Threads
Condition variables

Ожидание потоков

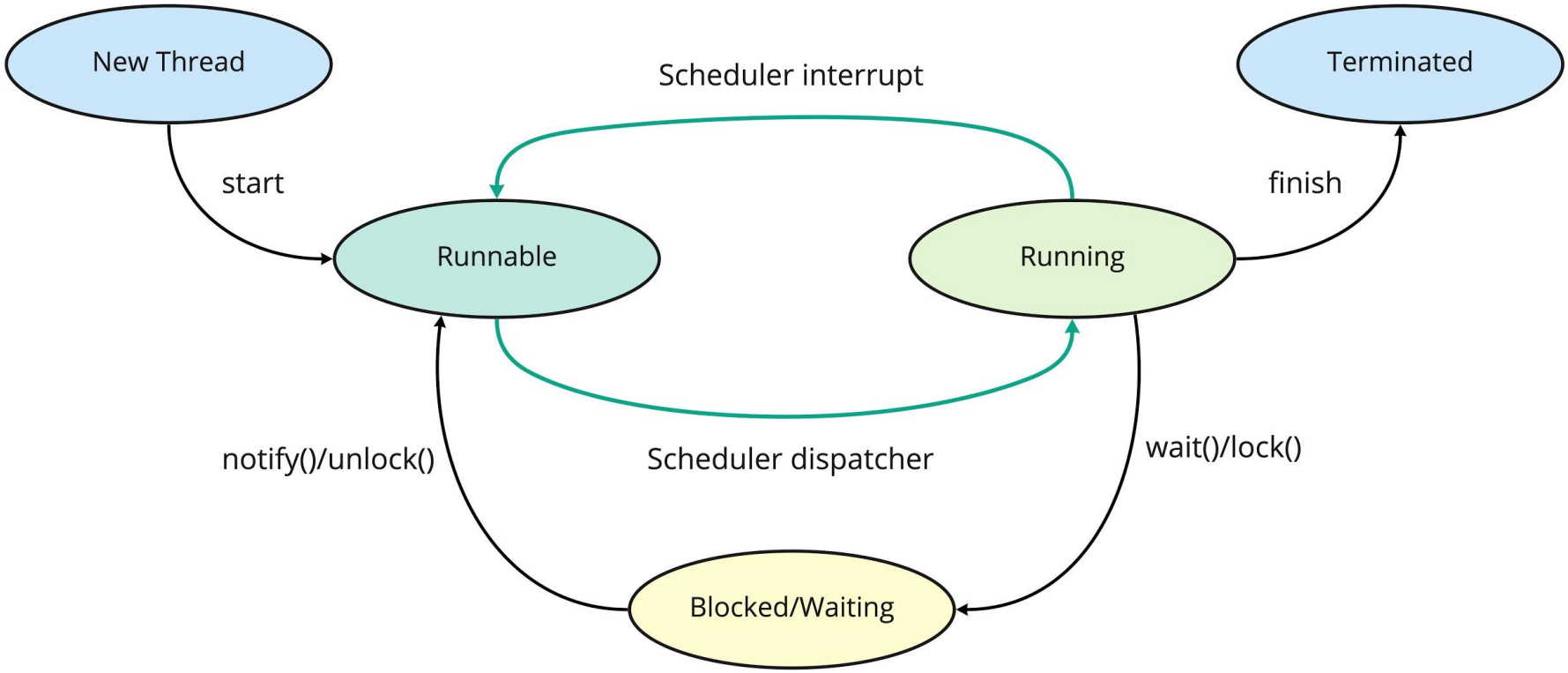
Ожидание (wait)
механизм который заставляет поток ждать пока какой либо другой поток его не “разбудит”

– **wait()** → ждать неопределенное время, пока его не разбудит другой поток
– **wait(long timeout)** → ждать **timeout** миллисекунд, по истечению которых активируется самостоятельно
– **wait(long timeout, int nanos)** → ждать **1_000_000 * timeout + nanos** наносекунд, по истечению которых активируется самостоятельно

Уведомление (notify)
Механизм уведомления “проснутся” для потока который ждет

– **notify()** → уведомляет случайный поток который ожидает на мониторе этого объекта о пробуждении
– **notifyAll()** → уведомляет все потоки которые ожидают на мониторе этого объекта о пробуждении

Thread Lifecycle
жизненный цикл



Барьер для N потоков

```
class Condition {
   private int count;
   final int THREAD_COUNT = 10;
   private final Object monitor = new Object();

   public void barrier(){
      synchronized (monitor){
         count++;
         if(count < THREAD_COUNT){
            try {
               monitor.wait();
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
         }else{
            monitor.notifyAll();
         }
      }
   }
}
```

Барьер для N потоков использование

```
public static void main(String[] args) throws InterruptedException {
    Condition condition = new Condition();

    Runnable r = () -> {
        condition.barrier();
        System.out.println("Worker " + Thread.currentThread().getId());
    };

    List<Thread> threads = Stream.generate(() -> new Thread(r))
            .limit(condition.THREAD_COUNT)
            .peek(Thread::start)
            .collect(Collectors.toList());
    for(Thread t : threads){
        t.join();
    }
}
```

Producer-Consumer
производитель-потребитель

```
public void prepareData() {
   synchronized (monitor){
      while(isReady){
         try {
            monitor.wait();
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      }
      System.out.println("Prepare data...");
      isReady = true;
      monitor.notifyAll();
   }
}
```

```
public void sendData(){
   synchronized (monitor){
      System.out.println("Waiting data...");
      while (!isReady){
         try {
            monitor.wait();
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      }
      System.out.println("Sending data...");
      isReady = false;
      monitor.notifyAll();
   }
}
```

Spurious wakeup
ложное пробуждение

может произойти в связи со сложной реализацией самого механизма wait - notify поэтому рекомендуется ожидать пробуждение всегда в цикле

Producer-Consumer использовагие

```
Thread producer = new Thread(()->{
  while (true){
     messenger.prepareData();
     try {
        Thread.sleep(1000);
     } catch (InterruptedException e) {
        e.printStackTrace();
     }
  }
});
```

```
Thread consumer = new Thread(()->{
  while (true){
     messenger.sendData();
     try {
        Thread.sleep(1500);
     } catch (InterruptedException e) {
        e.printStackTrace();
     }
  }
});
```

Condition variables
Условные переменные

– нужны как механизм взаимодействия потоков, в отличие от мьютексов
– всегда используется с мьютексом
– предназначена для уведомления событии
– атомарно освобождает мьютекс при wait()
– хорошо подходит для задач типа «производитель-потребитель»