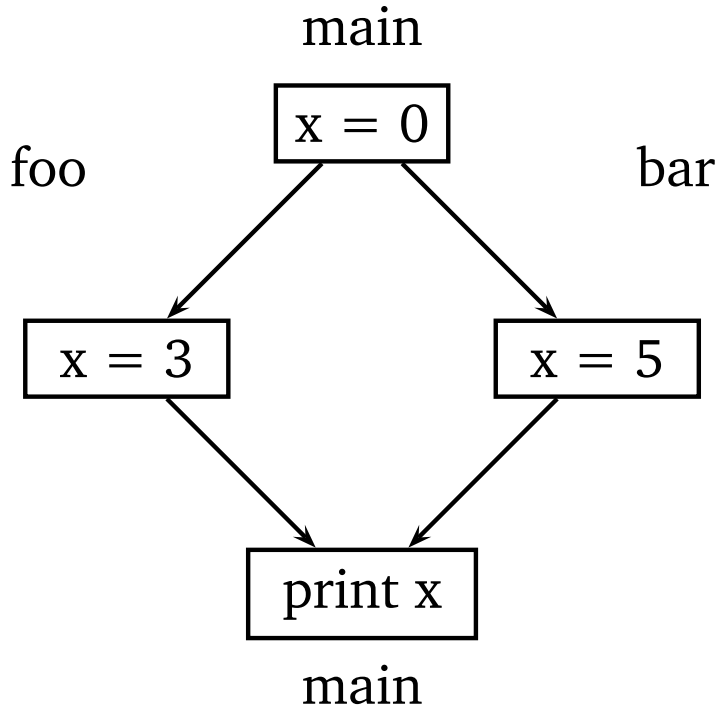
D^2C: Deterministic, Deadlock-free Concurrency

Nalini Vasudevan

Columbia University

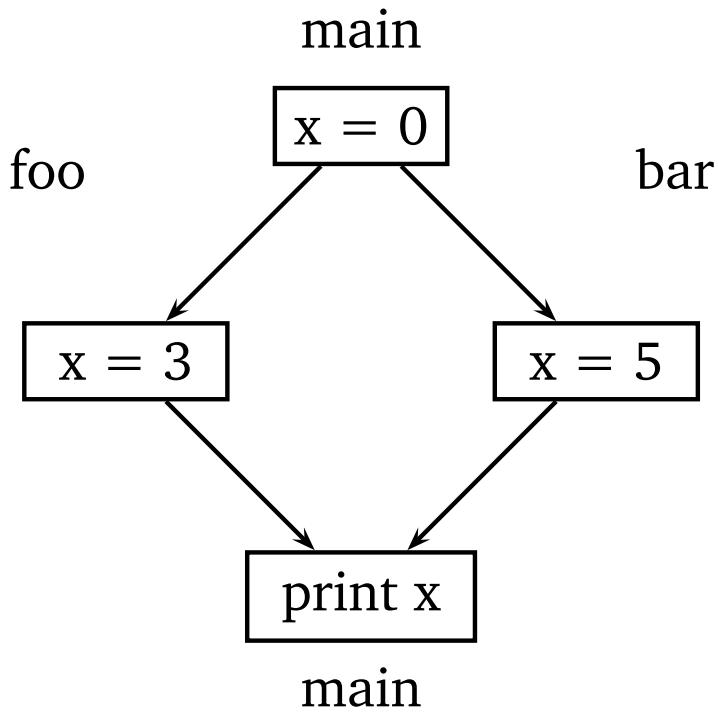
Data Races

```
foo(int &a){
    a = 3;
 }
bar(int &b){
    b = 5;
 }
 main() {
     int x = 0;
     spawn foo(x);
     bar(x);
     sync;
     print(x);
}
```



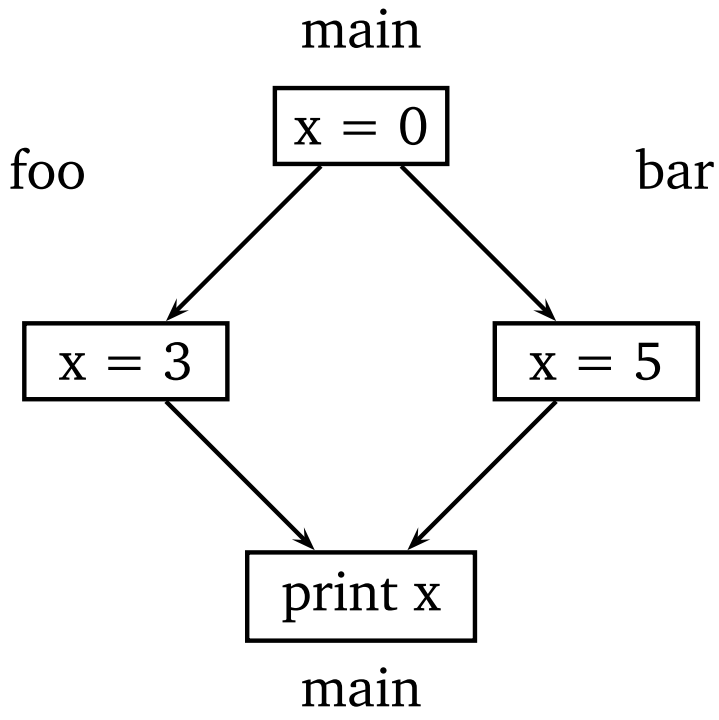
Eliminating Data Races

```
foo(int &a){
    lockx();
    a = 3;
    unlockx();
 }
bar(int &b){
    lockx();
    b = 5;
    unlockx();
 }

main() {
     int x = 0;
     spawn foo(x);
     bar(x);
     sync;
     print(x);
}
```



Eliminating Data Races

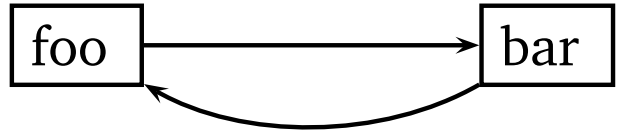
```
foo(int &a){
    lockx();
    a = 3;
    unlockx();
 }
bar(int &b){
    lockx();
    b = 5;
    unlockx();
 }

main() {
     int x = 0;
     spawn foo(x);
     bar(x);
     sync;
     print(x);
}
```



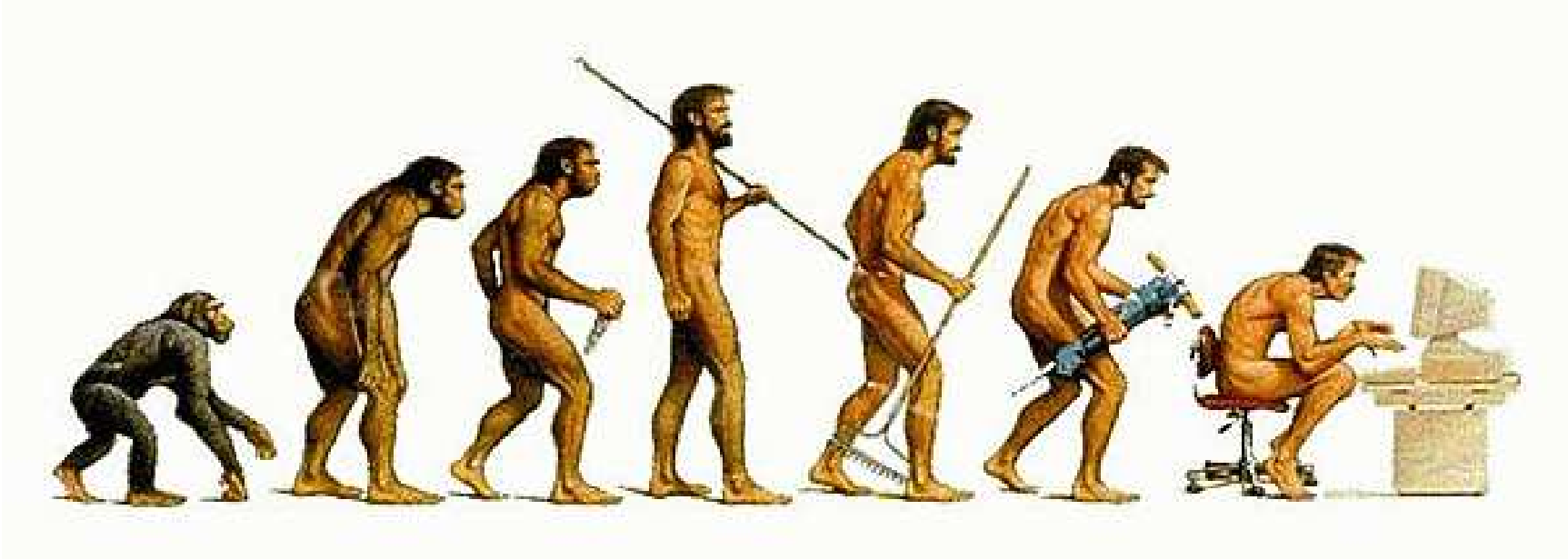
Non-determinism

Deadlocks

```
foo(int &a){
    lockx();
    locky();
    a = 3;
    unlocky();
    unlockx();
 }
bar(int &b){
    locky();
    lockx();
    b = 5;
    unlockx();
    unlocky();
 }

main() {
     int x = 0;
     spawn foo(x);
     bar(x);
     sync;
     print(x);
}
```



Motivation



Parallel Computers

Library Support

Parallel Languages

Performance

Non-Determinism

Deadlocks

Hard-to-Debug

Deterministic, Deadlock-Free Model

```
void f(shared int &a) {
 /* a is 0 */
  a = 3;
 /* a is 3 , x is still 0 */
  next; /* Apply reduction operator */
 /* a is now 8, x is 8 */
}
```

```
void g(shared int &b) {
 /* b is 0 */
  b = 5;
 /* b is 5, x is still 0 */
  next; /* Apply reduction operator */
 /* b is now 8, x is 8 */
}
```

```
main() {
  shared int (+) x = 0;
  /* If there are multiple writers, reduce
     using the + reduction operator */
  spawn f(x);
  g(x);
  /* x is 8 */
}
```

Without a Reducing Operator

```
void f(shared int &a) {
 /* a is 0 */
  a = 3;
 /* a is 3 , x is still 0 */
  next; /* Commit in spawn order */
 /* a is now 5, x is 5 */
}
```

```
void g(shared int &b) {
 /* b is 0 */
  b = 5;
 /* b is 5, x is still 0 */
  next; /* Commit in spawn order */
 /* b is now 5, x is 5 */
}
```

```
main() {
  shared int x = 0;
  /* If there are multiple writers,
     commit in the order of spawn */
  spawn f(x);
  g(x);
  /* x is 5 */
}
```

A Concrete Example

- Histogram Example

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

A Concrete Example

- Histogram Example

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

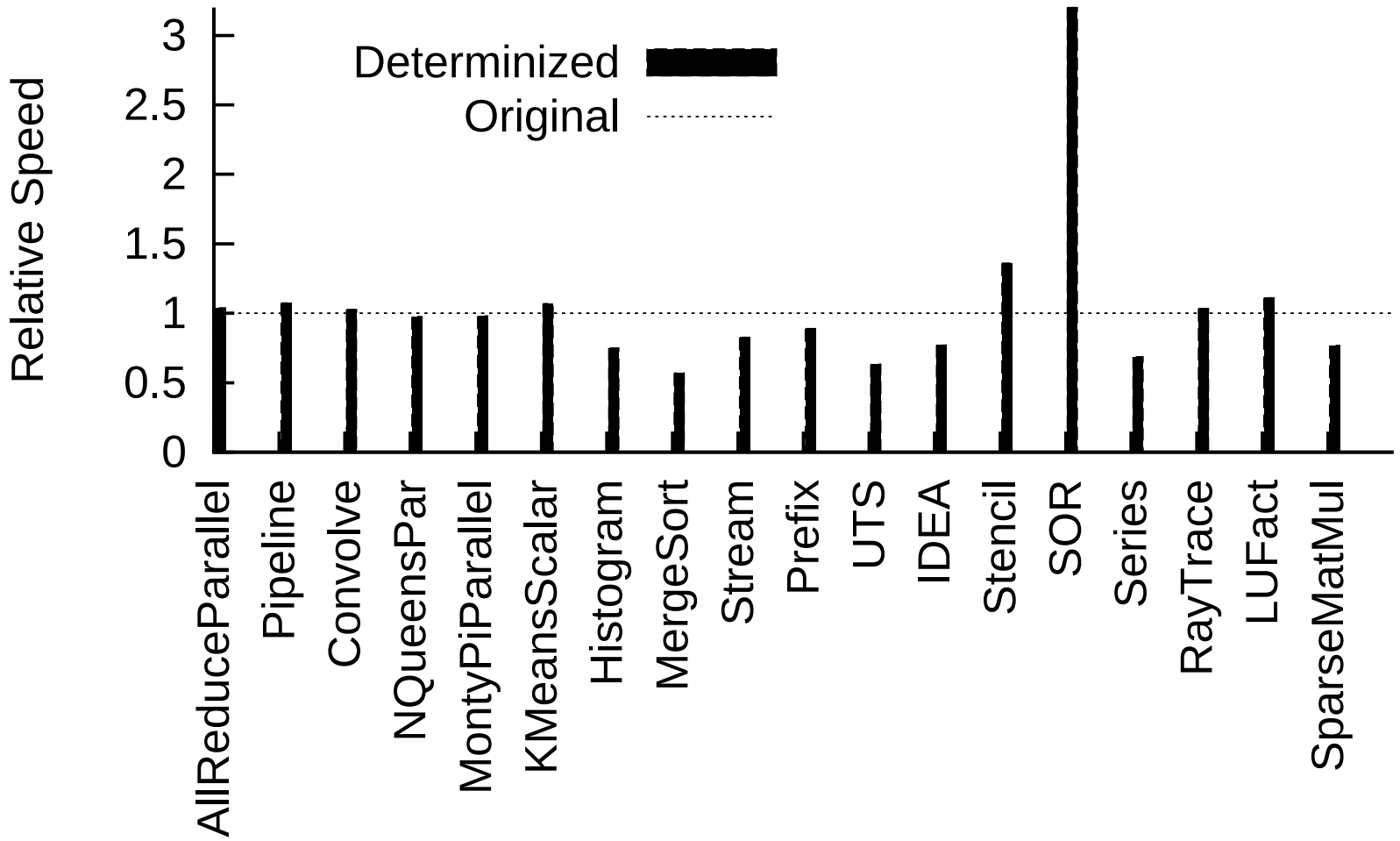
Problematic

A Concrete Example

- Histogram Example

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

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 ✓

Questions Guaranteed

Answers are not

Pipeline Example

```
main() {
  shared int (+) a = 0, b = 0;
  spawn (a) {
    int i = 0;
    for (;;) {
      a = ++i;
      next;
    }
  }
  spawn (a, b) {
    for (;;) {
      b = a + 1;
      next;
    }
  }
    for (;;) {
      int c = b + 1;
      next;
    }
}
```