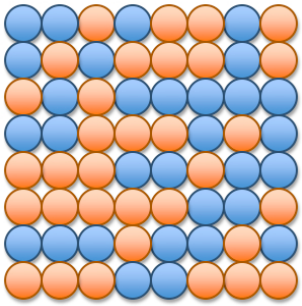


UNIT 10A
Concurrency: Pipelining & Sorting Networks

Concurrency

- Concurrency is the process of performing more than one process at a time.
- Computing has many ways to implement concurrency:
  - pipelining
  - parallel processing
  - multitasking
  - distributed computing

Pipelining

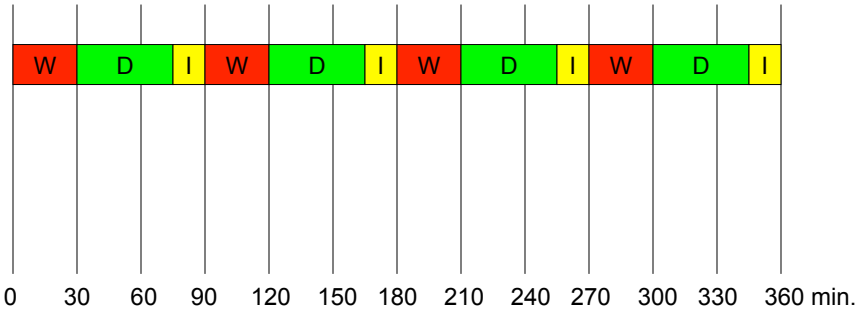
- Pipelining is similar to an assembly line.
  - Instead of completing one computation before starting another, each computation is split into simpler sub-steps, and computations are started as others are in progress.

Laundry Without Pipelining

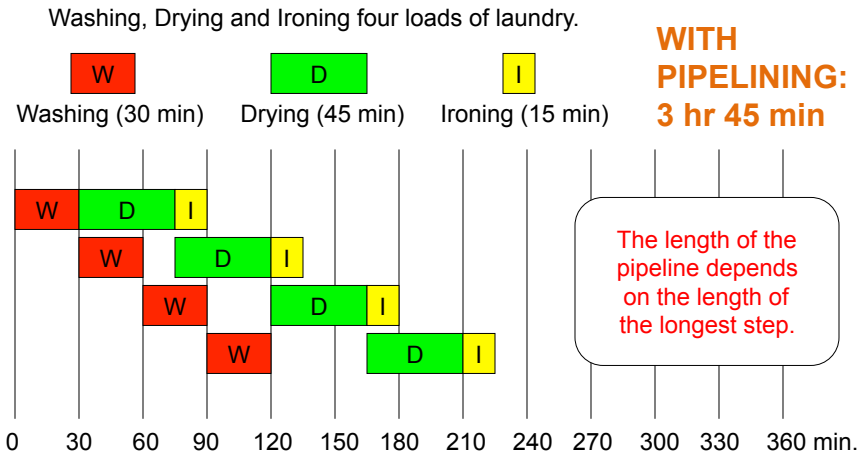
Washing, Drying and Ironing four loads of laundry.

W Washing (30 min) D Drying (45 min) I Ironing (15 min)

WITHOUT PIPELINING: 6 hr



Laundry With Pipelining



Matrix Multiplication

	hw	paper	exam1	exam2	exam3	final
student1	95	90	93	91	85	92
student2	73	80	75	63	79	75
student3	85	73	80	85	88	91
student4	50	65	50	60	56	47
student5	100	95	98	96	96	90
student6	75	75	75	75	75	75
student7	90	80	80	90	100	100
student8	88	80	80	70	60	55

	weight
hw	0.15
paper	0.1
exam1	0.15
exam2	0.15
exam3	0.15
final	0.3

	average
student1	
student2	
student3	
student4	
student5	
student6	
student7	
student8	

Matrix Multiplication

0 + 95*0.15 + 90*0.1 + 93*0.15 + 91*0.15 + 85*0.15 + 92*0.3 = 91.2

	hw	paper	exam1	exam2	exam3	final
student1	95	90	93	91	85	92
student2	73	80	75	63	79	75
student3	85	73	80	85	88	91
student4	50	65	50	60	56	47
student5	100	95	98	96	96	90
student6	75	75	75	75	75	75
student7	90	80	80	90	100	100
student8	88	80	80	70	60	55

	weight
hw	0.15
paper	0.1
exam1	0.15
exam2	0.15
exam3	0.15
final	0.3

	average
student1	91.2
student2	
student3	
student4	
student5	
student6	
student7	
student8	

Matrix Multiplication

0 + 73*0.15 + 80*0.1 + 75*0.15 + 63*0.15 + 79*0.15 + 75*0.3 = 74.0

	hw	paper	exam1	exam2	exam3	final
student1	95	90	93	91	85	92
student2	73	80	75	63	79	75
student3	85	73	80	85	88	91
student4	50	65	50	60	56	47
student5	100	95	98	96	96	90
student6	75	75	75	75	75	75
student7	90	80	80	90	100	100
student8	88	80	80	70	60	55

	weight
hw	0.15
paper	0.1
exam1	0.15
exam2	0.15
exam3	0.15
final	0.3

	average
student1	91.2
student2	74.0
student3	
student4	
student5	
student6	
student7	
student8	

Matrix Multiplication

0 + 85*0.15 + 73*0.1 + 80*0.15 + 85*0.15 + 88*0.15 + 91*0.3 = 85.3

	hw	paper	exam1	exam2	exam3	final
student1	95	90	93	91	85	92
student2	73	80	75	63	79	75
student3	85	73	80	85	88	91
student4	50	65	50	60	56	47
student5	100	95	98	96	96	90
student6	75	75	75	75	75	75
student7	90	80	80	90	100	100
student8	88	80	80	70	60	55

	weight
hw	0.15
paper	0.1
exam1	0.15
exam2	0.15
exam3	0.15
final	0.3

	average
student1	91.2
student2	74.0
student3	85.3
student4	
student5	
student6	
student7	
student8	

....and so on...

Matrix Multiplication

If each multiply/add takes 1 time unit,
this non-pipelined matrix multiplication takes 48 time units.

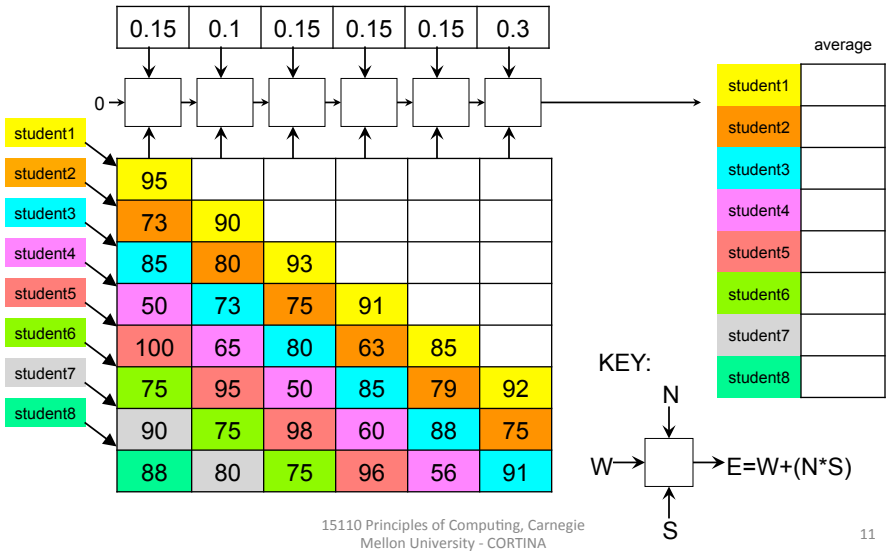
	hw	paper	exam1	exam2	exam3	final
student1	95	90	93	91	85	92
student2	73	80	75	63	79	75
student3	85	73	80	85	88	91
student4	50	65	50	60	56	47
student5	100	95	98	96	96	90
student6	75	75	75	75	75	75
student7	90	80	80	90	100	100
student8	88	80	80	70	60	55

	weight
hw	0.15
paper	0.1
exam1	0.15
exam2	0.15
exam3	0.15
final	0.3

	average
student1	91.2
student2	74.0
student3	85.3
student4	53.0
student5	95.0
student6	75.0
student7	92.0
student8	69.2

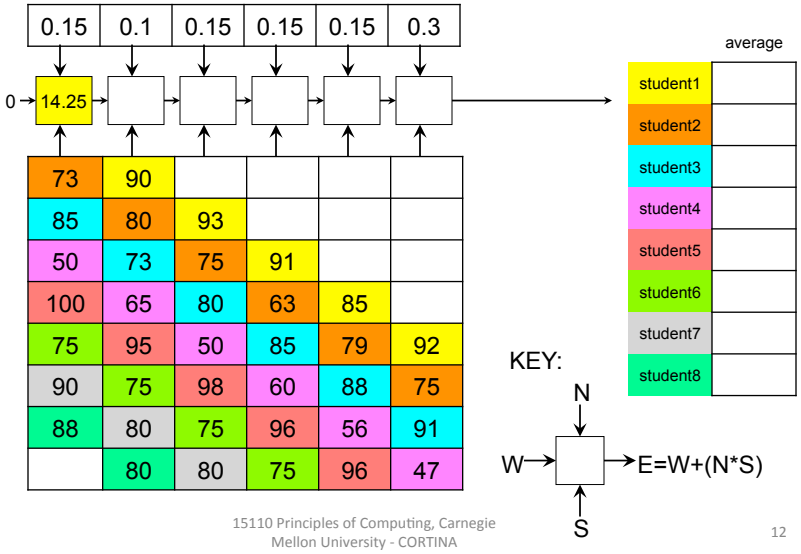
Faster Matrix Multiplication

using Pipelining



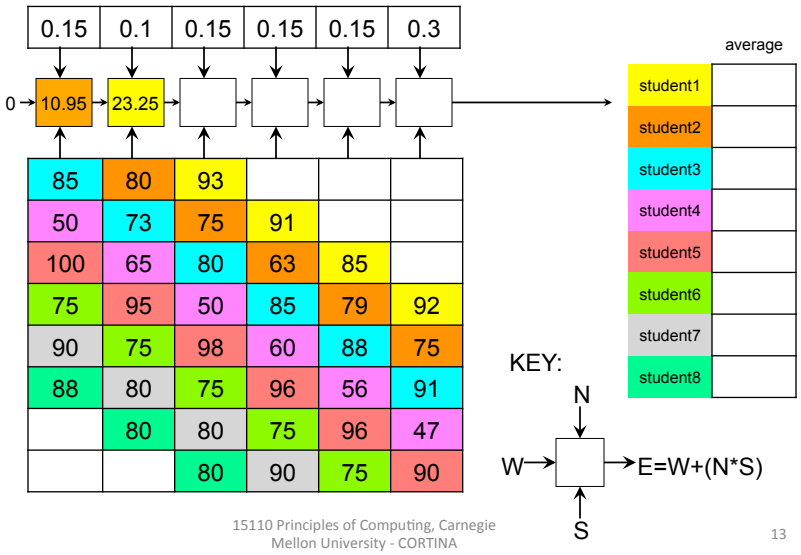
Faster Matrix Multiplication

using Pipelining



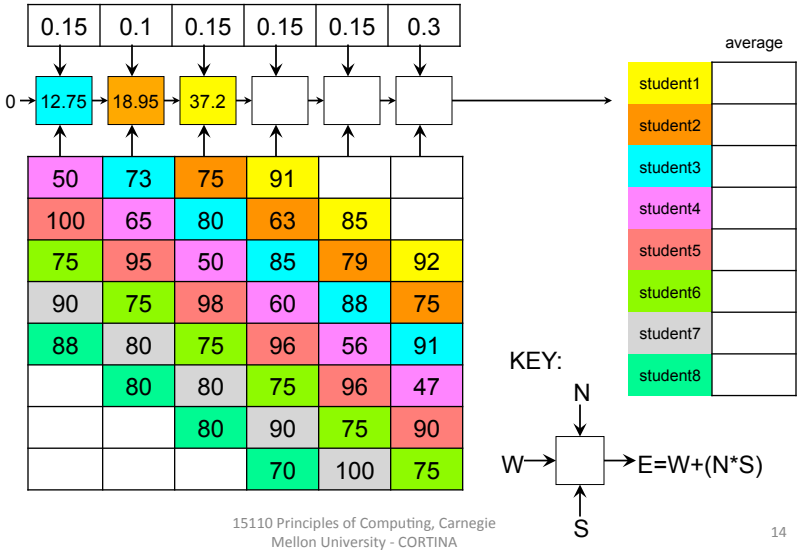
Faster Matrix Multiplication

using Pipelining



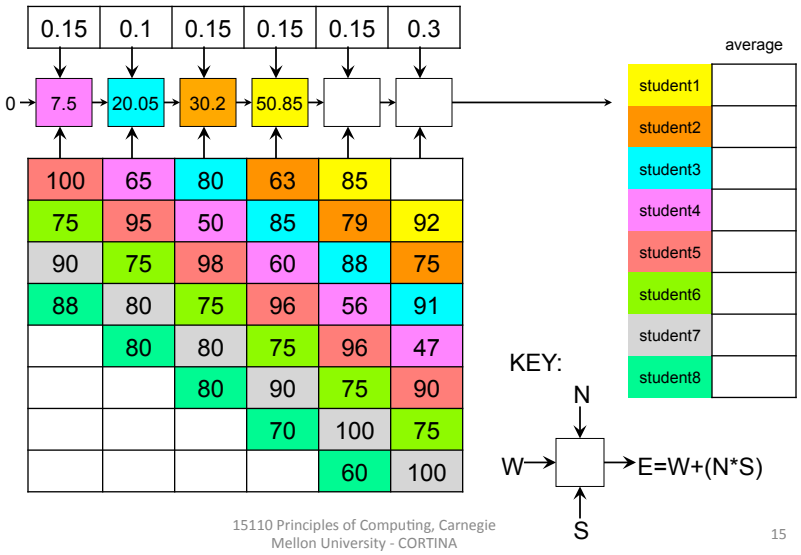
Faster Matrix Multiplication

using Pipelining



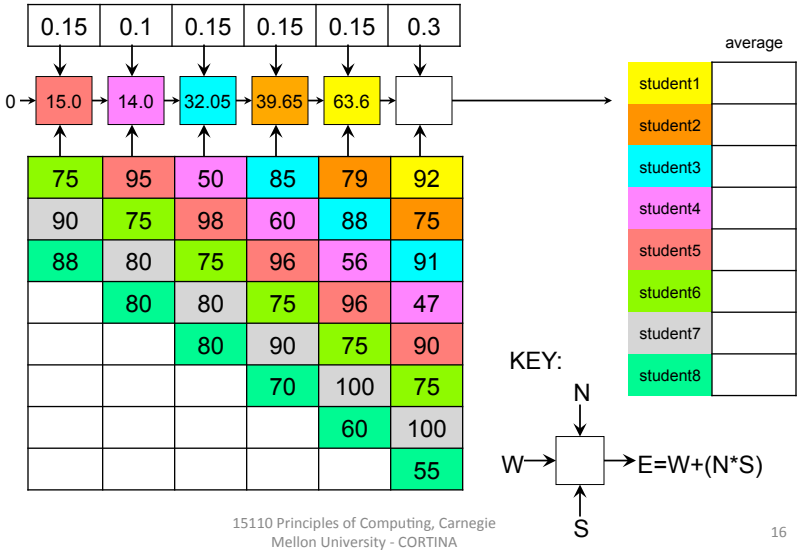
Faster Matrix Multiplication

using Pipelining



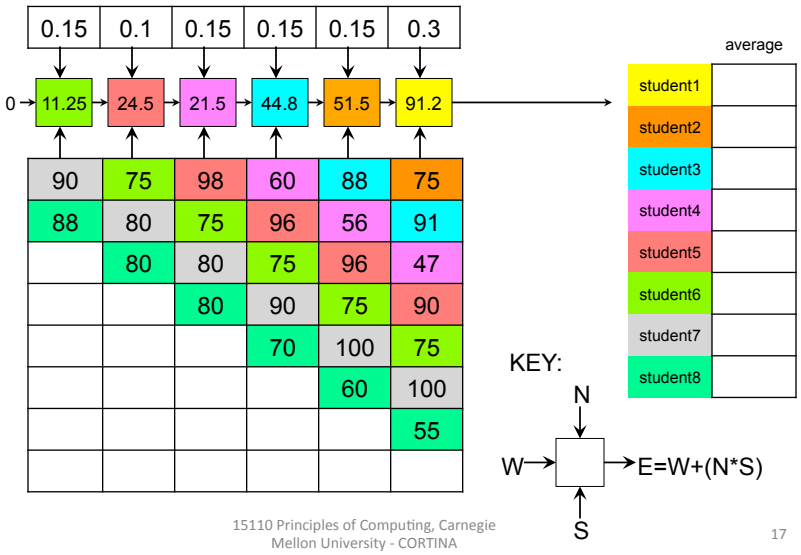
Faster Matrix Multiplication

using Pipelining



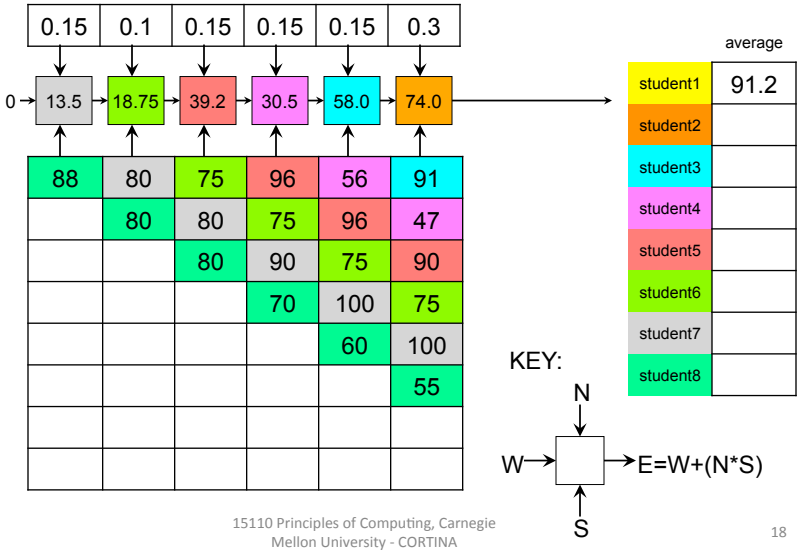
Faster Matrix Multiplication

using Pipelining



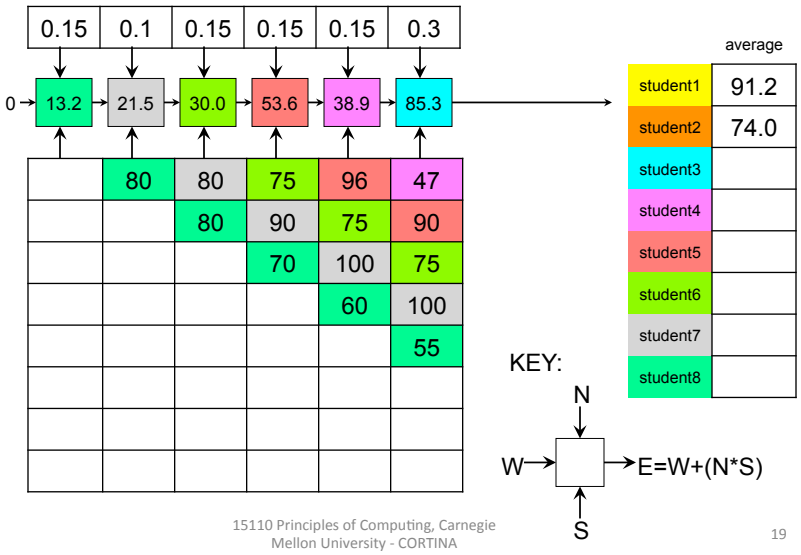
Faster Matrix Multiplication

using Pipelining



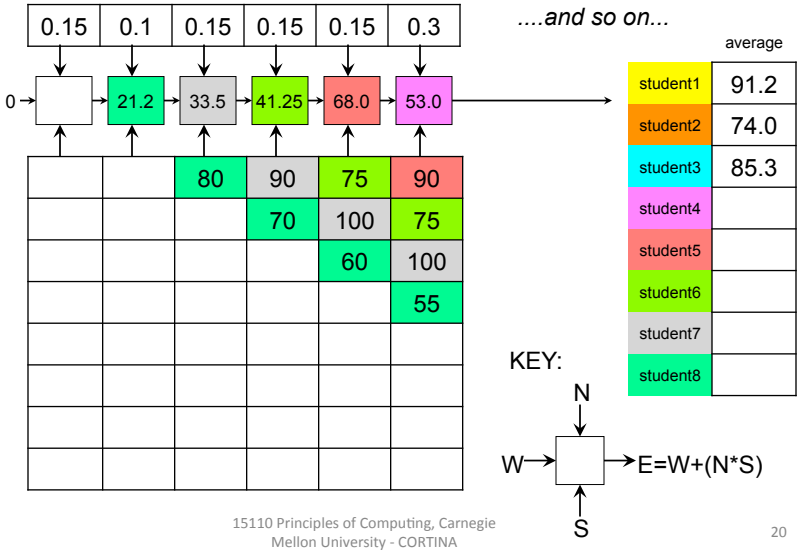
Faster Matrix Multiplication

using Pipelining



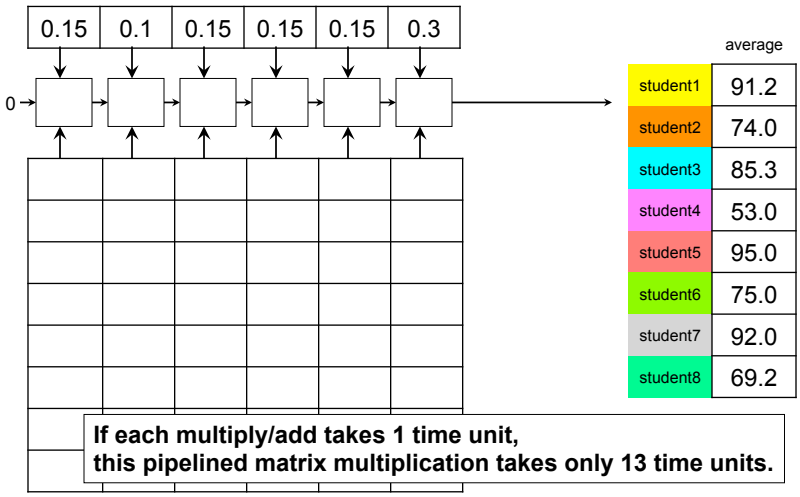
Faster Matrix Multiplication

using Pipelining



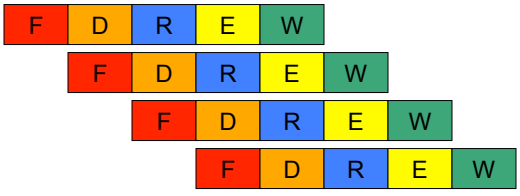
Faster Matrix Multiplication

using Pipelining



Pipelining in Computing

- Fetch instruction from memory
- Decode the instruction
- Read data from registers
- Execute the instruction
- Write the result into a register



Dealing with Dependencies

```
ADD R3, R2, R1
ADD R5, R4, R3
ADD R8, R7, R6
ADD R11, R10, R9
```

"Add the contents of R1 and R2 and store the results in R3."

This instruction depends on the result of the previous instruction. (This will hold up the pipeline.)

```
ADD R3, R2, R1
ADD R8, R7, R6
ADD R11, R10, R9
ADD R5, R4, R3
```

Reorder the instructions to minimize the delay on the pipeline due to the dependency, if possible.

Dealing with Dependencies

```
A: ADD R3, R2, R1
   SUB R6, R5, R4
   BEQ R6, R3, A
   MOV R2, R1
```

"Branch to label A if R3 = R6."

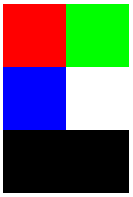
The BEQ instruction will stall in the pipeline since the final values of R3 and R6 are not known yet.

Possible solutions:

1. Assume the branch occurs. If we find later that R3 is not equal to R6, clear the pipeline and begin computing with the MOV instruction.
2. Start decoding the ADD and MOV instructions. When we know if R3 is equal to R6 or not, send the appropriate instructions into the pipeline for completion.

Bitmap Images

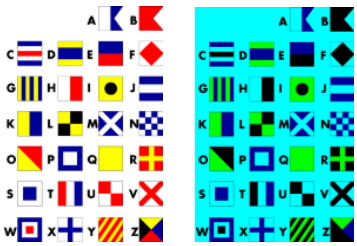
- screen consists of individual pixels
  - pixel = picture elements
- arranged into rows and columns
  - projector 1024x768
  - 720p = 1280x720
  - 1080p = 1920x1080
- Bitmap as a 3-D Python Lists



2 X 3 pixel image

```
bitmap = [[[255,0,0], [0,255,0]],
[[0,0,255], [255,255,255]],
[[0,0,0], [0,0,0]]]
```


Example: Image Processing



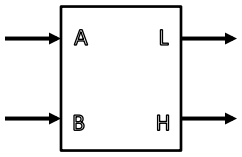
```
def remove_red(image):
    num_rows = len(image)
    num_columns = len(image[0])
    for row in range(0,num_rows):
        for column in range(0,num_columns):
            image[row][column][0] = 0
    return None
```

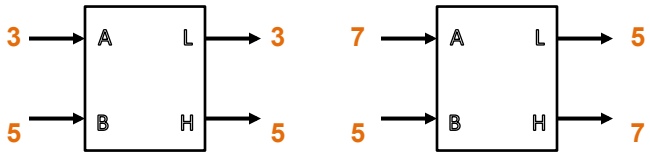
Example: Image Processing

- What order are the pixels processed?
  - row by row, one pixel at a time
- Does this matter?
  - not really: all pixel computations are independent of one another
  - if we have multiple processors (cores), we could have each work on part of the image independently → faster results
  - Graphical Processing Units (GPU)

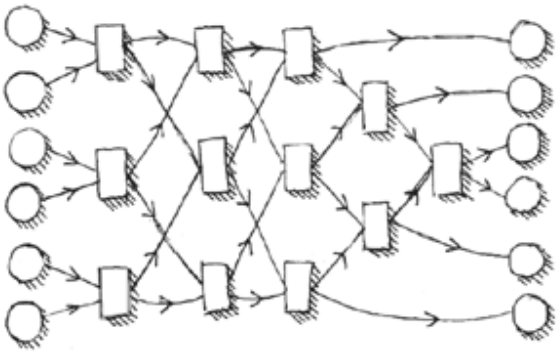
Example: Sorting Networks

- Comparator
  - Input: A and B
  - Output:
    If A <= B, then L = A and H = B
    If A > B, then L = B and H = A





Activity: Sorting Network Simulation



Input: [5, 1, 6, 3, 4, 2]

How many steps does this take . . . sequentially? concurrently?

Sorting Network: Wire Diagram