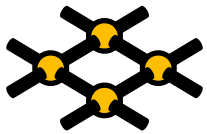
Vláknové programování
část VI

Lukáš Hejmánek, Petr Holub

`{xhejtman,hopet}@ics.muni.cz`



Laboratoř pokročilých síťových technologií

PV192

2012–03–27

Přehled přednášky

ThreadPoolExecutors Revisited

Uváznutí

Optimalizace výkonu

Měření

(příklady povětšinou převzaty z JCiP, Goetz)

ThreadFactory

- TPE vytváří vlákna pomocí ThreadFactory
  - metoda **`newThread`**
  - default ThreadFactory: nedémonická, bez speciálních nastavení
- Možnost předefinovat, jak se budou vytvářet vlákna
  - nastavení pojmenování vláken
  - vlastní třída vytvářených vláken (statistiky, ladění)
  - specifikace vlastního **`UncaughtExceptionHandler`**
  - nastavení priorit (raději nedělat)
  - nastavení démonického stavu (raději nedělat)
  - v případě použití bezpečnostních politik (security policies) lze použít **`privilegedThreadFactory`**
    - podědění oprávnění, **`AccessControlContext`** a **`contextClassLoader`** od vlákna vytvářejícího **`privilegedThreadFactory`**, nikoli od vlákna volajícího **`execute`**/**`submit`** (default)

ThreadFactory

```
public class MyThreadFactory implements ThreadFactory {
    private final String poolName;
    class MyAppThread extends Thread {
        public MyAppThread(Runnable runnable, String poolName) {
            super(runnable, poolName);
        }
    }

    public MyThreadFactory(String poolName) {
        this.poolName = poolName;
    }

    public Thread newThread(Runnable runnable) {
        return new MyAppThread(runnable, poolName);
    }
}
```

Modifikace Executorů za běhu

- settery a gettery na různé vlastnosti
- možnost přetypování executorů vyrobených přes factory metody (kromě `newSingleThreadExecutor`) na `ThreadPoolExecutor`
- omezení modifikací
  - nechceme nechat vývojáře šťourat do svých TPE
  - factory metoda `Executor.unconfigurableExecutorService`
    - bere `ExecutorService`
    - vrací omezenou `ExecutorService` pomocí `DelegatedExecutorService`, která rozšiřuje `AbstractExecutorService`
  - využíváno metodou `newSingleThreadExecutor` (vrací omezený Executor – ačkoli implementace ve skutečnosti používá TPE s jediným vláknem)

Modifikace TPE

- Háčky pro modifikace
  - `beforeExecute`
  - `afterExecute`
  - `terminated`
- Např. sběr statistik

Modifikace TPE

```
public class TimingThreadPool extends ThreadPoolExecutor {

    public TimingThreadPool() {
        super(1, 1, 0L, TimeUnit.SECONDS, null);
    }

    private final ThreadLocal<Long> startTime = new ThreadLocal<Long>();
    private final Logger log = Logger.getLogger("TimingThreadPool");
    private final AtomicLong numTasks = new AtomicLong();
    private final AtomicLong totalTime = new AtomicLong();

    protected void beforeExecute(Thread t, Runnable r) {
        super.beforeExecute(t, r);
        log.fine(String.format("Thread %s: start %s", t, r));
        startTime.set(System.nanoTime());
    }
```

Modifikace TPE

```
    protected void afterExecute(Runnable r, Throwable t) {
        try {
            long endTime = System.nanoTime();
            long taskTime = endTime - startTime.get();
            numTasks.incrementAndGet();
            totalTime.addAndGet(taskTime);
            log.fine(String.format("Thread %s: end %s, time=%dns",
                    t, r, taskTime));
        } finally {
            super.afterExecute(r, t);
        }
    }

    protected void terminated() {
        try {
            log.info(String.format("Terminated: avg time=%dns",
                    totalTime.get() / numTasks.get()));
        } finally {
            super.terminated();
        }
    }
}
```

Možné řešení kvízu

```
public class DynamicTPE {
    static class CustomTPE extends ThreadPoolExecutor {
        final int userProvidedPoolSize;
        final int queueCapacity;

        CustomTPE(int corePoolSize, int maximumPoolSize,
                  long keepAliveTime, TimeUnit unit,
                  BlockingQueue<Runnable> workQueue) {
            super(corePoolSize, maximumPoolSize,
                    keepAliveTime, unit, workQueue);
            userProvidedPoolSize = corePoolSize;
            queueCapacity = workQueue.size()
                    + workQueue.remainingCapacity();
        }

        @Override public Future<?> submit(Runnable task) {
            autoAdjustCorePoolSize();
            return super.submit(task);
        }

        @Override synchronized public void setCorePoolSize
                (int corePoolSize) {
            super.setCorePoolSize(corePoolSize);
        }
        @Override synchronized public void setMaximumPoolSize
                (int maximumPoolSize) {
            super.setMaximumPoolSize(maximumPoolSize);
        }
```

Možné řešení kvízu

```
        synchronized private void autoAdjustCorePoolSize() {
            final int queueRemaining = getQueue().remainingCapacity();
            final int extension;
            if (queueRemaining < 0.25 * queueCapacity) {
                extension = (int) Math.round(0.25 * queueCapacity
                        - queueRemaining);
                if (getCorePoolSize() + extension < getMaximumPoolSize())
                    setCorePoolSize(getCorePoolSize() + extension);
                else
                    setCorePoolSize(getMaximumPoolSize());
            }
            else if (queueRemaining > 0.75 * queueCapacity) {
                extension = (int) Math.round(queueRemaining
                        - 0.75 * queueCapacity);
                if (getCorePoolSize() - extension > userProvidedPoolSize)
                    setCorePoolSize(getCorePoolSize() - extension);
                else
                    setCorePoolSize(userProvidedPoolSize);
            }
        }

    }
```

Možné řešení kvízu

```
1   public static void main(String[] args) {
        ThreadPoolExecutor tpe = new CustomTPE(1, 10, 60, TimeUnit.SECONDS, new LinkedBlockingQue
3       tpe.setRejectedExecutionHandler(new ThreadPoolExecutor.CallerRunsPolicy());
        ArrayList<Future> taskList = new ArrayList<Future>();
5       Runnable r = new Runnable() {
            public void run() {
7               try {
                    Thread.sleep(30);
9               } catch (InterruptedException e) {
                }
11              System.out.println("bla");
            }
13      };
        for (int i = 0; i < 1000; i++) {
15          Future f = tpe.submit(r);
            taskList.add(f);
17          System.out.println("TPE corepoolsize: " + tpe.getCorePoolSize());
            System.out.println("TPE poolsize: " + tpe.getPoolSize());
19      }
        for (Future future : taskList) {
21          try {
                Object o = future.get();
23          } catch (InterruptedException e) {
            } catch (ExecutionException e) {
25          }
        }
27      tpe.shutdown();
    }
```

Možné řešení kvízu

```
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
```

```
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 1
TPE poolsize: 1
TPE corepoolsize: 2
TPE poolsize: 2
TPE corepoolsize: 4
TPE poolsize: 4
TPE corepoolsize: 7
TPE poolsize: 7
TPE corepoolsize: 10
TPE poolsize: 10
TPE corepoolsize: 10
TPE poolsize: 10
bla
bla
```

Možné řešení kvízu

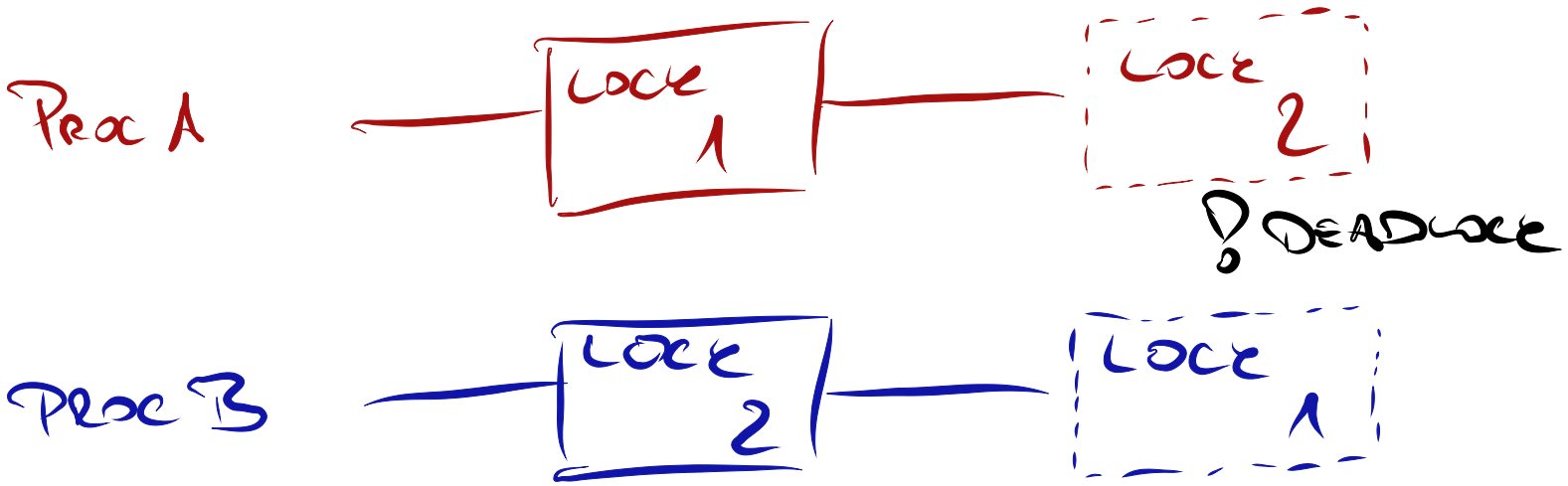
- Stojí nám to za to?
  - drahá synchronizace
  - režije se startováním a ukončováním vláken
  - nestačí vhodně nastavit `corePoolSize` a `allowCoreThreadTimeOut`?

Kompletně vlastní implementace TPE

- Zdrojové kódy:
  - **`http://kickjava.com/src/java/util/concurrent/ThreadPoolExecutor.java.htm`**
  - **`http://kickjava.com/src/java/util/concurrent/ScheduledThreadPoolExecutor.java.htm`**

Deadlock

- Deadlock – uváznutí, smrtelné objetí ;-)
- Vzájemné nekončící čekání na zámky



- Potřeba globálního uspořádání zámků
  - zamykání podle globálního uspořádání
- Možnost využití `Lock.tryLock()`
  - náhodný rovnoměrný back-off
  - náhodný exponenciální back-off
  - nelze použít s monitory
- Řešení deadlocků runtimem (ne v Javě)

Deadlock

```
    public static void transferMoney(Account fromAccount,
2                                    Account toAccount,
                                     DollarAmount amount)
4           throws InsufficientFundsException {
        synchronized (fromAccount) {
6           synchronized (toAccount) {
                if (fromAccount.getBalance().compareTo(amount) < 0)
8                   throw new InsufficientFundsException();
                else {
10                  fromAccount.debit(amount);
                    toAccount.credit(amount);
12              }
            }
14      }
    }
```

Deadlock

```
    public static void transferMoney(Account fromAccount,
2                                    Account toAccount,
                                     DollarAmount amount)
4           throws InsufficientFundsException {
        synchronized (fromAccount) {
6           synchronized (toAccount) {
                if (fromAccount.getBalance().compareTo(amount) < 0)
8                   throw new InsufficientFundsException();
                else {
10                  fromAccount.debit(amount);
                    toAccount.credit(amount);
12              }
            }
14      }
    }
```

Deadlock

```
    public void transferMoney(final Account fromAcct,
                              final Account toAcct,
                              final DollarAmount amount)
            throws InsufficientFundsException {
        class Helper {
            public void transfer() throws InsufficientFundsException {
                if (fromAcct.getBalance().compareTo(amount) < 0)
                    throw new InsufficientFundsException();
                else {
                    fromAcct.debit(amount);
                    toAcct.credit(amount);
                }
            }
        }
        int fromHash = System.identityHashCode(fromAcct);
        int toHash = System.identityHashCode(toAcct);
```

- `System.identityHashCode(o)` může vrátit pro dva různé objekty identický hash
  - řídký problém

Deadlock

```
        if (fromHash < toHash) {
2           synchronized (fromAcct) {
                synchronized (toAcct) {
4                   new Helper().transfer();
                }
6           }
        } else if (fromHash > toHash) {
8           synchronized (toAcct) {
                synchronized (fromAcct) {
10                  new Helper().transfer();
                }
12          }
        } else {
14          synchronized (tieLock) {
                synchronized (fromAcct) {
16                  synchronized (toAcct) {
                        new Helper().transfer();
18                  }
                }
20          }
        }
```

Deadlock

```
    private static Random rnd = new Random();
2
    public boolean transferMoney(Account fromAcct,
4                                Account toAcct,
                                 DollarAmount amount,
6                                long timeout,
                                 TimeUnit unit)
8           throws InsufficientFundsException, InterruptedException {
        long fixedDelay = getFixedDelayComponentNanos(timeout, unit);
10      long randMod = getRandomDelayModulusNanos(timeout, unit);
        long stopTime = System.nanoTime() + unit.toNanos(timeout);
```

Deadlock

```
        while (true) {
            if (fromAcct.lock.tryLock()) {
                try {
                    if (toAcct.lock.tryLock()) {
                        try {
                            if (fromAcct.getBalance().compareTo(amount) < 0)
                                throw new InsufficientFundsException();
                            else {
                                fromAcct.debit(amount);
                                toAcct.credit(amount);
                                return true;
                            }
                        } finally {
                            toAcct.lock.unlock();
                        }
                    }
                } finally {
                    fromAcct.lock.unlock();
                }
            }
            if (System.nanoTime() < stopTime)
                return false;
            NANOSECONDS.sleep(fixedDelay + rnd.nextLong() % randMod);
        }
```

Otevřená volání

```
1   class Taxi {
        @GuardedBy("this") private Point location, destination;
3       private final Dispatcher dispatcher;

5       public Taxi(Dispatcher dispatcher) {
            this.dispatcher = dispatcher;
7       }

9       public synchronized Point getLocation() {
            return location;
11      }

13      public synchronized void setLocation(Point location) {
            this.location = location;
15          if (location.equals(destination))
                dispatcher.notifyAvailable(this);
17      }

19      public synchronized Point getDestination() {
            return destination;
21      }

23      public synchronized void setDestination(Point destination) {
            this.destination = destination;
25      }
    }
```

Otevřená volání

```
class Dispatcher {
    @GuardedBy("this") private final Set<Taxi> taxis;
    @GuardedBy("this") private final Set<Taxi> availableTaxis;

    public Dispatcher() {
        taxis = new HashSet<Taxi>();
        availableTaxis = new HashSet<Taxi>();
    }

    public synchronized void notifyAvailable(Taxi taxi) {
        availableTaxis.add(taxi);
    }

    public synchronized Image getImage() {
        Image image = new Image();
        for (Taxi t : taxis)
            image.drawMarker(t.getLocation());
        return image;
    }
}
```

Otevřená volání

```
1   class Taxi {
2       @GuardedBy("this") private Point location, destination;
3       private final Dispatcher dispatcher;
4
5       public Taxi(Dispatcher dispatcher) {
6           this.dispatcher = dispatcher;
7       }
8
9       public synchronized Point getLocation() {
10          return location;
11      }
12
13      public synchronized void setLocation(Point location) {
14          this.location = location;
15          if (location.equals(destination))
16              dispatcher.notifyAvailable(this);
17      }
18
19      public synchronized Point getDestination() {
20          return destination;
21      }
22
23      public synchronized void setDestination(Point destination) {
24          this.destination = destination;
25      }
26  }
```

```
1   class Dispatcher {
2       @GuardedBy("this") private final Set<Taxi> taxis;
3       @GuardedBy("this") private final Set<Taxi> availableTaxis;
4
5       public Dispatcher() {
6           taxis = new HashSet<Taxi>();
7           availableTaxis = new HashSet<Taxi>();
8       }
9
10      public synchronized void notifyAvailable(Taxi taxi) {
11          availableTaxis.add(taxi);
12      }
13
14      public synchronized Image getImage() {
15          Image image = new Image();
16          for (Taxi t : taxis)
17              image.drawMarker(t.getLocation());
18          return image;
19      }
20  }
```

- **`setLocation` → `notifyAvailable`**
- **`getImage` → `getLocation`**

Otevřená volání

- Otevřené volání (open call)
  - volání metody, kdy volající nedrží žádný zámek
  - preferovaný způsob
- Převod na otevřené volání
  - synchronizace by měla být omezena na lokální proměnné
  - problém se zachováním sémantiky
- Možnost globálního zámku

Otevřená volání

```
class Taxi {
    @GuardedBy("this") private Point location, destination;
    private final Dispatcher dispatcher;

    public Taxi(Dispatcher dispatcher) { this.dispatcher = dispatcher; }

    public synchronized Point getLocation() { return location; }

    public void setLocation(Point location) {
        boolean reachedDestination;
        synchronized (this) {
            this.location = location;
            reachedDestination = location.equals(destination);
        }
        if (reachedDestination)
            dispatcher.notifyAvailable(this);
    }

    public synchronized Point getDestination() { return destination; }

    public synchronized void setDestination(Point destination) {
        this.destination = destination;
    }
}
```

Otevřená volání

```
    class Dispatcher {
2       @GuardedBy("this") private final Set<Taxi> taxis;
        @GuardedBy("this") private final Set<Taxi> availableTaxis;
4
        public Dispatcher() {
6           taxis = new HashSet<Taxi>();
            availableTaxis = new HashSet<Taxi>();
8       }

10      public synchronized void notifyAvailable(Taxi taxi) {
            availableTaxis.add(taxi);
12      }

14      public Image getImage() {
            Set<Taxi> copy;
16          synchronized (this) {
                copy = new HashSet<Taxi>(taxis);
18          }
            Image image = new Image();
20          for (Taxi t : copy)
                image.drawMarker(t.getLocation());
22          return image;
        }
24  }
```

Hladovění

- Hladovění (starvation) nastává, pokud je vláknu neustále odpírán zdroj, který je potřeba k dalšímu postupu
  - běžné použití zámků je férové
  - problém při nastavování priorit

```
        t.setPriority(Thread.MIN_PRIORITY); // 1
2       t.setPriority(Thread.NORM_PRIORITY); // 5
        t.setPriority(Thread.MAX_PRIORITY); // 10
```

    - problém platformové závislosti priorit
    - možná pomoc pro zvýšení responsivity GUI
  - typické pokusy o „řešení“ problémů

```
1       Thread.yield();
        Thread.sleep(100);
```

Další typy uváznutí

- Livelock
  - uváznutí, při němž se vlákno (aktivně) snaží o činnosti, která opakovaně selhává
  - náhodnostní exponenciální back-off
- Ztracené zprávy
  - `o.wait()` a `o.notify()` resp. `o.notifyAll` nemají mechanismus zdržení notifikace
  - pokud vlákno usne na `o.wait()` později, než mělo být notifikováno přes `o.notify`, nikdy se nevzbudí

Hledání problémů

- Výpis stavu JVM
  - `SIGQUIT` na unixech (ev. Ctrl-\ pokud mapuje na `SIGQUIT`)
  - Ctrl-Break na Windows

Hledání problémů

```
    public static void main(String[] args) {
2       final Object a = new Object();
        final Object b = new Object();
4
        Thread t1 = new Thread(new Runnable() {
6           public void run() {
                try {
8                   synchronized (a) {
                        Thread.sleep(1000);
10                      System.out.println("t1 - cekam na b");
                        synchronized (b) {
12                          System.out.println("t1 - jsem zde");
                        }
14                  }
                } catch (InterruptedException e) {
16              }
            }
18      });
```

Hledání problémů

```
        Thread t2 = new Thread(new Runnable() {
2           public void run() {
                try {
4                   synchronized (b) {
                        Thread.sleep(1000);
6                       System.out.println("t2 - cekam na a");
                        synchronized (a) {
8                           System.out.println("t2 - jsem zde");
                        }
10                  }
                } catch (InterruptedException e) {
12              }
            }
14      });

16      t1.start();
        t2.start();
```

Hledání problémů

```
$ java IntentionalDeadlock
t2 - cekam na a
t1 - cekam na b
2010-04-22 11:46:25
Full thread dump Java HotSpot(TM) Client VM (16.2-b04 mixed mode, sharing):

"DestroyJavaVM" prio=6 tid=0x020b1000 nid=0x164c waiting on condition [0x00000000]
   java.lang.Thread.State: RUNNABLE

"Thread-1" prio=6 tid=0x02149800 nid=0x1b4c waiting for monitor entry [0x0480f000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at IntentionalDeadlock$2.run(IntentionalDeadlock.java:35)
        - waiting to lock <0x243e6928> (a java.lang.Object)
        - locked <0x243e6930> (a java.lang.Object)
        at java.lang.Thread.run(Unknown Source)

"Thread-0" prio=6 tid=0x02146c00 nid=0x1a38 waiting for monitor entry [0x0477f000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at IntentionalDeadlock$1.run(IntentionalDeadlock.java:20)
        - waiting to lock <0x243e6930> (a java.lang.Object)
        - locked <0x243e6928> (a java.lang.Object)
        at java.lang.Thread.run(Unknown Source)
```

Hledání problémů

```
"Low Memory Detector" daemon prio=6 tid=0x02121400 nid=0xbd8 runnable [0x00000000]
   java.lang.Thread.State: RUNNABLE

"CompilerThread0" daemon prio=10 tid=0x02119800 nid=0x1708 waiting on condition [0x00000000]
   java.lang.Thread.State: RUNNABLE

"Attach Listener" daemon prio=10 tid=0x02118400 nid=0x13d0 runnable [0x00000000]
   java.lang.Thread.State: RUNNABLE

"Signal Dispatcher" daemon prio=10 tid=0x02115400 nid=0x5a0 waiting on condition [0x00000000]
   java.lang.Thread.State: RUNNABLE
```

```
Heap
 def new generation   total 4928K, used 466K [0x243b0000, 0x24900000, 0x29900000)
  eden space 4416K,  10% used [0x243b0000, 0x24424828, 0x24800000)
  from space 512K,    0% used [0x24800000, 0x24800000, 0x24880000)
  to   space 512K,    0% used [0x24880000, 0x24880000, 0x24900000)
 tenured generation   total 10944K, used 0K [0x29900000, 0x2a3b0000, 0x343b0000)
   the space 10944K,   0% used [0x29900000, 0x29900000, 0x29900200, 0x2a3b0000)
 compacting perm gen  total 12288K, used 42K [0x343b0000, 0x34fb0000, 0x383b0000)
   the space 12288K,   0% used [0x343b0000, 0x343ba960, 0x343baa00, 0x34fb0000)
    ro space 10240K,  51% used [0x383b0000, 0x388dae00, 0x388dae00, 0x38db0000)
    rw space 12288K,  54% used [0x38db0000, 0x394472d8, 0x39447400, 0x399b0000)
```

Hledání problémů

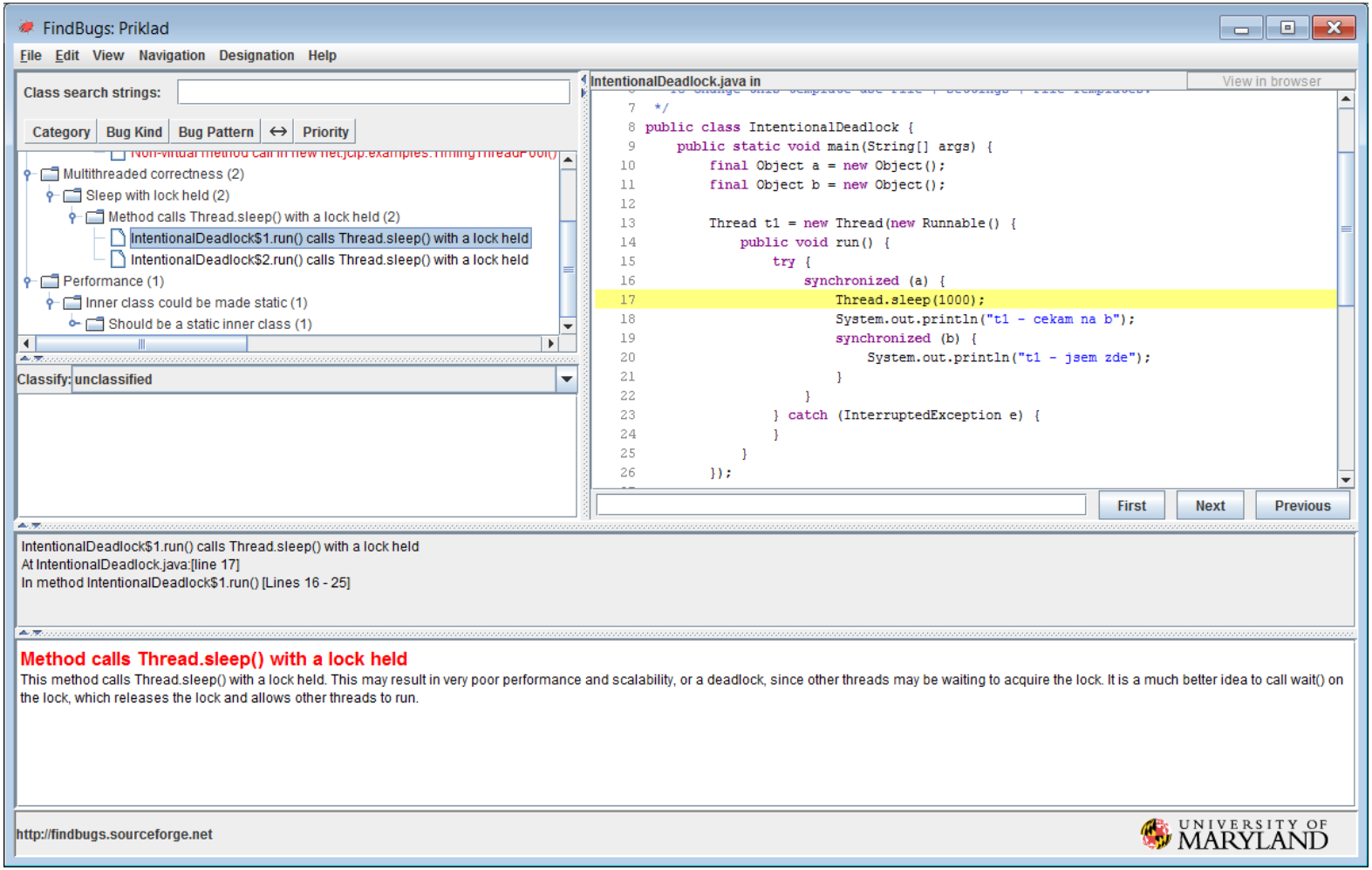
```
Found one Java-level deadlock:
============================
"Thread-1":
  waiting to lock monitor 0x020d53ac (object 0x243e6928, a java.lang.Object),
  which is held by "Thread-0"
"Thread-0":
  waiting to lock monitor 0x020d6c74 (object 0x243e6930, a java.lang.Object),
  which is held by "Thread-1"

Java stack information for the threads listed above:
===================================================
"Thread-1":
        at IntentionalDeadlock$2.run(IntentionalDeadlock.java:35)
        - waiting to lock <0x243e6928> (a java.lang.Object)
        - locked <0x243e6930> (a java.lang.Object)
        at java.lang.Thread.run(Unknown Source)
"Thread-0":
        at IntentionalDeadlock$1.run(IntentionalDeadlock.java:20)
        - waiting to lock <0x243e6930> (a java.lang.Object)
        - locked <0x243e6928> (a java.lang.Object)
        at java.lang.Thread.run(Unknown Source)

Found 1 deadlock.
```

Statická analýza kódu

- FindBugs
  `http://findbugs.sourceforge.net/`



Anotace

- Vícečlenný tým programátorů – předávání myšlenek
  - komentáře v kódy
  - anotace
    - anotace se dají použít i pro statickou analýzu kódu

```
import net.jcip.annotations.GuardedBy;
// http://www.javaconcurrencyinpractice.com/jcip-annotations.jar
```

Anotace

- Anotace tříd
  - `@Immutable`
  - `@ThreadSafe`
  - `@NotThreadSafe`
- Anotace polí
  - `@GuardedBy("this")`
    - monitor (intrinsic lock) na this
  - `@GuardedBy("jmenoPole")`
    - explicitní zámek na `jmenoPole` pokud je potomkem `Lock`
    - jinak monitor na `jmenoPole`
  - `@GuardedBy("JmenoTridy.jmenoPole")`
    - obdobné, odkazuje se statické pole jiné třídy
  - `@GuardedBy("jmenoMetody()")`
    - metoda `jmenoMetody()` vrací zámek
  - `@GuardedBy("JmenoTridy.class")`
    - literál třídy (objekt) pro pojmenovanou třídu

Omezování zámků

$$\text{zrychlení} \leq \frac{1}{s + \frac{1-s}{n}}$$

- JVM se snaží dělat
  - eliminaci synchronizací, které nemohou nastat (např. pomocí escape analysis – lokální objekt, který není nikdy publikován na haldu a je tudíž thread-local)
  - kombinace více zámků do jednoho (lock coarsening)
- Zbytečně nesynchronizovat
  - delegace bezpečnosti (thread safety delegation)
  - omezení rozsahu synchronizace (get in – get out principle, např. Taxi/Dispatcher)
  - dělení zámků (lock splitting) – pouze pro **nezávislé** proměnné/objekty
  - ořezávání zámků (lock stripping)
  - RW zámky
- Neprovádět object pooling na jednoduchých objektech
  - `new` je levnější jako `malloc`

```
2       synchronized (new Object()) {
            System.out.println("bleeee");
        }
```

Omezování zámků

$$\text{zrychlení} \leq \frac{1}{s + \frac{1-s}{n}}$$

- JVM se snaží dělat
  - eliminaci synchronizací, které nemohou nastat (např. pomocí escape analysis – lokální objekt, který není nikdy publikován na haldu a je tudíž thread-local)
  - kombinace více zámků do jednoho (lock coarsening)
- Zbytečně nesynchronizovat
  - delegace bezpečnosti (thread safety delegation)
  - omezení rozsahu synchronizace (get in – get out principle, např. Taxi/Dispatcher)
  - dělení zámků (lock splitting) – pouze pro **nezávislé** proměnné/objekty
  - ořezávání zámků (lock stripping)
  - RW zámky
- Neprovádět object pooling na jednoduchých objektech
  - `new` je levnější jako `malloc`

```
2       synchronized (new Object()) {
            System.out.println("bleeee");
        }
```

Omezování zámků

- Podobně jako dělení zámků, ale pro proměnný počet nezávislých proměnných/objektů
- Příklad ořezávání zámků – `ConcurrentHashMap`
  - 16 zámků
  - každý z N hash buckets je chráněný zámkem $N \mod 16$
  - předpokládáme rovnoměrné rozdělení položek mezi kbelíky

  $\Longrightarrow$ 16 paralelních přístupů

  $\Longrightarrow$ přístup k celé kolekci vyžaduje všech 16 zámků
  - rozdělení kumulativních polí do jednotlivých kbelíků

Interakce s JVM při měření

- Problém garbage collection
  - `-verbose:gc`
  - krátká měření: vybrat pouze běhy, v nichž nedošlo ke GC
  - dlouhé běhy: dostatečně dlouhé, aby se přítomnost GC projevila representativně
- Problém HotSpot kompilace
  - `-XX:+PrintCompilation`
  - dostatečný warm-up (minuty!)
  - mohou se vyskytovat rekompilace (optimalizace, nahrání nové třídy která zruší dosavadní předpoklady)
  - housekeeping tasks: oddělení nesouvisejících měření pauzou nebo restartem JVM

Délka zpracování obrázku

velikost obrázku	*čas běhu*
640×480	124,12983930928
1280×720	539,98450298239
1920×1080	1529,02398429008
4096×2160	10210,09238488922

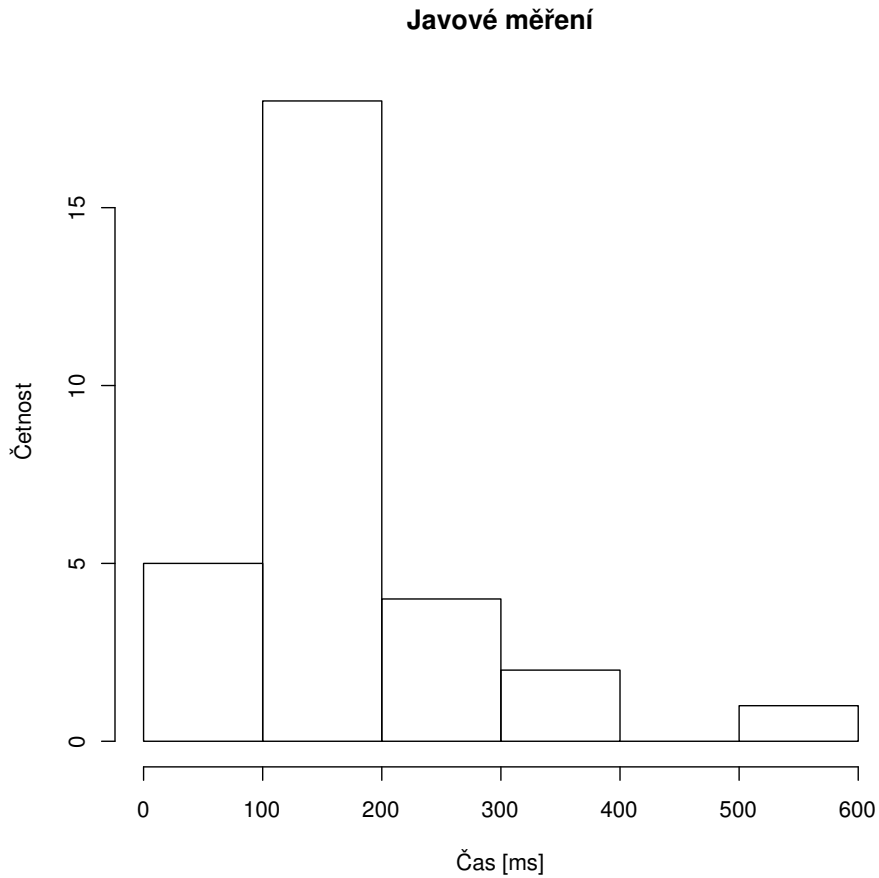
velikost obrázku	*čas běhu*
640 × 480	124,12983930928
1280 × 720	539,98450298239
1920 × 1080	1529,02398429008
4096 × 2160	10210,09238488922

Měříme délku výpočtu v Javě

```
> library(psych)
> runlength <- read.csv(file="java-example.table", head=FALSE, sep=","
> summary(runlength$V1)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  92.08  104.70  108.80  166.80  187.20  594.70
> describe(runlength$V1)
  var  n   mean     sd median trimmed   mad   min    max  range skew k
1   1 30 166.82 113.67 108.78   142.1 20.88 92.08 594.71 502.63 2.14
4.55
     se
1 20.75
```

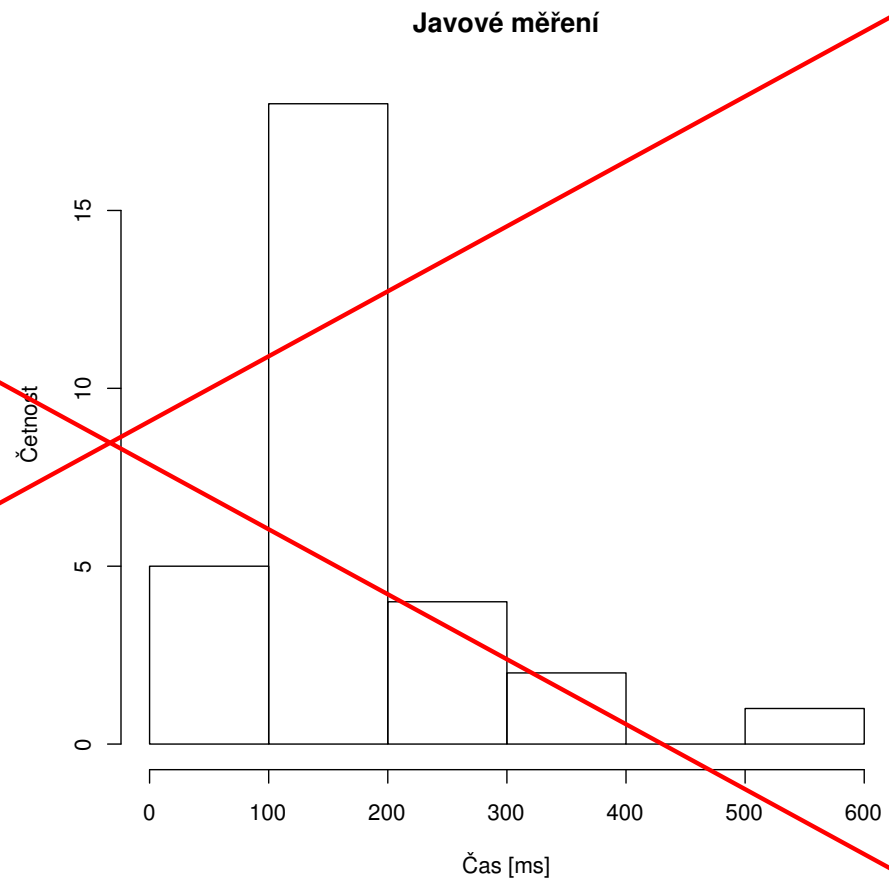
$$N = 30$$
$$\overline{x} = 166{,}82$$
$$sx = 113{,}67$$
$$s_{\overline{x}} = \frac{sx}{\sqrt{N}} = 20{,}75$$
$$t_{0{,}05;29} = 2{,}045$$

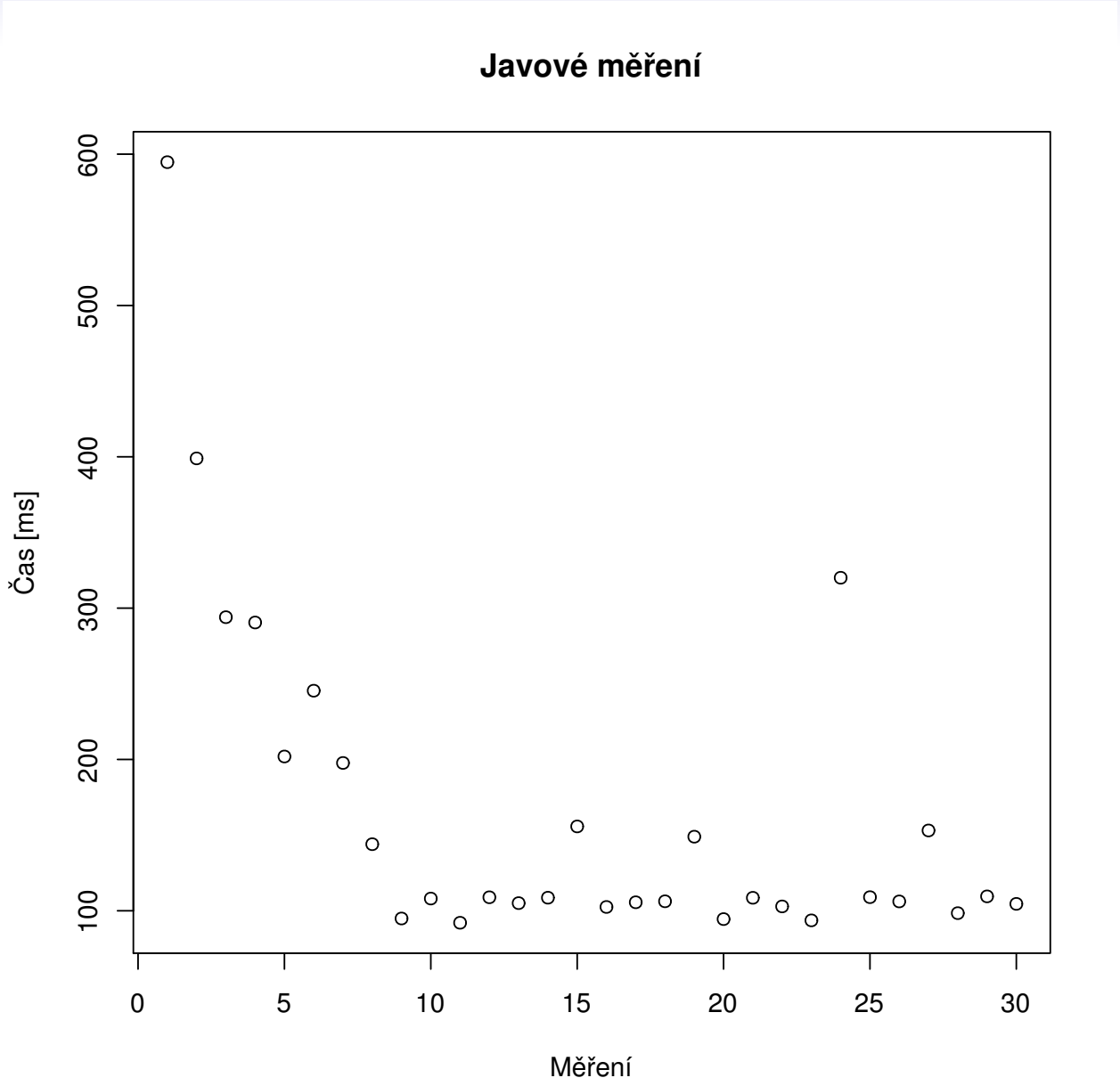
$$\overline{x} \pm t_{0{,}05;N-1} s_{\overline{x}} = 167 \pm 42\,\text{ms}$$

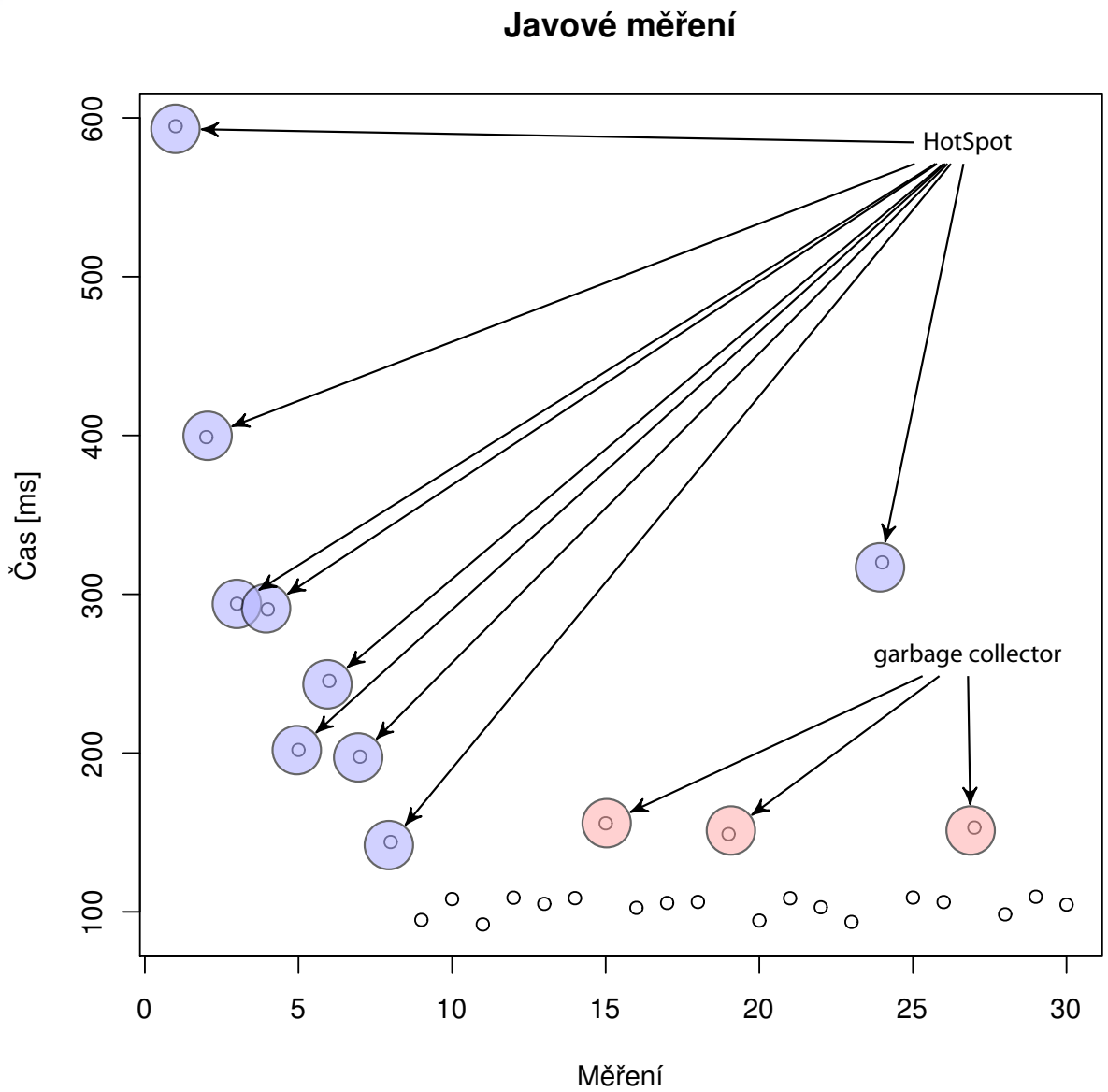


$$N = 30$$
$$\overline{x} = 166{,}82$$
$$sx = 113{,}67$$
$$s_{\overline{x}} = \frac{sx}{\sqrt{N}} = 20{,}75$$
$$t_{0,05;29} = 2{,}045$$

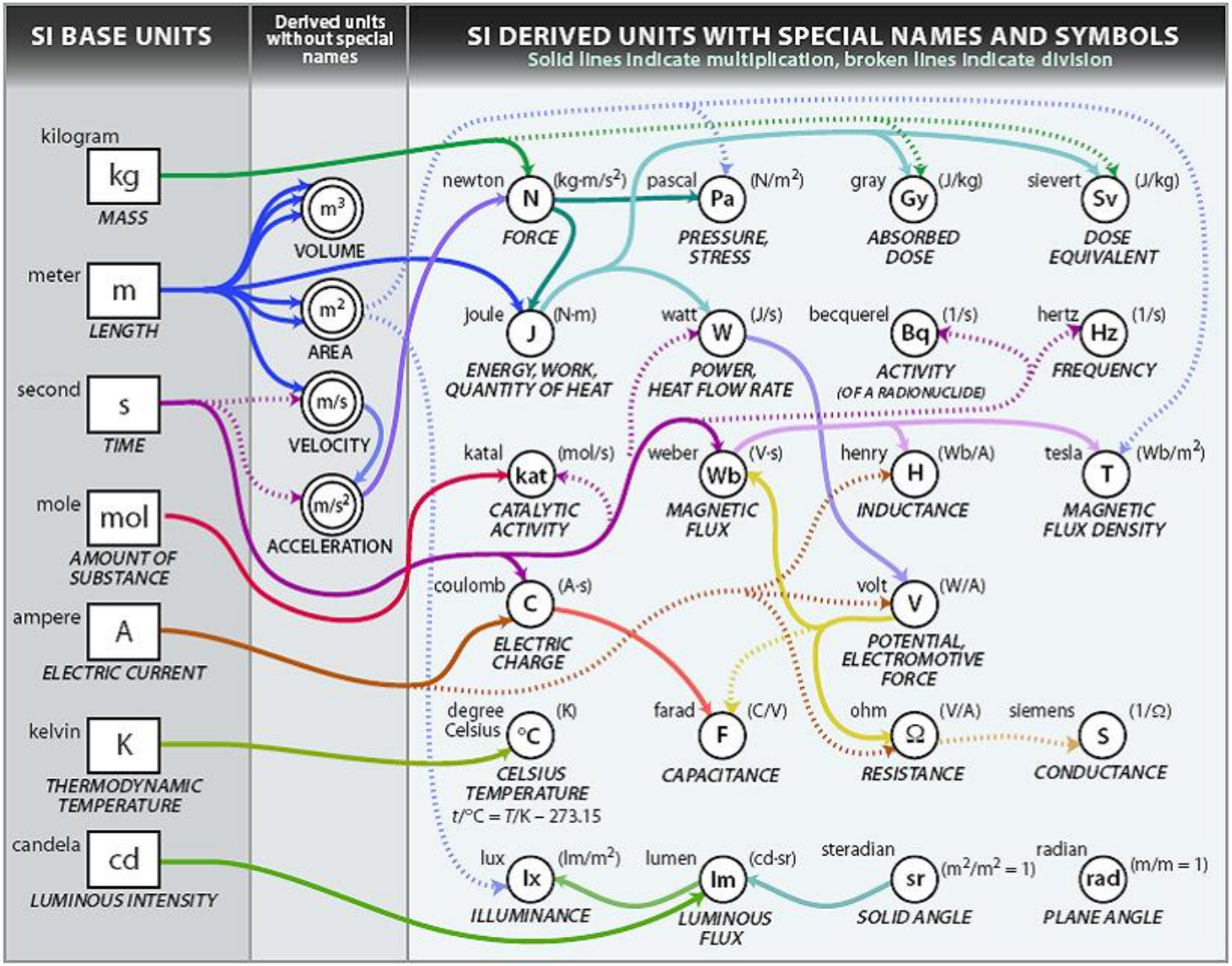
$$\overline{x} \pm t_{0,05;N-1} s_{\overline{x}} = 167 \pm 42\,\text{ms}$$




Javové měření
Čas [ms]
100
200
300
400
500
600
0
5
10
15
20
25
30
Měření


Javové měření
HotSpot
garbage collector
Čas [ms]
100
200
300
400
500
600
0
5
10
15
20
25
30
Měření

Soustava jednotek pro informatiky



Zdroj: **http://www.icrf.nl/Portals/106/SI_units_diagram(1).jpg**

Soustava jednotek pro informatiky

- Předpony nejen speciálně informatické

yocto-	10^{-24}	y	–	–	–
zepto-	10^{-21}	z	–	–	–
atto-	10^{-18}	a	–	–	–
femto-	10^{-15}	f	–	–	–
pico-	10^{-12}	p	–	–	–
nano-	10^{-9}	n	–	–	–
micro-	10^{-6}	μ	–	–	–
milli-	10^{-3}	m	–	–	–
kilo-	10^{3}	k	kibi	2^{10}	Ki
mega-	10^{6}	M	mebi	2^{20}	Mi
giga-	10^{9}	G	gibi	2^{30}	Gi
tera-	10^{12}	T	tebi	2^{40}	Ti
peta-	10^{15}	P	pebi	2^{50}	Pi
exa-	10^{18}	E	exbi	2^{60}	Ei
zetta-	10^{21}	Z	zebi	2^{70}	Zi
yotta-	10^{24}	Y	yobi	2^{80}	Yi

Amendment 2 to “IEC 60027-2: Letter symbols to be used in electrical technology – Part 2: Telecommunications and electronics” (1999)

Výsledky měření

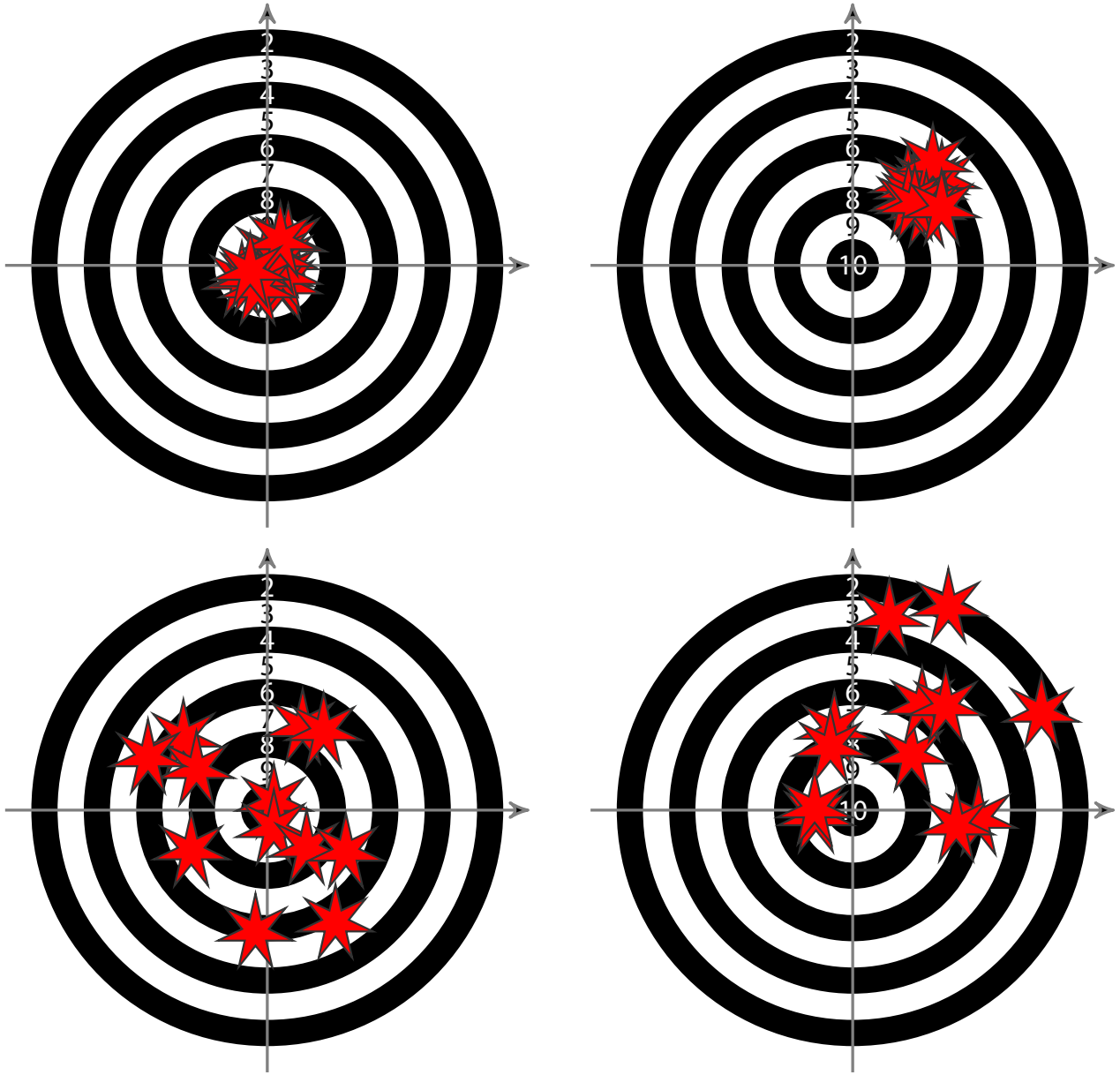
$$x = (\hat{\mu}_x \pm z_x) \text{ [jednotka]}$$

- $\hat{\mu}_x$… nejpravděpodobnější hodnota měřené veličiny
- z_x… interval spolehlivosti / přesnost
- jak tyto věci spočítat / odhadnout?

Chyby měření

- Klasifikace chyb podle místa vzniku
  - instrumentální (přístrojové) chyby
  - metodické chyby
  - teoretické chyby (principy, model)
  - chyby zpracování
- Klasifikace chyb podle původu
  - hrubé (omyly)
  - systematické
  - náhodné

Chyby měření

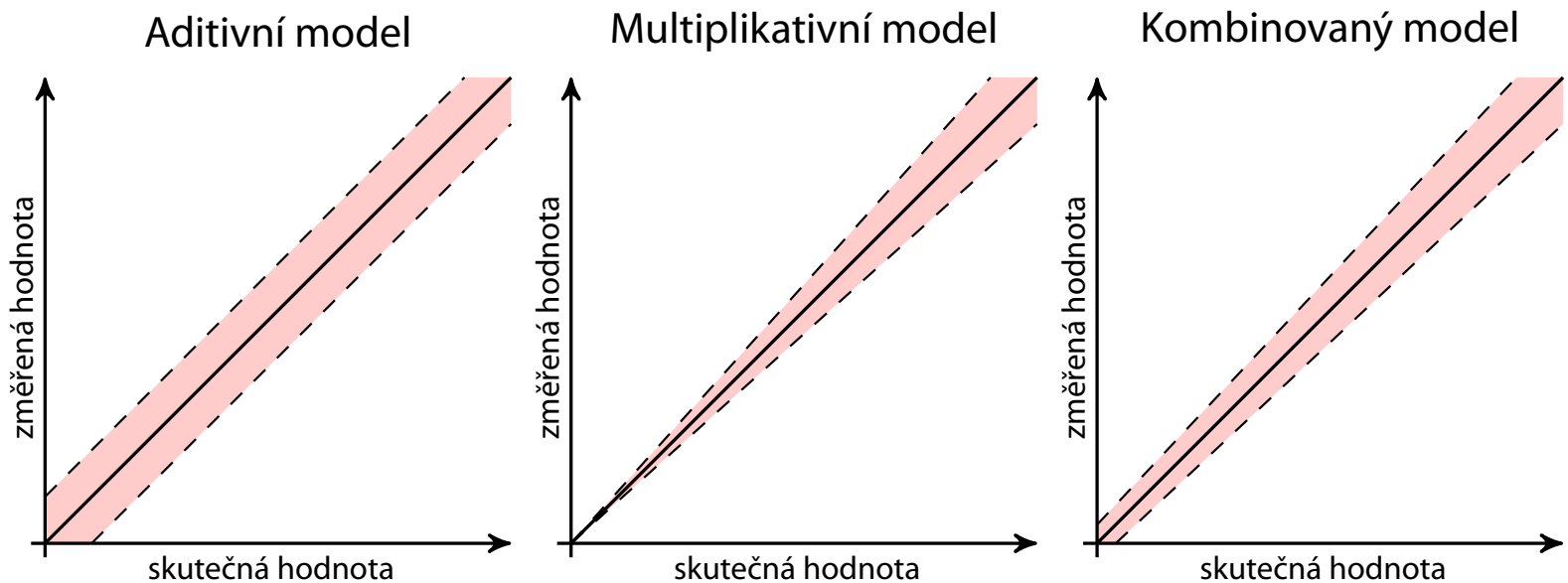


Přesnost měřících nástrojů

Přesnost přístroje … náhodná chyba

Správnost přístroje … systematická chyba

- Aditivní vs. multiplikativní chyby
- Mezní hodnota chyb
- Třída přesnosti přístroje



Náhodné chyby

aneb proč se běžně pracuje s normálním rozdělením chyb?

- Hypotéza elementárních chyb (Horák, 1958)
  - každá náhodná chyba v měření je složena z řady malých chyb
  - při velkém počtu měření se vyskytne zhruba stejný počet chyb kladných i záporných a malé chyby jsou početnější než velké

1. m elementárních náhodných vlivů
2. každý elementární vliv generuje chybu α (dále označováno jako případ a) nebo $-\alpha$ (dále případ b)
3. chyby a a b jsou stejně časté

  - dostáváme binomické rozdělení kumulace vlivů elementárních chyb

$$\binom{m}{0}a^m, \binom{m}{1}a^{m-1}b, \ldots, \binom{m}{l}a^{m-l}b^l, \ldots, \binom{m}{m}b^m$$

$$P(0) = \frac{1}{2^m}\binom{m}{m/2} \qquad P(\varepsilon_l) = \frac{1}{2^m}\binom{m}{l}, \varepsilon_l = (l-(m-l))\alpha = (2l-m)\alpha = 2s\alpha$$

Náhodné chyby

aneb proč se běžně pracuje s normálním rozdělením chyb?

- Co se stane, pokud $m \to \infty$?
  - pro sudá $m = 2k \implies k \to \infty$ (sudá, abychom měli $P(0)$)

$$P(\varepsilon) = P(2s\alpha) = \frac{1}{2^{2k}} \binom{2k}{k+s}$$

$$\frac{P(2s\alpha)}{P(0)} = \frac{\binom{2k}{k+s}}{\binom{2k}{k}} = \frac{k(k-1)\cdots(k-s+1)}{(k+1)(k+2)\cdots(k+s)} = \frac{\left(1-\frac{1}{k}\right)\left(1-\frac{2}{k}\right)\cdots\left(1-\frac{s-1}{k}\right)}{\left(1+\frac{1}{k}\right)\left(1+\frac{2}{k}\right)\cdots\left(1+\frac{s}{k}\right)}$$

  - pro $s \ll k$

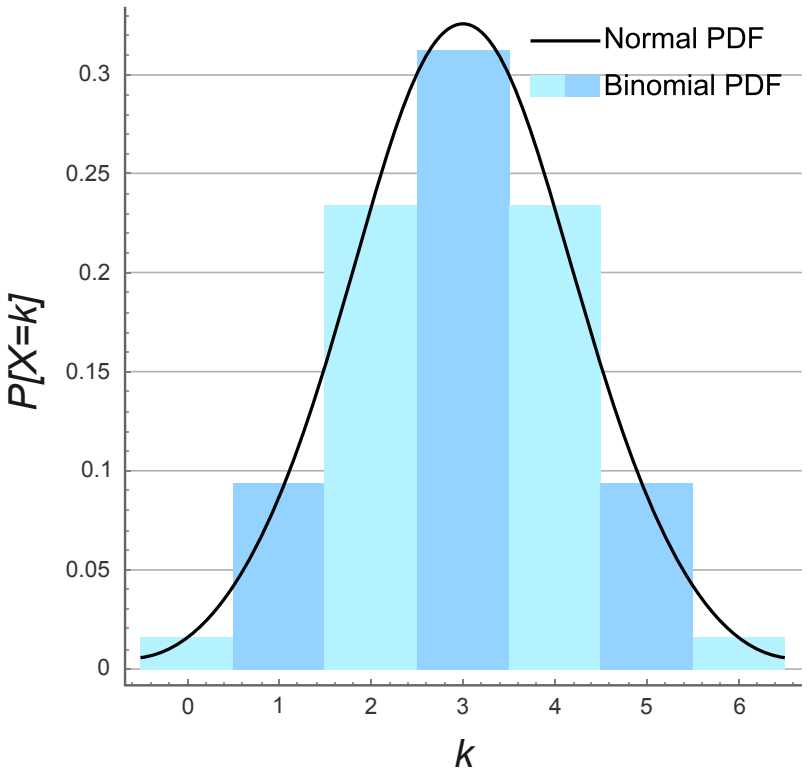
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots \approx x$$

$$\ln \frac{P(2s\alpha)}{P(0)} = -\frac{1}{k} - \frac{2}{k} - \cdots - \frac{s-1}{k} - \frac{1}{k} - \frac{2}{k} - \cdots - \frac{s}{k} = -\frac{2}{k}\frac{s(s-1)}{2} - \frac{s}{k} = -\frac{s^2}{k}$$

$$P(2s\alpha) = P(0)\mathrm{e}^{-\frac{s^2}{k}} = P(0)\mathrm{e}^{-\frac{\varepsilon^2}{4k\alpha^2}}$$

Binomické vs. normální rozdělení

Srovnání binomického a normálního rozdělení pro $p = 0{,}5$ a $n = 6$



Zdroj: **`http://en.wikipedia.org/wiki/File:Binomial_Distribution.svg`**

Studentovo rozdělení t

- Používá se pro normální rozdělení při malém vzorku

$$f(t) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{t^2}{\nu}\right)^{-(\nu+1)/2}$$

kde ν je počet stupňů volnosti.

  - odhad průměrů a chyby
  - t-test – odlišení průměrů

Odhad spolehlivosti

$$x = (\hat{\mu}_x \pm z_x) \text{ [jednotka]}$$

Statistická definice (Šťastný, 1997): Je-li výsledek měření $\hat{\mu}_x$ a z_x je chyba tohoto měření odpovídající míře jistoty p, pak skutečná hodnota měřené veličiny leží v intervalu $(\hat{\mu}_x \pm z_x)$ s pravděpodobností p.

- Intervaly
  - 0,68 – střední kvadratická chyba
  - 0,95
  - 0,99 – krajní chyba
- Zaokrouhlování
  - z_x nejvýše na 2 platná místa
  - $\hat{\mu}_x$ podle z_x

Odhad spolehlivosti

$$x = (\hat{\mu}_x \pm z_x) \text{ [jednotka]}$$

Pro normální rozdělení chyby

- $\hat{\mu}_x = \overline{x} = \frac{\sum_{i=1}^{N} x_i}{n}$
- s směrodatná odchylka jednoho měření, D rozptyl

$$s = \sqrt{D} = \sqrt{\frac{\sum_{i=1}^{N} (\overline{x} - x_i)^2}{n-1}}$$

- $s_{\overline{x}} = \sqrt{\sum_{i=1}^{N} (\frac{1}{n})^2 s_{x_i}}$ a protože měření byly prováděny za stejných podmínek

$$s_{\overline{x}} = \frac{s_x}{\sqrt{n}} = \sqrt{\frac{\sum_{i=1}^{N} (\overline{x} - x_i)^2}{n(n-1)}}$$

Odhad spolehlivosti

$$x = (\hat{\mu}_x \pm z_x) \text{ [jednotka]}$$

Pro normální rozdělení chyby

- $z_x = t_{(p;n-1)} s_{\overline{x}}$

n \ P	0,683	0,954	0,99	n \ P	0,683	0,954	0,99
1	1,8395	13,8155	63,6567	16	1,0329	2,1633	2,9208
2	1,3224	4,5001	9,9248	18	1,0292	2,1433	2,8784
3	1,1978	3,2923	5,8409	20	1,0263	2,1276	2,8453
4	1,1425	2,8585	4,6041	30	1,0176	2,0817	2,75
5	1,1113	2,6396	4,0321	40	1,0133	2,0595	2,7045
6	1,0913	2,5084	3,7074	50	1,0108	2,0463	2,6778
7	1,0775	2,4214	3,4995	60	1,0091	2,0377	2,6603
8	1,0673	2,3594	3,3554	70	1,0078	2,0315	2,6479
9	1,0594	2,3131	3,2498	80	1,0069	2,0269	2,6387
10	1,0533	2,2773	3,1693	90	1,0062	2,0234	2,6316
12	1,0441	2,2253	3,0545	100	1,0057	2,0206	2,6259
14	1,0377	2,1895	2,9768				

Odhad spolehlivosti

$$x = (\hat{\mu}_x \pm z_x)\ [\text{jednotka}]$$

Příklad – měření výšky válečku (Šťastný, 1997):

výška v [mm]	4,6	4,5	4,7	4,4	4,5	4,6	4,4	4,4	4,3	4,5

- $n = 10$
- $\overline{v} = 4{,}49\,[\text{mm}]$
- $s_{\overline{v}} = 0{,}038\,[\text{mm}]$
- $t_{(0{,}68;9)} = 1{,}059$
- $t_{(0{,}99;9)} = 3{,}250$

$$v = (4{,}49 \pm 0{,}04)\ \text{mm} \quad \text{pro } p = 0{,}68$$
$$v = (4{,}49 \pm 0{,}12)\ \text{mm} \quad \text{pro } p = 0{,}99$$

Zákon přenosu chyb

- Na základě Taylorova rozvoje do druhého členu

$$s_{\overline{z}}^2 = \sum_{i=1}^{N} \left(\frac{\partial z}{\partial x_i} \right)^2 s_{\overline{x_i}}^2 + 2 \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{\partial z}{\partial x_i} \frac{\partial z}{\partial x_j} s_{\overline{x_i}} s_{\overline{x_j}} \varrho_{ij} \ ,$$

kde $s_{\overline{x_i}}^2$ je rozptyl (variance) x_i a ϱ_{ij} je kovariance x_i a x_j.

Pro jednoduché případy, kdy x a y jsou nezávislé ($\varrho_{ij} = 0$):

- aditivní funkce $z = ax \pm by$

$$s_{\overline{z}} = \sqrt{a^2 s_{\overline{x}}^2 + b^2 s_{\overline{y}}^2} \ , \tag{1}$$

- multiplikativní funkce $z = ax^b y^c$

$$s_{\overline{z}} = \overline{z} \sqrt{\left(\frac{b s_{\overline{x}}}{\overline{x}} \right)^2 + \left(\frac{c s_{\overline{y}}}{\overline{y}} \right)^2} \ . \tag{2}$$

kde $\overline{z} = a\overline{x}^b \overline{y}^c$, protože

$$\sum_{i=1}^{N} \left(\frac{\partial z}{\partial x_i} \right)^2 s_{\overline{i}}^2 = \left(\frac{abx^b y^c s_{\overline{x}}}{x} \right)^2 + \left(\frac{ax^b c y^c s_{\overline{y}}}{y} \right)^2 = z^2 \left(\left(\frac{b s_{\overline{x}}}{x} \right)^2 + \left(\frac{c s_{\overline{y}}}{y} \right)^2 \right)$$

- Příklad použití: **`http://www.phy.ohiou.edu/~murphy/courses/sample.pdf`**

Skripta Fr. Šťastného (Sťastný, 1997)
http://amper.ped.muni.cz/jenik/nejistoty/