https://pixabay.com/vectors/magnifying-glass-magnifier-glass-189254/
https://zh.wikipedia.org/wiki/Linux

作業五：pthread的建立和等待

中正大學 作業系統實驗室
指導教授：羅習五

作業目標

- Pthread是POSIX所定義的執行緒模型，大部分的作業系統都支援
- 許多高階程式語言，底層使用pthread與作業系統核心的「執行緒模型」進行對接
  - 例如：java的jvm
  - 例如：OpenMP函數庫
- 了解平行運算的基本操作

建立和等待thread

```
1.  int main(int argc, char **argv) {
2.      //取得系統中的core的數量（包含v-core）
3.      numCPU = sysconf( _SC_NPROCESSORS_ONLN );
4.      //POSIX的threadID定義為pthread_t，這個不見得是整數，他也可以是資料結構
5.      //不要直接存取pthread_t
6.      pthread_t* tid = (pthread_t*)malloc(sizeof(pthread_t) * numCPU);
7.      //列印主程式的process id和threadID，threadID是Linux專有的
8.      printf("I am main funciton, my pid is %ld and my tid is %ld\n",
9.          (long)getpid(), gettid());
10.     printf("waiting for child threads\n");
11.     //產生執行緒，執行緒的起始函數為thread（自己取名字），可以傳給他一個64/32位元的參數
12.     for(long i=0; i< numCPU; i++)
13.         pthread_create(&tid[i],NULL,(void *) thread, (void*)i);
14.     //等待執行緒執行結束，請特別注意，任意一個thread可以等待對方，不一定要產生的執行緒去等
15.     for(int i=0; i< numCPU; i++)
16.         pthread_join(tid[i],NULL);
17.     printf("all threads finish their work\n");
18. }
```

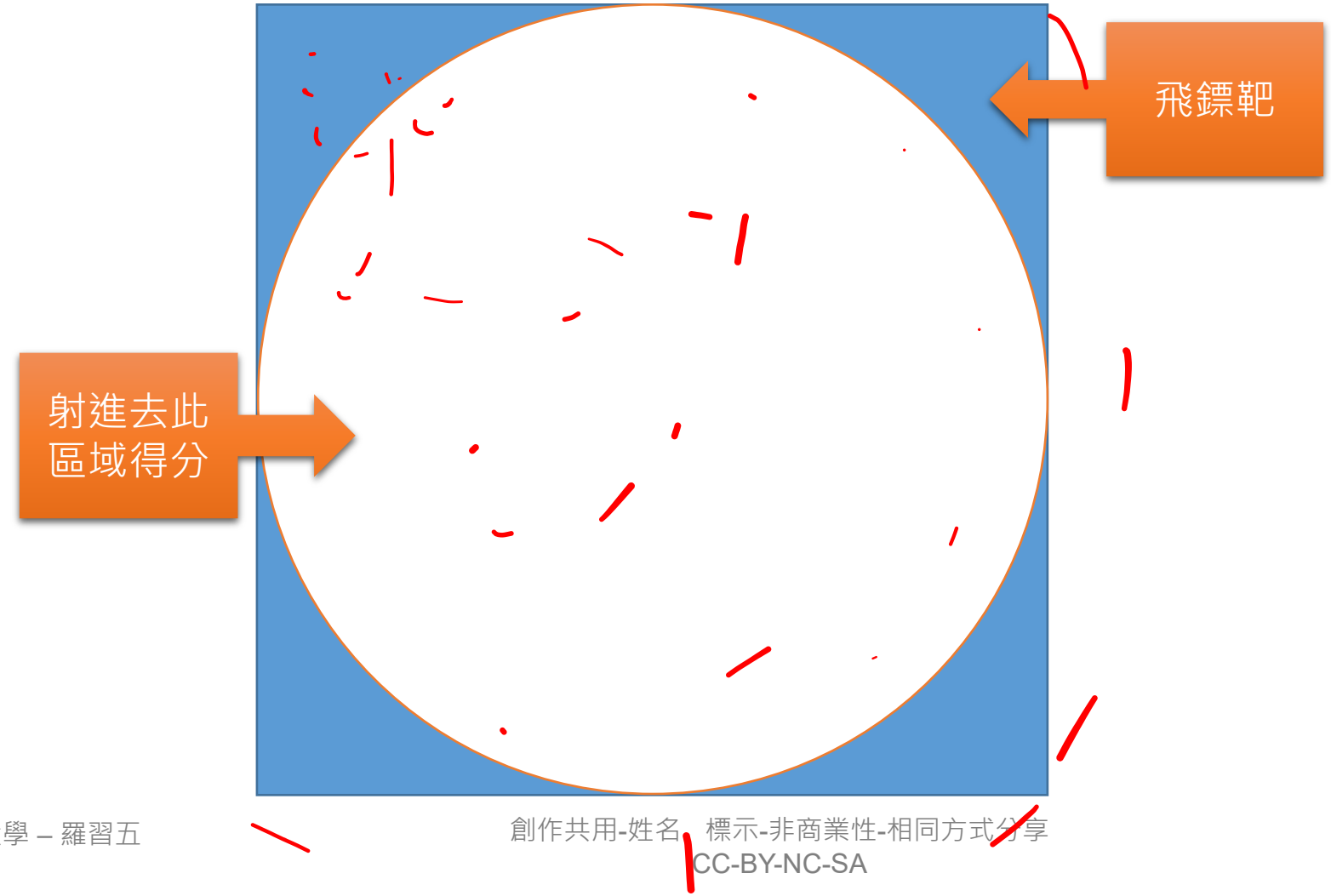
建立和等待thread

```
19.  __thread char name[100];  //前面冠上__thread字眼的都屬於thread local storage
20.  __thread int threadID = -1;
21.  void thread(void *givenName) {
22.      int givenID = (intptr_t)givenName; //小技巧，指標(64b)轉整數(32b)要用(intptr_t)才不會有警告
23.      threadID = gettid();   //設定值給TLS中的threadID
24.      printf("\tthread %02d is here.\n", givenID);
25.      //印出pid和tid，注意，threadID雖然是global變數，但因為它屬於TLS，因此每個thread所存取的threadID是不一樣的
26.      printf("\tmy pid is %ld and my tid is %ld\n", (long)getpid(), threadID);
27.      sprintf(name, "###%02d###", givenID);
28.      sleep(1);
29.      //同樣的name也屬於TLS
30.      printf("\tmy name is %s\n", name);
31.  }
```

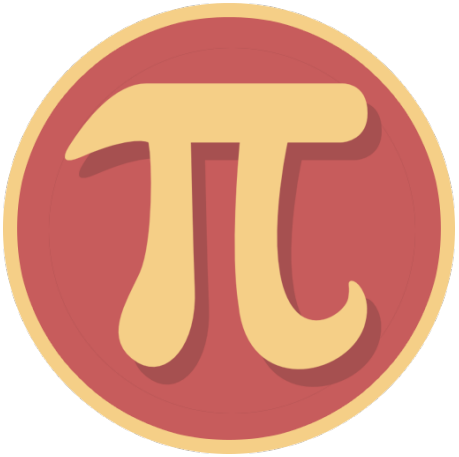
輸出結果

```
I am main funciton, my pid is 73691 and my tid is 73691
waiting for child threads
      thread🦖00 is here.
      my pid is 73691 and my tid is 73692
      thread🦖02 is here.
      my pid is 73691 and my tid is 73694
//省略一些輸出
      my name is 🦖###13###
      my name is 🦖###06###
      my name is 🦖###04###
all threads finish their work
```

計算pi

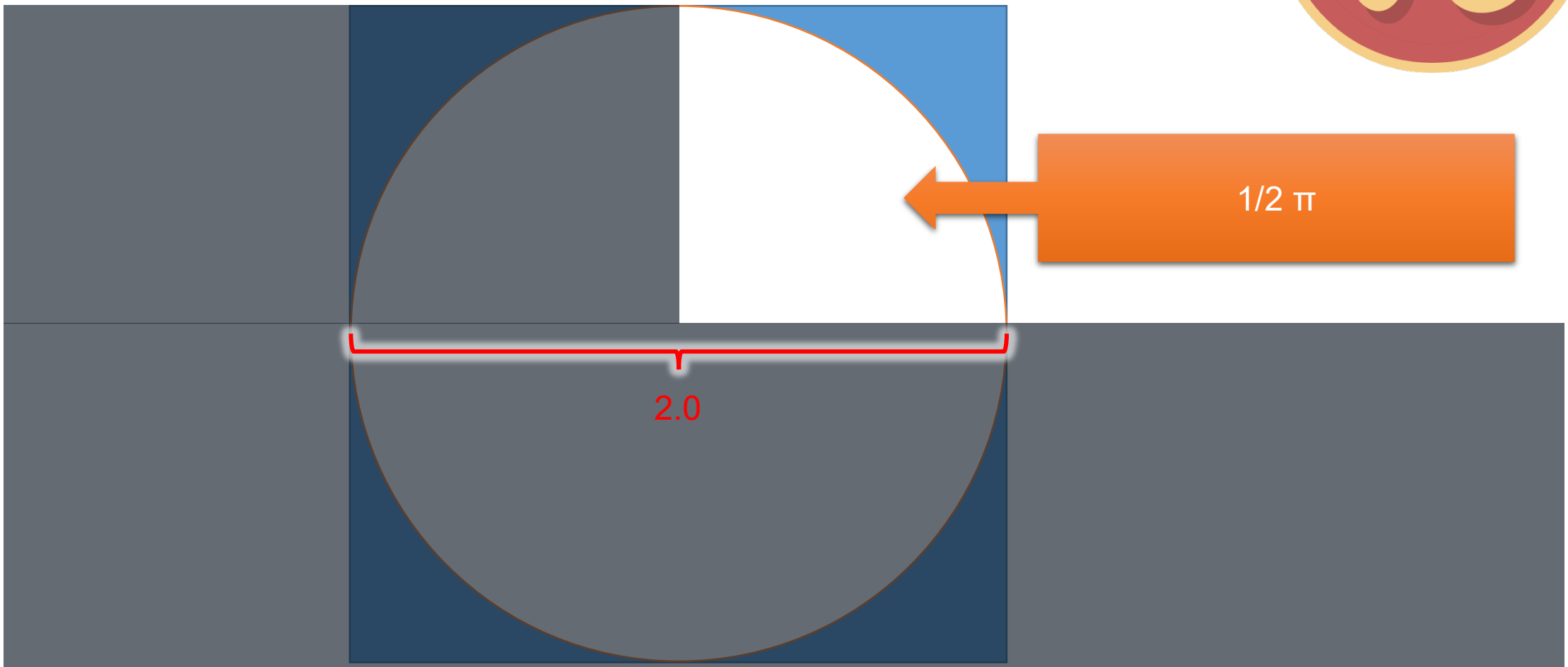


計算pi

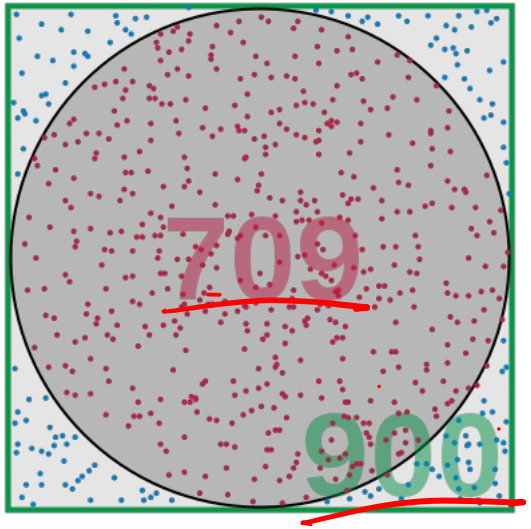


- 在亂射的情況下，飛鏢如果在靶內，得分的機率有多高?
  - 假設靶的長寬都是「二」公尺
  - 「目標」是一個圓，圓的直徑是二公尺。
  - 「圓面積」為 $2 \times \pi \times r^2 = 2\pi$ （平方米）
  - 靶面積為「1」，因此得分的機率是：「$\pi/2$」

計算pi



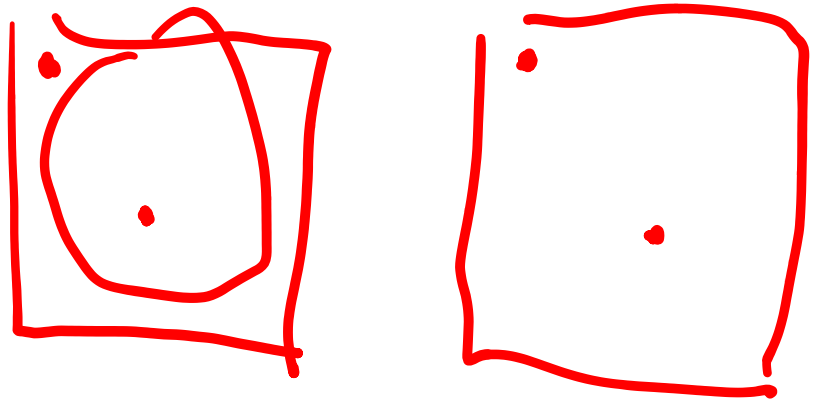
蒙特卡羅法(飛鏢逼近)



- 「積分」的時候可以使用這個方法
- https://zh.wikipedia.org/wiki/%E8%92%99%E5%9C%B0%E5%8D%A1%E7%BE%85%E6%96%B9%E6%B3%95
- https://zh.wikipedia.org/wiki/%E5%9C%86%E7%9A%84%E9%9D%A2%E7%A7%AF

計算pi的執行緒

```
1.    long score[100];
2.    void thread(void *givenName) {
3.        int id = (intptr_t)givenName;
4.        struct drand48_data r_data;
5.        double x, y, dist;
6.        //底下這一行是除錯用，我要確認每個thread的random stream是完全不同的
7.        srand48_r((long)givenName, &r_data);
8.        for (int i=0; i< loopCount; i++) {
9.            drand48_r(&r_data, &x);
10.           drand48_r(&r_data, &y);
11.           dist = sqrt(x*x + y*y);
12.           if (dist < 1)
13.               score[id]++;
14.       }
15.   }
```



計算pi的主程式

```
1.  int main(int argc, char **argv) {
2.      exename = argv[0];
3.      numCPU = sysconf( _SC_NPROCESSORS_ONLN );
4.      pthread_t* tid = (pthread_t*)malloc(sizeof(pthread_t) * numCPU);
5.      for (long i=0; i< numCPU; i++)
6.          pthread_create(&tid[i], NULL, (void *) thread, (void*)i);
7.      for (int i=0; i< numCPU; i++)
8.          pthread_join(tid[i], NULL);
9.      long total=0;
10.     for (int i=0; i< numCPU; i++) {
11.         total += score[i];
12.     }
13.     //這一行讓我知道飛鏢射中幾次，除錯用
14.     printf("%ld\n", total);
15.     printf("pi = %f\n", ((double)total)/(loopCount*numCPU)*4);
16. }
```

```
./pi-random

0.170828, 0.041630, 0.912433, 0.783235, 0.654037, 0.395642, 0.524840,
0.266444, 0.137247, 0.008049, 0.878851, 0.749653, 0.620456, 0.491258,
0.362060, 0.232863, 1256644909

pi = 3.141612
```

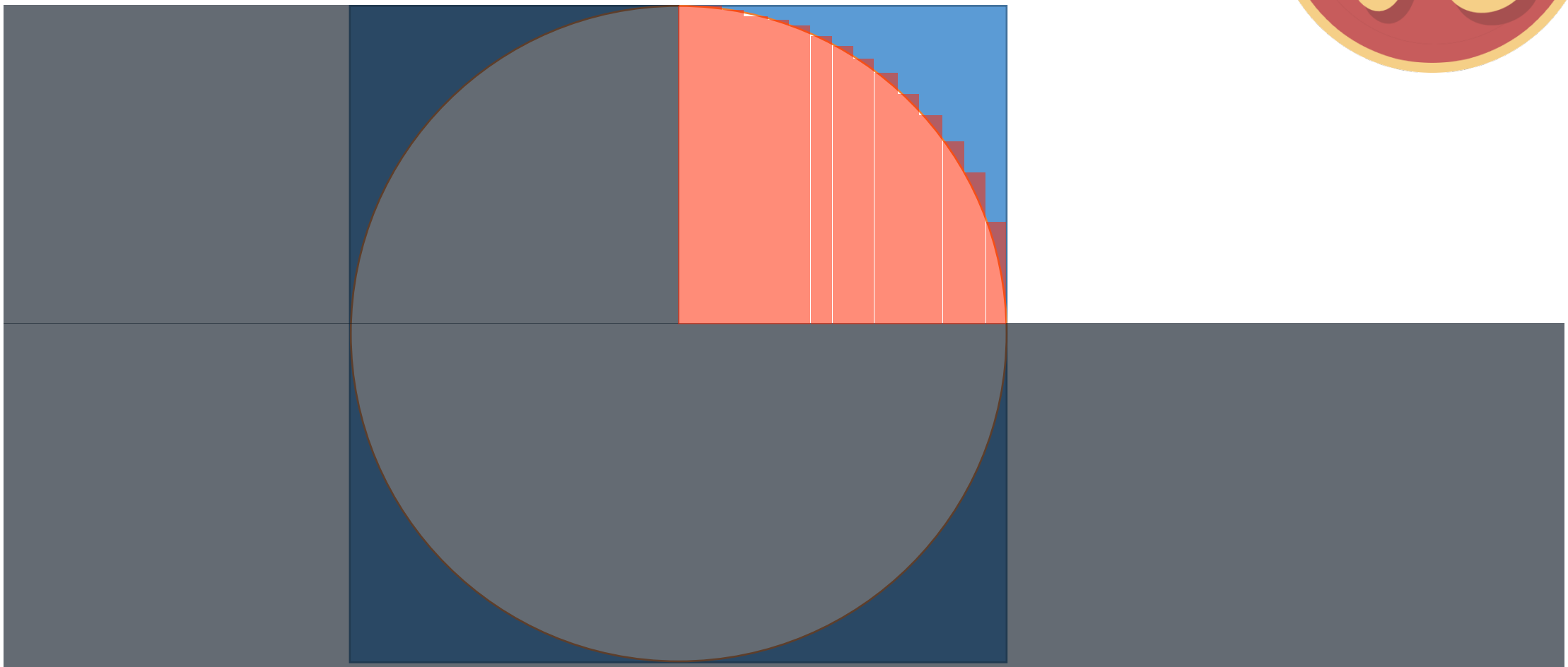
```
time ./pi-random

0.170828, 0.912433, 0.041630, 0.783235, 0.654037, 0.524840, 0.395642,
0.266444, 0.137247, 0.008049, 0.878851, 0.749653, 0.620456, 0.491258,
0.362060, 0.232863,

1256644909

pi = 3.141612

./pi-random  155.62s user 0.08s system 1570% cpu 9.914 total
```

使用time量測時間，發現pi總共執行了155.62的「CPU秒」，因為共有16核心，因此換算成「人間秒」為9.73秒，系統使用率為1590%，因為16個處理器，因此「百分比使用率為」99.3%

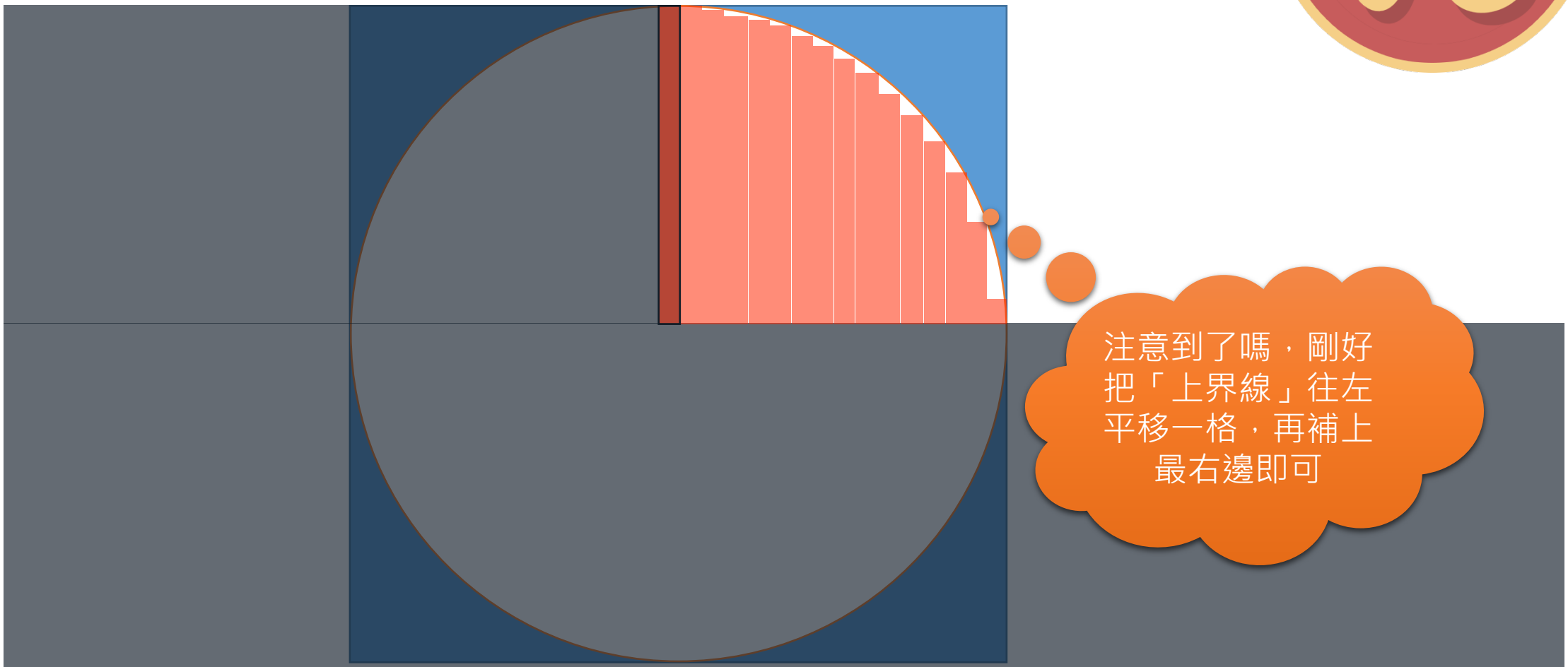
作業系統概論基於GNU/Linux

中正大學，資工系，作業系統實驗室，副教授 羅習五，shiwulo@gmail.com

計算pi (上界線)



計算pi (下界線)



注意到了嗎，剛好把「上界線」往左平移一格，再補上最右邊即可

作業

- 使用上下逼近法計算pi
- 詳細作業規範請參考 :