PPaDS MMXX

Matus Jokay, C-503, matus.jokay@stuba.sk
Roderik Ploszek, C-512, roderik.ploszek@stuba.sk

uim.fei.stuba.sk/predmet/i-ppds

konzultacie dohodou

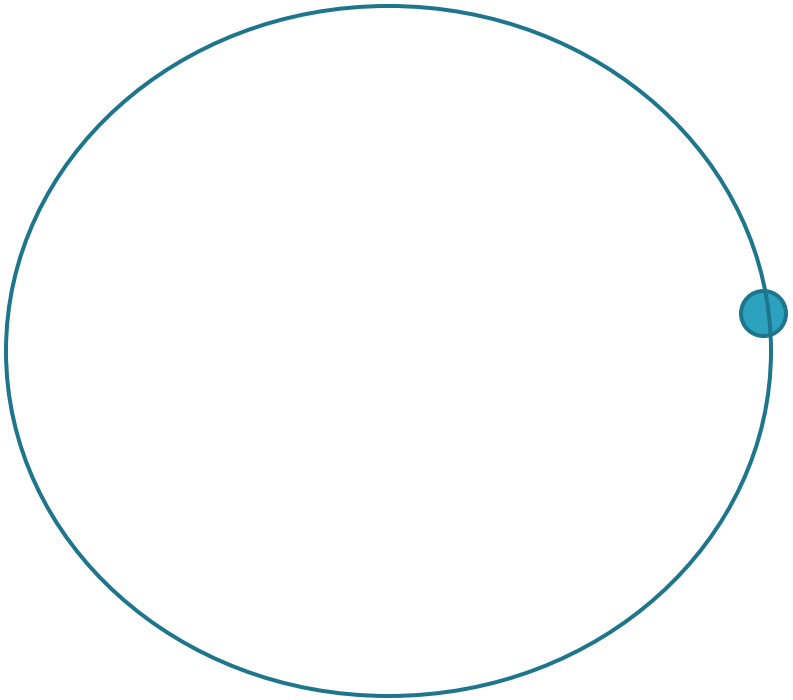
Opakovanie…

- Bariera
- Este raz bariera
- SimpleBarrier cez Event
- Fibonacci

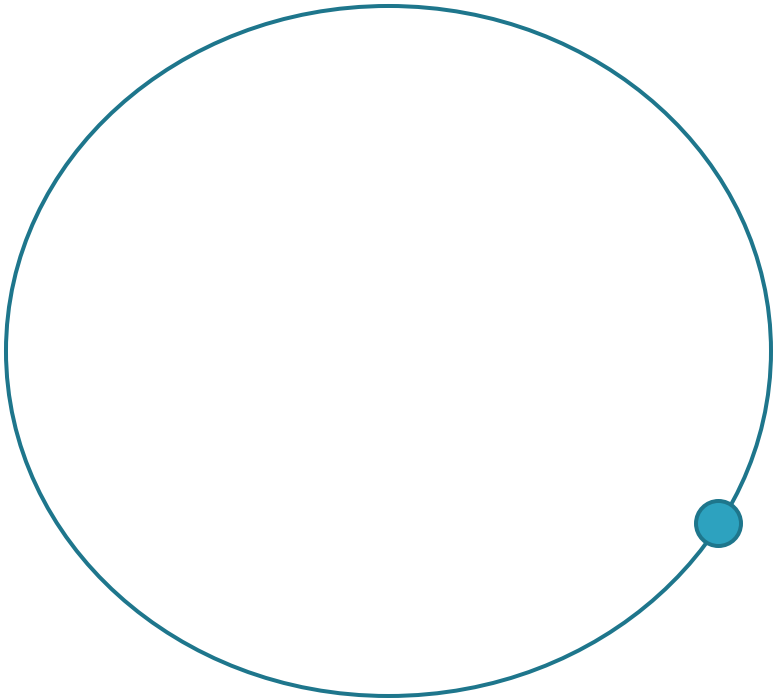
Bariera…

Bariera…

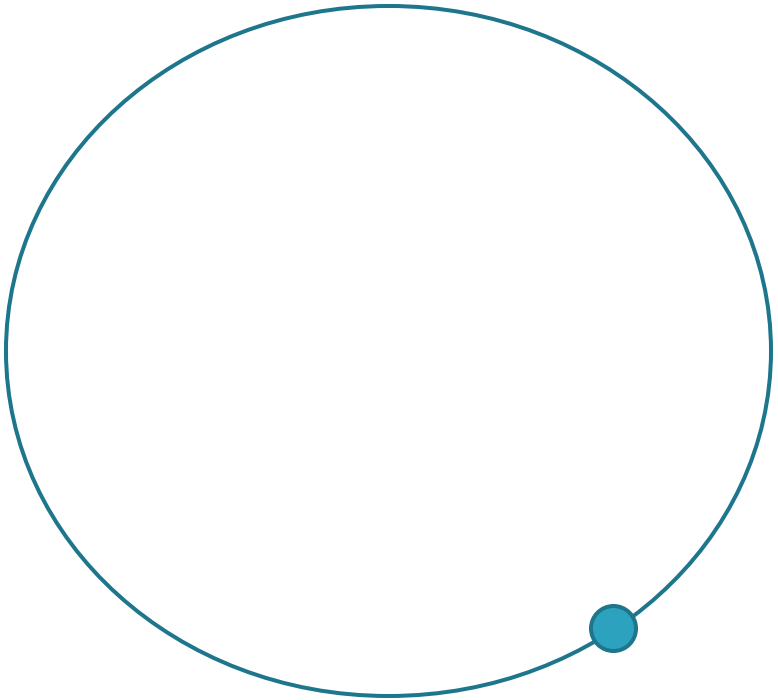
Bariera…



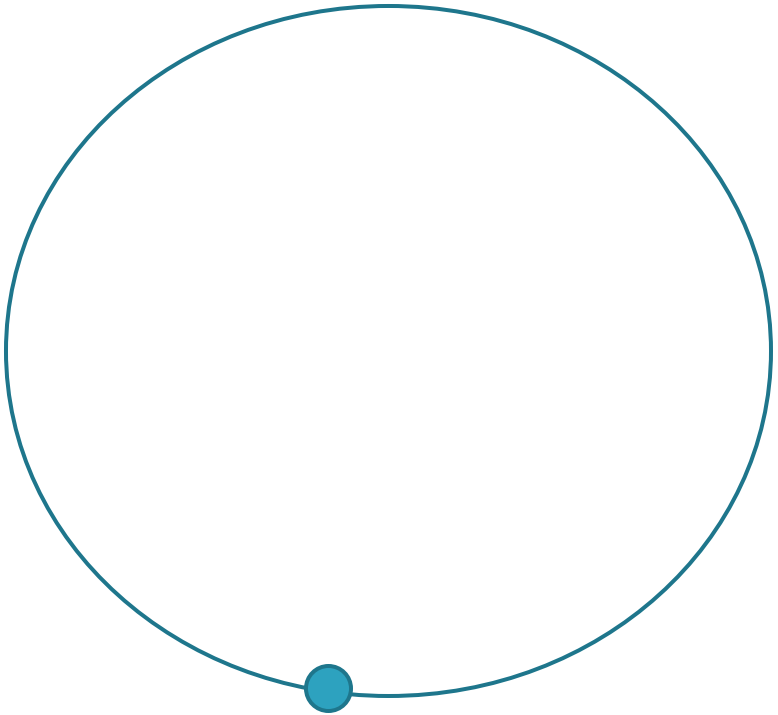
Bariera…



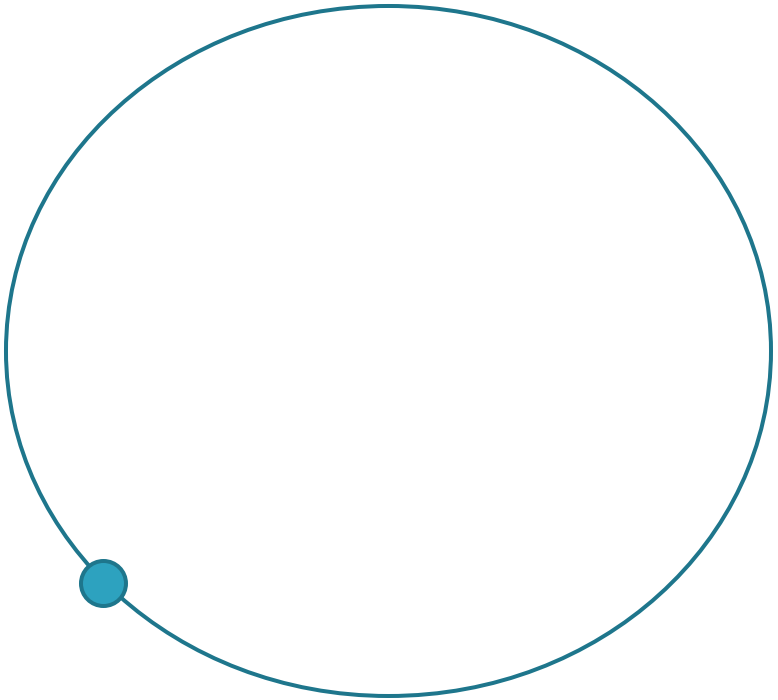
Bariera…



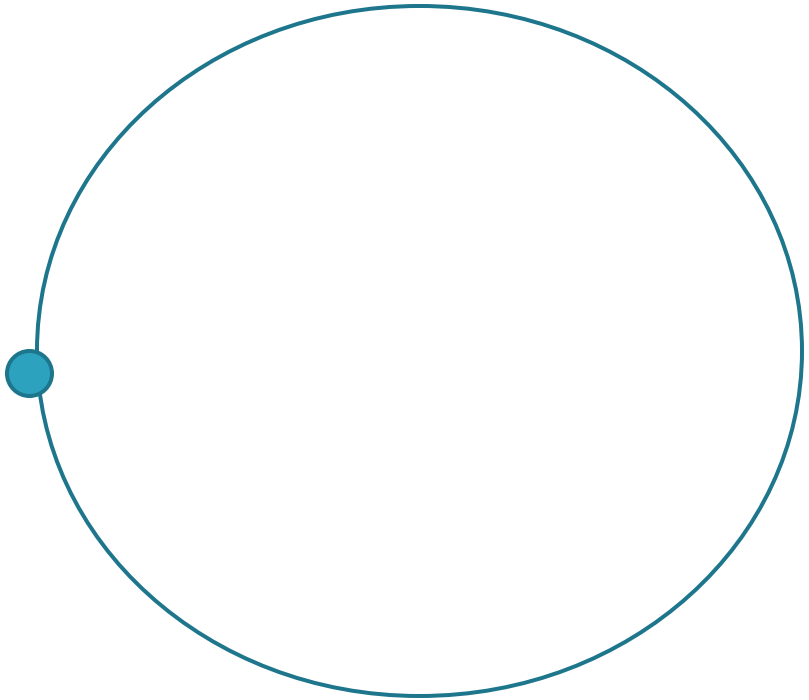
Bariera…



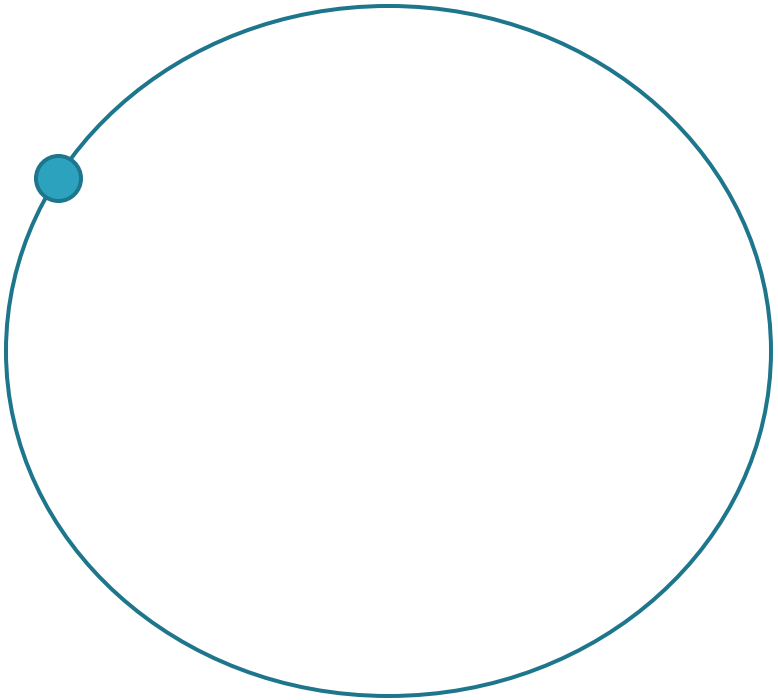
Bariera…



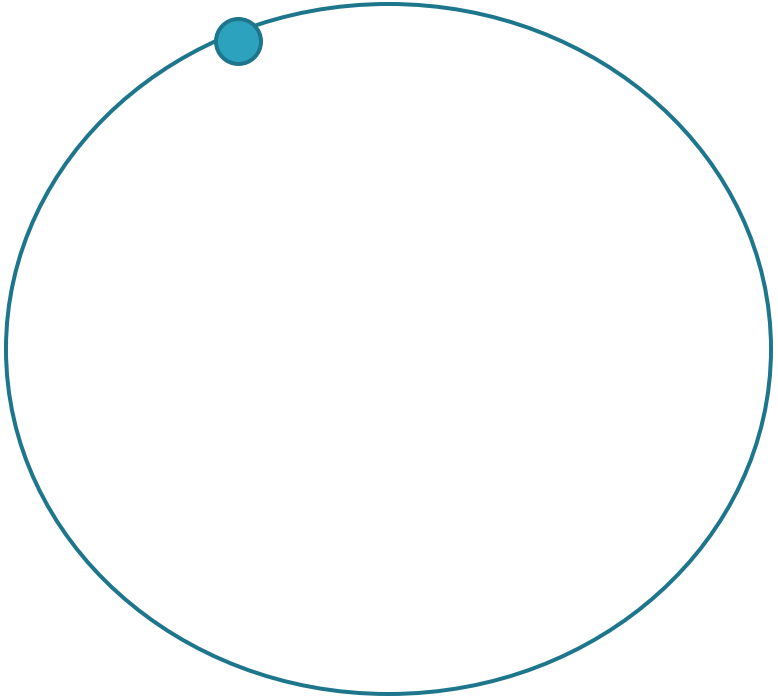
Bariera…



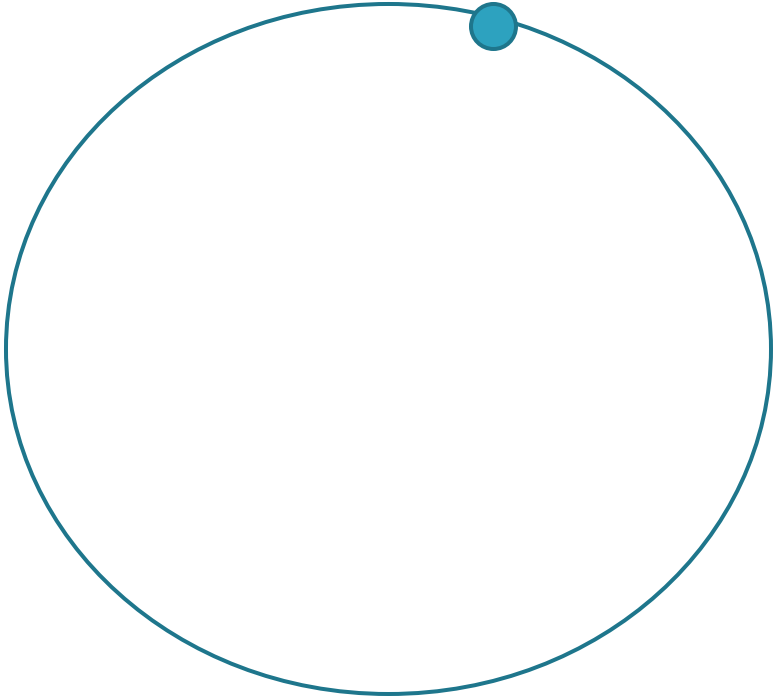
Bariera…



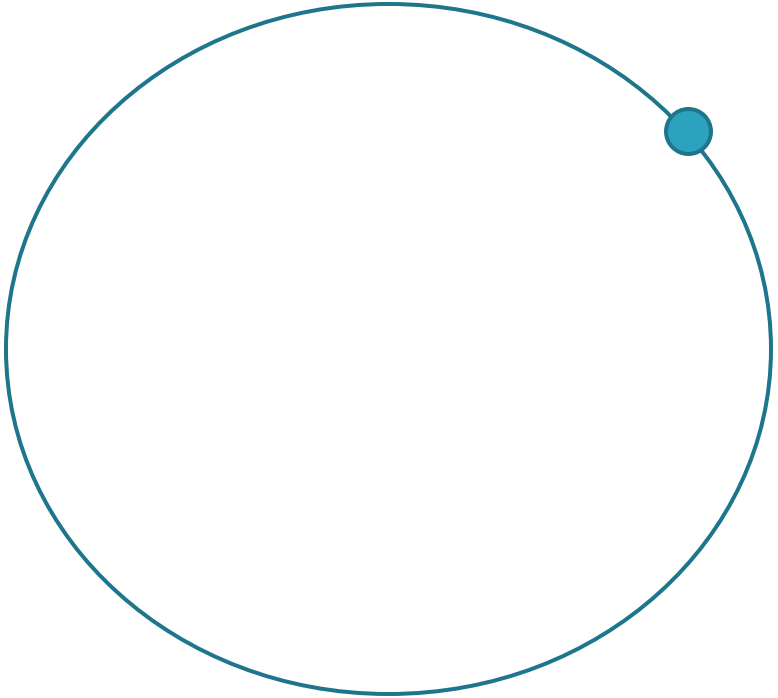
Bariera…



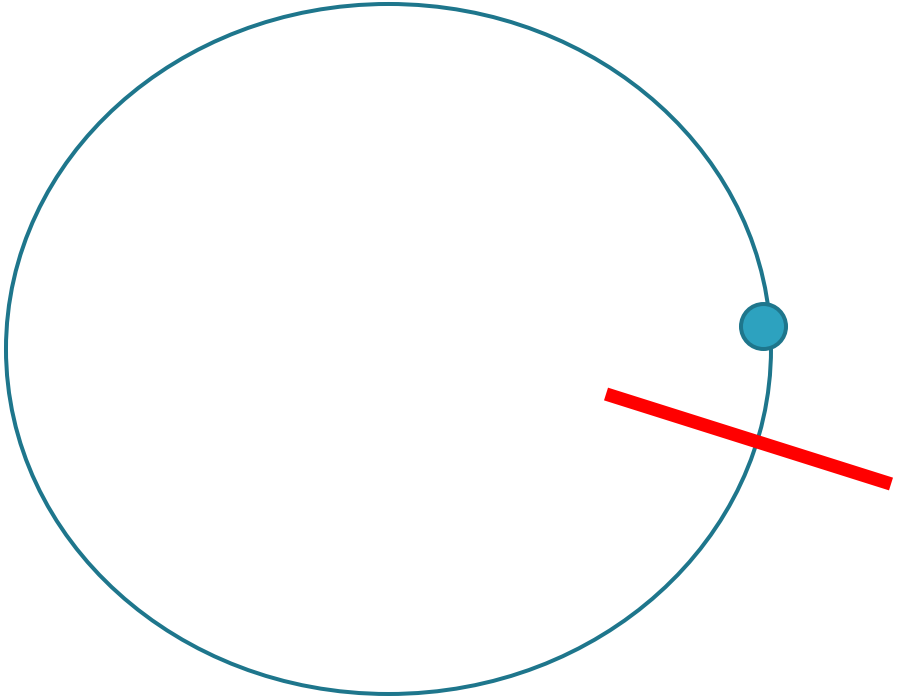
Bariera…



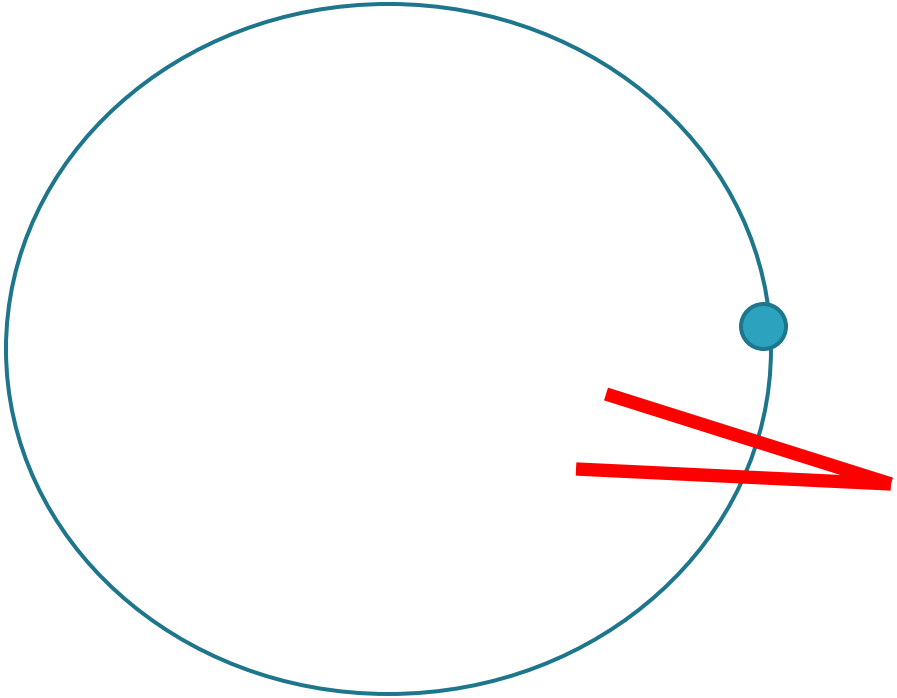
Bariera…



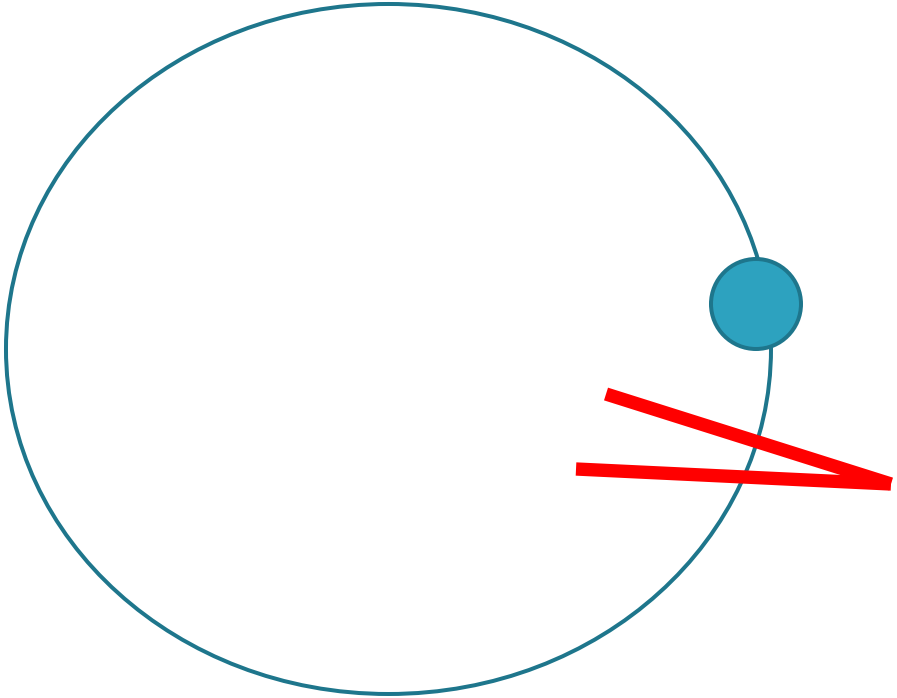
Bariera…



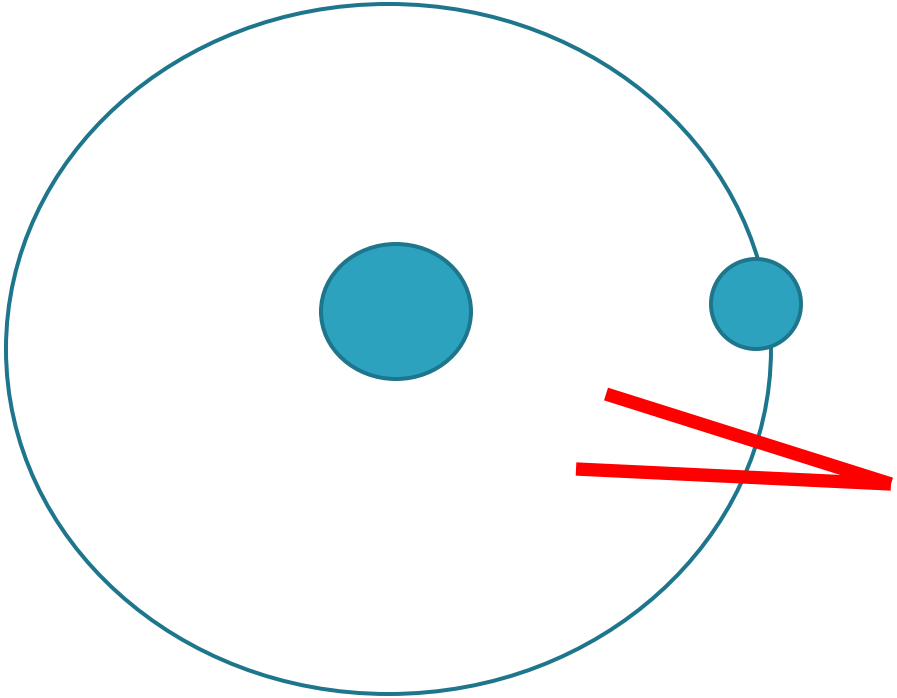
Bariera…



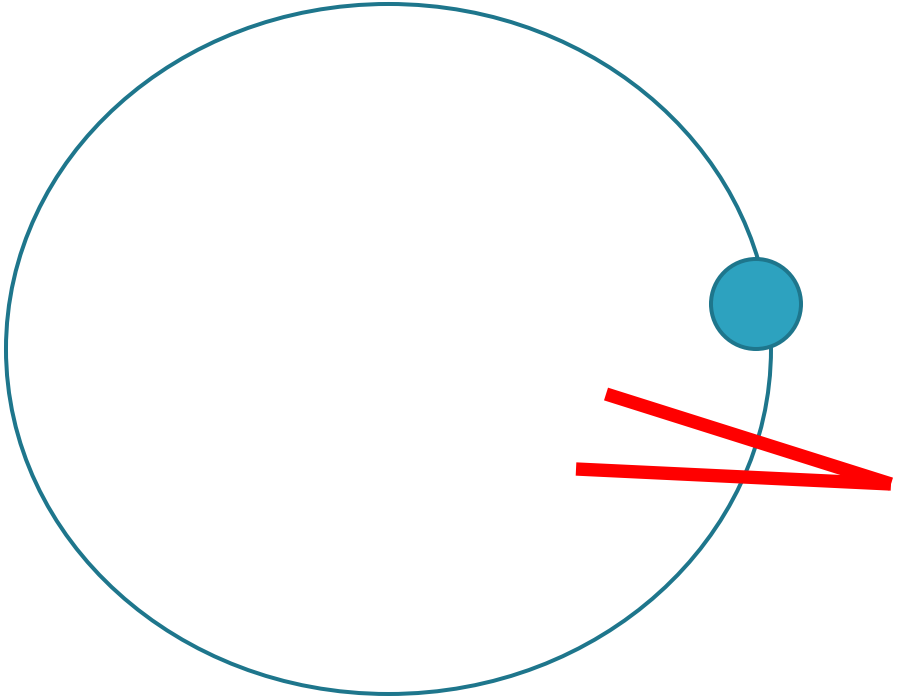
Bariera…



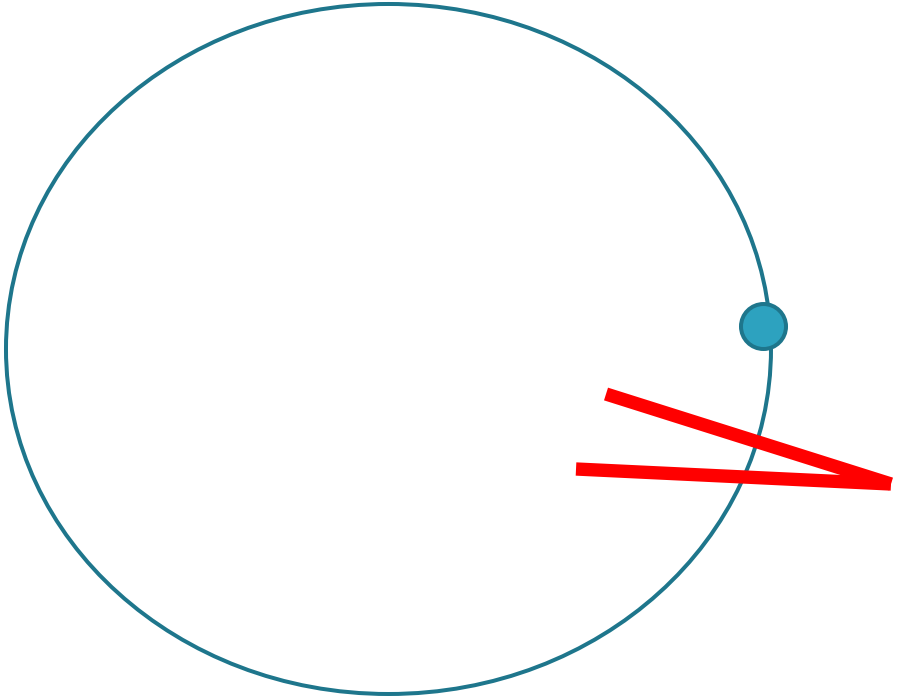
Bariera…



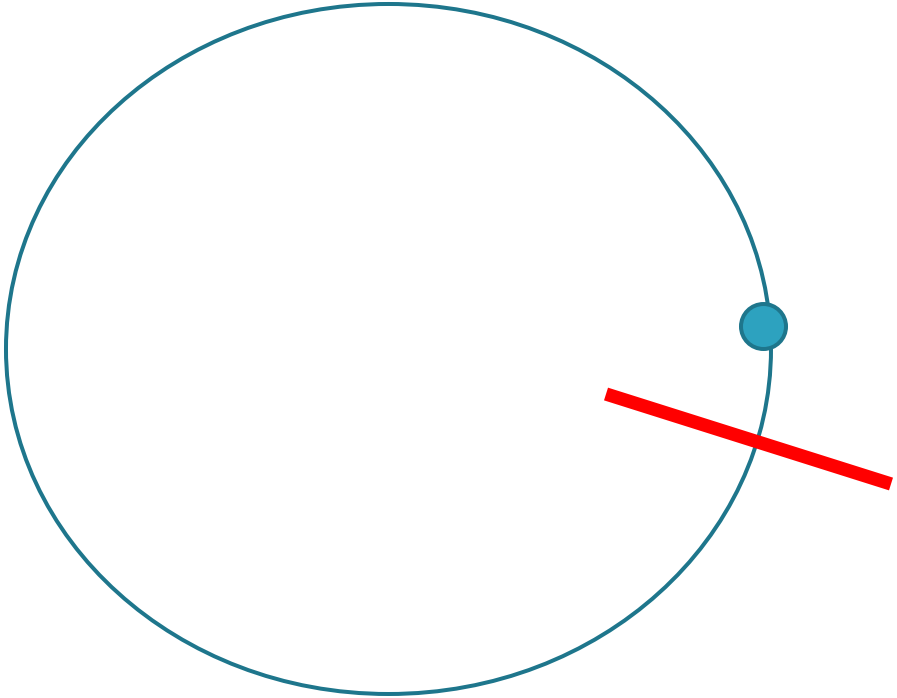
Bariera…



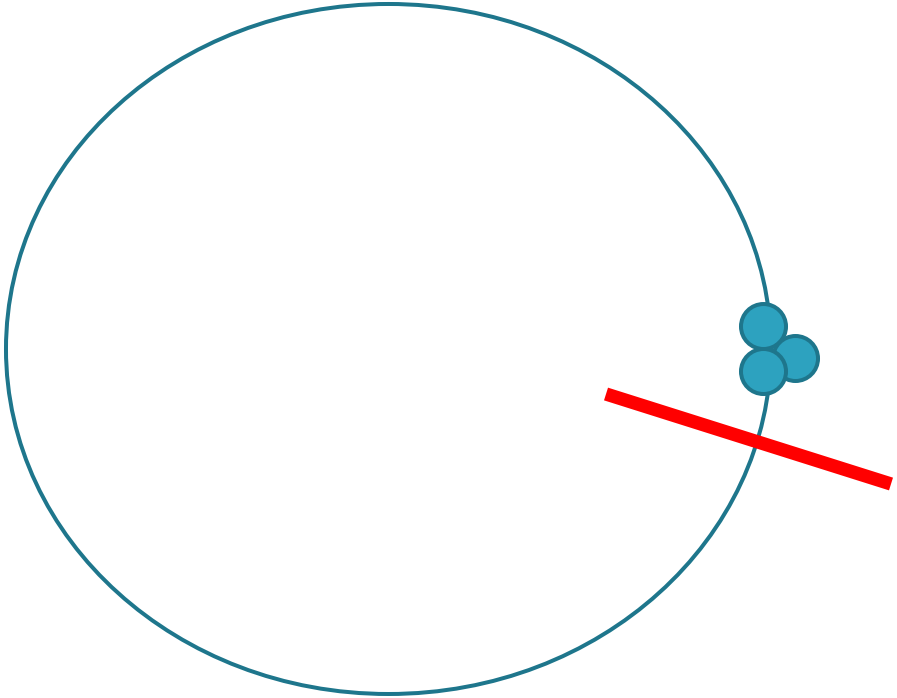
Bariera…



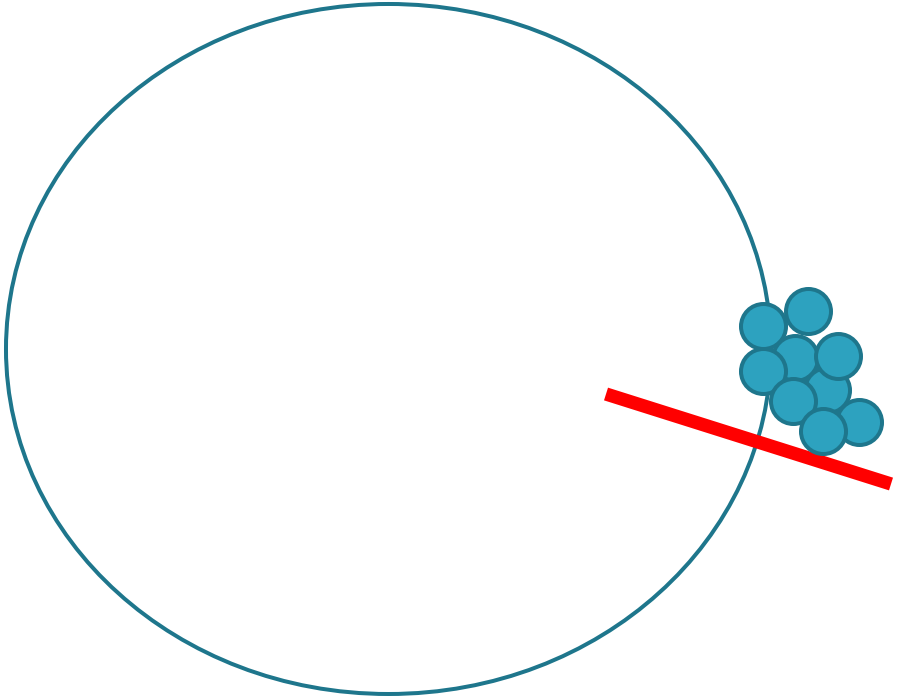
Bariera…



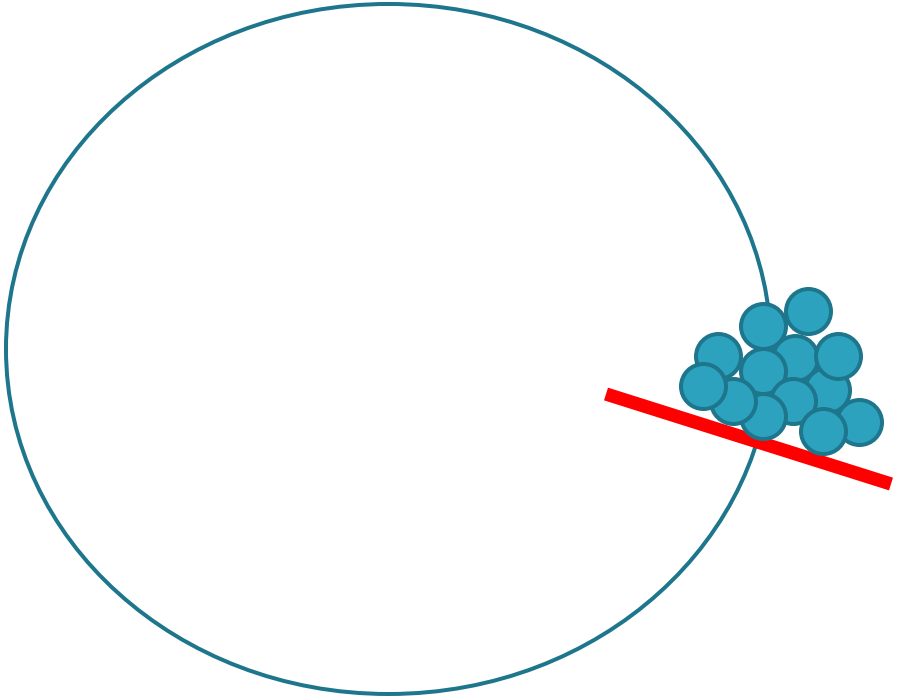
Bariera…



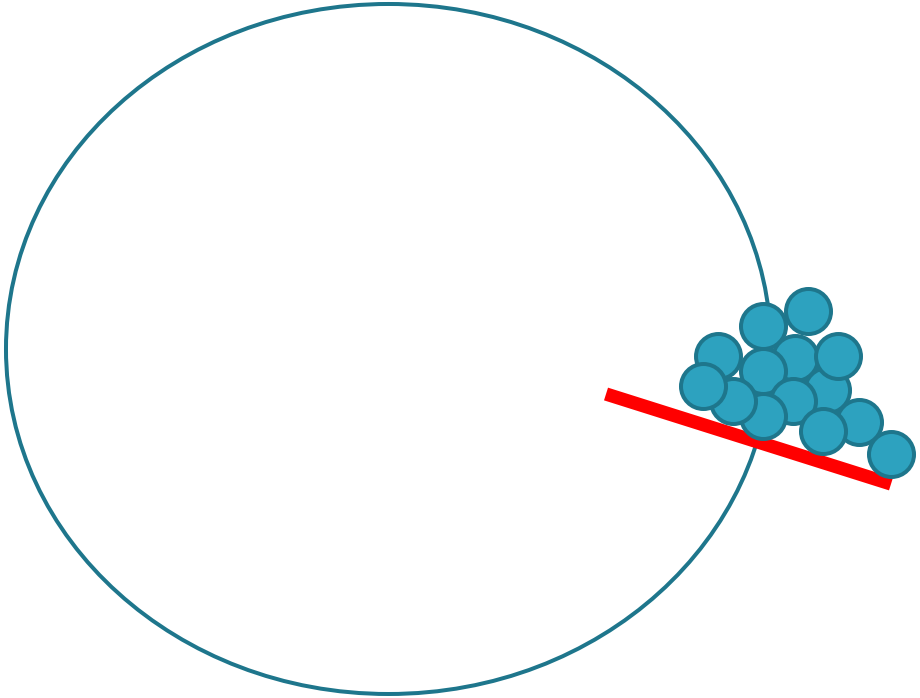
Bariera…



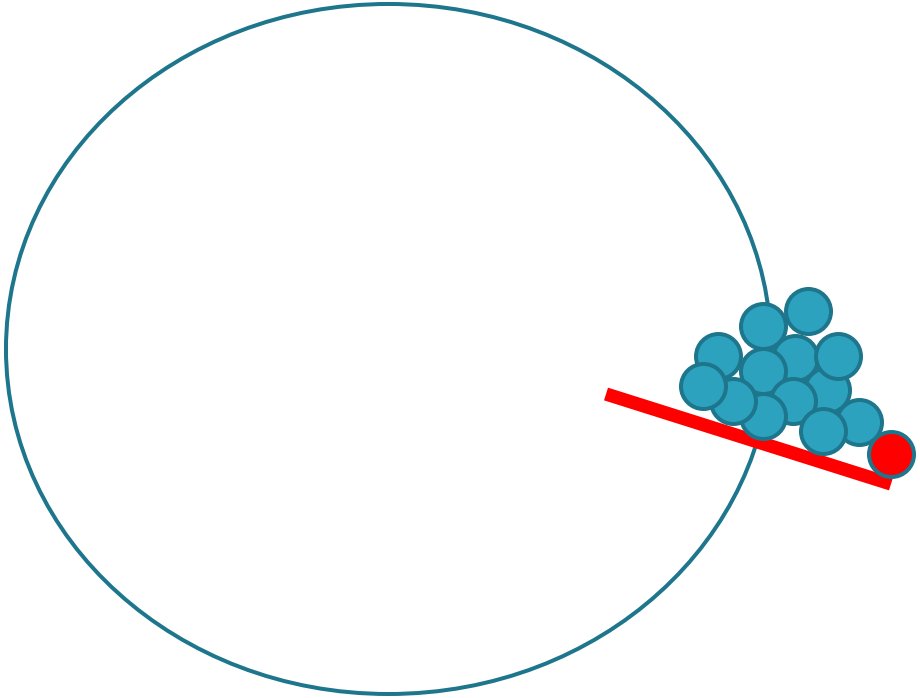
Bariera…



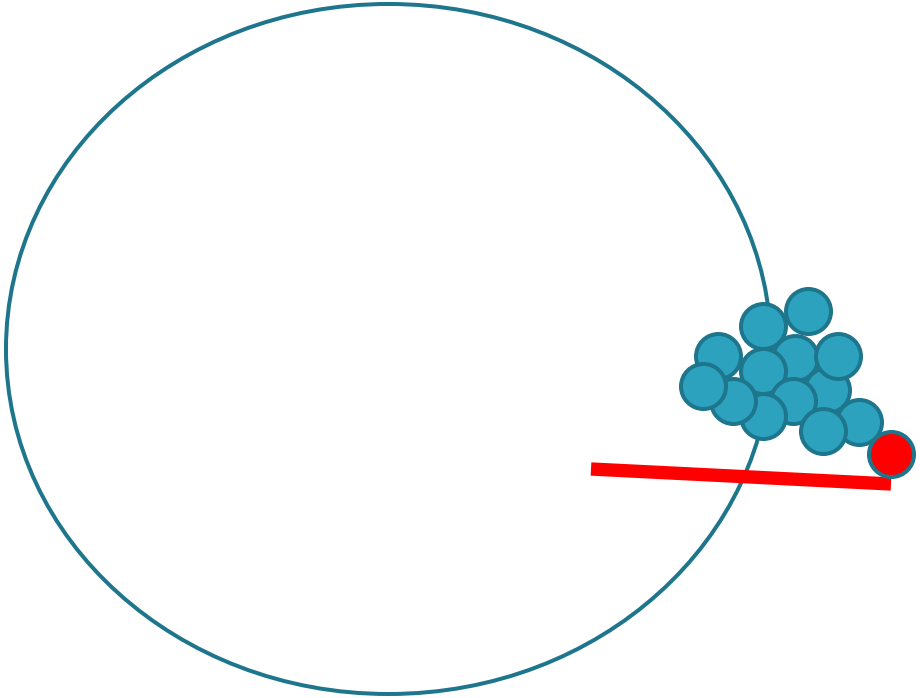
Bariera…



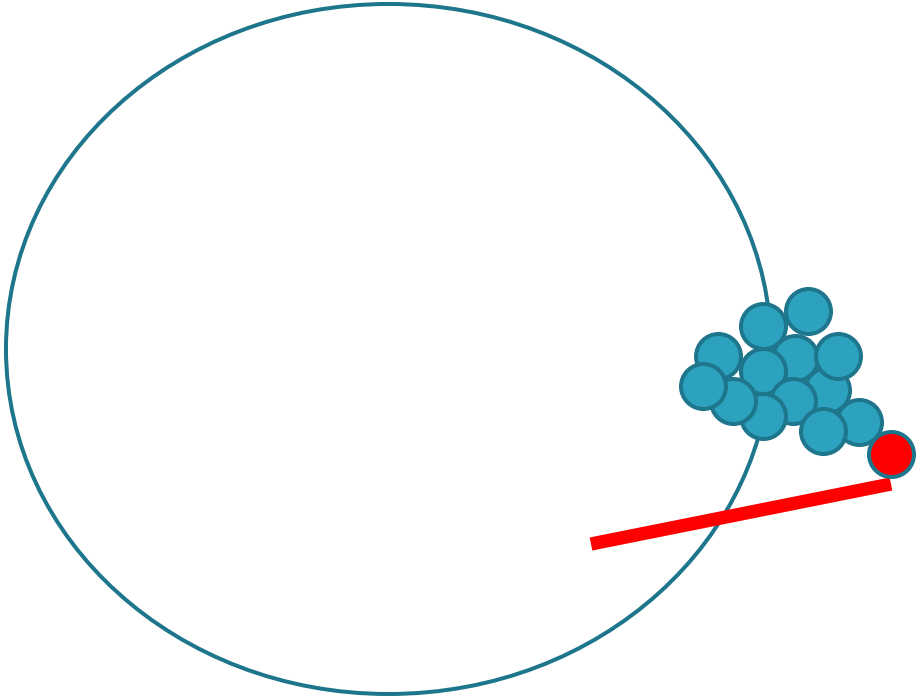
Bariera…



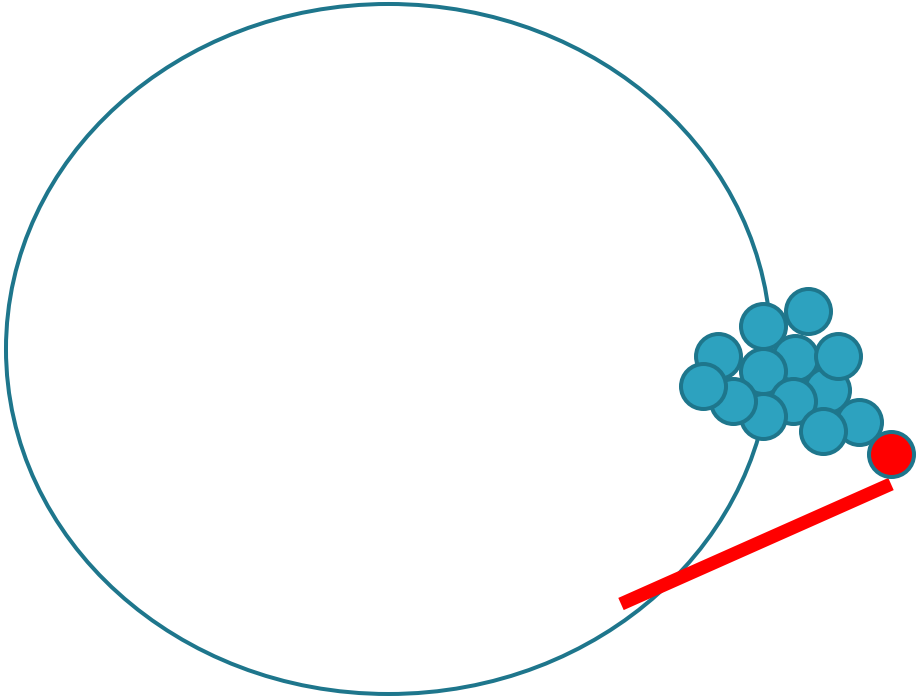
Bariera…



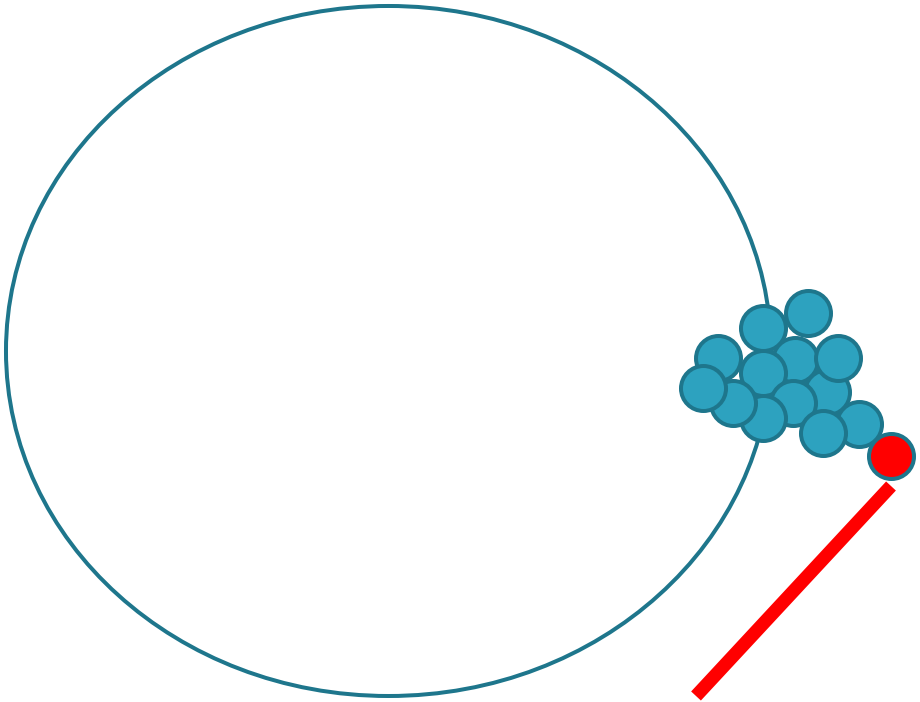
Bariera…



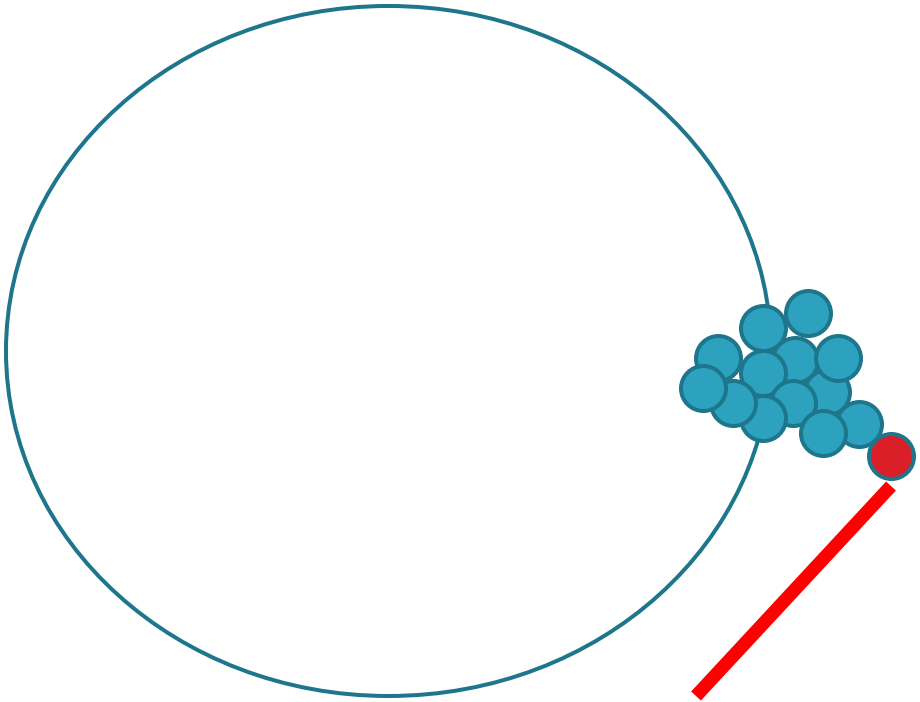
Bariera…



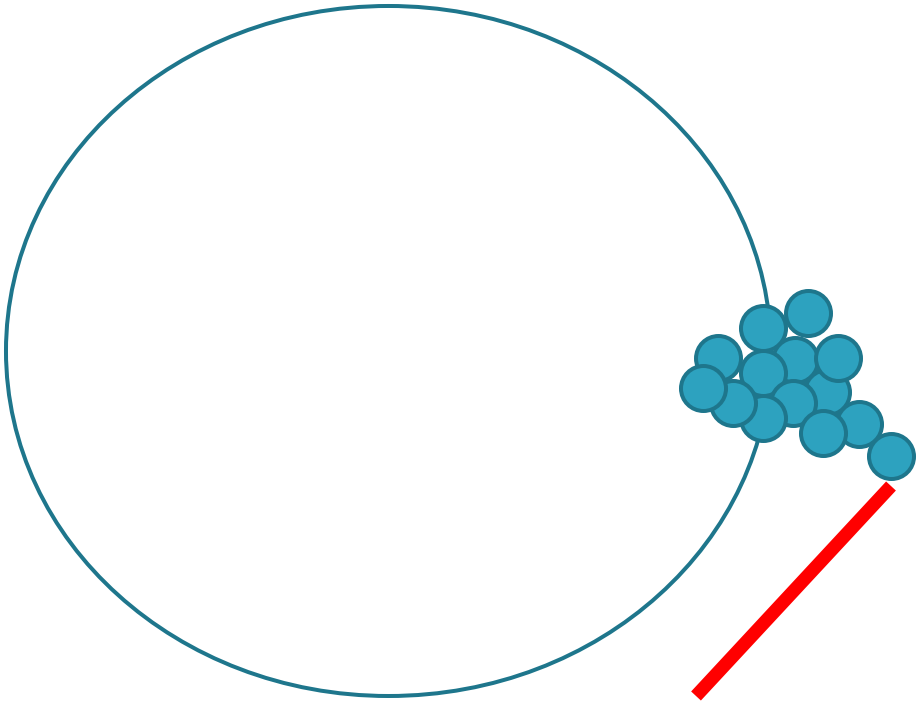
Bariera…



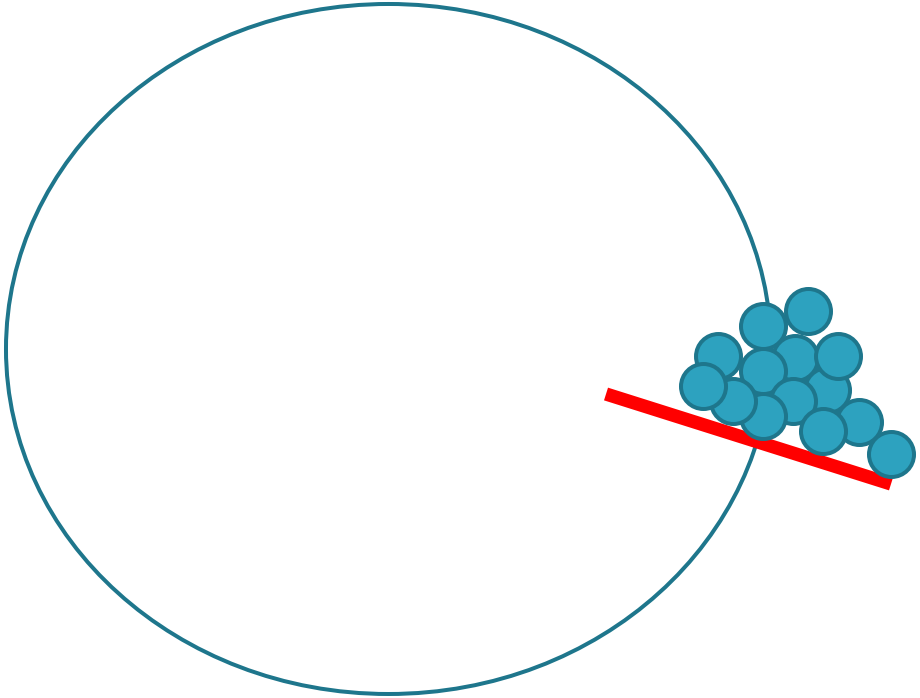
Bariera…



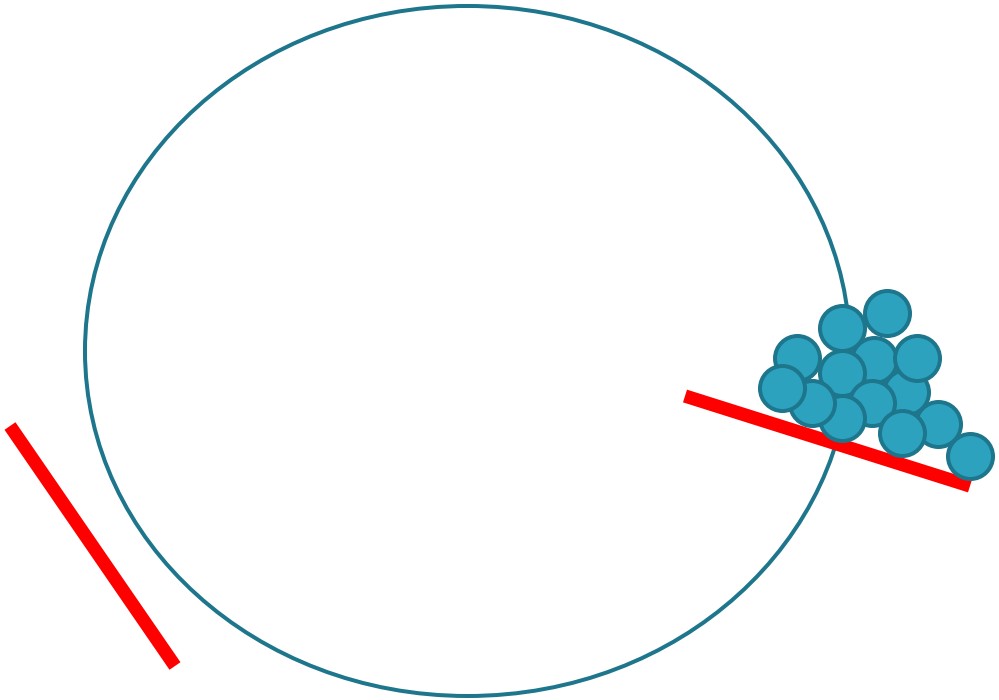
Bariera…



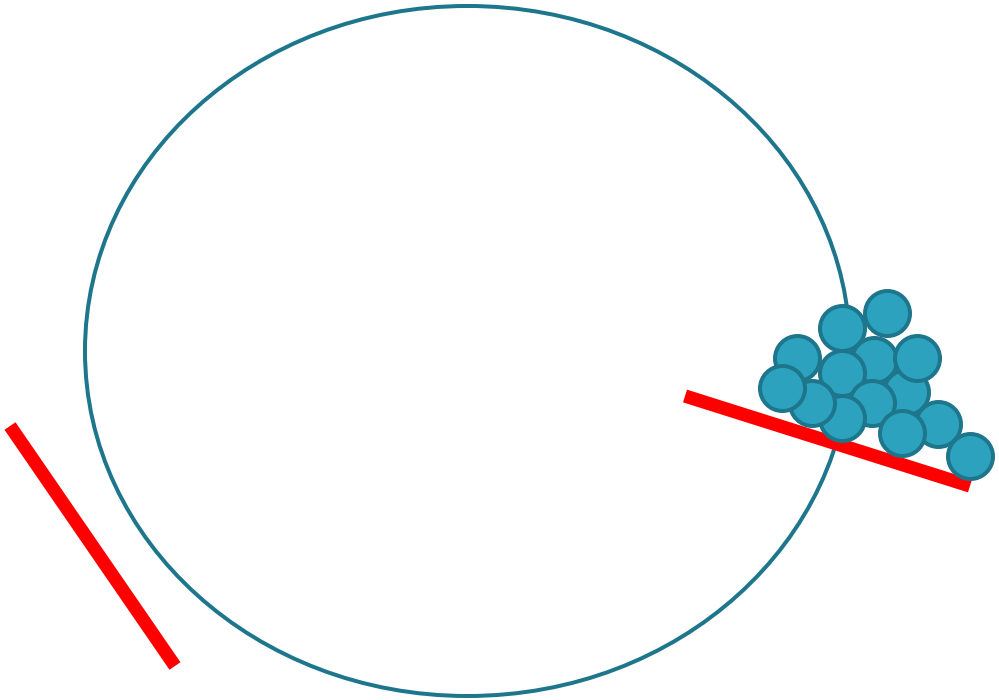
Bariera…



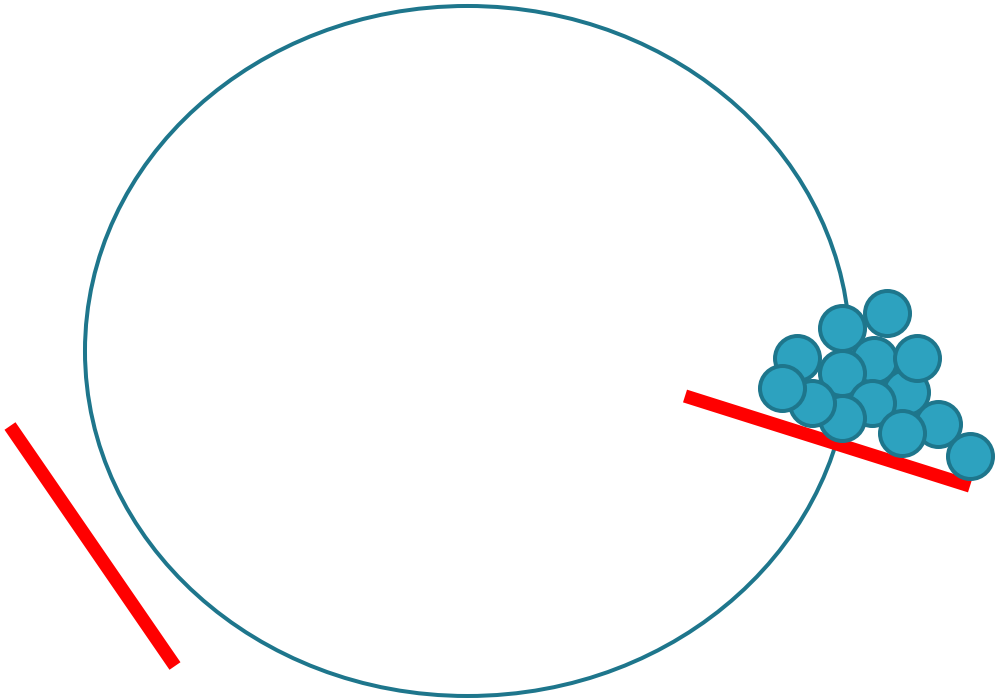
Bariera…



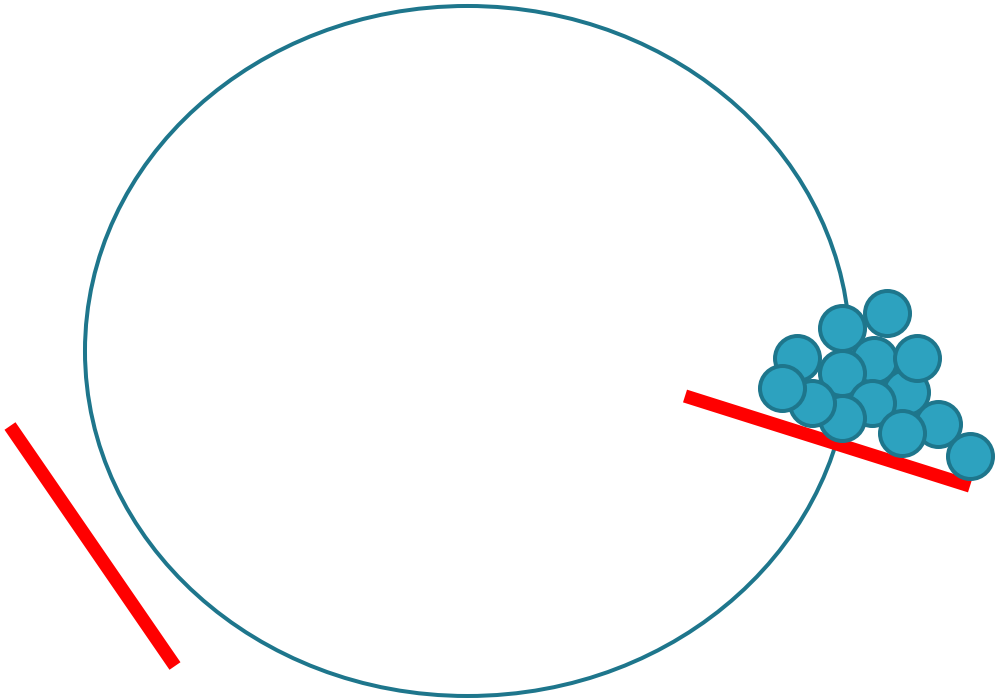
Bariera…



Bariera…



Bariera…



Este raz bariera

1) while True:
2) rendezvous
3) s_barr1()
4) nejaky kod
5) s_barr2()

Init(N):

s_barr1 ← SimpleBarrier(N)

s_barr2 ← SimpleBarrier(N)

Este raz bariera

```
1) while True:
2)     rendezvous
3)     s_barr1()
4)     nejaky kod
5)     s_barr2()
```

```
1) while True:
2)     s_barr2()
3)     rendezvous
4)     s_barr1()
5)     nejaky kod
```

Este raz bariera

1) while True:
2) s_barr2()
3) miesto stretnutia
4) s_barr1()
5) kod vykonavany po stretnuti sa

Este raz bariera

1) while True:
2) kod vykonavany pred stretnutim sa
3) s_barr2()
4) miesto stretnutia
5) s_barr1()
6) kod vykonavany po stretnuti sa

Este raz bariera

1) while True:
2) kod vykonavany pred stretnutim sa
3) s_barr2()
4) miesto stretnutia
5) s_barr1()
6) kod vykonavany po stretnuti sa

SimpleBarrier cez Semafor

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          T.signal(N)
6)  M.unlock()
7)  T.wait()
```

Init(N):

C ← 0

M ← Mutex()

T ← Semaphore(0)

SimpleBarrier cez Event

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
```

Init(N):

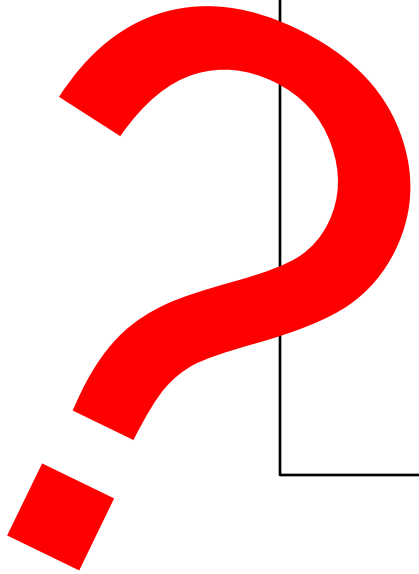
C ← 0

M ← Mutex()

E ← Event()

SimpleBarrier cez Event

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
```



Init(N):

C ← 0

M ← Mutex()

E ← Event()

SimpleBarrier cez Event

```
1) M.lock()
2)     C += 1
3)     if C == N:
4)         C = 0
5)         E.set()
6) M.unlock()
7) E.wait()
```

SimpleBarrier cez Event

```
1) M.lock()
2)       C += 1
3)       if C == N:
4)             C = 0
5)             E.set()
6) M.unlock()
7) E.wait()
```

```
1) while True:
2)       s_barrier1()
3)       s_barrier2()
```

SimpleBarrier cez Event

```
1) M.lock()
2)       C += 1
3)       if C == N:
4)             C = 0
5)             E.set()
6) M.unlock()
7) E.wait()
```

```
1) while True:
2)       s_barrier1()
3)       s_barrier2()
```

- Udalost ostava stale nastavena!

SimpleBarrier cez Event

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
8)
```

```
1)  while True:
2)      s_barrier1()
3)      s_barrier2()
```

- Opravime

SimpleBarrier cez Event

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
8)  E.clear()
```

```
1)  while True:
2)      s_barrier1()
3)      s_barrier2()
```

- Opravime

SimpleBarrier cez Event

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
8)  E.clear()
```

```
1)  while True:
2)      s_barrier1()
3)      s_barrier2()
```

- Samozrejme zle

SimpleBarrier cez Event

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
8)  E.clear()
```

```
1)  while True:
2)      s_barrier1()
3)      s_barrier2()
```

- Samozrejme zle
  - Vid ukazka…

SimpleBarrier cez Event

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
8)  E.clear()
```

```
1)  while True:
2)      s_barrier1()
3)      s_barrier2()
```

- Posledne vlakno, ktore **prejde cez** wait(), musi spustat clear()!

SimpleBarrier cez Event

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
```

```
8)
9)      C += 1
10)     if C == N:
11)         ....
12)
13)
```

SimpleBarrier cez Event

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
```

```
8)  M.lock()
9)      C += 1
10)     if C == N:
11)         …
12)
13) M.unlock()
```

SimpleBarrier cez Event

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
```

```
8)  M.lock()
9)      C += 1
10)     if C == N:
11)
12)         E.clear()
13) M.unlock()
```

SimpleBarrier cez Event

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
```

```
8)  M.lock()
9)      C += 1
10)     if C == N:
11)         C = 0
12)         E.clear()
13) M.unlock()
```

SimpleBarrier cez Event

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
```

```
8)  M.lock()
9)      C += 1
10)     if C == N:
11)         C = 0
12)         E.clear()
13) M.unlock()
```

SimpleBarrier cez Event OPTIM1

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          E.set()
6)  M.unlock()
7)  E.wait()
```

```
8)  M.lock()
9)      C -= 1
10)     if C == 0:
11)         C = 0
12)         E.clear()
13) M.unlock()
```

SimpleBarrier cez Event OPTIM1

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          E.set()
5)  M.unlock()
6)  E.wait()
```

```
7)  M.lock()
8)      C -= 1
9)      if C == 0:
10)         E.clear()
11) M.unlock()
```

SimpleBarrier cez Event OPTIM1

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          E.set()
5)  M.unlock()
6)  E.wait()
```

```
7)  M.lock()
8)      C -= 1
9)      if C == 0:
10)         E.clear()
11) M.unlock()
```

Nie optimalne… 2x serializacia

SimpleBarrier cez Event OPTIM2

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          E.set()
5)  M.unlock()
6)  E.wait()
```

Mozeme dat E.clear()

do prvej serializacie?

SimpleBarrier cez Event OPTIM2

```
1)  M.lock()
2)
3)
4)      C += 1
5)      if C == N:
6)          E.set()
7)  M.unlock()
```

```
7)  E.wait()
```

SimpleBarrier cez Event OPTIM2

```
1)  M.lock()
2)      if C == 0:
3)          E.clear()
4)      C += 1
5)      if C == N:
6)          E.set()
7)  M.unlock()
```

```
7)  E.wait()
```

SimpleBarrier cez Event OPTIM2

```
1)  M.lock()
2)      if C == 0:
3)          E.clear()
4)      C += 1
5)      if C == N:
6)          E.set()
7)  M.unlock()
```

```
7)  E.wait()
```

Musi iba jedno vlakno vykonavat E.clear()?

SimpleBarrier cez Event OPTIM3

```
1) M.lock()
2)     E.clear()
3)     C += 1
4)     if C == N:
5)         E.set()
6) M.unlock()
7) E.wait()
```

SimpleBarrier cez Event OPTIM3

```
1) M.lock()
2)       E.clear()
3)       C += 1
4)       if C == N:
5)             E.set()
6) M.unlock()
7) E.wait()
```

Pozor!

Zavisi od implementacie metody E.clear()!

V akom vztahu je ku metode E.wait()?

ReusableBarrier

1) b1()
2) b2()

Init(N):

b1 ← SimpleBarrier(N)

b2 ← SimpleBarrier(N)

Fibonacci

- N prvkov postupnosti (mimo! prvych 2)
  - fib_seq := {0, 1, 0, 0, …, 0}
  - len(fib_seq) is N+2
- Prvok postupnosti i:
  - fib_seq[i] = fib_seq[i-1] + fib_seq[i-2]

Fibonacci naïve

0, 1, x1, x2, x3, x4, …, x_n

x1 – vlakno necaka

x2 – vlakno caka na thr(x1)

x3 – caka sa na thr(x1) a thr(x2)

x4 – caka sa na thr(x2) a thr(x3)

…

x_n – caka na thr(x_n-1) a thr(x_n-2)

Fibonacci naïve

```
0, 1, x1, x2, x3, x4, …, x_n
T1:    --
T2:        T1
T3:        T1   T2
T4:             T2   T3
T5:                  T3   T4
T6:                       T4   T5
```

Fibonacci naïve

- Vidime, ze kazde vlakno, okrem prvych 2, caka vzdy na 2 predosle vlakna
- Vytvorme teda pre kazde vlakno (okrem prvych 2) synchronizacne objekty, napr. Semafory
- Kazde vlakno musi spravit (okrem prvych 2) 2x wait nad svojim semaforom, aby mohlo ist

Fibonacci naïve

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * N

Thread_i(id):

1) sem[id].wait()
2) sem[id].wait()
3) fib_seq[id+2] = fib_seq[id+1] + fib_seq[id]

Fibonacci naïve

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * N

Thread_i(id):

1) sem[id].wait()
2) sem[id].wait()
3) fib_seq[id+2] = fib_seq[id+1] + fib_seq[id]

Fibonacci naïve

Thread_i(id):

1) sem[id].wait()
2) sem[id].wait()
3) fib_seq[id+2] = fib_seq[id+1] + fib_seq[id]
4) sem[id+1].signal()
5) sem[id+2].signal()

Init(N):

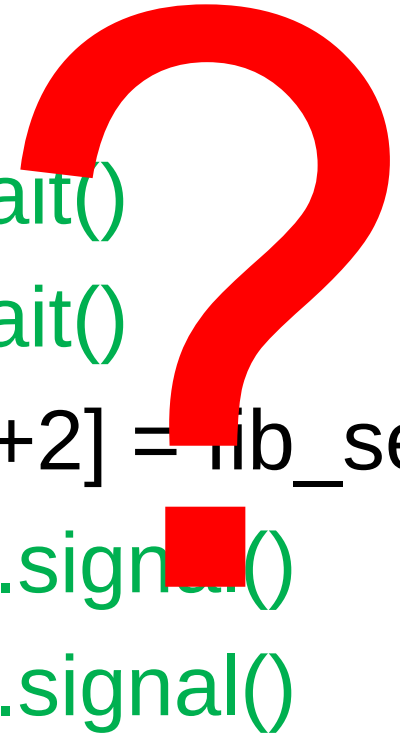
fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * N

Fibonacci naïve

Thread_i(id):

1) sem[id].wait()
2) sem[id].wait()
3) fib_seq[id+2] = fib_seq[id+1] + fib_seq[id]
4) sem[id+1].signal()
5) sem[id+2].signal()



Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * N

Fibonacci naïve

Thread_i(id):

1) sem[id].wait()
2) sem[id].wait()
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5) sem[id+2].signal()

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * N

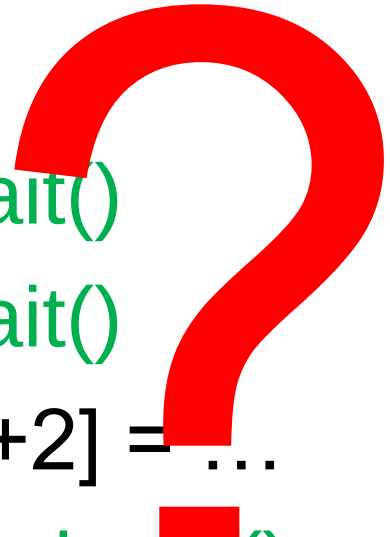
sem[0].signal(2)

sem[1].signal(1)

Fibonacci naïve

Thread_i(id):

1) sem[id].wait()
2) sem[id].wait()
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5) sem[id+2].signal()



Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * N

sem[0].signal(2)

sem[1].signal(1)

Fibonacci naïve

Thread_i(id):

1) sem[id].wait()
2) sem[id].wait()
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5) sem[id+2].signal()

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * N

sem[0].signal(2)

sem[1].signal(1)

Fibonacci naïve

Thread_i(id):

1) sem[id].wait()
2) sem[id].wait()
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5) sem[id+2].signal()

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * N

sem[0].signal(2)

sem[1].signal(1)

Fibonacci naïve

Thread_i(id):

1) sem[id].wait()
2) sem[id].wait()
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5) sem[id+2].signal()

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+2)

sem[0].signal(2)

sem[1].signal(1)

Fibonacci naïve

Thread_i(id):

1) sem[id].wait()
2) sem[id].wait()
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5) sem[id+2].signal()

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+2)

sem[0].signal(2)

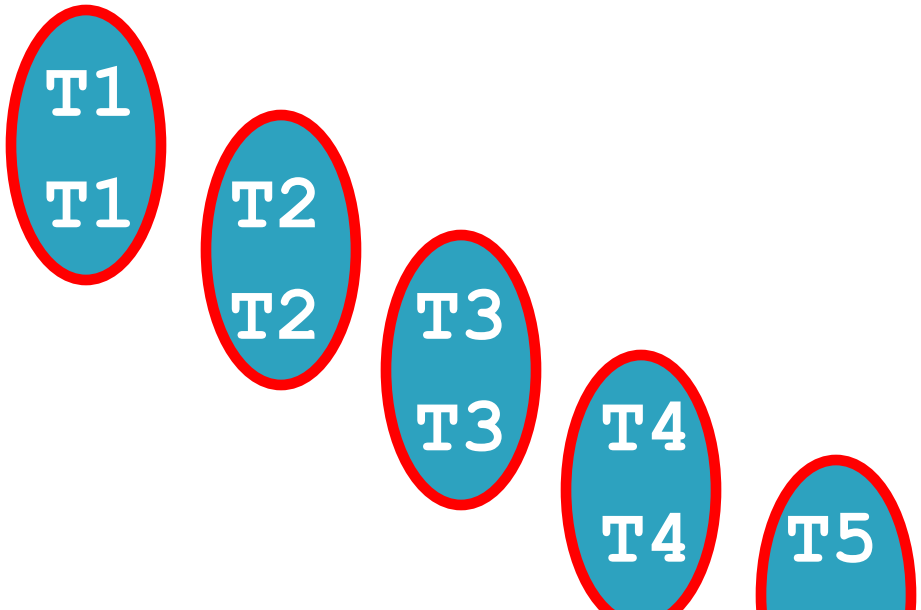
sem[1].signal(1)

Fibonacci naïve v2

```
0, 1, x1, x2, x3, x4, …, x_n
T1:    --
T2:         T1
T3:         T1   T2
T4:              T2   T3
T5:                   T3   T4
T6:                        T4   T5
```

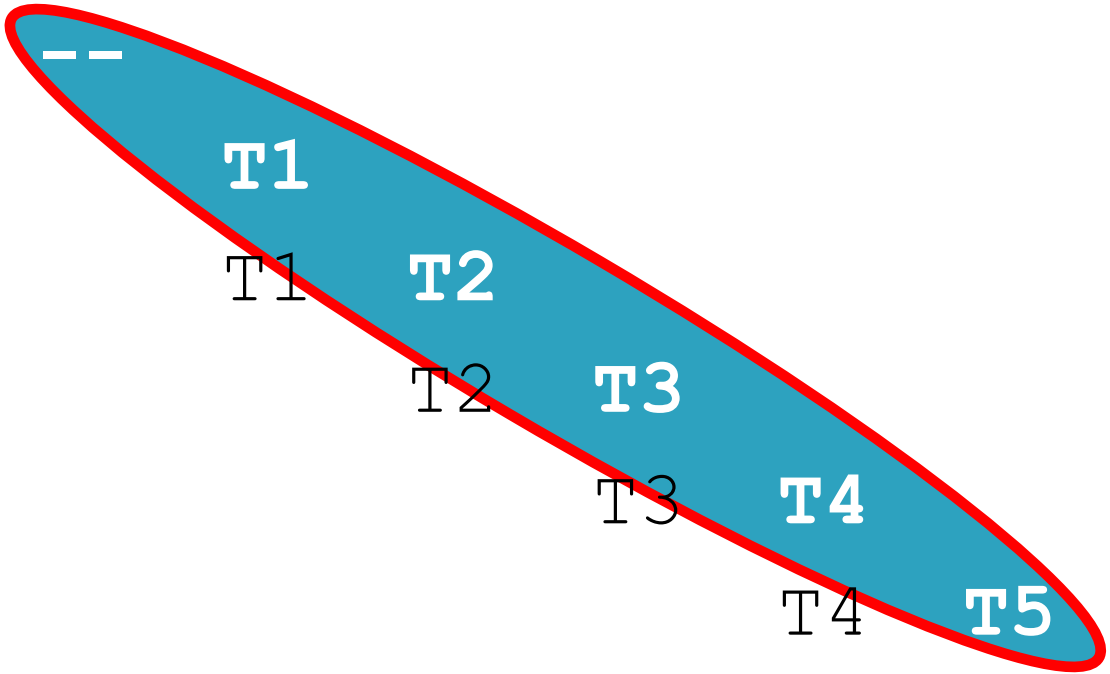
Fibonacci naïve v2

```
0, 1, x1, x2, x3, x4, …, x_n
T1:    --
T2:
T3:
T4:
T5:
T6:
```



Fibonacci naïve v2

```
0, 1, x1, x2, x3, x4, …, x_n
T1:    --
T2:        T1
T3:           T1   T2
T4:                T2   T3
T5:                     T3   T4
T6:                          T4   T5
```



Fibonacci naïve v2

```
0, 1, x1, x2, x3, x4, …, x_n
T1:    --
T2:        T1
T3:        T1   T2
T4:             T2   T3
T5:                  T3   T4
T6:                       T4   T5
```

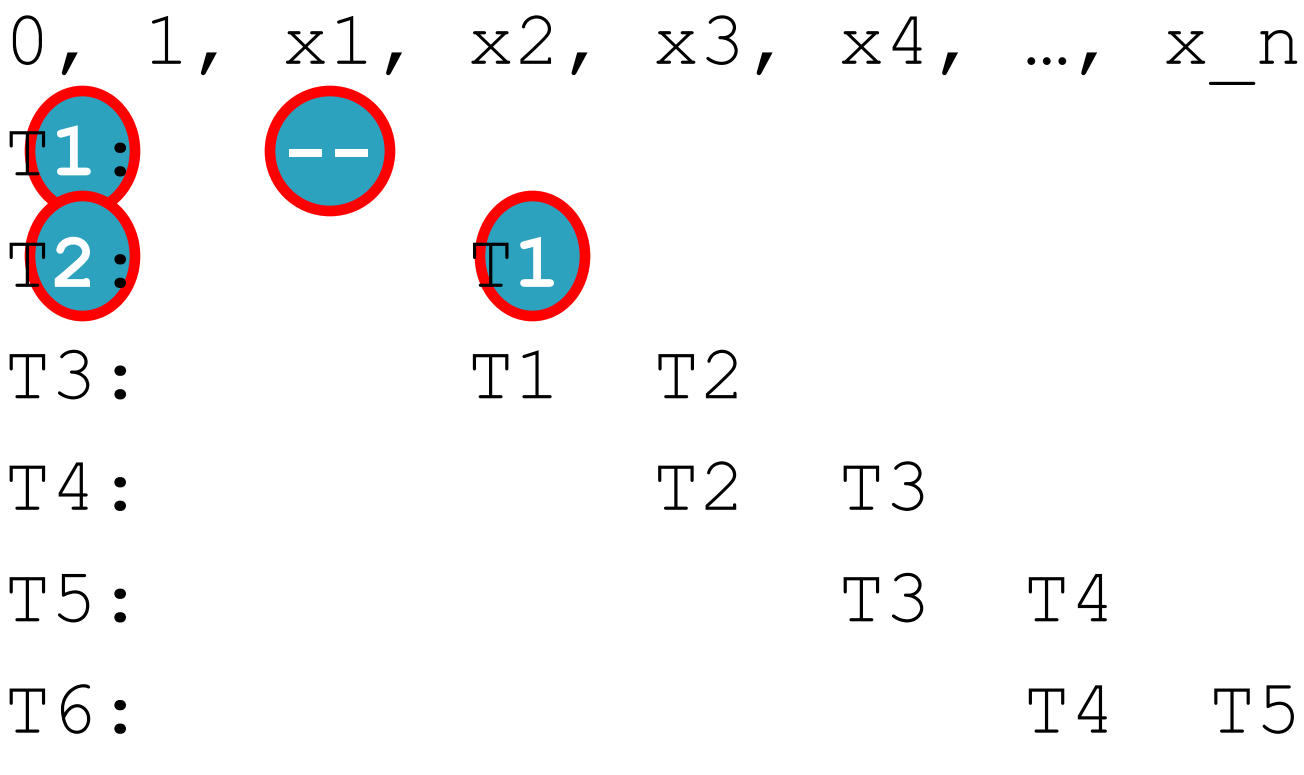
Fibonacci naïve v2

```
0, 1, x1, x2, x3, x4, …, x_n
T1:    --
T2:          T1
T3:          T1   T2
T4:               T2   T3
T5:                    T3   T4
T6:                         T4   T5
```

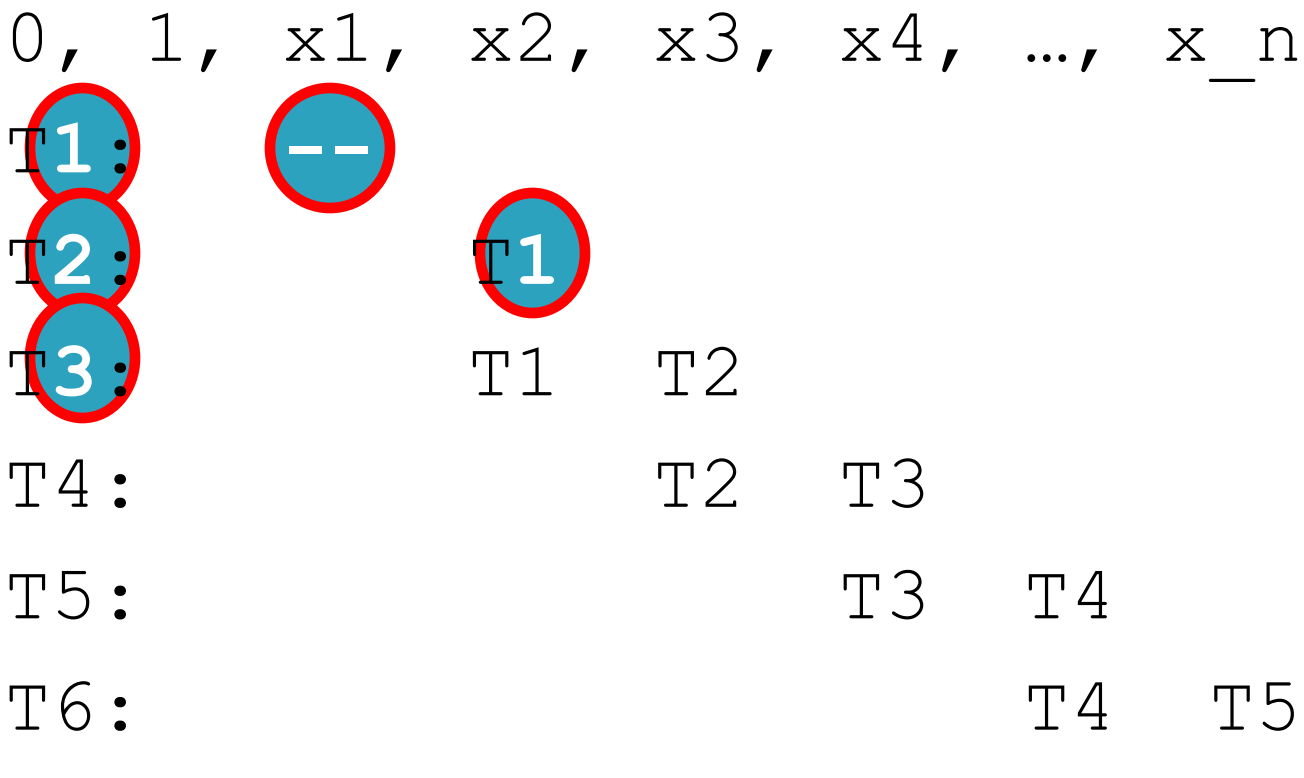
Fibonacci naïve v2

```
0, 1, x1, x2, x3, x4, …, x_n
T1:    --
T2:         T1
T3:         T1   T2
T4:              T2   T3
T5:                   T3   T4
T6:                        T4   T5
```

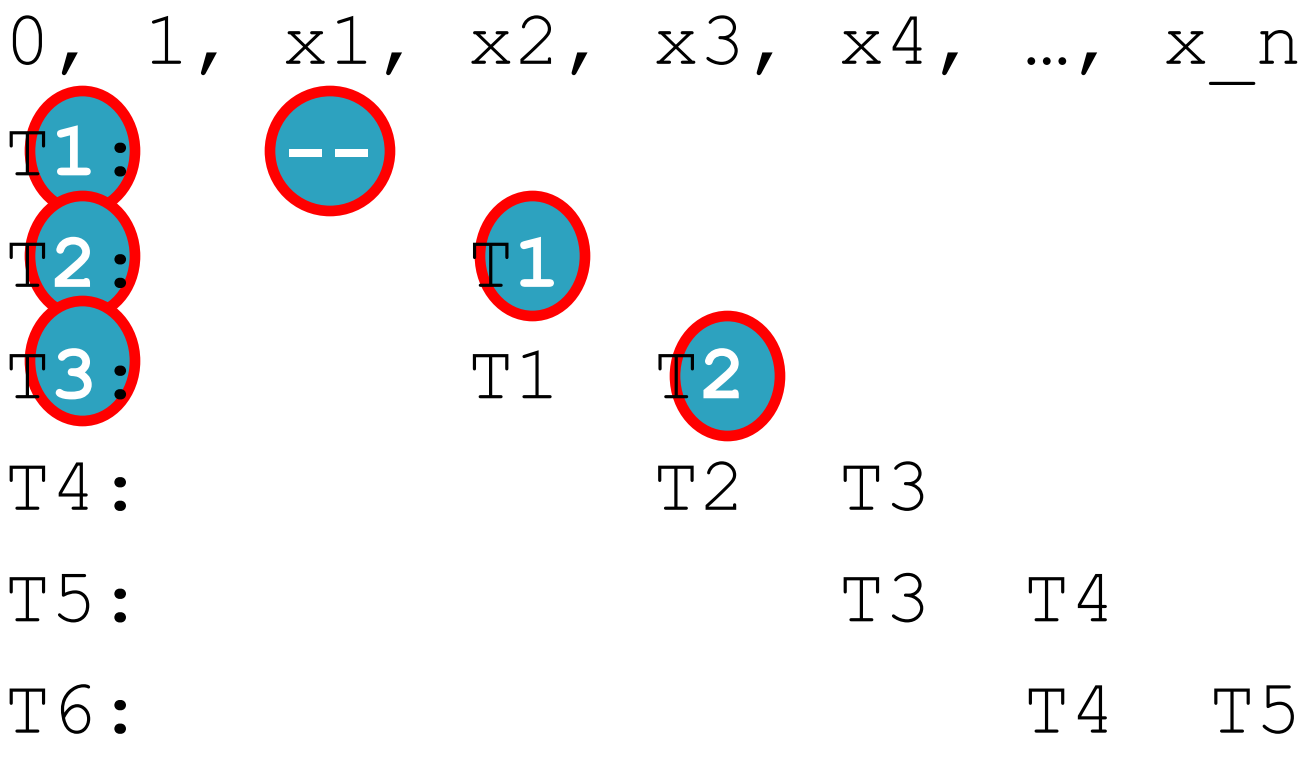
Fibonacci naïve v2



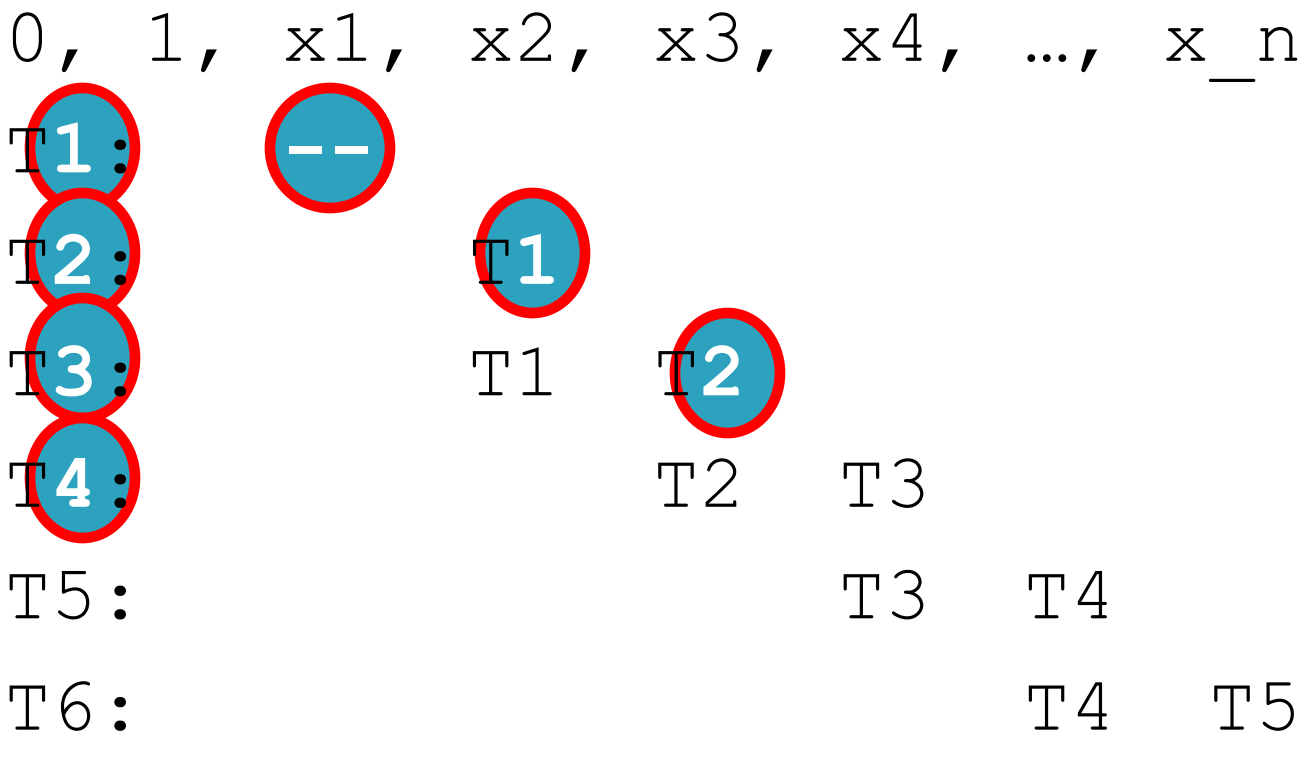
Fibonacci naïve v2



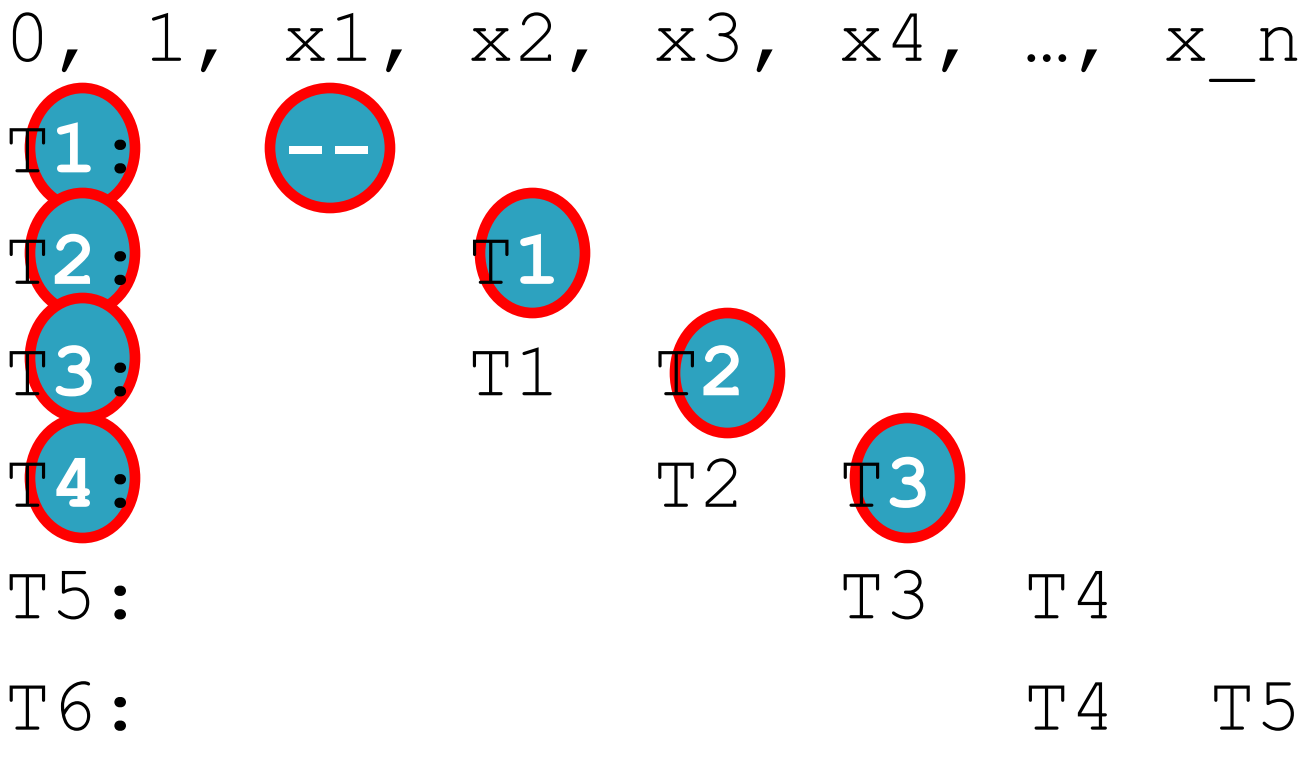
Fibonacci naïve v2



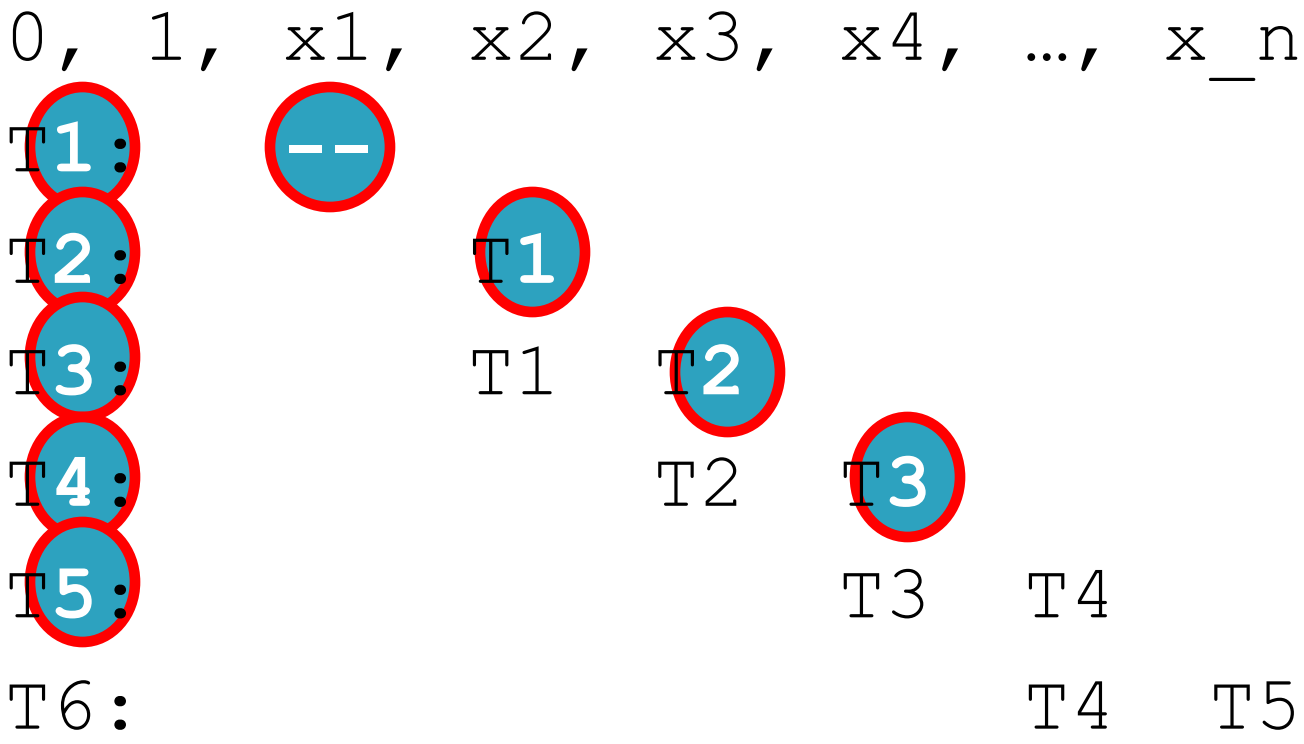
Fibonacci naïve v2



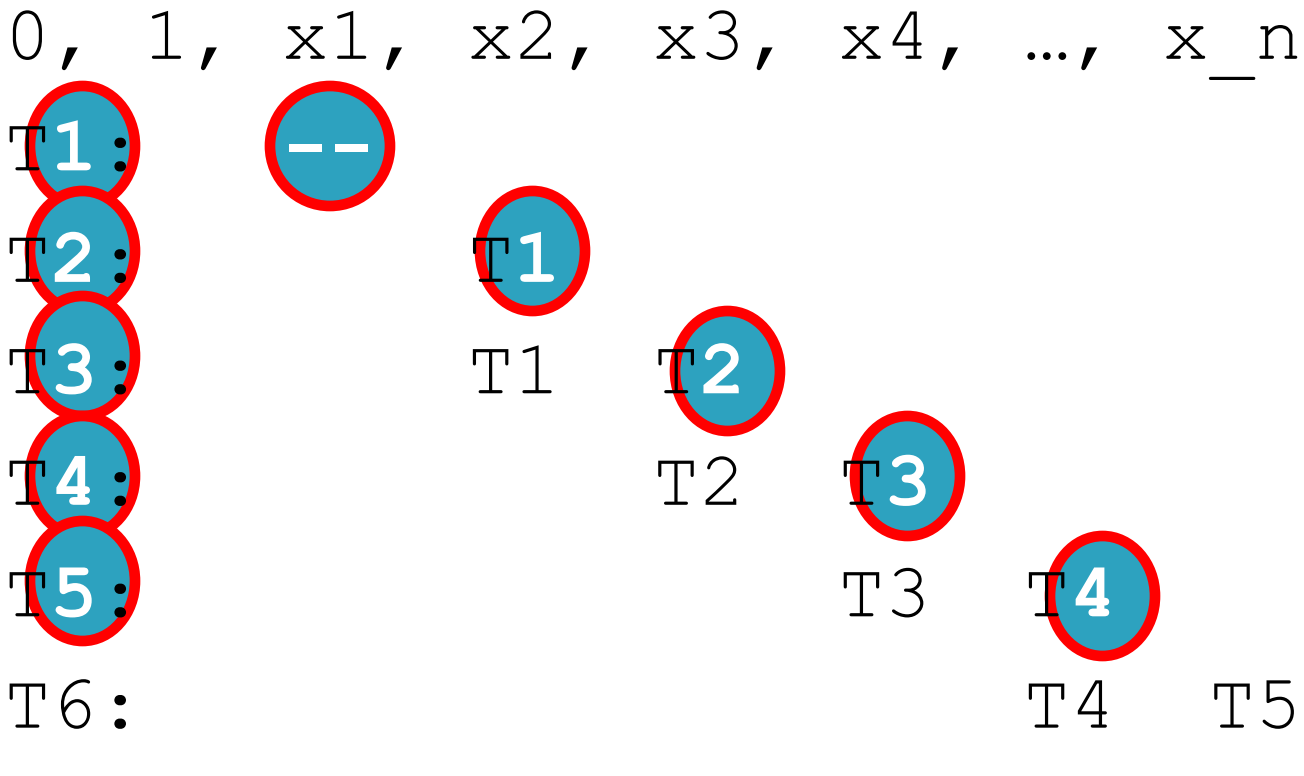
Fibonacci naïve v2



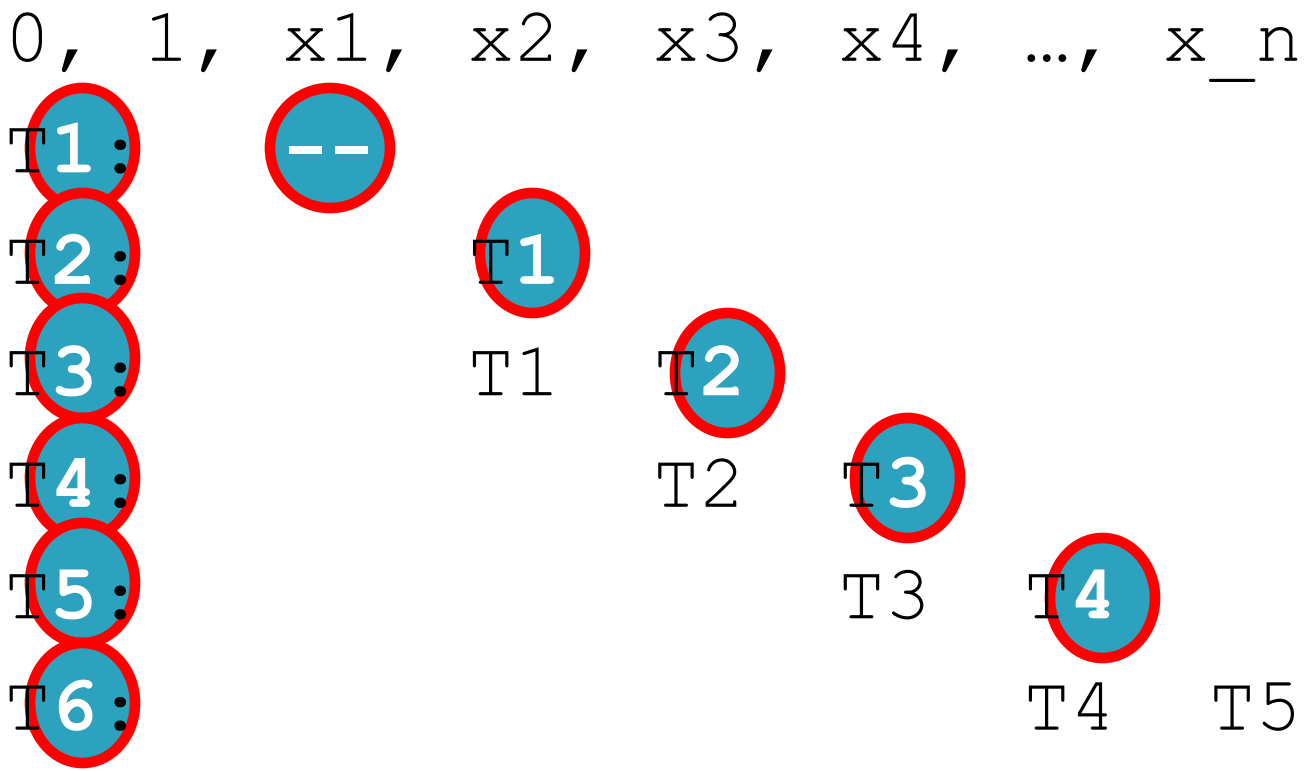
Fibonacci naïve v2



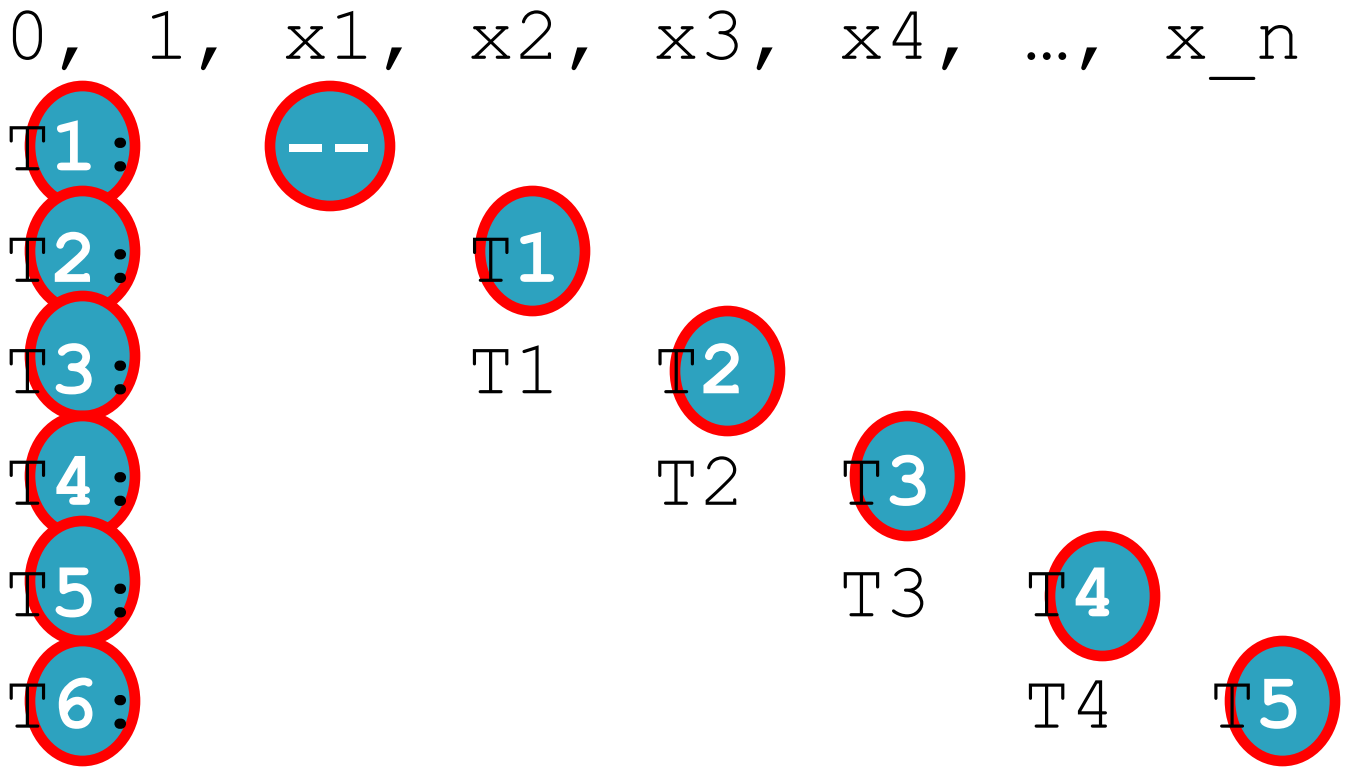
Fibonacci naïve v2



Fibonacci naïve v2



Fibonacci naïve v2



Fibonacci naïve v2

Thread_i(id):

1) sem[id].wait()
2) sem[id].wait()
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5) sem[id+2].signal()

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+2)

sem[0].signal(2)

sem[1].signal(1)

Fibonacci naïve v2

Thread_i(id):

1) sem[id].wait()
2)
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5) sem[id+2].signal()

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+2)

sem[0].signal(2)

sem[1].signal(1)

Fibonacci naïve v2

Thread_i(id):

1) sem[id].wait()
2)
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5)

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+2)

sem[0].signal(2)

sem[1].signal(1)

Fibonacci naïve v2

Thread_i(id):

1) sem[id].wait()
2)
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5)

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+2)

sem[0].signal(2)

Fibonacci naïve v2

Thread_i(id):

1) sem[id].wait()
2)
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5)

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+2)

sem[0].signal(2)

Fibonacci naïve v2

Thread_i(id):

1) sem[id].wait()
2)
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5)

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+2)

sem[0].signal(1)

Fibonacci naïve v2

Thread_i(id):

1) sem[id].wait()
2)
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5)

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+2)

sem[0].signal(1)

Fibonacci naïve v2

Thread_i(id):

1) sem[id].wait()
2)
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5)

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+2)

sem[0].signal(1)

Fibonacci naïve v2

Thread_i(id):

1) sem[id].wait()
2)
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5)

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+1)

sem[0].signal(1)

Fibonacci naïve v2

Thread_i(id):

1) sem[id].wait()
2)
3) fib_seq[id+2] = …
4) sem[id+1].signal()
5)

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+1)

sem[0].signal(1)

Fibonacci naïve v2

Thread_i(id):

1) sem[id].wait()
2) fib_seq[id+2] = …
3) sem[id+1].signal()

Init(N):

fib_seq ← [0,1]+[0]*N

sem ← Sem(0) * (N+1)

sem[0].signal(1)

Fibonacci alla S. Podhorec

- Hm… kolko mame synchronizacnych objektov?
- Da sa to na menej nez cca N?

Fibonacci alla S. Podhorec

```
0, 1, x1, x2, x3, x4, …, x_n
T1:    --
T2:        T1
T3:        T1   T2
T4:             T2   T3
T5:                  T3   T4
T6:                       T4   T5
```

Fibonacci alla S. Podhorec

```
   0, 1, x1, x2, x3, x4, x5, x6, …, xn
E1 s1 s2 s3
E2:          s1  s2  s3
E3:                  s1  s2 …
…
```

Fibonacci alla S. Podhorec

```
E1: s1 s2 s3
E2: s1 s2 s3
E3: s1 s2 s3
…
```

Fibonacci alla S. Podhorec

- Kolko synchronizacnych objektov potrebujeme?
- P semaforov, Q udalosti (napriklad)
- Ako urcime ich pocty?

Fibonacci alla S. Podhorec

- Potrebujeme, aby P*Q = N
- Takze si urcime cisla nasledovne:
  - If N%P == 0: Q = N / P
  - Else: Q = N / P + 1
- Vieme optimalizovat P, Q tak, aby sme pouzili co najmenej synchronizacnych objektov?

Fibonacci alla S. Podhorec

- Snad ano, ked sme zvladli Matematiku 1 ;)

Fibonacci alla S. Podhorec

- Snad ano, ked sme zvladli Matematiku 1 ;)
- Urobme si parcialnu derivaciu…

Fibonacci alla S. Podhorec

- Potrebujeme rozlozit N na sucin cisiel P a Q tak, aby P+Q bolo co najmensie
- Algoritmus:
  - for P in range(1, round(N**0.5)+1):
  - Q := N // P
  - if P*Q != N:
  - Q += 1
  - print(“%d + %d = %d”, P, Q, P+Q)

Fibonacci

- Hm… kolko mame teraz synchronizacnych objektov?
- Aka je zlozitost tohto riesenia vzhladom na pocet synchronizacnych objektov voci poctu vlakien, ktore sa maju synchronizovat?
- Da sa to na menej nez 2*N^(1/2)?

Fibonacci

```
0, 1, x1, x2, x3, x4, …, x_n
T1:    --
T2:         T1
T3:         T1   T2
T4:              T2   T3
T5:                   T3   T4
T6:                        T4   T5
```

Fibonacci

- Co potrebujeme dosiahnut?

- Synchronizacny vzor **SERIALIZACIA**
- T1 < T2 < T3 < T4 < … < T_n

Fibonacci

- Co potrebujeme dosiahnut?

- Synchronizacny vzor **SERIALIZACIA**
- T1 < T2 < T3 < T4 < … < T_n
  - Ak sa vykonava T4, T1 aj T2 aj T3 su uz vykonane!
  - Vseobecne plati, ze T_i < T_j, j>i

Fibonacci

```
0, 1, x1, x2, x3, x4, …, x_n
T1:    --
T2:         T1
T3:         T1   T2
T4:              T2   T3
T5:                   T3   T4
T6:                        T4   T5
```

Fibonacci s udalostou

- Problem serializacie znamena, ze:
  - T1 < T2, T2 < T3, T3 < T4, …
  - T1 < T2 < T3 < T4 < …
  - V 1 case iba 1 vlakno je aktivne!!!

Fibonacci s udalostou

1) fib_seq[id+2] = …

Init(N):

fib_seq ← [0,1]+[0]*N

Fibonacci s udalostou

1) e.wait()

2) fib_seq[id+2] = …

Init(N):

fib_seq ← [0,1]+[0]*N

Fibonacci s udalostou

1) e.wait()

2) fib_seq[id+2] = …

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

Fibonacci s udalostou

1) e.wait()

2) fib_seq[id+2] = …

3) e.signal()

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()

3)       fib_seq[id+2] = …

4)       e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()

3)      fib_seq[id+2] = …

4)      cnt += 1

5)      e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()

3)      fib_seq[id+2] = …

4)      cnt += 1

5)      e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

cnt ← 0

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()

3)          if cnt == thr_id: break

4)       fib_seq[id+2] = …

5)       cnt += 1

6)       e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

cnt ← 0

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()
3)          m.lock()
4)          if cnt == thr_id: break
5)          m.unlock()



6)       fib_seq[id+2] = …



7)       cnt += 1


8)       e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

cnt ← 0

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()
3)          m.lock()
4)          if cnt == thr_id: break
5)          m.unlock()

6)       fib_seq[id+2] = …

7)       cnt += 1

8)       e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

m ← Mutex()

cnt ← 0

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()
3)          m.lock()
4)          if cnt == thr_id: break
5)          m.unlock()
6)      m.unlock()


7)      fib_seq[id+2] = …


8)      cnt += 1

9)      e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

m ← Mutex()

cnt ← 0

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()
3)          m.lock()
4)          if cnt == thr_id: break
5)          m.unlock()
6)      m.unlock()



7)      fib_seq[id+2] = …


8)      m.lock()
9)      cnt += 1
10)     m.unlock()
11)     e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

m ← Mutex()

cnt ← 0

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()
3)          m.lock()
4)          if cnt == thr_id: break
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()
12)     e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

m ← Mutex()

cnt ← 0

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()
3)          m.lock()
4)          if cnt == thr_id: break
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()
12)     e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

m ← Mutex()

cnt ← 0

Fibonacci s udalostou

```
1)    while True:
2)        e.wait()
3)        m.lock()
4)        if cnt == thr_id: break
5)        m.unlock()
6)    m.unlock()

7)    e.clear()
8)    fib_seq[id+2] = …

9)    m.lock()
10)   cnt += 1
11)   m.unlock()
12)   e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

m ← Mutex()

cnt ← 0

- wait() – clear() – signal()

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # nech vsetky vlakna pokracuju z tejto fnc dalej
3)          m.lock()
4)          if cnt == thr_id: break
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                # nech vsetky vlakna pokracuju z tejto fnc dalej
3)          m.lock()                # vsetky su medzi R2-R3, iba jedno ide na R3 (vlakno Ti)
4)          if cnt == thr_id: break
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # nech vsetky vlakna pokracuju z tejto fnc dalej
3)          m.lock()                    # vsetky okrem Ti su medzi R2-R3
4)          if cnt == thr_id: break     # nech pre vlakno Ti plati podmienka, takze ide na R6
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # nech vsetky vlakna pokracuju z tejto fnc dalej
3)          m.lock()                    # vsetky okrem Ti su medzi R2-R3
4)          if cnt == thr_id: break     # nech pre vlakno Ti plati podmienka
5)          m.unlock()
6)      m.unlock()

7)      e.clear()               # Ti spravi clear(), stale mame ostatne vlakna medzi R2-R3
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # nech vsetky vlakna pokracuju z tejto fnc dalej
3)          m.lock()                    # postupne vsetky okrem Ti a Ti+1 idu dalej
4)          if cnt == thr_id: break     # nech pre vlakno Ti plati podmienka
5)          m.unlock()
6)      m.unlock()

7)      e.clear()               # Ti spravilo clear()!
8)      fib_seq[id+2] = …       # medzicasom nech vsetky vlakna okrem Ti+1 pokracuju zo stavu medzi
                                # R2-R3 dalej, takze sa znovu v cykle dostanu na e.wait()
9)      m.lock()
10)     cnt += 1
11)     m.unlock()
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()                    # vlakno Ti+1 je stale medzi R2-R3
4)          if cnt == thr_id: break
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1                # Ti spravi zvysenie pocitadla, vsetky okrem Ti+1 su na wait()
11)     m.unlock()
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()                    # vlakno Ti+1 je stale medzi R2-R3
4)          if cnt == thr_id: break     # podmienka uz neplati pre Ti, ale pre Ti+1!
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1                # Ti urobilo aktualizaciu podmienky
11)     m.unlock()
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()                    # vlakno Ti+1 je stale medzi R2-R3
4)          if cnt == thr_id: break     # podmienka uz neplati pre Ti, ale pre Ti+1!
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1                # Ti urobilo aktualizaciu podmienky
11)     m.unlock()              # Ti urobi unlock(), ale na R12 zatial nepokracuje!
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()                    # vlakno Ti+1 je stale medzi R2-R3
4)          if cnt == thr_id: break     # podmienka uz neplati pre Ti, ale pre Ti+1!
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()              # Ti urobilo unlock(), ale na R12 zatial nepokracuje
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()                    # vlakno Ti+1 pokracuje dalej
4)          if cnt == thr_id: break     #
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()              # Ti urobilo unlock(), ale na R12 zatial nepokracuje
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()                    #
4)          if cnt == thr_id: break     # podmienka pre Ti+1 je splnena, ide na R6
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()              # Ti urobilo unlock(), ale na R12 zatial nepokracuje
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()                    #
4)          if cnt == thr_id: break     #
5)          m.unlock()
6)      m.unlock()

7)      e.clear()               # po R6 moze vykonat R7
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()              # Ti urobilo unlock(), ale na R12 zatial nepokracuje
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()                    #
4)          if cnt == thr_id: break     #
5)          m.unlock()
6)      m.unlock()

7)      e.clear()               # po vykonani R7 mame problem!
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()              # Ti urobilo unlock(), ale na R12 zatial nepokracuje
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()                    #
4)          if cnt == thr_id: break     #
5)          m.unlock()
6)      m.unlock()

7)      e.clear()               # vlakno Ti+1 urobilo clear(), ale vlakno Ti nestihlo urobit signal()!
8)      fib_seq[id+2] = …  #

9)      m.lock()
10)     cnt += 1
11)     m.unlock()              # Ti urobilo unlock(), ale na R12 zatial nepokracuje
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()                    #
4)          if cnt == thr_id: break     #
5)          m.unlock()
6)      m.unlock()

7)      e.clear()               # chceli sme zachovat v kazdom cykle postupnost
8)      fib_seq[id+2] = …  # pred kazdym signal() urobit clear()!!!

9)      m.lock()
10)     cnt += 1
11)     m.unlock()              # Ti urobilo unlock(), ale na R12 zatial nepokracuje
12)     e.signal()
```

Fibonacci s udalostou

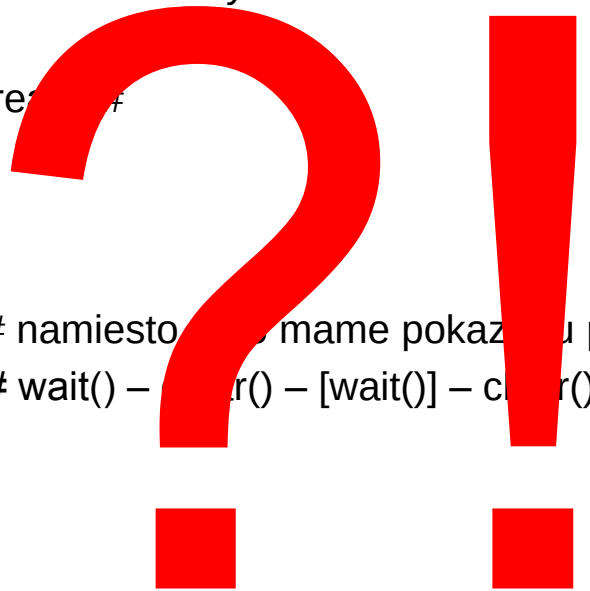
```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()                    #
4)          if cnt == thr_id: break     #
5)          m.unlock()
6)      m.unlock()

7)      e.clear()               # wait() – clear() – signal() --- [wait()] – clear() – signal() …
8)      fib_seq[id+2] = …  # … [wait()] – clear() – signal() --- [wait()] – clear() – signal() …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()              # Ti urobilo unlock(), ale na R12 zatial nepokracuje
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()                    #
4)          if cnt == thr_id: break     #
5)          m.unlock()
6)      m.unlock()

7)      e.clear()               # namiesto toho mame pokazenu postupnost
8)      fib_seq[id+2] = …  # wait() – clear() – [wait()] – clear() – signal() – signal() …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()              # Ti urobilo unlock(), ale na R12 zatial nepokracuje
12)     e.signal()
```

Fibonacci s udalostou

```
1)      while True:
2)          e.wait()                    # vsetky okrem Ti a Ti+1 cakaju vo funkcii wait()
3)          m.lock()
4)          if cnt == thr_id: bre
5)          m.unlock()
6)      m.unlock()

7)      e.clear()               # namiesto  mame pokaz  postupnost
8)      fib_seq[id+2] = …  # wait() –  r() – [wait()] – c  r() – signal() – signal() …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()              # Ti urobilo unlock(), ale na R12 zatial nepokracuje
12)     e.signal()
```



Fibonacci s udalostou hardened

```
1)      while True:
2)          e.wait()
3)          m.lock()
4)          if cnt == thr_id: break
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()
12)     e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

m ← Mutex()

cnt ← 0

Fibonacci s udalostou hardened

```
1)      while True:
2)          e.wait()
3)          m.lock()
4)          if cnt == thr_id: break
5)          m.unlock()
6)      m.unlock()

7)      e.clear()
8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()
12)     e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

m ← Mutex()

cnt ← 0

b← R_barrier_ex(N, e.clear())

Fibonacci s udalostou hardened

```
1)      while True:
2)          e.wait()
3)          b.wait(N-cnt)
4)          m.lock()
5)          if cnt == thr_id: break
6)          m.unlock()
7)      m.unlock()

8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()
12)     e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

m ← Mutex()

cnt ← 0

b← R_barrier_ex(N, e.clear())

Fibonacci s udalostou hardened

```
1)      while True:
2)          e.wait()
3)          b.wait(N-cnt)
4)          m.lock()
5)          if cnt == thr_id: break
6)          m.unlock()
7)      m.unlock()

8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()
12)     e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

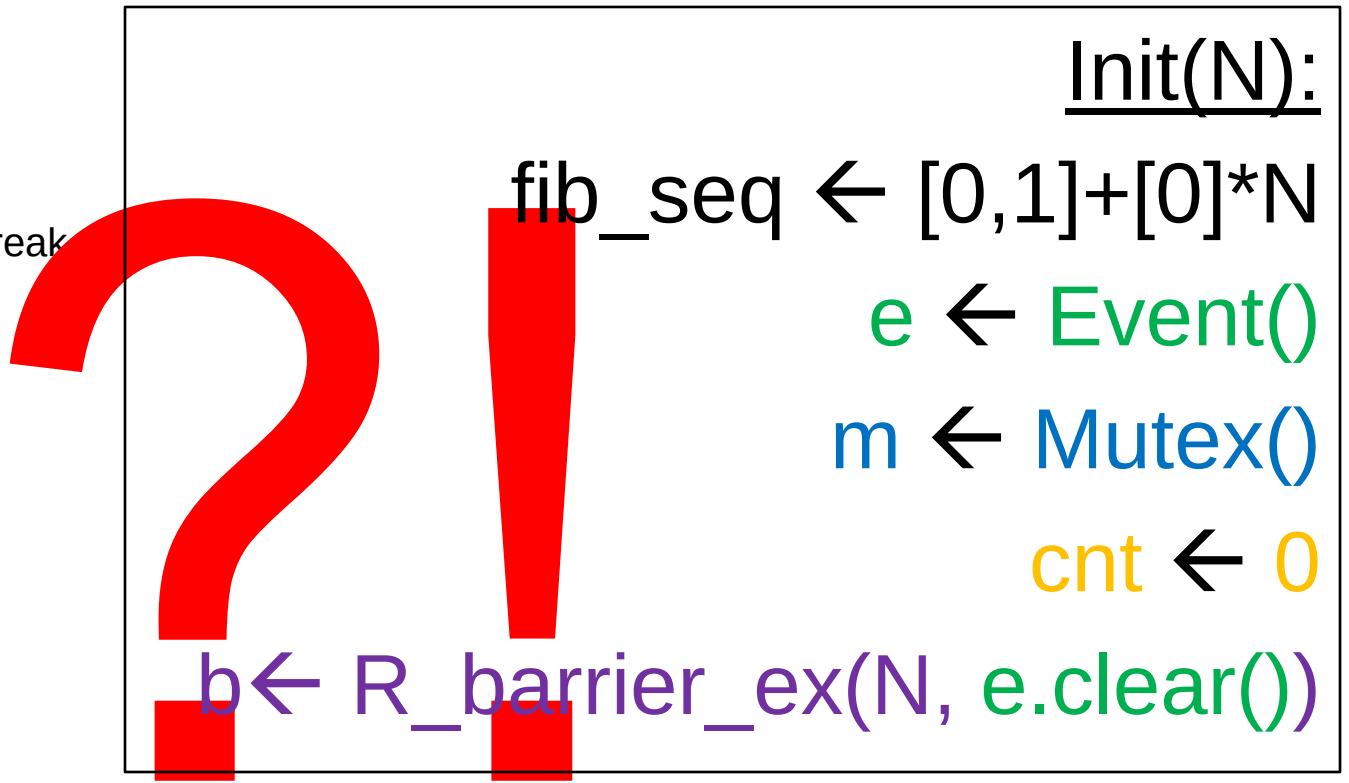
m ← Mutex()

cnt ← 0

b← R_barrier_ex(N, e.clear())

Fibonacci s udalostou hardened

```
1)      while True:
2)          e.wait()
3)          b.wait(N-cnt)
4)          m.lock()
5)          if cnt == thr_id: break
6)          m.unlock()
7)      m.unlock()

8)      fib_seq[id+2] = …

9)      m.lock()
10)     cnt += 1
11)     m.unlock()
12)     e.signal()
```



Fibonacci s udalostou hardened

```
1)    while True:
2)        e.wait()
3)        b.wait(N-cnt)
4)        m.lock()
5)        if cnt == this_id: break
6)        m.unlock()
7)    m.unlock()
8)    fib_seq[i+2] = ..
9)    m.lock()
10)   cnt += 1
11)   m.unlock()
12)   e.signal()
```

Init(N):

fib_seq ← [0,1]+[0]*N

e ← Event()

m ← Mutex()

cnt ← 0

b← R_barrier_ex(N, e.clear())

Riesi to nas problem???

Cvicenie

- Navrhnite ulohu Fib_seq pomocou synchronizacie

Cvicenie

- Navrhnite ulohu Fib_seq pomocou synchronizacie
  - s udalostou
  - s udalostou tak, aby bolo VZDY zabezpecene, ze pred kazdym signal() je prave jeden clear()

Cvicenie

- Navrhnite ulohu Fib_seq pomocou synchronizacie
  - s udalostou
  - s udalostou tak, aby bolo VZDY zabezpecene, ze pred kazdym signal() je prave jeden clear()
  - Vlakno, ktore vypocita svoje Fib. cislo, nech skonci!

Cvicenie

- Navrhnite ulohu Fib_seq pomocou synchronizacie
  - s udalostou
  - s udalostou OK clear() – signal()
- Odhalte (pripadne opravte) chyby navrhu

Cvicenie

- Navrhnite ulohu Fib_seq pomocou synchronizacie
  - s udalostou
  - s udalostou OK clear() – signal()
- Odhalte (pripadne opravte) chyby navrhu
- Implementujte

Cvicenie

- Navrhnite ulohu Fib_seq pomocou synchronizacie
  - s udalostou
  - s udalostou OK clear() – signal()
- Odhalte (pripadne opravte) chyby navrhu
- Implementujte
- Navrhnite a realizujte overenie spravnosti

Cvicenie

- Navrhnite ulohu Fib_seq pomocou synchronizacie
  - s udalostou
  - s udalostou OK clear() – signal()
- Odhalte (pripadne opravte) chyby navrhu
- Implementujte
- Navrhnite a realizujte **overenie** spravnosti