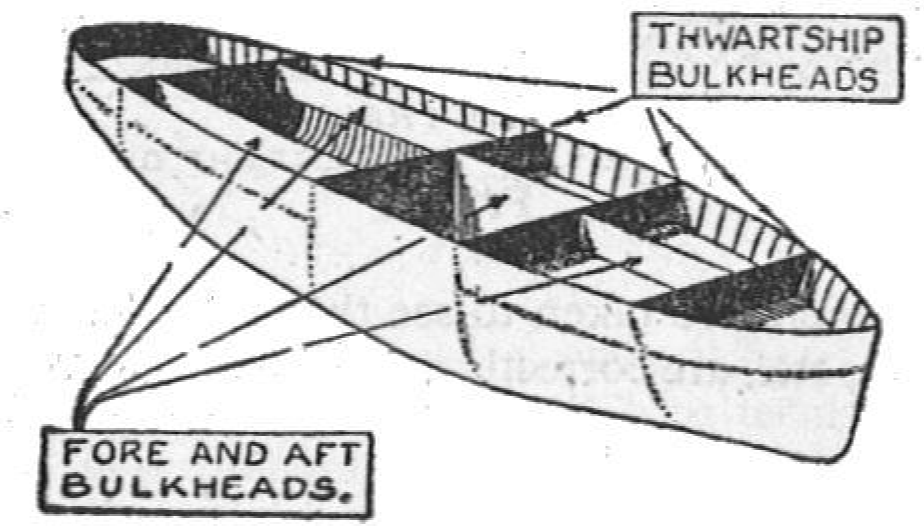
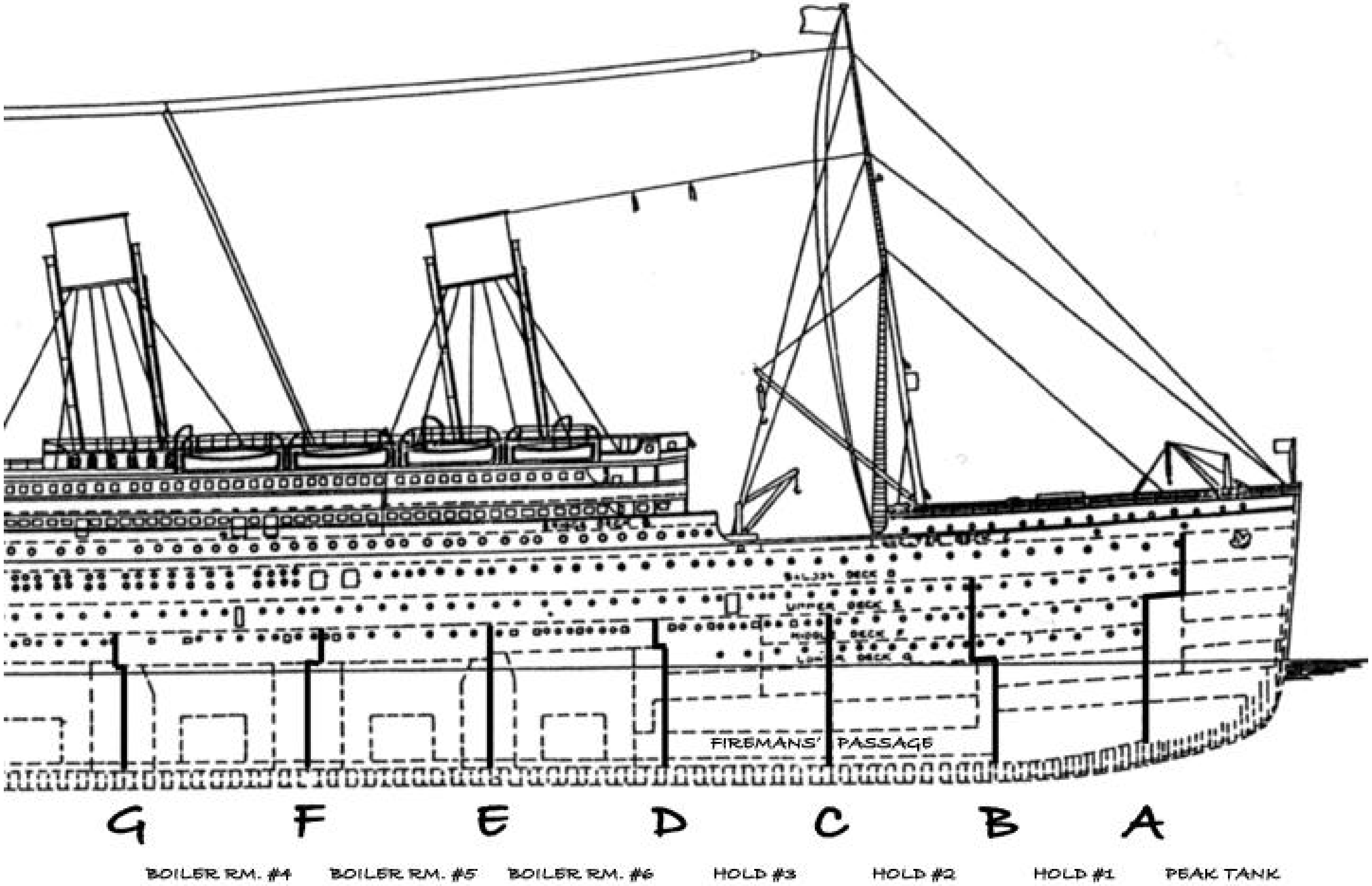
Secure Architecture Principles

Slides credit: John Mitchell

Basic idea: Isolation



A Seaman's Pocket-Book, 1943 (public domain)



Bulkheads & Compartments in the Bow Section

http://staff.imsa.edu/~esmith/treasurefleet/treasurefleet/watertight_compartments.htm

Dawn Song

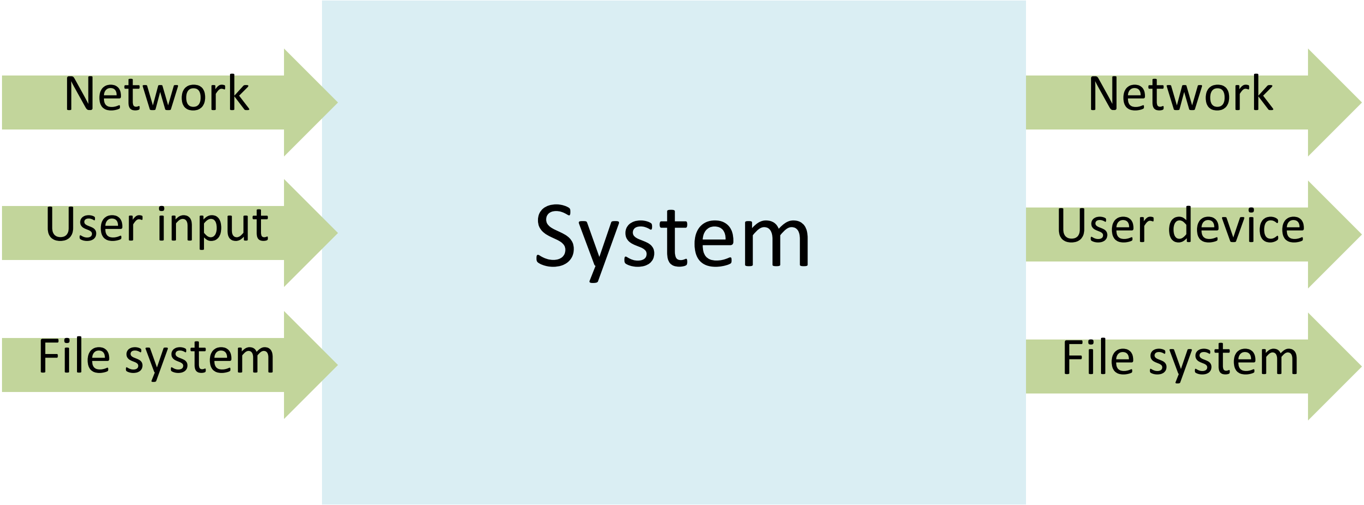
Principles of Secure Design

- Compartmentalization
  - Isolation
  - Principle of least privilege
- Defense in depth
  - Use more than one security mechanism
  - Secure the weakest link
  - Fail securely
- Keep it simple

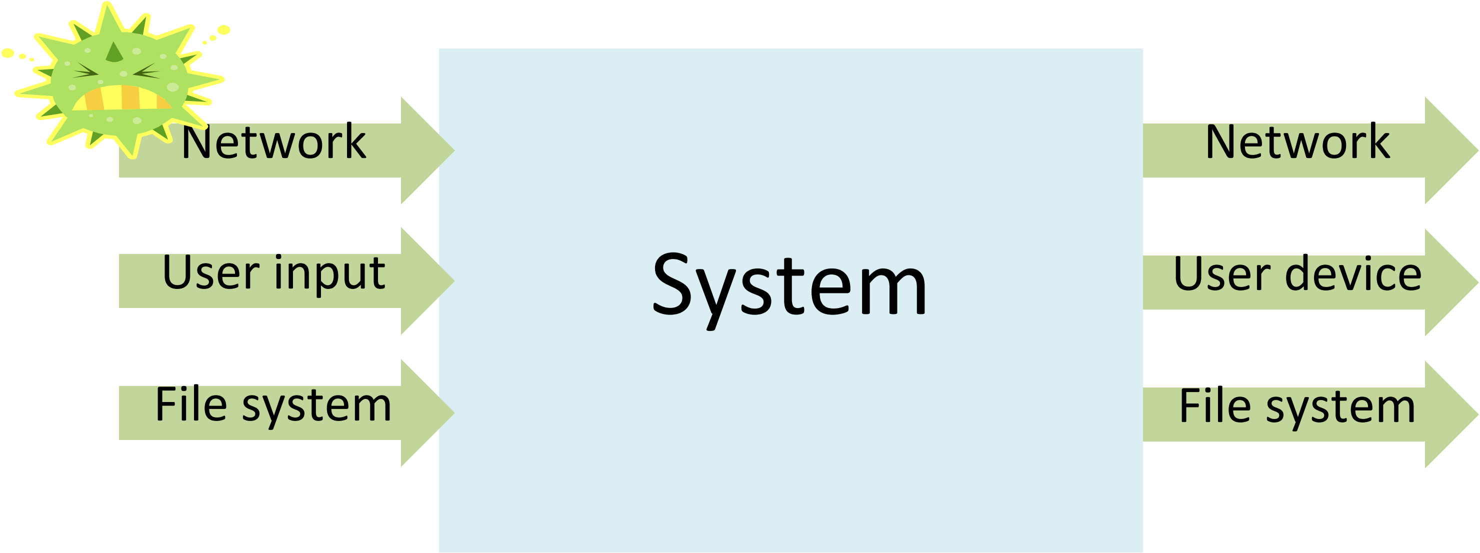
Principle of Least Privilege

- Privilege
  - Ability to access or modify a resource
- Principle of Least Privilege
  - A system module should only have the minimal privileges needed for intended purposes
- Requires compartmentalization and isolation
  - Separate the system into independent modules
  - Limit interaction between modules

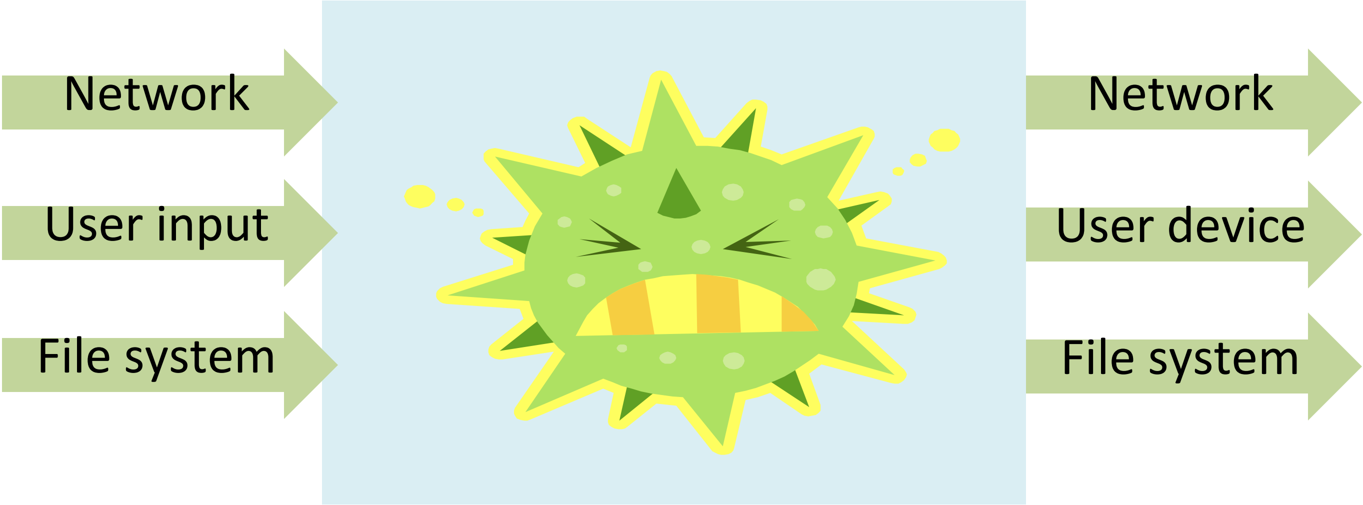
Monolithic design



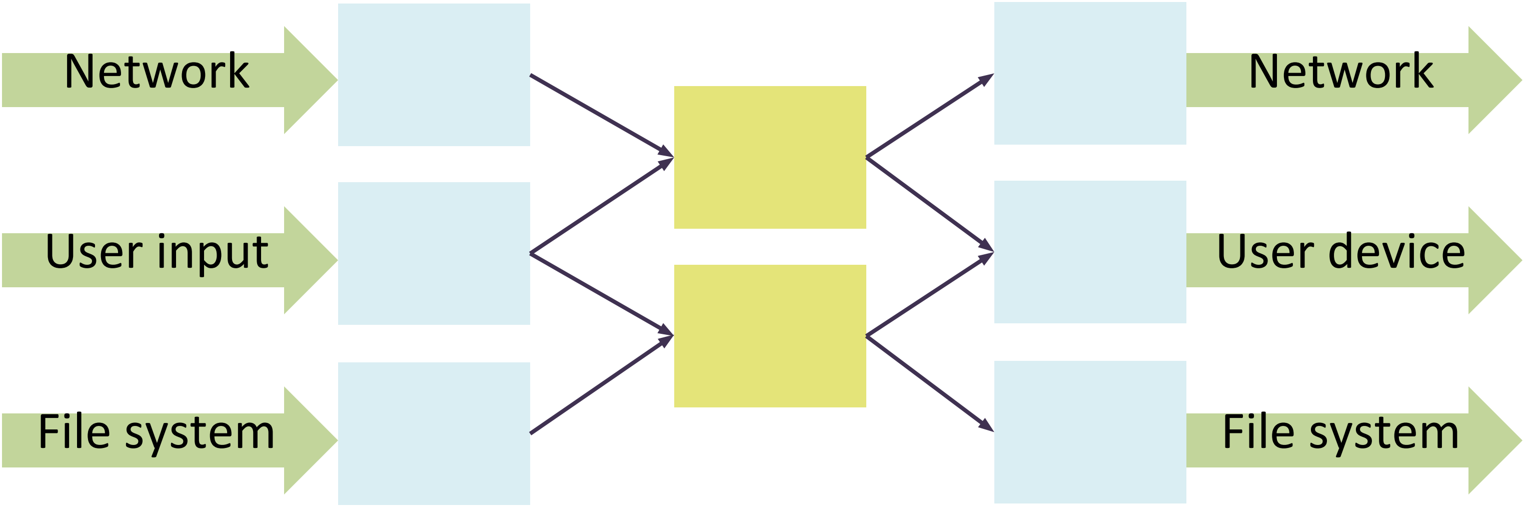
Monolithic design



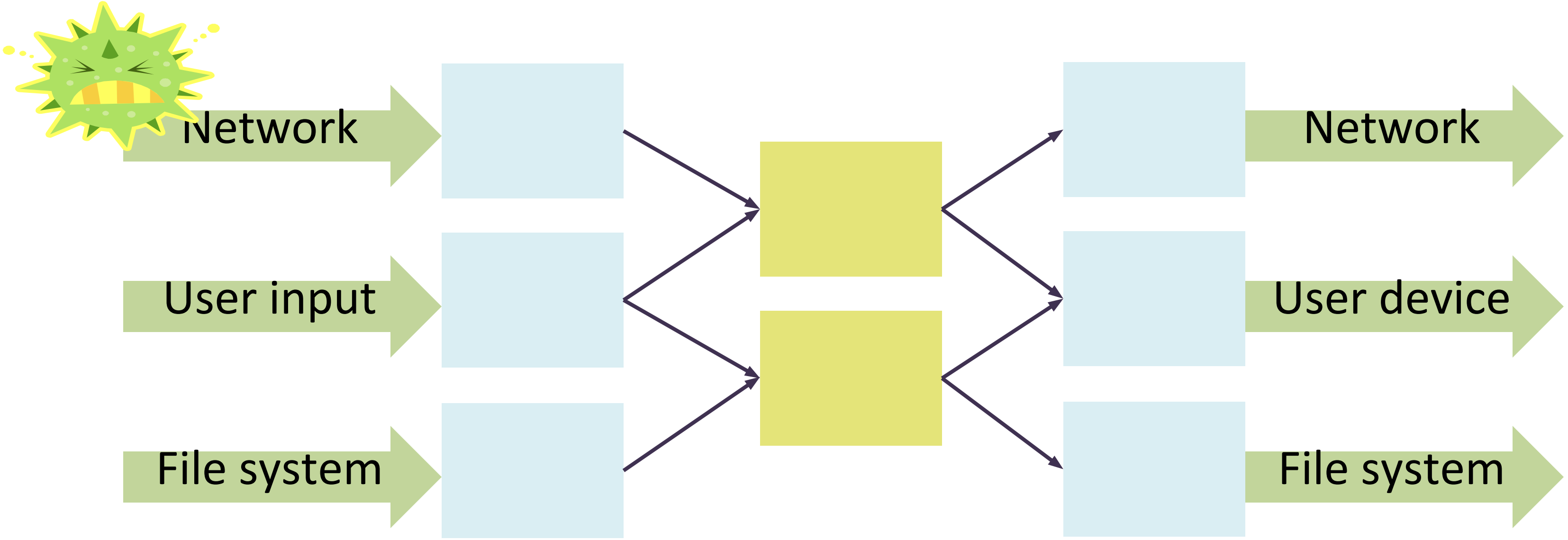
Monolithic design



Component design

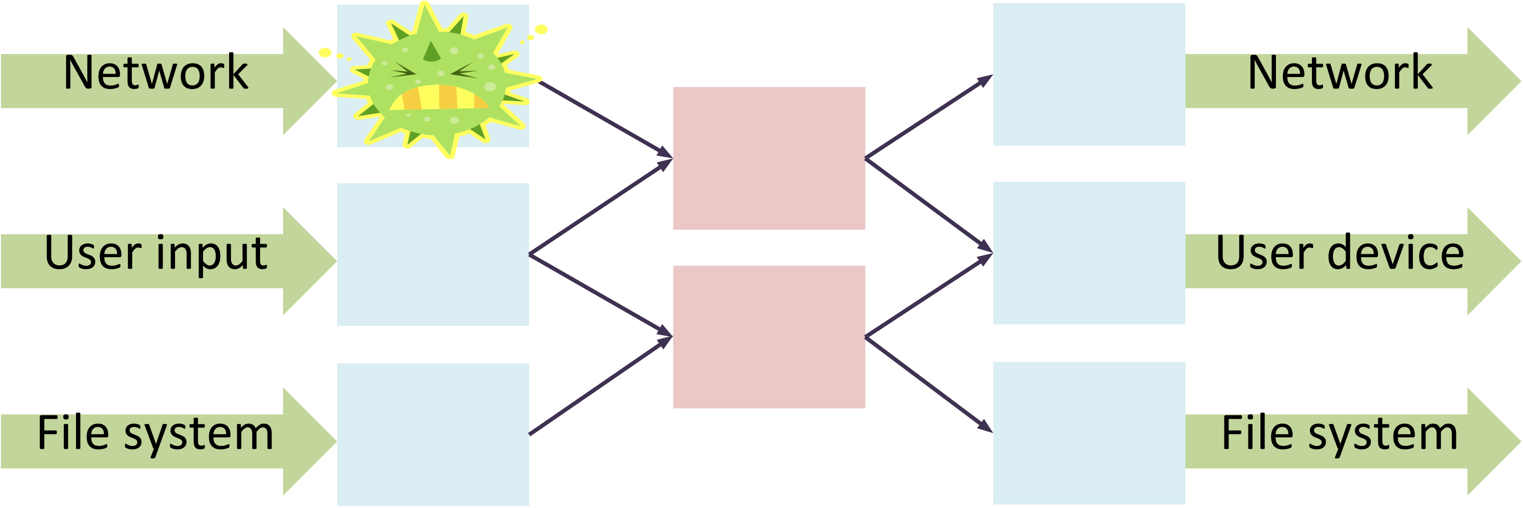


Component design



Dawn Song

Component design



Which of these are privileges that allow one component to affect another component or system?

Send a message on the network

Add two numbers stored in two local variables

Call a function defined in the same component

Call a function defined in a different component

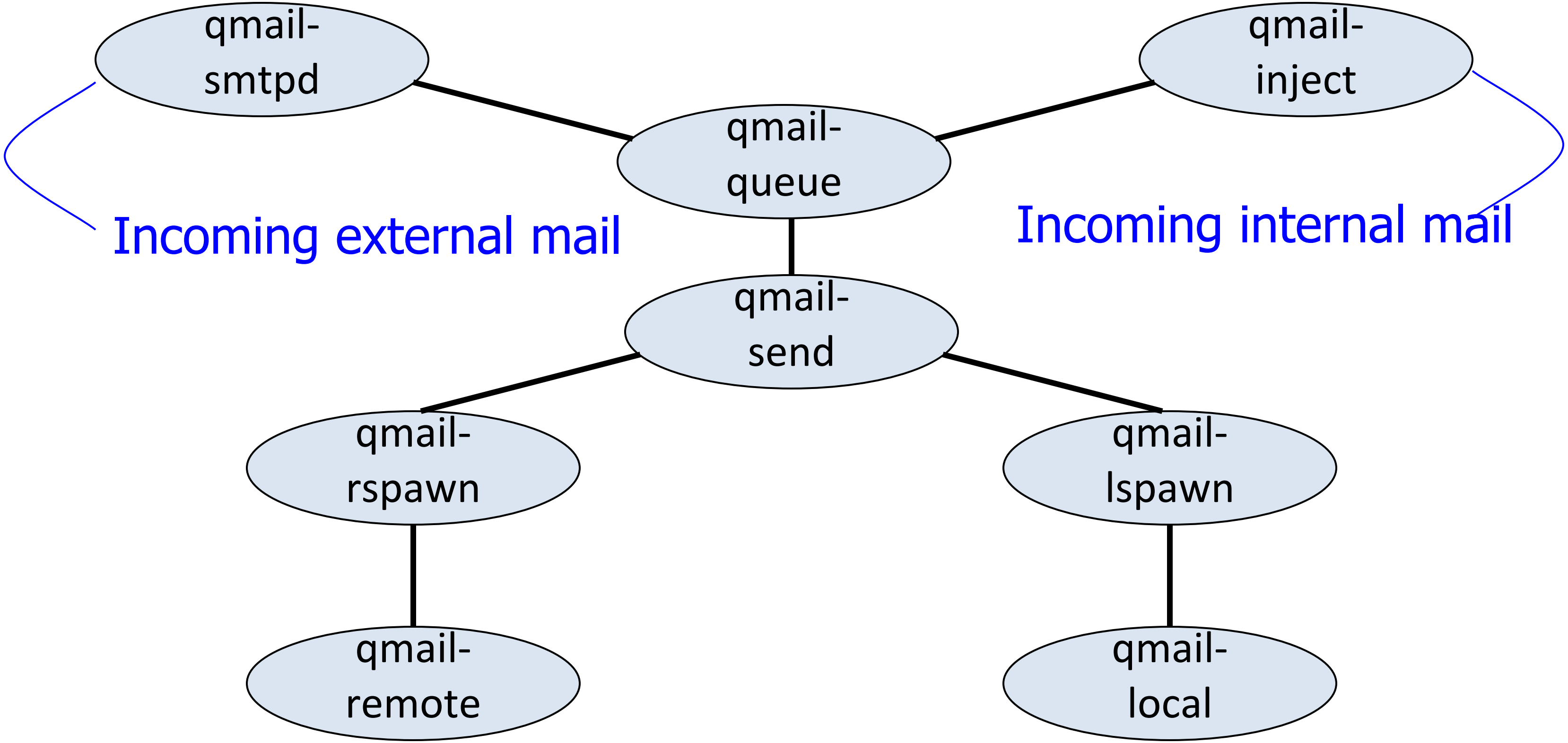
Example: Mail Agents

- Requirements
  - Receive and send email over external network
  - Place incoming email into local user inbox files
- Sendmail
  - Traditional Unix
  - Monolithic design
  - Historical source of many vulnerabilities
- Qmail
  - Component design

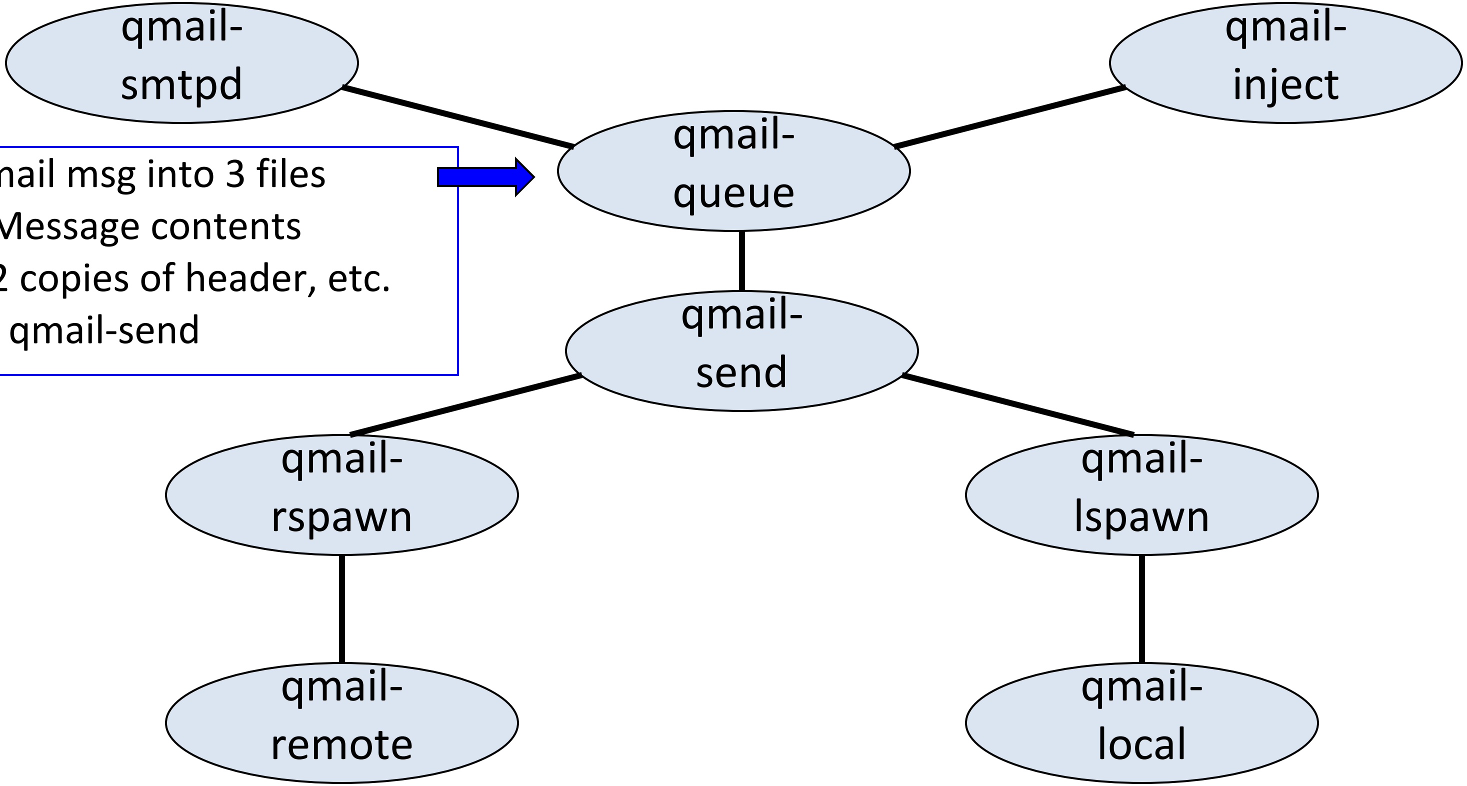
Qmail design

- Isolation
  - Separate modules run as separate “users”
  - Each user only has access to specific resources
- Least privilege
  - Each module has least privileges necessary
  - Only one “setuid” program
    - setuid allows a program to run as different users
  - Only one “root” program
    - root program has all privileges

Structure of qmail



Structure of qmail

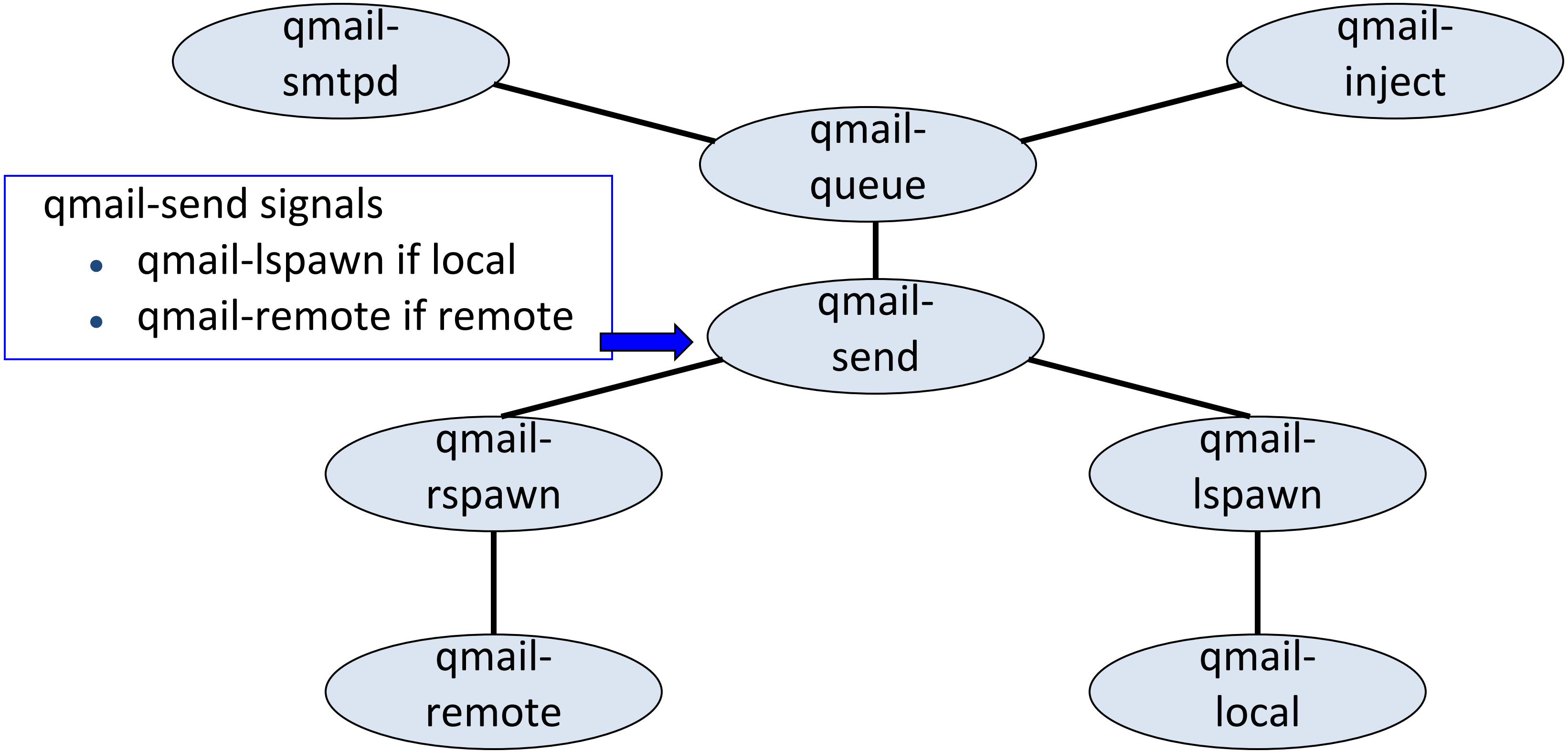


Splits mail msg into 3 files

- Message contents
- 2 copies of header, etc.

Signals qmail-send

Structure of qmail



qmail-smtpd

qmail-inject

qmail-queue

qmail-send

qmail-rspawn

qmail-lspawn

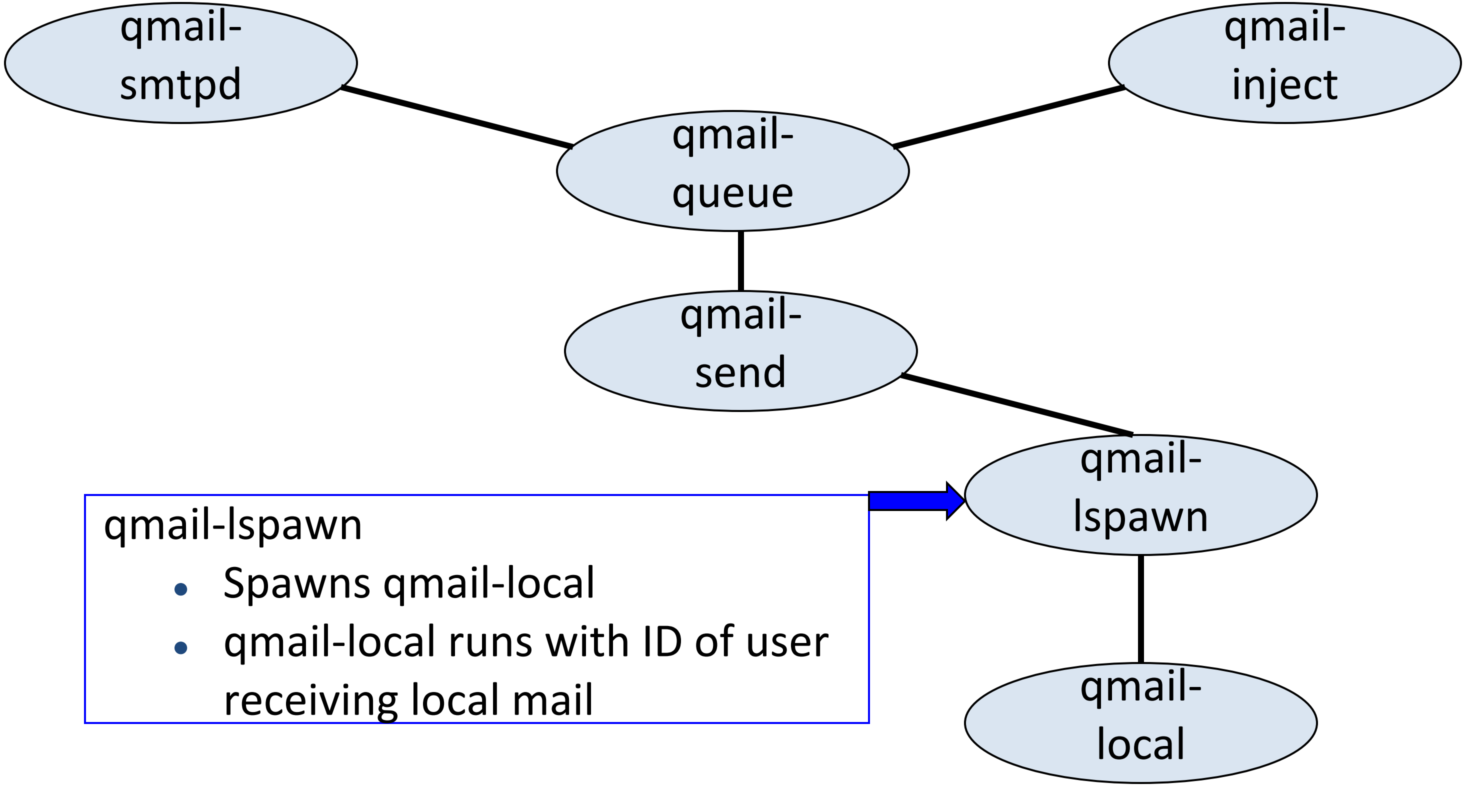
qmail-remote

qmail-local

qmail-send signals

- qmail-lspawn if local
- qmail-remote if remote

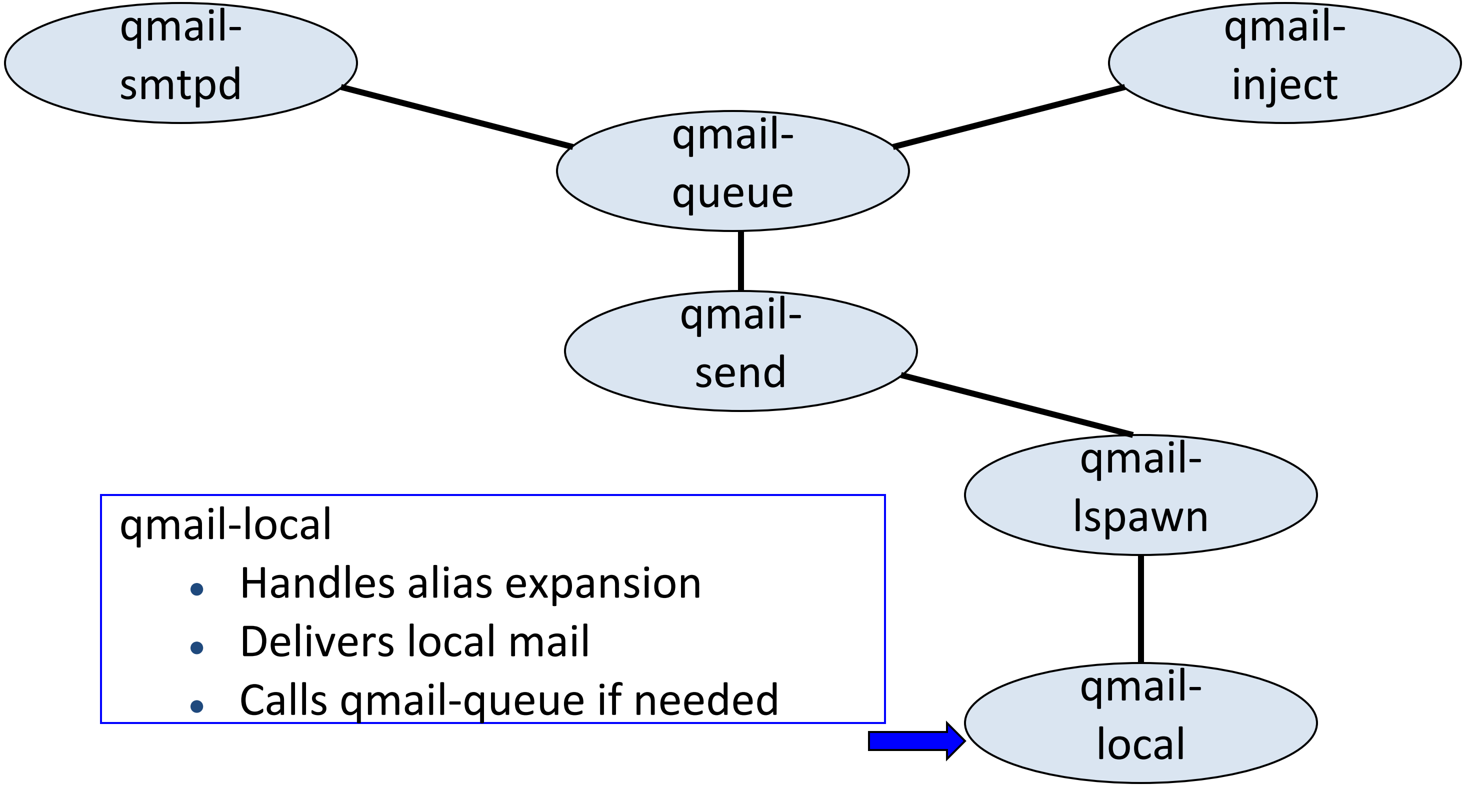
Structure of qmail



qmail-lspawn

- Spawns qmail-local
- qmail-local runs with ID of user receiving local mail

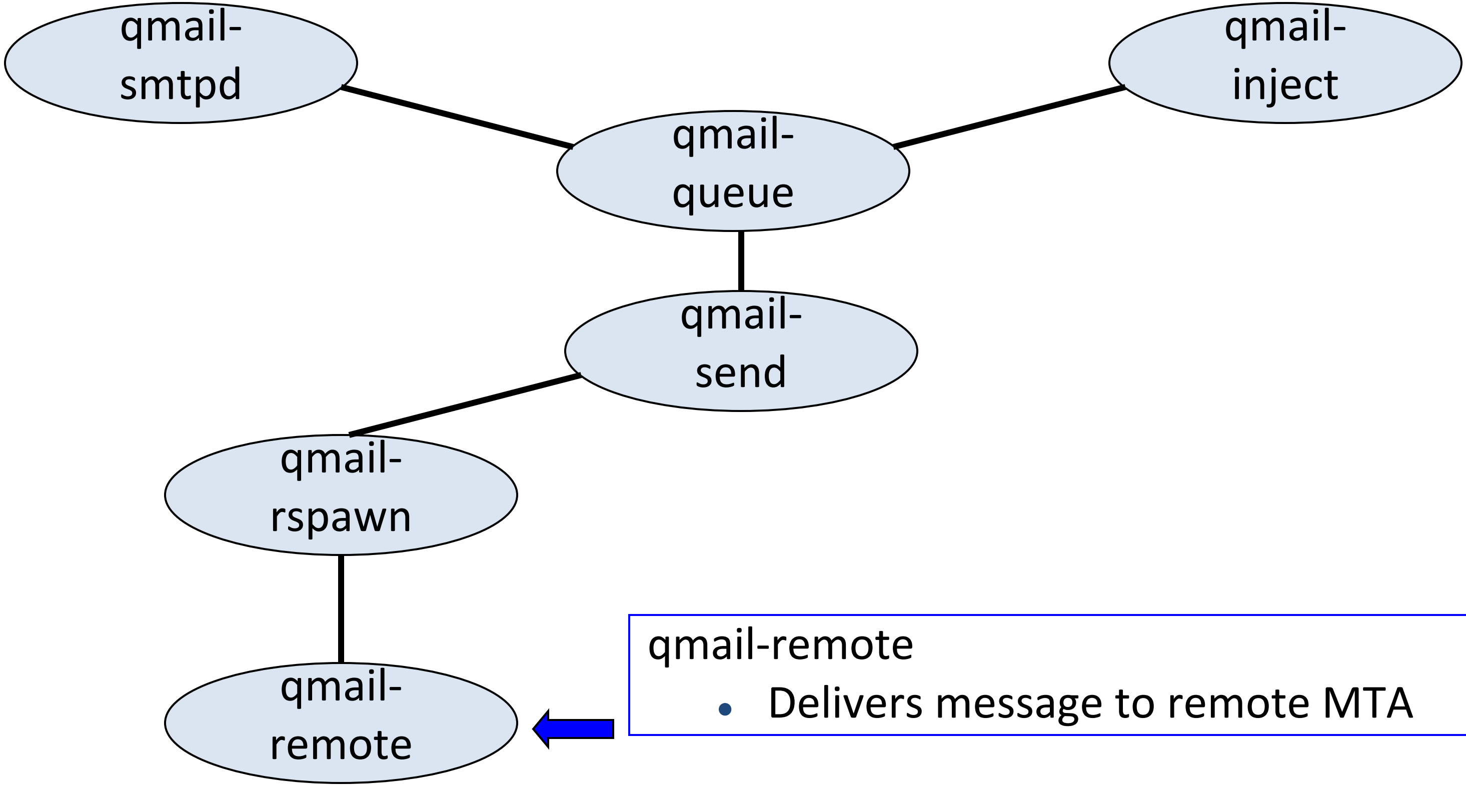
Structure of qmail



qmail-local

- Handles alias expansion
- Delivers local mail
- Calls qmail-queue if needed

Structure of qmail

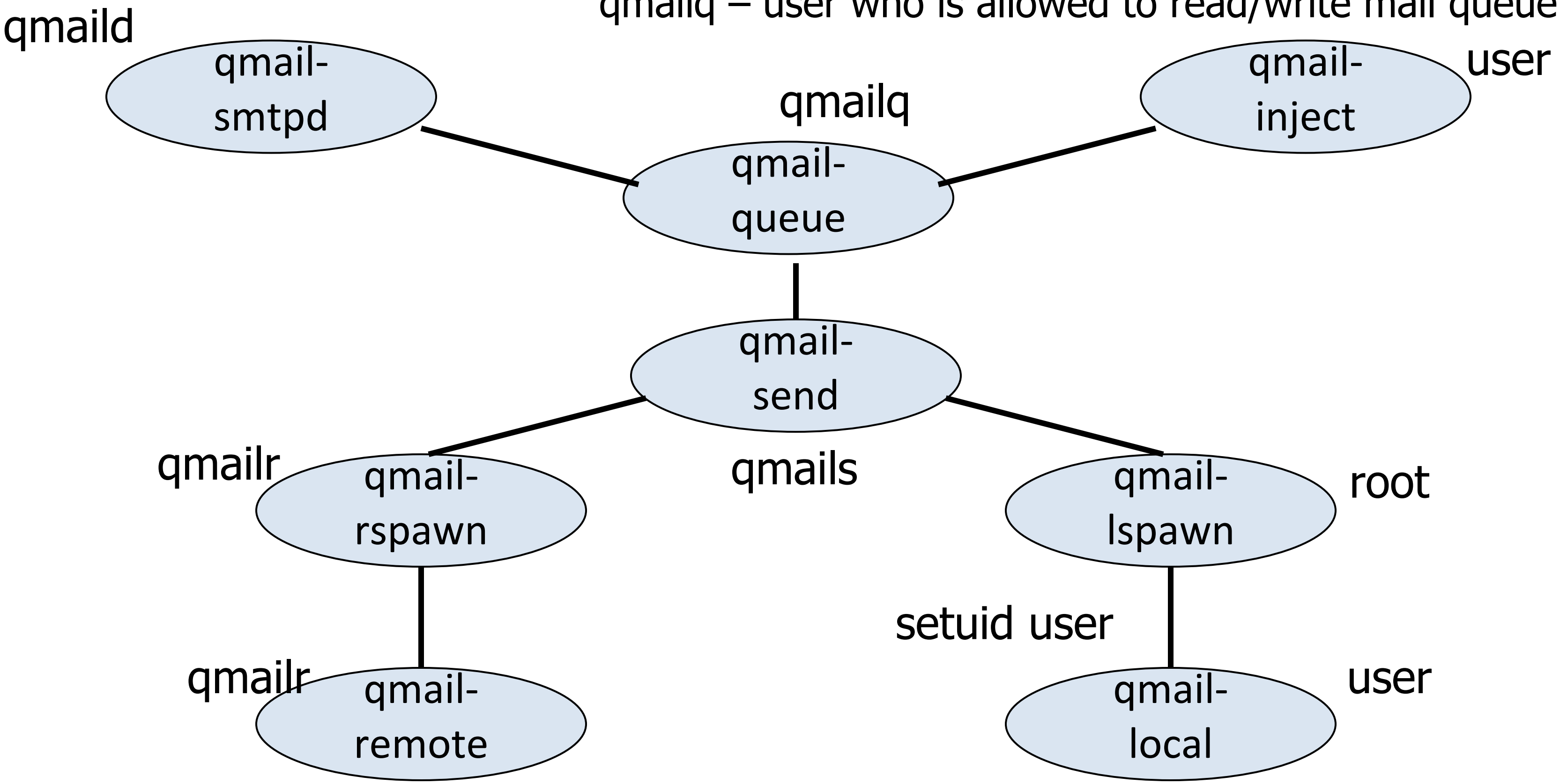


qmail-remote

- Delivers message to remote MTA

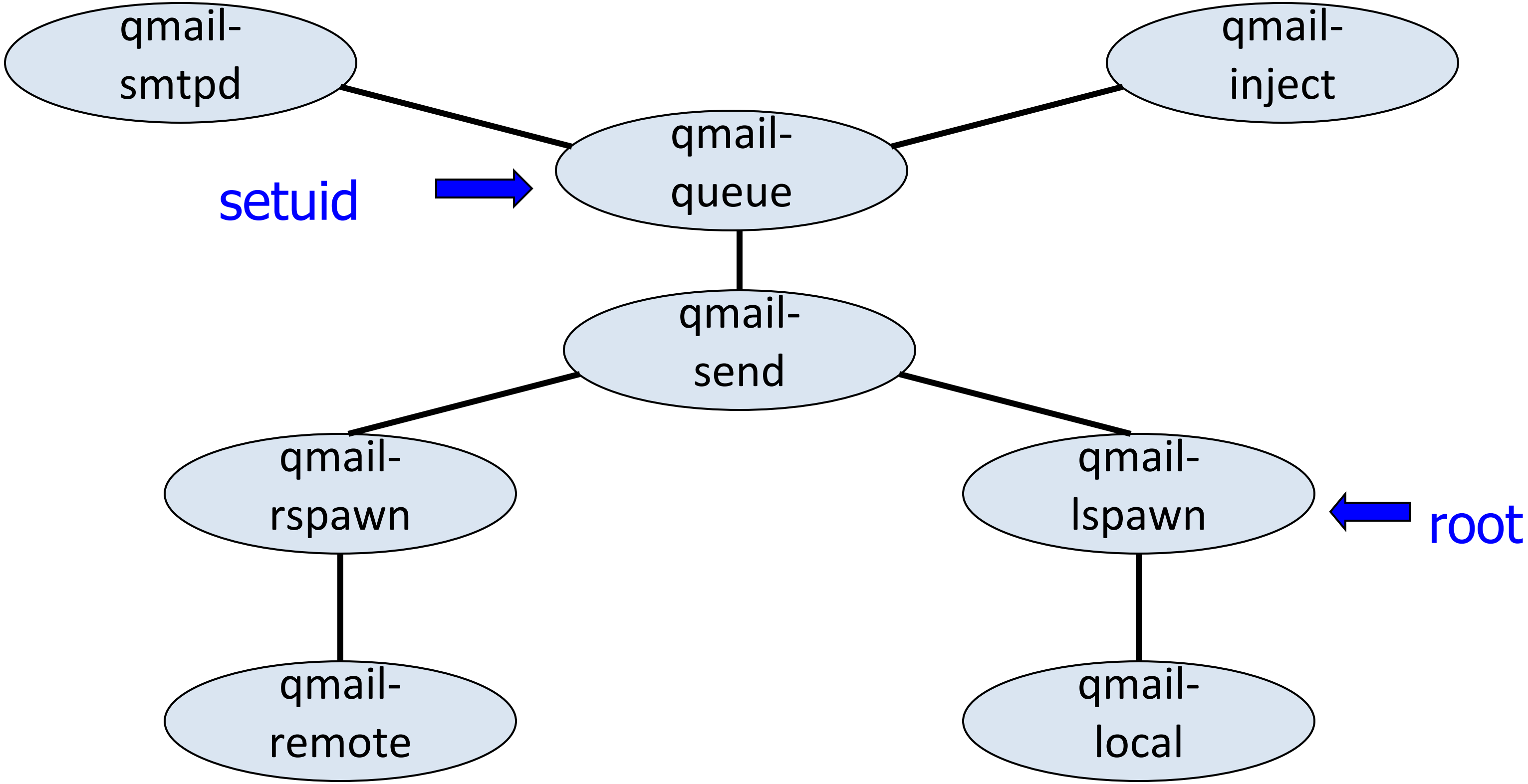
Isolation by Unix UIDs

qmailq – user who is allowed to read/write mail queue



Dawn Song

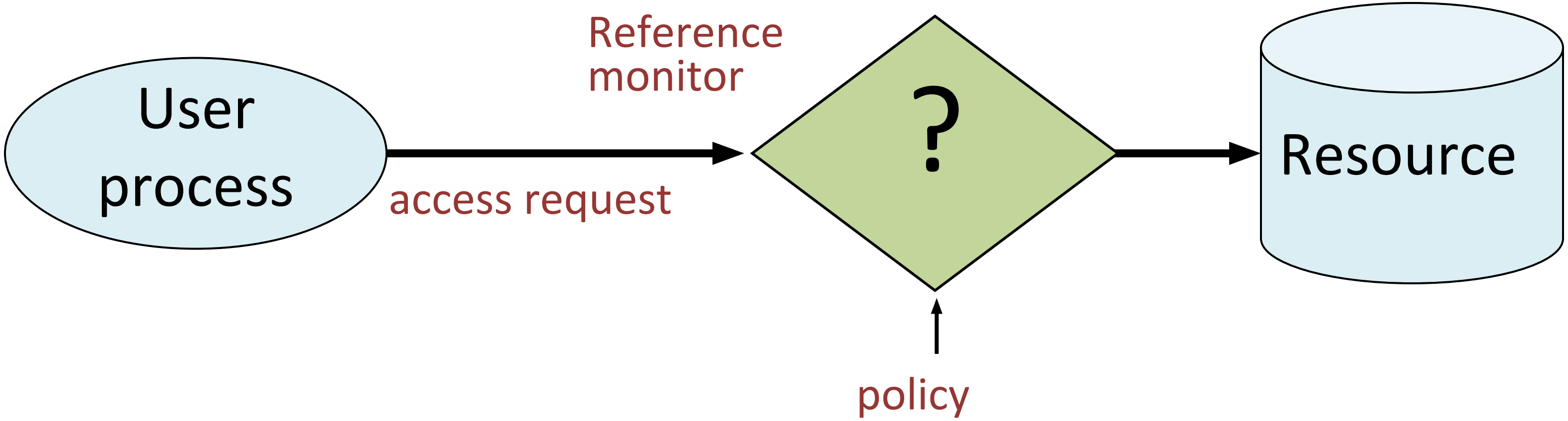
Least privilege



Access Control & Capabilities

Access control

- Assumptions
  - System knows who the user is
    - Authentication via name and password, other credential
  - Access requests pass through gatekeeper (reference monitor)
    - System must not allow monitor to be bypassed



Access control matrix [Lampson]

Objects

Subjects

	File 1	File 2	File 3	…	File n
User 1	read	write	-	-	read
User 2	write	write	write	-	-
User 3	-	-	-	read	read
…					
User m	read	write	read	write	read

Two implementation concepts

- Access control list (ACL)
  - Store column of matrix with the resource
- Capability
  - User holds a “ticket” for each resource
  - Two variations
    - store row of matrix with user, under OS control
    - unforgeable ticket in user space

	File 1	File 2	…
User 1	read	write	-
User 2	write	write	-
User 3	-	-	read
…			
User m	Read	write	write

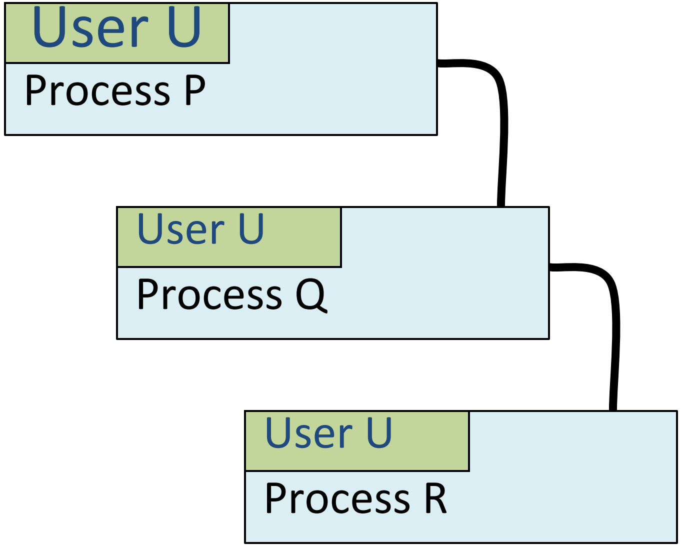
Access control lists are widely used, often with groups

Some aspects of capability concept are used in many systems

ACL vs Capabilities

- Access control list
  - Associate list with each object
  - Check user/group against list
  - Relies on authentication: need to know user
- Capabilities
  - Capability is unforgeable ticket
    - Random bit sequence, or managed by OS
    - Can be passed from one process to another
  - Reference monitor checks ticket
    - Does not need to know identify of user/process

ACL vs Capabilities



User U

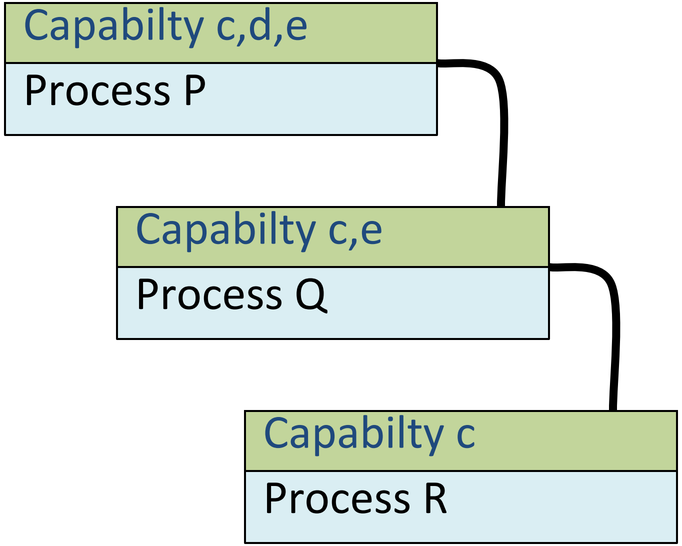
Process P

User U

Process Q

User U

Process R



Capabilty c,d,e

Process P

Capabilty c,e

Process Q

Capabilty c

Process R

ACL vs Capabilities

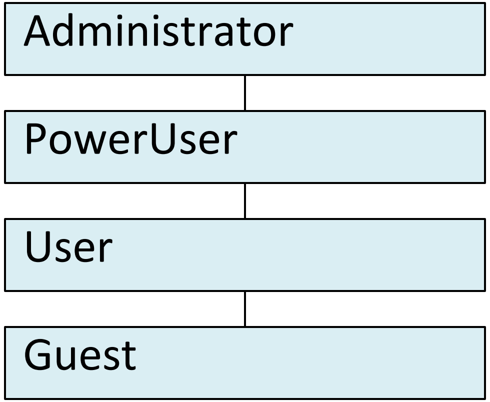
- Delegation
  - Cap: Process can pass capability at run time
  - ACL: Try to get owner to add permission to list?
    - More common: let other process act under current user
- Revocation
  - ACL: Remove user or group from list
  - Cap: Try to get capability back from process?
    - Possible in some systems if appropriate bookkeeping
      - OS knows which data is capability
      - If capability is used for multiple resources, have to revoke all or none …
    - Indirection: capability points to pointer to resource
      - If $C \rightarrow P \rightarrow R$, then revoke capability C by setting $P=0$

Roles (also called Groups)

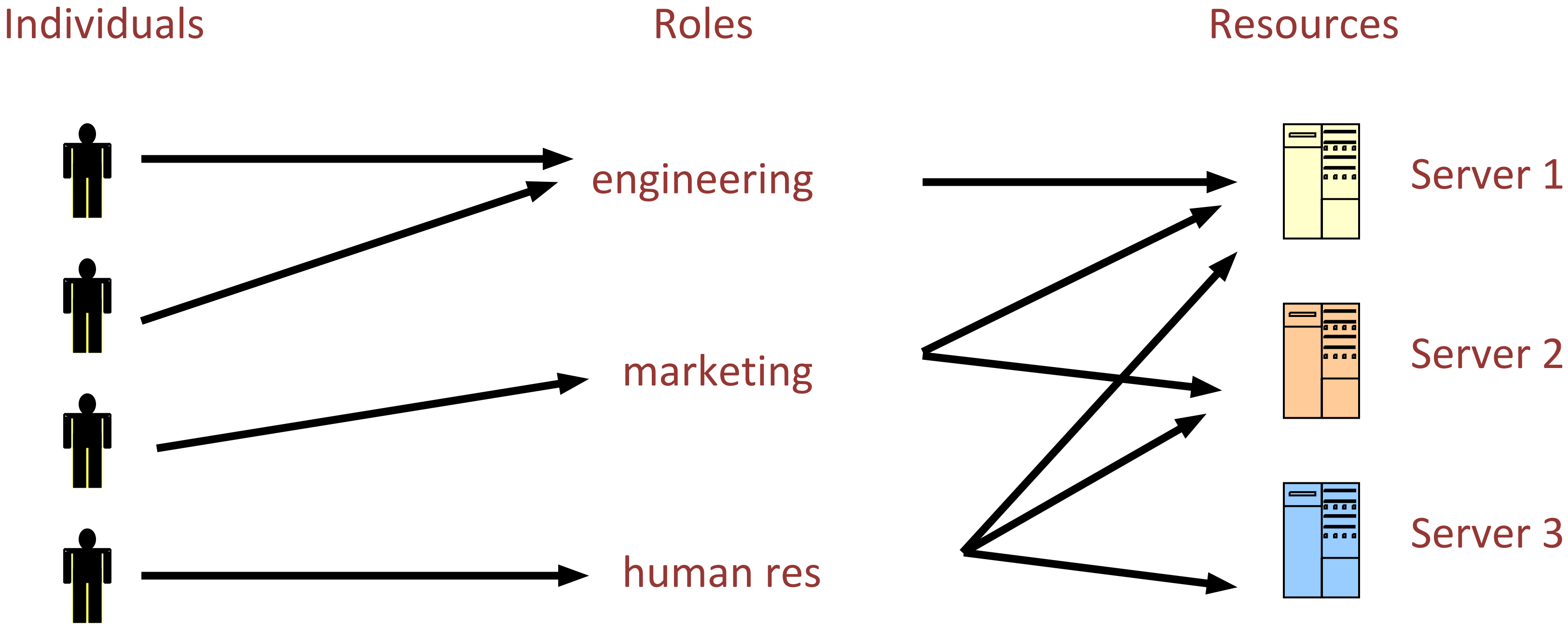
- Role = set of users
  - Administrator, PowerUser, User, Guest
  - Assign permissions to roles; each user gets permission
- Role hierarchy
  - Partial order of roles
  - Each role gets

    permissions of roles below
  - List only new permissions

  given to each role



Role-Based Access Control



Individuals

Roles

Resources

engineering

marketing

human res

Server 1

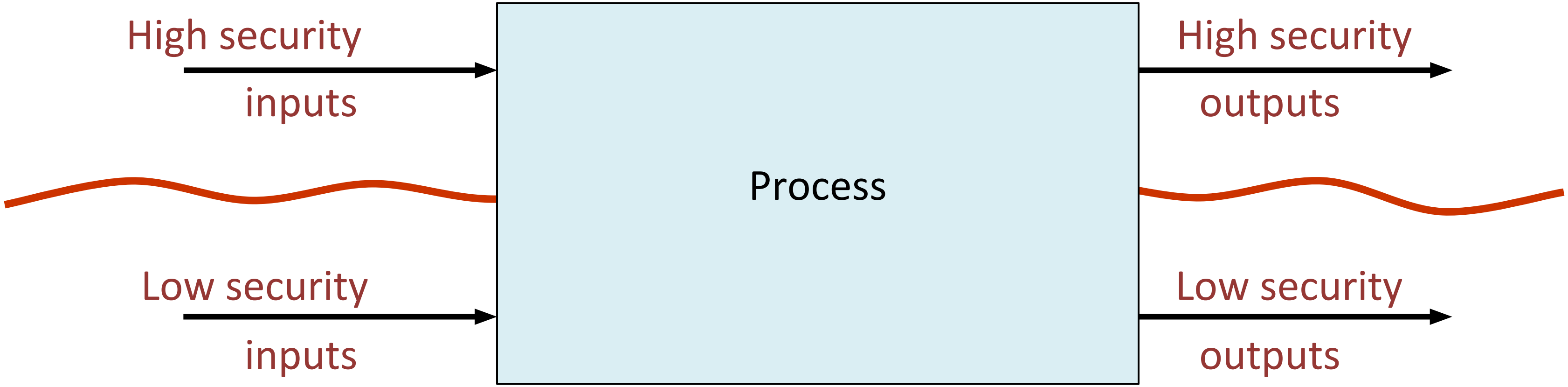
Server 2

Server 3

Advantage: user's change more frequently than roles

Dawn Song

Information flow



High security inputs

Low security inputs

Process

High security outputs

Low security outputs

Dawn Song

Security Architecture Examples

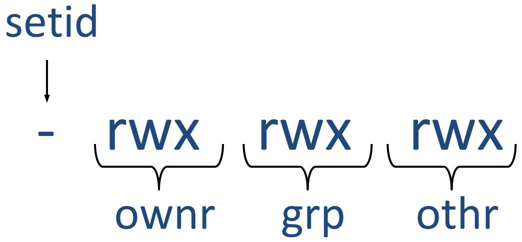
Unix access control

- File has access control list (ACL)
  - Grants permission to user ids
  - Owner, group, other
- Process has user id
  - Inherit from creating process
  - Process can change id
    - Restricted set of options
  - Special “root” id
    - Bypass access control restrictions

	File 1	File 2	…
User 1	read	write	-
User 2	write	write	-
User 3	-	-	read
…			
User m	Read	write	write

Unix file access control list

- Each file has owner and group
- Permissions set by owner
  - Read, write, execute
  - Owner, group, other
  - Represented by vector of four octal values
- Only owner, root can change permissions
  - This privilege cannot be delegated or shared
- Setid bits – Discuss in a few slides



setid

- rwx rwx rwx

ownr grp othr

Question

- Owner can have fewer privileges than other
  - What happens?
    - Owner gets access?
    - Owner does not?

Prioritized resolution of differences

```
if user = owner then owner  permission
     else if user in group then group  permission
          else other  permission
```

Privileged Programs

- Privilege management is coarse-grained in today's OS
  - Root can do anything
- Many programs run as root
  - Even though they only need to perform a small number of priviledged operations
- What's the problem?
  - Privileged programs are juicy targets for attackers
  - By finding a bug in parts of the program that do not need privilege, attacker can gain root

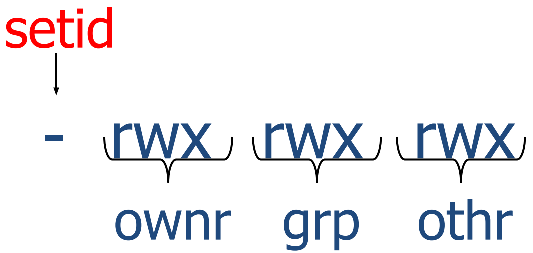
What Can We Do?

- Drop privilege as soon as possible
- Ex: a network daemon only needs privilege to bind to low port # (<1024) at the beginning
  - Solution?
  - Drop privilege right after binding the port
- What benefit do we gain?
  - Even if attacker finds a bug in later part of the code, can't gain privilege any more
- How to drop privilege?
  - Setuid programming in UNIX

Unix file permission

- Each file has owner and group
- Permissions set by owner
    - Read, write, execute
    - Owner, group, other
    - Represented by vector of

      four octal values
- Only owner, root can change permissions
    - This privilege cannot be delegated or shared
- Setid bits



setid

- rwx rwx rwx

ownr grp othr

Effective user id (EUID) in UNIX

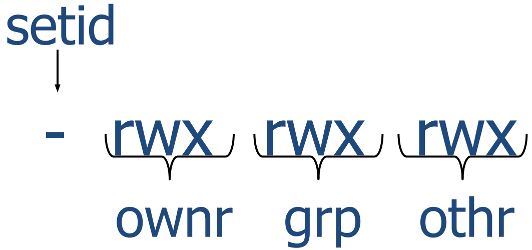
- Each process has three Ids
  - Real user ID (RUID)
    - same as the user ID of parent (unless changed)
    - used to determine which user started the process
  - Effective user ID (EUID)
    - from set user ID bit on the file being executed, or sys call
    - determines the permissions for process
      - file access and port binding
  - Saved user ID (SUID)
    - So previous EUID can be restored
- Real group ID, effective group ID, used similarly

Operations on UIDs

- Root
  - ID=0 for superuser root; can access any file
- Fork and Exec
  - Inherit three IDs, except exec of file with setuid bit
- Setuid system calls
  - seteuid(newid) can set EUID to
    - Real ID or saved ID, regardless of current EUID
    - Any ID, if EUID=0
- Details are actually more complicated
  - Several different calls: setuid, seteuid, setreuid

Setid bits on executable Unix file

- Three setid bits
  - Setuid – set EUID of process to ID of file owner
  - Setgid – set EGID of process to GID of file
  - Sticky
    - Off: if user has write permission on directory, can rename or remove files, even if not owner
    - On: only file owner, directory owner, and root can rename or remove file in the directory

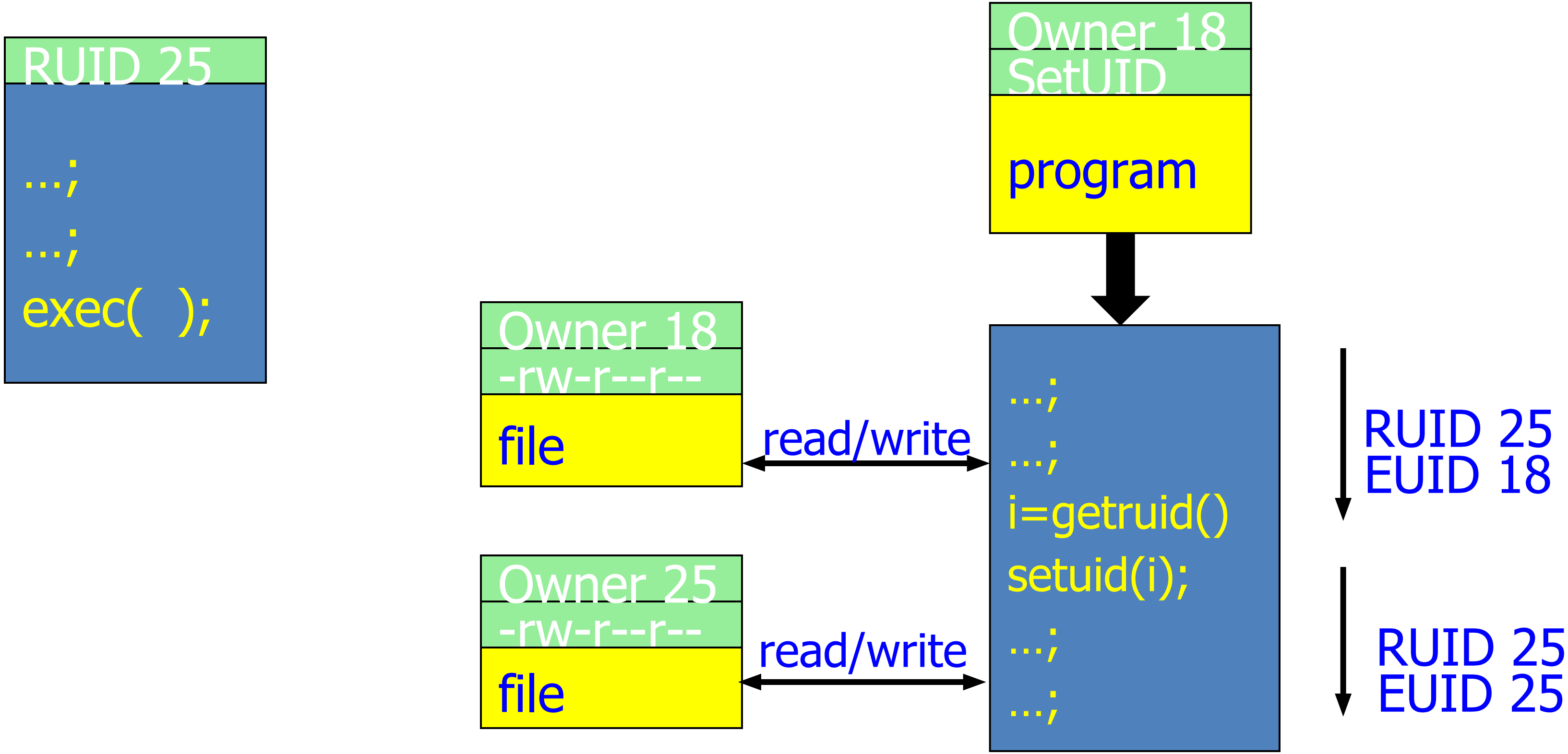


setid

- rwx rwx rwx

ownr grp othr

Drop Privilege



RUID 25
…;
…;
exec( );
Owner 18
SetUID
program
Owner 18
-rw-r--r--
file
read/write
Owner 25
-rw-r--r--
file
read/write
…;
…;
i=getruid()
setuid(i);
…;
…;
RUID 25
EUID 18
RUID 25
EUID 25