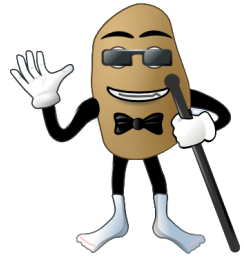
Część 2
Nie wszyscy naraz, czyli synchronizacja dla opornych

Maciej J. Mrowiński

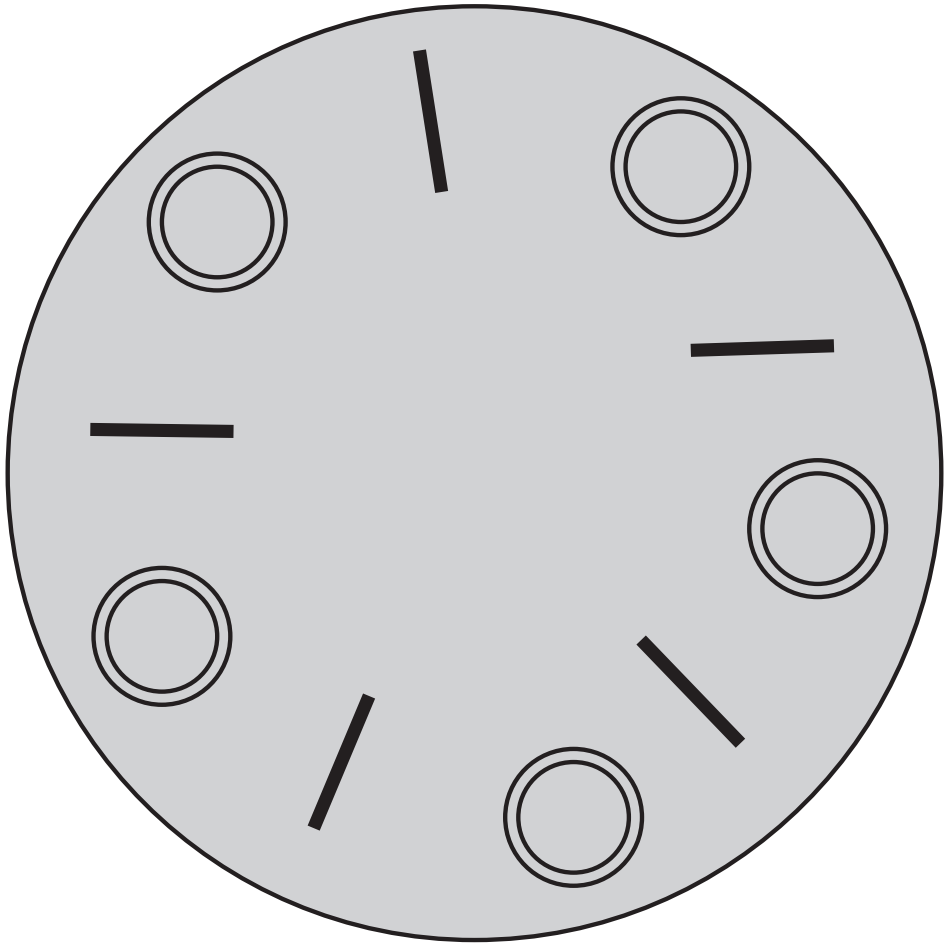
mrow@if.pw.edu.pl

Wydział Fizyki
Politechnika Warszawska

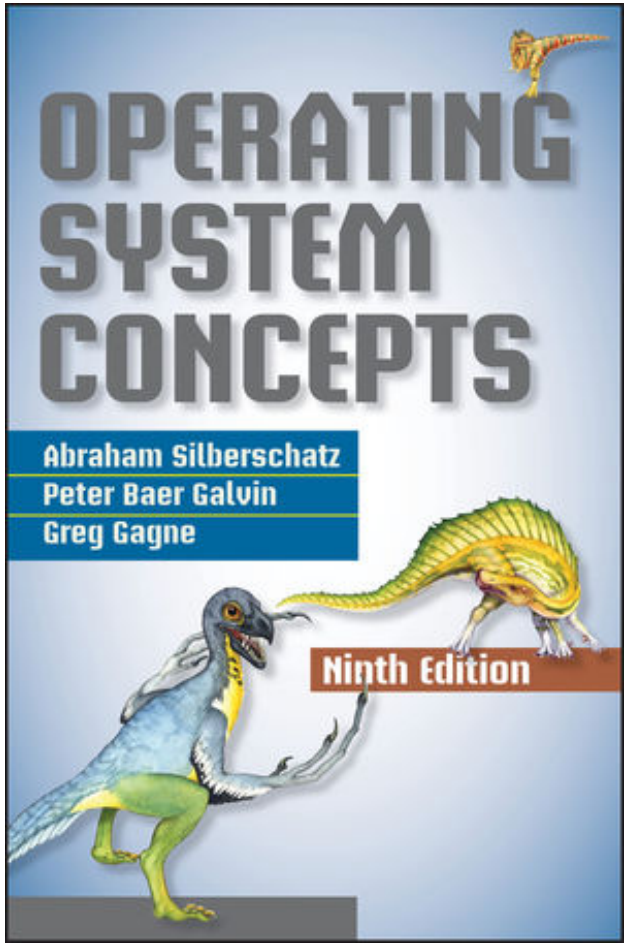
7 grudnia 2018

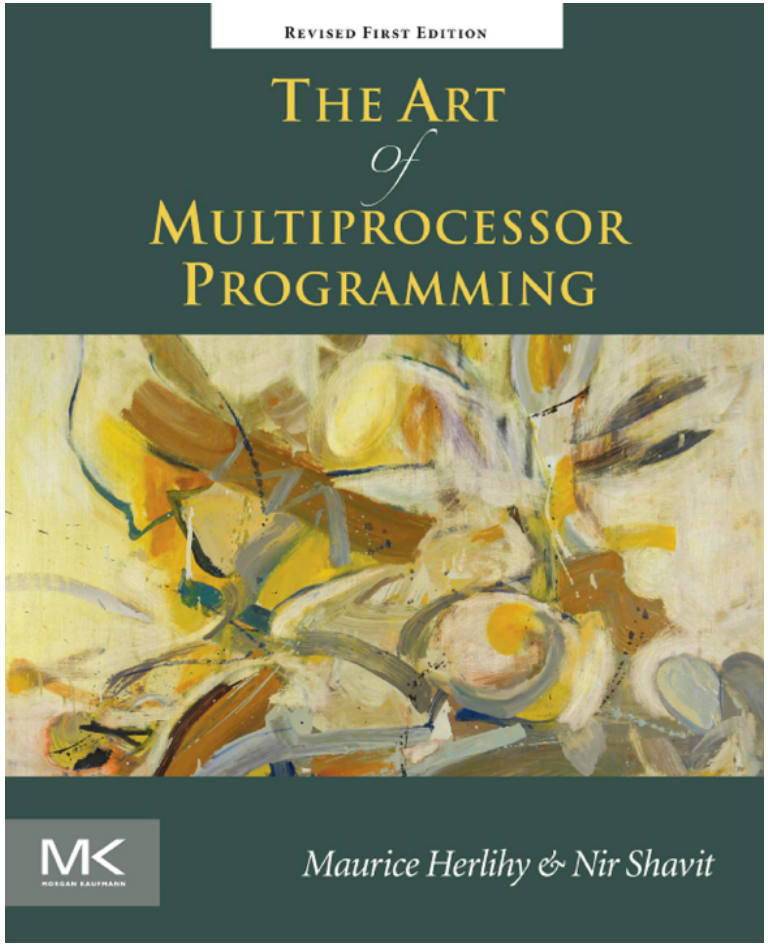


Pierwszy przykład poglądowy - ucztujący filozofowie



Dzisiejszy Odcinek Sponsoruje...





Szukamy liczb pierwszych

Szukamy liczb pierwszych na przedziale $[1, 10^{10}]$

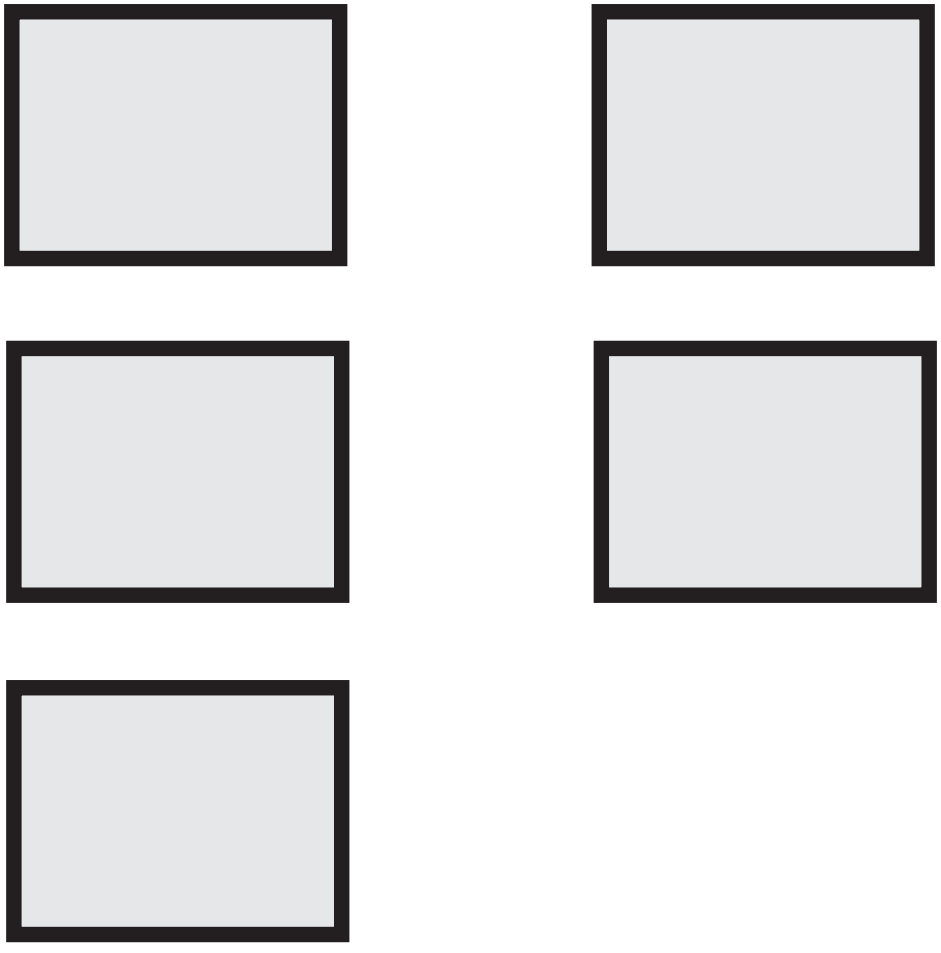
```
1      void primePrint() {
2         int id = ThreadID.get();
3         int block = power(10, 9);
4         for(int i = (id*block)+1; i <= (id+1)*block; ++i) {
5            if(isPrime(i))
6               print(i);
7         }
8      }
```

Ile możemy zyskać?

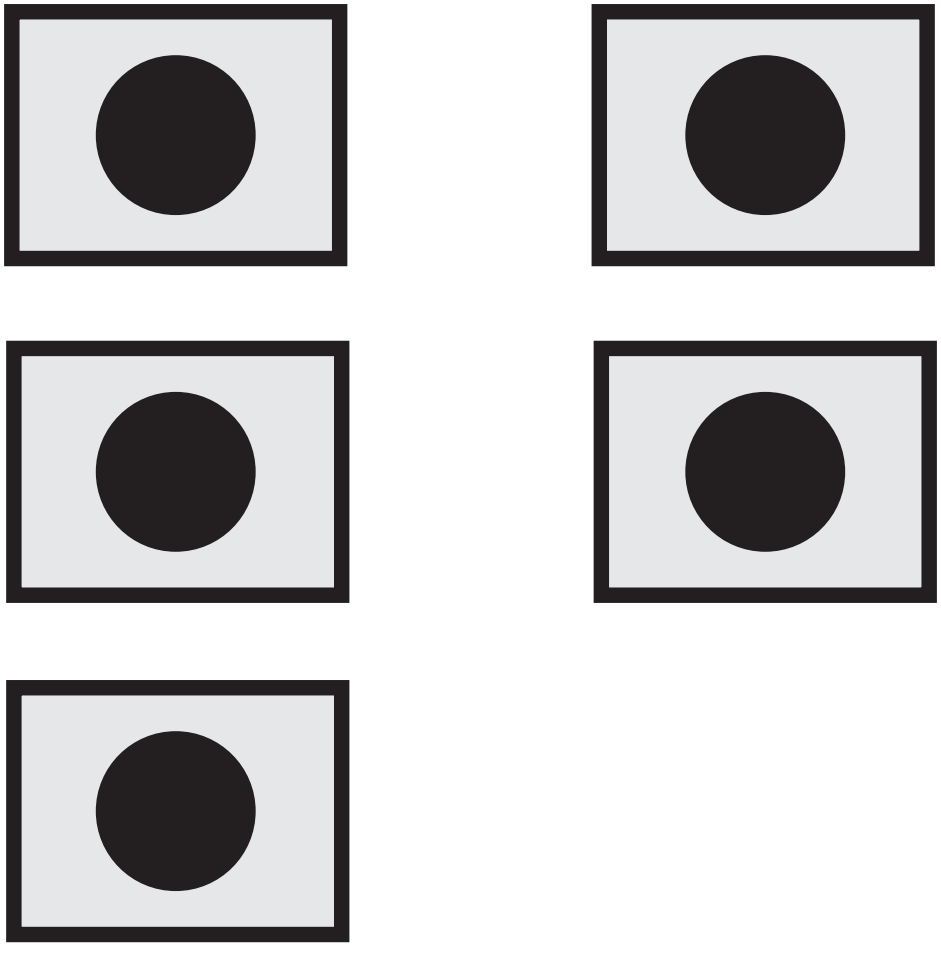
Prawo Amdahla

$$S = \frac{1}{1 - p + \frac{p}{n}}$$

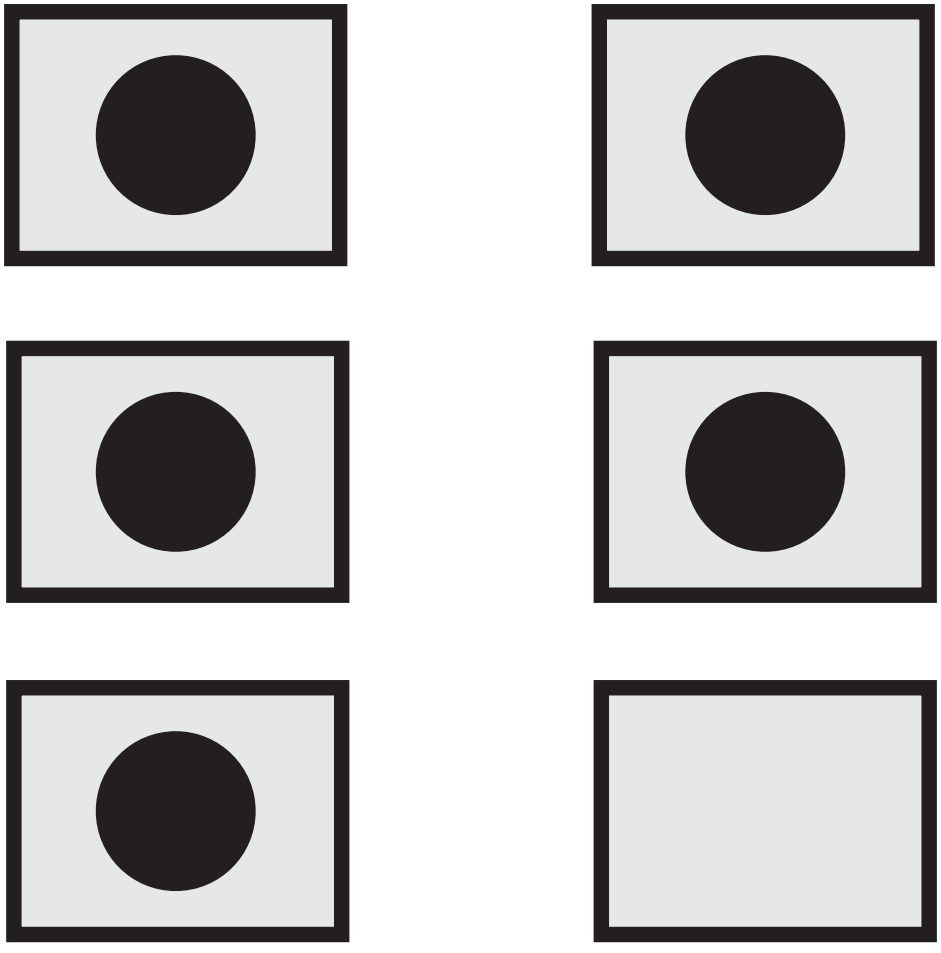
Prawo Amdahla



Prawo Amdahla



Prawo Amdahla



Prawo Amdahla

$$S = \frac{1}{1 - \frac{5}{6} + \frac{1}{6}} = 3$$

Prawo Amdahla

Dla $n = 10$ i $p = 0.9$ mamy

$$S \approx 5$$

Szukamy dalej liczb pierwszych

Wracamy do szukania liczb pierwszych

```
1  Counter counter = new Counter(1);
2
3  void primePrint() {
4     long number = 0;
5     long limit = power(10, 10);
6     while(number < limit) {
7        number = counter.getAndIncrement();
8        if(isPrime(i))
9           print(i);
10 }
```

Wracamy do szukania liczb pierwszych

```
1  public class Counter {
2      private long value;
3
4      public Counter(int i) {
5          value = i;
6      }
7
8      public long getAndIncrement() {
9          return value++;
10     }
11 }
```

Wracamy do szukania liczb pierwszych

```
1  public class Counter {
2     private long value;
3
4     public Counter(int i) {
5        value = i;
6     }
7
8     public long getAndIncrement() {
9        long temp = value;
10       value = temp + 1;
11       return temp;
12    }
13 }
```

Blokady (lock) i sekcje krytyczne (critical section)

Blokada

```
1  public interface Lock {
2      public void lock();
3      public void unlock();
4  }
```

Counter z blokadą

```
1  public class Counter {
2     private long value;
3     private Lock lock;
4
5     public long getAndIncrement() {
6        lock.lock();
7        try {
8           long temp = value;
9           value = temp + 1;
10          return temp;
11       } finally {
12          lock.unlock();
13       }
14    }
15 }
```

Co z tą blokadą?

Lock - podejście 001

```
1  class LockOne implements Lock {
2     private boolean[] flag = new boolean[2];
3
4     public void lock() {
5        int i = ThreadID.get();
6        int j = 1 - i;
7        flag[i] = true;
8        while(flag[j]);
9     }
10
11    public void unlock() {
12       int i = ThreadID.get();
13       flag[i] = false;
14    }
15 }
```

Pożądane cechy blokady

- wzajemne wykluczanie (mutual exclusion)
- brak zakleszczeń (deadlock)
- brak zagłodzenia (starvation)

Lock - podejście 010

```
1  class LockTwo implements Lock {
2     private int victim;
3
4     public void lock() {
5        int i = ThreadID.get();
6        victim = i;
7        while(victim == i);
8     }
9
10    public void unlock() {}
11 }
```

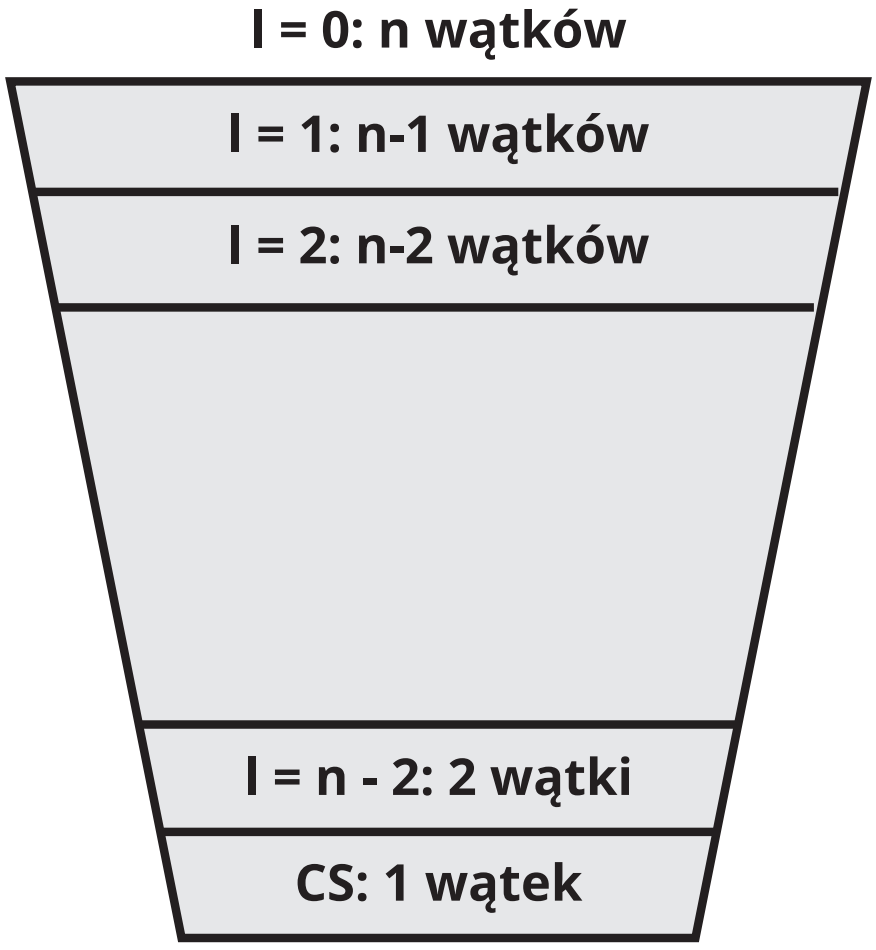
Lock - podejście 011 - algorytm Petersona

```
1  class Peterson implements Lock {
2     private boolean[] flag = new boolean[2];
3     private int victim;
4
5     public void lock() {
6        int i = ThreadID.get();
7        int j = 1 - i;
8        flag[i] = true;
9        victim = i;
10       while(flag[j] && victim == i);
11    }
12
13    public void unlock() {
14       int i = ThreadID.get();
15       flag[i] = false;
16    }
17 }
```

Lock - podejście 100 - Filter Lock

```
1  class Filter implements Lock {
2     int[] level;
3     int[] victim;
4
5     public Filter(int n) {
6        level = new int[n];
7        victim = new int[n];
8     }
9
10    public void lock() {
11       int me = ThreadID.get();
12       for(int i = 1; i < n; i++) {
13          level[me] = i;
14          victim[i] = me;
15
16          while((there exists k != me) (level[k] >= i &&
                victim[i] == me));
17       }
18    }
19    public void unlock() {
20       int me = ThreadID.get();
21       level[me] = 0;
22    }
23 }
```

Lock - podejście 100 - Filter Lock



Lock - podejście 101 - Bakery Lock

```
1  class Bakery implements Lock {
2     boolean[] flag;
3     Label[] label;
4
5     public Bakery (int n) {
6        flag = new boolean[n];
7        label = new Label[n];
8     }
9
10    public void lock() {
11       int i = ThreadID.get();
12       flag[i] = true;
13       label[i] = max(label[0], ...,label[n-1]) + 1;
14       while ((there exists k != i)(flag[k] &&
15             (label[k],k) << (label[i],i)));
16    }
17
18    public void unlock() {
19       flag[ThreadID.get()] = false;
20    }
21 }
```

Przykład praktyczny - Singleton i double-checked locking

Singleton

```
1  public class Singleton {
2    private static Singleton singleton = null;
3
4    public static synchronized Singleton getInstance() {
5      if(singleton == null) {
6        singleton = new Singleton();
7      }
8      return singleton;
9    }
10 }
```

Singleton

```
1  public class Singleton {
2    private static Singleton singleton = null;
3
4    public static Singleton getInstance() {
5      if(singleton == null) {
6        synchronized(Singleton.class) {
7          singleton = new Singleton();
8        }
9      }
10     return singleton;
11   }
12 }
```

Singleton

```
1  public class Singleton {
2    private static Singleton singleton = null;
3
4    public static Singleton getInstance() {
5      if(singleton == null) {
6        synchronized(Singleton.class) {
7          if(singleton == null) {
8            singleton = new Singleton();
9          }
10       }
11     }
12     return singleton;
13   }
14 }
```

Singleton

```
1  public class Singleton {
2    private static Singleton singleton = null;
3
4    public static Singleton getInstance() {
5      if(singleton == null) {
6        synchronized(Singleton.class) {
7          if(singleton == null) {
8            mem = allocate();
9            singleton = mem;
10           ConstructorSingleton(singleton);
11         }
12       }
13     }
14     return singleton;
15   }
16 }
```

Singleton

```
1  public class Singleton {
2     private static class SingletonWrapper {
3        static Singleton singleton = new Singleton();
4     }
5
6     public static Singleton getInstance() {
7         return SingletonWrapper.singleton;
8     }
9  }
```

Wracamy do blokad

Klasy Atomic*

Klasy Atomic*

java.util.concurrent.atomic

- AtomicBoolean
- AtomicInteger
- AtomicReference<T>
- ...

Klasy Atomic*

Przykład - AtomicInteger:

- int getAndSet(int newValue)
- int addAndGet(int delta)
- boolean compareAndSet(int expect, int update)
- int getAndIncrement()
- int incrementAndGet()

Klasa ThreadLocal<T>

Klasa ThreadLocal<T>

ThreadLocal<T>

- T get()
- protected T initialValue()
- void set(T value)
- static <S> ThreadLocal<S> withInitial(Supplier<? extends S> supplier)

Klasa ThreadLocal<T>

```
1  public class ThreadID {
2     private static final AtomicInteger nextID =
3        new AtomicInteger(0);
4
5     private static final ThreadLocal<Integer> threadID =
6        ThreadLocal.withInitial(nextID::getAndIncrement);
7
8     public static int get() {
9        return threadID.get();
10    }
11 }
```

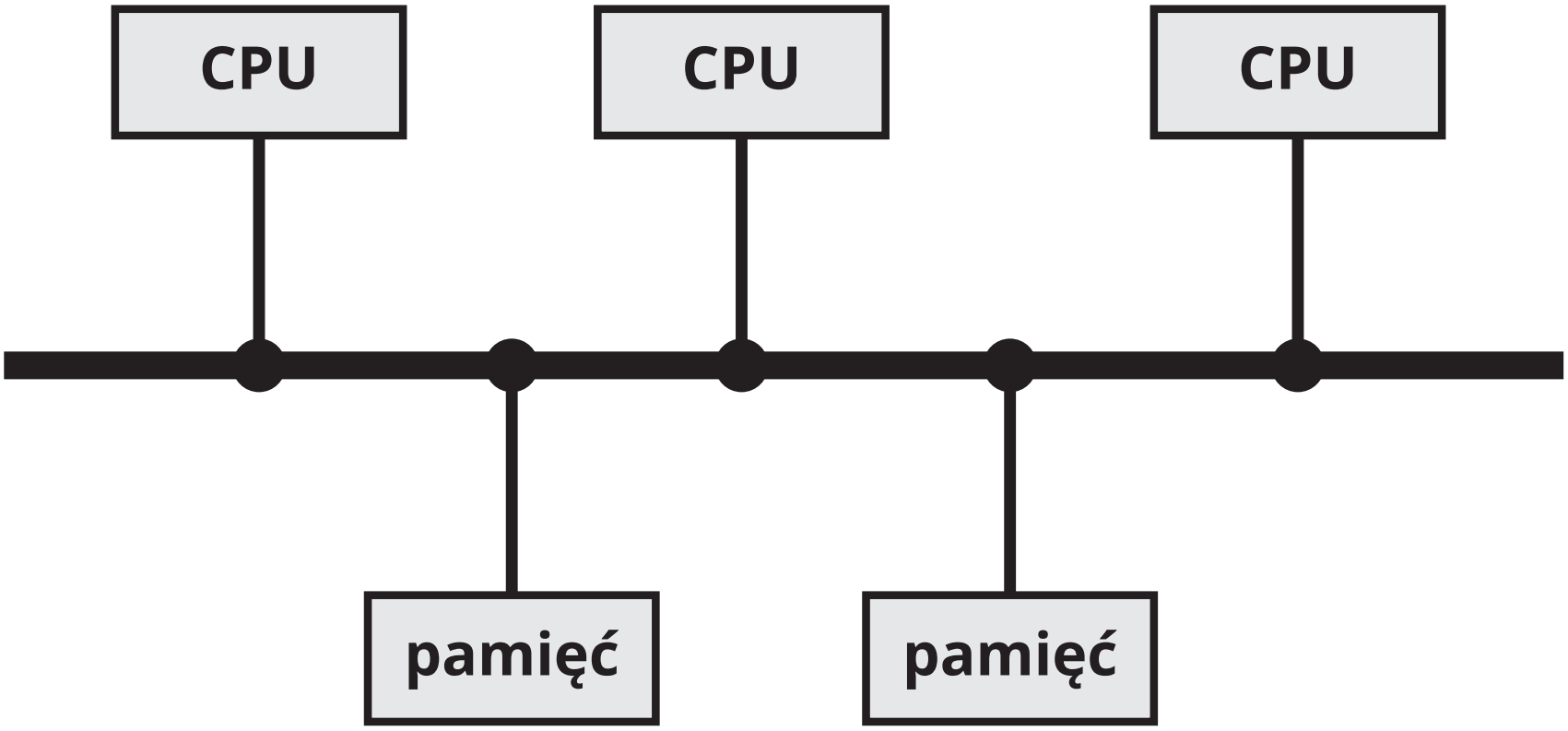
Test-And-Set Lock

```
1  public class TASLock implements Lock {
2     AtomicBoolean state = new AtomicBoolean(false);
3
4     public void lock() {
5        while(state.getAndSet(true));
6     }
7
8     public void unlock() {
9        state.set(false);
10    }
11 }
```

Test-Test-And-Set Lock

```
1  public class TTASLock implements Lock {
2     AtomicBoolean state = new AtomicBoolean(false);
3
4     public void lock() {
5        while(true) {
6           while(state.get());
7
8           if(!state.getAndSet(true))
9              return;
10       }
11    }
12
13    public void unlock() {
14       state.set(false);
15    }
16 }
```

Komunikacja



Backoff Lock

```
1  public class Backoff {
2     private final int minDelay, maxDelay;
3     private int limit;
4     private final Random random;
5
6     public Backoff(int min, int max) {
7        minDelay = min;
8        maxDelay = max;
9        limit = minDelay;
10       random = new Random();
11    }
12
13    public void backoff() throws InterruptedException {
14       int delay = random.nextInt(limit);
15       limit = Math.min(maxDelay, 2 * limit);
16       Thread.sleep(delay);
17    }
18 }
```

Backoff Lock

```
1  public class BackoffLock implements Lock {
2     private AtomicBoolean state = new AtomicBoolean(false);
3     private static final int MIN_DELAY = ...;
4     private static final int MAX_DELAY = ...;
5
6     public void lock() {
7        Backoff backoff = new Backoff(MIN_DELAY, MAX_DELAY);
8        while(true) {
9           while(state.get());
10          if(!state.getAndSet(true)) {
11             return;
12          } else {
13             backoff.backoff();
14          }
15       }
16    }
17
18    public void unlock() {
19       state.set(false);
20    }
21 }
```

Blokady z kolejkami

Array Lock

```
1  public class ALock implements Lock {
2     ThreadLocal<Integer> mySlotIndex = new ThreadLocal<>();
3
4     AtomicInteger tail;
5     volatile boolean[] flag;
6     int capacity;
7
8     public ALock(int capacity) {
9        this.capacity = capacity;
10       tail = new AtomicInteger(0);
11       flag = new boolean[capacity];
12       flag[0] = true;
13    }
14
15    public void lock() {
16       int slot = tail.getAndIncrement() % capacity;
17       mySlotIndex.set(slot);
18       while(!flag[slot]);
19    }
20
21    public void unlock() {
22       int slot = mySlotIndex.get();
23       flag[slot] = false;
24       flag[(slot + 1) % capacity] = true;
25    }
26 }
```

CLH Lock (prawie)

```
1  public class CLHLock implements Lock {
2     private static class QNode {
3        volatile boolean locked = false;
4     }
5
6     AtomicReference<QNode> tail = new AtomicReference<>(new QNode());
7     ThreadLocal<QNode> myNode = ThreadLocal.withInitial(QNode::new);
8
9     public void lock() {
10       QNode qnode = myNode.get();
11       qnode.locked = true;
12       QNode pred = tail.getAndSet(qnode);
13       while(pred.locked);
14    }
15
16    public void unlock() {
17       QNode qnode = myNode.get();
18       qnode.locked = false;
19    }
20 }
```

CLH Lock

```
1  public class CLHLock implements Lock {
2     private static class QNode {
3        volatile boolean locked = false;
4     }
5
6     AtomicReference<QNode> tail = new AtomicReference<>(new QNode());
7     ThreadLocal<QNode> myPred = new ThreadLocal<>();
8     ThreadLocal<QNode> myNode = ThreadLocal.withInitial(QNode::new);
9
10    public void lock() {
11       QNode qnode = myNode.get();
12       qnode.locked = true;
13       QNode pred = tail.getAndSet(qnode);
14       myPred.set(pred);
15       while(pred.locked);
16    }
17
18    public void unlock() {
19       QNode qnode = myNode.get();
20       qnode.locked = false;
21       myNode.set(myPred.get());
22    }
23 }
```

MCS Lock (prawie)

```
1   public class MCSLock implements Lock {
2      private static class QNode {
3         volatile boolean locked = false;
4         volatile QNode next = null;
5      }
6
7      AtomicReference<QNode> tail = new AtomicReference<>(null);
8      ThreadLocal<QNode> myNode = ThreadLocal.withInitial(QNode::new);
9
10     public void lock() {
11        QNode qnode = myNode.get();
12        QNode pred = tail.getAndSet(qnode);
13        if(pred != null) {
14           qnode.locked = true;
15           pred.next = qnode;
16           while(qnode.locked);
17        }
18     }
19
20     public void unlock() {
21        QNode qnode = myNode.get();
22        if(qnode.next == null)
23           return;
24
25        qnode.next.locked = false;
26        qnode.next = null;
27     }
28  }
```

MCS Lock

```
1   public class MCSLock implements Lock {
2      private static class QNode {
3         volatile boolean locked = false;
4         volatile QNode next = null;
5      }
6
7      AtomicReference<QNode> tail = new AtomicReference<>(null);
8      ThreadLocal<QNode> myNode = ThreadLocal.withInitial(QNode::new);
9
10     public void lock() {
11        QNode qnode = myNode.get();
12        QNode pred = tail.getAndSet(qnode);
13        if(pred != null) {
14           qnode.locked = true;
15           pred.next = qnode;
16           while(qnode.locked);
17        }
18     }
19
20     public void unlock() {
21        QNode qnode = myNode.get();
22        if(qnode.next == null) {
23           if(tail.compareAndSet(qnode, null))
24              return;
25           while(qnode.next == null);
26        }
27        qnode.next.locked = false;
28        qnode.next = null;
29     }
30  }
```

Monitor

Problem

```
1  mutex.lock();
2  try {
3      queue.enq(x);
4  } finally {
5      mutex.unlock();
6  }
```

Monitor

java.util.concurrent.locks.Lock

- void lock()
- void lockInterruptibly()
- boolean tryLock()
- boolean tryLock(long time, TimeUnit unit)
- void unlock()
- Condition newCondition()

Condition

java.util.concurrent.locks.Condition

- void await()
- boolean await(long time, TimeUnit unit)
- long awaitNanos(long nanosTimeout)
- void awaitUninterruptibly()
- boolean awaitUntil(Date deadline)
- void signal()
- void signalAll()

Przykład - BoundedBuffer

```
1   class BoundedBuffer {
2      final Lock lock = new ReentrantLock();
3      final Condition notFull  = lock.newCondition();
4      final Condition notEmpty = lock.newCondition();
5      final Object[] items = new Object[100];
6      int putptr, takeptr, count;
7
8      public void put(Object x) throws InterruptedException {
9         lock.lock();
10        try {
11           while(count == items.length) notFull.await();
12           items[putptr] = x;
13           if(++putptr == items.length) putptr = 0;
14           ++count;
15           notEmpty.signal();
16        } finally {
17           lock.unlock();
18        }
19     }
20     public Object take() throws InterruptedException {
21        lock.lock();
22        try {
23           while(count == 0) notEmpty.await();
24           Object x = items[takeptr];
25           if(++takeptr == items.length) takeptr = 0;
26              --count;
27           notFull.signal();
28           return x;
29        } finally {
30           lock.unlock();
31        }
32     }
33  }
```

Reentrant Lock

Przykład - Reentrant Lock

```
1  public class SimpleReentrantLock implements Lock {
2     private final Lock lock = new SimpleLock();
3     private final Condition condition = lock.newCondition();
4     private int owner = 0;
5     private int holdCount = 0;
```

Przykład - Reentrant Lock

```
6       public void lock() {
7           int me = ThreadID.get();
8           lock.lock();
9           try {
10              if(owner == me) {
11                  holdCount++;
12                  return;
13              }
14              while(holdCount != 0) condition.await();
15
16              owner = me;
17              holdCount = 1;
18          } finally {
19              lock.unlock();
20          }
21      }
22
23      public void unlock() {
24          lock.lock();
25          try {
26              if(holdCount == 0 || owner != ThreadID.get())
27                  throw new IllegalMonitorStateException();
28              holdCount--;
29              if(holdCount == 0)
30                  condition.signal();
31          } finally {
32              lock.unlock();
33          }
34      }
35  }
```

Readers–Writers Problem

ReadWriteLock

ReadWriteLock

- Lock readLock()
- Lock writeLock()

ReadWriteLock

```
1   public class SimpleReadWriteLock implements ReadWriteLock {
2      private int readers = 0;
3      private boolean writer = false;
4      private final Lock lock = new ReentrantLock();
5      private final Condition condition = lock.newCondition();
6      private final Lock readLock = new ReadLock();
7      private final Lock writeLock = new WriteLock();
8
9      public Lock readLock() {
10        return readLock;
11     }
12
13     public Lock writeLock() {
14        return writeLock;
15     }
```

ReadWriteLock

```
16      private class ReadLock implements Lock {
17         public void lock() {
18            lock.lock();
19            try {
20               while(writer) condition.await();
21               readers++;
22            } finally {
23               lock.unlock();
24            }
25         }
26
27         public void unlock() {
28            lock.lock();
29            try {
30               readers--;
31               if(readers == 0)
32                  condition.signalAll();
33            } finally {
34               lock.unlock();
35            }
36         }
37      }
```

ReadWriteLock

```
38      private class WriteLock implements Lock {
39          public void lock() {
40              lock.lock();
41              try {
42                  while(readers > 0 || writer) condition.await();
43                  writer = true;
44              } finally {
45                  lock.unlock();
46              }
47          }
48
49          public void unlock() {
50              lock.lock();
51              try {
52                  writer = false;
53                  condition.signalAll();
54              } finally {
55                  lock.unlock();
56              }
57          }
58      }
59  }
```

ReadWriteLock

```
38      private class WriteLock implements Lock {
39          public void lock() {
40              lock.lock();
41              try {
42                  while(writer) condition.await();
43
44                  writer = true;
45                  while(readers > 0) condition.await();
46              } finally {
47                  lock.unlock();
48              }
49          }
50
51          public void unlock() {
52              writer = false;
53              condition.signalAll();
54          }
55      }
56  }
```

Przykład praktyczny - listy i zbiory

Set i Node

```
1   public interface Set<T> {
2      boolean add(T x);
3      boolean remove(T x);
4      boolean contains(T x);
5   }
6
7   private class Node<T> {
8      volatile T item;
9      volatile int key;
10     volatile Node next;
11  }
```

Lista - coarse-grained

CoarseList

```
1   public class CoarseList<T> {
2      private Node head;
3      private Lock lock = new ReentrantLock();
4
5      public CoarseList() {
6         head = new Node(Integer.MIN_VALUE);
7         head.next = new Node(Integer.MAX_VALUE);
8      }
9
10     public boolean add(T item) {
11        Node pred, curr;
12        int key = item.hashCode();
13        lock.lock();
14        try {
15           pred = head;
16           curr = pred.next;
17           while(curr.key < key) {
18              pred = curr;
19              curr = curr.next;
20           }
21           if(key == curr.key) {
22              return false;
23           } else {
24              Node node = new Node(item);
25              node.next = curr;
26              pred.next = node;
27              return true;
28           }
29        } finally {
30           lock.unlock();
31        }
32     }
```

CoarseList

```
33      public boolean remove(T item) {
34         Node pred, curr;
35         int key = item.hashCode();
36         lock.lock();
37         try {
38            pred = head;
39            curr = pred.next;
40            while(curr.key < key) {
41               pred = curr;
42               curr = curr.next;
43            }
44            if(key == curr.key) {
45               pred.next = curr.next;
46               return true;
47            } else {
48               return false;
49            }
50         } finally {
51            lock.unlock();
52         }
53      }
54
55      ...
56   }
```

Lista - fine-grained

FineList

```
1     public boolean add(T item) {
2        int key = item.hashCode();
3        head.lock();
4        Node pred = head;
5        try {
6           Node curr = pred.next;
7           curr.lock();
8           try {
9              while(curr.key < key) {
10                pred.unlock();
11                pred = curr;
12                curr = curr.next;
13                curr.lock();
14             }
15             if(curr.key == key) {
16                return false;
17             }
18             Node newNode = new Node(item);
19             newNode.next = curr;
20             pred.next = newNode;
21             return true;
22          } finally {
23             curr.unlock();
24          }
25       } finally {
26          pred.unlock();
27       }
28    }
```

FineList

```
29      public boolean remove(T item) {
30         Node pred = null, curr = null;
31         int key = item.hashCode();
32         head.lock();
33         try {
34            pred = head;
35            curr = pred.next;
36            curr.lock();
37            try {
38               while(curr.key < key) {
39                  pred.unlock();
40                  pred = curr;
41                  curr = curr.next;
42                  curr.lock();
43               }
44               if(curr.key == key) {
45                  pred.next = curr.next;
46                  return true;
47               }
48               return false;
49            } finally {
50               curr.unlock();
51            }
52         } finally {
53            pred.unlock();
54         }
55      }
```

Lista - optymistyczna

OptimisticList

```
1       private boolean validate(Node pred, Node curr) {
2          Node node = head;
3          while(node.key <= pred.key) {
4             if(node == pred)
5                return pred.next == curr;
6             node = node.next;
7          }
8          return false;
9       }
```

OptimisticList

```
10      public boolean add(T item) {
11         int key = item.hashCode();
12         while(true) {
13            Node pred = head;
14            Node curr = pred.next;
15            while(curr.key < key) {
16               pred = curr; curr = curr.next;
17            }
18            pred.lock(); curr.lock();
19            try {
20               if(validate(pred, curr)) {
21                  if(curr.key == key) {
22                     return false;
23                  } else {
24                     Node node = new Node(item);
25                     node.next = curr;
26                     pred.next = node;
27                     return true;
28                  }
29               }
30            } finally {
31               pred.unlock(); curr.unlock();
32            }
33         }
34      }
```

OptimisticList

```
35      public boolean remove(T item) {
36         int key = item.hashCode();
37         while(true) {
38            Node pred = head;
39            Node curr = pred.next;
40            while(curr.key < key) {
41               pred = curr; curr = curr.next;
42            }
43            pred.lock(); curr.lock();
44            try {
45               if(validate(pred, curr)) {
46                  if(curr.key == key) {
47                     pred.next = curr.next;
48                     return true;
49                  } else {
50                     return false;
51                  }
52               }
53            } finally {
54               pred.unlock(); curr.unlock();
55            }
56         }
57      }
```

OptimisticList

```
58      public boolean contains(T item) {
59         int key = item.hashCode();
60         while(true) {
61            Node pred = head;
62            Node curr = pred.next;
63            while(curr.key < key) {
64               pred = curr; curr = curr.next;
65            }
66            pred.lock(); curr.lock();
67            try {
68               if(validate(pred, curr)) {
69                  return(curr.key == key);
70               }
71            } finally {
72               pred.unlock(); curr.unlock();
73            }
74         }
75      }
```

Lista - leniwa

LazyList

```
1       private boolean validate(Node pred, Node curr) {
2          return !pred.marked && !curr.marked && pred.next == curr;
3       }
```

LazyList

```
4       public boolean add(T item) {
5          int key = item.hashCode();
6          while(true) {
7             Node pred = head;
8             Node curr = head.next;
9             while(curr.key < key) {
10               pred = curr; curr = curr.next;
11            }
12            pred.lock();
13            try {
14               curr.lock();
15               try {
16                  if(validate(pred, curr)) {
17                     if(curr.key == key) {
18                        return false;
19                     } else {
20                        Node node = new Node(item);
21                        node.next = curr;
22                        pred.next = node;
23                        return true;
24                     }
25                  }
26               } finally {
27                  curr.unlock();
28               }
29            } finally {
30               pred.unlock();
31            }
32         }
33      }
```

LazyList

```
34      public boolean remove(T item) {
35         int key = item.hashCode();
36         while(true) {
37            Node pred = head;
38            Node curr = head.next;
39            while(curr.key < key) {
40               pred = curr; curr = curr.next;
41            }
42            pred.lock();
43            try {
44               curr.lock();
45               try {
46                  if(validate(pred, curr)) {
47                     if(curr.key != key) {
48                        return false;
49                     } else {
50                        curr.marked = true;
51                        pred.next = curr.next;
52                        return true;
53                     }
54                  }
55               } finally {
56                  curr.unlock();
57               }
58            } finally {
59               pred.unlock();
60            }
61         }
62      }
```

LazyList

```
63      public boolean contains(T item) {
64         int key = item.hashCode();
65         Node curr = head;
66         while(curr.key < key)
67            curr = curr.next;
68         return curr.key == key && !curr.marked;
69      }
```

Przykład praktyczny - kolejki

Typy kolejek (i nie tylko)

Pojemność:

- bounded
- unbounded

Metody:

- total
- partial
- synchronous

Kolejka - BoundedQueue (partial)

BoundedQueue

```
1   public class BoundedQueue<T> {
2      private class Node {
3         public T value;
4         public volatile Node next;
5
6         public Node(T value) {
7            this.value = value;
8         }
9      }
10
11     ReentrantLock enqLock = new ReentrantLock();
12     Condition notFullCondition = enqLock.newCondition();
13     ReentrantLock deqLock = new ReentrantLock();
14     Condition notEmptyCondition = deqLock.newCondition();
15
16     volatile Node head = new Node(null);
17     volatile tail = head;
18
19     AtomicInteger size = new AtomicInteger(0);
20     int capacity;
21
22     public BoundedQueue(int capacity) {
23        this.capacity = capacity;
24     }
```

BoundedQueue

```
25      public void enq(T x) {
26         boolean mustWakeDequeuers = false;
27         enqLock.lock();
28         try {
29            while(size.get() == capacity)
30               notFullCondition.await();
31            Node e = new Node(x);
32            tail.next = e;
33            tail = e;
34            if(size.getAndIncrement() == 0)
35               mustWakeDequeuers = true;
36         } finally {
37            enqLock.unlock();
38         }
39         if(mustWakeDequeuers) {
40            deqLock.lock();
41            try {
42               notEmptyCondition.signalAll();
43            } finally {
44               deqLock.unlock();
45            }
46         }
47      }
```

BoundedQueue

```
48      public T deq() {
49         T result;
50         boolean mustWakeEnqueuers = false;
51         deqLock.lock();
52         try {
53            while(size.get() == 0)
54               notEmptyCondition.await();
55            result = head.next.value;
56            head = head.next;
57            if(size.getAndDecrement() == capacity) {
58               mustWakeEnqueuers = true;
59            }
60         } finally {
61            deqLock.unlock();
62         }
63         if(mustWakeEnqueuers) {
64            enqLock.lock();
65            try {
66               notFullCondition.signalAll();
67            } finally {
68               enqLock.unlock();
69            }
70         }
71         return result;
72      }
73   }
```

Kolejka - UnboundedQueue (total)

UnboundedQueue

```
1       public void enq(T x) {
2          enqLock.lock();
3          try {
4             Node e = new Node(x);
5             tail.next = e;
6             tail = e;
7          } finally {
8             enqLock.unlock();
9          }
10      }
```

UnboundedQueue

```
11      public T deq() throws EmptyException {
12         T result;
13         deqLock.lock();
14         try {
15            if(head.next == null) {
16               throw new EmptyException();
17            }
18            result = head.next.value;
19            head = head.next;
20         } finally {
21            deqLock.unlock();
22         }
23         return result;
24      }
```

Kolejka - UnboundedQueue (total, bez blokad)

UnboundedQueue

```
1       private class Node {
2          public T value;
3          public AtomicReference<Node> next = new AtomicReference<Node>(null);
4
5          public Node(T value) {
6             this.value = value;
7          }
8       }
```

UnboundedQueue

```
 9      public void enq(T value) {
10         Node node = new Node(value);
11         while(true) {
12            Node last = tail.get();
13            Node next = last.next.get();
14            if(last == tail.get()) {
15               if(next == null) {
16                  if(last.next.compareAndSet(next, node)) {
17                     tail.compareAndSet(last, node);
18                     return;
19                  }
20               } else {
21                  tail.compareAndSet(last, next);
22               }
23            }
24         }
25      }
```

UnboundedQueue

```
26      public T deq() throws EmptyException {
27         while(true) {
28            Node first = head.get();
29            Node last = tail.get();
30            Node next = first.next.get();
31            if(first == head.get()) {
32               if(first == last) {
33                  if(next == null) {
34                     throw new EmptyException();
35                  }
36                  tail.compareAndSet(last, next);
37               } else {
38                  T value = next.value;
39                  if(head.compareAndSet(first, next))
40                     return value;
41               }
42            }
43         }
44      }
```

Kolejka - SynchronousQueue

SynchronousQueue

```
1   public class SynchronousQueue<T> {
2      private T item = null;
3      private boolean enqueuing;
4      private Lock lock;
5      private Condition condition;
6
7      ...
```

SynchronousQueue

```
 8      public void enq(T value) {
 9         lock.lock();
10         try {
11            while(enqueuing)
12               condition.await();
13            enqueuing = true;
14            item = value;
15            condition.signalAll();
16            while(item != null)
17               condition.await();
18            enqueuing = false;
19            condition.signalAll();
20         } finally {
21            lock.unlock();
22         }
23      }
```

SynchronousQueue

```
24      public T deq() {
25         lock.lock();
26         try {
27            while(item == null)
28               condition.await();
29            T t = item;
30            item = null;
31            condition.signalAll();
32            return t;
33         } finally {
34            lock.unlock();
35         }
36      }
```

Pule wątków

Tworzenie nowych wątków w Java

(po 100 slajdach i 6 wykładach!)

Thread

java.lang.Thread

- Thread()
- Thread(Runnable target)
- void run()
- void start()
- void interrupt()
- bool interrupted() / bool isInterrupted()
- void join() / void join(long millis)

Przykład - Hello World!

```
1   Thread t1 = new Thread() {
2      @Override
3      public void run() {
4         while(!isInterrupted()) {
5            System.out.println("Hello, multithreaded world!");
6            try {
7                  sleep(1000);
8            } catch (InterruptedException e) {
9               return;
10           }
11        }
12     }
13  };
14
15  t1.start();
16  try {
17     Thread.sleep(5000);
18     t1.interrupt();
19     t1.join();
20  } catch (InterruptedException e) {}
```

Runnable

java.lang.Runnable

- void run()

Przykład - Hello World! II

```
1   Runnable r1 = new Runnable() {
2      @Override
3      public void run() {
4         while(!Thread.currentThread().isInterrupted()) {
5            System.out.println("Hello, multithreaded world!");
6            try {
7               Thread.sleep(1000);
8            } catch (InterruptedException e) {
9               return;
10           }
11        }
12     }
13  };
14  Thread t1 = new Thread(r1);
15  t1.start();
16
17  try {
18     Thread.sleep(5000);
19     t1.interrupt();
20     t1.join();
21  } catch (InterruptedException e) {}
```

Przykład - mnożenie macierzy

Przykład - mnożenie macierzy

```
1   class MMThread {
2      double[][] a, b, c;
3      int n;
4
5      private class Worker extends Thread {
6         int row, col;
7
8         Worker(int row, int col) {
9            this.row = row; col = this.col;
10        }
11
12        public void run() {
13           double dotProduct = 0.0;
14           for(int i = 0; i < n; i++)
15              dotProduct += a[row][i] * b[i][col];
16           c[row][col] = dotProduct;
17        }
18     }
19
20     public MMThread(double[][] a, double[][] b) {
21        n = a.length;
22        this.a = a;
23        this.b = b;
24        c = new double[n][n];
25     }
```

Przykład - mnożenie macierzy

```
26      void multiply() {
27         Worker[][] worker = new Worker[n][n];
28
29         for(int row = 0; row < n; ++row)
30            for(int col = 0; col < n; ++col)
31               worker[row][col] = new Worker(row,col);
32
33         for(int row = 0; row < n; ++row)
34            for(int col = 0; col < n; ++col)
35               worker[row][col].start();
36
37         for(int row = 0; row < n; ++row)
38            for (int col = 0; col < n; ++col)
39               worker[row][col].join();
40      }
41   }
```

Executor i ExecutorService

Executors

java.util.concurrent.Executors

- ExecutorService newCachedThreadPool()
- ExecutorService newFixedThreadPool(int nThreads)

ExecutorService

java.util.concurrent.ExecutorService

- void execute(Runnable command)
- Future<?> submit(Runnable task)
- Future<T> submit(Runnable task, T result)
- Future<T> submit(Callable<T> task)
- List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks)
- void shutdown()
- boolean awaitTermination(long timeout, TimeUnit unit)

Przykład - mnożenie macierzy

```
26      void multiply() {
27         ExecutorService es =
                Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
28
29         for(int row = 0; row < n; ++row)
30            for(int col = 0; col < n; ++col)
31               es.execute(new Worker(row,col));
32
33         es.shutdown();
34         while(true) {
35            try {
36               if(es.awaitTermination(1, TimeUnit.HOURS))
37                  return;
38            } catch (InterruptedException e) {}
39         }
40      }
41   }
```

(No) Future

Callable

java.util.concurrent.Callable<V>

- V call()

Future

java.util.concurrent.Future<V>

- V get()
- V get(long timeout, TimeUnit unit)
- boolean cancel(boolean mayInterruptIfRunning)
- boolean isCancelled()
- boolean isDone()

java.util.concurrent.FutureTask<V>

- FutureTask(Callable<V> callable)
- FutureTask(Runnable runnable, V result)
- void run()

Przykład - cache

```
1  public class Memorizer<A, V> implements Computable<A, V> {
2     private final ConcurrentMap<A, Future<V>> cache =
3        new ConcurrentHashMap<A, Future<V>>();
4     private final Computable<A, V> c;
5
6     public Memorizer(Computable<A, V> c) { this.c = c; }
7
8     public V compute(A arg) throws InterruptedException {
9        while(true) {
10          Future<V> f = cache.get(arg);
11          if(f == null) {
12             Callable<V> eval = new Callable<V>() {
13                public V call() throws InterruptedException {
14                   return c.compute(arg);
15                }
16             };
17             FutureTask<V> ft = new FutureTask<V>(eval);
18             f = cache.putIfAbsent(arg, ft);
19             if(f == null) {
20                f = ft;
21                ft.run();
22             }
23          }
24          try {
25             return f.get();
26          } catch (CancellationException e) {
27             cache.remove(arg, f);
28          } catch (ExecutionException e) {
29             throw new RuntimeException(e.getCause());
30          }
31       }
32    }
33 }
```

Przykład - generowanie obrazków

Przykład - generowanie obrazków

```
1   private static class RandomImageCreator implements Callable<BufferedImage> {
2      private Random random = new Random();
3      private int size;
4
5      public RandomImageCreator(int size) {
6         this.size = size;
7      }
8
9      @Override
10     public BufferedImage call() throws Exception {
11        BufferedImage bi
12           = new BufferedImage(size, size, BufferedImage.TYPE_INT_ARGB);
13        for(int w = 0; w < size; ++w)
14           for(int h = 0; h < size; ++h) {
15              Color c = new Color(random.nextInt(256),
16                                  random.nextInt(256),
17                                  random.nextInt(256));
18              bi.setRGB(w, h, c.getRGB());
19           }
20        return bi;
21     }
22  }
```

Przykład - generowanie obrazków

```
23  ExecutorService es
24     = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
25
26  int numberOfImages = 1000;
27  int size = 4096;
28
29  ArrayList<Future<BufferedImage>> list = new ArrayList<>(numberOfImages);
30  for(int i = 0; i < numberOfImages; ++i)
31     list.add(es.submit(new RandomImageCreator(size)));
32
33  for(Future<BufferedImage> f : list)
34     System.out.println(f.get().getWidth());
```

ForkJoinPool

ForkJoinPool

java.util.concurrent.ForkJoinPool

- static ForkJoinPool commonPool()
- ForkJoinPool(int parallelism)
- void execute(ForkJoinTask<?> task)
- T invoke(ForkJoinTask<T> task)
- ForkJoinTask<T> submit(ForkJoinTask<T> task)

ForkJoinPool

java.util.concurrent.ForkJoinTask<V>

- ForkJoinTask<V> fork()
- V invoke()
- V join()

java.util.concurrent.RecursiveAction

- protected abstract void compute()

java.util.concurrent.RecursiveTask<V>

- protected abstract V compute()

Przykład - suma liczb w tablicy

Przykład - suma liczb w tablicy

```
1   private static class IntSummator extends RecursiveTask<Integer> {
2      private int[] array;
3      private int offset;
4      private int size;
5
6      public IntSummator(int[] array, int offset, int size) {
7         this.array = array;
8         this.offset = offset;
9         this.size = size;
10     }
11
12     @Override
13     public Integer compute() {
14        if(size == 2)
15           return array[offset] + array[offset + 1];
16
17        int newSize = size / 2;
18
19        IntSummator task1 = new IntSummator(array, offset, newSize);
20        IntSummator task2 = new IntSummator(array, offset + newSize, newSize);
21
22        task1.fork();
23        int value2 = task2.compute();
24        int value1 = task1.join();
25
26        return value1 + value2;
27     }
28  }
```

Przykład - suma liczb w tablicy

```
29  ForkJoinPool pool = new ForkJoinPool(Runtime.getRuntime().availableProcessors());
30  int[] array = IntStream.generate(() -> 1).limit(1 << 20).toArray();
31  IntSummator summator = new IntSummator(array, 0, array.length);
32  System.out.println(pool.invoke(summator));
```

Przykład - suma liczb w tablicy

```
29   int[] array = IntStream.generate(() -> 1).limit(1 << 20).toArray();
30   IntSummator summator = new IntSummator(array, 0, array.length);
31   System.out.println(summator.invoke());
```

Przykład - suma liczb w tablicy

```
22   task1.fork();
23   int value2 = task2.compute();
24   int value1 = task1.join();
```

```
22   task1.fork();
23   int value2 = task2.invoke();
24   int value1 = task1.join();
```

```
22   task1.fork();
23   task2.fork();
24   int value2 = task2.join();
25   int value1 = task1.join();
```

```
22   task1.fork();
23   task2.fork();
24   int value1 = task1.join();
25   int value2 = task2.join();
```

Jak to panie działa?!

Jak to panie działa?!

ZŁODZIEJSTWO W BIAŁY DZIEŃ!

WorkStealingThread

```
1  public class WorkStealingThread {
2     DEQueue[] queue;
3     Random random = new Random();
4
5     public WorkStealingThread(DEQueue[] myQueue) {
6        queue = myQueue;
7     }
8
9     public void run() {
10       int me = ThreadID.get();
11       Runnable task = queue[me].popBottom();
12       while(true) {
13          while(task != null) {
14             task.run();
15             task = queue[me].popBottom();
16          }
17          while(task == null) {
18             Thread.yield();
19             int victim = random.nextInt(queue.length);
20             if(!queue[victim].isEmpty()) {
21                task = queue[victim].popTop();
22             }
23          }
24       }
25    }
26 }
```

BDEQueue (prawie)

```
1   public class BDEQueue {
2      Runnable[] tasks;
3      volatile int bottom;
4      AtomicReference<Integer> top;
5
6      public BDEQueue(int capacity) {
7         tasks = new Runnable[capacity];
8         top = new AtomicReference<Integer>(0);
9         bottom = 0;
10     }
11
12     public void pushBottom(Runnable r){
13        tasks[bottom] = r;
14        bottom++;
15     }
16
17     boolean isEmpty() {
18        int localTop = top.get();
19        int localBottom = bottom;
20        return localBottom <= localTop;
21     }
```

BDEQueue (prawie)

```
22      public Runnable popTop() {
23         int oldTop = top.get(), newTop = oldTop + 1;
24         if(bottom <= oldTop)
25            return null;
26         Runnable r = tasks[oldTop];
27         if(top.compareAndSet(oldTop, newTop))
28            return r;
29          return null;
30      }
31      public Runnable popBottom() {
32         if(bottom == 0)
33            return null;
34         bottom--;
35         Runnable r = tasks[bottom];
36         int oldTop = top.get(), newTop = 0;
37         if(bottom > oldTop)
38            return r;
39         if(bottom == oldTop) {
40            bottom = 0;
41            if(top.compareAndSet(oldTop, newTop))
42               return r;
43         }
44         bottom = 0;
45         top.set(newTop);
46         return null;
47      }
48   }
```

BDEQueue

```
1   public class BDEQueue {
2      Runnable[] tasks;
3      volatile int bottom;
4      AtomicStampedReference<Integer> top;
5
6      public BDEQueue(int capacity) {
7         tasks = new Runnable[capacity];
8         top = new AtomicStampedReference<Integer>(0, 0);
9         bottom = 0;
10     }
11
12     public void pushBottom(Runnable r){
13        tasks[bottom] = r;
14        bottom++;
15     }
16
17     boolean isEmpty() {
18        int localTop = top.getReference();
19        int localBottom = bottom;
20        return localBottom <= localTop;
21     }
```

BDEQueue

```
22      public Runnable popTop() {
23         int[] stamp = new int[1];
24         int oldTop = top.get(stamp), newTop = oldTop + 1;
25         int oldStamp = stamp[0], newStamp = oldStamp + 1;
26         if(bottom <= oldTop)
27            return null;
28         Runnable r = tasks[oldTop];
29         if(top.compareAndSet(oldTop, newTop, oldStamp, newStamp))
30            return r;
31          return null;
32      }
33      public Runnable popBottom() {
34         if(bottom == 0)
35            return null;
36         bottom--;
37         Runnable r = tasks[bottom];
38         int[] stamp = new int[1];
39         int oldTop = top.get(stamp), newTop = 0;
40         int oldStamp = stamp[0], newStamp = oldStamp + 1;
41         if(bottom > oldTop)
42            return r;
43         if(bottom == oldTop) {
44            bottom = 0;
45            if(top.compareAndSet(oldTop, newTop, oldStamp, newStamp))
46               return r;
47         }
48         bottom = 0;
49         top.set(newTop,newStamp);
50         return null;
51      }
52   }
```

Bariery

Bariery - przykładowy problem

```
1   while(true) {
2       frame.prepare();
3       frame.display();
4   }
```

Bariery - przykładowy problem

```
1   int me = ThreadID.get();
2   while(true) {
3      frame[me].prepare();
4      frame[me].display();
5   }
```

Bariery - przykładowy problem

```
1   public interface Barrier {
2      public void await();
3   }
4
5   private Barrier b;
6
7   while(true) {
8      frame[my].prepare();
9      b.await();
10     frame[my].display();
11  }
```

SimpleBarrier

```
1  public class SimpleBarrier implements Barrier {
2     AtomicInteger count;
3     int size;
4
5     public SimpleBarrier(int n){
6        count = new AtomicInteger(n);
7        size = n;
8     }
9
10    public void await() {
11       int position = count.getAndDecrement();
12       if(position == 1) {
13          count.set(size);
14       } else {
15          while(count.get() != 0) {};
16       }
17    }
18 }
```

SimpleBarrier

```
1   public SenseBarrier(int n) {
2      count = new AtomicInteger(n);
3      size = n;
4      sense = false;
5      threadSense = new ThreadLocal<Boolean>() {
6         protected Boolean initialValue() { return !sense; };
7      };
8   }
9
10  public void await() {
11     boolean mySense = threadSense.get();
12     int position = count.getAndDecrement();
13     if(position == 1) {
14        count.set(size);
15        sense = mySense;
16     } else {
17        while(sense != mySense) {}
18     }
19     threadSense.set(!mySense);
20  }
```

Tymczasem w Java

java.util.concurrent.CountDownLatch

- CountDownLatch(int count)
- void await()
- boolean await(long timeout, TimeUnit unit)
- void countDown()
- long getCount()

java.util.concurrent.CyclicBarrier

- CyclicBarrier(int parties)
- CyclicBarrier(int parties, Runnable barrierAction)
- int await()
- int await(long timeout, TimeUnit unit)
- int getNumberWaiting()
- int getParties()
- boolean isBroken()
- void reset()

Przykład - prosty model opinii

```
1  public class SimpleOpinionModel {
2     private CyclicBarrier barrier;
3     private int[][] opinions;
4     private int length, steps;
5
6     private class WorkerThread extends Thread {
7        private int offset, elements;
8        public WorkerThread(int offset, int elements) {
9           this.offset = offset;
10          this.elements = elements;
11       }
12       public void run() {
13          boolean even = true;
14          for(int i = 0; i < steps; ++i) {
15             int prevIndex = even ? 0 : 1;
16             int nextIndex = 1 - prevIndex;
17
18             for(int j = offset; j < offset + elements; ++j) {
19                int left = (j - 1 + length) % length;
20                int right = (left + 2) % length;
21                if(opinions[prevIndex][left] == opinions[prevIndex][right])
22                   opinions[nextIndex][j] = opinions[prevIndex][left];
23                else
24                   opinions[nextIndex][j] = opinions[prevIndex][j];
25             }
26             even = !even;
27             try {
28                barrier.await();
29             } catch (Exception e) {}
30          }
31       }
32    }
```

Przykład - prosty model opinii

```
33      public int[] runSimulation(int[] initialCondition, int steps) {
34          this.steps = steps;
35          length = initialCondition.length;
36          opinions = new int[2][];
37          opinions[0] = Arrays.copyOf(initialCondition, length);
38          opinions[1] = new int[length];
39
40          int numThreads = Runtime.getRuntime().availableProcessors();
41          int elements = initialCondition.length / numThreads;
42
43          WorkerThread[] threads = new
                WorkerThread[Runtime.getRuntime().availableProcessors()];
44          for(int i = 0; i < threads.length; ++i)
45              threads[i] = new WorkerThread(i * elements, i != threads.length - 1 ?
                    elements : initialCondition.length - (numThreads - 1) * elements);
46
47          for(int i = 0; i < threads.length; ++i)
48              threads[i].start();
49
50          try {
51              for(int i = 0; i < threads.length; ++i)
52              threads[i].join();
53          } catch (InterruptedException e) {
54              throw new RuntimeException(e);
55          }
56
57          if(steps % 2 == 0)
58              return opinions[0];
59          return opinions[1];
60      }
61  }
```

Wyrażenia lambda i strumienie

Interfejsy funkcyjne

java.util.function.Function<T,R>

- R apply(T t)
- default <V> Function<T,V> andThen(Function<? super R,? extends V> after)
- default <V> Function<V,R> compose(Function<? super V,? extends T> before)
- static <T> Function<T,T> identity()

Interfejsy funkcyjne

java.util.function.Function<T,R>

- R apply(T t)
- default <V> Function<T,V>
  andThen(Function<? super R,? extends V> after)
- default <V> Function<V,R>
  compose(Function<? super V,? extends T> before)
- static <T> Function<T,T> identity()

Interfejsy funkcyjne

java.util.function

- BiFunction<T,U,R>
- BinaryOperator<T>
- Consumer<T>
- DoubleFunction<R>
- Supplier<T>
- ...

Wyrażenia lambda

```
1   class Student {
2      private String name;
3      private int group;
4
5      public Student(String name, int group) {
6         this.name = name;
7         this.group = group;
8      }
9
10     public String getName() { return name; }
11     public void setName(String name) { this.name = name; }
12     public int getGroup() { return group; }
13     public void setGroup(int group) { this.group = group; }
14
15     @Override
16     public String toString() {
17        return name + " (" + group + ")";
18     }
19
20     public boolean isSameGroup(Student s) {
21        return group == s.group;
22     }
23
24     public static Student defaultStudent(int group) {
25        return new Student("John Default", group);
26     }
27
28  }
```

Wyrażenia lambda

```
1  Function<Student, String> nameFunction = new Function<Streams.Student, String>() {
2      @Override
3      public String apply(Student student) {
4          return student.getName();
5      }
6  };
```

Wyrażenia lambda

```
1  var nameFunction = new Function<Streams.Student, String>() {
2     @Override
3     public String apply(Student student) {
4        return student.getName();
5     }
6  };
```

Wyrażenia lambda

```
1   Function<Student, String> nameFunction = (Student s) -> s.getName();
2   Function<Student, String> nameFunction = (s) -> s.getName();
3   Function<Student, String> nameFunction = s -> s.getName();
4   Function<Student, String> nameFunction = Student::getName;
5
6   IntFunction<Student> intFunction = Student::defaultStudent;
7
8   BiFunction<String, Integer, Student> biFunction = Student::new;
9
10  BiFunction<Student, Student, Boolean> biFunction = Student::isSameGroup;
11  BiFunction<Student, Student, Boolean> biFunction = (s1, s2) -> s1.isSameGroup(s2);
12
13  IntFunction<Student[]> arrayFactory = Student[]::new;
```

Strumienie

```
1   Student[] students = {new Student("Feliksa", 1),
2                         new Student("Bozydar", 2),
3                         new Student("Wieslawa", 2),
4                         new Student("Szczepan", 2),
5                         new Student("Apolonia", 3)};
6
7   for(int i = 0; i < students.length; ++i)
8      if(students[i].getGroup() == 3)
9         System.out.println(students[i]);
```

Strumienie

```
1   Student[] students = {new Student("Feliksa", 1),
2                         new Student("Bozydar", 2),
3                         new Student("Wieslawa", 2),
4                         new Student("Szczepan", 2),
5                         new Student("Apolonia", 3)};
6
7   for(Student s : students)
8      if(s.getGroup() == 3)
9         System.out.println(s);
```

Strumienie

```
1   Student[] students = {new Student("Feliksa", 1),
2                         new Student("Bozydar", 2),
3                         new Student("Wieslawa", 2),
4                         new Student("Szczepan", 2),
5                         new Student("Apolonia", 3)};
6
7   Arrays.stream(students)
8      .filter(s -> s.getGroup() == 3)
9      .forEach(System.out::println);
```

Strumienie

```
1   Student[] students = {new Student("Feliksa", 1),
2                         new Student("Bozydar", 2),
3                         new Student("Wieslawa", 2),
4                         new Student("Szczepan", 2),
5                         new Student("Apolonia", 3)};
6
7   LinkedList<String> list = new LinkedList<>();
8   for(Student s : students)
9      if(s.getGroup() == 2)
10        list.add(s.toString());
11
12  List<String> list = Arrays.stream(students)
13        .filter(s -> s.getGroup() == 2)
14        .map(Student::toString)
15        .collect(Collectors.toList());
16
17  LinkedList<String> list = Arrays.stream(students)
18        .filter(s -> s.getGroup() == 2)
19        .map(Student::toString)
20        .collect(Collectors.toCollection(LinkedList::new));
```

Strumienie

```
1   Stream.generate(() -> new String("Hello!"))
2      .limit(10)
3      .forEach(System.out::println);
4
5   Stream.iterate(new Student("John", 0), s -> new Student("John", s.getGroup() + 1))
6      .limit(10)
7      .forEach(System.out::println);
8
9   IntStream.range(0, 10)
10     .forEach(System.out::println);
11
12  random.ints(10, 0, 50)
13     .forEach(System.out::println);
```

Strumienie

```
1   int[] arr = IntStream.generate(() -> 1)
2         .limit(100)
3         .toArray();
4
5   int sum = Arrays.stream(arr)
6         .parallel()
7         .sum();
8
9   int sum = Arrays.stream(arr)
10        .parallel()
11        .reduce(0, Integer::sum);
```

Strumienie

```
1   Random random = new Random();
2   Student[] array = random.ints(100, 0, 5)
3      .parallel()
4      .mapToObj(Student::defaultStudent)
5      .sorted((s1, s2) -> Integer.compare(s1.getGroup(), s2.getGroup()))
6      .toArray(Student[]::new);
```