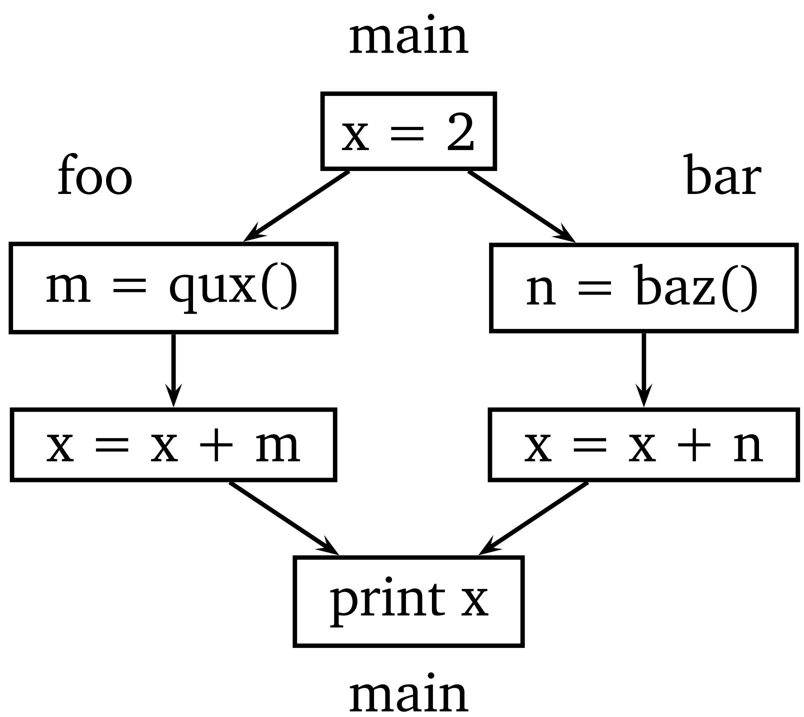
Dealing with Concurrency Problems

Nalini Vasudevan

Columbia University

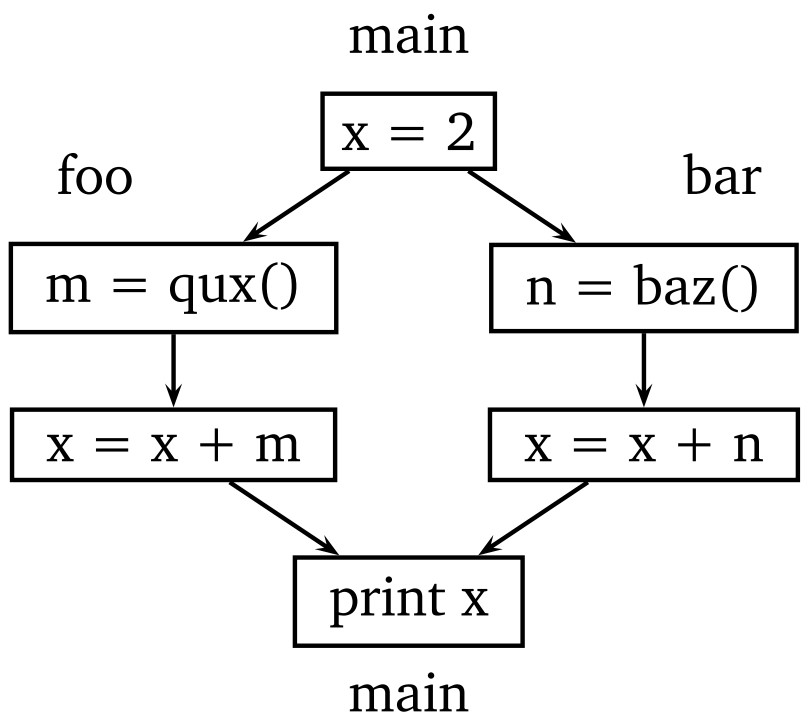
What is the output?

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



What is the output?

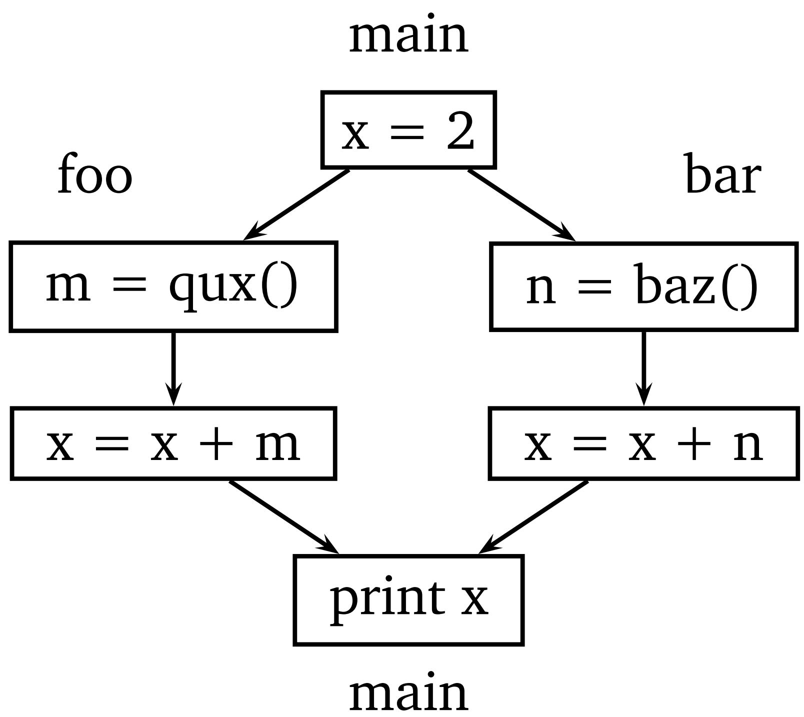
```
int x;
foo(){
    int m;
    m = qux();
    x = x + m;
}
bar(){
    int n;
    n = baz();
    x = x + n;
}
main() {
    x = 2;
    foo() par bar();
    print(x);
}
```



Data Race

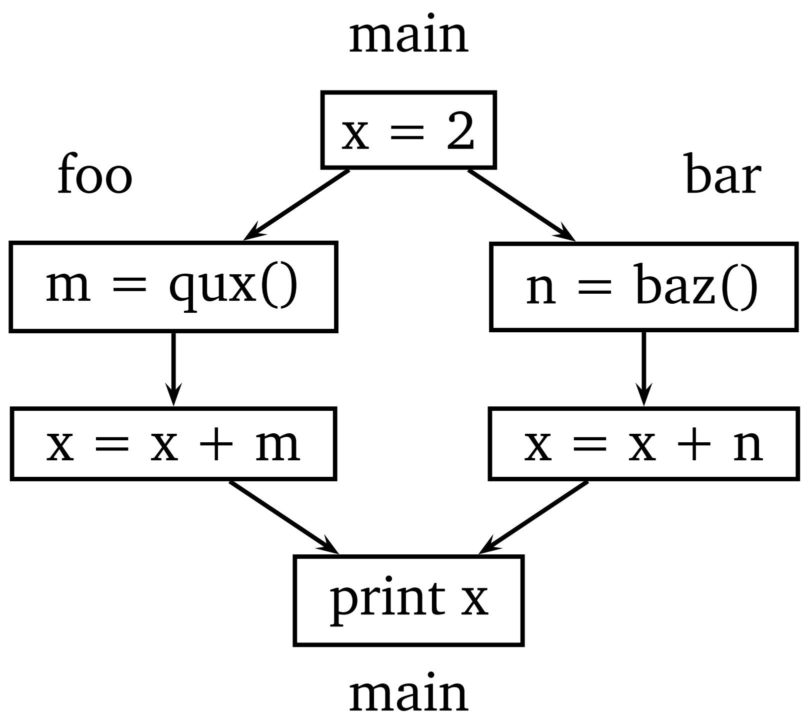
Eliminating Data Races

```
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;
    foo() par bar();
    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;
     foo() par bar();
     print(x);
}
```

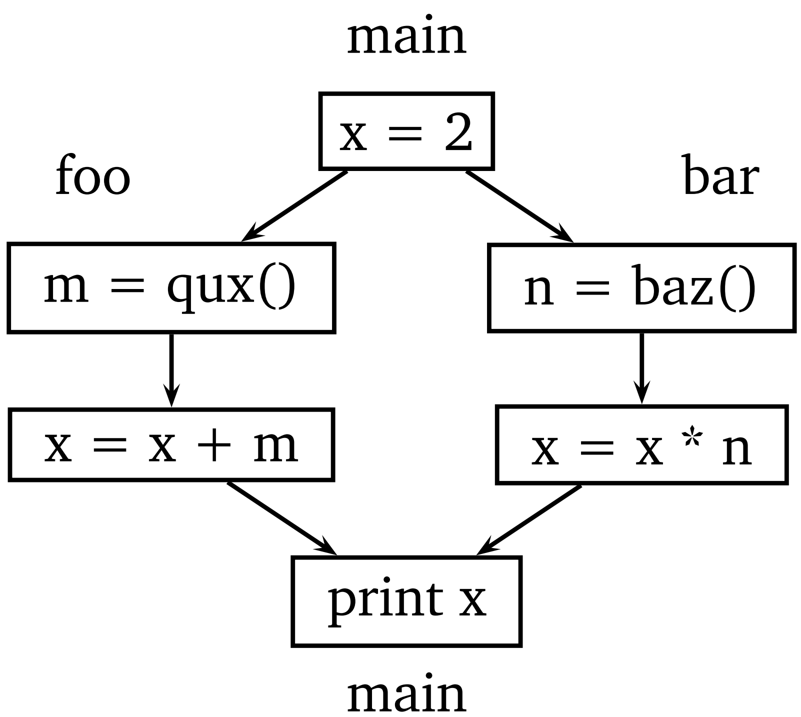


if $m = n = 2$

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

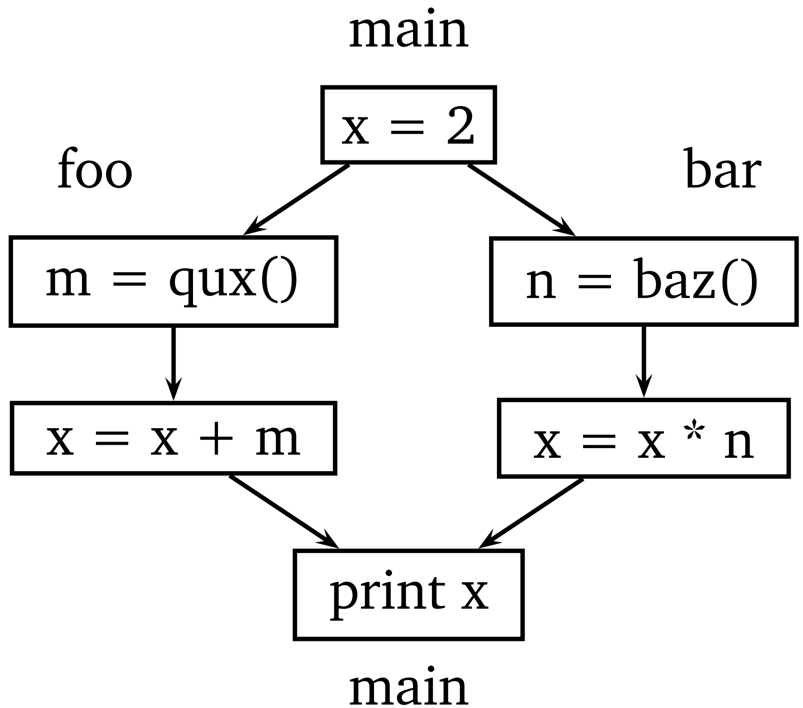
Another Example

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



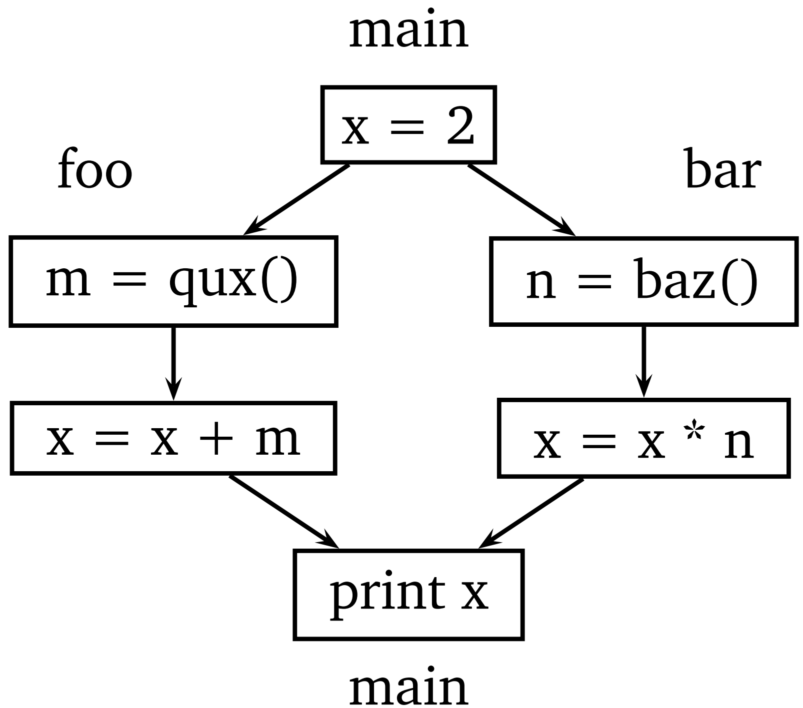
Eliminating Data Races

```
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;
     foo() par bar();
     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;
     foo() par bar();
     print(x);
}
```



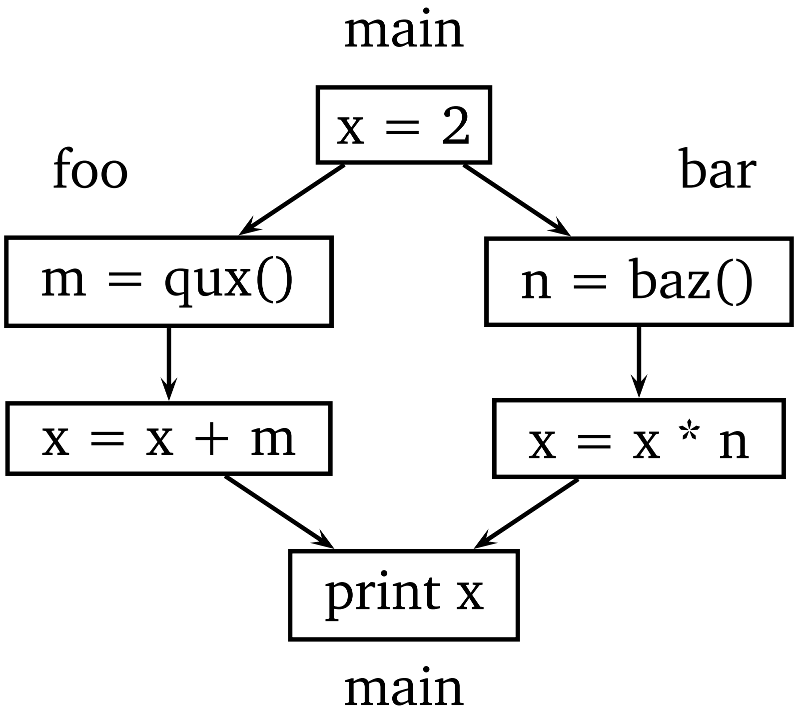
if m = n = 2

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;
     foo() par bar();
     print(x);
}
```



if m = n = 2

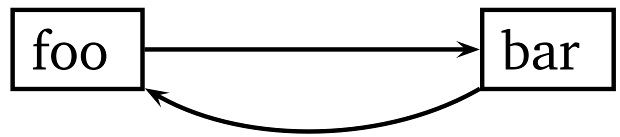
x = (2 + 2) * 2 = 8

x = (2 * 2) + 2 = 6

Non-determinism

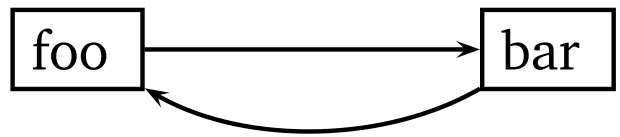
Problem with Locks

```
int x = 0;
int y = 0;

foo() {
    lockx();
    locky();
    x++;
    y++;
    unlocky();
    unlockx();
}

bar(){
    locky();
    lockx();
    y++;
    x++;
    unlockx();
    unlocky();
 }

main() {
     foo(x) par bar(x);
     print(x);
}
```



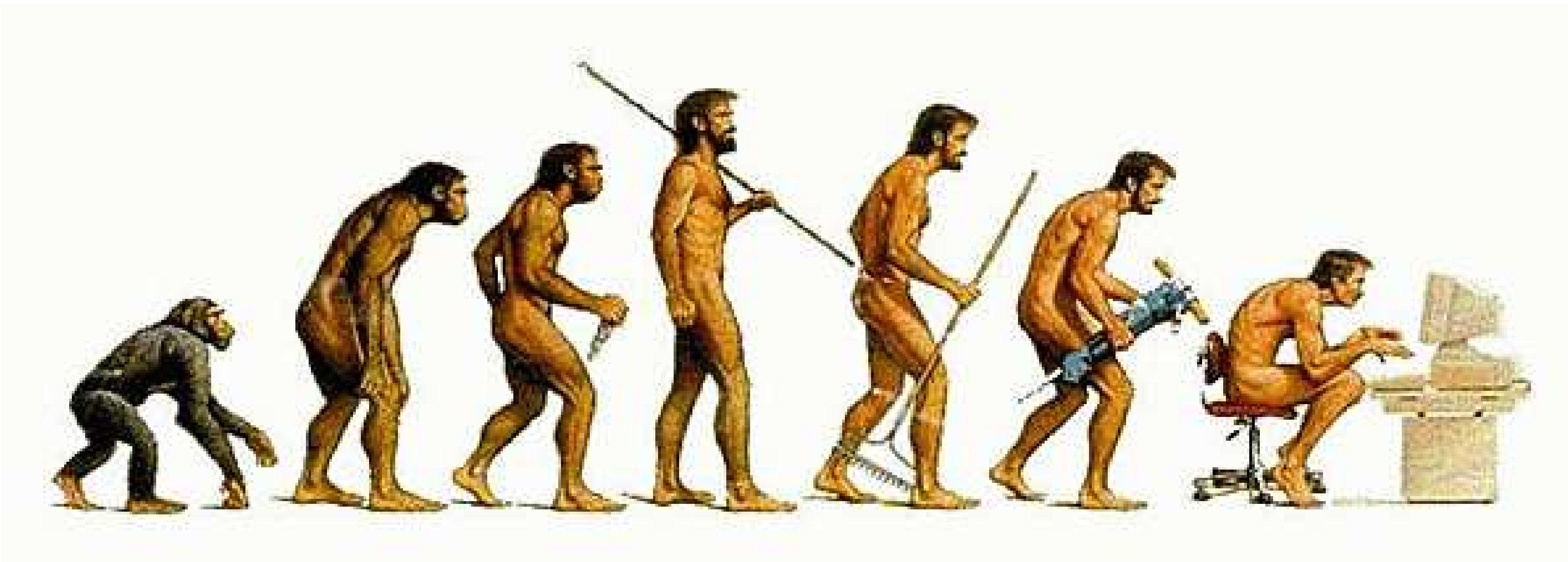
Problem with Locks

```
int x = 0;
int y = 0;

foo() {
    lockx();
    locky();
    x++;
    y++;
    unlocky();
    unlockx();
}

bar(){
    locky();
    lockx();
    y++;
    x++;
    unlockx();
    unlocky();
 }

main() {
     foo(x) par bar(x);
     print(x);
}
```



Deadlock

Motivation



Library Support

Performance

Deadlocks

Parallel Computers

Parallel Languages

Non-Determinism

Hard-to-Debug

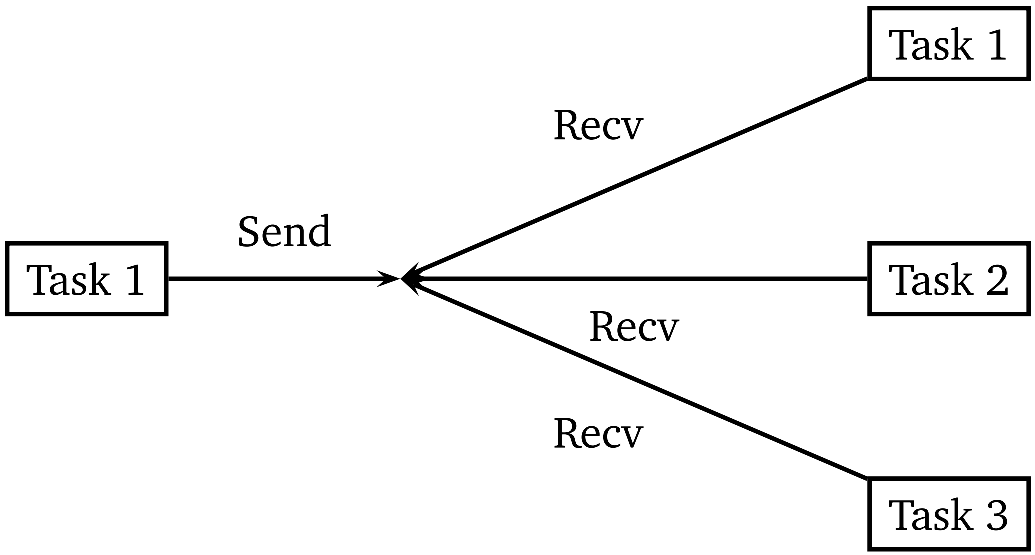
Motivation

Determinism ?

Deadlock Freedom ? Efficiency ?

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;
}
```

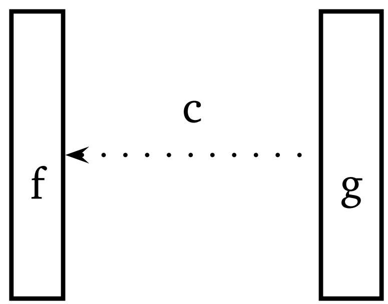
Additional Constructs

$stmt_1$ *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);
}
```



Compiling to Quad-Core [DATE 2008]

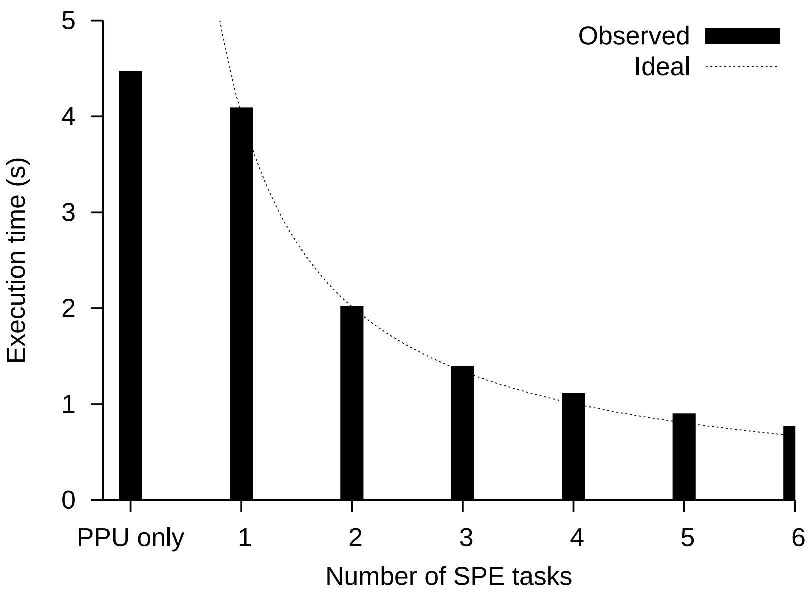
- Intel Quad Core Machine
- Each task mapped to a pthread
- Example: JPEG decoder

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.

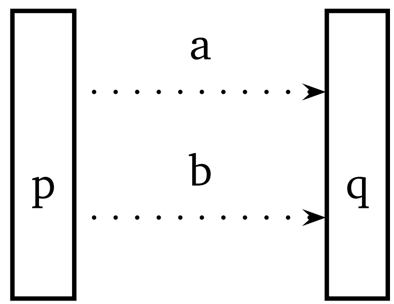
Compiling to Cell [SAC 2009]

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



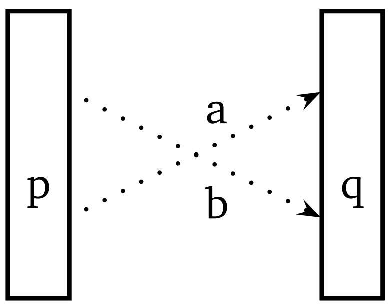
More Examples in SHIM

```
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;
  }
}
```

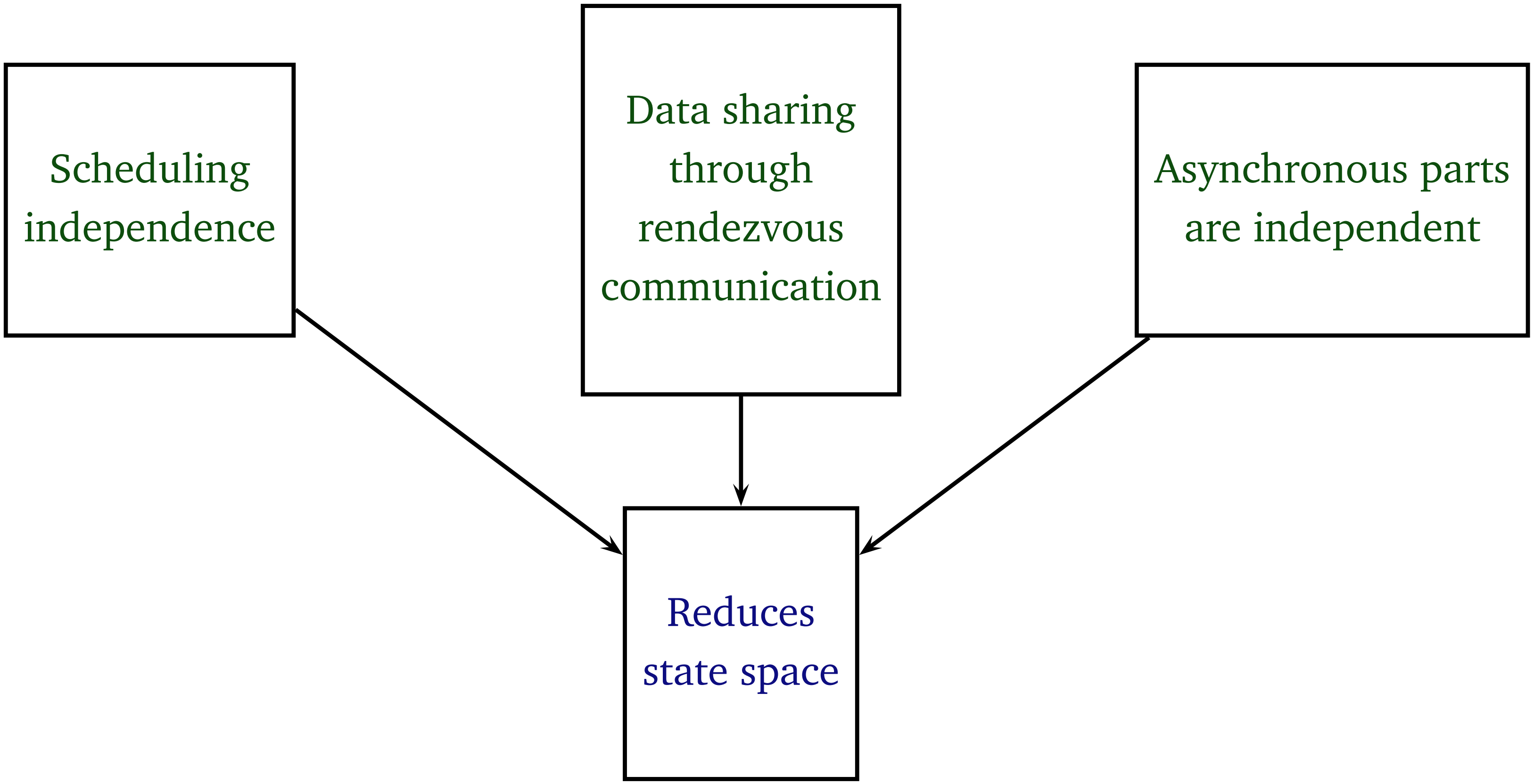


The Problem

```
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;
  }
}
```



Static Deadlock Detection



Just pick one schedule

Deadlocks in SHIM

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

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 */
  f(x) par g(x) par h(x);
  /* 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++) par {
        int index = a[i];
        b[index]++;
  }
  print (b);
}
```

Deterministic, Deadlock-free Model

- Histogram Example

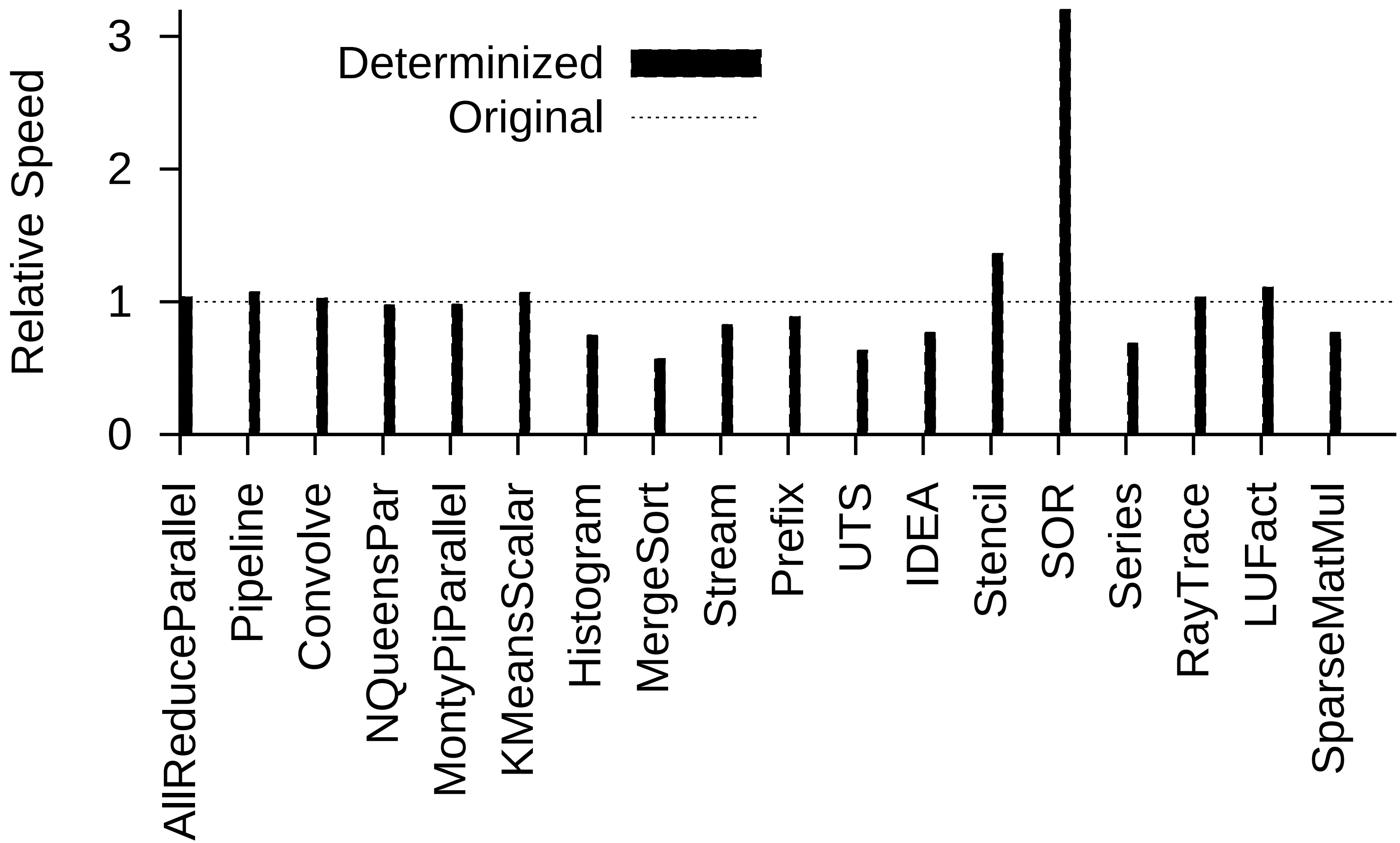
```
void histogram(int a[], int n) {
  shared int (+) b[10]
  for (int i = 0; i < n; i++) par {
        int index = a[i];
        b[index] = 1;
        next;
  }
  print (b);
}
```

Deterministic, Deadlock-free Model

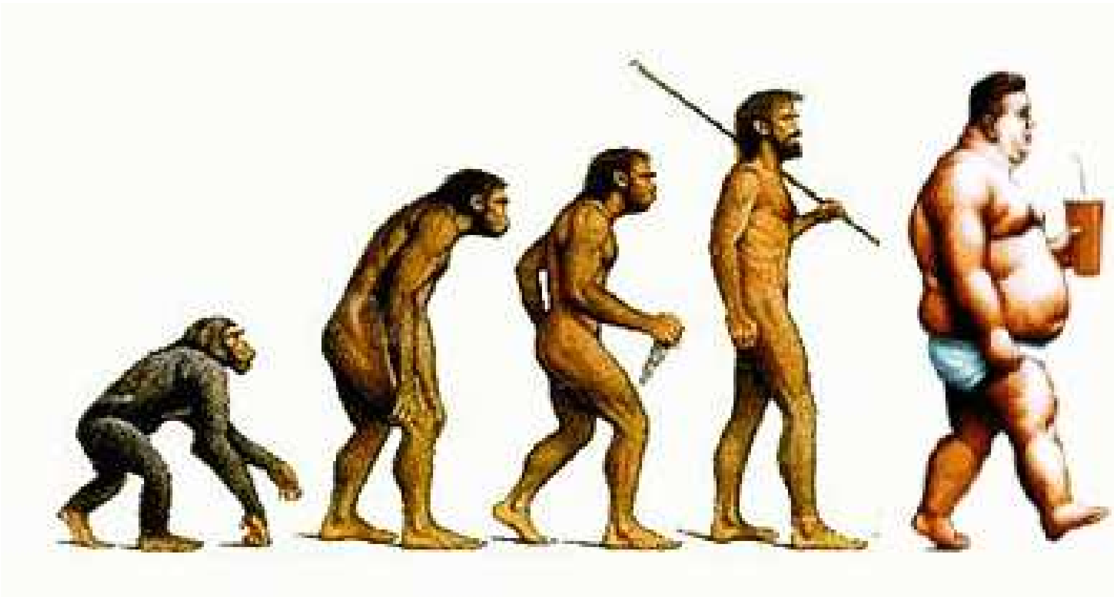
Determinism ✓

Deadlock Freedom ✓ Efficiency **?**

Deterministic, Deadlock-free Model



Future Work [PLDI'09 Fun Ideas and Thoughts]



Parallel Computers

Library Support

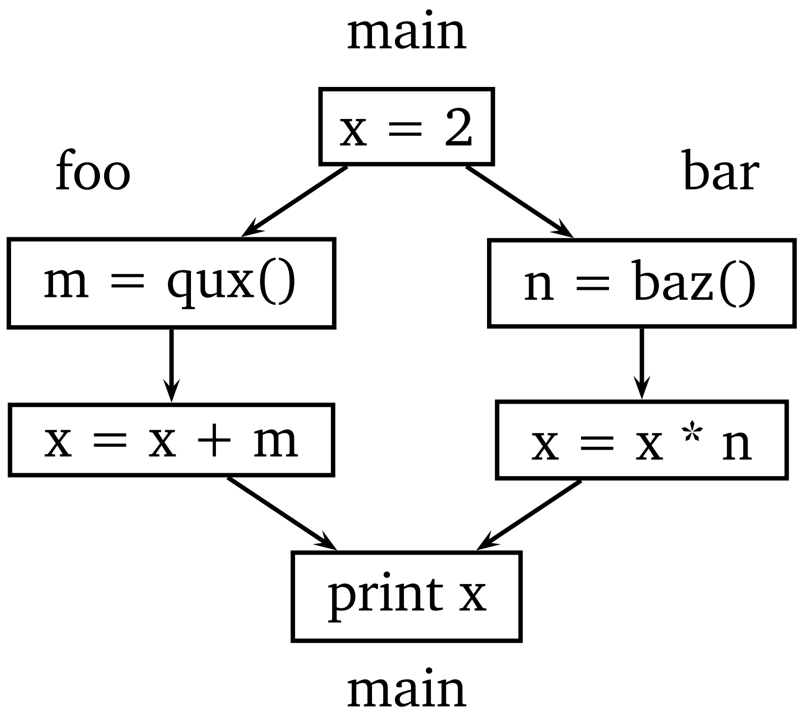
Parallel Languages

Performance

A Determinizing Compiler!

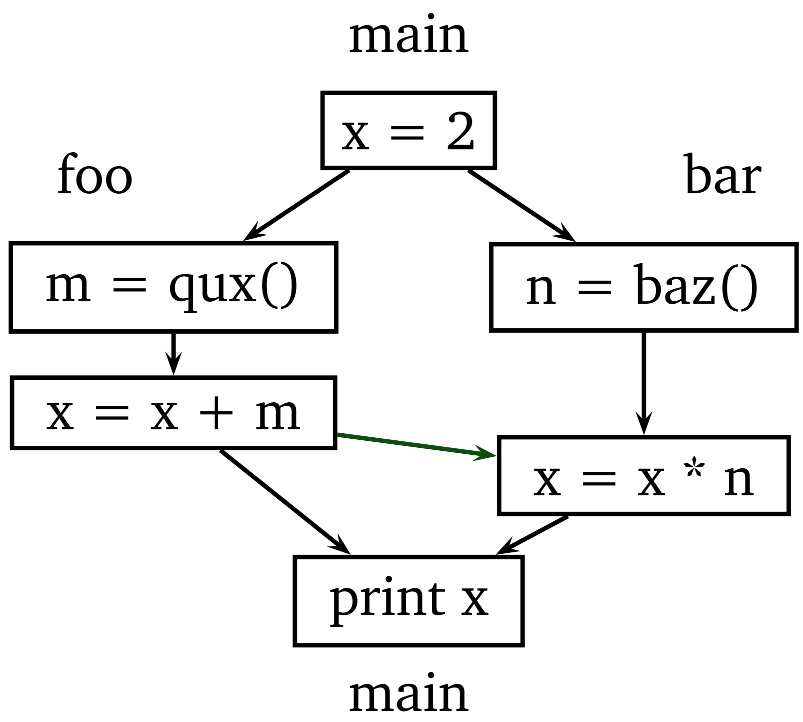
The Example

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



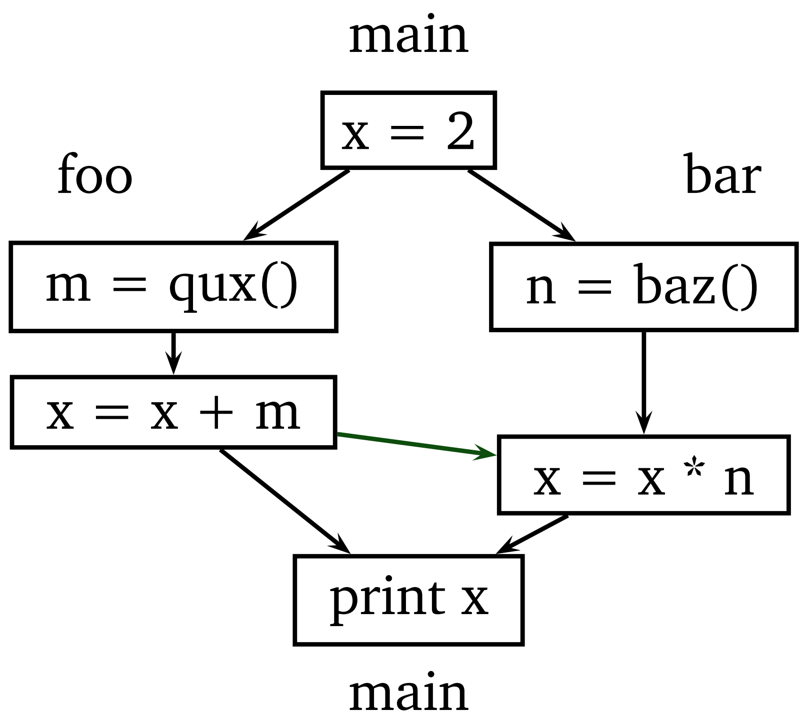
The Determinizing Compiler's Role

```
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;
    foo() par bar();
    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;
    foo() par bar();
    print(x);
}
```



if m = n = 2

x = (2 + 2) * 2 = 8

Always!

Scalability

- Synchronization is expensive
- Synchronization = Sequentializing

Scalability

- Synchronization is expensive
- Synchronization = Sequentializing

Suppose a program runs for 100s on a single processor. 80% of the program can be parallelized. What is the speed up on running the program with

1. 2 processors
2. 4 processors
3. 8 processors

Scalability

- Synchronization is expensive
- Synchronization = Sequentializing

Suppose a program runs for 100s on a single processor. 80% of the program can be parallelized. What is the speed up on running the program with

1. 2 processors
2. 4 processors
3. 8 processors

Ans: 1.66, 2.5, 3.33 [Using Amdahl's law]

The Ultimate Goal

Determinism ✓

Deadlock Freedom ✓ Efficiency ✓