Sandboxing

CS-576 Systems Security

Instructor: Georgios Portokalidis

Fall 2018

Sandboxing Means Isolation

Why?

Software has bugs

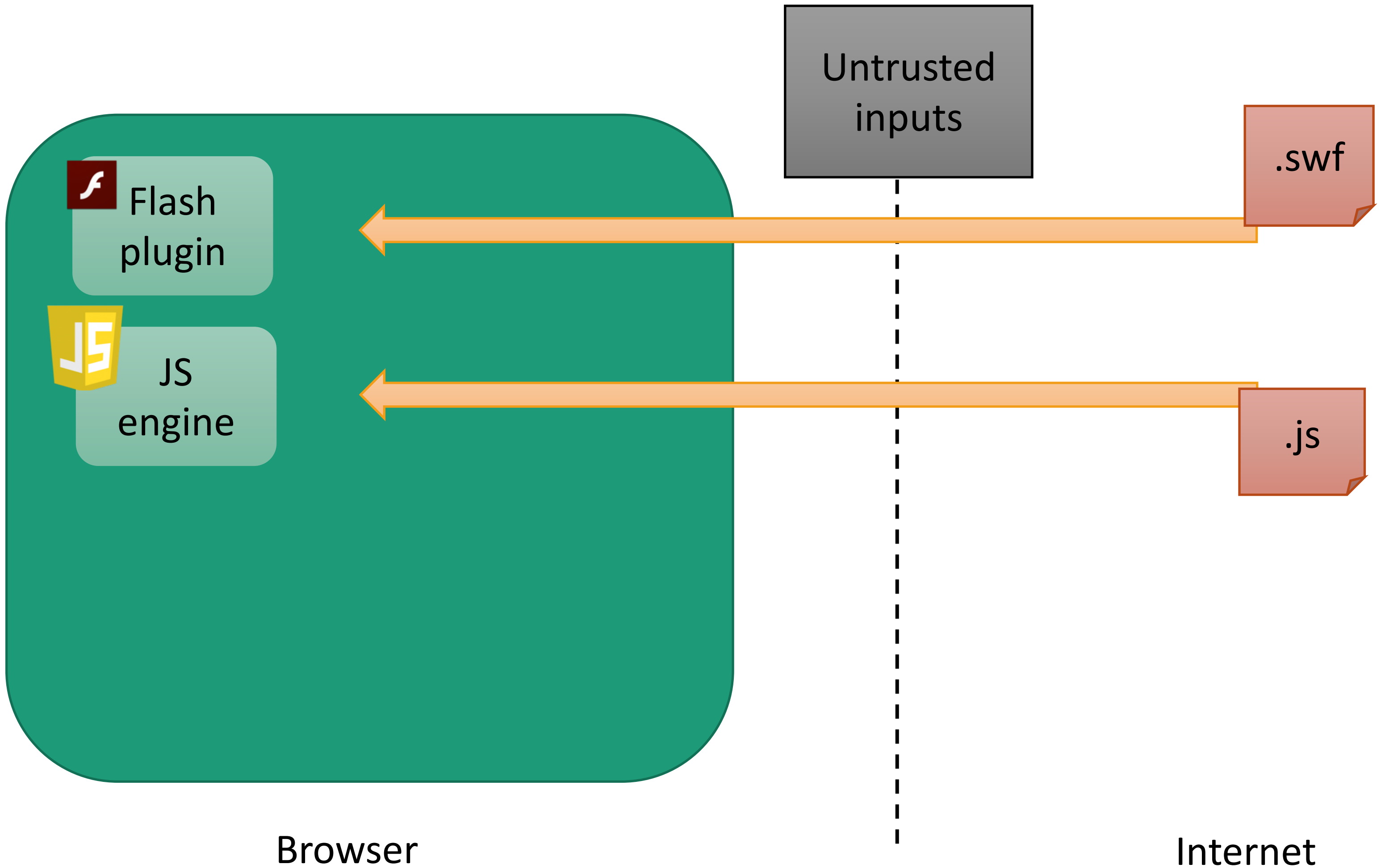
Defenses slip

Untrusted code

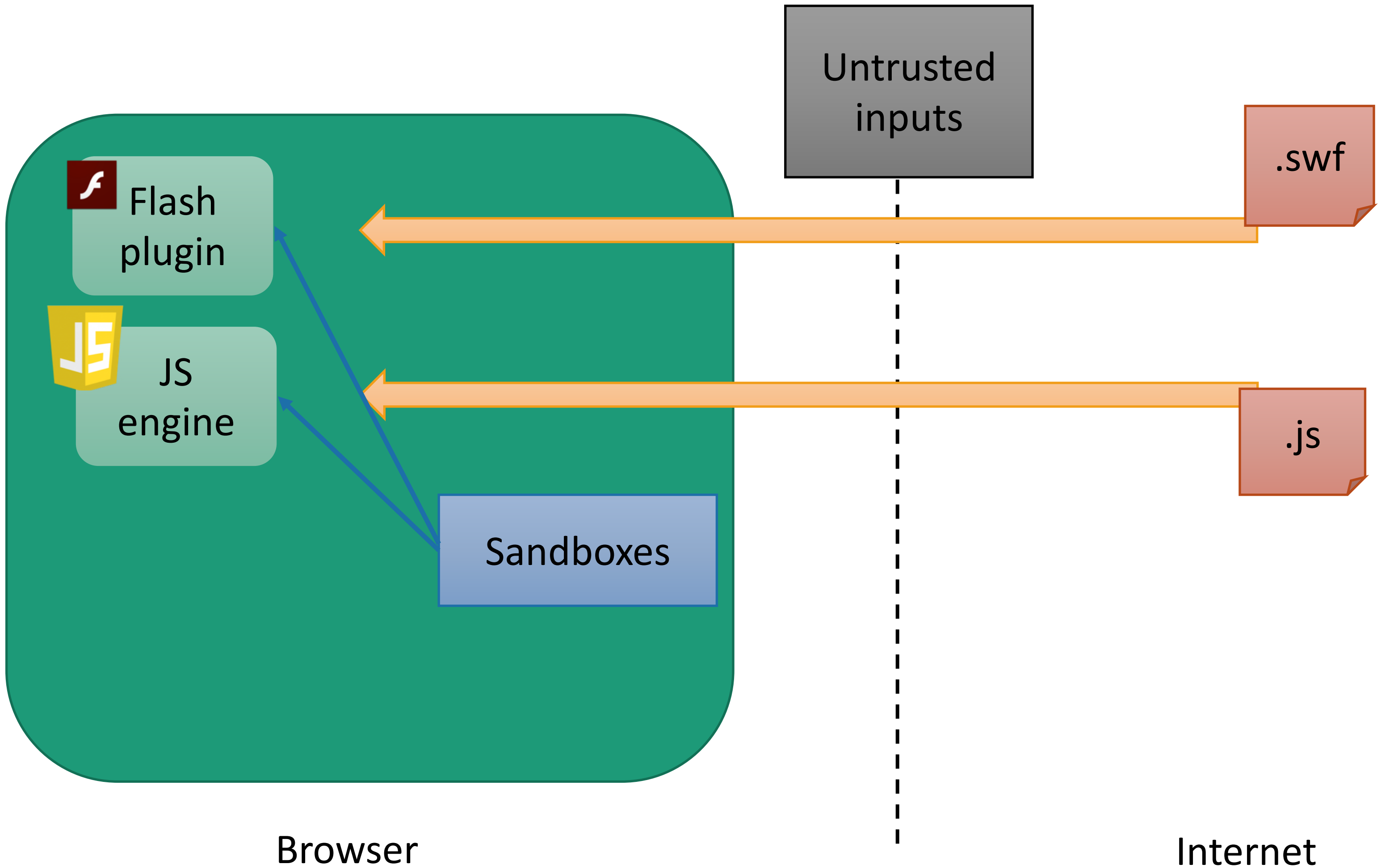
Compartmentalization limits interference and damage!

“a sandbox is a security mechanism for separating running programs”
-- wikipedia

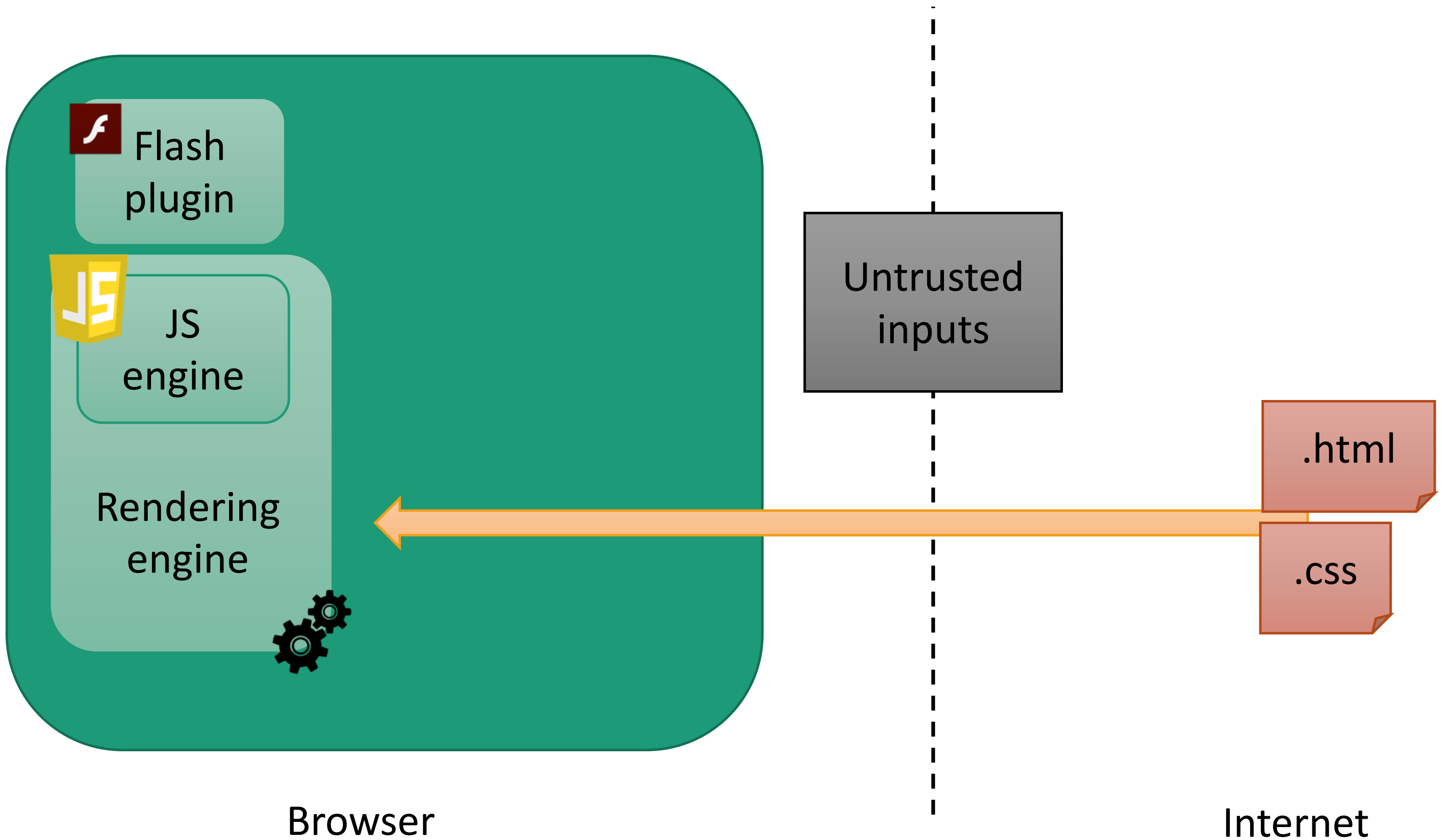
Opportunities for Sandboxing: Browsers



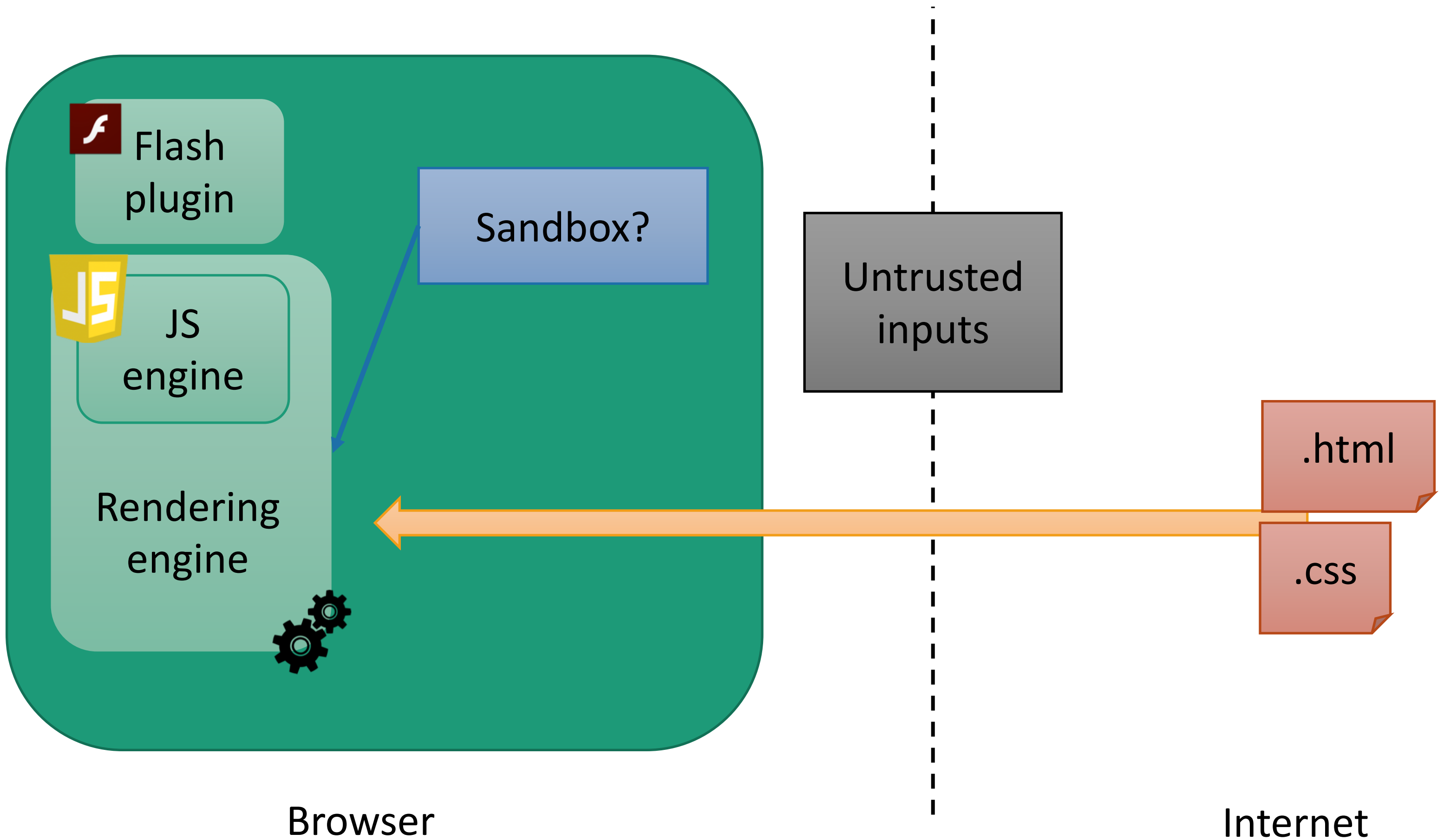
Opportunities for Sandboxing: Browsers



Untrusted Code in Browsers



Untrusted Code in Browsers



Sandboxing Methods

VM-based

- Run entire OS in isolation

OS-based

- Process-wide
- Available system calls and capabilities are restricted

Language-based

- Language isolates components

Inline reference monitor

- Integrated into untrusted code during compilation, code generation, or through emulation
- Security checks injected to enforce policy

Sandboxing Methods

VM-based

- Run entire OS in isolation

OS-based

- Process-wide
- Available system calls and capabilities are restricted

Language-based

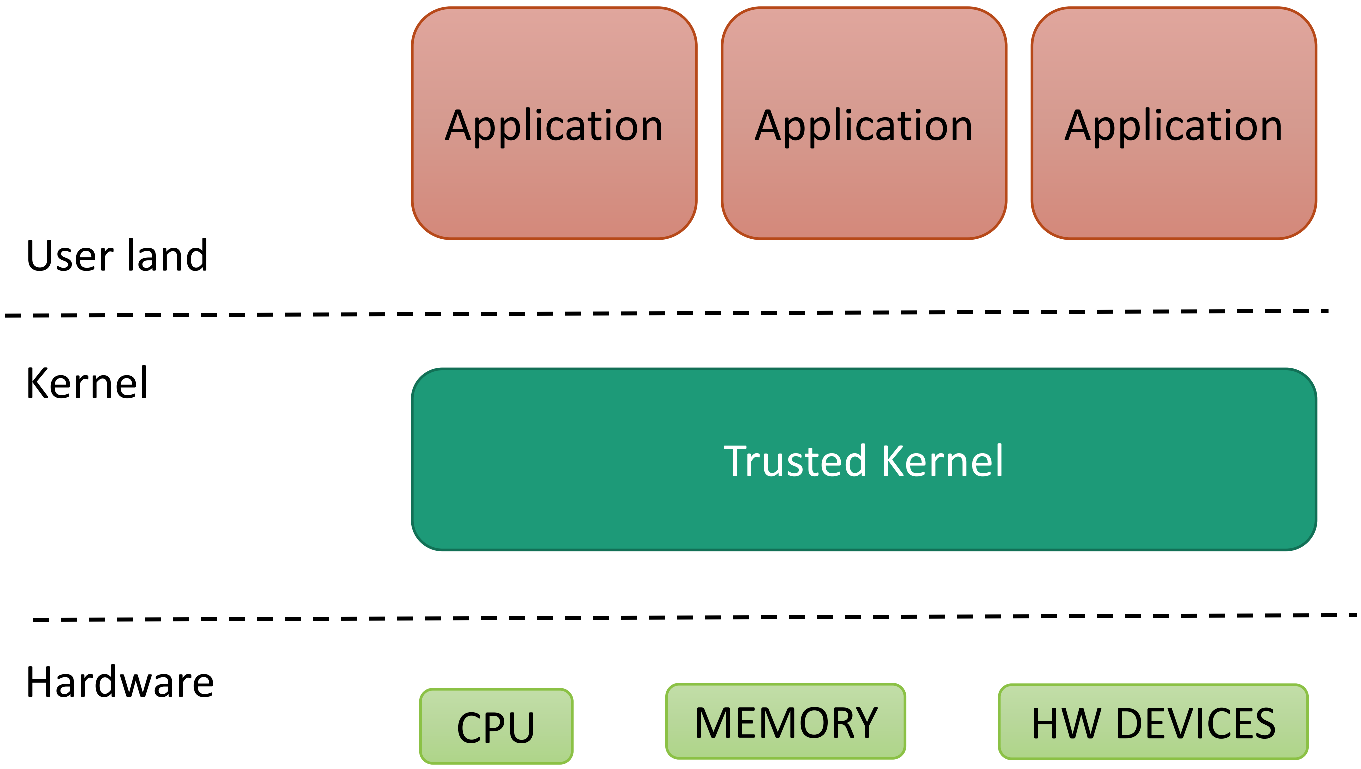
- Language isolates components

Inline reference monitor

- Integrated into untrusted code during compilation, code generation, or through emulation
- Security checks injected to enforce policy

Lets Refresh What We Know About OSes

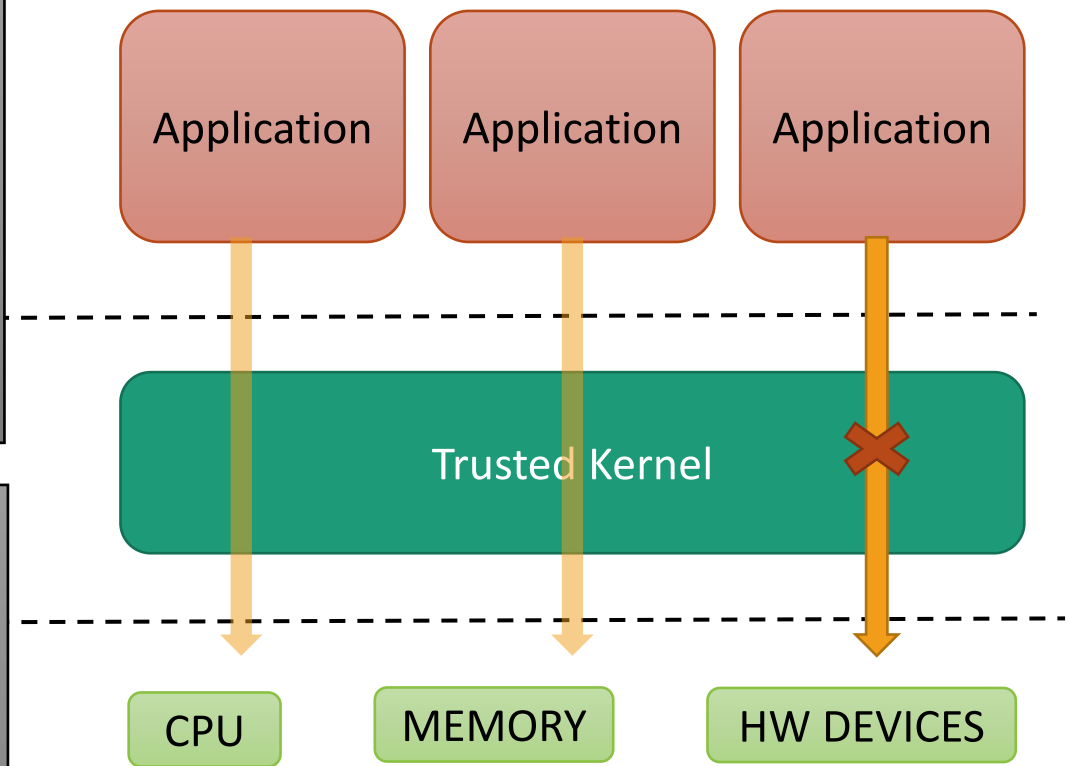
OS Access Control of HW



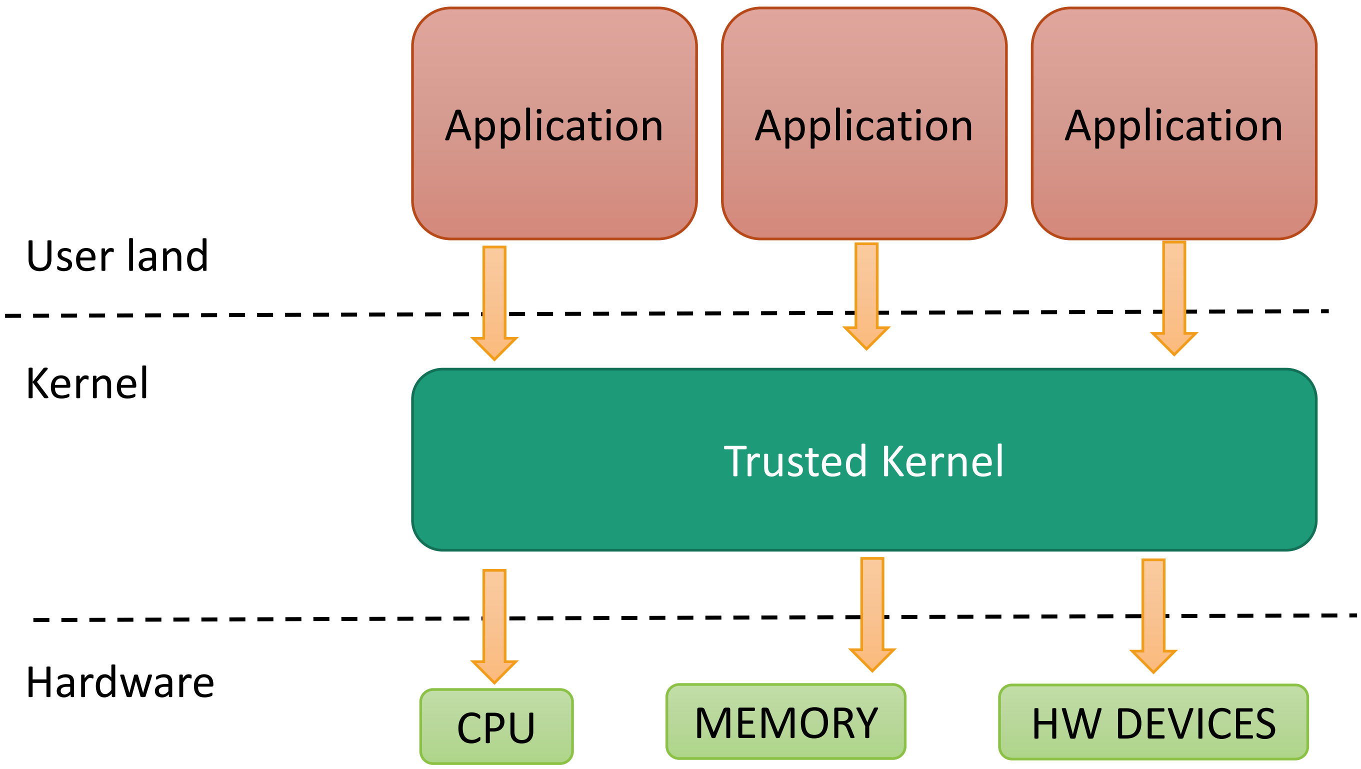
OS Access Control of HW

User space has restricted access to CPU and memory

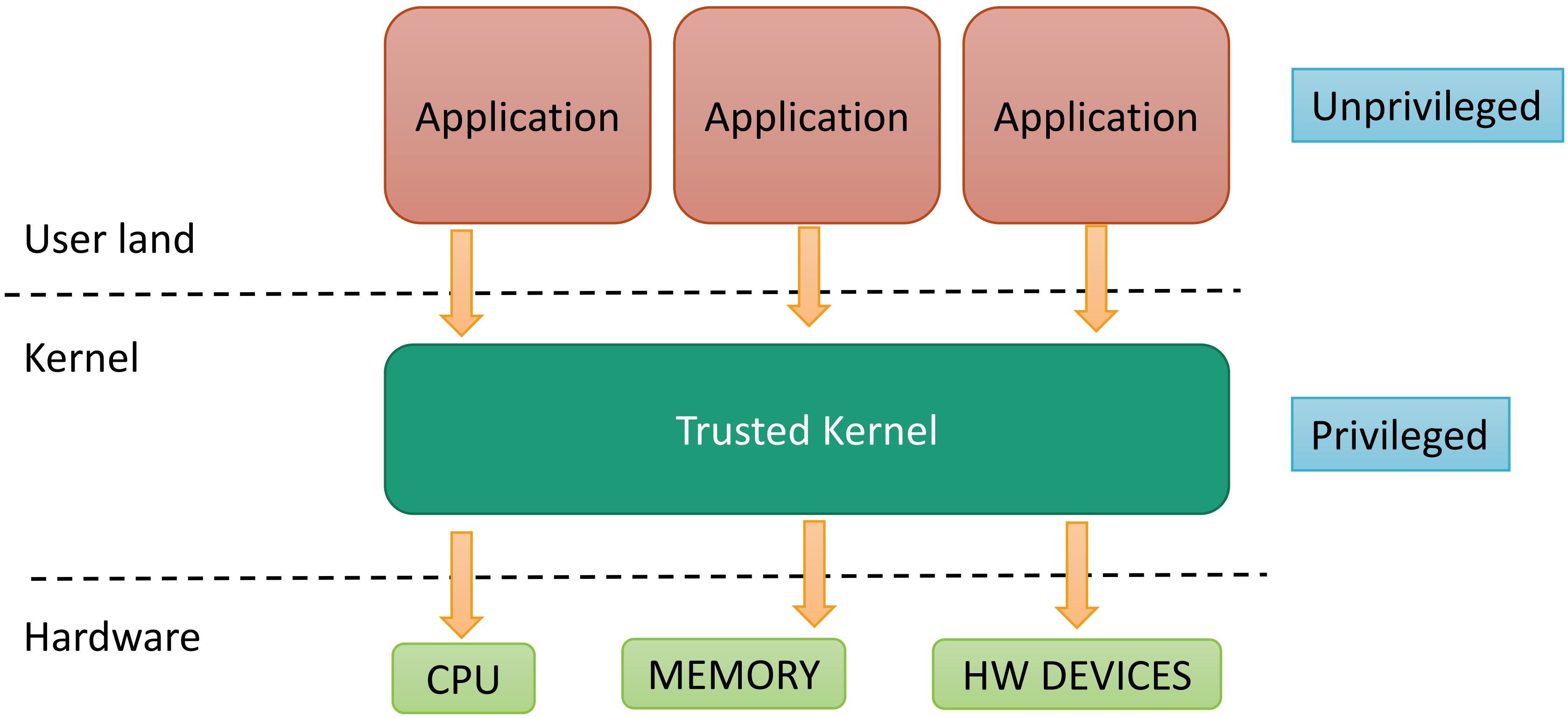
User space cannot access HW devices



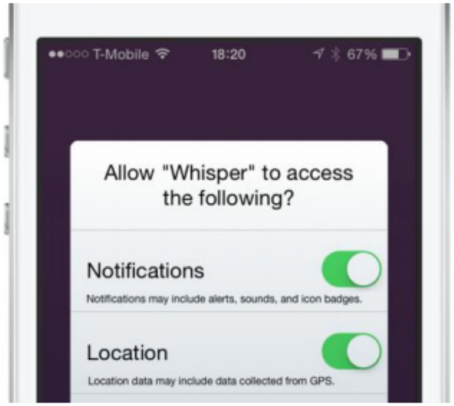
OS Access Control of HW

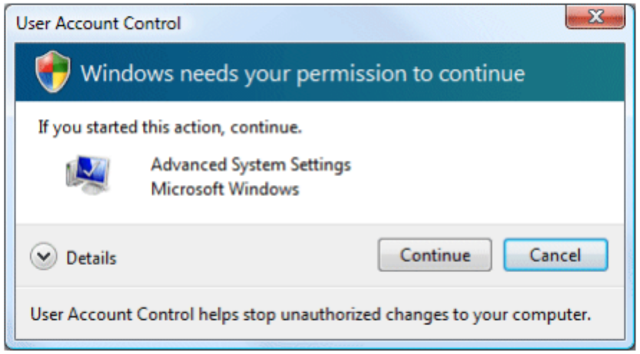


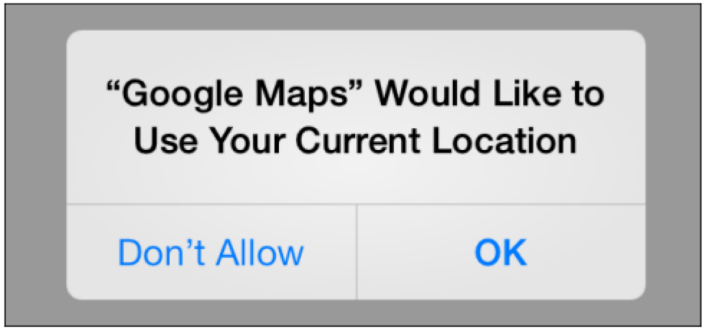
OS Access Control of HW



Example of OS-Level Access Control to HW

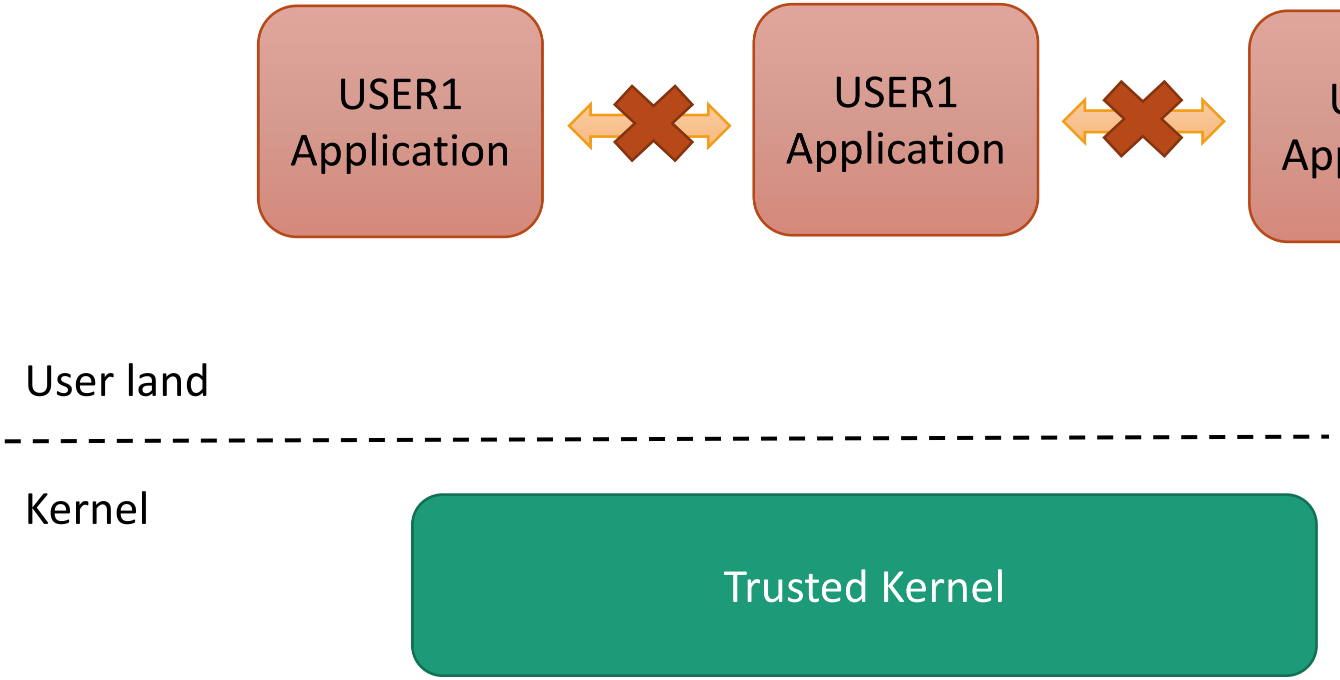






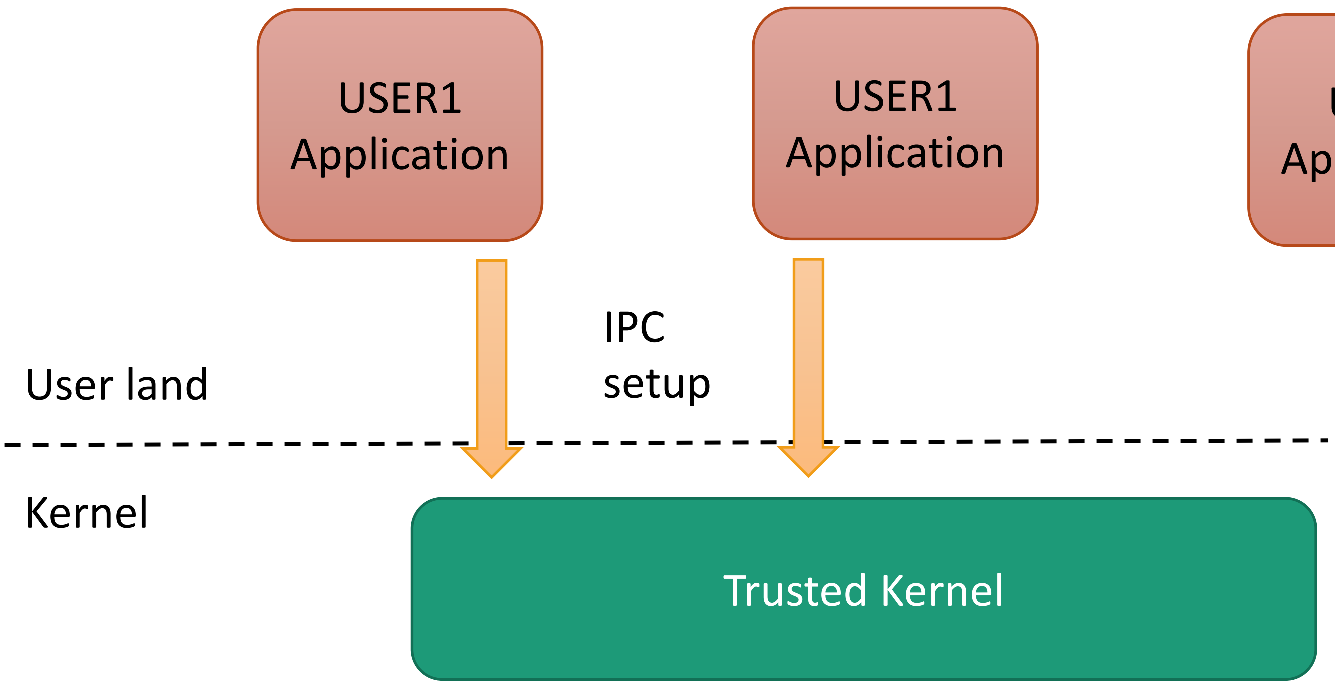
Process-level Isolation

Processes cannot directly access each other's state



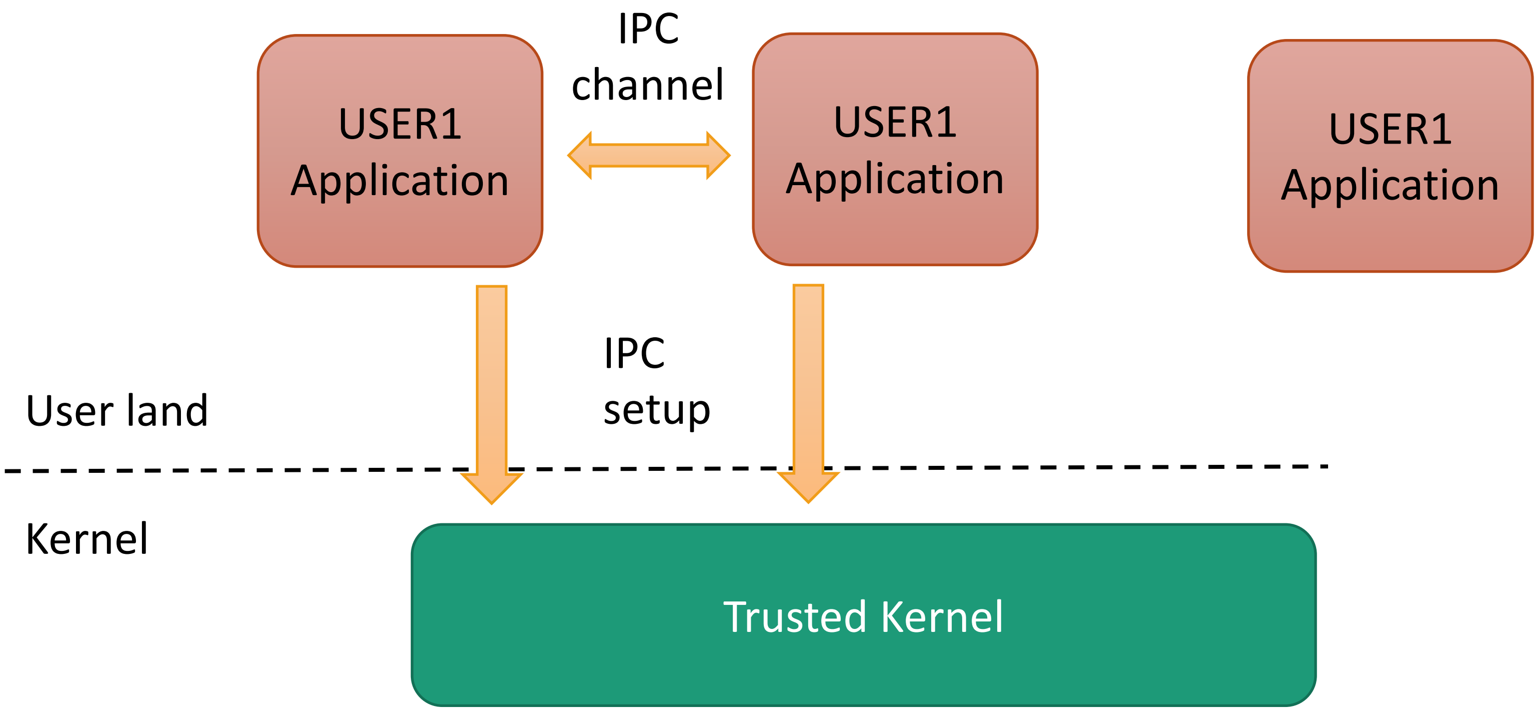
Process-level Isolation

The kernel can setup inter-process communication



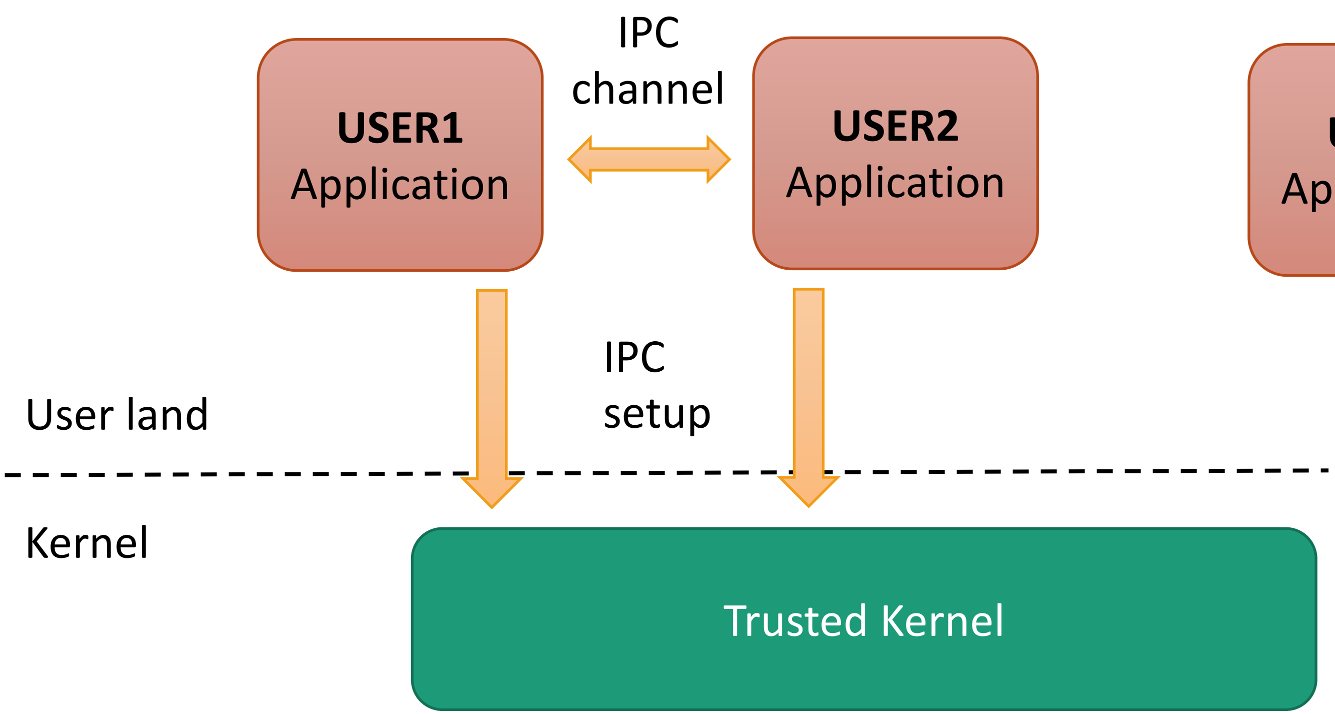
Process-level Isolation

The kernel can setup inter-process communication



Process-level Isolation

Same for processes owned by different users



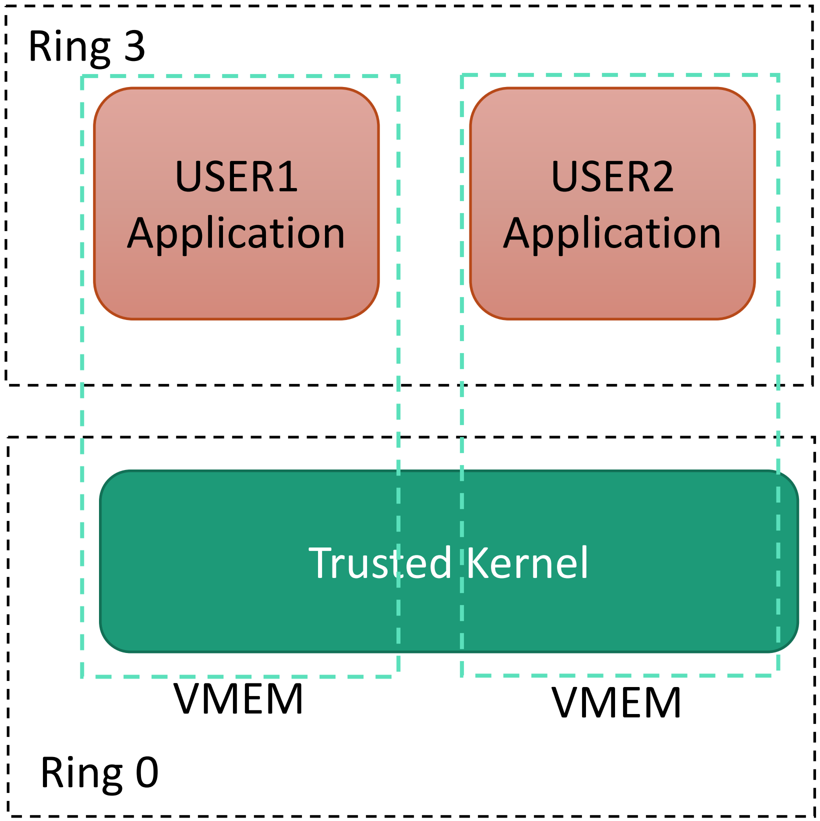
Hardware-based Enforcement

The memory-management unit (MMU) provides virtual memory

Execution rings separate user and kernel space

- Indicated by bits in CPU status register

Processes are isolated into different virtual memory address spaces



Sandboxing Methods

VM-based
- Run entire OS in isolation

OS-based
- Process-wide
- Available system calls and capabilities are restricted

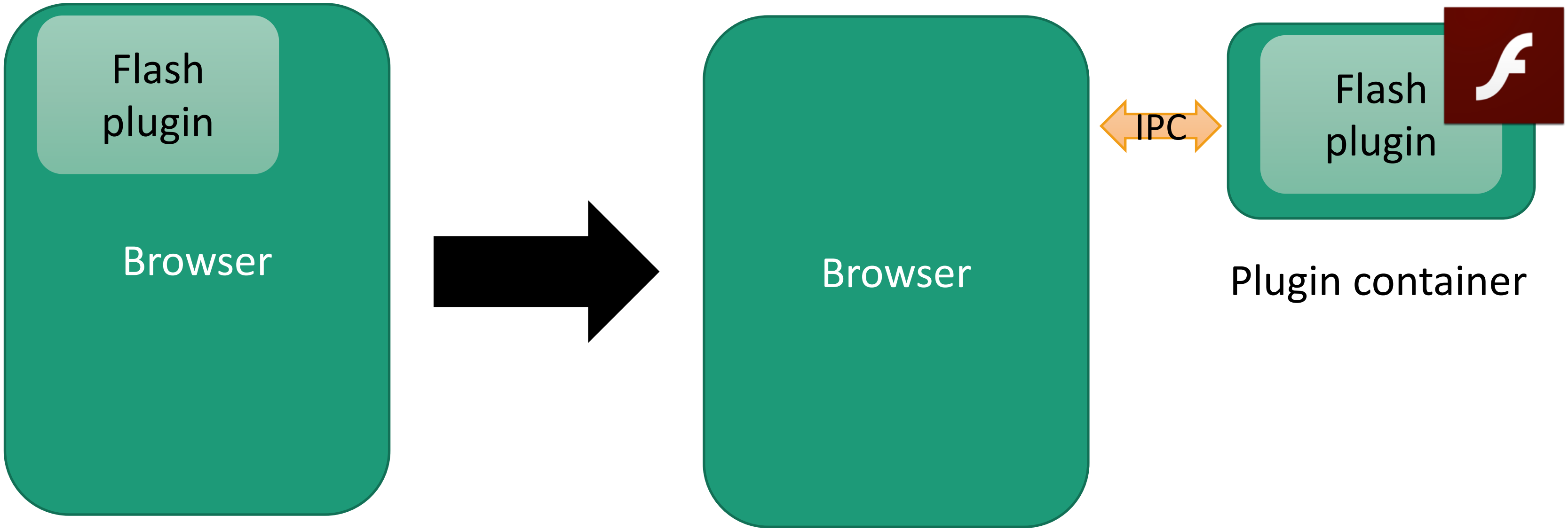
Language-based
- Language isolates components

Inline reference monitor
- Integrated into untrusted code during compilation, code generation, or through emulation
- Security checks injected to enforce policy

Building on Process Isolation

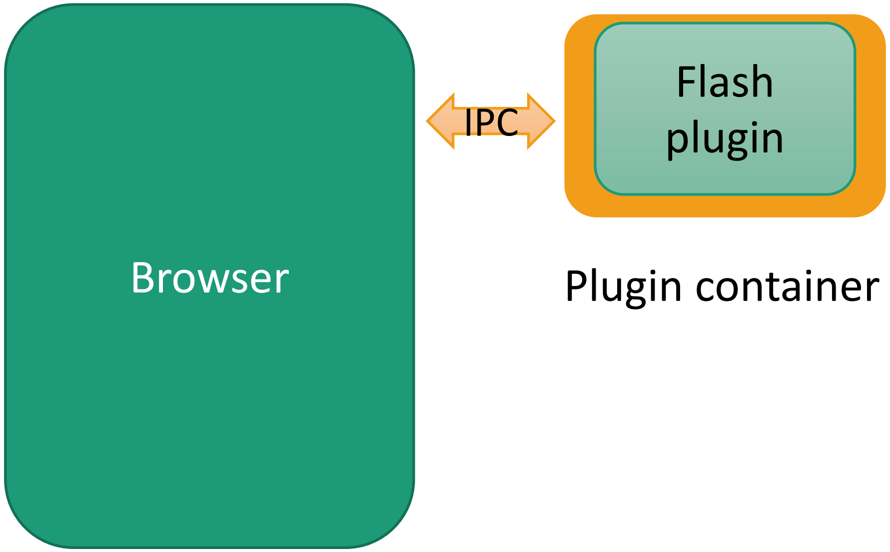
Run code in its own process space to isolate it from browser process

Congratulations you have just executed untrusted code from the Internet!



Building on Process Isolation

Container must have limited privileges



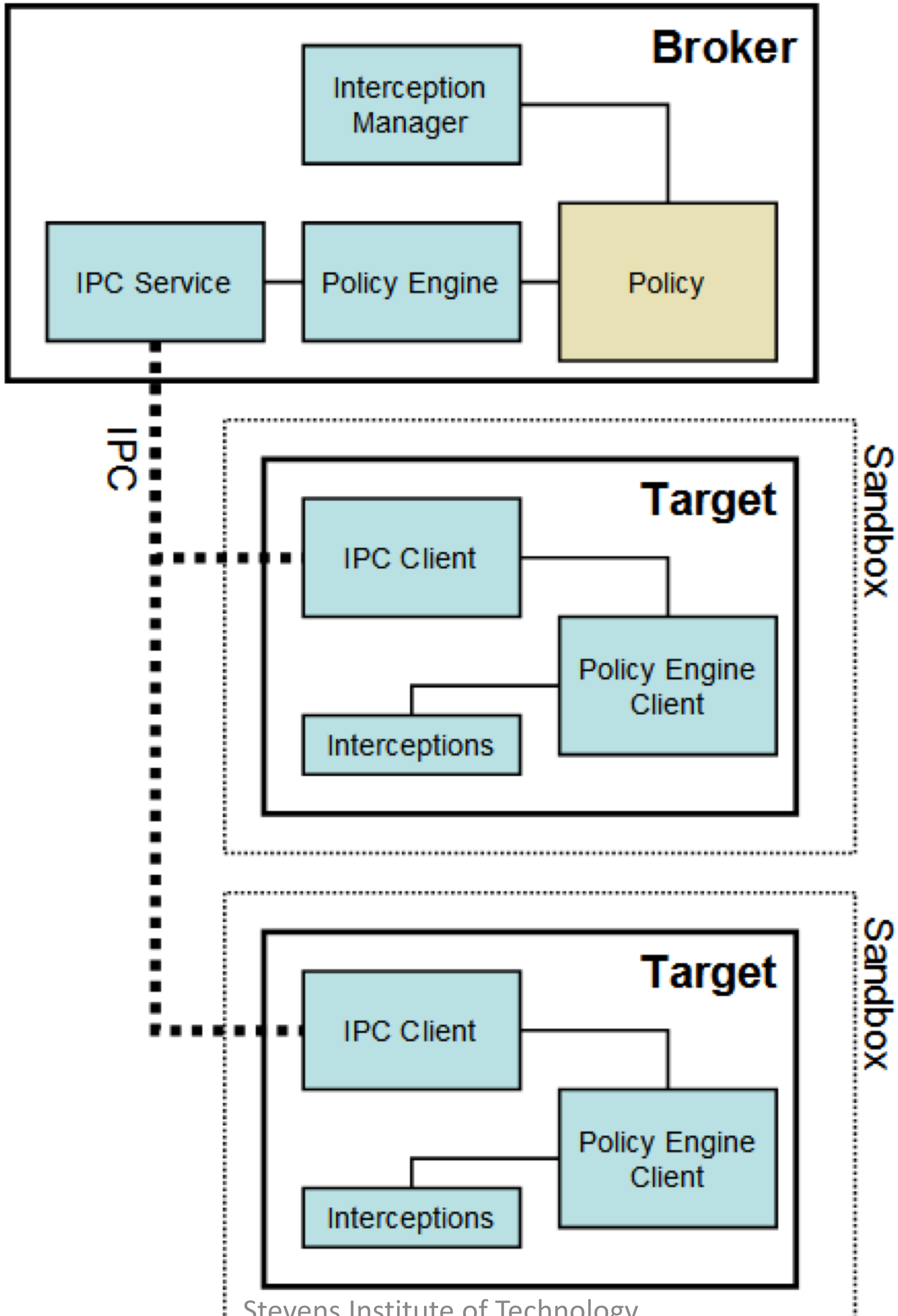
Chromium Sandboxing in Linux

Chromium runs plugins and the rendering engine for each tab in a separate process

Rendering processes are sandboxed

Sandboxed processes are managed by a broker process over IPC

https://chromium.googlesource.com/chromium/src/+/master/docs/linux_sandboxing.md


Broker
Interception Manager
IPC Service
Policy Engine
Policy
IPC
Target
IPC Client
Policy Engine Client
Interceptions
Sandbox
Target
IPC Client
Policy Engine Client
Interceptions
Sandbox

Process Sandbox: SUID

A helper binary with the setuid bit set is used

The SUID bit causes the execution of the process as root

- Enables access to privileged kernel APIs, such as namespaces

chroot() is used to change the process' root directory

- Take away file system access from the process

Process is placed in new PID namespace

- Process cannot terminate or signal processes outside the namespace

Process is placed in new network namespace

- Restrict network access of process

Finally drop super-user privileges

Process Sandbox: User Namespaces

User namespaces are an unprivileged API

Used as an alternative to SUID sandbox

A process is placed a new namespace

Isolates:

- Filesystem
- Network
- PID
- IPC

User Namespaces

A newly launched process can be put in a new namespace

- Through the clone() system call

Available namespaces

Namespace	Constant	Isolates
Cgroup	CLONE_NEWCGROUP	Cgroup root directory
IPC	CLONE_NEWIPC	System V IPC, POSIX message queues
Network	CLONE_NEWNET	Network devices, stacks, ports, etc.
Mount	CLONE_NEWNS	Mount points
PID	CLONE_NEWPID	Process IDs
User	CLONE_NEWUSER	User and group IDs
UTS	CLONE_NEWUTS	Hostname and NIS domain name

Reading material: https://lwn.net/Articles/531114/

Process Sandbox: SECCOMP BPF

Filters the kernel APIs available to a process

Used together with previous sandboxes

Aims to protect the kernel from a malicious process

Available system calls are defined using **Berkeley packet filters**

- Filters are compiled to a program that enforces policy

SECCOMP BPF Programs

Programs consist of instructions that can check the values of various system calls and their arguments

- Cannot dereference pointers

BPF can be hard to write and looks like assembly

Example:

- a BPF load operation (BPF_LD), for a word (BPF_W), using the value in the instruction as an offset into the data area (BPF_ABS)

```
BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct seccomp_data, arch)))
```

- a jump-if-equal instruction (BPF_JMP | BPF JEQ) that compares the value in the instruction, which is known as "k", (BPF_K) to the value in the accumulator. So, if the architecture is x86-64, this jump will skip the next instruction (the offset of "1" for the jump true destination), otherwise it will execute it ("0" for jump false)

```
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K , AUDIT_ARCH_X86_64 , 1, 0)
```

SECCOMP BPF Programs

What you can do

Filter specific system calls

- Or particular arguments to them

Define what action to take when an invalid call is made

What you can't do

Dereference pointer arguments

Remove an installed filter

Libsecomp

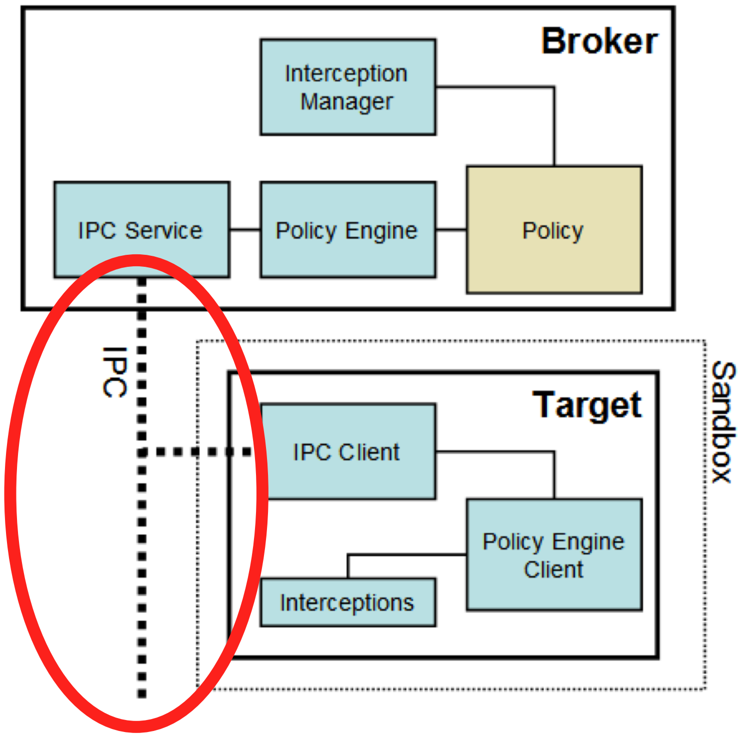
Offers a simpler API for writing filters

```
int seccomp_rule_add(uint32_t action, int syscall, unsigned int arg_cnt, ...);

seccomp_rule_add(SCMP_ACT_ALLOW, SCMP_SYS(close), 0);
```

```c
#include <stdio.h>   /* printf */
#include <unistd.h>  /* dup2: just for test */
#include <seccomp.h> /* libseccomp */

int main() {
  printf("step 1: unrestricted\n");

  // Init the filter
  scmp_filter_ctx ctx;
  ctx = seccomp_init(SCMP_ACT_KILL); // default action: kill

  // setup basic whitelist
  seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(rt_sigreturn), 0);
  seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit), 0);
  seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(read), 0);
  seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(write), 0);

  // setup our rule
  seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(dup2), 2,
                        SCMP_A0(SCMP_CMP_EQ, 1),
                        SCMP_A1(SCMP_CMP_EQ, 2));

  // build and load the filter
  seccomp_load(ctx);
  printf("step 2: only 'write' and dup2(1, 2) syscalls\n");

  // Redirect stderr to stdout
  dup2(1, 2);
  printf("step 3: stderr redirected to stdout\n");

  // Duplicate stderr to arbitrary fd
  dup2(2, 42);
  printf("step 4: !! YOU SHOULD NOT SEE ME !!\n");

  // Success (well, not so in this case...)
  return 0;
}
```

Limitations of OS and VM-based Sandboxing

Context switches between broker and sandboxed processes can be expensive



Sandboxing Methods

VM-based

- Run entire OS in isolation

OS-based

- Process-wide
- Available system calls and capabilities are restricted

Language-based

- Language isolates components

Inline reference monitor

- Integrated into untrusted code during compilation, code generation, or through emulation
- Security checks injected to enforce policy

Example: JS/Java

The language and the runtime environment/VM is enforcing security

- Memory safe languages
- Memory corruption or leakage is not possible (at least in theory)

Access control done at the API level, for example:

- Which files can be loaded
- Which frames are accessible through the DOM
- Where can code be loaded from
- **The VM acts as a reference monitor**

Sandboxing Methods

VM-based

- Run entire OS in isolation

OS-based

- Process-wide
- Available system calls and capabilities are restricted

Language-based

- Language isolates components

Inline reference monitor

- Integrated into untrusted code during compilation, code generation, or through emulation
- Security checks injected to enforce policy

Sandboxing Unsafe Languages

Pointers can be used to potential read/write arbitrary memory

Memory accesses need to be isolated first

- Can rarely rely on HW to contain memory operations
- Software checks are introduced in application code

Software-fault Isolation

Run multiple programs in the same address space that run in isolation

Each program runs in a different logical fault domain

Programs can access memory within their domain

- Ensures memory secrecy and integrity

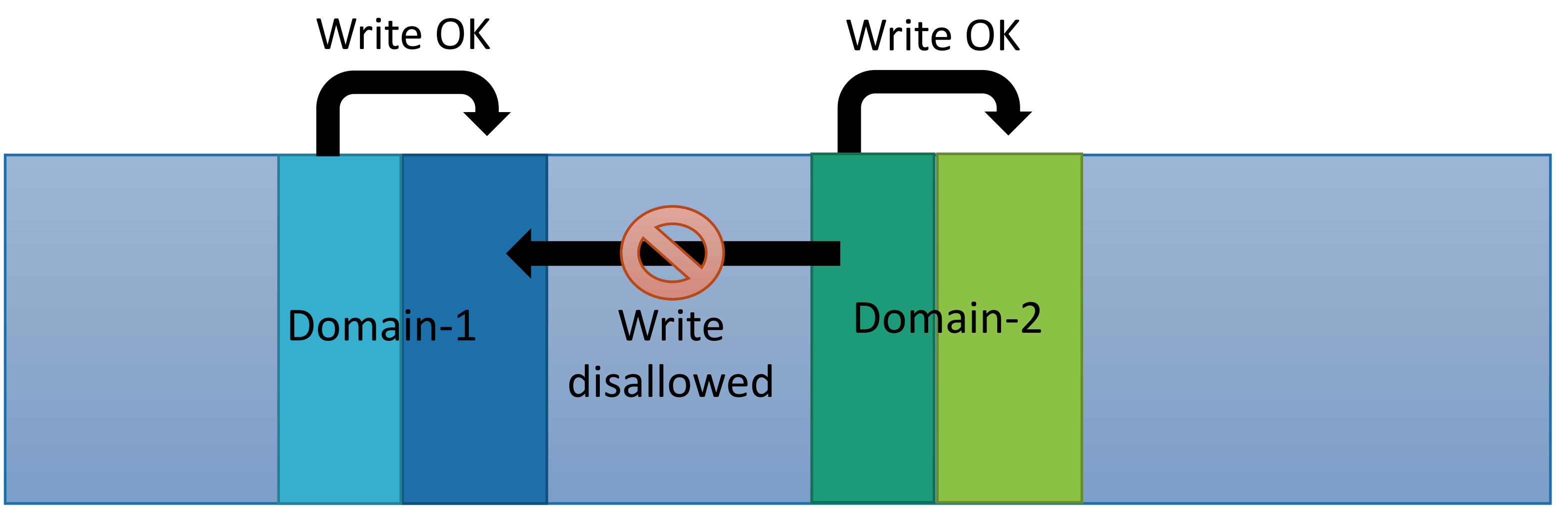
Code within a domain cannot call/jump to code in other domains

- Unless through secure interfaces

Software-fault Isolation

Programs can only access memory within their domain

- **Ensures memory secrecy and integrity**



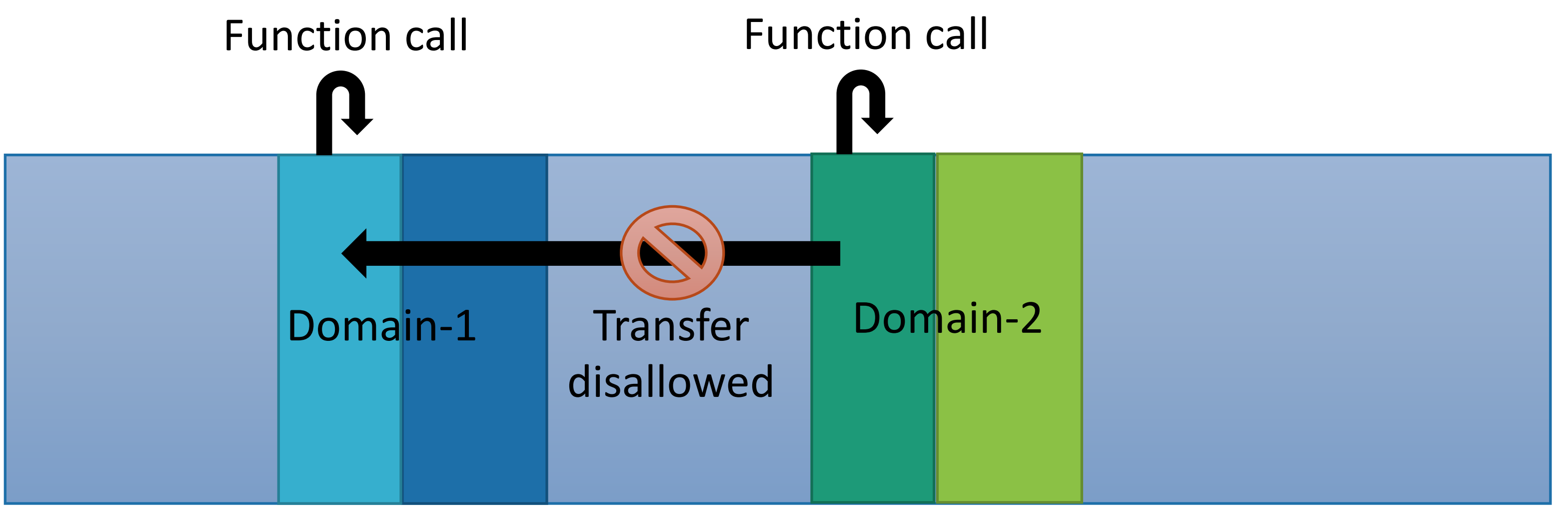
Software-fault Isolation

Programs can only access memory within their domain

- Ensures memory secrecy and integrity

Code within a domain cannot call/jump to code in other domains

- **Unless through secure interfaces**



Software-fault Isolation

Programs can only access memory within their domain

- Ensures memory secrecy and integrity

Code within a domain cannot call/jump to code in other domains

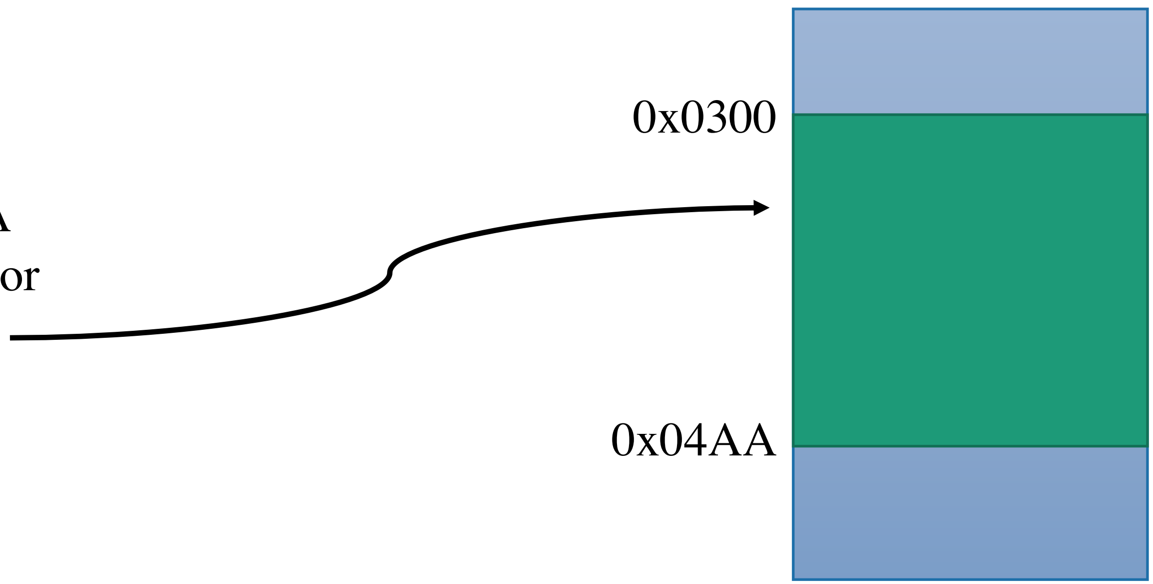
- Unless through secure interfaces

Modify programs during compilation or by rewriting to enforce these properties

Constraining Memory Accesses

Through boundary checking

cmp 0x0300
if less Error
cmp 0x04AA
if greater Error
write x

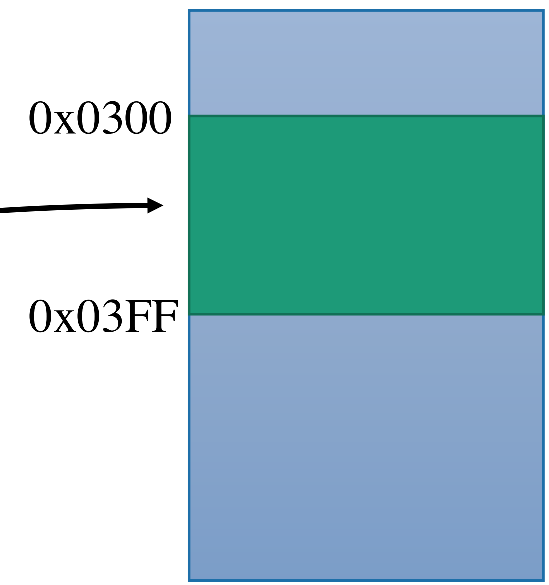


Constraining Memory Accesses

We can improve the boundary checks

- By allocating domains in aligned memory ranges
- Using bit masking to help with checking

```
tmp := x & FF00
cmp tmp 0300
if not equal Error
write x
```

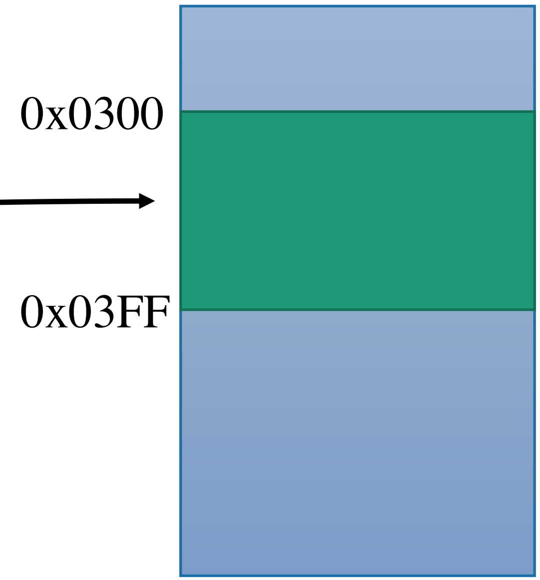


Constraining Memory Accesses

Further improvements

- Do not detect error
- Constrain memory access to domain

tmp : = x & 00FF
tmp : = tmp | 0300
write tmp

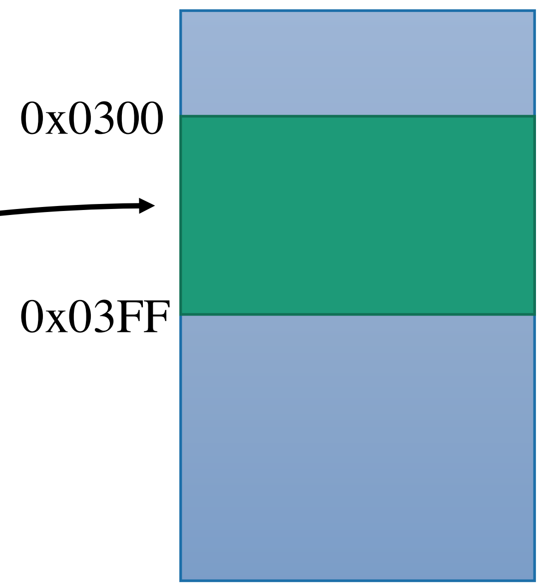


Constraining Memory Accesses

Eliminating temporary registers is not always a good idea

Malicious code could bypass masking operations

```
…
x : = x & 00FF
x : = x | 0300
write x
```

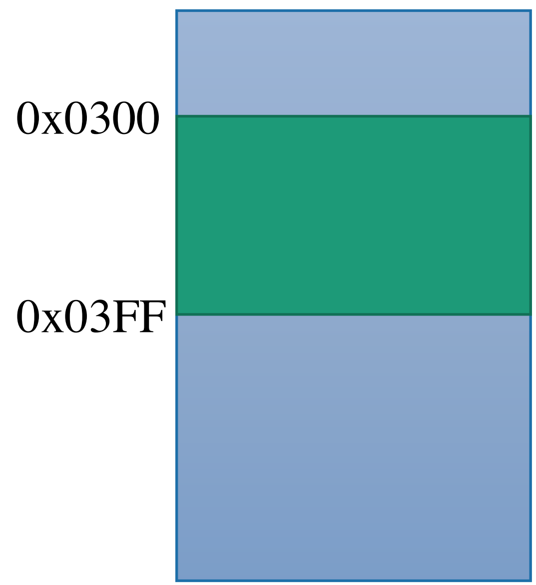


Time Of Check Time Of Use (TOCTOU)

Constraining Memory Accesses

Can malicious code bypass checks with temporary registers?

tmp : = x & 00FF
tmp : = tmp | 0300
write tmp

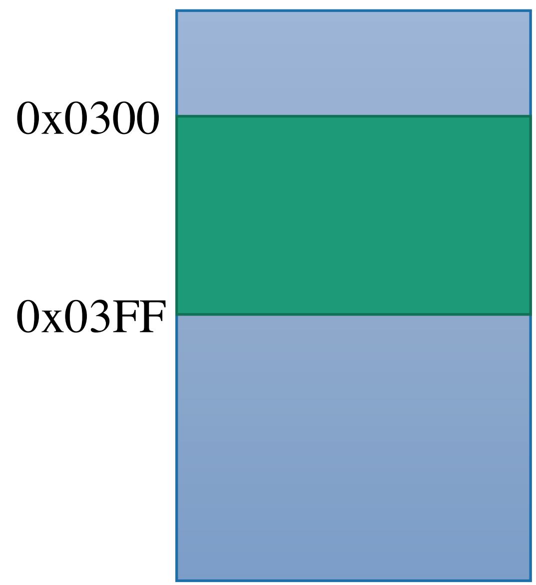


Constraining Memory Accesses

Can malicious code bypass checks with temporary registers?

tmp : = x & 00FF
tmp : = tmp | 0300
write tmp

?



- tmp has not been initialized and will probably cause the program to crash.
- Can be forced to crash by setting tmp to bad address (e.g., 0xffffffff) after write

Constraining Control Flow

Sandboxes are mainly to used to constrain untrusted code so obviously this is a general problem

…
jmp ptr ?

Constraining Control Flow

Similar tricks can be applied

… **jmp ptr**	→	… tptr : = ptr & 00FF tptr : = tptr | 0300 **jmp tptr**
… **call ptr**	→	… tptr : = ptr & 00FF tptr : = tptr | 0300 **call tptr**
… **ret**	→	**?**

Constraining Control Flow

Naive approach

ret →

pop tptr
tptr : = tptr & 00FF
tptr : = tptr | 0300
jmp ptr

CISC Trouble

Constraining within the domain is not enough

- Instructions may be hidden within instructions in CISC programs

Pseudo Fixed-size Instructions

Align every “pseudo” instruction on a 32-byte boundary

- 0x1F bits are always zero

Force pointer so it can only point to a pseudo instruction

```
pop tptr
tptr : = tptr & 00E0
tptr : = tptr | 0300
jmp ptr
```

Benefits of SFI

No context switches

Faster if run-time checks are faster than context switching

Google Native Client (NaCL)

A sandboxing technology for running a subset of Intel x86, ARM, or MIPS **native** code in a sandbox

https://developer.chrome.com/native-client

NaCL programs are compiled with modified compiler

Supports subset of language

Produces sandboxed programs

Escaping Sandboxes

Exploitation of a sandboxed component grants limited control

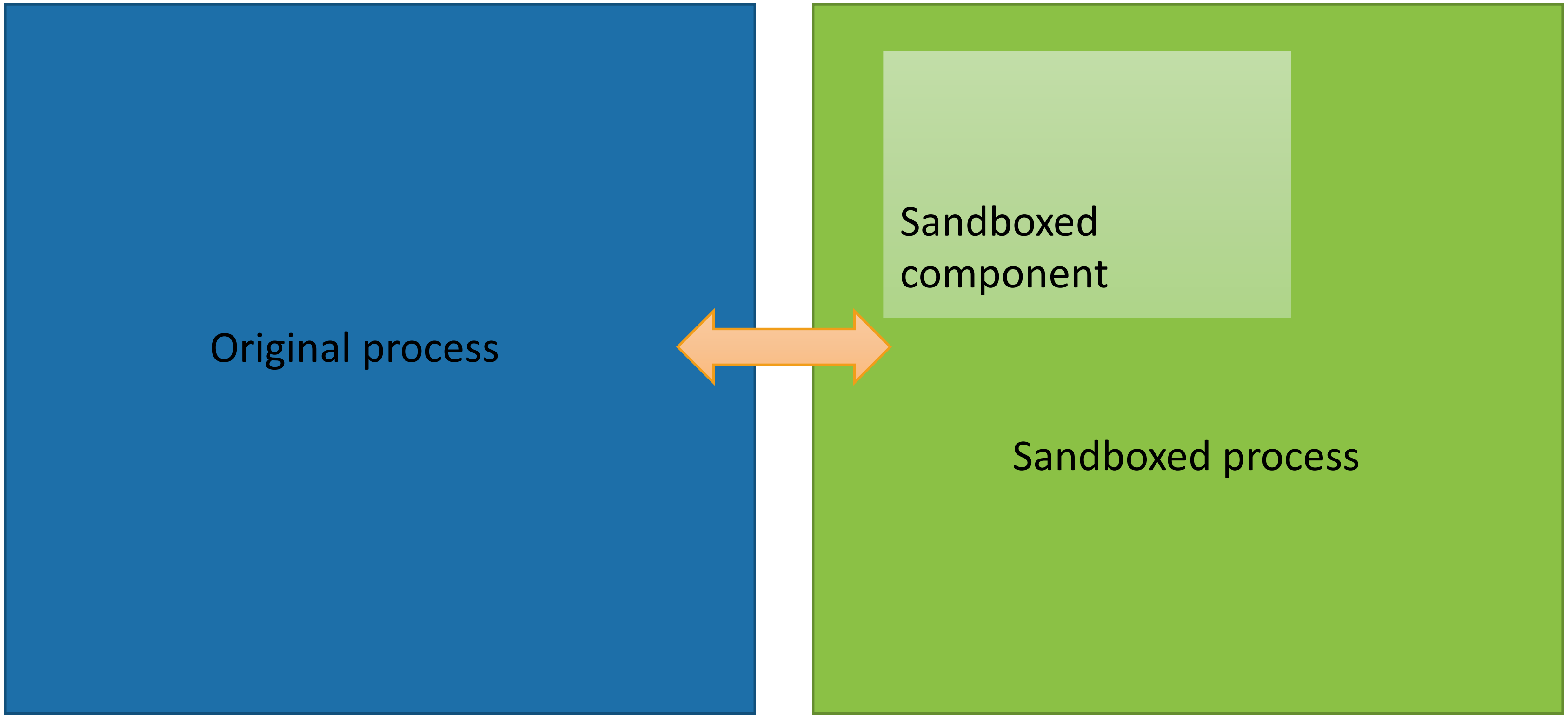
But sandboxes may have bugs

Multiple exploits in different components are usually required

In 2012's pwnium competition 14 bugs where needed to take down chrome

- http://blog.chromium.org/2012/05/tale-of-two-pwnies-part-1.html

Multiple Layers of Sandboxes

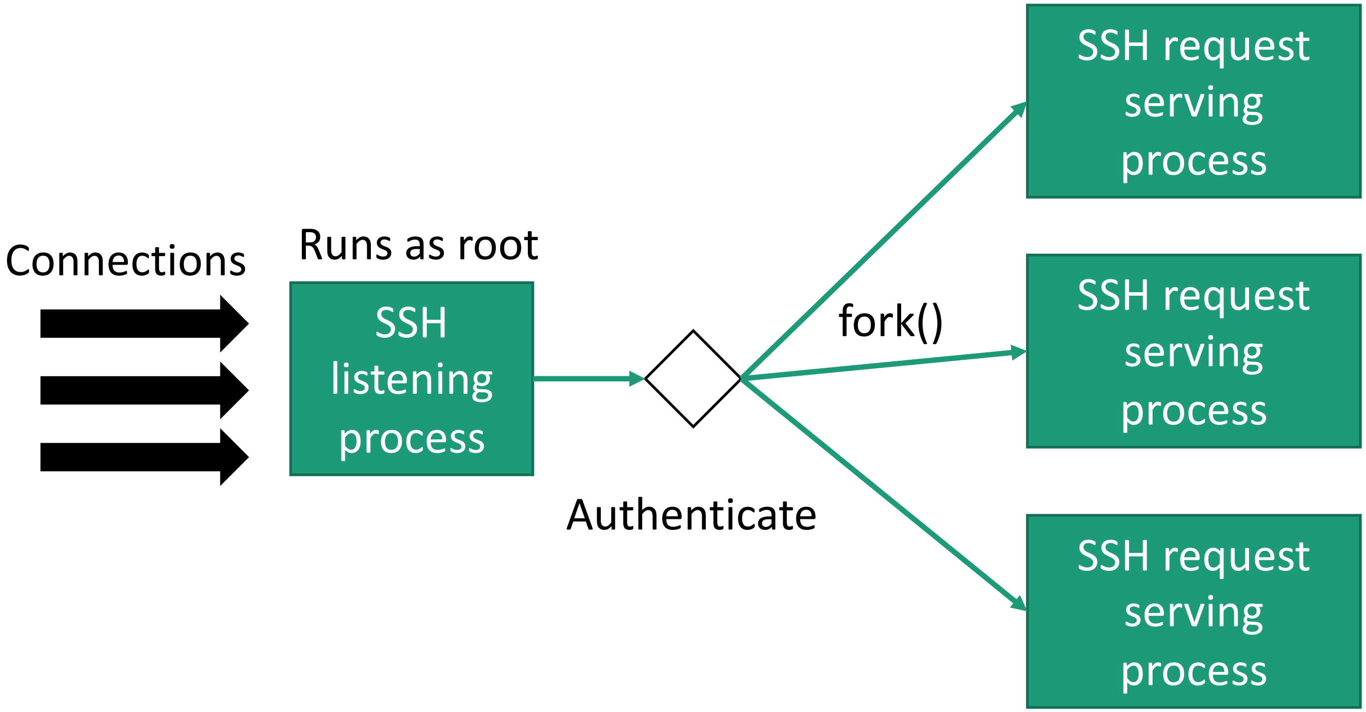


Other Use Cases for Isolation

Process-level Isolation from the OS is frequently used to realize the principle of least privilege in servers

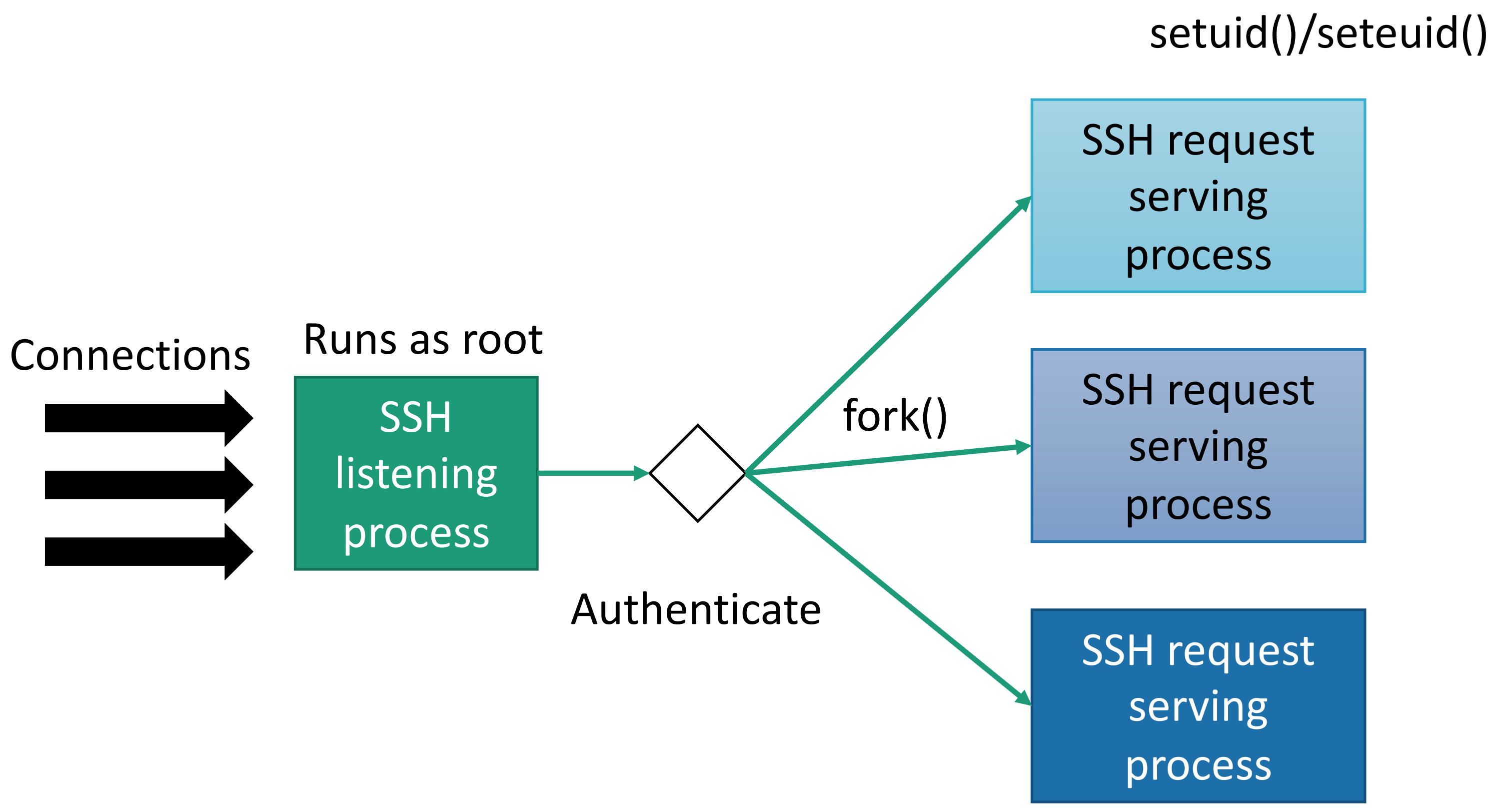
Examples: SSH, Web servers

SSH



How is access control done here?

SSH



Process drop privileges and run as the authenticated user