Introduction to Operating Systems
CS 1550

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

- Upcoming deadlines
  - **All deadlines moved to Monday May 1st at 11:59 pm**
    - But please don't wait to last minute!
  - Homework 11, 12, Bonus Homework
    - lowest two homework assignments dropped
  - Lab 4 and Lab 5
  - Quiz 3 and Quiz 4
    - lowest two of the labs and quizzes dropped
  - Project 4 **(no late deadline)**

Final Exam

- **Wednesday 4/26 8:00-9:50**
  - same classroom
  - coffee will be served!
- Same format as midterm
- **Non-cumulative**
- Study guide and practice test on Canvas
- **Review Session** during Finals' Week
  - Date and time TBD
  - recorded

Bonus Opportunities

- **Bonus Homework (**1%)
- **Post-Course Quiz on Canvas (**1%)
- 1% bonus for class when

  OMETs response rate >= 80%

  - Currently at 46%
  - Deadline is Sunday 4/23

Previous Lecture …

- Miscellaneous issues in File Systems
  - open-file tables
  - quota table
  - journaling file system
  - buffering
  - Max partition size
  - linking vs. copying
  - effective disk access time

This Lecture …

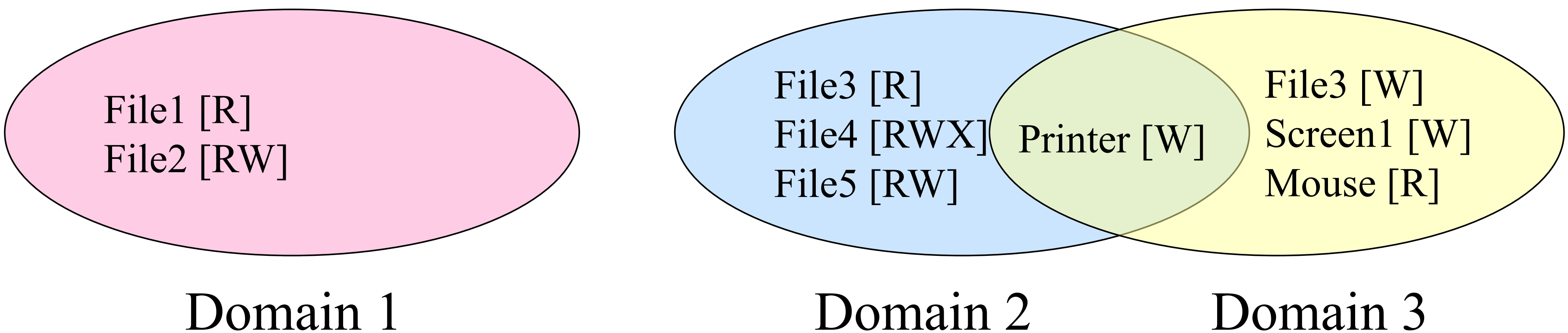
- Protection and Security in operating systems

Problem of the Day: Protection

- Protection is about **controlling access** of
  - programs, processes, or users to
  - system resources
    - (e.g., memory pages, files, devices, CPUs)
- OS can **enforce** policies, but can't decide what policies are appropriate
- How to enforce **who can access what**?
- Access Control Specification:
  - Correct
  - enable an implementation that is
    - efficient
    - easy to use (or nobody will use them!)

Protection domains

- A process operates within a **protection domain**
  - defines what resources accessible by the process
- Each domain lists **objects** with permitted **operations**
- Objects can have different permissions in different domains
- How can this arrangement be specified more formally?



Protection matrix

	File1	File2	File3	File4	File5	Printer1	Camera
Domain1	Read	Read Write					
Domain2			Read	Read Write Execute	Read Write	Write	
Domain3			Write			Write	Read

- Each **domain** has a **row** in the matrix
- Each **object** (resource) has a **column** in the matrix
- Entry i, j has the **permissions** of Domain i on Object j
- Who's allowed to modify the protection matrix?
  - What changes can they make?
- How to efficiently enforce the matrix?

Domains as objects in the protection matrix

	File1	File2	File3	File4	File5	Printer1	Camera	**Dom1**	**Dom2**	**Dom3**
Dom 1	Read	Read Write						Modify		
Dom 2			Read	Read Write Execute	Read Write	Write		Modify		
Dom 3			Write			Write	Read		Enter	

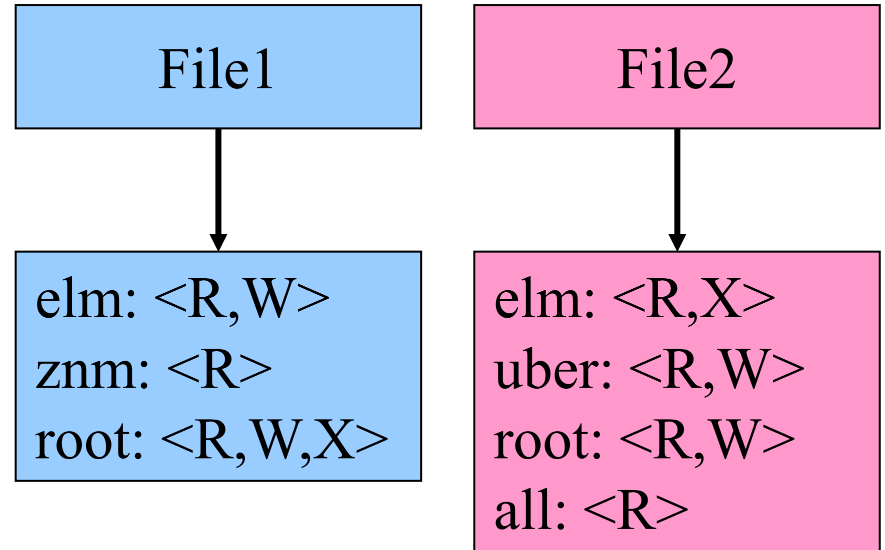
- Specify permitted operations on domains in the matrix
  - Domains can modify other domains
  - Domains may (or may not) be able to **modify themselves**
  - Some domain **transfers** (switching) permitted, others not
- Doing this allows **flexibility** in specifying domain permissions
  - Retains ability to restrict modification of domain policies

Representing the protection matrix

- Need an **efficient** representation of the matrix
  - also called *access control matrix*
- Most entries in the matrix are **empty**!
- Two approaches:
  - Store permissions with each **object**
    - **access control list**
  - Store permissions with each **domain**
    - **capabilities**

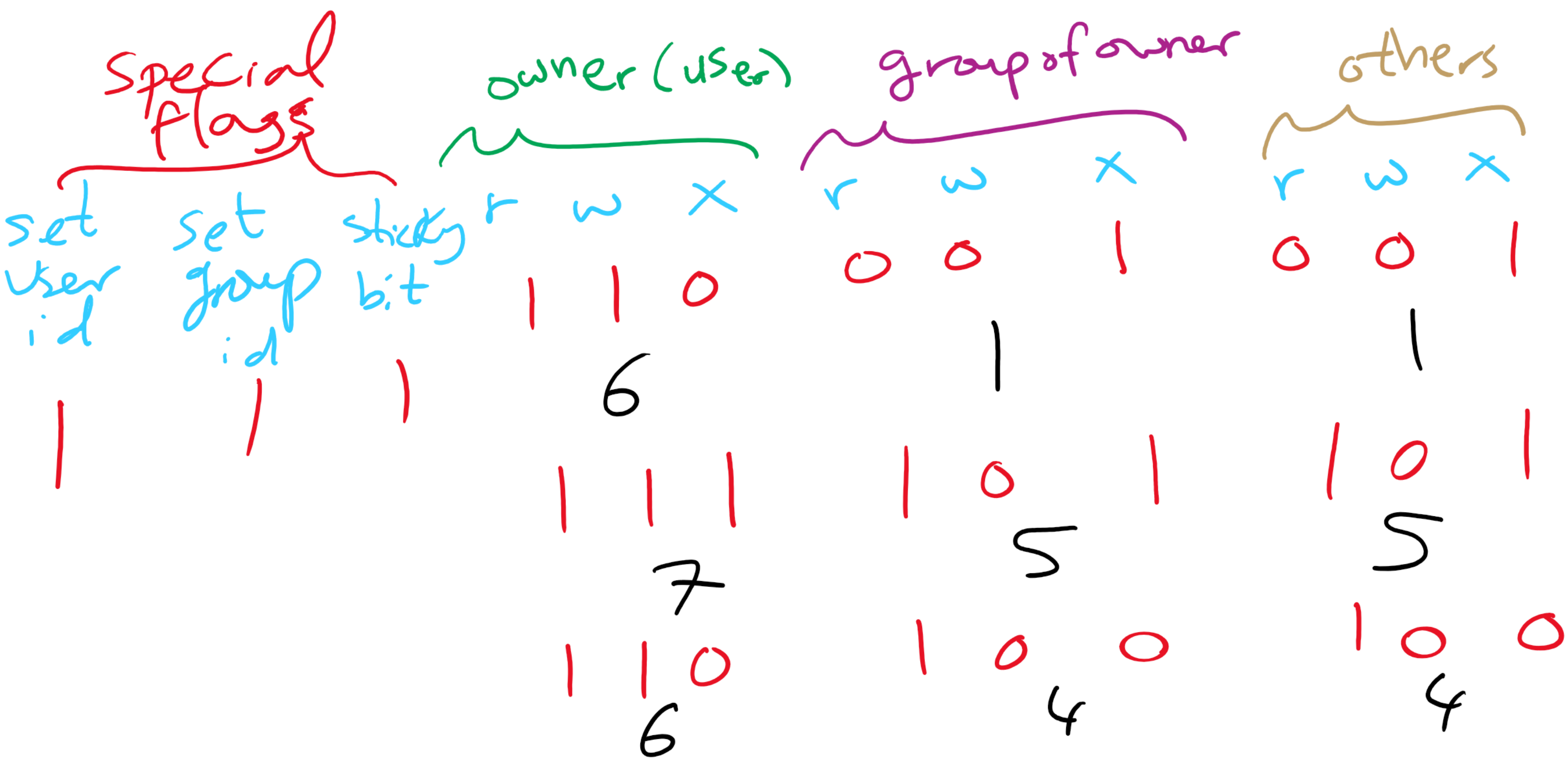
Access control lists (ACLs)

- Each object has a list attached to it
- List has
  - Protection domain
    - e.g., User, Group of users, Other
  - Access rights
    - e.g., Read, Write, Execute
- No entry for domain => no rights for that domain
- Operating system checks permissions when access is needed
- How are ACLs **secured**?
  - Kept in kernel



Access control lists in the real world

- Unix file system
  - Access list for each file has exactly **three domains** on it
    - User (owner), Group, Others
  - Rights include read, write, execute: interpreted differently for directories and files

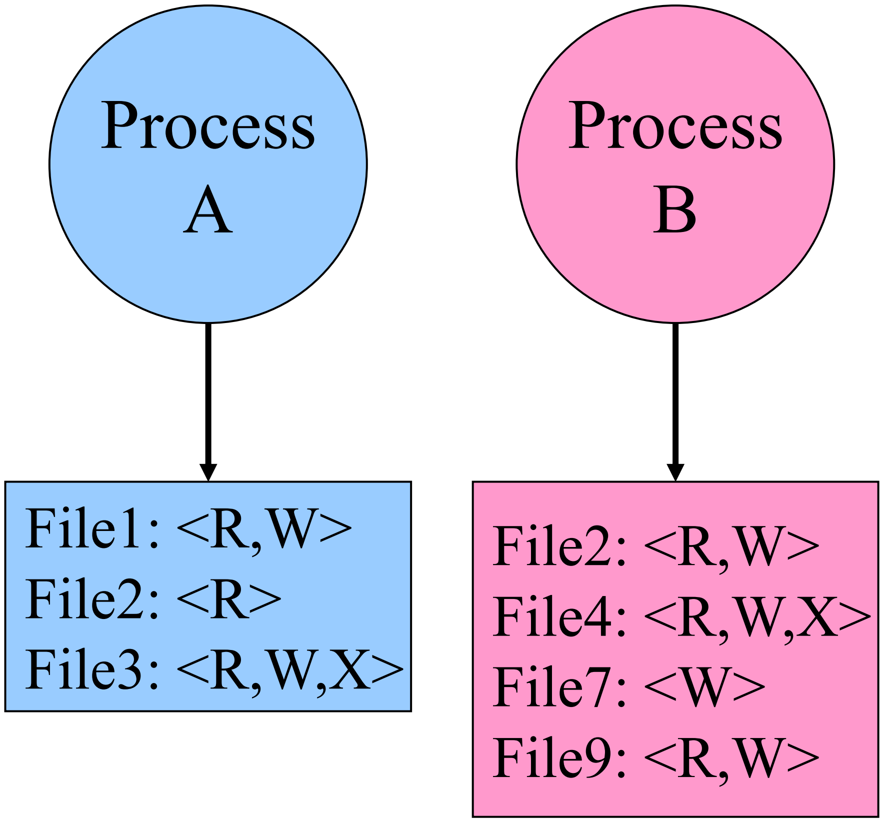


Access control lists in the real world

- Andrew File System (AFS)
  - Access lists apply to directories
    - files inherit rights from **the directory they're in**
    - **what does that statement imply about AFS?**
  - Possible rights:
    - read, write, lock (for files in the directory)
    - lookup, insert, delete (for the directories themselves)
    - administer (ability to add or remove rights from the ACL)

Capabilities

- Each domain has a capability list
- One entry per accessible object
  - Object name
  - Object permissions
- **not listed → no access**
- How are capabilities **secured**?
  - Kept in kernel
  - **Cryptographically secured**

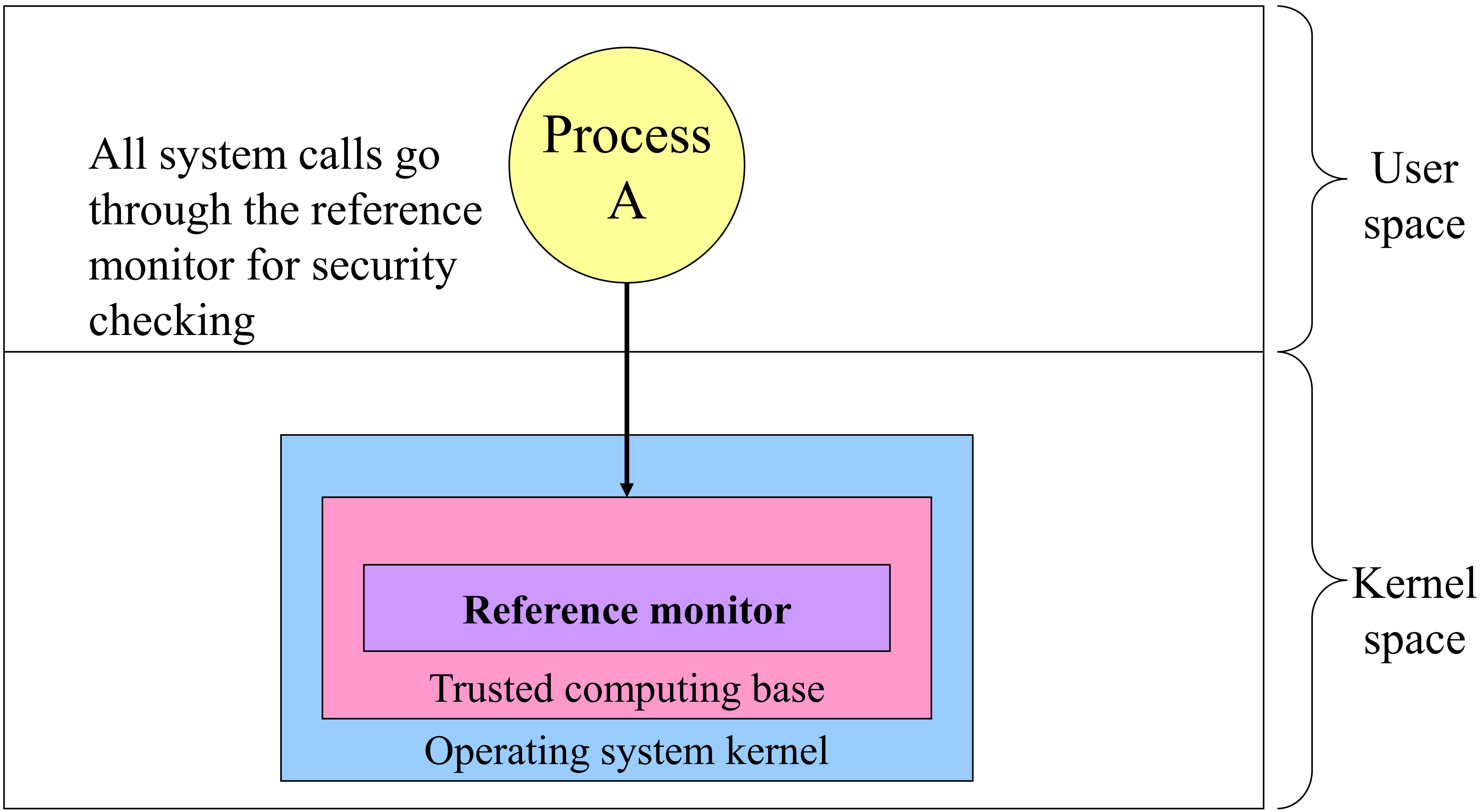


Cryptographically protected capability

Server	Object	Rights	*H(Object,Rights,**Check**)*

- Capability handed to processes and **verified** cryptographically
    - better for widely **distributed** systems where capabilities can't be centrally checked
- H() is a cryptographically secure one-way hash function
    - e.g., SHA-3, SHA-256
- Rights include generic rights (read, write, execute) and
    - Copy capability, Copy object, Remove capability, Destroy object
- Server has a secret (Check) and uses it to verify presented capabilities
    - how?
- Alternatively, use public-key signature techniques

Reference monitor

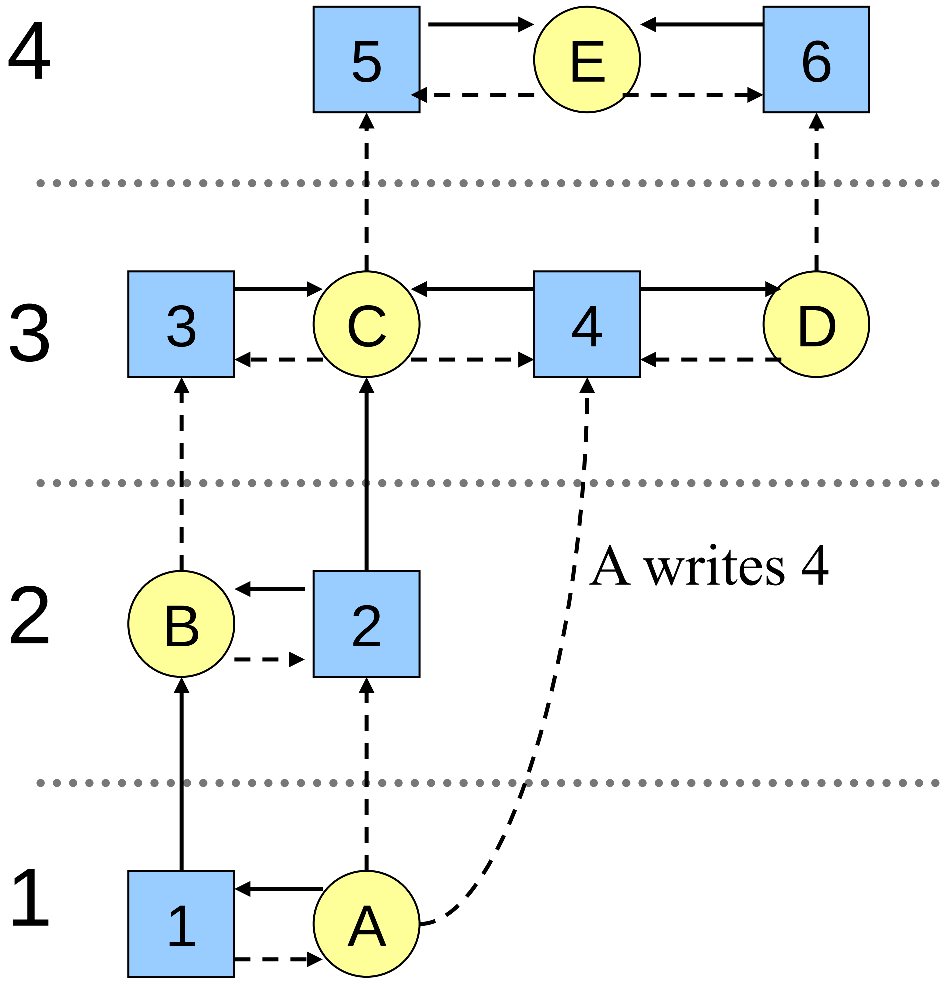


Formal models of protection

- OS can **enforce** policies, but can't decide what policies are authorized
- Is it possible to go from an "authorized" matrix to an "unauthorized" one?
- Limited set of **primitive operations** on access matrix
  - Create/delete object
  - Create/delete domain
  - Insert/remove right
- Primitives can be combined into **protection commands**
- In general, this question is **undecidable**
  - May be provable for limited cases

Bell-La Padula multilevel security model

- Processes and objects have security levels (e.g., 1-4)
- Goal: Prevent information from leaking from higher levels to lower levels
- Simple security property
  - Process at level *k* can **only read** objects at levels *k* or lower
- * property
  - Process at level *k* can **only write** objects at levels *k* or **higher**
- **Read down, write up**



Biba multilevel integrity model

- Goal: guarantee **integrity** of data
  - e.g., prevent planting fake information at a higher level
- Simple integrity property
  - A process can **write only** objects at its security level or lower
- The integrity * property
  - A process can **read only** objects at its security level or higher
- **Read up, write down**

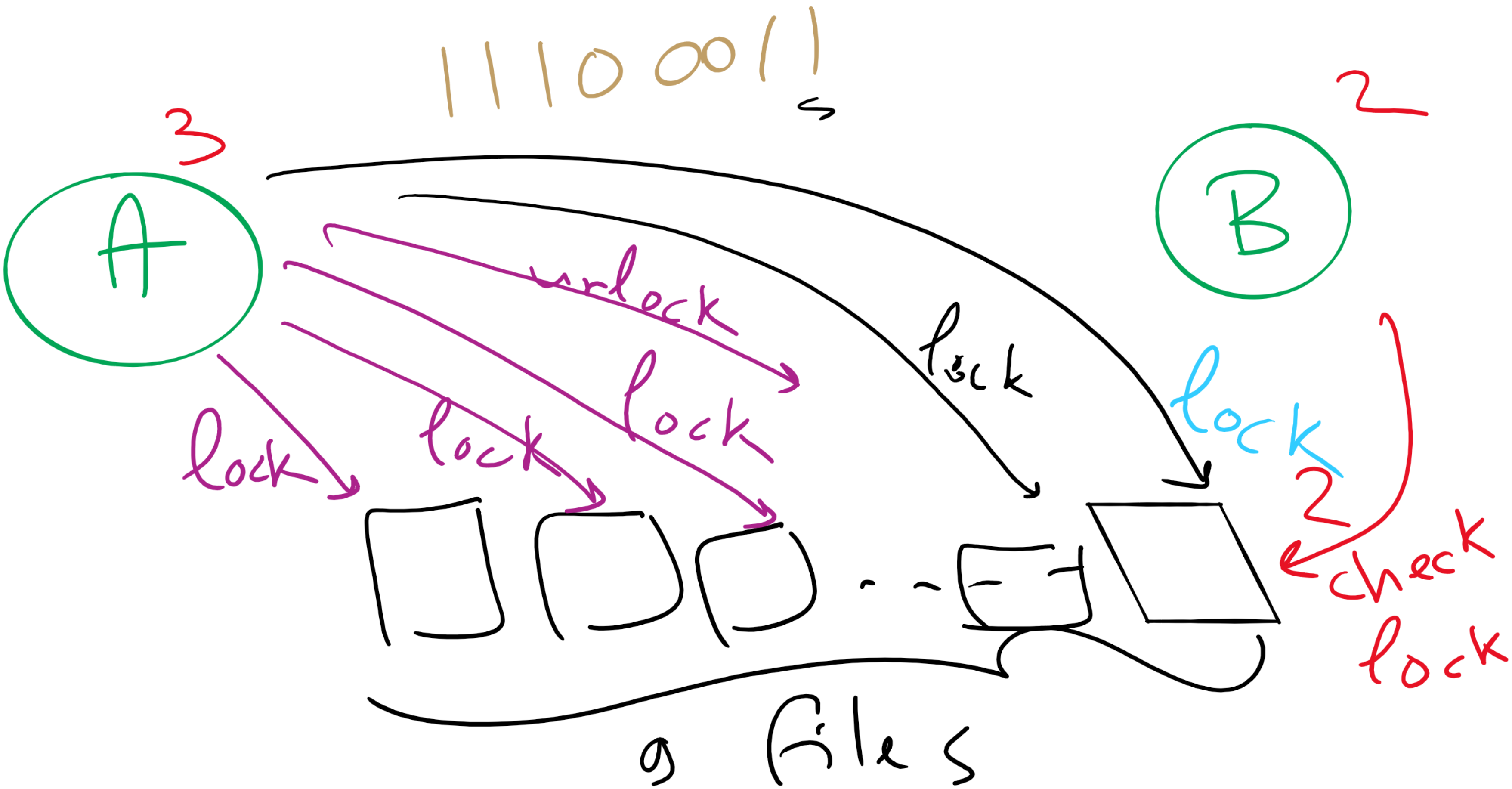
Covert channels

- Circumvent security model by using more **subtle ways of passing information**
- Send data using "**side effects**"
  - Allocating resources
  - Using the CPU
  - Locking a file
  - Making small changes in legal data exchange
- *Very* difficult to **plug leaks** by covert channels!

Covert channel using file locking

- Process A and Process B want to exchange information using file locking
- Assume *n*+1 files accessible to both A and B
- A sends information by
  - Locking files 0..*n*-1 according to an *n*-bit quantity to be conveyed to B
  - Locking file *n* to indicate that information is available
- B gets information by
  - Reading the lock state of files 0..*n*+1
  - Unlocking file *n* to show that the information was received
- May not even need access to the files (on some systems) to detect lock status!

Covert Channel Using File Locking



Steganography

- Hide information in other data
- Picture on right has text of 5 Shakespeare plays
  - Encrypted, inserted into low order bits of color values

Zebras

Hamlet, Macbeth, Julius Caesar
Merchant of Venice, King Lear

Protection vs Security

Protection is an internal problem

- Assumes users are authenticated and programs are run only by authorized users

Security = Protection + defending attacks from external environment

Security environment: threats

Goal	Threat
Data confidentiality	Exposure of data
Data integrity	Tampering with data
System availability	Denial of service

- Security goals:
  - Confidentiality
  - Integrity
  - Availability
- Someone attempts to subvert the goals
  - Fun
  - Commercial gain

Security Problem 1: Password Attacks

- Passwords can be
  - stolen,
  - guessed, or
  - cracked
- How would you defend against these attacks?

User authentication

- Problem: how does the computer know who you are?
- Solution: use *authentication* to identify
  - Something the user knows
  - Something the user has
  - Something the user is
- This must be done before user can use the system
- Important: from the computer's point of view…
  - Anyone who can duplicate your ID *is* you
  - Fooling a computer isn't all that hard…

Password Stealing

- Stealing the password file
- Social Engineering
  - e.g., spoofing login screen
- Key loggers
  - e.g., trojan horse programs

How should an OS store passwords?

- Passwords should be memorable?
- Passwords shouldn’t be stored “in the clear”
  - Password file is often readable by all system users!
  - Password must be checked against entry in this file
- Solution: use hashing to hide “real” password
  - One-way function converting password to meaningless string of digits (Unix password hash, SHA-2)
  - Difficult to find another password that hashes to the same string
  - Knowing the hashed value and hash function gives no clue to the original password

Storing passwords

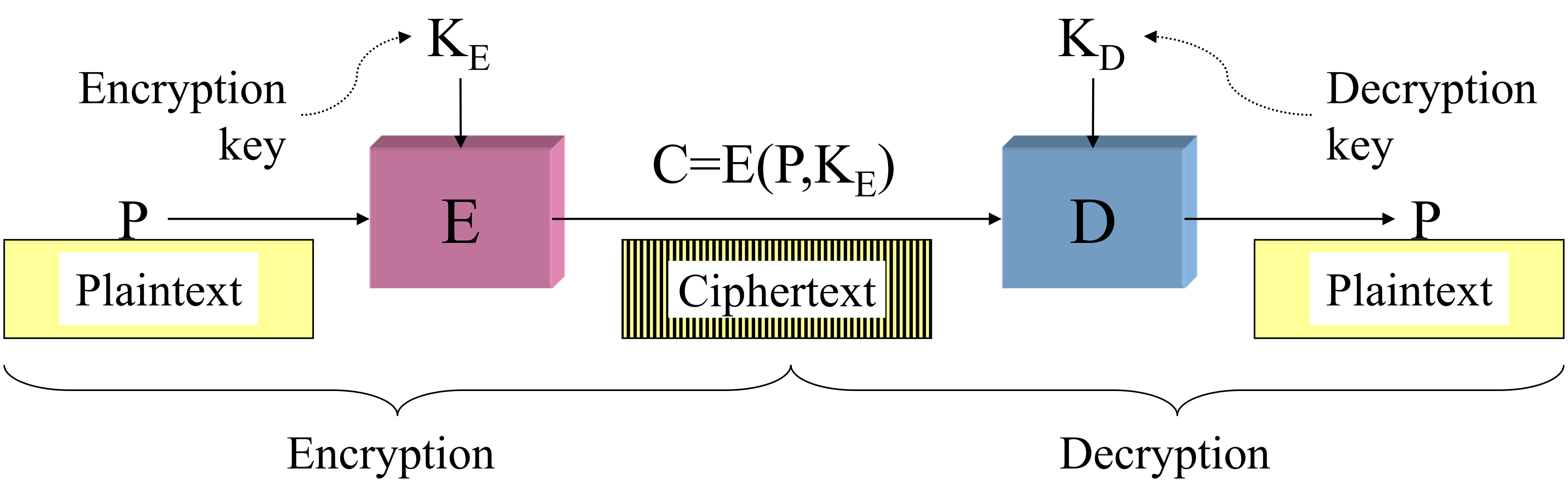
- Some OSs use *encryption algorithms* to hash the passwords
  - Use the password as the key, not the plain text
  - But, what is encryption?

Cryptography

- Goal: keep information from those who aren't supposed to see it
  - Do this by "scrambling" the data
- Use a well-known algorithm to scramble data
  - Algorithm has two inputs: data & key
  - Key is known only to "authorized" users
  - Relying upon the secrecy of the algorithm is a *very* bad idea (see WW2 Enigma for an example…)
- Cracking codes is ***very*** difficult, *Sneakers* and other movies notwithstanding

Cryptography basics

- Algorithms (E, D) are widely known
- Keys (K_E, K_D) may be less widely distributed
- For this to be effective, the ciphertext should be the only information that's available to the world
- Plaintext is known only to the people with the keys (in an ideal world…)



Secret-key encryption

- Also called symmetric-key encryption
- Monoalphabetic substitution
  - Each letter replaced by different letter
- Vigenere cipher
  - Use a multi-character key
    THEMESSAGE
    ELMELMELME
    XSQQPEWLSI
- Both are easy to break!
- Given the encryption key, easy to generate the decryption key
- Alternatively, use different (but similar) algorithms for encryption and decryption

Modern encryption algorithms

- Data Encryption Standard (DES)
  - Uses 56-bit keys
  - Same key is used to encrypt & decrypt
  - Keys used to be difficult to guess
    - Needed to try 2^{55} different keys, on average
    - Modern computers can try millions of keys per second with special hardware
    - For $250K, EFF built a machine that broke DES quickly in 1998
- Current algorithms (AES, Blowfish) use 128 bit keys
  - Adding one bit to the key makes it twice as hard to guess
  - Must try 2^{127} keys, on average, to find the right one
  - At 10^{15} keys per second, this would require over 10^{21} seconds, or 1000 billion years!
  - Modern encryption isn't usually broken by brute force…

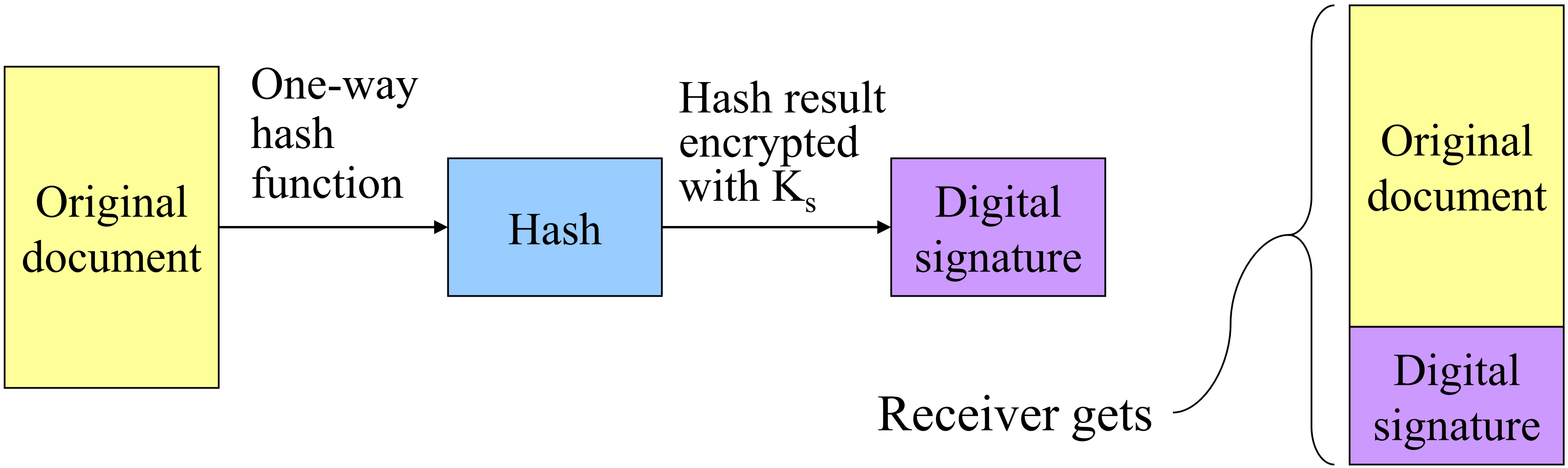
Unbreakable codes

- There ***is*** such a thing as an unbreakable code: one-time pad
  - Use a truly random key as long as the message to be encoded
  - XOR the message with the key a bit at a time
- Code is unbreakable because
  - Key could be anything
  - Without knowing key, message could be anything with the correct number of bits in it
- Difficulty: distributing key is as hard as distributing message
- Difficulty: generating truly random bits
  - Can’t use computer random number generator!
  - May use physical processes
    - Radioactive decay
    - Leaky diode
    - Lava lamp (!) [https://www.atlasobscura.com/places/encryption-lava-lamps]

Public-key cryptography

- Instead of using a single shared secret, keys come in pairs
  - One key of each pair distributed widely (*public key*), K_p
  - One key of each pair kept secret (*private or secret key*), K_s
  - Two keys are inverses of one another, but not identical
  - Encryption & decryption are the same algorithm, so $E(K_p,E(K_s,M)) = E(K_s,E(K_p,M)) = M$
- Currently, most popular method involves primes and exponentiation
  - Difficult to crack unless large numbers can be factored
  - Very slow for large messages

Digital signatures



- Digital signature computed by
  - Applying one-way hash function to original document
  - Encrypting result with sender's *private* key
- Receiver can verify by
  - Applying one-way hash function to received document
  - Decrypting signature using sender's public key
  - Comparing the two results: equality means document unmodified

Pretty Good Privacy (PGP)

- Uses public key encryption
  - Facilitates key distribution
  - Allows messages to be sent encrypted to a person (encrypt with person's public key)
  - Allows person to send message that must have come from her (encrypt with person's private key)
- Problem: public key encryption is very slow
- Solution: use public key encryption to exchange a shared key
  - Shared key is relatively short (~128 bits)
  - Message encrypted using symmetric key encryption
- PGP can also be used to authenticate sender
  - Use digital signature and send message as plaintext

Social Engineering

- Convince a system programmer to add a trap door
- Beg admin's secretary (or other people) to help a poor user who forgot password
- Pretend you're tech support and ask random users for their help in debugging a problem

Login spoofing

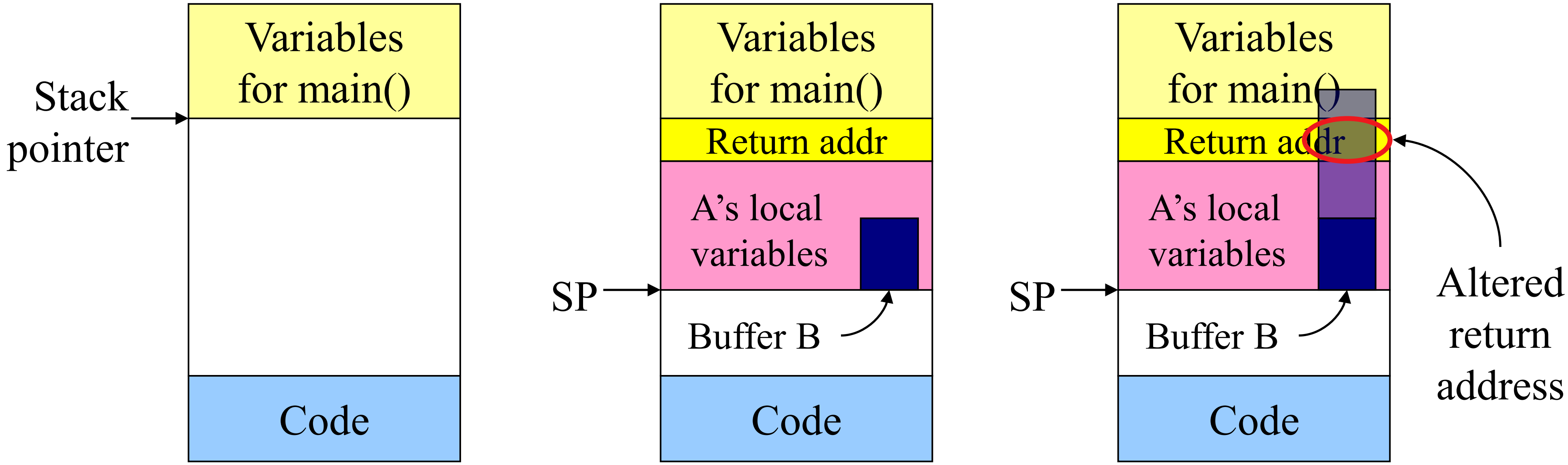
Real login screen Phony login screen

- No difference between real & phony login screens
- Intruder sets up phony login, walks away
- User logs into phony screen
  - Phony screen records user name, password
  - Phony screen prints “login incorrect” and starts real screen
  - User retypes password, thinking there was an error
- Solution: don’t allow certain characters (ctrl+alt+delete) to be “caught”

Trojan horses

- Free program made available to unsuspecting user
  - Actually contains code to do harm
  - May do something useful as well…
- Altered version of utility program on victim's computer
  - Trick user into running that program
- Example (getting superuser access?)
  - Place a file called **ls** in your home directory
    - File creates a shell in /tmp with privileges of whoever ran it
      - cp /bin/bash /tmp/.SecretShell && chmod 4755 /tmp/.SecretShell
    - File then actually runs the real ls
  - Complain to your sysadmin that you can't see any files in your directory
  - Sysadmin runs ls in your directory
    - Hopefully, he runs *your* ls rather than the real one (depends on his search path)

Buffer overflow



- Buffer overflow is a big source of bugs in operating systems
  - Most common in user-level programs that help the OS do something
  - May appear in "trusted" daemons
- Exploited by modifying the stack to
  - Return to a different address than that intended
  - Include code that does something malicious
- Accomplished by writing past the end of a buffer on the stack

Password Guessing: Sample breakin (from LBL)

LBL> **telnet elxsi**
ELXSI AT LBL
LOGIN: **root**
PASSWORD: **root**
INCORRECT PASSWORD, TRY AGAIN
LOGIN: **guest**
PASSWORD: **guest**
INCORRECT PASSWORD, TRY AGAIN
LOGIN: **uucp**
PASSWORD: **uucp**
WELCOME TO THE ELXSI COMPUTER AT LBL

Moral: change all the default system passwords!

Password Cracking

- Offline cracking
  - The attacker has the password files
    - password files contains password hashes
- Online cracking
  - The attacker doesn't have the password file

Offline Password Cracking

- Passwords can be cracked
  - Hackers can get a copy of the password file
  - Run through dictionary words and names
    - Hash each name
    - Look for a match in the file
- Hashes can be pre-computed offline!
- Solution: use “salt”
  - Random characters added to the password before hashing
  - Salt characters stored “in the clear”
  - Increase the number of possible hash values for a given password
    - Actual password is “pass”
    - Salt = “aa” => hash “passaa”
    - Salt = “bb” => hash “passbb”
  - Result: cracker has to store many more combinations

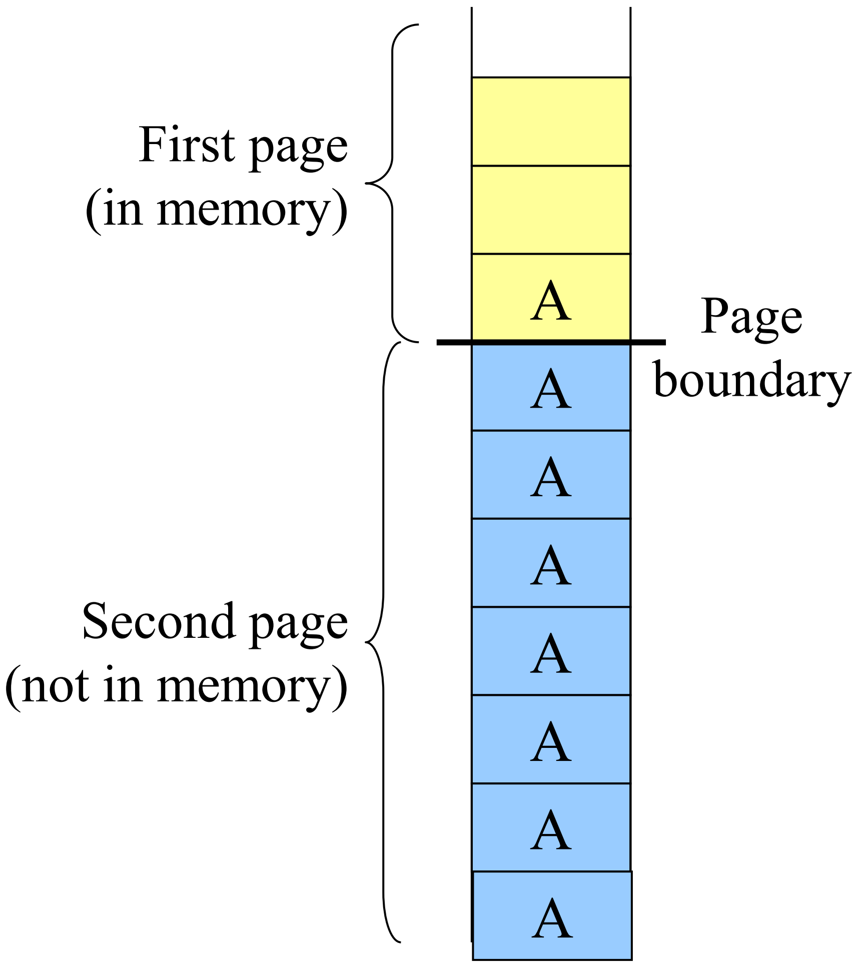
Online Password Cracking

Login: **elm** Password: **foobar** Welcome to Linux!	Login: **jimp** User not found! Login:	Login: **elm** Password: **barfle** Invalid password! Login:

- Successful login lets the user in
- If things don't go so well…
  - Login rejected after name entered
  - Login rejected after name and incorrect password entered
- Don't notify the user of incorrect user name until *after* the password is entered!
  - Early notification can make it easier to guess valid user names

Security flaws: TENEX OS password problem

- Cracking passwords using side-channel attack



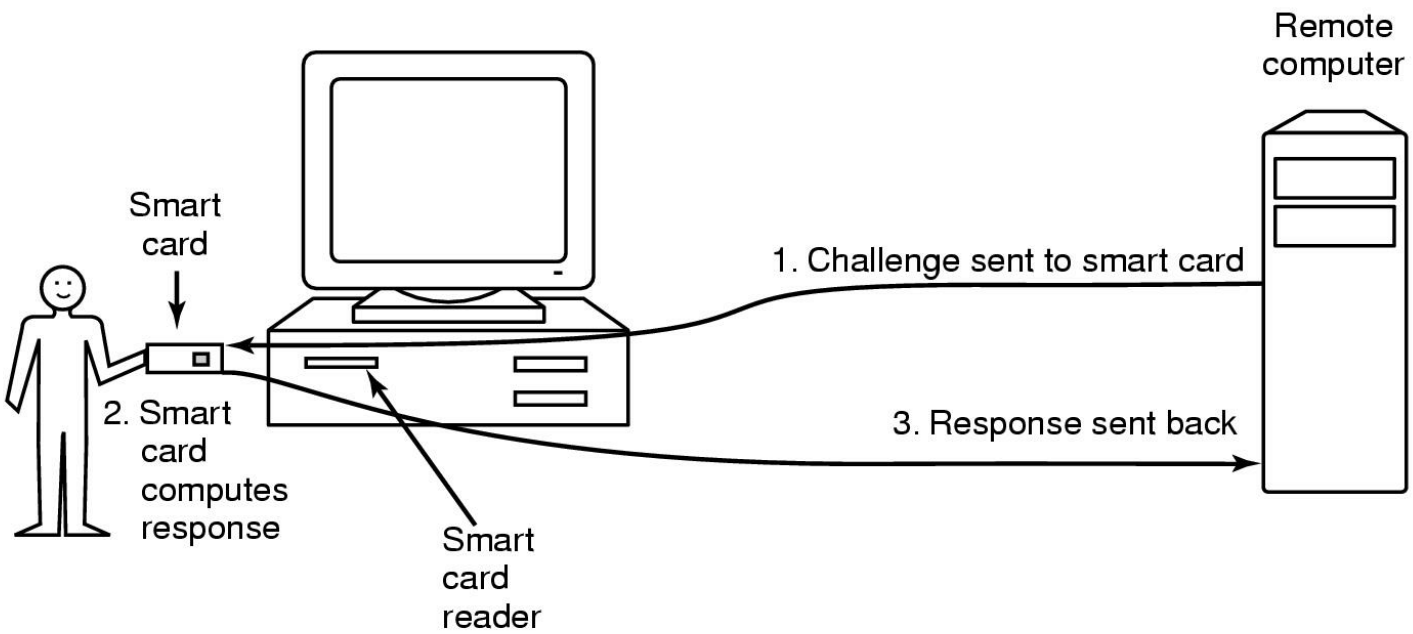
Bypassing Passwords

- Request “free” memory, disk space, tapes and just read what was left there (not zero-filled on dealloc)
- Try illegal system calls – if the system gets confused enough, you may be in
- Start a login and hit DEL, RUBOUT, or BREAK to possibly kill password checking
- Try to do specified DO NOTs

Countermeasures

- Limiting times when someone can log in
- Automatic callback at number prespecified
  - Can be hard to use unless there's a modem involved
- Limited number of login tries
  - Prevents attackers from trying lots of combinations quickly
- A database of all logins
- Simple login name/password as a trap
  - Security personnel notified when attacker bites
  - Variation: allow anyone to "log in," but don't let intruders do anything useful

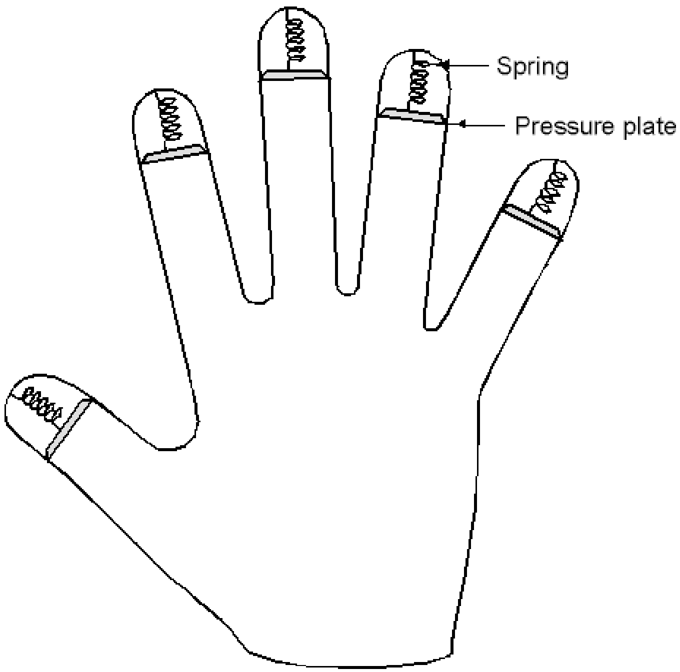
Authentication using a physical object



- Magnetic card
  - Stores a password encoded in the magnetic strip
  - Allows for longer, harder to memorize passwords
- Smart card
  - Card has secret encoded on it, but not externally readable
  - Remote computer issues challenge to the smart card
  - Smart card computes the response and proves it knows the secret

Authentication using biometrics

- Use basic body properties to prove identity
- Examples include
  - Fingerprints
  - Voice
  - Hand size
  - Retina patterns
  - Iris patterns
  - Facial features
- Potential problems
  - Duplicating the measurement
  - Stealing it from its original owner?



Security Problem 2: Viruses and Worms

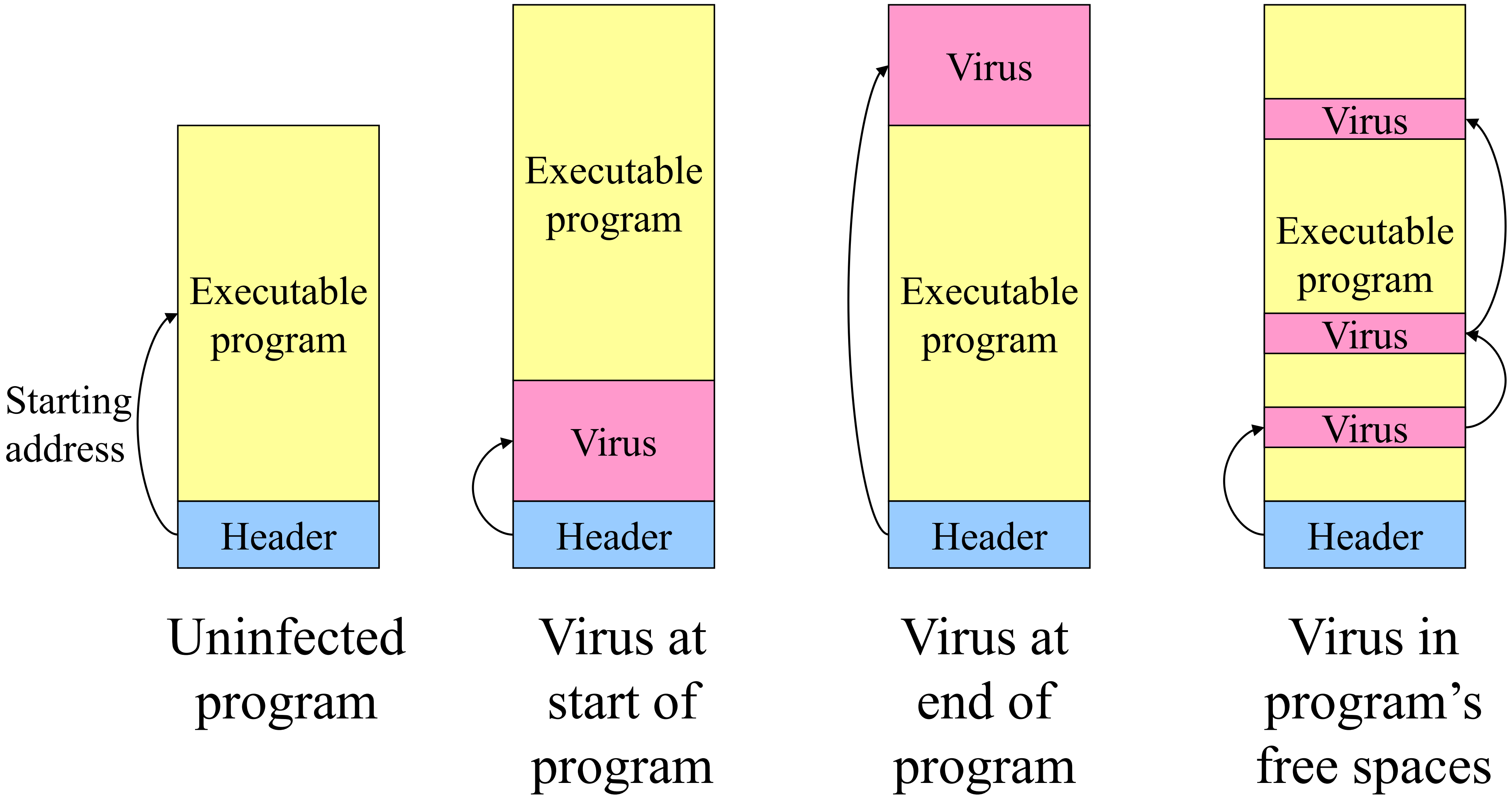
- Virus: program that embeds itself into other (legitimate) code to reproduce and do its job
  - Attach its code to another program
  - Additionally, may do harm

- Goals of virus writer
  - Quickly spreading virus
  - Difficult to detect
  - Hard to get rid of
  - Optional: does something malicious
    - e.g., Ransomware

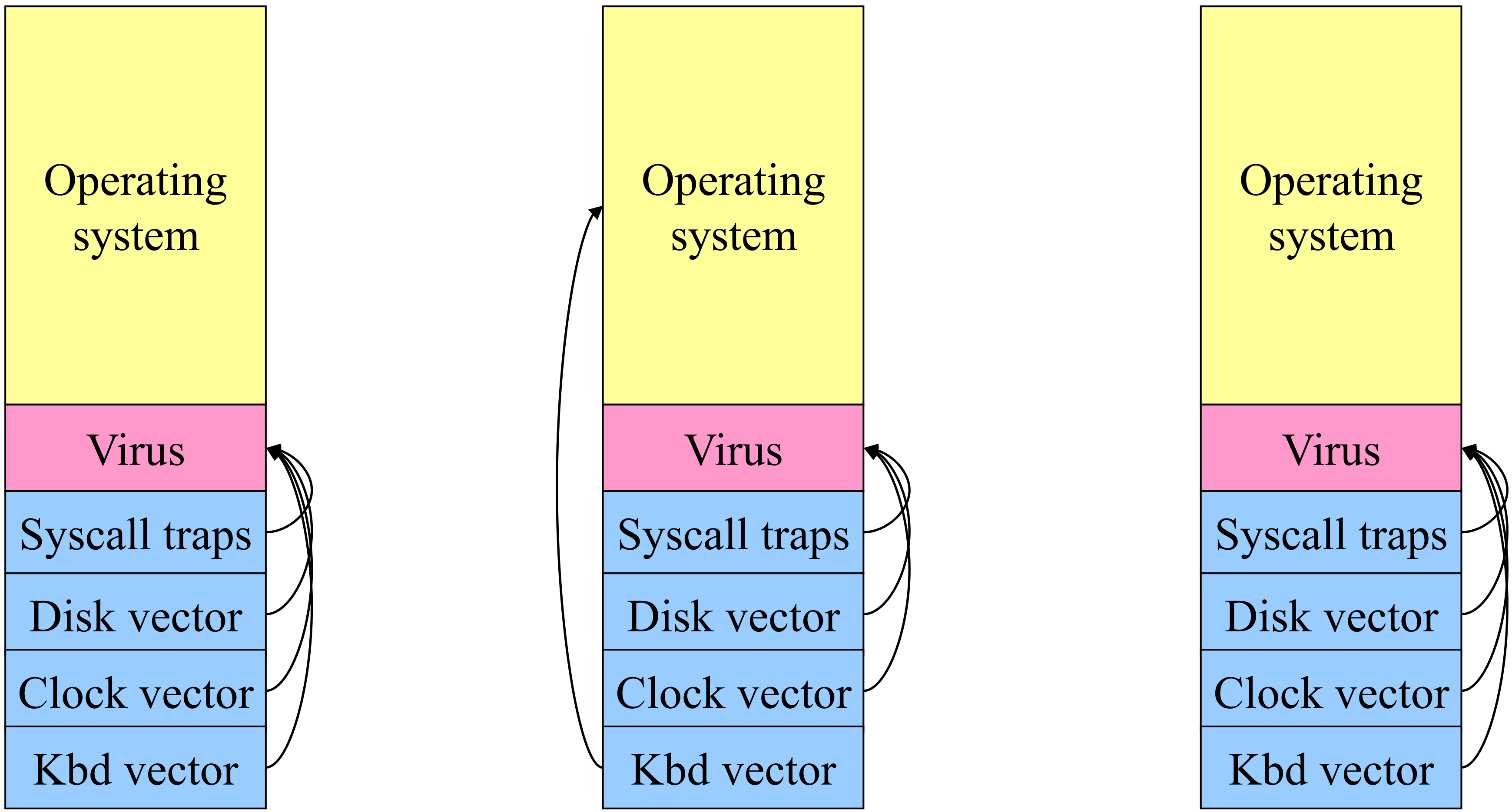
How viruses work

- Virus language
  - Assembly language: infects programs
  - “Macro” language: infects email and other documents
    - Runs when email reader / browser program opens message
    - Program “runs” virus (as message attachment) automatically
- **Inserted into another program**
  - Use tool called a “dropper”
  - May also infect system code (boot block, etc.)
- Virus dormant until program executed
  - Then infects other programs
  - Eventually executes its “payload”

Where viruses live in the program



Viruses infecting the operating system



Virus has captured interrupt & trap vectors

OS retakes keyboard vector

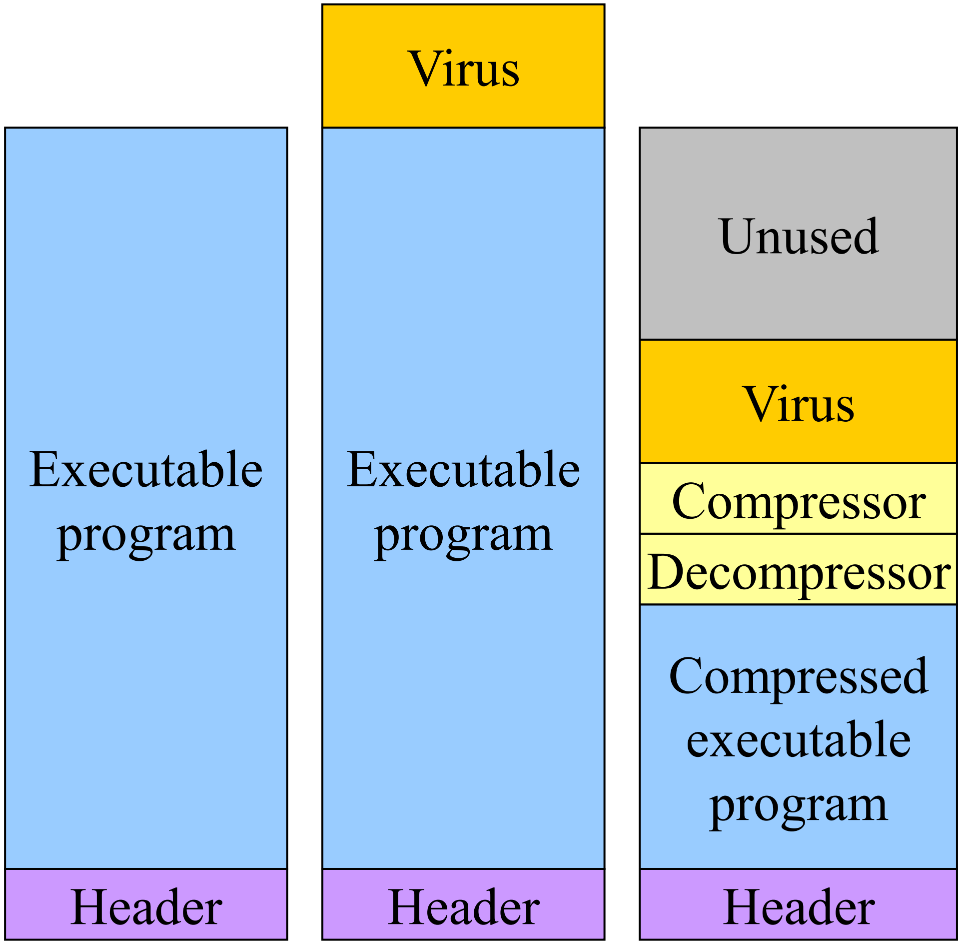
Virus notices, recaptures keyboard

How do viruses spread?

- Virus placed where likely to be copied
  - Popular download site
  - Photo site
- When copied and run
  - Infects programs on hard drive, flash drive
  - May try to spread over LAN or WAN
- Attach to innocent looking email
  - When it runs, use mailing list to replicate
  - May mutate slightly so recipients don’t get suspicious

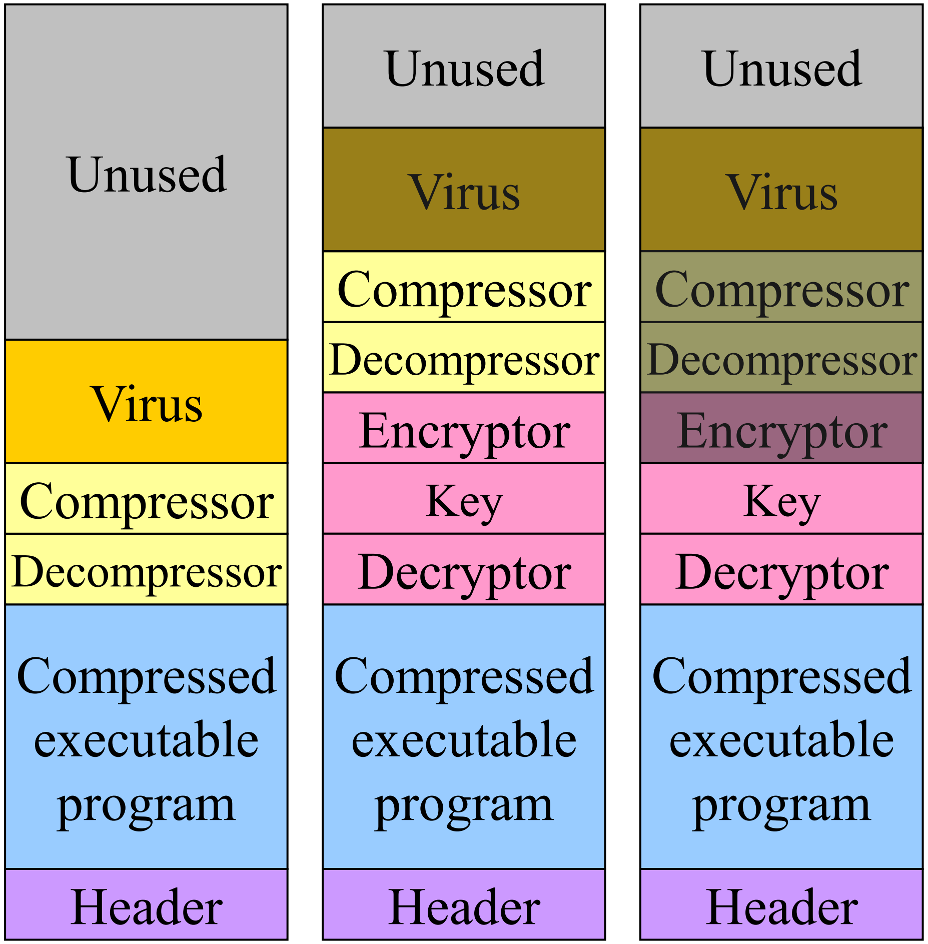
Hiding a virus in a file

- Start with an uninfected program
- Add the virus to the end of the program
  - Problem: file size changes
  - Solution: compression
- Compress infected program
  - Decompressor: for running executable
  - Compressor: for compressing newly infected binaries
  - Lots of free space (if needed)
- Problem (for virus writer): virus easy to recognize



Using encryption to hide a virus

- Hide virus by encrypting it
  - Vary the key in each file
  - Virus "code" varies in each infected file
  - Problem: lots of common code still in the clear
    - Compress / decompress
    - Encrypt / decrypt
- Even better: leave only decryptor and key in the clear
  - Less constant per virus
  - Use polymorphic code (more in a bit) to hide even this



Polymorphic viruses

- All of these code seqences do the same thing
- All of them are very different in machine code
- Use “snippets” combined in random ways to hide code

```
MOV A,R1
ADD B,R1
ADD C,R1
SUB #4,R1
MOV R1,X
```

(a)

```
MOV A,R1
NOP
ADD B,R1
NOP
ADD C,R1
NOP
SUB #4,R1
NOP
MOV R1,X
```

(b)

```
MOV A,R1
ADD #0,R1
ADD B,R1
OR R1,R1
ADD C,R1
SHL #0,R1
SUB #4,R1
JMP .+1
MOV R1,X
```

(c)

```
MOV A,R1
OR R1,R1
ADD B,R1
MOV R1,R5
ADD C,R1
SHL R1,0
SUB #4,R1
ADD R5,R5
MOV R1,X
MOV R5,Y
```

(d)

```
MOV A,R1
TST R1
ADD C,R1
MOV R1,R5
ADD B,R1
CMP R2,R5
SUB #4,R1
JMP .+1
MOV R1,X
MOV R5,Y
```

(e)

How can viruses be foiled?

- Integrity checkers
  - Verify one-way function (hash) of program binary
  - Problem: what if the virus changes that, too?
- Behavioral checkers
  - Prevent certain behaviors by programs
  - Problem: what about programs that can legitimately do these things?
- Avoid viruses by
  - Having a good (secure) OS
  - Installing only shrink-wrapped software (just hope that the shrink-wrapped software isn't infected!)
  - Using antivirus software
  - Not opening email attachments
- Recovery from virus attack
  - Hope you made a recent backup!
  - Recover by halting computer, rebooting from safe disk (CD-ROM?), using an antivirus program

What if I have to run untrusted code?

- Goal: run (untrusted) code on my machine
- Problem: how can untrusted code be prevented from damaging my resources?
- One solution: sandboxing
  - Memory divided into 1 MB sandboxes
  - Accesses may not cross sandbox boundaries
  - Sensitive system calls not in the sandbox
- Another solution: interpreted code
  - Run the interpreter rather than the untrusted code
  - Interpreter doesn't allow unsafe operations
- Third solution: signed code
  - Use cryptographic techniques to sign code
  - Check to ensure that mobile code signed by reputable organization

Worms vs. viruses

- Viruses require other programs to run
- Worms are self-running (separate process)
- The 1988 Internet Worm
  - Consisted of two programs
    - Bootstrap to upload worm
    - The worm itself
  - Exploited bugs in sendmail and finger
  - Worm first hid its existence
  - Next replicated itself on new machines
  - Brought the Internet (1988 version) to a screeching halt