COMP 3355
Mobile Code Security

K.P. Chow

University of Hong Kong

Mobile Code

- Dynamically downloadable executable content is referred to as mobile code
- Includes platform-independent executable content which is transferred over a communication network, crossing different security perimeters, and is automatically executed upon arrival at the destination host
- Examples are Java applets, ActiveX controls, JavaScript, Safe-Tcl, servlet
- How about “browser helper applications” and plug-ins?

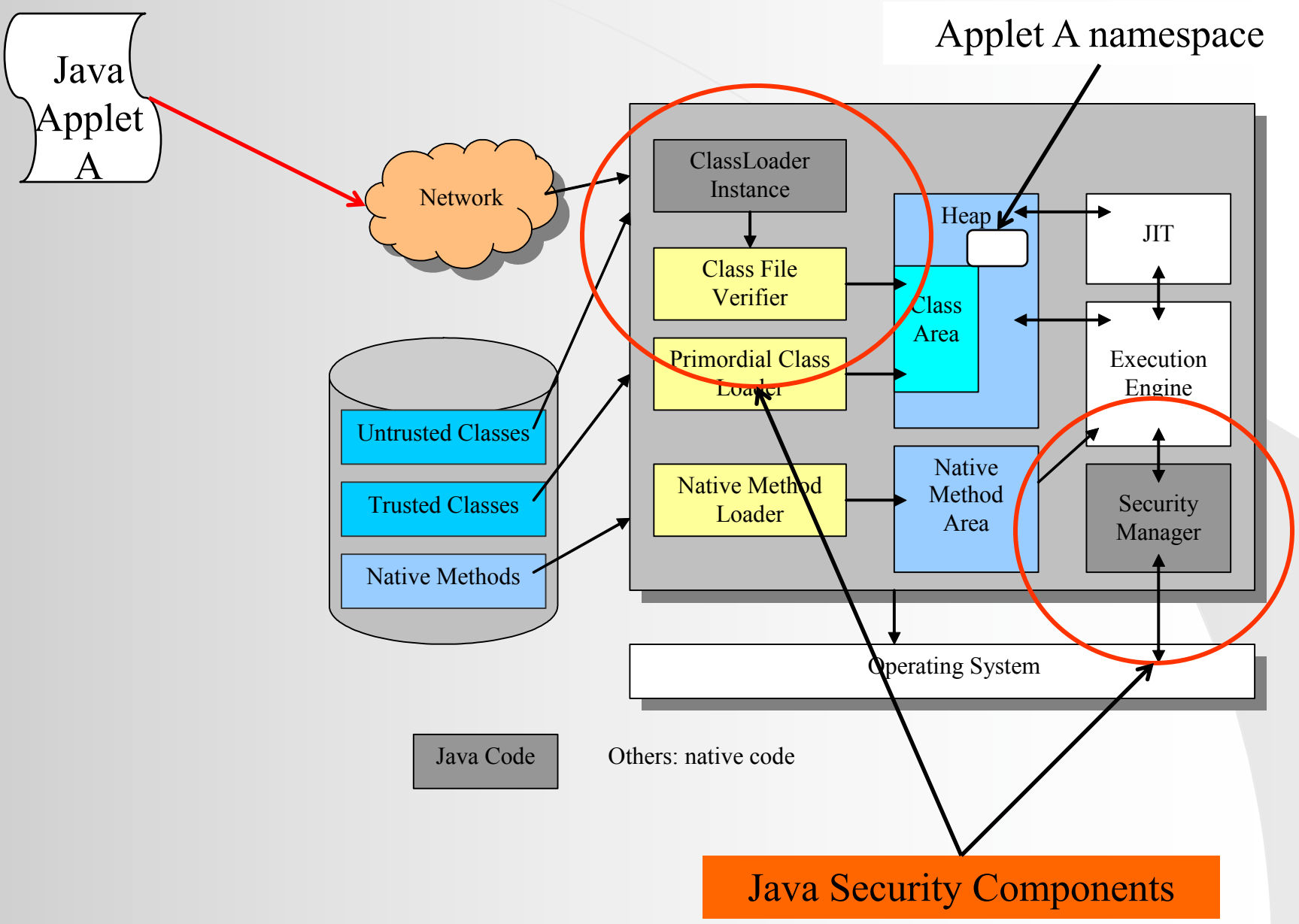
Mobile Code Security Mechanisms

- Code signing
- Least privileges
- Code monitoring: resources usage monitoring
- Mobile code runtime environment
  - Memory model partition the memory into safe and unsafe regions
  - Safety check ensure every memory access refers to a safe memory location, e.g. Java type safety
  - Resource access control by security manager, e.g. Java access controller
- Proof-carrying code: the proof that a piece of mobile code adheres to a safety policy is encoded and transmitted together with the code

Java Safety

- Safety denotes absence of undesirable behavior that can cause system hazards, e.g.
  - Java manages memory by reference and does not allow pointer arithmetic
  - Java provides “final” modifier, which disables overriding class or method
  - Java supports automatic memory management, which prevents memory leak and covert channel
- Java is a strongly typed language
  - An object must always be accessed in the same way and illegal casting is impossible, e.g. an integer cannot be casted to a pointer and access anywhere in the system
  - Static type checking by bytecode verifier protects against forget pointers, access restriction violation, stack overflows, …
  - Dynamic type checking ensure no array bound overflows or type incompatibilities

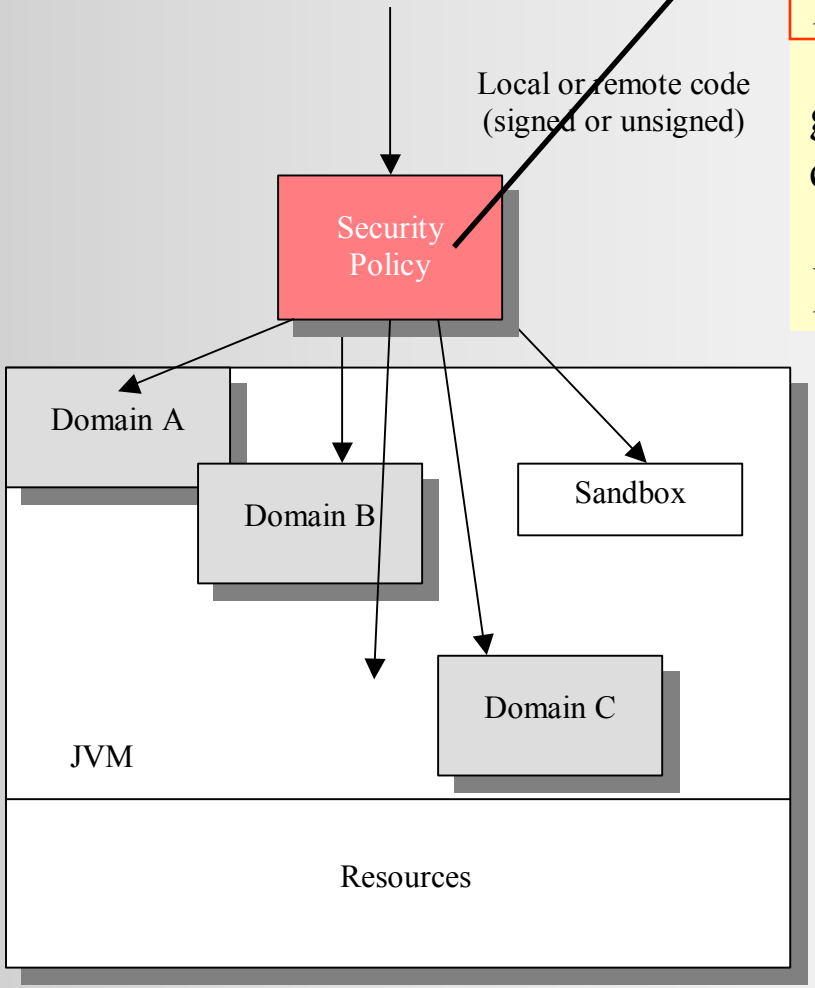
Java Applet Execution



Java 2 Security Model

```
grant codeBase "http://www.oreilly.com/" {
    permission java.io.FilePermission
        "/tmp/*", "read";
    permission java.lang.RuntimePermission
        "queuePrintJob";
}
```

```
grant signedBy "HKGovt,HKSAR",
codeBase "http://info.gov.hk" {
    permission java.security.AllPermission;
};
```

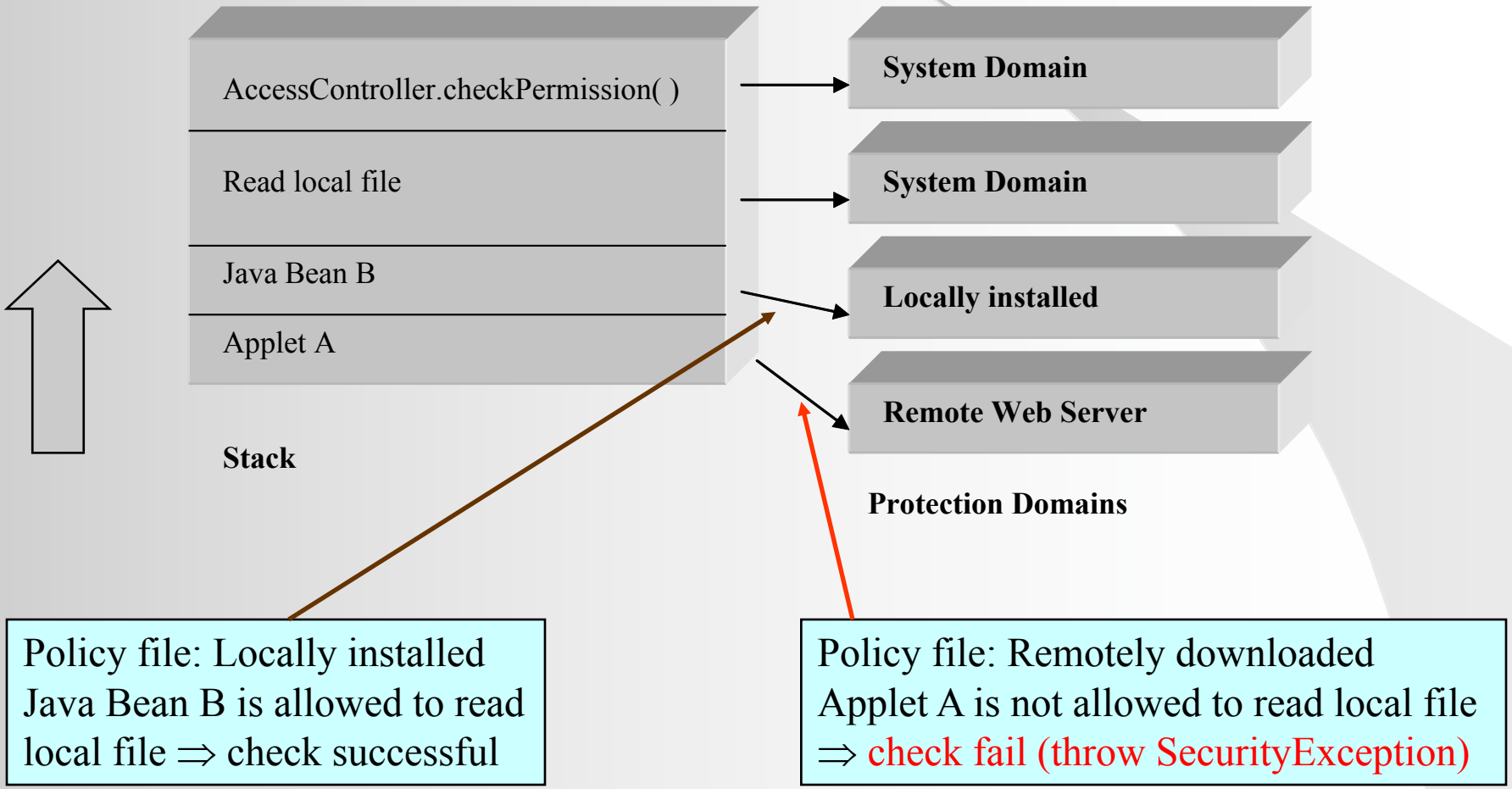


A protection domain

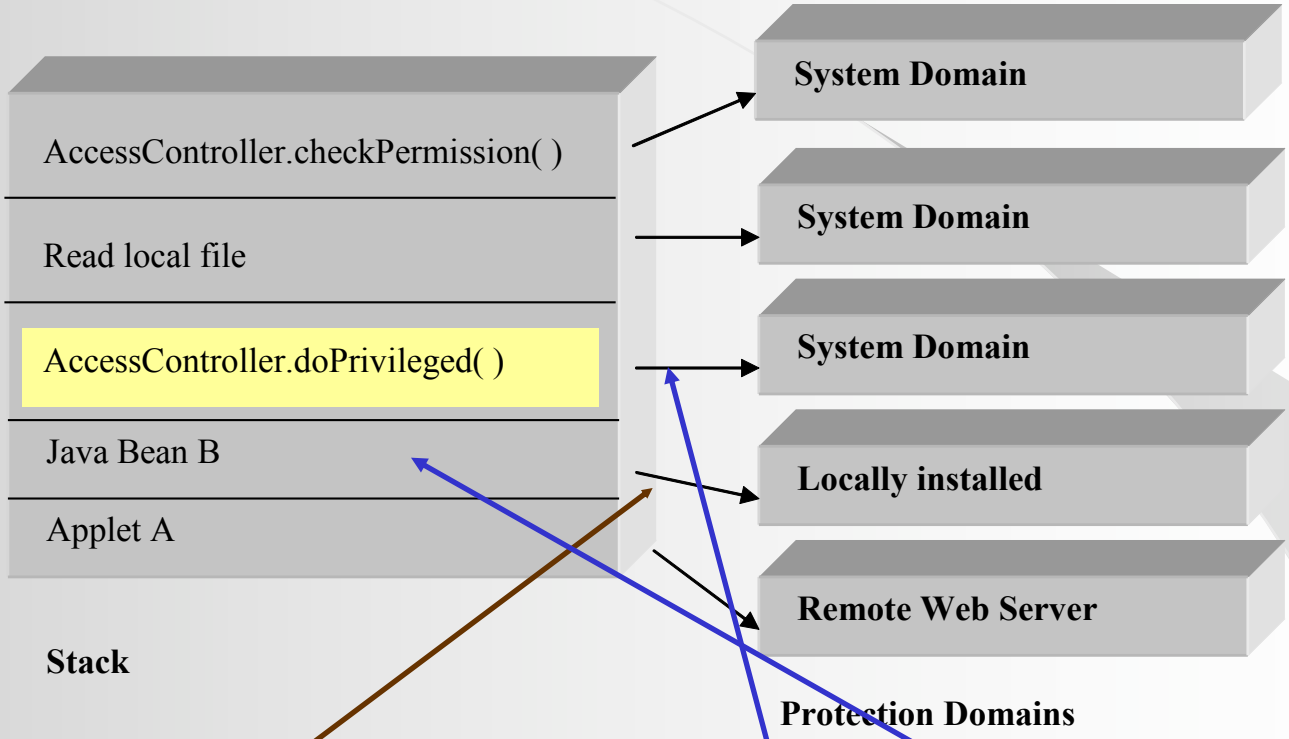
Java Applet Run-Time Access Control

- A Java bean B is installed on a desktop PC which can read local files, the ProtectionDomain of the bean's class has permission to read local files
- An applet A loaded from a Web server that calls the bean B, and A does not have local file read permission
- When A calls B, and B try to read local file F, AccessController.checkPermission("Local file F", "read") will be called
- checkPermission("Local file F", "read") checks against B's protection domain will be succeed while checkPermission("Local file F", "read") checks against A's protection domain will be fail, and SecurityException will be thrown

Java Applet Run-Time Access Control (Stack Check)



Java Applet Run-Time Access Control (Enable Privilege)



Last check here ⇒ check successful

Policy file: Locally installed Java Bean B is allowed to read local file

AccessController.doPrivileged() makes an annotation on the stack frame indicating that checkPermission() will stop *here* when search through the stack frames

Java Signed Applet key issue

- There is a program (or a piece of code) sent from the Web server to the client (i.e. the browser)
- Can I have an easy Yes/No ‘test’ to decide whether the program is safe to run or not?
- The PKI (Public Key Infrastructure) and the browser provides one such solution

The Signed Applet Example

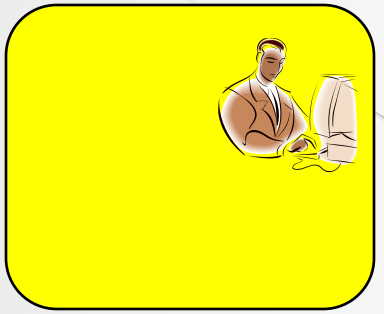
- Signed Applet - Java Applet with ‘digital signature’
- Treat the Applet as a ‘document’ from Server to Client
- The Applet will have an extra document, called ‘digital signature’ attached to it.
  - The “Applet + digital_signature” is a Signed Applet
  - When the Server creates this Applet, the Server will put in this digital_signature as well
  - Only the Server (which holds a “private key”) can create this digital_signature
- Client will ‘verify the digital signature’
- If the verification process is ok, Client will allow the Applet to execute
- **Result:** only Applet from trusted server will be executed

The Signed Applet Technology

- What is the technology that the client used, to ‘verify a signed Applet’? - PKI
- Server, will create the digital_signature using “the server’s private key”
- Client, will verify the digital signature, using the server’s public key stored in the ‘Public Key Certificate’’
- The Public Key Certificate of the Server will be sent from Server to Client when the Applet is loaded, or in some previous connections
- Client, using the ‘Root Cert’ + the server’s Public Key Cert + the Signed Applet, to perform the verification

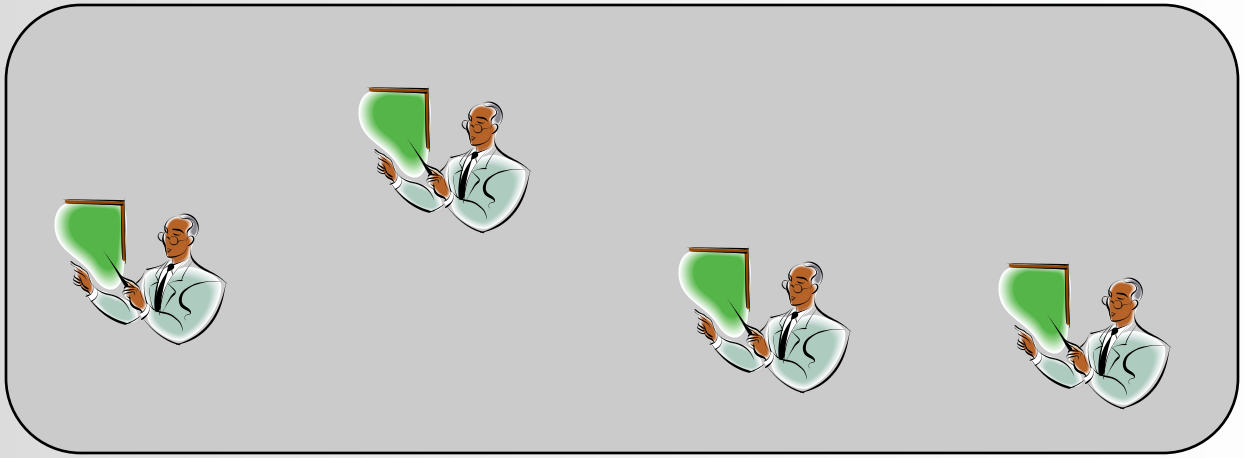
How the “Root Certs” are used?

Server (S1)



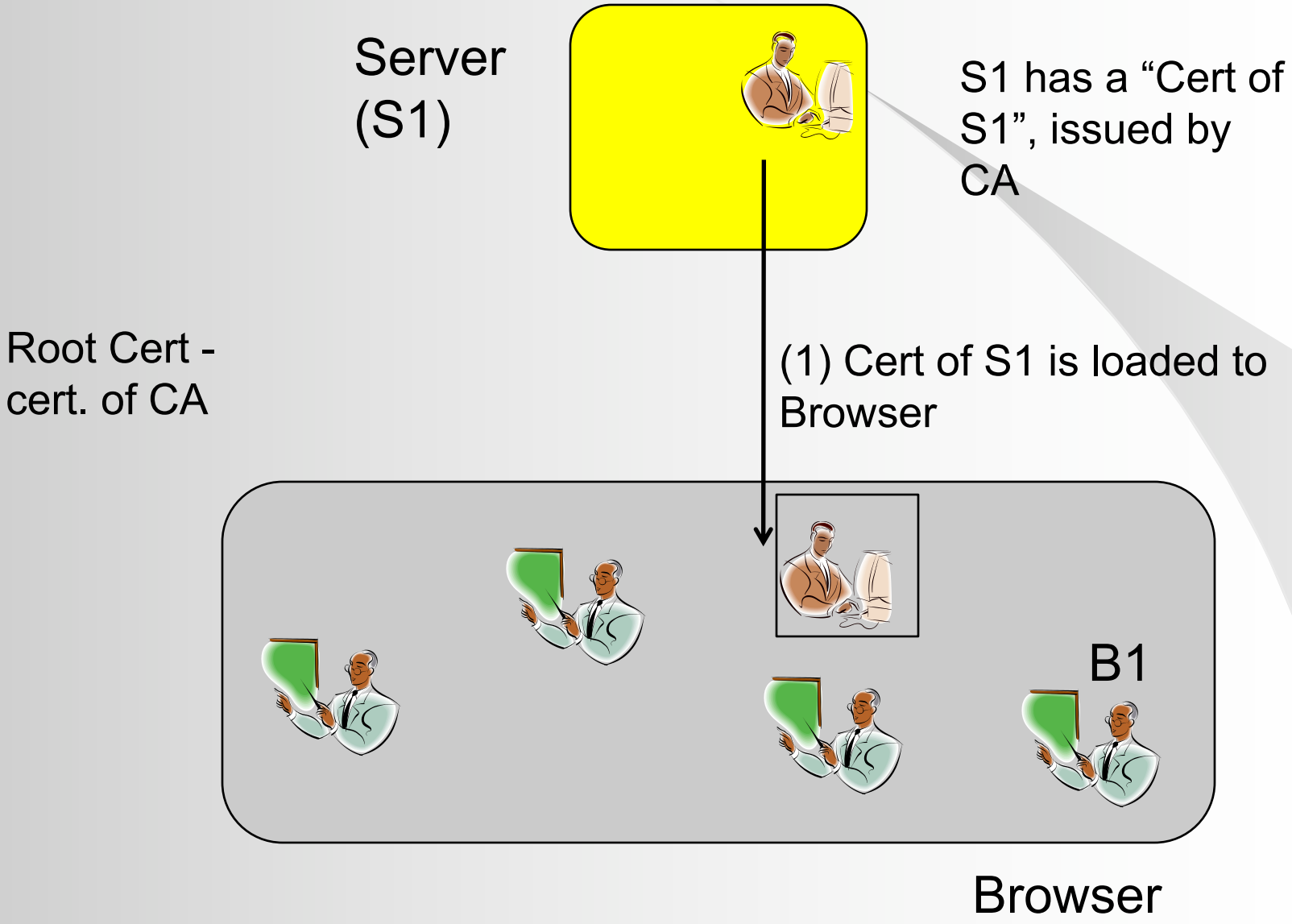
S1 has a “Cert of S1”, issued by CA

Root Cert - cert. of CA

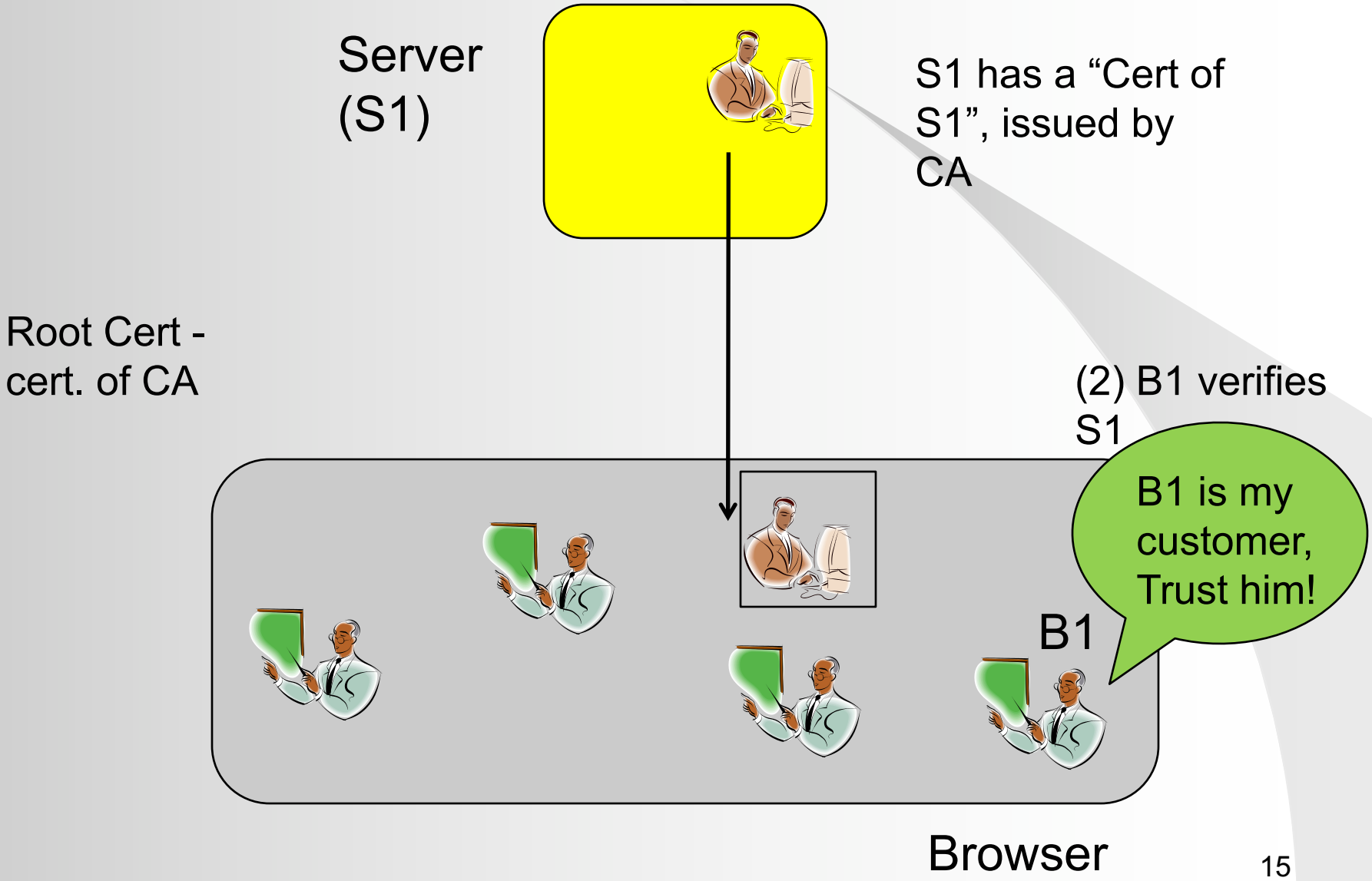


Browser

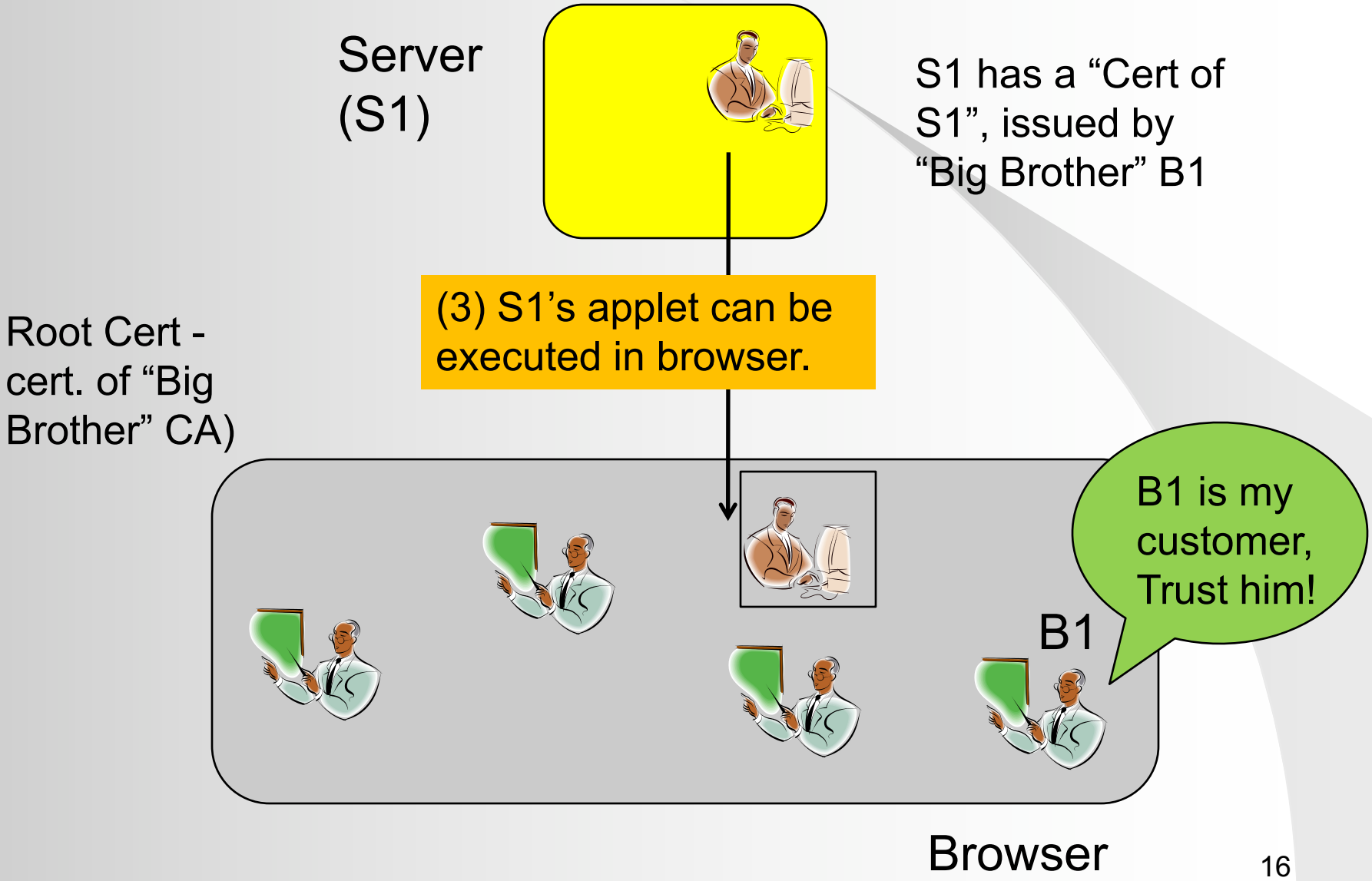
During Authentication (e.g. signed Applet)



During Authentication (e.g. signed Applet)



During Authentication (e.g. signed Applet)



ActiveX Controls

- ActiveX controls
  - Microsoft's collection of technologies and protocols for downloading executable code over the Internet
  - based on a technology that was not originally developed with network security issues in mind
  - something "between" a plug-in and a Java applet: extend browser functionality and allow automatically downloaded with a web page
- Types:
  - Native machine code: most dangerous type
  - Java bytecode: usually run inside the Java sandbox
  - A mixture of native code and bytecode

Authenticode

- ActiveX controls can be digitally signed using Authenticode, Microsoft's digital signature technology using public key certificate
- A native code ActiveX control's signature is checked only once: if the signature is valid and the signer is trusted, the control is downloaded
- A Java bytecode ActiveX control's signature is also checked at downloading
- The actual access permissions are determined at runtime

Authenticode Access Permission

- When IE downloads potentially dangerous content, it checks to see whether the code is digitally signed by a trusted certificate.
- Depending on your browser Security setting (High, Medium, Medium-Low, Low, or Custom) for the zone the content is originating from and the success or failure of the Authenticode verification process, the browser determines whether or not to automatically run the content.
- When prompted to allow or deny signed content, the user is allowed to inspect the signer's digital certificate.

JavaScript

- JavaScript provides a means of commanding the browser, including its graphical elements, from an HTML file
- JavaScript is inherently more secure than Java applets because it cannot directly access the file system or open network connections (sandbox)
- JavaScript are permitted to access only to data in the current document or closely related document (from the same site). No access is granted to the local file system, memory space of other running programs, etc.

JavaScript Security

- Principle: there is not reason to trust randomly encountered code, and should treat them as hostile
- Security issues
  - It has access to all user information in the browser → potential privacy violation
  - DoS attacks are possible by opening a large number of windows until the browser freezes
- Access control management: "**Same Origin Policy**"
  - scripts from one web site do not have access to information such as usernames, passwords, or cookies sent to another site
  - E.g. using the handle returned by window.open(), the browser performs the same origin check

JavaScript Same Origin Policy

- Verifying that the URL of the document in the target window has the same origin as the document containing the calling script
- 2 documents has the same origin if they were loaded from the same server using the same protocol and port
- E.g. a script loaded from http://www.example.com/dir/page.html can gain access to any objects loaded from www.example.com using HTTP

Same Origin Policy Examples

URL of Target Window	Result of the Same Origin Check with www.example.com	Reason
http://www.example.com/index.html	Passes	Same domain and protocol
http://www.example.com/other1/other2/index.html	Passes	Same domain and protocol
http://www.example.com:8080/dir/index.html	Does not pass	Different port
http://www2.example.com/index.html	Does not pass	Different server
http://otherdomain.com/index.html	Does not pass	Different domain
ftp://www.example.com/index.html	Does not pass	Different protocol

External Script

- Externally linked scripts are considered part of the page they are embedded in, and thus can be linked in from other domain
- E.g. the page at http://www.somesite.com/index.html could include the following script:

  <script type="text/javascript"

  src=http://www.example.com/scripts/somescript.js></script>

  – This script will load and work as expected
- The linked scripts are considered part of the page they are linked into. For the above example, if somescript.js tries access another window, it is considered to have come from http://www.somesite.com and subject to the some origin check

Mobile Code Security Mechanisms

	Java Applet	Authenticode	Javascript
Code signing	Yes	Yes	No (*)
Least privileges	Yes	No	No
Mobile code running environment	JVM	IE	Browser
Proof-carrying code	Java security policy	No	Same origin policy

(*) except in Mozilla browsers