PPaDS MMXXI

Matúš Jókay, C-503, matus.jokay@stuba.sk

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

konzultácie elektronicky

mail, discord

Opakovanie…

- Este raz bariera
- SimpleBarrier cez Event
- Fibonacci

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):

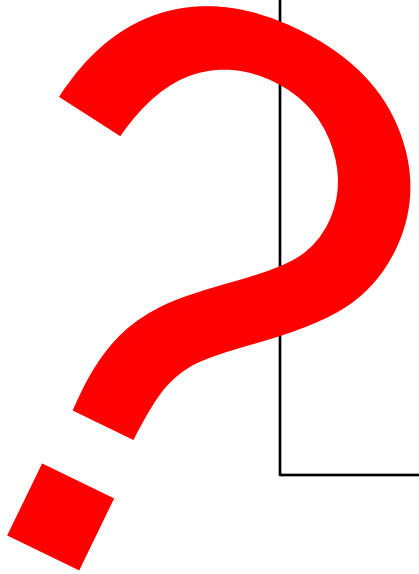
C ← 0

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()
8)  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()
8)  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()
```

```
8)  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):

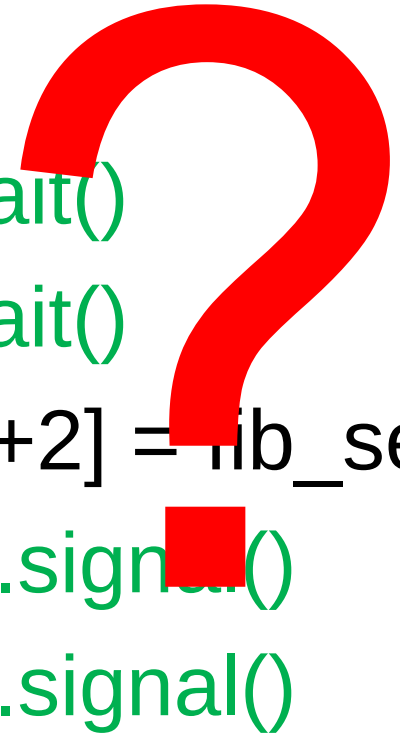
fib_seq ← [0,1]+[0]*N

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

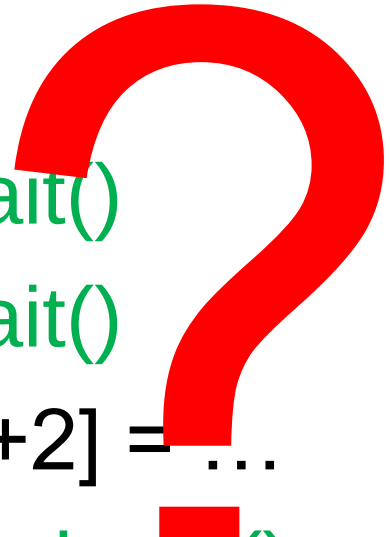
sem[0].signal(2)

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)

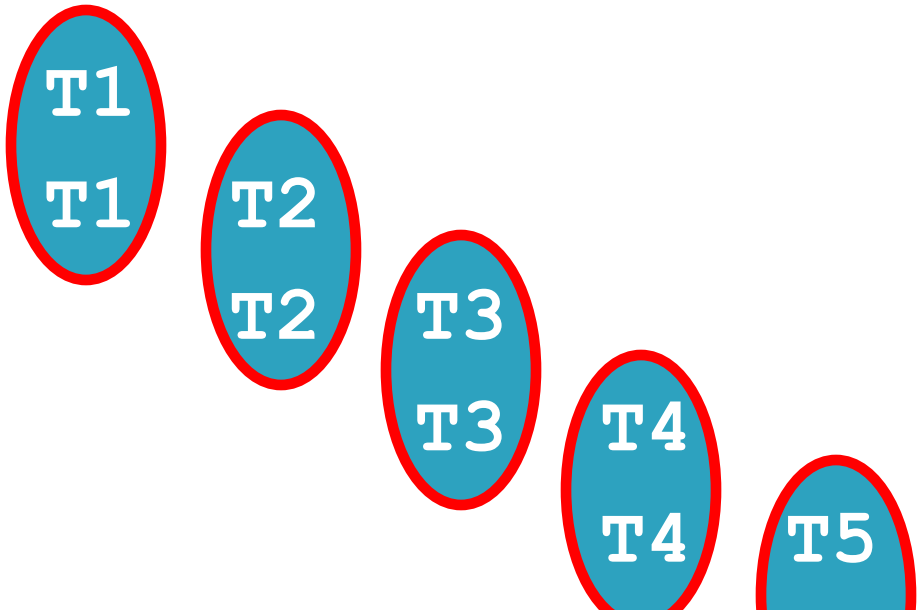
sem[1].signal(1)

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
```

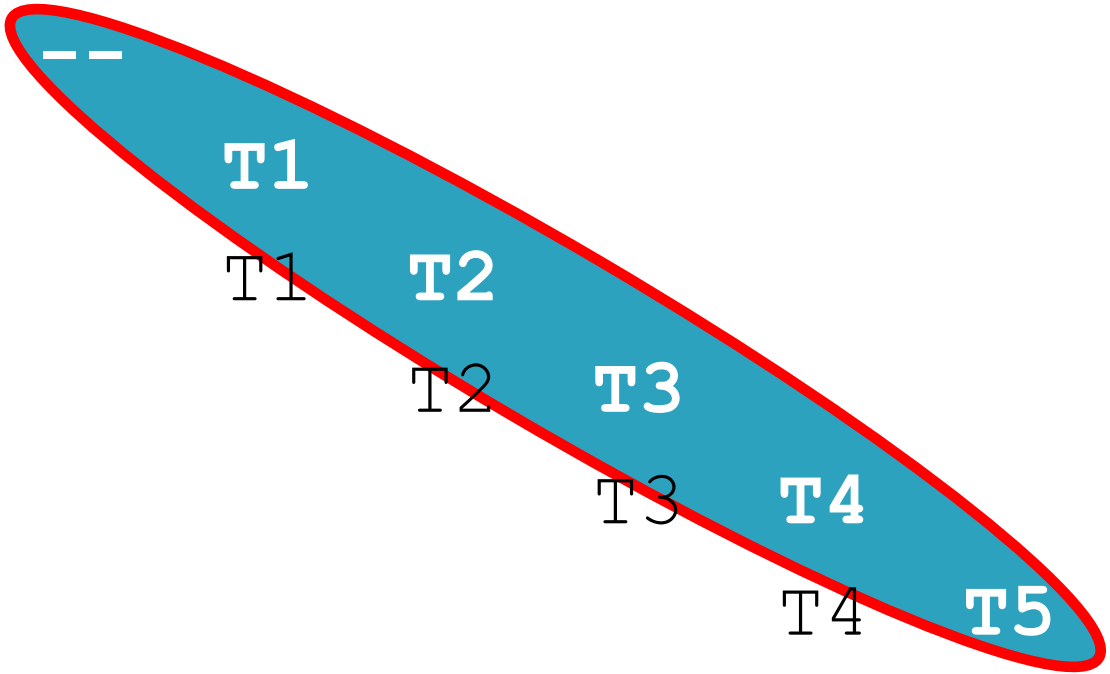
Fibonacci naïve v2

```
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
```

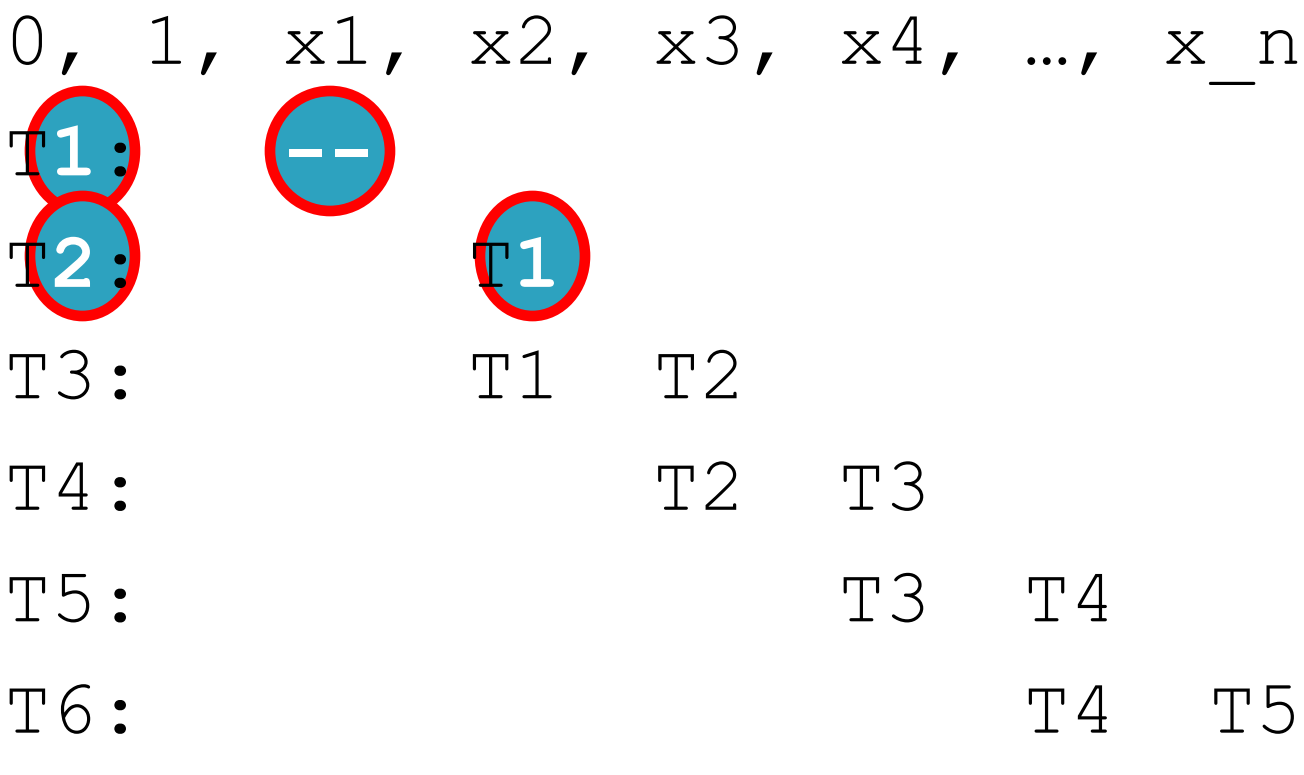
Fibonacci naïve v2

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

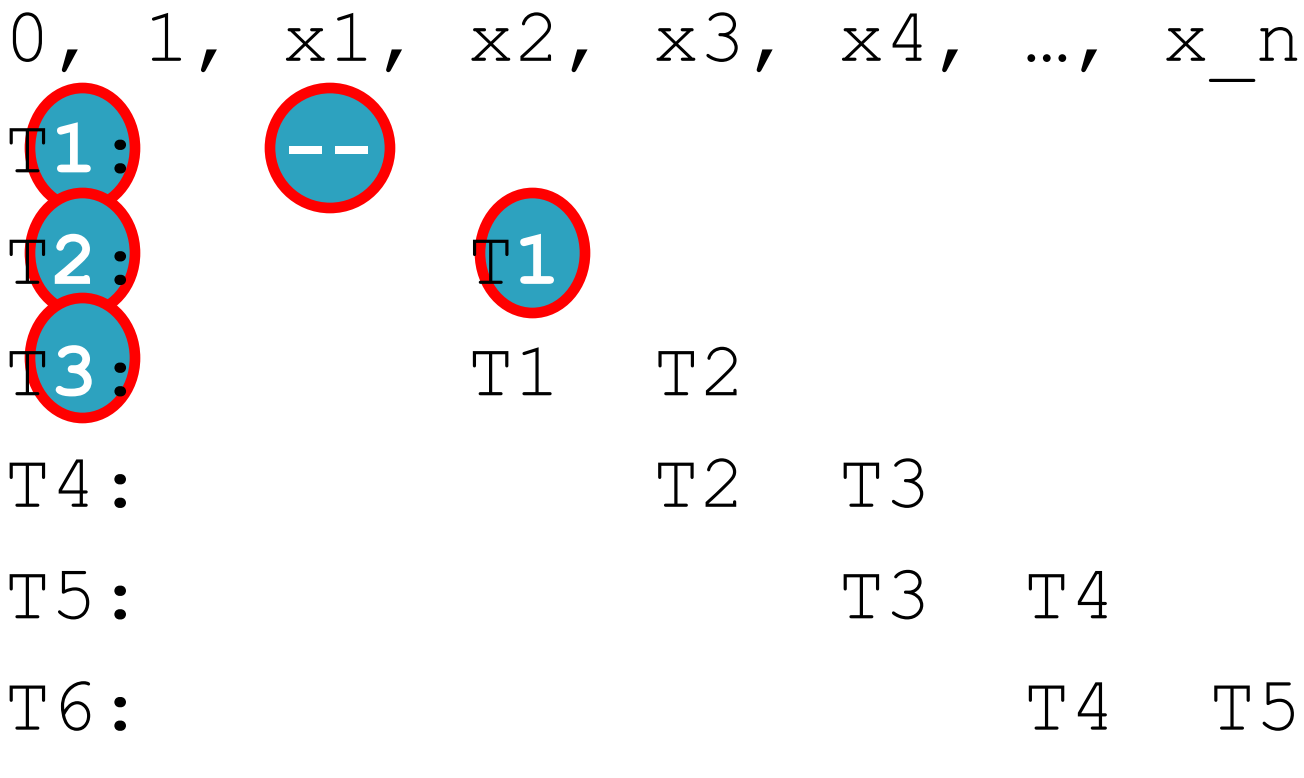
Fibonacci naïve v2

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

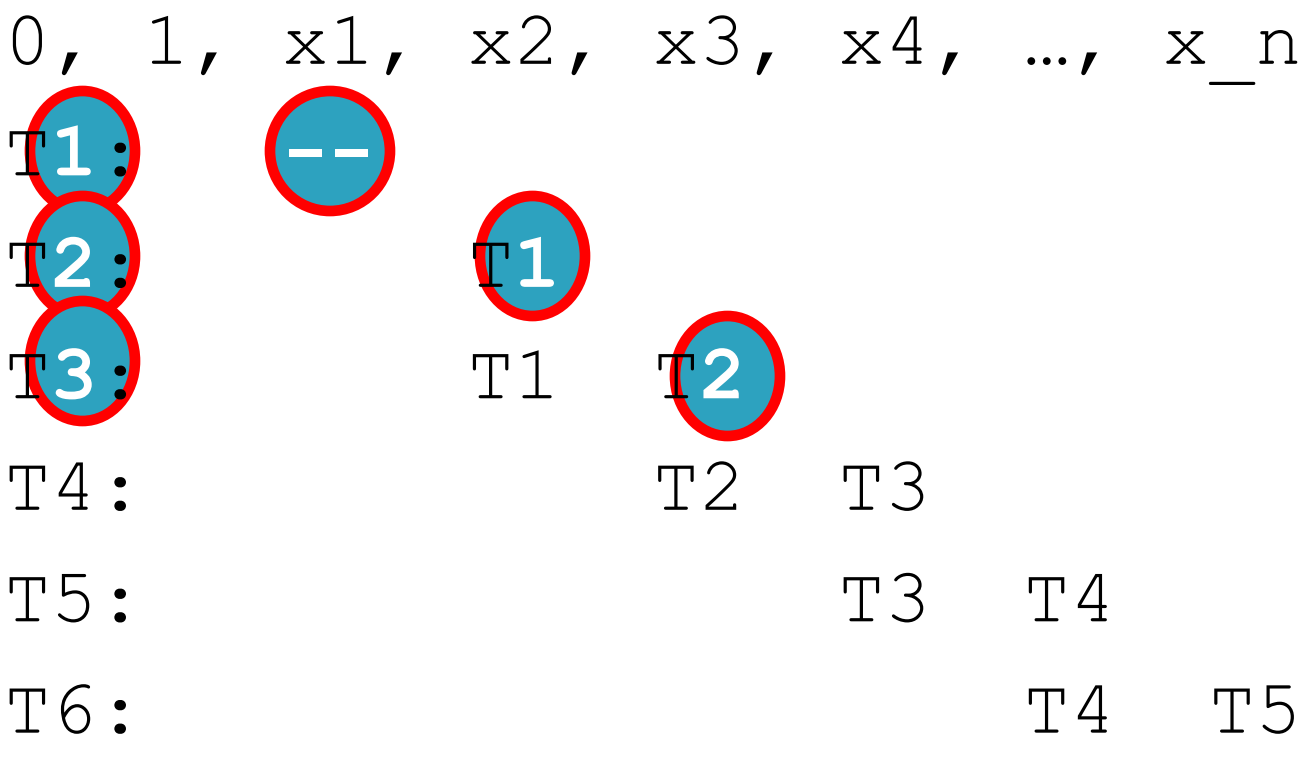
Fibonacci naïve v2



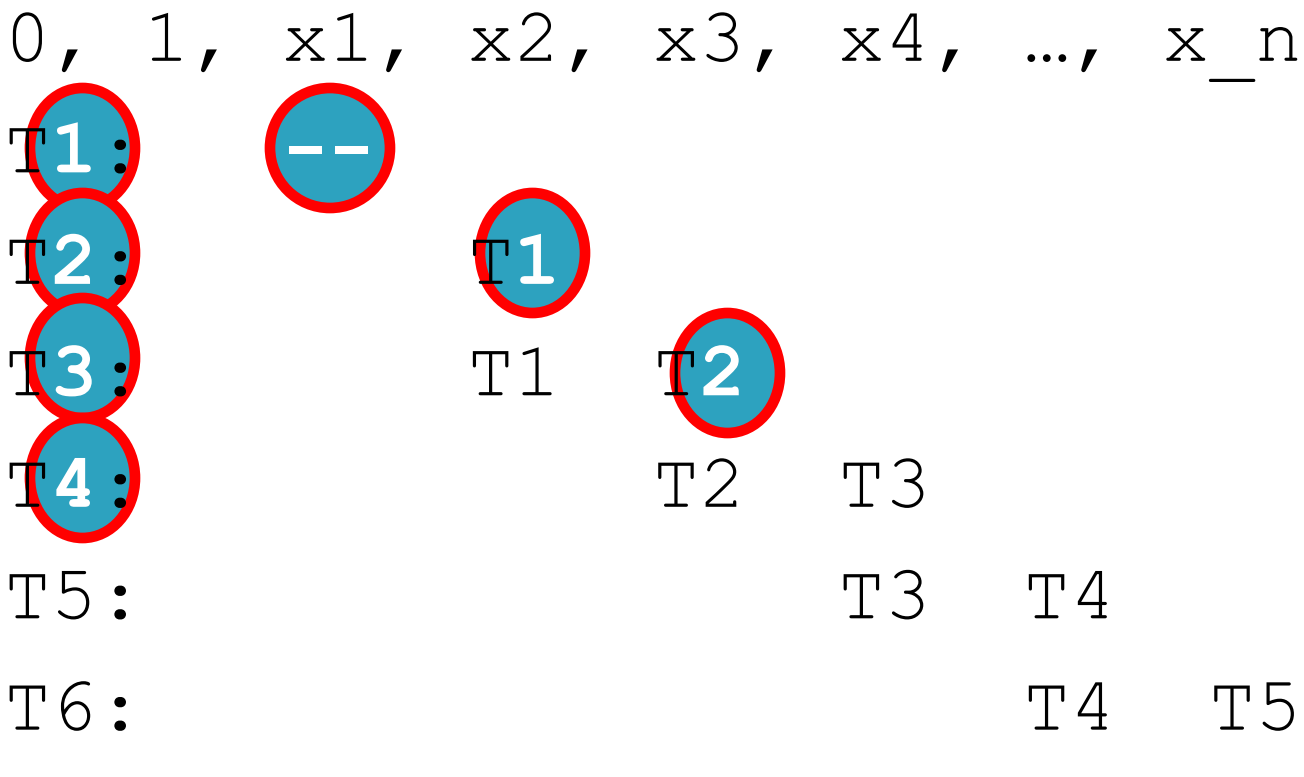
Fibonacci naïve v2



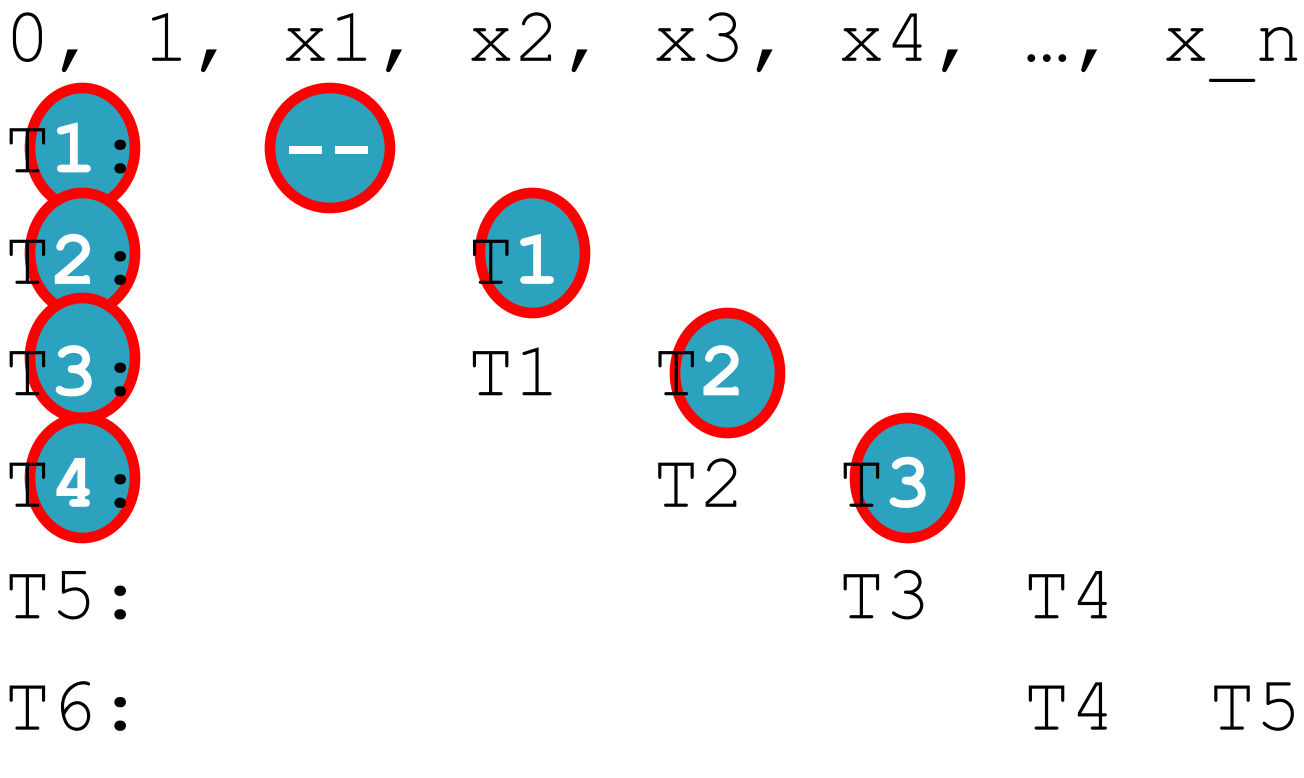
Fibonacci naïve v2



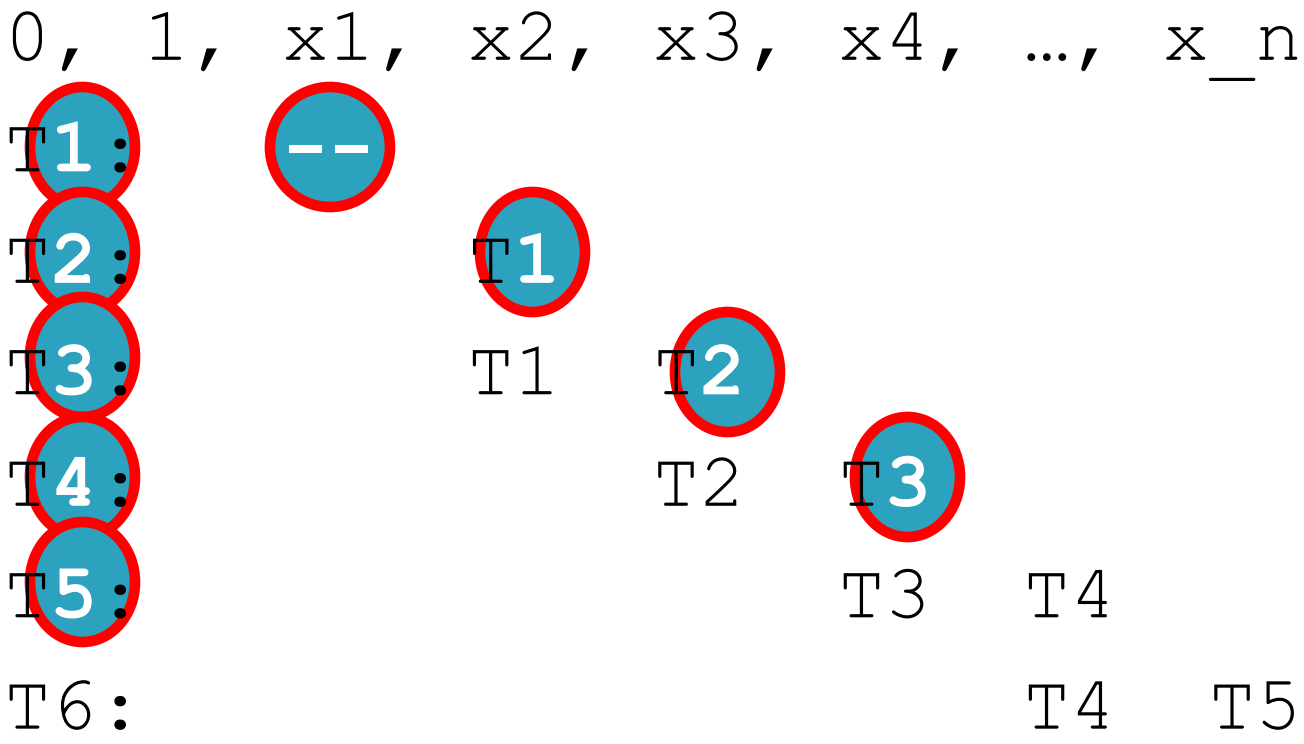
Fibonacci naïve v2



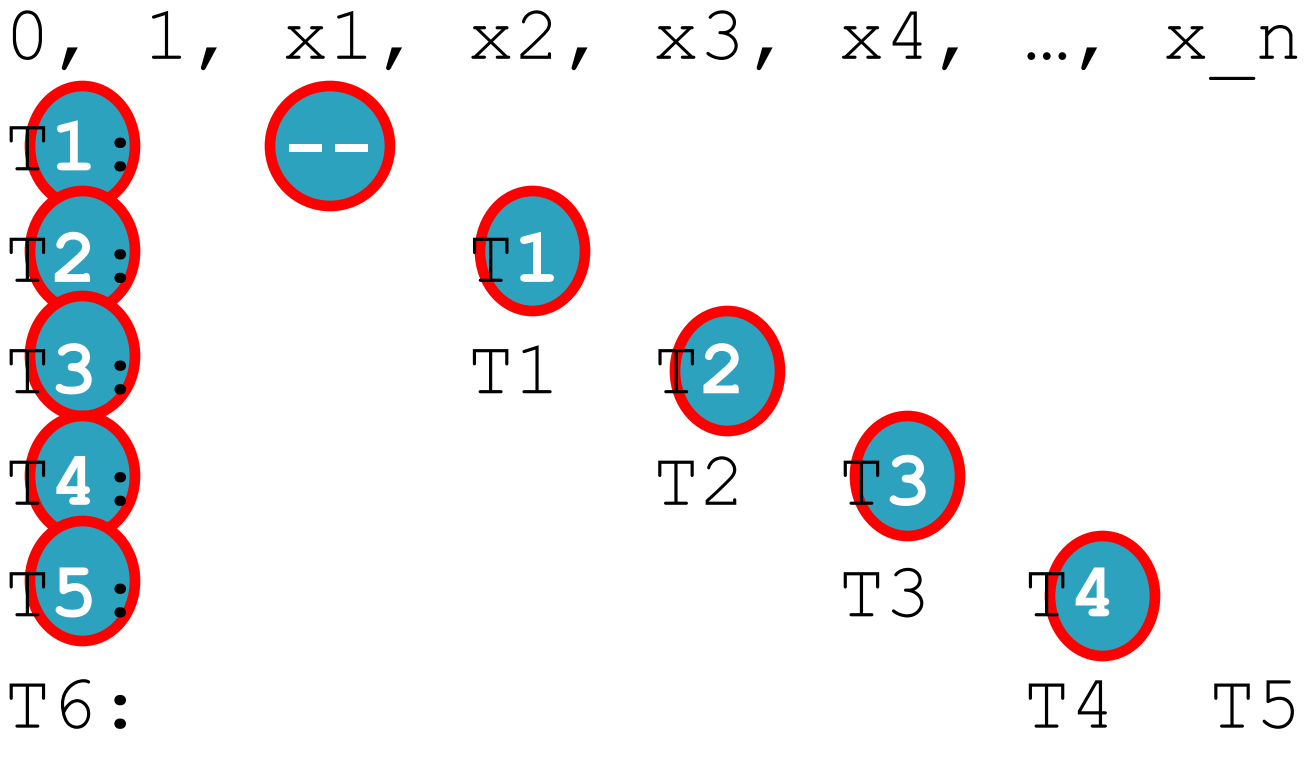
Fibonacci naïve v2



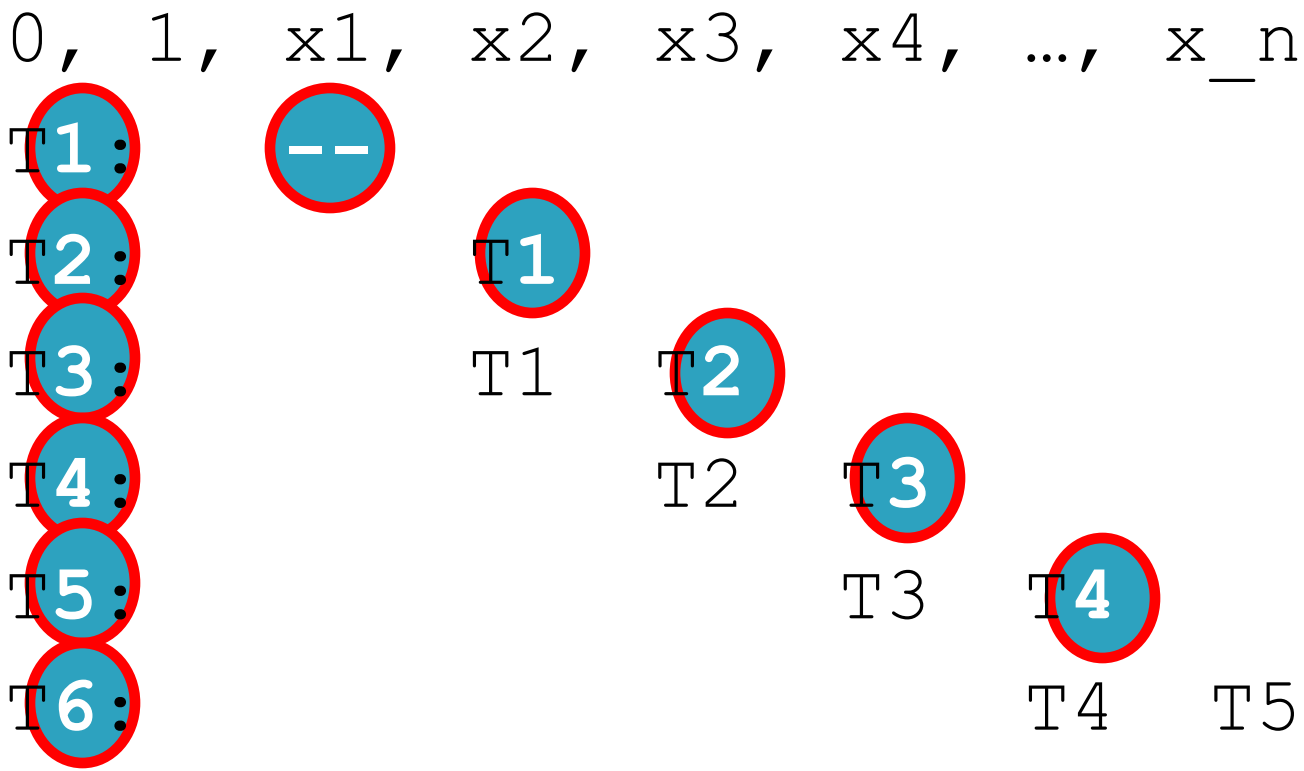
Fibonacci naïve v2



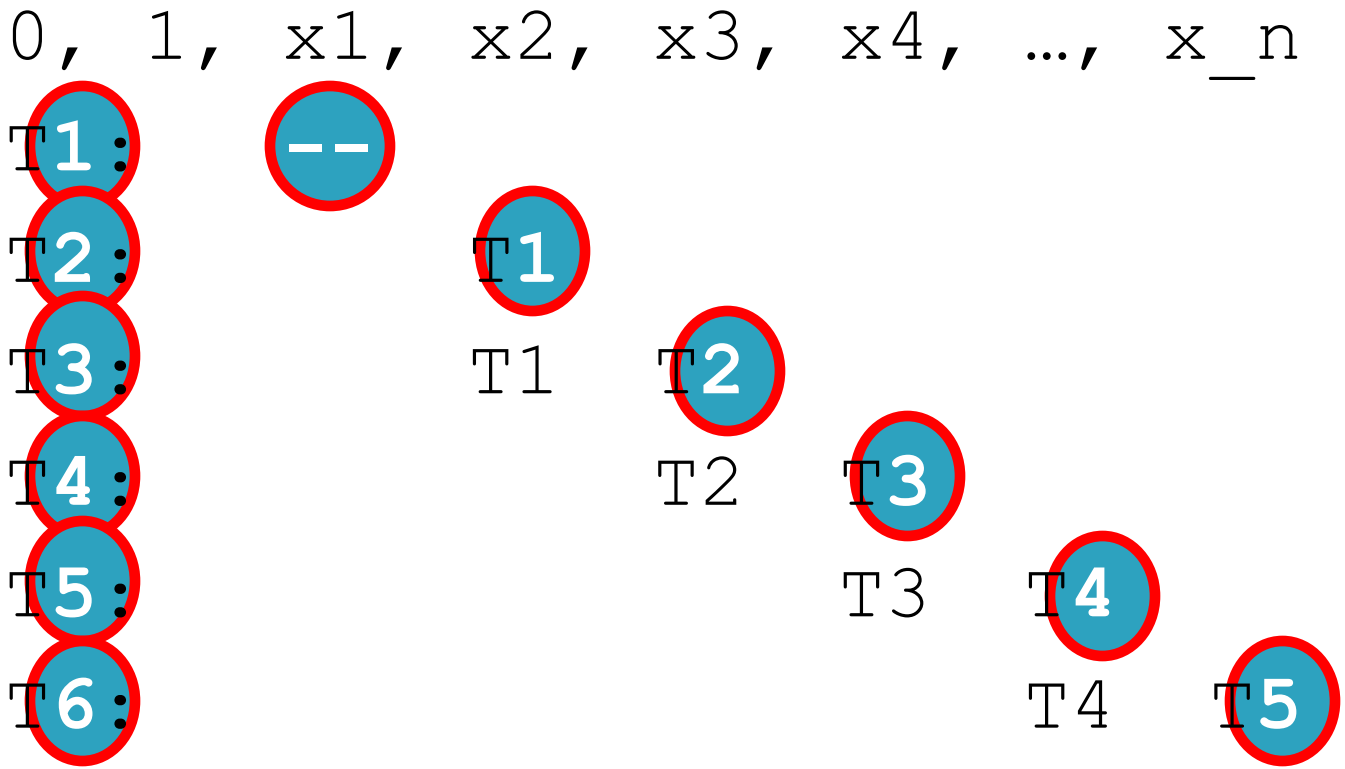
Fibonacci naïve v2



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
- Kód:

```
for P in range(1, round(N**0.5)+1):
    Q := N // P
    if P*Q != N:
        Q += 1
    print(f“{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

- 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

- Doteraz sme riesili tzv. "pasivne" cakanie (angl. sleep-wait)
- Jestvuje aj iny pristup, "aktivne" cakanie (spin-wait)

Fibonacci spin

Thread_i(id):

1) while id+2 != len(fib_seq):
2) pass
3) m.lock()
4) fib_seq.append(…)
5) m.unlock()

Init(N):

fib_seq ← [0,1]

m ← Mutex()

Fibonacci spin

```
Thread_i(id):
1)  while id+2 != len(fib_seq):
2)      pass
3)  m.lock()
4)  fib_seq.append(…)
5)  m.unlock()
```

```
Thread_i():
1)  while index < len(arr):
2)      m.lock()
3)      arr[index] += 1
4)      index += 1
5)      m.unlock()
```

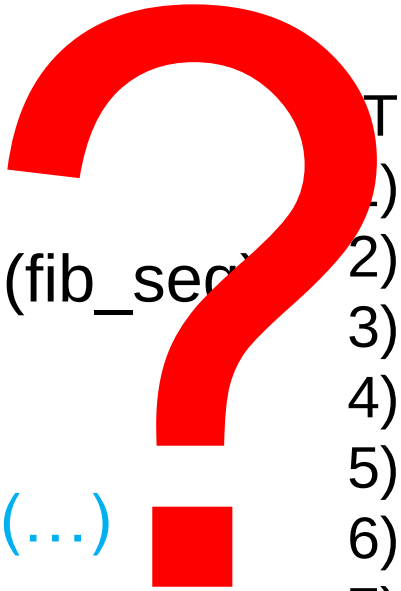
Fibonacci spin

```
Thread_i(id):
1)    while True:
2)        if id+2 == len(fib_seq):
3)            break
4)    m.lock()
5)    fib_seq.append(…)
6)    m.unlock()
```

```
Thread_i():
1)    while True:
2)        if index >= len(arr):
3)            break
4)        m.lock()
5)        arr[index] += 1
6)        index += 1
7)        m.unlock()
```

Fibonacci spin

```
Thread_i(id):
1)    while True:
2)        if id+2 == len(fib_seq):
3)            break
4)    m.lock()
5)    fib_seq.append(…)
6)    m.unlock()
```



```
Thread_i():
1)    while True:
2)        if index >= len(arr):
3)            break
4)        m.lock()
5)        arr[index] += 1
6)        index += 1
7)        m.unlock()
```

Fibonacci spin

```
Thread_i(id):
1)    while True:
2)        m.lock()
3)        if id+2 == len(fib_seq):
4)            break
5)        m.unlock()
6)    fib_seq.append(…)
7)    m.unlock()
```

```
Thread_i():
1)    while True:
2)        m.lock()
3)        if index >= len(arr):
4)            break
5)        arr[index] += 1
6)        index += 1
7)        m.unlock()
8)    m.unlock()
```