Calyx Tutorial

PLDI 2023

https://calyxir.org/tuto

Time	Topic
9-9:20am	Introduction to Calyx, and setting up
9:20-9:55am	Your first Calyx program
9:55-10am	`fud`, the hardware tool composer
10-10:10am	MrXL, a `map-reduce` frontend
10:10-10:50am	Implement a `map` operation for MrXL
10:50-10:55am	Cider, the Calyx interactive debugger
10:55-11am	Pollen, a pangenome analysis DSL
11-11:20am	Break
11:20am-12:20pm	Contest: extensions to MrXL!
12:20-12:30pm	Award ceremony and closing remarks

opening remarks

- Hardware is important and Verilog is bad... we can fix that
  - Compilers are cool (as opposed to hand-writing fixed-function code)
- Setting up: check out Docker container, run fud check, run runt

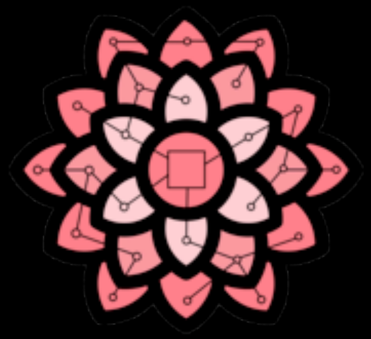
- Companies are investing in custom hardware
- Countries are investing in hardware design
- And mostly importantly, you can write papers about hardware

- But, using Verilog is a pain
  - Bad language
  - Bad tools
  - Complex constraints (name drop things like timing closure)

1. Install Docker
2. Get the Docker container
3. Run the following commands in the container
    - fud check …
    - runt -i …

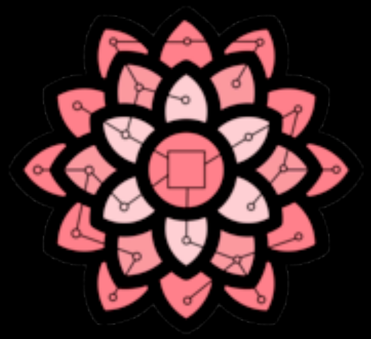
- Calyx programs have components
  - Which have cells, wires, and control
  - Wires are structured using groups
  - Control operators schedule them
- Link to language tutorial

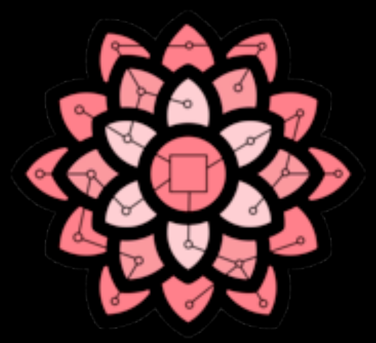

Calyx



Calyx

MrXL

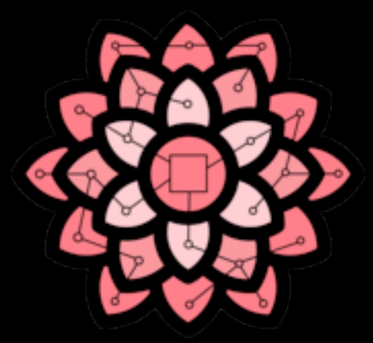




MrXL

tvm

Calyx



MrXL

tvm

MyDSL

Calyx

MrXL

`fud`, the Calyx driver

`fud`, the Calyx driver

```
fud e squares.mrxl --from mrxl --to calyx
```

`fud`, the Calyx driver

```
fud e squares.mrxl --from mrxl --to calyx
```

`fud`, the Calyx driver

```
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;
```

`fud`, the Calyx driver

`fud`, the Calyx driver

```
fud e squares.mrxl --from mrxl --to verilog
```

`fud`, the Calyx driver

```
fud e squares.mrxl --from mrxl --to verilog
```

`fud`, the Calyx driver

```
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;
```

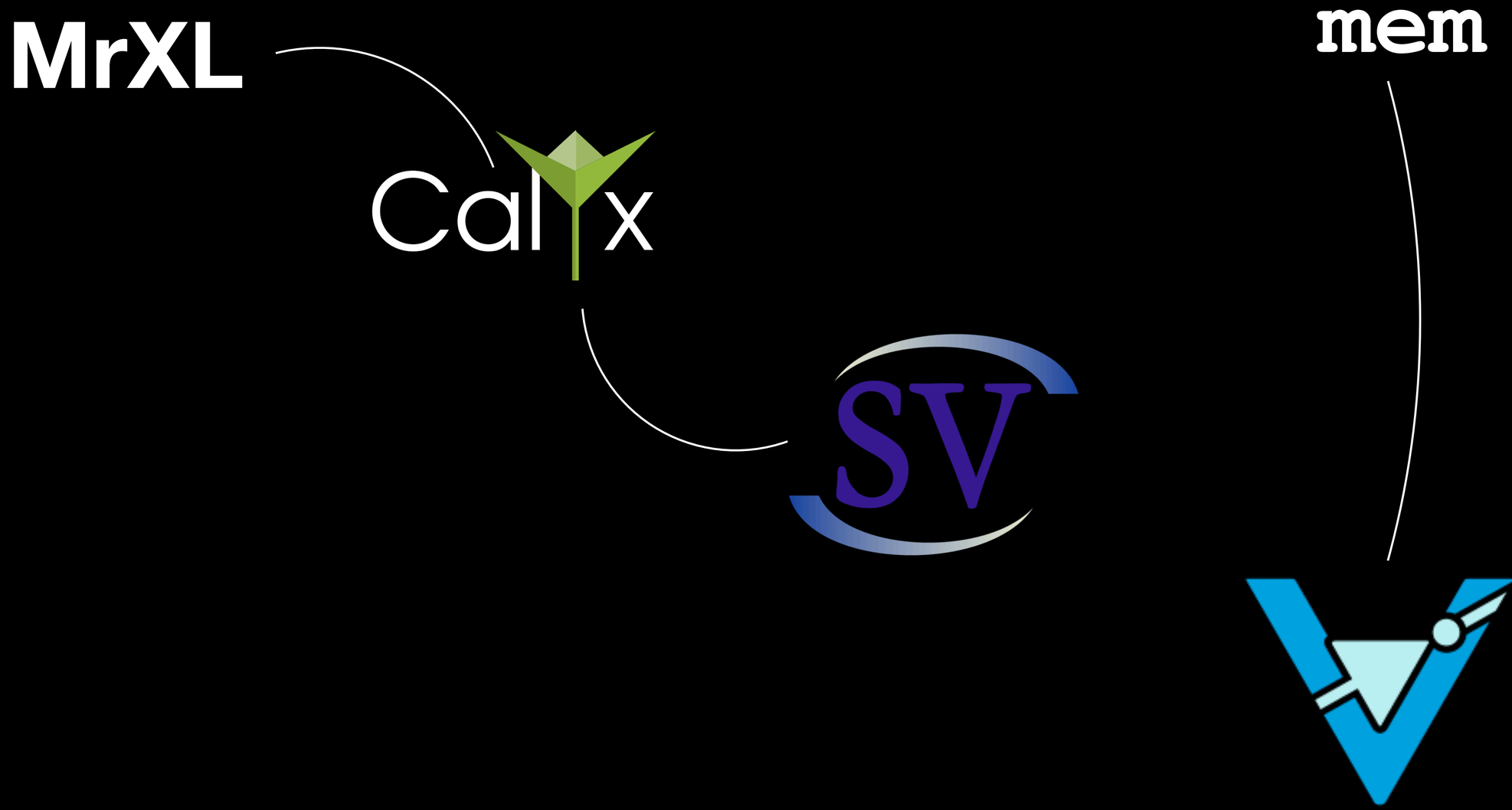
MrXL

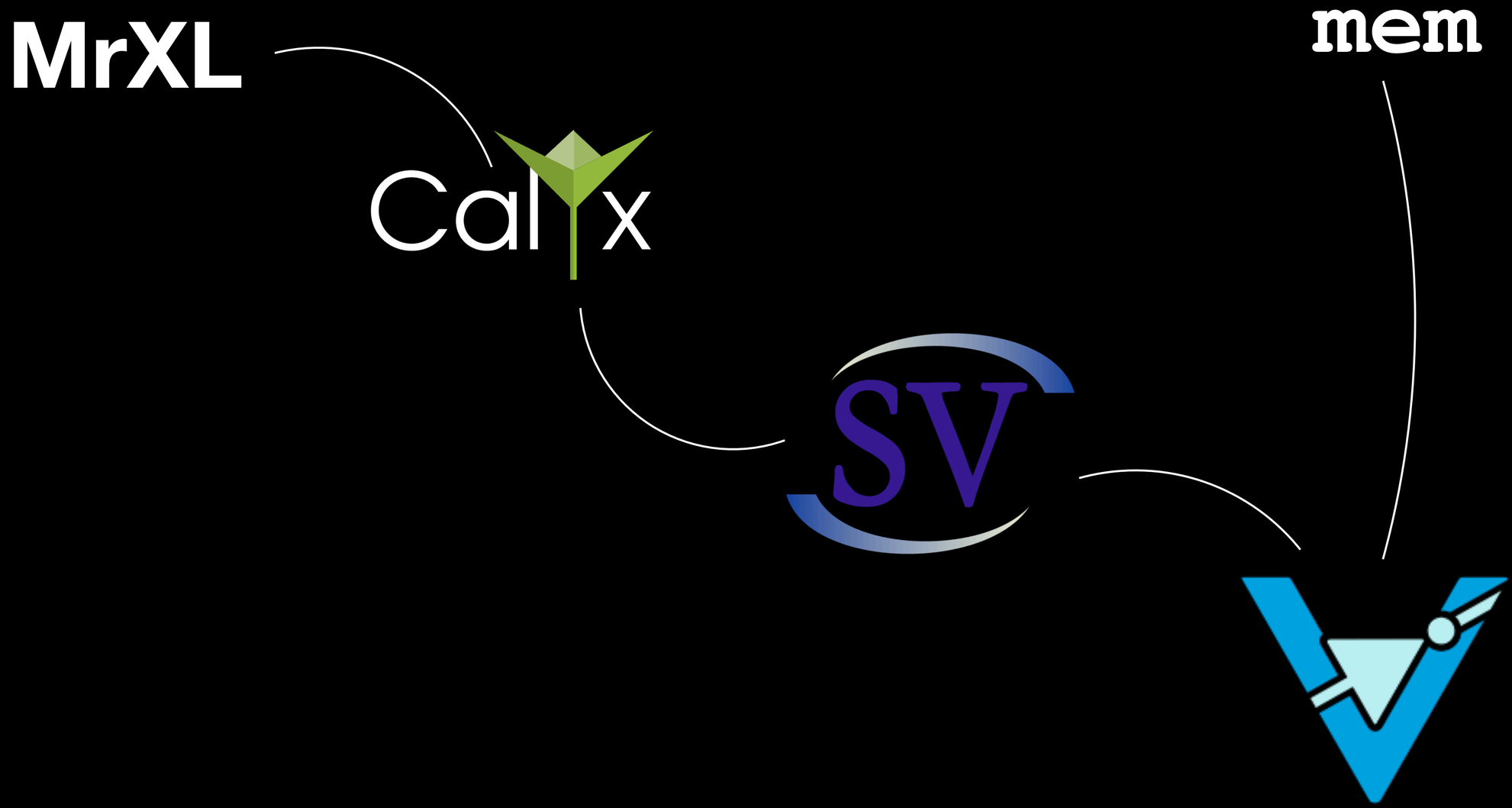
MrXL

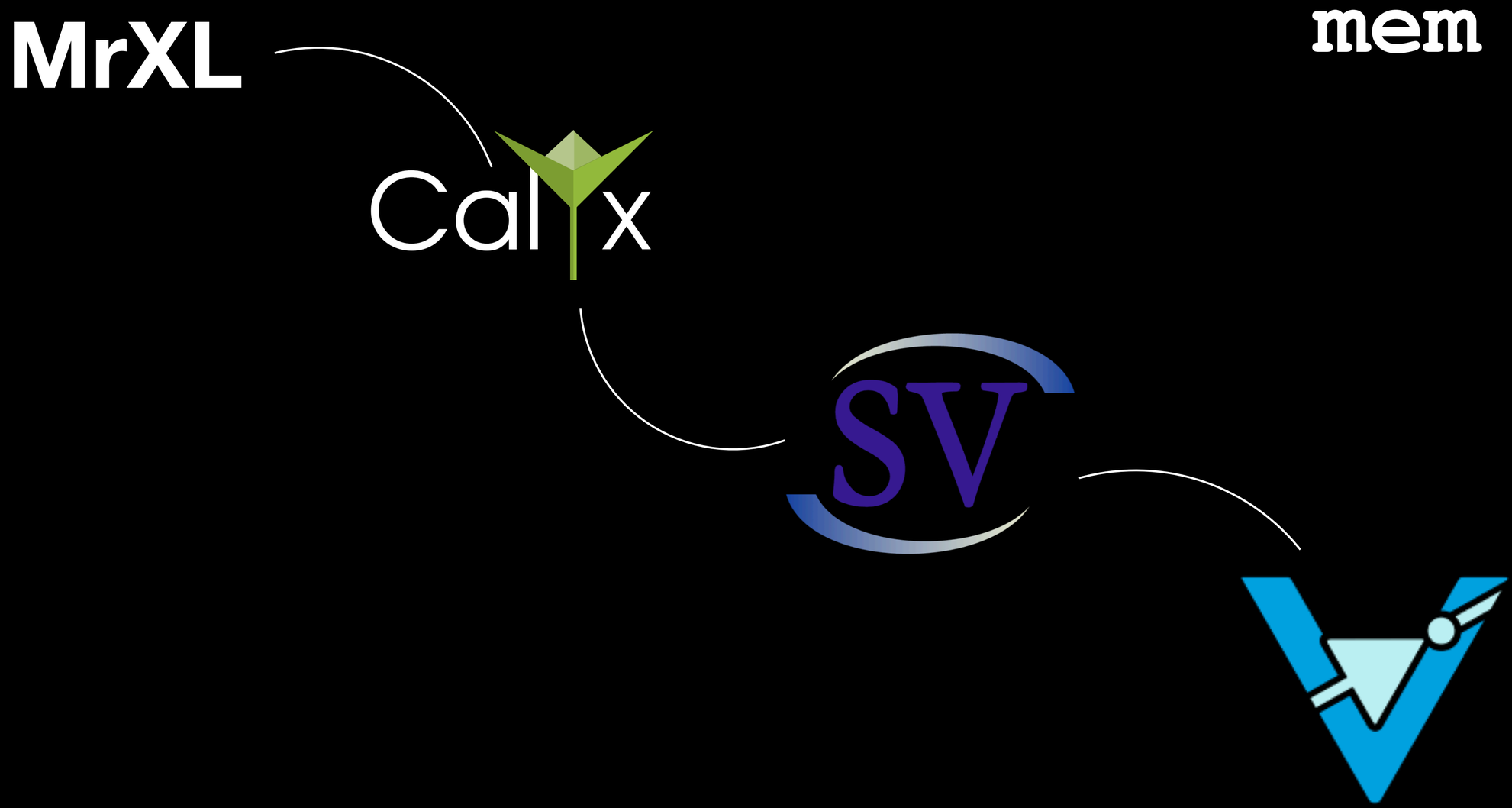
mem

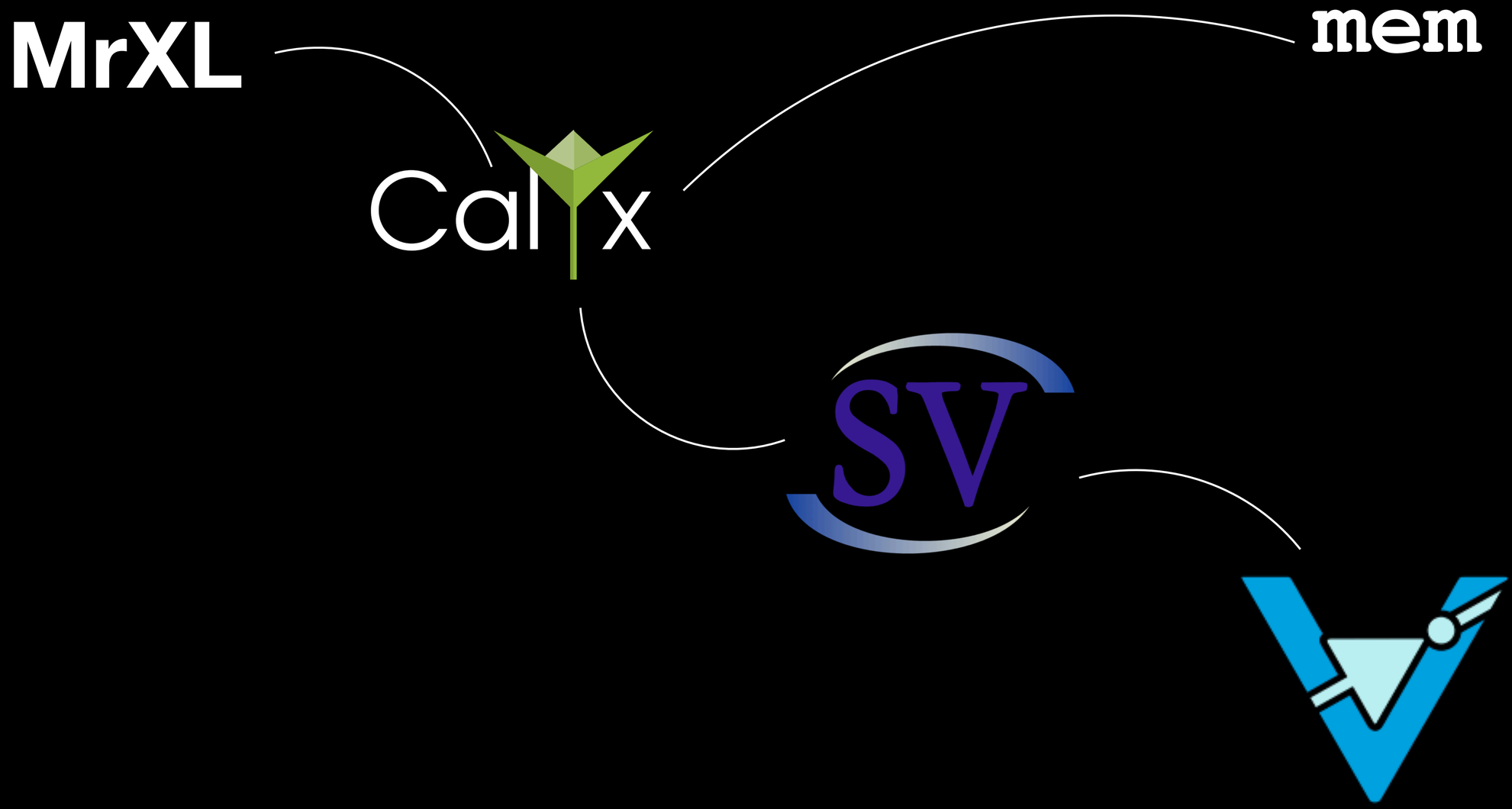
MrXL

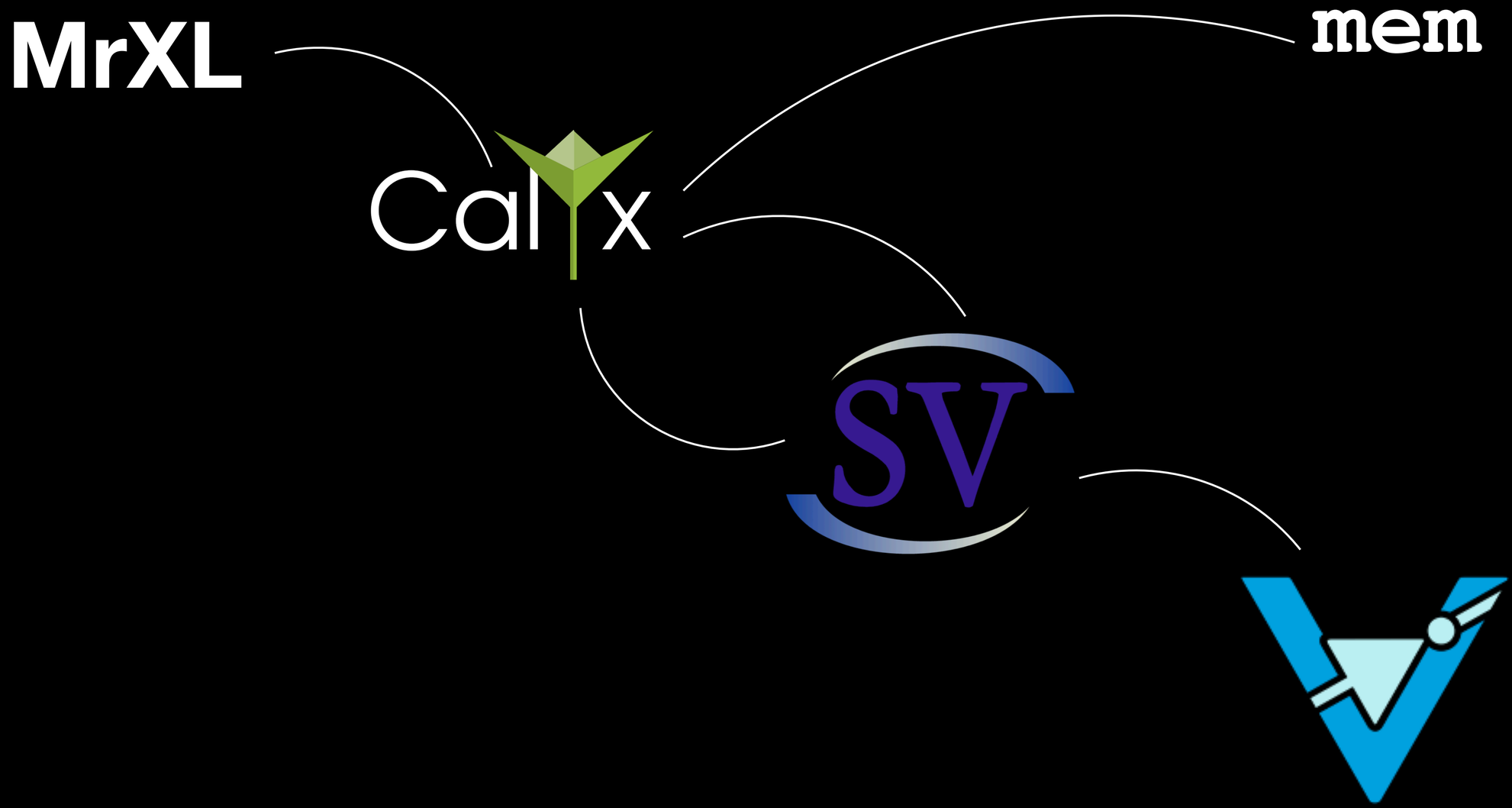
mem

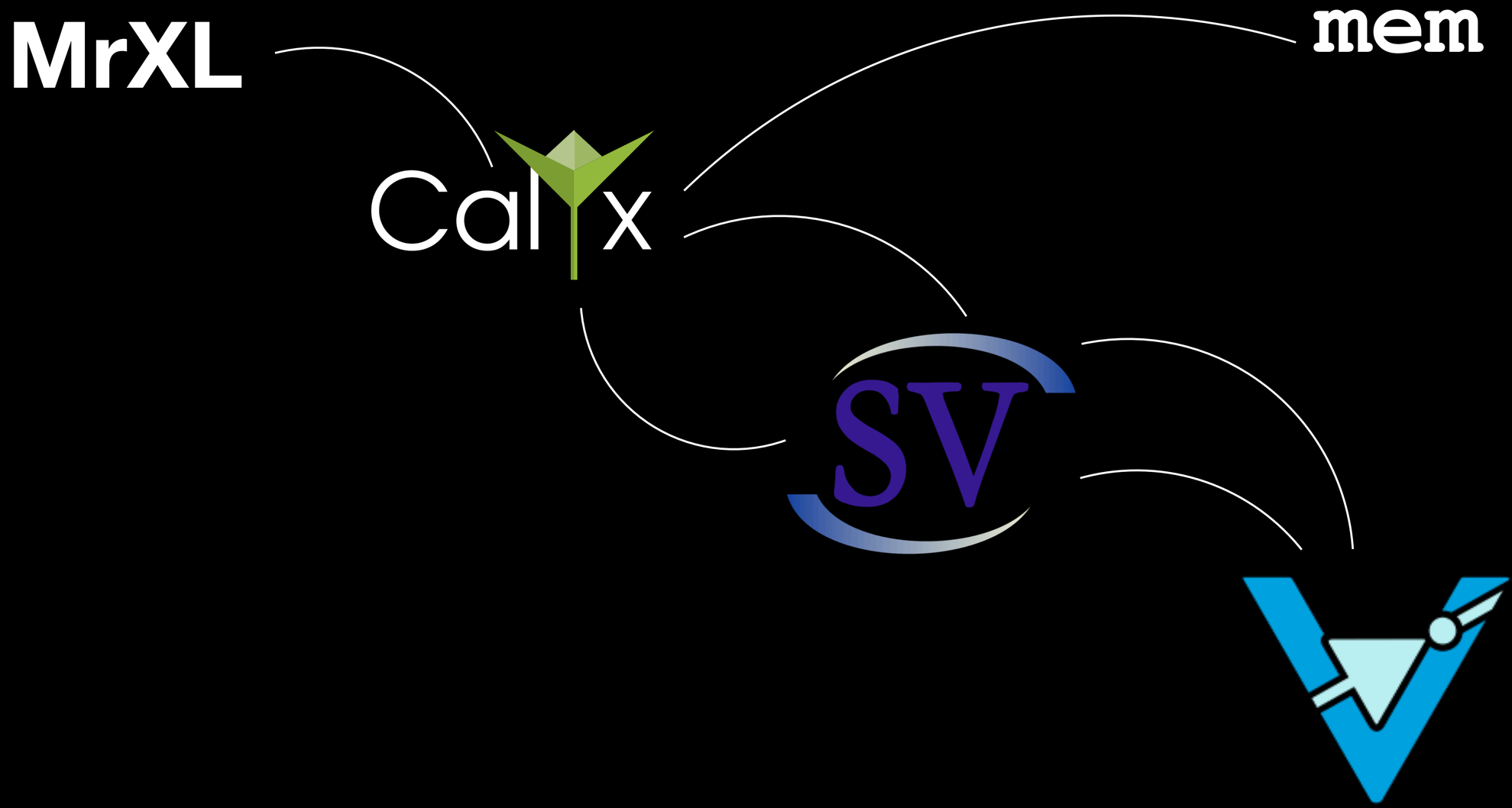

MrXL
Calyx
SV
mem


MrXL
Calyx
SV
mem


MrXL
Calyx
SV
mem


MrXL
Calyx
mem
SV


MrXL
Calyx
mem
SV


MrXL
mem
Calyx
SV

```
fud e squares.mrxl --from mrxl --to dat \
--through verilog -s mrxl.data squares.mrxl.data
```

```
fud e squares.mrxl --from mrxl --to dat \
--through verilog -s mrxl.data squares.mrxl.data
```

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

```
fud e squares.mrxl --from mrxl --to dat \
--through verilog -s mrxl.data squares.mrxl.data -n
```

```
fud e squares.mrxl --from mrxl --to dat \
--through verilog -s mrxl.data squares.mrxl.data -n
```

```
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.mrxl --from mrxl --to calyx
```

```
fud e squares.mrxl --from mrxl --to verilog
```

```
fud e squares.mrxl --from mrxl --to dat \
--through icarus-verilog -s mrxl.data squares.mrxl.data
```

```
fud e squares.mrxl --from mrxl --to dat \
--through verilog -s mrxl.data squares.mrxl.data
```

```
fud e squares.mrxl --from mrxl --to interpreter-out \
-s mrxl.data squares.mrxl.data
```

```
fud e squares.mrxl --from mrxl --to interpreter-out \
-s mrxl.data squares.mrxl.data
```

```
fud e squares.xclbin --from xclbin --to fpga \
-s fpga.data squares.mrxl.data
```

MrXL: `map-reduce` accelerator

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

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.

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!
Work in `frontends/mrxl`

anatomy of a frontend

program.mrxl

program.futil

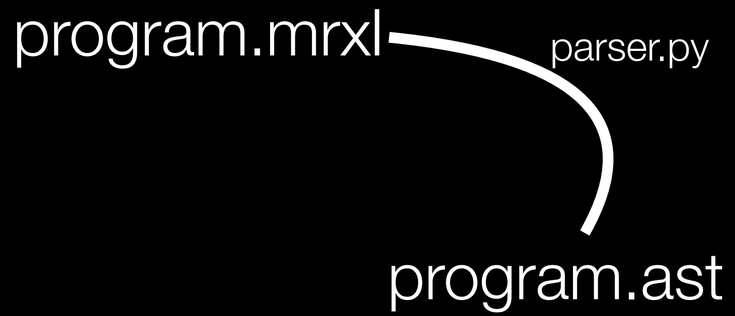
anatomy of a frontend

program.mrxl

program.ast

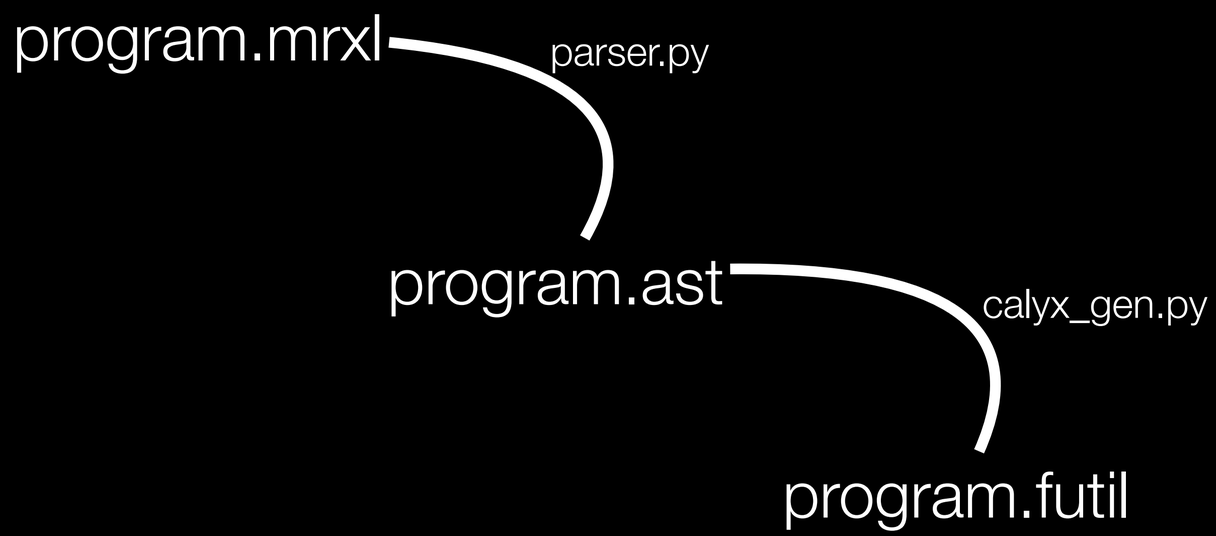
program.futil

anatomy of a frontend

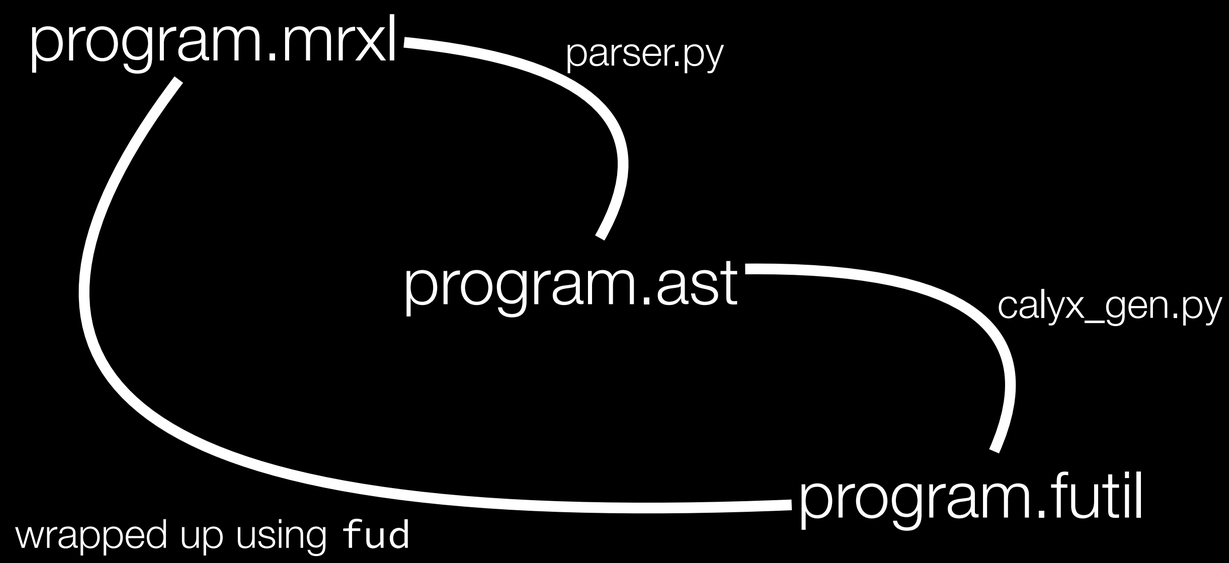


program.futil

anatomy of a frontend



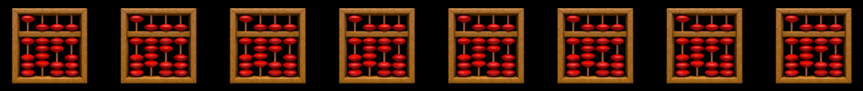
anatomy of a frontend



memory banking

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

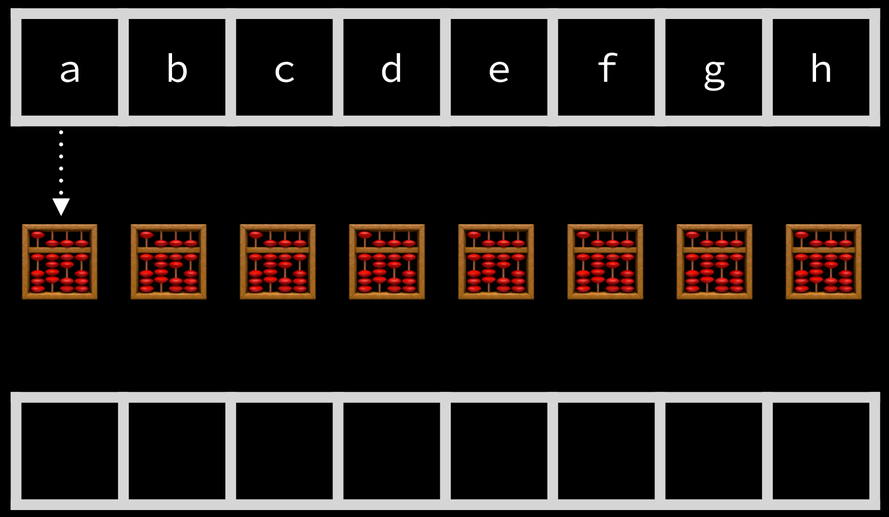
Say I give you all the compute
you could ask for:



memory banking

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

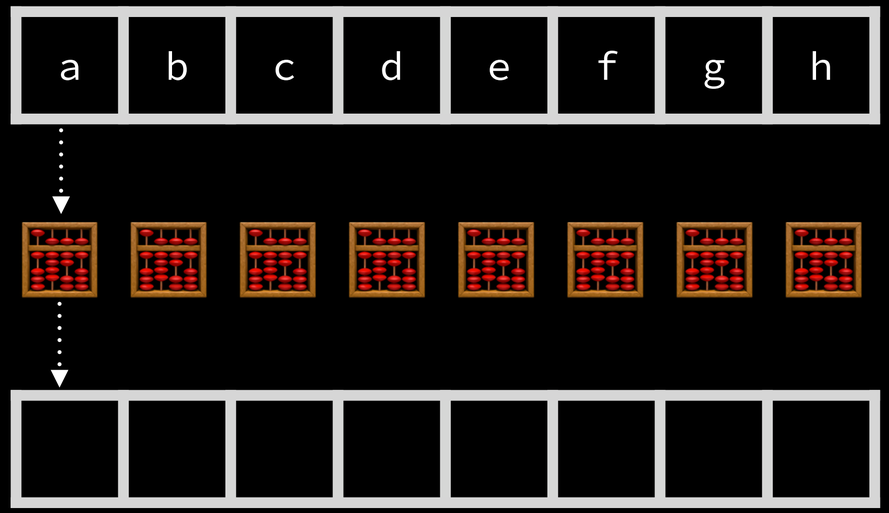
Say I give you all the compute
you could ask for:



memory banking

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

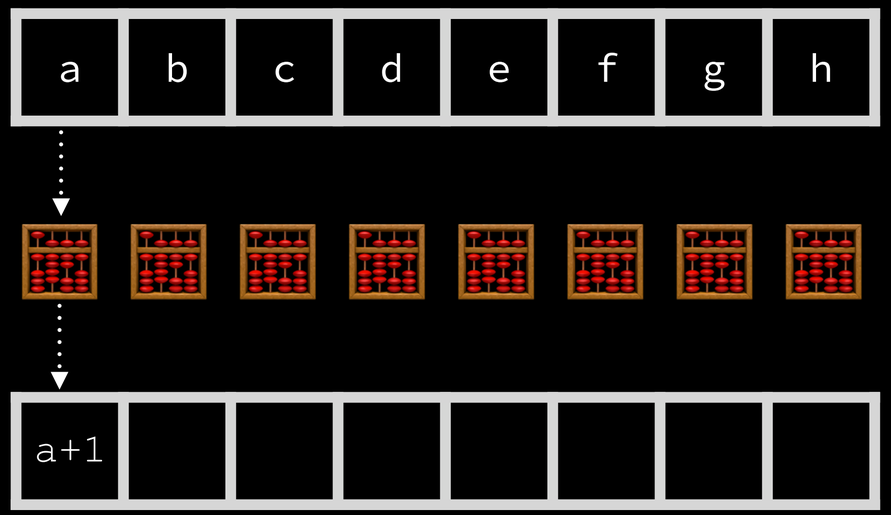
Say I give you all the compute
you could ask for:



memory banking

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

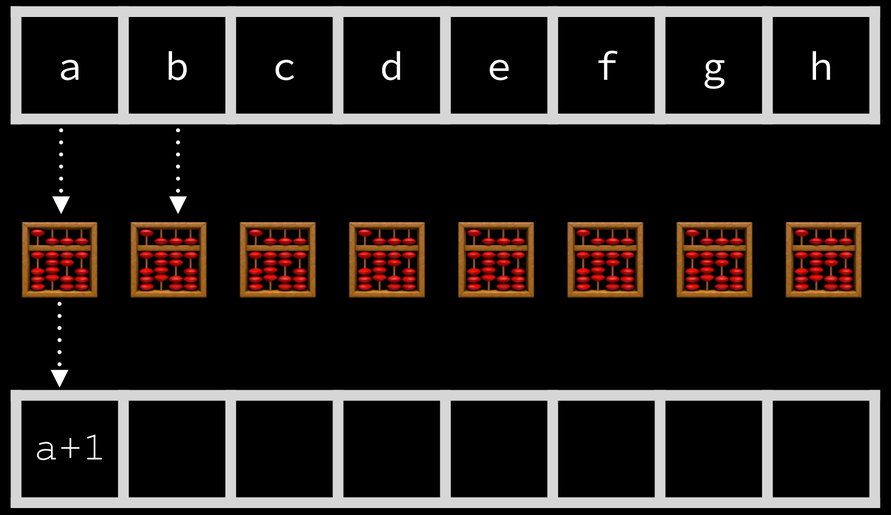
Say I give you all the compute
you could ask for:



memory banking

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

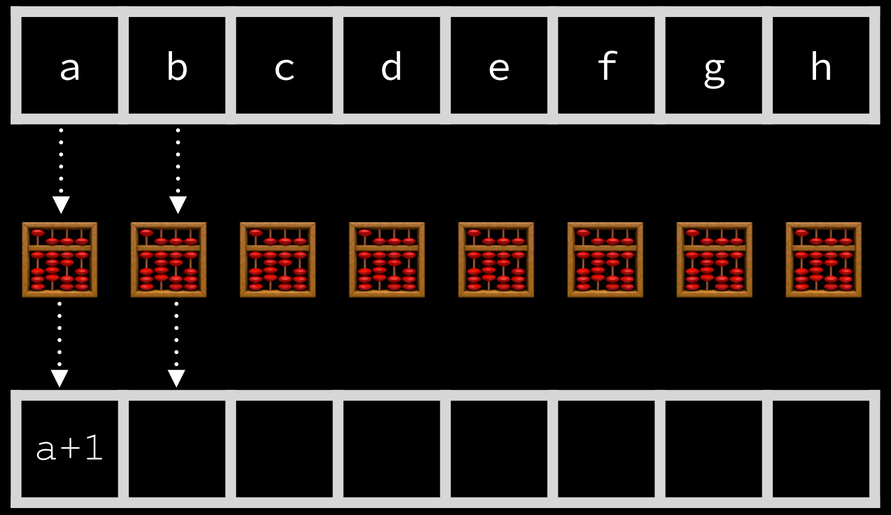
Say I give you all the compute
you could ask for:



memory banking

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

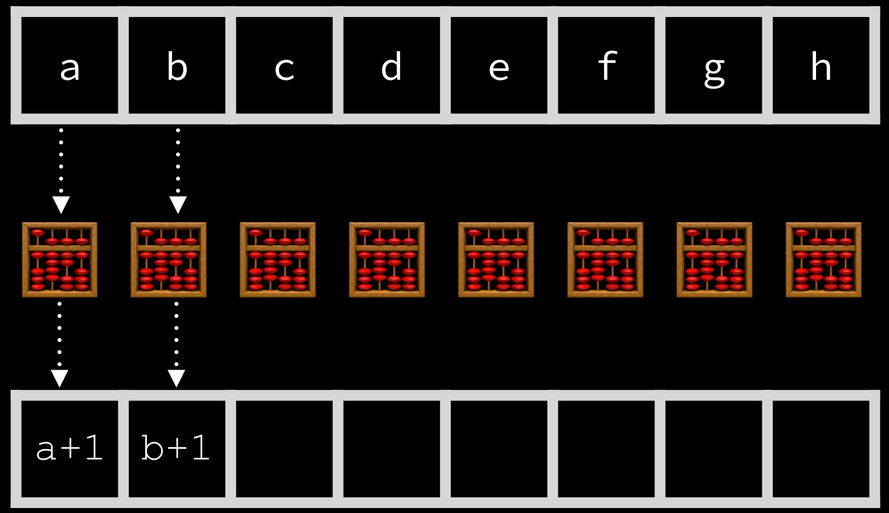
Say I give you all the compute
you could ask for:



memory banking

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

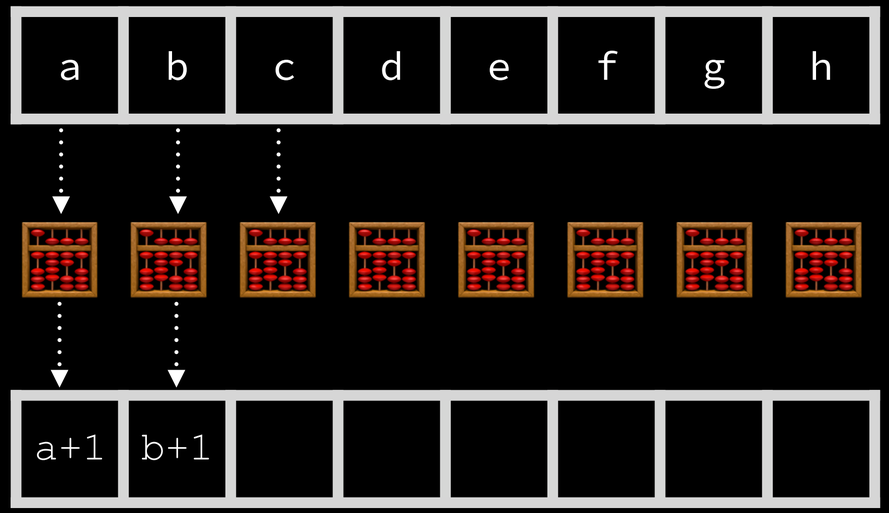
Say I give you all the compute
you could ask for:



memory banking

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

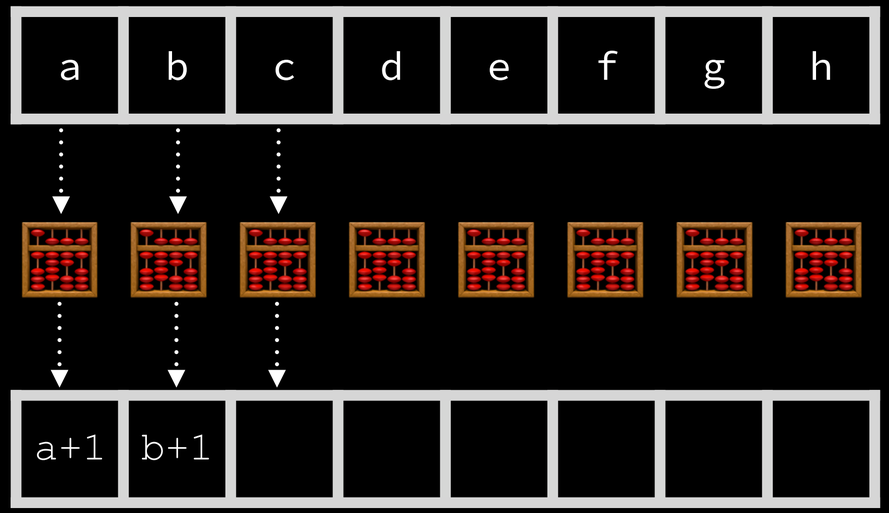
Say I give you all the compute
you could ask for:



memory banking

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

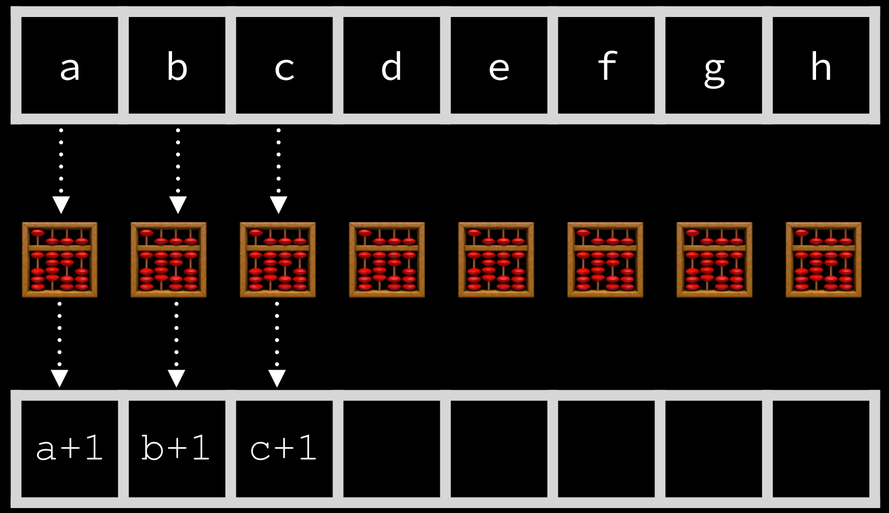
Say I give you all the compute
you could ask for:



memory banking

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

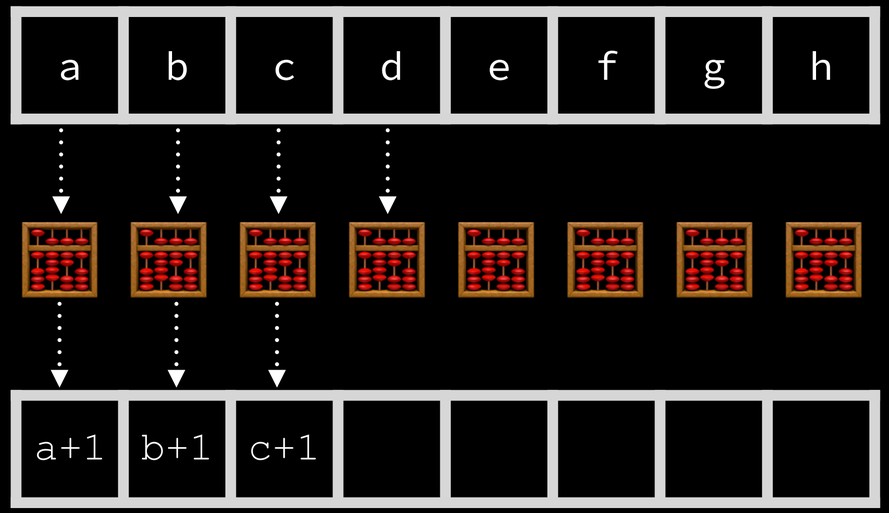
Say I give you all the compute
you could ask for:



memory banking

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

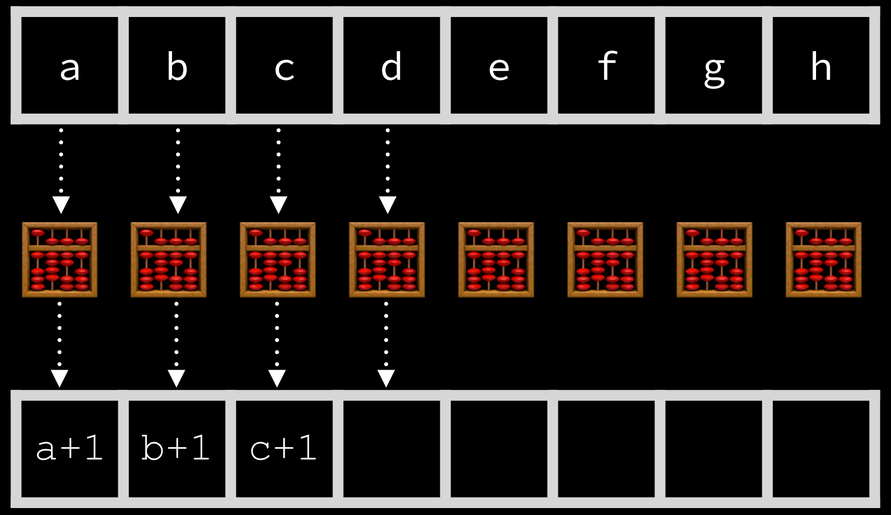
Say I give you all the compute
you could ask for:



memory banking

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

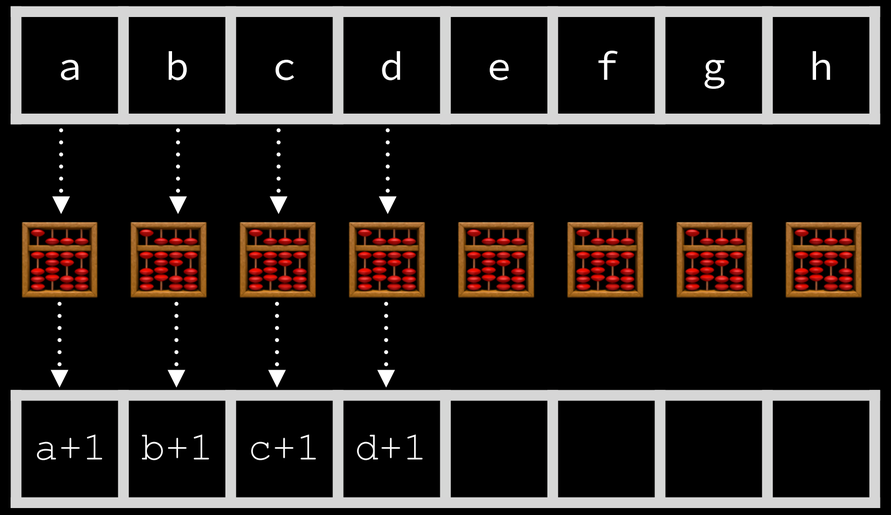
Say I give you all the compute
you could ask for:



memory banking

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

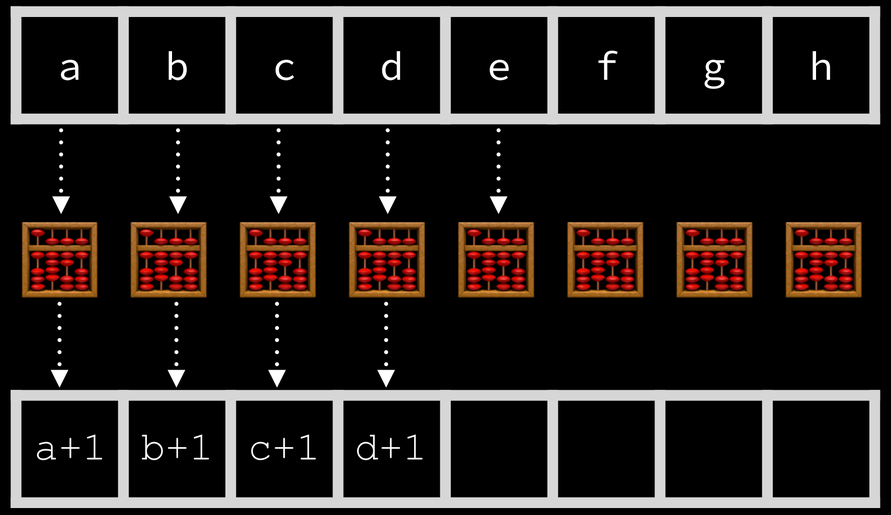
Say I give you all the compute
you could ask for:



memory banking

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

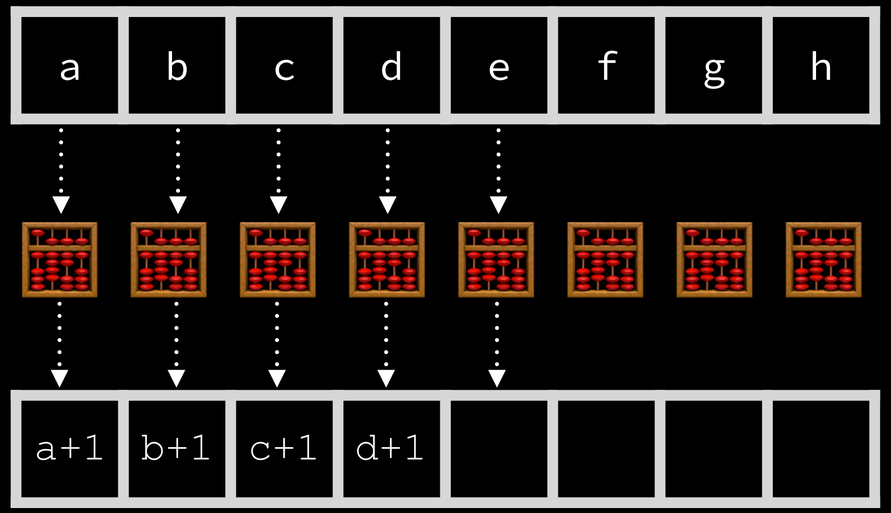
Say I give you all the compute
you could ask for:



memory banking

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

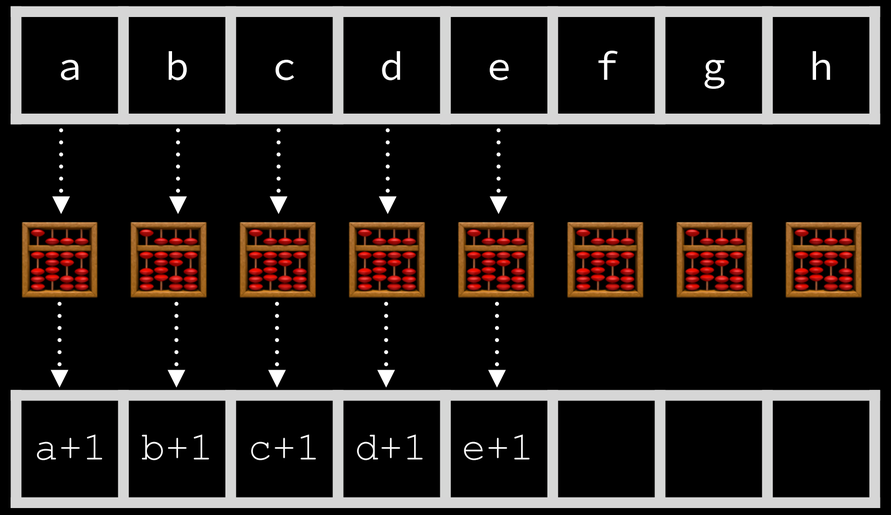
Say I give you all the compute
you could ask for:



memory banking

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

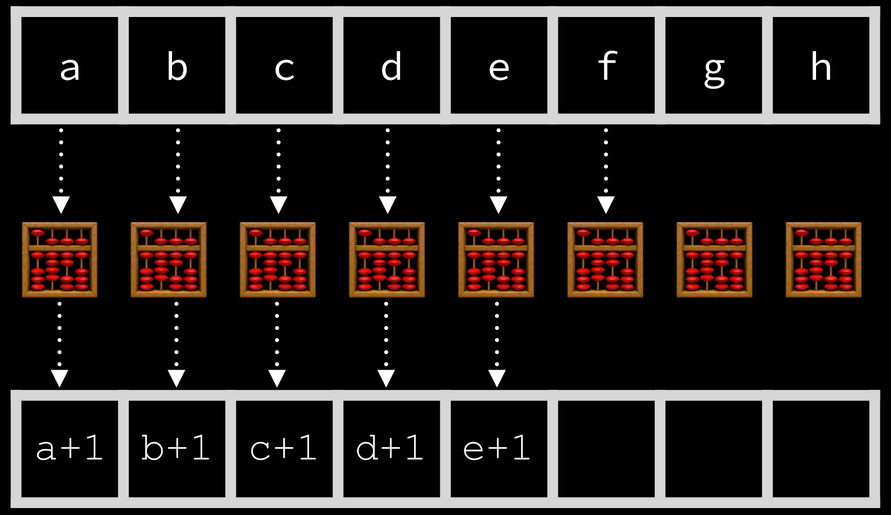
Say I give you all the compute
you could ask for:



memory banking

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

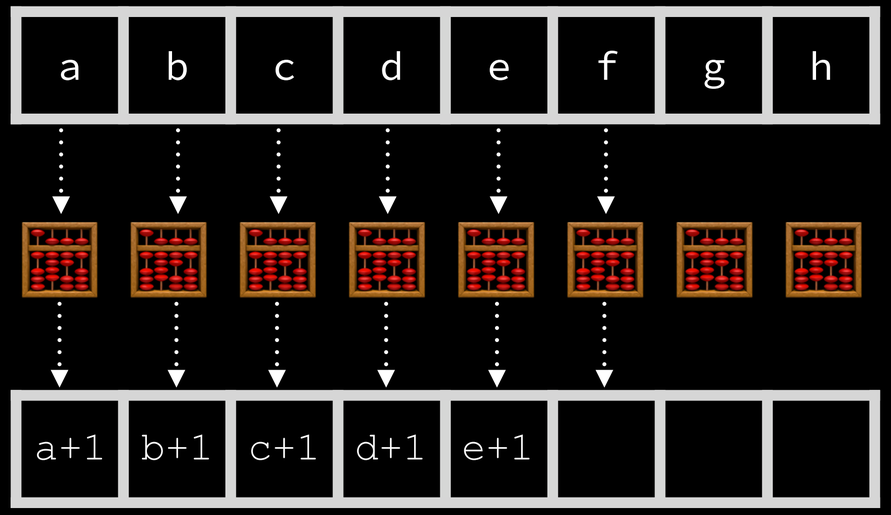
Say I give you all the compute
you could ask for:



memory banking

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

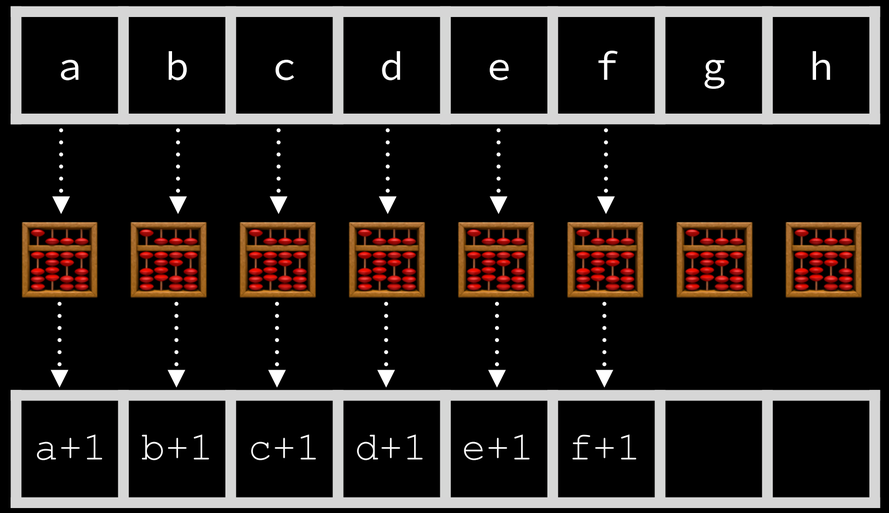
Say I give you all the compute
you could ask for:



memory banking

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

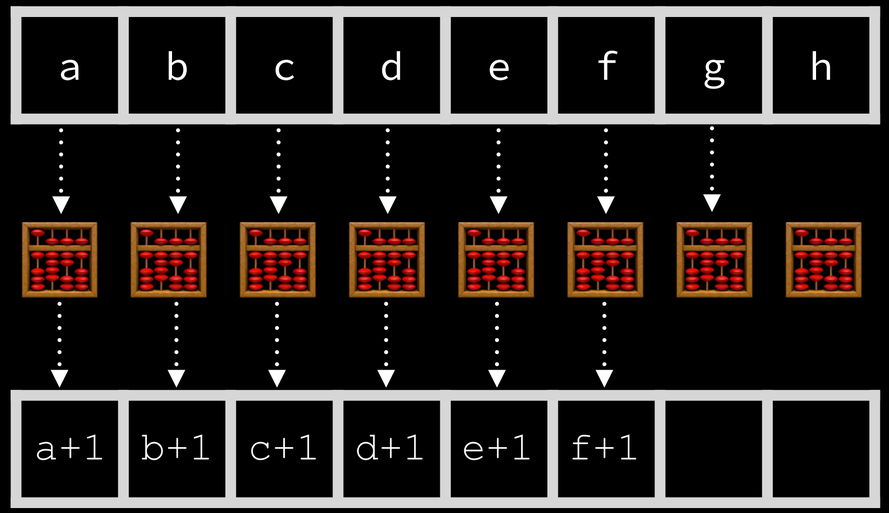
Say I give you all the compute
you could ask for:



memory banking

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

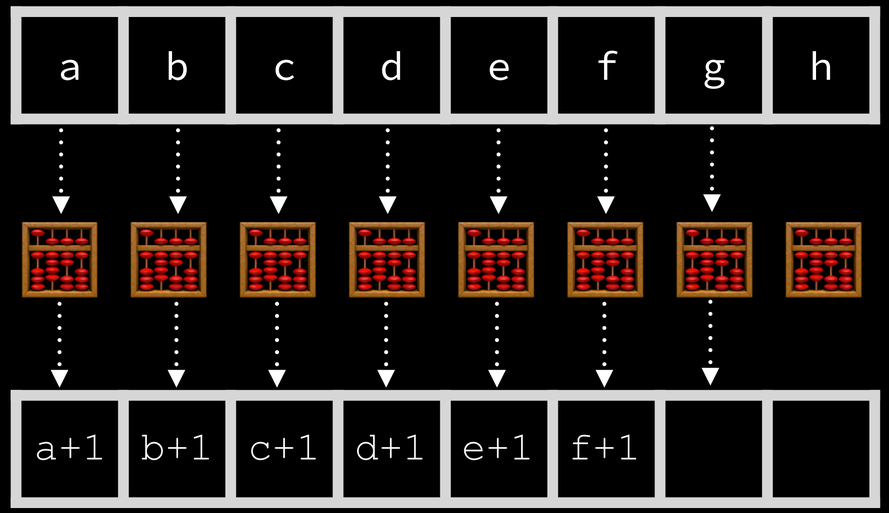
Say I give you all the compute
you could ask for:



memory banking

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

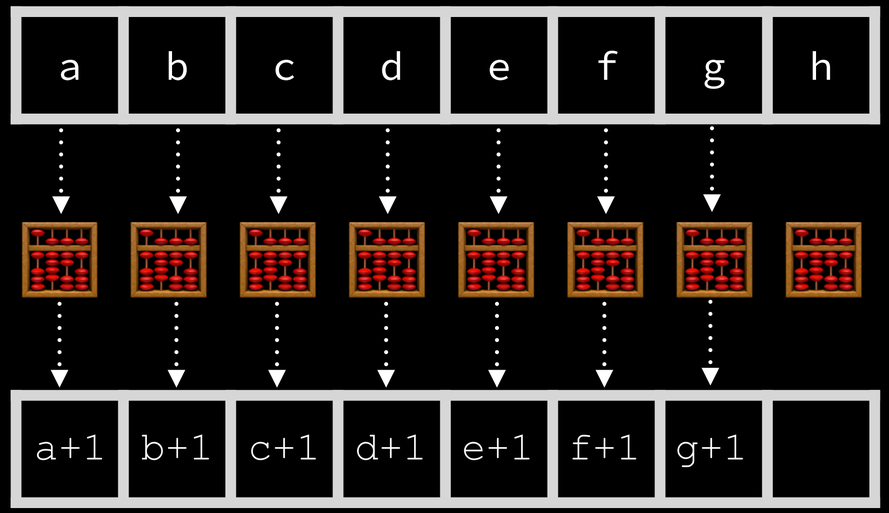
Say I give you all the compute
you could ask for:



memory banking

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

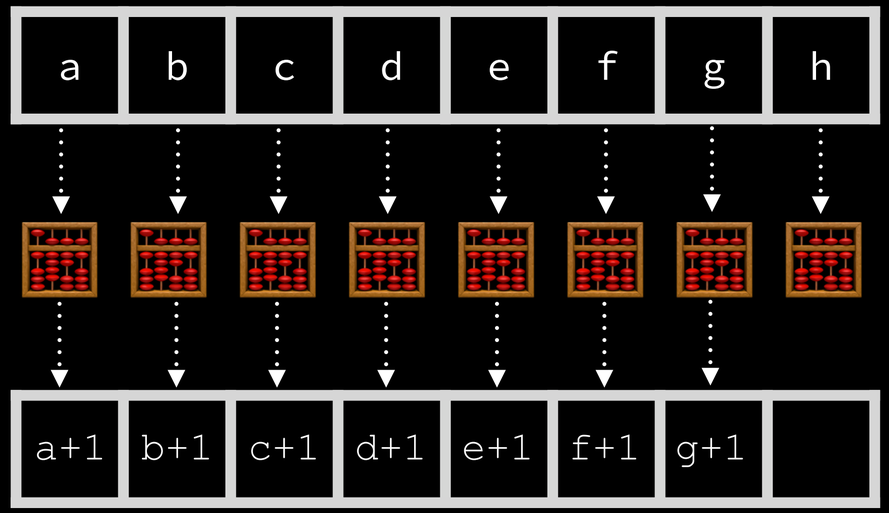
Say I give you all the compute
you could ask for:



memory banking

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

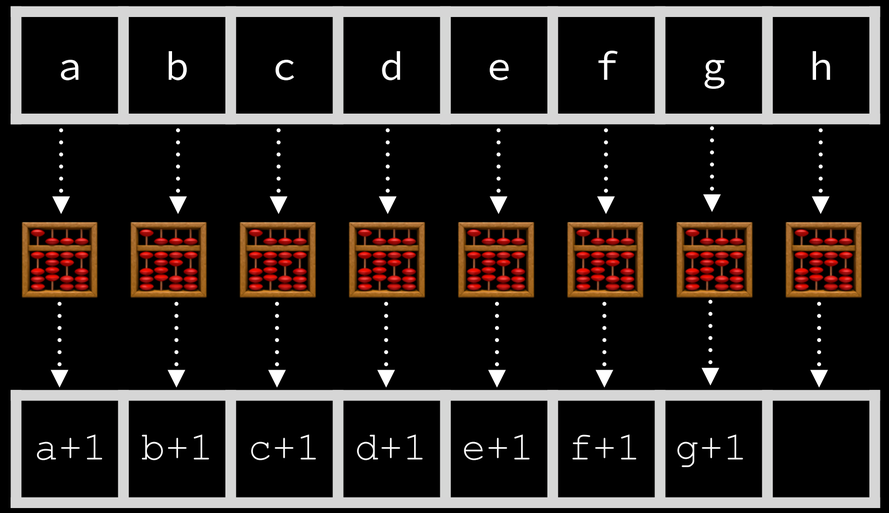
Say I give you all the compute
you could ask for:



memory banking

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

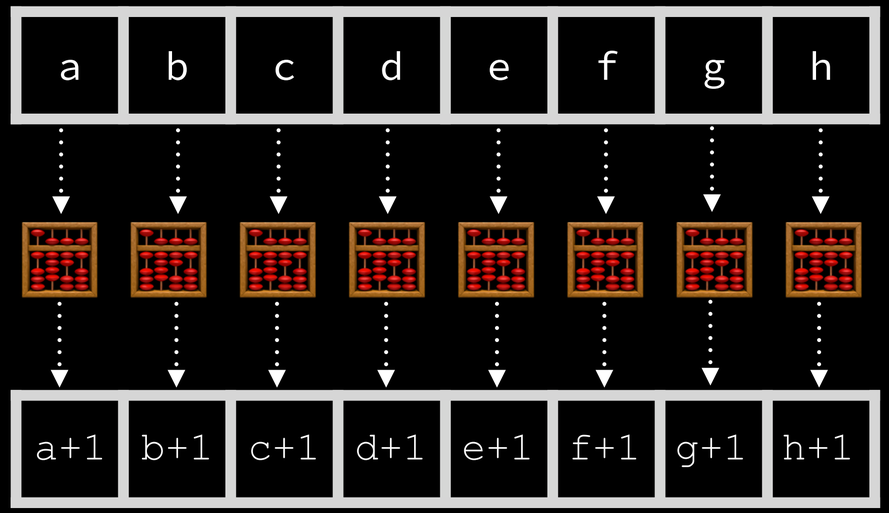
Say I give you all the compute
you could ask for:



memory banking

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

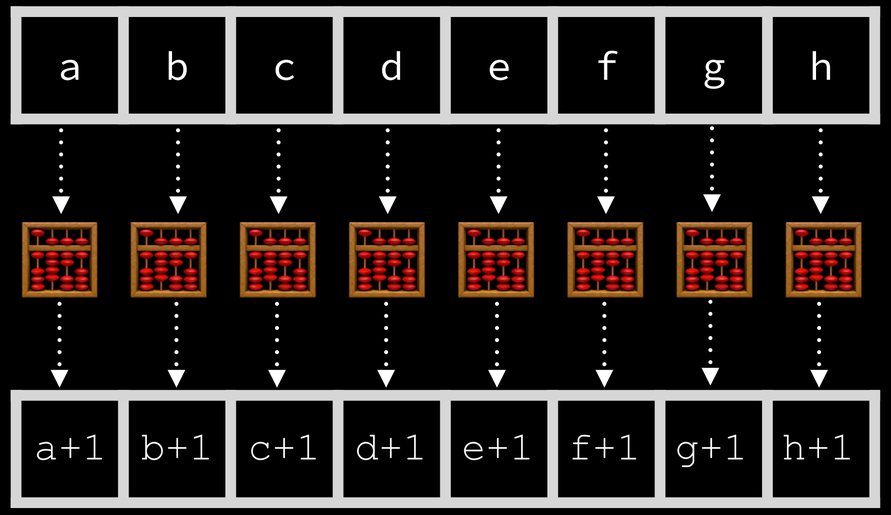
Say I give you all the compute
you could ask for:



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:

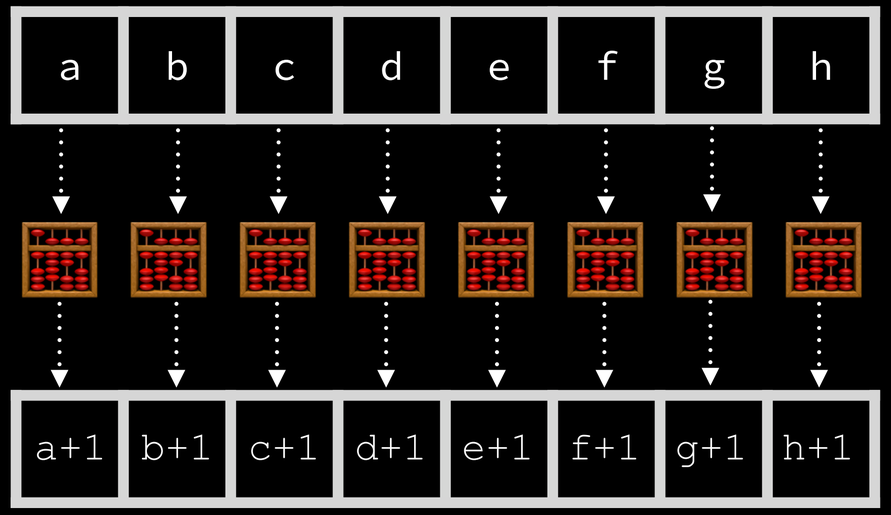


But if we *bank* the arrays,
we really can parallelize:

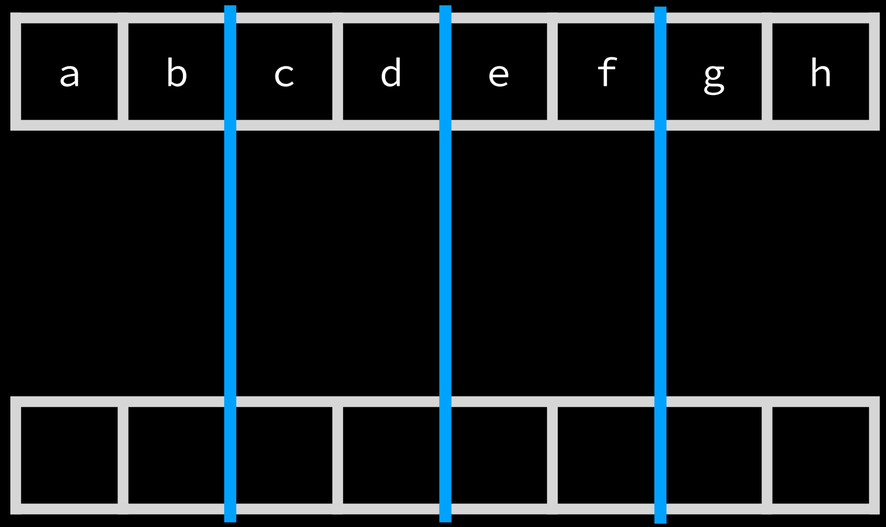
memory banking

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:



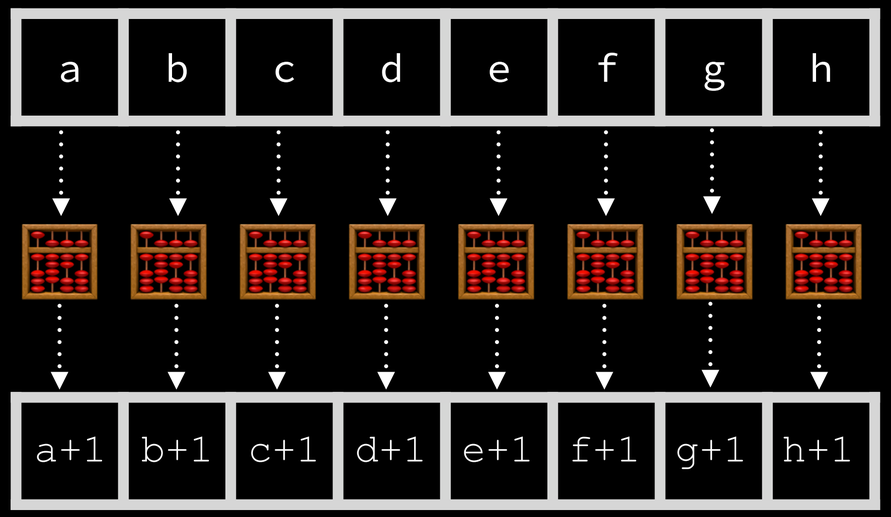
But if we *bank* the arrays,
we really can parallelize:



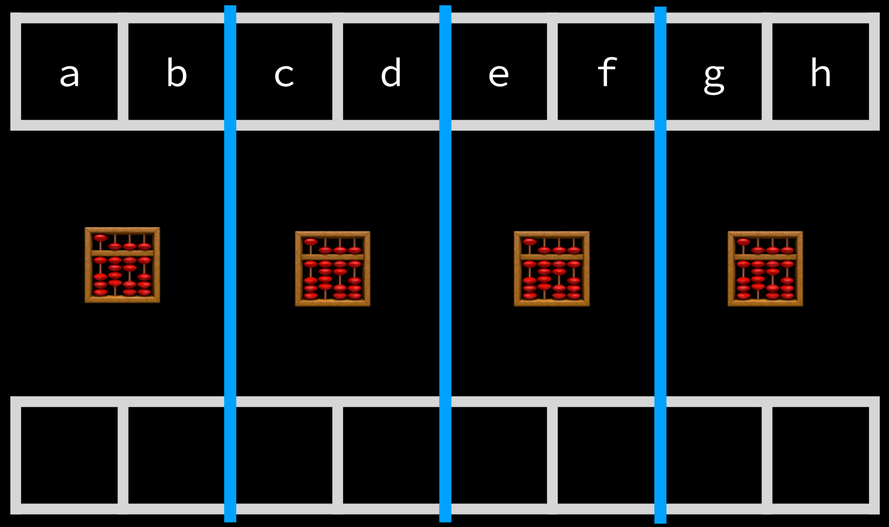
memory banking

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:



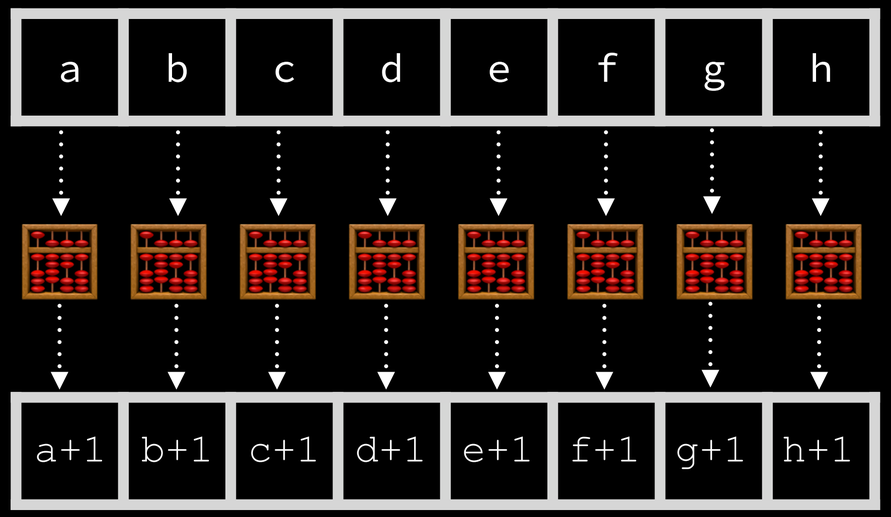
But if we *bank* the arrays,
we really can parallelize:



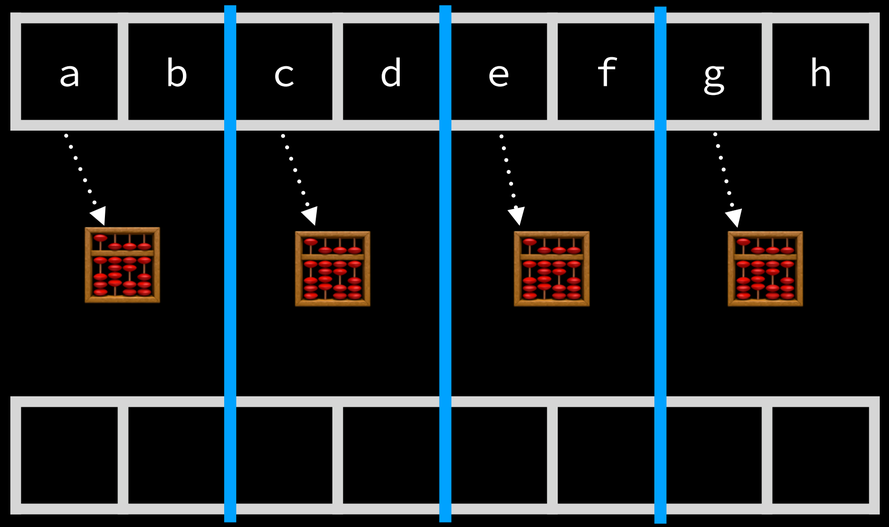
memory banking

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:



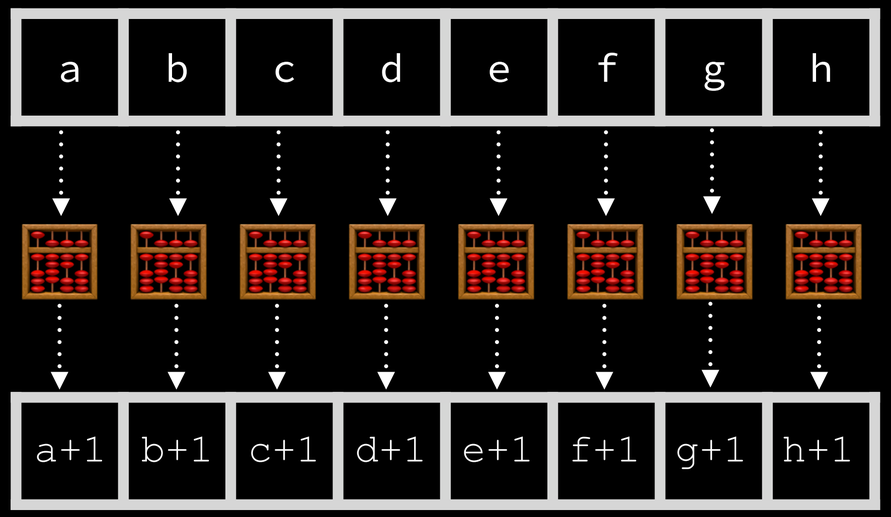
But if we *bank* the arrays,
we really can parallelize:



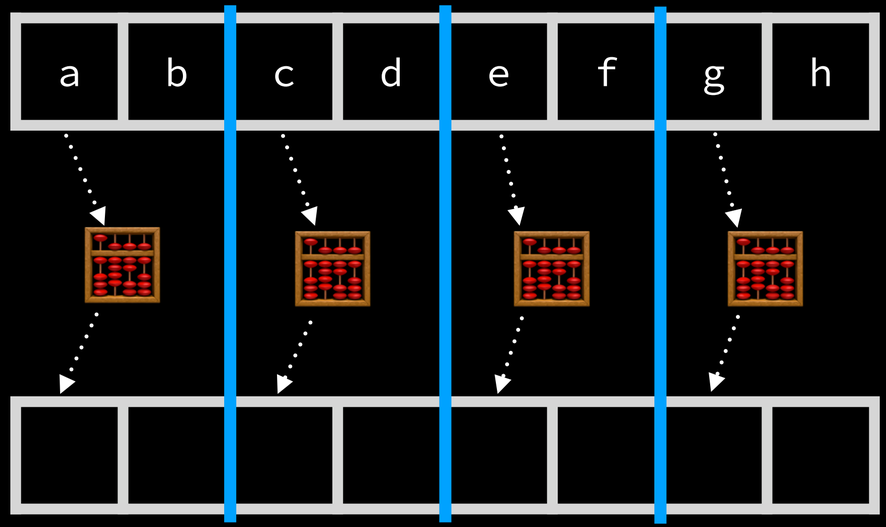
memory banking

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:



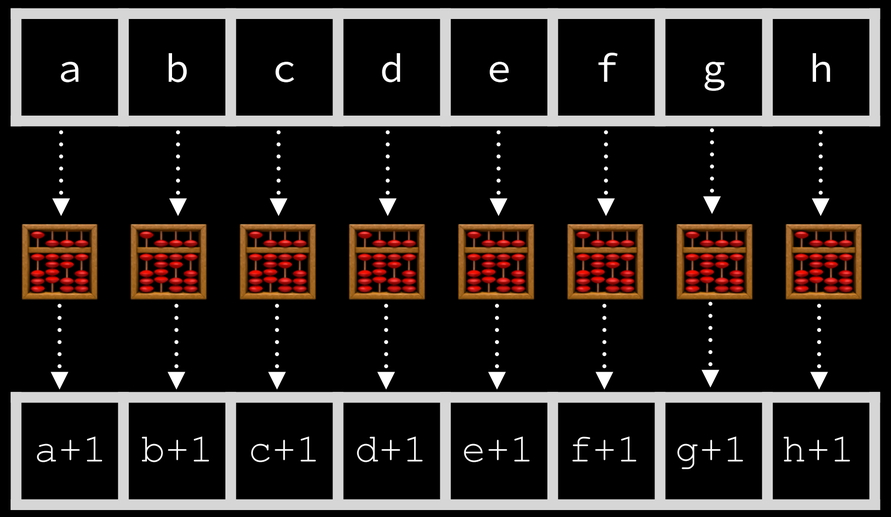
But if we *bank* the arrays,
we really can parallelize:



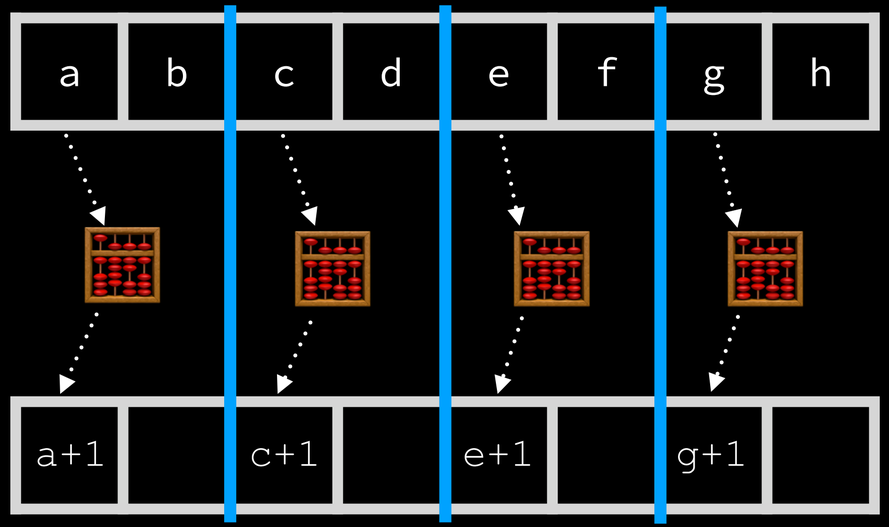
memory banking

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:



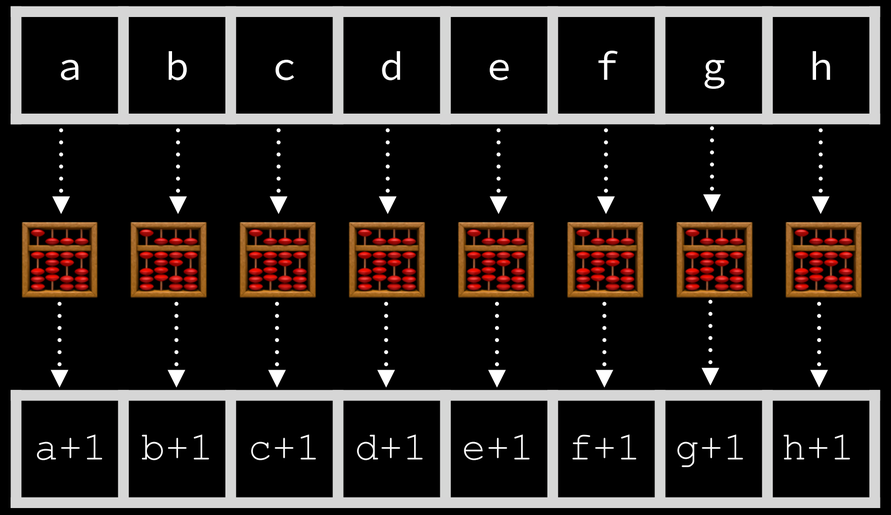
But if we *bank* the arrays,
we really can parallelize:



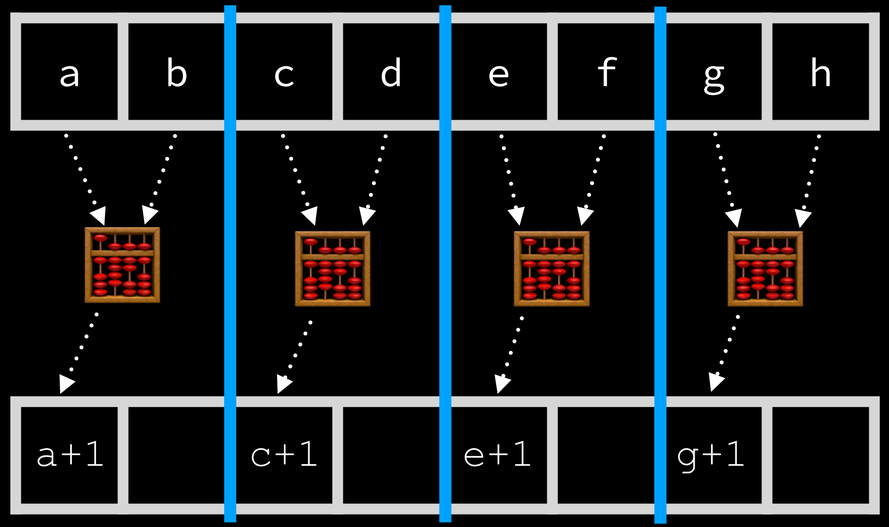
memory banking

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:



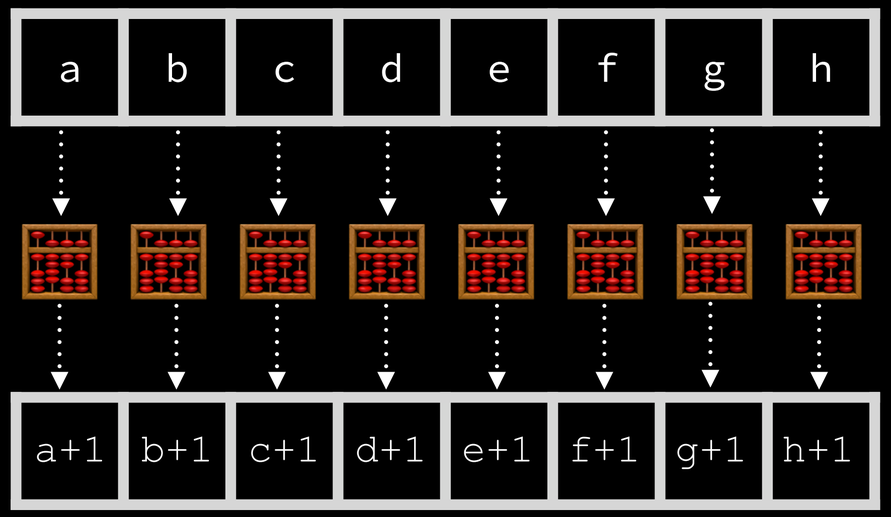
But if we *bank* the arrays,
we really can parallelize:



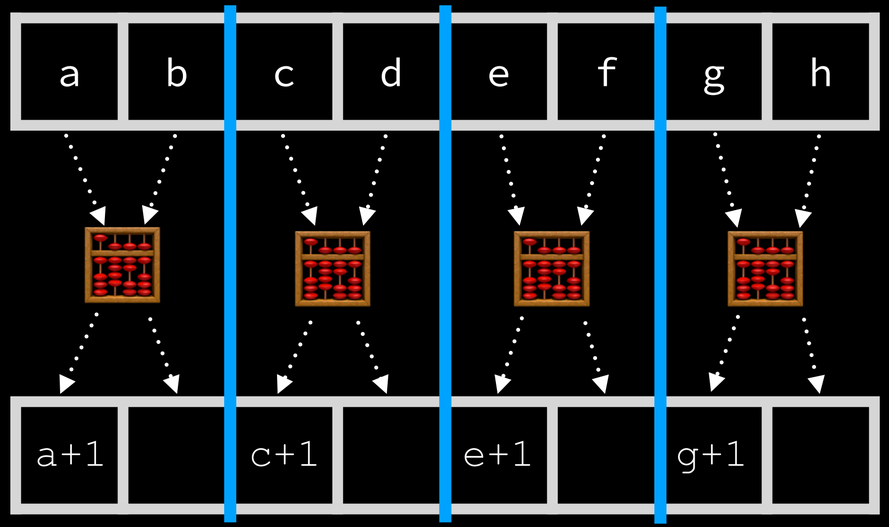
memory banking

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:



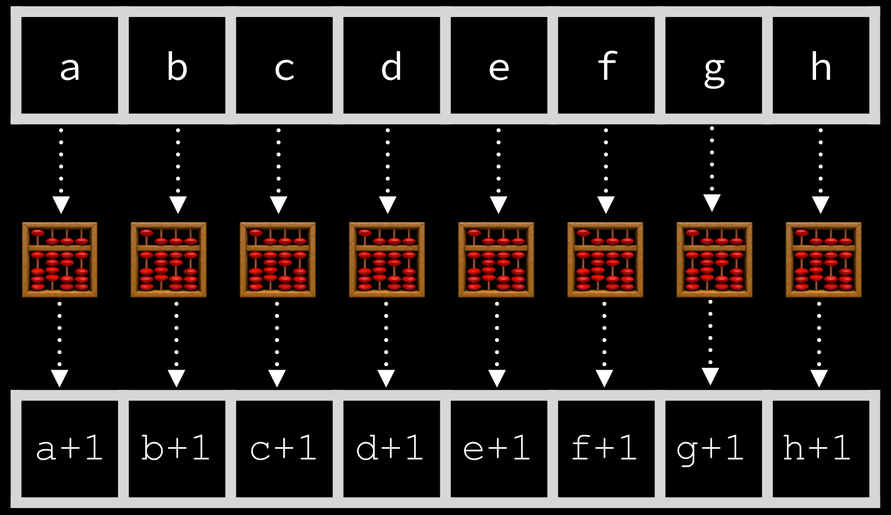
But if we *bank* the arrays,
we really can parallelize:



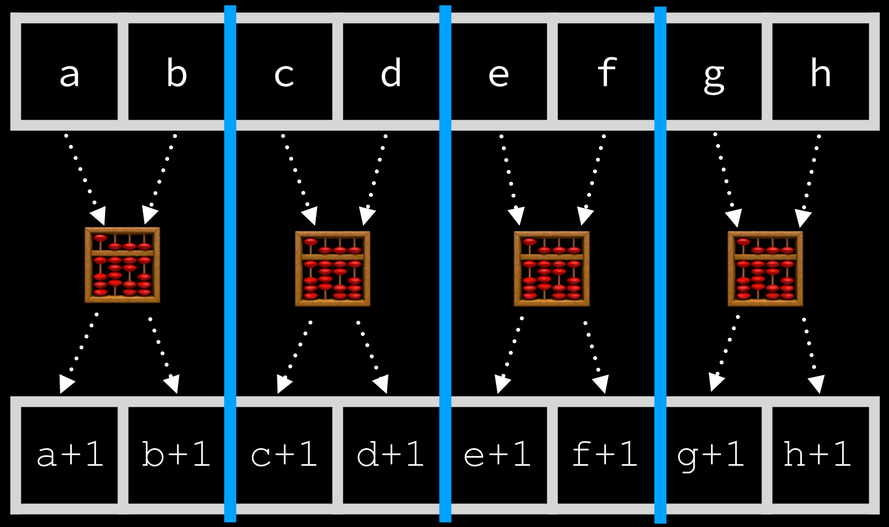
memory banking

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!

give MrXL a `map` operation!

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

give MrXL a `map` operation!

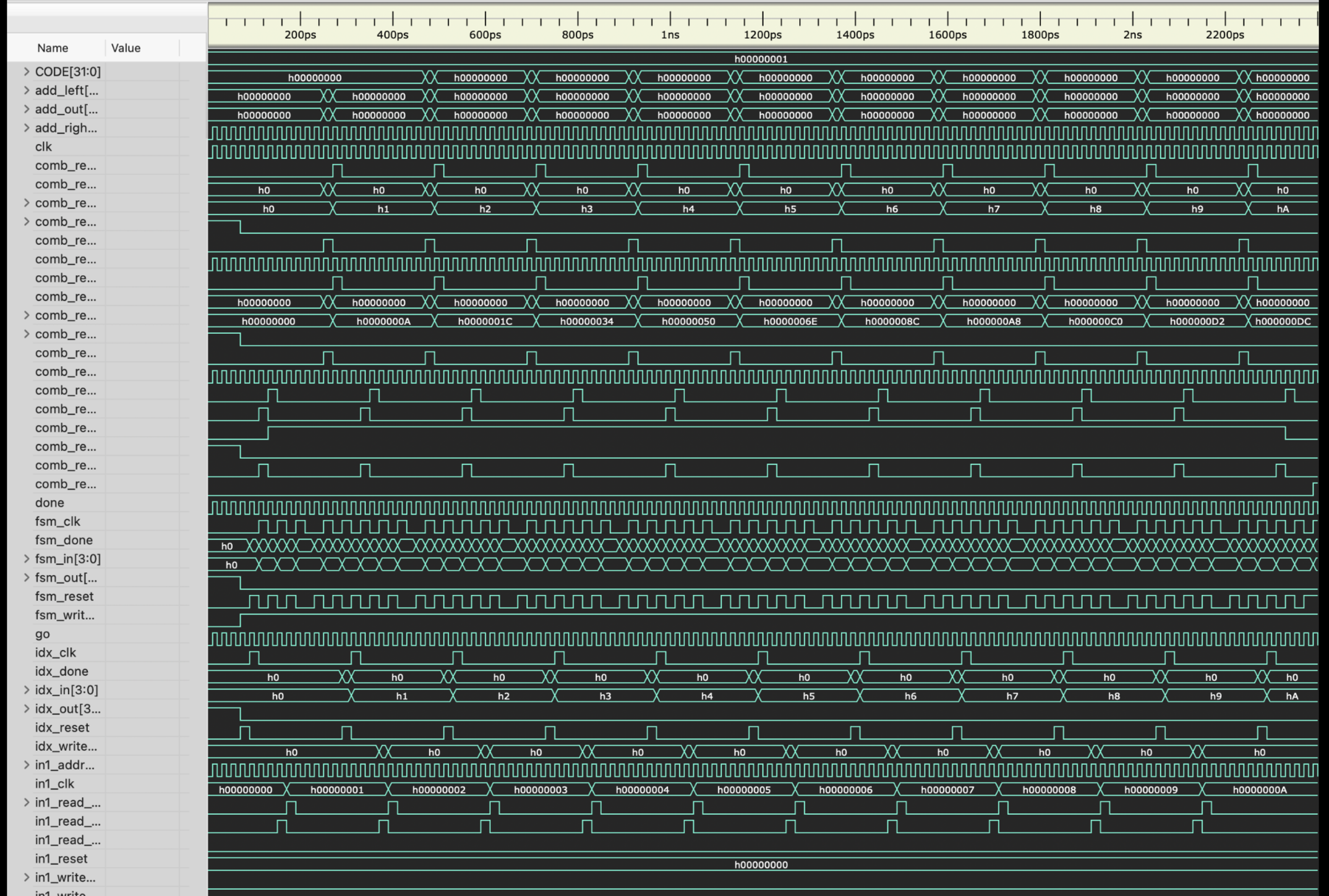
Psst: consider implementing just `add` first, end-to-end

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

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

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

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

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

```
read_mem
```

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

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

```
read_mem
  do_mul
```

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
```

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

```
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
```

```
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
```

```
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
```

```
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
```

```
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
```

```
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
```

```
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);
}
```

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

the bug

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

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

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;
}

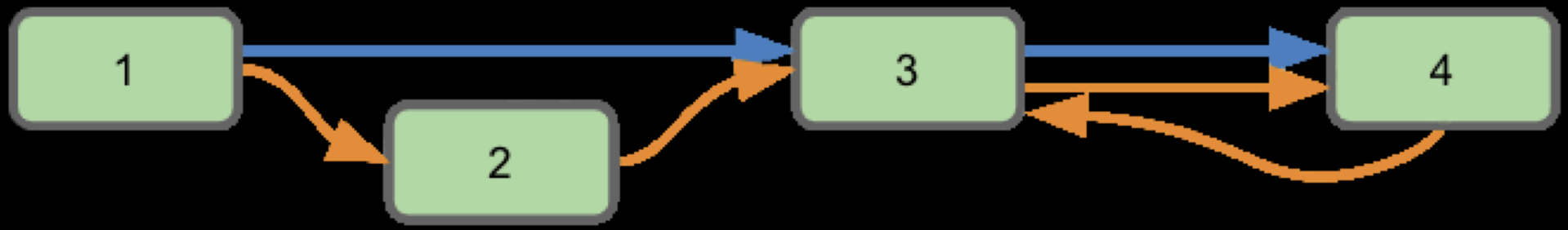
using Cider

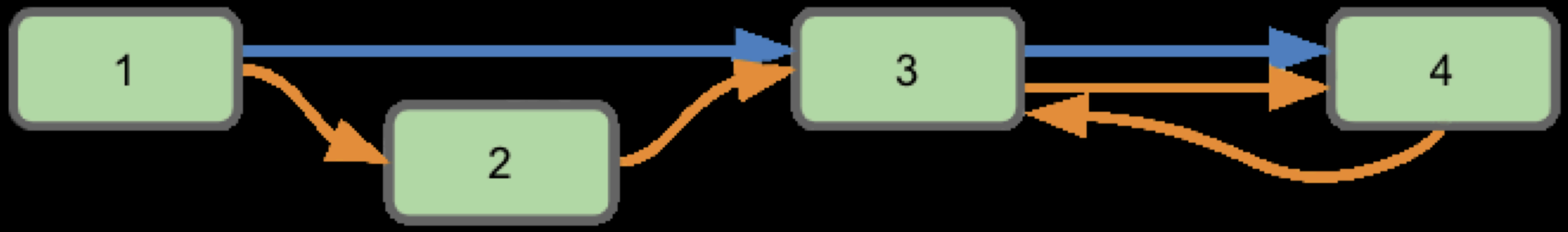
```
fud e --from mrxl test/sos.mrxl --to interpreter-out
```

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

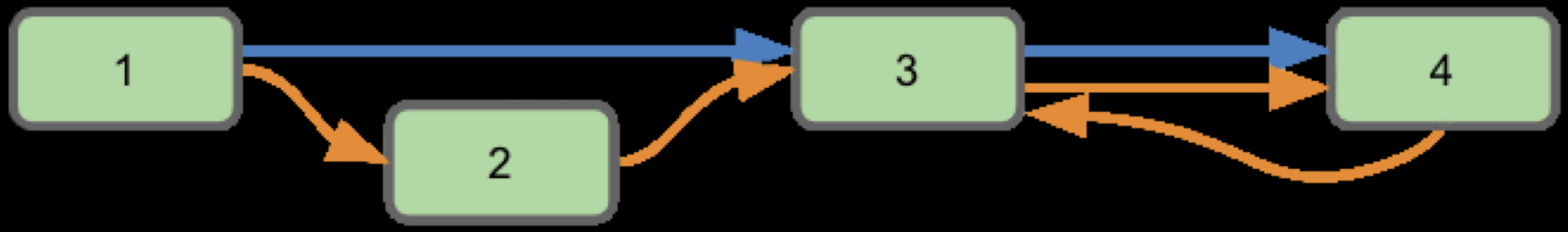
Read more: https://docs.calyxir.org/debug/cider.html


poxen


1
2
3
4




```
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;
}
```

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

```
exine depth -n 2
```

=

```
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

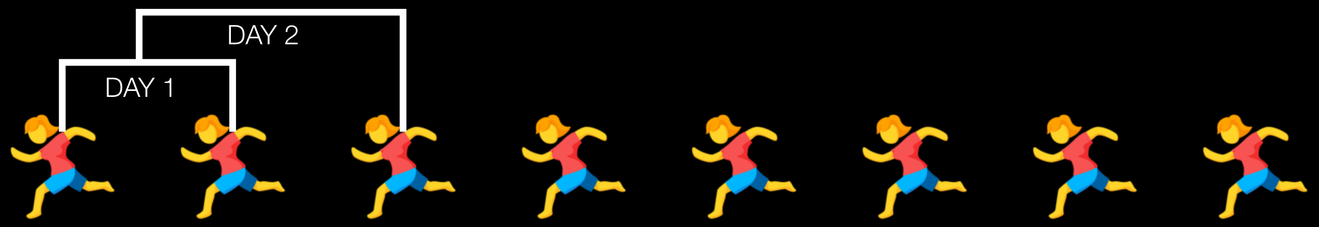
How best to run a tennis tournament?

reduction trees

How best to run a tennis tournament?

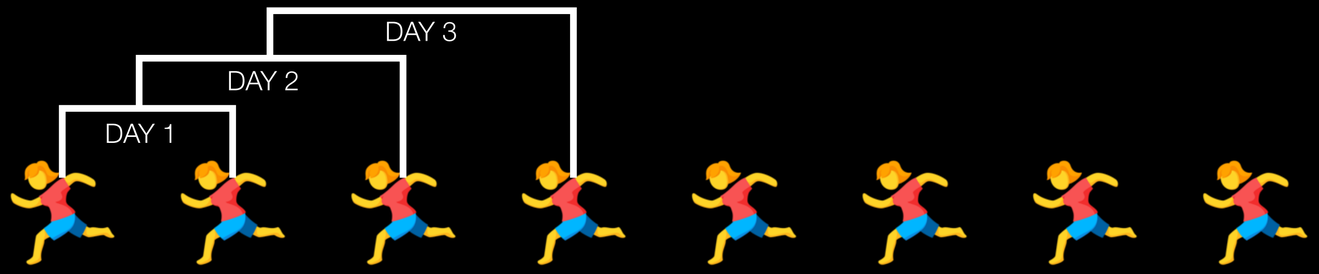
reduction trees

How best to run a tennis tournament?



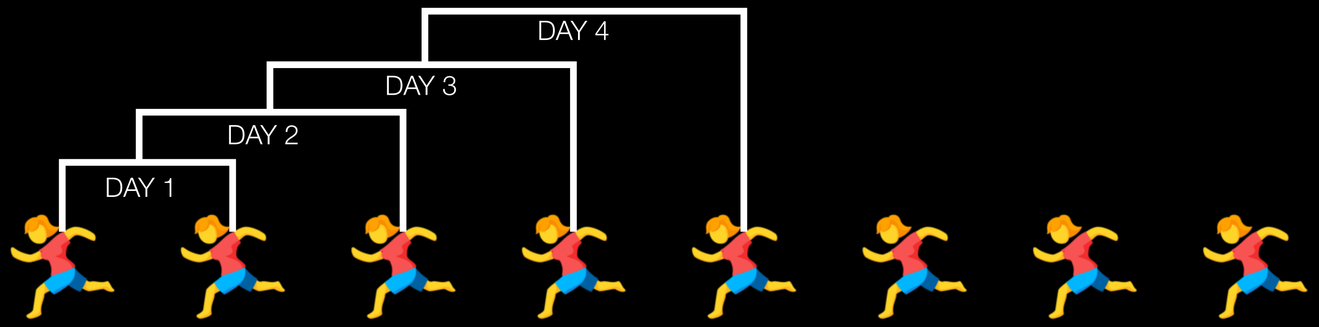
reduction trees

How best to run a tennis tournament?



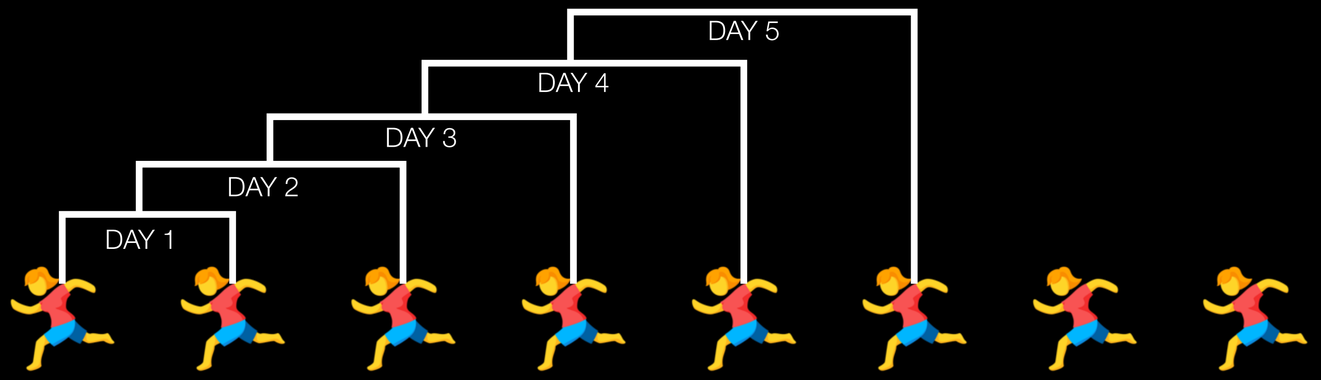
reduction trees

How best to run a tennis tournament?



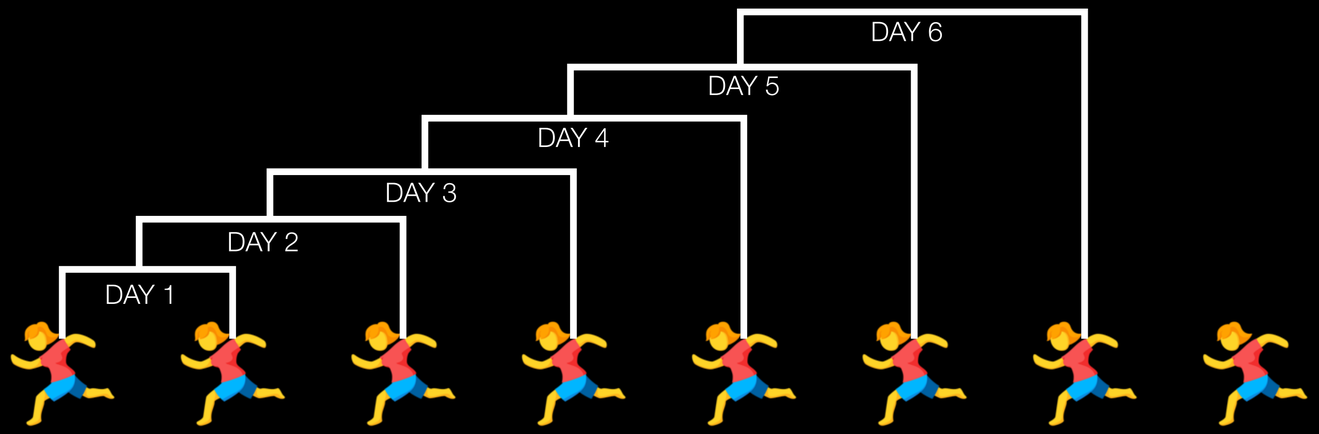
reduction trees

How best to run a tennis tournament?



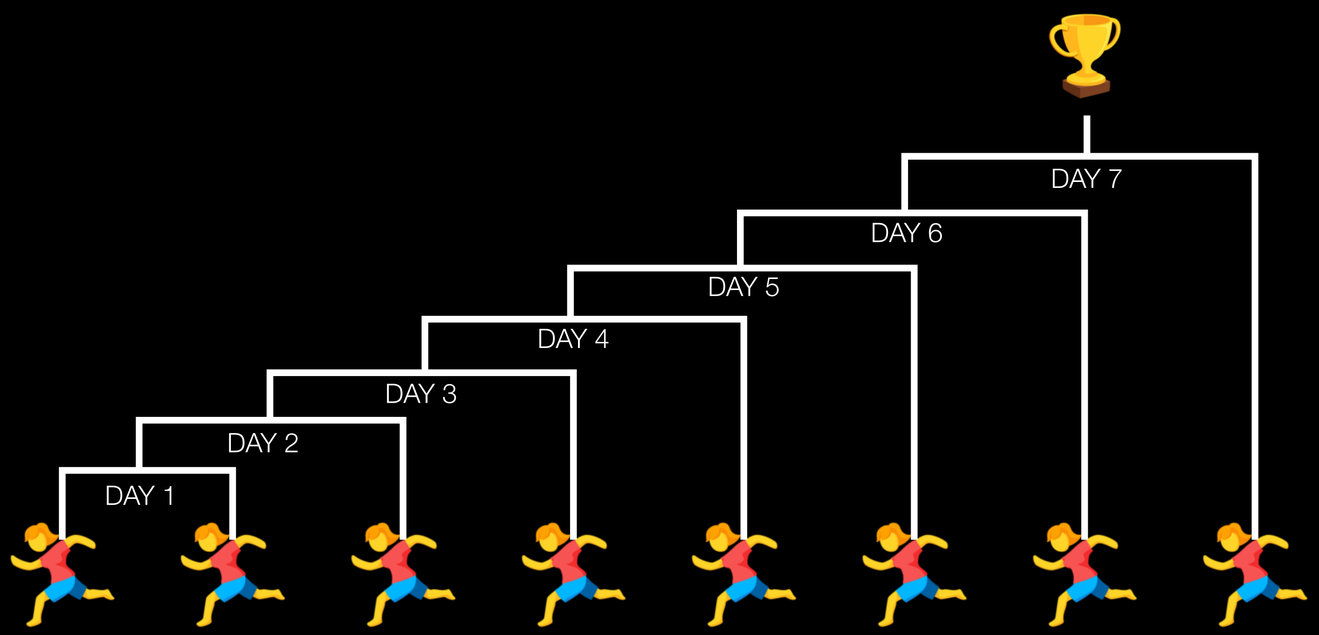
reduction trees

How best to run a tennis tournament?



reduction trees

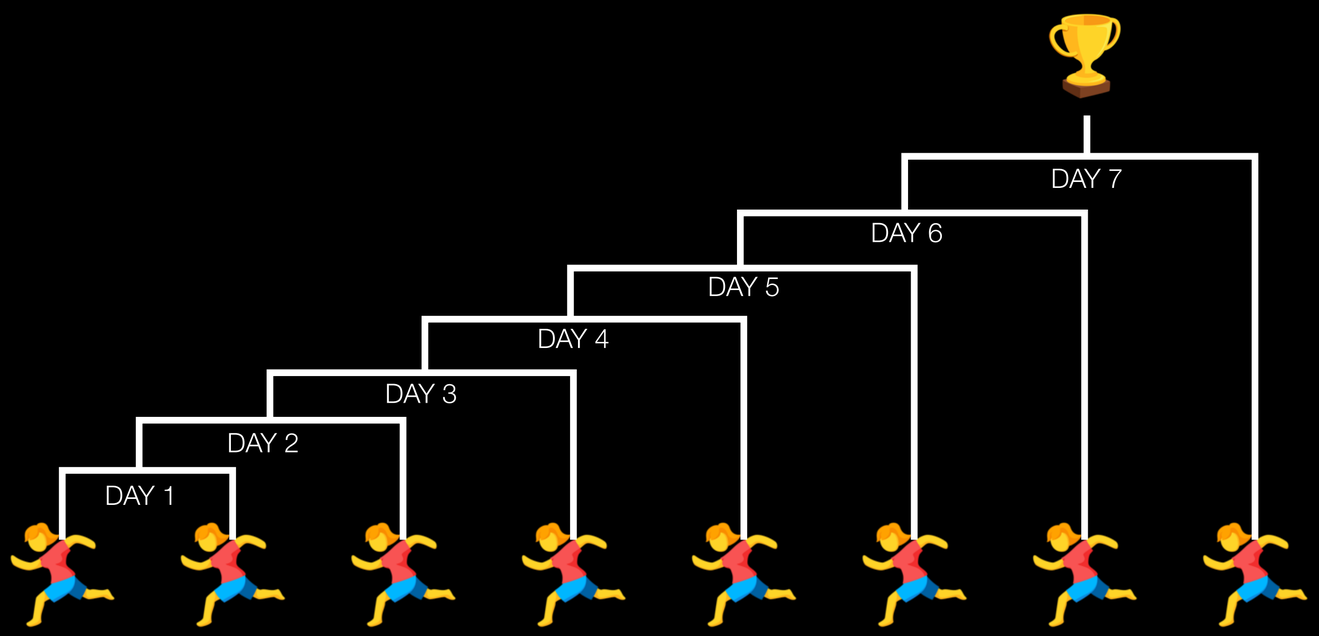
How best to run a tennis tournament?



reduction trees

How best to run a tennis tournament?

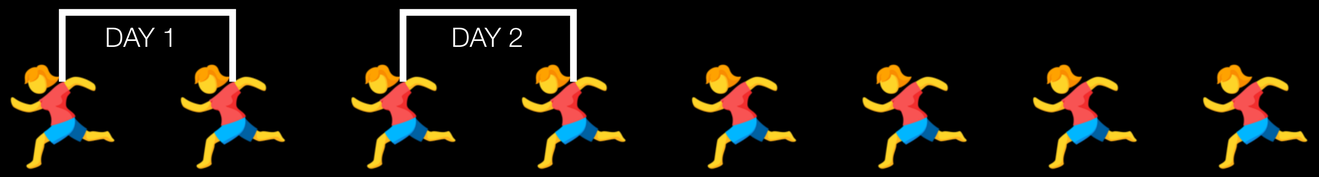
This clearly has problems…



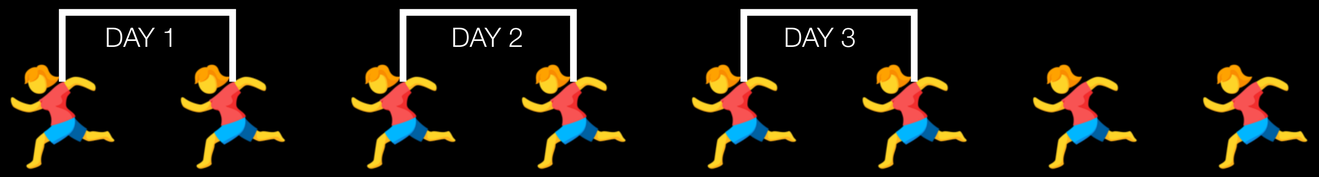
reduction trees

reduction trees

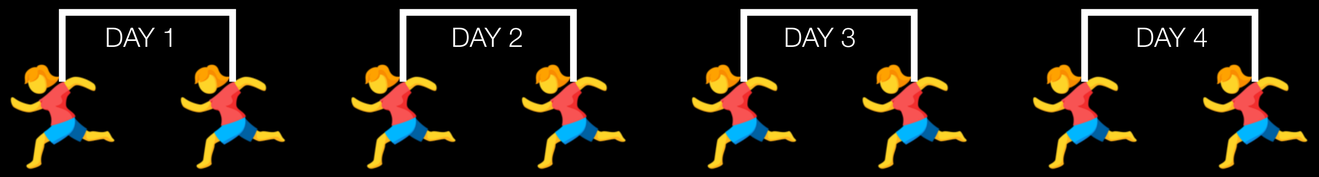
reduction trees



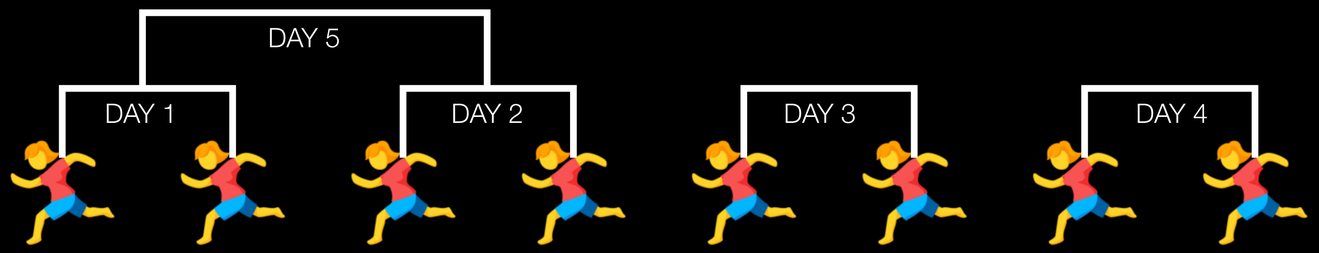
reduction trees



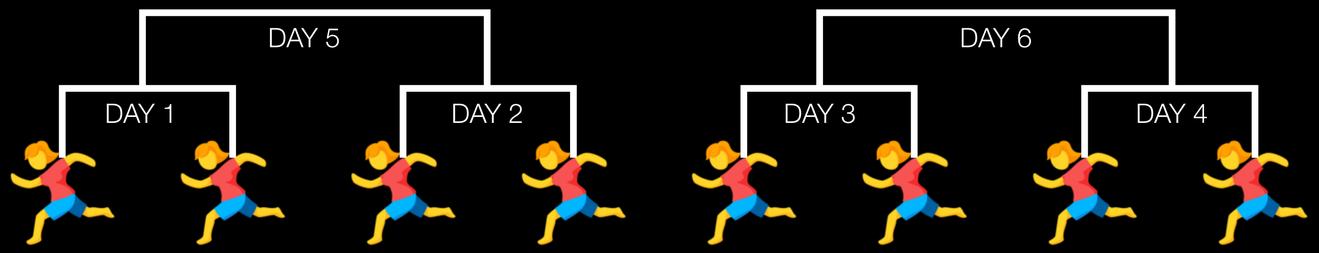
reduction trees



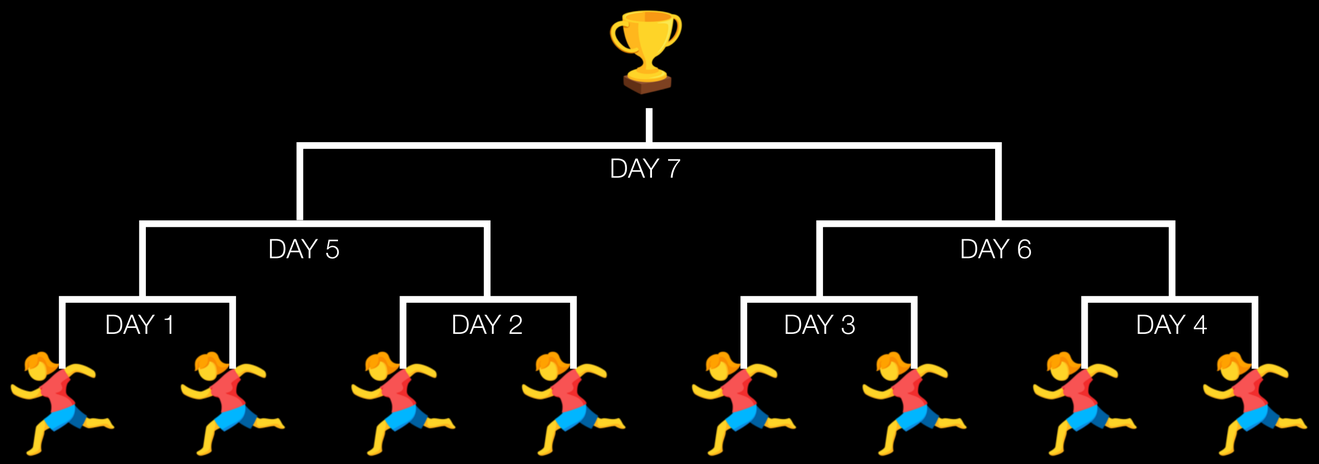
reduction trees



reduction trees

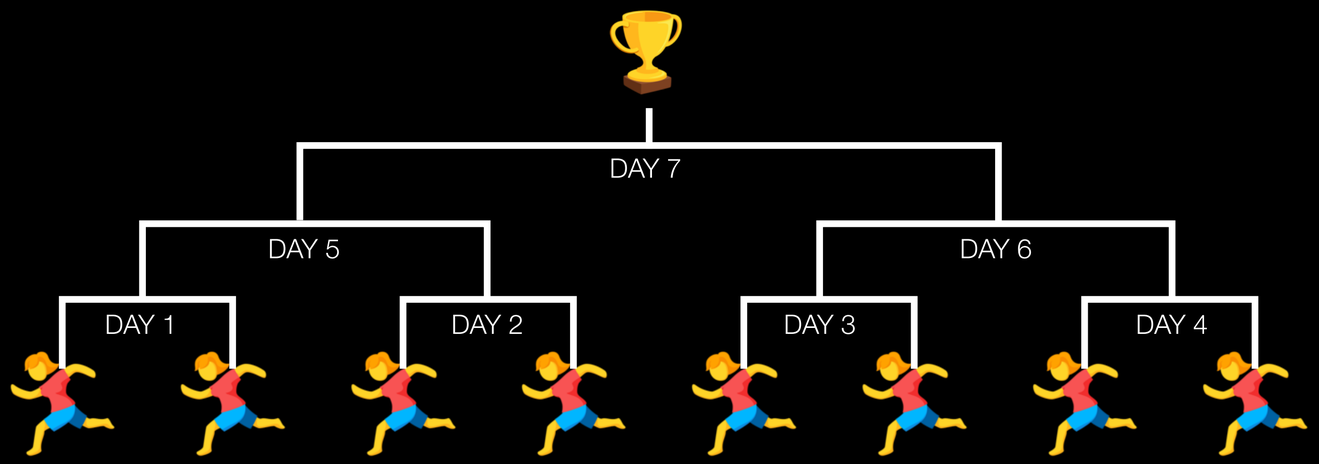


reduction trees



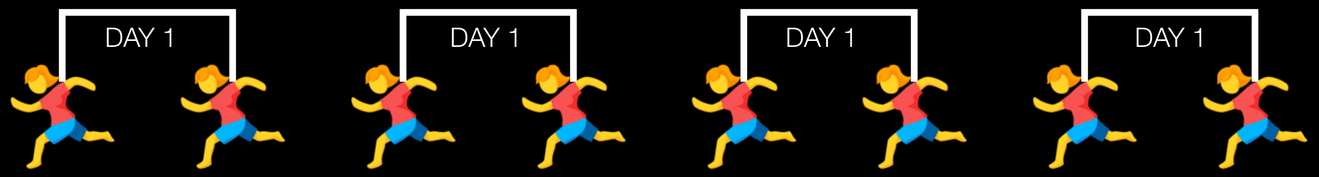
reduction trees

A little better!

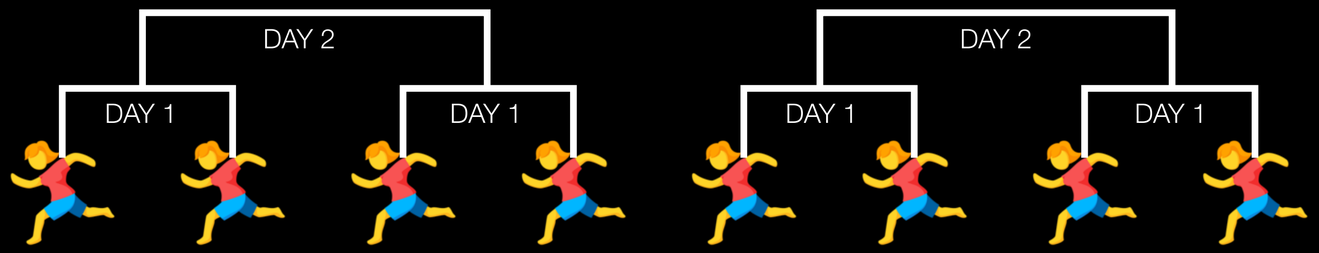


reduction trees

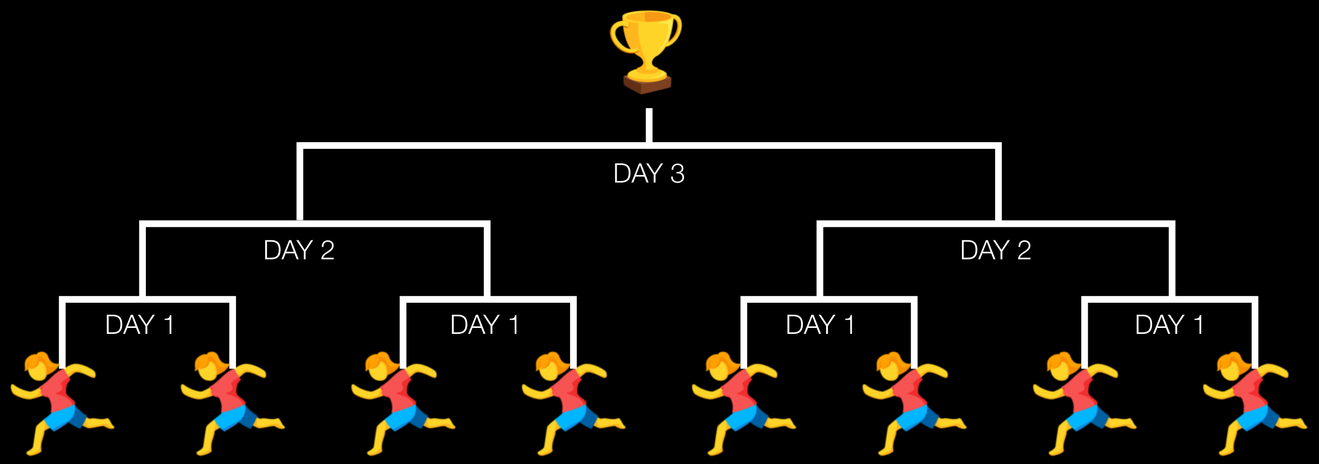
reduction trees



reduction trees

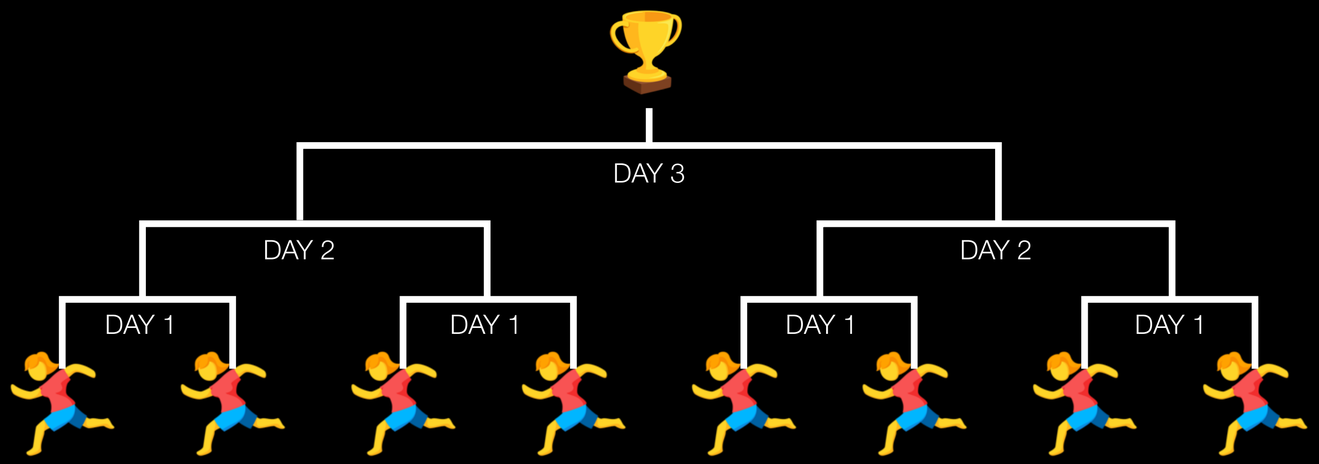


reduction trees



reduction trees

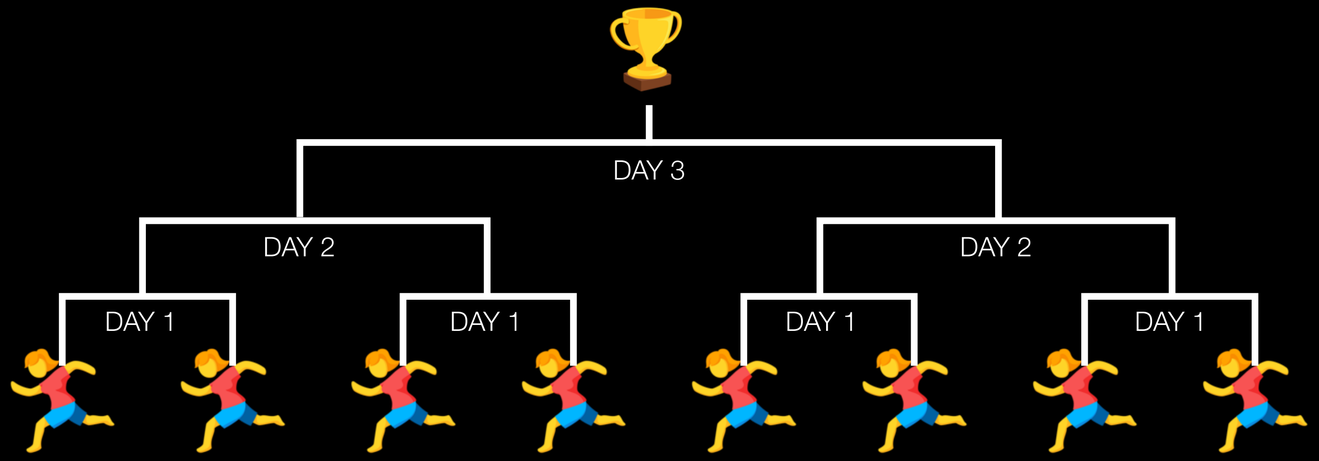
Better still…



reduction trees

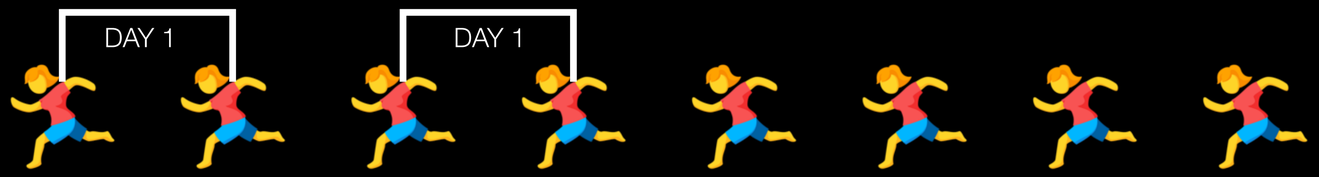
Better still…

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

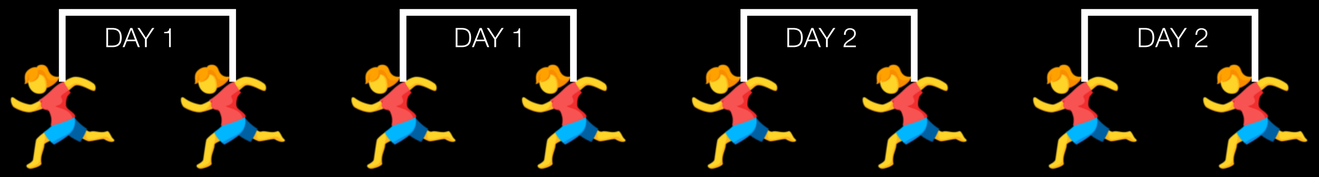


reduction trees

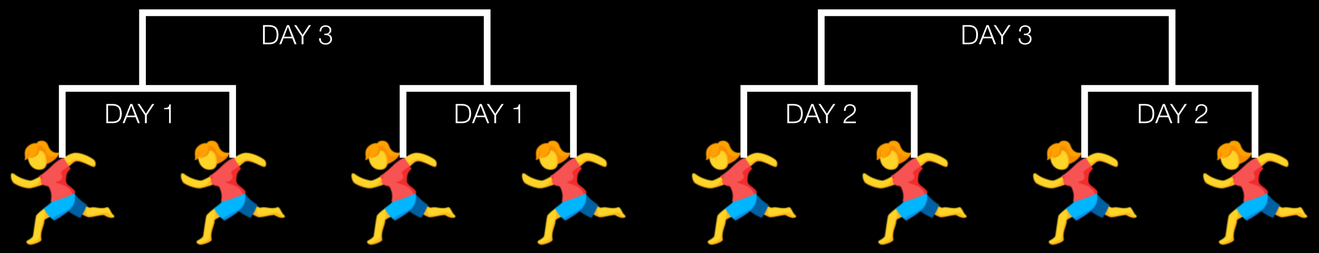
reduction trees



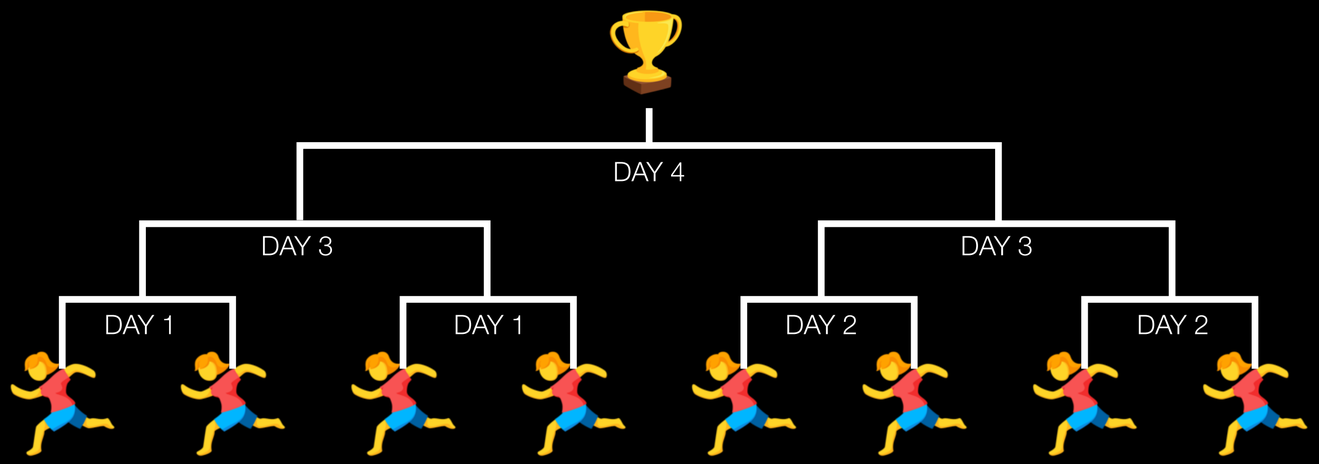
reduction trees



reduction trees

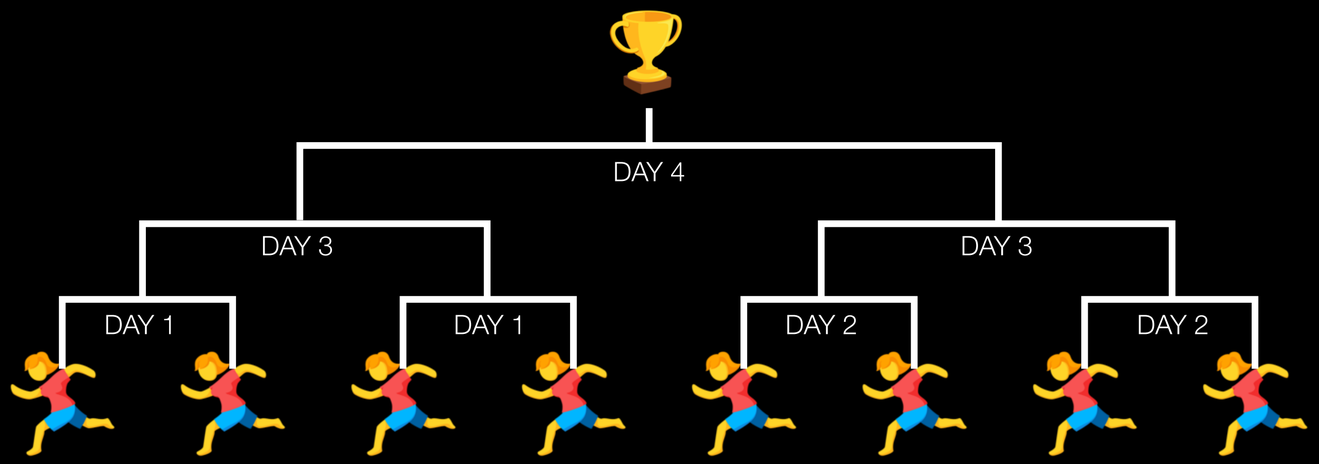


reduction trees



reduction trees

We can do this with two courts!

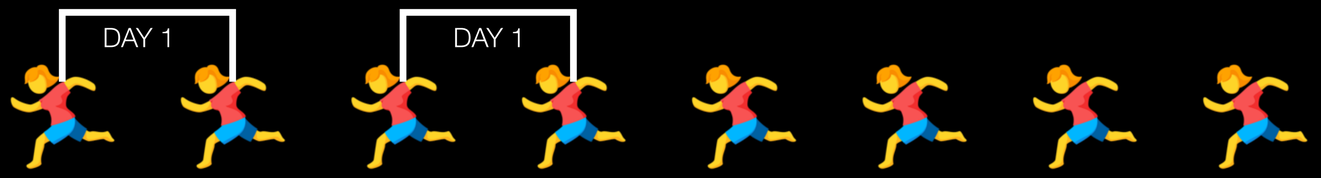


reduction trees

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

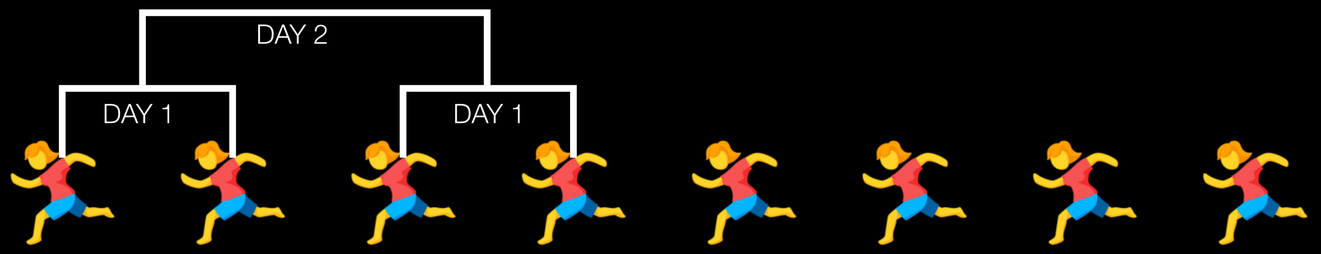
reduction trees

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



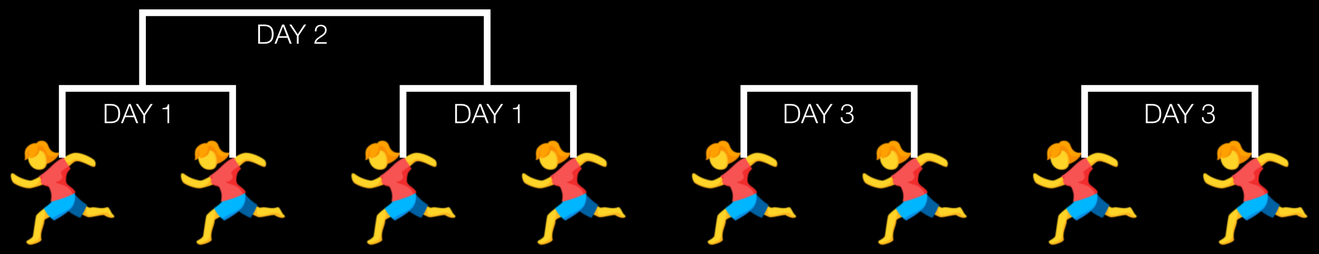
reduction trees

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



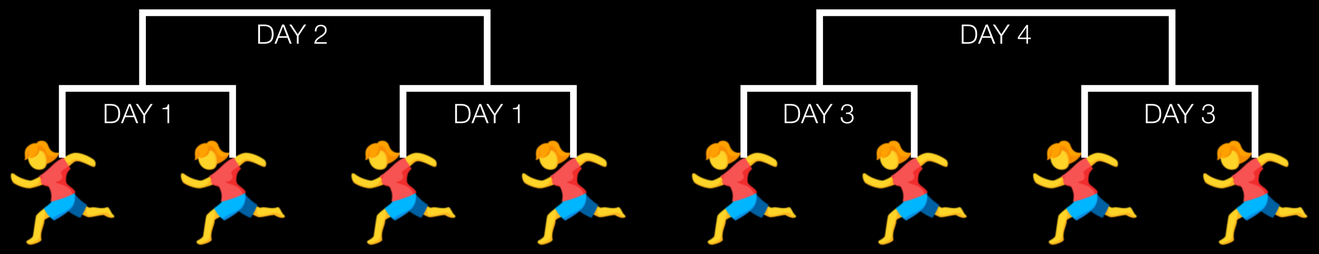
reduction trees

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



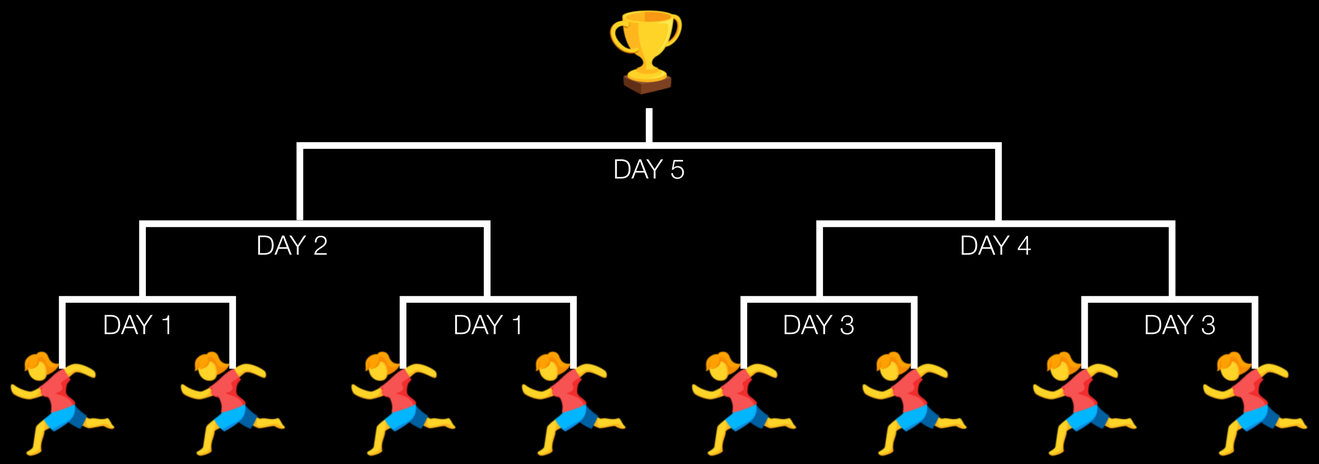
reduction trees

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



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!

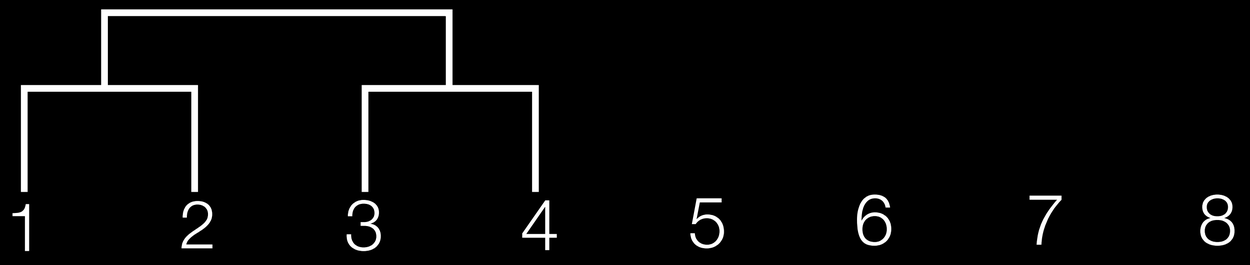
1 2 3 4 5 6 7 8

reduction trees

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

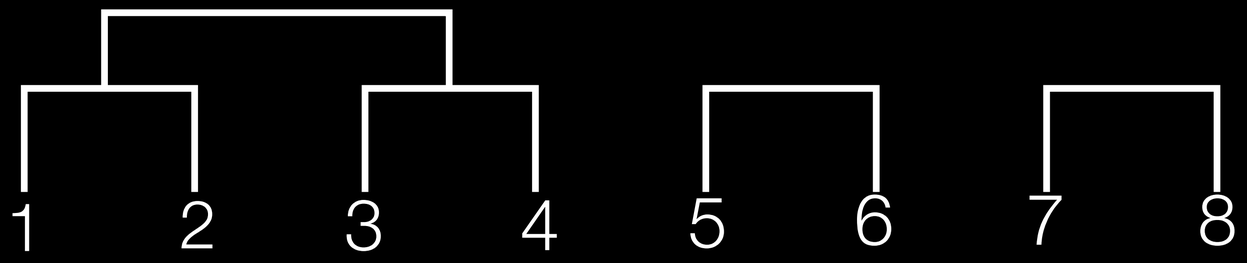
reduction trees

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



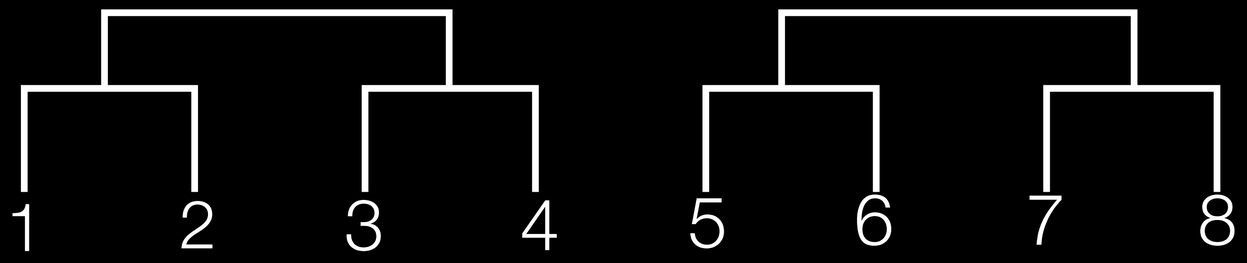
reduction trees

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



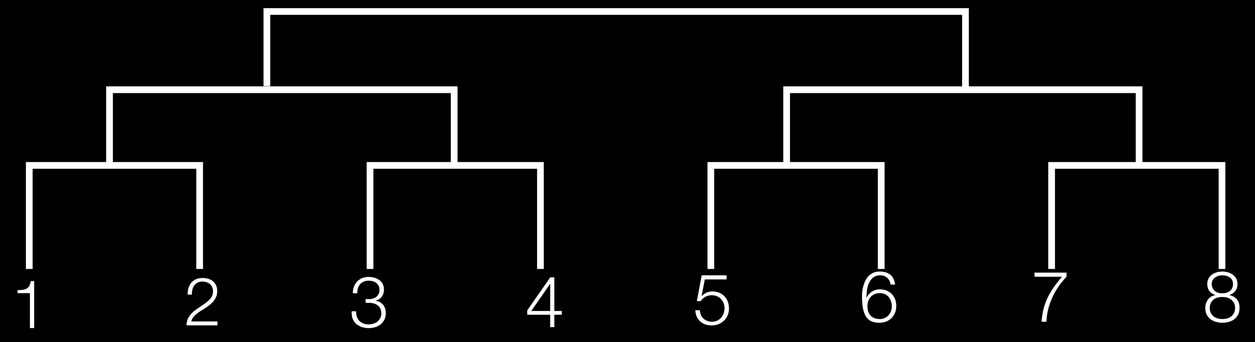
reduction trees

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



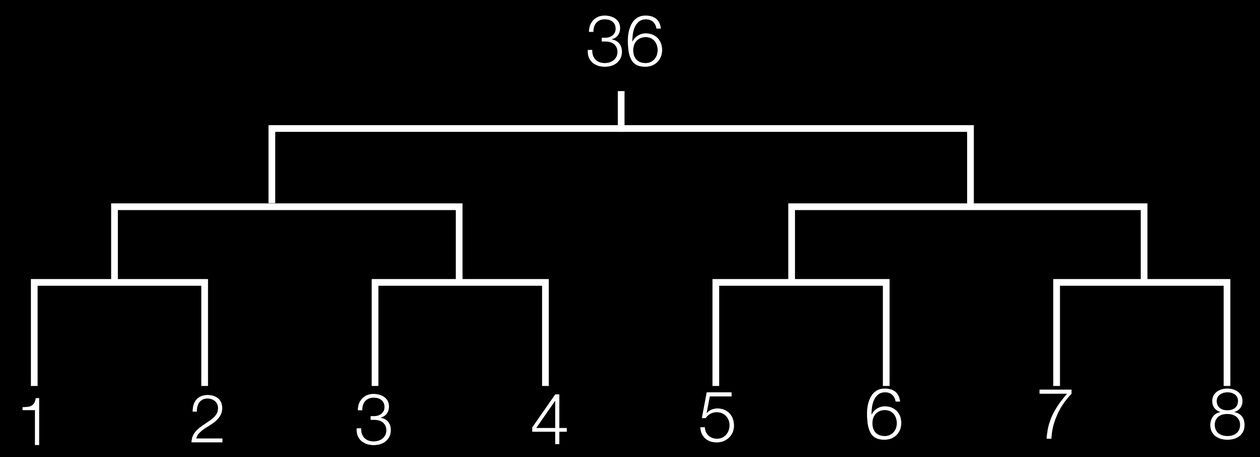
reduction trees

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



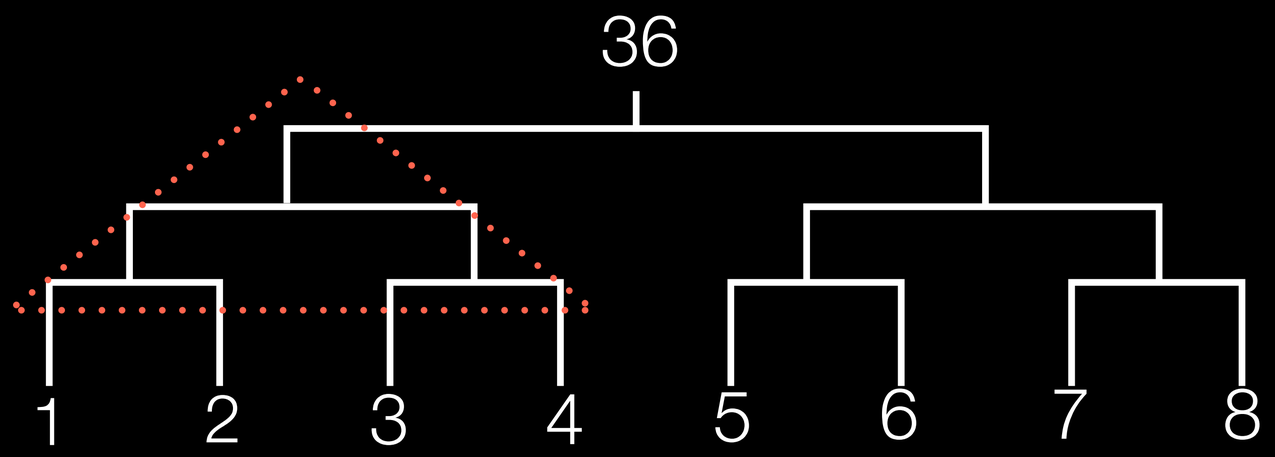
reduction trees

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



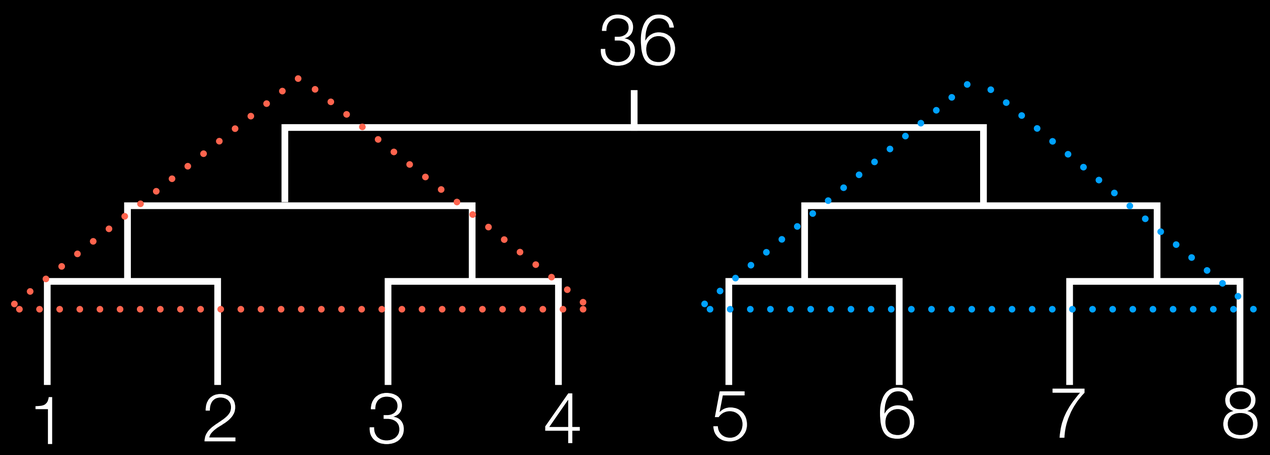
reduction trees

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



reduction trees

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



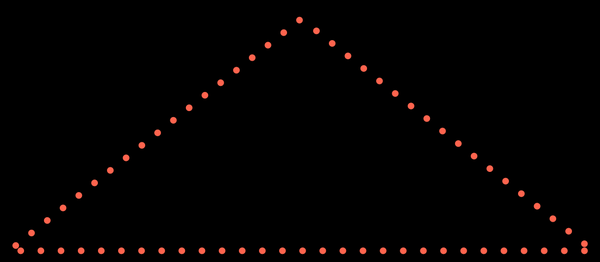
reduction trees

How would you handle 16 numbers?

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

reduction trees

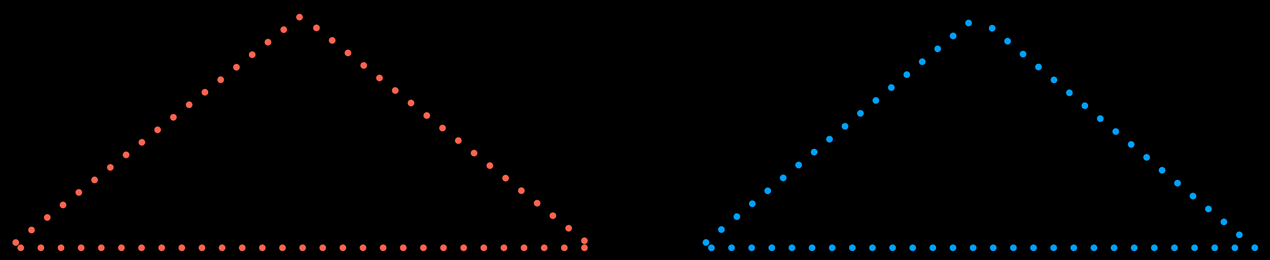
How would you handle 16 numbers?



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

reduction trees

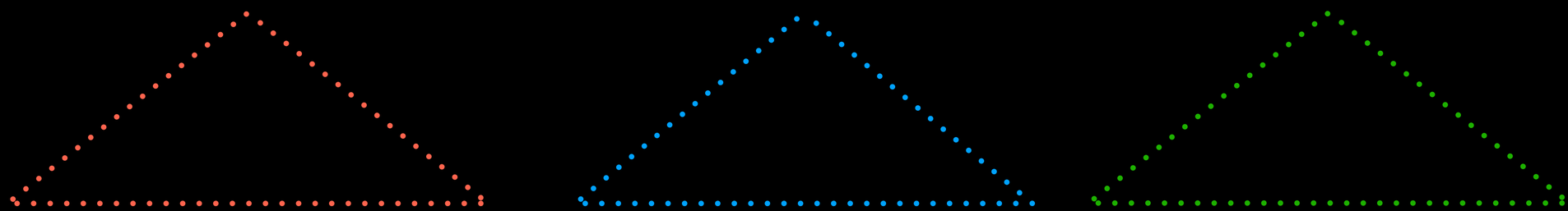
How would you handle 16 numbers?



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

reduction trees

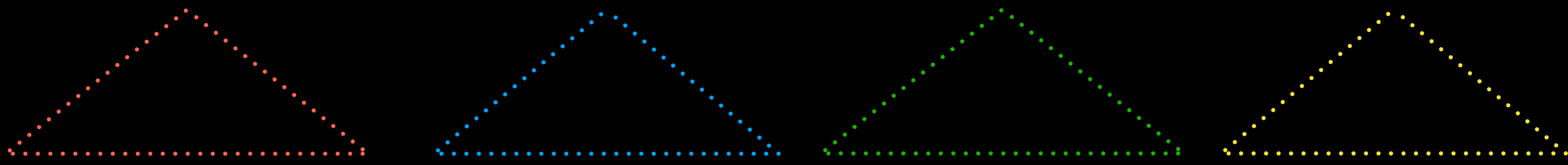
How would you handle 16 numbers?



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

reduction trees

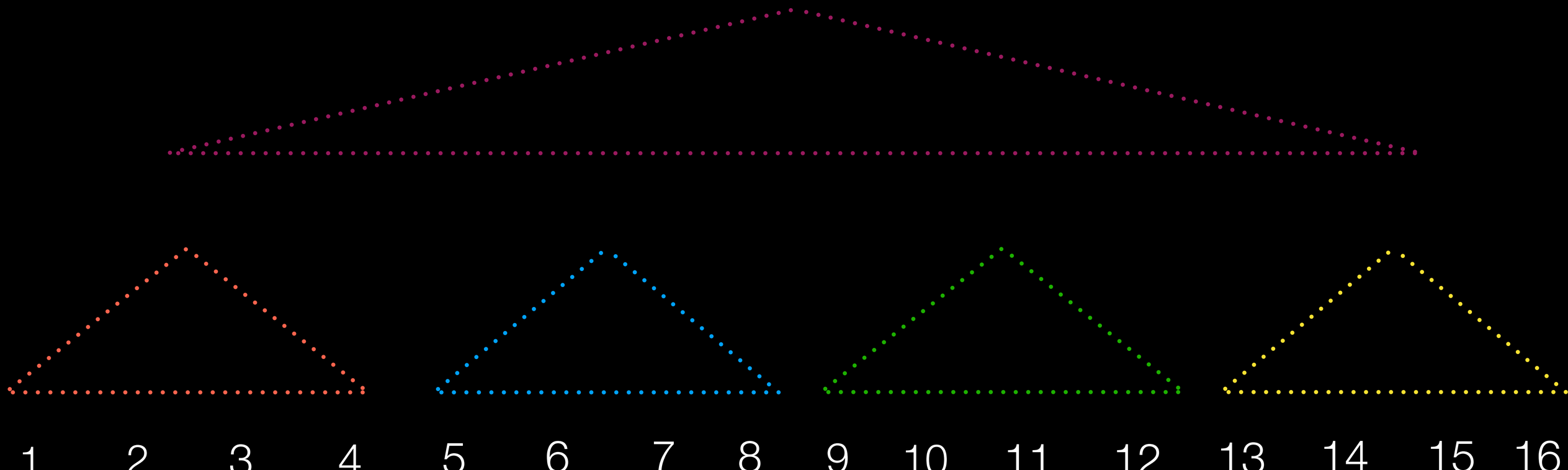
How would you handle 16 numbers?



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

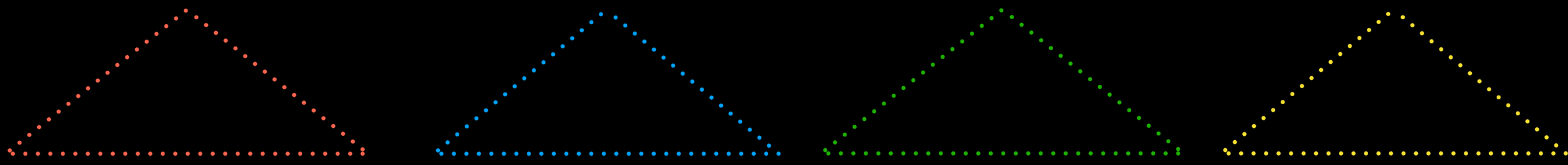
reduction trees

How would you handle 16 numbers?



reduction trees, feat. banking

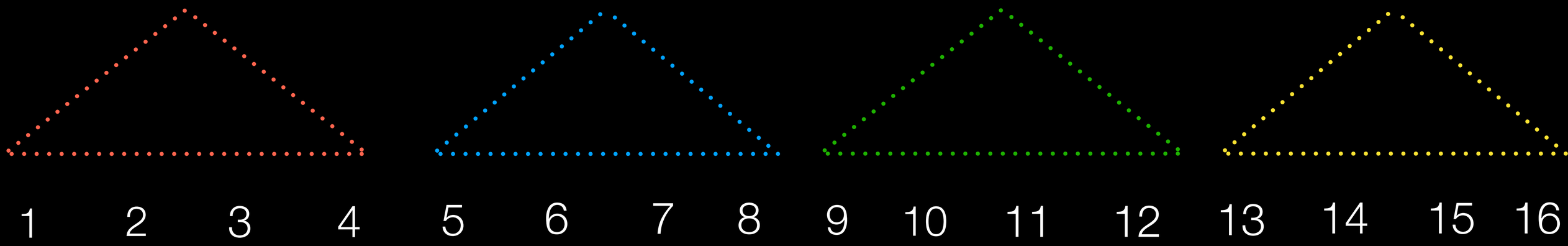
Where would you place the 16 numbers?



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

reduction trees, feat. banking

Where would you place the 16 numbers?

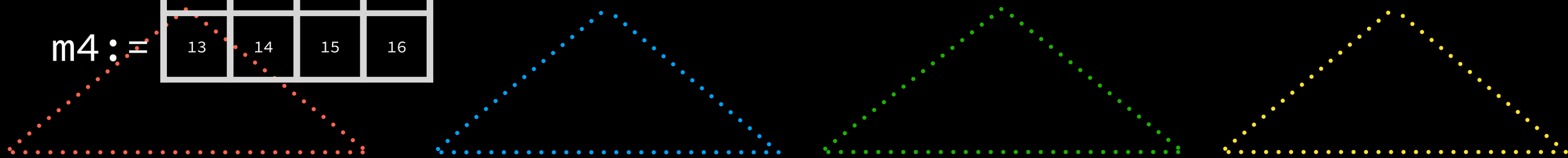


reduction trees, feat. banking

Where would you place the 16 numbers?

mem:=	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

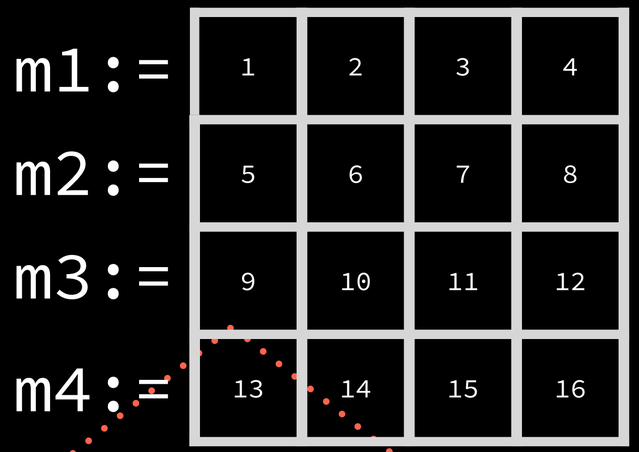
m1:=	1	2	3	4
m2:=	5	6	7	8
m3:=	9	10	11	12
m4:=	13	14	15	16

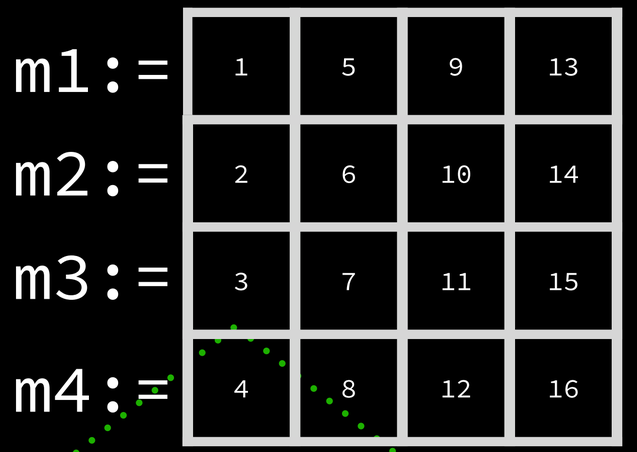


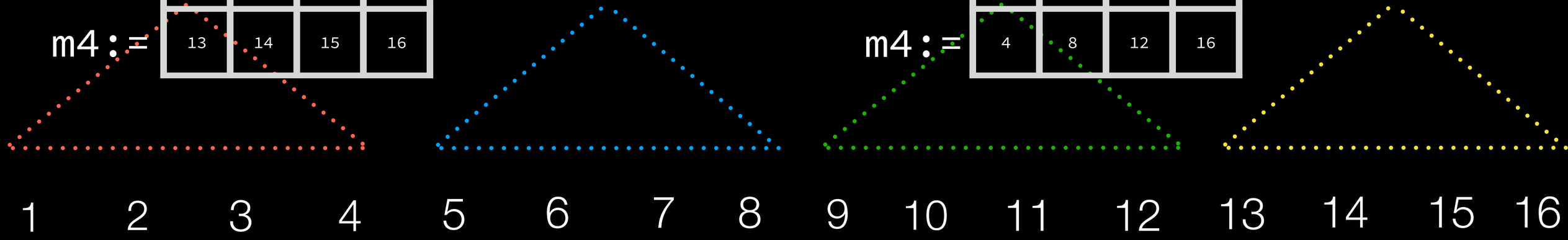
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

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?

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 }
```

arbitrage

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


```
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.

arbitrage

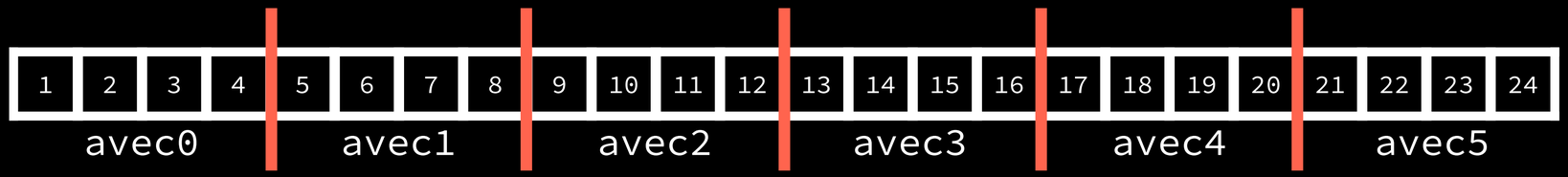
What if I want to parallelize the same memory, but with different banking factors?

```
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.

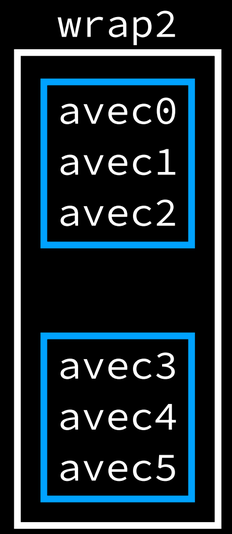


arbitrage

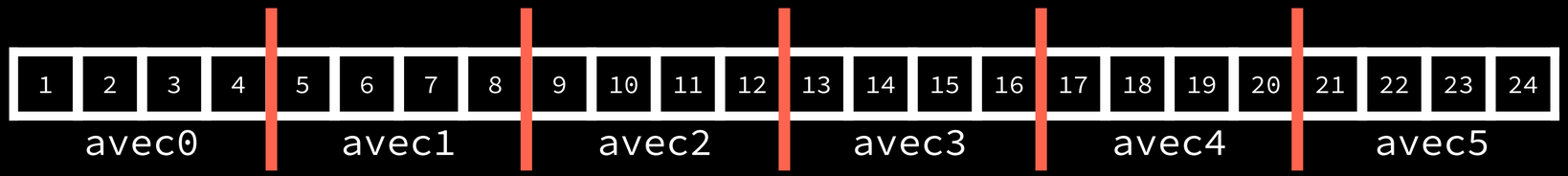
What if I want to parallelize the same memory, but with different banking factors?


```
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.



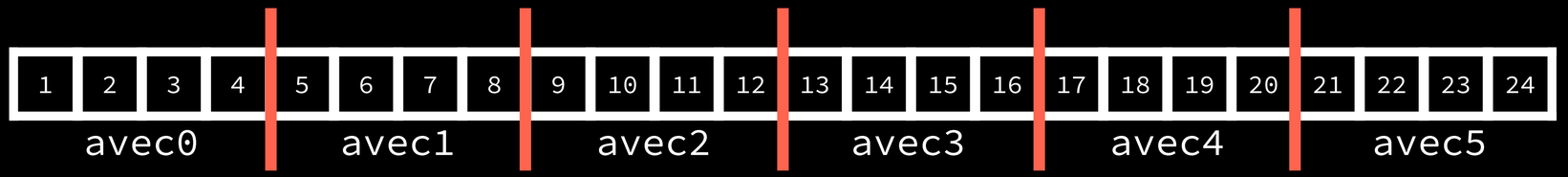
arbitrage

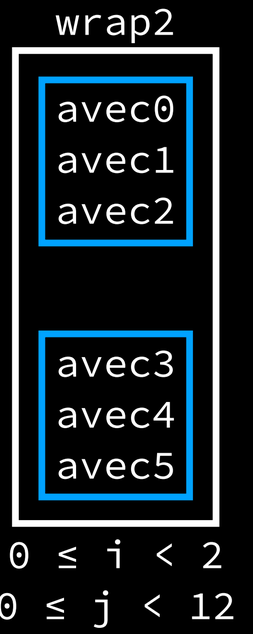
What if I want to parallelize the same memory, but with different banking factors?


```
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.





$0 \le i < 2$

$0 \le j < 12$

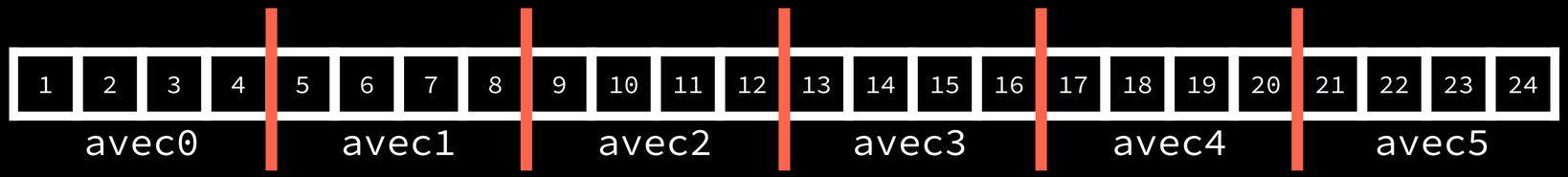
arbitrage

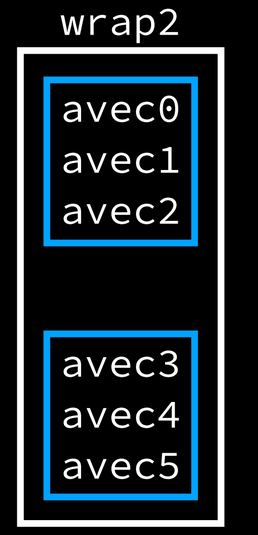
What if I want to parallelize the same memory, but with different banking factors?


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

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

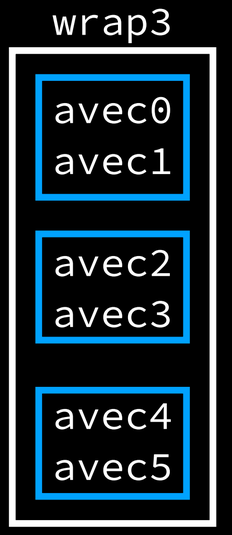
We need to break `avec` into `lcm(2,3)=6` banks, and then arbitrate between those.





$0 \le i < 2$

$0 \le j < 12$



arbitrage

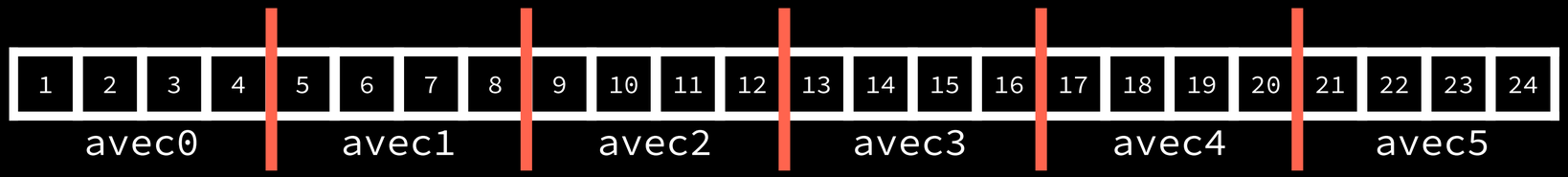
What if I want to parallelize the same memory, but with different banking factors?

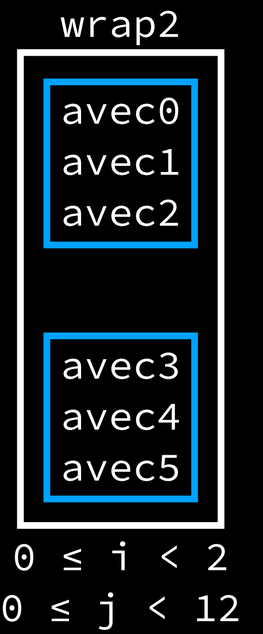
```
avec :=
```

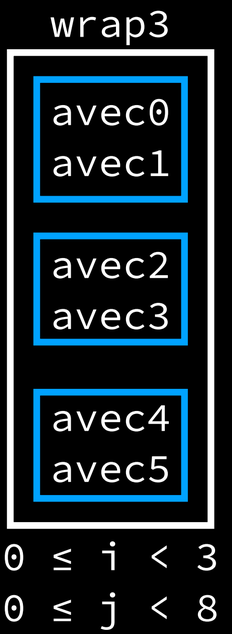

```
squares := map 3 (a <- avec) { a * a }

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

- you can work on calyx frontends/calyx/calyx backends
- here's people working on calyx!
- verification, optimization, lots of buzzwords :D