Defending (Against) Weak Applications

Agenda

Access Control

Isolation

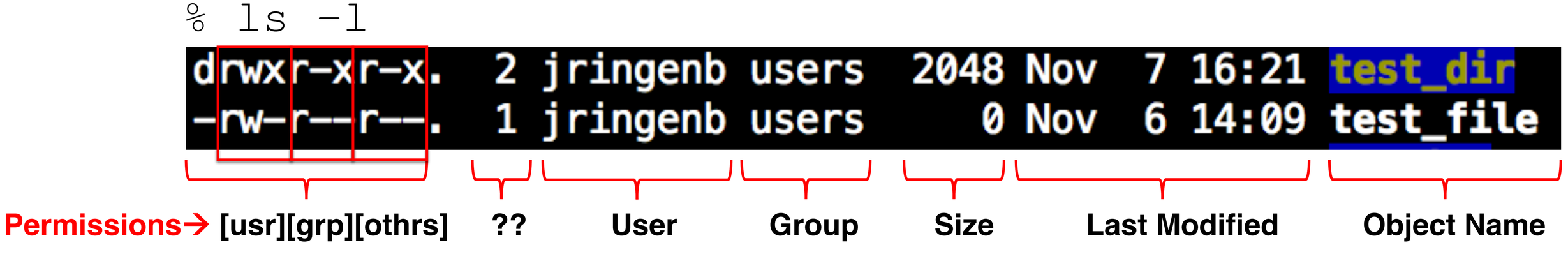
Sandboxing

Virtual Machines

Access Control

Access Control List: list of rights attached to an object

- This allows for a low-level, fine-grained approach
- Simple implementation: Unix permissions (user, group, others)
  - Applies to individual files and directories



Access Control

Access Control List: list of rights attached to an object

- This allows for a low-level, fine-grained approach
- Another implementation: AFS ACLs
  - No longer per-file, but there are many more options:

```
% man fs listacl
% man fs setacl
% man pts
```

Access Control

```
% fs listacl/setacl…
```

The ‘fs listacl’ command displays the access control list (ACL)
The ‘fs setacl’ command sets the access control list (ACL)

AFS permissions:

```
a (administer) Change the entries on the ACL.
d (delete)     Remove/move files and subdirectories
i (insert)     Add files or subdirectories
k (lock)       Set read locks or write locks
l (lookup)     List the files and subdirectories and run fs listacl
r (read)       Read the contents of files
w (write)      Modify the contents of files in the directory
```

Access Control

```
% pts
```

The commands in the pts command suite are the administrative interface to the Protection Server…

There are several categories of commands:

```
Commands to create and remove Protection Database entries:
  pts creategroup
  pts createuser
  pts delete
Commands to administer and display group membership:
  pts adduser
  pts membership
  pts removeuser
etc…
```

Access Control

Example: Updating the ‘engr151’ AFS group

- Check current membership:

```
pts membership engr151
```

- Add users:

```
pts adduser <uniqname> engr151
```

- Remove users:

```
pts removeuser <uniqname> engr151
```

Access Control

Example: Making a sampleProjects directory for ENGR 151

```
% mkdir /afs/umich.edu/class/engr151/<term>/sampleProjects
% cd /afs/umich.edu/class/engr151/<term>
% fs setacl -clear -dir sampleProjects -acl engr151 all
  engr151:members rl system:administrators all
% fs listacl sampleProjects
  Access list for sampleProjects/ is
  Normal rights:
     engr151:members rl
     engr151 rlidwka
     system:administrators rlidwka
```

Services

Daemons (servers/processes): web, mail, SSH, etc.

- Run on well-known ports < 1024 (HTTP, HTTPS, FTP, DNS, SSH, etc.) which are privileged
  - “Regular” processes cannot access these ports

- Daemons need root access to acquire resources
  - A web daemon needs port 80
  - However…we don’t trust the applications that a server may be running (e.g. a php script)

So let’s drop root

Running as root is scary – you can do whatever you want…

Principle of Least Privilege: your process should run with as few resources as possible and give them up when done.

Example: Starting up a web server daemon using a startup script

1. Start service (e.g. web server) as root
2. Acquire resources (e.g. port 80)
3. Drop root by changing server to deprivileged user (e.g. ‘nobody’)
4. The server can now use port 80, but can no longer act as root
5. It can also only do specific things (e.g. write log files)

Process Abstraction

Each process maintains the illusion of resources: RAM, CPU

But in reality, they are shared: examples*???*

How could our process interfere with another process?

Therefore, we really want to run our processes ***in isolation****…*

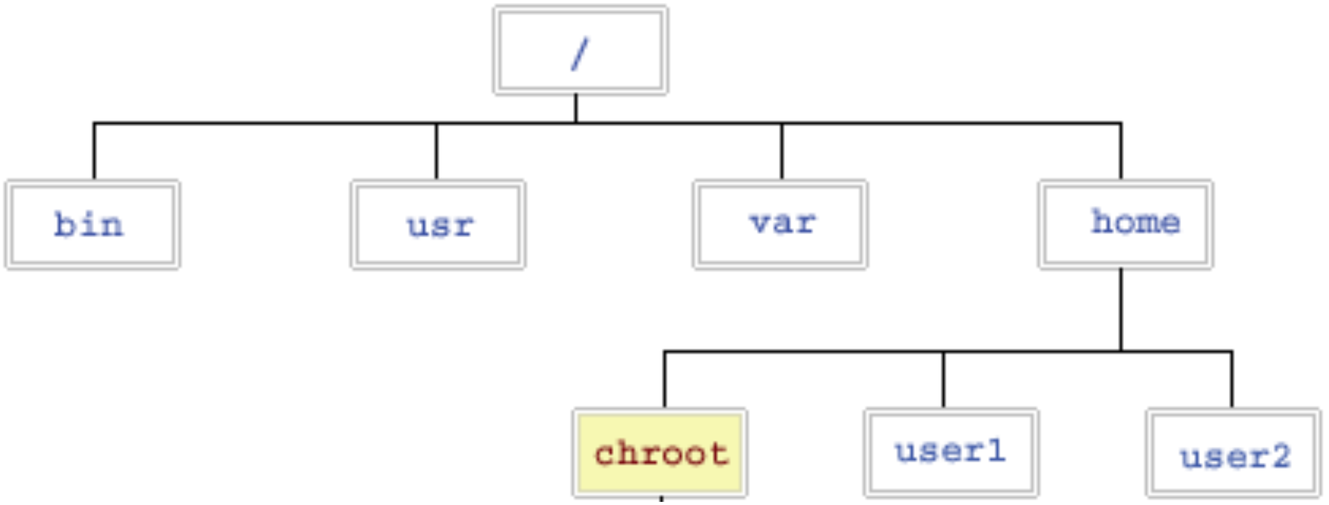
How do we achieve Isolation?

Classic example: chroot (“change root”) ← have to be root to call it

```
% chroot --help
Usage: chroot [OPTION] NEWROOT [COMMAND [ARG]...]
  or:  chroot OPTION
'chroot' changes the root to the directory NEWROOT
(which must exist) and then runs COMMAND with
optional ARGS.  If COMMAND is not specified, the
default is the value of the 'SHELL' environment
variable or '/bin/sh' if not set, invoked with the'-
i' option.
```

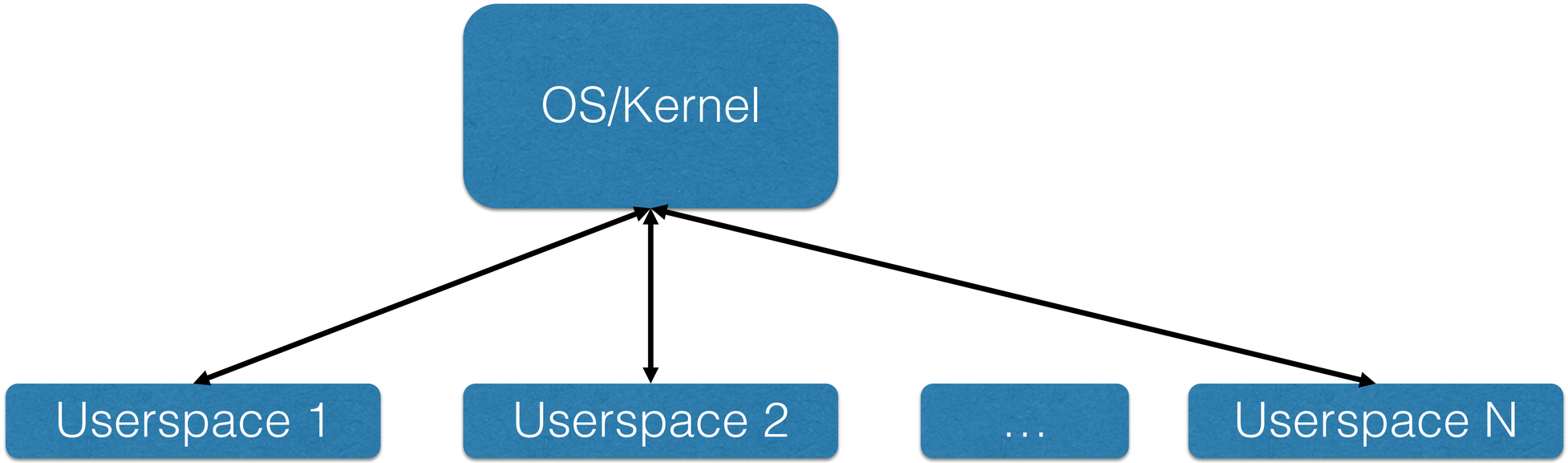
Result: Changes root directory for current process and its children to a different directory.

chroot



Containers – further limiting process interference

A virtualization at the level of the operating system which creates multiple isolated userspace instances (e.g. Docker and Kubernetes)



Restrictions: must be same OS/kernel
Cool things: disk quotas, CPU/RAM limits, root privilege isolation, etc…

Same-OS Abstraction

Each process now maintains the illusion of resources: RAM, CPU, file system, and OS

In reality, the actual RAM, CPU, OS, and file system are still shared…but the containers can manage these resources better.

However, if a process can manage to get into the OS/kernel, you can control all the containers!

How else can we ensure applications behave?

Sometimes the world doesn’t fit nicely into isolation models

For example, the UNIX model advocates letting users screw up all their stuff without messing up the overall system.

We really want a way to enforce specific rights and privileges per application.

Sandboxing

Define a set of allowable actions for a given application

Intercept privileged actions if the applications tries to run them

Example: SELinux (https://en.wikipedia.org/wiki/Security-Enhanced_Linux)

- Every application can have different fine-grained privileges on different files/resources
- For example, a web server can open sockets, but not create directories
- This requires that someone configures the permissions for each application

iOS Sandbox

Hardened container model

Apps can’t touch other apps

Apps can’t touch the system

Apps have to ask for permissions

(Jailbreaking allows you to escape the sandbox)

iOS Sandbox Downsides

Apps need to know what resources they will use ahead of time (e.g. camera, location services, etc…)

Focuses on restrictions, not privileges

Hard to have separate privileges in different parts of an app

Desktop (MacOS) App Sandboxing

Each app runs in its own sandbox (but they don't "feel" like iOS apps).

An app's permissions are bound to its identity at compile time so that only an approved version of the app will run with proper permissions.

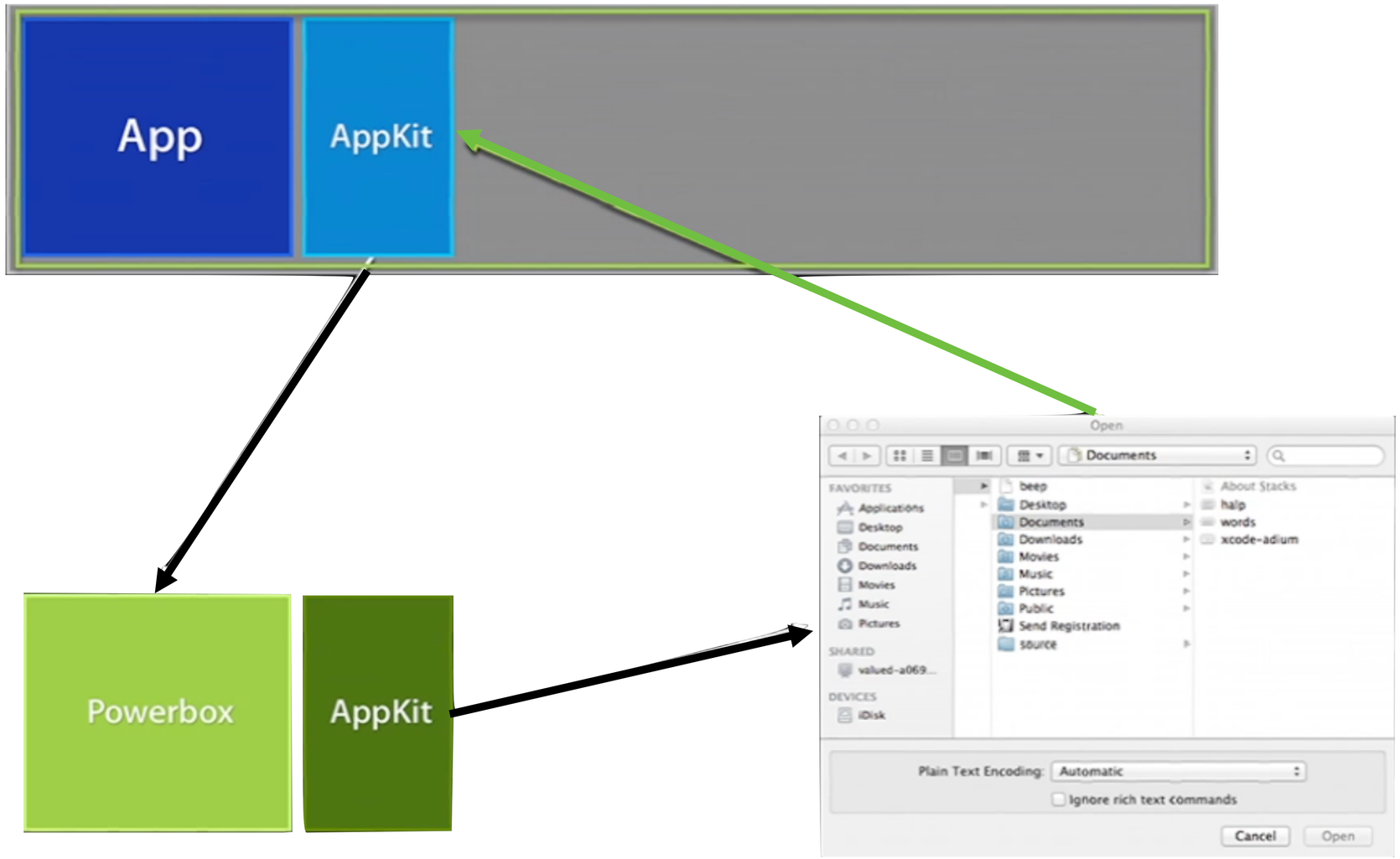
The user controls access to documents, but an app can only access them if the user loads them explicitly.

MacOS Entitlements (Permissions)

These must all be requested at compile time:

- Filesystem
  - User-selected files; Downloads folder
- Network client; network server
- Devices
  - Camera; microphone; printing; USB
- Personal information
  - Address book; calendars; location
- Assets
  - Music; movies; pictures

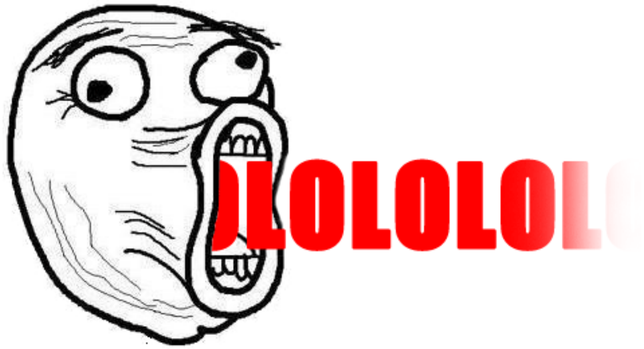
Divining User Intent



Example: Chat

Historically, chat application security:

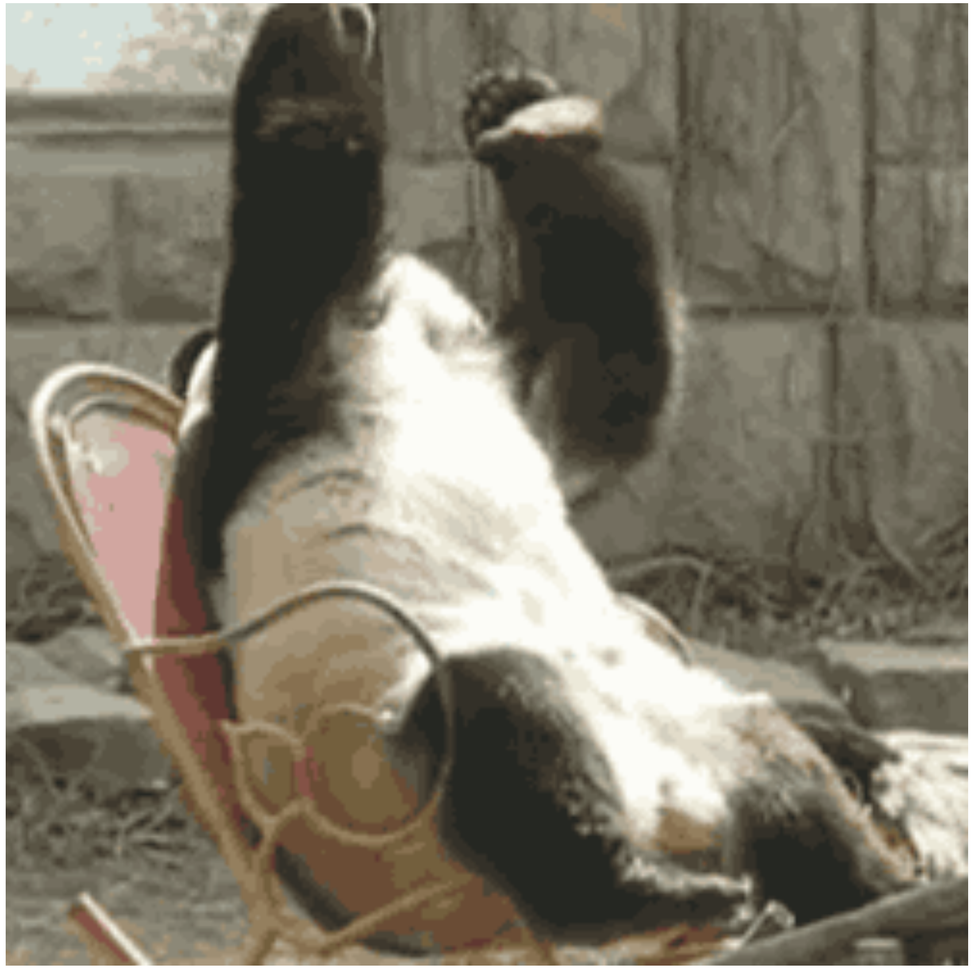
What could a ***compromised*** chat application ***in App Sandbox*** do?



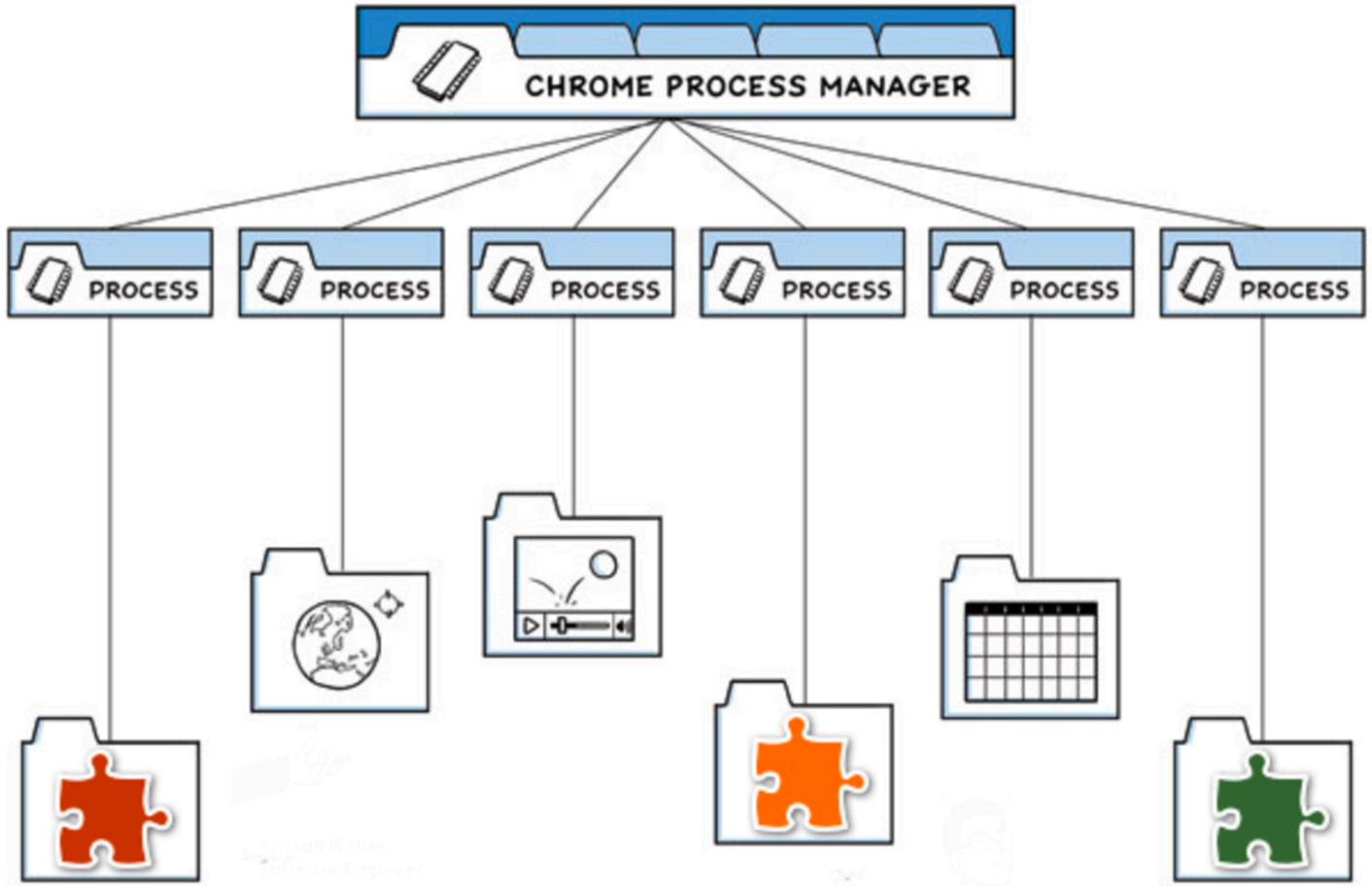
Which of these does a browser not need?

- Filesy
  - Use ted files; Downloads folder
- Network c etwork server
- Devices
  - Camera; micr printing; USB
- Personal informatio
  - Address book; caler cation
- Assets
  - Music; movies; pictures

It needs them all!!!



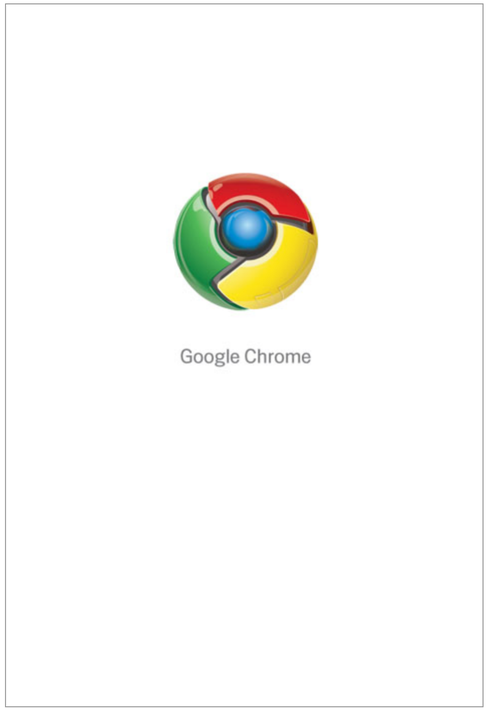
Chrome Sandboxing

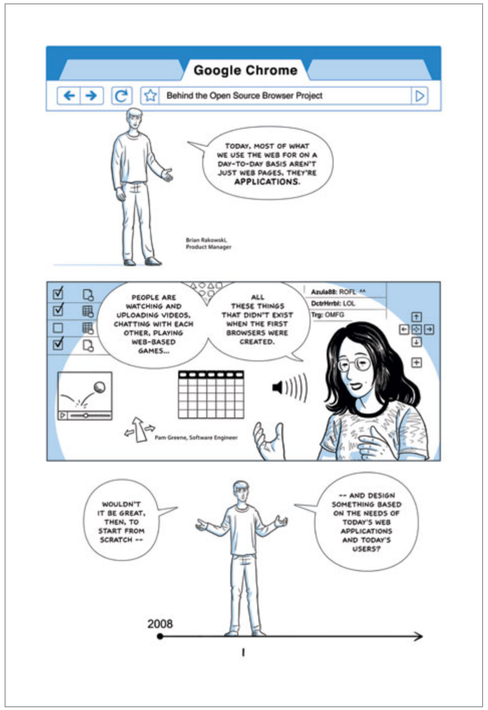


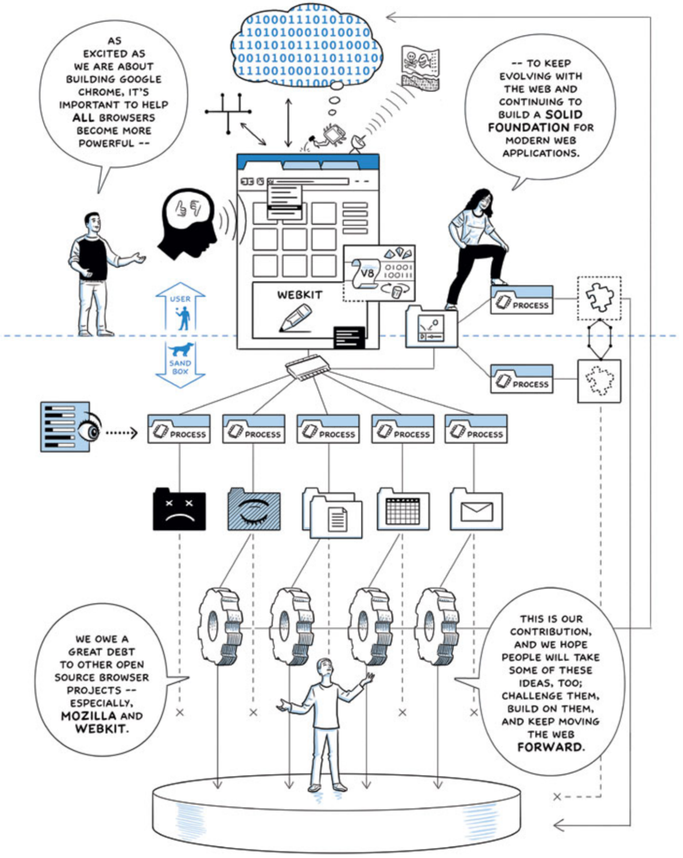
Chrome Sandbox Graphic Novel

Not-actually-required reading:
https://www.google.com/googlebooks/chrome/

Especially Parts 1 (pp. 3–11) and 4 (pp. 25–33)







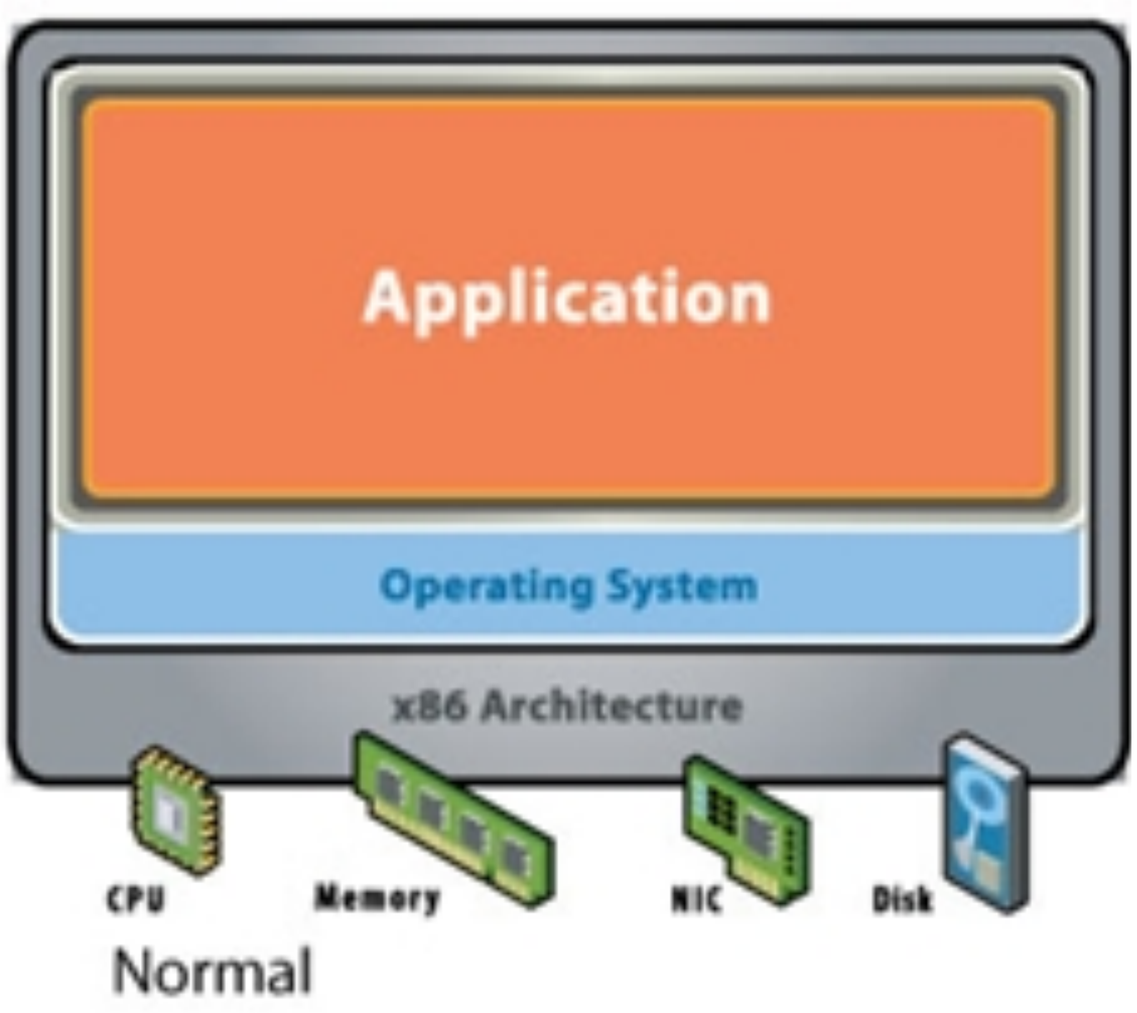
So Far

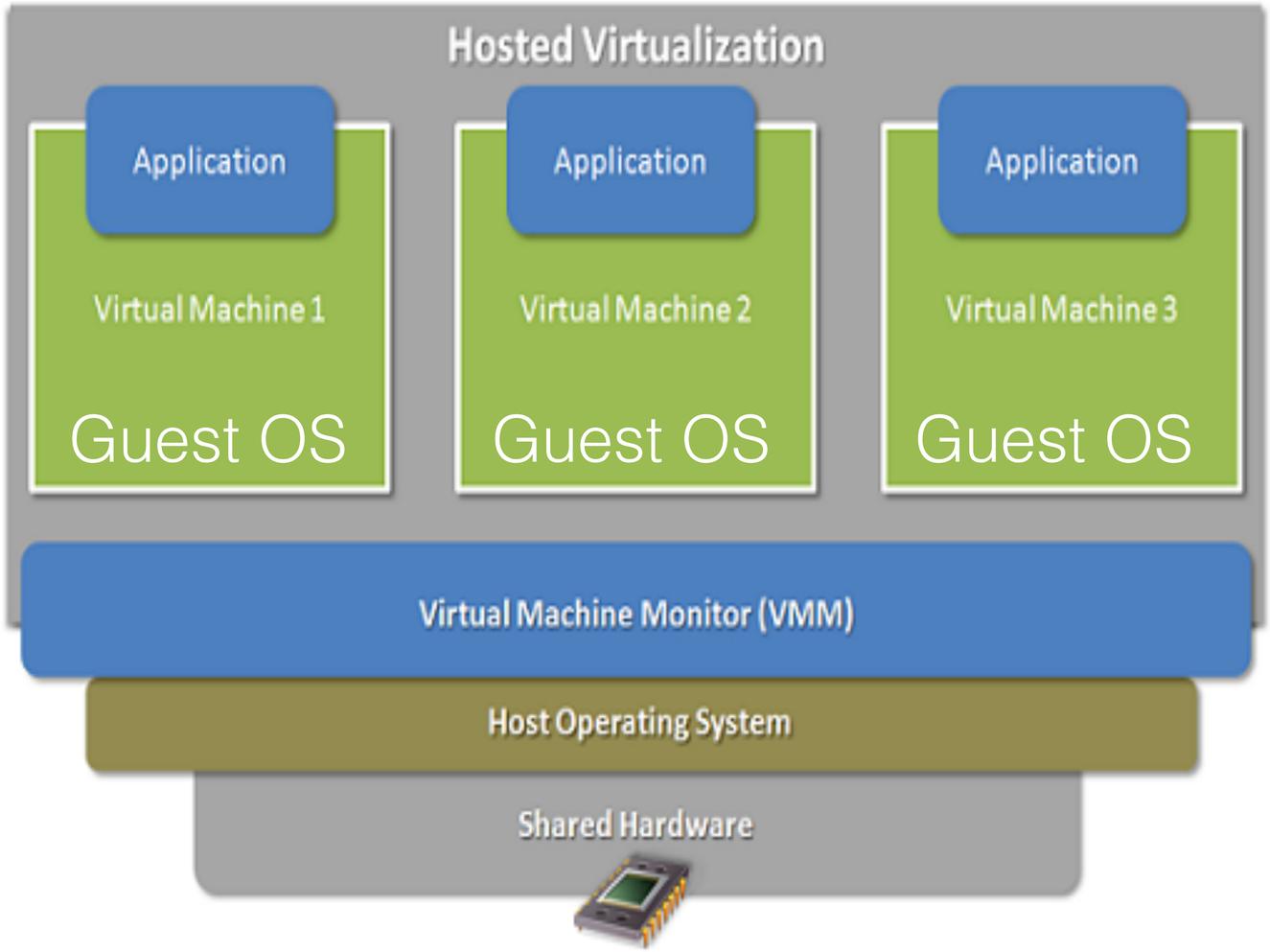
Common platform

What about multiple OSes?

Virtual Machines

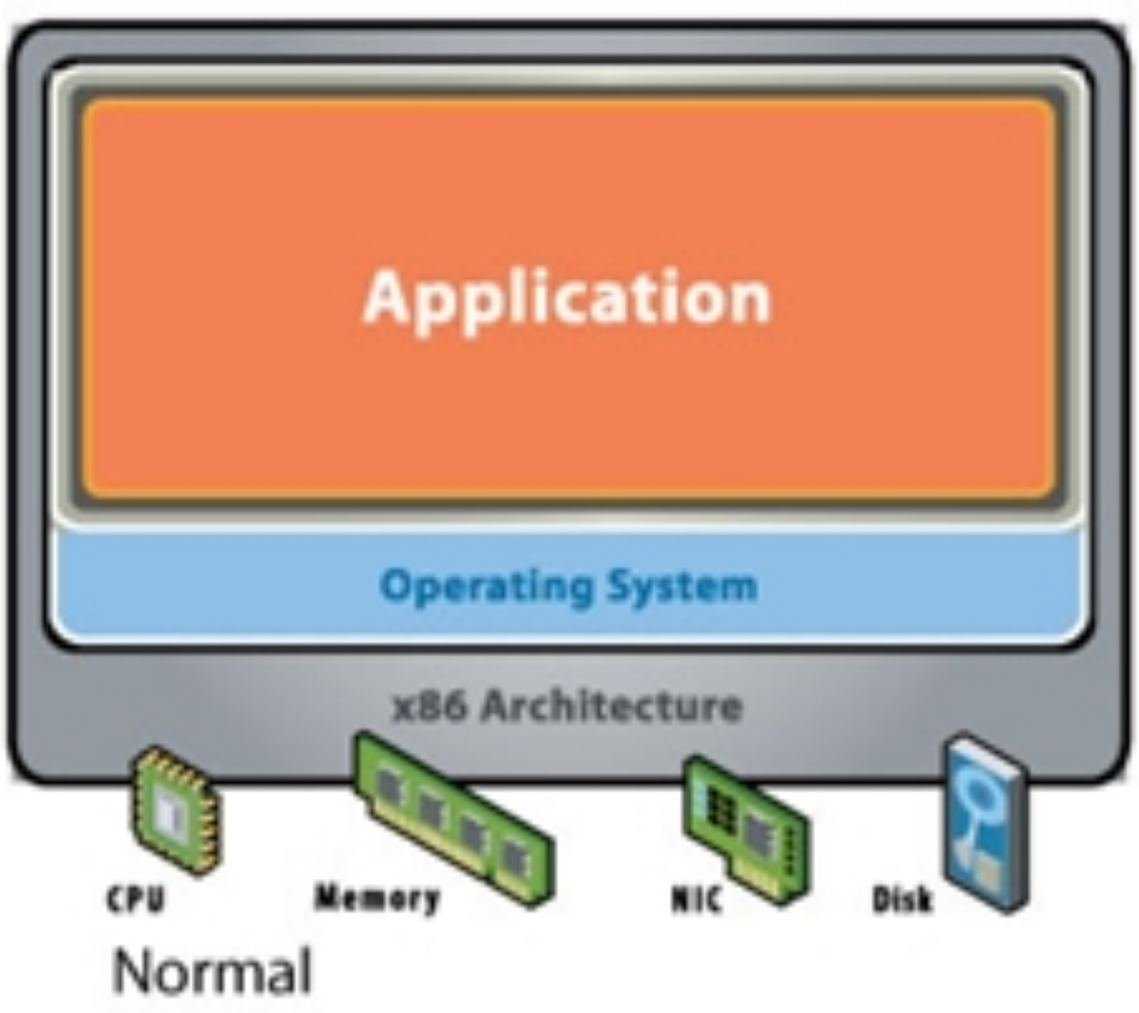
Laptop use case



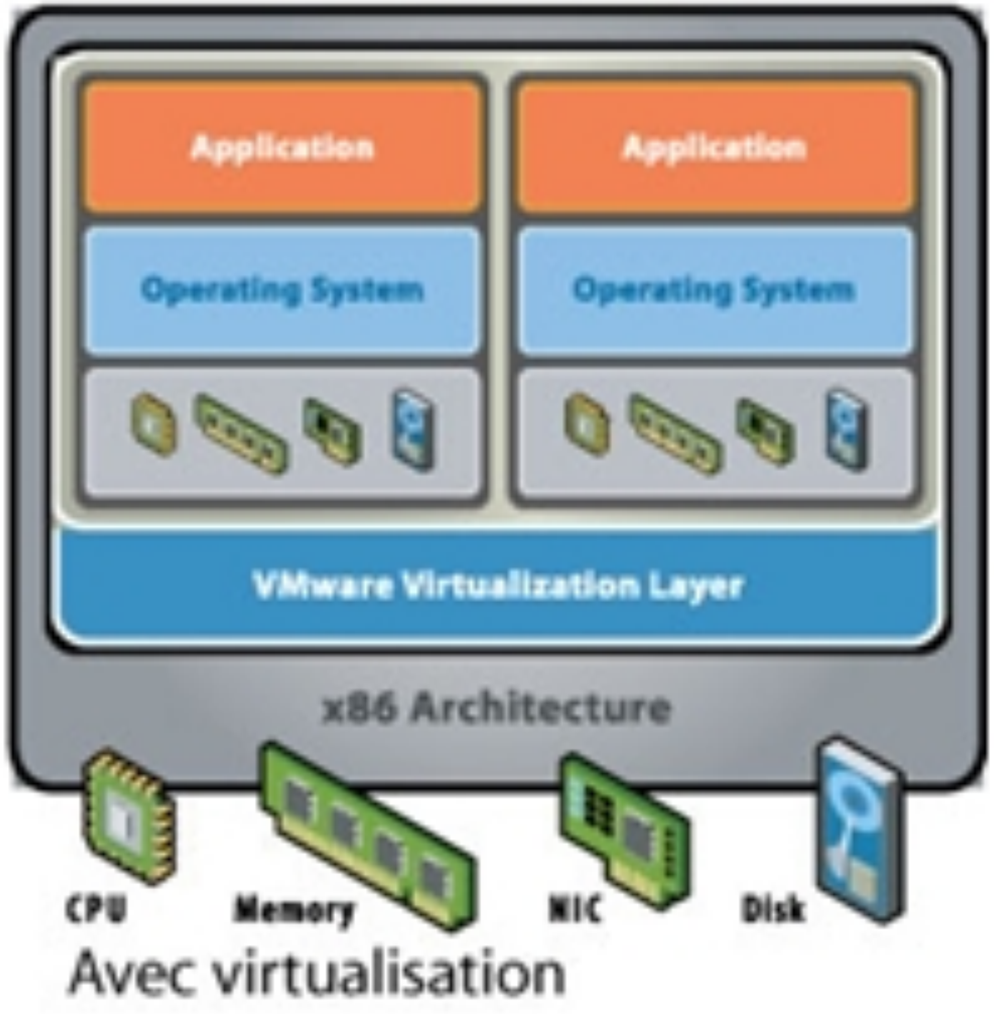


Virtual Machines

More efficient use case
(use Hypervisor layer instead of host OS)



Normal



Avec virtualisation

VMs vs. Containers

Vagrant vs. Docker - https://youtu.be/pGYAg7TMmp0?t=39

Advantages/Disadvantages of each?