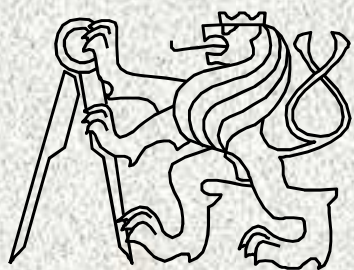


# VLÁKNA



A0B36PR2-Programování 2

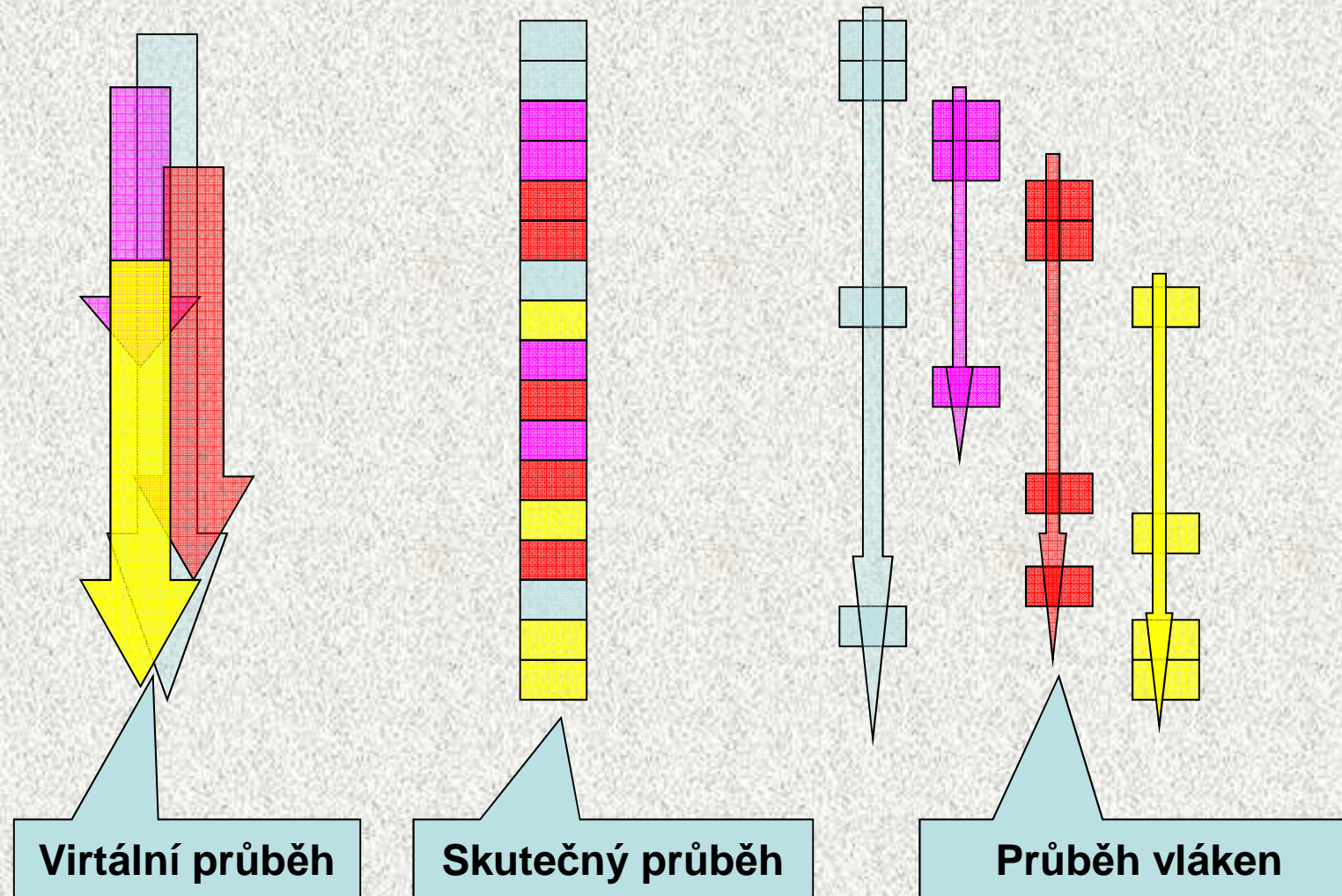
Fakulta elektrotechnická

České vysoké učení technické

# Vlákna v Javě

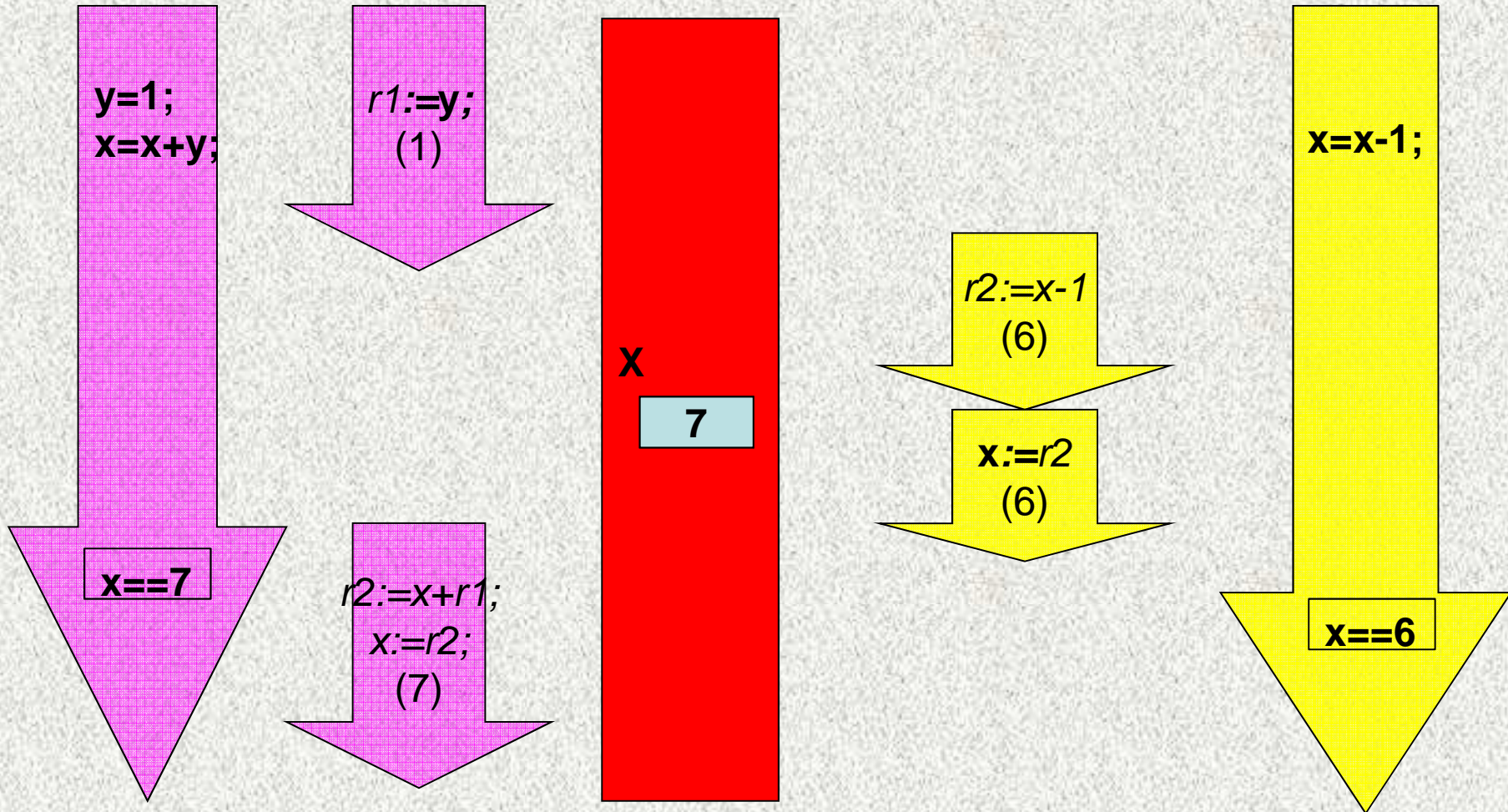
- Oblasti použití
  - Nesoulad časových nároků různých nezávislých částí programu
  - Dlouhé periferní přenosy, čekání na odezvu na vstupu
  - Simulační výpočty, opakující se výpočty
  - Úlohy typu producent/konzument, server/klient

# Vlákna – průběh skutečný a virtuální



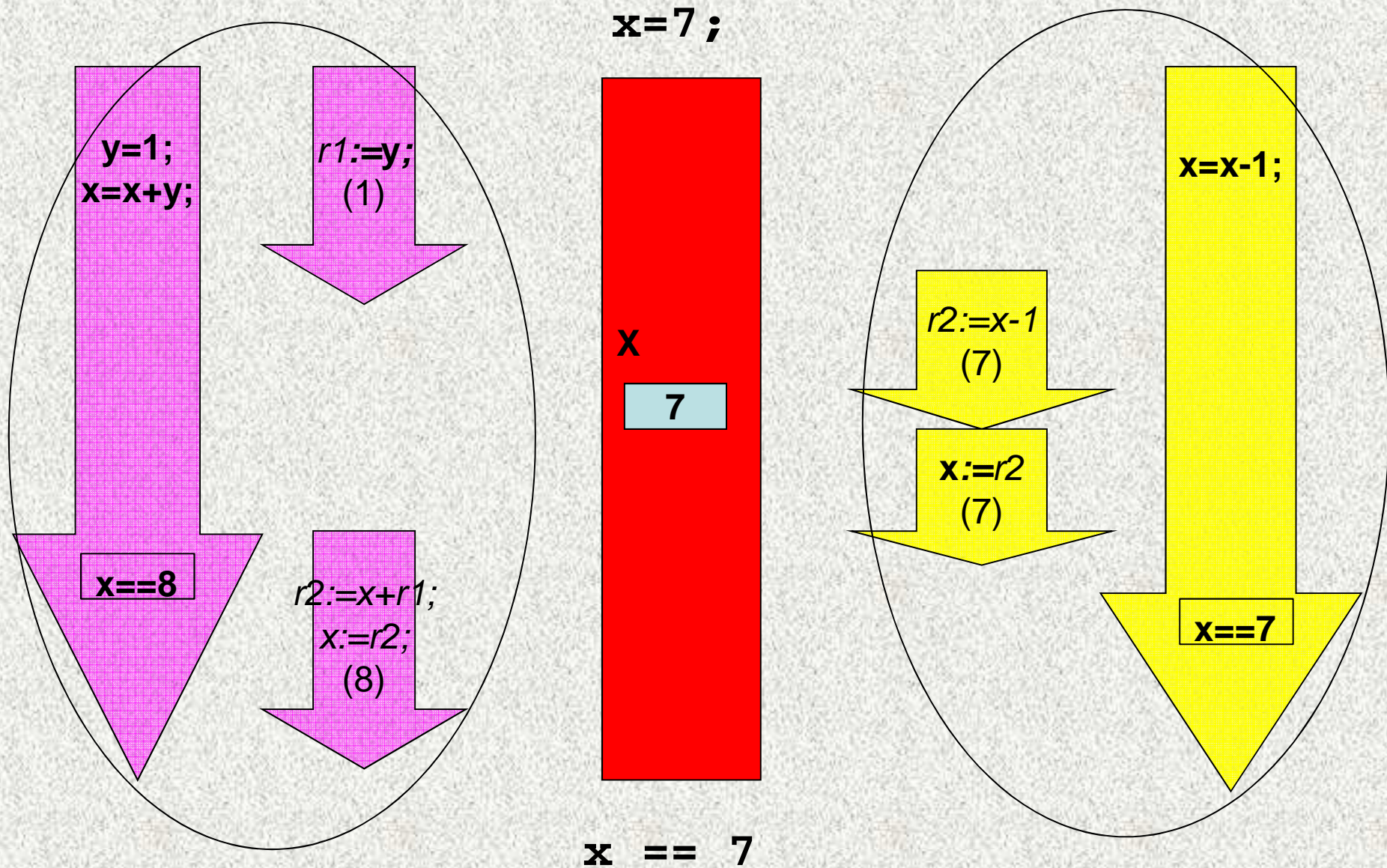
# Problém sdílení prostředků, kolize

**x=7;**



**x == 7**

# Problém sdílení prostředků, synchronizace



## Vlákna v Javě, potomek třídy **Thread**

- Vlákno je instancí třídy **Thread**
- v konstruktoru třídy (jejíž instance budou vlákna) voláme konstruktor nadřazené třídy **Thread** a předáváme jméno vlákna
- metodu třídy **Thread run( )** překryjeme
- metoda **run( )** určuje činnost vlákna
  - nespouští se sama
- metoda **start( )** spouští metodu **run( )**

# Vlákna v Javě, potomek třídy **Thread**, příklad

```
class MyThreadExtend extends Thread {
    // Construct a new thread.
    MyThread_Extend(String name) {
        super(name); // name thread
        start(); // start the thread
    }
    // Begin execution of new thread.
    public void run() {
        System.out.println(getName() + " starting.");
        //...
        System.out.println(getName() + " terminating.");
    }
}

class ExtendThread {
    public static void main(String args[]) {
        System.out.println("Main thread starting.");
        MyThreadExtend mt = new MyThreadExtend("Child #1");
        // ...
        System.out.println("Main thread ending.");
    }
}
```

# Vlákna v Javě, potomek třídy `Thread`, příklad

```
public class MyThreadExtend1 extends Thread {
    private MyThreadExtend1(String jmeno) {
        super(jmeno);
    }
    // @Override
    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println("Cykl " + i + ". " + getName());
            try {
                Thread.sleep(10);
            }
            catch (InterruptedException e) {
                System.out.println("Nezadouci probuzeni - " + getName());
            }
        }
    }
    public static void main(String[] args) {
        MyThreadExtend1 vlakno1 = new MyThreadExtend1("prvni");
        vlakno1.start();
        new MyThreadExtend1("druhe").start();
        new MyThreadExtend1("treti").start();
    }
}
```



## Vlákna v Javě, implementace rozhraní **Runnable**

- Vlákno zkonstruujeme na jakémkoli objektu třídy, která implementuje rozhraní **Runnable**
- třída musí implementovat metodu **run( )**
  - metoda **run( )** určuje činnost vlákna
  - nespouští se sama
- vhodné definovat metodu **start( )** - není to „povinné“, ale odstraní to nejednoznačnost
- vytvoříme instanci třídy **Thread** a předáme objekt (referenci na instanci třídy) třídy, který bude probíhat jako samostatné vlákno
- vlákno spustíme metodou **start( )** ta odstartuje metodu **run( )**

# Vlákna v Javě, implementace rozhraní `Runnable`, příklad I

```
class MyThread implements Runnable {
    String thrdName; int count;

    MyThread(String name) {
        thrdName = name; count = 0;
    }
    // Entry point of thread.
    public void run() {
        System.out.println(thrdName + " starting.");
        //.. System.out.println("In " + thrdName +
            ", count is " + count);
        System.out.println(thrdName + " terminating.");
    }
}

class UseThreads {
    public static void main(String args[]) {
        System.out.println("Main thread starting.");
        // First, construct a UseThreads object.
        MyThread mt = new MyThread("Child #1");
        // Next, construct a thread from that object.
        Thread newThrd = new Thread(mt);
        // Finally, start execution of the thread.
        newThrd.start();
        do {
            // ...
            while(mt.count!=10);
        }
        System.out.println("Main thread ending.");
    }
}
```

# Vlákna v Javě, implementace rozhraní `Runnable`, příklad II

```
class MyThreadI implements Runnable {
    Thread thrd; int count;

    // Construct a new thread.
    MyThreadI(String name) {
        thrd = new Thread(this, name); count = 0;
        thrd.start(); // start the thread
    }

    // Begin execution of new thread.
    public void run() {
        System.out.println(thrd.getName() + " starting.");
        //.. System.out.println("In " + thrd.getName() +
            ", count is " + count);
        System.out.println(thrd.getName() + " terminating.");
    }
}

class UseThreads_Implemented_Improved {
    public static void main(String args[]) {
        System.out.println("Main thread starting.");
        MyThreadI mt = new MyThreadI("Child #1");
        // cover: construct a thread on the MyThreadI object and starts
        do {
            // ...
            while(mt.count!=10);

            System.out.println("Main thread ending.");
        }
    }
}
```

# Vlákna v Javě, implementace rozhraní `Runnable`, příklad III

```
public class UseThreads1 implements Runnable {
    private Thread vlakno;
    String thrdName;
    public void start() {
        vlakno = new Thread(this);
        vlakno.start();
    }
    UseThreads1(String name) {
        thrdName = name;
    }
    // @Override
    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println("Cykl " + i + ". " + this.thrdName);
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                System.out.println("Nezadouci probuzeni - " + this.thrdName);
            }
        }
    }
    public static void main(String[] args) {
        UseThreads1 vlakno1 = new UseThreads1("prvni");
        vlakno1.start();
        new UseThreads1("druhe").start();
        new UseThreads1("treti").start();
    }
}
```

# Vlákna v Javě, další metody

- `String getName()` – vrací jméno vlákna
- `int getPriority()` – vrací prioritu vlákna
- `boolean isAlive()` – zda vlákno běží
- `void join()` – čeká na ukončení vlákna
- `static void sleep()` – pozastaví vlákno na určenou dobu
- `static void yield()` – běžící vlákno předá řízení jinému

# Řízené ukončení činnosti vláken I

- Nejméně vhodné – testováním hodnoty proměnné procesu vlákna

```
class MoreThreads_Multiple {
    public static void main(String args[]) {
        System.out.println("Main thread starting.");

        MyThreadM mt1 = new MyThreadM("Child #1");
        MyThreadM mt2 = new MyThreadM("Child #2");
        MyThreadM mt3 = new MyThreadM("Child #3");

        do {
            //...
        } while (mt1.count < 10 ||
                mt2.count < 10 ||
                mt3.count < 10);

        System.out.println("Main thread ending.");
    }
}
```

# Řízené ukončení činnosti vláken II

- Testováním „živosti“ vlákna metodou `boolean isAlive()`
  - Virtuální stroj nastaví hodnotu `false` při skončení vlákna

```
class MoreThreads_Alive {
    public static void main(String args[]) {
        System.out.println("Main thread starting.");

        MyThreadA mt1 = new MyThreadA("Child #1");
        MyThreadA mt2 = new MyThreadA("Child #2");
        MyThreadA mt3 = new MyThreadA("Child #3");

        do {
            //...
        } while (mt1.thrd.isAlive() ||
                mt2.thrd.isAlive() ||
                mt3.thrd.isAlive());

        System.out.println("Main thread ending.");
    }
}
```

# Čekání na ukončení činnosti vláken

- Metoda `void join()` čeká na ukončení vlákna

```
class MoreThreads_Join {
    public static void main(String args[]) {
        System.out.println("Main thread starting.");

        MyThreadJ mt1 = new MyThreadJ("Child #1");
        MyThreadJ mt2 = new MyThreadJ("Child #2");
        MyThreadJ mt3 = new MyThreadJ("Child #3");

        try {
            mt1.thrd.join();
            System.out.println("Child #1 joined.");
            mt2.thrd.join();
            System.out.println("Child #2 joined.");
            mt3.thrd.join();
            System.out.println("Child #3 joined.");
        }
        catch (InterruptedException exc) {
            System.out.println("Main thread interrupted.");
        }
        System.out.println("Main thread ending.");
    }
}
```



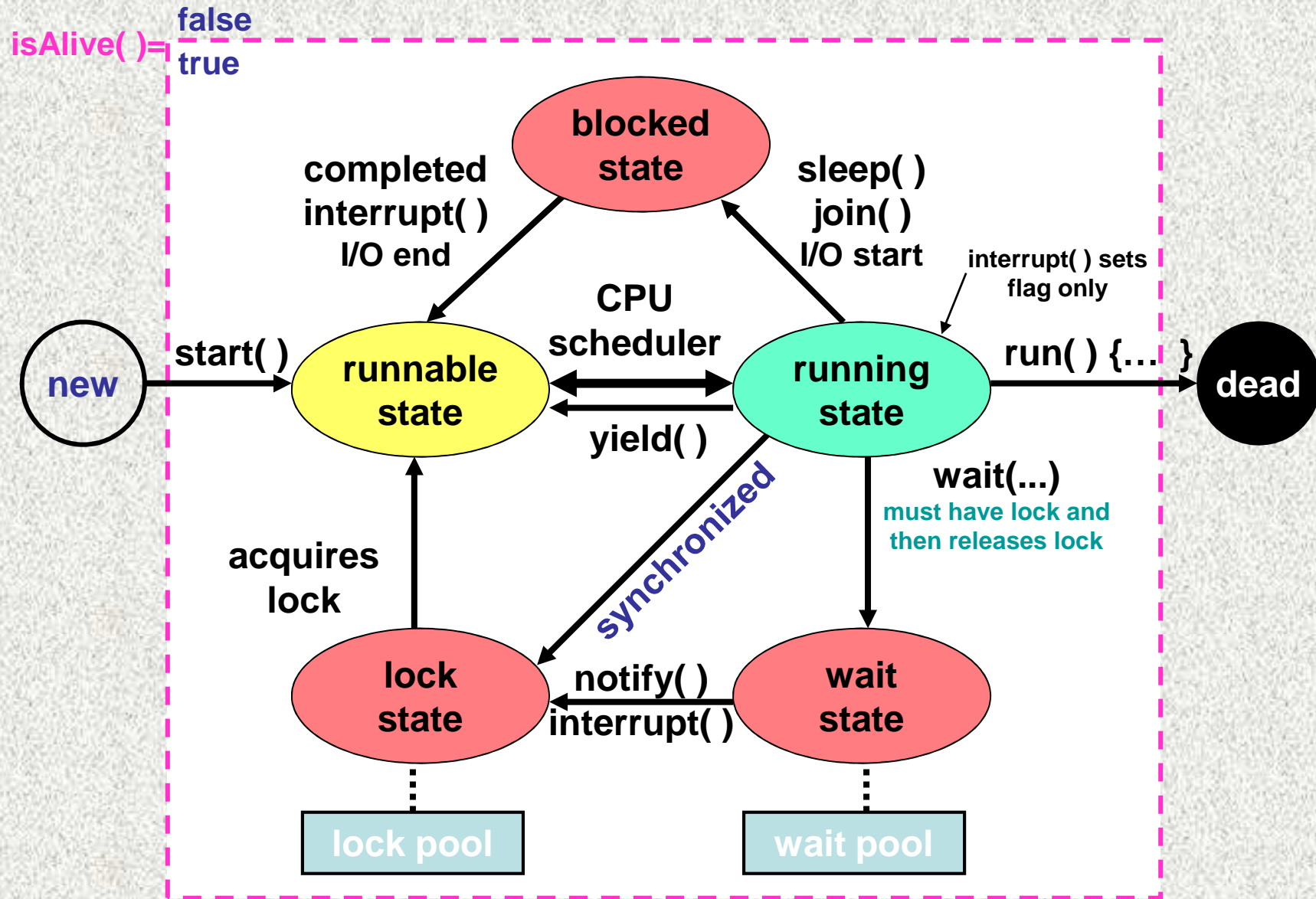
# Stavy vláken

- 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 vlákn



# Priority vláken

Jsou-li běhuschopná dvě vlákna, pak je řízení předáno tomu vláknku, 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, 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

# Priorita vláken, příklad

- Vstup/výstup má nejvyšší prioritu, kdykoli přeruší, když je možnost předávání dat
  1. Vlákno provádí výpočet
  2. Vlákno vypisuje okamžitý stav výpočtu, vyšší priorita
    - Okamžitě na žádost přeruší výpočet a skončí program

# Pomocná třída Vstup

```
class Vstup extends Thread {
    public boolean hotovo = false;
    public void run() {
        byte[] pole = new byte[10];
        Thread.currentThread().setPriority(MAX_PRIORITY);
        while (hotovo == false) {
            try {
                System.in.read(pole);
                if (pole[0] == 'K') {
                    hotovo = true;
                }
            }
            catch (IOException e) {
                System.out.println("Chyba vstupu");
                hotovo = true;
            }
        }
    }
}
```

nastavení nejvyšší  
priority

Čte nějaké hodnoty ze vstupu  
a ukládá je do pole b.

# Priorita vláken - čekání na vstup

```
class Priority_input extends Thread {  
    public void run() {  
        long i = 0;  
        while (Vstup.hotovo == false) {  
            System.out.print(i++ + "\n");  
        }  
    }  
}  
  
public static void main(String[] args) {  
    Vstup vlVstup = new Vstup();  
    vlVstup.start();  
    Priority_input vlVypis = new Priority_input();  
    vlVypis.start();  
}  
}
```



nejvyšší priorita



# Synchronizace činnosti vláken

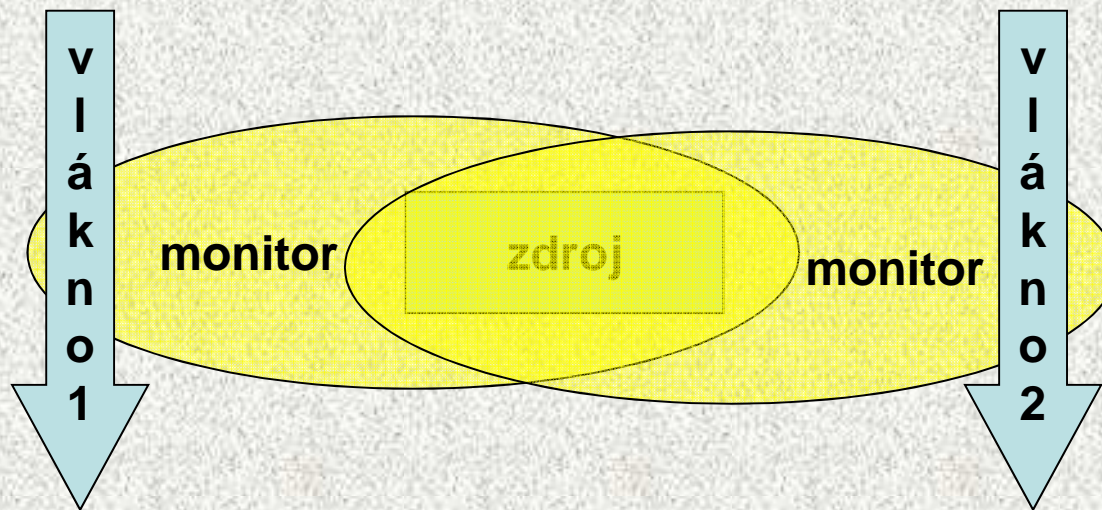
- Vlákna spolupracují, problém sdílení datového prostoru
  - Problém nedeterminovanosti přístupu ke společným prostorům
    - Řešení pomocí proměnné třídy nevhodné!!
- Možné řešení je tzv. **monitor** (kritická sekce)
  - objekt, který vláknu zpřístupní zdroj (paměť, linku,..),
  - v daném okamžiku aktivně 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 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 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) { ... }`



# Synchronizace metody

```
class SumArray {
    private int sum;

    synchronized int sumArray(int nums[]) {
        sum = 0; // reset sum

        for(int i=0; i<nums.length; i++) {
            sum += nums[i]; // sum the array elements
            System.out.println("Running total for " +
                Thread.currentThread().getName() +
                " is " + sum);
            try {
                Thread.sleep(10); // allow task-switch
            }
            catch(InterruptedException exc) {
                System.out.println("Main thread interrupted.");
            }
        }
        return sum;
    }
}
```

# Synchronizace metody

```
class Sync {  
    public static void main(String args[]) {  
        int a[] = {1, 2, 3, 4, 5};  
  
        MyThreadSy mt1 = new MyThreadSy("Child #1", a);  
        MyThreadSy mt2 = new MyThreadSy("Child #2", a);  
    }  
}
```

# Synchronizace metody

```
class MyThreadSy implements Runnable {
    Thread thrd;
    static SumArray sa = new SumArray();
    int a[];
    int answer;
    // Construct a new thread.
    MyThreadSy(String name, int nums[]) {
        thrd = new Thread(this, name);
        thrd.start(); // start the thread
        a = nums;
    }

    // Begin execution of new thread.
    public void run() {
        int sum;
        System.out.println(thrd.getName() + " starting.");
        answer = sa.sumArray(a);
        System.out.println("Sum for " + thrd.getName() +
            " is " + answer);

        System.out.println(thrd.getName() + " terminating.");
    }
}
```

# Synchronizace příkazu

```
class MyThreadP implements Runnable {
    Thread thrd;
    final static SumArrayP sa = new SumArrayP(); // synchronization this object!!
    int a[];
    int answer;
    // Construct a new thread.
    MyThreadP(String name, int nums[]) {
        thrd = new Thread(this, name);
        thrd.start(); // start the thread
        a = nums;
    }
    // Begin execution of new thread.
    public void run() {
        int sum;

        System.out.println(thrd.getName() + " starting.");
        //*****
        // synchronize calls to sumArrayP()
        synchronized(sa) {
            answer = sa.sumArrayP(a);
        }
        //*****
        System.out.println("Sum for " + thrd.getName() +
            " is " + answer);

        System.out.println(thrd.getName() + " terminating.");
    }
}
```

# Komunikace mezi vlákny

- Uvnitř synchronizované metody lze volat metody, které umožní ovládat komunikaci mezi vlákny:

`void wait()` resp `wait(čas)` – dočasně zastaví vlákno do doby pokynu metodou `notify()` resp. `notifyAll()` nebo zastaví vlákno na určenou dobu, tím se uvolní monitor pro ostatní vlákna

`void notify()` – probouzí pozastavené vlákno metodou `wait()`, není určeno které, a převezme monitor

`void notifyAll()` – probouzí všechna vlákna pozastavená metodou `wait()`, monitoru se zmocní vlákno s nejvyšší prioritou

# Komunikace mezi vlákny

Aplikace, která bude střídavě volat vlákna – „linearizace“

```
class ThreadCom {  
    public static void main(String args[]) {  
        TickTock tt = new TickTock();  
        MyThreadT mt1 = new MyThreadT("Tick", tt);  
        MyThreadT mt2 = new MyThreadT("Tock", tt);  
    }  
}
```

```
run:  
Tick Tock  
Tick Tock  
Tick Tock  
Tick Tock  
Tick Tock
```



# Komunikace mezi vlákny

```
class TickTock {
    synchronized void tick(boolean running) {
        if(!running) { // stop the clock
            notify(); // notify any waiting threads
            return;
        }

        System.out.print("Tick ");
        notify(); // let tock() run
        try {
            wait(); // wait for tock() to complete
        }
        catch(InterruptedException exc) {
            System.out.println("Thread interrupted.");
        }
    }

    synchronized void tock(boolean running) { // dtto
    }
}
```

# Komunikace mezi vlákny

```
class MyThreadT implements Runnable {
    Thread thrd;
    TickTock ttOb;

    // Construct a new thread.
    MyThreadT(String name, TickTock tt) {
        thrd = new Thread(this, name);
        ttOb = tt;
        thrd.start(); // start the thread
    }

    // Begin execution of new thread.
    public void run() {

        if(thrd.getName().compareTo("Tick") == 0) {
            for(int i=0; i<5; i++) ttOb.tick(true);
            ttOb.tick(false);
        }
        else {
            for(int i=0; i<5; i++) ttOb.tock(true);
            ttOb.tock(false);
        }
    }
}
```

# Porušená komunikace mezi vlákny

```
class TickTockB {  
  
    synchronized void tick(boolean running) {  
        if(!running) { // stop the clock  
            return;  
        }  
  
        System.out.print("Tick ");  
    }  
  
    synchronized void tock(boolean running) {  
        if(!running) { // stop the clock  
            return;  
        }  
  
        System.out.println("Tock");  
    }  
}
```

# Komunikace mezi vlákny, příklad

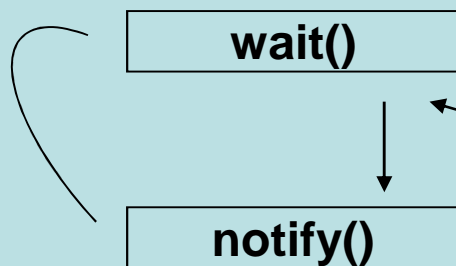
## Vložit do Boxu

### Není obsazeno

- 1) „vložit do boxu“
- 2) `obsazeno = !obsazeno`
- 3) `notify()` zakazníka

### Je obsazeno

- 1) `wait()`



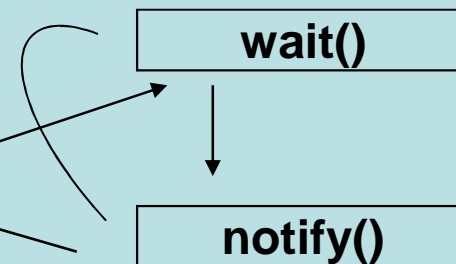
## Vyjmout z Boxu

### Je obsazeno

- 1) „vyjmout z boxu“
- 2) `obsazeno = !obsazeno`
- 3) `notify()` prodavače

### Není obsazeno

- 1) `wait()`



# Komunikace mezi vlákny, příklad

```
public class Synchro_Zak_Prodavac {  
  
    public static void main(String[] args) {  
        Box f = new Box();  
        Vydavatel t1 = new Vydavatel(f);  
        Zakaznik t2 = new Zakaznik(f);  
    }  
}
```

# Komunikace mezi vlákny, příklad

```
class Vydavatel implements Runnable {
    Box f;
    Thread t;
    Vydavatel(Box f) {
        this.f = f;
        t = new Thread(this, "Vydavatel");
        t.start();
    }
    public void run() {
        for (int i = 0; i < 5; i++) {
            f.vlozit(i);
        }
    }
}

class Zakaznik implements Runnable {
    Box f;
    Thread t;
    Zakaznik(Box f) {
        this.f = f;
        t = new Thread(this, "Zakaznik");
        t.start();
    }
    public void run() {
        for (int i = 0; i < 5; i++) {
            f.vyjmout();
        }
    }
}
```

# Komunikace mezi vlákny, příklad

```
class Box {
    int cislo;
    boolean obsazeno = false;
    synchronized int vyjmout() {
        if (!obsazeno) {
            try {
                wait(); //zastavi toto vlakno,jeho probuzeni metodou notify() jinym vlaknem
            } catch (InterruptedException v) {
                System.out.println("Vyjmout: (InterruptedException ");
            }
        }
        obsazeno = !obsazeno;
        System.out.println( Thread.currentThread().getName() + " vyjme: " + cislo );
        notify(); // spust vlakno, ktere pozastavilo toto vlakno
        return cislo; // výběr
    }

    synchronized void vlozit(int cislo) {
        if (obsazeno) {
            try {
                wait(); // zastavi toto vlakno,jeho probuzeni metodou notify() jinym vlaknem
            } catch (InterruptedException v) {
                System.out.println("Vlozit: (InterruptedException ");
            }
        }
        this.cislo = cislo; // vložení
        obsazeno = !obsazeno;
        System.out.println( Thread.currentThread().getName() + " vlozi " + cislo );
        notify(); // spust vlakno, ktere pozastavilo toto vlakno
    }
}
```

# Použití hlavního vlákna

```
class UseMain {
    public static void main(String args[]) {
        Thread thrd;

        // Get the main thread.
        thrd = Thread.currentThread();

        // Display main thread's name.
        System.out.println("Main thread is called: " + thrd.getName());

        // Display main thread's priority.
        System.out.println("Priority: " + thrd.getPriority());

        System.out.println();

        // Set the name and priority.
        System.out.println("Setting name and priority.\n");
        thrd.setName("Thread #1");
        thrd.setPriority(Thread.NORM_PRIORITY+3);

        System.out.println("Main thread is now called: " + thrd.getName());

        System.out.println("Priority is now: " + thrd.getPriority());
    }
}
```



