Многонишково програмиране в Java

хон. ас. инж. Боян Петров
b_petrov@tu-sofia.bg

Процесите в операционната система

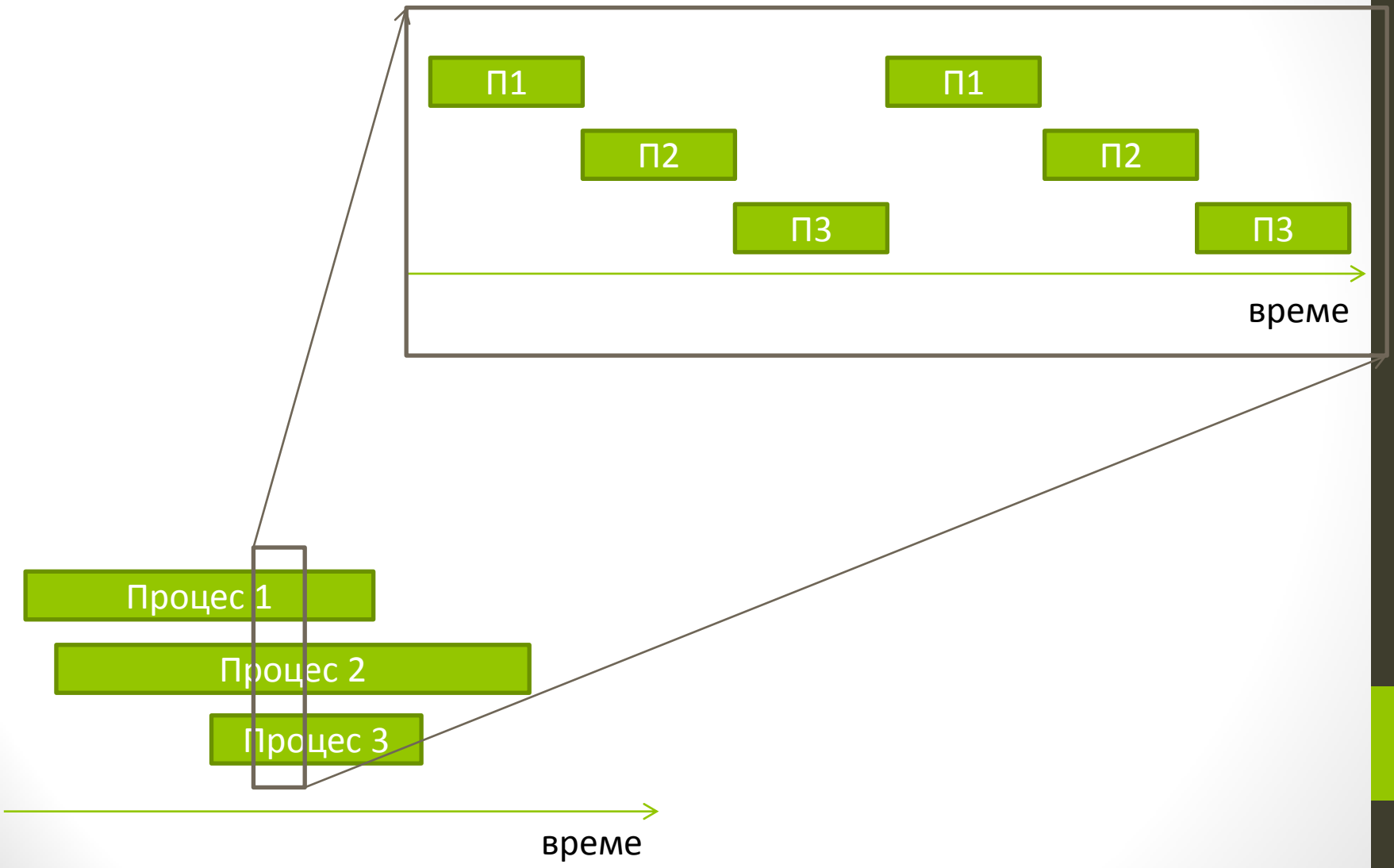
- Процес – инстанция на програма по време на изпълнението си.

- Многозадачност(Multitasking) – способността на ОС да превключва между няколко процеси.

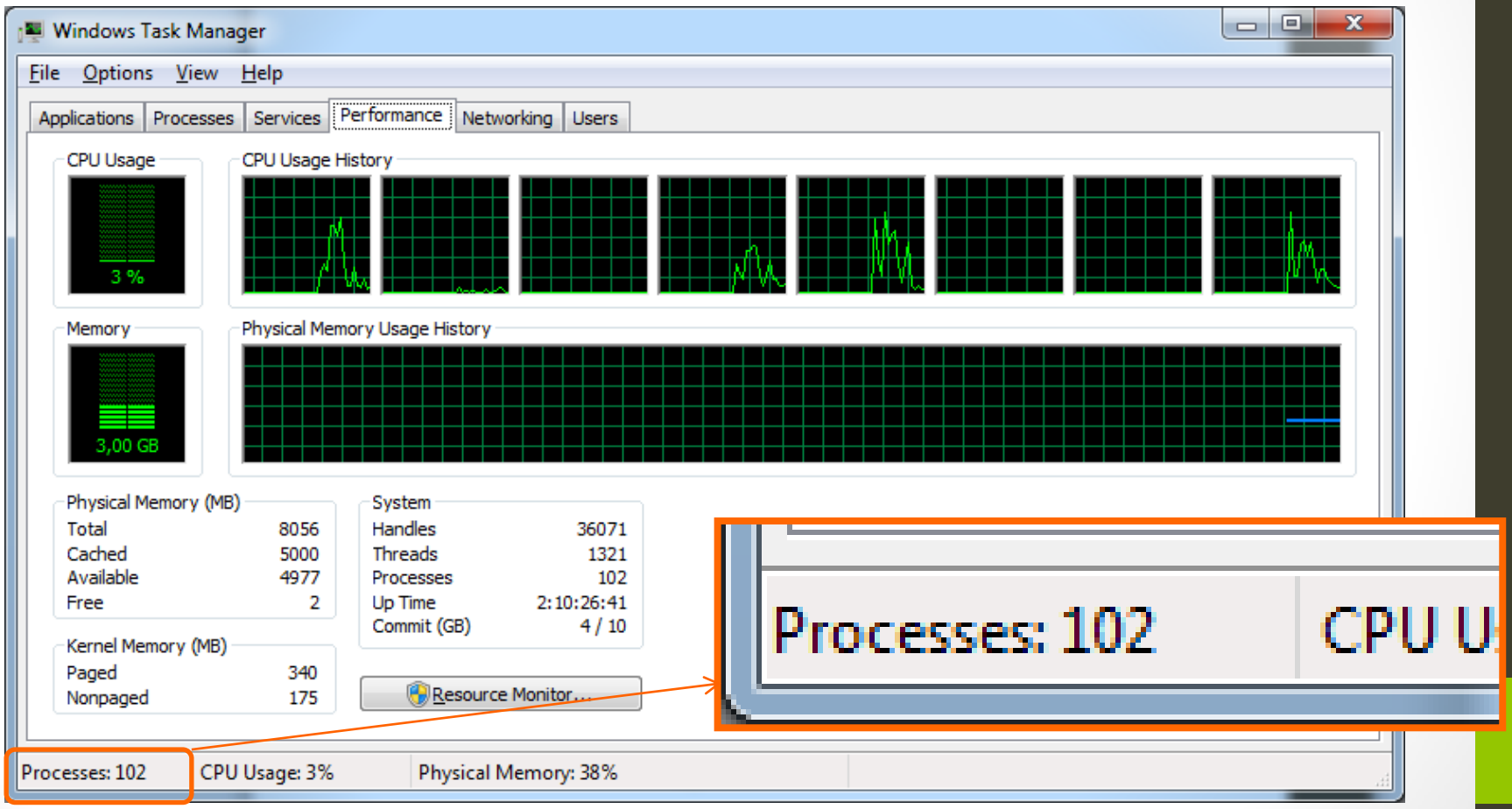
Процес

- Изпълняват се едновременно
- Притежават собствени ресурси (памет, отворени файлове) – недостъпни за останалите процеси.
- Процесите могат да комуникират помежду си посредством средства, предоставени от операционната система

Псевдопаралелизъм



Съвременните ОС



Нишки

- Част от един процес
- Средство за разпаралеляване на програмите (да се изпълнява паралелно повече от една област от кода)
- Споделят общите ресурси на процеса
- Превключването между нишките е по-бързо спрямо процесите
- Всеки процес има поне една нишка в себе си

Основна цел на нишките

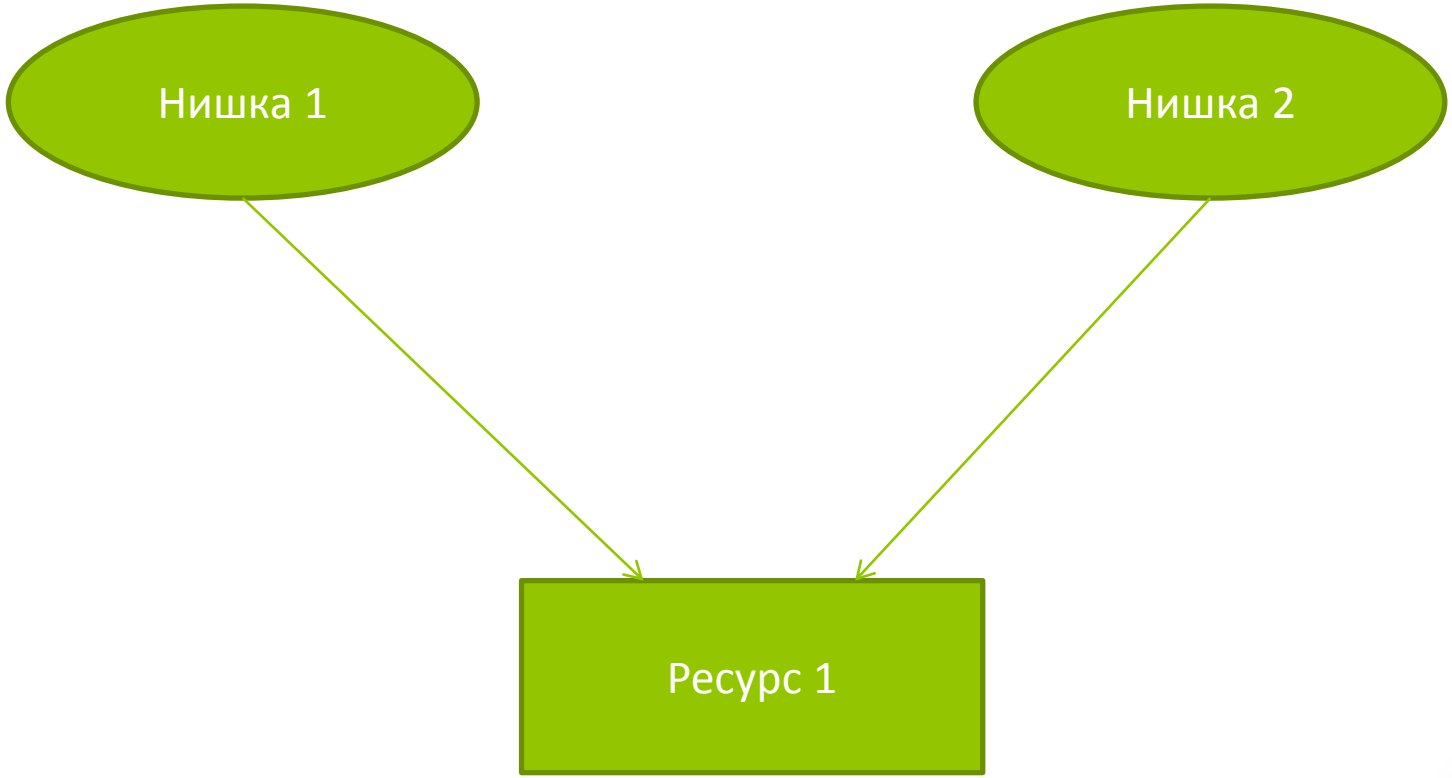
Предимства на нишките

- Осигуряват ускоряване на обработката на информация
- Дават възможност програмата да не „зависне“ при изпълнението на продължителна обработка
- Предоставят възможност за едновременна обработка на няколко външни източници да данни
- Комуникират помежду си посредством променливи

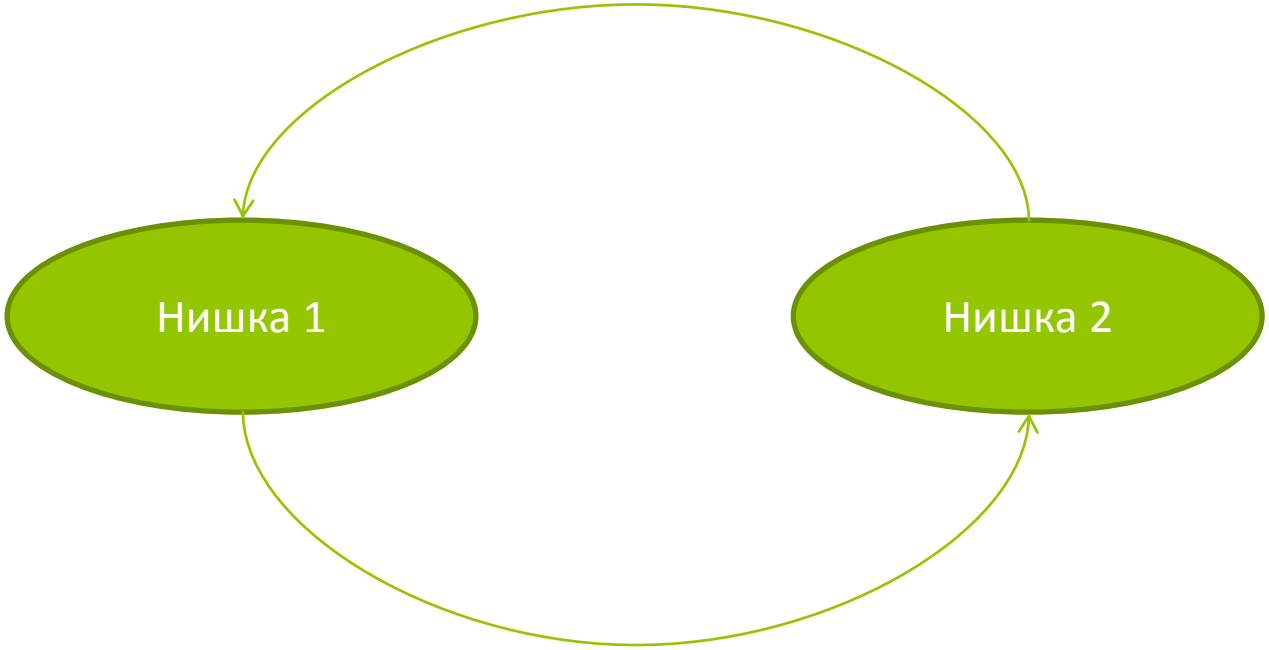
Недостатъци на нишките

- Едновременната работа с един и същи ресурс може да причини „състезание“ на данните
- Повече нишки не означават по-бърза програма
- Трудни са за дебъгване
- Налагат използването на паралелни алгоритми

Състезание на нишките



„Мъртва хватка“

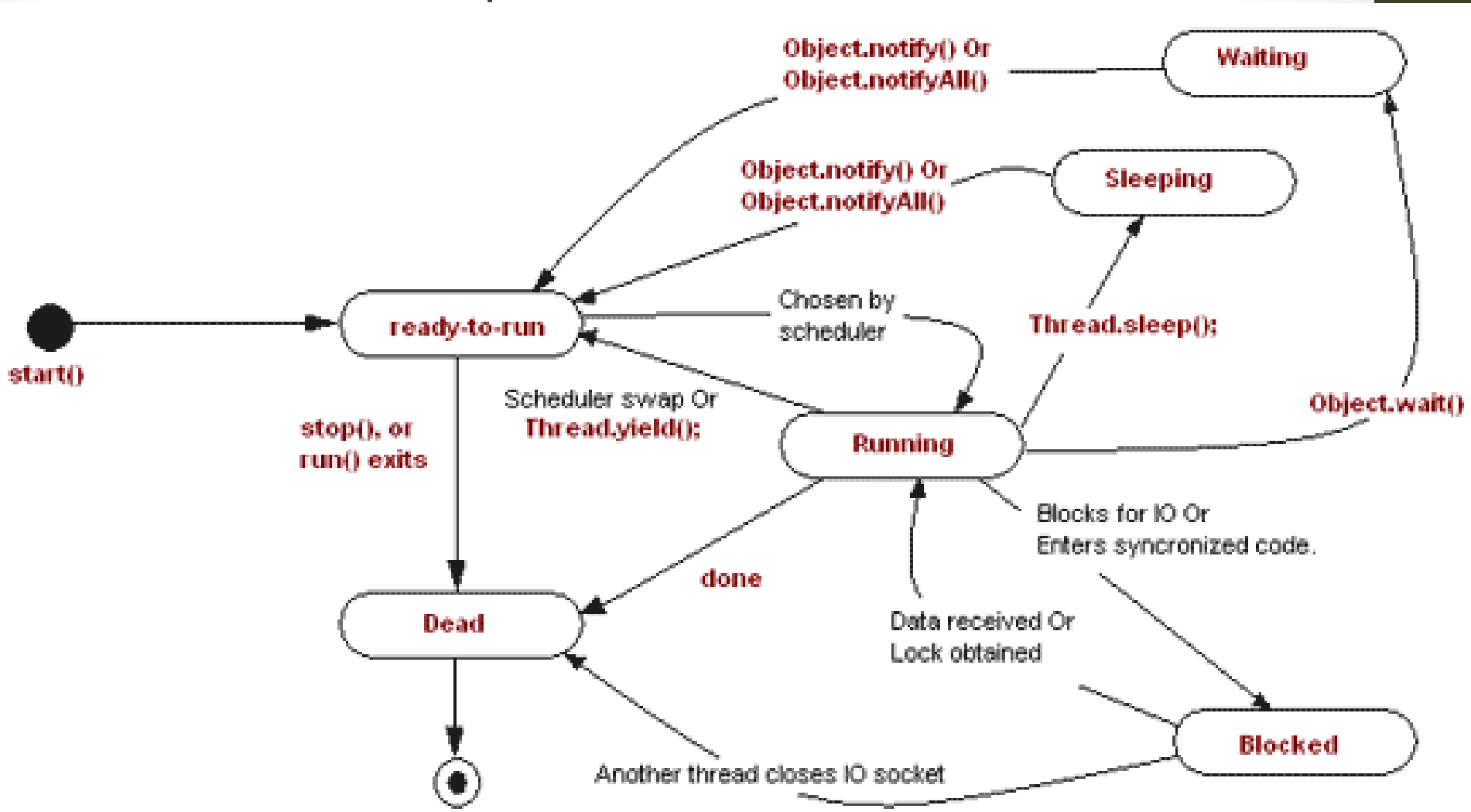


Нишките в Java

- Клас Thread
- Интерфейс Runnable

- Нишките са обекти

Жизнен цикъл на нишка



Създаване на нишка - 1

```
public class MyThread extends Thread
{
	@Override
	public void run()
	{
	…
	}
}
```

Използване:

```
MyThread myThread = new MyThread(…);
myThread.start();
```

Създаване на нишка - 2

```
public class MyRunnable implements Runnable
{
	@Override
	public void run()
	{
	…
	}
}
```

Използване:

```
Thread myRunnable = new Thread(new MyRunnable(…));
myRunnable.start();
```

Операции с нишки

- Thread.yield() – отстъпва изпълнението си на някоя друга нишка
- Thread.sleep(miliseconds) – нишката заспива за зададено време
- anotherThread.join() – изчаква приключването на дадена нишка

Прекъсване на нишка

- anotherThread.interrupt() – дава заявка за прекъсване на дадена нишка
- Прекъсваната нишка може да игнорира прекъсването или да започне процедура по прекратяване дейността си
- Всяка нишка може да проверява дали е прекъсната с:
  - this.IsInterrupted() – само проверява дали някой я е прекъснал
  - Thread.interrupted() – проверява и изчиства състоянието си за прекъснатост

Приоритет на нишките

- Всяка нишка има определен приоритет
  - Thread.MIN_PRIORITY (1)
  - Thread.MAX_PRIORITY (10)
- По-високият приоритет означва, че за изпълнението на нишката ще се изолзва повече процесорно време
- Задаване на приоритет – anotherThread.setPriority(int)

Синхронизация на нишки

- Имат за цел да решават проблемите, породени от състезанието на нишките повреме на работата с данните.

- Пример: банкова сметка

Синхронизация на метод

- Само една нишка може да изпълнява даден метод в даден момент

- Всички останали нишки са заспали и ще бъдат събудени една по една след приключване на метода

- Реализация:

```
synchronized Type method(Parameters)
{
//Изпълнение на работа
}
```

Заключване на обект

- Похват, при който в даден момент даден обект може да се използва само от една нишка.
- Останалите нишки, които искат да ползват този обект, са заключени.
- Заключеният обект се нарича „ключалка“ или монитор
- Използва се в случаи, когато кодът, който трябва да се заключи, е прекалено малък.

Пример за заключване на блок по обект

```
Object obj = …
…
synchronized(obj) //Заключване на обект
{
   //Работа с обекта
} //Отключване на обект
```

Ако нишка се опита да заключи обект, който тя самата е заключила преди това, то тя няма да се само блокира

Проблеми при мониторите

- Мониторите карат нишките да се изчакват по време на работа с даден обект. Това може да доведе нарушаване на идеята за паралелно програмиране.

- Мониторите е препоръчително да се използват за много кратко време. Например в рамките на четене/запис на данни от/в даден обект.

Изчакване и уведомяване

- Използват се за случаи от типа „Производител – консуматор“
- Тук производителят уведомява консуматора посредством методите notify() и notifyAll()
- Консуматорът е в режим на изчакване wait(), когато получи уведомление, продължава работа.
- Методите notify(), notifyAll() и wait() задължително се изпълняват в synchronized блок спрямо обекта на използване.

Пример

```
//Производител:
public void produce()
{
   while(true)
       synchronized(store)
       {
          store.add(new Product());
          store.notify();
       }
}
//Клиент:
public void consume()
{
   while(true)
        synchonized(store)
        {
                 while(store.hasProducts())
                          store.get();
                 store.wait();
        }
}
```

Благодаря за вниманието!