Last time
Last time

Distributed systems
Last time

Distributed systems
  • Architectures
Last time

Distributed systems

- Architectures
  - Client-server
Last time

Distributed systems

- Architectures
  - Client-server
  - Peer-to-peer
Last time

Distributed systems

- Architectures
  - Client-server
  - Peer-to-peer
- Message passing
Last time

Distributed systems

- Architectures
  - Client-server
  - Peer-to-peer

- Message passing
  - Protocols
Last time

Distributed systems

- Architectures
  - Client-server
  - Peer-to-peer
- Message passing
  - Protocols

System design principles
Last time

Distributed systems
  • Architectures
    • Client-server
    • Peer-to-peer
  • Message passing
    • Protocols

System design principles
  • Modularity
Last time

Distributed systems

- Architectures
  - Client-server
  - Peer-to-peer
- Message passing
  - Protocols

System design principles

- Modularity
- Interfaces
Today: Parallel Computation
Today: Parallel Computation

Why is parallel computation important?
Today: Parallel Computation

Why is parallel computation important?

What is parallel computation?
Today: Parallel Computation

Why is parallel computation important?

What is parallel computation?

Some examples in Python
Today: Parallel Computation

Why is parallel computation important?

What is parallel computation?

Some examples in Python

Some problems with parallel computation
Transistors
Transistors

Computers execute instructions by manipulating the flow of electricity through transistors.
Transistors

Computers execute instructions by manipulating the flow of electricity through transistors.

Transistors are made from semiconductors, like silicon.
Transistors

Computers execute instructions by manipulating the flow of electricity through transistors.

Transistors are made from semiconductors, like silicon.

More transistors = more power.
Transistors

Computers execute instructions by manipulating the flow of electricity through transistors.

Transistors are made from semiconductors, like silicon.

More transistors = more power.

Transistors are now less than 100 nanometers in size.
Transistors

Computers execute instructions by manipulating the flow of electricity through **transistors**.

Transistors are made from semiconductors, like silicon.

More transistors = more power.

Transistors are now less than 100 nanometers in size.
Transistors

Computers execute instructions by manipulating the flow of electricity through **transistors**.

Transistors are made from semiconductors, like silicon.

More transistors = more power.

Transistors are now less than 100 nanometers in size.

Microprocessor
Transistors

Computers execute instructions by manipulating the flow of electricity through transistors.

Transistors are made from semiconductors, like silicon.

More transistors = more power.

Transistors are now less than 100 nanometers in size.

Microprocessor

Transistors are arranged into “integrated circuits” on single pieces of hardware.
Transistors

Computers execute instructions by manipulating the flow of electricity through transistors.

Transistors are made from semiconductors, like silicon.

More transistors = more power.

Transistors are now less than 100 nanometers in size.

Microprocessor

Transistors are arranged into “integrated circuits” on single pieces of hardware.

A microprocessor, or processor is a large integrated circuit of transistors where a computer’s instructions are executed.
Microprocessors
Microprocessors

1971

Intel 4000
2300 Transistors
Microprocessors

1971

Intel 4000
2300 Transistors

1981

National Semiconductor NS3008
~10,000 Transistors
Microprocessors

1971

Intel 4000
2300 Transistors

1981

National Semiconductor NS3008
~10,000 Transistors

1993

Intel Pentium
~3 million transistors
Microprocessors

1971
Intel 4000
2300 Transistors

1981
National Semiconductor NS3008
~10,000 Transistors

1993
Intel Pentium
~3 million transistors

2000’s
AMD 64
~243 million transistors
Moore’s law
Moore’s law

In 1965, the co-founder of Intel, Gordon Moore predicted that the number of transistors that could be fit onto a single chip would double every year.
Moore’s law

In 1965, the co-founder of Intel, Gordon Moore predicted that the number of transistors that could be fit onto a single chip would double every year.

46 years later, that prediction is still true.
More transistors every year

Microprocessor Transistor Counts 1971-2011 & Moore’s Law

The graph shows the increasing number of transistors in microprocessors from 1971 to 2011, with a trend line indicating that the transistor count doubles every two years. The data points correspond to different microprocessor models introduced over the years.

Date of introduction:

1971
1980
1990
2000
2011

Transistor count:

2,300
10,000
100,000
1,000,000
10,000,000
100,000,000
1,000,000,000
2,600,000,000
Physical limits

Instead of trying to fit more transistors into a single processor, we are turning to multiple processors.
Physical limits

Manufacturers are reaching physical limits

Instead of trying to fit more transistors into a single processor, we are turning to multiple processors.
Physical limits

Manufacturers are reaching physical limits
  - Transistors size limits

Instead of trying to fit more transistors into a single processor, we are turning to multiple processors.
Physical limits

Manufacturers are reaching physical limits
  - Transistors size limits
  - Instructions speed limits

Instead of trying to fit more transistors into a single processor, we are turning to multiple processors.
Physical limits

Manufacturers are reaching physical limits

- Transistors size limits
- Instructions speed limits

The solution: multiple microprocessors

Instead of trying to fit more transistors into a single processor, we are turning to multiple processors.
Parallel Computation
Parallel Computation

A program (a set of instructions, a piece of code)
Parallel Computation

A program (a set of instructions, a piece of code)

Executed simultaneously by multiple processors
Parallel Computation

A program (a set of instructions, a piece of code)

Executed simultaneously by multiple processors

In a shared memory environment
Parallel computing example

\[ x = 5 \]
\[ x = \text{square}(x) \]
\[ y = 6 \]
\[ y = y + 1 \]
Parallel computing example

x = 5
x = square(x)
y = 6
y = y+1
write 5 -> x
Parallel computing example

\[
\begin{align*}
x &= 5 \\
x &= \text{square}(x) \\
y &= 6 \\
y &= y+1 \\
\text{write} &\ 5 \rightarrow x \\
\text{read} &\ x: 5
\end{align*}
\]
Parallel computing example

\[
\begin{align*}
x & = 5 \\
x & = \text{square}(x) \\
y & = 6 \\
y & = y+1
\end{align*}
\]

write 5 \rightarrow x
read x: 5
calculate 5*5: 25
Parallel computing example

\[
x = 5 \\
x = \text{square}(x) \\
y = 6 \\
y = y + 1 \\
\]

write $5 \rightarrow x$

read $x: 5$

calculate $5*5: 25$

write $25 \rightarrow x$
Parallel computing example

\[ x = 5 \]
\[ x = \text{square}(x) \]
\[ y = 6 \]
\[ y = y + 1 \]
\[ \text{write } 5 \rightarrow x \]
\[ \text{read } x: \quad 5 \]
\[ \text{calculate } 5 \times 5: \quad 25 \]
\[ \text{write } 25 \rightarrow x \]
\[ \text{write } 6 \rightarrow y \]
Parallel computing example

\[
\begin{align*}
x &= 5 \\
x &= \text{square}(x) \\
y &= 6 \\
y &= y+1 \\
\text{write} & \quad 5 \rightarrow x \\
\text{read} & \quad x: \ 5 \\
\text{calculate} & \quad 5 \times 5: \ 25 \\
\text{write} & \quad 25 \rightarrow x \\
\text{write} & \quad 6 \rightarrow y \\
\text{read} & \quad y: \ 6
\end{align*}
\]
Parallel computing example

\[
x = 5
\]
\[
x = \text{square}(x)
\]
\[
y = 6
\]
\[
y = y+1
\]

write 5 -> x
read x: 5
calculate 5*5: 25
write 25 -> x
write 6 -> y
read y: 6
calculate 6+1: 7
Parallel computing example

\[
x = 5 \\
x = \text{square}(x) \\
y = 6 \\
y = y+1 \\
\text{write } 5 \rightarrow x \\
\text{read } x: \ 5 \\
\text{calculate } 5*5: \ 25 \\
\text{write } 25 \rightarrow x \\
\text{write } 6 \rightarrow y \\
\text{read } y: \ 6 \\
\text{calculate } 6+1: \ 7 \\
\text{write } y \rightarrow 7
\]
Parallel computing example

\[ x = 5 \]
\[ x = \text{square}(x) \]
\[ y = 6 \]
\[ y = y + 1 \]

read \( x \): 5
\text{calculate} 5 \times 5: 25
write 25 -&gt; x

read \( y \): 6
\text{calculate} 6 + 1: 7
write \( y \) -&gt; 7
Parallel computing example

\[
\begin{align*}
x &= 5 \\
x &= \text{square}(x) \\
y &= 6 \\
y &= y+1
\end{align*}
\]
Parallel computing example

\[ x = 5 \]
\[ x = \text{square}(x) \]

\[ y = 6 \]
\[ y = y + 1 \]
Parallel computing example

\[
x = 5 \\
x = \text{square}(x)
\]

\[
y = 6 \\
y = y + 1
\]

\[\text{P1} \]
write 5 \(\rightarrow\) x

\[\text{P2} \]
write 6 \(\rightarrow\) y
Parallel computing example

\[
x = 5 \\
x = \text{square}(x)
\]

\[
y = 6 \\
y = y+1
\]

P1
write 5 -> x
read x: 5

P2
write 6 -> y
read y: 6
Parallel computing example

\[
\begin{align*}
  x &= 5 \\
  x &= \text{square}(x) \\
  y &= 6 \\
  y &= y+1
\end{align*}
\]

**P1**
write 5 -> x
read x: 5
calculate 5*5: 25

**P2**
write 6 -> y
read y: 6
calculate 6+1: 7
Parallel computing example

\[
\begin{align*}
\text{P1} & \quad \text{write 5} \rightarrow x \\
& \quad \text{read x: 5} \\
& \quad \text{calculate 5*5: 25} \\
& \quad \text{write 25} \rightarrow x
\end{align*}
\]

\[
\begin{align*}
\text{P2} & \quad \text{write 6} \rightarrow y \\
& \quad \text{read y: 6} \\
& \quad \text{calculate 6+1: 7} \\
& \quad \text{write 7} \rightarrow y
\end{align*}
\]

\[
\begin{align*}
x &= 5 \\
x &= \text{square}(x) \\
y &= 6 \\
y &= y+1
\end{align*}
\]
Parallel computing example

\[
\begin{align*}
x & = 5 \\
x & = \text{square}(x)
\end{align*}
\]

\[
\begin{align*}
y & = 6 \\
y & = y+1
\end{align*}
\]

**P1**
- write 5 -> x
- read x: 5
- calculate 5*5: 25
- write 25 -> x

**P2**
- write 6 -> y
- read y: 6
- calculate 6+1: 7
- write 7 -> y

\[
\begin{align*}
x & = 25 \\
y & = 7
\end{align*}
\]
Shared memory
Shared memory

\[ x = 5 \]
Shared memory

\[ x = 5 \]

\[ x = \text{square}(x) \quad y = x + 1 \]
Shared memory

\[
\begin{align*}
x & = 5 \\
x & = \text{square}(x) \\
y & = x + 1
\end{align*}
\]
Shared memory

\[ x = 5 \]

\[ x = \text{square}(x) \]  \[ y = x + 1 \]

\underline{P1}  \quad \text{read } x: \ 5  \quad \underline{P2}
Shared memory

\[ x = 5 \]

\[ x = \text{square}(x) \]  \hspace{1cm}  \[ y = x + 1 \]

\underline{P1}
read x: 5
calculate 5*5: 25

\underline{P2}
read x: 5
Shared memory

\[
x = 5
\]

\[
x = \text{square}(x)
\]

\[
y = x + 1
\]

**P1**
read \( x \): 5
calculate \( 5 \times 5 \): 25
write 25 -> x

**P2**
read \( x \): 5
calculate \( 5 + 1 \): 6
Shared memory

\[ x = 5 \]

\[ x = \text{square}(x) \]

\[ y = x + 1 \]

\begin{align*}
\text{P1} & \\
\text{read } x & : 5 \\
\text{calculate } 5 \times 5 & : 25 \\
\text{write } 25 & \rightarrow x
\end{align*}

\begin{align*}
\text{P2} & \\
\text{read } x & : 5 \\
\text{calculate } 5 + 1 & : 6 \\
\text{write } 6 & \rightarrow y
\end{align*}
Shared memory

\[ x = 5 \]

\[ x = \text{square}(x) \]

\[ y = x + 1 \]

**P1**

read x: 5

calculate 5*5: 25

write 25 -> x

**P2**

read x: 5

calculate 5+1: 6

write 6 -> y

\[ x = 25 \]

\[ y = 6 \]
How many different values of $x$ and $y$ can there be?

Quiz:

How many different values of $x$ and $y$ can there be at the end?
Shared memory
Shared memory

\[ x = 5 \]
Shared memory

\[ x = 5 \]

\[ x = \text{square}(x) \quad \text{and} \quad x = x + 1 \]
Shared memory

\[
x = 5
\]

\[
x = \text{square}(x)
\]

\[
x = x + 1
\]
Shared memory

x = 5

\[ x = \text{square}(x) \]

P1

\[ x = x + 1 \]

P2
Shared memory

\[ x = 5 \]

\[ x = \text{square}(x) \]

\[ x = x + 1 \]

P1
read \( x \): 5

P2
Shared memory

\[ x = 5 \]

\[ x = \text{square}(x) \]

\[ x = x + 1 \]

P1
read x: 5
calculate 5*5: 25

P2
read x: 5
Shared memory

\[
x = 5
\]

\[
\text{P1}
\]
read \(x\): 5
calculate \(5 \times 5\): 25
write 25 -> \(x\)

\[
\text{P2}
\]
read \(x\): 5
calculate \(5 + 1\): 6
Shared memory

\[ x = 5 \]

\[ \text{P1} \quad \text{read } x: 5 \]
\[ \text{calculate } 5*5: 25 \]
\[ \text{write } 25 \rightarrow x \]

\[ \text{P2} \quad \text{read } x: 5 \]
\[ \text{calculate } 5+1: 6 \]
\[ \text{write } 6 \rightarrow x \]
Shared memory

\[
x = 5
\]

\[
x = \text{square}(x)
\]

\[
x = x + 1
\]

P1
read \( x \): 5
calculate \( 5 \times 5 \): 25
write 25 -> \( x \)

P2
read \( x \): 5
calculate \( 5 + 1 \): 6
write 6 -> \( x \)

\[
x = 6
\]
How many different values of $x$ can there be?

Quiz:

How many different values of $x$ can there be at the end?
Shared memory

\[ x = 5 \]

\[ x = \text{square}(x) \quad x = x + 1 \]
Shared memory

\[ x = 5 \]

\[ x = \text{square}(x) \]  \hspace{1cm}  \[ x = x + 1 \]

P1  \hspace{1cm}  P2
Shared memory

\[ x = 5 \]

\[ x = \text{square}(x) \]

\[ x = x + 1 \]

P1

P2

read \( x: 5 \)
Shared memory

\[ x = 5 \]

\[ \text{P1} \]

\[ x = \text{square}(x) \]

read \( x \): 5

\[ \text{P2} \]

\[ x = x + 1 \]

read \( x \): 5
Shared memory

$$x = 5$$

| $$x = \text{square}(x)$$ | $$x = x + 1$$ |

P1

read x: 5

calculate $5 \times 5$: 25

P2

read x: 5

calculate $5 + 1$: 6
Shared memory

\[ x = 5 \]

\[ \boxed{x = \text{square}(x)} \]

**P1**

read x: 5

calculate 5*5: 25

\[ x = x + 1 \]

**P2**

read x: 5

calculate 5+1: 6

write 6 -> x
Shared memory

\[ x = 5 \]

\[
\begin{align*}
\text{P1} & \quad x & = & \text{square}(x) \\
\text{read } x: \ 5 & \quad \text{calculate } 5 \cdot 5: \ 25 \\
& \quad \text{write } 25 \rightarrow x
\end{align*}
\]

\[
\begin{align*}
\text{P2} & \quad x & = & x + 1 \\
\text{read } x: \ 5 & \quad \text{calculate } 5 + 1: \ 6 \\
& \quad \text{write } 6 \rightarrow x
\end{align*}
\]
**Shared memory**

\[
x = 5
\]

\[
x = \text{square}(x)
\]

**P1**

- Read \(x\): 5
- Calculate \(5 \times 5\): 25
- Write 25 -> \(x\)

\[
x = x + 1
\]

**P2**

- Read \(x\): 5
- Calculate \(5 + 1\): 6
- Write 6 -> \(x\)

\[
x = 25
\]
Parallel computing example: bank balance
Parallel computing example: bank balance

def make_withdraw(balance):
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        ...
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        global balance
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:

 Tuesday, November 15, 2011
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            # code to withdraw funds

Tuesday, November 15, 2011
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount

Parallel computing example: bank balance
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
        print(balance)
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
        print(balance)
    return withdraw
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
    return withdraw

w = make_withdraw(10)
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
        print(balance)
    return withdraw

w = make_withdraw(10)

w(8)  

w(7)
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
    return withdraw
```
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
    return withdraw

w = make_withdraw(10)
balance = 10
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
    return withdraw

w = make_withdraw(10)
balance = 10

w(8) w(7)
```
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
        print(balance)
    return withdraw

w = make_withdraw(10)
balance = 10
w(8)
w(7)
```

2 or 3
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
        return withdraw

w = make_withdraw(10)
balance = 10
w(8)
2 or 3
w(7)

print('Insufficient funds')
```

19 Tuesday, November 15, 2011
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
        print(balance)
    return withdraw
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
    return withdraw

w = make_withdraw(10)
balance = 10
```

```python
w(5)
balance
```
**Parallel computing example: bank balance**

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)

    return withdraw

w = make_withdraw(10)
balance = 10

w(8) w(7)
```

```
w(8)
```

Tuesday, November 15, 2011
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
    return withdraw

w = make_withdraw(10)
balance = 10

w(8) | w(7)

read global balance: 10
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
        return withdraw

w = make_withdraw(10)
balance = 10

w(8)
read global balance: 10
read amount: 8

w(7)
read global balance: 10
```
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
    return withdraw

w = make_withdraw(10)
balance = 10

w(8)
read global balance: 10
read amount: 8
8 > 10: False

w(7)
read global balance: 10
read amount: 7
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
    return withdraw

w = make_withdraw(10)
balance = 10

w(8)
read global balance: 10
read amount: 8
8 > 10: False

w(7)
read global balance: 10
read amount: 7
7 > 10: False
```
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
        return withdraw

w = make_withdraw(10)
balance = 10

w(8)
read global balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2

w(7)
read global balance: 10
read amount: 7
7 > 10: False
if False
```

*Tuesday, November 15, 2011*
Parallel computing example: bank balance

def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
        return withdraw

w = make_withdraw(10)
balance = 10

w(8)
read global balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2

w(7)
read global balance: 10
read amount: 7
7 > 10: False
if False
10 - 7: 3
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
        return withdraw

w = make_withdraw(10)
balance = 10

w(8)
read global balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2

w(7)
read global balance: 10
read amount: 7
7 > 10: False
if False
10 - 7: 3
```
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
        print(balance)
        return withdraw

w = make_withdraw(10)
balance = 10
w(8)
w(7)

read global balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
print 2

read global balance: 10
read amount: 7
7 > 10: False
if False
10 - 7: 3
write balance -> 3
```
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
        return withdraw

w = make_withdraw(10)
balance = 10
w(8)

w(7)
```

read global balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
print 2

read global balance: 10
read amount: 7
7 > 10: False
if False
10 - 7: 3
write balance -> 3
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        global balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
        print(balance)
    return withdraw

w = make_withdraw(10)
balance = 10
w(8)
```

**w(8)**

- read global balance: 10
- read amount: 8
- 8 > 10: False
- if False
- 10 - 8: 2
- write balance -> 2
- print 2

**w(7)**

- read global balance: 10
- read amount: 7
- 7 > 10: False
- if False
- 10 - 7: 3
- write balance -> 3
- print 3
Next time: how to fix these problems

Locks, semaphores, conditions