第4讲

安全体系结构（1）

大纲

- 隔离和最小特权
- 访问控制概念
- 操作系统机制
- 浏览器机制

安全设计原则

- 划分（Compartmentalization）
  - 特权隔离(Privilege separation)
  - 最小特权原则(Least privilege)
- 深度防御
  - 使用多种安全机制
  - 保护薄弱环节（Secure the weakest link）
  - 安全的错误处理（Fail securely）
- 把问题简单化

Fail securely

```
DWORD dwRet = IsAccessAllowed(...);
if (dwRet == ERROR_ACCESS_DENIED) {
  // Security check failed.
  // Inform user that access is denied.
} else {
  // Security check OK.
}
```

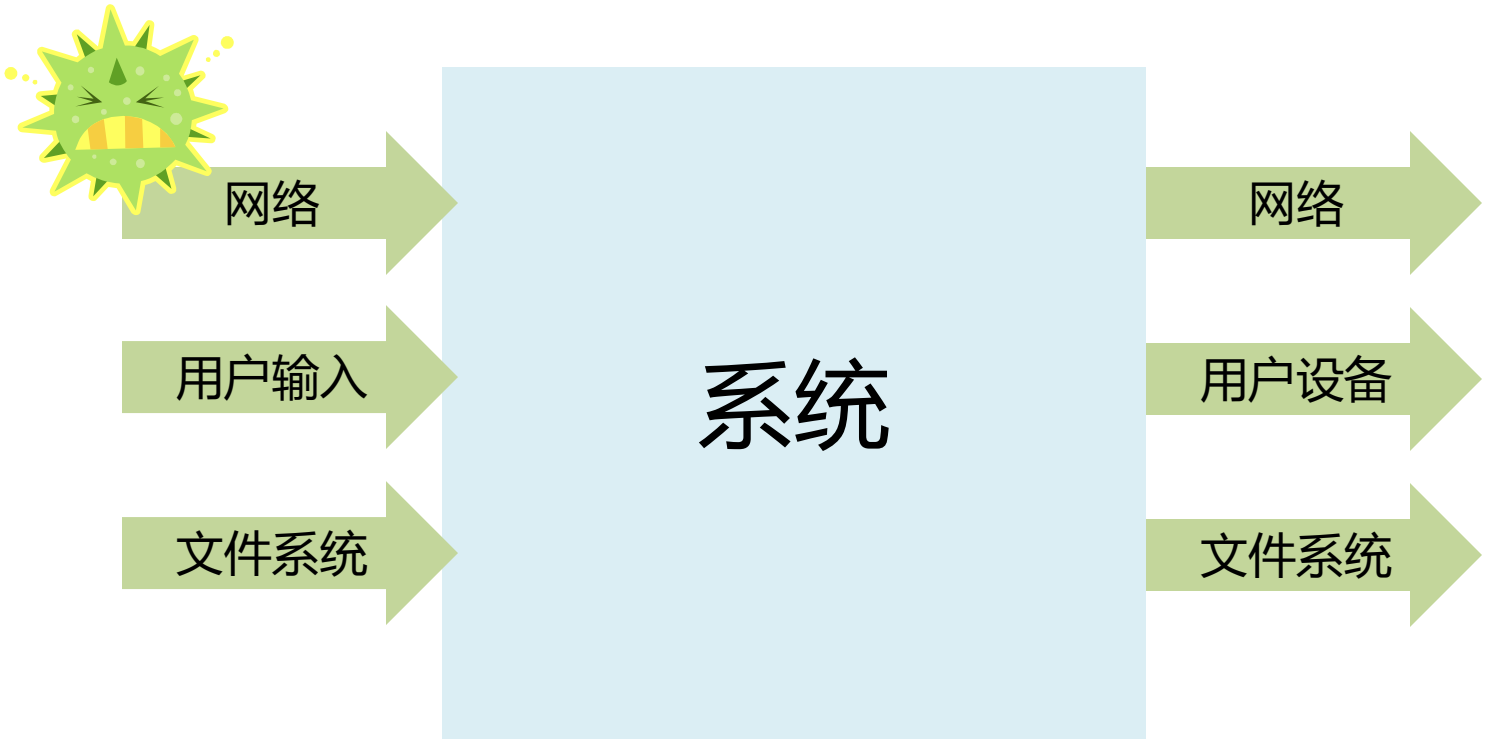
问题？

```
DWORD dwRet = IsAccessAllowed(...);
if (dwRet == NO_ERROR) {
  // Secure check OK.
  // Perform task.
} else {
  // Security check failed.
  // Inform user that access is denied.
}
```

正确的方法
默认拒绝！

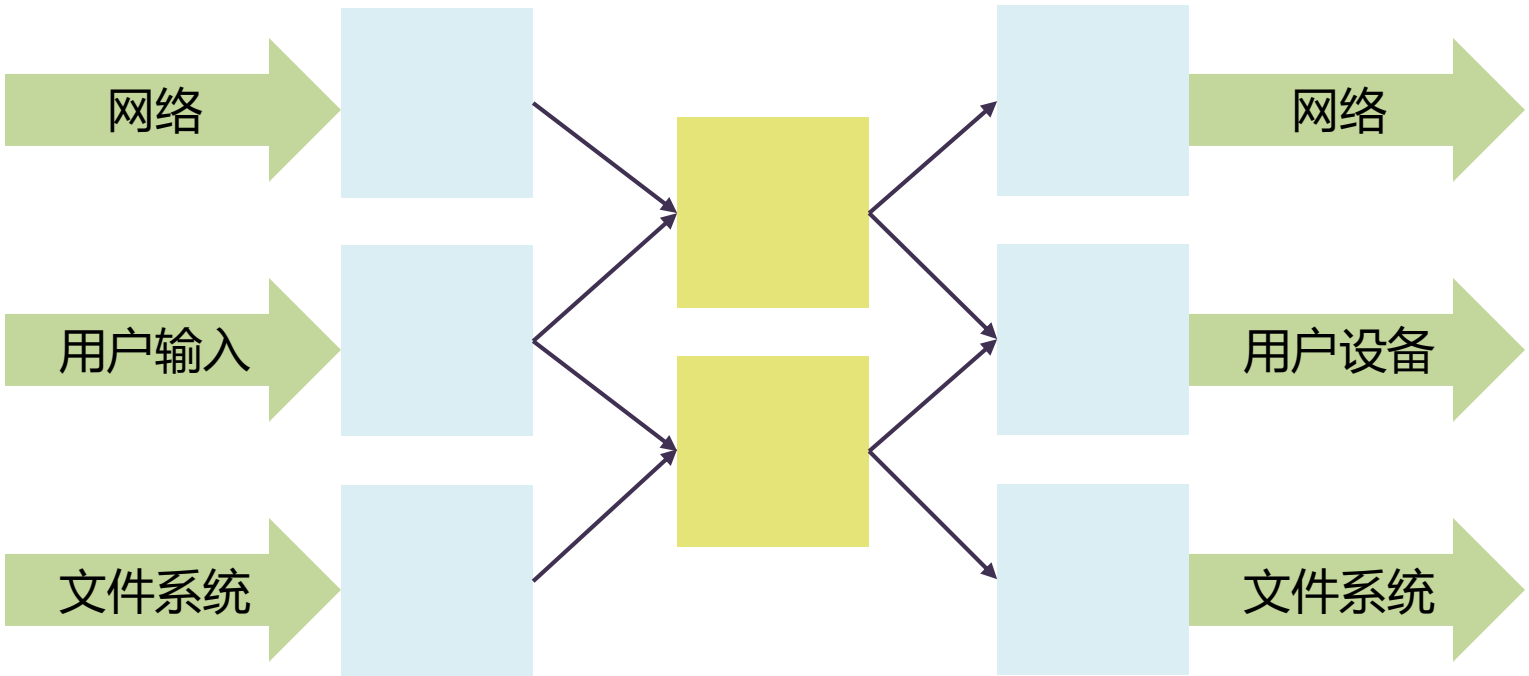
整体式设计（Monolithic design）

整体式设计（Monolithic design）

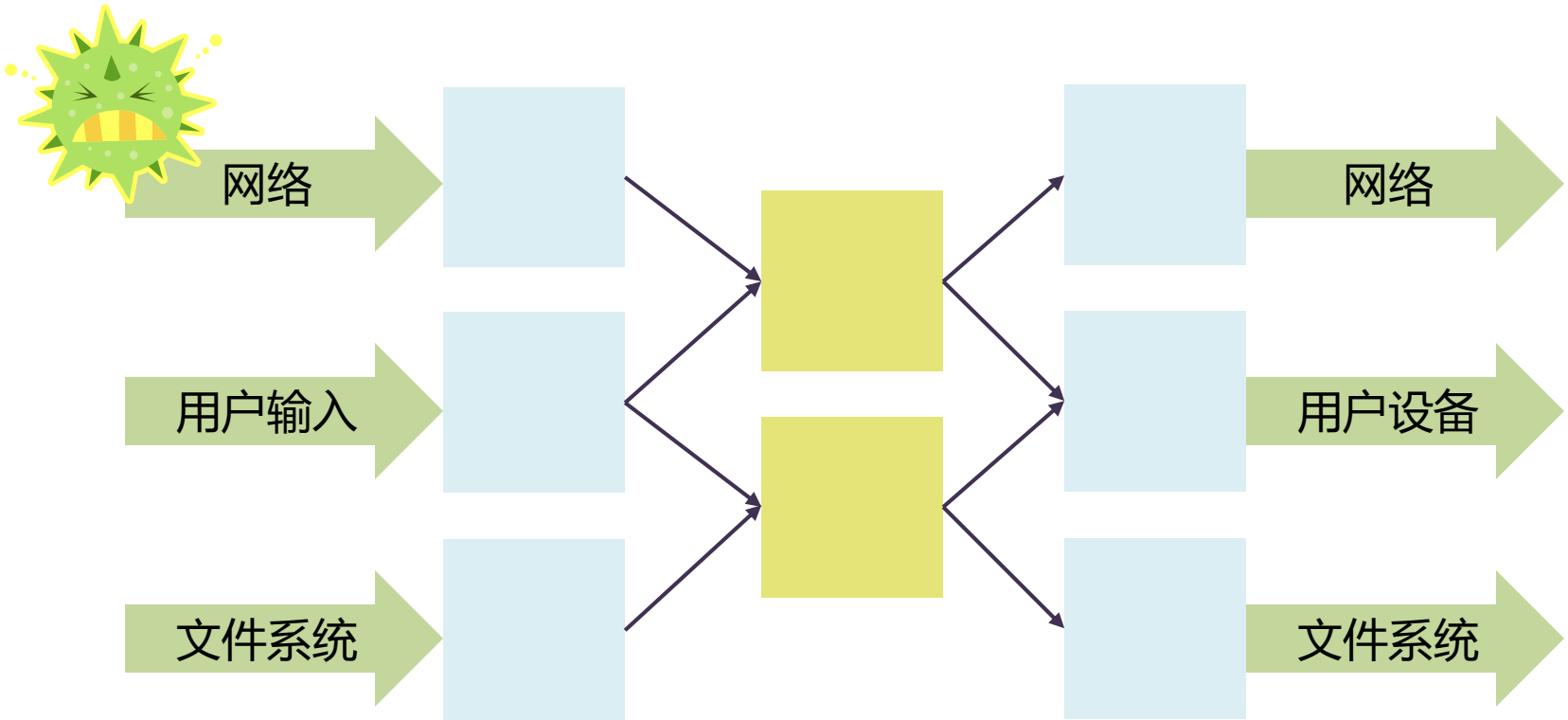


整体式设计（Monolithic design）

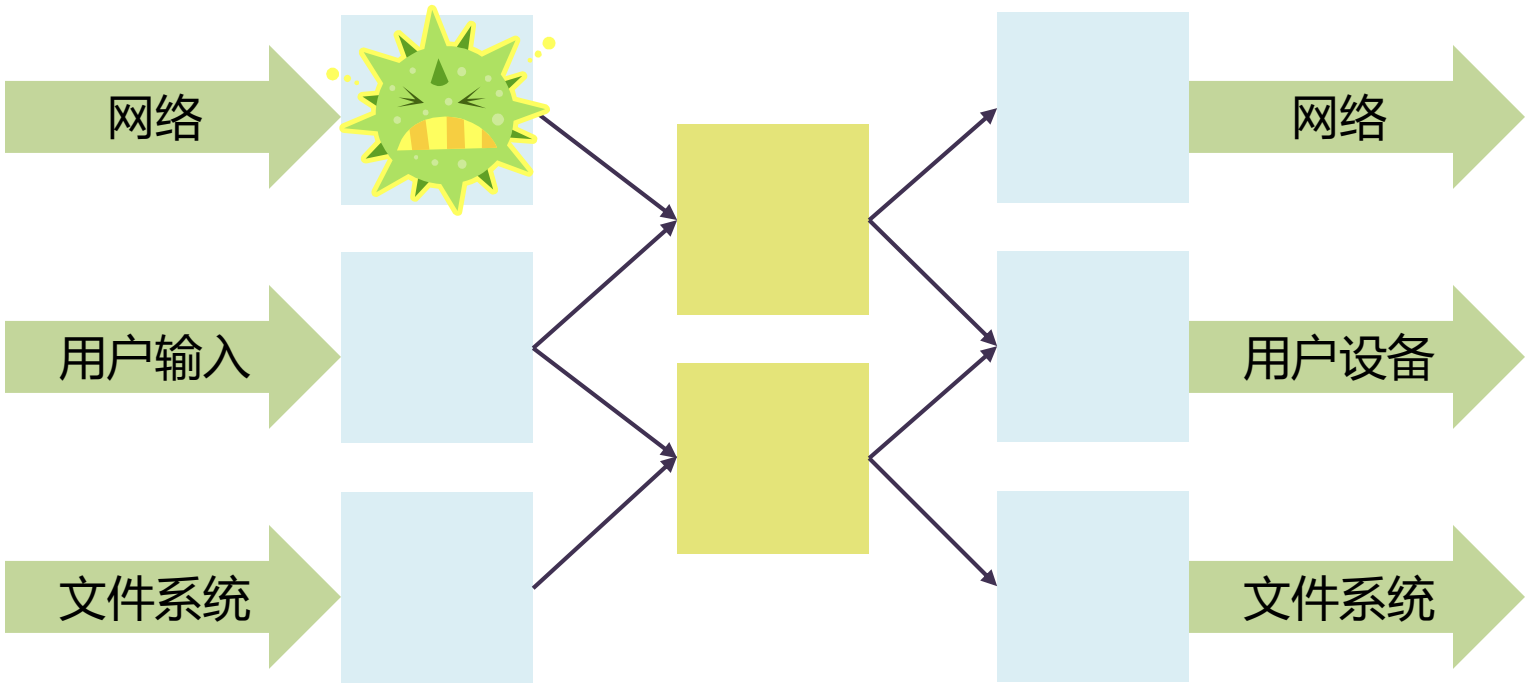
组件式设计（Component design）



组件式设计（Component design）



组件式设计（Component design）



最小特权原则

- 什么是特权?
  - 访问或修改系统资源的能力
- 假设存在划分（compartmentalization）和隔离（isolation）
  - 将系统分割为独立的模块
  - 限制模块间的交互
- 最小特权原则
  - 系统模块只能具有其预期目的所需的最小权限

举例：操作系统内核

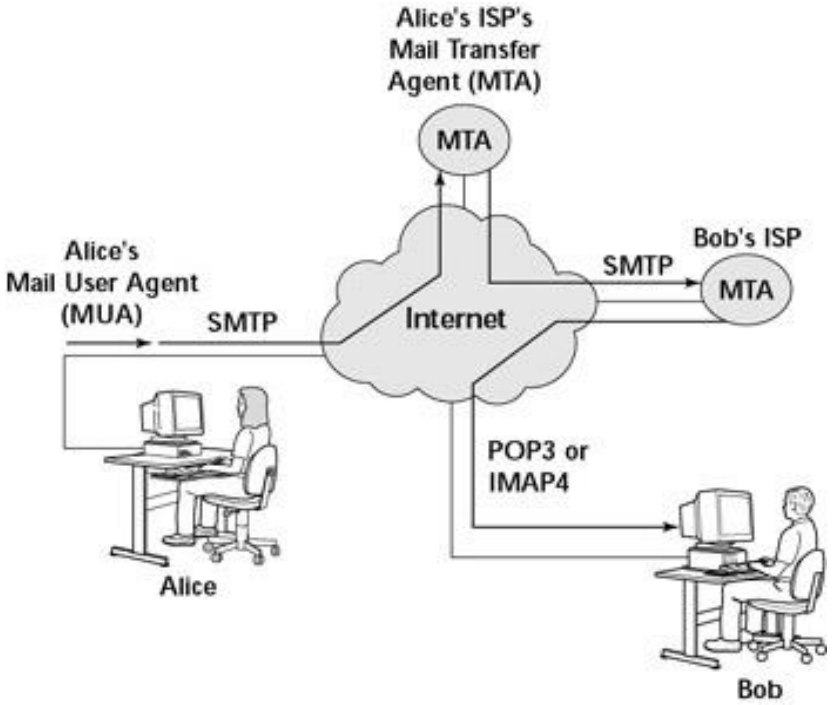
- 操作系统内核的两大设计阵营
  - 单内核（Micro kernel）
    - 把内核从整体上作为一个单独的大过程来实现，并同时运行在一个单独的地址空间；
    - 具有简单和高性能的特点；
  - 微内核（Monolithic kernel）
    - 内核的功能被划分为独立的过程；
    - 理想情况下，只有最基本的功能才运行在特权模式下，内存管理、设备管理、文件系统等功能都运行在用户空间，并保持独立地运行在各自的地址空间；
    - 充分的模块化、减小内存需求、高移植性；
    - 不能像单内核那样直接调用函数，而是通过消息传递处理微内核通信：
      - 采用进程间通信(IPC)机制，故有性能问题；

举例：邮件传输代理

- 邮件传输代理（MTA）
  - 通过网络接收和发送电子邮件
  - 将传入的电子邮件放入本地的用户收件箱

- 两个实例：
  - Sendmail
    - 基于传统的Unix
    - 整体式设计
    - 许多漏洞
  - Qmail
    - 划分的设计（Compartmentalized design）

简化的邮件传输



Most used mail transfer agents

MTA	Exim	Postfix	Sendmail	Qmail	Microsoft Exchange
Server OS support	Unix-like	Cross-platform	Cross-platform	Unix-like	Windows Server
License	GPLv2	IBM Public License	Sendmail License	Public domain	Proprietary or closed-source software

OS Basics

- 进程间隔离
  - 每个进程有一个UID
    - 具有相同UID的两个进程拥有相同权限
  - 进程可能会访问文件、sockets…
    - 根据UID授予进程权限

- 基于进程间隔离的划分
  - 由UID定义的Compartment
  - 由系统资源允许动作定义的Privileges

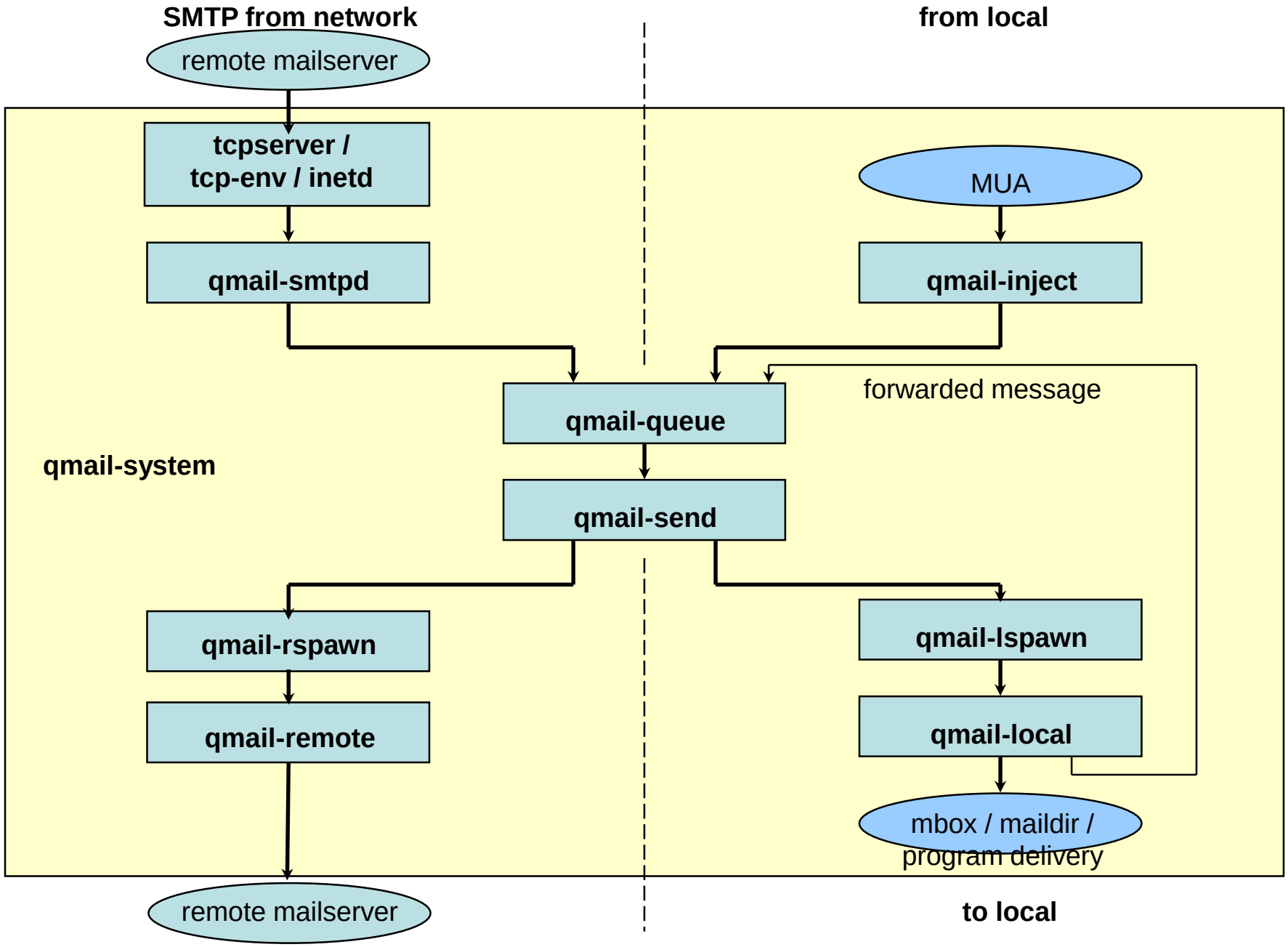
Qmail 设计

- 基于OS隔离机制的划分
  - 不同的模块作为操作系统不同用户运行
  - 每个用户只能访问指定的系统资源
- 最小特权
  - 每个UID分配最小权限
  - 只有一个“setuid”程序模块
    - 允许程序以不同的用户身份运行
  - 只有一个“root”程序模块
    - root程序拥有所有的特权

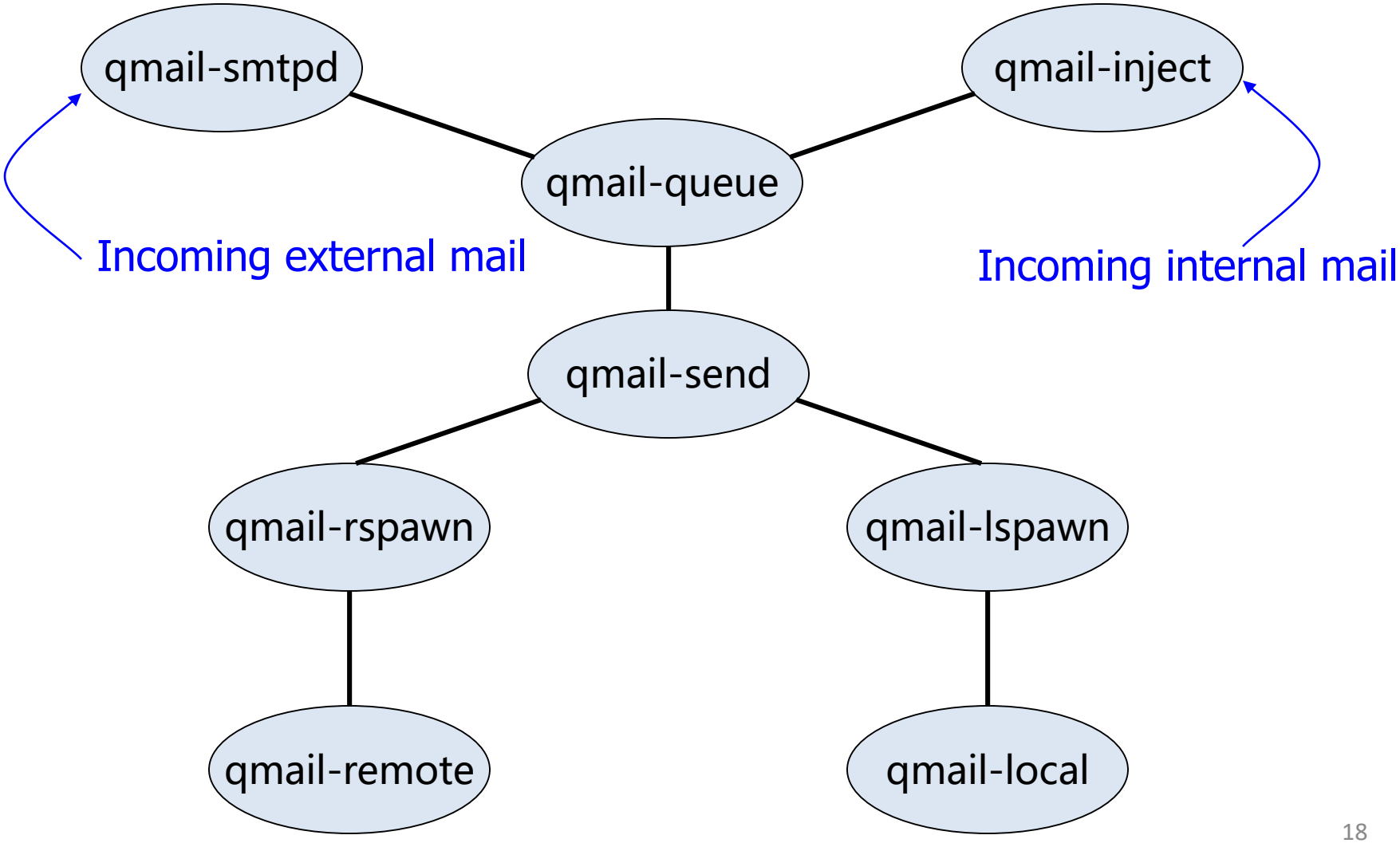
https://en.wikipedia.org/wiki/Qmail

qmail is a mail transfer agent (MTA) that runs on Unix. It was written, starting December 1995, by Daniel J. Bernstein as a more secure replacement for the popular Sendmail program. Originally license-free software, qmail's source code was later dedicated in the public domain by the author.[2]

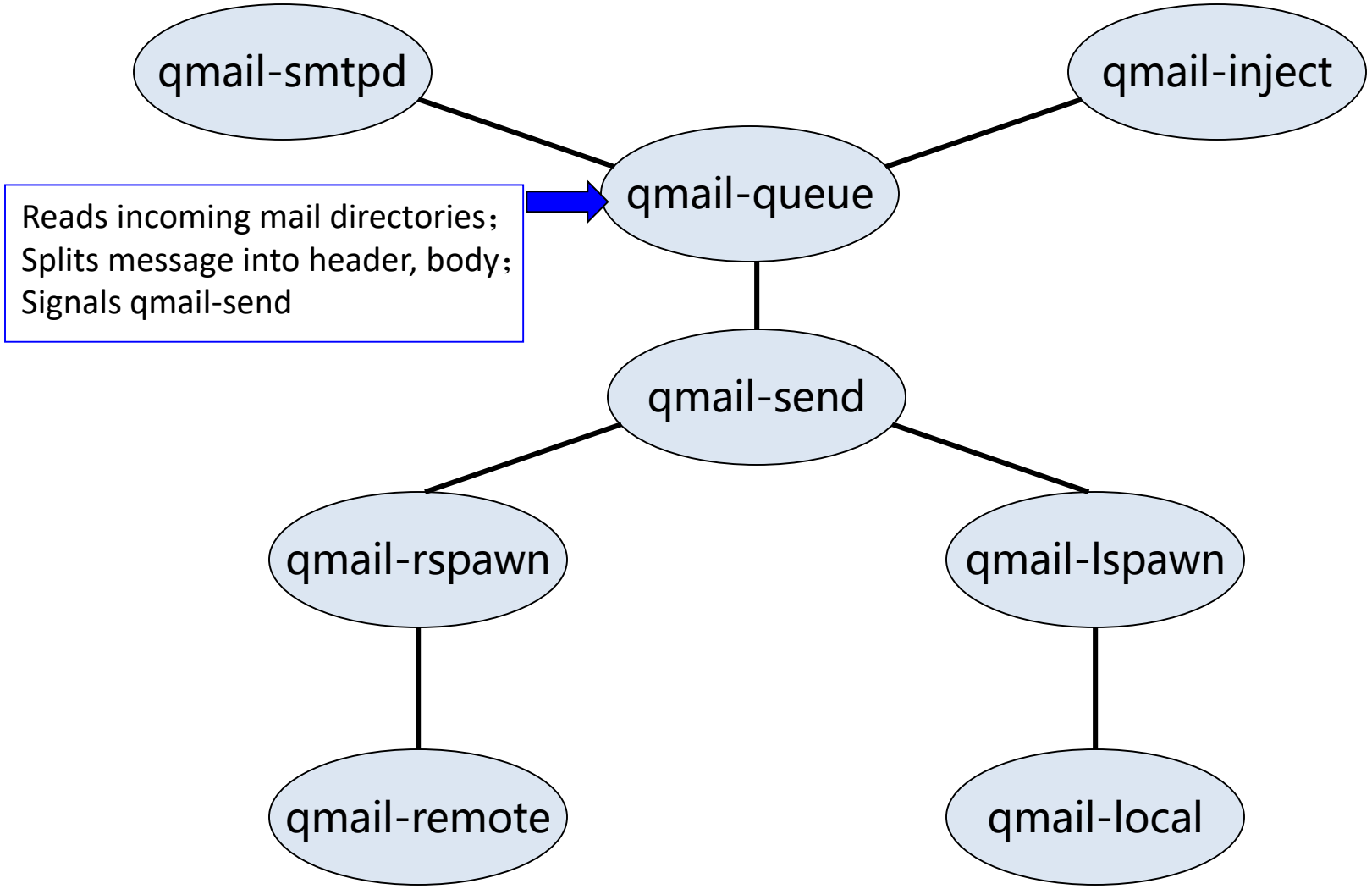
THE BIG Qmail PICTURE



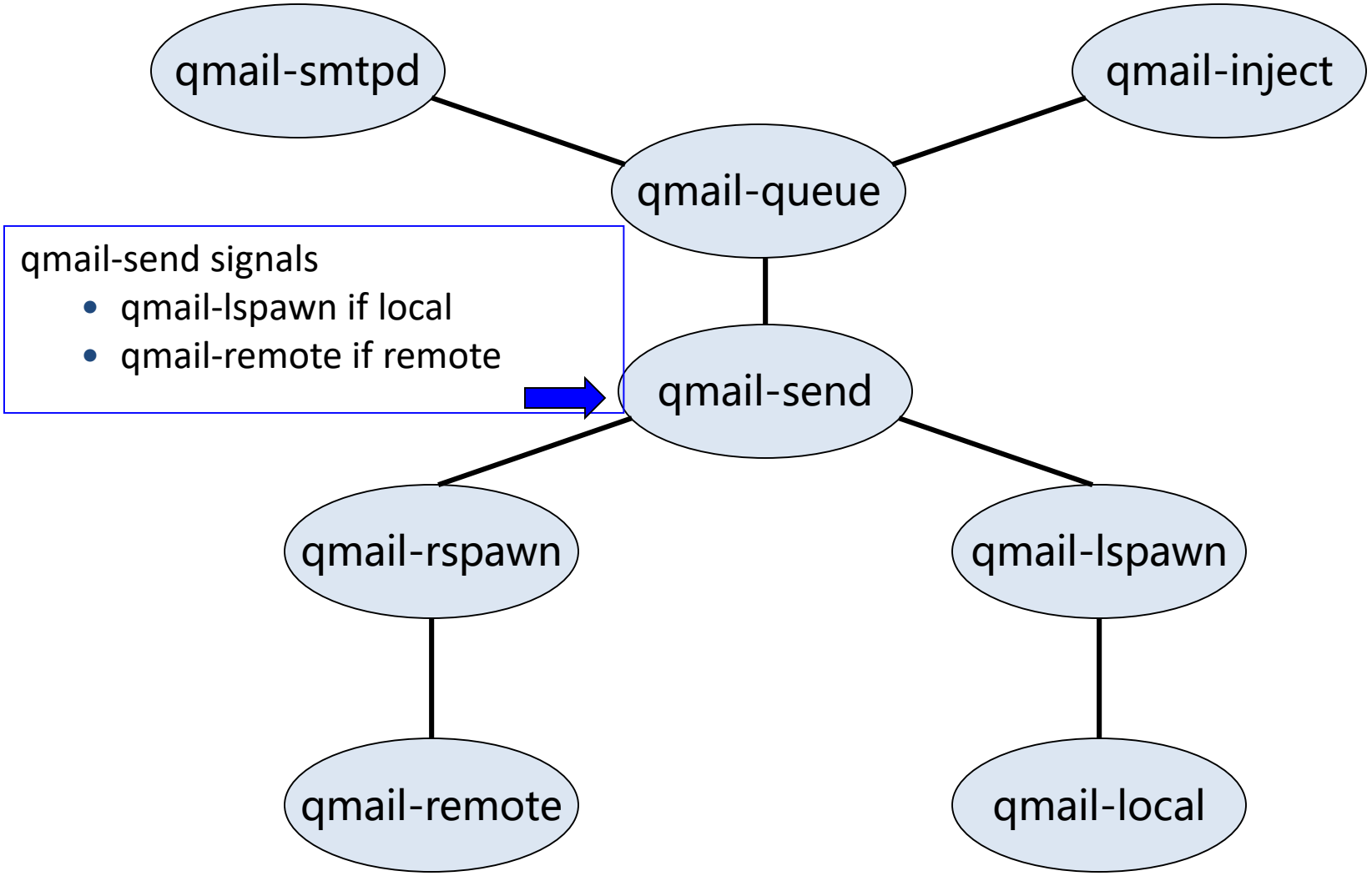
Qmail架构


qmail-smtpd
qmail-inject
qmail-queue
Incoming external mail
Incoming internal mail
qmail-send
qmail-rspawn
qmail-lspawn
qmail-remote
qmail-local

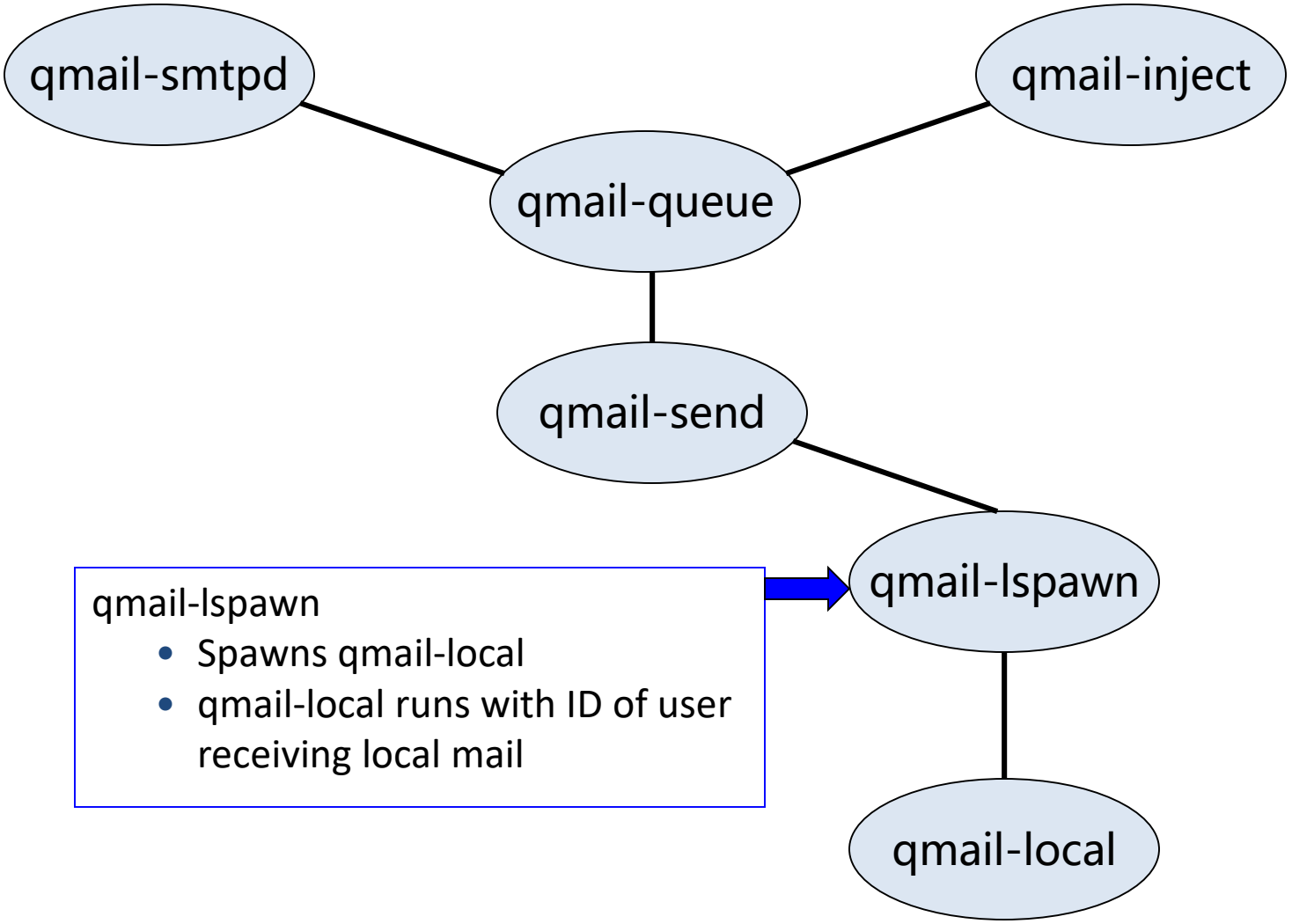
Qmail架构


qmail-smtpd
qmail-inject
qmail-queue
Reads incoming mail directories；
Splits message into header, body；
Signals qmail-send
qmail-send
qmail-rspawn
qmail-lspawn
qmail-remote
qmail-local

Qmail架构


qmail-smtpd
qmail-inject
qmail-queue
qmail-send signals
qmail-lspawn if local
qmail-remote if remote
qmail-send
qmail-rspawn
qmail-lspawn
qmail-remote
qmail-local

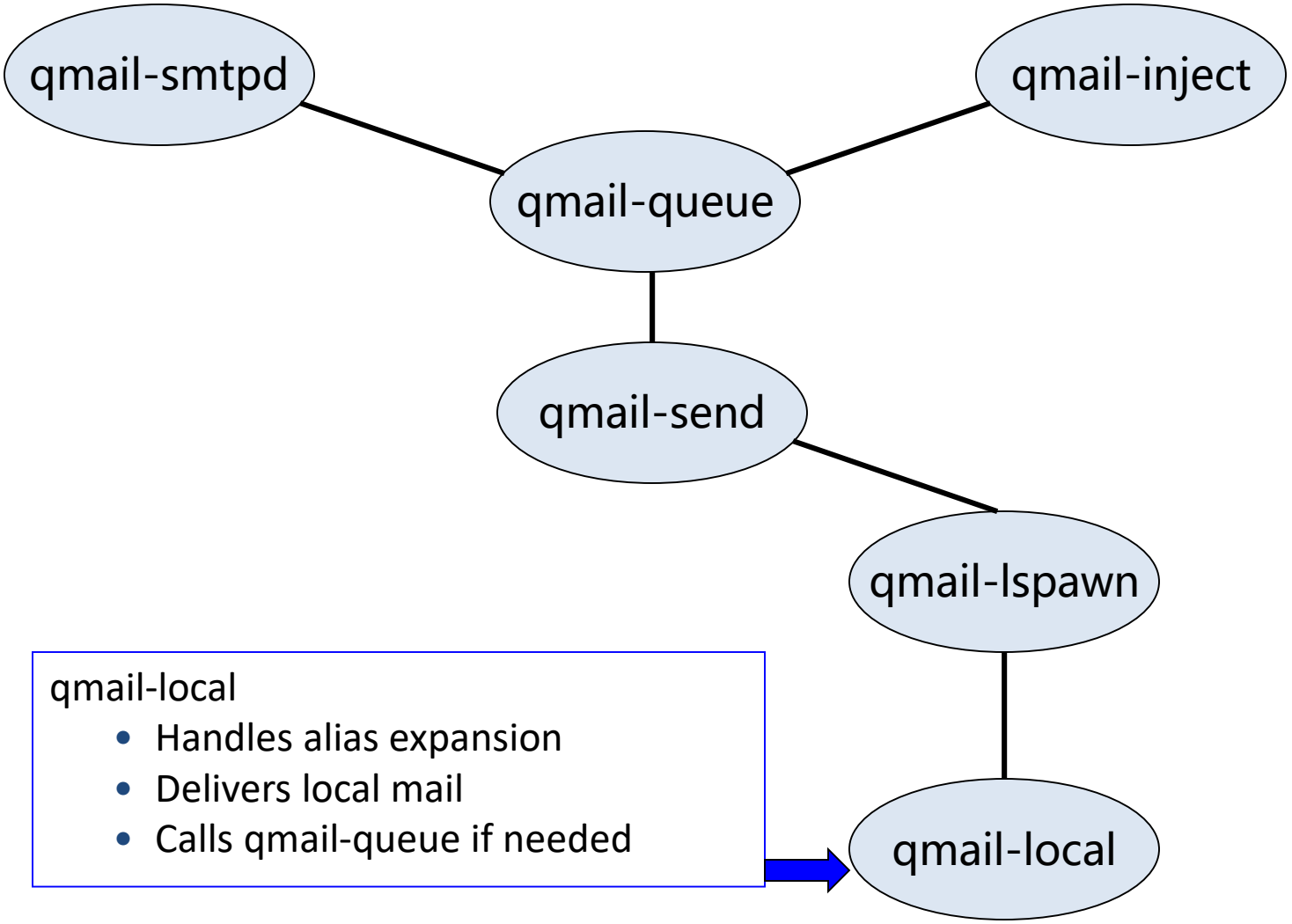
Qmail架构



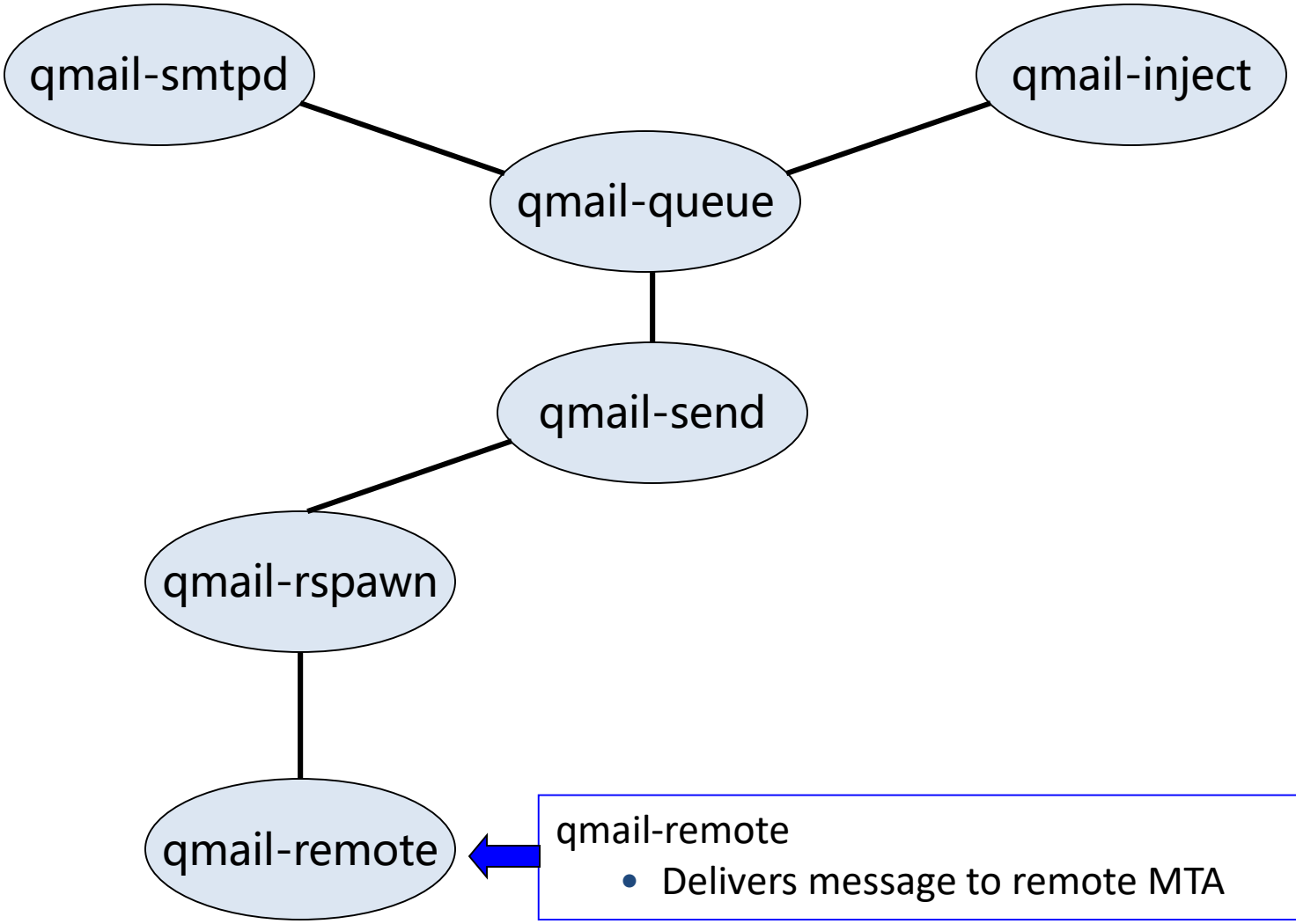
qmail-lspawn

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

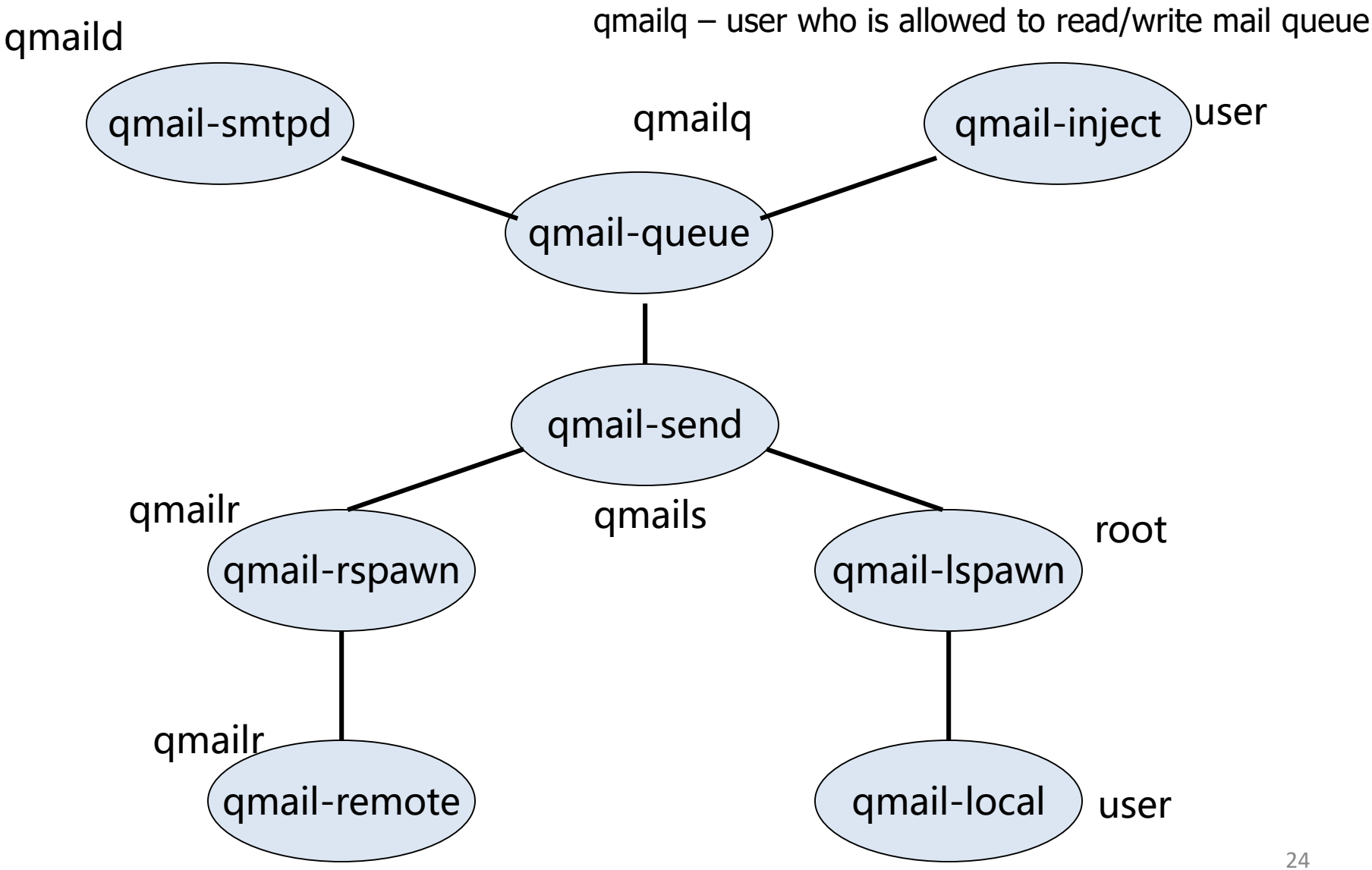
Qmail架构


qmail-smtpd
qmail-inject
qmail-queue
qmail-send
qmail-lspawn
qmail-local
qmail-local
• Handles alias expansion
• Delivers local mail
• Calls qmail-queue if needed

Qmail架构

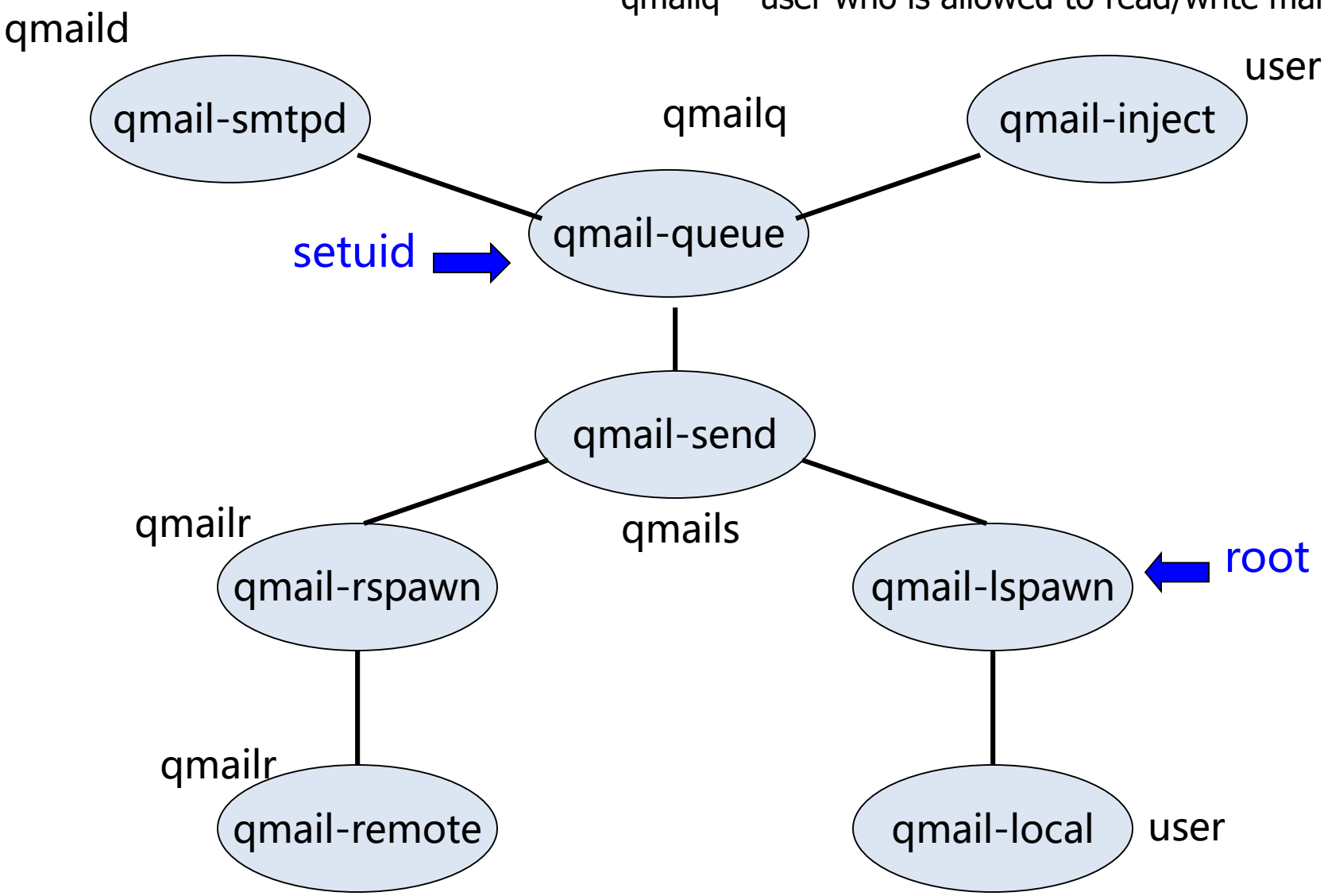

qmail-smtpd
qmail-inject
qmail-queue
qmail-send
qmail-rspawn
qmail-remote
qmail-remote
• Delivers message to remote MTA

通过Unix UIDs隔离



最小特权

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

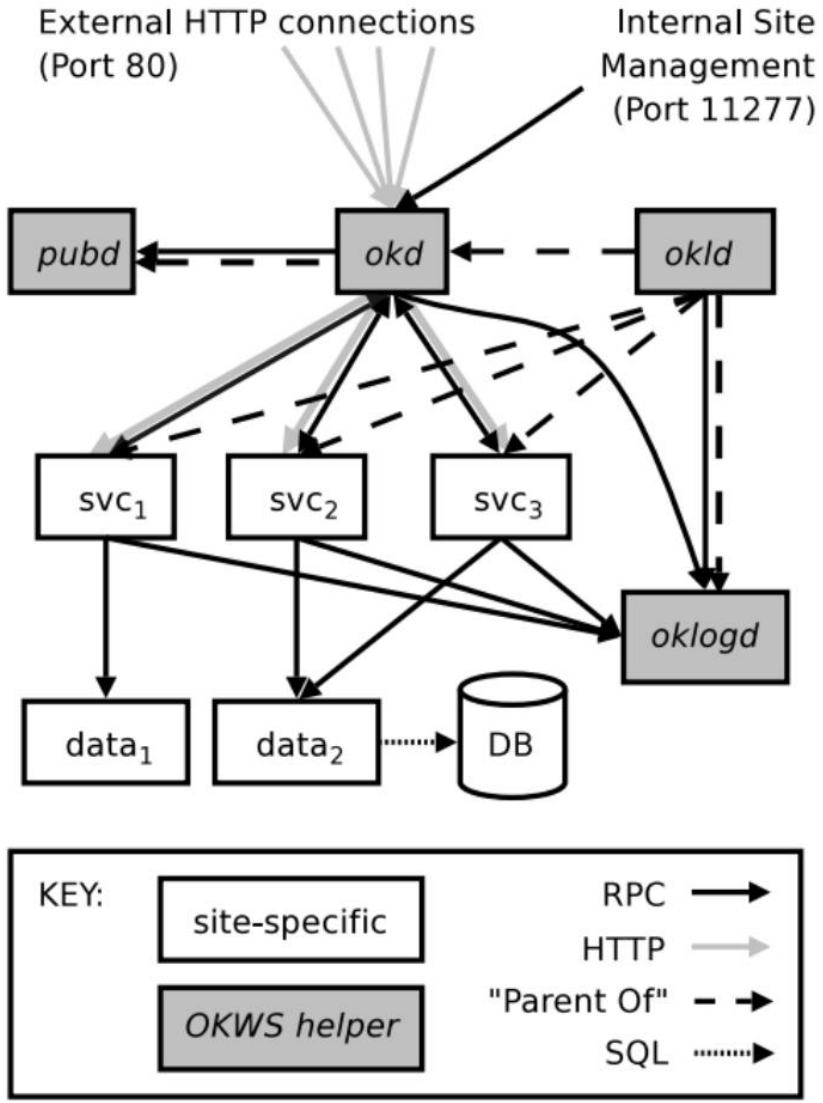


Sendmail vs qmail

- Do as little as possible in setuid programs
  - Of 20 recent sendmail security holes, 11 worked only because the entire sendmail system is setuid
  - Only qmail-queue is setuid
    - Its only function is add a new message to the queue
    - Setuid to the user ID of the qmail queue owner, not root
    - No setuid root binaries
- Do as little as possible as root
  - The entire sendmail system runs as root
    - Operating system protection has no effect
  - Only qmail-lspawn runs as root.

OKWS

- OKWS is a Web server, specialized for building fast and secure Web services



https://github.com/OkCupid/okws

https://vm-web.pdos.csail.mit.edu/papers/okws-usenix04.pdf

OKWS的设计

- 隔离: each service runs with own UID
  - Each service run in a *chroot* jail
  - Communication limited to structured RPC between service and DB
- 最小特权
  - Each *UID* is unique non privileged user
  - Only *okld* (launcher daemon) runs as root

OKWS的设计

process	*chroot* jail	run directory	uid	gid
okld	/var/okws/run	/	root	wheel
pubd	/var/okws/htdocs	/	www	www
oklogd	/var/okws/log	/	oklogd	oklogd
okd	/var/okws/run	/	okd	okd
svc_1	/var/okws/run	/cores/51001	51001	51001
svc_2	/var/okws/run	/cores/51002	51002	51002
svc_3	/var/okws/run	/cores/51003	51003	51003

Linux内核隔离技术：命名空间（namespace）

- **Namespace 定义：**

> Namespaces are a feature of the Linux kernel that partitions kernel resources such that one set of processes sees one set of resources while another set of processes sees a different set of resources. The feature works by having the same namespace for a set of resources and processes, but those namespaces refer to distinct resources.

- Namespace是Linux内核的一项特性，用于**进程间资源隔离**的一种技术
- 对内核资源进行分区，使不同的进程组可以看到不同分组的资源
- Linux 默认提供了多种 Namespace，**用于对多种不同资源进行隔离**

命名空间（namespace）

命名空间	宏	隔离内容
PID	CLONE_NEWPID	进程ID
Network	CLONE_NEWNET	网络设备、栈、端口等
Mount	CLONE_NEWNS	挂载点
IPC	CLONE_NEWIPC	System V IPC, POSIX消息队列
UTS	CLONE_NEWUTS	主机名和NIS域名
User	CLONE_NEWUSER	用户和组ID
Cgroups	CLONE_NEWCGROUP	Cgroups 目录视图

命名空间（namespace）

- **Namespace的历史过程**

2002-2013年间，Linux社区按需逐步实现 PID、 NET、 IPC、 UTS、 USER 等 namespaces

命名空间（namespace）

- **Namespace概述**

PID 命名空间

不同用户的进程就是通过PID命名空间隔离开的，不同PID命令空间中进程的PID号的计数完全独立

NET 命名空间

网络隔离是通过network命名空间实现的，每个net命名空间有独立的网络设备、IP 地址、路由表、/proc/net 目录

MNT 命名空间

mount命名空间允许不同命名空间的进程看到的文件结构不同；**联合chroot，可以将一个进程放到一个特定的目录执行**，即实现了进程文件系统的隔离

命名空间（namespace）

- IPC 命名空间

  Linux常见的进程间交互方法(inter process communication - IPC)， 包括**信号量、消息队列和共享内存**等。IPC命令空间对此类资源进行了隔离

- UTS 命名空间

  不同UTS(UNIX Time Sharing)命名空间中的进程可以拥有不同的主机名（hostname）、域名（domain name）和一些版本信息

- USER 命名空间

  不同USER命名空间中的进程可以有不同的用户ID和组ID （uid和gid）

命名空间创建

以下3个系统调用会被用于namespaces:

- `clone()`：创建新的进程时创建新的namespaces，并将新创建的进程加入到新的namespace里面

```
int clone (..., unsigned long, clone_flags, ...);
```

- 参数clone_flags表示使用哪些CLONE_ *标志位:

PID	CLONE_NEWPID
Network	CLONE_NEWNET
Mount	CLONE_NEWNS
IPC	CLONE_NEWIPC
UTS	CLONE_NEWUTS
User	CLONE_NEWUSER
Cgroups	CLONE_NEWCGROUP

命名空间创建

- `unshare()`：不会创建新的进程，但是会创建新namesapce，并把当前的进程加入到该namespace里面

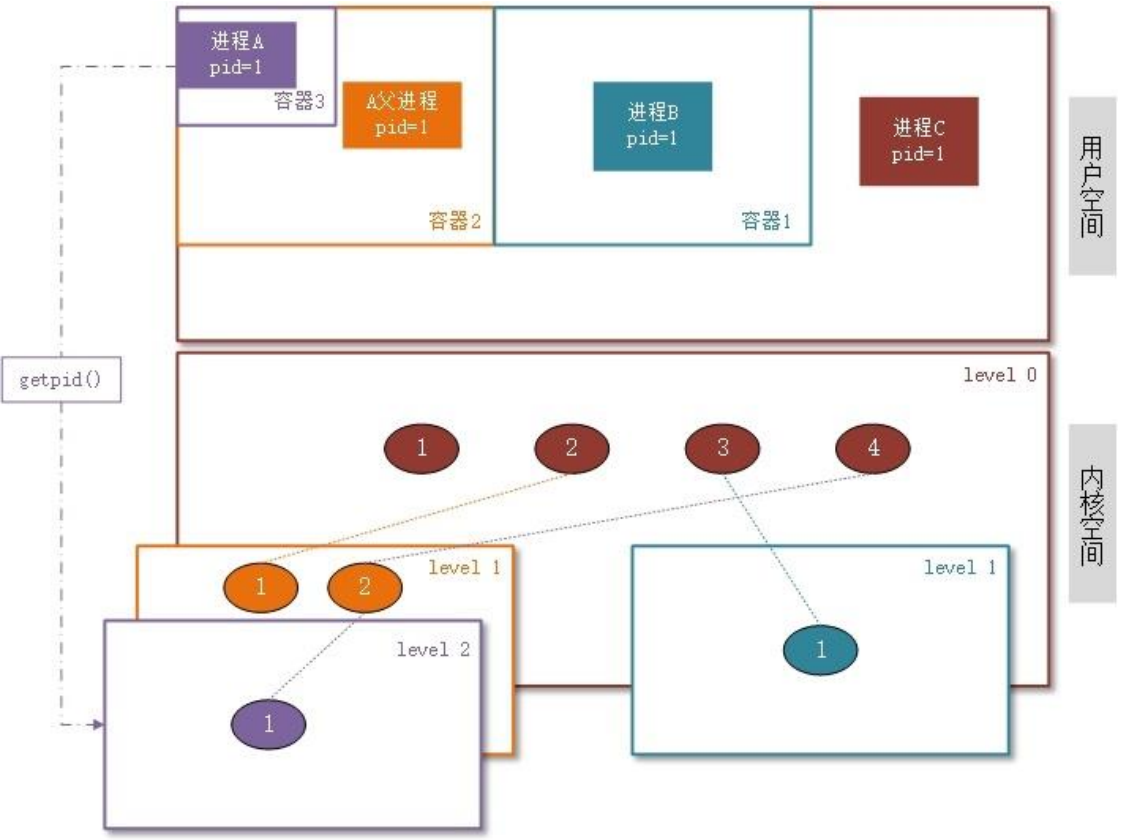
```
int unshare(int flags);
```

- `setns()`：将进程加入到一个已经存在的namespace里

```
int setns(int fd, int nstype);
```

  - 参数fd表示我们要加入的namespace的文件描述符。如：/proc/[pid]/ns下面对应的文件描述符
  - 参数nstype标识需要加入namespace的名称，setns会检查fd指向的namespace类型是否与该名称相符（0表示不检查）

PID Namespace



- 内核为所有的PID Namespace维护了一个**树状结构**，最顶层的是系统初始化创建的Root Namespace

- **PID Namespace形成一个等级体系**：父节点可以看到子节点中的进程，反过来子节点无法看到父节点PID Namespace里面的进程

PID Namespace

```
package main

import (
        "log"
        "os"
        "os/exec"
        "syscall"
)

func main() {
        cmd := exec.Command("sh")
        cmd.SysProcAttr = &syscall.SysProcAttr{
                Cloneflags: syscall.CLONE_NEWUTS | syscall.CLONE_NEWIPC | syscall.CLONE_NEWPID,
        }
        cmd.Stdin = os.Stdin
        cmd.Stdout = os.Stdout
        cmd.Stderr = os.Stderr
        if err := cmd.Run(); err != nil {
                log.Fatal(err)
        }
}
```

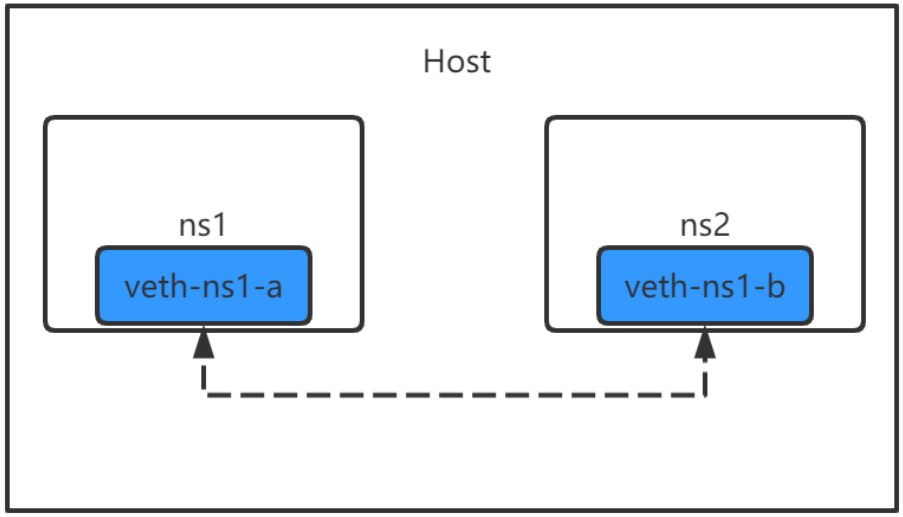
PID Namespace

```
#define STACK_SIZE (1024 * 1024)

static char child_stack[STACK_SIZE];
char* const child_args[] = {
  "/bin/bash",
  NULL
};

int child_main(void* args) {
  printf("Inside child process!\n");
  sethostname("NewNamespace",12);
  execv(child_args[0],child_args);
  return 1;
}

int main() {
  printf("Beginning:\n");
  int child_pid = clone(child_main,child_stack + STACK_SIZE, CLONE_NEWPID | CLONE_NEWIPC | CLONE_NEWUTS |
SIGCHLD,NULL);
  waitpid(child_pid,NULL,0);
  printf("Exiting\n");
  return 0;
}
```

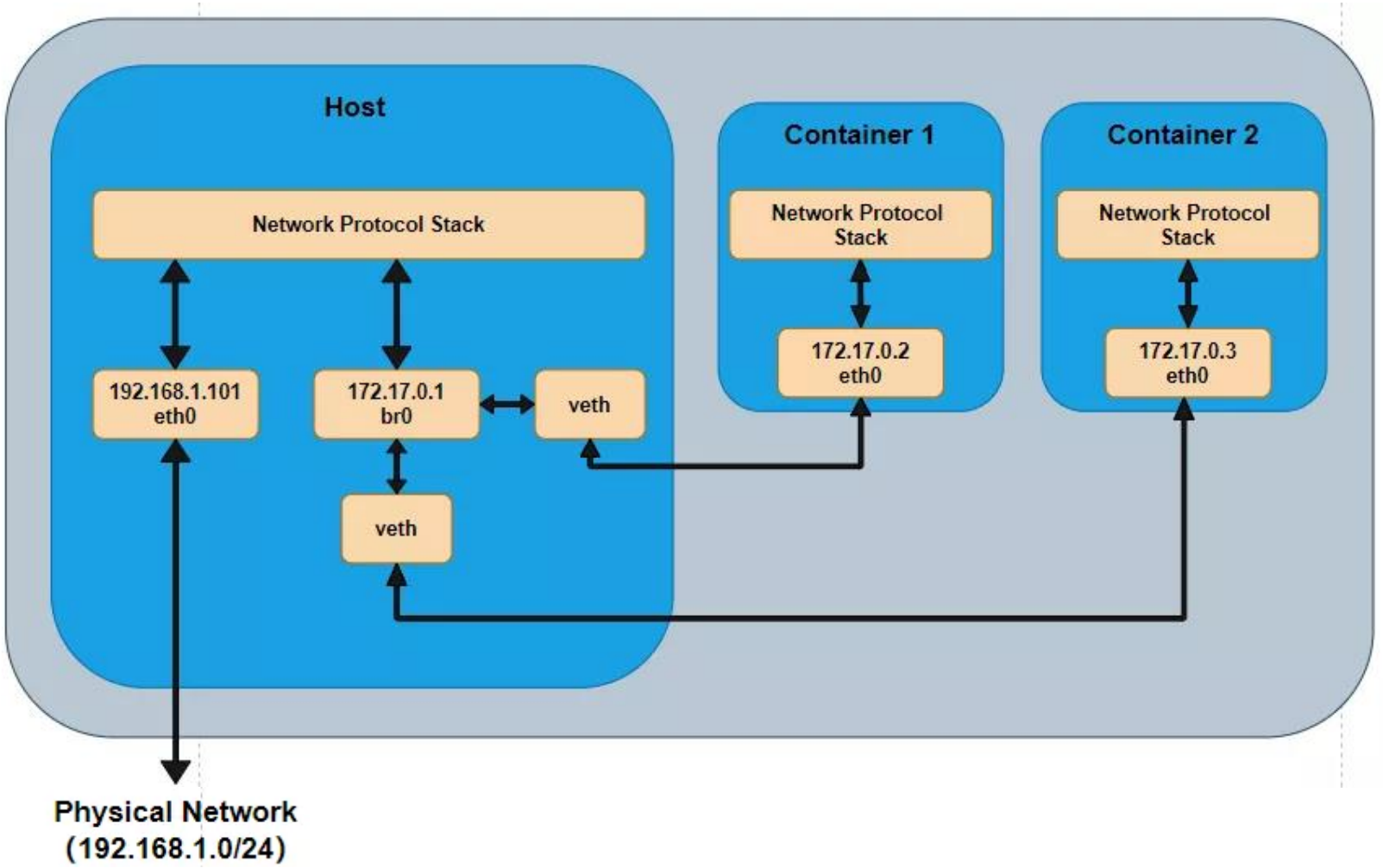
Network Namespace

Network namespaces 隔离了与网络相关的系统资源：

- network devices - 网络设备
- IPv4 and IPv6 protocol stacks - IPv4、IPv6 的协议栈
- IP routing tables - IP 路由表
- firewall rules - 防火墙规则
- /proc/net （即 /proc/<pid>/net）
- /sys/class/net
- /proc/sys/net 目录下的文件
- 端口、socket
- UNIX domain abstract socket namespace



Network Namespace

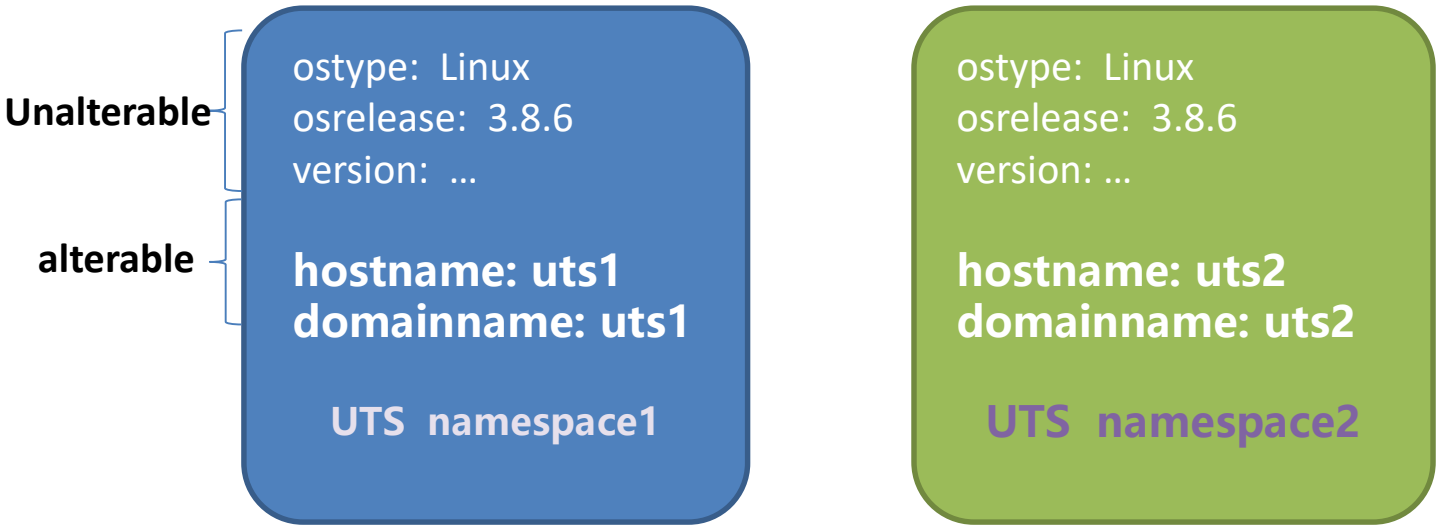


IPC Namespace

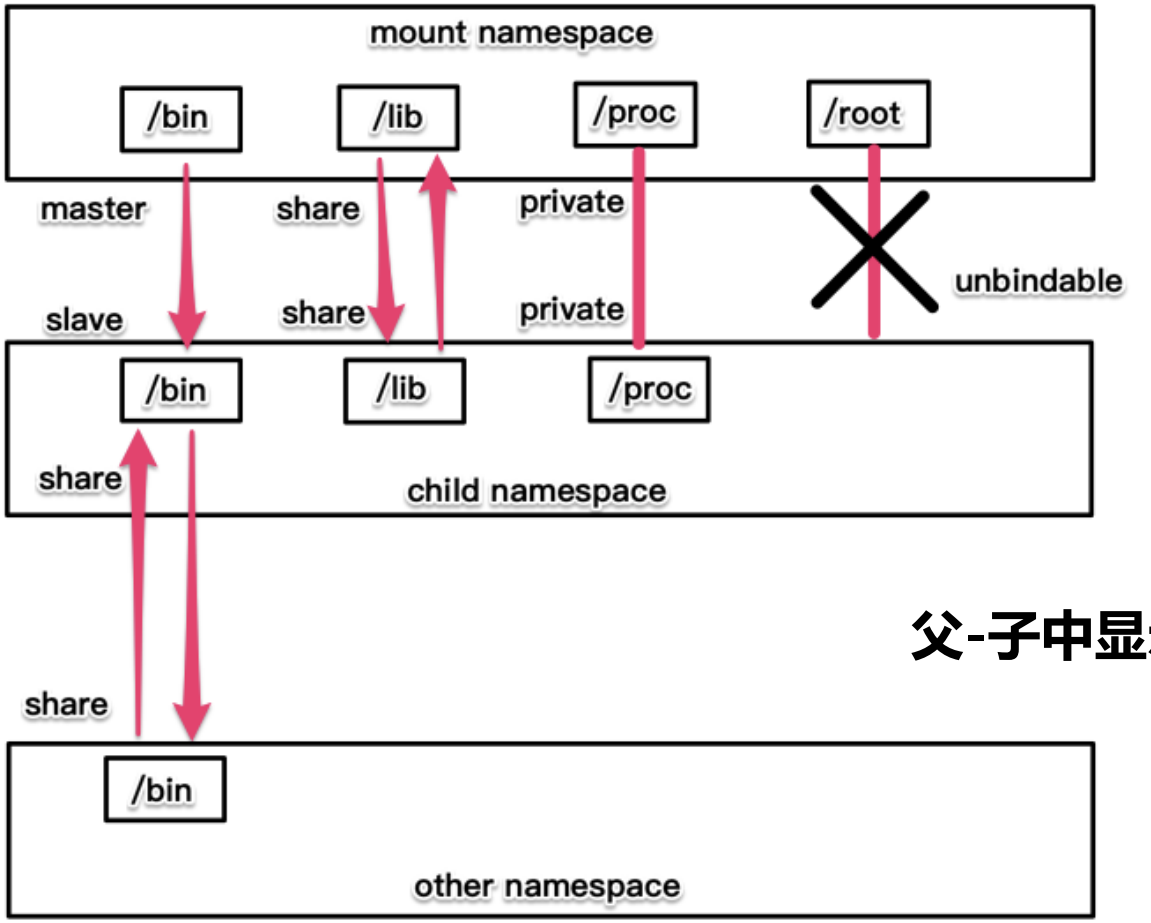
- **IPC Namespace 特性**
  - **IPC namespaces 隔离了 IPC 资源**，如 System V IPC objects
  - 每个 IPC namespace 都有着自己的一组 System V IPC 标识符，以及 POSIX 消息队列系统
  - **在IPC namespace中创建的对象**，对所有该namespace下的成员均可见，对其他 namespace下的成员均不可见
  - **当IPC namespace被销毁时**（空间里的最后一个进程都被停止删除时），在 IPC namespace 中创建的objects也会被销毁

UTS Namespace

UTS（UNIX Time-sharing System）命名空间允许每个容器拥有**独立的hostname和domain name**，使其在网络上可以被视作一个独立的节点而非主机上的一个进程



Mount Namespace

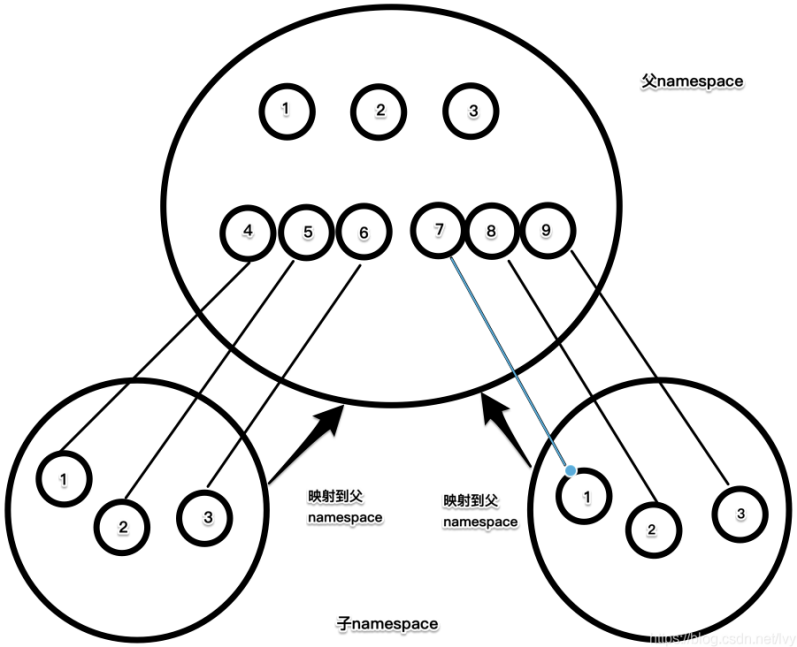


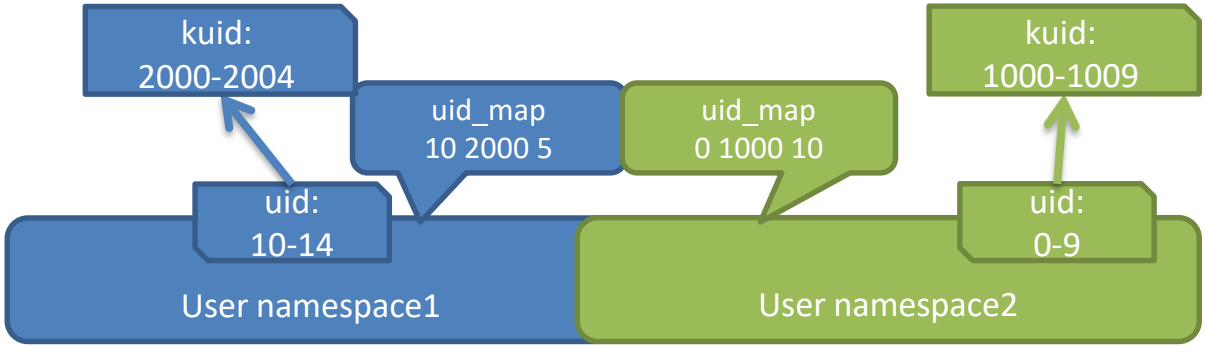
父-子中显示不同的文件系统视图

User Namespace

User Namespace允许**Namespace间可以映射用户和用户组ID**：

- 一个进程在Namespace里面的用户和用户组ID可以与Namespace外面的用户和用户组ID不同
- 一个普通进程(Namespace外面的用户ID非0)在Namespace里面的用户和用户组ID可以为0，即普通进程在Namespace里面可以拥有root特权的权限





Android 进程隔离

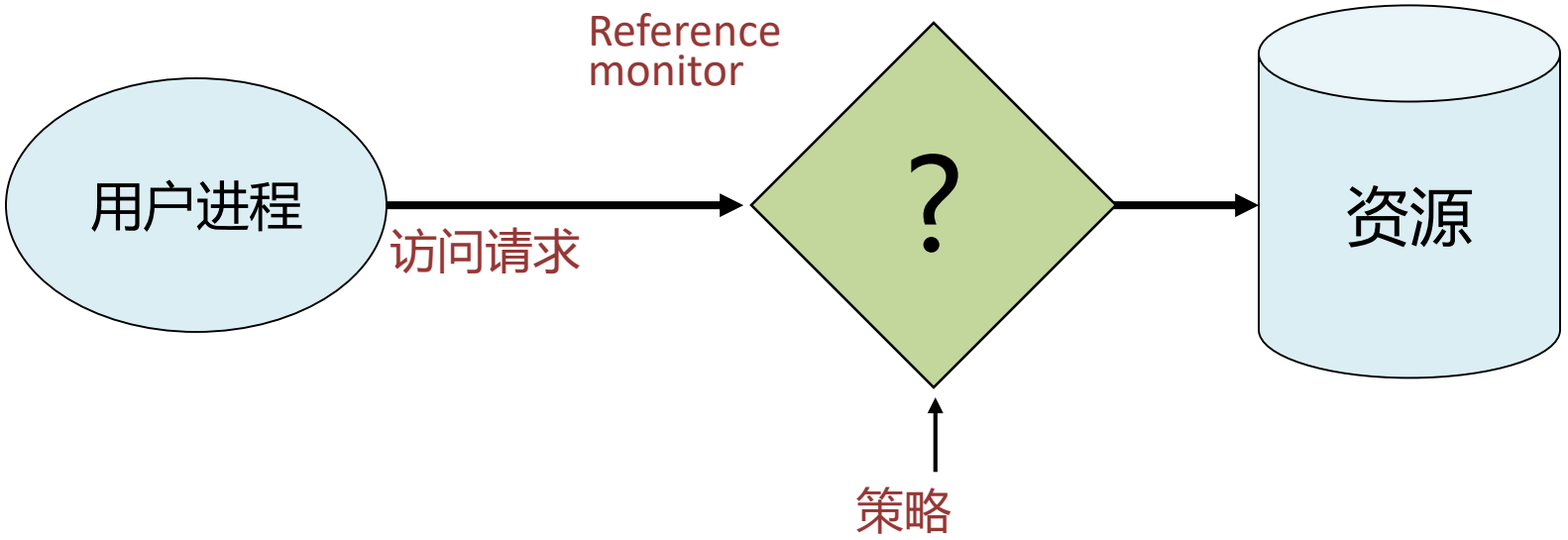
- Android 应用沙箱
  - 隔离：每个应用在单独的VM中运行，并具有唯一UID
    - 提供内存保护
    - 使用Unix域套接字保护通信
    - 只有 ping， zygote (产生其他进程)以root权限运行
  - 交互：引用监控器(reference monitor)检查组件间通信的权限
  - 最小权限：应用声明权限
    - 用户在安装时授予访问权限

大纲

- 隔离和最小特权
- 访问控制概念
- 操作系统机制
- 浏览器机制

访问控制

- 假设
  - 系统知道用户是谁
    - 通过用户名和密码认证，或其他凭证
  - 访问请求通过gatekeeper（Reference Monitor）
    - 系统不允许被绕过



访问控制矩阵[Lampson]

Objects

Subjects

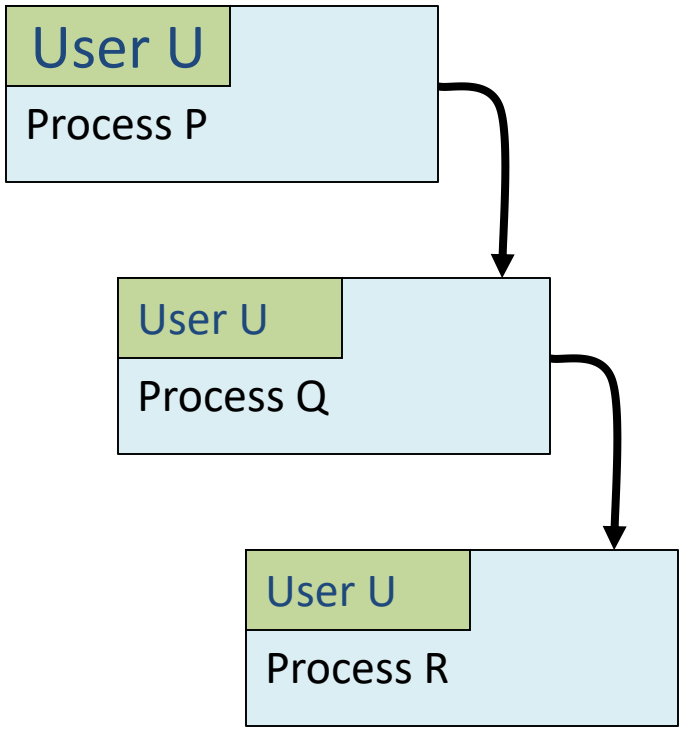
	File 1	File 2	File 3	…	File n
User 1	read	write	-	-	read
User 2	write	write	write	-	-
User 3	-	-	-	read	read
…					
User m	read	write	read	write	read

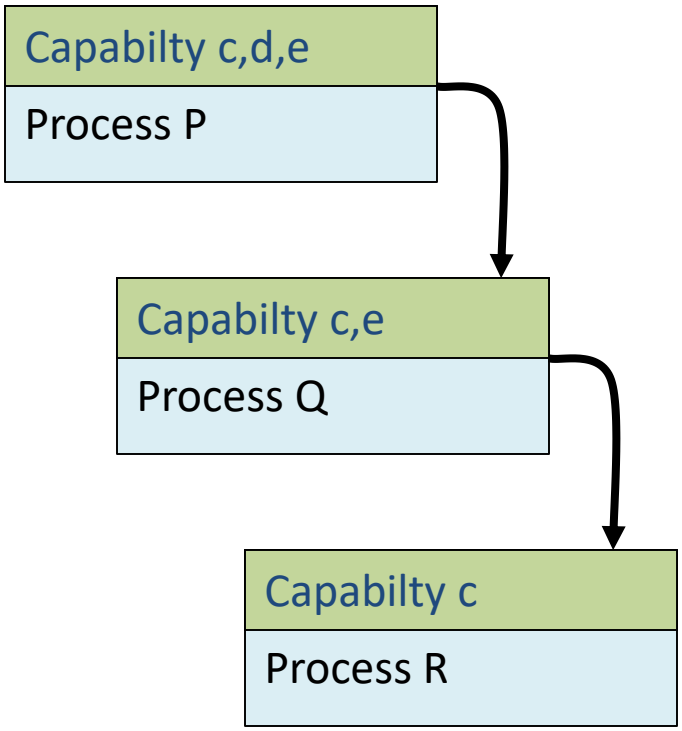
访问控制列表 vs 访问能力表

- 访问控制列表
  - 将每个对象和列表关联
  - 根据列表检查用户/组
  - 依赖身份认证：需要知道用户

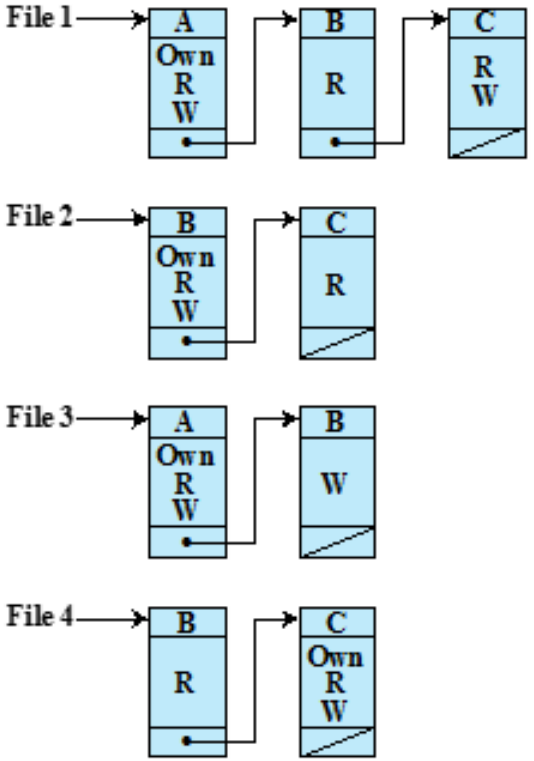
- 访问能力表
  - 访问能力表是不可伪造的标签(ticket)
    - 可以从一个进程传递给另一个进程
  - Reference monitor检查标签
    - 不需要知道用户/进程的身份

访问控制列表 vs 访问能力表

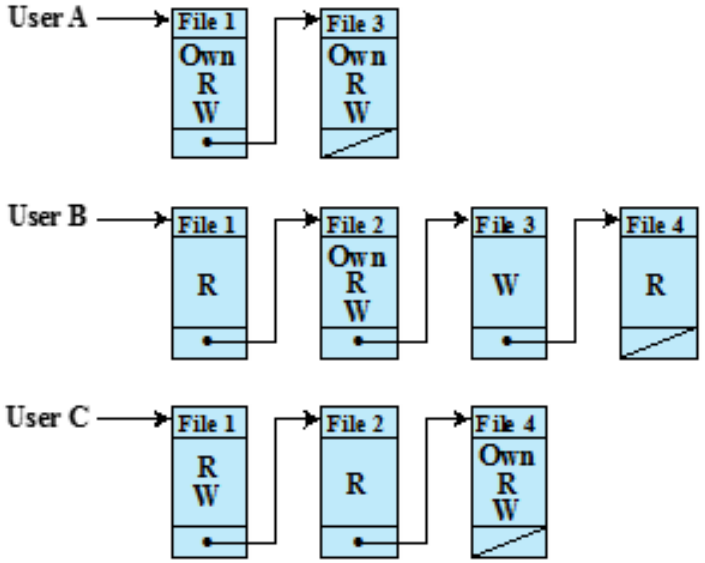




访问控制列表 vs 访问能力表



(b) Access control lists for files of part (a)



(c) Capability lists for files of part (a)

访问控制列表 vs 访问能力表

- 委托代理(Delegation)
  - 访问能力表(Cap)：进程可以在运行时传递访问能力
  - 访问控制列表(ACL)：尝试让owner添加权限到列表中？
    - 更常见的是，让其他进程在当前用户下执行（setuid）
- 撤销
  - 访问控制列表(ACL)：从列表中移除用户或组
  - 访问能力表(Cap)：尝试从进程中撤回访问能力？

大纲

- 隔离和最小特权
- 访问控制概念
- 操作系统机制
- 浏览器机制

Unix 访问控制

- 进程具有用户id
  - 从父进程继承
  - 进程可以改变id
    - 受限的选项集
  - 特殊的“root”id
    - 绕过访问控制限制

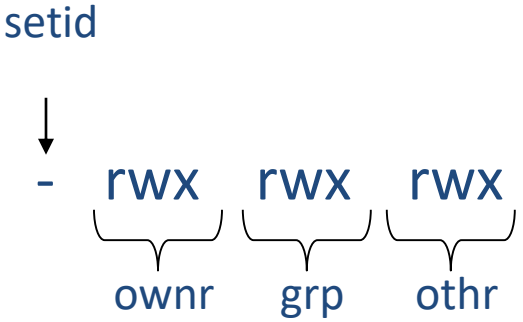
	文件1	文件2	…
用户 1	读	写	-
用户 2	写	写	-
用户 3	-	-	读
…			
用户 m	读	写	写

- 文件具有访问控制列表(ACL)
  - 授予用户ID权限
  - 所有者，组，其他

如：-rwxrwxr-x 1 demo demo 9216 Apr 15 22:12 test

Unix 文件访问控制列表

- 每个文件有三种用户
  - 所有者、组、其它（owner, group, other）
- 设置的权限
  - Read, write, execute
  - 由四个八进制值的向量表示
- 只有owner、root可以改变权限
  - 此权限不能委托或共享
- setid 位



进程的UID

- 每个进程有3个id
  - Real user ID (RUID)
    - 与父进程的用户ID相同 (除非改变)
    - 用来决定哪个用户启动进程
  - Effective user ID (EUID)
    - 来自被执行程序文件的setuid位，或者与父进程相同（setuid未设置的情况）
    - 决定进程的权限
      - 文件访问和端口绑定
  - Saved user ID (SUID)
    - 用于保存和恢复EUID
- Real group ID, Effective group ID, 类似使用

进程操作和IDs

- root
  - ID=0 代表超级用户 root；可以访问任何文件
- fork and exec
  - 继承3种IDs
  - exec带有setuid位的程序文件，euid变为文件的owner
- setuid 系统调用
  - seteuid(newid) 可以设置EUID为：
    - RUID或SUID
    - 如果EUID为0，EUID可以设置成任意ID
  - 几个不同的调用： setuid，seteuid，setreuid, setresuid

http://man7.org/linux/man-pages/man2/seteuid.2.html

可执行文件的setid 位

- 3种setid 位

chmod 4755 your_program

  - setuid – 将进程EUID设置为文件所有者ID
  - setgid – 将进程EGID设置为文件GID
  - 粘滞位(sticky)
    - 关闭：如果用户有目录的写权限，即使不是所有者，也可以重命名或删除文件
    - 开启：只有文件所有者、目录所有者以及root可以重命名或删除目录中文件

SUID案例：passwd 命令

当普通用户尝试修改密码，可以使用**passwd命令**，此命令文件的所有者是root。

passwd命令会尝试编辑系统配置文件，如/etc/passwd，/etc/shadow等。

所以passwd命令文件设置为SUID，给普通用户root权限，这样就可以更新/etc/shadow以及其他文件。

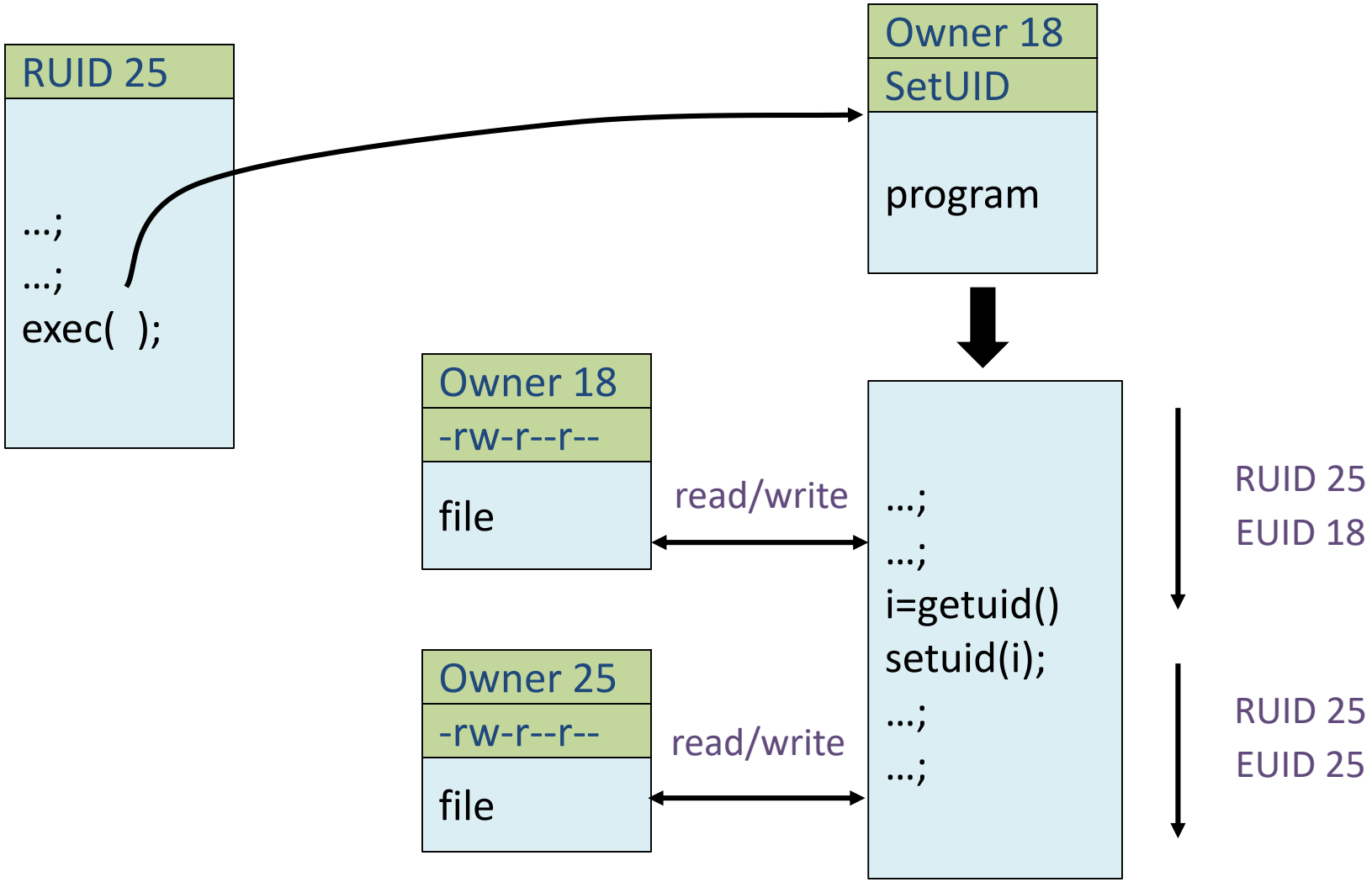
粘滞位(sticky)

- 当目录被设置了粘滞位权限以后，即便用户对该目录有写入权限，也不能删除该目录中其他用户的文件数据
- 设置了粘滞位之后，可以保持一种动态的平衡：允许各用户在目录中任意写入、删除数据，但是禁止随意删除其他用户的数据

```
# ls -ld /var/tmp
drwxrwxrwt  2   sys   sys   512   Jan 26 11:02  /var/tmp
```

```
- T refers to when the execute permissions are off.
- t refers to when the execute permissions are on.
```

举例

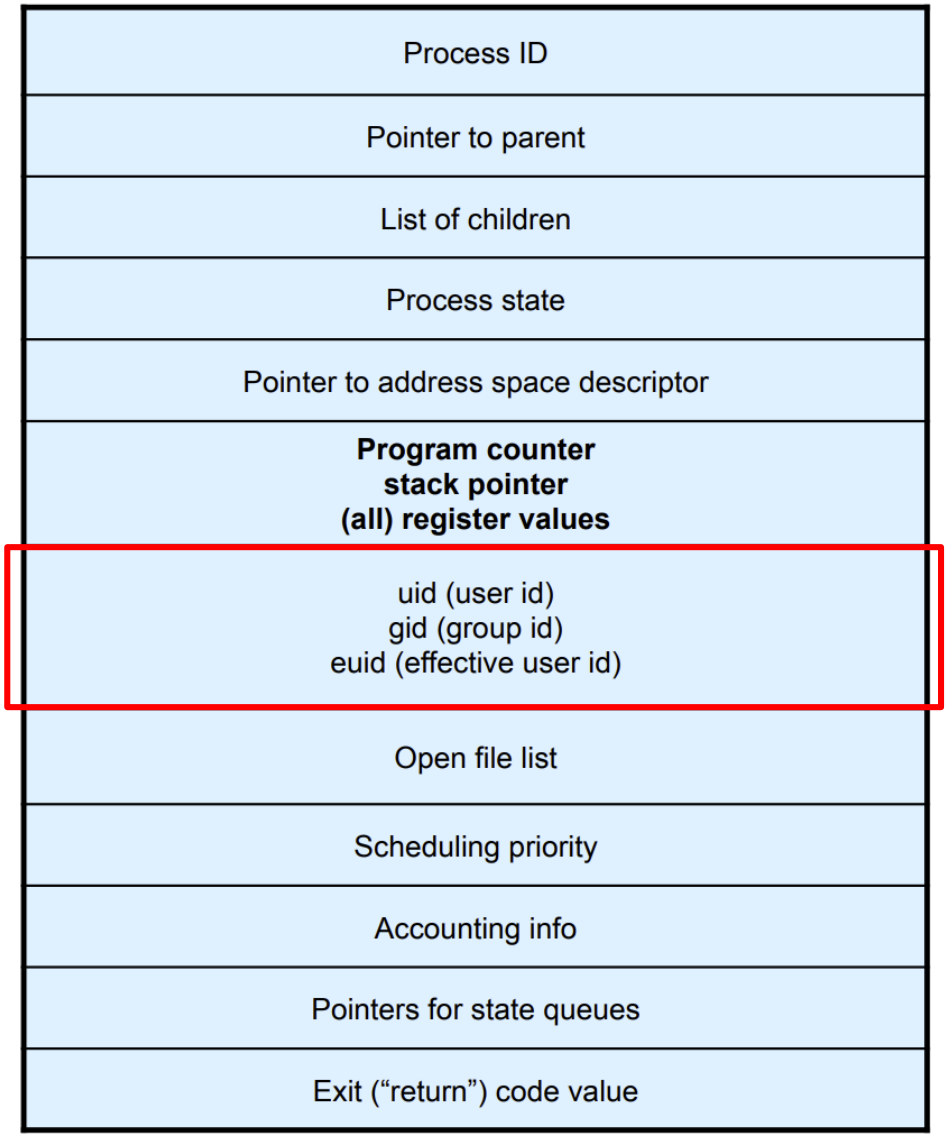


getuid() returns the real user ID of the calling process.

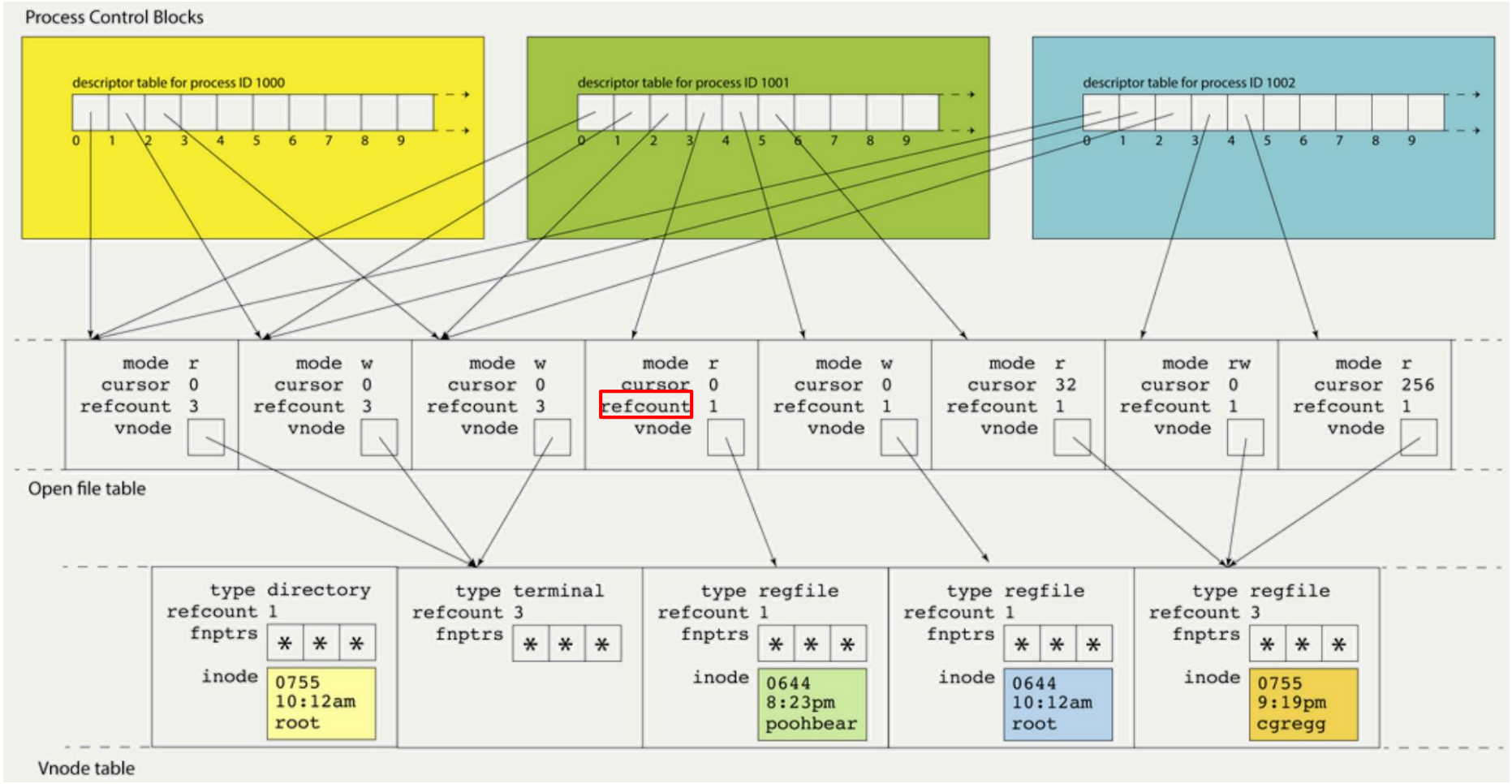
setuid() sets the effective user ID of the calling process.

IDs的位置

进程控制块
(PCB)



PCB和文件


Process Control Blocks
descriptor table for process ID 1000
descriptor table for process ID 1001
descriptor table for process ID 1002
0 1 2 3 4 5 6 7 8 9
mode r
cursor 0
refcount 3
vnode
mode w
cursor 0
refcount 3
vnode
mode w
cursor 0
refcount 3
vnode
mode r
cursor 0
refcount 1
vnode
mode w
cursor 0
refcount 1
vnode
mode r
cursor 32
refcount 1
vnode
mode rw
cursor 0
refcount 1
vnode
mode r
cursor 256
refcount 1
vnode
Open file table
type directory
refcount 1
fnptrs
inode
0755
10:12am
root
type terminal
refcount 3
fnptrs
type regfile
refcount 1
fnptrs
inode
0644
8:23pm
poohbear
type regfile
refcount 1
fnptrs
inode
0644
10:12am
root
type regfile
refcount 3
fnptrs
inode
0755
9:19pm
cgregg
Vnode table

setuid 编程

- 小心使用Setuid 0！
  - root可以做任何事；
  - 最小特权原则 – 当不再需要root权限时，改变EUID

```
// https://launchpadlibrarian.net/240241543/dash_0.5.8-2.1ubuntu2.diff.gz
// main() function in main.c has following changes:

++ uid = getuid();
++ gid = getgid();

++ /*
++  * To limit bogus system(3) or popen(3) calls in setuid binaries,
++     * require -p flag to work in this situation.
++  */
++ if (!pflag && (uid != geteuid() || gid != getegid())) {   ①
++         setuid(uid);
++         setgid(gid);
++         /* PS1 might need to be changed accordingly. */
++         choose_ps1();
++ }
```

dash shell针对setuid的保护

Unix 总结

- 优点
  - 保护大多数用户
  - 灵活性高
- 问题
  - 使用root权限太危险
  - 无法假定部分root权限，而不是所有的root权限

隔离、特权的弱点

- 面向网络的守护进程
  - 具有网络端口的root进程对所有的远程客户端开放，如sshd，ftpd，sendmail…
- Rootkits
  - 通过动态加载内核模块进行系统扩展
- 环境变量
  - 系统变量，如LIBPATH，在多个应用间共享状态。攻击者可以改变LIBPATH，从而加载攻击者提供的动态库

隔离、特权的弱点

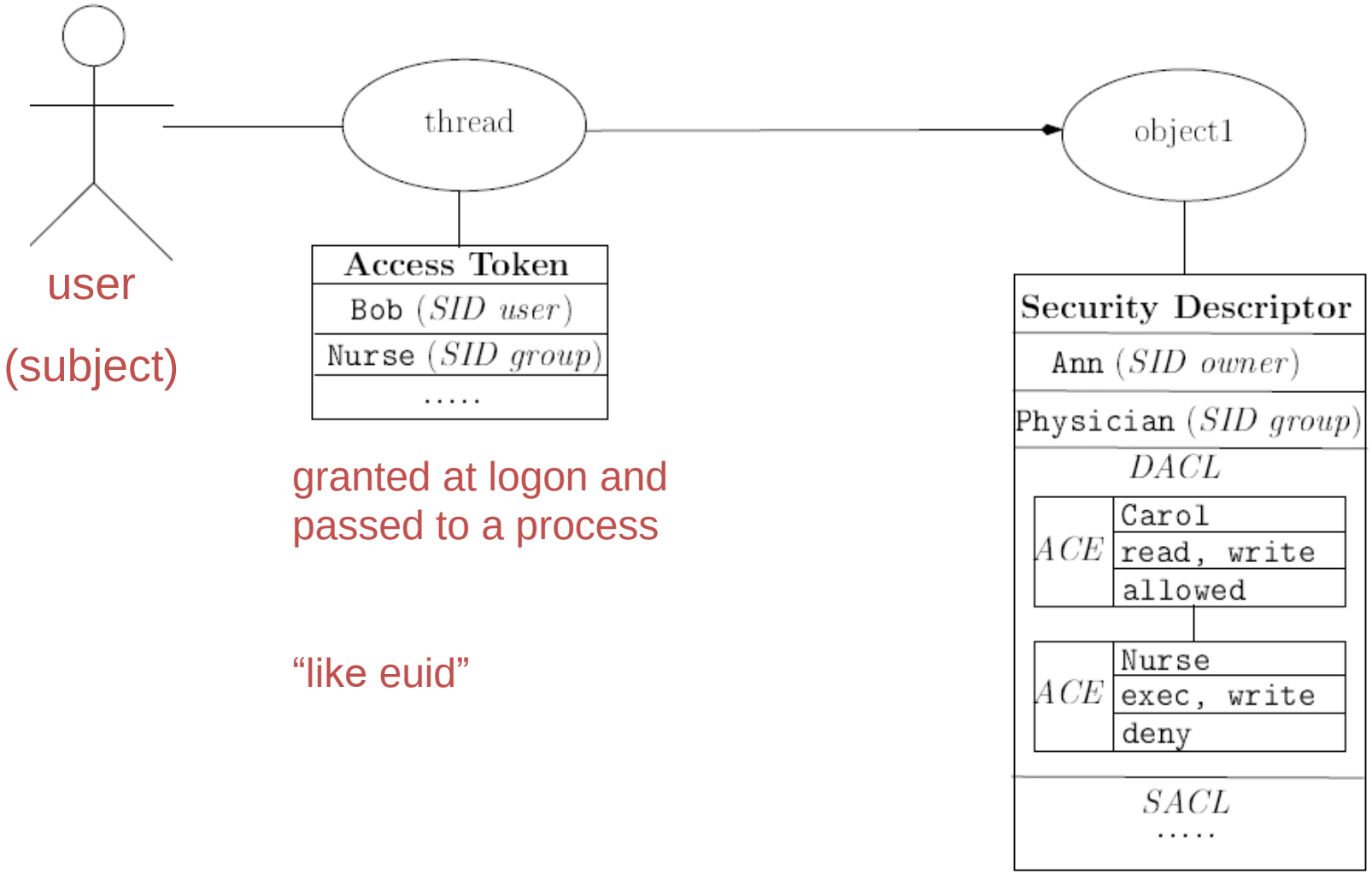
- 共享资源
  - 由于任何进程可以在/tmp目录下创建文件，一个不可信的进程可能创建文件，而该文件可以被任意系统进行使用。

- Time-of-Check-to-Time-of-Use (TOCTTOU)
  - 通常，root进程使用系统调用来决定用户是否具有对某个文件的权限，如/tmp/X。
  - 在访问授权之后，文件打开之前，用户可能改变文件/tmp/X为符号链接，其目标文件可能是/etc/shadow。

Windows中的访问控制

- 一些基本功能类似于Unix
  - 为用户组和用户设定访问权限
    - Read, modify, change owner, delete
- 一些补充的概念
  - 令牌（Tokens）
  - 安全属性
- 通常
  - 比Unix更灵活
    - 可以定义新权限
    - 可以给出部分，而不是所有管理员权限

Windows中的访问控制

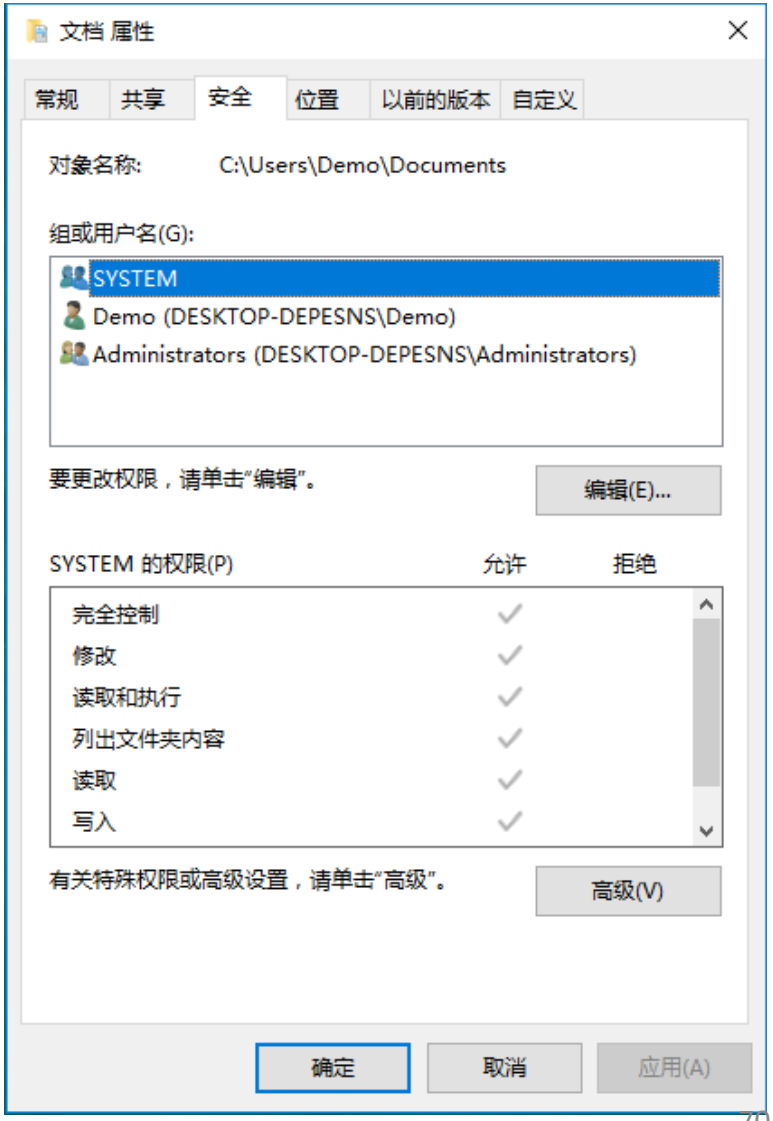


granted at logon and passed to a process

“like euid”

使用SID识别主体

- 安全ID (SID)
  - 身份 (替代 UID)
    - SID 修订号
    - 48位权限值
    - 可变数量的相对标识 (RIDs)
  - 用户、组、计算机、域、域成员都有SID



进程具有一组令牌（tokens）

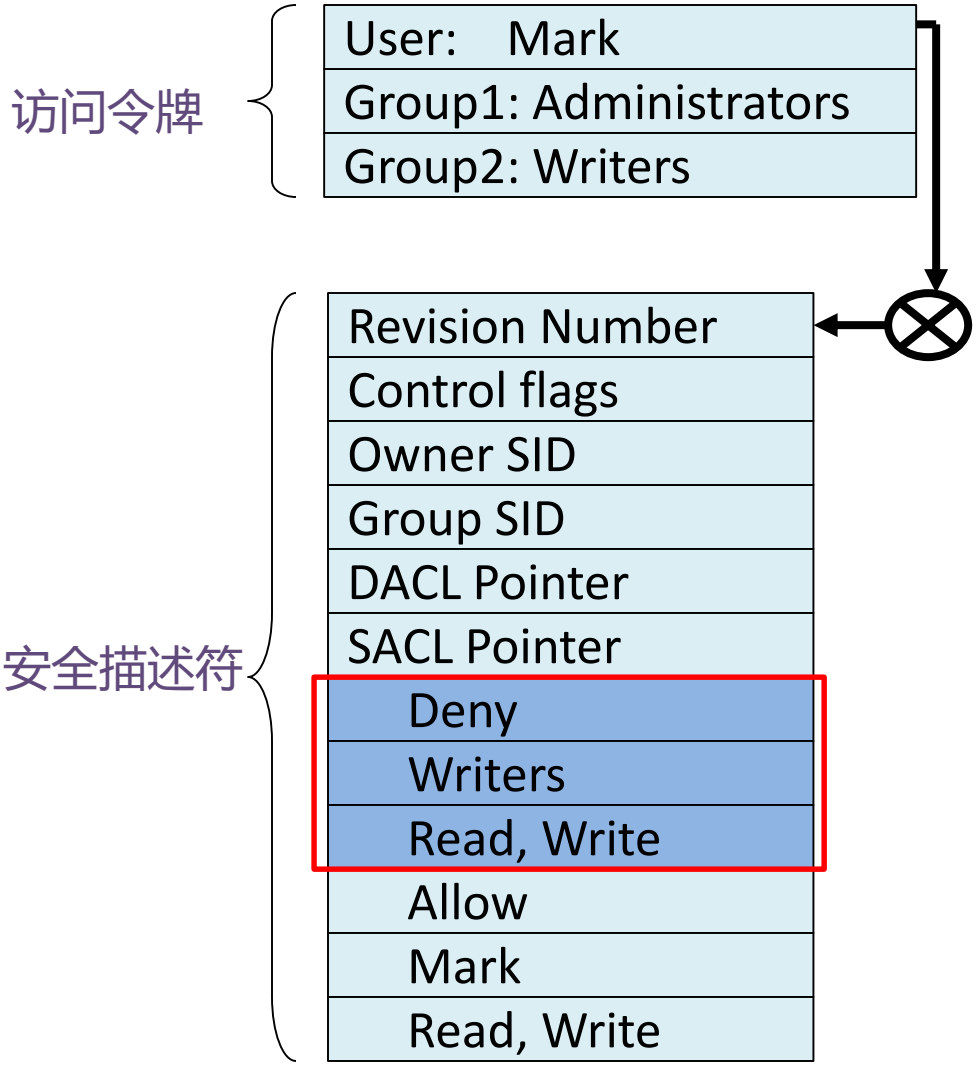
- 安全上下文
  - 特权、账户、与进程或线程相关联的组
  - 表现为一组令牌（tokens）
- 安全引用监控器（Security Reference Monitor）
  - 使用令牌来识别进程或线程的安全上下文
- 模拟令牌(Impersonation token)
  - 临时使用的不同的安全上下文，通常是另一个用户的上下文

客体具有安全描述符

- 与客体关联的安全描述符
  - 指定谁可以对客体执行什么操作

- 几个字段
  - 头部
    - 描述符修订号
    - 控制标志，描述符属性
      - 如，描述符内存布局
  - 客体owner的SID
  - 客体group的SID
  - 两个附加的可选列表：
    - 自主访问控制列表 (DACL) – user、group…
    - 系统访问控制列表 (SACL) – 系统日志…

访问请求举例



- 用户 Mark 请求写权限
- 描述符中拒绝组Writers的rw权限
- reference monitor拒绝请求

(DACL用于访问， SACL用于审计和日志)

优先级：

显式拒绝
显式允许
继承拒绝
继承允许

模拟令牌(相比于setuid)

- 进程采用另一个进程的安全属性
  - 客户端将模拟令牌传递给服务器

- 客户端指定服务器的模拟级别

Impersonation level	Description
SecurityAnonymous	The server cannot impersonate or identify the client.
SecurityIdentification	The server can get the identity and privileges of the client, but cannot impersonate the client.
SecurityImpersonation	The server can impersonate the client's security context on the local system.
SecurityDelegation	The server can impersonate the client's security context on remote systems.

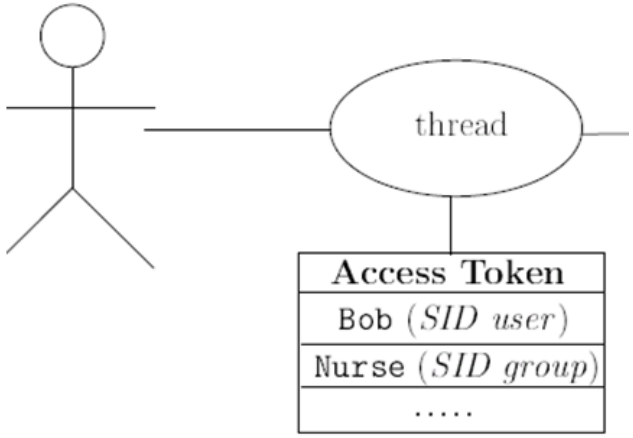
https://www.cnblogs.com/artech/archive/2011/07/03/impersonation01.html

https://docs.microsoft.com/en-us/windows/win32/secauthz/impersonation-levels

模拟令牌

进程/线程可以具有三种令牌:

- the primary access token (e.g. from parent process)
- the impersonation access token that contains the security context of a different user, can contain more privileges.
- a saved access token = like saved euid.



隔离、特权的弱点

- 类似Unix的问题
  - 如，Rootkits 动态加载内核模块
- Windows注册表
  - 全局分层数据库用于存储所有程序的数据
  - 注册表项可以与限制访问的安全上下文关联
  - 注册表的安全至关重要
- 默认开启
  - 历史上，许多Windows部署启用完全权限

大纲

- 隔离和最小特权
- 访问控制概念
- 操作系统机制
- 浏览器机制

Web浏览器和操作系统类比

操作系统

- 主体：进程
  - 用户ID (UID, SID)
  - 自主访问控制
- 客体
  - 文件
  - 网络
  - …
- 漏洞
  - 不可信程序
  - 缓冲区溢出
  - …

Web浏览器

- 主体：网页内容 (JavaScript)
  - “源”(Origin)
  - 强制访问控制
- 客体
  - 文档对象模型(DOM)
  - 框架(frame)
  - Cookies / 本地存储
- 漏洞
  - 跨站脚本(Cross-site scripting)
  - 实现错误
  - …

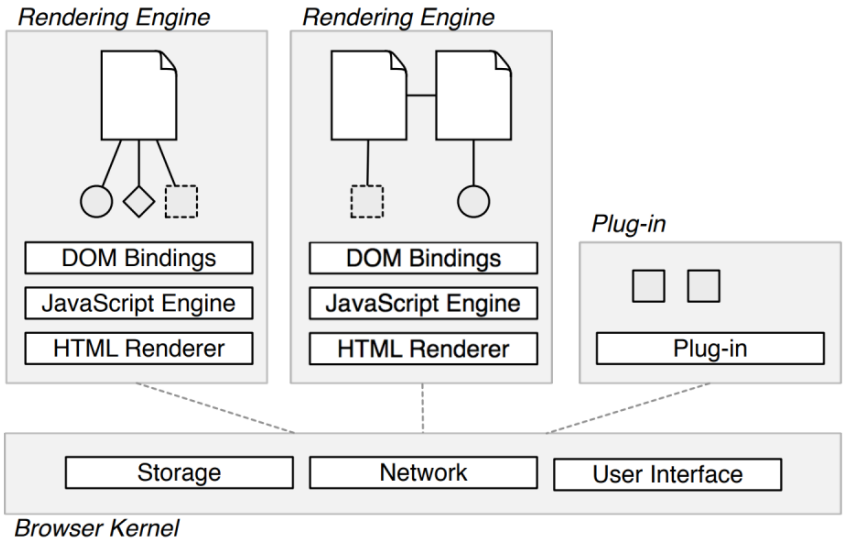
Web浏览器执行自己的内部策略。如果浏览器实现被破坏，这个机制就变得不可靠了。

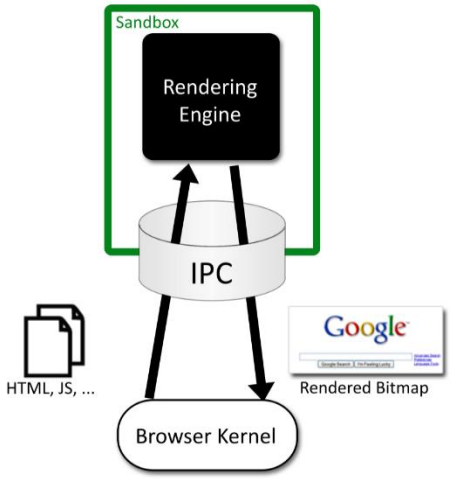
安全策略的组成

- frame-frame关系
  - canScript(A,B)
    - Frame A可以执行一个操纵frame B的任意DOM元素的脚本吗?
  - canNavigate(A,B)
    - Frame A能否改变frame B内容的origin?
- frame-principal关系
  - readCookie(A,S), writeCookie(A,S)
    - Frame A能否读/写站点S的cookies?

Chromium 安全架构

- 浏览器(“内核")
  - 所有权限(文件系统，网络)
- 渲染引擎(Rendering engine)
  - 多达20个进程
  - 沙箱(Sandboxed)
- 每类插件(如：Flash, Silverlight)一个进程
  - 浏览器的所有权限

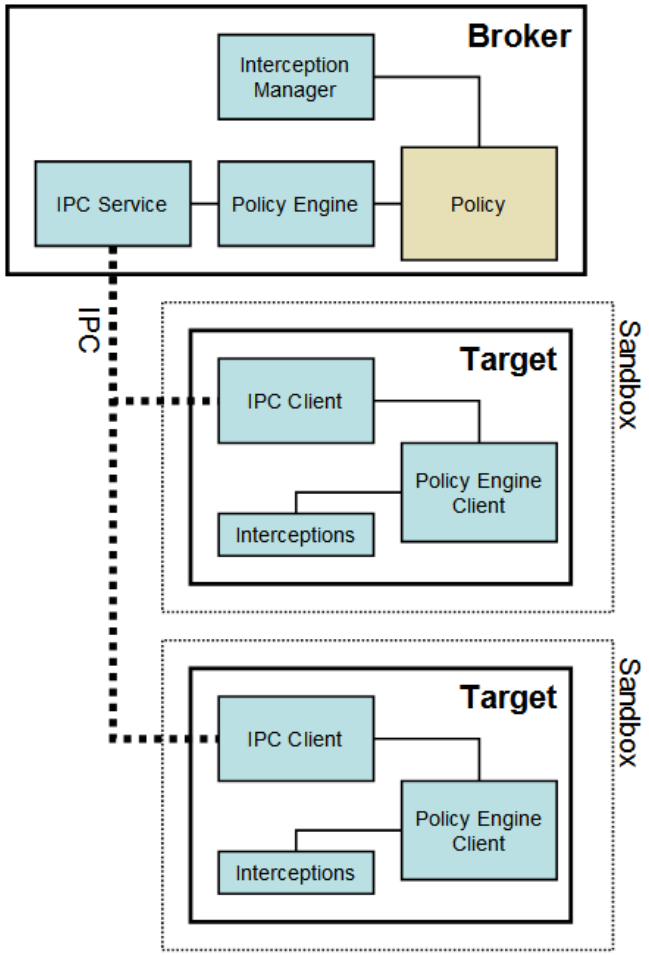




https://www.chromium.org/developers/design-documents/process-models

Chromium

沙箱通信组件



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

浏览器模块的任务分配

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

Both
URL parsing
Unicode parsing

利用OS进行隔离

- 基于四种OS机制的沙箱
  - 限制令牌
  - Windows作业对象(job object)
  - Windows桌面对象(desktop object)
  - 完整性级别：仅Windows Vista（及以后版本）

- 具体说明，渲染引擎
  - 通过将SIDS转换为DENY_ONLY、添加restricted SID，并调用AdjustTokenPrivileges，来调整安全令牌
  - 在单独的Windows作业（job）对象中运行，限制创建新进程、读取或写入剪贴板的能力…
  - 运行在单独的桌面（desktop）上，减轻了对某些Windows API（如SetWindowsHookEx）的安全性检查的缺乏，并减轻了某些不受保护的对象（如HWND_BROADCAST）的使用范围，使得这些对象的范围仅限于当前桌面

https://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf

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

总结

- 安全原则
  - 隔离
  - 最小特权原则
  - Qmail的例子
- 访问控制概念
  - 矩阵，访问控制列表ACL，访问能力表
- 操作系统机制
  - Unix
    - 文件系统，Setuid
  - Windows
    - 文件系统，令牌
- 浏览器安全架构
  - 隔离和最小特权