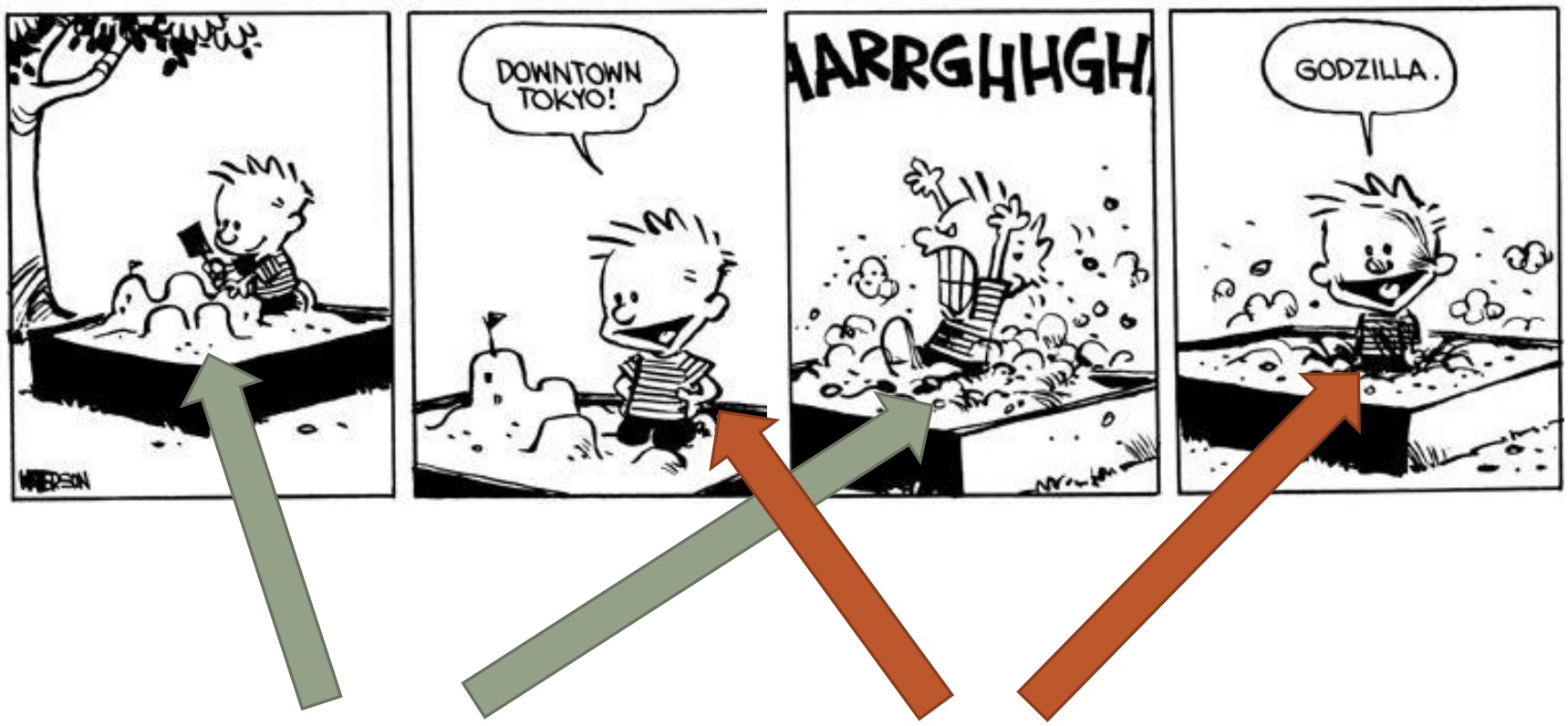
Sandboxing and Virtualization

CS361S

FALL 2020

LECTURE NOTES

Sandboxing: the Concept



(Copyright Use)

There is no need to request permission if you wish to use a cartoon for educational or classroom usage. Classroom use refers to public and private schools (grade K-12), home schooling (grade K-12), college dissertations, college thesis papers and other restricted college usage only. They MAY NOT be used in presentations outside of the classroom such as employee training, as handouts for non-student use or in any manner that is not a classroom setting at the K to college level. All other use will be charged a permission fee. If your use falls under the classroom use as stated above, you may use up to seven (7) cartoons per year at no costs as part of our fair use policy.

- http://syndication.andrewsmcmeel.com/licensing_permissions/educational_use

Code Trustworthiness

Running a program is ***DANGEROUS***

You are allowing the author(s) to ***CONTROL*** your computer!

Large corporate vendors have incentives to behave

- Commercial incentives (selling product)
- Brand/company reputation
- Lawsuit threats

Even then, bad code happens (e.g., Sony Rootkit)

Less Trustworthy Code

Until the Internet, most code written by corporate vendors

After the Internet, potential code from every website

Incentives of each author varies but includes malice

Impossible to tell if a program is good or bad (halting problem)

(Note: we should all be terrified of the Internet)

The Problem with Computers

Computer processors do ***exactly*** what they're told

They have no ability to decide if they ***should***

What if they're told to do something ***harmful***?

A lot of tech goes into figuring out what should be done

- Operating System
- Anti-virus
- Device permissions

A Sandbox:

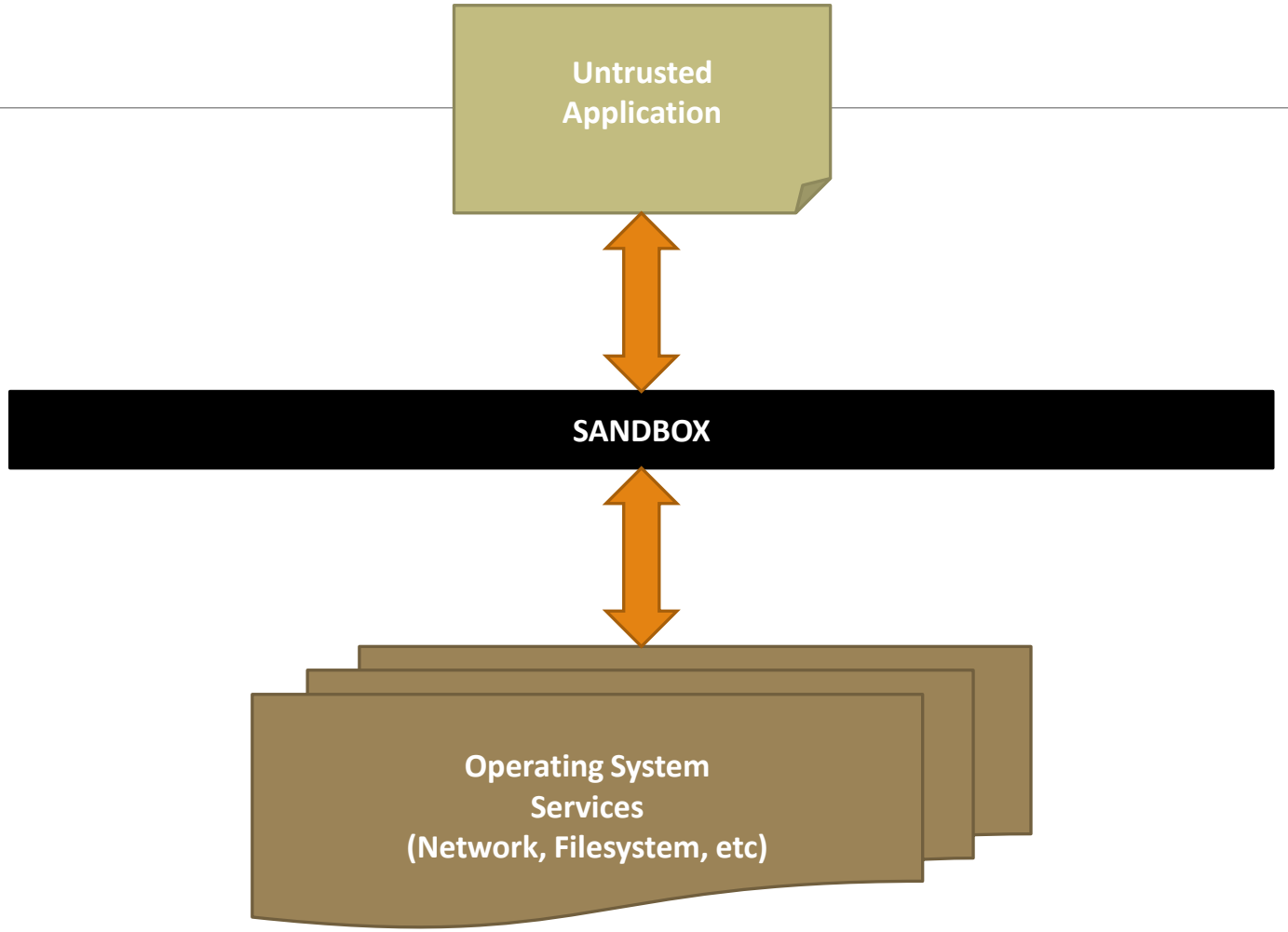
Concept: environment where destruction doesn't matter

In practice: a virtualized layer between code and resources

The virtualization layer enables:

- Policy Enforcement
- "Sensor" (inputs to App) Translations
- "Command" (outputs from App) Translations

Result: ***no direct access to critical resources***


Untrusted
Application
SANDBOX
Operating System
Services
(Network, Filesystem, etc)

Policy Enforcement

Most common use of a Sandbox

Each incoming request ***and response*** can be inspected

- Can be allowed, denied, **or modified**
- Evaluate type, parameters, state of the system, etc

Examples:

- Network Access (Deny, Same-Origin Policy)
- File Access (Read Only, Write-to-Temp)
- Even memory allocations

Sensor/Command Translations

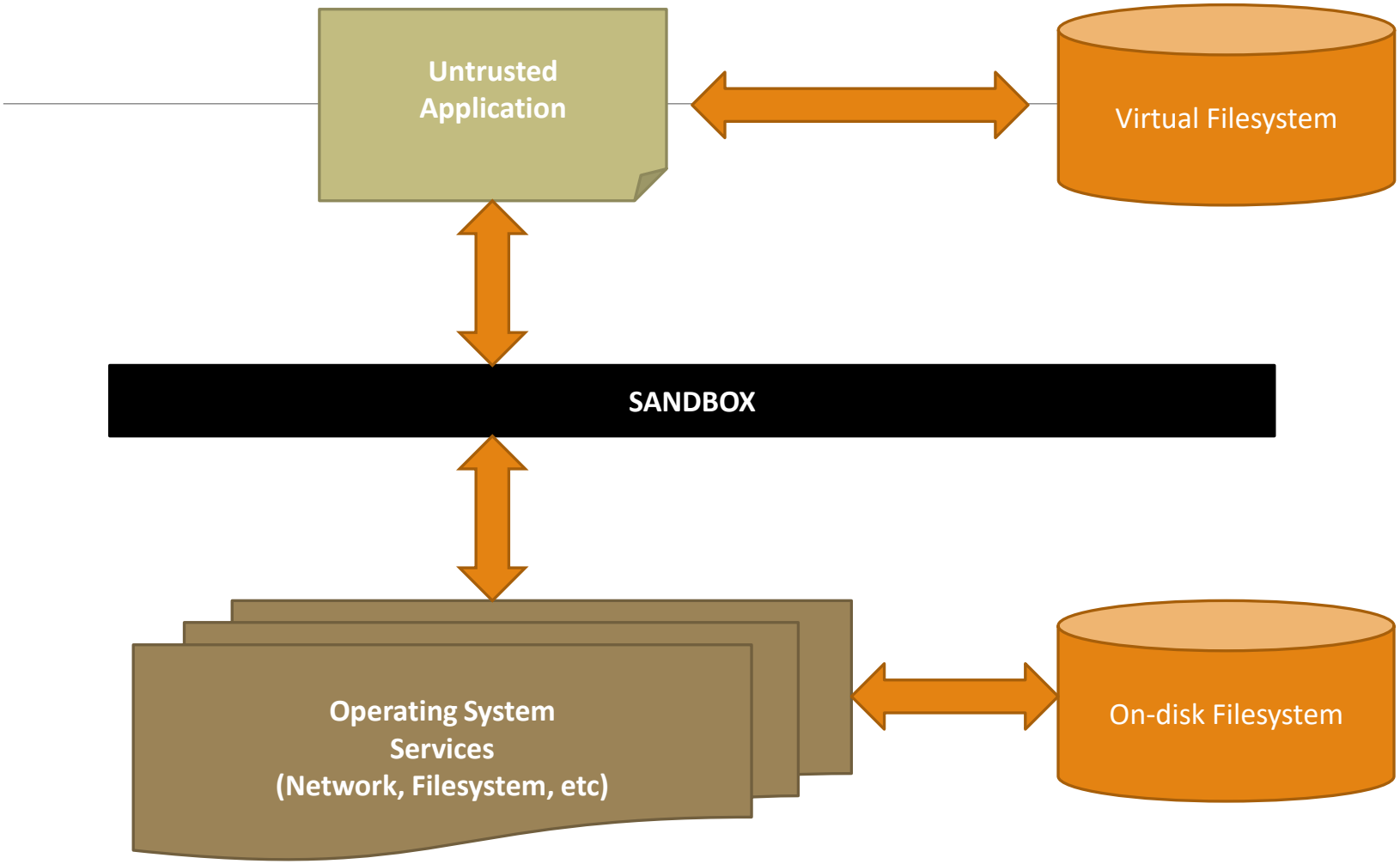
Policy is not just about allow/deny but rewrite/modify

Command rewriting:

- Rewrite command (e.g. syscall) with safer parameter
- Rewrite command (e.g. syscall) to safer call with similar semantics

Sensor rewriting:

- (control what the application “sees”)
- Rewrite entire sensor system (e.g., virtual filesystem)


Untrusted Application
Virtual Filesystem
SANDBOX
Operating System Services (Network, Filesystem, etc)
On-disk Filesystem

Mobile Code Sandboxes

Java was the first to introduce a language sandbox

Java code is virtualized by definition

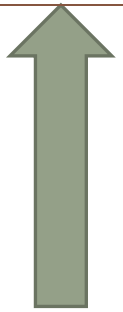
- All code compiles to an intermediate byte code
- A Java Virtual machine executes the byte code
- One VM per architecture
- Permits “write once run anywhere”

Every access to a resource is mediated by the VM

VM has a security policy that determines access

Java mkdir Example

```
public boolean mkdir(String path) throws IOException {
    SecurityManager security = System.getSecurityManager();
    if (security != null) {
        security.checkWrite(path);
    }
    return mkdir0();
}
```



Low-level (raw) access

Policy check/Reference Monitor

Java Applets

Default Java applications have almost no policy limitations

Applets, designed for web use, had extensive limitations

- Network connections limited by same origin policy
- Can only read/write to /tmp
- etc

Sandbox Woes

From the very beginning, Java applets failed

To many ways to “escape” the sandbox

To many vulnerabilities in VM

“Solution” of signed code was not sufficient

Performance never justified the risks

No longer recommended, supported

The PyPy Sandbox

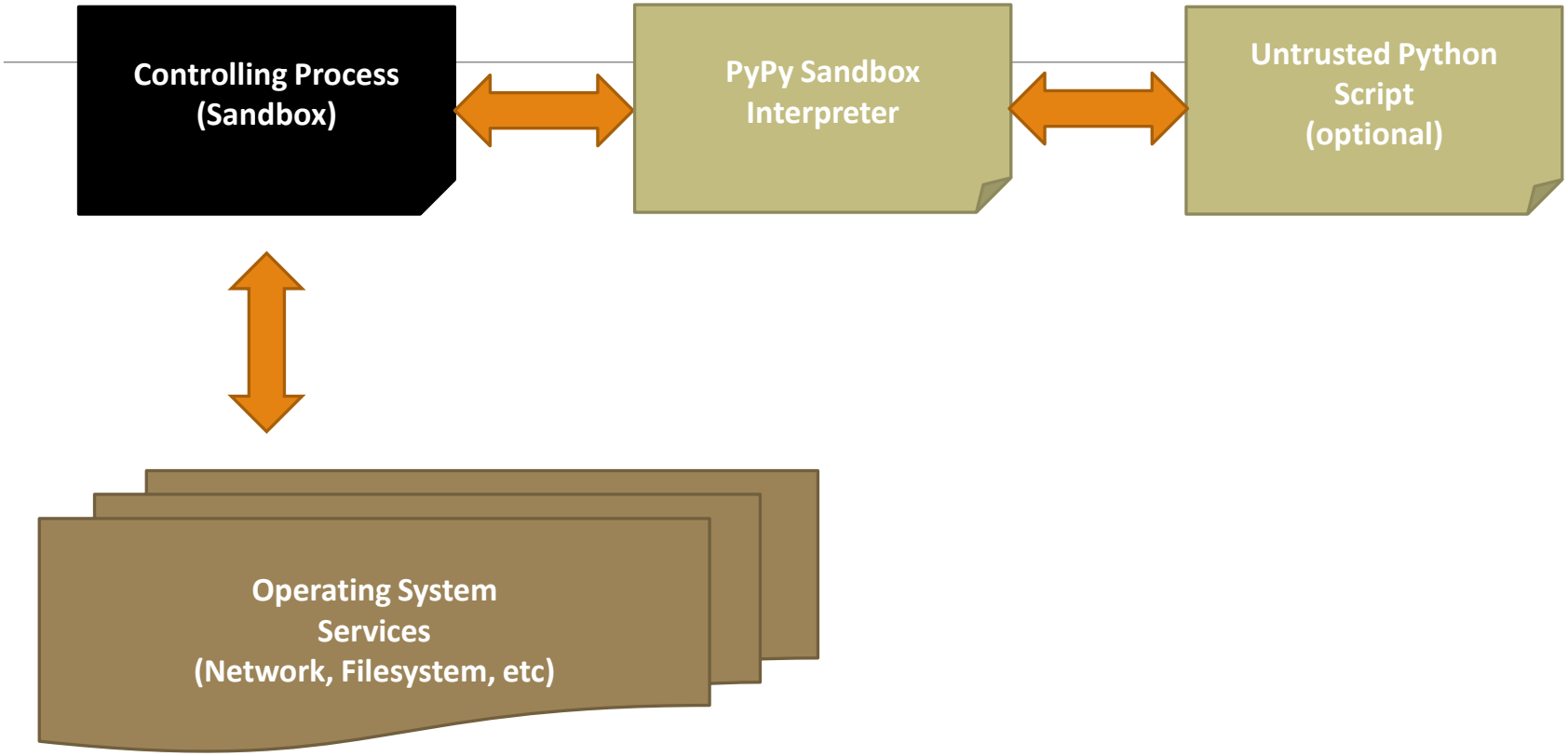
Useful for learning the sandbox concepts

Creates a limited PyPy (Python) Interpreter

- No direct calls to the OS (system calls, etc)
- Does not allow dynamic libraries

Enforcement process receives OS calls over a pipe

- For permitted calls, performs the call and sends back result
- Can modify the request and/or results


Controlling Process
(Sandbox)
PyPy Sandbox
Interpreter
Untrusted Python
Script
(optional)
Operating System
Services
(Network, Filesystem, etc)

Infinite Variety of Sandboxes

Different enforcement processes enforce different policies

Controlling process does not have to be Python

The PyPy project provides a default controlling process called “pypy_interact.py”

- Can run a python “shell” or execute a script
- Many OS subsystems completely disabled including network
- Read only virtual file system
  - /bin – virtual bin directory with pypy and a few required directories
  - /tmp – temp directory that potentially maps to a real directory
  - NOTE: the interpreter lives in the sandbox and executes the script from virtual /tmp!

```
$ mkdir my_sandbox_tmp
$ echo "this is a test" > my_sandbox_tmp/datafile.txt
$ ./pypy_interact.py --tmp=my_sandbox_tmp pypy3-c-sandbox
```

```
>>>> import os
>>>> os.listdir('/')
['dev', 'bin', 'tmp']
```

```
>>>> os.listdir('/tmp')
['datafile.txt']
```

```
>>>> f = open('/tmp/datafile.txt')
>>>> f.read()
'this is a test\n'
```

```
>>>> open('/tmp/newfile.txt','w+')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: '/tmp/newfile.txt'
>>>> open('/tmp/datafile.txt','a+')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
PermissionError: [Errno 1] Operation not permitted: '/tmp/datafile.txt'
```

Running an Untrusted Script

Contents of “dangerous_script.py”

```
import os

print("Script Current Working Dir:
{}".format(os.getcwd()))

print("Contents of root dir:
{}".format(os.listdir('/')))

print("Try to delete /tmp dir with a system call.")

os.system('rm -rf /tmp')
```

```
$ ls my_sandbox_tmp/
dangerous_script.py  datafile.txt

./pypy_interact.py --tmp=my_sandbox_tmp pypy3-c-sandbox /tmp/dangerous_script.py

Script Current Working Dir: /tmp
Contents of root dir: ['bin', 'tmp', 'dev']
Try to delete /tmp dir with a system call.
Traceback (most recent call last):
  File "/tmp/dangerous_script.py", line 5, in <module>
    os.system('rm -rf /tmp')
RuntimeError
```

Sample Enforcement Functions

```
def do_ll_os__ll_os_write(self, fd, data):
    if fd == 1:
        self._output.write(data.decode())
        self._output.flush()
        return len(data)
    if fd == 2:
        self._error.write(data.decode())
        return len(data)
    raise OSError("trying to write to fd %d" % (fd,))


def do_ll_os__ll_os_read(self, fd, size):
    f = self.get_file(fd, throw=False)
    if f is None:
        return super().do_ll_os__ll_os_read(fd, size)
    else:
        if not (0 <= size <= (2**64)):
            raise OSError(errno.EINVAL, "invalid read size")
        # don't try to read more than 256KB at once here
        return f.read(min(size, 256*1024))
```

OS Sandboxes

Android and Apple sandbox apps for devices

Goal #1: Isolate applications from each other

- Filesystem is usually per-app
- Limited cross-app interactions

Goal #2: Protect applications from spying

- Access permissions for hardware

Some movement toward sandboxing desktops too

Sample Apple Permissions

Network

`com.apple.security.network.server`

A Boolean value indicating whether your app may listen for incoming network connections.

`com.apple.security.network.client`

A Boolean value indicating whether your app may open outgoing network connections.

Hardware

Camera Entitlement

A Boolean value that indicates whether the app may capture movies and still images using the built-in camera.
Key: com.apple.security.device.camera

`com.apple.security.device.microphone`

A Boolean value that indicates whether the app may use the microphone.

File Access

`com.apple.security.files.user-selected.read-only`

A Boolean value that indicates whether the app may have read-only access to files the user has selected using an Open or Save dialog.

Full Virtualization

The “ultimate” sandbox is a completely fake computer

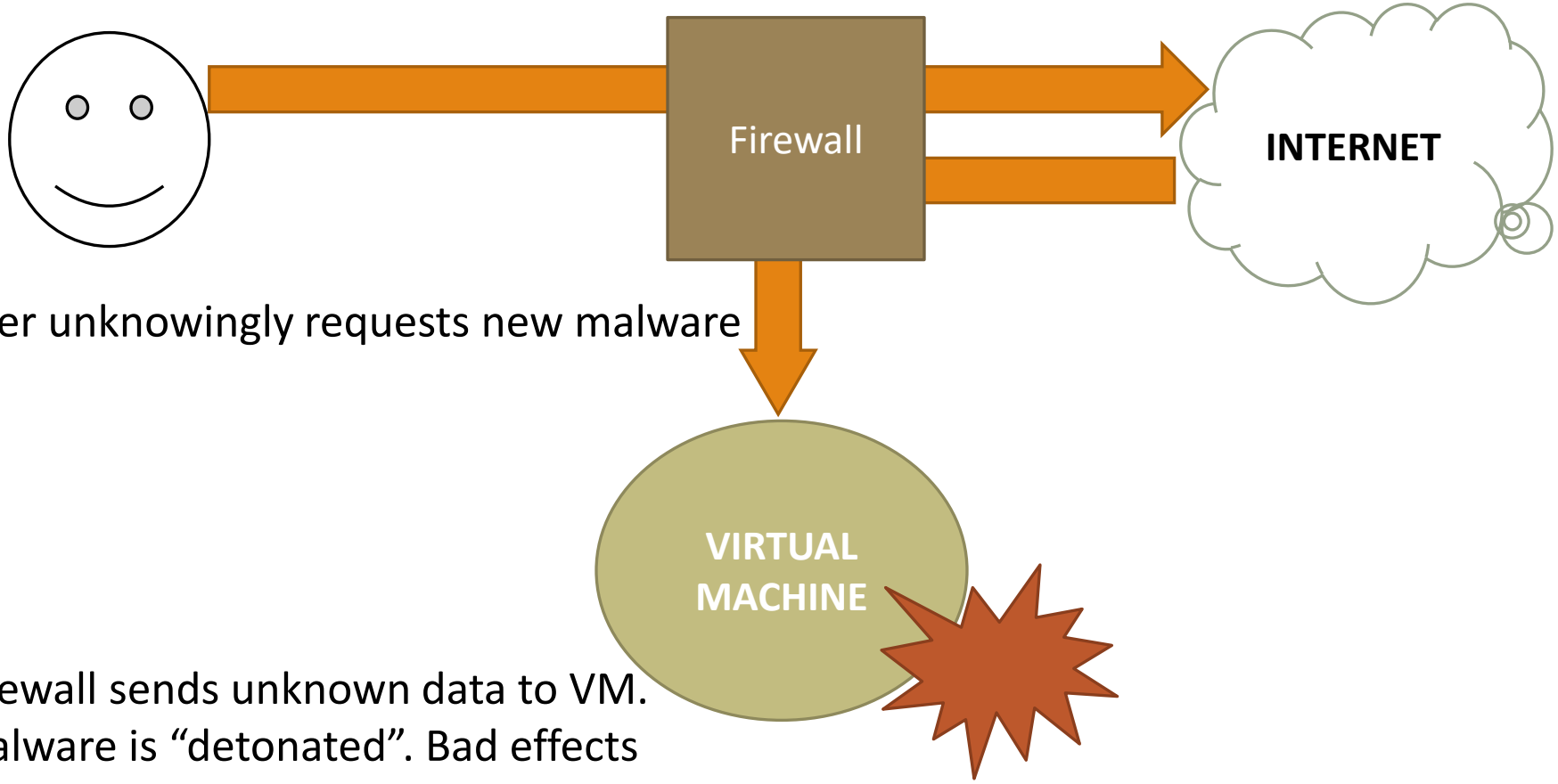
An infected virtual system can be discarded

Checkpoints enable returning to a past state

Data backups prevent lost data

Virtual machines can be fully isolated

Use 1: Controlled Detonation



User unknowingly requests new malware

Firewall sends unknown data to VM. Malware is “detonated”. Bad effects observed and download is blocked.

Limitations

No command line parameters/user interactions

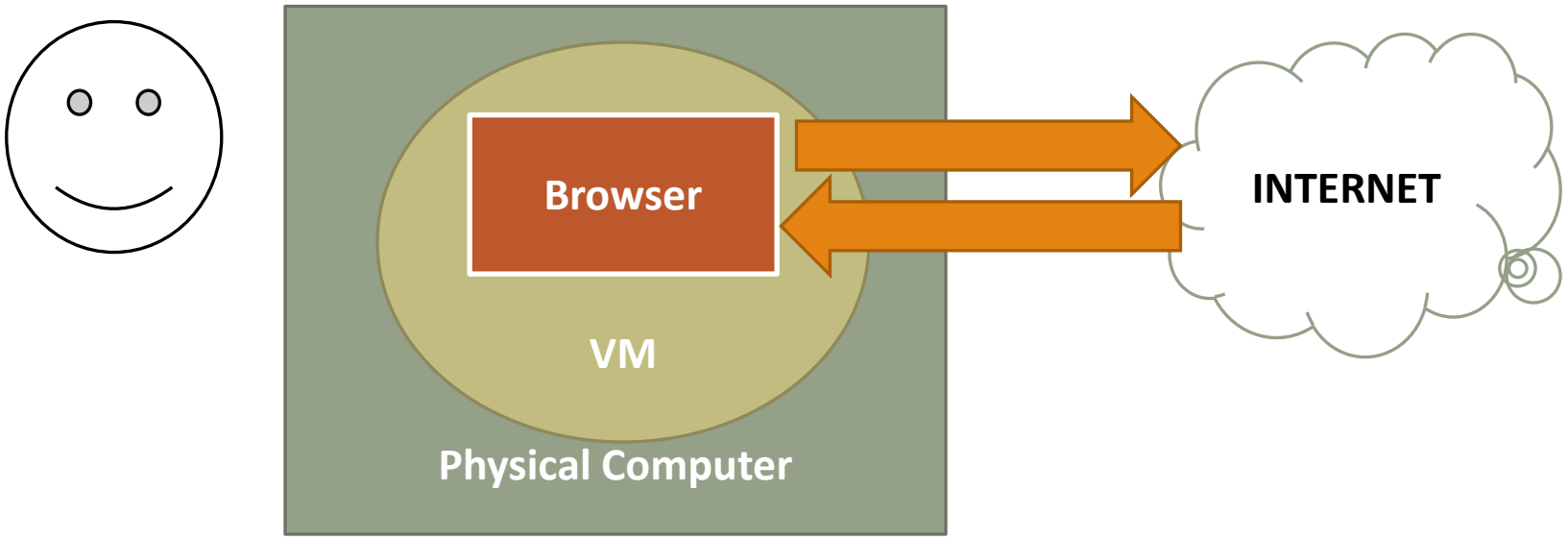
Limited time execution (e.g., 1 minute)

Bad effects must be observable within these constraints

Some malware checks for VM

- VM Graphics drivers
- Filesystem irregularities
- Fingerprints/profiling

Use #2: Browsing Proxy



Limitations

User may copy malware from VM to physical machine

High performance costs

Many VM's not "locked down" out of the box

- May still have network access
- User may configure easy file transfers