What the Enterprise Can Learn from Your Mom

Jim Weirich
Chief Scientist
EdgeCase

EdgeCase
software artisans

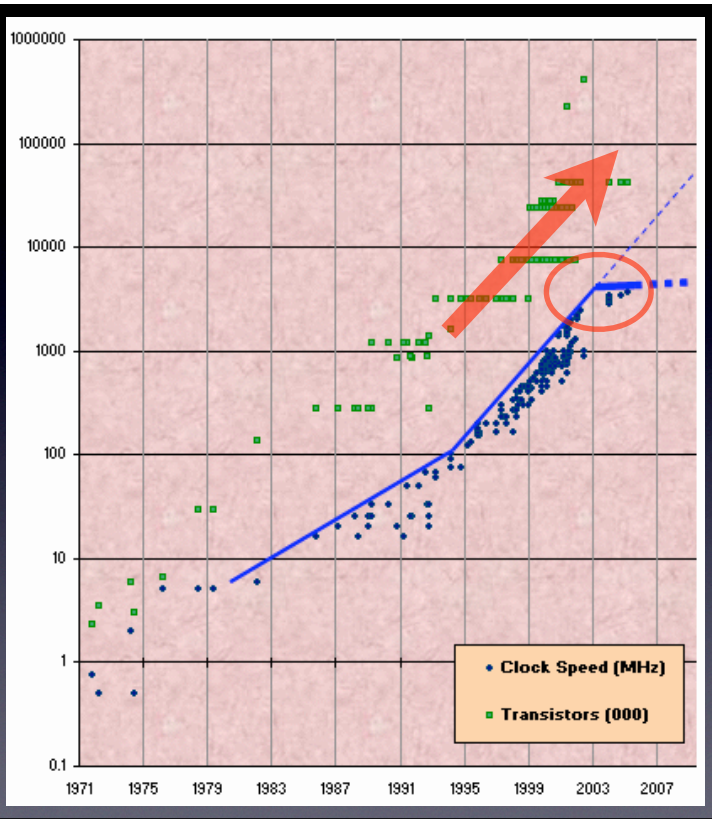
Mom?

TANSTAAFL

The Free Lunch Is Over
A Fundamental Turn Toward Concurrency in Software
By Herb Sutter

- http://www.gotw.ca/publications/concurrency-ddj.htm
  - Published early 2005

Moore's Law

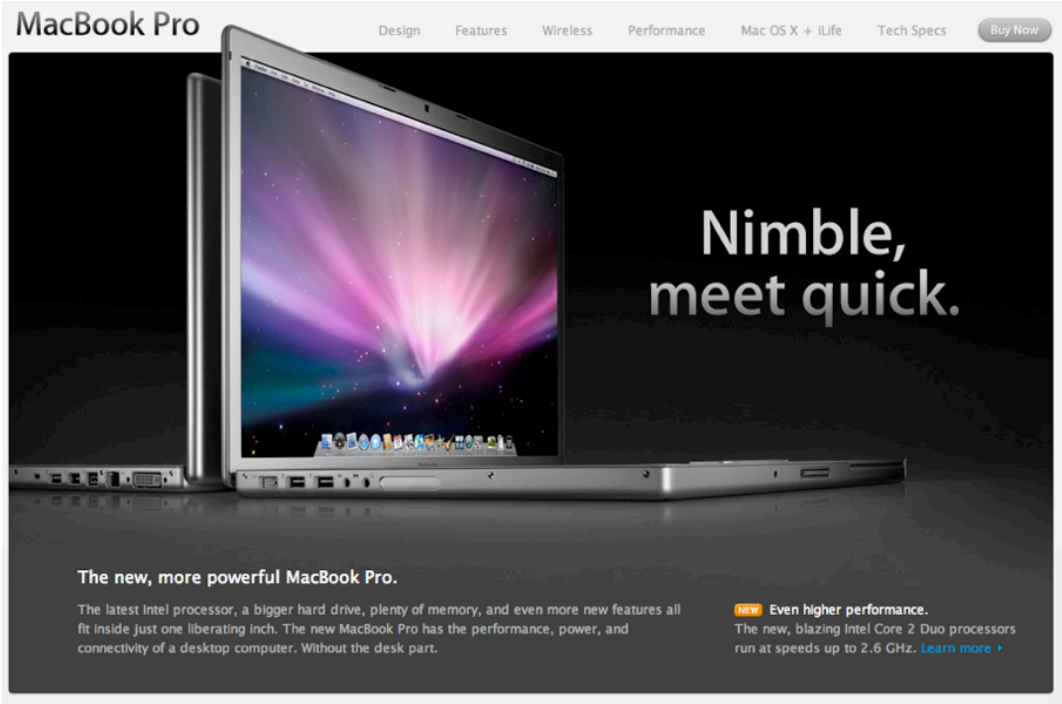


Past Performance Gains

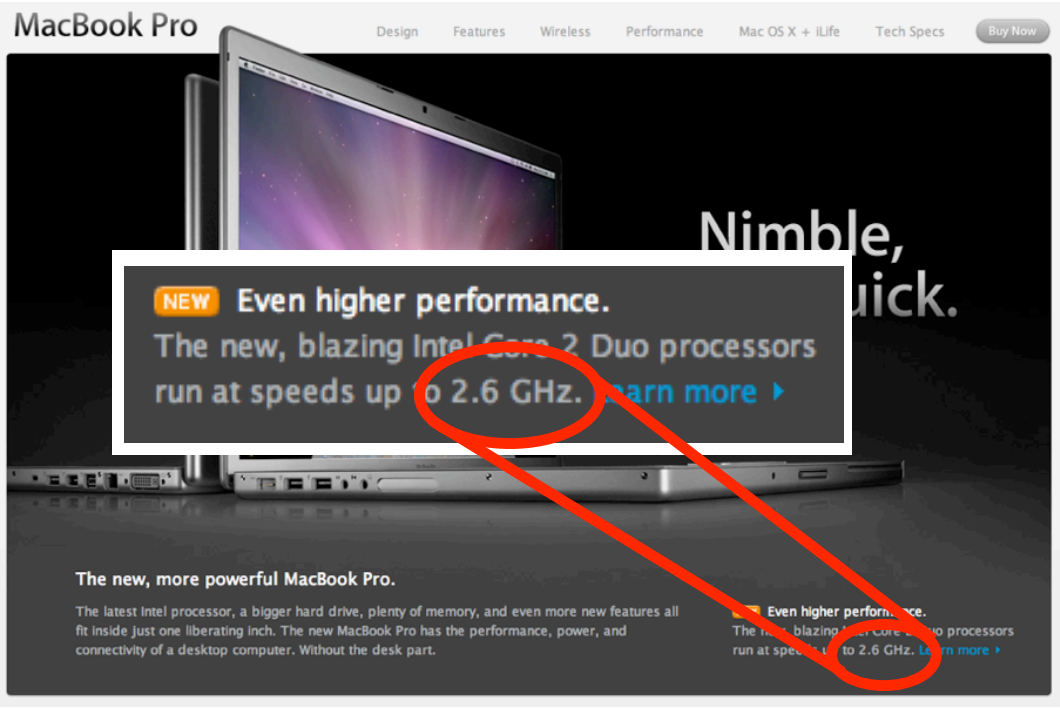
- Clock Speed
- Execution Optimization
- Cache


About This Mac
Mac OS X
Version 10.5.4
Software Update...
Processor 2 GHz Intel Core Duo
Memory 2 GB 667 MHz DDR2 SDRAM
More Info...
TM & © 1983–2008 Apple Inc.
All Rights Reserved.

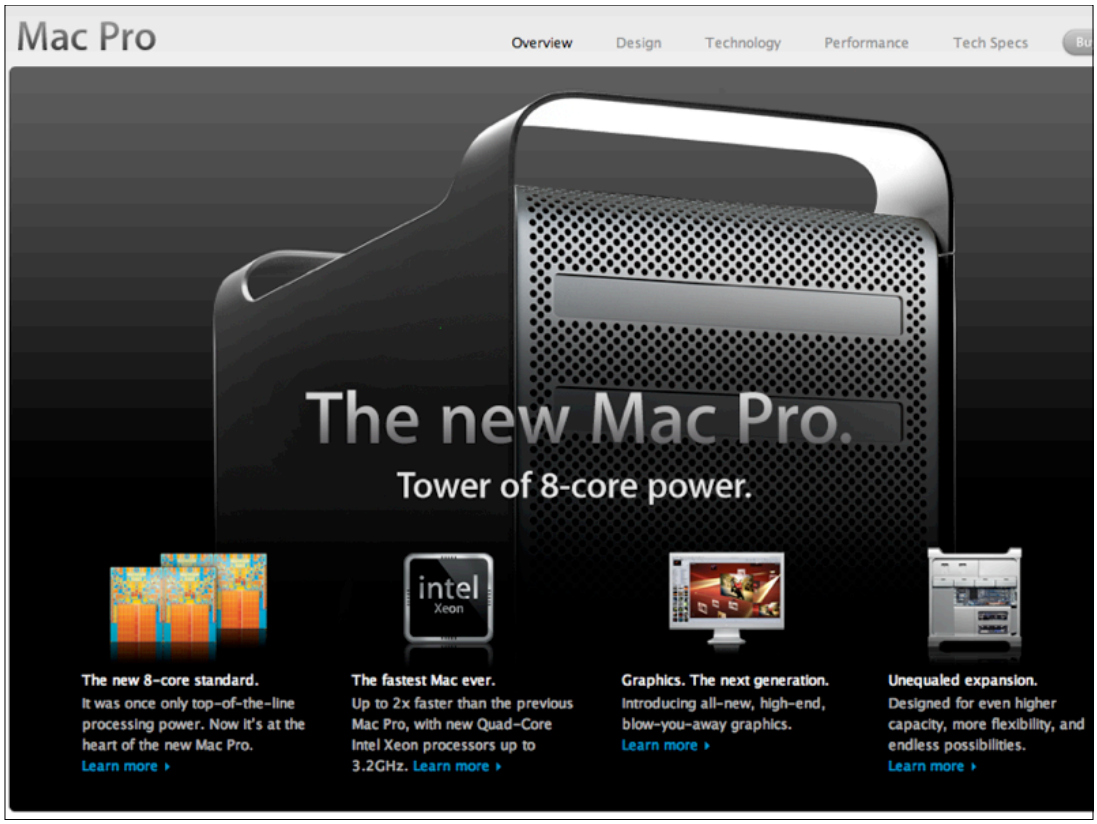
7


MacBook Pro
Design
Features
Wireless
Performance
Mac OS X + iLife
Tech Specs
Buy Now
Nimble,
meet quick.
The new, more powerful MacBook Pro.
The latest Intel processor, a bigger hard drive, plenty of memory, and even more new features all fit inside just one liberating inch. The new MacBook Pro has the performance, power, and connectivity of a desktop computer. Without the desk part.
NEW Even higher performance.
The new, blazing Intel Core 2 Duo processors run at speeds up to 2.6 GHz. Learn more

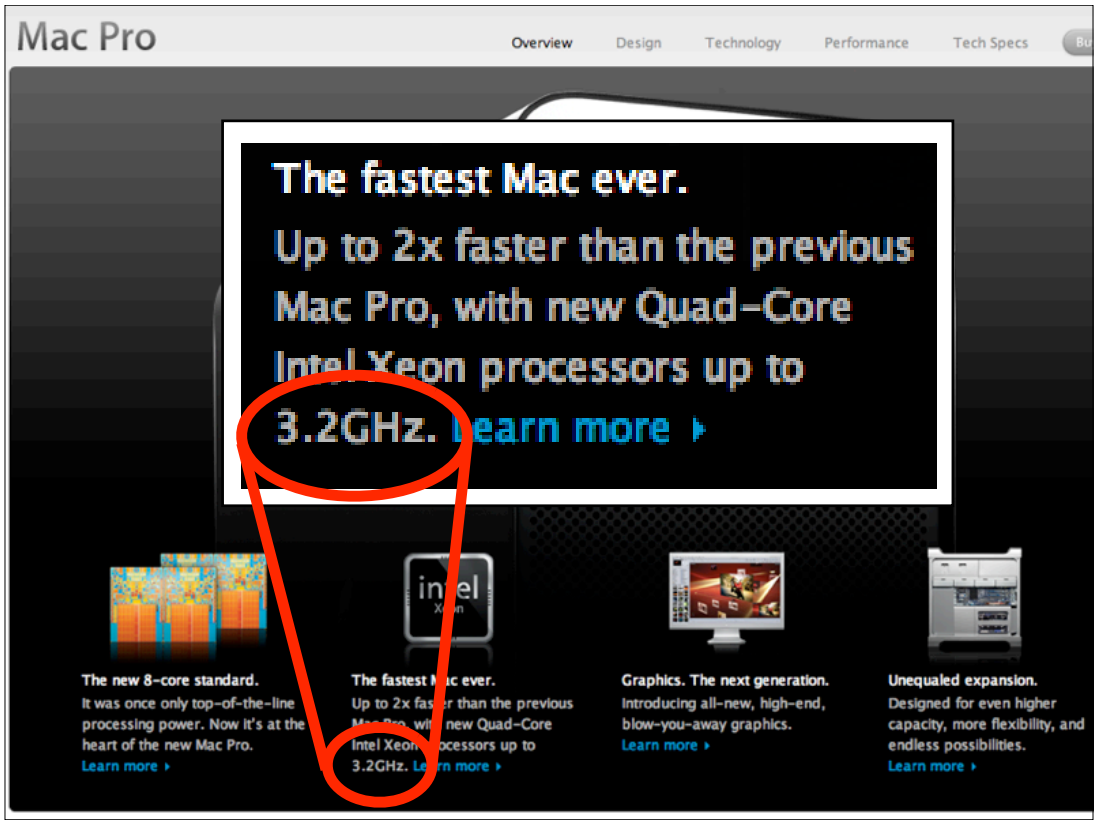
8


MacBook Pro
Design
Features
Wireless
Performance
Mac OS X + iLife
Tech Specs
Buy Now
Nimble,
NEW
Even higher performance.
The new, blazing Intel Core 2 Duo processors
run at speeds up to 2.6 GHz.
Learn more
The new, more powerful MacBook Pro.
The latest Intel processor, a bigger hard drive, plenty of memory, and even more new features all fit inside just one liberating inch. The new MacBook Pro has the performance, power, and connectivity of a desktop computer. Without the desk part.

9

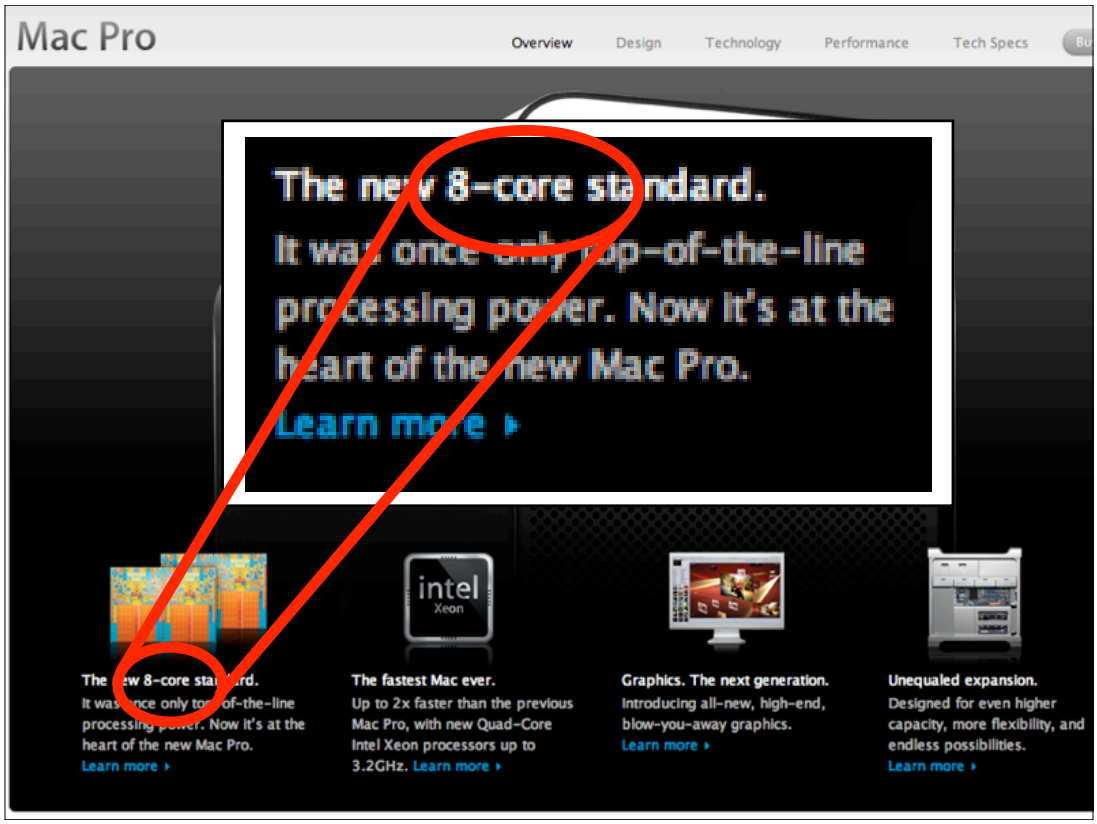

Mac Pro
Overview
Design
Technology
Performance
Tech Specs
The new Mac Pro.
Tower of 8-core power.
intel
Xeon
The new 8-core standard.
It was once only top-of-the-line processing power. Now it's at the heart of the new Mac Pro.
Learn more
The fastest Mac ever.
Up to 2x faster than the previous Mac Pro, with new Quad-Core Intel Xeon processors up to 3.2GHz. Learn more
Graphics. The next generation.
Introducing all-new, high-end, blow-you-away graphics.
Learn more
Unequaled expansion.
Designed for even higher capacity, more flexibility, and endless possibilities.
Learn more

10



Future Performance Gains

- Hyperthreading
- Multicore
- Cache



Back to Herb Sutter!

Applications will increasingly need to be concurrent if they want to fully exploit continuing exponential CPU throughput gains

Efficiency and performance optimization will get more, not less, important

100 Core CPUs?

So, What *Can* the Enterprise Learn from Your Mom?

17

However ...

18

Charles Miller

In its place I would put *Java Concurrency in Practice*. Every new Atlassian developer gets handed this book and ordered to read it immediately. ***Writing multi-threaded code is hard***, and a number of the things Java does under the hood to make its multi-threading more efficient makes it even harder. Unless you understand the subtleties described in this book of how Java shares data between threads, ***you will screw it up*** in some almost-impossible-to-debug way. [emphasis mine]

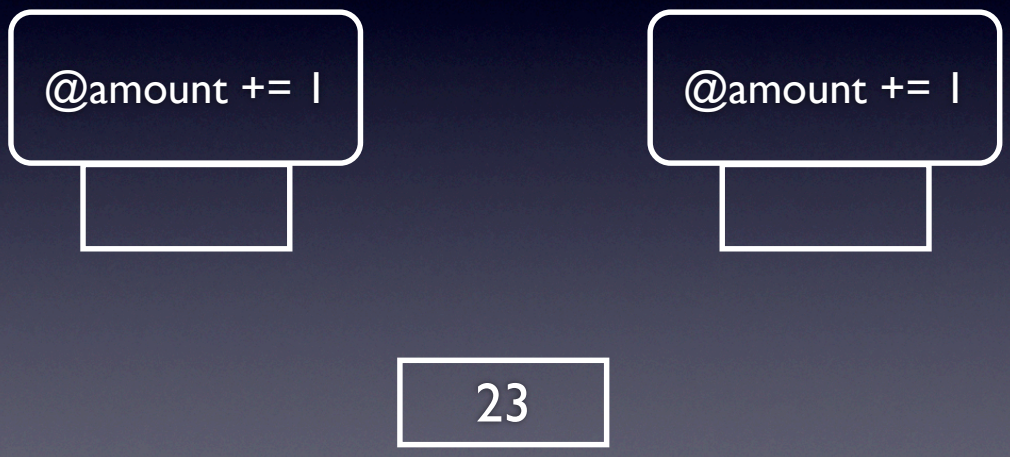
http://fishbowl.pastiche.org/2008/08/07/recommended_reading_for_java_d/

So, you want to write a concurrent program ...

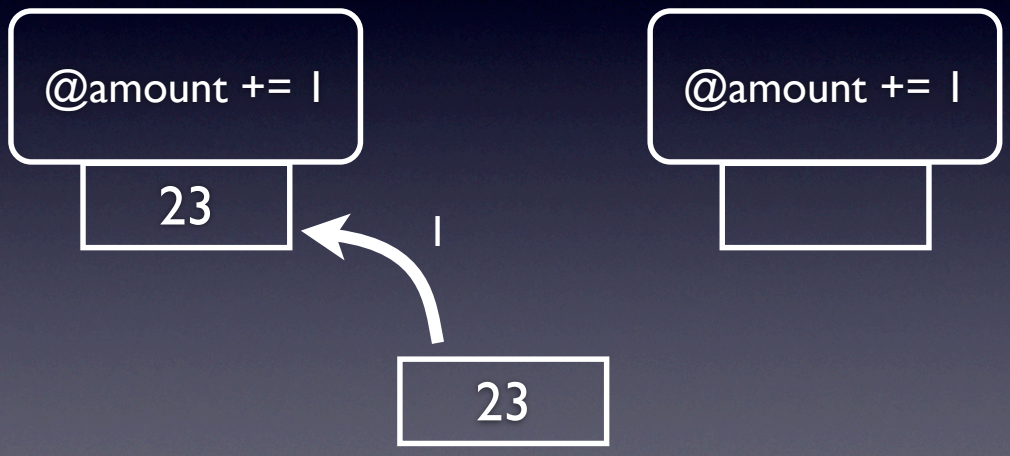
Demo: race1.rb

What happened?

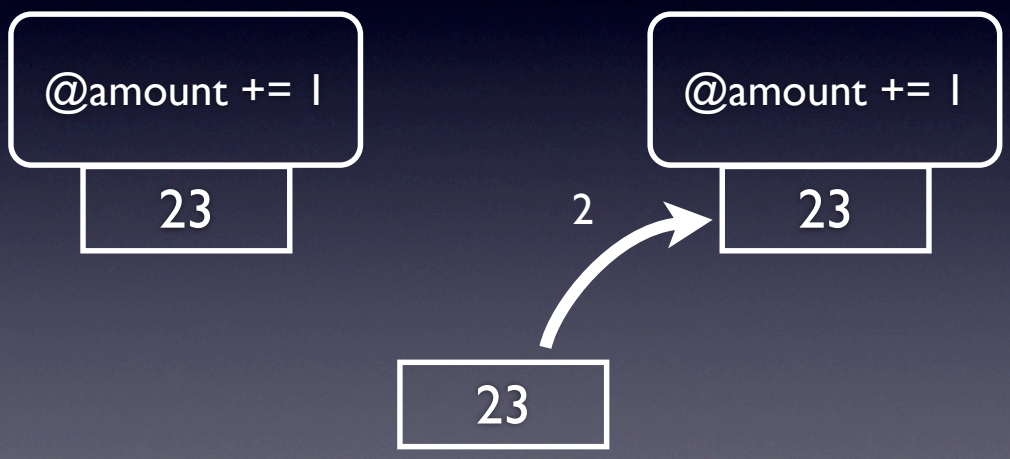
The Setup



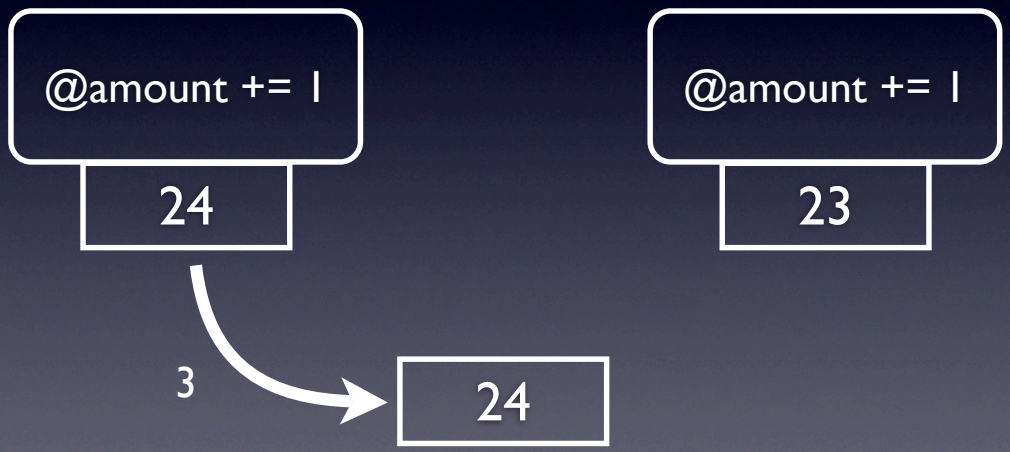
Step 1



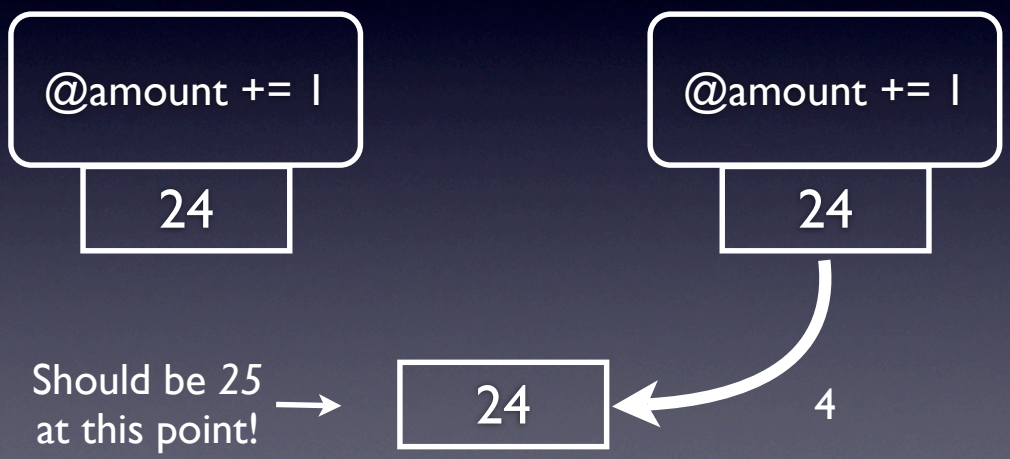
Step 2



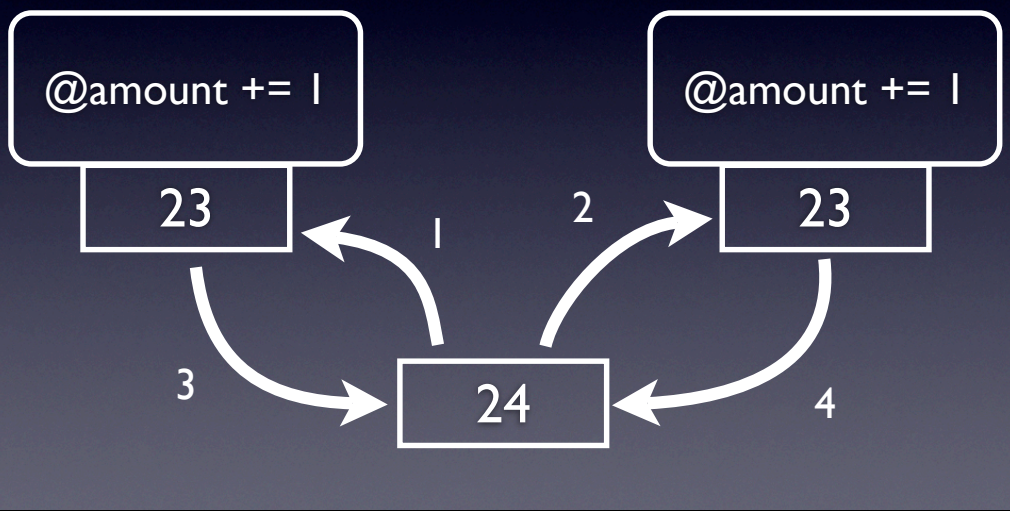
Step 3



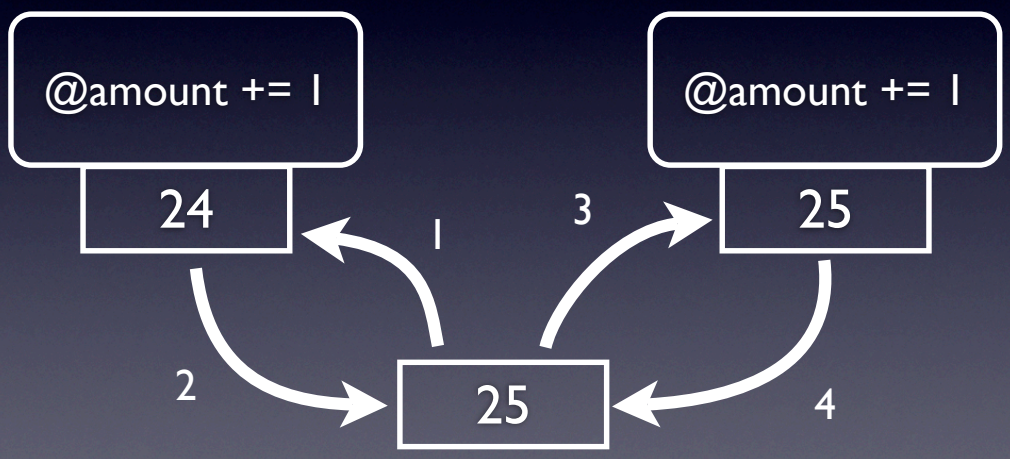
Step 4



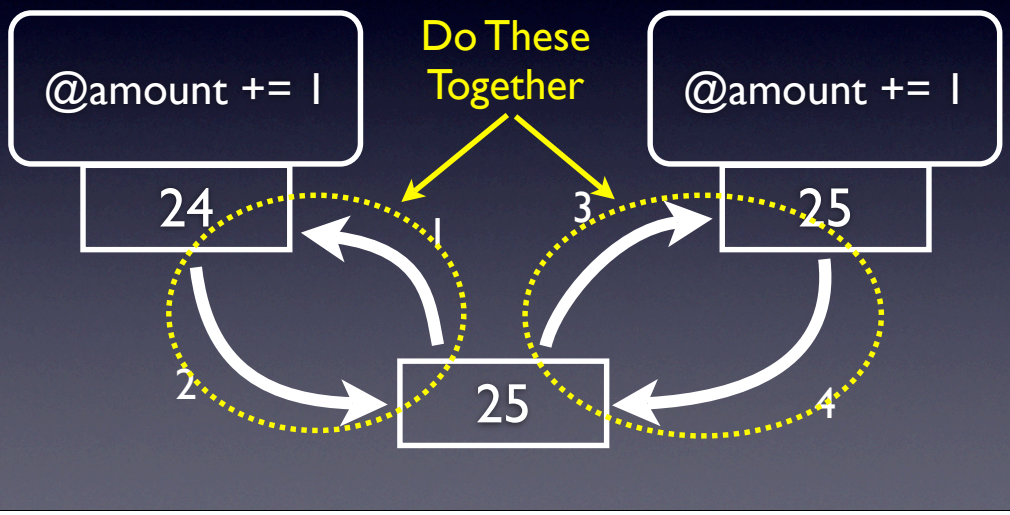
Race Condition



Reordering Steps



Mutual Exclusion



```
DISABLE CONTEXT SWITCHING
  account.credit(1)
RE-ENABLE CONTEXT SWITCHING
```



```
require 'thread'
mutex = Mutex.new

...

mutex.synchonize do
  account.credit(1)
end
```

Demo: race4.rb

Demo: race7.rb

To Be Safe, You Must:

(1) Protect ***every*** shared memory access with a synchronizing lock.

Yes, ***EVERY*** access.

To Be Safe, You Must:

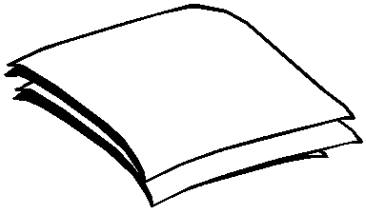
(2) Be aware of extended situations that need to be atomic.

To Be Safe, You Must:

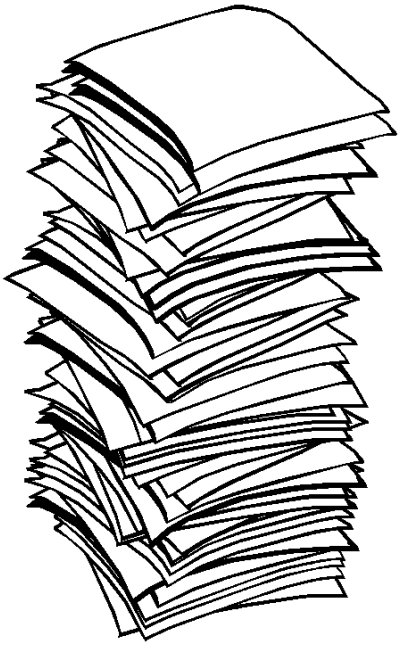
(3) Have a strategy to avoid deadlock in the presence of multiple locks.

To Be Safe, You Must:

(4) Evaluate every single library used by your program to see if they also follow rules 1 - 3.



41



42

Horror Stories

- Real time data collection (1 in a million)
- Rake multitask dependencies
- Double Checked Lock

Double Checked Lock

```
public static Singleton getInstance()
{
  if (instance == null)
  {
    synchronized(Singleton.class) {  //1
      if (instance == null)          //2
        instance = new Singleton();  //3
    }
  }
  return instance;
}
```

The theory behind double-checked locking is perfect. Unfortunately, reality is entirely different.

Concurrent Programming is

HARD

And it is hard because of

Shared Memory

What can we do to make concurrent programming easier?

Avoid

Shared Memory

Are We Blug Programmers?

... Languages less powerful than Blub are obviously less powerful, because they're missing some feature he's used to. But when our hypothetical Blub programmer looks in the other direction, up the power continuum, he doesn't realize he's looking up. What he sees are merely weird languages... Blub is good enough for him, because he thinks in Blub.

-- *Paul Graham, Beating the Averages*

The Power of Messsaging

(Erlang)

Imagine a Language with

- No variables
- No assignment statements
- No explicit loops

Imagine a Language with

- No variables
- No assignment statements
- No explicit loops

- Only Constants
- Pattern Matching
- Recursion
  - (tail recursion)

Function Definitions

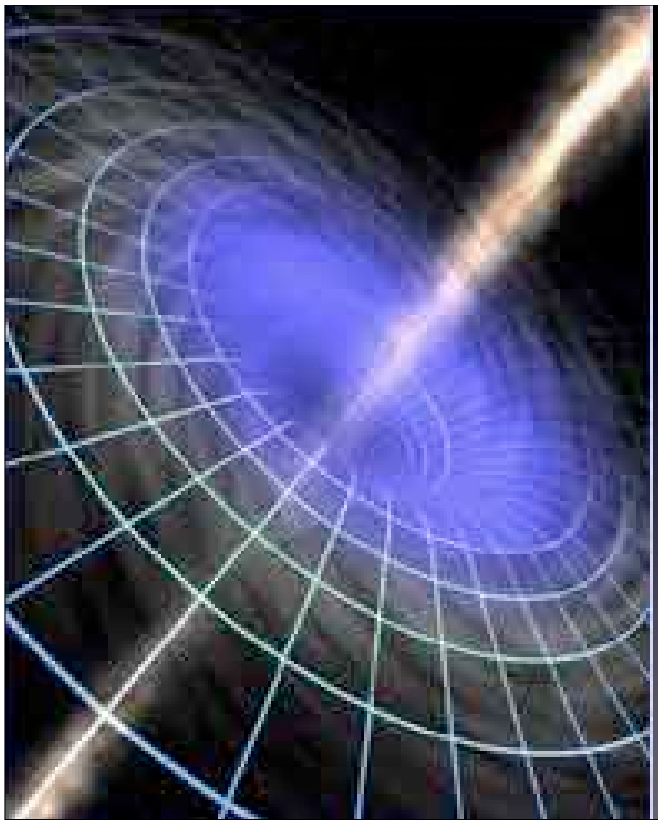
```
fact(0) ->
    1;
fact(N) ->
    N * fact(N-1).
```


Function Definitions

```
fact2(N) ->
    fact2(N, 1).

fact2(0, Acc) ->
    Acc;
fact2(N, Acc) ->
    fact2(N-1, N * Acc).
```

Erlang Demo

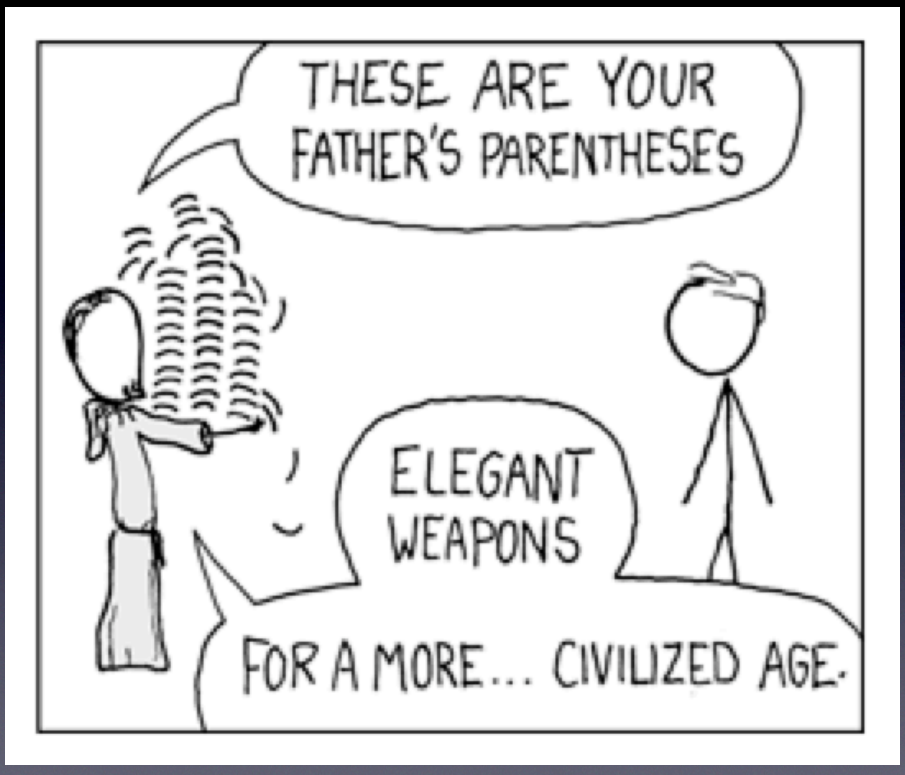


Bending Time and Space

(Clojure)

What? Lisp?

(isn't that Epic Fail?)



Quick Lisp Primer

Data Structures

```
1 321          ; Numbers
a fido         ; Names
(1 2 3)        ; Listss
[1 2 3]        ; Arrays
```

Calling Functions

```
(+ 2 4)                ; => 6
(count '(a b c))       ; => 3
```

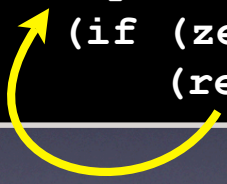
```
'(+ 2 4)   ; => (+ 2 4)
```


Defining Functions

```
(defn factorial [n]
  (if (zero? n) 1
      (* n (factorial (- n 1))) ))
```

Defining Functions

```
(defn factorial [n]
  (loop [i n  acc 1]
    (if (zero? i) acc
        (recur (- i 1) (* i acc) ))
```



Sequences

```
(def s '(peanut butter and jelly))

(first s)   ; => peanut
(rest s)    ; => (butter and jelly)
(cons 'fresh s)
            ; => (fresh peanut butter
                  and jelly)

(take 2 s)  ; => (peanut butter)
(drop 2 s)  ; => (and jelly)
```

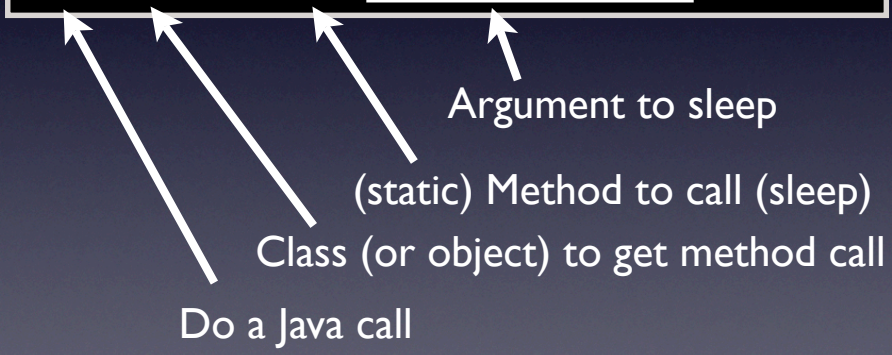
Other Cool Stuff

```
(repeat 1)          ; => (1 1 1 1 1 1...)
(cycle '(a b))      ; => (a b a b a b...)
(iterate (fn [n] (+ 1 n)) 0)
                    ; => (0 1 2 3 4 5...)
```


Java Interface

```
(import '(java.util.concurrent Executors))

(. Thread (sleep (* 1000 seconds))
```



No Modifiable State

(except ...)

- Vars - Thread Local Variables (impossible to share)

```
(def v 123)

(binding [v 321]  v)  # => 321
```

- v is bound to 321, but only for the current thread.

No Modifiable State

(except ...)

- Refs - STM (Sharable, but only in a transaction)

```
(def r (ref 0)
(deref r) ; => 0

(ref-set r 1)   ; FAILS
(dosync
  (ref-set r 1)) ; In Transaction
(deref r)  ; => 1
```

Clojure Demo

Summary

Concurrent Programming is Hard
(primarily due to shared mutable state)

Hard enough that you probably want to
avoid doing it in a traditional sequential
programming language.

Functional Languages provide many advantages when dealing with concurrency.

Don’t be a Blub Programmer.

Thank You!

git://github.com/jimweirich/presentation_enterprise_mom.git

Contact Info

- Jim Weirich
- Web: http://onestepback.org
- EMail: jim.weirich@gmail.com
- Twitter: jimweirich