INTRO TO PERFORMANCE TUNING

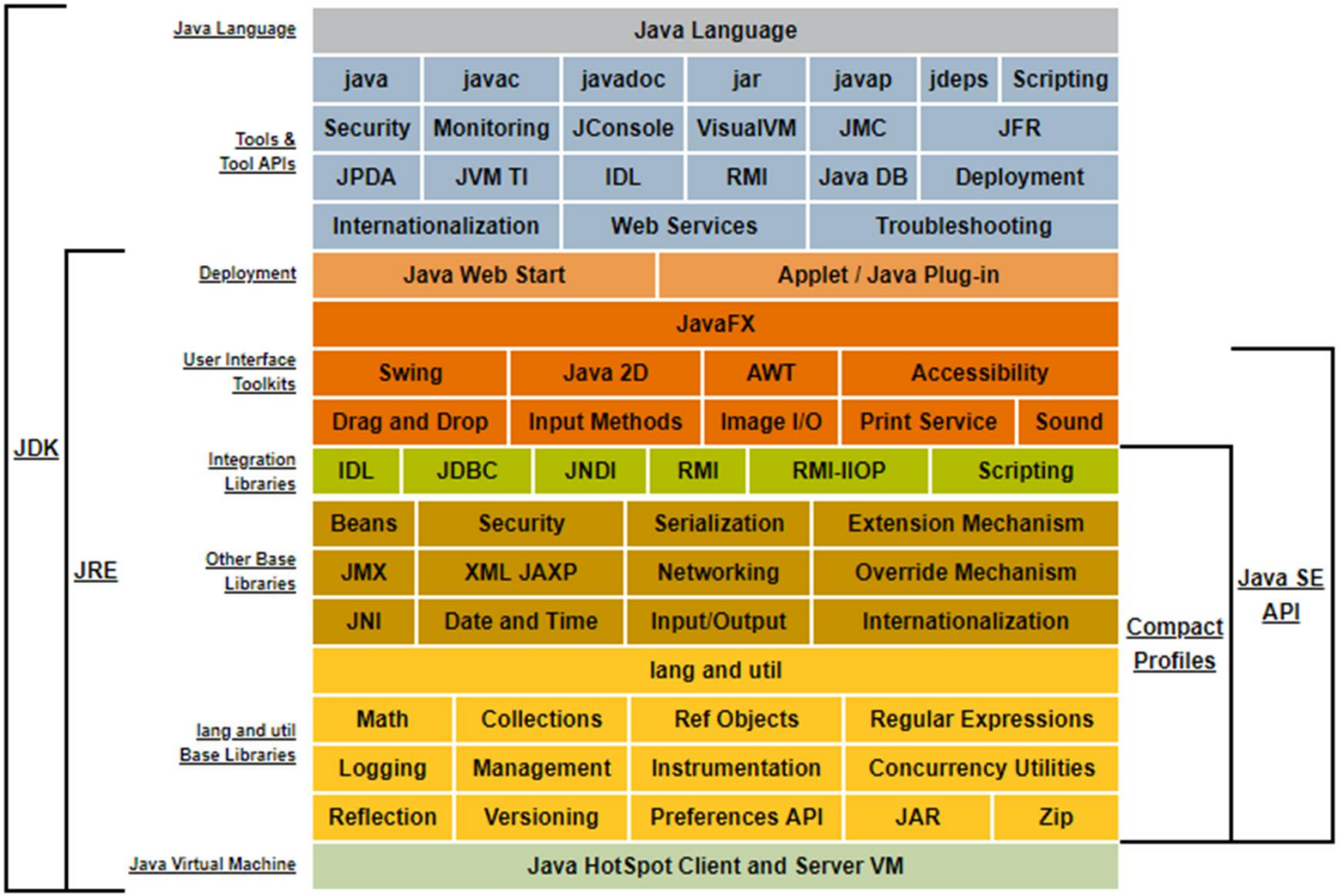
Performance Tuning

- Introduction to Performance Tuning
- Overview of JDK/JRE/JVM
- JVM Architecture and Internals
- Potential Performance Bottlenecks
- Detecting Performance Bottlenecks
- Monitoring and Profiling Tools
- Performance Tuning Techniques

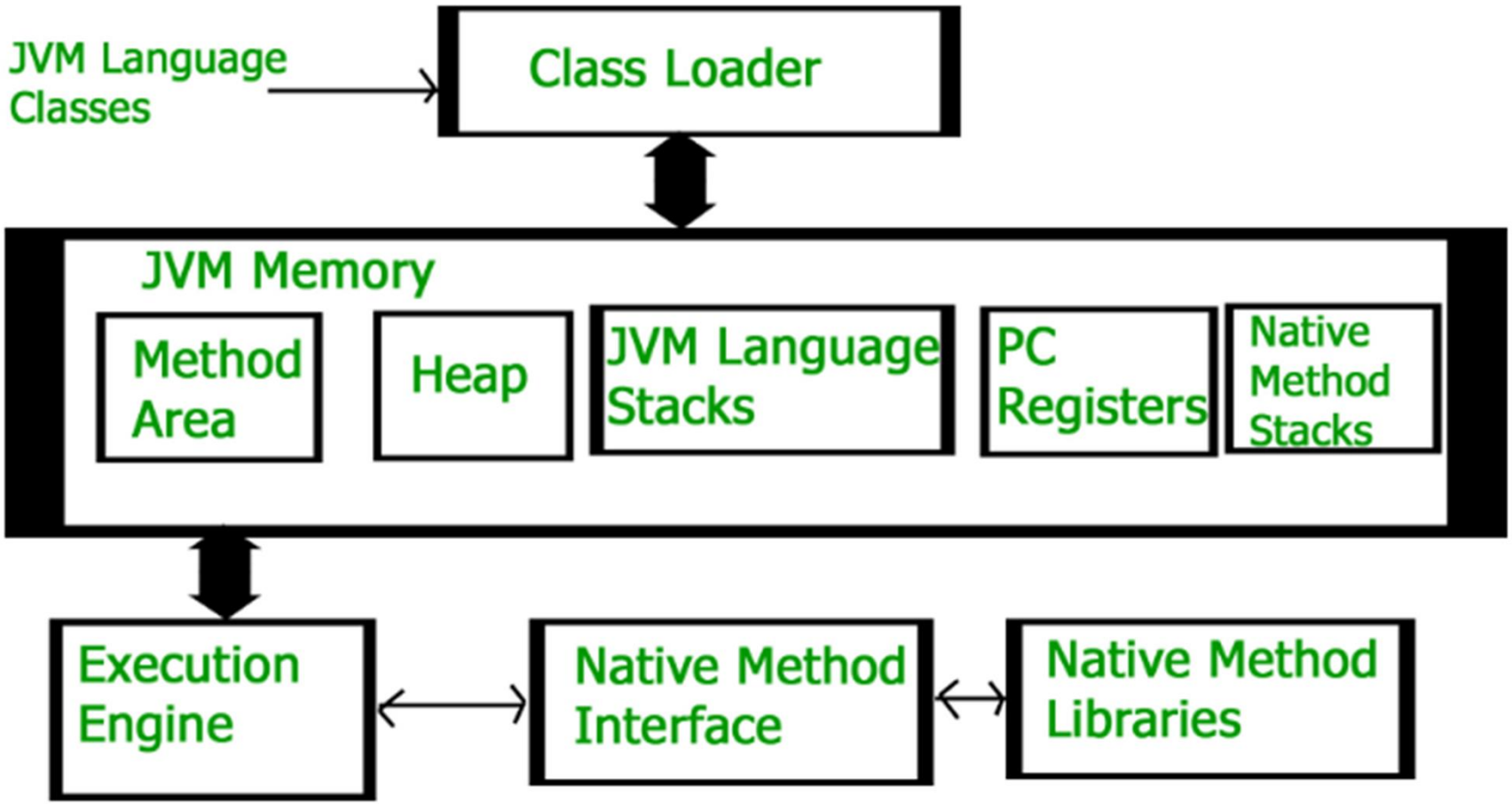
Optimal Performance

- the ability of a program to perform its computing tasks within the business response time requirements
- the ability of an application to fulfill its business functions under high volume
- in a timely manner, with high reliability and low latency

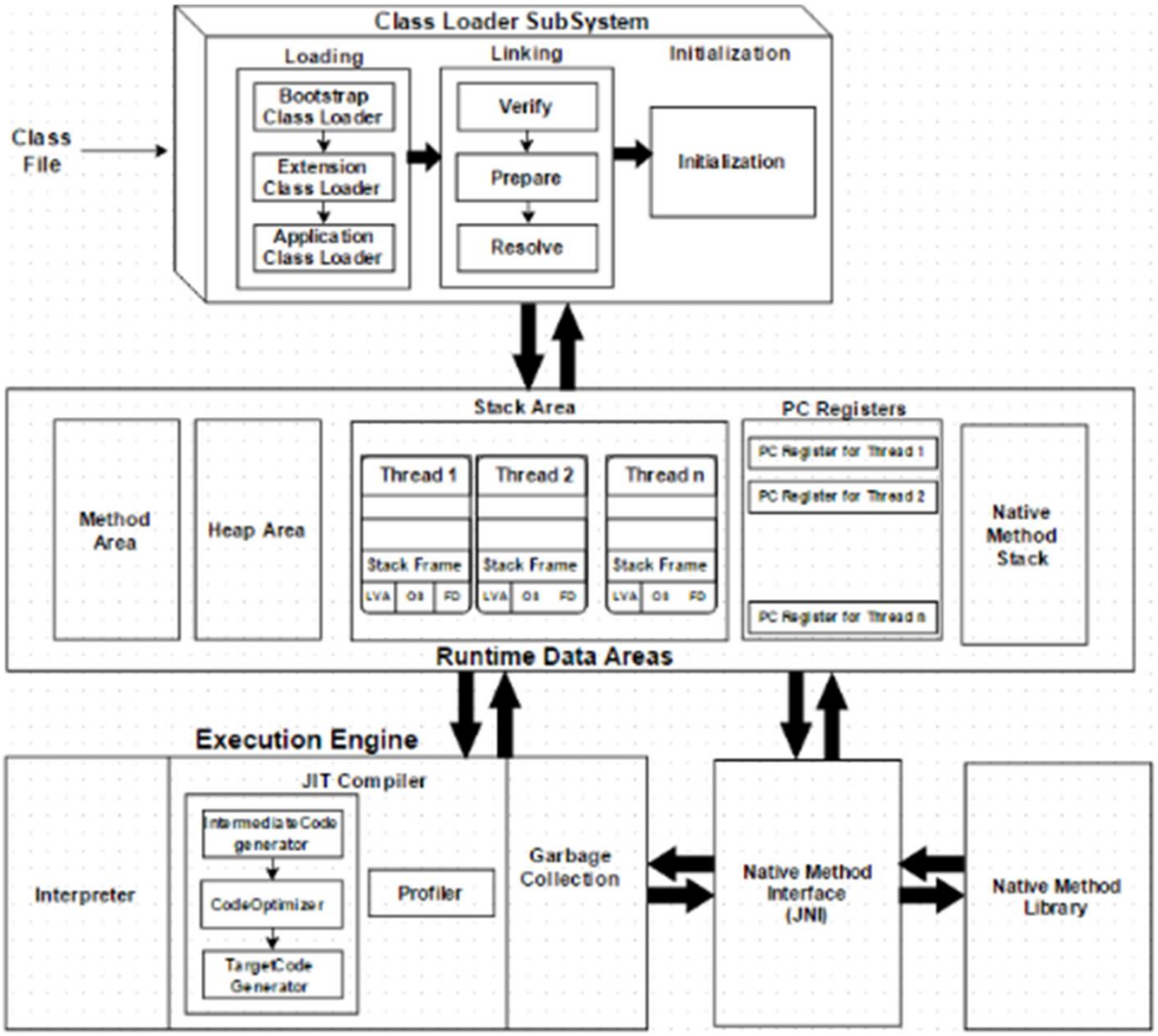
Java Conceptual Model (JVM/JRE/JDK)



JVM Architecture



JVM Architecture (detailed)



JVM Components – Class Loader Subsystem

- **Loading** - Classes will be loaded by this component
  - Boot Strap – Loads classes from the bootstrap classpath
  - Extension – Loads classes which are inside the ext folder
  - Application – Loads from Application Level Classpath, Environment Variable etc
- Linking
  - Verify - Bytecode verifier will verify whether the generated bytecode is proper or not
  - Prepare - For all static variables memory will be allocated and assigned with default values
  - Resolve - All symbolic memory references are replaced with the original references from Method Area
- Initialization
  - All static variables will be assigned with the original values, and the static block will be executed

JVM Components – Runtime Data Area

- Method Area - All the class level data will be stored here, including static variables
- Heap Area - All the Objects and their corresponding instance variables and arrays will be stored here
- Stack Area - For every thread, a separate runtime stack will be created.
  All local variables will be created in the stack memory.
- PC Registers - Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction
- Native Method Stacks - Native Method Stack holds native method information.
  For every thread, a separate native method stack will be created.

JVM Components – Execution Engine

- Interpreter
- JIT Compiler
  - Intermediate Code Generator
  - Code Optimizer
  - Target Code Generator
  - Profiler
- Garbage Collectors
- Java Native Interface
- Native Method Libraries

JVM Internals - Memory Management

- Memory Spaces
  - Heap - Primary storage of the Java program class instances and arrays
    - Young Generation [Eden Space, Survivor Space]
    - Old Generation
  - PermGen/Metaspace - Primary storage for the Java class metadata
  - Native Heap - native memory storage for the threads, stack, code cache including objects such as MMAP files and third party native libraries

JVM Internals – Garbage Collectors

- Serial Garbage Collector - Single threaded. Freezes all app threads during GC
- Parallel Garbage Collector - Multi threaded. Freezes all app threads during GC
- Concurrent Mark Sweep - Multi threaded with shorter GC pauses
- G1 Garbage Collector - Divides heap space into many regions and GCs region have more garbage

JVM Internals – Hotspot

- Region of a computer program where a high proportion of executed instructions occur or where most time is spent during the program's execution
- **Client VM** - Tuned for quick loading. It makes use of interpretation.
- **Server VM** - Loads more slowly, putting more effort into producing highly optimized JIT compilations to yield higher performance
- **Tiered Compilation** - uses both the client and server compilers in tandem to provide faster startup time than the server compiler, but similar or better peak performance

Potential Performance Bottlenecks

- Memory Leaks
- High CPU Utilization
- Thread Concurrency Issues
- Garbage Collection Overhead
- Network Latency/Timeouts

Detecting Memory Leaks

- System/ClassLoader Level
- Application Level
  - Analyze Heap Dumps
  - Perform Memory profiling to find out the exact cause

Heap Dump Analysis

- Different ways to capture
  - Actuator Endpoint
  - Jmap
  - OutOfMemoryError
  - jconsole -> MBean -> HotSpotDiagonostic
  - Jvisualvm

Heap Dump Analysis (contd.)

- Different ways to analyze
  - Jhat
  - Visualvm
  - Eclipse MAT (Memory Analyzer)
  - IBM Heap Dump Analyzer
  - Jprofiler
  - YourKit

Detecting reasons for High CPU Utilization

- Possible Reasons:
  - There are excessive GC cycles going on
  - Too many Application threads active
  - Code problems such as ‘infinite loops’ or excessive backend calls
  - The application is indeed working hard and the host does not have enough CPU (this is actually a good reason)

Detecting reasons for High CPU Utilization (contd.)

- Recommendations:
  - Get into the habit of periodically monitoring the CPU utilized by your application
  - Perform CPU profiling to find out the exact cause
  - Use good quality APM tool to monitor CPU regularly and generate alert
  - Load test the application to find out CPU utilization and optimize
  - Inspect the code issues and fix

Profiling

- Different ways to analyze
  - Jhat
  - Visualvm
  - Eclipse MAT (Memory Analyzer)
  - IBM Heap Dump Analyzer
  - Jprofiler
  - YourKit

Detecting Garbage Collection Overhead

- Capture and Log garbage collection details
- Analyze Garbage Collection logs
- GC Log Analyzer tools
  - Gceasy
  - GC Plot
  - GC Viewer

Detecting Concurrency Issues

- Possible Reasons:
  - Thread Contention
  - Deadlock

- Capture and analyze the Thread Dumps

Thread Dump Analysis

- Different ways to capture
  - jstack
  - jvisualvm

- Different ways to analyze
  - Jstack
  - Jvisualvm
  - IBM Thread Dump Analyzer
  - FastThread

Understanding Thread Dump

Thread State

- **NEW:** The thread is created but has not been processed yet.
- **RUNNABLE:** The thread is occupying the CPU and processing a task. (It may be in WAITING status due to the OS's resource distribution.)
- **BLOCKED:** The thread is waiting for a different thread to release its lock in order to get the monitor lock.
- **WAITING:** The thread is waiting by using a wait, join or park method.
- **TIMED_WAITING:** The thread is waiting by using a sleep, wait, join or park method. (The difference from WAITING is that the maximum waiting time is specified by the method parameter, and WAITING can be relieved by time as well as external changes.)

Understanding Thread Dump (contd.)

Thread Dump Patterns by Type

- **BLOCKED:** When Unable to Obtain a Lock (BLOCKED)
- **BLOCKED:** When in Deadlock Status
- **RUNNABLE**: When Continuously Waiting to Receive Messages from a Remote Server
- **WAITING:** When Waiting

How to Solve Problems by Using Thread Dump

- When the Processing Performance is Abnormally Slow (Memory / CPU utilization seems to be fine)
  - Acquire the list of threads with BLOCKED status after getting the thread dumps several times
  - Inadequate configurations
  - Abnormal connection
- When the CPU Usage is Abnormally High
  - Extract the thread that has the highest CPU usage
  - After acquiring the thread dump, check the thread's action

Performance Tuning Techniques

- JVM Tuning
  - Memory Tuning
  - GC Tuning
  - JIT Tuning
- Code Optimization
- Caching
- Load Balancing
- Distributed Computing
- Web/App Server and Middleware Tuning
- Operating System / Network Tuning

Java Monitoring and Management Tools

Tool	Description
jcmd	JVM Diagnostic Commands tool - Sends diagnostic command requests to a running Java Virtual Machine
jconsole	Graphical tool for monitoring a Java virtual machine
visualvm	Provides memory and CPU profiling, heap dump analysis, memory leak detection, access to MBeans, and garbage collection
jmc	Java Mission Control - includes tools to monitor and manage your Java application without introducing the performance overhead
jps	JVM Process Status Tool - Lists instrumented HotSpot Java virtual machines on a target system
jstat	JVM Statistics Monitoring Tool - Attaches to an instrumented HotSpot Java virtual machine and collects and logs performance statistics
jinfo	Prints configuration information for a given process
jmap	Prints shared object memory maps or heap memory details of a given process
jhat	Starts a web server on a heap dump file (for example, produced by jmap -dump), allowing the heap to be browsed
jstack	Prints a stack trace of threads for a given process

Thank You!