JKU

JOHANNES KEPLER
UNIVERSITY LINZ

JAVA PERFORMANCE

PR SW2 S18
Dr. Prähofer
DI Leopoldseder

OUTLINE

1. What is performance ?
   1. Benchmarking
2. What is Java performance ?
   1. Interpreter vs JIT
3. Tools to measure performance
4. Memory Performance
   1. GC Performance
5. Compiler Performance
   1. Optimization Patterns
6. Java Performance Rules

DISCLAIMER

- There are multiple **dedicated** courses covering Java performance and performance monitoring
  - ST: **`Java Performance Monitoring and Benchmarking`**, Lengauer (Summer Term)
    - What to benchmark
    - When to benchmark
    - How to benchmark
  - ST: **`Dynamic Compilation and Run-time Optimization in Virtual machines`**, Leopoldseder/Wirth (Summer Term)
    - Interpreters
    - Compilers
    - Dynamic Compilation
    - Optimizations in Dynamic Compilers

OUTLINE

1. **What is performance ?**
    1. Benchmarking
2. What is Java performance ?
    1. Interpreter vs JIT
3. Tools to measure performance
4. Memory Performance
    1. GC Performance
5. Compiler Performance
    1. Optimization Patterns
6. Java Performance Rules

TYPES OF PERFORMANCE

- Runtime performance
  - How fast does a program **finish**?
  - What is the **throughput** of an application ?
- Memory Performance
  - How large is the maximal memory footprint at runtime?
  - What is the average memory footprint?
  - Executable sizes?
- Cache Performance
  - DCache Utilization
  - ICache Utilization
- Network performance
  - Nr of open Sockets
  - Average Socket Open Time
- Database performance

TYPES OF PERFORMANCE

- Runtime performance
  - How fast does a program **finish**?
  - What is the **throughput** of an application ?
- Memory Performance
  - How large is the maximal memory footprint at runtime?
  - What is
  - Executa
- Cache Pe
  - DCach
  - ICache
- Network performance
  - Nr of open Sockets
  - Average Socket Open Time
- Database performance
- ….

Everything that can be **measured / observed** about a program or state of the system

WHAT IS PERFORMANCE USED FOR?

- Determine the performance of a system under test?
- Problem
  - Performance is **always relative** (to a system)
- Therefore we need a **reference** we can compare to
  - The reference can be
    - A different version of the same software on the same machine
    - A different machine
    - A different operating system
    - A different programming language
    - A different algorithm
    - ….

METRICS FOR PERFORMANCE

- Several metrics exist to measure various performance aspects of a system under test
  - Instructions Per Cycle
  - Instructions Per Second
  - Cycles Per Second
  - Floating Point Operations per Second
  - Data transmission rate
  - Latency
  - DB queries per second
  - Live heap size
  - Startup heap size
  - Number of live objects
  - Runtime (application level)
  - Throughput (application level)

GENERAL RULE

- Every performance number is **relative**
  - Based on a specific hardware configuration
  - Programs cannot be executed in complete isolation
  - OS can always interfere

Therefore **define** in which **aspects** of **performance** you are **interested** and **measure them….**

In a **reproducible experiment** running the system under test in **isolation** (as far as possible)

MYTHS

- There exists no oracle that makes statements like *“my computer is faster than yours valid”*
- Performance is always relative and depends on the hardware, software, external influences and the application that is used to measure performance

WHAT AFFECTS PERFORMANCE

- Hardware
  - CPU
    - ISA
    - Number of sockets, cores, threads, etc.
    - Number of registers
    - Instruction set extensions (e.g. AVX)
    - ICache & Dcache (L1,L2,L3)
    - Cache Type
  - Random Access Memory
  - Persistent Memory
    - HDD
    - SSD
- Software
  - Operating System
  - Program language
  - Compiler & Linker
  - Filesystem
- External Influences
  - Temperature
  - Air Pressure
  - Power Supply
  - Earth magnetism
  - Radiation

THE ART OF BENCHMARKING

- Creating & running reproducible workloads to measure different performance metrics of a system is called combined under the umbrella term **benchmarking**
- **A benchmark** is an application that can be executed (reproducibly) in an experiment to measure a defined metric of interest

- For every metric of interest research ot industry has developed benchmarks to measure performance

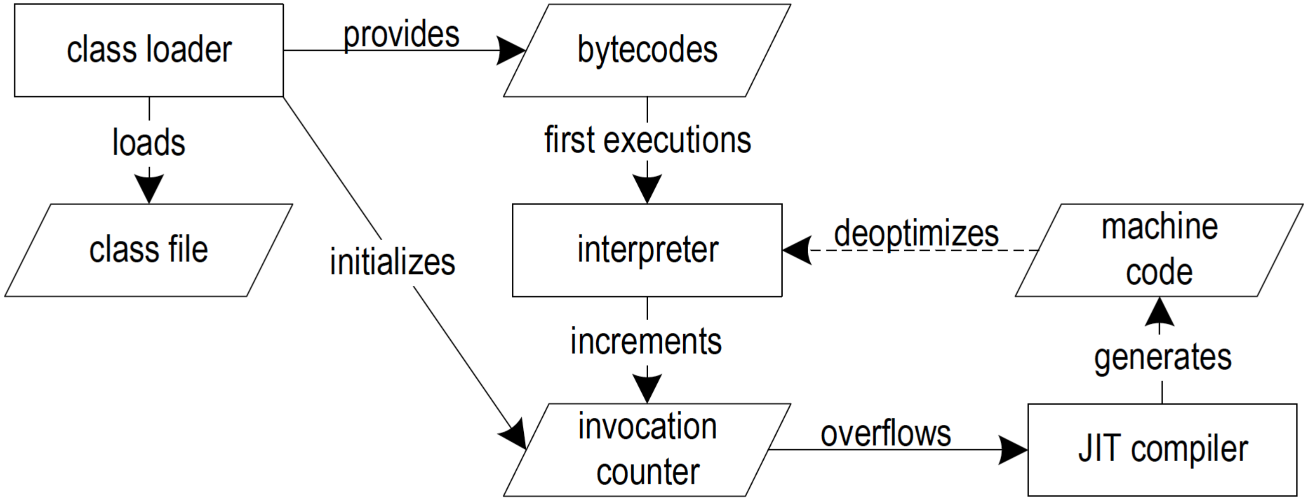
OUTLINE

1. What is performance ?
   1. Benchmarking
2. **What is Java performance ?**
   1. Interpreter vs JIT
3. Tools to measure performance
4. Memory Performance
   1. GC Performance
5. Compiler Performance
   1. Optimization Patterns
6. Java Performance Rules

JAVA PERFORMANCE PARADOXES

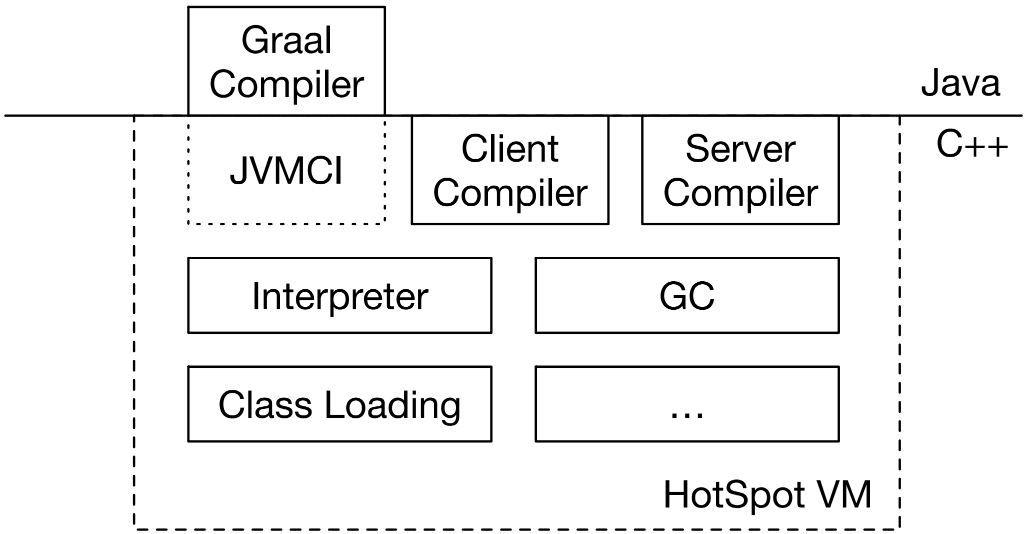
JAVA

- Traditionally Java is an interpreted language that uses just-in-time compilation at runtime to compile portions of important code to machine code



COMPONENTS OF A JVM (HOTSPOT)

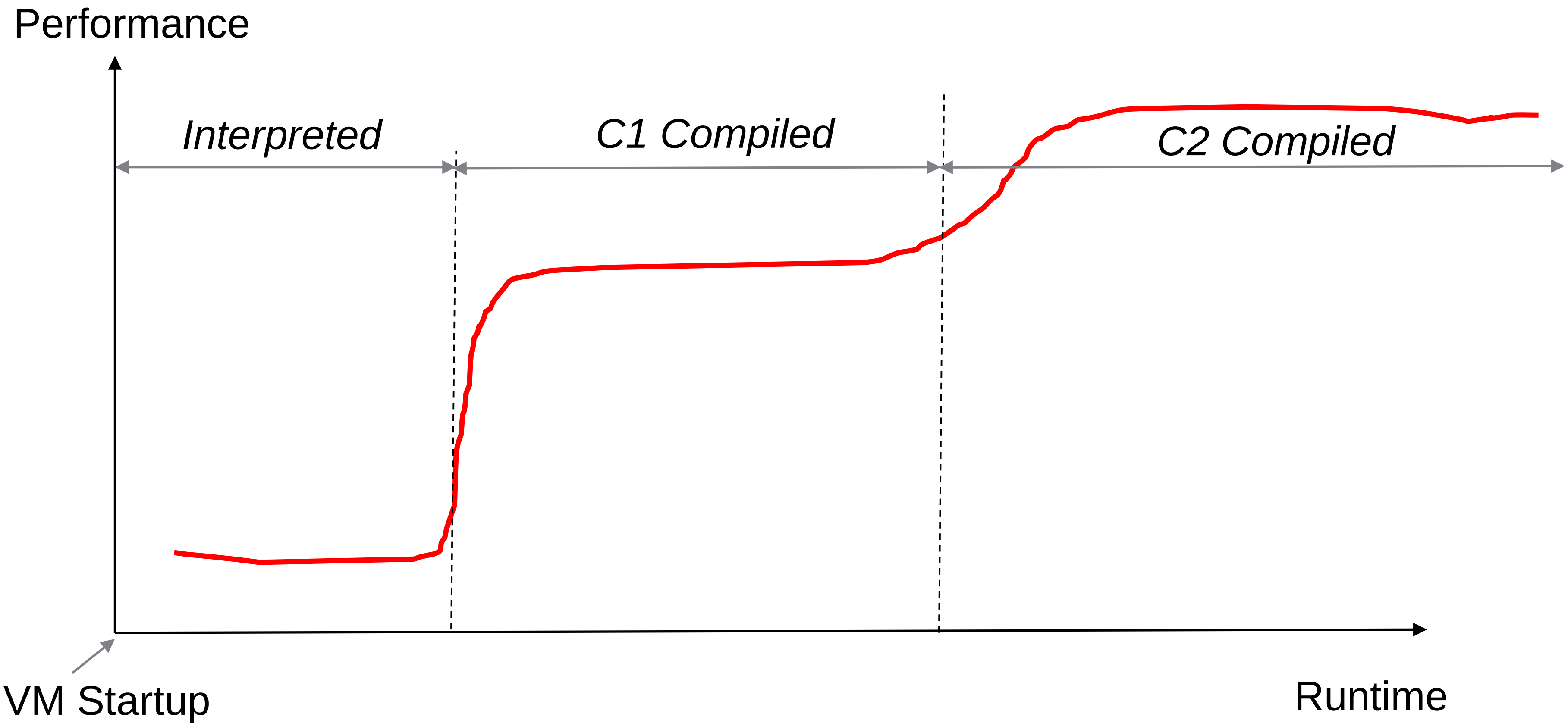
- HotSpot the VM in the OpenJDK is the defacto state of the art virtual machine for Java



OUTLINE

1. What is performance ?
   1. Benchmarking
2. What is Java performance ?
   1. **Interpreter vs JIT**
3. Tools to measure performance
4. Java Virtual Machine
   1. JIT Compiler & VM
5. Compiler Performance
   1. Optimization Patterns
6. Java Performance Rules

INTERPRETER AND COMPILER



HOTSPOTS

- Not all code is compiled at runtime to machine code
- Only important parts are compiled, those are called **hot spots**
  - Frequently executed methods
  - Frequently executed loops

OPTIMIZATIONS

- The just-in-time compilers of the VM are very smart, they try to optimize your code to make is as fast as possible by
  - Evaluating constant expressions at compile time
  - Inlining methods into callers
  - Removing object allocations if they are not needed (or only an objects fields are needed)
  - Remove unnecessary locking
  - Unrolling loops
  - Remove expressions from loops if they are not dependent on loop variables
  - Remove never executed code
  - ….

Do not try to be smarter than the VM, write clean code, the VM will figure out a way to optimize it properly…

OUTLINE

1. What is performance ?
   1. Benchmarking
2. What is Java performance ?
   1. Interpreter vs JIT
3. **Tools to measure performance**
4. Memory Performance
   1. GC Performance
5. Compiler Performance
   1. Optimization Patterns
6. Java Performance Rules

TOOLS FOR PERFORMANCE MEASUREMENT

- There exist a multitude of different tools to measure different performance metrics

Workflow for performance monitoring

1. *Define Metrics of interest*
2. *Define benchmark*
3. *Define benchmark workload (input, constant if possible)*
4. *Define measurement / monitoring tool*
5. *Measure performance (serval times, use statistics to make sense)*

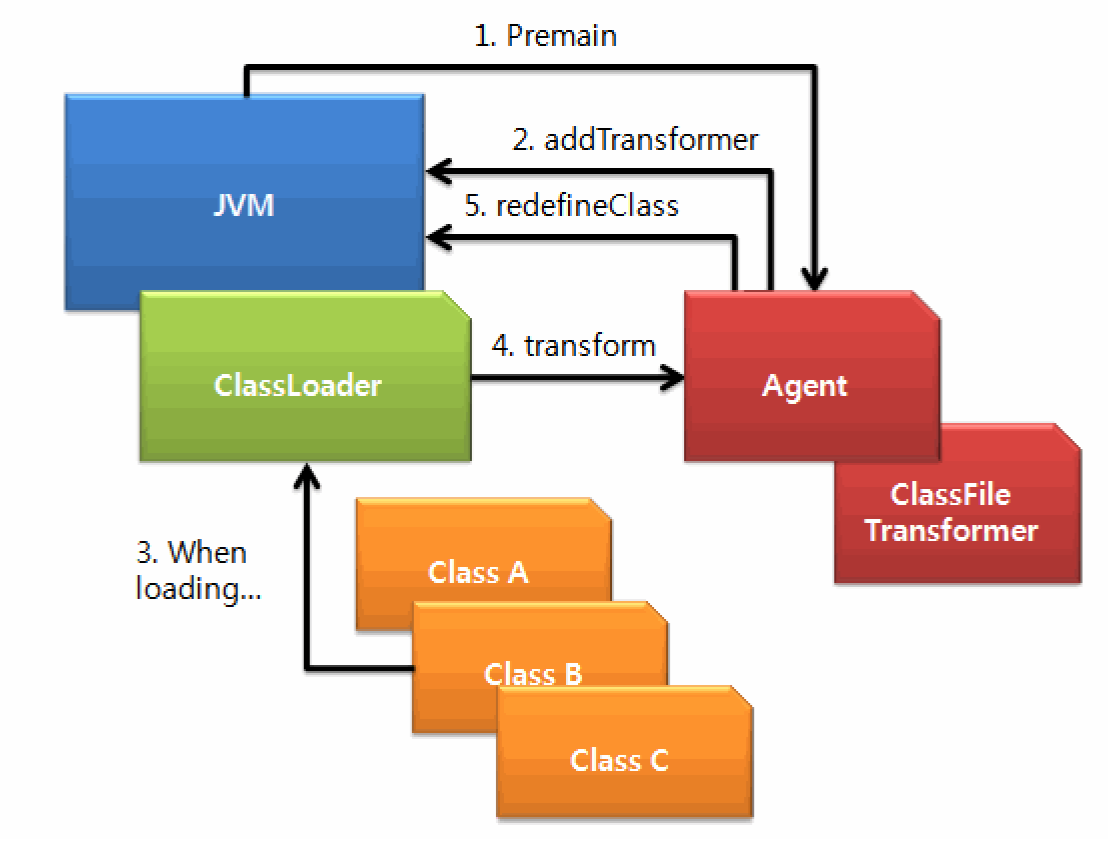
PROFILING….

- Is the task of analyzing a system under test by means of defined metrics
- We use a very simple characterization for different forms of profiling
  - Static Profiling: Offline analysis of the program
    - Source code complexity
    - Source level allocations
    - Source level loops
    - ….
  - Dynamic Profiling: Online analysis during execution of the program
    - Instrumentation based profiling
    - Sampling profiling
    - Event based profiling

INSTRUMENTATION BASED PROFILING

- Works by changing the application code itself to measure various metrics of interest
  - Number of method calls
  - Number of loop iterations
  - Number of different dynamic types at a callsite
  - Invocation Counts
  - Number of Allocations
  - ….
- Typically Very High Overhead
- For Java instrumentation takes typically place at bytecode level

BYTECODE INSTRUMENTATION



https://www.barcelonajug.org/2015/04/java-agents.html

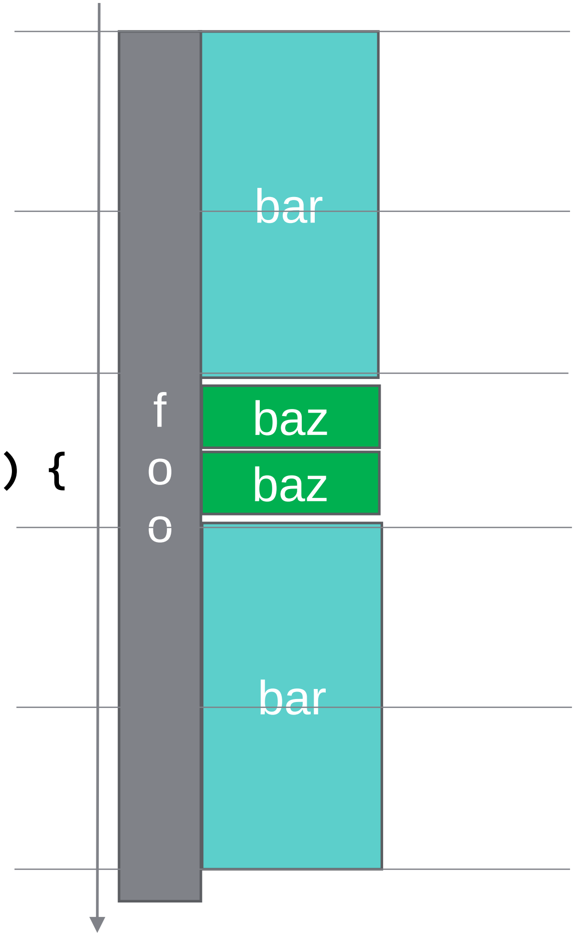
SAMPLING BASED PROFILING

- Periodically inspecting a programs execution stack

```
static void bar() {
  Thread.sleep(20ms);
}

static void baz() {
  Thread.sleep(5ms);
}

static void foo()  {
 for (int i = 0; i < 10000; i++) {
  if (i % 2 == 0) {
   bar();
  } else {
   baz();
   baz();
  }
 }
}
```



Sampling frequency
10 ms

Sampled Times
- Foo: 6 Samples
- Bar: 6 Samples
- Baz: 0 Samples

EVENT BASED PROFILING

- Requires runtime support
- Runtime creates special **events** during execution that are send to registered listeners at runtime
  - At Method calls
  - Object allocations
  - Thread creation
  - OS Calls
  - ….
- Most prominent event based profiler
  - JMVTI: Java virtual machine tool interface

OUTLINE

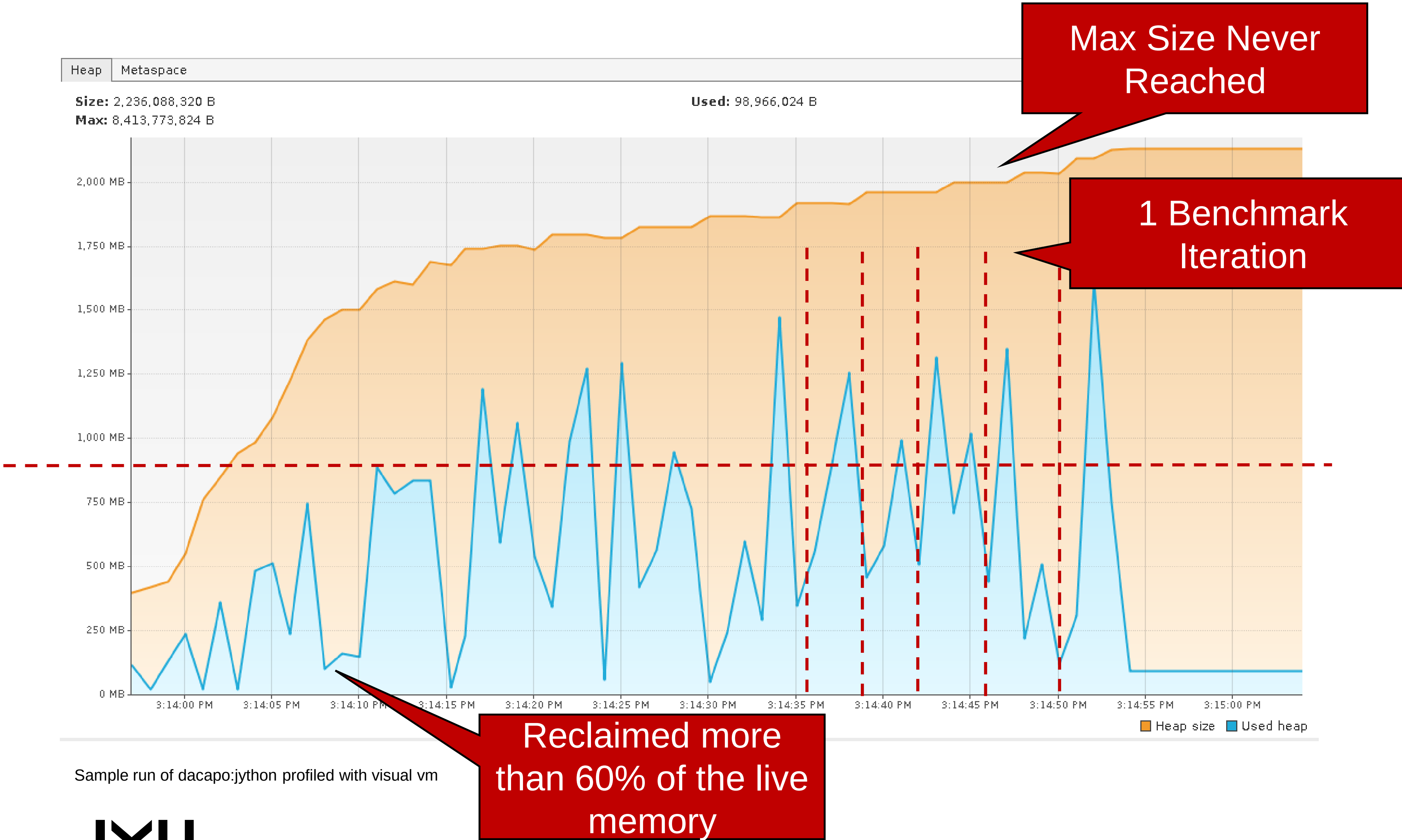
MEMORY PERFORMANCE

- Java uses automatic memory management to handle the reclaiming of unused memory
- Memory is reclaimed by a **garbage collector**
- **Generational hypothesis**
- GC is a program that
  - Finds all object references (mark live objects from GC roots)
  - Finds those still in use (referenced)
  - And collects the rest of them by reclaiming their space
  - [Optionally] Compacts the heap to fight fragmentation
- Several GC implementations exist
  - Serial
  - Parallel
  - CMS
  - G1
  - ZGC

GC IMPACT

- GC takes time and consumes CPU time
  - Can compete with application
- GCs implementation typically uses multiple GC threads
  - Collects objects concurrently
  - Distinction between GC thread and application thread (called mutator)
  - Sometimes GC needs to stop mutators to perform (parts of) the collection
    - Can take time
    - Can be a performance bottle neck

HEAP SIZE DURING APPLICATION



Sample run of dacapo:jython profiled with visual vm

JKU

OUTLINE

1. What is performance ?
    1. Benchmarking
2. What is Java performance ?
    1. Interpreter vs JIT
3. Tools to measure performance
4. Memory Performance
    1. GC Performance
5. **Compiler Performance**
    1. Optimization Patterns
6. Java Performance Rules

COMPILER OPTIMIZATION

- Optimizing compilers are essential to generate fast machine code
- However, application performance is bounded by problem space, application type, etc.
  - If 99% of the time an application is waiting for network requests optimizing 1% of the rest won’t do much
- What can be optimized by the compiler
  - CPU Bound problems
    - Optimize computations
    - Optimize Loops
    - …
  - Memory Bound problems
    - Remove allocation
    - Removes object locking
    - Reduce GC pressure

COMPILER OPTIMIZATIONS

- Dedicated course **Advanced Compiler Construction,** Mössenböck
- Complex transformations on a program
  - Constant Folding: `int a = 1 * 2;` → `int a = 2;`
  - Dead code elimination

```
static final boolean DEBUG = false;
void foo(){
 if(DEBUG) print("…info….");
 doWork();
}
```

→

```
void foo(){doWork();}
```

COMPILER OPTIMIZATIONS

- Inlining

```
void foo(){
 bar();
}
void bar(){
  …do work…
}
```

→

```
void foo(){
  …do work…
}
```

COMPILER OPTIMIZATIONS

- Loop Invariant Code Motion

```
void int foo(int[] arr,int x,int y){
  for(int i=0;i<arr.length;i++){
    arr[i] = arr[i]+(x*y);
  }
}
```

→

```
void int foo(int[] arr,int x,int y){
  int tmp = x*y;
  for(int i=0;i<arr.length;i++){
    arr[i] = arr[i]+tmp;
  }
}
```

ESCAPE ANALYSIS

```
long uselessAllocation(){
     return new Long(System.currentTimeMillis()).value;
}
                   →

long uselessAllocation(){
     return System.currentTimeMillis();
}
```

OUTLINE

PERFORMANCE RULES

- The compiler is always smarter than you are
- Write clean code, the compiler will figure out how to optimize it
- Never optimize something without measuring it first, always measure, find out what is slow and optimize then (*“Premature optimization is the root of all evil”*)
- Most objects die young (makes them cheap to be used)
- Always measure, never assume
- Non functional requirements (like performance) cannot be ignored until the end of a project, they are either important then we design for them or not important (*performance is always important*)

ASSIGNMENT 6

Performance Optimization with JNI and profiling

SORTING AN ARRAY OF MUTLI DIMENSIONAL ARRAYS

- Download the maven eclipse project from the course website
  - Maven clean
  - Maven install: Runs annotation processor and builds depencies
- 2 Tasks
  - Implement the given method with JNI
    - Measure and report which version is faster
  - Implement an optimized version for the task you may use any feature of the JVM you like but **NO** library function but you can
    - Use threads
    - Use JNI
    - Tweak the compiler
    - ….

JMH

- Tool to produce reproducible Java experiments by running tests in isolation multiple times and performing statistical analysis on it
- Based on byte code generation for annotated methods
- http://openjdk.java.net/projects/code-tools/jmh/
- See http://tutorials.jenkov.com/java-performance/jmh.html for more details

THANK YOU

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Strasse 69
4040 Linz, Austria
www.jku.at

JOHANNES KEPLER
UNIVERSITY LINZ