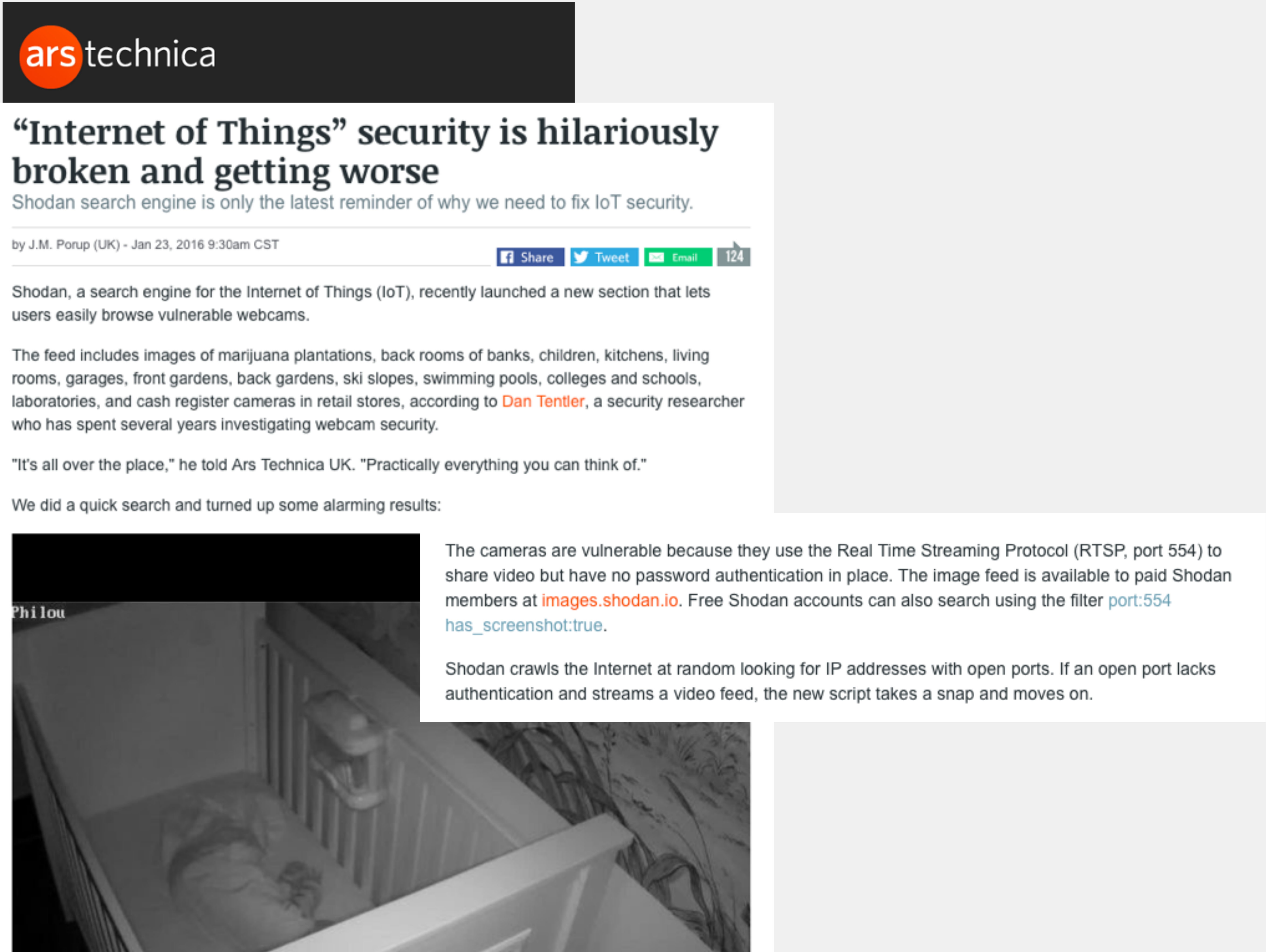


ars technica

"Internet of Things" security is hilariously broken and getting worse

Shodan search engine is only the latest reminder of why we need to fix IoT security.

by J.M. Porup (UK) - Jan 23, 2016 9:30am CST

Shodan, a search engine for the Internet of Things (IoT), recently launched a new section that lets users easily browse vulnerable webcams.

The feed includes images of marijuana plantations, back rooms of banks, children, kitchens, living rooms, garages, front gardens, back gardens, ski slopes, swimming pools, colleges and schools, laboratories, and cash register cameras in retail stores, according to Dan Tentler, a security researcher who has spent several years investigating webcam security.

"It's all over the place," he told Ars Technica UK. "Practically everything you can think of."

We did a quick search and turned up some alarming results:

The cameras are vulnerable because they use the Real Time Streaming Protocol (RTSP, port 554) to share video but have no password authentication in place. The image feed is available to paid Shodan members at images.shodan.io. Free Shodan accounts can also search using the filter port:554 has_screenshot:true.

Shodan crawls the Internet at random looking for IP addresses with open ports. If an open port lacks authentication and streams a video feed, the new script takes a snap and moves on.

cs642

computer
security

/operating system security

adam everspaugh
ace@cs.wisc.edu

principles

Principles of Secure Designs

- Compartmentalization
  / Isolation
  / Least privilege

- Defense-in-depth
  / Use more than one security mechanism
  / Secure the weakest length
  / Fail securely

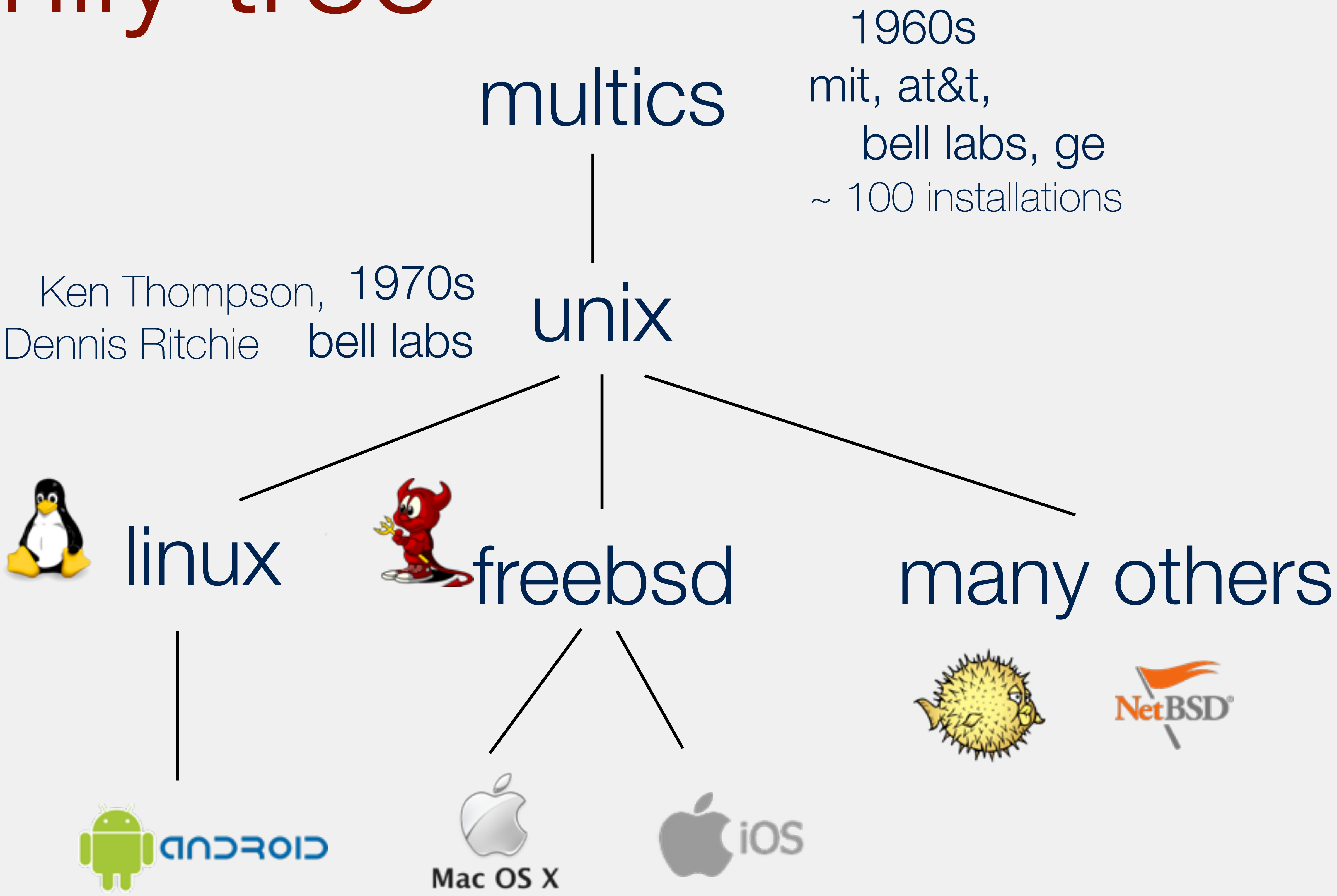
- Keep it simple
  / Economy of mechanism
  / Psychological acceptability
  / Good defaults

- Open Design

Have you used UNIX
since noon today?

poll

family tree



multics

1960s
mit, at&t,
bell labs, ge
~ 100 installations

unix

Ken Thompson,
Dennis Ritchie

1970s
bell labs

linux

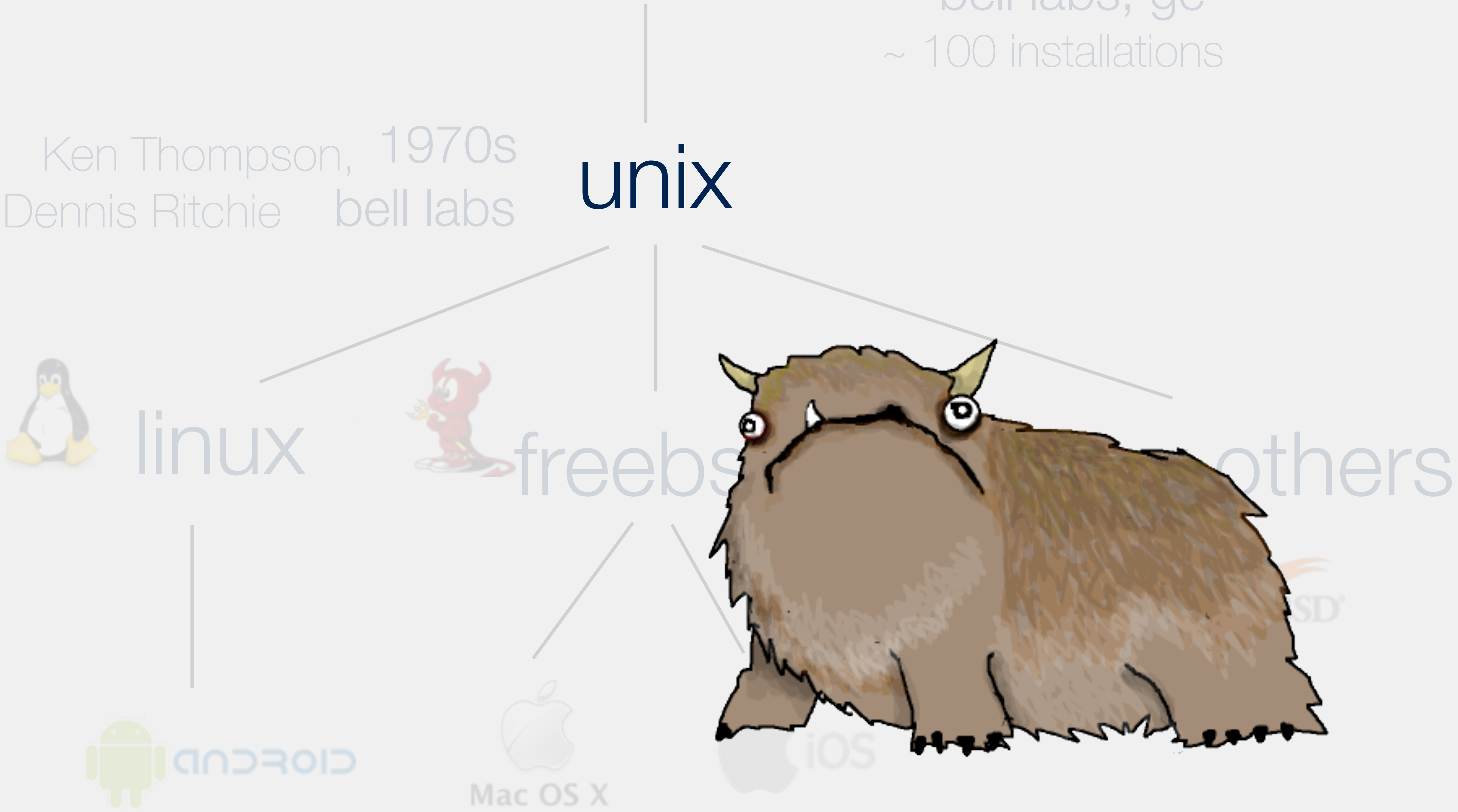
freebsd

many others

family tree

multics

1960s
mit, at&t,
bell labs, ge
~ 100 installations



Ken Thompson,
Dennis Ritchie

1970s
bell labs

unix

linux

freebs

others

Have you used UNIX
since noon today?

poll

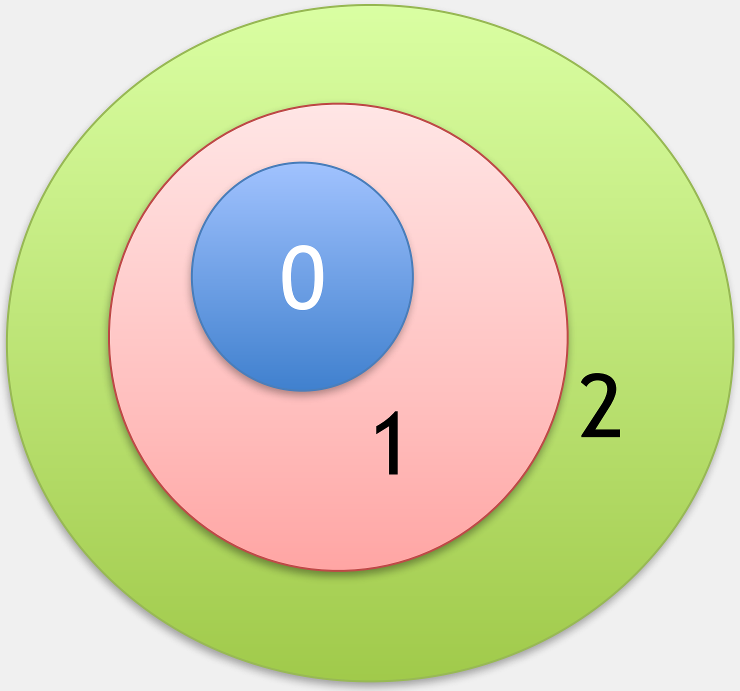
multics

- Lots of design innovations - including lots of security innovations
- Segmentation and virtual memory
- Shared memory multiprocessor (SMP)

F. Corbato, MIT

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

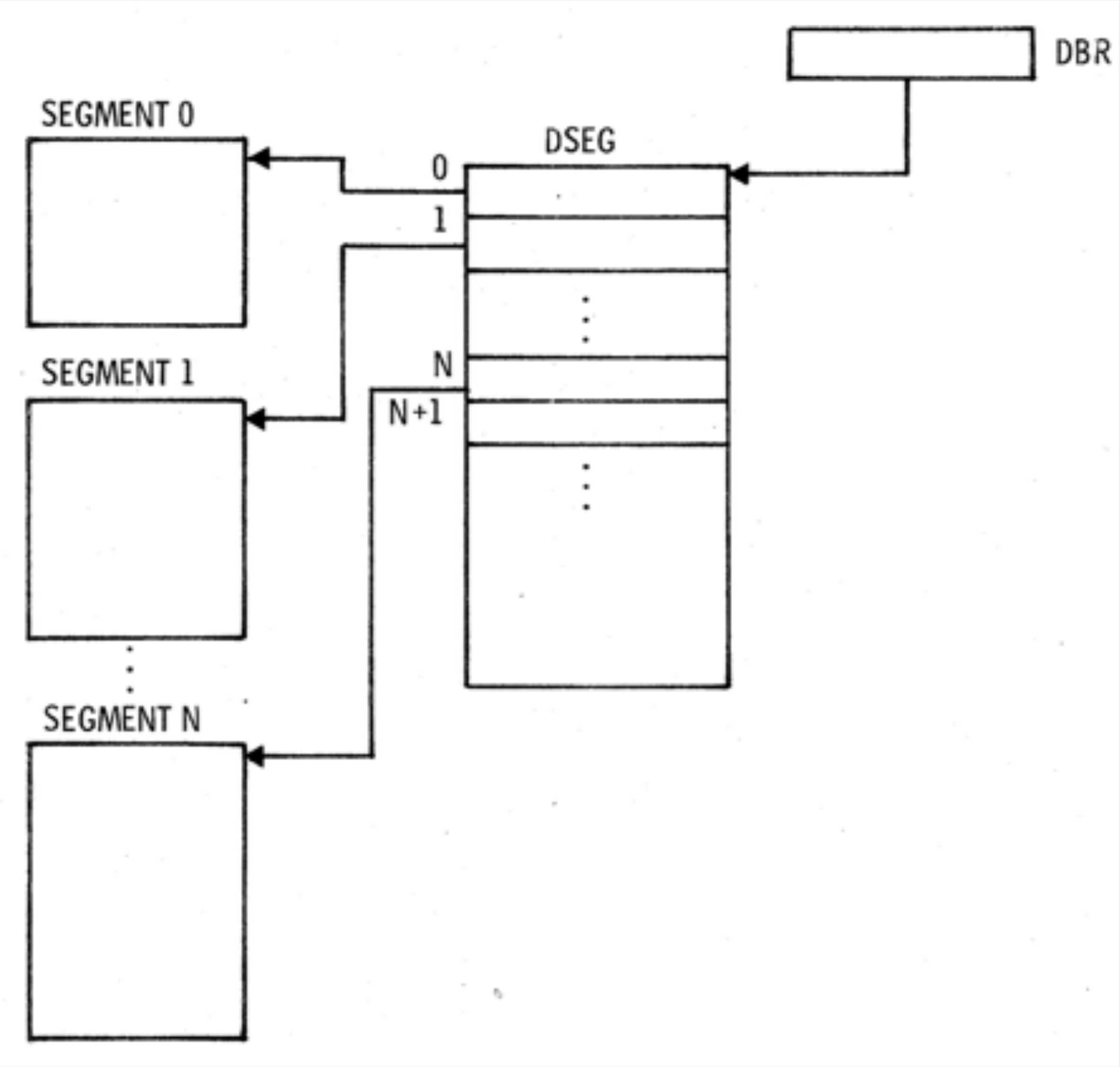
memory isolation

/ virtual memory

/ program and data stored in segments

/ descriptor control field
// read, write, execute

/ segments are access controlled



pw storage

enciphered passwords

“I was no cryptanalyst … Joe [Weizenbaum] had suggested I store the square of the password, but I knew people could take square roots, so I squared and ANDed with a mask to discard some bits.”
– T. Van Vleck

- Later ones used DES, but Multics predates DES

- Today, UNIX systems store a HASH(pw)

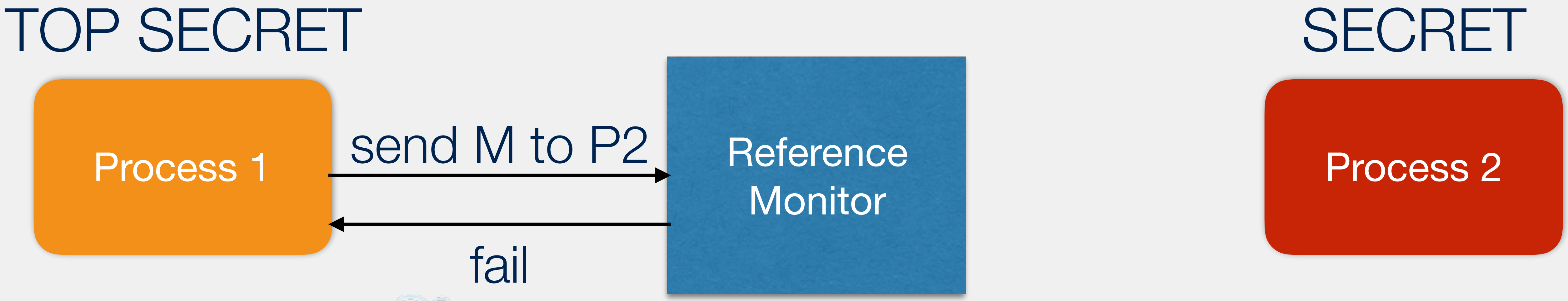
reference monitor

Reference monitor or security kernel

/ Monitors all data access

/ Enforces security policy

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



TOP SECRET

SECRET

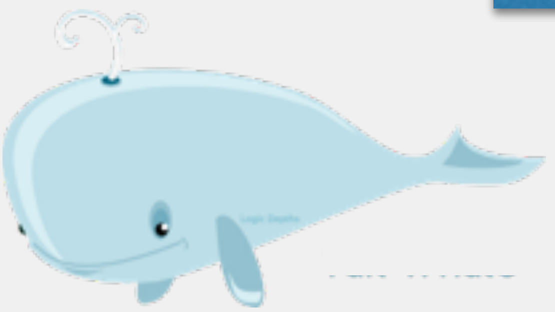
Process 1

send M to P2

fail

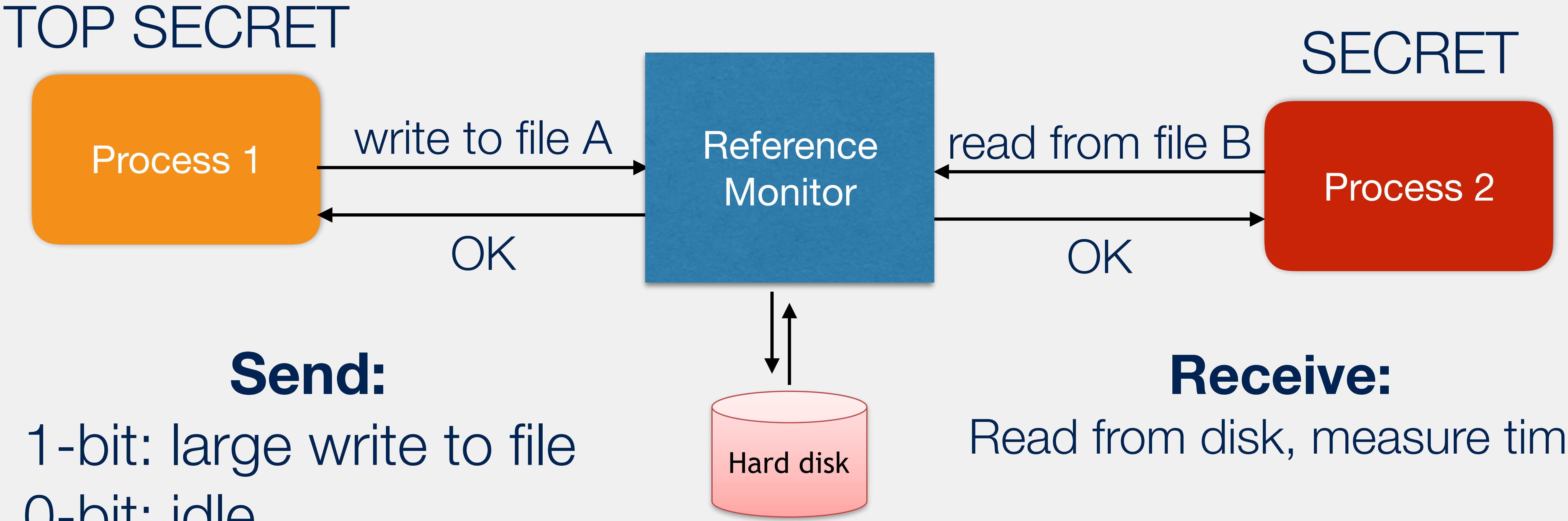
Reference Monitor

Process 2



red team

/ Karger and Schell, 1974



Send:

1-bit: large write to file

0-bit: idle

Receive:

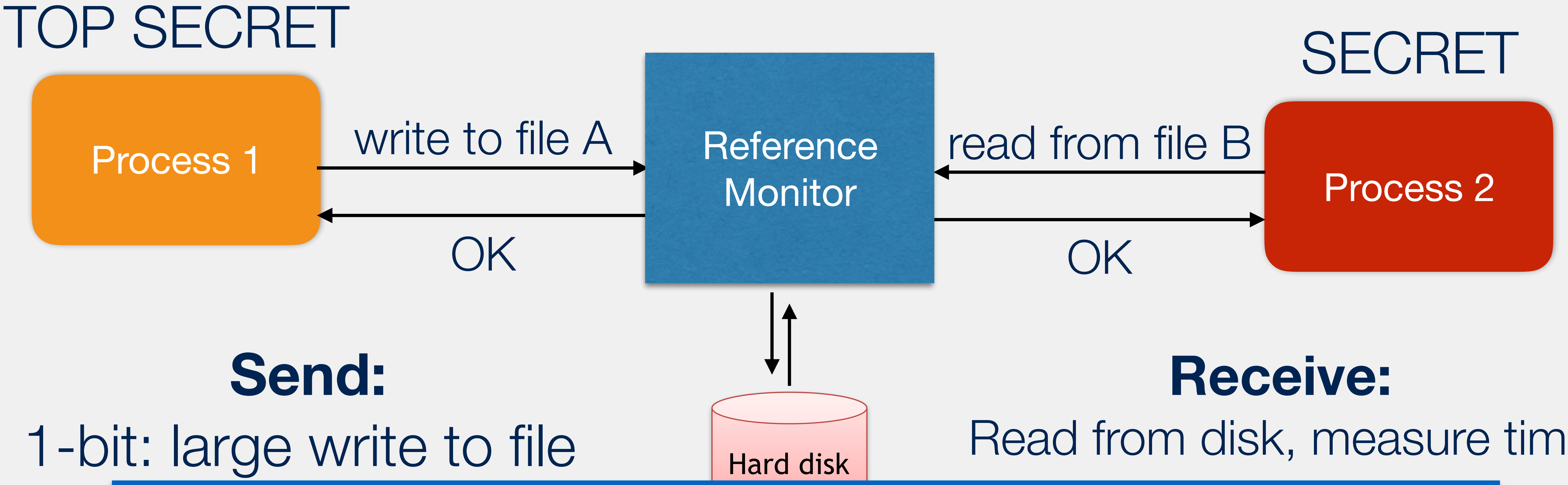
Read from disk, measure time

longer read time = 1-bit

shorter read time = 0-bit

red team

/ Karger and Schell, 1974



Send:

1-bit: large write to file

0-b

Receive:

Read from disk, measure time

bit

bit

Covert channel: circumvents reference monitor and security policy

access control

galapagos-05.cs.wisc.edu

```
/home/ace
  /scripts
  /Pictures
  /upd-encryption
```

```
/home/rist
   /lectures
   /projects
   /gitbucket
```

```
/home/sscott
  /Projects
  /latex
  /rust
```

```
/etc/nginx
  web-server-private-key.pem
```

access control

Permitted operations

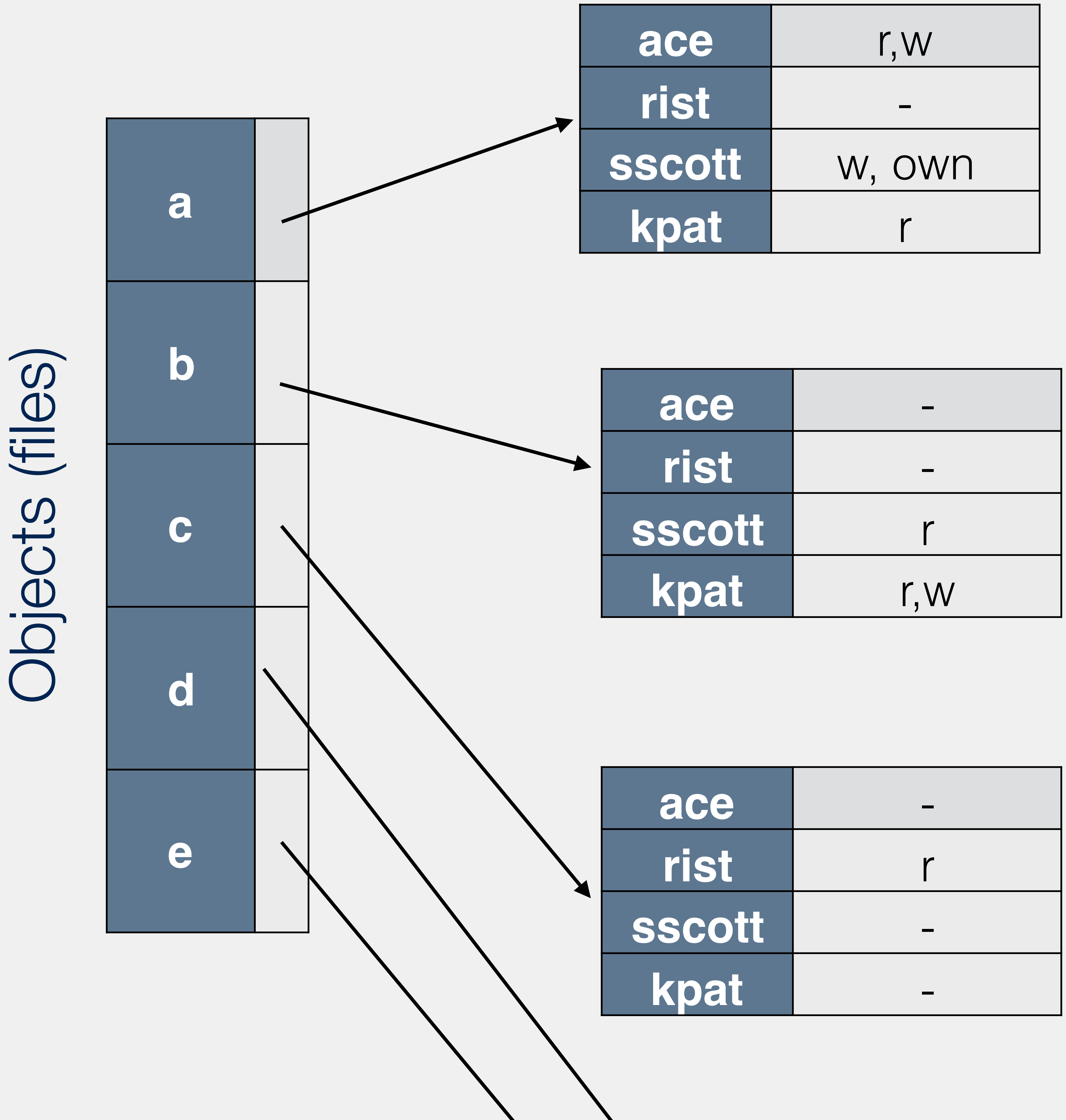
Objects (files)

Subjects (users)

	a	b	c	d	e
ace	r,w	-	r,w, own	-	r
rist	-	-	r	r	r,w
sscott	w, own	r	r	-	-
kpat	r	r,w	r,w	-	r

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

access control list



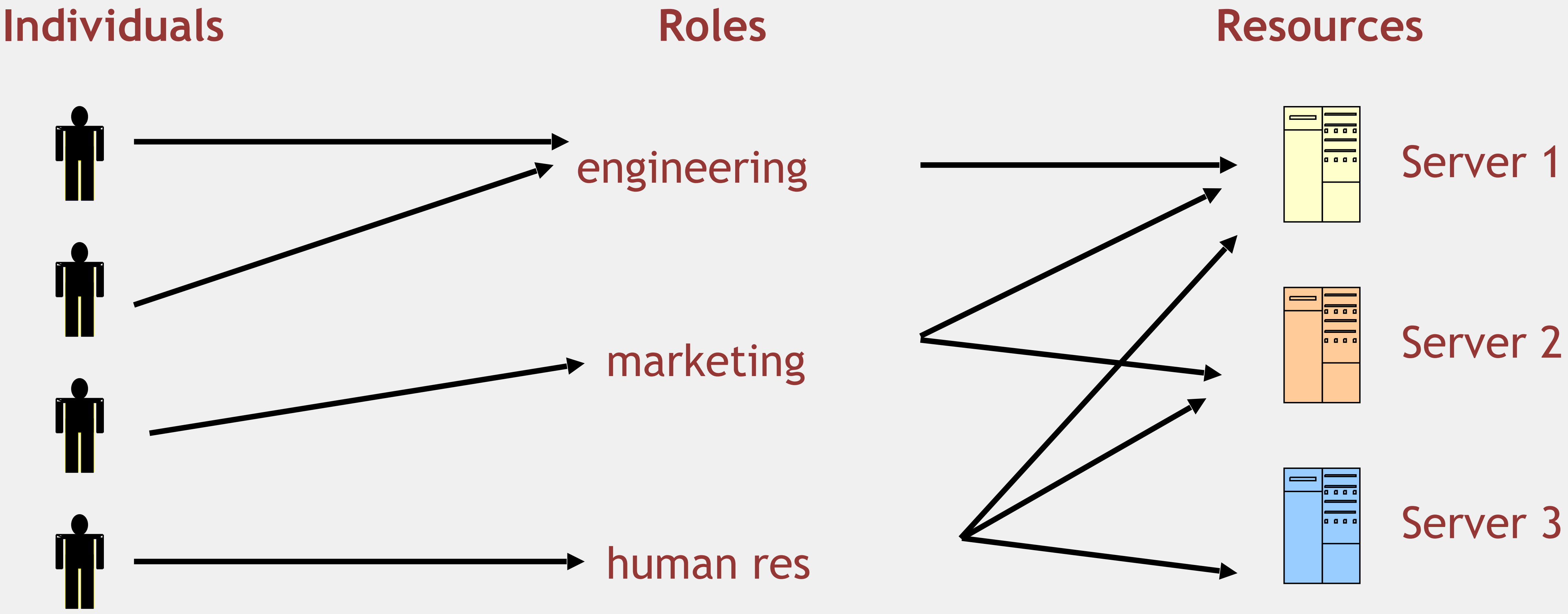
ace	r,w
rist	-
sscott	w, own
kpat	r

ace	-
rist	-
sscott	r
kpat	r,w

ace	-
rist	r
sscott	-
kpat	-

roles

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



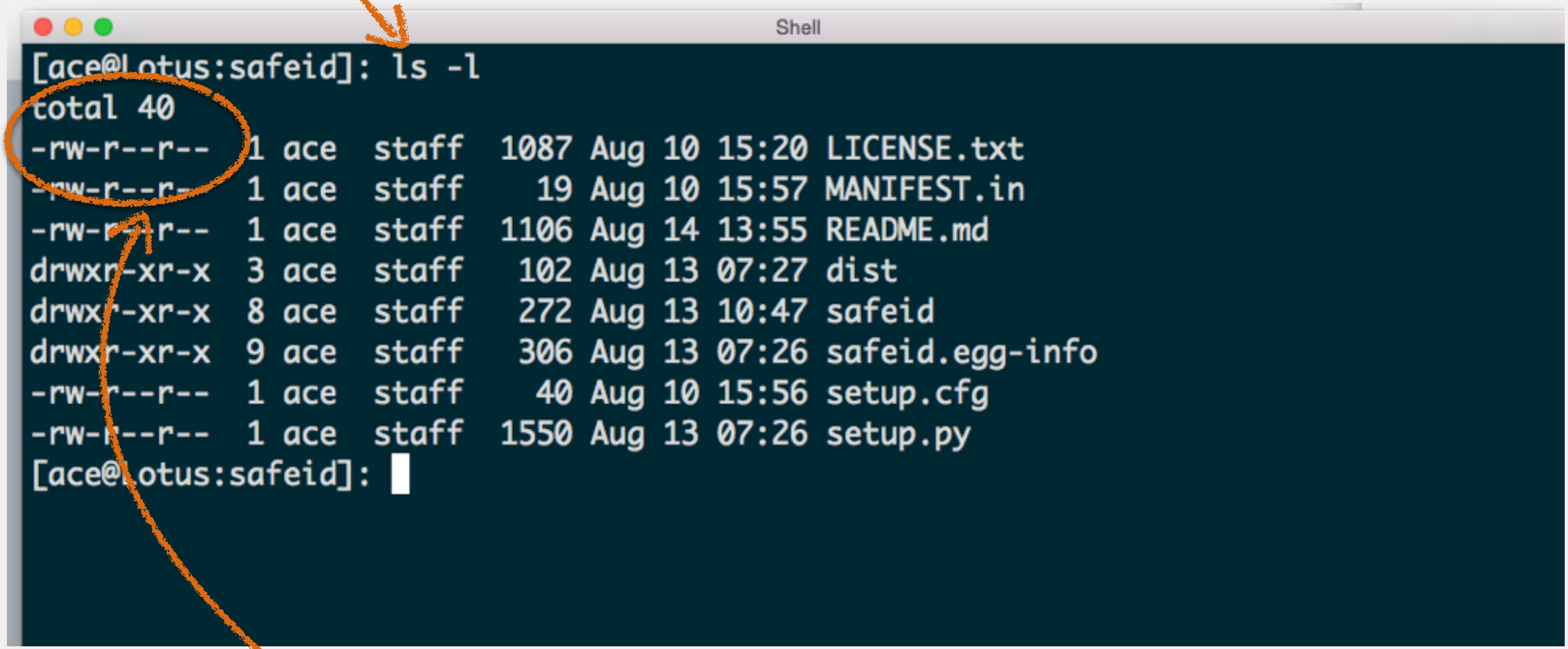
Advantages:

/ many users, few roles

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

unix access control

view file permissions



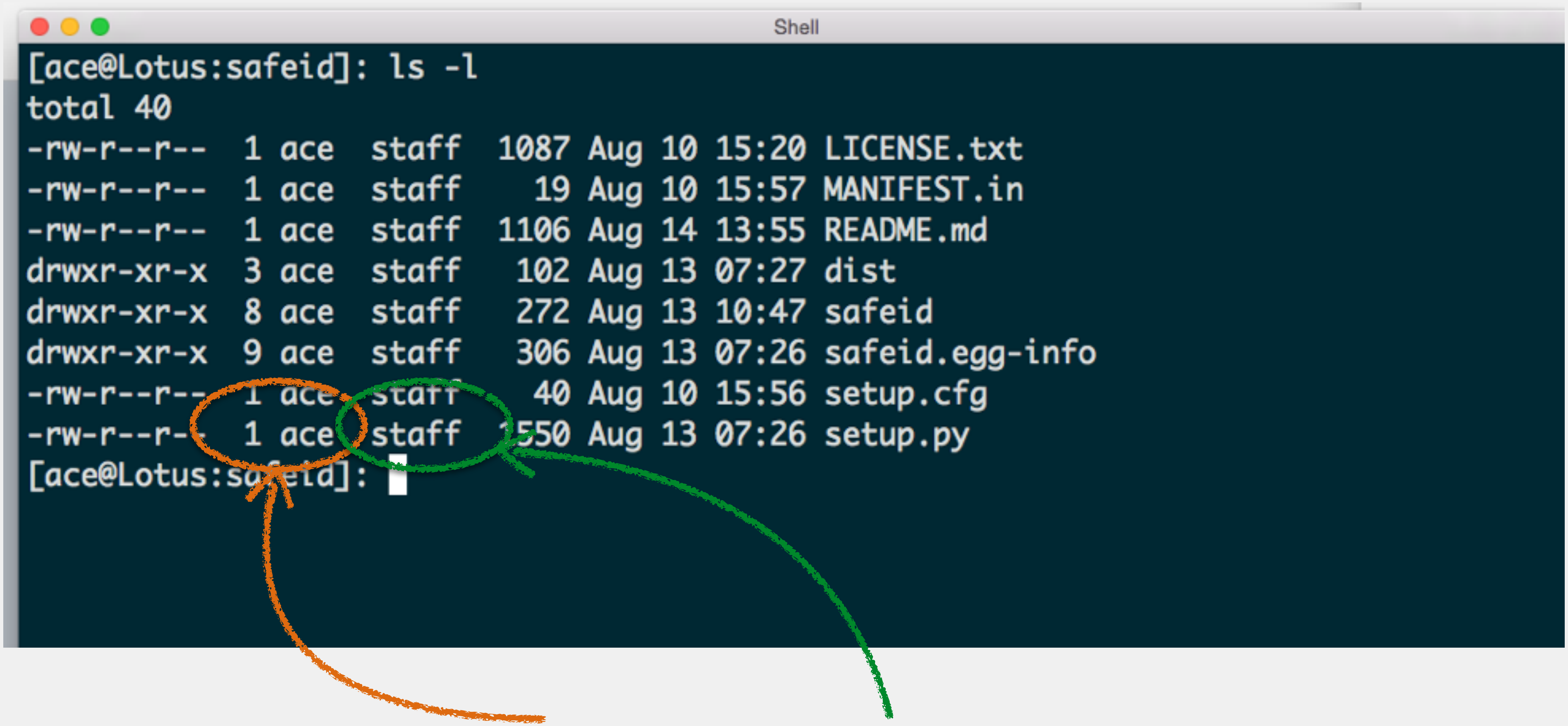
```
[ace@Lotus:safeid]: ls -l
total 40
-rw-r--r--  1 ace  staff  1087 Aug 10 15:20 LICENSE.txt
-rw-r--r--  1 ace  staff    19 Aug 10 15:57 MANIFEST.in
-rw-r--r--  1 ace  staff  1106 Aug 14 13:55 README.md
drwxr-xr-x  3 ace  staff   102 Aug 13 07:27 dist
drwxr-xr-x  8 ace  staff   272 Aug 13 10:47 safeid
drwxr-xr-x  9 ace  staff   306 Aug 13 07:26 safeid.egg-info
-rw-r--r--  1 ace  staff    40 Aug 10 15:56 setup.cfg
-rw-r--r--  1 ace  staff  1550 Aug 13 07:26 setup.py
[ace@Lotus:safeid]: 
```

access control list

unix access control

- 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

unix file system

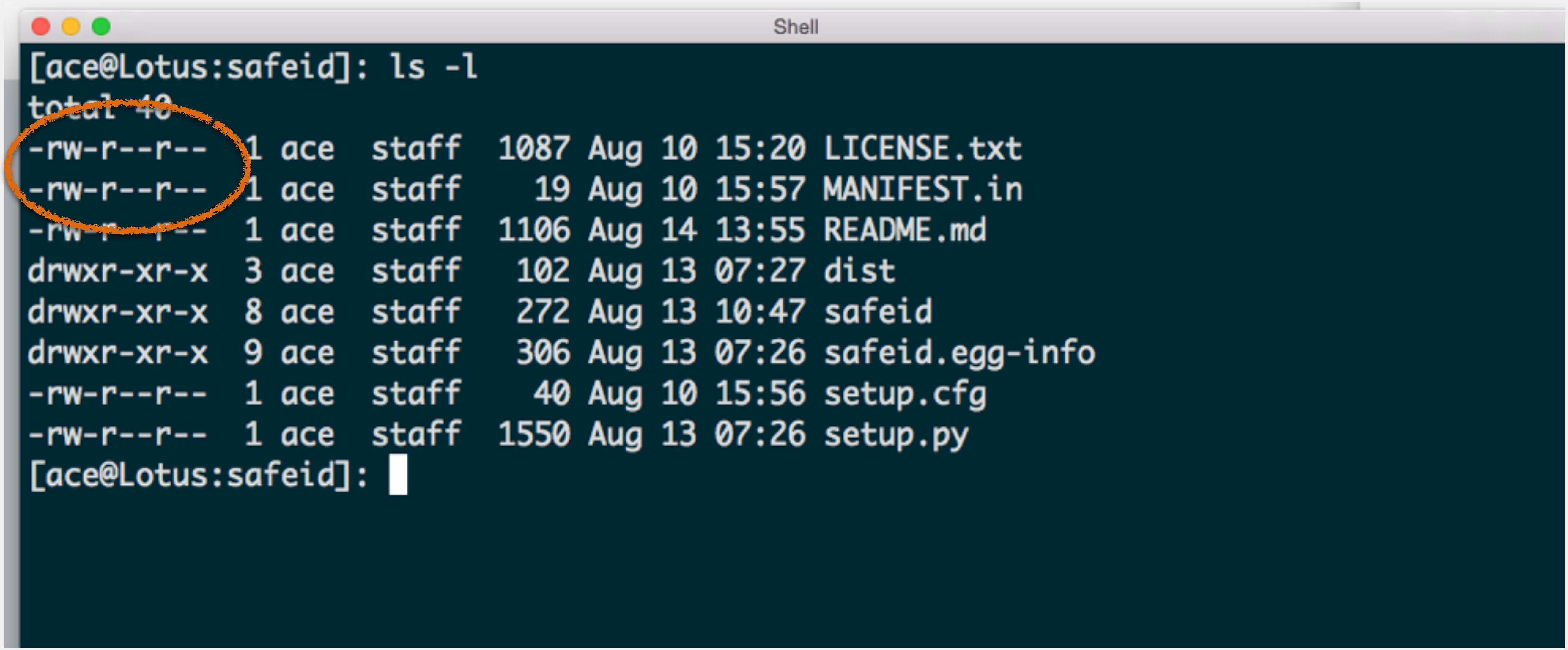


```
[ace@Lotus:safeid]: ls -l
total 40
-rw-r--r--  1 ace  staff  1087 Aug 10 15:20 LICENSE.txt
-rw-r--r--  1 ace  staff    19 Aug 10 15:57 MANIFEST.in
-rw-r--r--  1 ace  staff  1106 Aug 14 13:55 README.md
drwxr-xr-x  3 ace  staff   102 Aug 13 07:27 dist
drwxr-xr-x  8 ace  staff   272 Aug 13 10:47 safeid
drwxr-xr-x  9 ace  staff   306 Aug 13 07:26 safeid.egg-info
-rw-r--r--  1 ace  staff    40 Aug 10 15:56 setup.cfg
-rw-r--r--  1 ace  staff  1550 Aug 13 07:26 setup.py
[ace@Lotus:safeid]: 
```

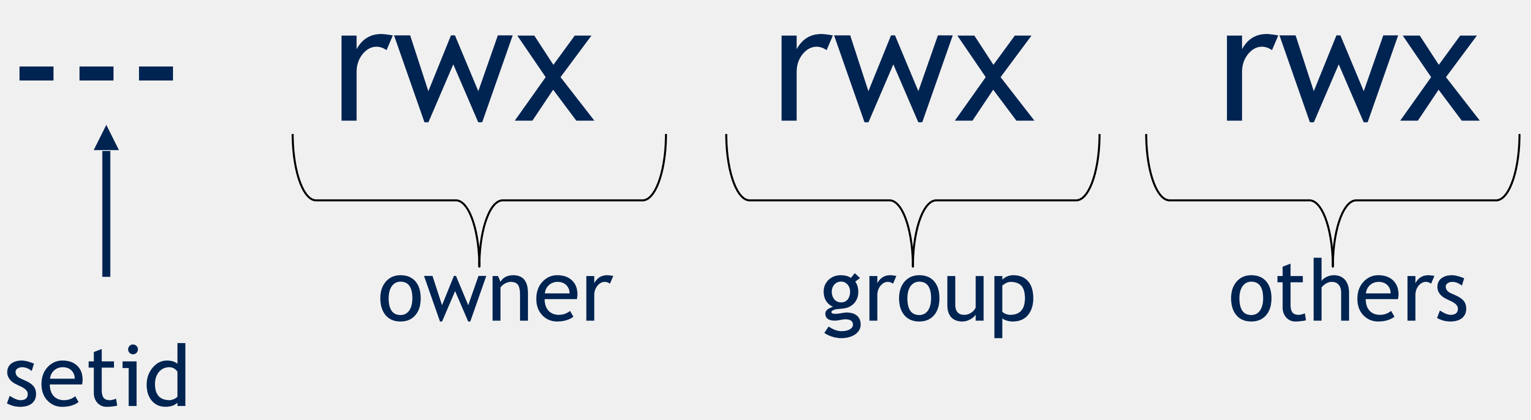
Each file assigned: owner and a group

Basic operations: read, write, execute

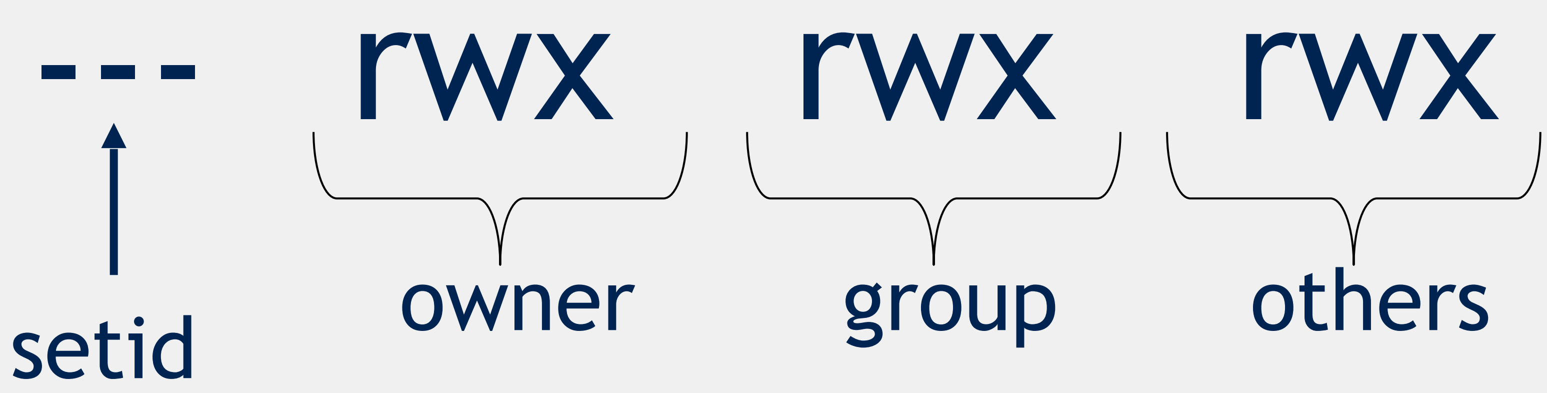
unix acl



```
[ace@Lotus:safeid]: ls -l
total 40
-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
-rw-r--r--  1 ace  staff  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
drwxr-xr-x  9 ace  staff   306 Aug 13 07:26 safeid.egg-info
-rw-r--r--  1 ace  staff    40 Aug 10 15:56 setup.cfg
-rw-r--r--  1 ace  staff  1550 Aug 13 07:26 setup.py
[ace@Lotus:safeid]: 
```

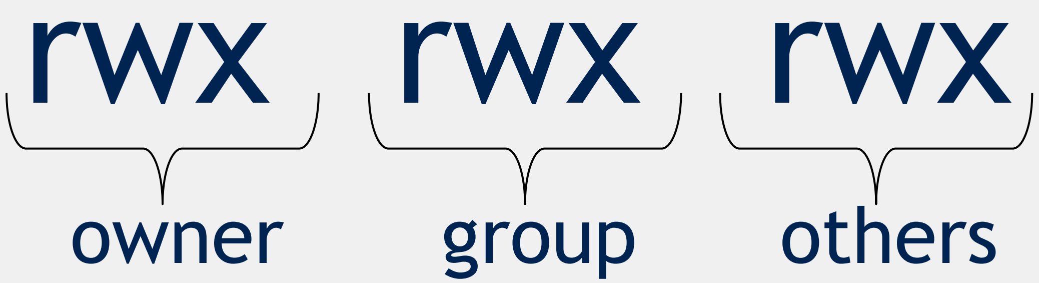


unix acls

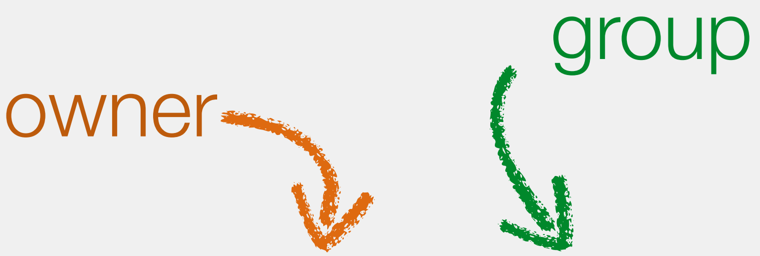


- Permissions set by owner (or root)
- Determining if an action is permitted:
  - if uid == 0 (root): allow anything
  - else if uid == *owner*: use *owner* permissions
  - else if uid in *group*: use *group* permissions
  - else: use *other* permissions
- Only owner, root can change permissions
  - This privilege cannot be delegated or shared
- Setid bits – Discuss in a few slides

exercise


rwx rwx rwx

owner group others


group

owner

```
-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
```

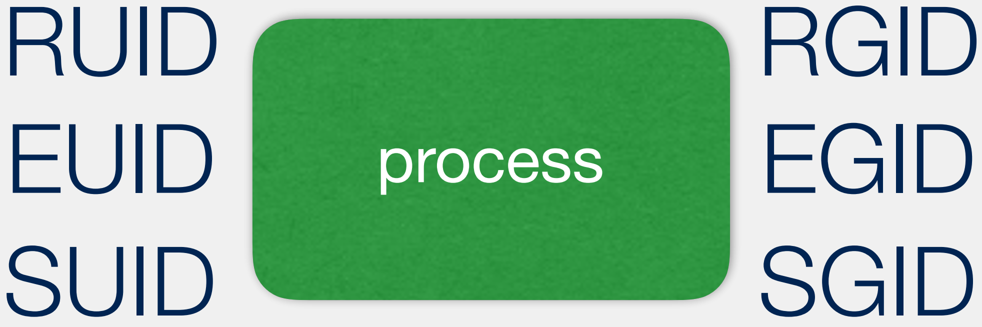
```
staff:*:29:ace,sscott,kpat,rist
web:*:31:ace,kpat,rist
dev:*:32:ace,sscott,pbriggs
```

Can sscott read the file README.md?

Can ace write to setup.cfg?

Which users can append to deploy.log?

process ids



Real User ID

/ same as the UID of parent

/ indicates who started this process

Effective User ID

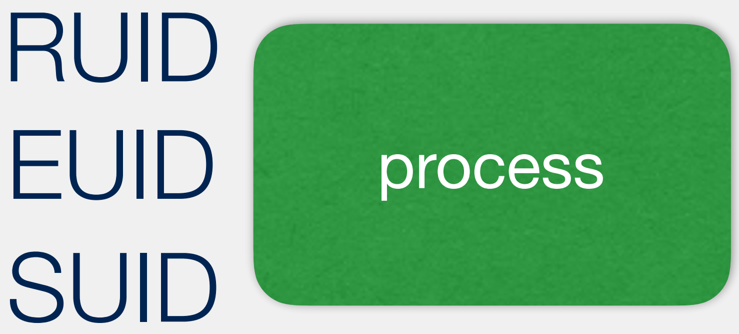
/ current permissions for this process

Saved User ID

/ previous EUID so that it can be restored

Also: Real Group ID, Effective Group ID,

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?

- Many UNIX systems store passwords in the file /etc/shadow

- Who should be able to read this file? Write this file?

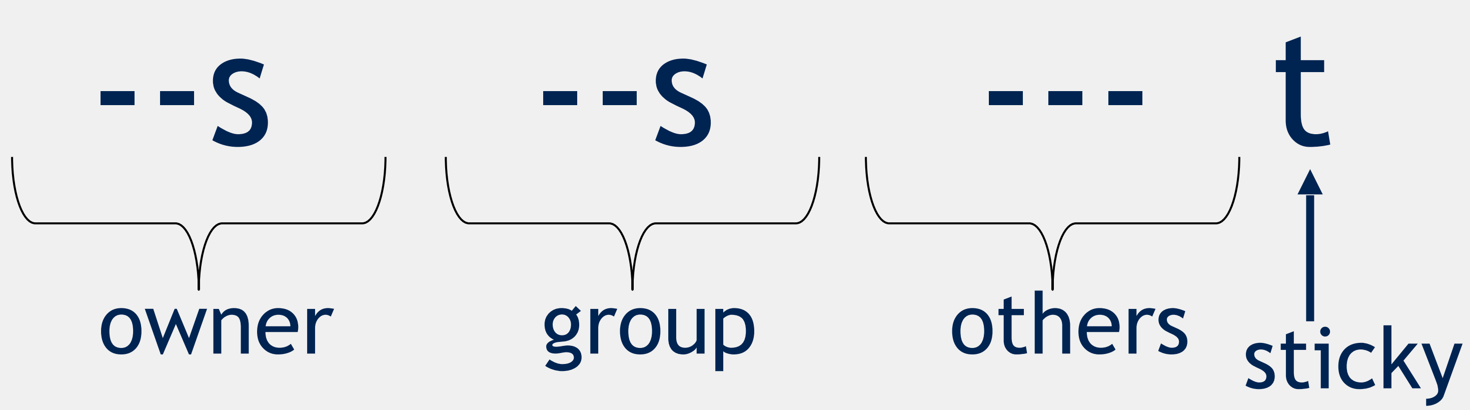
- Users change passwords using /usr/bin/passwd

- What EUID does this process run as?

- How can it write updates to the password file?

setid bits

setid



--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

```
[ace:/usr/bin/]: ls -l
...
-rwsr-xr-x 1 root root 47032 Feb 17  2014 passwd
...
-rwxr-sr-x 1 root  tty 19024 Feb 12  2015 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);
```

better

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

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

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

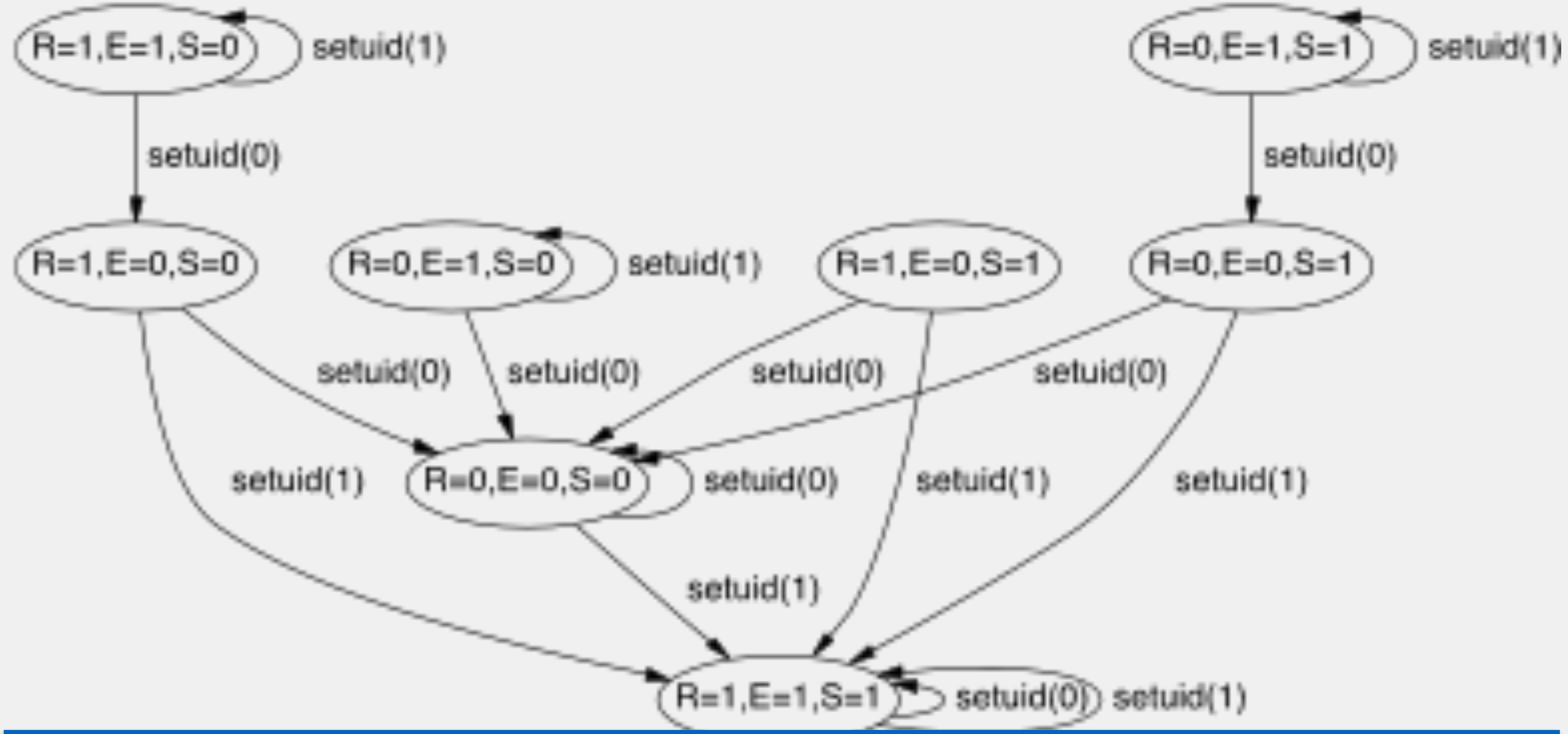
```
19   file = open("/tmp/myfile", "r");
        error: errno=13 (Permission denied).
```

What security design principle?

> Least privilege

setid

/ In practice, setid is even more complicated



Q: Violates which secure design principles?

[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