Mutual Exclusion

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

Mutual Exclusion

- Today we will try to formalize our understanding of mutual exclusion

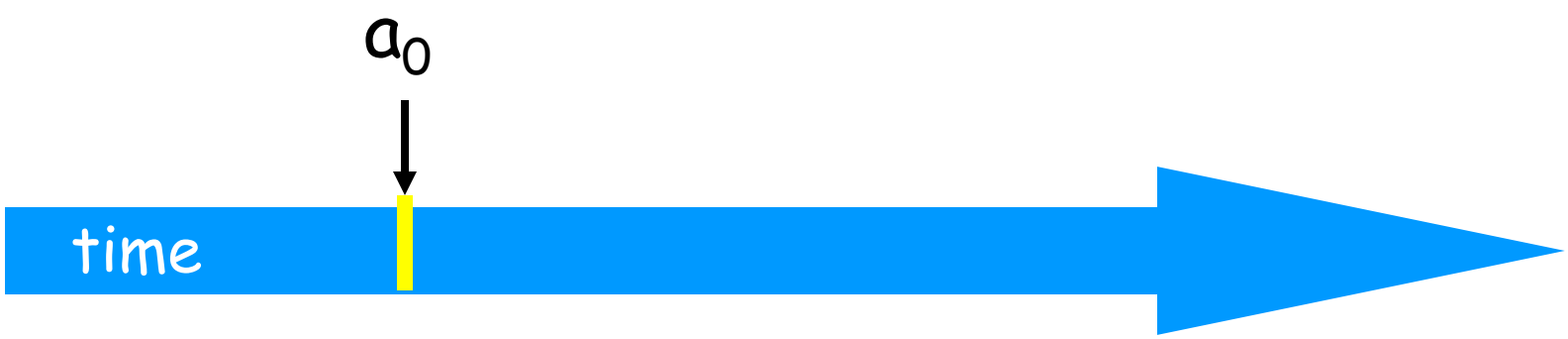
Time

- "Absolute, true and mathematical time, of itself and from its own nature, flows equably without relation to anything external." (I. Newton, 1689)

- "Time is, like, Nature's way of making sure that everything doesn't happen all at once." (Anonymous, circa 1968)

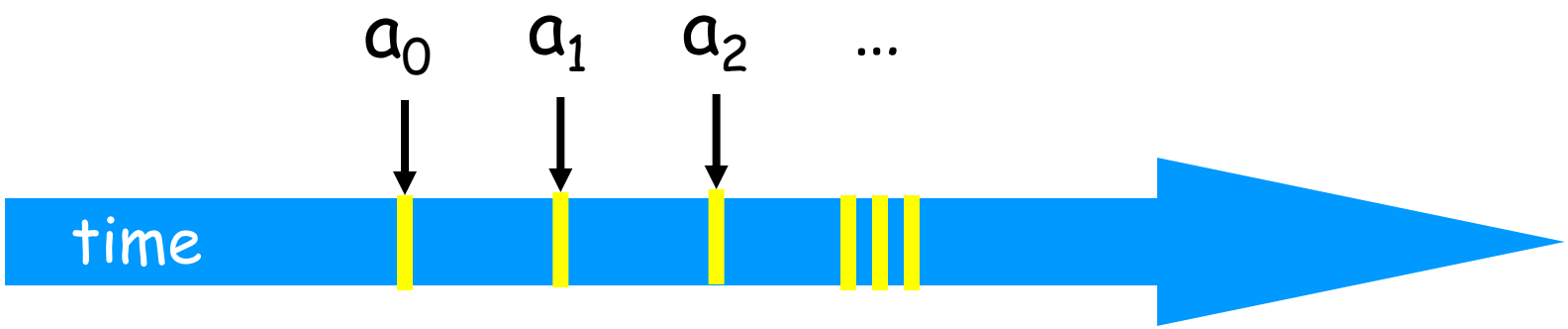
Events

- An *event* a_0 of thread A is
  - Instantaneous
  - No simultaneous events (break ties)



Threads

- A *thread* A is (formally) a sequence $a_0, a_1, \ldots$ of events
  - Notation: $a_0 \rightarrow a_1$ indicates order

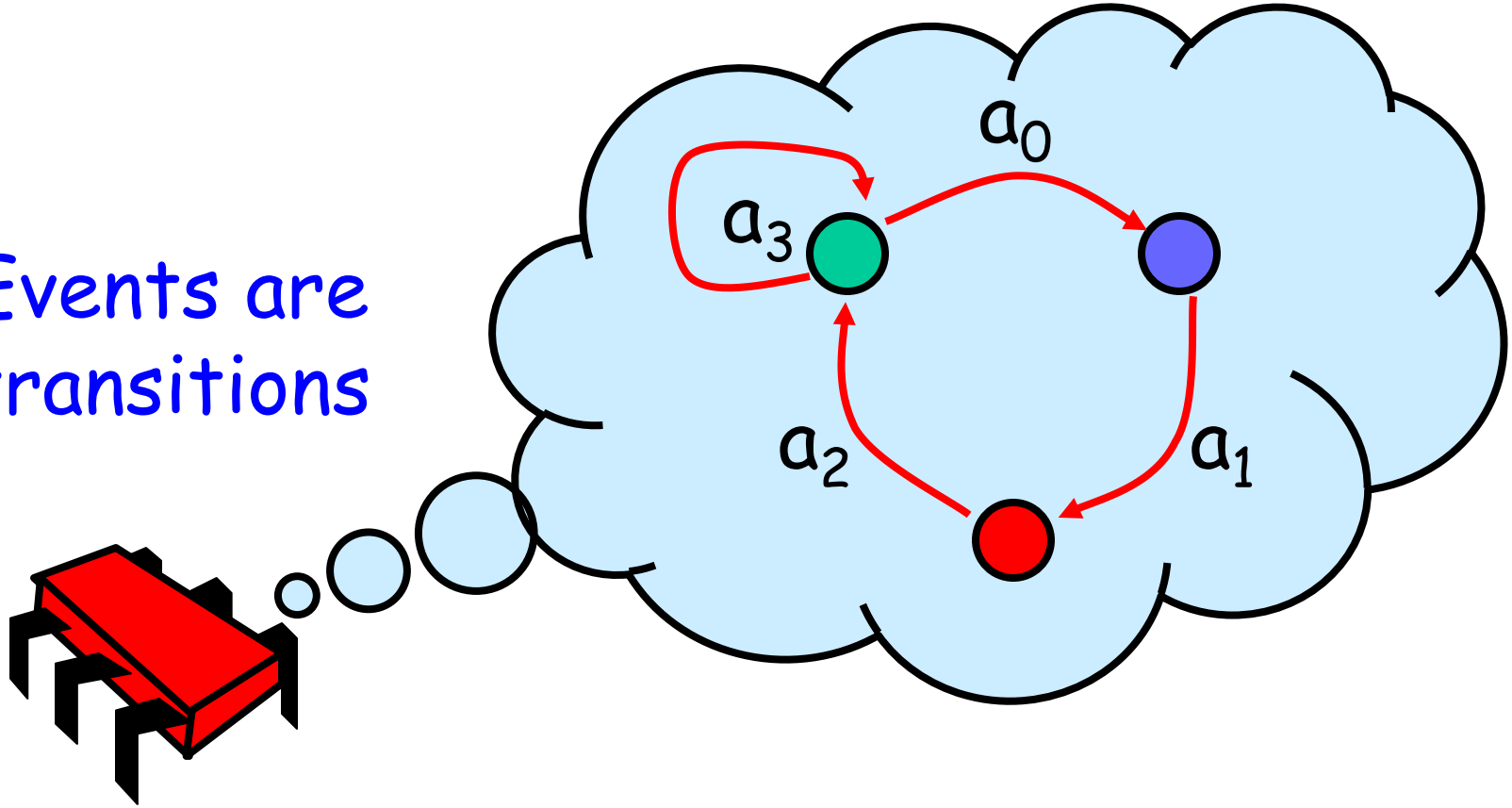


Example Thread Events

- Assign to shared variable
- Assign to local variable
- Invoke method
- Return from method
- Lots of other things …

Threads are State Machines

Events are transitions



States

- Thread State
  - Program counter
  - Local variables
- System state
  - Object fields (shared variables)
  - Union of thread states

Concurrency

- Thread A

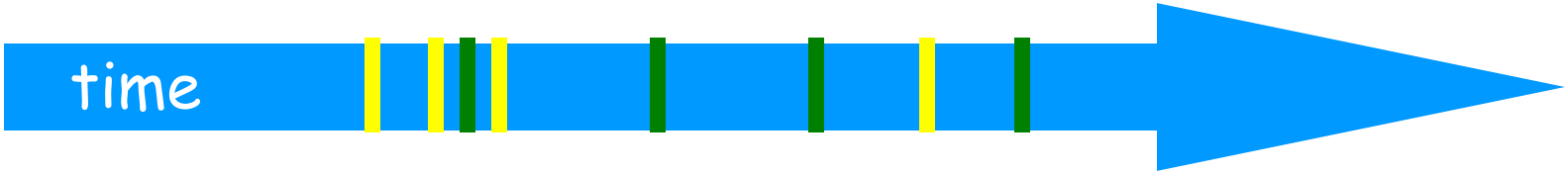
Concurrency

- Thread A

- Thread B

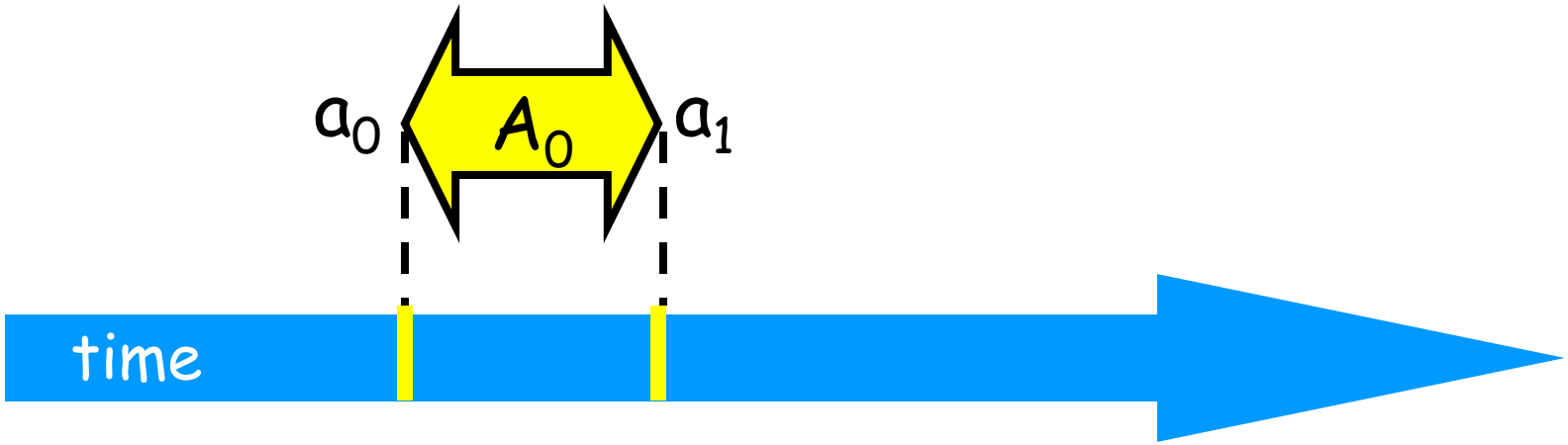
Interleavings

- Events of two or more threads
  - Interleaved
  - Not necessarily independent (why?)

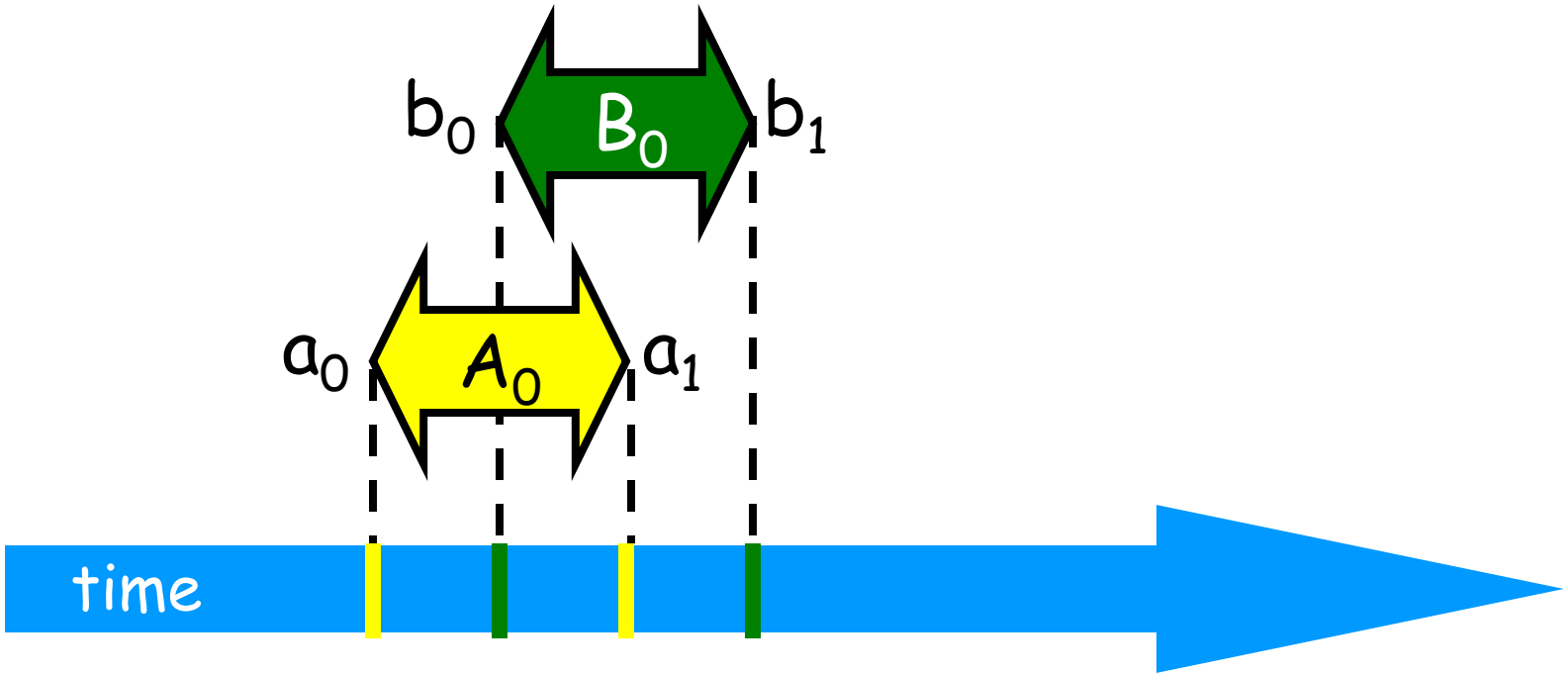


Intervals

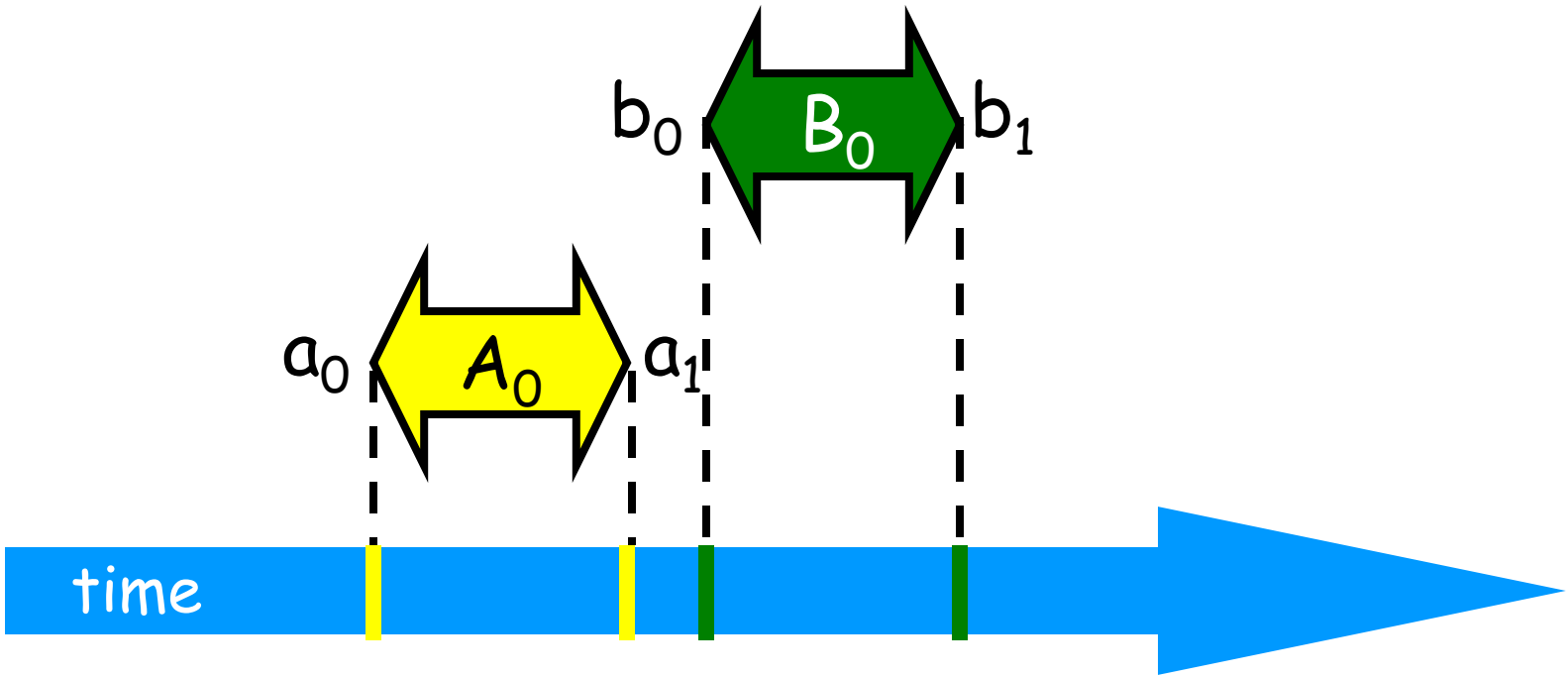
- An *interval* $A_0 = (a_0, a_1)$ is
  - Time between events a_0 and a_1



Intervals may Overlap

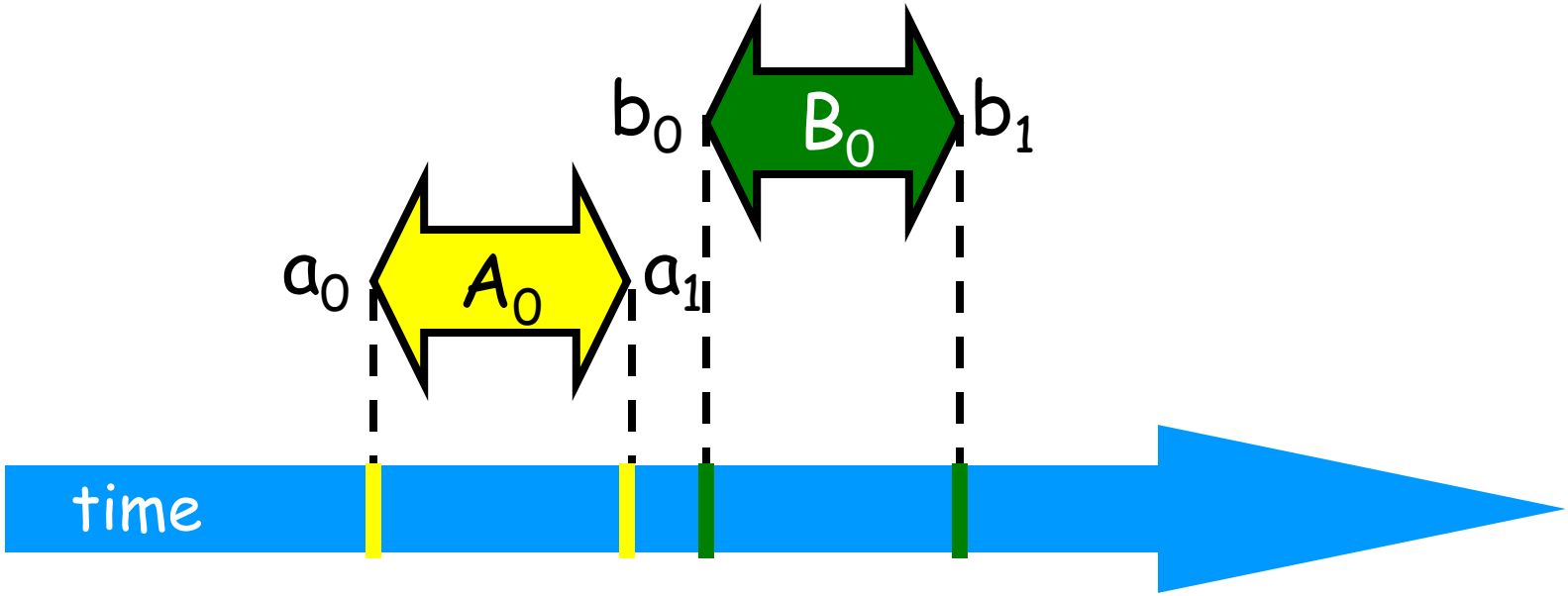


Intervals may be Disjoint

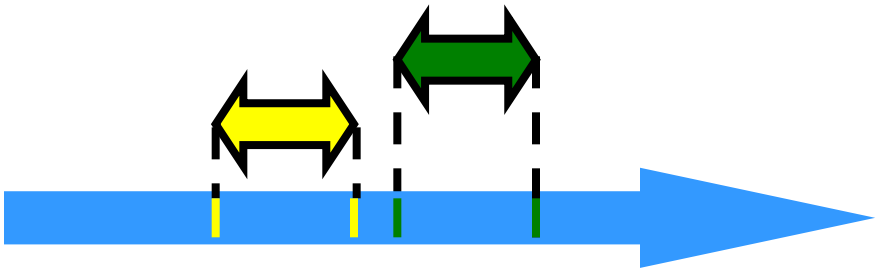


Precedence

Interval A_0 precedes interval B_0

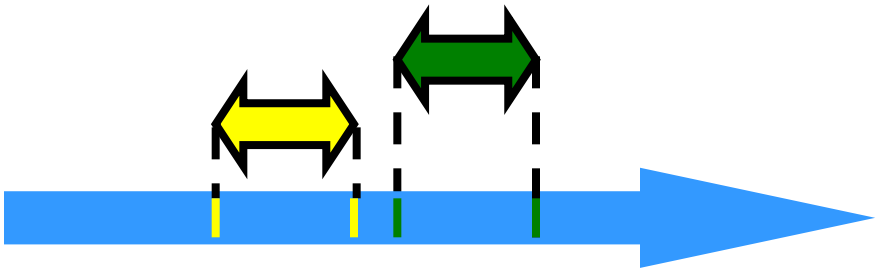


Precedence



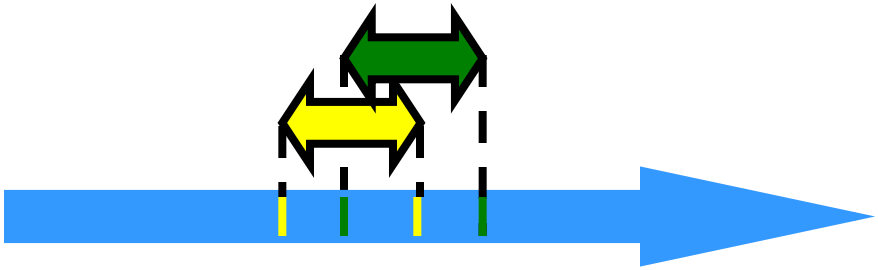
- Notation: $A_0 \rightarrow B_0$
- Formally,
  - End event of A_0 before start event of B_0
  - Also called "happens before" or "precedes"

Precedence Ordering



- Remark: $A_0 \rightarrow B_0$ is just like saying
  - 1066 AD → 1492 AD,
  - Middle Ages → Renaissance,
- Oh wait,
  - what about this week vs this month?

Precedence Ordering



- Never true that A → A
- If A → B then not true that B → A
- If A → B & B → C then A → C
- Funny thing: A → B & B → A might both be false!

Partial Orders

(you may know this already)

- Irreflexive:
  - Never true that $A \rightarrow A$
- Antisymmetric:
  - If $A \rightarrow B$ then not true that $B \rightarrow A$
- Transitive:
  - If $A \rightarrow B$ & $B \rightarrow C$ then $A \rightarrow C$

Total Orders

(you may know this already)

- Also
  - Irreflexive
  - Antisymmetric
  - Transitive
- Except that for every distinct A, B,
  - Either A → B or B → A

Repeated Events

```
while (mumble) {
   a0; a1;
}
```

k-th occurrence of event a_0

a_0^k

k-th occurrence of interval $A_0 = (a_0, a_1)$

A_0^k

Implementing a Counter

```
public class Counter {
  private long value;

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

Make these steps *indivisible* using locks

Locks (Mutual Exclusion)

```
public interface Lock {

 public void lock();

 public void unlock();
}
```

Locks (Mutual Exclusion)

```
public interface Lock {

 public void lock();

 public void unlock();
}
```

acquire lock

Locks (Mutual Exclusion)

```
public interface Lock {
  public void lock();
  public void unlock();
}
```

acquire lock

release lock

Using Locks

```
public class Counter {
  private long value;
  private Lock lock;
  public long getAndIncrement() {
   lock.lock();
   try {
    int temp = value;
    value = value + 1;
   } finally {
     lock.unlock();
   }
   return temp;
  }}
```

Using Locks

```
public class Counter {
  private long value;
  private Lock lock;
  public long getAndIncrement() {
   lock.lock();
   try {
    int temp = value;
    value = value + 1;
   } finally {
    lock.unlock();
   }
   return temp;
  }}
```

acquire Lock

Using Locks

```
public class Counter {
  private long value;
  private Lock lock;
  public long getAndIncrement() {
   lock.lock();
   try {
    int temp = value;
    value = value + 1;
   } finally {
    lock.unlock();
   }
   return temp;
  }}
```

Release lock
(no matter what)

Using Locks

```
public class Counter {
  private long value;
  private Lock lock;
  public long getAndIncrement() {
   lock.lock();
   try {
    int temp = value;
    value = value + 1;
   } finally {
     lock.unlock();
   }
   return temp;
  }}
```

Critical section

Mutual Exclusion

- Let CS_i^k ⇔ be thread i's k-th critical section execution

Mutual Exclusion

- Let CS_i^k ⇔ be thread i's k-th critical section execution
- And CS_j^m ⇔ be thread j's m-th critical section execution

Mutual Exclusion

- Let CS_i^k ⇔ be thread i's k-th critical section execution
- And CS_j^m ⇔ be j's m-th execution
- Then either
  - ⇔ ⇔ or ⇔ ⇔

Mutual Exclusion

- Let CS_i^k ⇔ be thread i's k-th critical section execution
- And CS_j^m ⇔ be j's m-th execution
- Then either
  - ⇔ ⇔ or ⇔ ⇔

$CS_i^k \rightarrow CS_j^m$

Mutual Exclusion

- Let CS_i^k ⇔ be thread i's k-th critical section execution
- And CS_j^m ⇔ be j's m-th execution
- Then either
  - ⇔ ⇔ or ⇔ ⇔

$CS_i^k \rightarrow CS_j^m$

$CS_j^m \rightarrow CS_i^k$

Deadlock-Free

- System as a whole makes progress
  - Even if individuals starve

Starvation-Free

- Individual threads make progress