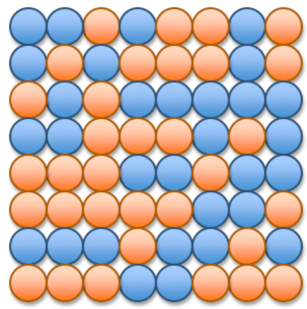


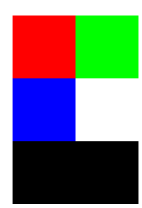
UNIT 10B

Concurrency: Moore's Law Revisited & Sorting Networks

Review: 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 Ruby Arrays

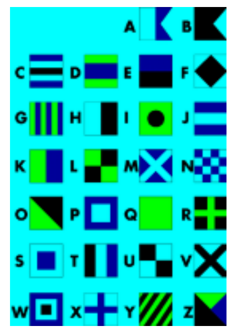
```
bitmap = [[[255,0,0], [0,255,0]],
[[0,0,255], [255,255,255]],
[[0,0,0], [0,0,0]]]
```



2 X 3 pixel image

Example: Image Processing

Remove Red operation



Example: Image Processing

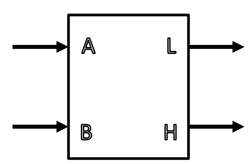
```
def remove_red(image)
   num_rows = image.length
   num_columns = image[0].length
   for row in 0..num_rows do
     for column in 0..num_columns do
          image[row][column][0] = 0
     end
   end
   return nil
end
```

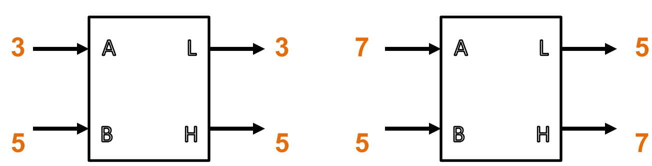
Example: Image Processing

- 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)

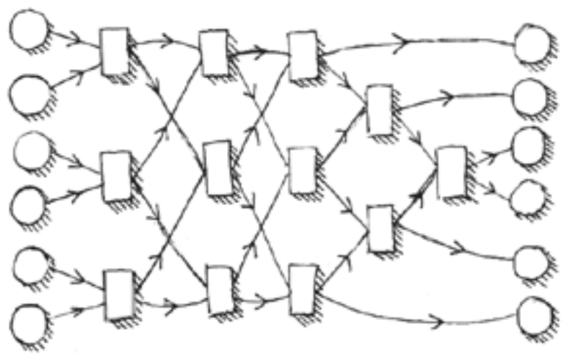
Example: Sorting Networks

- 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?

Review: Merge Operation for Merge Sort

- Merge Operation
- Takes two sorted lists (a and b)
- Returns one sorted list containing elements of a and b
- Can we do this concurrently? How?

```
>> merge([27,49,84,91],
         [17,32,53,63])
=> [17,27,32,49,53,63,84,91]
```

```
def merge(a,b)
    i, j = 0, 0
    c = []
    while i < a.length
            and j < b.length do
        if a[i] <= b[j] then
            c << a[i]
            i = i + 1
        else
            c << b[j]
            j = j + 1
        end
    end
    return c + a[i..-1]
                  + b[j..-1]
end
```

An Observation

- Merge with Odd and Even Elements Marked

  >> merge([27, 49, 84, 91, 92, 93],[17, 32, 53, 63, 95, 98])

  => [17, 27, 32, 49, 53, 63, 84, 91, 92, 93, 95, 98]

  - elements initially at even indices
  - elements initially at odd indices
- Do you see a pattern?
  - How many even/odd elements are in result[0..i]?
  - In result[0..i]:
    - always, at least as many even as odd
    - always, at most two more even than odd
    - when i is even, there is exactly one more even than odd

A Strategy for Merging

Procedure for Merging a and b

- Parameters: two sorted lists a and b   Result: one sorted list c
  - Split a into even_a and odd_a
  - Split b into even_b and odd_b
  - Recursively, merge even_a and even_b into even_c
  - Recursively, merge odd_a and odd_b into odd_c
  - interleave even_c and odd_c to get an almost-sorted c
  - swap neighbors, as necessary, to completely sort c

a = 27 49 84 91 b = 17 32 53 63

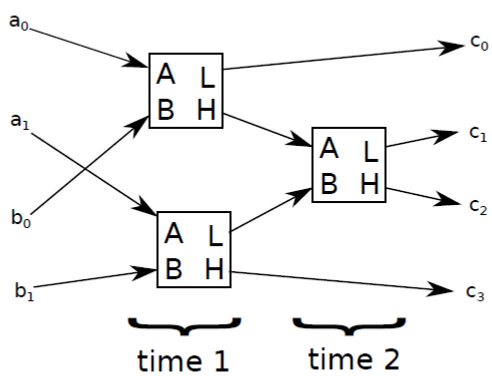
even_a = 27 84 odd_a = 49 91 even_b = 17 53 odd_b = 32 63

even_c = 17 27 53 84 odd_c = 32 49 63 91

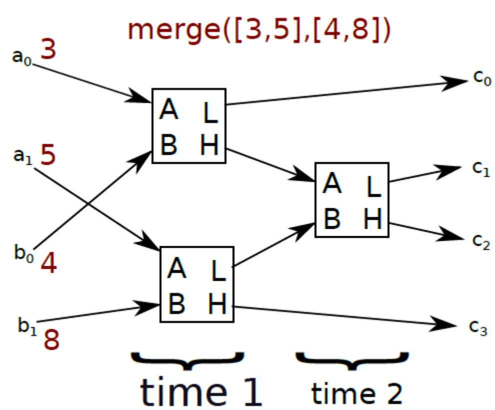
c = 17 32 27 49 53 63 84 91

c = 17 27 32 49 53 63 84 91

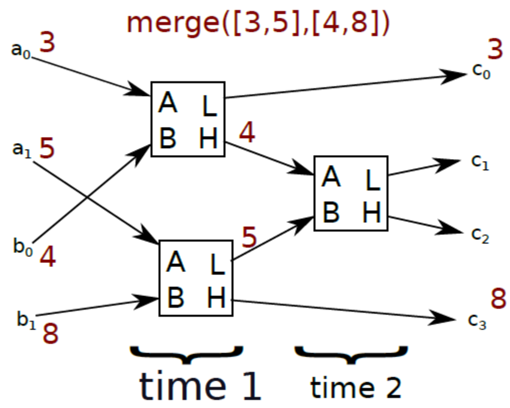
2 X 2 Merge Network



2 X 2 Merge Network



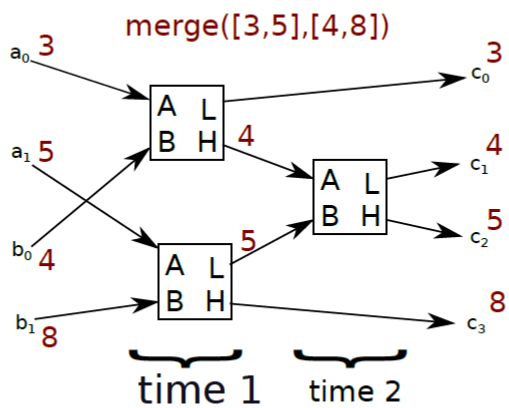
2 X 2 Merge Network



15110 Principles of Computing, Carnegie Mellon University - CORTINA

13

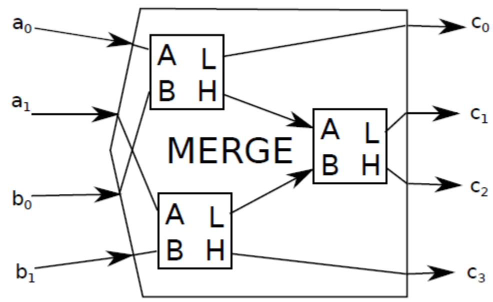
2 X 2 Merge Network



15110 Principles of Computing, Carnegie Mellon University - CORTINA

14

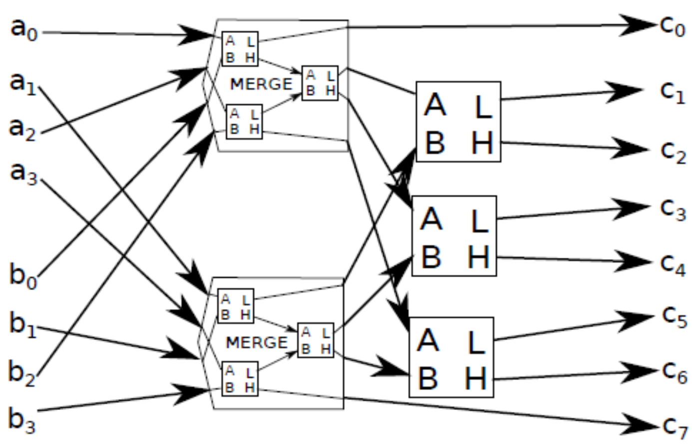
2 X 2 Merge Network Abstraction



15110 Principles of Computing, Carnegie Mellon University - CORTINA

15

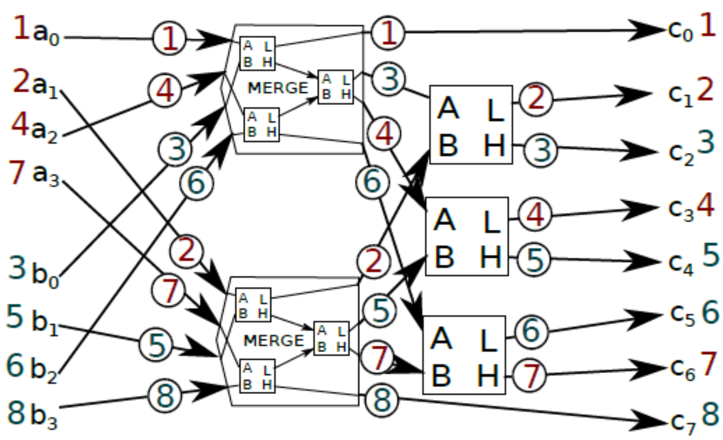
4 X 4 Odd-Even Merge



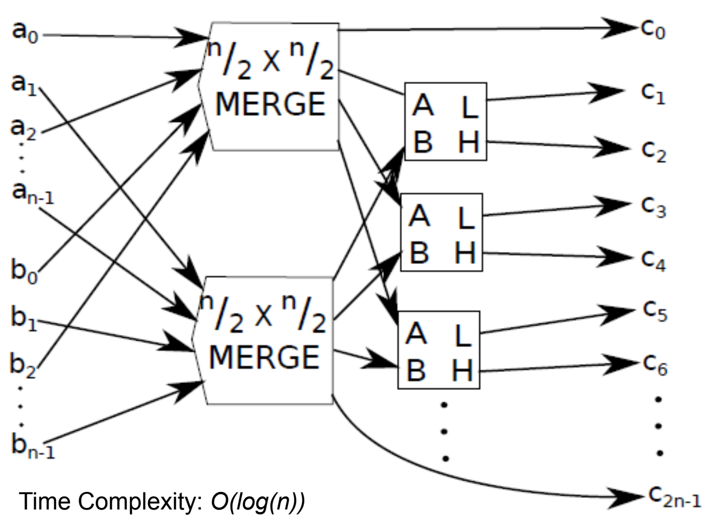
15110 Principles of Computing, Carnegie Mellon University - CORTINA

16

4 X 4 Odd-Even Merge



n X n Odd-Even Merge



Time Complexity: *O(log(n))*

Summary

- Multi-cores allow us to think about computation as a concurrent process.
- Some applications are naturally "parallel" and can be split up into smaller subproblems that are solved independently by individual processors/cores.
  - image processing
  - sorting