

Introduction

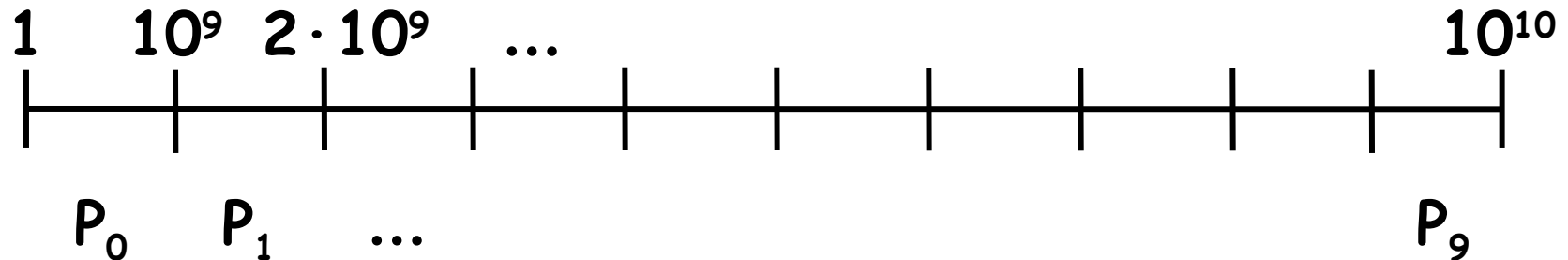
Companion slides for
The Art of Multiprocessor
Programming
by Maurice Herlihy & Nir Shavit

(Abridged version. Original at <http://booksite.elsevier.com/9780123705914/?ISBN=9780123705914>)

Parallel Primality Testing

- Challenge
 - Print primes from 1 to 10^{10}
- Given
 - Ten-processor multiprocessor
 - One thread per processor
- Goal
 - Get ten-fold speedup (or close)

Load Balancing



- Split the work evenly
- Each thread tests range of 10^9

Procedure for Thread i

```
void primePrint {  
    int i = ThreadID.get(); // IDs in {0..9}  
    for (j = i*109+1, j<(i+1)*109; j++) {  
        if (isPrime(j))  
            print(j);  
    }  
}
```

Issues

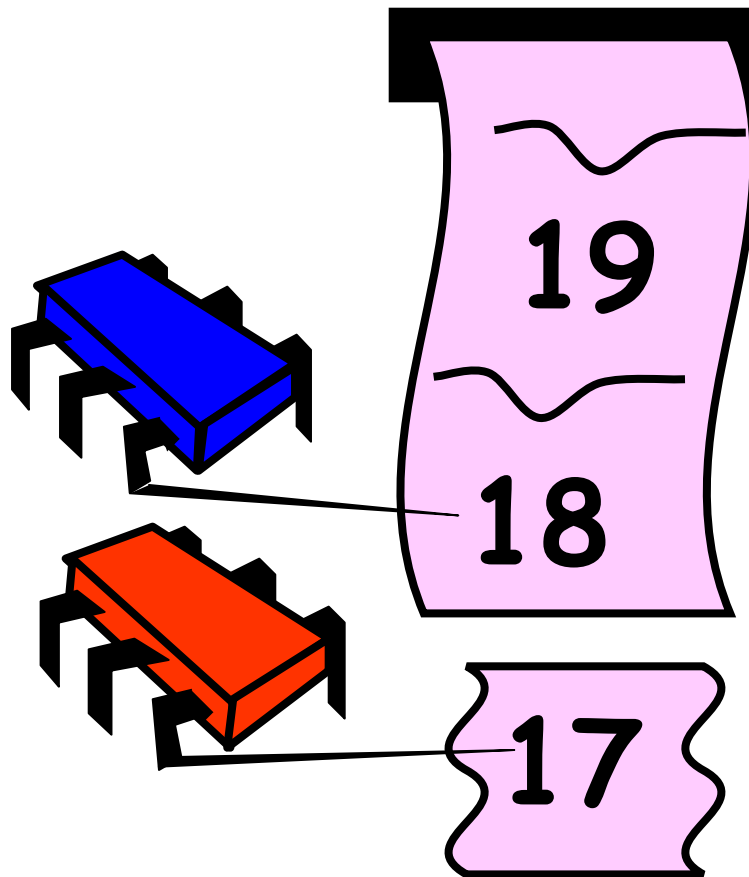
- Higher ranges have fewer primes
- Yet larger numbers harder to test
- Thread workloads
 - Uneven
 - Hard to predict

Issues

- Higher ranges have fewer primes
- Yet larger numbers harder to test
- Thread workloads
 - Uneven
 - Hard to predict
- Need *dynamic* load balancing

rejected

Shared Counter



each thread
takes a
number

Procedure for Thread *i*

```
int counter = new Counter(1);  
  
void primePrint {  
    long j = 0;  
    while (j < 1010) {  
        j = counter.getAndIncrement();  
        if (isPrime(j))  
            print(j);  
    }  
}
```


Procedure for Thread *i*

Counter counter = new Counter(1);

```
void primePrint {  
    long j = 0;  
    while (j < 1010) {  
        j = counter.getAndIncrement();  
        if (isPrime(j))  
            print(j);  
    }  
}
```

**Shared counter
object**

Procedure for Thread i

```
Counter counter = new Counter(1);
```

```
void primePrint {
```

```
    long j = 0;
```

```
    while (j < 1010) {
```

```
        j = counter.getAndIncrement();
```

```
        if (isPrime(j))
```

```
            print(j);
```

```
    }
```

```
}
```

Stop when every
value taken

Procedure for Thread *i*

```
Counter counter = new Counter(1);
```

```
void primePrint {
```

```
    long j = 0;
```

```
    while (j < 1010) {
```

```
        j = counter.getAndIncrement();
```

```
        if (isPrime(j))
```

```
            print(j);
```

```
    }
```

```
}
```

**Increment &
return each new
value**

Counter Implementation

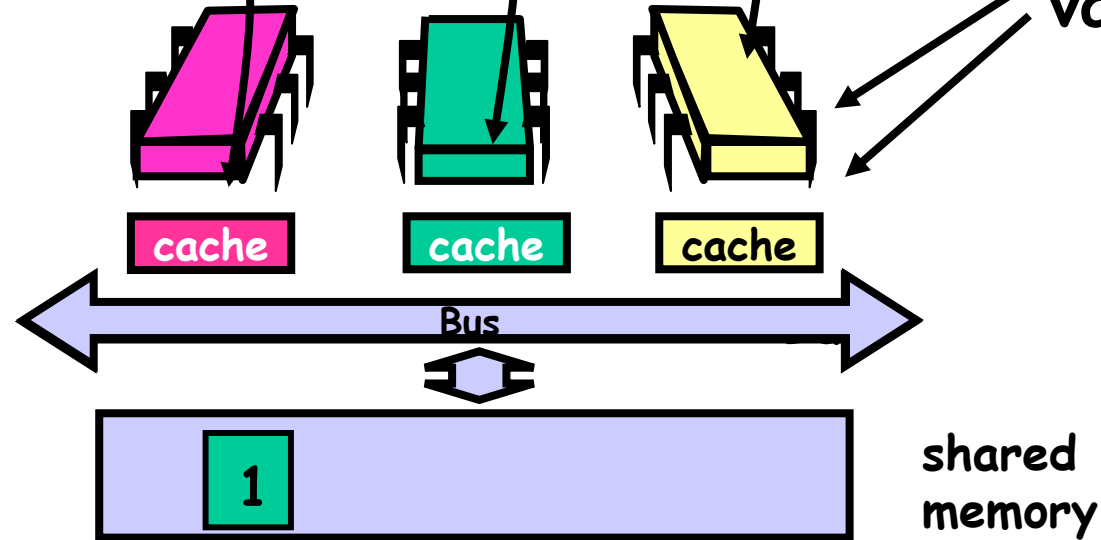
```
public class Counter {  
    private long value;  
  
    public long getAndIncrement() {  
        return value++;  
    }  
}
```

Where Things Reside

```
void primePrint {  
    int i =  
    ThreadID.get(); // IDs  
    in {0..9}  
    for (j = i*10+1,  
        j<(i+1)*10; j++) {  
        if (isPrime(j))  
            print(j);  
    }  
}
```

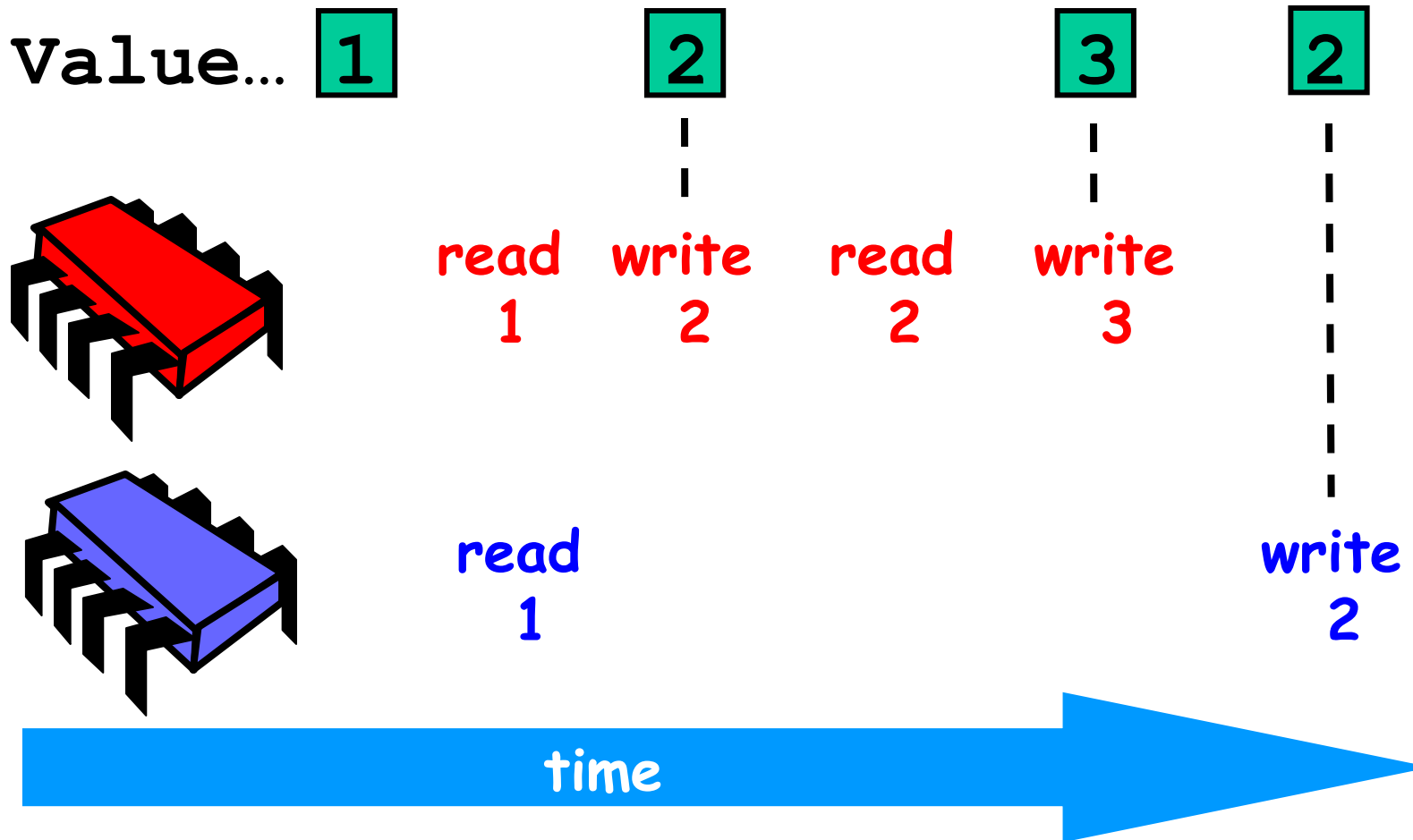
code

Local
variables



shared counter

Not so good...



Challenge

```
public class Counter {  
    private long value;
```

```
    public long getAndIncrement() {  
        temp = value;  
        value = temp + 1;  
        return temp;  
    }  
}
```

**OK for single thread,
not for concurrent threads**

Challenge

```
public class Counter {  
    private long value;  
  
    public long getAndIncrement() {  
        temp = value;  
        value = temp + 1;  
        return temp;  
    }  
}
```

**Make these steps
atomic (indivisible)**

This work is licensed under a Creative Commons Attribution-ShareAlike 2.5 License.

- You are free:
 - to Share — to copy, distribute and transmit the work
 - to Remix — to adapt the work
- Under the following conditions:
 - Attribution. You must attribute the work to “The Art of Multiprocessor Programming” (but not in any way that suggests that the authors endorse you or your use of the work).
 - Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.
- For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to
 - <http://creativecommons.org/licenses/by-sa/3.0/>.
- Any of the above conditions can be waived if you get permission from the copyright holder.
- Nothing in this license impairs or restricts the author's moral rights.