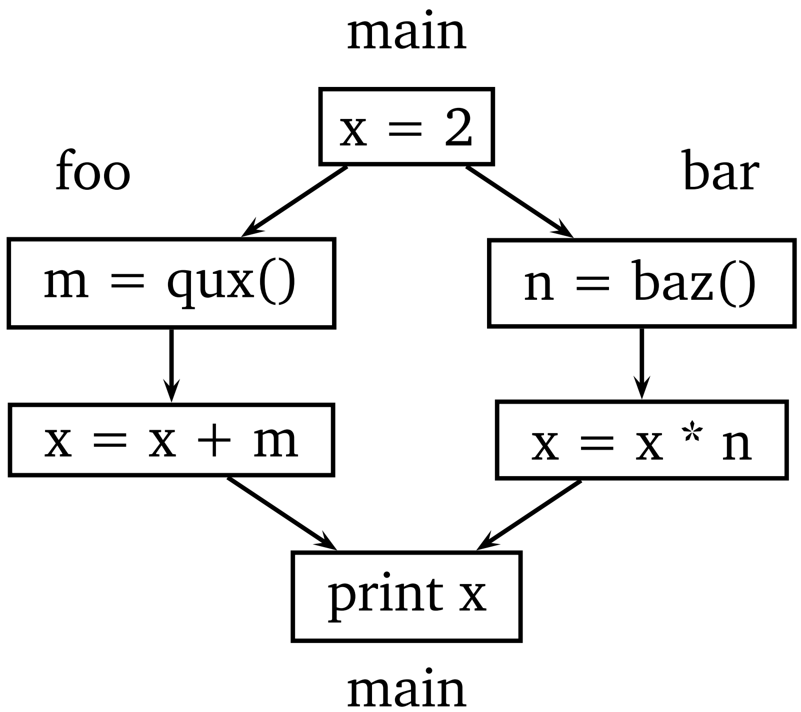
Efficient, Deterministic, and Deadlock-free Concurrency

Nalini Vasudevan

Columbia University

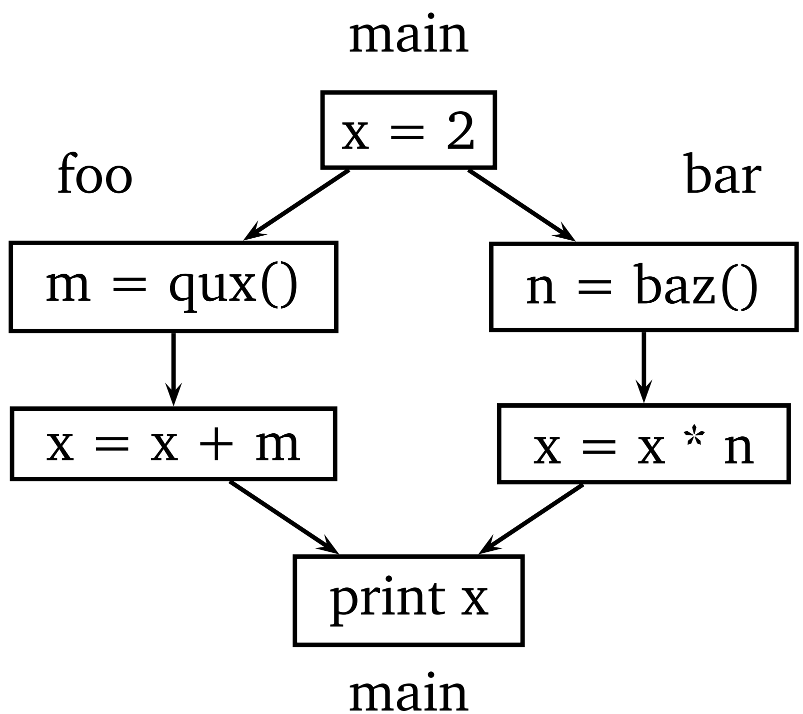
Data Races

```
int x;
foo(){
    int m;
    m = qux();
    x = x + m;
 }
bar(){
    int n;
    n = baz();
    x = x * n;
 }
 main() {
     x = 2;
     spawn foo();
     spawn bar();
     sync;
     print(x);
}
```



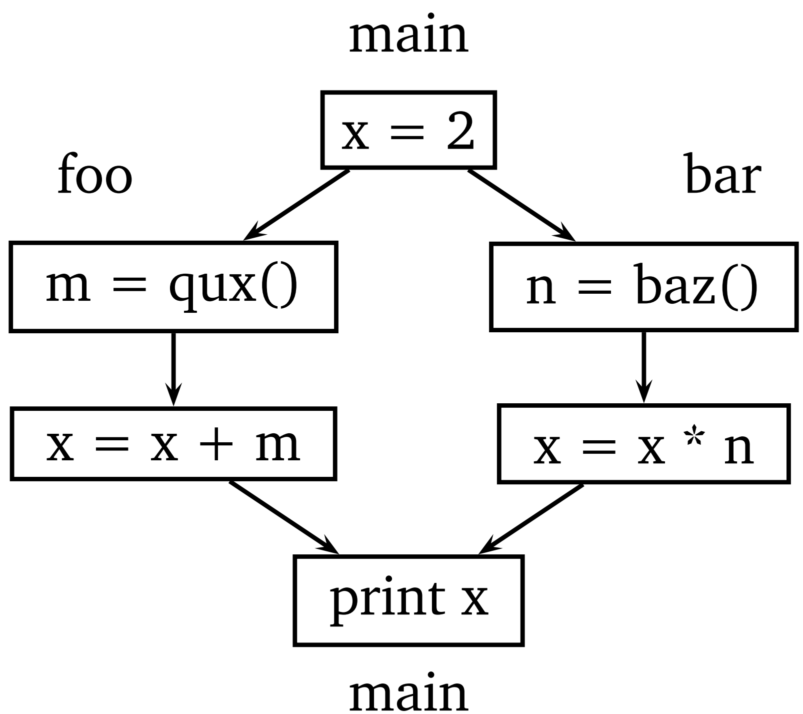
Eliminating Data Races

```
int x;
foo(){
    int m;
    m = qux();
    lock(x);
      x = x + m;
    unlock(x);
  }
bar(){
    int n;
    n = baz();
    lock(x);
      x = x * n;
    unlock(x);
  }
  main() {
     x = 2;
     spawn foo();
     spawn bar();
     sync;
     print(x);
}
```



Eliminating Data Races

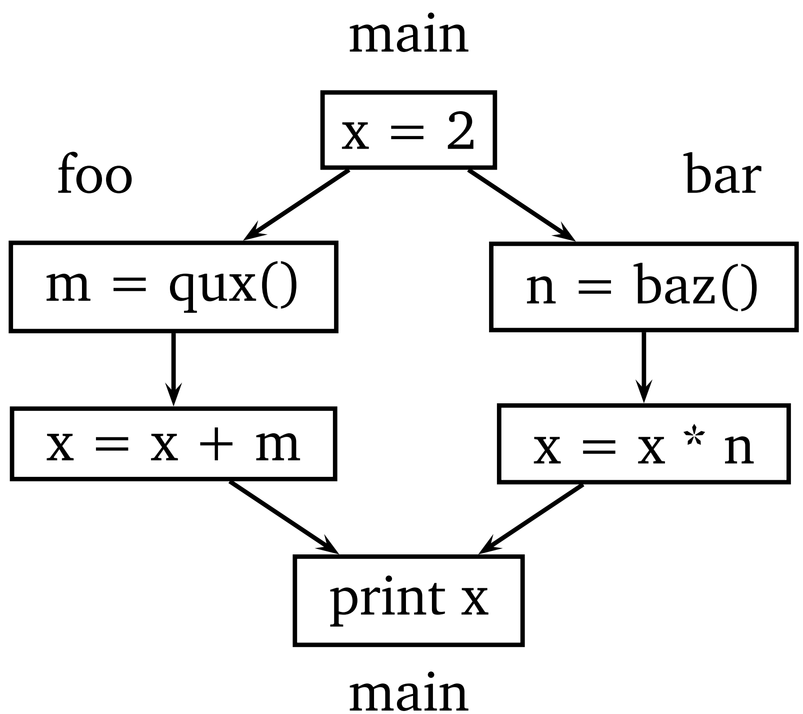
```
int x;
foo(){
    int m;
    m = qux();
    lock(x);
      x = x + m;
    unlock(x);
 }
bar(){
    int n;
    n = baz();
    lock(x);
      x = x * n;
    unlock(x);
 }
 main() {
    x = 2;
    spawn foo();
    spawn bar();
    sync;
    print(x);
}
```



if m = n = 2

Eliminating Data Races

```
int x;
foo(){
    int m;
    m = qux();
    lock(x);
      x = x + m;
    unlock(x);
  }
bar(){
    int n;
    n = baz();
    lock(x);
      x = x * n;
    unlock(x);
  }
  main() {
     x = 2;
     spawn foo();
     spawn bar();
     sync;
     print(x);
}
```



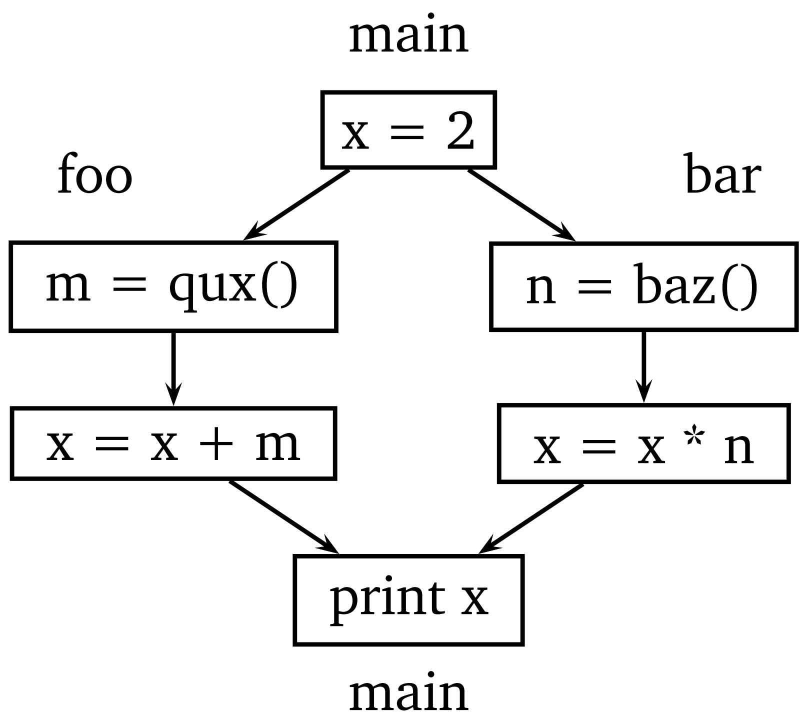
if m = n = 2

x = (2 + 2) * 2 = 8

x = (2 * 2) + 2 = 6

Eliminating Data Races

```
int x;
foo(){
    int m;
    m = qux();
    lock(x);
      x = x + m;
    unlock(x);
  }
bar(){
    int n;
    n = baz();
    lock(x);
      x = x * n;
    unlock(x);
  }
  main() {
     x = 2;
     spawn foo();
     spawn bar();
     sync;
     print(x);
}
```



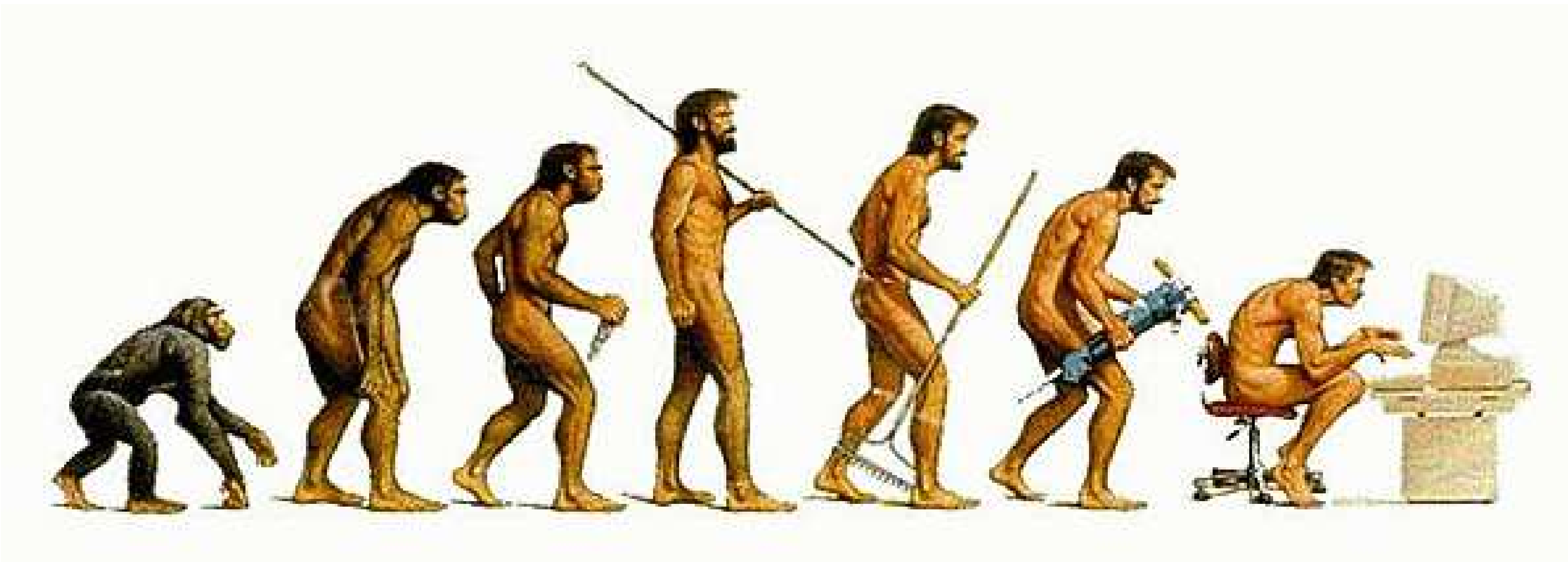
if $m = n = 2$

$x = (2 + 2) * 2 = 8$

$x = (2 * 2) + 2 = 6$

Non-determinism

Motivation



Library Support

Performance

Non-Determinism

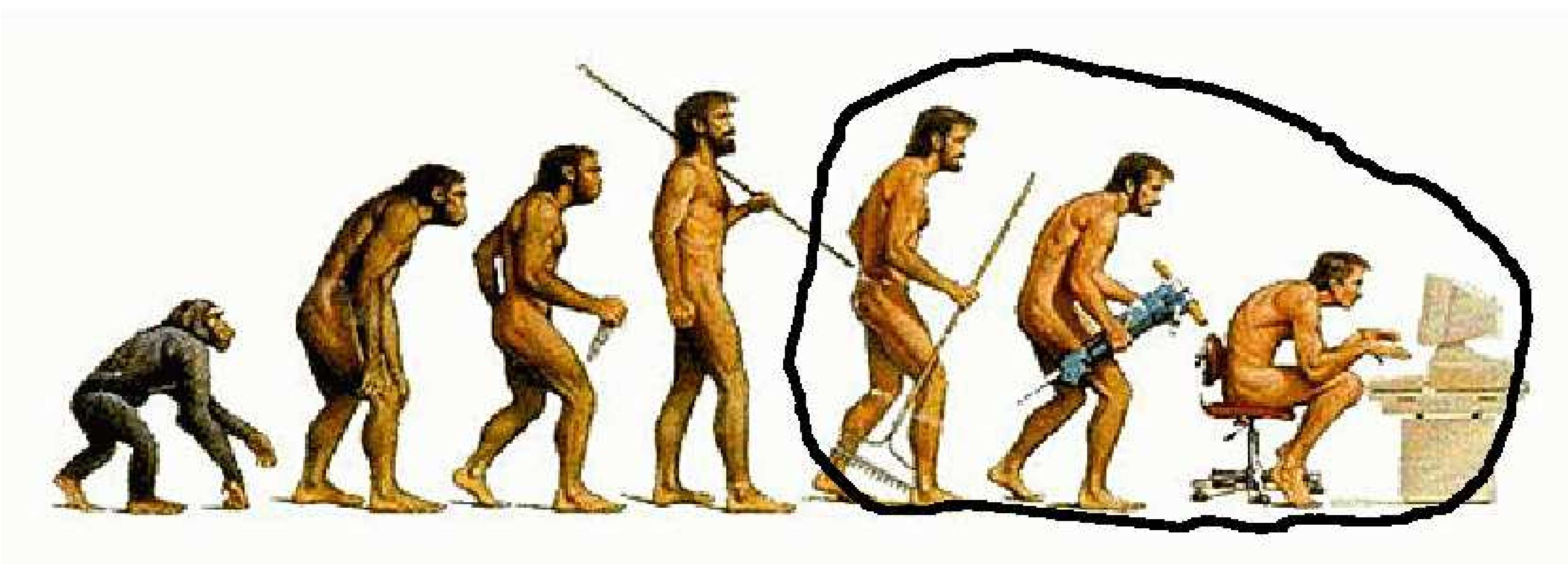
Parallel Computers

Parallel Languages

Data Races

Hard-to-Debug

Motivation



Library Support

Performance

Non-Determinism

Parallel Computers

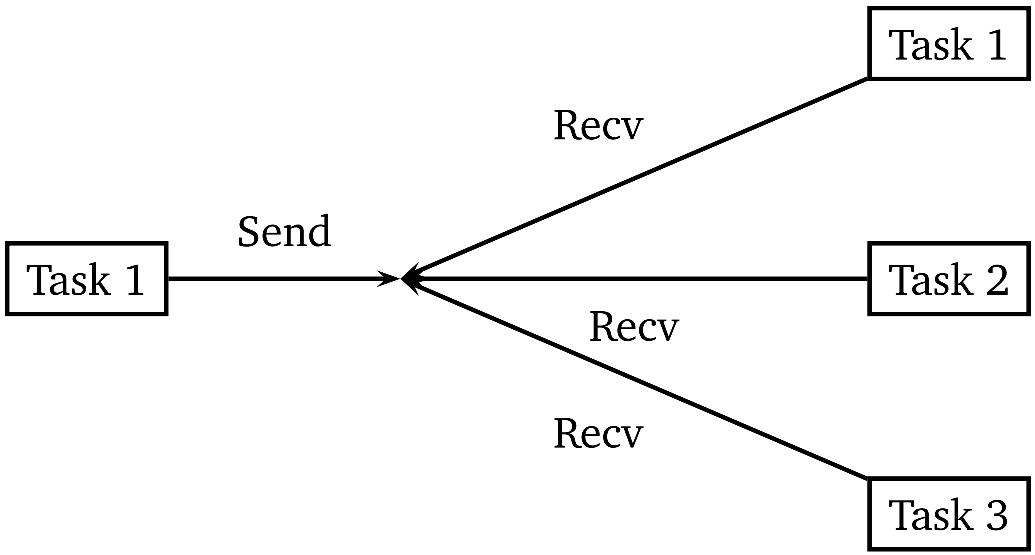
Parallel Languages

Data Races

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

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

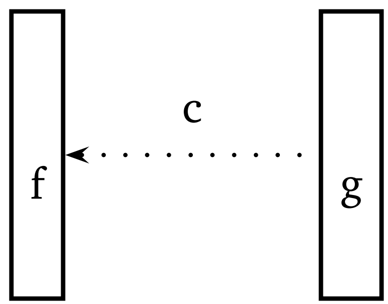
Additional Constructs

$stmt_1$ *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

How	Compiling SHIM to Shared Memory Multicores	DATE 2008
efficient	Compiling SHIM to Heterogeneous Multicores	SAC 2009
is	Compilation techniques for SHIM	SES 2010
determinism?	Simple and Fast Biased Locks	PACT 2010
	A SHIM-like Library in Haskell	IPDPS 2008
Does determinism	Static Deadlock Detection for SHIM	MEMOCODE 2008
guarantee	Compositional Deadlock Detection	EMSOFT 2009
deadlock-freedom?	Run-time deadlock detection in SHIM	HotPar 2010
Does determinism	Buffer Sharing in SHIM Programs	MEMOCODE 2009
simplify	Buffer Sharing in Rendezvous Programs	TCAD 2010
verification?	Analysis and Specialization of Clocks in X10	CC 2009
What	Overview and Shortcomings of SHIM	IPDPS Proc. 2010
next?	A Determinizing Compiler	PLDI WACI 2009

Compiling to Quad-Core [DATE 2008]

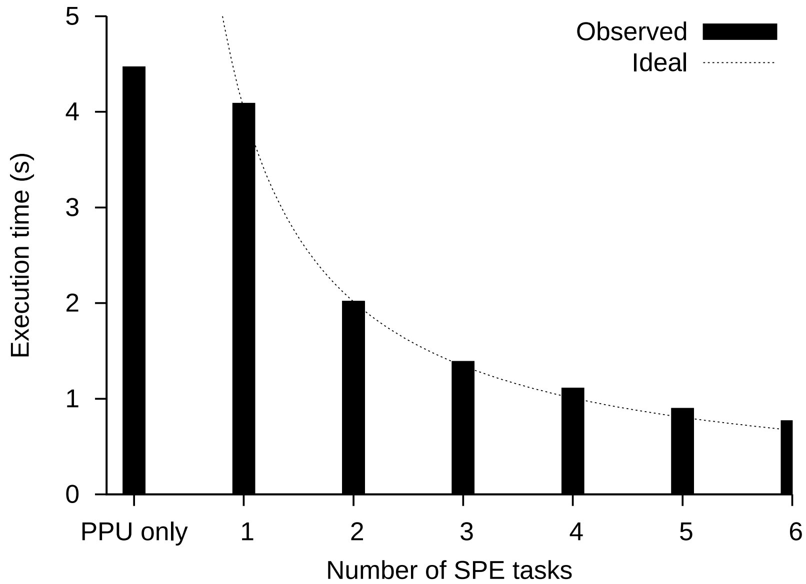
- Intel Quad Core Machine
- Each task mapped to a pthread
- Example: JPEG decoder

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 $\times$ 10800 image that expands to 668 MB.

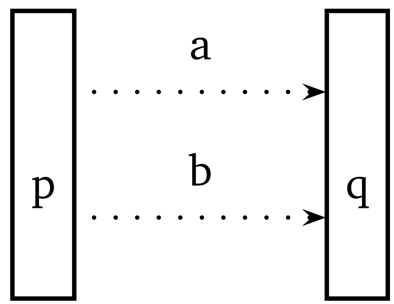
Compiling to Cell [SAC 2009]

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



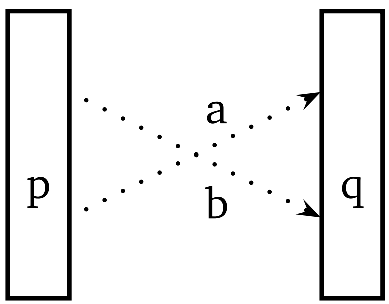
More Examples in SHIM

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



The Problem

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



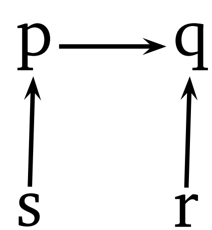
Runtime Deadlock Detection [HotPar 2010]

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

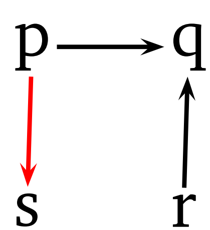
Runtime Deadlock Detection

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

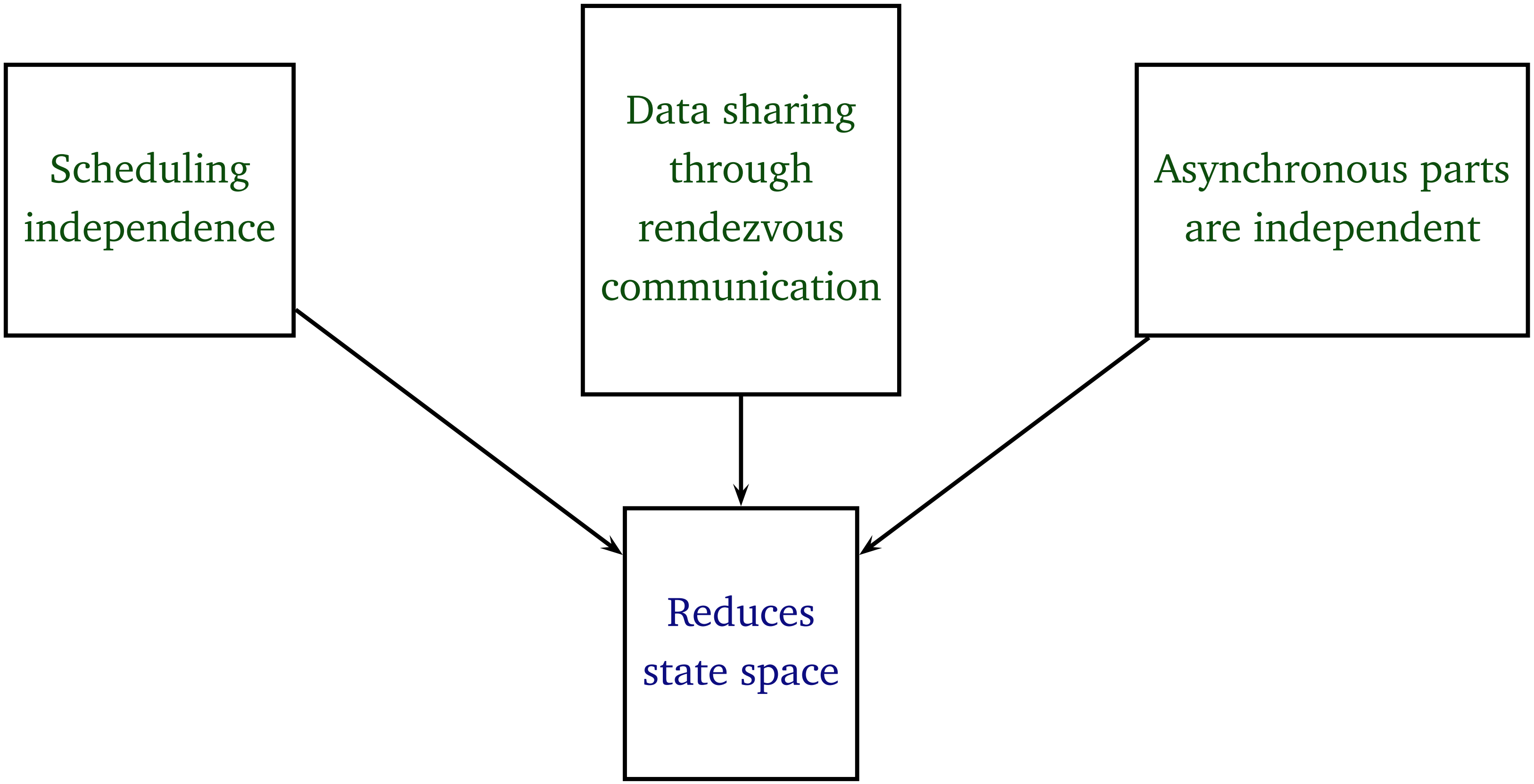
A possible SHIM network



An impossible SHIM network



Static Deadlock Detection



Just pick one schedule

Deadlock Detection [MEMOCODE 2008]

- Using NuSMV

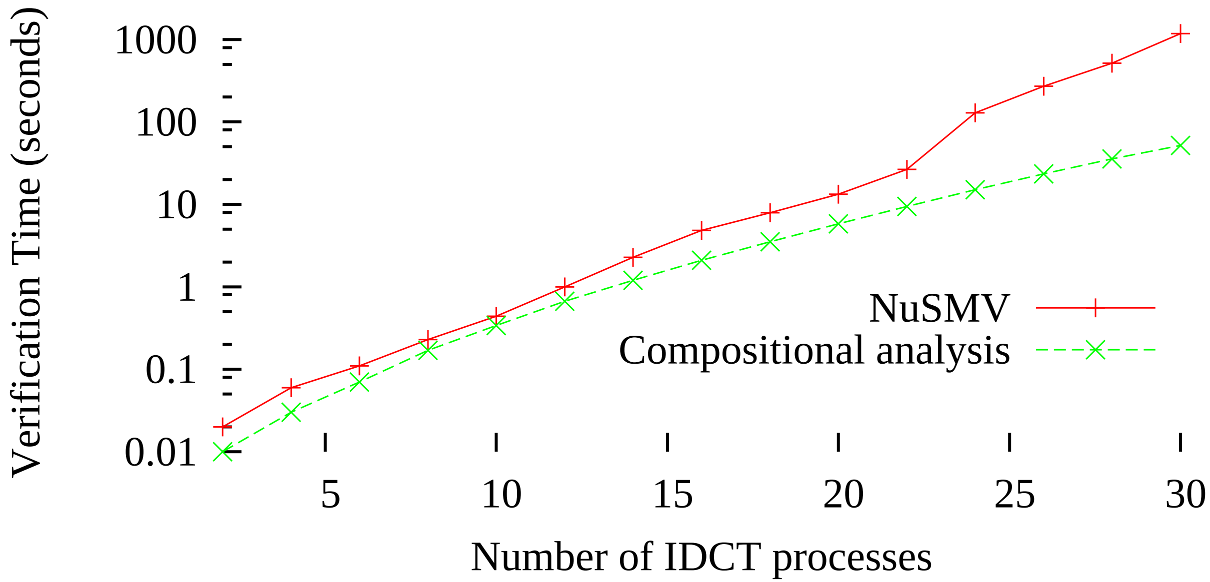
Example	Lines	Channels	Tasks	Result	Runtime	Memory
Source-Sink	35	2	11	No Deadlock	0.2 s	3.9 MB
Pipeline	30	7	13	No Deadlock	0.1	2.0
Prime Sieve	35	51	45	No Deadlock	1.7	25.4
Berkeley	40	3	11	No Deadlock	0.2	7.2
FIR Filter	100	28	28	No Deadlock	0.4	13.4
Bitonic Sort	130	65	167	No Deadlock	8.5	63.8
Framebuffer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

Deadlocks in SHIM

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

More Verification [EMSOFT 2009, MEMOCODE 2009, TCAD 2010]

- Compositional Deadlock Detection



- Buffer Sharing
  - Can two channels be active simultaneously?

More Verification [CC 2009]

- Analysis of clocks in X10
  - E.g.: A clock is used by just two tasks.
  - Specialization based on analysis

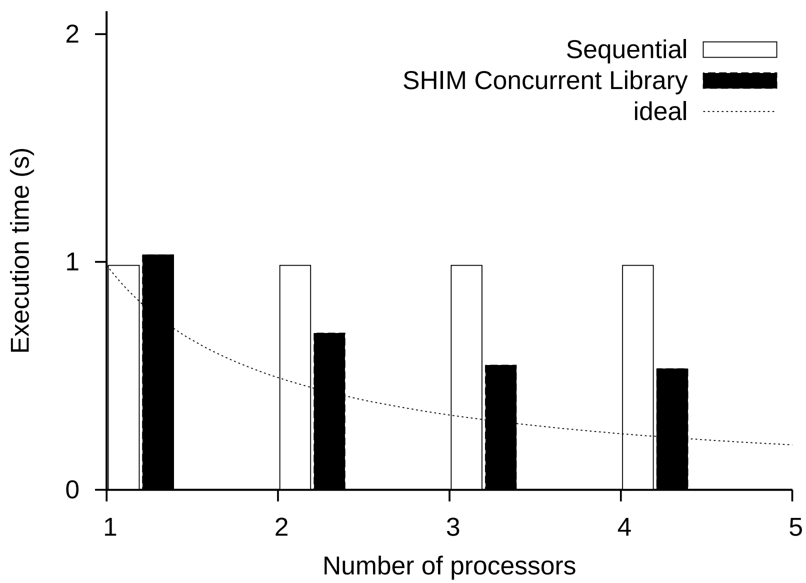
More Verification [CC 2009]

- Analysis of clocks in X10
  - E.g.: A clock is used by just two tasks.
  - Specialization based on analysis

Determinism simplifies verification

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



Deterministic, Deadlock-Free Model

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

```
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 */
}
```

Deterministic, Deadlock-free Model

- Histogram Example

```
void histogram(int a[], int n) {
  int b[10];
  for (int i = 0; i < n; i++) {
    spawn {
        int index = a[i];
        b[index]++;
    }
  }
  print (b);
}
```

Deterministic, Deadlock-free Model

- Histogram Example

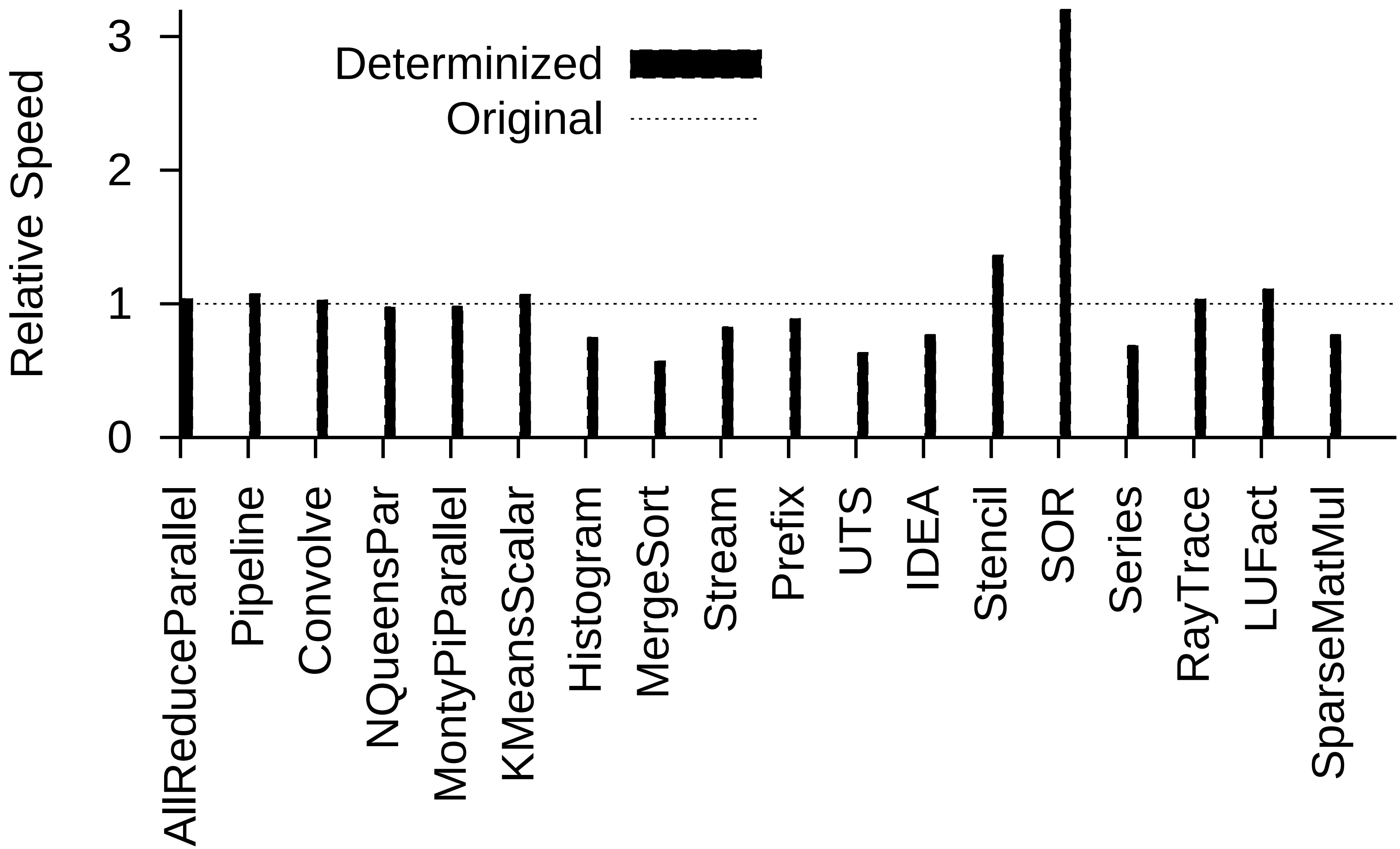
```
void histogram(int a[], int n) {
  shared int (+) b[10]
  for (int i = 0; i < n; i++) {
    spawn {
        int index = a[i];
        b[index] = 1;
        next;
    }
  }
  print (b);
}
```

Deterministic, Deadlock-free Model

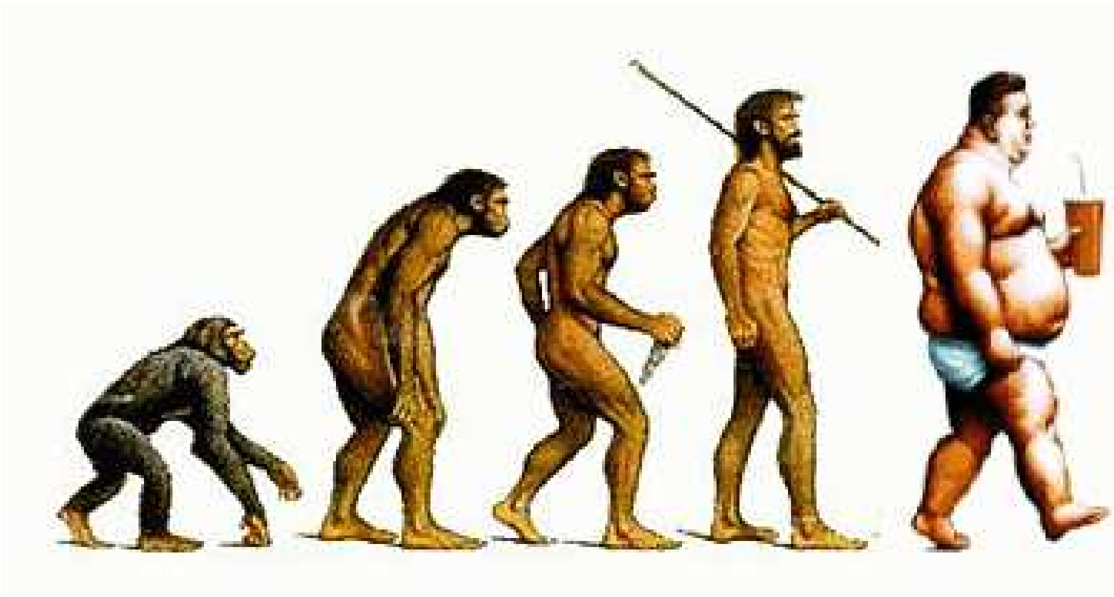
Determinism ✓

Deadlock Freedom ✓   Efficiency **?**

Deterministic, Deadlock-free Model



Future Work [PLDI'09 Fun Ideas and Thoughts]



Library Support

Performance

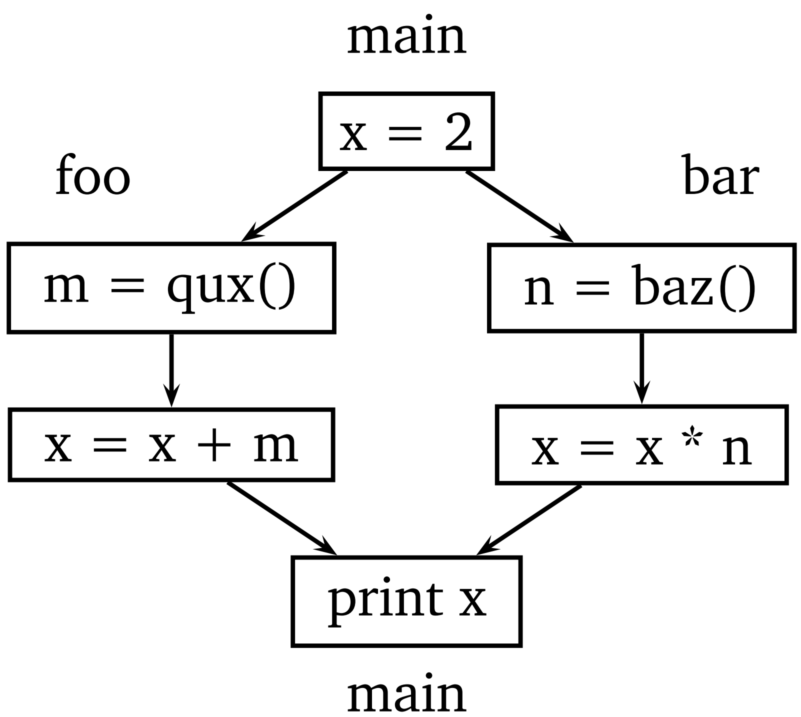
Parallel Computers

Parallel Languages

A Determinizing Compiler!

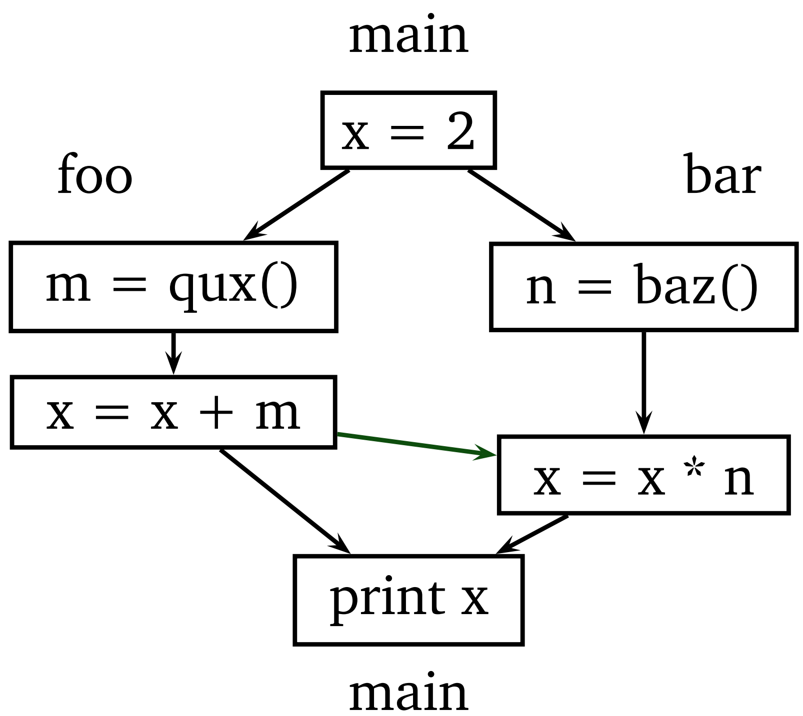
The Example

```
int x;
foo(){
    int m;
    m = qux();
    x = x + m;
 }
bar(){
    int n;
    n = baz();
    x = x * n;
 }
 main() {
     x = 2;
     spawn foo();
     spawn bar();
     sync;
     print(x);
}
```



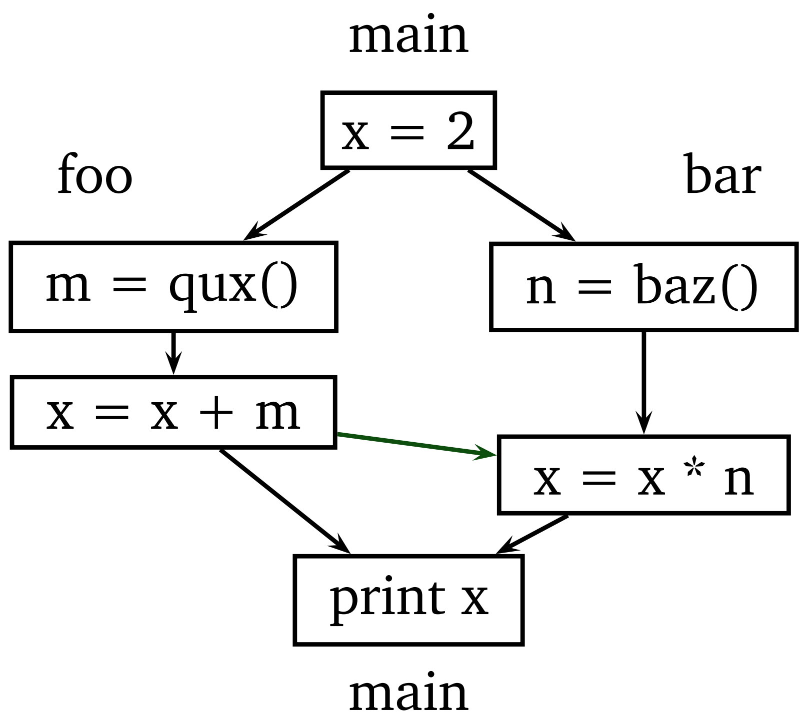
The Determinizing Compiler's Role

```
int x;
foo(){
    int m;
    m = qux();
    x = x + m;
    sync(x);
}
bar(){
    int n;
    n = baz();
    sync(x);
    x = x * n;
}
main() {
    x = 2;
    spawn foo();
    spawn bar();
    sync;
    print(x);
}
```



The Determinizing Compiler’s Role

```
int x;
foo(){
    int m;
    m = qux();
    x = x + m;
    sync(x);
  }
bar(){
    int n;
    n = baz();
    sync(x);
    x = x * n;
  }
  main() {
     x = 2;
     spawn foo();
     spawn bar();
     sync;
     print(x);
}
```



if m = n = 2

x = (2 + 2) * 2 = 8

Always!

The Ultimate Goal