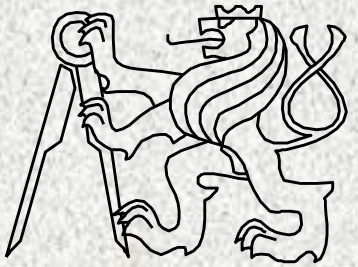
VLÁKNA



BD6B36PJV 8

Fakulta elektrotechnická
České vysoké učení technické

Dvě možnosti implementace vláken

1. Třída, která je potomek třídy **`extend Thread`**
   - Tam kde potřebujeme více metod k práci s vlákny (`isAlive()`, `join()`, `sleep()`,..)
2. Třída implementující **`interface Runnable`**
   - Potřebujeme jen, aby objekty třídy pracovaly „ve vláknu“, měly schopnost „být vláknem“

Každý program v Javě má implicitní vlákno

- Z něj vznikají další paralelní (dceřiná) vlákna
- Měly bychom zajistit, aby hlavní vlákno končilo poslední (uvolnění zdrojů)
- Odkaz na hlavní vlákno **`Thread currentThread()`** v hlavním programu a tak můžeme vlákno řídit

Vlákna v Javě, implementace rozhraní `Runnable`

- Vlákno zkonstruujeme na jakémkoli objektu třídy, která implementuje rozhraní `Runnable`
- vytvoříme instanci třídy `Thread` a předáme objekt (referenci na instanci třídy) třídy, který bude probíhat jako samostatné vlákno
  - `new Thread(Runable <třída>, String <jméno vlákna>);`
- třída musí implementovat jednu metodu rozhraní `Runnable` : `run()`
  - metoda `run()` určuje činnost vlákna
  - nespouští se sama!
- Vhodné resp. nutné definovat metodu `start()` - není to „povinné“, ale odstraní to nejednoznačnost
- vlákno spustíme metodou `start()`, ta odstartuje metodu `run()`

Vytvoření vlákna `implements`

Ukázky:

```
public class VytvoreniVlakna1 implements Runnable {
public class VytvoreniVlakna2 implements Runnable {

public class DemoVlakno0R {
```

Vlákna v Javě, potomek třídy `Thread`

- vlákno je instancí třídy `Thread`
- v konstruktoru třídy (jejíž instance budou vlákny) voláme konstruktor nadřazené třídy `Thread` a předáváme jméno vlákna
- metodu třídy `Thread` `run()` překryjeme pomocí `super()`.
- metoda `run()` určuje činnost vlákna
  - nespouští se sama
- metoda `start()` spouští metodu `run()`

Vytvoření vlákna `extends`

Ukázky:

```
public class VytvoreniVlaknaThread extends Thread {

public class DemoVlakno0T {
```

Více vláken souběžně bez řízení

- Počet vláken není omezen, vlákna lze pojmenovat
- Dva způsoby:
  - Vytvoříme vlákno (konstruktor **`Thread`**) v konstruktoru a předáme mu odkaz na **`this`** objekt (třídy, která je **`Runnable`** – implementuje (překryjeme) **`run()),`** který bude „ve vláknu“ a odstartujeme metodou
    - Vytvořením objektu se vše provede
  - Vytvoříme vlákno (konstruktor **`Thread)`** v konstruktoru a předáme mu odkaz na **`this`** objekt (třídy, která je **`Runnable`** – implementuje **`run()),`** který bude „ve vláknu“
    - Vlastní start provedeme až po vytvoření objektu tak, ze zavoláme metodu atributu typu **`Thread – start()`**

Více vláken souběžně bez řízení

- Pomocí metody `Thread.sleep()` můžeme vlákna uspávat
- Hlavní vlákno nemusí skončit jako poslední z vláken, zařídíme pomocí `join()-` je to dáno časem života hlavního vlákna

Více vláken souběžně bez řízení

Ukázky:

```
public class ViceVlaken implements Runnable {

public class DemoVlakno2_vice_vlaken {
public class DemoVlakno2a_vice_vlaken {
```

Vlákna v Javě, další metody `Thread`

`String getName()` – vrací jméno vlákna

`int getPriority()` – vrací prioritu vlákna

`void join()` – vlákno čeká na ukončení vlákna

`static void sleep()` – pozastaví vlákno na určenou dobu

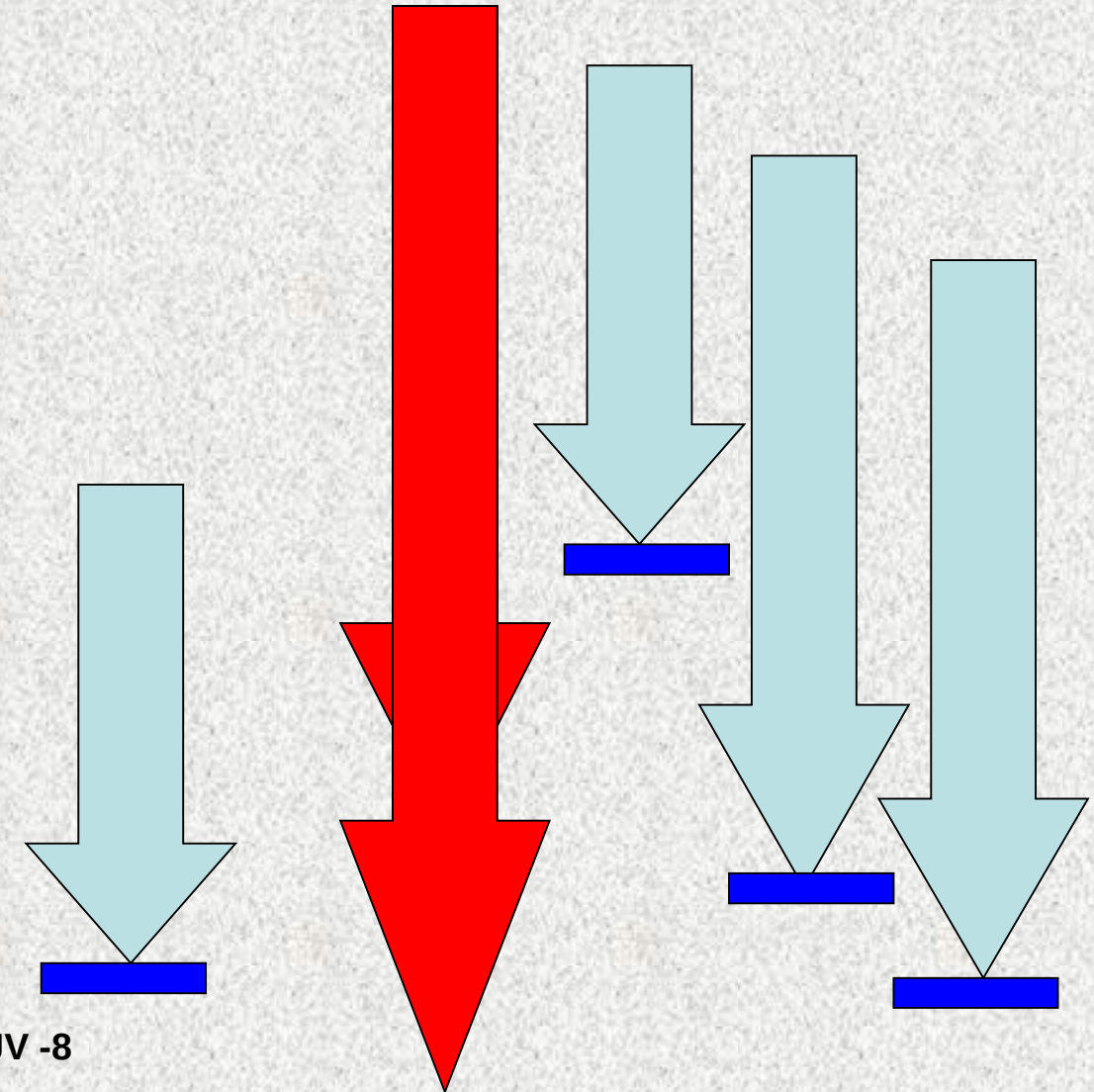
`boolean isAlive()` – zda vlákno běží

`static void yield()` – běžící vlákno předá řízení jinému

Čekání na ukončení běhu vlákna

- `static void sleep()` – pozastaví vlákno na určenou dobu
- `boolean isAlive()` – zjistí, zda vlákno běží
- `void join()` – vlákno čeká na ukončení dceřiného vlákna, aktuální vlákno se zastaví a bude čekat na skončení běhu onoho vlákna.
  - Lze využít i verze s časovým limitem - pak se bude čekat maximálně po zvolenou dobu, čeká, dokud se dceřiný proces neukončí
  - Vlákno je slušné a počká až mateřské dokončí nebo čeká jen učitou dobu, určitě hlavní vklákno nepředběhne

Čekání na ukončení běhu vlákna



Čekání na ukončení běhu vlákna

Ukázky:

```
public class Join implements Runnable {
public class IsAlive  implements Runnable {


public class DemoVlakno3_join_isAlive {
```

Priority vláken

Jsou-li běhuschopná dvě vlákna, pak je řízení předáno tomu vláknu, které má vyšší prioritu a je ve stavu runnable

O předání řízení (přidělení CPU) se stará JVM Scheduler.

- Nastavení priority **`setPriority()`**
- Zjištění priority **`getPriority()`**
- Hodnoty priority
  - **`MAX_PRIORITY  - 10`**
  - **`MIN_PRIORITY  -  1`**
  - **`NORM_PRIORITY -  5`**

Priorita vláken

Pravidla plánovače:

- Běží vždy to, co má z běhuschopných vláken nejvyšší prioritu
- Je-li více vláken se stejnou prioritou, je řízení postupně předáváno všem v náhodném pořadí, možno vynutit předání `yield()`
- Vlákno s nižší prioritou mohou získat řízení, jen když vlákna s vyšší prioritou se dostanou do neběhuschopného stavu, vlákno s vyšší prioritou nelze přinutit k předání řízení standardním mechanismem plánování, jen zásahem zvenku
- Pokud se do běhuschopného stavu dostane vlákno s vyšší prioritou, je běžící vlákno okamžitě přinuceno předat řízení ve prospěch tohoto vlákna (preemptivní plánování)

Priorita vláken - pravidla

- Čekání na vstup - typický případ, běžící vlákno je přerušeno vláknem vyšší priority
- Vlákno je automaticky uvedeno do stavu neběhuschopné, když čeká na vstup/výstup
- Vstup/výstup má nejvyšší prioritu, kdykoli přeruší, když je možnost předávání dat
- Není třeba další synchronizace, spolupráce je asynchronní

Priority vláken

Ukázky:

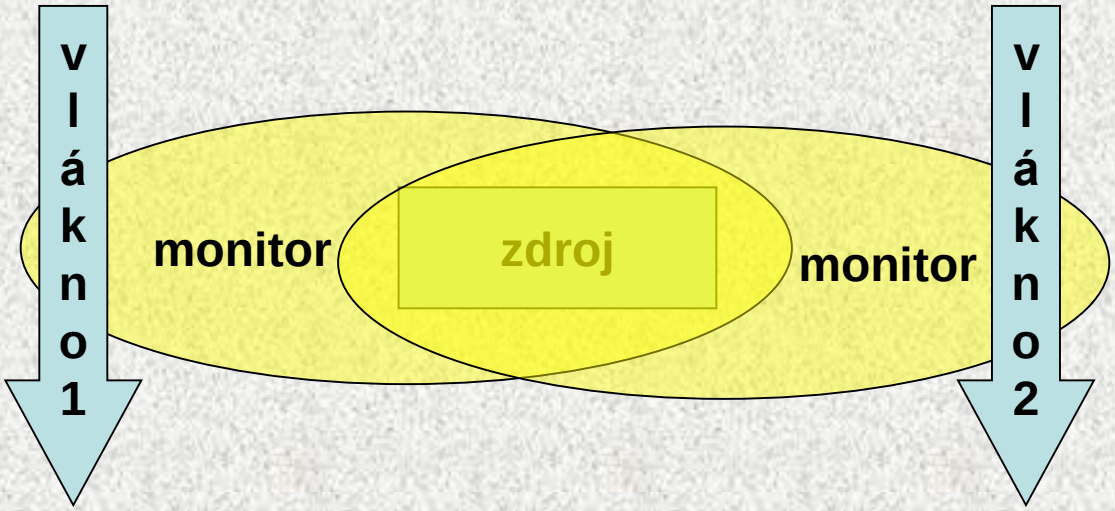
```
class Priority implements Runnable {

public class DemoVlakno4_priorita {
```

Synchronizace v Javě

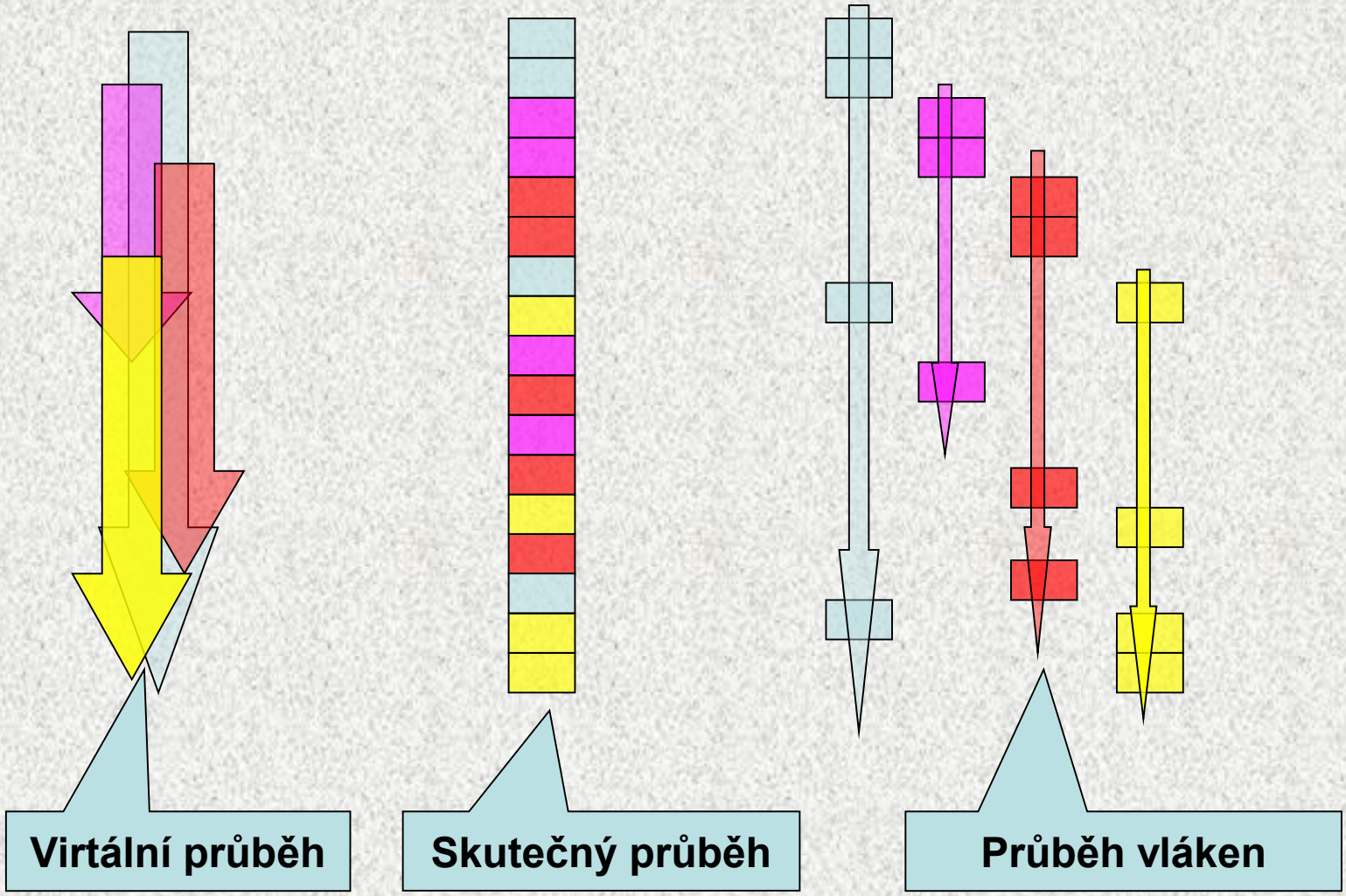
Hlavní synchronizačním primitivem jsou monitory

Každý objekt má automaticky přiřazen svůj monitor.

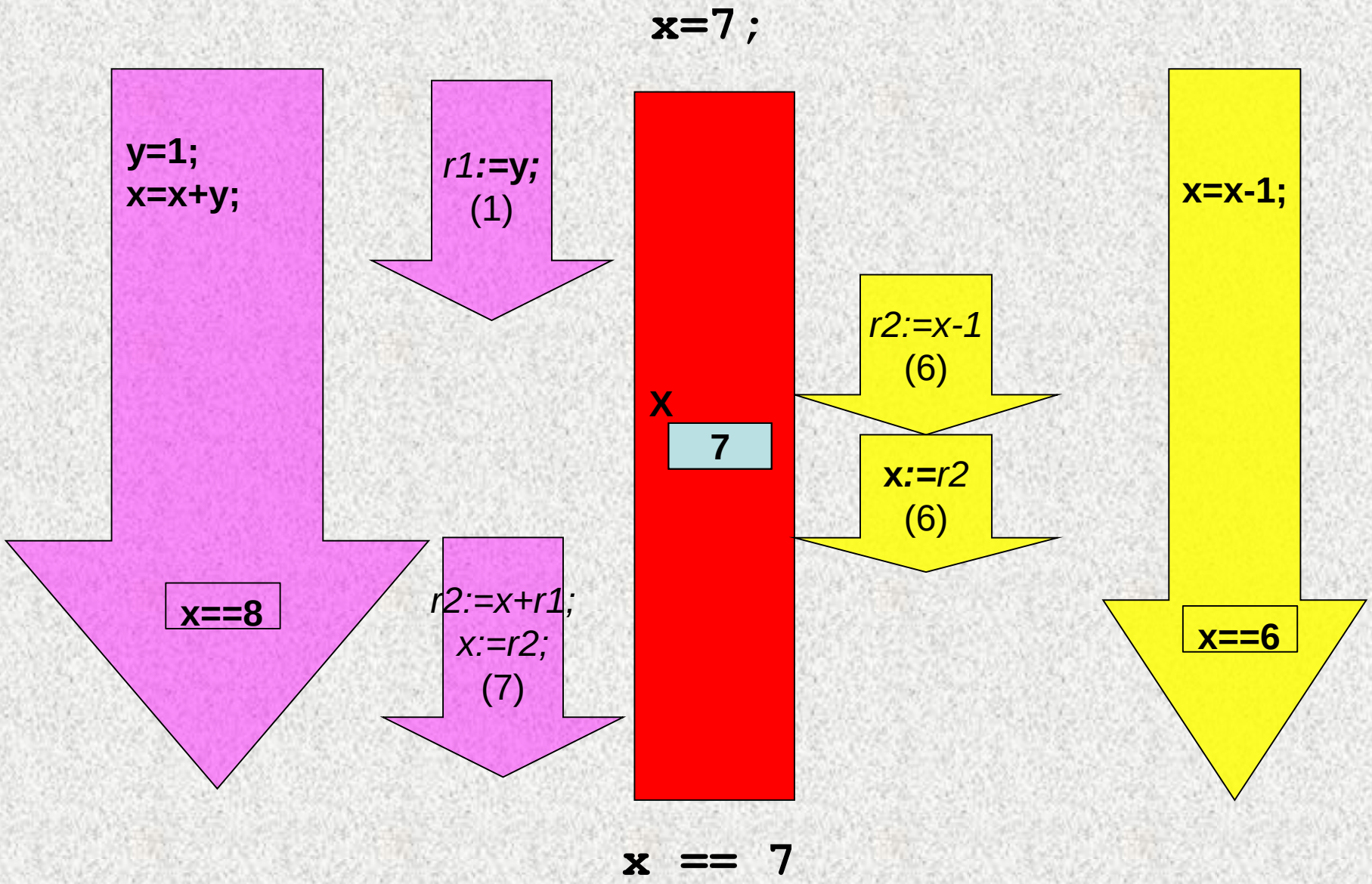
1. Metody, které patří do monitoru, jsou označeny pomocí klíčového slova **`synchronized`**.
2. Do monitoru libovolného objektu však lze obalit libovolný příkaz (blok) kódu pomocí konstrukce pro objekt, který tuto metodu volá: **`synchronized(objekt) { ... }`**



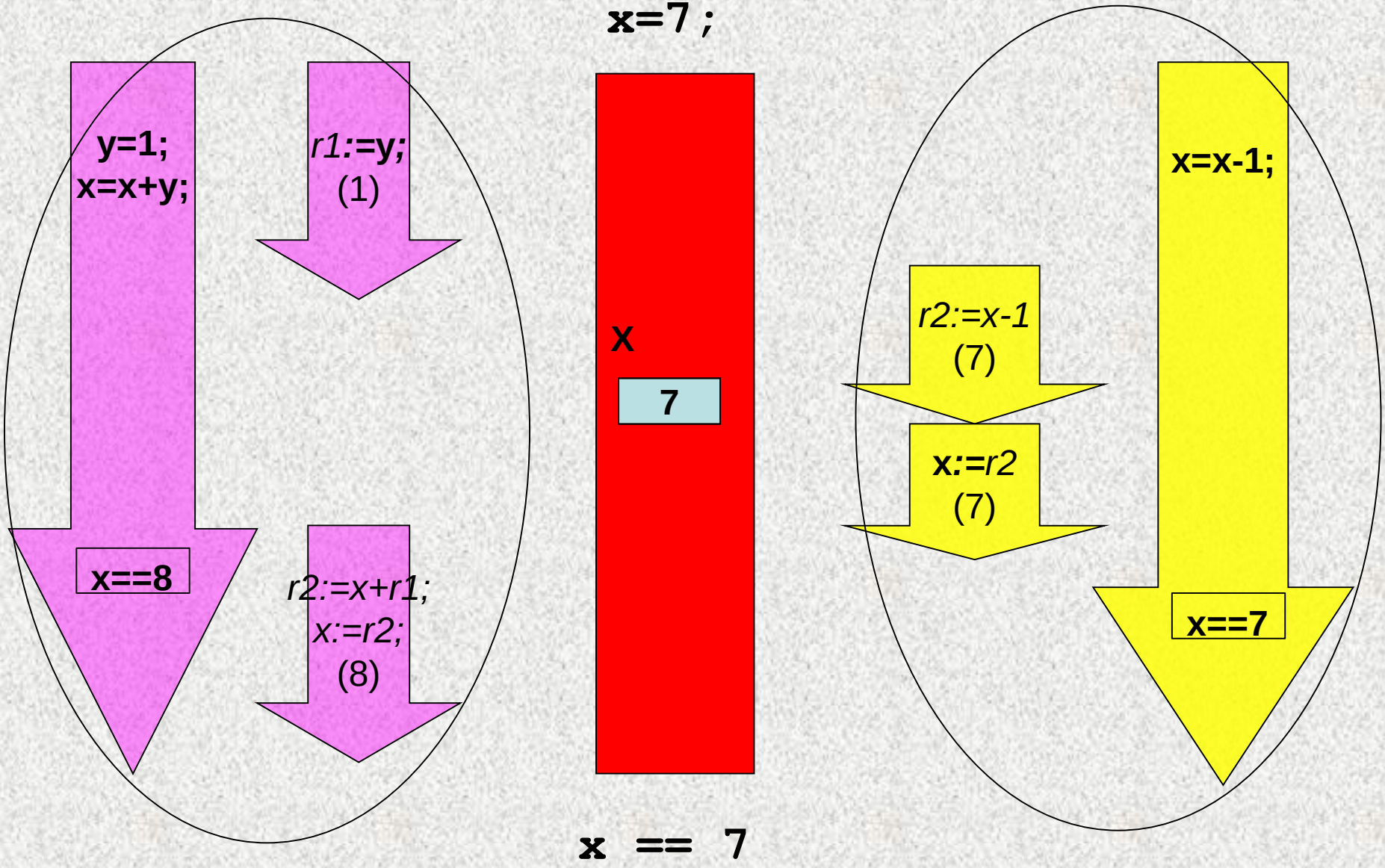
Vlákna – průběh skutečný a virtuální



Problém sdílení prostředků, kolize



Problém sdílení prostředků, synchronizace



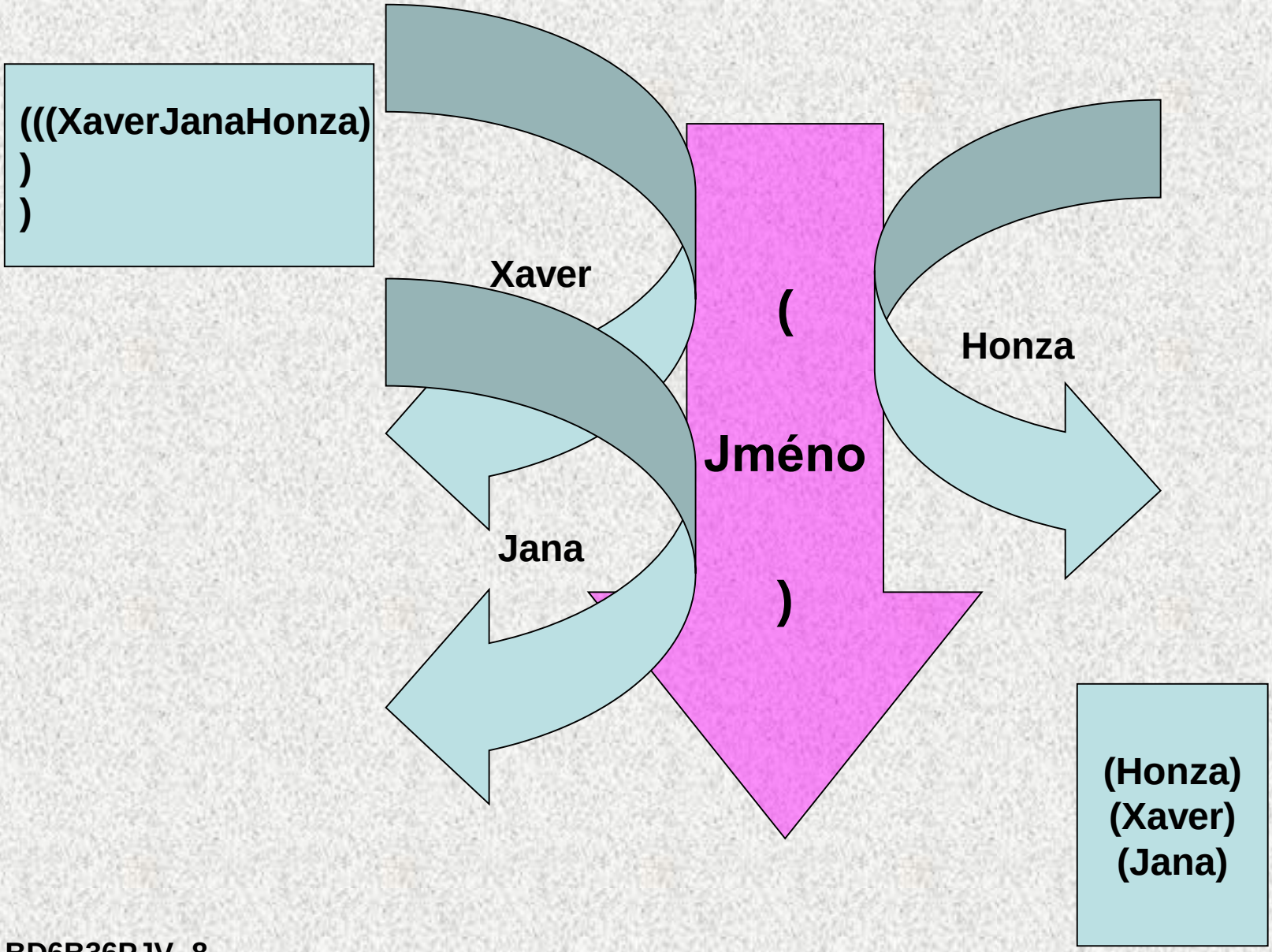
Synchronizace činnosti vláken

- Vlákna spolupracují, problém sdílení datového prostoru
  - Problém nedeterminovanosti přístupu ke společným prostorům
- Možné řešení je tzv. monitor (kritická sekce)
  - objekt, který vláknu zpřístupní zdroj (paměť, linku,..),
  - v daném okamžiku aktivně umožní monitor používat jen jedno vlákno
  - „pro daný časový interval vlákno vlastní monitor“, monitor smí vlastnit jen jedno vlákno
  - vlákno běží, jen když vlastní monitor, jinak čeká
- Všechny objekty v Javě „mají“ monitor
  - Vlákno vstoupí do monitoru **voláním metody** s přívlastkem **`synchronized`** (**lze i příkaz**) - tuto metodu resp. příkaz nazýváme synchronizovanou
  - O vláknu, které úspěšně jako první zavolá synchronizovanou metodu říkáme, že je „uvnitř“ metody a má k dispozici všechny zdroje používanými touto metodou
  - Jiné vlákno, které volá synchronizovanou metodu čeká, dokud aktivní vlákno synchronizovanou metodu neopustí (nebo uvolní zámek pomocí **`wait`**)
- Synchronizace - řízená „linearizace vláken“

Synchronizace činnosti vláken

- Synchronizace
  1. **metodou** – v třídě, která bude vstupovat do vlákna
  2. **příkazem** – mimo třídy, ve vláknu se určí, který příkaz kterého objektu je „kritický“

Příklad na závorky, synchronizovaná metoda



Synchronizovaná metoda

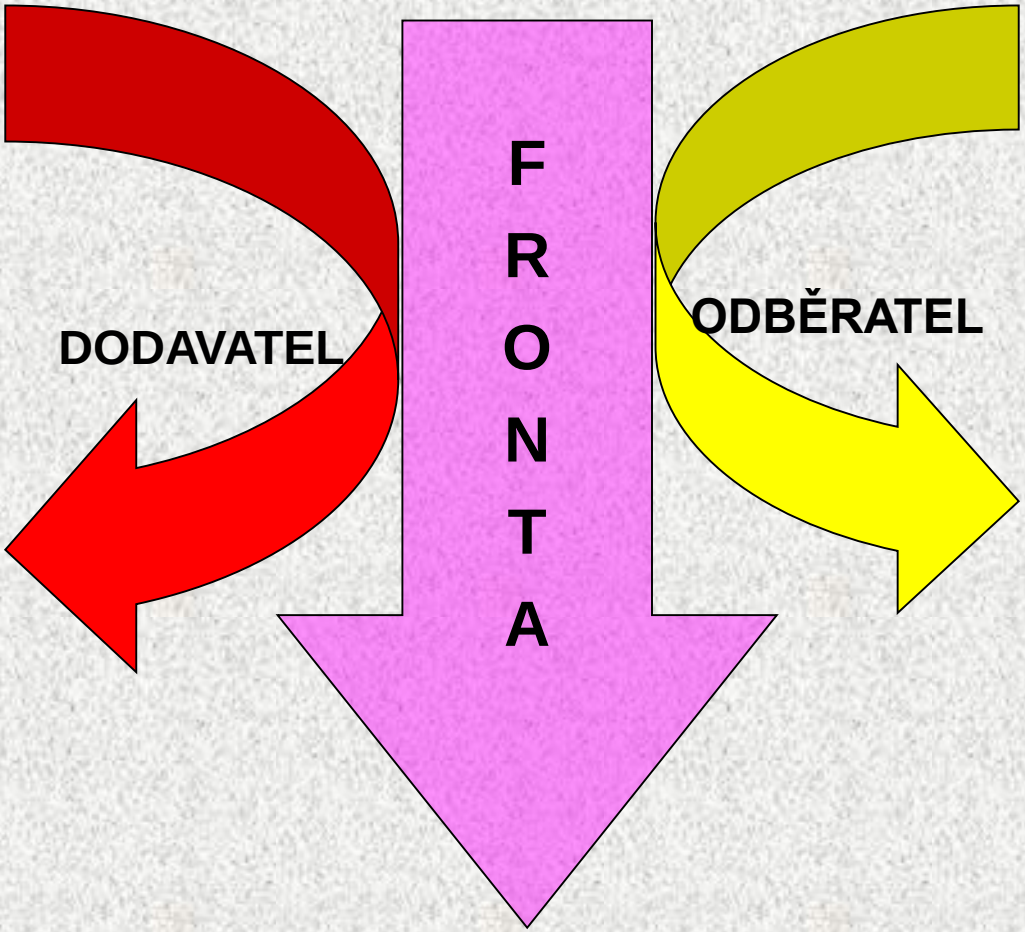
Ukázky:

```
public class Synchronizace {
public class Synchronizace2 {


public class DemoVlakno5_zavorky {
```

Komunikace mezi vlákny

- Nejedná se o společnou metodu různých vláken kterou vlákna musí používat synchronně
- Jedná se o různé metody různých vláken nad týmž prostorem - uvnitř synchronizované metody lze volat metody, které umožní ovládat komunikaci mezi vlákny:
- Řeší se to synchronizovanou metodou, která obsahuje:
  - **`wait()`** – uvolni monitor a pozastaví svou činnost
    - do doby signálu od **`notify()`** jiného vlákna
    - do daného času
  - **`notify()`** – signál pro vlákno pozastavené **`wait()`** aby obnovilo činnost a převzalo monitor
    - **`notifyAll()`** – probudí všechna vlákna pozastavená metodou **`wait()`**, monitoru se zmocní vlákno s nejvyšší prioritou

Komunikace mezi vlákny



Komunikace mezi vlákny, příklad

vydavatel

Vlozit do Fronty

Není obsazeno

1) **vložit**
2) **obsazeno = !obsazeno**
3) **notify() zakazníka**

Je obsazeno

1) **wait()**

zákazník

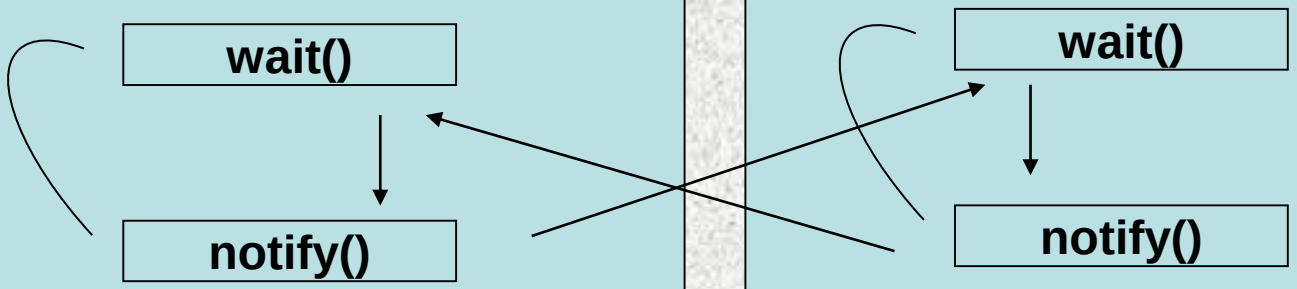
Vyjmout z Fronty

Je obsazeno

1) **vyjmout**
2) **obsazeno = !obsazeno**
3) **notify() vydavatele**

Není obsazeno

1) **wait()**



Komunikace mezi vlákny

Ukázky:

```
public class WaitNotify {


public class DemoVlakno7 {
public class DemoVlakno7_komunikace {
```

Stavy vláken

- Nové vlákno, spuštěno **`start()`**
- Běžící **`(running)`**, jedno z vláken je běžící, ostatní čekají
- Běhuschopné **`(runnable)`**, je spuštěno, ale neběží, čeká na předání řízení
- Neběhuschopné
  - Blokováno (blocked)
    - uspáno **`sleep()`**, **`join()`**, čeká na O/I
  - Čekající (wait)
    - čeká **`wait()`**
- Mrtvé vlákno, skončila metoda **`run()`**

Doba a pořadí předávání řízení závisí na:

- na stavu okolních vláken,
- na prioritě vlákna,
- na schopnostech OS.

Stavy vláken

Ukázky:

```
class Suspend {

public class DemoVlakno8 {
```

Stavy vlákna