Concurrent Programming (Part 2)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
  - Concurrent processes vs. concurrent threads
  - Race conditions
  - Preventing race conditions
  - Thread safety

Agenda

- **Process vs. thread concurrency**
- Race conditions
- Preventing race conditions: lock in user
- Preventing race conditions: lock in resource
- Thread safety

Process vs. Thread Concurrency

- Difference #1
  - **Process**-level concurrency
    - Multiple **processes** run concurrently
    - Parent process **forks** and **waits** for a child process
  - **Thread**-level concurrency
    - Multiple **threads** run concurrently within the same process
    - Within a process, parent thread **spawns** (and **joins**) a child thread

Process vs. Thread Concurrency

- Terminology review

Type of Concurrency	Generic Terms	Python Terms
Process-level concurrency	fork/wait	fork/**join**
Thread-level concurrency	spawn/join	spawn/join

Process vs. Thread Concurrency

- Difference #2
  - **Process**-level concurrency
    - Forking & context switching are relatively **slow**
  - **Thread**-level concurrency
    - Spawning & context switching are relatively **fast**

Process vs. Thread Concurrency

- Difference #3
  - **Process**-level concurrency
    - Concurrent processes **do not share objects**
  - **Thread**-level concurrency
    - Concurrent threads **do share objects**

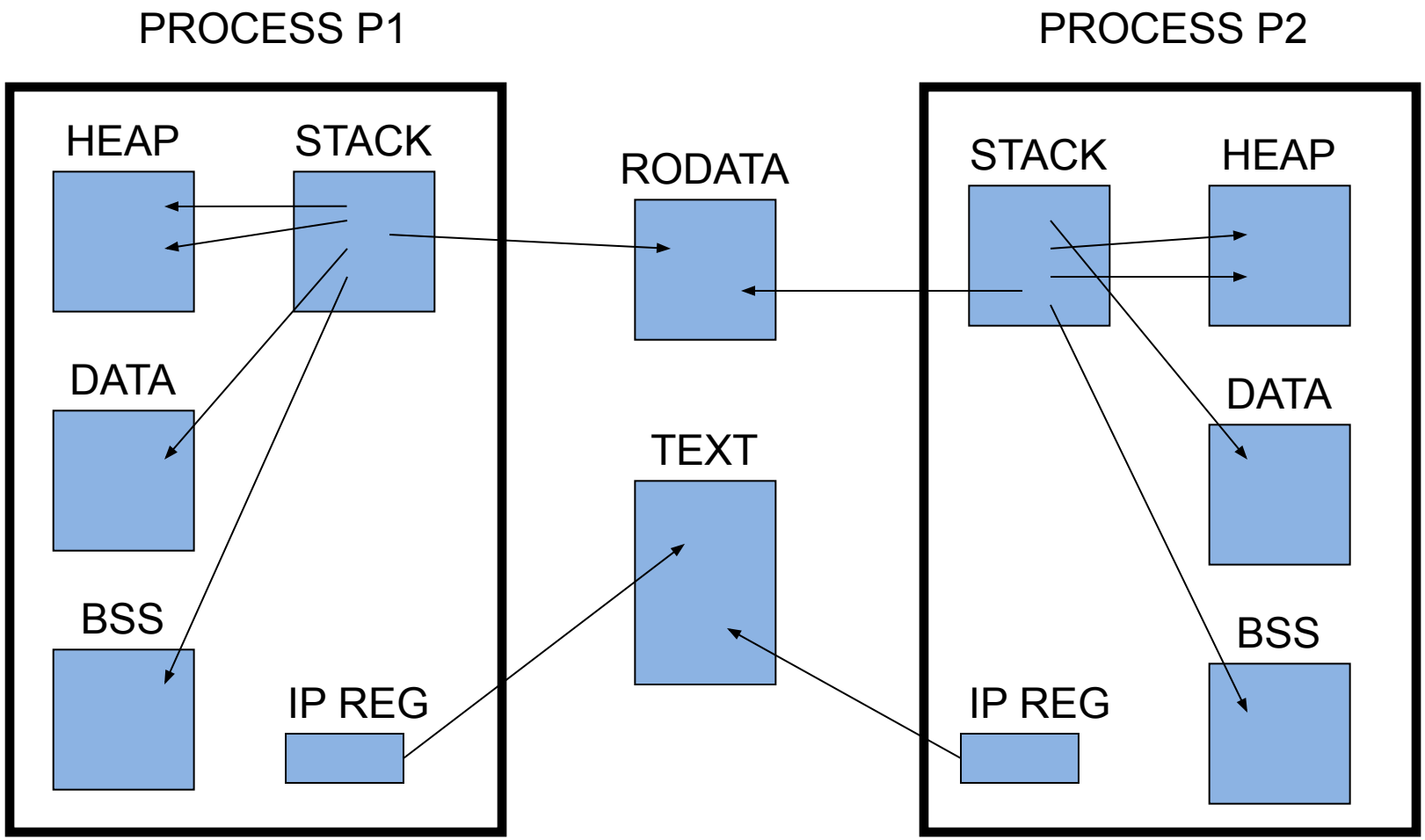
- Elaboration…

Process vs. Thread Concurrency

- **Process-level** concurrency
  - P1 and P2 **do not share** objects
    - P1 and P2 have (initially identical but) distinct memory address spaces

Process vs. Thread Concurrency

Concurrent Processes



Process vs. Thread Concurrency

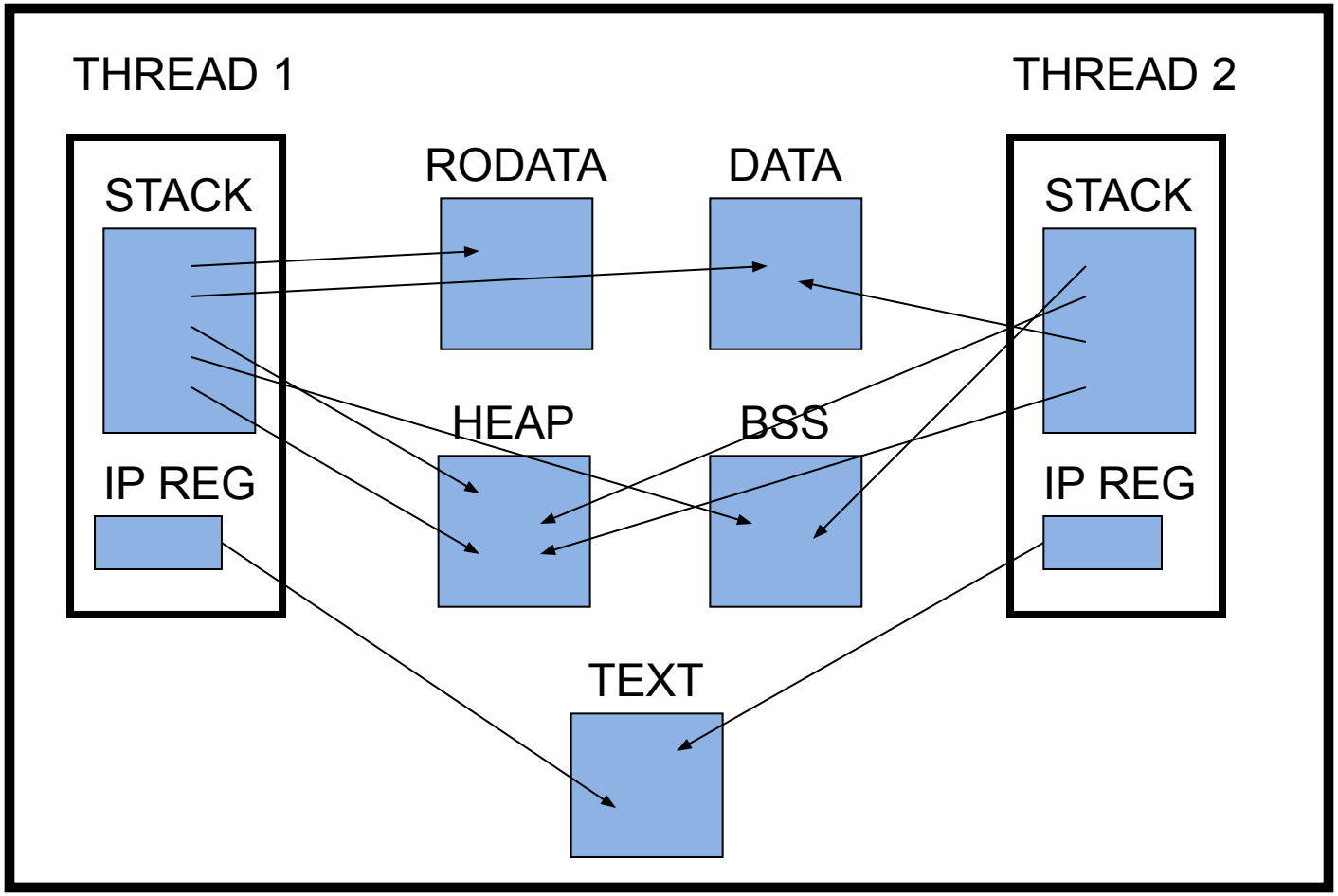
- See **processsharing.py**

Process vs. Thread Concurrency

- **Thread-level** concurrency
  - T1 and T2 **share** objects
    - T1 and T2 have distinct **STACK** sections
    - T1 and T2 share the RODATA, DATA, BSS, and **HEAP** sections

Process vs. Thread Concurrency

Concurrent Threads



Process vs. Thread Concurrency

- See **threadsharing.py**

Agenda

- Process vs. thread concurrency
- **Race conditions**
- Preventing race conditions: lock in user
- Preventing race conditions: lock in resource
- Thread safety

Race Conditions

- **Problem:**
  - Threads can share objects
  - Danger if multiple threads update/access the same object concurrently
  - ***Race condition***
    - Outcome depends upon thread scheduling

Race Conditions

- See **race.py**

```
$ python race.py
1
2
3
4
5
6
7
8
9
10
8
6
4
2
0
Final balance: 0
$
```

```
$ python race.py
1
2
3
4
-1
5
6
-3
-5
-7
-9
7
8
9
10
Final balance: 10
$
```

```
$ python race.py
1
2
3
4
5
6
7
8
9
6
4
10
2
0
-2
Final balance: -2
$
```

Race Conditions

- **Note:**
  - Use of shared `BankAcct` object by multiple threads causes unpredictable behavior
  - race.py contains a **race condition**

Agenda

- Process vs. thread concurrency
- Race conditions
- **Preventing race conditions: lock in user**
- Preventing race conditions: lock in resource
- Thread safety

Preventing Race Conditions: Lock in User

- **Observation**:
  - While a thread is executing `deposit()` or `withdraw()` on a particular `bank_acct` object…
  - No other thread should be able to execute `deposit()` or `withdraw()` on that `bank_acct` object

Preventing Race Conditions: Lock in User

- **Solution**: ***Locking***
  - Each object has an associated **lock**
  - All threads that will use object X agree to a **pact**: must acquire lock on X before using X
    - Current thread **acquires** lock on X
    - Other threads cannot acquire lock on X until current thread **releases** lock on X
  - (Adds lots of overhead)

Preventing Race Conditions: Lock in User

- **Approach 1**: Locking in **user** of shared object

Preventing Race Conditions: Lock in User

- See **lockinuser.py** (cont.)

```
$ python lockinuser.py
1
2
3
4
5
6
7
8
9
10
8
6
4
2
0
Final balance: 0
$
```

```
$ python lockinuser.py
1
2
3
4
2
0
-2
-4
-6
-5
-4
-3
-2
-1
0
Final balance: 0
$
```

Preventing Race Conditions: Lock in User

- See **lockinuserw.py**
  - Uses `with` statement

Agenda

- Process vs. thread concurrency
- Race conditions
- Preventing race conditions: lock in user
- **Preventing race conditions: lock in resource**
- Thread safety

Preventing Race Conditions: Lock in Resource

- **Approach 2**: Locking in shared resource/object itself

Preventing Race Conditions: Lock in Resource

- See **lockinresource.py** (cont.)

```
$ python lockinresource.py
1
2
3
4
5
6
7
8
9
10
8
6
4
2
0
Final balance: 0
$
```

```
$ python lockinresource.py
1
2
3
1
-1
-3
-5
-7
-6
-5
-4
-3
-2
-1
0
Final balance: 0
$
```

Preventing Race Conditions: Lock in Resource

- See **lockinresourcew.py**
  - Uses `with` statement

Preventing Race Conditions: Lock in Resource

- Which locking approach is better?
  - **User**-level locking: sometimes **faster**
  - **Resource**-level locking: **safer**

Agenda

- Process vs. thread concurrency
- Race conditions
- Preventing race conditions: lock in user
- Preventing race conditions: lock in resource
- **Thread safety**

Thread Safety

- Recall lockinresource.py
  - A context switch can occur between any 2 **machine lang** instructions
  - Implications:
    - `get_balance()` should be protected by locking
    - _balance should be private
      - But cannot be

Thread Safety

- ***Thread safety***
  - Oversimplification…
  - An object is **thread-safe** if all of its methods are “locked” & all of its fields are private

Thread Safety

- **Java**
  - Methods can be locked (`synchronized`)
  - Fields **can** be private
  - Objects can be thread-safe
- **Python**
  - Methods can be locked
  - Fields **cannot** be private
  - Any object that has fields cannot be thread-safe

Summary

- We have covered:
  - Concurrent processes vs. concurrent threads
  - Race conditions
  - Preventing race conditions
  - Thread safety