61A Lecture 32

November 16th, 2011

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

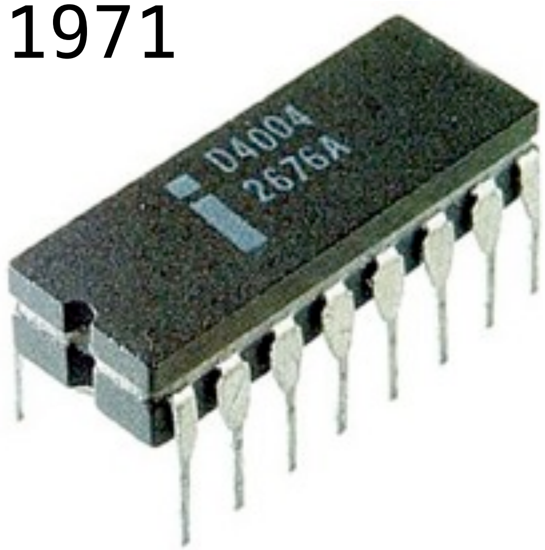
Transistors are arranged into “integrated circuits” on single pieces of hardware.

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

Microprocessors

Microprocessors

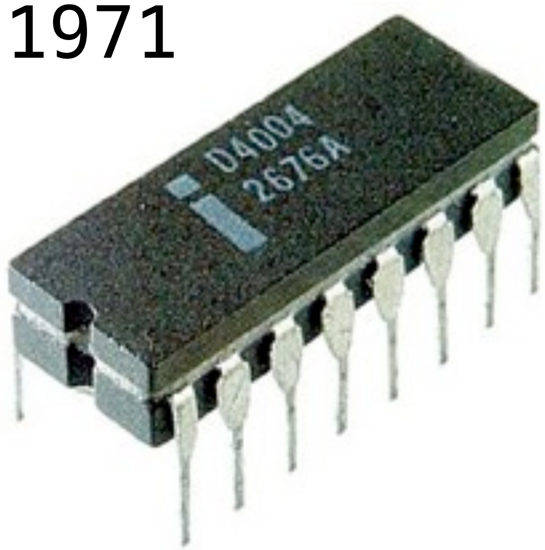
1971



Intel 4000
2300 Transistors

Microprocessors

1971



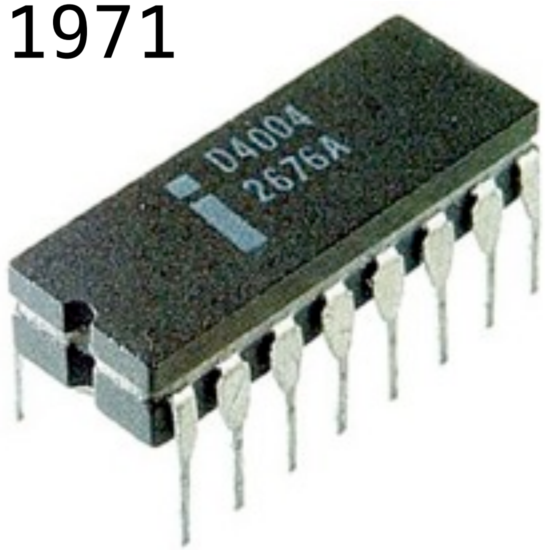
Intel 4000
2300 Transistors

1981

National Semiconductor NS3008
~10,00 Transistors

Microprocessors

1971



Intel 4000
2300 Transistors

1981

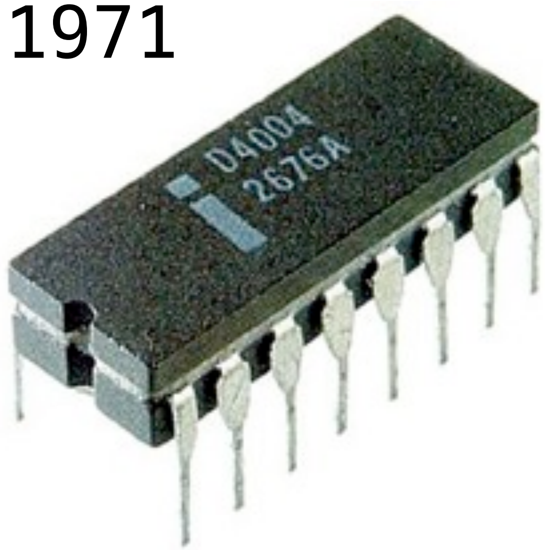
National Semiconductor NS3008
~10,00 Transistors

1993

Intel Pentium
~3 million transistors

Microprocessors

1971



Intel 4000
2300 Transistors

1981

National Semiconductor NS3008
~10,00 Transistors

1993

Intel Pentium
~3 million transistors

2000's

AMD 64
~243 million transistors

Moore's law

Moore's law

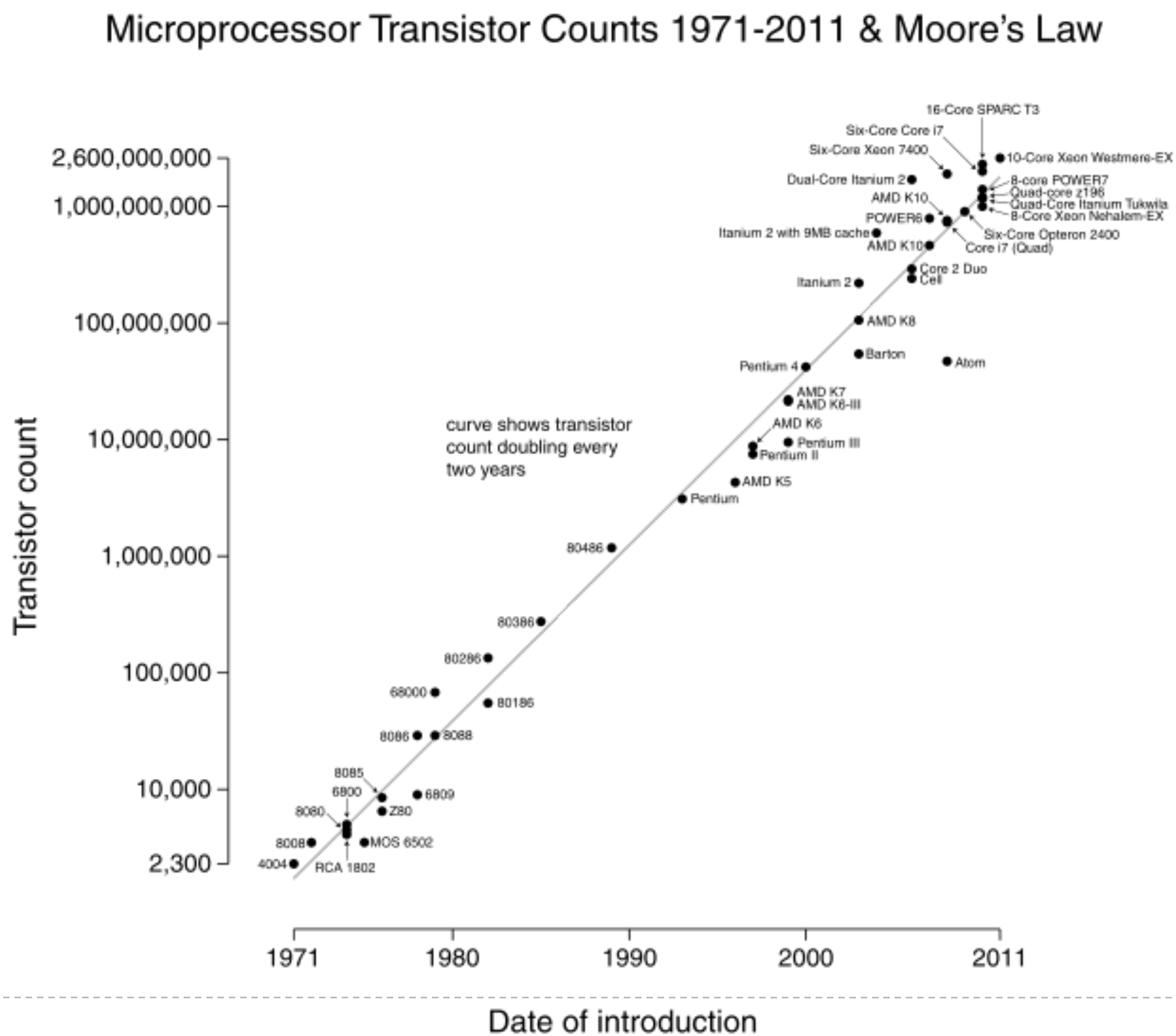
In 1965, the co-founder of Intel, Gordon Moore predicted that the number of transistors that could be fit onto a single chip would double every year.

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



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 = 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

```
x = 5
x = square(x)
y = 6
y = y+1

write 5 -> x
read x: 5
```

Parallel computing example

```
x = 5
x = square(x)
y = 6
y = y+1
```

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

Parallel computing example

```
x = 5
x = square(x)
y = 6
y = y+1
```

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

Parallel computing example

```
x = 5
x = square(x)
y = 6
y = y+1
```

```
write 5 -> x
read x: 5
calculate 5*5: 25
write 25 -> x
write 6 -> y
```

Parallel computing example

```
x = 5
x = 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
```

Parallel computing example

```
x = 5
x = 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 = 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
write y-> 7
```

Parallel computing example

```
x = 5
x = square(x)
y = 6
y = y+1

read x: 5
calculate 5*5: 25
write 25 -> x
read y: 6
calculate 6+1: 7
write y-> 7
```

Parallel computing example

```
x = 5
x = square(x)
```

```
y = 6
y = y+1
```

Parallel computing example

```
x = 5
x = square(x)
```

```
y = 6
y = y+1
```

P1

P2

Parallel computing example

```
x = 5
x = square(x)
```

```
y = 6
y = y+1
```

P1

```
write 5 -> x
```

P2

```
write 6 -> y
```

Parallel computing example

```
x = 5
x = square(x)
```

```
y = 6
y = y+1
```

<u>P1</u>

```
write 5 -> x
read x: 5
```

<u>P2</u>

```
write 6 -> y
read y: 6
```

Parallel computing example

```
x = 5
x = square(x)
```

```
y = 6
y = y+1
```

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

```
x = 5
x = square(x)
```

```
y = 6
y = y+1
```

```
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
```

Parallel computing example

```
x = 5
x = square(x)
```

```
y = 6
y = y+1
```

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
```

```
x = 25
y = 7
```

Shared memory

Shared memory

x = 5

Shared memory

```
x = 5
```

```
x = square(x)
```

```
y = x + 1
```

Shared memory

```
x = 5
```

```
x = square(x)
```

```
y = x + 1
```

P1

P2

Shared memory

```
x = 5
```

```
x = square(x)
```

```
y = x + 1
```

P1

```
read x: 5
```

P2

Shared memory

```
x = 5
```

```
x = square(x)
```

```
y = x + 1
```

P1

```
read x: 5
calculate 5*5: 25
```

P2

```
read x: 5
```

Shared memory

```
x = 5
```

```
x = square(x)
```

```
y = x + 1
```

<u>P1</u>

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

<u>P2</u>

```
read x: 5
calculate 5+1: 6
```

Shared memory

```
x = 5
```

```
x = 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
```

Shared memory

```
x = 5
```

```
x = 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 = square(x)
```

```
x = x + 1
```

Shared memory

```
x = 5
```

```
x = square(x)
```

```
x = x + 1
```

Shared memory

```
x = 5
```

```
x = square(x)
```

```
x = x + 1
```

P1

P2

Shared memory

```
x = 5
```

`x = square(x)`	`x = x + 1`
P1	**P2**
`read x: 5`	

Shared memory

```
x = 5
```

```
x = square(x)
```

```
x = x + 1
```

P1

```
read x: 5
calculate 5*5: 25
```

P2

```
read x: 5
```

Shared memory

```
x = 5
```

```
x = square(x)
```

```
x = x + 1
```

P1

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

P2

```
read x: 5
calculate 5+1: 6
```

Shared memory

```
x = 5
```

```
x = square(x)
```

```
x = x + 1
```

P1

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

P2

```
read x: 5
calculate 5+1: 6
write 6 -> x
```

Shared memory

```
x = 5
```

```
x = square(x)

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

```
x = x + 1

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 = square(x)
```

```
x = x + 1
```

Shared memory

```
x = 5
```

```
x = square(x)
```

<u>P1</u>

```
x = x + 1
```

<u>P2</u>

Shared memory

```
x = 5
```

```
x = square(x)
```

P1

```
x = x + 1
```

P2

```
read x: 5
```

Shared memory

```
x = 5
```

```
x = square(x)
```

P1

```
read x: 5
```

```
x = x + 1
```

P2

```
read x: 5
```

Shared memory

```
x = 5
```

```
x = square(x)
```

P1

```
read x: 5
calculate 5*5: 25
```

```
x = x + 1
```

P2

```
read x: 5

calculate 5+1: 6
```

Shared memory

```
x = 5
```

```
x = square(x)
```

```
x = x + 1
```

P1

```
read x: 5
calculate 5*5: 25
```

P2

```
read x: 5

calculate 5+1: 6
write 6 -> x
```

Shared memory

```
x = 5
```

```
x = square(x)
```

P1

```
read x: 5
calculate 5*5: 25

write 25 -> x
```

```
x = x + 1
```

P2

```
read x: 5

calculate 5+1: 6
write 6 -> x
```

Shared memory

```
x = 5
```

```
x = square(x)
```

```
x = x + 1
```

P1

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

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:
```

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:
```

Parallel computing example: bank balance

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

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)
```

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)
```

w(8)	w(7)

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)
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)
```

```
w(7)
```

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 2 or 3
```

```
w(8)
```

```
w(7)
```

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 2 or 3
```

```
w(8)
```

```
w(7)
```

```
print('Insufficient funds')
```

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)
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)
```

```
w(7)
```

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

```
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`	`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

```
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
```

```
w(7)

read global balance: 10
read amount: 7
7 > 10: False
```

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	
read amount: 8	read global balance: 10
8 > 10: False	read amount: 7
if False	7 > 10: False
10 - 8: 2	if False

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

```
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~~ 2
```

w(8)	w(7)
read global balance: 10	
read amount: 8	read global balance: 10
8 > 10: False	read amount: 7
if False	7 > 10: False
10 - 8: 2	if False
write balance -> 2	10 - 7: 3

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 2
```

```
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
```

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 2 3
```

```
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
```

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 2 3
```

w(8)	w(7)
read global balance: 10	
read amount: 8	read global balance: 10
8 > 10: False	read amount: 7
if False	7 > 10: False
10 - 8: 2	if False
write balance -> 2	10 - 7: 3
print 2	write balance -> 3
	print 3

Next time: how to fix these problems

Locks, semaphores, conditions