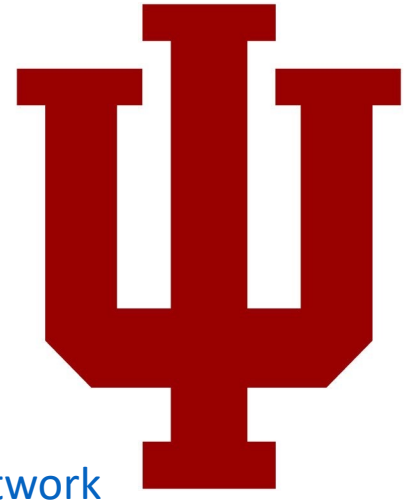
16: Hardware Acceleration III

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University



Some material taken from:
https://github.com/trekhleb/homemade-machine-learning/tree/master/homemade/neural_network
http://cs231n.github.io/neural-networks-1/

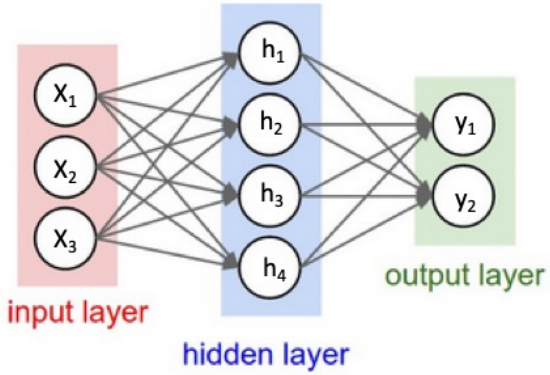
Announcements

- P6 due Friday! DMA in C
- P7 is out
  - Maybe not AG yet?

Project 7: Accelerate Dot Product

- Given: Slow Dot-Product in Hardware
- Your Task: Accelerate it!
- How? Pipelining + Parallelism (later)
- Groups of 2 allowed.

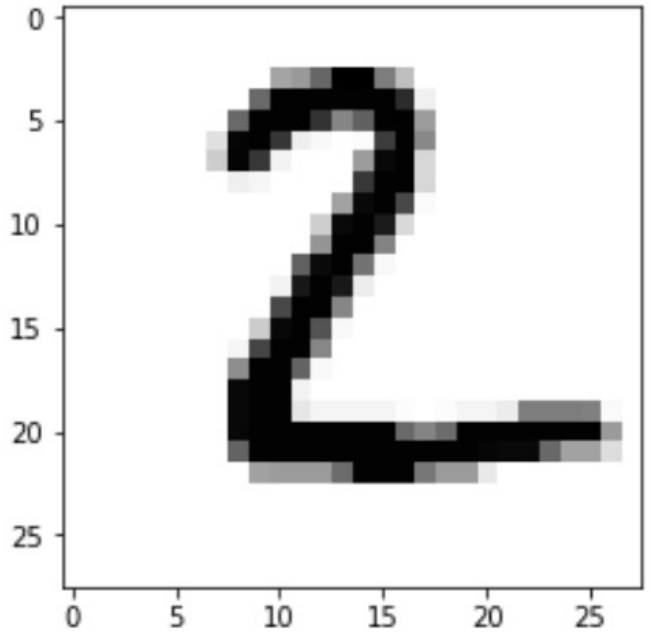
Why Dot Product?




```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Simple Neural Network

```
================================
Index: 0
Image:
```




```
ML Classification Result: 2
Real Value: 2
Correct Result: True
================================
```

- Takes in image of number
- Returns integer value
- How? artificial neural network

Matrix Multiplication (Dot Product)

$$\begin{bmatrix} i_0 & i_1 \end{bmatrix} \times \begin{bmatrix} w_{00} & w_{10} & w_{20} \\ w_{01} & w_{11} & w_{21} \end{bmatrix} = \begin{bmatrix} o_0 & o_1 & o_2 \end{bmatrix}$$

$$o_0 = i_0 \cdot w_{00} + i_1 \cdot w_{01}$$

$$o_1 = i_0 \cdot w_{10} + i_1 \cdot w_{11}$$

$$o_2 = i_0 \cdot w_{20} + i_1 \cdot w_{21}$$

Alternative Dot Computations

$$[0.1 \quad 0.2] \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = [0.9 \quad 1.2 \quad 1.5]$$

$[0 \quad 0 \quad 0]$ ← temp result

$0.1 \cdot [1 \quad 2 \quad 3] + [0 \quad 0 \quad 0] =$

$[0.1 \quad 0.2 \quad 0.3] + [0 \quad 0 \quad 0] = [0.1 \quad 0.2 \quad 0.3]$ ← temp result

$0.2 \cdot [4 \quad 5 \quad 6] + [0.1 \quad 0.2 \quad 0.3] =$

$[0.8 \quad 1.0 \quad 1.2] + [0.1 \quad 0.2 \quad 0.3] = [0.9 \quad 1.2 \quad 1.5]$

Floating-Point math takes 8 cycles.

- Floating-Point is complicated.
- How do we work around an 8 cycle latency?
- Pipelining!

input:

(2) 10

(3) 11

6

cycle1

10

* 1

10

cycle2

10

* 10

100

cycle3

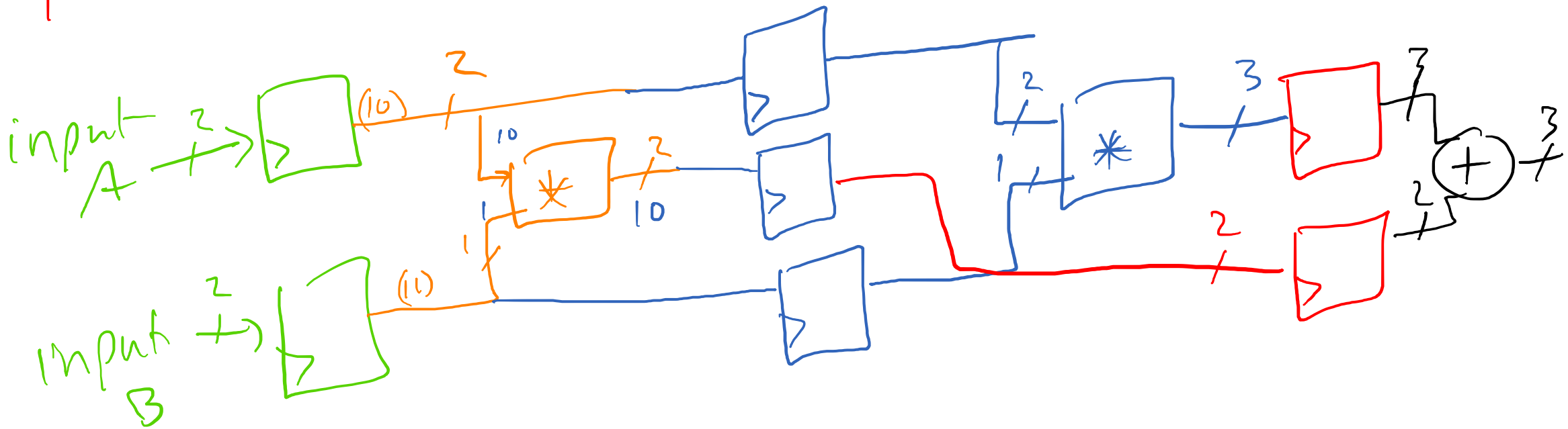
10

+100

110

output

110 (6)



Pipelining

- FMAC takes 8 cycles for 1 value
- But can accept a new value every cycle.

- Latency: 8 cycles / value
- Throughput: 1 value / cycle

Latency vs. Throughput

- Latency: How long does an individual operation take?
- Throughput: How many operations can you complete in a given time?

Pipelining

Pipelined Dot Computations

$$\begin{bmatrix} 0.1 & 0.2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 0.9 & 1.2 & 1.5 \end{bmatrix}$$

Problems with Pipelined FMAC

Latency!

Latency on Pipelined FMAC

- Solution: Stall at the end of a row.
- Drain the pipeline.

Parallelize Alternative Dot Computations?

$$[0.1 \quad 0.2] \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = [0.9 \quad 1.2 \quad 1.5]$$

$$[0.1] \; [1 \; 2 \; 3] = [0.1 \quad 0.2 \quad 0.3]$$

1 FMAC

$$[0.2] \; [4.5.6] + [0.8 \quad 1.0 \quad 1.2]$$

$$[0.9 \quad 1.2 \quad 1.5]$$

1 FMAC

+1 ADD

Parallelize Alternative Dot Computations?

$$[0.1 \quad 0.2] \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = [0.9 \quad 1.2 \quad 1.5]$$

$[0, 0, 0]$ ← result

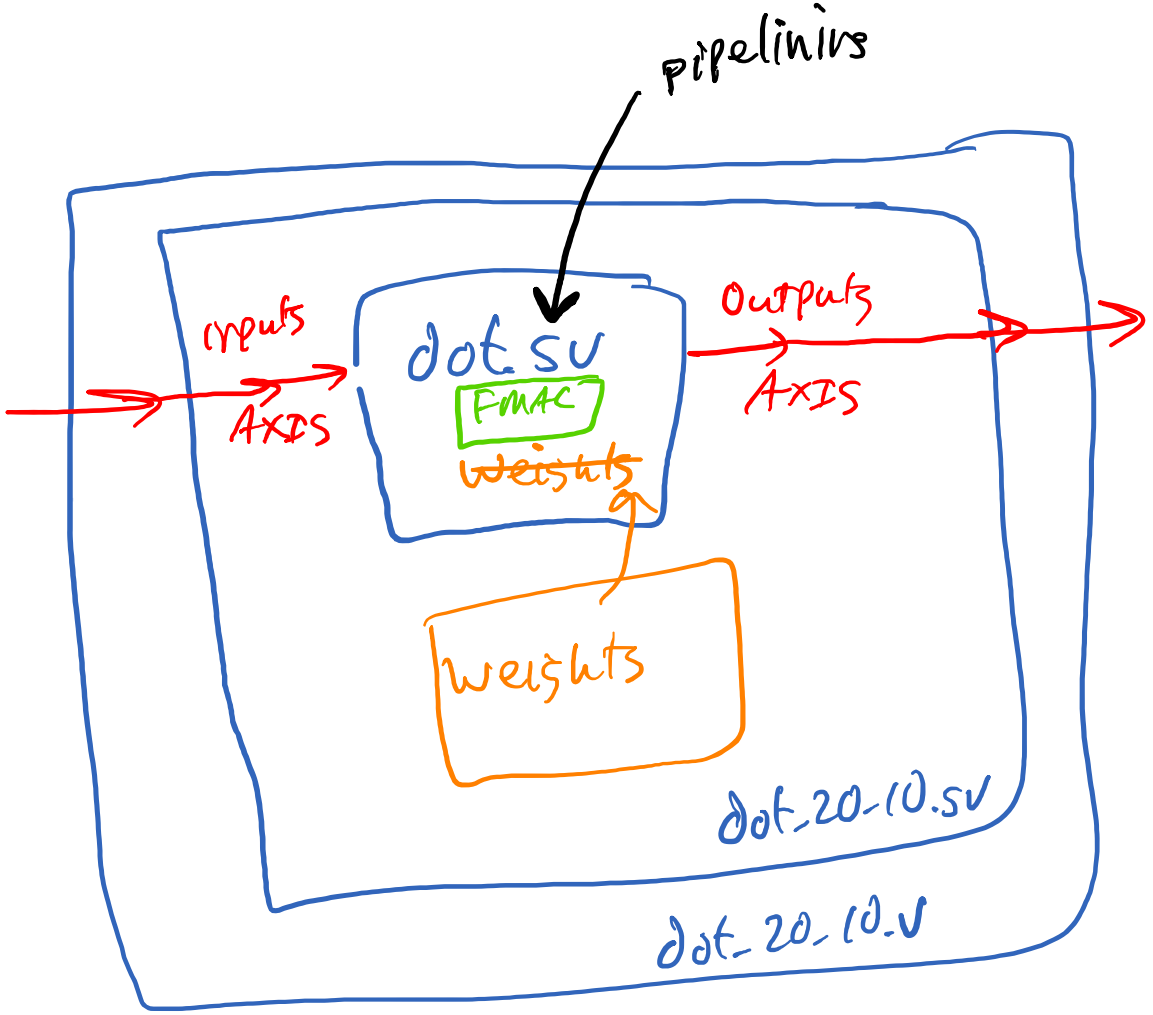
$0.1 \cdot [1\ 2\ 3] + [0\ 0\ 0] =$

$[0.1\ 0.2\ 0.3] + [0\ 0\ 0] = [0.1\ 0.2\ 0.3]$ ← temp result

$0.2 \cdot [4\ 5\ 6] + [0.1\ 0.2\ 0.3] =$

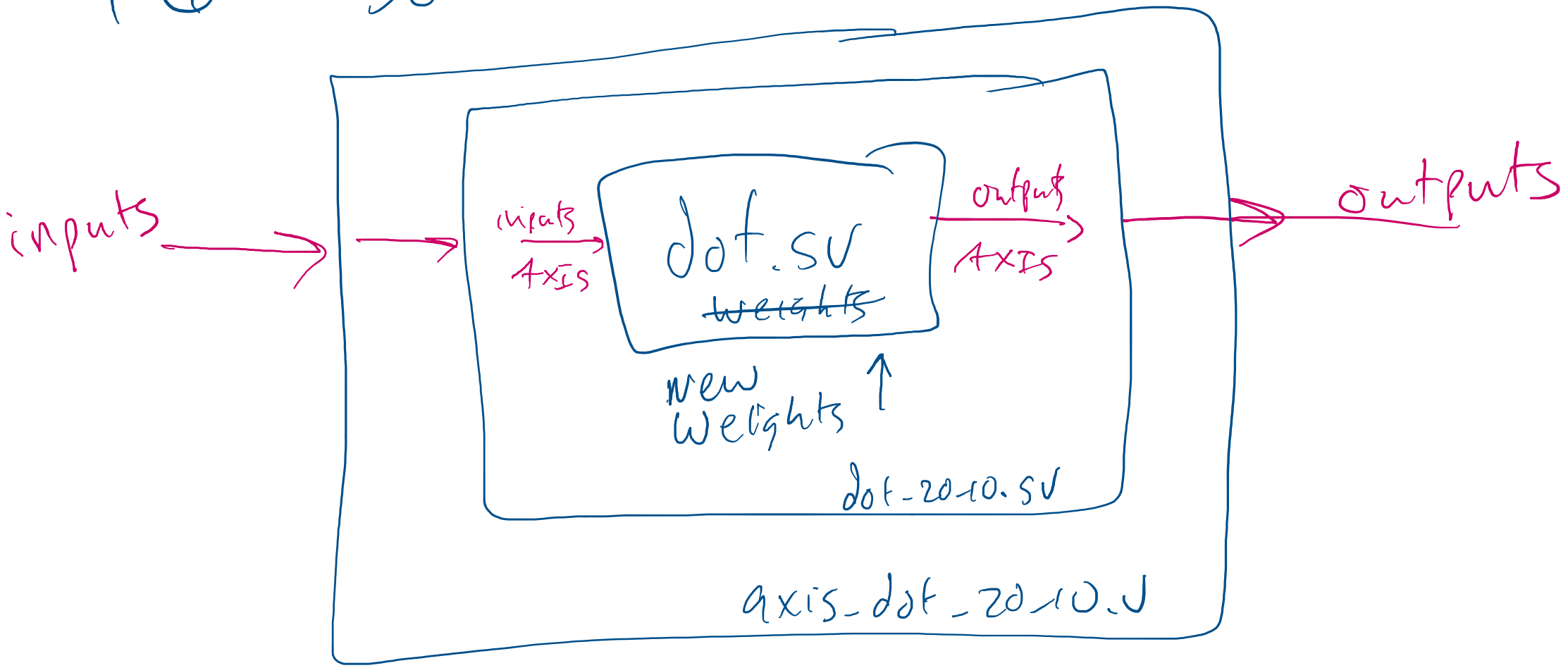
$[0.8\ 1.0\ 1.2] + [0.1\ 0.2\ 0.3] = [0.9\ 1.2\ 1.5]$

Dot Data flow

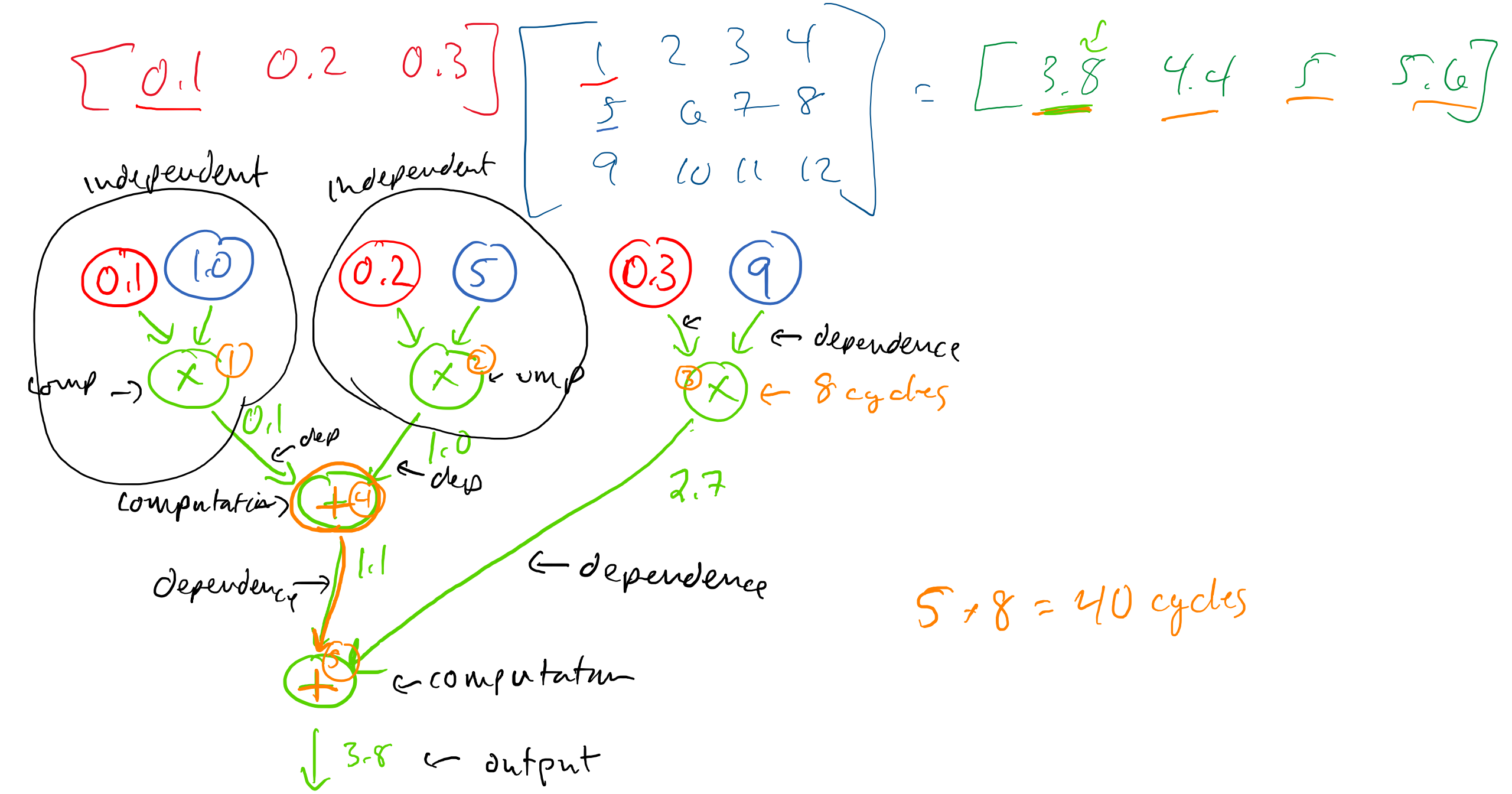


$[0.1 \ 0.2] \begin{bmatrix} 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$

P6 Dot Data flow

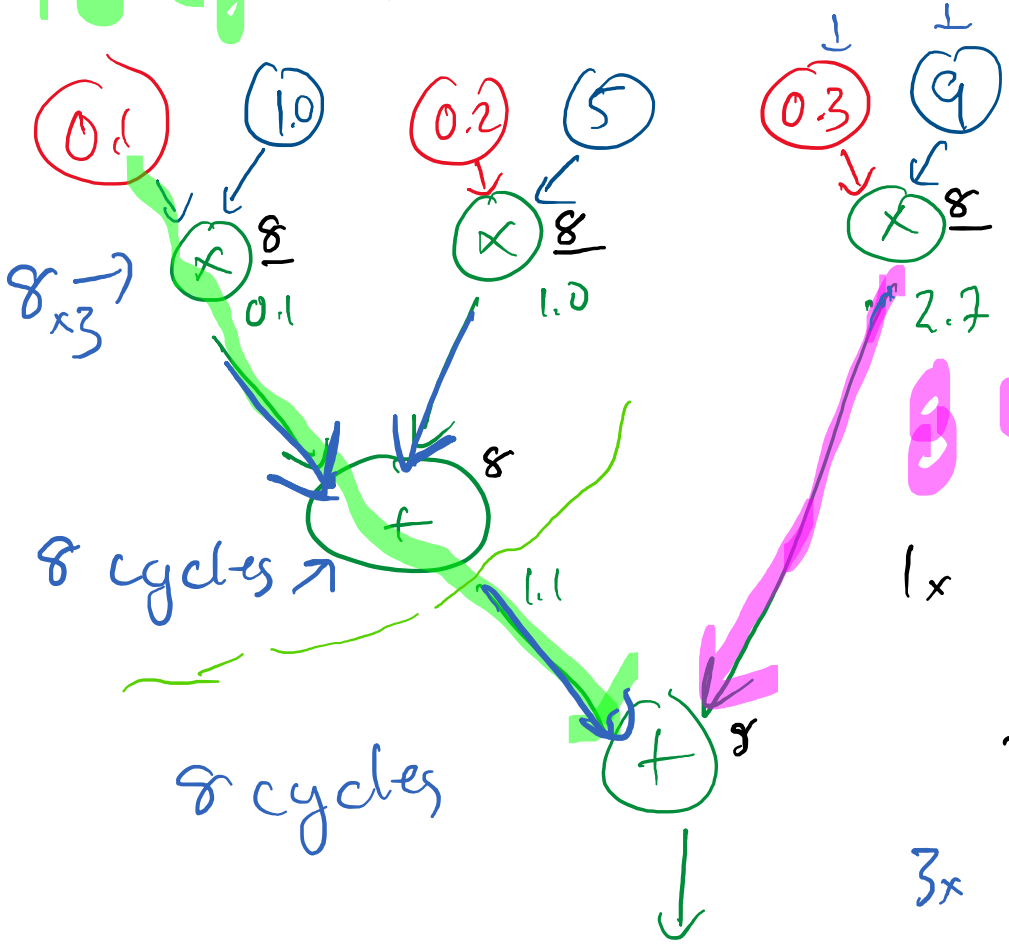


Next Time: More parallelization


[0.1 0.2 0.3]
[1 2 3 4; 5 6 7 8; 9 10 11 12]
= [3.8 4.4 5 5.6]
independent
independent
0.1
1.0
0.2
5
0.3
9
comp
comp
dependence
8 cycles
0.1
dep
1.0
dep
computation
2.7
1.1
dependency
dependence
computation
3.8
output
5 * 8 = 40 cycles

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$

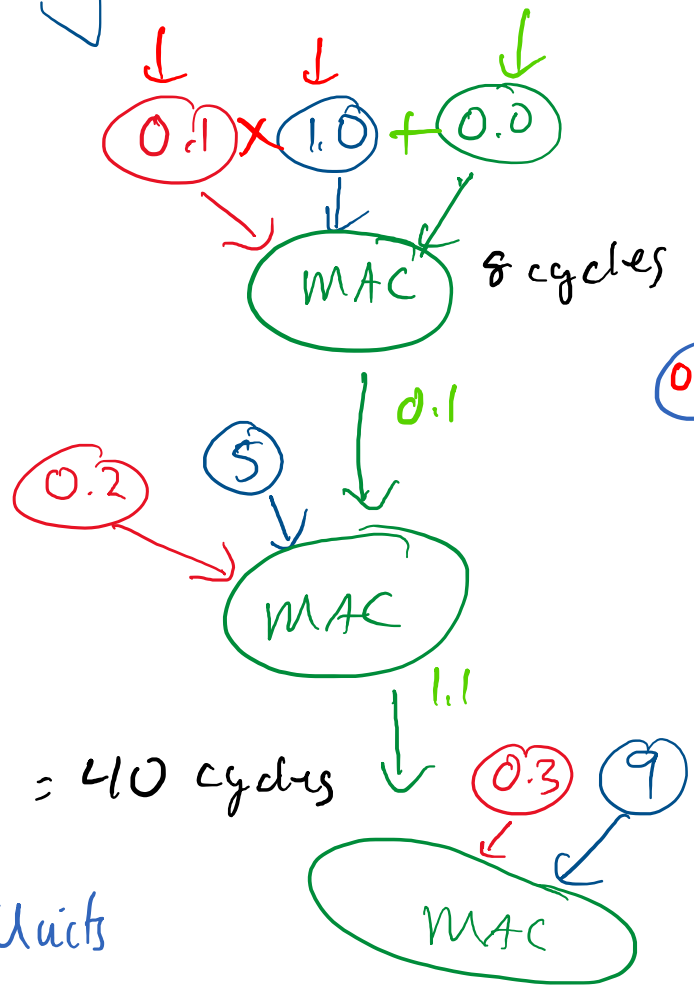
16 cycles

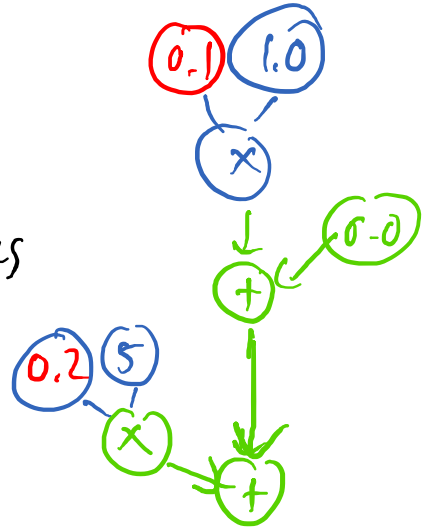


8 cycles

1x Floating-Point Unit

= 8 · 5 cycles = 40 cycles



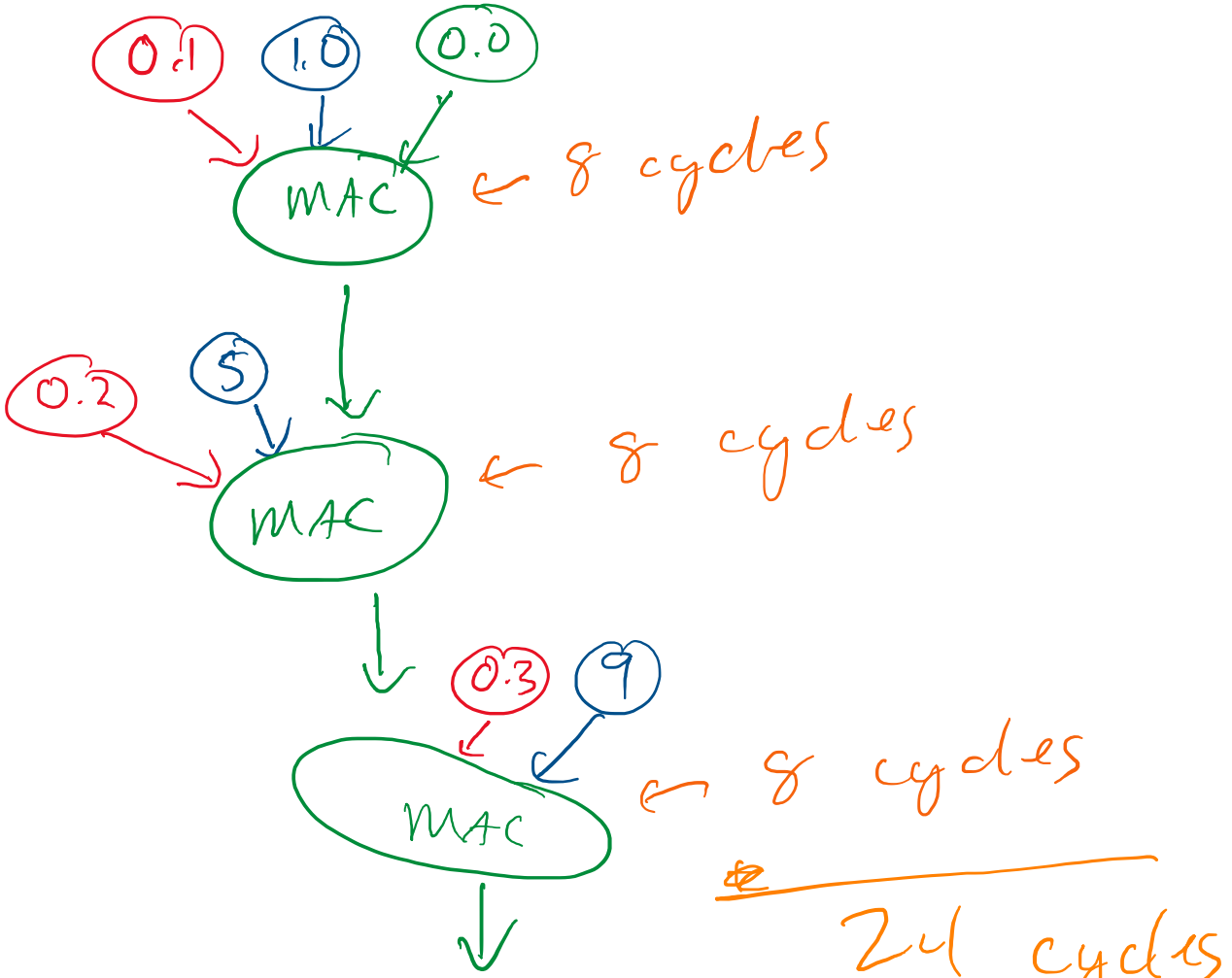


3x Floating Point Units

= 8 cycles + 8 + 8 = 24 cycles

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$

24 cycle — 24 cycles — 24 cycle — 24 cycle



24 cycles

= 24.4

= 96 cycles

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$

$0.1 \cdot 1 + 0 = 0.1$ ← +8 cycles

→ $0.2 \cdot 5 + 0.1 = 1.1$ ← +8 cycles

$0.3 \cdot 9 + 1.1 = 3.8$ ← +8 cycles

→ result: 3.8 ← 8 + 8 + 8 = 24 cycles

4.4 ← 24 cycles

5 ← 24

5.6 ← 24

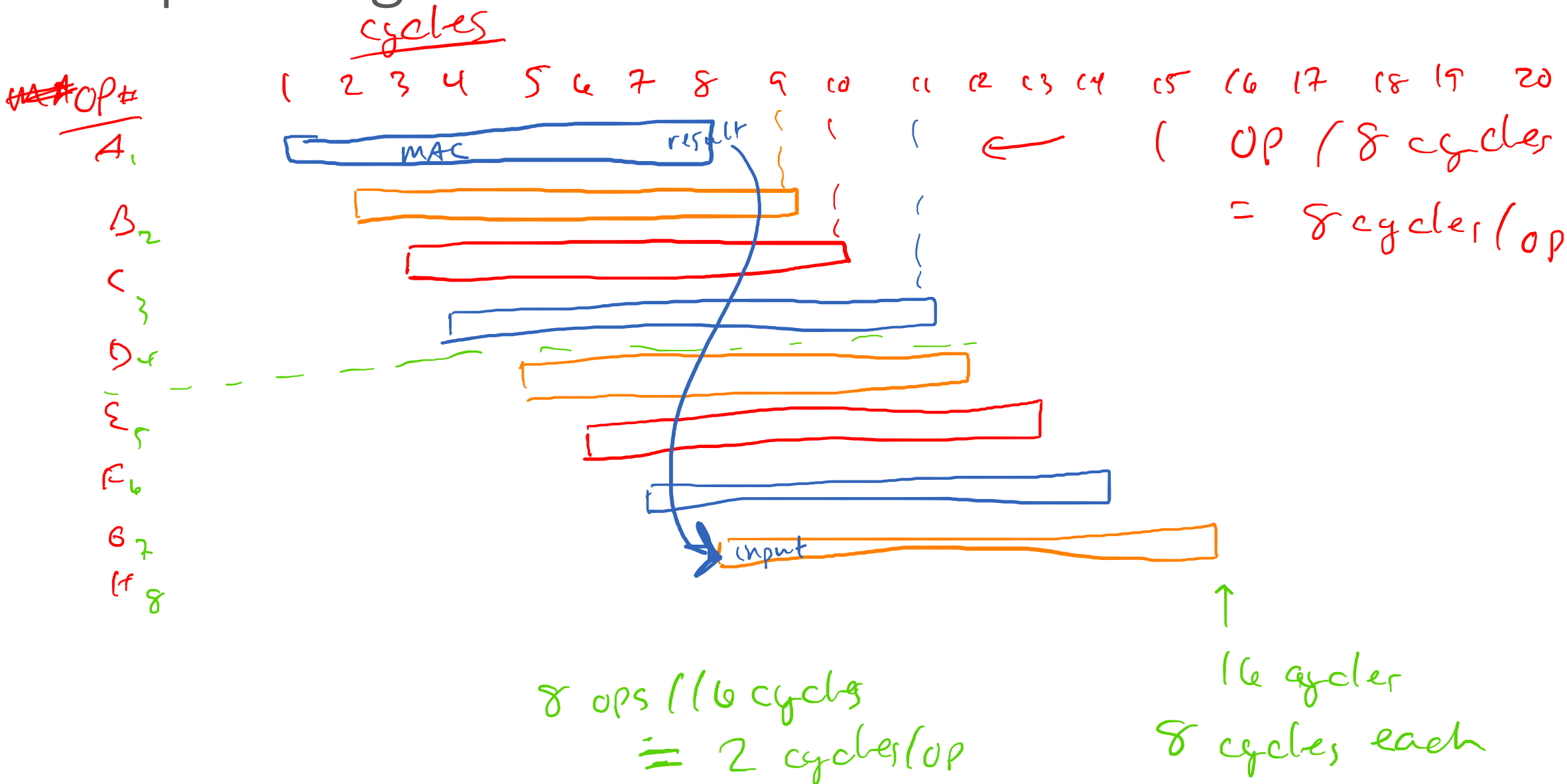
96 cycles

Pipelining

- FMAC takes 8 cycles for 1 value
- But can accept a new value every cycle.

- Latency: 8 cycles / value
- Throughput: 1 value / cycle

Pipelining


cycles
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
OP#
A1
B2
C3
D4
E5
F6
G7
H8
MAC
result
input
1 OP / 8 cycles
= 8 cycles / OP
8 ops / 16 cycles
= 2 cycles/op
16 cycles
8 cycles each

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$

$0.1 \cdot 1 + 0 = 0.1$ (8 cycles)

$0.2 \cdot 5 + 0.1 = 1.1$ 8

$0.3 \cdot 9 + 1.1 = 3.8$ (8)

$0.1 \cdot 2 + 0 = 0.2$

$0.2 \cdot 6 + 0.2 = 1.4$

$0.3 \cdot 10 + 1.4 = 4.4$

$0.1 \cdot 1 + 0 = 0.1$ ←

$0.1 \cdot 2 + 0 = 0.2$ ←

$\Rightarrow$ $0.1 \cdot 3 + 0 = 0.3$ ←

$0.1 \cdot 4 + 0 = 0.4$ ←

4 pipelined MAC in 11 cycles

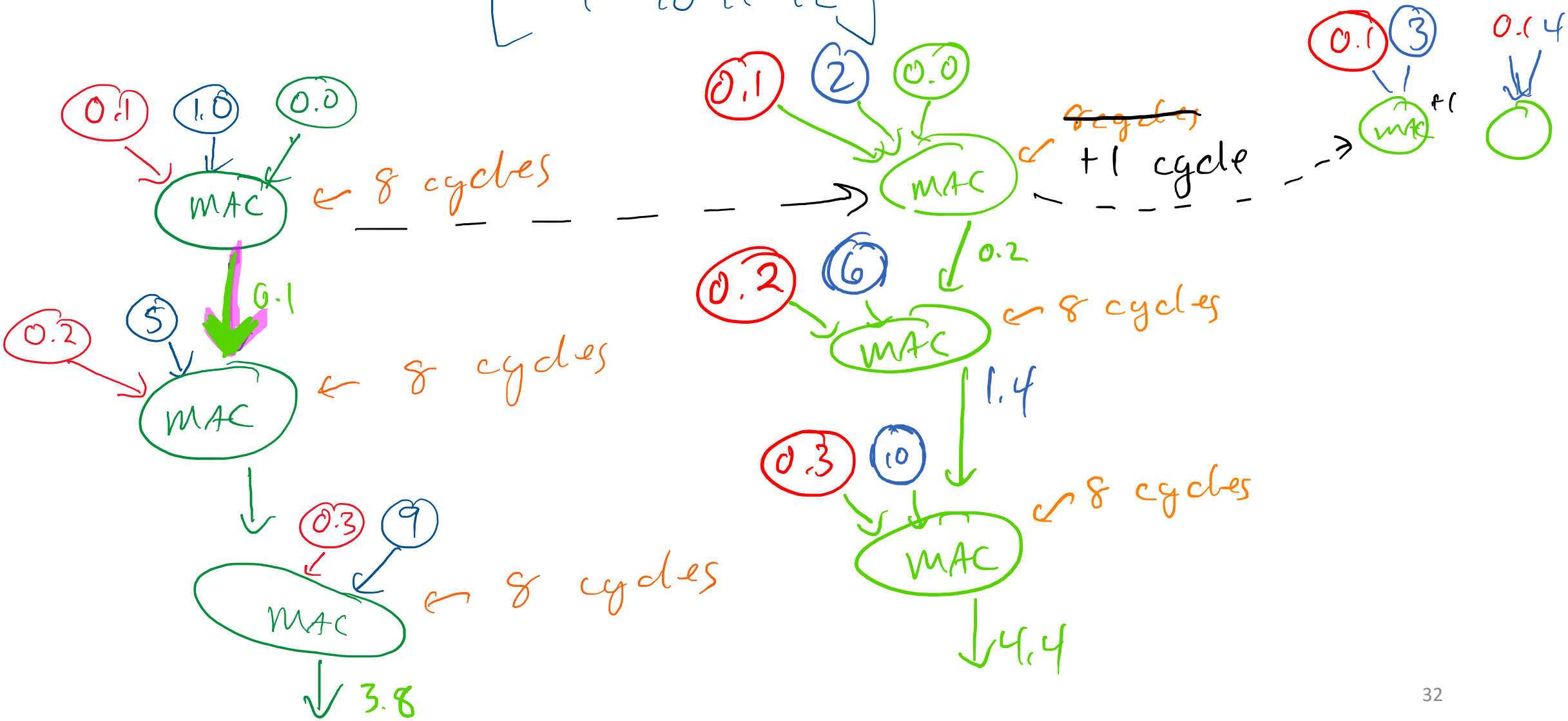
$0.2 \cdot 5 + 0.1 =$

$0.2 \cdot 6 + 0.2 =$

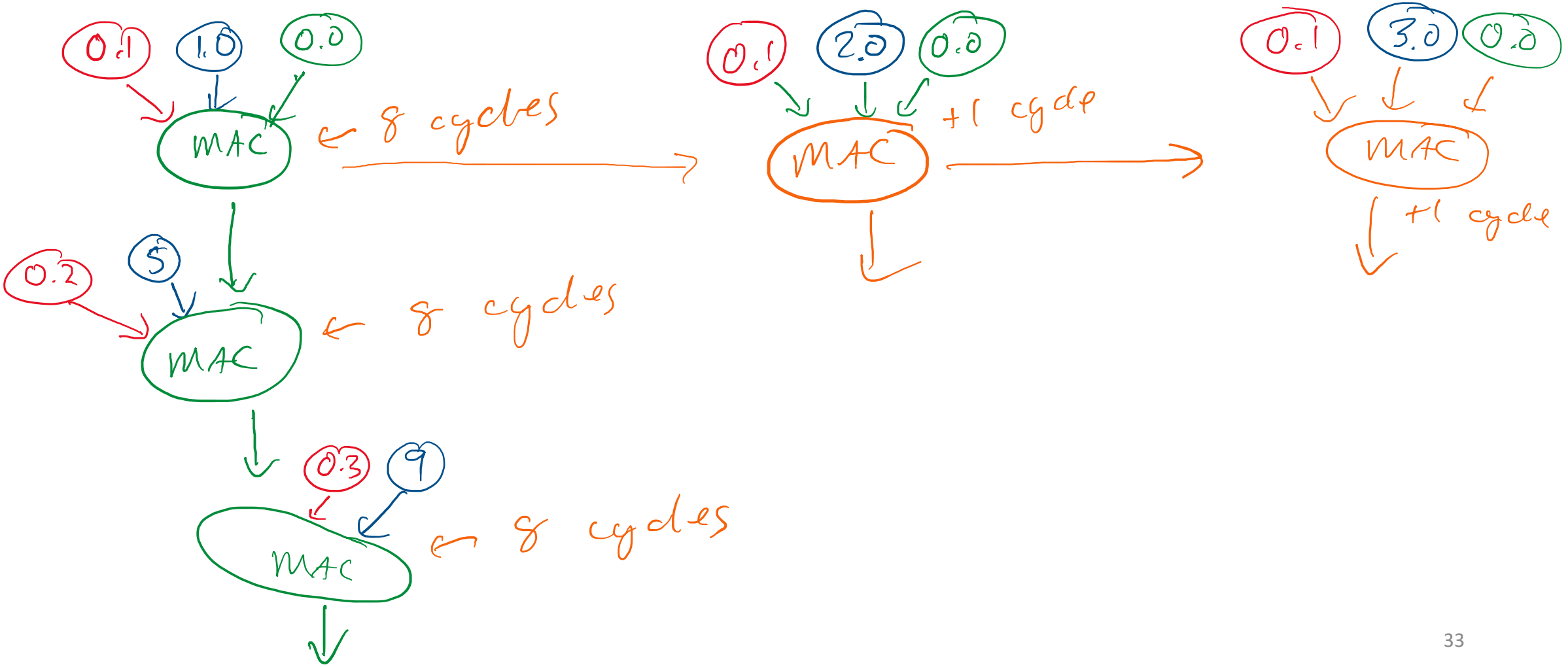
$0.2 \cdot 7 + 0.3 =$

$0.2 \cdot 8 + 0.4 =$

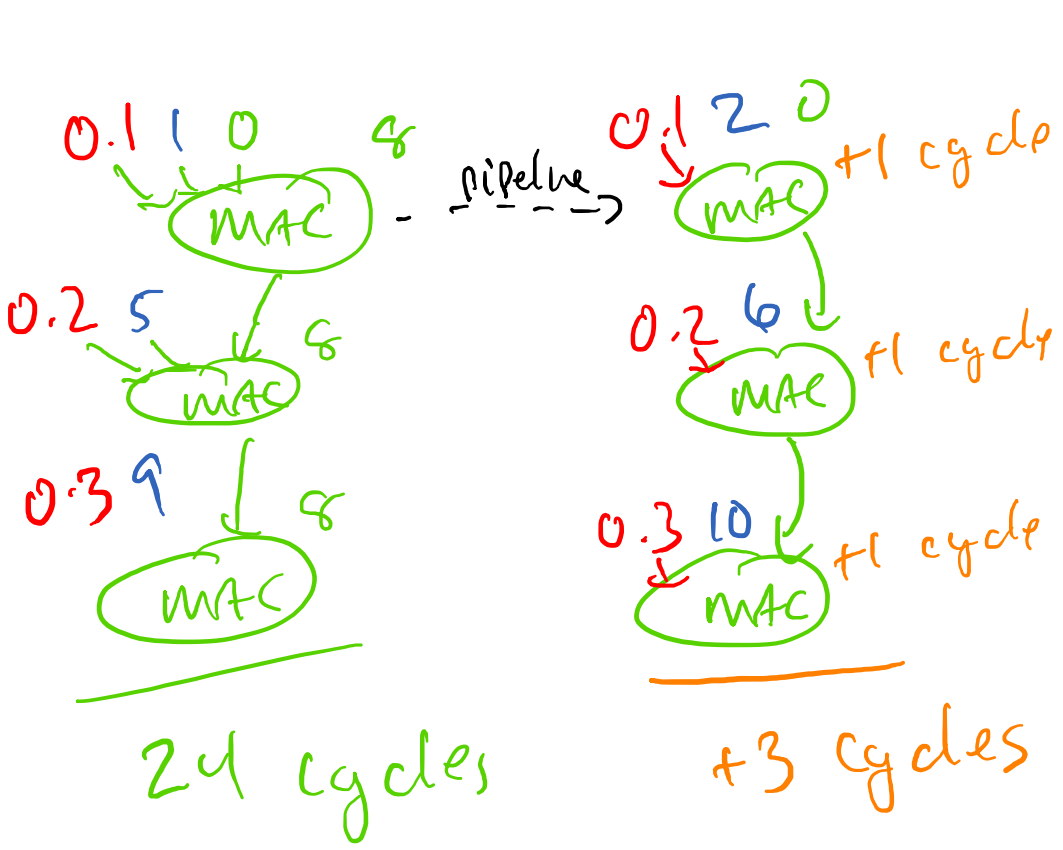
$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$

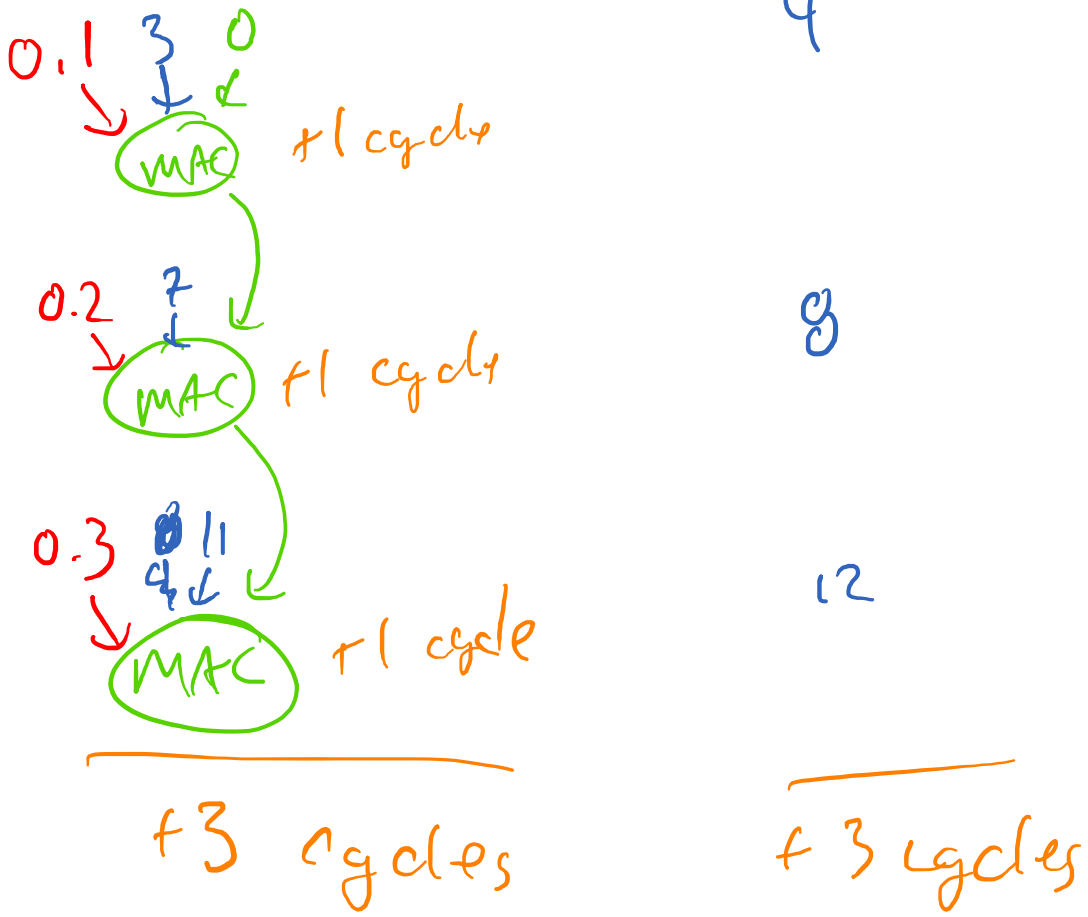


$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$



$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$





24 + 3 + 3 + 3 = 33 cycles

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$

$$\begin{array}{r} 2 \\ \times\ 3 \\ \hline 6 \end{array} = \begin{array}{r} 10 \\ \times\ 11 \\ \hline 110 \end{array}$$

$$\begin{array}{r} 10 \\ \times\ 10 \\ \hline 100 \end{array} + \begin{array}{r} 10 \\ \times\ 1 \\ \hline 10 \end{array} = 110$$

$$\begin{array}{r} 2 \\ \times\ 2 \\ \hline 4 \end{array} + \begin{array}{r} 2 \\ \times\ 1 \\ \hline 2 \end{array} = 6$$

cycle 1: $\begin{array}{r} 10 \\ \times\ 1 \\ \hline 10 \end{array}$

cycles 2: $\begin{array}{r} 10 \\ \times\ 10 \\ \hline 100 \end{array}$

cycle 3: $10 + 100 = 110$

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$

Unpipelined:
8 cycles / MAC
X 4 MACs / input
X 3 inputs
---
96 cycles

pipelined:
8 cycles for 1st MAC
and +1 cycle for 2nd+ MAC
X 4 MACs / input
---
8 + 1 + 1 + 1 = 11 cycles

11 cycles / input
X 3 inputs
---
33 cycles

Hardware Parallelism

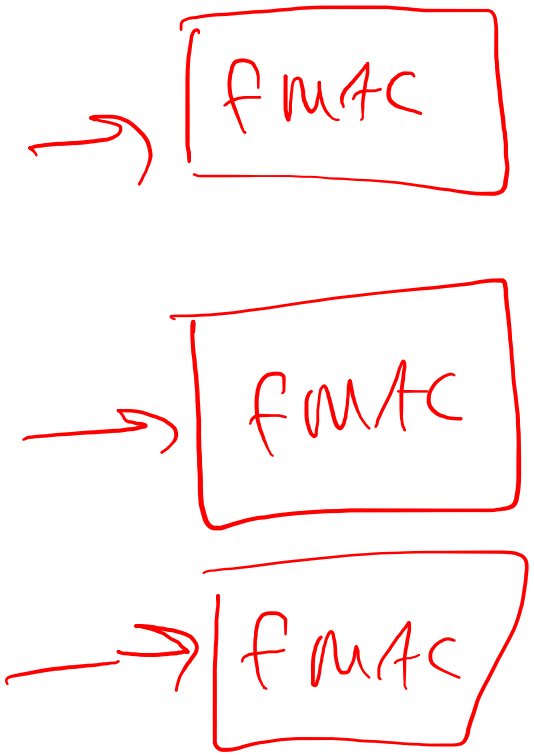
- CPU: 1 Floating-Point Unit
- FPGA? 10 Floating-Point Units?
  20 ?
  100 ?

Finding Parallelism

- Some some computation that doesn’t depend on other computation’s results
- Shared Inputs are OK.

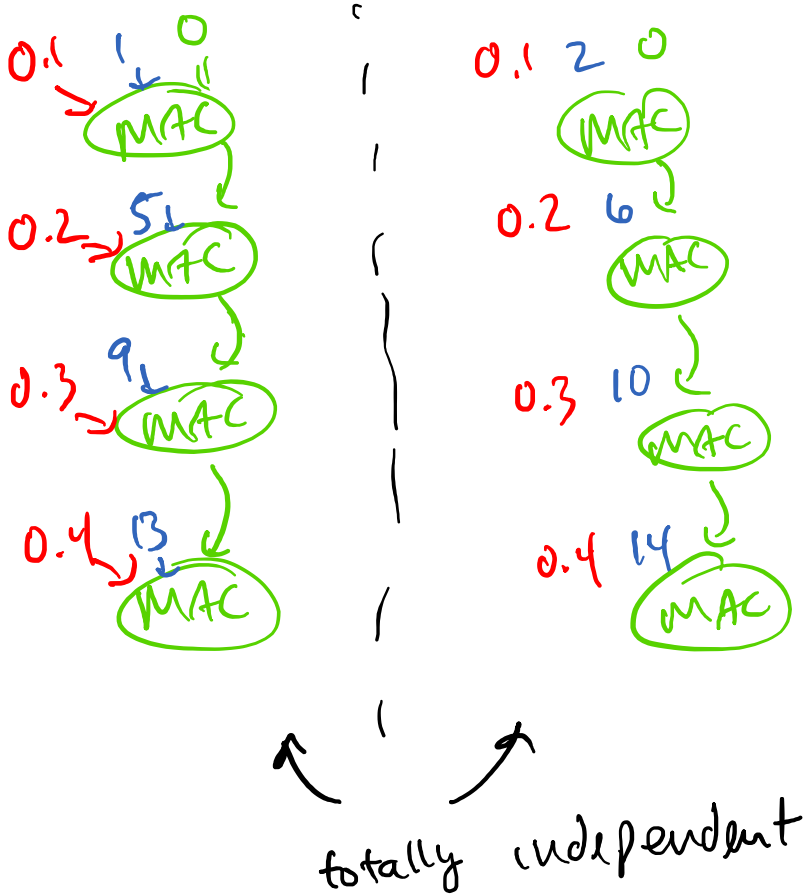
What can we do with 2+ FMACs?

dot



$$[0.1 \quad 0.2 \quad 0.3 \quad 0.4] \cdot \left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right] = [9 \quad 10 \quad 11 \quad 12]$$

independent



$$[0.1 \quad 0.2 \quad 0.3 \quad 0.4] \begin{bmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 10 \\ 13 & 14 \end{bmatrix} = [9 \quad 10]$$

$$[0.1 \quad 0.2 \quad 0.3 \quad 0.4] \begin{bmatrix} 3 & 4 \\ 7 & 8 \\ 11 & 12 \\ 15 & 16 \end{bmatrix} = [11, \quad 12]$$

$$[9 \quad 10 \quad 11 \quad 12]$$

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} = \begin{bmatrix} 9 & 10 & 11 & 12 \end{bmatrix}$$

0.1 0.2 0.3 0.4
× 1 × 5 × 9 × 13
0.1 + 1 2.7 + 5.2
1.1 + 7.9
9

0.1 0.2 0.3 0.4
× 2 × 6 × 10 × 14
0.2 + 1.2 3.0 + 5.6
1.4 + 8.6
10

0.1 0.2 0.3 0.4
× 3 × 7 × 11 × 15
0.3 + 1.4 3.3 + 6.0
1.7 + 9.3
11

0.1 0.2 0.3 0.4
× 4 × 8 × 12 × 16
0.4 + 1.6 3.6 + 6.4
2.0 + 10
12

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} = \begin{bmatrix} 9 & 10 & 11 & 12 \end{bmatrix}$$

0.1 0.2 0.3 0.4
× 1 × 5 × 9 × 13
0.1 + 1 2.7 + 5.2
1.1 + 7.9
9

0.1 0.2 0.3 0.4
× 2 × 6 × 10 × 14
0.2 + 1.2 3.0 + 5.6
1.4 + 8.6
10

independent

0.1 0.2 0.3 0.4
× 3 × 7 × 11 × 15
0.3 + 1.4 3.3 + 6.0
1.7 + 9.3
11

0.1 0.2 0.3 0.4
× 4 × 8 × 12 × 16
0.4 + 1.6 3.6 + 6.4
2.0 + 10
12

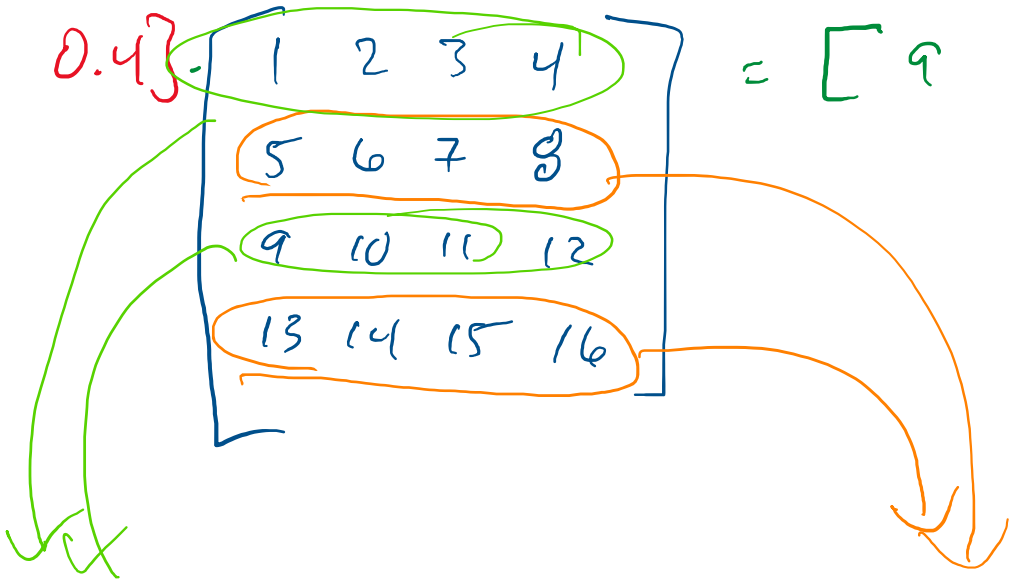
independent

$$[0.1 \quad 0.2 \quad 0.3 \quad 0.4] \cdot \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} = [9 \quad 10 \quad 11 \quad 12]$$

$$[0.1 \quad 0.3] \begin{bmatrix} 1 & 3 \\ 5 & 7 \\ 9 & 11 \\ 13 & 15 \end{bmatrix} + [0.2 \quad 0.4] \begin{bmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \\ 14 & 16 \end{bmatrix} = [9 \quad 10 \quad 11 \quad 12]$$

WRONG!

$$[0.1 \quad 0.2 \quad 0.3 \quad 0.4] \cdot \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} = [9 \quad 10 \quad 11 \quad 12]$$



$$[0.1 \quad 0.3] \begin{bmatrix} 1 & 2 & 3 & 4 \\ 9 & 10 & 11 & 12 \end{bmatrix} + [0.2 \quad 0.4] \begin{bmatrix} 5 & 6 & 7 & 8 \\ 13 & 14 & 15 & 16 \end{bmatrix} =$$

$$[2.8 \quad 3.2 \quad 3.6 \quad 4.0] + [6.2 \quad 6.8 \quad 7.4 \quad 8]$$

$$= [9 \quad 10 \quad 11 \quad 12]$$

Can we parallelize Dot?

```
# how its done in dot.sv
def pydot(inputs,weights):
    inputs = inputs[0] # remove outer nesting
    outs = np.zeros(weights.shape[1], dtype=np.float32)
    for i in range(weights.shape[0]): # input length
        for j in range(weights.shape[1]): # output length
            outs[j] = outs[j] + weights[i][j] * inputs[i]
    return outs
```

Can we parallelize Dot?

```python
# how its done in dot.sv
def pydot(inputs,weights):
    inputs = inputs[0] # remove outer nesting
    outs = np.zeros(weights.shape[1], dtype=np.float32)
    for i in range(weights.shape[0]): # input length
        for j in range(weights.shape[1]): # output length
            outs[j] = outs[j] + weights[i][j] * inputs[i]
    return outs
```

```python
def par_pydot(inputs, weights):
    par_inputs = [inputs[:,::2], inputs[:,1::2]]
    par_weights = [weights[::2,:], weights[1::2,:]]

    par_outputs = [pydot(par_inputs[0], par_weights[0]),
                   pydot(par_inputs[1], par_weights[1])]

    outputs = par_outputs[0] + par_outputs[1]
    return outputs
```

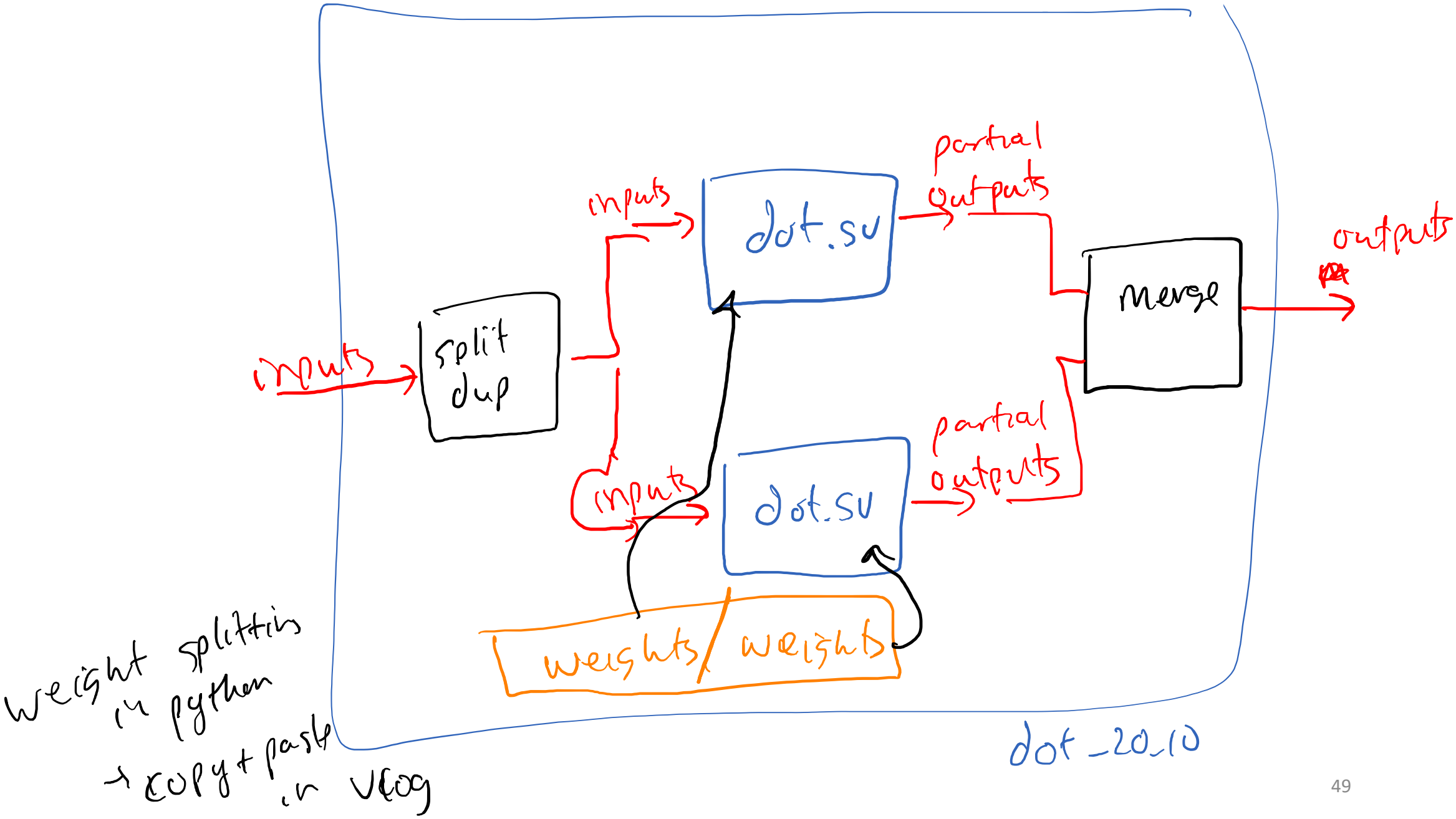
Can we parallelize Dot?

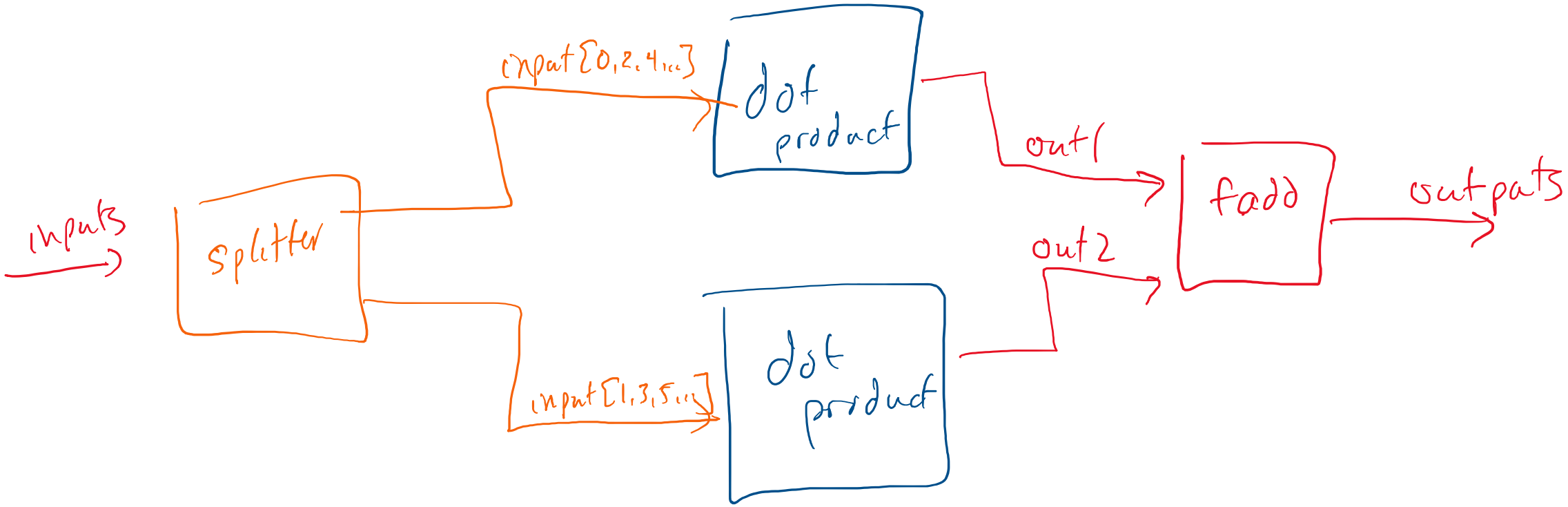
```python
# how its done in dot.sv
def pydot(inputs,weights):
    inputs = inputs[0] # remove outer nesting
    outs = np.zeros(weights.shape[1], dtype=np.float32)
    for i in range(weights.shape[0]): # input length
        for j in range(weights.shape[1]): # output length
            outs[j] = outs[j] + weights[i][j] * inputs[i]
    return outs
```

```python
def par_pydot(inputs, weights):
    par_inputs = [inputs[:,::2], inputs[:,1::2]]
    par_weights = [weights[::2,:], weights[1::2,:]]

    par_outputs = [pydot(par_inputs[0], par_weights[0]),
                   pydot(par_inputs[1], par_weights[1])]

    outputs = par_outputs[0] + par_outputs[1]
    return outputs
```


inputs
split dup
inputs
dot.sv
partial outputs
inputs
dot.sv
partial outputs
merge
outputs
weights
weights
dot_20_10
weight splitting in python
→ copy + paste in vlog


inputs
splitter
input[0,2,4,...]
input[1,3,5,...]
dot product
dot product
out1
out2
fadd
outputs

16: Hardware Acceleration III

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University