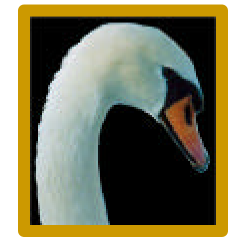
JiST –
Java in Simulation Time

for the

Scalable Simulation of Mobile Ad hoc Networks



Rimon Barr
<barr@cs.cornell.edu>
Wireless Network Laboratory
Advisor: Prof. Zygmunt J. Haas

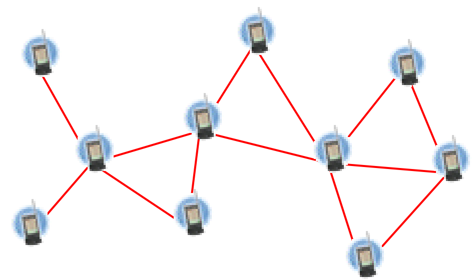
19 November 2003

http://www.cs.cornell.edu/barr/repository/jist/

motivation

- discrete event simulations are useful and needed
- but, most 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** - $<10/km^2$; or
  - reduce network **traffic** - few packets per node

  i.e. limited simulation scalability



- A university campus
  - **30,000** students, $< 4\ km^2$, 1 device/student
- The United States military
  - **100-150,000** troops, clustered around cities
- Sensor networks, smart dust, Ubicomp
  - **Hundreds of thousands** of cheap wireless devices distributed across the environment

Simulation scalability is important

existing wireless simulators

sequential

ns2 is the gold standard
- written in C++ with Tcl bindings
- created for TCP simulation, modified for wireless networks
- processor and memory intensive
- sequential; max. ~500 nodes
- recently "fixed" for ~5000 nodes

OpNet – popular commercial option
- good modeling capabilities
- poor scalability

GloMoSim
- implemented in Parsec, a custom C-like language
- implements "node aggregation," to conserve memory
- shown ~10,000 nodes on NUMA machine (SPARC 1000, est. $300k)

custom-made simulators
- fast, specialized computation
- lack sophisticated execution and also credibility

parallel

PDNS – parallel distributed ns2
- event loop uses RTI-KIT
- uses fast inter-connect to distribute memory requirements
- shown ~100,000 nodes

SWAN
- parallelized and distributed using the DaSSF framework
- similar capabilities to GloMoSim
- shown ~100,000 nodes

rule of thumb: extra 10x in scale, using at least 10x hardware and cost

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 - write program in a standard language
    - implicit optimization, concurrency, distribution, portability, fault-tolerance, etc.

how do we build simulators?

systems

- simulation kernels
  - control scheduling, IPC, clock
  - processes run in virtual time
  - e.g. TimeWarp OS, Warped

  👍 transparency 👎 efficiency

- simulation libraries
  - move functionality to user-space for performance; monolithic prog.
  - usually event-oriented
  - e.g. Yansl, Compose, ns2

  👎 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, Parsec/GloMoSim

- application-specific languages
  - e.g. Apostle, TeD

  ◇? transparency ◇? efficiency

  👍👎 new language

virtual machines

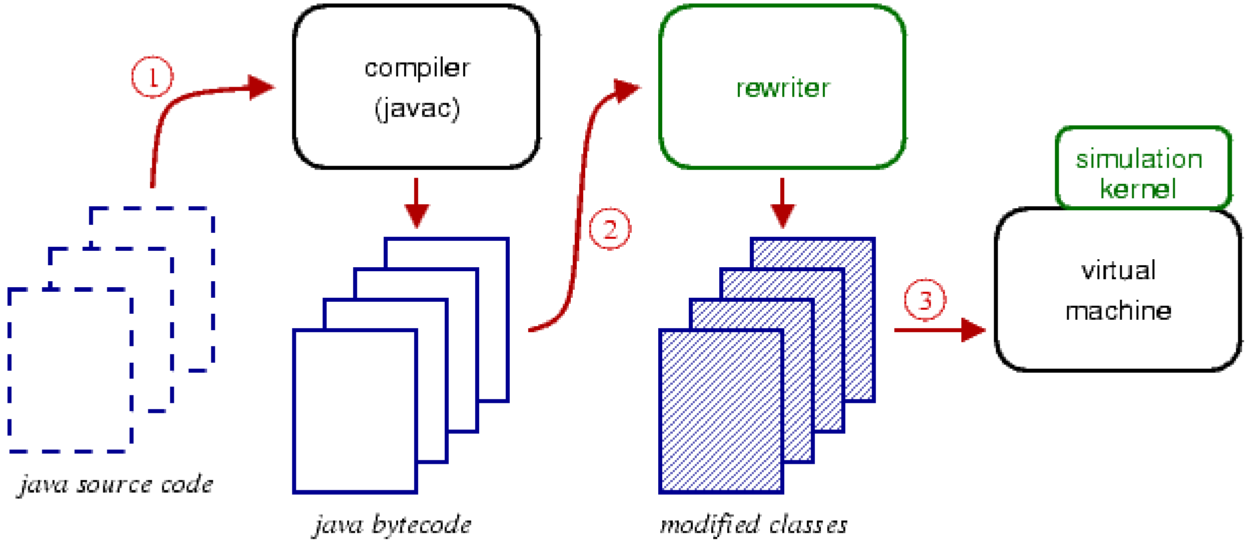
the jist approach

- 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

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

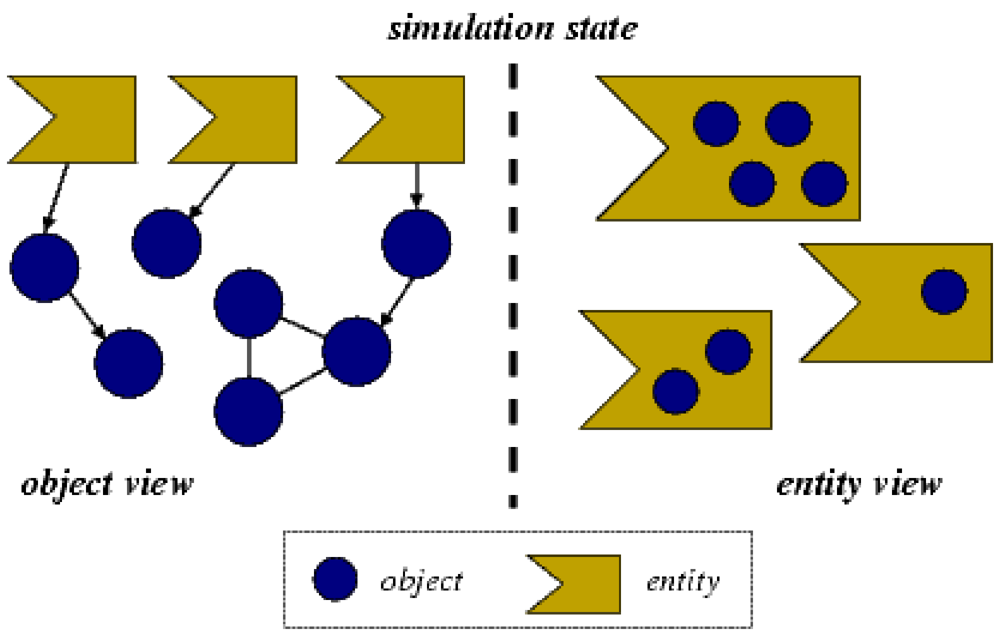
system architecture

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
   - progress of time is dependent on program progress
   - instructions take zero (simulation) time
   - time explicitly advanced by the program: `sleep(time)`
3. Rewritten program interacts with simulation kernel



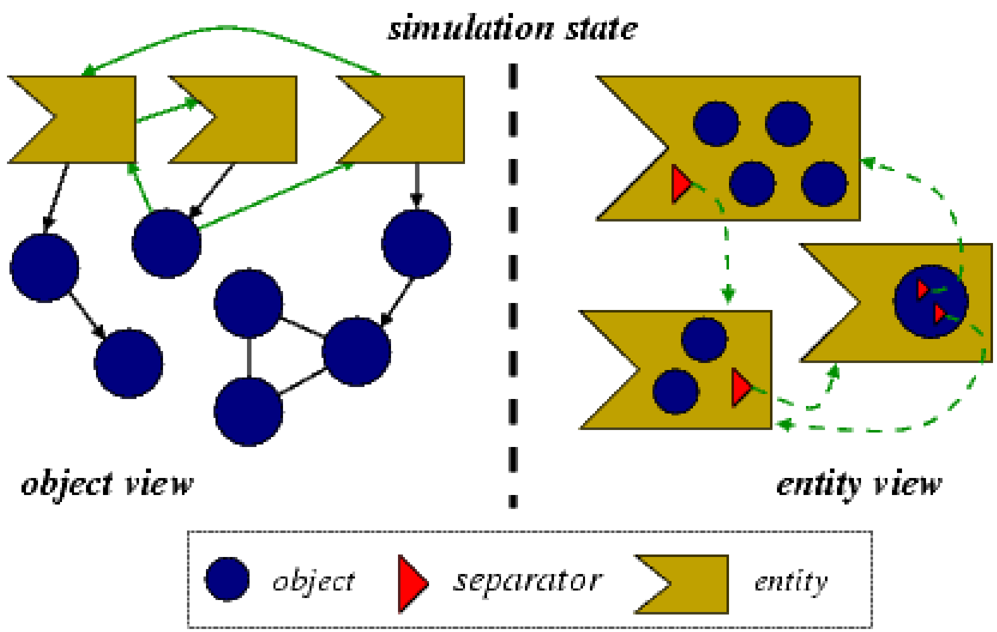
jist object model

- 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 a JKernel process in spirit, but 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



jist api

- `JistAPI` class is the JiST kernel system call interface
- permits standard Java compilation and execution

```
// used in hello example
interface Entity                - tag object as entity
long getTime()                  - return simulation time
void sleep(long ticks)          - advance simulation time

// others, to be introduced shortly
interface Timeless              - tag object as timeless
interface Proxiable             - tag object as proxiable
Entity proxy(target, intface) - create proxy entity
class Continuation ext. Error – tag method as blocking
void run(type,name,args,...)  - run program or script
void runAt(Runnable r)        - schedule procedure
void endAt(long time)           - end simulation
Channel createChannel()       - simulation time CSP Channel
void installRewrite(rewriter) – install transformation
EntityRef THIS                  - this entity reference
EntityRef ref(Entity e) - reference of an entity
// ... and more
```

*

a basic example

- the “hello world” of event simulations

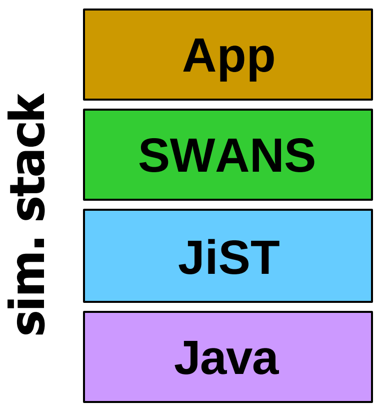
```
class HelloWorld implements JistAPI.Entity
{
  public void hello()
  {
    JistAPI.sleep(1);
    hello();
    System.out.println("hello world, " +
       "time=" + JistAPI.getTime() );
  }
}
```

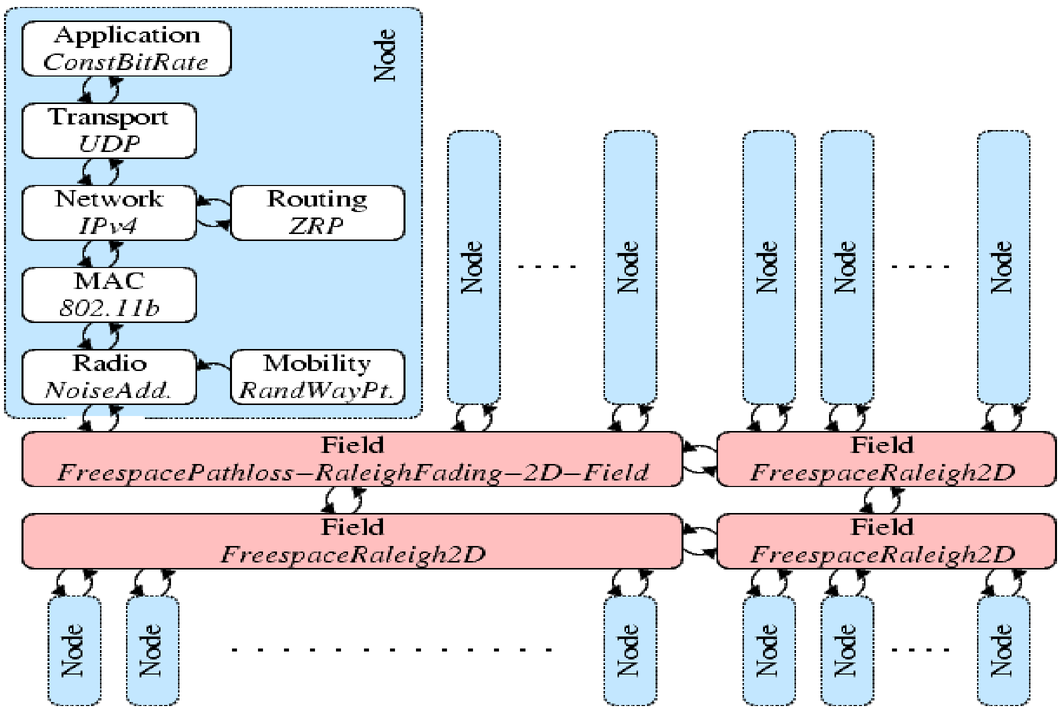
- demo!

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

SWANS

- **<u>S</u>calable <u>W</u>ireless <u>A</u>d hoc <u>N</u>etwork <u>S</u>imulator**
  - **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 signal propagation**
  - **component-based simulation architecture written in Java**

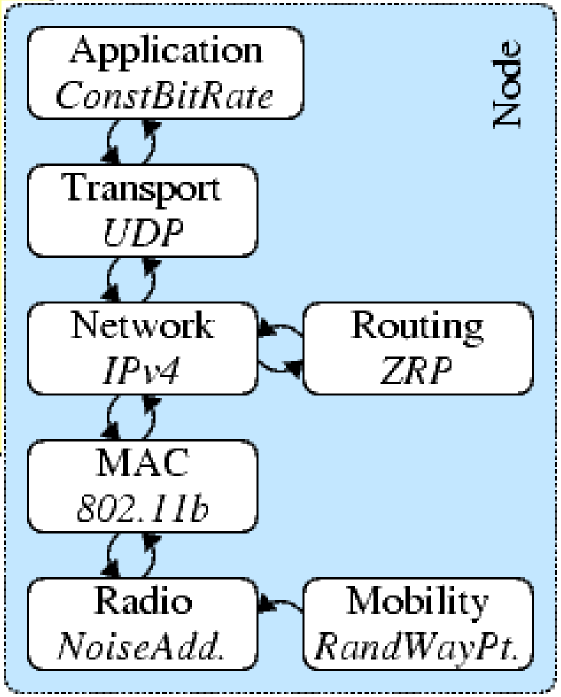




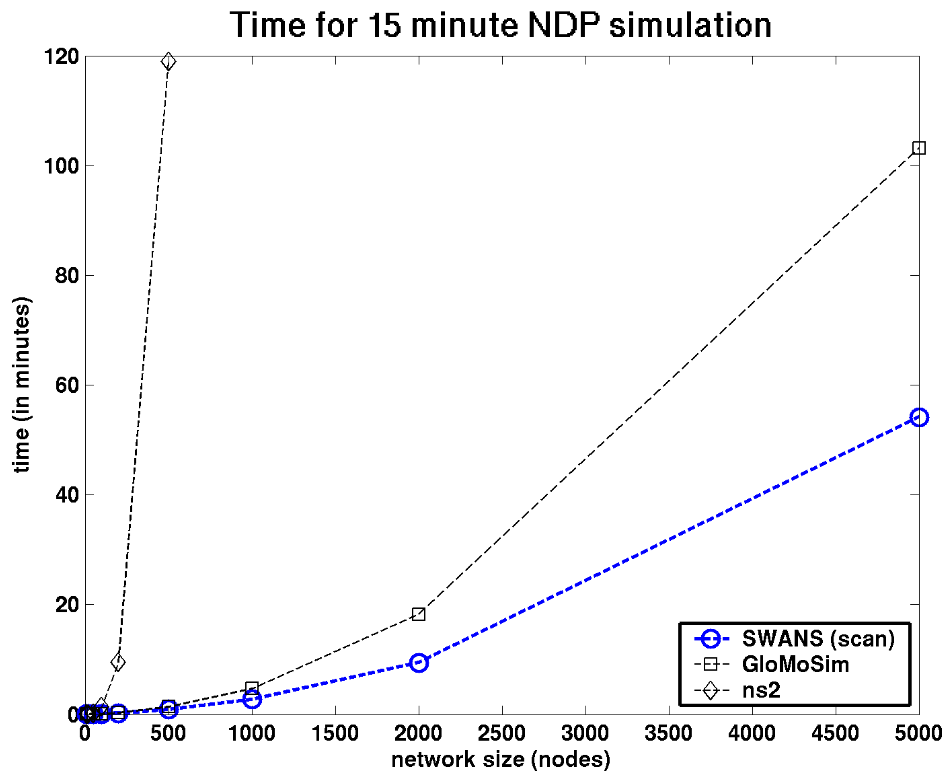
SWANS components

function	implementation
application	- ***heartbeat***; ***any Java network application***
transport	- ***UDP***; ***TCP*** [Tamtoro]
network	- ***IPv4***
routing	- ***ZRP***; ***DSR*** [Viglietta]; ***AODV*** [Lin]
link	- ***802.11b***; ***naïve***; ***wired***
placement	- ***random***; ***input file***
mobility	- ***static***; ***random waypoint***; ***input file***
interference	- ***independent***, ns2; ***additive***, GloMoSim
fading	- ***zero***; ***Raleigh***; ***Rician***
pathloss	- ***free-space***; ***two-ray***
propagation algorithm	- ***linear scan***, ns2; ***flat binning***, GloMoSim; ***hierarchical binning***

	files	classes	lines	semi
JiST	25	95	11265	2892
SWANS	67	152	17809	5263
Other	26	53	3994	1602
	118	**300**	**33068**	**9757**

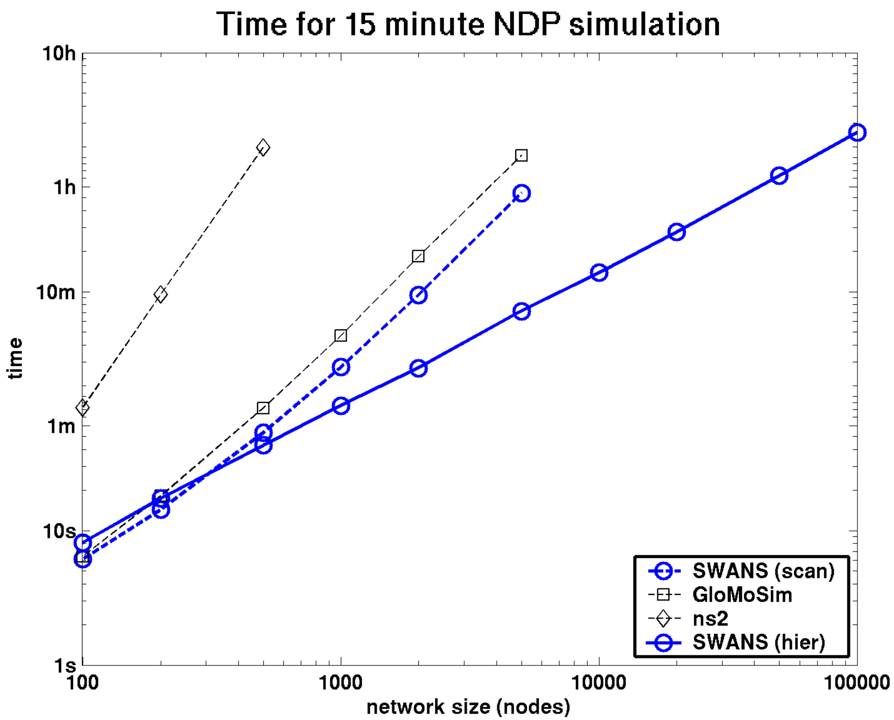


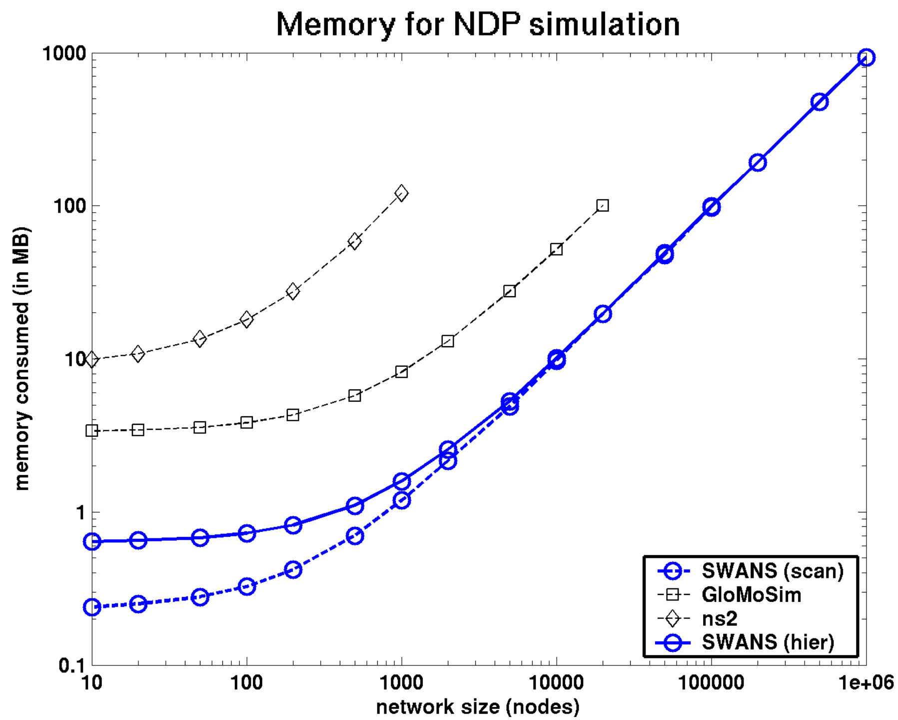
SWANS performance



- simulation configuration
  - application - heartbeat neighbor discovery
  - field - $5x5km^2$; free-space path loss; zero fading
  - mobility - random waypoint: v=2-5m, p=10s
  - radio - additive noise; standard power, gain, etc.
  - stack - 802.11b, IPv4, UDP

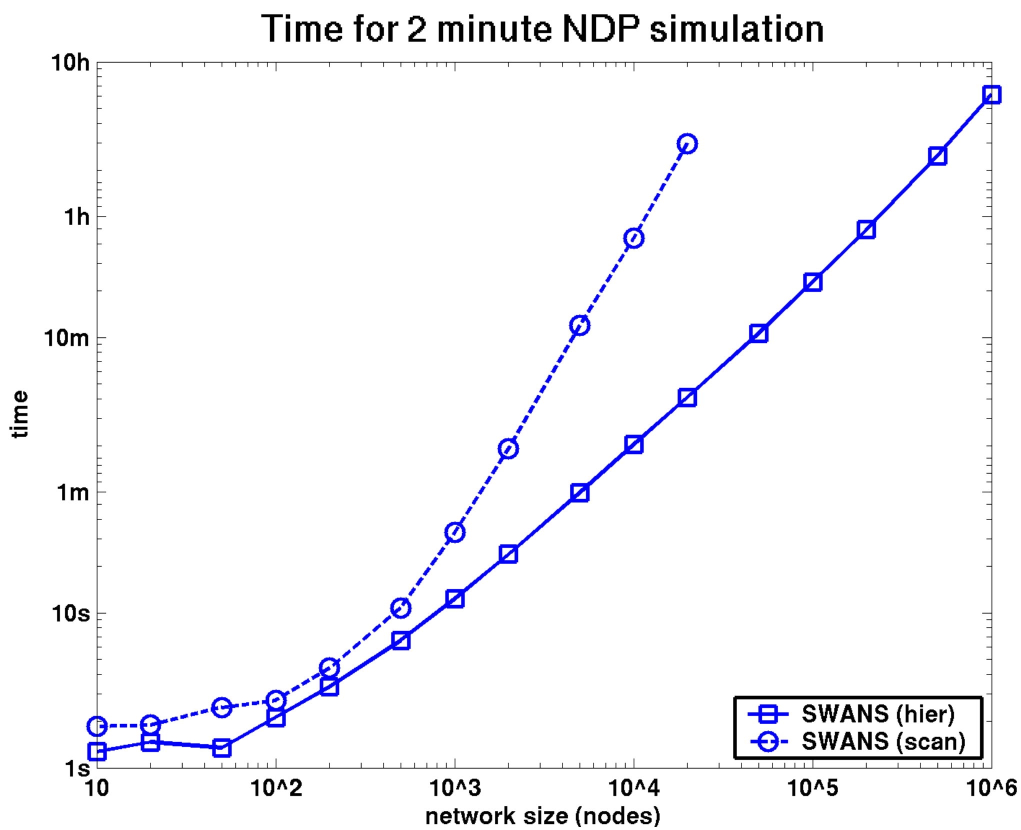
SWANS performance





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**

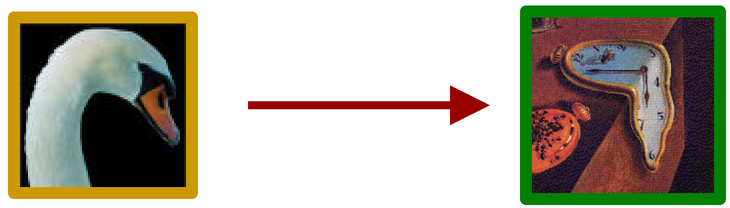
SWANS performance



t=2m	SWANS-hier			
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**

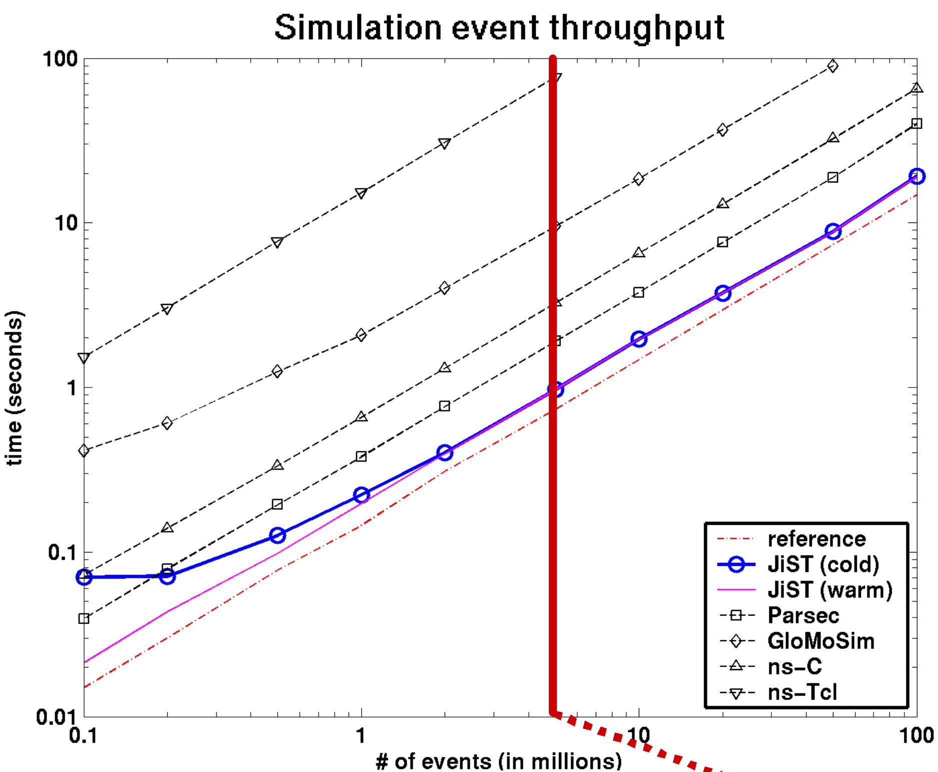
summary

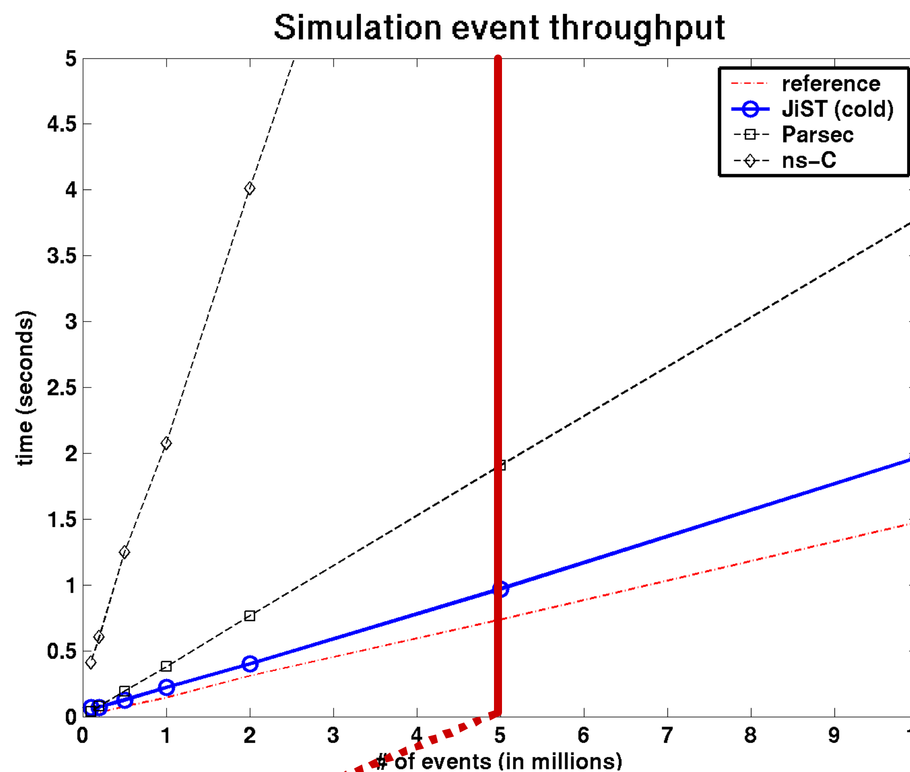
- **SWANS scalability**
  - **can simulate million node wireless networks**
  - **hierarchical binning allows linear scaling with network size**
- **SWANS is a JiST application**
  - **a simulation program written using the "JiST approach"**

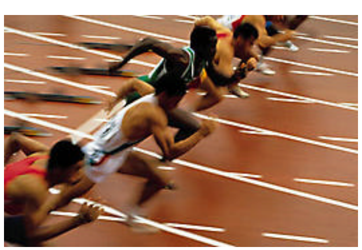


- **scalability depends on:**
  - **time – efficient simulation event processing**
  - **space – efficient simulation state encoding**

jist micro-benchmark: event throughput

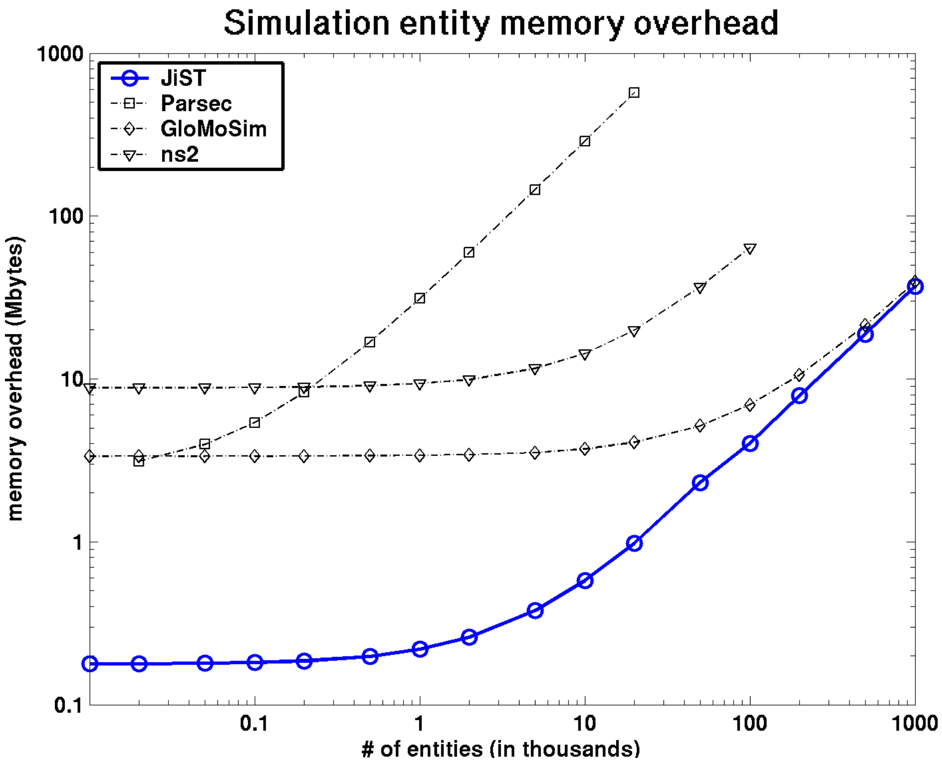


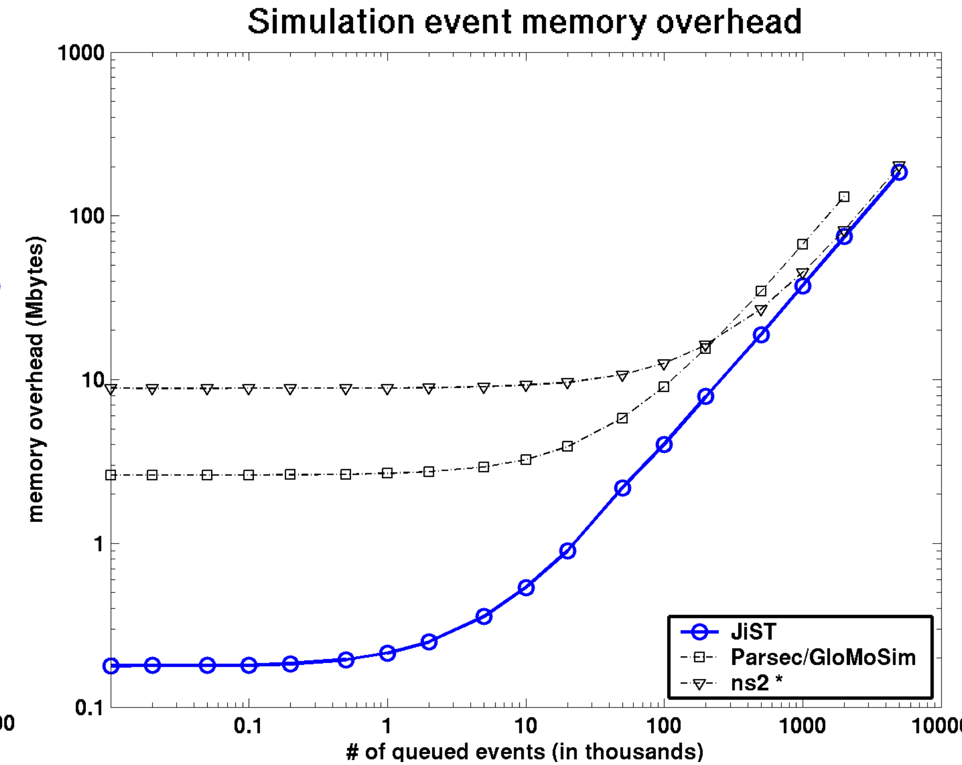




5x10^6 events	time (sec)	vs. reference	vs. JiST
reference	0.74		0.76x
JiST	**0.97**	1.31x	
Parsec	1.91	2.59x	**1.97x**
ns2-C	3.26	4.42x	**3.36x**
GloMoSim	9.54	12.93x	**9.84x**
ns2-Tcl	76.56	103.81x	**78.97x**

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

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
distribution	provides a single system image abstraction
concurrency	model supports parallel and speculative execution

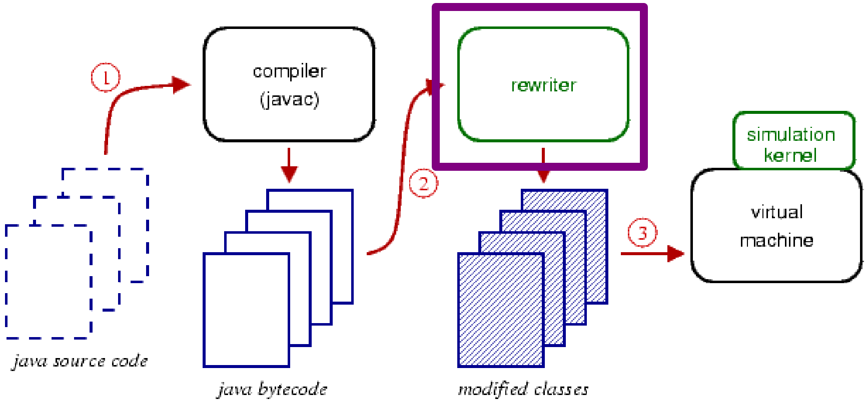
- **hardware**-oriented benefits

cost	COTS hardware and clusters
portability	runs on everything

rewriter

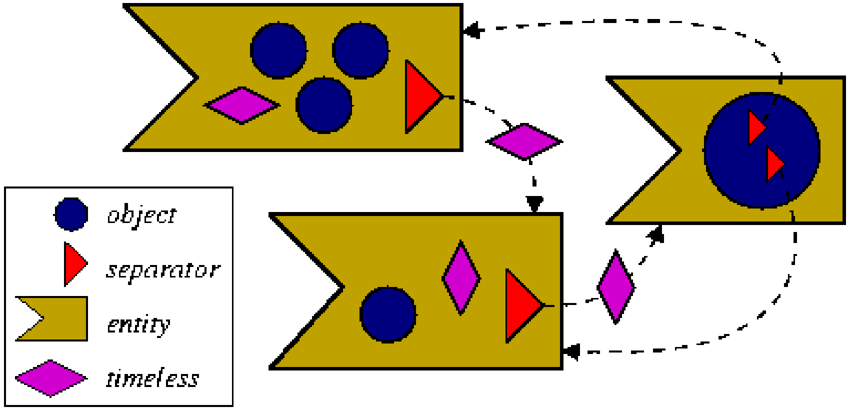
- **rewriter properties**
  - **dynamic class loader**
  - **no source code access required**
  - **operates on application packages, not system classes**
  - **uses Apache Byte Code Engineering Library (BCEL)**
  - **allows orthogonal additions, transformations and optimizations**

- **rewriting phases**
  - **application-specific rewrites**
  - **verification**
  - **add entity self reference**
  - **intercept entity state access**
  - **add method stub fields**
  - **intercept entity invocations**
  - **modify entity creation**
  - **modify entity references**
  - **modify typed instructions**
  - **continuable analysis**
  - **continuation transformation**
  - **translate JiST API calls**



zero-copy semantics

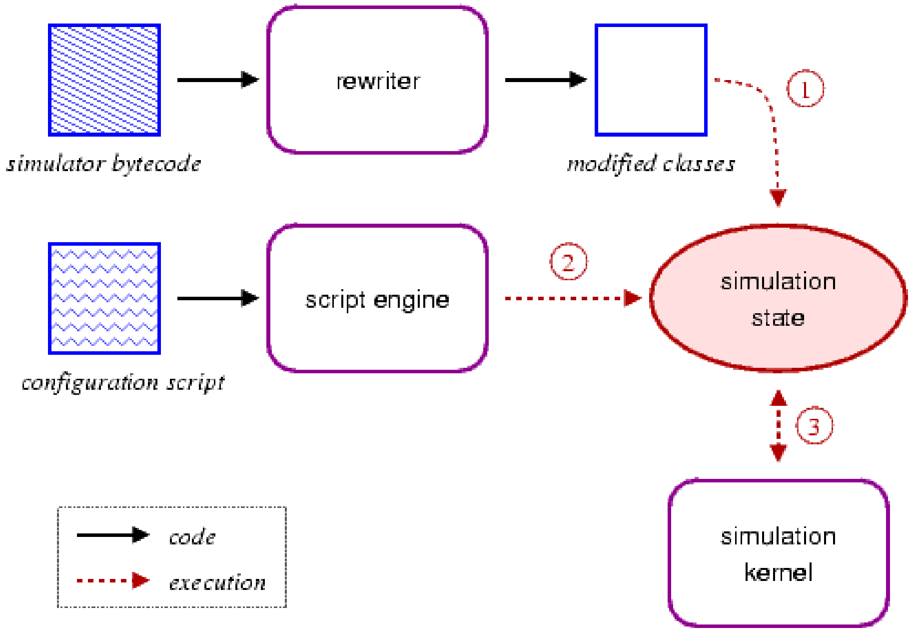
- timeless object: a temporally stable object
  - inferred statically as open-world immutable
  - or tagged explicitly with the `Timeless` interface
- benefits
  - pass-by-reference saves memory copy
    - zero-copy semantics for inter-entity communication
  - saves memory for common shared objects
    - e.g. broadcast network packets
    - rewrite `new` of common types to `hashcons`



configurability

- **configurability is essential for simulators**
  1. **source level reuse; recompilation**
  2. **configuration files read by driver program**
  3. **driver program is a scripting language engine**

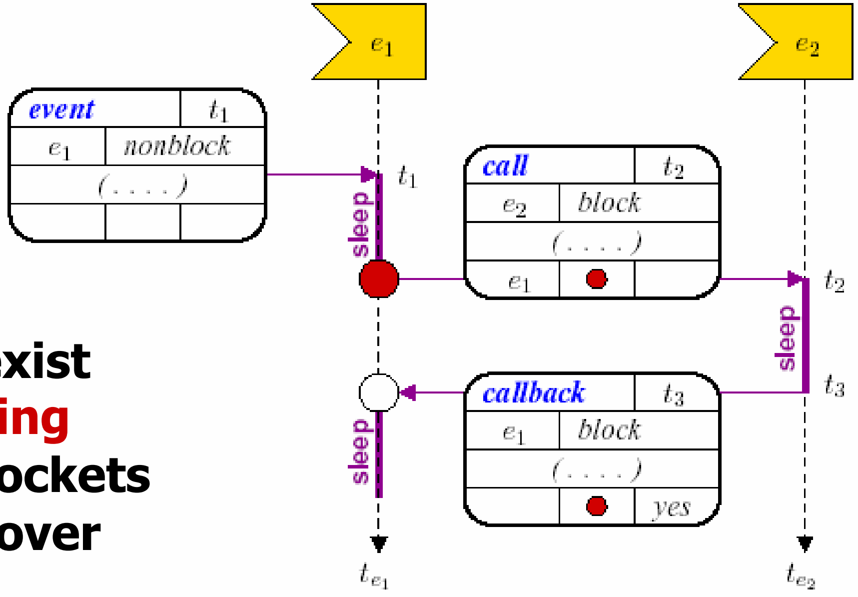
- **support for multiple scripting languages by reflection**
  - **no additional code**
  - **no memory overhead**
  - **no performance hit**
  - **Bsh - scripted Java**
    Jython - Python
  - **Smalltalk, Tcl, Ruby, Scheme and JavaScript**



simulations using real applications

- using entity method invocations…
  - one can easily write **event-driven** entities.
  - what about **process-oriented** simulation?

- **blocking events**
  - any entity method that "throws" a `Continuation` exception
  - event processing frozen at invocation
  - continues after call event completes, at some later simulation time

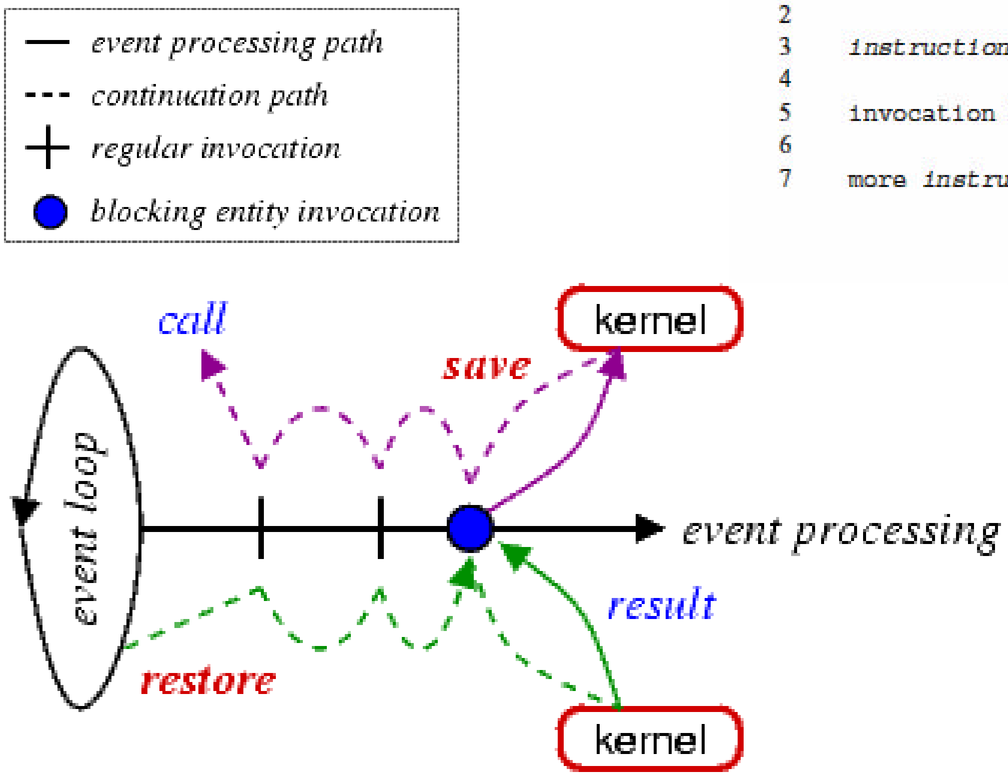


- benefits
  - no explicit process
  - blocking and non-blocking coexist
  - akin to **simulation time threading**
  - can build simulated network sockets
  - can **run standard applications** over these simulated sockets

*

capturing continuations

- **mark entity method as blocking: `throws Continuation`**
- **saving and restoring the stack is non-trivial in Java!**



Before CPS transform:

```
METHOD continuable:

  instructions

  invocation BLOCKING

  more instructions
```

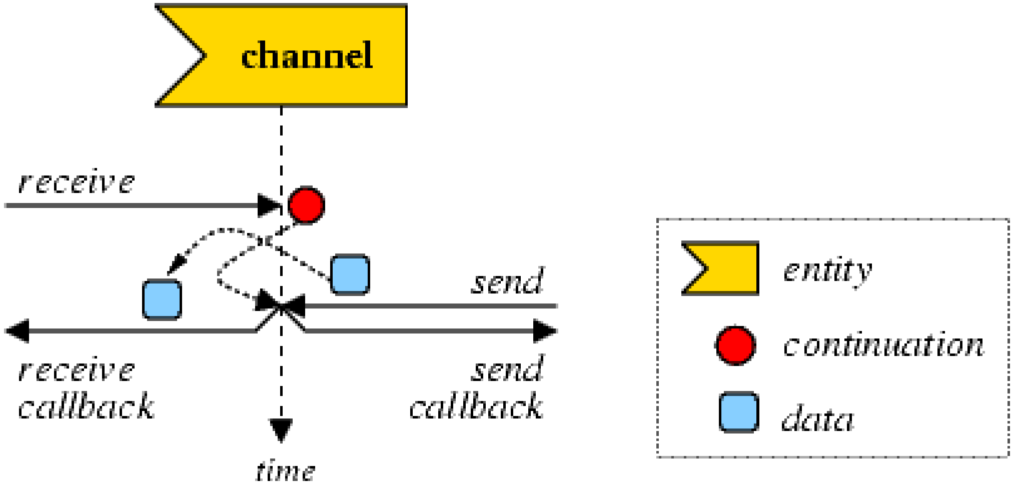
After CPS transform:

```
METHOD continuable:
  if jist.isRestoreMode:
    jist.popFrame f
    switch f.pc:
      case PC1:
        restoreLocals f.locals
        restoreStack f.stack
        goto PC1
      ...

  instructions

  setPC f.pc, PC1
  saveLocals f.locals
  saveStack f.stack
PC1:
  invocation BLOCKING
  if jist.isSaveMode:
    jist.pushFrame f
    return

  more instructions
```

simulation time concurrency

using continuations...

- **simulation time `Thread`**
  - **cooperative concurrency**
  - **can also support pre-emptive, but not necessary**
- **simulation time concurrency primitives:**
  - **CSP Channel: `JistAPI.createChannel()`**
  - **locks, semaphores, barriers, monitors, FIFOs, ...**



rewriter flexibility

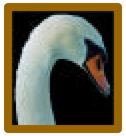
- **simulation time transformation**
  - extend Java object model with entities
  - extend Java execution model with events
  - language-based simulation kernel
- **extensions to the model**
  - **timeless objects:** pass-by-reference to avoid copy, saves memory
  - **reflection:** scripting, simulation 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 [Carothers ’99]**
  - **e.g. stack-less process oriented simulation [Booth ’97]**

summary

- JiST – Java in Simulation Time
  - convert virtual machine into simulation platform
  - 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 simulation research platform
  - merges systems- and language-based approaches to simulator construction
    - efficient, transparent and standard

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

- SWANS – Scalable Wireless Ad hoc Network Simulator
  - built atop JiST, proof of concept
  - component-based framework
  - runs standard Java networking applications
  - uses hierarchical binning to perform signal propagation
  - scales to networks of a million nodes on a uni-processor



JiST –
Java in Simulation Time

for the

Scalable Simulation of Mobile Ad hoc Networks

THANK YOU.