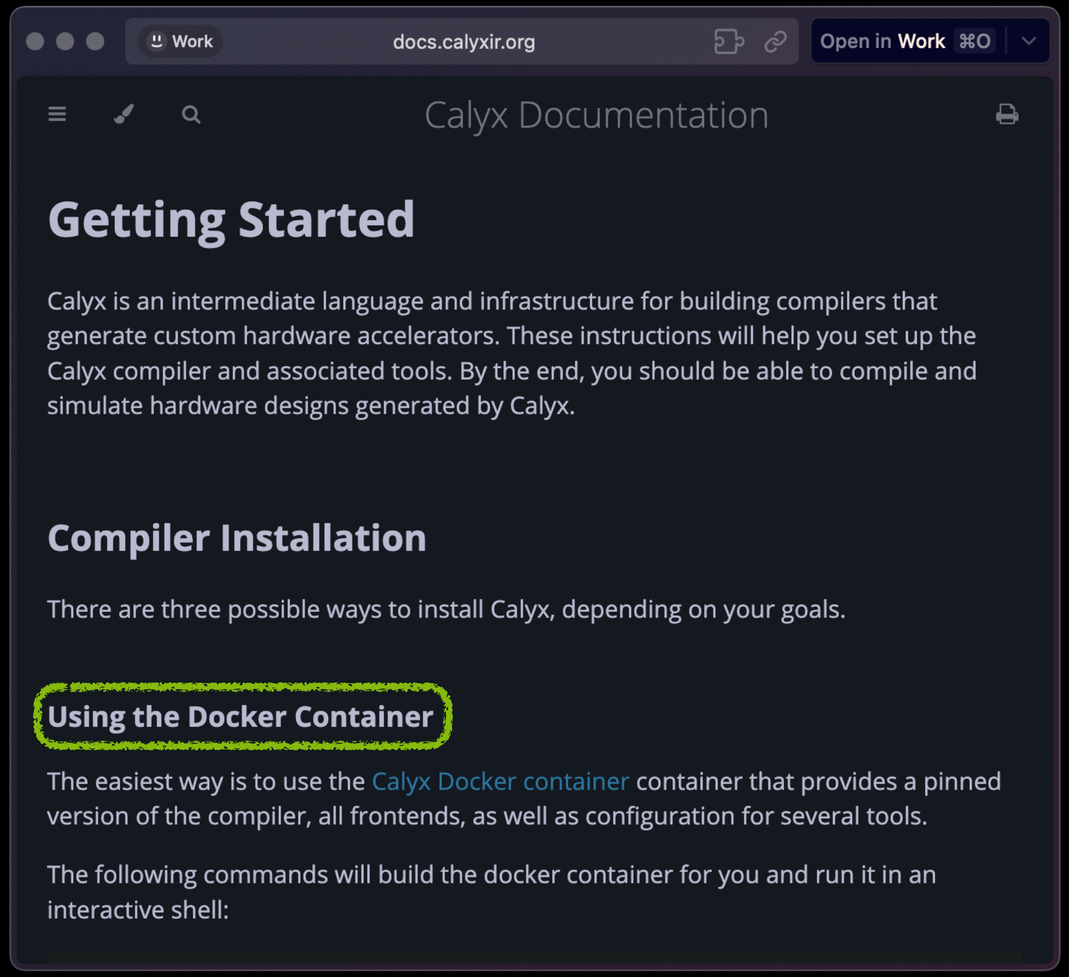
Calyx Tutorial

Computer **A**rchitecture and **Pr**ogramming **A**bstractions

capra.cs.cornell.edu

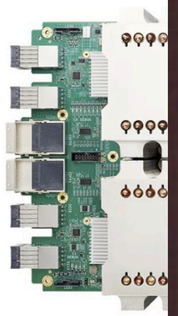
- Install Docker
- Get the Calyx Image
- Run sanity checks

(These will take some time!)

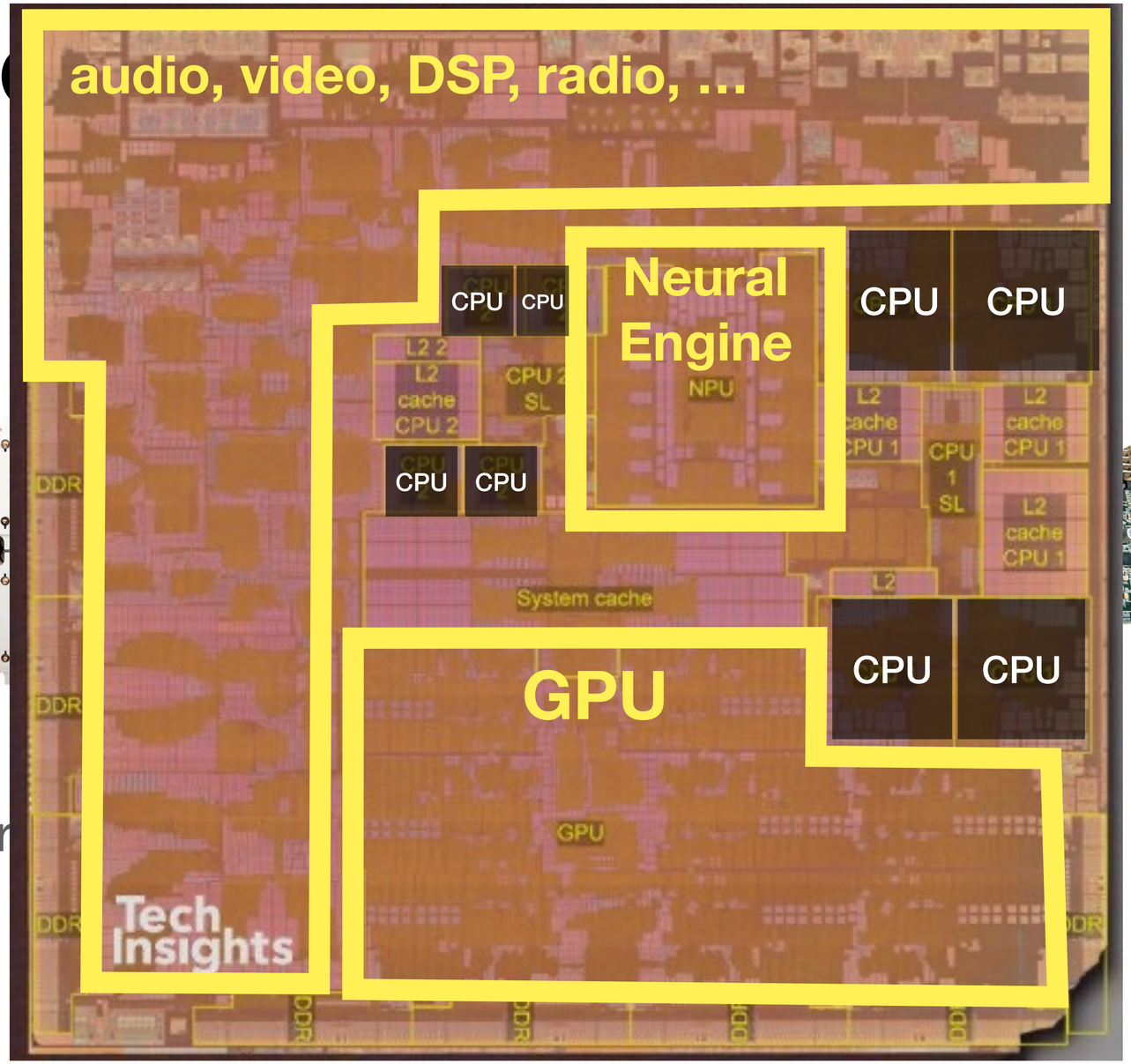


docs.calyxir.org

Har



Google
Tensor Pr



Apple
M1

Microsoft
Catapult

Hardware description languages (HDLs)

VHDL


```
assign a = b + c
```

add integers

Hardware description languages (HDLs)

VHDL


```
module seq_mult (p, rdy, clk, reset, a, b);
   input clk, reset;
   input [7:0] a, b;
   output [15:0] p;
   output rdy;

   reg [15:0] p;
   reg [15:0] multiplier;
   reg [15:0] multiplicand;
   reg rdy;
   reg [4:0] ctr;

always @(posedge clk or posedge reset) begin
    if (reset)
    begin
      rdy <= 0;
      p <= 0;
      ctr <= 0;
      multiplier <= {{8{a[7]}}, a};
      multiplicand <= {{8{b[7]}}, b};
    end
    else
    begin
      if(ctr < 16)
          begin
          if(multiplier[ctr]==1)
              begin
              multiplicand = multiplicand<<ctr;
              p <= p + multiplicand;
              end
          ctr <= ctr+1;
          end
       else
           begin
           rdy <= 1;
           end
    end
  end

endmodule
```

multiply integers

Hardware description langu

SystemVerilog

VHDL

CHISEL

PyMTL

```
        addRecFN_sub(
            control, 1'b1, recA, recB, roundingMode, recOut,
exceptionFlags);
    /*------------------------------------------------------------------------
--
    *------------------------------------------------------------------------
-*/
    wire sameOut;
    sameRecFN#(expWidth, sigWidth) sameRecFN(recOut, recExpectOut, sameOut);
    /*------------------------------------------------------------------------
--
    *------------------------------------------------------------------------
-*/
    integer errorCount, count, partialCount;
    initial begin
        /*--------------------------------------------------------------------
--
        *--------------------------------------------------------------------
-*/
        $fwrite('h80000002, "Testing 'addRecF%0d_sub'", formatWidth);
        if ($fscanf('h80000000, "%h %h", control, roundingMode) < 2) begin
            $fdisplay('h80000002, ".\n--> Invalid te
            `finish_fail;
        end
        $fdisplay(
            'h80000002,
            ", control %H, rounding mode %0d:",
            control,
            roundingMode
        );
        /*--------------------------------------------
--
        *--------------------------------------------
-*/
        errorCount = 0;
        count = 0;
        partialCount = 0;
        begin :TestLoop
            while (
                $fscanf(
                    'h80000000,
                    "%h %h %h %h",
                    a,
                    b,
                    expectOut,
                    expectExceptionFlags
                ) == 4
            ) begin
                #1;
                partialCount = partialCount + 1;
                if (partialCount == 10000) begin
                    count = count + 10000;
                    $fdisplay('h80000002, "%0d...",
                    partialCount = 0;
                end
                if (
                    !sameOut || (exceptionFlags !==
                ) begin
                    if (errorCount == 0) begin
                        $display(
            "Errors found in 'addRecF%0d_sub', contr
%0d:",
                            formatWidth,
                            control,
                            roundingMode
                        );
                    end
                    $write("%H %H", recA, recB);
                    if (formatWidth > 64) begin
                        $write("\n\t");
                    end else begin
                        $write("  ");
                    end
                    $write("=> %H %H", recOut, exceptionFlags);
                    if (formatWidth > 32) begin
                        $write("\n\t");
                    end else begin
                        $write("  ");
                    end
                    $display(
                        "expected %H %H", recExpectOut
```

Not Secure — jhauser.us

Berkeley HardFloat Release 1: Verilog Modules

John R. Hauser
2019 July 29

Contents

1. Introduction
2. Limitations
3. Acknowledgments and License
4. HardFloat Package Directory Structure
5. Floating-Point Representations
   - 5.1. Standard Formats
   - 5.2. Recoded Formats
   - 5.3. Raw Deconstructions
6. Common Control and Mode Inputs
   - 6.1. Control Input
   - 6.2. Rounding Mode
7. Exception Results
8. Specialization
   - 8.1. Width and Default Value for the Control Input
   - 8.2. Integer Results on Exceptions
   - 8.3. NaN Results
9. Main Modules
   - 9.1. Conversions Between Standard and Recoded Floating-Point (`fNToRecFN`, `recFNToFN`)
   - 9.2. Conversions from Integer (`iNToRecFN`, `iNToRawFN`)
   - 9.3. Conversions to Integer (`recFNToIN`)
   - 9.4. Conversions Between Formats (`recFNToRecFN`)
   - 9.5. Addition and Subtraction (`addRecFN`, `addRecFNToRaw`)
   - 9.6. Multiplication (`mulRecFN`, `mulRecFNToRaw`, `mulRecFNToFullRaw`)
   - 9.7. Fused Multiply-Add (`mulAddRecFN`, `mulAddRecFNToRaw`)
   - 9.8. Division and Square Root (`divSqrtRecFN_small`, `divSqrtRecFNToRaw_small`)
   - 9.9. Comparisons (`compareRecFN`)
10. Common Submodules
    - 10.1. `isSigNaNRecFN`

add floating-point numbers

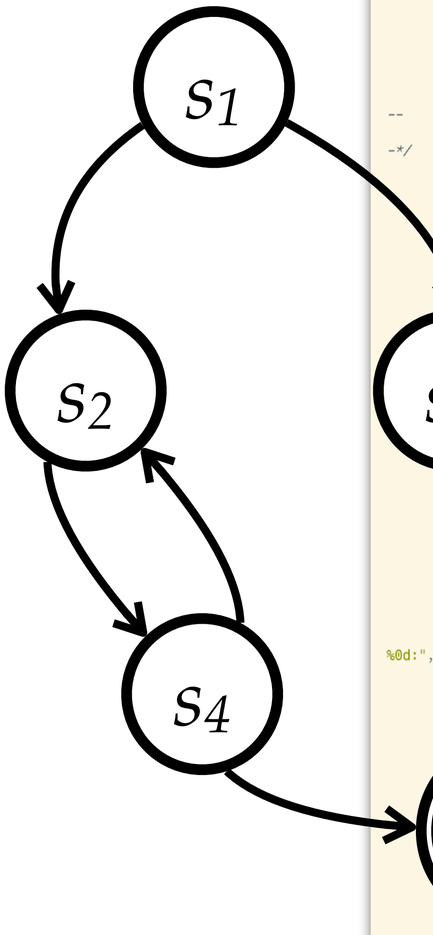
Hardware description langu

SystemVerilog

VHDL

CHISEL

PyMTL




```
        addRecFN_sub(
            control, 1'b1, recA, recB, roundingMode, recOut,
exceptionFlags);
    /*------------------------------------------------------------------------
--
    *------------------------------------------------------------------------
-*/
    wire sameOut;
    sameRecFN#(expWidth, sigWidth) sameRecFN(recOut, recExpectOut, sameOut);
    /*------------------------------------------------------------------------
--
    *------------------------------------------------------------------------
-*/
    integer errorCount, count, partialCount;
    initial begin
        /*--------------------------------------------------------------------
--
        *--------------------------------------------------------------------
-*/
        $fwrite('h80000002, "Testing 'addRecF%0d_sub'", formatWidth);
        if ($fscanf('h80000000, "%h %h", control, roundingMode) < 2) begin
            $fdisplay('h80000002, ".\n--> Invalid te
            `finish_fail;
        end
        $fdisplay(
            'h80000002,
            ", control %H, rounding mode %0d:",
            control,
            roundingMode
        );
        /*--------------------------------------
--
        *--------------------------------------
-*/
        errorCount = 0;
        count = 0;
        partialCount = 0;
        begin :TestLoop
            while (
                $fscanf(
                    'h80000000,
                    "%h %h %h %h",
                    a,
                    b,
                    expectOut,
                    expectExceptionFlags
                == 4
            n
            ;
            artialCount = partialCount + 1;
            if (partialCount == 10000) begin
                count = count + 10000;
                $fdisplay('h80000002, "%0d...",
                partialCount = 0;
            end
            f (
                !sameOut || (exceptionFlags !==
            begin
                if (errorCount == 0) begin
                    $display(
        "Errors found in 'addRecF%0d_sub', contr
%0d:",
                        formatWidth,
                        control,
                        roundingMode
                    );
                end
                te("%H %H", recA, recB);
                rmatWidth > 64) begin
                ite("\n\t");
                 begin
                te("  ");

                "=> %H %H", recOut, exceptionFlags);
                rmatWidth > 32) begin
                    $write("\n\t");
                end else begin
                    $write("  ");
                end
                $display(
                    "expected %H %H", recExpectOut
```

Not Secure — jhauser.us

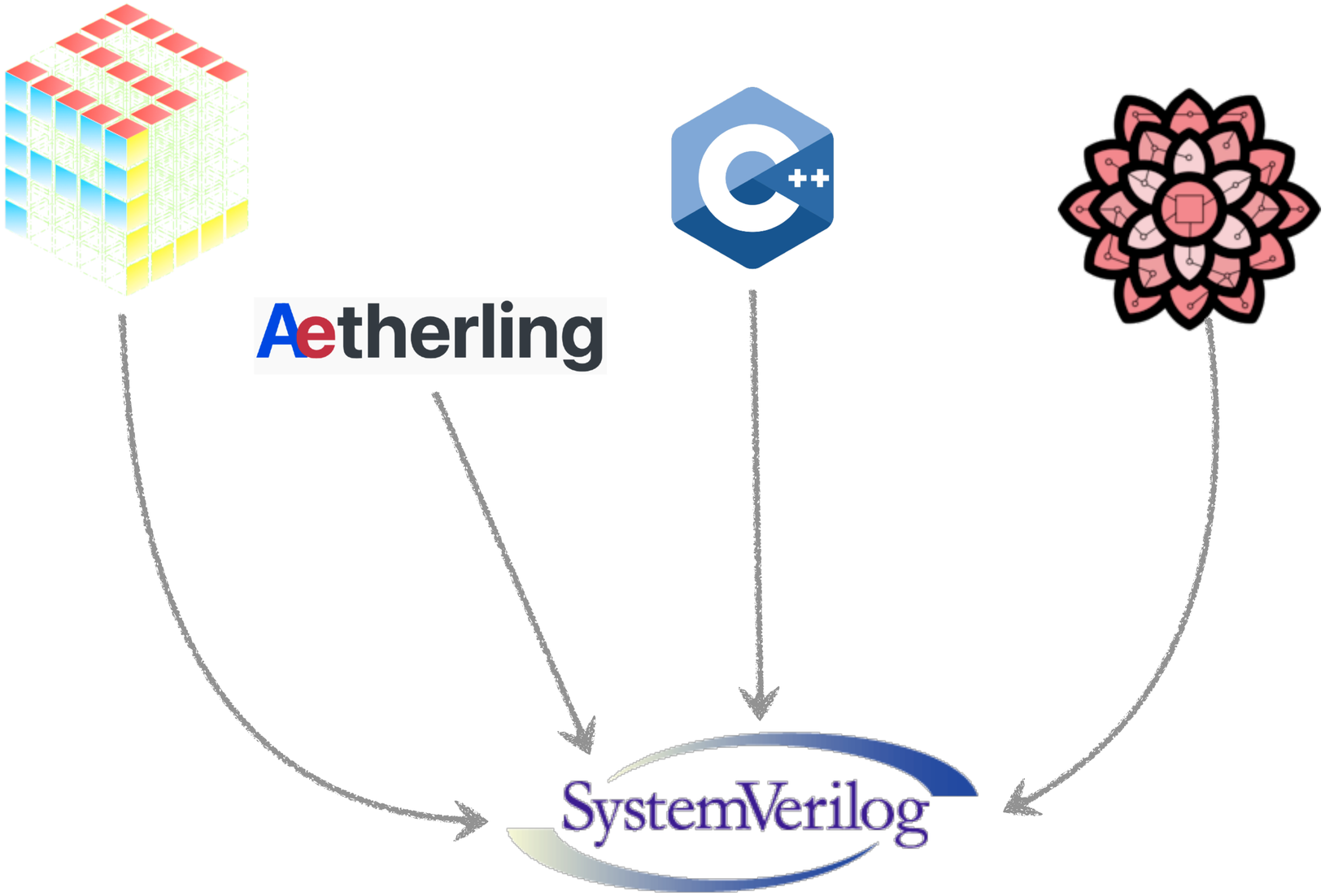
Berkeley HardFloat Release 1: Verilog Modules

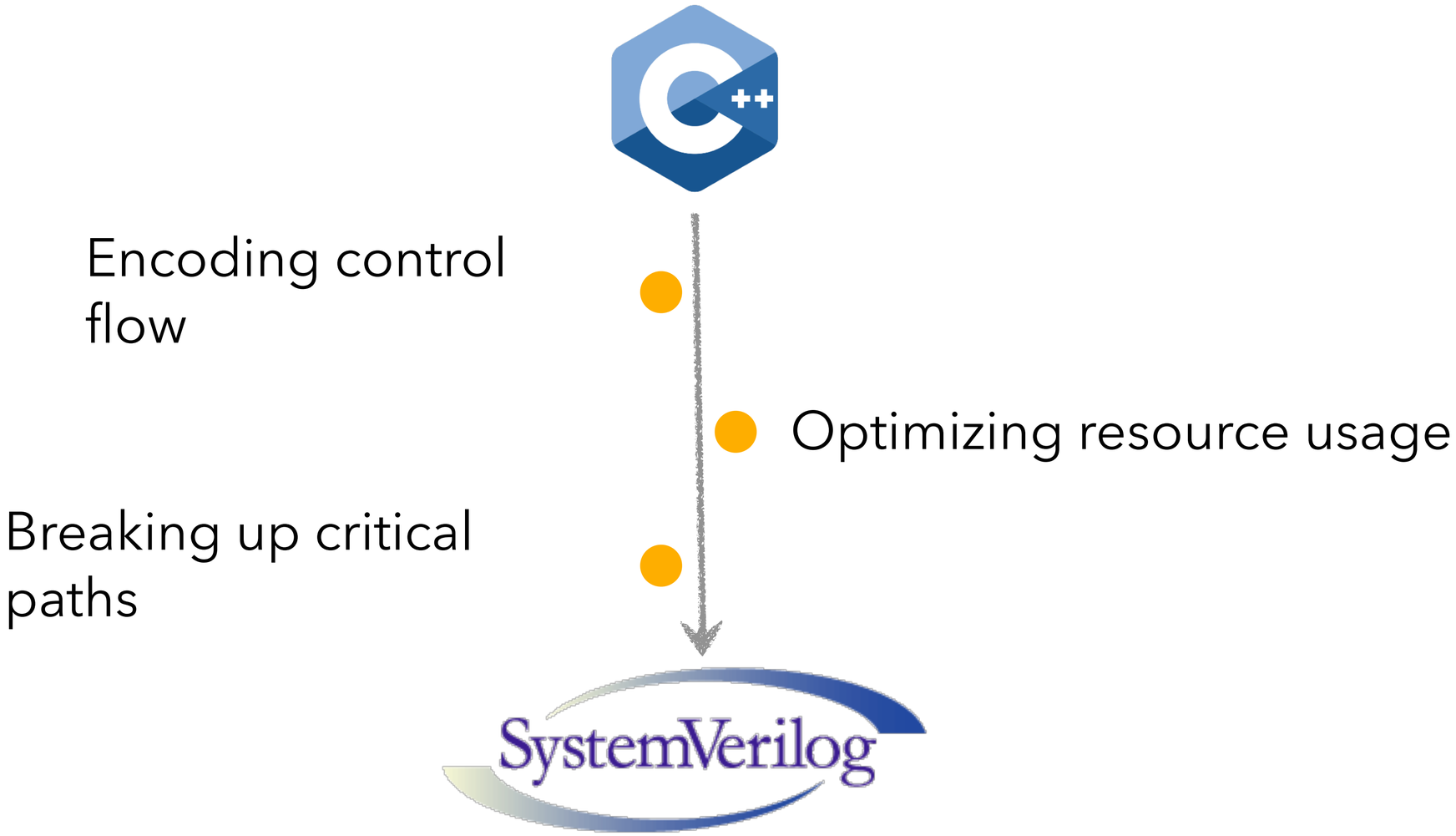
John R. Hauser
2019 July 29

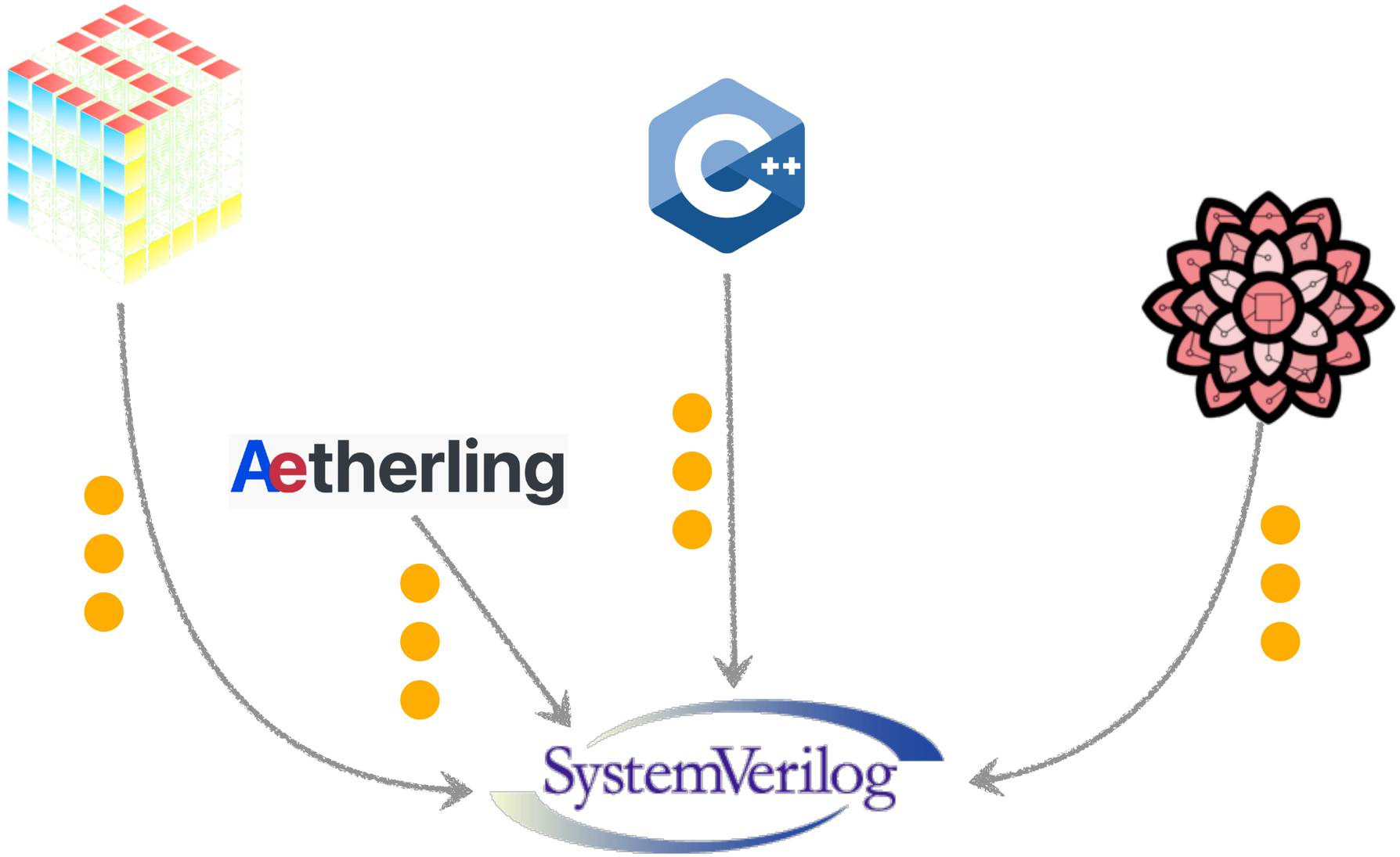
Contents

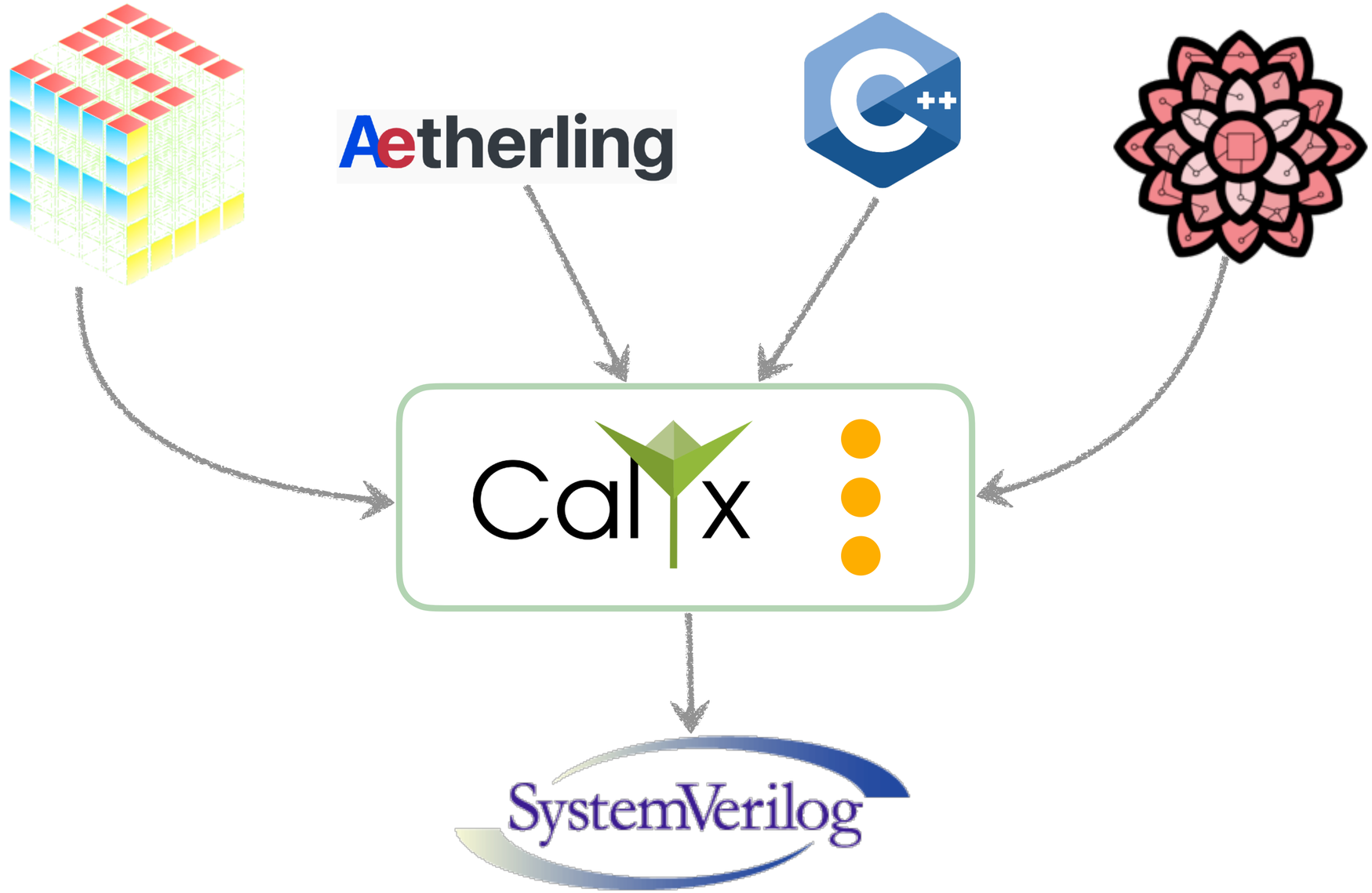
1. Introduction
2. Limitations
3. Acknowledgments and License
4. HardFloat Package Directory Structure
5. Floating-Point Representations
    5.1. Standard Formats
    5.2. Recoded Formats
    5.3. Raw Deconstructions
6. Common Control and Mode Inputs
    6.1. Control Input
    6.2. Rounding Mode
7. Exception Results
8. Specialization
    8.1. Width and Default Value for the Control Input
    8.2. Integer Results on Exceptions
    8.3. NaN Results
9. Main Modules
    9.1. Conversions Between Standard and Recoded Floating-Point (`fNToRecFN`, `recFNToFN`)
    9.2. Conversions from Integer (`iNToRecFN`, `iNToRawFN`)
    9.3. Conversions to Integer (`recFNToIN`)
    9.4. Conversions Between Formats (`recFNToRecFN`)
    9.5. Addition and Subtraction (`addRecFN`, `addRecFNToRaw`)
    9.6. Multiplication (`mulRecFN`, `mulRecFNToRaw`, `mulRecFNToFullRaw`)
    9.7. Fused Multiply-Add (`mulAddRecFN`, `mulAddRecFNToRaw`)
    9.8. Division and Square Root (`divSqrtRecFN_small`, `divSqrtRecFNToRaw_small`)
    9.9. Comparisons (`compareRecFN`)
10. Common Submodules
    10.1. `isSigNaNRecFN`

add and multiply several floating-point numbers


Aetherling
SystemVerilog


Encoding control flow
Optimizing resource usage
Breaking up critical paths
SystemVerilog


Aetherling
SystemVerilog


Aetherling
Calyx
SystemVerilog

Captures control flow

Missing structure and time

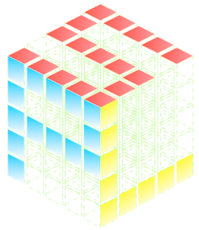
Captures structure and time

Missing control flow

High-level control flow

Low-level structure

Image by Eucalyp



High Level Descriptions **Hardware Descriptions**


```
int a, b, c, d;
if (a % 10)
   x = a * b
else
   y = c * d
```

```
int a, b, c, d;
if (a % 10)
   x = a * b
else
   y = c * d
```


```
component do_add(inputs) -> (outputs) {
  cells {}
  wires {}
  control {}
}
```

Components encapsulate **hardware structure** and **control flow**

```
int a, b, c, d;
if (a % 10)
   x = a * b
else
   y = c * d
```


```
component do_add(a: 32, …) -> (out: 32) {
  cells {}
  wires {}
  control {}
}
```

Components define **sized** input and output ports

```
int a, b, c, d;
if (a % 10)
   x = a * b
else
   y = c * d
```

Calyx

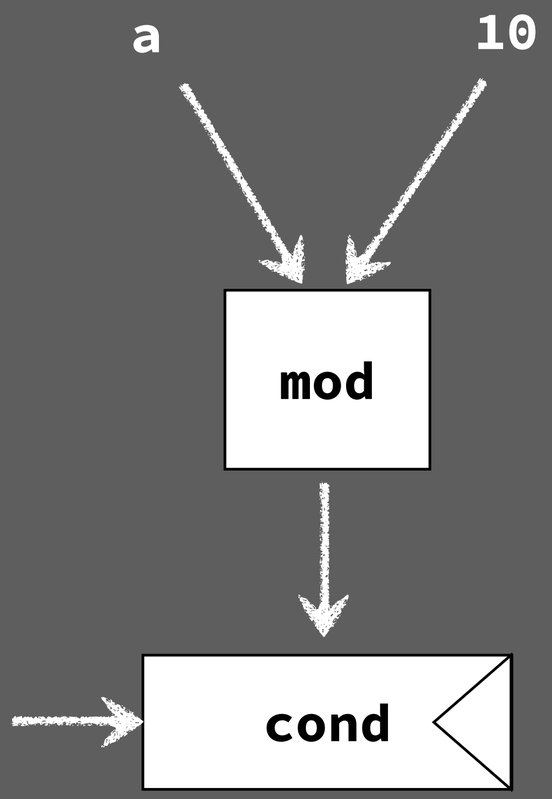
```
component do_add(a: 32, …) -> (out: 32) {
  cells {
    mod = std_mod(32);
    cond = std_reg(1);
  }
  wires {}
  control {}
}
```

Cells define **sub-components** required by a given component

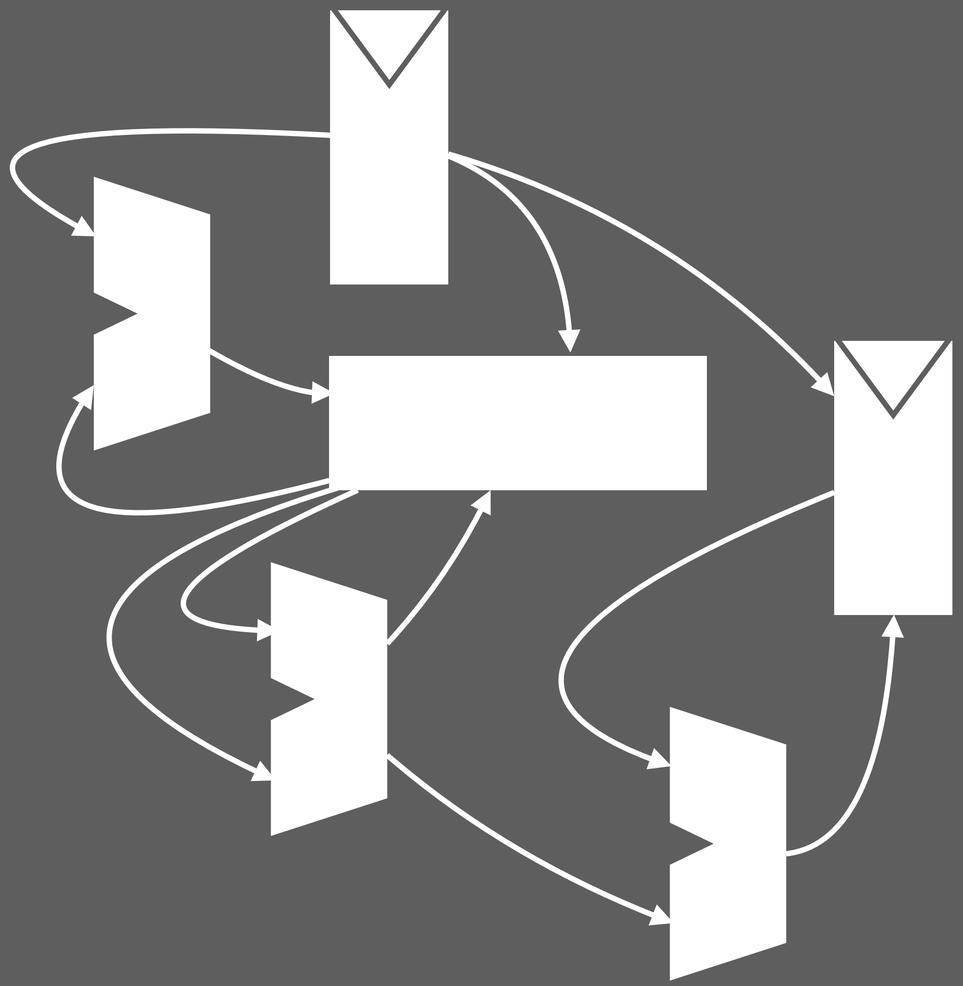
```
int a, b, c, d;
if (a % 10)
   x = a * b
else
   y = c * d
```


```
component do_add(a: 32, …) -> (out: 32) {
  cells { … }
  wires {
    mod.right = 32'd10;
    mod.left = count;
    cond.in = mod.out;
    cond.write_en = 1'd1;
  }
  control {}
}
```

Wires defines **connections** between submodules

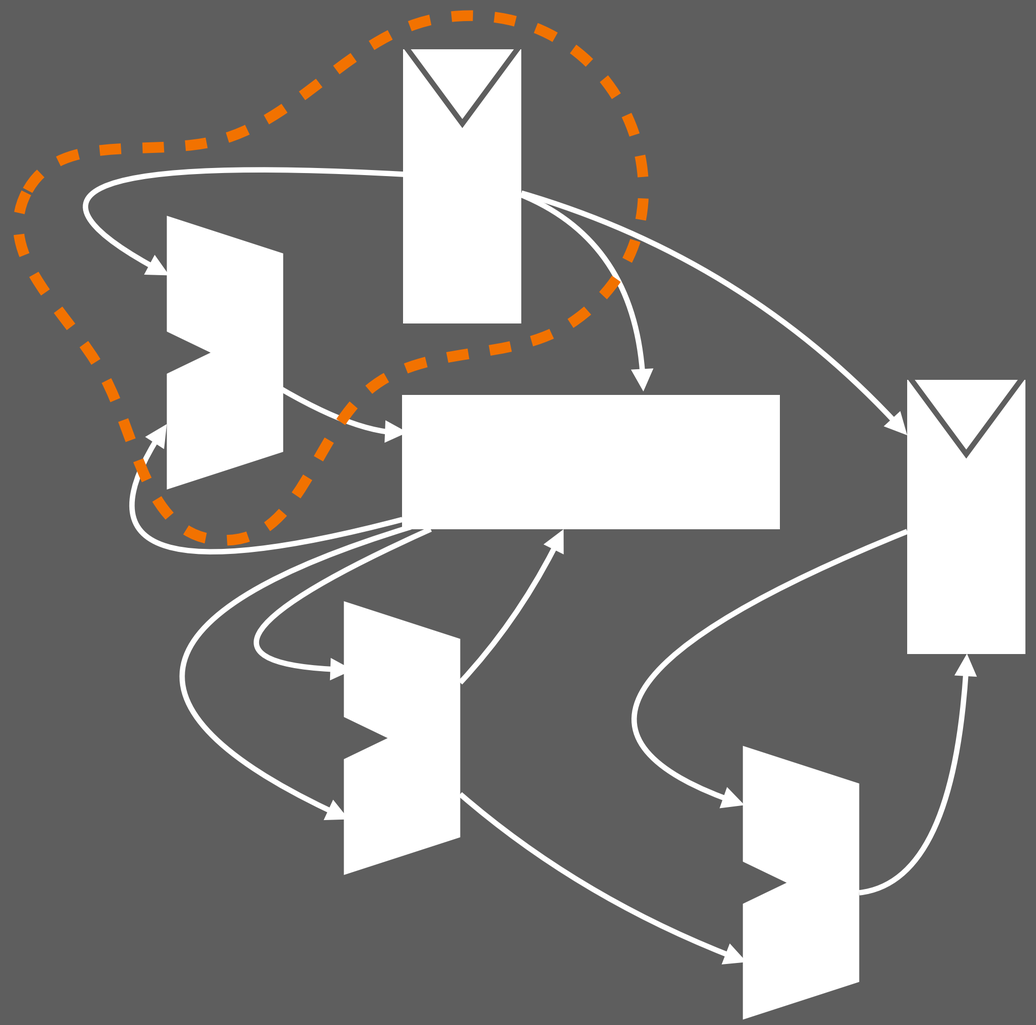



```
component do_add(a: 32, …) -> (out: 32) {
  cells {
    mod = std_mod(32);
    cond = std_reg(1);
  }
  wires {
    mod.right = 32'd10;
    mod.left = count;
    cond.in = mod.out;
    cond.write_en = 1'd1;
  }
  control {}
}
```



Calyx

```
component do_add(a: 32, …) -> (out: 32) {
  cells {
    mod = std_mod(32);
    cond = std_reg(1);
  }
  wires {
    mod.right = 32'd10;
    mod.left = count;
    cond.in = mod.out;
    cond.write_en = 1'd1;
  }
  control {}
}
```



Calyx

```
component do_add(a: 32, …) -> (out: 32) {
  cells { … }
  wires {
    group do_cond {
      mod.right = 32'd10;
      mod.left = count;
      cond.in = mod.out;
      cond.write_en = 1'd1;
      do_cond[done] = cond.done;
    }
  }
  control {}
}
```

```
int a, b, c, d;
if (a % 10)
  x = a * b
else
  y = c * d
```

Groups describe the
"**instructions**" for the accelerator

Calyx

```
component do_add(a: 32, …) -> (out: 32) {
  cells { … }
  wires {
    group do_cond {
      mod.right = 32'd10;
      mod.left = a;
      cond.in = mod.out;
      cond.write_en = 1'd1;
      do_cond[done] = cond.done;
    }
  }
  control {}
}
```

```
int a, b, c, d;
if (a % 10)
  x = a * b
else
  y = c * d
```

Control defines the
execution schedule

Calyx

```
component do_add(a: 32, …) -> (out: 32) {
  cells { … }
  wires {
    group do_cond {
      mod.right = 32'd10;
      mod.left = a;
      cond.in = mod.out;
      cond.write_en = 1'd1;
      do_cond[done] = cond.done;
    }
  }
  control {
    seq { do_cond; do_cond; do_cond }
  }
}
```

```
int a, b, c, d;
if (a % 10)
  x = a * b
else
  y = c * d
```

```
component do_add(a: 32, …) -> (out: 32) {
  cells { … }
  wires {
    group do_cond { … }
    group upd_x { … }
    group upd_y { … }
  }
  control {}
}
```

```
int a, b, c, d;
if (a % 10)
  x = a * b
else
  y = c * d
```

```
component do_add(a: 32, …) -> (out: 32) {
  cells { … }
  wires {
    group do_cond { … }
    group upd_x { … }
    group upd_y { … }
  }
  control {
    seq {
     do_cond;
     if cond.out { upd_x } else { upd_y }
  }
}
```

```
int a, b, c, d;
if (a % 10)
   x = a * b
else
   y = c * d
```

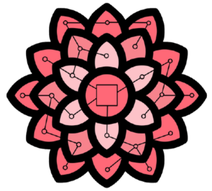
- Generate **groups** to **encode computation**
- Use **control** to **schedule execution**

Calyx

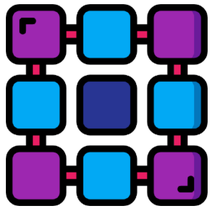
```
component do_add(a: 32, …) -> (out: 32) {
  cells { … }
  wires {
    group do_cond { … }
    group upd_x { … }
    group upd_y { … }
  }
  control {
    seq {
     do_cond;
     if cond.out { upd_x } else { upd_y }
  }
}
```

LLVM
CIRCT

TVM



Dahlia



Systolic
Arrays

And others …

Calyx

- Modular, **pass-based** compiler
- **40** optimization passes
- **15** analyses

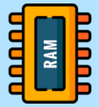
Software-like
Debugging

ASPLOS '23

Pangenomics
Accelerator

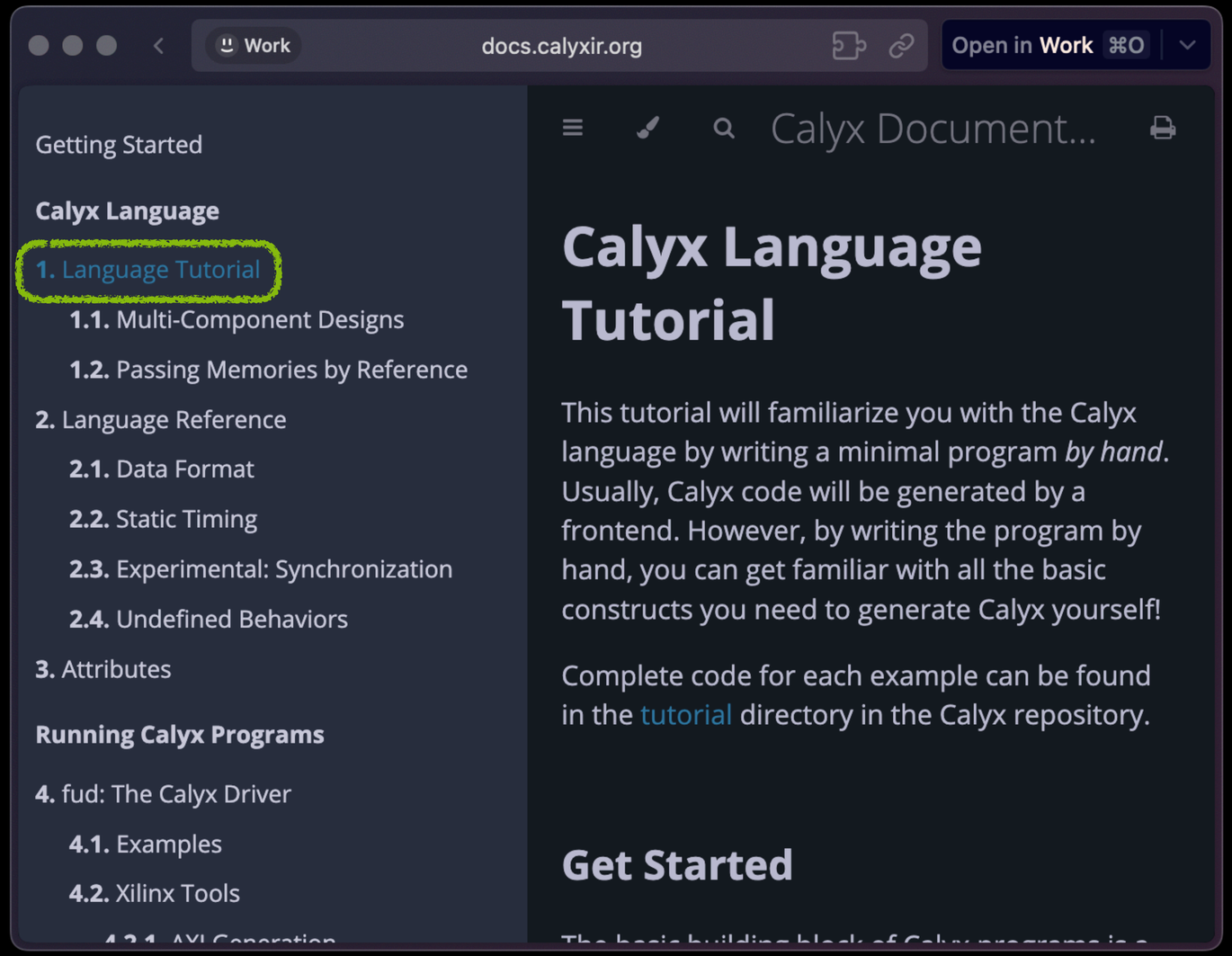
FPGA Execution

Automatic AXI Generation



Write your first Calyx program!

P.S. Remember to install,
setup your favorite editor



docs.calyxir.org

`fud`, the Calyx driver

MrXL ——

```
fud e squares
```

```
import "primitives/core.futil";
import "primitives/binary_operators.futil";
component main() → () {
  cells {
    @external avec_b0 = std_mem_d1(32, 2, 32);
    @external avec_b1 = std_mem_d1(32, 2, 32);
    @external squares_b0 = std_mem_d1(32, 2, 32);
    @external squares_b1 = std_mem_d1(32, 2, 32);
    idx_b0_0 = std_reg(32);
    incr_b0_0 = std_add(32);
    lt_b0_0 = std_lt(32);
    mul_b0_0 = std_mult_pipe(32);
    idx_b1_0 = std_reg(32);
    incr_b1_0 = std_add(32);
    lt_b1_0 = std_lt(32);
    mul_b1_0 = std_mult_pipe(32);
  }
  wires {
    group incr_idx_b0_0 {
      incr_b0_0.left = idx_b0_0.out;
      incr_b0_0.right = 32'd1;
      idx_b0_0.in = incr_b0_0.out;
      idx_b0_0.write_en = 1'd1;
      incr_idx_b0_0[done] = idx_b0_0.done;
    }
    comb group cond_b0_0 {
      lt_b0_0.left = idx_b0_0.out;
```

```
calyx
```

`fud`, the Calyx driver

MrXL —

```
fud e squar
```

```
module main(
  input logic go,
  input logic clk,
  input logic reset,
  output logic done
);
// COMPONENT START: main
string DATA;
int CODE;
initial begin
    CODE = $value$plusargs("DATA=%s", DATA);
    $display("DATA (path to meminit files): %s", DATA);
    $readmemh({DATA, "/avec_b0.dat"}, avec_b0.mem);
    $readmemh({DATA, "/avec_b1.dat"}, avec_b1.mem);
    $readmemh({DATA, "/squares_b0.dat"}, squares_b0.mem);
    $readmemh({DATA, "/squares_b1.dat"}, squares_b1.mem);
end
final begin
    $writememh({DATA, "/avec_b0.out"}, avec_b0.mem);
    $writememh({DATA, "/avec_b1.out"}, avec_b1.mem);
    $writememh({DATA, "/squares_b0.out"}, squares_b0.mem);
    $writememh({DATA, "/squares_b1.out"}, squares_b1.mem);
end
logic [31:0] avec_b0_addr0;
logic [31:0] avec_b0_write_data;
logic avec_b0_write_en;
logic avec_b0_clk;
```

→ 

rilog


MrXL
mem
Calyx
SV

```
fud will perform the following steps:
  - mrxl.mktmp: Make temporary directory to store Verilator build files.
  - mrxl.set_mrxl_prog: Set stages.mrxl.prog as `input`
  - mrxl.mrxl-data.get_mrxl_prog: Dynamically retrieve the value of stages.mrxl.prog
  - mrxl.mrxl-data.convert_mrxl_data_to_calyx_data: Converts MrXL input into calyx input
  - transform: transform input to String
  - mrxl.save_data: Save verilog.data in `tmpdir` and update stages.verilog.data
  - mrxl.run_mrxl: mrxl
  - calyx.run_futil: /Users/ps/Research/calyx-ref/calyx/target/debug/calyx -l /Users/ps/Research/calyx-ref/calyx -b verilog
  - transform: transform input to Path
  - verilog.mktmp: Make temporary directory to store Verilator build files.
  - verilog.get_verilog_data: Dynamically retrieve the value of stages.verilog.data
  - verilog.check_verilog_for_mem_read: Read input verilog to see if `verilog.data` needs to be set.
  - verilog.json_to_dat: Converts a `json` data format into a series of `.dat` files inside the given
            temporary directory.
  - verilog.compile_with_verilator: verilator --trace {input_path} /Users/ps/Research/calyx-ref/calyx/fud/icarus/tb.sv --binary --top-module TOP --Mdir {tmpdir_name} -fno-inline
  - verilog.simulate: Simulates compiled Verilator code.
  - verilog.output_json: Convert .dat files back into a json and extract simulated cycles from log.
  - verilog.cleanup: Cleanup Verilator build files that we no longer need.
```

```
fud e squares.mr                to dat \
--through verilo                uares.mrxl.data
```


```
{
  "cycles": 17,
  "memories": {
    "avec_b0": [
      0,
      1
    ],
    "avec_b1": [
      4,
      5
    ],
    "squares_b0": [
      0,
      1
    ],
    "squares_b1": [
      16,
      25
    ]
  }
}
```

MrXL

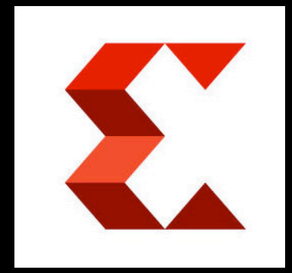
tvm

MyDSL

Calyx

SV

CIDer



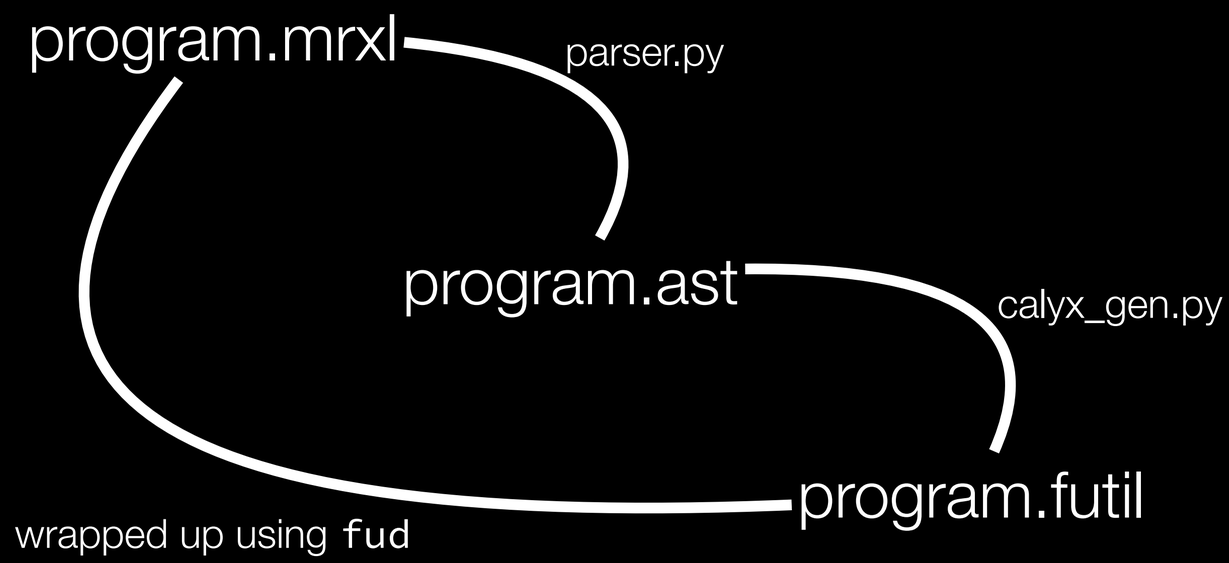
MrXL: `map-reduce` accelerator

Real frontends are not built in a day,
so let's work on a toy frontend for Calyx.

We will introduce MrXL, and then you will implement its `map` operation.

Watch your directories!
The initial commands are run from `calyx/`
Eventually you will work in `calyx/frontends/mrxl`

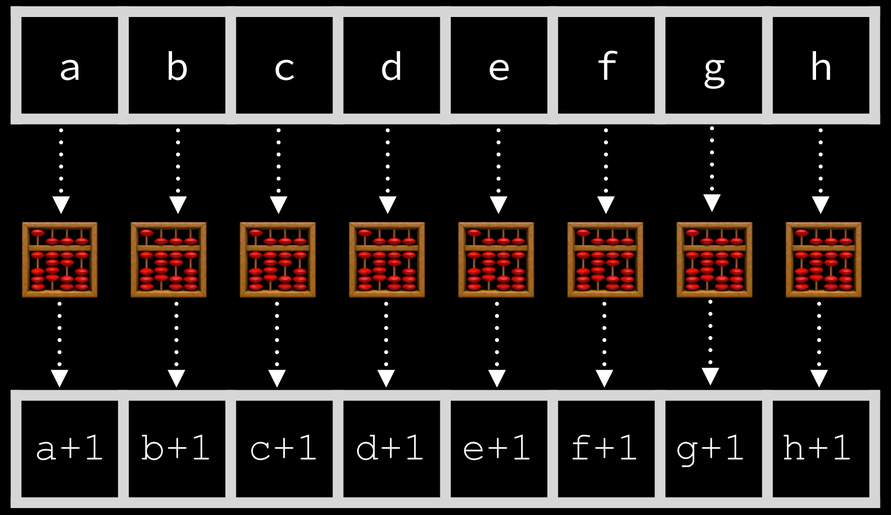
anatomy of a frontend



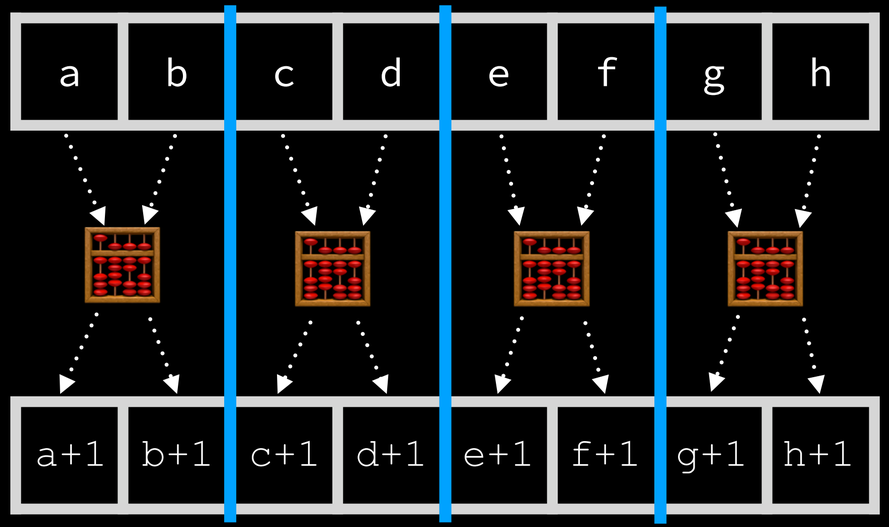
memory banking

Reads from a memory have a limitation:
one read/write per tick of the clock.

Say I give you all the compute
you could ask for:



But if we *bank* the arrays,
we really can parallelize:



give MrXL a `map` operation!

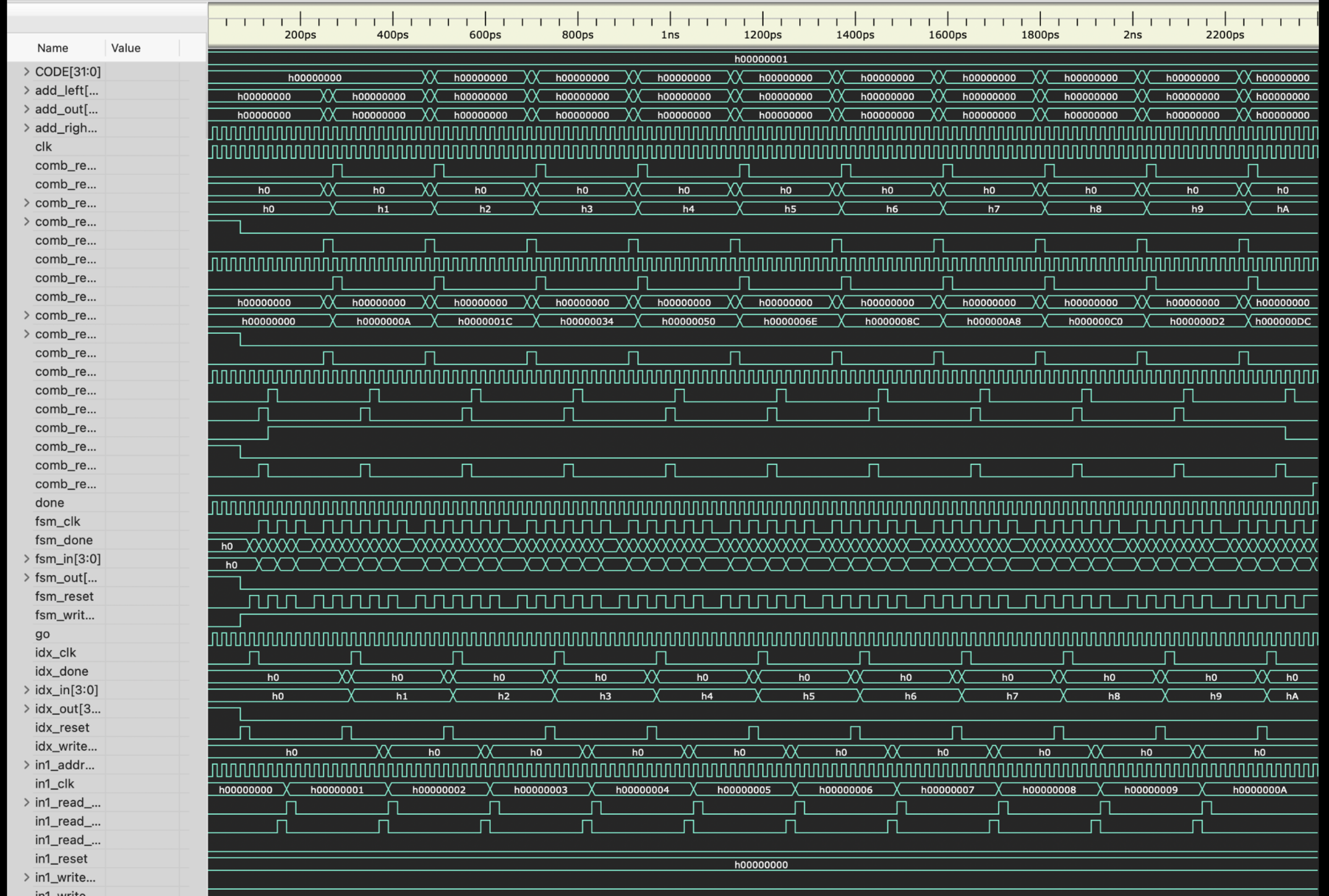
Psst: consider implementing just `add` first, end-to-end

pass this flag
to run your version

```
fud e --from mrxl test/sos.mrxl \
      --to dat --through verilog \
      -s mrxl.flags "--my-map "  \
      -s mrxl.data test/sos.mrxl.data
```

study the implementation
that’s in place

Cider


Name
Value
200ps
400ps
600ps
800ps
1ns
1200ps
1400ps
1600ps
1800ps
2ns
2200ps
CODE[31:0]
add_left[...
add_out[...
add_righ...
clk
comb_re...
done
fsm_clk
fsm_done
fsm_in[3:0]
fsm_out[...
fsm_reset
fsm_writ...
go
idx_clk
idx_done
idx_in[3:0]
idx_out[3...
idx_reset
idx_write...
in1_addr...
in1_clk
in1_read_...
in1_reset
in1_write...
h00000001
h00000000
h0
h1
h2
h3
h4
h5
h6
h7
h8
h9
hA
h0000000A
h0000001C
h00000034
h00000050
h0000006E
h0000008C
h000000A8
h000000C0
h000000D2
h000000DC
h00000001
h00000002
h00000003
h00000004
h00000005
h00000006
h00000007
h00000008
h00000009
h0000000A

Cider: Calyx Interpreter and Debugger

Provides a GDB-like debugging experience for Calyx programs

Insight: Use Calyx ***Groups*** as coarse time units

```
for i in 0..4:
    z[i] = x[i] * y[i]
```

```
read_mem
do_mul
write_mem
upd_idx
```

```
for i in 0..4:
    z[i] = x[i] * y[i]
```

Observed behavior: does not terminate

```
> watch after upd_idx with print-state \u idx_reg
    idx_reg = 1
    idx_reg = 2
    idx_reg = 3
WARN - Integer overflow, source: idx_adder
    idx_reg = 0
    idx_reg = 1
```

the bug

```
cells {
    idx_reg = reg(2);
    idx_adder = reg(2);
}
```

```
cells {
    idx_reg = reg(3);
    idx_adder = reg(3);
}
```

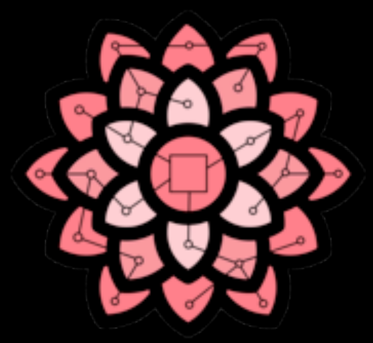
```
while idx_reg <= 3 {
    read_mem;
    do_mul;
    write_mem;
    upd_idx;
}
```

using Cider

```
fud e --from mrxl test/sos.mrxl --to interpreter-out
```

```
fud e --from mrxl test/sos.mrxl --to debugger
```

Read more: https://docs.calyxir.org/debug/cider.html

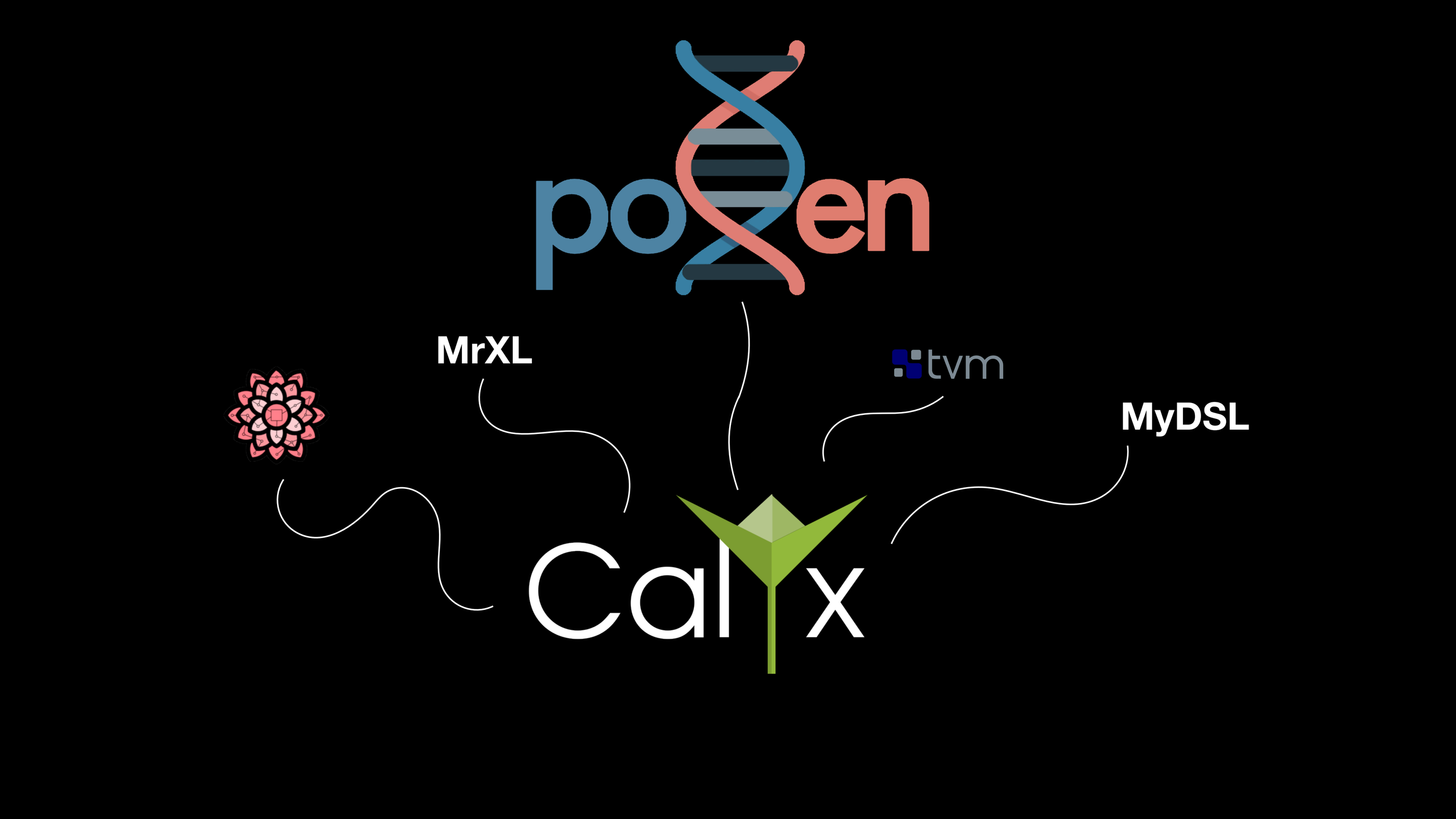


MrXL

tvm

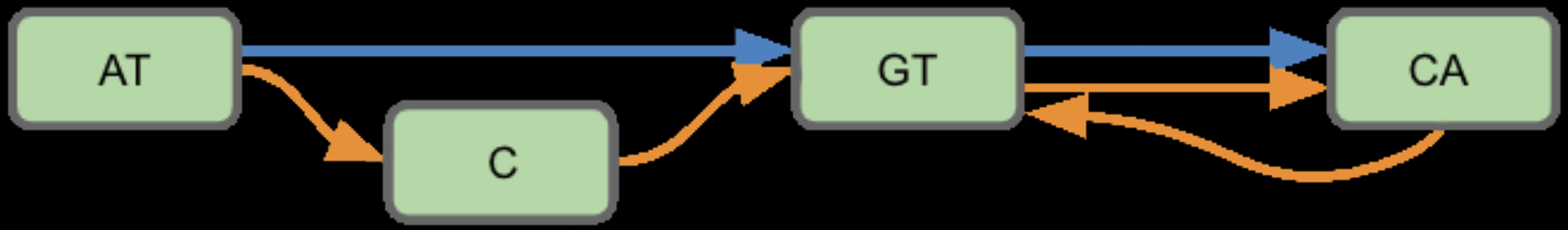
MyDSL

Calyx


poLen
MrXL
tvm
MyDSL
Calyx

Pollen, an accelerator generator for pangenomic graph queries

Pollen, an accelerator generator for pangenomic graph queries



size of a human pangenomic graph:

259,525,394 base pairs in a chromosome
481,945 nodes per person
4,643,780 nodes per pangenomic graph

size of a human pangenomic graph:

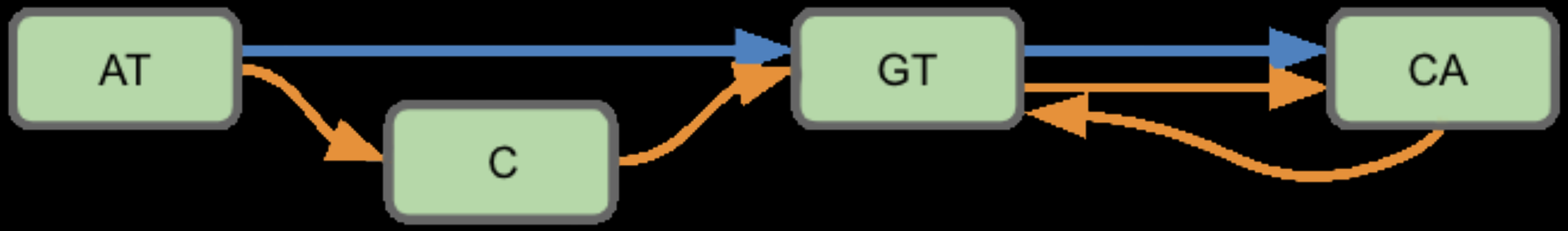
259,525,394 base pairs in a chromosome
481,945 nodes per person
4,643,780 nodes per pangenomic graph

~30GB

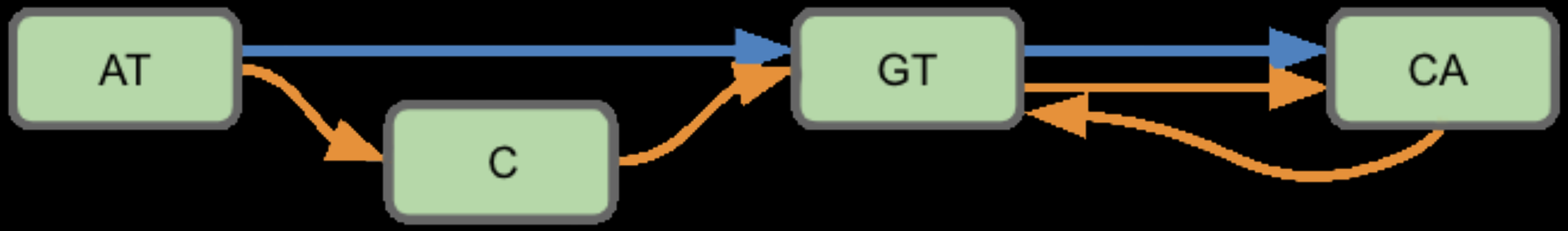
size of a human pangenomic graph:

259,525,394 base pairs in a chromosome
481,945 nodes per person
4,643,780 nodes per pangenomic graph

~30GB
Larger for efficient computations




```
out_graph g;
parset depth[int, g];
for segment in graph.segments {
  emit segment.steps.size() to depths;
}
```




```
out_graph g;
parset depth[int, g];
for segment in graph.segments {
  emit segment.steps.size() to depths;
}
```

depth.pollen

```
out_graph g;
parset depth[int, g];
for segment in graph.segments {
  emit segment.steps.size() to depths;
}
```

depth.pollen

```
out_graph g;
parset depth[int, g];
for segment in graph.segments {
   emit segment.steps.size() to depths;
}
```

exine depth depth.pollen -n 2

depth.pollen

```
out_graph g;
parset depth[int, g];
for segment in graph.segments {
  emit segment.steps.size() to depths;
}
```

exine depth depth.pollen -n 2

≅

depth.mrxl

```
output depths : int[4]
depths := map 2 (s <- graph.segments) {s.steps.size()}
```

takeaways

Domain experts with **minimal** hardware knowledge can make use of Calyx

Calyx can be a **backend** for complex DSLs

The skills you've gained generating map are **broadly applicable** to all sorts of language features

have a break,
have a Kit Kat

MrXL: extensions

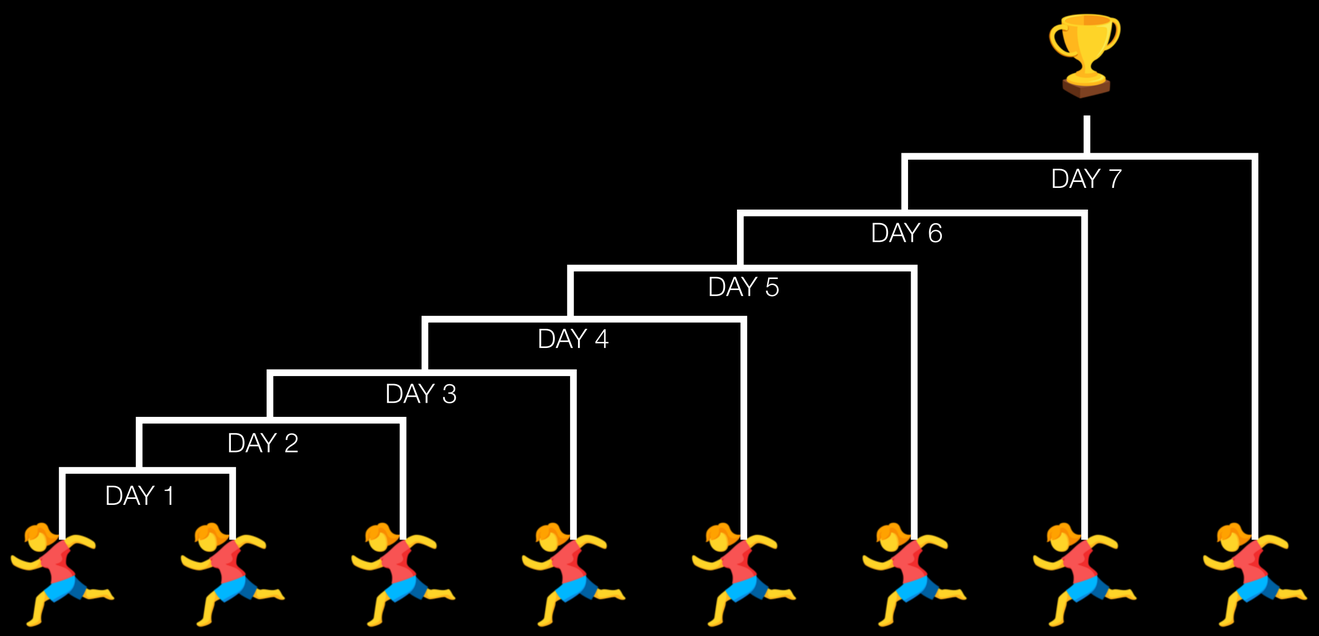
Three options:

1. Implement the reduce operation.
2. Allow the same memory to banked repeatedly.
3. Office hours!

reduction trees

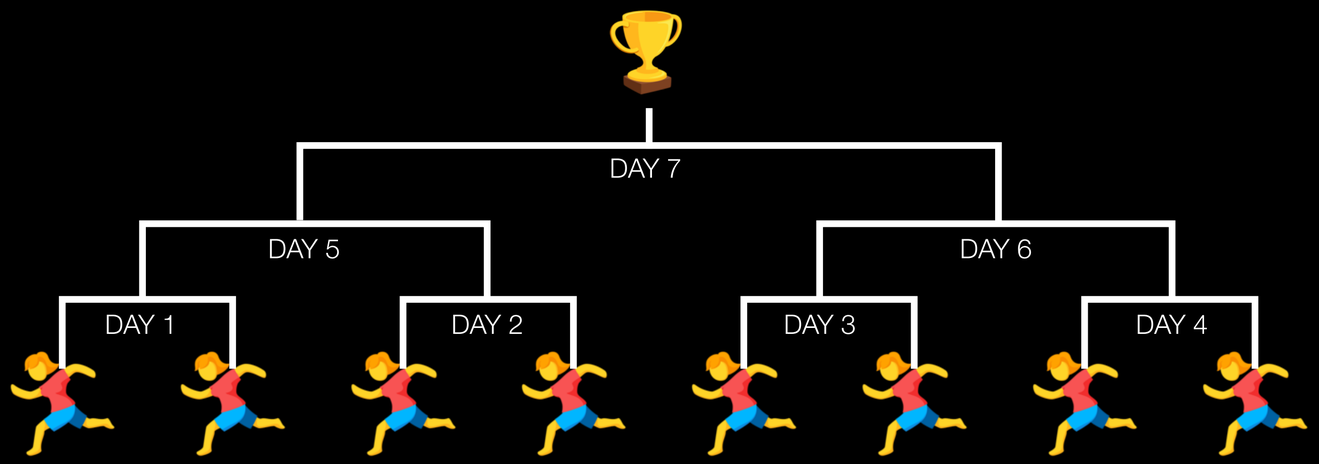
How best to run a tennis tournament?

This clearly has problems…



reduction trees

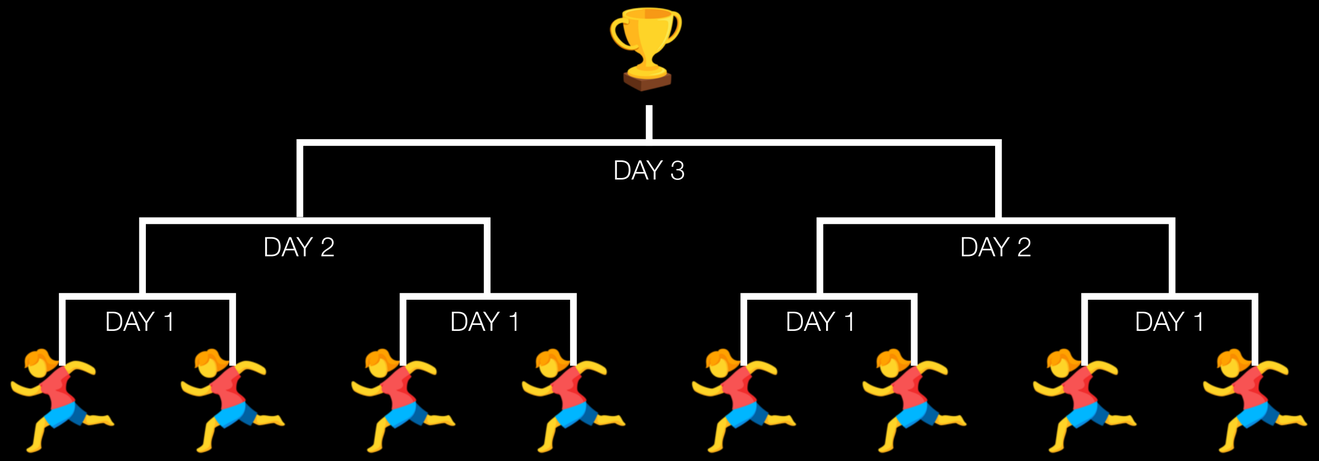
A little better!



reduction trees

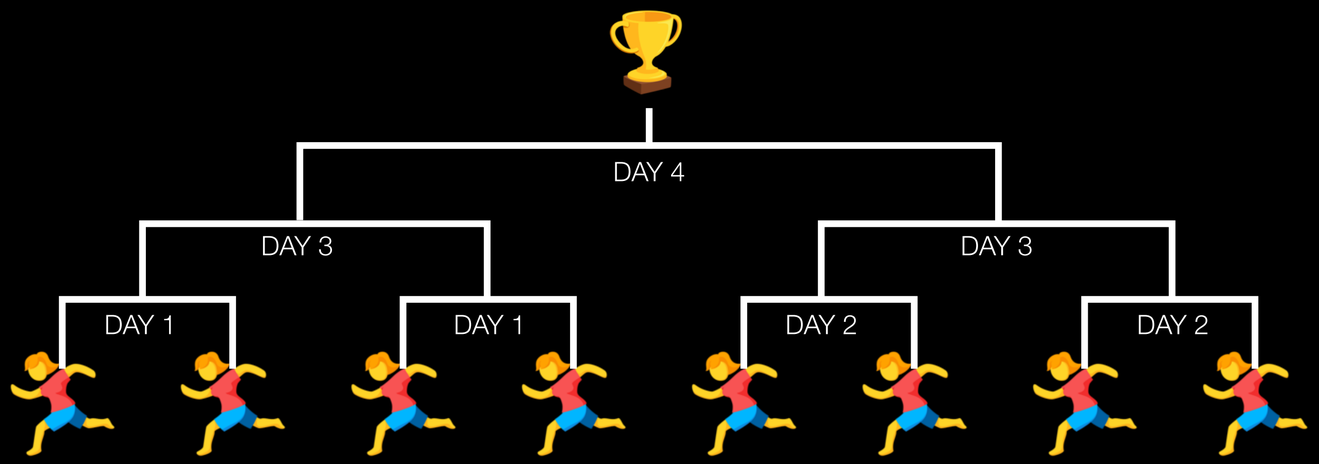
Better still…

But hang on, do we have four courts at the same time?



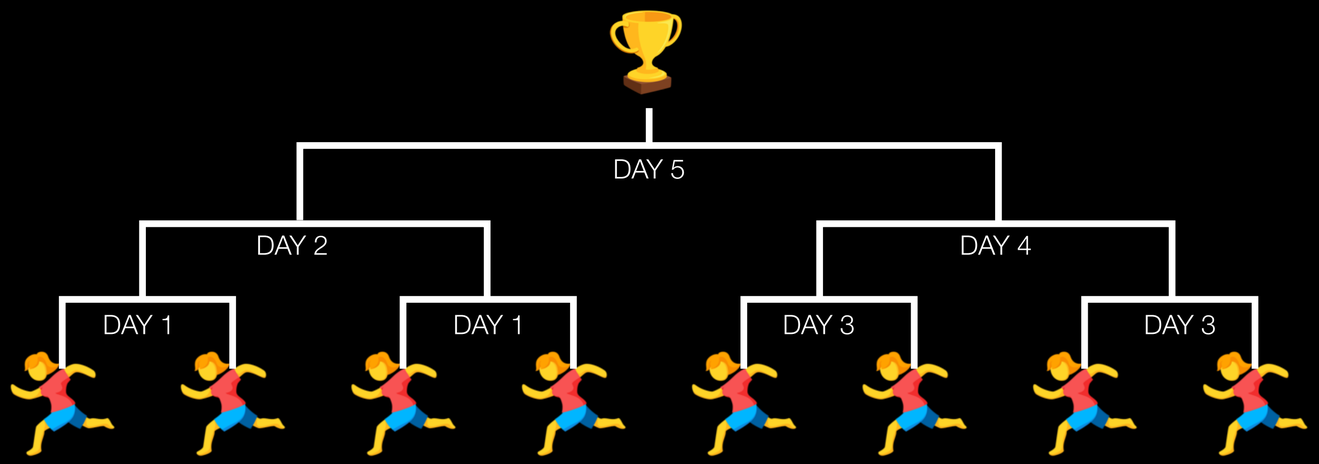
reduction trees

We can do this with two courts!



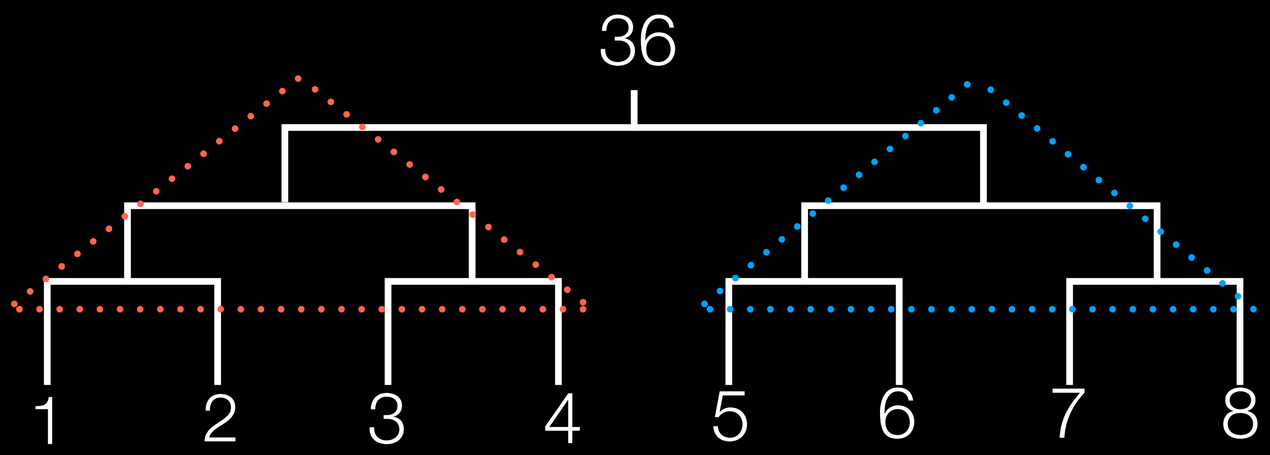
reduction trees

Ugh it’s getting expensive to transport players to and fro.
Here’s another option…



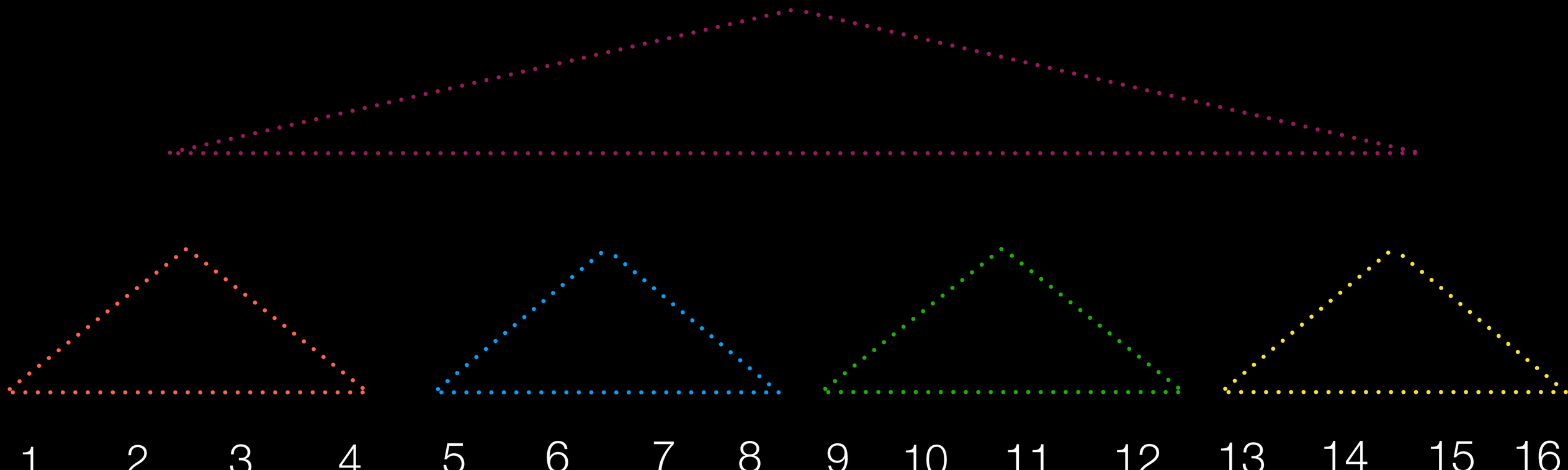
reduction trees

This just in:
the players are numbers;
the matches are addition!



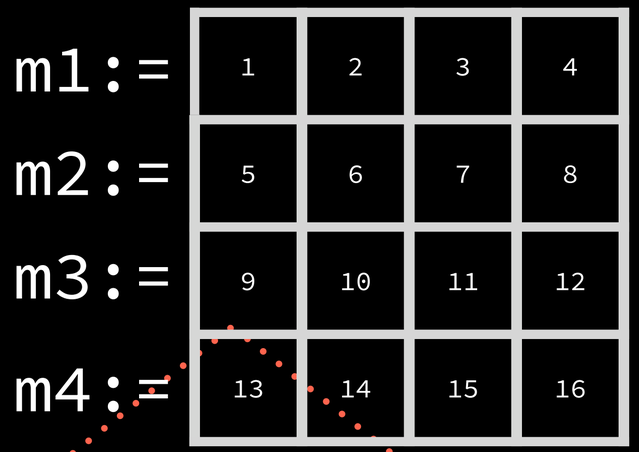
reduction trees

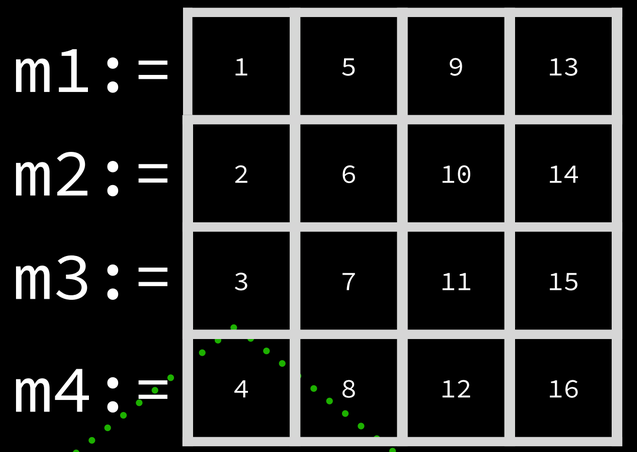
How would you handle 16 numbers?

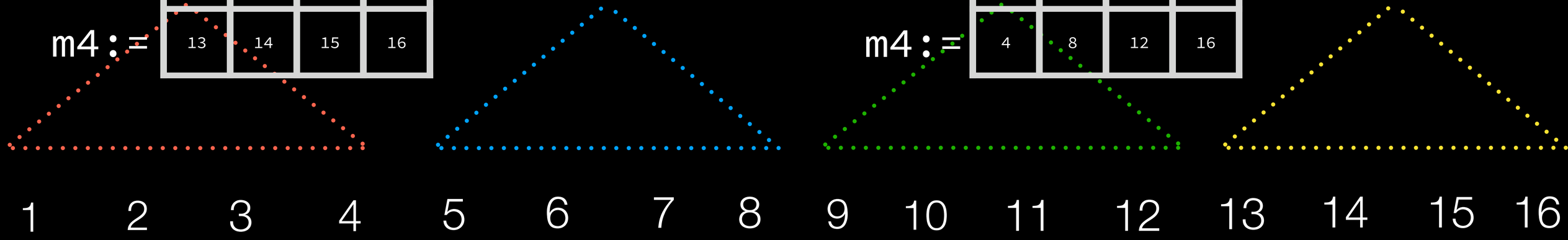


reduction trees, feat. banking

Where would you place the 16 numbers?







arbitrage

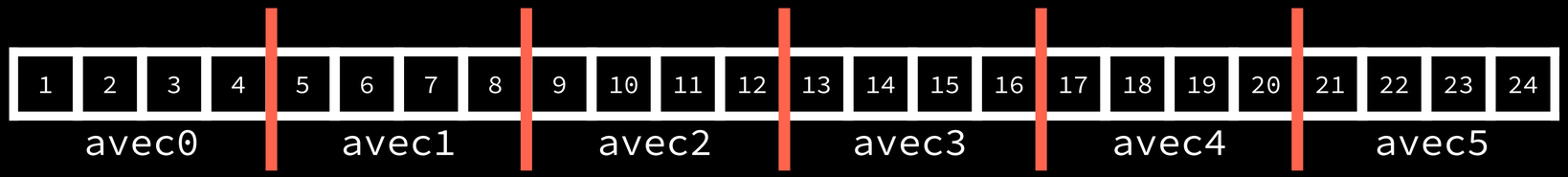
What if I want to parallelize the same memory, but with different banking factors?

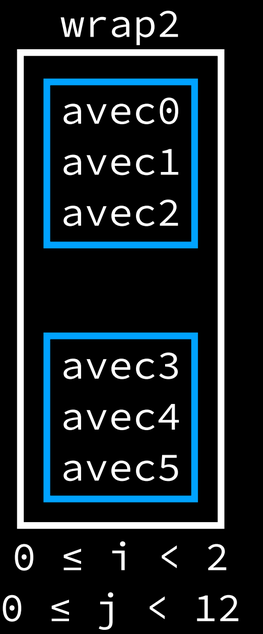
```
avec :=
```

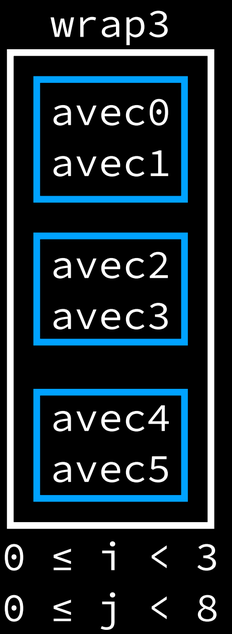

```
squares := map 3 (a <- avec) { a * a }

add_1 := map 2 (a <- avec) { a + 1 }
```

We need to break `avec` into `lcm(2,3)=6` banks, and then arbitrate between those.







make MrXL better!

closing remarks