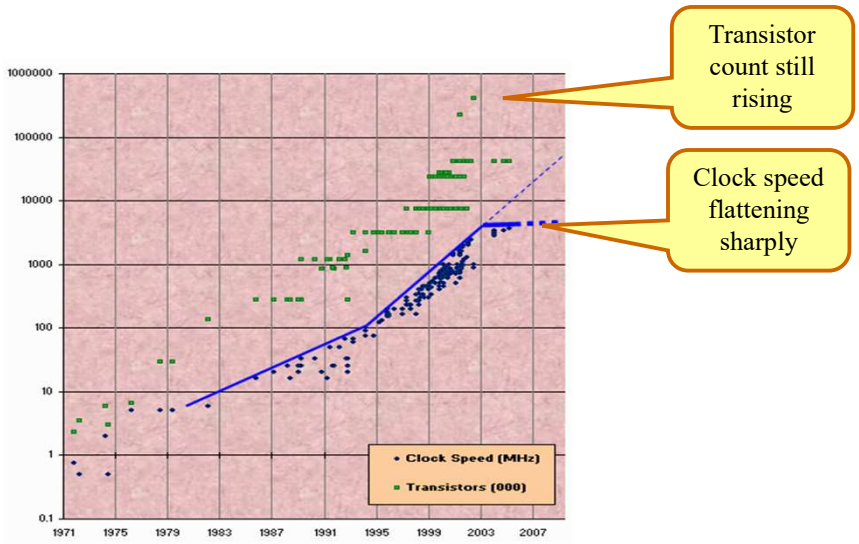
Project 4

CSCI-341: Software Engineering

CSIT at UDC

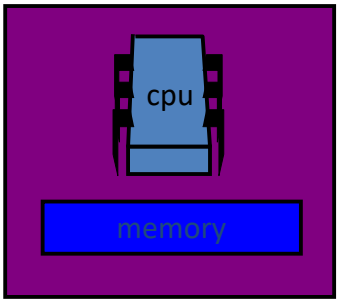
1

Moore's Law



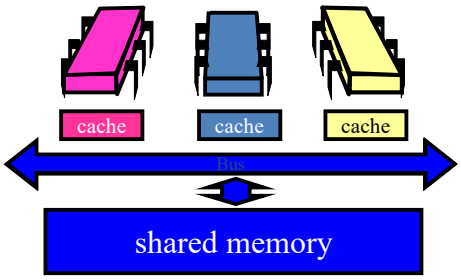
2

Still on some of your desktops: The Uniprocessor



3

In the Enterprise: The Shared Memory Multiprocessor (SMP)



4

Multicores Are Here

- Intel Core i9 (10 Cores) [Cnet]
- AMD Ryzen has 32 cores and Symmetrical Multi-Threading [Tech News]
- Meet KiloCore, a 1,000-core processor so efficient it could run on an AA battery [PC World]

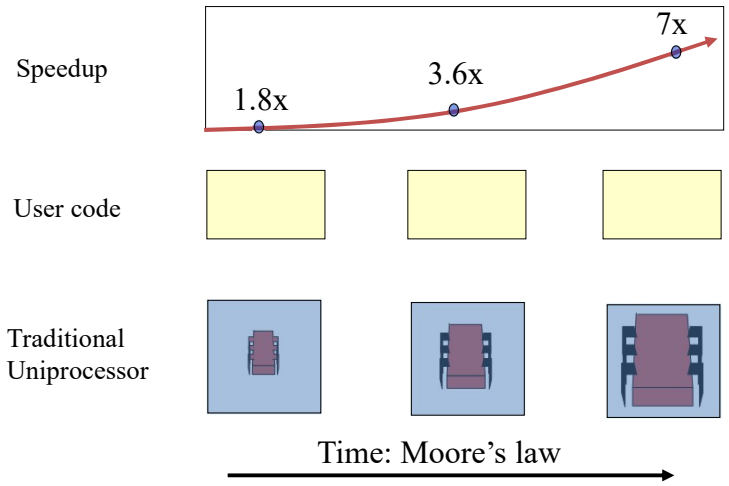
5

Why do we care?

- Time no longer cures software bloat
  - The “free ride” is over
- When you double your program’s path length
  - You can’t just wait 6 months
  - Your software must somehow exploit twice as much concurrency

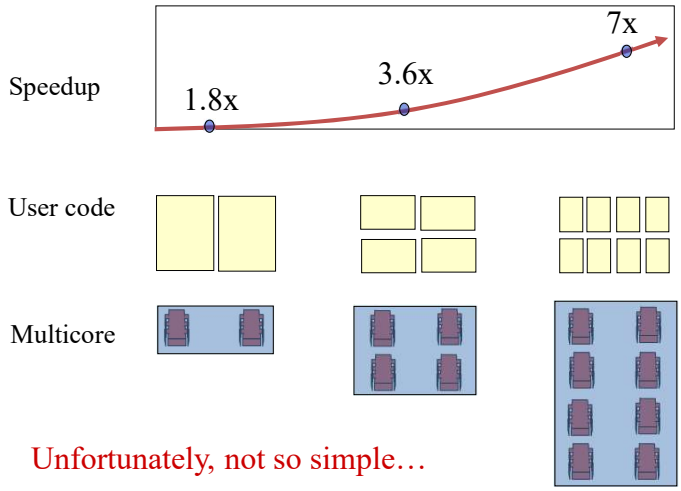
6

Traditional Scaling Process



7

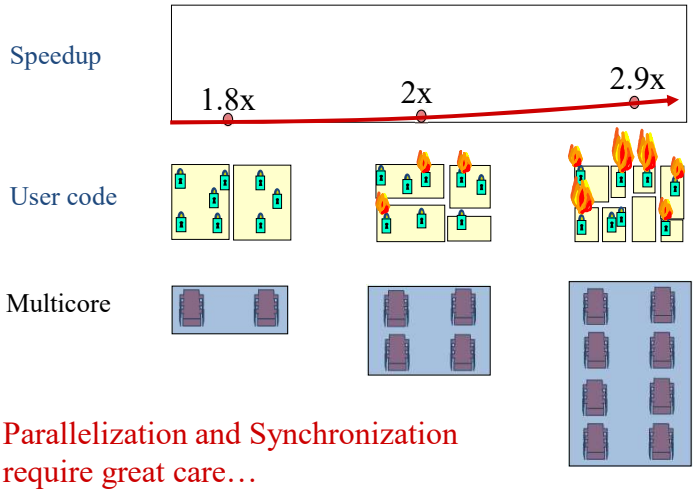
Multicore Scaling Process



Unfortunately, not so simple…

8

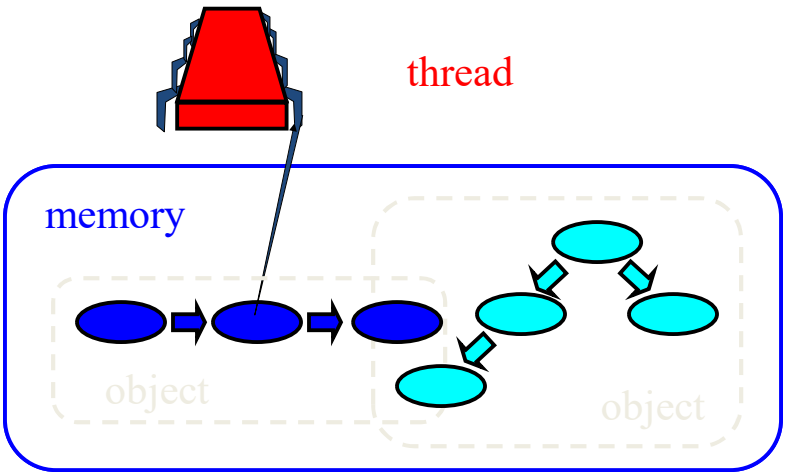
Real-World Scaling Process



Parallelization and Synchronization require great care…

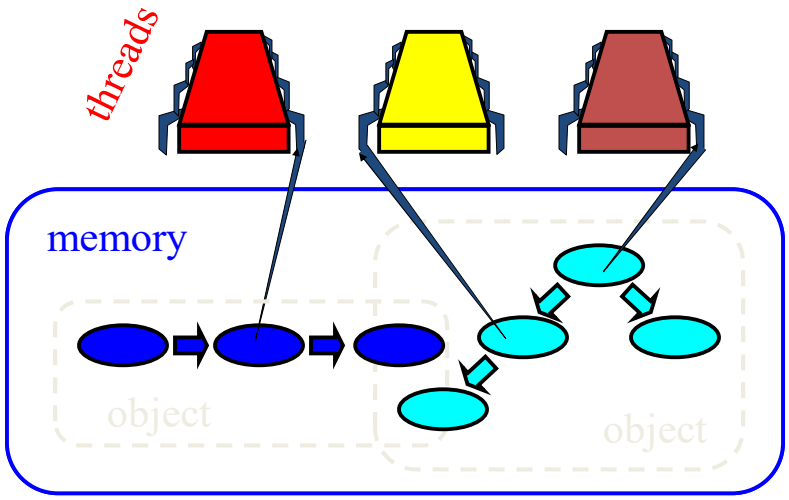
9

Sequential Computation



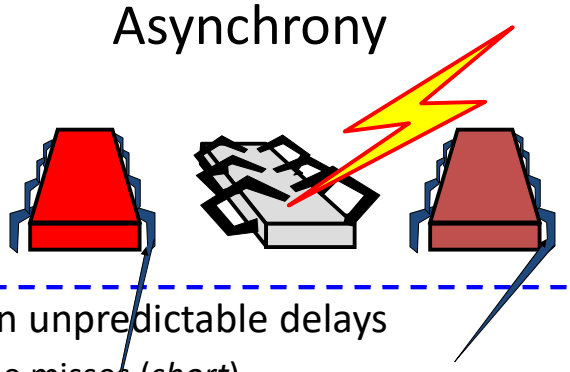
10

Concurrent Computation



11

Asynchrony



- Sudden unpredictable delays
  - Cache misses (*short*)
  - Page faults (*long*)
  - Scheduling quantum used up (*really long*)

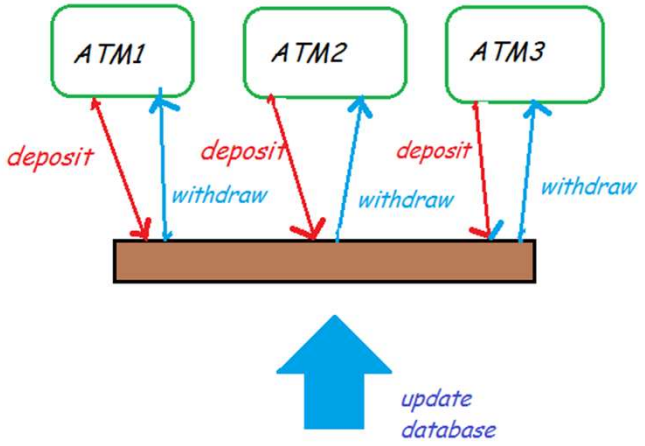
12

Distributed Systems

- **Advantages:**
  - **No bottlenecks from sharing processors**
  - **No central point of failure**
  - **Processing can be localized for efficiency**
- **Disadvantages:**
  - **Complexity**
  - **Communication overhead**
  - **Distributed control**

13

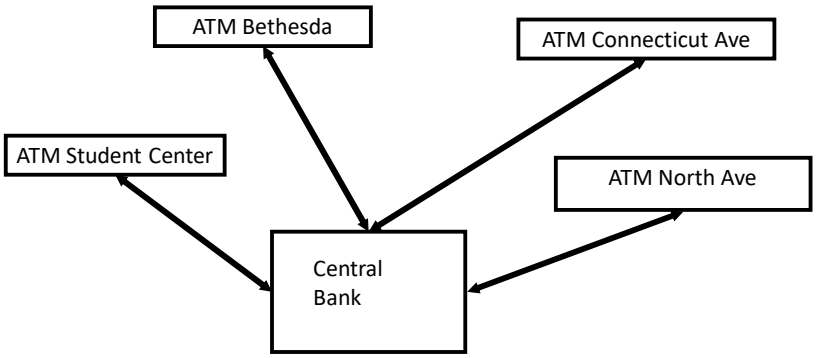
An Example of Distributed Concurrent Program



14

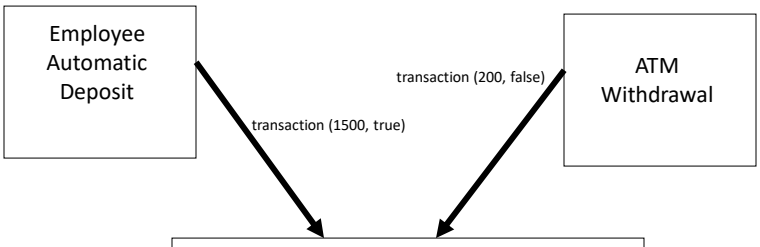
Distributed Systems

Multiple computers working together with no central program "in charge."



15

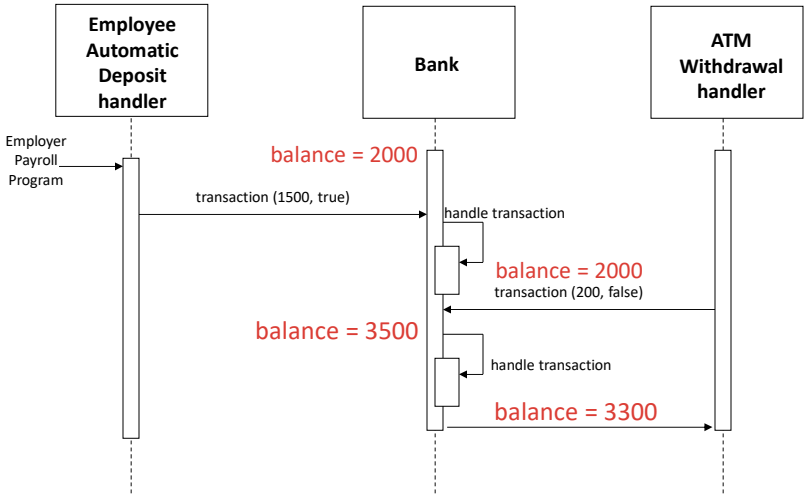
A Concurrent World on ATM




```
double balance;  // a global variable

void transaction (double amount, boolean deposit)
{
   if (deposit)
            balance = balance + amount;
   else
            balance = balance - amount;
}
```

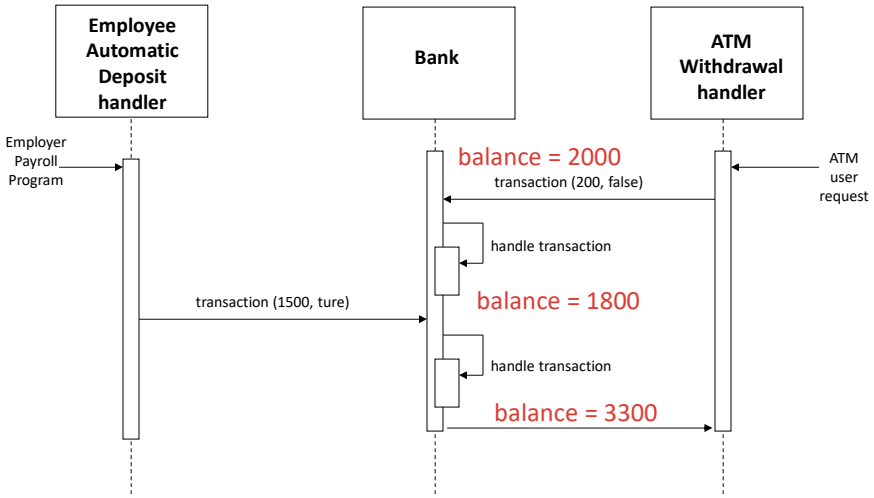
16

A Wrong Diagram



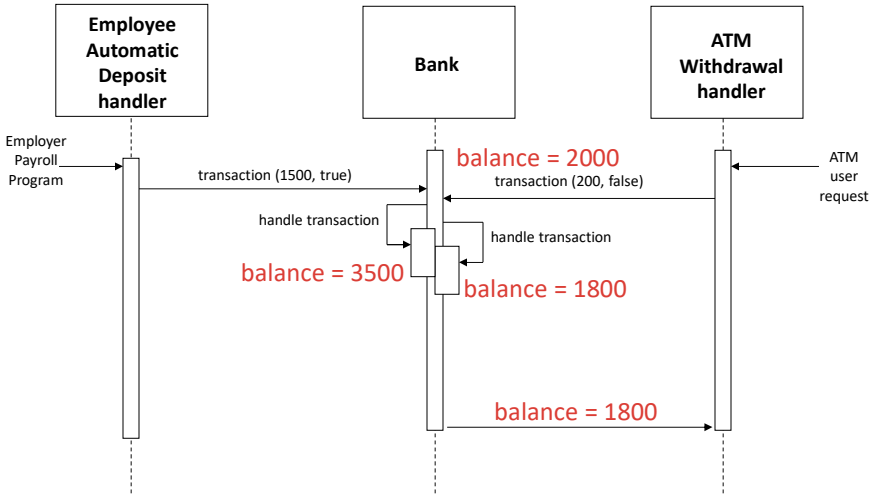
17

A Wrong Diagram



18

A Wrong Diagram



19

Counter Implementation

```
public class Counter {
  private long value;

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

Source: Art of Multiprocessor Programming

20

Counter Implementation

```
public class Counter {
  private long value;

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

OK for single thread,
not for concurrent threads

Source: Art of Multiprocessor Programming

21

What It Means

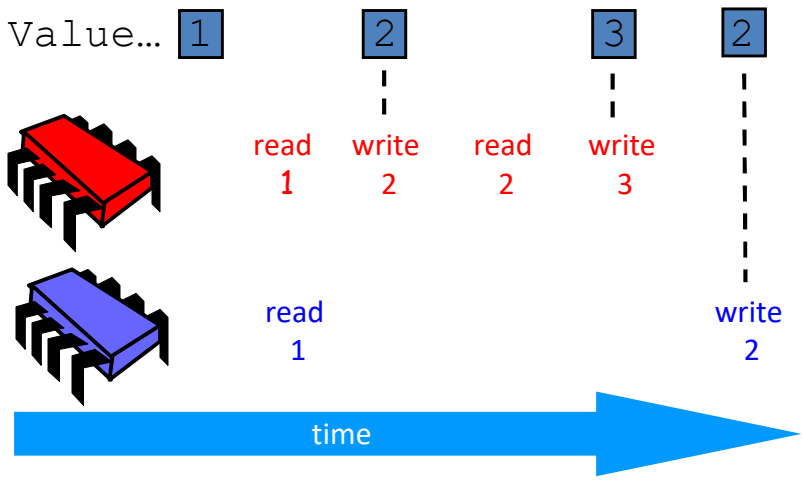
```
public class Counter {
  private long value;

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

```
temp  = value;
value = temp + 1;
return temp;
```

Source: Art of Multiprocessor Programming

22

Not so good…



Source: Art of Multiprocessor Programming

23

Challenge

```
public class Counter {
  private long value;

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

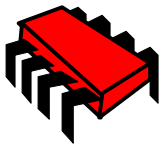
Make these steps *atomic* (indivisible)

Source: Art of Multiprocessor Programming

24

Hardware Solution

```
public class Counter {
  private long value;

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



ReadModifyWrite() instruction

Source: Art of Multiprocessor Programming

25

An Aside: Java™

```
public class Counter {
  private long value;

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

Source: Art of Multiprocessor Programming

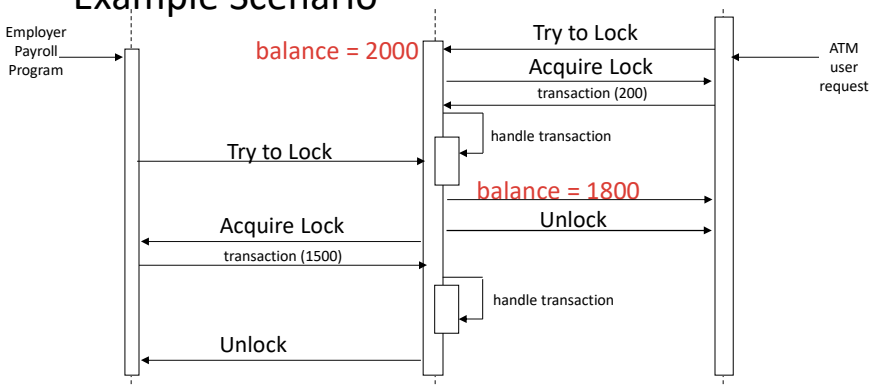
26

Concurrency Control in Sequence Diagram

27

Pessimistic Concurrency Control

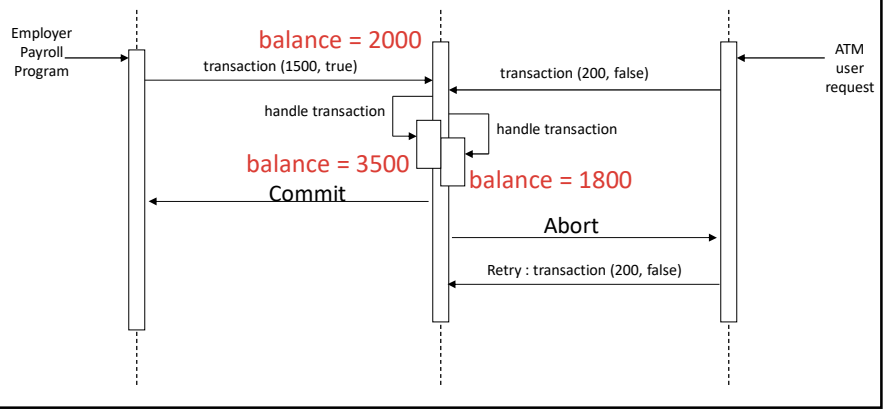
- Try to Lock (or Lock) and Unlock
- Example Scenario



28

Optimistic Concurrency Control

- Commit and Abort
- Example Scenario



29

Sample of Specification

- If the ATM is running out of money or no card should be accepted, An error message is displayed
- The ATM has to check if the entered card is a valid cash card
- If the cash card is valid the ATM should read the serial number and bank code
- If password and serial number are ok, the authorization process is finished.
- Select an action
- Perform a transaction

etc

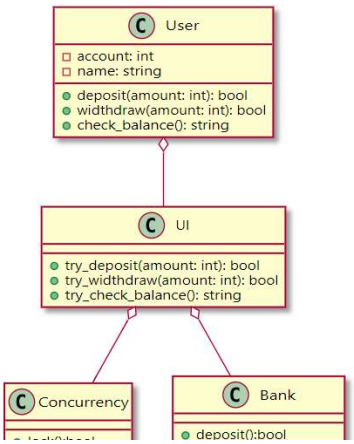
30

ATM in PlantUML

31

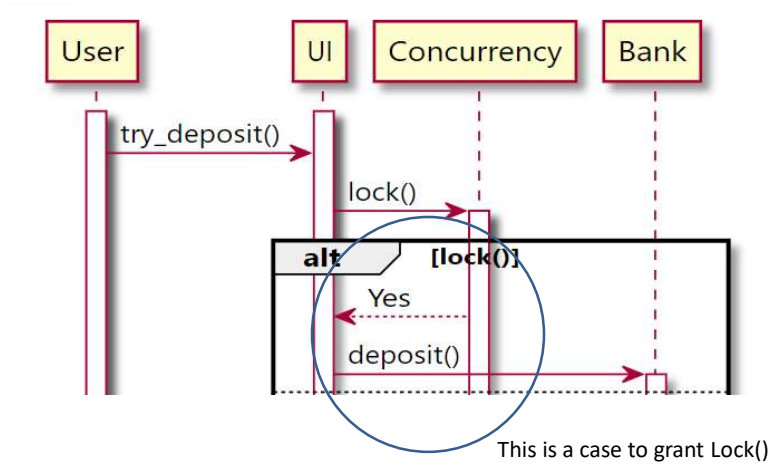
Class Diagram

- Main Classes:
  - User, UI, Concurrency, Bank
- Optional Classes:
  - Authentication, Account, Banker



32

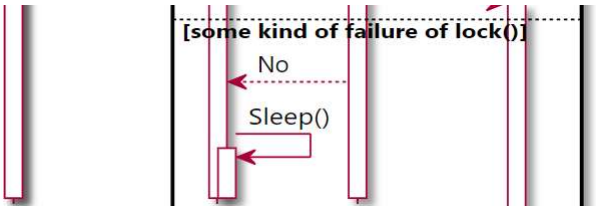
Sequence Diagram



This is a part of project 4

33

Lock vs Unlock



Once you deposit successfully, you must unlock.

34

Design three cases – deposit(), withdraw() and check_balance()

35

Design Guideline in PlantUML

- All operations for each class must participate in a sequence diagram.
- At least two operations must be defined in each class.
- Class and sequence diagrams must be feasible.

36

Group Project Option

- Up to two students
- A student
  - More than 4 classes
- Two students
  - More than 7 classes
  - Put you and your partner's names to a DOC/PDF and both students must submit the same DOC/PDF.

37