A Case for Cooperative Scheduling in X10's Managed Runtime

X10 Workshop 2014
June 12, 2014

Shams Imam, Vivek Sarkar
Rice University

Task-Parallel Model

- Worker Threads

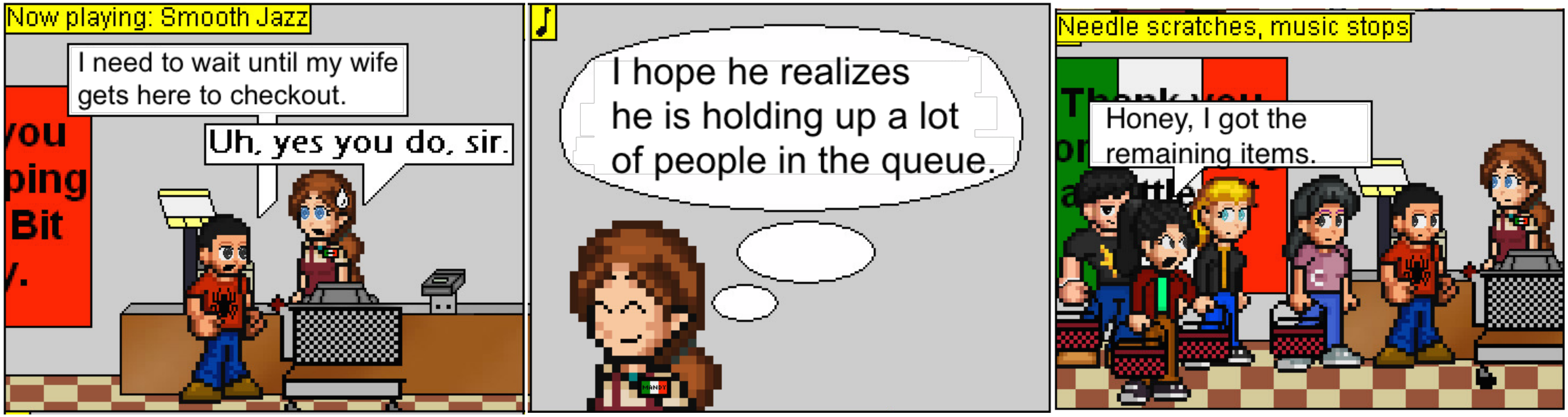
Please ignore the DP on the cartoons
source: http://www.deviantart.com/art/Randomness-20-178737664

Task-Parallel Model

- Tasks, Work Queues, and Worker Threads
- Runtime manages load balancing and synchronization

source: http://www.deviantart.com/art/Randomness-20-178737664

Synchronization Constraints



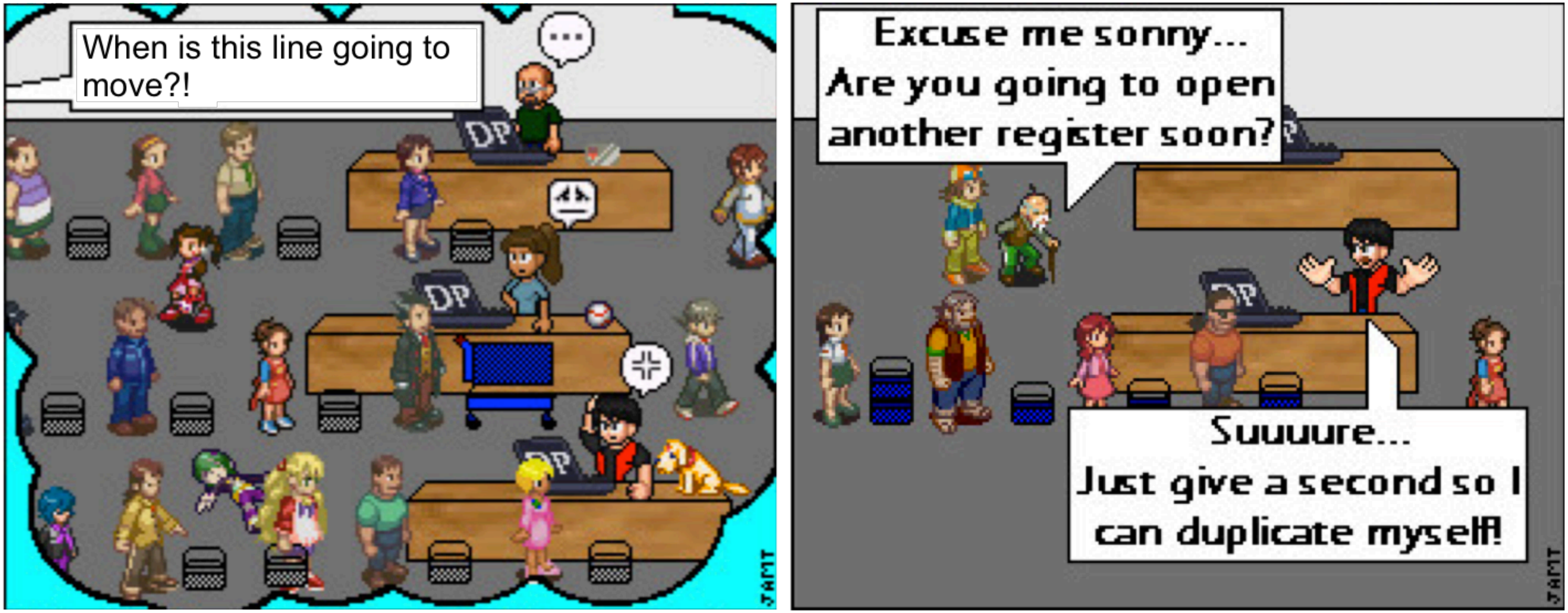
- Dependences between tasks
- Prevent an executing task from making further progress
  - Needs to synchronize with other task(s)

source: http://viper-x27.deviantart.com/art/Checkout-Lane-Guest-Comic-161795346

X10 Synchronization

- Current synchronization constructs
  - Finish
  - Futures
  - Clocks
  - Atomic Blocks
  - More in the future?
- Current implementation blocks worker threads
  - For most constructs (everything other than finish)

Current Solution to Synchronization: Block Worker Threads



Thread blocking approaches do not scale!

source: http://www.deviantart.com/art/Randomness-5-90424754

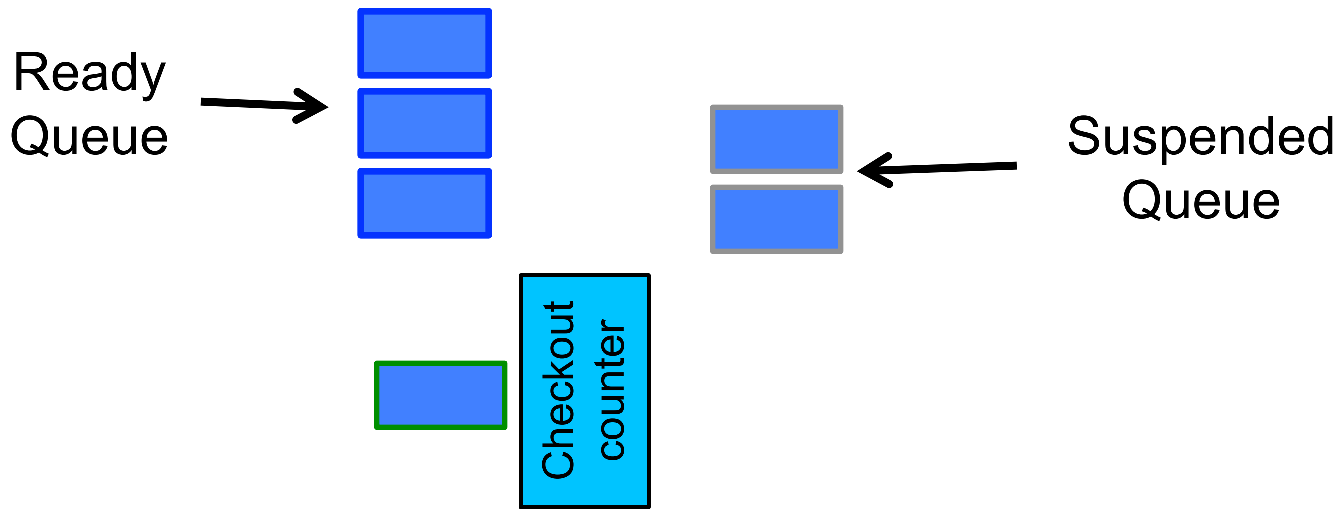
Proposed Solution

- A Cooperative Approach is more efficient

source: http://www.deviantart.com/art/3-000-Views-120361172

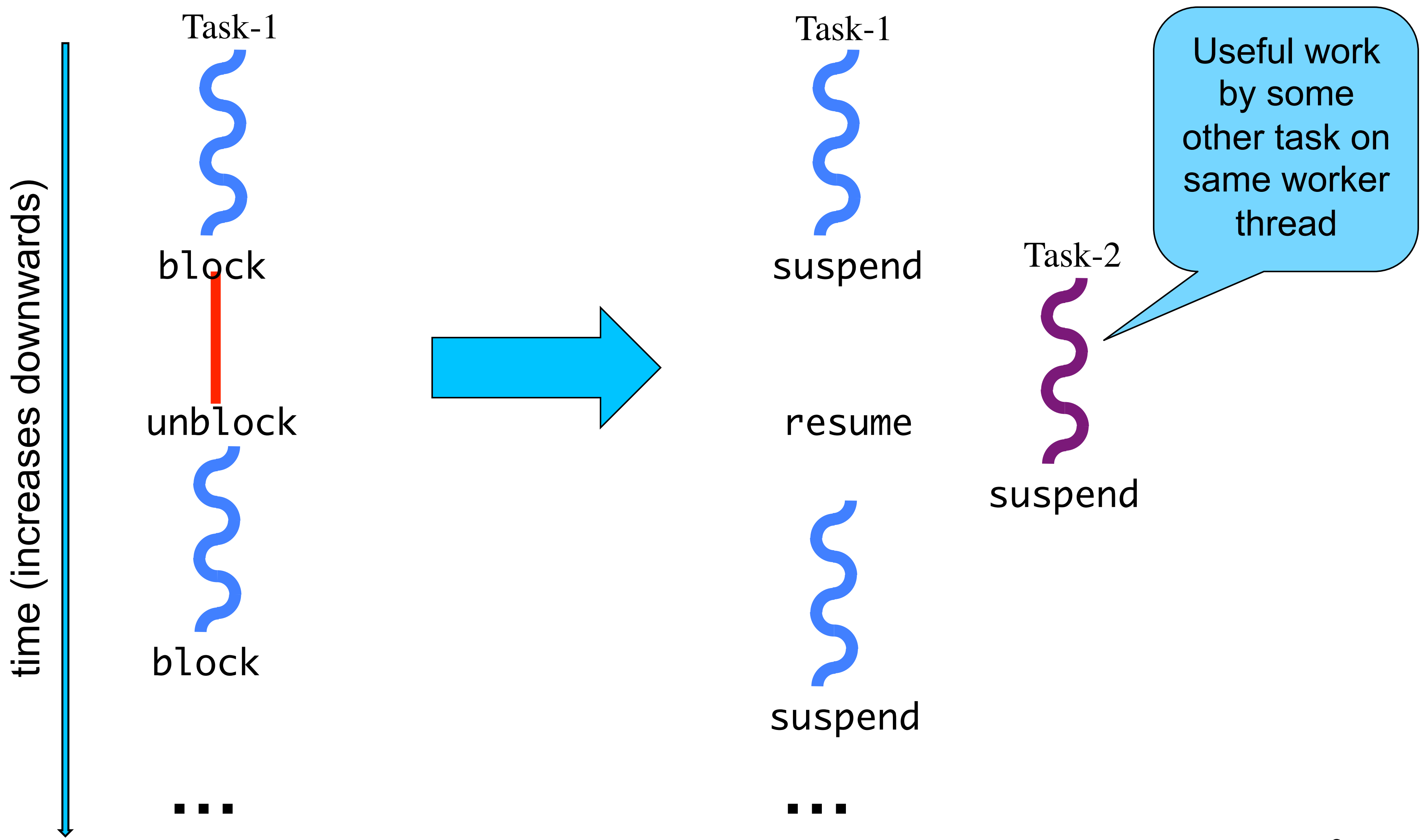
Cooperative Scheduling

- Task decides to actively suspend itself and **yield** control back to the runtime
- Task is added back into the ready queue when the task can make progress



Cooperative Scheduling: http://en.wikipedia.org/wiki/Computer_multitasking#Cooperative_multitasking

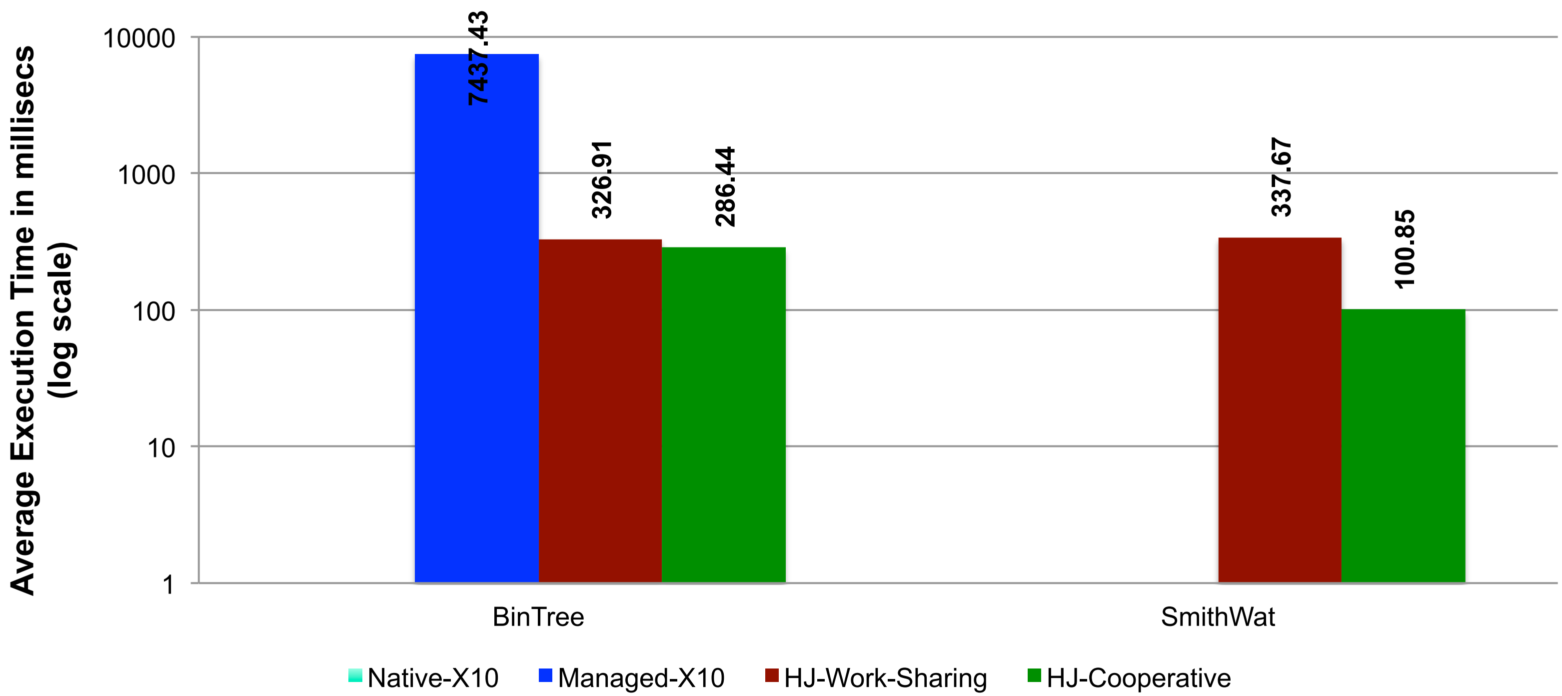
Cooperative Scheduling (contd)



Experimental Setup

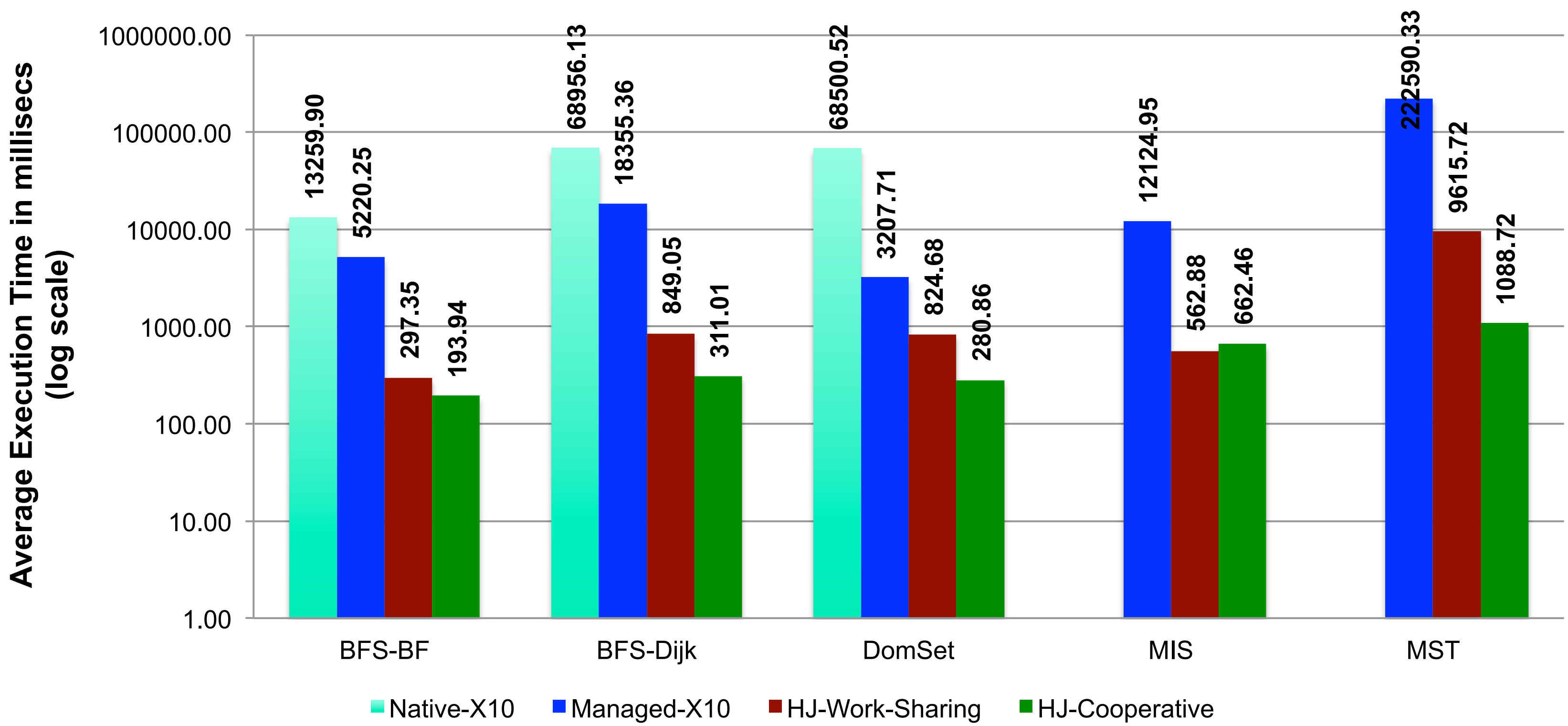
- 12-core 2.8 GHz Intel Westmere
  - 48 GB of RAM
  - Threads bound to cores (using `taskset` command)
  - JDK 1.7
- Habanero-Java language v1.3.1
  - Default scheduler = work-sharing
  - Cooperative scheduler enabled via option [ECOOP 2014]
- X10 version 2.3.1-2
  - Compared against native and managed runtime
  - Compiled using `-OPTIMIZE=true` flag
- Benchmarks run with single place
  - 12 worker threads per place

Future Benchmarks



- HJ includes future construct
- X10 includes future library (x10.util.concurrent.Future)
- HJ and X10 versions are identical except for future syntax
- SmithWaterman on X10 reports “too many threads” error!

Clock + Atomic Benchmarks



- IMSuite Benchmarks: Input size of 512 nodes
- HJ/X10 versions are identical
  - clock/phaser
  - atomic/isolated

Technical Details

- Delimited Continuations
- Event-Driven Controls

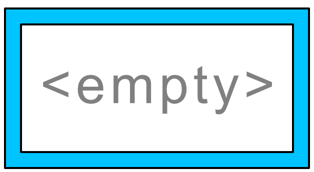
One-shot Delimited Continuations

- Rest of the computation from a well-defined outer boundary
  - i.e. represents a sub-computation
- *Suspend* the state of a computation at any point
- *Resume* the computation, later, from that point
- One-shot: resumed at most once

Defined in “Continuations in the Java Virtual Machine”, Iulian Dragos et al.

Event-Driven Control (EDC)

- Binds a value and a list of runnable blocks
  - Runnable blocks are just code snippets
- Dynamic single-assignment of value (event)

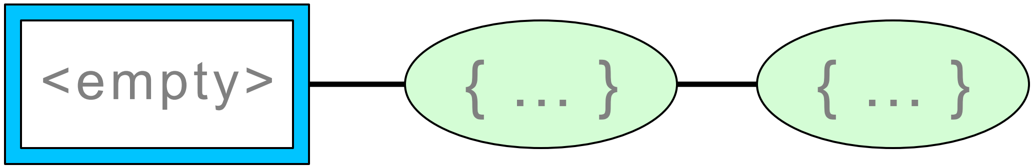


The EDC is initially empty

Defined in “Cooperative Scheduling of Parallel Tasks with General Synchronization Patterns”, ECOOP 2014.

Event-Driven Control (EDC)

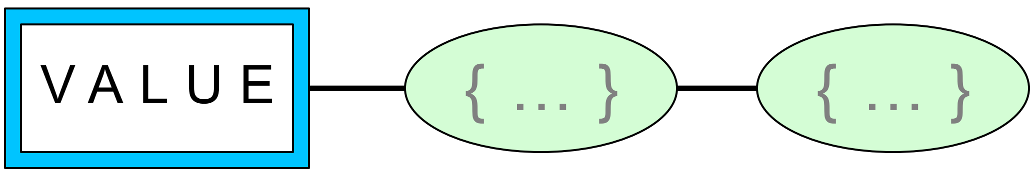
- Binds a value and a list of runnable blocks
- Dynamic single-assignment of value (event)



Runnable blocks attach to the EDC and are not triggered until value is available (i.e. until event is satisfied)

Event-Driven Control (EDC)

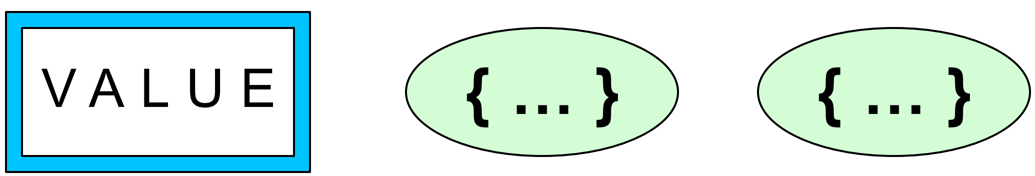
- Binds a value and a list of runnable blocks
- Dynamic single-assignment of value (event)



Eventually, a value becomes available in the EDC (follows from deadlock freedom property of finish, futures, clocks, atomic)

Event-Driven Control (EDC)

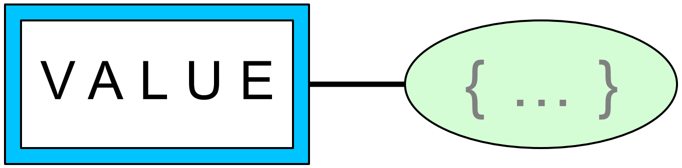
- Binds a value and a list of runnable blocks
- Dynamic single-assignment of value (event)



This enables execution of runnable blocks attached to the EDC

Event-Driven Control (EDC)

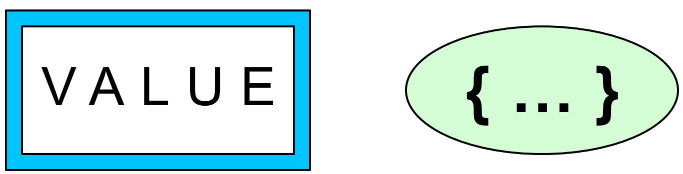
- Binds a value and a list of runnable blocks
- Dynamic single-assignment of value (event)



Subsequent runnable block attachment requests…

Event-Driven Control (EDC)

- Binds a value and a list of runnable blocks
- Dynamic single-assignment of value (event)



Synchronously execute the runnable block
(e.g. schedule a task into the work queue)

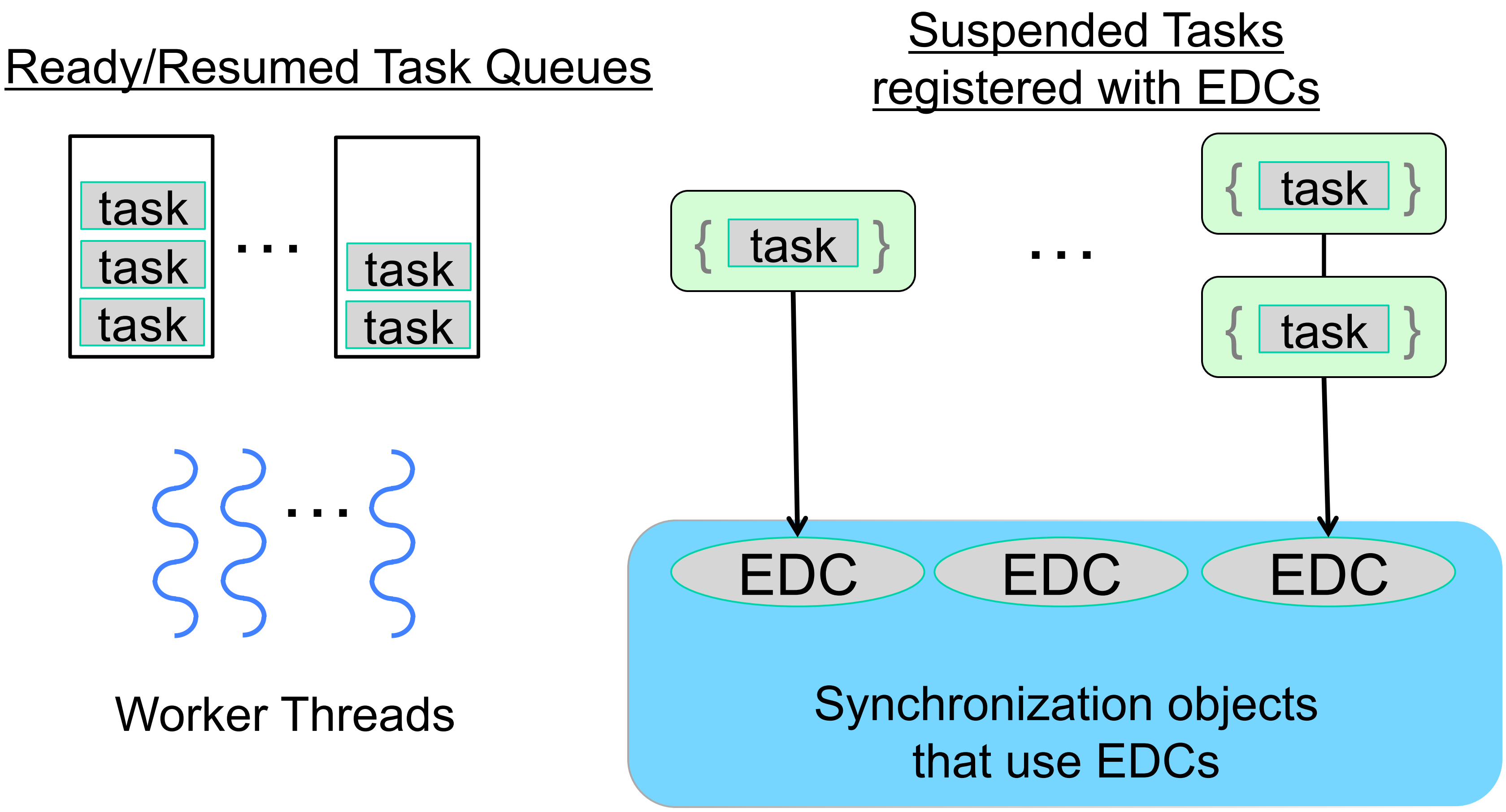
Event-Driven Control API

- `currentTaskId`():
  - returns a unique id of the currently executing task
- `newEDC`():
  - factory method to create EDC instance
- `suspend`(*anEdc*):
  - the current task is suspended if the EDC has not been resolved
  - Implementation attaches runnable block to resume task
- *anEdc*.`getValue`()
  - retrieves the value associated with the EDC
  - safe to call this method if execution proceeds past a call to suspend()
- *anEdc*.`setValue`(*aValue*)
  - resolves the EDC
  - triggers the execution of any EBs

Cooperative Runtime

- We expose EDCs as an API in our runtime.
  - Read / Write / Query on value
  - Suspend till value becomes available
- Continuations not exposed to developer
  - Notorious for being hard to use and to understand
- Developers write thread-based code
  - Compiler handles CPS code transformations
  - One-shot delimited continuations implemented more efficiently than general continuations

Cooperative Runtime



Benefits of Cooperative Runtime

- Bound the number of worker threads
- Threads never block
  - Additional threads do not need to be created
  - (Tasks may suspend)
- Do not need more than one worker thread
  - Computations can be made serializable
  - Can help in reproducibility and debugging

Synchronization Constructs

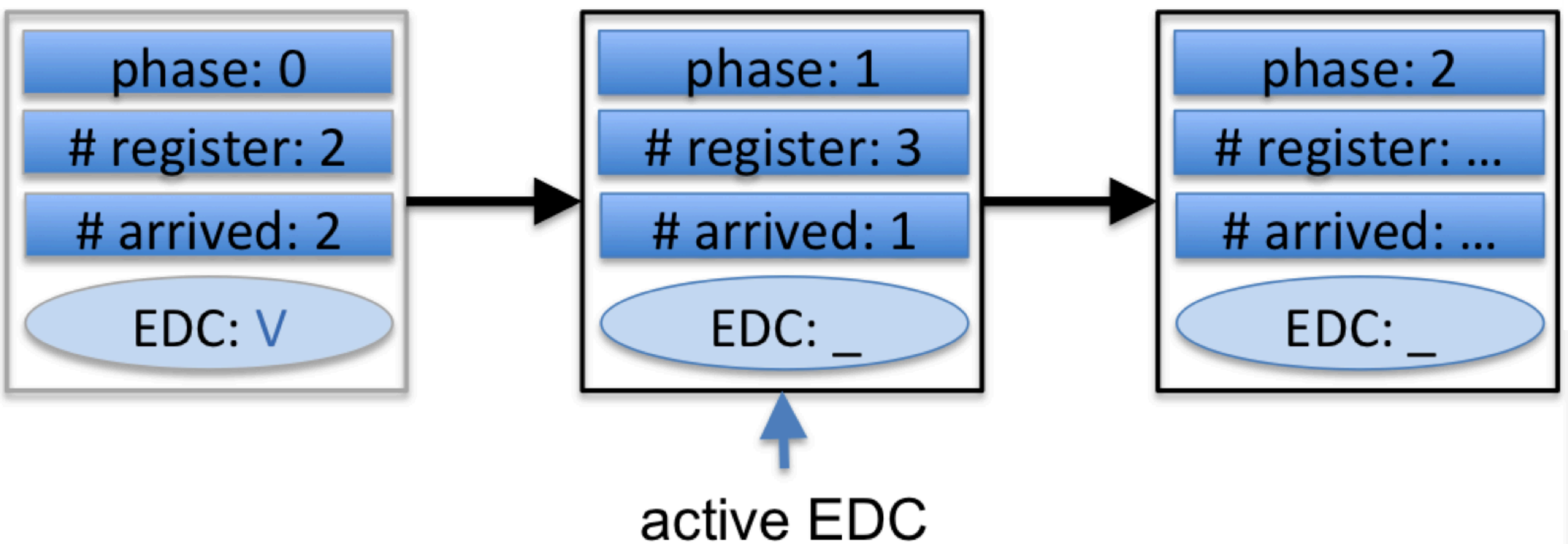
- Key idea is to:
  - Translate the coordination constraints into producer-consumer constraints on EDCs
  - Use Delimited Continuations to suspend consumers when waiting on item(s) from producer(s)
- Any task-parallel Synchronization Constraint can be supported.
  - Both deterministic and non-deterministic constructs
  - Including atomic/isolated and actors

Implementation Recipe

- Async-Finish
  - Atomic counter to track in-flight spawned tasks
  - Single EDC resolved when count reaches zero
- Futures
  - Single EDC to store future value
  - EDC resolved when future task is executed
- Atomic/Isolated blocks
  - Linked-list of EDCs to grant tasks permission to execute
  - During ‘unlock’ resolve the value of the next EDC in the list
  - Use one list per place for X10’s place-local atomic operations

Implementation Recipe

- Clocks
  - One EDC per phase
  - Track tasks registered and arrived using atomic counters for each phase
  - Resolve EDC when counts become equal



Summary

- Cooperative runtime for scheduling tasks
- Using
  - One-shot Delimited Continuations
  - Event-Driven Controls
- Can support any task-parallel synchronization
- Foundations of approach described in ECOOP 2014 paper
- This work extended those results with comparison with X10

Future work

- Cooperative scheduling for library implementation of Habanero-Java (Hjlib)
- Pre-emptive Scheduling
  - Suspend long running tasks for fairness
  - Support priorities
- Eureka Computations
  - Support for Cilk-like abort statement with sound semantics

source: http://www.cartoonstock.com/cartoonview.asp?catref=bso0035

Questions

- Cooperative runtime for scheduling tasks
- Using
  - One-shot Delimited Continuations
  - Event-Driven Controls

```
import x10.audience.Questions;
```

- Can support any task-parallel synchronization
- Foundations of approach described in ECOOP 2014 paper
- This work extended those results with comparison with X10

Backup-Slides

Acknowledgments

- Vivek Sarkar
- Rest of the Habanero Group
  - Vincent Cave
  - Akihiro Hayashi
  - Sagnak Tasirlar
  - Jisheng Zhao

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()

Console:

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume(); // will invoke run()
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()
resume()

Console:

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()
resume()
run()

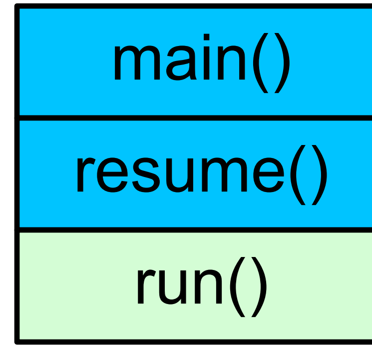
Console:

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack



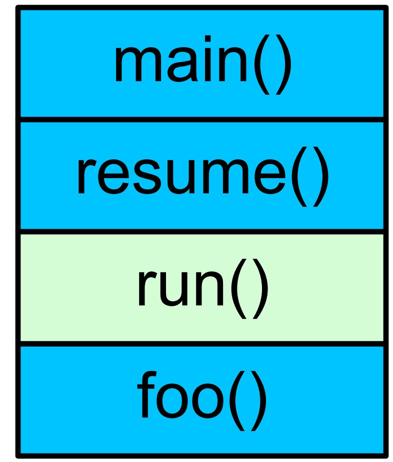
Console:

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack



Console:

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()
resume()
run()
foo()

Console:

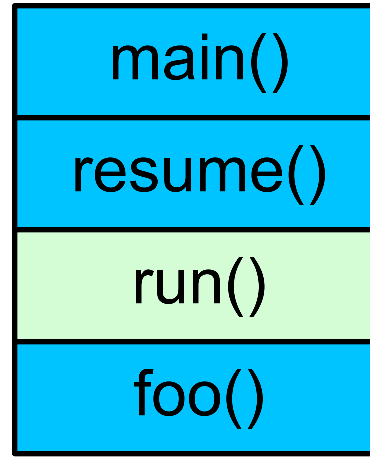
foo: A

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack



Console:

```
foo: A
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()

Console:

```
foo: A
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()

Console:

```
foo: A
 cause: foo-2
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()

Console:

```
foo: A
 cause: foo-2
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()
resume()

Console:

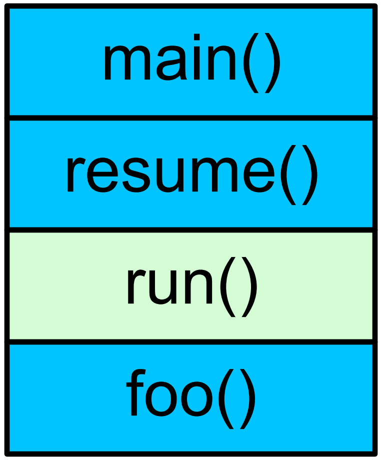
```
foo: A
 cause: foo-2
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack



Console:

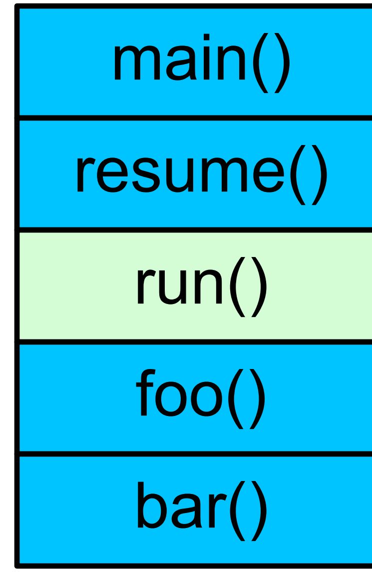
```
foo: A
 cause: foo-2
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack



Console:

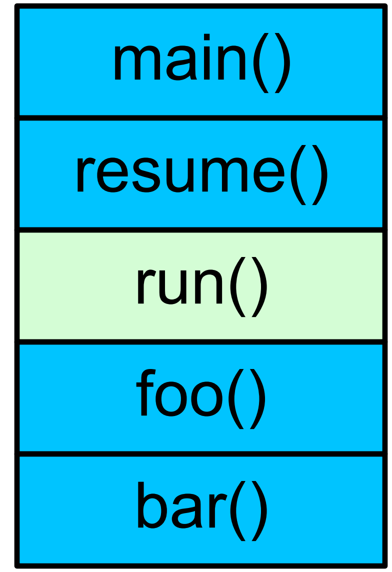
```
foo: A
 cause: foo-2
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack



Console:

```
foo: A
 cause: foo-2
bar: B 3
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()
resume()
run()
foo()

Console:

```
foo: A
 cause: foo-2
bar: B 3
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()
resume()
run()

Console:

```
foo: A
 cause: foo-2
bar: B 3
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()

Console:

```
foo: A
 cause: foo-2
bar: B 3
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()

Console:

```
foo: A
 cause: foo-2
bar: B 3
 cause: null
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()

Console:

```
foo: A
 cause: foo-2
bar: B 3
 cause: null
```

Delimited Continuations

- Rest of the computation from a well-defined outer boundary

```
class Primer extends DelimCont {
  public static void main(String[] args) {
    DelimCont c = new Primer();
    do {
      c.resume();
      println(" cause = " + c.cause());
    } while(!c.completed());
  }
  @Boundary @Override public void run() {
    foo(2);
  }
  public void foo(int x) {
    println("foo: A");
    DelimCont.suspend("foo-" + x);
    bar(x + 1);
  }
  public void bar(int x) {
    println("bar: B " + x);
  }
}
```

Call Stack

main()

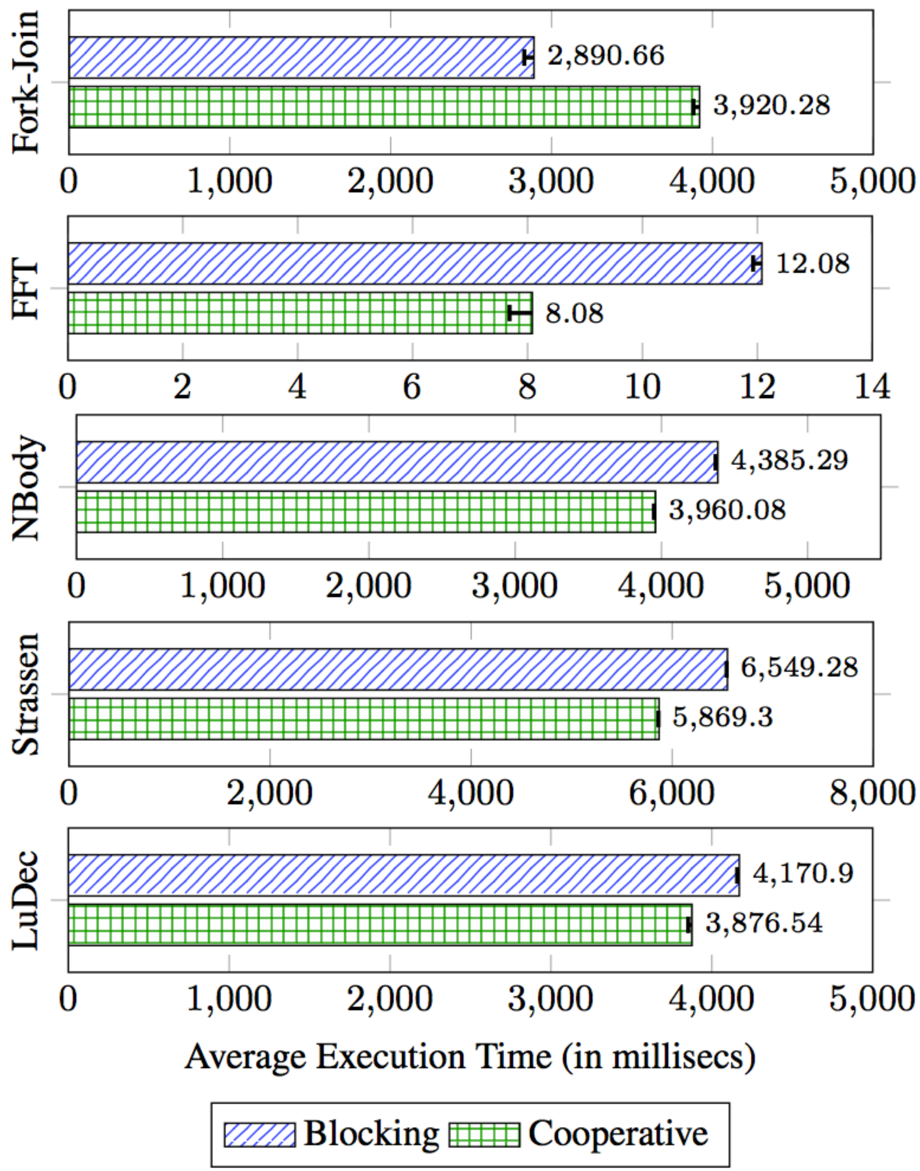
Console:

```
foo: A
 cause: foo-2
bar: B 3
 cause: null
```

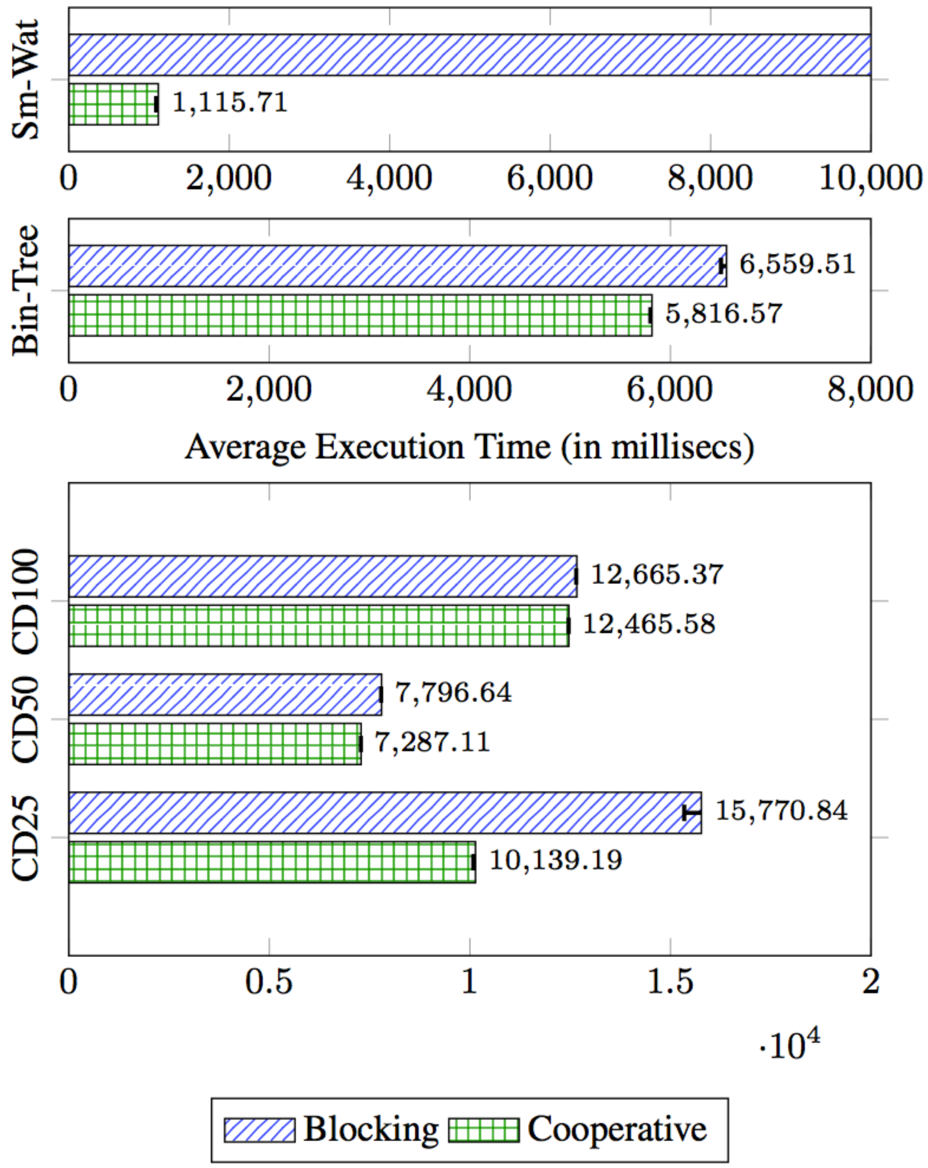
Experimental Results

- 8-core (2 quad-core sockets) 2.83 GHz Intel Xeon Harpertown SMP node
- 16 GB of RAM per node (8 GB per core)
- Red Hat Linux (RHEL 5.8)
- Each core has a 32 kB L1 cache and a 6 MB L2 cache
- Java Hotspot JDK 1.7
- Habanero-Java (HJ) 1.3.1- r33926

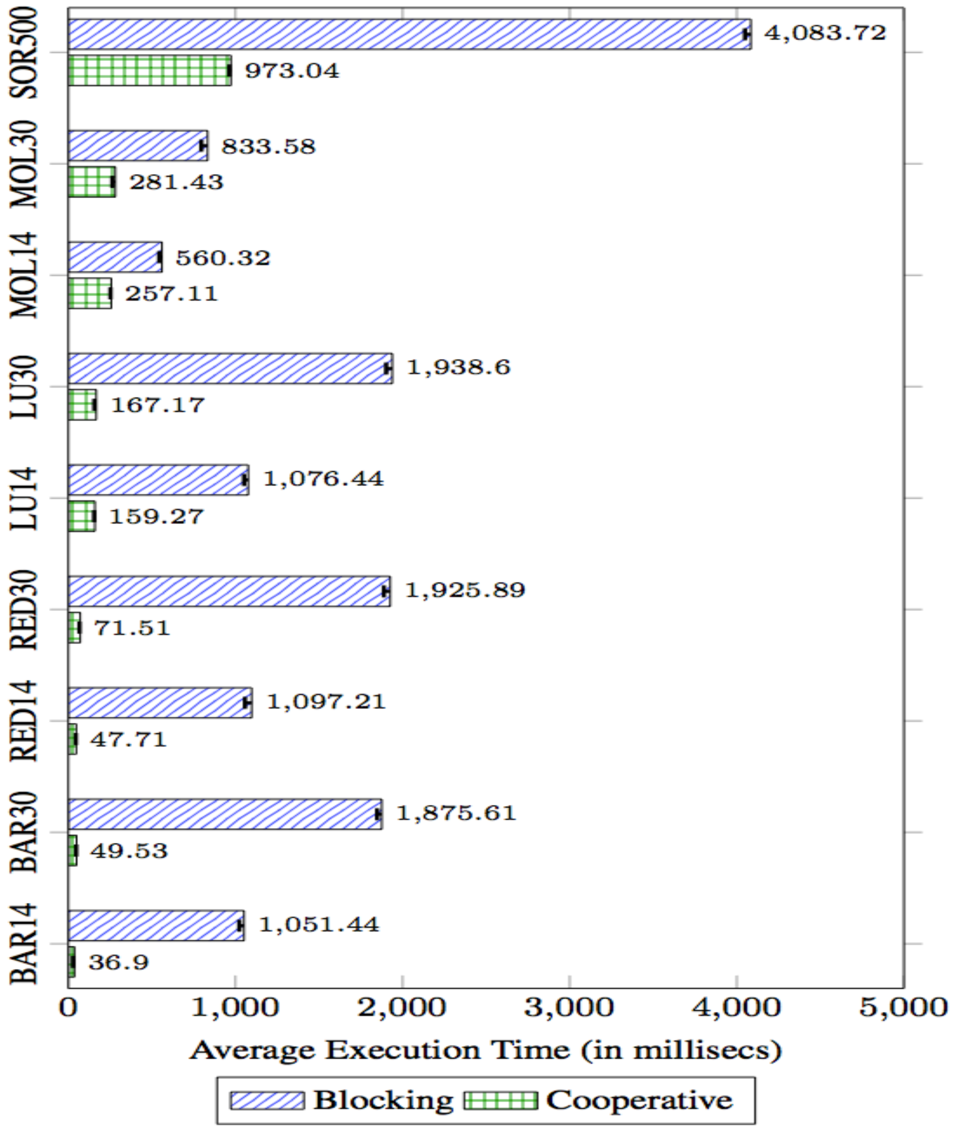
Fork/Join Benchmarks



Future Benchmarks



Phaser Benchmarks



Promising Results

source: http://www.deviantart.com/art/Randomness-5-90424754

Cooperative Runtime – Call Stack

- **Help-first** policy
  - Task has a stack of its own
  - Task can be executed by any of the worker threads
- Task wrapped to form a Delimited Continuation