3

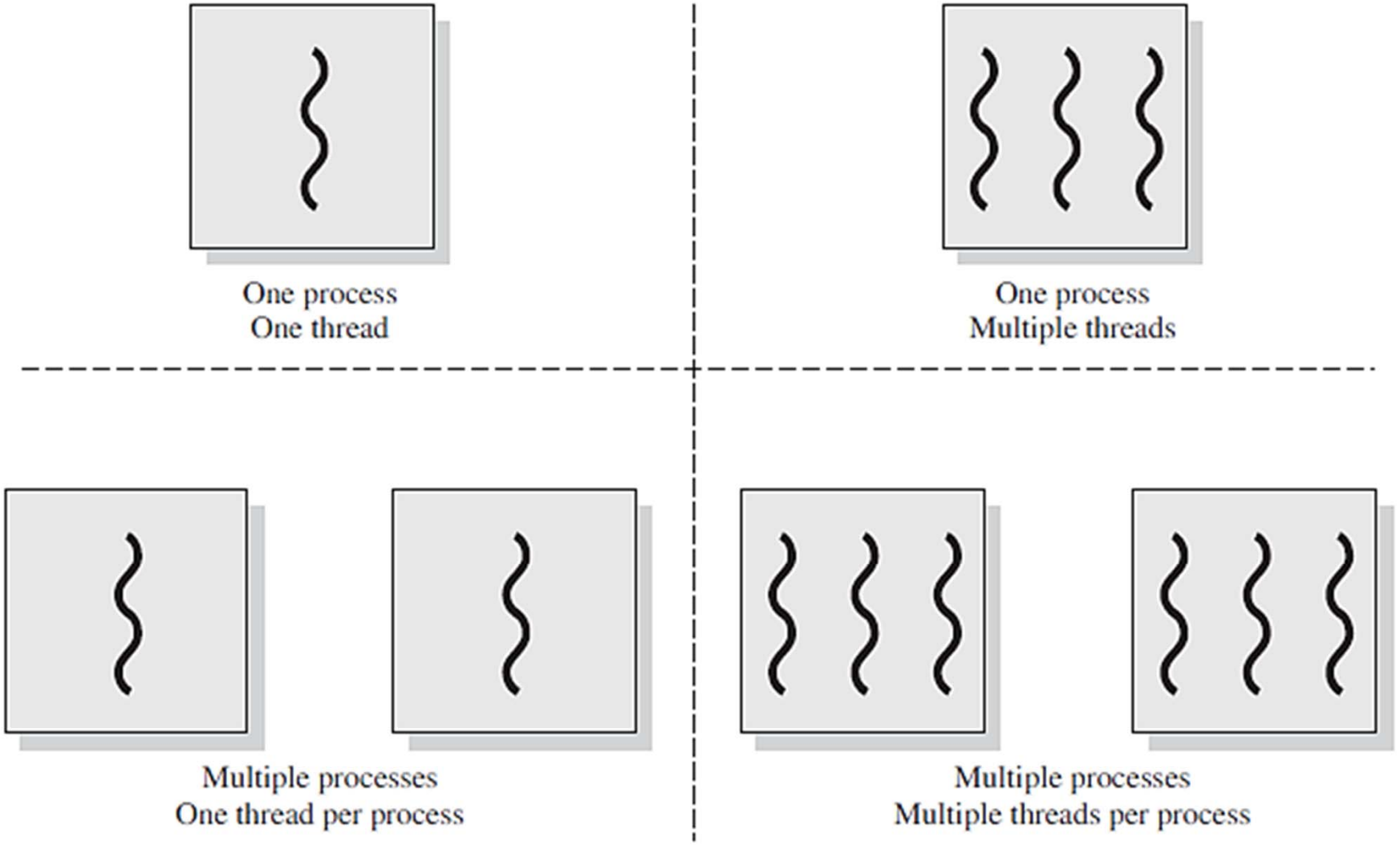
Chapter 3
Threads & Concurrency

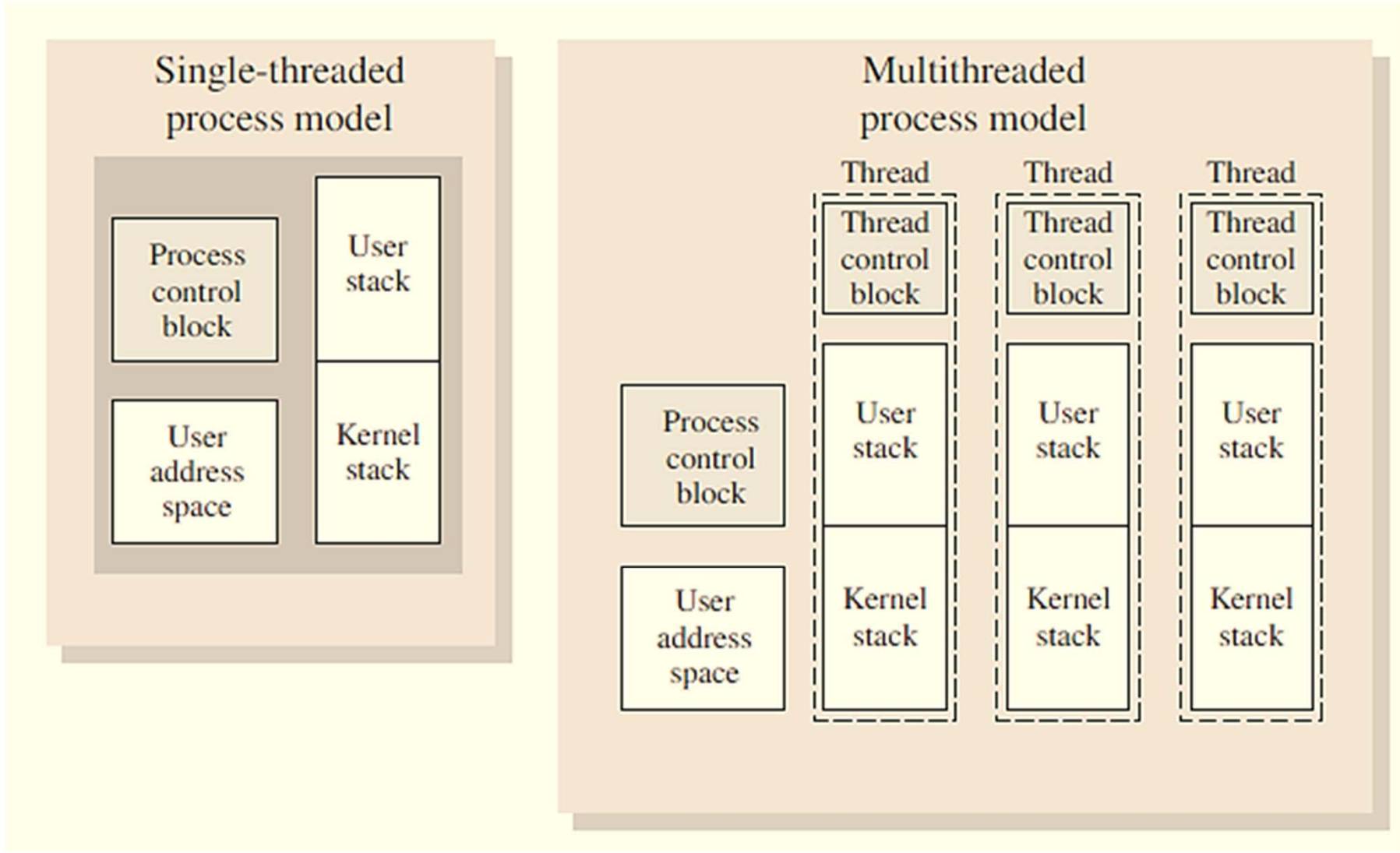
Asst.Prof.Dr. Supakit Nootyaskool
IT-KMITL

Topics

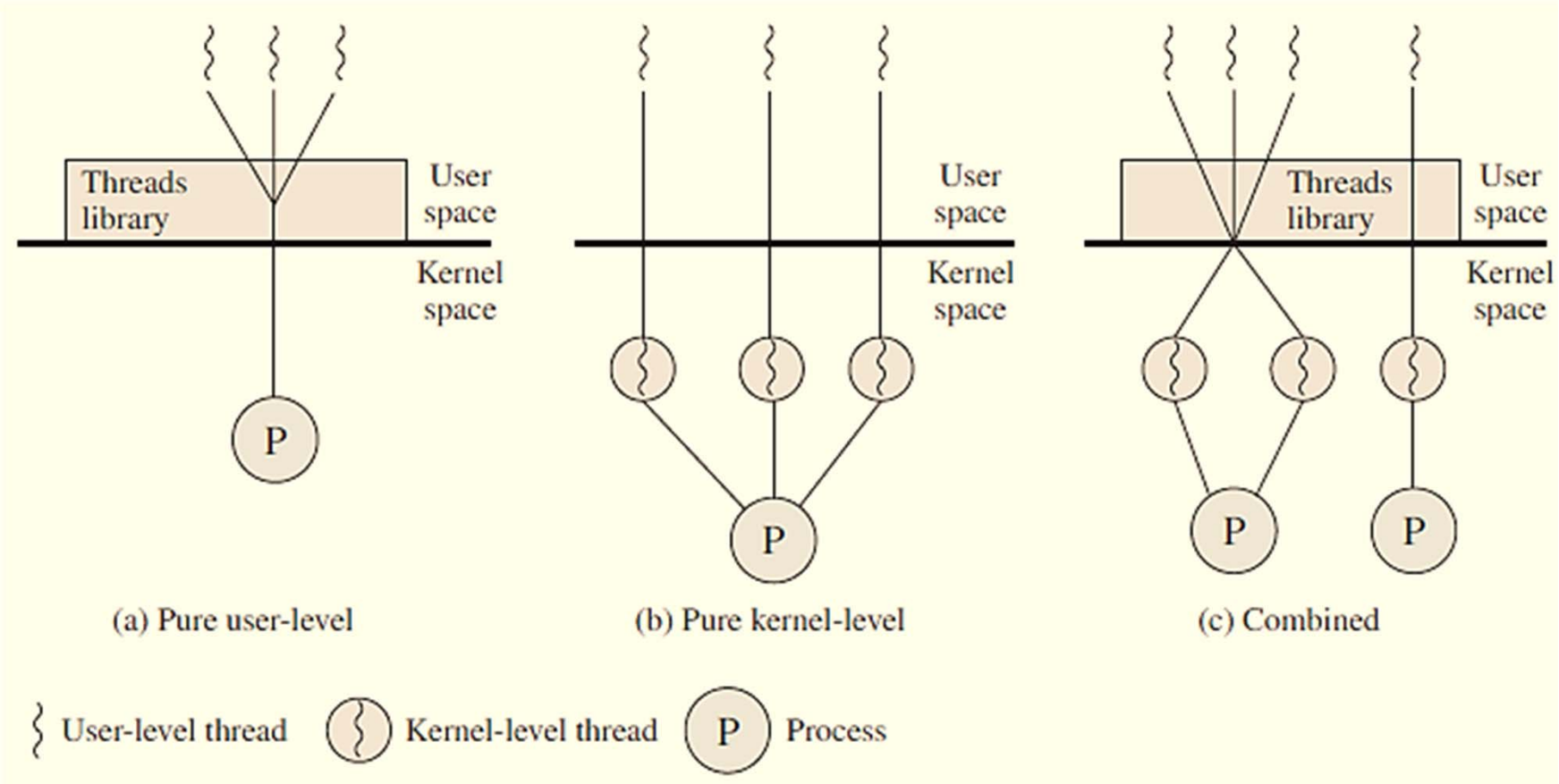
- Process and thread
- User-level thread and kernel-level thread
- Atomic operation
- Critical section
- Mutual exclusion
- Deadlock
- Livelock
- Race condition
- Starvation

3.1 Process and Thread

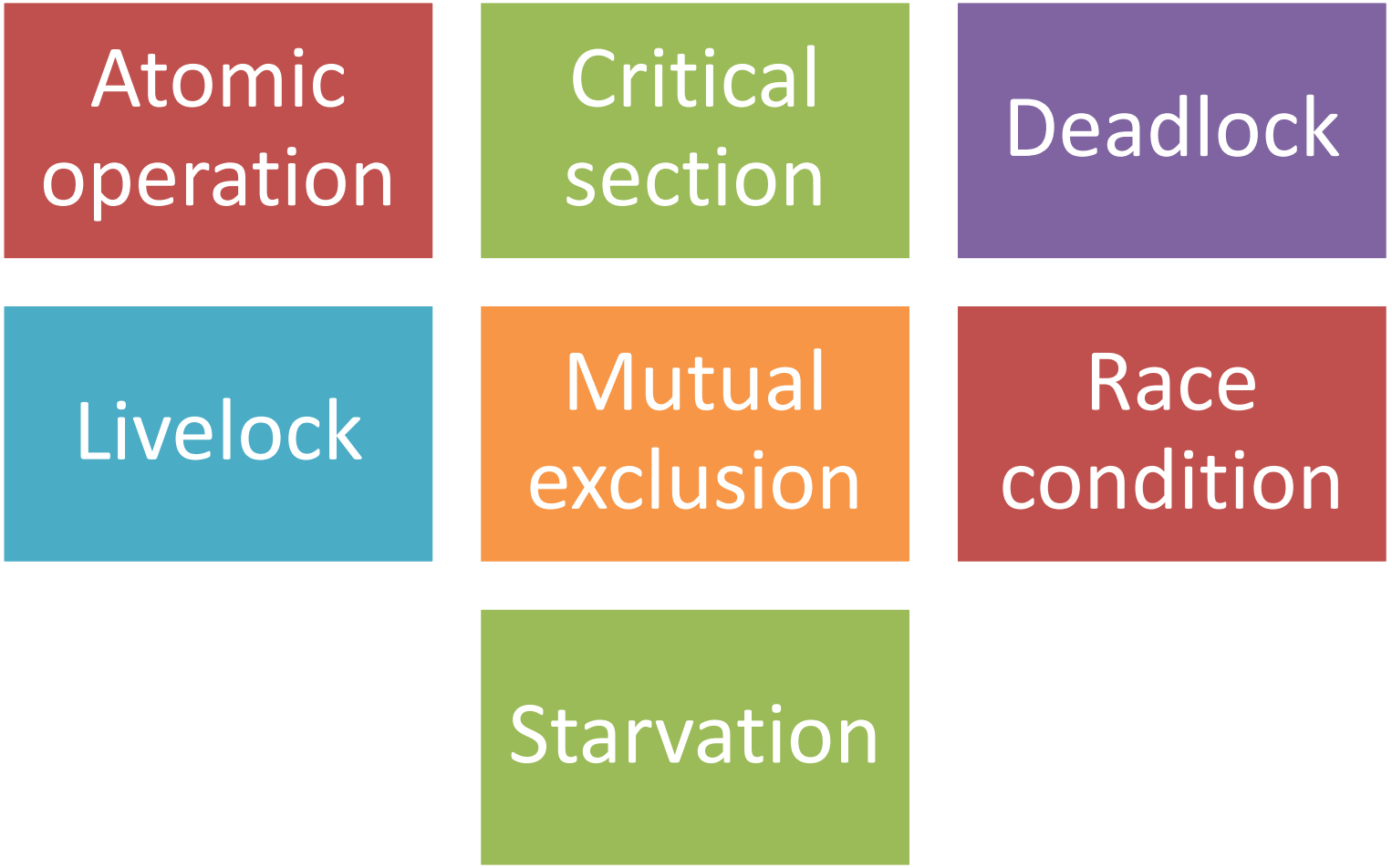



Single-threaded process model
Process control block
User address space
User stack
Kernel stack
Multithreaded process model
Process control block
User address space
Thread
Thread control block
User stack
Kernel stack
Thread
Thread control block
User stack
Kernel stack
Thread
Thread control block
User stack
Kernel stack

3.2 User-level / Kernel-level threads



3.3 Some key terms related to concurrency



3.3.1 Atomic operation

- Atomic operation is completed task by running a sequence of one instruction within a process.
- During atomic operation running the interrupt are disabled.

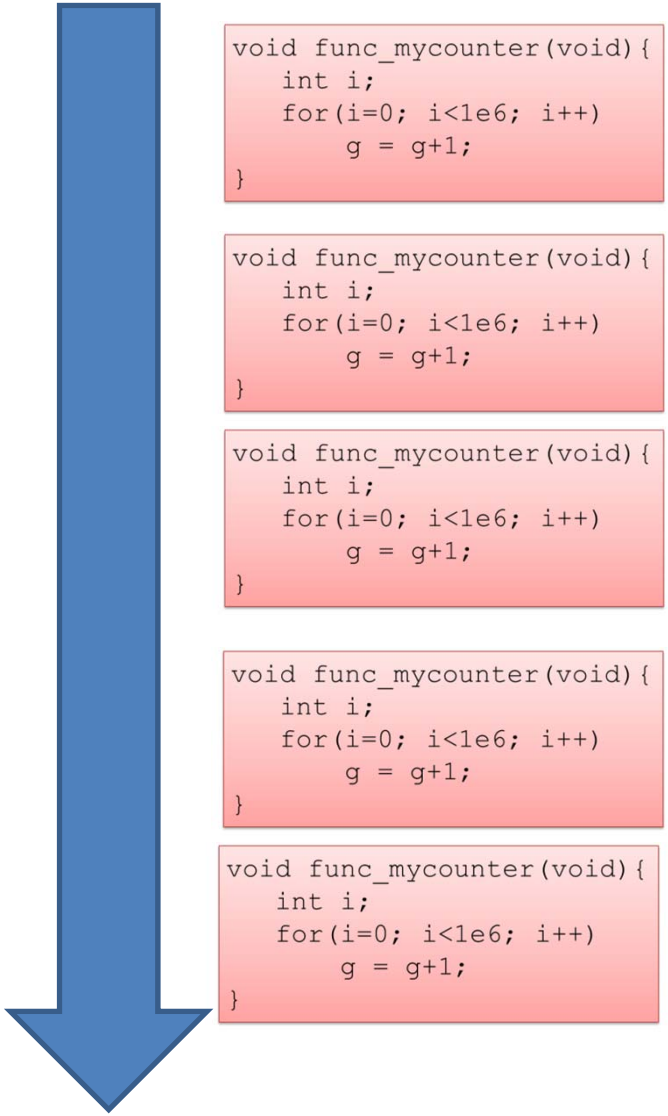
3.3.2 Race condition

A situation in multithread run on a shared resource. Each thread race do the job on the shared result that cause of unexpected result.

```
int g;  //global variable
```

```
void func_mycounter(void){
   int i;
   for(i=0; i<1e6; i++)
       g = g+1;
}
```

Single thread



```
void func_mycounter(void){
    int i;
    for(i=0; i<1e6; i++)
        g = g+1;
}
```

```
void func_mycounter(void){
    int i;
    for(i=0; i<1e6; i++)
        g = g+1;
}
```

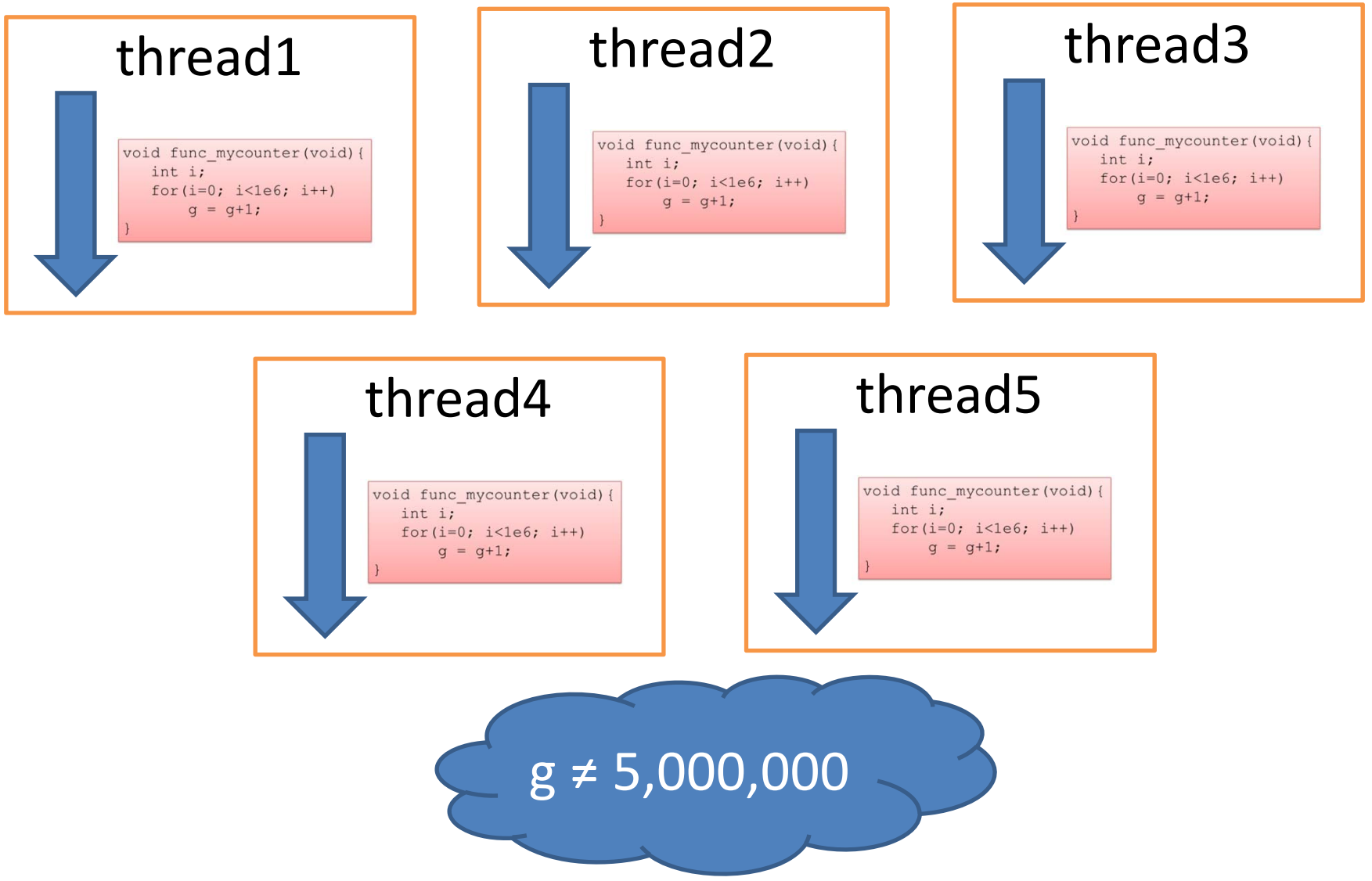
```
void func_mycounter(void){
    int i;
    for(i=0; i<1e6; i++)
        g = g+1;
}
```

```
void func_mycounter(void){
    int i;
    for(i=0; i<1e6; i++)
        g = g+1;
}
```

```
void func_mycounter(void){
    int i;
    for(i=0; i<1e6; i++)
        g = g+1;
}
```

g = 1e6 x 5
= 5,000,000

Multithread—Race condition



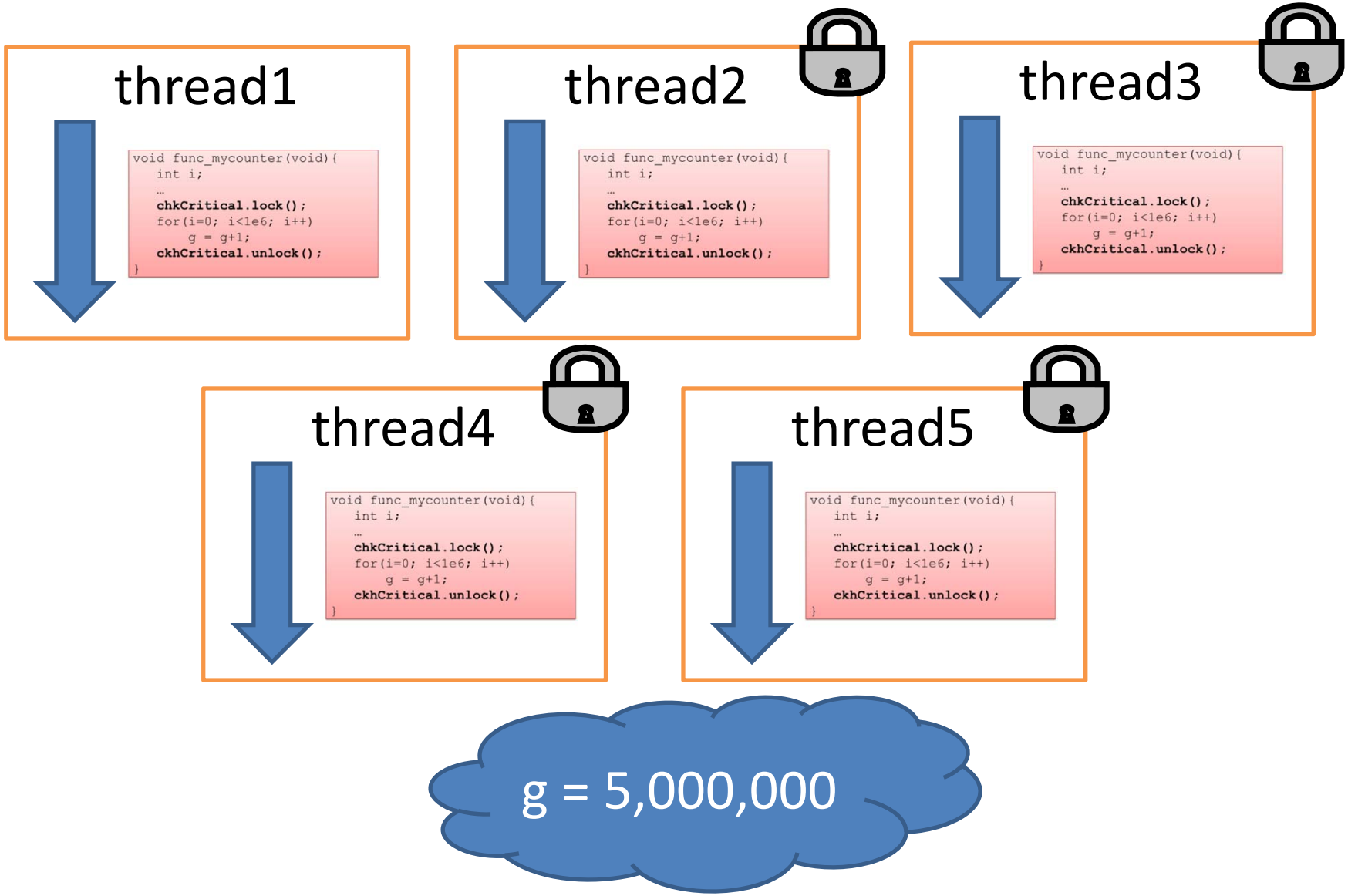
3.3.3 Critical section

A code part that a thread access to shared resource is called critical section. Only one thread can run in critical section, called mutual exclusion.

```
int g;  //global variable
mutex  g_mutex;
```

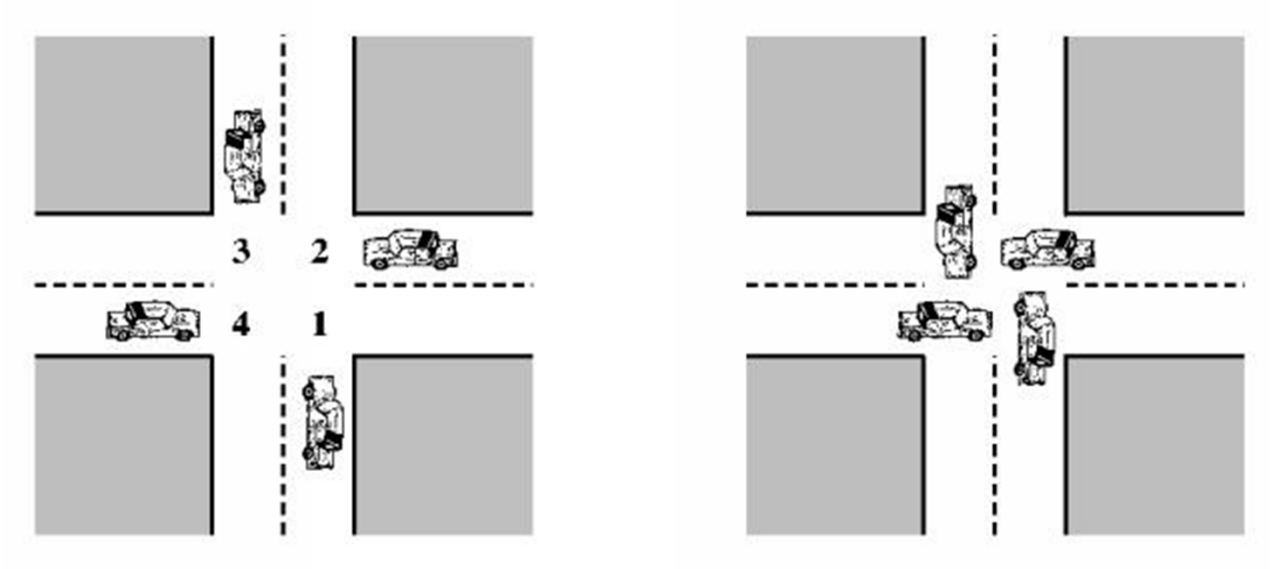
```
void func_mycounter(void){
    int i;
    …
    chkCritical.lock();
    for(i=0; i<1e6; i++)
        g = g+1;
    ckhCritical.unlock();
}
```

Multithread—Critical section



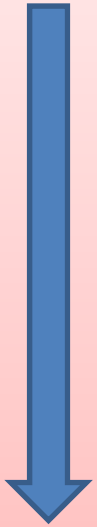
3.3.4 Deadlock

- A situation in which two or more processes are unable to proceed because each is waiting for one of the other do something.

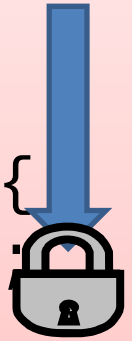


Multithread—Deadlock (Case1)

```
void funcA(void){
    MutexA.lock();
    ..
    MutexB.lock();
    ..
    MutexB.unlock();
    MutexA.unlock();
}
```

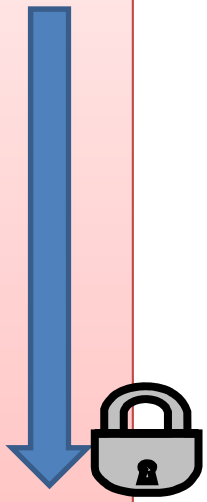


```
void funcB(void){
    MutexB.lock();
    ..
    MutexA.lock();
    ..
    MutexA.unlock();
    MutexB.unlock();
}
```

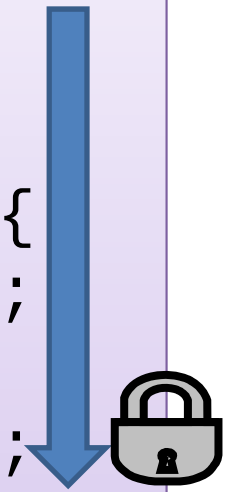


Multithread—Deadlock (Case2)

```
void funcA(void){
    MutexA.lock();
    ..
    MutexB.lock();
    ..
    MutexB.unlock();
    MutexA.unlock();
}
```

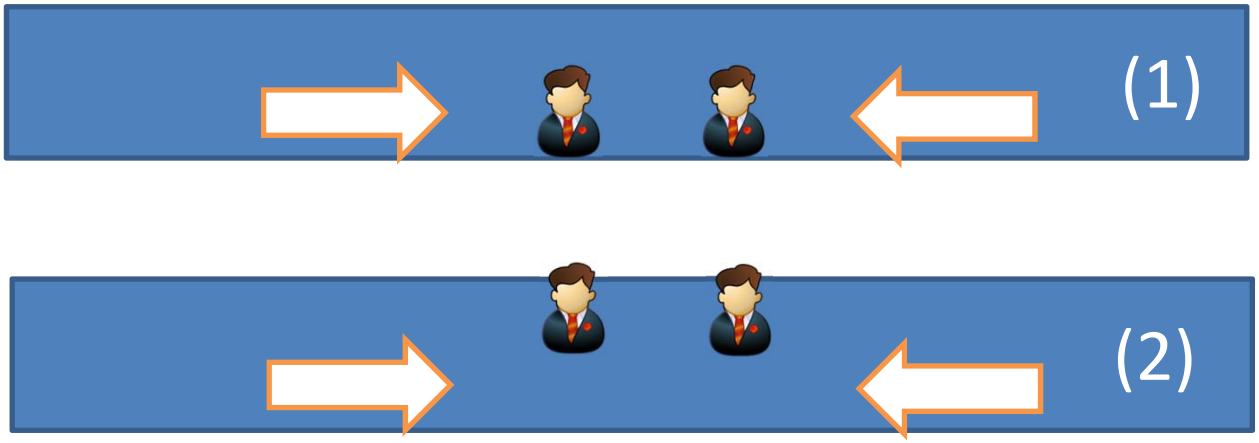


```
void funcB(void){
    MutexB.lock();
    ..
    MutexA.lock();
    ..
    MutexA.unlock();
    MutexB.unlock();
}
```



3.3.5 Livelock

- A situation in which two or more process continuously change their states in response to changes in the other process without doing any useful work



3.3.6 Starvation

- A situation which a runnable process is overlooked indefinitely by the scheduler; although it is able to proceed, it is never chosen.

Summary