Authorization

Authentication vs Authorization

- Authentication — Are you who you say you are?
  - Restrictions on who (or what) can access system
- **Authorization** — Are you allowed to do that?
  - Restrictions on actions of authenticated users
- Authorization is a form of **access control**

Access control policy models

- Access control policy models: how access control policies are configured and managed system-wide.
  - **Discretionary** Access Control (DAC)
  - **Mandatory** Access Control (MAC)
  - **Role-Based** Access Control (RBAC)

Discretionary Access Control (DAC)

- Definition: An individual user can set an access control mechanism to allow or deny access to an object
- Relies on the object owner to control access
- DAC is widely implemented in most operating systems, and we are quite familiar with it
  - In UNIX file protection, the owner of a file controls read, write and execute privilege
- Strength of DAC: **Flexibility**: a key reason why it is widely known and implemented in main-stream operating systems

Mandatory Access Control (MAC)

- Definition: **A system-wide policy decrees who is allowed to have access; individual user cannot alter that access.**
- Relies on the system to control access
- Traditional MAC mechanisms have been tightly coupled to a few security models
- Recently, systems supporting flexible security models start to appear (e.g., SELinux, Trusted Solaris, TrustedBSD, etc.)

Role-Based Access Control

- Users are associated with roles; roles with permissions
- A user has a permission only if the user has an authorized role which is associated with that permission

Access control mechanism

- Access control mechanism: how access control is implemented in systems
  - Access control matrices
  - Access control list
  - Capabilities
  - …

Lampson's Access Control Matrix

- **Subjects** (users) index the rows
- **Objects** (resources) index the columns

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	---	---
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

Are You Allowed to Do That?

- **Access control matrix** has **all** relevant info
- Could be 1000's of users, 1000's of resources
- Then matrix with 1,000,000's of entries
- How to manage such a large matrix?
- Need to check this matrix before access to any resource is allowed
- How to make this efficient?

Access Control Lists (ACLs)

- ACL: store access control matrix by **column**
- Example: ACL for **insurance data** is in **blue**

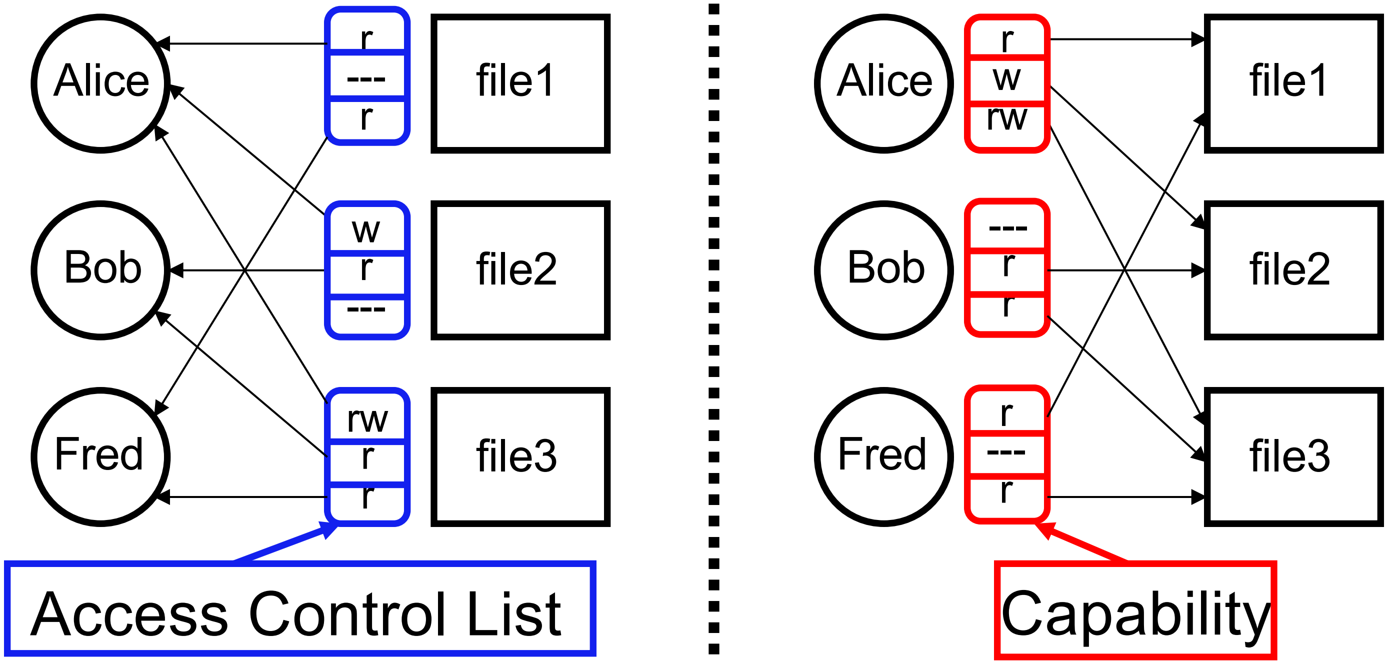
	OS	Accounting program	Accounting data	**Insurance data**	Payroll data
Bob	rx	rx	r	---	---
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

Capabilities (or C-Lists)

- Store access control matrix by **row**
- Example: Capability for **Alice** is in **red**

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	---	---
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

ACLs vs Capabilities



- Note that arrows point in opposite directions…

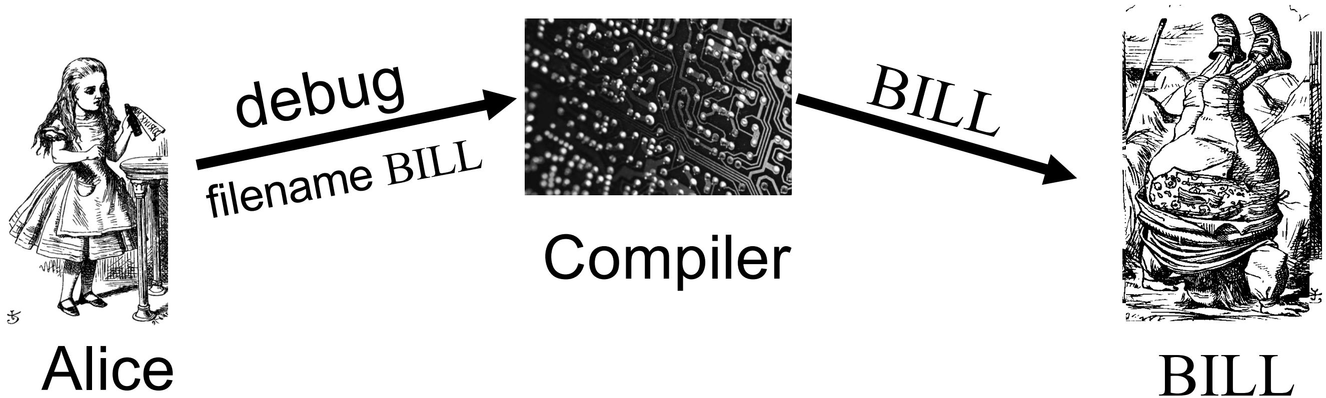
Confused Deputy Problem

- Two resources
  - Compiler and BILL file (billing info)
- Compiler can write file BILL
- Alice can invoke compiler with a debug filename
- Alice not allowed to write to BILL

- Access control matrix

	Compiler	BILL
Alice	x	---
Compiler	rx	rw

ACL's and Confused Deputy



- Compiler is **deputy** acting on behalf of Alice
- Compiler is **confused**
  - Alice is not allowed to write BILL
- Compiler has confused its rights with Alice's

Confused Deputy

- Compiler acting for Alice is confused
- With ACLs, difficult to avoid this problem
- With Capabilities, easier to prevent problem
  - Capabilities make it easy to **delegate** authority
  - In the previous example, Alice can delegate her authority **over** the BILL file to the compiler
    - Give her C-list to compiler
    - Compiler use Alice C-list to check privilege

Summary: ACLs vs Capabilities

- ACLs
  - Good when users manage their own files
  - Protection is data-oriented
  - Easy to change rights to a resource

- Capabilities
  - “A capability is a token, ticket, or key that gives the possessor permission to access an entity or object in a computer system” – Dennis and Van Horn in 1966
  - Easy to delegate---avoid the confused deputy
  - Easy to add/delete users
  - More difficult to implement: create, delegate, revoke, delete, enable, disable, …

Case study: Unix-like Systems

- Is access to the file system in Linux based on ACLs or capabilities?
- Run command: `getfacl test.dat`

```
# file: test.dat
# owner: xzhang
# group: xzhang
user::rw-
group::rw-
other::r--
```

Access control based on three classes: owner, group, other

- Or simply: ls –l test.dat

110110100 or 664

```
-rw-rw-r-- 1 xzhang xzhang 20 Sep 12 2013 test.dat
```

Program executed by a normal user

/* /etc/passwd file has a permission 644 */

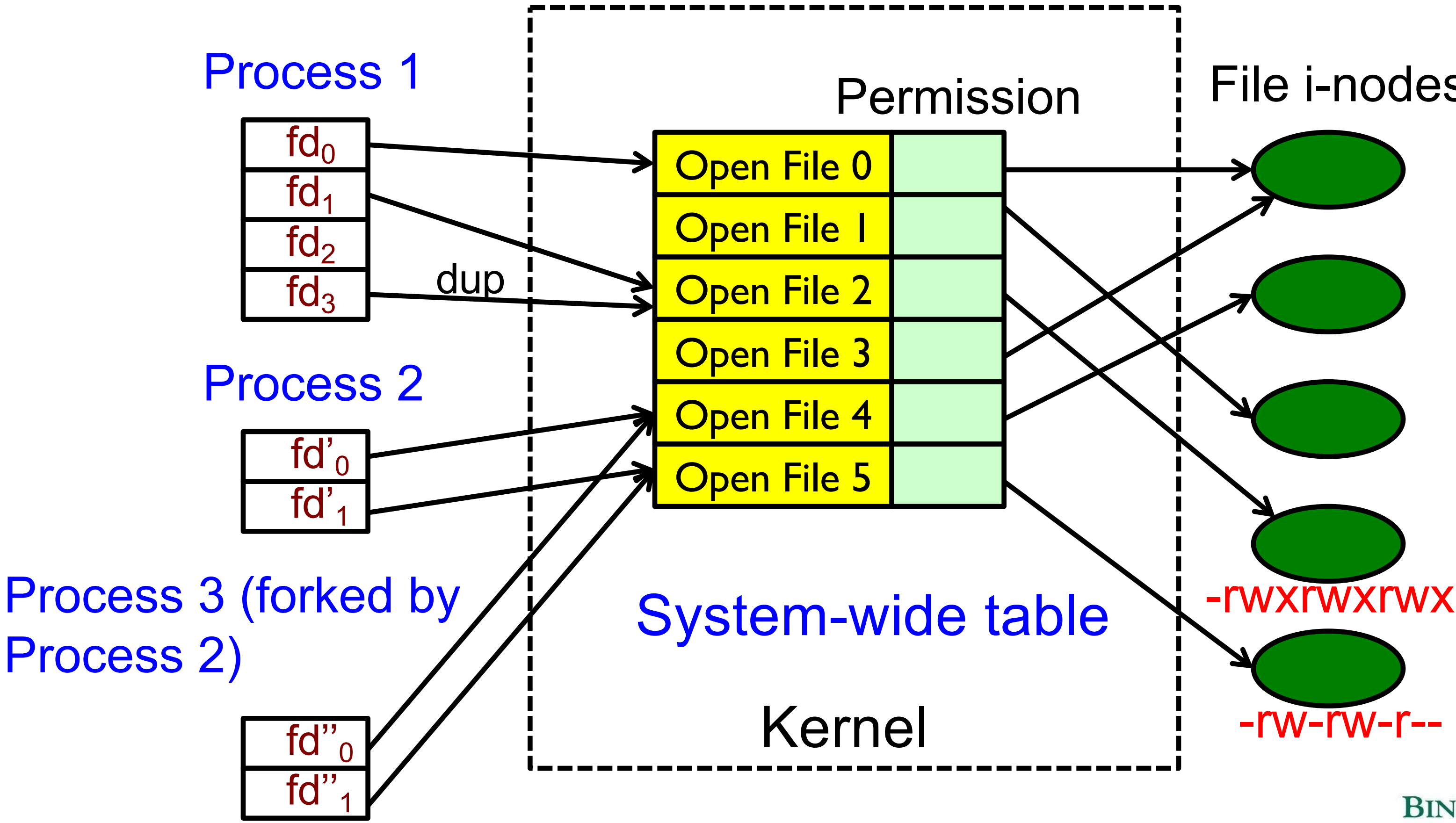
- 1: `f = open("/etc/passwd", "r");`
- 2: `read(f, buf, 10);` → Succeed
- 3: `write(f, buf, 10);` → Fail

/* Before the following statement is executed, the root modifies the permission on /etc/passwd to 600, i.e., normal users cannot read this file any more. */

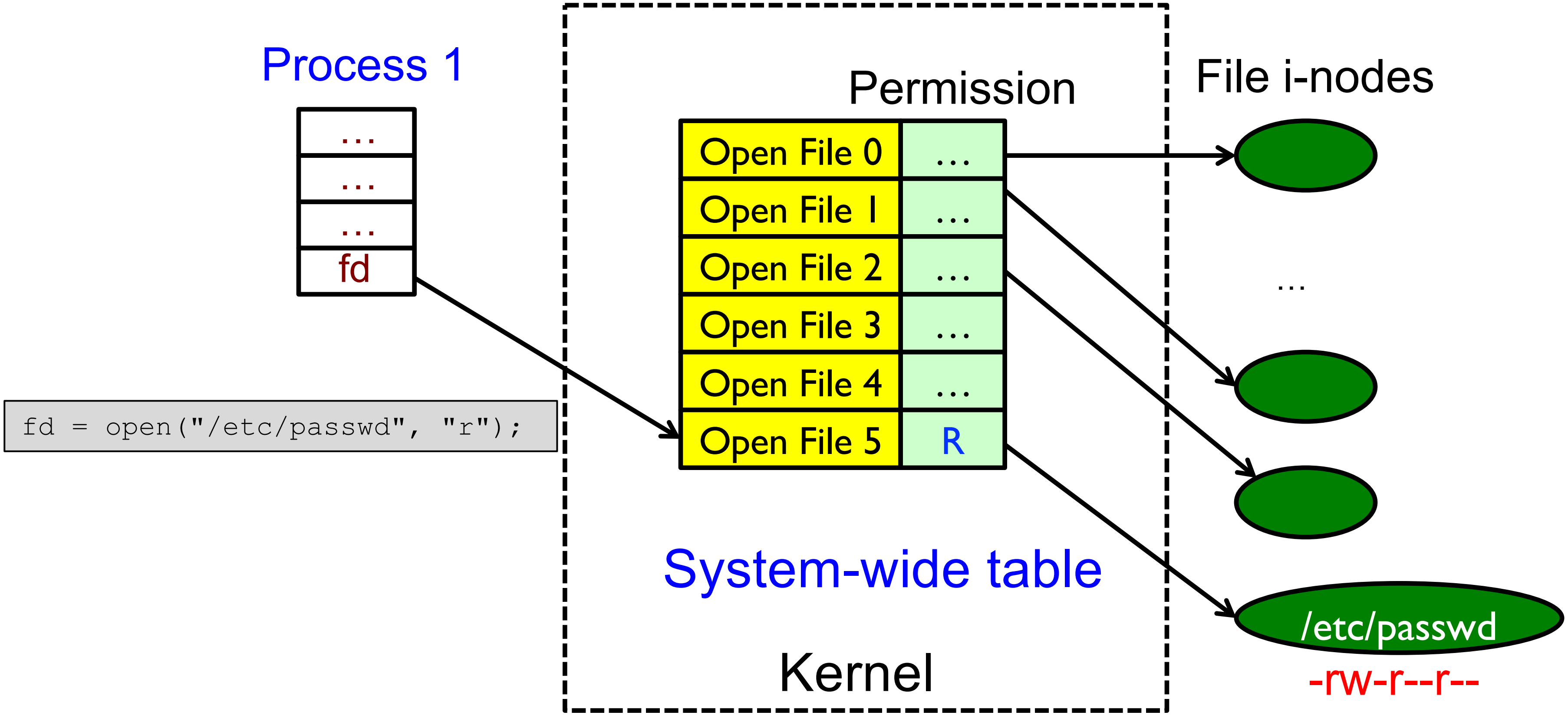
- 4: `read(f, buf, 10);` → Succeed, or fail?

It will succeed. Why?

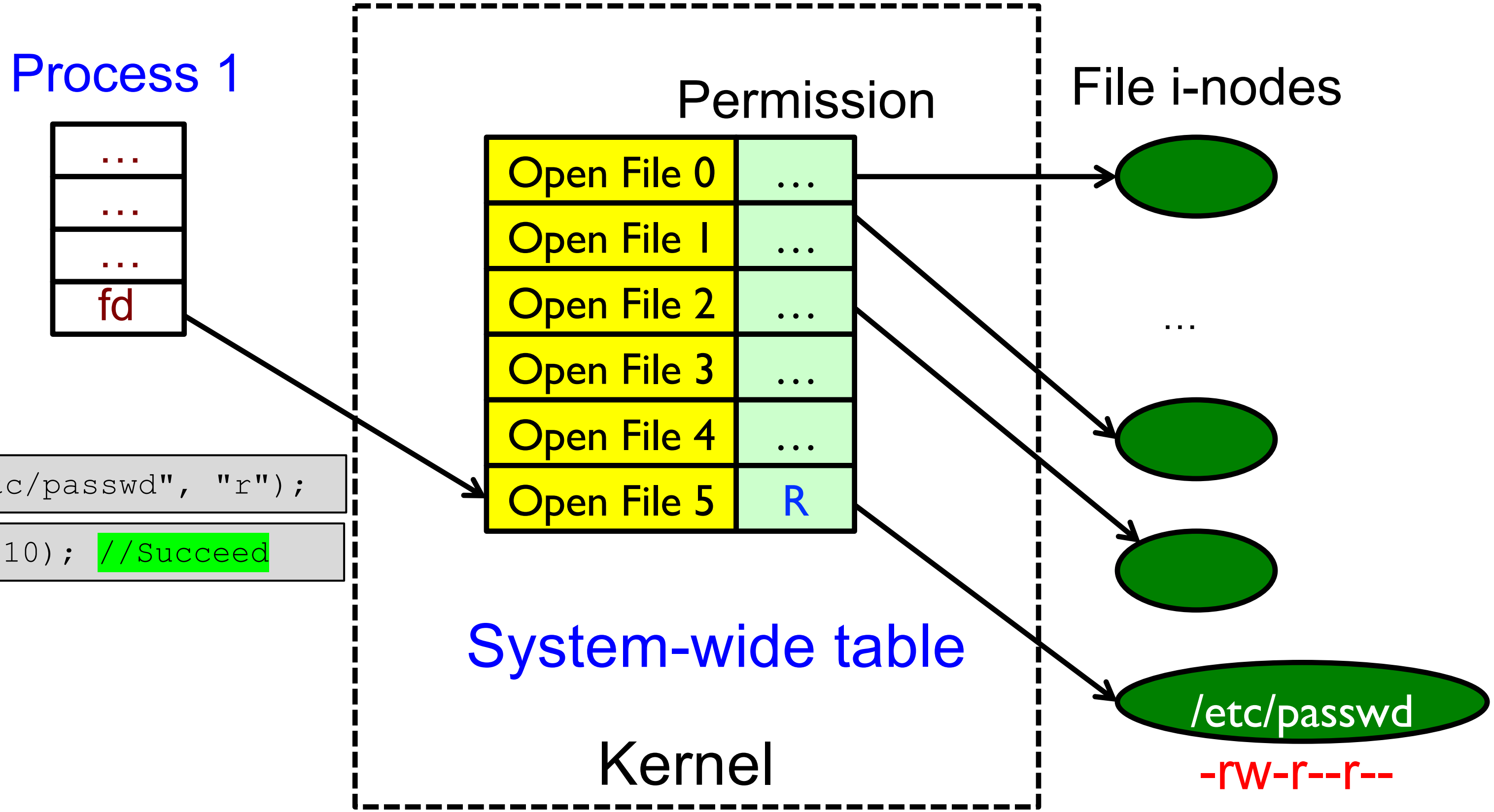
Illustration



Illustration

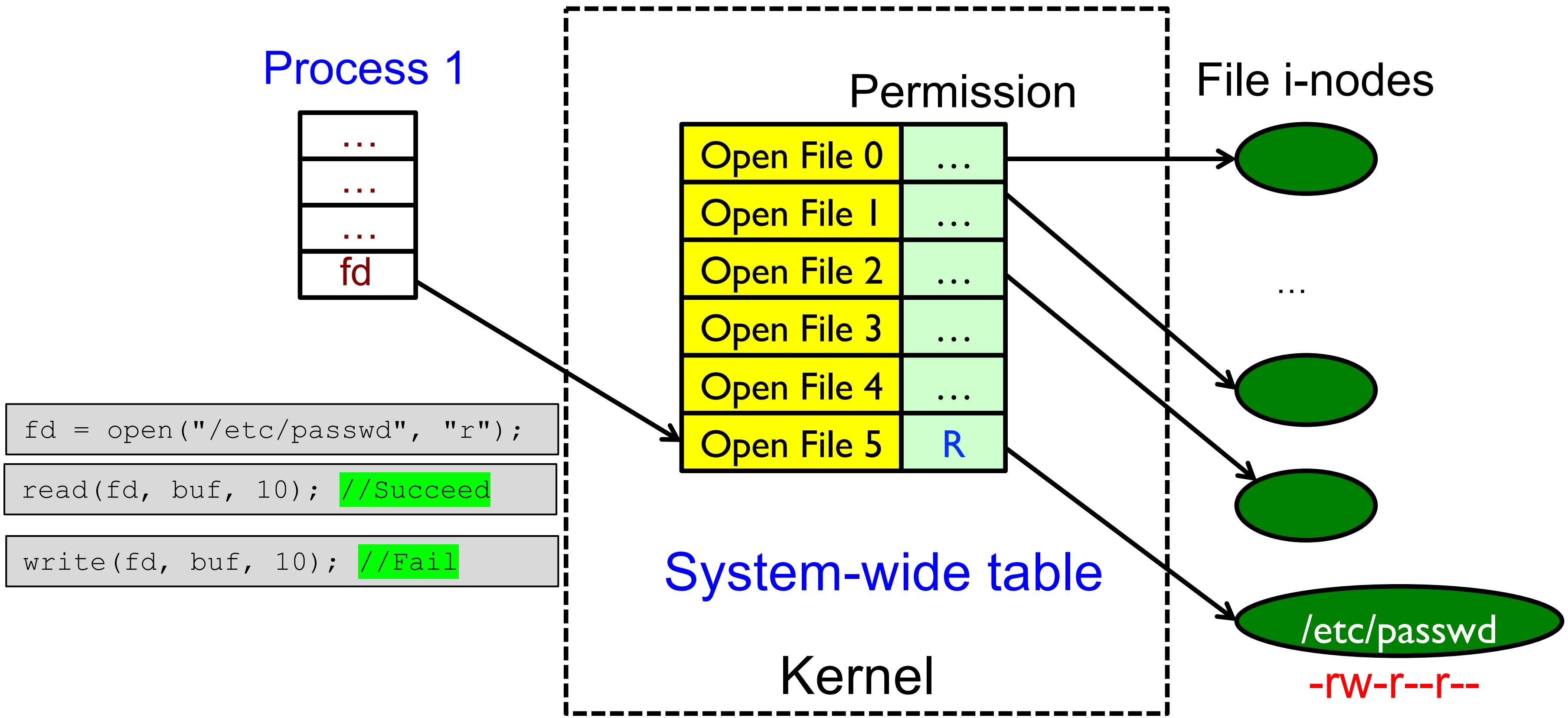


Illustration

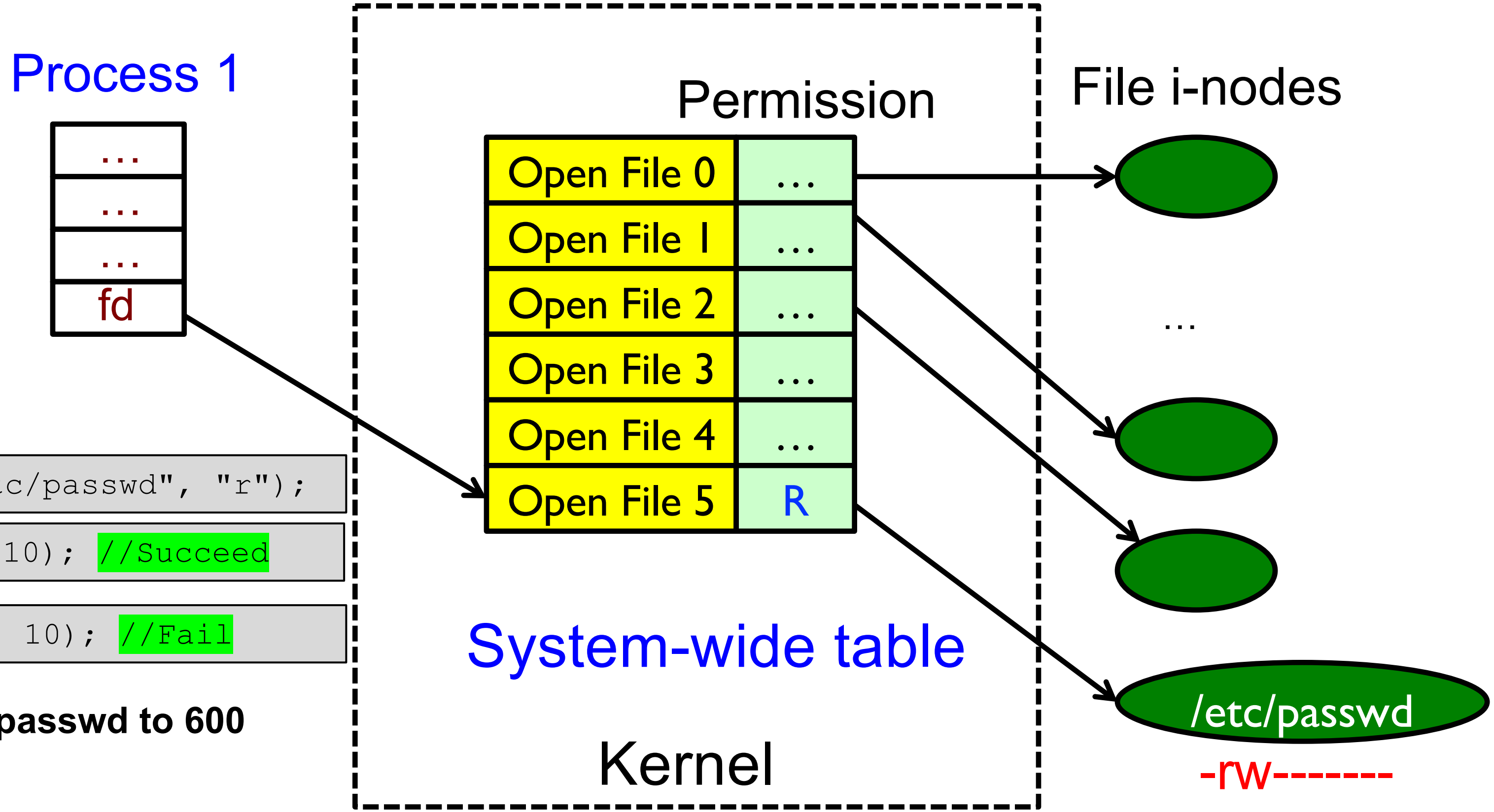



```
fd = open("/etc/passwd", "r");
read(fd, buf, 10); //Succeed
```

Illustration



Illustration



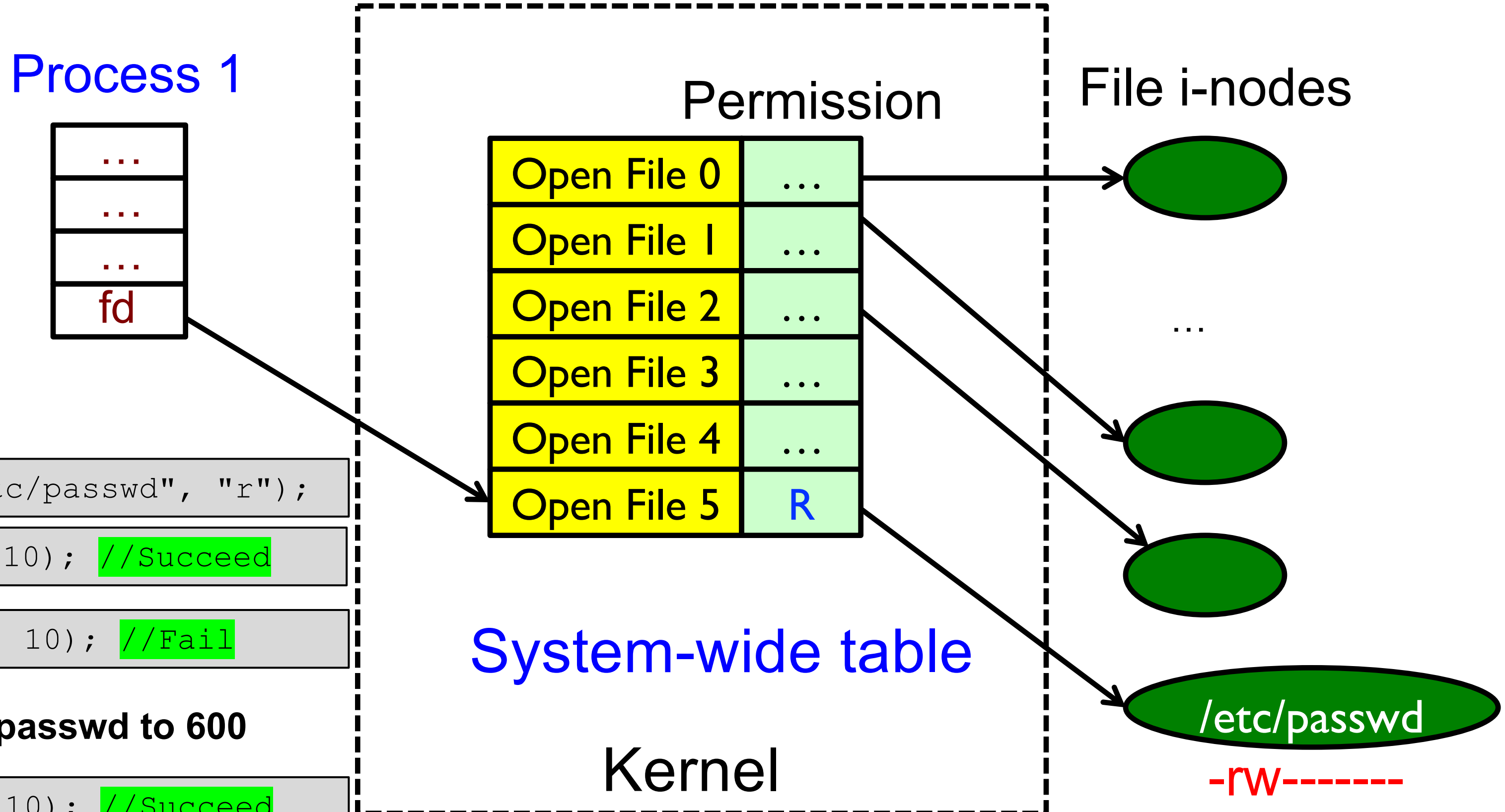

```
fd = open("/etc/passwd", "r");
```

```
read(fd, buf, 10); //Succeed
```

```
write(fd, buf, 10); //Fail
```

Change /etc/passwd to 600

Illustration




```
fd = open("/etc/passwd", "r");
read(fd, buf, 10); //Succeed
write(fd, buf, 10); //Fail
```

Change /etc/passwd to 600

```
read(fd, buf, 10); //Succeed
```

Multilevel Security (MLS) Models

Access Control Matrix

Object

Subject	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	---	---
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw

Subject

Flat!

Why hierarchical?

- Subjects may need to have different levels of access rights
  - **National lab**: users < group managers < division leaders < principal associate directors < director
  - **University**: students < professors < department chair < dean < president
  - …
- Objects may have different sensitive levels
  - Shared with general public
  - Shared by all employees
  - Shared by all managers
  - Shared by upper managers
  - …

Multi-level security?

Classifications and Clearances

- **Classifications** apply to **objects**
- **Clearances** apply to **subjects**
- US Department of Defense (DoD) uses 4 levels for classification:

  TOP SECRET
  SECRET
  CONFIDENTIAL
  UNCLASSIFIED
- US Department of Energy clearance level:

  Q: top secret

  L: secret

  U: unclassified
- A subject with a SECRET clearance is allowed access to objects classified SECRET or lower but not to objects classified TOP SECRET

Subjects and Objects

- Let O be an **object**, S a **subject**
  - O has a classification
  - S has a clearance
  - Security **level** denoted $L(O)$ and $L(S)$
- For DoD levels, we have

  TOP SECRET > SECRET > CONFIDENTIAL > UNCLASSIFIED

Multilevel Security (MLS)

- MLS needed when subjects/objects at different levels use/on **same system**
- MLS is a form of **Access Control**
  - **Subjects can only access objects they have the necessary clearance**
- Military and government interest in MLS for many decades
  - Lots of research into MLS
  - Strengths and weaknesses of MLS well understood (almost entirely theoretical)
  - Many possible uses of MLS outside military

MLS Applications

- Classified government/military systems
- **Business example:** info restricted to
  - Senior management only, all management, everyone in company, or general public
- Confidential medical info, databases, etc.
- Usually, MLS not a viable technical system
  - More of a legal device than technical system

MLS Security Models

- MLS models explain **what** needs to be done
- Models **do not** tell you **how** to implement
- Models are descriptive, not prescriptive
  - That is, high level description, not an algorithm
- There are many MLS models
- We'll discuss simplest MLS model
  - Other models are more realistic
  - Other models also more complex, more difficult to enforce, harder to verify, etc.

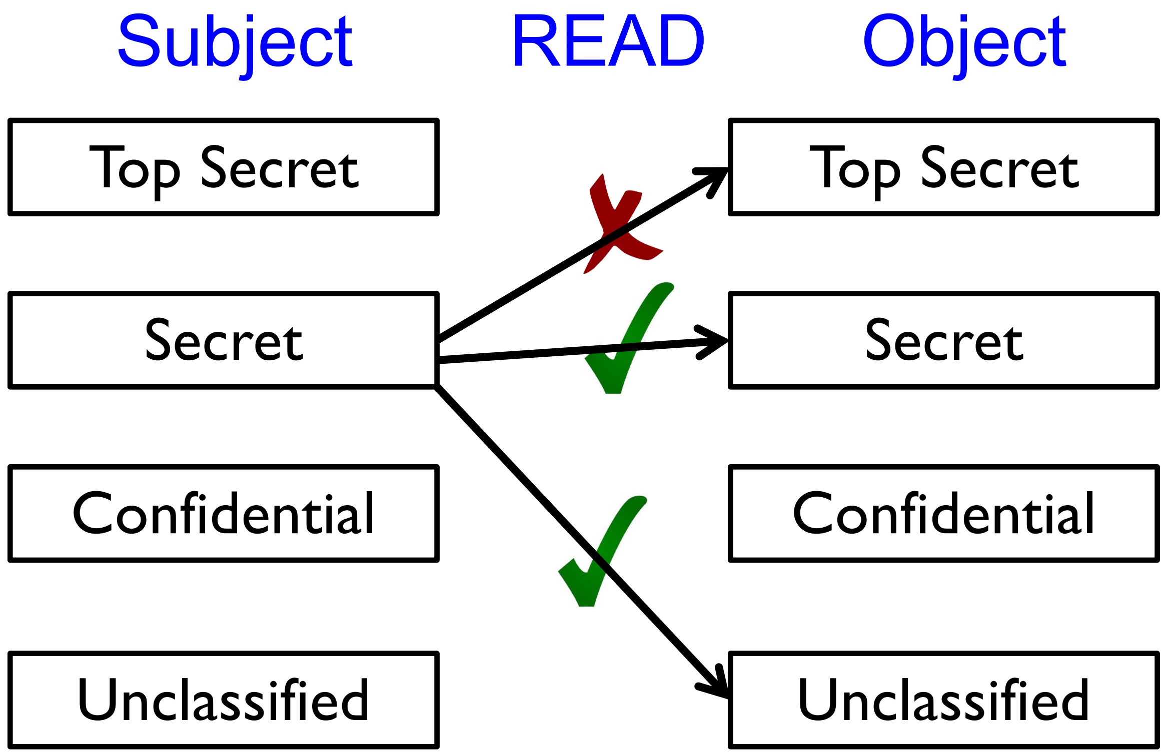
Bell-LaPadula

- BLP security model designed to express essential requirements for MLS
- BLP deals with **confidentiality**
  - To prevent unauthorized reading
- Recall that O is an object, S a subject
  - Object O has a classification
  - Subject S has a clearance
  - Security level denoted L(O) and L(S)
- Consists of **simple security condition** and ***-property (star property)**

Simple Security Condition

No read up!

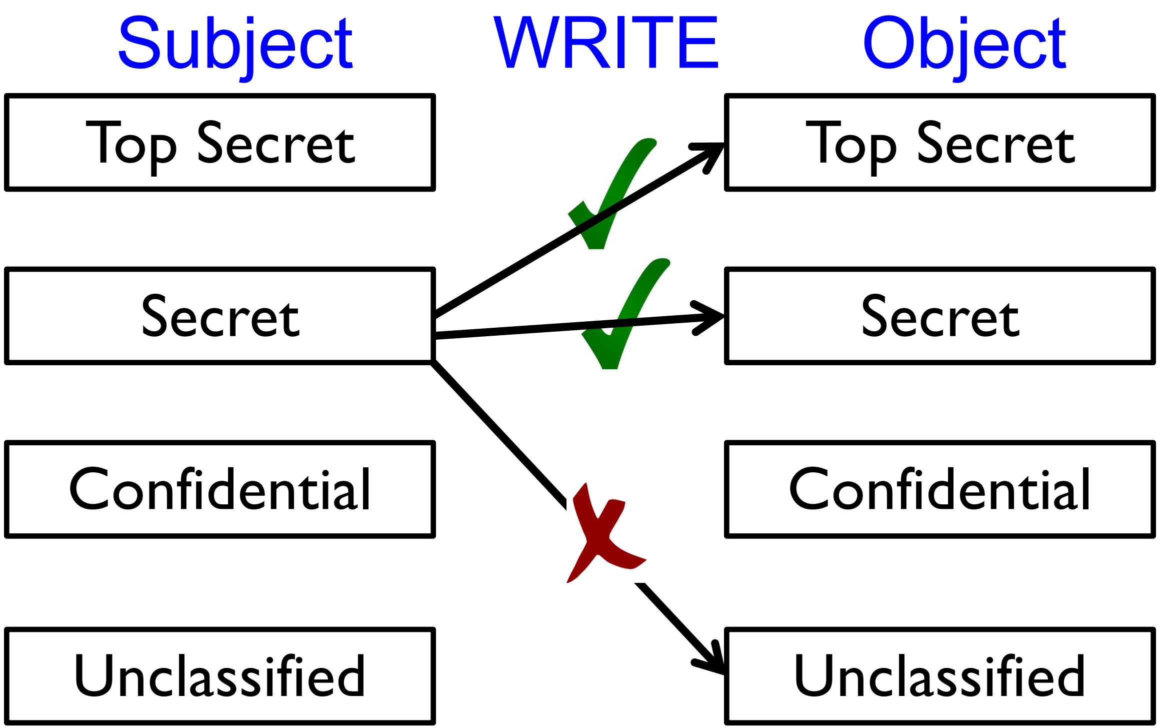
- **Simple Security Condition**: S can read O if and only if $L(O) \leq L(S)$



*-Property (Star Property)

No write down!

- ***-Property** (**Star Property**): S can write O if and only if L(S) ≤ L(O)



Bell-LaPadula

- BLP consists of

  Simple Security Condition: S can read O if and only if $L(O) \leq L(S)$

  ***-Property** (**Star Property**): S can write O if and only if $L(S) \leq L(O)$

- **No read up, no write down**

McLean's Criticisms of BLP

- McLean: BLP is "so trivial that it is hard to imagine a realistic security model for which it does not hold"
- McLean's "**system Z**" allowed administrator to <u>reclassify</u> object, then "write down"
  - For instance, a SECRET subject is not allowed to write on an UNCLASSIFIED object; but what if the classification level of the object is changed to, say, TOP SECRET?
  - Violates spirit of BLP, but **not** expressly forbidden in statement of BLP

B and LP's Response

- BLP enhanced with **tranquility property**
  - Strong tranquility: security labels never change
  - Weak tranquility: security label can only change if it does not violate "established security policy"

Strong tranquility impractical

- Strong tranquility impractical in real world
  - DoD constantly declassifies documents
  - Difficult to enforce "**least privilege**"
    - **Example**: a TOP SECRET clearance visits an UNCLASSIFIED webpage
    - **Solution**: give users lowest privilege for current work, and upgrade as needed
  - **High watermark principle**: any object less than the user's security level can be opened, but the object is **relabeled** to reflect the highest security level currently open.
    - **Example**: If user A is writing a **CONFIDENTIAL** document, and checks the **unclassified** dictionary, the dictionary becomes **CONFIDENTIAL**.

Weak tranquility

- Weak tranquility: security label can only change if it does not violate "established security policy"
- Weak tranquility can defeat system Z
- Weak tranquility allows for **least privilege**
- But the property is vague
  - Change of security levels doesn't violate "established security policy"…

BLP: The Bottom Line

- BLP is simple, probably too simple
- BLP is one of the few security models that can be used to prove things about systems
- BLP has inspired other security models
  - Most other models try to be more realistic
  - Other security models are more complex
  - Models difficult to analyze, apply in practice

Biba's Model

- BLP for confidentiality, Biba for **integrity**
  - Biba is to prevent unauthorized writing
- Biba is (in a sense) the dual of BLP
- Integrity model
  - Suppose you trust the integrity of **O1** but not **O2**
  - If object **O** includes **O1** and **O2** then you cannot trust the integrity of **O**
- Integrity level of O is **minimum** of the integrity of any object in O
- **Low water mark** principle for integrity

Biba

- Let $I(O)$ denote the **integrity** of object O and $I(S)$ denote the integrity of subject S

- Biba can be stated as

  Write Access Rule: S can write O if and only if $I(O) \leq I(S)$

  (if S writes O, the integrity of $O \leq$ that of S)

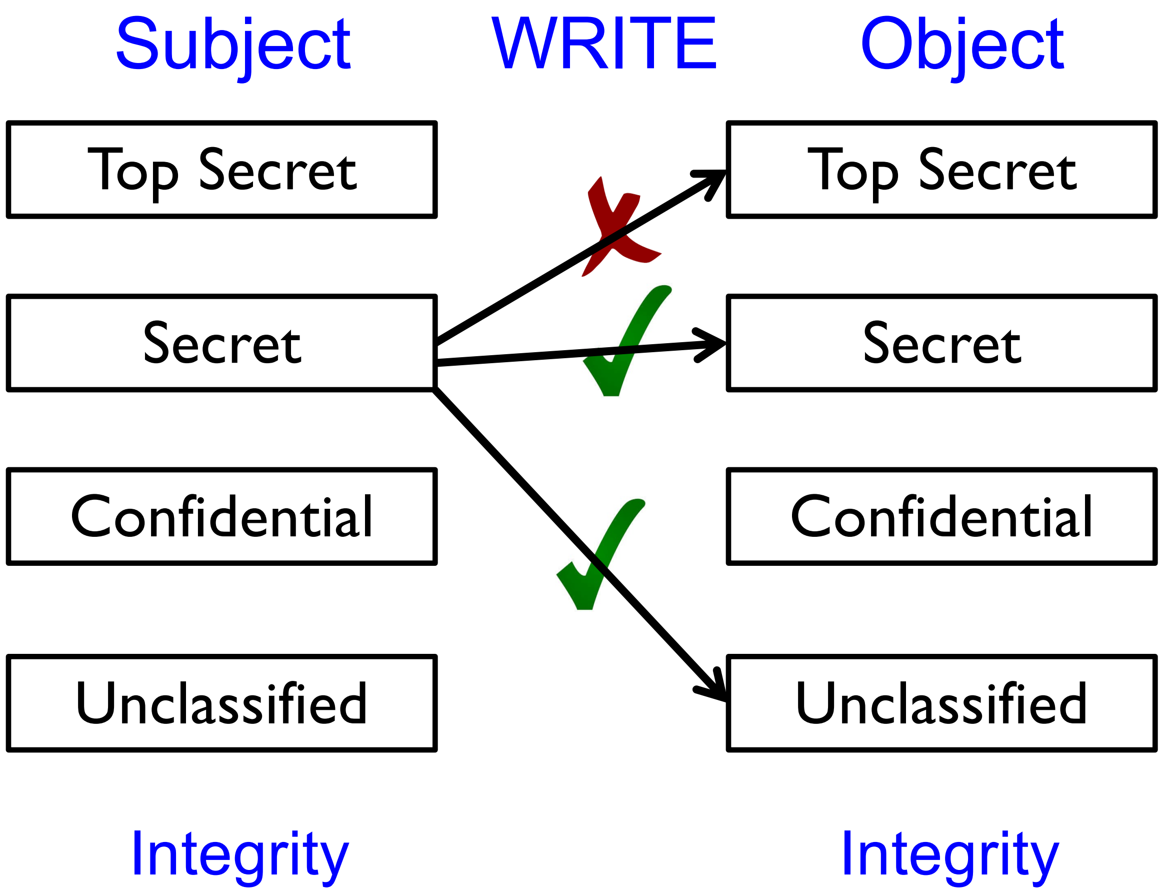
  Biba's Model: S can read O if and only if $I(S) \leq I(O)$

  (if S reads O, the integrity of $S \leq$ that of O)

- Often, replace Biba's Model with

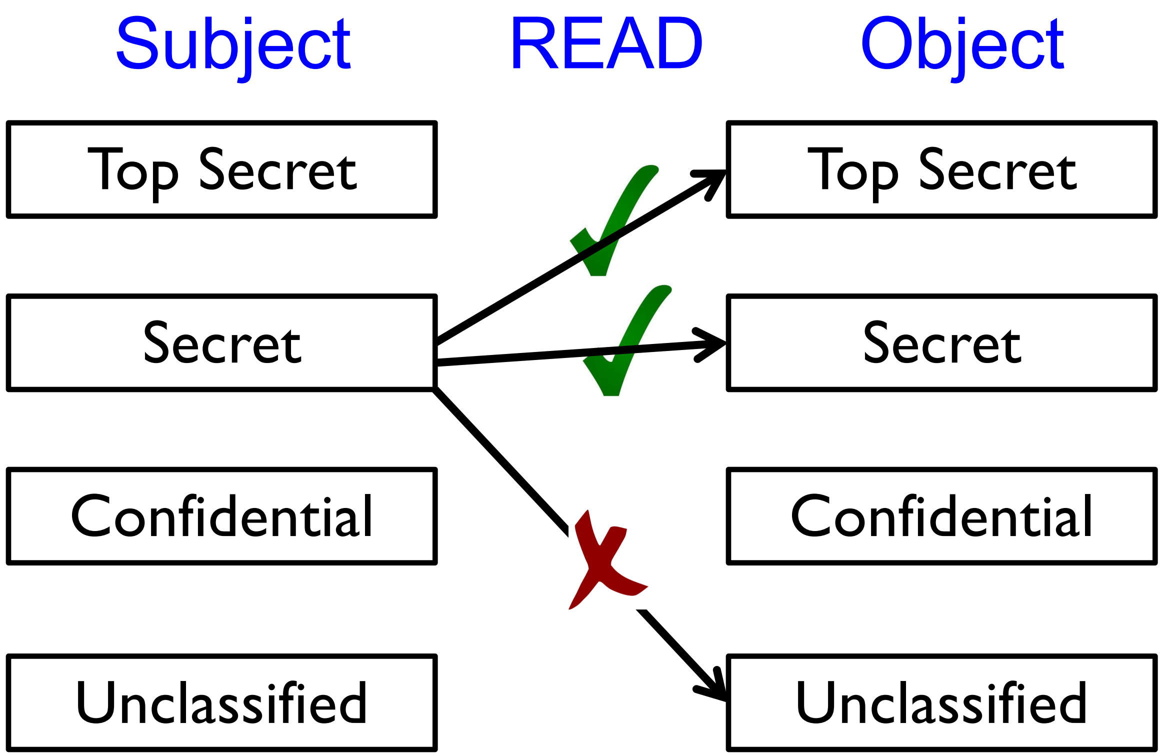
  Low Water Mark Policy: If S reads O, then $I(S) = \min(I(S), I(O))$

Write access rule: S can write O if and only if I(O) ≤ I(S)

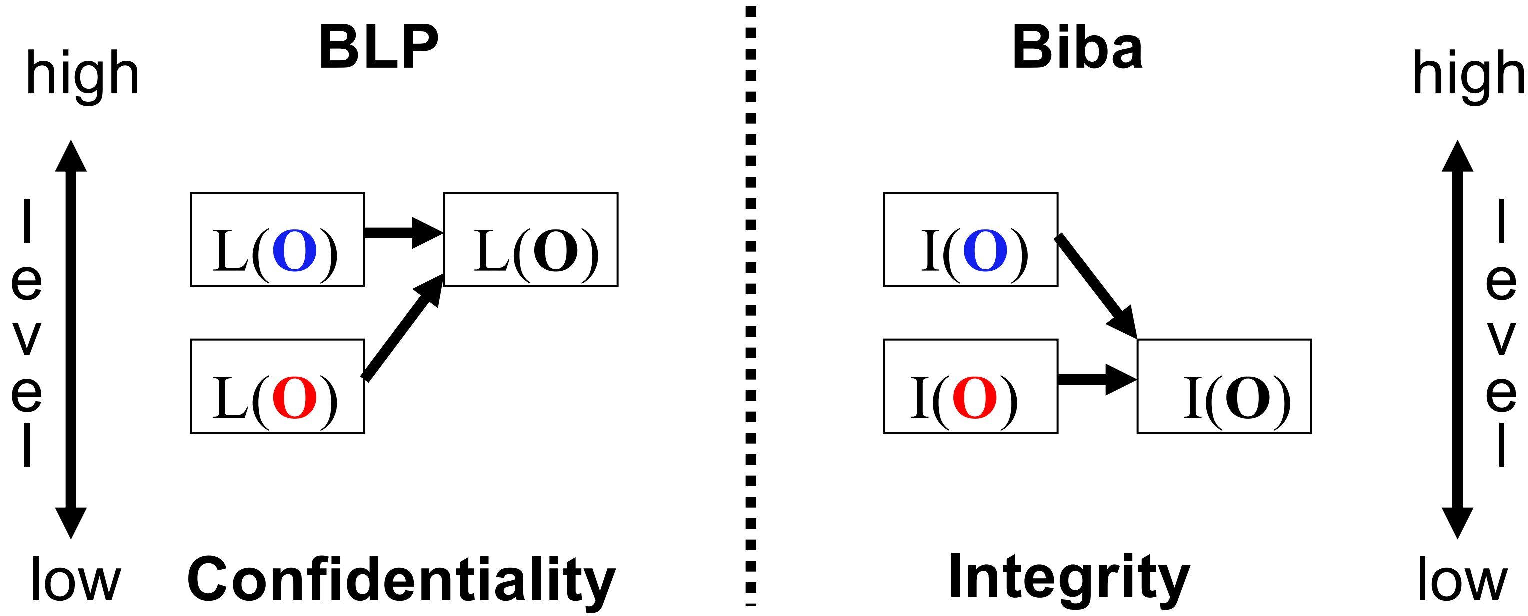


BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Biba's model: S can read O if and only if $I(S) \leq I(O)$



BLP vs Biba



High watermark

Low watermark

Compartments

Compartments

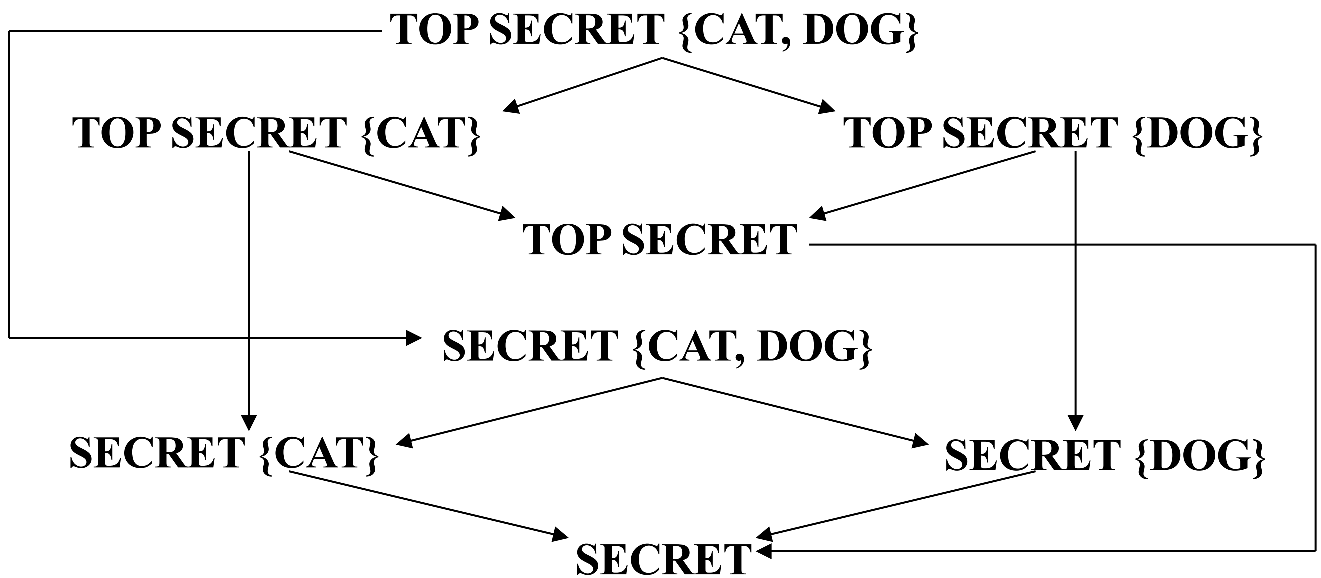
- Multilevel Security (MLS) enforces access control **up and down**
- Simple hierarchy of security labels is generally *not* flexible enough
- Compartments enforces restrictions **across different domains**
- Suppose **TOP SECRET** divided into **TOP SECRET {CAT}** and **TOP SECRET {DOG}**
- Both are **TOP SECRET** but information flow restricted across the **TOP SECRET** level

Compartments – Why?

- Why compartments?
  - Why not create a new classification level?
- May not want either of
  - **TOP SECRET {CAT} ≥ TOP SECRET {DOG}**
  - **TOP SECRET {DOG} ≥ TOP SECRET {CAT}**
- Compartments designed to enforce the **need to know** principle
  - Regardless of clearance, you only have access to info that **you need to know** to do your job

Compartments

- Arrows indicate "≥" relationship



- Not all classifications are comparable, e.g., **TOP SECRET {CAT}** vs **SECRET {CAT, DOG}**

Partial ordering

Covert Channel

Covert Channel

- Security policies (e.g., MLS and compartments) are designed to restrict legitimate channels of communication
- May be other ways for information to flow
- For example, resources shared at different levels of MLS could be used to “signal” information
- **Covert channel**: a communication path not intended as such by system’s designers

Covert Channel Example

- Alice has **TOP SECRET** clearance, Bob has **CONFIDENTIAL** clearance
- Suppose the file space shared by all users
- Alice creates file FileXYzW to signal “1” to Bob, and removes file to signal “0”
- Once per minute Bob lists the files
  - If file FileXYzW does not exist, Alice sent 0
  - If file FileXYzW exists, Alice sent 1
- Alice can leak **TOP SECRET** info to Bob!

Covert Channel Example

Alice:	Create file	Delete file	Create file		Delete file
Bob:	Check file	Check file	Check file	Check file	Check file
Data:	1	0	1	1	0
Time:					

BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK

Covert Channel - requirement

- Other possible covert channels?
  - Print queue
  - ACK messages
  - Network traffic, etc.
- When does covert channel exist?
  1. Sender and receiver have a **shared** resource
  2. Sender able to vary some property of resource that receiver can observe
  3. "Communication" between sender and receiver can be synchronized

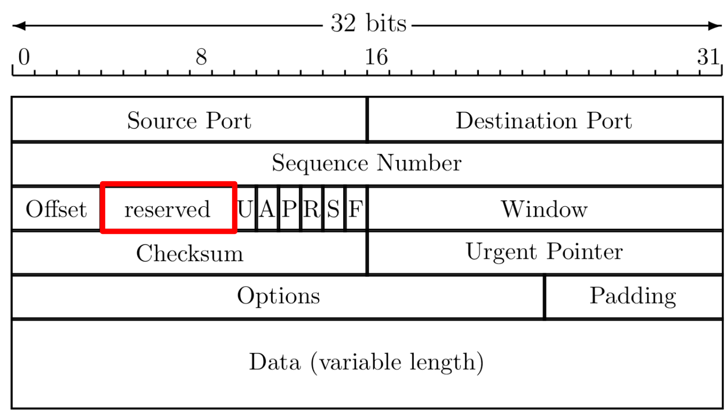
Covert Channel - Mitigation

- So, covert channels are everywhere
- “Easy” to eliminate covert channels:
  - Eliminate all shared resources…
  - …and all communication
- Virtually impossible to eliminate covert channels in any useful system
  - DoD guidelines: **reduce covert channel capacity** to no more than 1 bit/second
  - Implication? DoD has given up on *eliminating* covert channels!

Covert Channel – Mitigation Example

- Consider 100MB **TOP SECRET** file
  - Plaintext stored in **TOP SECRET** location
  - Ciphertext (encrypted with AES using 256-bit key) stored in **UNCLASSIFIED** location
- Suppose we reduce covert channel capacity to 1 bit per second
- It would take more than **25 years** to leak entire document through a covert channel
- But it would take less than **5 minutes** to leak 256-bit AES key through covert channel!

Real-World Covert Channel



- Hide data in TCP header “reserved” field
- Or use covert_TCP, tool to hide data in
  - Sequence number
  - ACK number

Real-World Covert Channel – sequence number

- Hide data in TCP sequence numbers
- Tool: covert_TCP
- Sequence number X contains covert info