Efficient, Deterministic, and Deadlock-free Concurrency

Thesis Proposal

Nalini Vasudevan

Columbia University

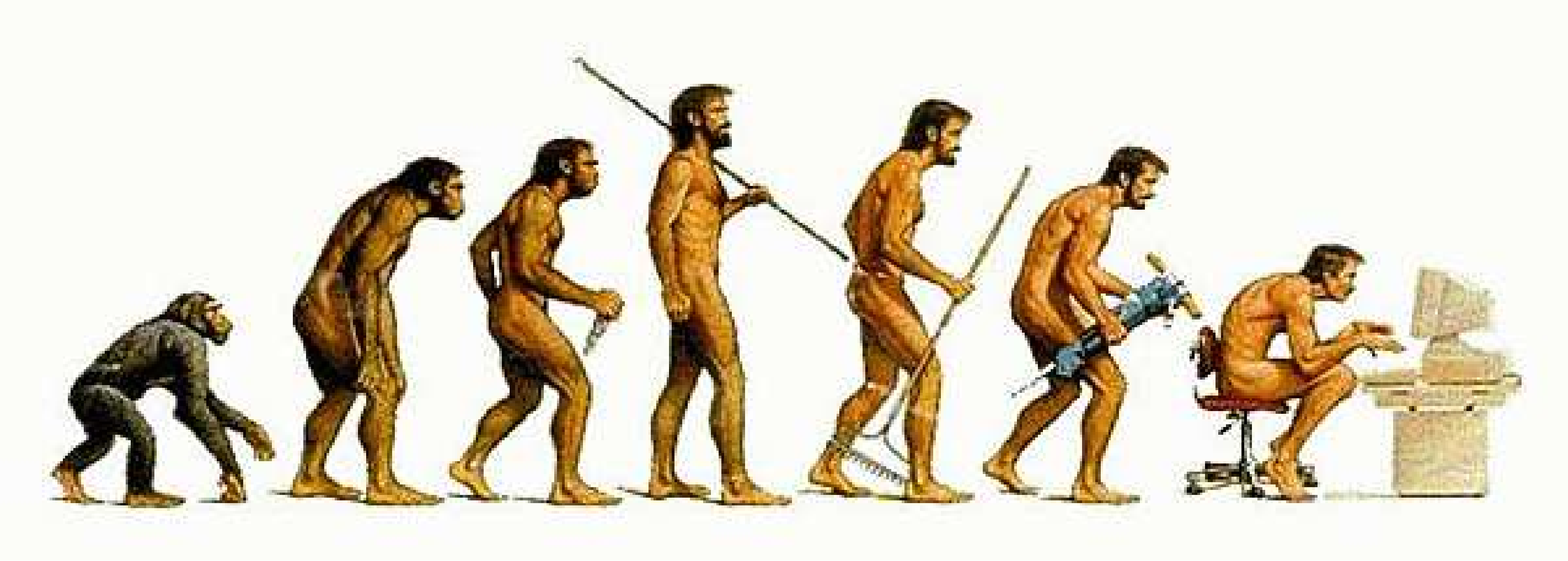
Data Races

```
void f(shared int &a) {

  a++;

}
void g(shared int &b) {

   b = b * 3;

}
main() {
  shared int x = 1;
  spawn f(x)
  spawn g(x);
  sync; /* Wait for f and g to finish */
  print x;
}
```

Non-Determinism

```
lock p;
void f(shared int &a) {
   lock (p);
   a++;
   unlock (p);
}
void g(shared int &b) {
   lock (p);
   b = b * 3;
   unlock (p);
}
main() {
   shared int x = 1;
   spawn f(x)
   spawn g(x);
   sync; /* Wait for f and g to finish */
   print x;
}
```

Motivation



Parallel Computers

Library Support

Parallel Languages

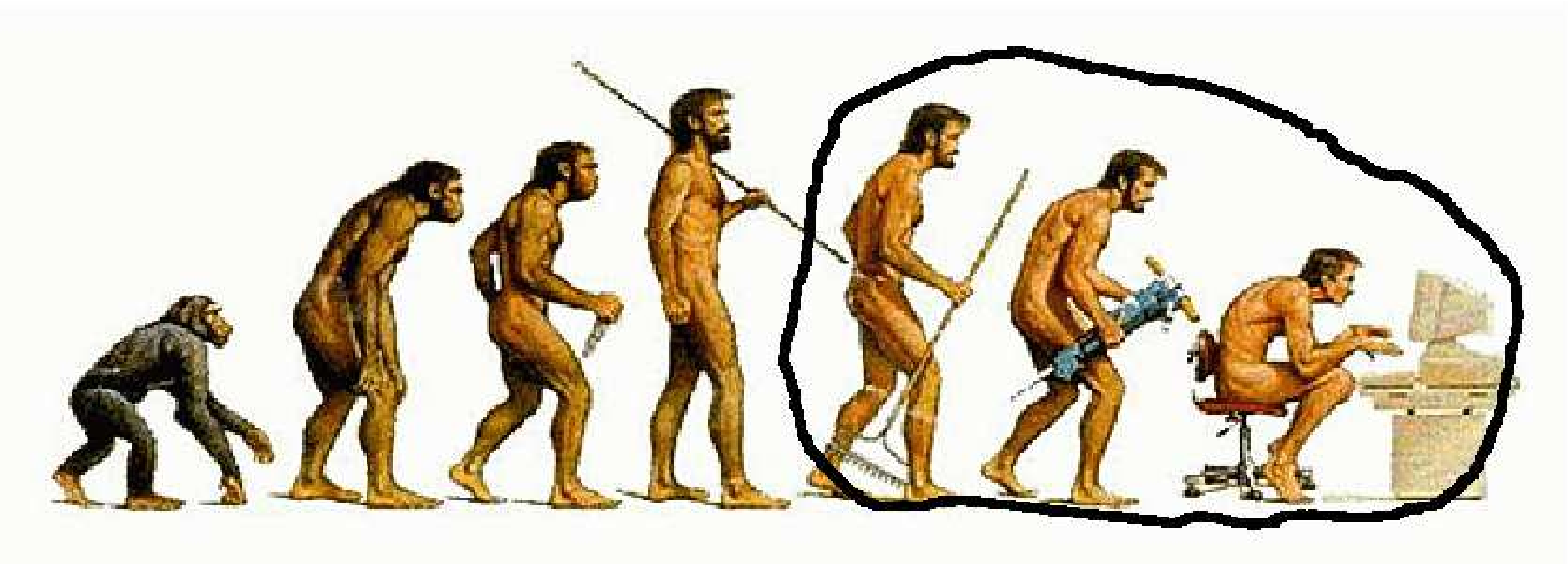
Performance

Data Races

Non-Determinism

Hard-to-Debug

Motivation



Parallel Computers

Library Support

Parallel Languages

Performance

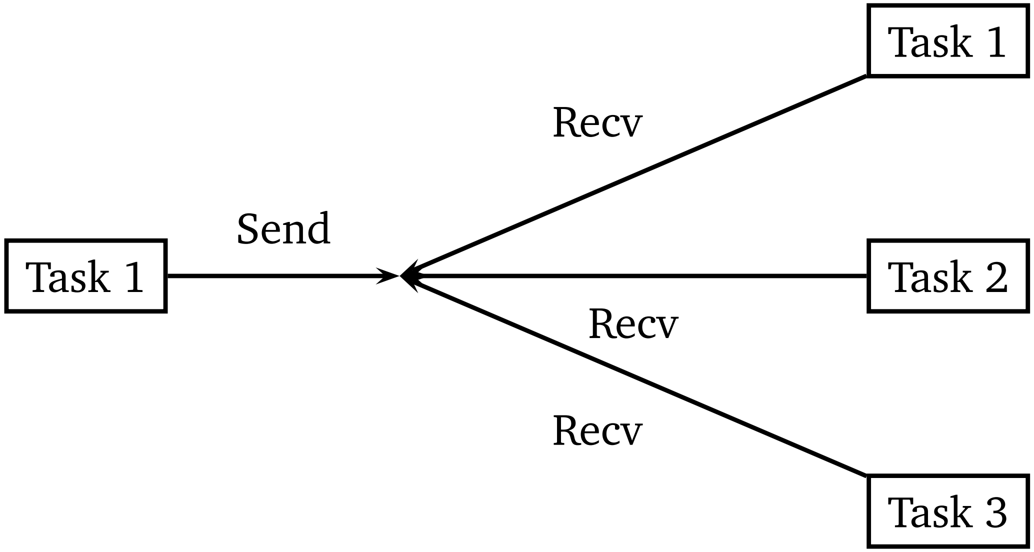
Data Races

Non-Determinism

Hard-to-debug

Determinism: The SHIM Model

- Stands for *Software Hardware Integration Medium*
- Race free, scheduling independent, concurrent model
- Blocking synchronous rendezvous communication



The SHIM Language

An imperative language with familiar C/Java-like syntax

```
int32 gcd(int32 a, int32 b) {
  while (a != b) {
    if (a > b)
      a -= b;
    else
      b -= a;
  }
  return a;
}
```

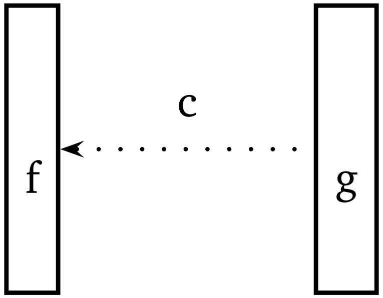
Additional Constructs

$stmt_1$ *par* $stmt_2$	Run $stmt_1$ and $stmt_2$ concurrently
send var	Send on channel *var*
recv var	Receive on channel *var*

Communication

- Blocking: wait for all processes connected to *c*

```
void f(chan int a) { // a is a copy of c
  a = 3; // change local copy
  recv a; // receive (wait for g)
  // a now 5
}
void g(chan int &b) { // b is an alias of c
   b = 5; // sets c
   send b; // send (wait for f)
  // b now 5
}
void main() {
  chan int c = 0;
  f(c); par g(c);
}
```



Overview

Timeline	Work	Progress
Spring 2007	Compiling SHIM to Shared Memory Multicores	DATE 2008
Summer 2007	A SHIM-like Library in Haskell	IPDPS 2008
Fall 2007	Static Deadlock Detection for SHIM	MEMOCODE 2008
Spring 2008	Compiling SHIM to Heterogeneous Multicores	SAC 2009
Summer 2008	Analysis and Specialization of Clocks in X10	CC 2009
Fall 2008	Buffer Sharing in SHIM Programs	MEMOCODE 2009
Spring 2009	Compositional Deadlock Detection	EMSOFT 2009
Fall 2009	Overview and Ideas for Thesis	IPDPS Workshop 2010
Spring 2010	Deterministic Concurrency in X10	In progress at IBM
Fall 2010	Run-time deadlock detection in SHIM	To do
Spring 2011	Thesis Writing and Defense	To do
After graduation	A Determinizing Compiler	PLDI WACI 2009

Compiling to Quad-Core [DATE 2008]

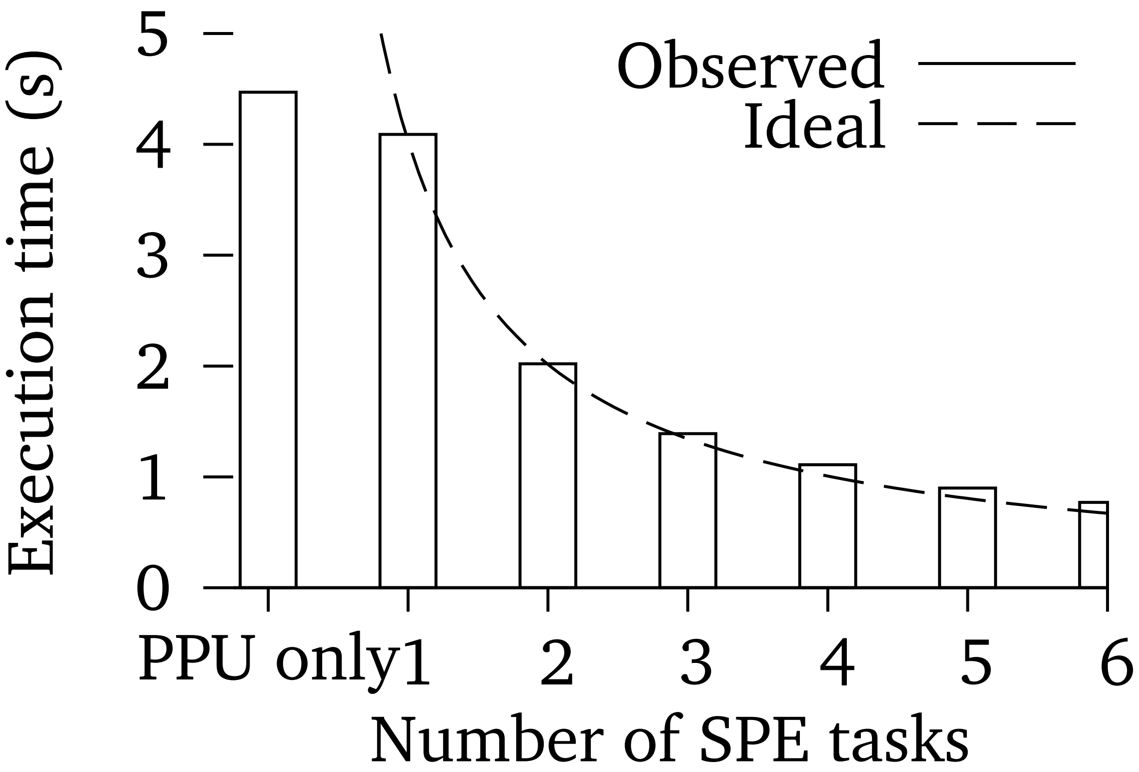
- Intel Quad Core Machine
- Each task mapped to a pthread
- Example: JPEG decoder

Cores	Tasks	Time	Speedup
1	Sequential	25s	1.0
4	3	16	1.6
4	4	9.3	2.7
4	5	8.7	2.9
4	6	8.2	3.05
4	7	8.6	2.9

Run on a 20 MB 21600 × 10800 image that expands to 668 MB.

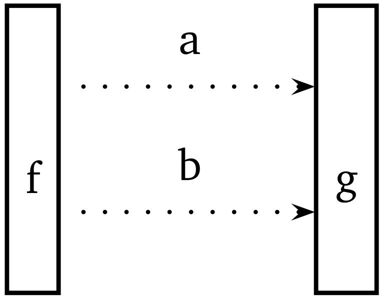
Compiling to Cell [SAC 2009]

- Generated Code for a Heterogeneous Multicore
- Computationally intensive tasks mapped on the SPUs
- Example: FFT



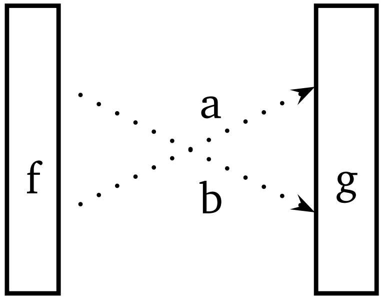
More Examples in SHIM

```
void main() {
 chan int a, b;
 {
     // Task 1
     send a = 5; // send a
     send b = 10; // send b
 } par {
     // Task 2
     int c;
     recv a; // recv a
     recv b; // recv b
     c = a + b;
  }
}
```

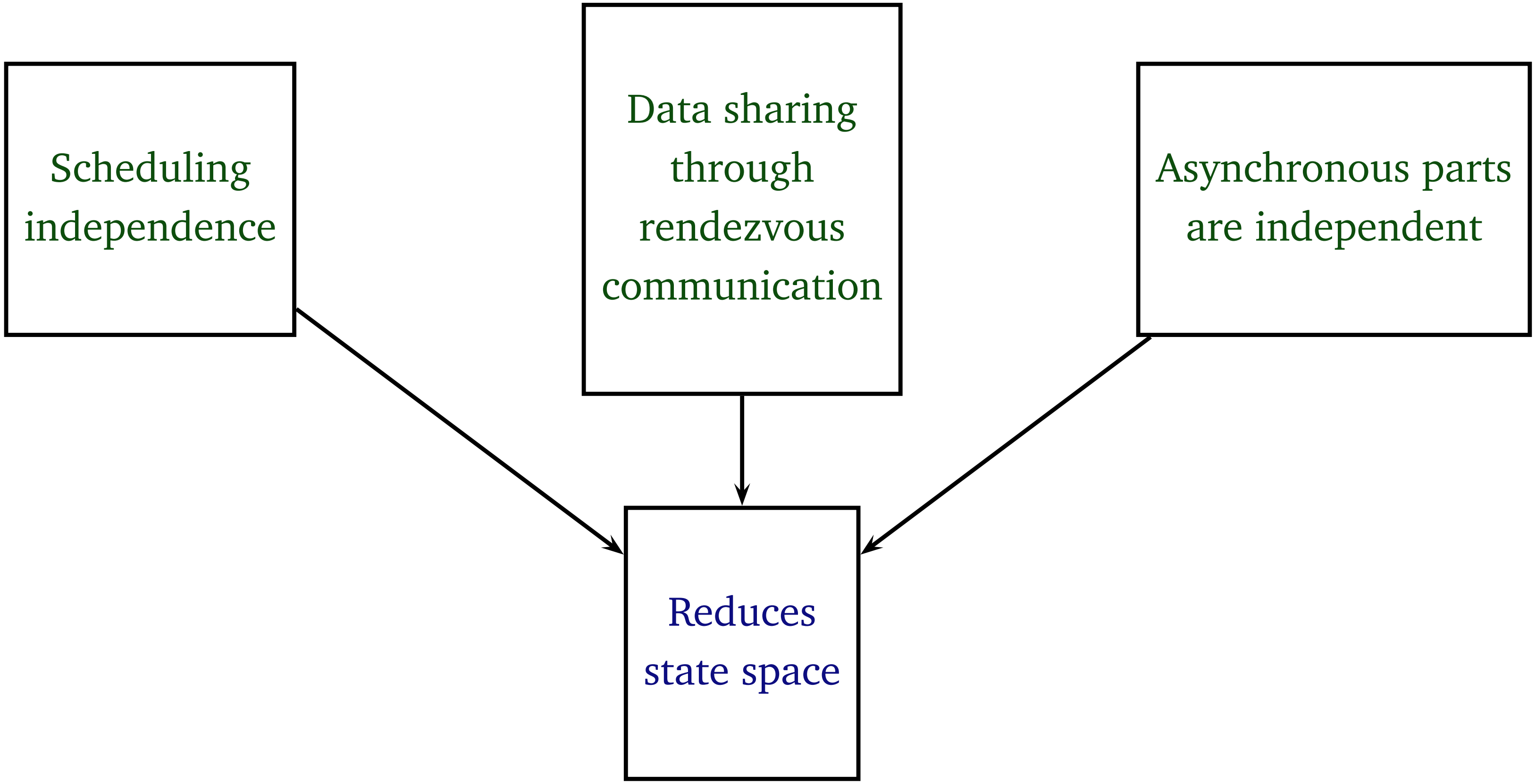


The Problem

```
void main() {
 chan int a, b;
 {
     // Task 1
     send a = 5; // send a
     send b = 10; // send b
 } par {
     // Task 2
     int c;
     recv b; // recv b
     recv a; // recv a
     c = a + b;
  }
}
```



SHIM design for static analysis



Scheduling independence

Data sharing through rendezvous communication

Asynchronous parts are independent

Reduces state space

Deadlocks in SHIM

- Why SHIM? No data races.
- Deadlocks in SHIM are deterministic (always reproducible).
- SHIM's philosophy: It prefers deadlocks to races.

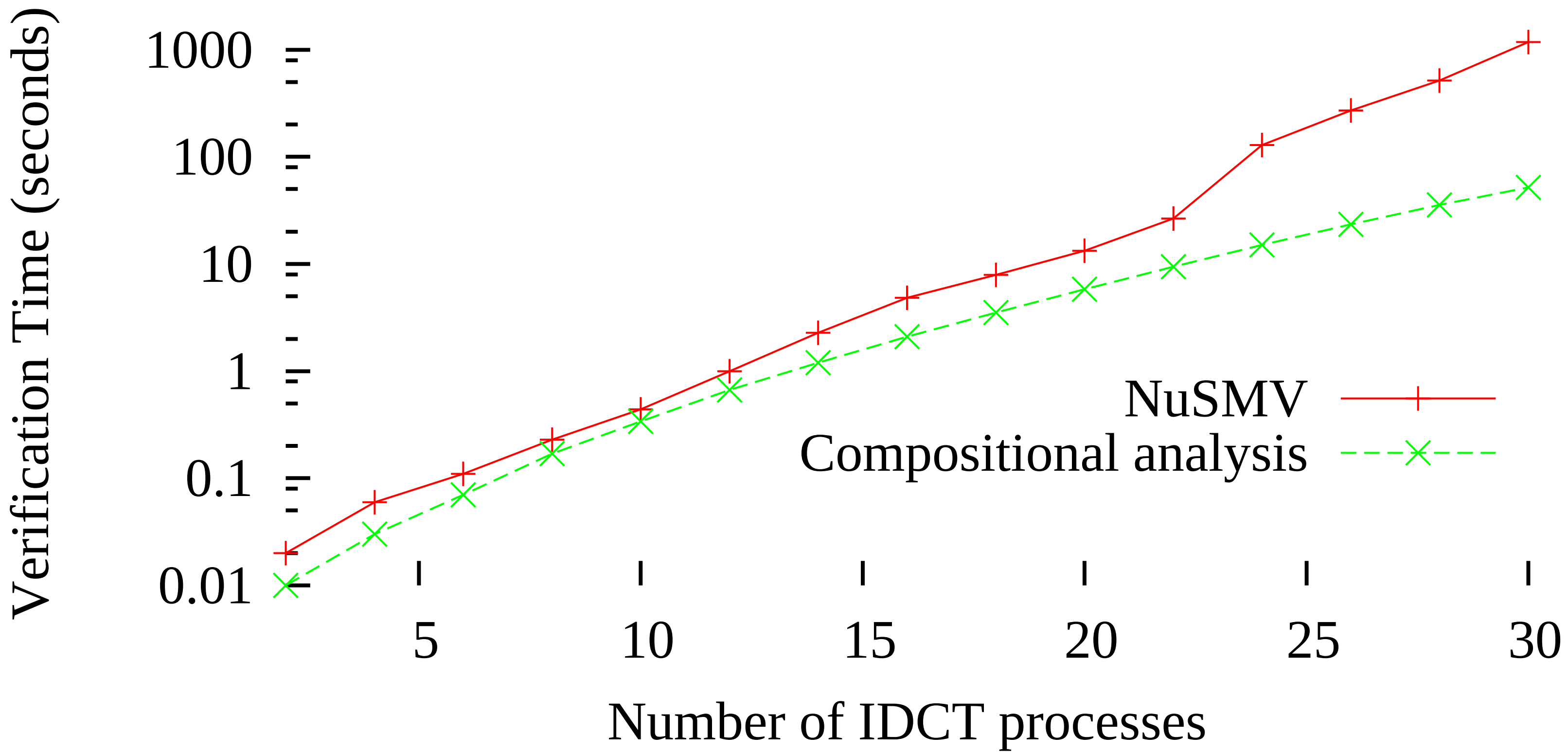
Just pick one schedule

Deadlock Detection [MEMOCODE 2008]

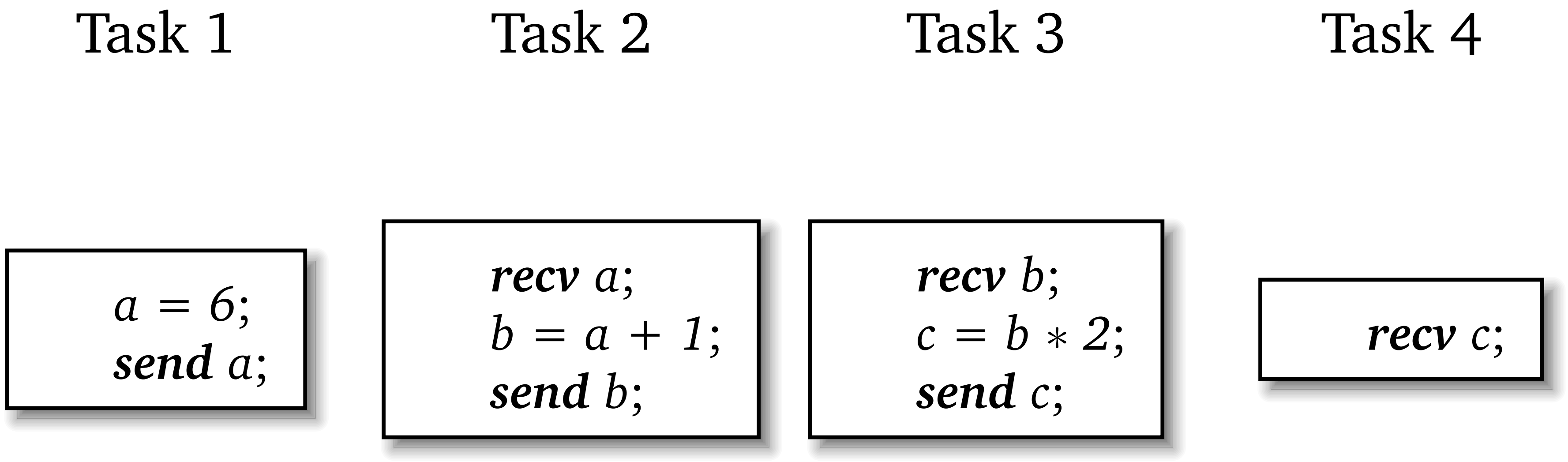
- Using NuSMV

Example	Lines	Channels	Tasks	Result	Runtime	Memory
Source-Sink	35	2	11	No Deadlock	0.2 s	3.9 MB
Pipeline	30	7	13	No Deadlock	0.1	2.0
Prime Sieve	35	51	45	No Deadlock	1.7	25.4
Berkeley	40	3	11	No Deadlock	0.2	7.2
FIR Filter	100	28	28	No Deadlock	0.4	13.4
Bitonic Sort	130	65	167	No Deadlock	8.5	63.8
Framebuffer	220	11	12	No Deadlock	1.7	11.6
JPEG Decoder	1025	7	15	No Deadlock	0.9	85.6

Compositional Deadlock Detection



Buffer Sharing



Task 1

$a = 6;$
send $a;$

Task 2

recv $a;$
$b = a + 1;$
send $b;$

Task 3

recv $b;$
$c = b * 2;$
send $c;$

Task 4

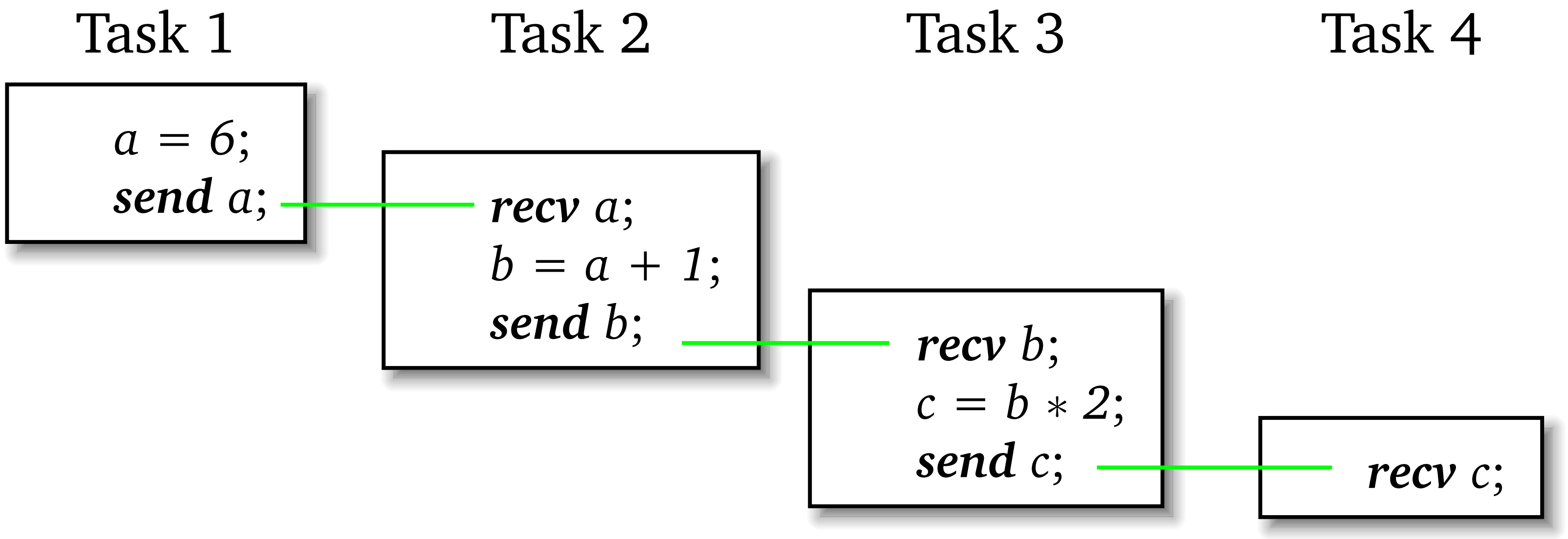
recv $c;$

Buffer Sharing

Task 1	Task 2	Task 3	Task 4
$a = 6$; ***send*** a;	***recv*** a; $b = a + 1$; ***send*** b;	***recv*** b; $c = b * 2$; ***send*** c;	***recv*** c;

- Use rendezvous model of communication

Buffer Sharing



Task 1

a = *6*;
send *a*;

Task 2

recv *a*;
b = *a* + *1*;
send *b*;

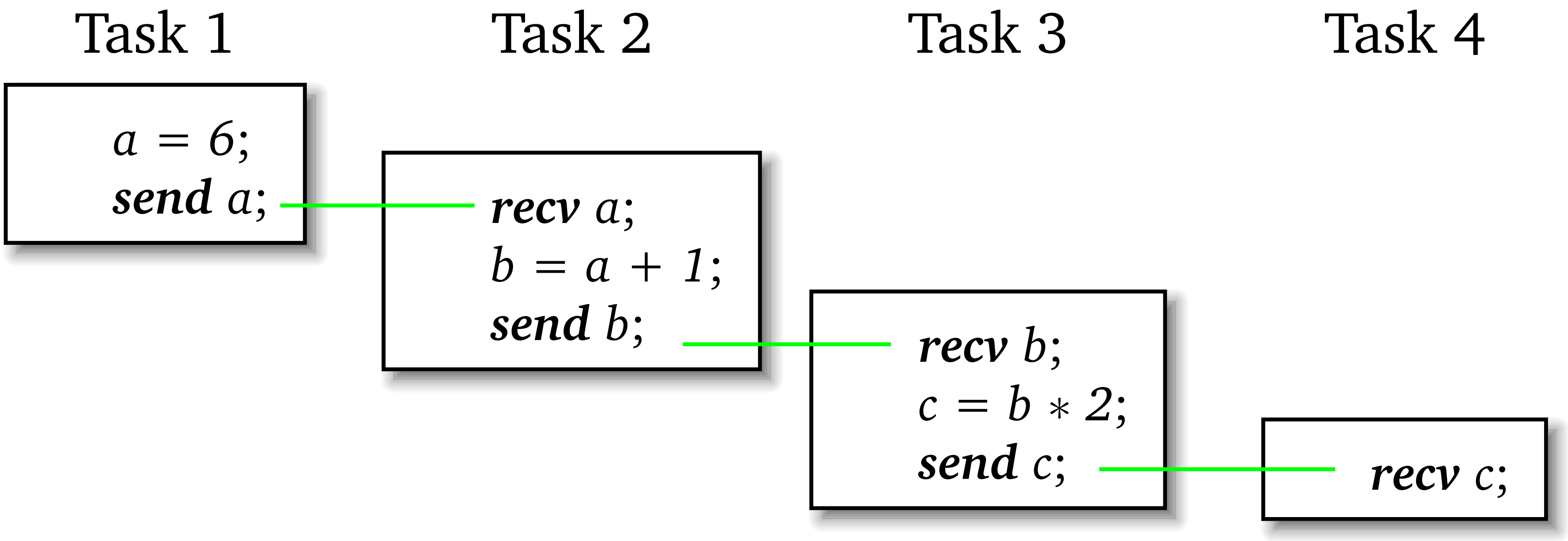
Task 3

recv *b*;
c = *b* ∗ *2*;
send *c*;

Task 4

recv *c*;

Buffer Sharing



Task 1

$a = 6$;
send a;

Task 2

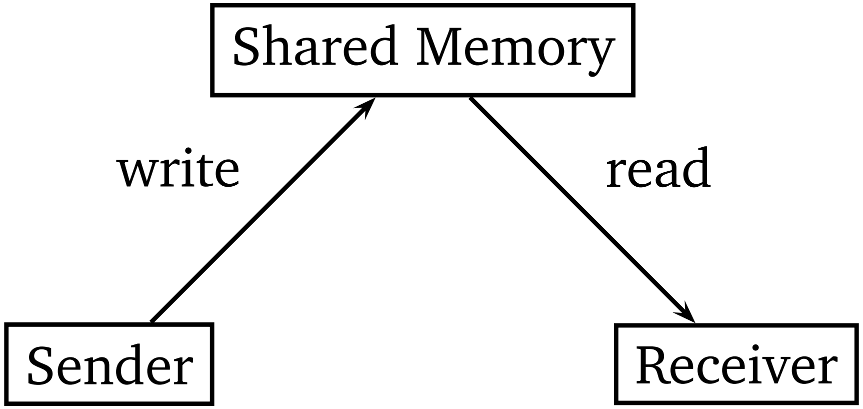
recv a;
$b = a + 1$;
send b;

Task 3

recv b;
$c = b * 2$;
send c;

Task 4

recv c;



Shared Memory

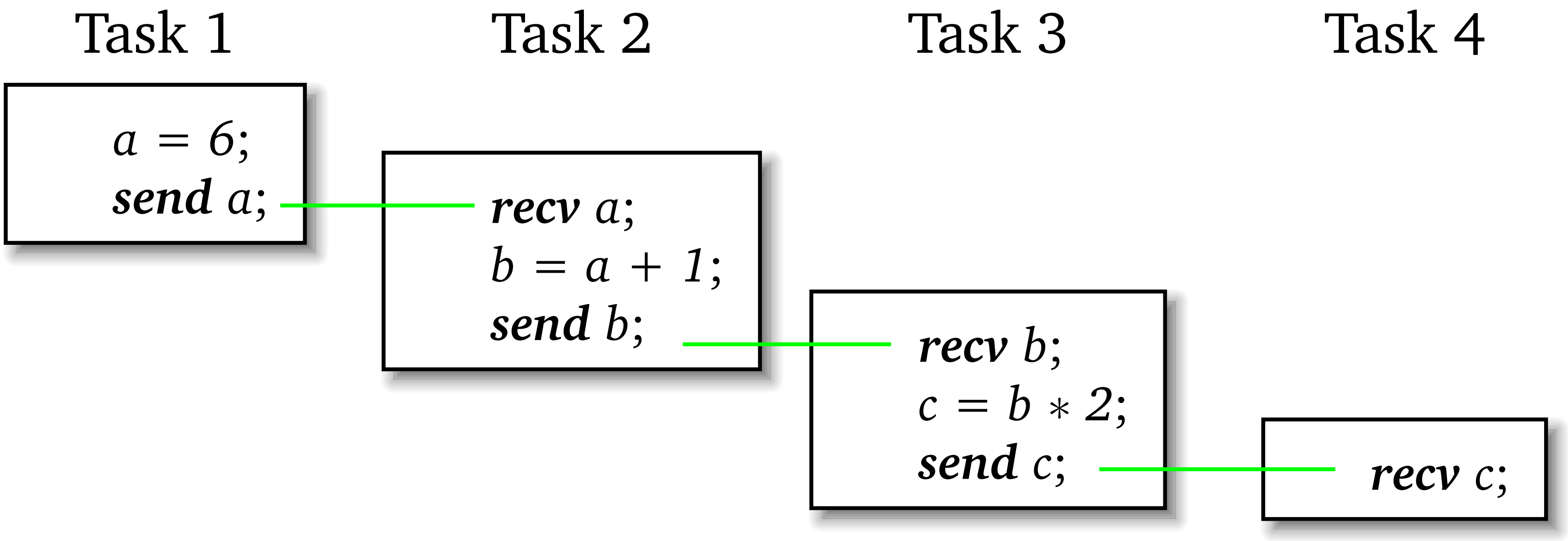
write

read

Sender

Receiver

Buffer Sharing



Task 1

a = *6*;
send *a*;

Task 2

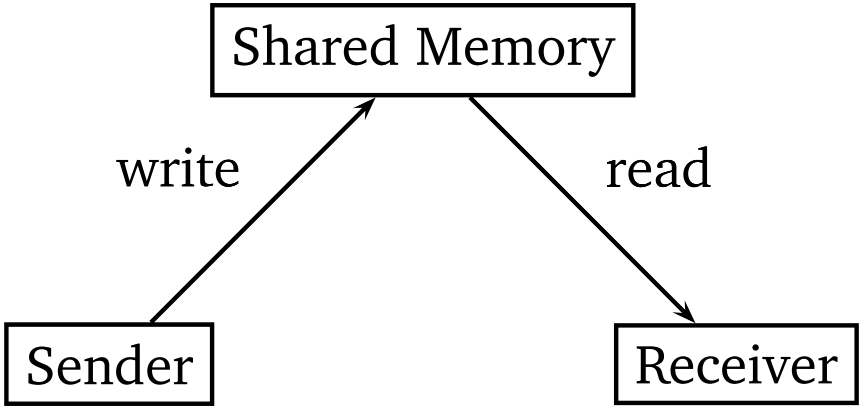
recv *a*;
b = *a* + *1*;
send *b*;

Task 3

recv *b*;
c = *b* ∗ *2*;
send *c*;

Task 4

recv *c*;



Shared Memory

write

read

Sender

Receiver

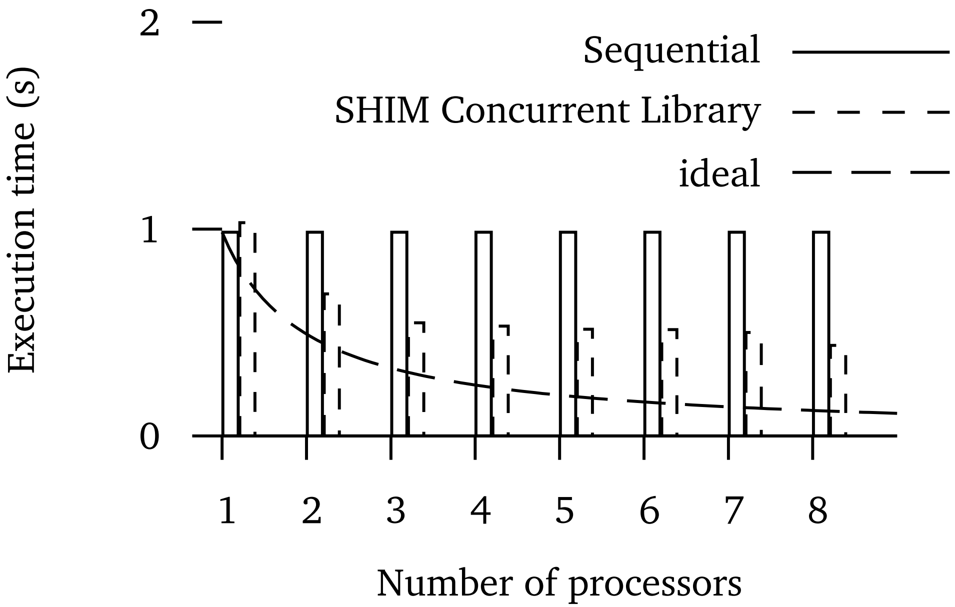
- a, b, and c can share buffers

Buffer Reduction: Results [MEMOCODE 2009]

Example	Lines	Channels	Tasks	Bytes Saved	Buffer Reduction	Runtime
Source-Sink	35	2	11	4	50 %	0.1 s
Pipeline	35	5	9	16388	25	0.1
Bitonic Sort	35	5	13	12	60	0.1
Prime Sieve	40	5	16	12	60	0.5
Berkeley	40	3	11	4	33.33	0.6
FIR Filter	110	28	28	52	46.43	13.8
Framebuffer	185	11	16	28	0.002	1.3
FFT	230	14	15	344068	50	0.6
JPEG Decoder	1020	7	15	772	50.13	1.8

SHIM as a Library [IPDPS 2009]

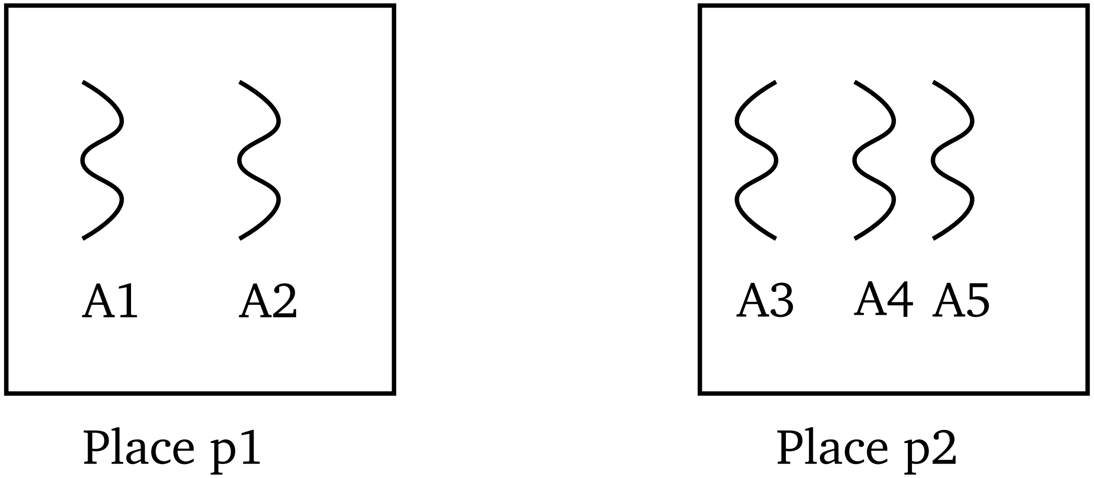
- Implemented in Haskell
- APIs that mimic *par*, *send* and *recv*
- Programmer's job to use the library correctly
- Example: Systolic Filter



The X10 Programming Language

The X10 Programming Language

- Concurrent programming model
- Activities are light weight threads
- Places are distributed memory locations



X10: asyncs and clocks

- Activities created using *async*

```
async {                        async (p2) {
/* executed locally */         /* executed at p2 */
}                              }
```

- Clocks are used for synchronization

```
clock c = new clock();
async clocked (c) {
  a = 1;
  c.next();
}
async clocked (c) {
  c.next();
  a = 2;
}
```

Special Case Optimization [CC 2009]

- Common patterns of clocks
  - Example: A clock is used locally (in one place)
- Used specialized implementation for that pattern

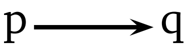
Example	Clocks	Lines	Speed	Analysis Time	
			Up	Base	NuSMV
Linear Search	1	35	35.2%	33.5s	0.4s
Relaxation	1	55	87.6	6.7	0.3
All Reduction Barrier	1	65	1.5	27.2	0.1
Pascal’s Triangle	1	60	20.5	25.8	0.4
Prime Number Sieve	1	95	213.9	34.7	0.4
N-Queens	1	155	1.3	24.3	0.5
LU Factorization	1	210	5.7	20.6	0.9
MolDyn JGF Bench.	1	930	2.3	35.1	0.5
Pipeline	2	55	31.4	7.5	0.5
Edmiston	2	205	14.2	29.9	0.5

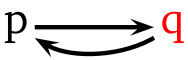
Future Work

- Resolving deadlocks in SHIM at runtime
- Deterministic, Deadlock-free Constructs

Runtime Deadlock Resolver for SHIM

```
void main() {
 chan int a = 1, b = 1;
 {
     // Task p
     send a = 5; // send a
     recv b; // send b
 } par {
     // Task q
     int c;
     send b = 10; // recv b
     recv a; // recv a
     c = a + b;
  }
}
```

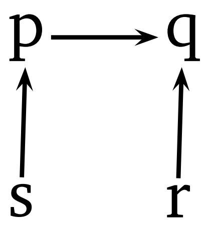




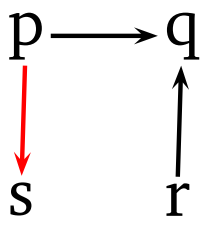
Runtime Deadlock Resolver for SHIM

- Generally, cycle detection algorithm is exponential
- SHIM's semantics makes it simpler

A possible SHIM network



An impossible SHIM network



Deterministic, Deadlock-Free Constructs

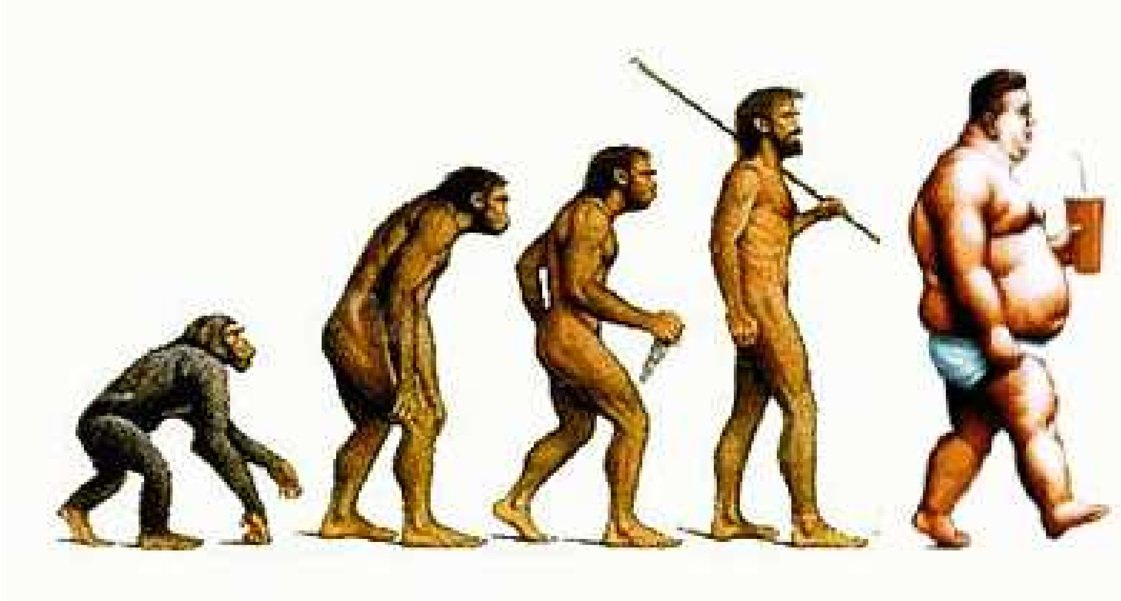
```
void f(shared int &a) {
 /* a is 1 */
  a = 3;
 /* a is 3 , x is still 1 */
  next; /* Apply reduction operator */
 /* a is now 8, x is 8 */
}

void g(shared int &b) {
 /* b is 1 */
  b = 5;
 /* b is 5, x is still 1 */
  next; /* Apply reduction operator */
 /* b is now 8, x is 8 */
}
```

```
void h (shared int &c) {
 /* c is 1 , x is still 1 */
  next;
 /* c is now 8, x is 8 */
}

main() {
  shared int (+) x = 1;
  /* If there are multiple writers, reduce
    using the + reduction operator */
  spawn f(x);
  spawn g(x);
  spawn h(x);
  sync;
  /* x is 8 */
}
```

Related Work

- Programming Models
  - Esterel
  - StreamIt
  - Cilk
  - X10
- Tools
  - Deterministic Replay Systems
  - Kendo
  - DMP

Long Term Goal [PLDI'09 Fun Ideas and Thoughts]



Library Support

Performance

Parallel Computers

Parallel Languages

A Determinizing Compiler!