8th april 2008

www.parisjug.org

Concurrency & Performance

www.parisjug.org

Xebia aneo the other solution OceaneConsulting Prestation Java BK Consulting

Sponsors

Xebia

aneo the other solution

OceaneConsulting Prestation Java

BK Consulting

License

Paternité-Pas d'Utilisation Commerciale-Partage des Conditions Initiales à l'Identique
2.0 France

http://creativecommons.org/licenses/by-nc-sa/2.0/fr/

Disclaimer

Any performance tuning advice provided in this presentation.....

will be wrong!

Me

- Work as independent (a.k.a. freelancer)
  - performance tuning services
  - benchmarking
  - Java performance tuning course and seminars
- Co-author: www.javaperformancetuning.com
- Contributing editor: www.theserverside.com
- Nominated Sun Java Champion
- Blah blah blah

Change the way you think about performance tuning

Motivation

Changes in hardware are now redefining the rules of coding, design, and Architecture

Old Thinking

Rethink Architecture

My View

- Started performance tuning in late 80s
- Cray supercomputers
  - Fortran, C, CAL, Special purpose languages
- Special Purpose Devices (VHDL)
- Smalltalk Systems
- Java Platform (97)

How did we get better performance?

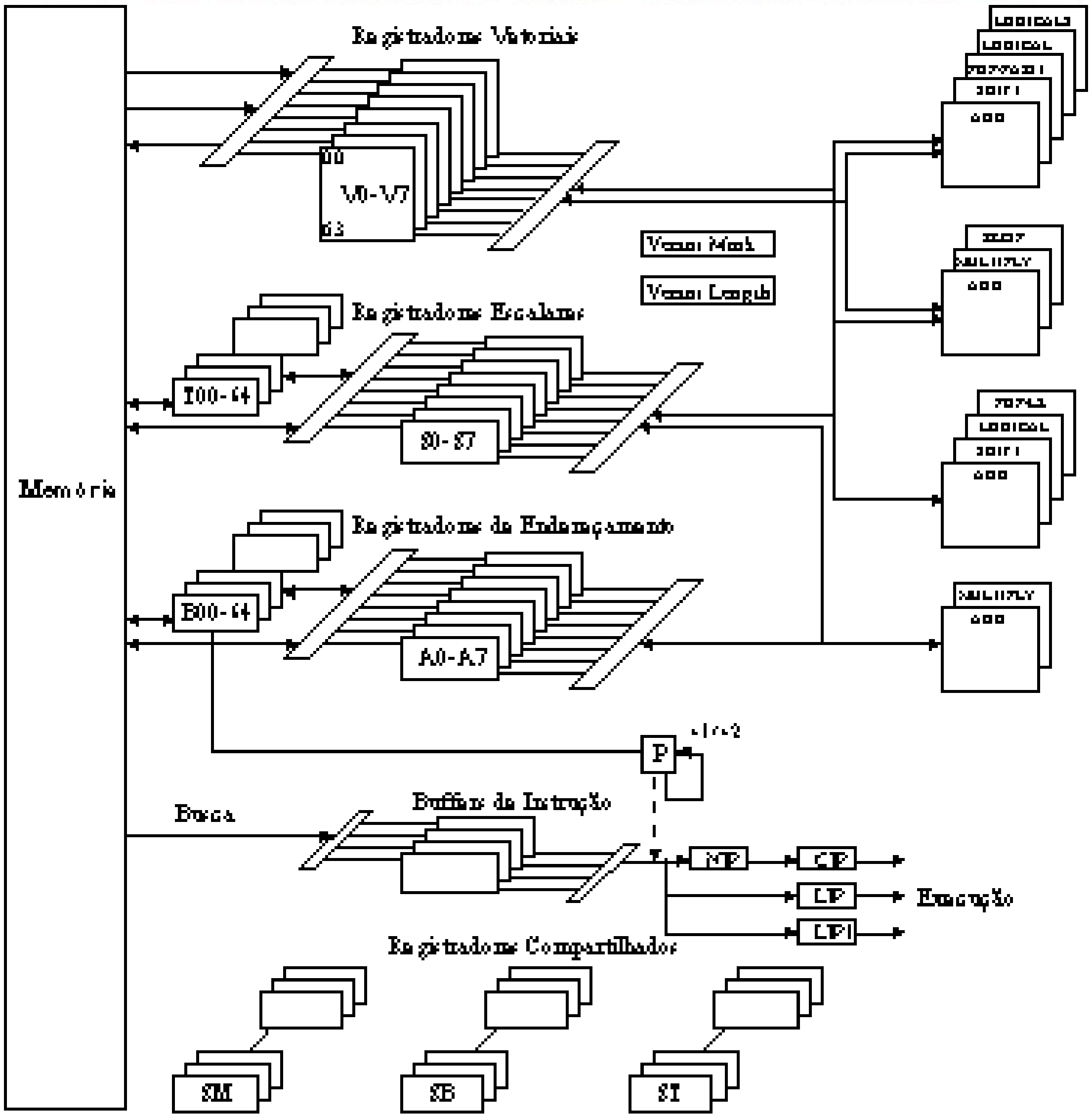
Historical Improvements

- Sometimes better algorithms
- Mostly faster Hardware
  - Clock speeds (read CPU)
  - Bus
  - Memory
  - Networks
- Exotic hardware

Machine Specific Optimizations

Needed to study existing or create new hardware

Cray CPU Block Diagram



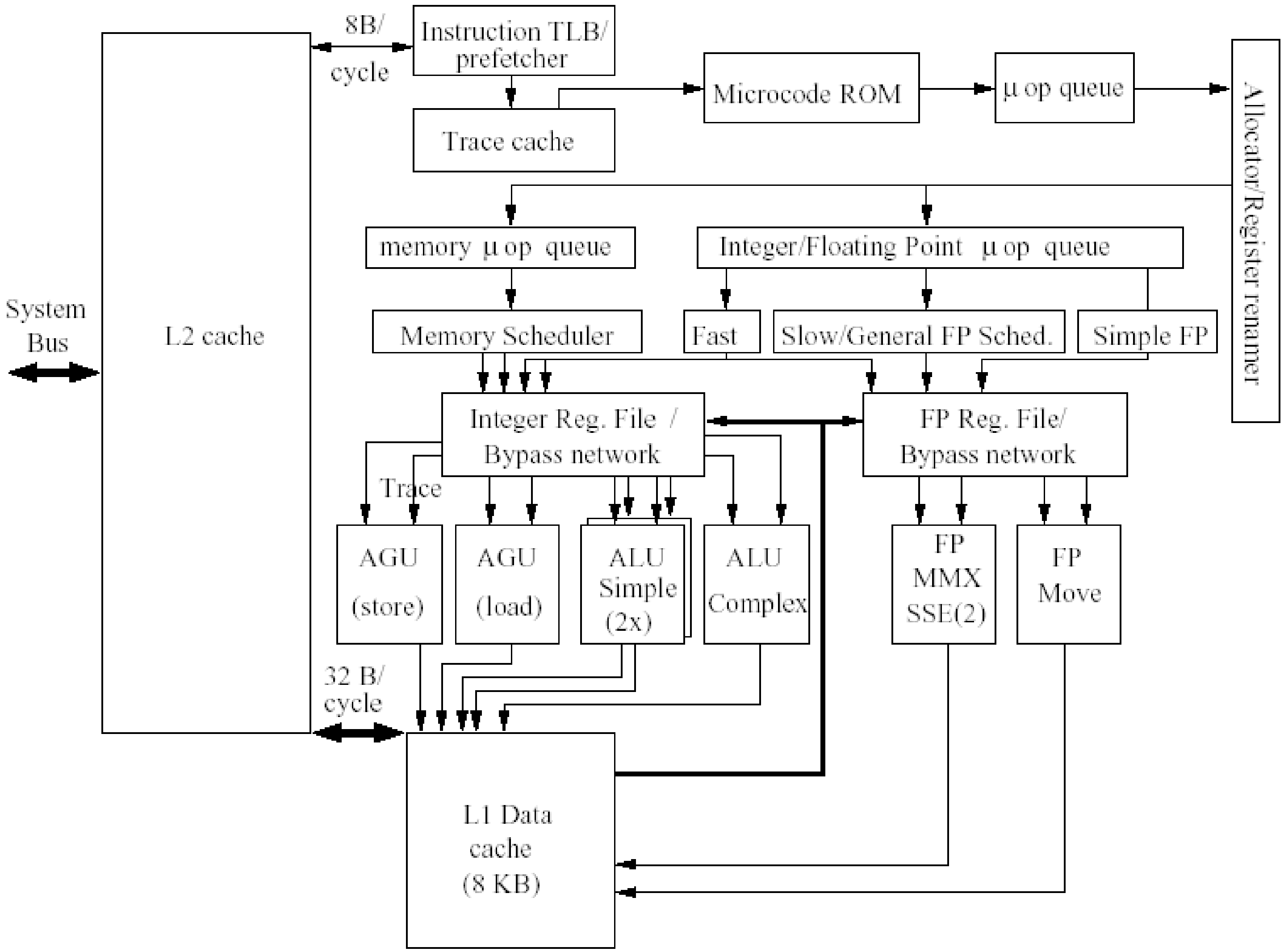
Developers Adapted to Hardware

- Code needed to utilize key features altering coding style
  - Short loops with no branching
  - regular memory strides
    - always increment loop counters by 1
  - statistically acceptable errors
- Align short loops and functions on instruction buffer boundaries

#pragma _CRI align function1, ...

National Center for Atmospheric Research

Intel Zeon Block Diagram



High Performance Early Days

Parallel Computing

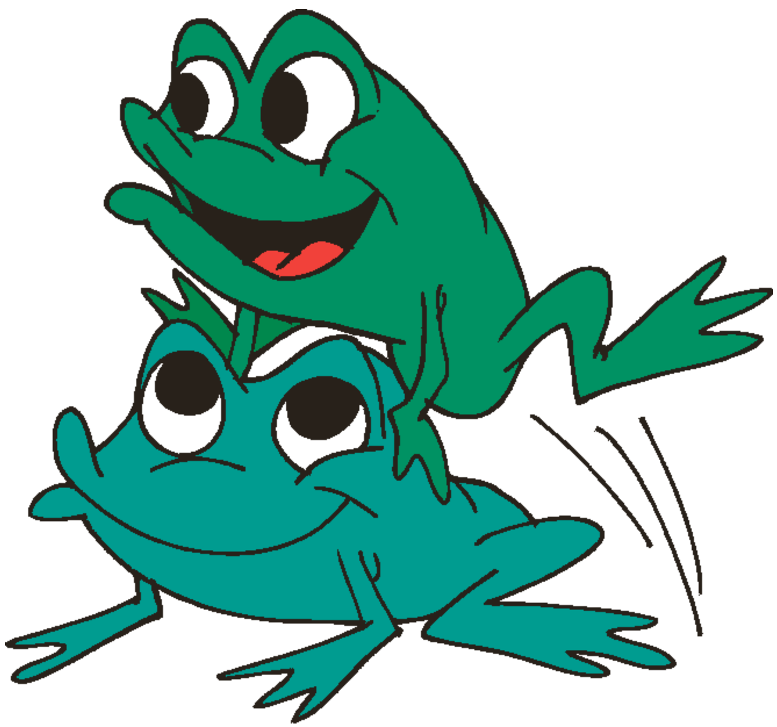
Threading Early On

- Process mostly single threaded
- "threading" limited for forking the process

```
if ( fork() == 0)
    childProcess();
    exit();
else
    parentProcess();
```

- Clumsy at best
  - two different processes
  - difficult to share results
  - difficult to coordinate activities

Hardware/OS/Language Leapfrog



Posix Threading Support (Early 90s)

- UNIX kernels single threaded (80)
- SunOS is made SMP safe (91)
  - entire kernel is protected with a single lock
  - threaded in 93
- AIX pthread support 93?
- Windows NT released 93
  - simplified alternative to pthreads
- HP-UX POSIX suffers setback (95)

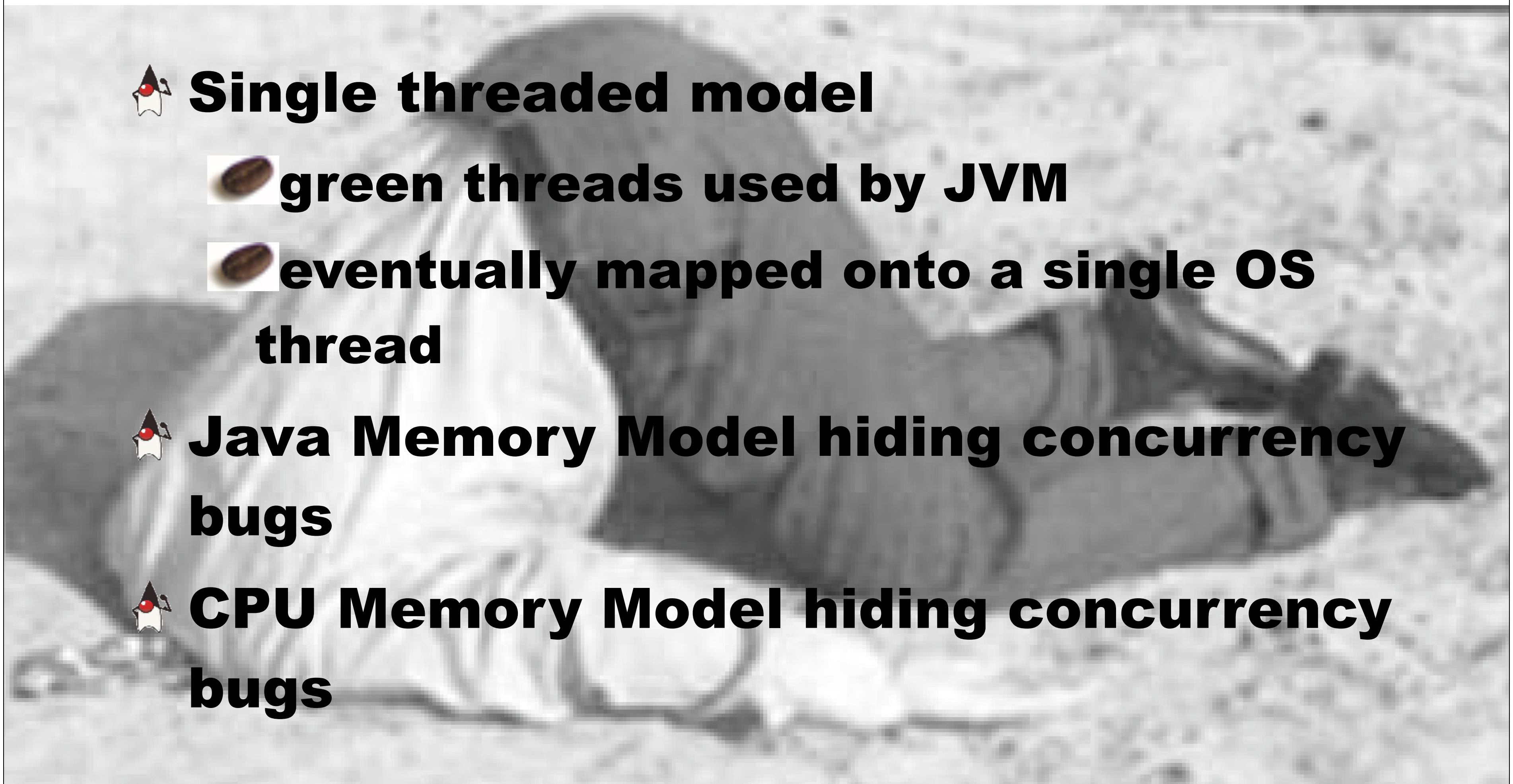
Languages Play Catch-up

- Java Platform explodes onto the scene (96)
  - support for distributed and parallel computing
- Strong play to virtualize hardware
  - cross-platform threading model

Java Thread Support

- Synchronized statement and modifier
  - map to OS level locks
- volatile keyword
  - no one knows what it does
- java.lang.Thread
- java.lang.Object.wait()
- java.lang.Object.notify()
- java.lang.Object.notifyAll()

Java Threading 1.0



- Single threaded model
  - green threads used by JVM
  - eventually mapped onto a single OS thread
- Java Memory Model hiding concurrency bugs
- CPU Memory Model hiding concurrency bugs

Memory Models

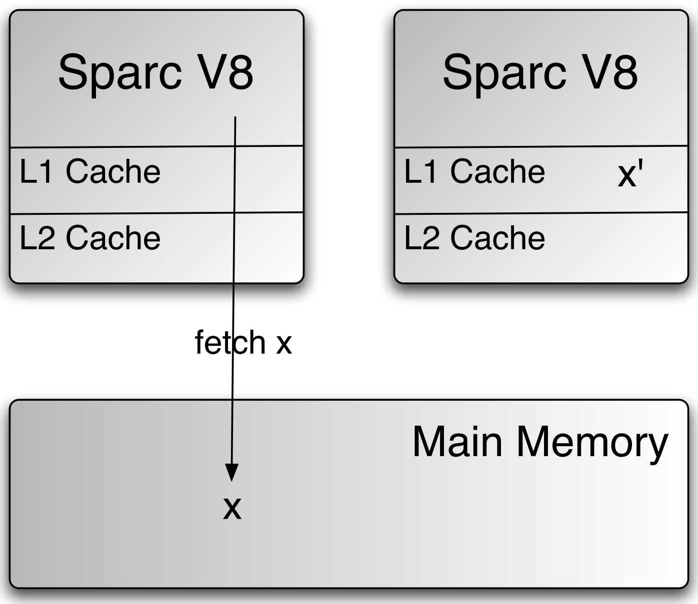
- Formal specification of how memory operations will function
  - ensure consistency in our view of variables
  - enforces strict ordering of memory operations
  - allow or disallow compiler optimizations
- Java Memory Model
- Chip level Memory Model
  - Intel
  - AMD
  - Sparc
  - PowerPC

Hints of the Future

Beginning with J2SE 1.4.1, the Java HotSpot Server VM does not support operations on chips with Sparc V8 architecture

Hardware Plays Catch-up

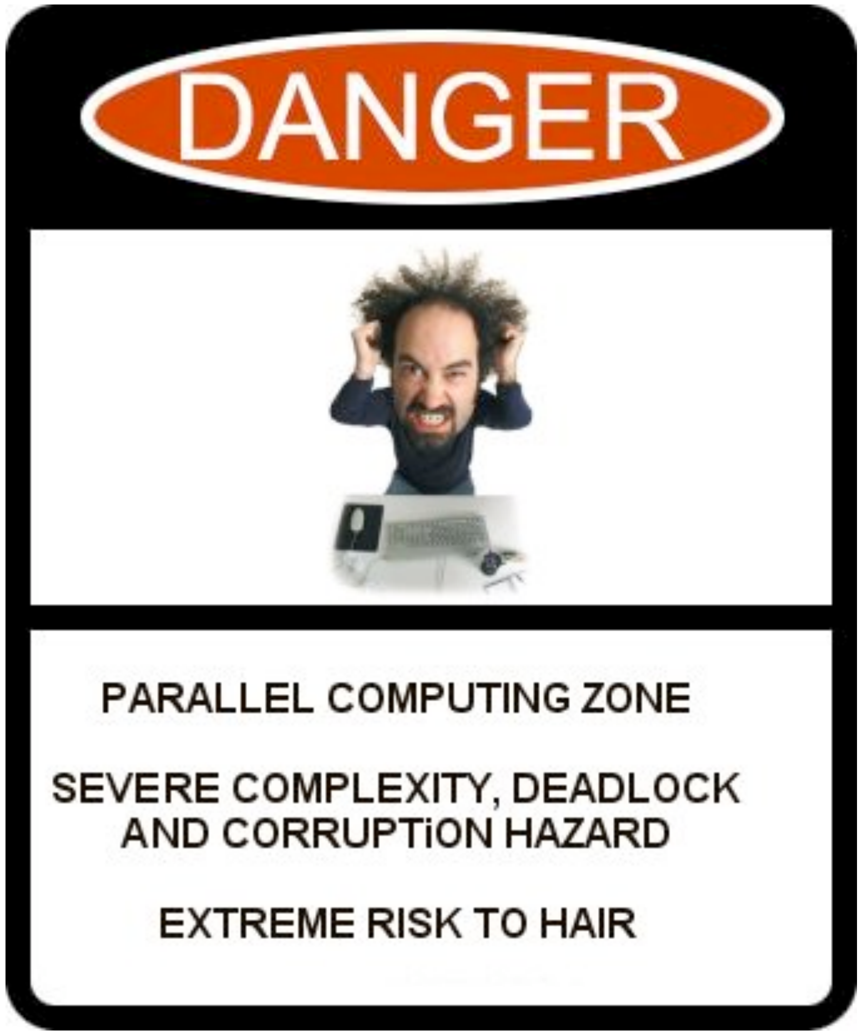
- Sparc V9 contain pseudo instructions to sync L1, L2 cache with main memory on multi-cpu machines



Hardware Acceleration Slows

- Intel announces that focus will shift from clock speed to multi-core/hyperthreading
  - multi-core Xeon processors ship late 2005
- 2007, C|Net reports, Intel and Microsoft state that software needs to heed Moore’s law

New Programming Environment



Kabutz: Law of Sudden Riches

- We no longer have uni-processor systems to hide behind

- Applications suddenly have more CPU
  - bigger problem for older 3rd party libraries

Dangers

- All existing threading bugs start exposing themselves
- We have to worry about
  - deadlock
  - live lock
  - thread stalls
  - race conditions
- Lock contention
  - serialized execution
- Strange behavior in clusters
- .....

Database Vendors React

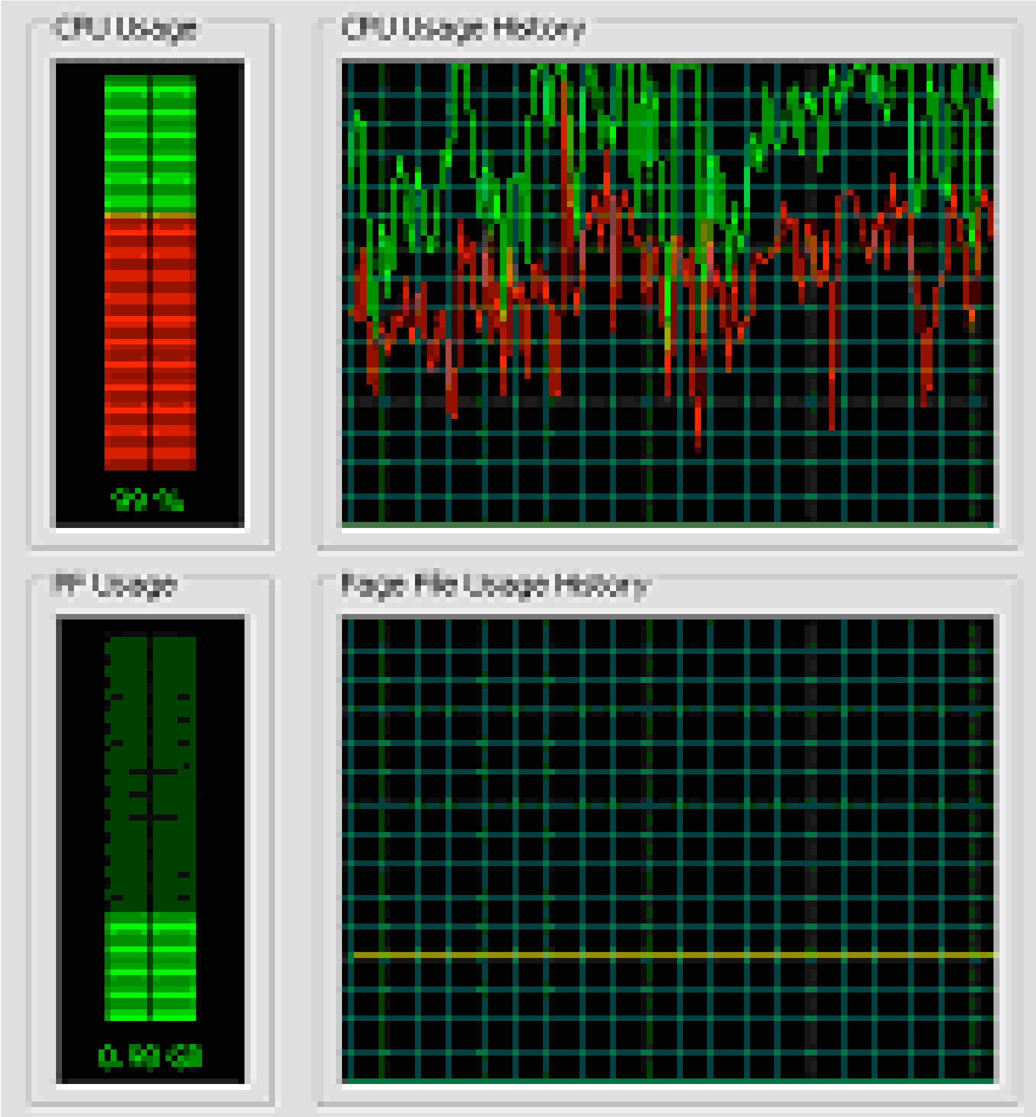
Laptops go Multi-core

Late 2006, ~50% of Java performance course attendees show up with multi-core laptops

Multi-core is a fact of life

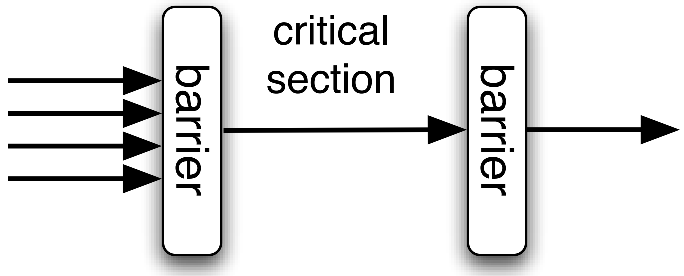
- Developers must deal with concurrency
  - truly threaded applications are more the norm
  - Multi-core puts more pressure on
    - memory
    - I/O resources
    - shared variables
    - Databases?

- Sharing is a big performance issue
  - points of serialization now hurt more than ever


CPU Usage
CPU Usage History
PF Usage
Page File Usage History

Little's Law

- Maths explaining the relationship between locking and throughput



$$\lambda = 1 / \mu$$

$$\mu = 10\text{ms}, \lambda = 100 \text{ tps}$$

$$\mu = 100\text{ms}, \lambda = 10 \text{ tps}$$

Amdahl's Law

- Maths to explain relationship between serialized execution and processor utilization

$$\frac{1}{F + \frac{(1 - F)}{N}}$$

  - F -> 0 number of utilized CPU -> N
  - F -> 1 number of utilized CPU -> 1

Serialized Execution

Amdahl's Law

- Amdahl says; things work until we need to share (or otherwise cooperate)
  - CPU
    - both computational units and L1/L2 cache
  - bus is locked too all other threads while in use
  - memory/Java Heap (Garbage collection)
  - I/O (disk, keyboard, console, files)
  - network
  - data in memory (locks)

Are You Awake?

- L1/L2 caches can thrash

```
for ( int i = 0; i < matrix.length; i++) {
        for ( int j = 0; j < matrix[i].length; j++) {
            matrix[i][j] *= 2;
```

- benches in 430ms

```
for ( int i = 0; i < matrix.length; i++)
        for ( int j = 0; j < matrix[i].length; j++)
            matrix[j][i] *= 2;
```

- benches in 2750ms

Locking is Pessimistic

The glass is half full

Reducing Contention

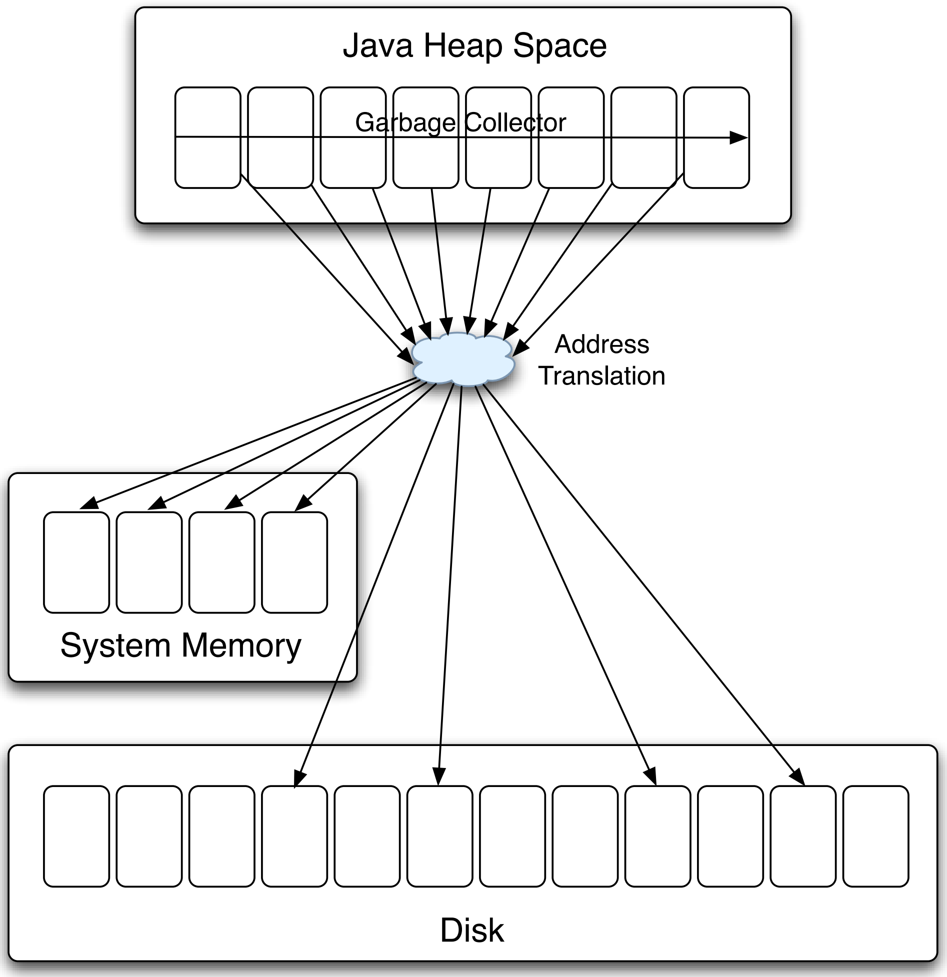
- Share nothing designs
- Pipelined designed
  - messaging and mail boxes
- Minimize transactions
  - duration
  - numbers
- Minimize locking
  - Concurrency package
- Garbage collection
- Hotspot/JIT

Careful use of Databases

Automated Memory Management

- GC is “stop-the-world”
  - GC needs exclusive access to Java heap
  - all application threads must be paused
  - point of serialization in your application
- GC is CPU intensive
  - application pause time tied to clock speed
- An improperly configured Java heap hinders performance
  - Too small => too frequent, risk OOME
  - Too large => long pause times

Law of Leaky Abstractions



Keeping Friends Close

- Large page support now on all platforms
  - keeps related objects on the same page
  - helps avoid TLB misses (expensive to resolve)

  - lock pages into RAM
  - Solaris support is up to 256m (depending on class of machine)
  - Linux/Windows is up to 4m

Garbage Collection

- 1.5 parallel becomes default
  - consider using concurrent
- 1.6 support escape analysis
  - references that remain local can be dealt with more efficiently

More to Come?

- Dominate chip architecture is cache-coherent non-uniform memory access (NUMA)
  - local access is very quick
  - remote access is much slower
  - encourages thread/core affinity
    - mitigates L1/L2 cache coherency issues
    - reduces contention on bus and remote memory

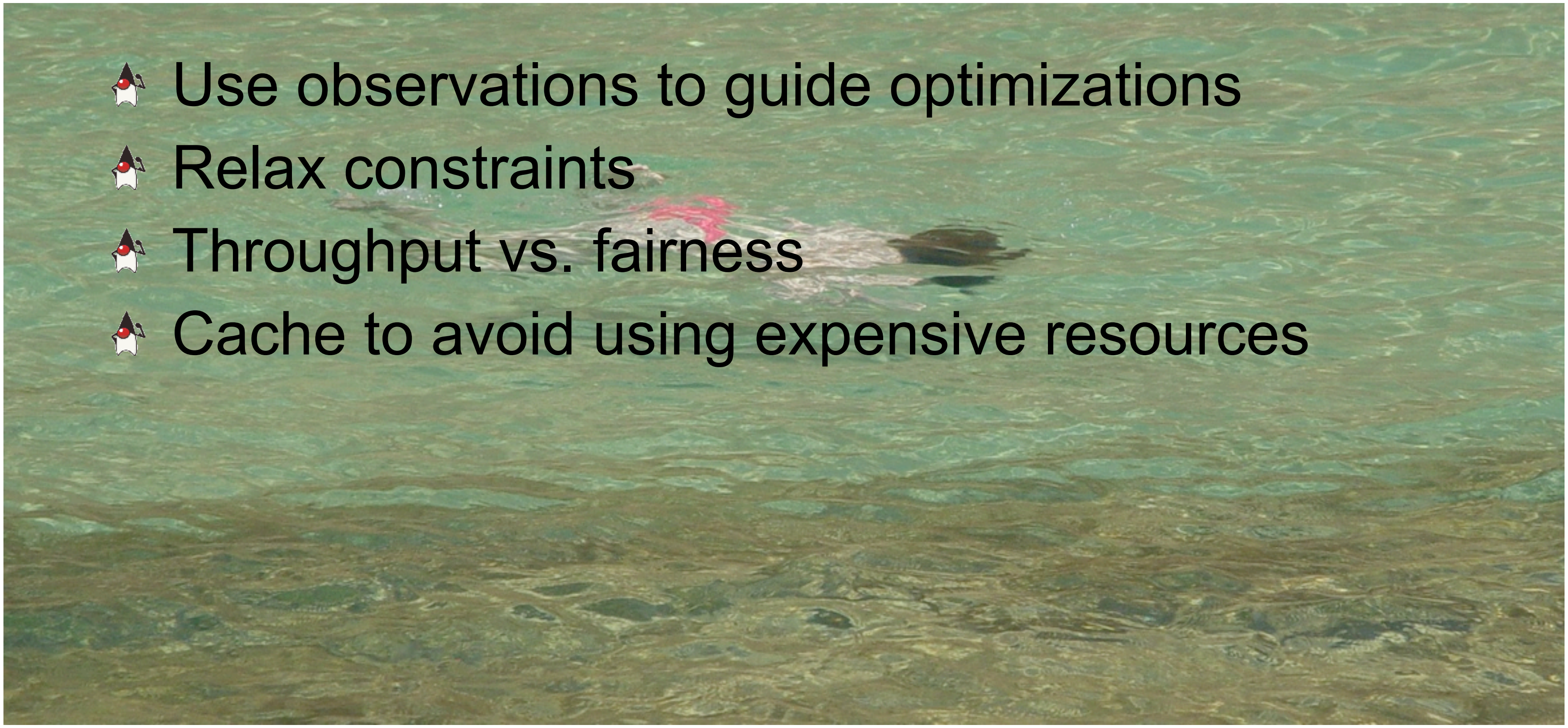
Garbage Collection Improvements

- GC/JVM allocations aware of NUMA
  - localized allocations GC'd faster
  - localized allocations stay remain in CPU cache
  - enabled using -XX:useNUMA (1.6 Update 2)
    - Solaris is simple
    - Windows and Linux require more complex configuration
- http://java.sun.com/javase/technologies/hotspot/largememory.jsp

Locking

- Acquiring a lock is expensive
  - maybe
- Vast majority of locks are not contended
- RDB vendors have known for more than 20 years, locking kills performance
  - what can we learn from RDBs

Optimizations



- Use observations to guide optimizations
- Relax constraints
- Throughput vs. fairness
- Cache to avoid using expensive resources

Hardware to Reduce Contention

- Transactional Memory
  - looks more like an optimistic transaction
  - lock defines “transactional region”
  - allows all threads simultaneous access
  - hardware watches for write-write conflict
  - thread rollback and memory repair

AZUL SYSTEMS Capacity. Plugged in.™

Sun microsystems

Software Improvements

- JSE 5.0 provides a laundry list of improvements aimed at reducing contention
  - atomic variables
  - improved volatile
  - java.util.concurrent (JSR 166)
- semantically richer concurrency
  - Collections with copy on write semantics
  - ConcurrentHashMap
  - ReentrantLock
  - ReadWriteLock

Choice

Which is best?

Monitoring

```
public void run() {
    boolean detected = false;
    while ( running) {
        if ( ( counter < 0) || (counter > 2)) {
            if ( ! detected) {
                System.out.println( "Corrupted " + counter);
                detected = true;
            }
        }
    }
}
```

Mutator

```
private int counter = 0;

Runnable mutator = new Runnable() {
    public void run() {
        long localCount = 0;
        while ( running) {
            counter++;
            counter--;
            localCount++;
        }
        addToTotalCount( localCount);
    }
};
```

Volatile

```
volatile private int counter = 0;

Runnable mutator = new Runnable() {
    public void run() {
        long localCount = 0;
        while ( running) {
            counter++; counter--;  localCount++;
        }
        addToTotalCount( localCount);
    }
};
```

Doubly Synchronized

```
// Instance based counter
private int counter = 0;

// Runnable block
Runnable mutator = new Runnable() {
    public void run() {
        long localCount = 0;
        while ( running) {
            synchronized( this) { counter++;}
            synchronized(this) { counter--; }
            localCount++;
        }
        addToTotalCount( localCount);
    }
};
```

Synchronized

```
// Instance based counter
private int counter = 0;

// Runnable block
Runnable mutator = new Runnable() {
    public void run() {
        long localCount = 0;
        while ( running) {
            synchronized {
                counter++; counter--;
            }
            localCount++;
        }
        addToTotalCount( localCount);
    }};
```

Doubly Reentrant Lock

```
// Instance based counter
private int counter = 0;
private ReentrantLock lock;

// Runnable block
    try {
        lock.lock();
        counter++;
    } finally { lock.unlock(); }
    try {
        lock.lock();
        counter--;
    } finally { lock.unlock(); }
```

Reentrant Lock

```
// Instance based counter
private int counter = 0;
private ReentrantLock lock;

// Runnable block
try {
    lock.lock();
    counter++;
    counter--;
} finally {
    lock.unlock();
}
```

AtomicInteger

```
// Instance based counter
private AtomicInteger counter;

// Runnable block
while ( running) {
    counter.incrementAndGet();
    counter.decrementAndGet();
    localCount++;
}
```

Results

Bench	Counter
Not Thread Safe	750526139
Volatile	333765152
Double Synchronized	28829033
Synchronized	28799357
Double Locked	28966764
Locked	28830148
AtomicInteger	203393689

JDK 1.5.0_10, Intel 3.4 Ghz Hyper-threaded, Window XP

Locking

- java.util.concurrent.locks.Lock
  - Allows you to park threads
    - for a specific amount of time including forever
  - ReentrantLock
    - comes with a fairness setting
  - ReentrantReadWriteLock
    - support multiple readers
    - writer blocks all access

Compare and Set

- Atomic primitive wrappers rely on CAS
  - unsynchronized thead safe type
  - good for atomic operations
- CAS is used to support thread safe lock-free algorithms
  - needs support from the hardware

cas mem_addr, old_value, new_value

Coming Soon?

- Cliff Click's lock-less concurrent HashTable
  - still a research project
  - extremely complex implementation
  - allows race conditions to determine state in the supporting state-machine
  - relies on CAS
- FIFO, LIFO?

JVM Improvements

- JSE 6.0 adds to the list of features that can reduce contention
  - spin-waits (adaptive spinning)
  - lock coarsening
  - lock elision (with escape analysis)
  - biased locking
  - altered notify semantics (less lock jamming)

Does any of this stuff actually work?

How Good is Escape Analysis

- Bench devised by Jeroen Borgers (Xebia)

```
public String concatBuffer( String s1, String s2, String s3) {
    StringBuffer sb = new StringBuffer();
    sb.concat( s1);
    sb.concat( s2);
    sb.concat( s3);
    return sb.toString();
}

public String concatBuilder( String s1, String s2, String s3) {
    StringBuilder sb = new StringBuilder();
    sb.concat( s1);
    sb.concat( s2);
    sb.concat( s3);
    return sb.toString();
}
```

Lock Overhead

Benchmark	StringBuffer	StringBuilder	%Overhead
Baseline	7896	2760	186%
Escape Analysis	7875	2756	185%
Elimination	4068	2739	48%
Biased	5489	2843	93%
Escape, Elimination	4078	2813	45%
Escape, Biased	5500	2849	93%
Elimination, Biased	4718	2812	68%
Escape, Elimination, Biased	4740	2828	68%

The Future is Clear

- Processors will contain
  - many more cores
- Memory will be segmented
  - local segments as part of a global space
- Applications will continue need to be hardware aware
- Languages improvements
  - could closures offer better expression of parallelism?
  - Totally new language?

The Future is Clear

Operating systems and hardware are being optimized to better support virtual machines

Contention is Under Siege

pessimistic ⟸ Improving Concurrency ⟹ optimistic

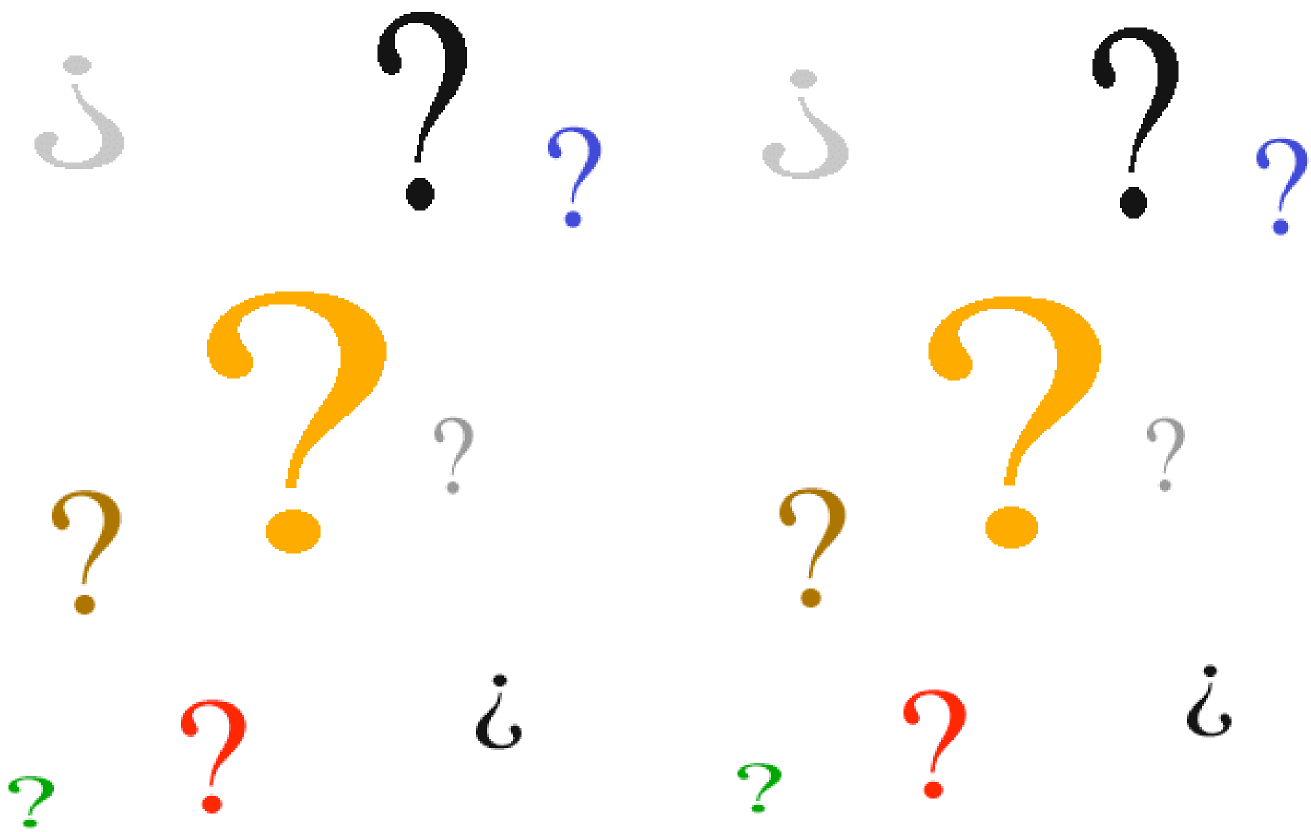

Are you prepared to deliver twice
as much concurrency in the next
18 months?

Thanks for your attention

www.parisjug.org

Xebia
aneo the other solution
OceaneConsulting Prestation Java
BK Consulting

Concurrency & Performance



Sponsors

Xebia

aneo the other solution

OceaneConsulting Prestation Java

BK Consulting