DOMP: Deterministic OpenMP

Amittai Aviram
Bryan Ford
Department of Computer Science
Yale University
{amittai.aviram, bryan.ford}@yale.edu

SUMMARY

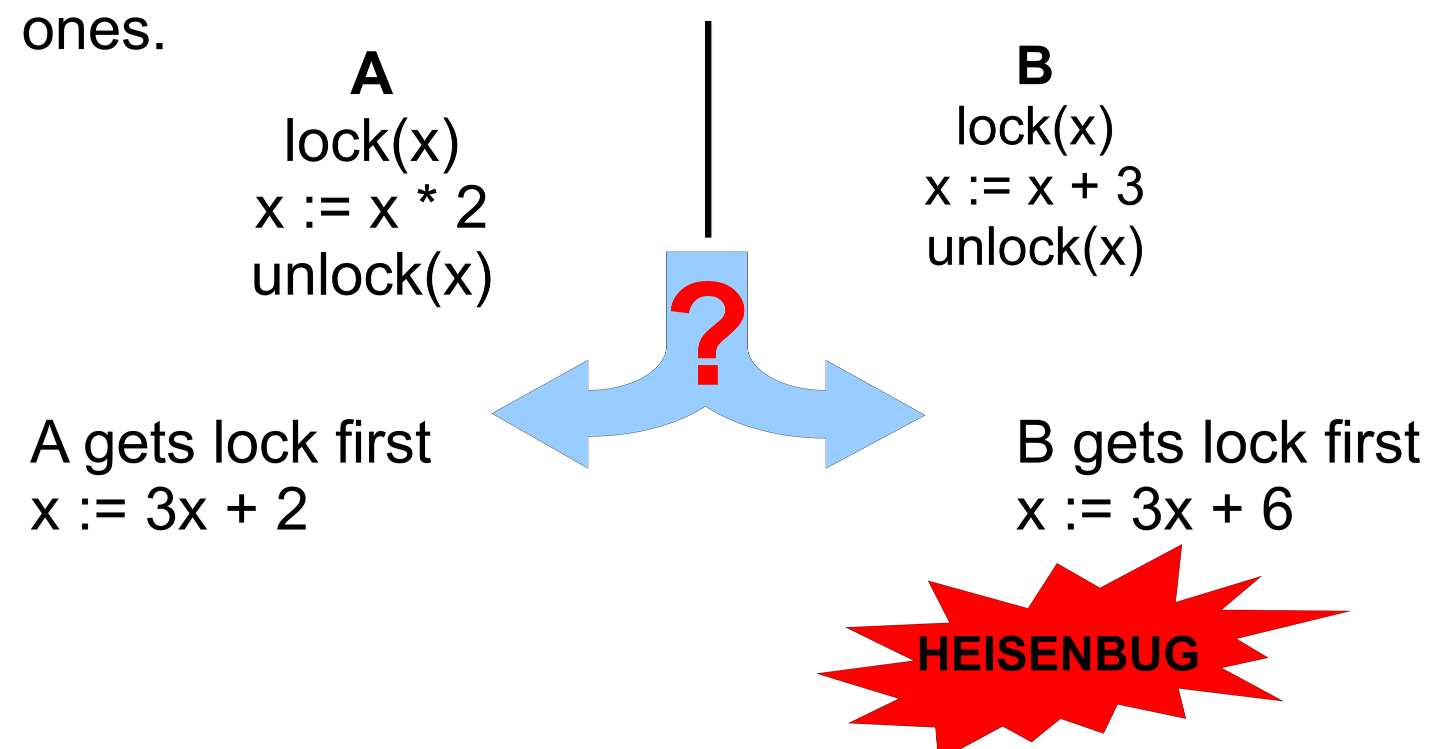
How to guarantee the deterministic execution of parallel programs and prevent race conditions and heisenbugs is now an important research topic. One obstacle is that conventional programming models rely on *naturally nondeterministic* synchronization abstractions. With *naturally deterministic* abstractions, programming logic alone, immune to timing, determines which threads synchronize, how, and at what point, making data races impossible. By contrast, nondeterministic primitives, such as mutex locks and condition variables, place the burden on the programmer to head off data races. New programming languages may address this problem, but legacy code must be rewritten. Alternatively, deterministic thread scheduling [1] handles conventional languages—making race conditions repeatable without eliminating them. A novel OS, Determinator, handles conventional languages and eliminates data races entirely, but only supports code using a handful of naturally deterministic synchronization abstractions [2]. To address this limitation and provide both race-free determinism and programming expressiveness, we propose a deterministic version of the well-established OpenMP parallel programming API: DOMP (*D*eterministic *OpenMP*).

References

1. Joseph Devietti, Brandon Lucia, Luis Ceze, and Mark Oskin. DMP: Deterministic shared memory multiprocessing. In *14th ASPLOS*, March, 2009.
2. Amittai Aviram, Shu-Chun Weng, Sen Hu, and Bryan Ford. Efficient system-enforced deterministic parallelism. In *9th OSDI*, October, 2010.

Mutexes Do Not Eliminate All Data Races

Atomicity prevents low-level data races but not high-level ones.



Deterministic Scheduling Does Not Eliminate Data Races

Data races can be reproduced, but their persistence can still cause problems.

```
int x = 5;

// Start parallel execution here.
{
  // Thread A
  {
    if (input_is_as_usual)
      do_a_lot();
    x++;
  }
  // Thread B
  {
    do_a_little();
    x++;
  }
}
```

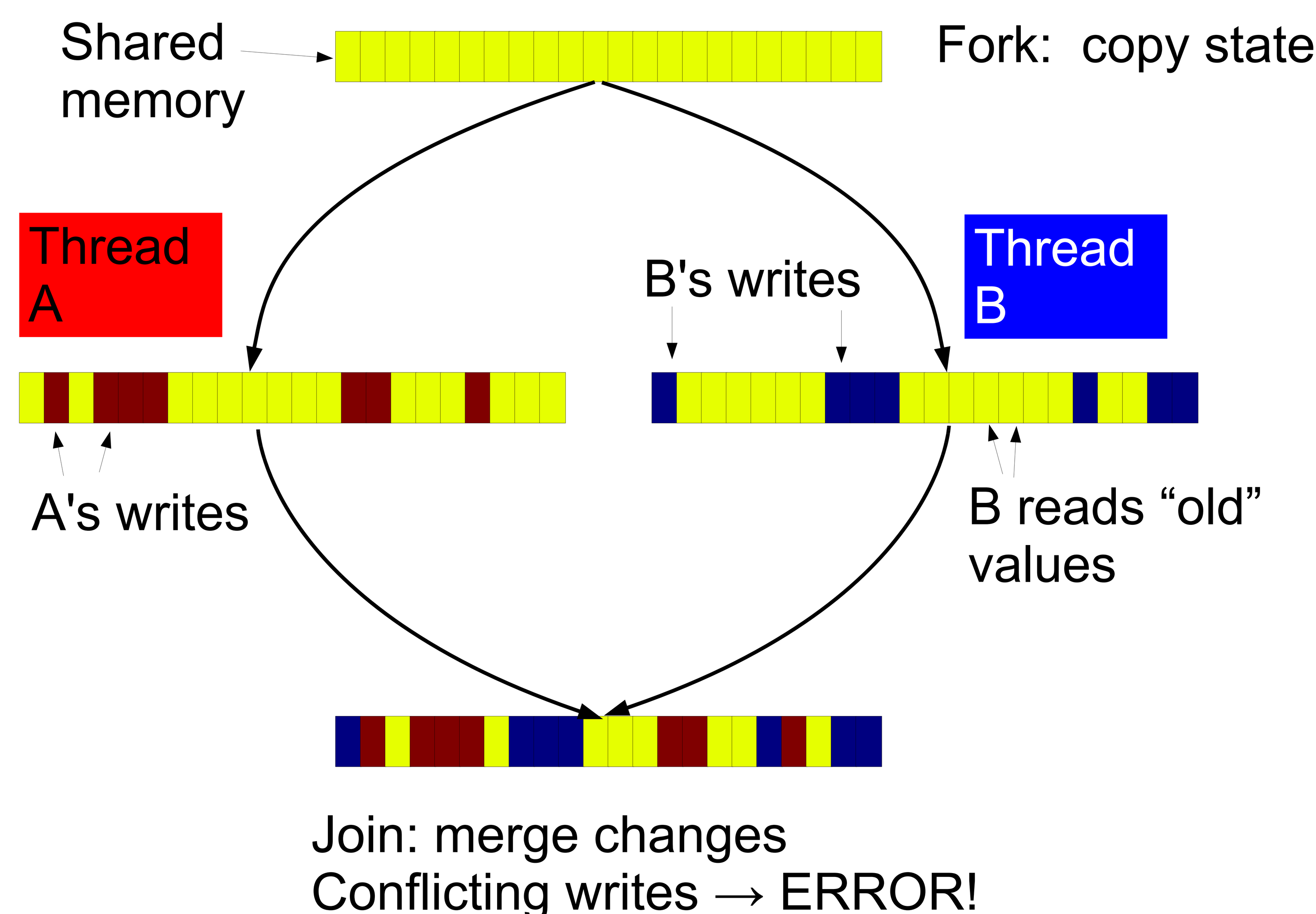
Programmer forgets to synchronize

Tests run great!

On “unusual” input, deterministic scheduler may *always* give 6! ☹

This code should *never* to work!

DOMP's Race-Free Deterministic Runtime



DOMP's Deterministic Abstractions

Parallel Work Sharing

```
void matrixMultiply(int n, int m, int p,
    double ** A, double ** B, double ** C) {
    #pragma omp parallel for
    for (int i = 0; i < n; i++)
      for (int j = 0; j < p; j++) {
        C[i][j] = 0.0;
        for (int k = 0; k < m; k++)
          C[i][j] += A[i][k] * B[k][j];
      }
}
```

Gives each thread chunk of work and copy of shared data

Merges children's writes into parent's copy (if no data races)

Reductions: Race-Free Aggregate Results

```
int x = 5;
#pragma omp parallel sections reduction(+: x)
{
  #pragma omp section
  {
    if (input_is_as_usual)
      do_a_lot();
    x++;
  }
  #pragma omp section
  {
    do_a_little();
    x++;
  }
}
printf("x = %d\n", x) // Always prints 7 !
```

Each thread gets a copy of *x*

At join, parent's *x* is overall sum