KONKURENTNÉ PROGRAMOVANIE

5. cvičenie: Koordinácia vlákien

Zadanie prenesené z minula

- Stiahnite si z Gitu poslednú verziu a balíček:
  - cviko04.zadanie

1. Zabezpečte súbežnú prácu úlohy Searcher a úlohy FileAnalyzer cez blokovaný rad
2. Zvýšte počet vlákien súbežne vykonávajúcich úlohu FileAnalyzer
3. **Zvýšte počet vlákien súbežne vykonávajúcich úlohu Searcher**

Riešenie

- Potrebujeme rad adresárov na prehľadávanie
- Spustíme N vlákien s úlohou Searcher
- Úloha Searcher končí, keď už žiaden adresár nebude prehľadávaný
  - Pozor na dočasne prázdny rad adresárov na prehľadávanie
  - AtomicInteger unprocessedDirs
- Aspoň jedna úloha Searcher vygeneruje N otrávených piluliek
- Na ukončenie potenciálne donekonečna čakajúcich úloh Searcher môžeme opäť využiť otrávené pilulky

java.util.concurrent

- **Konkurentné kolekcie**
  - **ConcurrentHashMap, ConcurrentSkipListMap, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet**
- **Rady**, dvojsmerné rady (zásobník a rad v jednom)
  - Implementácie BlockingQueue, BlockingDeque
    - **LinkedBlockingQueue, ArrayBlockingQueue**, SynchronousQueue, PriorityBlockingQueue, DelayQueue, LinkedTransferQueue, LinkedBlockingDeque
- **Synchronizéry**
  - Semaphore, **CountDownLatch**, CyclicBarrier, Phaser, Exchanger
- Exekútory

CountDownLatch

- Funguje ako brána, ktorá sa otvorí, keď sa dosiahne konečný stav
- Blokované čakanie na otvorenie:
  - countDownLatch.**await();**
- Štartovací stav: N
- Hocijaké vlákno môže zavolať dekrementáciu
  - countDownLatch.**countDown();**
- Konečný stav – hodnota 0 – brána sa otvorí

Bariéra - CyclicBarrier

- Podobné ako Latch
  - Otvorí sa, keď dôjde k bráne N vlákien
  - Po otvorení a prepustení čakajúcich vlákien sa brána zavrie a dá sa znova použiť
  - Latch – čakanie na udalosť – **latch.await()**
  - Bariéra – pozdržanie vlákien na koordináciu medzivýsledkov - **bariera.await()**
  - Môžeme definovať akciu po dobehnutí vlákien, po ktorej vykonaní, sa vlákna pustia cez bránu
- Použitie
  - Máme sériu problémov kde (i+1)-vý problém závisí na výsledku i-teho.
  - Riešenie každého problému sa realizuje v N vláknach

Phaser

- Vylepšená bariéra
- Každé otvorenie znamená začiatok ďalšej fázy
- V každej fáze sa môže meniť počet vlákien, na ktoré sa čaká
  - Inkrementácia počtu vlákien, na ktoré sa bude čakať v tejto fáze – **register()**
  - Dekrementácia počtu vlákien, na ktoré sa bude čakať v ďalšej fáze - **arriveAndDeregister()**

Phaser

- Dôjdenie k bráne
  - Ak čakám na ostatných - **arriveAndAwaitAdvance()**
  - Ak nečakám na ostatných (ostatní vedia, že som došiel) - **arrive()**
  - Ak nečakám na ostatných a končím - **arriveAndDeregister()**
  - Keď k bráne došlo toľko vlákien, na koľko sa čakalo, pustia sa čakajúce vlákna a začína ďalšia fáza
- Zistenie, koľkí už došli k bráne - **getArrivedParties()**
- Môžem čakať na ukončenie x-tej fázy (na mňa nik nečakal) – **awaitAdvance(x)**

Phaser

- Monitorovanie fázera
  - Počet vlákien registrovaných v tejto fáze - **getRegisteredParties()**
  - Počet vlákien, ktoré už došli k bráne - **getArrivedParties()**
  - Počet vlákien, na ktoré ešte treba čakať - **getUnarrivedParties()**
  - Poradie aktuálnej fázy - **getPhase()**

Exchanger

- Ak si dve vlákna chcú vymeniť objekty rovnakého typu
  - Na začiatku má prvé vlákno referenciu na prvý objekt a druhé vlákno na druhý objekt
  - Na konci má prvé vlákno referenciu na druhý objekt a druhé vlákno na prvý objekt
  - Výmena sa udeje naraz
- Použitie
  - Jeden pripravuje súvislý balíček objektov
  - Druhý spracúva súvislý balíček objektov a na konci vráti spracovaný, na základe ktorého prvý pripraví nový balíček
- Väčšinou vieme nahradiť s jednosmerným jednoprvkovým SynchronousQueue

Semaphore

- Kontroluje počet aktivít, ktoré môžu mať súčasne prístup k nejakému zdroju
  - počet zadaný v konštruktore
- Žiadosť o vstup - **acquire()**
- Vystúpenie – **release()**
- Použitie
  - Prístup k množine konektorov na databázu (dá sa aj cez ArrayBlockingQueue)
  - Obmedzenie počtu prvkov v kolekcii

Obmedzenie veľkosti množiny

```
public class BoundedHashSet<T> {
 private final Set<T> set;
 private final Semaphore sem;

 public BoundedHashSet(int bound) {
   this.set = Collections.synchronizedSet(new HashSet<T>());
   sem = new Semaphore(bound);
 }

 public boolean add(T o) throws InterruptedException {...}

 public boolean remove(Object o) {...}

}
```

Obmedzenie veľkosti množiny

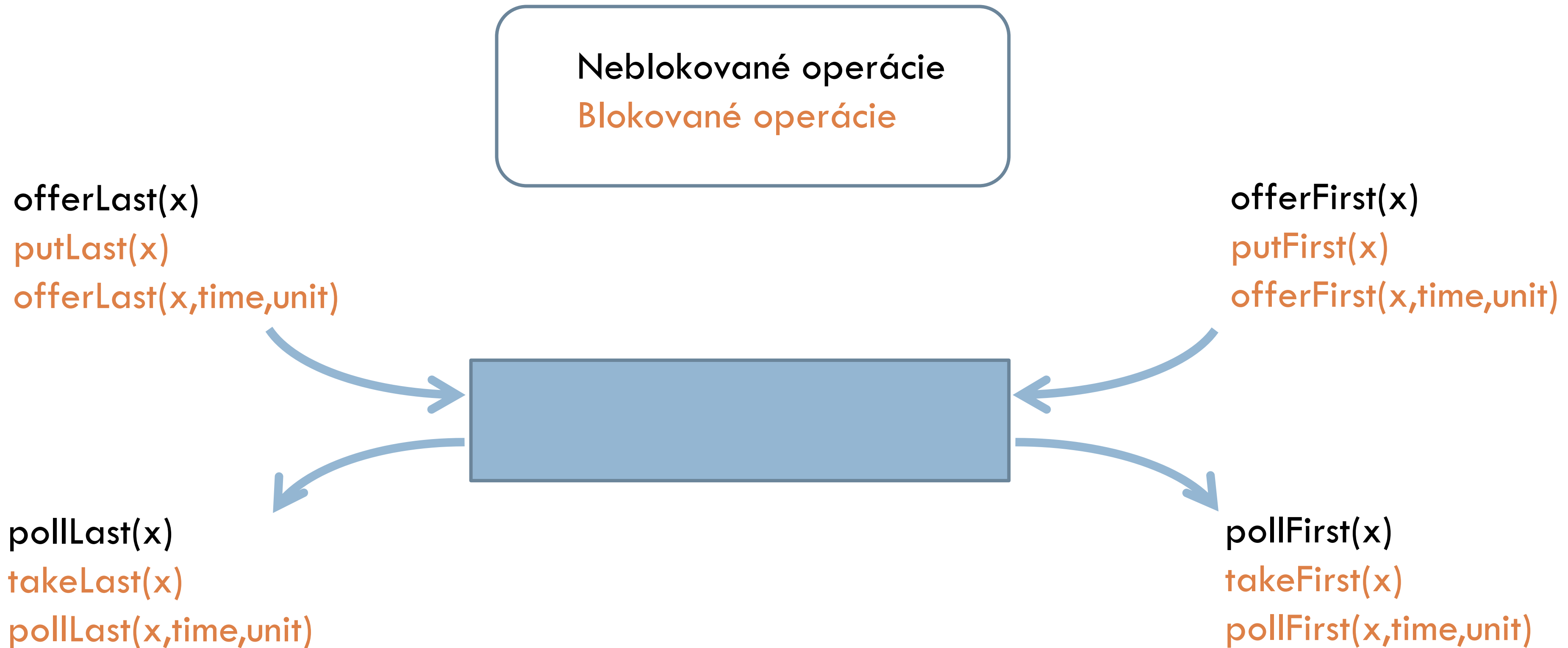
```java
public boolean add(T o) throws InterruptedException {
  sem.acquire();
  boolean wasAdded = false;
  try {
    wasAdded = set.add(o);
    return wasAdded;
  }
  finally {
    if (!wasAdded)
      sem.release();
  }
}
```

```java
public boolean remove(Object o) {
  boolean wasRemoved = set.remove(o);
  if (wasRemoved)
    sem.release();
  return wasRemoved;
}
```

java.util.concurrent

- **Konkurentné kolekcie**
  - **ConcurrentHashMap, ConcurrentSkipListMap, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet**
- **Rady, dvojsmerné rady (zásobník a rad v jednom)**
  - Implementácie BlockingQueue, **BlockingDeque**
    - **LinkedBlockingQueue, ArrayBlockingQueue**, SynchronousQueue, PriorityBlockingQueue, DelayQueue, LinkedTransferQueue, **LinkedBlockingDeque**
- **Synchronizéry**
  - **Semaphore, CountDownLatch, CyclicBarrier, Phaser, Exchanger**
- Exekútory

BlockingDeque – obojsmerný rad



Návrhový vzor Work stealing

- Každé vlákno má vlastnú deque
- **Princíp:** Vlákno si pridáva nové úlohy samé. Keď všetky svoje úlohy vykonalo, ukradne nejakú úlohu inému vláknu, ktoré má ešte roboty dosť
- **Výhoda:** konkurentné je iba kradnutie, inak sa hrám na svojom (vhodné aj pre distribuované výpočty)
- Typicky
  - Rozdeľuj a panuj
  - Webcrawler – na mnohých stránkach sú linky na ďalšie podstránky – t.j. nové stránky na spracovanie

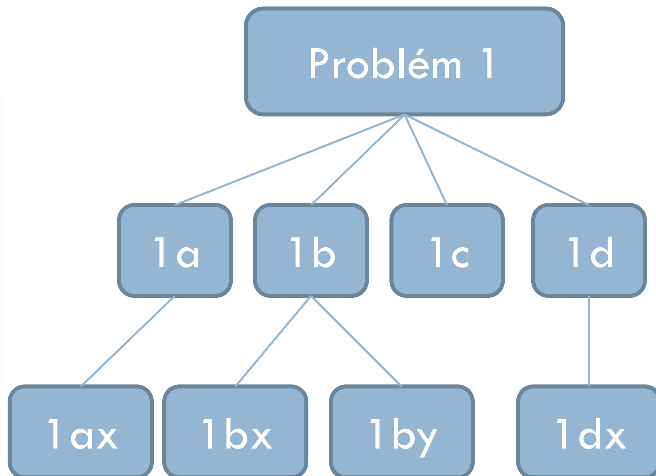
Work stealing

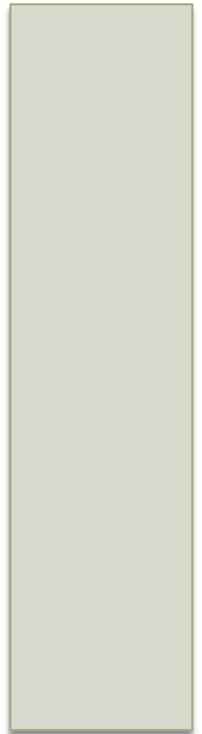
- Vlákno si úlohy pridáva na začiatok vlastného deque
  - deque.**offerFirst(x);**
- Ak ide robiť ďalšiu úlohu, najprv skúsi zobrať zo začiatku svojho deque
  - X x = deque.**pollFirst();**
  - Lokálne ako LIFO (zásobník) - prehľadávanie do hĺbky
- Ak je jeho deque prázdne (x == null), vezme si úlohu z konca deque iného vlákna, ktoré ešte má úlohy
  - X x = cudzieDeque.**pollLast();**
  - Kradnem zo spodku zásobníka inému vláknu

Work stealing

Vlákno 1

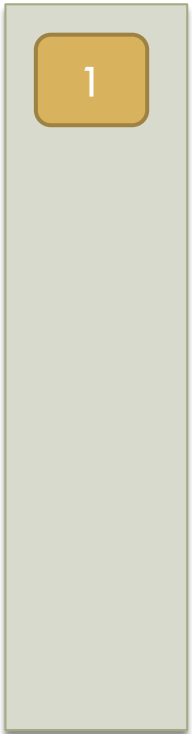
Vlákno 2

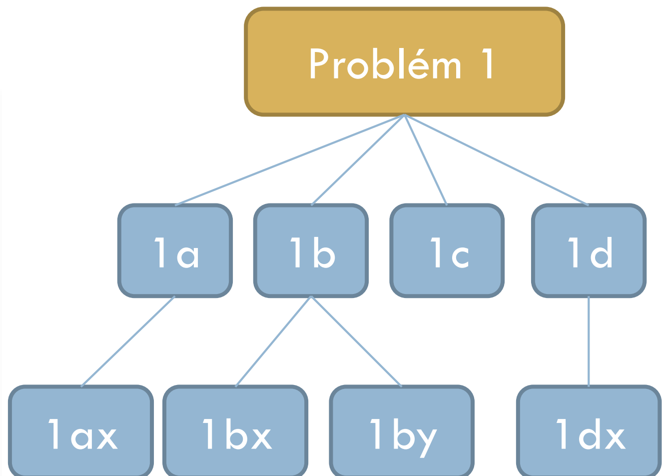


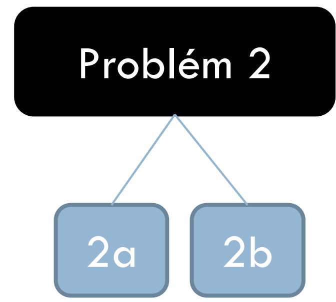


Work stealing

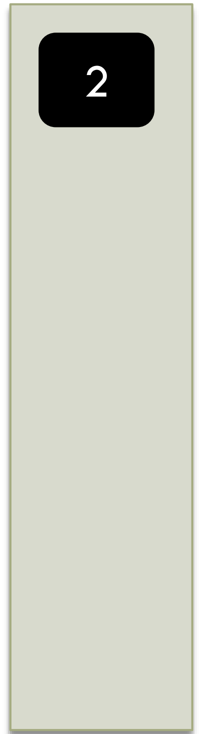
Vlákno 1







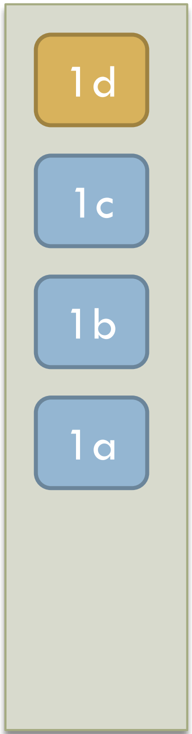
Vlákno 2

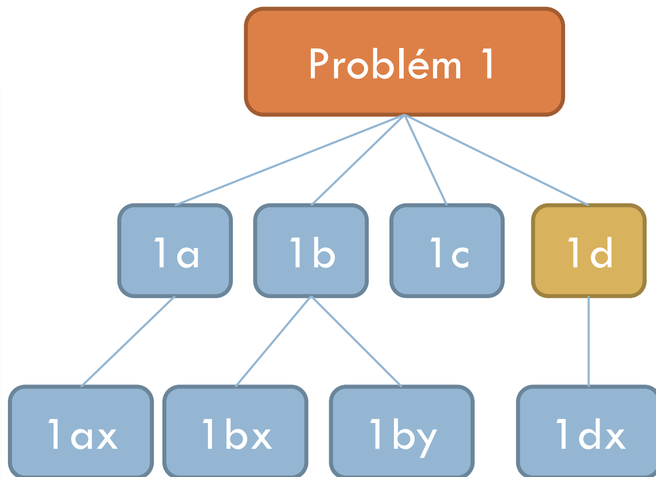


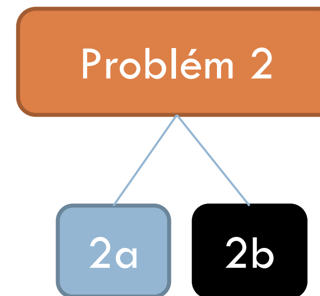
Work stealing

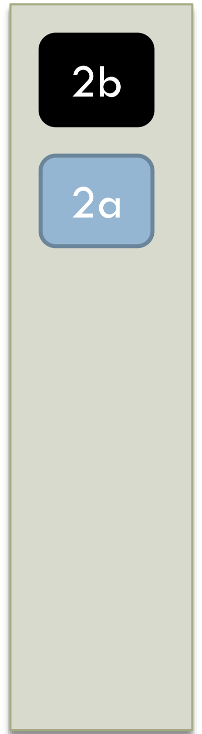
Vlákno 1

Vlákno 2



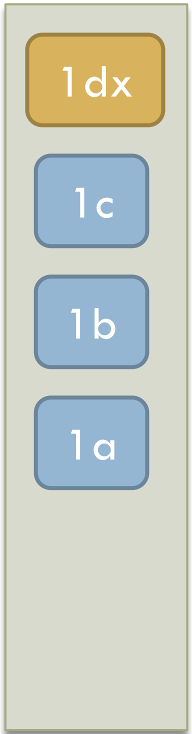


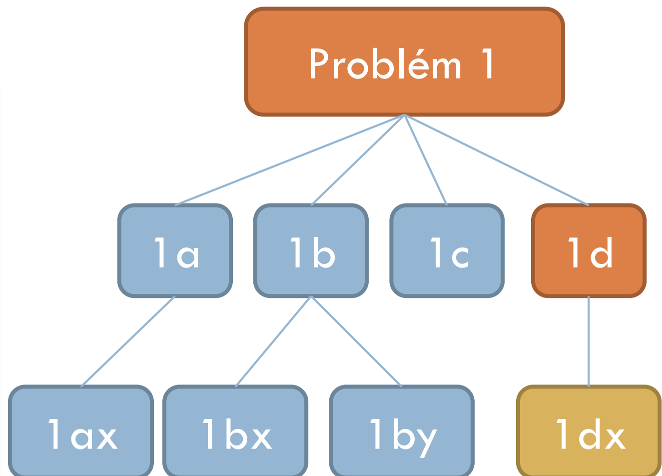


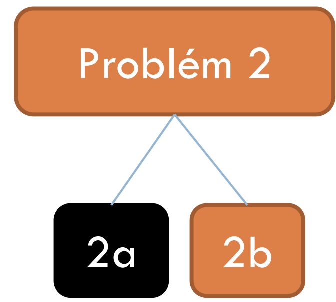


Work stealing

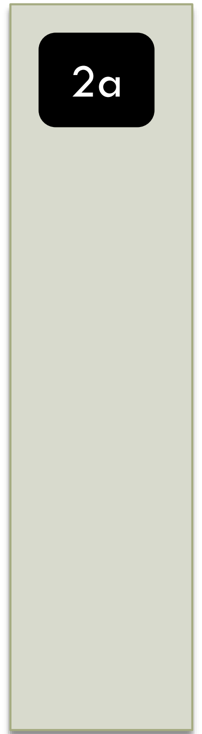
Vlákno 1







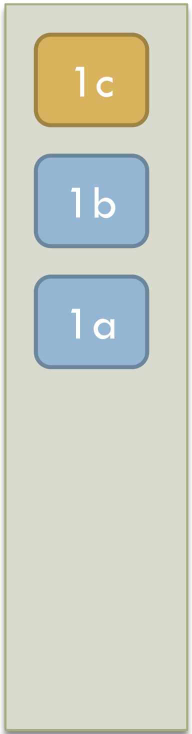
Vlákno 2

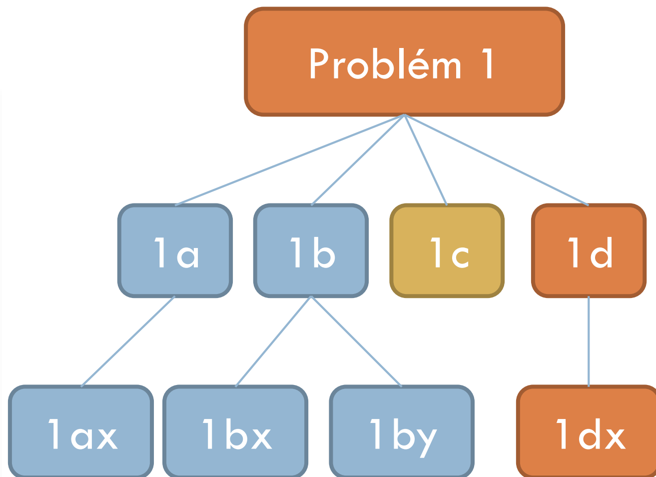


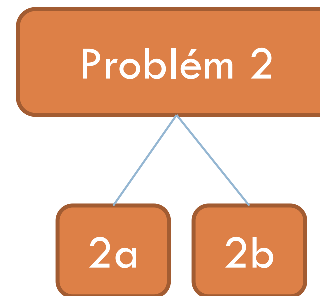
Work stealing

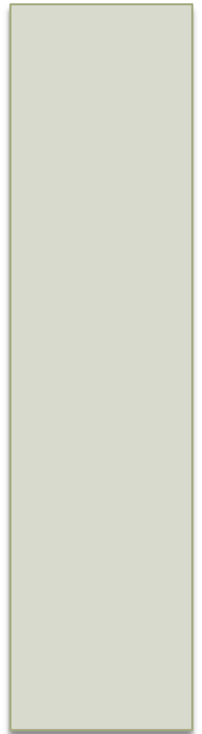
Vlákno 1

Vlákno 2





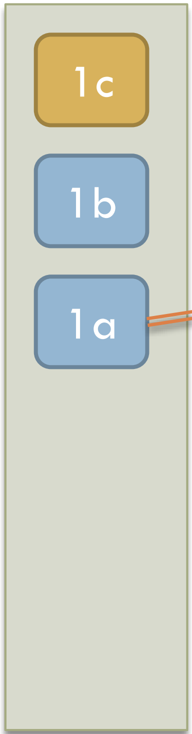


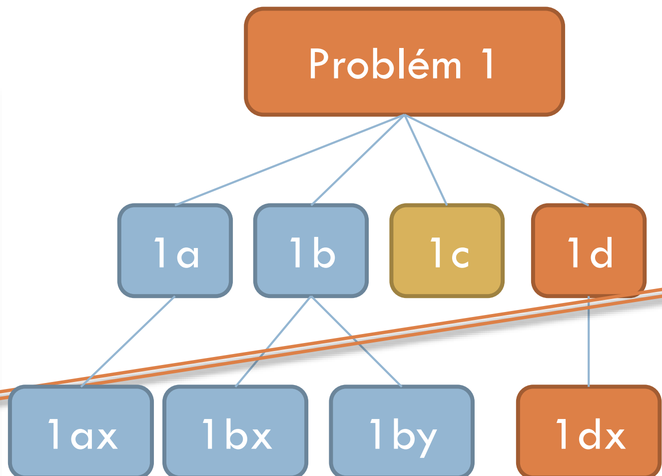


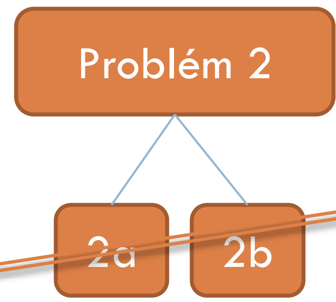
Work stealing

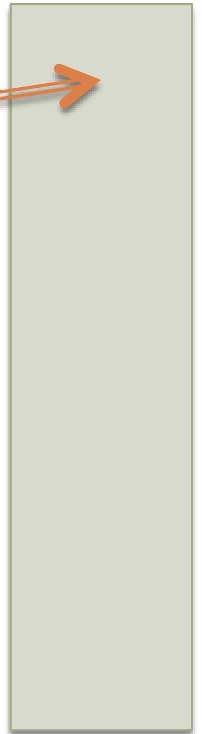
Vlákno 1

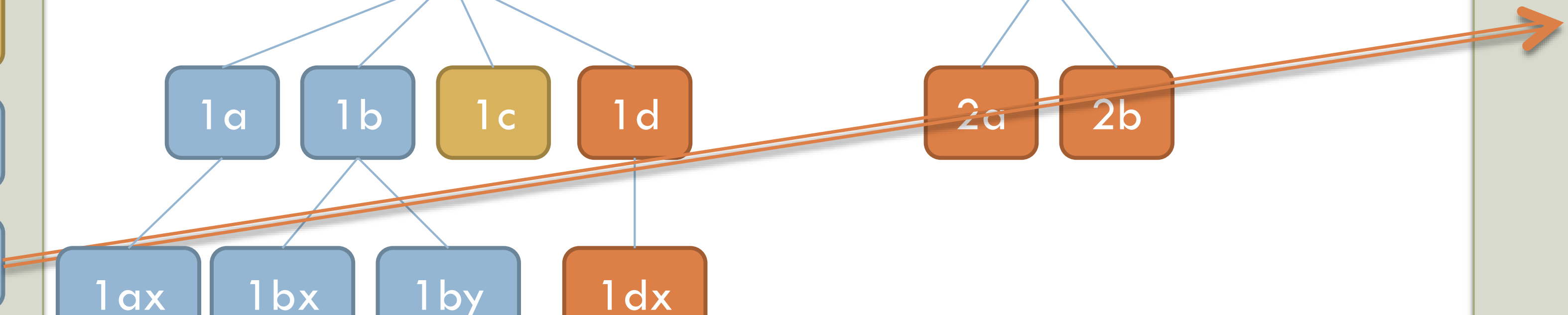
Vlákno 2





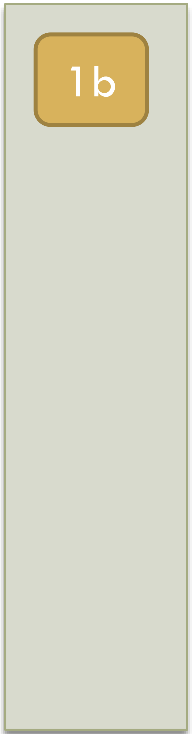


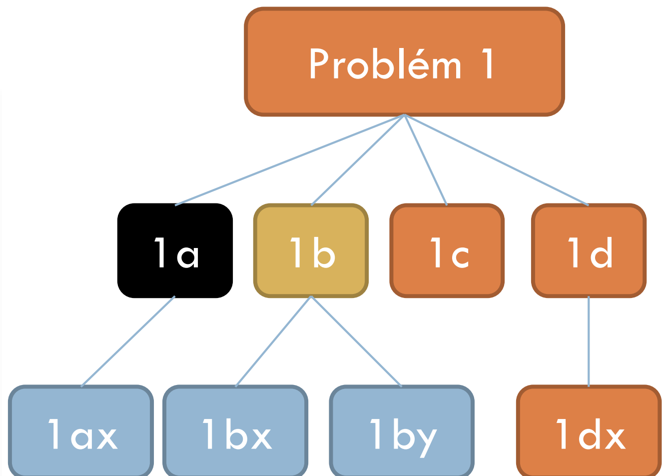


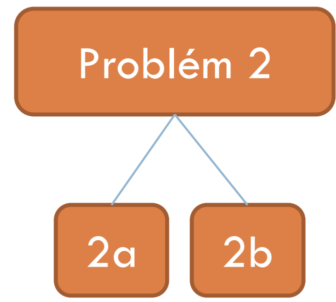


Work stealing

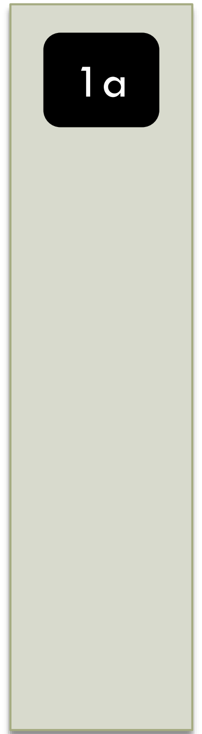
Vlákno 1





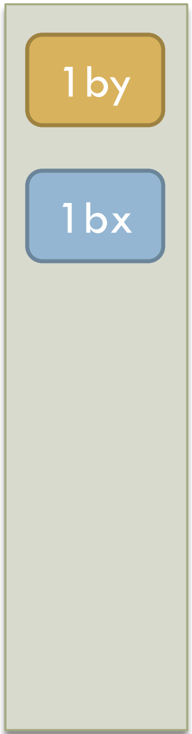


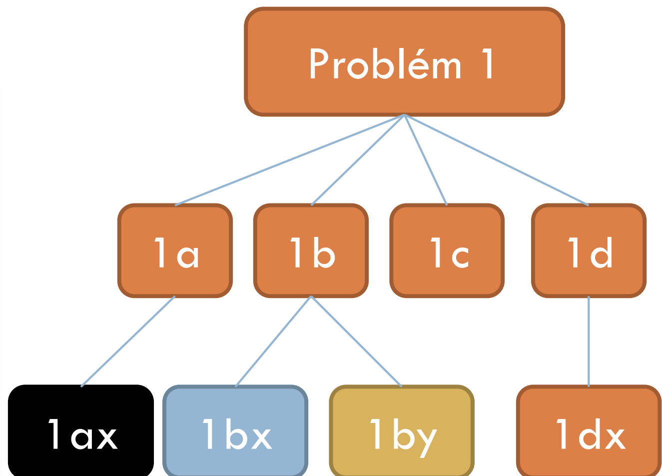
Vlákno 2

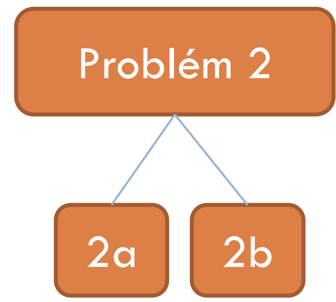


Work stealing

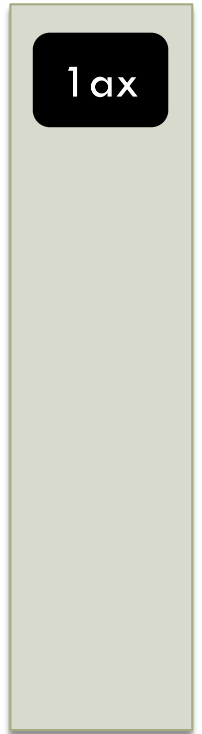
Vlákno 1







Vlákno 2



Zadanie 2

- Modifikujte riešenie posledného zadania tak, aby úloha Searcher bola realizovaná cez work stealing

Riešenie

- Každá úloha Searcher bude mať vlastné BlockingDeque, ale zároveň musí poznať aj BlockingDeque ostatných
- Podadresáre pridávam na začiatok svojho deque
- Ak je moje deque prázdne, musím sa popozerať na konce iných
  - Čo ak ani v nich nič nie je? Budem sa cykliť donemoty?
  - Využijeme semafor
    - release() po pridaní nového podadresára
    - acquire() pred čítaním z deque
    - Počet releasov == počet adresárov vo všetkých deque dohromady
    - Ak ma pustí acquire(), určite nejaký adresár nájdem (u seba alebo inde) – alebo nájdem poison pill a skončím