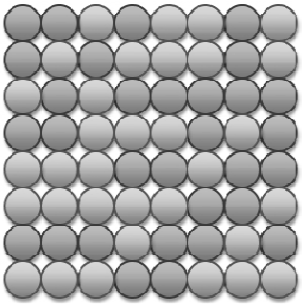


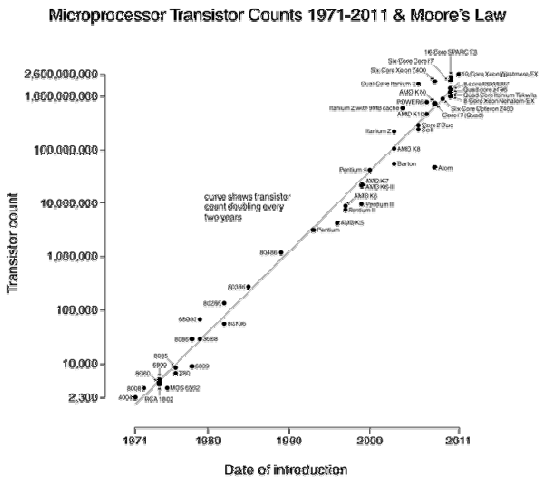
UNIT 10A

Concurrency: Moore's Law Revisited & Sorting Networks

Concurrency

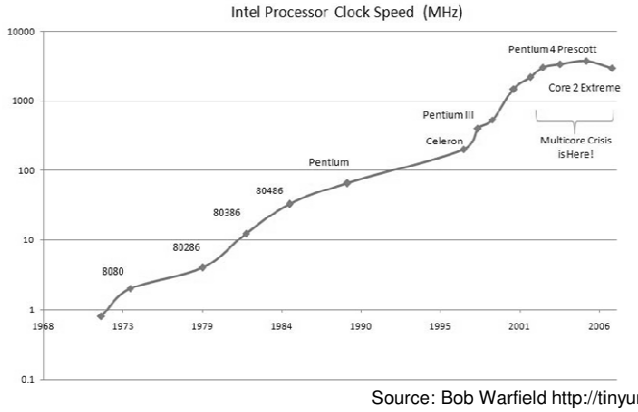
- Concurrency is the process of performing more than one process at a time.
- Computing has many ways to implement concurrency:
  - parallel processing
  - pipelining
  - multitasking
  - distributed computing

Recall Moore's Law



Source: Wikimedia Commons http://tinyurl.com/3d7qf3m

Clock Speed Is No Longer Increasing

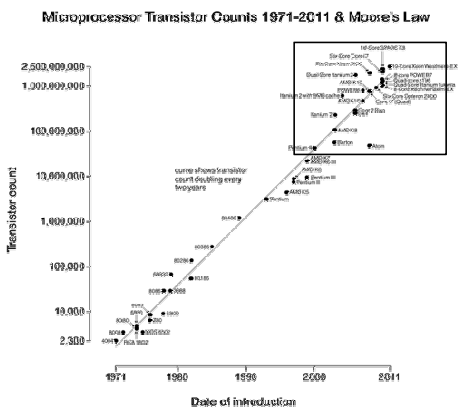


Source: Bob Warfield http://tinyurl.com/3pt6we9

- 1 MHz is 1,000,000 cycles/second
- Each cycle is like "step" operation in MARS

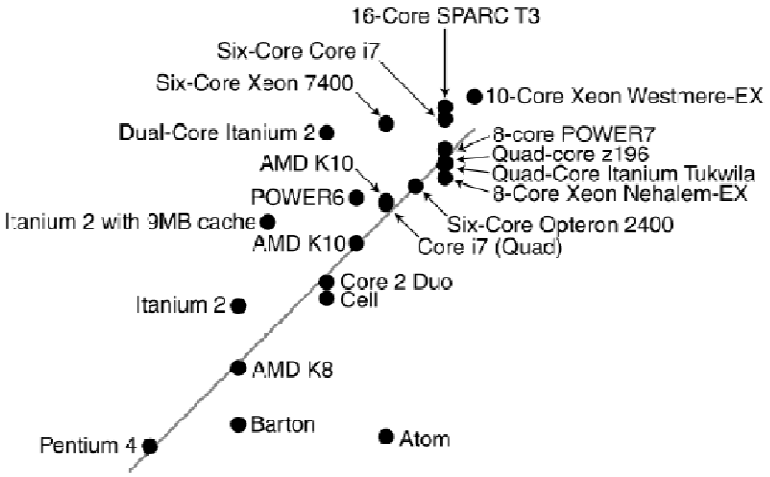
Moore's Law

- MARS single *pc* (program counter) indicates next instruction
- multi-core executing simultaneously at multiple *pc's*
- prediction: thousands of cores



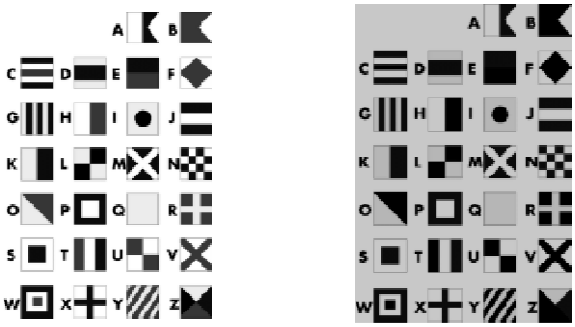
Source: Wikimedia Commons http://tinyurl.com/3d7qf3m

New Processors are Multi-Cores



Example: Image Processing

Remove Red operation



Example: Image Processing

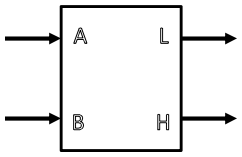
```
def remove_red(image)
num_rows = image.length
num_columns = image[0].length
    for row in 0..num_rows do
         for column in 0..num_columns do
              image[row][column][0] = 0
         end
    end
    return nil
end
```

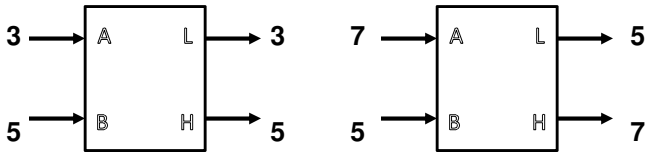
Example: Image Processing

- 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

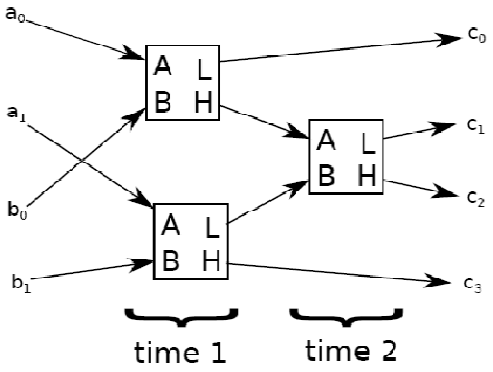




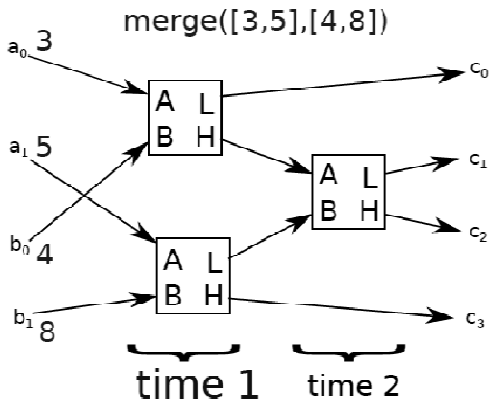
Merge Sort

```
def msort(list)
    return list if list.length == 1
    halfway = list.length/2
    list1 = list[0..halfway-1]
    list2 = list[halfway..list.length-1]
    newlist1 = msort(list1)
    newlist2 = msort(list2)
    newlist = merge(newlist1, newlist2)
    return newlist
end
>> merge([1, 2, 4, 7],[3, 5, 6])
=> [1, 2, 3, 4, 5, 6, 7]
```

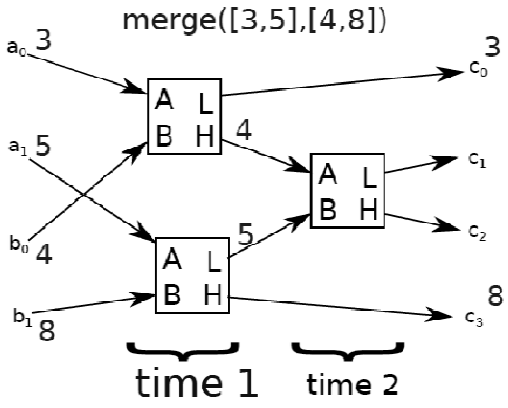

2 X 2 Merge Network



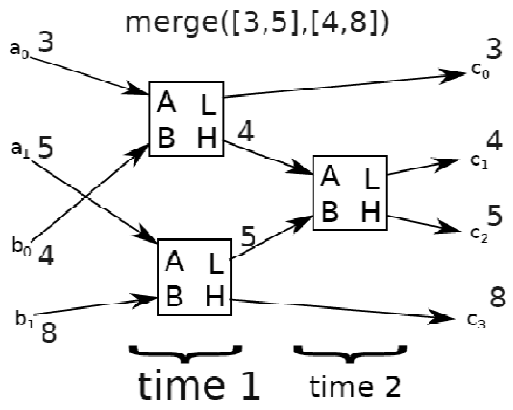
2 X 2 Merge Network



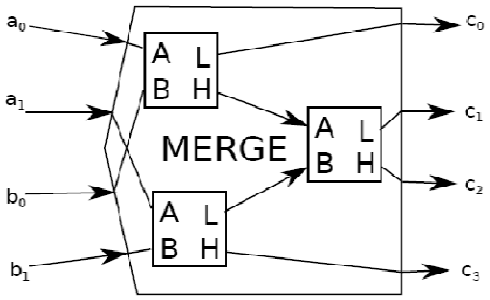
2 X 2 Merge Network



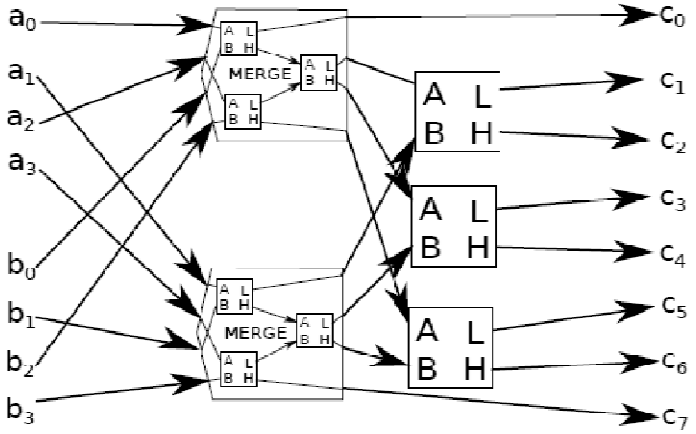
2 X 2 Merge Network



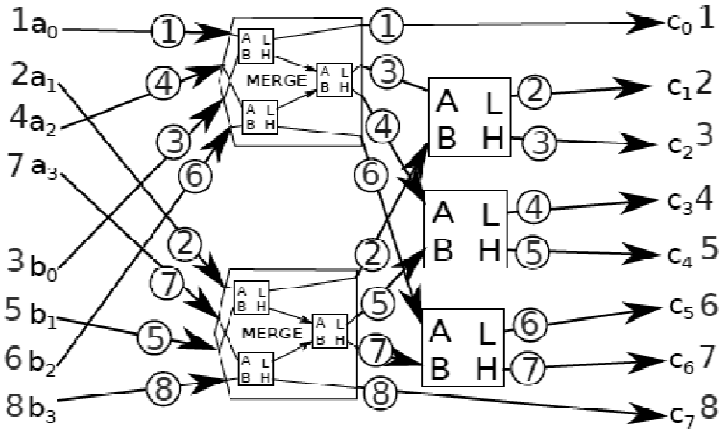
2 X 2 Merge Network Abstraction



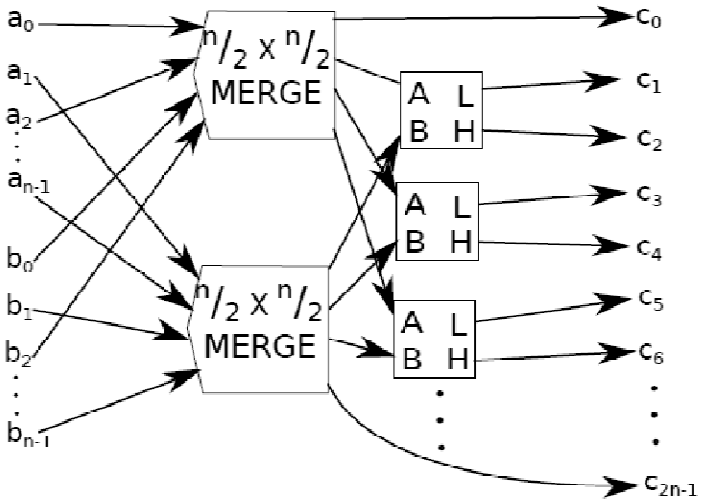
4 X 4 Odd-Even Merge



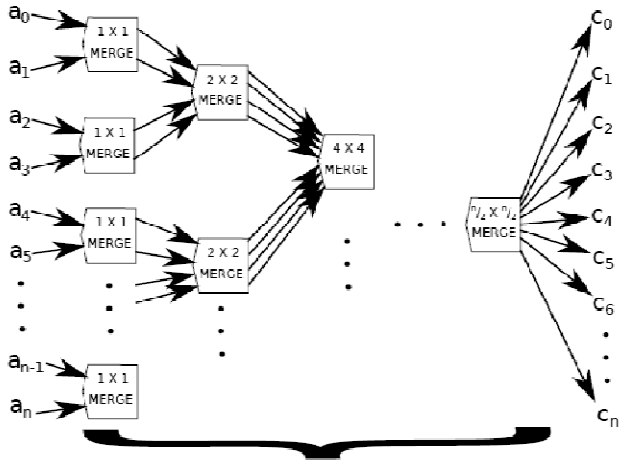
4 X 4 Odd-Even Merge



n X n Odd-Even Merge



Concurrent Merge Sort using a Sorting Network



time: O(log n) merges = O((log n)^2) comparisons

Serial vs. Concurrent Merge Sort

Time Complexity of Merging *n Elements:*

1. sequential *O(n)*
2. sorting network *O(log n)*

Time Complexity of Sorting *n Elements:*

1. sequential: *O(n log n)*
2. sorting network: *O((log n)^2)*

Space / Silicon complexity of Sort:

1. sequential: *O(n) words of memory*
2. sorting network: *O(n(log n)^2) comparison elements*

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