CIS 551 / TCOM 401

Computer and Network Security

Spring 2009
Lecture 12

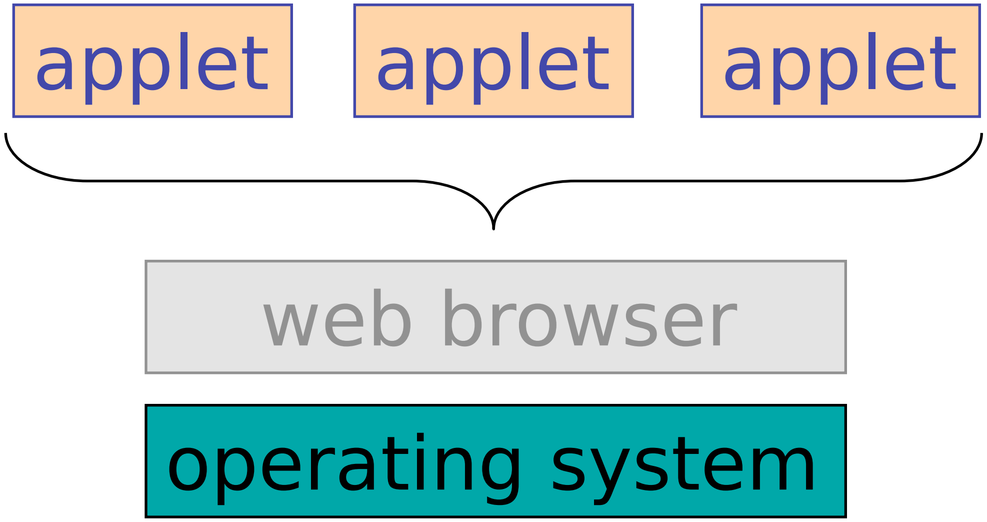
Announcements

- Plan for Today:
  - Java & C# Access Control: Stack Inspection
  - Software certification
- Project 2 reminder
  - Due: Friday, 11:59 pm
- Project 3 will be up soon

- TALK: “Securing Internet Routing”
  - Sharon Goldberg of Princeton University
  - 3:00 *TODAY* (right after class)
  - Wu & Chen Auditorium, Levine

Mobile Code

- Modern languages like Java and C# have been designed for Internet applications and extensible systems



- PDAs, Cell Phones, Smart Cards, …

Java and C# Security

- Static Type Systems
  - Memory safety and jump safety
- Run-time checks for
  - Array index bounds
  - Downcasts
  - Access controls
- Virtual Machine / JIT compilation
  - Bytecode verification
  - Enforces encapsulation boundaries (e.g. private field)
- Garbage Collected
  - Eliminates memory management errors
- Library support
  - Cryptography, authentication, …

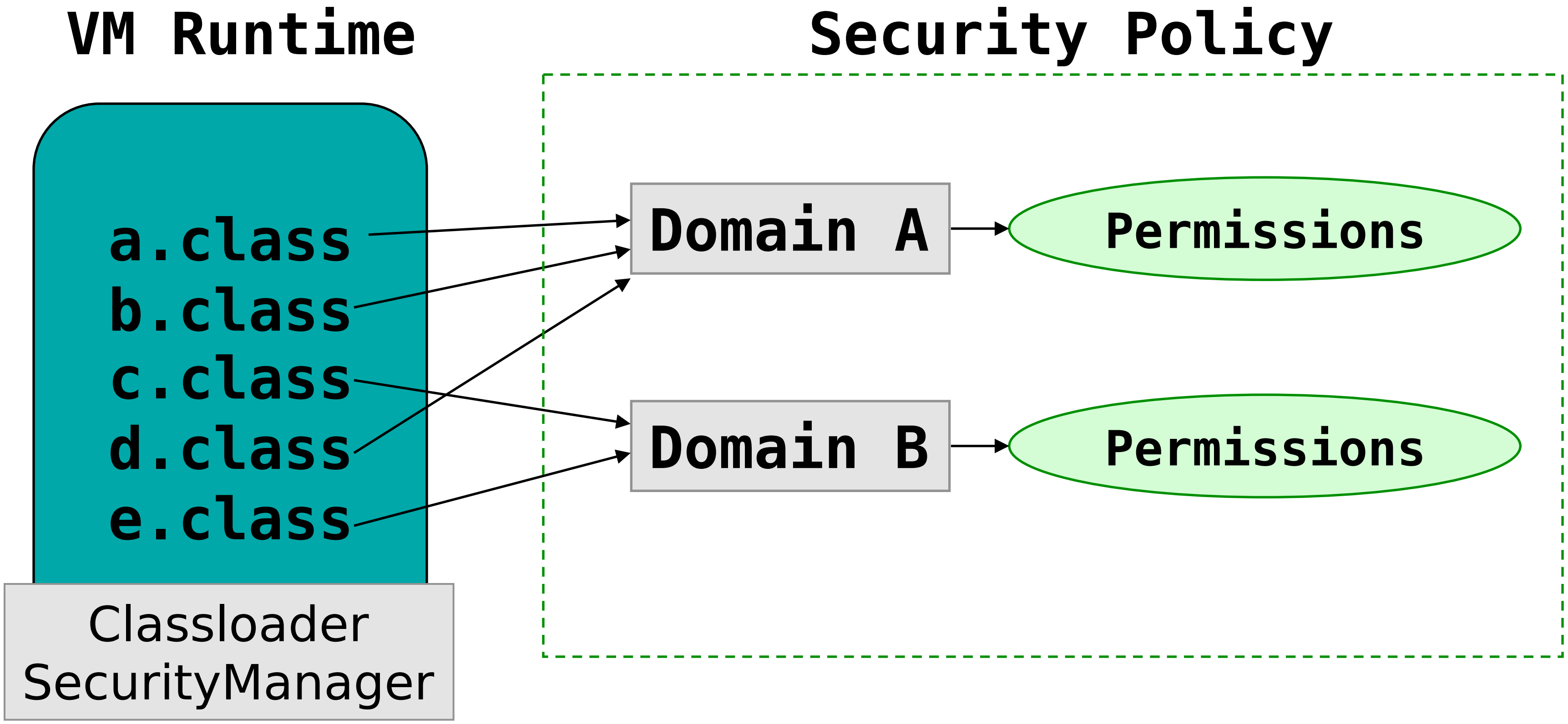
Applet Security Problems

- Protect OS & other valuable resources.
- Applets should not:
  - crash browser or OS
  - execute “rm –rf /”
  - be able to exhaust resources
- Applets should:
  - be able to access *some* system resources (e.g. to display a picture)
  - be isolated from each other
- Principles of least privileges and complete mediation apply

Access Control for Applets

- What level of granularity?
  - Applets can touch some parts of the file system but not others
  - Applets can make network connections to some locations but not others
- Different code has different levels of trustworthiness
  - www.l33t-hax0rs.com vs. www.java.sun.com
- Trusted code can call untrusted code
  - e.g. to ask an applet to repaint its window
- Untrusted code can call trusted code
  - e.g. the paint routine may load a font
- How is the access control policy specified?
- How is it enforced?

Java Security Model



Kinds of Permissions

- java.security.Permission Class

```
perm = new java.io.FilePermission("/tmp/abc","read");

java.security.AllPermission
java.security.SecurityPermission
java.security.UnresolvedPermission
java.awt.AWTPermission
java.io.FilePermission
java.io.SerializablePermission
java.lang.reflect.ReflectPermission
java.lang.RuntimePermission
java.net.NetPermission
java.net.SocketPermission
…
```

Code Trustworthiness

- How does one decide what protection domain the code is in?
  - Source (e.g. local or applet)
  - Digital signatures
    - C# calls this “evidence based”
- How does one decide what permissions a protection domain has?
  - Configurable – administrator file or command line
- Enforced by the classloader

Example Java Policy

```
grant codeBase “http://www.l33t-hax0rz.com/*” {
  permission java.io.FilePermission(“/tmp/*”, “read,write”);
}

grant codeBase “file://$JAVA_HOME/lib/ext/*” {
  permission java.security.AllPermission;
}

grant signedBy “trusted-company.com” {
  permission java.net.SocketPermission(…);
  permission java.io.FilePermission(“/tmp/*”, “read,write”);
  …
}
```

Policy information stored in:

`$JAVA_HOME/lib/security/java.policy`
`$USER_HOME/.java.policy`
(or passed on command line)

Example Trusted Code

Code in the System protection domain

```
void fileWrite(String filename, String s) {
  SecurityManager sm = System.getSecurityManager();
  if (sm != null) {
    FilePermission fp = new FilePermission(filename,“write”);
    sm.checkPermission(fp);
    /* … write s to file filename (native code) … */
  } else {
    throw new SecurityException();
  }
}
```

```
public static void main(…) {
  SecurityManager sm = System.getSecurityManager();
  FilePermission fp = new FilePermission(“/tmp/*”,“write,…”);
  sm.enablePrivilege(fp);
  UntrustedApplet.run();
}
```

Example Client

Applet code obtained from
http://www.l33t-hax0rz.com/

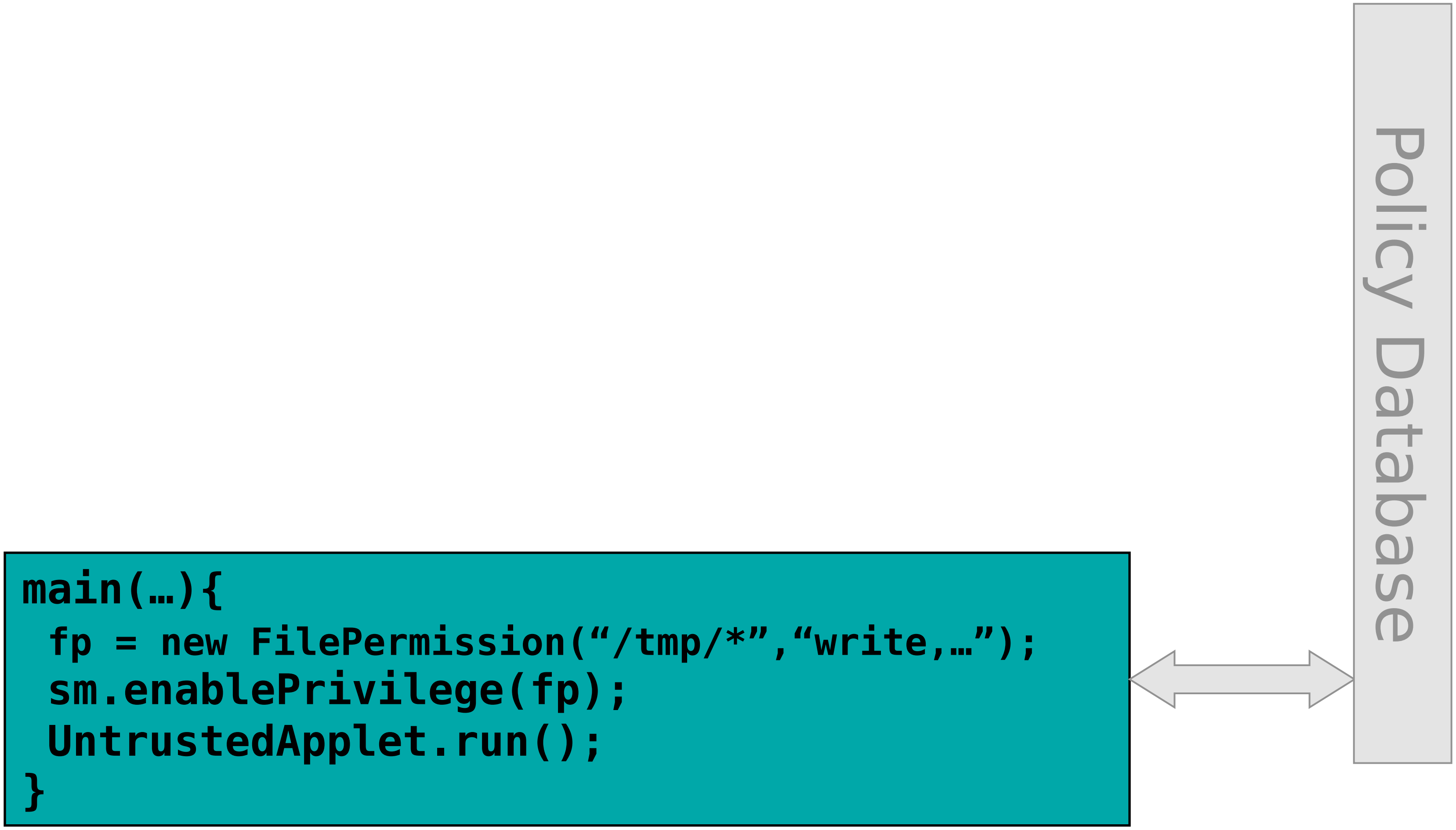
```
class UntrustedApplet {
  void run() {
    ...
    s.FileWrite(“/tmp/foo.txt”, “Hello!”);
    ...
    s.FileWrite(“/home/stevez/important.tex”, “kwijibo”);
    ...
  }
}
```

Stack Inspection

- Stack frames are annotated with their protection domains and any enabled privileges.

- During inspection, stack frames are searched from most to least recent:
  - fail if a frame belonging to someone not authorized for privilege is encountered
  - succeed if activated privilege is found in frame

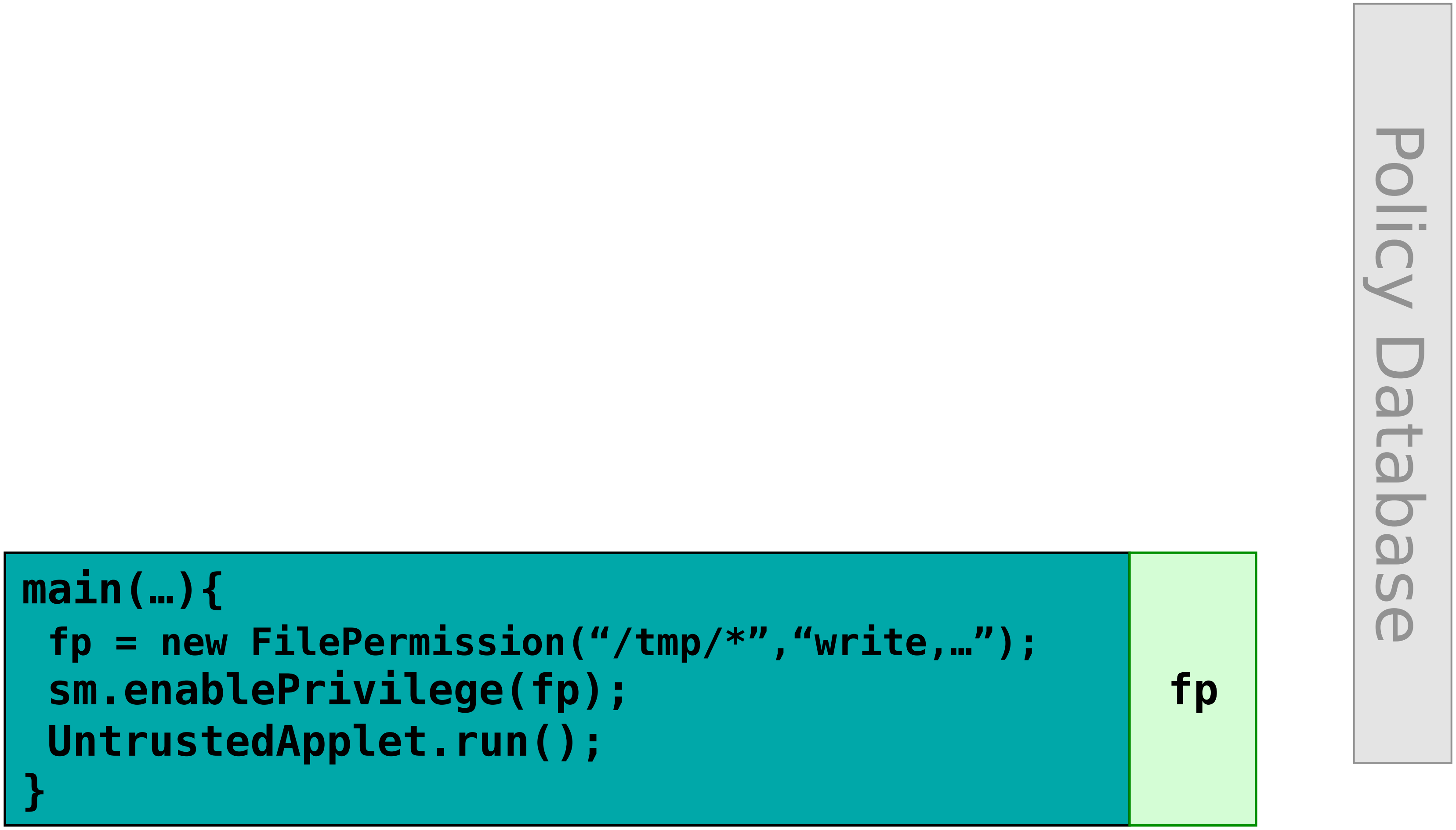
Stack Inspection Example



```
main(…){
 fp = new FilePermission(“/tmp/*”,“write,…”);
 sm.enablePrivilege(fp);
 UntrustedApplet.run();
}
```

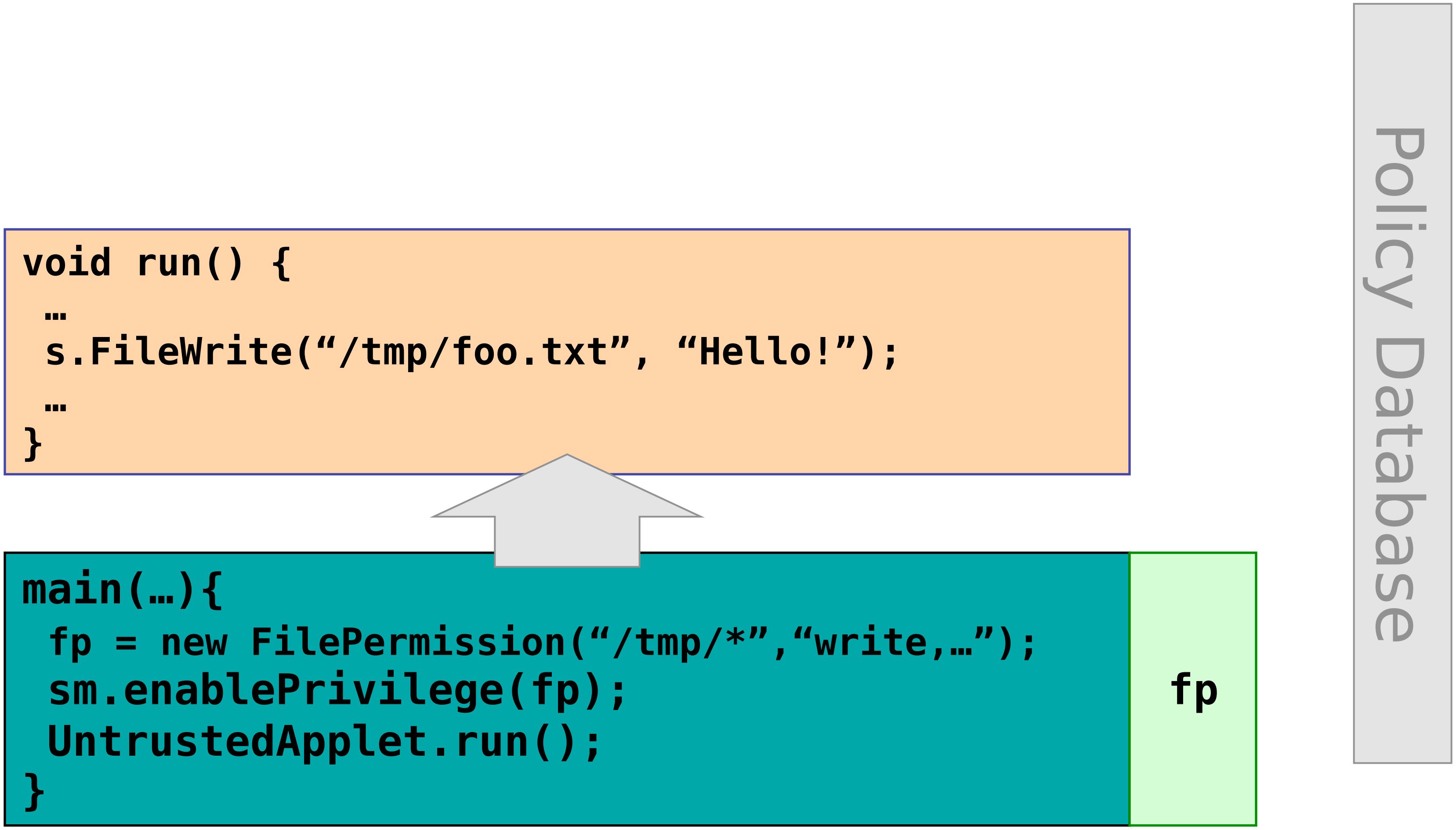
Policy Database

Stack Inspection Example



```
main(…){
 fp = new FilePermission(“/tmp/*”,“write,…”);
 sm.enablePrivilege(fp);
 UntrustedApplet.run();
}
```

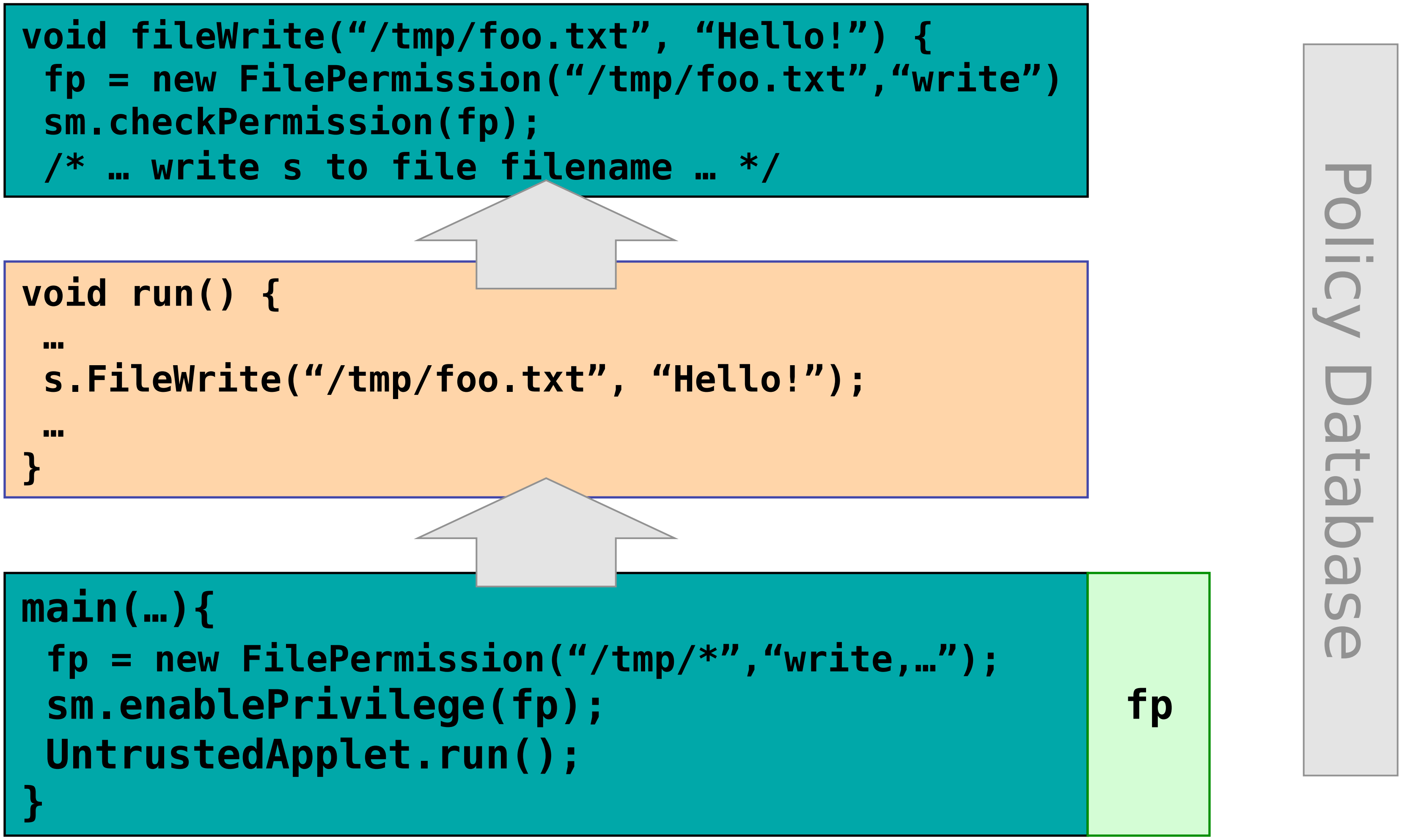
Stack Inspection Example



```
void run() {
 …
 s.FileWrite(“/tmp/foo.txt”, “Hello!”);
 …
}
```

```
main(…){
 fp = new FilePermission(“/tmp/*”,“write,…”);
 sm.enablePrivilege(fp);
 UntrustedApplet.run();
}
```

Stack Inspection Example

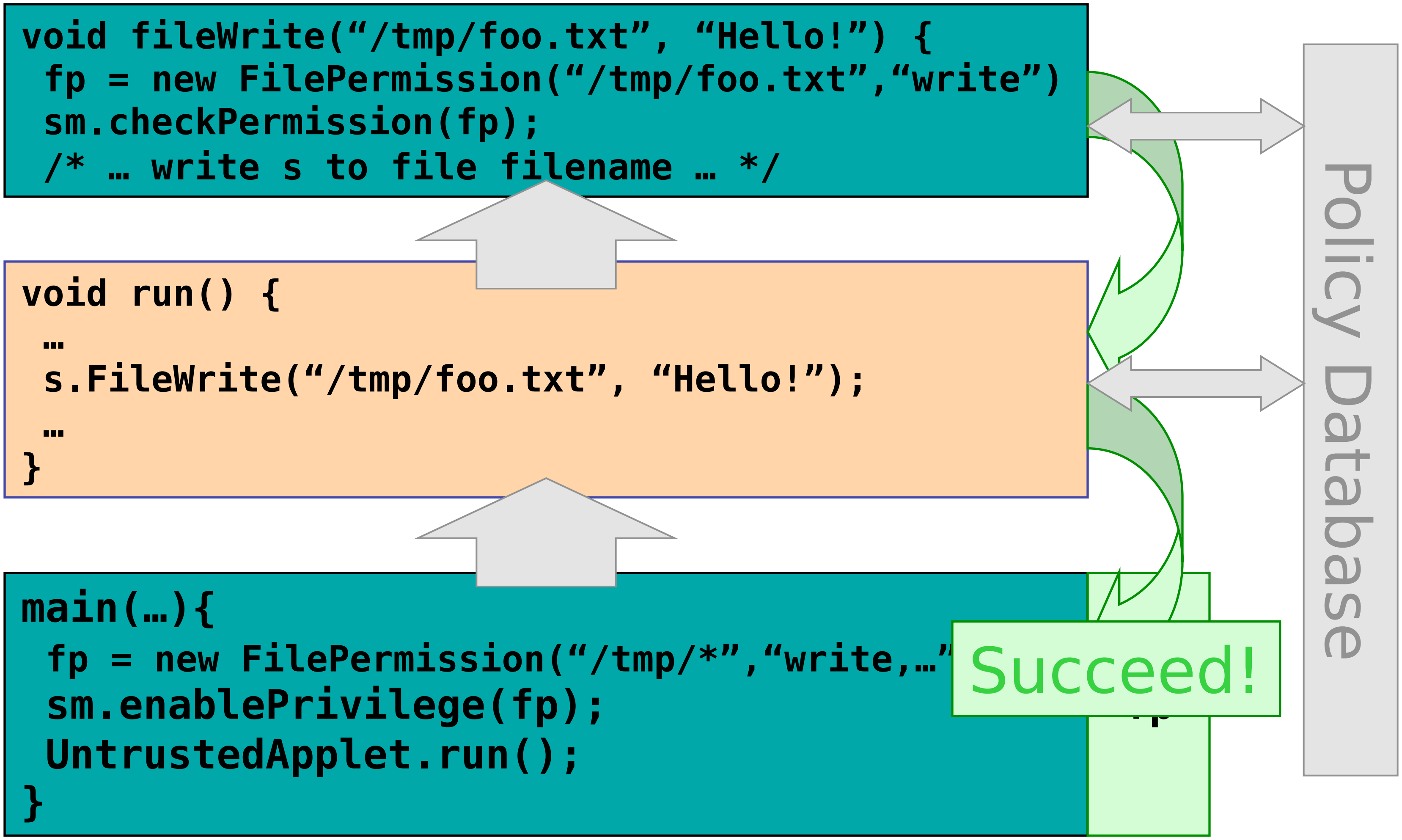


```
void fileWrite(“/tmp/foo.txt”, “Hello!”) {
 fp = new FilePermission(“/tmp/foo.txt”,“write”)
 sm.checkPermission(fp);
 /* … write s to file filename … */
```

```
void run() {
 …
 s.FileWrite(“/tmp/foo.txt”, “Hello!”);
 …
}
```

```
main(…){
 fp = new FilePermission(“/tmp/*”,“write,…”);
 sm.enablePrivilege(fp);
 UntrustedApplet.run();
}
```

Stack Inspection Example

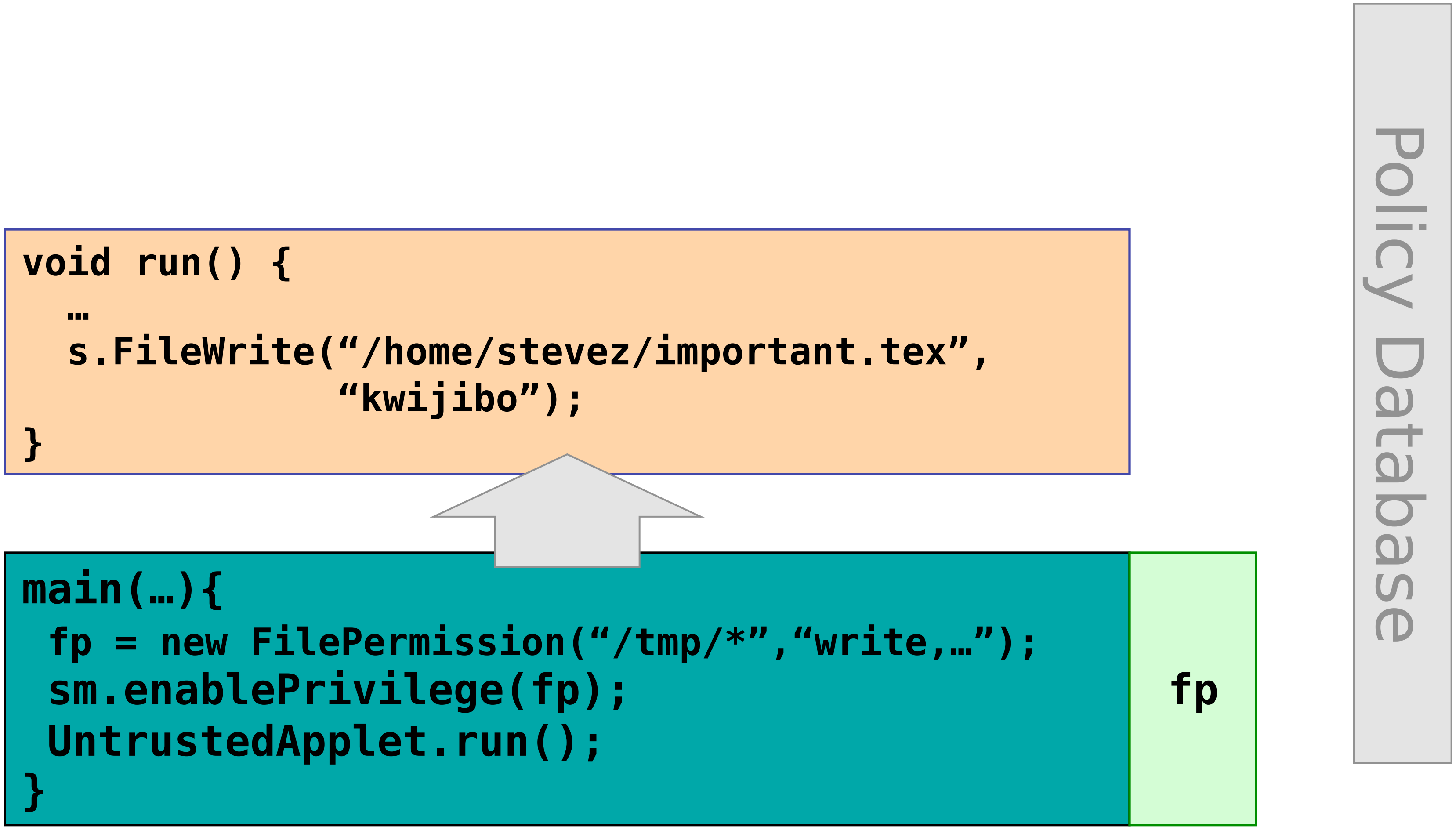


```
void fileWrite(“/tmp/foo.txt”, “Hello!”) {
 fp = new FilePermission(“/tmp/foo.txt”,“write”)
 sm.checkPermission(fp);
 /* … write s to file filename … */
```

```
void run() {
 …
 s.FileWrite(“/tmp/foo.txt”, “Hello!”);
 …
}
```

```
main(…){
 fp = new FilePermission(“/tmp/*”,“write,…”
 sm.enablePrivilege(fp);
 UntrustedApplet.run();
}
```

Stack Inspection Example



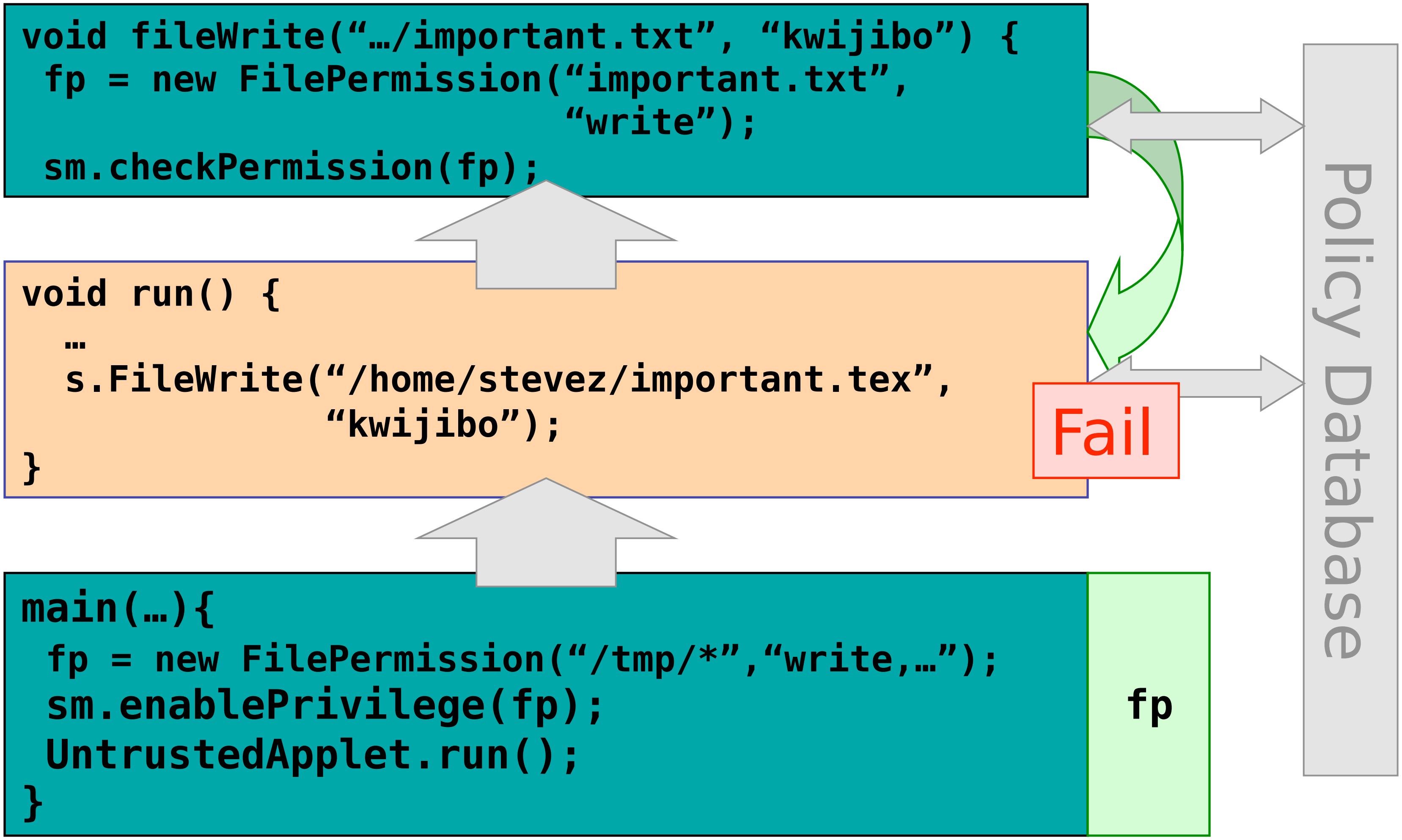
```
void run() {
  …
  s.FileWrite(“/home/stevez/important.tex”,
                “kwijibo”);
}
```

```
main(…){
 fp = new FilePermission(“/tmp/*”,“write,…”);
 sm.enablePrivilege(fp);
 UntrustedApplet.run();
}
```

fp

Policy Database

Stack Inspection Example



```
void fileWrite("…/important.txt", "kwijibo") {
 fp = new FilePermission("important.txt",
                         "write");
 sm.checkPermission(fp);
```

```
void run() {
  …
  s.FileWrite("/home/stevez/important.tex",
              "kwijibo");
}
```

```
main(…){
 fp = new FilePermission("/tmp/*","write,…");
 sm.enablePrivilege(fp);
 UntrustedApplet.run();
}
```

Other Possibilities

- The **`fileWrite`** method could enable the write permission itself
  - Potentially dangerous, should not base which file to write on data provided by the applet
  - … but no enforcement in Java (information flow would help here)

- A trusted piece of code could *disable* a previously granted permission
  - Terminate the stack inspection early

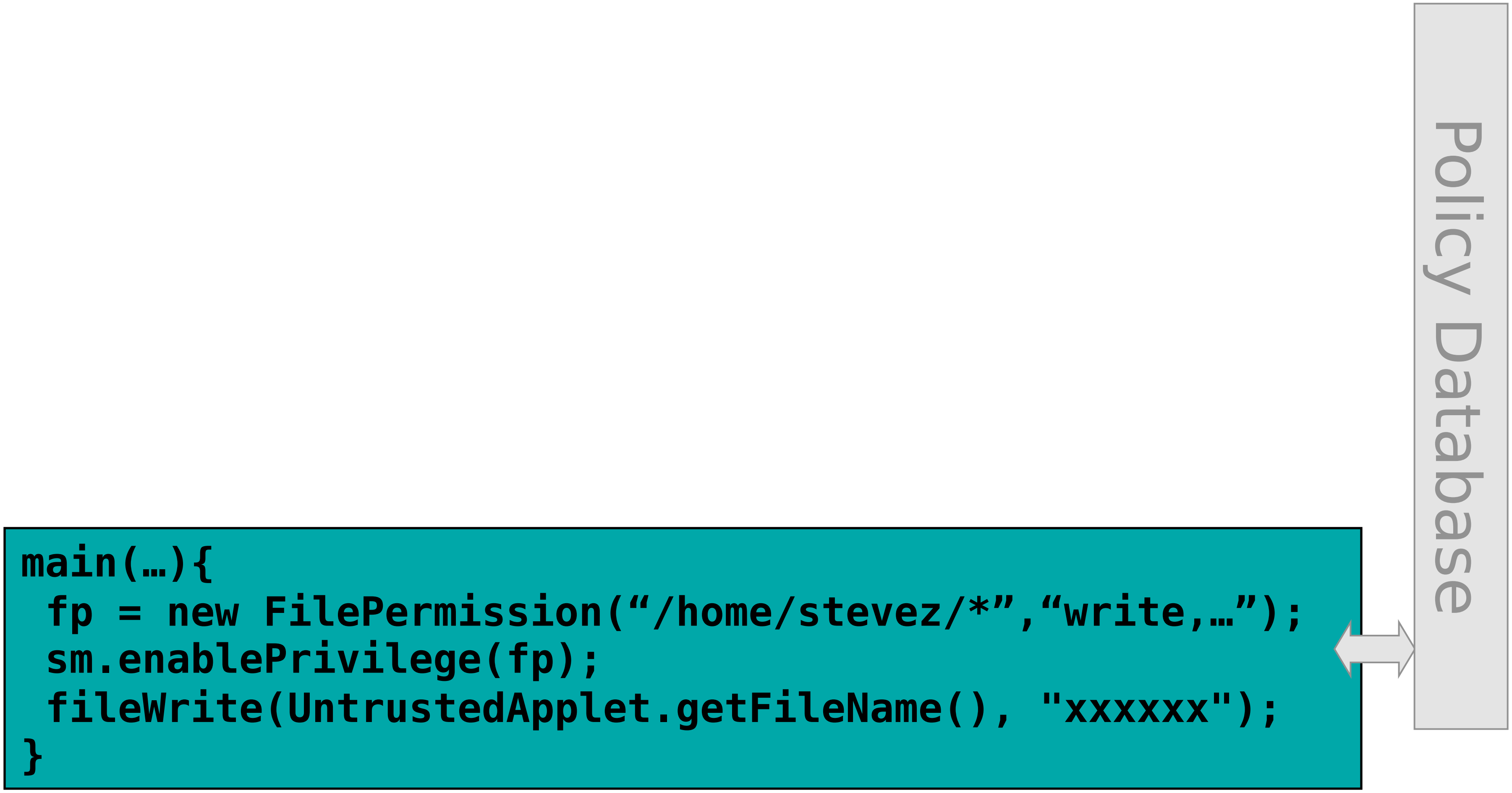
Stack Inspection Algorithm

```
checkPermission(T) {
  // loop newest to oldest stack frame
  foreach stackFrame {
    if (local policy forbids access to T by class executing in
        stack frame) throw ForbiddenException;

    if (stackFrame has enabled privilege for T)
      return;  // allow access

    if (stackFrame has disabled privilege for T)
      throw ForbiddenException;
  }

  // end of stack
  if (Thunderbird || …) throw ForbiddenException;
  if (MS IE || JDK || …) return;
}
```

Stack Inspection

- Stack inspection seems appealing:
  - Fine grained, flexible, configurable policies
  - Distinguishes between code of varying degrees of trust
- But…
  - How do we understand what the policy is?
  - Semantics tied to the operational behavior of the program (defined in terms of stacks!)
  - Changing the program (e.g. optimizing it) may change the security policy
  - Policy is distributed throughout the software, and is not apparent from the program interfaces.
  - Is it any good?
  - It's not complete!

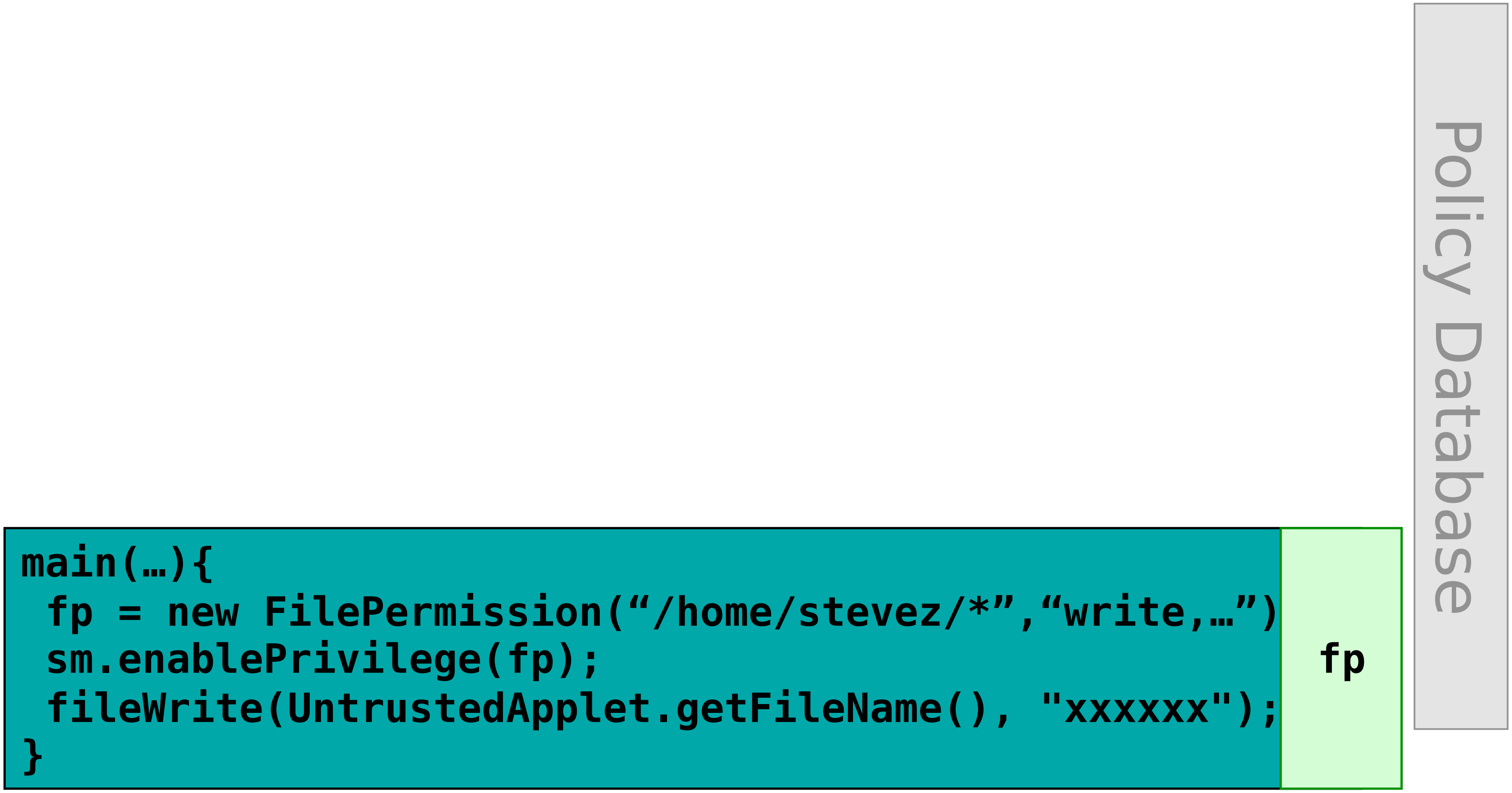
Problem with Stack Inspection



```
main(…){
 fp = new FilePermission(“/home/stevez/*”,“write,…”);
 sm.enablePrivilege(fp);
 fileWrite(UntrustedApplet.getFileName(), "xxxxxx");
}
```

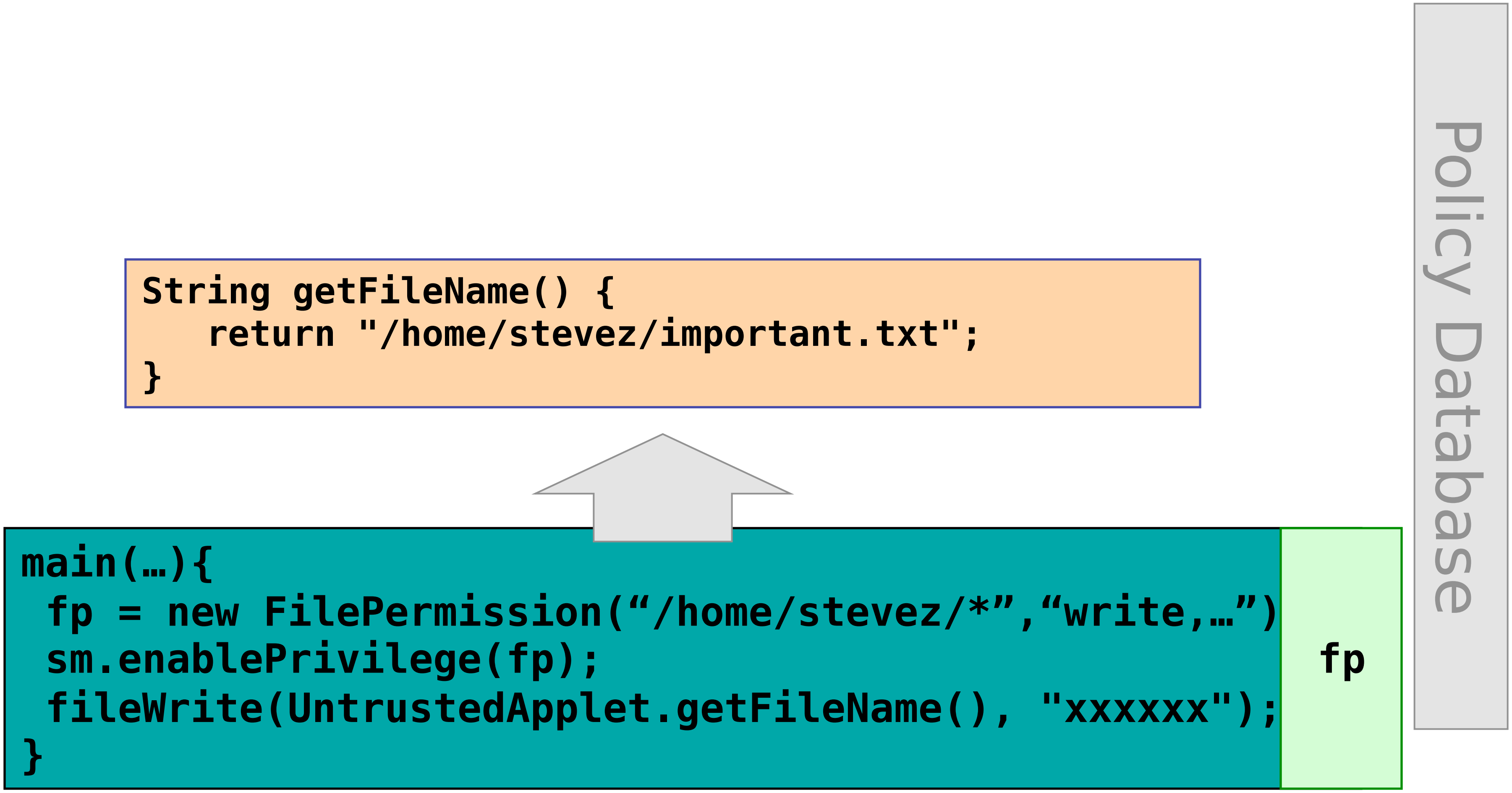
Policy Database

Problem with Stack Inspection



```
main(…){
 fp = new FilePermission(“/home/stevez/*”,“write,…”)
 sm.enablePrivilege(fp);
 fileWrite(UntrustedApplet.getFileName(), "xxxxxx");
}
```

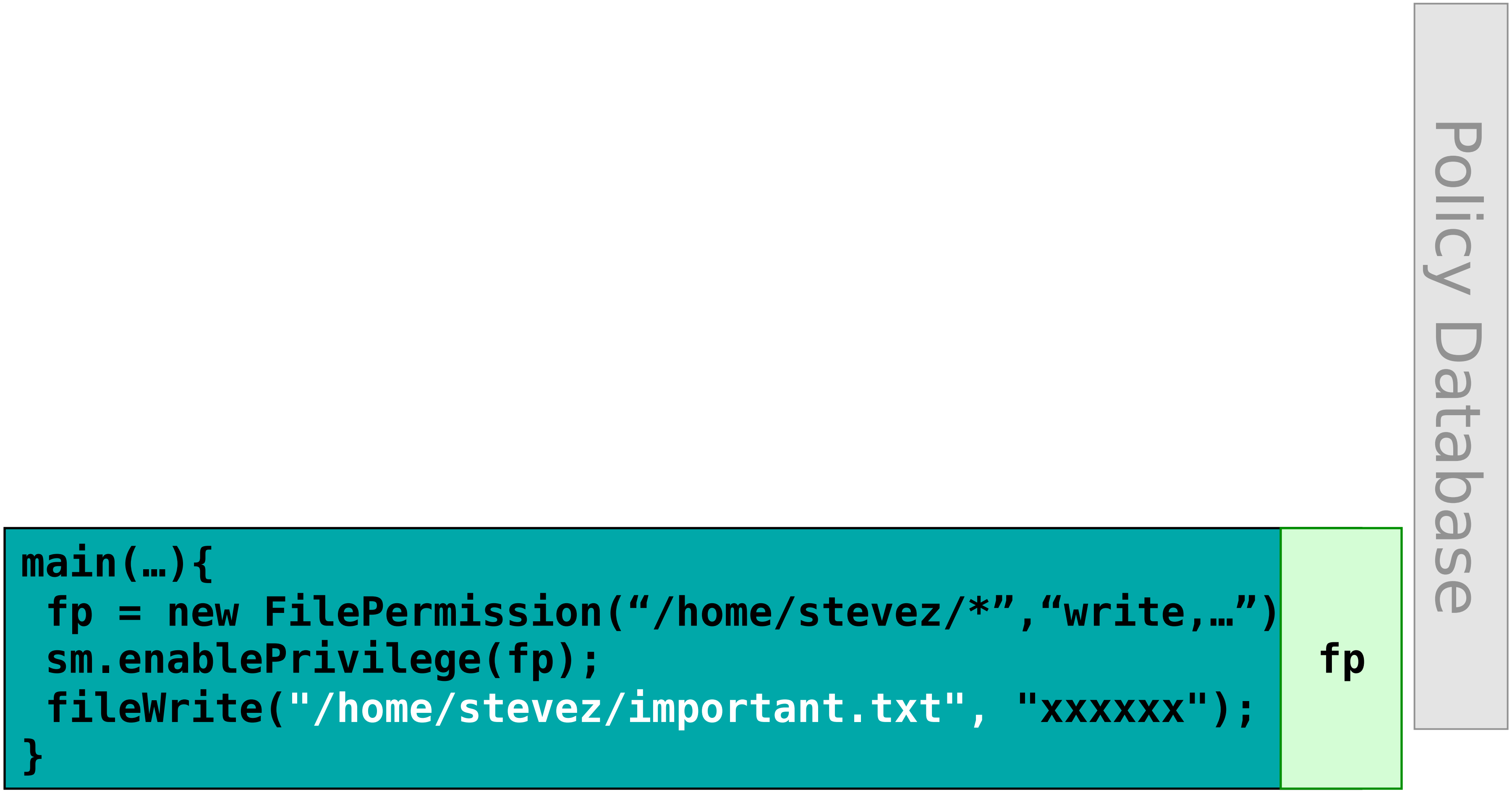
Problem with Stack Inspection



```
String getFileName() {
   return "/home/stevez/important.txt";
}
```

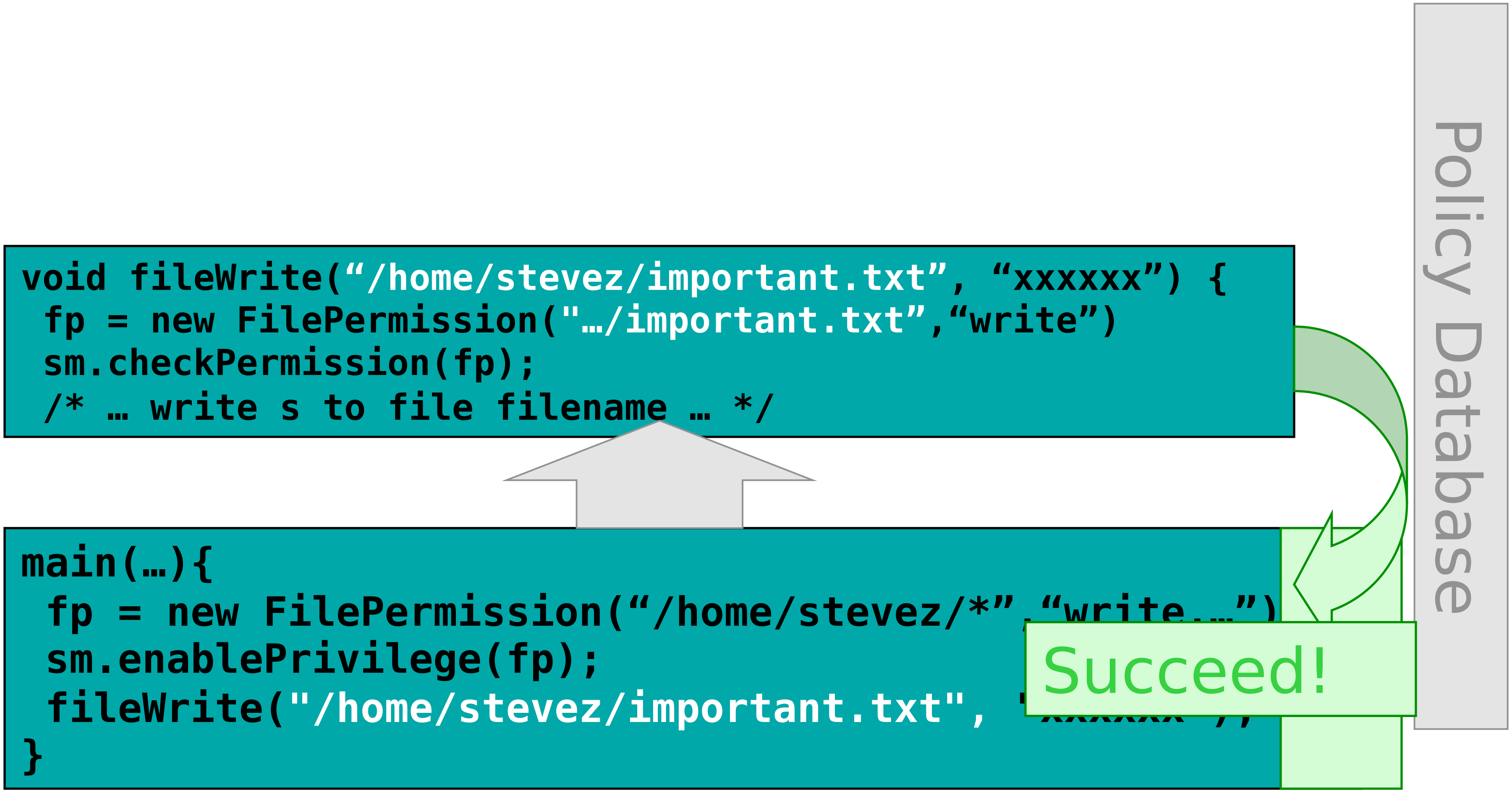
```
main(…){
 fp = new FilePermission(“/home/stevez/*”,“write,…”)
 sm.enablePrivilege(fp);
 fileWrite(UntrustedApplet.getFileName(), "xxxxxx");
}
```

Problem with Stack Inspection



```
main(…){
 fp = new FilePermission(“/home/stevez/*”,“write,…”)
 sm.enablePrivilege(fp);
 fileWrite("/home/stevez/important.txt", "xxxxxx");
}
```

Problem with Stack Inspection



```
void fileWrite(“/home/stevez/important.txt”, “xxxxxx”) {
 fp = new FilePermission("…/important.txt”,“write”)
 sm.checkPermission(fp);
 /* … write s to file filename … */
```

```
main(…){
 fp = new FilePermission(“/home/stevez/*”,“write,…”)
 sm.enablePrivilege(fp);
 fileWrite("/home/stevez/important.txt", "xxxxxx");
}
```

Stack Inspection: Final thoughts

- Question: How does taint tracking relate to this problem with stack inspection?

- Related Papers (not required reading):
  - A Systematic Approach to Static Access Control
    François Pottier, Christian Skalka, Scott Smith
  - Stack Inspection: Theory and Variants
    Cédric Fournet and Andrew D. Gordon
  - Understanding Java Stack Inspection
    Dan S. Wallach and Edward W. Felten

Question:

- Suppose you have gone through the cost/benefit and risk analysis to determine the securty requirements for a computer system.
- How do you know whether a system meets its security requirements?

- Class answers:

Assurance methods

- Testing
  - Regression testing, automation tools, etc.
  - Can demonstrate existence of flaw, not absence
- Validation
  - Requirements checking
  - Design and code reviews
    - Sit around table, drink lots of coffee, …
  - Module and system testing
- Formal verification
  - Develop a rigorous (mathematical) specification of the system
  - Prove (using tools or by hand) that the implementation meets the specification
  - Time-consuming, painstaking process
  - Has been done for some systems. (See www.praxis-his.com)

Rainbow Series

DoD Trusted Computer Sys Evaluation Criteria (Orange Book)

Audit in Trusted Systems (Tan Book)

Configuration Management in Trusted Systems (Amber Book)

Trusted Distribution in Trusted Systems (Dark Lavender Book)

Security Modeling in Trusted Systems (Aqua Book)

Formal Verification Systems (Purple Book)

Covert Channel Analysis of Trusted Systems (Light Pink Book)

… many more

http://www.fas.org/irp/nsa/rainbow.htm

Orange Book Requirements (TCSEC)

- TCSEC = Trusted Computer System Evaluation Criteria

- Security Policy
- Accountability
- Assurance
- Documentation

- Next few slides: details not important …
  - Main point: Higher levels require more work …, documentation and configuration management are part of the criteria

Common Criteria

- Three parts
  - CC Documents
    - Protection profiles: requirements for category of systems
      - Functional requirements
      - Assurance requirements
  - CC Evaluation Methodology
  - National Schemes (local ways of doing evaluation)
- Endorsed by 14 countries
- Replaces TCSEC
  - CC adopted 1998
  - Last TCSEC evaluation completed 2000

http://www.niap-ccevs.org/cc-scheme/

http://www.commoncriteriaportal.org/

Protection Profiles

- Requirements for categories of systems
  - Subject to review and certified
- Example: Controlled Access PP (CAPP_V1.d)
  - Security functional requirements
    - Authentication, User Data Protection, Prevent Audit Loss
  - Security assurance requirements
    - Security testing, Admin guidance, Life-cycle support, …
  - Assumes non-hostile and well-managed users
  - Does not consider malicious system developers

Evaluation Assurance Levels 1 – 4

EAL 1: Functionally Tested

- Review of functional and interface specifications
- Some independent testing

EAL 2: Structurally Tested

- Analysis of security functions, including high-level design
- Independent testing, review of developer testing

EAL 3: Methodically Tested and Checked

- Development environment controls; configuration mgmt

EAL 4: Methodically Designed, Tested, Reviewed

- Informal spec of security policy, Independent testing

Evaluation Assurance Levels 5 – 7

EAL 5: Semiformally Designed and Tested

- Formal model, modular design
- Vulnerability search, covert channel analysis

EAL 6: Semiformally Verified Design and Tested

- Structured development process

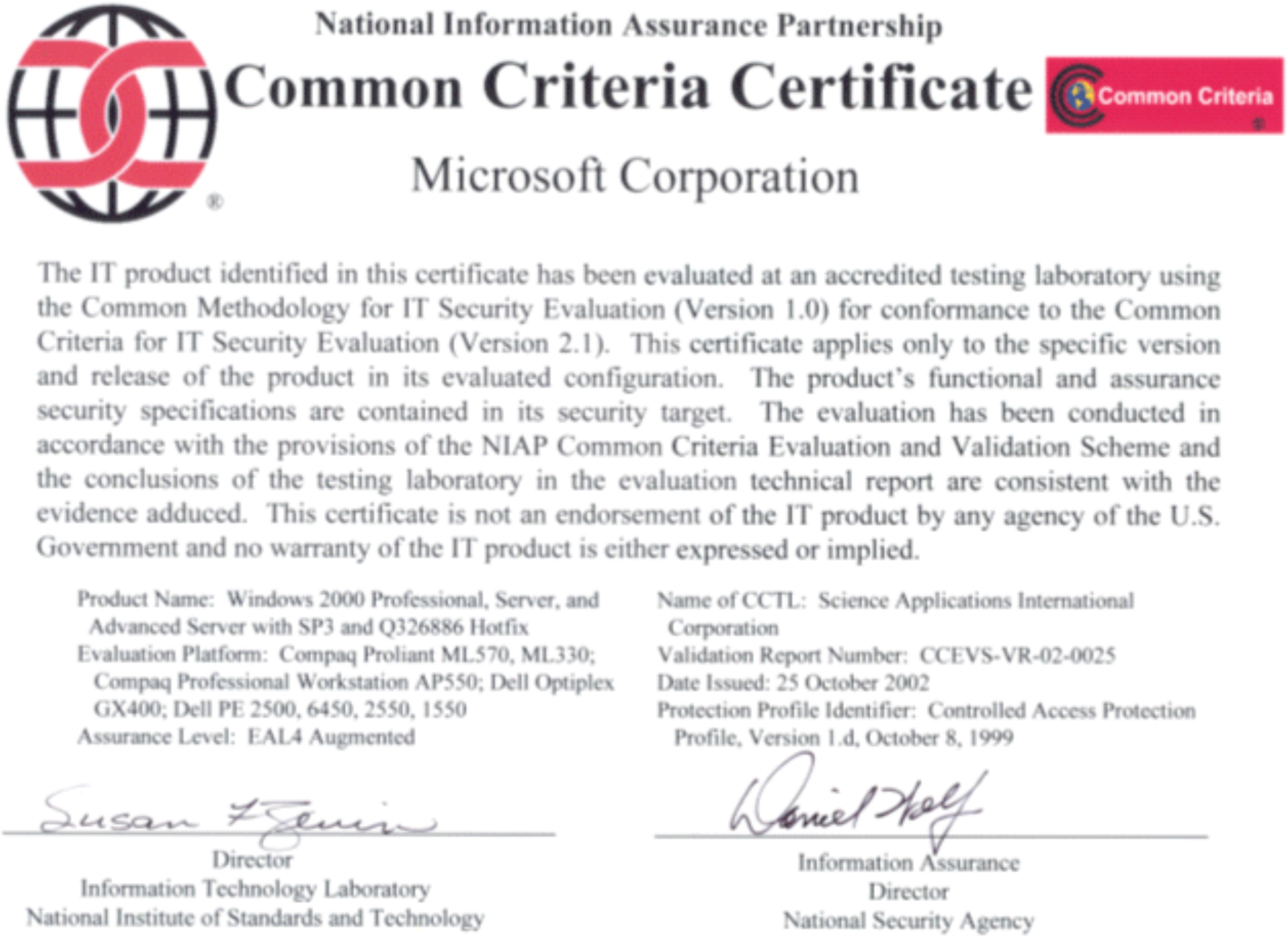
EAL 7: Formally Verified Design and Tested

- Formal presentation of functional specification
- Product or system design must be simple
- Independent confirmation of developer tests

Example: Windows 2000, EAL 4+

- Evaluation performed by SAIC
- Used “Controlled Access Protection Profile”
- Level EAL 4 + Flaw Remediation
  - “EAL 4 … represents the highest level at which products not built specifically to meet the requirements of EAL 5-7 ought to be evaluated.”

    (EAL 5-7 requires more stringent design and development procedures …)
  - Flaw Remediation
- Evaluation based on specific configurations
  - Produced configuration guide that may be useful



National Information Assurance Partnership

Common Criteria Certificate

Microsoft Corporation

The IT product identified in this certificate has been evaluated at an accredited testing laboratory using the Common Methodology for IT Security Evaluation (Version 1.0) for conformance to the Common Criteria for IT Security Evaluation (Version 2.1). This certificate applies only to the specific version and release of the product in its evaluated configuration. The product's functional and assurance security specifications are contained in its security target. The evaluation has been conducted in accordance with the provisions of the NIAP Common Criteria Evaluation and Validation Scheme and the conclusions of the testing laboratory in the evaluation technical report are consistent with the evidence adduced. This certificate is not an endorsement of the IT product by any agency of the U.S. Government and no warranty of the IT product is either expressed or implied.

Product Name: Windows 2000 Professional, Server, and Advanced Server with SP3 and Q326886 Hotfix
Evaluation Platform: Compaq Proliant ML570, ML330; Compaq Professional Workstation AP550; Dell Optiplex GX400; Dell PE 2500, 6450, 2550, 1550
Assurance Level: EAL4 Augmented

Name of CCTL: Science Applications International Corporation
Validation Report Number: CCEVS-VR-02-0025
Date Issued: 25 October 2002
Protection Profile Identifier: Controlled Access Protection Profile, Version 1.d, October 8, 1999

Susan F Zevin

Director
Information Technology Laboratory
National Institute of Standards and Technology

Daniel Wolf

Information Assurance
Director
National Security Agency