# X10 --- a New Programming Model for Productive Scalable Parallel Programming

**PMUA Workshop**

**June 21, 2005**

**Vivek Sarkar**

**(vsarkar@us.ibm.com)**

**IBM T.J. Watson Research Center**

This work has been supported in part by the
Defense Advanced Research Projects Agency (DARPA)
under contract No. NBCH30390004.

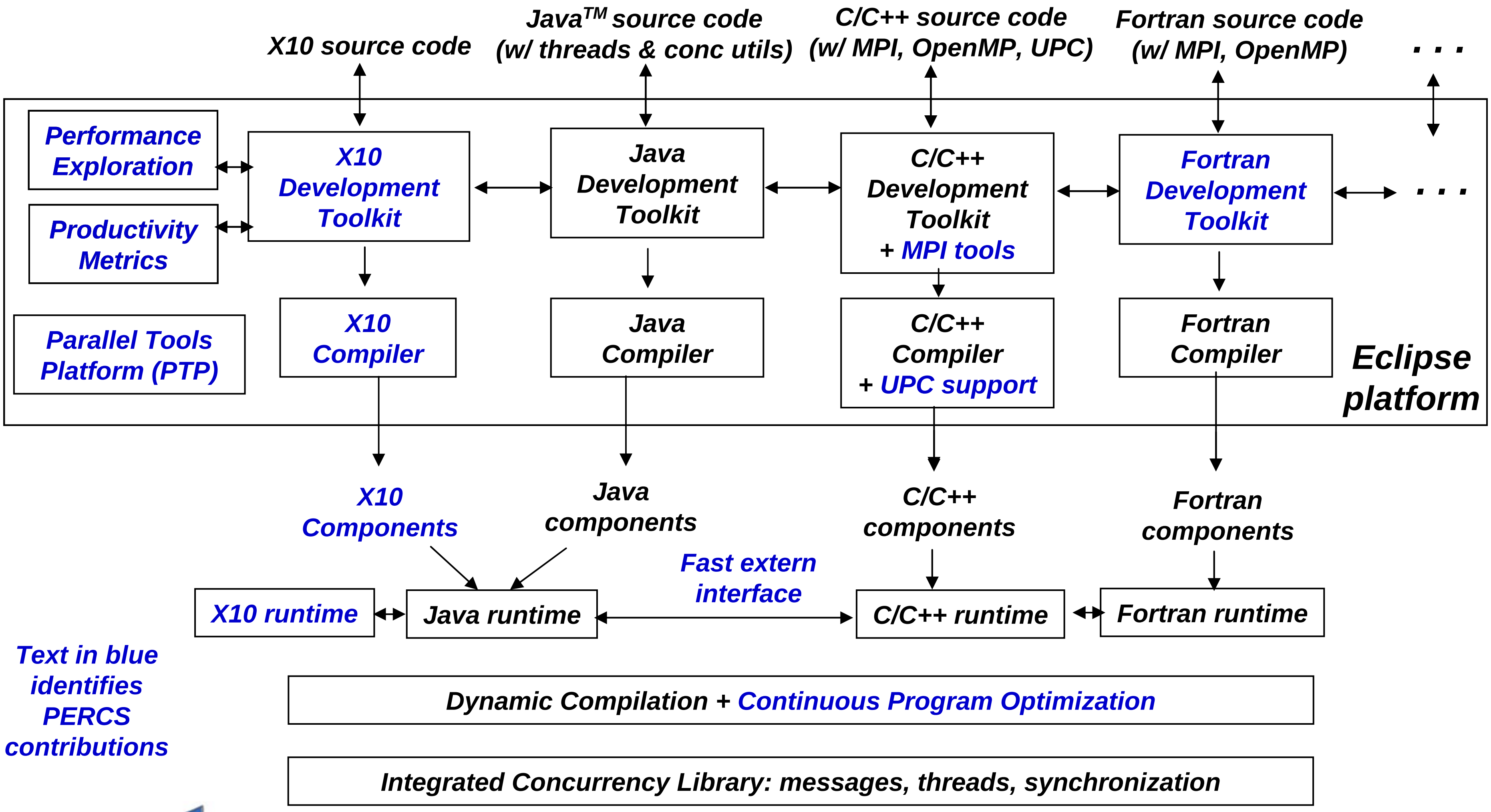
# Acknowledgments

- **X10 core team**
  - Philippe Charles
  - Chris Donawa
  - Kemal Ebcioglu
  - Christian Grothoff
  - Allan Kielstra
  - Douglas Lovell
  - Maged Michael
  - Christoph von Praun
  - Vijay Saraswat
  - Vivek Sarkar
- **PERCS Tools**
  - Marina Biberstein
  - Julian Dolby
  - Robert Fuhrer
  - Emmanuel Geay
  - Matthias Hauswirth
  - Peter Sweeney
  - Beth Tibbitts
  - Frank Tip
  - Mandana Vaziri
- **PERCS Productivity**
  - Catalina Danis
  - Christine Halverson
- **PERCS University partners (Prog Model & Tools)**
  - DePaul University (X10)
  - MIT (StreamIt)
  - Purdue University (X10)
  - UC Berkeley (StreamBit)
  - U. Delaware (Atomic sections)
  - U. Illinois (Fortran plug-in)
  - UT Austin (Componentization)
  - Vanderbilt University (Productivity metrics)
- **PERCS Prog Model & Tools Team Leads**
  - Kemal Ebcioglu, Vivek Sarkar
- **PERCS Principal Investigator**
  - Mootaz Elnozahy
- **PERCS Project Manager**
  - Vickie Robinson

# Scalability Challenges for Scientific Applications

- Applications need to harness multiple heterogeneous levels of parallelism and locality
  - Cluster, SMP, multi-cores, SPU’s, SIMD, TLP, ILP
- Domain decomposition is already running into scaling limits at Tera-scale
- Load balance efficiency (Tavg/Tmax) is becoming a key limitation to scalability
- Synchronous and bulk-synchronous programming models further limit scalability …
  - Frequent use of global barriers and global communications
- … as do programming models based on message passing and locks
  - Frequent use of blocking operations
- Applications are getting increasingly complicated in their use of sparse, irregular, and adaptive techniques
- Expertise Gap: domain scientists vs. system experts

# PERCS Programming Model, Tools and Compilers: Overall Architecture



X10 source code

Java$^{TM}$ source code (w/ threads & conc utils)

C/C++ source code (w/ MPI, OpenMP, UPC)

Fortran source code (w/ MPI, OpenMP)

. . .

Performance Exploration

Productivity Metrics

Parallel Tools Platform (PTP)

X10 Development Toolkit

Java Development Toolkit

C/C++ Development Toolkit + MPI tools

Fortran Development Toolkit

. . .

X10 Compiler

Java Compiler

C/C++ Compiler + UPC support

Fortran Compiler

Eclipse platform

X10 Components

Java components

C/C++ components

Fortran components

Fast extern interface

X10 runtime

Java runtime

C/C++ runtime

Fortran runtime

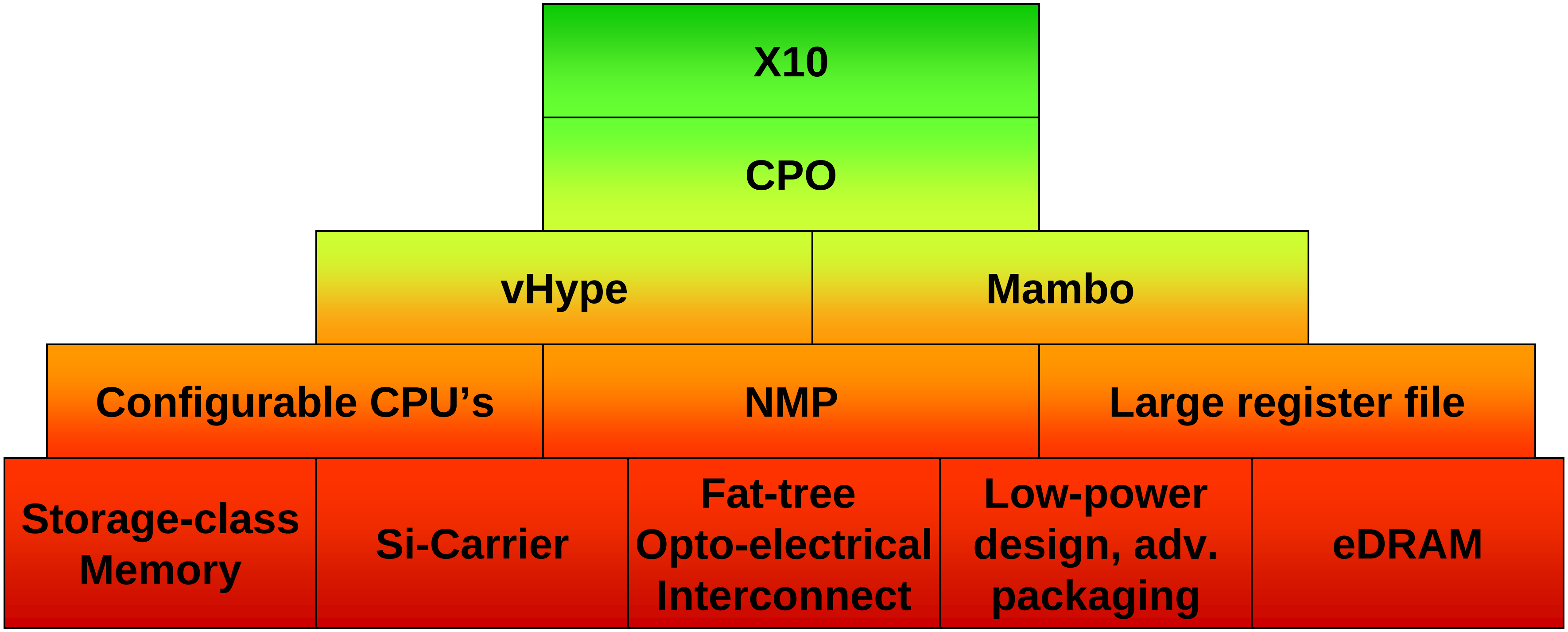
Dynamic Compilation + Continuous Program Optimization

Integrated Concurrency Library: messages, threads, synchronization

Text in blue identifies PERCS contributions

PERCS = Productive Easy-to-use Reliable Computing Systems

# PERCS Technology Bets



X10

CPO

vHype

Mambo

Configurable CPU’s

NMP

Large register file

Storage-class Memory

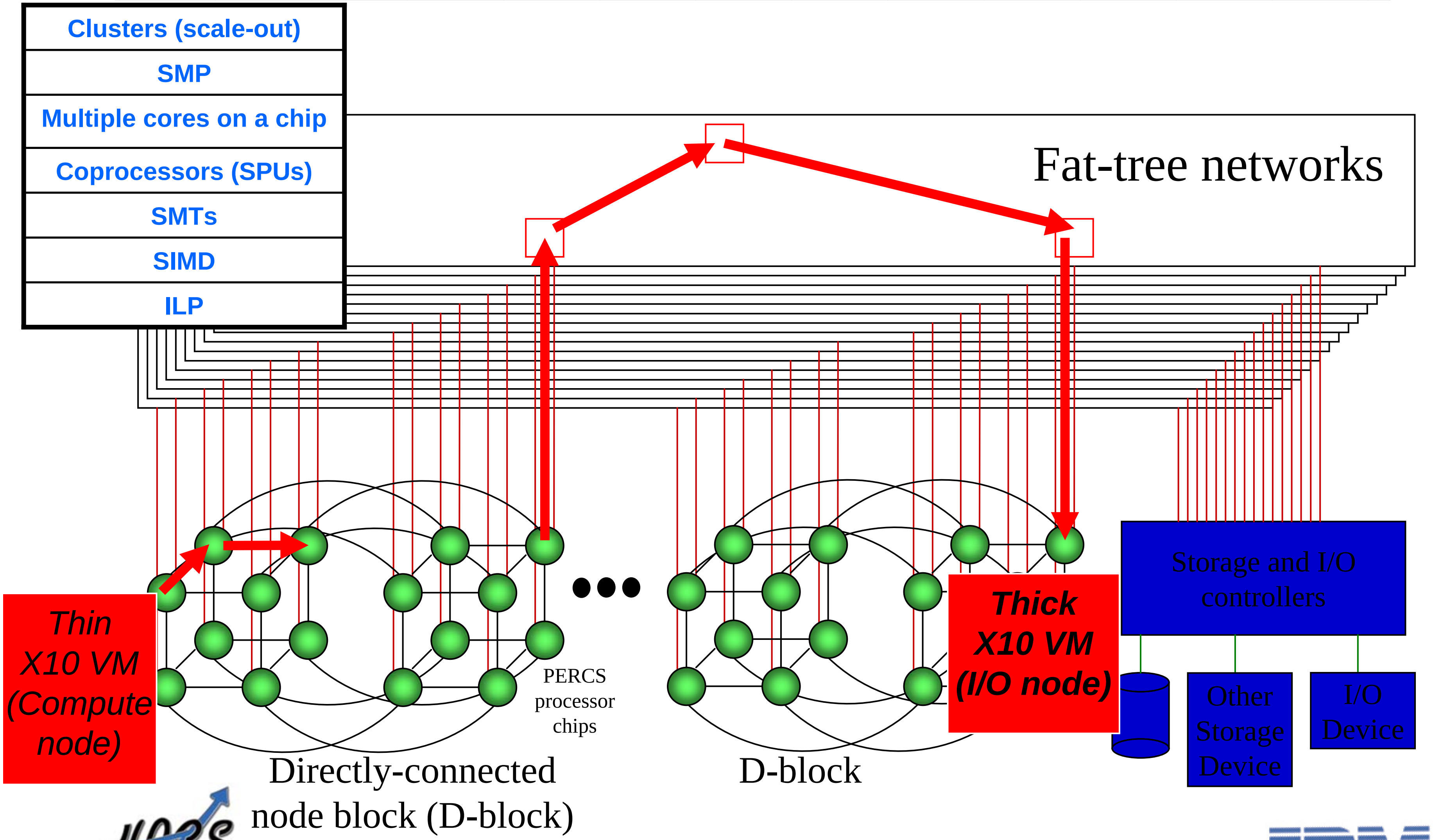
Si-Carrier

Fat-tree Opto-electrical Interconnect

Low-power design, adv. packaging

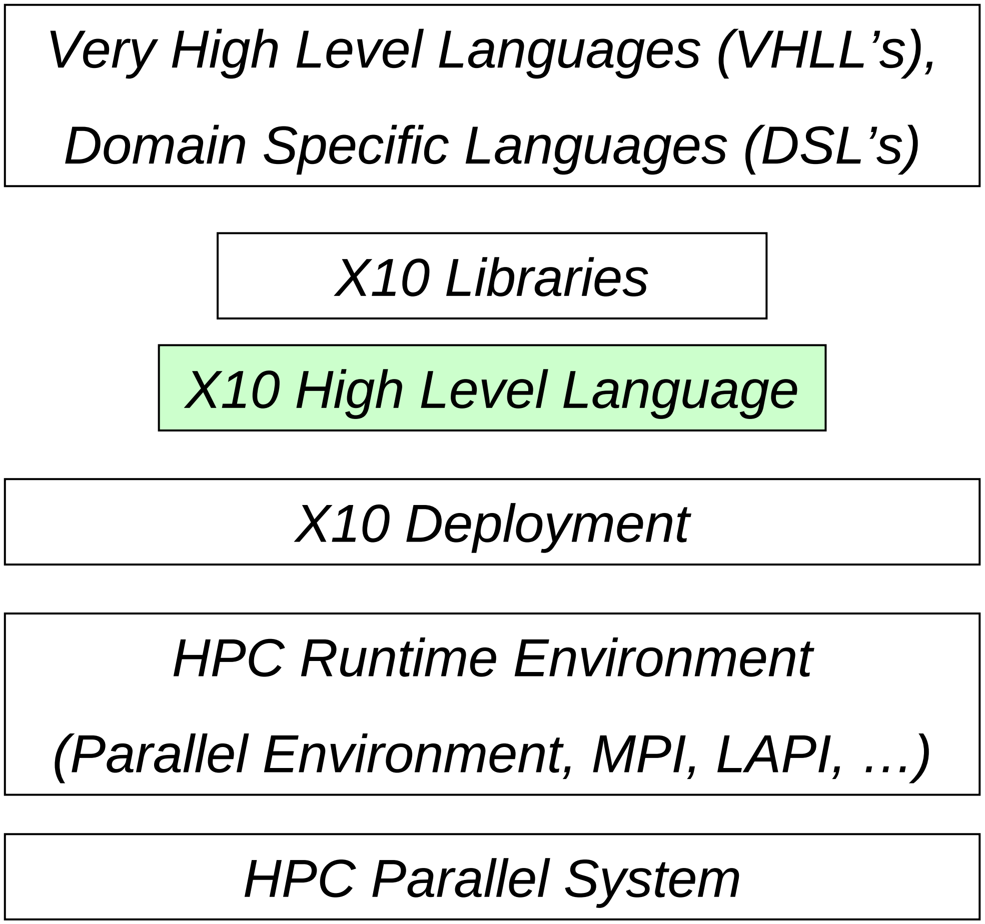
eDRAM

# Future X10 Environment:
# X10 Deployment on a PERCS HPC system



- Clusters (scale-out)
- SMP
- Multiple cores on a chip
- Coprocessors (SPUs)
- SMTs
- SIMD
- ILP

# Future X10 Environment



Very High Level Languages (VHLL's),

Domain Specific Languages (DSL's)

*Implicit parallelism,*

*Implicit data distributions*

X10 Libraries

X10 High Level Language

*X10 places --- abstraction of explicit control & data distribution*

X10 Deployment

*Mapping of places to nodes in HPC Parallel Environment*

HPC Runtime Environment

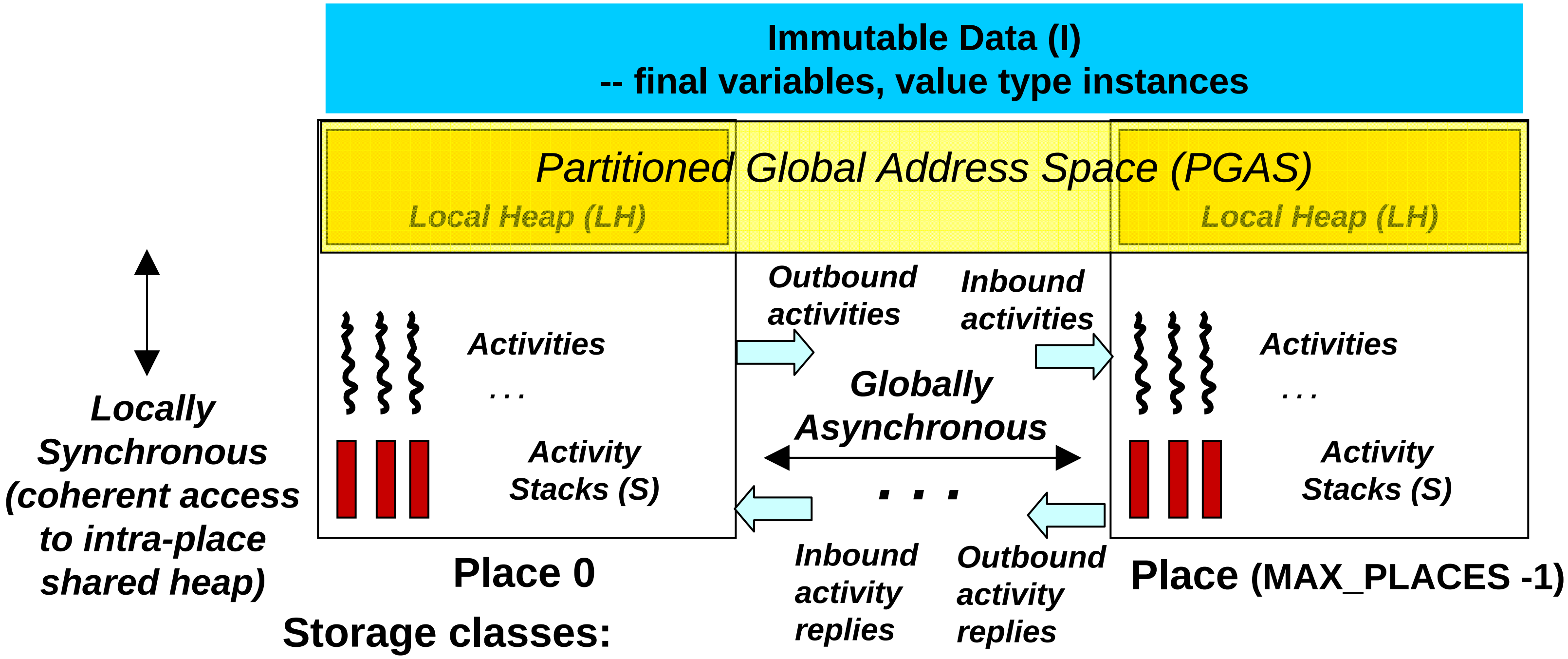
(Parallel Environment, MPI, LAPI, …)

*Primitive constructs for parallelism, communication, and synchronization*

HPC Parallel System

*Target system for parallel application*

# Overview of X10 Programming Model



Storage classes:

- Immutable Data (I)
- PGAS
  - Local Heap (LH)
  - Remote Heap (RH)
- Activity Stacks (S)

- *Place* = collection of activities & objects
  - Activities and data objects do not move after being created (but place-processor mapping can be changed)
- *Scalar object, O* -- maps to a single place specified by O.location
- *Array object, A* – may be local to a place or distributed across multiple places, as specified by A.distribution

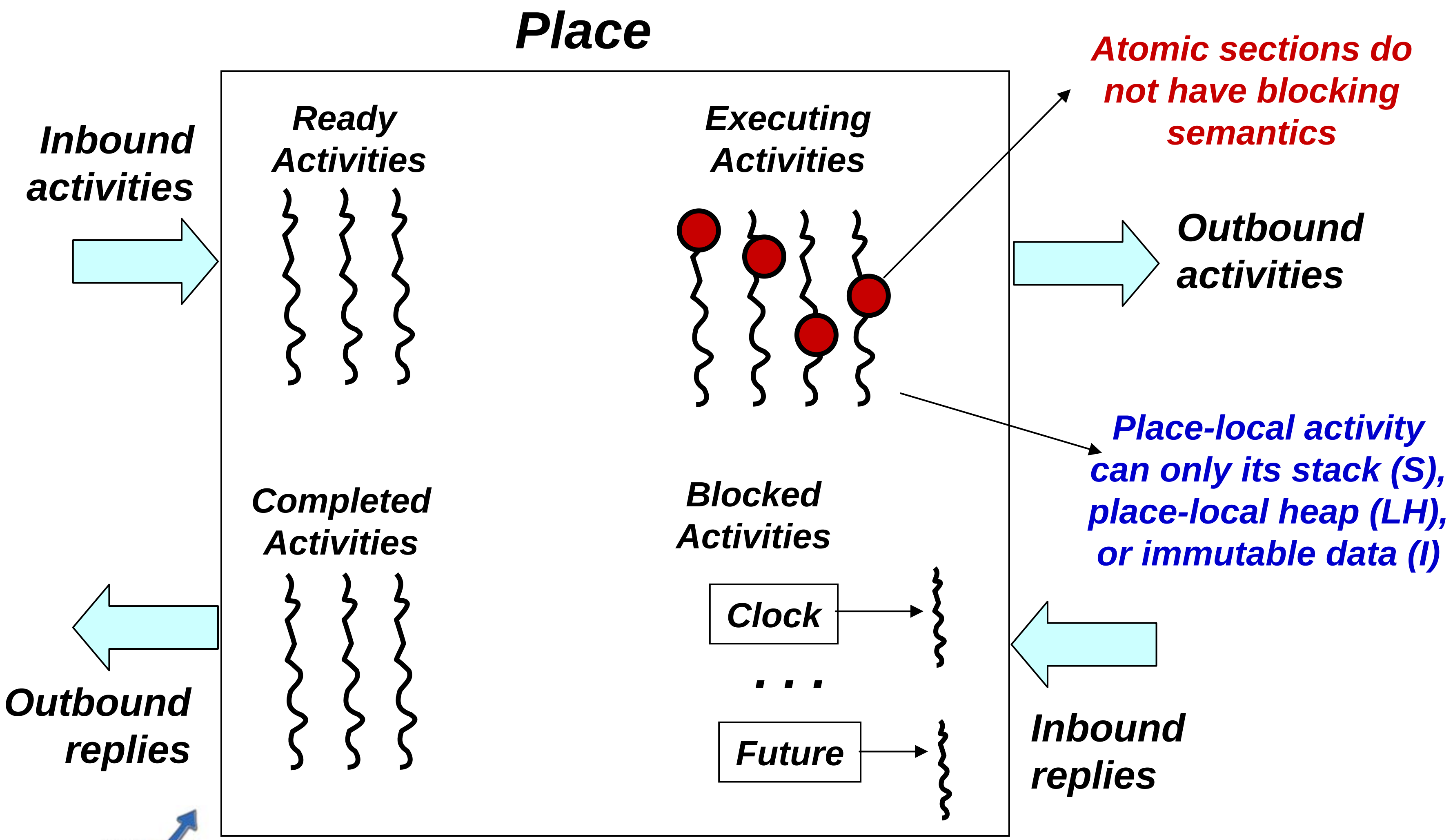
# Locality Rule

- *Any access to a mutable (shared heap) datum must be performed by an activity located at the place as the datum*
- No data sharing permitted for stack locations
  - Not even between parent activity's stack and child activity's stack
- Local-to-remote (LH → RH) and remote-to-local (RH → LH) heap references are freely permitted
- However, *direct access* via a remote heap reference is not permitted!
- Inter-place data accesses can only be performed by creating remote activities …
  - … with weaker ordering guarantees than intra-place data accesses
- The locality rule is checked at runtime by default
  - BadPlaceException is thrown on an access to a remote reference
  - Locality checks can be optimized (analogous to optimization of bounds checks and type checks)

# Memory Model

- X10 focus is on data-race-free applications
- Programmer uses atomic / finish / force / clock operations to avoid data races
  - X10 programming environment also includes data race detection tool
- No data races can occur on data that is activity-local or immutable
- Globally Asynchronous …
  - Weak ordering of inter-place activities
- … and Local Synchronous (GALS)
  - Guaranteed coherence for local heap --- all writes to same shared location are observed in same order by all activities in the same place

# Activity Execution within a Place



Place

Inbound activities

Ready Activities

Executing Activities

Atomic sections do not have blocking semantics

Outbound activities

Place-local activity can only its stack (S), place-local heap (LH), or immutable data (I)

Completed Activities

Blocked Activities

Clock

. . .

Future

Outbound replies

Inbound replies

# X10 vs. Java$^{TM}$ languages

- X10 is an extended subset of the Java language
  - Base language = Java 1.4 language
    - Java 5 features (generics, metadata, etc.) are currently not supported in X10
  - Notable features removed from Java language
    - Concurrency --- threads, synchronized, etc.
    - Java arrays – replaced by X10 arrays
  - Notable features added to Java language
    - Concurrency – async, finish, atomic, future, force, foreach, ateach, clocks
    - Distribution --- points, distributions
    - X10 arrays --- multidimensional distributed arrays, array reductions, array initializers,
    - Serial constructs --- nullable, const, extern, value types
- X10 supports both OO and non-OO programming paradigms

# Calling foreign functions from X10 programs

- Java methods
  - Can be called directly from X10 programs
    - Makes ecosystem of Java libraries automatically available to X10 programmer
  - Java class will be loaded automatically as part of X10 program execution
- C functions
  - Need to use extern declaration in X10, and perform a System.loadLibrary() call

# X10 v0.409 Cheat Sheet

*Stm:*

async [ ( *Place* ) ] [clocked *ClockList* ] *Stm*

finish *Stm*

atomic *Stm*

when ( *SimpleExpr* ) *Stm*

next; *c.resume()* *c.drop()*

for( *point p : Region* ) *Stm*

foreach ( *point p : Region* ) *Stm*

ateach ( *point p : Distribution* ) *Stm*

*ClassModifier : Kind*

*Kind :*
value | reference

*DataType:*

*ClassName* | *InterfaceName* | *ArrayType*

nullable *DataType*

future *DataType*

*Expr:*
*ArrayExpr*
*FutureExpr . force()*
*here*

*MethodModifier:* atomic

*x10.lang has the following classes (among others)*

point, range, region, dist, clock, array

Some of these are supported by special syntax.

# X10 v0.409 Cheat Sheet: Array support

***ArrayExpr:***

| | |
|---|---|
| new ArrayType ( Formal ) { Stm } | |
| *Distribution Expr* | -- Lifting |
| *ArrayExpr* [ *Region* ] | -- Section |
| *ArrayExpr* \| *Distribution* | -- Restriction |
| *ArrayExpr* \|\| *ArrayExpr* | -- Union |
| *ArrayExpr*.overlay(*ArrayExpr*) | -- Update |
| *ArrayExpr*. scan( *[fun [, ArgList]* ) | |
| *ArrayExpr*. reduce( *[fun [, ArgList]* ) | |
| *ArrayExpr*.lift( *[fun [, ArgList]* ) | |

***ArrayType:***

*Type [Kind]* [ ]

*Type [Kind]* [ region(N) ]

*Type [Kind]* [ *Region* ]

*Type [Kind]* [ *Distribution* ]

***Region:***

| | |
|---|---|
| *Expr : Expr* | -- 1-D region |
| [ *Range, …, Range* ] | -- Multidimensional Region |
| *Region* && *Region* | -- Intersection |
| *Region* \|\| *Region* | -- Union |
| *Region* – *Region* | -- Set difference |
| *BuiltinRegion* | |

***Distribution:***

| | |
|---|---|
| *Region -> Place* | -- Constant Distribution |
| *Distribution* \| *Place* | -- Restriction |
| *Distribution* \| *Region* | -- Restriction |
| *Distribution* \|\| *Distribution* | -- Union |
| *Distribution* – *Distribution* | -- Set difference |
| *Distribution*.overlay ( *Distribution* ) | |
| *BuiltinDistribution* | |

***Language supports type safety, memory safety, place safety, clock safety***

# RandomAccess Example in X10

```
public boolean run() {
 distribution D = distribution.factory.block(TABLE_SIZE);
 long[.] table = new long[D] (point [i]) { return i; }
 long[.] RanStarts = new long[distribution.factory.unique()]
    (point [i]) { return starts(i);};
 long[.] SmallTable = new long value[TABLE_SIZE]
    (point [i]) {return i*S_TABLE_INIT;};
 finish ateach (point [i] : RanStarts ) {
  long ran = nextRandom(RanStarts[i]);
  for (int count: 1:N_UPDATES_PER_PLACE) {
   int J = f(ran);
   long K = SmallTable[g(ran)];
   async atomic table[J] ^= K;
   ran = nextRandom(ran);
  }
 }
return table.sum() == EXPECTED_RESULT;
}
```

**Allocate and initialize table as a block-distributed array.**

**Allocate and initialize RanStarts with one random number seed for each place.**

**Allocate a small immutable table that can be copied to all places.**

**Everywhere in parallel, repeatedly generate random table indices and atomically read/modify/write table element.**

# ArrayCopy example: example of high-level optimizations of async activities

Version 1 (orginal):

```
<value T, D, E>  public static void
   arrayCopy( T[D] a, T[E] b) {
      // Spawn an activity for each index to
      // fetch and copy the value
      ateach (i : D.region)
          a[i] = async b[i];
}
```

Version 2 (optimized):

```
<value T, D, E>  public static void
   arrayCopy( T[D] a, T[E] b) {
     // Spawn one activity per place
     ateach ( D.places )
        for ( j : D | here )
           a[i] = async b[i];
}
```

Version 3 (further optimized):

```
<value T, D, E>  public static void
   arrayCopy( T[D] a, T[E] b) {
   // Spawn one activity per D-place and one
   // future per place p to which E maps an
   // index in (D | here).
     ateach ( D.places ) {
        region LocalD = (D | here).region;
        ateach ( p : E[LocalD] ) {
           region RemoteE = (E | p).region;
           region Common =
                          LocalD && RemoteE;
           a[Common] = async  b[Common];
        }
     }
}
```

# Relating optimizations for past programming paradigms to X10 optimizations

| Programming paradigm | Activities | Storage classes | Important optimizations |
|---|---|---|---|
| Message-passing e.g., MPI | Single activity per place | Place local | Message aggregation, optimization of barriers & reductions |
| Data parallel e.g., HPF | Single global program | Partitioned global | SPMDization, synchronization & communication optimizations |
| PGAS e.g., Titanium, UPC | Single activity per place | Partitioned global, place local | Localization, SPMDization, synchronization & communication optimizations |
| DSM e.g., TreadMarks | Multiple | Partitioned global, activity local | Data layout optimizations, page locality optimizations |
| NUMA | Single activity per place | Partitioned global, activity local | Data distribution, synchronization & communication optimizations |
| Co-processor e.g., STI Cell | Single activity per place | Partitioned-global, place-local | SIMDization, data communication, & synchronization optimizations |
| Futures / active messages | Multiple | Place-local, activity local | Message aggregation, synchronization optimization |
| Full X10 | Multiple activities in multiple places | Partitioned-global, place-local, activity-local | All of the above |

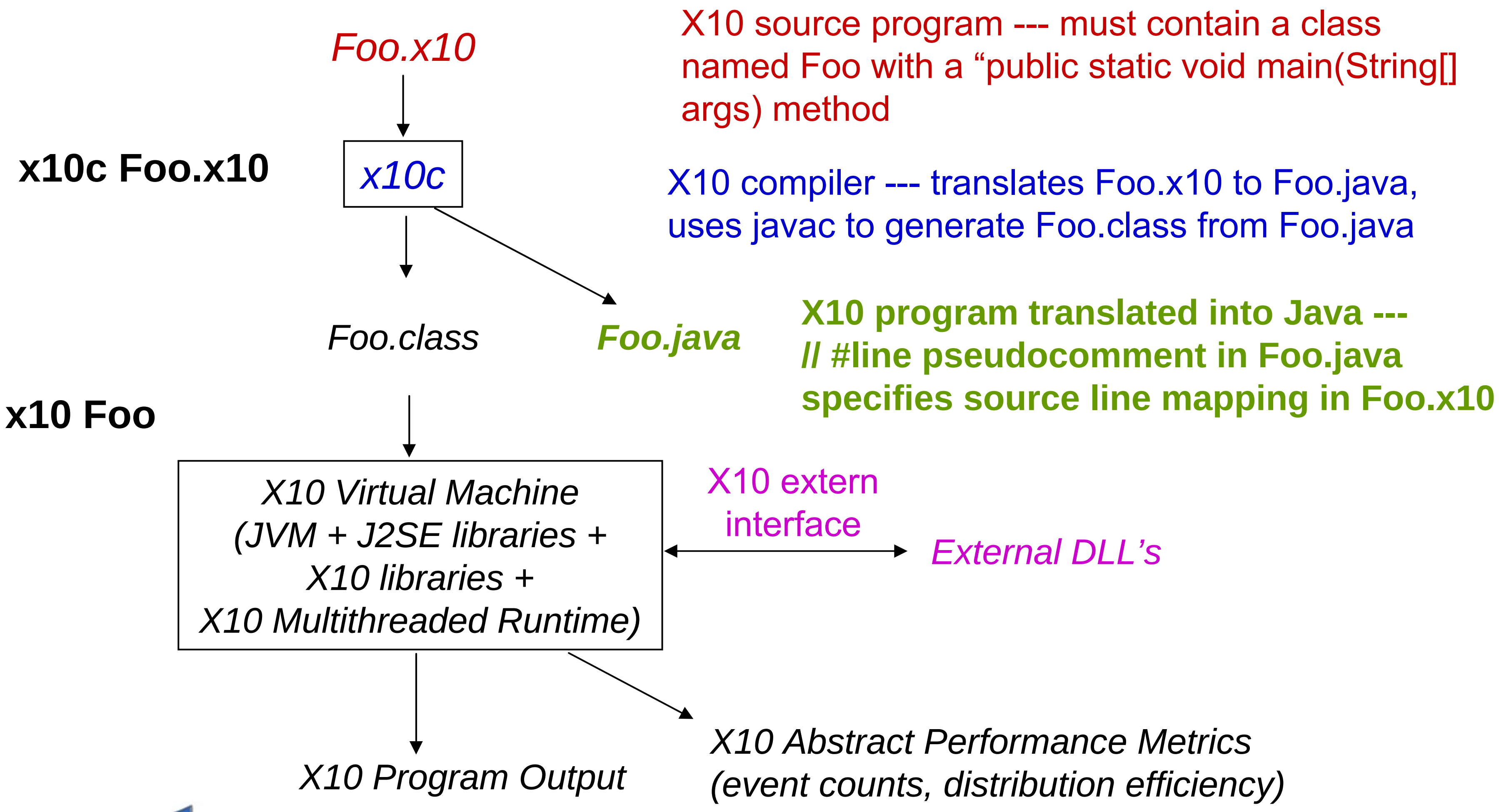
# Support for irregular computations --- generalize distributed arrays to distributed collections (work in progress)

- **Distributed Collections**
  - Map collection elements to places
  - Collection<D,E> identifies a collection with distribution D and element type E
- **Parallel iterators**
  - foreach (e : C) { … }
  - ateach ( e : C ) { … here … }
- **Sequential iterator**
  - for (e : C)

# X10 status and schedule

- 6/2003 PERCS programming model concept (end of PERCS Phase 1)
- 7/2004 Start of PERCS Phase 2
- 2/2004 Kickoff of X10 as concrete embodiment of PERCS programming model as a new language
- 7/2004 First draft of X10 language specification
- 2/2005 First X10 implementation -- unoptimized single-VM prototype
  - » Emulates distributed parallelism in a single process
- 5/2005 X10 productivity study at Pittsburgh Supercomputing Center
- 7/2005 Results from X10 application & productivity studies
- 2H2005 Revise language based on application & productivity feedback
- 1/2006 Second X10 implementation – optimized multi-VM prototype
- 6/2006 Open source release of X10 reference implementation
- 7/2006 Phase 3 scheduled to start ….

# Current X10 Environment: Unoptimized Single-VM Implementation



*Foo.x10*

X10 source program --- must contain a class named Foo with a “public static void main(String[] args) method

**x10c Foo.x10**

*x10c*

X10 compiler --- translates Foo.x10 to Foo.java, uses javac to generate Foo.class from Foo.java

*Foo.class*

***Foo.java***

**X10 program translated into Java --- // #line pseudocomment in Foo.java specifies source line mapping in Foo.x10**

**x10 Foo**

*X10 Virtual Machine (JVM + J2SE libraries + X10 libraries + X10 Multithreaded Runtime)*

X10 extern interface

*External DLL’s*

*X10 Program Output*

*X10 Abstract Performance Metrics (event counts, distribution efficiency)*

***Caveat: this is a prototype implementation with many limitations.***

# Parallel Programming Pitfalls: Deadlock

- X10 guarantee
  - Any program written with async, finish, atomic, foreach, ateach, and clock parallel constructs will never deadlock
- Unrestricted use of future and force may lead to deadlock:
  - f1 = future { a1() } ;
  - f2 = future { a2() };
  - int a1() { … f2.force(); … }
  - Int a2() { … f1.force(); … }
- Restricted use of future and force in X10 can preserve guaranteed freedom from deadlocks
  - Sufficient condition #1: ensure that activity that creates the future also performs the force() operation
  - Sufficient condition #2: . . .

# Parallel Programming Pitfalls: Data Races

- A data race occurs when two (or more) threads/activities can access the same shared location in parallel such that one of the accesses is a write operation
  - Can also occur with asynchronous activities e.g., DMA, I/O
- Example:
  - Thread 0: a++ ; b-- ;
  - Thread 1: a++ ; b--;
  - Data race can violate invariant that (a+b) is constant
  - Data race may also prevent multiple increments from being combined correctly
- X10 guidelines for avoiding data races
  - Use atomic methods and blocks without worrying about deadlock
  - Declare data to be immutable (i.e., final or value type instance) or thread-local whenever possible

# Scalability Challenges for Scientific Applications: Summary of PERCS solutions

- Applications need to harness multiple heterogeneous levels of parallelism and locality
  - ➔ Write portable code in X10 using places, async's, and other language constructs
- Domain decomposition is already running into scaling limits at Tera-scale
  - ➔ X10 integrates cluster-level and thread-level parallelism with first-class language support
- Load balance efficiency is becoming a key limitation to scalability
  - ➔Use PERCS CPO to optimize X10 distributions and deployment
- Synchronous and bulk-synchronous programming models further limit scalability …
  - ➔ X10 programs are asynchronous by default; finish and clocks are more restricitive in scope than global barriers
- … as do programming models based on message passing and locks
  - ➔ X10 offers easy-to-use non-blocking constructs (async, atomic)
- Applications are getting increasingly complicated in their use of sparse, irregular, and adaptive techniques
  - ➔ X10 regions and distributions should be well suited to irregular applications --- adaptive techniques are well suited to PERCS CPO
- Expertise Gap: domain scientists vs. system experts
  - ➔ PERCS tools are focused on separation of concerns between domain scientists and system experts