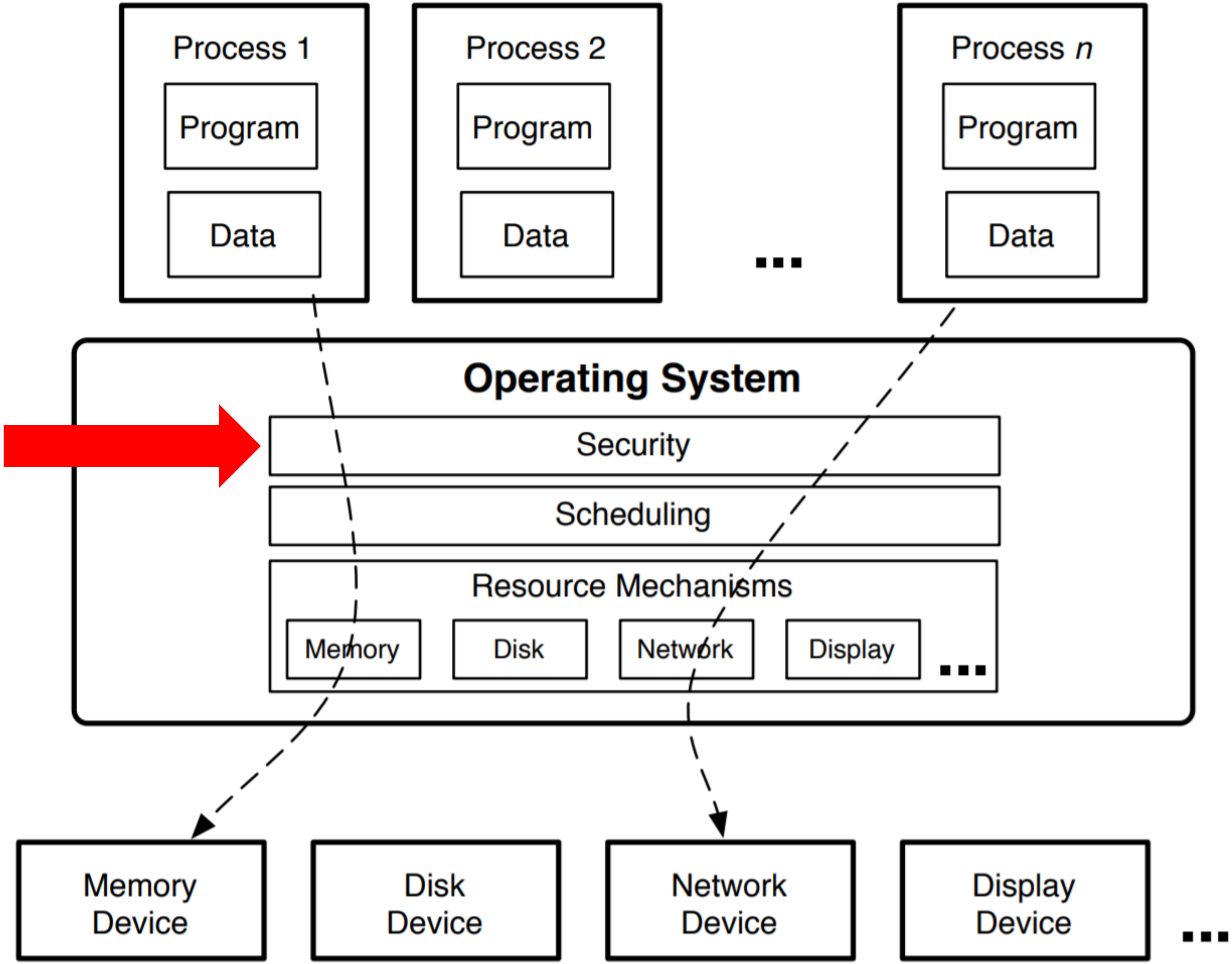
CS 642
OS SECURITY

Earlence Fernandes

(Some slides are graciously borrowed from Ristenpart, Everspaugh)


Process 1
Program
Data
Process 2
Program
Data
...
Process *n*
Program
Data
Operating System
Security
Scheduling
Resource Mechanisms
Memory
Disk
Network
Display
...
Memory Device
Disk Device
Network Device
Display Device
...

Have you used UNIX in the past 30 seconds?

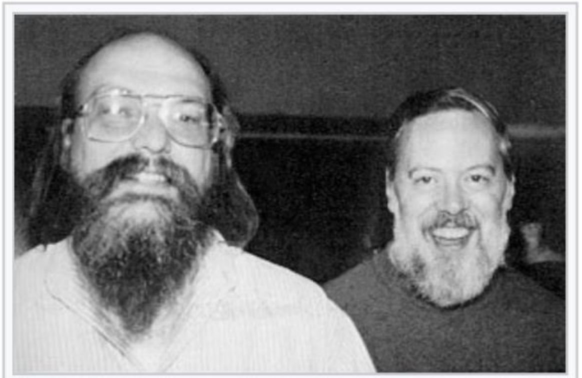
poll

Family Tree

multics

1960s MIT, AT&T, Bell Labs, GE

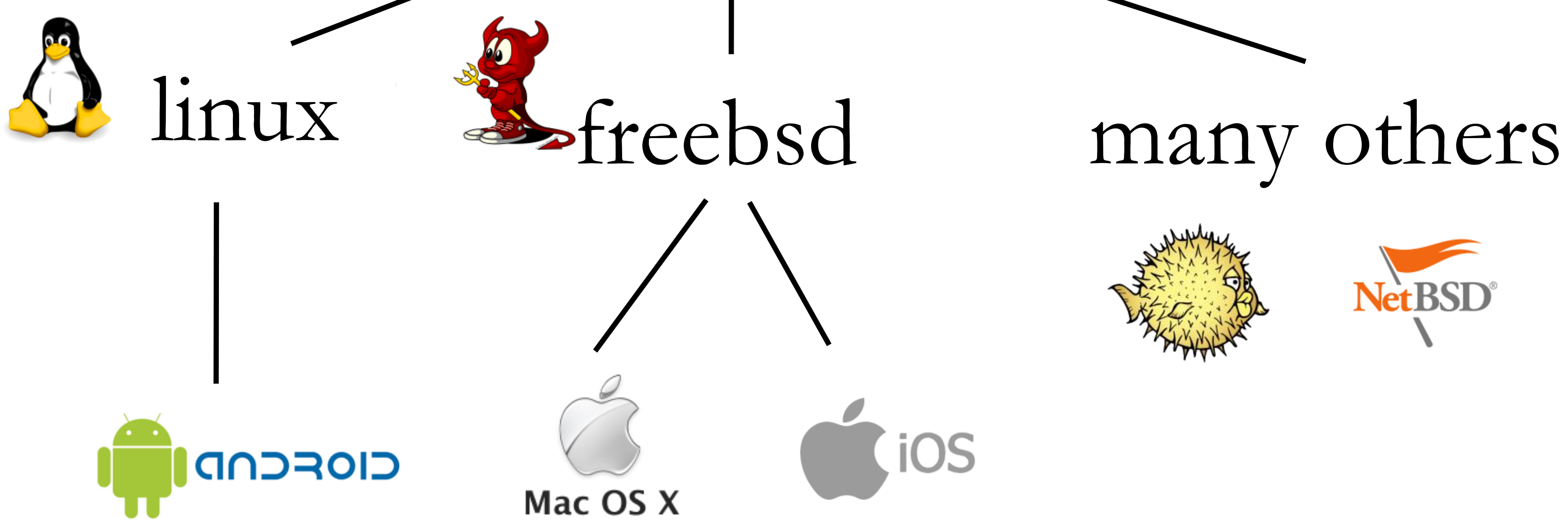
~ 100 installations



Ken Thompson and Dennis Ritchie

1970s
bell labs

unix

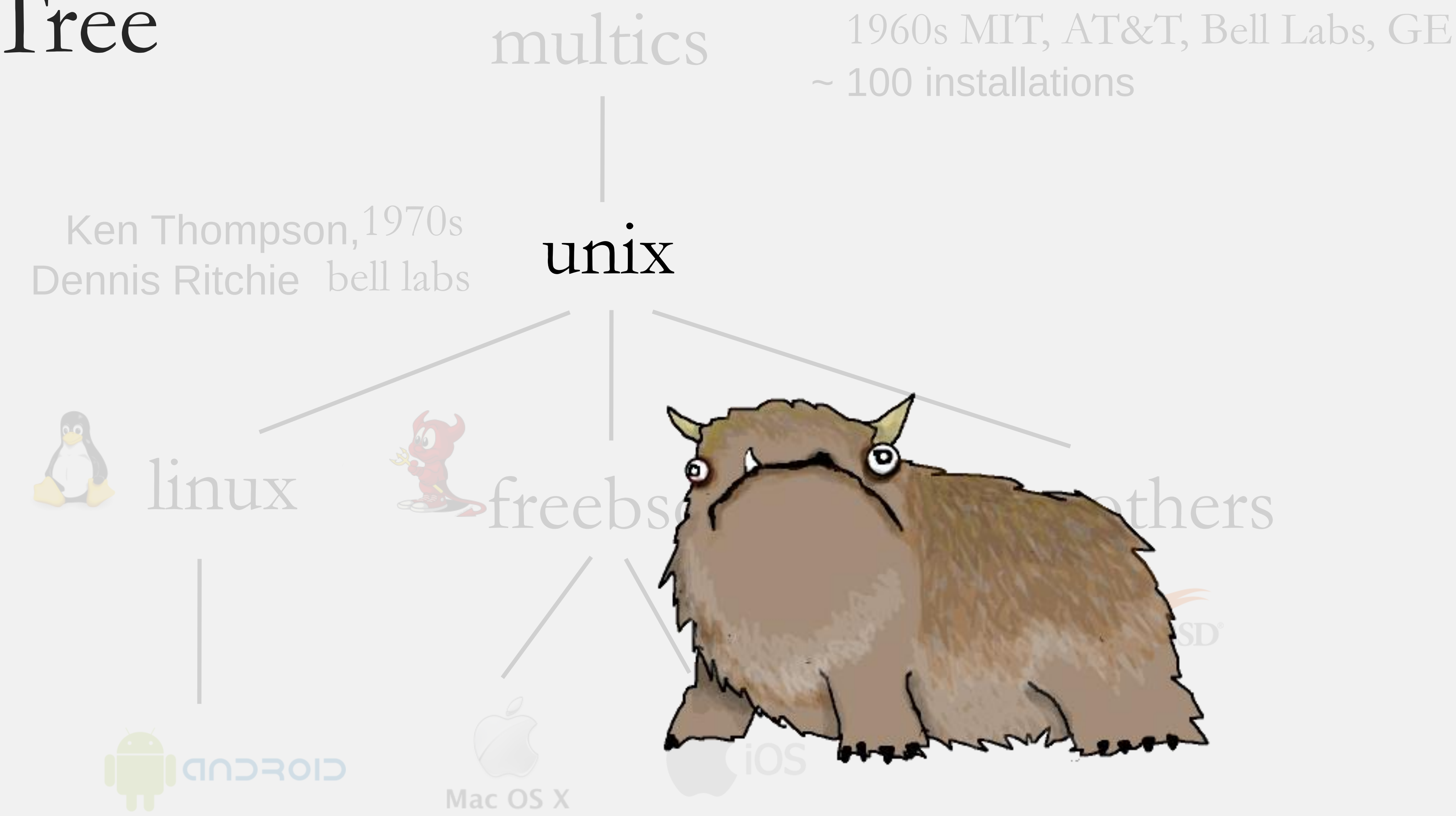


linux

freebsd

many others

Family Tree



multics

1960s MIT, AT&T, Bell Labs, GE

~ 100 installations

Ken Thompson,
Dennis Ritchie

1970s
bell labs

unix

linux

freebs

thers

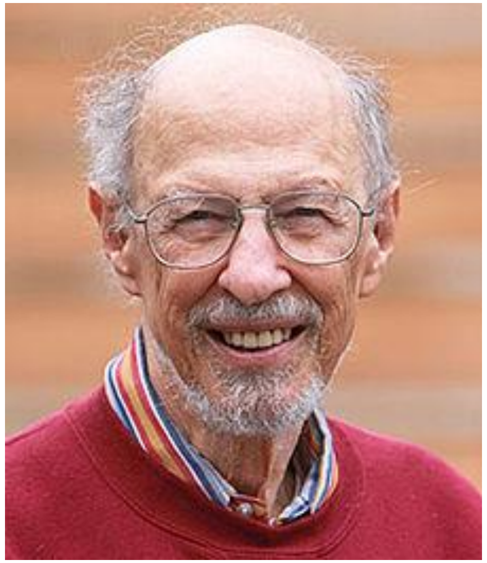
Have you used UNIX in the past 30 seconds?

poll

Multics: ancestors for many OSs

- Lots of design innovations - including lots of security innovations
  - Shared memory multiprocessor (SMP)
  - Single-level store → Segmentation and virtual memory
  - Dynamic linking
  - Run-time hardware reconfiguration
  - Hierarchical file system

Designed to be secure from the beginning

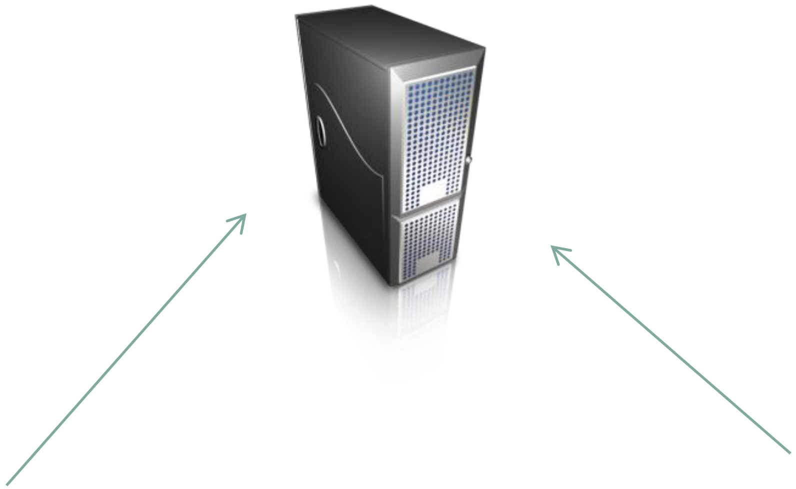


Fernando J. Corbató

MIT

Multi-level security (MLS)

- Military and other government entities want to use time-sharing too



Top secret data

Unclassified data

Sensitivity levels

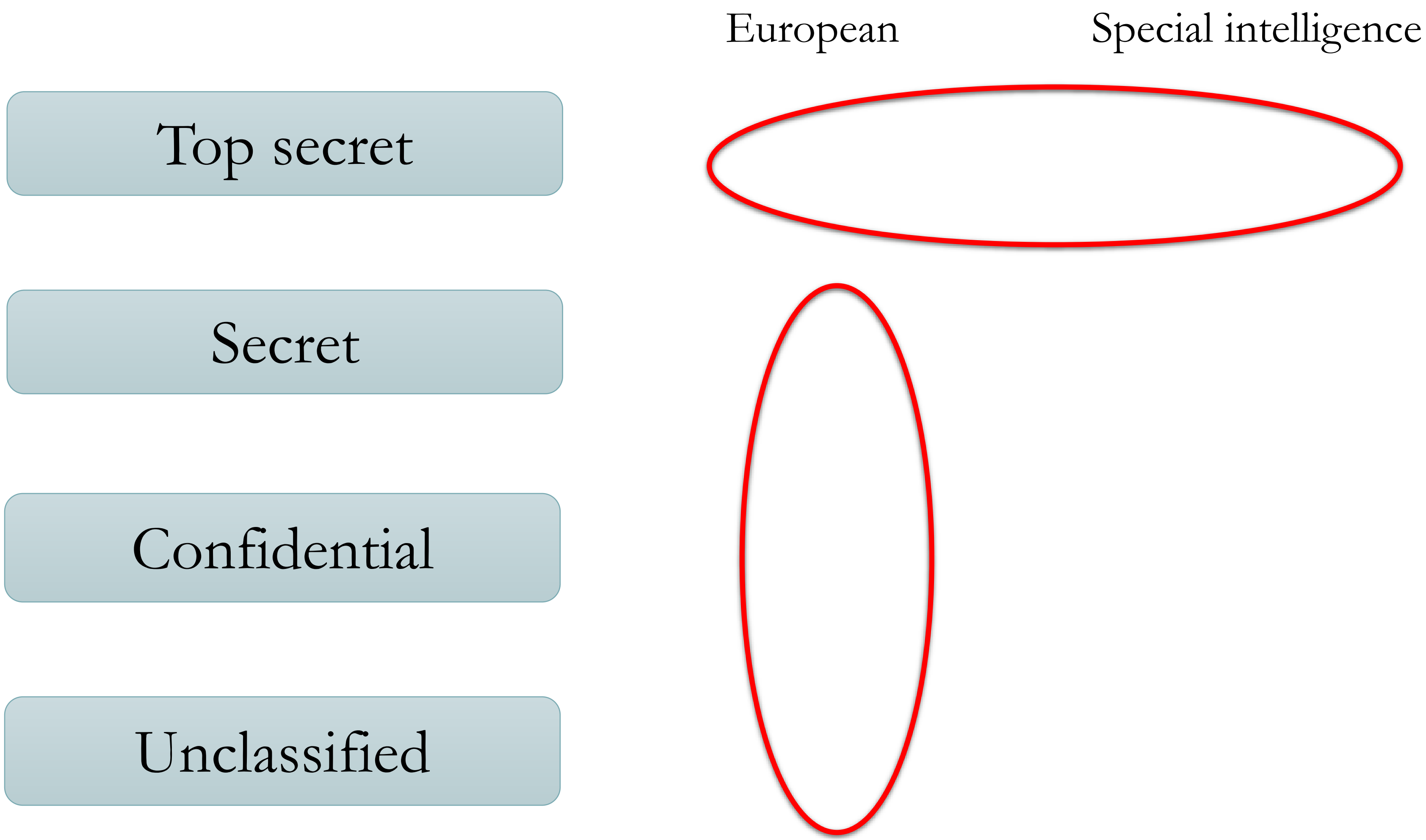
Top secret

Secret

Confidential

Unclassified

Sensitivity levels and *compartments*



European

Special intelligence

Top secret

Secret

Confidential

Unclassified

Security label

- Security label L = (`S`, `C`)
  - `S` is classification level (Top secret, secret, …)
  - `C` is compartment (Europe, Special intelligence…)

Dominance relationship: `L1 ≤ L2`

`(S1,C1) ≤ (S2,C2)`

`S1 < S2` (`S1` "less secret" than `S2`)

`C1` subset of `C2`

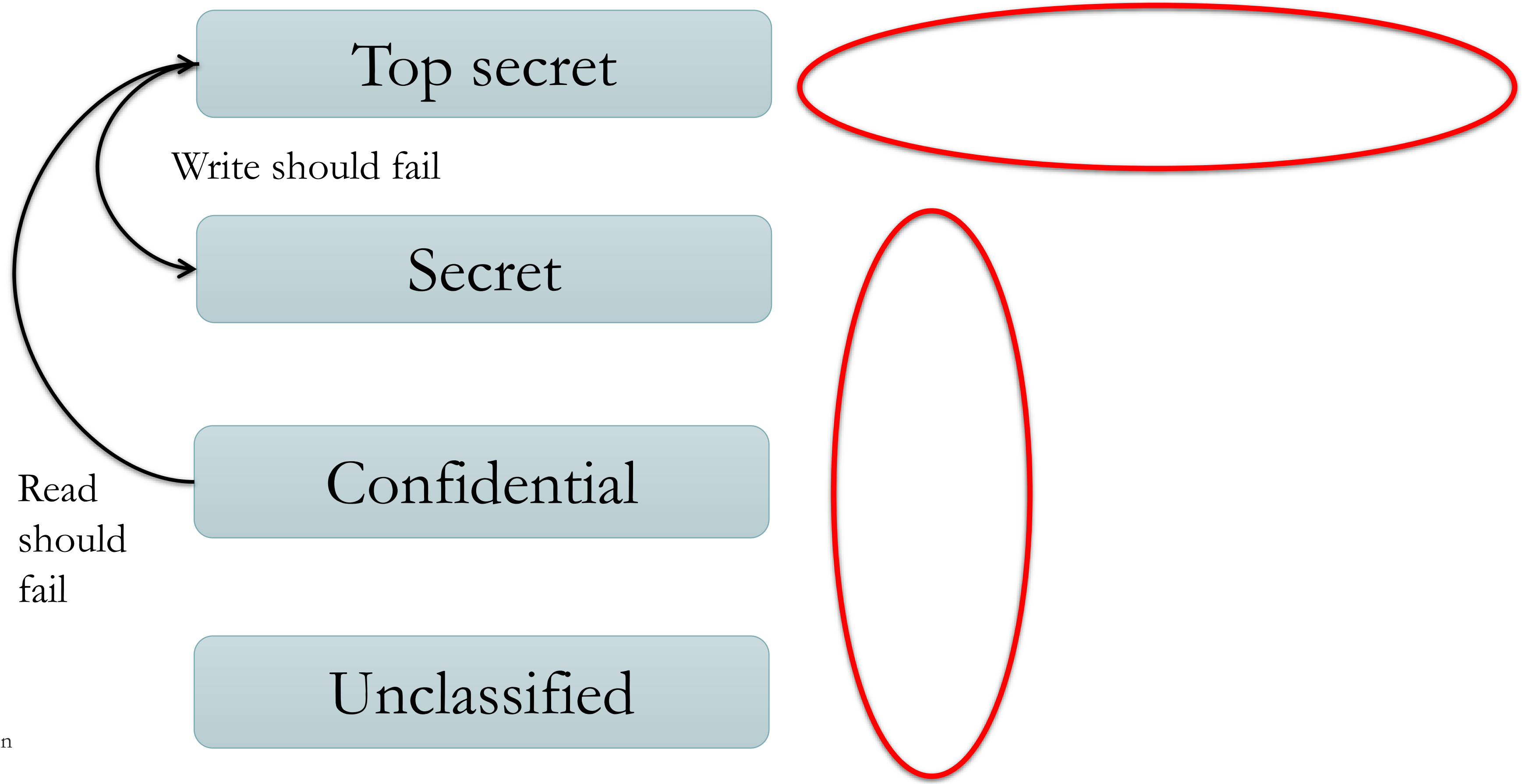
Example:

(Secret, {European}) ≤ (Top Secret, {European, Special Intel})

Bell-LaPadula Confidentiality Model

Information should not flow down: “no reads up”, “no writes down”

European Special intelligence



Top secret

Write should fail

Secret

Confidential

Read should fail

Unclassified

Bell-LaPadula Confidentiality Model

Information should not flow down: “no reads up”, “no writes down”

Simple security condition

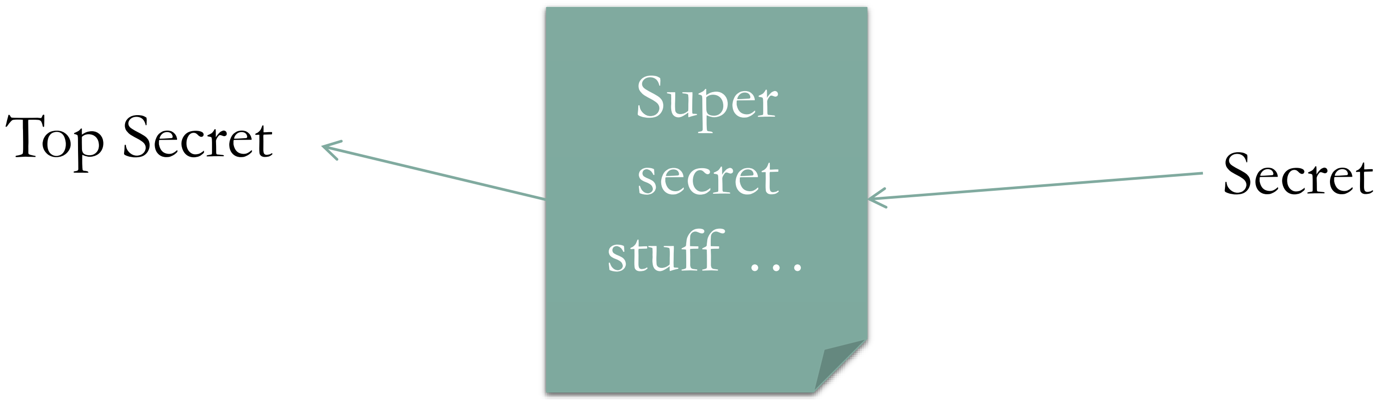
User with (S1,C1) can read file with (S2, C2) if?

~~(S1,C1) ≤ (S2,C2)~~ or (S1,C1) ≥ (S2,C2)

*-property

User with (S1,C1) can write file with (S2,C2) if?

(S1,C1) ≤ (S2,C2) or ~~(S1,C1) ≥~~ (S2,C2)



Top Secret

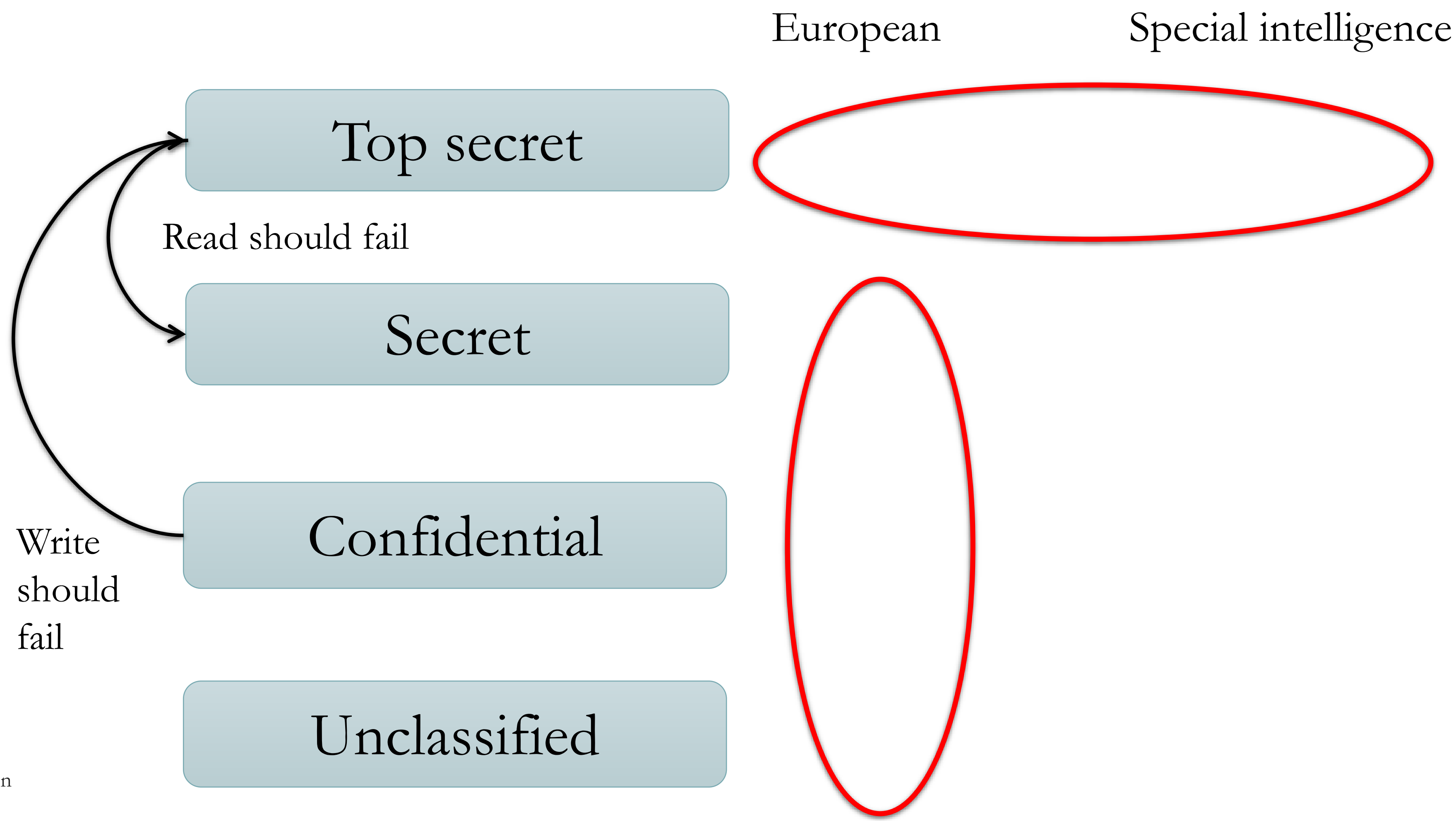
Super secret stuff …

Secret

Say we have just Bell-Lapadula in effect… what could go wrong?

Biba integrity model

"no read down", "no writes up"



Biba integrity model

“no read down”, “no writes up”

Simple integrity condition

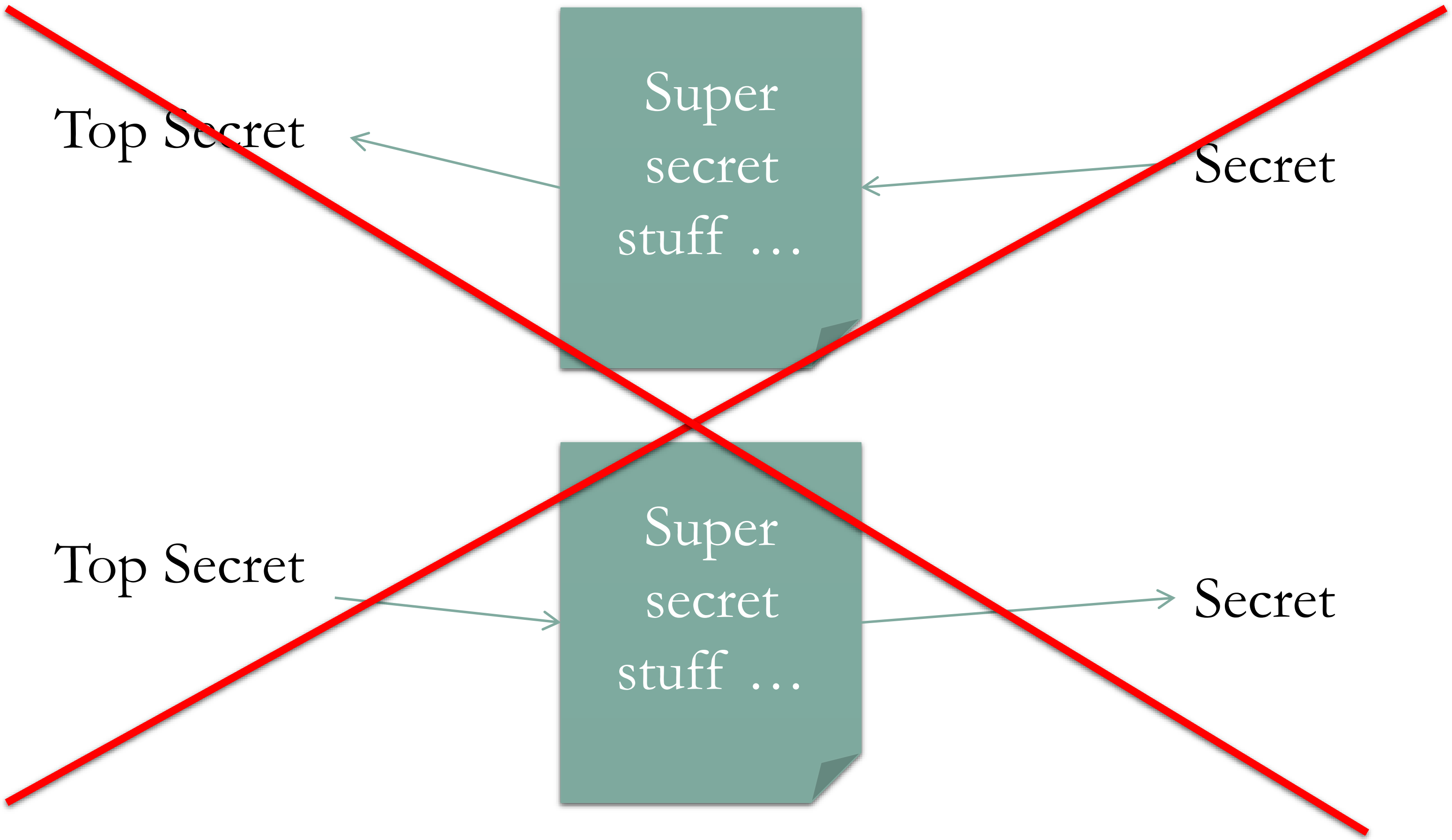
User with (S1,C1) can read file with (S2,C2) if?

`(S1,C1) ≤ (S2,C2)` or ~~(S1,C1) ≥ (S2,C2)~~

*-property

User with (S1,C1) can write file with (S2,C2) if

~~(S1,C1) ≤ (S2,C2)~~ or (S1,C1) ≥ (S2,C2)



If we combine them… one can only communicate in same sensitivity

Other policy models

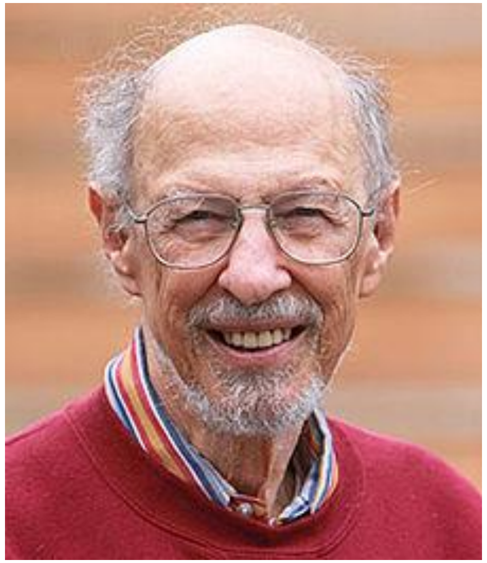
- Capability model
- Decentralized information flow control
- Take-grant protection model
- Chinese wall
- Clarke-Wilson integrity model

A good reference is:
Bishop, Computer Security: Art and Science

Multics: ancestor for many OSs

- Lots of design innovations - including lots of security innovations
  - Shared memory multiprocessor (SMP)
  - Single-level store → Segmentation and virtual memory
  - Dynamic linking
  - Run-time hardware reconfiguration
  - Hierarchical file system

Designed to be secure from the beginning



Fernando J. Corbató
MIT

Several security mechanisms in MULTICS

- HW security controls
  - Memory segmentation
  - Master mode (Secure mode)

- SW security controls
  - Protection rings
  - Access control lists

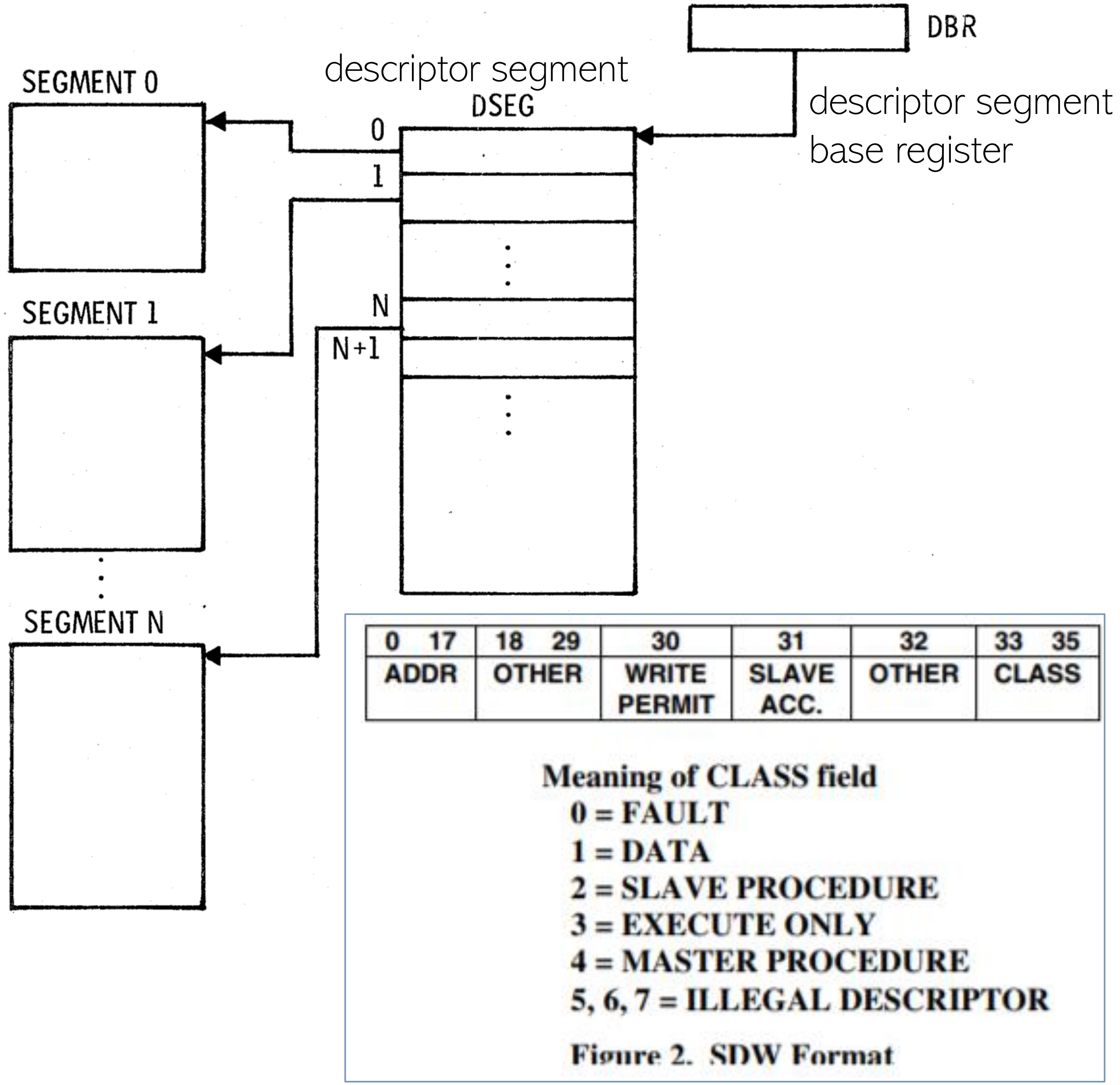
- Procedural security controls
  - “Enciphered” passwords
  - Login audit trail

/ Karger and Schell,
1974

Multics Security Evaluation: Vulnerability Analysis

HW security control / Memory Isolation

- virtual memory
- program and data stored in segments
- descriptor control field
  // read, write, execute
- segments are access controlled



0 17	18 29	30	31	32	33 35
ADDR	OTHER	WRITE PERMIT	SLAVE ACC.	OTHER	CLASS

Meaning of CLASS field
0 = FAULT
1 = DATA
2 = SLAVE PROCEDURE
3 = EXECUTE ONLY
4 = MASTER PROCEDURE
5, 6, 7 = ILLEGAL DESCRIPTOR

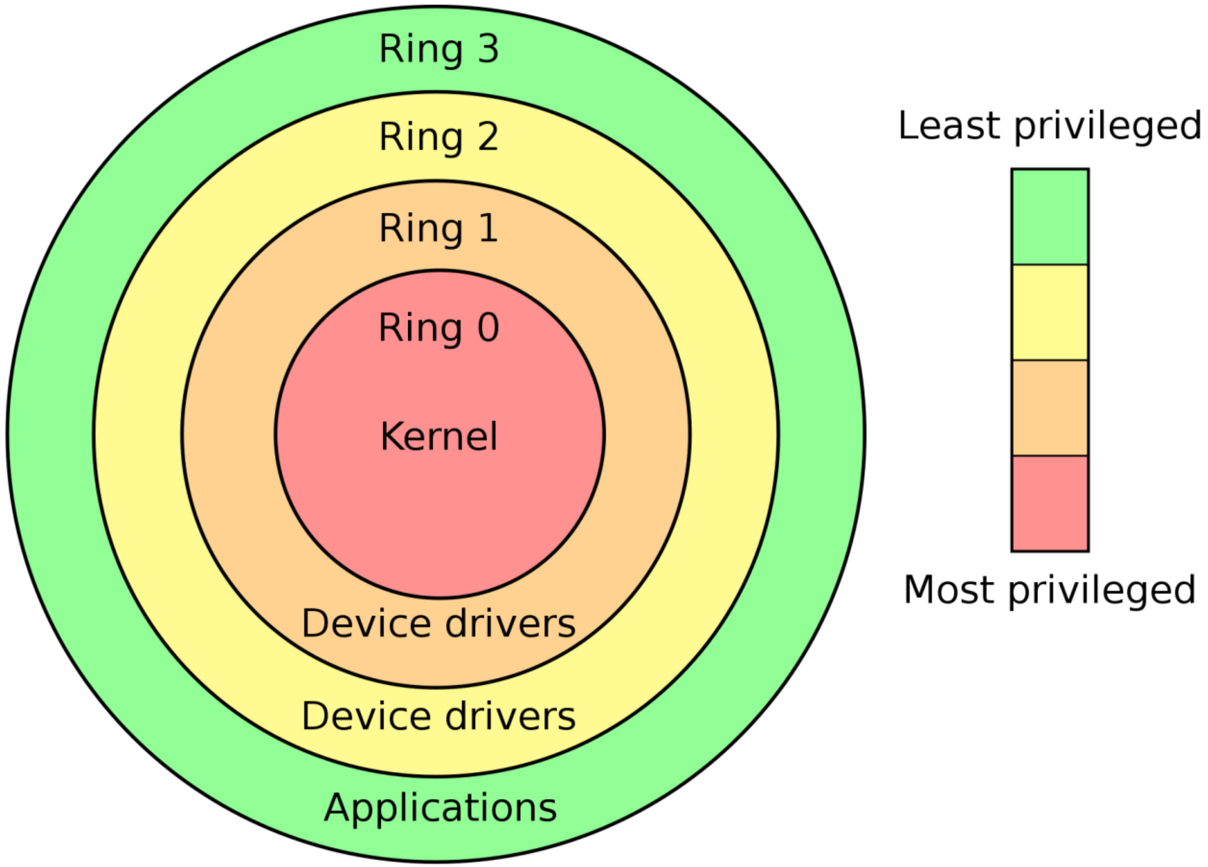
Figure 2. SDW Format

https://www.acsac.org/2002/papers/classic-multics-orig.pdf

SW security control / Protection Rings

Protection rings 0-7
in which processes execute

- Lower number = higher privilege
- Ring 0 is supervisor
- Inherit privileges over higher levels



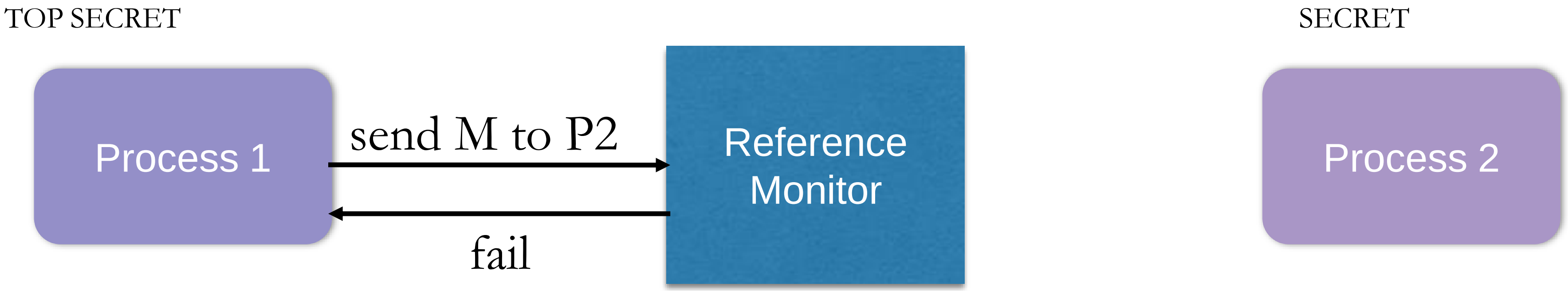
Protection rings included in all typical CPUs today
and
used by most operating systems

Reference Monitor

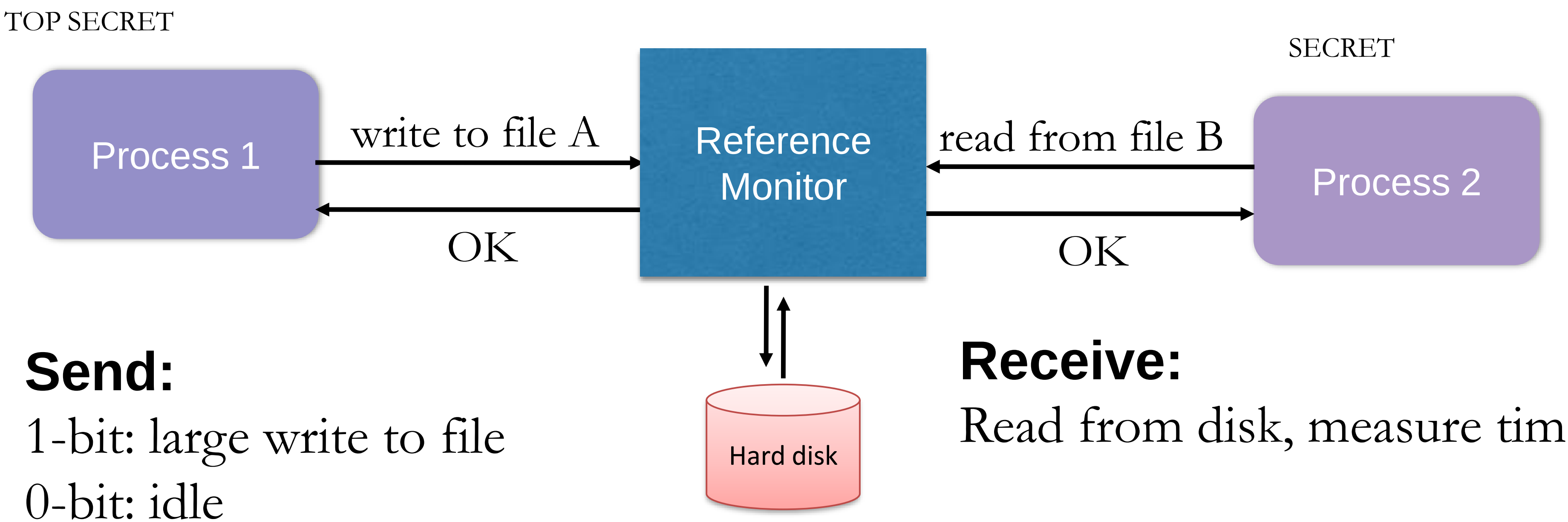
Reference monitor or security kernel

- Monitors all data access
- Enforces security policy

Multics security policy: no flow from “high classification” to “lower classification”



But covert channel…



Send:

1-bit: large write to file

0-bit: idle

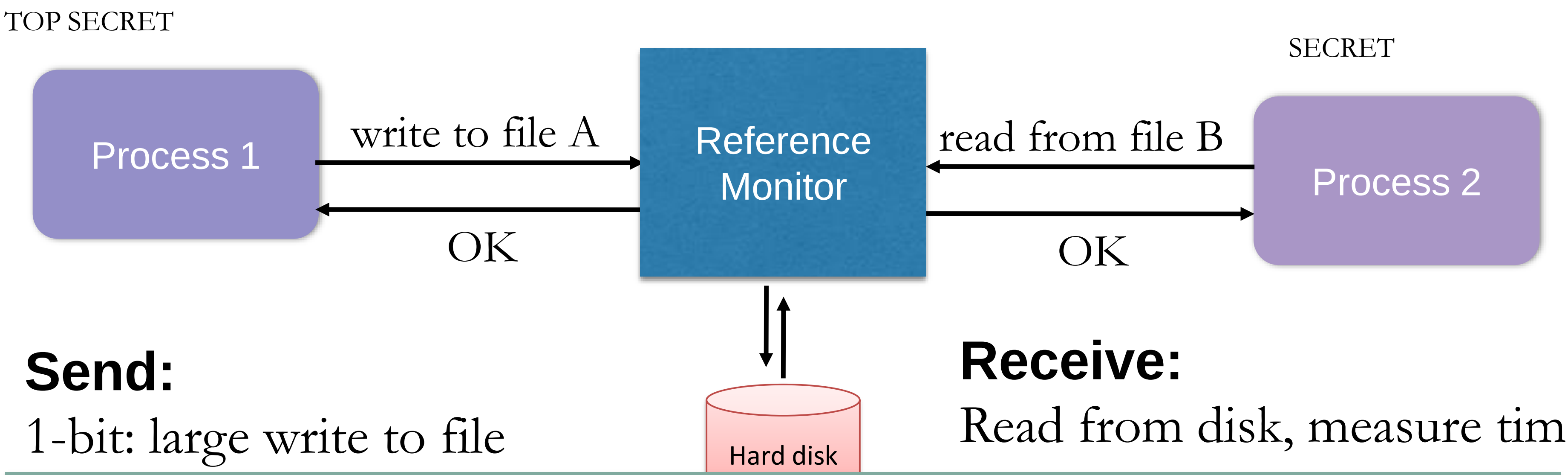
Receive:

Read from disk, measure time

longer read time = 1-bit

shorter read time = 0-bit

But covert channel…



Send:

1-bit: large write to file

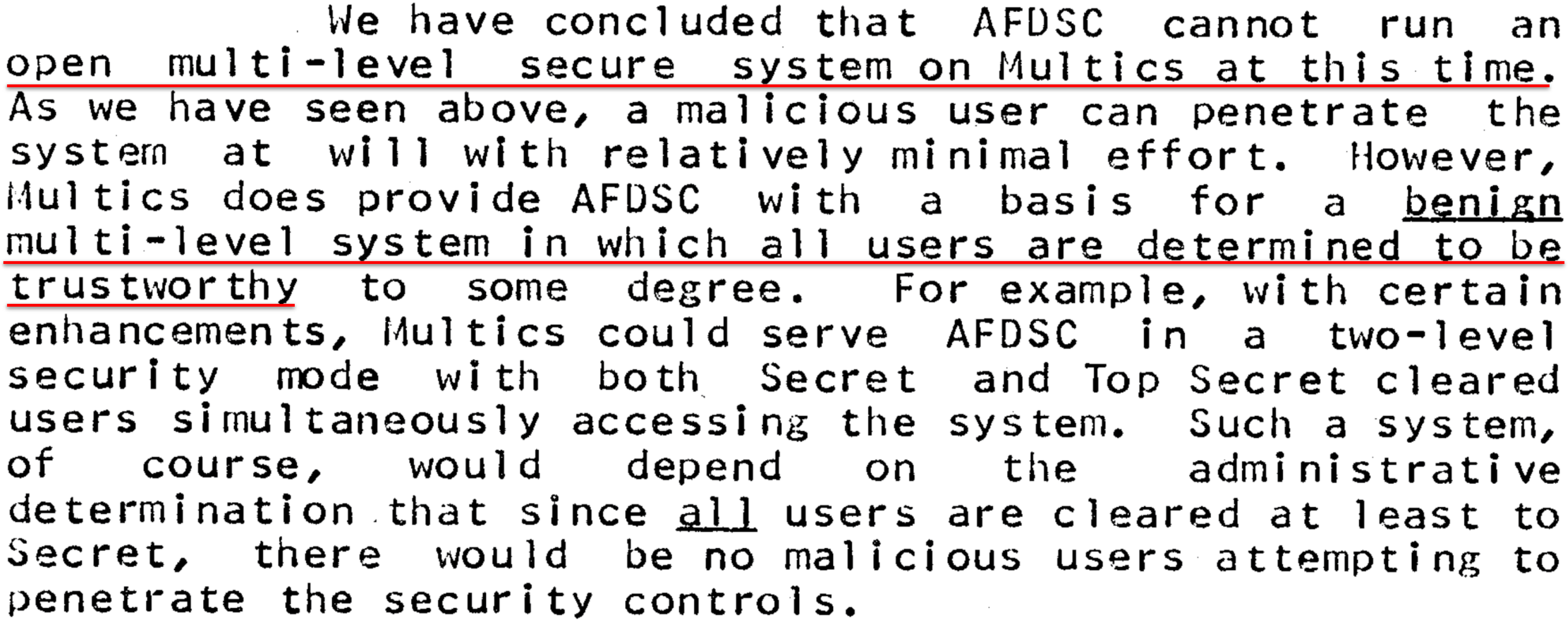
Receive:

Read from disk, measure time

Covert channel: circumvents reference monitor and security policy

shorter read time [illegible] 0-bit

Karger and Schell: security analysis of Multics



We have concluded that AFDSC cannot run an open multi-level secure system on Multics at this time. As we have seen above, a malicious user can penetrate the system at will with relatively minimal effort. However, Multics does provide AFDSC with a basis for a benign multi-level system in which all users are determined to be trustworthy to some degree. For example, with certain enhancements, Multics could serve AFDSC in a two-level security mode with both Secret and Top Secret cleared users simultaneously accessing the system. Such a system, of course, would depend on the administrative determination that since all users are cleared at least to Secret, there would be no malicious users attempting to penetrate the security controls.

Access Control

/home/rahul
./scripts
./teaching
./pwdata

/home/sujay
./lectures
./projects
./gitbucket

/home/hugh
./Projects
./latex
./Courses

/etc/init.d
./sshd
./xrdp ….

Access Control Matrix

Permitted Operations

Objects (files)

Subjects (users)

	a	**b**	**c**	**d**	**e**
rahul	r,w	-	r,w, own	-	r
sujay	-	-	r	r	r,w
hugh	w, own	r	r	-	-
kpat	R	r,w	r,w	-	r

Access control matrix: [Lampson, Graham, Denning; 1971]

But, too much space overhead to have a 2D matrix!

In practice though …

Rarely used

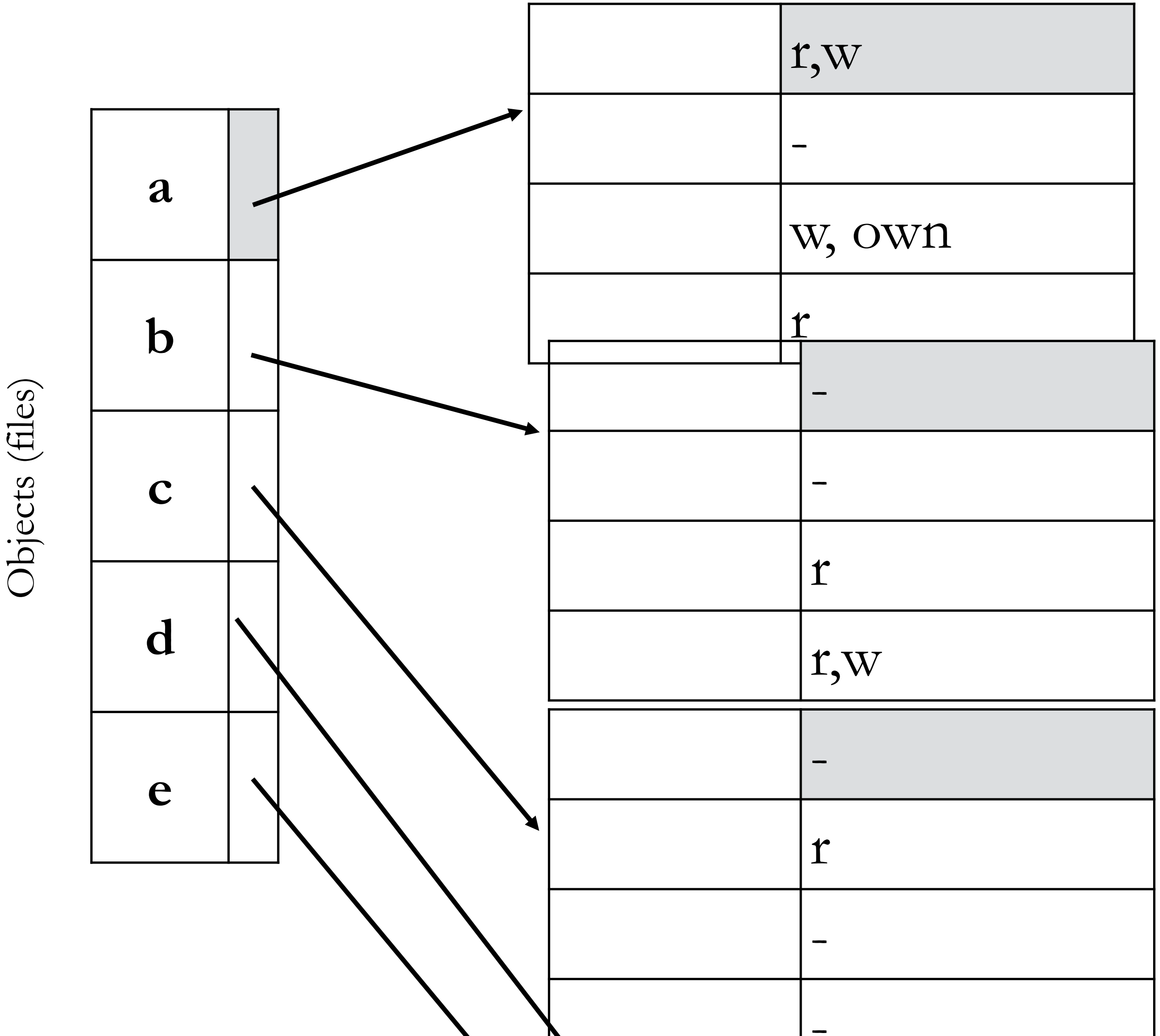
1. Access control lists

- Each file contain lists of users with their permissions (column in AC matrix)
- Need explicit user authentication
- Process must be given permissions
- Reference monitor must protect permission setting

2. Capability-based security

- Tickets granted to users to perform some operation
- Stores each user's capabilities (row in AC matrix)
- Token-based approach,
  - no need to for explicit auth
- Tokens can be passed around
- Reference monitor must manage tokens

Access Control List (ACL)



Objects (files)

a

b

c

d

e

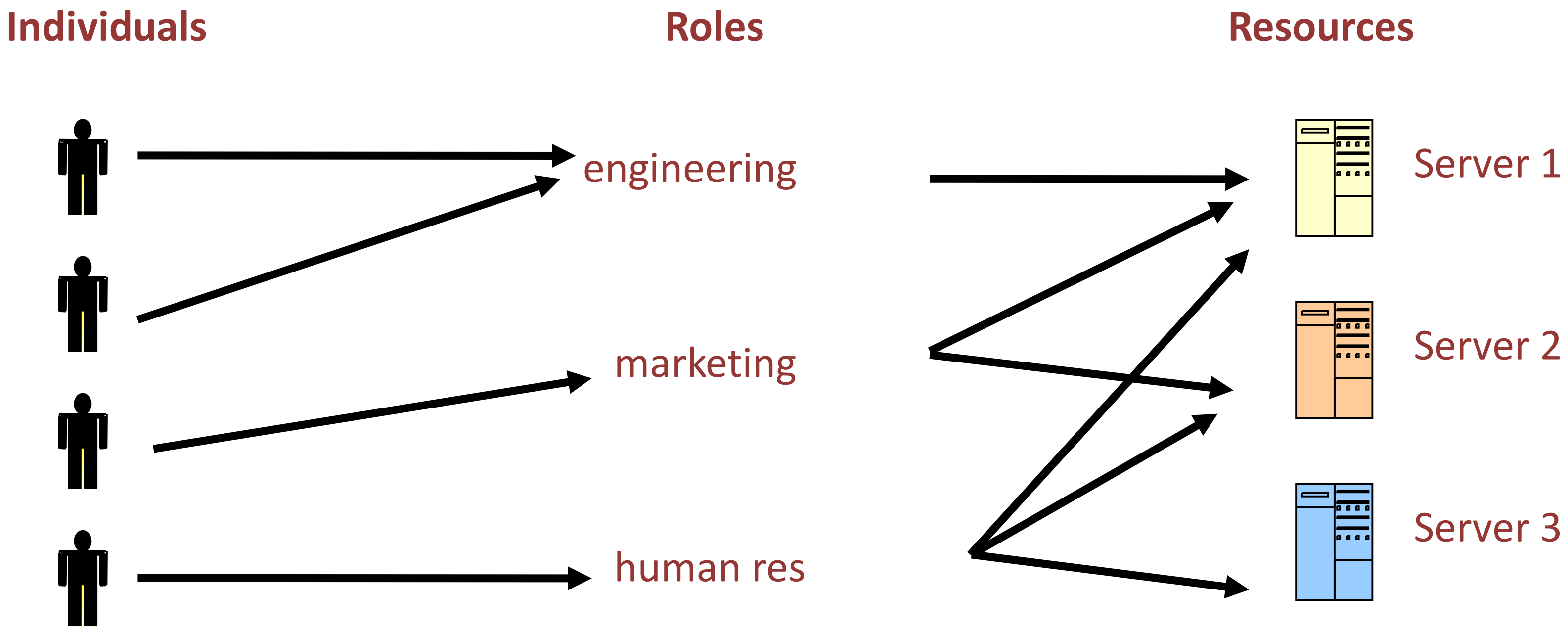
	r,w
	-
	w, own
	r

	-
	-
	r
	r,w

	-
	r
	-
	-

Roles (groups)

- Role-based access control
- Role = set of users



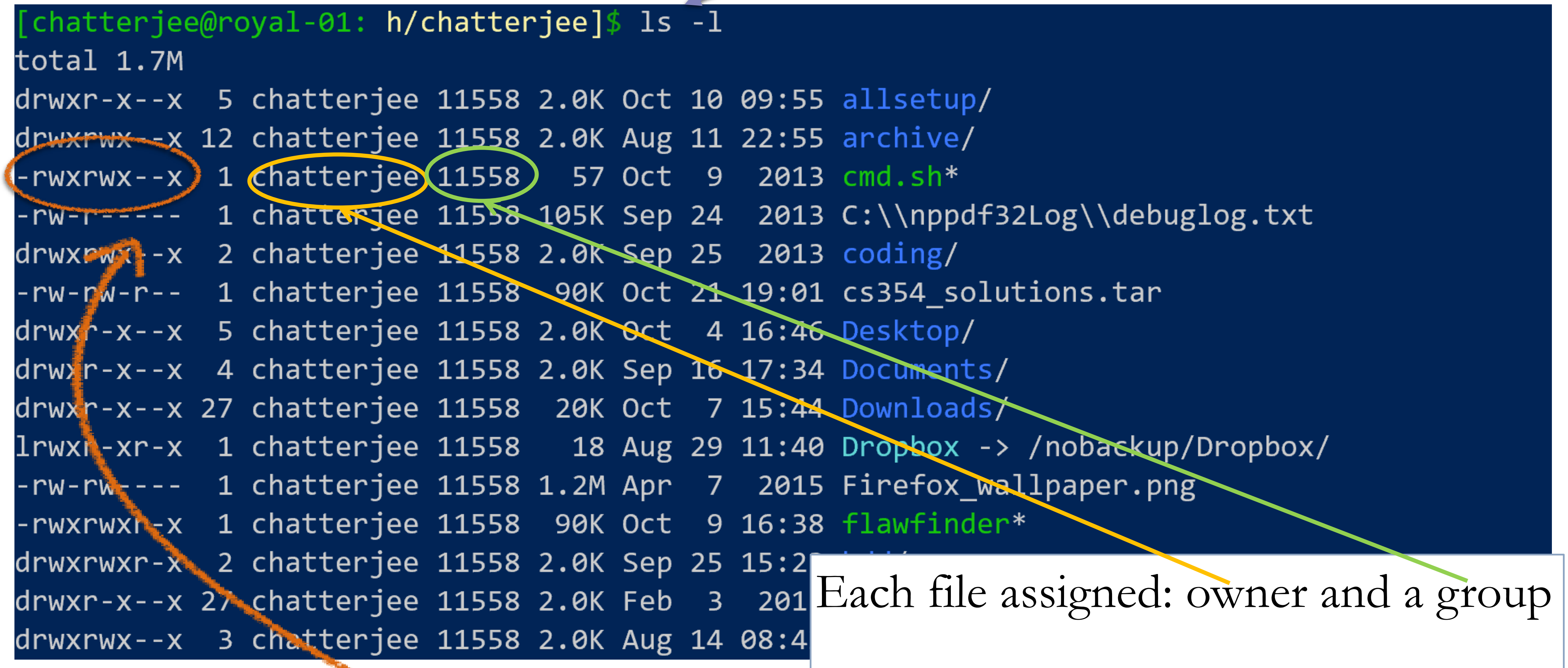
Advantages:

/ many users, few roles

/ individuals come-and-go frequently, groups are more stable

UNIX access control

View file permissions



```
[chatterjee@royal-01: h/chatterjee]$ ls -l
total 1.7M
drwxr-x--x  5 chatterjee 11558 2.0K Oct 10 09:55 allsetup/
drwxrwx--x 12 chatterjee 11558 2.0K Aug 11 22:55 archive/
-rwxrwx--x  1 chatterjee 11558   57 Oct  9  2013 cmd.sh*
-rw-r-----  1 chatterjee 11558 105K Sep 24  2013 C:\\nppdf32Log\\debuglog.txt
drwxrwx--x  2 chatterjee 11558 2.0K Sep 25  2013 coding/
-rw-rw-r--  1 chatterjee 11558  90K Oct 21 19:01 cs354_solutions.tar
drwxr-x--x  5 chatterjee 11558 2.0K Oct  4 16:46 Desktop/
drwxr-x--x  4 chatterjee 11558 2.0K Sep 16 17:34 Documents/
drwxr-x--x 27 chatterjee 11558  20K Oct  7 15:44 Downloads/
lrwxr-xr-x  1 chatterjee 11558   18 Aug 29 11:40 Dropbox -> /nobackup/Dropbox/
-rw-rw----  1 chatterjee 11558 1.2M Apr  7  2015 Firefox_wallpaper.png
-rwxrwxr-x  1 chatterjee 11558  90K Oct  9 16:38 flawfinder*
drwxrwxr-x  2 chatterjee 11558 2.0K Sep 25 15:2
drwxr-x--x 27 chatterjee 11558 2.0K Feb  3  201
drwxrwx--x  3 chatterjee 11558 2.0K Aug 14 08:4
```

access control list

Each file assigned: owner and a group

Basic operations: read, write, execute

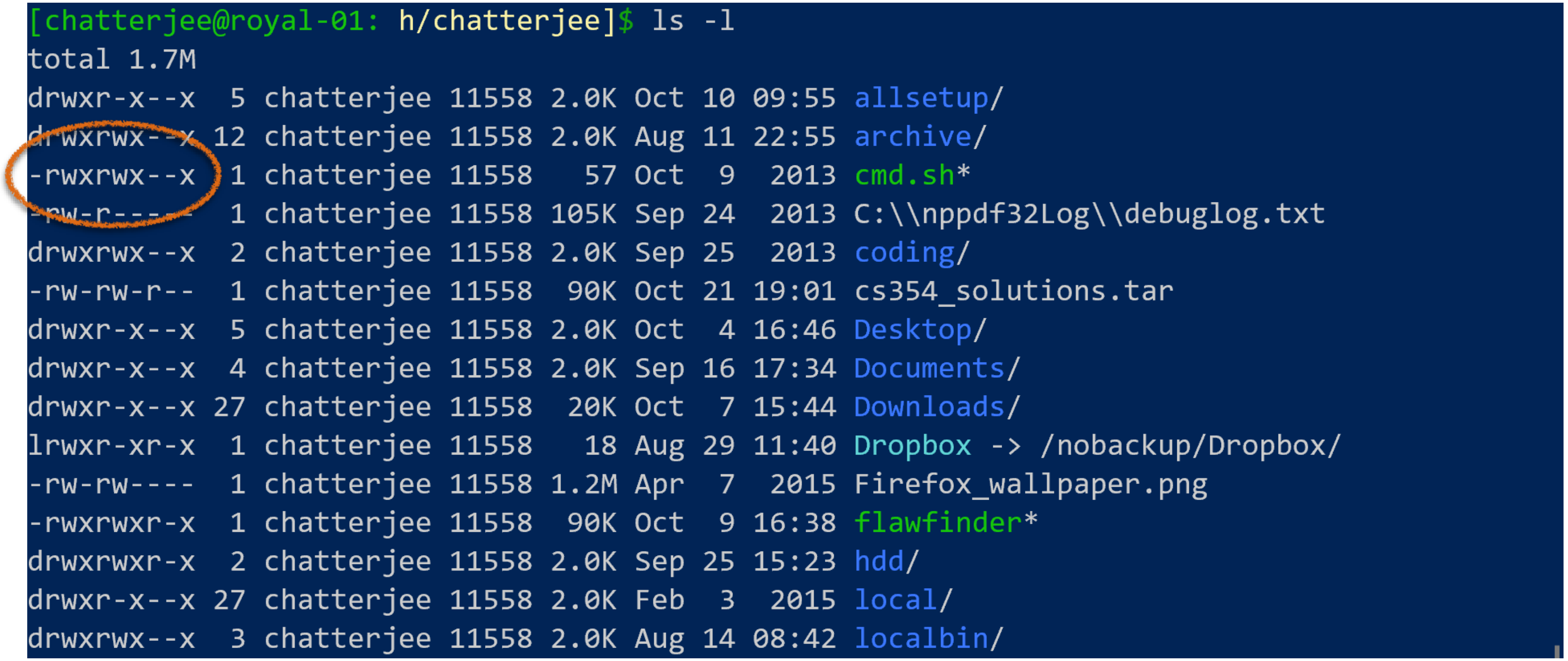
UNIX access control details

- Unix uses **role-based** access control
- Role => *group*
- Individual (or process) => *user id (uid)*

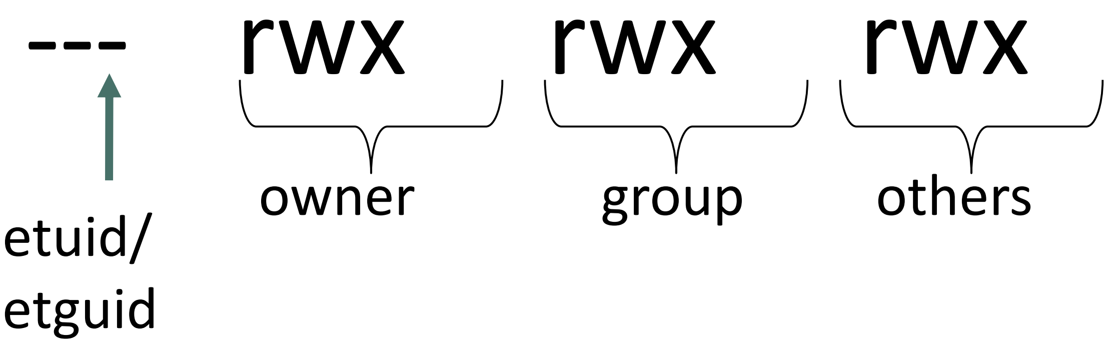
- Special user ID: uid 0
  - root user
  - permitted to do *anything*
  - for any file: can read, write, change permissions, change owners

Each file has
1. Owner
   a) User
   b) Group
2. ACL
   a) Owner's access
   b) Group's access
   c) World's access

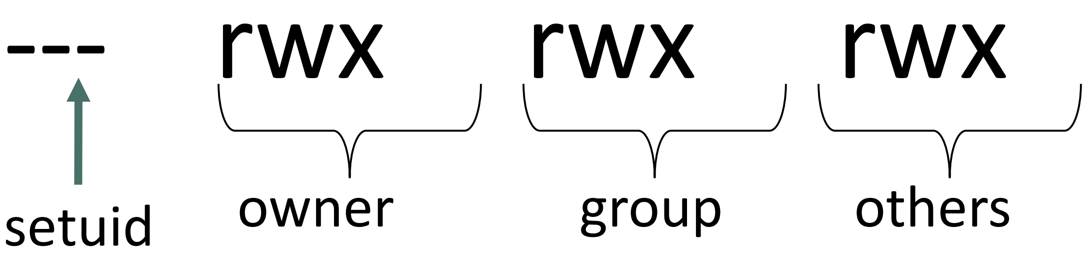
UNIX ACL



```
[chatterjee@royal-01: h/chatterjee]$ ls -l
total 1.7M
drwxr-x--x  5 chatterjee 11558 2.0K Oct 10 09:55 allsetup/
drwxrwx--x 12 chatterjee 11558 2.0K Aug 11 22:55 archive/
-rwxrwx--x  1 chatterjee 11558   57 Oct  9  2013 cmd.sh*
-rw-r-----  1 chatterjee 11558 105K Sep 24  2013 C:\\nppdf32Log\\debuglog.txt
drwxrwx--x  2 chatterjee 11558 2.0K Sep 25  2013 coding/
-rw-rw-r--  1 chatterjee 11558  90K Oct 21 19:01 cs354_solutions.tar
drwxr-x--x  5 chatterjee 11558 2.0K Oct  4 16:46 Desktop/
drwxr-x--x  4 chatterjee 11558 2.0K Sep 16 17:34 Documents/
drwxr-x--x 27 chatterjee 11558  20K Oct  7 15:44 Downloads/
lrwxr-xr-x  1 chatterjee 11558   18 Aug 29 11:40 Dropbox -> /nobackup/Dropbox/
-rw-rw----  1 chatterjee 11558 1.2M Apr  7  2015 Firefox_wallpaper.png
-rwxrwxr-x  1 chatterjee 11558  90K Oct  9 16:38 flawfinder*
drwxrwxr-x  2 chatterjee 11558 2.0K Sep 25 15:23 hdd/
drwxr-x--x 27 chatterjee 11558 2.0K Feb  3  2015 local/
drwxrwx--x  3 chatterjee 11558 2.0K Aug 14 08:42 localbin/
```



UNIX ACLs



--- rwx rwx rwx

setuid owner group others

- Permissions set by owner (or root)
- Determining if an action is permitted:

```
if uid == 0 (root):  allow anything
else if uid == owner:  use owner  permissions
else if uid in group:  use group  permissions
else:  use other  permissions
```

- Only owner, root can change permissions
  - This privilege cannot be delegated or shared
- Setid bits – Discuss in a few slides

Exercise

owner

group

$$\underbrace{\text{rwx}}_{\text{owner}}\ \underbrace{\text{rwx}}_{\text{group}}\ \underbrace{\text{rwx}}_{\text{others}}$$

```
-rw-r--r--  1 ace  staff  1087 Aug 10 15:20 LICENSE.txt
-rw-r--r--  1 ace  staff    19 Aug 10 15:57 MANIFEST.in
-r---w-r--  1 ace  dev    1106 Aug 14 13:55 README.md
drwxr-xr-x  3 ace  staff   102 Aug 13 07:27 dist
drwxr-xr-x  8 ace  staff   272 Aug 13 10:47 safeid
drwxrwxr-x  9 ace  staff   306 Aug 13 07:26 safeid.egg
-r--------  1 ace  web      40 Aug 10 15:56 setup.cfg
-rw--w-r-x  1 ace  dev    1550 Aug 13 07:26 deploy.log
```

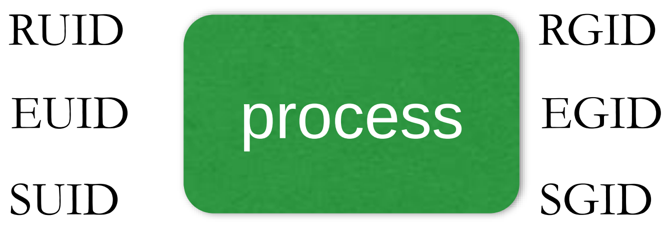
```
staff:*:29:ace,sscott,kpat,rist
web:*:31:ace,kpat,rist
dev:*:32:ace,sscott,pbriggs
```

Can sscott read the file README.md?

Can ace write to setup.cfg?

Which users can append to deploy.log?

Process IDs



RUID
EUID
SUID

process

RGID
EGID
SGID

Real User ID

/ same as the UID of parent

/ indicates who started this process

Effective User ID

/ current permissions for this process

Saved User ID

/ previous EUID so that it can be restored

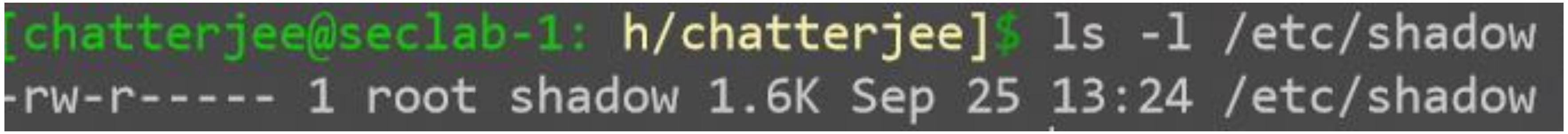
Also: Real Group ID, Effective Group ID,

How to re/set process IDs

RUID
EUID
SUID
process

- `fork/exec`
  - new process inherits all three UIDs (except for setid bit explained later)
- `seteuid(newid)` system call
  - changes EUID
  - can only change to saved UID or real UID
  - unless EUID == 0 in which case can set any ID
- Also `seteguid()`

Why?

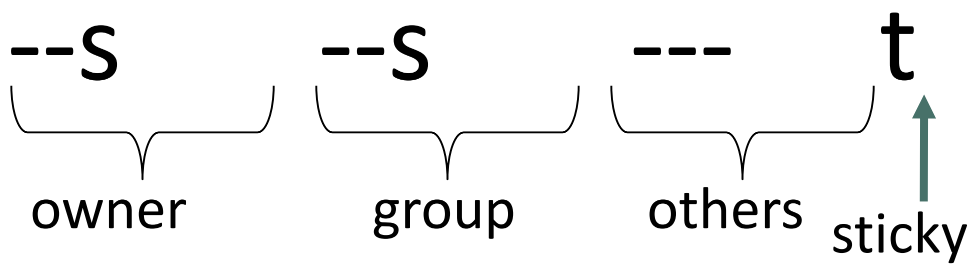


```
[chatterjee@seclab-1: h/chatterjee]$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1.6K Sep 25 13:24 /etc/shadow
```

- Many UNIX systems store passwords in the file `/etc/shadow`
- Who should be able to read this file? Write this file?
- Users change passwords using `/usr/bin/passwd`
- What EUID does this process run as?
- How can it write updates to the password file?

setid bits

setid bits



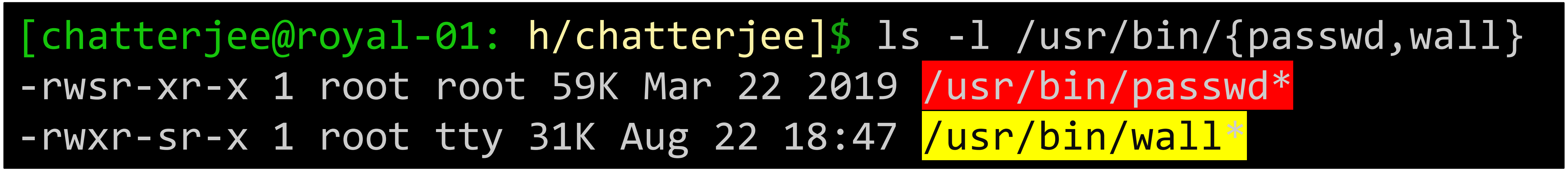
--s --s --- t

owner group others sticky

- `setuid`: on execute, set EUID of new process to file owner's UID
- `setgid`: on execute, set EGID of new process to file owner's GID
- `sticky bit` (for directories)
  - When set, restricts deletion and renaming of files

`setuid/gid`: Permits *necessary* privilege escalation

Exercise think-pair-share



```
[chatterjee@royal-01: h/chatterjee]$ ls -l /usr/bin/{passwd,wall}
-rwsr-xr-x 1 root root 59K Mar 22 2019 /usr/bin/passwd*
-rwxr-sr-x 1 root tty 31K Aug 22 18:47 /usr/bin/wall*
```

When `passwd` is started: what are the RUID, EUID, and SUID values?

When `wall` is started: what are the RUID, EUID, and SUID?

What are the RGID, EGID, and SGID?

Vulnerabilities

```
-rwsr-xr-x 1 root root 5090 Jan 16  2015 tmp-read*
```

```
...
if (access("/tmp/myfile", R_OK) != 0) {
  exit(-1);
}

file = open("/tmp/myfile", "r");
read(file, buf, 1024);
close(file);
printf("%s\n", buf);
```

Q: Where’s the vulnerability?

TOCTTOU

```
access("/tmp/myfile", R_OK)
```

```
ln -sF /home/root/.ssh/id_rsa /tmp/myfile
```

```
open("/tmp/myfile", "r");

printf("%s\n", buf);
```

Race condition between attacker and tmp-read

Prints root user's private SSH key

Vulnerability called: time-of-check to time-of-use (TOCTTOU)

Better

```
euid = geteuid();
ruid = getuid();
seteuid(ruid);       // drop privileges
file = open("/tmp/myfile", "r");
read(file, buf, 1024);
close(file);
print("%s\n", buf);
```

Prevents that attack

/etc/passwd: ace:*:19: ...

```
EUID

0     euid = geteuid();
0     ruid = getuid();
19    seteuid(ruid);          // drop privileges
```

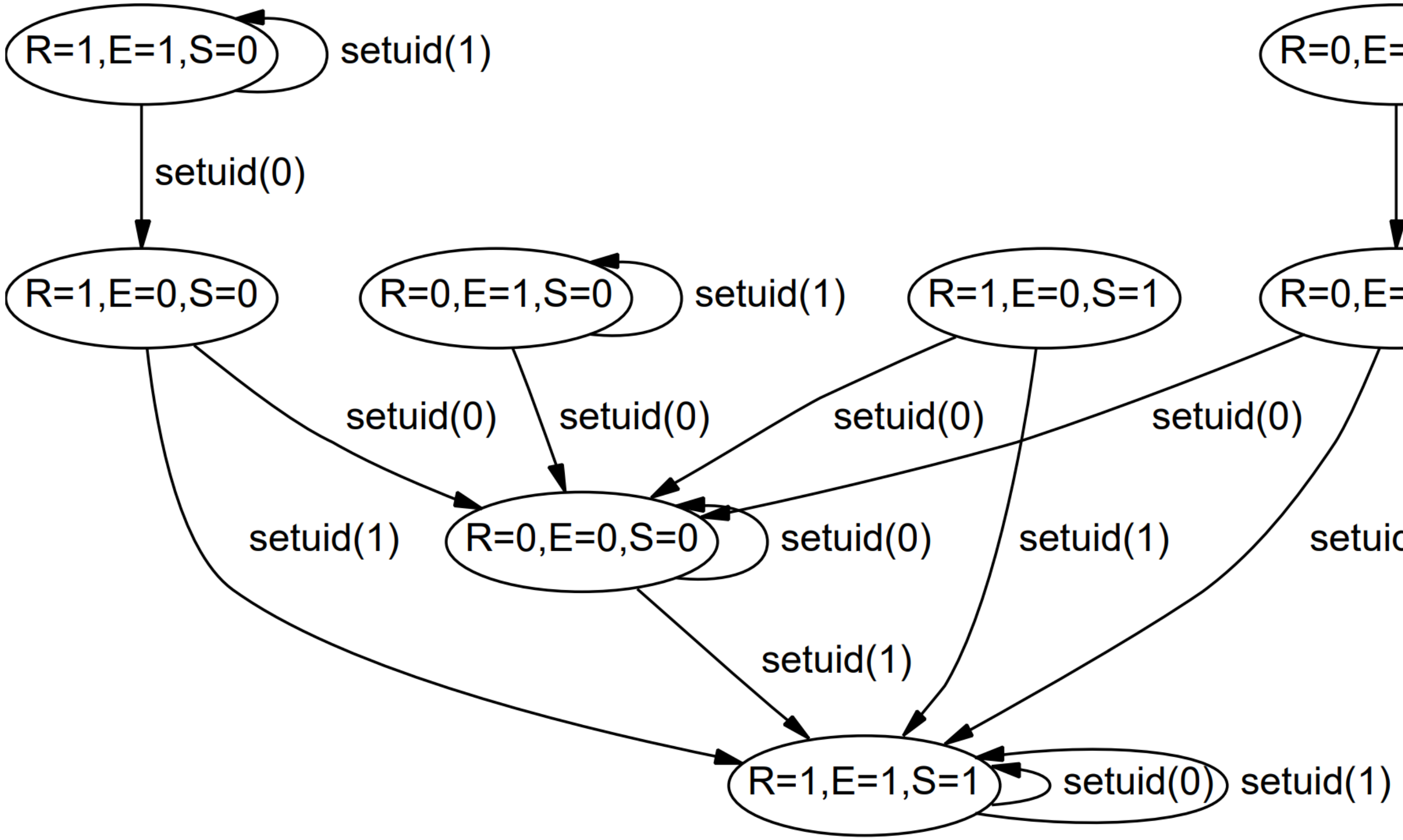
```
ln -sF /home/root/.ssh/id_rsa /tmp/myfile
```

```
19    file = open("/tmp/myfile", "r");
         error: errno=13 (Permission denied).
```

What security design principle?

> Least privilege

setid / In practice, setid is even more complicated



(a) An FSA describing *setuid* in Linux 2.4.18

[Chen, Wagner, Dean. *Setuid Demystified*]

setid

- setid permits necessary privilege escalation

⁎ Source of many privilege escalation vulnerabilities
/race conditions (tocttou)
/control-flow hijacking

Recap

- Principles for Secure Designs
- Multics: security design features, covert channel
- Access control matrix and ACLs
- Unix file access control
- setid bits and seteuid system call