Course Name:
Advanced Java

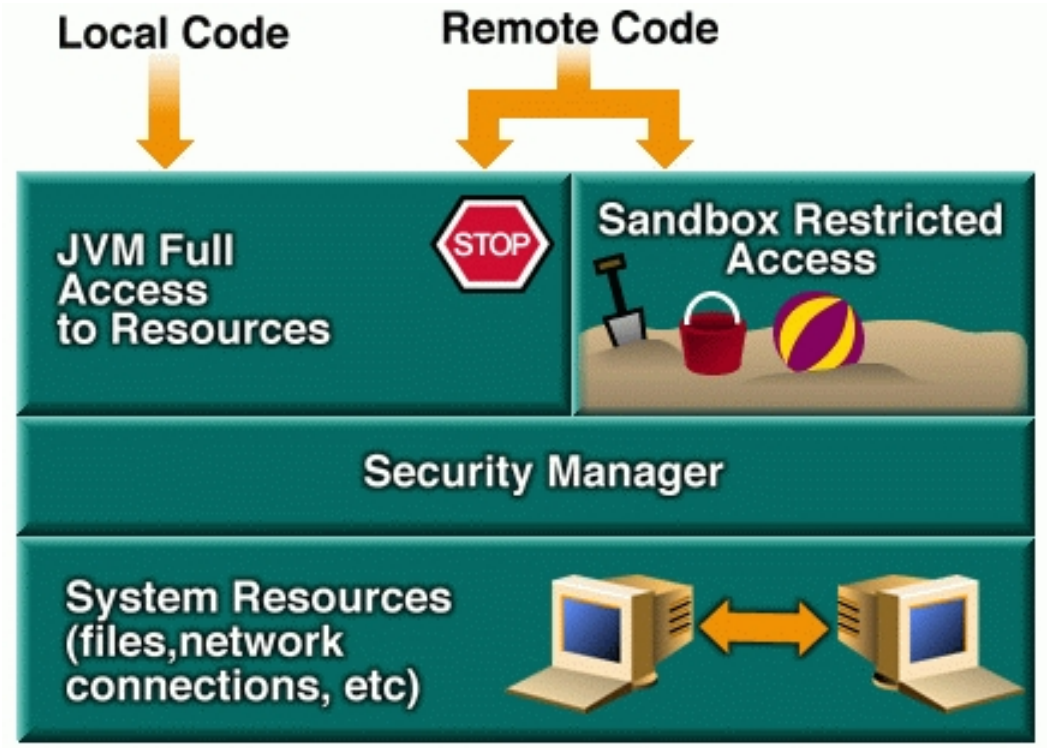
Lecture 25
Topics to be covered

- SECURITY
- Security Models
- Class Loaders
- Bytecode Verification
- Security Managers

JDK 1.0 Security Model

- Original security model, “sandbox”
- Very restricted environment
- All incoming code is considered untrusted
  - Access to limited resources inside the sandbox
- Local code is trusted
  - Full access to system resources
- Security manager determines the access limit

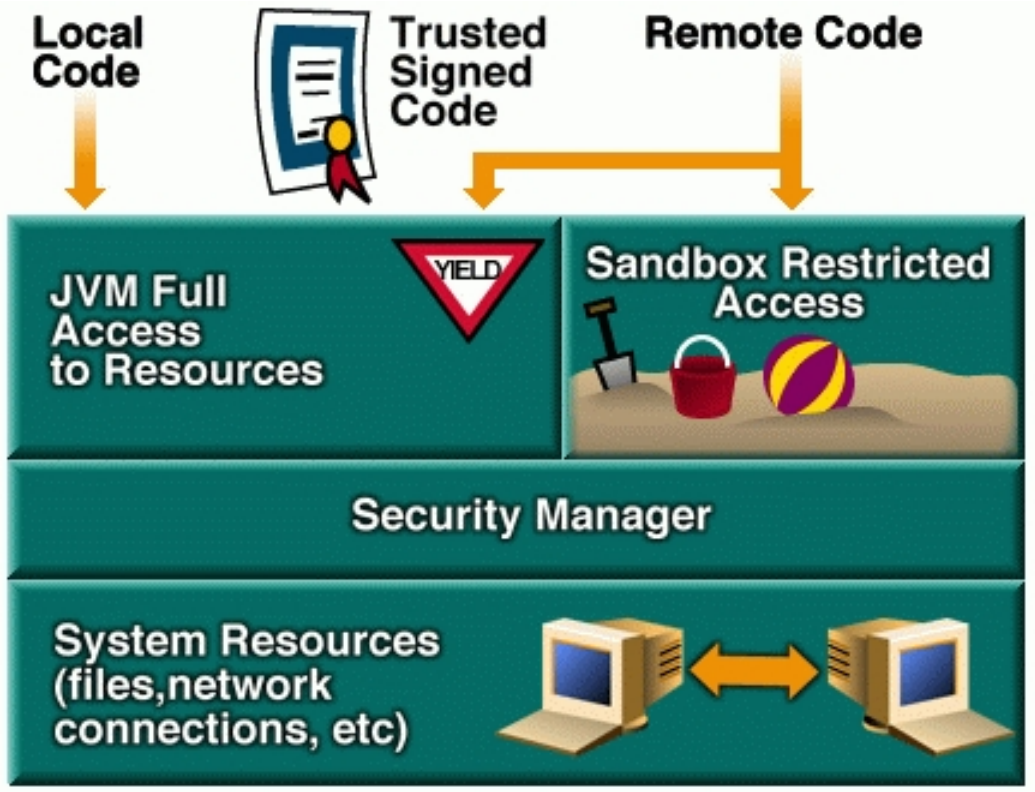
JDK 1.0


Local Code
Remote Code
JVM Full Access to Resources
STOP
Sandbox Restricted Access
Security Manager
System Resources (files,network connections, etc)

JDK 1.1 Security Model

- New concept: “Signed applet”
- Digitally signed applet is treated like local code
  - Packaged in a JAR file along with the signature
  - Full access
- Unsigned applets go through sandbox

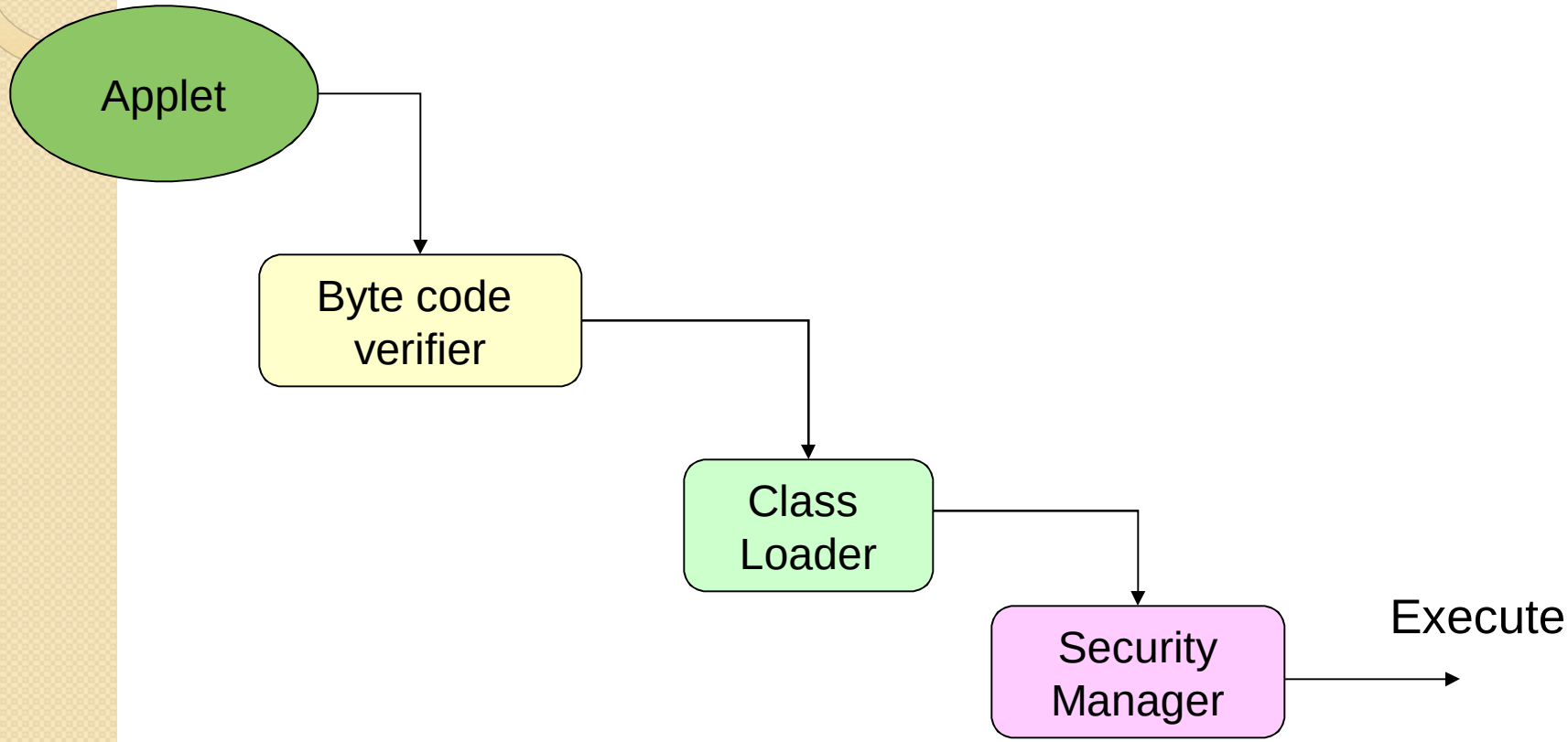
JDK 1.1


Local Code
Trusted Signed Code
Remote Code
YIELD
JVM Full Access to Resources
Sandbox Restricted Access
Security Manager
System Resources (files,network connections, etc)

Check Points

- Compiler and bytecode verifier
  - Allow only legitimate Java bytecode
- Classloader
  - Defines a local name space for the code to ensure its execution doesn't interfere with other programs
- Security manager
  - Apply access restriction to untrusted code

Check Points



A flaw in any of these subsystems may cause a security hole

Bytecode Verifier

- Bytecode – Compilation of class file in a platform-independent form
- The applet bytecode is verified statically to verify the bytecode format
  - Begins with right **“magic number”**
    - **attribute** of all **java class files**
  - Is not **truncated** or have **extra bytes appended**
  - Contains **recognized attributes** of proper length
  - Do not contain any **unrecognized info**

Bytecode Verifier

- Static type checking is difficult to implement
  - Hostile compilers can create instructions that processor can execute but java compiler can not generate
    - How should bytecode verifier detect non-standard bytecode?
  - Flaws can be exploited

Class loader

- Ensure that fundamental Java classes are not replaced by other classes referenced by applets
  - i.e. replace the security manager and skip the security checks
- Class tag indicates which class loader has installed it
  - Determine the privilege level

Class Loader

- Built-in classes have a special class loader
- Applet Class Loader creates its own namespace
- Classes in one namespace can not reference classes in another namespace
- Predefined path for finding classes
  - The built-in classes
  - Applet’s own namespace classes

Security Manager

- Provides dynamic security checks
- All access requests are sent to security manager
  - Based on the class's privileges, the request is denied or honored
- Security managers are customizable
  - Good or bad?