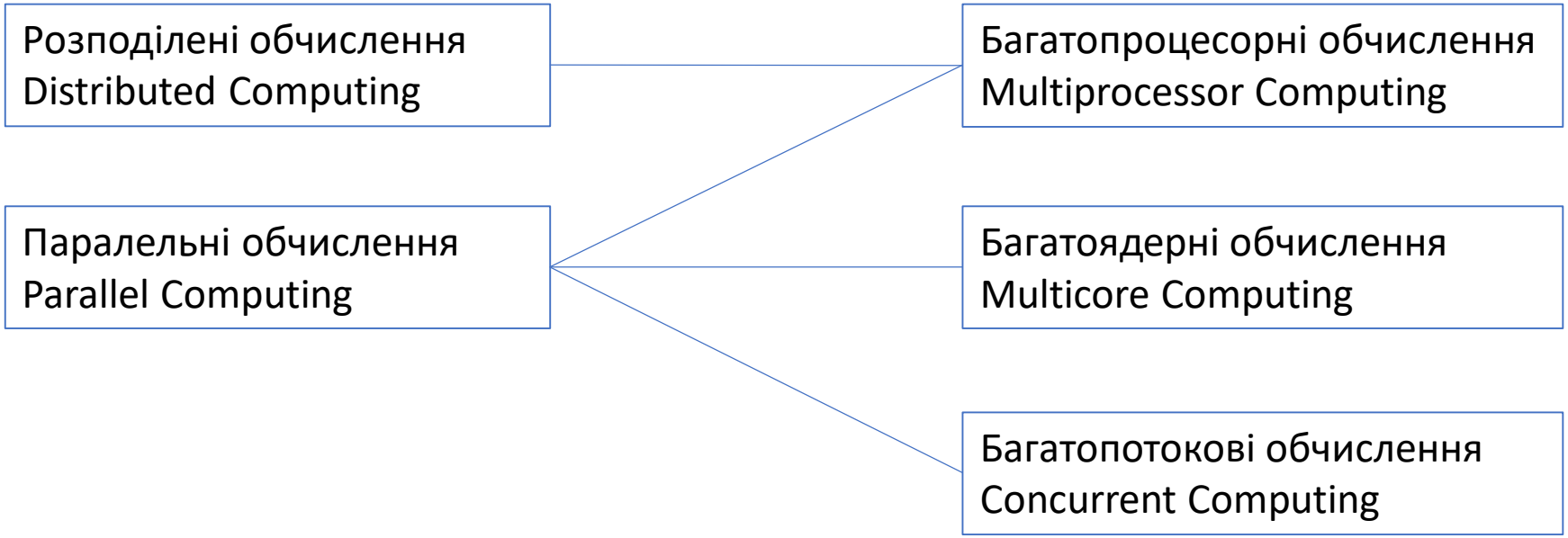
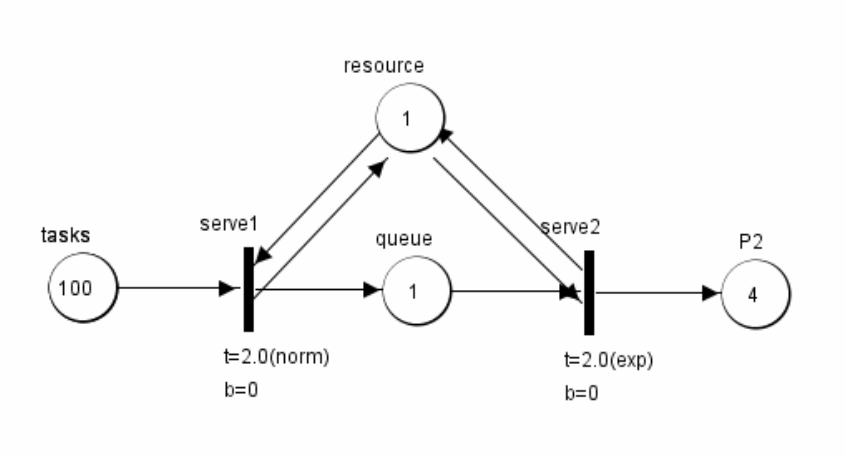
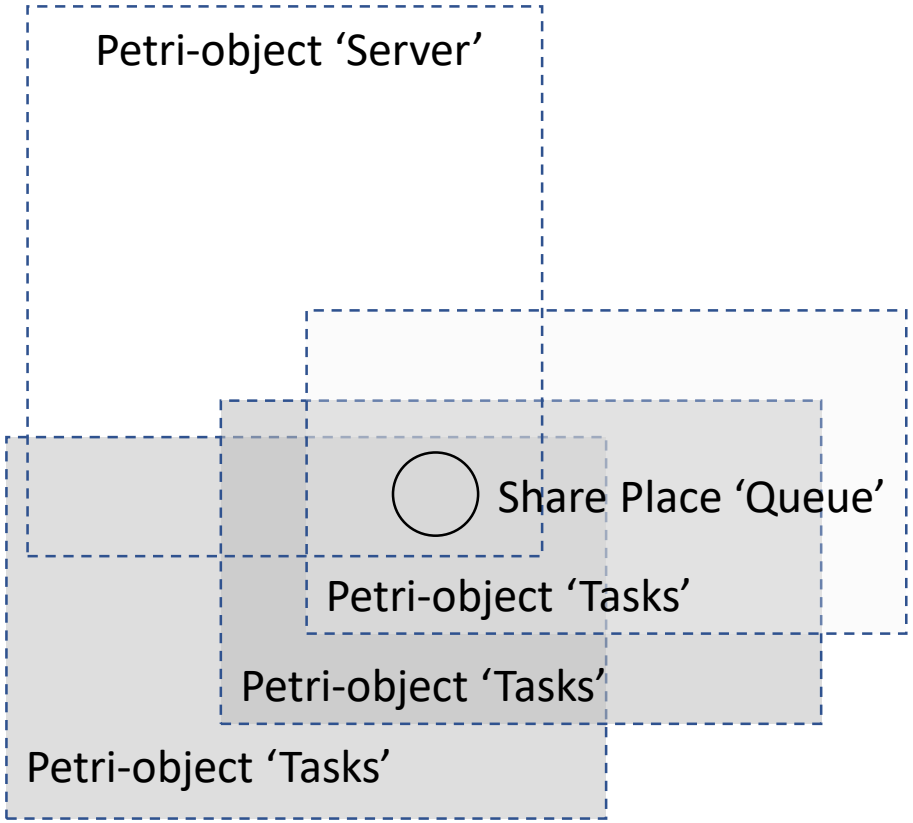
МОДЕЛЮВАННЯ РОЗПОДІЛЕНИХ ТА ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ СТОХАСТИЧНИМИ МЕРЕЖАМИ ПЕТРІ

Термінологія



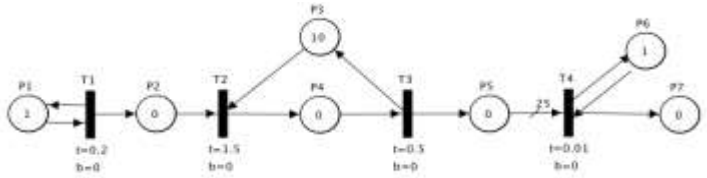
Стохастичні мережі Петрі та Петрі-об'єктний підхід



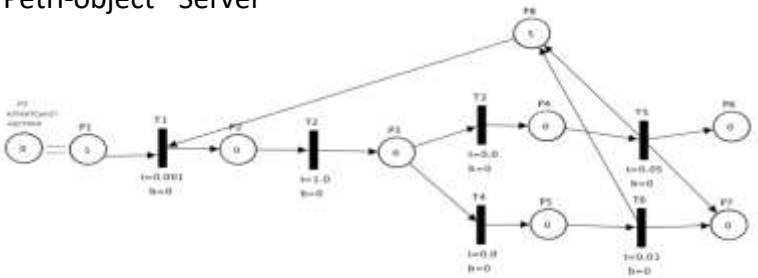


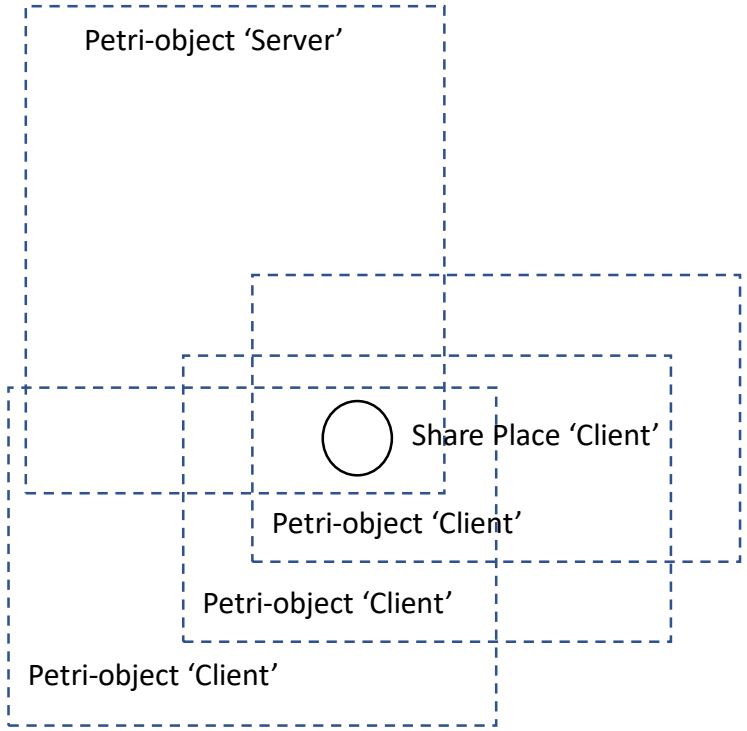
Моделювання клієнт-серверного додатку (розподілені обчислення)

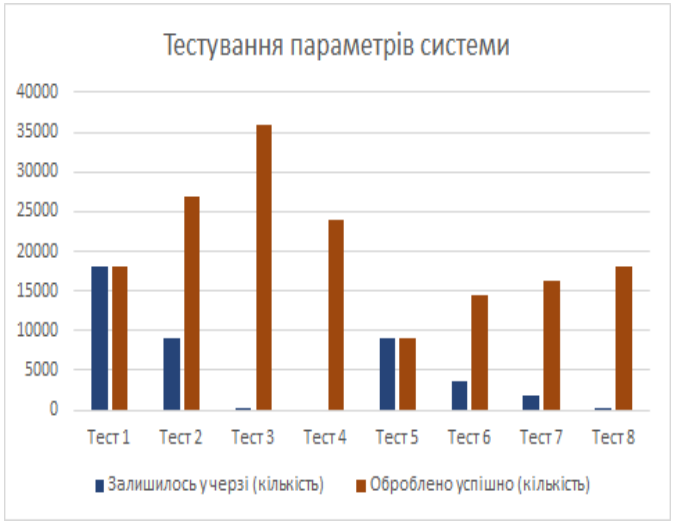
Petri-object ‘Client’



Petri-object ‘Server’







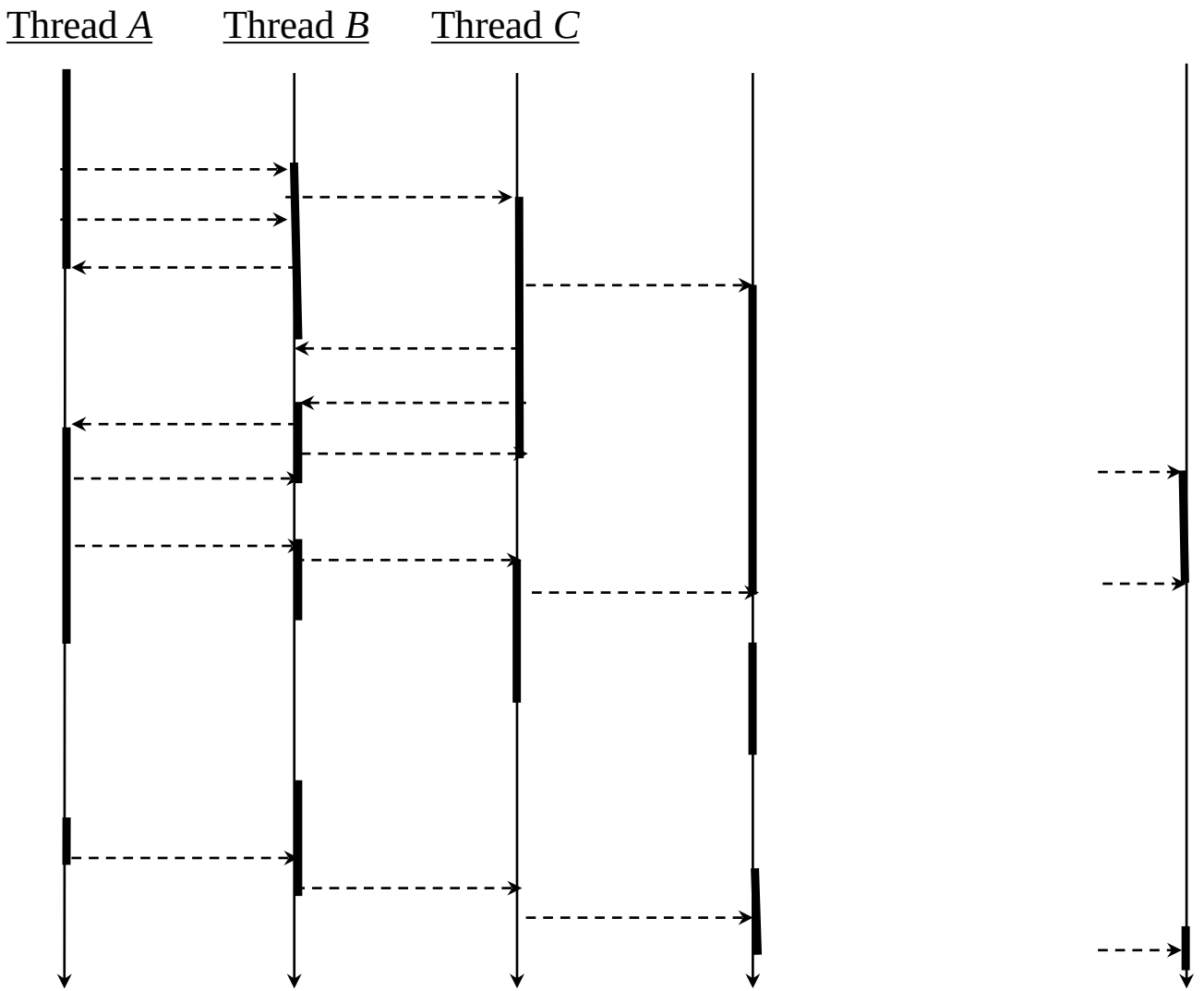
[Шишкін, В. І. Програмно-апаратний комплекс розумного відеореєстратора : магістерська дис. : 126 Інформаційні системи та технології / Шишкін Владислав Ігорович. – Київ, 2018. – 85 с.]

Багатопоточне програмування

Проблеми:

- Deadlock
- Starvation
- Livelock
- Memory consistency error

Існуючі дебагери орієнтовані на відлагодження послідовних, не паралельних програм



Проблеми розробки паралельних програм

Проблеми паралельних програм:

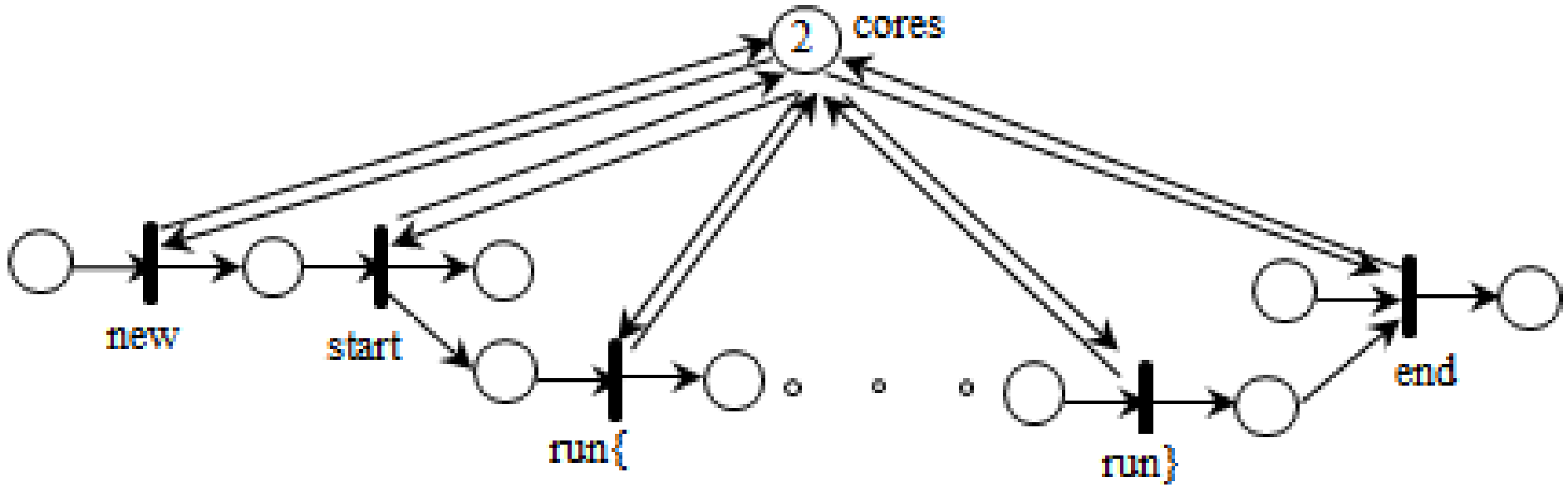
- можливість виникнення взаємного блокування роботи потоків (deadlock), неможливість завершення роботи потоків (livelock), неможливість захоплення ресурсу потоком (starvation),
- помилка при спільному використанні ресурсу (memory consistency error, «гонка» потоків),
- сповільнення роботи потоків через синхронізацію дій.

Проблеми, що ускладнюють процес розробки та тестування паралельних програм:

- недетермінований порядок інструкцій, виконуваних потоками,
- залежність результату запуску паралельної програми від обчислювальних ресурсів, на яких вона запускається.

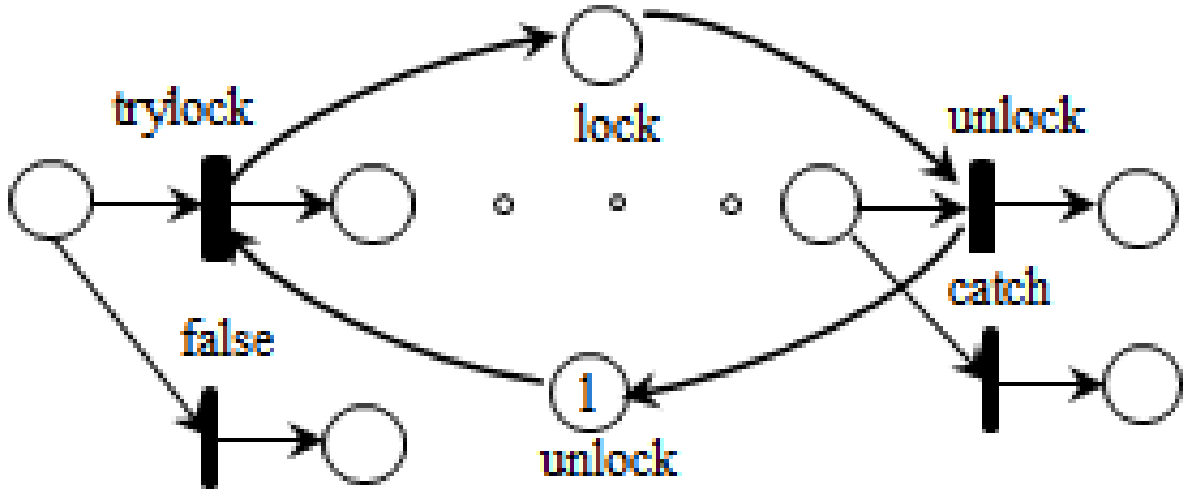
Реалізація створення потоку, початку та кінця його роботи

```
public static void main(String[] args) {
    Thread thread = new Thread(new Runnable());
    thread.start();
}
```



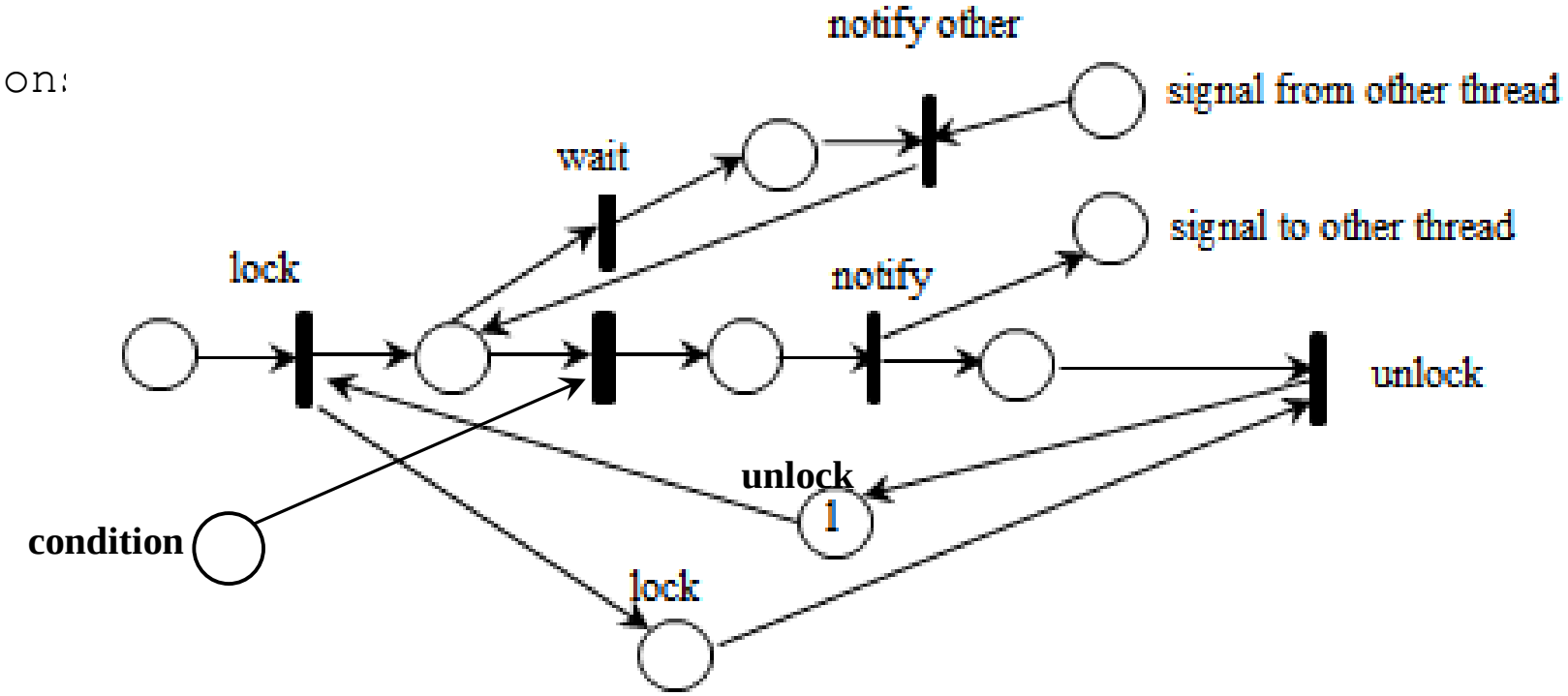
Блокування потоку

```
//https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/Lock.html#tryLock()
Lock lock = ...;
if (lock.tryLock()) {
     try {
         // manipulate protected state
     } finally {
         lock.unlock();
     }
} else {
// perform alternative
}
```



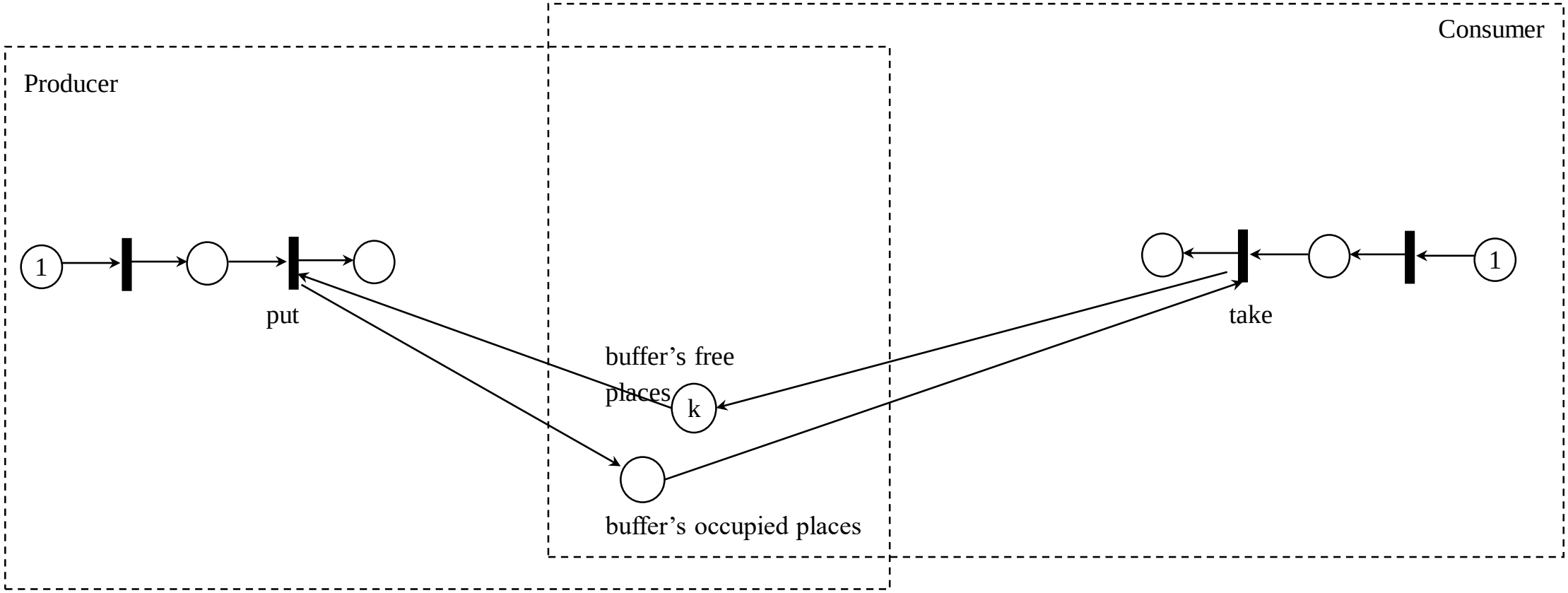
Синхронізація дій потоку

```
public synchronized void method() throws InterruptedException {
    while (!condition()) { // guarded block
      wait();
    }
      …// some action
    notifyAll();
}
```



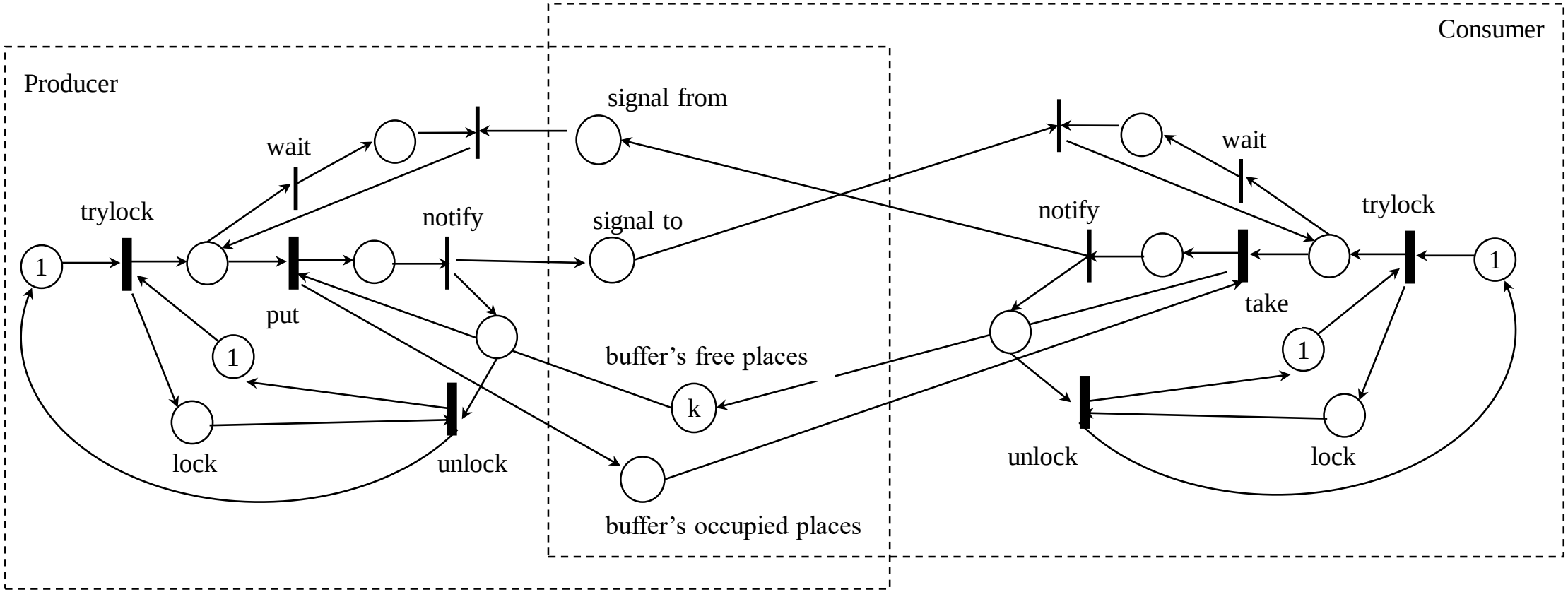
Модель Producer – Consumer.

Приклад **Guarded block** [Oracle: The Java Tutorials]



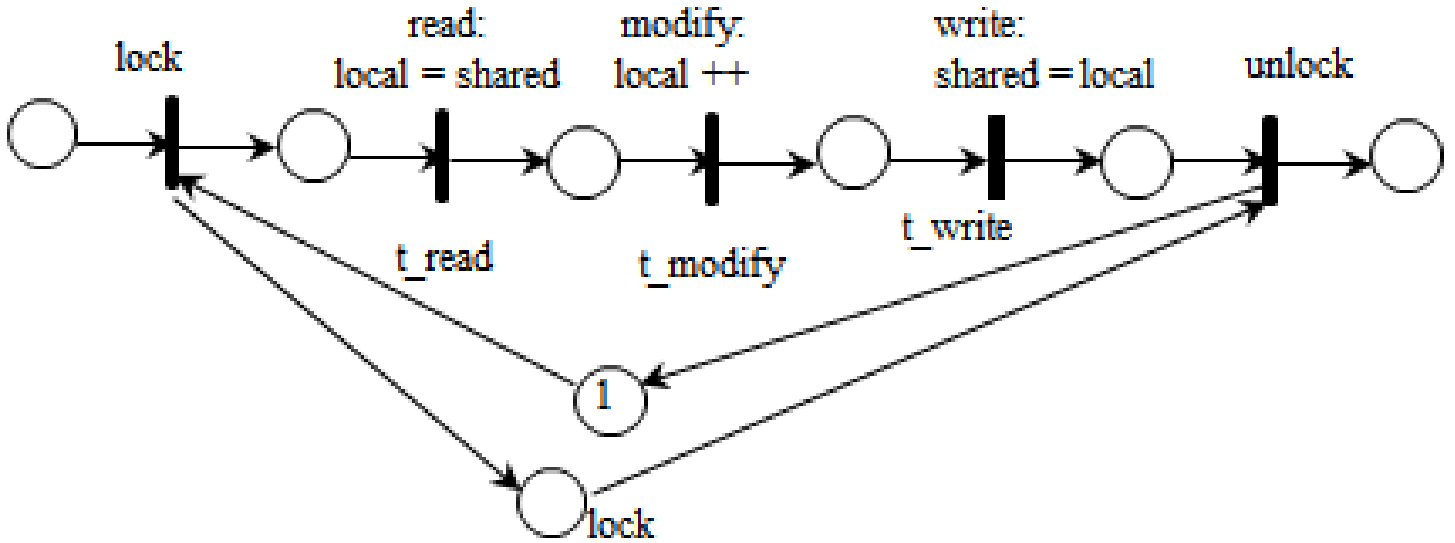
Модель Producer – Consumer.

Приклад Guarded block [Oracle: The Java Tutorials]



Доступ до спільних даних

```
public synchronized void  incMethod(){
      local++;
}
```



Deadlock: приклад об’єктів Friend [Oracle: The Java Tutorials]

```
public class Deadlock {
   static class Friend {
       private final String name;
       public Friend(String name) { this.name = name; }
       public String getName() { return this.name; }
       public synchronized void bow(Friend other) {
           System.out.format("%s: %s" + " has bowed to me!%n", this.name, bower.getName());

           other.bowBack(this);
       }
       public synchronized void bowBack(Friend bower) {
         System.out.format("%s: %s" + " has bowed back to me!%n", this.name, bower.getName());
       }
    }
    public static void main(String[] args) {
       final Friend a = new Friend(”A");
       final Friend b = new Friend("B");
       new Thread(new Runnable() {
                          public void run() { a.bow(b); } }
                 ).start();
       new Thread(new Runnable() {
                           public void run() { b.bow(a); } }


                 ).start();
     }
 }
```

Deadlock: приклад об’єктів Friend з The Java Tutorials

```
public class Deadlock {
   static class Friend {
      private final String name;
      public Friend(String name) { this.name = name; }
      public String getName() { return this.name; }
      public synchronized void bow(Friend other) {
          System.out.format("%s: %s" + " has bowed to me!%n", this.name, bower.getName());

          other.bowBack(this); // захоплення локера об’єкта other
      }
      public synchronized void bowBack(Friend bower) {
        System.out.format("%s: %s" + " has bowed back to me!%n", this.name, bower.getName());
      }
    }
    public static void main(String[] args) {
      final Friend a = new Friend(”A");
      final Friend b = new Friend("B");
      new Thread(new Runnable() {
                        public void run() { a.bow(b); } } // захоплення локера a, потім b
                ).start();
      new Thread(new Runnable() {
                         public void run() { b.bow(a); } } //  захоплення локера b, потім a


                ).start();
     }
 }
```

Safelock: приклад потоків Friend з The Java Tutorials

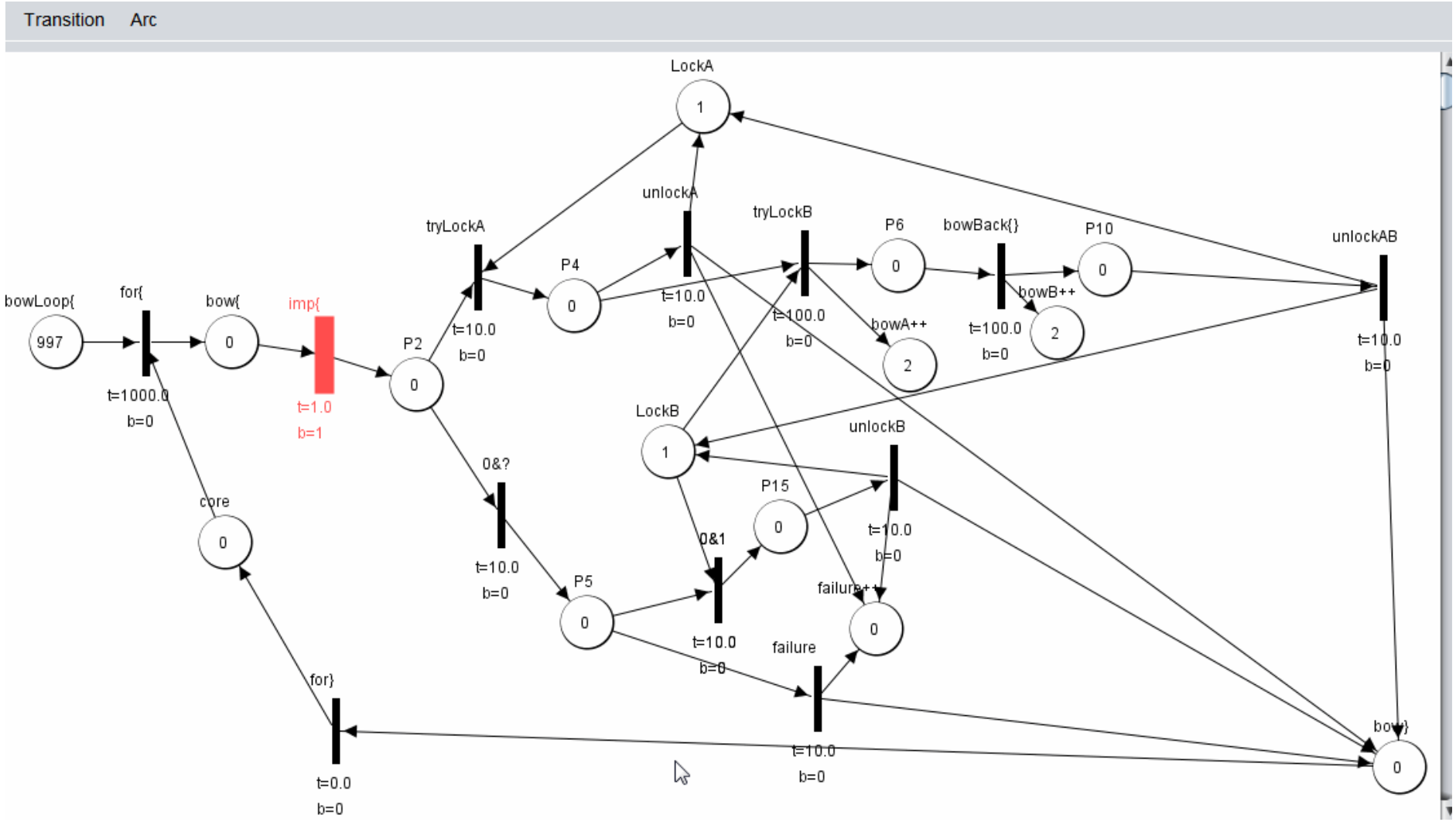
```
public class Safelock {
  static class Friend {
    private final String name;
    private final Lock lock = new ReentrantLock();
    public Friend(String name) { this.name = name; }
    public String getName() { return this.name; }

    public boolean impendingBow(Friend other) {
      Boolean myLock = false;
      Boolean yourLock = false;
      try {
        myLock = lock.tryLock();
        yourLock = other.lock.tryLock();
      } finally {
        if (! (myLock && yourLock)) {
             if (myLock) {
                 lock.unlock();
             }
             if (yourLock) {
                  other.lock.unlock();
             }
        }
      }
      return myLock && yourLock;
    }
```

```
  public void bow(Friend other) {
   if (impendingBow(other)) {
       try {
         System.out.format("%s: %s has"+
            " bowed to me!%n", this.name,
                                  bower.getName());
         bower.bowBack(this);
       } finally {
         lock.unlock();
         other.lock.unlock();
       }
   } else {
        System.out.format("%s: %s started" +
           " to bow to me, but saw that" +
           " I was already bowing to" +
           " him.%n", this.name, other.getName());

   }
  }
  public void bowBack(Friend other) {
    System.out.format("%s: %s has" +
             " bowed back to me!%n", this.name,
                                    other.getName());
  }
}
```

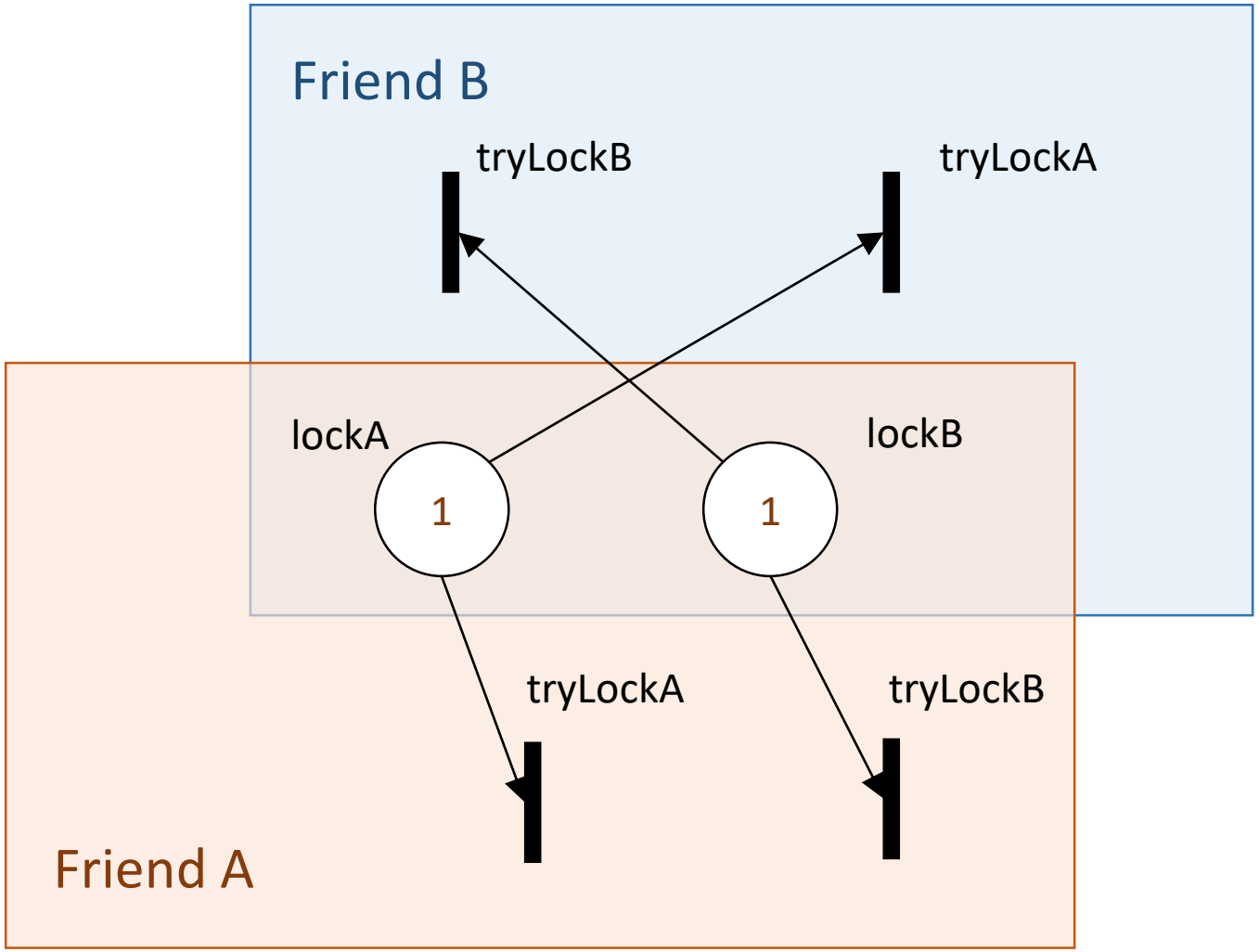
Safelock: приклад потоків Friend з The Java Tutorials

```
 static class BowLoop implements Runnable {
   private Friend bower; private Friend bowee;
   public BowLoop(Friend one, Friend other) {
       this.bower = one;
       this.bowee = other;
   }
   public void run() {
       Random random = new Random();
       for (;;) {
         try {
           Thread.sleep(random.nextInt(10));
       } catch (InterruptedException e) {}
       bowee.bow(bower);
    }
   }
 }

 public static void main(String[] args) {
   final Friend a = new Friend(”A");
   final Friend b = new Friend(”B");
   new Thread(new BowLoop(a, b)).start();
   new Thread(new BowLoop(b, a)).start();
 }
}
```

Моделювання конфлікту потоків: мережа Петрі об’єкту Friend



Зв’язки між Петрі-об’єктами



Програмний код створення Петрі-об’єктів

```
//Petri-objects creation
      class Friend extends PetriSim {
           public Friend(String name, int loop) throws ExceptionInvalidNetStructure {
                super(NetLibrary.CreateNetFriendUsingCores(name, loop, 2)); // 2 cores
            }
           public void addFriend(Friend other){
                 this.getNet().getListP()[7] = other.getNet().getListP()[2]; //lockOther = lock
                 this.getNet().getListP()[15] = other.getNet().getListP()[15]; // coresOther = cores
            }
       }
       Friend friendA = new Friend("A", 1000);
       Friend friendB = new Friend("B", 1000);
       Friend friendC = new Friend("C", 1000);
       Friend friendD = new Friend("D", 1000);

       friendA.addFriend(friendB);
       friendA.addFriend(friendC);
       friendA.addFriend(friendD);

       friendB.addFriend(friendA);
       friendB.addFriend(friendC);
       friendB.addFriend(friendD);

       friendC.addFriend(friendA);
       friendC.addFriend(friendB);
       friendC.addFriend(friendD);

       friendD.addFriend(friendA);
       friendD.addFriend(friendB);
       friendD.addFriend(friendC);
```

Програмний код створення Петрі-об’єктної моделі

```
public static void main(String[] args) throws ExceptionInvalidNetStructure {
        ArrayList<PetriSim> list = new ArrayList<>();

       //Petri-objects creation

       list.add(friendA);
       list.add(friendB);
       list.add(friendC);
       list.add(friendD);

       PetriObjModel model = new PetriObjModel(list);

       model.setIsProtokol(false);
       model.go(100000000);
}
```

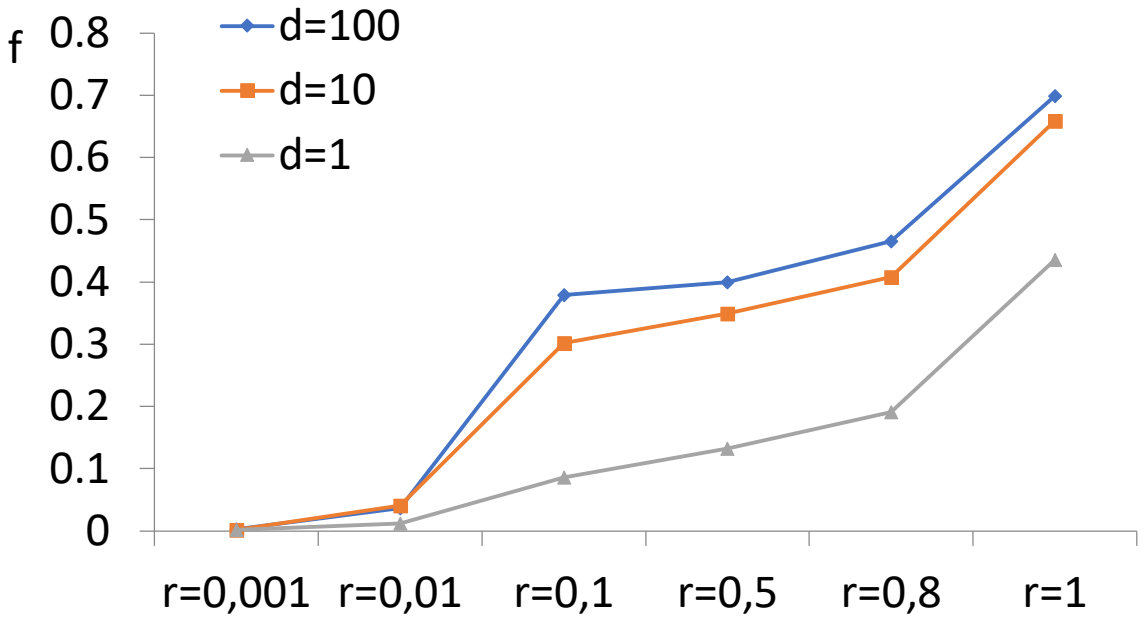
Експериментальне дослідження залежності появи конфлікту переходів від часових затримок

4 ▪ 3 = 12 потоків

d - часова затримка переходу “for{“ (відповідає команді sleep() у програмі)

d ▪ r – часова затримка інших переходів, де r – співвідношення часових затримок переходів (відповідає виконанню простих інструкцій)

f – відносна частота появи невдалої спроби (конфлікту потоків)



[Stetsenko I.V., Dyfuchyna O. Simulation of Multithreaded Algorithms Using Petri-Object Models. In: Hu Z., Petoukhov S., Dychka I., He M. (eds) Advances in Computer Science for Engineering and Education. ICCSEEA 2018. Advances in Intelligent Systems and Computing, vol 754, Springer, Cham, pp.391-401.]

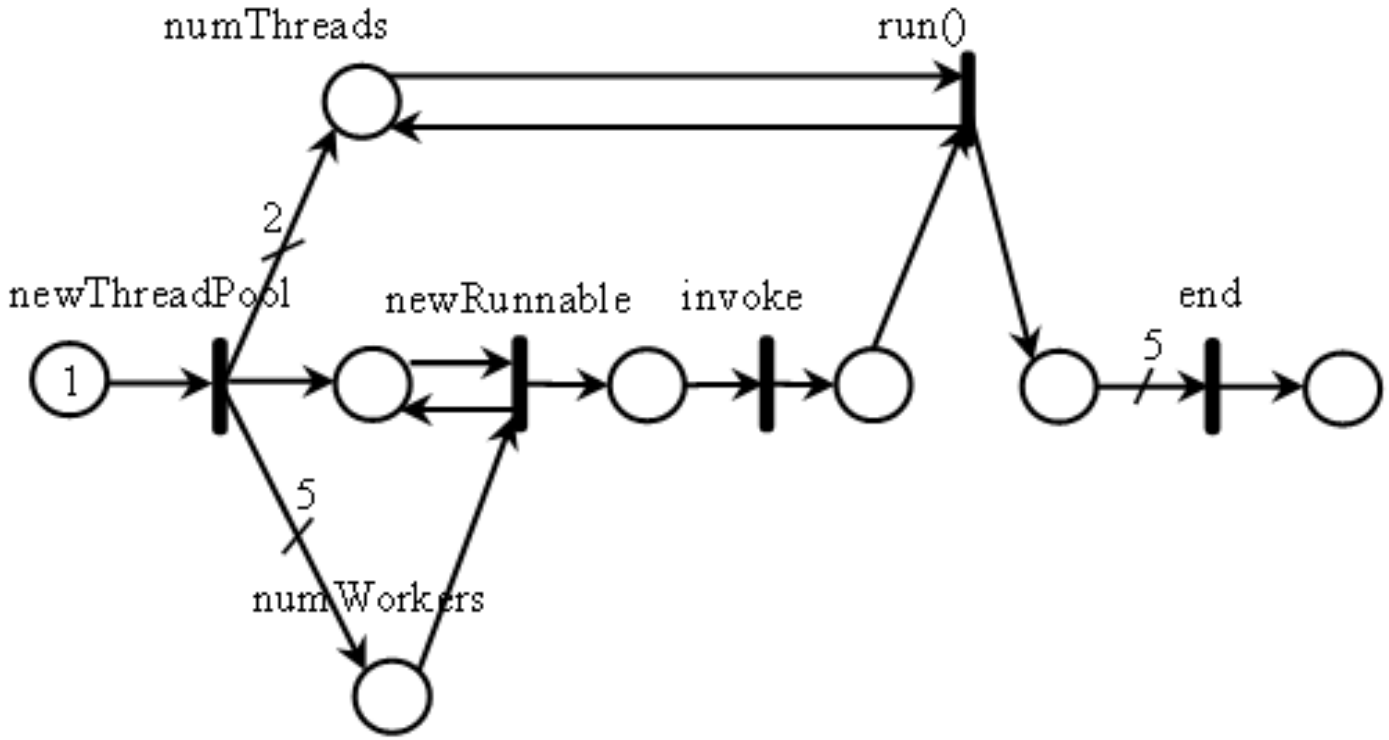
Оцінка точності результатів моделювання

Кількість потоків	Багатопоточна програма	Імітаційна модель (d=100, r=0.01)	Похибка
2	0.981450	0.978650	0.29%
12	0.959042	0.963417	0.46%
10 (90 workers)	0.987777	0.979080	0.88%
20 (380 workers)	0.988047	0.980910	0.72%
50 (2450 workers)	0.995212	0.981585	1.37%

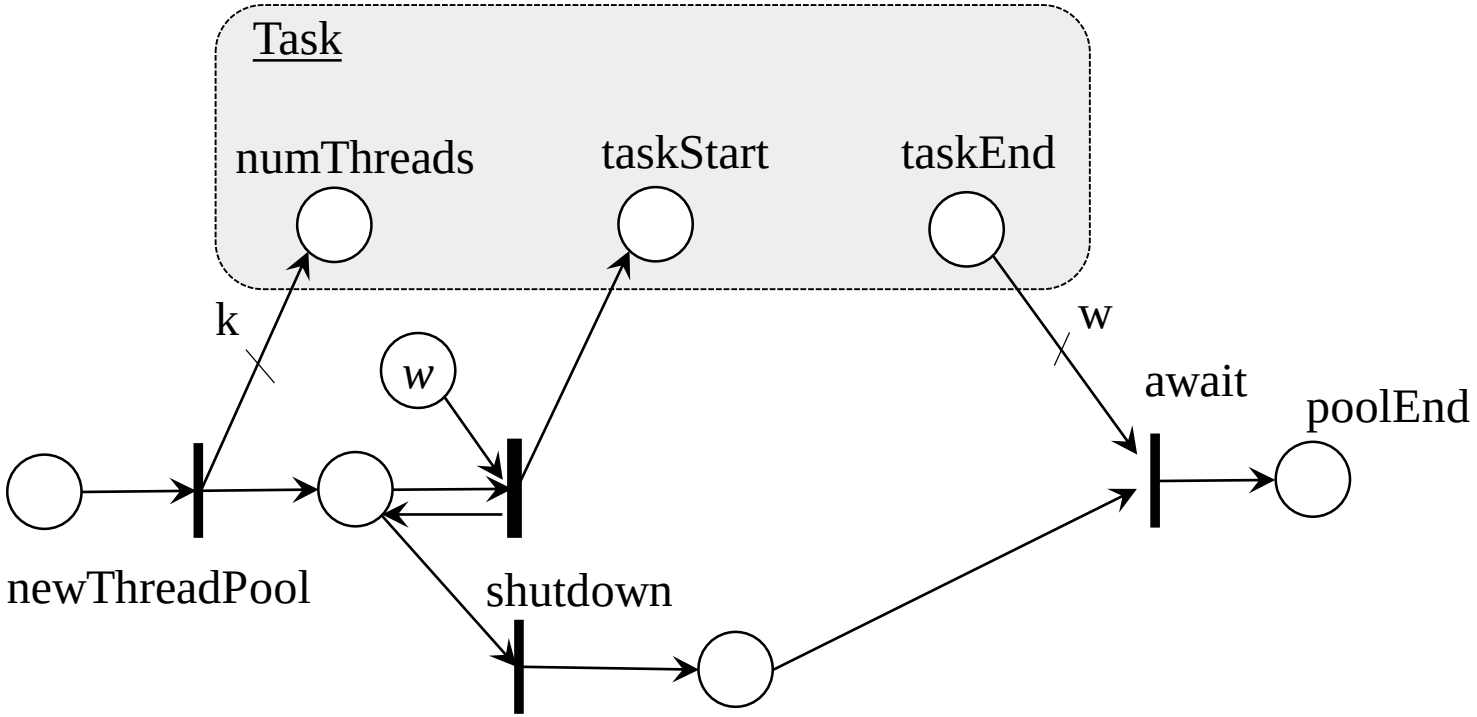
Пул потоків

```
ExecutorService executor =
     Executors.newFixedThreadPool(2);
for (int i = 0; i < 5; i++) {
  Runnable task =
       new TaskThread();
  executor.execute(task);
}
```

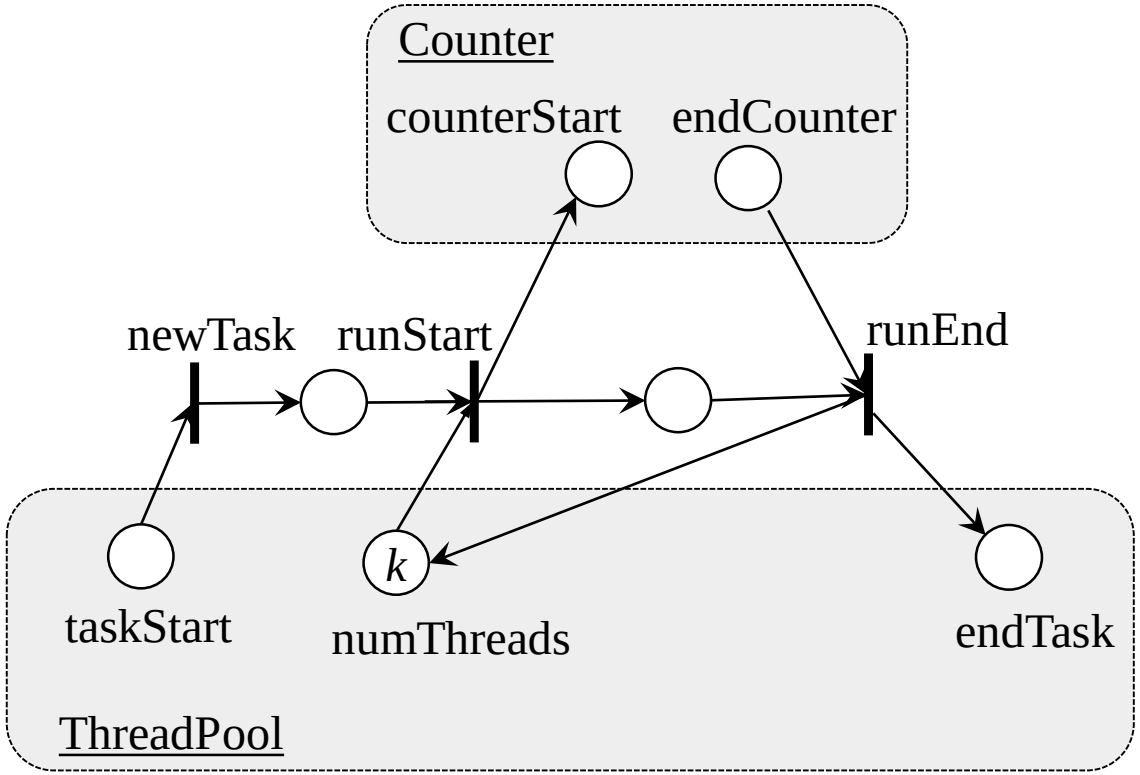
TaskThread – клас,
що імплементує інтерфейс Runnable.



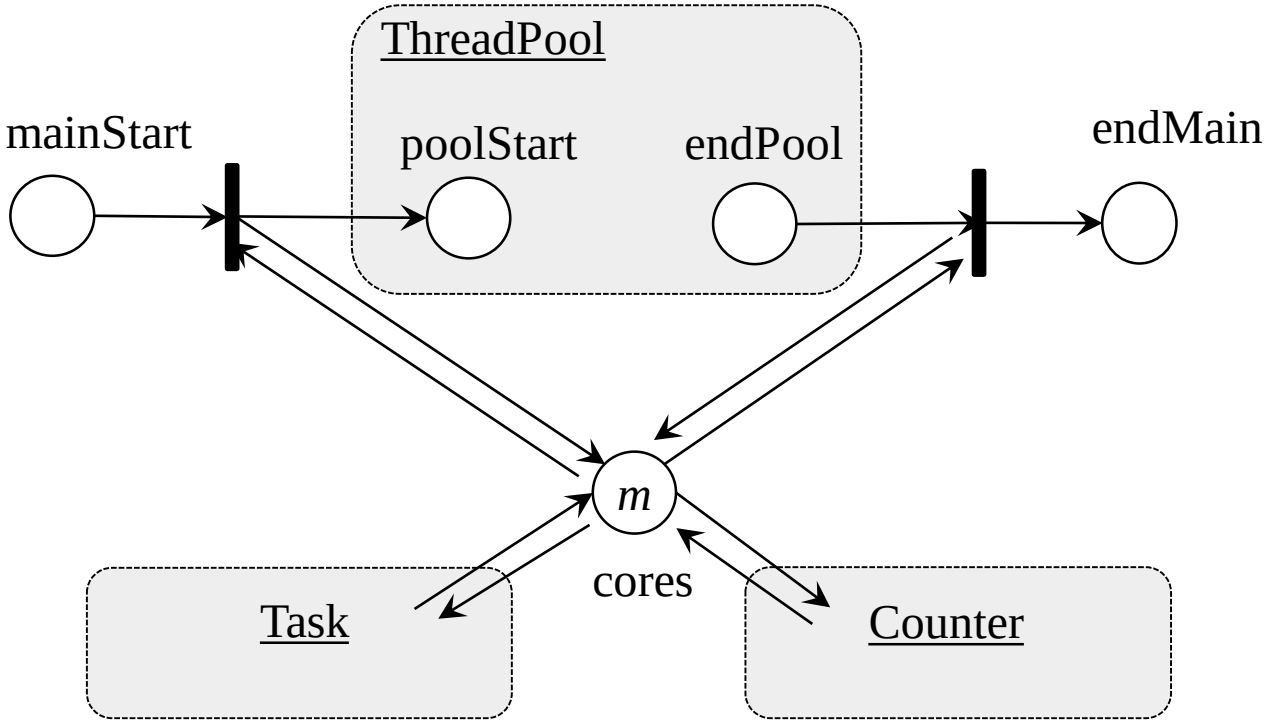
Петрі-об’єкт *ThreadPool*



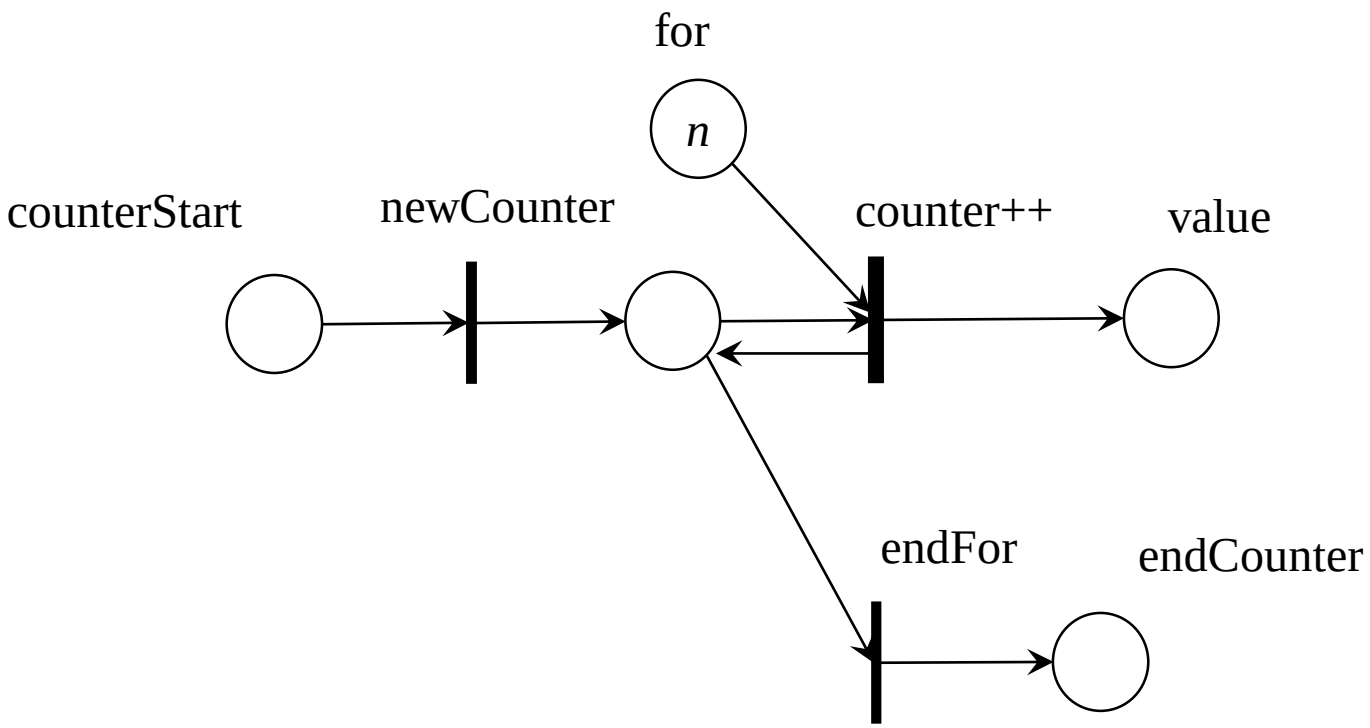
Петрі-об’єкт *Task*



Моделювання пулу потоків. Петрі-об’єкт *Main*



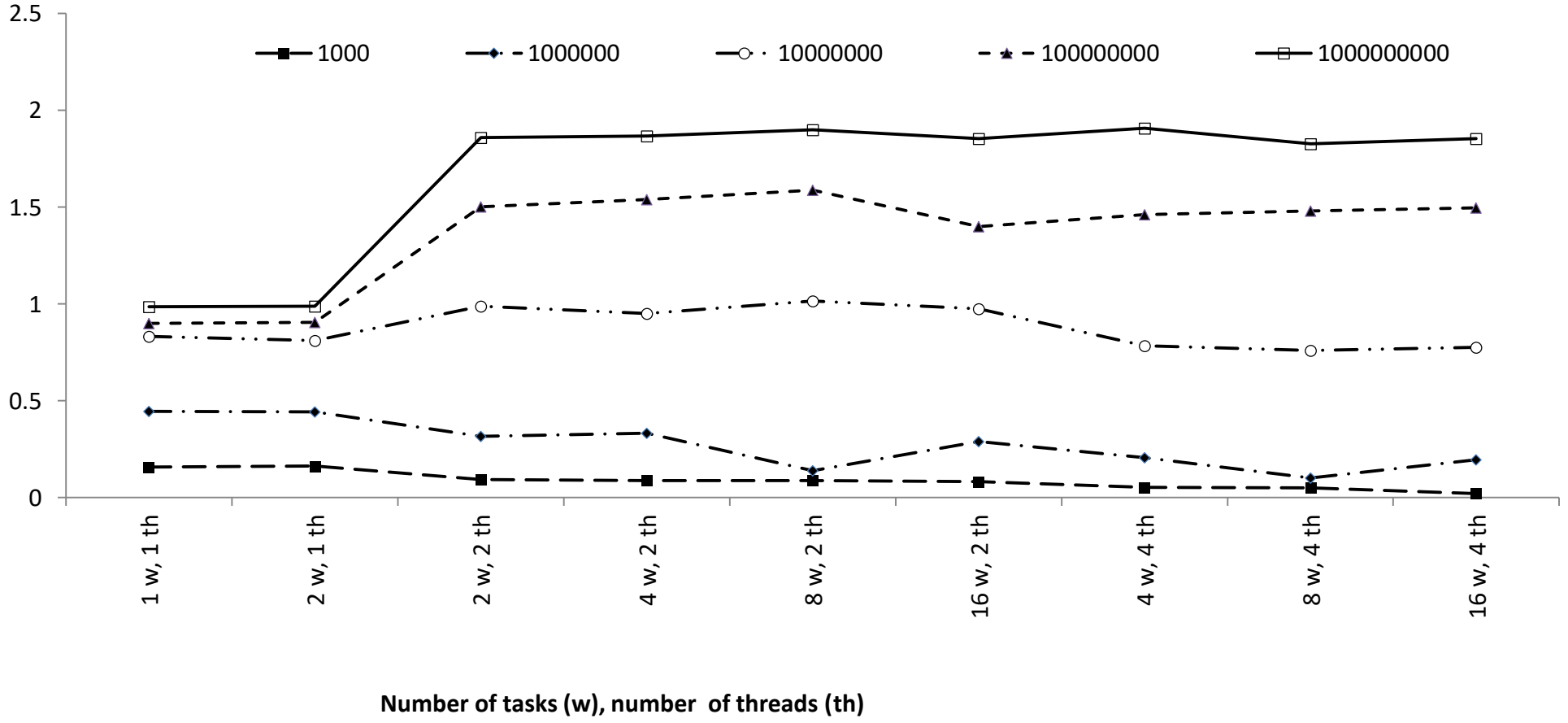
Петрі-об’єкт *Counter*



Експериментальне дослідження Java Thread Pool

Залежність прискорення від кількості потоків та кількості задач для різної складності обчислень, що запускаються на обчислення в пул потоків (1-ядерний комп’ютер)

$$S = \frac{T_{sequential}}{T_{parallel}}$$



[Stetsenko I.V., Dyfuchyna O. Thread Pool Parameters Tuning Using Simulation. In: Hu Z., Petoukhov S., Dychka I., He M. (eds) Advances in Computer Science for Engineering and Education II. ICCSEEA 2019. Advances in Intelligent Systems and Computing, vol 938, Springer, Cham., pp.78-89.]

Точність результатів моделювання

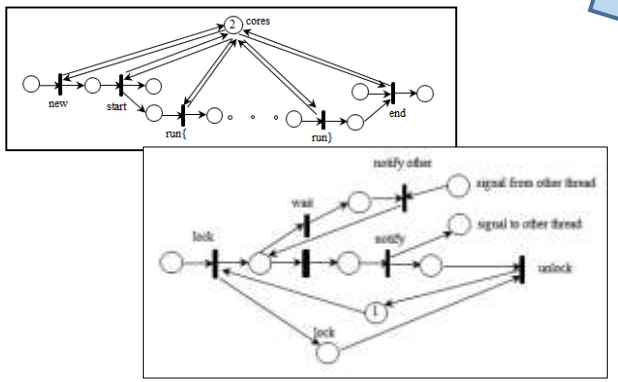
Number of tasks (w) and threads (th)	1 w, 1 th	2 w, 1 th	2 w, 2 th	4 w, 2 th	8 w, 2 th	16 w, 2 th	4 w, 4 th	8 w, 4 th	16 w, 4 th
Program	0.985	0.990	1.859	1.869	1.901	1.856	1.907	1.828	1.855
Model	0.998	0.998	1.989	1.984	1.976	1.977	1.985	1.977	1.960
Accuracy	1%	1%	7%	6%	4%	7%	4%	8%	6%

Проектування паралельних програм

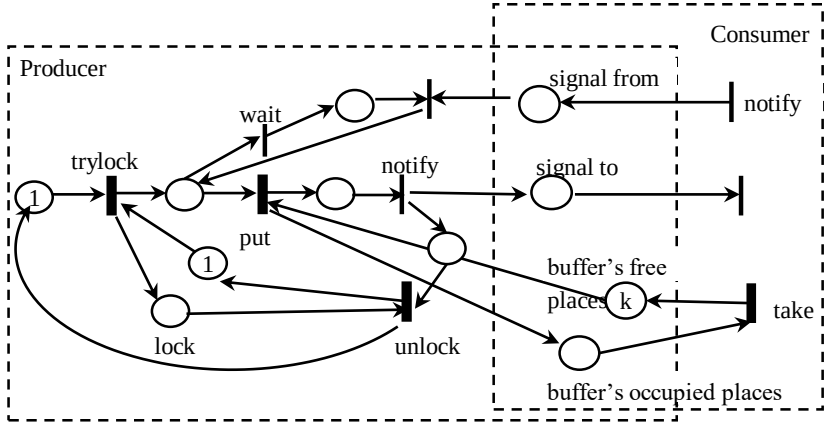
Програмний код

```
private final Lock lock = new ReentrantLock();
try{
  lock = lock.tryLock();
  …// synchronized actions
} finally {
   lock.unlock();
}
```

Мережі Петрі-об’єктів



Петрі-об'єктна модель



- Виявлення помилок в управлінні потоками
- Оцінювання продуктивності програми при різній кількості обчислювального ресурсу