



# Secure Architecture Principles

---

Isolation and Least Privilege

Access Control Concepts

Operating Systems

Browser Isolation and Least Privilege



# Secure Architecture Principles

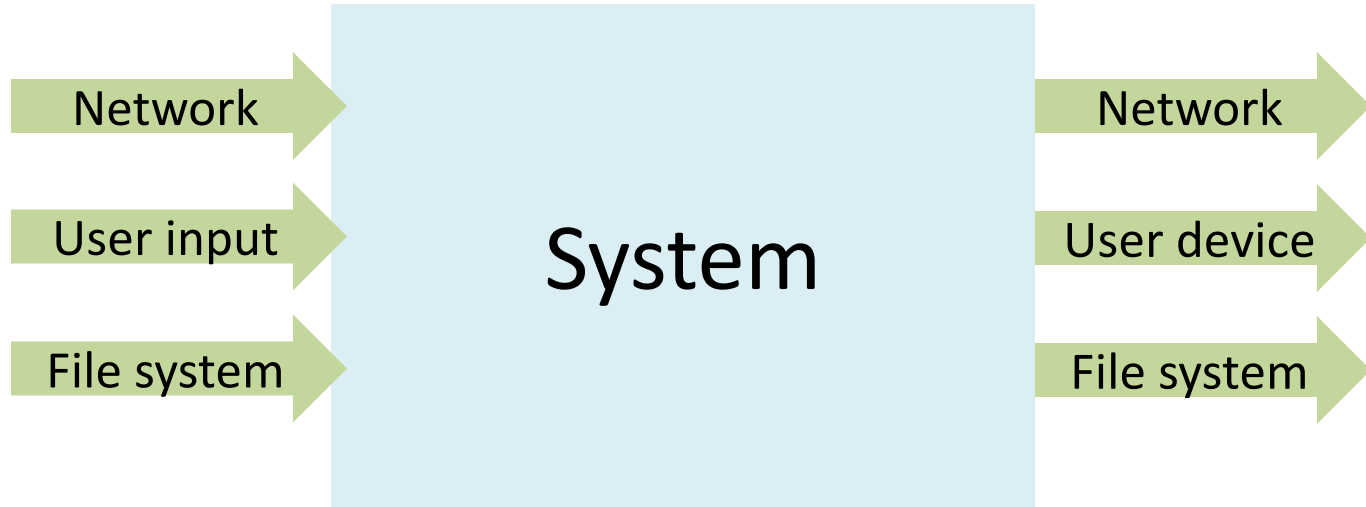
---

Isolation and  
Least Privilege

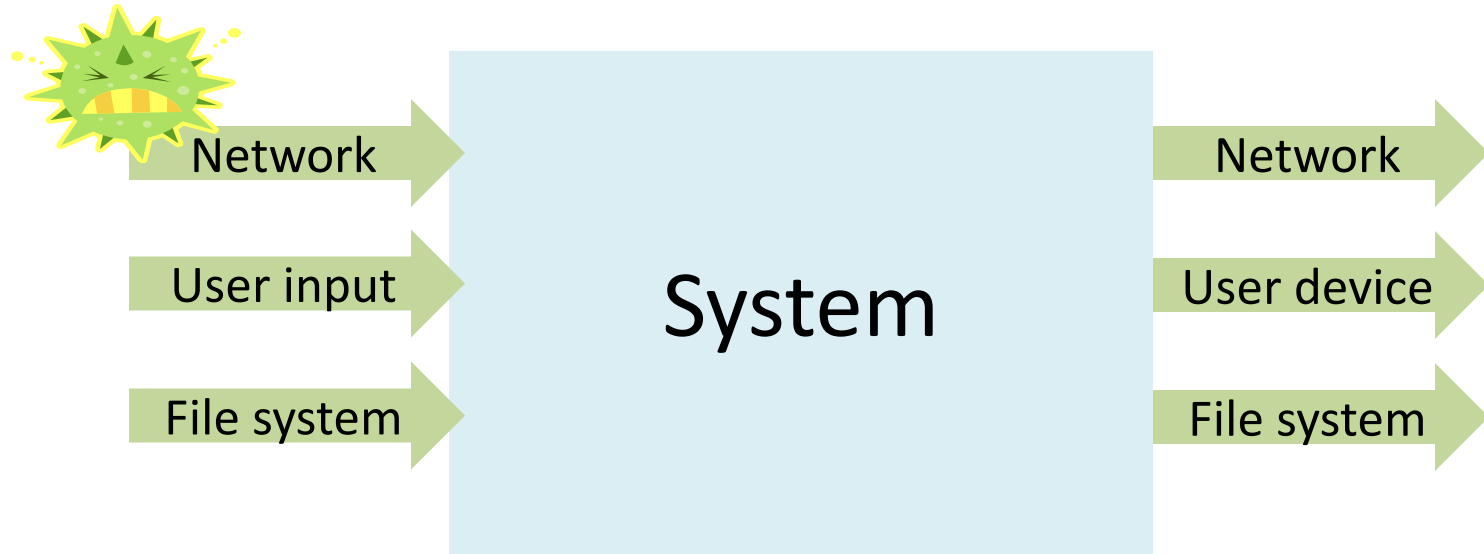
# Principles of Secure Design

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

# Monolithic design



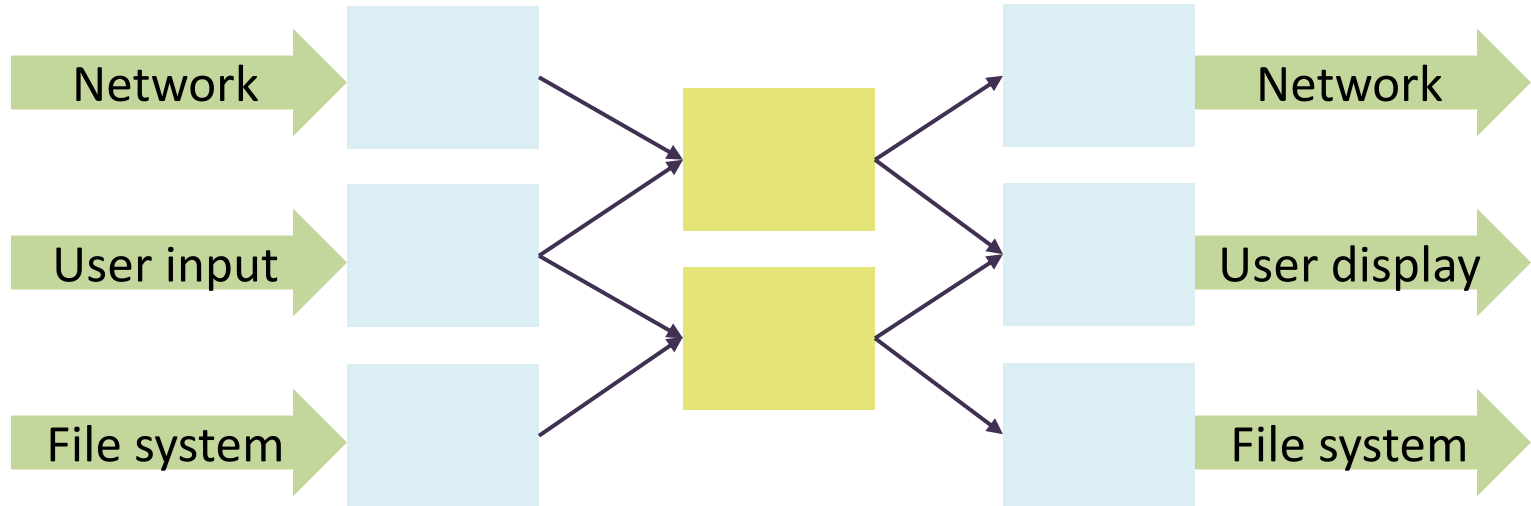
# Monolithic design



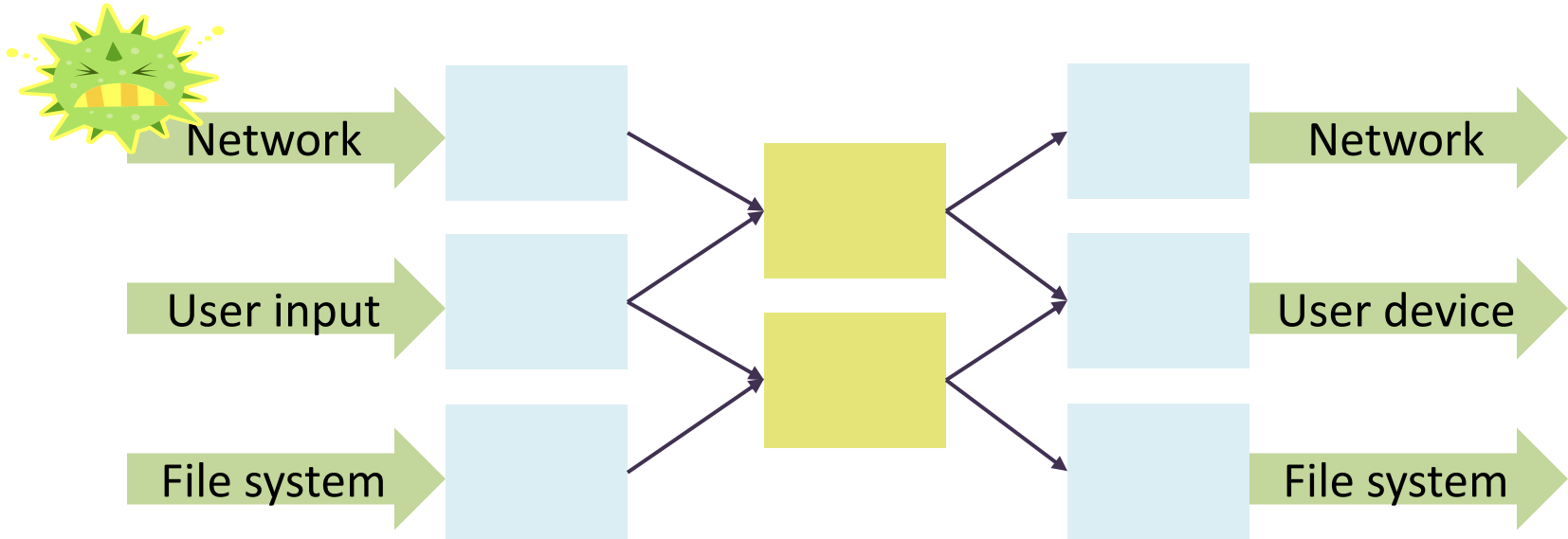
# Monolithic design



# Component design

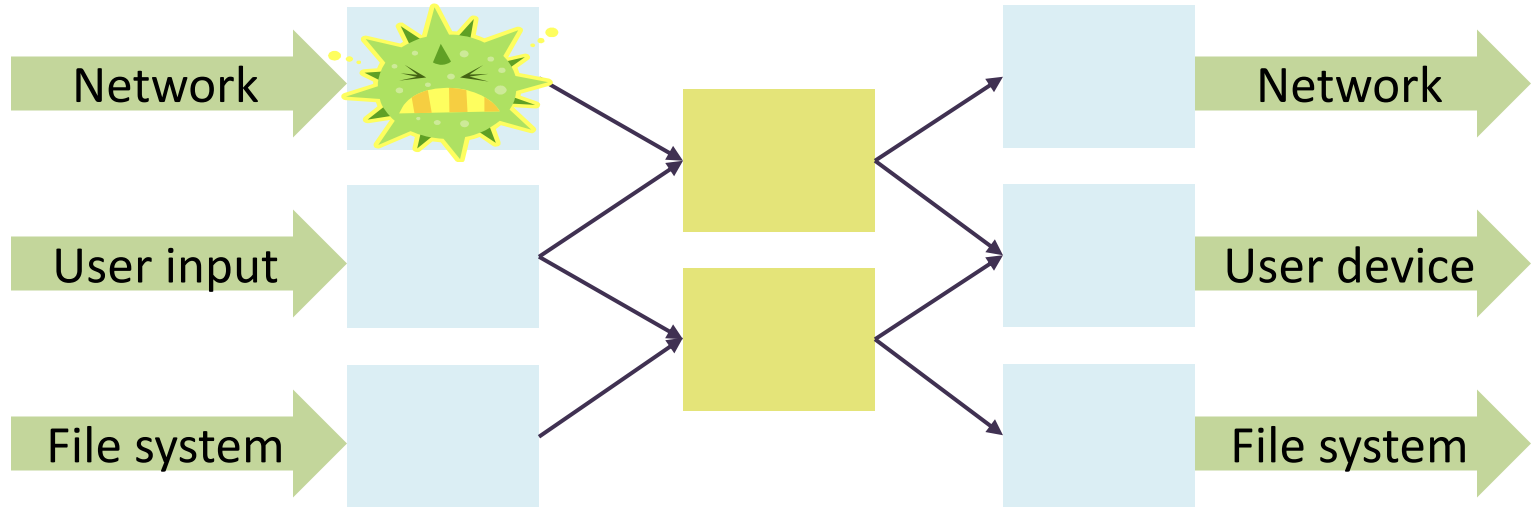


# Component design





# Component design



# Principle of Least Privilege

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

# Example: Mail Agent

- 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
  - Compartmentalized design

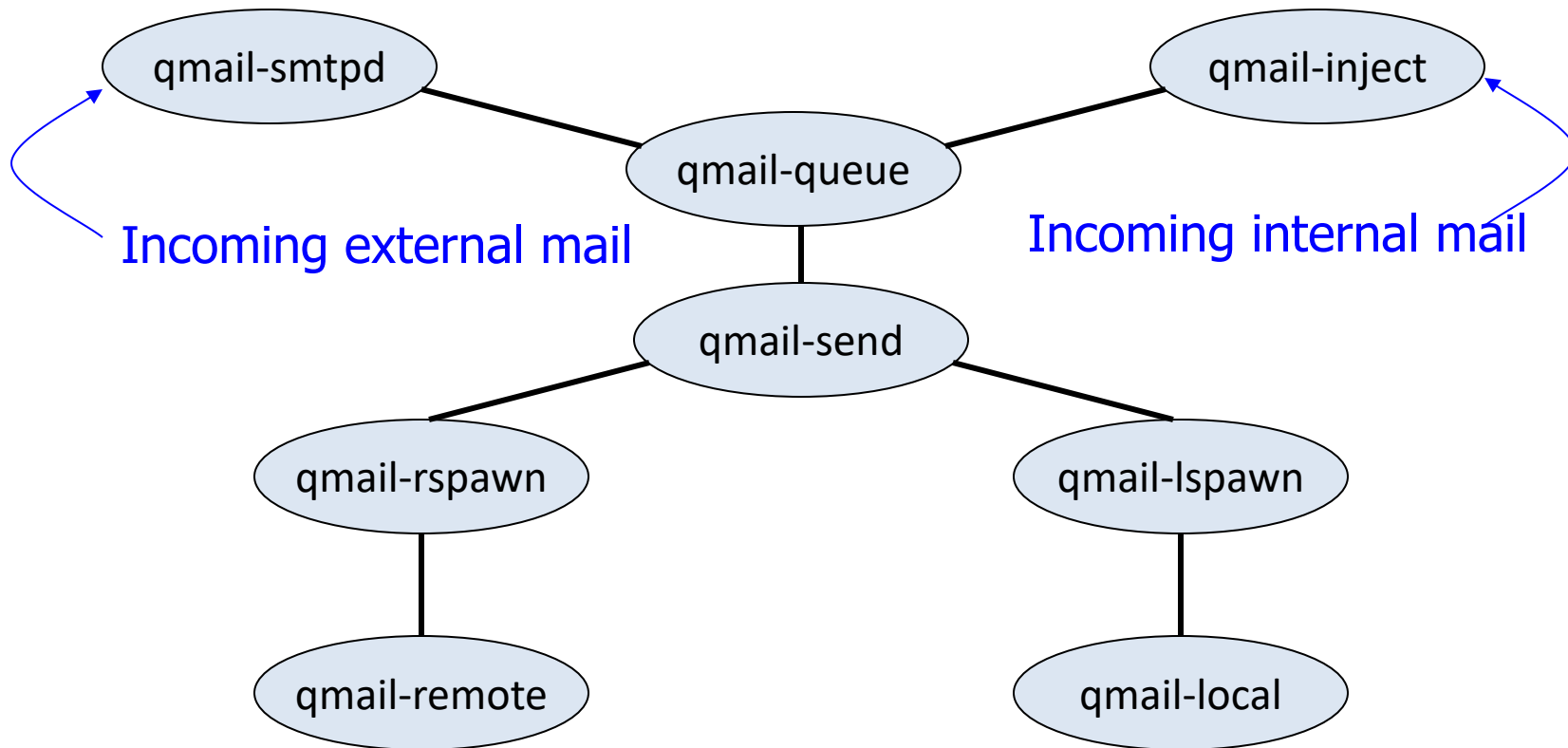
# OS Basics (before examples)

- Isolation between processes
  - Each process has a UID
    - Two processes with same UID have same permissions
  - A process may access files, network sockets, ....
    - Permission granted according to UID
- Relation to previous terminology
  - Compartment defined by UID has Privileges defined by allowed actions on system resources

# Qmail design

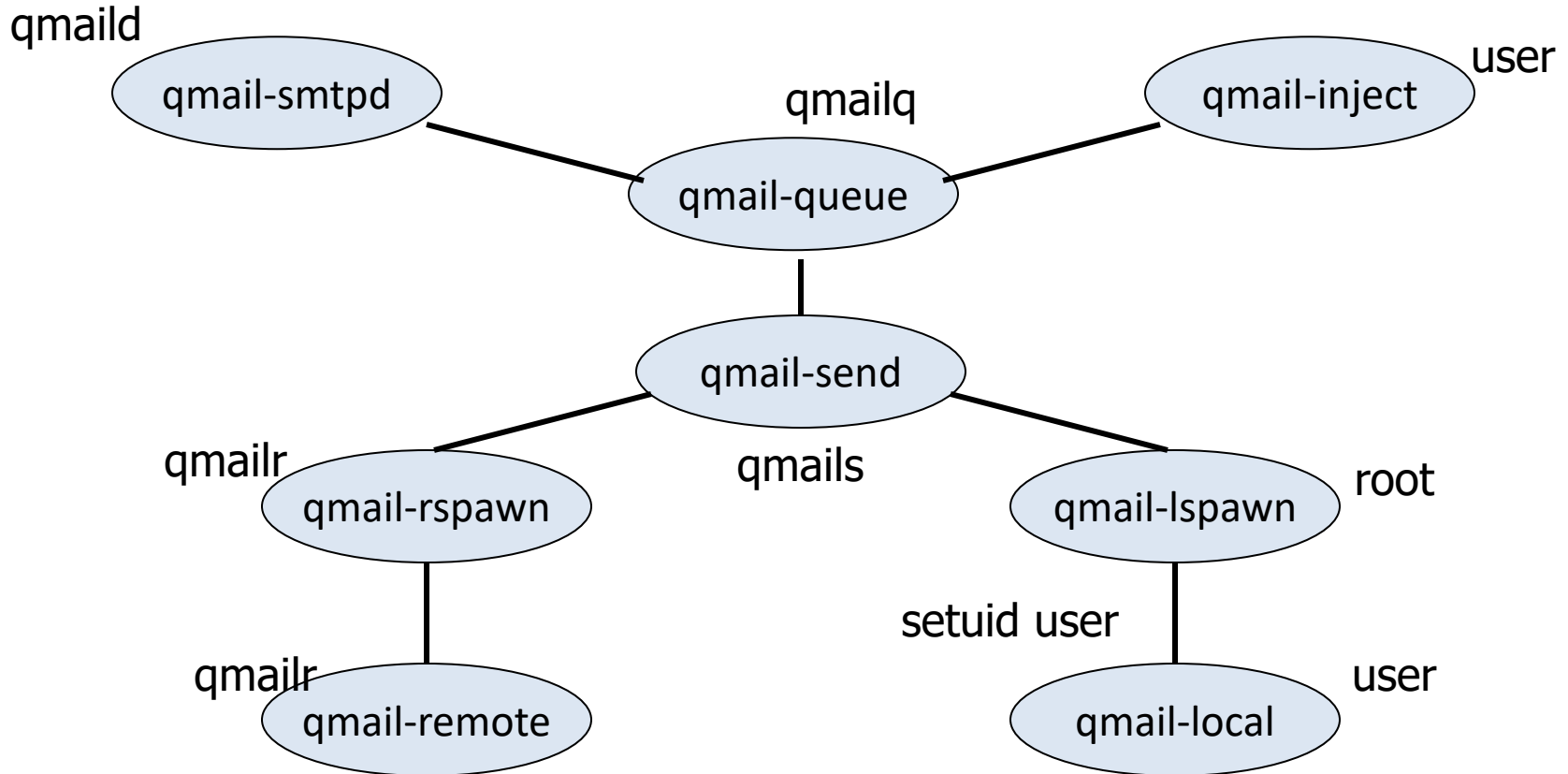
- Isolation based on OS isolation
  - Separate modules run as separate “users”
  - Each user only has access to specific resources
- Least privilege
  - Minimal privileges for each UID
  - Only one “setuid” program
    - setuid allows a program to run as different users
  - Only one “root” program
    - root program has all privileges

# Structure of qmail

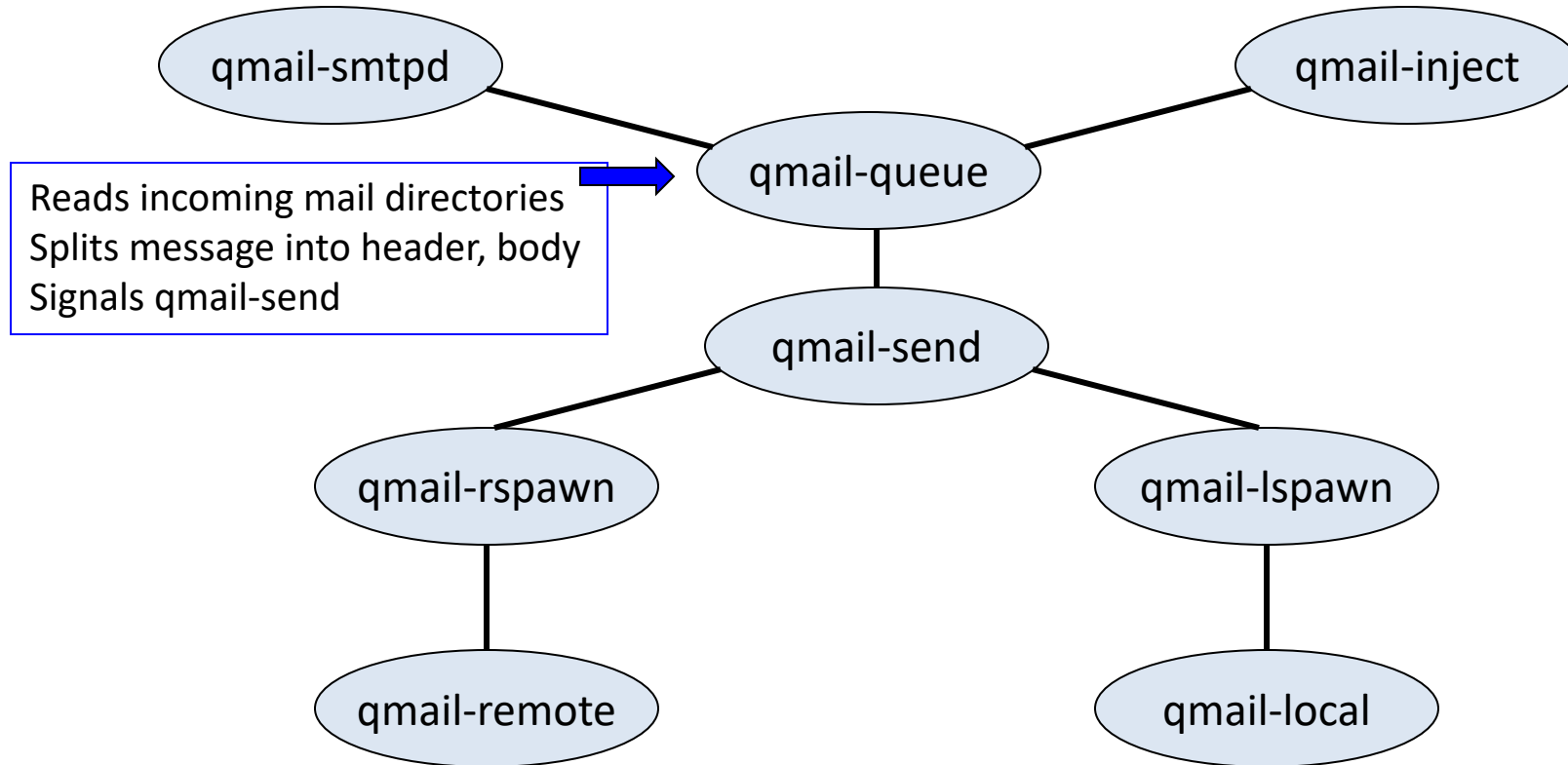


# Isolation by Unix UIDs

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

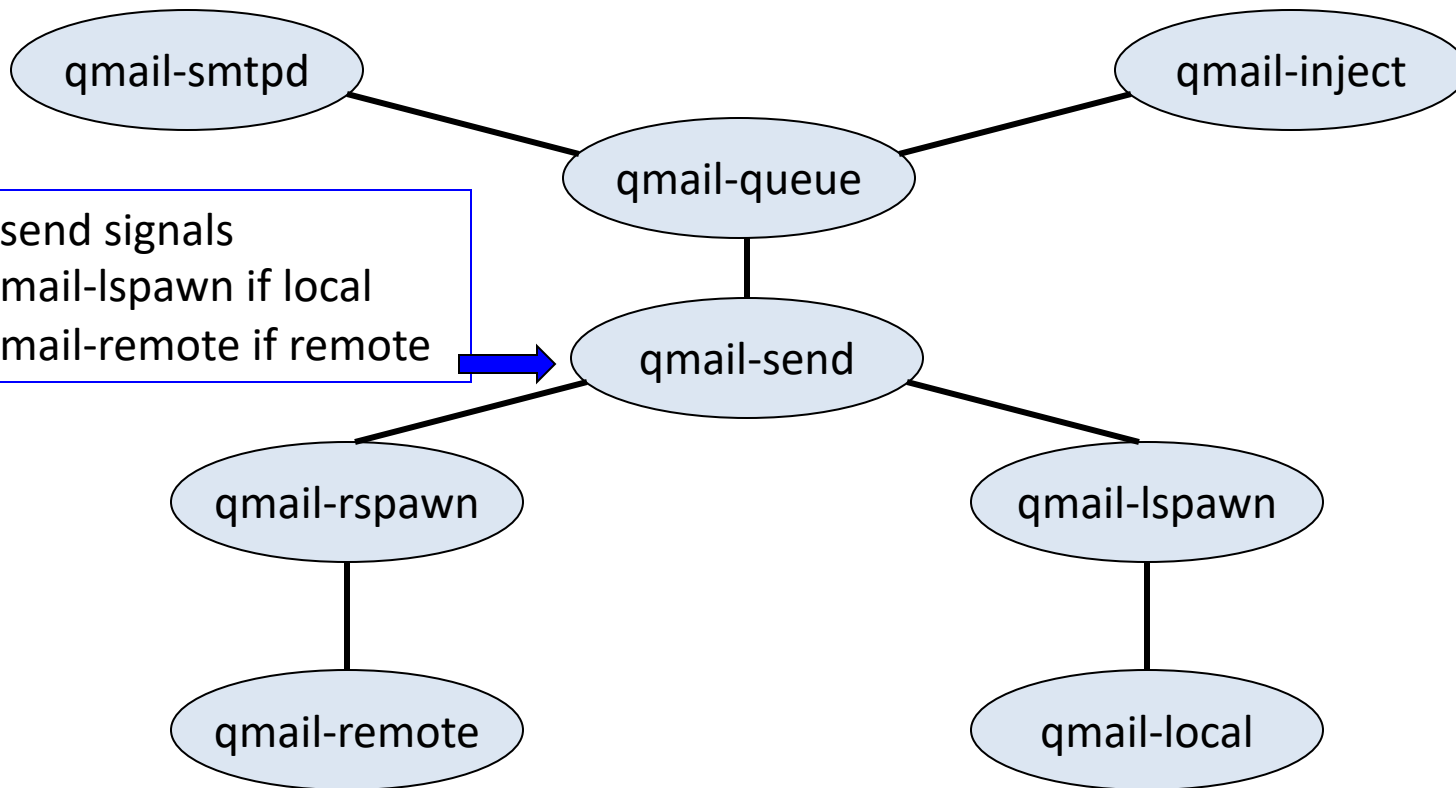


# Structure of qmail

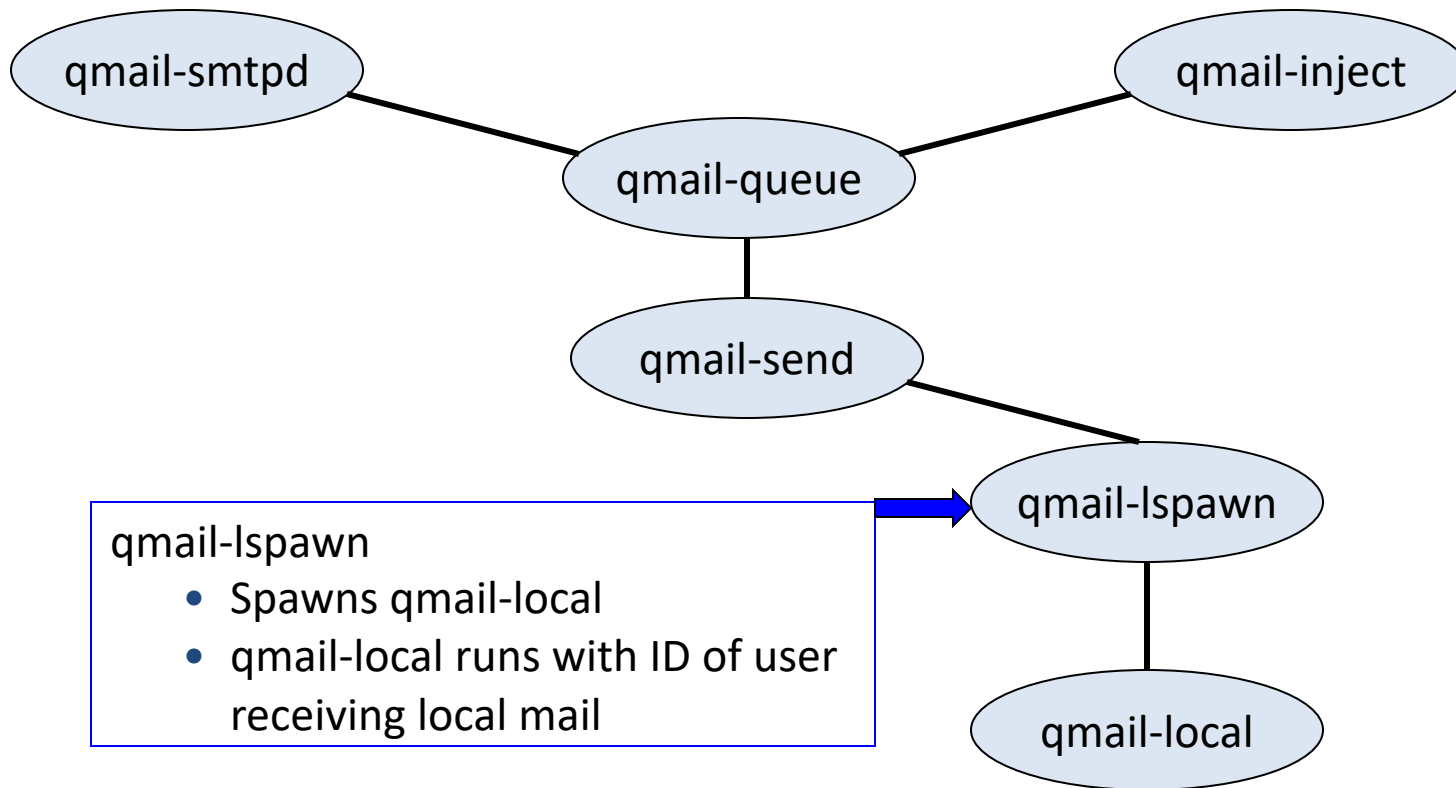




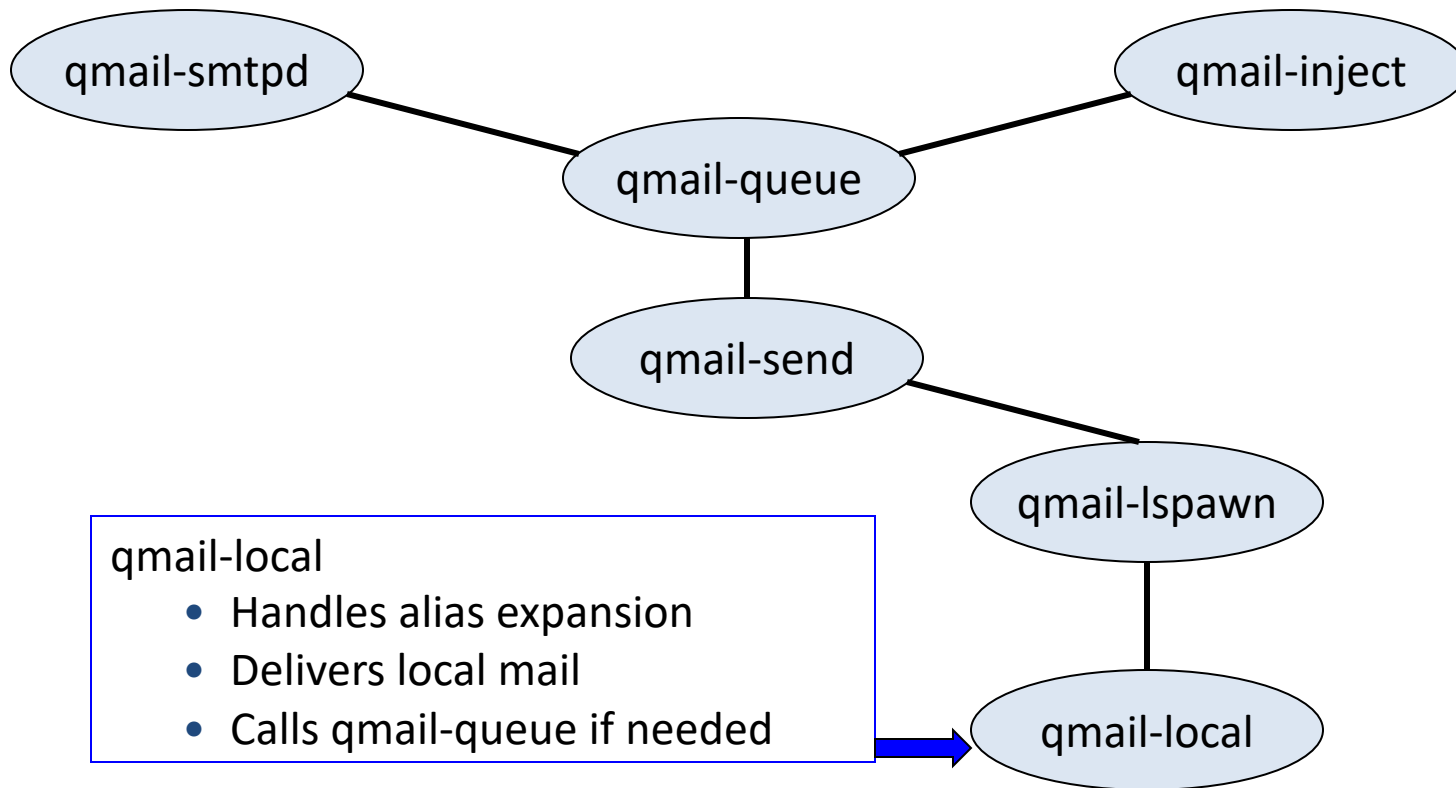
# Structure of qmail



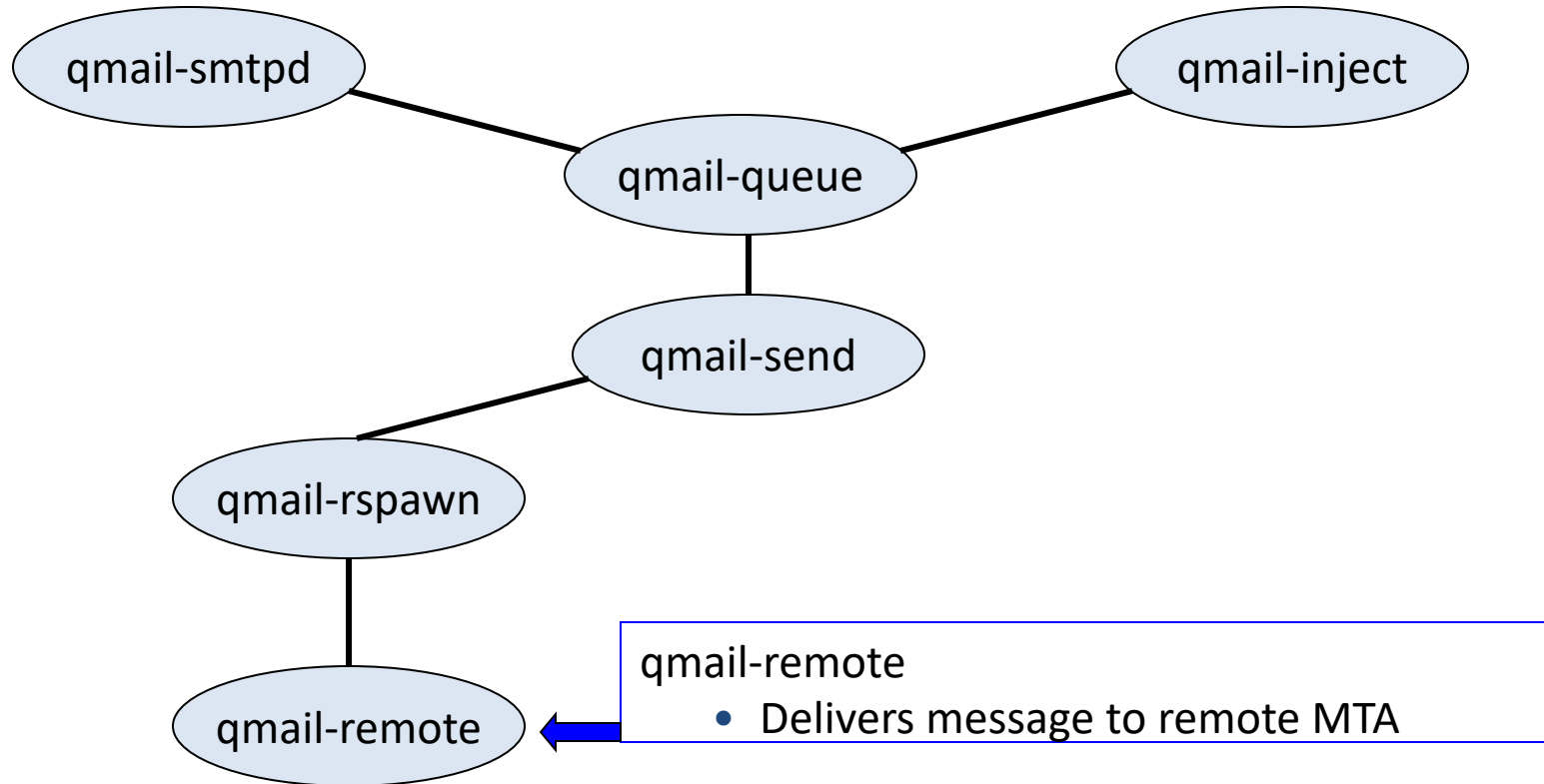
# Structure of qmail



# Structure of qmail

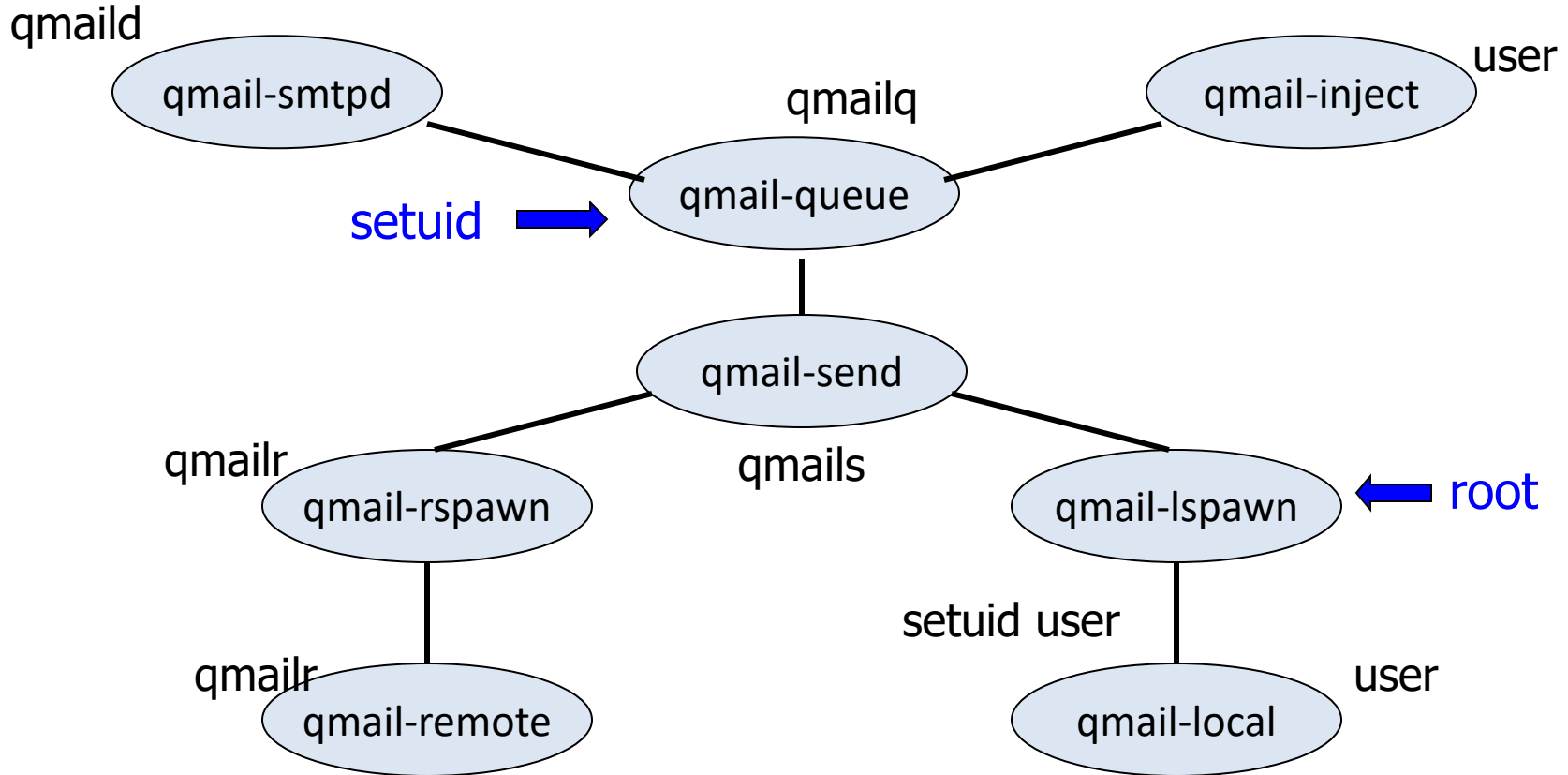


# Structure of qmail

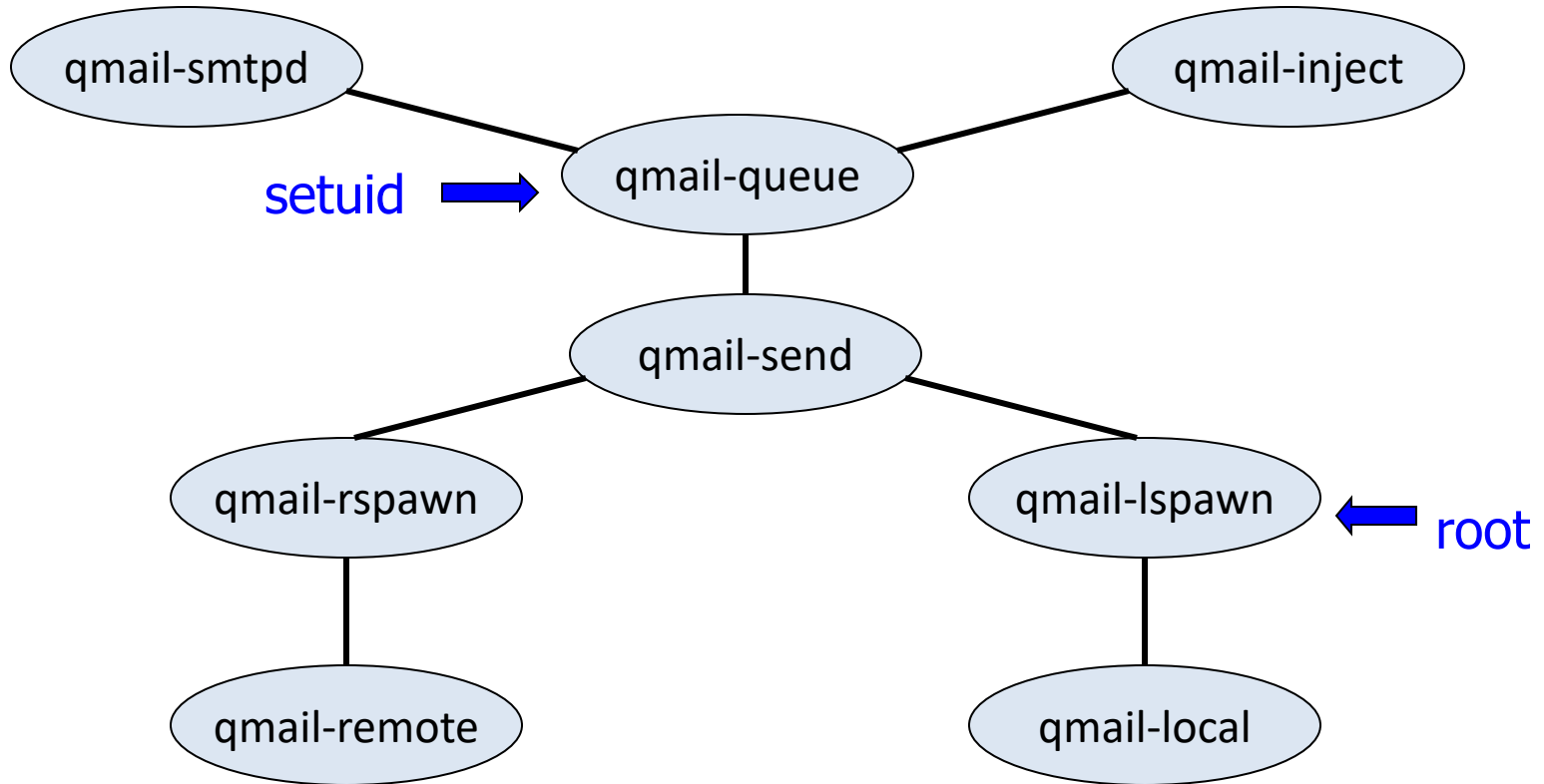


# Isolation by Unix UIDs

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



# Least privilege



# Android process isolation

- Android application sandbox
  - Isolation: Each application runs with its own UID in own VM
    - Provides memory protection
    - Communication protected using Unix domain sockets
    - Only ping, zygote (spawn another process) run as root
  - Interaction: reference monitor checks permissions on inter-component communication
  - Least Privilege: Applications announces permission
    - User grants access at install time



# Secure Architecture Principles

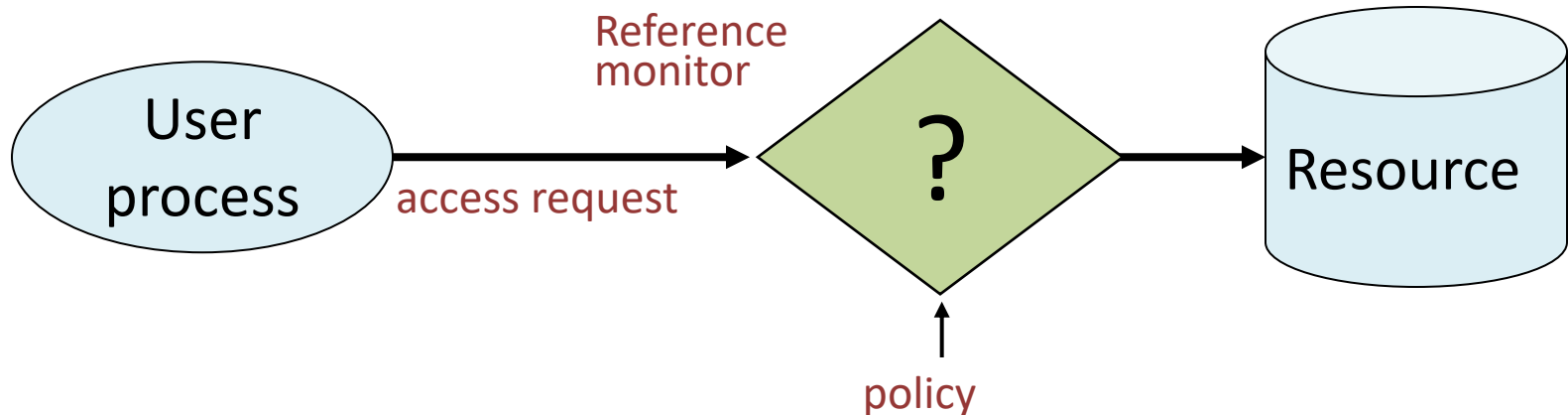
---

## Access Control Concepts

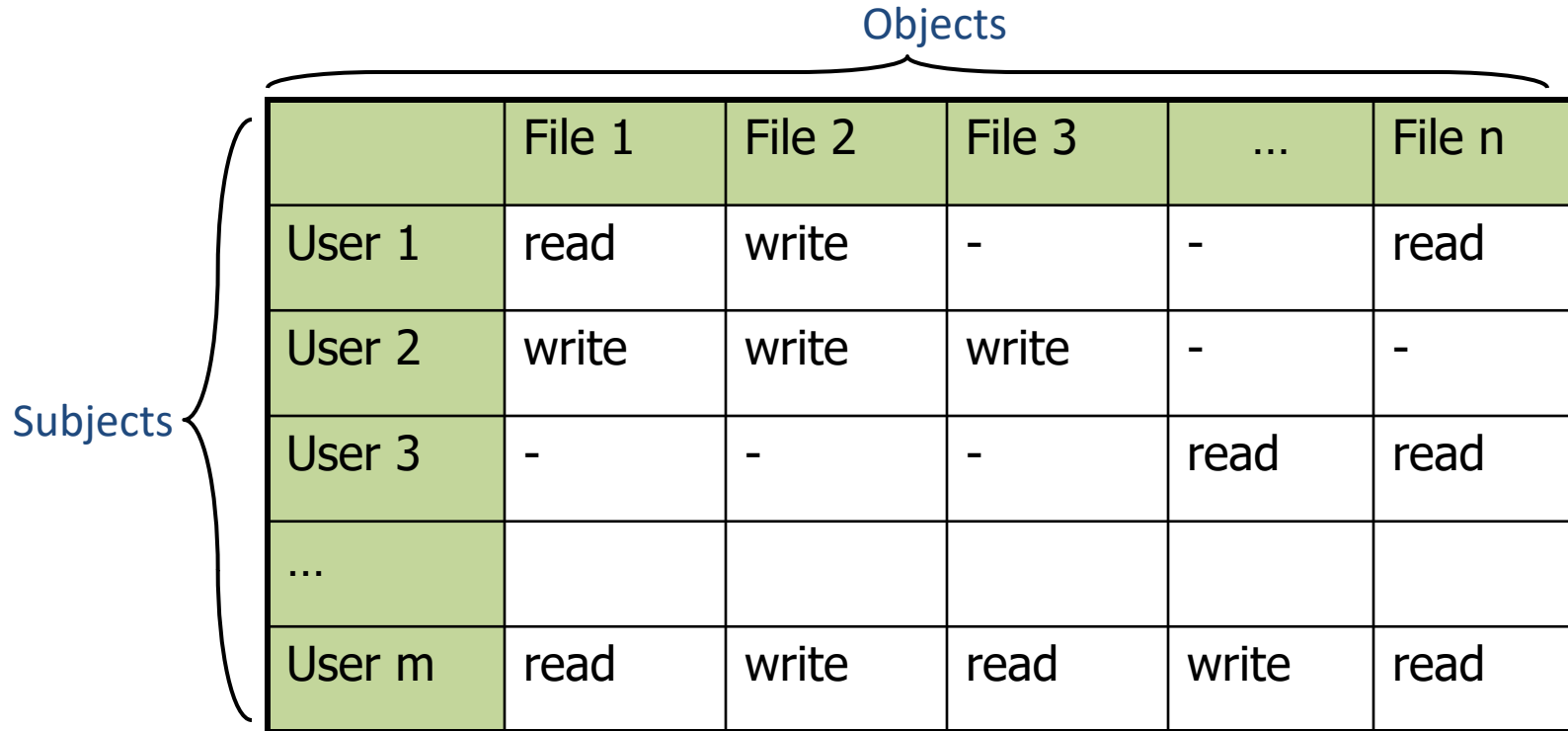


# Access control

- 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]



The diagram illustrates an Access Control Matrix. A large curly bracket on the left, labeled 'Subjects' in blue, groups the rows of the matrix. A large curly bracket on top, labeled 'Objects' in blue, groups the columns. The matrix itself is a table with a header row and several data rows. The header row lists objects: 'File 1', 'File 2', 'File 3', an ellipsis, and 'File n'. The data rows list subjects: 'User 1', 'User 2', 'User 3', an ellipsis, and 'User m'. Each cell in the matrix contains a permission value, either 'read', 'write', or a hyphen '-' indicating no permission.

	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

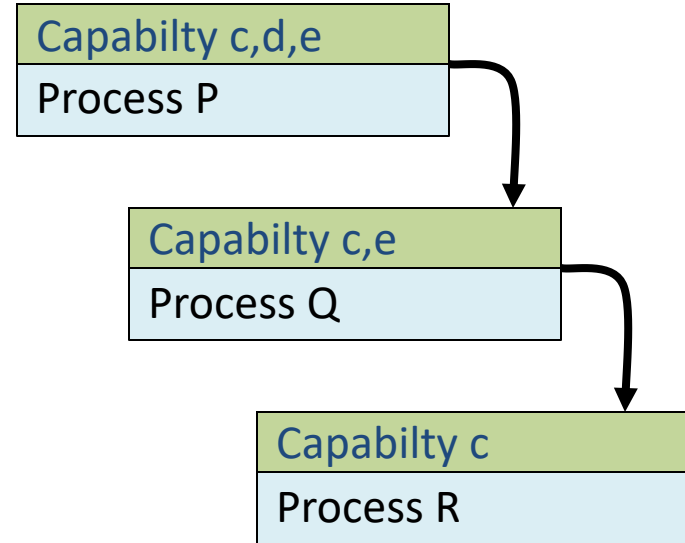
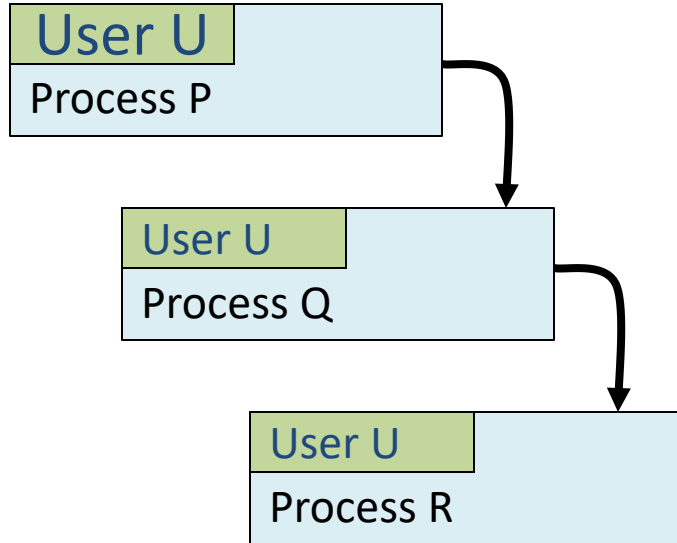
Access control lists are widely used, often with groups

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 identity of user/process

# ACL vs Capabilities

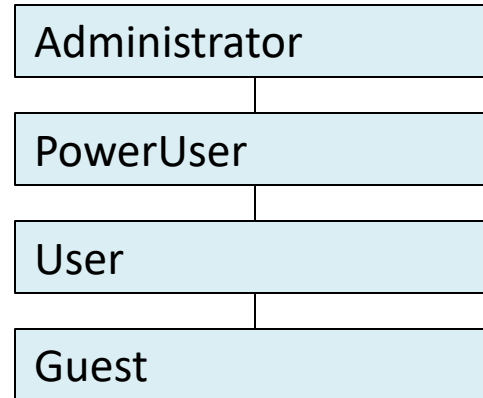


# ACL vs Capabilities

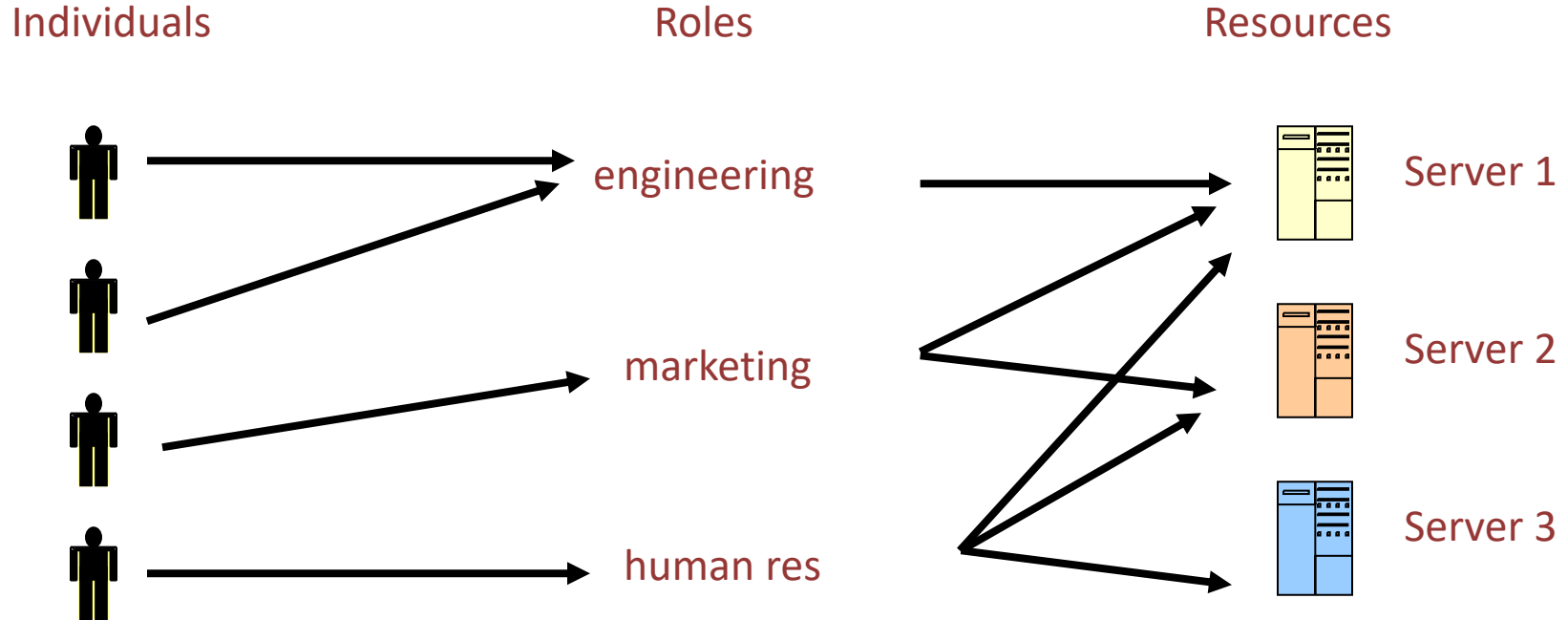
- Delegation
  - Cap: Process can pass capability at run time
  - ACL: Try to get owner to add permission to list?
    - More common: let other process act under current user
- Revocation
  - ACL: Remove user or group from list
  - Cap: Try to get capability back from process?
    - Possible in some 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



Advantage: users change more frequently than roles





# Secure Architecture Principles

---

## Operating Systems

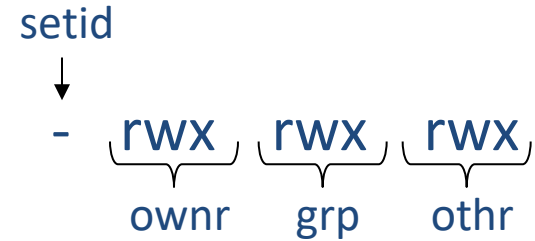
# Unix access control

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

	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



# 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

# Process effective user id (EUID)

- Each process has three IDs (+ more under Linux)
  - 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

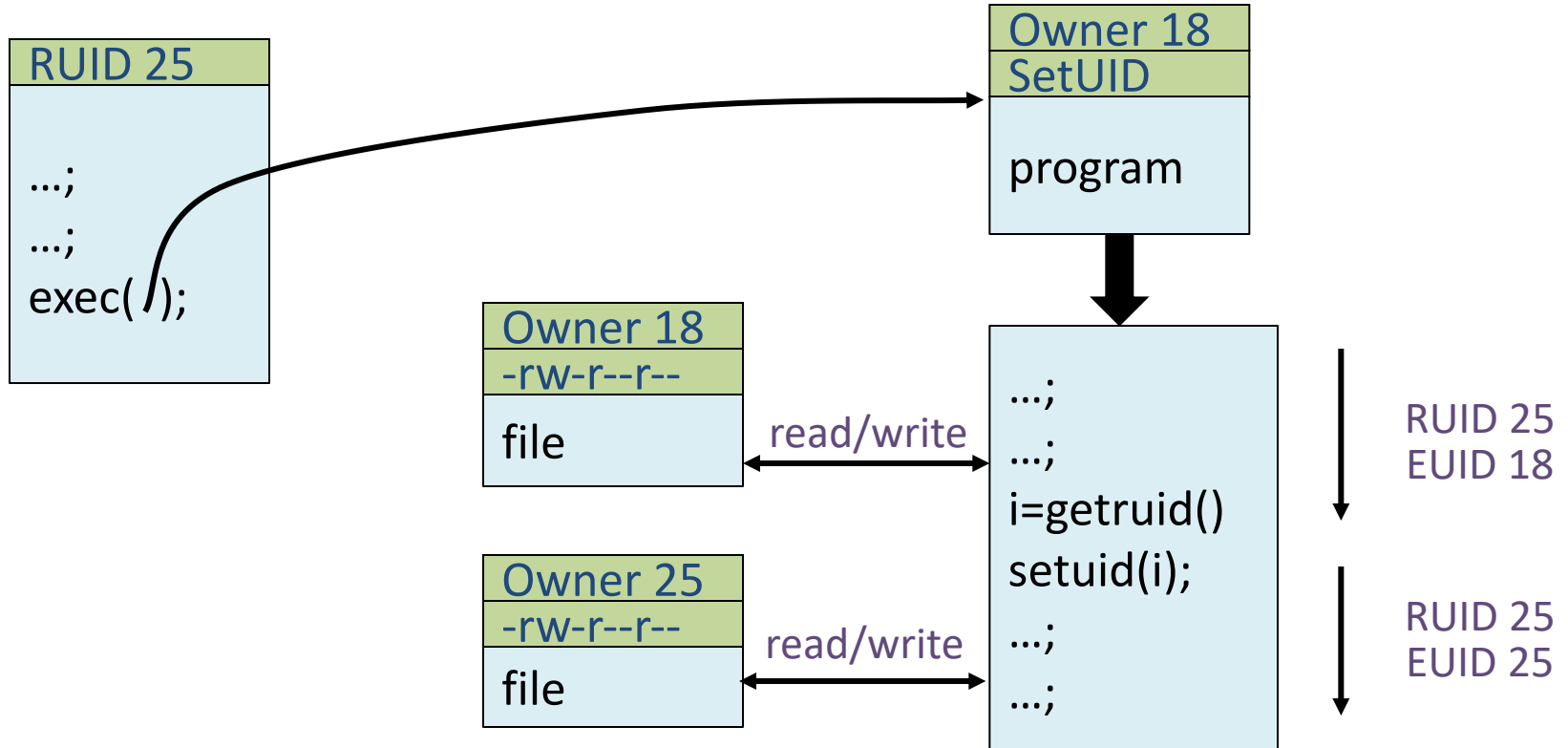
# Process Operations and IDs

- 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

# Example





# Setuid programming

- Be Careful with Setuid 0 !
  - Root can do anything; don' t get tricked
  - Principle of least privilege – change EUID when root privileges no longer needed

# Unix summary

- Good things
  - Some protection from most users
  - Flexible enough to make things possible
- Main limitation
  - Too tempting to use root privileges
  - No way to assume some root privileges without all root privileges

# Weakness in isolation, privileges

- Network-facing Daemons
  - Root processes with network ports open to all remote parties, e.g., sshd, ftpd, sendmail, ...
- Rootkits
  - System extension via dynamically loaded kernel modules
- Environment Variables
  - System variables such as LIBPATH that are shared state across applications. An attacker can change LIBPATH to load an attacker-provided file as a dynamic library

# Weakness in isolation, privileges

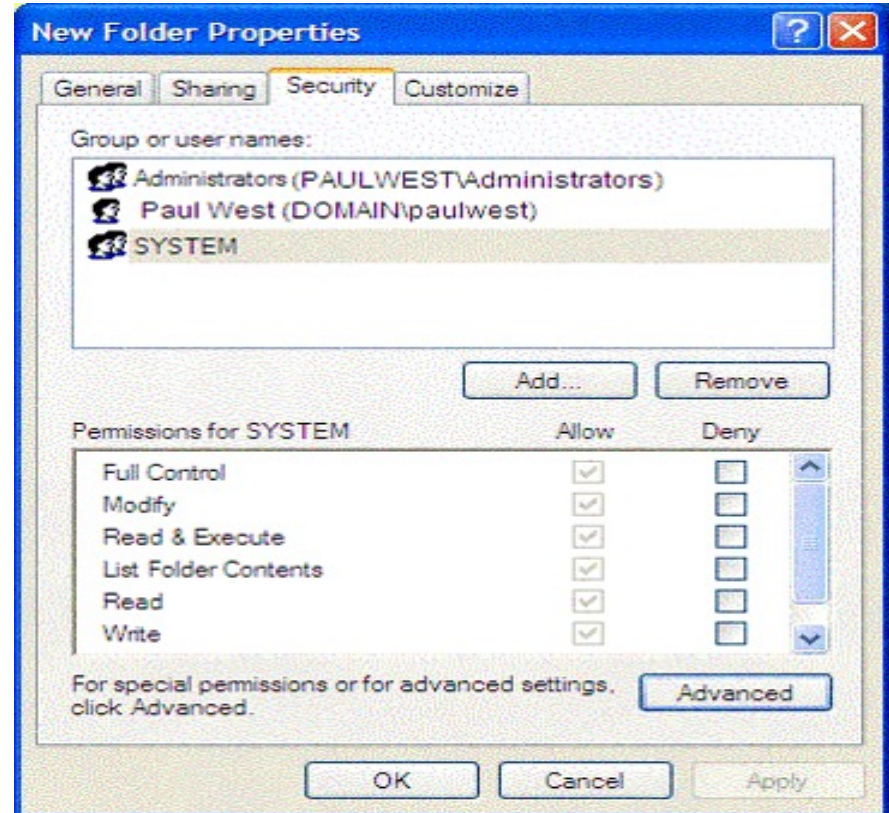
- Shared Resources
  - Since any process can create files in /tmp directory, an untrusted process may create files that are used by arbitrary system processes
- Time-of-Check-to-Time-of-Use (TOCTTOU)
  - Typically, a root process uses system call to determine if initiating user has permission to a particular file, e.g. /tmp/X.
  - After access is authorized and before the file open, user may change the file /tmp/X to a symbolic link to a target file /etc/shadow.

# Access control in Windows

- Some basic functionality similar to Unix
  - Specify access for groups and users
    - Read, modify, change owner, delete
- Some additional concepts
  - Tokens
  - Security attributes
- Generally
  - More flexible than Unix
    - Can define new permissions
    - Can give some but not all administrator privileges

# Identify subject using SID

- Security ID (SID)
  - Identity (replaces UID)
    - SID revision number
    - 48-bit authority value
    - variable number of Relative Identifiers (RIDs), for uniqueness
  - Users, groups, computers, domains, domain members all have SIDs



# Process has set of tokens

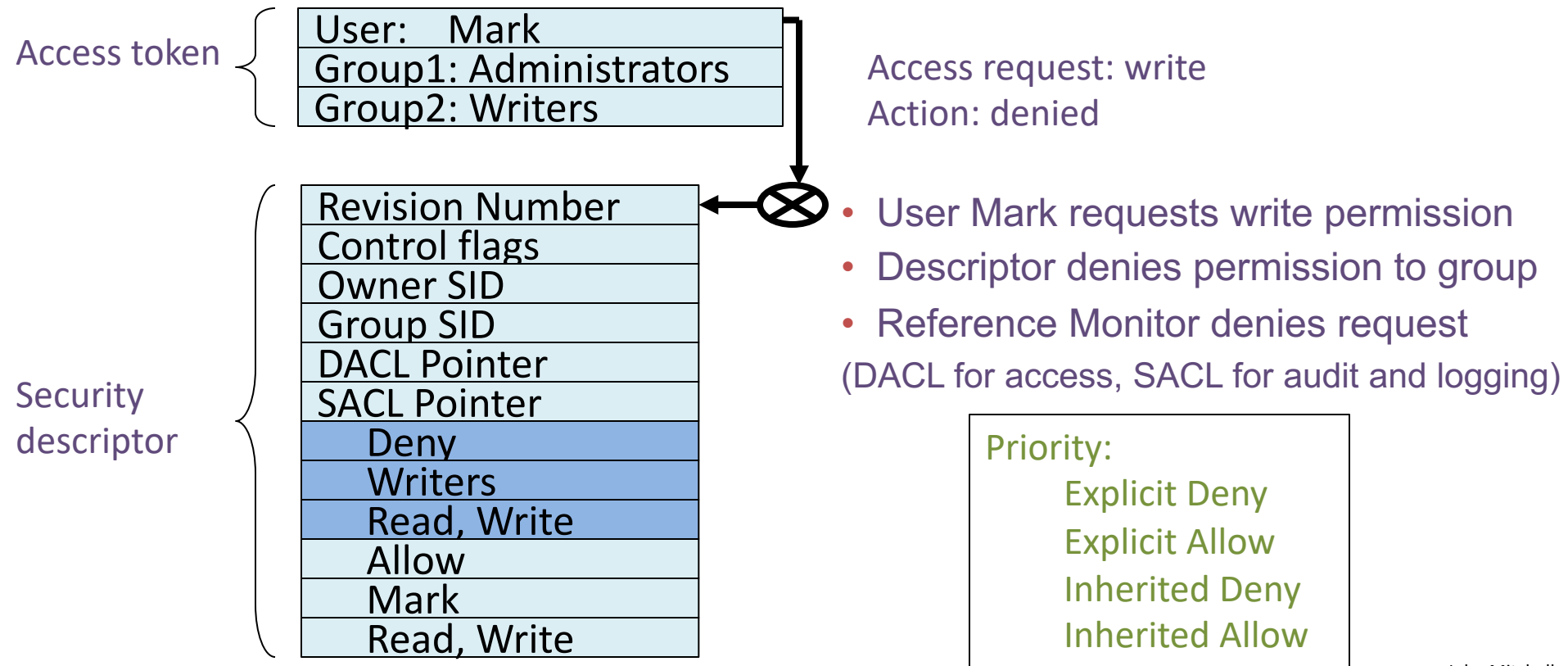
- Security context
  - Privileges, accounts, and groups associated with the process or thread
  - Presented as set of tokens
- Security Reference Monitor
  - Uses tokens to identify the security context of a process or thread
- Impersonation token
  - Used temporarily to adopt a different security context, usually of another user

# Object has security descriptor

- Security descriptor associated with an object
  - Specifies who can perform what actions on the object
- Several fields
  - Header
    - Descriptor revision number
    - Control flags, attributes of the descriptor
      - E.g., memory layout of the descriptor
  - SID of the object's owner
  - SID of the primary group of the object
  - Two attached optional lists:
    - Discretionary Access Control List (DACL) – users, groups, ...
    - System Access Control List (SACL) – system logs, ..



# Example access request



# Impersonation Tokens (compare to setuid)

- Process adopts security attributes of another
  - Client passes impersonation token to server
- Client specifies impersonation level of server
  - Anonymous
    - Token has no information about the client
  - Identification
    - server obtain the SIDs of client and client's privileges, but server cannot impersonate the client
  - Impersonation
    - server identify and impersonate the client
  - Delegation
    - lets server impersonate client on local, remote systems



# Secure Architecture Principles

---

## Browser Isolation and Least Privilege

# Web browser: an analogy

## Operating system

- Subject: Processes
  - Has User ID (UID, SID)
  - Discretionary access control
- Objects
  - File
  - Network
  - ...
- Vulnerabilities
  - Untrusted programs
  - Buffer overflow
  - ...

## Web browser

- Subject: web content (JavaScript)
  - Has “Origin”
  - Mandatory access control
- Objects
  - Document object model
  - Frames
  - Cookies / localStorage
- Vulnerabilities
  - Cross-site scripting
  - Implementation bugs
  - ...

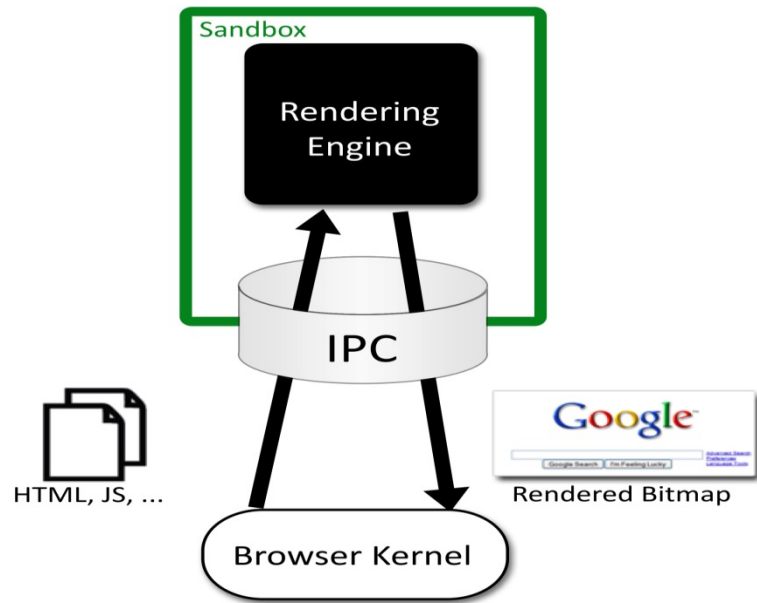
The web browser enforces its own internal policy. If the browser implementation is corrupted, this mechanism becomes unreliable.

# Components of security policy

- Frame-Frame relationships
  - `canScript(A,B)`
    - Can Frame A execute a script that manipulates arbitrary/nontrivial DOM elements of Frame B?
  - `canNavigate(A,B)`
    - Can Frame A change the origin of content for Frame B?
- Frame-principal relationships
  - `readCookie(A,S)`, `writeCookie(A,S)`
    - Can Frame A read/write cookies from site S?

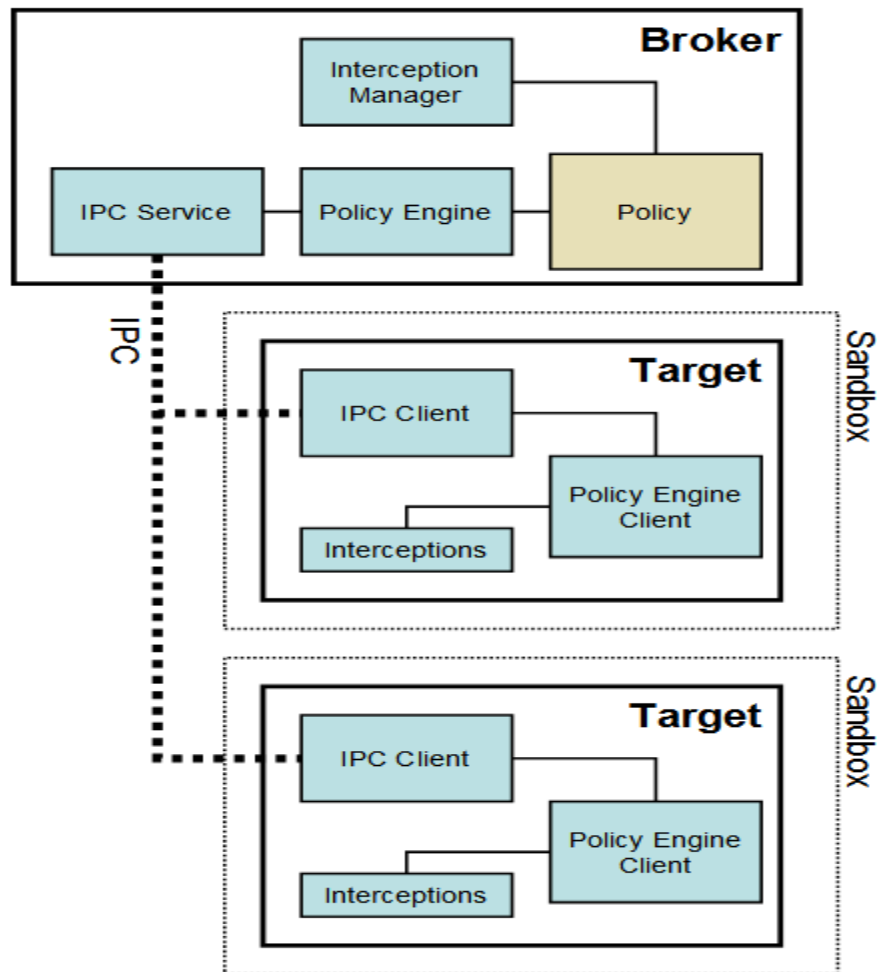
# Chromium Security Architecture

- Browser ("kernel")
  - Full privileges (file system, networking)
- Rendering engine
  - Up to 20 processes
  - Sandboxed
- One process per plugin
  - Full privileges of browser



# Chromium

Communicating sandbox components



See: <http://dev.chromium.org/developers/design-documents/sandbox/>

# Design Decisions

- Compatibility
  - Sites rely on the existing browser security policy
  - Browser is only as useful as the sites it can render
  - Rules out more “clean slate” approaches
- Black Box
  - Only renderer may parse HTML, JavaScript, etc.
  - Kernel enforces coarse-grained security policy
  - Renderer to enforces finer-grained policy decisions
- Minimize User Decisions



# Task Allocation

Rendering Engine	Browser Kernel
HTML parsing	Cookie database
CSS parsing	History database
Image decoding	Password database
JavaScript interpreter	Window management
Regular expressions	Location bar
Layout	Safe Browsing blacklist
Document Object Model	Network stack
Rendering	SSL/TLS
SVG	Disk cache
XML parsing	Download manager
XSLT	Clipboard
<b>Both</b>	
URL parsing	
Unicode parsing	

# Leverage OS Isolation

- Sandbox based on four OS mechanisms
  - A restricted token
  - The Windows *job* object
  - The Windows *desktop* object
  - Windows Vista only: integrity levels
- Specifically, the rendering engine
  - adjusts security token by converting SIDS to DENY\_ONLY, adding restricted SID, and calling AdjustTokenPrivileges
  - runs in a Windows Job Object, restricting ability to create new processes, read or write clipboard, ..
  - runs on a separate desktop, mitigating lax security checking of some Windows APIs

See: <http://dev.chromium.org/developers/design-documents/sandbox/>

# Summary

- Security principles
  - Isolation
  - Principle of Least Privilege
  - Qmail example
- Access Control Concepts
  - Matrix, ACL, Capabilities
- OS Mechanisms
  - Unix
    - File system, Setuid
  - Windows
    - File system, Tokens, EFS
- Browser security architecture
  - Isolation and least privilege example