ENGI 7894/9869

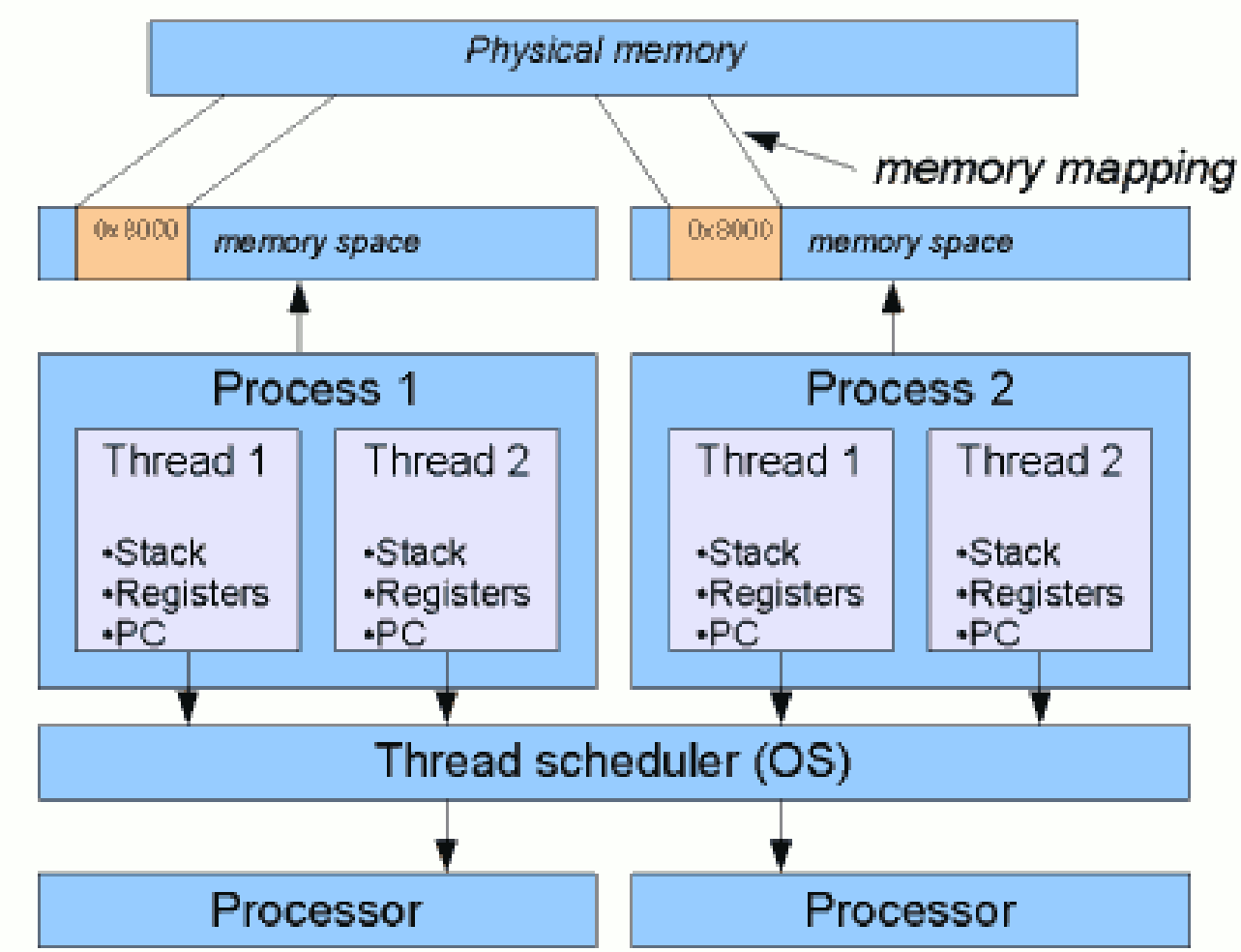
Thread, Process and Synchronization

Presented by
Md Monjur Ul Hasan

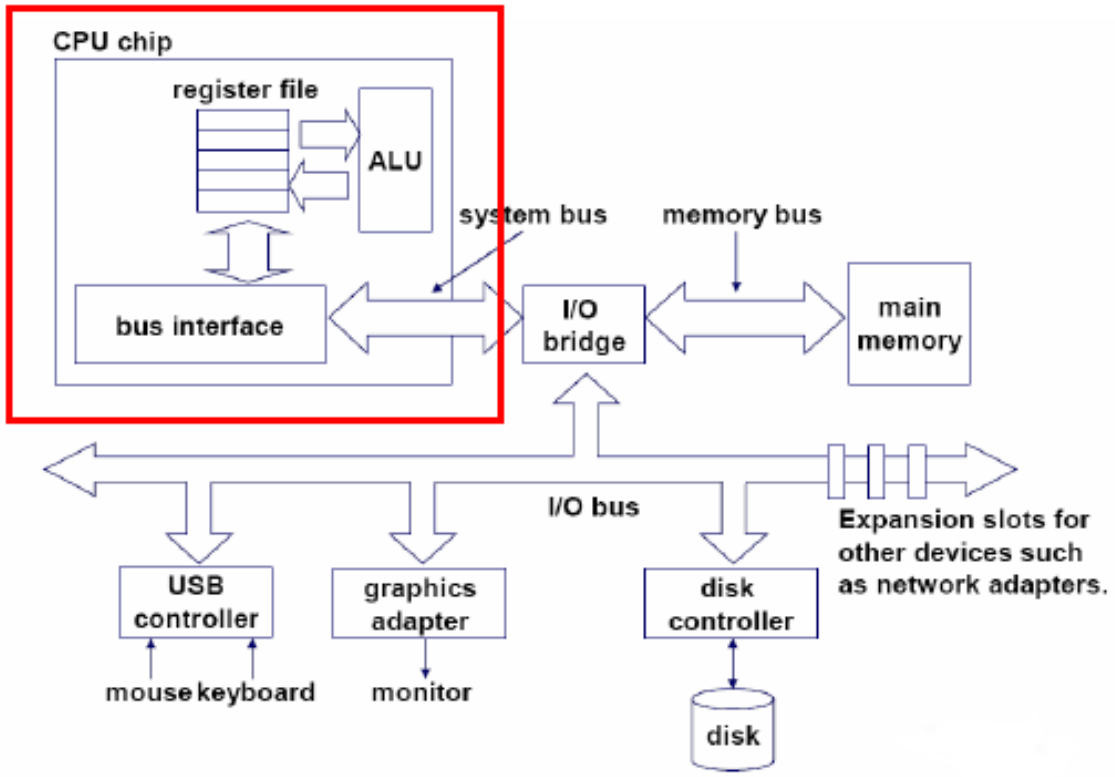
Topics

- Critical Section
- Locks (2 Threads)
  - Lock One
  - Lock Two
  - Peterson Lock

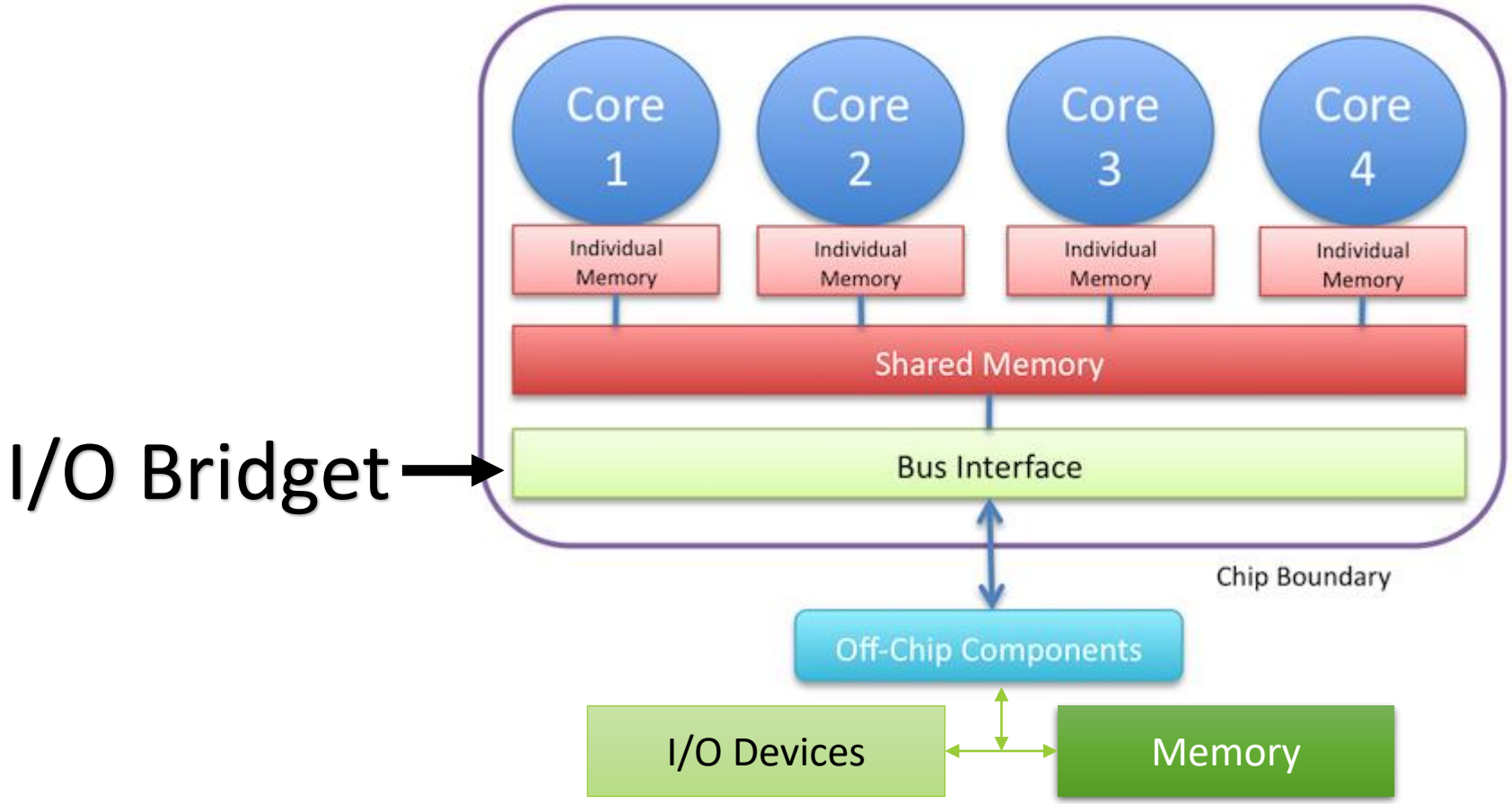
Process Vs Threads



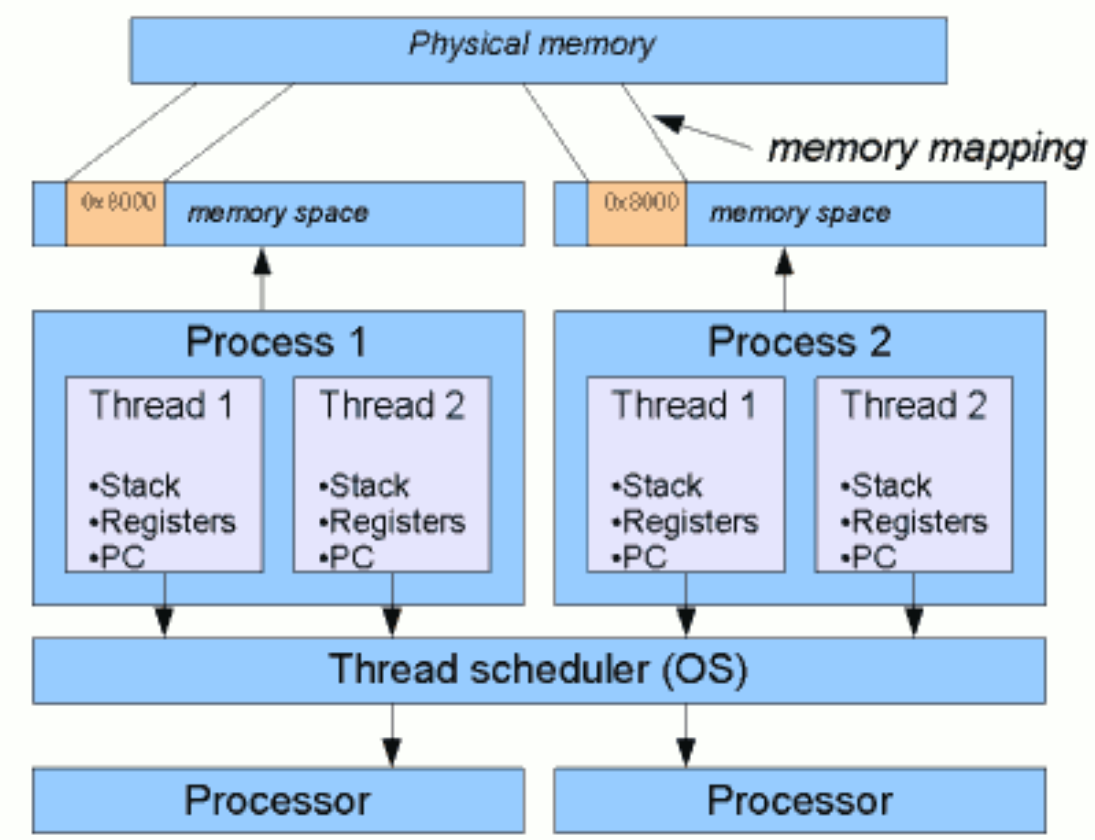
Single Core Architecture



Multicore Architecture



```
class bankAccount
{
        long amount = 1;
        public void add(long value)
        {
           this.amount += value;
        }
        public long get() {
           return this.amount;
        }
 }
```




```
class bankAccount
{
        long amount = 1;
        public void add(long value)
        {
                this.amount += value;
        }
        public long get() {
                return this.amount;
        }
 }
```

Amdahl's Law

$$S(N) = \frac{1}{(1-p) + \frac{p}{N}}$$

```
class bankAccount
{
        long amount = 1;
        public void add(long value)
        {
                this.amount += value;
        }
        public long get() {
                return this.amount;
        }
}
```

- Where,
  - S(N) is the speedup of the concurrent execution over the sequential execution
  - P is the percentage of the application that can be run in parallel
  - N is the number of processor (parallel thread) that can execute at a time

Question or Comments From Previous Class?

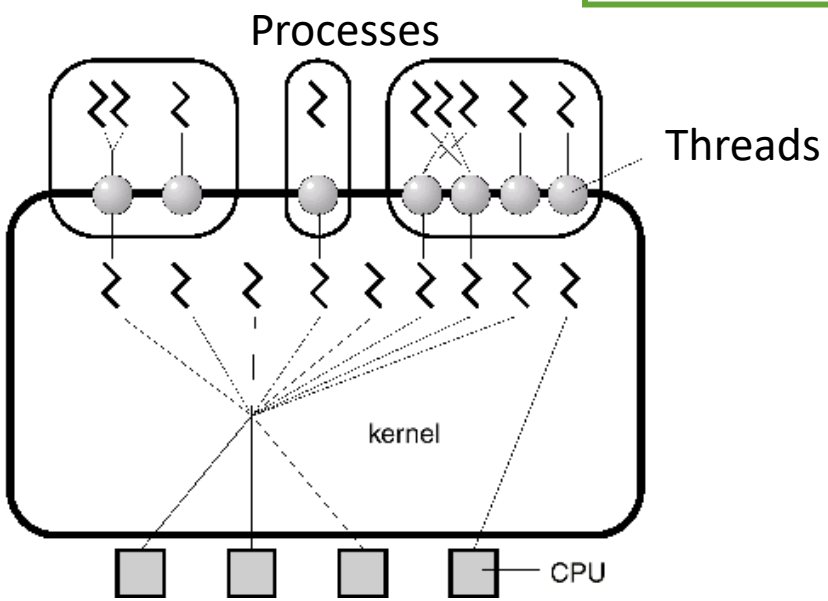
A Home (Analogy…)

Critical Section

Critical Section (CS) …

```
function readDB(String dbName,
Set<String> words){

    dbScanner input =
        new DBScanner(new DBFile(file));

    while (input.hasNext()) {
        String word = input.next();
        words.add(word);
    }
}
```

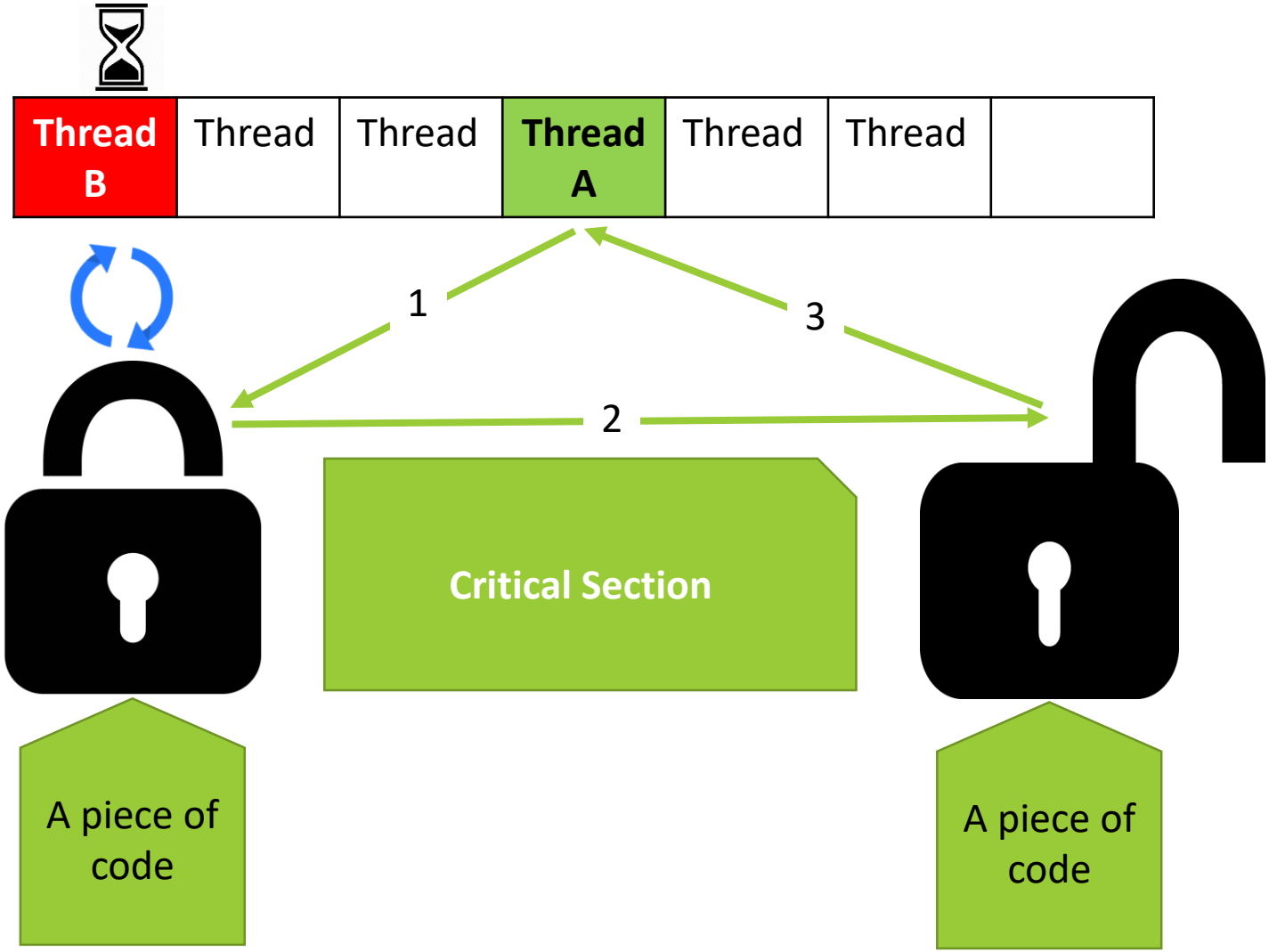
```
class bankAccount
{
        long amount = 1;
        public void add(long value)
        {
                this.amount += value;
        }
        public long get() {
                return this.amount;
        }
}
```



Is critical section just one statement?

```
a = x-y;
b = x + y;
x = x*x;
y = y*a;
```

Lock


Thread B
Thread
Thread
Thread A
Thread
Thread
1
2
3
Critical Section
A piece of code
A piece of code

Lock Application

```
public interface iLock {
        public void lock(); // before entering critical section
        public void unlock(); // after leaving critical section
}
```

```
iLock L;
String[] sites =  {"site1.db", "site2.db", … , "siten.db"};
Set<String> words = new HashSet<String>();
L =  new lock(); // class lock implements iLock {…}
for (String site: sites) {
    Thread t =  new Thread (readDB(site, words));
    t.start();
}

function readDB(String dbName, Set<String> words){

    dbScanner input =
        new DBScanner(new DBFile(file));

    while (input.hasNext()) {
        String word = input.next();
        L.lock();
         words.add(word);
        L.unlock();
    }
}
```

Lock Working Principle

Thread 1

```
dbScanner input =
   new DBScanner(new
DBFile(file));

    while (input.hasNext()) {
        String word =
             input.next();
        L.lock();
          words.add(word);
        L.unlock();
    }
```

Thread 2

```
dbScanner input =
   new DBScanner(new
DBFile(file));

    while (input.hasNext()) {
        String word =
             input.next();
        L.lock();
          words.add(word);
        L.unlock();
    }
```

1	T1::L.lock(); **<-Acquiring a lock**	
2	T1:: words.add(word); **<- CS**	T2::L.lock(); **<-Acquiring a lock**
3	T1::L.unlock(); **<-Releasing a lock**	*T2::Wait*
4		T2::L.lock(); **<-Acquire Success**
5		T2::words.add(word)
6		T2::L.unlock() **<-Releasing a lock**

Ensure Unlocking

```
dbScanner input =
   new DBScanner(new
DBFile(file));

    while (input.hasNext()) {
        String word =
              input.next();
        L.lock();
         words.add(word);
        L.unlock();
    }
```

```
dbScanner input =
   new DBScanner(new
DBFile(file));

    while (input.hasNext()) {
        String word =
              input.next();
        L.lock();
        try{
         words.add(word);
          …
        }
        finally{
          L.unlock();
        }
    }
```

Question or Comments?

How to implement a “Lock”

Implementation of Lock

Definition

- A way of ensuring the mutual exclusion for CS
- Also called Mutex.

Lock type

- **Spin Lock**
- Practical Lock

Lock Property

- Mutual Exclusion
- Freedom of deadlock
- Freedom of Starvation
- Fairness

Locking Type

- Coarse grain Locking
- Fine grain Locking
- …

Lock One

```
class LockOne implements iLock
{
	private boolean[] flag = new boolean[2];
	// thread-local index, 0 or 1
	public void lock()
	{
		int i = ThreadID.get();
		int j = 1 - i;
		flag[i] = true;
		while (flag[j]) {} // wait
	}

	public void unlock()
	{
		int i = ThreadID.get();
		flag[i] = false;
	}
}
```

Lock 1: Mutual Exclusion…

Thread 0

```
class LockOne implements iLock
{
   public void lock()
   {
         flag[0] = true; // i = 0; j = 1
         while (flag[1]) {} // wait
   }

   public void unlock()
   {
         flag[0] = false;
   }
}
```

Thread 1

```
class LockOne implements iLock
{
   public void lock()
   {
         flag[1] = true;// i=1;j=0
         while (flag[0]) {} // wait
   }

   public void unlock()
   {
         flag[1] = false;
   }
}
```

$CS_i \implies$ Critical Section *i executed*

$E_1 \rightarrow E_2 \implies$ Event E_1 Precedes event E_2 $\quad (E_1 \nrightarrow E_2)$

Lock 1 : Mutual Exclusion…

Thread 0

```
class LockOne implements iLock
{
    public void lock()
    {
         flag[0] = true;
         while (flag[1]) {} // wait
    }

    public void unlock()
    {
         flag[0] = false;
    }
}
```

Thread 1

```
class LockOne implements iLock
{
    public void lock()
    {
         flag[1] = true;
         while (flag[0]) {} // wait
    }

    public void unlock()
    {
         flag[1] = false;
    }
}
```

$$Write_0(flag[0] = true) \rightarrow Read_0(flag[1] == false) \rightarrow CS_0$$
$$Write_1(flag[1] = true) \rightarrow Read_1(flag[0] == false) \rightarrow CS_1$$

Contradiction: $CS_0 \nrightarrow CS_1$ and $CS_1 \nrightarrow CS_0$

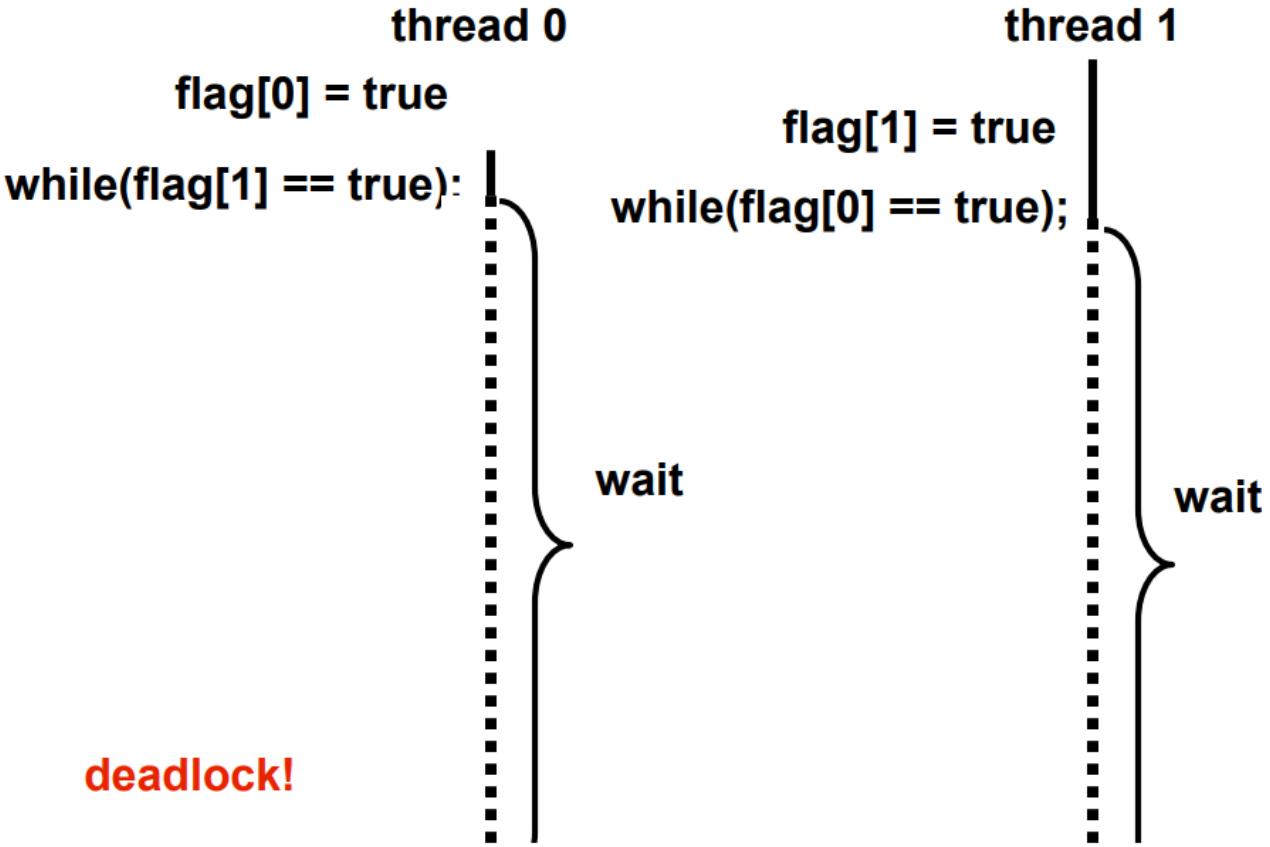
$$\boldsymbol{Read_0(flag[1] == false) \rightarrow write_1(flag[1] = true) \rightarrow CS_0}$$

$$\boxed{\boldsymbol{Read_1(flag[0] == false)}} \rightarrow write_0(flag[0] = true) \rightarrow CS_1$$

Question or Comments?

Lock one

Deadlock? Starvation?



Lock Two (Second Chance)

```
class LockTwo implements iLock
{
        private int flag victim

        public void lock()
        {
                int i = ThreadID.get();
                victim = i
                while (victim == i) {} // wait
        }

        public void unlock()
        {
        }
}
```

DIY: Proof the mutual exclusion?

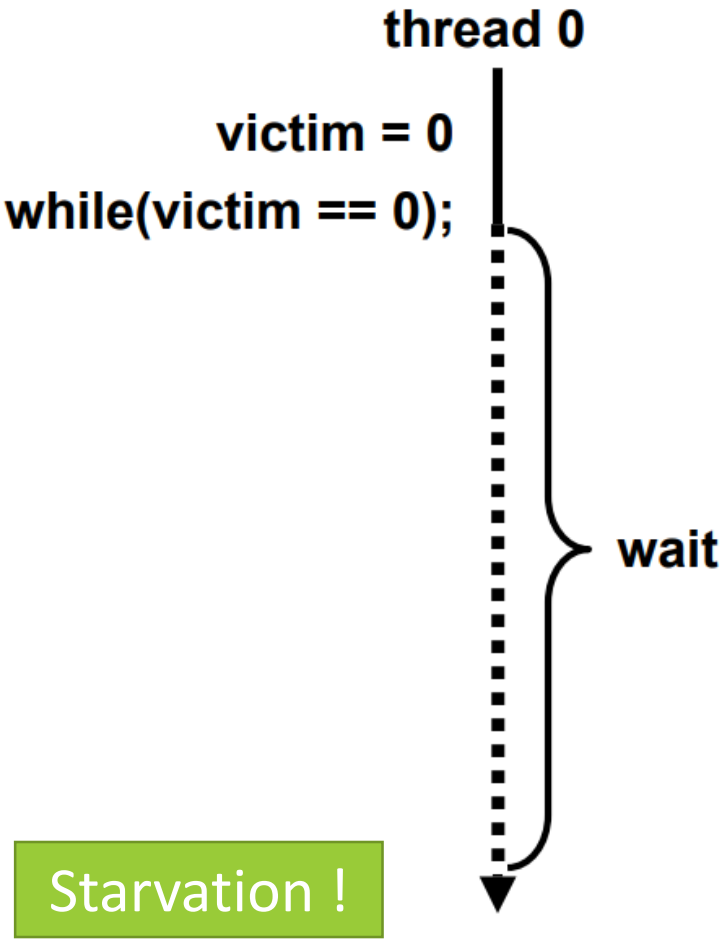
Lock Two

```
class LockTwo implements iLock
{
        private int flag victim

        public void lock()
        {
                int i = ThreadID.get();
                victim = i
                while (victim == i) {} // wait
        }

        public void unlock()
        {
        }
}
```

Deadlock free?

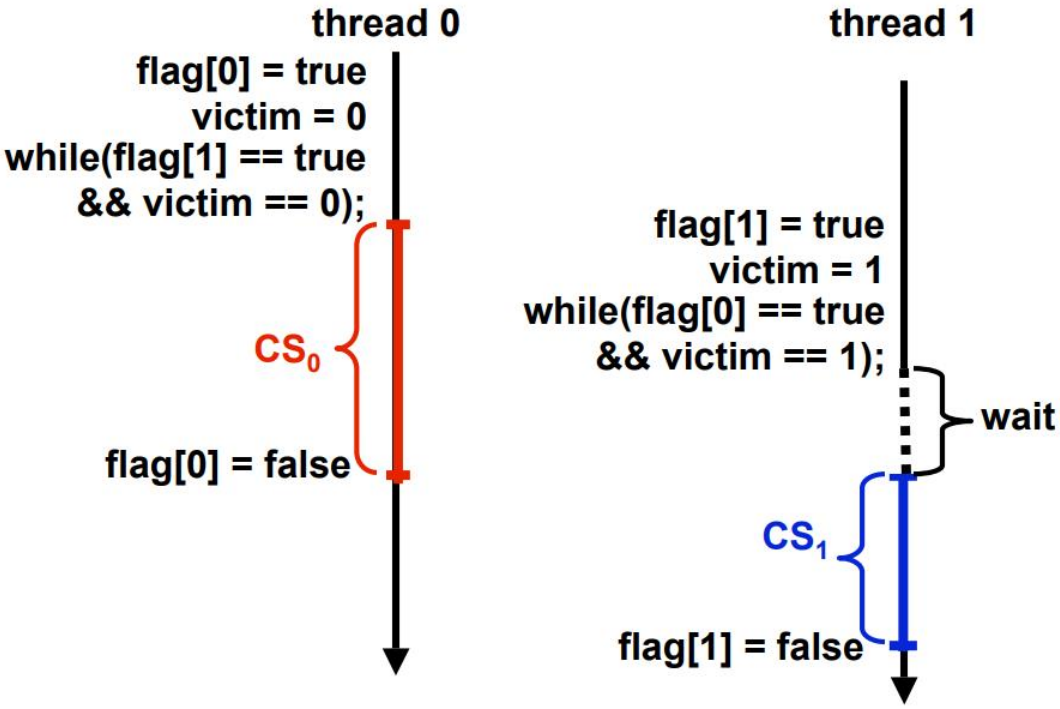
Starvation free?

Lock Two



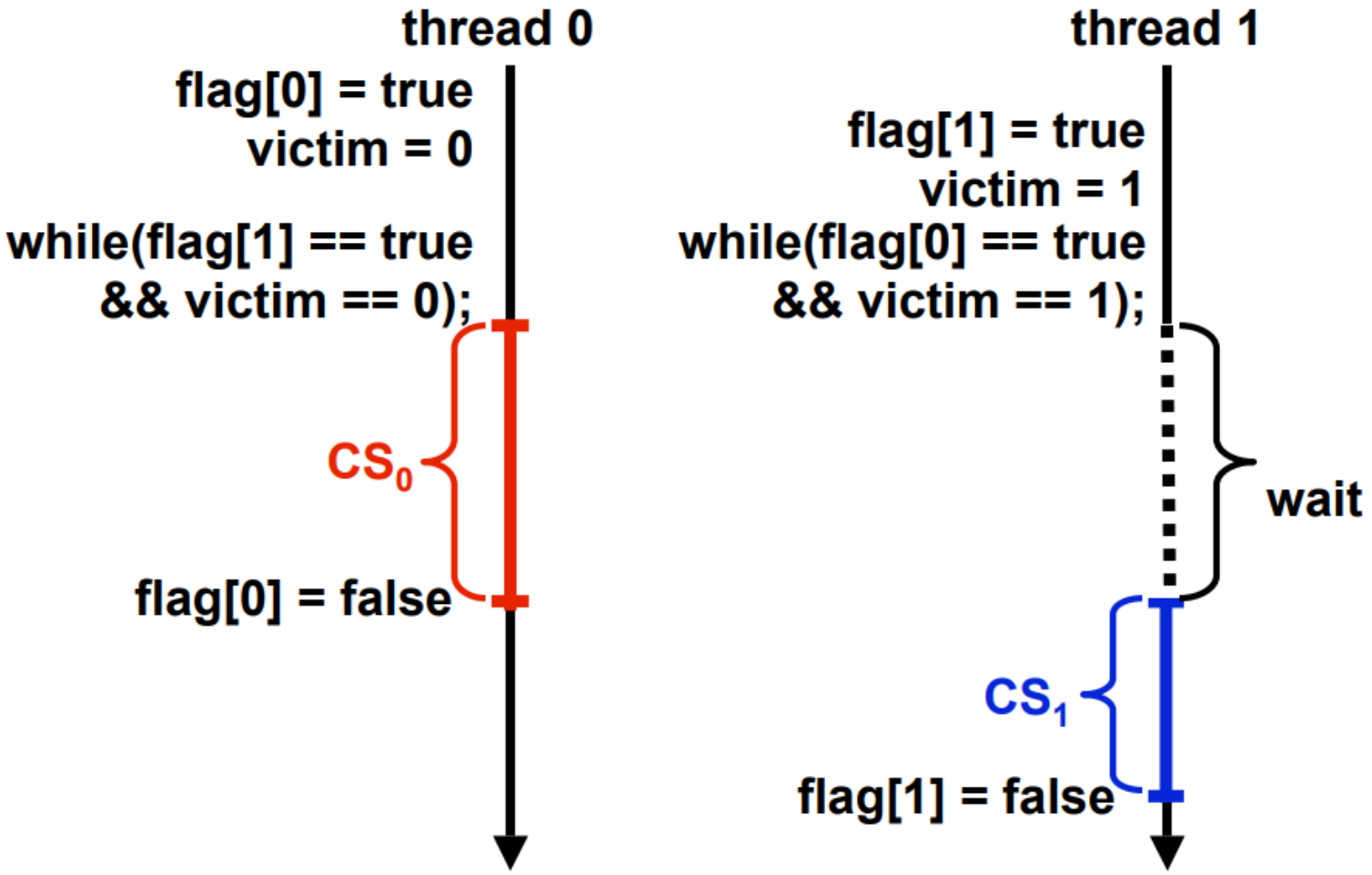
Peterson Lock

```
class Peterson implements iLock
{
      // thread-local index, 0 or 1
      private boolean[] flag = new boolean[2];
      private int victim;
      public void lock()
      {
                  int i = ThreadID.get();
                  int j = 1 - i;
                  flag[i] = true;
                  victim = i;
                  while (flag[j] && victim == i) {};
      }
      public void unlock()
      {
                  int i = ThreadID.get();
                  flag[i] = false;
      }
}
```



DIY: Proof the mutual exclusion?

Peterson Lock (Concurrent Acquires)



Paterson Lock: Freedom of Starvation

- **Proof by Contradiction**
  - **Wolog:** Thread 0 is not able to enter to CS
    - flag[1] == true && victim == 0
    - It means:
      - Either thread 1 repeatedly entering and exiting CS. Or
      - It stuck
    - If entering and exiting frequently:
      - it must set flag[1] when exit.
      - Set victim = 1 when re-entering
      - **Contradict**

```
class Peterson implements iLock
{
    // thread-local index, 0 or 1
    private boolean[] flag = new boolean[2];
    private int victim;
    public void lock()
    {
        int i = ThreadID.get();
        int j = 1 - i;
        flag[i] = true;
        victim = i;
        while (flag[j] && victim == i) {};
    }
    public void unlock()
    {
        int i = ThreadID.get();
        flag[i] = false;
    }
}
```

- **Freedom from Starvation => Deadlock Free**

Thank you for your attention

Any Questions?

Acknowledgement:
Vivek Sarkar, Research Professor, Department of Computer Science, RICE University

Critical Section (Cont.)

Definition	• Continuous Code • Atomic
Condition	• Mutual Exclusive
Good Implementation	• Allow maximum concurrency