JiST – Java in Simulation Time

An efficient, unifying approach to simulation using virtual machines

Rimon Barr, Zygmunt Haas, Robbert van Renesse

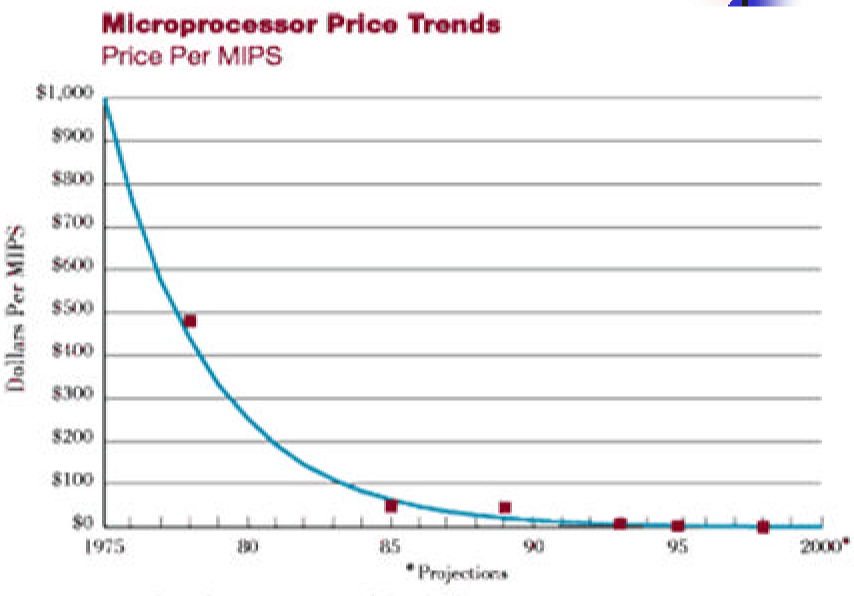
rimon@acm.org haas@ece.cornell.edu rvr@cs.cornell.edu

Cornell University, USA

8 September 2004

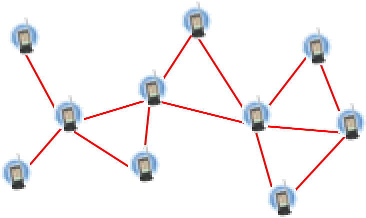
http://jist.ece.cornell.edu/

motivation: simulation



- cost per MIPS declining
  - e.g. Pentium Xeon:
  - ~10,000 MIPS @ ~$200
- emphasis on computation
  - vs. analytical methods
  - vs. empirical methods
  - simulators are needed
- e.g., wireless networks
- published ad hoc network simulations
  - lack network size - ~500 nodes; or
  - compromise detail - packet level; or
  - curtail duration - few minutes; or
  - are of sparse density - $<10/km^2$

  i.e. limited simulation scalability [Riley02]



Simulation scalability is important

what is a simulation?

- unstructured simulation: computers compute
- time structured: event-oriented vs. process-oriented

- discrete event simulator is a program that:
  - encodes the simulation model
  - stores the state of the simulated world
  - performs events at discrete simulation times
  - loops through a temporally ordered event queue
  - works through simulation time as quickly as possible

- desirable properties of a simulator:

• correctness	- valid simulation results
• efficiency	- performance in terms of throughput and memory
• transparency	- separate correctness from efficiency:
	- write “simple” program in a standard language
	- provide implicit optimization, concurrency, distribution, portability, etc.

how do we build simulators?

systems

- **simulation kernels**
  - control scheduling, IPC, clock
  - processes run in virtual time
  - e.g. TimeWarp OS [Jefferson87], Warped [Martin96]

  transparency efficiency

- **simulation libraries**
  - move functionality to user-space for performance; monolithic prog.
  - usually event-oriented
  - e.g. Yansl [Joines94], Compose [Martin95], ns2 [McCanne95]

  transparency efficiency

languages

- generic **simulation languages**
  - introduce entities, messages and simulation time semantics
  - event and state constraints allow optimization
  - both event and process oriented
  - e.g. Simula [Dahl66], Parsec [Bagrodia98] / GloMoSim [Zeng98]
- application-specific languages
  - e.g. Apostle [Bruce97], TeD [Perumalla98]

  transparency efficiency

  new language

virtual machines

virtual machine-based simulation

- **Proposal:**

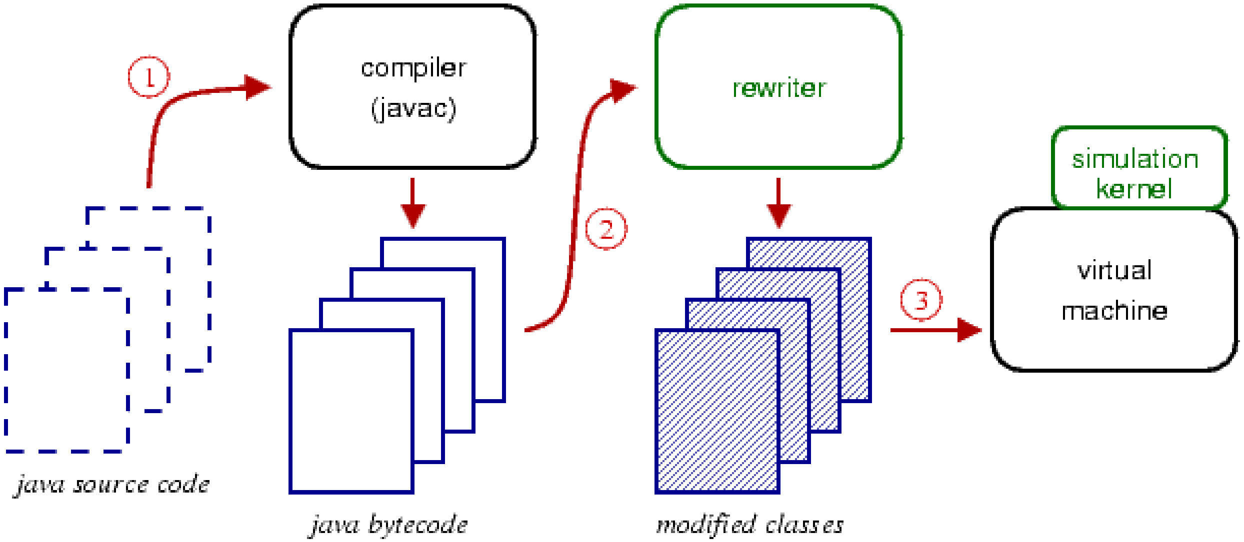
A virtual machine-based simulator benefits from the advantages of both the traditional systems and language-based designs by leveraging standard compilers and language runtimes as well as ensuring efficient simulation execution through transparent cross-cutting program transformations and optimizations.

- **JiST – Java in Simulation Time**
  - **converts a virtual machine into a simulation platform**
  - **no new language, no new library, no new runtime**
  - **merges modern language and simulation semantics**
    - **combines systems-based and languages-based approaches**
    - **result: virtual machine-based simulation**

	kernel	library	language	**JiST**
transparent	++		++	++
efficient		+	+	++
standard	++	++		++

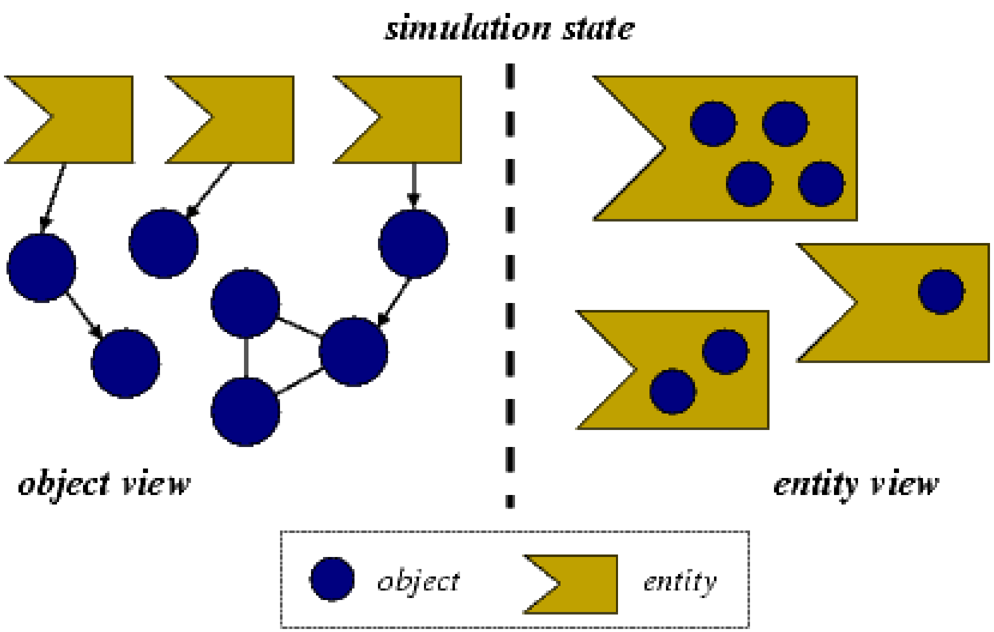
system architecture

1. Compile simulation with standard Java compiler
2. Run simulation within JiST (within Java); simulation classes are dynamically rewritten to introduce simulation time semantics:
   - extend the Java object model and execution model
   - instructions take zero (simulation) time
   - time explicitly advanced by the program: `sleep(time)`
   - progress of time is dependent on program progress
3. Rewritten program interacts with simulation kernel



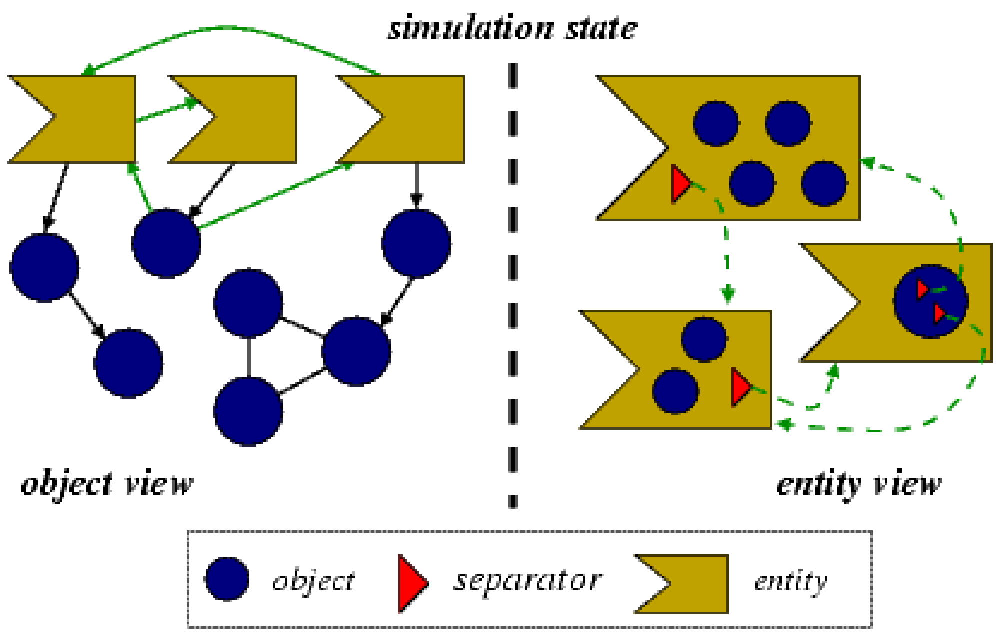
jist object model

- program state contained in objects
- objects contained in entities
  - think of an entity as a simulation component
  - an entity is any class tagged with the `Entity` interface
  - each entity runs at its own simulation time
  - as with objects, entities do not share state
  - akin to JKernel [Hawblitzel98] process in spirit, without the threads!



jist execution model

- entity methods are an event interface
  - simulation time invocation
  - non-blocking; invoked at caller entity time; no continuation
  - like co-routines, but scheduled in simulation time
- entity references replaced with separators
  - event channels; act as state-time boundary
  - demarcate a TimeWarp-like process, but at finer granularity



a basic example

- **the "hello world" of event simulations**

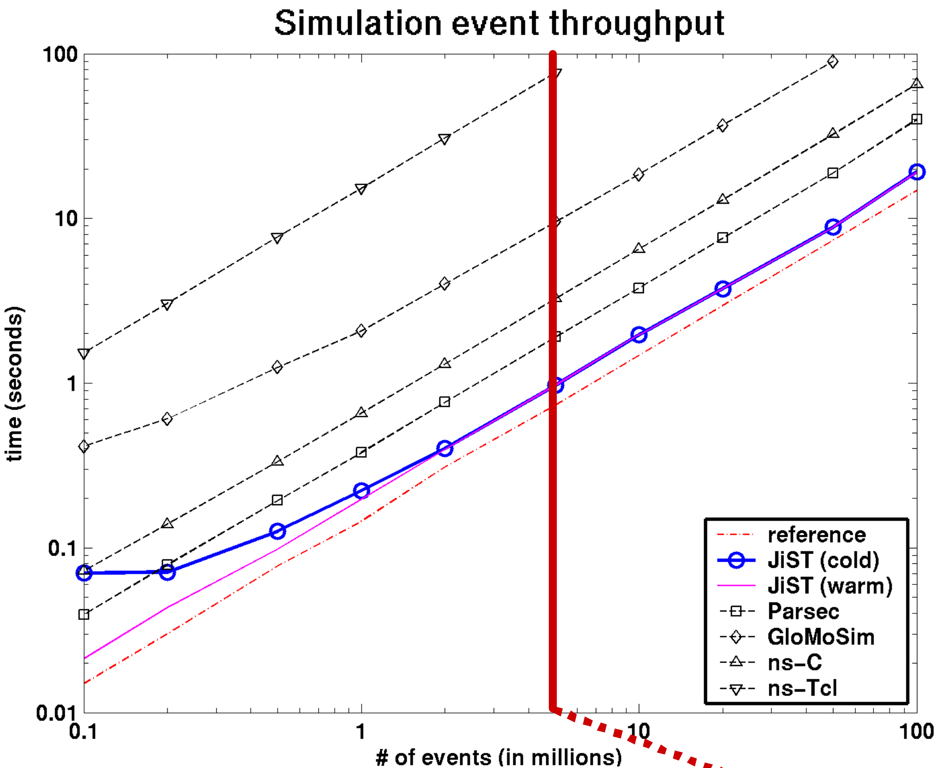
```
class HelloWorld implements JistAPI.Entity
{
  public void hello()
  {
    JistAPI.sleep(1);
    hello();
    System.out.println("hello world, " +
       "time=" + JistAPI.getTime() );
  }
}
```

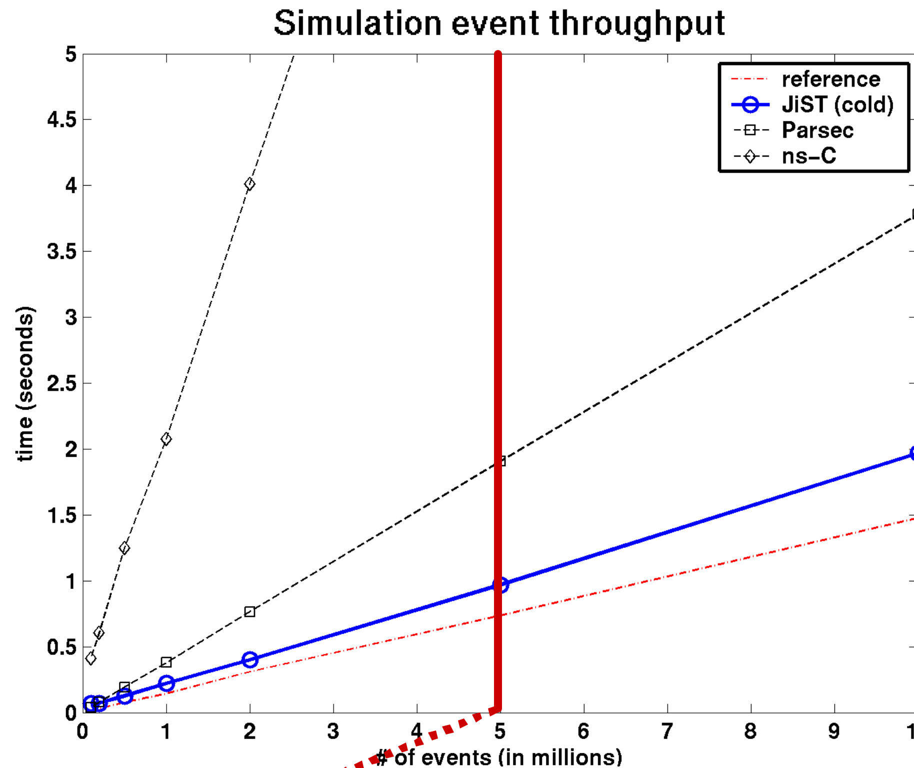
- **demo!**

Java	JiST

`Stack overflow @hello`	`hello world, time=1`
	`hello world, time=2`
	`hello world, time=3`
	`etc.`

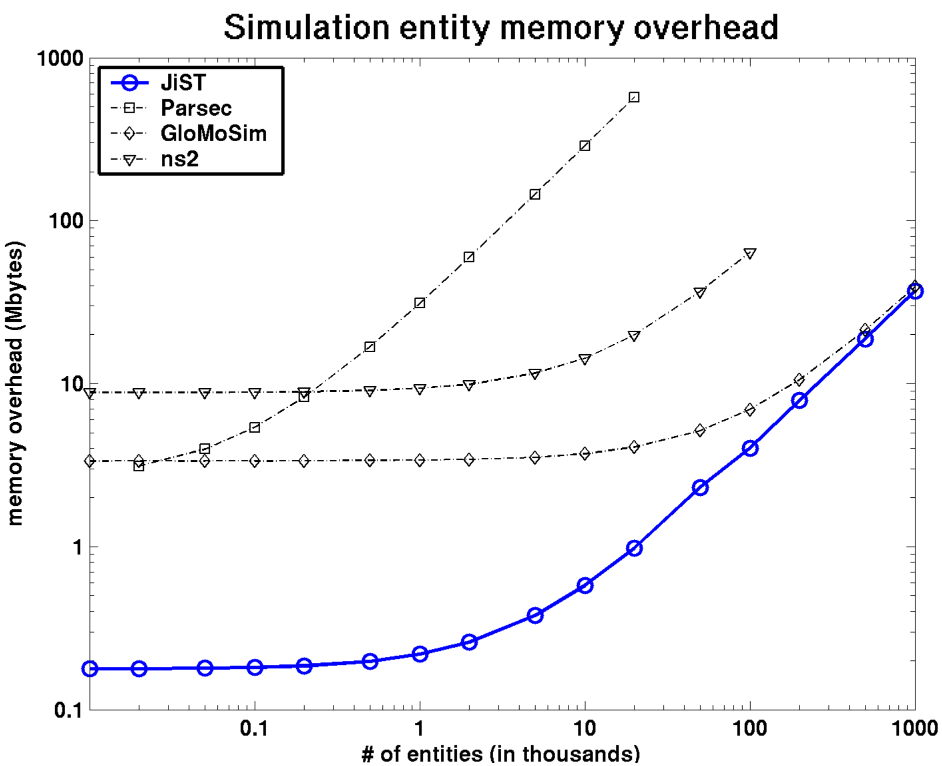
jist micro-benchmark: event throughput

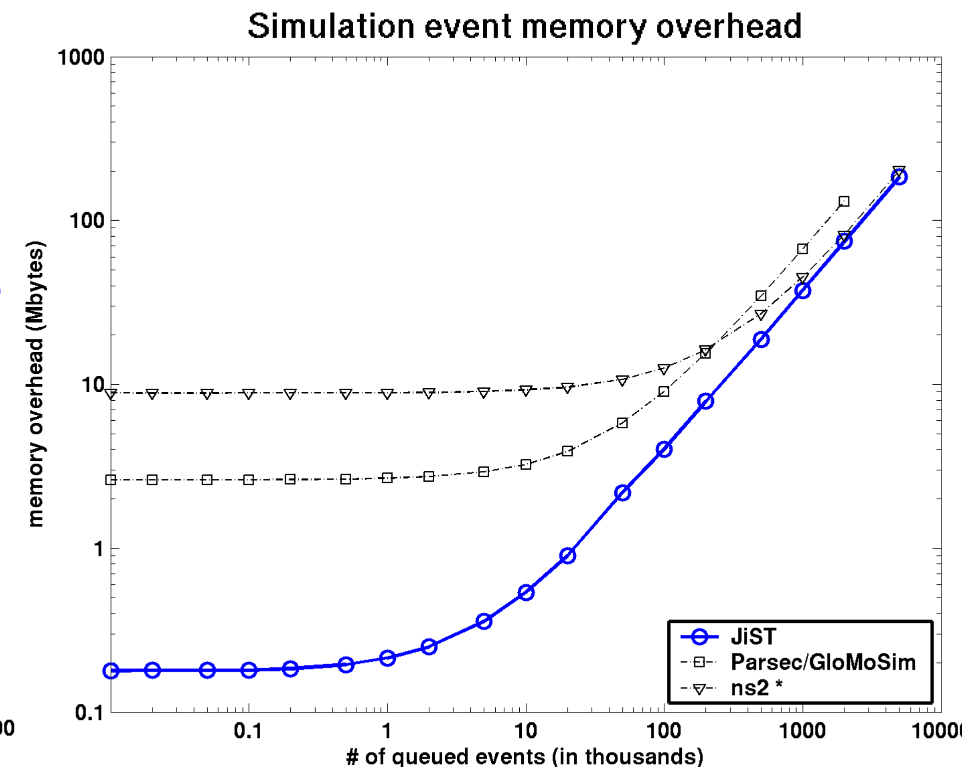




5x10^6 events	time (sec)	vs. reference	vs. JiST
reference	0.74		0.76x
JiST	0.97	1.31x	
Parsec	1.91	2.59x	1.97x
ns2-C	3.26	4.42x	3.36x
GloMoSim	9.54	12.93x	9.84x
ns2-Tcl	76.56	103.81x	78.97x

jist micro-benchmark: memory overhead

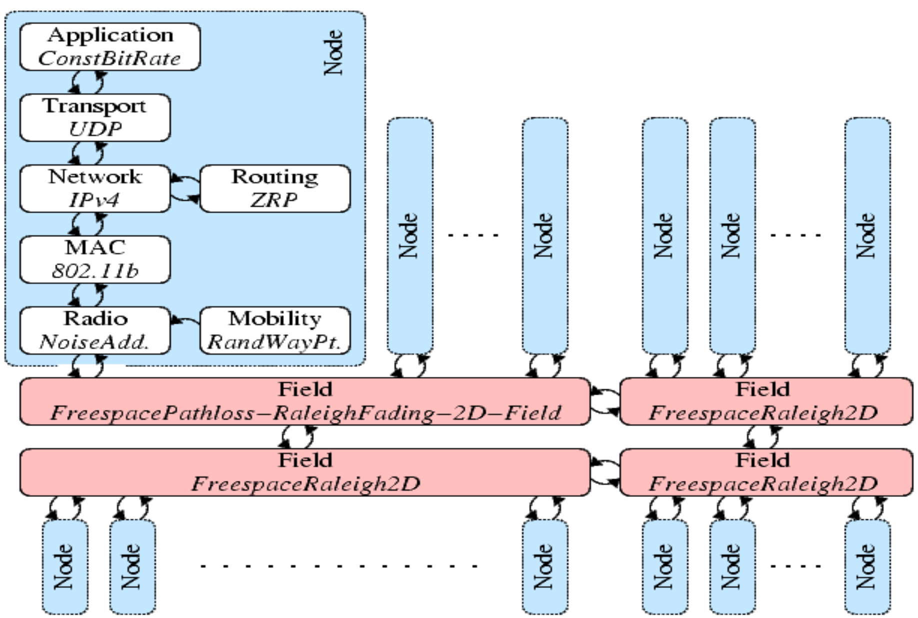


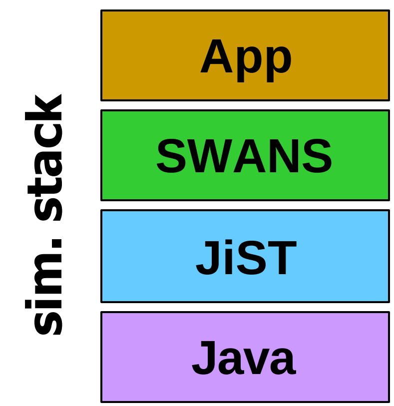


memory	per entity	per event	10K nodes sim.
JiST	**36 B**	**36 B**	**21 MB**
GloMoSim	36 B	64 B	35 MB
ns2 *	544 B	40 B	74 MB
Parsec	28536 B	64 B	2885 MB

SWANS

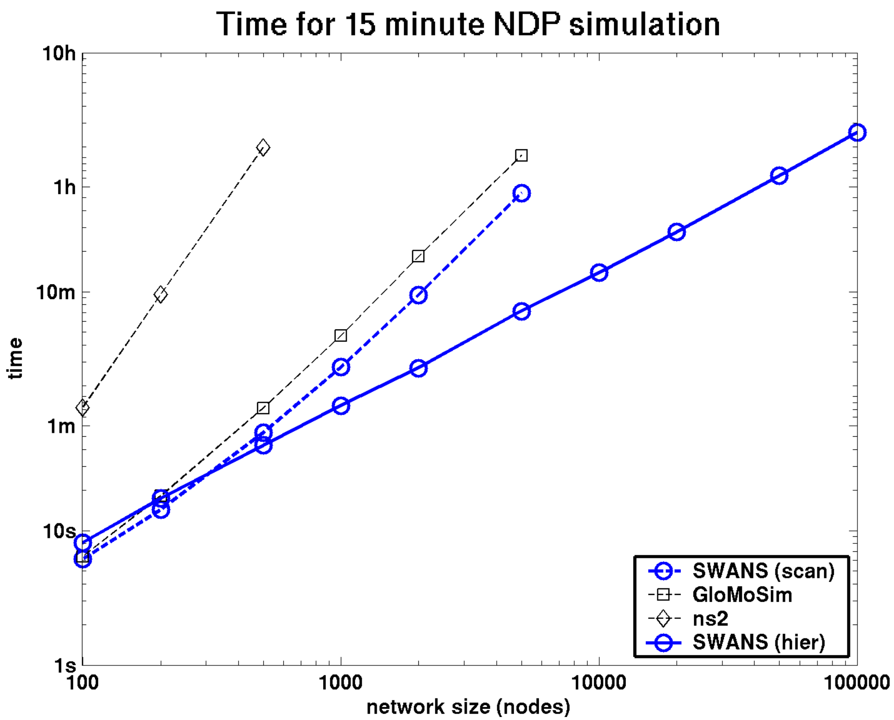
- **Scalable Wireless Ad hoc Network Simulator**
  - **similar functionality to ns2 [McCanne95] and GloMoSim [Zeng98], but…**
  - **runs standard Java network applications over simulated networks**
  - **can simulate networks of 1,000,000 nodes sequentially, on a single commodity uni-processor**
  - **runs on top of JiST; SWANS is a JiST application**
  - **uses hierarchical binning for efficient propagation**
  - **component-based architecture written in Java**

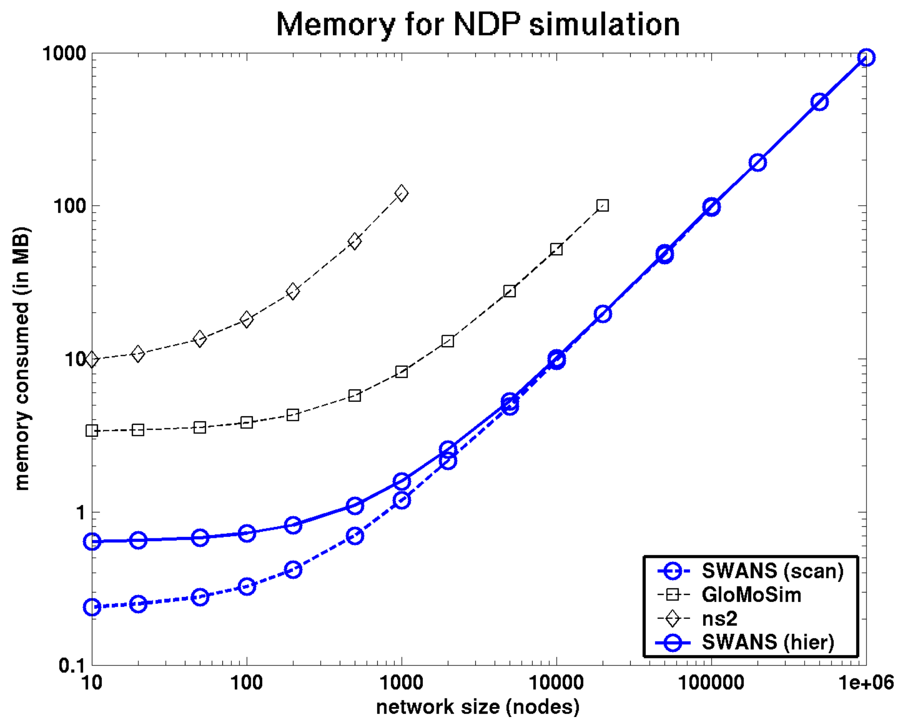




	files	classes	lines	semi
JiST	29	117	14256	3530
SWANS	85	220	29157	6586
Other	32	80	7204	2525
	146	**417**	**50617**	**12641**

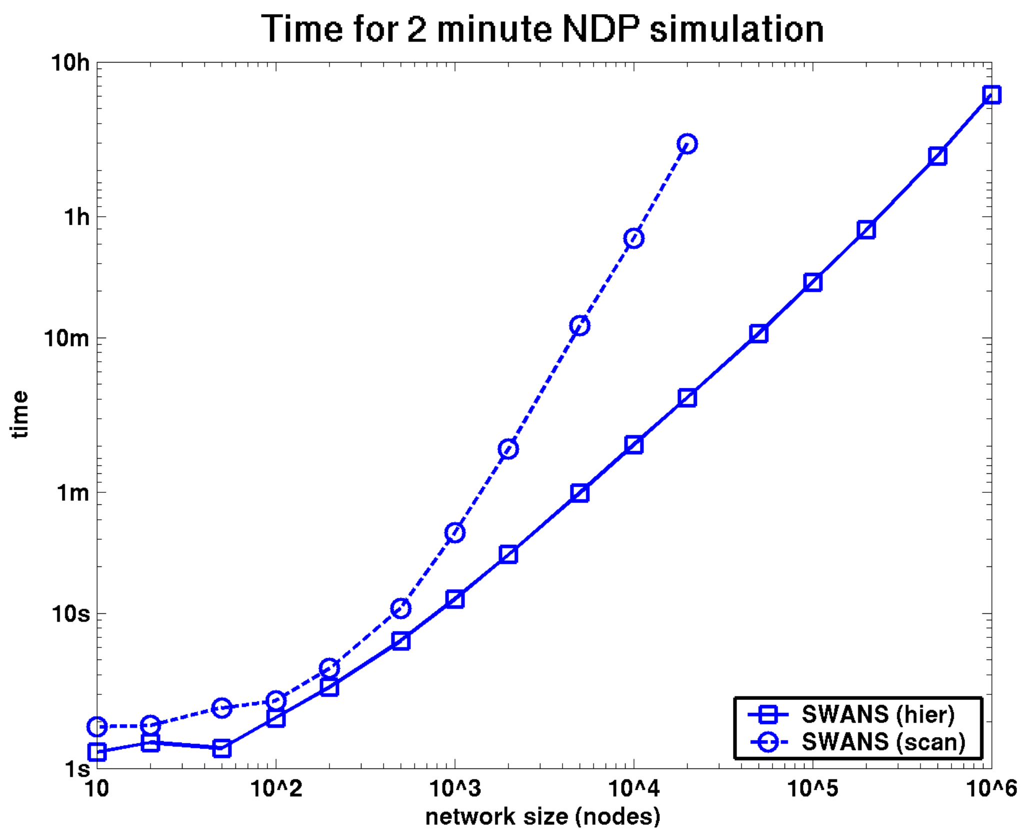
SWANS performance





t=15m	ns2		GloMoSim		SWANS		SWANS-hier	
nodes	**time**	**memory**	**time**	**memory**	**time**	**memory**	**time**	**memory**
500	7136.3 s	**58761 KB**	81.6 s	5759 KB	53.5 s	700 KB	***43.1 s***	1101 KB
5000			**6191.4 s**	**27570 KB**	**3249.6 s**	4887 KB	***433.0 s***	5284 KB
50000						**47717 KB**	***4377.0 s***	**49262 KB**

SWANS performance



t=2m	SWANS-hier NDP simulation			
nodes	10,000	100,000	**1 million**	**per node**
initial memory	13 MB	100 MB	**1000 MB**	**1.0 KB**
avg. memory	45 MB	160 MB	**1200 MB**	**1.2 KB**
time	2 m	25 m	**5.5 h**	**20 ms**

benefits of the jist approach

more than just performance…

- **application-oriented benefits**

type safety	source and target statically checked
event types	not required (implicit)
event structures	not required (implicit)
debugging	dispatch source location and state available

- **language-oriented benefits**

Java	standard language, compiler, runtime
garbage collection	cleaner code, memory savings
reflection	script-based simulation configuration
safety	fine grained isolation
robustness	no memory leaks, no crashes

- **system-oriented benefits**

IPC	no context switch, no serialization, zero-copy
Java kernel	cross-layer optimization
rewriting	no source-code access required, cross-cutting program transformations and optimizations
distribution	provides a single system image abstraction
concurrency	model supports parallel and speculative execution

- **hardware-oriented benefits**

cost	COTS hardware and clusters
portability	runs on everything

rewriter flexibility

- **simulation time transformation**
  - extend Java object model with entities
  - extend Java execution model with events
  - language-based simulation kernel

- **extensions to the model**
  - **timeless objects:** pass-by-reference to avoid copy, saves memory
  - **reflection:** scripting, simulation configuration, tracing
  - **tight event coupling:** cross-layer optimization, debugging
  - **proxy entities:** interface-based entity definition
  - **blocking events:** call and callback, CPS transformation, standard applications
  - **simulation time concurrency:** Threads, Channels and other synch. primitives
  - **distribution:** location independence of entities, single system image abstraction
  - **parallelism:** concurrent and speculative execution
  - **orthogonal additions, transformations and optimizations**

- **platform for simulation research**
  - **e.g. reverse computations in optimistic simulation [Carothers99]**
  - **e.g. stack-less process oriented simulation [Booth97]**

summary

- **JiST – Java in Simulation Time**
  - **prototype virtual machine-based simulation platform**
  - **merges systems and language-based approaches**

	kernel	library	language	**JiST**
transparent	++		++	++
efficient		+	+	++
standard	++	++		++

  - **runs SWANS: Scalable Wireless Ad hoc Network Simulator**

  - **efficient: both in terms of throughput and memory**
  - **flexible: timeless objects, reflection-based scripting, tight event coupling, proxy entities, continuations and blocking methods, simulation time concurrency, distribution, concurrency … serve as a research platform**

JiST – Java in Simulation Time

An efficient, unifying approach to simulation using virtual machines

THANK YOU.

http://jist.ece.cornell.edu/