Technology Folklore

Martin Thompson & Dave Farley

http://code.google.com/p/disruptor/

http://www.davefarley.net

http://mechanical-sympathy.blogspot.com/

LMAX

Who are we?

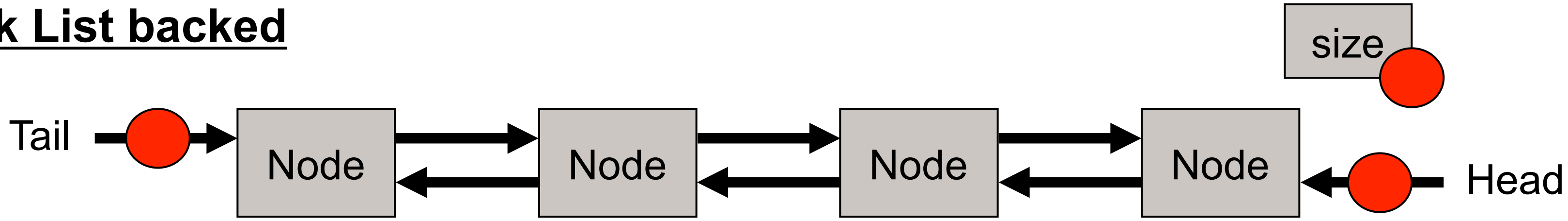
betfair

LMAX

Disruptor

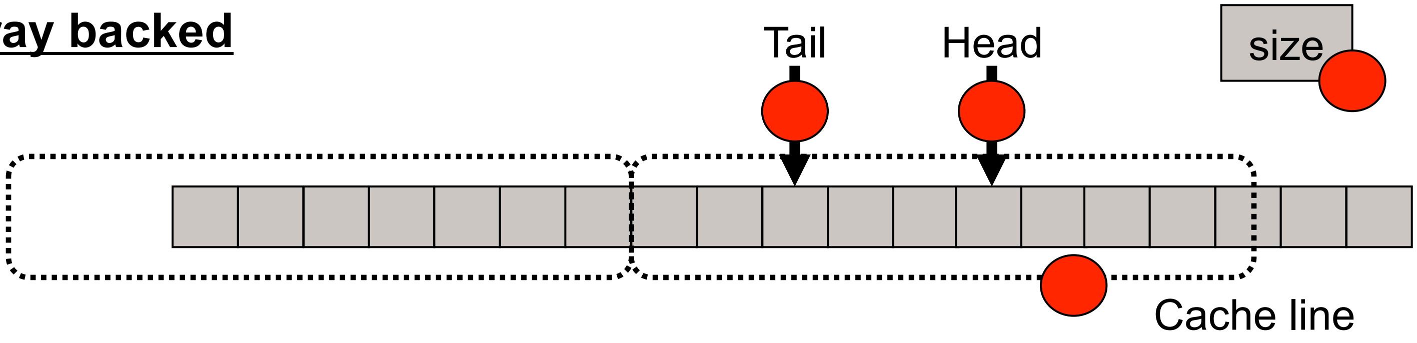
Sample Folklore:
Queues, an efficient way to exchange data

Link List backed



- Hard to limit size
- O(n) access times if not head or tail
- Generates garbage which can be significant

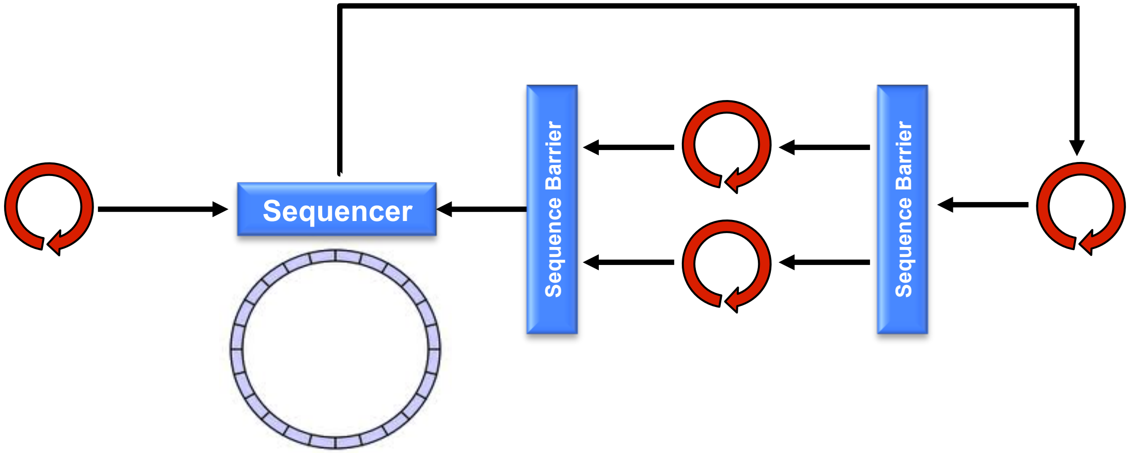
Array backed



- Cannot resize easily
- Difficult to get *P *C correct
- O(1) access times for any slot and cache friendly

Some Results

Disruptor



Test	Queue	Disruptor	Factor
OnePublisherToOneProcessorUniCastThroughputTest	2,366,171	72,087,993	30.5
OnePublisherToThreeProcessorDiamondThroughputTest	1,590,126	63,358,798	39.8
OnePublisherToThreeProcessorMultiCastThroughputTest	191,661	54,165,692	282.6
OnePublisherToThreeProcessorPipelineThroughputTest	1,289,199	71,562,125	55.5
OnePublisherToThreeWorkerPoolThroughputTest	2,175,593	10,412,567	4.8

A Question...

What is the most successful invention in human history?

A Question...

SCIENCE

What is the most successful invention in human history?

The Scientific Method

- **Characterization** Make a guess based on experience and observation.
- **Hypothesis** Propose an explanation.
- **Deduction** Make a prediction from the hypothesis.
- **Experiment** Test the prediction.

Stand Back!
We're going to try some science!

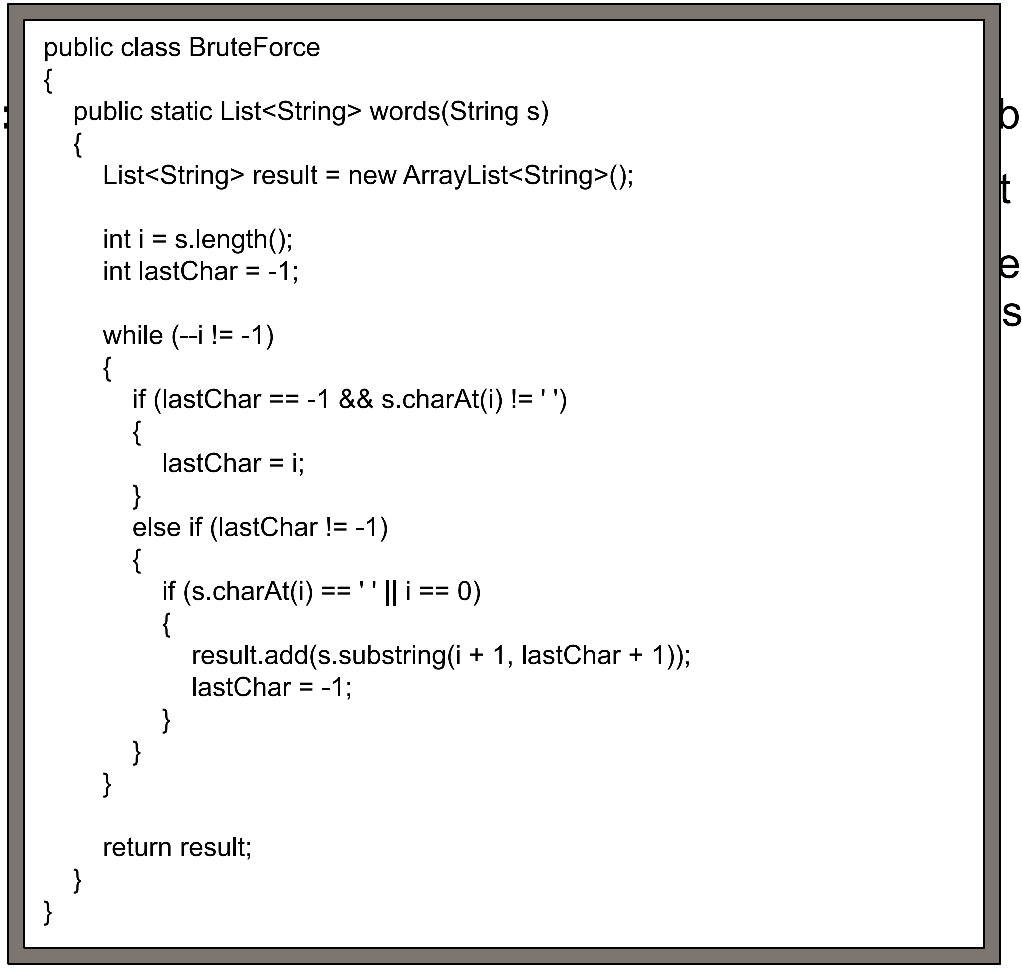
LMAX

Myth – CPU performance has stopped increasing

- **Characterization:** My computer is modern but my code is not noticeably faster.
- **Hypothesis:** We have reached the limits! CPU performance isn’t increasing anymore.
- **Deduction:** If this is the case then an algorithm run on the newest processors will perform at roughly the same rate as on older processors.
- **Experiment:** …

Myth – CPU performance has stopped increasing

- **Characterization:** bly faster.
- **Hypothesis:** t increasing anymore.
- **Deduction:** est processors will ssors.
- **Experiment:**



```
public class BruteForce
{
    public static List<String> words(String s)
    {
        List<String> result = new ArrayList<String>();

        int i = s.length();
        int lastChar = -1;

        while (--i != -1)
        {
            if (lastChar == -1 && s.charAt(i) != ' ')
            {
                lastChar = i;
            }
            else if (lastChar != -1)
            {
                if (s.charAt(i) == ' ' || i == 0)
                {
                    result.add(s.substring(i + 1, lastChar + 1));
                    lastChar = -1;
                }
            }
        }

        return result;
    }
}
```

Myth – CPU performance has stopped increasing

- **Characterization:** My computer is modern but my code is not noticeably faster.
- **Hypothesis:** We have reached the limits! CPU performance isn't increasing anymore.
- **Deduction:** If this is the case then an algorithm run on the newest processors will perform at roughly the same rate as on older processors.
- **Experiment:** …

Processor Name	Model	Operations/sec	Release Date
Intel(R) Core 2 Duo(TM)	CPU P8600 @ 2.40GHz	1434	(2006)
Intel(R) Xeon(R)	CPU E5620 @ 2.40GHz	1768	(2009)
Intel(R) Core(TM)	CPU i7-2677M @ 1.80GHz	2202	(2010)
Intel(R) Core(TM)	CPU i7-2720QM @ 2.20GHz	2674	(2010)

Myth – Go Parallel to scale – part I

- **Characterization:** I can do more work by executing tasks in parallel.
- **Hypothesis:** I can increase the rate at which I do work by increasing the number of threads that I do work on.
- **Deduction:** If this is the case then we should be able to measure higher throughput as we add more threads.
- **Experiment:** Let's increment a 64 bit counter, a simple Java long, 500 million times…

Method	Time (ms)
Single thread	300
Single thread with lock	10,000
Two threads with lock	224,000
Single thread with CAS	5,700
Two threads with CAS	30,000

Myth – Go Parallel to scale – part II

- **Characterization:** I can do more work by executing tasks in parallel.
- **Hypothesis:** I can increase the rate at which I do work by increasing the number of threads that I do work on.
- **Deduction:** If this is the case then we should be able to measure higher throughput as we add more threads.
- **Experiment:** …

Myth – Go Parallel to scale – part II

- **Characterization:** I can do more work by executing tasks in parallel
- **Hypoth** of
- **Deduct** nput
- **Experir**

The Experiment:

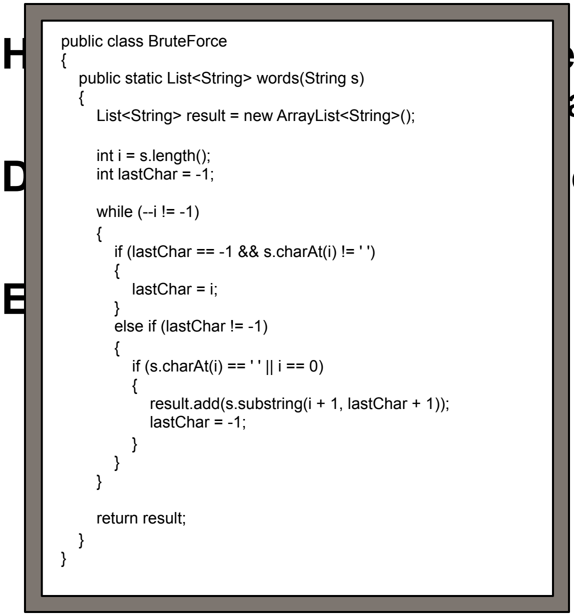
From Guy Steele's talk at the Strange Loop Conference

(http://www.infoq.com/presentations/Thinking-Parallel-Programming)

Tested with copy the text of ‘Alice in Wonderland’

Myth – Go Parallel to sc

- **Characterization:** I can do more work b
- H ease the rat at I do work
- D e case ther more threa
- E

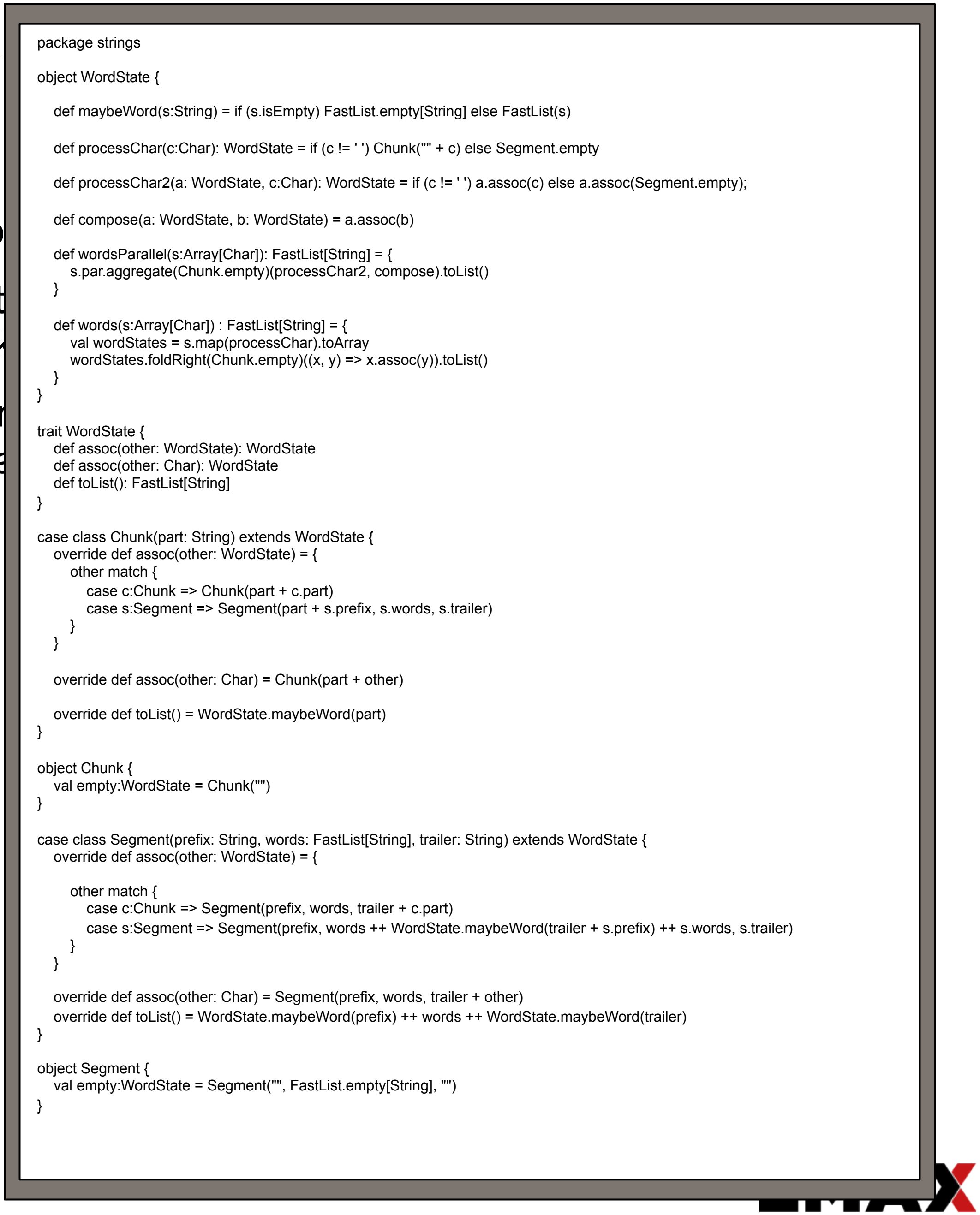


```
public class BruteForce
{
    public static List<String> words(String s)
    {
        List<String> result = new ArrayList<String>();

        int i = s.length();
        int lastChar = -1;

        while (--i != -1)
        {
            if (lastChar == -1 && s.charAt(i) != ' ')
            {
                lastChar = i;
            }
            else if (lastChar != -1)
            {
                if (s.charAt(i) == ' ' || i == 0)
                {
                    result.add(s.substring(i + 1, lastChar + 1));
                    lastChar = -1;
                }
            }
        }

        return result;
    }
}
```



```
package strings

object WordState {

    def maybeWord(s:String) = if (s.isEmpty) FastList.empty[String] else FastList(s)

    def processChar(c:Char): WordState = if (c != ' ') Chunk("" + c) else Segment.empty

    def processChar2(a: WordState, c:Char): WordState = if (c != ' ') a.assoc(c) else a.assoc(Segment.empty);

    def compose(a: WordState, b: WordState) = a.assoc(b)

    def wordsParallel(s:Array[Char]): FastList[String] = {
        s.par.aggregate(Chunk.empty)(processChar2, compose).toList()
    }

    def words(s:Array[Char]) : FastList[String] = {
        val wordStates = s.map(processChar).toArray
        wordStates.foldRight(Chunk.empty)((x, y) => x.assoc(y)).toList()
    }
}

trait WordState {
    def assoc(other: WordState): WordState
    def assoc(other: Char): WordState
    def toList(): FastList[String]
}

case class Chunk(part: String) extends WordState {
    override def assoc(other: WordState) = {
        other match {
            case c:Chunk => Chunk(part + c.part)
            case s:Segment => Segment(part + s.prefix, s.words, s.trailer)
        }
    }

    override def assoc(other: Char) = Chunk(part + other)

    override def toList() = WordState.maybeWord(part)
}

object Chunk {
    val empty:WordState = Chunk("")
}

case class Segment(prefix: String, words: FastList[String], trailer: String) extends WordState {
    override def assoc(other: WordState) = {

        other match {
            case c:Chunk => Segment(prefix, words, trailer + c.part)
            case s:Segment => Segment(prefix, words ++ WordState.maybeWord(trailer + s.prefix) ++ s.words, s.trailer)
        }
    }

    override def assoc(other: Char) = Segment(prefix, words, trailer + other)
    override def toList() = WordState.maybeWord(prefix) ++ words ++ WordState.maybeWord(trailer)
}

object Segment {
    val empty:WordState = Segment("", FastList.empty[String], "")
}
```

Myth – Go Parallel to scale – part II

- **Characterization:** I can do more work by executing tasks in parallel.
- **Hypothesis:** I can increase the rate at which I do work by increasing the number of threads that I do work on.
- **Deduction:** If this is the case then we should be able to measure higher throughput as we add more threads.
- **Experiment:** …

Test	Lines of Code	Ops/Sec
Scala: Parallel Collections	61	400
Java: Imperative single threaded solution	33	1,600

Myth – Adding a batching algorithm increases latency

- **Characterization:** Adding a batching algorithm increases latency
- **Hypothesis:** Waiting for the batch to fill will always add latency
- **Deduction:** If this is the case then we can never exceed the maximum rate at which a serial approach will work.
- **Experiment:** …

Myth – Adding a batching algorithm increases latency

- **Characterization:** Adding a batching algorithm increases latency
- **Hypothesis:** /
- **Deduction:** naximum rate at which
- **Experiment:**

Send 10 concurrent messages to an IO device with 100us latency

1. Batching can be implemented as a wait with a timeout
2. Send what is available as soon as possible then loop

Myth – Adding a batching algorithm increases latency

- **Characterization:** Adding a batching algorithm increases latency
- **Hypothesis:** Waiting for the batch to fill will always add latency
- **Deduction:** If this is the case then we can never exceed the maximum rate at which a serial approach will work.
- **Experiment:** …

	Min (us)	Mean (us)	Max (us)
Serial	100	500	1000
Batch Type 2	100	190	200

- ***Little's Law comes into play on points of serialisation***

Common Folklore We Have Encountered

- Queues are an efficient way to do message passing
- SSDs are much faster than spinning disks
- Operating system schedulers do the right thing
- A local network hop is expensive
- JDK Collection classes are high performance
- Transactional systems need a relational database
- Common messaging platforms are fast
- Java serialization for marshalling objects
- TCP is the obvious protocol for communications
- XML parsers are fast enough
- Short lived objects are free
- You must build high performance systems in C++

Wrap up

Q & A

jobs@lmax.com