

# Titanium: Parallel Java

<http://titanium.cs.berkeley.edu/>

# “Hello World”

```
class HelloWorld {  
    public static void main( String[] args ) {  
        System.out.println(  
            "Hello from " + Ti.thisProc() +  
            " of " + Ti.numProcs() + "!");  
    }  
}
```

# Matrix Multiplication

```
public static void matMul( double [2d] a,  
                           double [2d] b,  
                           double [2d] c ) {  
    foreach( ij in c.domain() ) {  
        double [1d] aRowi = a.slice(1, ij[1]);  
        double [1d] bColj = b.slice(2, ij[2]);  
        foreach( k in aRowi.domain() ) {  
            c[ ij ] += aRowi[ k ] * bColj[ k ];  
        }  
    }  
}
```

# Contents

- The Good
- The Bad
- The Ugly

# The Good

- “Single”, Synchronization, and Collectives
- Arrays and Domains
- Unordered Iteration
- Immutable Classes

# “Single”, Synchronization, and Collectives

- Titanium processes must synchronize at the same textual point in the program:
  - Legal barrier example:

```
{ do_stuff(1); Ti.barrier(); do_stuff }
```
  - Illegal barrier example:

```
{ if(Ti.thisProc()==0) Ti.barrier();  
  else Ti.barrier(); }
```
- This is enforced by the concept of a “single” value, a value that is guaranteed to be the same on all processes.

# “Single”, Synchronization, and Collectives

- Variables and functions can be declared single, which tells the compiler that they generate the same value across all processes.
  - Only single values can be assigned to variables declared single
  - Single functions can only call other single functions, and access only single variables
- The compiler can infer whether or not a function or value is single

# “Single”, Synchronization, and Collectives

- Example:

```
int master, masterchoice; ...  
choice = broadcast masterchoice from master;
```

- Broadcast has implied synchronization:

- All threads must wait for the source thread to reach the broadcast, and all will receive the same value
- However, there are no other guarantees made

# Arrays and Domains

- Two types of Arrays in Titanium
  - Java arrays
    - Work exactly the same as arrays in Java
    - “Multi-dimensional” arrays are really single-dimensional arrays of references to arrays
  - Titanium arrays
    - Support true multi-dimensional arrays
    - Support for collective exchange, reduce, and scan (prefix reduce)
    - Efficient support for getting subsets of domains: slicing, translating, union, difference, intersection, shrink, border etc.

# Arrays and Domains

- Titanium Arrays
  - Indexed by “points”
    - n-tuples of ints
  - Set of all points is the domain of the array
    - Arrays must have a rectangular domain...
    - ...but can iterate over non-rectangular sub-domains

# Matrix Multiplication

```
public static void matMul( double [2d] a,  
                           double [2d] b,  
                           double [2d] c ) {  
    foreach( ij in c.domain() ) {  
        double [1d] aRowi = a.slice(1, ij[1]);  
        double [1d] bColj = b.slice(2, ij[2]);  
        foreach( k in aRowi.domain() ) {  
            c[ ij ] += aRowi[ k ] * bColj[ k ];  
        }  
    }  
}
```

# Unordered Iteration

- Add a `foreach` construct:

```
foreach (p in r) { ... A[p] ... }
```

- Iterations can occur in any order
- Simple indexing
  - `p` is a point
  - `r` is a domain
- **NOT** a parallelism construct

# Immutable Classes

- Optimized for small classes
  - Pass-by-value
  - Stored on stack
- Fields are implicitly `final`, and can only be assigned in the constructor
- Cannot inherit from other classes
- Must have a 0-argument constructor

# The Bad

- Operator Overloading
  - Bad Idea
  - Really Bad Idea
  - Seriously, Java didn't do it for a **GOOD** reason

# The Ugly

- Explicit Memory Management
  - Default behavior is implicit memory management with a garbage collector, same as Java.
  - Programmer can allocate memory “regions”, allocate objects out of these regions, then explicitly free all objects in the regions.
  - Makes the compiler writer’s job easier and allows programmers to “tune” the code for better performance.

# Questions?