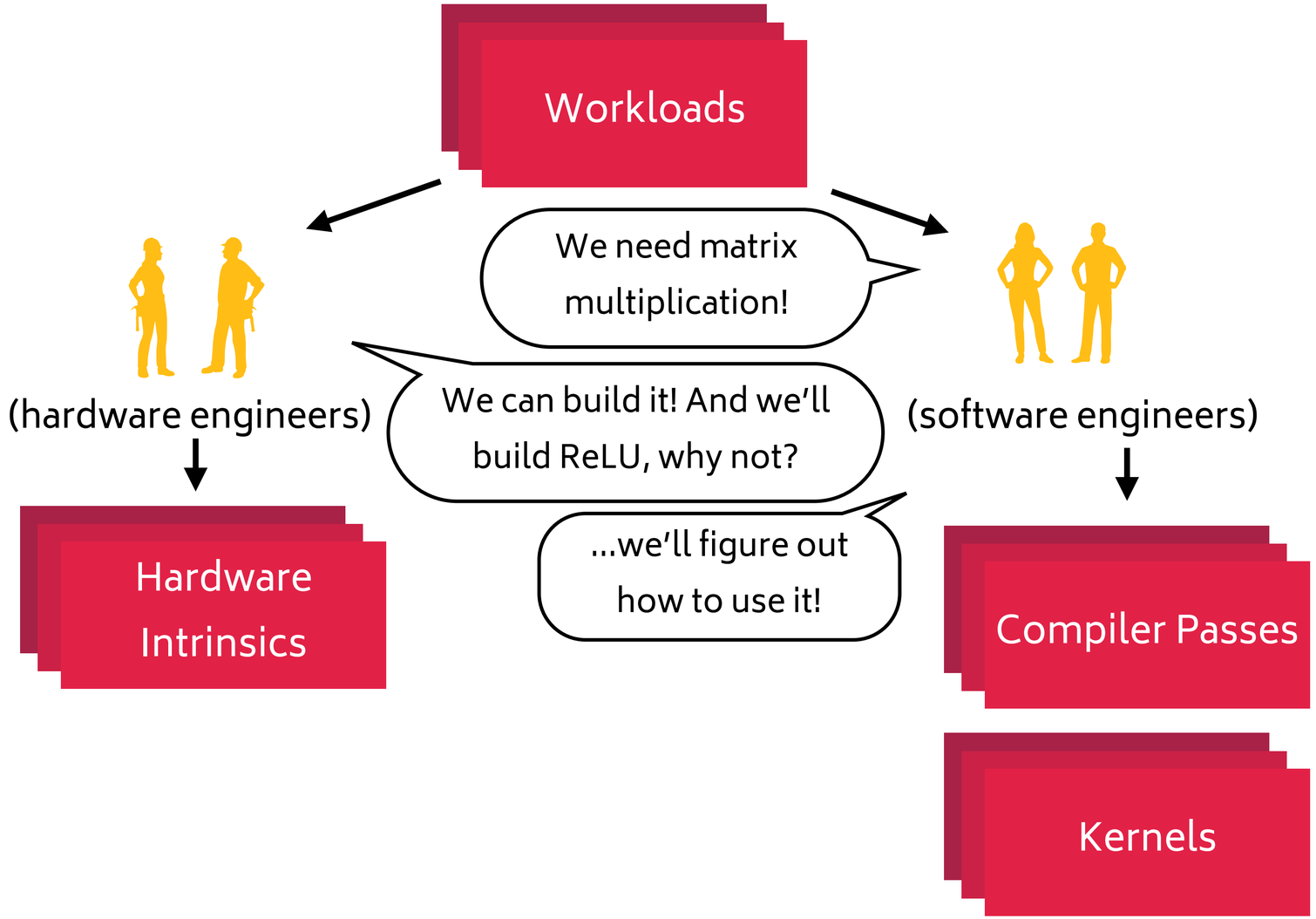
An Oversimplified View of HW–SW stack design



An Oversimplified View of HW–SW stack design

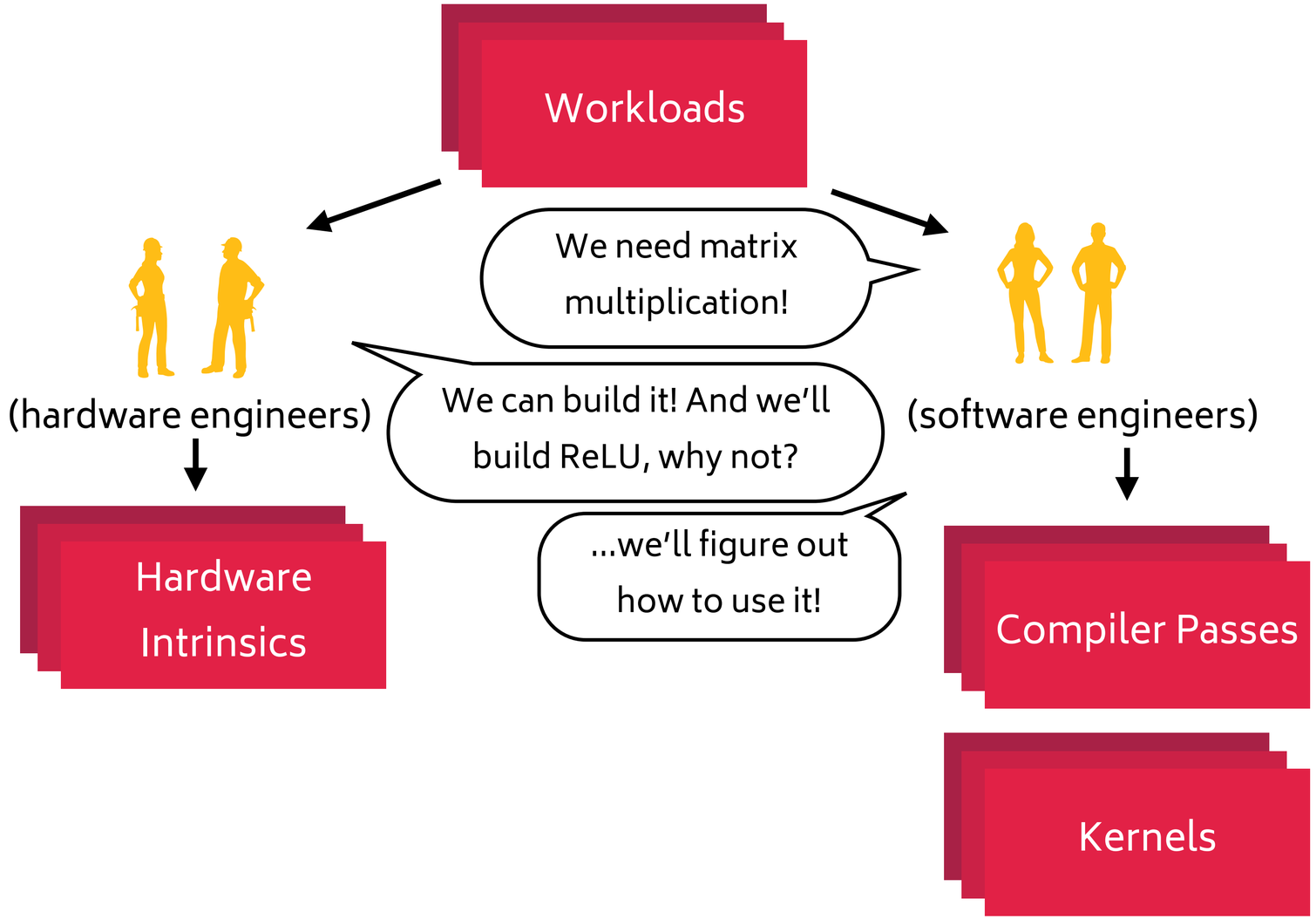


An Oversimplified View of HW–SW stack design



This system makes sense because chips and compilers are massive projects.

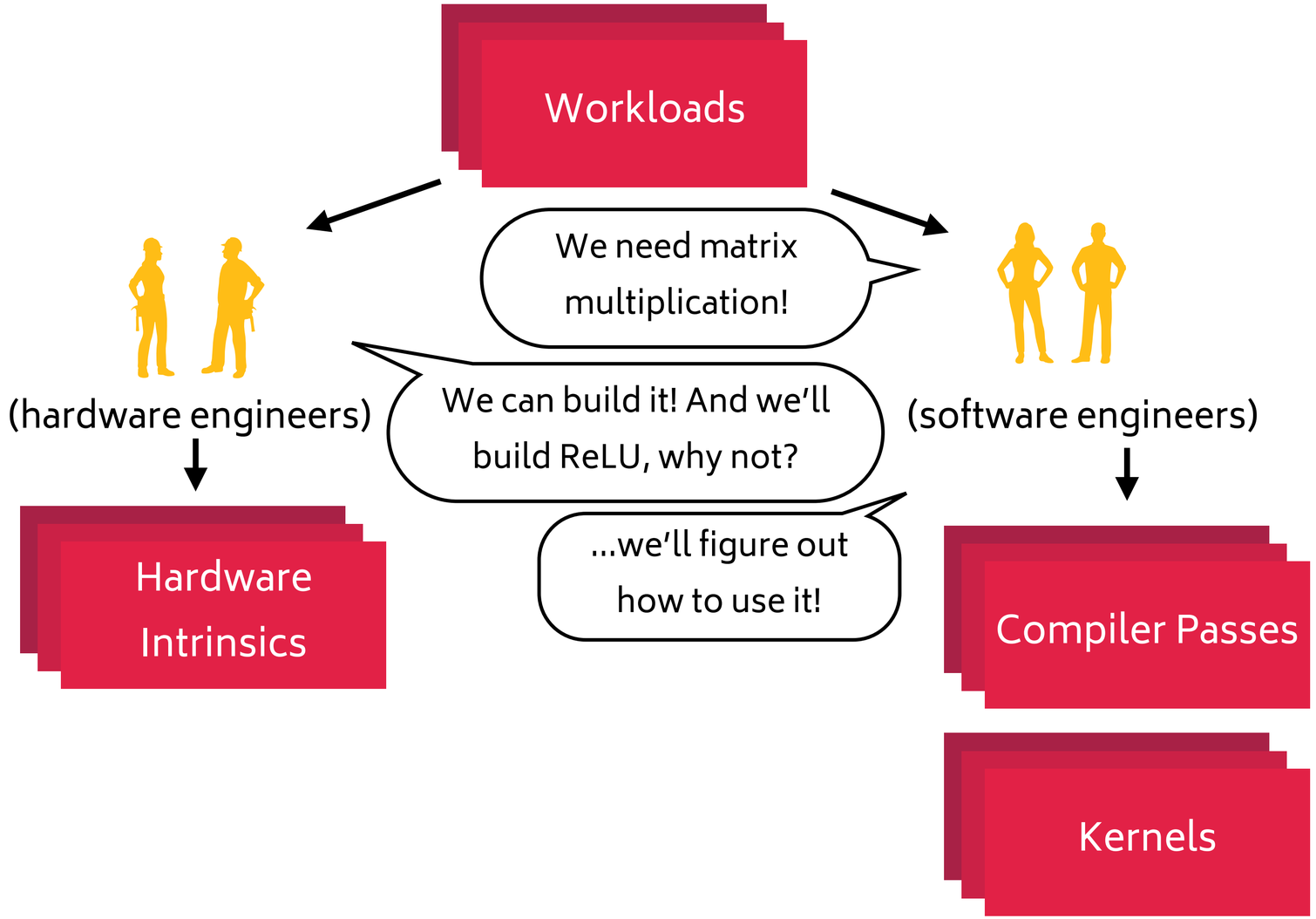
An Oversimplified View of HW–SW stack design



This system makes sense because chips and compilers are massive projects.

...but we can't help but notice the inefficiencies!

An Oversimplified View of HW–SW stack design

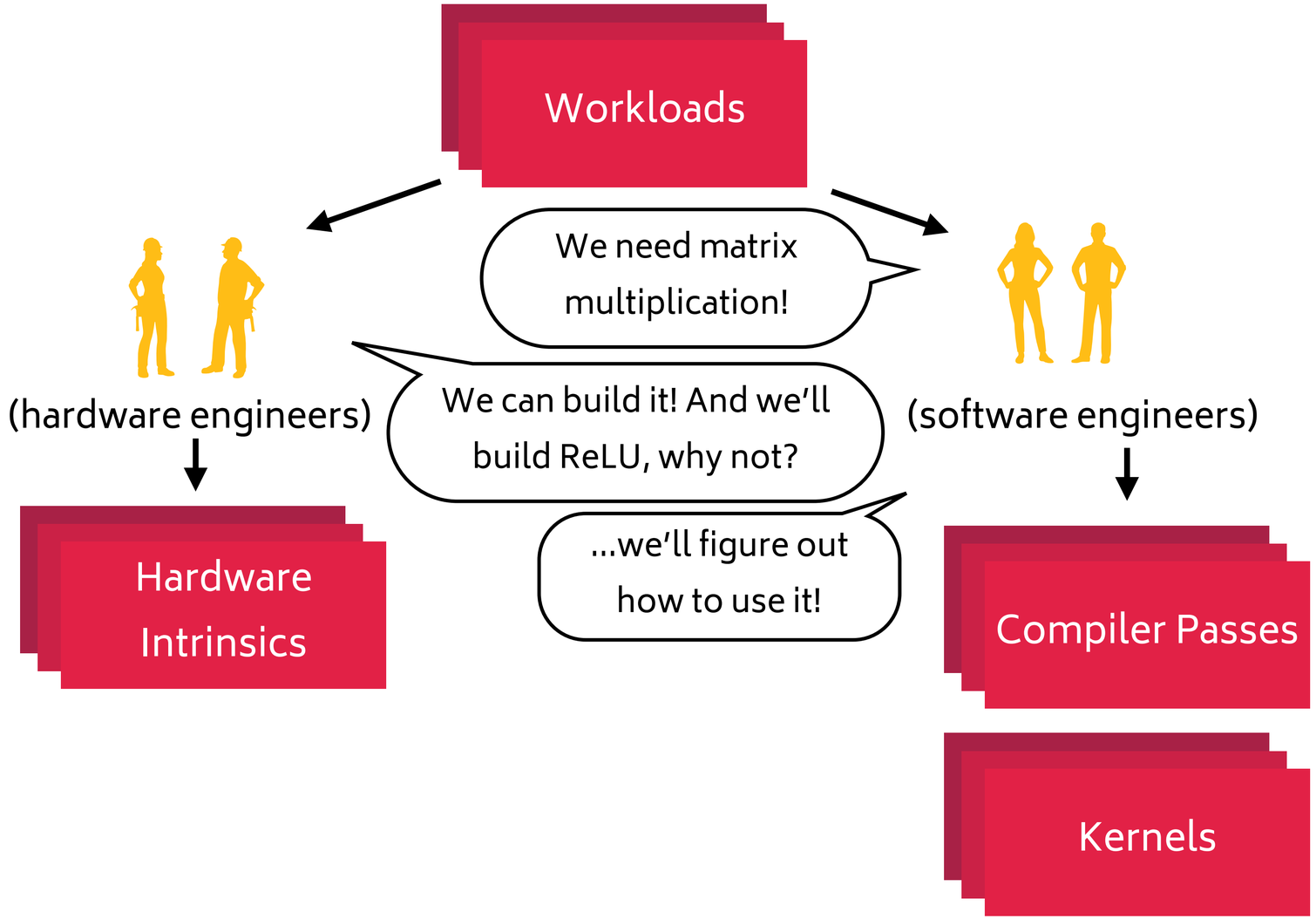


This system makes sense because chips and compilers are massive projects.

...but we can't help but notice the inefficiencies!

- Slow

An Oversimplified View of HW–SW stack design

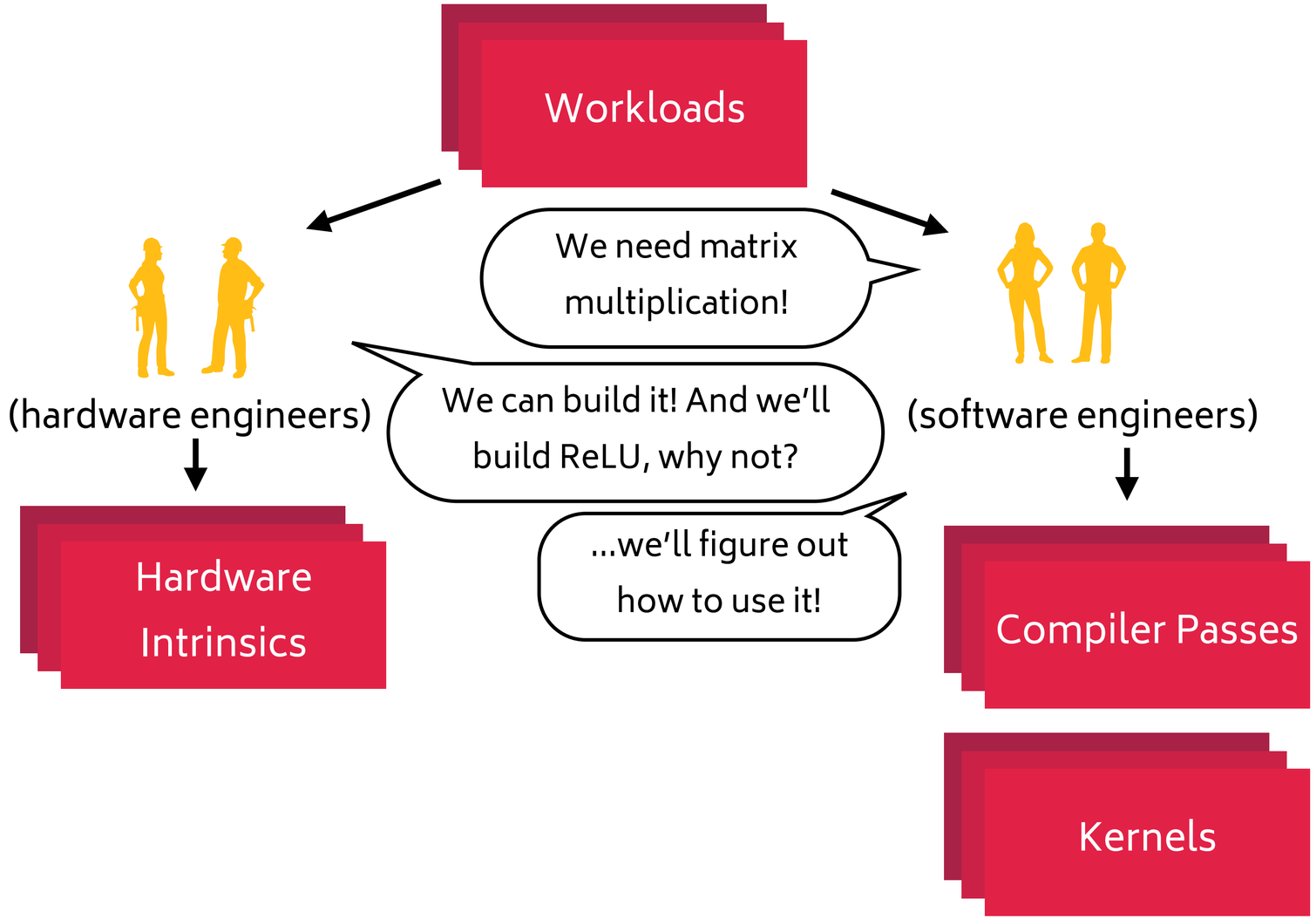


This system makes sense because chips and compilers are massive projects.

...but we can't help but notice the inefficiencies!

- Slow
- Misses design points

An Oversimplified View of HW–SW stack design



This system makes sense because chips and compilers are massive projects.

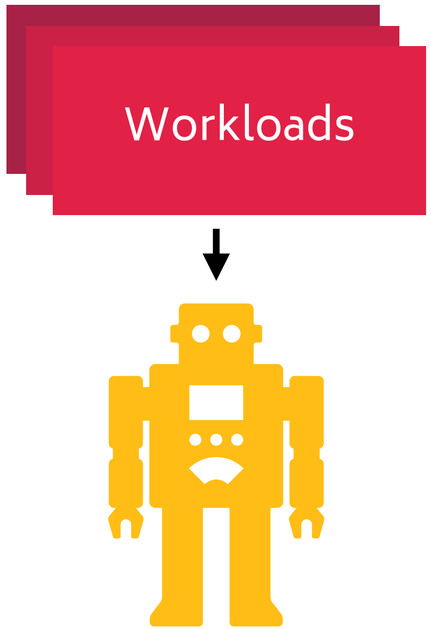
...but we can't help but notice the inefficiencies!

- Slow
- Misses design points
- Design knowledge is lost in communication!

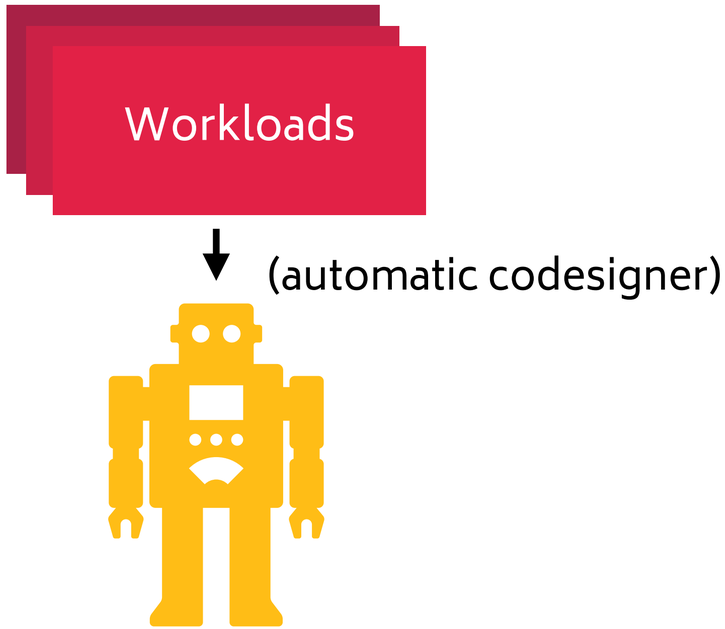
Why not design the hardware and software at the same time?

A Hardware–Software Codesign Approach

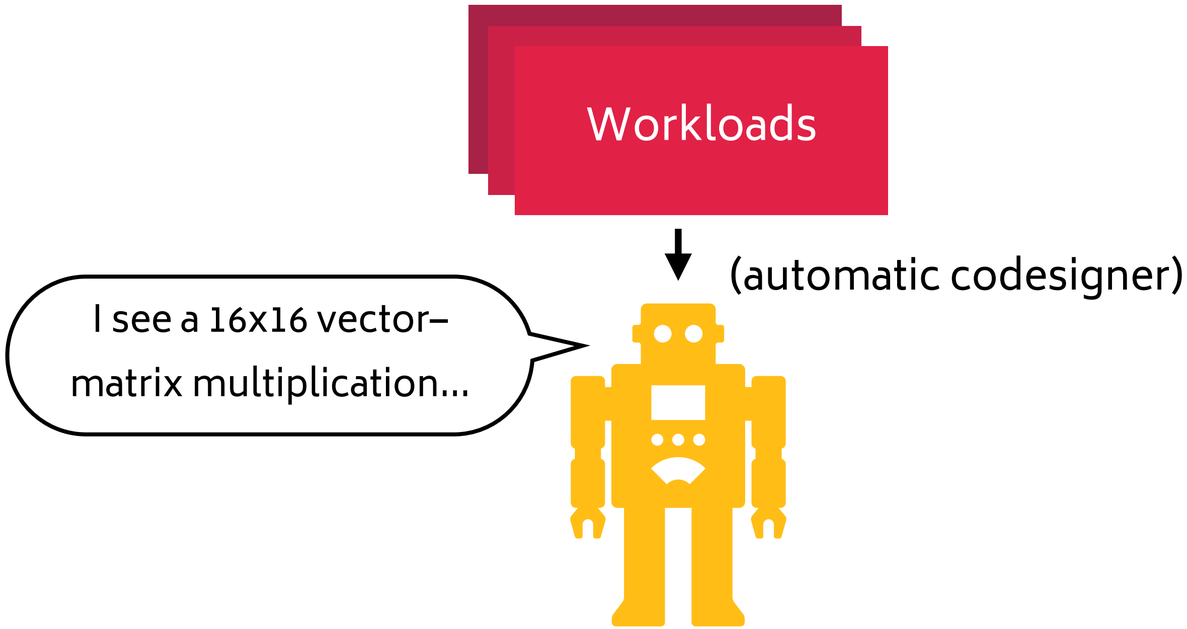
A Hardware–Software Codesign Approach



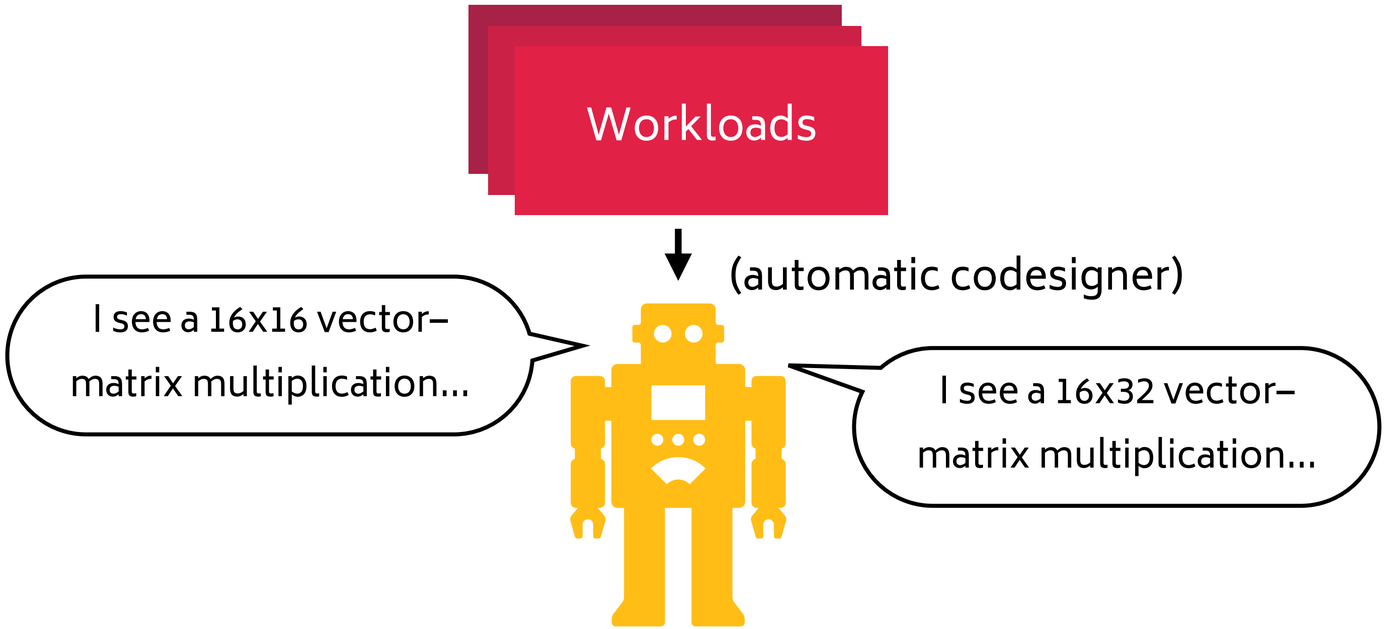
A Hardware–Software Codesign Approach



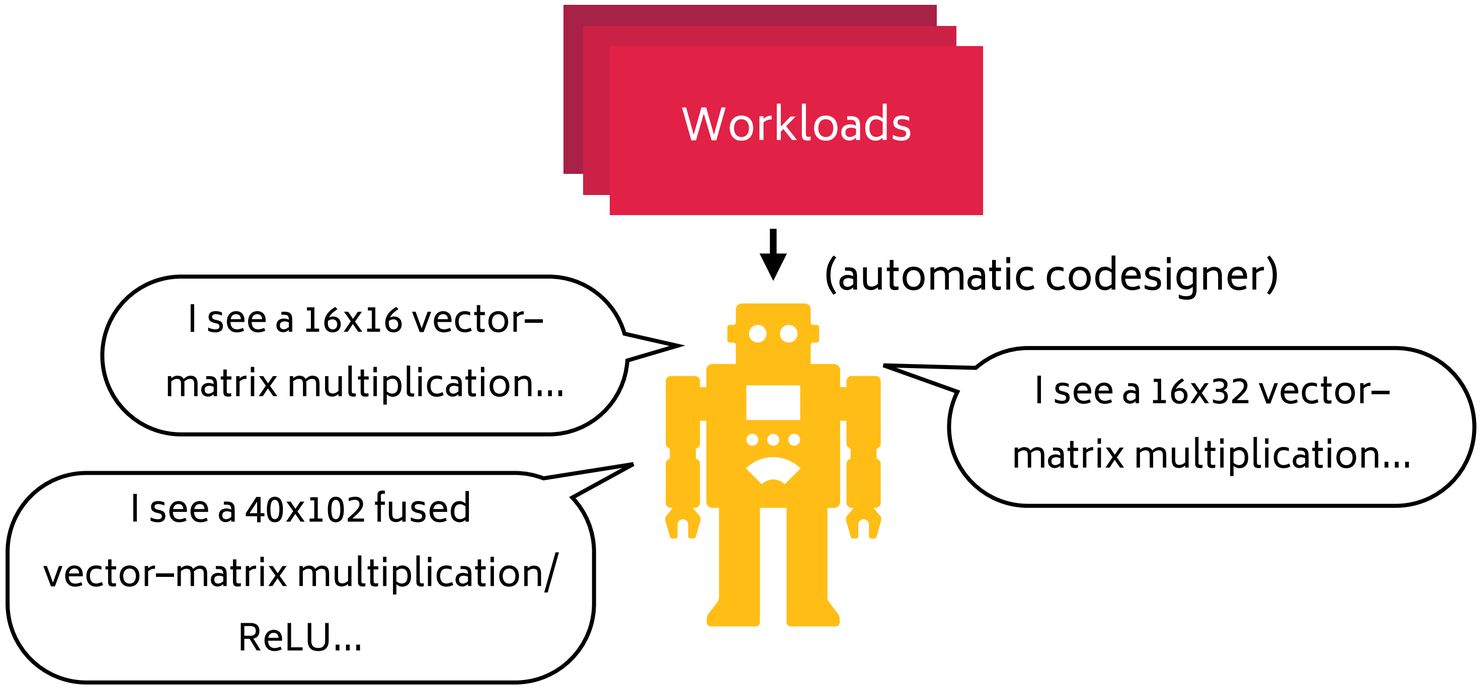
A Hardware–Software Codesign Approach



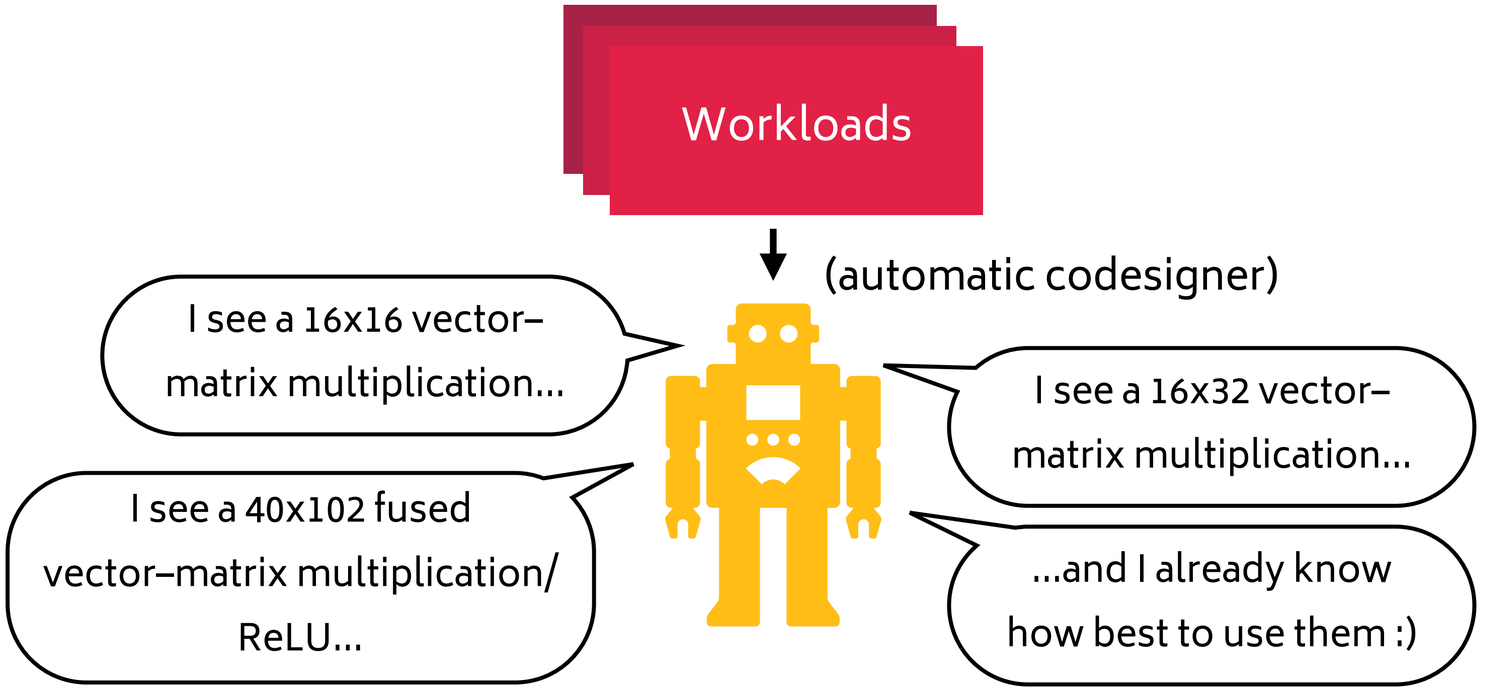
A Hardware–Software Codesign Approach



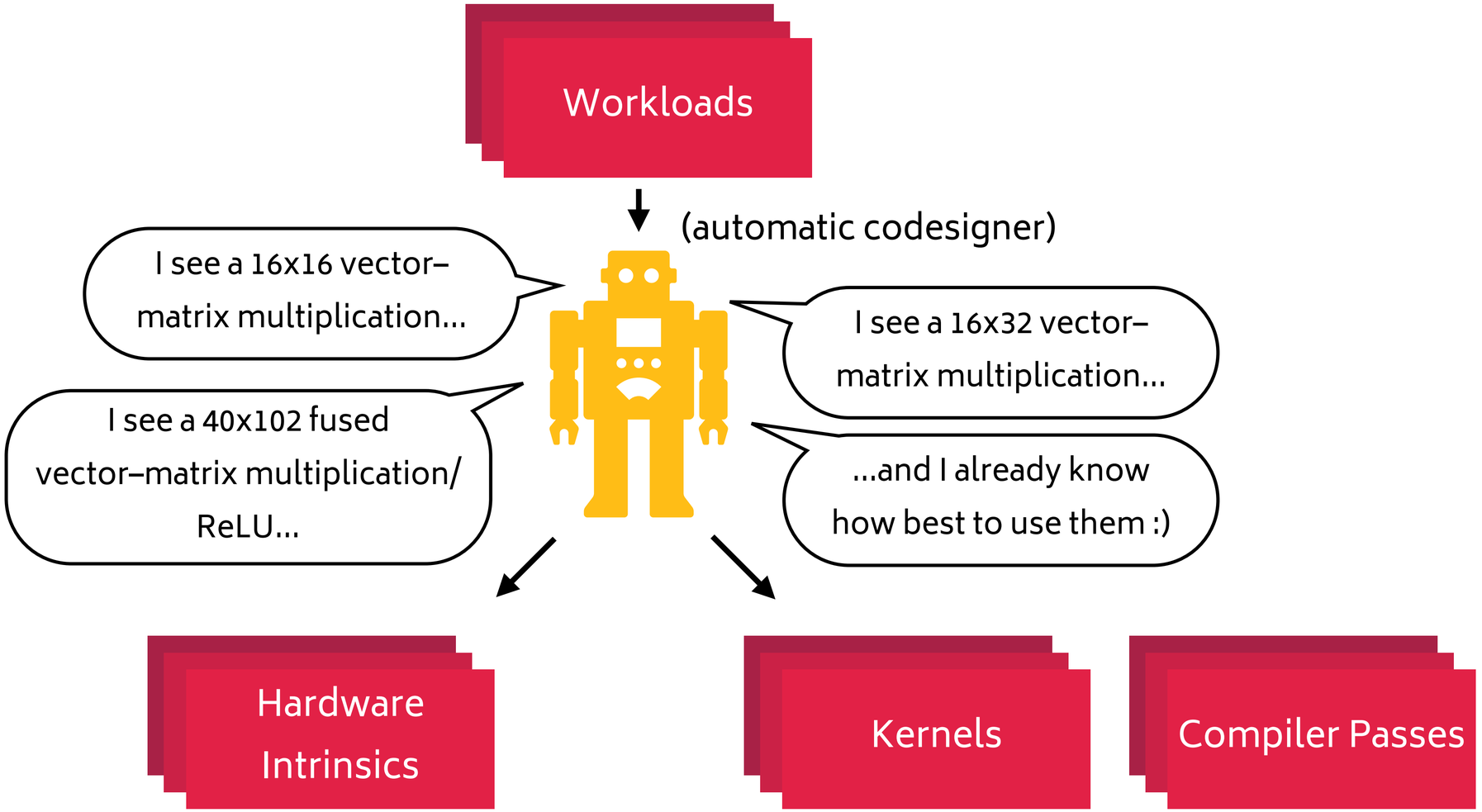
A Hardware–Software Codesign Approach



A Hardware–Software Codesign Approach



A Hardware–Software Codesign Approach



…this is not a new idea. **Hardware–software codesign** is a dream that people have had for some time.

…this is not a new idea. **Hardware–software codesign** is a dream that people have had for some time.

Hardware–software codesign often raises a healthy amount of skepticism due to its meager success in the past.

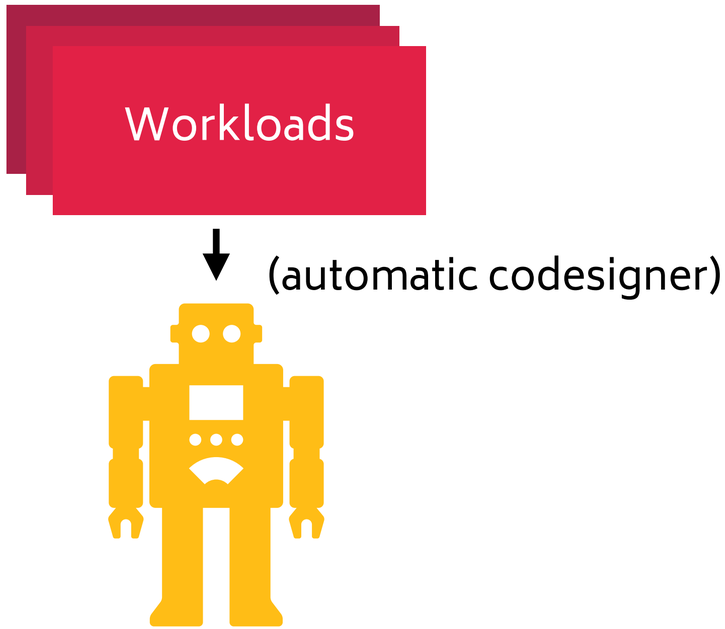
…this is not a new idea. **Hardware–software codesign** is a dream that people have had for some time.

Hardware–software codesign often raises a healthy amount of skepticism due to its meager success in the past.

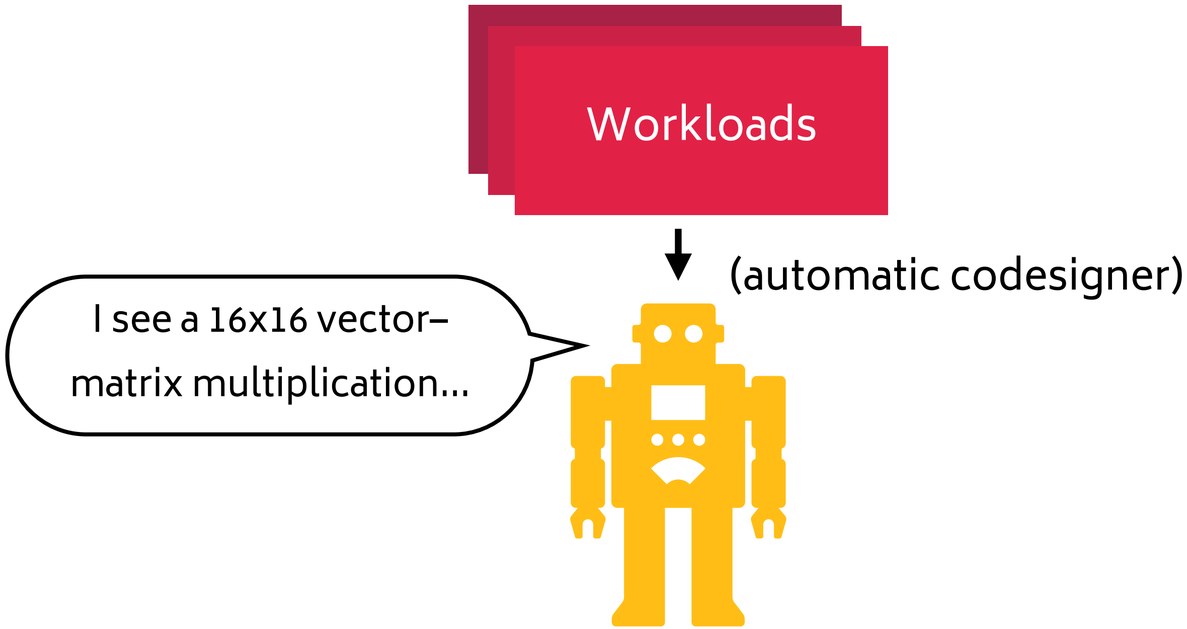
One of the primary issues is that **the space of potential hardware–software designs is massive!**

A Hardware–Software Codesign Approach

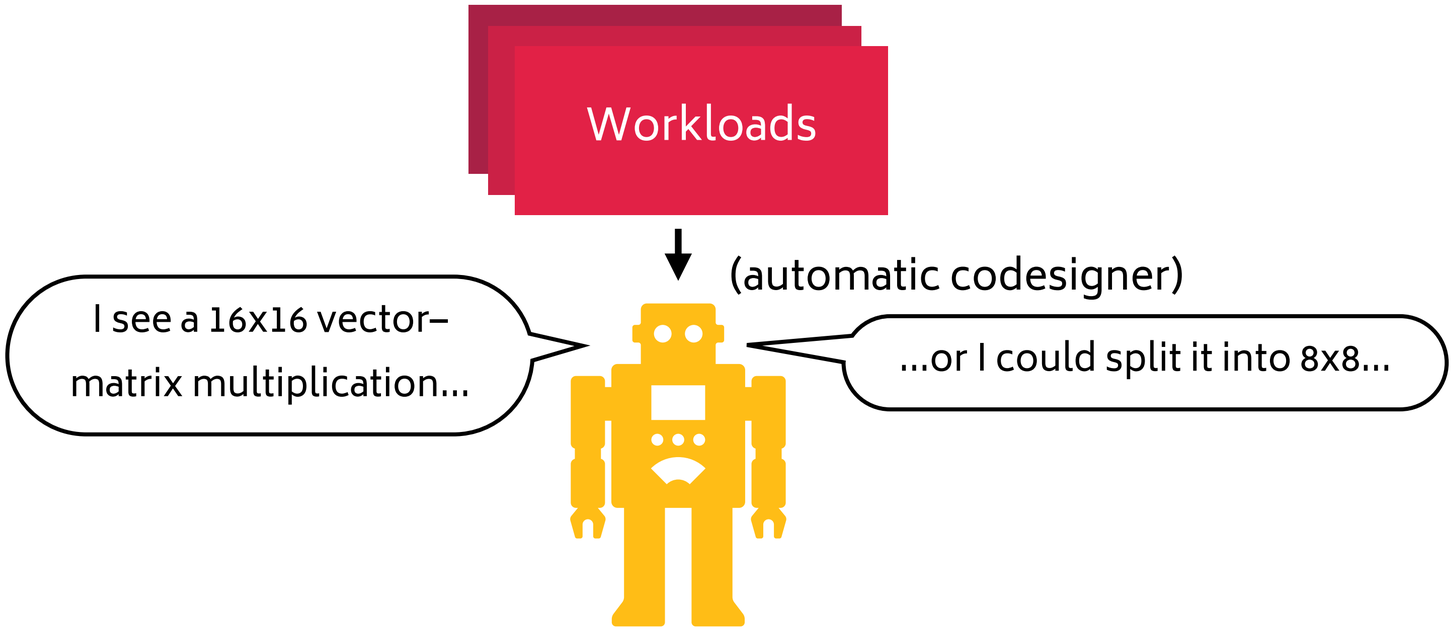
A Hardware–Software Codesign Approach



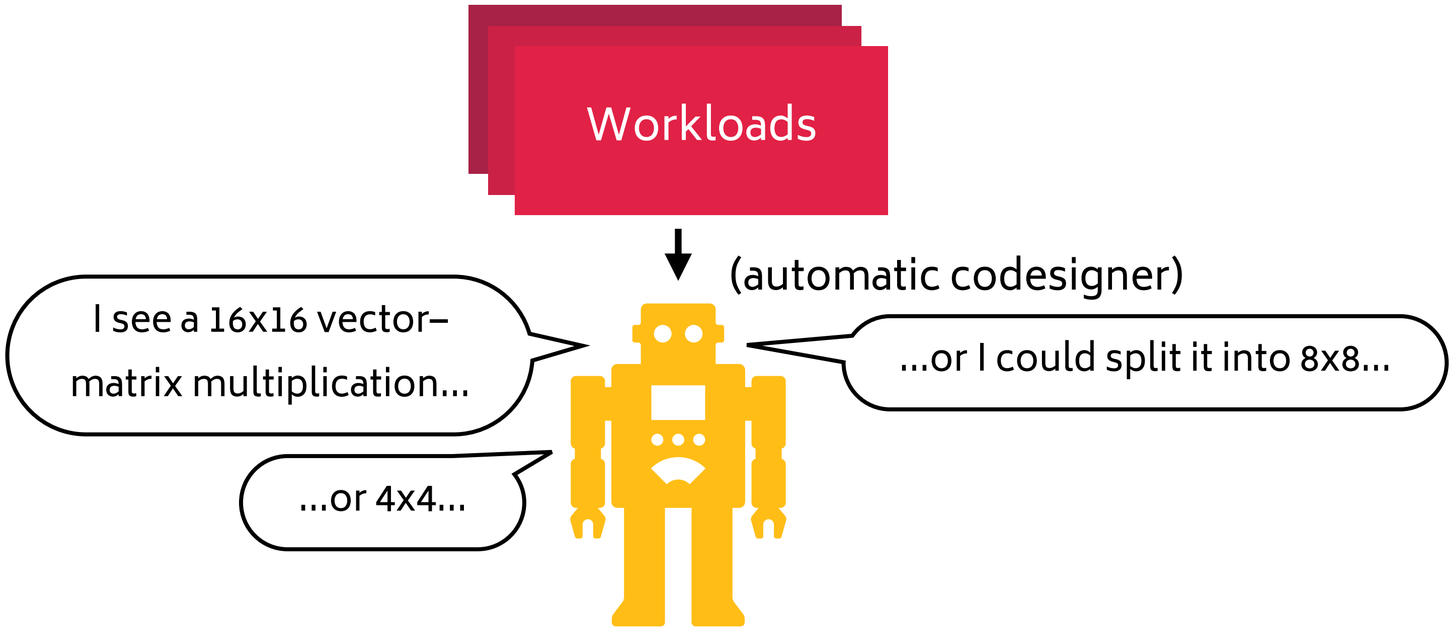
A Hardware–Software Codesign Approach



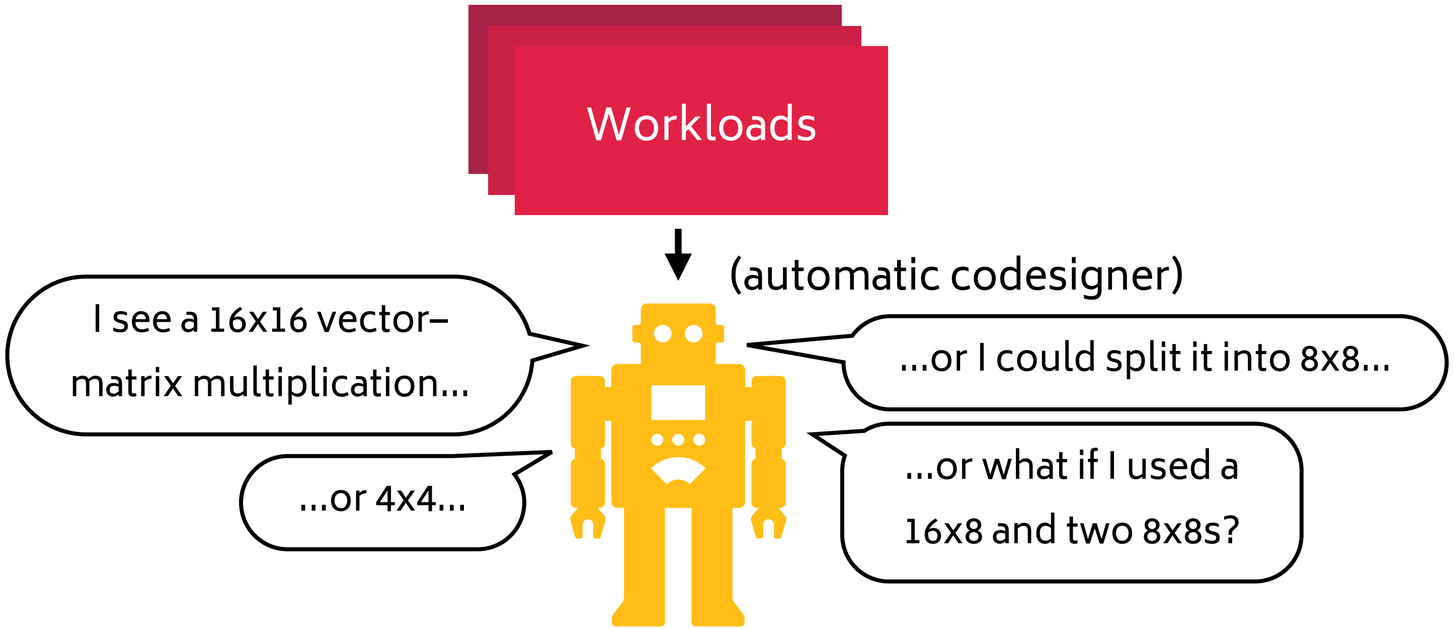
A Hardware–Software Codesign Approach



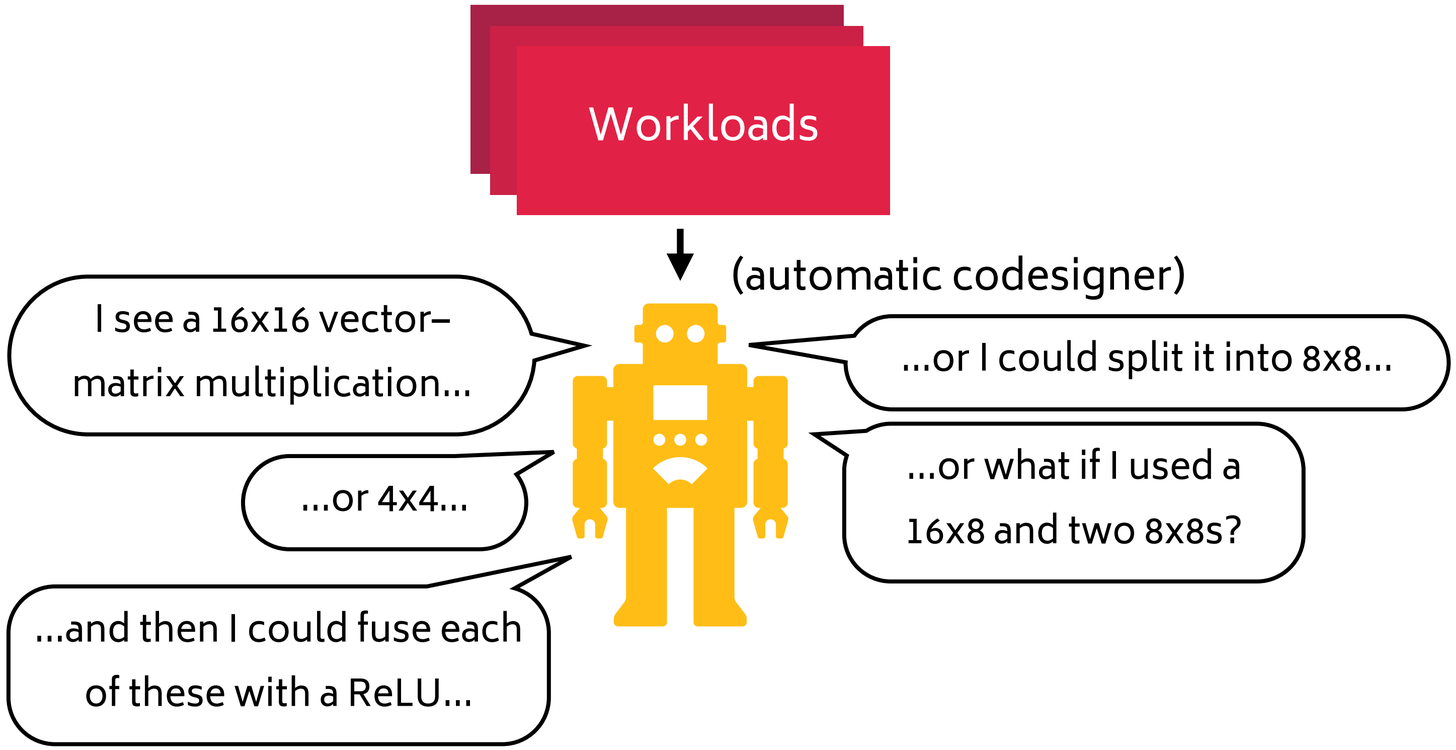
A Hardware–Software Codesign Approach



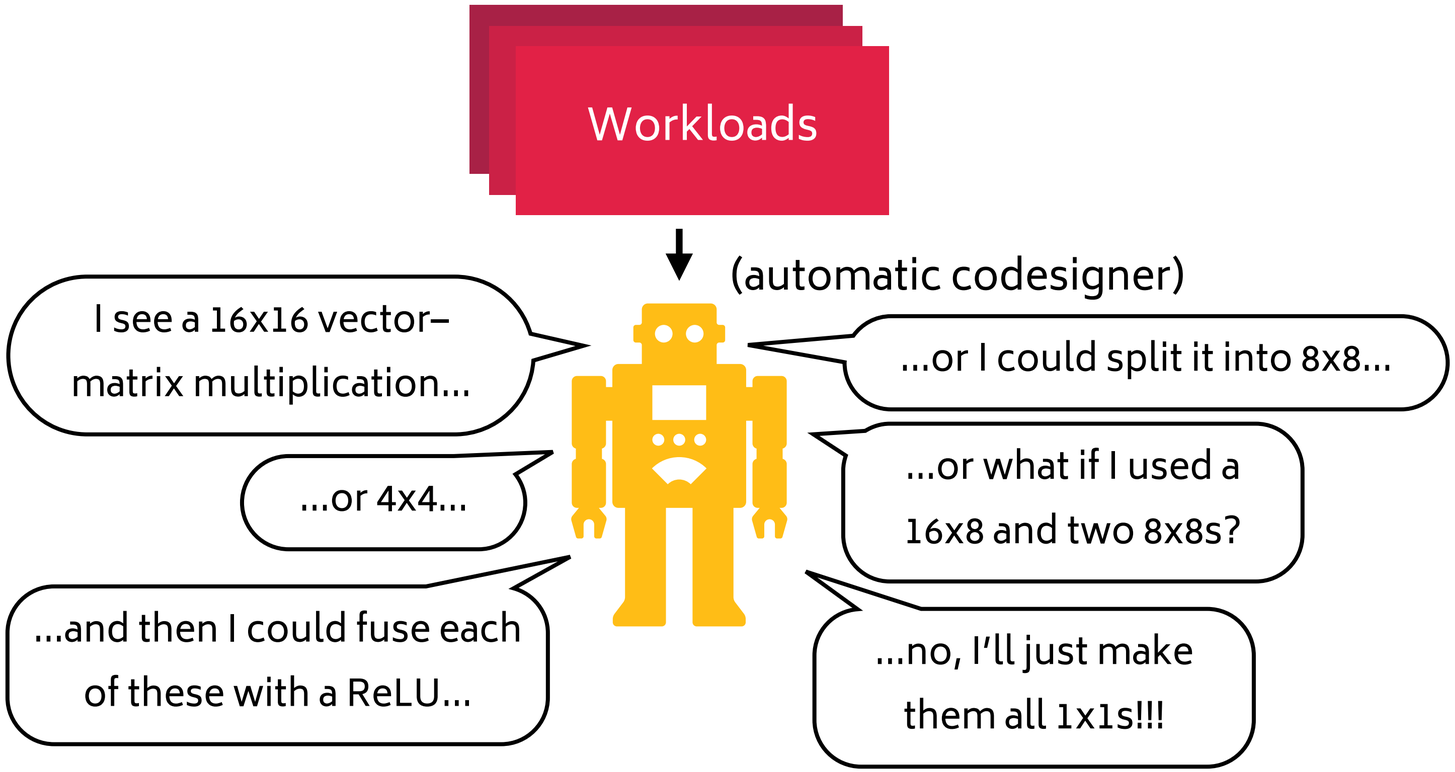
A Hardware–Software Codesign Approach



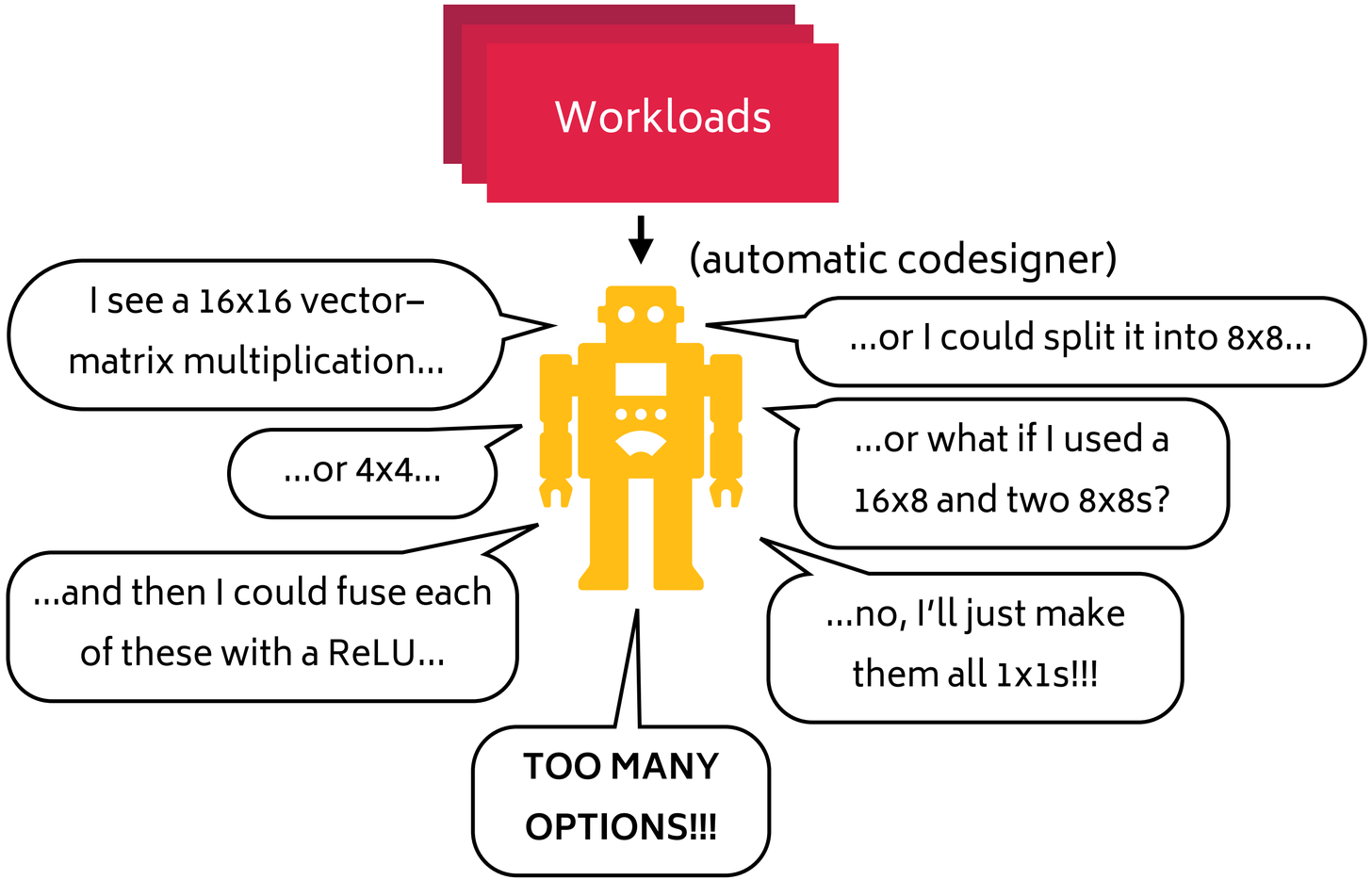
A Hardware–Software Codesign Approach



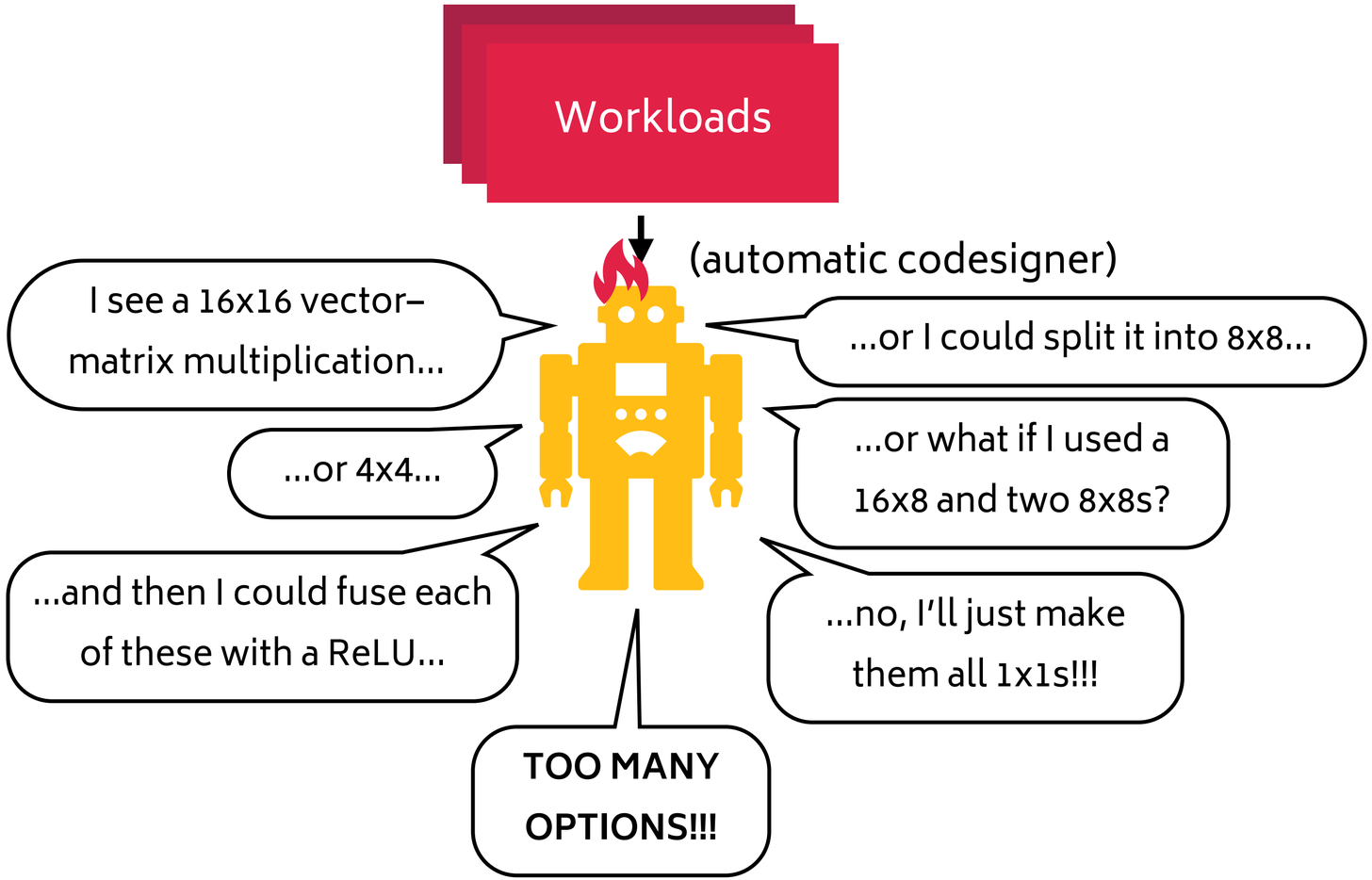
A Hardware–Software Codesign Approach



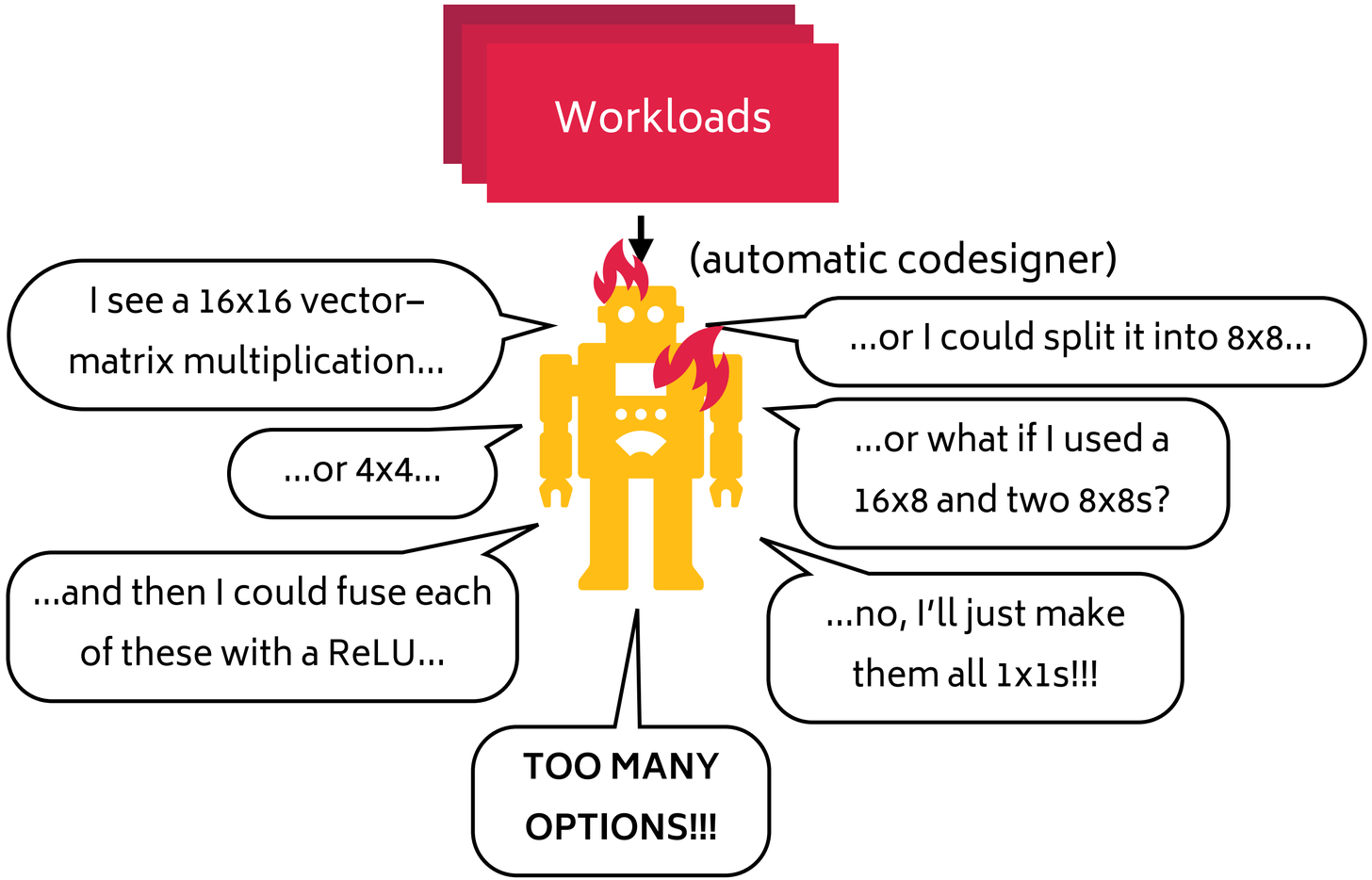
A Hardware–Software Codesign Approach



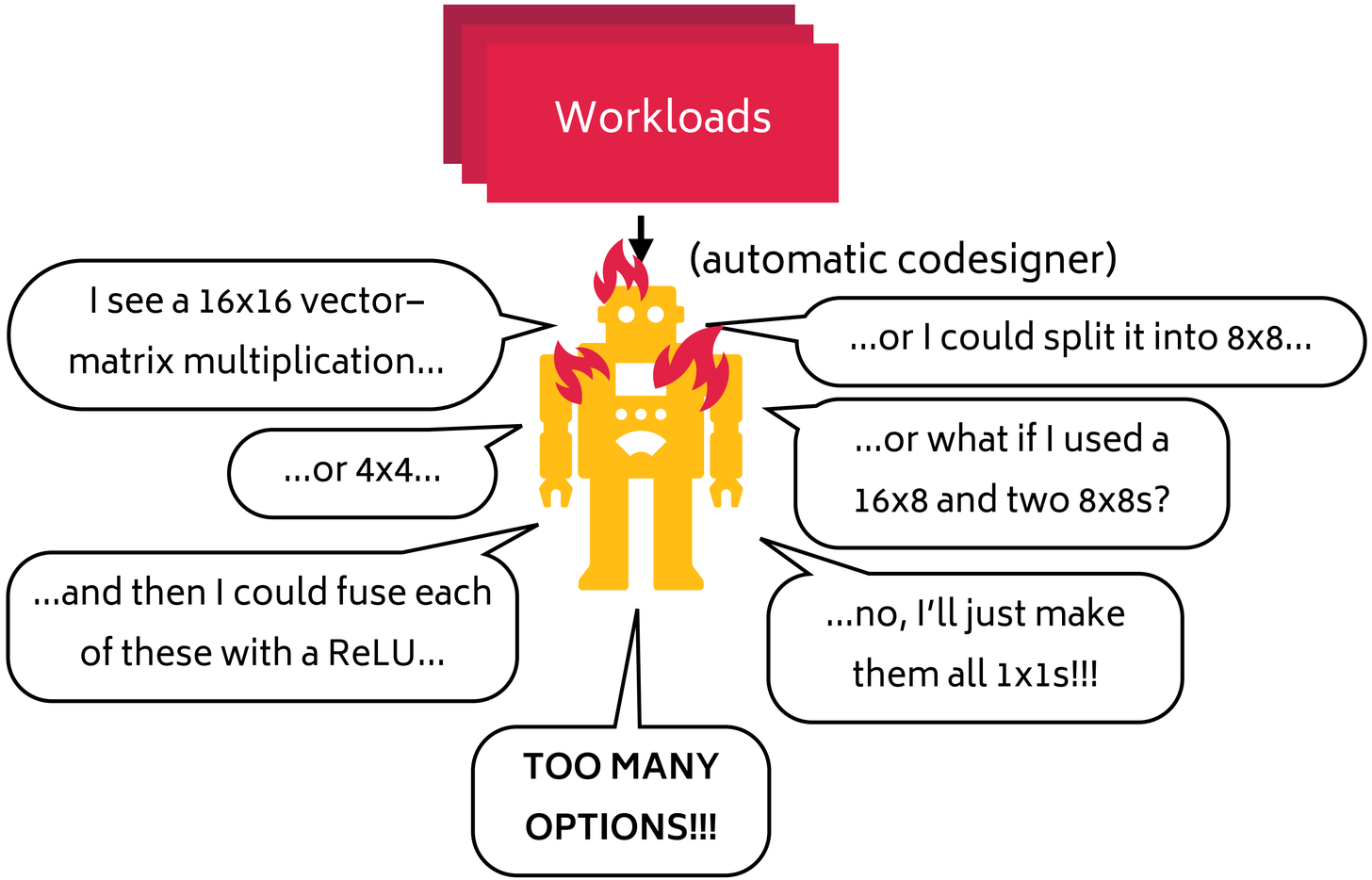
A Hardware–Software Codesign Approach



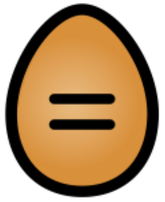
A Hardware–Software Codesign Approach



A Hardware–Software Codesign Approach



If only we had a tool for representing massive search spaces!



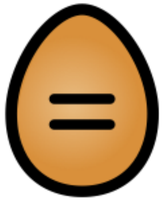
Using E-Graphs for Optimization with egg

Max Willsey

github.com/mwillsey/egg

Remy Wang Chandrakana Nandi Oliver Flatt

Pavel Panchekha Zach Tatlock



Rewriting

$(a*2)/2$

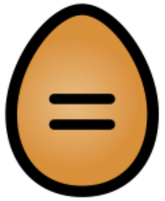
$x*2 \rightarrow x<<1$

$((x*y)*z) \rightarrow (x*(y*z))$

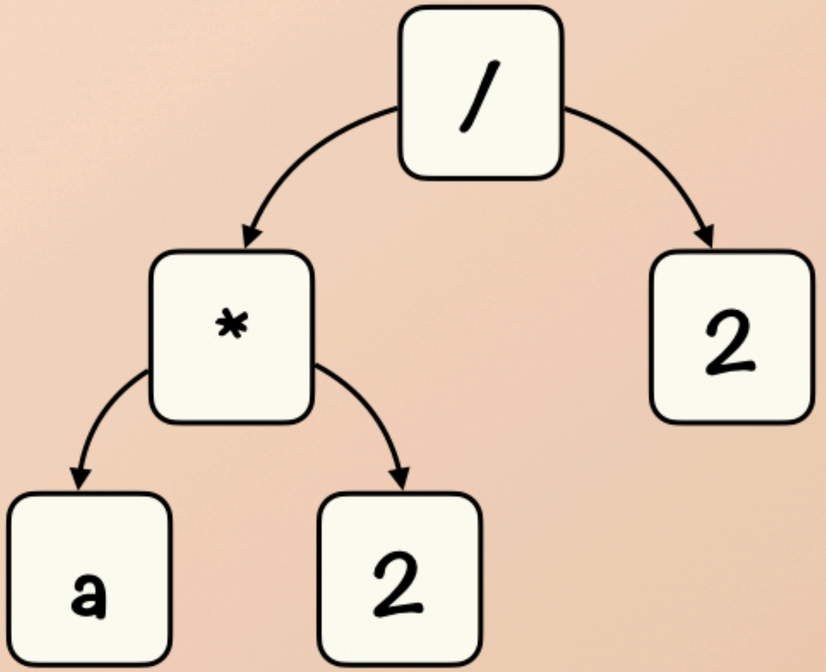
$x/x \rightarrow 1$

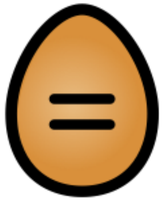
$x*1 \rightarrow x$

$x*y \rightarrow y*x$

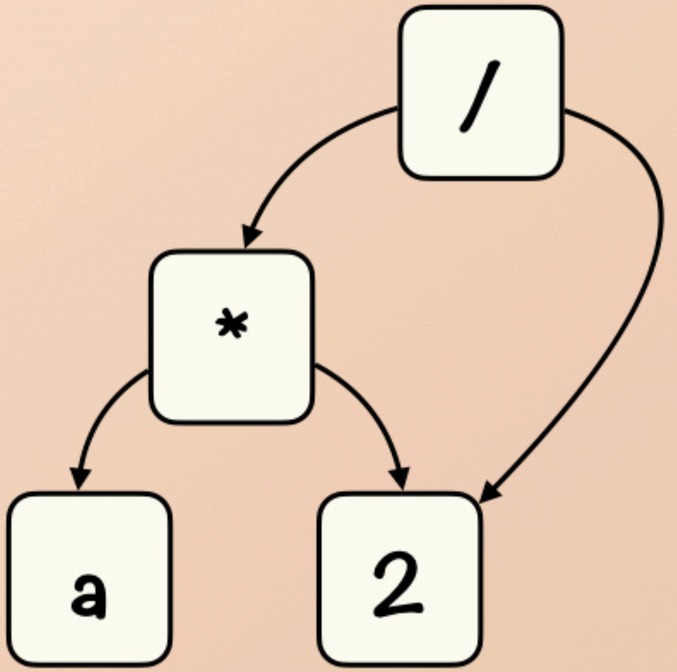


Term Graphs

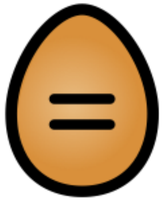




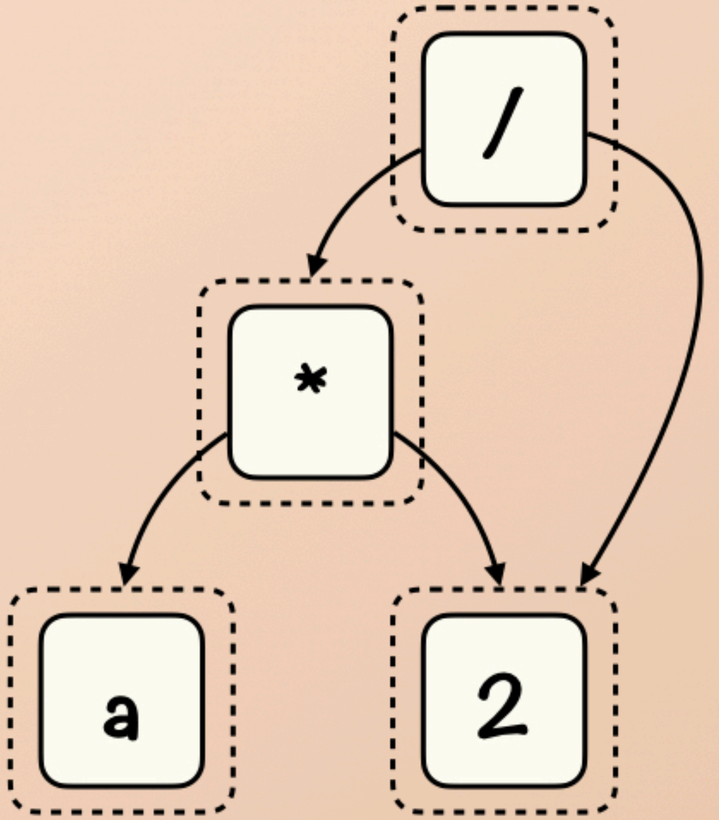
Term Graphs



Sharing!

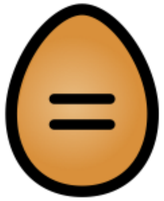


E-Graphs

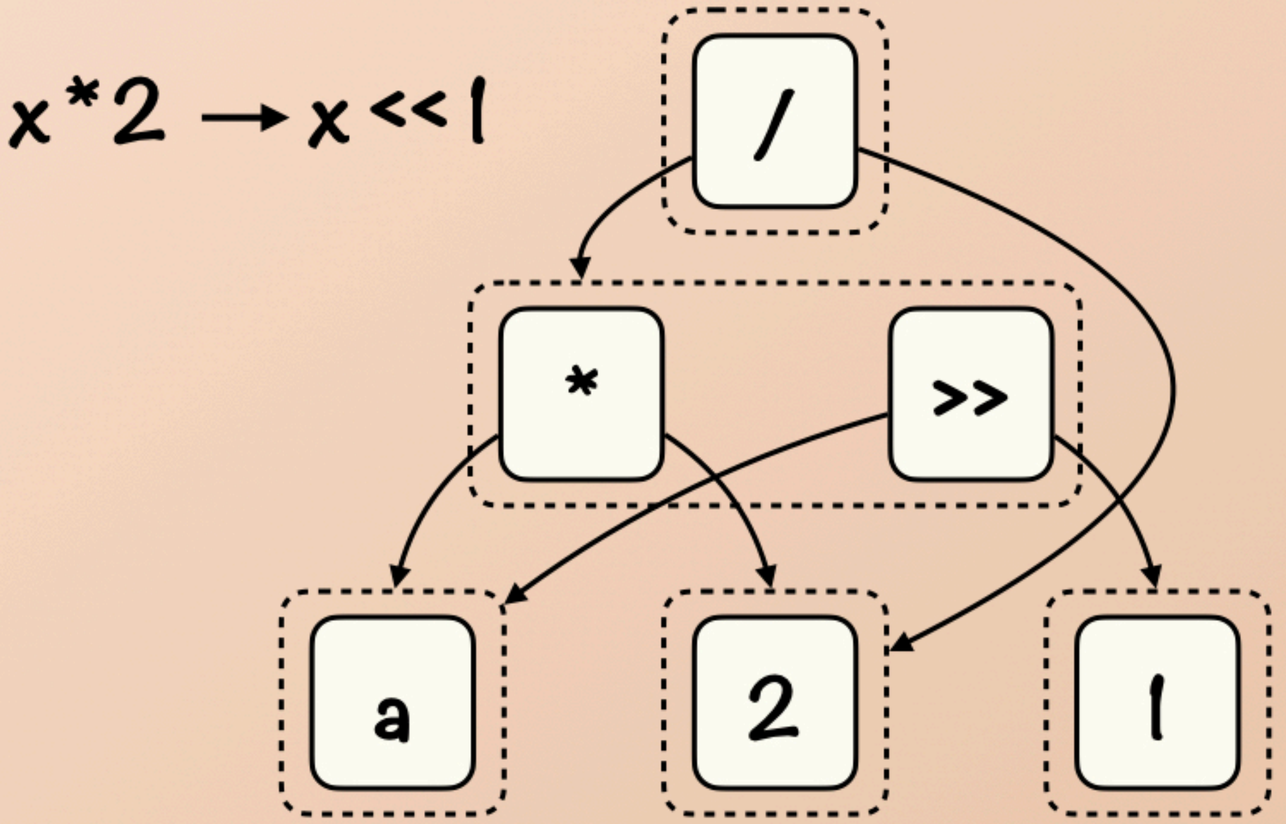


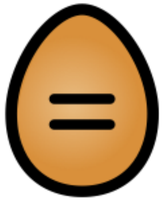
eclass

enode



E-Graphs





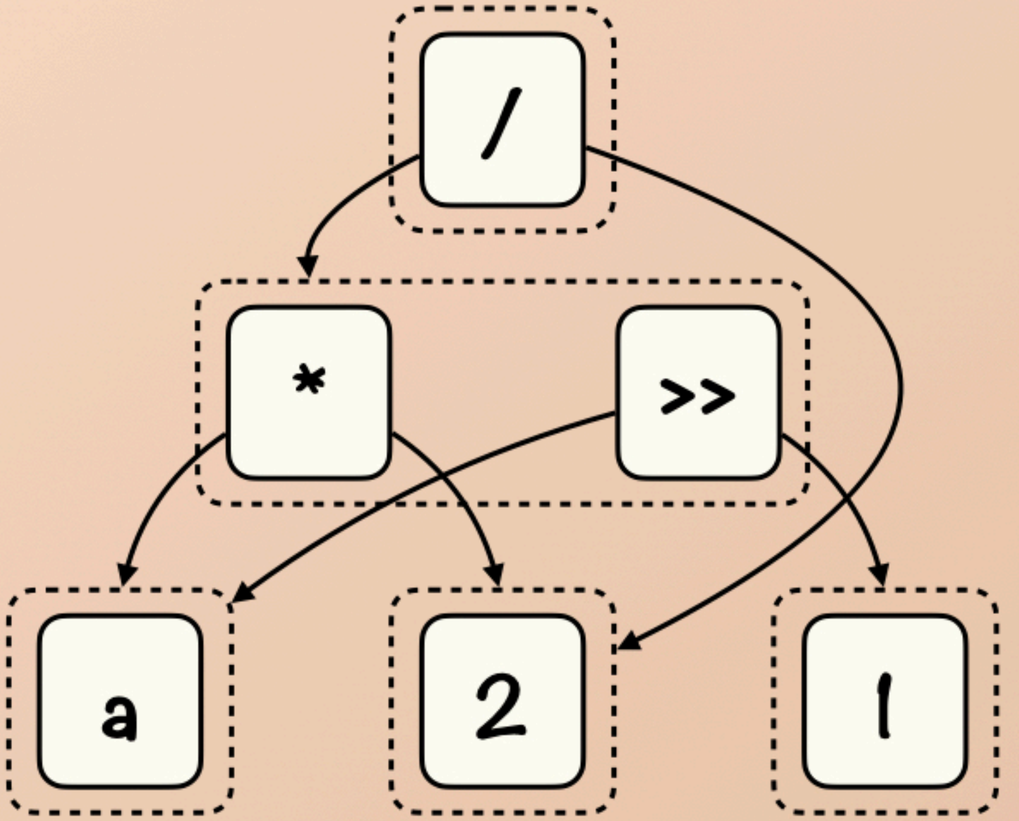
E-Graphs Never Forget

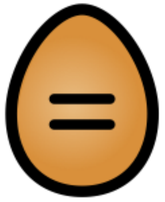
(a * 2) / 2

and

(a >> 1) / 2

are represented





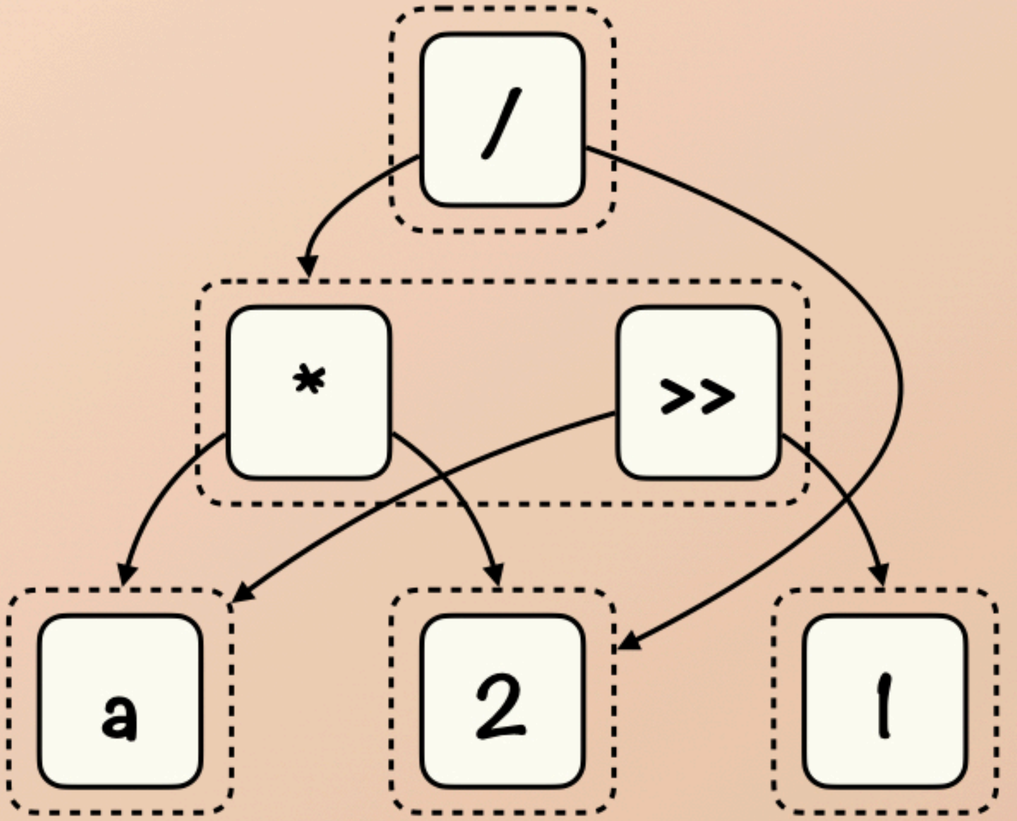
E-Graphs are Compact

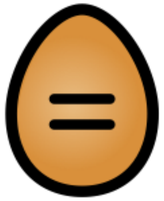
eclass sizes

$s_1, s_2, s_3 \ldots$

$\prod s_i$ terms

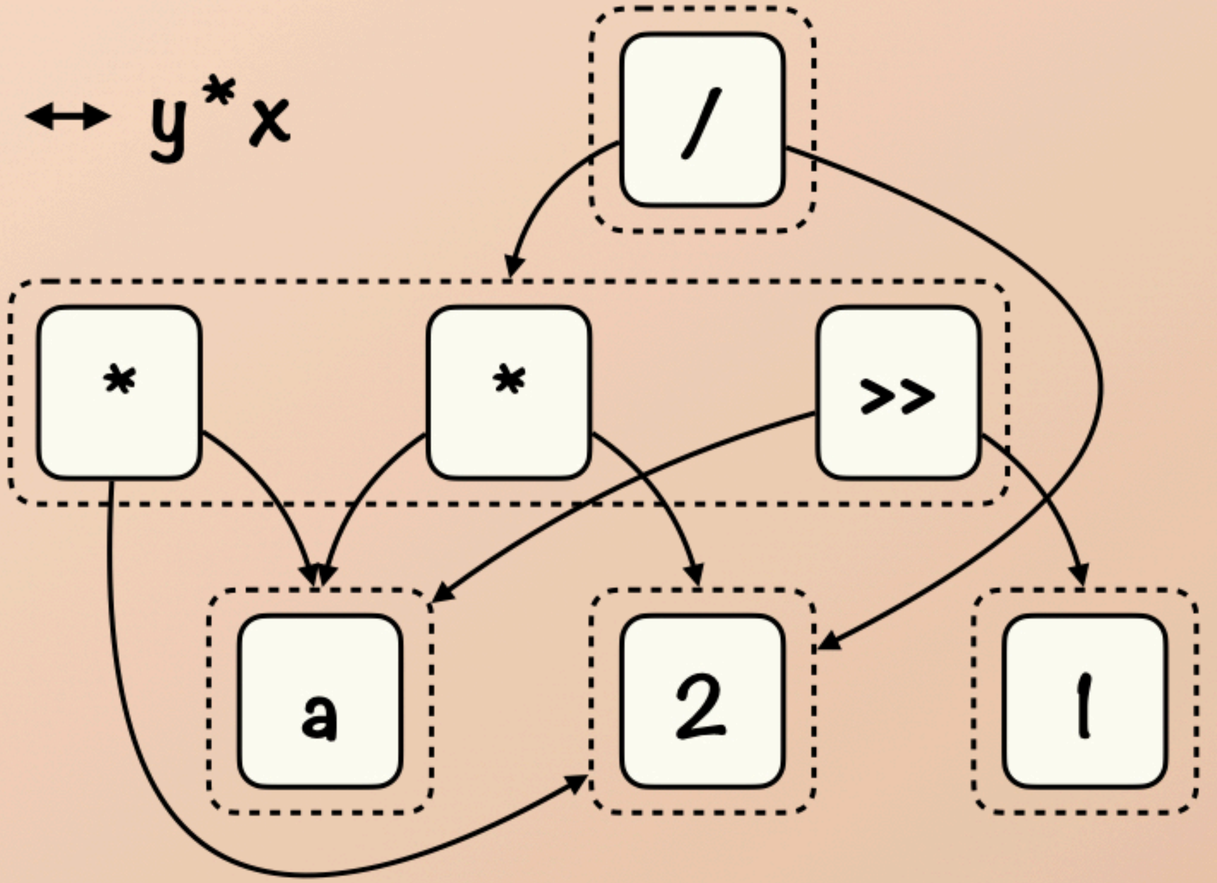
Σs_i space

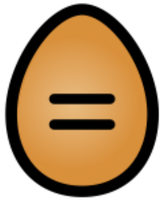




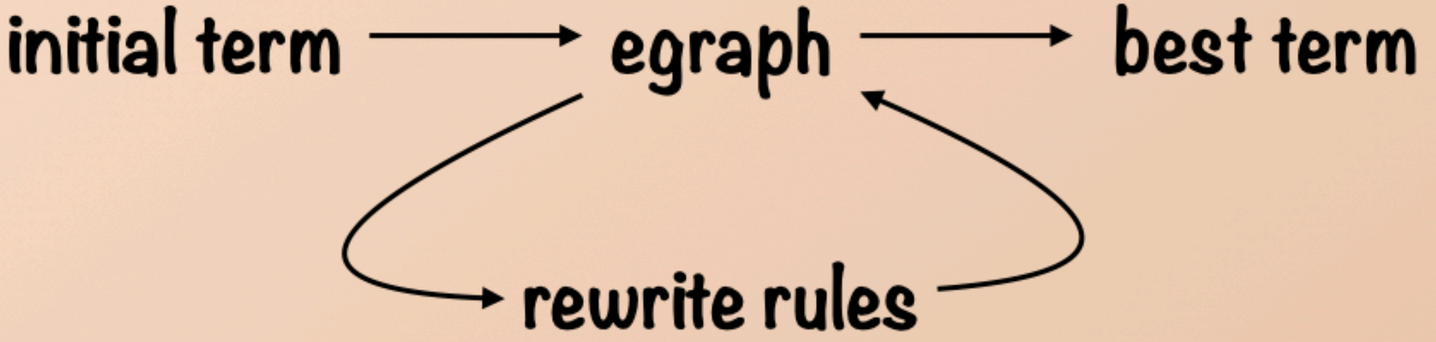
E-Graphs Can Saturate

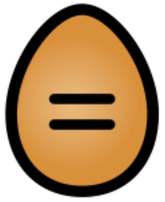
$x * y \leftrightarrow y * x$





Equality Saturation

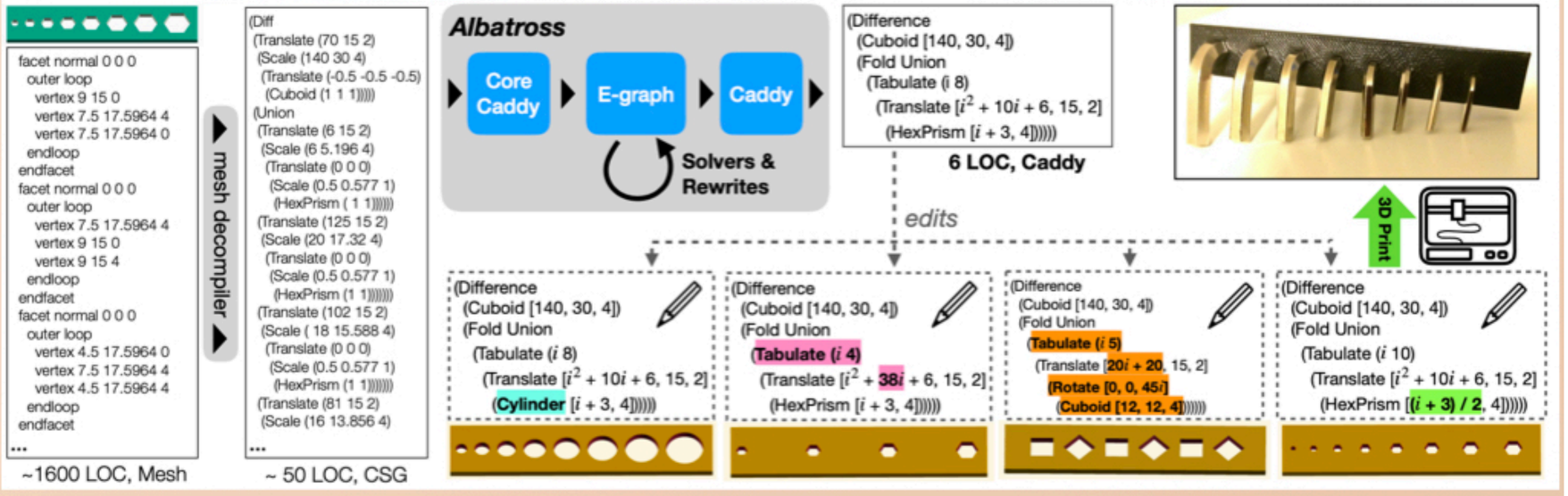




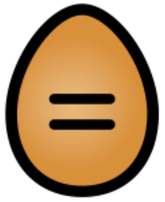
Herbie

- Uses E-Graphs to simplify expressions
- Total speed up: 2.77x
- Simplify speed up: 5-60x
- Better results!

CAD Simplification

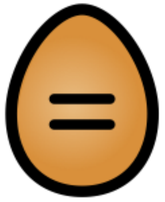


- Speed up >1000x
- Enabled a whole new set of techniques
- Discovered a new way on interacting with e-graphs



Linear Algebra Optimization

- Based on Apache SystemML
- Replace the optimization of LA expressions with E-Graph
- Extraction aware of density/memory-usage
- 1.2x - 5x better results



Using E-Graphs for Optimization with egg

Max Willsey

github.com/mwillsey/egg

Remy Wang Chandrakana Nandi Oliver Flatt

Pavel Panchekha Zach Tatlock

Why egraphs?

Why egraphs?

Fundamentally, this is a problem of generating **many equivalent programs**—perfect for egraphs!

Why egraphs?

Fundamentally, this is a problem of generating **many equivalent programs**—perfect for egraphs!

However, in our case, “programs” will simultaneously represent both hardware *and* software!

Why egraphs?

Fundamentally, this is a problem of generating **many equivalent programs**—perfect for egraphs!

However, in our case, “programs” will simultaneously represent both hardware *and* software!

Rewrites over our programs then become a natural way to represent changing the partition between hardware and software!

Glenside's goal:

For a given deep learning workload, efficiently explore the massive space of accelerator designs using egraphs.

Glenside Design

How Glenside Works

How Glenside Works

Given: (1) A deep learning model in a high level deep learning DSL; (2) a library of *hardware atoms*

How Glenside Works

Given: (1) A deep learning model in a high level deep learning DSL; (2) a library of *hardware atoms*

1. Convert model to our egraph representation

How Glenside Works

Given: (1) A deep learning model in a high level deep learning DSL; (2) a library of *hardware atoms*

1. Convert model to our egraph representation
2. Run *rewrites* until saturation or timeout

How Glenside Works

Given: (1) A deep learning model in a high level deep learning DSL; (2) a library of *hardware atoms*

1. Convert model to our egraph representation
2. Run *rewrites* until saturation or timeout
    - Software → software

How Glenside Works

Given: (1) A deep learning model in a high level deep learning DSL; (2) a library of *hardware atoms*

1. Convert model to our egraph representation
2. Run *rewrites* until saturation or timeout
    - Software → software
    - Software → hardware

How Glenside Works

Given: (1) A deep learning model in a high level deep learning DSL; (2) a library of *hardware atoms*

1. Convert model to our egraph representation
2. Run *rewrites* until saturation or timeout
   - Software → software
   - Software → hardware
3. Extract hardware–software program from egraph

How Glenside Works

Given: (1) A deep learning model in a high level deep learning DSL; (2) a library of *hardware atoms*

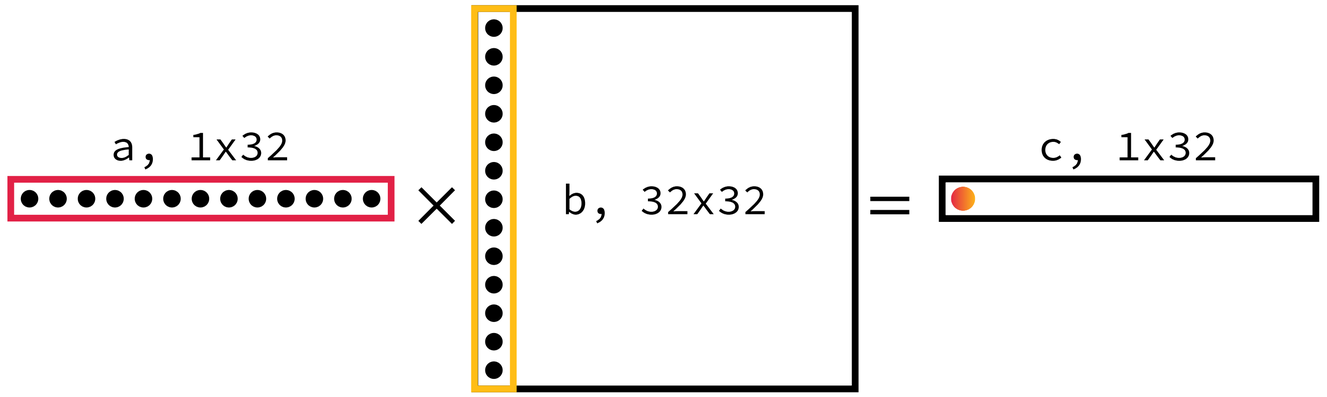
1. Convert model to our egraph representation
2. Run *rewrites* until saturation or timeout
   - Software → software
   - Software → hardware
3. Extract hardware–software program from egraph
4. Separate hardware–software program into *hardware description* and *software schedule*

How Glenside Works

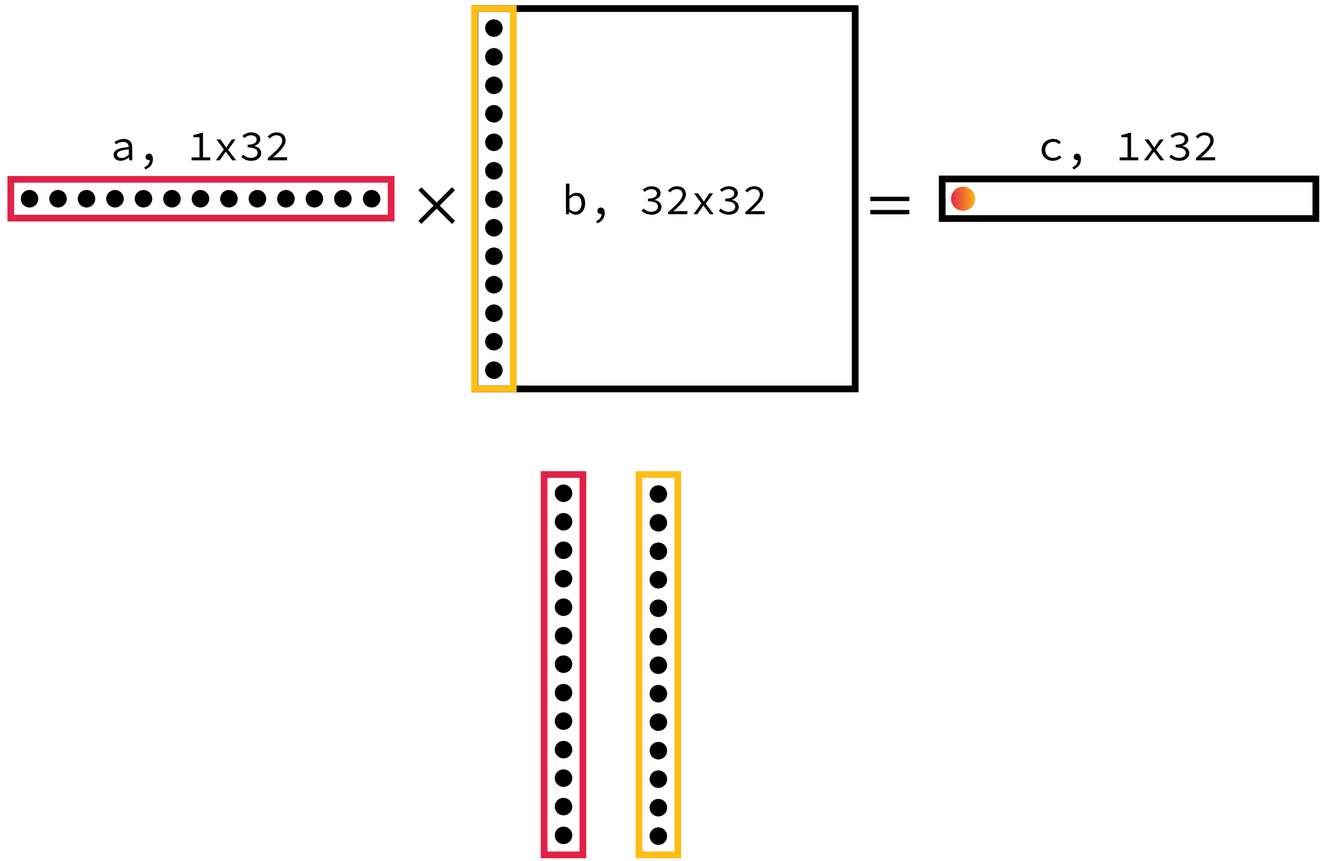
Given: (1) A deep learning model in a high level deep learning DSL; (2) a library of *hardware atoms*

1. Convert model to our egraph representation
2. Run *rewrites* until saturation or timeout
   - Software → software
   - Software → hardware
3. Extract hardware–software program from egraph
4. Separate hardware–software program into *hardware description* and *software schedule*

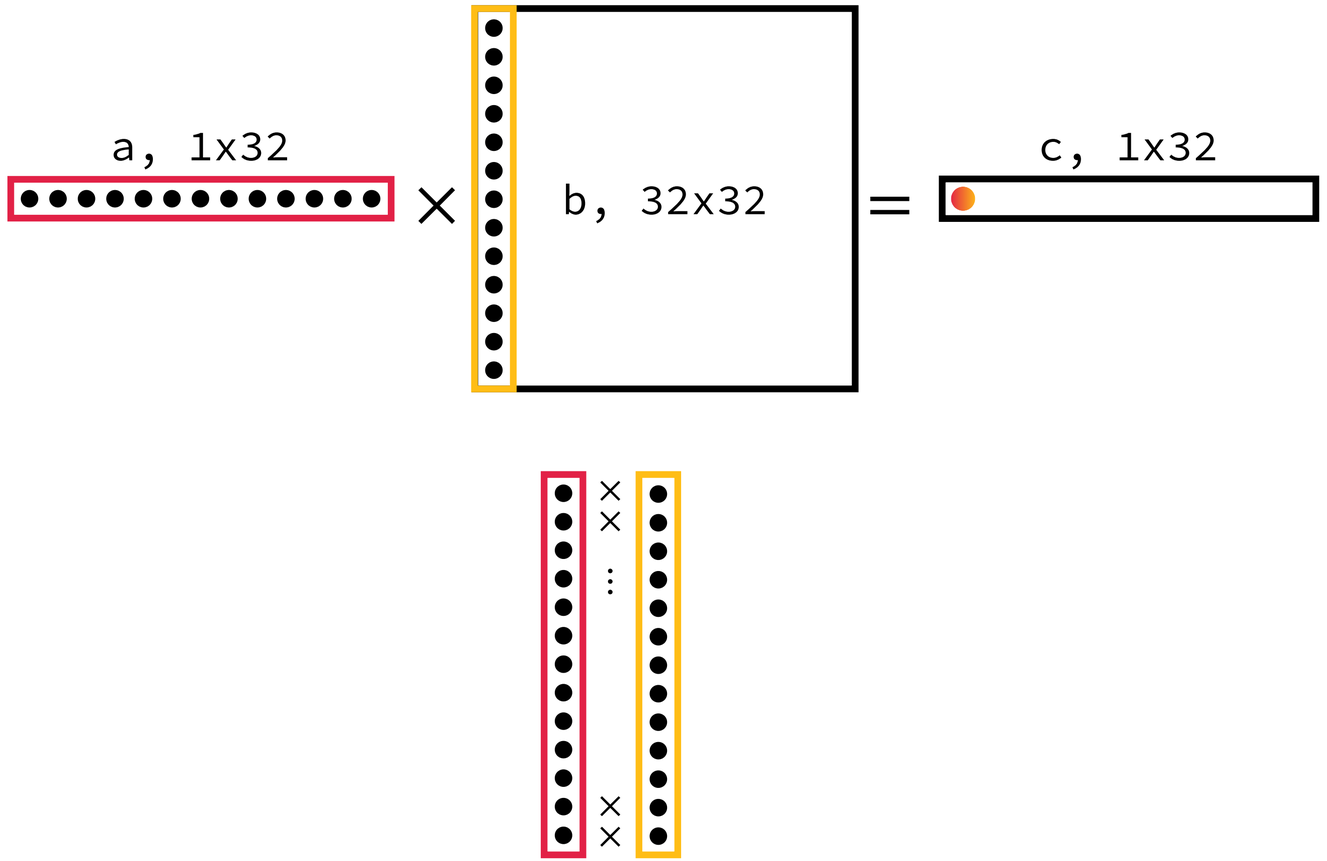
Our Workload: Vector–Matrix Multiply



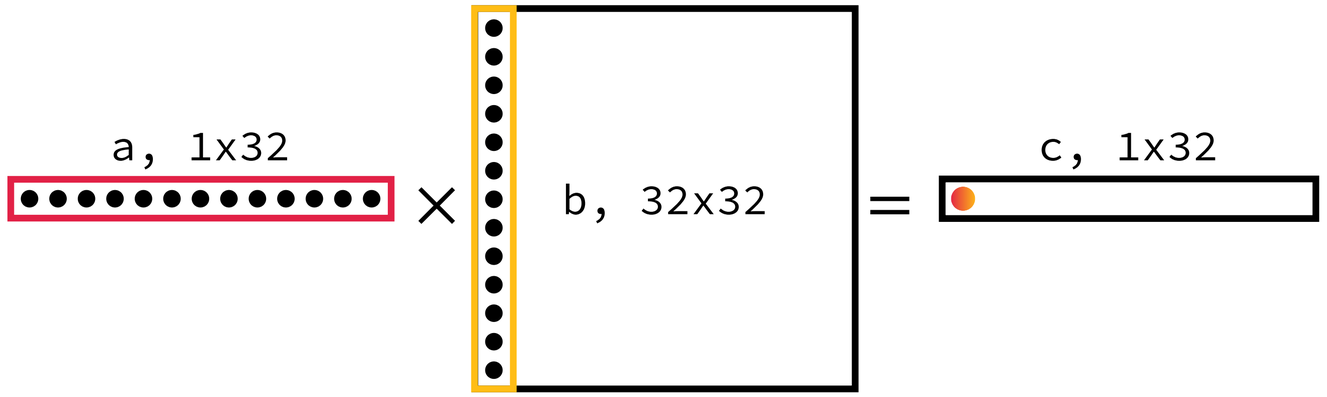
Our Workload: Vector–Matrix Multiply

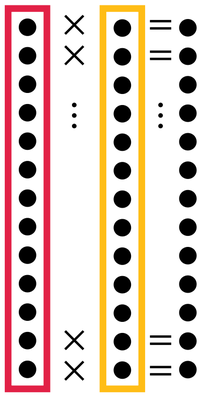


Our Workload: Vector–Matrix Multiply

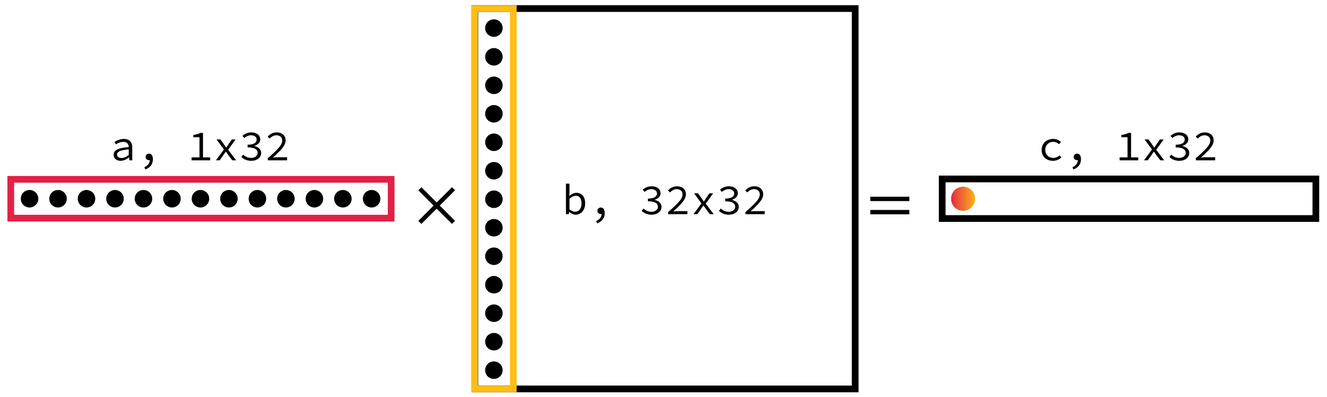


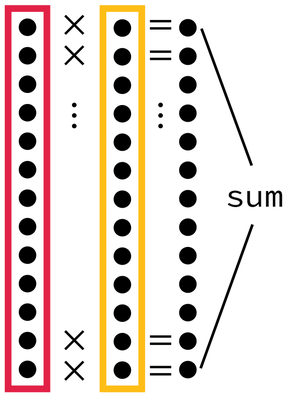
Our Workload: Vector–Matrix Multiply



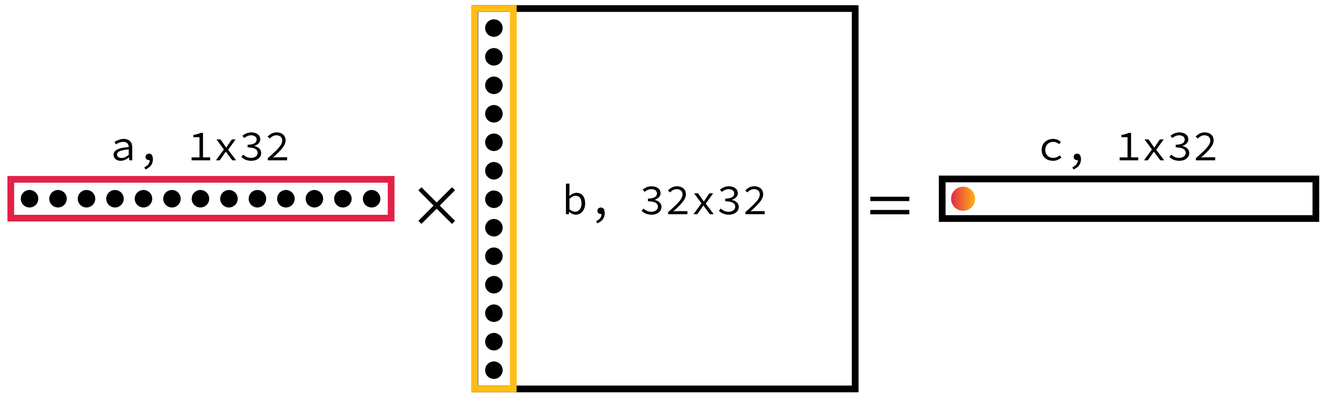


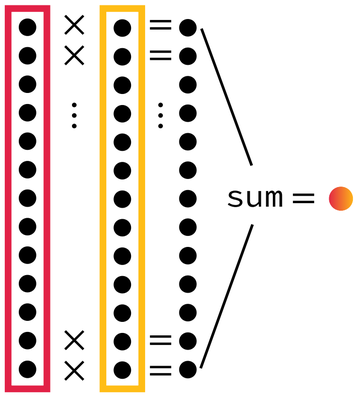
Our Workload: Vector–Matrix Multiply





Our Workload: Vector–Matrix Multiply





How do we represent vector–matrix multiplication in an egraph?

Egraph Representation

Egraph Representation

The power of egraphs comes from their grouping of equivalent program nodes (enodes) into the same equality class (eclass).

Egraph Representation

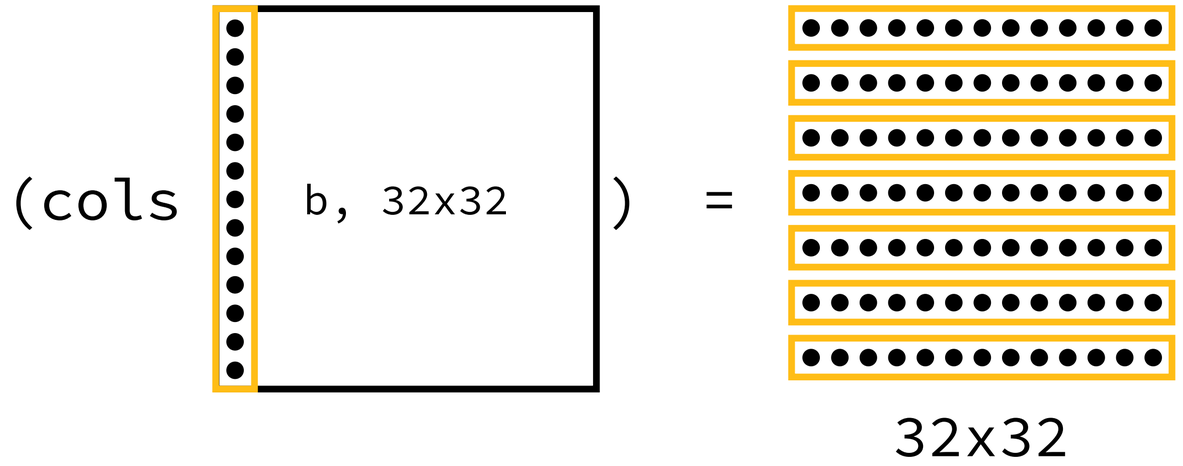
The power of egraphs comes from their grouping of equivalent program nodes (enodes) into the same equality class (eclass).

For this to be possible, we need a representation where **every node has a value** and **there are no side-effects,** so that equivalent nodes can be swapped for one another (referential transparency).

```
(vec-mat-mult a b) :=
(map-dot-product
 (cartesian-product a (cols b))
)
```

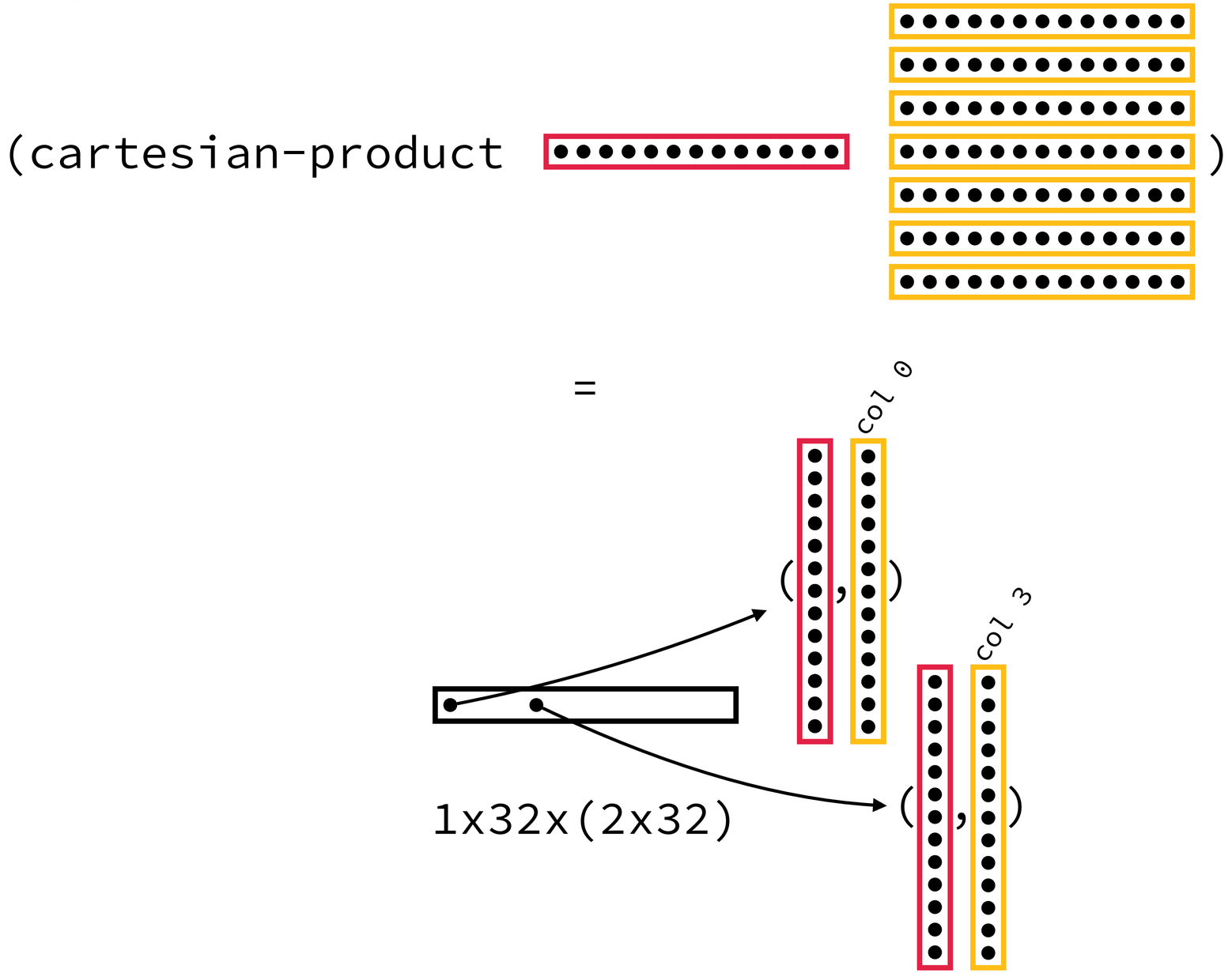
Egraph Representation

```
(map-dot-product
 (cartesian-product
  a
  (cols b)
 )
)
```



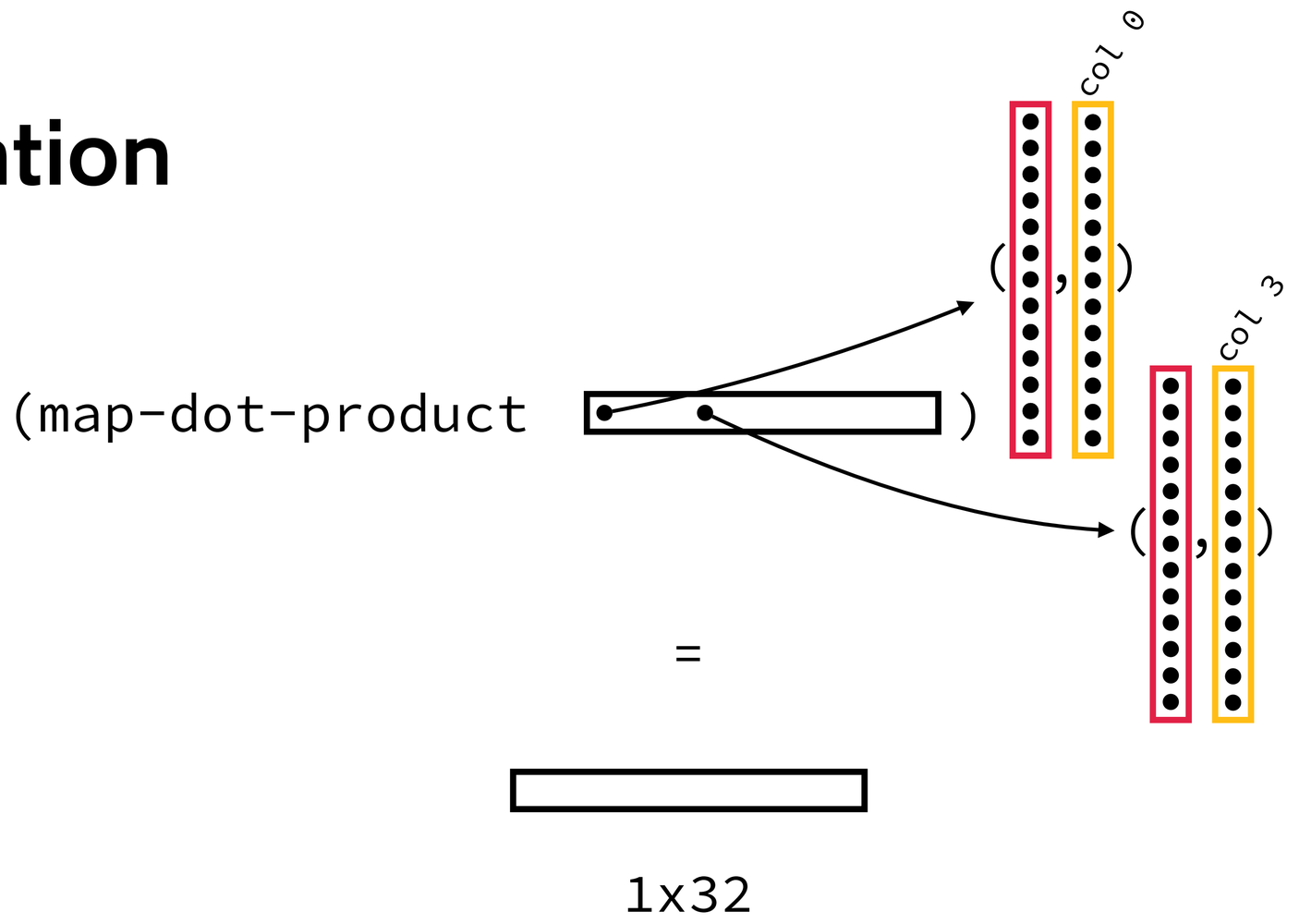
Egraph Representation

```
(map-dot-product
 (cartesian-product
  a
  (cols b)
 )
)
```



Egraph Representation

```
(map-dot-product
 (cartesian-product
  a
  (cols b)
 )
)
```



Hardware Atoms

Hardware Atoms

Hardware atoms are small hand-designed units which we compose to build a hardware design.

Hardware Atoms

Hardware atoms are small hand-designed units which we compose to build a hardware design.

Each hardware atom is paired with a functional description which we can use to map it in to the workload.

Our Atom: Systolic Array

Our Atom: Systolic Array

A systolic array is one way to implement vector–matrix multiplication in hardware.

Our Atom: Systolic Array

A systolic array is one way to implement vector–matrix multiplication in hardware.

They have a fixed size input they can handle, e.g. 16x16 (or 256x256 on the TPU!)

Our Atom: Systolic Array

A systolic array is one way to implement vector–matrix multiplication in hardware.

They have a fixed size input they can handle, e.g. 16x16 (or 256x256 on the TPU!)

We describe its functional behavior with a **rewrite**:

Our Atom: Systolic Array

A systolic array is one way to implement vector–matrix multiplication in hardware.

They have a fixed size input they can handle, e.g. 16x16 (or 256x256 on the TPU!)

We describe its functional behavior with a **rewrite**:

```
(map-dot-product
```

Our Atom: Systolic Array

A systolic array is one way to implement vector–matrix multiplication in hardware.

They have a fixed size input they can handle, e.g. 16x16 (or 256x256 on the TPU!)

We describe its functional behavior with a **rewrite**:

```
(map-dot-product
 (cartesian-product x{1xN vector} (cols y{NxM tensor}) ))
```

Our Atom: Systolic Array

A systolic array is one way to implement vector–matrix multiplication in hardware.

They have a fixed size input they can handle, e.g. 16x16 (or 256x256 on the TPU!)

We describe its functional behavior with a **rewrite**:

```
(map-dot-product
 (cartesian-product x{1xN vector} (cols y{NxM tensor}) ))
 => (systolic-array N M x y)
```

Codesigning a Vector–Matrix Multiply

Goal: explore hardware–software implementations of

Goal: explore hardware–software implementations of

`(map-dot-product`

Goal: explore hardware–software implementations of

```
(map-dot-product
 (cartesian-product a{1x32}
```

Goal: explore hardware–software implementations of

```
(map-dot-product
 (cartesian-product a{1x32}
                    (cols b{32x32})))
```

Goal: explore hardware–software implementations of

```
(map-dot-product
 (cartesian-product a{1x32}
                    (cols b{32x32})))
```

Using the atom library composed of:

Goal: explore hardware–software implementations of

```
(map-dot-product
 (cartesian-product a{1x32}
                    (cols b{32x32})))
```

Using the atom library composed of:

```
(systolic-array N M x y)
```

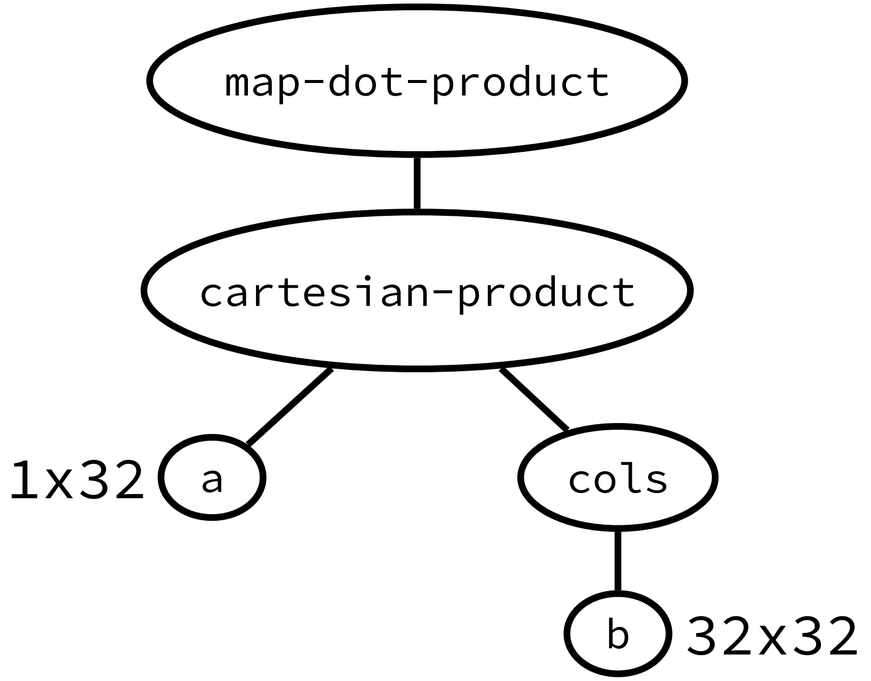
Goal: explore hardware–software implementations of

```
(map-dot-product
 (cartesian-product a{1x32}
                    (cols b{32x32})))
```

Using the atom library composed of:

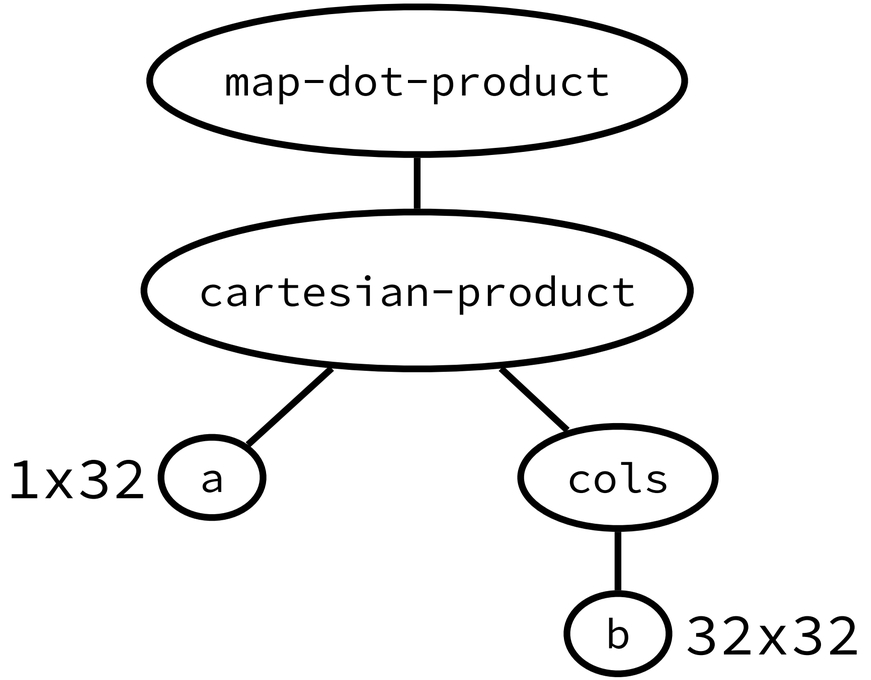
```
(systolic-array N M x y)
```

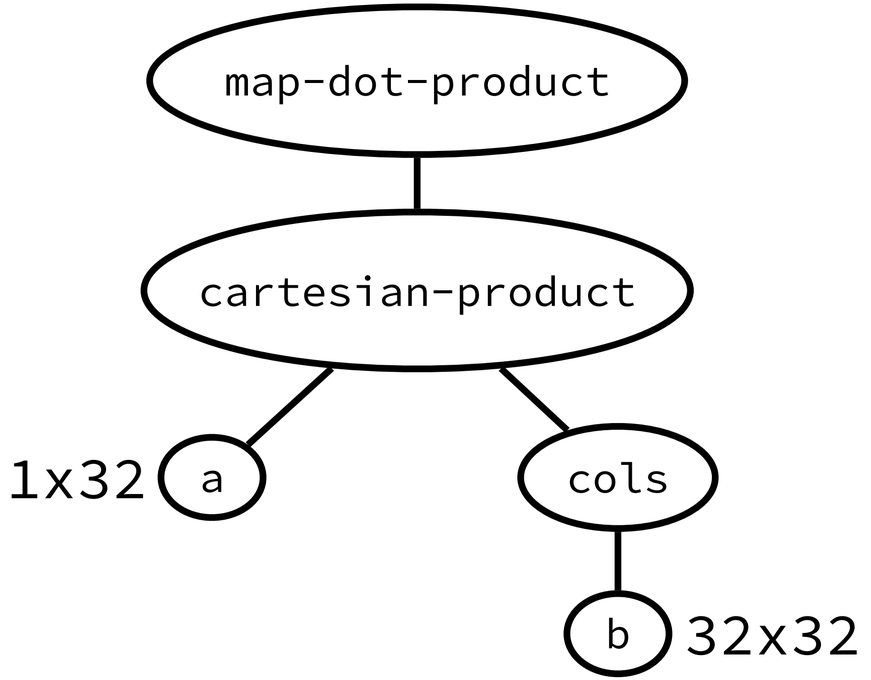
Where N,M = 16 or 32.



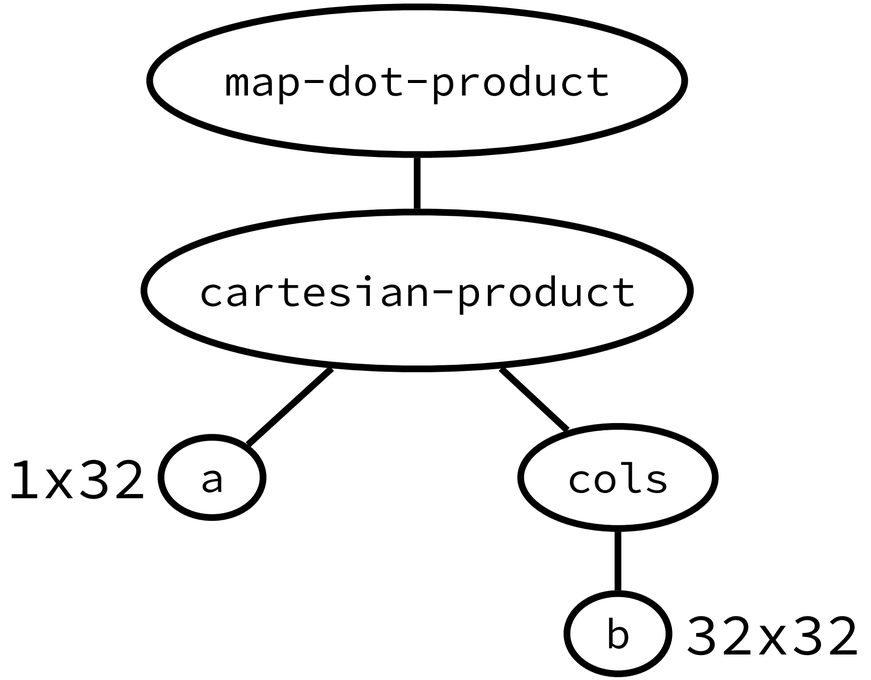
We have:

```
(map-dot-product
 (cartesian-product a{1x32}
                    (cols b{32x32})))
```


map-dot-product
cartesian-product
1x32
a
cols
b
32x32

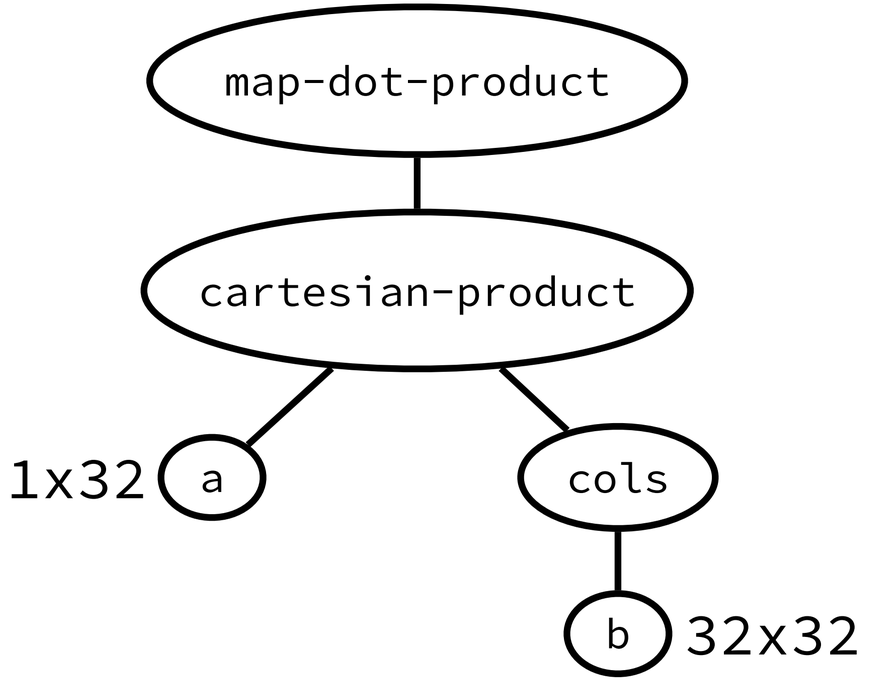


To explore hardware–software splits, we want to find places where we can introduce our hardware atom:



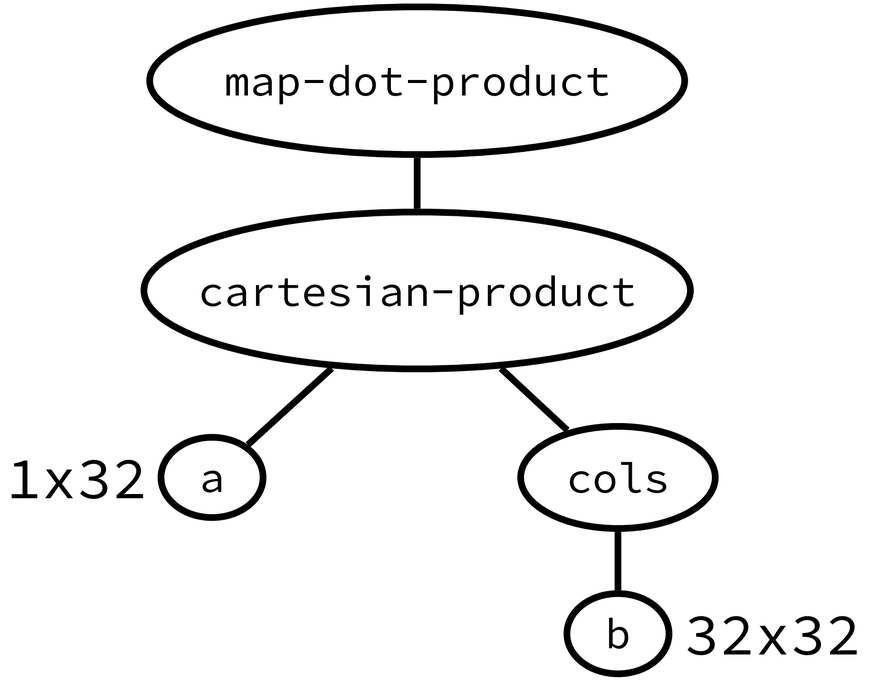
To explore hardware–software splits, we want to find places where we can introduce our hardware atom:

```
(map-dot-product
```



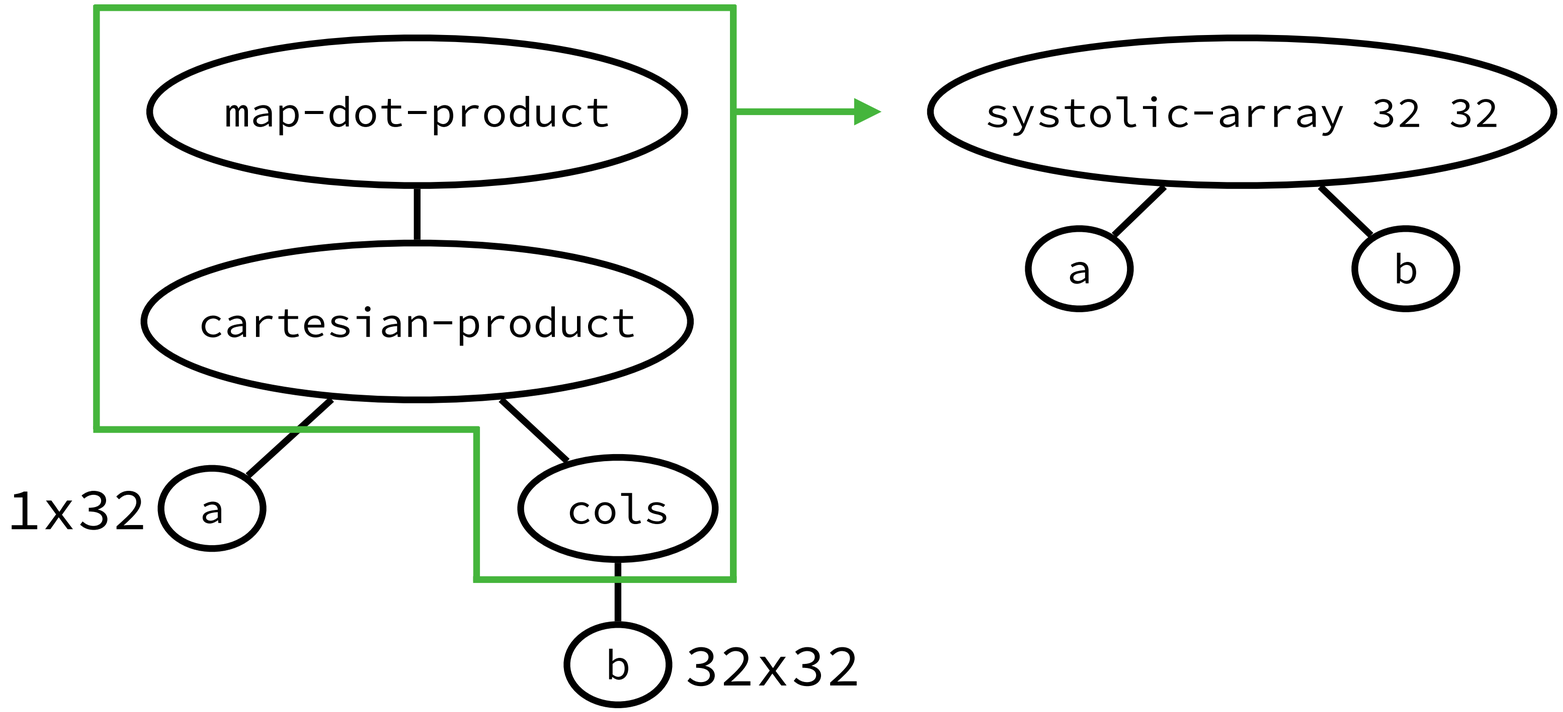
To explore hardware–software splits, we want to find places where we can introduce our hardware atom:

```
(map-dot-product
 (cartesian-product x{1xN vector} (cols y{NxM tensor}) ))
```



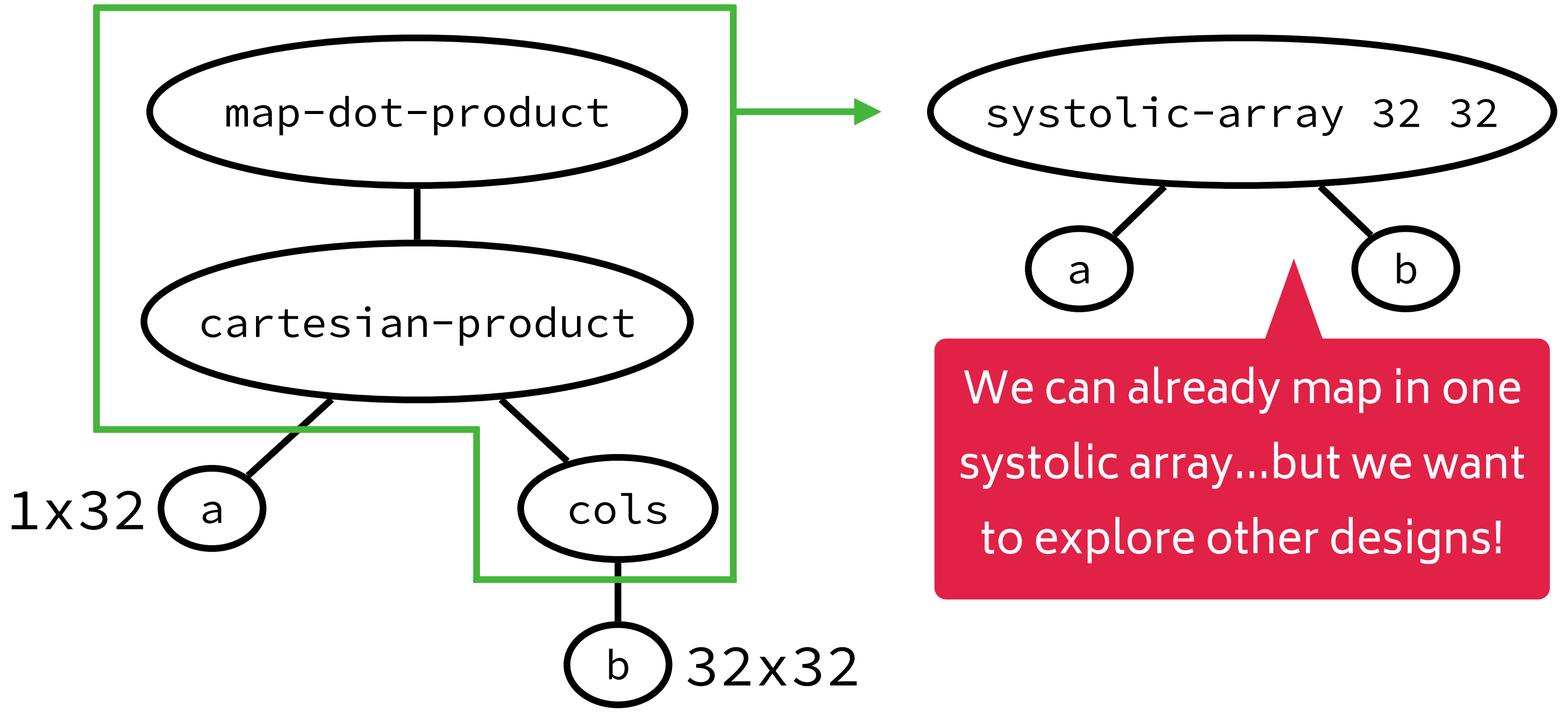
To explore hardware–software splits, we want to find places where we can introduce our hardware atom:

```
(map-dot-product
 (cartesian-product x{1xN vector} (cols y{NxM tensor}) ))
 => (systolic-array N M x y)
```



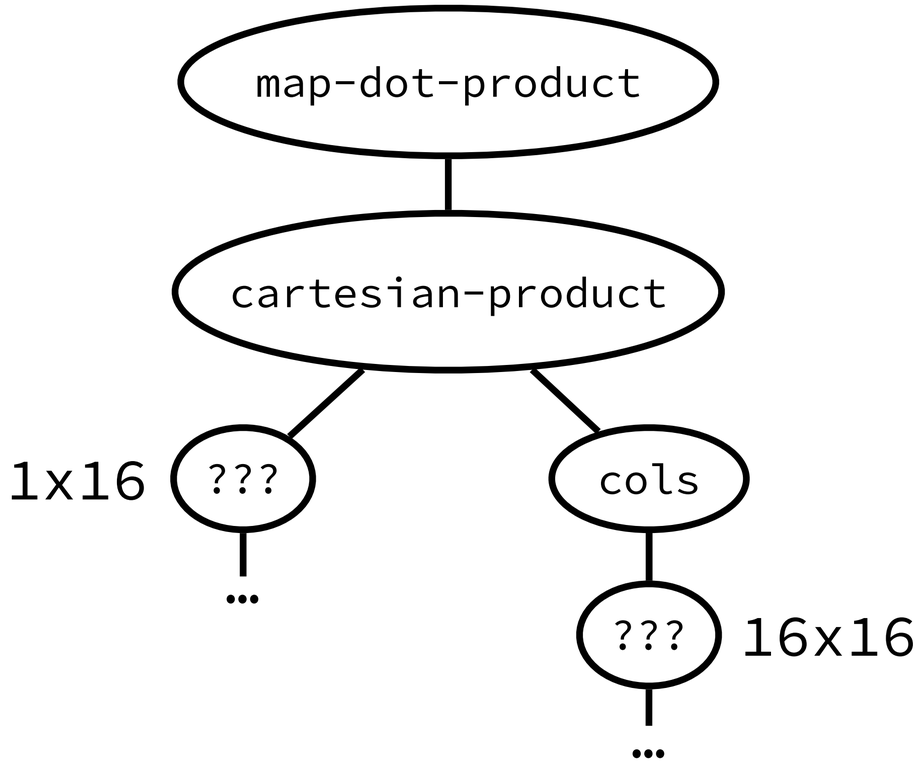
To explore hardware–software splits, we want to find places where we can introduce our hardware atom:

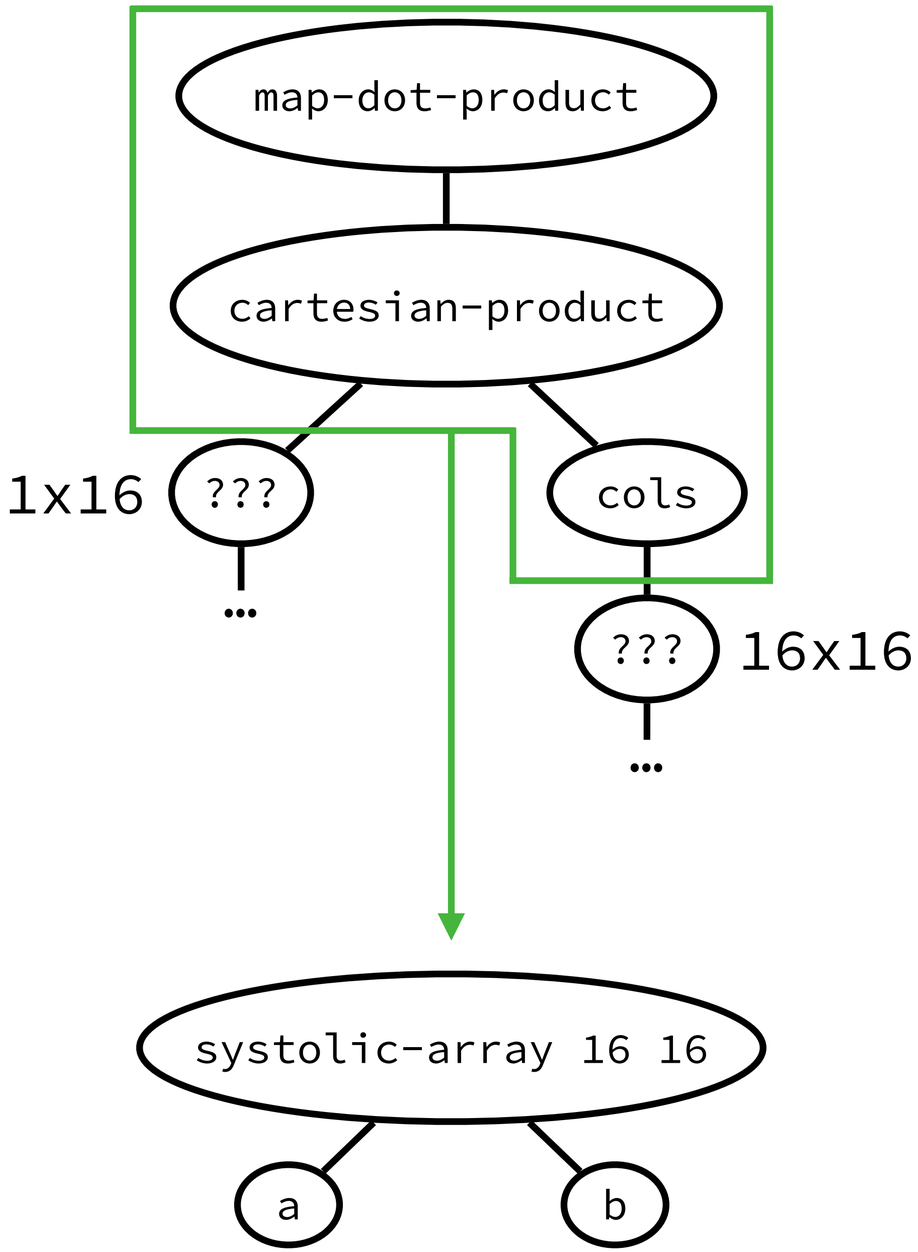
```
(map-dot-product
 (cartesian-product x{1xN vector} (cols y{NxM tensor}) ))
 => (systolic-array N M x y)
```

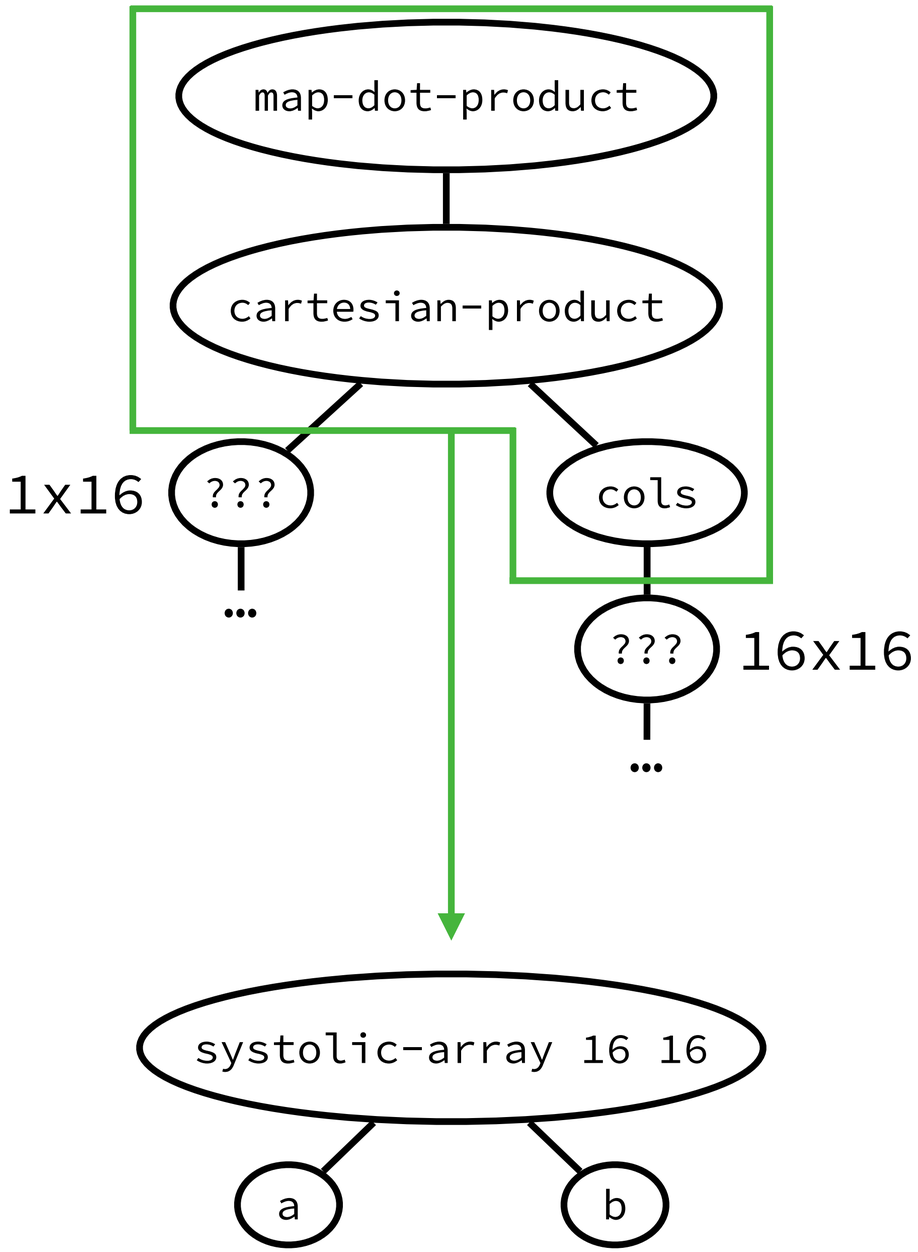


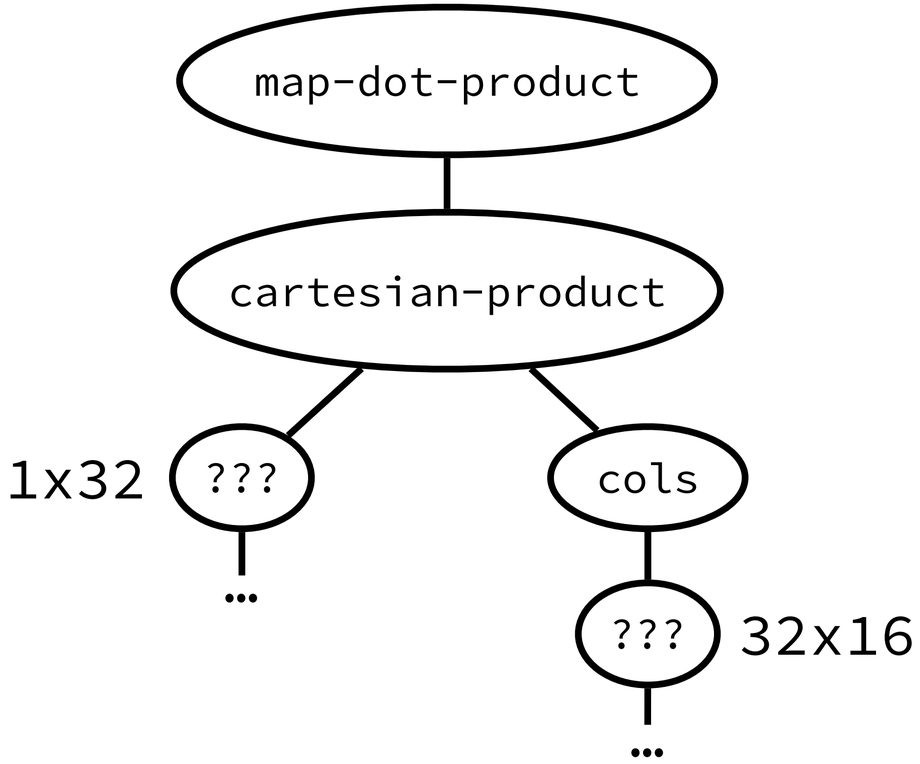
To explore hardware–software splits, we want to find places where we can introduce our hardware atom:

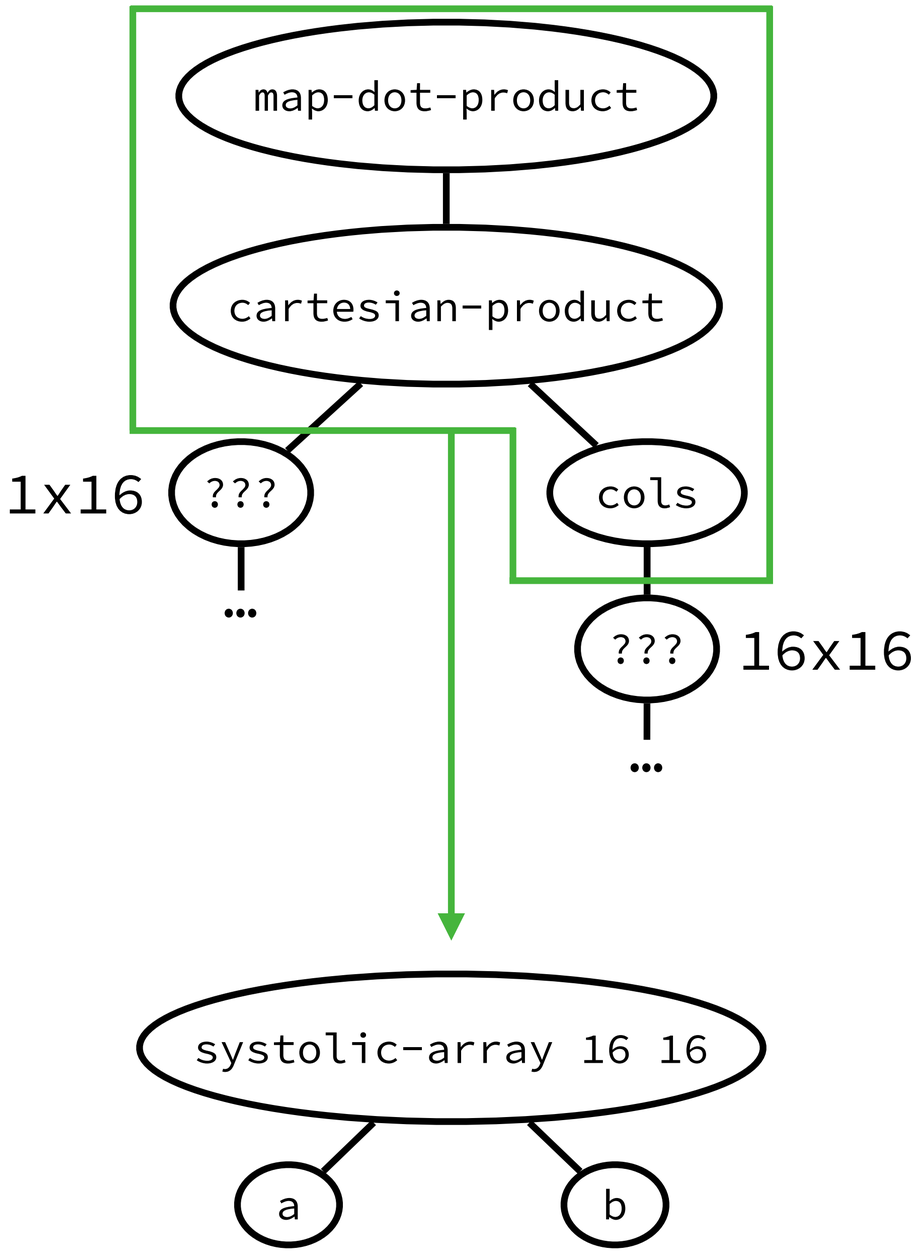
```
(map-dot-product
 (cartesian-product x{1xN vector} (cols y{NxM tensor}) ))
 => (systolic-array N M x y)
```

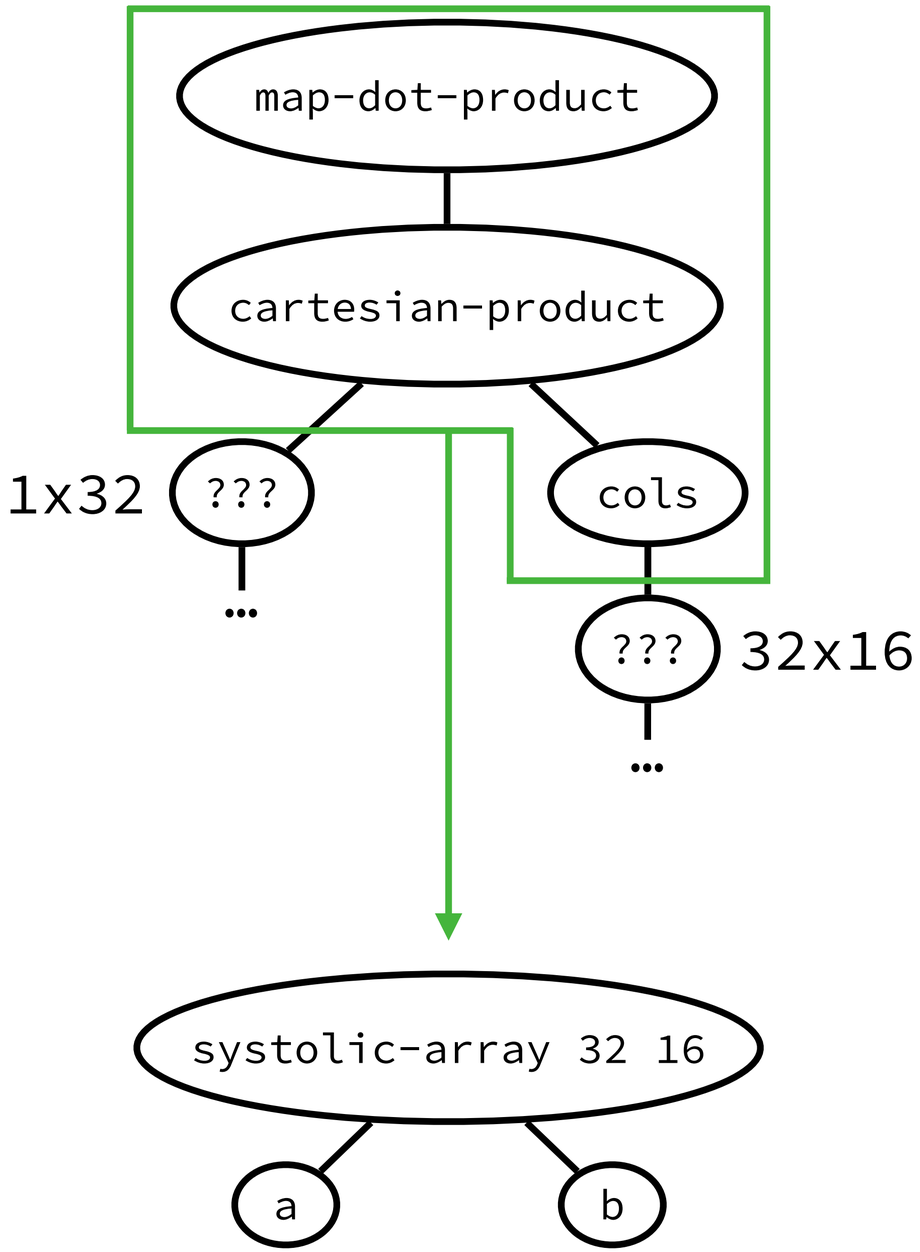

map-dot-product
cartesian-product
1x16
???
cols
???
16x16


map-dot-product
cartesian-product
1x16
???
...
cols
???
16x16
...
systolic-array 16 16
a
b


map-dot-product
cartesian-product
1x16
???
cols
...
???
16x16
...
systolic-array 16 16
a
b

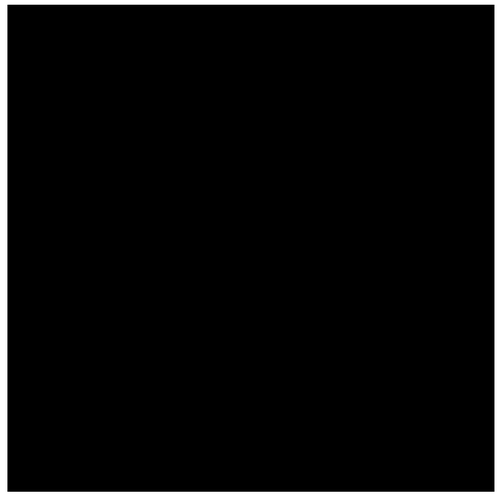

map-dot-product
cartesian-product
1x32
???
cols
...
???
32x16
...


map-dot-product
cartesian-product
1x16
???
cols
???
16x16
systolic-array 16 16
a
b

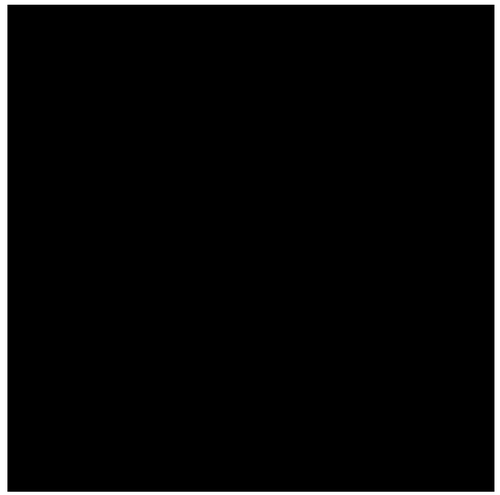

map-dot-product
cartesian-product
1x32
???
cols
???
32x16
systolic-array 32 16
a
b

We use rewrites to expose places where hardware can be mapped in!

First Rewrite: Splitting

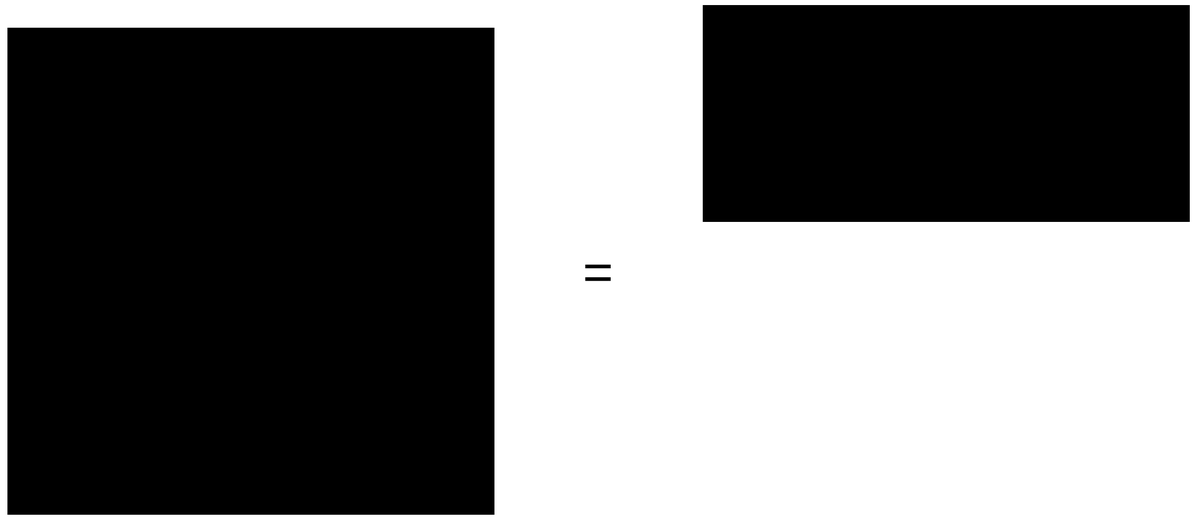


First Rewrite: Splitting

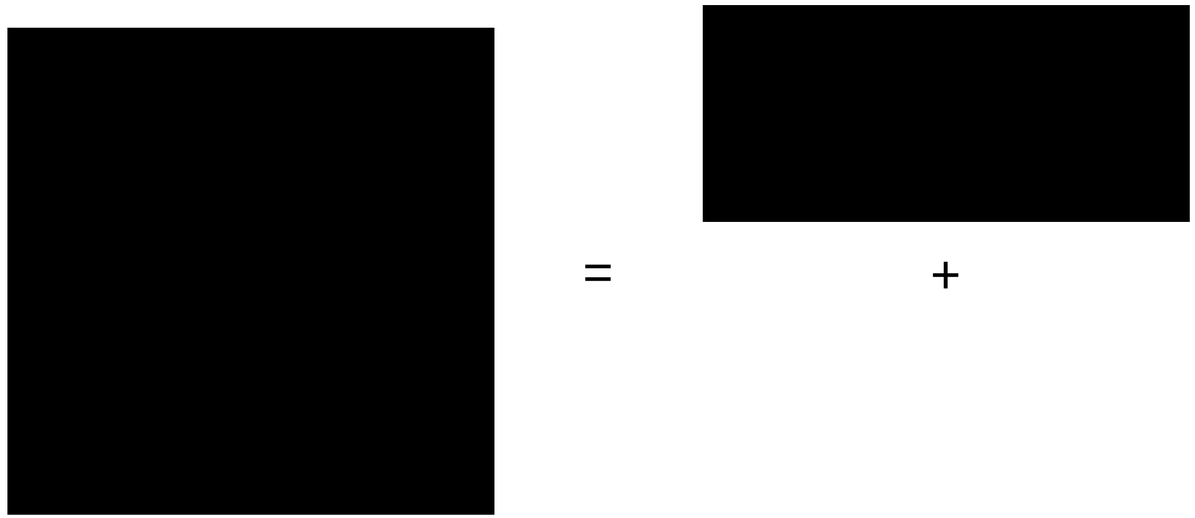


=

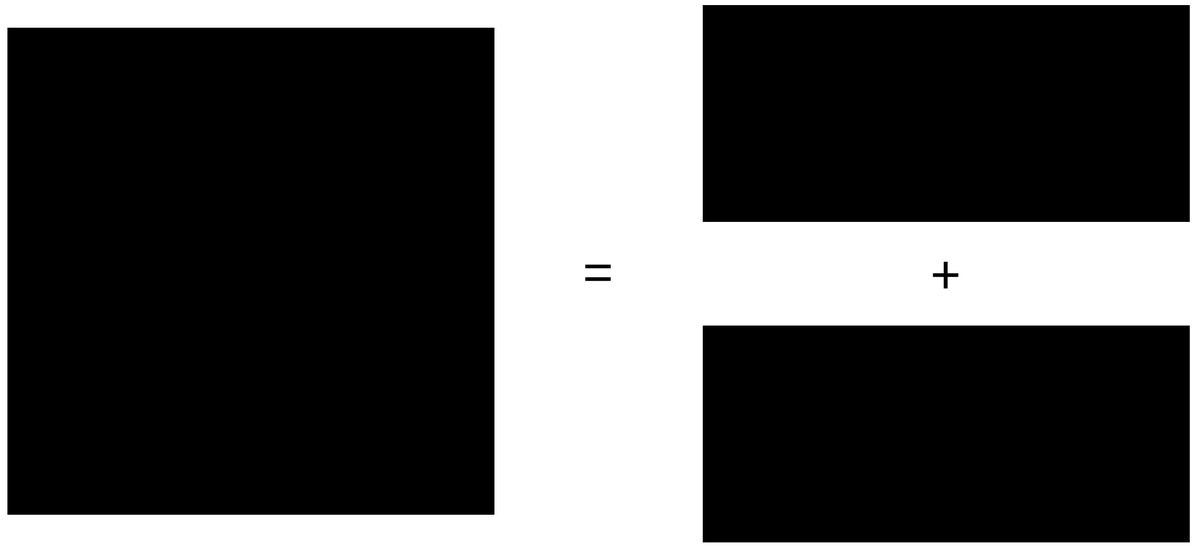
First Rewrite: Splitting



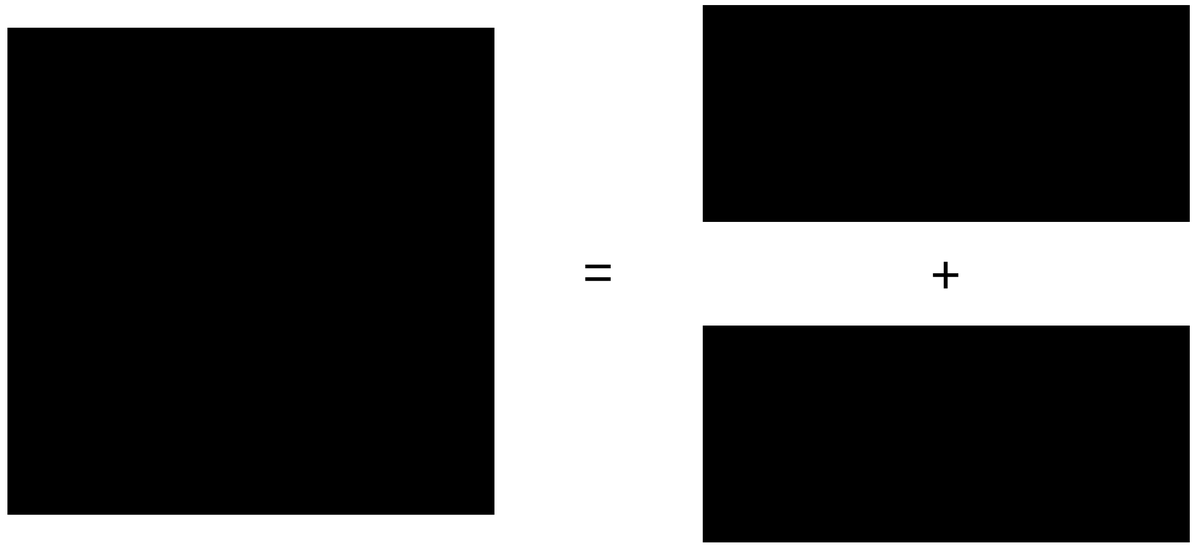
First Rewrite: Splitting



First Rewrite: Splitting

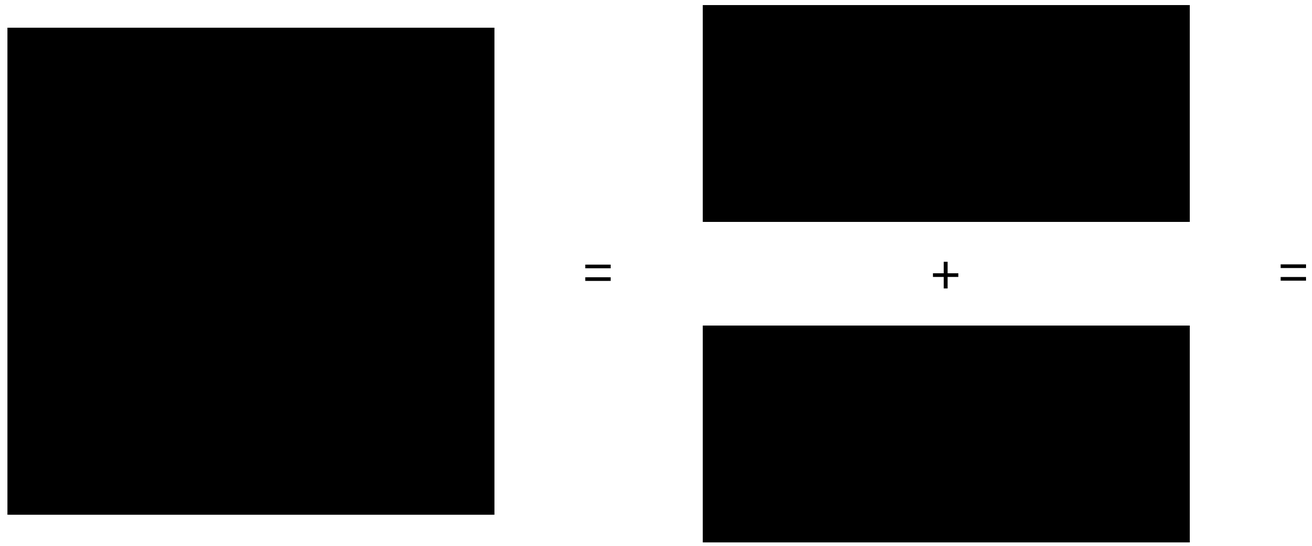


First Rewrite: Splitting



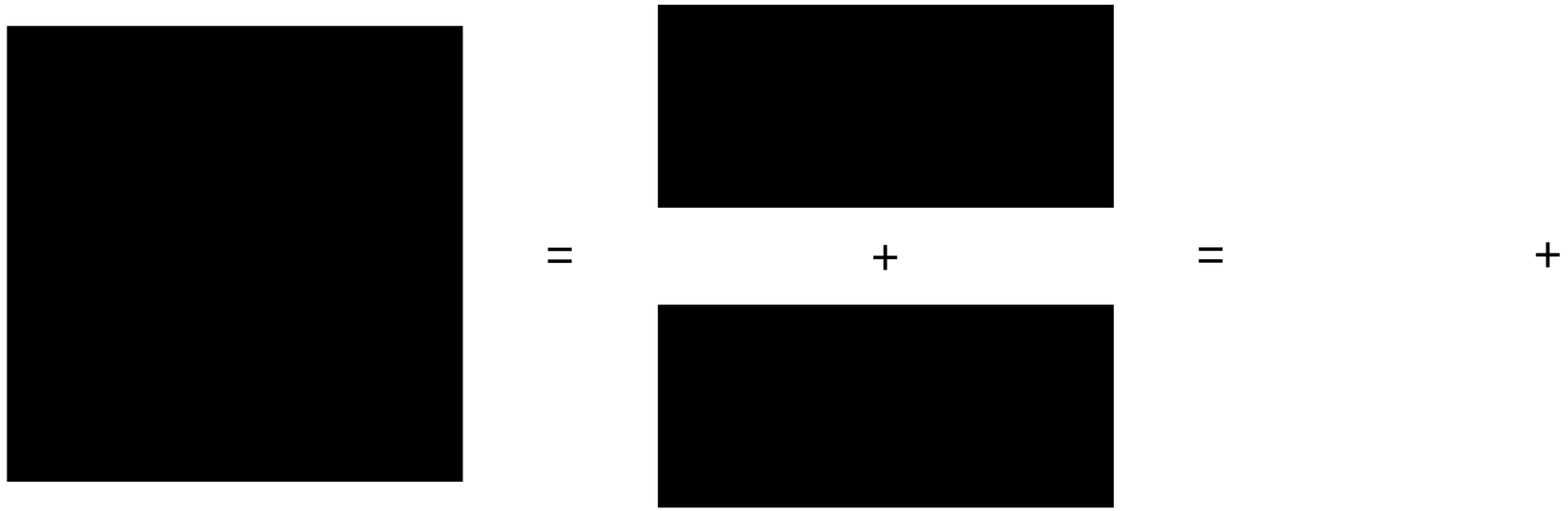

```
a => (concat (slice a ...) (slice a ...))
```

First Rewrite: Splitting



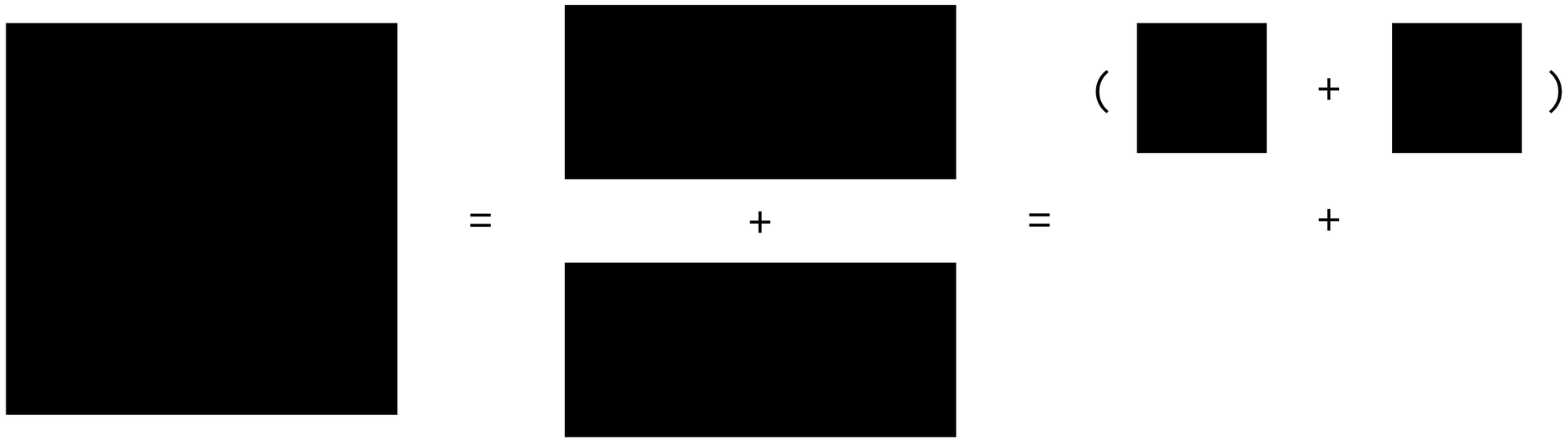

```
a => (concat (slice a ...) (slice a ...))
```

First Rewrite: Splitting



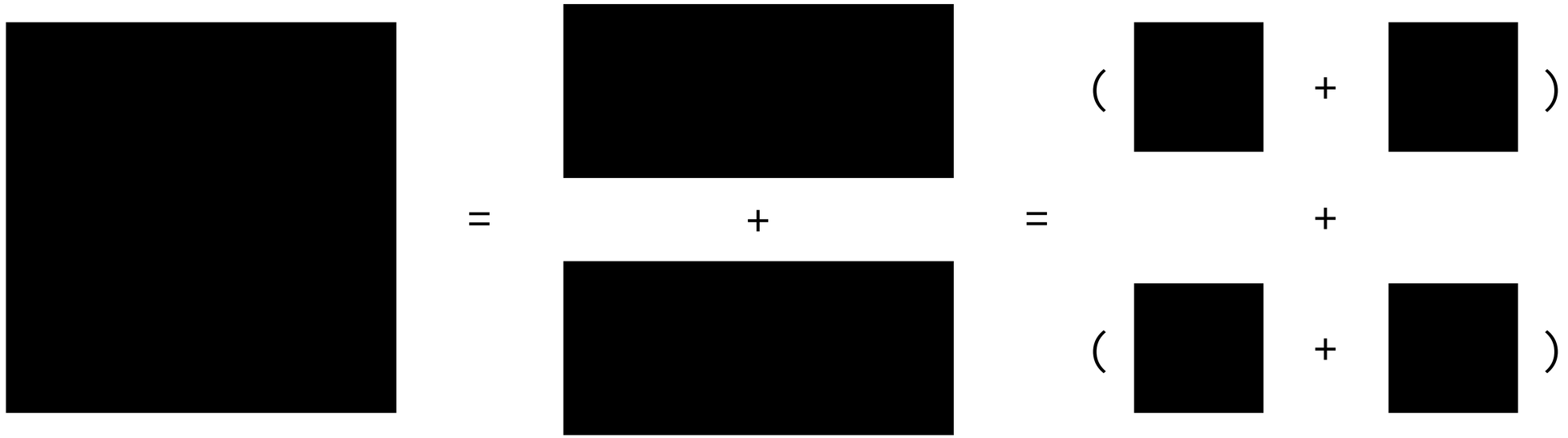

```
a => (concat (slice a ...) (slice a ...))
```

First Rewrite: Splitting

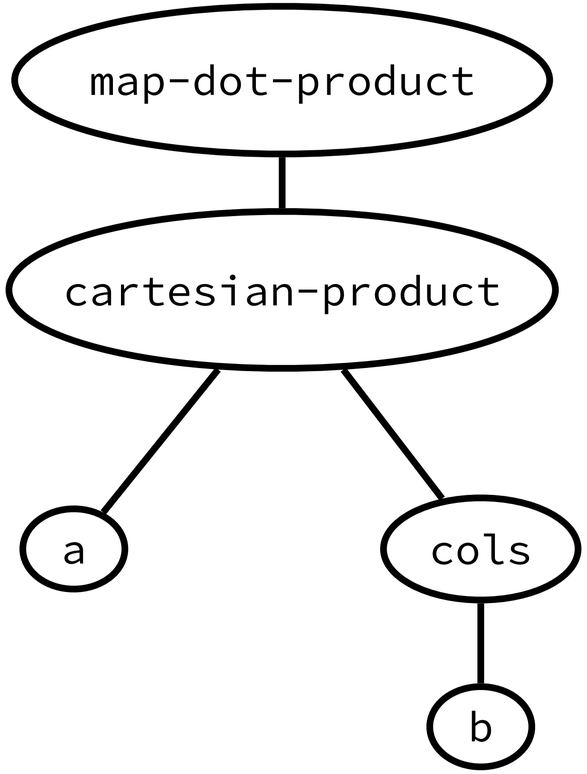


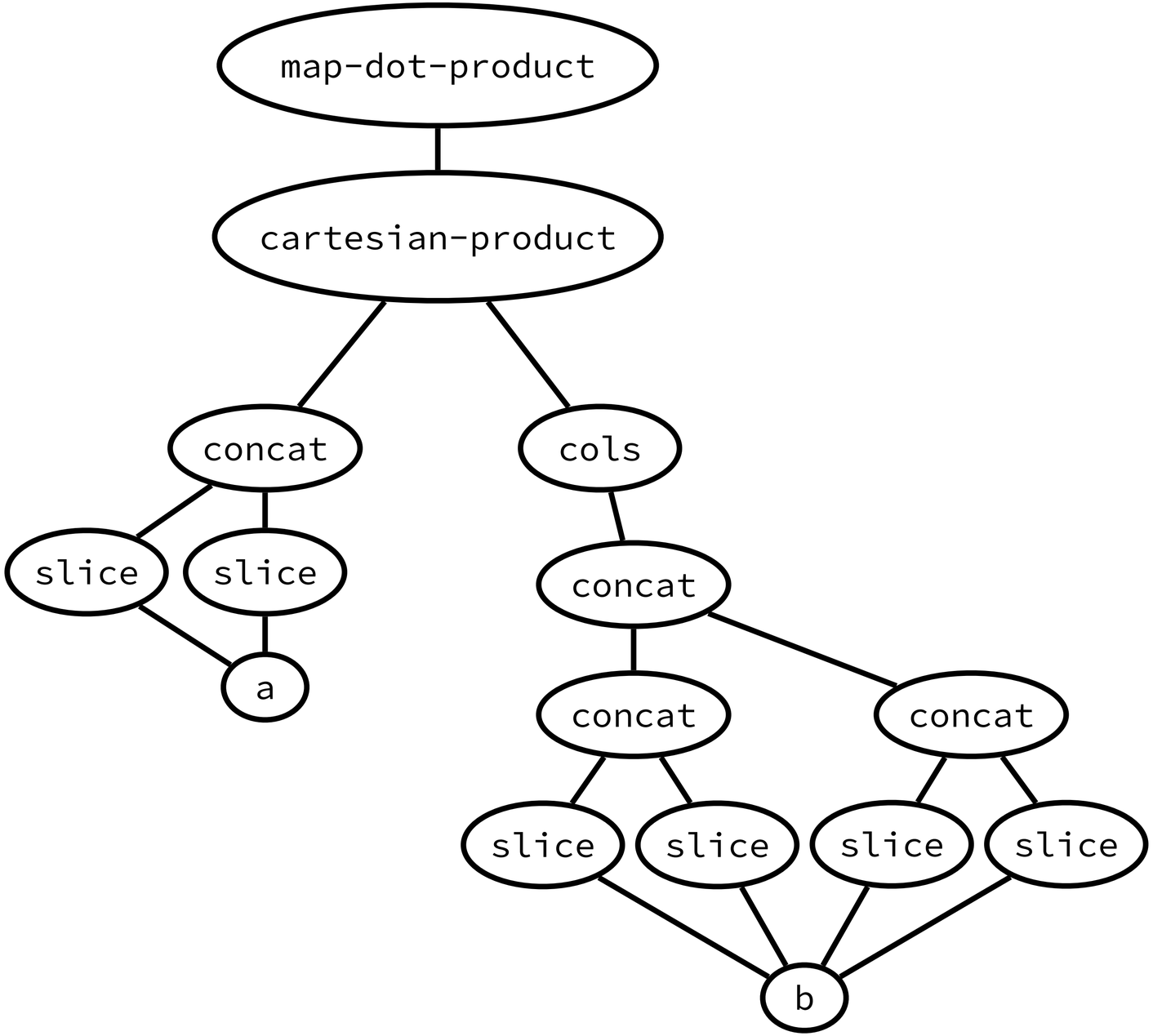

```
a => (concat (slice a ...) (slice a ...))
```

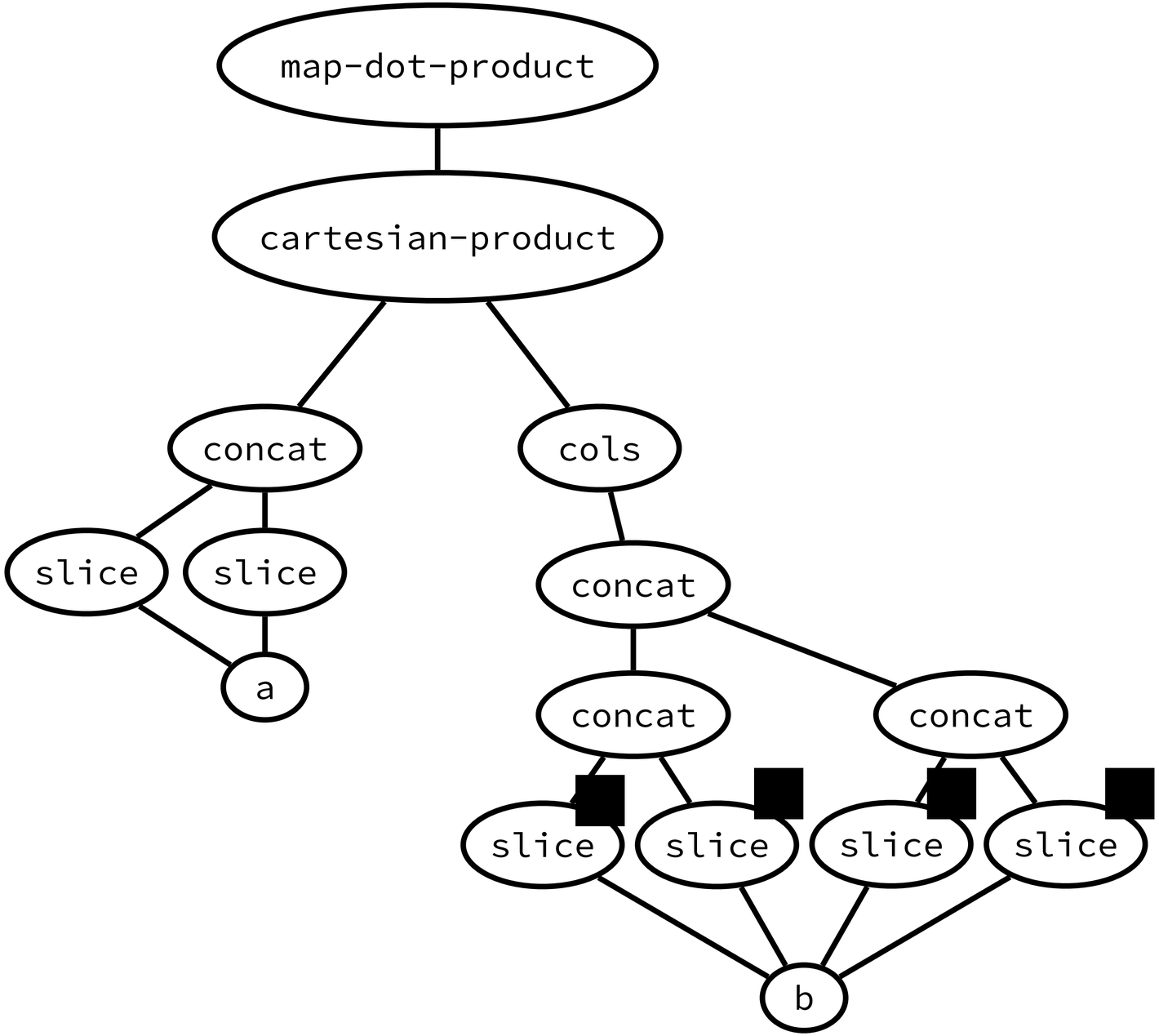
First Rewrite: Splitting

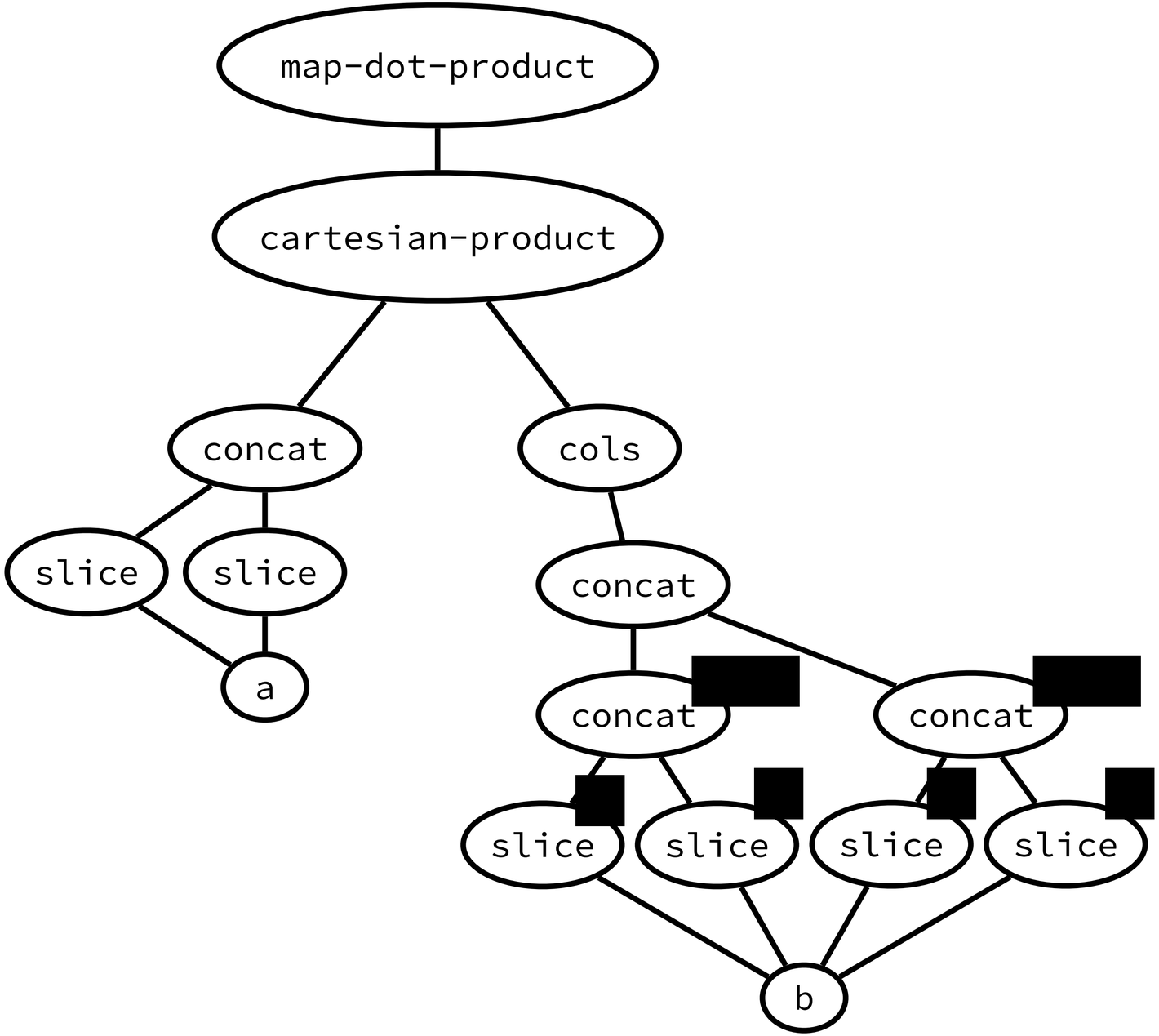


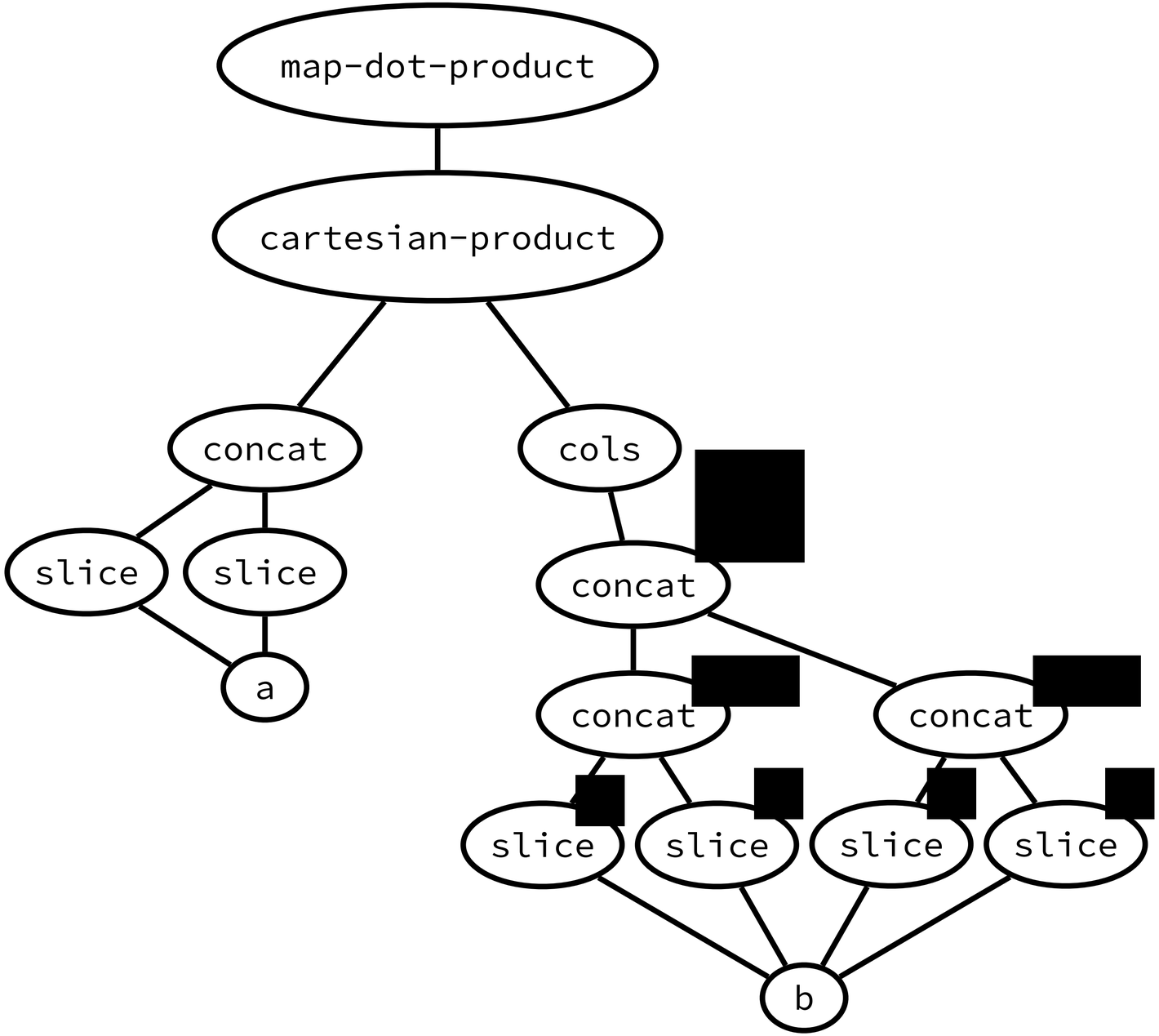

```
a => (concat (slice a ...) (slice a ...))
```

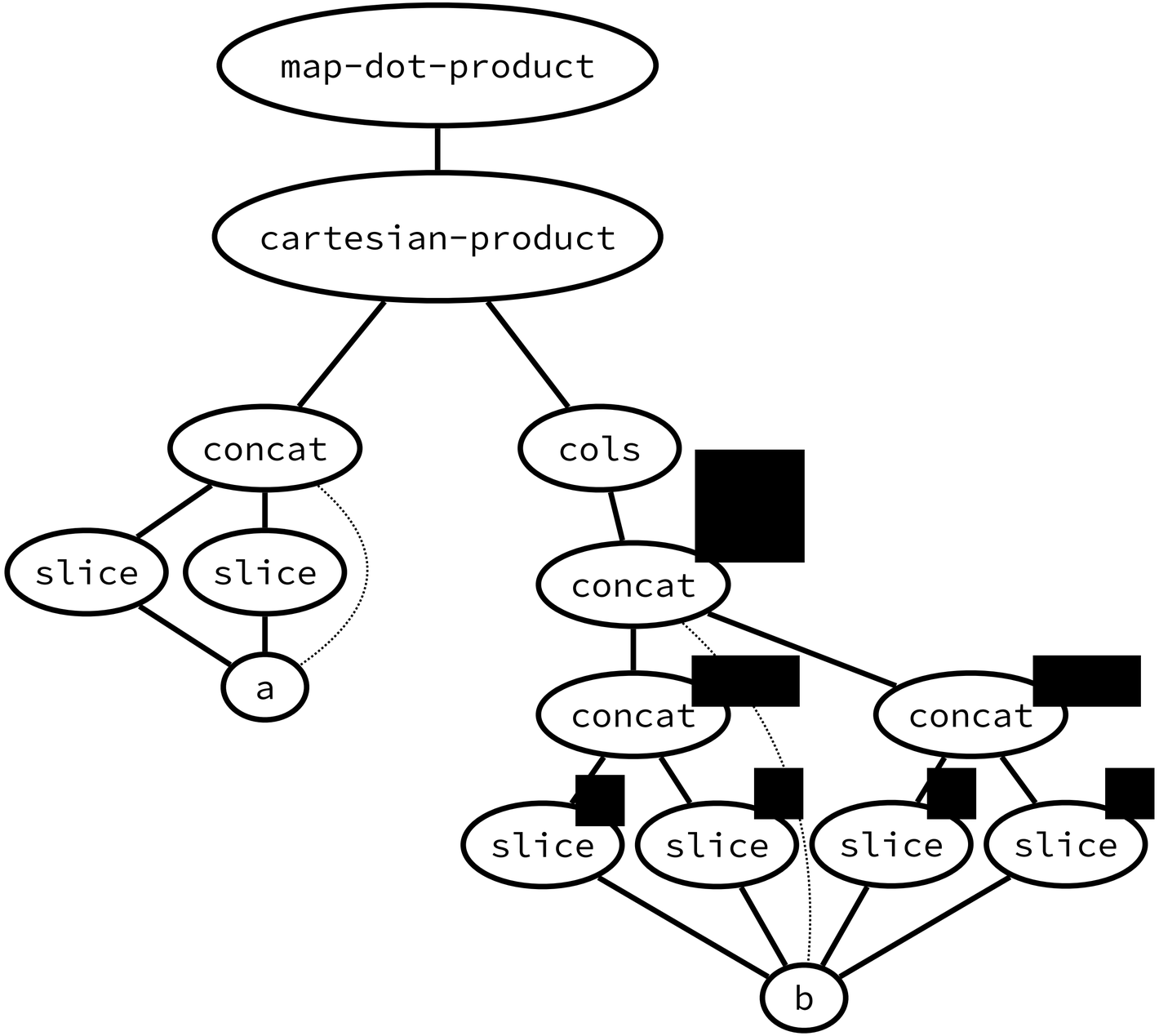

map-dot-product
cartesian-product
a
cols
b

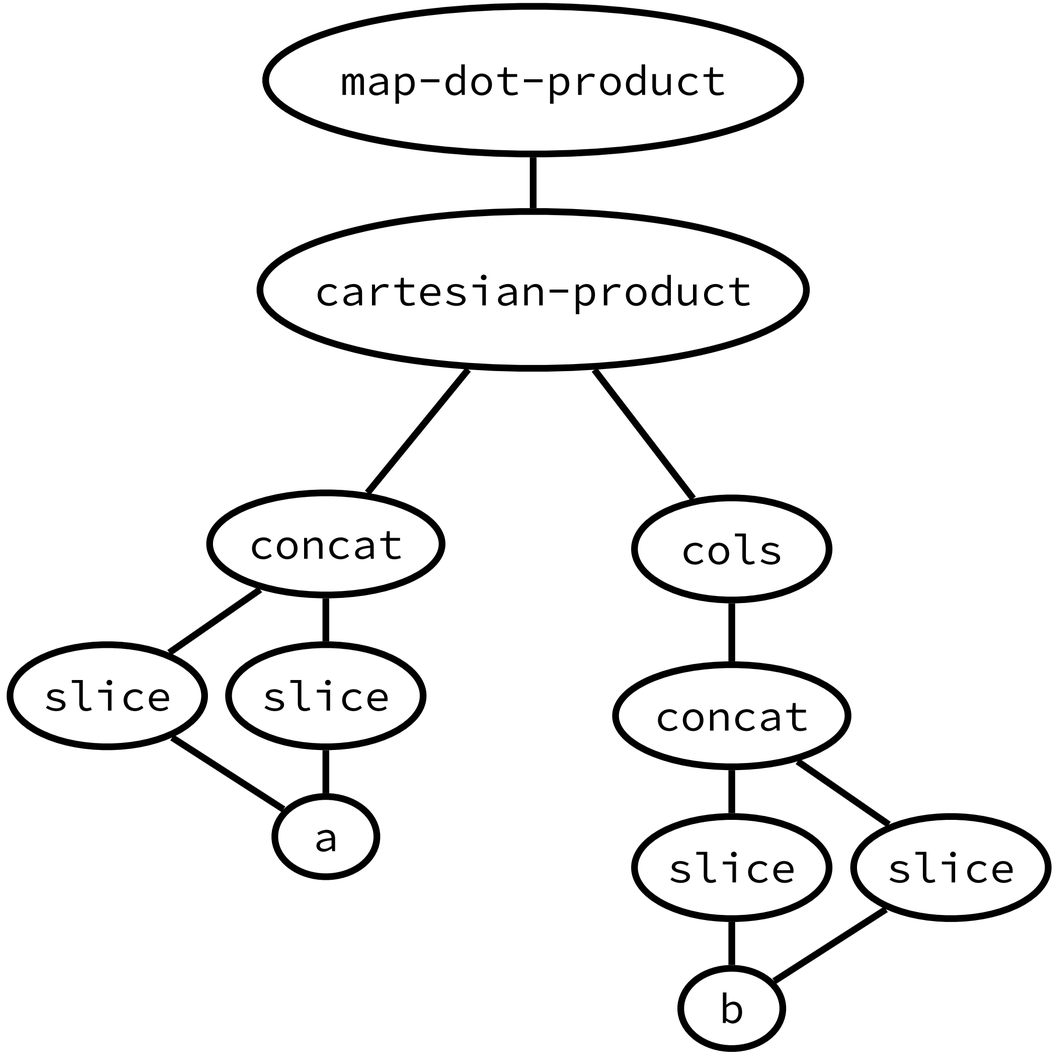

map-dot-product
cartesian-product
concat
cols
slice
slice
a
concat
concat
concat
slice
slice
slice
slice
b


map-dot-product
cartesian-product
concat
cols
slice
slice
a
concat
concat
concat
slice
slice
slice
slice
b

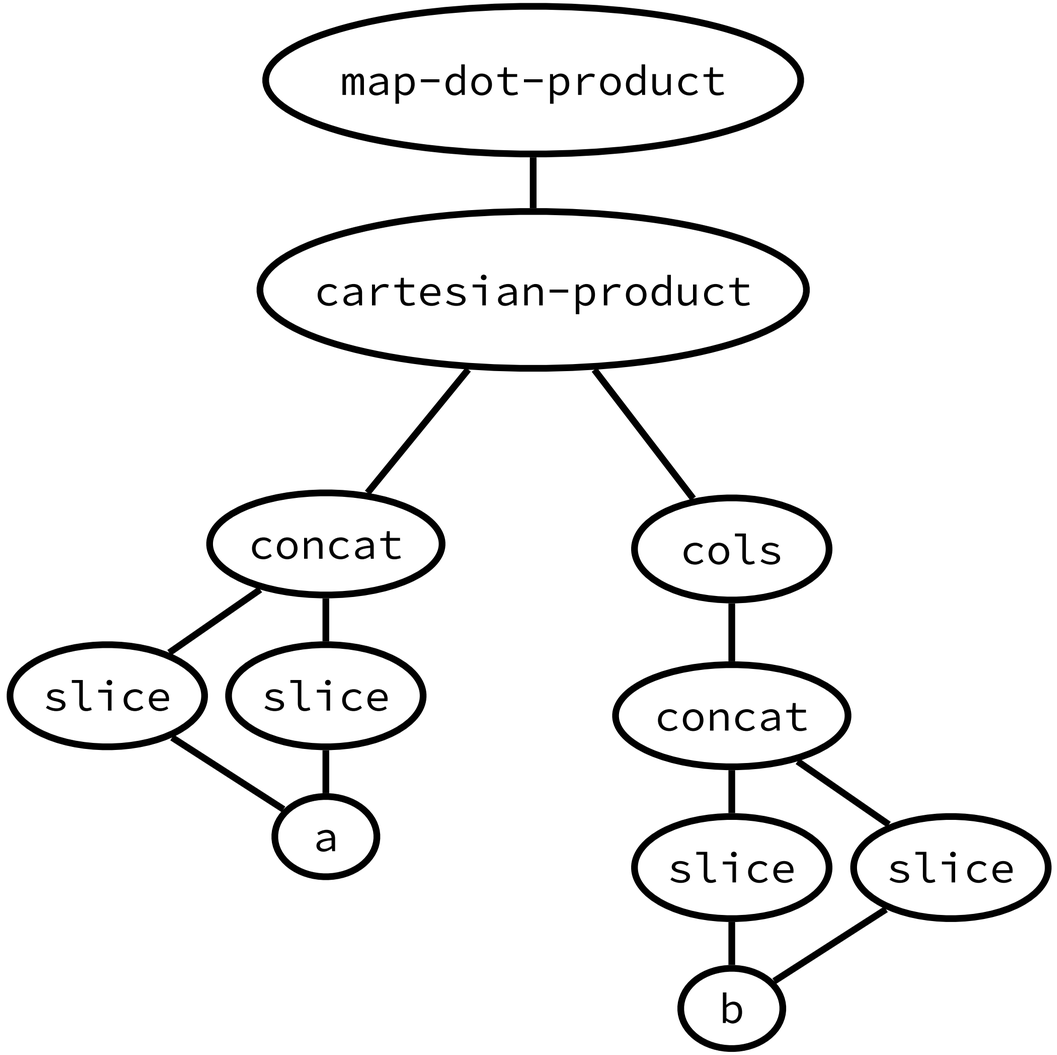

map-dot-product
cartesian-product
concat
cols
slice
slice
a
concat
concat
concat
slice
slice
slice
slice
b


map-dot-product
cartesian-product
concat
cols
slice
slice
a
concat
concat
concat
slice
slice
slice
slice
b

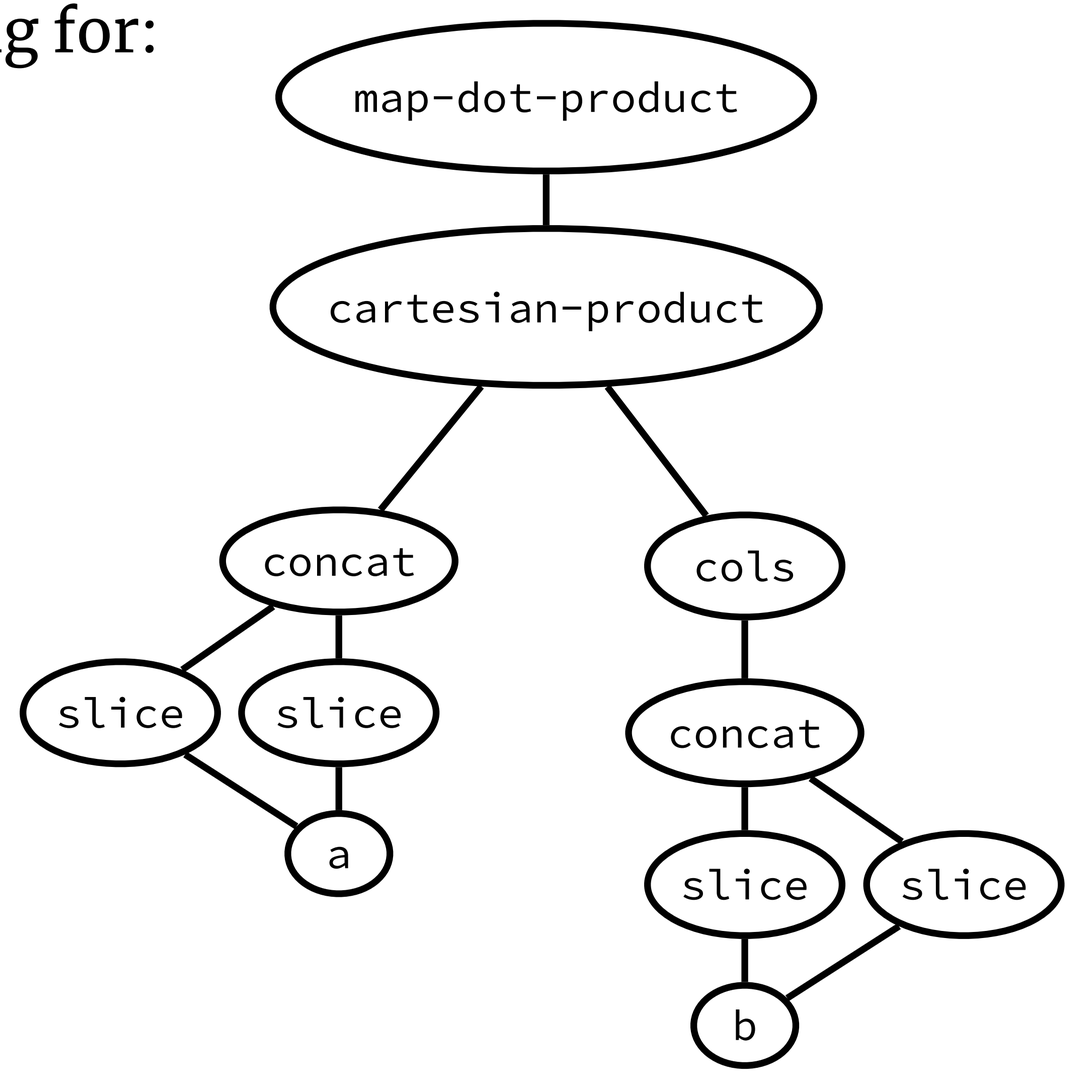

map-dot-product
cartesian-product
concat
cols
slice
slice
a
concat
concat
concat
slice
slice
slice
slice
b



(let’s keep this graph simple!)

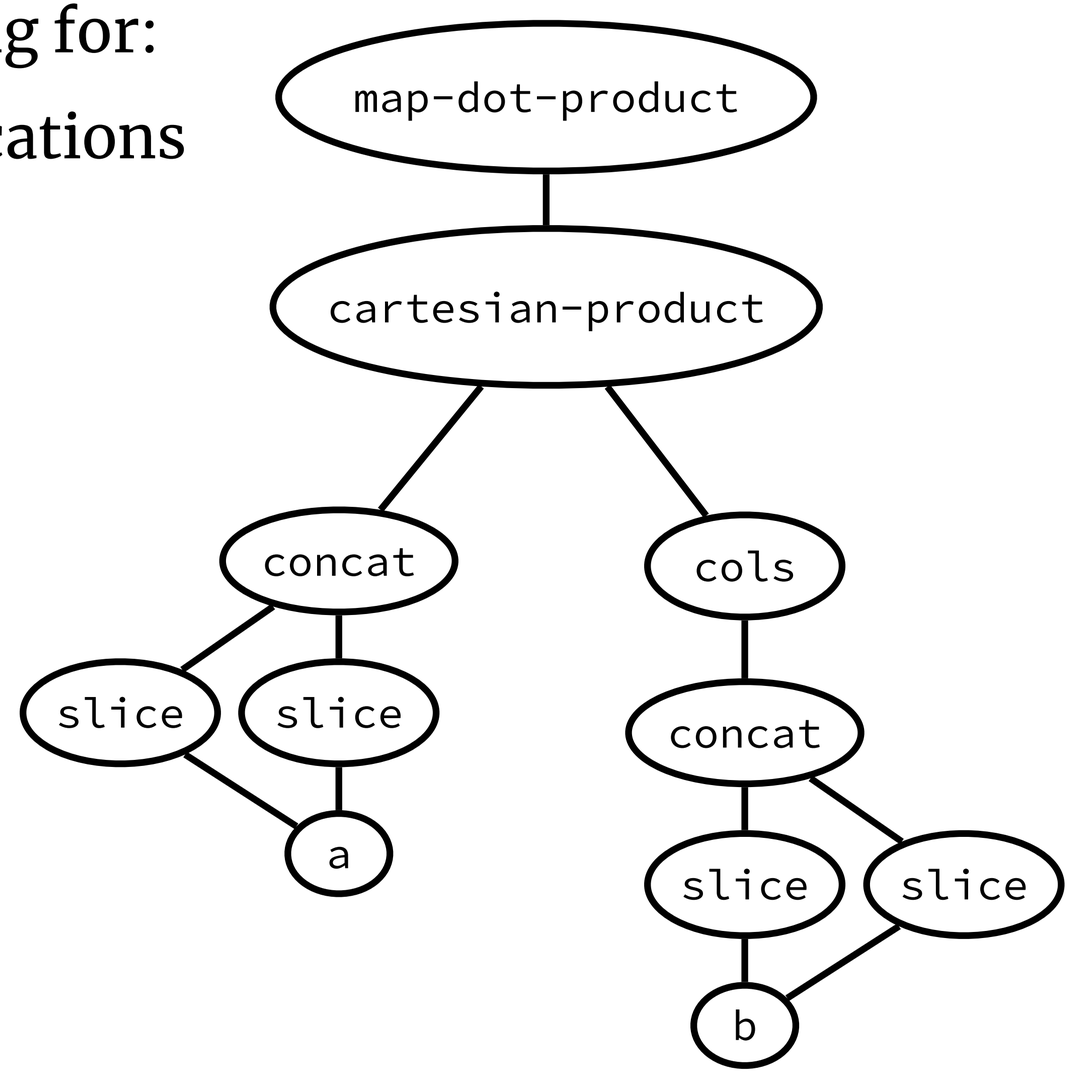

map-dot-product
cartesian-product
concat
cols
slice
slice
concat
a
slice
slice
b

Recall what we're looking for:



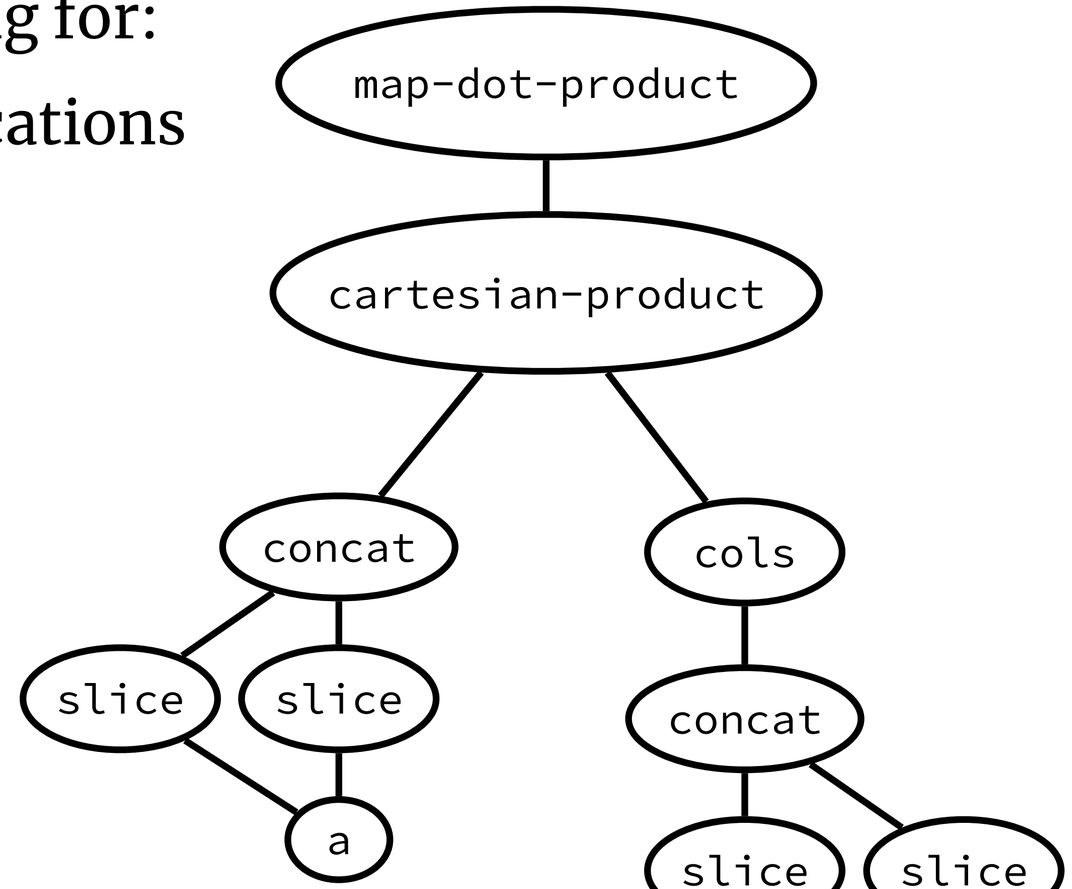
Recall what we're looking for:

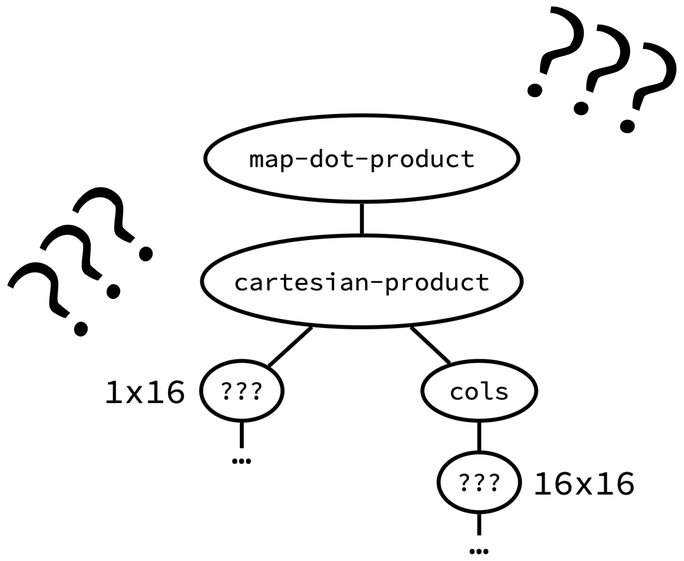
Vector–matrix multiplications of different sizes!



Recall what we're looking for:

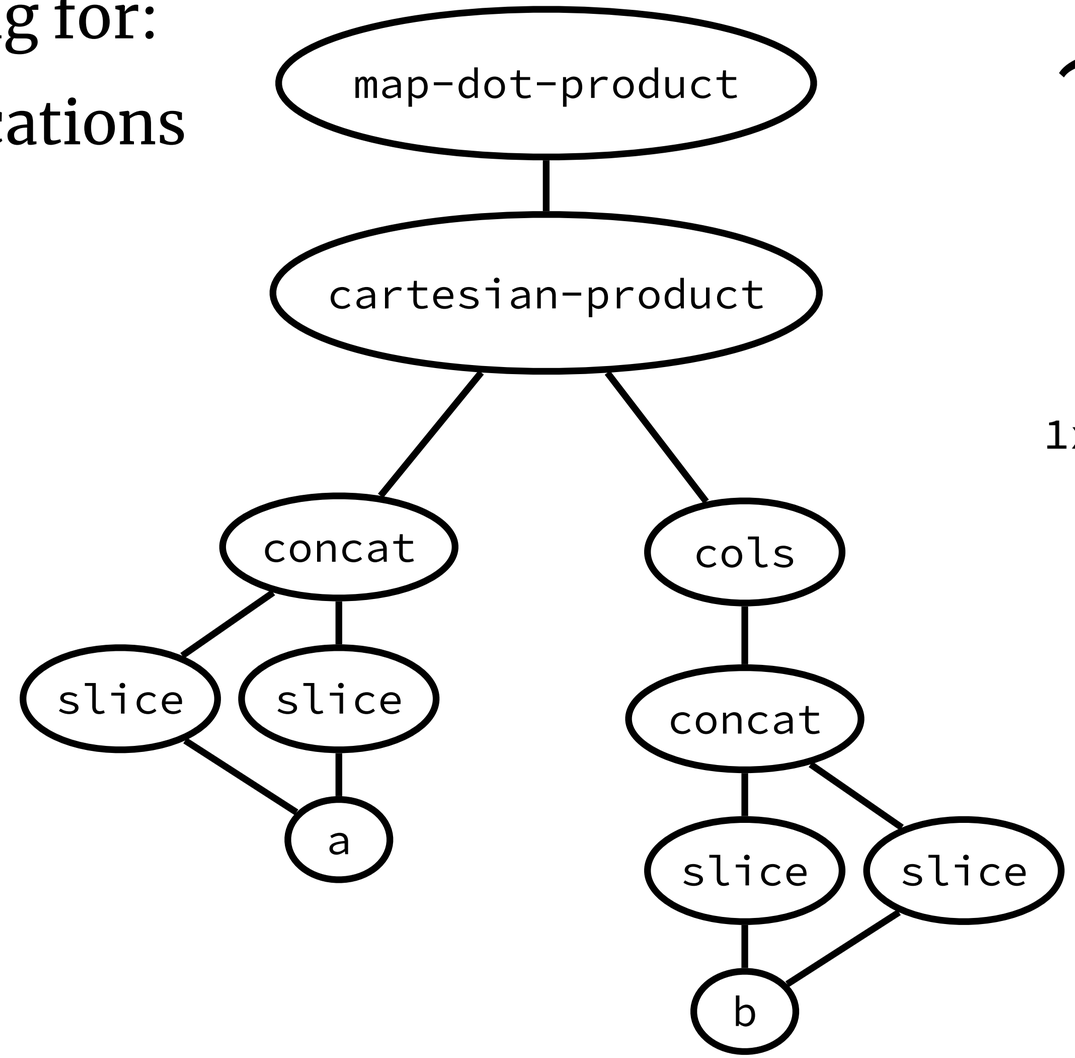
Vector–matrix multiplications of different sizes!

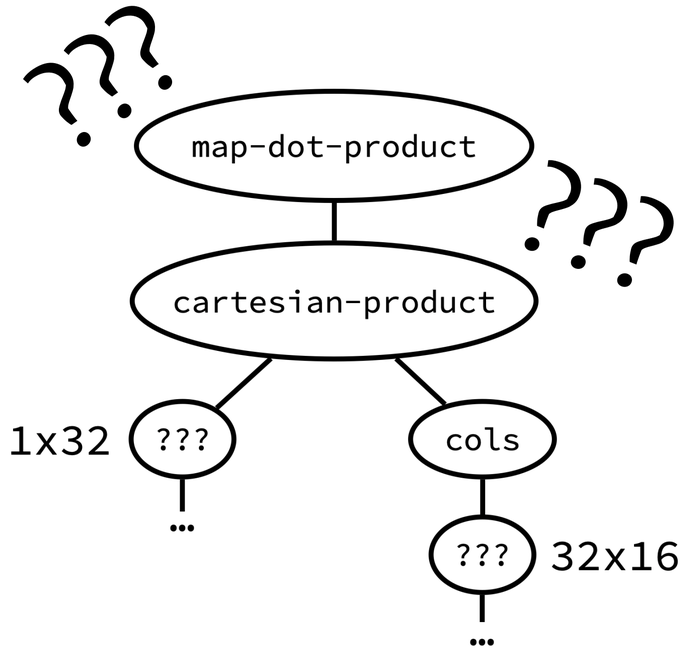


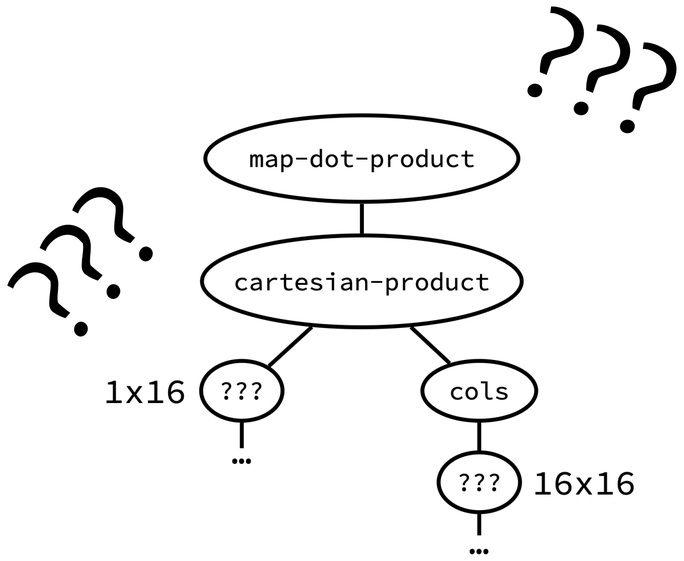


Recall what we're looking for:

Vector–matrix multiplications of different sizes!

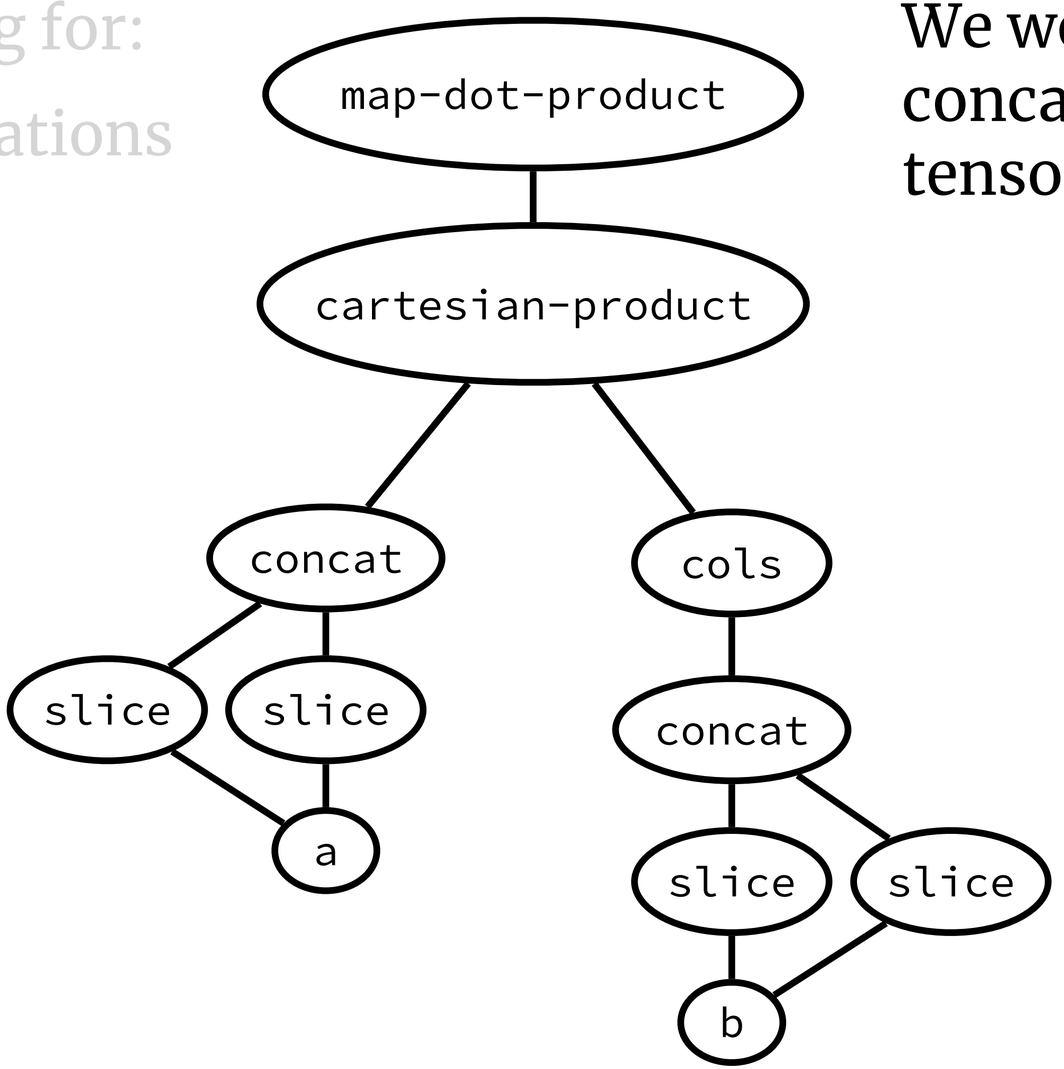






Recall what we're looking for:

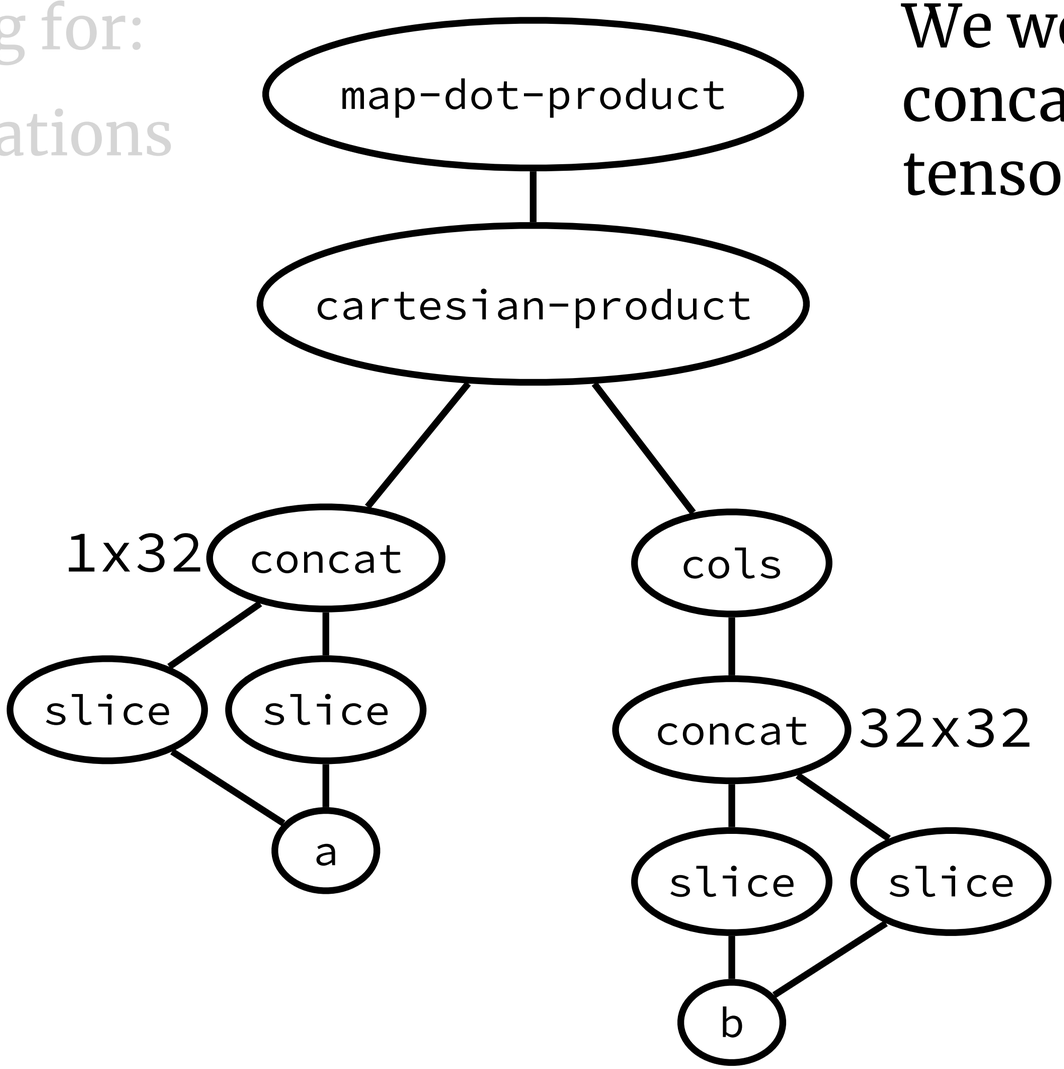
Vector–matrix multiplications of different sizes!



We won't find them here...the concats are "hiding" the tensor slices!

Recall what we're looking for:

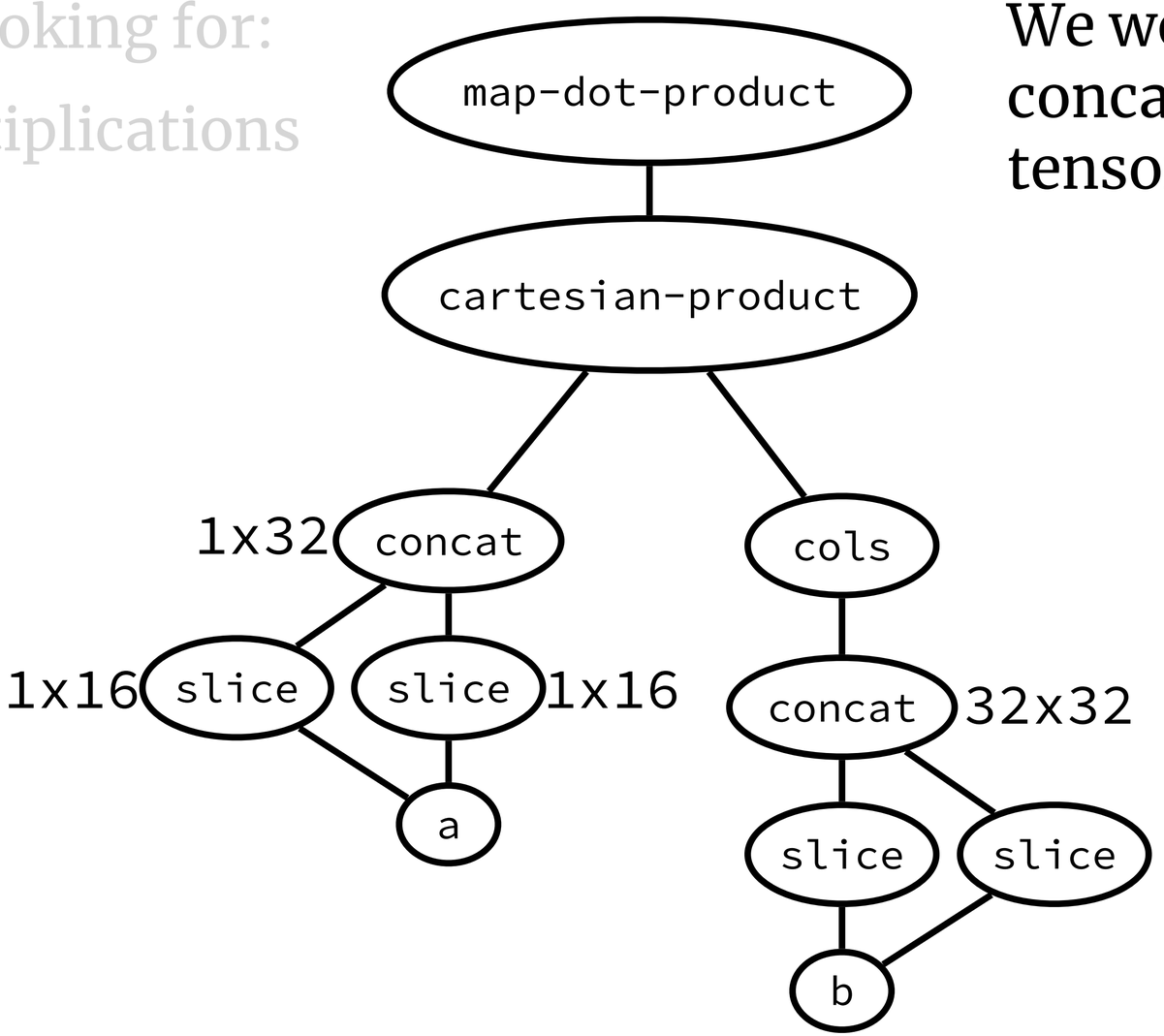
Vector–matrix multiplications of different sizes!



We won't find them here...the concats are "hiding" the tensor slices!

Recall what we're looking for:

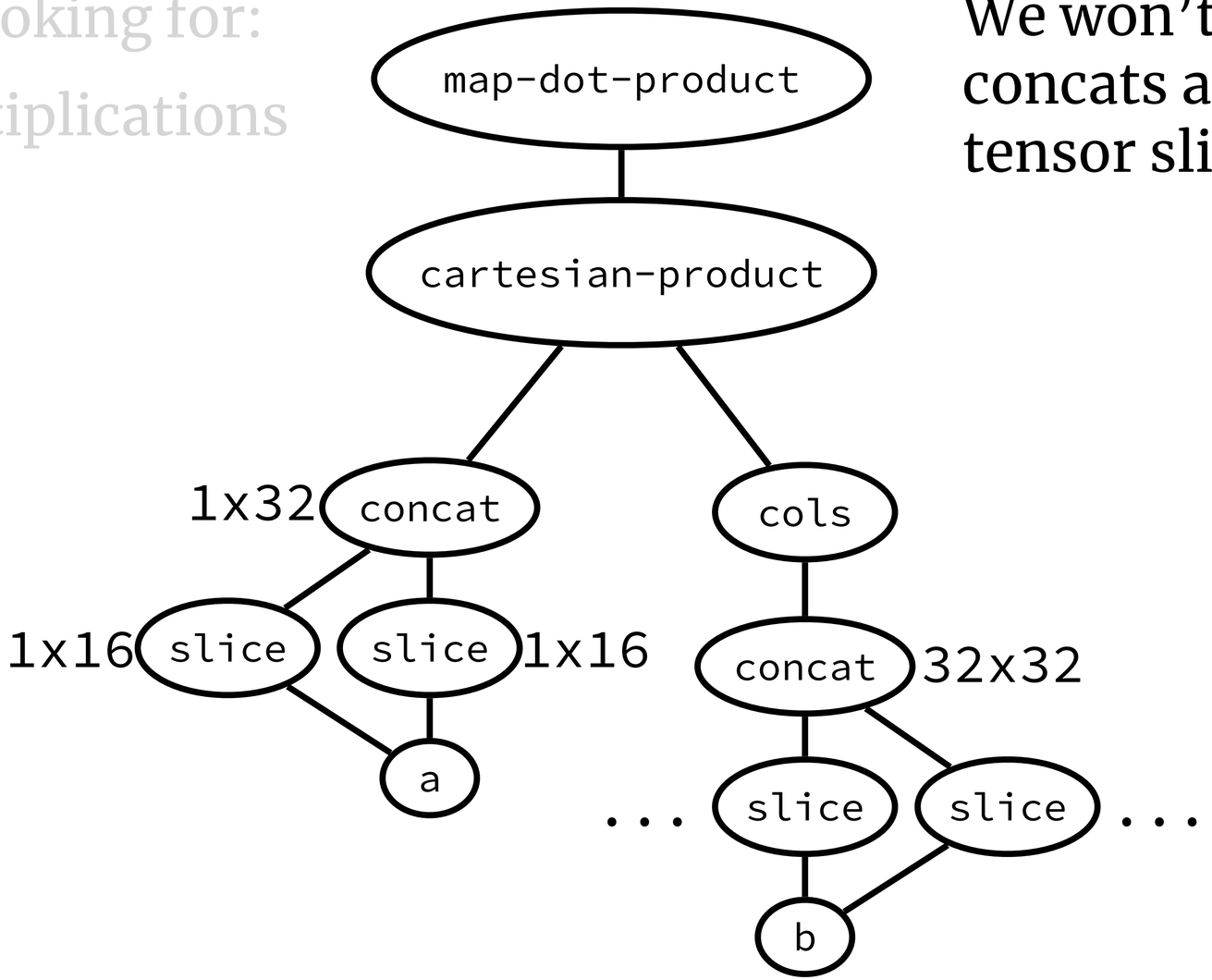
Vector–matrix multiplications of different sizes!



We won't find them here...the concats are "hiding" the tensor slices!

Recall what we're looking for:

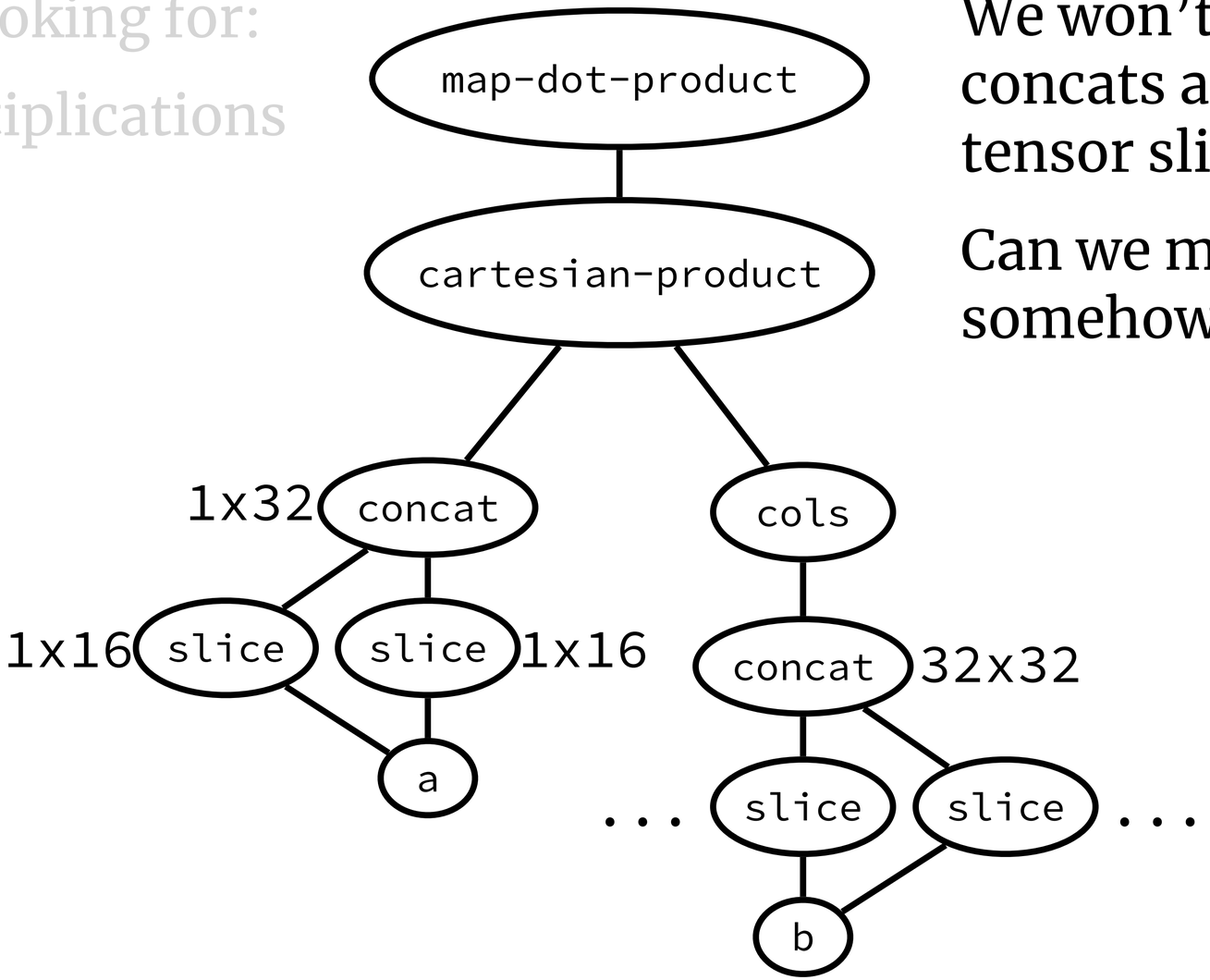
Vector–matrix multiplications of different sizes!



We won't find them here...the concats are "hiding" the tensor slices!

Recall what we're looking for:

Vector–matrix multiplications of different sizes!

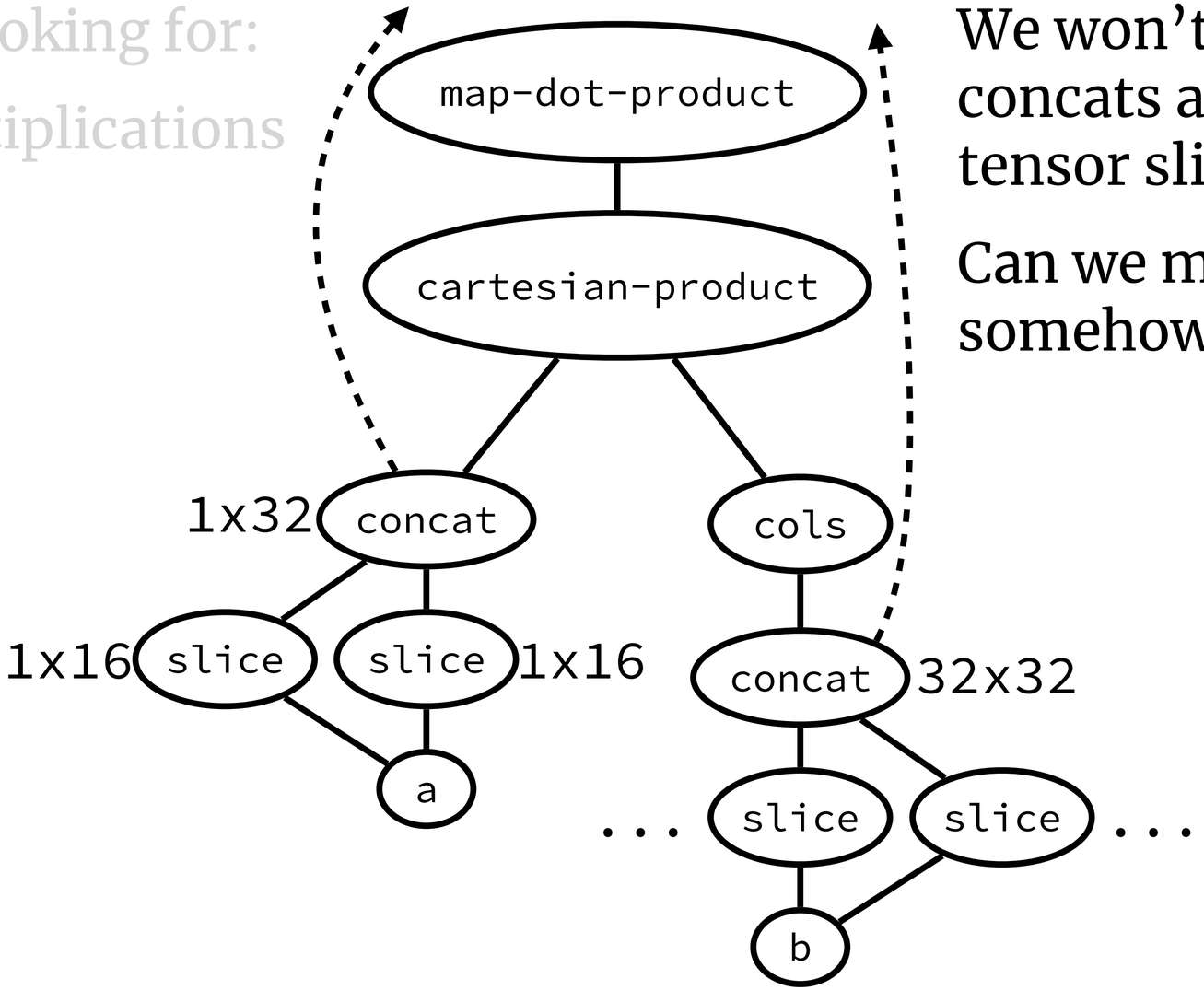


We won't find them here...the concats are "hiding" the tensor slices!

Can we move the concats somehow?

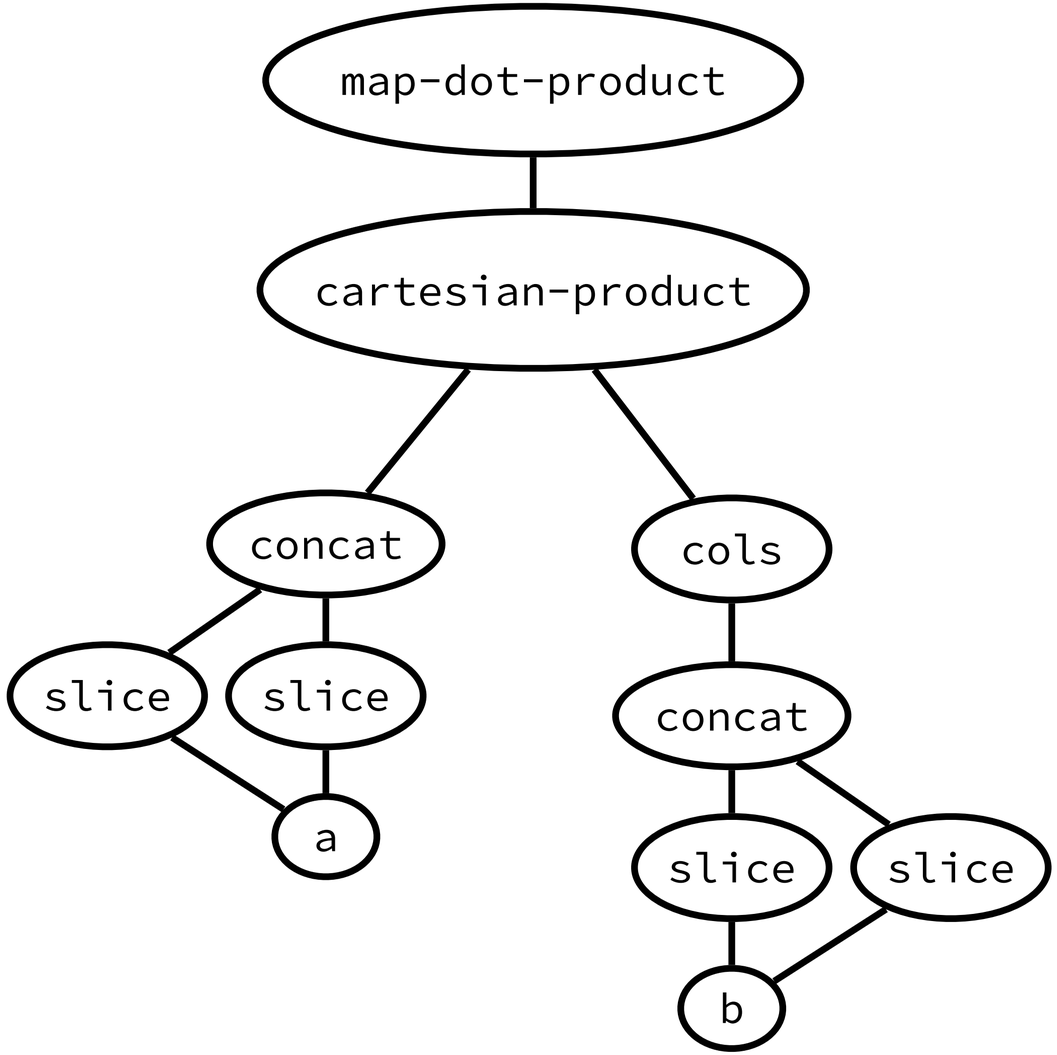
Recall what we're looking for:

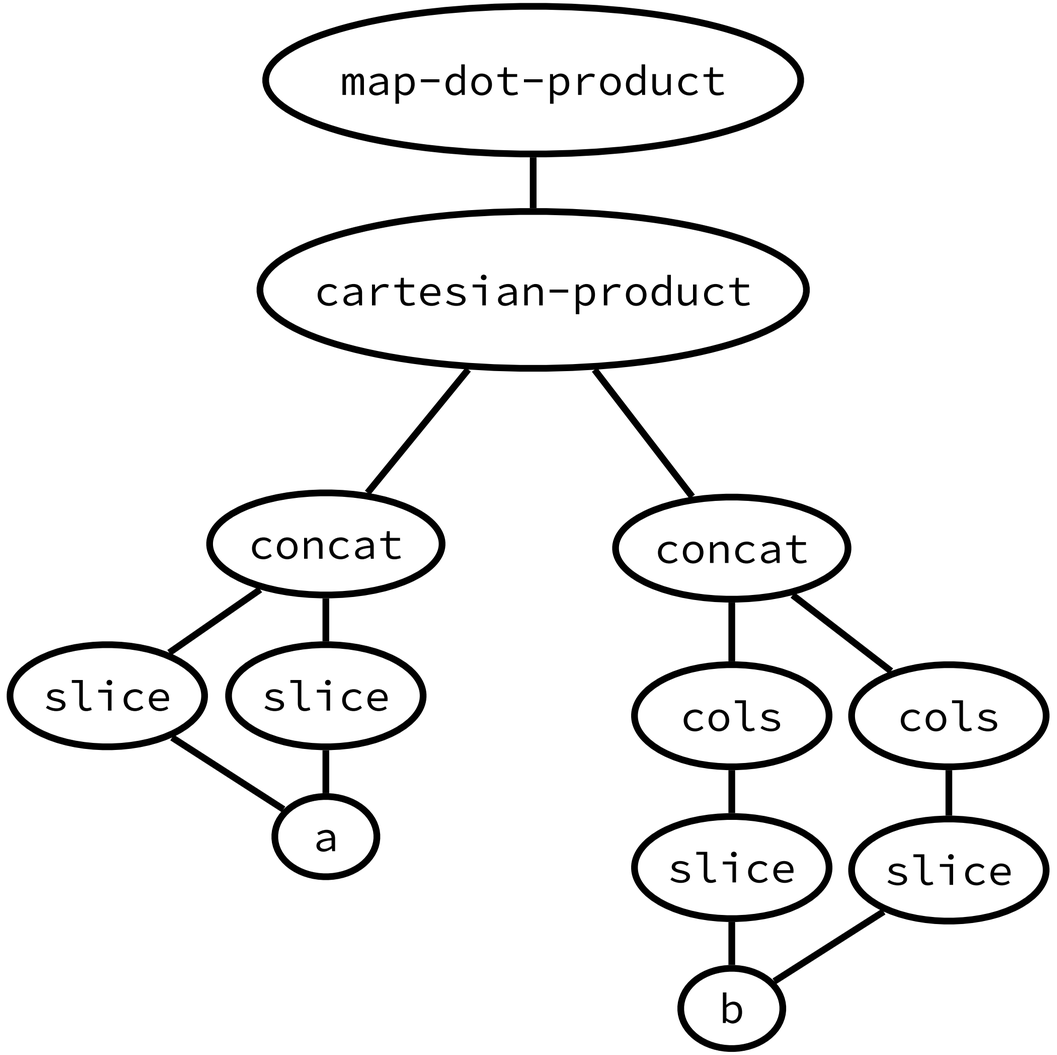
Vector–matrix multiplications of different sizes!



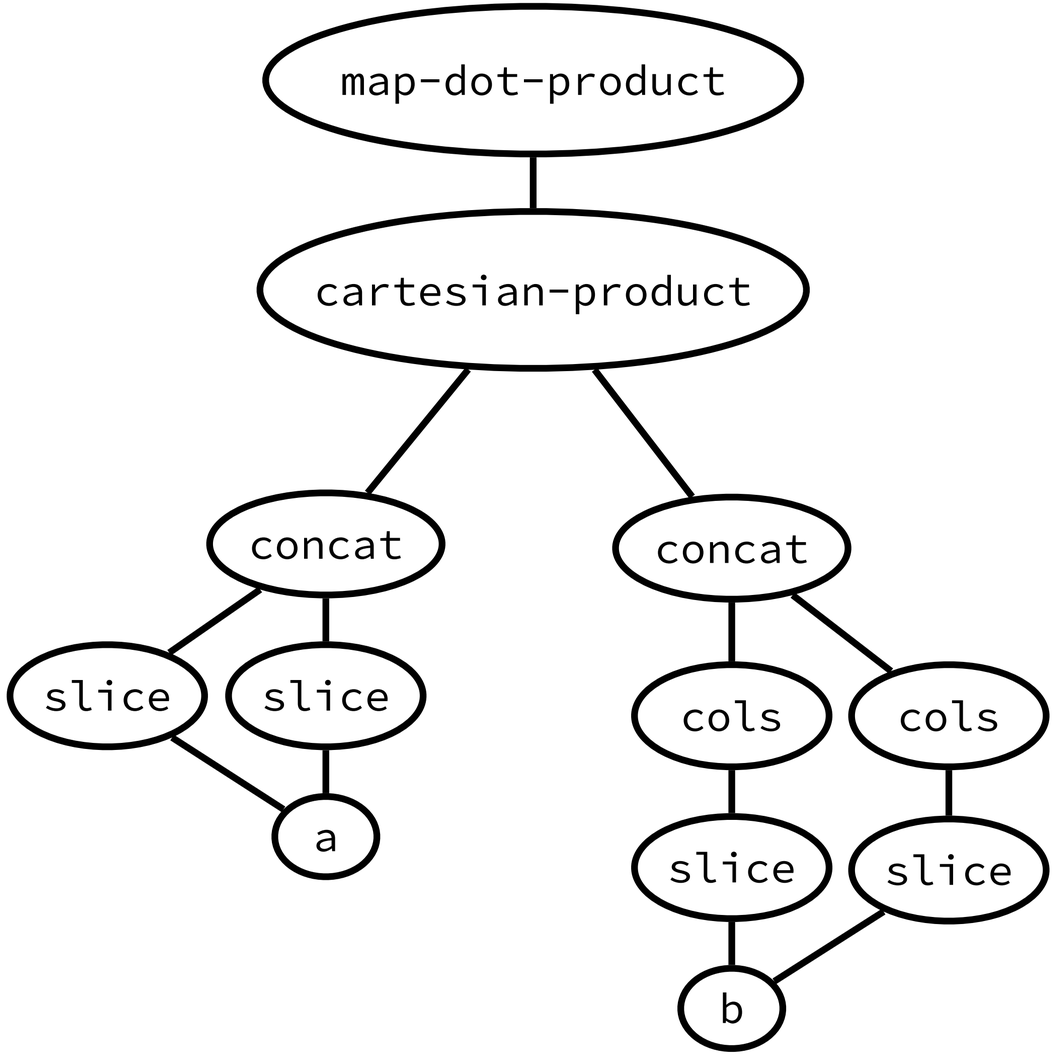
We won't find them here...the concats are "hiding" the tensor slices!

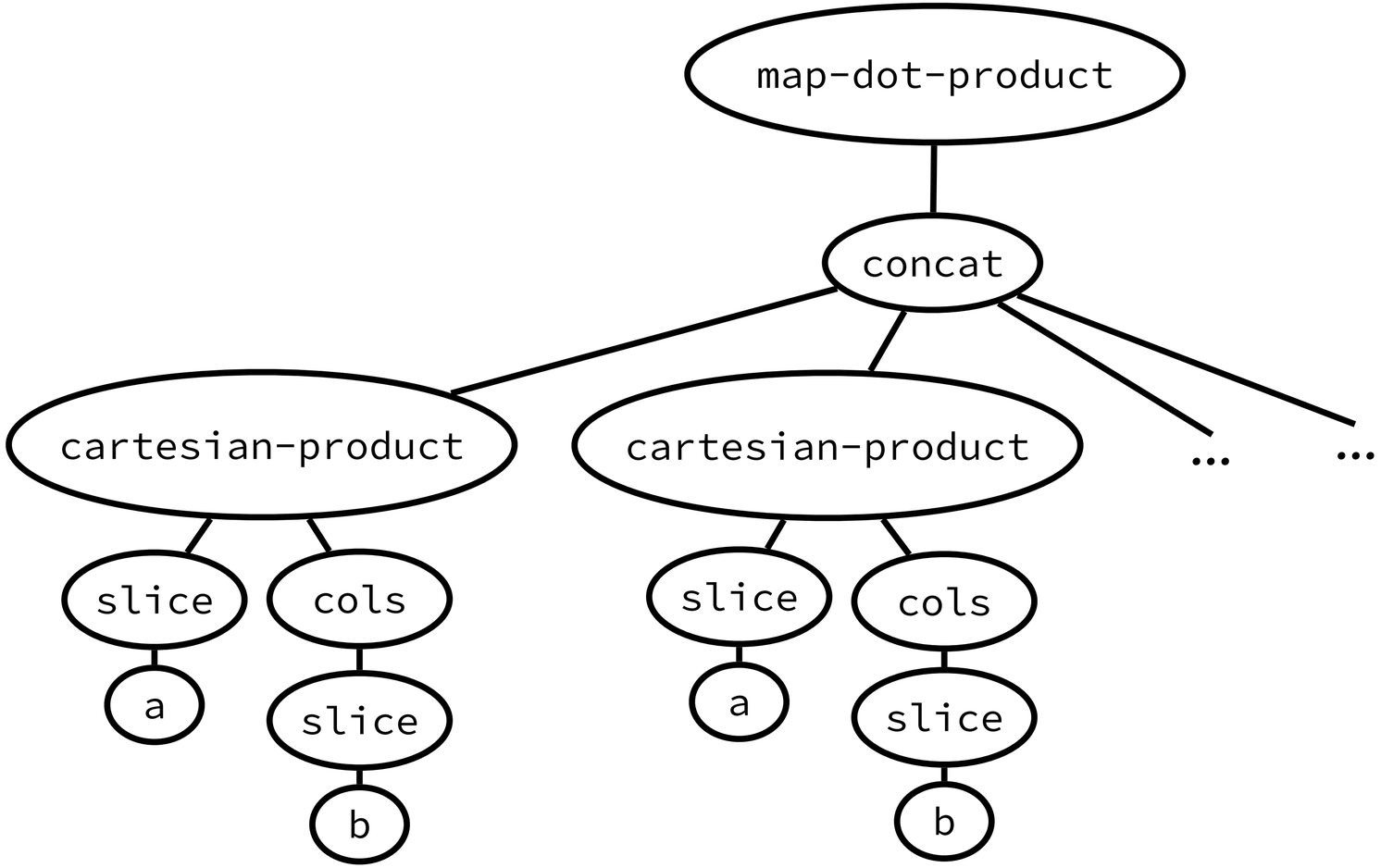
Can we move the concats somehow?

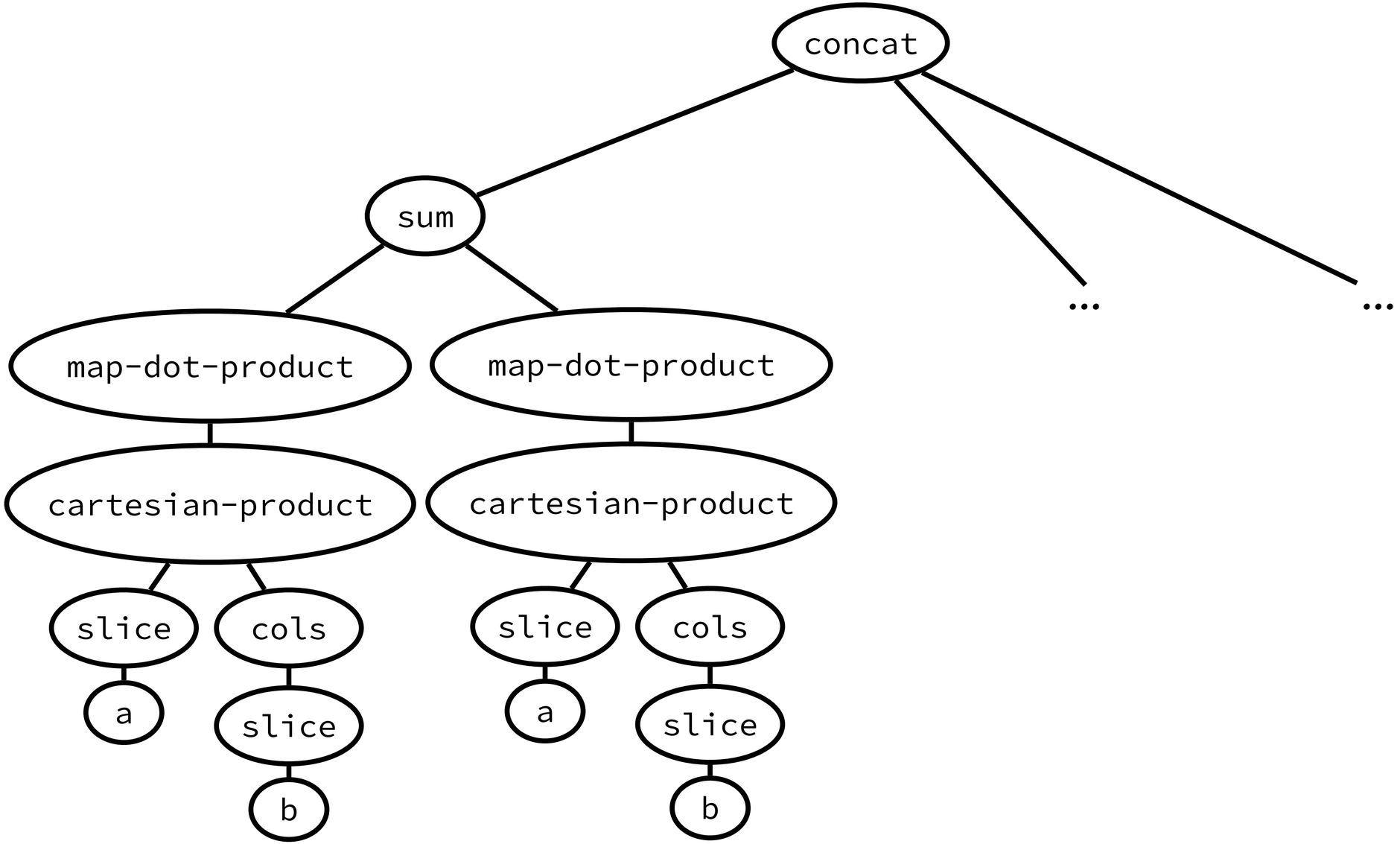

map-dot-product
cartesian-product
concat
cols
slice
slice
concat
a
slice
slice
b

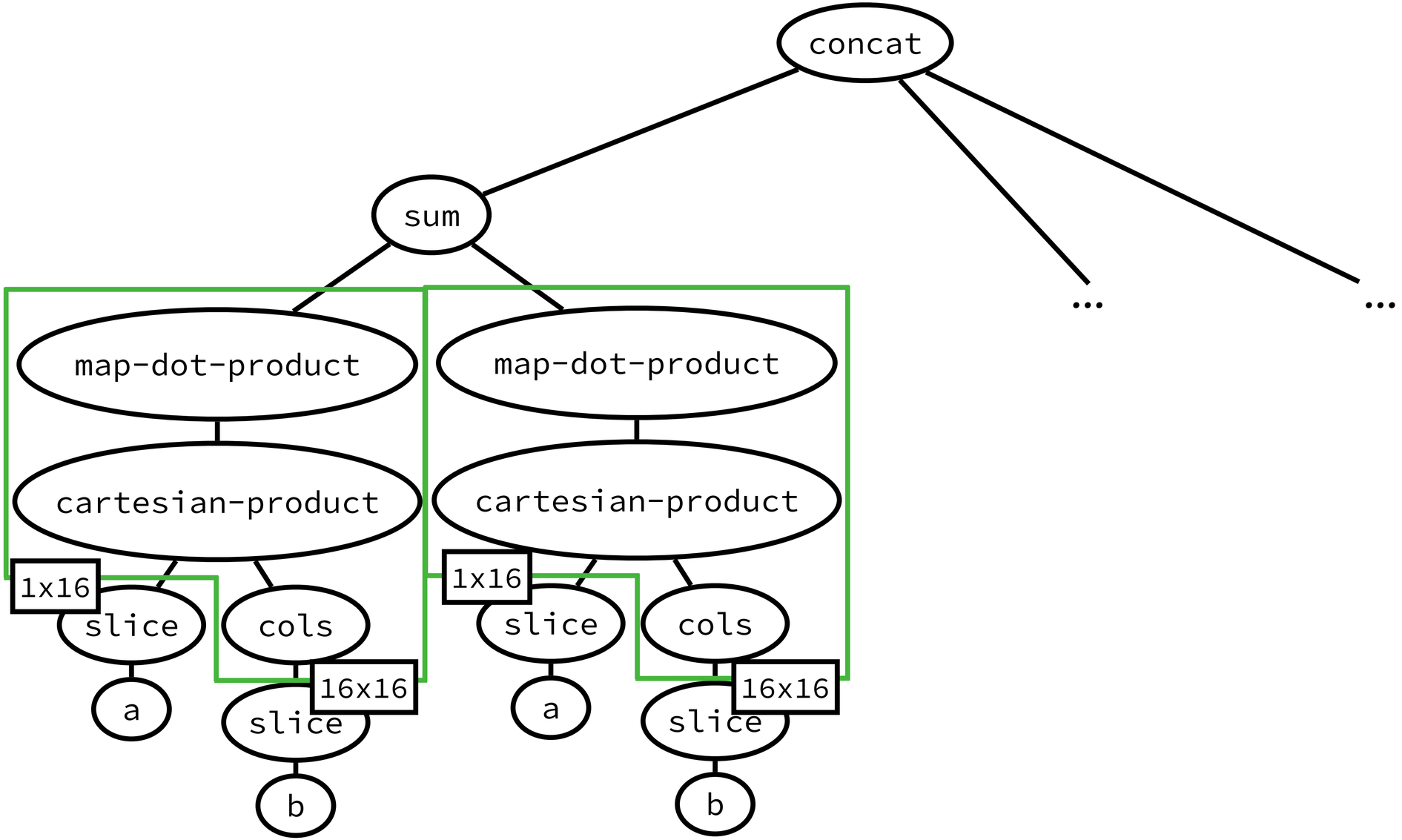


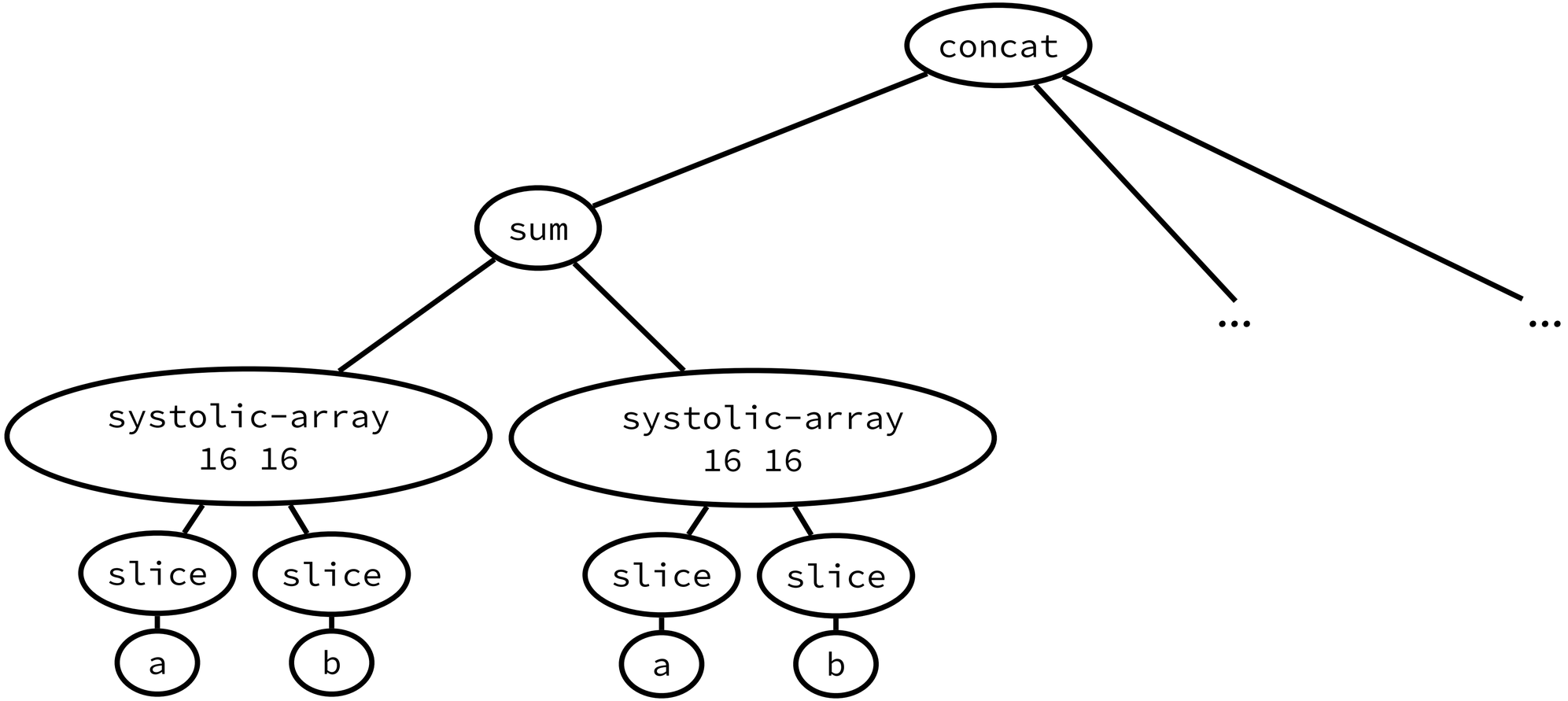

```
egg::rewrite!("bubble-concat-through-cols";
              "(cols (concat a b))"
              => "(concat (cols a) (cols b))")
```

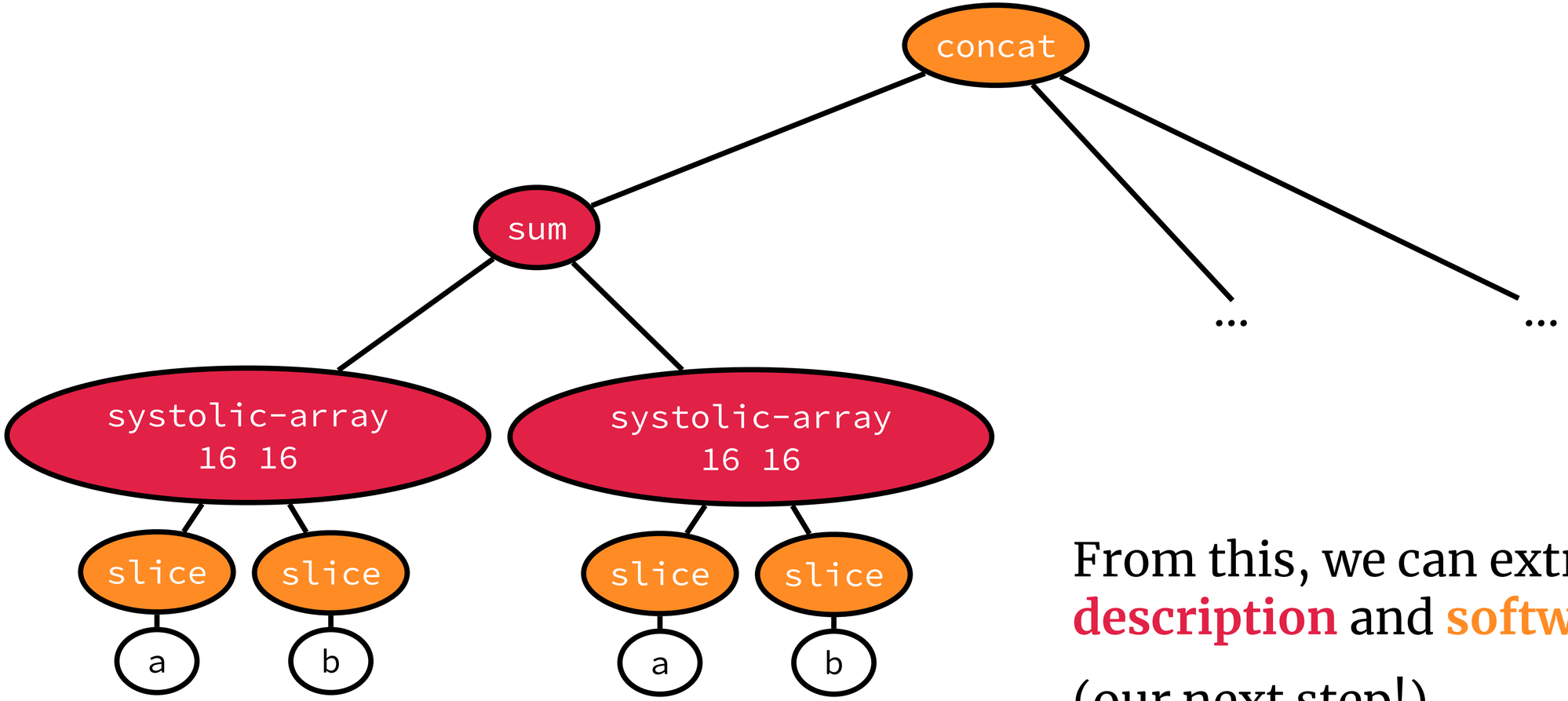

map-dot-product
cartesian-product
concat
concat
slice
slice
cols
cols
a
slice
slice
b


map-dot-product
concat
cartesian-product
cartesian-product
...
...
slice
cols
slice
cols
a
slice
a
slice
b
b


concat
sum
...
...
map-dot-product
map-dot-product
cartesian-product
cartesian-product
slice
cols
slice
cols
a
slice
a
slice
b
b


concat
sum
...
...
map-dot-product
map-dot-product
cartesian-product
cartesian-product
1x16
1x16
slice
cols
slice
cols
16x16
16x16
a
a
slice
slice
b
b


concat
sum
...
...
systolic-array
16 16
systolic-array
16 16
slice
slice
slice
slice
a
b
a
b



From this, we can extract a **hardware description** and **software schedule**.

(our next step!)

Early Results

Early Results

```
let program = "
 (map-dot-product
  (cartesian-product
   v-32
   (cols t-32-32)
  )
 )
 "
.parse()
.unwrap();
```

Early Results

```
let program = "
 (map-dot-product
  (cartesian-product
   v-32
   (cols t-32-32)
  )
 )
 "
.parse()
.unwrap();
```

```
let rws = vec![
    rewrites::split(0, 16, true),
    rewrites::split(1, 16, true),
    rewrites::collapse_nested_slices(),
    rewrites::bubble_concat_through_rows_axis_0(),
    rewrites::bubble_concat_through_rows_axis_1(),
    rewrites::bubble_concat_through_cols_axis_0(),
    rewrites::bubble_concat_through_cols_axis_1(),
    rewrites::bubble_concat_through_cartesian_product_not_last_axis_left(),
    rewrites::bubble_concat_through_cartesian_product_not_last_axis_right(),
    rewrites::bubble_concat_through_cartesian_product_last_axis(),
    rewrites::bubble_concat_through_map_dot_product_not_last_axis(),
    rewrites::bubble_concat_through_map_dot_product_last_axis(),
    rewrites::systolic_array_vector_matrix(),
];
```

Early Results

```
let program = "
 (map-dot-product
  (cartesian-product
   v-32
   (cols t-32-32)
  )
 )
 "
.parse()
.unwrap();
```

```
let rws = vec![
    rewrites::split(0, 16, true),
    rewrites::split(1, 16, true),
    rewrites::collapse_nested_slices(),
    rewrites::bubble_concat_through_rows_axis_0(),
    rewrites::bubble_concat_through_rows_axis_1(),
    rewrites::bubble_concat_through_cols_axis_0(),
    rewrites::bubble_concat_through_cols_axis_1(),
    rewrites::bubble_concat_through_cartesian_product_not_last_axis_left(),
    rewrites::bubble_concat_through_cartesian_product_not_last_axis_right(),
    rewrites::bubble_concat_through_cartesian_product_last_axis(),
    rewrites::bubble_concat_through_map_dot_product_not_last_axis(),
    rewrites::bubble_concat_through_map_dot_product_last_axis(),
    rewrites::systolic_array_vector_matrix(),
];
```

```
let (egraph, id) = egg::EGraph::<Language, Meta>::from_expr(&program);
let runner = egg::Runner::new().with_egraph(egraph).run(&rws);
```

```
"(concat
  (elementwise-add
   (bsg-systolic-array 16 16
    (slice v-32 0 16)
    (slice t-32-32 0 16 0 16)
   )
   (bsg-systolic-array 16 16
    (slice v-32 16 32)
    (slice t-32-32 16 32 0 16)
   )
  )
  (elementwise-add
   (bsg-systolic-array 16 16
    (slice v-32 0 16)
    (slice t-32-32 0 16 16 32)
   )
   (bsg-systolic-array 16 16
    (slice v-32 16 32)
    (slice t-32-32 16 32 16 32)
   )
  )
  0)"
.parse::<Pattern<_>>()
.unwrap()
.search_eclass(&runner.egraph, id)
.expect("Did not find expected program");
```

```
"(bsg-systolic-array 32 32 v-32 t-32-32)"
    .parse::<Pattern<_>>()
    .unwrap()
    .search_eclass(&runner.egraph, id)
    .expect("Did not find expected program");
```

```
"(concat
  (bsg-systolic-array 32 16
   v-32
   (slice t-32-32 0 32 0 16)
  )
  (bsg-systolic-array 32 16
   v-32
   (slice t-32-32 0 32 16 32)
  )
  0)"
.parse::<Pattern<_>>()
.unwrap()
.search_eclass(&runner.egraph, id)
.expect("Did not find expected program");
```

```
"(elementwise-add
  (bsg-systolic-array 16 32
   (slice v-32 0 16)
   (slice t-32-32 0 16 0 32)
  )
  (bsg-systolic-array 16 32
   (slice v-32 16 32)
   (slice t-32-32 16 32 0 32)
  )
 )"
.parse::<Pattern<_>>()
.unwrap()
.search_eclass(&runner.egraph, id)
.expect("Did not find expected program");
```

```
"(concat
  (bsg-systolic-array 32 16
   v-32
   (slice t-32-32 0 32 0 16)
  )
  (elementwise-add
   (bsg-systolic-array 16 16
    (slice v-32 0 16)
    (slice t-32-32 0 16 16 32)
   )
   (bsg-systolic-array 16 16
    (slice v-32 16 32)
    (slice t-32-32 16 32 16 32)
   )
  )
  0)"
.parse::<Pattern<_>>()
.unwrap()
.search_eclass(&runner.egraph, id)
.expect("Did not find expected program");
```

Wrapping Up

Wrapping Up

- Initial success of this first experiment is promising!

Wrapping Up

- Initial success of this first experiment is promising!
- Many questions left to be answered

Wrapping Up

- Initial success of this first experiment is promising!
- Many questions left to be answered
  - Will this representation of tensor programs continue to work?

Wrapping Up

- Initial success of this first experiment is promising!
- Many questions left to be answered
  - Will this representation of tensor programs continue to work?
  - How do we pull a “compiler” out of an egraph?

Wrapping Up

- Initial success of this first experiment is promising!
- Many questions left to be answered
  - Will this representation of tensor programs continue to work?
  - How do we pull a “compiler” out of an egraph?
  - How to implement a complex, multi-objective extraction process?

Wrapping Up

- Initial success of this first experiment is promising!
- Many questions left to be answered
  - Will this representation of tensor programs continue to work?
  - How do we pull a “compiler” out of an egraph?
  - How to implement a complex, multi-objective extraction process?
  - How to represent hardware sharing?





PLSE

sampl

sampa

PAUL G. ALLEN SCHOOL

W

Thank you!

https://github.com/gussmith23/glenside

https://justg.us