X10 Programming:
Towards High Productivity High Performance Systems in the post-Moore's Law Era

Vivek Sarkar
(vsarkar@us.ibm.com)
Senior Manager, Programming Technologies
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
  - Christoph von Praun
  - Vijay Saraswat
  - Vivek Sarkar
- **X10 Tools**
  - Julian Dolby
  - Robert Fuhrer
  - Frank Tip
  - Mandana Vaziri

- **Publications**

1. "X10: An Object-Oriented Approach to Non-Uniform Cluster Computing", P. Charles, C. Donawa, K. Ebcioglu, C. Grothoff, A. Kielstra, C. von Praun, V. Saraswat, V. Sarkar. OOPSLA conference, October 2005 (to appear).
2. "Concurrent Clustered Programming", V. Saraswat, R. Jagadeesan. CONCUR conference, August 2005.
3. "X10: an Experimental Language for High Productivity Programming of Scalable Systems", K. Ebcioglu, V. Sarkar, V. Saraswat. P-PHEC workshop, February 2005.

Outline

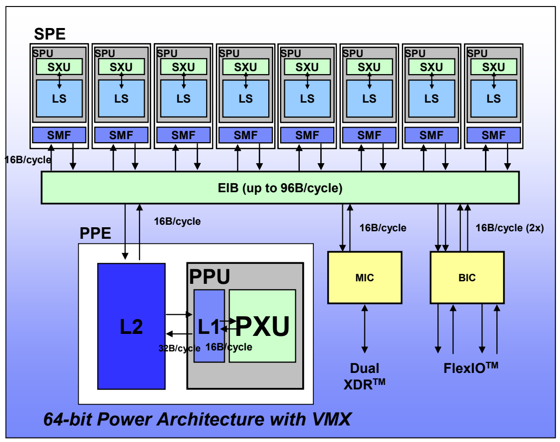
1. X10 Execution Model
   - Integration of multiple levels of concurrency and asynchronous data transfer
2. X10 Language and Environment
   - Extended subset of the Java™ language
   - X10 environment is integrated into Eclipse ecosystem
   - Synergies between HPC VM technologies (X10) and real-time VM technologies (Metronome)

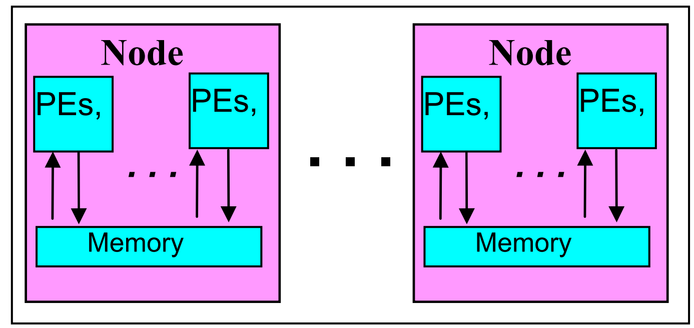
Future System Trends

Parallelism scaling replaces frequency scaling as foundation for increased performance

Scale-up Parallelism	Heterogeneous Parallelism (Co-processors, accelerators)	Scale-out Parallelism

SMP
Multi-core
SMT's
SIMD
ILP





Implications to software:

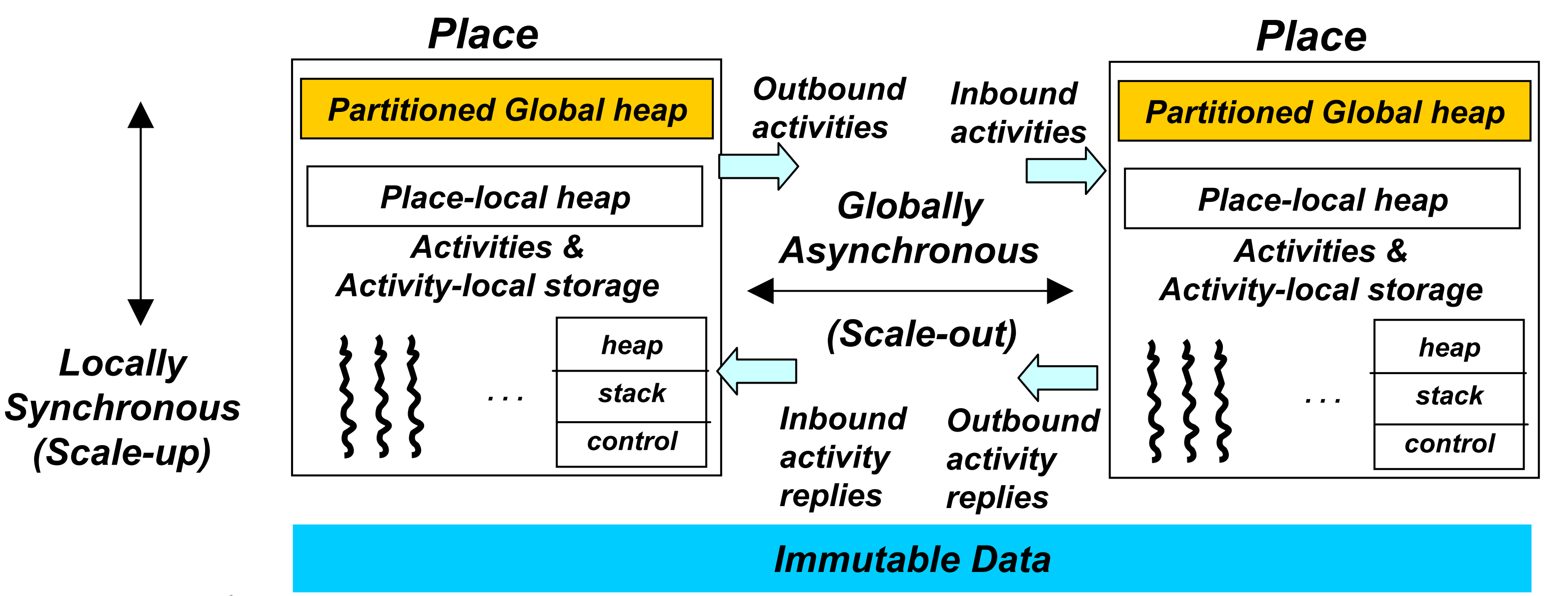
1) Exploit intra-process parallelism with non-uniform data affinities
2) Exploit inter-process parallelism in tightly coupled clusters of distributed nodes

Overview of X10 Execution Model

- *Asynchronous activities*
  - Unification of task parallelism and asynchronous data transfers
  - Ultra-lightweight “async” threads, augmented with (optional) loop-level constructs (“foreach”, “ateach”)
- *Coordination of parallel control flow*
  - “finish” and “clock” constructs
- *Coordination of data accesses*
  - “atomic” blocks, “future” and “force” constructs
- *Places*
  - Extension of Partitioned Global Address Space (PGAS) to Threaded Partitioned Global Address Space (T-PGAS)
  - Place = collection of non-migrating activities and mutable data
  - An activity can create a new activity at a local or remote place

Locality Rule in X10 Execution Model

- Any access to a shared mutable datum must be performed by an activity *at the same place* as the datum
  - Immutable data can be freely access from any place
- A BadPlaceException is thrown when the Locality Rule is violated



Threaded Partitioned Global Address Space model (T-PGAS)

X10 Execution Model: Examples

```
1) finish {
      async (A[R]) A[R] = 99; // Initiate remote put
      // Do other work in parallel
   }

2)   // Combine remote get and remote put, A[L] = A[R]
   async (A[R]){ final int v = A[R];
                 async (A[L]) A[L] = v; }

3) async (T[j]) atomic T[j]^=k; // Asynchronous atomic block

4) ateach ( point[j] : A.distribution)
       A[j] = f(j); // Equivalent to async(A[j]) A[j] = f(j)
```

X10 Execution Model: Examples

```
1) finish {
      async (A[R]) A[R] = 99; // Initiate remote put
      // Do other work in parallel
   }
```

Any local variable accessed by a child activity must be declared as final

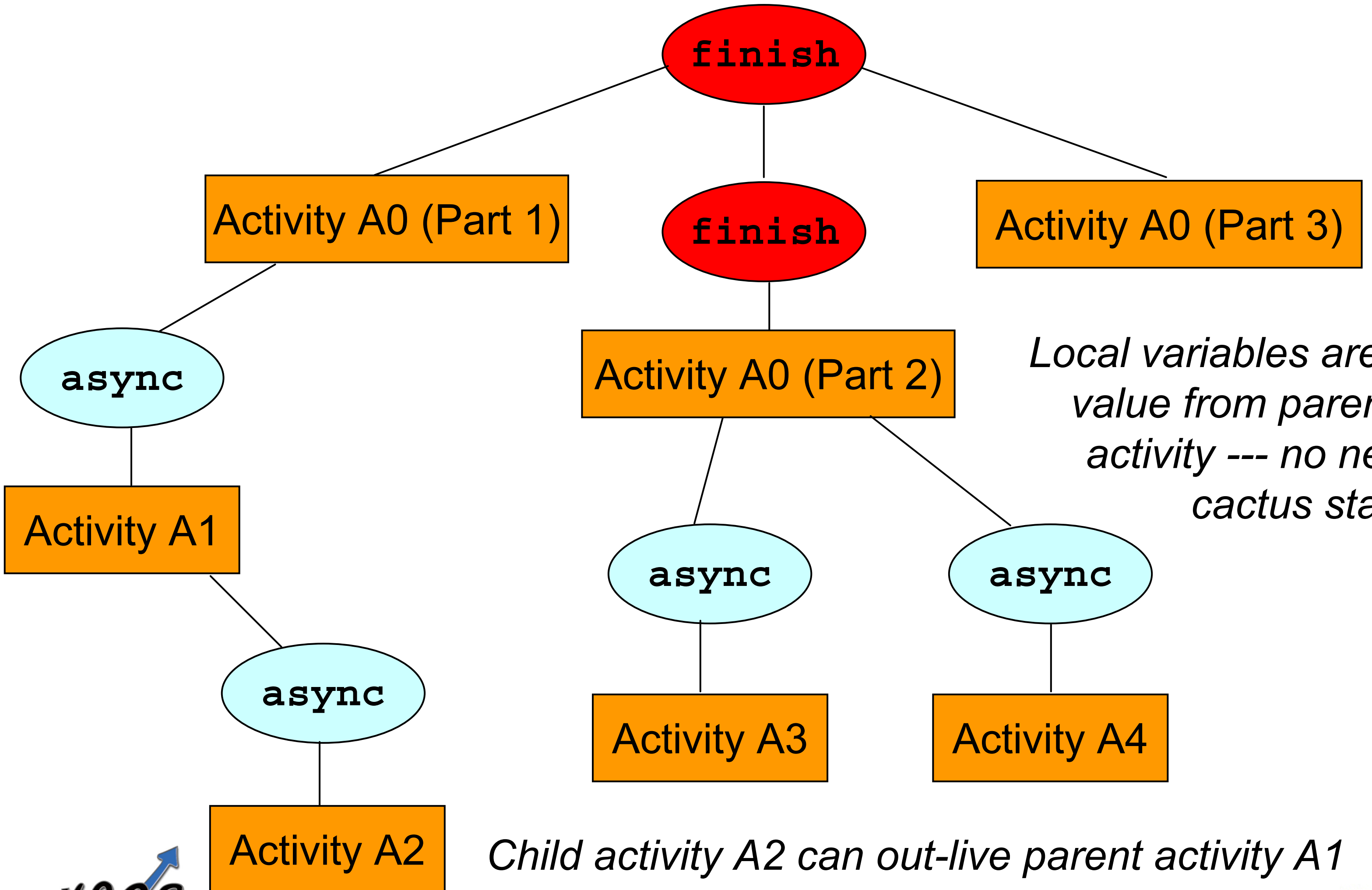
```
2)   // Combine remote get and remote put, A[L] = A[R]
   async (A[R]){ final int v = A[R];
                 async (A[L]) A[L] = v; }

3) async (T[j]) atomic T[j]^=k; // Asynchronous atomic block

4) ateach ( point[j] : A.distribution)
       A[j] = f(j); // Equivalent to async(A[j]) A[j] = f(j)
```

Activity body is in-line --- need not be extracted into a separate method or class

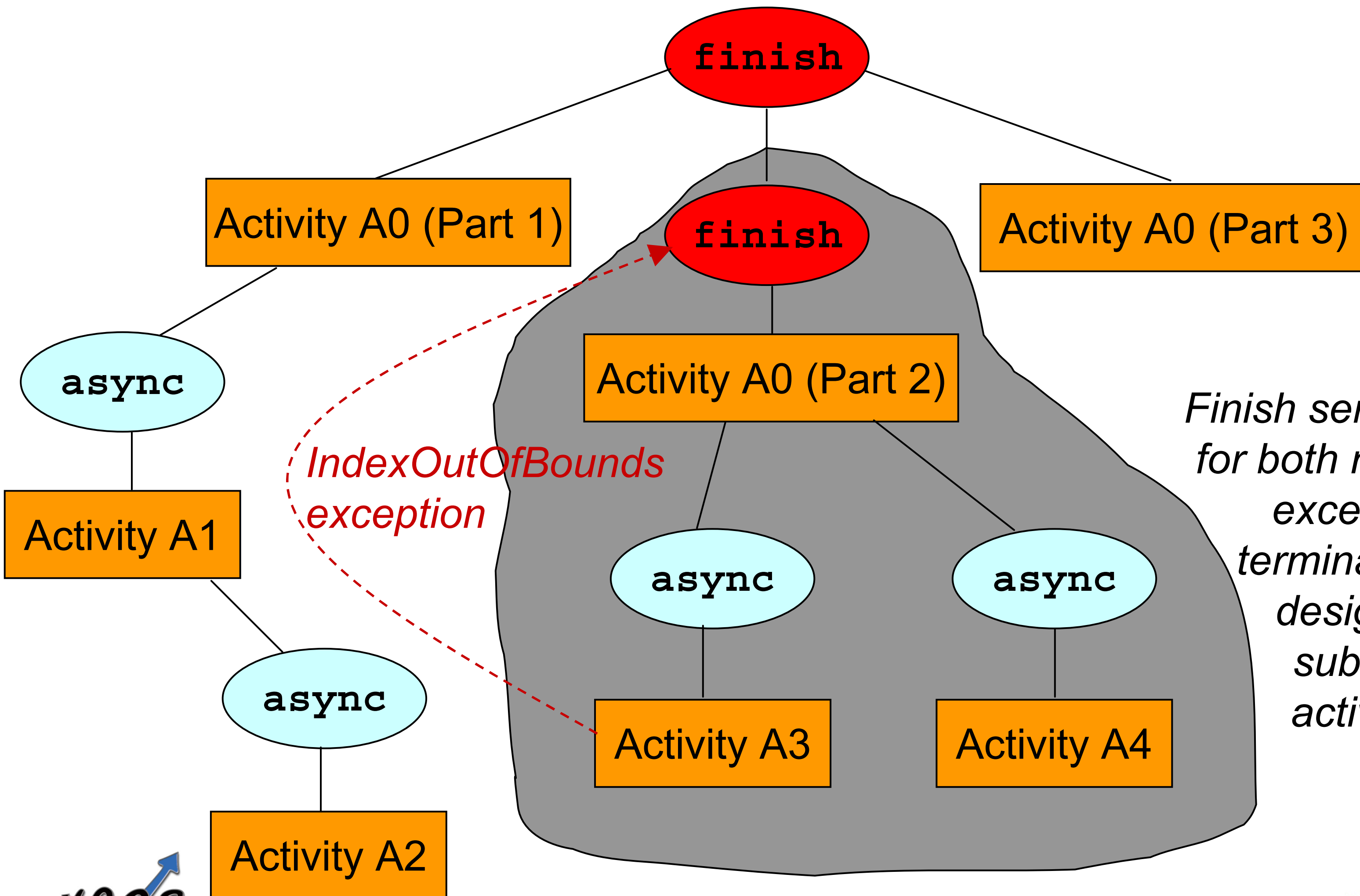
X10 Dynamic Activity Invocation Tree



Local variables are passed by value from parent to child activity --- no need for a cactus stack

Child activity A2 can out-live parent activity A1

X10 Dynamic Activity Invocation Tree



Finish serves as root for both normal and exceptional termination (for designated subset of activities)

Summary of X10 Execution model

Advantages:

- Any program written with atomic, async, finish, foreach, ateach, and clock parallel constructs *will never deadlock*
- Inter-node and intra-node parallelism integrated in a single model
- Remote activity invocation subsumes one-sided data transfer, remote atomic operations, active messages, . . .
- Finish subsumes point-to-point and team synchronization
- All remote data accesses are performed as activities ➔ rules for ordering of remote accesses follows simply from concurrency model

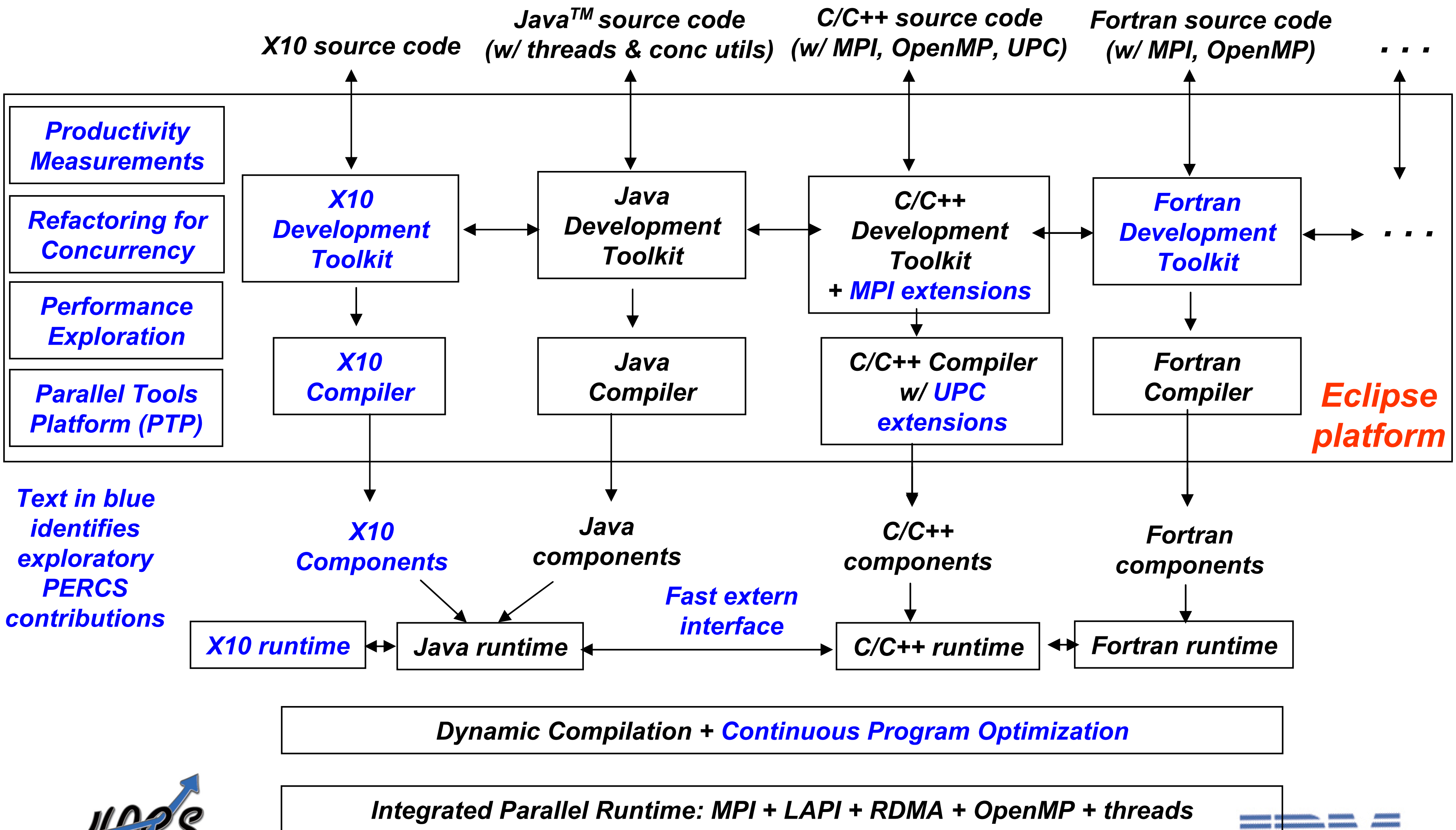
Applications:

- Can be easily mapped to multiple levels of parallel hardware (SIMD, SMT, coprocessors, cache prefetch, SMP, clusters, …)
- Can be used as target for multiple high level languages
  - X10 language serves as an exemplar

Outline

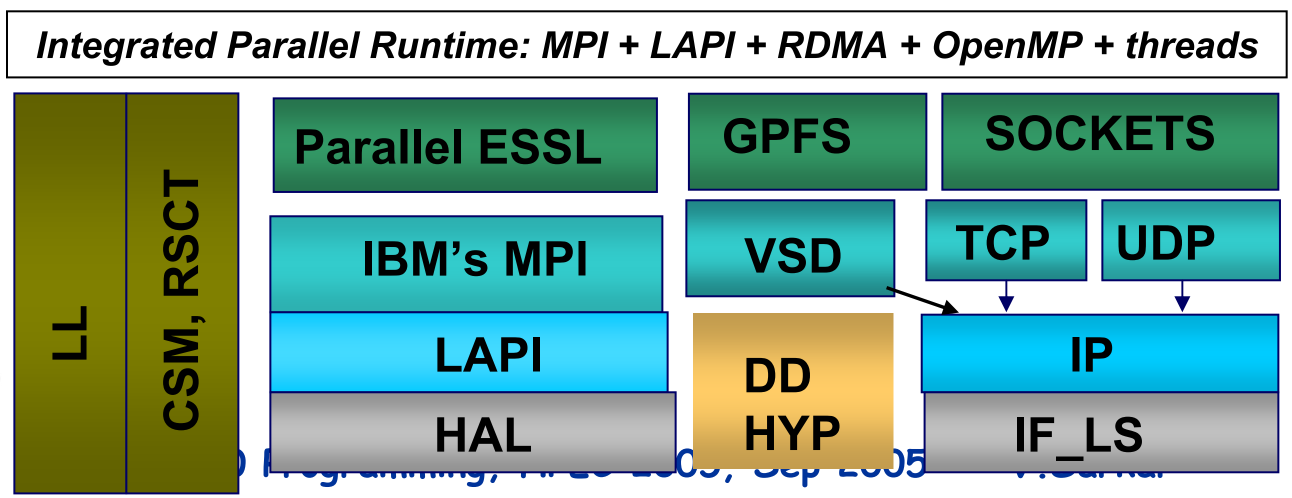
1. X10 Execution Model
   - Integration of multiple levels of concurrency and asynchronous data transfer
2. X10 Language and Environment
   - Extended subset of the Java™ language
   - X10 environment is integrated into Eclipse ecosystem
   - Synergies between HPC VM technologies (X10) and real-time VM technologies (Metronome)

PERCS Programming Model, Tools and Compilers
(PERCS = Productive Easy-to-use Reliable Computer Systems)



PERCS Programming Model: Position of X10 Language in Software Stack

Very High Level Languages (VHLL's), Domain Specific Languages (DSL's)	*Implicit parallelism, Implicit data distributions*
Components	*Domain-specific frameworks*
Libraries	*Collections, concurrency utils, …*
X10 Language	***X10 places and activities***
Deployment	*Mapping of places & activities to nodes in HPC Platform*
Managed Runtime	*Safety guarantees + dynamic comp.*



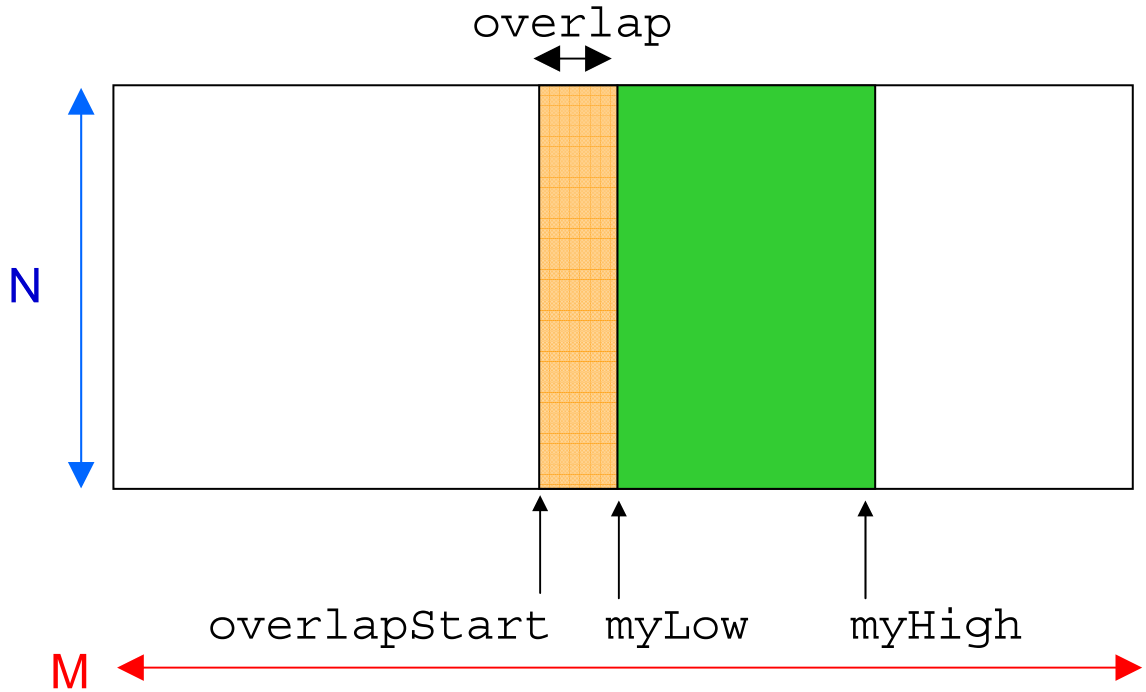
X10 vs. Java™ languages

- X10 is an extended subset of the Java language
  - Base language = Java 1.4 language
    - Java 5 features (generics, metadata, etc.) will be supported in the future
  - 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

Sequence Comparison Example: Local Alignment

- Goal: find the best matching subregions in a pair of sequences (e.g., DNA, RNA, sequence) so as to narrow down set of candidates for identifying biological relationships

Parallelization Algorithm



Each processor computes columns myLow..myHigh using columns overlapStart..myLow-1 as warm-up

X10 Version of Sequence Alignment (Serial Version)

```
void computeMatrix(int[.] A, value char[] c1, value char[] c2,
                   int firstCol, int lastCol) {
  // Dynamic programming algorithm
  for ( point[i,j] : [1:N,firstCol:lastCol] )
    M[i,j] = min4(0, a[i-1,j] + Gap, a[i,j-1] + Gap,
                   a[i-1,j-1] + (c1[i]==c2[j] ? Match : MisMatch));
}

// Main program
const int N = c1.length, M = c2.length;
. . .
A = new int[[0:N,0:M]];
computeMatrix(A, c1, c2, 1, M);
. . .
```

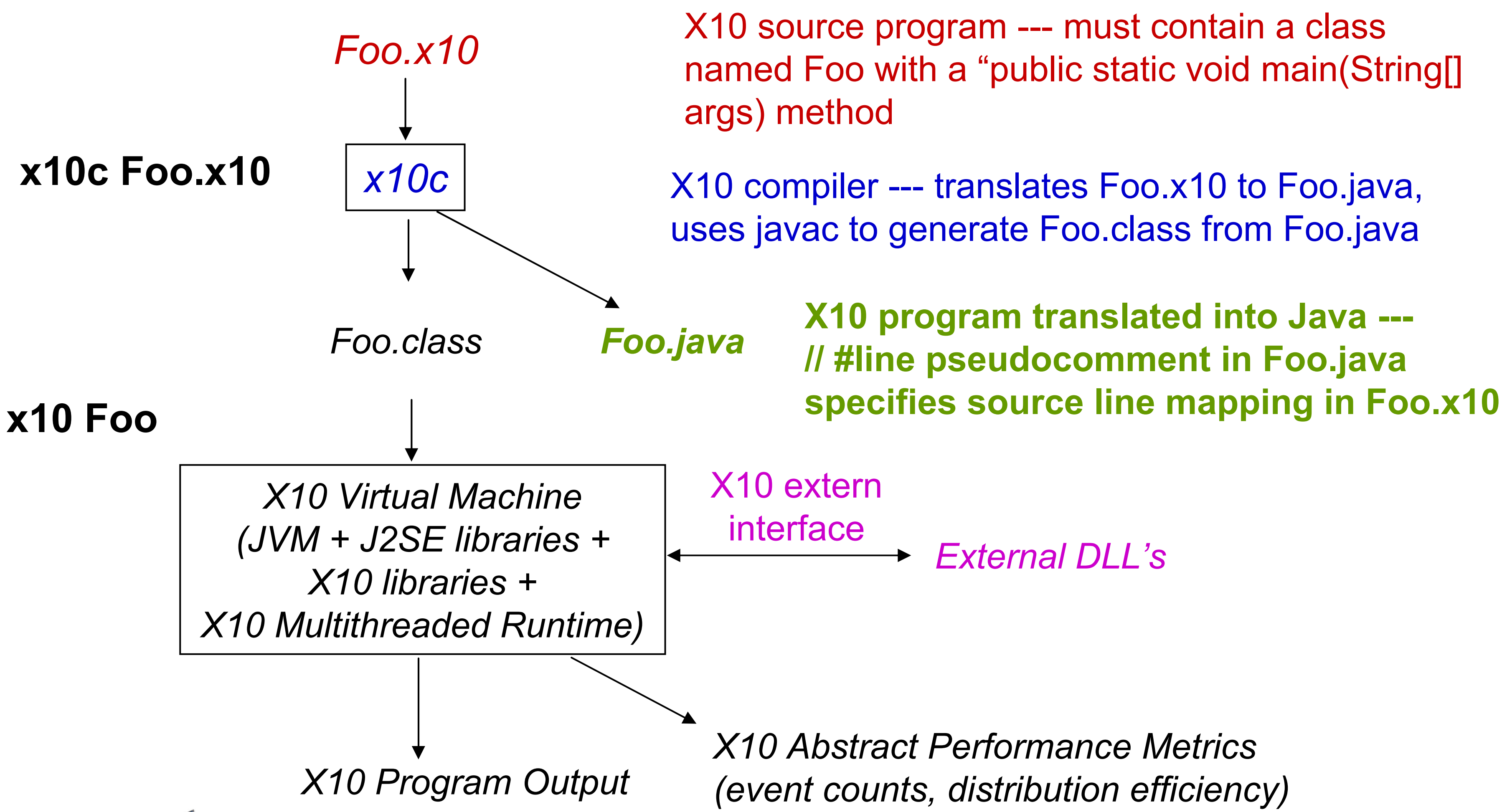
X10 Version of Sequence Alignment (Distributed Parallel Version)

```
// Allocate A with a [*,block] distribution
int[.] A = new int[dist.blockColumns([0:N,0:M])];
final int overlap = ceilFrac(N*(-Match),Gap) + N;
// SPMD computation at each place
finish ateach(point [i] : dist.unique()) {
  final dist myD = A.distribution | here; // sub-distribution for this place
  final int myLow = myD.region.rank(1).low();
  final int myHigh = myD.region.rank(1).high();
  final int overlapStart = max(0,myLow-overlap);
  final dist warmupD = [0:N,overlapStart:myLow]->here;
  final int [.] W = new int[warmupD]; // W = local warmup array
  computeMatrix(W, c1, c2, overlapStart+1, myLow);
  foreach (point[i]:[0:N]) A[i,myLow] = W[i,myLow]; // Copy col myLow
  // Compute my section of global array A
  computeMatrix(A, c1, c2, myLow+1, myHigh);
}
```

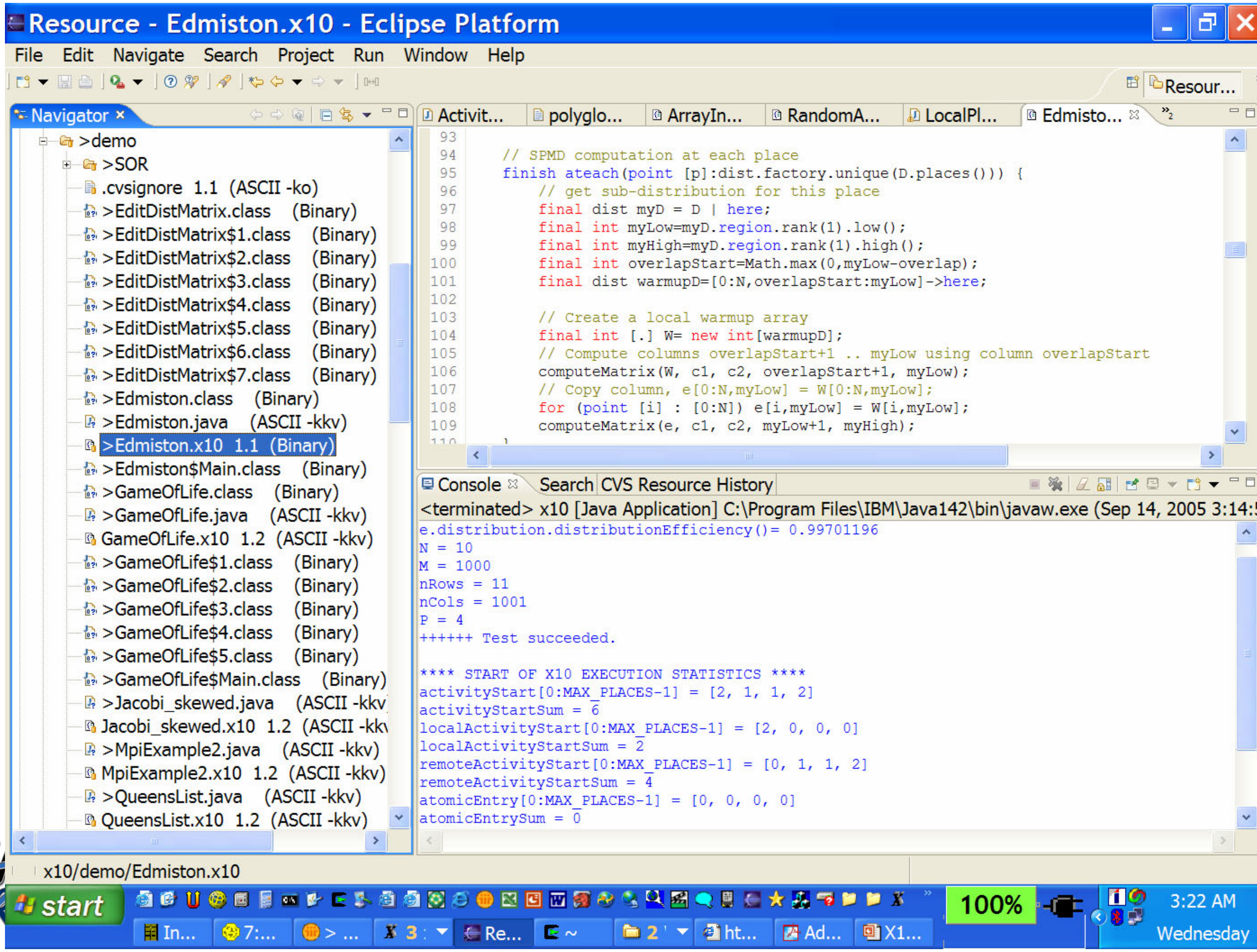
X10 Status

- **Reference implementation**
  - Used in PSC productivity study and university pilots
  - Nightly regression tests (~ 240 unit tests)
  - X10 application set starting to grow beyond unit tests
  - Plan for open source release at end of Phase 2
- **Performance Prototype**
  - Initial design for mapping X10 to LAPI using product J9 VM
  - Implementation has just begun
    - Bring-up of "hello world" X10 application on multiple nodes
- **X10 Development Toolkit (X10DT)**
  - Eclipse tools with basic language support (syntax highlighting, etc.)
  - Work started on X10-specific *refactorings*
    - Extract Async
    - Introduce atomic sections
- **Static Analysis and Ahead-Of-Time Optimization** (just starting)
  - Optimization of BadPlaceException checks
  - Use of static analysis to enhance Extract Async refactoring

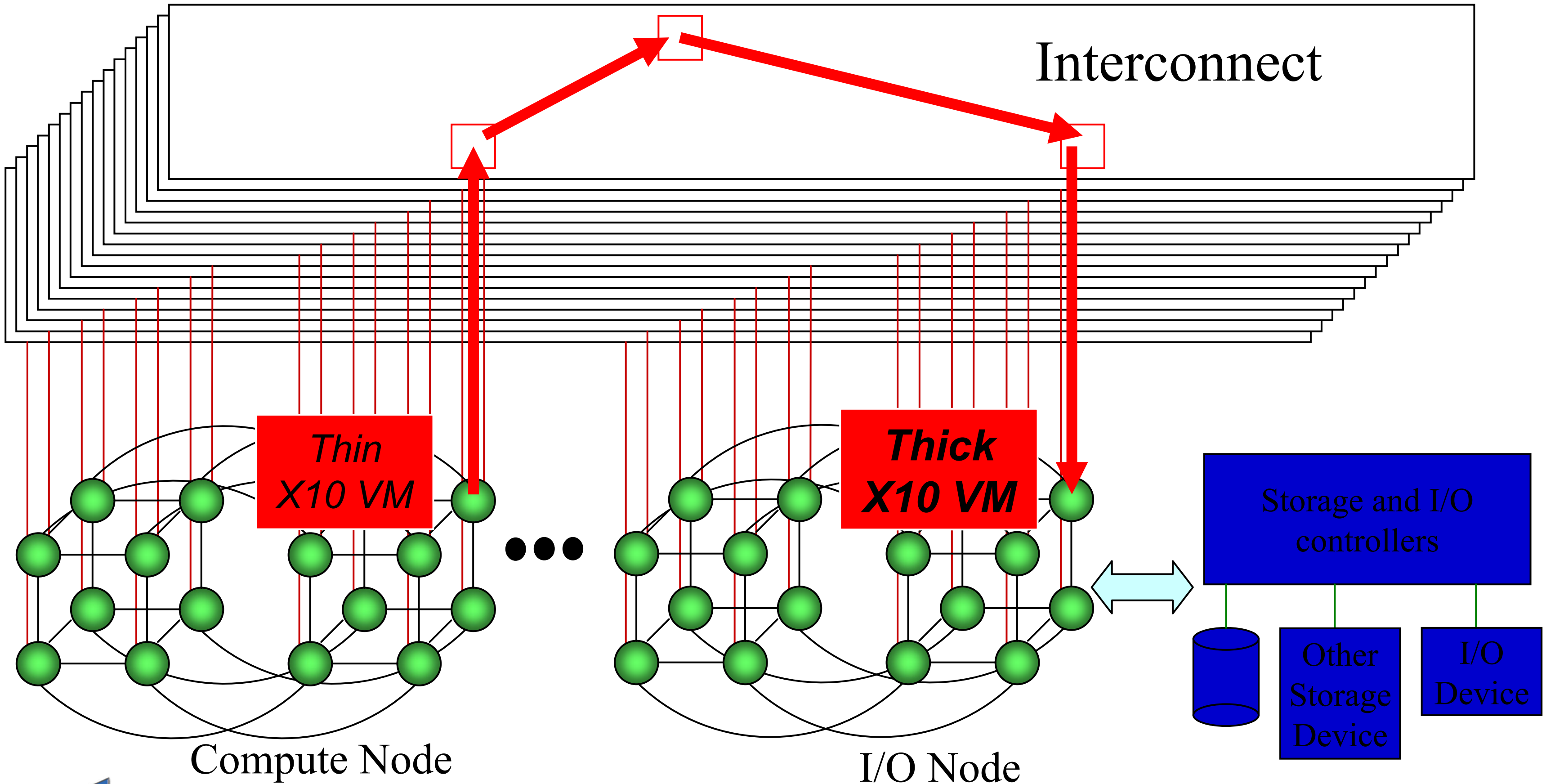
X10 Reference Implementation



X10 Reference Implementation: Screen Shot



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



Towards Increased Productivity in High Performance Embedded Computing: Expanding the frontiers of Virtual Machine Technologies

HPC VM Enhancements --- IBM PERCS/X10 project

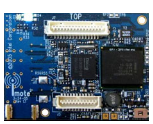
Real-time VM Enhancements --- IBM Metronome project

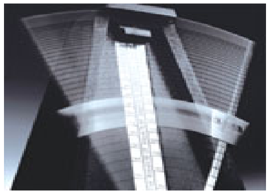
Commodity Virtual Machines

Reduced GC pause times improves HPC VM performance

Metronome → ***X10***

Reduced overhead for asynchronous atomic operations times improves Real-time VM performance

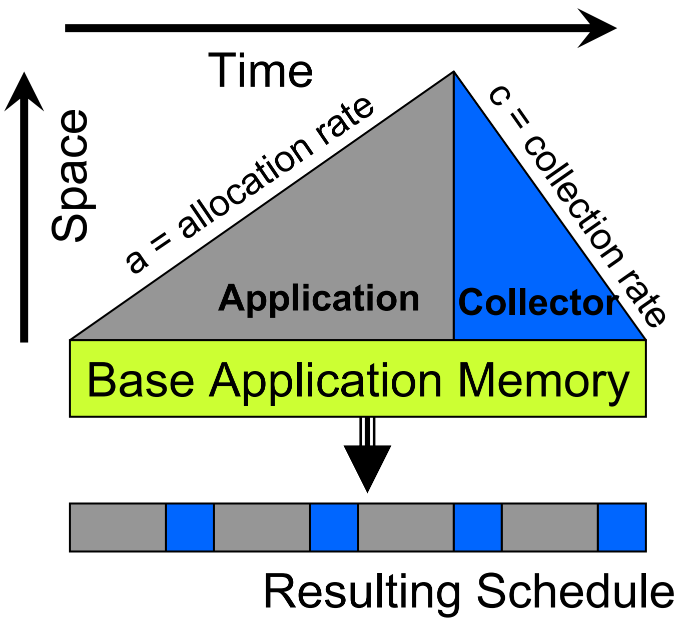


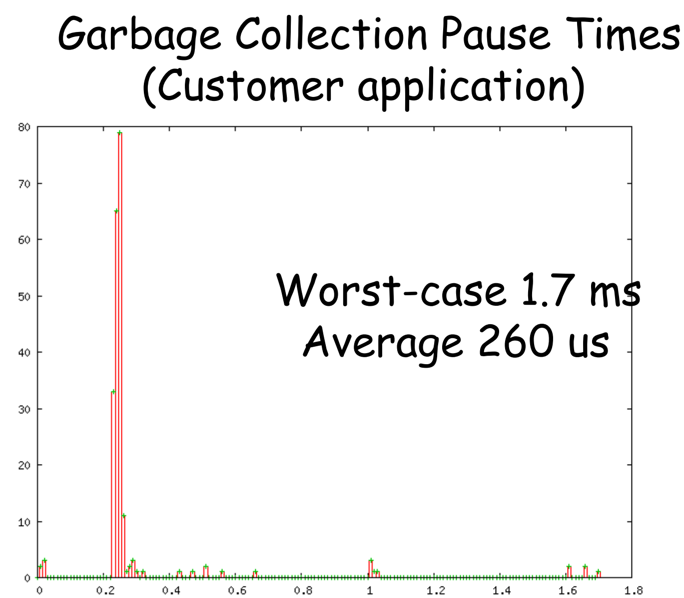


IBM Metronome project: Real-time Garbage Collection

David Bacon, Perry Cheng, David Grove, V.T. Rajan, Martin Vechev

- **Garbage collection is fundamental to Java's value proposition**
  - **Safety, reliability, programmer productivity**
  - **But also causes the most non-determinism (100 ms – 10 s latencies)**
  - **RTSJ standard does not support use of garbage collection for real-time**
- **Metronome is our hard real-time garbage collector**
  - **Worst-case 2 ms latencies; high throughput and utilization**
  - **100x better than competitors' best garbage collection technology**





Summary

- X10 Execution Model is designed for productivity and scalability
  - X10 language is our preferred embodiment, but we are also plan to explore other manifestations
- X10 tools are integrated into a common development environment (Eclipse)
  - We expect that the Parallel Tools Platform (PTP) project will seed a new community ecosystem for parallel tools
- Where we are looking for collaboration on X10
  - Porting applications to X10 for evaluation
  - Volunteers productivity studies
  - Standardization of T-PGAS runtime
    - multithreading with asynchronous one-sided data transfers
- Did not have time to cover
  - Clocks, futures, array language details, …
  - Additional advances in Java technologies (and their use in non-Java langs)
  - Additional work on improving productivity & expertise gap in PERCS project