ThoughtWorks® Deutschland

Parallel Programming with Java 7

Wolf Schlegel and Fabian Köhler

July 2012

bcs SPA 2012

Some theory
New kids on the block
All hands on deck
Wrap up

Some theory	14:00 – 14:20
New kids on the block	14:20 – 14:30
All hands on deck	14:30 – 16:30
Wrap up	16:30 – 16:45

Part 1
Some theory

Why parallel programming?

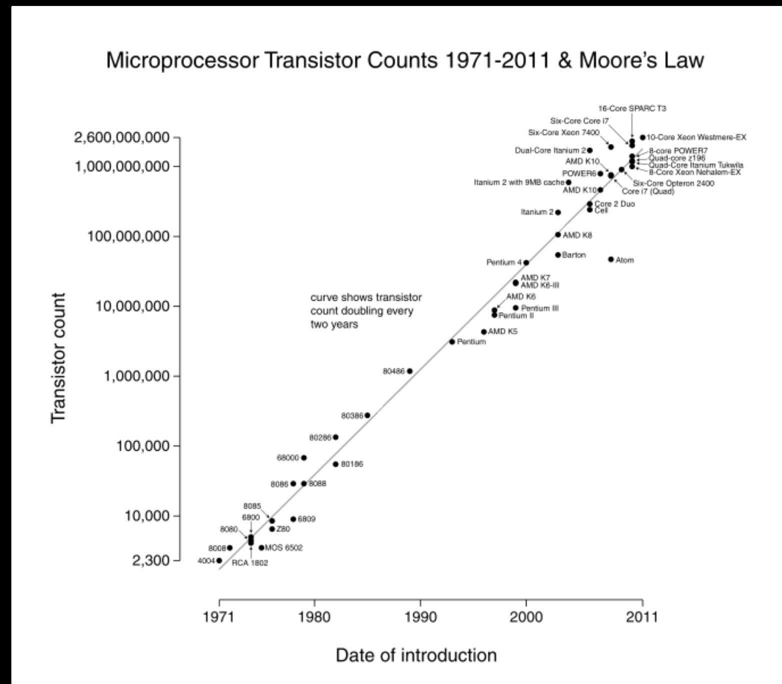
Grand challenge problems

- Modelling DNA structures
- Weather forecasting

Parallel programming splits problems into parts

- Parts are solved in parallel by multiple processors

No more free lunch



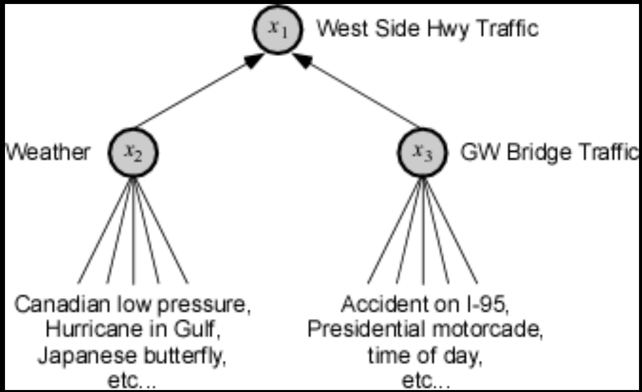
* Source: http://de.wikipedia.org/wiki/Mooresches_Gesetz

Partitioning

Data partitioning is applied to data (aka domain decomposition)

Functional decomposition concurrently executes independent functions

- Less common than data partitioning



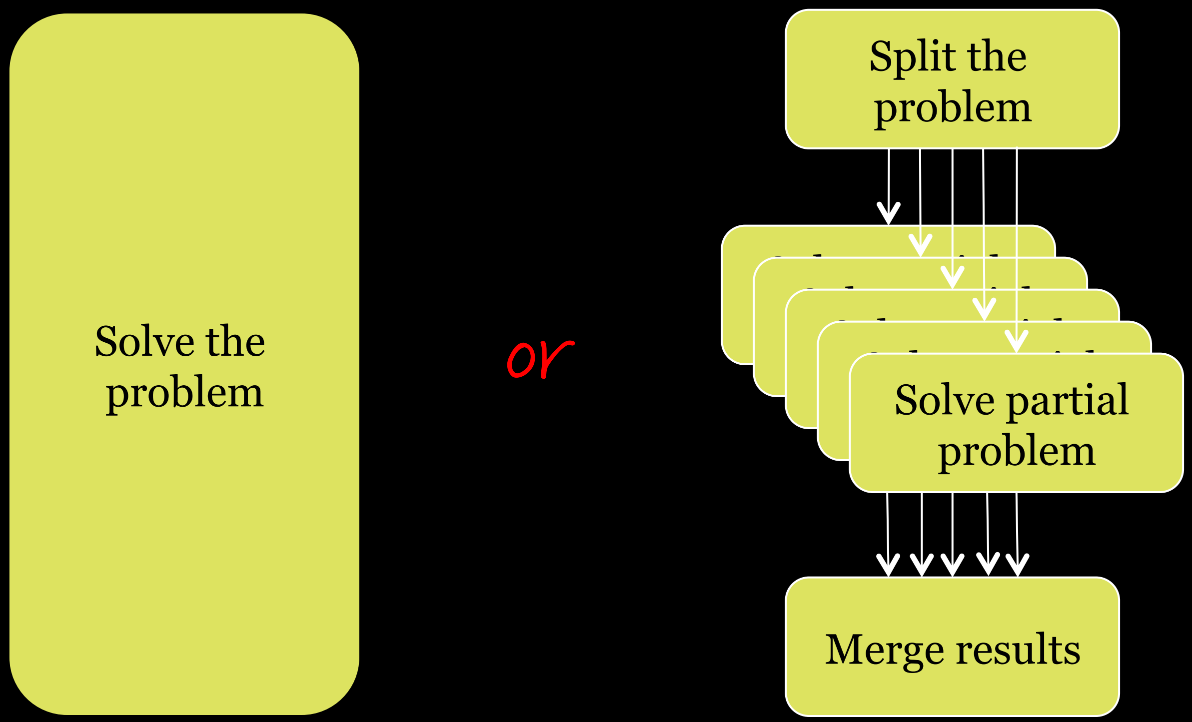
 Source: http://en.wikipedia.org/wiki/Functional_decomposition

Divide-and-conquer

Divide-and-conquer divides a problem into similar sub-problems Sub-problems that have the same form as the original.

Problem can be repeated recursively

Which way is faster?



ThoughtWorks®

Amdahl's law (1/3)

S = speedup factor S = function(n,f) where

n = number of processors/cores

f = fraction of the computation that cannot be parallelised

Amdahl's law (2/3)

S = speedup factor = function(n,f) where

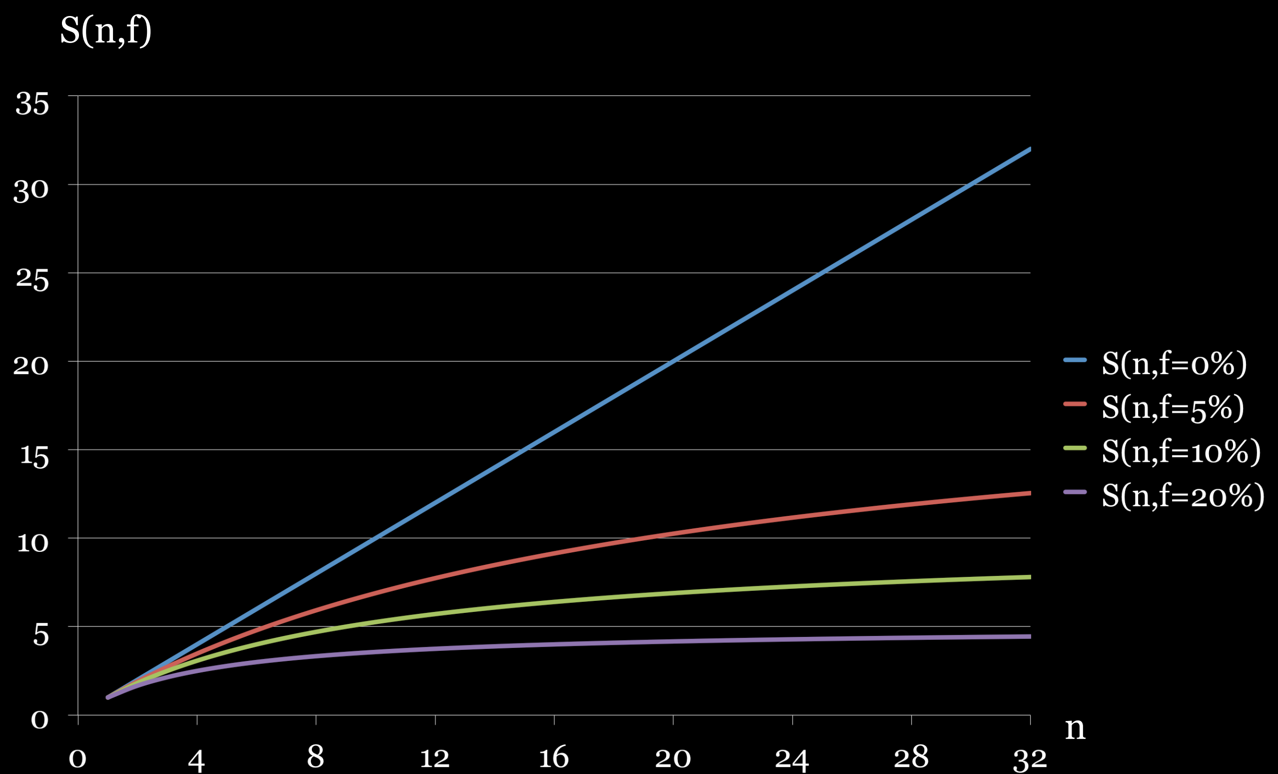
n = number of processors/cores

f = fraction of the computation that cannot be parallelised

$$S(n) = \frac{single\ processor\ execution\ time}{n\ processor\ execution\ time} = \frac{n}{1 + (n-1)f}$$

$$\lim_{n \to \infty} S(n) = \frac{1}{f}$$

Amdahl's law (3/3)



How to benefit from multiple CPUs/cores in Java

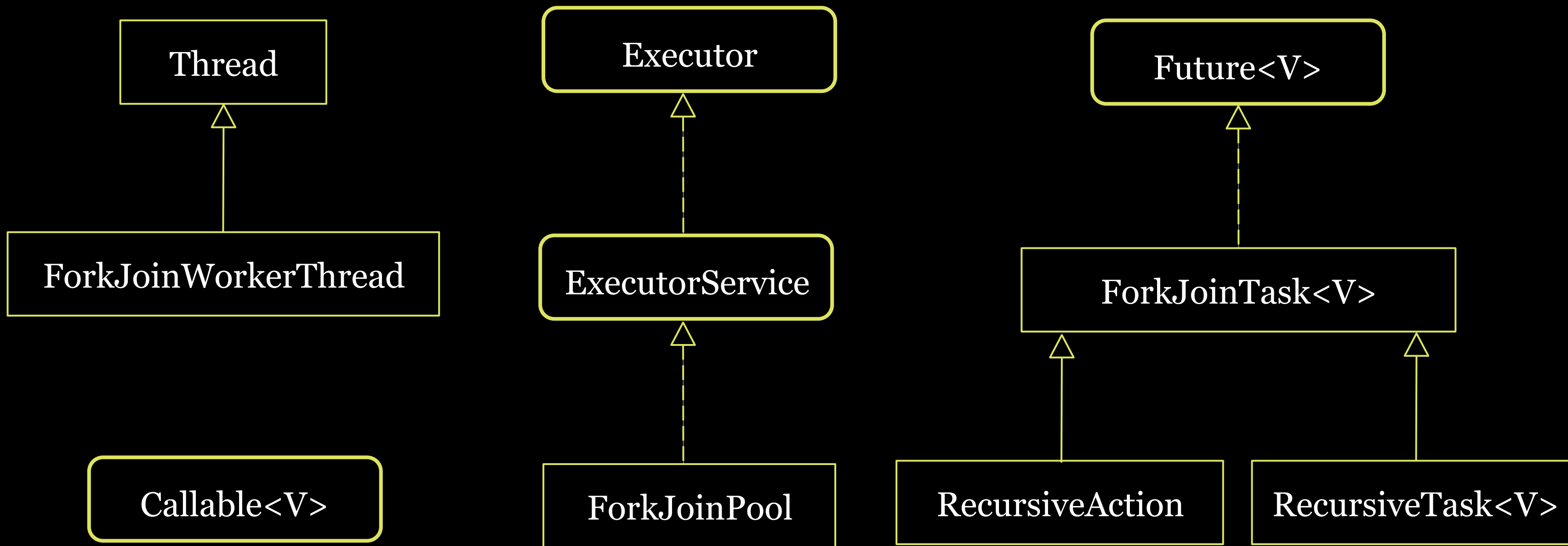
Implicitly – magic done under the hood by the JVM

Explicitly – using Java 7's fork/join framework

This session focuses on the fork/join framework

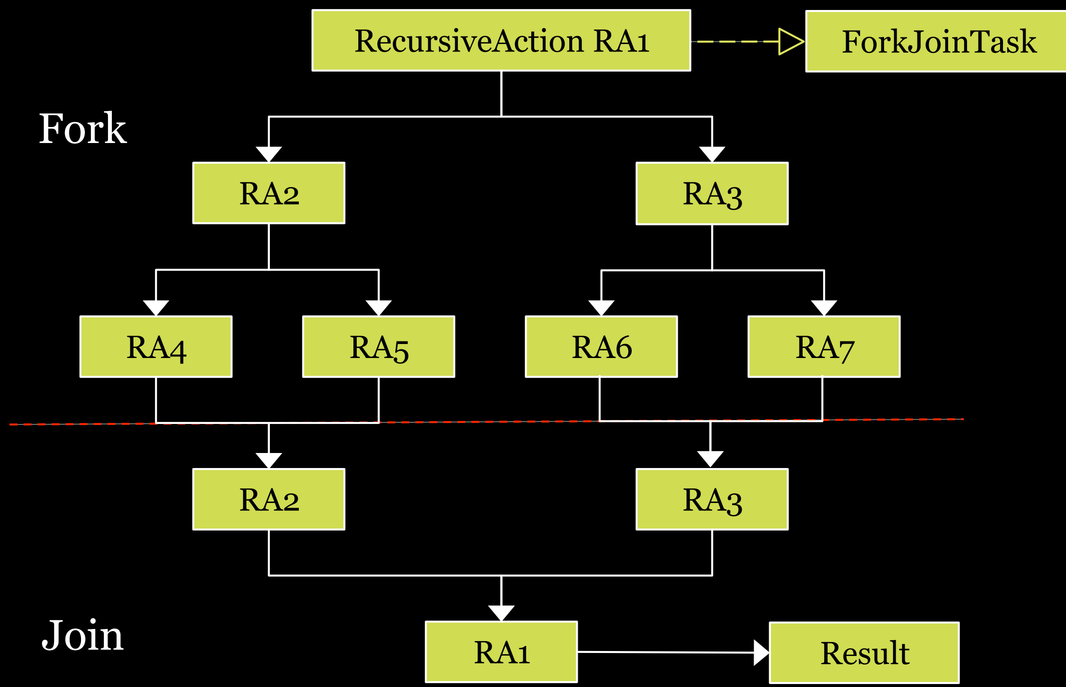
Part 2
New kids on the block

New classes on the block



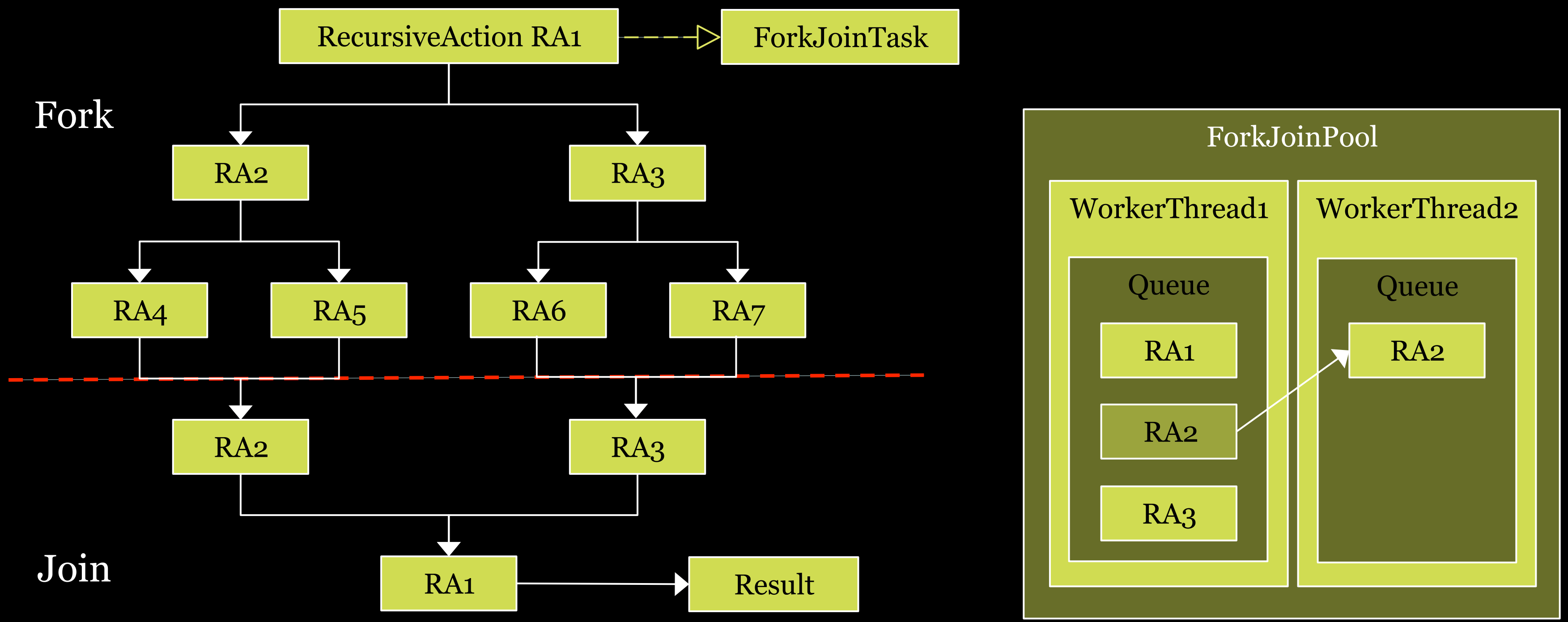
ThoughtWorks®

Anatomy of the fork/join interaction (1/2)



ThoughtWorks®

Anatomy of the fork/join interaction (2/2)



Part 3
All hands on deck

How to implement a ForkJoinTask generally

```
if (my portion of the work is small enough)
  do the work directly
else
  split my work into two pieces and fork them
  merge/join the results
```


ThoughtWorks®

Testing fork/join

Assert that...

... the complete task works as expected

... the core computation works as expected

... splitting the task does not corrupt input data

... merging results behaves as expected

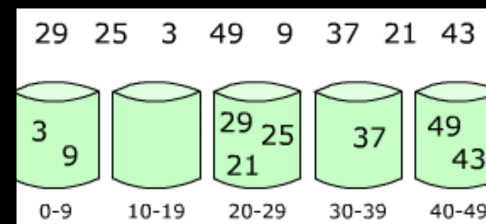
A guided example: Calculating checksums

Setting up your environment

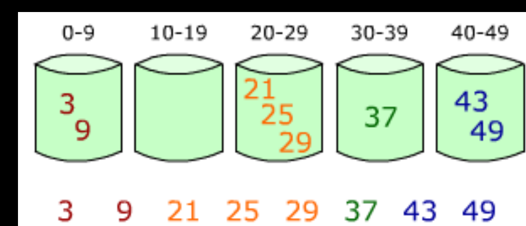
- Sources and slides are on the stick or clone them from
  - https://github.com/fkoehler/TW-Fork-Join-Workshop
- Follow the instructions in the README file
- Get the JDK from
  - http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1637583.html

Exercise 1: Bucket sort

- Implement a sequential and a parallel version of bucket sort
- Put elements into buckets that represent ranges



- Sort and concatenate buckets



- Source: http://en.wikipedia.org/wiki/Bucket_sort

ThoughtWorks®

Exercise 1: Bucket sort as pseudo code

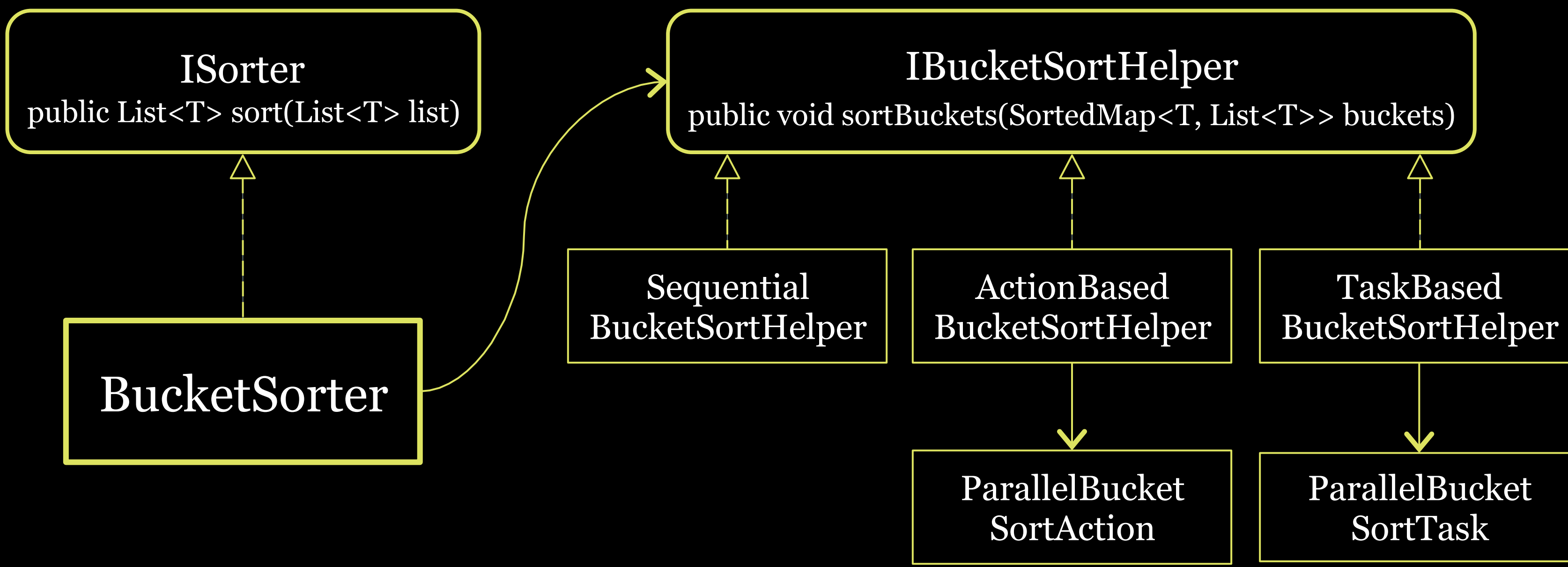
* ```
  function bucketSort(array, n) is
     buckets ← new array of n empty lists

     for i = 0 to (length(array)-1) do
        insert array[i] into buckets[whichBucket(array[i])]

     for i = 0 to n - 1 do
        nextSort(buckets[i])

     return the concatenation of buckets[0], ..., buckets[n-1]
  ```
* Source: http://en.wikipedia.org/wiki/Bucket_sort
  ```

Exercise 1: One way to realise bucket sort

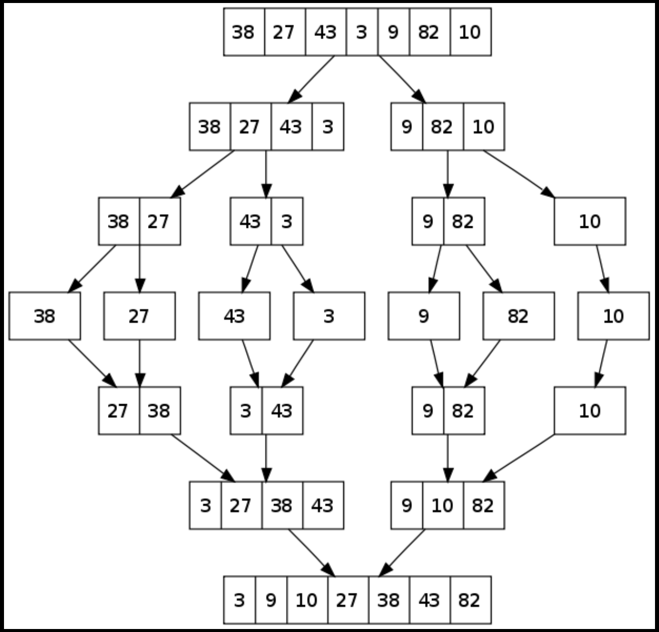


Exercise 2: Tuning the parallel bucket sort

- Buckets of equal width work well when the list to sort is evenly populated
- When the distribution of elements is skewed, the parallel bucket sort may degrade (why?)
- Run your parallel bucket sort with different kinds of lists (i.e. evenly populated ones and skewed ones)
- Improve the performance of your parallel bucket sort if required

Exercise 3: Merge sort

- Implement a sequential and a parallel version of merge sort
  - If the list is of length 0 or 1 the list is already sorted
  - Divide the unsorted list into two sublists of about half the size
  - Sort each sublist recursively by re-applying the merge sort
  - Merge the two sublists back into one sorted list



- Source: http://en.wikipedia.org/wiki/Merge_sort

ThoughtWorks®

Exercise 3: Merge sort as pseudo code (1/2)

```
function merge_sort(m)
    if length(m) ≤ 1
        return m
    var list left, right, result
    var integer middle = length(m) / 2
    for each x in m up to middle
         add x to left
    for each x in m after or equal middle
         add x to right

    left = merge_sort(left)
    right = merge_sort(right)
    result = merge(left, right)

    return result
```

 Source: http://en.wikipedia.org/wiki/Merge_sort

Exercise 3: Merge sort as pseudo code (2/2)

```
function merge(left,right)
    var list result
    while length(left) > 0 or length(right) > 0
        if length(left) > 0 and length(right) > 0
            if first(left) ≤ first(right)
                append first(left) to result
                left = rest(left)
            else
                append first(right) to result
                right = rest(right)
        else if length(left) > 0
            append first(left) to result
            left = rest(left)
        else if length(right) > 0
            append first(right) to result
            right = rest(right)
    end while
    return result
```

 Source: http://en.wikipedia.org/wiki/Merge_sort

Exercise 4: Searching strings in text files

- Implement a parallel search for a string in a text file
- The search string can be assumed to contain no whitespace (i.e. no spaces or tabs)
- Have a look at the following class:

  `com.thoughtworks.fjw.search.SimpleStringSearchTest`

  - The class contains examples for string based searching and reading from a text file

Exercise 5: Searching strings revisited

- Revisit your solution for the previous exercise
- Drop the assumption so that search strings can contain whitespace
- Consider how dropping the assumption affects the way you partition the text to search

Part 4
Wrap up

B. Wilkinson & M. Allen: Parallel Programming,
Prentice Hall, New Jersey 1999

Thank you

How can we help?

ThoughtWorks is a global custom software solutions consultancy trusted by many of the world's leading businesses with their most complex and critical systems.

We deliver consulting grounded in delivery expertise, build custom applications and help organisations across all market sectors to drive IT efficiency – working to an exceptionally high standard.

Contact us

Wolf Schlegel

+49 173 543 7465

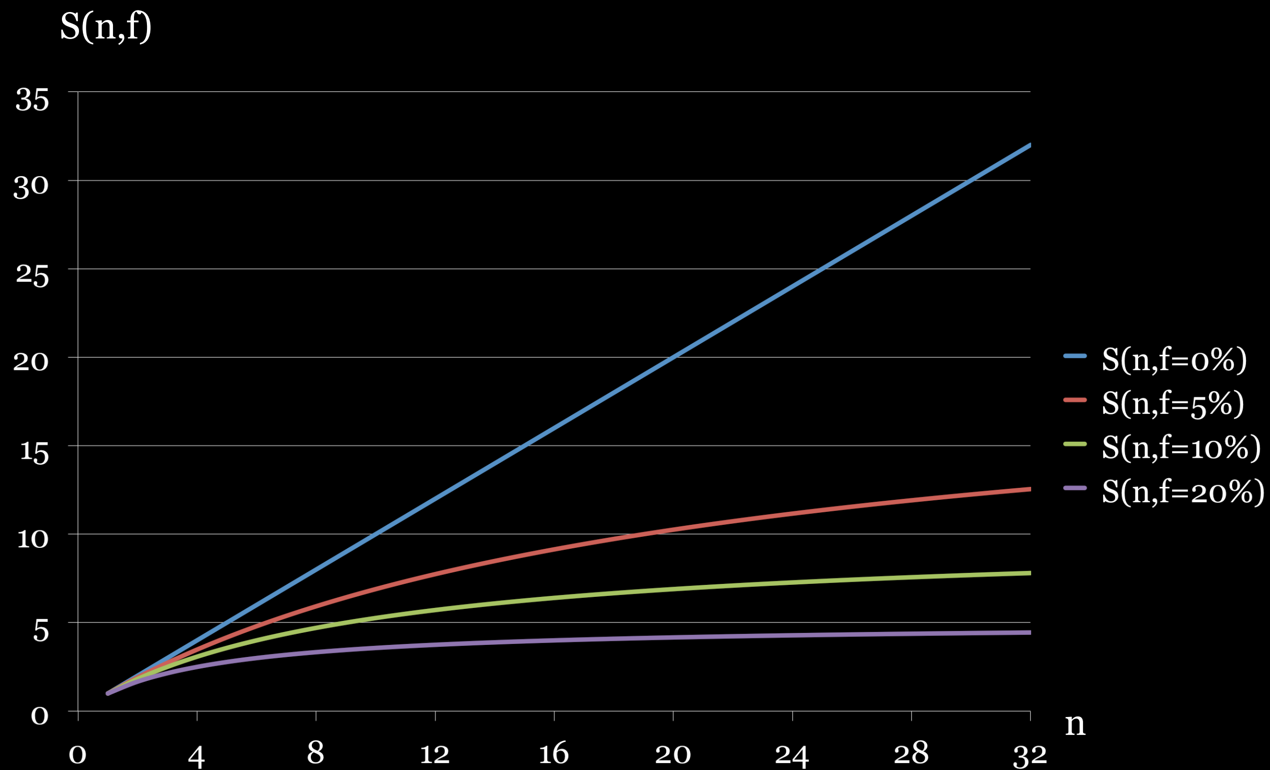
wolf.schlegel@thoughtworks.com

www.thoughtworks.com

ThoughtWorks®

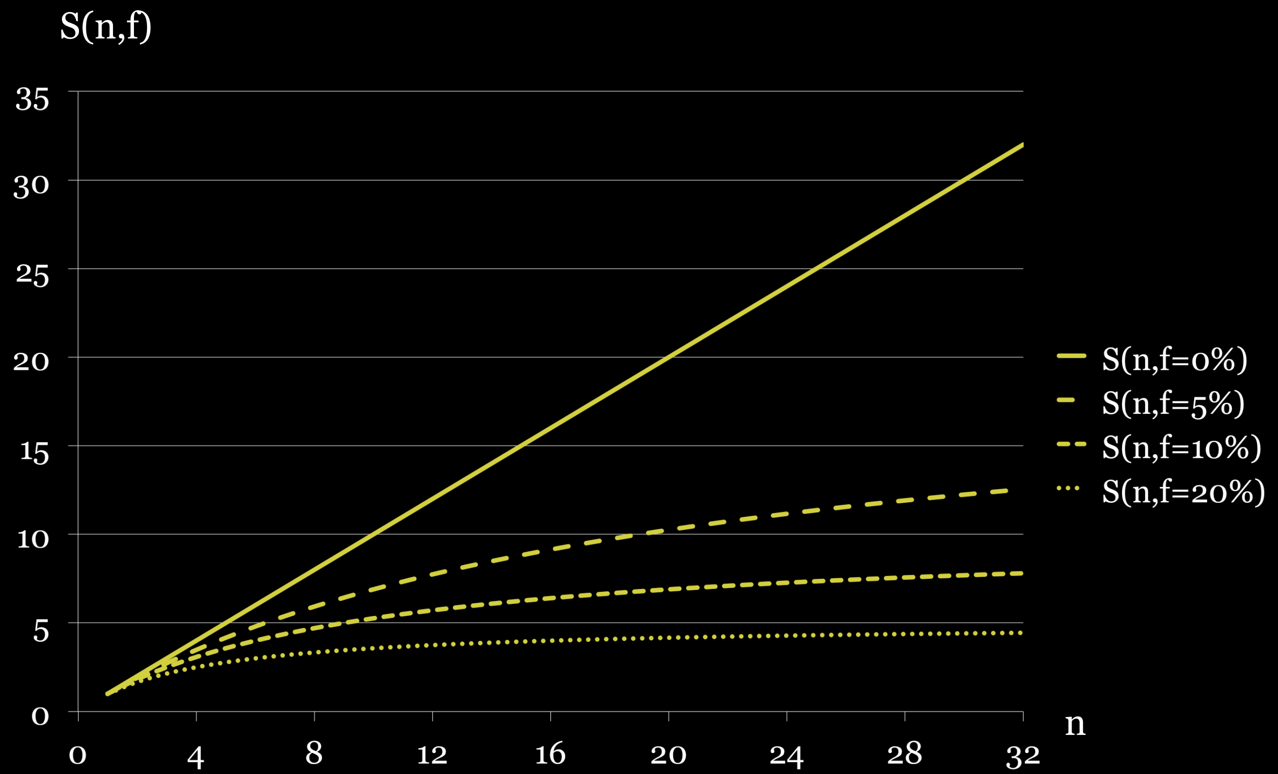
Backup slides

Amdahl's law (3/3)



ThoughtWorks®

Amdahl's law (3/3)



ThoughtWorks®

Thread state diagram