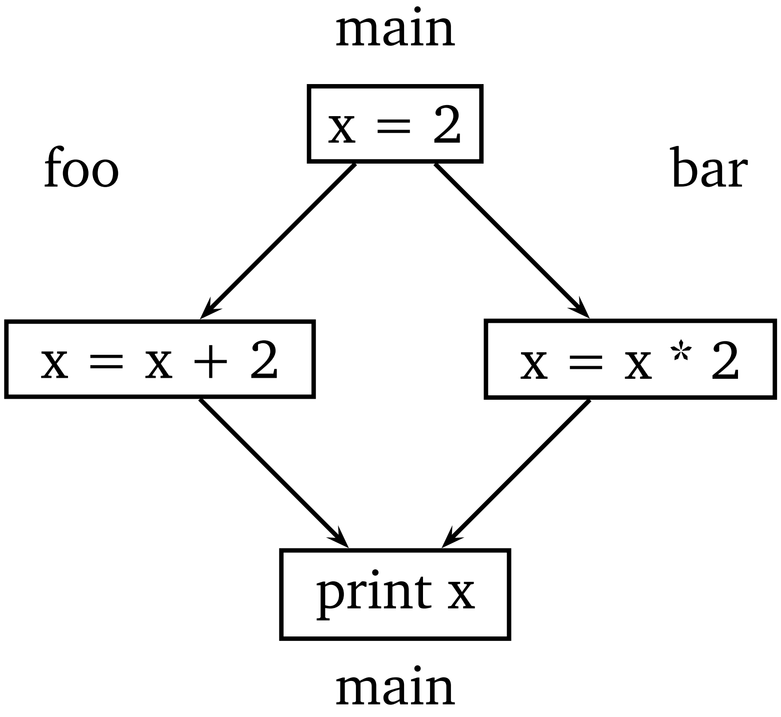
Deterministic, Deadlock-free Concurrency

Nalini Vasudevan

Columbia University

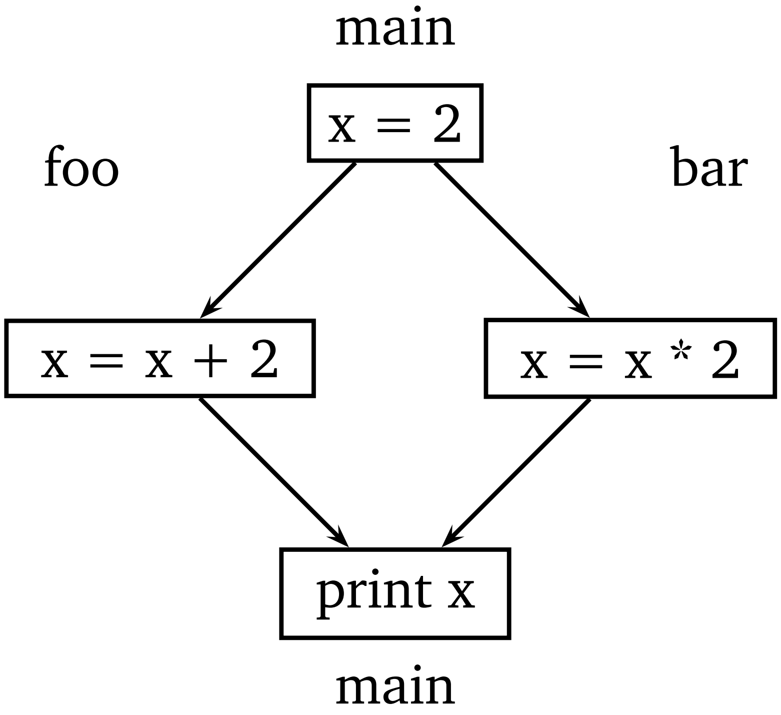
Data Races

```
 int x;
 foo(){
     int m;
     x = x + 2;
   }
 bar(){
     int n;
     x = x * 2;
   }
   main() {
       x = 2;
       finish {
           async foo();
           async bar();
       }
       print(x);
}
```



Eliminating Data Races

```
int x;
foo(){
    int m;
    lock(x);
      x = x + 2;
    unlock(x);
  }
bar(){
    int n;
    lock(x);
      x = x * 2;
    unlock(x);
  }
  main() {
     x = 2;
     finish {
        async foo();
        async bar();
     }
     print(x);
}
```



main

x = 2

foo

bar

x = x + 2

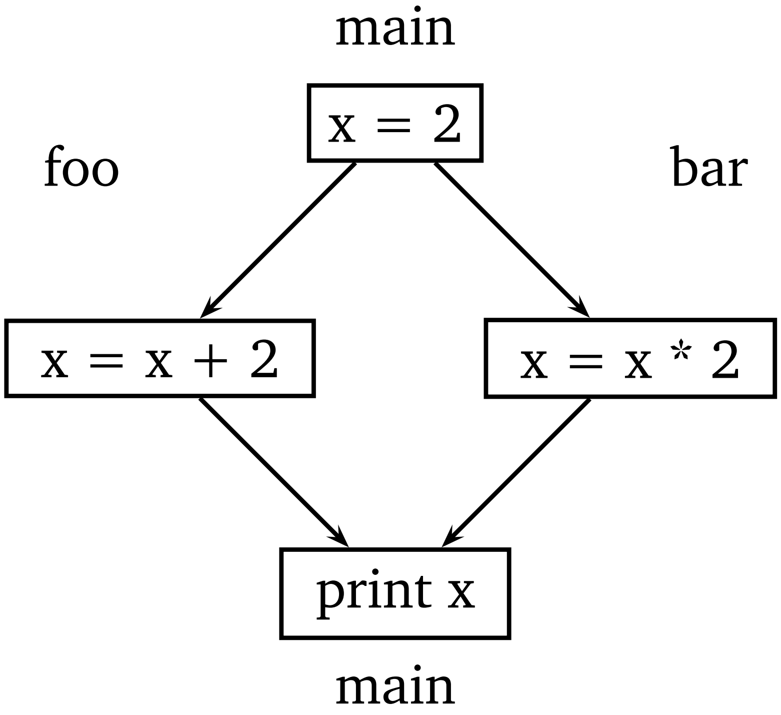
x = x * 2

print x

main

Eliminating Data Races

```
int x;
foo(){
    int m;
    lock(x);
      x = x + 2;
    unlock(x);
  }
bar(){
    int n;
    lock(x);
      x = x * 2;
    unlock(x);
  }
  main() {
     x = 2;
     finish {
        async foo();
        async bar();
     }
     print(x);
}
```

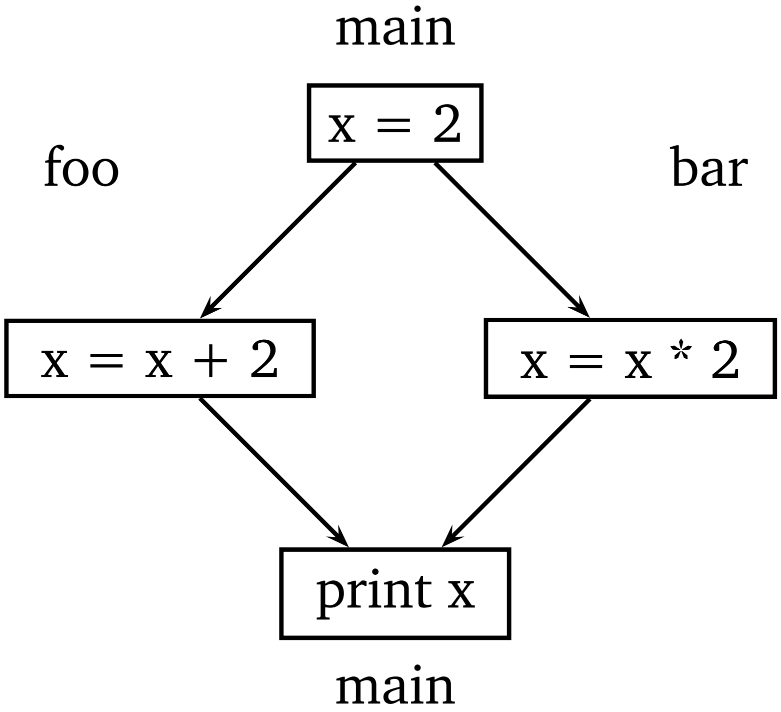


$$x = (2 + 2) * 2 = 8$$

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

Eliminating Data Races

```
int x;
foo(){
    int m;
    lock(x);
      x = x + 2;
    unlock(x);
 }
bar(){
    int n;
    lock(x);
      x = x ∗ 2;
    unlock(x);
 }
 main() {
     x = 2;
     finish {
        async foo();
        async bar();
     }
     print(x);
}
```



main

x = 2

foo

bar

x = x + 2

x = x * 2

print x

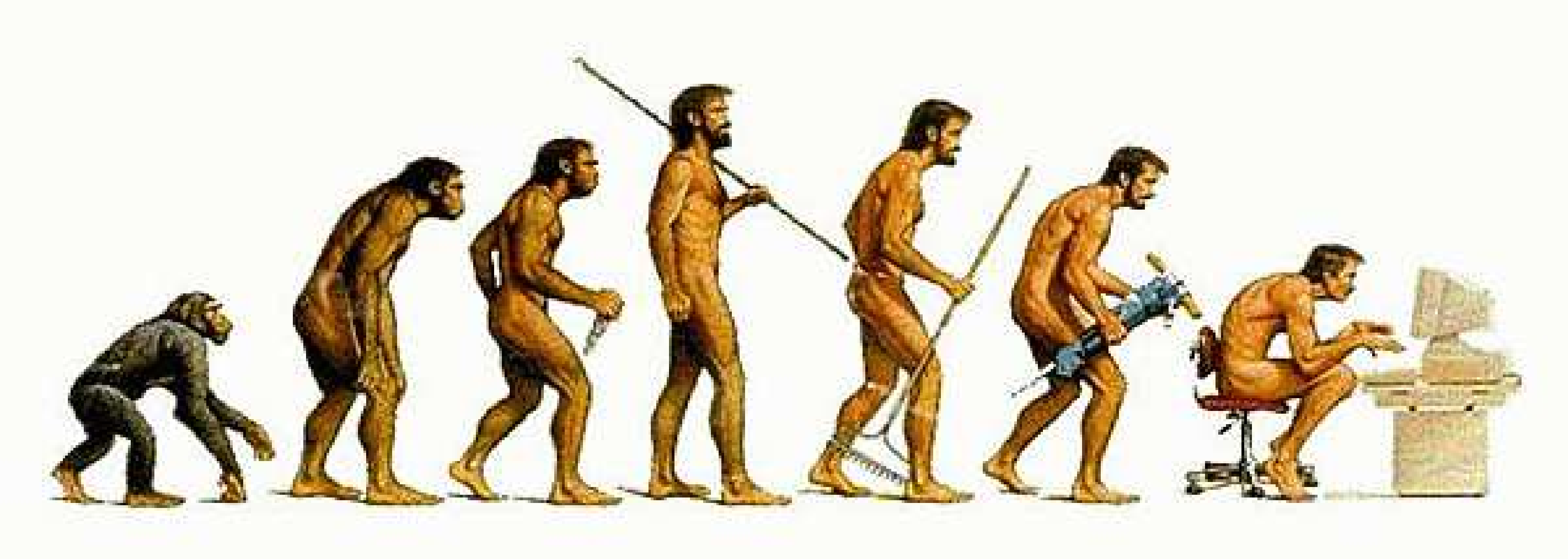
main

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

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

Non-determinism

Motivation



Parallel Computers

Library Support

Parallel Languages

Performance

Data Races

Non-Determinism

Hard-to-Debug

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 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 */
  async f(x);
  async g(x);
  h(x);
  /* x is 8 */
}
```

A Concrete Example

- Histogram Example

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

A Concrete Example

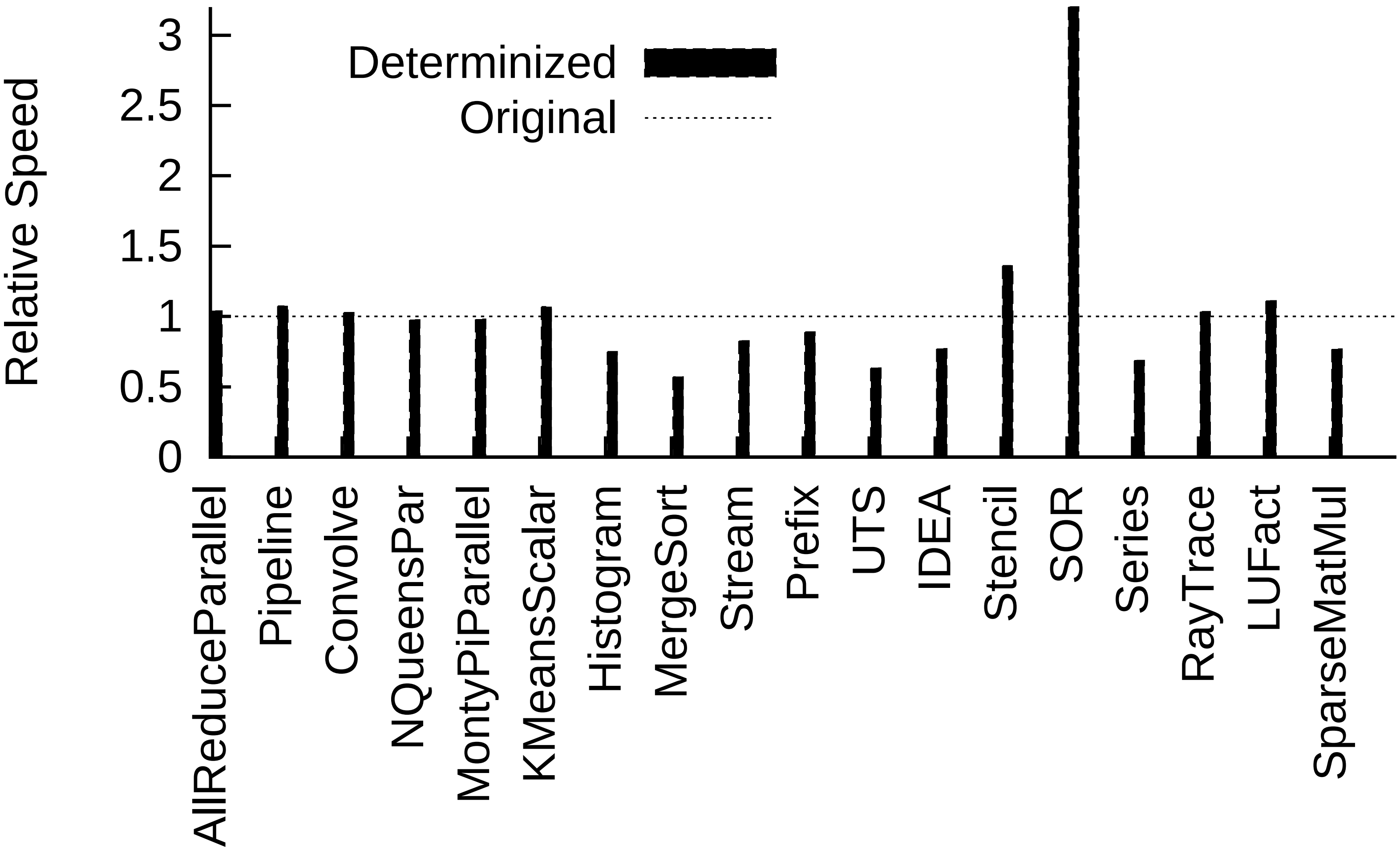
- Histogram Example

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

Implementation in X10

```
val a = Rail.make[Int](N);
/* Initialize a */
..
val c = new Clock();
val b = Rail.make[Int @ Clocked[Int] (c, Int.+, 0)](10);
for((i:Int) in 0..N-1) {
   async clocked(c) {
      val bin = a(i);
      b(bin) = 1;
   }
}
next;
```

Results


Relative Speed
3
2.5
2
1.5
1
0.5
0
Determinized
Original
AllReduceParallel
Pipeline
Convolve
NQueensPar
MontyPiParallel
KMeansScalar
Histogram
MergeSort
Stream
Prefix
UTS
IDEA
Stencil
SOR
Series
RayTrace
LUFact
SparseMatMul

The Ultimate Goal

Determinism ✓

Deadlock Freedom ✓

Efficiency ✓

Acknowledgment

- Julian Dolby
- Vijay Saraswat
- Olivier Tardieu