Processi e Thread

Programmi e processi

- Un **programma** è un file eseguibile residente sul disco
- Il **processo** è una istanza di un programma in esecuzione.
  - È l'unità di esecuzione all'interno del S.O.
  - L'esecuzione di un **processo** è solitamente sequenziale
    - le istruzioni vengono eseguite in sequenza, secondo l'ordine specificato nel testo del programma
  - Un S.O. Multiprogrammato consente l'esecuzione concorrente di più processi.

Programma = entità passiva

Processo = entità attiva

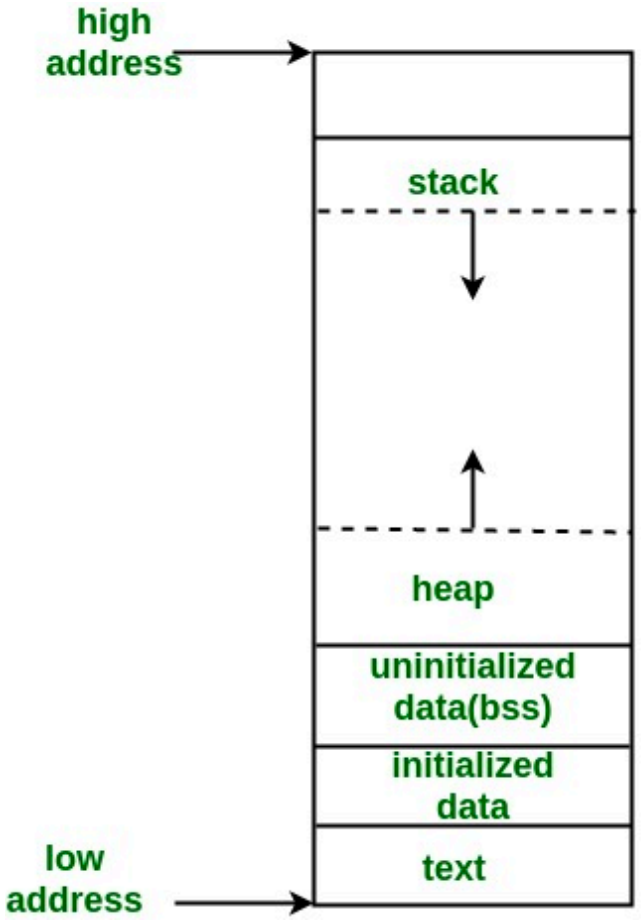
Processi nel dispositivo

- adb shell
  - in “platform-tools”
- ps
  - mostra i processi
- kill 321
  - elimina il processo 321

```
root      24    2     0       0     c004b2c4 00000000 S hid_compat
root      25    2     0       0     c004b2c4 00000000 S rpciod/0
root      26    1     740     248   c0158eb0 afd0d8ac S /system/bin/sh
system    27    1     812     284   c01a94a4 afd0db4c S /system/bin/servicemanager
root      28    1     3736    416   ffffffff afd0e1bc S /system/bin/vold
root      29    1     3716    432   ffffffff afd0e1bc S /system/bin/netd
root      30    1     668     248   c01b52b4 afd0e4dc S /system/bin/debuggerd
radio     31    1     5392    664   ffffffff afd0e1bc S /system/bin/rild
root      32    1     102456  25716 c009b74c afd0dc74 S zygote
media     33    1     22764   3388  ffffffff afd0db4c S /system/bin/mediaserver
root      34    1     812     308   c02181f4 afd0d8ac S /system/bin/installd
keystore  35    1     1616    320   c01b52b4 afd0e4dc S /system/bin/keystore
root      36    1     740     248   c003da38 afd0e7bc S /system/bin/sh
root      37    1     852     352   c00b8fec afd0e90c S /system/bin/qemud
root      39    1     3380    172   ffffffff 0000ecc4 S /sbin/adbd
root      50    36    796     308   c02181f4 afd0d8ac S /system/bin/qemu-props
system    58    32    178504  31012 ffffffff afd0db4c S system_server
app_23    106   32    139364  20432 ffffffff afd0eb08 S jp.co.omronsoft.openwnn
radio     111   32    148008  22868 ffffffff afd0eb08 S com.android.phone
app_25    123   32    146400  24156 ffffffff afd0eb08 S com.android.launcher
system    132   32    137460  19336 ffffffff afd0eb08 S com.android.settings
app_0     150   32    148848  25888 ffffffff afd0eb08 S android.process.acore
app_9     157   32    132052  19124 ffffffff afd0eb08 S com.android.alarmclock
app_39    171   32    131832  17904 ffffffff afd0eb08 S ppl.test.appwidgettest
app_22    178   32    132412  18612 ffffffff afd0eb08 S com.android.music
app_12    186   32    134032  19060 ffffffff afd0eb08 S com.android.quicksearchbox
app_7     194   32    131244  18080 ffffffff afd0eb08 S com.android.protips
app_2     200   32    133820  19424 ffffffff afd0eb08 S android.process.media
app_15    212   32    144984  19932 ffffffff afd0eb08 S com.android.mms
app_30    226   32    135512  20380 ffffffff afd0eb08 S com.android.email
```

Caratteristiche di un processo

- **Spazio di memoria privato**
  - il codice che viene eseguito (text)
  - una zona dati
    - dati non inizializzati, o BSS, e dati inizializzati.
  - uno stack
  - uno heap
- **Process ID**
  - numero intero non negativo
  - PID
- **Risorse**
  - file aperti
  - connessioni di rete
  - accesso a dispositivi
- **Stato**
  - init, ready, running, sleeping

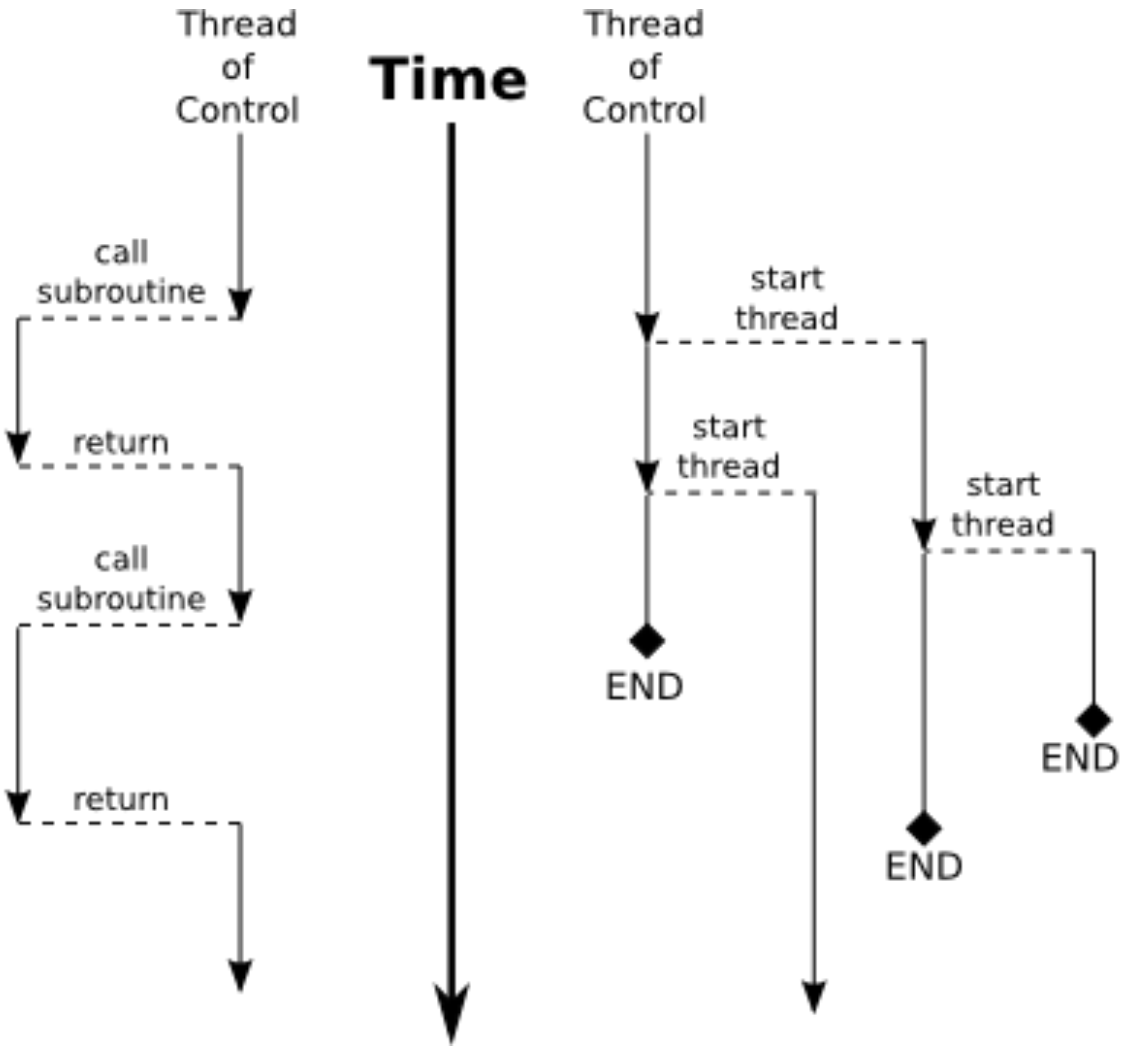


https://www.geeksforgeeks.org/memory-layout-of-c-program/

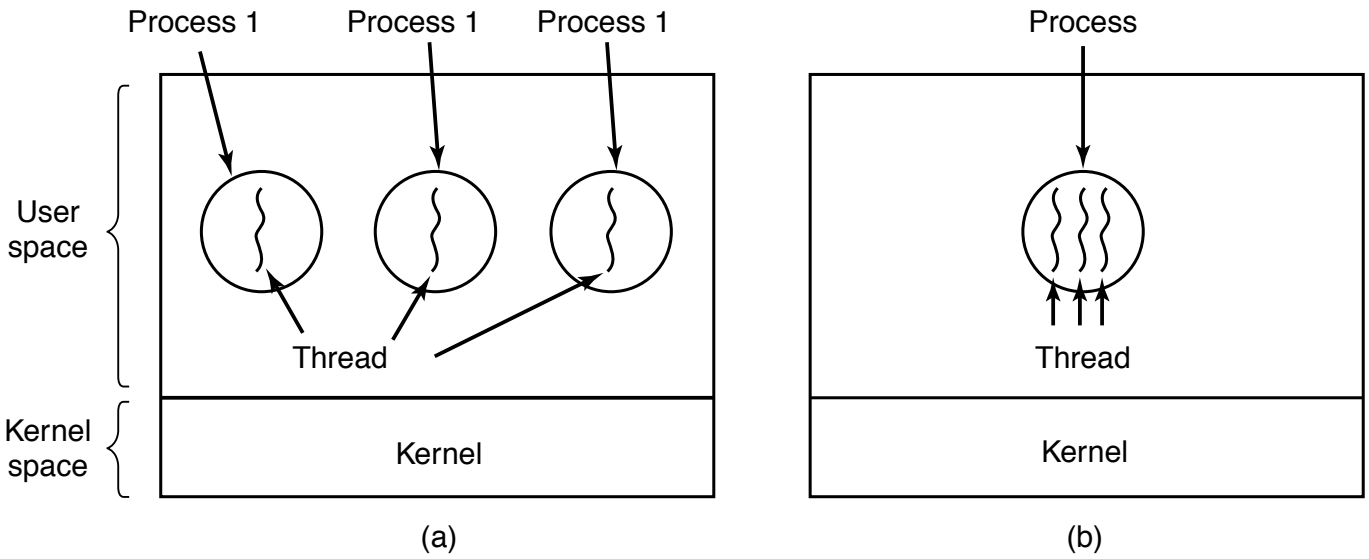
I Thread o Thread of Control

- Un thread rappresenta un flusso di esecuzione all'interno di un processo
  - una sequenza di istruzioni che vengono eseguite una dopo l'altra
- Ogni programma Java ha almeno un thread;
  - creato dalla Java VM quando esegue il programma
  - esegue della routine principale del programma
- Il thread principale può creare altri thread
  - possono continuare anche dopo che il thread principale è terminato.

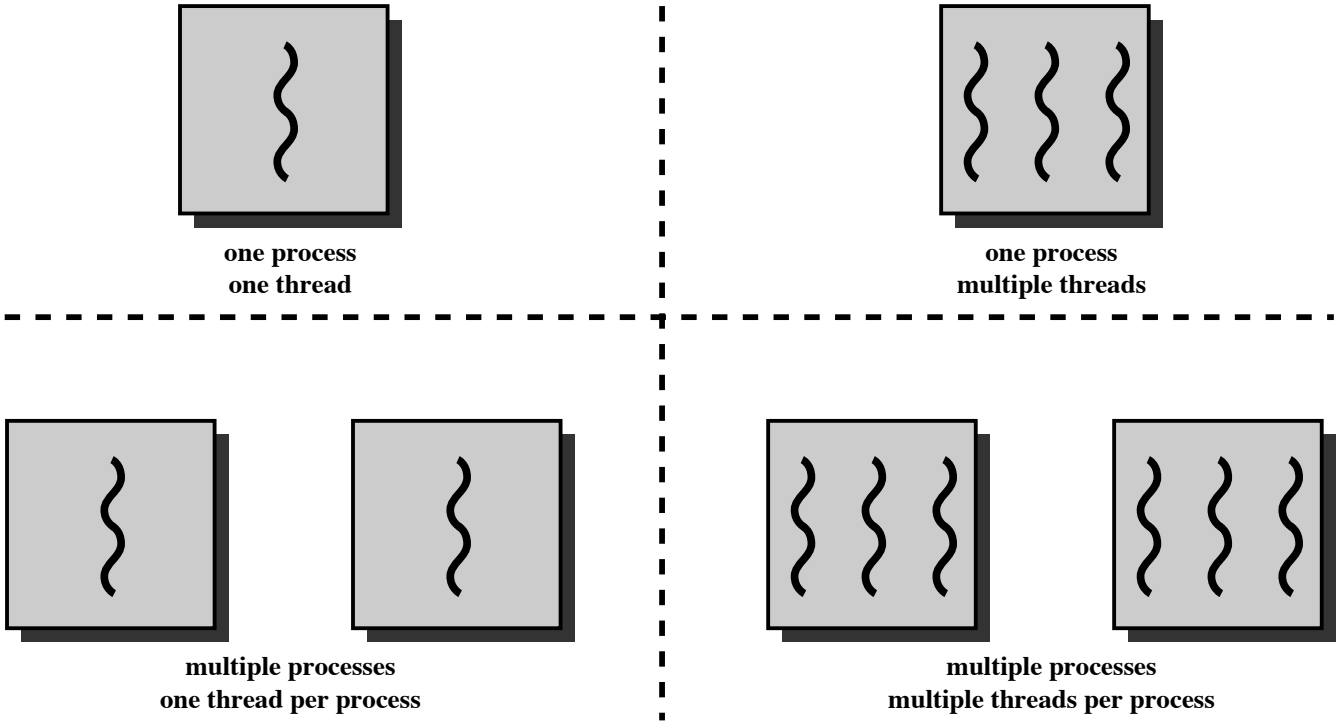
Diversi Thread



Thread e processi

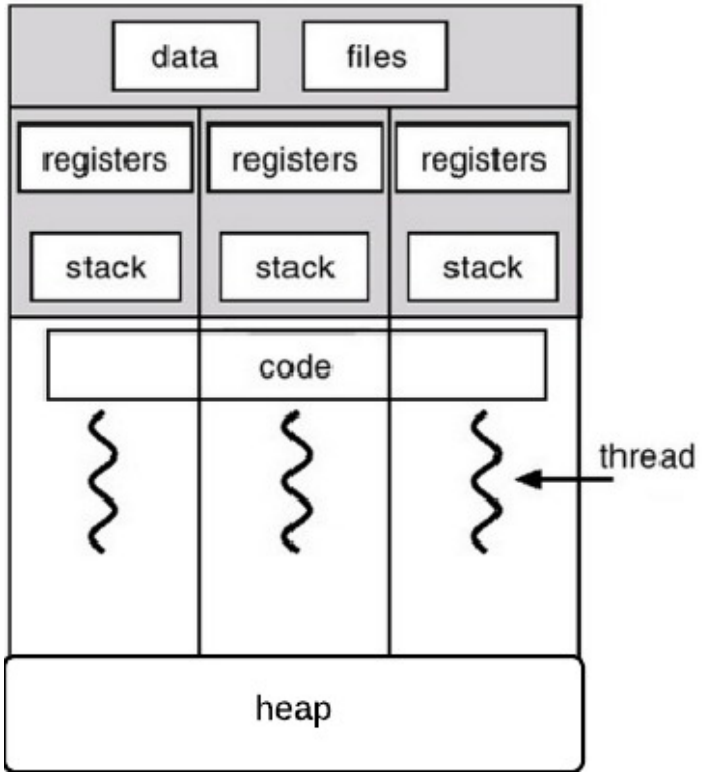


Quattro possibilità



Condivisione delle risorse e della memoria

- I thread di processo condividono le risorse del processo
  - stesso spazio di indirizzamento e stessi dati



Multi-Thread vs Multi-Process

- Vantaggi
  - Maggiore efficienza (ciclo di vita e scheduling)
  - La comunicazione/coordinamento tra thread è più semplice da programmare (rispetto a quella tra processi)
-
- Svantaggi:
  - Maggiore complessità di progettazione e programmazione
    - i processi devono essere “pensati" paralleli
    - sincronizzazione tra i thread
    - gestione dello scheduling tra i thread può essere demandato all'utente

THREAD E JAVA

Creare Thread in JAVA

- In Java i thread sono classi che implementano il metodo **run()** che diventa la nuova “main”
  - il metodo non va chiamato direttamente, ma lo deve fare il SO
- Per scrivere il metodo run() ho due possibilità:
  - estendere la classe Thread
  - implementare l’interfaccia Runnable
- Finito run() il thread muore
  - Non è possibile “resuscitarlo”, è necessario crearne uno nuovo

Esempio

```
class MyThread extends Thread {      // il thread
   public void run() {
       System.out.println(" thread running ... ");
   }
}

class ThreadEx {
   public static void main(String [] args  ) {
       MyThread t = new MyThread();
       t.start();
   }
}
```

Esempio

```
class MyThread implements Runnable{
   public void run()
   {
       System.out.println(" thread running ... ");
   }
}


class ThreadEx {
        public static void main(String [] args  ) {
       Thread t = new Thread(new MyThread());
       t.start();
   }
}
```

Ping PONG

```
public class PingPong extends Thread {
    private String parola;
    private long ritardo;

    PingPong(String cosaDire, long attesa) {
        parola = cosaDire;
        ritardo = attesa;
    }

    public void run() {
        try {
            for (;;) {
                System.out.println(parola + " ");
                Thread.sleep(ritardo);
            }
        } catch (InterruptedException e) {
            return;
        }
    }

    public static void main(String[] args) {
        new PingPong("ping", 3330).start();
        new PingPong("PONG", 1000).start();
    }
}
```

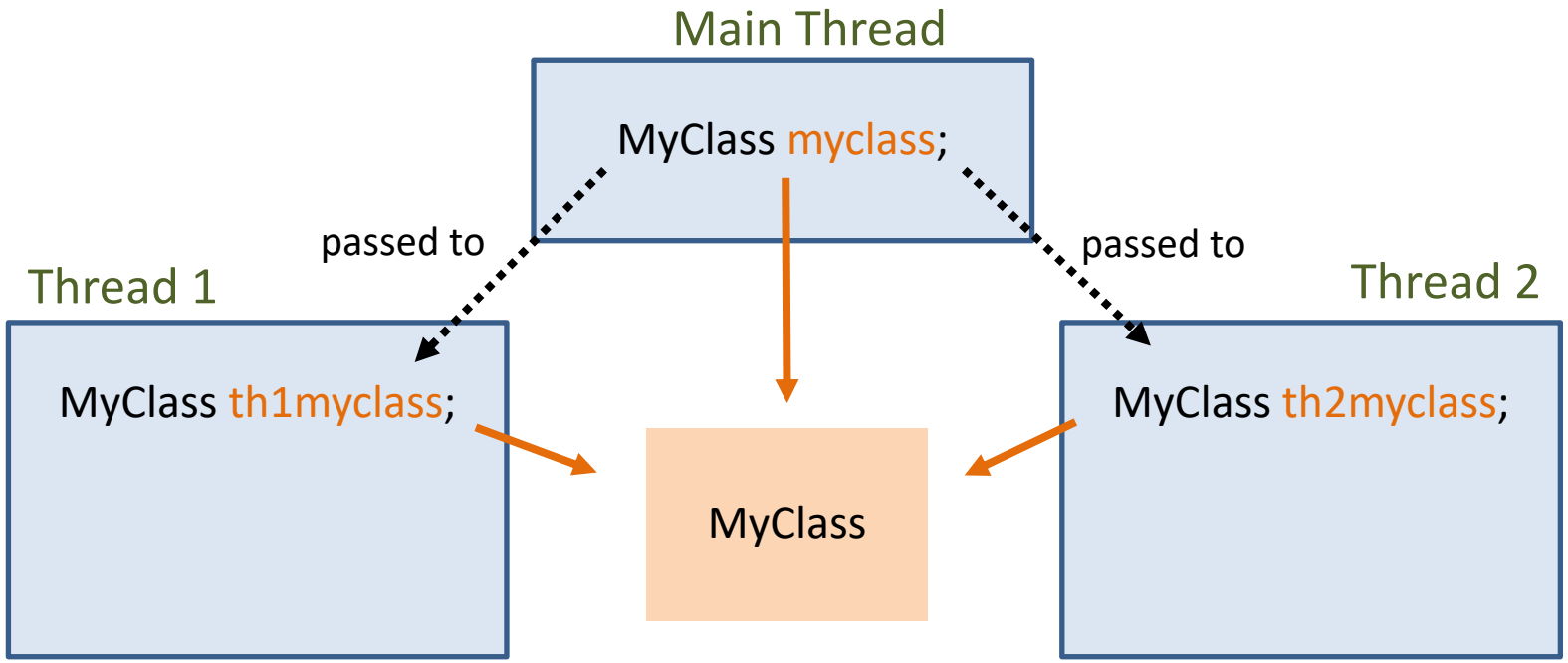
Riferimenti a Thread

- Nomi di thread
  - È possibile dare un nome ad un thread in due modi:
    - tramite un parametro di tipo String al **costruttore**
    - come parametro del metodo **setName**()
  - Il nome di un thread è solo per comodità del programmatore
    - java non lo utilizza

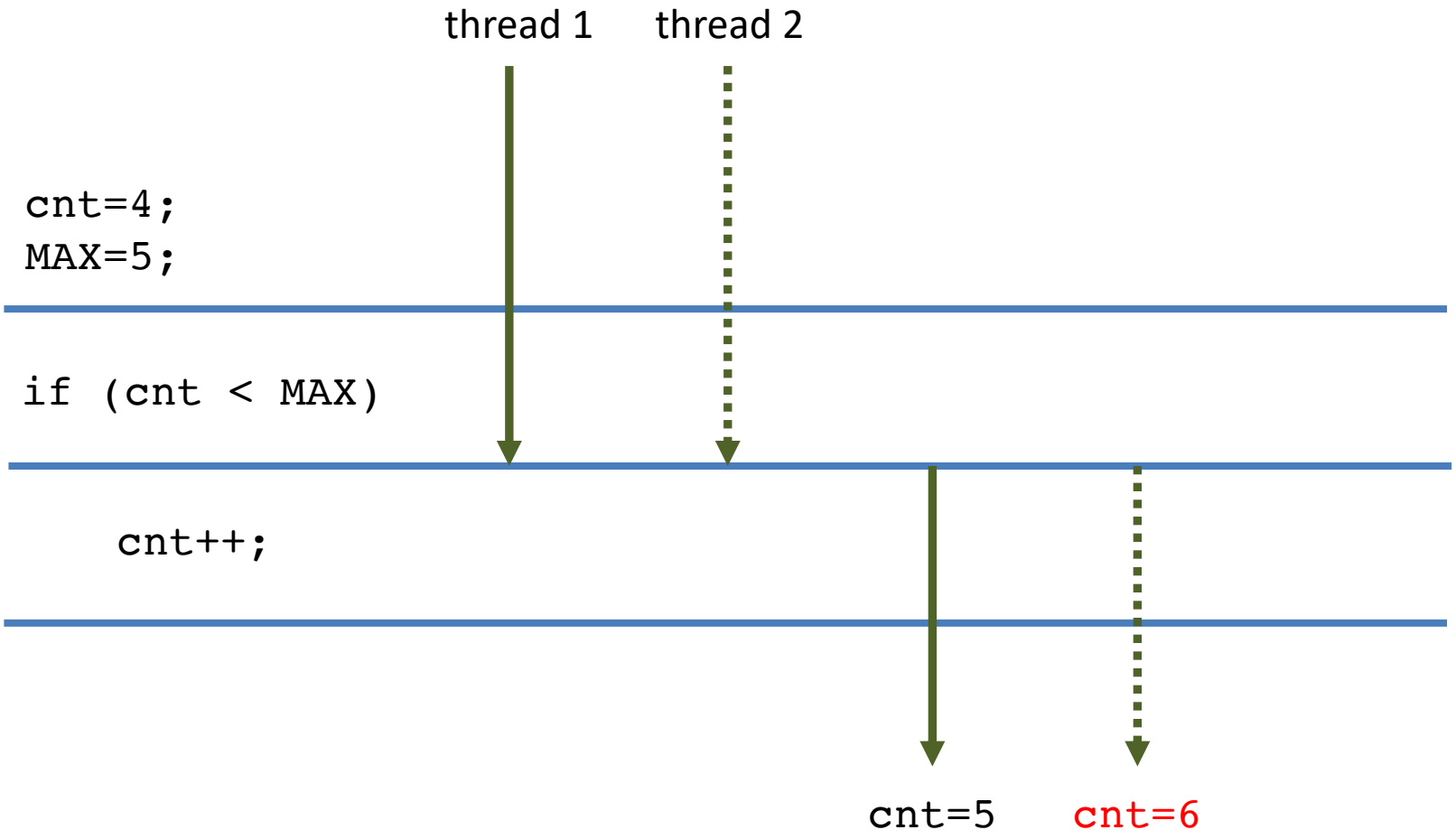
A volte è necessario poter intervenire su un thread in esecuzione senza però conoscere a priori quale esso sia:

- Thread.currentThread()
  - Metodo statico che ritorna un oggetto di tipo Thread
    - il riferimento al thread correntemente in esecuzione

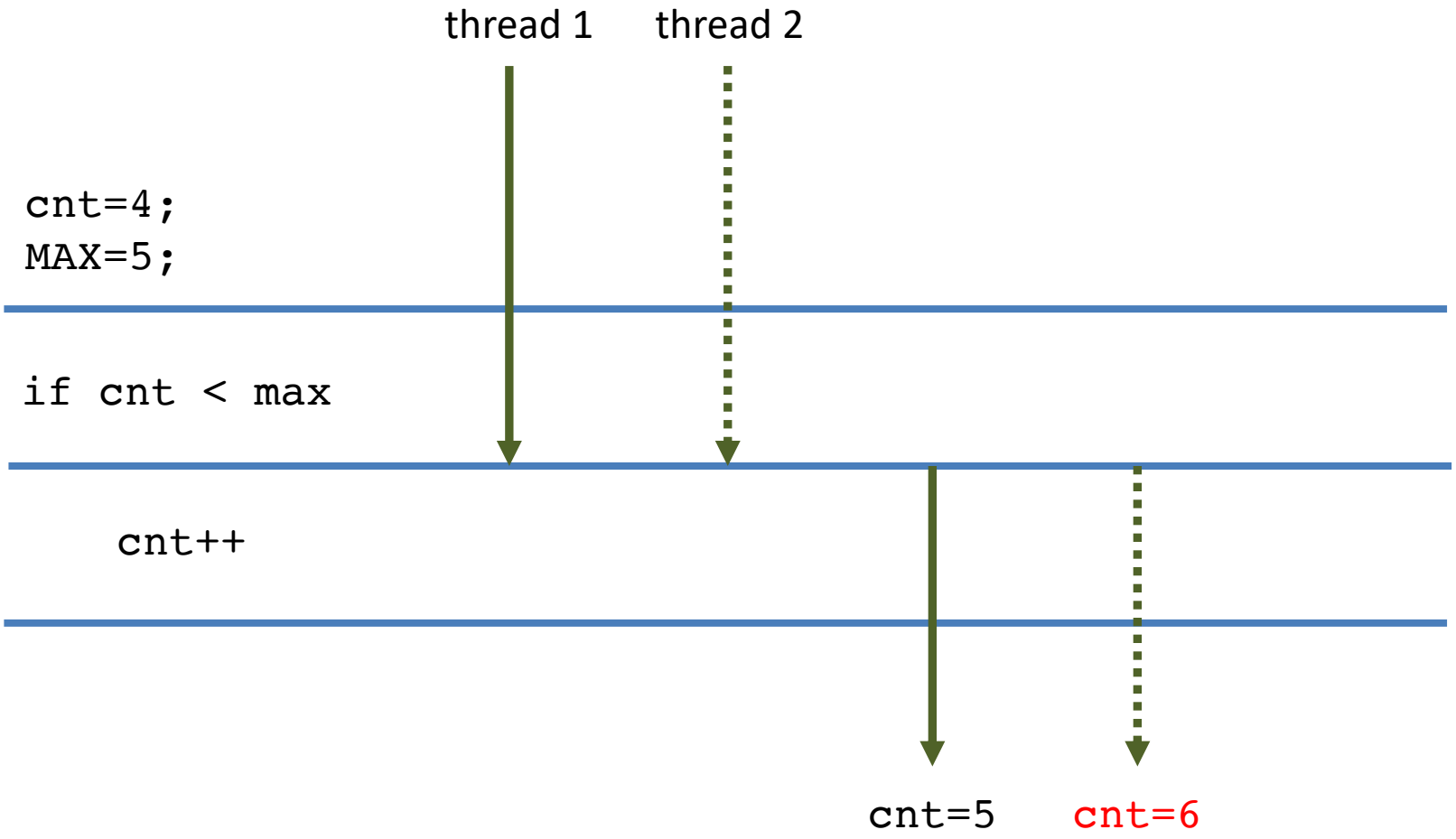
Thread ed valori condivisi



Accesso a valori condivisi



Accesso a valori condivisi



Non è garantito che un thread esegua
una sequenza di operazioni senza essere interrotto!!

Sezioni Critiche

- Gruppi di istruzioni eseguiti in modo escusivo da un solo Thread
  - modificatore synchronized
- Metodi

```
synchronized int incrCnt(){
        if(cnt<MAX)
                cnt++
}
```

- Blocchi di codice

```
Object o = new Object();
...
synchronized (o) {
        if(cnt<MAX)
                cnt++

}
```

Quando un thread entra in una sezione critica, gli altri thread che provano ad accedere la stessa sezione sono bloccati in una coda

Coordinare Thread

- La sincronizzazione realizza la modalità di accesso in mutua esclusione

- Non ci sono garanzie sull'ordine di accesso alla sezione critica
  - Per stabilire un particolare ordine di esecuzione (o di accesso) dei thread è necessario implementare delle strategie che coordinano le attività mediante metodi dipendenti dall'applicazione
  - L'uso dei livelli di priorità non è sufficiente da solo a stabilire un ordine di accesso e quindi diversi thread con la stessa priorità accedono in modo casuale

Comunicazione fra thread

- I metodi di sincronizzazione della classe Object:
  - wait() – mette un thread in attesa
  - notify() – sveglia un thread in attesa
  - notifyAll() – sveglia tutti i thread in attesa

- Quando wait() mette un thread in attesa di un evento si aspetta che un altro thread per notificare l'evento invochi notify() sullo stesso oggetto su cui si effettua l'attesa

- Più thread possono essere in attesa sullo stesso oggetto e un notify può risvegliarne uno qualsiasi, senza ordine per questo si usa in genere risvegliarli tutti con un notifyAll

Una coda

```
class Queue {
  private Element head, tail;

  public synchronized Object get(){
      try {
            while (head == null) wait();
      }catch(InterruptedException e){
            return null;
      }
      Element p = head;
      head = head.next;
      if (head == null)
            tail = null;
      return p.item;
  }
```

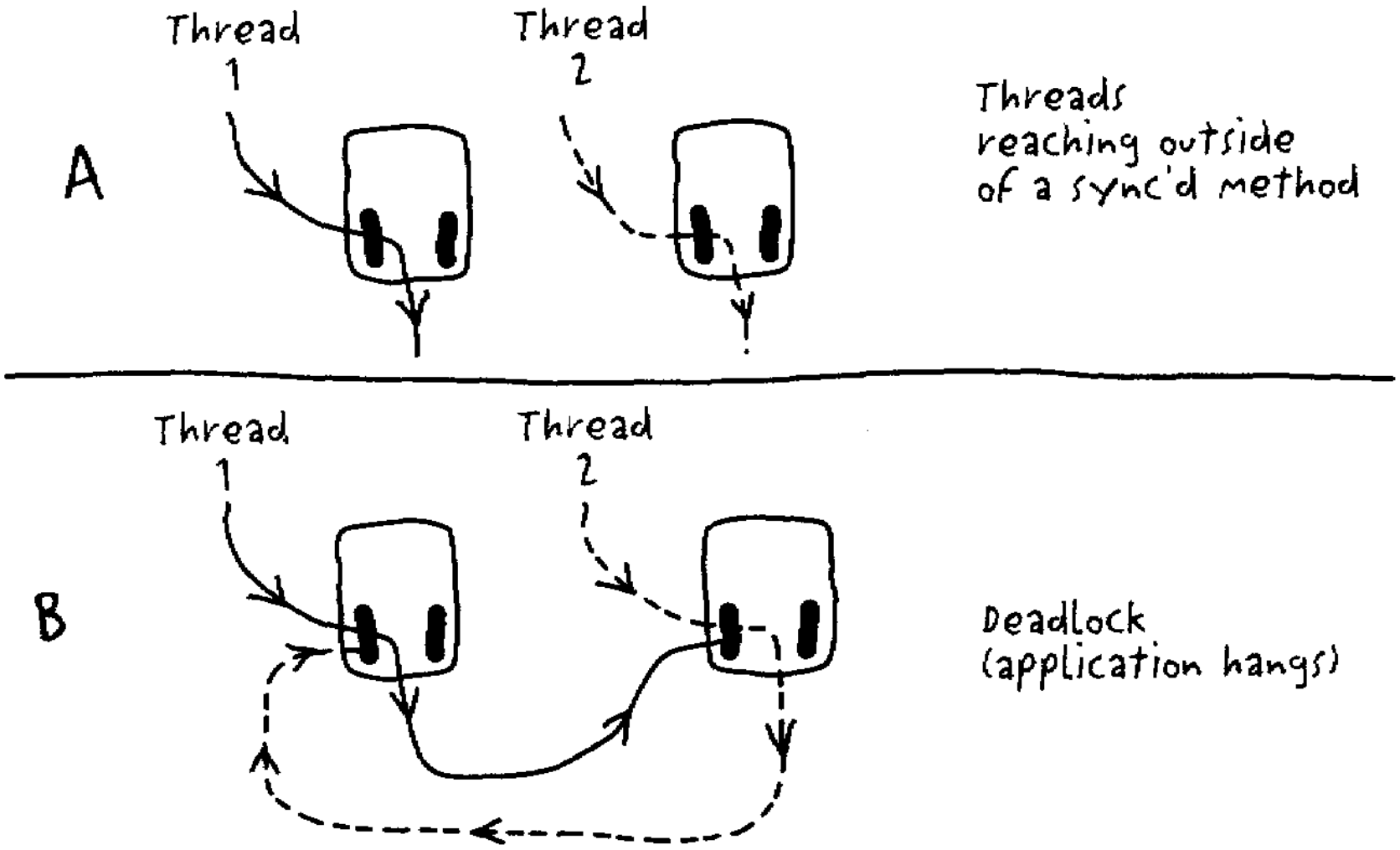
…

Una coda

…

```
   public synchronized void append(Object obj) {
       Element p = new Element(obj);
       if (tail == null)
             head = p;
       else
             tail.next = p;
       p.next = null;
       tail = p;
       notifyAll();
   }
}
```

DeadLock


Thread 1
Thread 2
A
Threads reaching outside of a sync'd method
Thread 1
Thread 2
B
Deadlock (application hangs)

Controllo del tempo

```
Runnable threadTask = new Runnable() {
        public void run() {
                while(true){
                        try {
                                Thread.sleep(2000);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                        doSomething();
                }
        }
};
…
(new Thread(threadTask)).start();
```

Timer

```
…
TimerTask timerTask = new TimerTask() {
        @Override
        public void run() {
                doSomething();
        }
};
…
Timer timer = new Timer();
…
timer.schedule(timerTask, 2000,10000); // delay, period
```

Android e processi e thread

- Quando un componente di un app parte e **non** ci sono altri componenti in esecuzione, Android crea un nuovo processo Linux per l'app con un singolo thread di esecuzione
- Di default tutti i componenti di una applicazione sono eseguiti nello stesso processo e thread (main thread)
- Quando un componente di un app parte e ci sono altri componenti in esecuzione, allora il componente è eseguito nel processo già avviato ed usa lo stesso thread di esecuzione
  - è comunque possibile eseguire componenti diversi in processi separati ed è possibile creare nuovi thread per ogni processo

Attributo process

- <activity>, <service>, <receiver>, e <provider> supportano l'attributo process che serve a specificare **il nome** del processo in cui il componente deve essere eseguito

```
<activity android:name="SecondActivity" android:process=":new_process" />
```

- Configurazioni
  - Un processo per i componenti di una app
  - Un processo per componente
  - Più app possono condividere lo stesso processo
    - se usano lo stesso user id e lo stesso certificato
- Per impostare un comportamento unico per tutti i componenti di una app si può applicare l'attributo android:process al tag <application>

Importanza dei processi

- I processi sono ordinati per importanza a seconda dei componenti che eseguono e dello stato di questi
- Livelli:
  - Foreground process
    - processo con cui l'utente sta interagendo (direttamente o indirettamente)
  - Visible process
    - è in vista ma non totalmente (e.g. c'è un dialog sopra)
  - Service process
    - Un servizio in esecuzione
  - Background process
    - Una activity nascosta
  - Empty process
    - un processo che non ha componenti in esecuzione

- Se più componenti sono in esecuzione il processo assume l'importanza di quello massimo
- Se un processo dipende da un altro questo può aumentare l'importanza

Thread e Android

- Come in java!!!

Esempio:

```
public void onClick(View v) {
   new Thread(new Runnable() {
      public void run() {
         Bitmap b = loadImageFromNetwork("http://example.com/image.png");
         mImageView.setImageBitmap(b);
      }
   }).start();
}
```

Vedremo che non faremo MAI così!!!

AsyncTask

```
class MyAsyncTask extends AsyncTask<Integer, String, Long> {
        protected void onPreExecute() {
                                ...
        }
        protected Long doInBackground(Integer... params) {
                                ...
        }
        protected void onProgressUpdate(String... values) {
                                ...
        }
        protected void onPostExecute(Long time) {
                                ...
        }
}
```

new MyAsyncTask().execute()

```
...
onPreExecute()
...
doInBackground(Integer... params)
        publishProgress(...);
...
onProgressUpdate(String... values)
...
onPostExecute(Long time)
...
```

Esempio

```
class MyAsyncTask extends AsyncTask<Integer, String, Long> {
        protected Long doInBackground(Integer... params) {
                long start = System.currentTimeMillis();
                for (Integer integer : params) {
                        publishProgress("start processing "+integer);
                        doLongOperation();
                        publishProgress("done processing "+integer);
                }
                return start - System.currentTimeMillis();
        }
        public void doLongOperation() {
                try { Thread.sleep(1000); } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
...
```

Esempio

```
        ...
        protected void onProgressUpdate(String... values) {
                updateUI(values[0]);
        }
        protected void onPostExecute(Long time) {
                updateUI("Done with all the operations, it took:" + time
+ " millisecondes");
        }
        protected void onPreExecute() {
                updateUI("Starting process");
        }
}
```

Cancellare AsyncTask

```
protected String doInBackground(String... params) {
   for(int i=0;i<100;i++){
      if(isCancelled()){
         break;
      }
      try{Thread.sleep(200);}catch(InterruptedException ie){}
      if(isCancelled()){
         break;
      }
      publishProgress(i);
      if(isCancelled()){
         break;
      }
   }
   return "risultato";
}
```