SYSTEMS RESEARCH GROUP

DEPARTMENT OF COMPUTER SCIENCE THE UNIVERSITY OF HONG KONG

Towards Easy-to-use PGAS Parallel Programming –

The Distributed JVM Approach

Cho-Li Wang (王卓立)

The University of Hong Kong (香港大學)

CSO'10, Huangsan, China

Outline

1. Era of Petaflop Computing
2. PGAS Programing Language
3. Distributed Java Virtual Machine
4. Profile-guided locality management
5. Performance Evaluation

Era of PetaFlop Computing

Top500 Supercomputer List (Nov/2009)

Rank	Site	Computer/Year Vendor	# of cores	Linpack (R_{max})	R_{peak} (Teraflops/s)
1	Oak Ridge National Laboratory (USA)	**Jaguar – Cray** 2009, Cray Inc.	224162	**1759.00**	2331.00
2	DOE/NNSA/LANL, USA	**Roadrunner** , 2009 IBM	122400	**1042.00**	1375.78
3	National Institute for Computational Sciences/USA	**Kraken XT5** 2009, Cray Inc.	98928	831.70	1028.85
4	Forschungszentrum Juelich (FZJ) Germany	**JUGENE - Blue Gene/P** 2009 IBM	294912	825.50	1002.70
5	National SuperComputer Center in Tianjin/NUDT China	**Tianhe-1 天河一号, 2009 NUDT**	71680	563.10	1206.19
6	NASA/USA	Pleiades - SGI Altix ICE 8200EX,2009 SGI	56320	544.30	673.26
7	DOE/NNSA/LLNL (USA)	BlueGene/L/ 2007 IBM	212992	478.20	596.38
8	Argonne National Laboratory, USA	Blue Gene/P Solution / 2007 IBM	163840	458.61	557.06
9	Texas Advanced Computing Center,USA	Ranger - SunBlade x6420,2008, Sun Microsystems	62976	433.20	579.38
10	Sandia National Laboratories,USA	Red Sky - Sun Blade x6275, 2009 Sun Microsystems	41616	423.90	487.74
	OE/NNSA/LLNL,USA	Dawn - Blue Gene/P/ 2009 IBM	147456	415.70	501.35
	ow State University, Russia	Lomonosov - T-Platforms T-Blade2, 2009	35360	350.10	414.42
	ngszentrum Juelich, Germany	JUROPA - Sun Constellation,2009 Bull SA	26304	274.80	308.28
	upercomputing Center South Korea,	TachyonII - Sun Blade x6048, 2009, Sun Microsystems	26232	274.80	307.44

10^3	kilo
10^6	mega
10^9	giga
10^{12}	tera
10^{15}	peta
10^{18}	exa

Top 5 machines achieved PetaFlop computing power

China's Tianhe-1 Petaflop Computers

Hybrid structure: 6,144 Intel Xeon E5540 CPUs + 5,120 GPUs (ATI Radeon HD4870)

5th in TOP500
Peak performance: 1.2 PetaFLOPS
LINPACK score : 563.1 TeraFLOPS

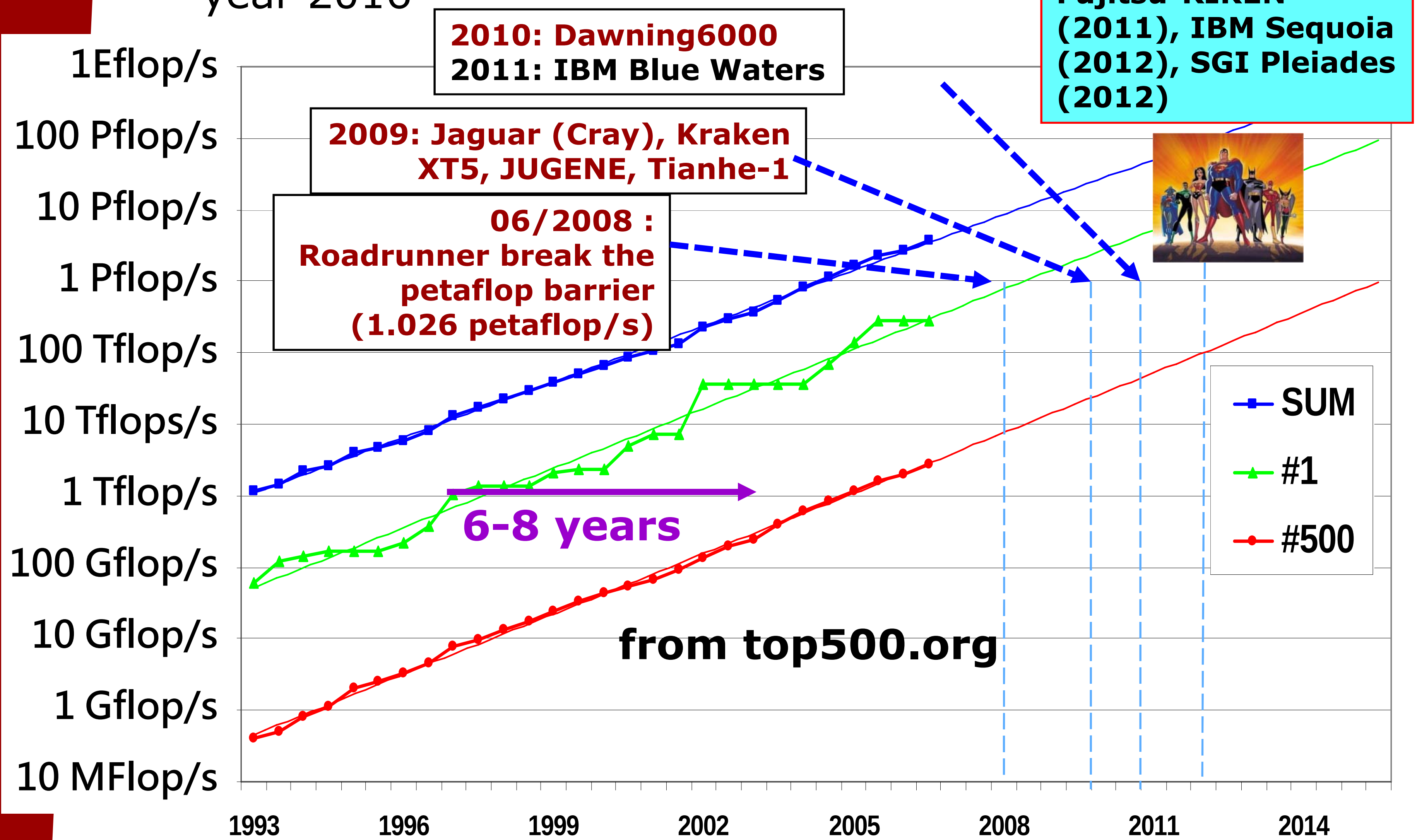
512 Operation Nodes In 20 cabinets **2560 Compute Nodes In 80 cabinets**

Source: Institute of Computer, NUDT

Petaflop Supercomputers with >1M cores

100 Petaflops system most likely in the year 2016

10 petaflops league: Cray Cascade (2010), Fujitsu-RIKEN (2011), IBM Sequoia (2012), SGI Pleiades (2012)



IBM Sequoia (20 petaflops)

To be installed at Lawrence Livermore National Laboratory

A petascale Blue Gene/Q supercomputer : **1.6 million processor cores** divided into 98,304 nodes placed within 96 Racks, record the amount of memory installed, equivalent to 1.6 petabytes

Dawning 6000 Petaflop Computer

- **Dawning 6000 consists of two parts,**
  - **<u>Dawning Nebulae (星云) GPU cluster</u>: 5000 blades, each contains two six-core INTEL 6-core X5650 2.66GHz processors and one NVIDIA C2050 Fermi GPU card. QDR Infiniband. <u>Peak : 3.5 Petaflops</u>. Linpack 1.27 Petaflops. (2nd in TOP500, May 30, 2010)**
  - **<u>Loongson (龙芯) cluster</u>: about 5000 blades w/ 8000 to 10,000 8-core Godson-3B processor (under development)**
- **Located at National Supercomputing Shenzhen Center (国家超级计算深圳中心)**
- **Total investment: 800M RMB (8亿元)**

8-core 龙芯 3

用一台普通电脑分析30年的气象数据需要20多年，而使用这台千万亿次超级计算机只需1小时

New Landscape of Parallel Computer Architecture

- **Multi-core Architectures**
  - **Conventional multicore approach (2, 4, 8 cores) -→ manycore technology (hundreds or even thousands of cores)**
  - **Employs simpler cores running at modestly lower clock frequencies**
- **Hardware accelerators**
  - **FPGA (Cray XD1, SGI RASC), GPU (Tianhe-1, Dawning6000, TSUBAME), Cell, ClearSpeed (TSUBAME) and vector processors, LINPACK?**
- **Networking:**
  - **RDMA : A one-sided put/get message can be handled directly by a network interface with RDMA support**
  - **TCP Offload Engine (TOE)**
  - **Most systems use either a 4X 10 Gbit/s (SDR), 20 Gbit/s (DDR) or 40 Gbit/s (QDR) connection.**
  - **End-to-end MPI latency : 1.07 microseconds**
  - **10 Gigabit Ethernet go mainstream (fallen to $500 per port)**

From Multi-core to Manycore

Micro-architecture	Clock Rate (GHz)	Cores	Threads Per Core	Caches
IBM Power 7	3.00 - 3.14	4-8	4	32KB+32KB Private L1 256KB Private L2 4MB Shared L3
Sun/Oracle Niagara2	1.2-1.6	4-8	8	8KB+8KB Private L1 4MB Shared L2
Intel Westmere	1.86 - 2.66	4-8	2	32KB+32KB Private L1 256KB Private L2 12-24 MB Shared L3
Intel Harpertown	2.00 - 3.40	4	2	32KB+32KB Private L1 2x6MB L2 Cache
AMD Magny-Cours	1.7 - 2.3	12 or 16	1	64KB+64KB Private L1 512KB Private L2 2x6 MB Shared L3
Intel Single-Chip Cloud	1.0	48	1	16KB L1 Cache 256KB Private L2 Cache 16KB Msg Buffer per Tile
Intel Terascale	~ 4	80	1 ?	3KB Instruction + 2KB Data on each Core
Tilera Tile-GX	1.5	100	1 ?	32KB+32KB Private L1 256KB L2 Private L2 26MB Distributed L3

Outline

1 Era of Petaflop Computing

2 PGAS Programing Language

3 Distributed Java Virtual Machine

4 Profile-guided Locality Management

5 Performance Evaluation

10

Predictions

- Parallelism will explode
  - Number of cores will double every 12-24 months
  - Petaflop (million processor) machines will be common in HPC by 2015
- **Performance will become a software problem**
  - Parallelism and locality are key
  - Concurrency is the next major revolution in how we write software
- **A new programming model will emerge for petaflop computing**

Do we put enough emphasis on software?

Berkeley's Dr. Kathy Yelick (director of NERSC) :

No. Unfortunately, the race for each major performance milestone, has resulted in a de-emphasis on software.

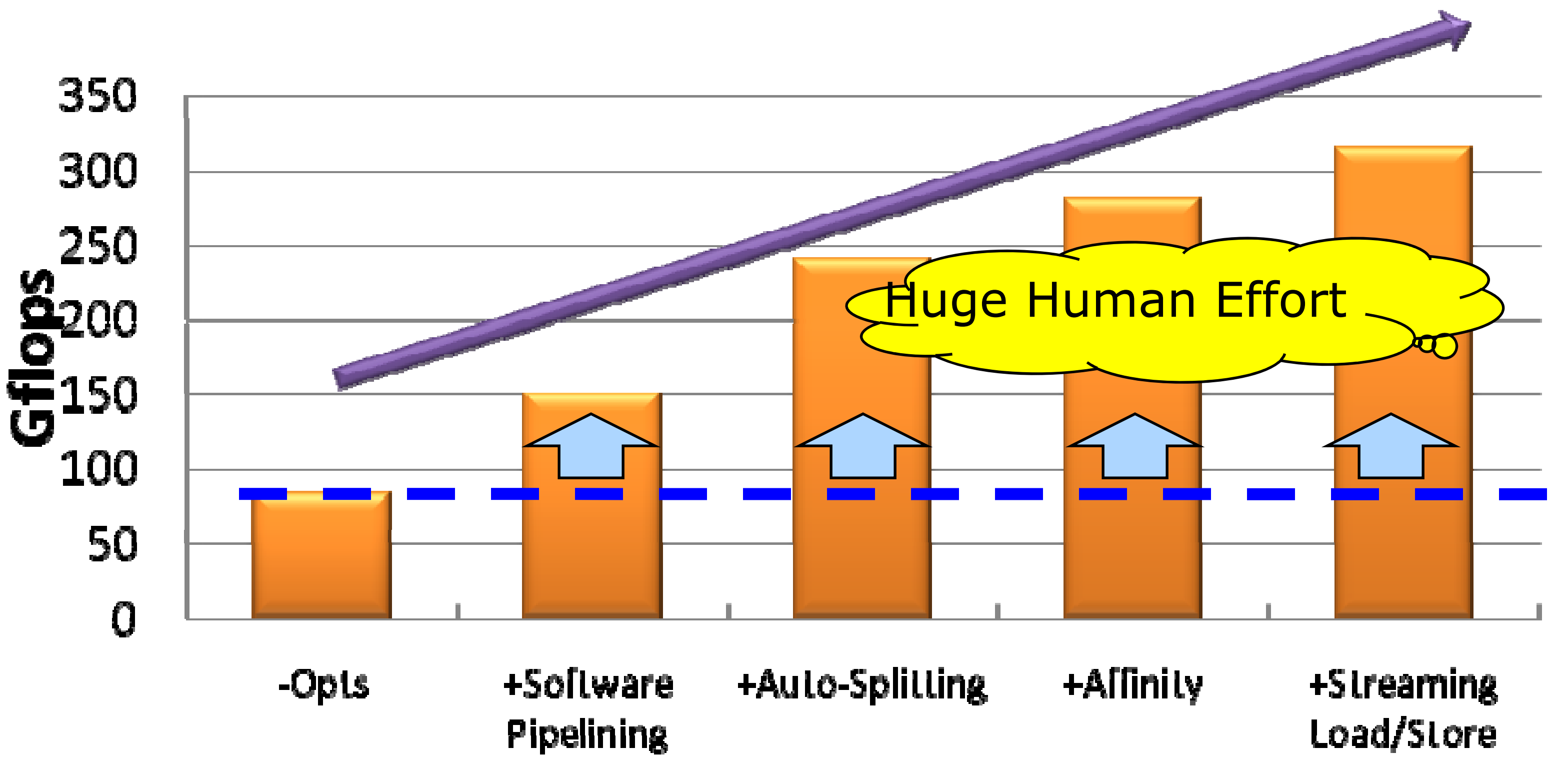
Source: The Software Challenges of Petascale Computing

Parallel Programming

- **Most parallel programs are written using:**
  - **Message passing**
    - **Examples: CM5's CMMD, PVM, IBM's MPL,**
    - **Current standard: MPI (MPICH-1, MPICH-2, LAM/MPI..**
    - **Usually used for scientific applications with C++/Fortran, or Java (JavaMPI, G-JavaMPI)**
    - **Scales easily: user controlled data layout**
    - **Hard to use: send/receive matching, message packing/unpacking**
  - **Shared memory**
    - **Examples: OpenMP, pthreads, Java**
    - **Usually for non-scientific applications**
    - **Easier to program: direct reads and writes to shared data**
    - **Hard to scale: (mostly) limited to SMPs, no concept of locality**

Optimizing is Hard !

- **Tianhe-1 Experience: Scaling LINPACK performance from 20% to 70% of each CPU-GPU pair**



Source: *Dr. Chunyuan Zhang*, National University of Defense Technology

Parallel Programming environments since the 90's

Do you like to design another ONE ?

ABCPL
ACE
ACT++
Active messages
Adl
Adsmith
ADDAP
AFAPI
ALWAN
AM
AMDC
AppLeS
Amoeba
ARTS
Athapascan-0b
Aurora
Automap
bb_threads
C4
CC++
Chu
Charlotte
Charm
Charm++
Cid
Cilk
CM-Fortran
Converse
Code
COOL

CORRELATE
CPS
CRL
CSP
Cthreads
CUMULVS
DAGGER
DAPPLE
Data Parallel C
DC++
DCE++
DDD
DICE.
DIPC
DOLIB
DOME
DOSMOS.
DRL
Express
Falcon
Filaments
FM
FLASH
The FORCE
Fork
Fortran-M
FX
GA
GAMMA
Glenda

GLU
GUARD
HAsL.
Haskell
HPC++
JAVAR.
HORUS
HPC
IMPACT
ISIS.
JAVAR
JADE
Java RMI
javaPG
JavaSpace
JIDL
Joyce
Khoros
Karma
KOAN/Fortran-S
LAM
Lilac
Linda
JADA
WWWinda
ISETL-Linda
ParLin
Eilean
P4-Linda
Glenda
POSYBL
Objective-Linda
LiPS
Locust
Lparx
Lucid
Maisie
Manifold

Mentat
Legion
Meta Chaos
Midway
Millipede
CparPar
Mirage
MpC
MOSIX
Modula-P
Modula-2*
Multipol
MPI
MPC++
Munin
Nano-Threads
NESL
NetClasses++
Nexus
Nimrod
NOW
Objective
Linda
Occam
Omega
OpenMP
Orca
OOF90
P++
P3L
p4-Linda
Pablo
PADE
PADRE
Panda
Papers
AFAPI.
Para++
Paradigm

Parafrase2
Paralation
Parallel-C++
Parallaxis
ParC
ParLib++
ParLin
Parmacs
Parti
pC
pC++
PCN
PCP:
PH
PEACE
PCU
PET
PETSc
PENNY
Phosphorus
POET.
Polaris
POOMA
POOL-T
PRESTO
P-RIO
Prospero
Proteus
QPC++
PVM
PSI
PSDM
Quake
Quark
Quick Threads
Sage++
SCANDAL
SAM

pC++
SCHEDULE
SciTL
POET
SDDA.
SHMEM
SIMPLE
Sina
SISAL.
distributed
smalltalk
SMI.
SONiC
Split-C.
SR
Sthreads
Strand.
SUIF.
Synergy
Telegrphos
SuperPascal
TCGMSG.
Threads.h++.
TreadMarks
TRAPPER
uC++
UNITY
UC
V
ViC*
Visifold V-NUS
VPE
Win32 threads
WinPar
WWWinda
XENOOPS
XPC
Zounds
ZPL

Let me add one more?

Source: John Urbanic, Pittsburgh Supercomputing Center

The Software challenges of Petaflop computing

- **New algorithmic approaches to increase the levels of concurrency on the order of 10^8**
- **Developing effective methodologies for assessing and exploiting data locality (high cache hit rates) in the deep memory hierarchies**
- **Hide latency by utilizing low-level parallelism (e.g., prefetch queues and multithreading)**
- **Design algorithms and implementations that permit easy recovery from system failures**
- **Performance monitoring facilities (accurate timers and operation counters, out-of-cache loads and stores) and dynamic load balancing**
- **Accuracy and stability of numerical methods: formal methods to certify the correctness of petaflops algorithms and hardware logic designs**
- **New languages and constructs (alternatives to HPF, OpenMP, MPI,..) ??**

Programmability in HPC

- **Relevant research area in the last years**
  - **Growing interest on easier programming**
- **HPCS project (DARPA)**
  - **High-performance High-Productivity Programming**
  - **New languages that focus on programmability (IBM X10, Cray CHAPEL, Sun Fortress)**
- **PGAS (Partitioned Global Address Space):**
  - **Target global address space, multithreading platforms**
  - **Aim for high levels of scalability**
  - **<u>Research languages :</u>**
    - **Co-Array Fortran (CAF)**
    - **Unified Parallel C (UPC)**
    - **Titanium (Java)**

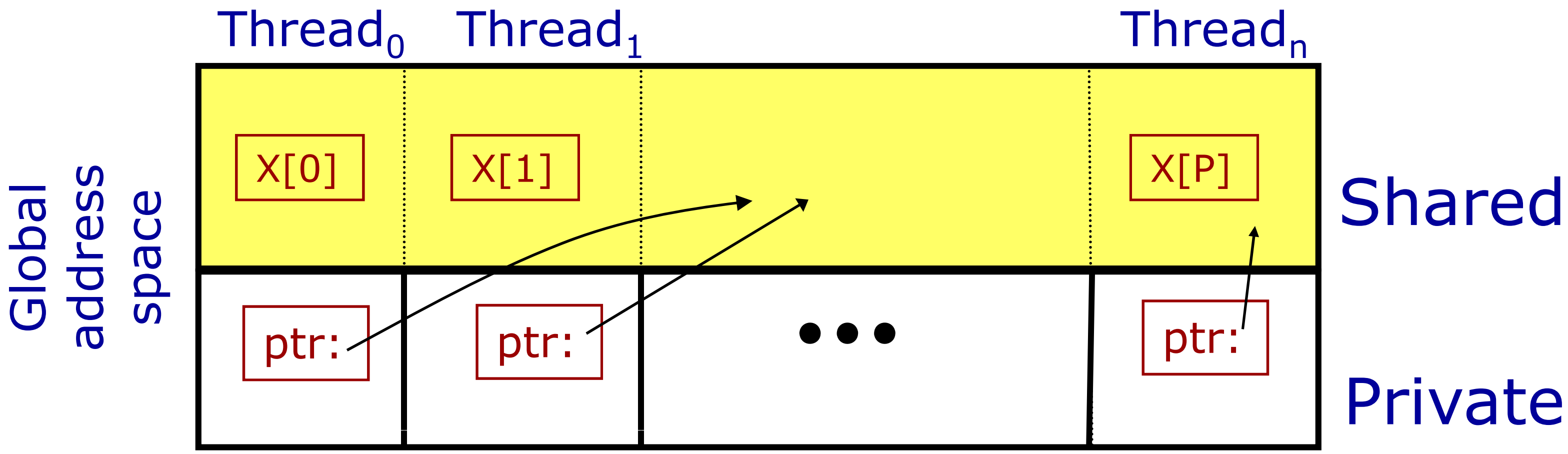
HPCS High Productivity Computer Systems

Features of PGAS Languages

- **Explicitly-parallel programming model with SPMD parallelism**
  - **Fixed at program start-up, typically 1 thread per processor**
- **Global address space model of memory**
  - **Allows programmer to directly represent distributed data structures**
  - **Can access local and remote data with same mechanisms**
- **Address space is logically partitioned**
  - **Local vs. remote memory (two-level hierarchy) – handled by users**
- **Programmer control over performance critical decisions (** burden to users **)**
  - **Data layout and communication**
- **Base languages differ: Co-Array Fortran (CAF) Unified Parallel C (UPC), Titanium (Java)**

Source: Yelick's (UCB) CS267 Lecture

Global Address Space Eases Programming



- **The languages share the global address space abstraction**
  - **Shared memory is partitioned by processors**
  - **Remote memory may stay remote: no automatic caching implied**
  - **One-sided communication through reads/writes of shared variables**
  - **Both individual and bulk memory copies**
- **Differ on details**
  - **Some models have a separate private memory area**
  - **Distributed array generality and how they are constructed**

Source: Yelick's (UCB) CS267 Lecture

Programmer Productivity?

- **Languages (or language technologies) that solve real problems can succeed [Todd A. Proebsting, Microsoft Research, 2002]:**
  - **Even if slow**
  - **Even with simple types**
  - **Even without academic significance (no papers?)**
  - **Even without rocket science**
  - **If useful**
- **Programmer Productivity:**
  - **Write programs correctly (50% of crashes caused by 1% of bugs)**
  - **Write programs quickly**
  - **Write programs easily**
- **Why?**
  - **Decreases support cost**
  - **Decreases development cost**
  - **Decreases time to market/solution**
  - **Increases satisfaction**

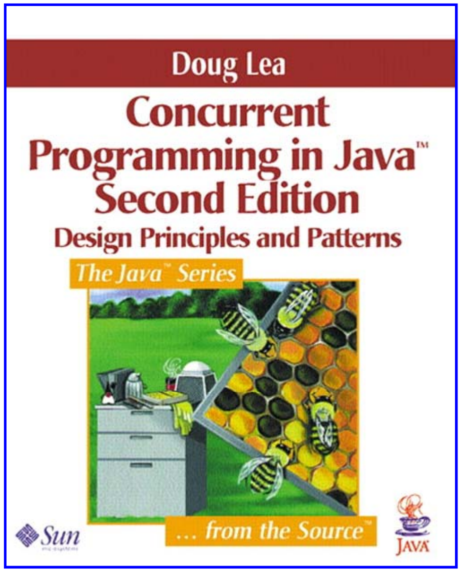
But … .. "New Language Fear"

- **Long-Live Language Needed:**
  - **Large-scale codes: *portability* is top priority.**
  - **Large-scale codes lifetimes : 10 to 30 years.**
  - **High-performance computers : 3-5 years between generations .**
  - **They can't risk spending 5-10 years writing their code in a new language only to find that the new language didn't gain general acceptance and support.**
- **Fear of learning new language:**
  - **Some people say that "if there's a lot of pain involved, they won't switch to a new programming language."**
- **How can you motivate people to migrate to a more efficient new language? Or do they have to ?**

Why Java for HPC ?

- Good programmability for potential HPC
  - Expressive grammar: simplified C++
  - Concurrent language: multithreading support at language level (Portable way of parallel programming)
  - Platform independence: bytecode (write once, run everywhere !)
  - Runtime: GC, safety checking, etc.
  - Libraries: a huge increasing list
  - Deliver 65%-90% of performance of the best Fortran programs; compete with C++:
  - Java-based next-gen languages : X10 (IBM), Titanium, Fortress (Sun)
- Easy to learn.
  - Write Java programs quickly
  - Write Java programs easily
  - Less bugs (?)

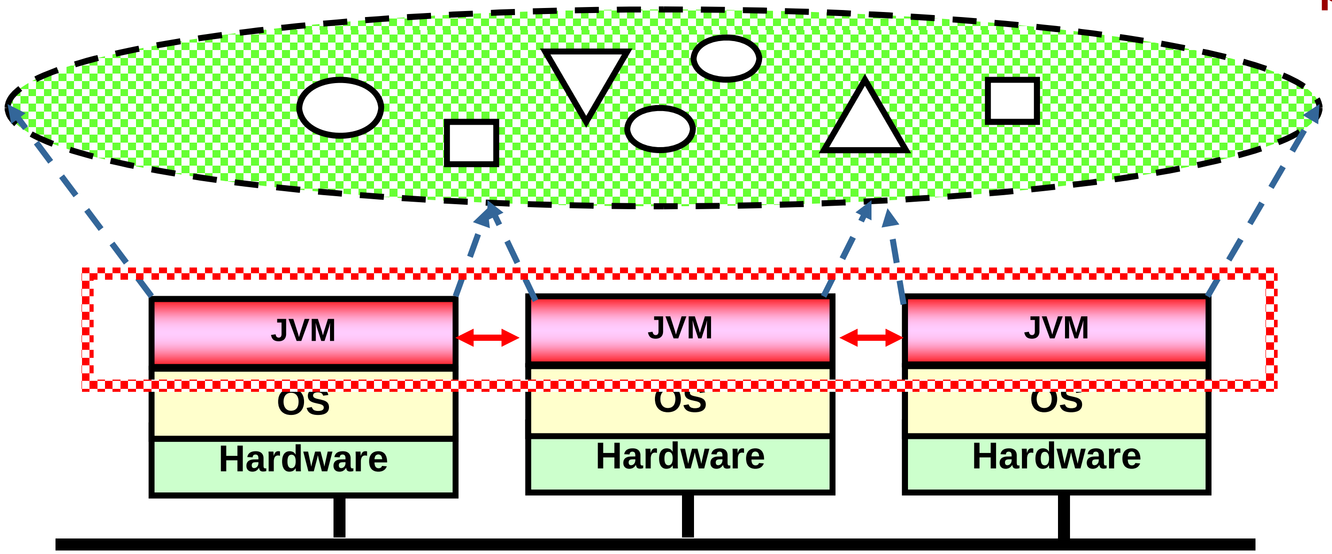
"Java as the first language"



Our Approach

Distributed Java Virtual Machine

Single system image (SSI) illusion to threads of a Java program



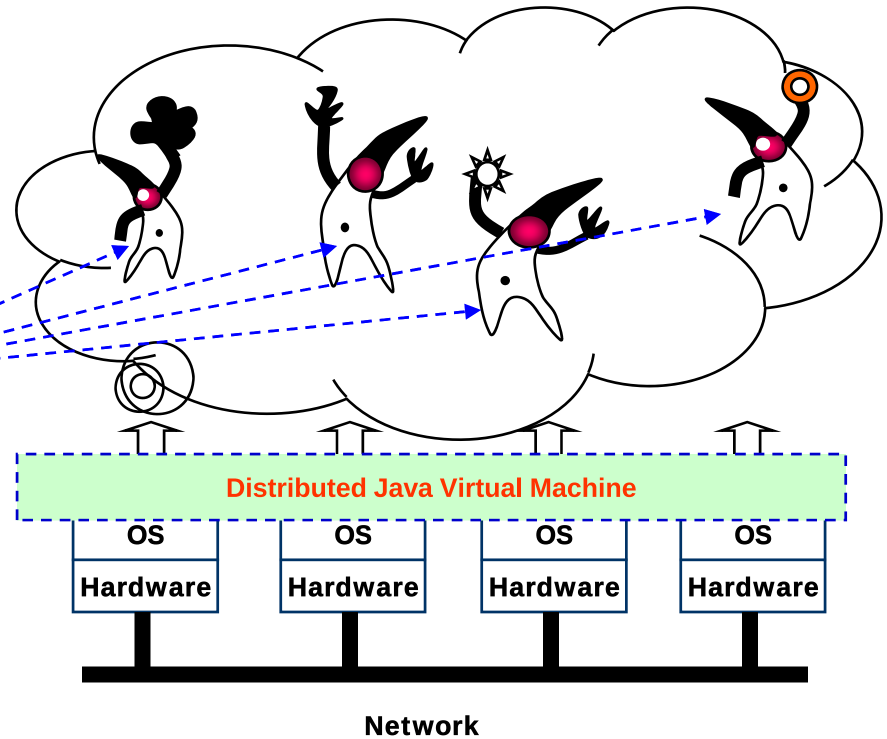
Distributed Java Virtual Machine

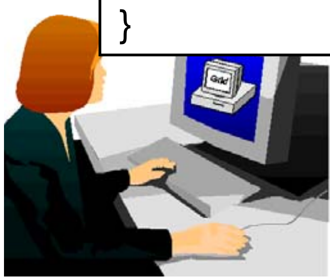
```
class worker extends Thread {
   private long n;
   public worker(long N) { n = N; }
   public void run() { long sum= 0;
    for(long i = 0; i < n; i++) sum += i;
    System.out.println("Sum = " + sum);}
}
public class test { static final int N=100;

  public static void main(String args[]) {
    worker[] w= new worker[N];
    Random r = new Random();
  for (int i=0; i<N; i++)
    w[i] = new worker(r.nextLong());
 for (int i=0; i<N; i++)  w[i].start();
 try{ for (int i=0; i<N; i++) w[i].join();}
 catch (Exception e){}}
}
```

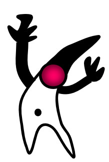
Multithreaded Java application

DJVM hides the physical boundaries between machines
Support thread migration





programmer

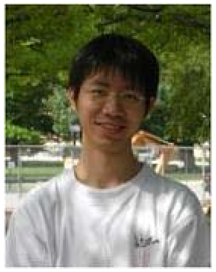


Java thread

History and Roadmap of JESSICA

- **JESSICA V1.0 (1996-1999)**
  - **Execution mode: Interpreter Mode**
  - **JVM kernel modification (Kaffe JVM)**
  - **Global Heap: built on top of TreadMarks (Lazy Release Consistency + homeless)**
- **JESSICA V2.0 (2000-2006)**
  - **Execution mode: JIT-Compiler Mode (full speed)**
  - **JVM kernel modification (Kaffe JVM)**
  - **Lazy Release Consistency + migrating-home protocol**
- **JESSICA V3.0 (2008~2010?)**
  - **Built above JVM (JVMTI)**
  - **Support Large Object Space**
  - **For any JVM. Run @ full speed of the underlying JVM.**
- **JESSICA v.4 (2009~)**
  - **Software transactional memory model**
  - **Multicore/GPU cluster**

Past Members



King Tin LAM,

Chenggang Zhang

Kinson Chan

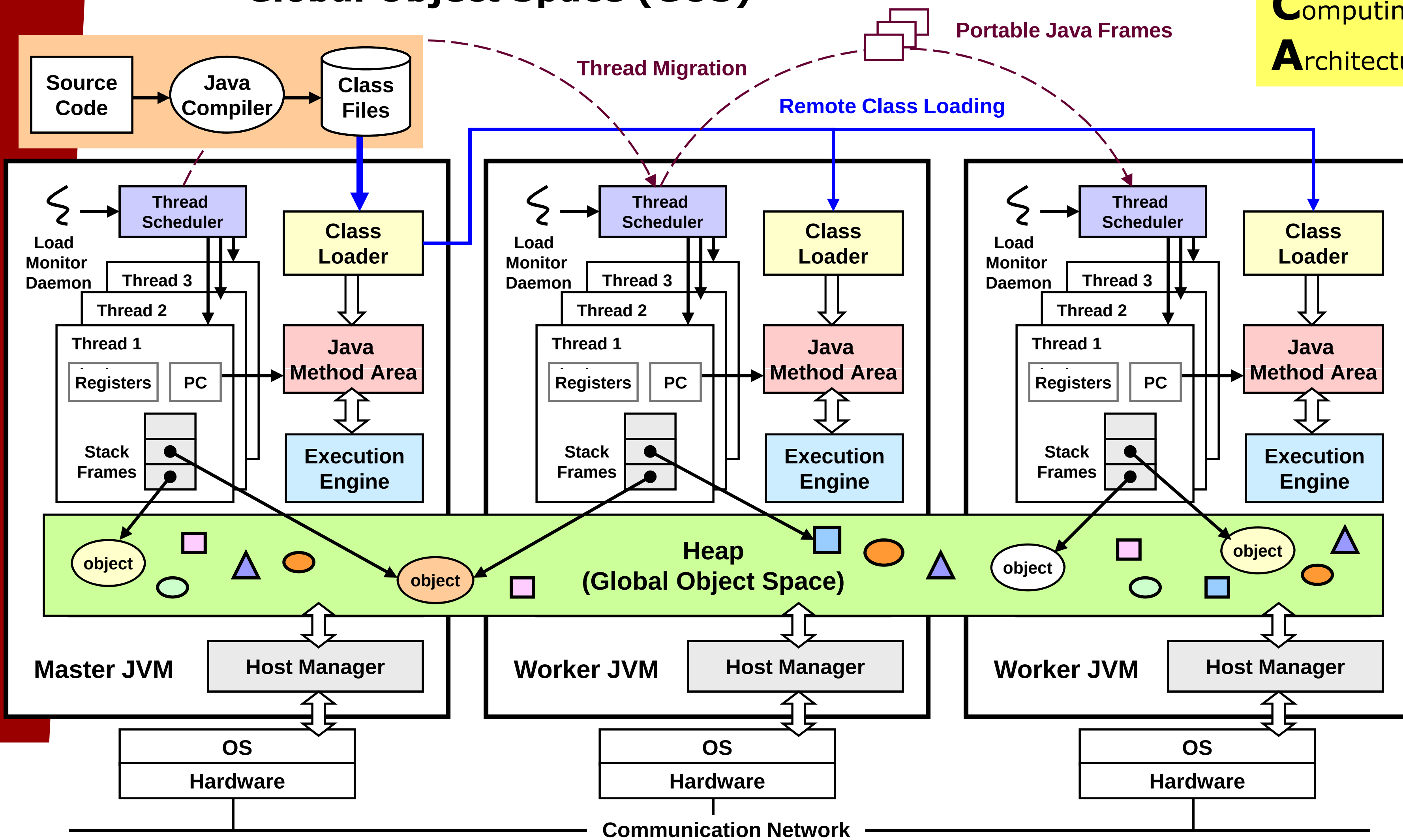
Ricky Ma

Current Members

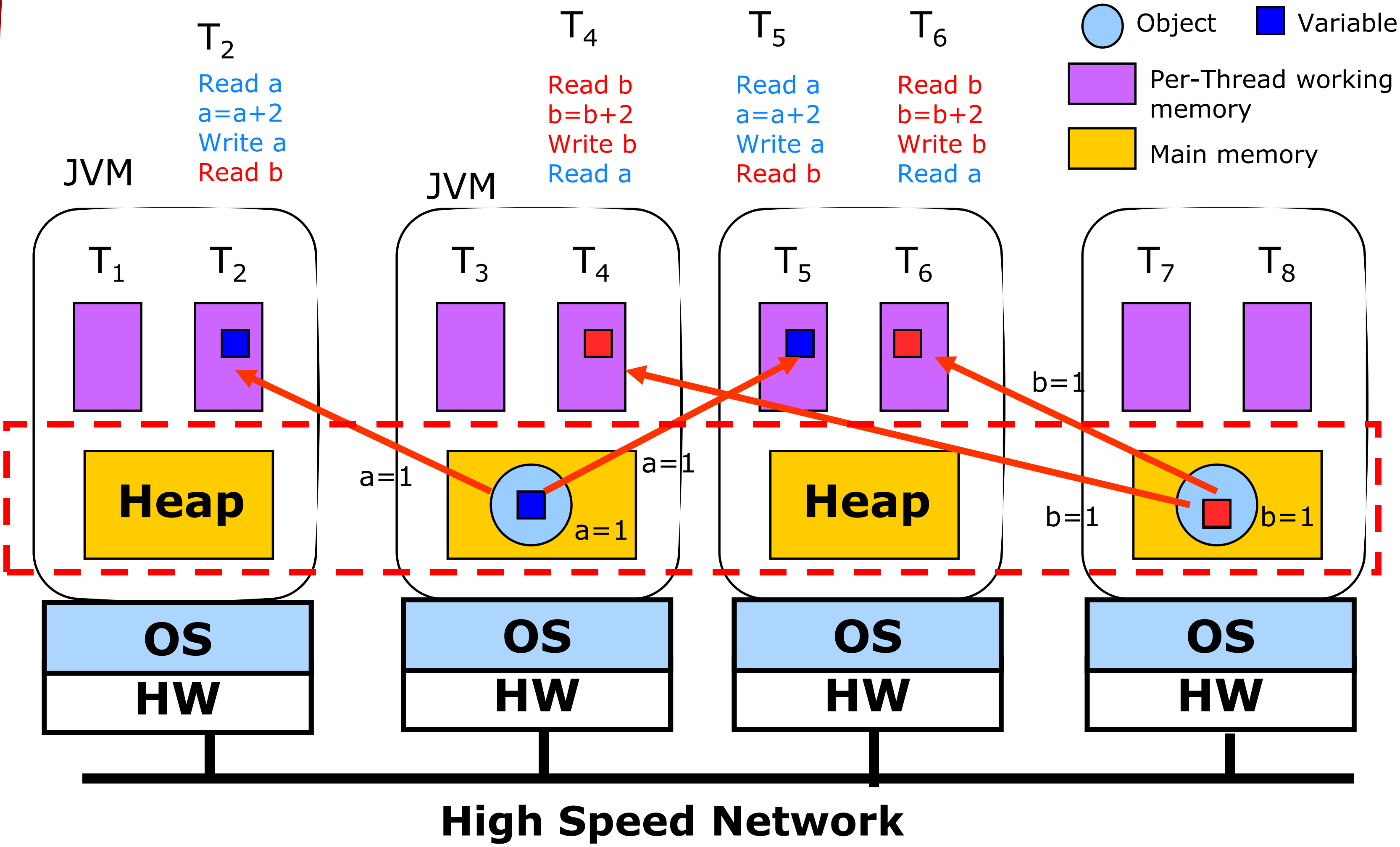
JESSICA Distributed Java VM

- **A cluster-wide JVM with**
  - **Dynamic thread mobility in JIT mode**
  - **Global Object Space (GOS)**

Java
Enabled
Single
System
Image
Computing
Architecture



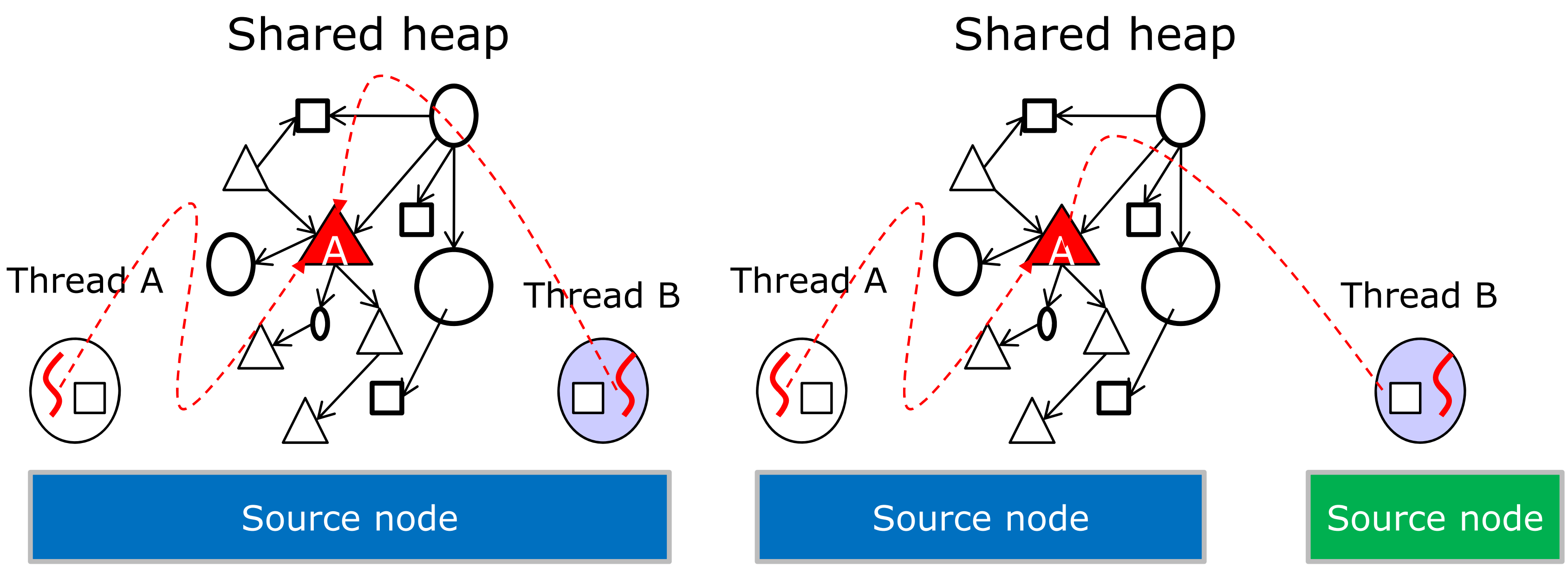
Problem 1: Memory Consistency



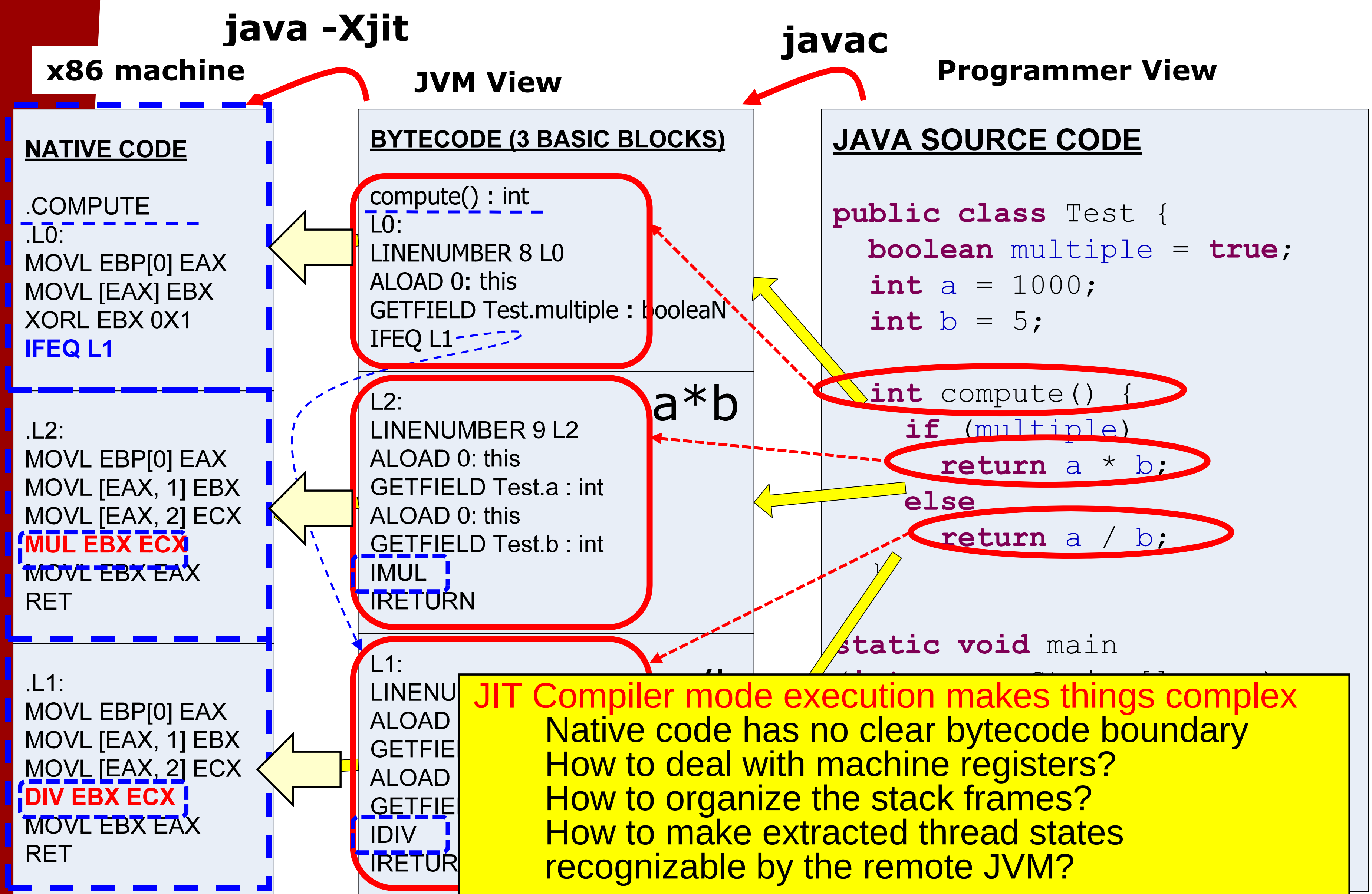
When a write becomes visible to another thread ? How ?

Solution: Global Object Space (GOS)

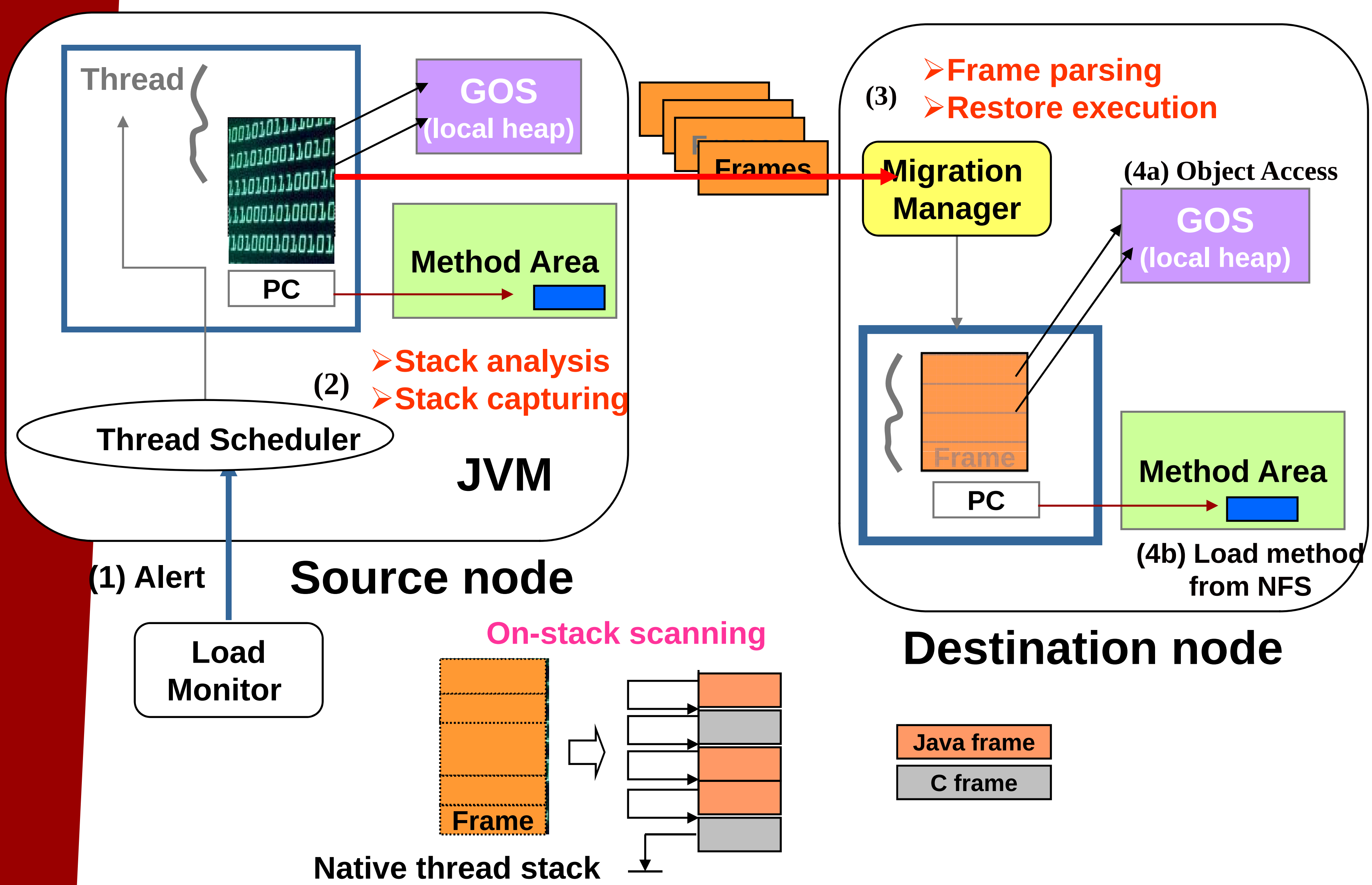
- **Per-object granularity, no false sharing**
- ***Home-based Lazy Release Consistency* (HLRC)**
  - **Home-based variant of LRC: always fetch latest object/page from its home**
  - **No traffic if object unchanged**
  - ***Object home migration*: better locality**
- **Connectivity-based object prefetching: more accurate**



Problem 2: Thread migration under JITC Mode



Thread Migration in JIT Compiler Mode



Thread Migration in JIT Compiler Mode

- Dynamic Native Code Instrumentation
  - **Migration points selection**
    - **Delayed to the head of loop basic block or method**
  - **Register context handler**
    - **Spill dirty registers at migration point without invalidation so that native codes can continue the use of registers**
    - **Use register recovering stub at restoring phase**
  - **Variable type deduction**
    - **Spill type in stacks using compression**
  - **Java frames linking**
    - **Discover consecutive Java frames**

Problem 3: Improve Locality

- Remote memory access is the scalability killer!
- Remote >> local latency (assume in 50-60ns)
  - Infiniband cluster (1-2μs): 20 x slower!
  - Ethernet cluster (100μs): 2,000 x slower!!
  - Grid/Internet (av. 500ms): 10,000,000 x slower!!!

- **"To speed up" ≈ "Reduce as much remote access as possible"**
- **The key is to improve locality**

Solution: Profile-Guided PGAS (PG²AS)

- ***Profile-Guided PGAS (PG²AS)***
  - **A built-in runtime profiler instead of humans for digging out the locality hints**
- **Profile-guided adaptive locality management**
  - **Thread migration**
  - **Object home migration**
  - **Object prefetching**
- **Challenges:**
  - **How does the runtime know which threads to migrate can make the most locality benefit?**
  - **Difficult to decide if no global inter-thread sharing information**
- **Solution: Track sharing % threads**
  - **T1 accesses O1, O3, O5, …**
  - **T2 accesses O1, O2, O3, …**
  - **Sharing % T1 & T2: O1, O3**

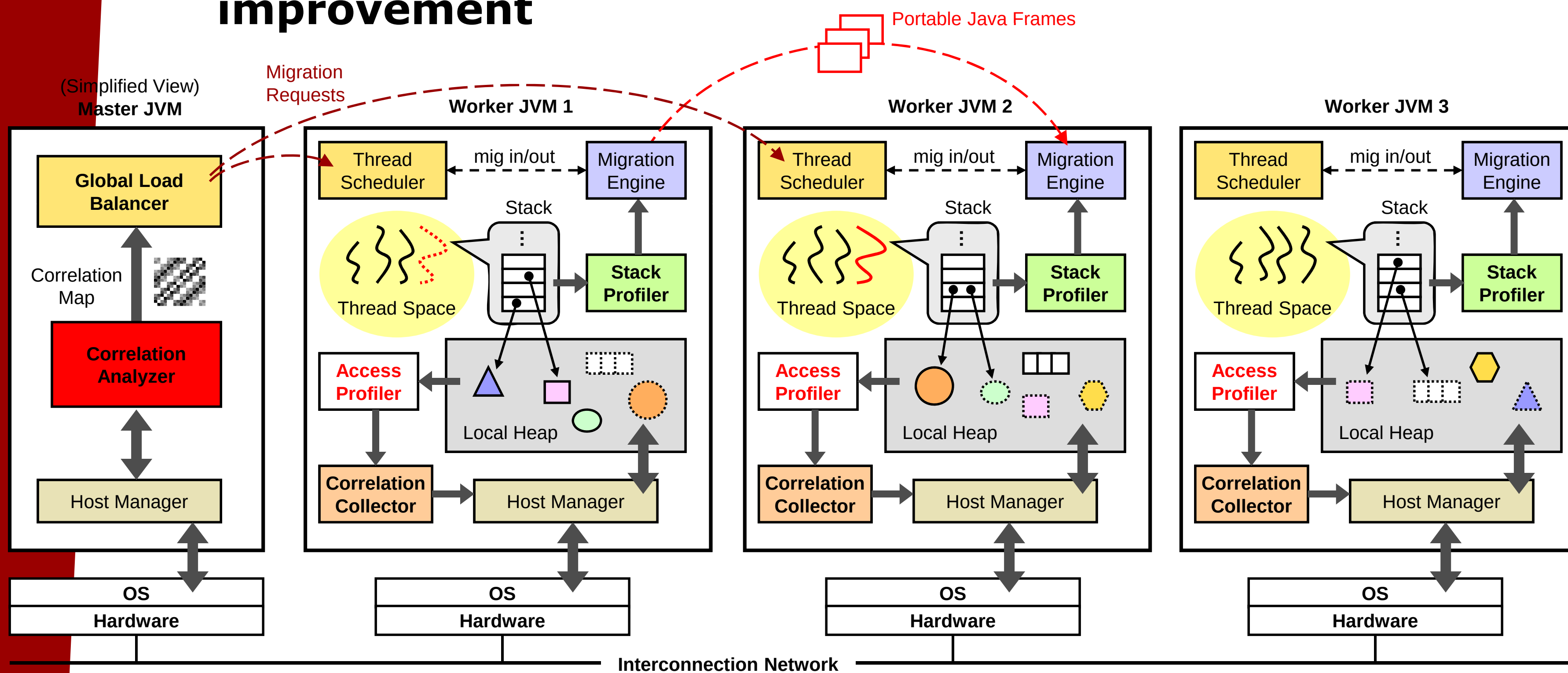
Outline

1. Era of Petaflop Computing
2. PGAS Programing Language
3. Distributed Java Virtual Machine
4. Profile-guided locality management
5. Performance Evaluation

33

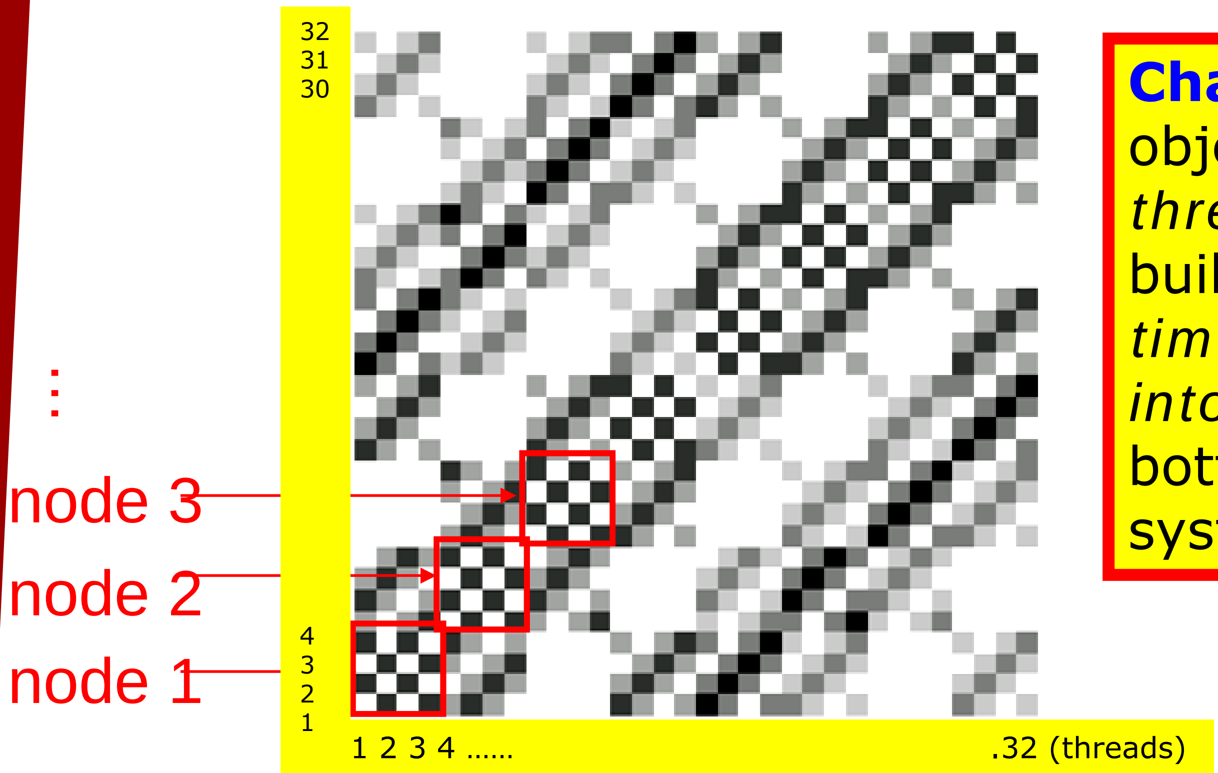
PG-JESSICA: Profile-Guided Version

- ***Access profiler***: **track object access over heap to deduce inter-thread sharing -> *thread-thread relation***
- ***Stack profiler***: **track the set of frequent objects accessed by each thread -> *thread migration cost***
- ***Correlation analyzer***: **profile-guided decisions on dynamic thread migration -> global locality improvement**



Thread Correlation Map (TCM)

- **Thitikamol and Keleher; D-CVM (1999)**
  - **Proposed "Active Correlation Tracking" (Page)**
  - **Thread Correlation Map (TCM): a 2D histogram of shared data volume between each pair of threads.**
    - **Grayscale(x,y) = sharing amount of thread x and y**
    - **TCM(1,1) = TCM(2,2) = TCM(3,3) = … = 0**

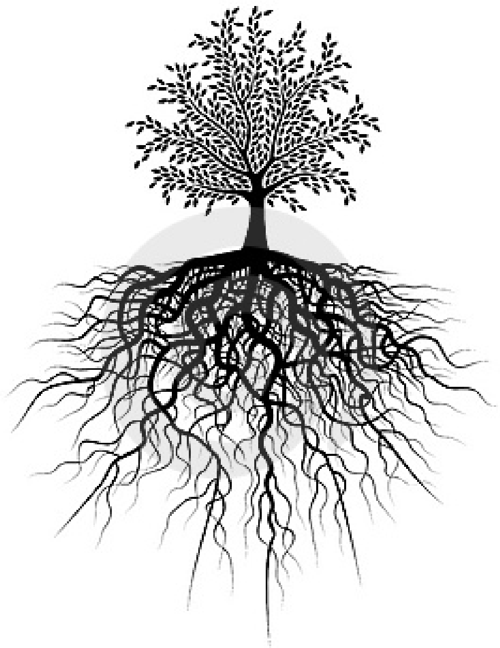


Water-Spatial (32 threads placed on 8 nodes)

Challenge: Given *M* objects shared by *N threads, TCM* building take $O(MN^2)$ *time. M can grow into a scalability* bottleneck in the system.

"Sticky Set"

- ***Sticky Set (SS) :*** **a subset of working set of a thread, includes only those frequently used objects.**
  - **"Sticky" : if the thread is migrated, objects in SS are more likely to be fetched again.**
  - **SS should be detected and moved along with the thread to save most object misses after migration.**



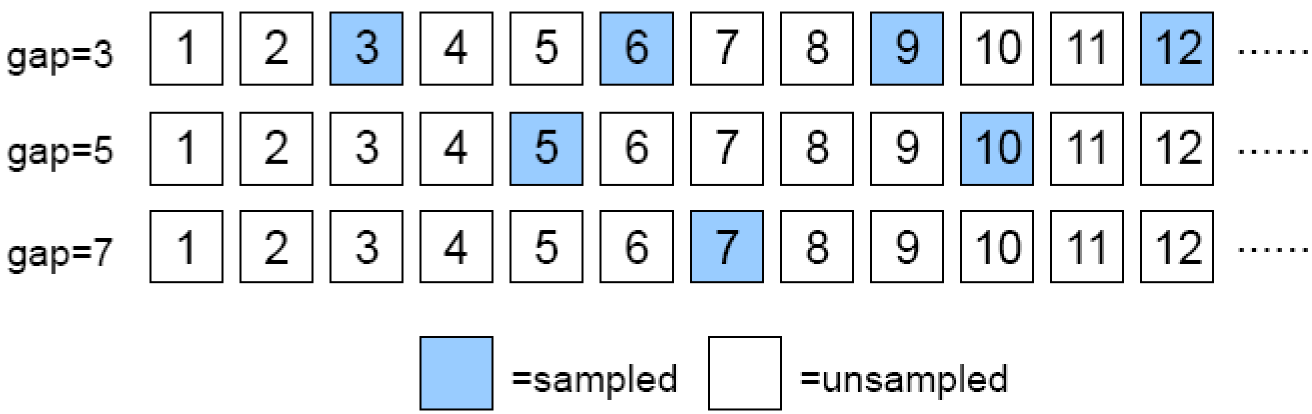
Summary of Our Solution

- **What we want to do:**
  1. **Model thread sharing (inter-thread correlation)**
  2. **Model indirect thread migration cost**
- **Profiling results:**
  1. **Thread Correlation Map (TCM)**
  2. **Per-thread Sticky Set (SS)**
- **Use both to design new migration policy**
  1. **Correlation-driven**
  2. **Cost-aware**
- **How we profile them efficiently?**
  1. **Adaptive object sampling → TCM**
  2. **Adaptive stack sampling → SS**

Details : King Tin Lam, Yang Luo, Cho-Li Wang, "Adaptive Sampling-Based Profiling Techniques for Optimizing the Distributed JVM Runtime," 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS2010), April 19-23, ATLANTA, USA

Adaptive Object Sampling (AOS)

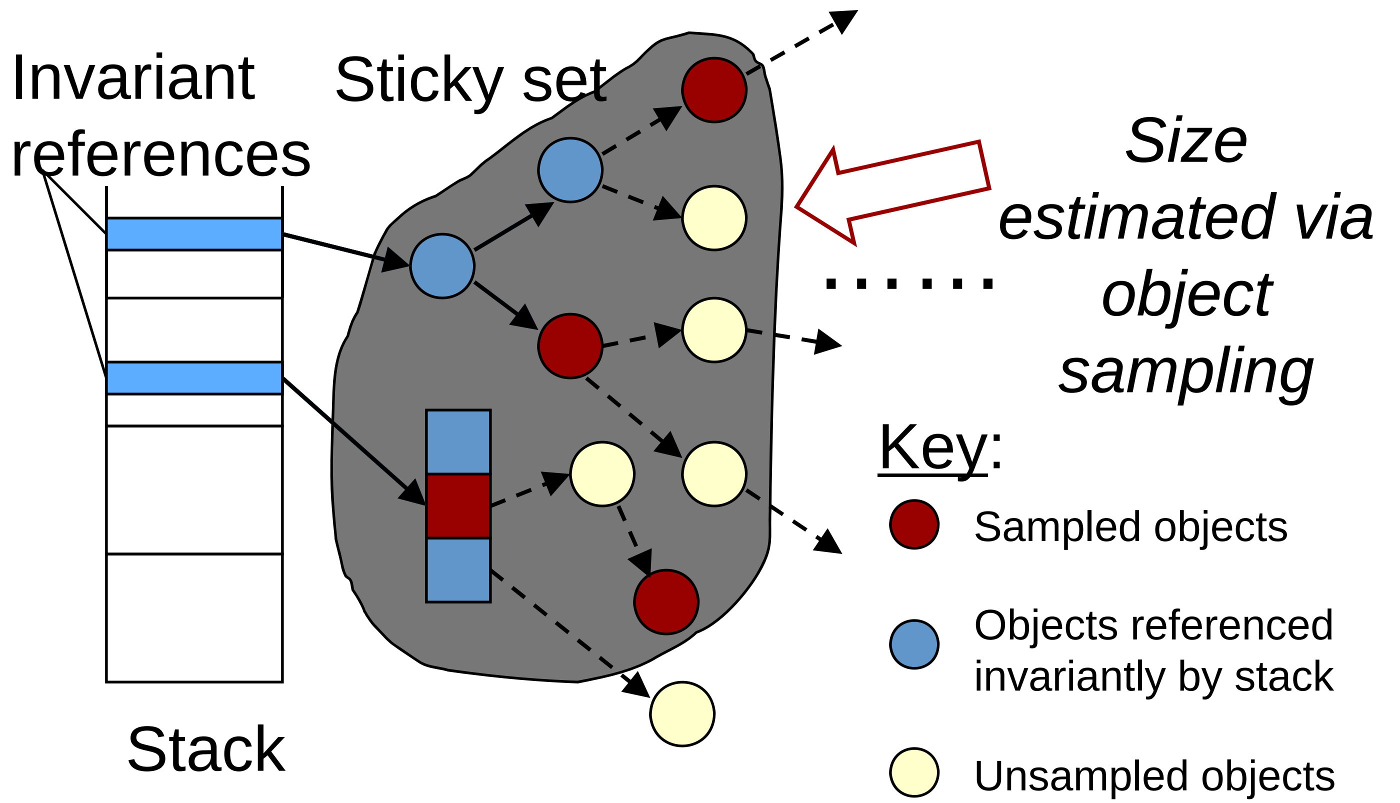
- Each object has a "**sequence number**", unique among objects within the same class.
- Sample the object if sequence # is divisible by the current "**sampling gap**" (selected and changed at runtime to strike a balance of cost and accuracy)
- Sampling rate:
  - 1X = sample 1 object per page of heap
  - 1024X means "full sampling"
  - For a class of size s, *sampling at* rate nX, sampling gap = $S_p / (s \times n)$, where S_p is the page size (usually 4KB).



Stack Invariants

- **JVM is a "stack machine"**
  - **Stack variables can be hint of constantly accessed objects**
  - ***Stack invariants :*** **Those references constantly stay in the stack across snapshots taken. Good hints of SS.**
  - **Usually stack invariants are the entry points of SS and important data structures like Hashmap, TreeMap, Linked List**

Stack Invariants (Cont')



Adaptive Stack Sampling: Adjustable timer controlling which period of time to do stack sampling. Stack frame added with "visited" flag. If not touched across two sampling rounds, no need to sample it.

Outline

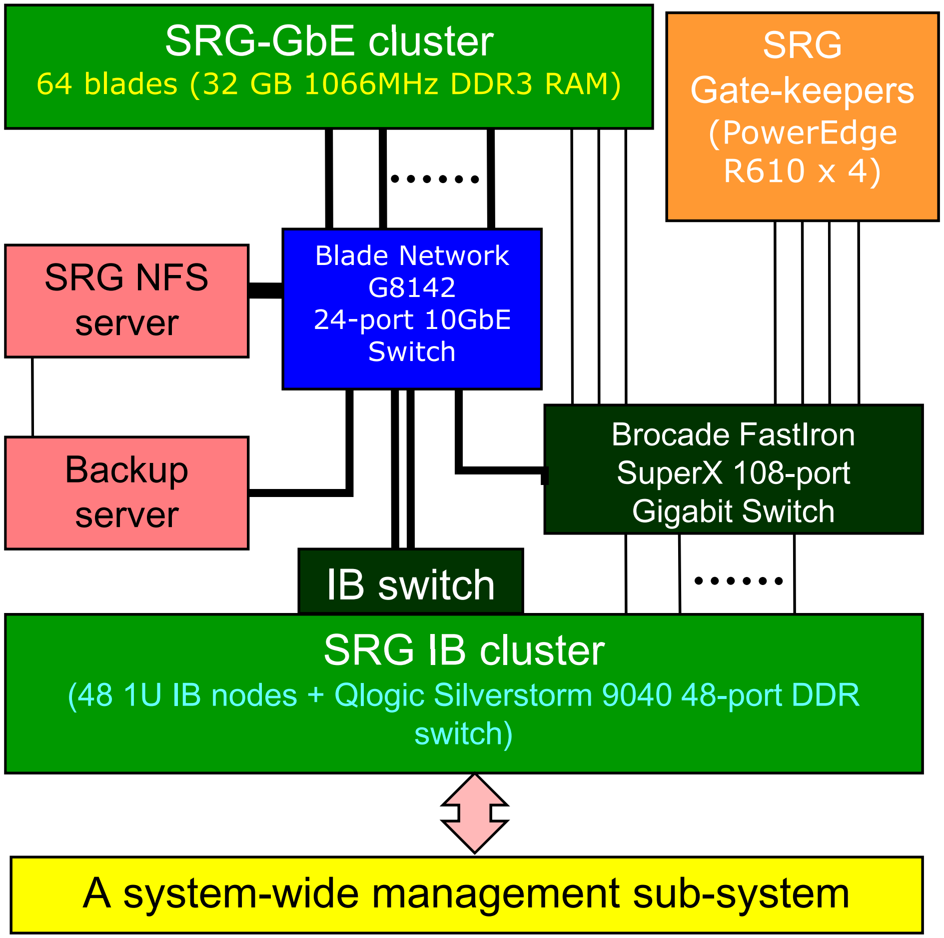
1. Era of Petaflop Computing
2. PGAS Programing Language
3. Distributed Java Virtual Machine
4. Profile-guided locality management
5. Performance Evaluation

41

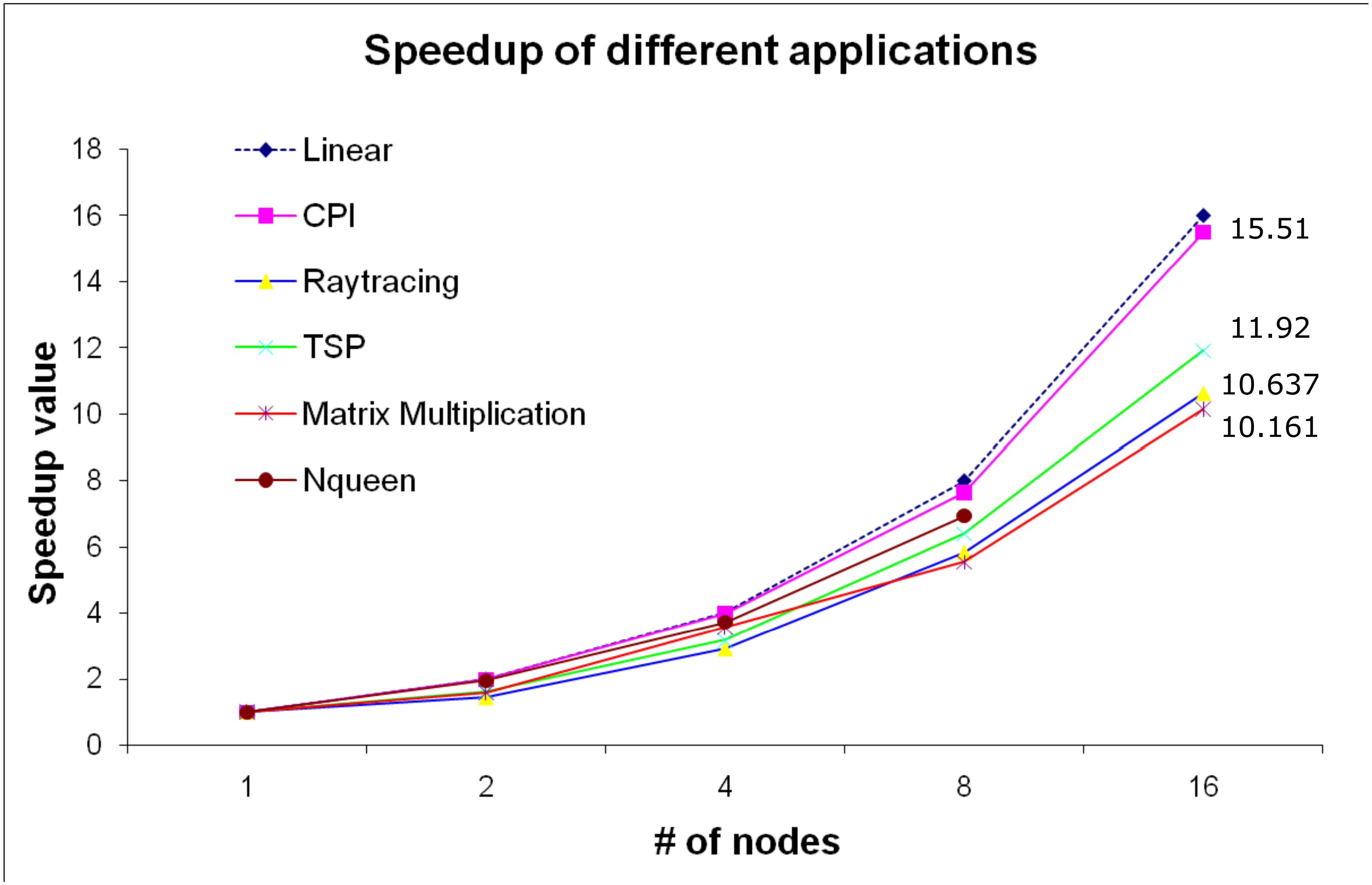
Testing Environment: HKU Gideon-II Cluster

- **240 SMP blade servers (19.43 TFlop/s)**
  - **Expected to grow to 25+ TFlop/s upon Phase 2's completion in late 2010.**
- **Node configuration : Dell PowerEdge R610/M610**
  - **2 x Intel Nehalem-based Quad-core Xeon 2.53GHz**
  - **32 GB 1066MHz DDR3 RAM and SAS disks**
- **Networking:**
  - **4X DDR Infiniband (20 Gbit/s): 80 nodes (not used)**
  - **Gigabit Ethernet (1 Gbit/s): 160 nodes**
  - **OS: RedHat Enterprise Linux, Scientific Linux, Fedora Linux.**
- **Production run in September, 2009**

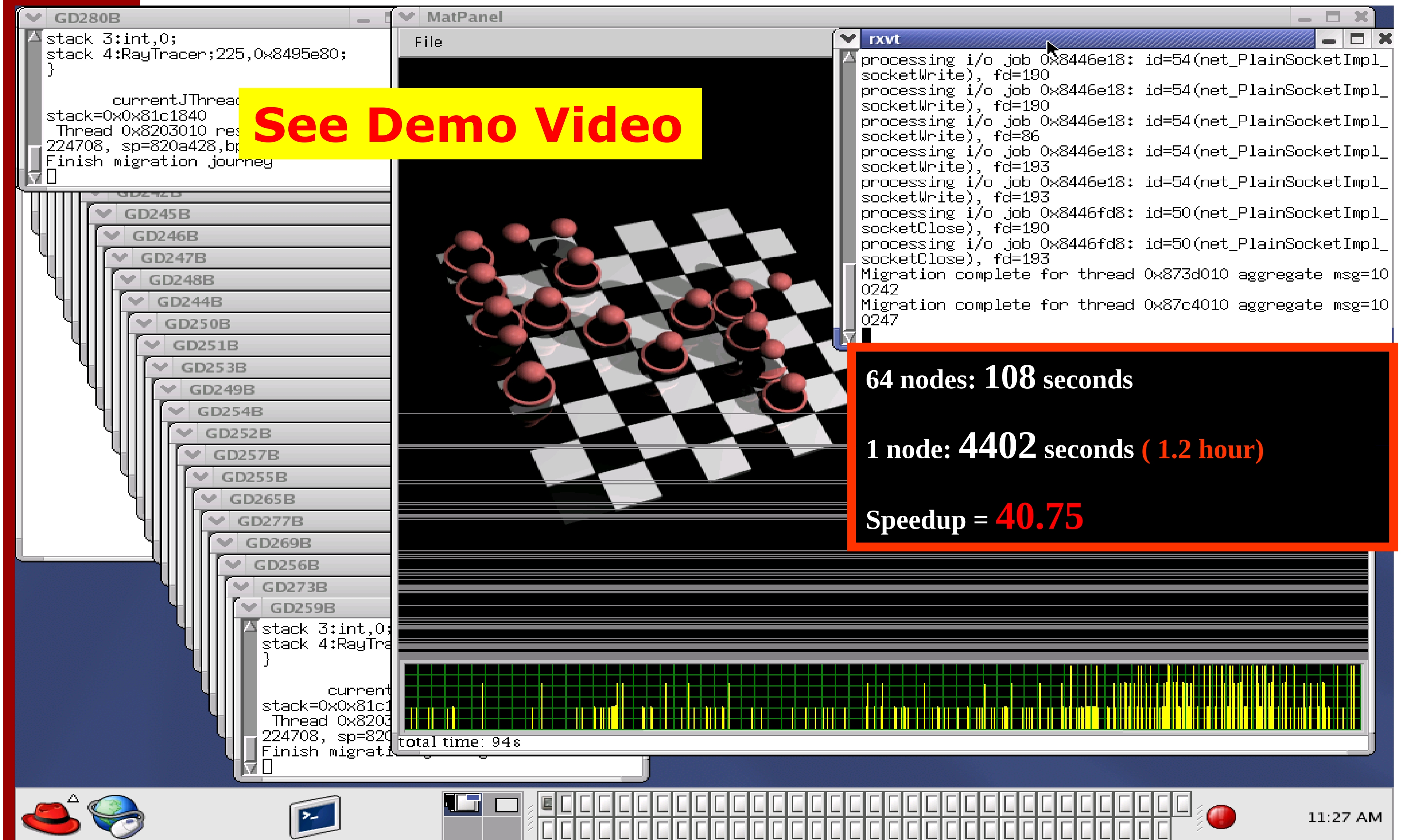
Computer Science (Systems Research Group)



Speedup of JAVA applications on JESSICA2



Ray Tracing on JESSICA2 (64 PCs)



See Demo Video

64 nodes: 108 seconds

1 node: 4402 seconds (1.2 hour)

Speedup = 40.75

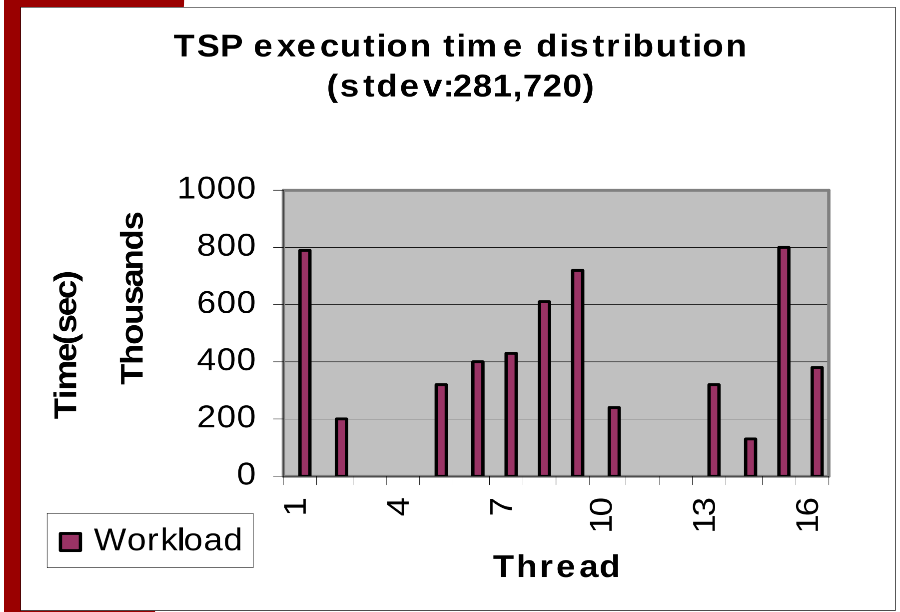
Dynamic Native Code Instrumentation

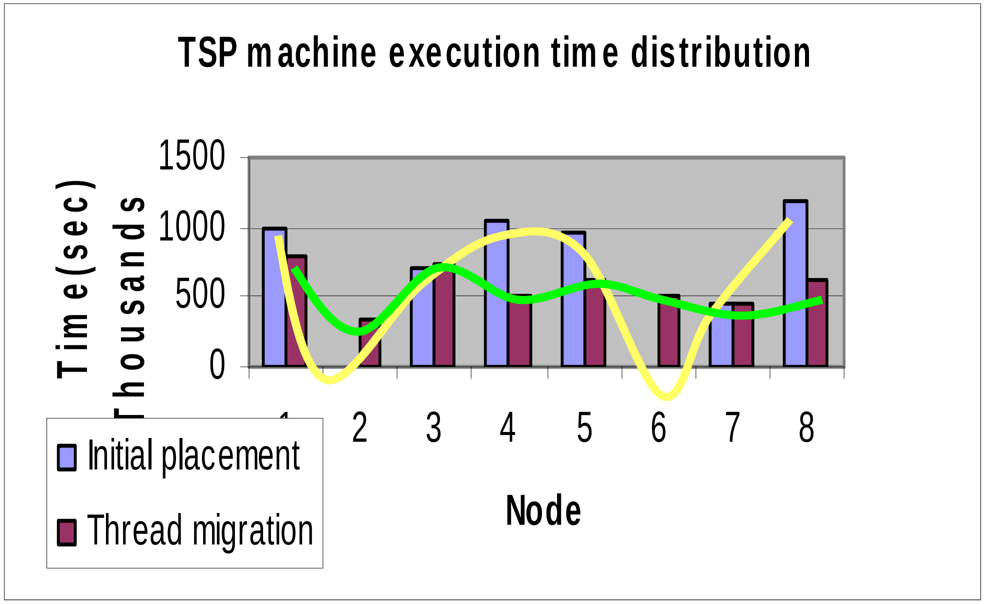
Time and space Overhead Analysis

Benchmarks	Time (seconds)		Space(native code/bytecode)	
	No migration	Migration	No migration	Migration
compress	11.31	11.39(+0.71%)	6.89	7.58(+10.01%)
jess	30.48	30.96(+1.57%)	6.82	8.34(+22.29%)
raytrace	24.47	24.68(+0.86%)	7.47	8.49(+13.65%)
db	35.49	36.69(+3.38%)	7.01	7.63(+8.84%)
javac	38.66	40.96(+5.95%)	6.74	8.72(+29.38%)
mpegaudio	28.07	29.28(+4.31%)	7.97	8.53(+7.03%)
mtrt	24.91	25.05(+0.56%)	7.47	8.49(+13.65%)
jack	37.78	37.90(+0.32%)	6.95	8.38(+20.58%)
Average		(+2.21%)		(+15.68%)

(Gideon-I)

Thread migration for irregular applications (1) : TSP





8 nodes, 16 threads, TSP 13 cities, (object sharing: shortest path)		
	Initial placement	Thread migration (5 times)
Time (sec)	**1203.10**	**793.317** (–33.6%)
Stdev	438,444.1	152,463.1

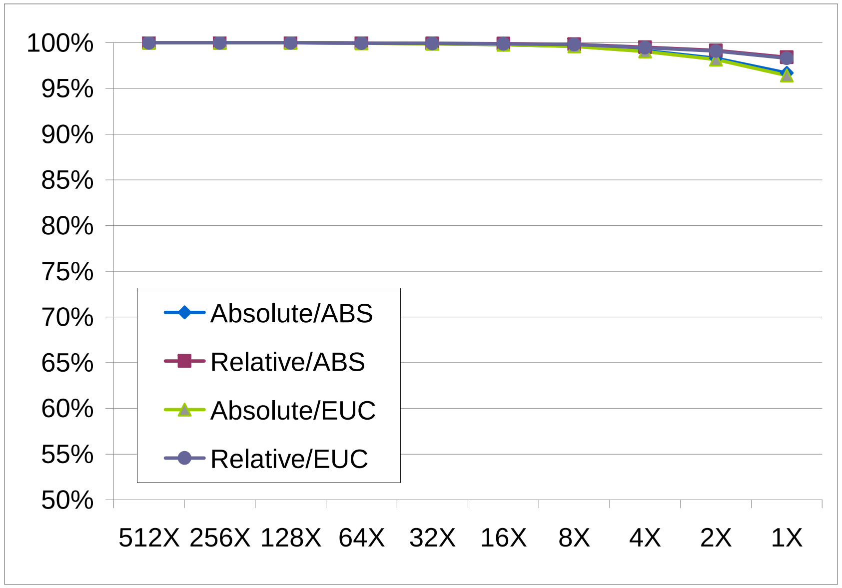
(Gideon-I)

Stack Profiling Overhead

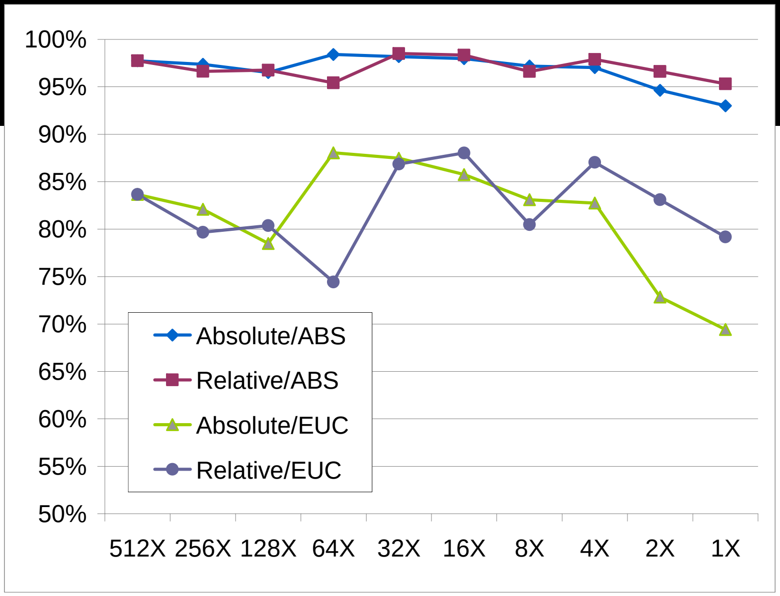
- **Timer-based control of stack sampling phases saves over half of overheads**
- **Lazy extraction saves up to 1/3 overheads**

Bench mark	Data Set Size	Baseline Exe Time	+ Stack Sampling Overhead				+ Sticky-set Footprinting Overhead				+ Sticky-set Resolution Overhead
			Immediate Extraction		*Lazy Extraction*		*Nonstop*		*Timer-based (100ms)*		
			4ms	*16ms*	*4ms*	*16ms*	*4X*	*Full*	*4X*	*Full*	
SOR	1K×1K	6201	6216 (0.24%)	6207 (0.10%)	6211 (0.17%)	6206 (0.08%)	6714 (8.28%)	6707 (8.17%)	6519 (5.13%)	6480 (4.50%)	6639 (1.85%)
Barnes -Hut	4K	93857	94947 (1.16%)	94657 (0.85%)	94697 (0.89%)	95209 (1.44%)	98968 (5.45%)	102190 (8.88%)	93649 (-0.22%)	102334 (9.03%)	97585 (4.20%)
Water-Spatial	512	59105	59232 (0.21%)	59161 (0.09%)	59209 (0.17%)	59124 (0.03%)	59834 (1.23%)	61985 (4.87%)	59501 (0.67%)	60313 (2.04%)	60002 (0.84%)

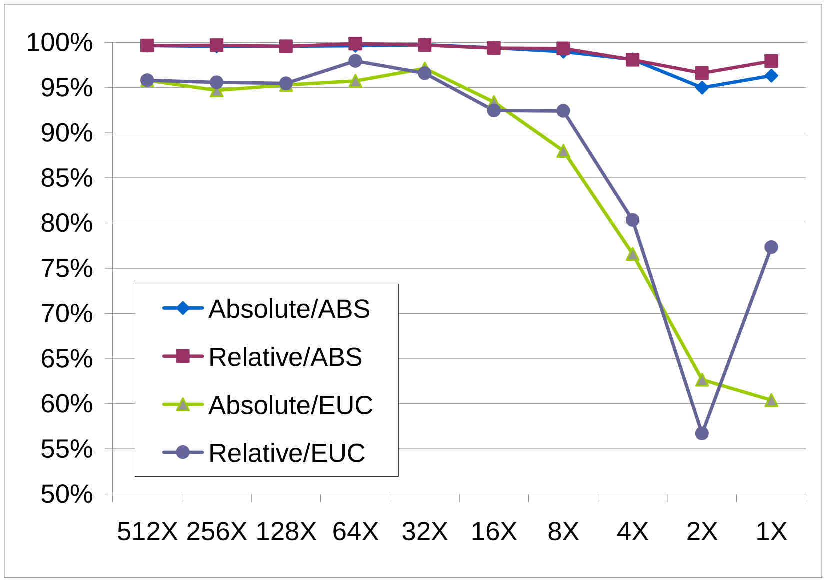
Accuracy of AOS (Cont')



(a) SOR



(c) Water-Spatial



(b) Barnes-Hut

(Euclidean distance)

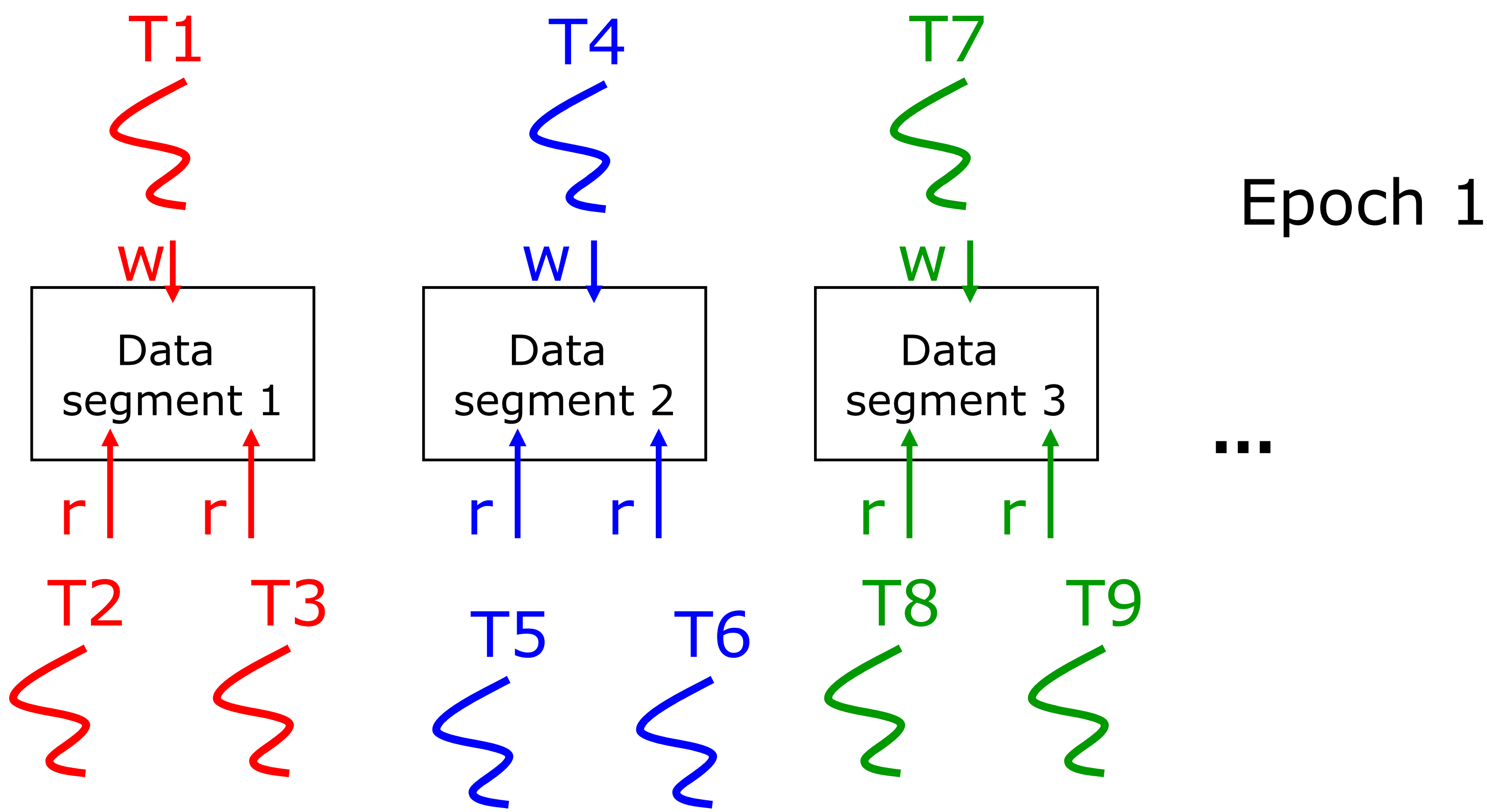
$$E_{EUC} = \frac{\sqrt{\sum_{i=1}^{N} \sum_{j=1}^{N} (a_{ij} - b_{ij})^2}}{\sqrt{\sum_{i=1}^{N} \sum_{j=1}^{N} (b_{ij})^2}}$$

(Absolute distance)

$$E_{ABS} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} \left| a_{ij} - b_{ij} \right|}{\sum_{i=1}^{N} \sum_{j=1}^{N} \left| b_{ij} \right|}$$

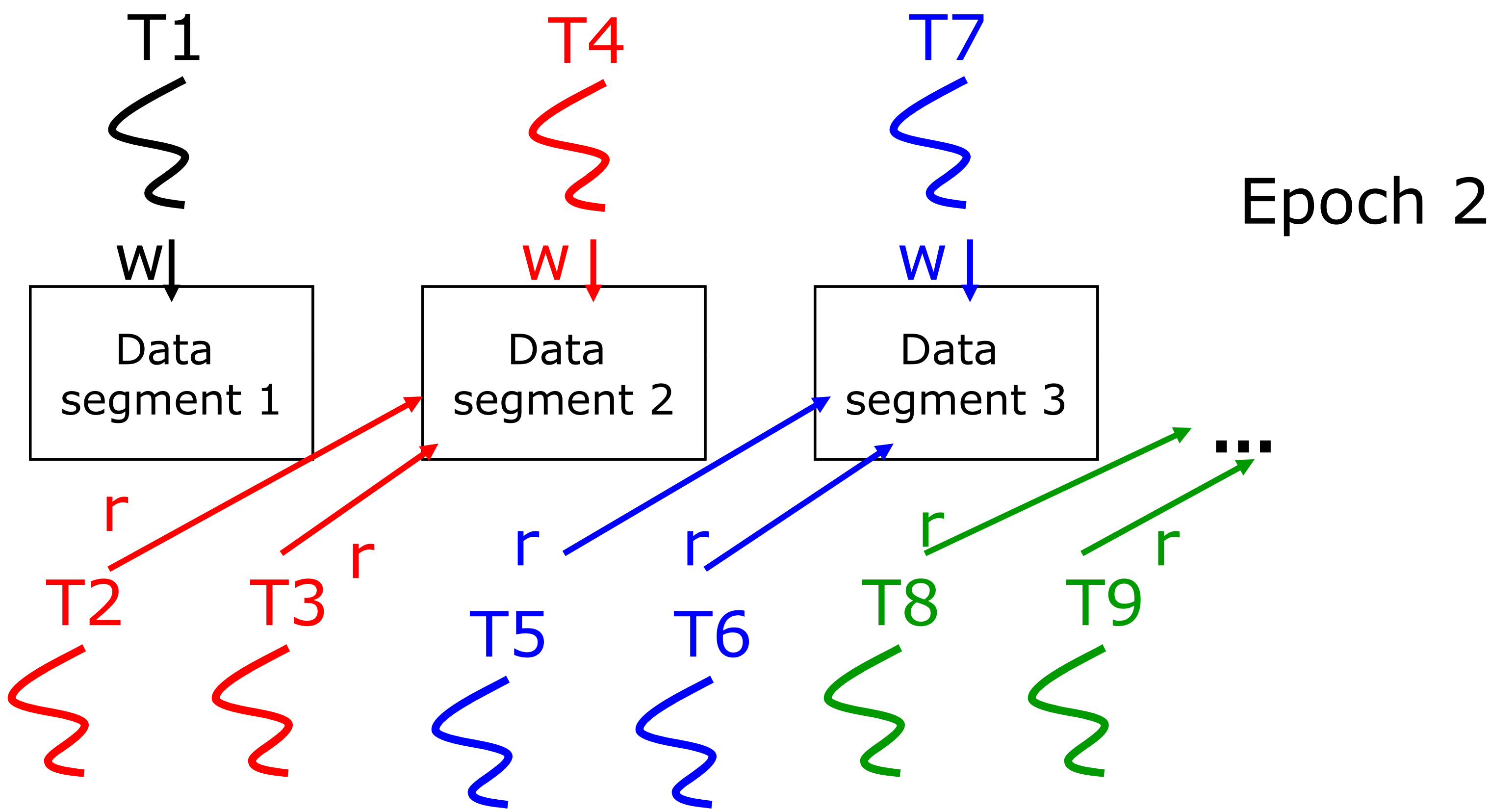
Profile-Guided Thread Migration

- **We assess this using a CRM application "Customer Analytics" with dynamic change in sharing patterns.**



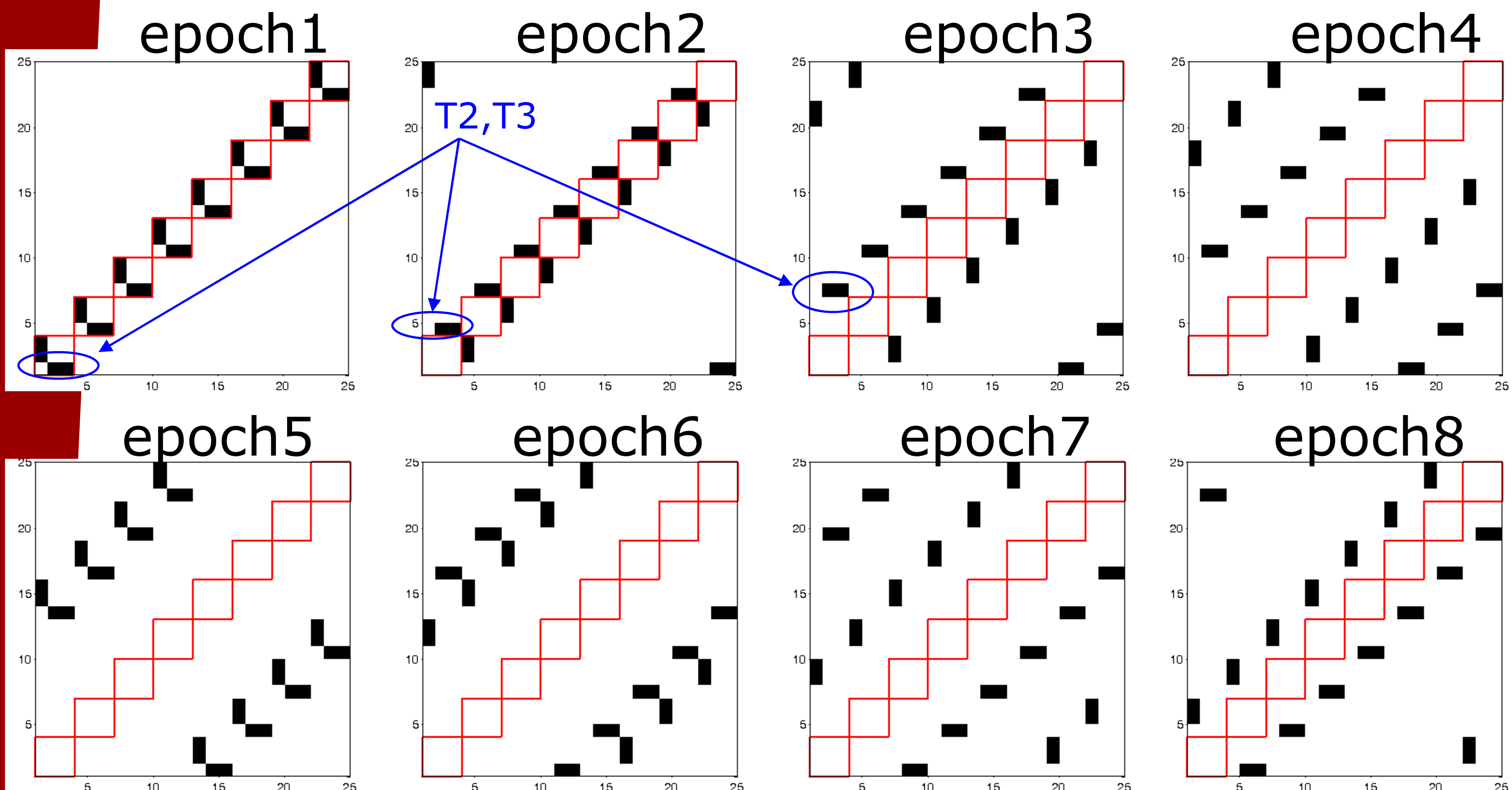
Effect of Profile-Guided Thread Migration

- **We assess this using a CRM application "Customer Analytics" with dynamic change in sharing patterns.**



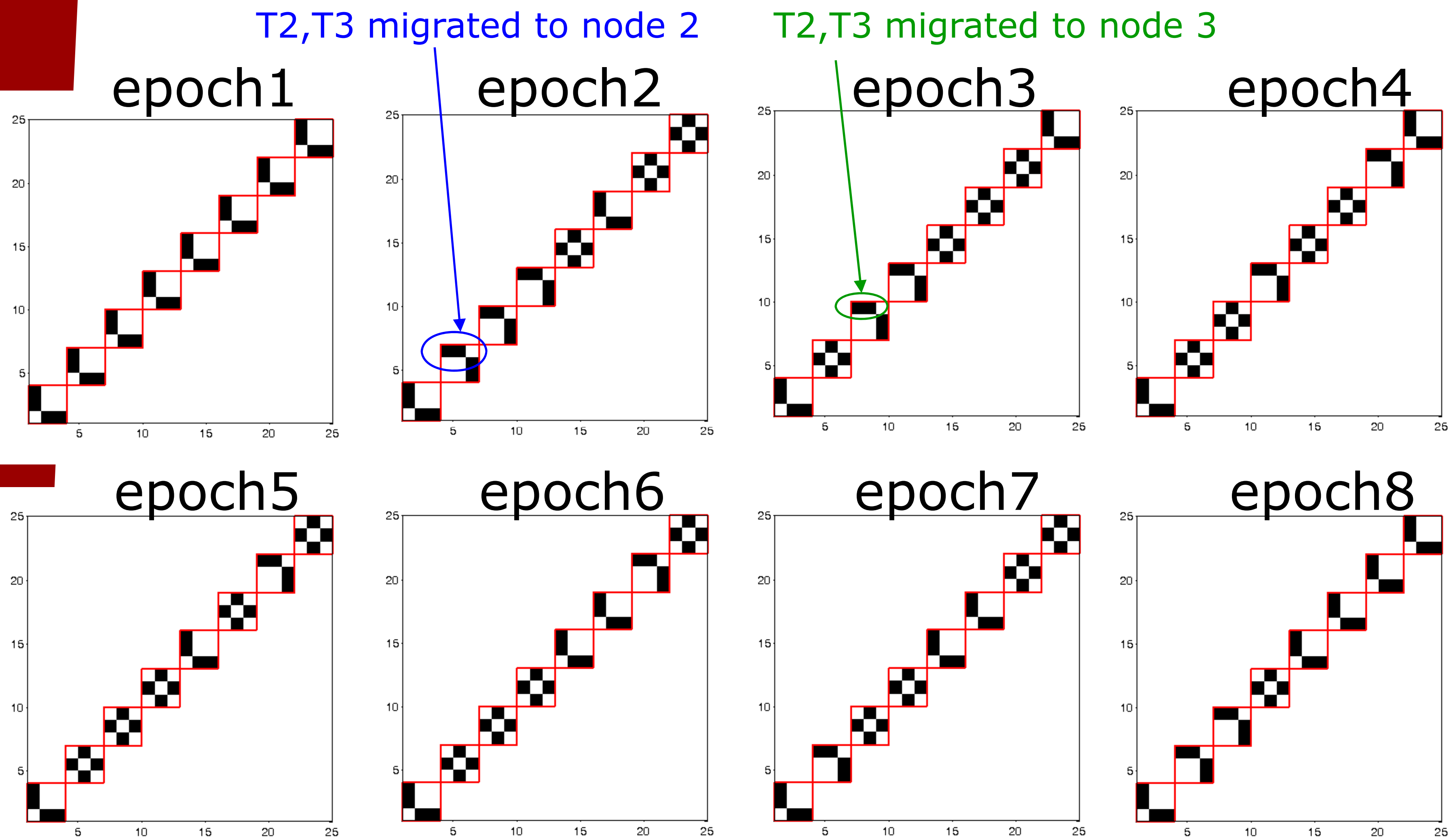
Effect of Profile-Guided Thread Migration

- **Without thread migration, locality is not preserved (out of red boxes denoting node boundaries) as time goes by.**

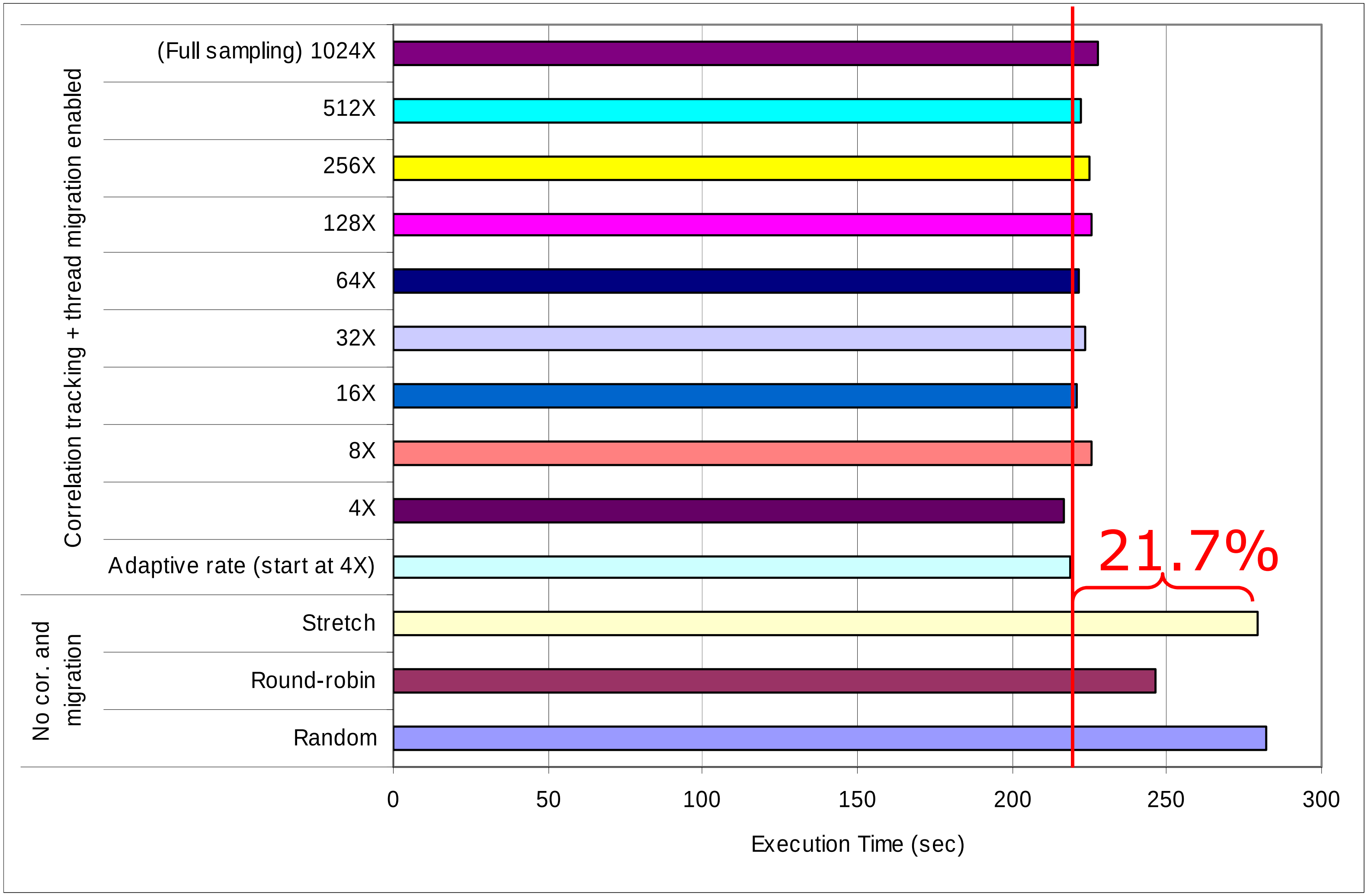


Effect of Profile-Guided Thread Migration

- **With correlation-driven thread migration**



Performance Gain



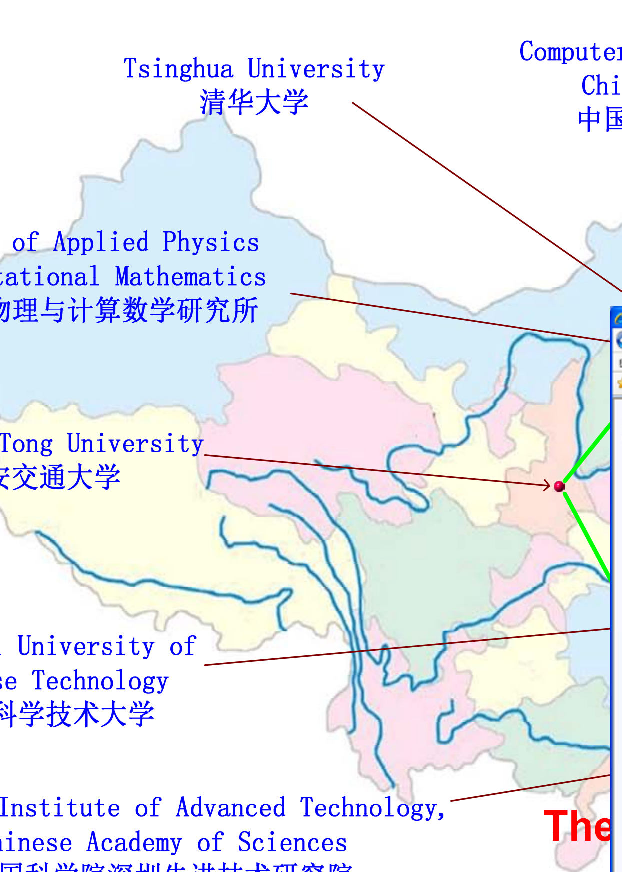
Conclusion

- **Distributed Java Virtual Machine can provide a high-performance platform for running multithreaded Java applications on clusters**
- **Java thread migration helps to improve the performance, flexibility, and scalability of DJVM**
- **A couple of advanced profiling strategies for optimizing locality**
  - **Adaptive object sampling**
  - **Online stack sampling**
- **Towards PGAS Parallel Programming – why not JESSICA ("Easy-to-use")**

JESSICA Launched to CNGrid HKU Portal

China National Grid
香港大学网格节点

国家863计划



Tsinghua University
清华大学

Computer Network Information Center
Chinese Academy of Sciences
中国科学院计算机网络信息中心

Shandong University
山东大学

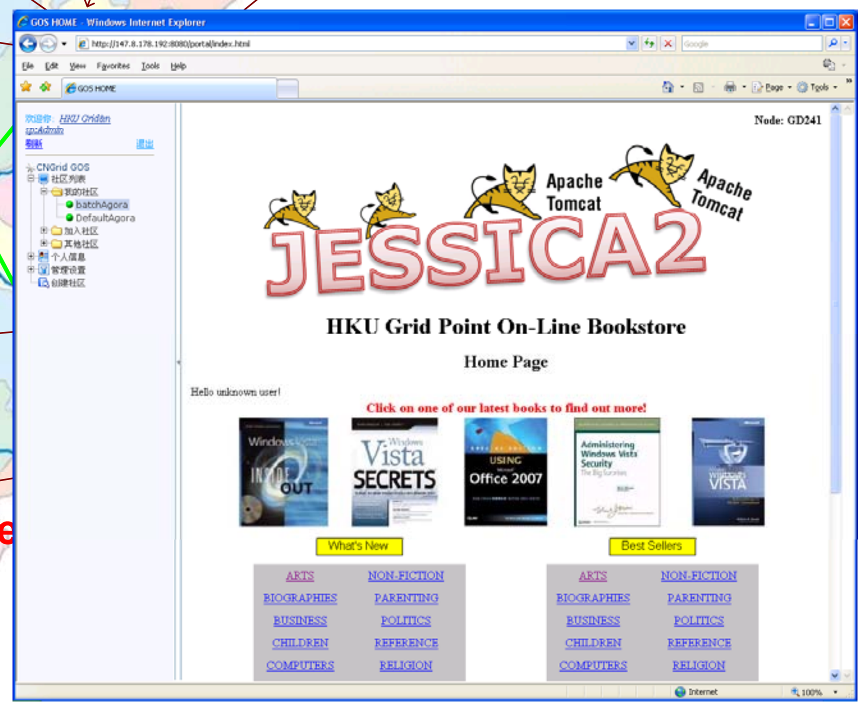
Institute of Applied Physics
and Computational Mathematics
北京应用物理与计算数学研究所

Xian JiaoTong University
西安交通大学

National University of
Defense Technology
国防科学技术大学

ShenZhen Institute of Advanced Technology,
Chinese Academy of Sciences
中国科学院深圳先进技术研究院

The



Thanks!

For more information:

JESSICA2 Project

http://www.cs.hku.hk/~clwang/projects/JESSICA2.html

C.L. Wang's webpage:

http://www.cs.hku.hk/~clwang/