Cooperating Threads

Sarah Diesburg

Operating Systems

CS 3430

Independent Threads

- No states shared with other threads
- Deterministic computation
  - Output depends on input
- Reproducible
  - Output does not depend on the order and timing of other threads
- Scheduling order does not matter
- e.g., compilers

Cooperating Threads

- Shared states
- Nondeterministic
- Nonreproducible
- Example: 2 threads sharing the same display

Thread A	Thread B
printf(“ABC”);	printf(“123”);

- You may get “A12BC3”

So, Why Allow Cooperating Threads?

So, Why Allow Cooperating Threads?

- Shared resources
  - e.g., a single processor
- Speedup
  - Occurs when threads use different resources at different times
- Modularity
  - An application can be decomposed into threads

Some Concurrent Programs

- If threads work on separate data, scheduling does not matter

```
Thread A      Thread B
x = 1;        y = 2;
```

Some Concurrent Programs

- If threads share data, the final values are not as obvious

```
Thread A        Thread B
x = 1;          y = 2;
x = y + 1;      y = y * 2;
```

- What are the indivisible operations?

Atomic Operations

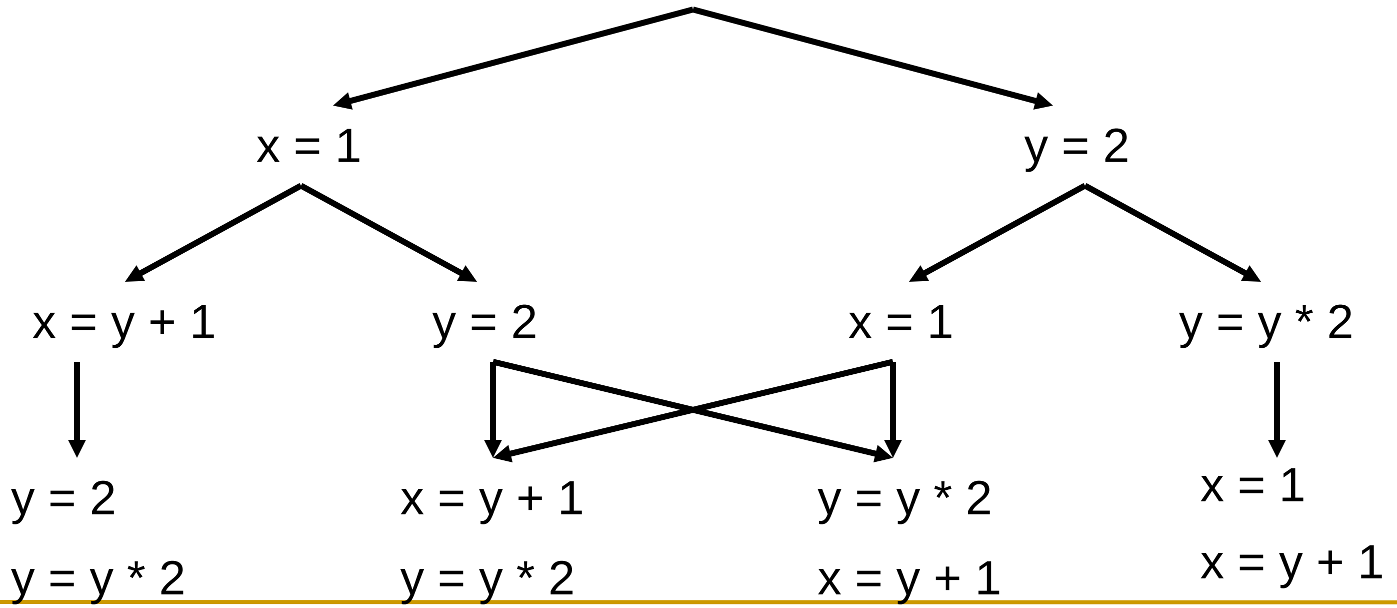
- An ***atomic operation*** always runs to completion; it's all or nothing
    - e.g., memory loads and stores on most machines
- Many operations are not atomic
    - Double precision floating point store on 32-bit machines

Suppose…

- Each C statement is atomic
- Let’s revisit the example…

All Possible Execution Orders

Thread A	Thread B
x = 1;	y = 2;
x = y + 1;	y = y * 2;



x = 1

y = 2

x = y + 1

y = 2

x = 1

y = y * 2

y = 2

y = y * 2

x = y + 1

y = y * 2

y = y * 2

x = y + 1

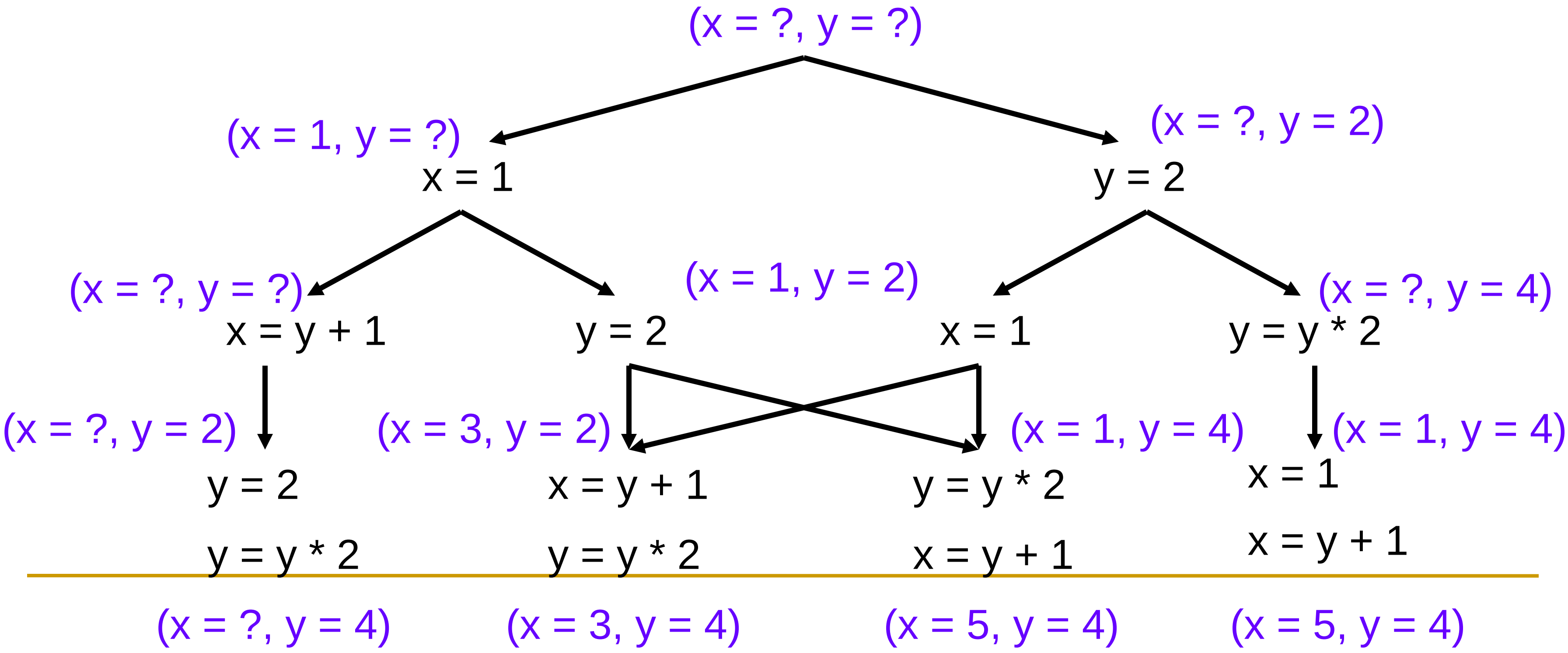
x = 1

x = y + 1

A decision tree

All Possible Execution Orders

Thread A	Thread B
x = 1;	y = 2;
x = y + 1;	y = y * 2;



(x = ?, y = ?)

(x = 1, y = ?)
x = 1

(x = ?, y = 2)
y = 2

(x = ?, y = ?)
x = y + 1

y = 2

(x = 1, y = 2)

x = 1

(x = ?, y = 4)
y = y * 2

(x = ?, y = 2)
y = 2
y = y * 2

(x = 3, y = 2)
x = y + 1
y = y * 2

(x = 1, y = 4)
y = y * 2
x = y + 1

(x = 1, y = 4)
x = 1
x = y + 1

(x = ?, y = 4)

(x = 3, y = 4)

(x = 5, y = 4)

(x = 5, y = 4)

Another Example

- Assume each C statement is atomic
  - Both threads are in the same address space

```
Thread A                    Thread B
j = 0;                      j = 0;
while (j < 10) {            while (j > -10) {
   ++j;                          --j;
}                           }
printf("A wins");           printf("B wins");
```

So…

- Who wins?
- Can the computation go on forever?

- ***Race conditions*** occur when threads share data, and their results depend on the timing of their executions…