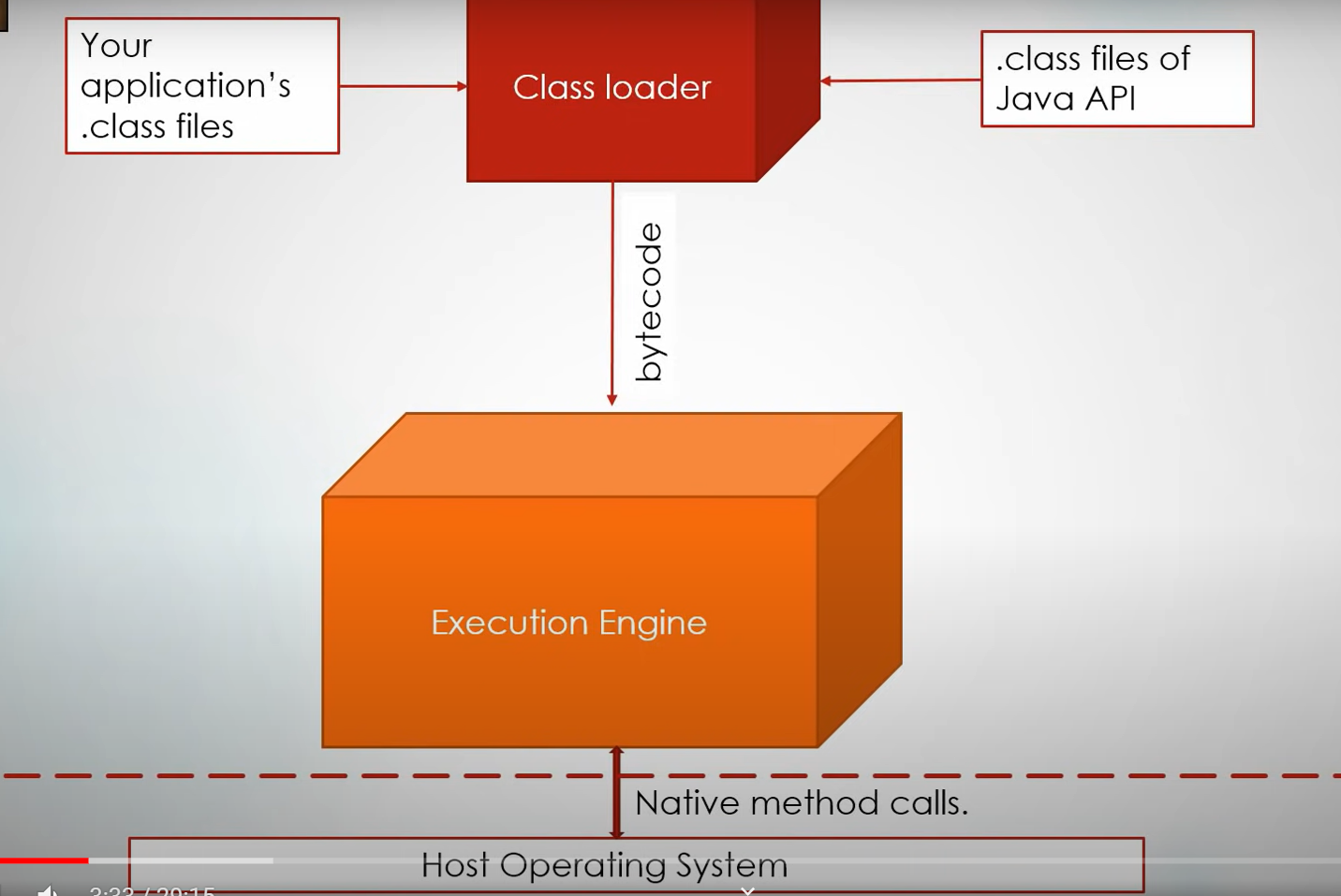
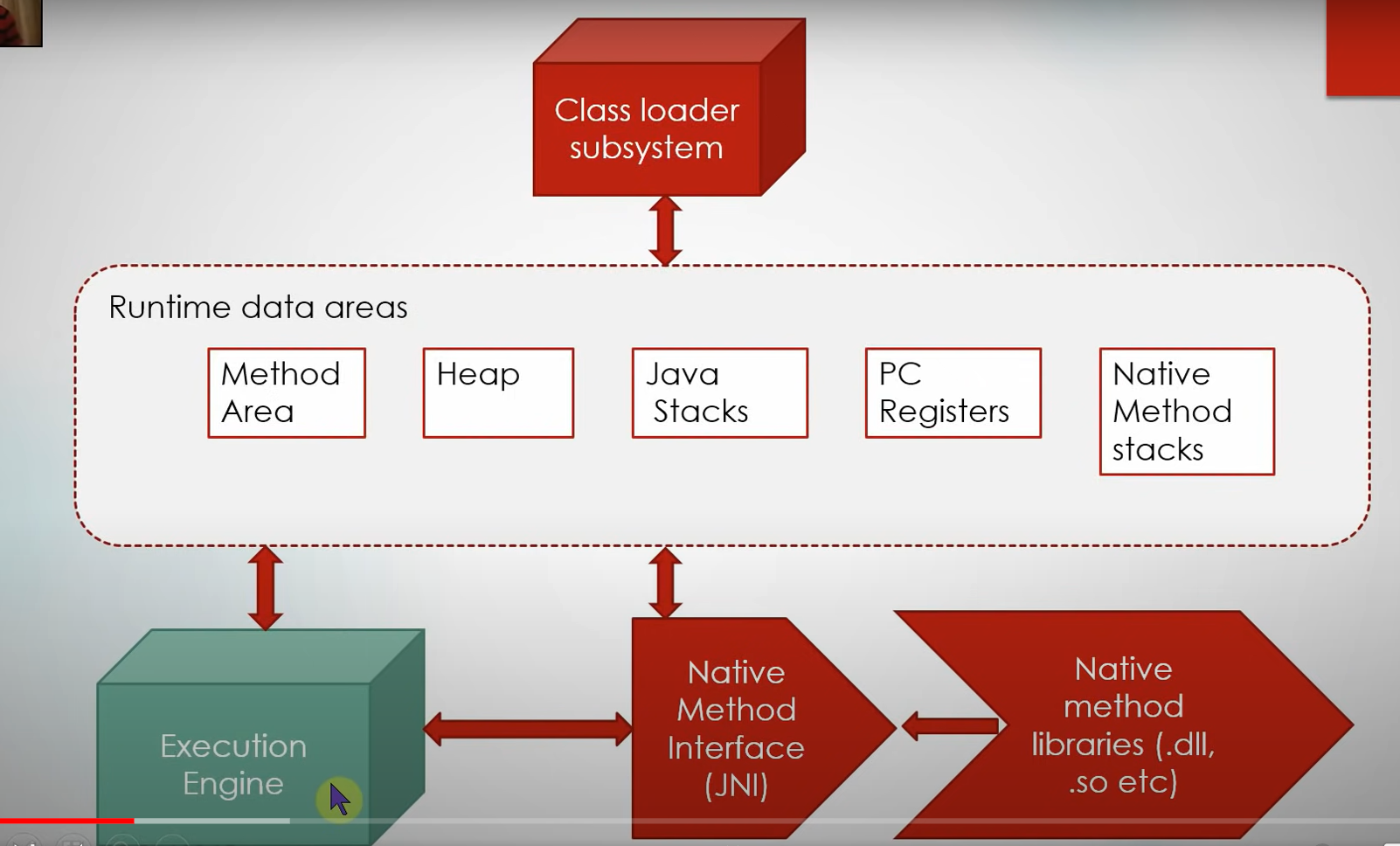
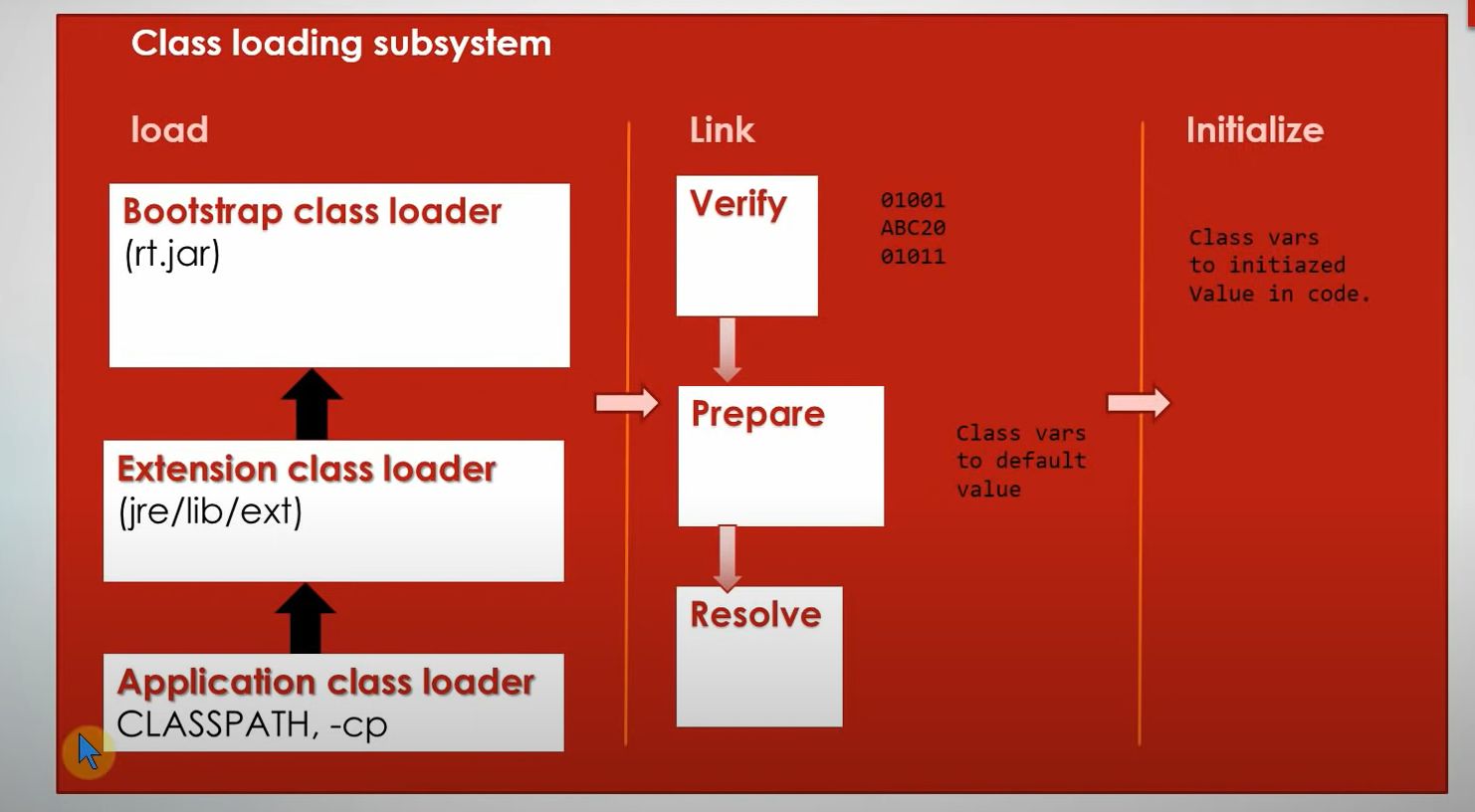
2 JVM architecture





2.1 class loading system



### **Phase Load**

### **1)Bootstrap Class Loader**

**Bootstrap class loader serves as a parent of all the other *ClassLoader* instances**.

**This bootstrap class loader is part of the core JVM and is written in native code**

### **2) Extension Class Loader**

**The extension class loader is a child of the bootstrap class loader and takes care of loading the extensions of the standard core Java classes so that it's available to all applications running on the platform**

**3) System Class Loader**

**It loads files found in the classpath environment variable, -classpath or -cp command line option.**

**Phase Link**

[**https://blog.csdn.net/DXH9701/article/details/107968981**](https://blog.csdn.net/DXH9701/article/details/107968981)

**1)Verify**

**Check if it is a valid Java byte code.**

**2)prepare**

**Memory is allocated for the static variables inside a classifier.**

**IMP: It is only memory allocation that is only for the class variables not for the instance variables. And this variables will be set to its default value like bool = false.**

**eg . private static integer.**

**3)resolve**

**Exception Class not found happens in this part.**

**The Java virtual machine replaces class symbolic references with direct references.**

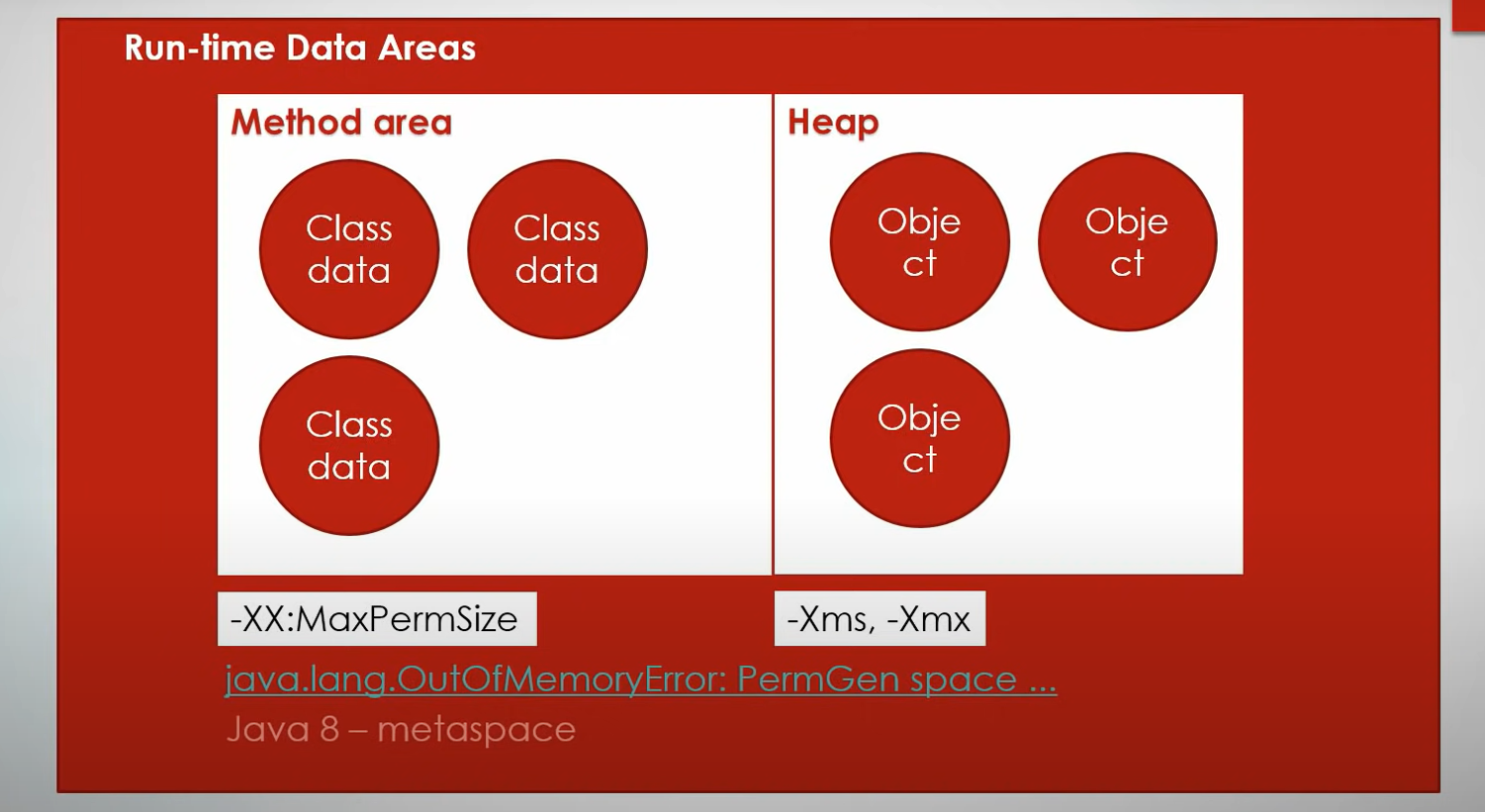
[**https://www.cnblogs.com/qlky/p/7643524.html**](https://www.cnblogs.com/qlky/p/7643524.html)

**Phase initialization**

**Initialize those static variables in class**

**初始化是为类的静态变量赋予正确的初始值，准备阶段和初始化阶段看似有点矛盾，其实是不矛盾的，如果类中有语句：private static int a = 10，它的执行过程是这样的，首先字节码文件被加载到内存后，先进行链接的验证这一步骤，验证通过后准备阶段，给a分配内存，因为变量a是static的，所以此时a等于int类型的默认初始值0，即a=0,然后到解析（后面在说），到初始化这一步骤时，才把a的真正的值10赋给a,此时a=10。**

**2.2 Runtime data areas**

****

### PC (Program Counter) Register

**It is local to each thread and contains the address of the JVM instruction that the thread is currently executing.**

### Stack

**It is local to each thread and stores parameters, local variables and return addresses during method calls. A StackOverflow error can occur if a thread demands more stack space than is permitted. If the stack is dynamically expandable, it can still throw OutOfMemoryError.**

### Heap

**It is shared among all the threads and contains objects, classes’ metadata, arrays, etc., that are created during run-time. It is created when the JVM starts and is destroyed when the JVM shuts down. You can control the amount of heap your JVM demands from the OS using certain flags (more on this later). Care has to be taken not to demand too less or too much of the memory, as it has important performance implications. Further, the GC manages this space and continually removes dead objects to free up the space.**

### Method Area

**This run-time area is common to all threads and is created when the JVM starts up. It stores per-class structures such as the constant pool (more on this later), the code for constructors and methods, method data, etc. The JLS does not specify if this area needs to be garbage collected, and hence, implementations of the JVM may choose to ignore GC. Further, this may or may not expand as per the application’s needs. The JLS does not mandate anything with regard to this.**

### Run-Time Constant Pool

**The JVM maintains a per-class/per-type data structure that acts as the symbol table (one of its many roles) while linking the loaded classes.**

### Native Method Stacks

**When a thread invokes a native method, it enters a new world in which the structures and security restrictions of the Java virtual machine no longer hamper its freedom. A native method can likely access the runtime data areas of the virtual machine (it depends upon the native method interface), but can also do anything else it wants.**

**2.3 execution engine**

**The execution engine is the Central Component of the** [**java virtual machine**](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)**(JVM). It communicates with various memory areas of the JVM. Each thread of a running application is a distinct instance of the virtual machine’s execution engine. Execution engine executes the byte code which is assigned to the run time data areas in JVM via class loader. Java Class files are executed by the execution engine.**

**Execution Engine contains three main components for executing Java Classes. They are:**

1. **Interpreter: It reads the byte code and** [**interprets**](https://www.geeksforgeeks.org/compiler-vs-interpreter-2/)**(convert) into the machine code(native code) and executes them in a sequential manner. This component runs the application from the command line by accepting a filename argument. The problem with the interpreter is that it interprets every time, even the same method multiple times, which reduces the performance of the system. To overcome this problem JIT Compilers is introduced in 1.1 version.**
2. [JIT Compiler](https://www.geeksforgeeks.org/just-in-time-compiler/)**: JIT compiler counterbalances the interpreter’s disadvantage of slow execution and improves the performance.**
   * **At run time, the JVM loads the class files, the semantic of each is determined and appropriate computations are performed. The additional processor and memory usage during interpretation makes a Java application perform slowly as compared to a native application.**
   * **The JIT compiler aids in improving the performance of Java programs by compiling bytecode into native machine code at run time.**
   * **The JIT compiler is enabled throughout, while it gets activated when a method is invoked. For a compiled method, the JVM directly calls the compiled code, instead of interpreting it. Theoretically speaking, If compiling did not require any processor time or memory usage, the speed of a native compiler and that of a Java compiler would have been same.**
   * **JIT compilation requires processor time and memory usage. When the java virtual machine first starts up, thousands of methods are invoked. Compiling all these methods can significantly affect startup time, even if the end result is a very good performance optimization.**
3. **Profiler: This is a tool which is the part of JIT Compiler is responsible to monitor the java bytecode constructs and operations at the JVM level.**
4. [Garbage Collector](https://www.geeksforgeeks.org/garbage-collection-java/)**: This is a program in java that manages the memory automatically. It is a** [**daemon thread**](https://www.geeksforgeeks.org/daemon-thread-java/) **which always runs in the background. This basically frees up the heap memory by destroying unreachable methods.**