Last time
Last time

Why is parallel computation important?
Last time

Why is parallel computation important?

What is parallel computation?
Last time

Why is parallel computation important?

What is parallel computation?

Some examples in Python
Last time

Why is parallel computation important?

What is parallel computation?

Some examples in Python

Some problems with parallel computation
Parallel computation terminology
Parallel computation terminology

Processor
Parallel computation terminology

Processor

- One of (possibly) many pieces of hardware responsible for executing instructions
Parallel computation terminology

Processor
  ▪ One of (possibly) many pieces of hardware responsible for executing instructions

Thread
Parallel computation terminology

Processor
- One of (possibly) many pieces of hardware responsible for executing instructions

Thread
- One of (possibly) many simultaneous sequences of instructions, being executed in a shared memory environment
Parallel computation terminology

Processor
- One of (possibly) many pieces of hardware responsible for executing instructions

Thread
- One of (possibly) many simultaneous sequences of instructions, being executed in a shared memory environment

Shared memory
Parallel computation terminology

Processor

- One of (possibly) many pieces of hardware responsible for executing instructions

Thread

- One of (possibly) many simultaneous sequences of instructions, being executed in a shared memory environment

Shared memory

- The environment in which threads are executed, containing variables that are accessible to all the threads.
Today: dealing with shared memory
Today: dealing with shared memory

“Vulnerable sections” of a program
Today: dealing with shared memory

“Vulnerable sections” of a program
  - Critical Sections
Today: dealing with shared memory

“Vulnerable sections” of a program

- Critical Sections
- Atomicity
Today: dealing with shared memory

“Vulnerable sections” of a program
- Critical Sections
- Atomicity

Correctness
Today: dealing with shared memory

“Vulnerable sections” of a program
  ▪ Critical Sections
  ▪ Atomicity

Correctness
  ▪ What does “correctness” mean for parallel computation?
Today: dealing with shared memory

“Vulnerable sections” of a program
  ▪ Critical Sections
  ▪ Atomicity

Correctness
  ▪ What does “correctness” mean for parallel computation?

Protecting vulnerable sections
Today: dealing with shared memory

“Vulnerable sections” of a program
- Critical Sections
- Atomicity

Correctness
- What does “correctness” mean for parallel computation?

Protecting vulnerable sections
- Locks
Today: dealing with shared memory

“Vulnerable sections” of a program
- Critical Sections
- Atomicity

Correctness
- What does “correctness” mean for parallel computation?

Protecting vulnerable sections
- Locks
- Semaphores
Today: dealing with shared memory

“Vulnerable sections” of a program
- Critical Sections
- Atomicity

Correctness
- What does “correctness” mean for parallel computation?

Protecting vulnerable sections
- Locks
- Semaphores
- Conditions
Today: dealing with shared memory

“Vulnerable sections” of a program
  - Critical Sections
  - Atomicity

Correctness
  - What does “correctness” mean for parallel computation?

Protecting vulnerable sections
  - Locks
  - Semaphores
  - Conditions

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

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

Friday, November 18, 2011
Parallel computing example: bank balance

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

w = make_withdraw(10)
balance = 10

w(8)  # prints 'Insufficient funds'
w(7)  # prints 2
```
Parallel computing example: bank balance

```python
def make_withdraw(balance):
    def withdraw(amount):
        nonlocal 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

def make_withdraw(balance):
    def withdraw(amount):
        nonlocal 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)

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

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

Friday, November 18, 2011
def make_withdraw(balance):
    def withdraw(amount):
        nonlocal 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)
def make_withdraw(balance):
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
    return withdraw

w = make_withdraw(10)
balance = 10

w(8)
read balance: 10

w(7)
def make_withdraw(balance):
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
        return withdraw

w = make_withdraw(10)
balance = 10

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

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

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

w = make_withdraw(10)
balance = 10

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

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

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

w = make_withdraw(10)
balance = 10

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

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

Friday, November 18, 2011
Parallel computing example: bank balance

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

w = make_withdraw(10)
balance = 10

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

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

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

w = make_withdraw(10)
balance = 10

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

w(7)
read 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):
        nonlocal balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
    return withdraw

w = make_withdraw(10)
balance = 10
w(8)
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
w(7)
read balance: 10
read amount: 7
7 > 10: False
if False
10 - 7: 3
```
def make_withdraw(balance):
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
        return withdraw

w = make_withdraw(10)
balance = 10

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

w(7)
read 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):
        nonlocal balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
        print(balance)
    return withdraw

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

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

w(7)

read 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):
        nonlocal balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
        print(balance)
    return withdraw

w = make_withdraw(10)
balance = 10

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

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

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

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

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

w(7)

read balance: 10
read amount: 7
7 > 10: False
if False
10 - 7: 3
write balance -> 3
print 3

$15 withdrawn from a $10 account?
With $3 left? Inconceivable!
```
Parallel computing example: bank balance

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

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

```javascript
w(8)  w(7)  2 or 3
```

print('Insufficient funds')
Another problem: vector mathematics

\[ A = B + C \]
\[ V = M \times A \]
Vector mathematics

$$B = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 \\ 5 \end{pmatrix} \quad M = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}$$
Vector mathematics

\[ A = B + C \]
\[ V = M \times A \]

\[ B = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 \\ 5 \end{pmatrix} \quad M = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \]
Vector mathematics

\[ A = B + C \]
\[ V = M \times A \]

\[
\begin{bmatrix}
2 \\
5 \\
12
\end{bmatrix} \quad \begin{bmatrix}
12
\end{bmatrix} \quad \begin{bmatrix}
2 \\
0 \\
5
\end{bmatrix} \quad \begin{bmatrix}
0 \\
5 \\
12
\end{bmatrix} \quad \begin{bmatrix}
1 & 2
\end{bmatrix}
\]
**Vector mathematics**

\[
\begin{align*}
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix} \\
V &= \begin{pmatrix} 12 \\ 12 \end{pmatrix} \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A \\
A_2 &= B_2 + C_2 \\
V_2 &= M_2 \cdot A
\end{align*}
\]
Vector mathematics

$$A = B + C$$
$$V = M \times A$$

\[
A = \begin{pmatrix} 2 \\ 5 \end{pmatrix}, \quad V = \begin{pmatrix} 12 \\ 12 \end{pmatrix}, \quad B = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \quad C = \begin{pmatrix} 0 \\ 5 \end{pmatrix}, \quad M = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}
\]

\[
P_1: \quad A_1 = B_1 + C_1, \quad V_1 = M_1 \cdot A
\]

\[
P_2: \quad A_2 = B_2 + C_2, \quad V_2 = M_2 \cdot A
\]
Vector mathematics

\[ A = B + C \]
\[ V = M \times A \]

\[
A = \begin{pmatrix} 2 \\ 5 \\ 12 \end{pmatrix} \quad V = \begin{pmatrix} 12 \\ 12 \end{pmatrix} \quad B = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 \\ 5 \end{pmatrix} \quad M = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}
\]

\[
A_1 = B_1 + C_1 \\
V_1 = M_1 \cdot A
\]

\[
A_2 = B_2 + C_2 \\
V_2 = M_2 \cdot A
\]

\[ P1 \quad \text{read } B1: 2 \]

\[ P2 \]
**Vector mathematics**

$$A = \begin{pmatrix} 2 \\ 5 \end{pmatrix} \quad V = \begin{pmatrix} 12 \\ 12 \end{pmatrix}$$

$$B = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 \\ 5 \end{pmatrix} \quad M = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}$$

\[A_1 = B_1 + C_1\]
\[V_1 = M_1 \cdot A\]

**P1**
read \(B_1\): 2
read \(C_1\): 0

\[A_2 = B_2 + C_2\]
\[V_2 = M_2 \cdot A\]

**P2**
Vector mathematics

\[ A = \begin{pmatrix} 2 \\ 5 \end{pmatrix}, \quad V = \begin{pmatrix} 12 \\ 12 \end{pmatrix}, \quad B = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \quad C = \begin{pmatrix} 0 \\ 5 \end{pmatrix}, \quad M = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \]

\[ A_1 = B_1 + C_1, \quad V_1 = M_1.A \]

P1
read B1: 2
read C1: 0
calculate 2+0: 2

P2

A_2 = B_2 + C_2
V_2 = M_2.A
Vector mathematics

\[
\begin{align*}
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix} \quad V = \begin{pmatrix} 12 \\ 12 \end{pmatrix} \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1.A
\end{align*}
\]

P1
- read B1: 2
- read C1: 0
- calculate 2+0: 2
- write 2 -> A1

A_2 = B_2 + C_2
V_2 = M_2.A

P2
- read B2: 0
### Vector mathematics

\[
\begin{align*}
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix} \\
V &= \begin{pmatrix} 12 \\ 12 \end{pmatrix} \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \\
A &= \begin{pmatrix} 2 \\ 0 \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A
\end{align*}
\]

**P1**
- read \(B_1\): 2
- read \(C_1\): 0
- calculate \(2 + 0\): 2
- write 2 \(\rightarrow\) \(A_1\)

**P2**
- read \(B_2\): 0
**Vector mathematics**

\[
\begin{align*}
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix} \quad V &= \begin{pmatrix} 12 \\ 12 \end{pmatrix} \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \quad A &= \begin{pmatrix} 2 \\ 0 \end{pmatrix}
\end{align*}
\]

<table>
<thead>
<tr>
<th>( A_1 )</th>
<th>( B_1 + C_1 )</th>
<th>( V_1 )</th>
<th>( M_1 \cdot A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_2 )</td>
<td>( B_2 + C_2 )</td>
<td>( V_2 )</td>
<td>( M_2 \cdot A )</td>
</tr>
</tbody>
</table>

**P1**
- read \( B_1: 2 \)
- read \( C_1: 0 \)
- calculate \( 2 + 0: 2 \)
- write \( 2 \rightarrow A_1 \)
- read \( M_1: (1 \ 2) \)

**P2**
- read \( B_2: 0 \)
- read \( C_2: 5 \)
### Vector mathematics

\[
\begin{align*}
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix}, \\
V &= \begin{pmatrix} 12 \\ 12 \end{pmatrix}, \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix}, \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}, \\
A &= \begin{pmatrix} 2 \\ 0 \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A
\end{align*}
\]

**P1**

- read \(B_1: 2\)
- read \(C_1: 0\)
- calculate \(2 + 0: 2\)
- write \(2 \rightarrow A_1\)
- read \(M_1: (1, 2)\)
- read \(A: (2, 0)\)

**P2**

- read \(B_2: 0\)
- read \(C_2: 5\)
- calculate \(5 + 0: 5\)
## Vector mathematics

\[
\begin{align*}
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix} \\
V &= \begin{pmatrix} 12 \\ 12 \end{pmatrix} \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \\
A &= \begin{pmatrix} 2 \\ 0 \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A
\end{align*}
\]

### P1
- read \( B_1: 2 \)
- read \( C_1: 0 \)
- calculate \( 2 + 0: 2 \)
- write \( 2 \rightarrow A_1 \)
- read \( M_1: (1 \ 2) \)
- read \( A: (2 \ 0) \)
- calculate \( (1 \ 2) \cdot (2 \ 0): 2 \)

### P2
- read \( B_2: 0 \)
- read \( C_2: 5 \)
- calculate \( 5 + 0: 5 \)
- write \( 5 \rightarrow A_2 \)
Vector mathematics

\[
A = \begin{pmatrix} 2 \\ 5 \end{pmatrix} \quad V = \begin{pmatrix} 12 \\ 12 \end{pmatrix} \quad B = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 \\ 5 \end{pmatrix} \quad M = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \quad A = \begin{pmatrix} 2 \\ 5 \end{pmatrix}
\]

\[
A_1 = B_1 + C_1 \\
V_1 = M_1 \cdot A
\]

**P1**
- read B1: 2
- read C1: 0
- calculate 2 + 0: 2
- write 2 -> A1
- read M1: (1 2)
- read A: (2 0)
- calculate (1 2) \cdot (2 0): 2

**P2**
- read B2: 0
- read C2: 5
- calculate 5 + 0: 5
- write 5 -> A2

\[A = B + C\]
\[V = M \cdot A\]
## Vector mathematics

\[ A = (2) \quad V = (12) \]

\[ B = (2) \quad C = (0) \quad M = (1, 2) \quad A = (2) \]

<table>
<thead>
<tr>
<th>( A_1 = B_1 + C_1 )</th>
<th>( A_2 = B_2 + C_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_1 = M_1 \cdot A )</td>
<td>( V_2 = M_2 \cdot A )</td>
</tr>
</tbody>
</table>

### P1

- Read \( B_1: 2 \)
- Read \( C_1: 0 \)
- Calculate \( 2 + 0: 2 \)
- Write 2 -> \( A_1 \)

### P2

- Read \( B_2: 0 \)
- Read \( C_2: 5 \)
- Calculate \( 5 + 0: 5 \)
- Write 5 -> \( A_2 \)

---

Friday, November 18, 2011
Vector mathematics

\[
\begin{align*}
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix} \\
V &= \begin{pmatrix} 12 \\ 12 \end{pmatrix} \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \\
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 . A
\end{align*}
\]

\[
\begin{align*}
A_2 &= B_2 + C_2 \\
V_2 &= M_2 . A
\end{align*}
\]

**P1**
- read B1: 2
- read C1: 0
- calculate \(2 + 0 = 2\)
- write \(2\) -> A1
- read M1: (1 2)
- read A: (2 0)
- calculate \((1 2) \cdot (2 0) = 2\)
- write \(2\) -> V1

**P2**
- read B2: 0
- read C2: 5
- calculate \(5 + 0 = 5\)
- write \(5\) -> A2
- read M2: (1 2)
- read A: (2 5)
Vector mathematics

\[ \begin{align*}
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix} \\
V &= \begin{pmatrix} 12 \\ 12 \end{pmatrix} \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \\
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix}
\end{align*} \]

\[ \begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 . A
\end{align*} \]

P1
read B1: 2
read C1: 0
calculate \( 2 + 0 = 2 \)
write 2 -> A1
read M1: (1 2)
read A: (2 0)
calculate \( (1 2) . (2 0) = 2 \)
write 2 -> V1

P2
read B2: 0
read C2: 5
calculate \( 5 + 0 = 5 \)
write 5 -> A2
read M2: (1 2)
read A: (2 5)
calculate \( (1 2) . (2 5) = 12 \)
Vector mathematics

A = B + C
V = MxA

\[ A = \begin{pmatrix} 2 \\ 5 \end{pmatrix} \quad V = \begin{pmatrix} 12 \\ 12 \end{pmatrix} \quad B = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 \\ 5 \end{pmatrix} \quad M = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \quad A = \begin{pmatrix} 2 \\ 5 \end{pmatrix} \]

\[ A_1 = B_1 + C_1 \]
\[ V_1 = M_1 \cdot A \]

P1
read B1: 2
read C1: 0
calculate 2 + 0: 2
write 2 -> A1
read M1: (1 2)
read A: (2 0)
calculate (1 2) . (2 0): 2
write 2 -> V1

P2
read B2: 0
read C2: 5
calculate 5 + 0: 5
write 5 -> A2
read M2: (1 2)
read A: (2 5)
calculate (1 2) . (2 5): 12
write 12 -> V2
Vector mathematics

\[ A = B + C \]
\[ V = M \times A \]

\[
\begin{bmatrix}
2 \\
5
\end{bmatrix}
\]
\[
\begin{bmatrix}
12 \\
12
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 \\
0
\end{bmatrix}
\]
\[
\begin{bmatrix}
0 \\
5
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 2
\end{bmatrix}
\]
\[
\begin{bmatrix}
2
\end{bmatrix}
\]

\[ A_1 = B_1 + C_1 \]
\[ V_1 = M_1 \times A \]

P1
read B1: 2
read C1: 0
calculate 2 + 0: 2
write 2 -> A1

read M1: (1 2)
read A: (2 0)
calculate (1 2) \cdot (2 0): 2
write 2 -> V1

\[
\begin{bmatrix}
2 \\
12
\end{bmatrix}
\]

P2
read B2: 0
read C2: 5
calculate 5 + 0: 5
write 5 -> A2
read M2: (1 2)
read A: (2 5)
calculate (1 2) \cdot (2 5): 2
write 12 -> V2
Vector mathematics

\[ A = B + C \]

\[ V = M \times A \]
Vector mathematics

\[ A = B + C \]
\[ V = M \times A \]
Vector mathematics

Step 1

\[ A = B + C \]

\[ V = M \times A \]
Vector mathematics

Step 1

\[ A = B + C \]

Step 2

\[ V = M \times A \]
Vector mathematics

Step 1

\[
A = B + C
\]

\[
V = M \times A
\]

Step 2

Threads must wait for each other. Only move on when all have finished previous step.
Correctness

The outcome should always be equivalent to some serial ordering of individual steps.

**serial ordering**: if the threads were executed individually, from start to finish, one after the other instead of in parallel.
Problem 1: inconsistent values

Need ways to make threads wait.
Problem 1: inconsistent values

Inconsistent values

Need ways to make threads wait.
Problem 1: inconsistent values

Inconsistent values
  - A thread reads a value and starts processing

Need ways to make threads wait.
Problem 1: inconsistent values

Inconsistent values

- A thread reads a value and starts processing
- Another thread changes the value

Need ways to make threads wait.
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

Need ways to make threads wait.
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

Problem 2: unsynchronized threads

Need ways to make threads wait.
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

Problem 2: unsynchronized threads

Unsynchronized threads

Need ways to make threads wait.
Problem 1: inconsistent values

Inconsistent values
  - A thread reads a value and starts processing
  - Another thread changes the value
  - The first thread’s value is inconsistent and out of date

Problem 2: unsynchronized threads

Unsynchronized threads
  - Operations is a series of steps

Need ways to make threads wait.
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

Problem 2: unsynchronized threads

Unsynchronized threads
- Operations is a series of steps
- Threads must wait until all have finished previous step

Need ways to make threads wait.
Problem 1: inconsistent values
Problem 1: inconsistent values

Inconsistent values
Problem 1: inconsistent values

Inconsistent values
  - A thread reads a value and starts processing
Problem 1: inconsistent values

Inconsistent values

- A thread reads a value and starts processing
- Another thread changes the value
Problem 1: inconsistent values

Inconsistent values

- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

P1

P2

harmless code
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

P1
harmless code
harmless code

P2
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

P1
- harmless code
- harmless code
- modify shared variable

P2
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

P1
harmless code
harmless code
modify shared variable
.............

P2
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

P1

harmless code
harmless code
modify shared variable
............
............

P2
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

P1

harmless code
harmless code
modify shared variable

............
............
............

P2

............
............
............
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

P1

harmless code
harmless code
modify shared variable

.............
.............
.............

P2

.............
Problem 1: inconsistent values

Inconsistent values

- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

P1

harmless code
harmless code
modify shared variable
.............
.............
.............
write shared variable

P2
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

Should not be interrupted by other threads that access same variable
Problem 1: inconsistent values

Inconsistent values

- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

\[
\begin{align*}
\text{P1} & \quad \text{P2} \\
\text{harmless code} & \quad \text{Should not be interrupted by other threads that access same variable} \\
\text{harmless code} & \quad \text{by other threads that access same variable} \\
modify \text{ shared variable} & \quad \text{access same variable} \\
\ldots \ldots \ldots & \\
\ldots \ldots \ldots & \\
\ldots \ldots \ldots & \\
\text{write shared variable} & \\
\text{harmless code} &
\end{align*}
\]
Problem 1: inconsistent values

Inconsistent values
- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

P1

harmless code
harmless code
modify shared variable
.............
.............
.............
write shared variable
harmless code
harmless code

P2

Should not be interrupted by other threads that access same variable
Inconsistent values

- A thread reads a value and starts processing
- Another thread changes the value
- The first thread’s value is inconsistent and out of date

P1

harmless code
harmless code
modify shared variable
..............
..............
..............
write shared variable
harmless code
harmless code

P2

Critical Section

Should not be interrupted by other threads that access same variable
Terminology
Terminology

“Critical section”
Terminology

“Critical section”
- A section of code that should not be interrupted
Terminology

“Critical section”
- A section of code that should not be interrupted
- Should be executed as if it is a single statement
Terminology

“Critical section”
- A section of code that should not be interrupted
- Should be executed as if it is a single statement
Terminology

“Critical section”
- A section of code that should not be interrupted
- Should be executed as if it is a single statement

“Atomic” and “Atomicity”
Terminology

“Critical section”
- A section of code that should not be interrupted
- Should be executed as if it is a single statement

“Atomic” and “Atomicity”
- Atomic: cannot be broken down into further pieces
**Terminology**

"Critical section"
- A section of code that should not be interrupted
- Should be executed as if it is a single statement

"Atomic" and "Atomicity"
- Atomic: cannot be broken down into further pieces
- Atomic (when applied to code): cannot be interrupted, like a single hardware instruction.
**Terminology**

“Critical section”
- A section of code that should not be interrupted
- Should be executed as if it is a single statement

“Atomic” and “Atomicity”
- Atomic: cannot be broken down into further pieces
- Atomic (when applied to code): cannot be interrupted, like a single hardware instruction.
- Atomicity: a guarantee that the code will not be interrupted.
Terminology

“Critical section”
- A section of code that should not be interrupted
- Should be executed as if it is a single statement

“Atomic” and “Atomicity”
- Atomic: cannot be broken down into further pieces
- Atomic (when applied to code): cannot be interrupted, like a single hardware instruction.
- Atomicity: a guarantee that the code will not be interrupted.

Critical sections need to have atomicity.
Protecting shared state with shared state
Protecting shared state *with shared state*

Use shared state to store signals
Protecting shared state with shared state

Use shared state to store signals

Signals can indicate:
Protecting shared state *with shared state*

Use shared state to store signals

Signals can indicate:
- A variable is in use
Protecting shared state with shared state

Use shared state to store signals

Signals can indicate:
  - A variable is in use
  - A step is complete (or not)
Protecting shared state with shared state

Use shared state to store signals

Signals can indicate:

- A variable is in use
- A step is complete (or not)
- How many threads are using a resource
Protecting shared state with shared state

Use shared state to store signals

Signals can indicate:
- A variable is in use
- A step is complete (or not)
- How many threads are using a resource
- Whether or not a condition is true
Protecting shared state with shared state

Use shared state to store signals

Signals can indicate:

- A variable is in use
- A step is complete (or not)
- How many threads are using a resource
- Whether or not a condition is true

Signals:
Protecting shared state \textit{with shared state}

Use shared state to store signals

Signals can indicate:
\begin{itemize}
  \item A variable is in use
  \item A step is complete (or not)
  \item How many threads are using a resource
  \item Whether or not a condition is true
\end{itemize}

Signals:
\begin{itemize}
  \item Locks or mutexes (mutual exclusions)
\end{itemize}
Protecting shared state with shared state

Use shared state to store signals

Signals can indicate:
- A variable is in use
- A step is complete (or not)
- How many threads are using a resource
- Whether or not a condition is true

Signals:
- Locks or mutexes (mutual exclusions)
- Semaphores
Protecting shared state with shared state

Use shared state to store signals

Signals can indicate:

- A variable is in use
- A step is complete (or not)
- How many threads are using a resource
- Whether or not a condition is true

Signals:

- Locks or mutexes (mutual exclusions)
- Semaphores
- Conditions
Protecting shared state \textit{with shared state}

Use shared state to store signals

Signals can indicate:
\begin{itemize}
  \item A variable is in use
  \item A step is complete (or not)
  \item How many threads are using a resource
  \item Whether or not a condition is true
\end{itemize}

Signals:
\begin{itemize}
  \item Locks or mutexes (mutual exclusions)
  \item Semaphores
  \item Conditions
\end{itemize}

Don’t physically protect shared state
Protecting shared state with shared state

Use shared state to store signals

Signals can indicate:
- A variable is in use
- A step is complete (or not)
- How many threads are using a resource
- Whether or not a condition is true

Signals:
- Locks or mutexes (mutual exclusions)
- Semaphores
- Conditions

Don’t physically protect shared state

Convention and shared rules for signals protect shared state.
Protecting shared state *with shared state*

Use shared state to store signals

Signals can indicate:
- A variable is in use
- A step is complete (or not)
- How many threads are using a resource
- Whether or not a condition is true

Signals:
- Locks or mutexes (mutual exclusions)
- Semaphores
- Conditions

Don’t physically protect shared state

Convention and shared rules for signals protect shared state.
- Like traffic signals “protect” an intersection
Locks

Implemented using real atomic hardware instructions.
Locks

Implemented using real atomic hardware instructions.

Used to signal that a shared resource is in use.
Locks

Implemented using real atomic hardware instructions.

Used to signal that a shared resource is in use.

acquire()
Locks

Implemented using real atomic hardware instructions.

Used to signal that a shared resource is in use.

acquire()

- “set” the signal.
Locks

Implemented using real atomic hardware instructions.

Used to signal that a shared resource is in use.

acquire()

- “set” the signal.
- No other threads will be able to acquire()
Locks

Implemented using real atomic hardware instructions.

Used to signal that a shared resource is in use.

acquire()
  ▪ “set” the signal.
  ▪ No other threads will be able to acquire()
  ▪ They will automatically wait until ...
Locks

Implemented using real atomic hardware instructions.

Used to signal that a shared resource is in use.

acquire()
  ▪ “set” the signal.
  ▪ No other threads will be able to acquire()
  ▪ They will automatically wait until ...

release()
Locks

Implemented using real atomic hardware instructions.

Used to signal that a shared resource is in use.

acquire()
  ▪ “set” the signal.
  ▪ No other threads will be able to acquire()
  ▪ They will automatically wait until ...

release()
  ▪ “unset” a signal.
Locks

Implemented using real atomic hardware instructions.

Used to signal that a shared resource is in use.

acquire()
  ▪ “set” the signal.
  ▪ No other threads will be able to acquire()
  ▪ They will automatically wait until ...

release()
  ▪ “unset” a signal.
  ▪ Any one thread that was waiting for acquire() will now succeed
Using locks: bank balance example

```python
def make_withdraw(balance):
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
    return withdraw
```
Using locks: bank balance example

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

w = make_withdraw(10)
balance = 10
```
Using locks: bank balance example

def make_withdraw(balance):
    def withdraw(amount):
        nonlocal 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)
Using locks: bank balance example

```python
def make_withdraw(balance):
    def withdraw(amount):
        nonlocal 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 balance: 10
```
Using locks: bank balance example

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

w = make_withdraw(10)
balance = 10

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

w(7)
read balance: 10
```

Friday, November 18, 2011
Using locks: bank balance example

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

w = make_withdraw(10)
balance = 10

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

w(7)
read balance: 10
read amount: 7
```
Using locks: bank balance example

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

w = make_withdraw(10)
balance = 10

w(8)
read balance: 10
read amount: 8
8 > 10: False
if False
w(7)
read balance: 10
read amount: 7
7 > 10: False
```

Friday, November 18, 2011
Using locks: bank balance example

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

w = make_withdraw(10)
balance = 10

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

w(7)
read balance: 10
read amount: 7
7 > 10: False
if False
```
Using locks: bank balance example

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

w = make_withdraw(10)
balance = 10

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

w(7)
read balance: 10
read amount: 7
7 > 10: False
if False
10 - 7: 3
```

Friday, November 18, 2011
Using locks: bank balance example

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

w = make_withdraw(10)
balance = 10

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

w(7)
read balance: 10
read amount: 7
7 > 10: False
if False
10 - 7: 3```
Using locks: bank balance example

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

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

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

w(7)

read balance: 10
read amount: 7
7 > 10: False
if False
10 - 7: 3
write balance -> 3
```

Friday, November 18, 2011
Using locks: bank balance example

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

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

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

```
read balance: 10
read amount: 7
7 > 10: False
if False
10 - 7: 3
write balance -> 3
```
Using locks: bank balance example

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

w = make_withdraw(10)
balance = 10

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

w(7)
read balance: 10
read amount: 7
7 > 10: False
if False
10 - 7: 3
write balance -> 3
print 3
```
Using locks: bank balance example

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

w = make_withdraw(10)
balance = 10
w(8)
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
print 2

w(7)
read balance: 10
read amount: 7
7 > 10: False
if False
10 - 7: 3
write balance -> 3
print 3
```
Using locks: bank balance example

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

critical section
Using locks: bank balance example

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

Critical section

New code
Using locks: bank balance example

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

critical section

New code

```
def make_withdraw(balance)
```
Using locks: bank balance example

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

def make_withdraw(balance):
    balance_lock = Lock()
```

New code
Using locks: bank balance example

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

return withdraw
```

New code

```python
def make_withdraw(balance):
    balance_lock = Lock()
    def withdraw(amount):
```
Using locks: bank balance example

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

Critical section

New code

```python
def make_withdraw(balance):
    balance_lock = Lock()
    def withdraw(amount):
        nonlocal balance
```
Using locks: bank balance example

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

New code

def make_withdraw(balance)
    balance_lock = Lock()
def withdraw(amount):
    nonlocal balance
    # try to acquire the lock
Using locks: bank balance example

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

New code

```python
def make_withdraw(balance):
    balance_lock = Lock()
    def withdraw(amount):
        nonlocal balance
        # try to acquire the lock
        balance_lock.acquire()
```
Using locks: bank balance example

```python
def make_withdraw(balance):
    def withdraw(amount):
        nonlocal balance
        # try to acquire the lock
        balance_lock.acquire()
        # once successful, enter the critical section
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
        print(balance)
        return withdraw
```

critical section

New code

```python
def make_withdraw(balance):
    balance_lock = Lock()
    def withdraw(amount):
        nonlocal balance
        # try to acquire the lock
        balance_lock.acquire()
        # once successful, enter the critical section
```
Using locks: bank balance example

```python
def make_withdraw(balance):
    def withdraw(amount):
        nonlocal balance
        # try to acquire the lock
        balance_lock.acquire()
        # once successful, enter the critical section
        if amount > balance:
            print('Insufficient funds')
        else:
            balance -= amount
            print(balance)
        return withdraw

balance_lock = Lock()
def withdraw(amount):
    nonlocal balance
    # try to acquire the lock
    balance_lock.acquire()
    # once successful, enter the critical section
    if amount > balance:
```
Using locks: bank balance example

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

critical section

New code

```python
def make_withdraw(balance):
    balance_lock = Lock()
    def withdraw(amount):
        nonlocal balance
        # try to acquire the lock
        balance_lock.acquire()
        # once successful, enter the critical section
        if amount > balance:
            print("Insufficient funds")
        else:
            balance = balance - amount
        print(balance)
        return withdraw
```
Using locks: bank balance example

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

balance_lock = Lock()
def withdraw(amount):
    nonlocal balance
    # try to acquire the lock
    balance_lock.acquire()
    # once successful, enter the critical section
    if amount > balance:
        print("Insufficient funds")
    else:
        balance = balance - amount
        print(balance)
```

New code
Using locks: bank balance example

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

Critical section

New code

```python
def make_withdraw(balance):
    balance_lock = Lock()
    def withdraw(amount):
        nonlocal balance
        # try to acquire the lock
        balance_lock.acquire()
        # once successful, enter the critical section
        if amount > balance:
            print("Insufficient funds")
        else:
            balance = balance - amount
```
Using locks: bank balance example

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

critical section

New code

```python
def make_withdraw(balance):
    balance_lock = Lock()
    def withdraw(amount):
        nonlocal balance
        # try to acquire the lock
        balance_lock.acquire()
        # once successful, enter the critical section
        if amount > balance:
            print("Insufficient funds")
        else:
            balance = balance - amount
        print(balance)
    return withdraw
```
Using locks: bank balance example

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

critical section

New code

```python
def make_withdraw(balance):
    balance_lock = Lock()
    def withdraw(amount):
        nonlocal balance
        # try to acquire the lock
        balance_lock.acquire()
        # once successful, enter the critical section
        if amount > balance:
            print("Insufficient funds")
        else:
            balance = balance - amount
        print(balance)
        # upon exiting the critical section, release the lock
```
Using locks: bank balance example

```python
def make_withdraw(balance):
    def withdraw(amount):
        nonlocal balance
        # try to acquire the lock
        balance_lock.acquire()
        # once successful, enter the critical section
        if amount > balance:
            print('Insufficient funds')
        else:
            balance = balance - amount
            print(balance)
        # upon exiting the critical section, release the lock
        balance_lock.release()
```

New code

```python
def make_withdraw(balance):
    balance_lock = Lock()
    def withdraw(amount):
        nonlocal balance
        # try to acquire the lock
        balance_lock.acquire()
        # once successful, enter the critical section
        if amount > balance:
            print("Insufficient funds")
        else:
            balance = balance - amount
            print(balance)
        # upon exiting the critical section, release the lock
        balance_lock.release()
```

critical section
Using locks: bank balance example
Using locks: bank balance example

```python
w = make_withdraw(10)
balance = 10
balance_lock = Lock()
```
Using locks: bank balance example

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

Friday, November 18, 2011
Using locks: bank balance example

\[
\begin{align*}
  w &= \text{make\_withdraw}(10) \\
  \text{balance} &= 10 \\
  \text{balance\_lock} &= \text{Lock}()
\end{align*}
\]

\[
\begin{array}{c|c}
  w(8) & w(7) \\
P1 & P2
\end{array}
\]
Using locks: bank balance example

```python
w = make_withdraw(10)
balance = 10
balance_lock = Lock()
```

P1
acquire balance_lock: ok

P2
Using locks: bank balance example

```
w = make_withdraw(10)
balance = 10
balance_lock = Lock() acquired by p1
```

P1
acquire balance_lock: ok

P2
Using locks: bank balance example

```python
w = make_withdraw(10)
balance = 10
balance_lock = Lock()  # acquired by p1

w(8)

P1
acquire balance_lock: ok
read balance: 10

w(7)

P2
acquire balance_lock: wait
```
Using locks: bank balance example

```python
w = make_withdraw(10)
balance = 10
balance_lock = Lock() acquired by p1

w(8)

P1
acquire balance_lock: ok
read balance: 10
read amount: 8

w(7)

P2
acquire balance_lock: wait
wait
wait
```
Using locks: bank balance example

```python
w = make_withdraw(10)
balance = 10
balance_lock = Lock() acquired by p1

w(8)
w(7)

P1
acquire balance_lock: ok
read balance: 10
read amount: 8
8 > 10: False

P2
acquire balance_lock: wait
wait
wait
```
Using locks: bank balance example

```python
w = make_withdraw(10)
balance = 10
balance_lock = Lock()  # acquired by p1

w(8)

P1
acquire balance_lock: ok
read balance: 10
read amount: 8
8 > 10: False
if False

w(7)

P2
acquire balance_lock: wait
wait
wait
wait
```
Using locks: bank balance example

\[
\begin{align*}
    w &= \text{make\_withdraw}(10) \\
    \text{balance} &= 10 \\
    \text{balance\_lock} &= \text{Lock()} \quad \text{acquired by p1}
\end{align*}
\]

P1
acquire balance\_lock: ok
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2

P2
acquire balance\_lock: wait
wait
wait
wait
wait

Friday, November 18, 2011
Using locks: bank balance example

```python
w = make_withdraw(10)
balance = 10
balance_lock = Lock()  # acquired by p1
```

P1

acquire balance_lock: ok
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2

P2

acquire balance_lock: wait
wait
wait
wait
wait
wait
Using locks: bank balance example

\[
\begin{align*}
  w &= \text{make\_withdraw}(10) \\
  \text{balance} &= 10 \\
  \text{balance\_lock} &= \text{Lock()} \quad \text{acquired by p1}
\end{align*}
\]

P1
acquire balance\_lock: ok
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
print 2

P2
acquire balance\_lock: wait
wait
wait
wait
wait
wait
Using locks: bank balance example

```python
w = make_withdraw(10)
balance = 10
balance_lock = Lock()  # acquired by p1

w(8)

P1
acquire balance_lock: ok
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
print 2
release balance_lock

w(7)

P2
acquire balance_lock: wait
wait
wait
wait
wait
wait
wait
wait
wait
```
Using locks: bank balance example

\[
\begin{align*}
w &= \text{make_withdraw}(10) \\
    \text{balance} &= 10 \\
    \text{balance\_lock} &= \text{Lock()}
\end{align*}
\]

\[
\begin{align*}
w(8) & & w(7)
\end{align*}
\]

**P1**
acquire balance\_lock: ok
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
print 2
release balance\_lock

**P2**
acquire balance\_lock: wait
wait
wait
wait
wait
wait
wait
wait
wait
wait
wait
Using locks: bank balance example

```python
w = make_withdraw(10)
balance = 10
balance_lock = Lock()
w(8)
w(7)

P1
acquire balance_lock: ok
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
print 2
release balance_lock

P2
acquire balance_lock: wait
wait
wait
wait
wait
wait
wait
acquire balance_lock: ok
```
Using locks: bank balance example

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

acquire balance_lock: ok
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
print 2
release balance_lock

acquire balance_lock: wait
wait
wait
wait
wait
wait
acquire balance_lock: ok
```
Using locks: bank balance example

```python
w = make_withdraw(10)
balance = 10
balance_lock = Lock()  # acquired by p2

P1
acquire balance_lock: ok
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
print 2
release balance_lock

P2
acquire balance_lock: wait
wait
wait
wait
wait
wait
acquire balance_lock: ok
read balance: 2
```
Using locks: bank balance example

\[
w = \text{make_withdraw}(10)\\
balance = 10\\
balance\_lock = \text{Lock()}
\]

P1
acquire balance\_lock: ok
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
print 2
release balance\_lock

P2
acquire balance\_lock: wait
wait
wait
wait
wait
wait
acquire balance\_lock: ok
read balance: 2
read amount: 7
Using locks: bank balance example

```
w = make_withdraw(10)
balance = 10
balance_lock = Lock() acquired by p2
```

**P1**
acquire balance_lock: ok
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
print 2
release balance_lock

**P2**
acquire balance_lock: wait
wait
wait
wait
wait
wait
wait
acquire balance_lock: ok
read balance: 2
read amount: 7
7 > 2: True
Using locks: bank balance example

```
w = make_withdraw(10)
balance = 10
balance_lock = Lock()  # acquired by p2

w(8)

P1
acquire balance_lock: ok
read balance: 10
read amount: 8
8 > 10: False
if False
10 - 8: 2
write balance -> 2
print 2
release balance_lock

w(7)

P2
acquire balance_lock: wait
wait
wait
wait
acquire balance_lock: ok
read balance: 2
read amount: 7
7 > 2: True
if True
```
Using locks: bank balance example

\[
\begin{align*}
  w &= \text{make\_withdraw}(10) \\
  \text{balance} &= 10 \\
  \text{balance\_lock} &= \text{Lock()} \\
\end{align*}
\]

w(8)

\begin{itemize}
  \item P1
    \begin{itemize}
      \item acquire balance\_lock: ok
      \item read balance: 10
      \item read amount: 8
      \item 8 > 10: False
      \item if False
      \item 10 - 8: 2
      \item write balance -> 2
      \item print 2
      \item release balance\_lock
    \end{itemize}
\end{itemize}

w(7)

\begin{itemize}
  \item P2
    \begin{itemize}
      \item acquire balance\_lock: wait
      \item wait
      \item wait
      \item wait
      \item if False
      \item if False
      \item if False
      \item write balance -> 2
      \item print 'Insufficient funds'
    \end{itemize}
\end{itemize}
Using locks: bank balance example

\[
\begin{align*}
  w &= \text{make\_withdraw}(10) \\
  \text{balance} &= 10 \\
  \text{balance\_lock} &= \text{Lock()} \\
\end{align*}
\]

\text{acquired by p2}

\text{P1}

\begin{align*}
  \text{acquire balance\_lock: ok} \\
  \text{read balance: 10} \\
  \text{read amount: 8} \\
  8 &> 10: \text{False} \\
  \text{if False} \\
  10 - 8 &:= 2 \\
  \text{write balance} &\rightarrow 2 \\
  \text{print 2} \\
  \text{release balance\_lock}
\end{align*}

\text{P2}

\begin{align*}
  \text{acquire balance\_lock: wait} \\
  \text{wait} \\
  \text{wait} \\
  \text{wait} \\
  \text{wait} \\
  \text{wait} \\
  \text{wait} \\
  \text{acquire balance\_lock: ok} \\
  \text{read balance: 2} \\
  \text{read amount: 7} \\
  7 &> 2: \text{True} \\
  \text{if True} \\
  \text{print 'Insufficient funds'} \\
  \text{release balance\_lock}
\end{align*}
Using locks: bank balance example

\[
\begin{align*}
    w &= \text{make_withdraw}(10) \\
    \text{balance} &= 10 \\
    \text{balance\_lock} &= \text{Lock}()
\end{align*}
\]

\[
\begin{align*}
    \text{P1} & \quad \text{acquire balance\_lock}: \text{ok} \\
    & \quad \text{read balance}: 10 \\
    & \quad \text{read amount}: 8 \\
    & \quad 8 > 10: \text{False} \\
    & \quad \text{if False} \\
    & \quad 10 - 8: 2 \\
    & \quad \text{write balance} \rightarrow 2 \\
    & \quad \text{print} \ 2 \\
    & \quad \text{release balance\_lock}
\end{align*}
\]

\[
\begin{align*}
    \text{P2} & \quad \text{acquire balance\_lock}: \text{wait} \\
    & \quad \text{wait} \\
    & \quad \text{wait} \\
    & \quad \text{wait} \\
    & \quad \text{wait} \\
    & \quad \text{wait} \\
    & \quad \text{acquire balance\_lock}: \text{ok} \\
    & \quad \text{read balance}: 2 \\
    & \quad \text{read amount}: 7 \\
    & \quad 7 > 2: \text{True} \\
    & \quad \text{if True} \\
    & \quad \text{print} \ '\text{Insufficient funds}' \\
    & \quad \text{release balance\_lock}
\end{align*}
\]
def make_withdraw(balance):  
    balance_lock = Lock()  
    def withdraw(amount):  
        nonlocal balance  
        # try to acquire the lock  
        balance_lock.acquire()  
        # once successful, enter the critical section  
        if amount > balance:  
            print("Insufficient funds")  
        else:  
            balance = balance - amount  
            print(balance)  
        # upon exiting the critical section, release the lock  
        balance_lock.release()
def make_withdraw(balance):
    balance_lock = Lock()
    def withdraw(amount):
        nonlocal balance
        # try to acquire the lock
        balance_lock.acquire()
        # once successful, enter the critical section
        if amount > balance:
            print("Insufficient funds")
        else:
            balance = balance - amount
            print(balance)
        # upon exiting the critical section, release the lock
        balance_lock.release()
def make_withdraw(balance):
    balance_lock = Lock()

def withdraw(amount):
    nonlocal balance
    # try to acquire the lock
    balance_lock.acquire()
    # once successful, enter the critical section
    if amount > balance:
        print("Insufficient funds")
    else:
        balance = balance - amount
        print(balance)
    # upon exiting the critical section, release the lock
    balance_lock.release()

No two processes can be in the critical section at the same time.
def make_withdraw(balance):
    balance_lock = Lock()

def withdraw(amount):
    nonlocal balance
    # try to acquire the lock
    balance_lock.acquire()
    # once successful, enter the critical section
    if amount > balance:
        print("Insufficient funds")
    else:
        balance = balance - amount
        print(balance)
    # upon exiting the critical section, release the lock
    balance_lock.release()

No two processes can be in the critical section at the same time.

Whichever gets to balance_lock.acquire() first gets to finish.
def make_withdraw(balance):
    balance_lock = Lock()
    def withdraw(amount):
        nonlocal balance
        # try to acquire the lock
        balance_lock.acquire()
        # once successful, enter the critical section
        if amount > balance:
            print("Insufficient funds")
        else:
            balance = balance - amount
            print(balance)
        # upon exiting the critical section, release the lock
        balance_lock.release()

No two processes can be in the critical section at the same time.

Whichever gets to balance_lock.acquire() first gets to finish.

All others have to wait until it’s finished.
def make_withdraw(balance):
    balance_lock = Lock()

def withdraw(amount):
   nonlocal balance
    # try to acquire the lock
    balance_lock.acquire()
    # once successful, enter the critical section
    if amount > balance:
        print("Insufficient funds")
    else:
        balance = balance - amount
        print(balance)
    # upon exiting the critical section, release the lock
    balance_lock.release()  # important, allows others to proceed

No two processes can be in the critical section at the same time.

Whichever gets to balance_lock.acquire() first gets to finish.

All others have to wait until it’s finished.
def make_withdraw(balance):
    balance_lock = Lock()

def withdraw(amount):
    nonlocal balance
    # try to acquire the lock
    balance_lock.acquire()
    # once successful, enter the critical section
    if amount > balance:
        print("Insufficient funds")
    else:
        balance = balance - amount
        print(balance)
    # upon exiting the critical section, release the lock
    balance_lock.release()  # important, allows others to proceed

No two processes can be in the critical section at the same time.

Whichever gets to balance_lock.acquire() first gets to finish.

All others have to wait until it’s finished.
Semaphores
Semaphores

Used to protect access to limited resources

Each has a limit, N

Can be acquire()'d N times

After that, processes trying to acquire() automatically wait

Until another process release()'s
Semaphores example: database

A database that can only support 2 connections at a time.
Semaphores example: database

A database that can only support 2 connections at a time.

# set up the semaphore
Semaphores example: database

A database that can only support 2 connections at a time.

```python
# set up the semaphore
db_semaphore = Semaphore(2)
```
Semaphores example: database

A database that can only support 2 connections at a time.

```python
# set up the semaphore
db_semaphore = Semaphore(2)

def insert(data):
```
Semaphores example: database

A database that can only support 2 connections at a time.

```python
# set up the semaphore
db_semaphore = Semaphore(2)

def insert(data):
    # try to acquire the semaphore
```
Semaphores example: database

A database that can only support 2 connections at a time.

```python
# set up the semaphore
db_semaphore = Semaphore(2)

def insert(data):
    # try to acquire the semaphore
    db_semaphore.acquire()
```
A database that can only support 2 connections at a time.

```python
# set up the semaphore
db_semaphore = Semaphore(2)

def insert(data):
    # try to acquire the semaphore
    db_semaphore.acquire()
    # if successful, proceed
```
Semaphores example: database

A database that can only support 2 connections at a time.

```python
# set up the semaphore
db_semaphore = Semaphore(2)

def insert(data):
    # try to acquire the semaphore
db_semaphore.acquire()
    # if successful, proceed
    database.insert(data)
```
Semaphores example: database

A database that can only support 2 connections at a time.

```python
# set up the semaphore
db_semaphore = Semaphore(2)

def insert(data):
    # try to acquire the semaphore
    db_semaphore.acquire()
    # if successful, proceed
    database.insert(data)
    # release the semaphore
```
Semaphores example: database

A database that can only support 2 connections at a time.

```python
# set up the semaphore
db_semaphore = Semaphore(2)

def insert(data):
    # try to acquire the semaphore
db_semaphore.acquire()
    # if successful, proceed
database.insert(data)
    # release the semaphore
    db_semaphore.release()
```
db_semaphore = Semaphore(2)

def insert(data):
    db_semaphore.acquire()
    database.insert(data)
    db_semaphore.release()
Example: database

db_semaphore = Semaphore(2)

def insert(data):
    db_semaphore.acquire()
    database.insert(data)
    db_semaphore.release()
Example: database

```python
db_semaphore = Semaphore(2)

def insert(data):
    db_semaphore.acquire()
    database.insert(data)
    db_semaphore.release()

insert(7)  insert(8)  insert(9)
P1          P2          P3
```
Example: database

```python
db_semaphore = Semaphore(2)

def insert(data):
    db_semaphore.acquire()
    database.insert(data)
    db_semaphore.release()
```

<table>
<thead>
<tr>
<th>P1</th>
<th>insert(7)</th>
<th>acquire db_semaphore: ok</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>insert(8)</td>
<td>acquire db_semaphore: wait</td>
</tr>
<tr>
<td>P3</td>
<td>insert(9)</td>
<td>acquire db_semaphore: ok</td>
</tr>
</tbody>
</table>
Example: database

db_semaphore = Semaphore(2)

def insert(data):
    db_semaphore.acquire()
    database.insert(data)
    db_semaphore.release()

insert(7)  
acquire db_semaphore: ok
read data: 7

insert(8)  
acquire db_semaphore: wait
wait

insert(9)  
acquire db_semaphore: ok
read data: 9
Example: database

```python
db_semaphore = Semaphore(2)

def insert(data):
    db_semaphore.acquire()
    database.insert(data)
    db_semaphore.release()
```

<table>
<thead>
<tr>
<th>insert(7)</th>
<th>insert(8)</th>
<th>insert(9)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P1</strong></td>
<td><strong>P2</strong></td>
<td><strong>P3</strong></td>
</tr>
<tr>
<td>acquire db_semaphore: ok</td>
<td>acquire db_semaphore: wait</td>
<td>acquire db_semaphore: ok</td>
</tr>
<tr>
<td>read data: 7</td>
<td>wait</td>
<td>wait</td>
</tr>
<tr>
<td>read global database</td>
<td>wait</td>
<td>read data: 9</td>
</tr>
</tbody>
</table>

Friday, November 18, 2011
Example: database

db_semaphore = Semaphore(2)

def insert(data):
    db_semaphore.acquire()
    database.insert(data)
    db_semaphore.release()
Example: database

\[
\text{db_semaphore} = \text{Semaphore}(2)
\]

```
def insert(data):
    \text{db_semaphore.acquire}()
    \text{database.insert(data)}
    \text{db_semaphore.release}()
```

<table>
<thead>
<tr>
<th>insert(7)</th>
<th>insert(8)</th>
<th>insert(9)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P1</strong></td>
<td><strong>P2</strong></td>
<td><strong>P3</strong></td>
</tr>
<tr>
<td>acquire \text{db_semaphore}: ok</td>
<td>acquire \text{db_semaphore}: wait</td>
<td>acquire \text{db_semaphore}: ok</td>
</tr>
<tr>
<td>read \text{data: 7}</td>
<td>read \text{wait}</td>
<td>read \text{data: 9}</td>
</tr>
<tr>
<td>read \text{global database}</td>
<td>read \text{wait}</td>
<td>read \text{global database}</td>
</tr>
<tr>
<td>insert \text{7 into database}</td>
<td>wait</td>
<td>insert \text{9 into database}</td>
</tr>
<tr>
<td>release \text{db_semaphore}: ok</td>
<td>wait</td>
<td>release \text{db_semaphore}: ok</td>
</tr>
</tbody>
</table>
Example: database

db_semaphore = Semaphore(2)

def insert(data):
    db_semaphore.acquire()
    database.insert(data)
    db_semaphore.release()

insert(7)

P1
acquire db_semaphore: ok
read data: 7
read global database
insert 7 into database
release db_semaphore: ok

insert(8)

P2
acquire db_semaphore: wait
wait
wait
acquire db_semaphore: ok
read data: 8

insert(9)

P3
acquire db_semaphore: ok
read data: 9
read global database
insert 9 into database
release db_semaphore: ok
Example: database

db_semaphore = Semaphore(2)

def insert(data):
    db_semaphore.acquire()
    database.insert(data)
    db_semaphore.release()

insert(7)  insert(8)  insert(9)

P1
acquire db_semaphore: ok
read data: 7
read global database
insert 7 into database
release db_semaphore: ok

P2
acquire db_semaphore: wait
wait
wait
acquire db_semaphore: ok
read data: 8
read global database

P3
acquire db_semaphore: ok
read data: 9
read global database
insert 9 into database
release db_semaphore: ok
Example: database

```python
db_semaphore = Semaphore(2)

def insert(data):
    db_semaphore.acquire()
    database.insert(data)
    db_semaphore.release()
```

<table>
<thead>
<tr>
<th>insert(7)</th>
<th>insert(8)</th>
<th>insert(9)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P1</strong></td>
<td><strong>P2</strong></td>
<td><strong>P3</strong></td>
</tr>
<tr>
<td>acquire db_semaphore: ok</td>
<td>acquire db_semaphore: wait</td>
<td>acquire db_semaphore: ok</td>
</tr>
<tr>
<td>read data: 7</td>
<td>wait</td>
<td>read data: 9</td>
</tr>
<tr>
<td>read global database</td>
<td>wait</td>
<td>read global database</td>
</tr>
<tr>
<td>insert 7 into database</td>
<td>wait</td>
<td>insert 9 into database</td>
</tr>
<tr>
<td>release db_semaphore: ok</td>
<td>release db_semaphore: ok</td>
<td>release db_semaphore: ok</td>
</tr>
</tbody>
</table>

```python
db_semaphore = Semaphore(2)

def insert(data):
    db_semaphore.acquire()
    database.insert(data)
    db_semaphore.release()
```
Example: database

db_semaphore = Semaphore(2)

def insert(data):
    db_semaphore.acquire()
    database.insert(data)
    db_semaphore.release()

<table>
<thead>
<tr>
<th>insert(7)</th>
<th>insert(8)</th>
<th>insert(9)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P1</strong></td>
<td><strong>P2</strong></td>
<td><strong>P3</strong></td>
</tr>
<tr>
<td>acquire db_semaphore: ok</td>
<td>acquire db_semaphore: wait</td>
<td>acquire db_semaphore: ok</td>
</tr>
<tr>
<td>read data: 7</td>
<td>wait</td>
<td>read data: 9</td>
</tr>
<tr>
<td>read global database</td>
<td>wait</td>
<td>read global database</td>
</tr>
<tr>
<td>insert 7 into database</td>
<td>wait</td>
<td>insert 9 into database</td>
</tr>
<tr>
<td>release db_semaphore: ok</td>
<td></td>
<td>release db_semaphore: ok</td>
</tr>
</tbody>
</table>
Conditions

Conditions are signals used to coordinate multiple processes.

Processes can `wait()` on a condition.

Other processes can `notify()` processes waiting for a condition.
Conditions example: vector mathematics

\[ A = B + C \]
\[ V = M \times A \]
Conditions example: vector mathematics

\[ \text{step1\_finished} = 0 \]

\[ A = B+C \]

\[ V = M \times A \]
Conditions example: vector mathematics

\[ A = B + C \]
\[ V = M \times A \]

\text{step1\_finished} = 0
\text{start\_step2} = \text{Condition()}
Conditions example: vector mathematics

\[ A = B + C \]
\[ V = M \times A \]

```python
step1_finished = 0
start_step2 = Condition()

def do_step_1(index):
```

Friday, November 18, 2011
Conditions example: vector mathematics

\[ A = B + C \]
\[ V = M \times A \]

```
step1_finished = 0
start_step2 = Condition()

def do_step_1(index):
```
Conditions example: vector mathematics

```python
step1_finished = 0
start_step2 = Condition()

def do_step_1(index):
    start_step2.acquire()
```

\[ A = B + C \]

\[ V = M \times A \]
Conditions example: vector mathematics

\[ A = B + C \]

\[ V = MxA \]

```
step1_finished = 0
start_step2 = Condition()

def do_step_1(index):
    start_step2.acquire()
    step1_finished += 1
```
Conditions example: vector mathematics

\[ A = B + C \]
\[ V = M \times A \]

\[
\text{step1\_finished} = 0 \\
\text{start\_step2} = \text{Condition}()
\]

\[
\text{def do\_step\_1(index):} \\
\text{start\_step2.acquire()} \\
\text{step1\_finished} += 1 \\
\text{if(step1\_finished == 2):}
\]
Conditions example: vector mathematics

\[ A = B + C \]
\[ V = M \times A \]

```python
step1_finished = 0
start_step2 = Condition()

def do_step_1(index):
    start_step2.acquire()
    step1_finished += 1
    if (step1_finished == 2):
        start_step2.notifyAll()
```

Friday, November 18, 2011
Conditions example: vector mathematics

\[ A = B + C \]
\[ V = M \times A \]

```python
step1_finished = 0
start_step2 = Condition()

def do_step_1(index):
    start_step2.acquire()
    step1_finished += 1
    if (step1_finished == 2):
        start_step2.notifyAll()
    start_step2.release()
```

Friday, November 18, 2011
Conditions example: vector mathematics

\[ A = B + C \]
\[ V = M \times A \]

```python
def do_step_1(index):
    start_step2.acquire()
    step1_finished += 1
    if(step1_finished == 2):
        start_step2.notifyAll()
    start_step2.release()

def do_step_2(index):
```

Friday, November 18, 2011
Conditions example: vector mathematics

\[ A = B + C \]
\[ V = MxA \]

```
step1_finished = 0
start_step2 = Condition()

def do_step_1(index):
    start_step2.acquire()
    step1_finished += 1
    if(step1_finished == 2):
        start_step2.notifyAll()
    start_step2.release()

def do_step_2(index):
    start_step2.wait()
```
Conditions example: vector mathematics

\[ A = B + C \]

\[ V = M \times A \]

```python
step1_finished = 0
start_step2 = Condition()

def do_step_1(index):
    start_step2.acquire()
    step1_finished += 1
    if(step1_finished == 2):
        start_step2.notifyAll()
    start_step2.release()

def do_step_2(index):
    start_step2.wait()
    V[index] = M[index] . A
```
Conditions example: vector mathematics

\[
\begin{align*}
\text{step1\_finished} &= 0 \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}
\end{align*}
\]

\text{start\_step2} = \text{Condition}()
Conditions example: vector mathematics

\[
\begin{align*}
\text{step1\_finished=} & 0 & B = \begin{pmatrix} 2 \\ 0 \end{pmatrix} & C = \begin{pmatrix} 0 \\ 5 \end{pmatrix} & M = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \\
\text{start\_step2} & = \text{Condition}() \\
A_1 & = B_1 + C_1 & A_2 & = B_2 + C_2 \\
V_1 & = M_1 \cdot A & V_2 & = M_2 \cdot A
\end{align*}
\]
Conditions example: vector mathematics

\[
\begin{align*}
\text{step1\_finished} &= 0 \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \quad C = \begin{pmatrix} 0 \\ 5 \end{pmatrix}, \quad M = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \\
\text{start\_step2} &= \text{Condition()}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 . A
\end{align*}
\]

\[
\begin{align*}
A_2 &= B_2 + C_2 \\
V_2 &= M_2 . A
\end{align*}
\]
Conditions example: vector mathematics

\[
\begin{align*}
\text{step1\_finished} &= 0 \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{start\_step2} &= \text{Condition()}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A
\end{align*}
\]

\[
\begin{align*}
A_2 &= B_2 + C_2 \\
V_2 &= M_2 \cdot A
\end{align*}
\]

\[
\begin{align*}
P_1 \\
\text{read B1: 2}
\end{align*}
\]

\[
\begin{align*}
P_2
\end{align*}
\]
Conditions example: vector mathematics

\[
\begin{align*}
\text{step1\_finished} &= 0 \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}
\end{align*}
\]

\[
\text{start\_step2} = \text{Condition}()
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1.A
\end{align*}
\]

\[
\begin{align*}
A_2 &= B_2 + C_2 \\
V_2 &= M_2.A
\end{align*}
\]

P1
read B1: 2
read C1: 0

P2
Conditions example: vector mathematics

\[ \begin{align*}
\text{step1\_finished} &= 0 \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}
\end{align*} \]

\[ \text{start\_step2} = \text{Condition()} \]

\[ \begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A \\
A_2 &= B_2 + C_2 \\
V_2 &= M_2 \cdot A
\end{align*} \]

\[ \begin{align*}
\text{P1} \\
&\text{read } B_1: 2 \\
&\text{read } C_1: 0 \\
&\text{calculate } 2 + 0: 2
\end{align*} \]

\[ \begin{align*}
\text{P2}
\end{align*} \]
Conditions example: vector mathematics

\[
\begin{align*}
\text{step1\_finished}=0 & \quad B = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 \\ 5 \end{pmatrix} \quad M = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \\
\text{start\_step2} = \text{Condition}() & \\
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A \\
A_2 &= B_2 + C_2 \\
V_2 &= M_2 \cdot A
\end{align*}
\]

\begin{align*}
P1 & \\
\text{read B1: 2} & \\
\text{read C1: 0} & \\
\text{calculate 2+0: 2} & \\
\text{write 2 -> A1} & \\
P2 & \\
\text{read B2: 0} &
\end{align*}
### Conditions example: vector mathematics

<table>
<thead>
<tr>
<th>Step 1 Conditions</th>
<th>Step 2 Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{step1_finished} = 0 )</td>
<td>( \text{start}_\text{step2} = \text{Condition}() )</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{A}_1 &= \text{B}_1 + \text{C}_1 \\
\text{V}_1 &= \text{M}_1 \cdot \text{A} \\
\text{P}1 \\
\text{read } \text{B}1: 2 \\
\text{read } \text{C}1: 0 \\
\text{calculate } 2 + 0: 2 \\
\text{write } 2 \rightarrow \text{A}1 \\
\end{align*}
\]

\[
\begin{align*}
\text{A}_2 &= \text{B}_2 + \text{C}_2 \\
\text{V}_2 &= \text{M}_2 \cdot \text{A} \\
\text{P}2 \\
\text{read } \text{B}2: 0 \\
\end{align*}
\]

\[
\begin{align*}
\text{B} &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
\text{C} &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
\text{M} &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \\
\text{A} &= \begin{pmatrix} 2 \\ 0 \end{pmatrix}
\end{align*}
\]
Conditions example: vector mathematics

\[ \text{step1}_{\text{finished}}=0 \quad \text{B=} \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad \text{C=} \begin{pmatrix} 0 \\ 5 \end{pmatrix} \quad \text{M=} \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \quad \text{A=} \begin{pmatrix} 2 \\ 0 \end{pmatrix} \]

\text{start}_{\text{step2}} = \text{Condition()}

\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A
\end{align*}

\begin{align*}
A_2 &= B_2 + C_2 \\
V_2 &= M_2 \cdot A
\end{align*}

\begin{align*}
P1 \\
\text{read } B_1: 2 \\
\text{read } C_1: 0 \\
\text{calculate } 2 + 0: 2 \\
\text{write } 2 \rightarrow A_1 \\
\text{acquire start}_{\text{step2}}: \text{ok}
\end{align*}

\begin{align*}
P2 \\
\text{read } B_2: 0 \\
\text{read } C_2: 0
\end{align*}
Conditions example: vector mathematics

\[
\begin{align*}
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix}, & C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix}, & M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}, & A &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
\text{step1\_finished} &= 0 \\
\text{start\_step2} &= \text{Condition}() \\
A_1 &= B_1 + C_1, & V_1 &= M_1 \cdot A \\
A_2 &= B_2 + C_2, & V_2 &= M_2 \cdot A
\end{align*}
\]

P1
read B1: 2
read C1: 0
calculate 2+0: 2
write 2 -> A1
acquire start\_step2: ok
write 1 -> step1\_finished

P2
read B2: 0
read C2: 0
calculate 5+0: 5
Conditions example: vector mathematics

\[
\begin{bmatrix}
    2 \\
    0
\end{bmatrix}
\begin{bmatrix}
    0 \\
    5
\end{bmatrix}
\begin{bmatrix}
    1 & 2 \\
    1 & 2
\end{bmatrix}
\begin{bmatrix}
    2 \\
    0
\end{bmatrix}
\]

\text{start\_step2 = Condition()}

\[
A_1 = B_1 + C_1 \\
V_1 = M_1 \cdot A
\]

\[
A_2 = B_2 + C_2 \\
V_2 = M_2 \cdot A
\]

\text{P1}
read B1: 2
read C1: 0
calculate 2+0: 2
write 2 -> A1
acquire start\_step2: ok
write 1 -> step1\_finished

\text{P2}
read B2: 0
read C2: 0
calculate 5+0: 5
Conditions example: vector mathematics

\[
\begin{align*}
\text{step1\_finished} &= 1 \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \\
A &= \begin{pmatrix} 2 \\ 0 \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A
\end{align*}
\]

P1

read B1: 2
read C1: 0
calculate 2 + 0: 2
write 2 -> A1
acquire start\_step2: ok
write 1 -> step1\_finished
step1\_finished == 2: false

P2

read B2: 0
read C2: 0
calculate 5 + 0: 5
write 5 -> A2
Conditions example: vector mathematics

\[
\begin{align*}
\text{step1\_finished} &= 1 \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \\
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix}
\end{align*}
\]

\text{start\_step2} = \text{Condition()}

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A
\end{align*}
\]

\[
\begin{align*}
A_2 &= B_2 + C_2 \\
V_2 &= M_2 \cdot A
\end{align*}
\]

\text{P1}

read B1: 2
read C1: 0
calculate 2 + 0: 2
write 2 \rightarrow A1
acquire start\_step2: ok
write 1 \rightarrow \text{step1\_finished}
step1\_finished == 2: false

\text{P2}

read B2: 0
read C2: 0
calculate 5 + 0: 5
write 5 \rightarrow A2
Conditions example: vector mathematics

\[
\begin{align*}
\text{start\_step2} &= \text{Condition()} \\
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A \\
\end{align*}
\]

\[
\begin{align*}
A_2 &= B_2 + C_2 \\
V_2 &= M_2 \cdot A \\
\end{align*}
\]

\[
\begin{align*}
P_1 & \quad \text{read } B_1: 2 \\
& \quad \text{read } C_1: 0 \\
& \quad \text{calculate } 2 + 0: 2 \\
& \quad \text{write } 2 \rightarrow A_1 \\
& \quad \text{acquire } \text{start\_step2}: \text{ok} \\
& \quad \text{write } 1 \rightarrow \text{step1\_finished} \\
& \quad \text{step1\_finished} = 2: \text{false} \\
& \quad \text{release } \text{start\_step2}: \text{ok} \\
\end{align*}
\]

\[
\begin{align*}
P_2 & \quad \text{read } B_2: 0 \\
& \quad \text{read } C_2: 0 \\
& \quad \text{calculate } 5 + 0: 5 \\
& \quad \text{write } 5 \rightarrow A_2 \\
& \quad \text{acquire } \text{start\_step2}: \text{ok} \\
\end{align*}
\]
Conditions example: vector mathematics

\[
\begin{align*}
\text{step1\_finished} &= 1 \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \\
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix}
\end{align*}
\]

\[\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A
\end{align*}\]

\[\begin{align*}
A_2 &= B_2 + C_2 \\
V_2 &= M_2 \cdot A
\end{align*}\]

**P1**
- read B1: 2
- read C1: 0
- calculate 2 + 0: 2
- write 2 -> A1
- acquire start\_step2: ok
- write 1 -> step1\_finished
- step1\_finished == 2: false
- release start\_step2: ok
- start\_step2: wait

**P2**
- read B2: 0
- read C2: 0
- calculate 5 + 0: 5
- write 5 -> A2
- acquire start\_step2: ok
- write 2 -> step1\_finished
Conditions example: vector mathematics

\[
\begin{align*}
\text{step1\_finished} &= 2 \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \quad C = \begin{pmatrix} 0 \\ 5 \end{pmatrix}, \quad M = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}, \quad A = \begin{pmatrix} 2 \\ 5 \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A
\end{align*}
\]

\[
\begin{align*}
A_2 &= B_2 + C_2 \\
V_2 &= M_2 \cdot A
\end{align*}
\]

P1
\[
\begin{align*}
\text{read } B_1: 2 \\
\text{read } C_1: 0 \\
\text{calculate } 2 + 0: 2 \\
\text{write } 2 \to A_1 \\
\text{acquire } \text{start\_step2}: \text{ok} \\
\text{write } 1 \to \text{step1\_finished} \\
\text{step1\_finished} == 2: \text{false} \\
\text{release } \text{start\_step2}: \text{ok} \\
\text{start\_step2}: \text{wait}
\end{align*}
\]

P2
\[
\begin{align*}
\text{read } B_2: 0 \\
\text{read } C_2: 0 \\
\text{calculate } 5 + 0: 5 \\
\text{write } 5 \to A_2 \\
\text{acquire } \text{start\_step2}: \text{ok} \\
\text{write } 2 \to \text{step1\_finished}
\end{align*}
\]
Conditions example: vector mathematics

\[
\begin{align*}
\text{step1\_finished} &= 2 \\
\begin{bmatrix} 2 \\ 0 \end{bmatrix} & C = \begin{bmatrix} 0 \\ 5 \end{bmatrix} \\
\begin{bmatrix} 1 & 2 \end{bmatrix} & M = \begin{bmatrix} 1 & 2 \end{bmatrix} \\
\begin{bmatrix} 2 \\ 5 \end{bmatrix} & A = \begin{bmatrix} 2 \\ 5 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 A
\end{align*}
\]

\[
\begin{align*}
A_2 &= B_2 + C_2 \\
V_2 &= M_2 A
\end{align*}
\]

\[
\begin{align*}
P_1 \\
\text{read } B_1: & 2 \\
\text{read } C_1: & 0 \\
\text{calculate } 2 + 0: & 2 \\
\text{write } 2 \rightarrow A_1 \\
\text{acquire } \text{start\_step2}: & \text{ok} \\
\text{write } 1 \rightarrow \text{step1\_finished} \\
\text{step1\_finished} == 2: & \text{false} \\
\text{release } \text{start\_step2}: & \text{ok} \\
\text{start\_step2}: & \text{wait} \\
\text{start\_step2}: & \text{wait}
\end{align*}
\]

\[
\begin{align*}
P_2 \\
\text{read } B_2: & 0 \\
\text{read } C_2: & 0 \\
\text{calculate } 5 + 0: & 5 \\
\text{write } 5 \rightarrow A_2 \\
\text{acquire } \text{start\_step2}: & \text{ok} \\
\text{write } 2 \rightarrow \text{step1\_finished} \\
\text{step1\_finished} == 2: & \text{true}
\end{align*}
\]
Conditions example: vector mathematics

\[
\begin{bmatrix}
2 & 0 & 1 \\
0 & 5 & 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 \\
5 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 \\
2 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 \\
5 \\
\end{bmatrix}
\]

\[
A_1 = B_1 + C_1 \\
V_1 = M_1.A
\]

\[
A_2 = B_2 + C_2 \\
V_2 = M_2.A
\]

P1
- read B1: 2
- read C1: 0
- calculate \(2 + 0\): 2
- write 2 -> A1
- acquire start_step2: ok
- write 1 -> step1_finished
- step1_finished == 2: false
- release start_step2: ok
- start_step2: wait
- start_step2: wait
- start_step_2: wait

P2
- read B2: 0
- read C2: 0
- calculate \(5 + 0\): 5
- write 5 -> A2
- acquire start_step2: ok
- write 2 -> step1_finished
- step1_finished == 2: true
- notifyAll start_step_2: ok
- start_step_2: wait
Conditions example: vector mathematics

\[
\begin{align*}
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix}, \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}, \\
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A
\end{align*}
\]

\[
\begin{align*}
A_2 &= B_2 + C_2 \\
V_2 &= M_2 \cdot A
\end{align*}
\]

\[
\begin{align*}
P1 & \\
\text{read } B_1: 2 & \\
\text{read } C_1: 0 & \\
\text{calculate } 2 + 0: 2 & \\
\text{write } 2 \rightarrow A_1 & \\
\text{acquire start_step2: ok} & \\
\text{write } 1 \rightarrow \text{step1_finished} & \\
\text{step1_finished} == 2: \text{false} & \\
\text{release start_step2: ok} & \\
\text{start_step2: wait} & \\
\text{start_step2: wait} & \\
\text{start_step2: wait} & \\
\text{read } M_1: (1 \ 2) & \\
\end{align*}
\]

\[
\begin{align*}
P2 & \\
\text{read } B_2: 0 & \\
\text{read } C_2: 0 & \\
\text{calculate } 5 + 0: 5 & \\
\text{write } 5 \rightarrow A_2 & \\
\text{acquire start_step2: ok} & \\
\text{write } 2 \rightarrow \text{step1_finished} & \\
\text{step1_finished} == 2: \text{true} & \\
\text{notifyAll start_step_2: ok} & \\
\end{align*}
\]
### Conditions example: vector mathematics

<table>
<thead>
<tr>
<th>[ \text{step1_finished} = 2 ]</th>
<th>[ \text{B} = \begin{pmatrix} 2 \ 0 \end{pmatrix}, \text{C} = \begin{pmatrix} 0 \ 5 \end{pmatrix}, \text{M} = \begin{pmatrix} 1 &amp; 2 \ 1 &amp; 2 \end{pmatrix}, \text{A} = \begin{pmatrix} 2 \ 5 \end{pmatrix} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \text{start_step2} = \text{Condition}() ]</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{A}_1 = \text{B}_1 + \text{C}_1 \]
\[ \text{V}_1 = \text{M}_1 \cdot \text{A} \]

\[ \text{P1} \]
read \( \text{B}_1 \): 2
read \( \text{C}_1 \): 0
calculate \( 2 + 0 \): 2
write 2 -> \( \text{A}_1 \)
acquire \( \text{start\_step2} \): ok
write 1 -> \text{step1\_finished}
step1\_finished == 2: false
release \( \text{start\_step2} \): ok
\[ \text{P2} \]
read \( \text{B}_2 \): 0
read \( \text{C}_2 \): 0
write 5-> \( \text{A}_2 \)
acquire \( \text{start\_step2} \): ok
write 2-> \text{step1\_finished}
step1\_finished == 2: true
notifyAll \( \text{start\_step2} \): ok
read \( \text{M}_1 \): (1 2)
read \( \text{A} \): (2 5)
read \( \text{M}_2 \): (1 2)
Conditions example: vector mathematics

\[
\begin{align*}
\text{step1\_finished} &= 2 \\
B &= \begin{pmatrix} 2 \\ 0 \end{pmatrix} \\
C &= \begin{pmatrix} 0 \\ 5 \end{pmatrix} \\
M &= \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \\
A &= \begin{pmatrix} 2 \\ 5 \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
A_1 &= B_1 + C_1 \\
V_1 &= M_1 \cdot A
\end{align*}
\]

\[
\begin{align*}
A_2 &= B_2 + C_2 \\
V_2 &= M_2 \cdot A
\end{align*}
\]

\[
\begin{align*}
P_1 \\
&\text{read } B_1: 2 \\
&\text{read } C_1: 0 \\
&\text{calculate } 2 + 0: 2 \\
&\text{write } 2 \rightarrow A_1 \\
&\text{acquire start\_step2: ok} \\
&\text{write } 1 \rightarrow \text{step1\_finished} \\
&\text{step1\_finished} == 2: \text{false} \\
&\text{release start\_step2: ok} \\
&\text{start\_step2: wait} \\
&\text{start\_step2: wait} \\
&\text{start\_step2: wait} \\
&\text{read } M_1: (1 \ 2) \\
&\text{read } A: (2 \ 5) \\
&\text{calculate } (1 \ 2) \cdot (2 \ 5): 12
\end{align*}
\]

\[
\begin{align*}
P_2 \\
&\text{read } B_2: 0 \\
&\text{read } C_2: 0 \\
&\text{calculate } 5 + 0: 5 \\
&\text{write } 5 \rightarrow A_2 \\
&\text{acquire start\_step2: ok} \\
&\text{write } 2 \rightarrow \text{step1\_finished} \\
&\text{step1\_finished} == 2: \text{true} \\
&\text{notifyAll start\_step2: ok} \\
&\text{read } M_2(1 \ 2) \\
&\text{read } A: (2 \ 5)
\end{align*}
\]
Deadlock

A condition in which threads are stuck waiting for each other forever
Deadlock example
Deadlock example

```python
>>> x_lock = Lock()
```
Deadlock example

```python
>>> x_lock = Lock()
>>> y_lock = Lock()
```
Deadlock example

```python
global x_lock, y_lock
x_lock = Lock()
y_lock = Lock()
x = 1
```
Deadlock example

```python
>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0
```
Deadlock example

```python
g>>> x_lock = Lock()
g>>> y_lock = Lock()
g>>> x = 1
g>>> y = 0
g>>> def compute():
```
Deadlock example

```python
global x_lock, y_lock
x_lock = Lock()
y_lock = Lock()
x = 1
y = 0
def compute():
    x_lock.acquire()
```

Friday, November 18, 2011
Deadlock example

```python
>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0
>>> def compute():
    x_lock.acquire()
    y_lock.acquire()
```
Deadlock example

```python
>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0
>>> def compute():
...     x_lock.acquire()
...     y_lock.acquire()
...     y = x + y
```
Deadlock example

```python
>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0
>>> def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
```
Deadlock example

```python
>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0
>>> def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
```
Deadlock example

```python
>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0
>>> def compute():
...     x_lock.acquire()
...     y_lock.acquire()
...     y = x + y
...     x = x * x
...     y_lock.release()
...     x_lock.release()
```
Deadlock example

```python
globals()['x_lock'] = Lock()
globals()['y_lock'] = Lock()
globals()['x'] = 1
globals()['y'] = 0

def compute():
    globals()['x_lock'].acquire()
    globals()['y_lock'].acquire()
    globals()['y'] = globals()['x'] + globals()['y']
    globals()['x'] = globals()['x'] * globals()['x']
    globals()['y_lock'].release()
    globals()['x_lock'].release()

def anti_compute():
```

Friday, November 18, 2011
Deadlock example

>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0
>>> def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()

>>> def anti_compute():
    y_lock.acquire()
Deadlock example

```python
>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0

```python
def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()
```python
>>> def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
```
Deadlock example

```python
>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0
>>> def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()

>>> def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
    x_lock.acquire()
    y = y - x
```

Friday, November 18, 2011
Deadlock example

```python
>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0
>>> def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()

>>> def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
    y = y - x
    y = sqrt(x)
```
Deadlock example

```python
>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0
>>> def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()

>>> def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
    y = y - x
    y = y - x
    x = sqrt(x)
    x_lock.release()
```
Deadlock example

```python
>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0

```python
def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()
```python
def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
    y = y - x
    x = sqrt(x)
    x_lock.release()
    y_lock.release()
```
Deadlock: example

def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()

def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
    y = y - x
    x = sqrt(x)
    x_lock.release()
    y_lock.release()
Deadlock: example

```python
def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()

def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
    y = y - x
    x = sqrt(x)
    x_lock.release()
    y_lock.release()
```

compute()  anti_compute()
Deadlock: example

def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()

def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
    y = y - x
    x = sqrt(x)
    x_lock.release()
    y_lock.release()
Deadlock: example

```python
def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()

def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
    y = y - x
    x = sqrt(x)
    x_lock.release()
    y_lock.release()
```

compute()

P1
acquire x_lock: ok

anti_compute()

P2
acquire y_lock: ok
Deadlock: example

```python
def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()

def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
    y = y - x
    x = sqrt(x)
    x_lock.release()
    y_lock.release()
```

```
compute()

P1
acquire x_lock: ok
acquire y_lock: wait
wait

anti_compute()

P2
acquire y_lock: ok
acquire x_lock:
wait
```
**Deadlock: example**

```python
def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()

def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
    y = y - x
    x = sqrt(x)
    x_lock.release()
    y_lock.release()
```

<table>
<thead>
<tr>
<th>compute()</th>
<th>anti_compute()</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>acquire x_lock: ok</td>
<td>acquire y_lock: ok</td>
</tr>
<tr>
<td>acquire y_lock: wait</td>
<td>acquire x_lock:</td>
</tr>
<tr>
<td>wait</td>
<td>wait</td>
</tr>
<tr>
<td>wait</td>
<td>wait</td>
</tr>
</tbody>
</table>

Friday, November 18, 2011
Deadlock: example

```python
def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()

compute()

P1
acquire x_lock: ok
acquire y_lock: wait
wait
wait
wait

P2
acquire y_lock: ok
acquire x_lock:
wait
wait
```

def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
    y = y - x
    x = sqrt(x)
    x_lock.release()
    y_lock.release()

anti_compute()
**Deadlock: example**

```python
def compute():
    x_lock.acquire()
    y_lock.acquire()
    y = x + y
    x = x * x
    y_lock.release()
    x_lock.release()

def anti_compute():
    y_lock.acquire()
    x_lock.acquire()
    y = y - x
    x = sqrt(x)
    x_lock.release()
    y_lock.release()
```

<table>
<thead>
<tr>
<th>compute()</th>
<th>anti_compute()</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>acquire x_lock: ok</td>
<td>acquire y_lock: ok</td>
</tr>
<tr>
<td>acquire y_lock: wait</td>
<td>acquire x_lock: wait</td>
</tr>
<tr>
<td>wait</td>
<td>wait</td>
</tr>
<tr>
<td>wait</td>
<td>wait</td>
</tr>
<tr>
<td>wait</td>
<td>wait</td>
</tr>
<tr>
<td>wait</td>
<td>wait</td>
</tr>
</tbody>
</table>
Deadlock: example

def compute():
    x_lock.acquire()
y_lock.acquire()
y = x + y
x = x * x
y_lock.release()
x_lock.release()

def anti_compute():
    y_lock.acquire()
x_lock.acquire()
y = y - x
x = sqrt(x)
x_lock.release()
y_lock.release()

compute()

P1
acquire x_lock: ok
acquire y_lock: wait
wait
wait
wait
wait
...

anti_compute()

P2
acquire y_lock: ok
acquire x_lock:
wait
wait
wait
wait
...

Deadlock
Next time

Sequences and Streams