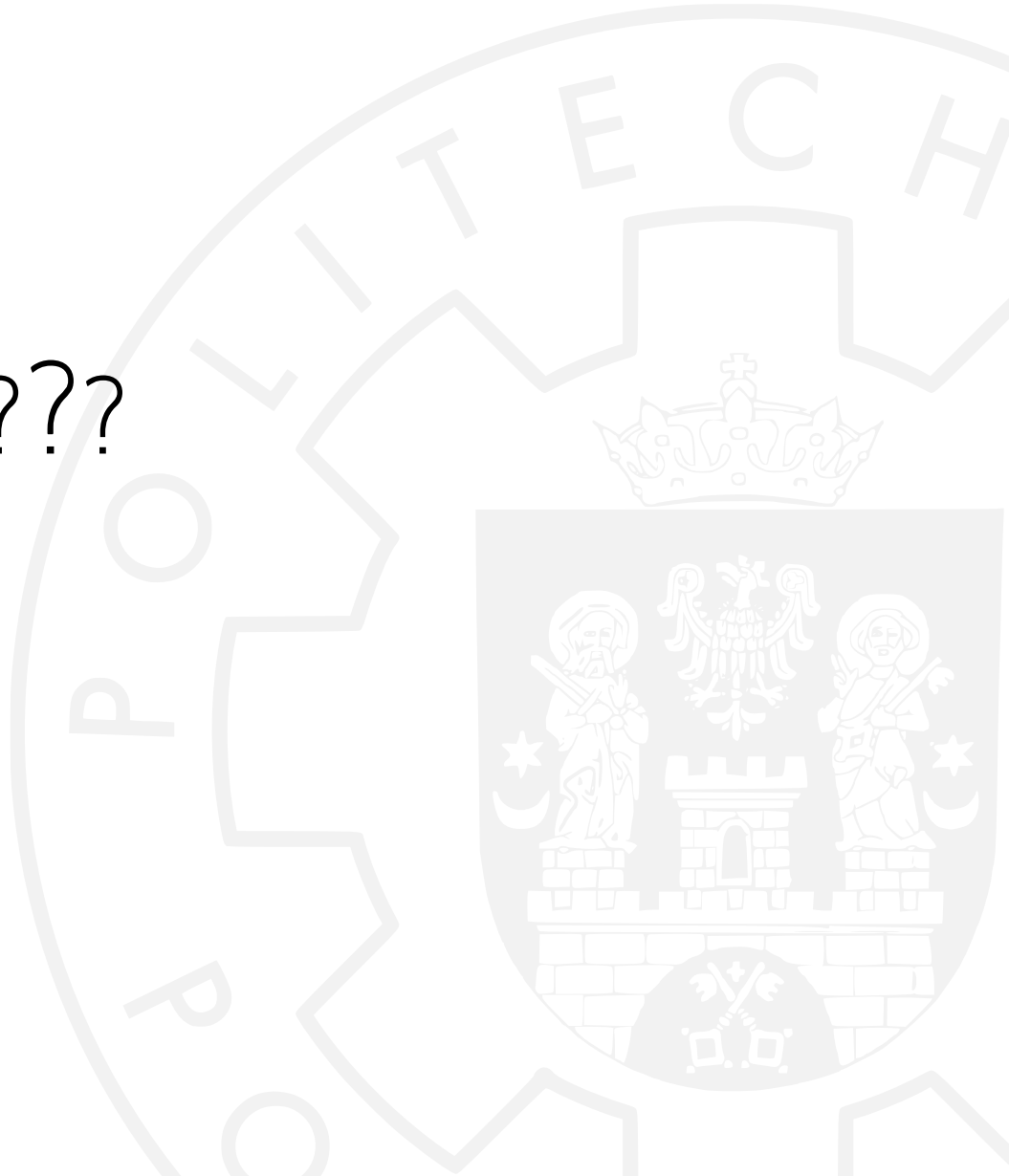
Bezbolesne Programowanie Współbieżne

Konrad Siek

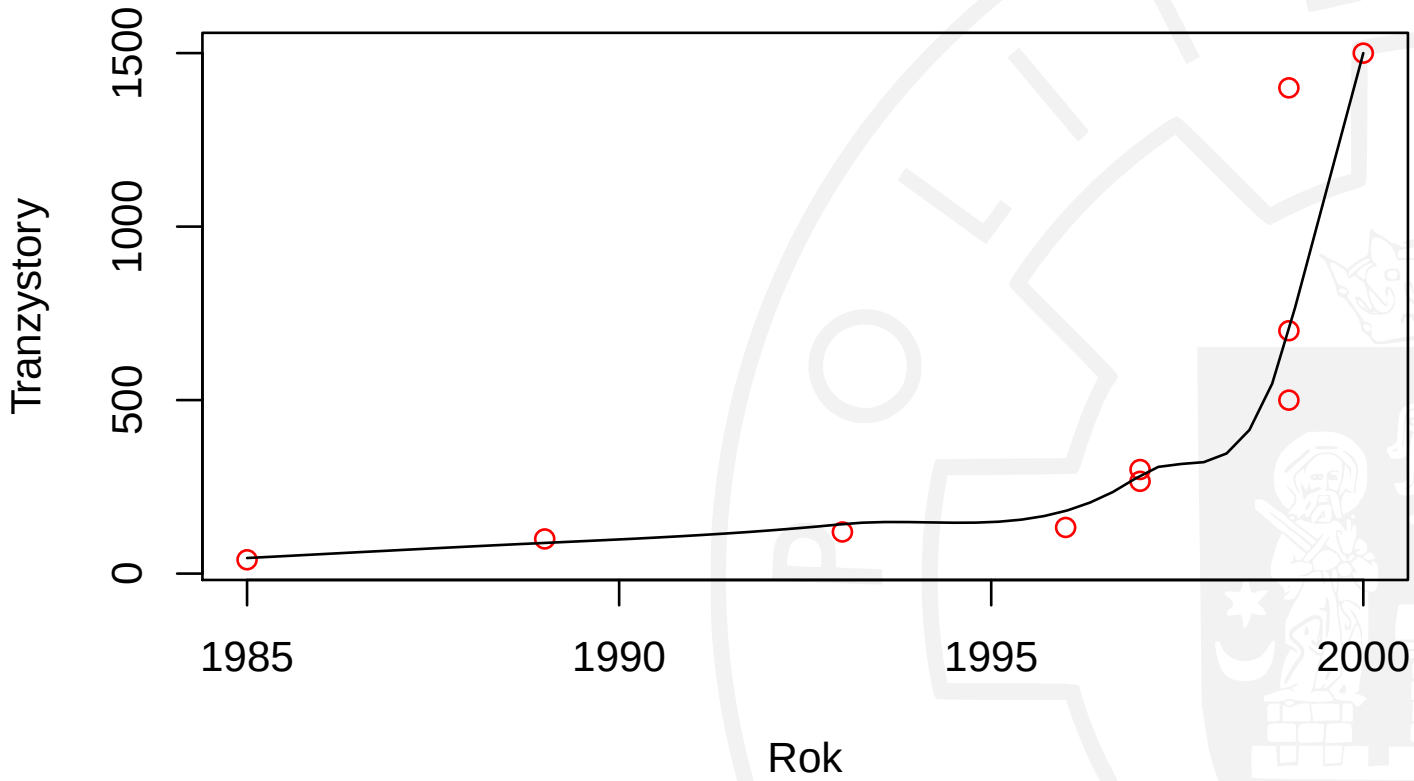
9 maj 2012

Lata 90-te

???



Częstotliwość procesorów

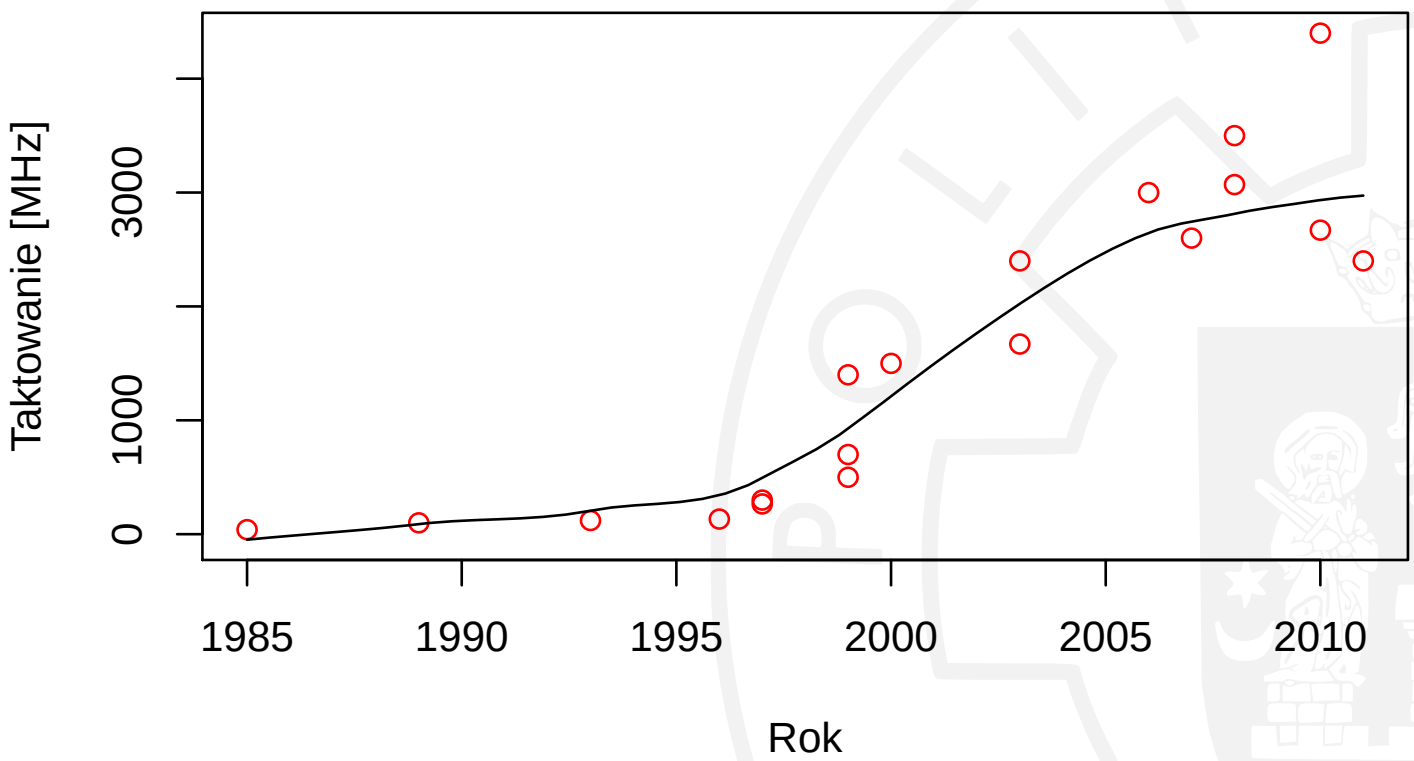


Lata 90-te

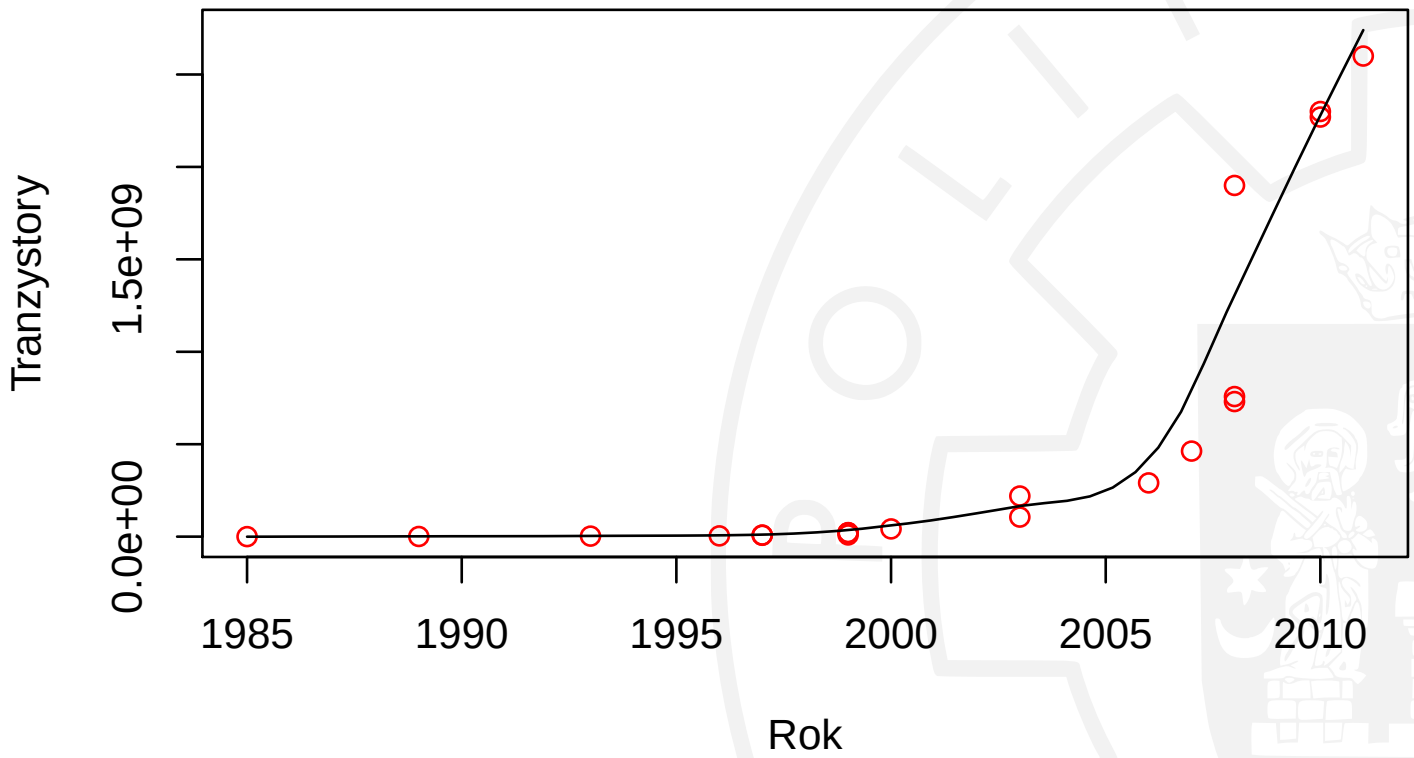
My new computer's got the clocks, it rocks,
but it was **obsolete before I opened the box**.
You say you've **had your desktop for over a week**?
Throw that junk away, man, it's an **antique**.
Your laptop is **a month old**? Well, that's great.
If you could use a nice, **heavy paperweight**...

It's all about the Pentiums, Weird Al, 1999

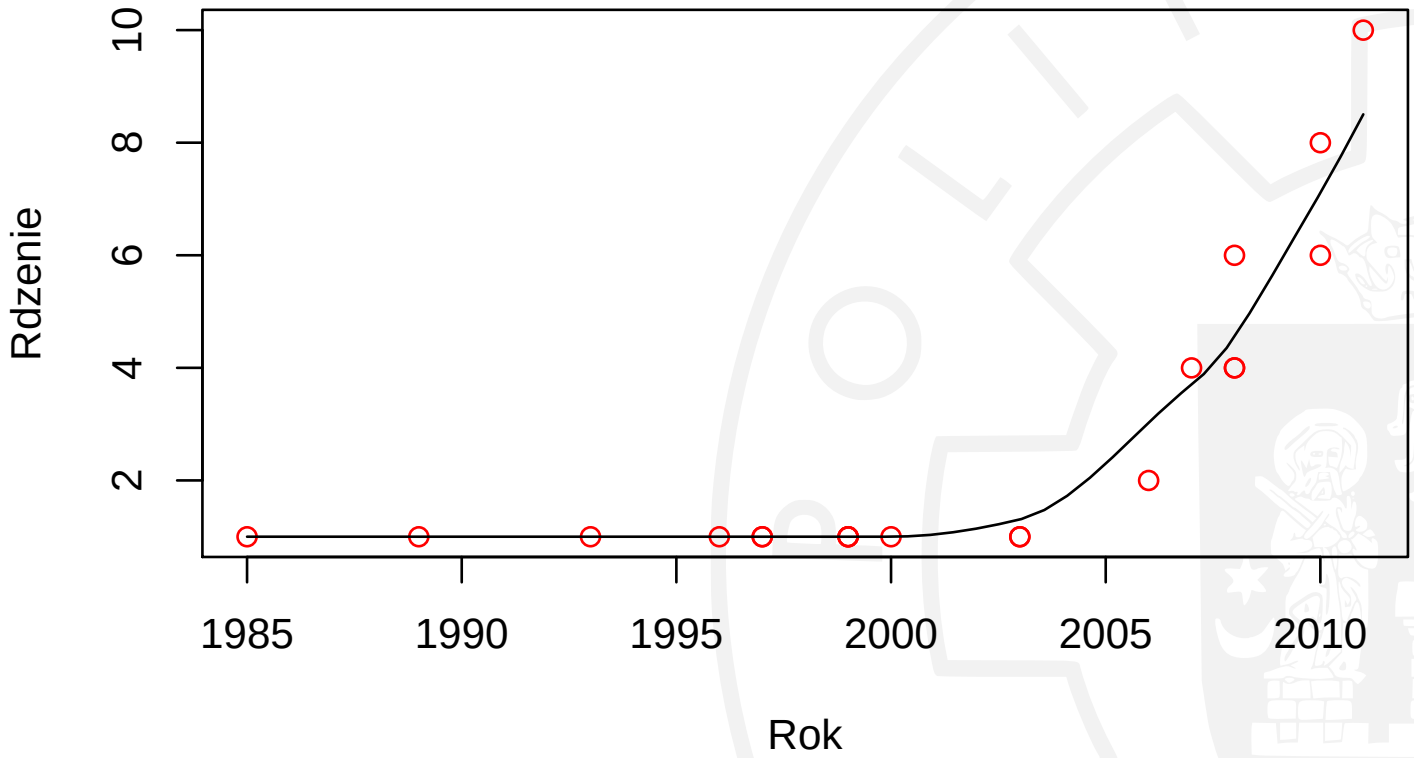
Częstotliwość procesorów



Prawo Moore-a *in action*



Liczby rdzeni w procesorach

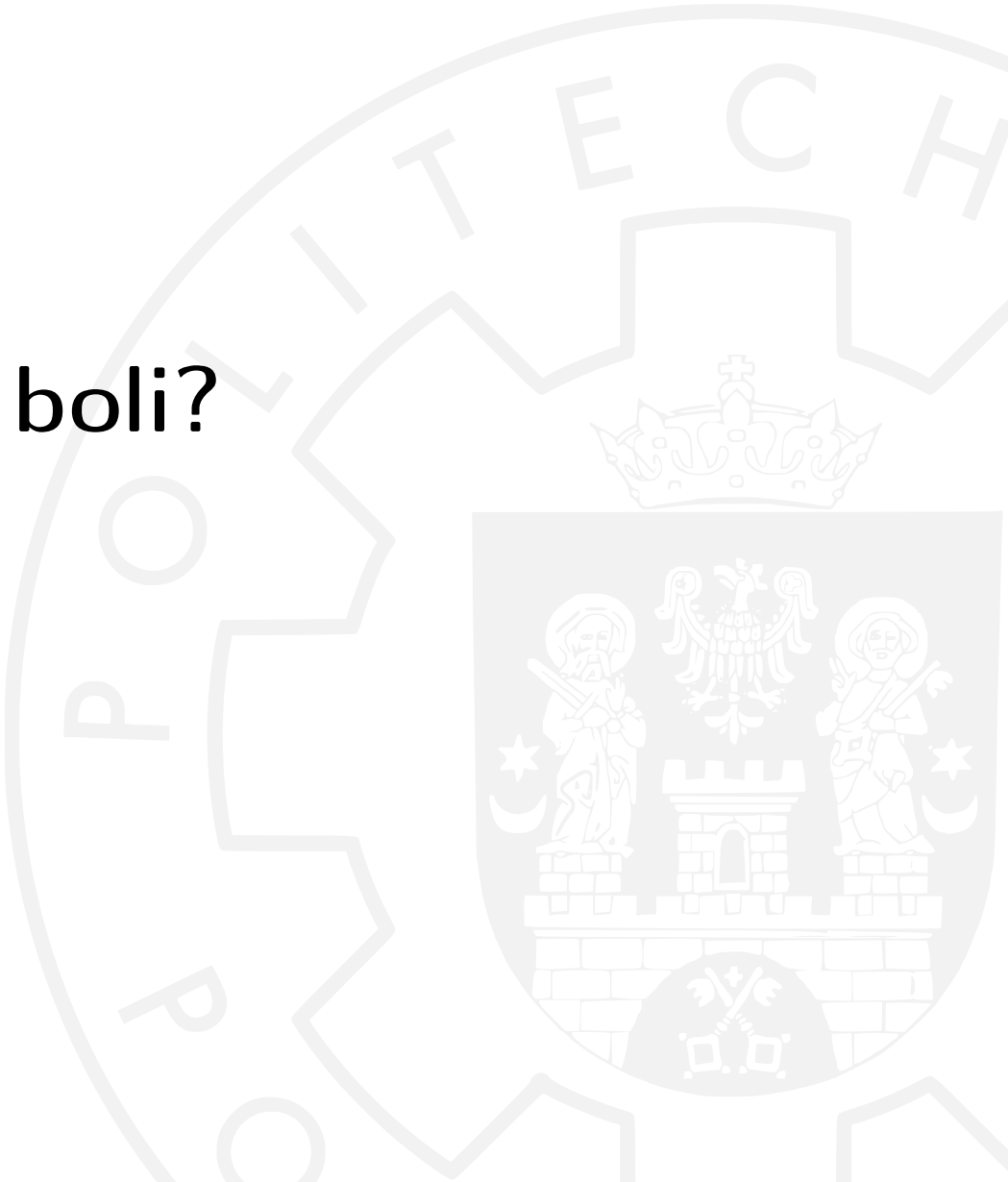


Współbieżność

Współbieżność

. . . ale współbieżność boli

. . . boli?



Rzeczy których nie lubie

- Szukanie bugów (bagów?)
- Wrzucanie kodu strukturalnego do funkcjonalnego
- Rozwiązywanie problemów synchronizacji

Rzeczy których nie lubie

- Szukanie bugów **(bagów?)**
- Wrzucanie kodu strukturalnego do funkcjonalnego
- Rozwiązywanie problemów synchronizacji

Rzeczy których nie lubie

- Szukanie **Heisenbugów**
- Wrzucanie kodu strukturalnego do funkcjonalnego
- Rozwiązywanie problemów synchronizacji

Heisenbug

Rzeczy których nie lubie

- Szukanie **Heisenbugów**
- Wrzucanie kodu strukturalnego do funkcjonalnego
- Rozwiązywanie problemów synchronizacji

Rzeczy których nie lubie

- Szukanie **Heisenbugów**
- Wrzucanie kodu **strukturalnego** do **funkcjonalnego**
- Rozwiązywanie problemów synchronizacji

Wrzucanie kodu strukturalnego do funkcjonalnego

```
class Elem:
    def __init__(self, transaction, t, next):
        self.transaction = transaction
        self.t = t
        self.next = next

_head = None

def insert(transaction, t):
    if _head is None:
        _head = Elem(transaction, t, None)
        return
    elem = _head
    while True:
        if elem.t <= t:
            temp = Elem(elem.transaction, elem.t, elem.next)
            elem.transaction = transaction
            elem.t = t
            elem.next = temp
            return
        else:
            if elem.next:
                elem = elem.next
            else:
                elem.next = Elem(transaction, t, None)
                return
```

Wrzucanie kodu strukturalnego do funkcjonalnego

```
from threading import RLock, Condition

class Elem:
    def __init__(self, transaction, t, next):
        self.transaction = transaction
        self.t = t
        self.next = next

_head = None

_global_lock = Condition()
_transaction_lock = RLock()

def insert(head, transaction, t):
    elem = head
    while True:
        if elem.t <= t:
            temp = Elem(elem.transaction, elem.t, elem.next)
            elem.transaction = transaction
            elem.t = t
            elem.next = temp
            return
        else:
            if elem.next:
                elem = elem.next
            else:
                elem.next = Elem(transaction, t, None)
                return
```

Wrzucanie kodu strukturalnego do funkcjonalnego

```
from threading import RLock, Condition

class Elem:
    def __init__(self, transaction, t, next):
        self.transaction = transaction
        self.t = t
        self.next = next

_head = None

_global_lock = Condition()
_transaction_lock = RLock()

def insert(head, transaction, t):
    elem = head
    while True:
        if elem.t <= t:
            temp = Elem(elem.transaction, elem.t, elem.next)
            elem.transaction = transaction
            elem.t = t
            elem.next = temp
            return
        else:
            if elem.next:
                elem = elem.next
            else:
                elem.next = Elem(transaction, t, None)
                return
```

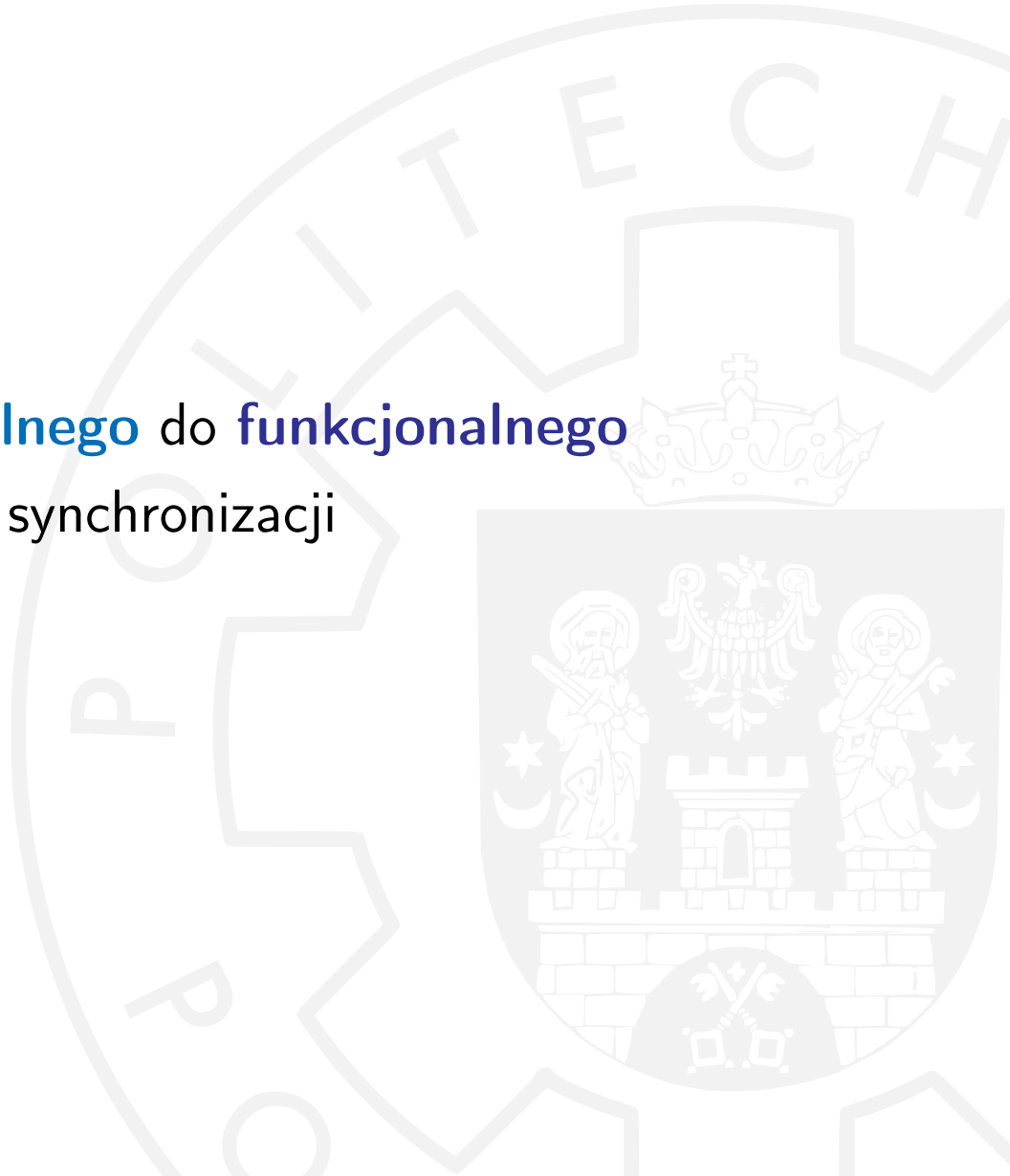
```
def lock(transaction, length):
    _global_lock.acquire()
    if _head is None:
        acquired = _transaction_lock.acquire(blocking=False)
    if acquired and _head is None:
        _global_lock.release()
        return
    else:
        if _head == null:
            _head = Elem(T, t, null)
        else:
            insert(_head, T, t)
    while True:
        _global_lock.wait()
        if _head.T = T:
            _transaction_lock.acquire()
            if _head is not None:
                _head = _head.next
            _global_lock.release()
            return
        else:
            _global_lock.notify()

def unlock():
    _global_lock.acquire()
    _transaction_lock.release()
    _global_lock.notify()
    _global_lock.release()
    return
```

Rzeczy których nie lubie

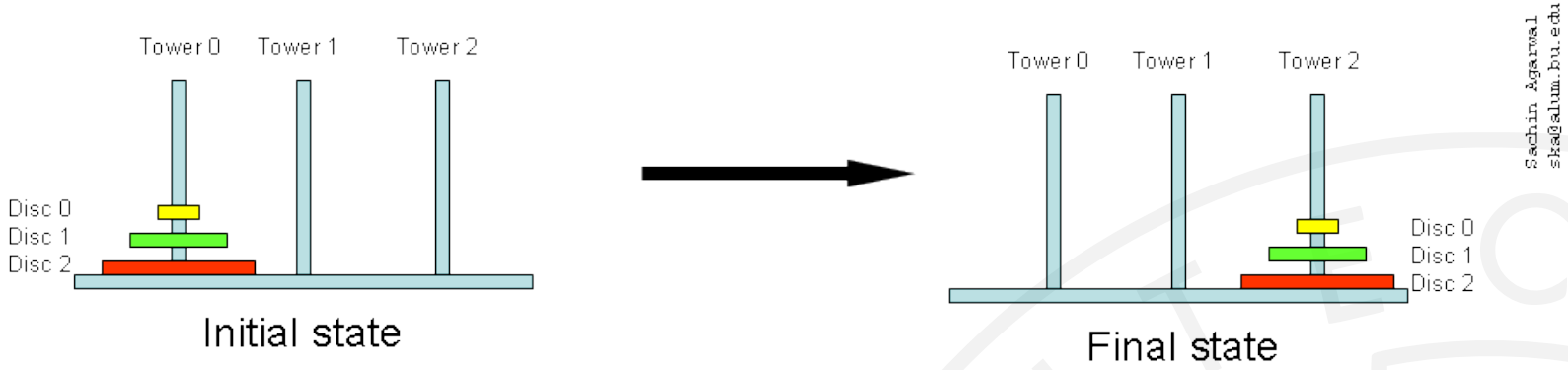
- Szukanie **Heisenbugów**
- Wrzucanie kodu **strukturalnego** do **funkcjonalnego**
- Rozwiązywanie problemów synchronizacji

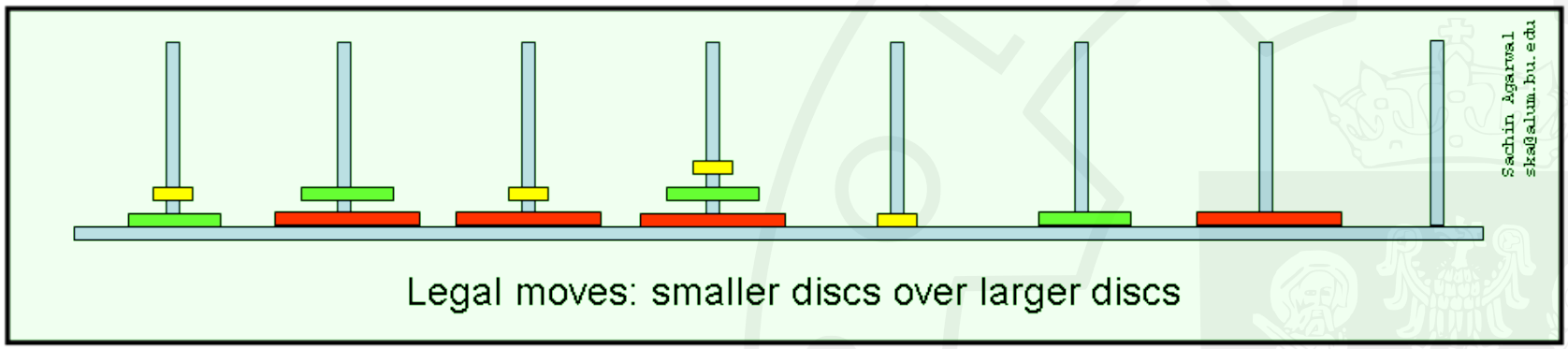
Rzeczy których nie lubie

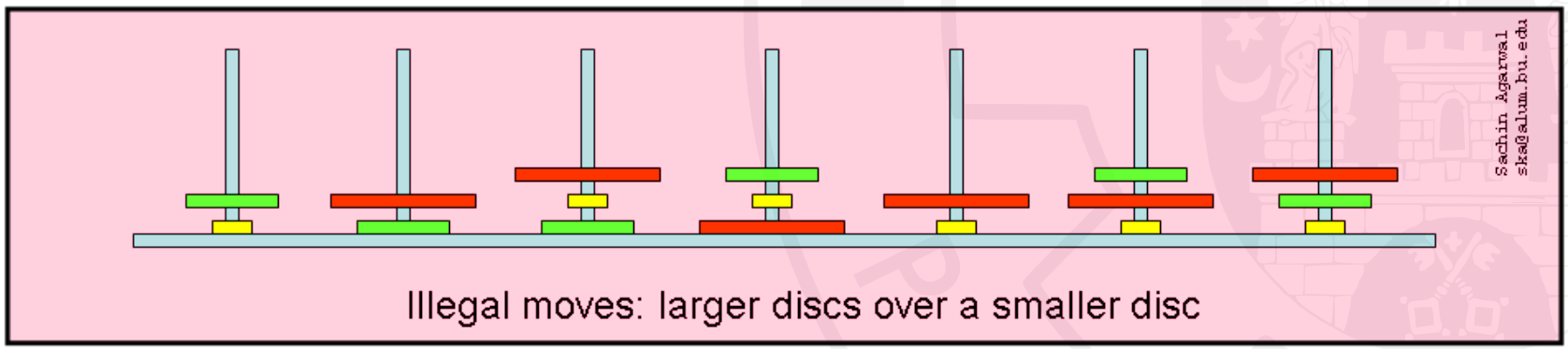
- Szukanie **Heisenbugów**
- Wrzucanie kodu **strukturalnego** do **funkcjonalnego**
- Rozwiązywanie problemów synchronizacji



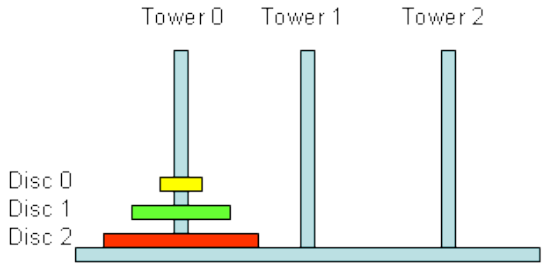
Rozwiązywanie problemów synchronizacji



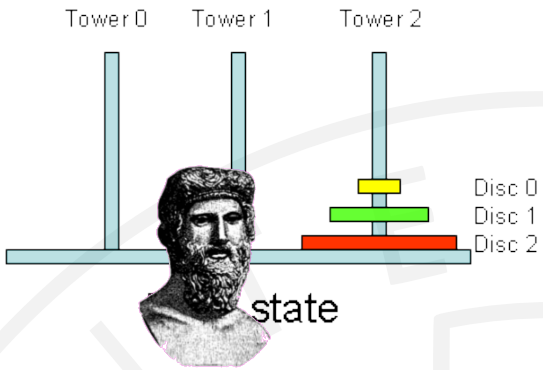




Rozwiązywanie problemów synchronizacji

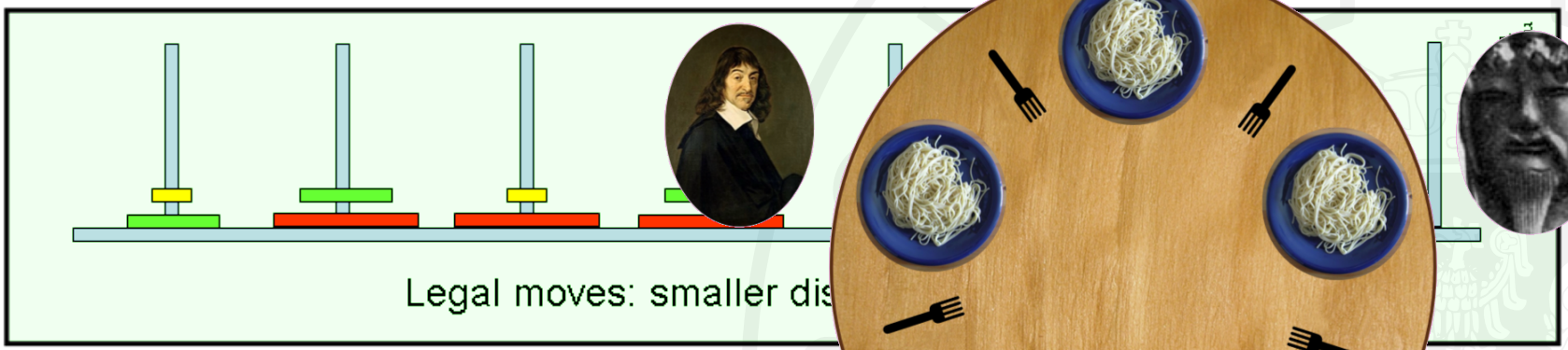


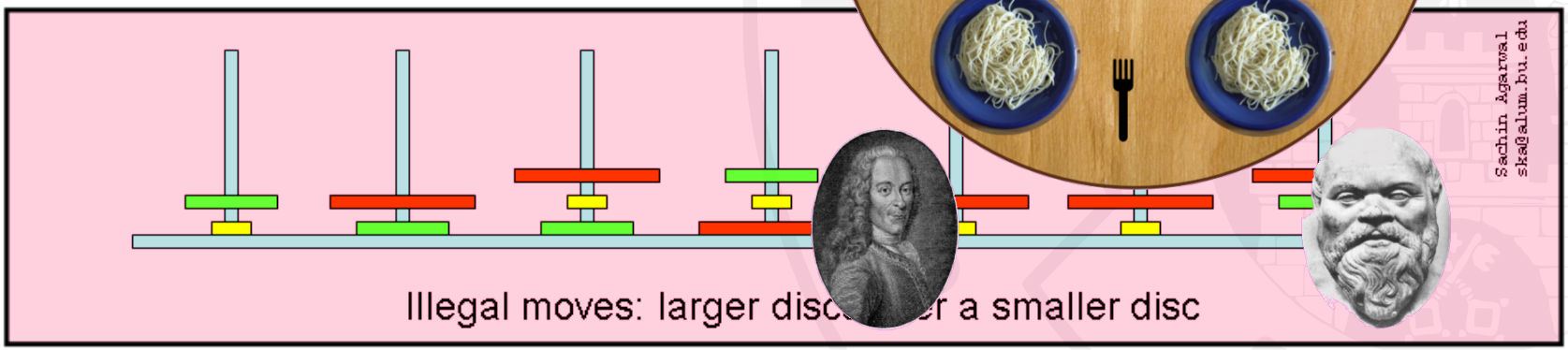
Initial state



state

Sachin Agarwal
ska@alum.bu.edu

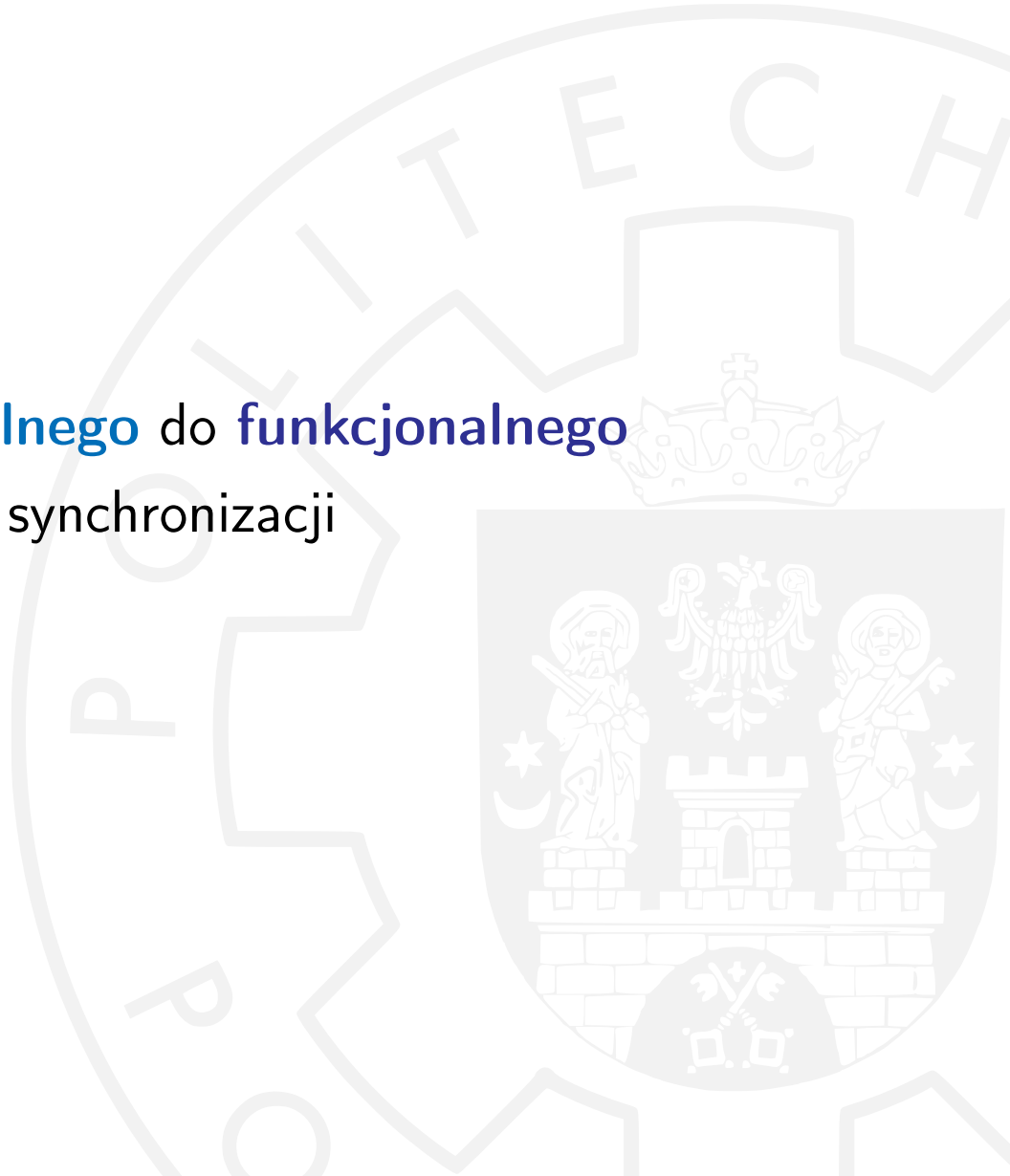




Sachin Agarwal
ska@alum.bu.edu

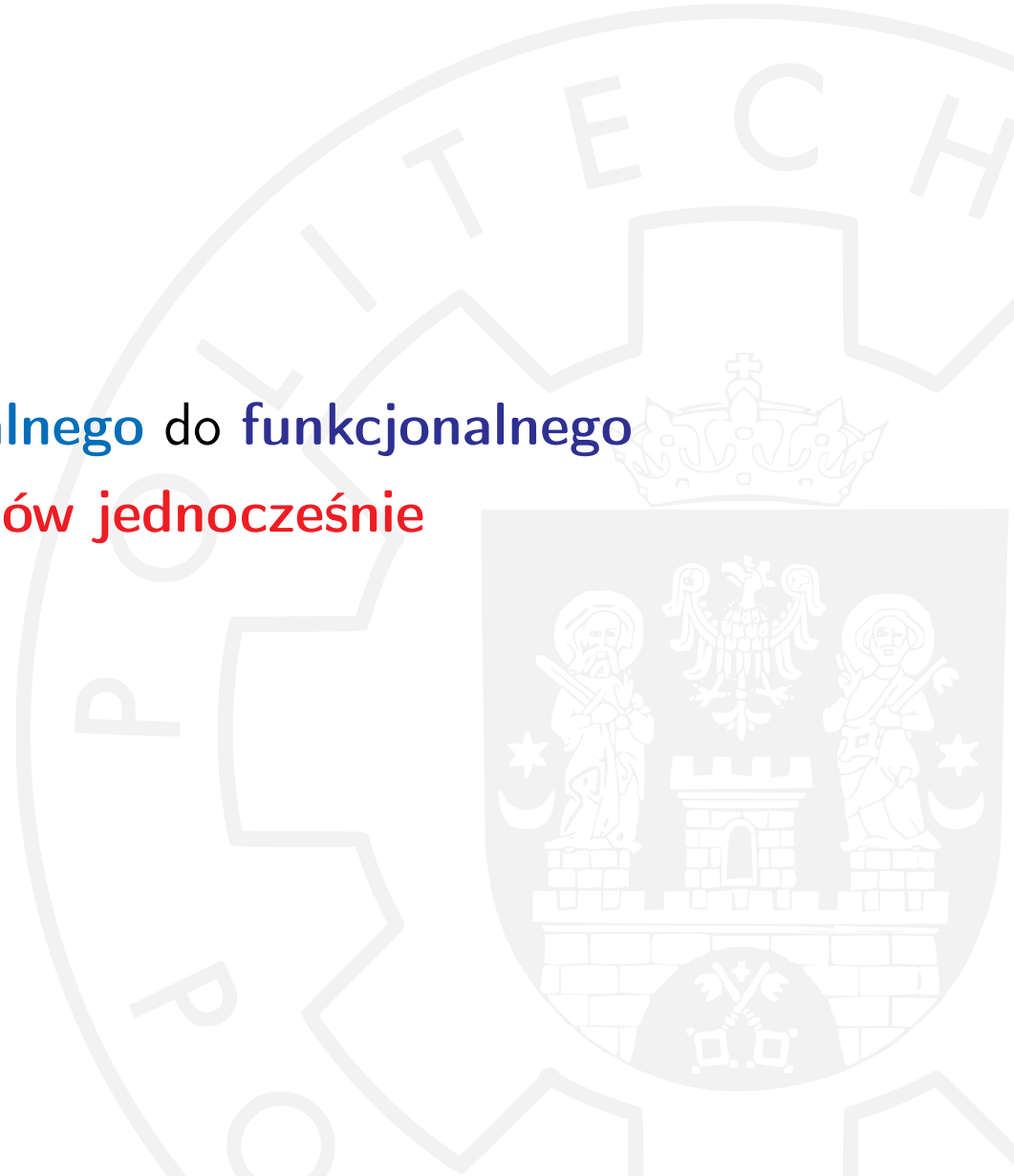
Rzeczy których nie lubie

- Szukanie **Heisenbugów**
- Wrzucanie kodu **strukturalnego** do **funkcjonalnego**
- Rozwiązywanie problemów synchronizacji



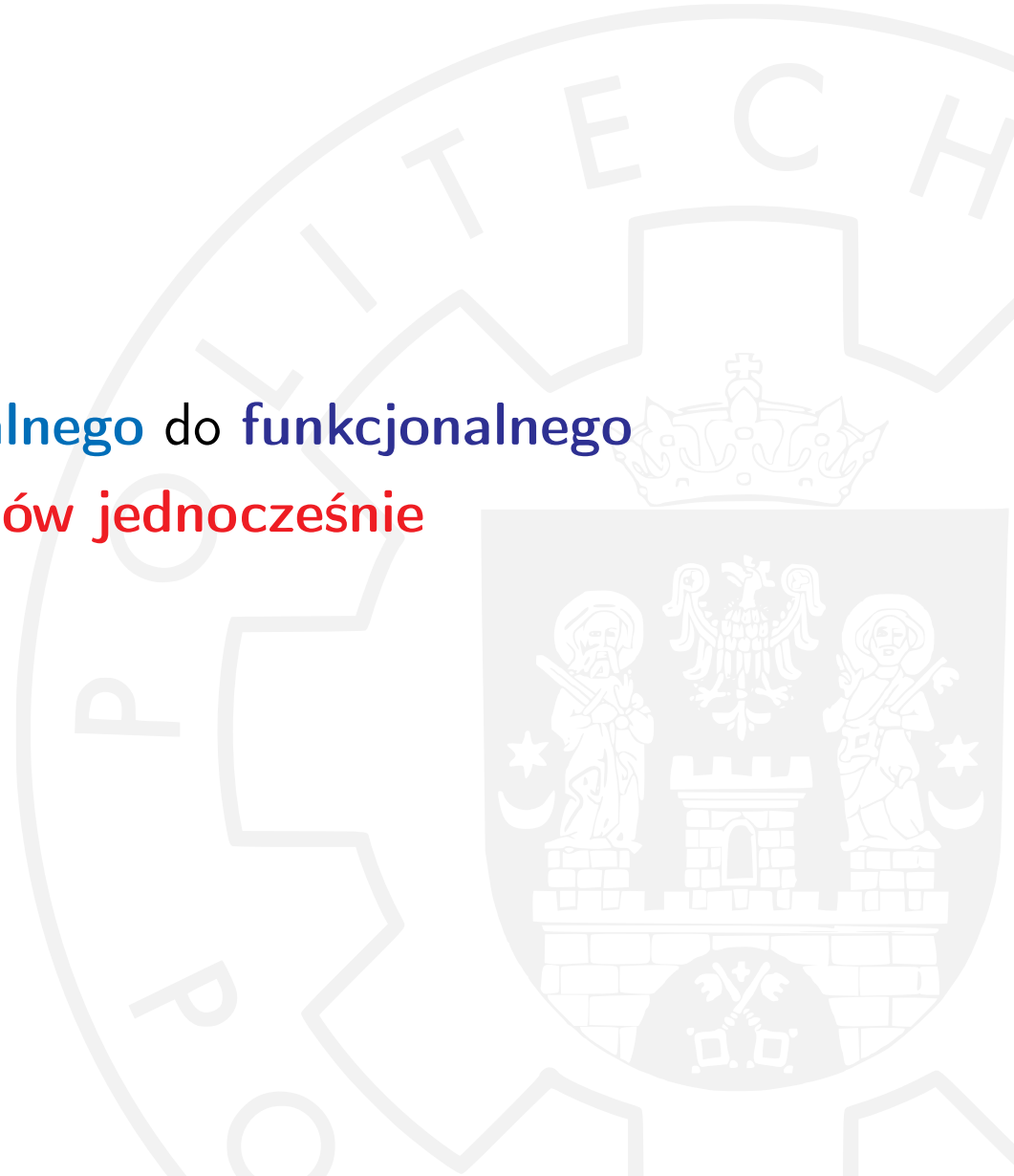
Rzeczy których nie lubie

- Szukanie **Heisenbugów**
- Wrzucanie kodu **strukturalnego** do **funkcjonalnego**
- Rozwiązywanie **2 problemów jednocześnie**



Rozwiązania których szukam

- Szukanie **Heisenbugów**
- Wrzucanie kodu **strukturalnego** do **funkcjonalnego**
- Rozwiązywanie **2 problemów jednocześnie**



Rozwiązania których szukam

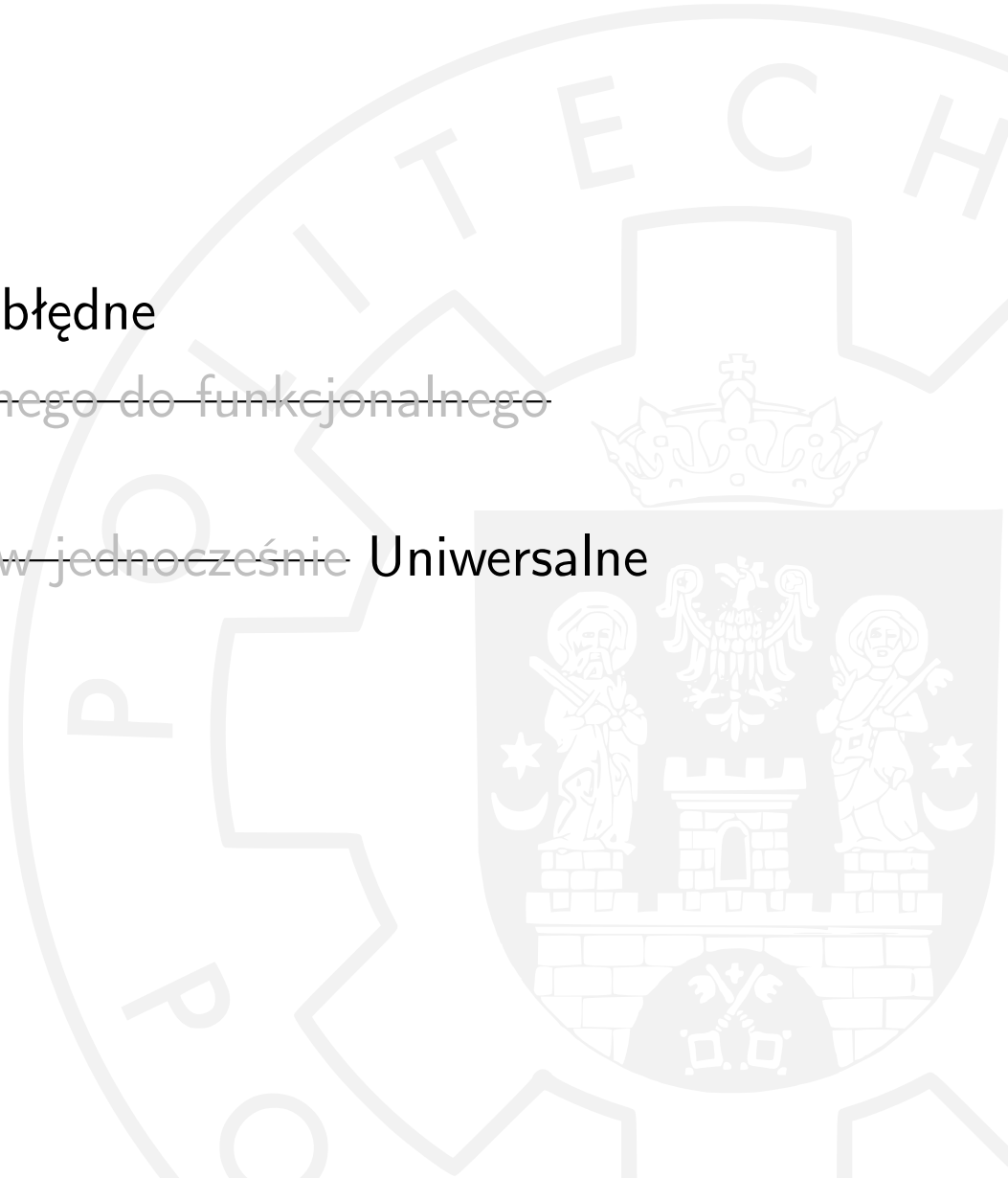
- Szukanie **Heisenbugów**
- Wrzucanie kodu **strukturalnego** do **funkcjonalnego**
- ~~Rozwiązywanie 2 problemów jednocześnie~~ Uniwersalne

Rozwiązania których szukam

- Szukanie **Heisenbugów**
- ~~Wrzucanie kodu strukturalnego do funkcjonalnego~~
  Proste w implementacji
- ~~Rozwiązywanie 2 problemów jednocześnie~~ Uniwersalne

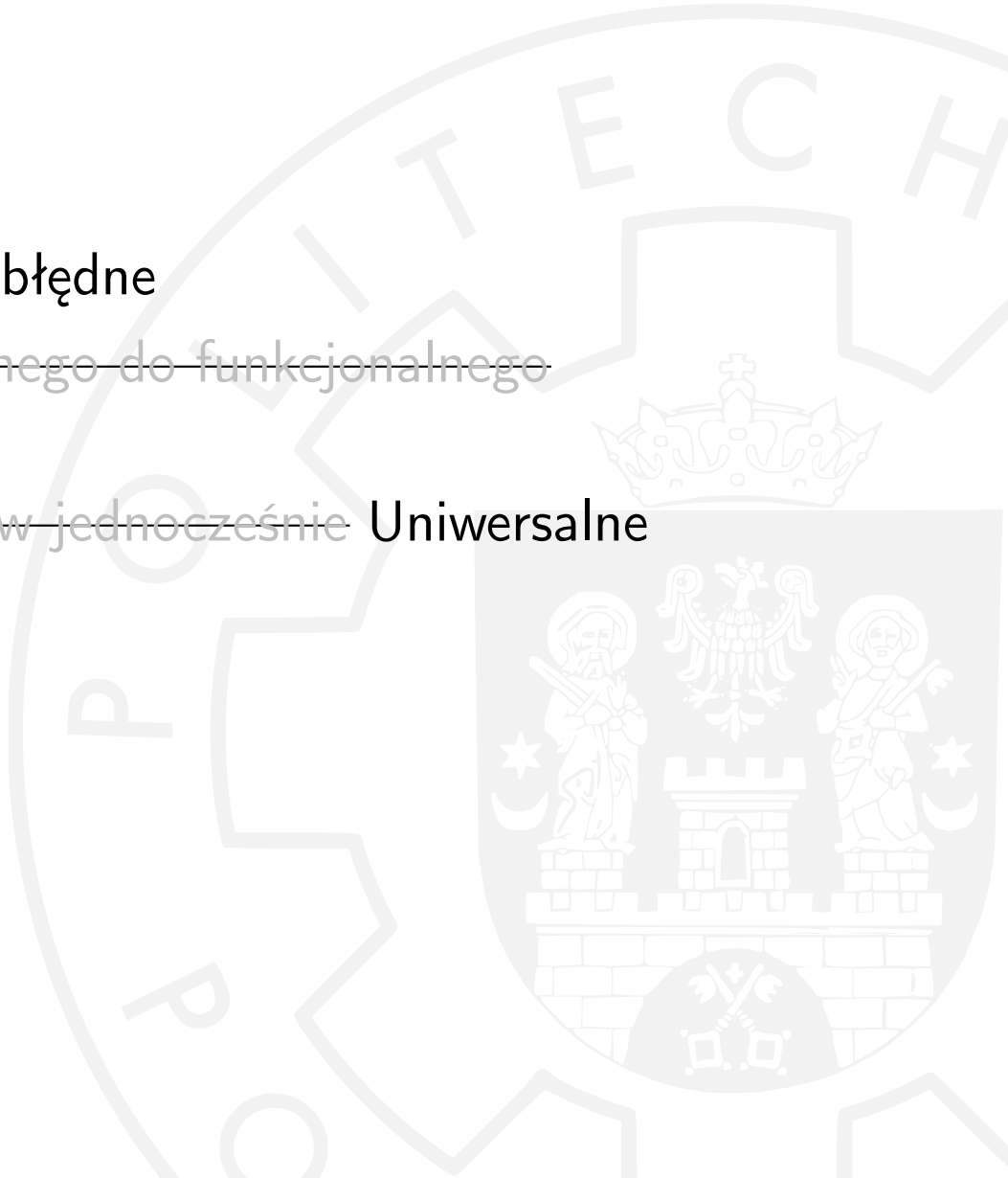
Rozwiązania których szukam

- ~~Szukanie Heisenbugów~~ Bezbłędne
- ~~Wrzucanie kodu strukturalnego do funkcjonalnego~~
  Proste w implementacji
- ~~Rozwiązywanie 2 problemów jednocześnie~~ Uniwersalne



Rozwiązania których szukam

- ~~Szukanie Heisenbugów~~ Bezbłędne
- ~~Wrzucanie kodu strukturalnego do funkcjonalnego~~
  Proste w implementacji
- ~~Rozwiązywanie 2 problemów jednocześnie~~ Uniwersalne
- Wydajne



Rozwiązania

Globalny zamek

```
from threading import Thread, Lock

_global_lock = Lock()
_shared_data = [0, 0, 0, 0]

class GLThread (Thread):
    def __init__(self, data, index):
        Thread.__init__(self)
        self.index = index
        self.data = data

    def run(self):
        with _global_lock:
            self.data[self.index] += 1
            my_data = self.data[self.index]
        print('set', self.index, 'to', my_data)

for i in range(0, 10):
    GLThread(_shared_data, i % len(_shared_data)).start()
```

Globalny zamek

```
from threading import Thread, Lock

_gl  ck = Lock()
_  data = [0  0, 0]

class LTh  (Thread):
    d  (self, dat  ):
        __init  elf)
        dex  dex
        .dat  dat

    (self)
    ith _gl  al_lo
        self.  a[  x]
        my  = s  a[self.index]
    print( et', self.index, 'to', my_data)

for i in range(0, 10):
    GLThread(_shared_data, i % len(_shared_data)).start()
```

Wątki funkcyjne

Duplikaty

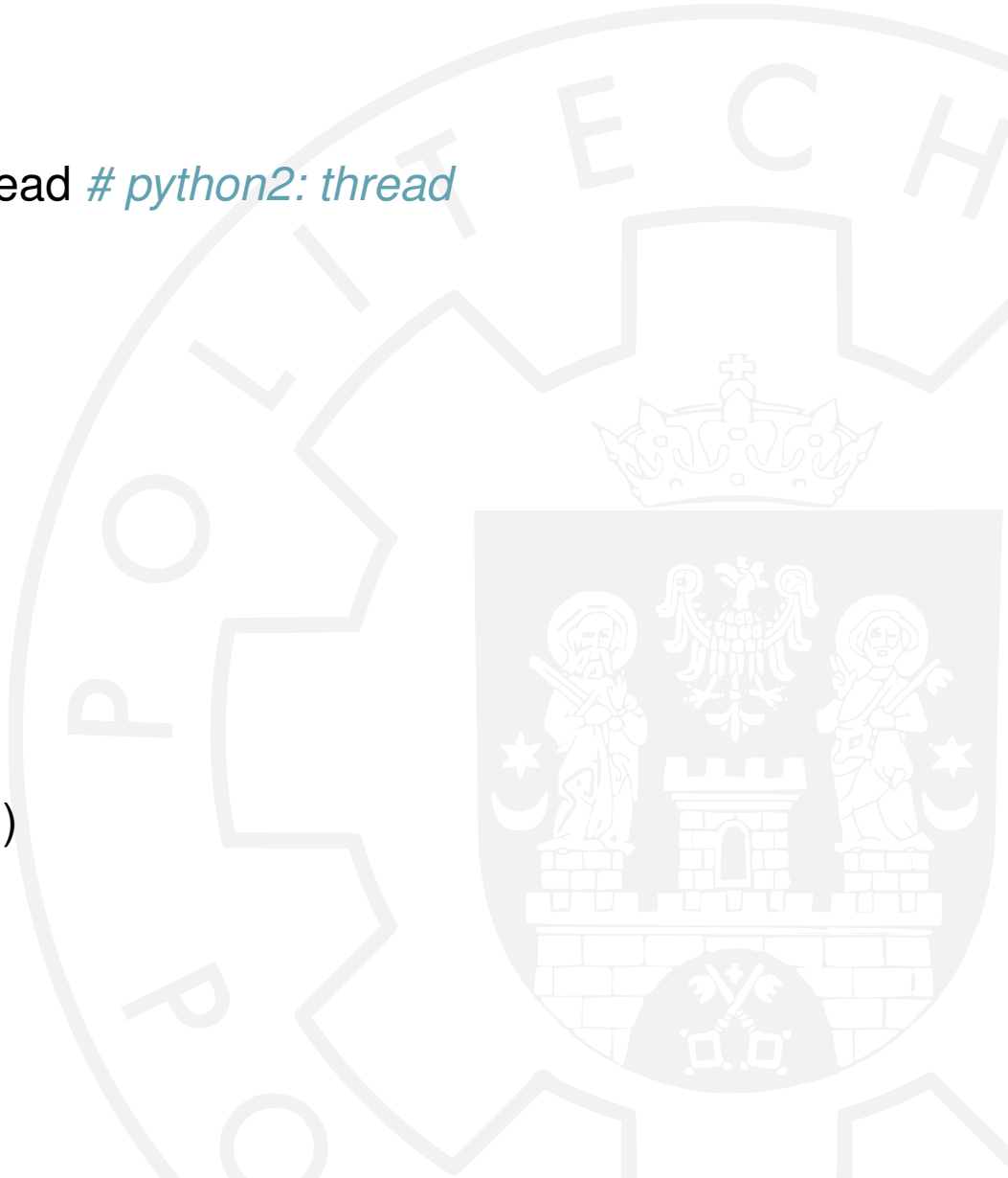

I'M NOT SAYING IT WAS ALIENS...
BUT IT WAS ALIENS
HD HISTORY
RELEASE DA KRACKON!
no
may be schizophrenic
okay, I'
...but at least I have each other.
THIS IS MAH JOB

Duplikaty

```
from hashlib import md5

hashes = [None] * len(files)

def make_hash(i):
    source = open(files[i], 'rb')
    data = source.read()
    source.close()
    hashes[i] = md5(data).digest()

for i in range(0, len(files)):
    make_hash(i)
```

Duplikaty

```
from hashlib import md5
from _thread import start_new_thread # python2: thread

hashes = [None] * len(files)

def make_hash(i):
    source = open(files[i], 'rb')
    data = source.read()
    source.close()
    hashes[i] = md5(data).digest()

for i in range(0, len(files)):
    start_new_thread(make_hash, (i,))
```



Duplikaty

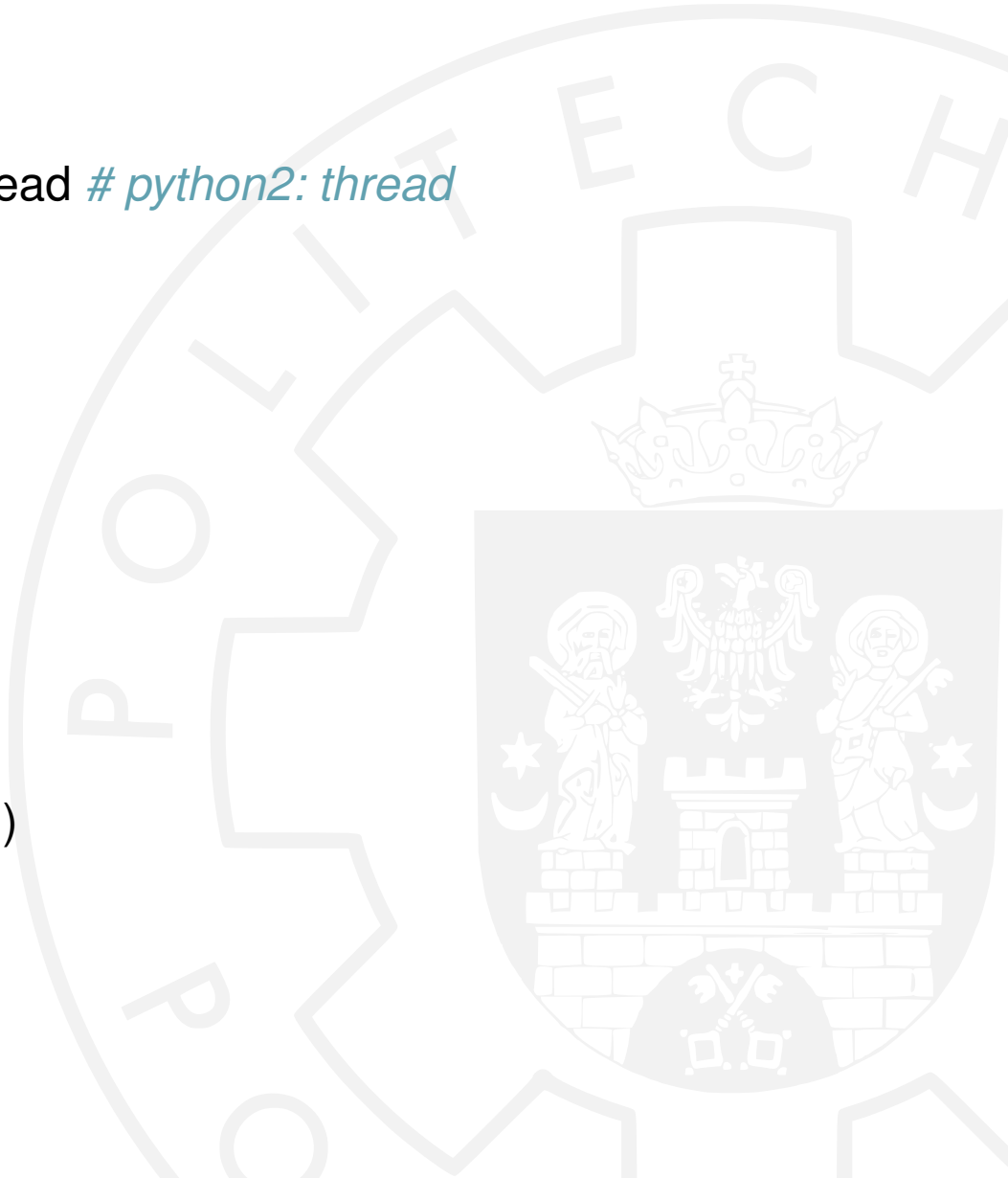

I'M NOT SAYING IT WAS ALIENS...
BUT IT WAS ALIENS
HD HISTORY
RELEASE DA KRACKON!
no
may be schizophrenic
okay, I'
...but at least I have each other.
THIS IS MAH JOB

Duplikaty


nom nom nom
hooo it are my birthday
Sum dayz
I'M NOT SAYING IT WAS ALIENS...
BUT IT WAS ALIENS
HISTORY
New Monorail Kitteh
Awaiting a
c dis? dis is teh motherboard
and dis.... r u listening to meh?
okay, I
...but at least I have each other.
de Plume
I are serious cat.
This is serious thread.
SO MUCH PUN.COM
iz mah house!

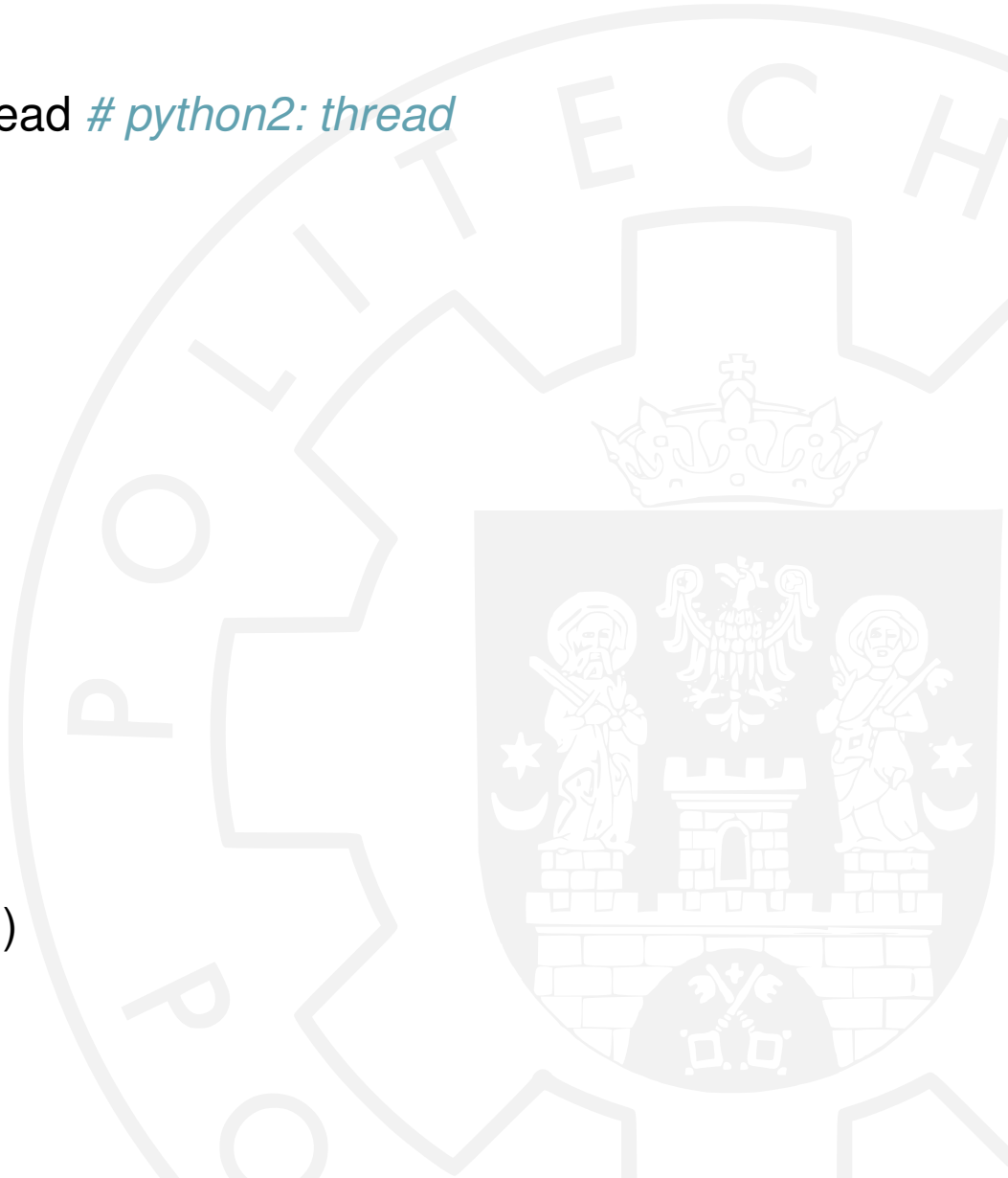
Duplikaty

```
from hashlib import md5
from _thread import start_new_thread # python2: thread

hashes = [None] * len(files)

def make_hash(i):
    source = open(files[i], 'rb')
    data = source.read()
    source.close()
    hashes[i] = md5(data).digest()

for i in range(0, len(files)):
    start_new_thread(make_hash, (i,))
```



Duplikaty

```
from hashlib import md5
from _thread import start_new_thread # python2: thread
from threading import Semaphore

hashes = [None] * len(files)
semaphore = Semaphore(4)

def make_hash(i):
    with semaphore:
        source = open(files[i], 'rb')
        data = source.read()
        source.close()
        hashes[i] = md5(data).digest()

for i in range(0, len(files)):
    start_new_thread(make_hash, (i,))
```



Duplikaty

```
from hashlib import md5
from _thread import start_new_thread # python2: thread
from threading import Semaphore, Barrier

hashes = [None] * len(files)
semaphore, barrier = Semaphore(4), Barrier(len(files) + 1)

def make_hash(i):
    with semaphore:
        source = open(files[i], 'rb')
        data = source.read()
        source.close()
        hashes[i] = md5(data).digest()
    barrier.wait()

for i in range(0, len(files)):
    start_new_thread(make_hash, (i,))

barrier.wait()
```

Wątki funkcyjne

- Preferują solipsyzm

Wątki funkcyjne

- Preferują solipsyzm
  - tylko odczyt
  - bez konfliktów

Wątki funkcyjne

- Preferują solipsyzm
  - tylko odczyt
  - bez konfliktów
- Niski koszt

Wątki funkcyjne

- Preferują solipsyzm
  - tylko odczyt
  - bez konfliktów
- Niski koszt
  - ten sam schemat
  - $<$10 lini kodu

Wątki funkcyjne

- Preferują solipsyzm
  - tylko odczyt
  - bez konfliktów
- Niski koszt
  - ten sam schemat
  - $<$10 lini kodu
- Znaczna poprawa efektywności

Wątki funkcyjne (Java)

```
final List<String> hashes = new ArrayList<String>();
for (int i = 0; i < files.length; i++) {
        hashes.add(null);
}
Collections.fill(hashes, null);
final Semaphore semaphore = new Semaphore(4);
final CyclicBarrier barrier = new CyclicBarrier(files.length + 1);

for (int i = 0; i < files.length; i++) {
        final int index = i;
        new Thread() {
                public void run() {
                        try {
                                semaphore.acquire();
                                hashes.set(index, createHash(files[index]));
                                semaphore.release();
                                barrier.await();
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
        }.start();
}

barrier.await();
```

```
private String createHash(String file) throws Exception {
        InputStream fis = new FileInputStream(file);
        MessageDigest d = MessageDigest.getInstance("MD5");
        byte[] buffer = new byte[1024];
        for (int n = 0; n != -1;) {
                n = fis.read(buffer);
                if (n > 0) {
                        d.update(buffer, 0, n);
                }
        }
        fis.close();
        return d.digest().toString();
}
```

Wątki funkcyjne (C♯)

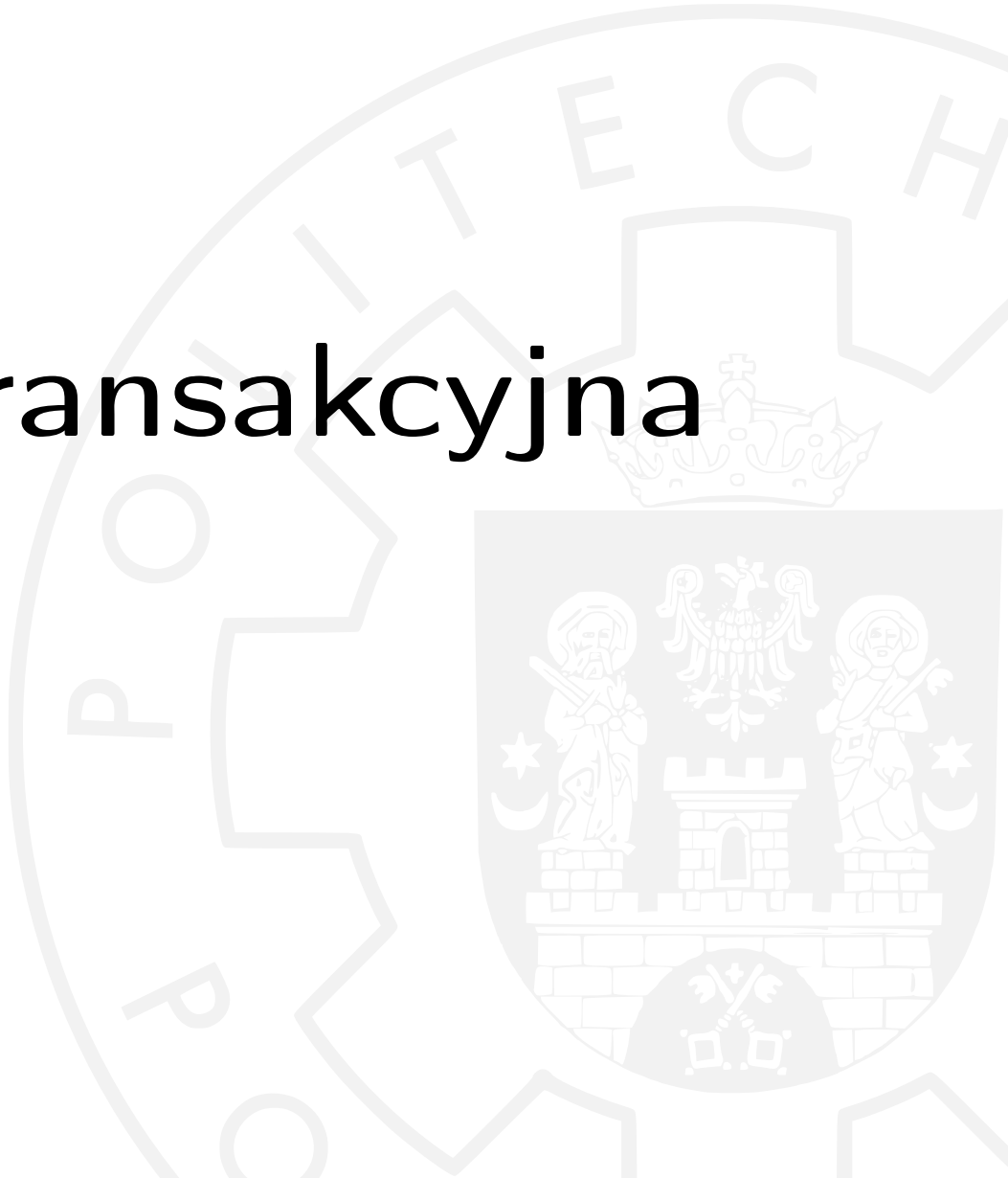
```
static void CreateHash(object arg)
{
    sem.WaitOne();

    // ...

    sem.Release();
}

static void Main(string[] args)
{
    foreach (var file in files)
    {
        var thread = new Thread(CreateHash);
        threads.Add(thread);
        thread.Start(file);
    }

    foreach (var thread in threads)
    {
        thread.Join();
    }
}
```

Wątki funkcyjne (C♯)

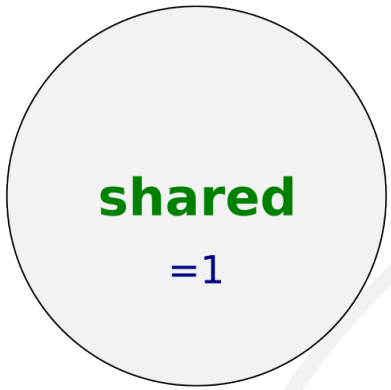
```
static void CreateHash(object arg)
{
    sem.WaitOne();

    // ...

    sem.Release();
}

static void Main(string[] args)
{
    foreach (var file in files)
    {
        var thread = new Thread(CreateHash);
        threads.Add(thread);
        thread.Start(file);
    }

    foreach (var thread in threads)
    {
        thread.Join();
    }
}
```

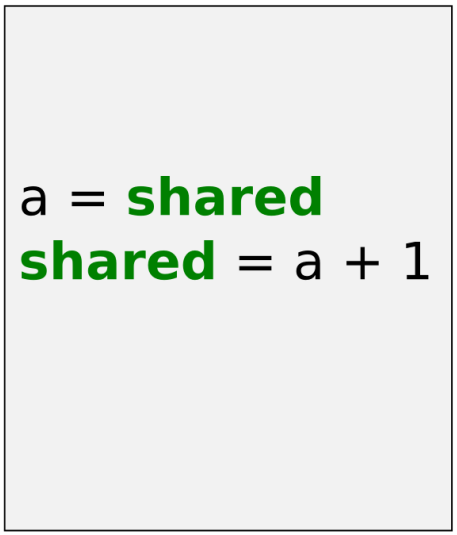
```
static void CreateHash(object arg)
{
    // ...
}

static void Main(string[] args)
{
    var p = new ParallelOptions {
        MaxDegreeOfParallelism = 12;
    };

    Parallel.ForEach(files, p, CreateHash);
}
```

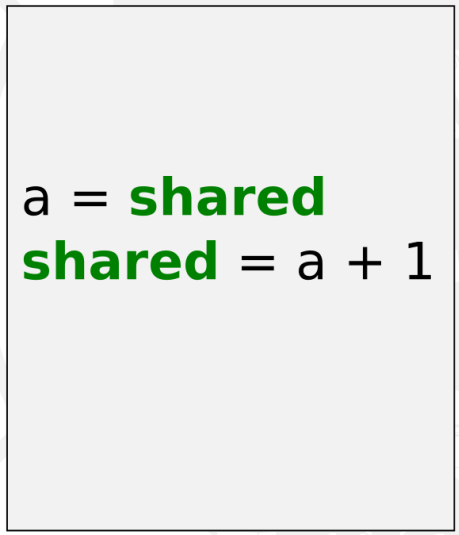
Pamięć transakcyjna



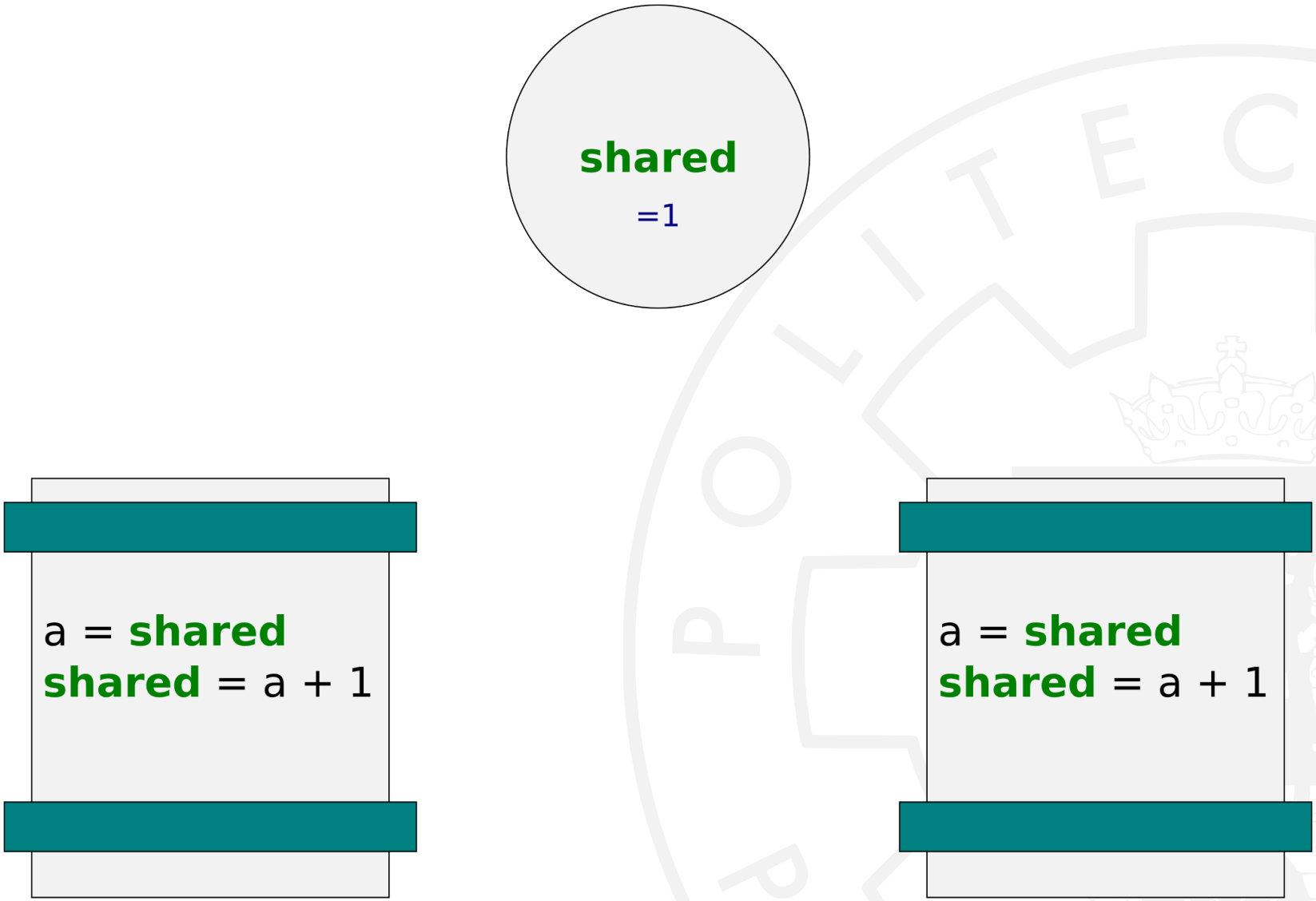
Pamięć transakcyjna



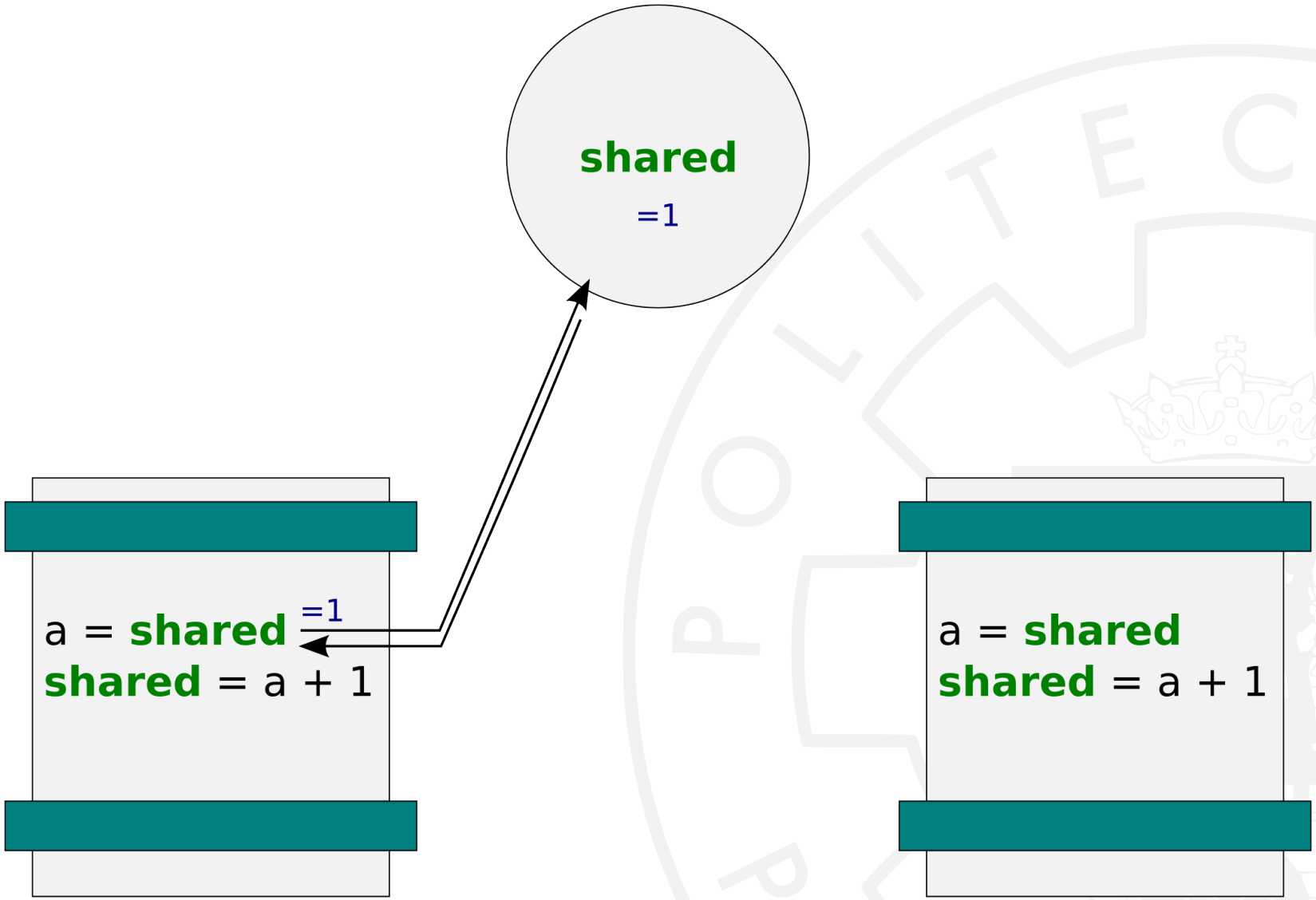




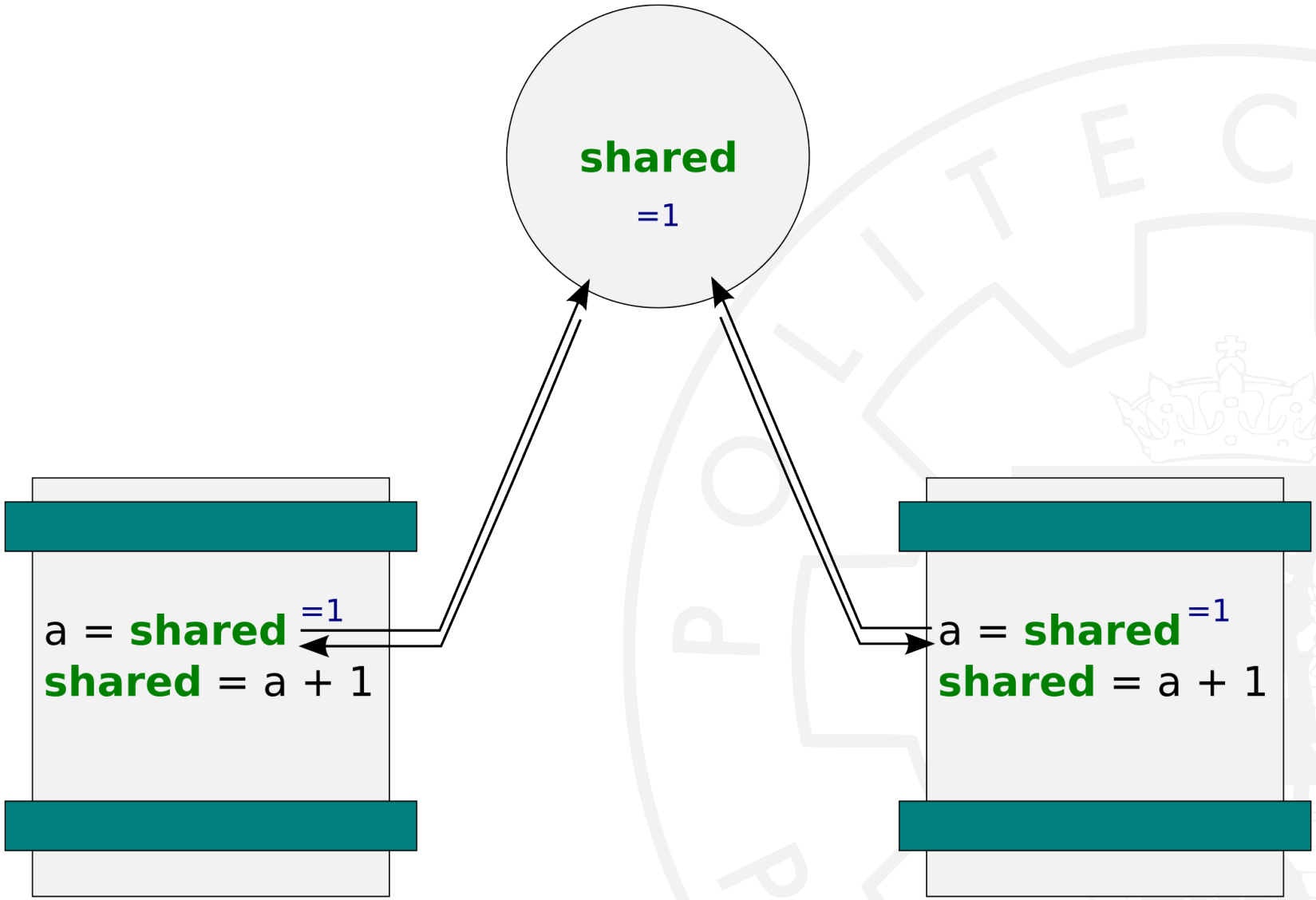
Pamięć transakcyjna



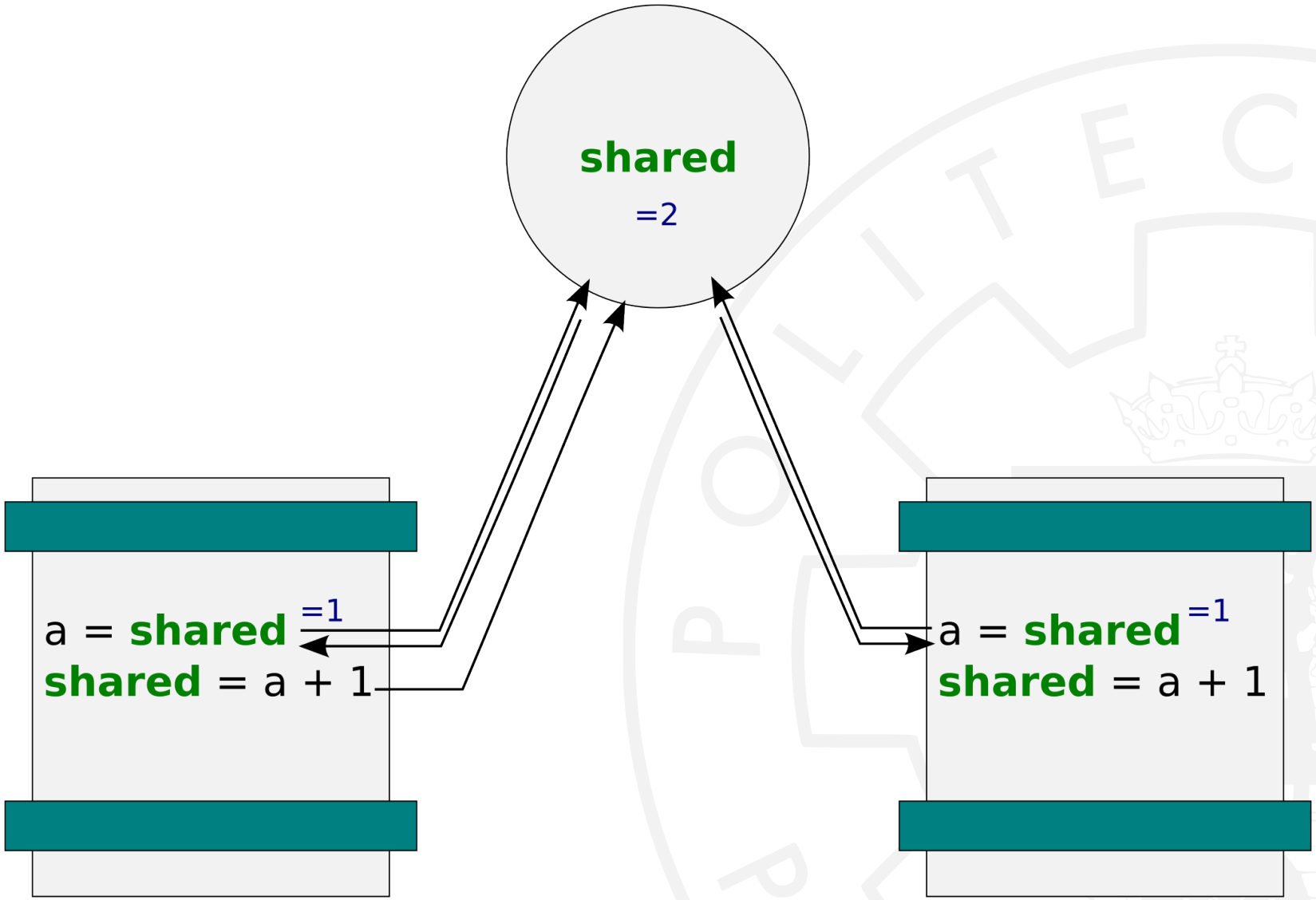
Pamięć transakcyjna



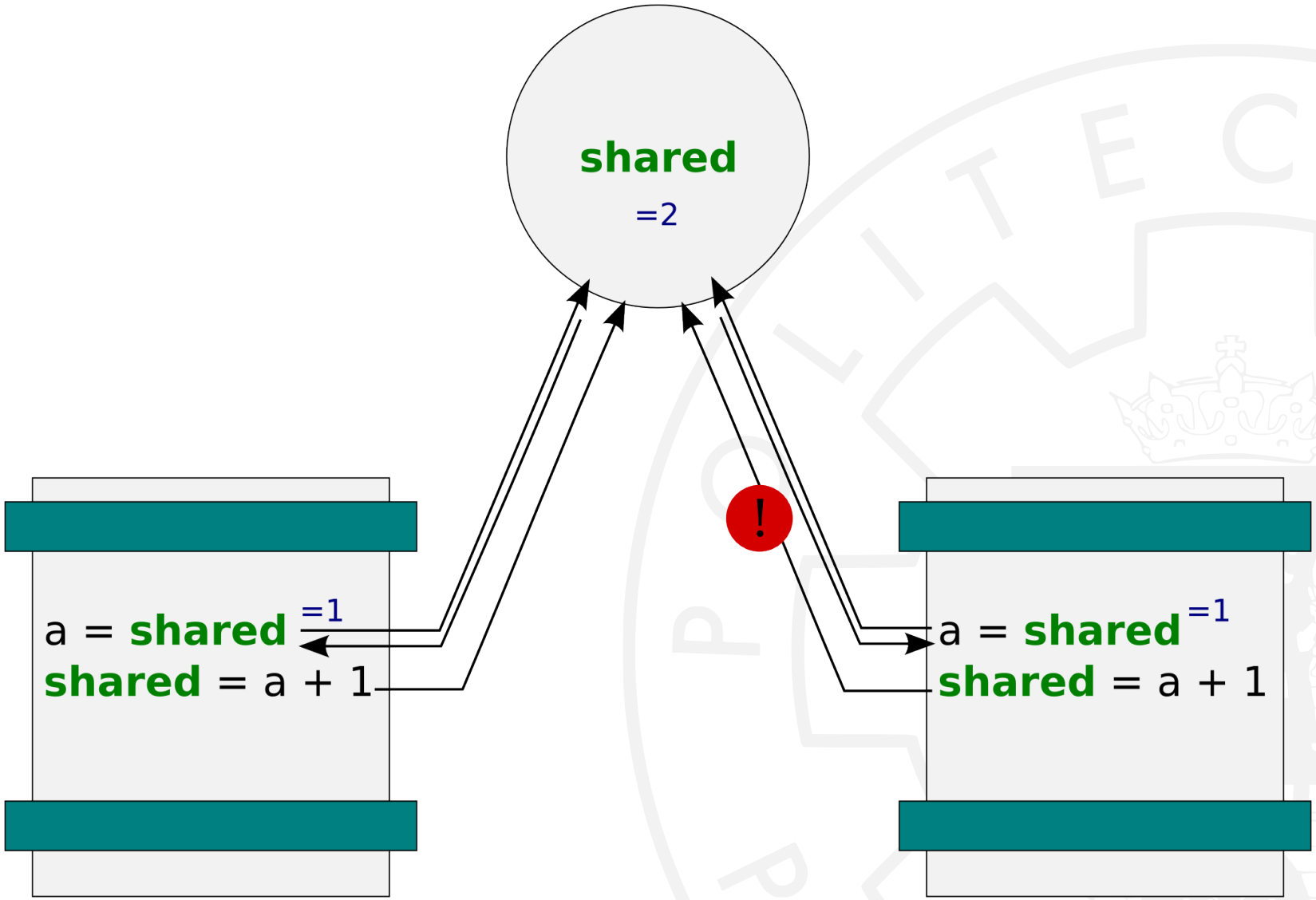
Pamięć transakcyjna



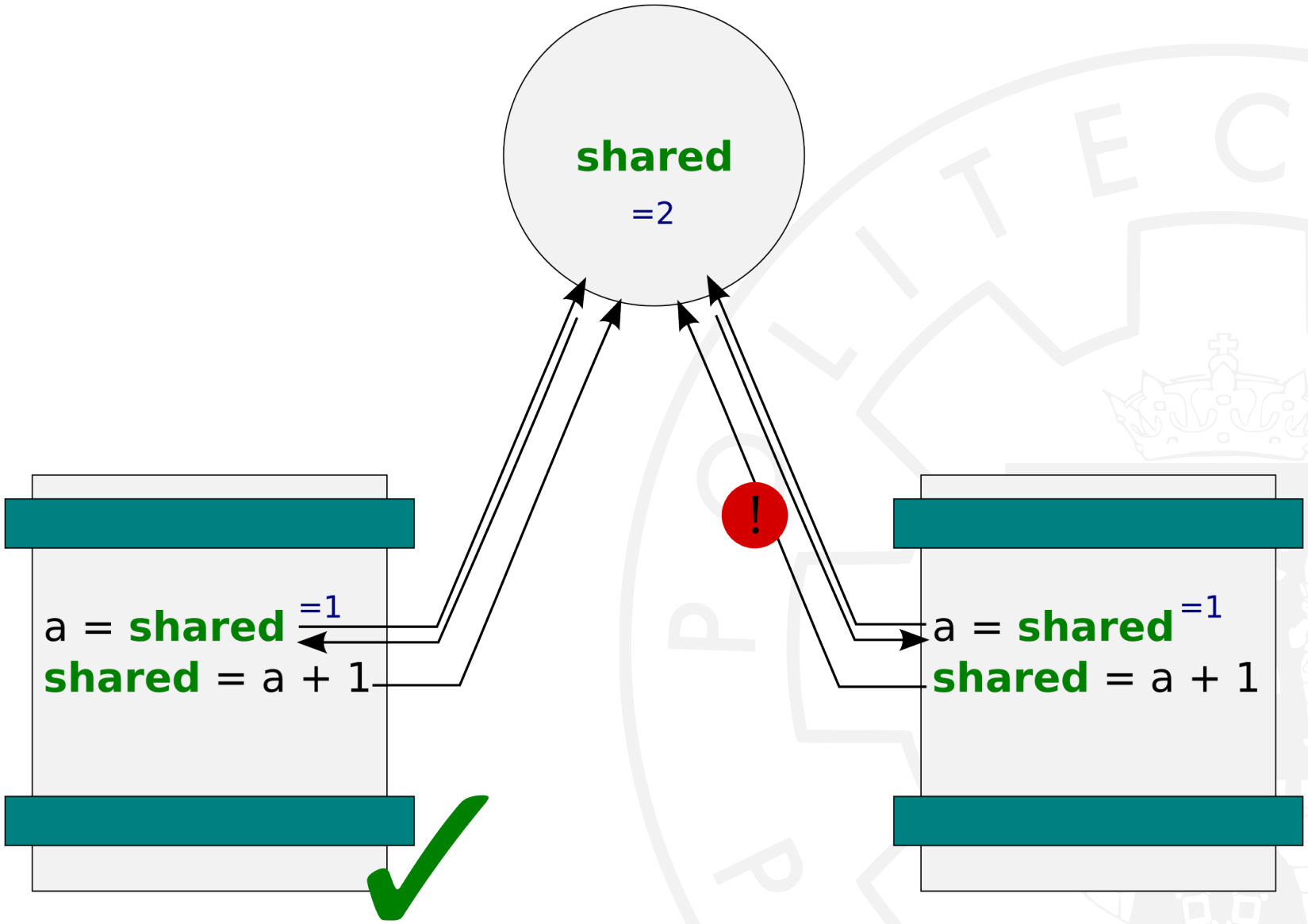
Pamięć transakcyjna



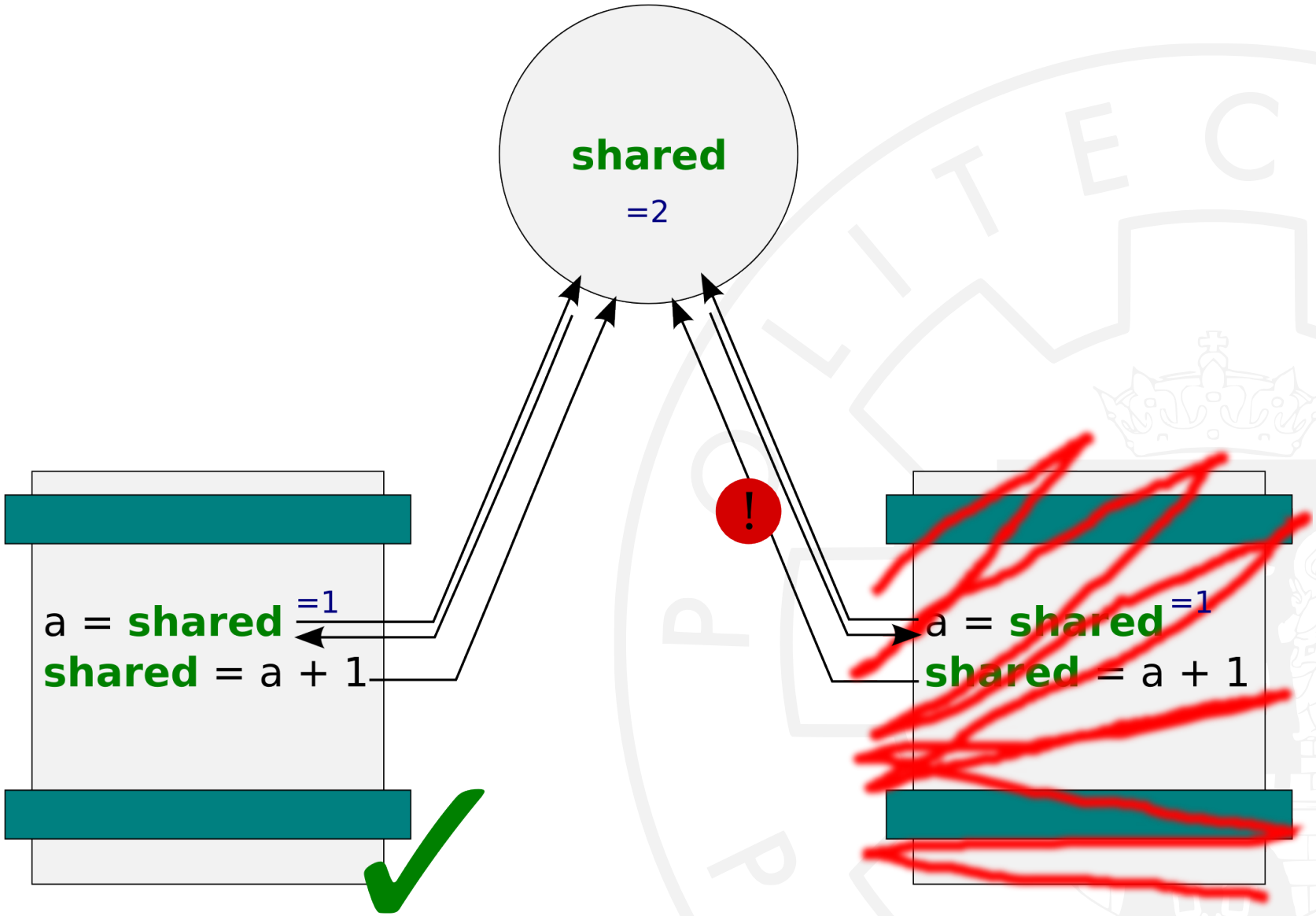
Pamięć transakcyjna



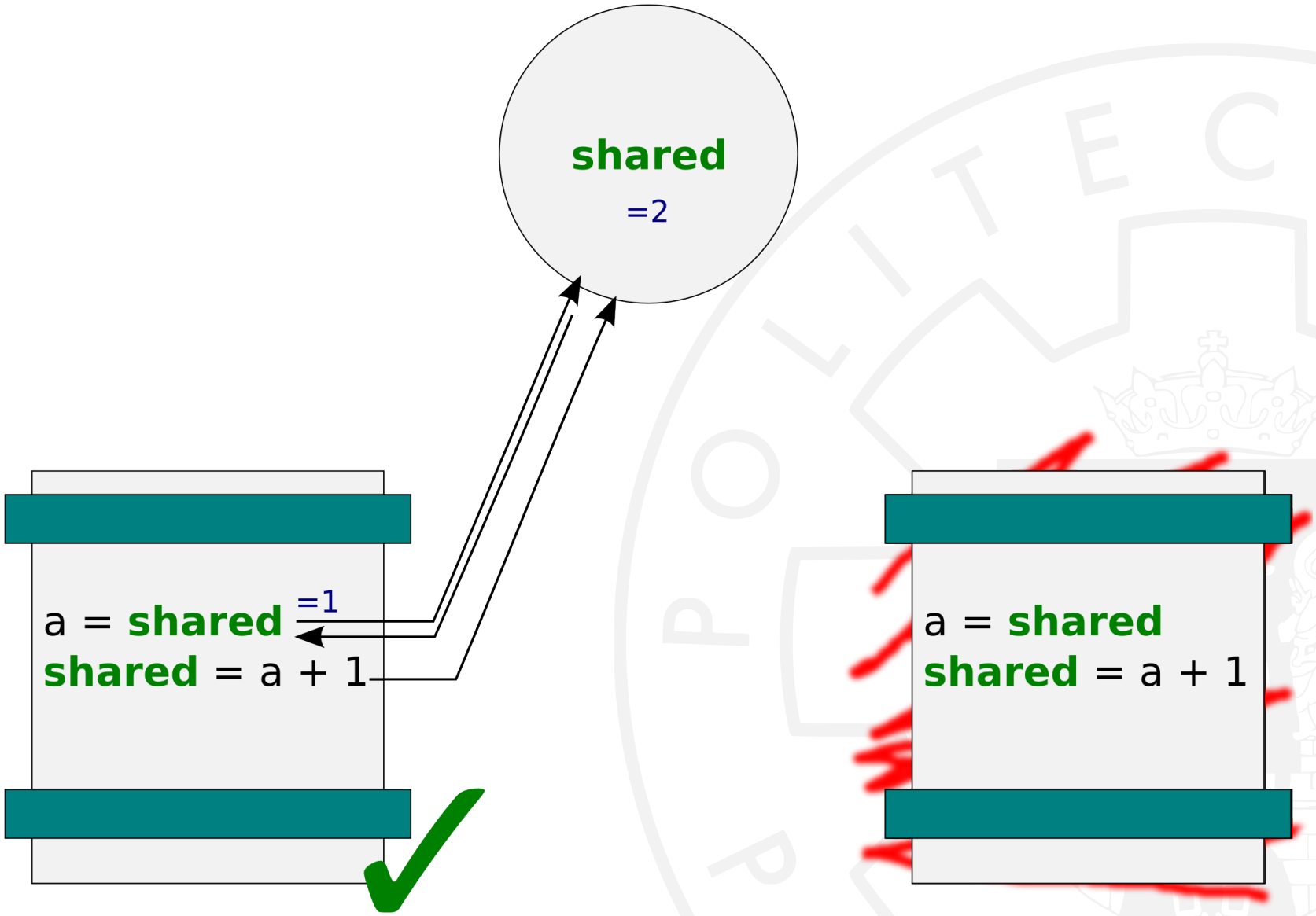
Pamięć transakcyjna



Pamięć transakcyjna



Pamięć transakcyjna



Pamięć transakcyjna

- **Python** – Kamaelia/Axon (`www.kamaelia.org`)
- **Java** – Deuce STM (`www.deucestm.org`)
- **C♯** – SXM (`research.microsoft.com`)
- **C/C++** – wbudowana w GCC (`gcc.gnu.org/wiki/TransactionalMemory`)

Bank

Bank

ACCOUNTS = 100
TRANSFERS = 10
accounts = {}

Bank

```
ACCOUNTS = 100
TRANSFERS = 10
accounts = {}

def total(ids):
    total = sum(accounts.values())

def transfer(a, b, sum):
    accounts[a] -= sum
    accounts[b] += sum
```

Bank

```
ACCOUNTS = 100
TRANSFERS = 10
accounts = {}

def init(ids, sum):
    for i in ids:
        accounts[i] = sum

def total(ids):
    total = sum(accounts.values())

def transfer(a, b, sum):
    accounts[a] -= sum
    accounts[b] += sum

ids = list(range(0, ACCOUNTS))
init(ids, 200)
```

Bank

```
from random import choice, randint

ACCOUNTS = 100
TRANSFERS = 10
accounts = {}

def init(ids, sum):
    for i in ids:
        accounts[i] = sum

def total(ids):
    total = sum(accounts.values())

def transfer(a, b, sum):
    accounts[a] -= sum
    accounts[b] += sum

ids = list(range(0, ACCOUNTS))
init(ids, 200)

for i in range(0, TRANSFERS):
    a, b = choice(ids), choice(ids)
    s = randint(5, 20)
    transfer(a, b, s)

total(ids)
```

Bank

```
from random import choice, randint
from Axon.STM import Store # Kamaelia (Axon 1.7)

ACCOUNTS = 100
TRANSFERS = 10
accounts = Store()

def init(ids, sum):
    for i in ids:
        accounts[i] = sum

def total(ids):
    total = sum(accounts.values())

def transfer(a, b, sum):
    accounts[a] -= sum
    accounts[b] += sum

ids = list(range(0, ACCOUNTS))
init(ids, 200)

for i in range(0, TRANSFERS):
    a, b = choice(ids), choice(ids)
    s = randint(5, 20)
    transfer(a, b, s)

total(ids)
```

Bank

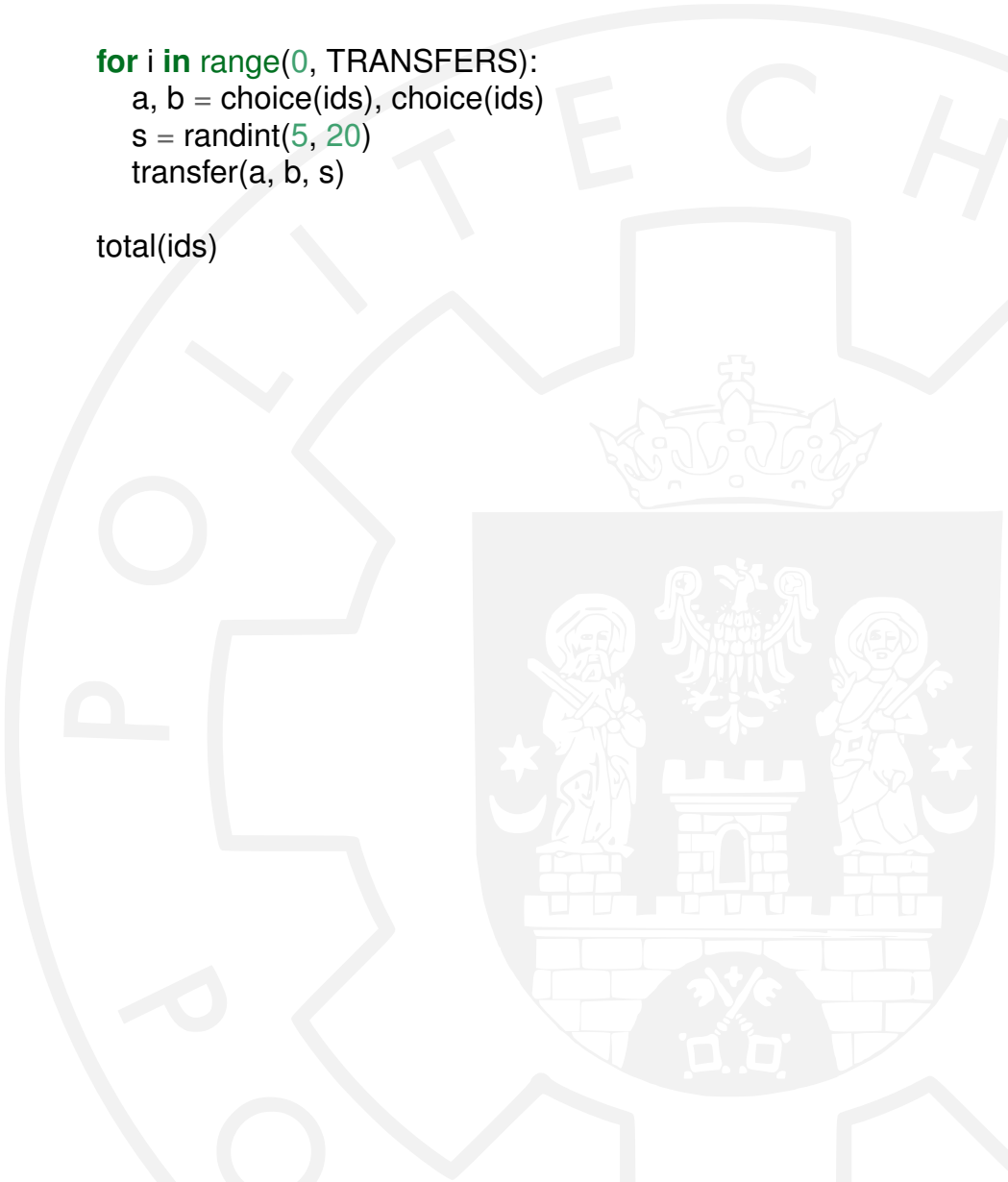
```
from random import choice, randint
from Axon.STM import Store # Kamaelia (Axon 1.7)

ACCOUNTS = 100
TRANSFERS = 10
accounts = Store()

def init(ids, sum):
    _accounts = accounts.using(*ids)
    for i in ids:
        _accounts[i].set(sum)
    _accounts.commit()

def total(ids):
    total = sum(accounts.values())

def transfer(a, b, sum):
    accounts[a] -= sum
    accounts[b] += sum

ids = list(range(0, ACCOUNTS))
init(ids, 200)

for i in range(0, TRANSFERS):
    a, b = choice(ids), choice(ids)
    s = random.randint(5, 20)
    transfer(a, b, s)

total(ids)
```

Bank

```
from random import choice, randint
from Axon.STM import Store # Kamaelia (Axon 1.7)

ACCOUNTS = 100
TRANSFERS = 10
accounts = Store()

def init(ids, sum):
    _accounts = accounts.using(*ids)
    for i in ids:
        _accounts[i].set(sum)
    _accounts.commit()

def total(ids):
    _acccounts = accounts.using(*ids)
    for i in ids:
        total += _accounts[i].value
    _accounts.commit()

def transfer(a, b, sum)
    accounts[a] -= sum
    accounts[b] += sum

ids = list(range(0, ACCOUNTS))
init(ids, 200)

for i in range(0, TRANSFERS):
    a, b = choice(ids), choice(ids)
    s = randint(5, 20)
    transfer(a, b, s)

total(ids)
```

Bank

```
from random import choice, randint
from Axon.STM import Store # Kamaelia (Axon 1.7)

ACCOUNTS = 100
TRANSFERS = 10
accounts = Store()

def init(ids, sum):
    _accounts = accounts.using(*ids)
    for i in ids:
        _accounts[i].set(sum)
    _accounts.commit()

def total(ids):
    _acccounts = accounts.using(*ids)
    for i in ids:
        total += _accounts[i].value
    _accounts.commit()

def transfer(a, b, sum)
    _accounts = accounts.using(a, b)
    _accounts[a].set(_accounts[a].value - sum)
    _accounts[b].set(_accounts[b].value + sum)
    _accounts.commit()

ids = list(range(0, ACCOUNTS))
init(ids, 200)

for i in range(0, TRANSFERS):
    a, b = choice(ids), choice(ids)
    s = randint(5, 20)
    transfer(a, b, s)

total(ids)
```



Bank

```
from random import choice, randint
from Axon.STM import Store # Kamaelia (Axon 1.7)

ACCOUNTS = 100
TRANSFERS = 10
accounts = Store()

def init(ids, sum):
    _accounts = accounts.using(*ids)
    for i in ids:
        _accounts[i].set(sum)
    _accounts.commit()

def total(ids):
    _acccounts = accounts.using(*ids)
    for i in ids:
        total += _accounts[i].value
    _accounts.commit()

def transfer(a, b, sum)
    _accounts = accounts.using(a, b)
    _accounts[a].set(_accounts[a].value - sum)
    _accounts[b].set(_accounts[b].value + sum)
    _accounts.commit()

ids = list(range(0, ACCOUNTS))
init(ids, 200)
```

```
from threading import Thread
from Axon.STM import BusyRetry, ConcurrentUpdate

class Transaction (Thread):
    def __init__(self, f, a):
        Thread.__init__(self)
        self._f = f
        self._a = a
    def run(self):
        while True:
            try:
                self._f(*self._a)
            except ConcurrentUpdate:
                continue
            except BusyRetry:
                continue
            break

for i in range(0, TRANSFERS):
    a, b = choice(ids), choice(ids)
    s = randint(5, 20)
    transfer(a, b, s)

total(ids)
```

Bank

```
from random import choice, randint
from Axon.STM import Store # Kamaelia (Axon 1.7)

ACCOUNTS = 100
TRANSFERS = 10
accounts = Store()

def init(ids, sum):
    _accounts = accounts.using(*ids)
    for i in ids:
        _accounts[i].set(sum)
    _accounts.commit()

def total(ids):
    _acccounts = accounts.using(*ids)
    for i in ids:
        total += _accounts[i].value
    _accounts.commit()

def transfer(a, b, sum)
    _accounts = accounts.using(a, b)
    _accounts[a].set(_accounts[a].value - sum)
    _accounts[b].set(_accounts[b].value + sum)
    _accounts.commit()

ids = list(range(0, ACCOUNTS))
init(ids, 200)
```

```
from threading import Thread
from Axon.STM import BusyRetry, ConcurrentUpdate

class Transaction (Thread):
    def __init__(self, f, a):
        Thread.__init__(self)
        self._f = f
        self._a = a
    def run(self):
        while True:
            try:
                self._f(*self._a)
            except ConcurrentUpdate:
                continue
            except BusyRetry:
                continue
            break

for i in range(0, TRANSFERS):
    a, b = choice(ids), choice(ids)
    s = randint(5, 20)
    t = Transaction(transfer, (a, b, s))
    t.start()

t = Transaction(total, (ids,))
t.start()
```

Bank

```
from random import choice, randint
from Axon.STM import Store # Kamaelia (Axon 1.7)

ACCOUNTS = 100
TRANSFERS = 10
accounts = Store()

def init(ids, sum):
    _accounts = accounts.using(*ids)
    for i in ids:
        _accounts[i].set(sum)
    _accounts.commit()

def total(ids):
    _acccounts = accounts.using(*ids)
    for i in ids:
        total += _accounts[i].value
    _accounts.commit()

def transfer(a, b, sum)
    _accounts = accounts.using(a, b)
    _accounts[a].set(_accounts[a].value - sum)
    _accounts[b].set(_accounts[b].value + sum)
    _accounts.commit()

ids = list(range(0, ACCOUNTS))
init(ids, 200)
```

```
from threading import Thread
from Axon.STM import BusyRetry, ConcurrentUpdate

class Transaction (Thread):
    def __init__(self, f, a):
        Thread.__init__(self)
        self._f = f
        self._a = a
    def run(self):
        while True:
            try:
                self._f(*self._a)
            except ConcurrentUpdate:
                continue
            except BusyRetry:
                continue
            break

transactions = []
for i in range(0, TRANSFERS):
    a, b = choice(ids), choice(ids)
    s = randint(5, 20)
    t = Transaction(transfer, (a, b, s))
    t.start()
    transactions.append(t)

t = Transaction(total, (ids,))
t.start()
transactions.append(t)
```

Bank

```
from random import choice, randint
from Axon.STM import Store # Kamaelia (Axon 1.7)

ACCOUNTS = 100
TRANSFERS = 10
accounts = Store()

def init(ids, sum):
    _accounts = accounts.using(*ids)
    for i in ids:
        _accounts[i].set(sum)
    _accounts.commit()

def total(ids):
    _acccounts = accounts.using(*ids)
    for i in ids:
        total += _accounts[i].value
    _accounts.commit()

def transfer(a, b, sum)
    _accounts = accounts.using(a, b)
    _accounts[a].set(_accounts[a].value - sum)
    _accounts[b].set(_accounts[b].value + sum)
    _accounts.commit()

ids = list(range(0, ACCOUNTS))
init(ids, 200)
```

```
from threading import Thread
from Axon.STM import BusyRetry, ConcurrentUpdate

class Transaction (Thread):
    def __init__(self, f, a):
        Thread.__init__(self)
        self._f = f
        self._a = a
    def run(self):
        while True:
            try:
                self._f(*self._a)
            except ConcurrentUpdate:
                continue
            except BusyRetry:
                continue
            break

transactions = []
for i in range(0, TRANSFERS):
    a, b = choice(ids), choice(ids)
    s = randint(5, 20)
    t = Transaction(transfer, (a, b, s))
    t.start()
    transactions.append(t)

t = Transaction(total, (ids,))
t.start()
transactions.append(t)

for t in transactions:
    t.join()
```

Bank

```
from random import choice, randint
from Axon.STM import Store # Kamaelia (Axon 1.7)

ACCOUNTS = 100
TRANSFERS = 10
accounts = Store()

def init(ids, sum):
    _accounts = accounts.using(*ids)
    [_accounts[i].set(sum) for i in ids]
    _accounts.commit()

def total(ids):
    _acccounts = accounts.using(*ids)
    total = sum([_accounts[i] for i in ids])
    _accounts.commit()

def transfer(a, b, sum):
    _accounts = accounts.using(a, b)
    _accounts[a].set(_accounts[a].value - sum)
    _accounts[b].set(_accounts[b].value + sum)
    _accounts.commit()

ids = list(range(0, ACCOUNTS))
init(ids, 200)
```

```
from threading import Thread
from Axon.STM import BusyRetry, ConcurrentUpdate

class Transaction (Thread):
    def __init__(self, f, a):
        Thread.__init__(self)
        self._f = f
        self._a = a
    def run(self):
        while True:
            try:
                self._f(*self._a)
            except ConcurrentUpdate:
                continue
            except BusyRetry:
                continue
            break

transactions = []
for i in range(0, TRANSFERS):
    a, b = choice(ids), choice(ids)
    s = randint(5, 20)
    t = Transaction(transfer, (a, b, s))
    t.start()
    transactions.append(t)

t = Transaction(total, (ids,))
t.start()
transactions.append(t)

[t.join() for t in transactions]
```

Pamięć transakcyjna

- Optymistyczna

Pamięć transakcyjna

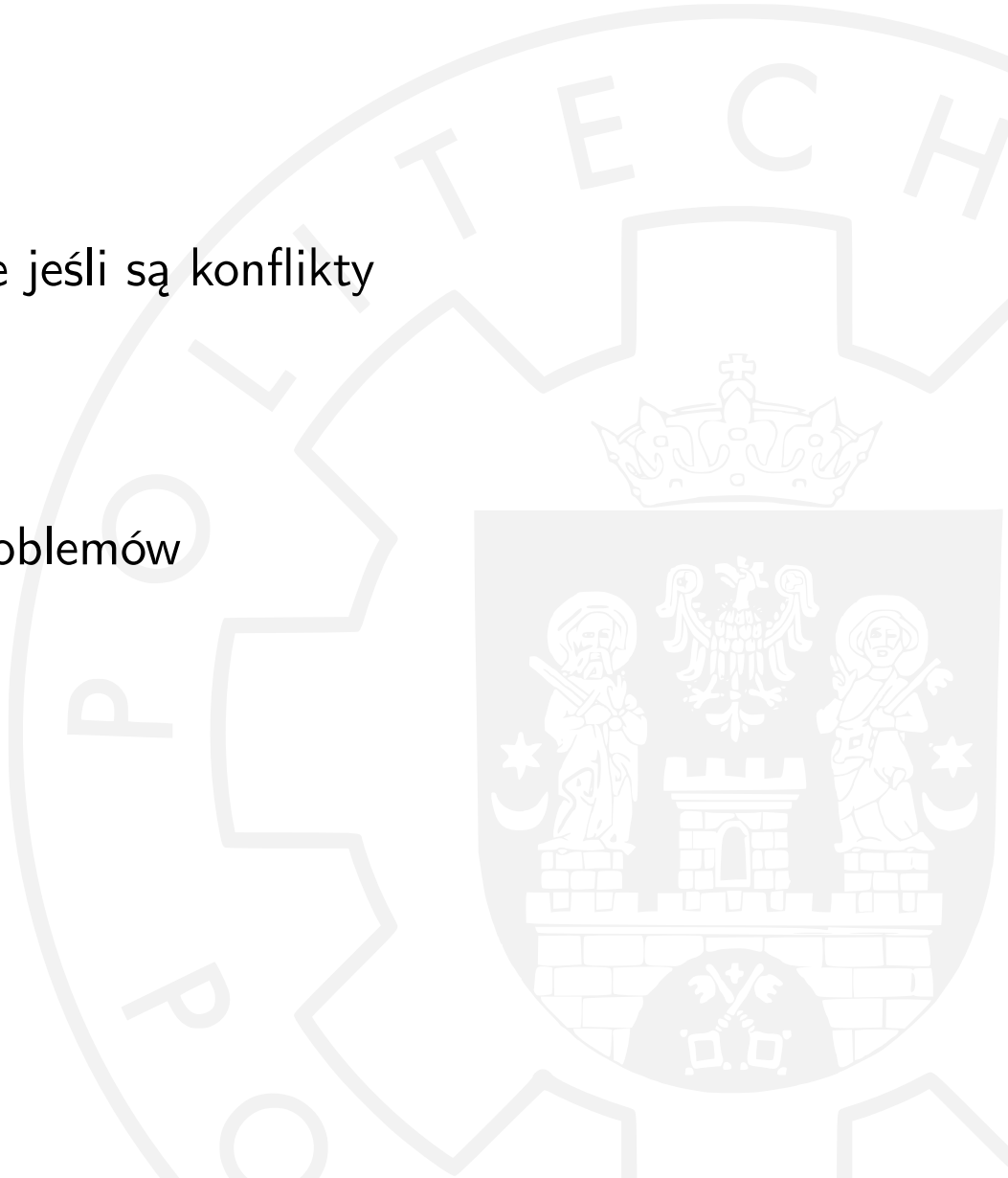
- Optymistyczna
  - czynności są powtarzane jeśli są konflikty
  - czuła na ilość transakcji

Pamięć transakcyjna

- Optymistyczna
  - czynności są powtarzane jeśli są konflikty
  - czuła na ilość transakcji
- Niski koszt

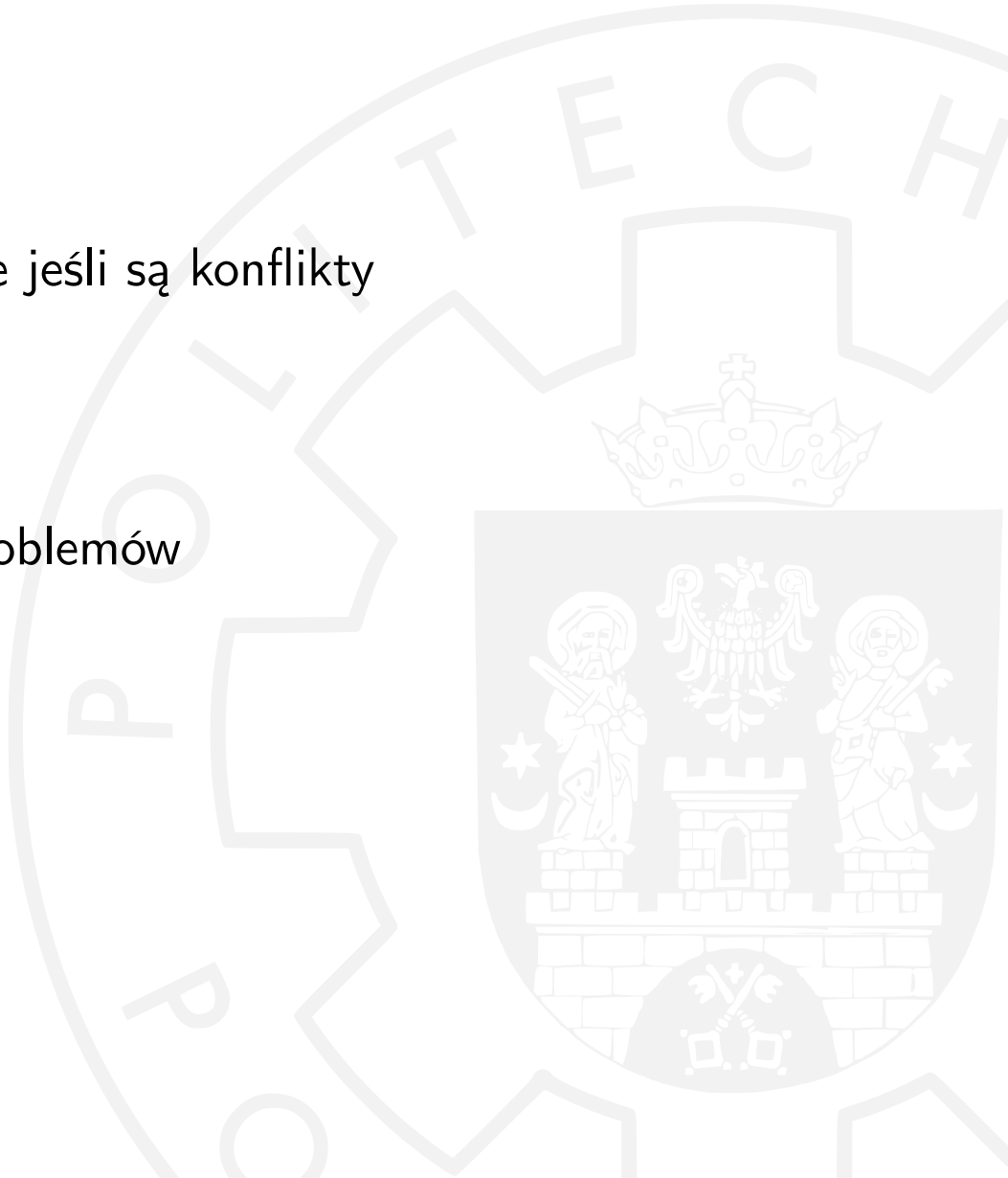
Pamięć transakcyjna

- Optymistyczna
  - czynności są powtarzane jeśli są konflikty
  - czuła na ilość transakcji
- Niski koszt
  - ten sam schemat
  - rozwiązuje większość problemów



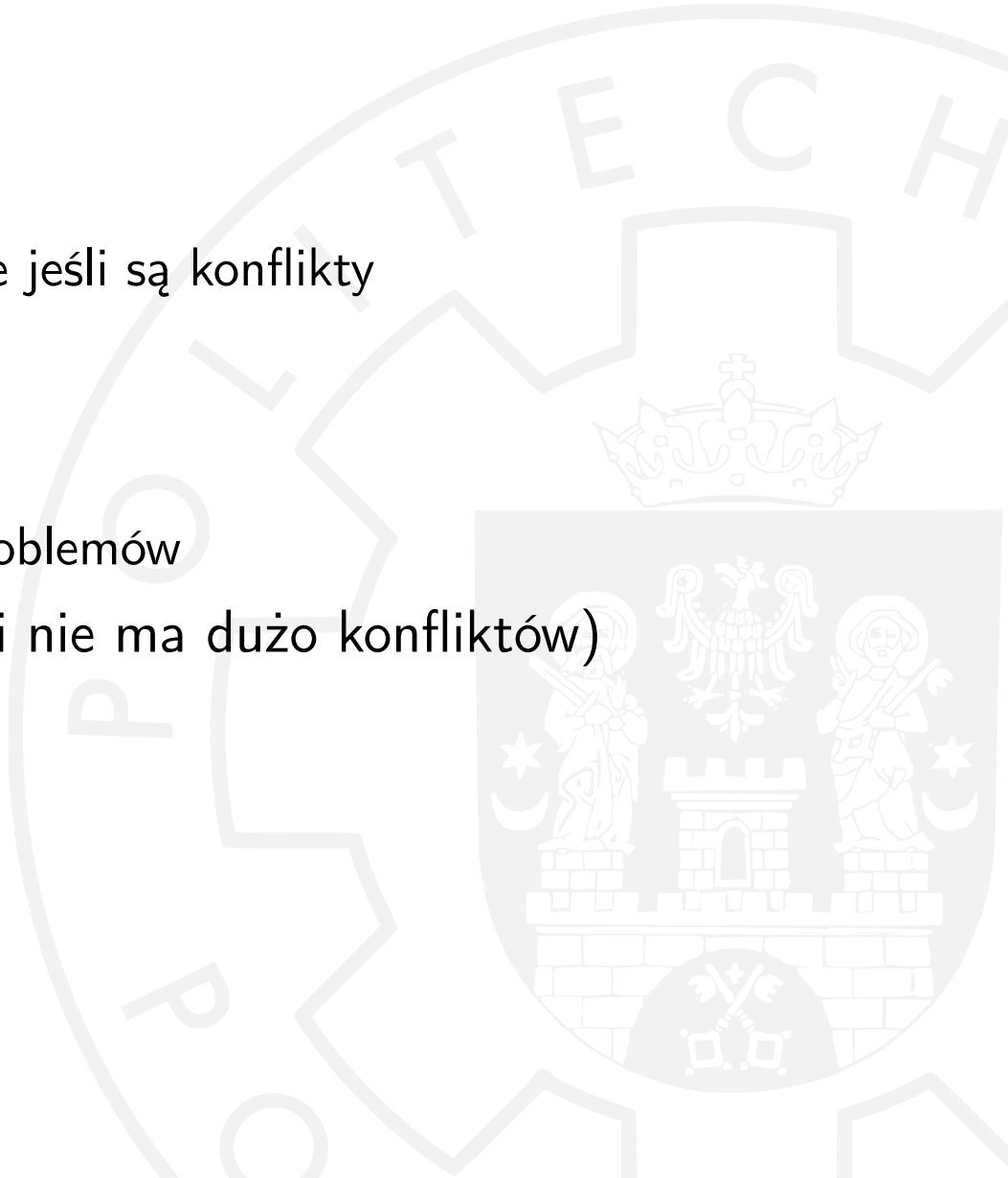
Pamięć transakcyjna

- Optymistyczna
  - czynności są powtarzane jeśli są konflikty
  - czuła na ilość transakcji
- Niski koszt
  - ten sam schemat
  - rozwiązuje większość problemów
- Poprawa efektywności



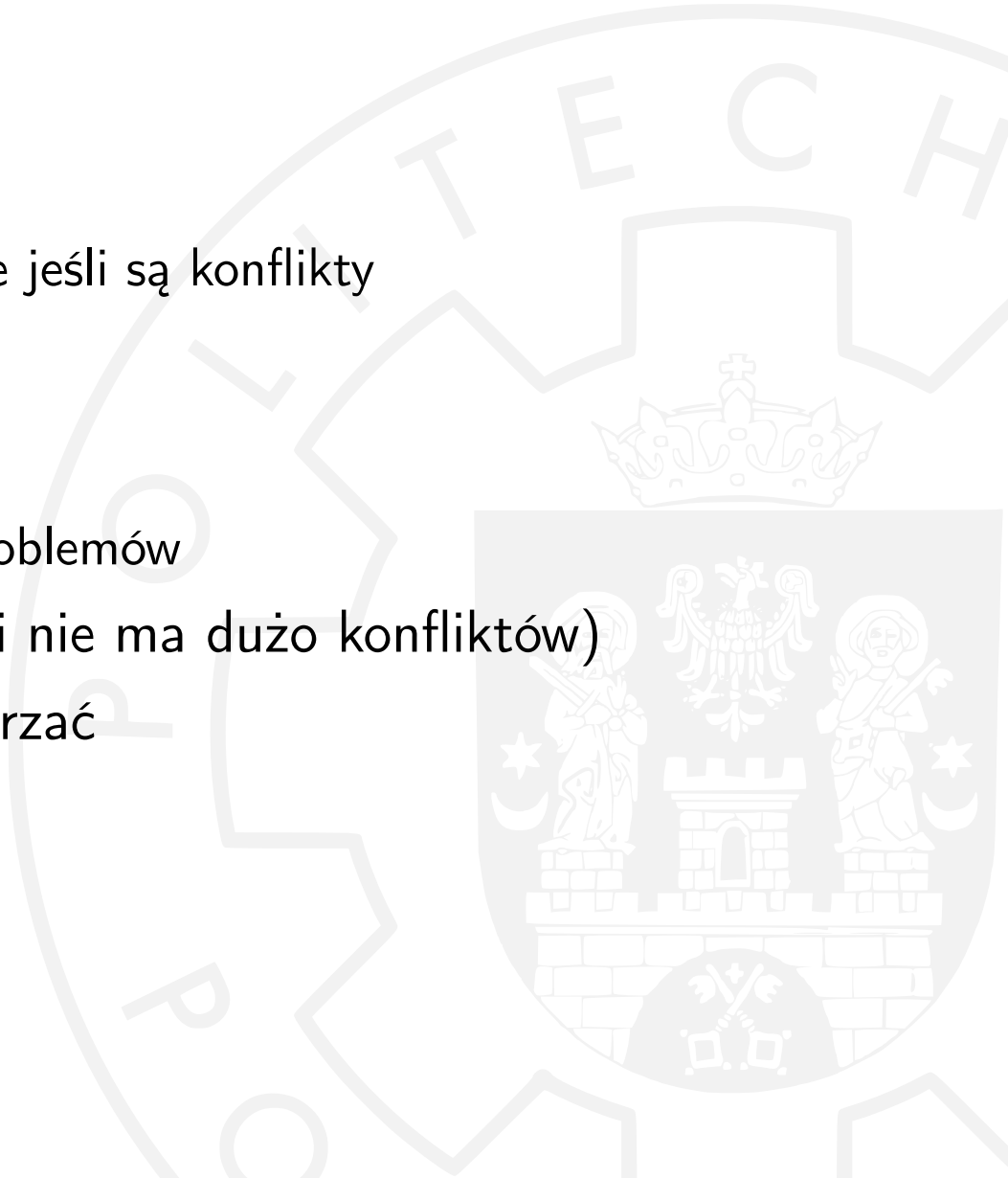
Pamięć transakcyjna

- Optymistyczna
  - czynności są powtarzane jeśli są konflikty
  - czuła na ilość transakcji
- Niski koszt
  - ten sam schemat
  - rozwiązuje większość problemów
- Poprawa efektywności (jeśli nie ma dużo konfliktów)



Pamięć transakcyjna

- Optymistyczna
  - czynności są powtarzane jeśli są konflikty
  - czuła na ilość transakcji
- Niski koszt
  - ten sam schemat
  - rozwiązuje większość problemów
- Poprawa efektywności (jeśli nie ma dużo konfliktów)
- Akcje muszą dać się powtarzać



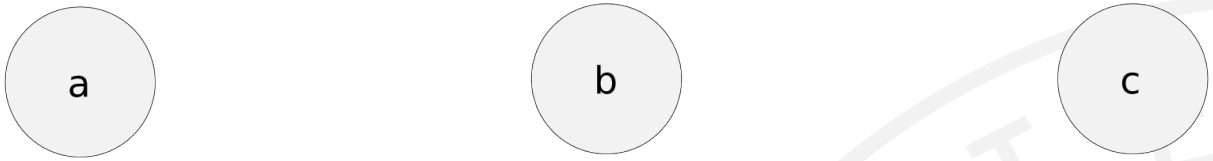
Bank

```
// Deuce STM
// compile with -javaagent:bin/deuceAgent.jar
public class Bank {
	public static final int TRANSFERS = 10;
	public static final int ACCOUNTS = 100;

	private final double[] accounts = new double[ACCOUNTS];

	public Bank() {
		for (int i = 0; i < accounts.length; i++) {
			accounts[i] = 200;
		}
	}

	@Atomic
	public void total() {
		double total = 0d;
		for (int i = 0; i < accounts.length; i++) {
			total += accounts[i];
		}
	}

	@Atomic
	public void transfer(int a, int b, double sum) {
		accounts[a] -= sum;
		accounts[b] += sum;
	}
}
```

```
public static void main(String[] args) {
   final Random random = new Random();
   final Bank bank = new Bank();
   List<Thread> transactions = new LinkedList<Thread>();

   for (int i = 0; i < Bank.TRANSFERS; i++) {
      final int a = random.nextInt(Bank.ACCOUNTS);
      final int b = random.nextInt(Bank.ACCOUNTS);
      final double sum = 5d + random.nextDouble() % 15d;
      Thread t = new Thread() {
         public void run() {
            bank.transfer(a, b, sum);
         }
      };
      t.start(); transactions.add(t);
   }

   Thread t = new Thread() {
      public void run() {
         bank.total();
      }
   };
   t.start(); transactions.add(t);

   for (Thread thread : transactions) {
      thread.join();
   }
}
```

Posortowane zamki

Posortowane zamki



T1

lock a, b

a -= sum
b += sum

unlock a, b

T2

lock b, c

b -= sum
c += sum

unlock b, c

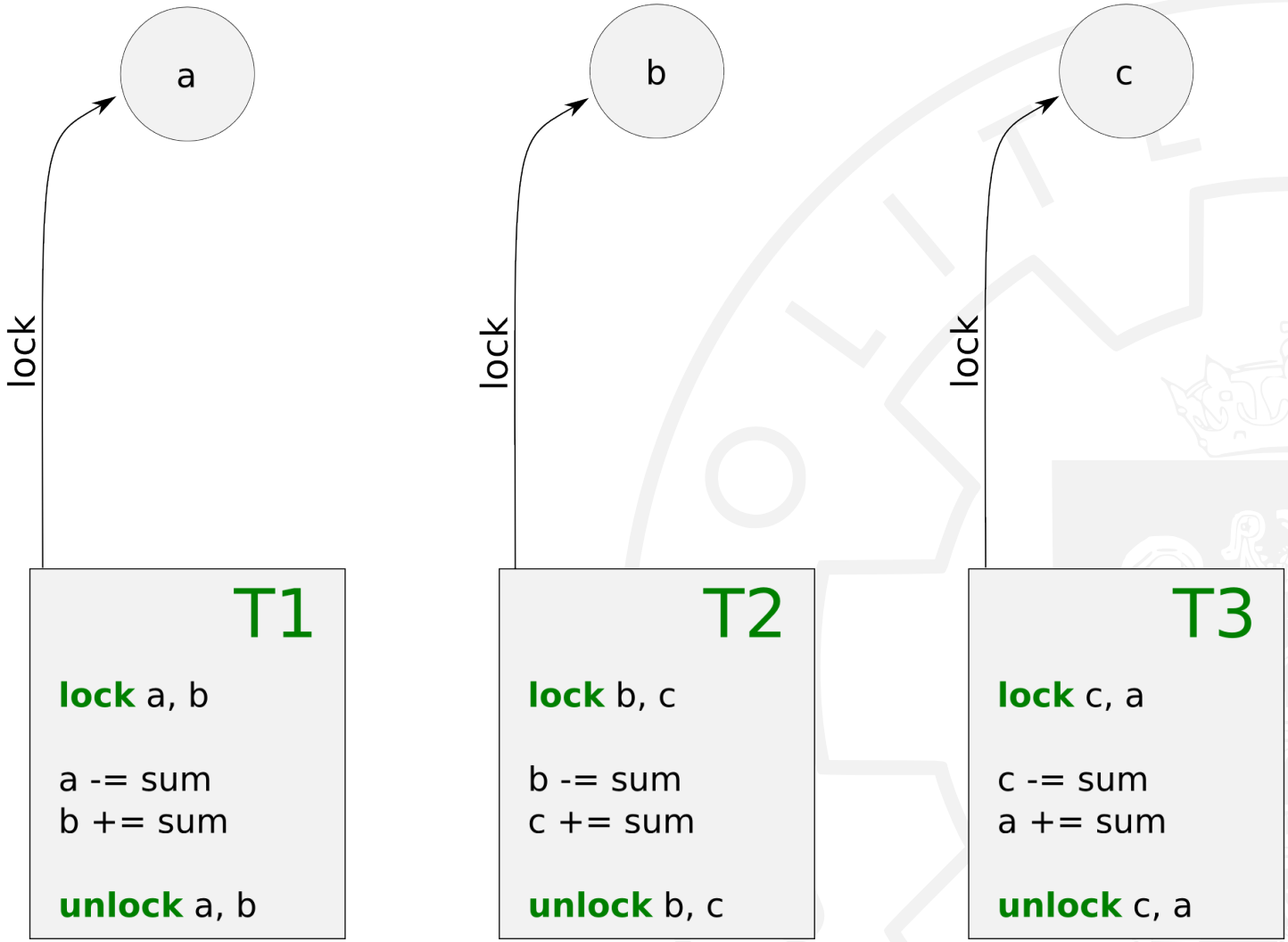
T3

lock c, a

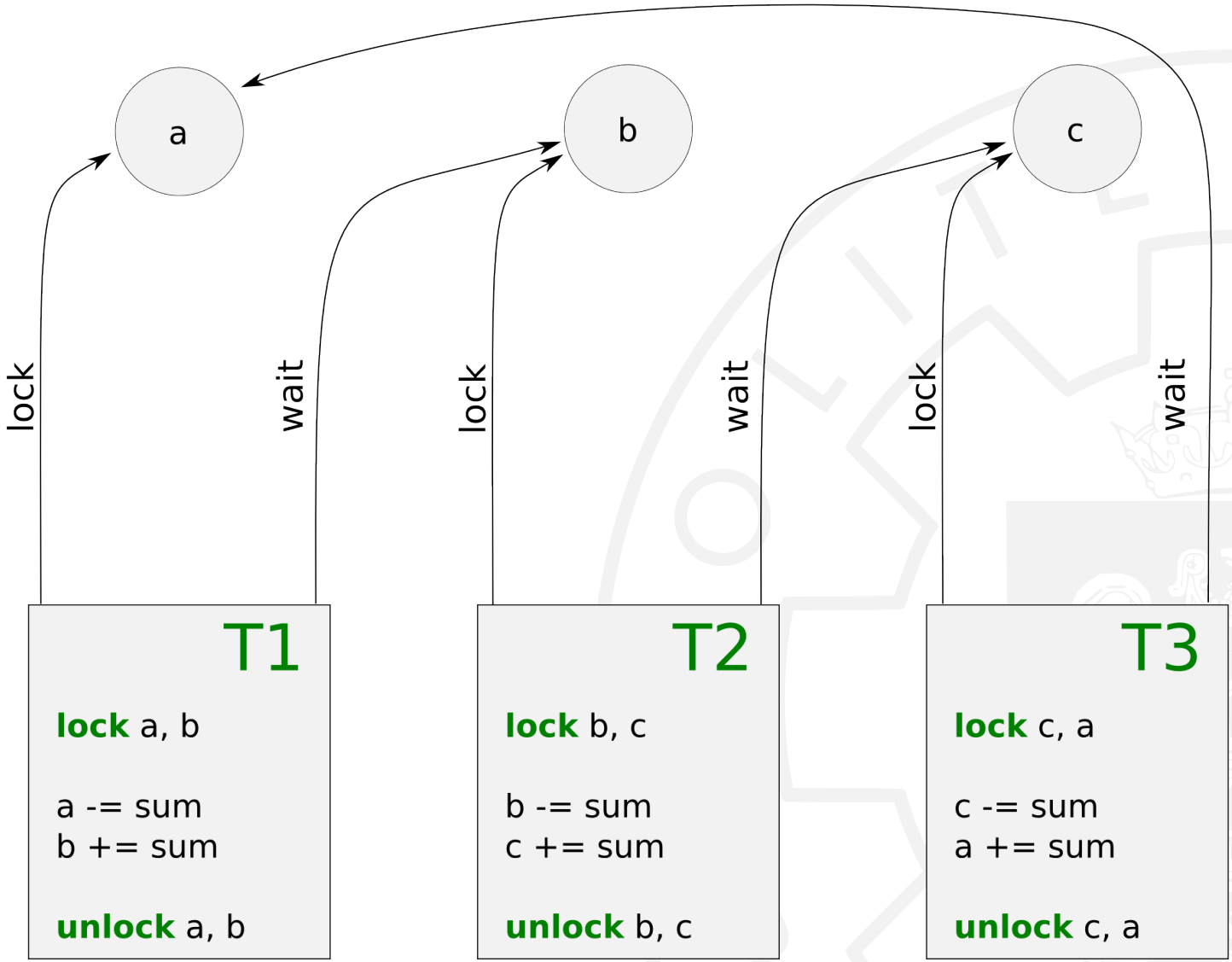
c -= sum
a += sum

unlock c, a

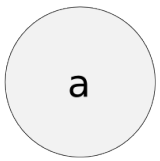
Posortowane zamki

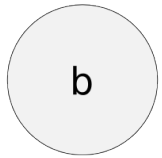


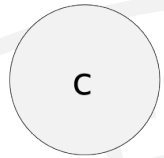
Posortowane zamki



Posortowane zamki







T1

lock a, b

a -= sum
b += sum

unlock a, b

T2

lock b, c

b -= sum
c += sum

unlock b, c

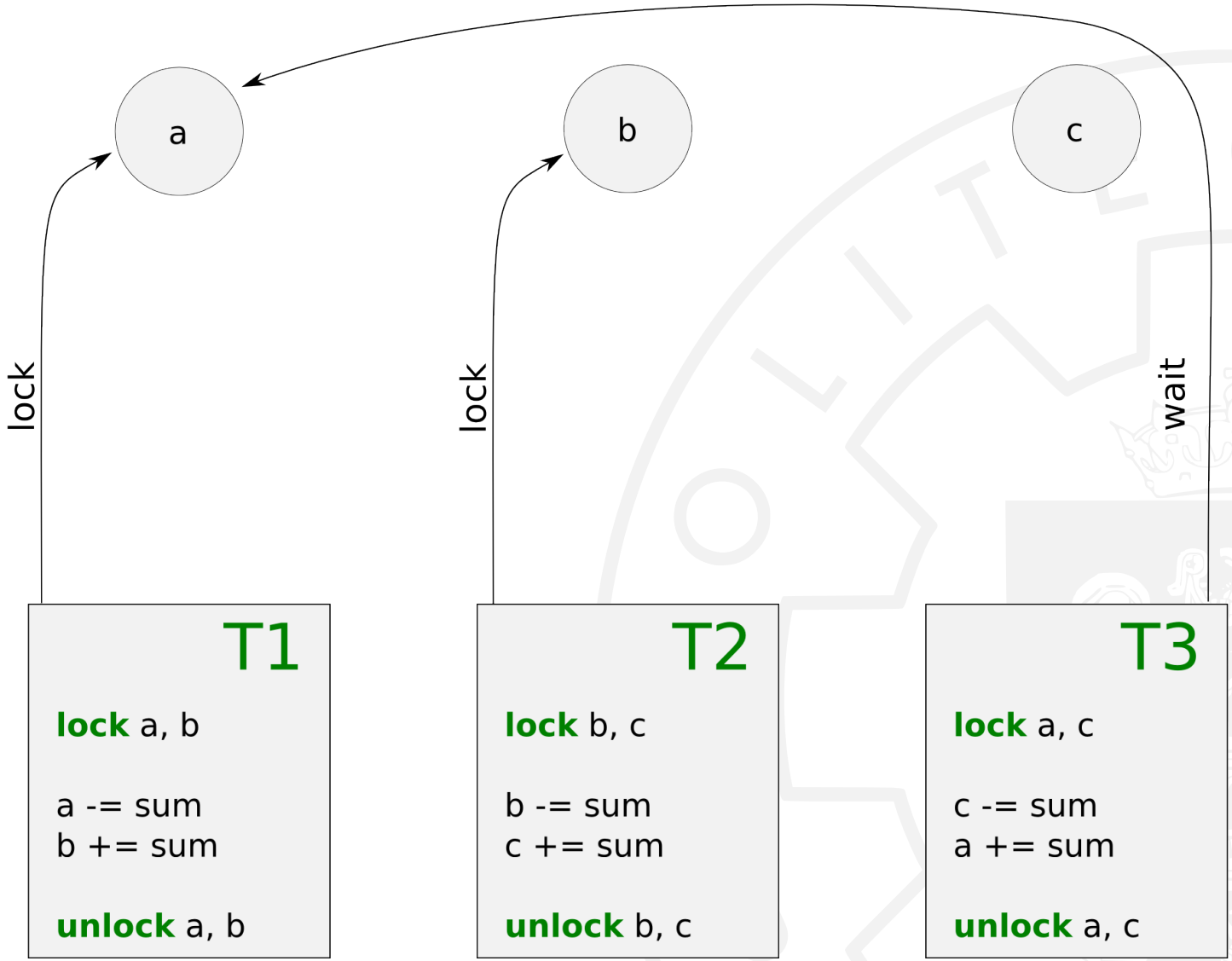
T3

lock a, c

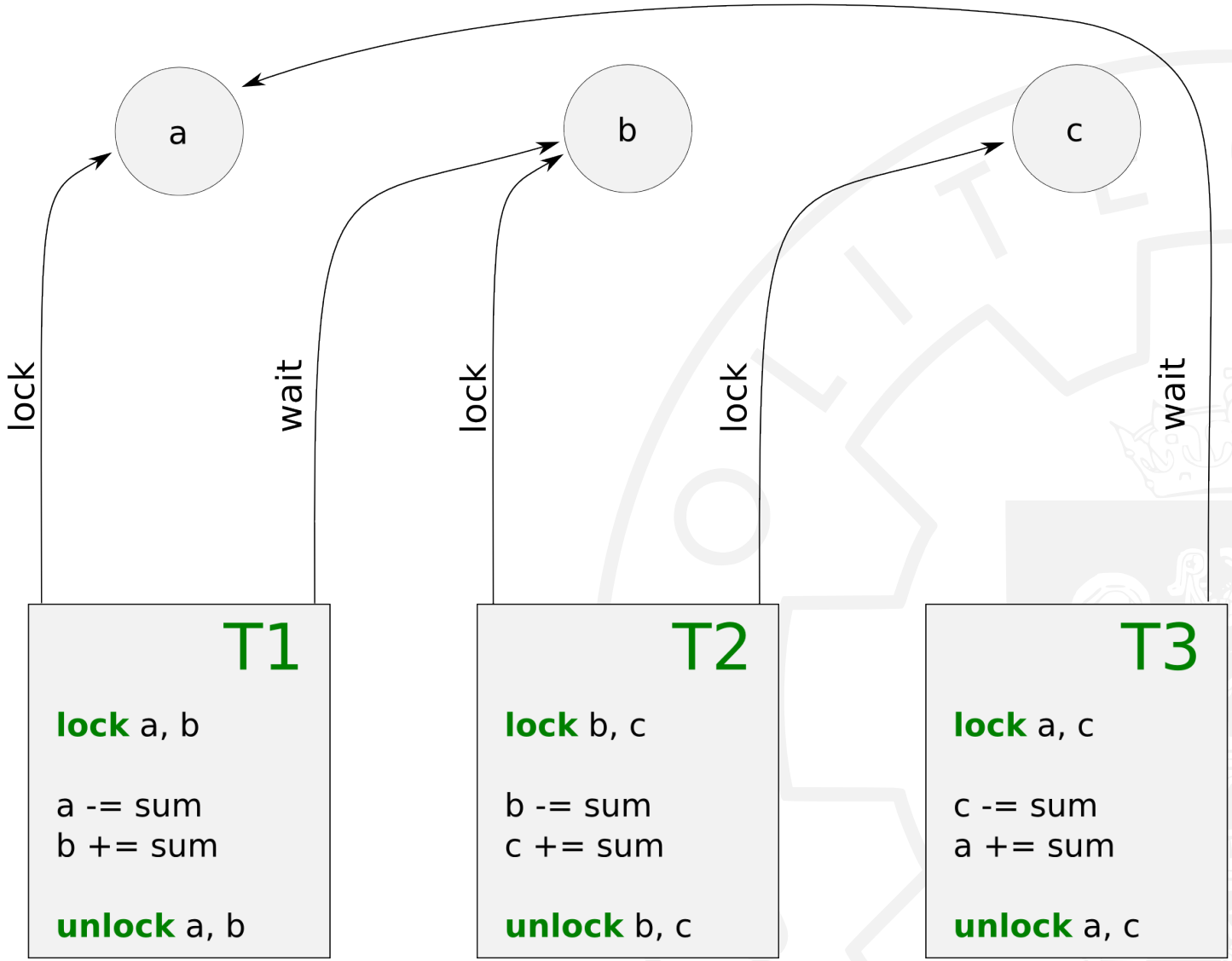
c -= sum
a += sum

unlock a, c

Posortowane zamki



Posortowane zamki



Posortowane zamki

```
from threading import Lock

accounts, locks = {}, {}

for i in range(0, ACCOUNTS):
    accounts[i] = 200
    locks[i] = Lock()

def transfer(a, b, sum):
    _locks = [locks[a], locks[b]]
    _locks.sort(key = lambda x: str(x))

    for lock in _locks:
        lock.acquire()

    accounts[a] -= sum
    locks[a].release()

    accounts[b] += sum
    locks[b].release()
```

Kiedy czego używać?

- **Wątki funcyjne**

Kiedy czego używać?

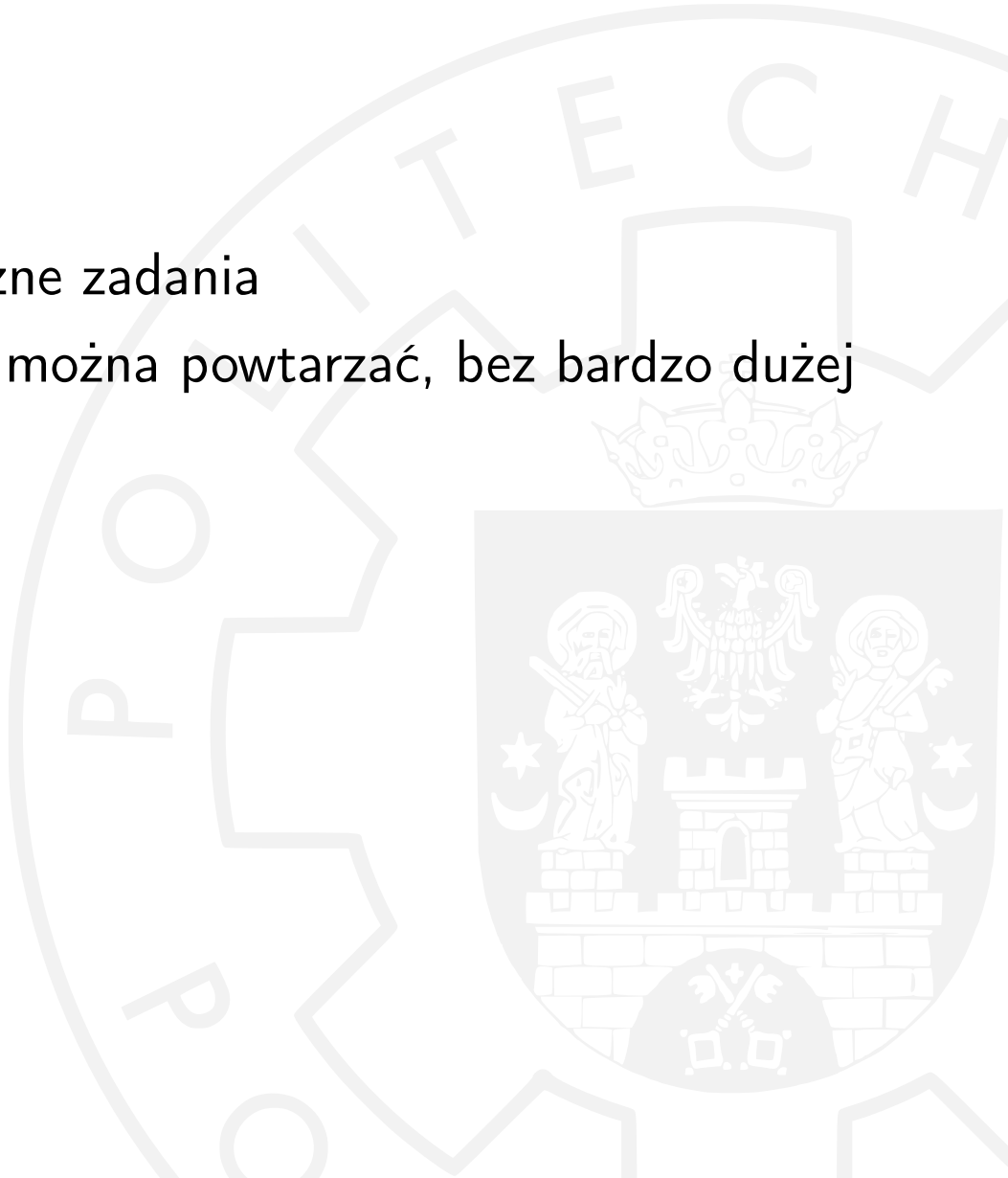
- **Wątki funcyjne** – niezależne zadania

Kiedy czego używać?

- **Wątki funcyjne** – niezależne zadania
- **Transakcje**

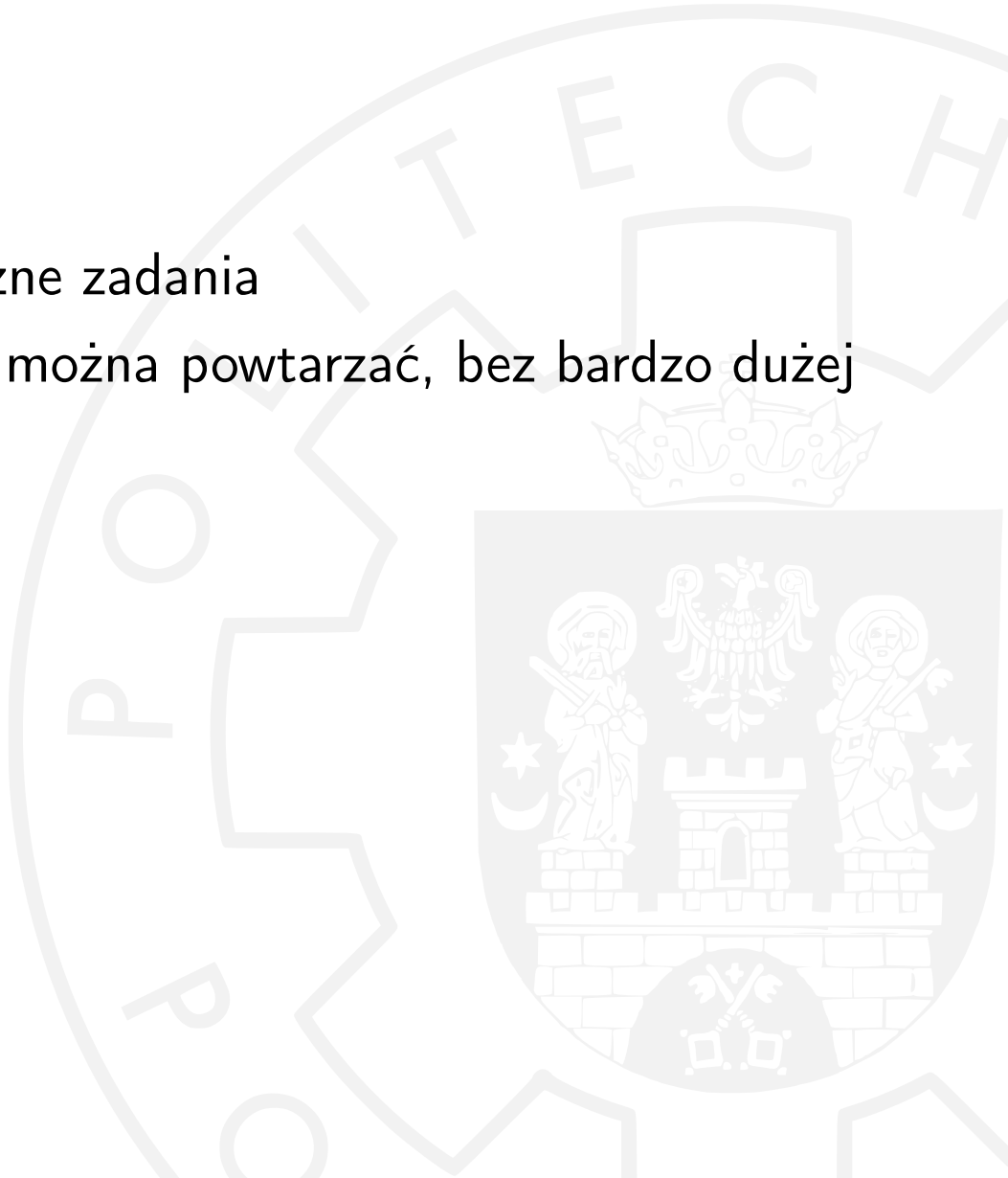
Kiedy czego używać?

- **Wątki funcyjne** – niezależne zadania
- **Transakcje** – wszystko co można powtarzać, bez bardzo dużej konkurencji o zasoby



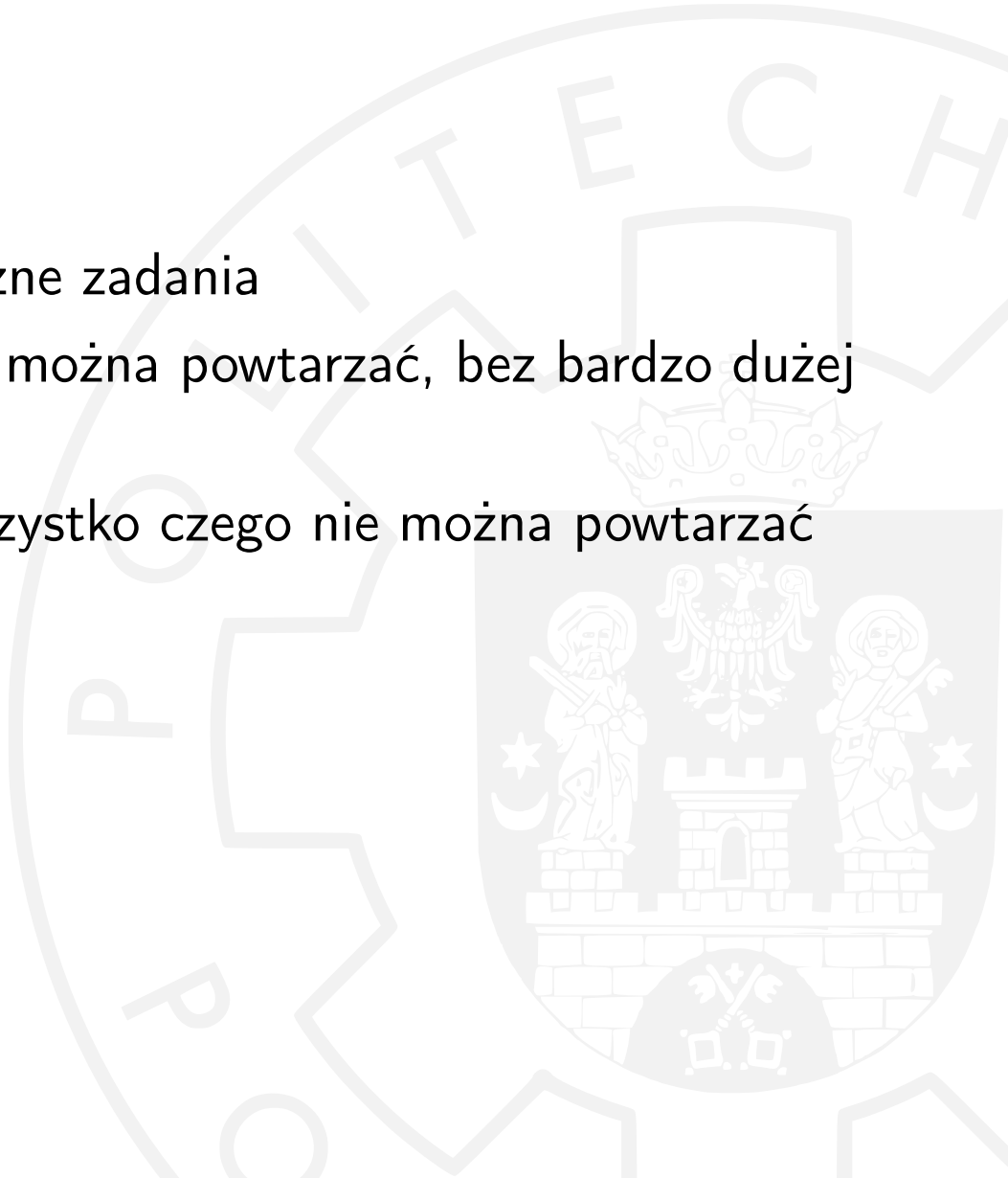
Kiedy czego używać?

- **Wątki funcyjne** – niezależne zadania
- **Transakcje** – wszystko co można powtarzać, bez bardzo dużej konkurencji o zasoby
- **Posortowane zamki**



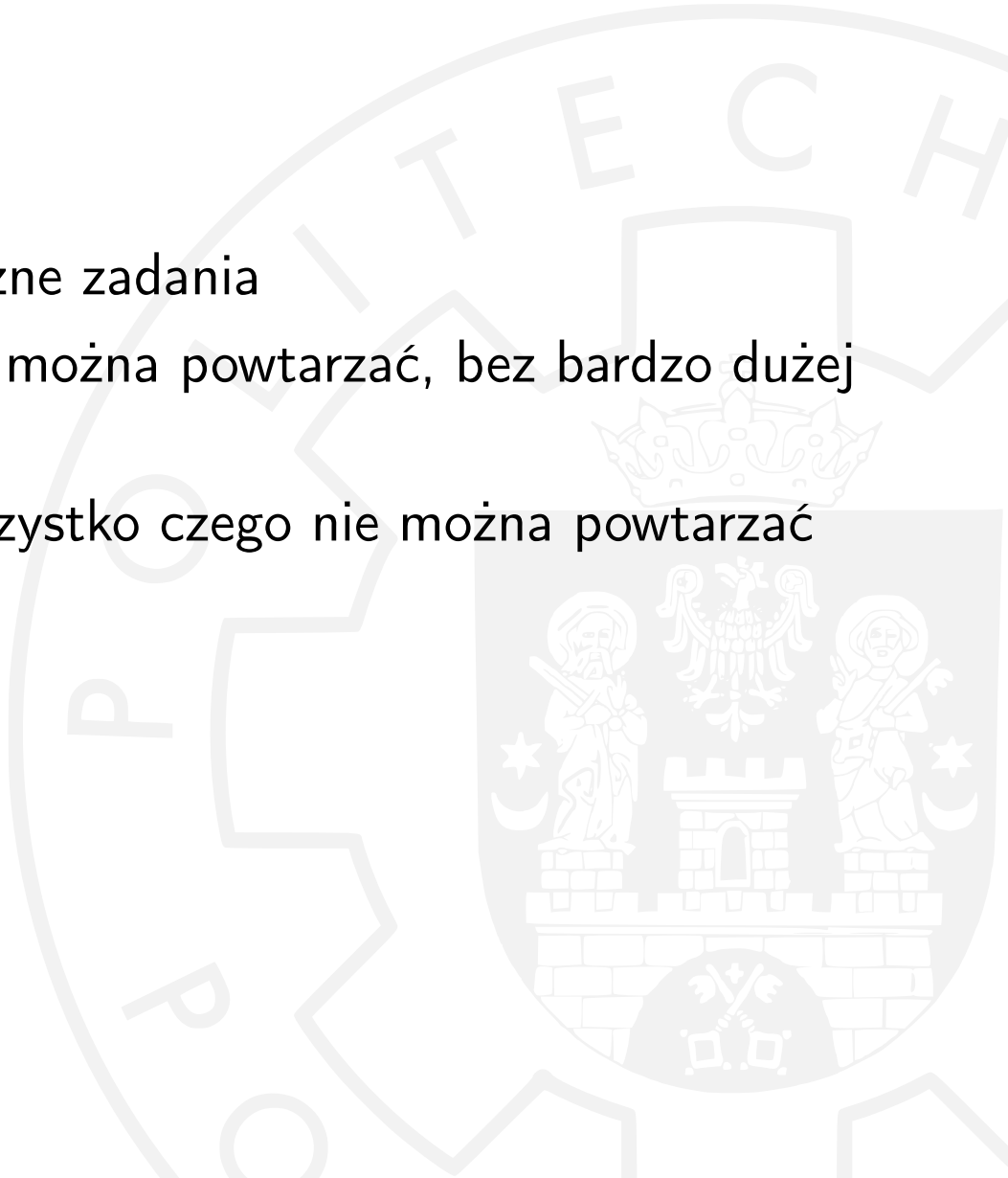
Kiedy czego używać?

- **Wątki funcyjne** – niezależne zadania
- **Transakcje** – wszystko co można powtarzać, bez bardzo dużej konkurencji o zasoby
- **Posortowane zamki** – wszystko czego nie można powtarzać



Kiedy czego używać?

- **Wątki funcyjne** – niezależne zadania
- **Transakcje** – wszystko co można powtarzać, bez bardzo dużej konkurencji o zasoby
- **Posortowane zamki** – wszystko czego nie można powtarzać
- **Globalny zamek**



Kiedy czego używać?

- **Wątki funcyjne** – niezależne zadania
- **Transakcje** – wszystko co można powtarzać, bez bardzo dużej konkurencji o zasoby
- **Posortowane zamki** – wszystko czego nie można powtarzać
- **Globalny zamek** – najlepiej nigdy

Kiedy czego używać?

- **Wątki funcyjne** – niezależne zadania
- **Transakcje** – wszystko co można powtarzać, bez bardzo dużej konkurencji o zasoby
- **Posortowane zamki** – wszystko czego nie można powtarzać
- **Globalny zamek** – najlepiej nigdy

Koło Naukowe SKiSR

coming soon

skisr-kolo@libra.cs.put.poznan.pl

Koło Naukowe SKiSR

coming soon

skisr-kolo@libra.cs.put.poznan.pl

konrad.siek@cs.put.edu.pl