Language-level Complex Transactions

C. Scott Ananian

cananian@csail.mit.edu

Computer Science and Artifical Intelligence Laboratory
Massachusetts Institute of Technology

Monitor Synchronization

```
public class Count {
 private int cntr = 0;
 void inc() {
  synchronized(this) {
   cntr = cntr + 1;
  }
 }
}
```

- Traditionally, monitors associated with each object provide mutual exclusion between concurrent accesses to the object.

Monitor Synchronization

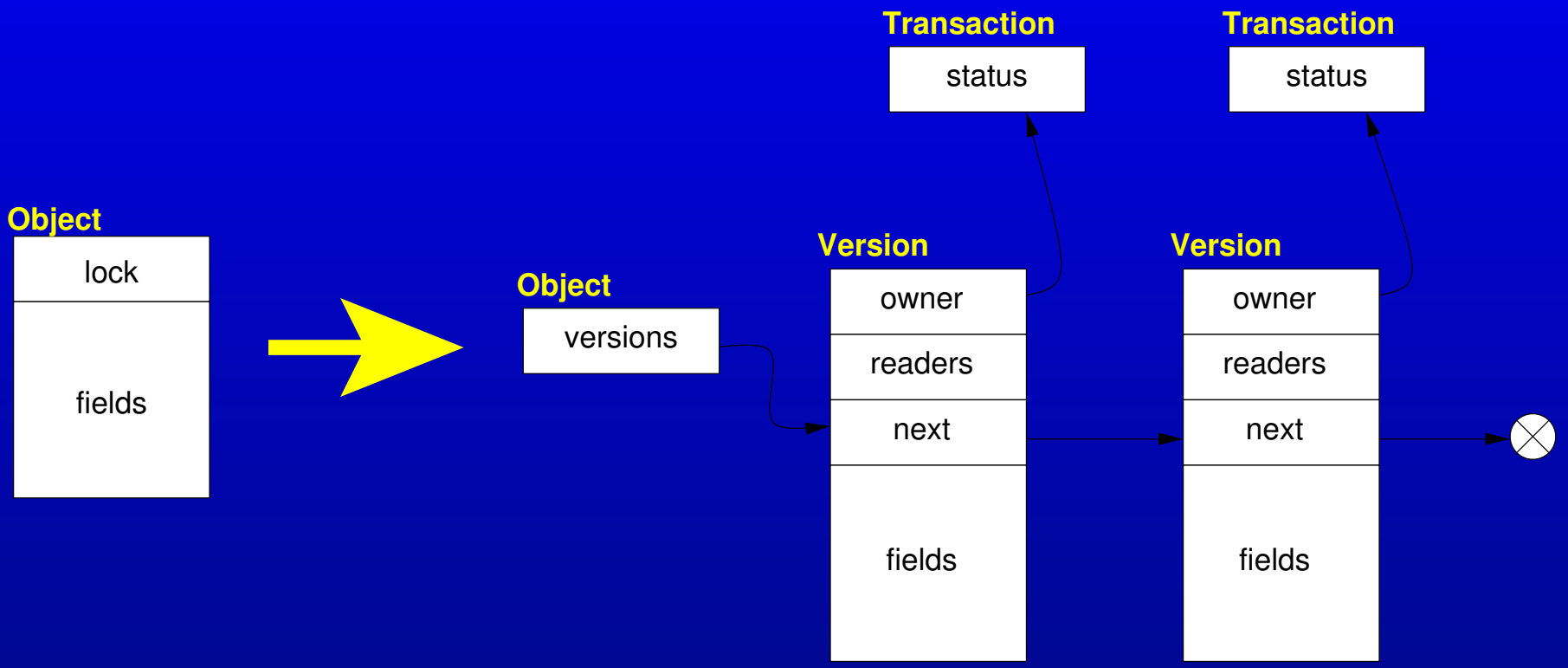
```
public class Count {
 private int cntr = 0;
 void inc() {
  synchronized(this) {
   cntr = cntr + 1;
  }
 }
}
```

$\Rightarrow$

```
public class Count {
 private int cntr = 0;
 void inc() {
  atomically {
   cntr = cntr + 1;
  }
 }
}
```

- Instead we provide an `atomic` block, and make linearizablity guarantees without (necessarily) providing mutual exclusion.

An implementation



Traditional ***Transactional***

Optimistic parallelism

```
for (...)
  optimistically {
    ...do an iteration ...
  }
```

```
conquer(A[n], n) {
  ...
  optimistic spawn
    conquer(A, n/2);
  optimistic spawn
    conquer(A+n/2, n-n/2);
}
```

Programmer notes that the iterations or spawns are *expected* to be independent. Iff there are dynamic dependencies, the computations are serialized.