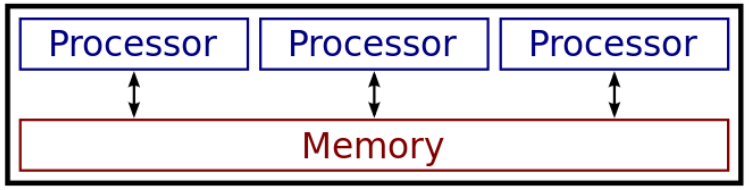

Semesterprojekt
Verteilte Echtzeitrecherche in Genomdaten

# An Introduction to Concurrent Programming in Java

Marc Bux (buxmarcn@informatik.hu-berlin.de)

# Concurrent Computing

- Parallel computing:
  Information exchange
  via shared memory
- Distributed computing:
  Information exchange
  via passing messages
- Typical Problems:
  - Conflicts & deadlocks
  - Node failures
  - Distribution of data & workload
- Architecture: centralized versus de-centralized



# What this talk is (not) about

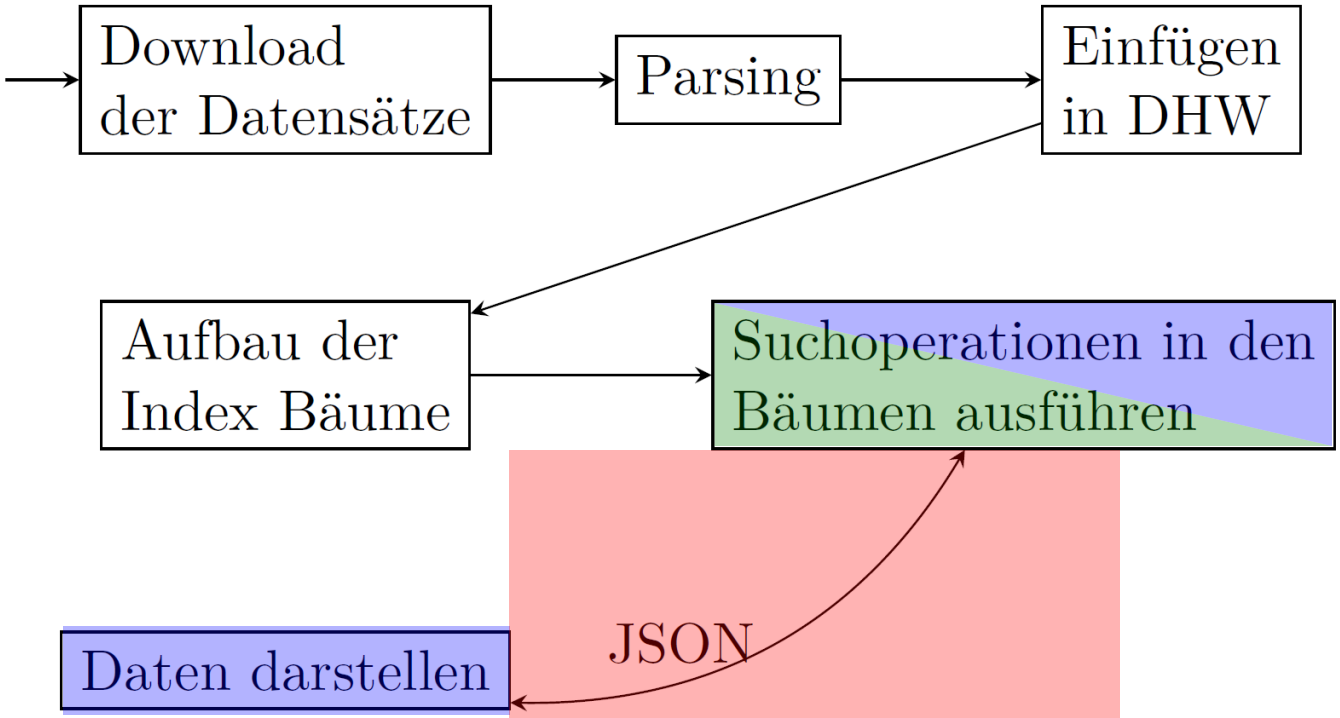
- What this talk is about:
  - Parallel computing: Threads, Locks
  - Distributed computing: Sockets, MPI
  - Data exchange formats: JSON, (XML, YAML)
  - Implementations in Java to get started with
- What this talk is not about:
  - Distributed search indices
  - Theoretical foundations
  - Technical implementations

# Where can you apply these concepts?

Parallel Computing

Distributed Computing

Data Exchange Formats

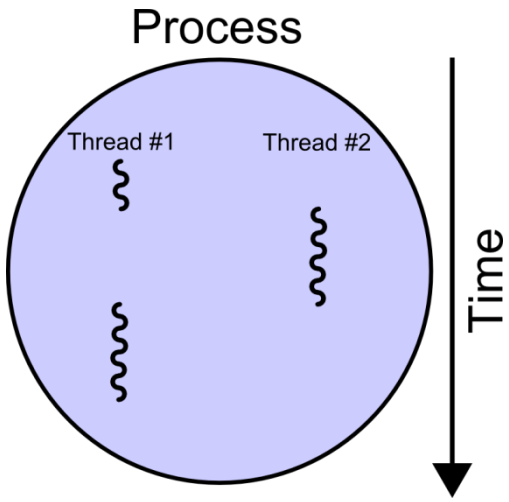


# Agenda

1. Parallel Computing
   - Threads
   - Locks
2. Distributed Computing
3. Date Exchange Formats

# Threads and Processes

- Process:
  - Instance of a program in execution
  - Separate entity with own heap space
  - Cannot access another process‘s data structures




"Multithreaded process" by I, Cburnett. Licensed under CC BY-SA 3.0 via Commons - https://commons.wikimedia.org/wiki/File:Multithreaded_process.svg#/media/File:Multithreaded_process.svg

- Thread:
  - Component of a process
  - Shares the process‘s resources
  - Has its own stack, but shares heap memory (data structures) with other threads

# Threads in Java

- In Java, threads can be implemented in two ways:
  1. Implement `java.lang.Runnable` interface
  2. Extend `java.lang.Thread` class
- The former is usually preferred to the latter
  - A class implementing the `Runnable` interface may extend another class
  - The `Thread` class brings some overhead with it

# Implementing java.lang.Runnable

```
public class RunnableCount implements Runnable {
        public int count = 0;
        public void run() {
                try {
                        while (count < 10) {
                                Thread.sleep(250);
                                System.out.println("count: " + count);
                                count++;
                        }
                } catch (InterruptedException e) {
                        System.out.println("RunnableCount interrupted.");
                }
        }
}

public static void main(String[] args) {
        RunnableCount runnableCount = new RunnableCount();
        Thread threadCount = new Thread(runnableCount);
        threadCount.start();
        while (runnableCount.count != 10) {
                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}
```

# Extending java.lang.Thread

```
public class ThreadCount extends Thread {
        public int count = 0;
        public void run() {
                try {
                        while (count < 10) {
                                Thread.sleep(250);
                                System.out.println("count: " + count);
                                count++;
                        }
                } catch (InterruptedException e) {
                        System.out.println("ThreadCount interrupted.");
                }
        }
}

public static void main(String[] args) {
        ThreadCount threadCount = new ThreadCount();
        threadCount.start();

        while (threadCount.count != 10) {
                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}
```

# Synchronization and Locks in Java

- Threads can attempt to modify shared resources at the same time
- Locks can be used to limit access to shared resources
- In Java, there are three common ways to implement locks:
  1. Implement a lock on an object by using the synchronized keyword for a method
  2. Implement a lock on an object by using the synchronized keyword for a block of code
  3. Manually implement a lock using the `java.util.concurrent.locks.Lock` interface

# Synchronized Methods in Java

```
public class SynchronizedCounter {
        private int c = 0;

        public synchronized void increment() {
                c++;
        }
}
```

# Synchronized Blocks in Java

```
public class SynchronizedCounter {
        private int c = 0;

        public void increment() {
                synchronized(this) {
                        c++;
                }
        }
}
```

# Synchronized Blocks in Java (cont.)

```
public class SynchronizedDoubleCounter {
        private int c1 = 0;
        private int c2 = 0;
        private Object lock1 = new Object();
        private Object lock2 = new Object();

        public void incrementC1() {
                synchronized(lock1) {
                        c1++;
                }
        }

        public void incrementC2() {
                synchronized(lock2) {
                        c2++;
                }
        }
}
```

# Manual use of locks in Java

```
public class SynchronizedCounter {
        private Lock lock;
        private int c = 0;

        public SynchronizedCounter() {
                lock = new ReentrantLock();
        }

        public void increment() {
                lock.lock();
                c++;
                lock.unlock();
        }
}
```

# Deadlocks

- Deadlock: state, in which two (or more) threads are waiting for one another
- Four conditions must be met:
  - Mutual exclusion: there is limited access / quantity to a resource
  - Hold and Wait: thread holding a resource A requests another resource B before releasing A
  - No Preemption: resources only released voluntarily
  - Circular Wait: multiple threads form a circular chain where each thread is waiting for another thread in the chain
- Livelock: risk in deadlock detection

# Agenda

1. Parallel Computing
2. Distributed Computing
   - Sockets
   - MPI
   - Large-Scale Distributed Processing Frameworks
3. Date Exchange Formats

# Sockets in Java

- (Network) socket: endpoint (IP address + port) of an inter-process communication across a network
- Used for low-level network communication via TCP
- Two types of sockets in Java:
  - `java.net.Socket` implements (client) sockets
  - `java.net.ServerSocket` implements server sockets that listen for connecting sockets on a port
- Java Remote Method Invocation (RMI): higher-level API based on sockets for communication between Java applications
- Objects of classes implementing the `java.io.Serializable` interface can be serialized and sent via sockets (using `ObjectInputStream`)

# The Server Side of a Socket in Java

```
try (
        ServerSocket serverSocket =
          new ServerSocket(portNumber);
        Socket clientSocket = serverSocket.accept();
        PrintWriter out =
          new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
          new InputStreamReader(clientSocket.getInputStream()));
) {
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
                out.println(inputLine);
        }
} catch (IOException e) {
        System.out.println(e.getMessage());
}
```

# The Client Side of a Socket in Java

```
try (
        Socket echoSocket =
          new Socket(hostName, portNumber);
        PrintWriter out =
          new PrintWriter(echoSocket.getOutputStream(), true);
        BufferedReader in =
          new BufferedReader(new InputStreamReader(echoSocket.getInputStream()));
        BufferedReader stdIn =
          new BufferedReader(new InputStreamReader(System.in))
) {
        String userInput;
        while ((userInput = stdIn.readLine()) != null) {
                out.println(userInput);
                System.out.println("echo: " + in.readLine());
        }
} catch (IOException e) {
        e.printStackTrace();
        System.exit(-1);
}
```

# Message-Passing Interface (MPI)

- MPI: a standard for message passing libraries in parallel computing
- Performant, portable across platforms, flexible wrt. underlying technology
- Abstract, high-level

```
comm.send(data, 5, MPI.DOUBLE, 1, 1);
Status status =
  comm.recv(data, 5, MPI.DOUBLE, MPI.ANY_SOURCE, 1);
```

- Implementations of MPI available for Java:
  - MPJ Express (http://mpj-express.org/)
  - OpenMPI (http://www.open-mpi.de/faq/?category=java)

# Performance comparison

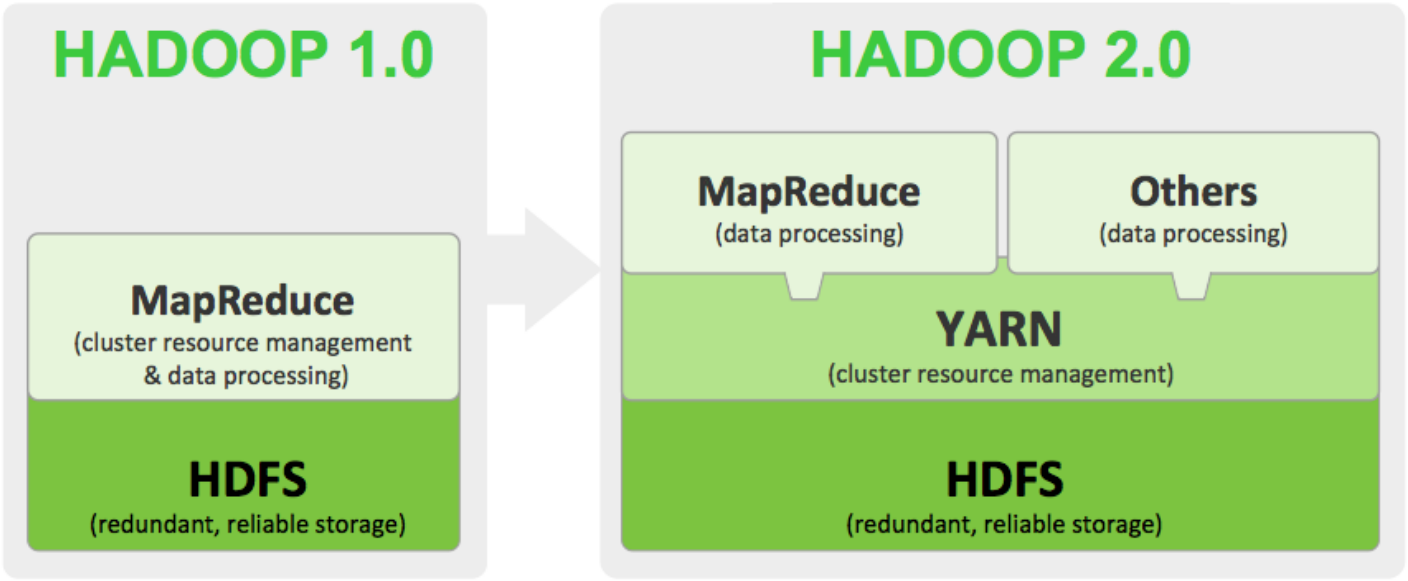
- Performance test: sort an integer array on a distributed infrastructure
  - 5 Intel pentium machines with 233 MHz
  - 100 Mbit network

| Array Size | Java Sockets | Java RMI | Java Sequential | MPI |
|---|---|---|---|---|
| 5000 | 636 | 5128 | 1841 | 119 |
| 10000 | 1694 | 10329 | 6864 | 333 |
| 15000 | 3343 | 18374 | 14648 | 602 |
| 20000 | 5525 | 29381 | 25183 | 998 |
| 25000 | 8252 | 42482 | 38336 | 1449 |
| 50000 | 30168 | 149866 | 144336 | 5072 |
| 100000 | 114602 | 554477 | 555069 | 18776 |

Qureshi, Kalim, and Haroon Rashid. "A performance evaluation of rpc, java rmi, mpi and pvm. "*Malaysian Journal of Computer Science* 18.2 (2005): 38-44.

# Large-Scale Distributed Processing Frameworks

- Apache Hadoop
  - comprises distributed filesystem HDFS and resource manager YARN




http://hortonworks.com/blog/apache-hadoop-2-is-ga/

- New cool kid in town: Apache Spark
  - Resilient Distributed Datasets (RDD)

# Agenda

1. Parallel Computing
2. Distributed Computing
3. Date Exchange Formats

# JSON

- Data Types: number, string, boolean, array, “object” (map), null

```
{
    "name": "Alex Rye",
    "deceased": false,
    "accounts": [
        {
            "bank": "Sparkasse",
            "balance": 3788
        },
        {
            "bank": "Commerzbank",
            "balance": 505
        }
    ],
    "gender": null
}
```

- Alternatives:
  - XML: more strict; separation of meta-data and data via tag attributes
  - YAML: less strict; superset of JSON with more features (comments, ordered maps, …)

# Questions

1. Parallel Computing
   - Threads
   - Locks
2. Distributed Computing
   - Sockets
   - MPI
   - Large-Scale Distributed Processing Frameworks
3. Date Exchange Formats