Chapter 7
Java Multi-Thread

Wang Yang

Outline

- **Review of Generic Types**
- **Why Threads ?**
- **Definition of Threads**
- **Usage of Threads**
- **Security of Threads**

Review of Generic Types

Generic Types

- Parameterized Type
- Generic Types
- Generic Methods
- Erasure
- Bounded Type Parameter
- Wildcard

Why Threads ?

Why Threads?

- Why Concurrent?
  - If we only have a process
  - you can’t write document with music
  - and with chat with your friends

Why Threads?

- If QQ has only a thread
  - you can only chat with a ppmm
- If QQ has multi threads
  - you can chat with multiple-ppmm

Threads is everywhere

- When you write document:
  - a thread receive your input;
  - a thread save the work in the background
  - a thread check your grammar

Threads is everywhere

- When you run QQ
  - multi-threads let you chat with multiple ones
  - multi-threads let you receive and send multiple files
  - a thread let you can listen music at the same time
  - a thread send you advertise each period

My Operating System

- 线程：997 进程：203

活动监视器 (所有进程)

CPU | 内存 | 能耗 | 磁盘 | 网络

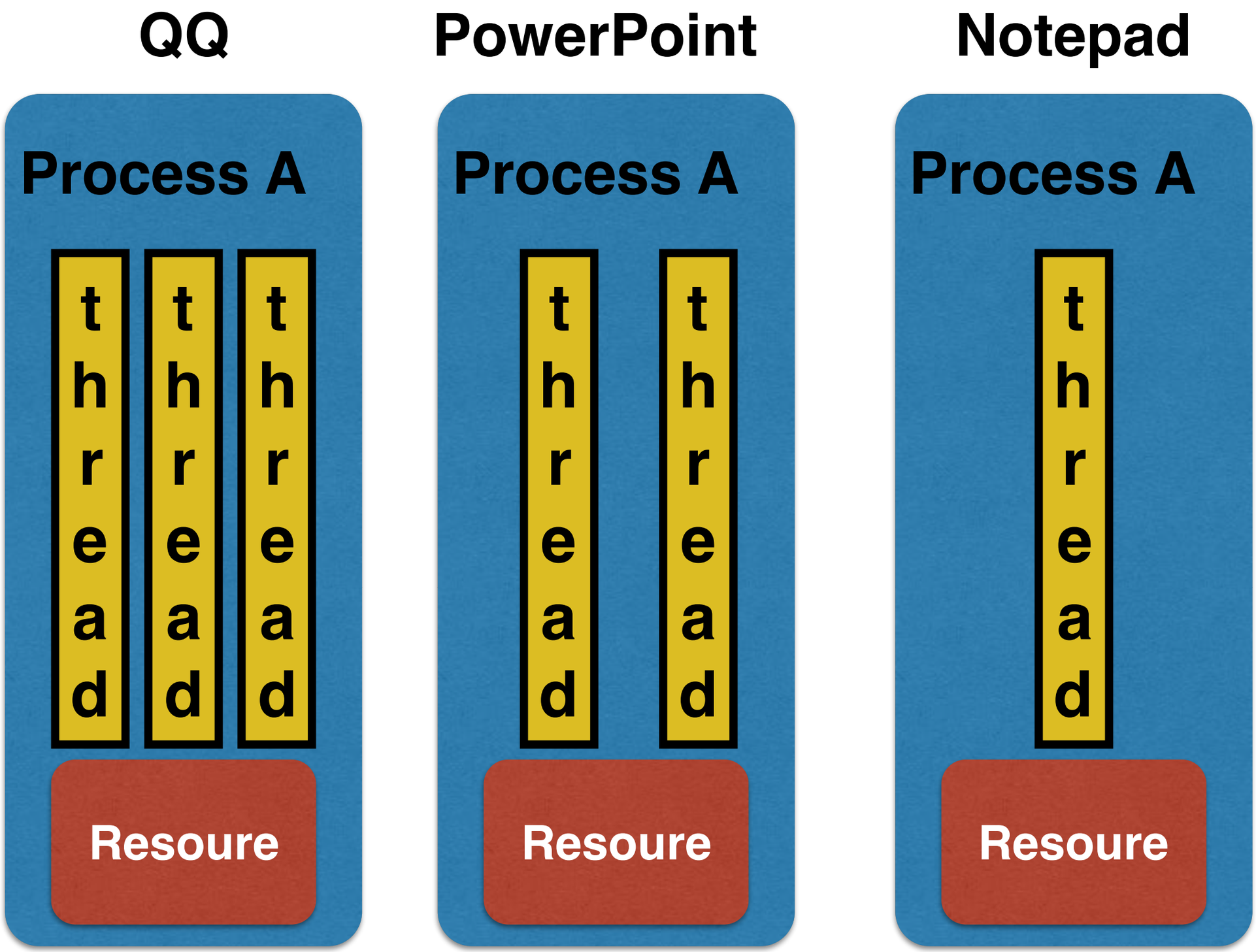
进程名称	已发送字节	已接收字节	已发送包	已接收包	PID	用户	线程
kernel_task	0 字节	0 字节	0	0	0	root	109
Firefox	178 KB	1.1 MB	332	1,220	17394	wyang	41
Google Chrome	87 KB	422 KB	171	454	15935	wyang	33
Thunderbird	0 字节	0 字节	0	0	19019	wyang	29
mysqld	0 字节	0 字节	0	0	2431	_mysql	25
PepperFlashPlayer (Chrome…	0 字节	0 字节	0	0	19127	wyang	23
Evernote	69 KB	148 KB	278	298	18503	wyang	18
Google Chrome 浏览器工作…	0 字节	0 字节	0	0	15941	wyang	16
Google Chrome 浏览器呈现器	0 字节	0 字节	0	0	19114	wyang	13
Google Chrome 浏览器呈现器	0 字节	0 字节	0	0	19117	wyang	13
Google Chrome 浏览器呈现器	0 字节	0 字节	0	0	19151	wyang	13
Google Chrome 浏览器呈现器	0 字节	0 字节	0	0	19287	wyang	13
Dashboard	0 字节	0 字节	0	0	11297	wyang	12
Dock	0 字节	0 字节	0	0	186	wyang	11
distnoted	0 字节	0 字节	0	0	173	wyang	11
softwareupdated	37 KB	19.5 MB	56	14,229	145	_software	11
Microsoft PowerPoint	0 字节	0 字节	0	0	17748	wyang	10
opendirectoryd	0 字节	0 字节	0	0	34	root	10
Finder	0 字节	0 字节	0	0	190	wyang	10
launchservicesd	0 字节	0 字节	0	0	52	root	9
galileod	0 字节	0 字节	0	0	62	root	9

Definition of Threads

Definition of Threads

- Java Thread
  - Piece of code (method) being able to run concurrently with others
  - A part of Process (进程)
  - A non-multithread process is a single-thread process
  - main method is a thread (called main thread)
  - run method is a thread(normal thread)
  - All threads in a process shares the resource of the process

Threads and Processes



the simplest thread

- main function
  - all process have one main thread
  - the main thread is the first thread in the process

```
public static void main(){
    System.out.println("Hello World");
}
```

IF you want more threads

- Thread Class
  - public void run() : like main function
  - each thread has a run method
  - each process has a main method
  - other methods :
    - threads communication : join,interrupt,
    - change thread state : start, sleep
    - get and set field : getName, setName, getPriority, setPriority

IF you want more threads

- Inherited from Thread class
  - Create a subclass of Thread class
  - Override the run() method
  - Run the object of this subclass

Use Thread Class

First I am a thread

```
public class MyThread extends Thread{
    public void run(){
        System.out.println("I am a normal thread");
    }

    public static void main(String args[]){
        Thread t = new MyThread();
        t.start();
        System.out.println("I am main thread");
    }
}
```

```
I am main thread
I am a normal thread
```

Use Thread Class

override the run method, that is thread execution point

```
public class MyThread extends Thread{
    public void run(){
        System.out.println("I am a normal thread");
    }

    public static void main(String args[]){
        Thread t = new MyThread();
        t.start();
        System.out.println("I am main thread");
    }
}
```

```
I am main thread
I am a normal thread
```

Use Thread Class

```
public class MyThread extends Thread{
    public void run(){
        System.out.println("I am a normal thread");
    }

    public static void main(String args[]){
        Thread t = new MyThread();
        t.start();
        System.out.println("I am main thread");
    }
}
```

define a thread ref

```
I am main thread
I am a normal thread
```

Use Thread Class

```
public class MyThread extends Thread{
    public void run(){
        System.out.println("I am a normal thread");
    }

    public static void main(String args[]){
        Thread t = new MyThread();
        t.start();
        System.out.println("I am main thread");
    }
}
```

create a thread object

```
I am main thread
I am a normal thread
```

Use Thread Class

```
public class MyThread extends Thread{
    public void run(){
        System.out.println("I am a normal thread");
    }

    public static void main(String args[]){
        Thread t = new MyThread();
        t.start();
        System.out.println("I am main thread");
    }
}
```

tell the thread begin to run

```
I am main thread
I am a normal thread
```

IF you want more threads

- But if MyThread extends thread
  - MyThread can’t extends other class
  - why I need a class that can do nothing except do threading …

IF you want more threads

- So usually we Implements Runnable interface
  - Create a Runnable class by implementing Runnable interface
  - the class can still extends other class
  - Assign this Runnable object to a Thread class
  - Run the Thread object

Implements the Runnable

First I am a runnable class

```
public class MyTask implements Runnable{
    public void run(){
        System.out.println("I am a runnable task");
    }
    public static void main(String[] args){
        Thread t = new Thread(new MyTask());
        t.start();
        System.out.println("I am main thread");
    }
```

```
I am main thread
I am a runnable task
```

Implements the Runnable

```
public class MyTask implements Runnable{
    public void run(){
        System.out.println("I am a runnable task");
    }
    public static void main(String[] args){
        Thread t = new Thread(new MyTask());
        t.start();
        System.out.println("I am main thread");
    }
```

implements run method

```
I am main thread
I am a runnable task
```

Implements the Runnable

```
public class MyTask implements Runnable{
    public void run(){
        System.out.println("I am a runnable task");
    }
    public static void main(String[] args){
        Thread t = new Thread(new MyTask());
        t.start();
        System.out.println("I am main thread");
    }
```

define a thread ref

```
I am main thread
I am a runnable task
```

Implements the Runnable

```
public class MyTask implements Runnable{
    public void run(){
        System.out.println("I am a runnable task");
    }
    public static void main(String[] args){
        Thread t = new Thread(new MyTask());
        t.start();
        System.out.println("I am main thread");
    }
```

create a thread object,and tell the thread use MyTask's run method

```
I am main thread
I am a runnable task
```

Implements the Runnable

```
public class MyTask implements Runnable{
    public void run(){
        System.out.println("I am a runnable task");
    }
    public static void main(String[] args){
        Thread t = new Thread(new MyTask());
        t.start();
        System.out.println("I am main thread");
    }
```

tell the thread begin to run

```
I am main thread
I am a runnable task
```

Thread and Resources

- Java Thread
  - method
  - local variable, parameter
- Resources
  - Objects
  - Class info
  - Static Variable

Thread and Resources

```
public class MyTask implements Runnable{
    FileInputStream fis;
    public MyTask(FileInputStream _fis){
        fis = _fis;
    }
    public void run(){
        //read content from the fis
        ....
    }
    public static void main(String[] args){
        FileInputStream fis = new FileInputStream("1.txt");
        Thread t = new Thread(new MyTask(fis));
        t.start();
    }
```

transfer the objects through Constructor

summary

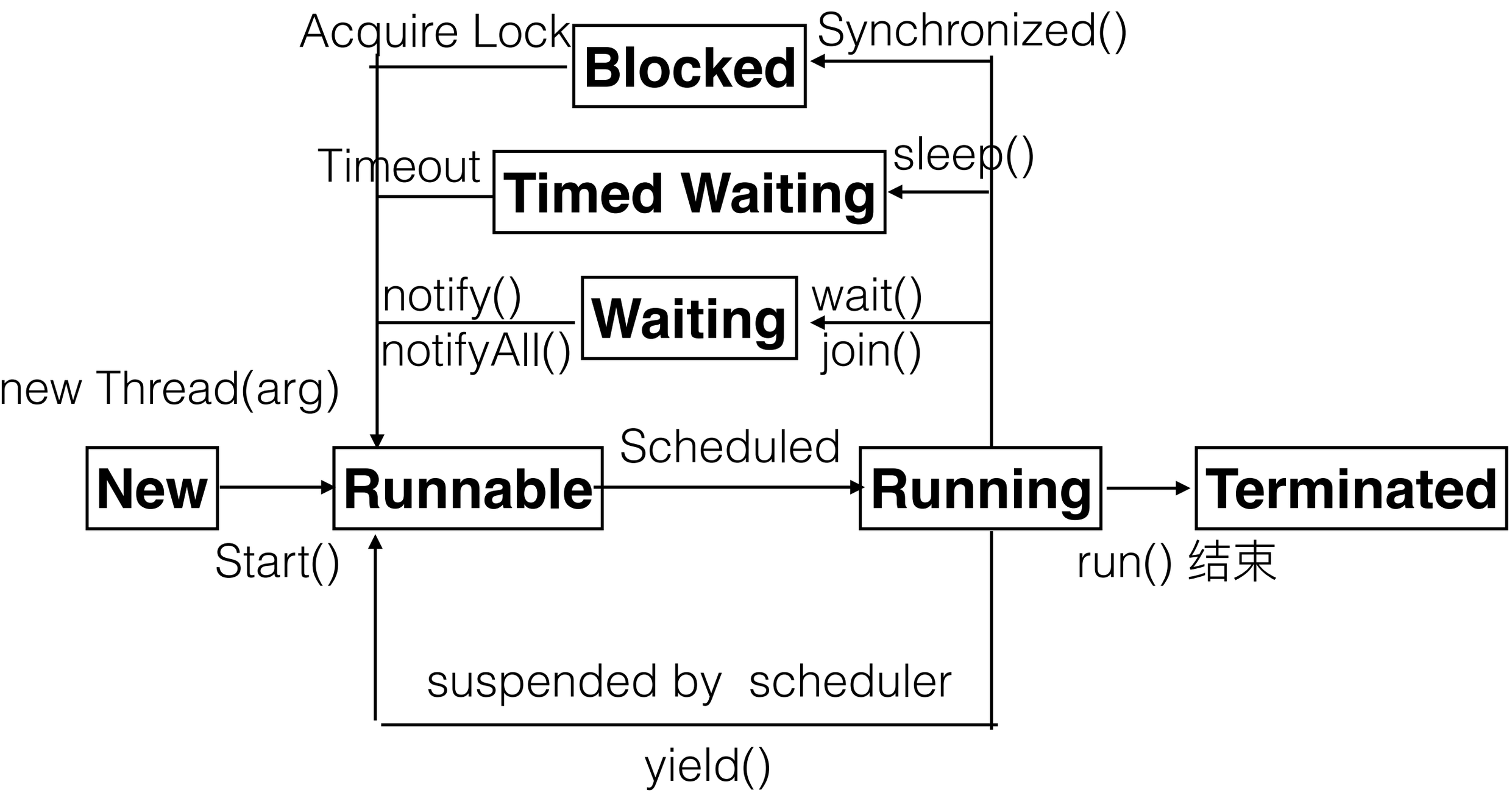
- thread : a method
- Thread Class
- Runnable Interface
- new Thread()
- new Thread(new runnable())
- start

Usage of Threads

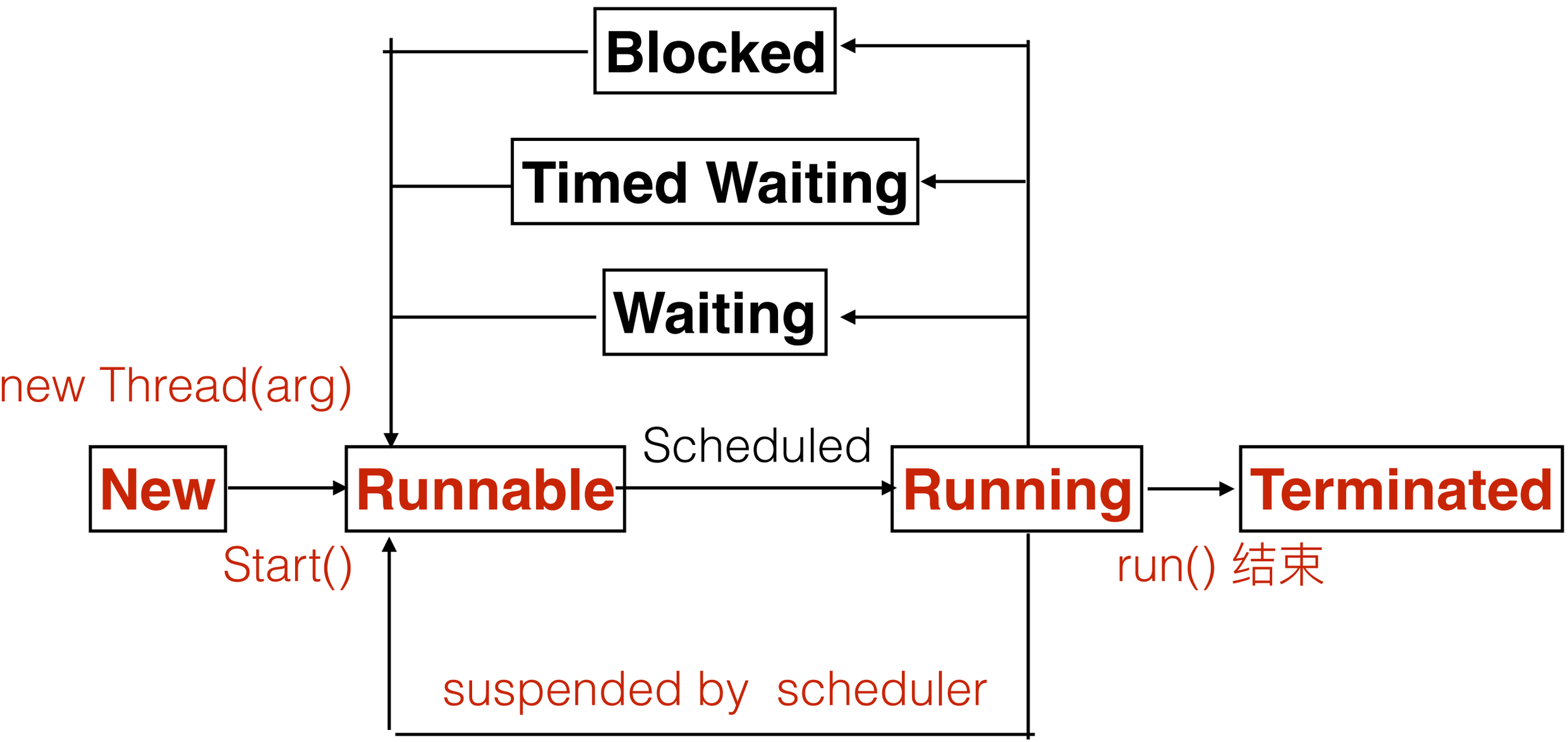
Thread State

- You can control the thread outside it
  - start to run
  - stop for a while
  - stop and wait some signal
  - exit

Thread State



Thread State



Sometime we want a thread wait another thread terminated

```
public class MyTask implements Runnable{
    FileInputStream fis;
    public MyTask(FileInputStream _fis){
        fis = _fis;
    }
    public void run(){
        //read content from the fis
        ....
    }
    public static void main(String[] args){
        FileInputStream fis = new FileInputStream("1.txt")
        Thread t = new Thread(new MyTask(fis));
        t.start();
        fis.close();
```

throw Exception because the fis has been closed

main thread exit before MyTask Thread

Sometime we want a thread wait another thread terminated

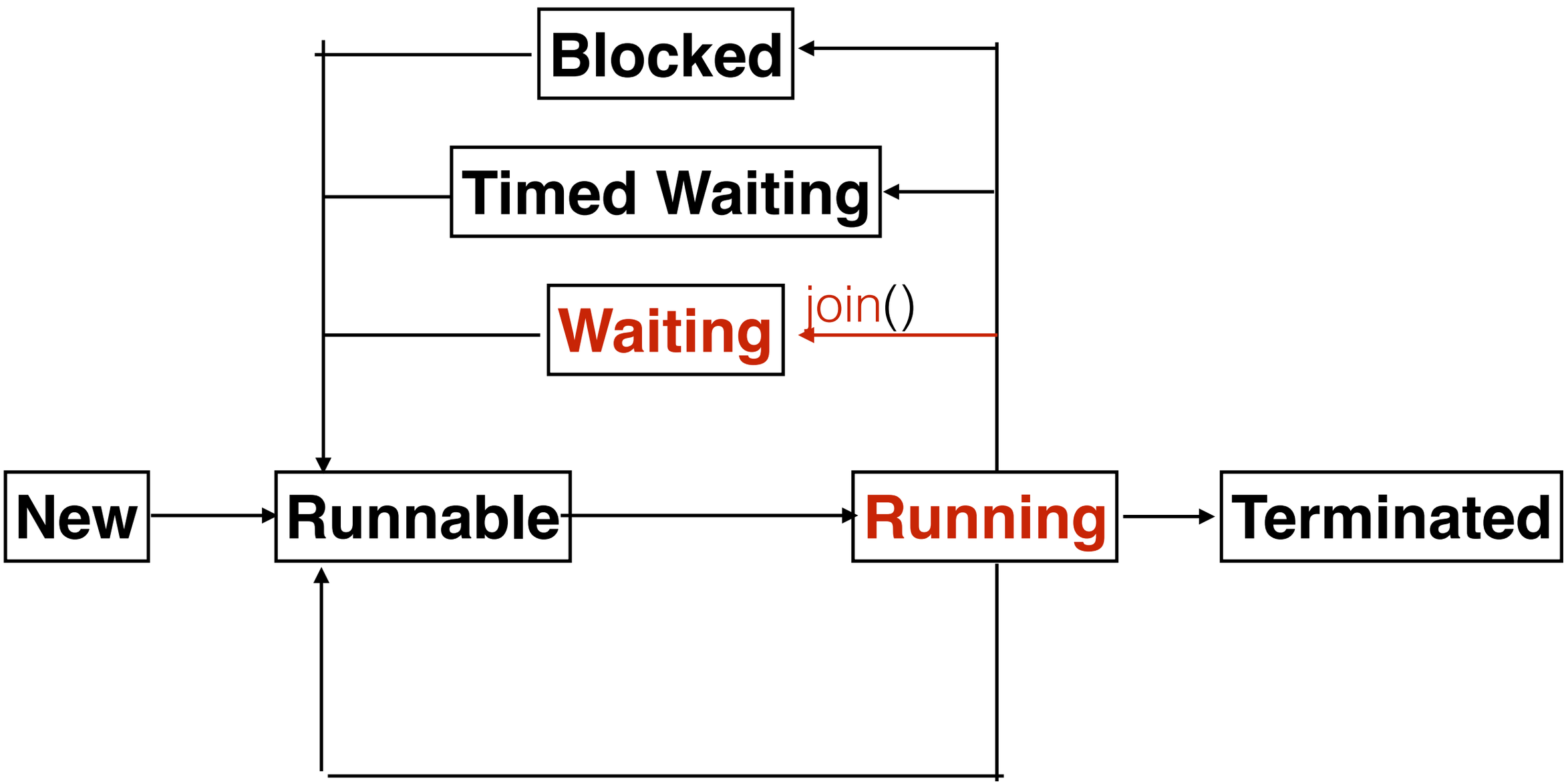
- join()
  - Waits for this thread to die
  - use to control the resource management
  - throws Interrupted Exception
  - overload
    - join() : wait until the thread terminated
    - join(long millis) : wait until the thread terminated or wait after millisecond time

Sometime we want a thread wait another thread

```
try{
    FileInputStream fis = new FileInputStream("1.txt")
    Thread t = new Thread(new MyTask(fis));
    t.join();
    fis.close();
} catch (InterruptedException e){
    System.out.println(e);
} catch (IOException e){
    System.out.println(e);
}
```

waiting for the MyTask Thread Terminated

Thread State



Sometimes we want wait but not terminated

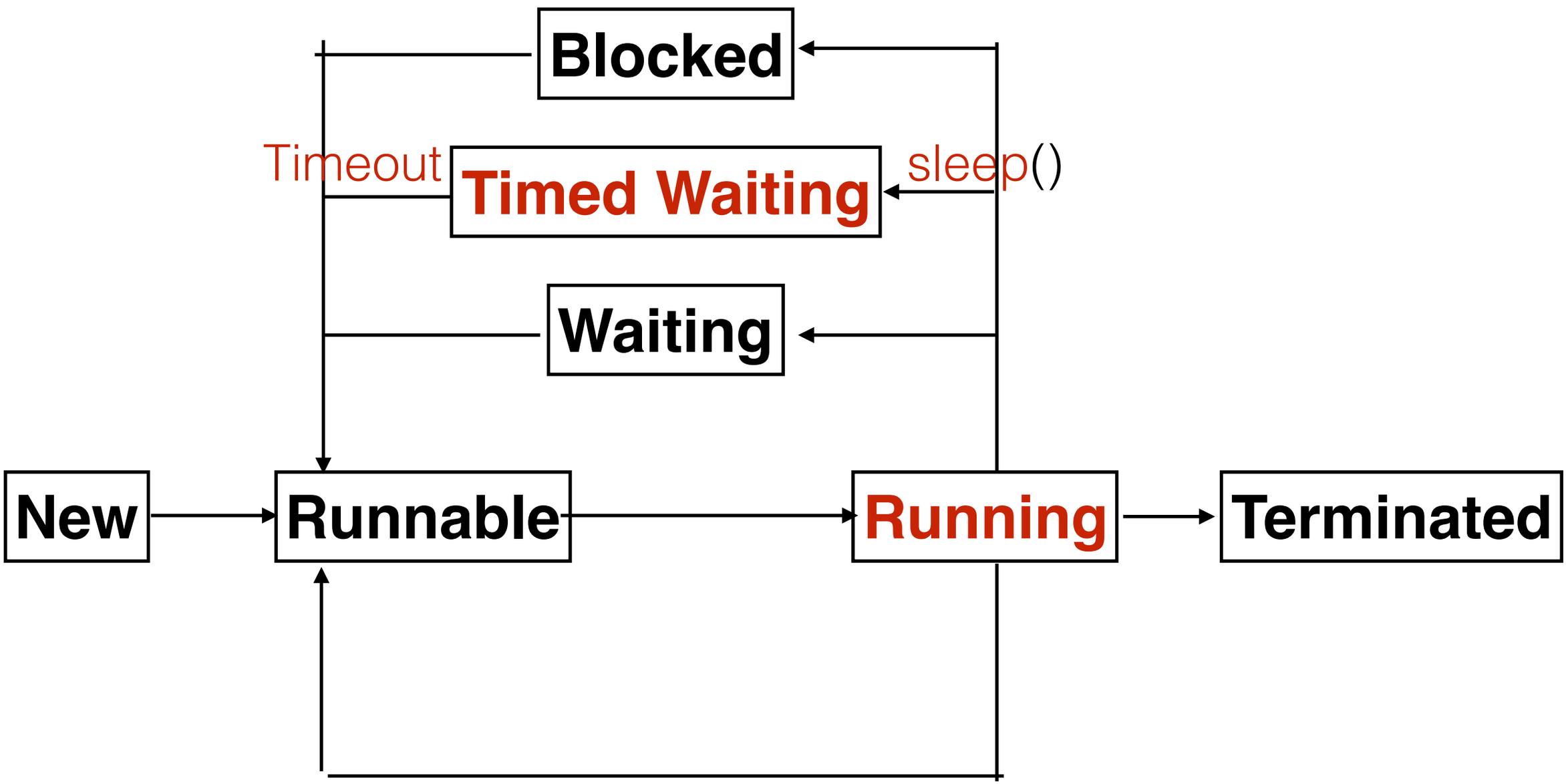
- sleep(long millis)
  - static method : Thread.sleep(millis)
  - Causes the currently executing thread to sleep
  - specified number of milliseconds : 1s is 1000ms
  - will throws InterruptedException

Sometimes we want wait but not terminated

```
public void run(){
    //read content from the fis
    try{
        //code ...
        Thread.sleep( 60 * 1000L);
        //code...
    } catch(InterruptedException e){
        System.out.println(e);
    }

}
```

Thread State



Sometime we want terminate other threads

Thread A has run or sleep hours

Thread B join A and join A and join A….

Sometime we want terminate other threads

Sometime we want terminate other threads

Thread A terminated

Thread B join A success

Sometime we want terminate other threads

- interrupt()
  - Interrupts this thread.
  - cause the thread receives InterruptedException
- interrupted()
  - return boolean
  - Tests whether the current thread has been interrupted.

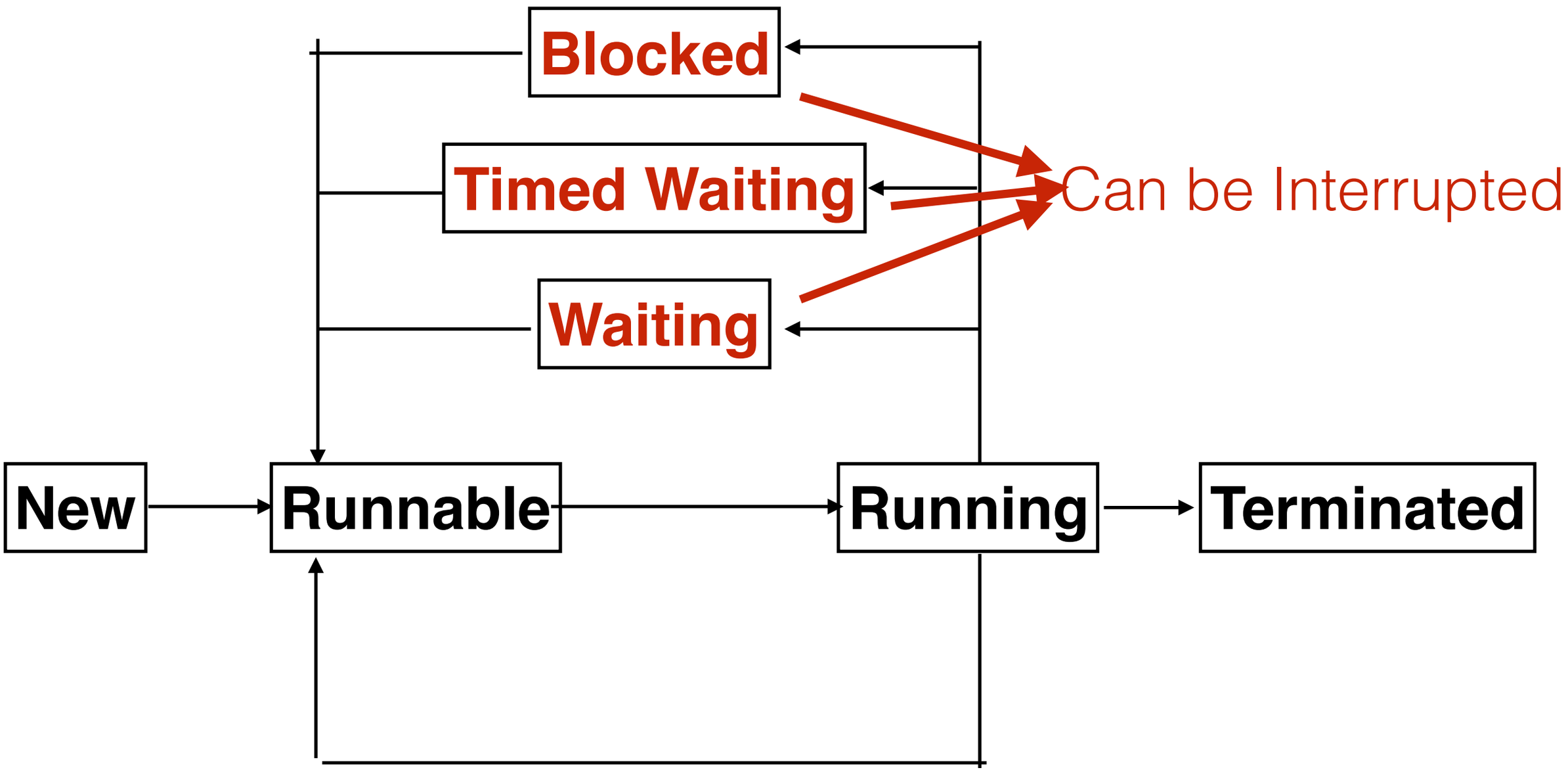
Sometime we want terminate other threads

```
try{
    //code ...
    Thread.sleep( 60 * 1000L);
    //code...
} catch(InterruptedException e){
    System.out.println(e);
}
```

Interrupt() cause the thread receive the exception

```
FileInputStream fis = new FileInputStream("1.txt");
Thread t = new Thread(new MyTask(fis));
t.start();
Thread.sleep(3*1000L);
t.interrupt();
t.join();
```

Thread State



Sometime we want terminate other threads

- Problem for t.interrupt()
  - only work when t is in block/waiting/timed waiting state
    - join
    - sleep
  - if the thread is in running state
    - the InterruptedException will not be thrower
    - the thread will checked the interrupted state by itself

Sometimes we want to set thread's field

Test if itself has been interrupted

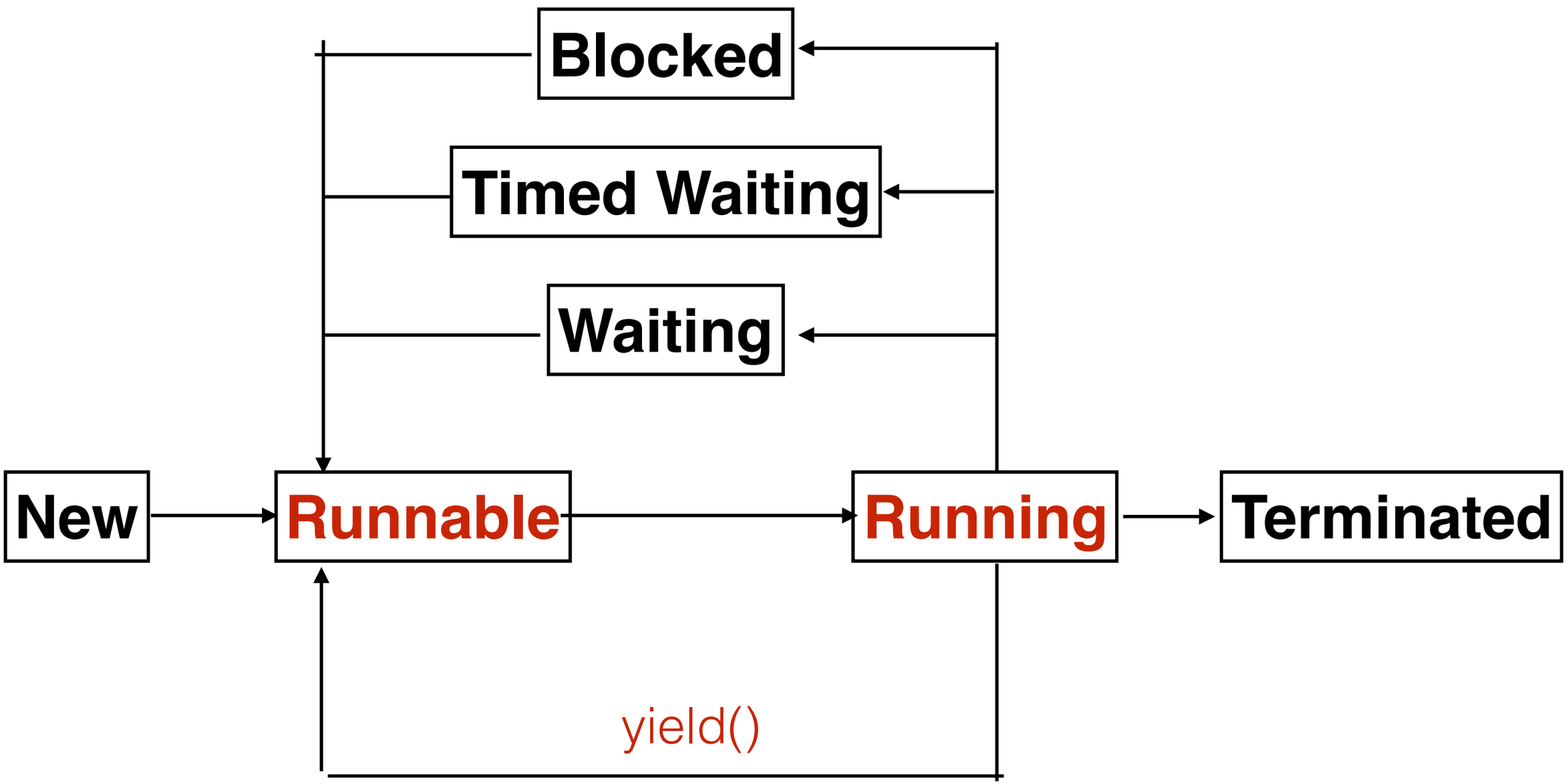
```
try{
    while(Thread.interrupted()){
        //code ...
        Thread.sleep( 60 * 1000L);
        //code...
    }
} catch(InterruptedException e){
    System.out.println(e);
}
```

static function

Sometime We want Voluntary Rescheduling

- Yield()
- Thread.yield()
  - Static Method
- Just a Hint
- No Guarantee

Sometime We want Voluntary Rescheduling



Summary

- join
- sleep
- interrupt/interrupted
- yield

Security of Threads

if we have a share account

```java
public class BankAccount {
    private long id;
    private long balance;
    public BankAccount(){
        balance = 0;
    }
    public long getBalance(){
        return balance;
    }
    public void deposit(long amount){
        balance += amount;
    }
}
```

```java
public class ATM extends Thread{
    private BankAccount account;
    public ATM(BankAccount account){
        this.account = account;
    }
    public void run(){
        for(int i = 0; i < 10000; i++)
            account.deposit(10);
    }
}
```

multiple people want to deposit money to it

```
public static void main(String args[]) throws Exce
    Thread[] t = new Thread[10];
    BankAccount account = new BankAccount();
    for(int i = 0; i < 10; i++){
        t[i] = new ATM(account);
    }
    for(int i = 0; i < 10; i++){
        t[i].start();
    }
    for(int i = 0; i < 10; i++){
        t[i].join();
    }
    System.out.println(account.getBalance());
}
```

balance should = 1000000

multiple people want to deposit money to it

```
public static void main(String args[]) throws Exce
    Thread[] t = new Thread[10];
    BankAccount account = new BankAccount();
    for(int i = 0; i < 10; i++){
        t[i] = new ATM(account);
    }
    for(int i = 0; i < 10; i++){
        t[i].start();
    }
    for(int i = 0; i < 10; i++){
        t[i].join();
    }
    System.out.println(account.getBalance());
}
```

balance = 320670, why

- when multiple thread modify one variable
  - something strange happen!
  - modify action is not a atomic action
    - read the origin value
    - do the algorithm
    - write the new value back

if two thread execute in following order

thread A	read balance from memory, balance = 0
thread B	read balance from memory, balance = 0
thread A	balance +=10, now result be 10
thread B	balance +=10, now result be 10
thread A	write 10 back to memory , balance = 10
thread B	write 10 back to memory , balance = 10

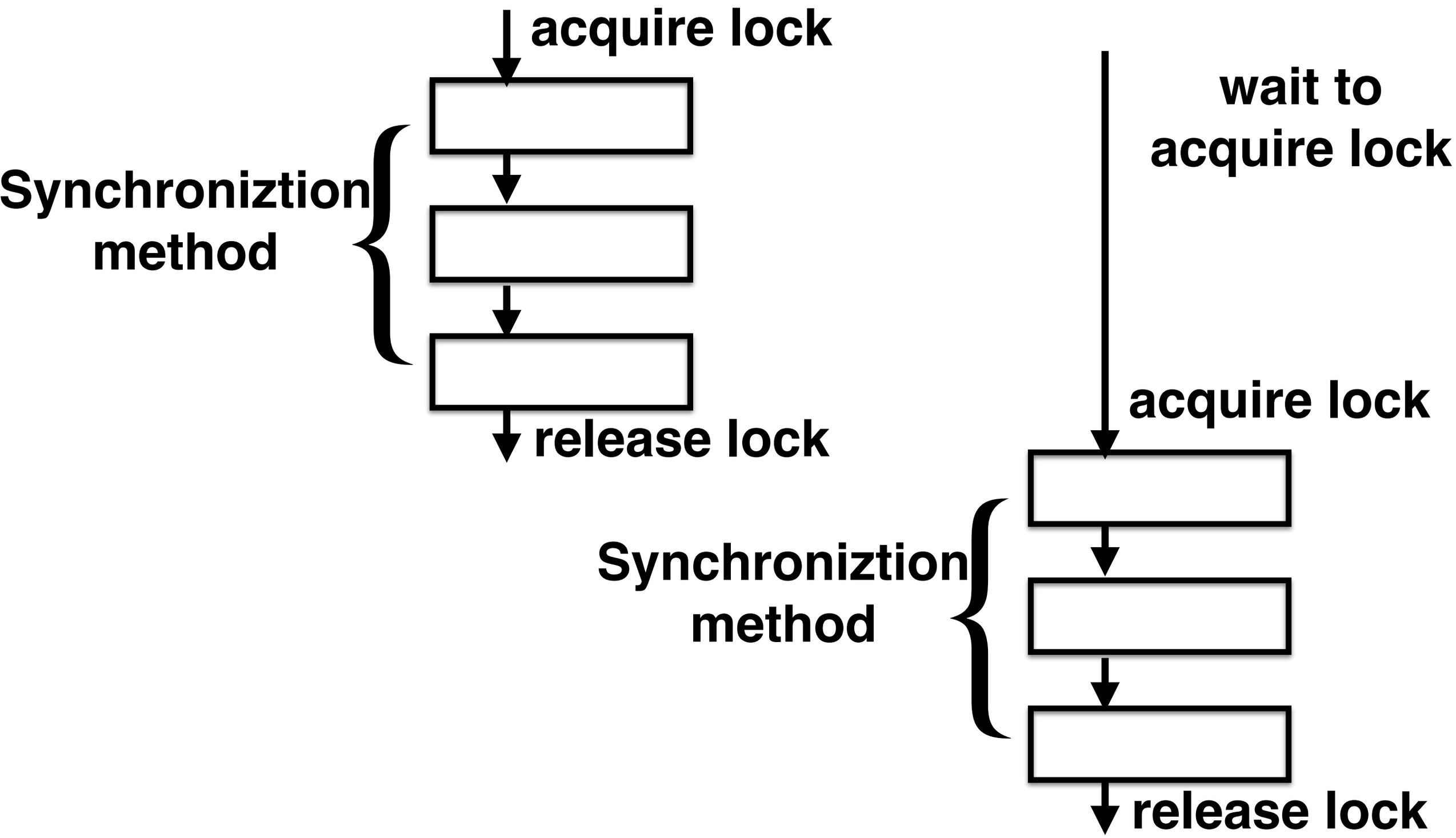
We get balance = 10

But we want race=20!!

Synchronization of Thread

- We need a lock to control the modify action of variable
- the lock let only one thread can modify the variable at one time
- the method with lock we call it Synchronization method

Synchronization Method



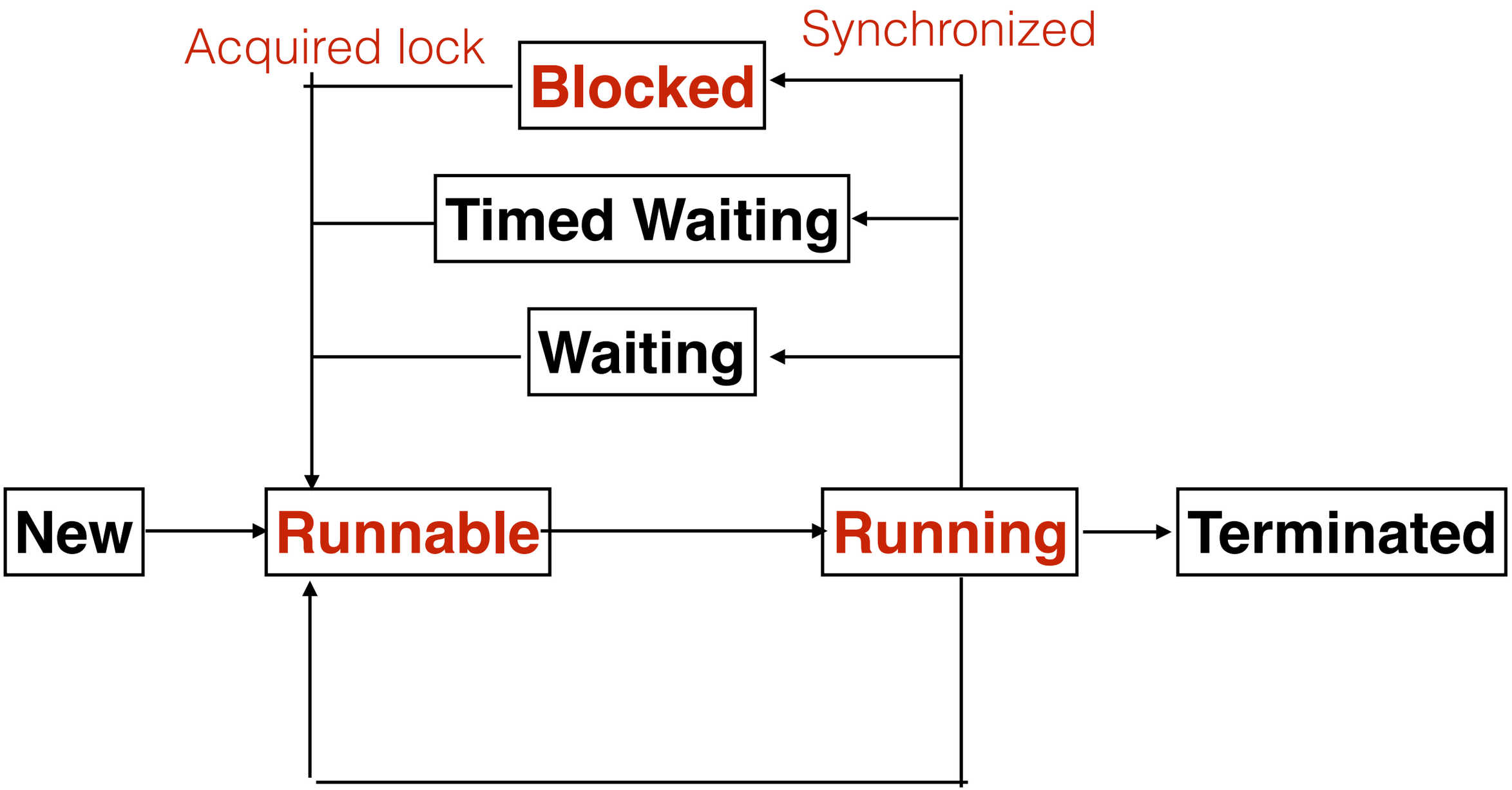
Synchronization Method

```
public class BankAccount {
    private long id;
    private long balance;
    public BankAccount(){
        balance = 0;
    }
    public synchronized long getBalance(){
        return balance;
    }
    public synchronized void deposit(long amount){
        balance += amount;
    }
```

Now we get

1000000$!!

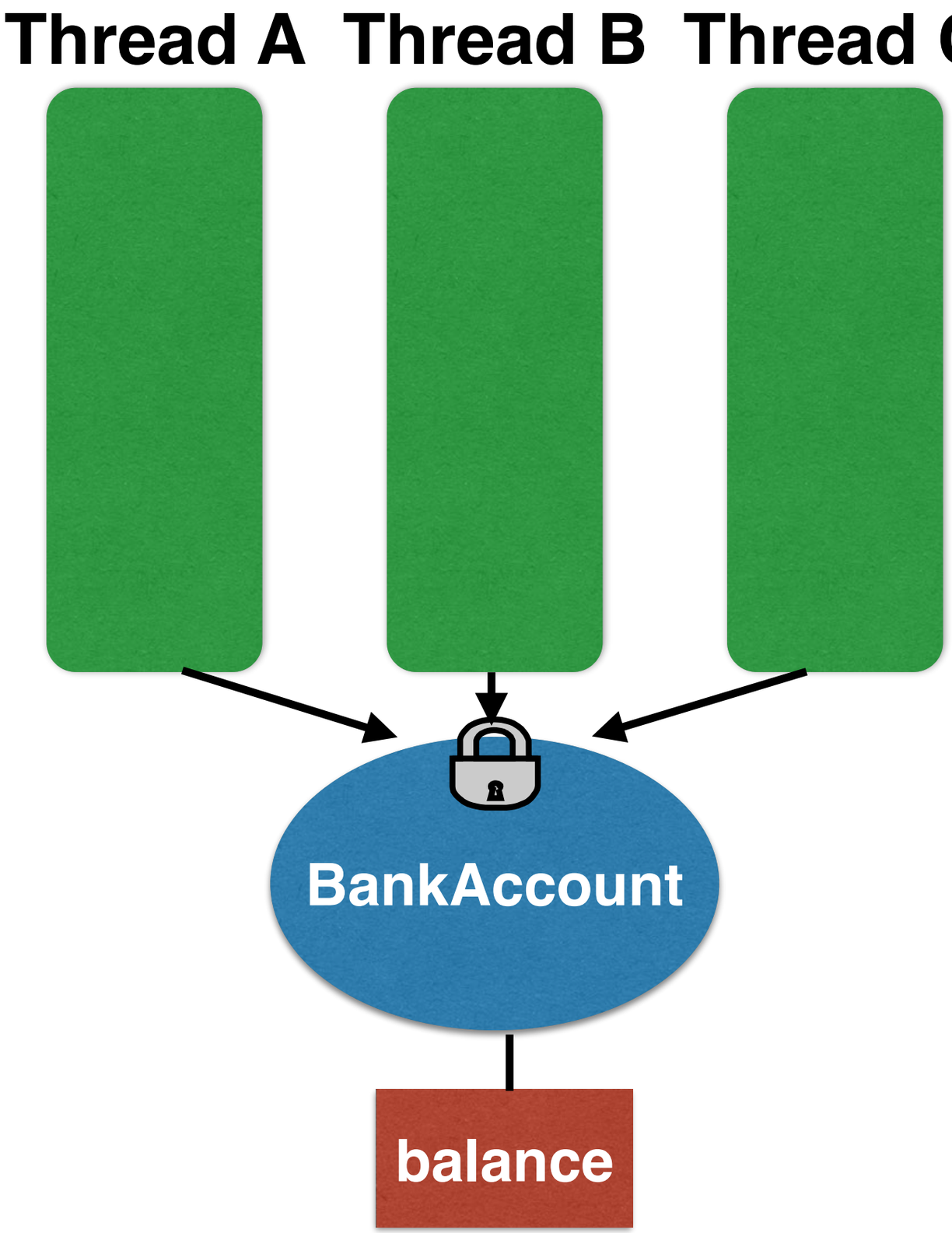
Thread State



Synchronization Method

- Each object has a lock and corresponding key
  - When the object has no synchronized methods, the lock does not work.
  - When the object has a synchronized methods, the lock begins to work.

Synchronization Method



Sometimes only piece of code need be protected

```
public synchronized void deposit(long amount){
    //code...
    //code...
    balance += amount;
    //code...
    //code...
}
```

if only this code need be synchronized

Synchronization Statement

- Synchronize a block of code, not the method
- format : synchronized(key) { code… }

```
public void deposit(long amount){
    //code...
    //code...
    synchronized(this){
        balance += amount;
    }
    //code...
    //code...
}
```

you need define the key, usually be this

Sometimes the lock may lead problems

```
public synchronized long getBalance(){
    return balance;
}
public synchronized void deposit(long amount){
    balance += amount;
}
public synchronized void withdraw(long amount)
    while(balance < amount){
        Thread.sleep(1000);
    }
    balance -= amount;
}
```

if balance > 1000, we get money

or we wait some guy to deposit mone

how to release the lock?

Thread A

deadlock

I will wait until the balance>money, then withdraw money

Thread B

I want deposit money, but I can't get the lock…

We need a way to release the lock temporarily

Wait & Notify

- wait()
- must be in a synchronized method or statement
- release the lock of the object the thread acquired
- the thread state switches from running to waiting
- until another thread invokes the notify() method or the notifyAll() method for this object.
- can have timeout parameter
  - wait(long timeout)

Wait & Notify

```
public synchronized void deposit(long amount){
    balance += amount;
    notifyAll();
}
public synchronized void withdraw(long amount) {
    while(balance < amount){
        wait();
    }
    balance -= amount;
}
```

Wait & Notify

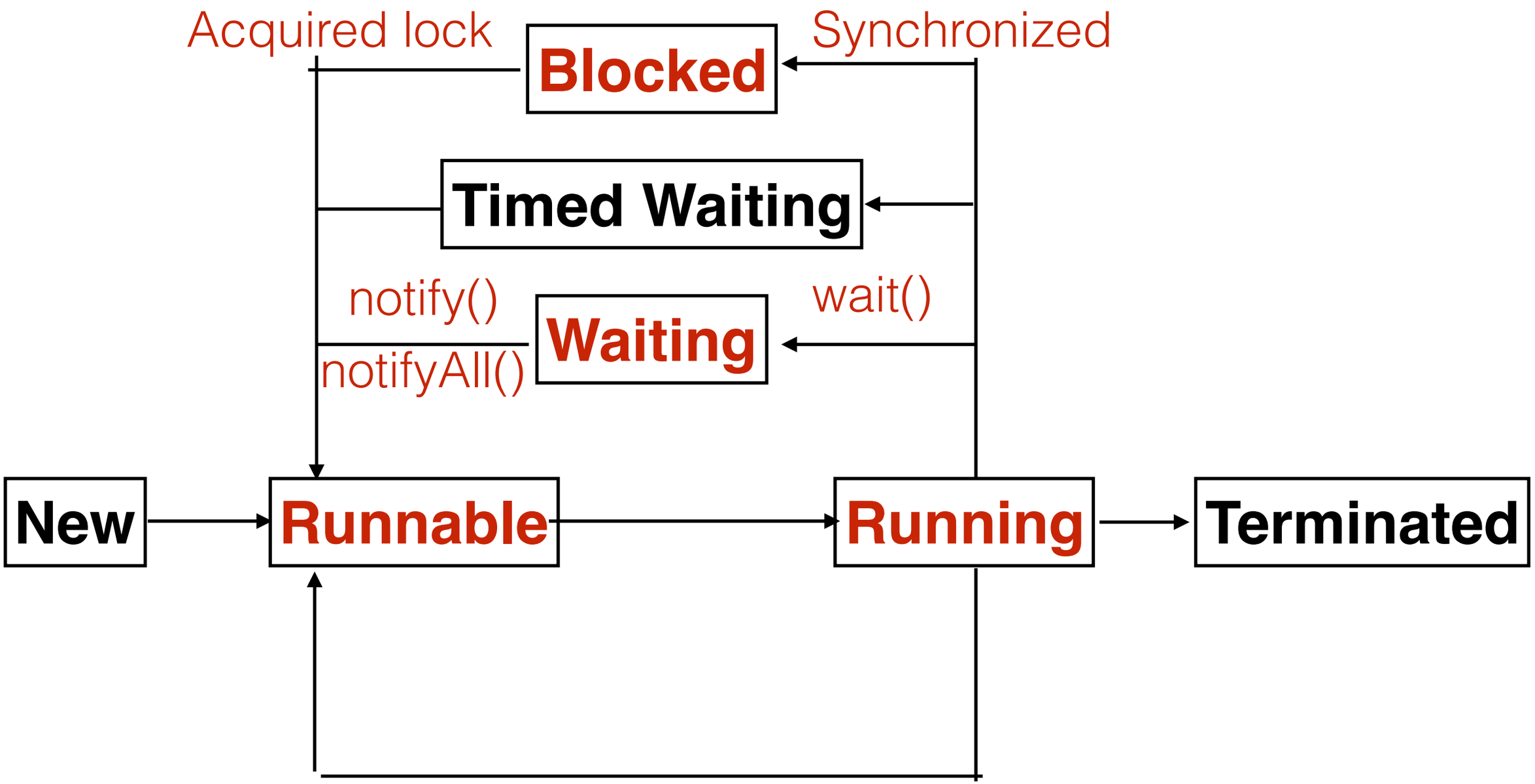
- Tips
  - notify doesn’t mean the condition is fulfilled
    - multiple people withdraw the money
  - notify will random select a thread to wake
  - always use the notifyAll()

Wait & Notify

```
public synchronized void withdraw(long amount)
    if(balance < amount){
        wait();
    }
    balance -= amount;
}
```

when you get notify, maybe the balance is below the amount again

Thread State



Summary

- synchronized method
- synchronized statement
- wait/notify/notifyAll