Confinement
(Running Untrusted Programs)

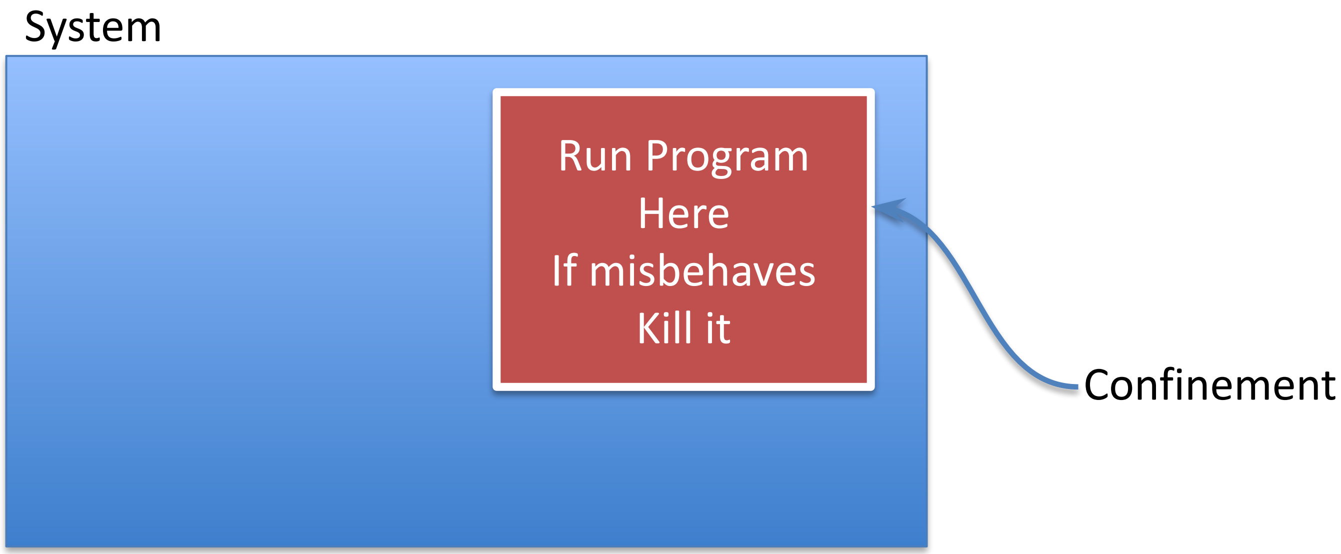
Chester Rebeiro

Indian Institute of Technology Madras

Untrusted Programs

- How to run untrusted programs and not harm your system?

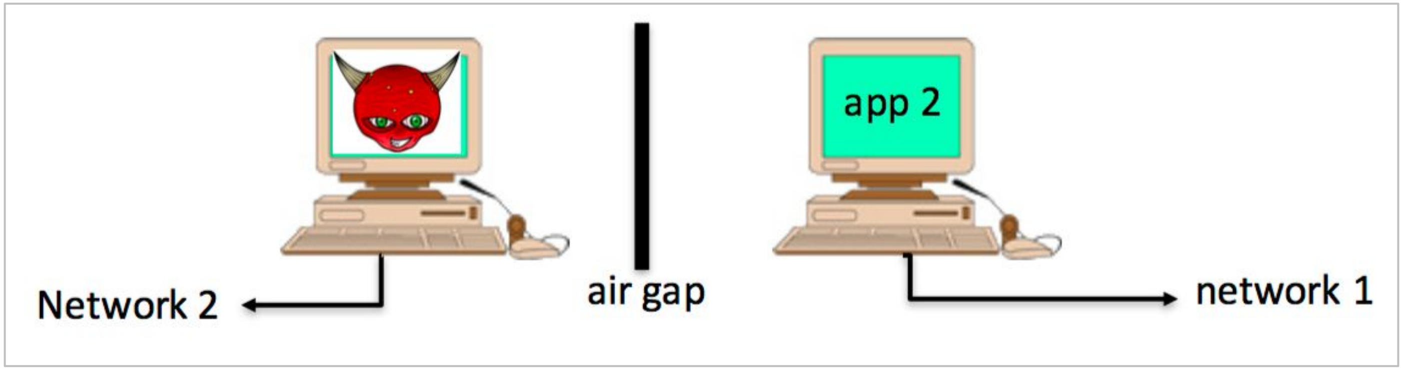
  Answer: Confinement (sometimes called sandbox)



Confinement

- Can be implemented at several levels

Air Gap: Run untrusted app in an isolated hardware



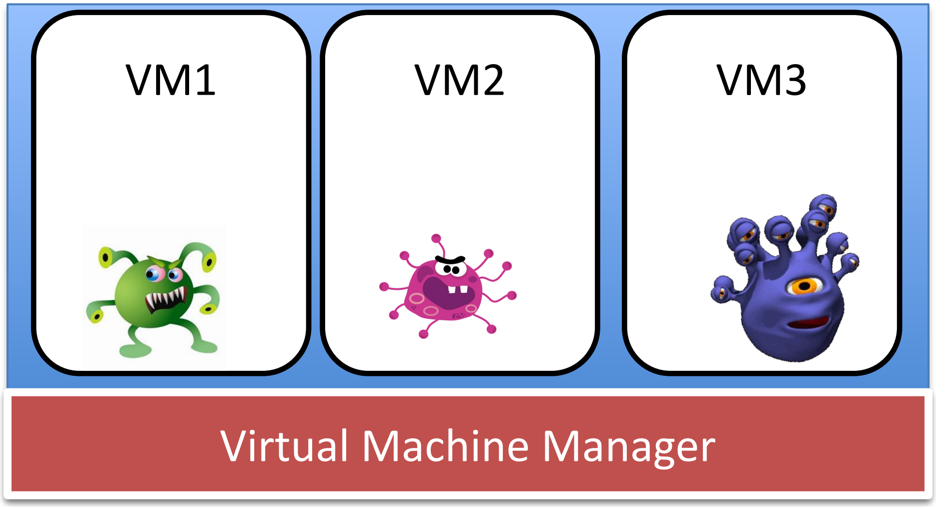
Not easy in practice but (almost) fool proof

Very coarse granularity

Confinement

- Can be implemented at several levels

Virtual Machines: Run untrusted app in a different VM

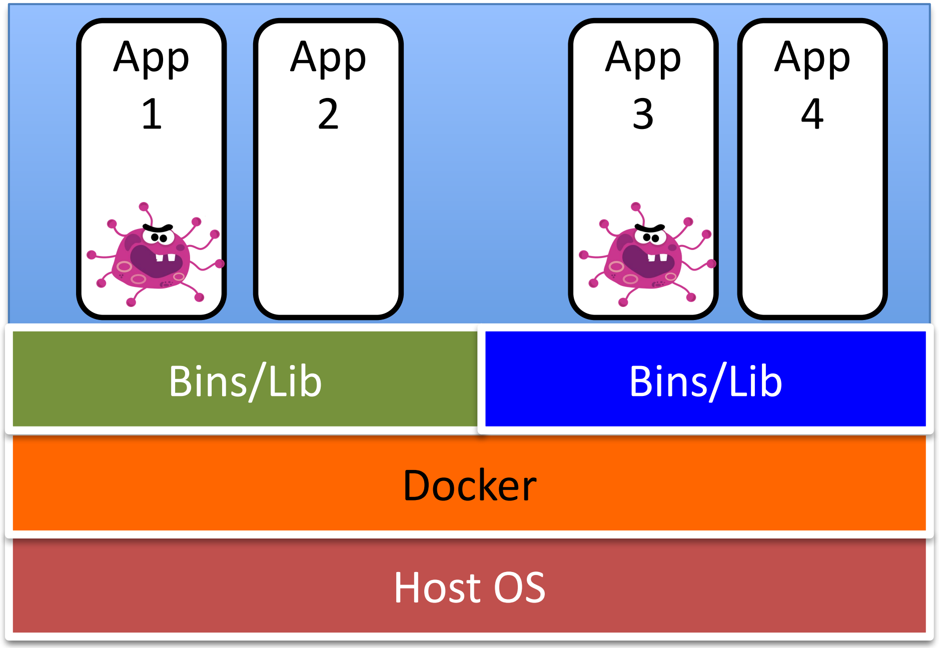


coarse granularity

Confinement

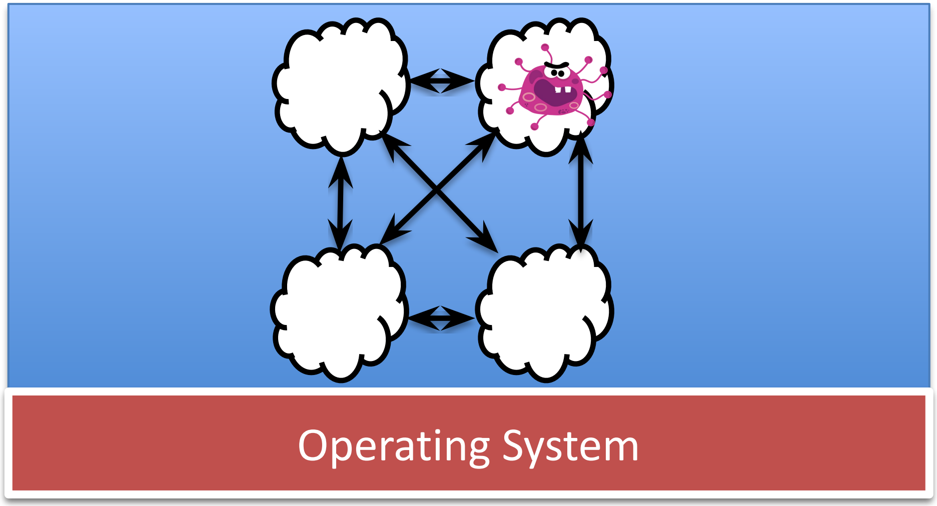
- Can be implemented at several levels

Containers: Run untrusted apps in different containers



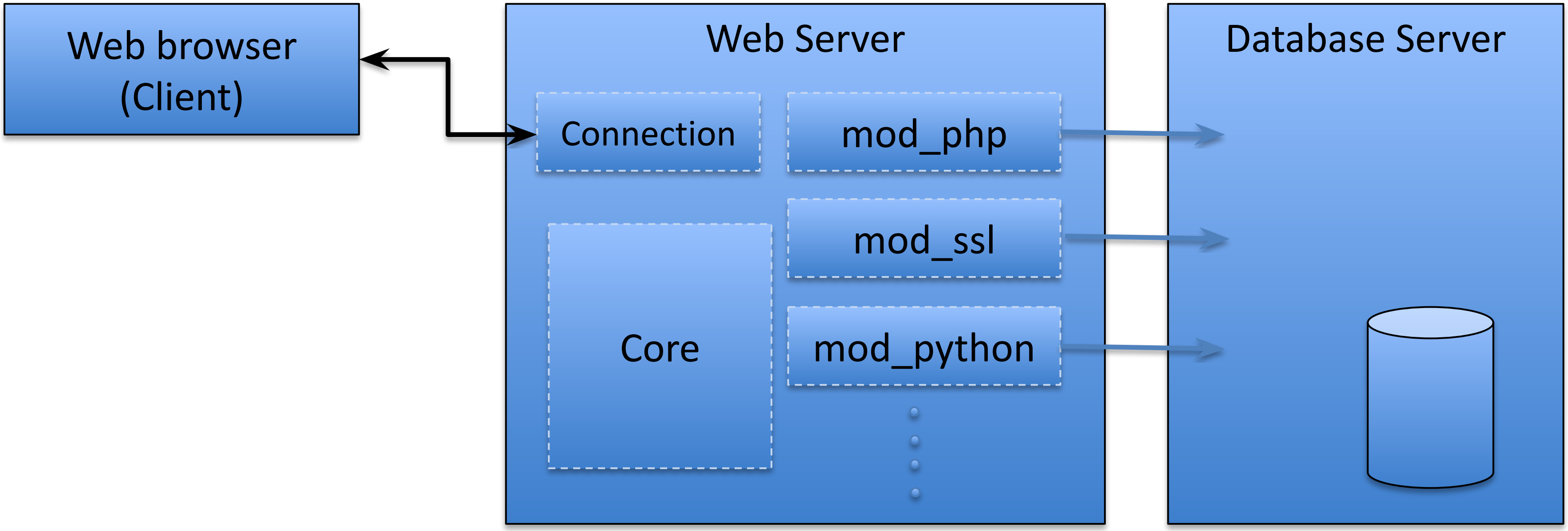
Confinement within a process (using RPCs)

- Run each module as a different process (different address spaces)
  - Use RPCs to communicate between modules
  - Hardware ensures that one process does not affect another



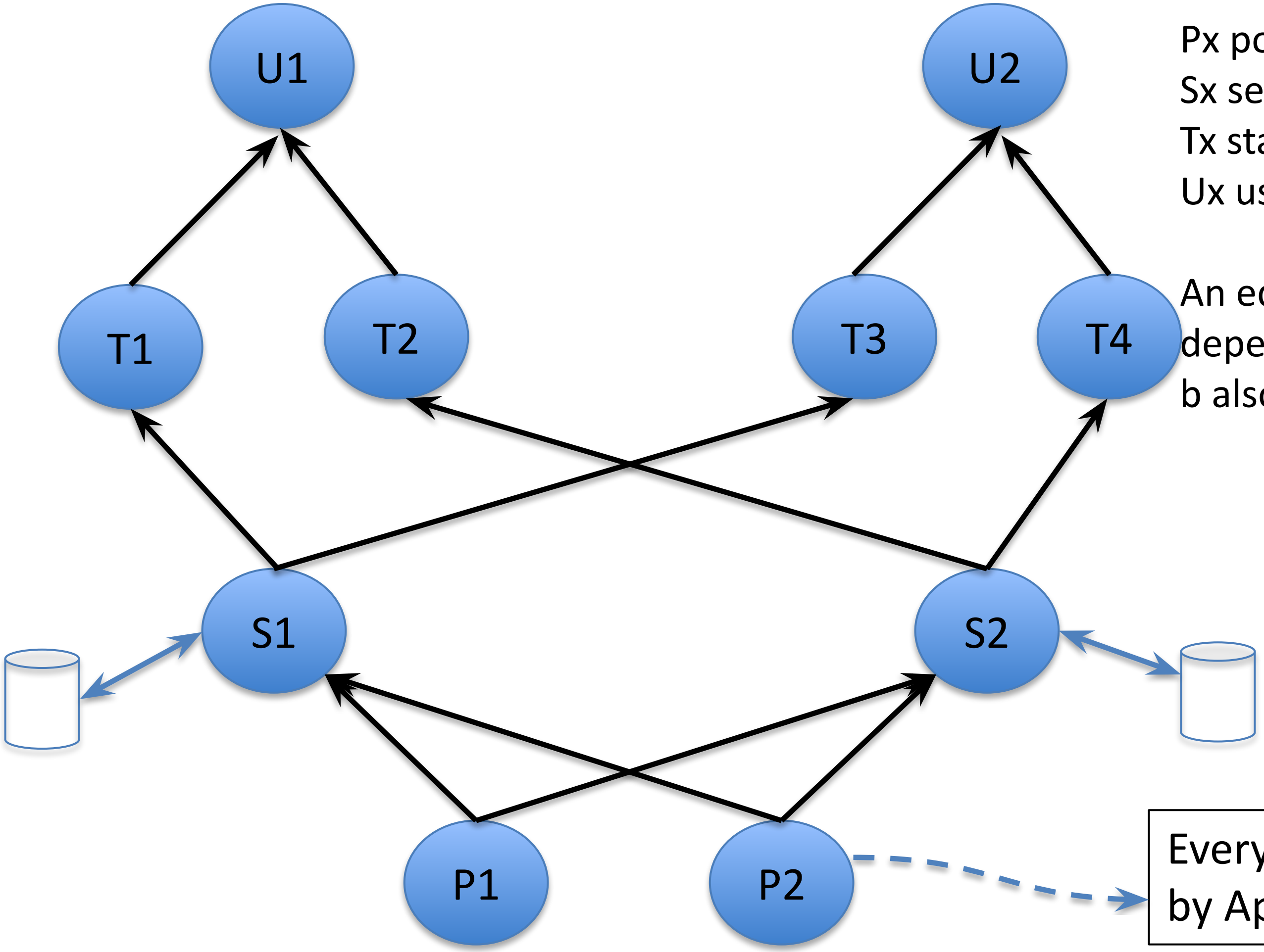
Case Study: OKWS Web Server

A typical webserver architecture



- Logical separate modules present in a single address space
- A pool of processes created at startup
- Weak security

Apache Webserver (Dependency Graph)

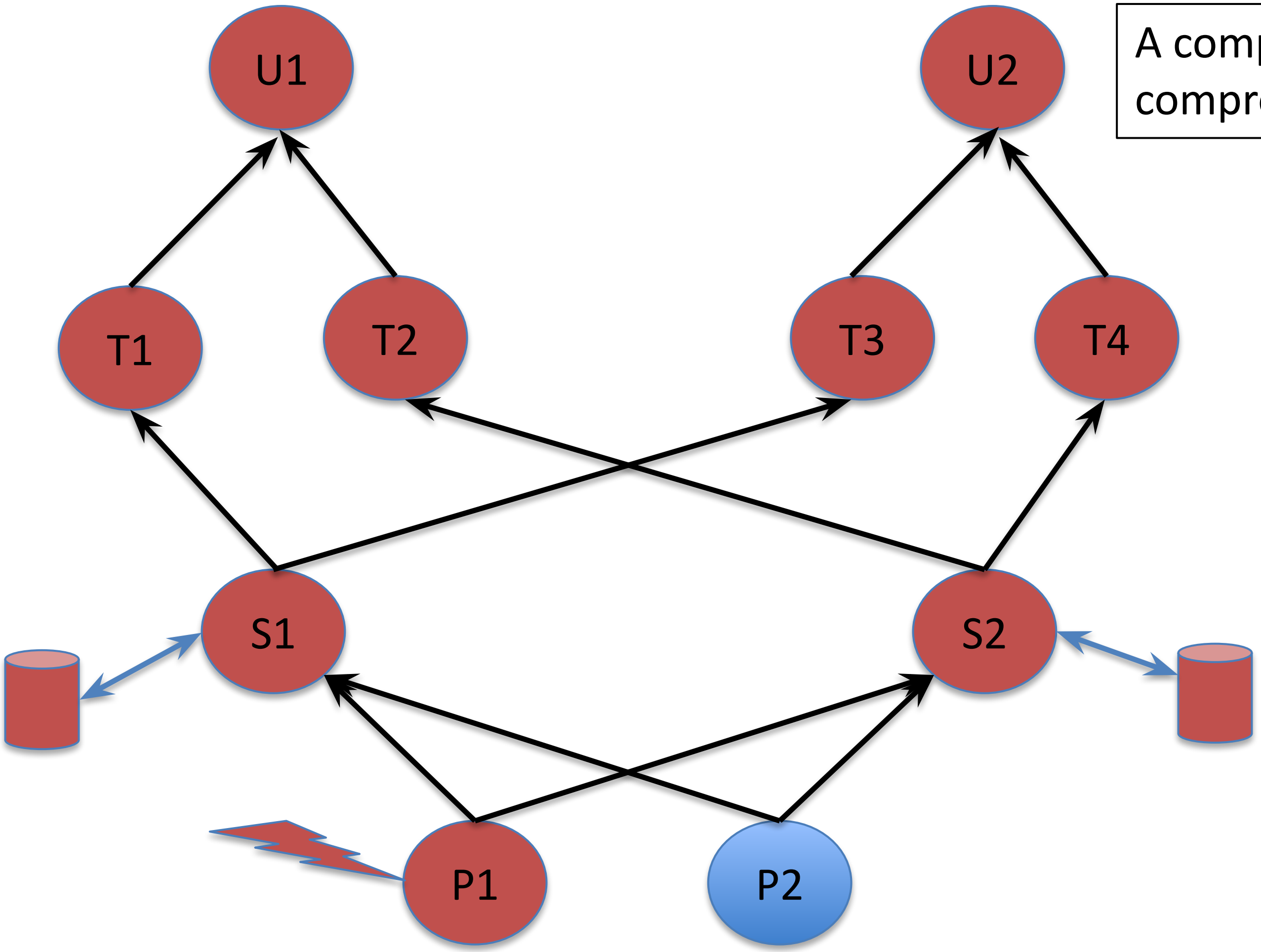


Px pool of processes
Sx services access private databases
Tx state data
Ux users

An edge from (a, b) implies b’s depenence on a. If a gets compromised b also will be compromised.

Every child process created by Apache, Includes all services

A compromised process
(Apache Webserver)



A compromised child process will compromise all services

Known attacks on Web Servers

- A bug in one website can lead to an attack in another website
  example: Amazon holds credit card numbers. If it happens to share the same web server as other users this could lead to trouble.

- Some known attacks on Apache's webserver and its standard modules
  - Unintended data disclosure (2002)
    users get access to sensitive log information
  - Buffer overflows and remote code execution (2002)
  - Denial of service attacks (2003)
  - Due to scripting extensions to Apache

Principle of Least Privileges

- Decompose system into subsystems
- Grant privileges in fine grained manner
- Minimal access given to subsystems to access system data and resources
- Narrow interfaces between subsystems that only allow necessary operations
- Assume exploit more likely to occur in subsystems closer to the user (eg. network interfaces)
- Security enforcement done outside the system (eg. by OS)

Achieving Confinement

Through Unix Tools

- **chroot:** define the file system a process can see
- **setuid:** set the uid of a process to confine what it can do
- **Passing file descriptors:** a privileged parent process can open a file and pass the descriptor to an unprivileged child

OKWS Webserver (designed for least privileges)

each independent service runs in an independent process

Do not expose more code/services than required! Tradeoff security vs performance

Each service should run in a separate chroot

Allow access to only necessary files.

Each process should run as an unprivileged user.

Prevent interfering with other processes

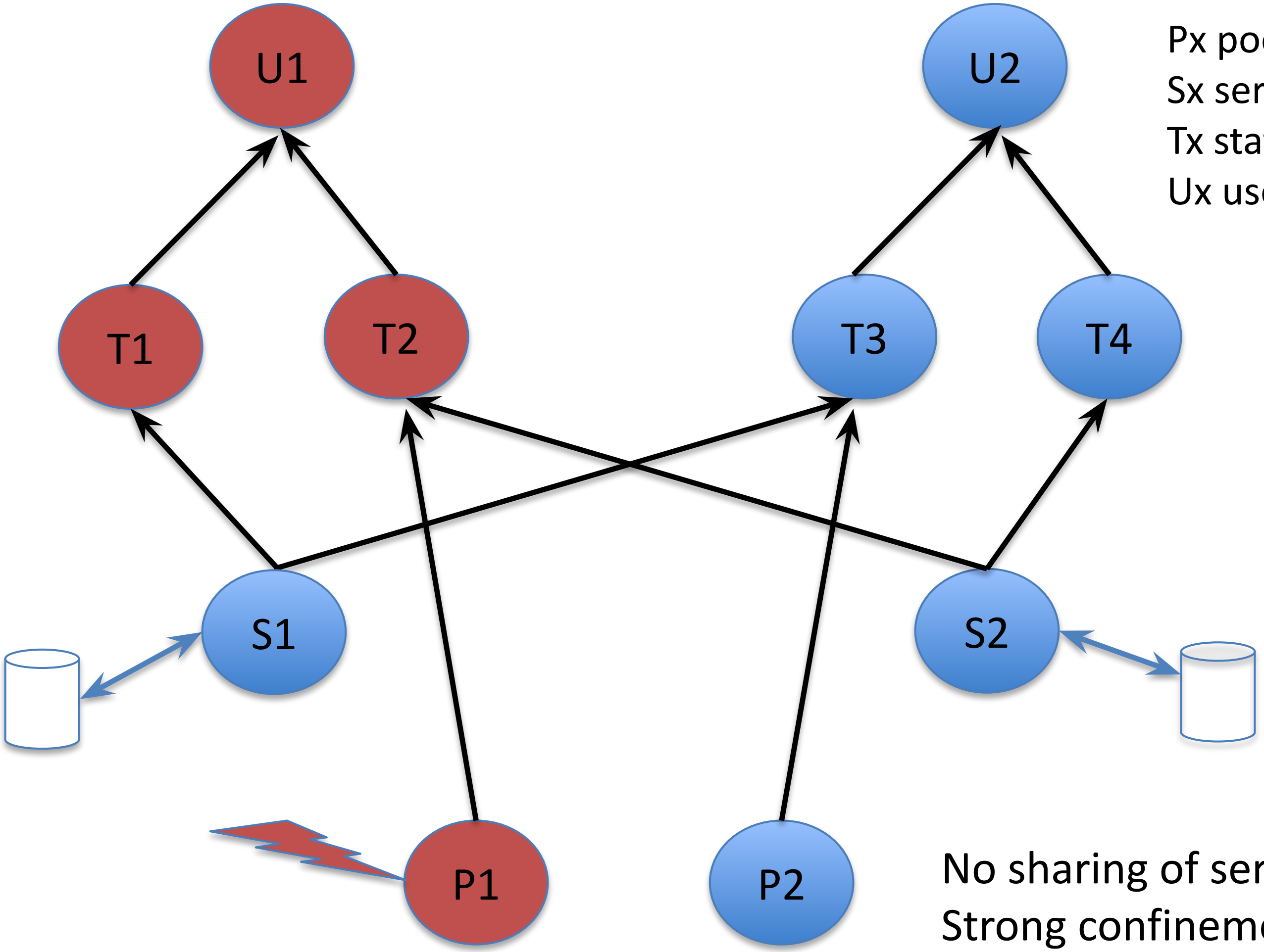
Narrow set of database access privileges

Prevent unrequired access to the DB service

https://www.usenix.org/event/usenix04/tech/general/full_papers/krohn/krohn.pdf

https://www.okcupid.com

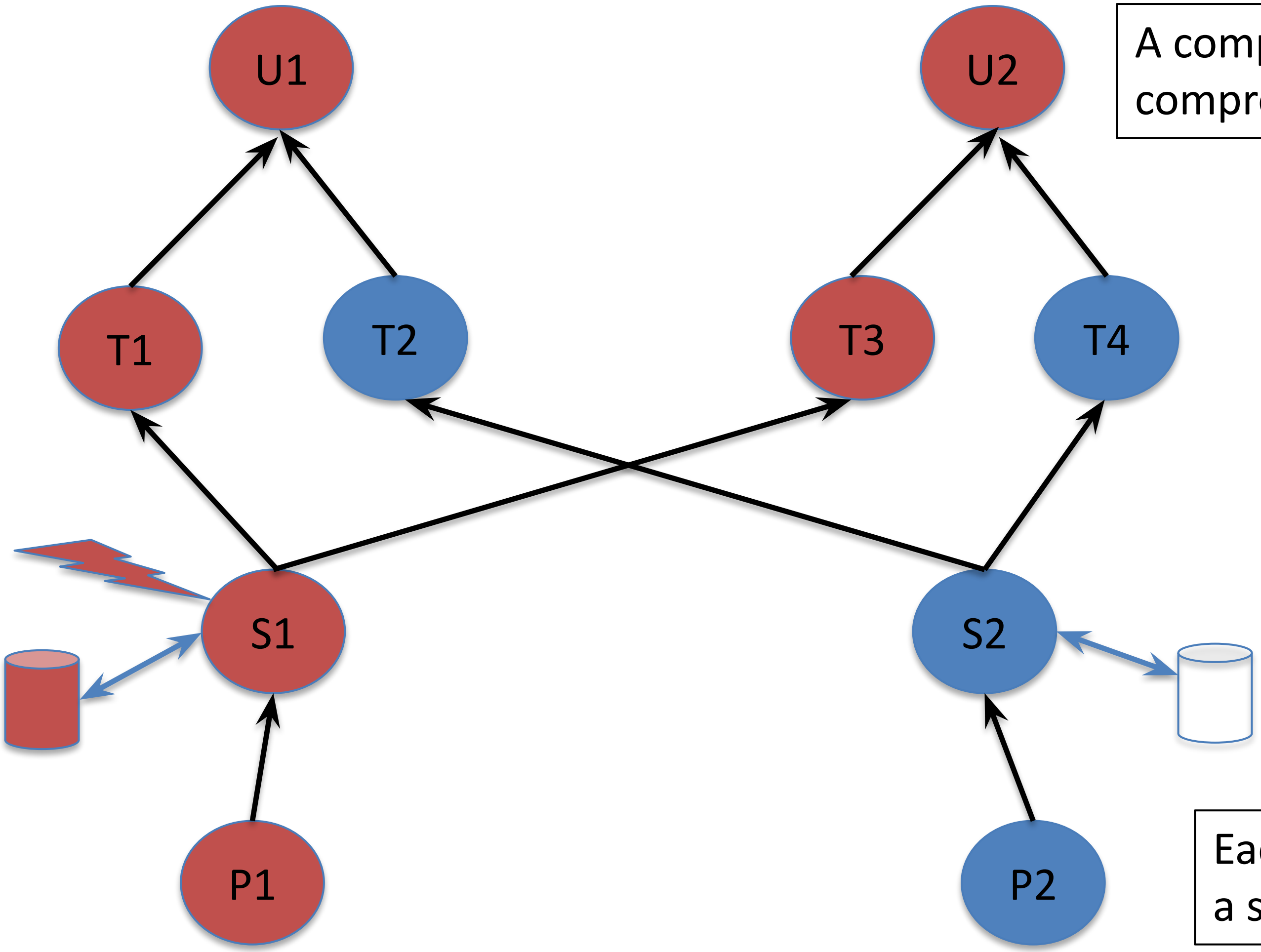
Strict Confinement



Px pool of processes
Sx services access private databases
Tx state data
Ux users

No sharing of services or processes;
Strong confinement;
Low performance due to too many processes

OKWS



A compromised service will compromise only the data

Each service associated with a single process

OKWS Design

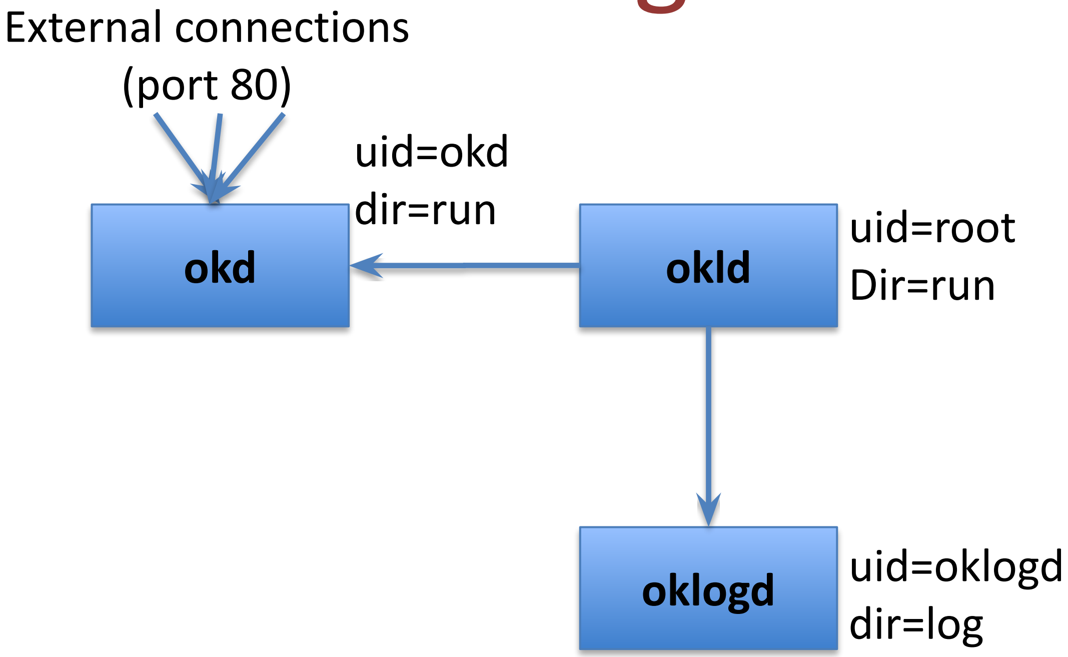
okld

uid=root
dir=run

runs as superuser; bootstrapping; chroot directory is run

Monitors processes; relaunches them if they crash

OKWS Design



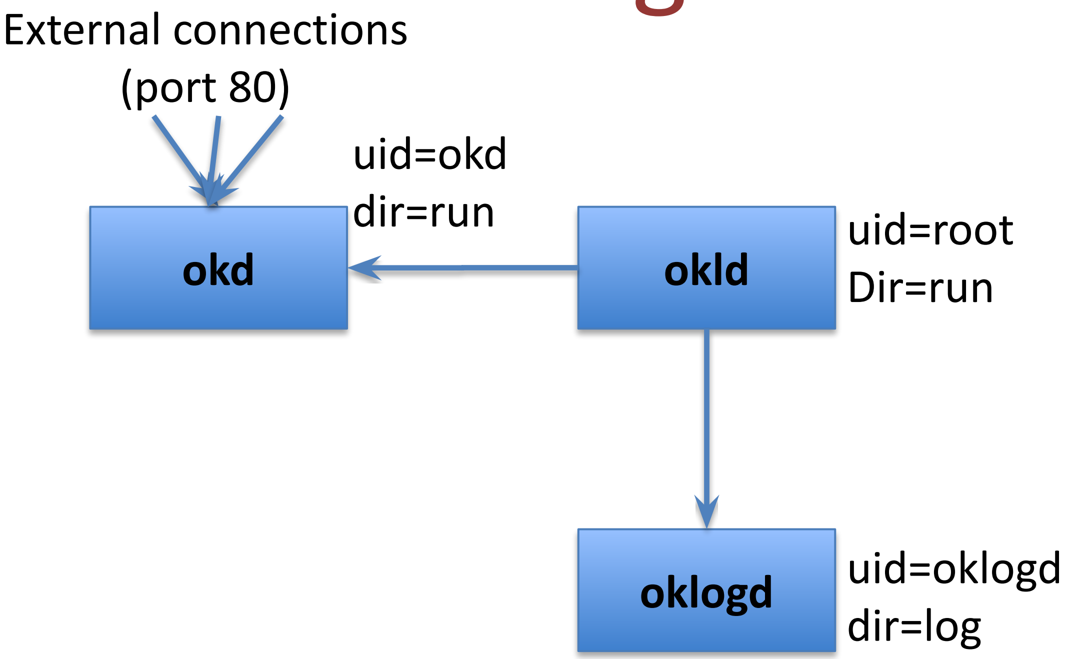
Launch okd (demux daemon) to route traffice to appropriate service ;

If request is valid, forwards the request to the appropriate service

If request is invalid, send HTTP 404 error to the remote client

If request is broken, send HTTP 500 error to the remote client

OKWS Design



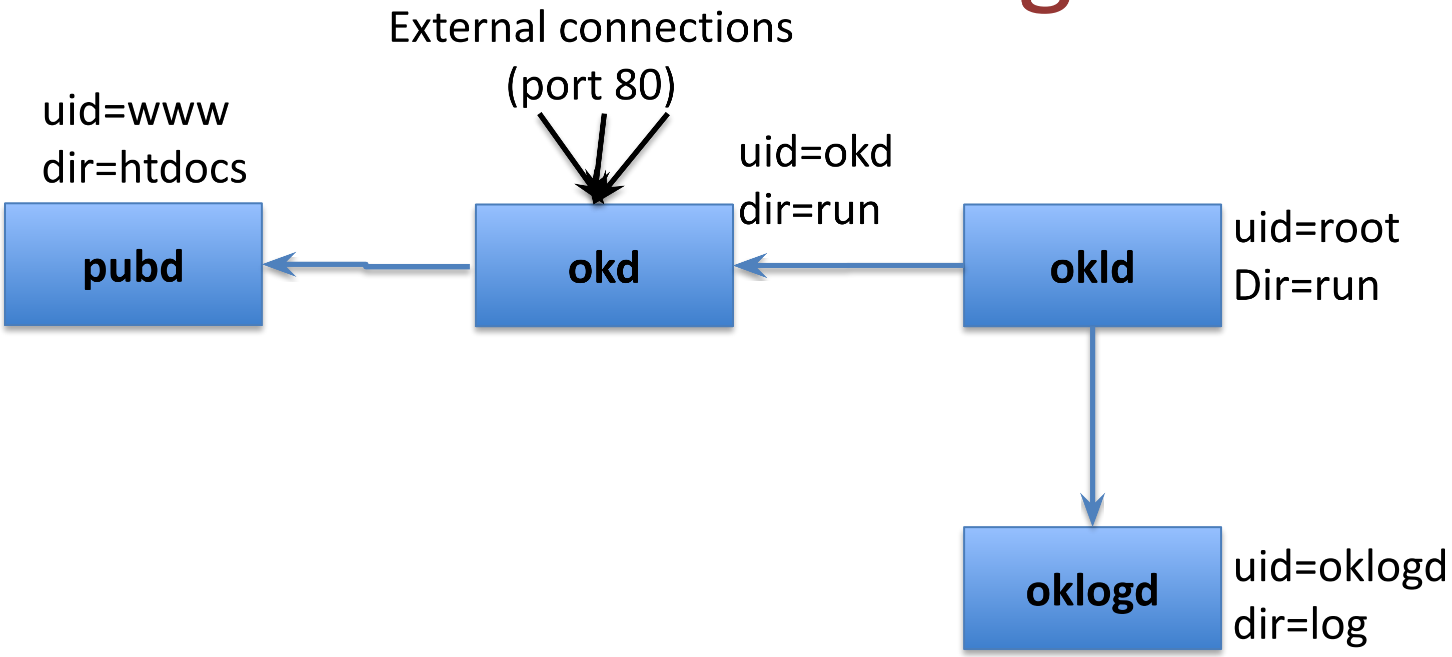
Launch okd (demux daemon) to route traffice to appropriate service ;

oklogd daemon to write log entries to disk

chroot into their own runtime jail (within a jail, each process has just enough access privileges to read shared libraries on startup, dump core files if crash)

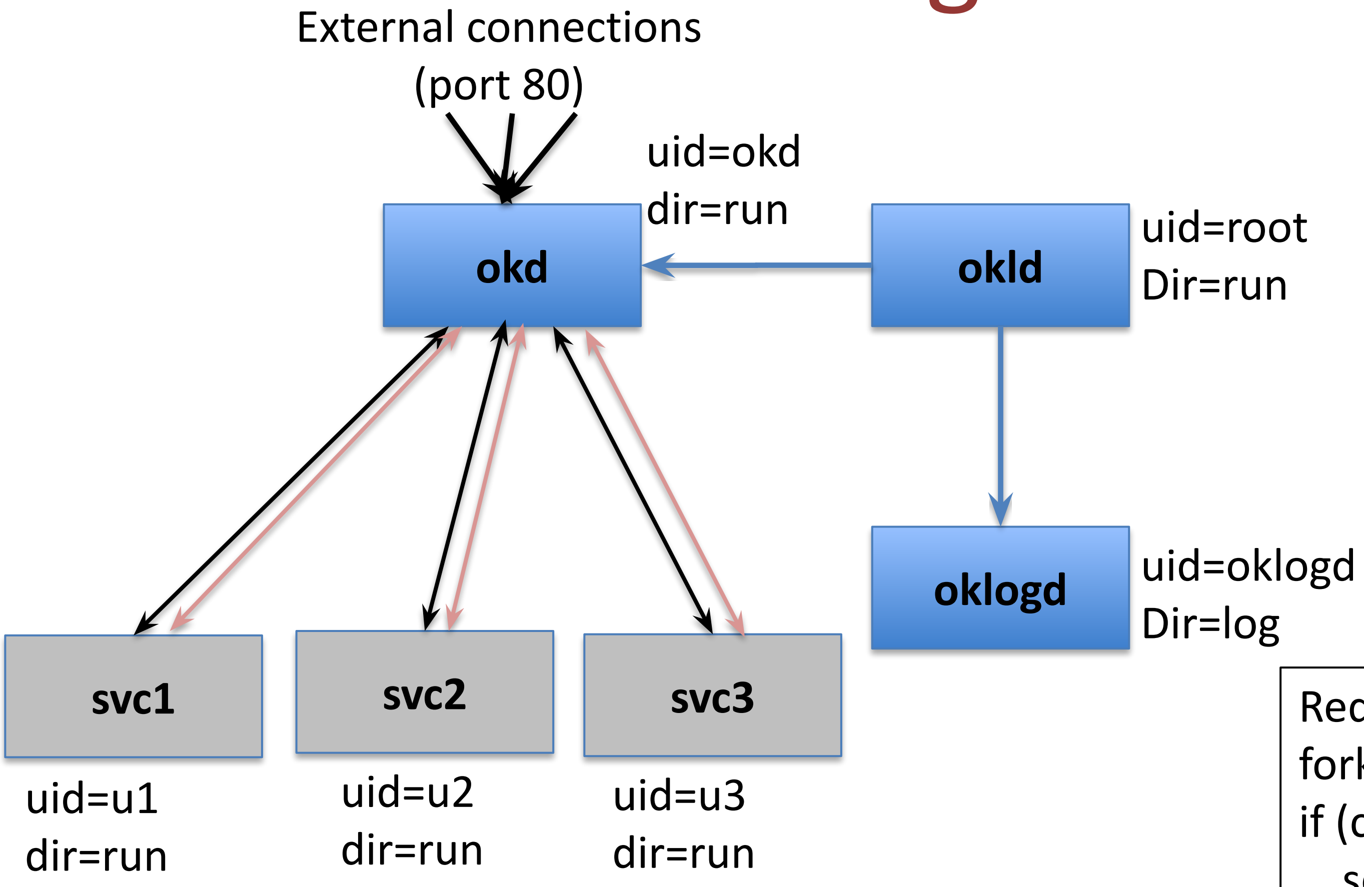
Each service runs as an unprivileged user

OKWS Design



pubd: provides minimal access to local configuration files

OKWS Design



```
Request 2 sockets
fork()
if (child process){
   setuid()
   chroot()
   exec()
}
```

okld launch services; each service in its chroot with its own uid

Logging

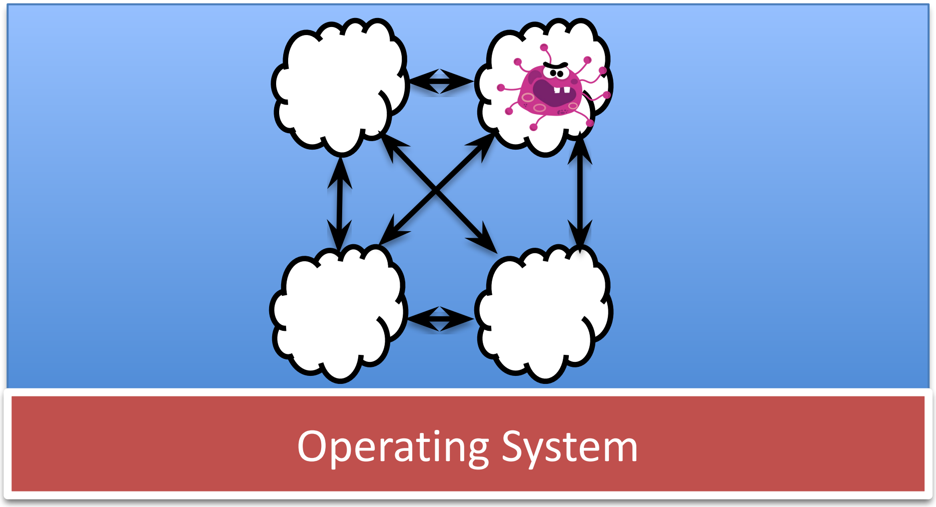
- Each service uses the same logging file
  - They use the oklogd to write into the file via RPCs
  - oklogd runs in its own chroot jail
    - Any compromised service will not be able to modify / read the log
    - A compromised service may be able to write arbitrary messages to the log (noise)

Confinement within a process (using RPCs)

- Overheads due to communication may be significant

  communication involves a trap to OS , context switch overheads (saving the context, switching address spaces, flushing of TLB, etc.), coping of arguments from caller to callee stacks. This **happens twice on every RPC**: on invocation and also on return

- Assumes Hardware and OS is secure (Hardware and OS may have bugs)
- Applications now highly dependent on OS (one app cannot work on another OS, may be a problem for instance for web-applications)



Operating System

Web Browser Confinement

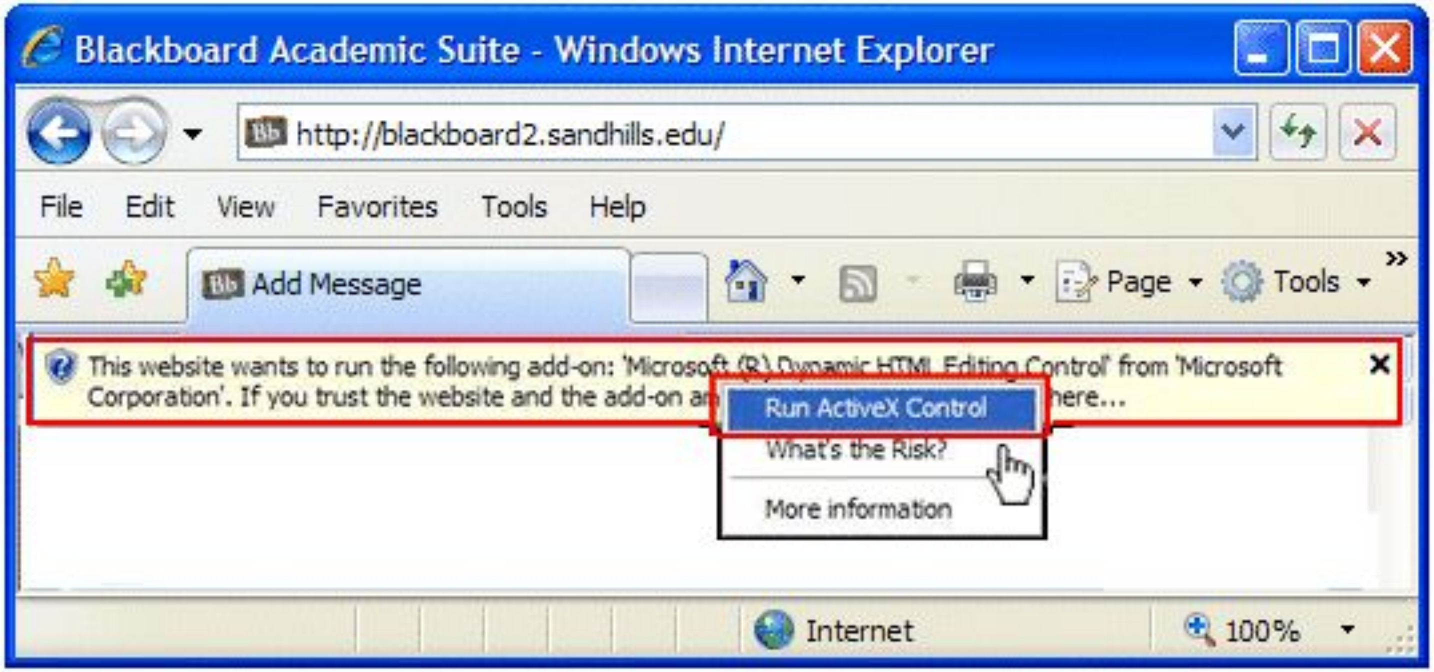
- Why run C/C++ code in web browser
  - Javascript highly restrictive / very slow
  - Not suitable for high end graphics / web games
  - Would permit extensive client side computation

- Why not to run C/C++ code in web browser
  - Security!
    Difficult to trust C/C++ code

Web Browser Confinement

- How to allow an untrusted module to load into a web-browser?
  - Trust the developer / User decides

    Active X



This website wants to run the following add-on: 'Microsoft (R) Dynamic HTML Editing Control' from 'Microsoft Corporation'. If you trust the website and the add-on a here...

Web Browser Confinement

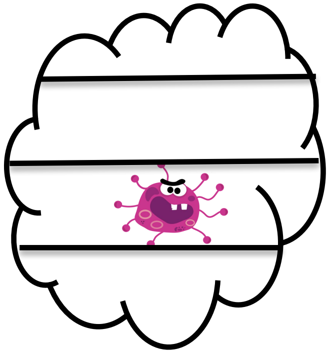
- How to allow an C/C++ in a web-browser?
  - Trust the developer / User decides

    Active X
  - Fine grained confinement
    - (eg. NACL from Google)
    - Uses Software Fault Isolation

Fine Confinement within a Process

- How to
  - restrict a module from jumping outside its module
  - Restrict read/modification of data in another module

(jumping outside a module and access to data outside a module should be done only through prescribed interfaces)

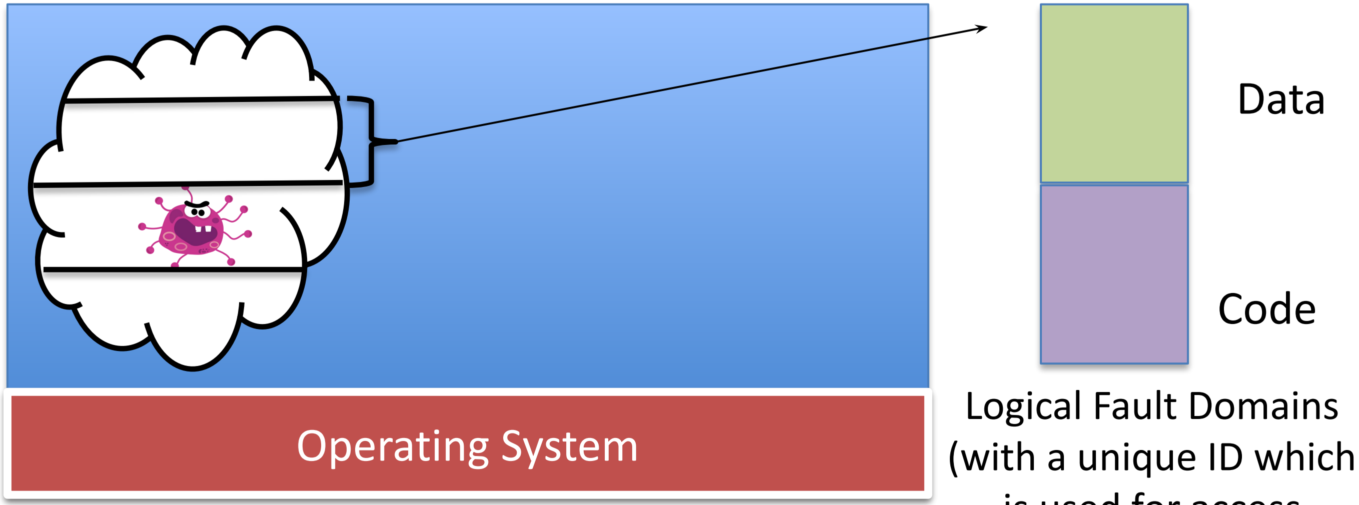


Application

Fine Grained Confinement

(Software Fault Isolation)

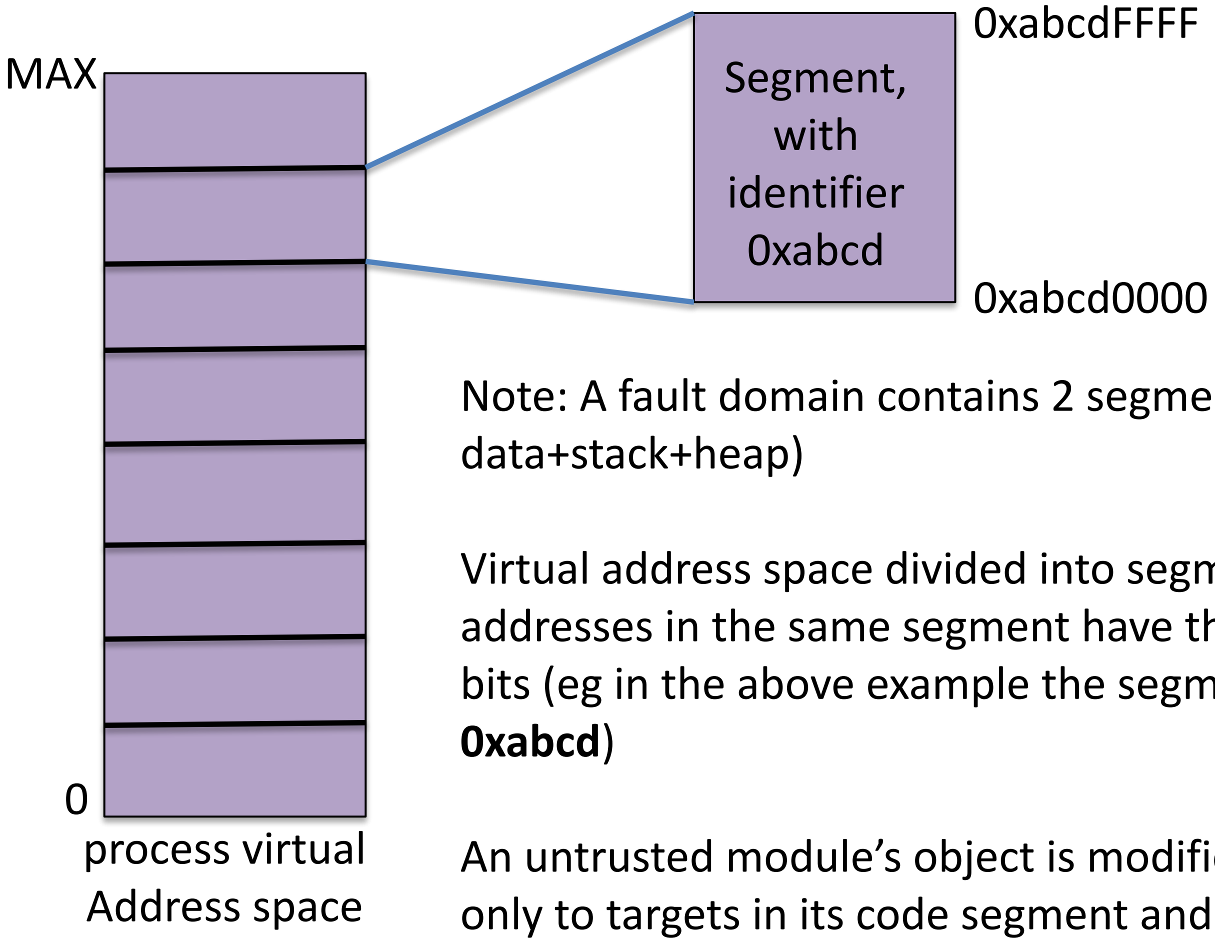
- process space partitioned into logical fault domains.
- Each fault domain contains data, code, and an unique ID
- Code in one domain not allowed to read/modify data in another domain.
- Code in one domain cannot jump to another domain.
- The only way is through a low cost cross-fault-domain RPC interface not involving the OS..



Logical Fault Domains (with a unique ID which is used for access control)

Efficient Software Fault Isolation, http://www.ece.cmu.edu/~ece732/readings/wahbe-sfi.pdf

Segments and Segment Identifier

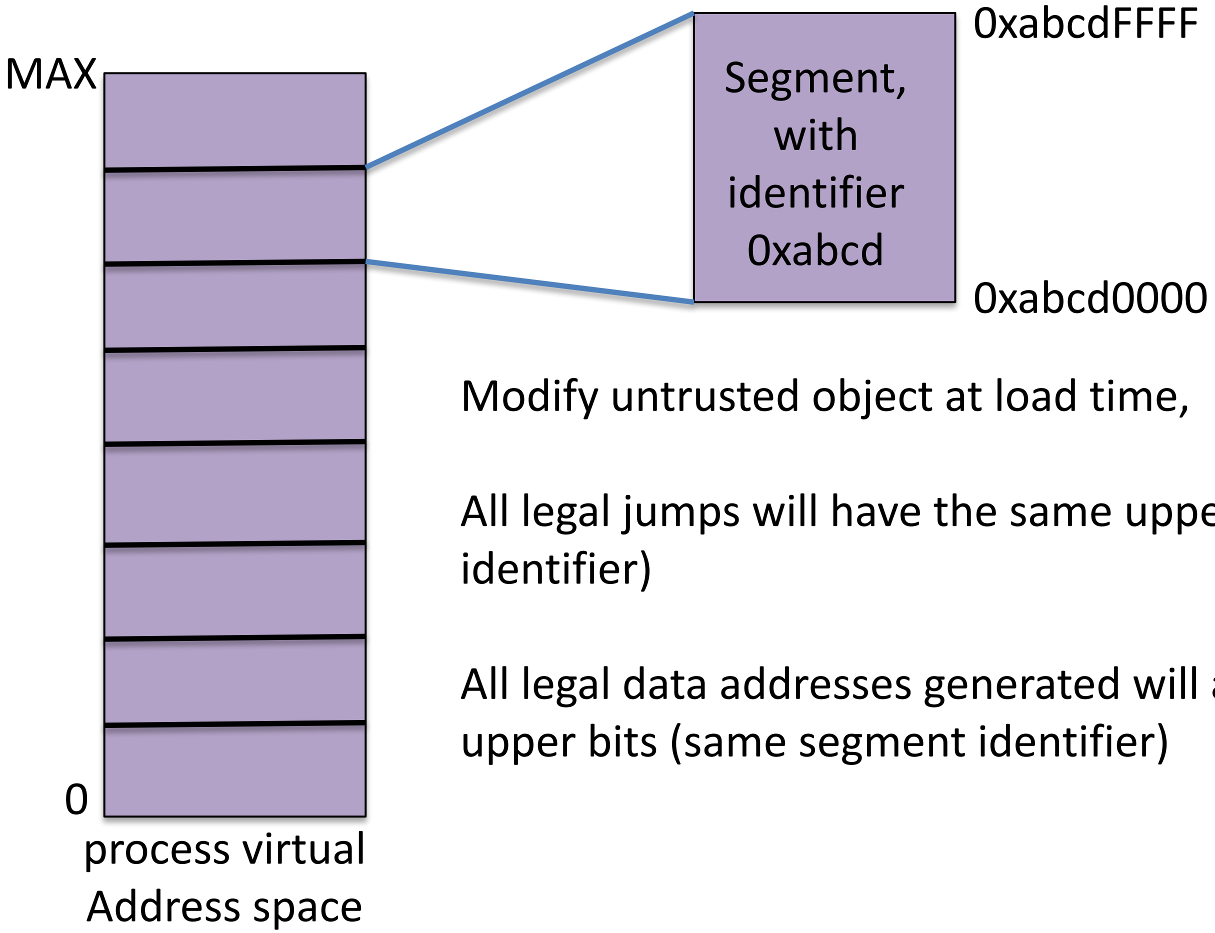


Note: A fault domain contains 2 segments (code; data+stack+heap)

Virtual address space divided into segments such that addresses in the same segment have the same upper address bits (eg in the above example the segment identifier is **0xabcd**)

An untrusted module's object is modified so that it can jump only to targets in its code segment and read/write only to addresses within its data segment.

Segments and Segment Identifier


0xabcdFFFF
MAX
Segment, with identifier 0xabcd
0xabcd0000
0
process virtual Address space

Modify untrusted object at load time,

All legal jumps will have the same upper bits (same segment identifier)

All legal data addresses generated will also have the same upper bits (same segment identifier)

Achieving Segmentation

- Binary rewriting statically
  - At the time of loading, parse through the untrusted module to determine all memory read and write instructions and jump instructions.
  - Use unique ID (upper bits) to determine if the target address is legal
  - Rewriting can be done either at compile time (modifying compiler) or at load time.
    (currently only compile time rewriting feasible)
  - A verifier also needed when the module is loaded into the fault domain.

Safe & Unsafe Instructions

Safe Instructions:

- Most instructions are safe (such as ALU instr)
- Many of the target addresses can be resolved statically
  (jumps and data addresses within the same segment id. These are also **safe instructions**)

Safe Instructions

- Compile time techniques / Load time techniques
  - Scan the binary from beginning to end.
  - Reliable disassembly: by scanning the executable linear
    - variable length instructions may be issues

```
25  CD  80  00  00                        CD  80  00  00
AND %eax, 0x000080CD                      INT $0x80
```

    - A jump may land in the middle of an instruction
    - Two ways to deal with this—
      - Ensure that all instructions are at 32 byte offsets
      - Ensure that all Jumps are to 32 byte offset

```
AND eax, 0xffffffe0
JMP *eax
```

Safe Instructions

- Compile time techniques / Load time techniques
  - Scan the binary from beginning to end.
  - Reliable disassembly: by scanning the executable linear
    - variable length instructions may be issues

```
25  CD  80  00  00                                  CD  80  00  00
AND %eax, 0x000080CD                                INT $0x80
```

    - A jump may land in the middle of an instruction
    - Two ways to deal with this—
      - Ensure that all instructions are at 32 byte offsets
      - Ensure that all Jumps are to 32 byte offset

```
AND eax, 0xffffffe0
JMP *eax
```

Unsafe Instructions

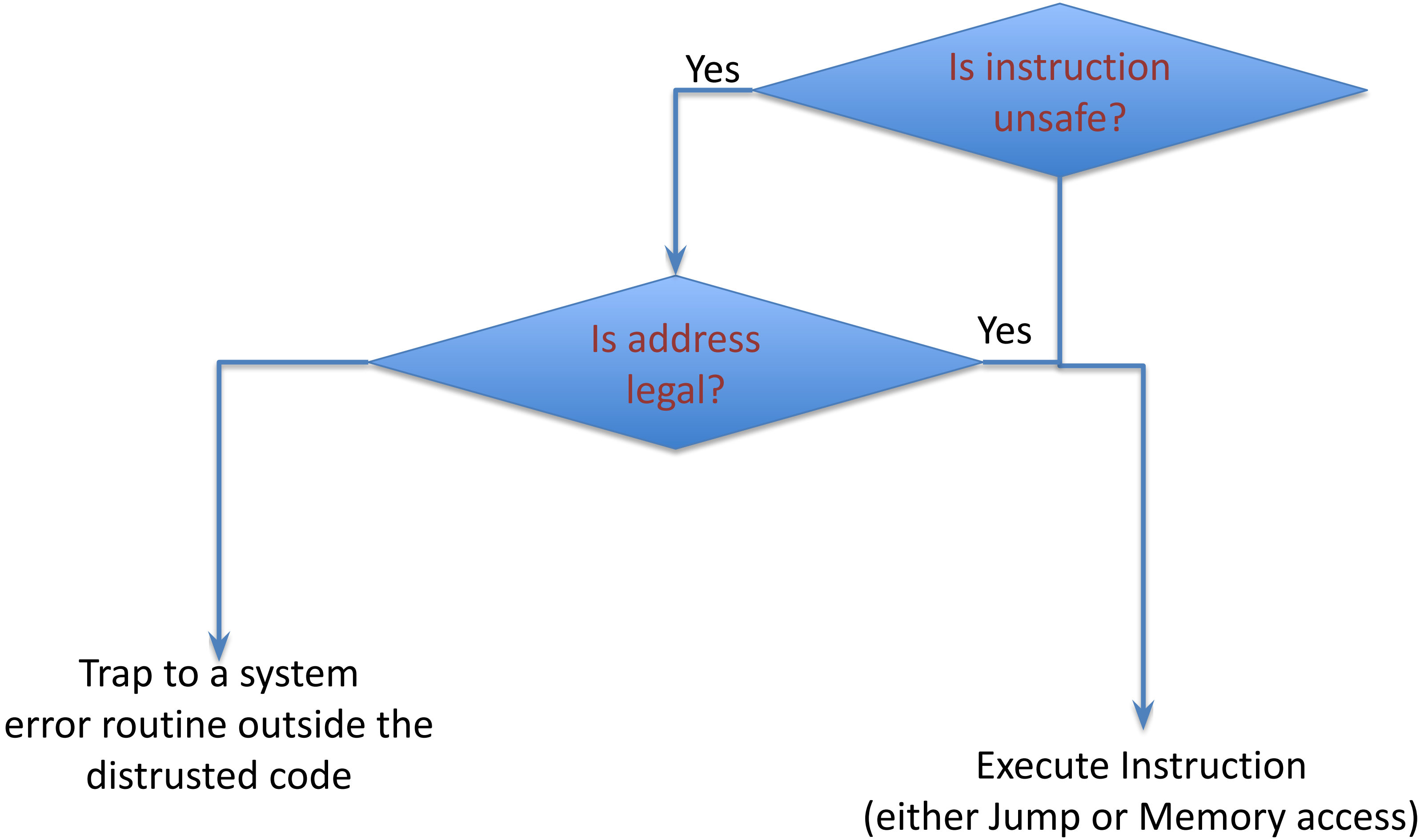
Prohibited Instructions:

- Eg. int, syscall, etc.

Unsafe Instructions: Cannot be resolved statically.

- For example *store 0x100, [r0]*
- Unsafe targets need to be validated at **runtime**
- Jumps based on registers (eg. Call *eax), and Load/stores that use indirect addressing are unsafe.
  Eg. JMP *eax

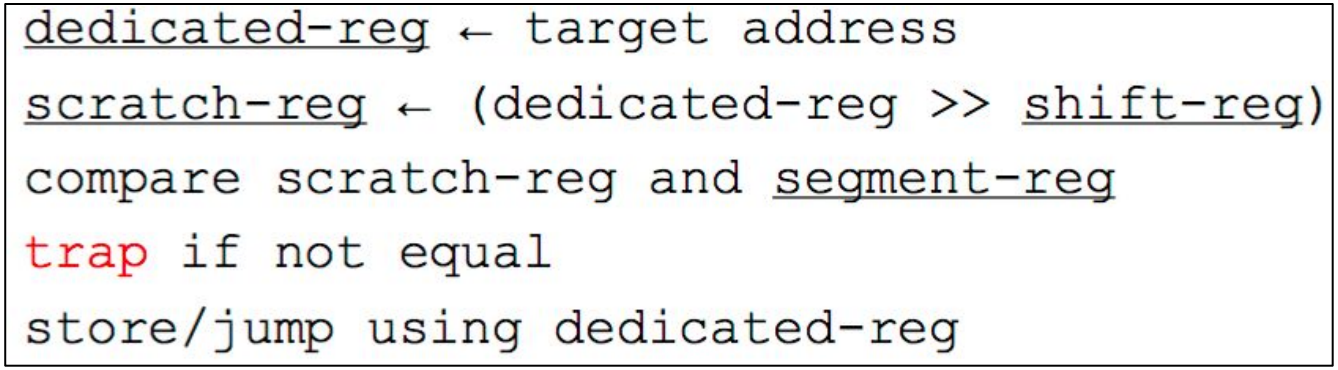
Runtime Checks for Unsafe Instructions (segment matching)


Is instruction unsafe?
Yes
Is address legal?
Yes
Trap to a system error routine outside the distrusted code
Execute Instruction (either Jump or Memory access)

Run Time Checks Segment Matching

Insert code for every unsafe instruction that would trap if the store is made outside of the segment

4 registers required (underlined registers)



```
dedicated-reg ← target address
scratch-reg ← (dedicated-reg >> shift-reg)
compare scratch-reg and segment-reg
trap if not equal
store/jump using dedicated-reg
```

Overheads increase due to additional instructions but the increase is not as high as with RPCs across memory modules.

Address Sandboxing

- Segment matching is strong checking.
  - Able to detect the faulting instruction (via the trap)
- Address Sandboxing : Performance can be improved if this fault detection mechanism is dropped.
  - Performance improved by not making the comparison but forcing the upper bits of the target address to be equal to the segment ID
  - Cannot catch illegal addresses but prevents module from illegally accessing outside its fault domain.

Segment Matching : Check :: Address Sandboxing : Enforce

Ensure Valid Instructions

- How to ensure that jump targets are at valid instruction locations
  - Ensure that all instructions are at 32 byte offsets
  - Ensure that all Jumps are to 32 byte offset

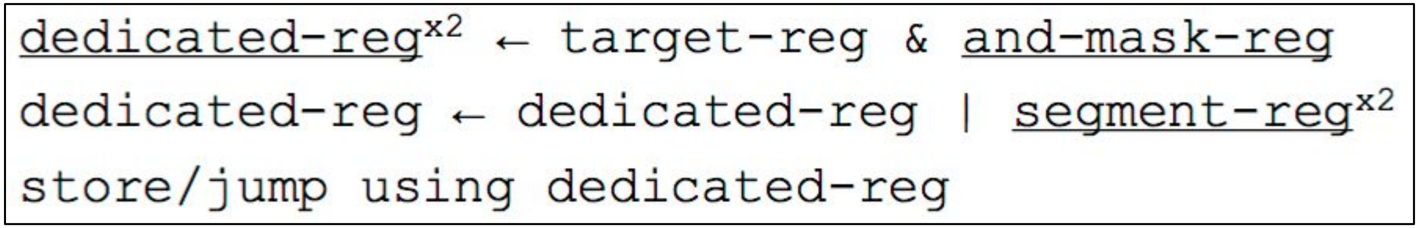
```
AND eax, 0xffffffe0
JMP *eax
```

```
25  CD  80  00  00
AND %eax, 0x000080CD
```

```
CD  80  00  00
INT $0x80
```

Address Sandboxing

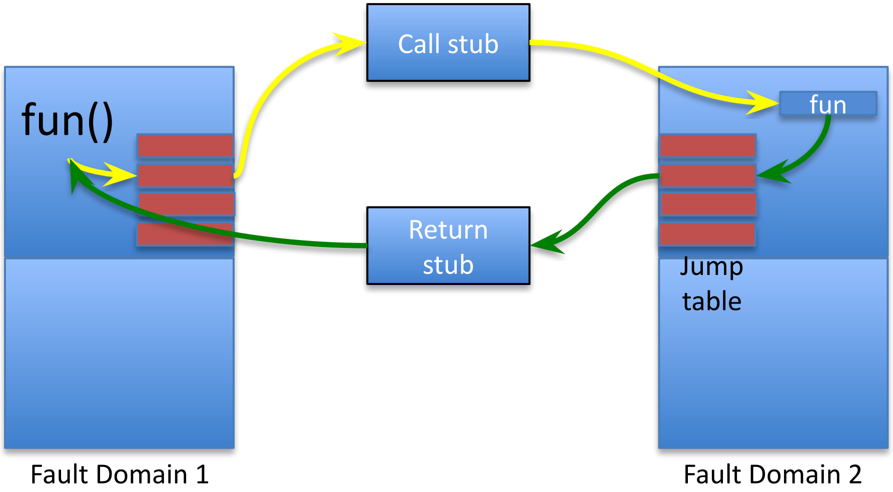
Requires 5 dedicated registers



```
dedicated-reg^x2 ← target-reg & and-mask-reg
dedicated-reg ← dedicated-reg | segment-reg^x2
store/jump using dedicated-reg
```

Enforces that the upper bits of the dedicated-reg contains the segment identifier

Calls between Fault Domains (light weight cross-fault-domain-RPC)



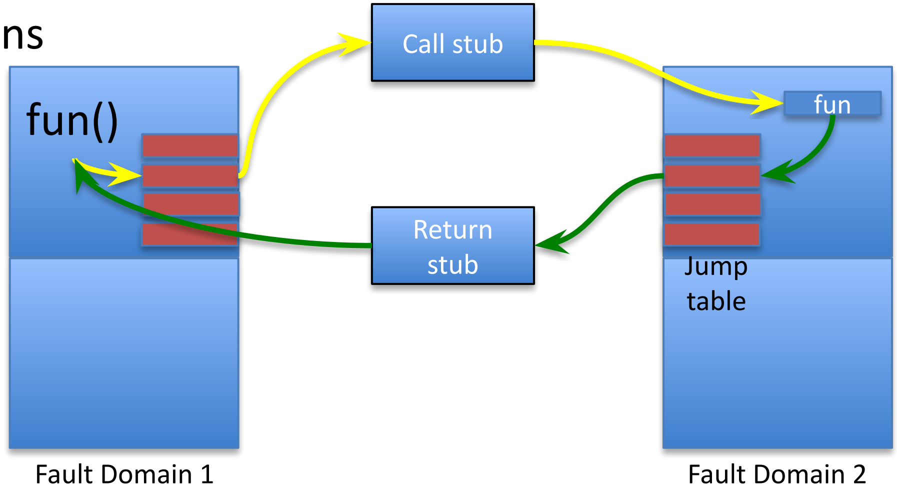
Safe calls outside a fault domain is by jump tables.

Each entry in jump table is a control transfer instruction whose target address is a legal entry point outside the domain.

Maintained in the read only segment of the program therefore cannot be modified.

Calls between Fault Domains (cross-fault-domain-RPC)

- A pair of stubs for each pair of fault domains
- Stubs are trusted
- Present outside the fault domains
- Responsible for
  - copying cross-domain arguments between domains
  - manages machine state (store/restore registers as required)
  - Switch execution stack
  - They can directly copy call arguments to the target domain
- Cheap
  - No traps, no context switches

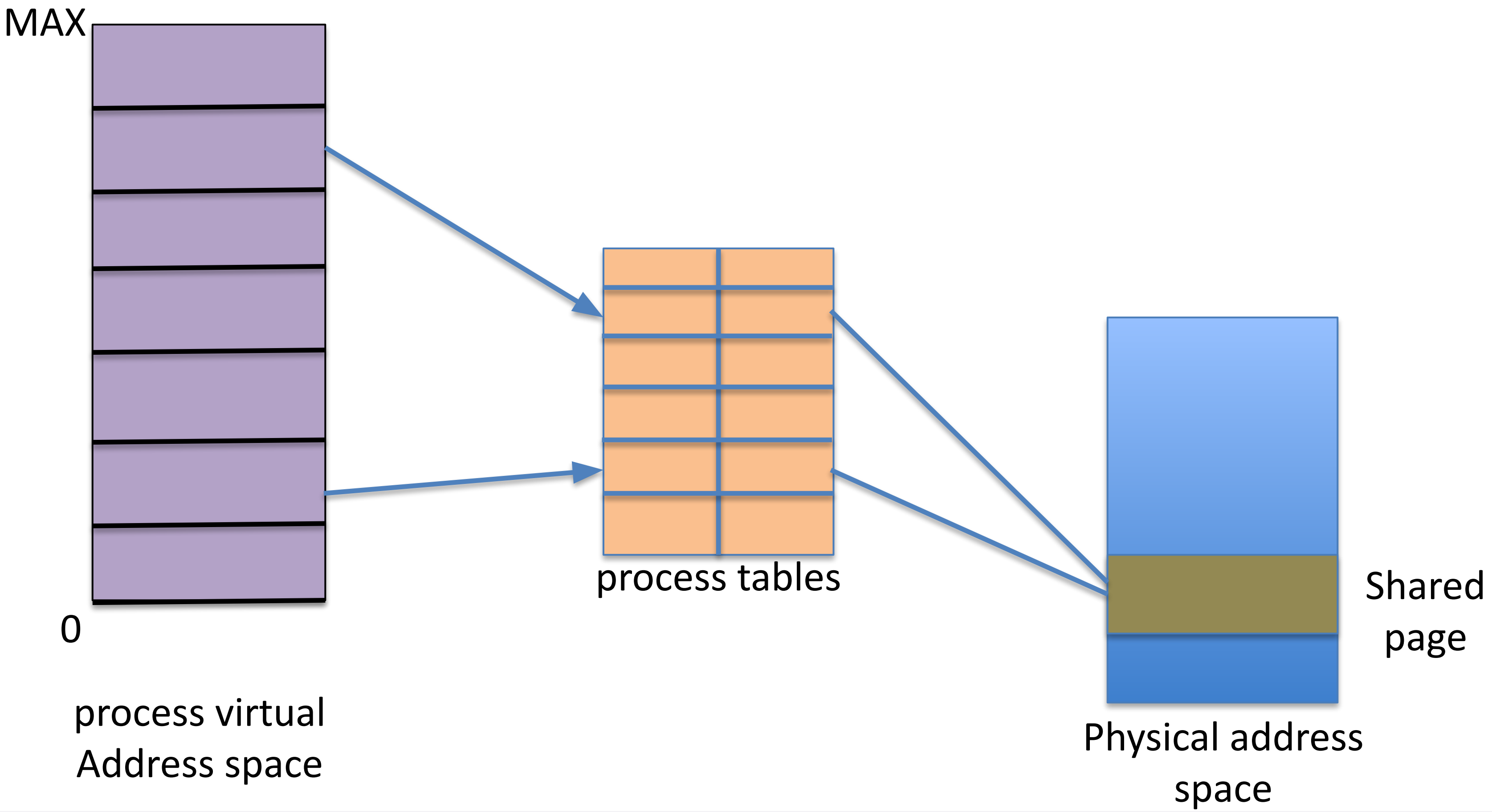


System Resources

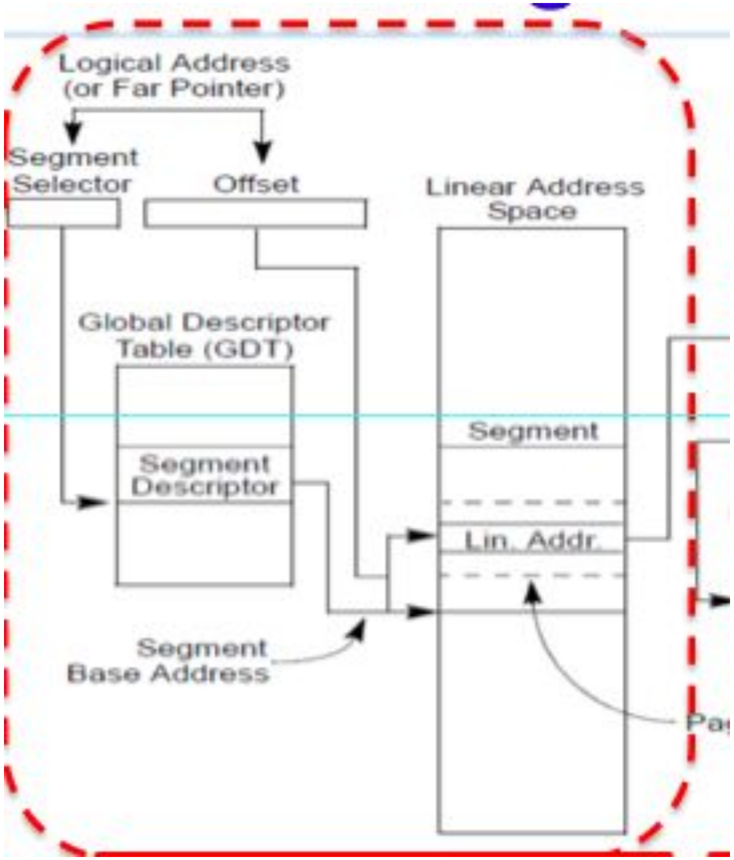
- How to ensure that one fault domain does not alter system resources used by another fault domain
  - For example, does not close the file opened by another domain
- One way,
  - Let the OS know about the fault domains
  - So, the OS keeps track if such violations are done at the system level
- Another (more portable way),
  - Modify the executable so that all system calls are made through a well defined interface called *cross-fault-domain-RPC*.
  - The cross-fault-domain-RPC will make the required checks.

Shared Data (Global / Heap Variables)

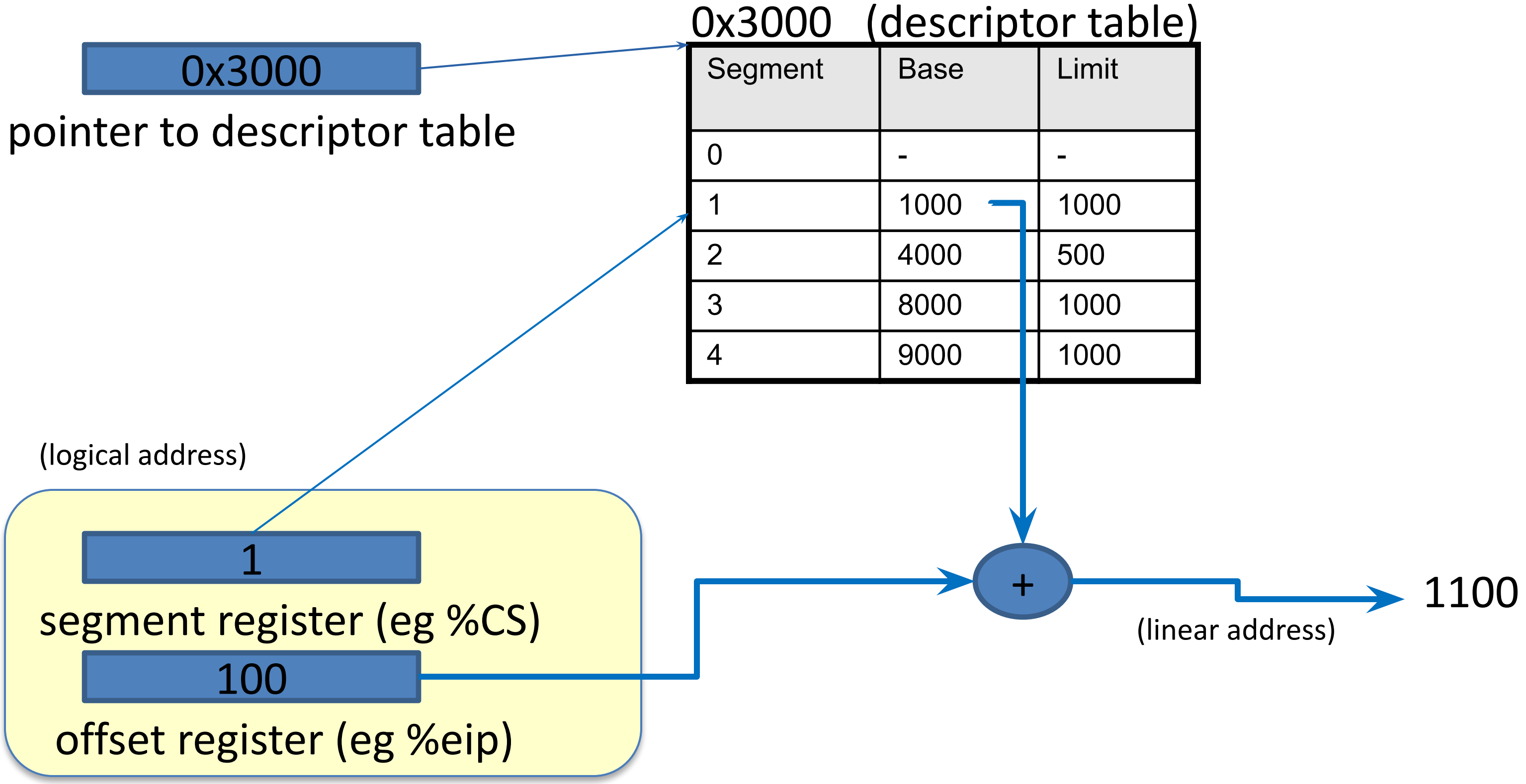
- Page tables in kernel modified so that shared memory mapped to every segment that needs access to it



Segmentation
(Hardware Support for Sandboxing)



Segmentation Example



0x3000

pointer to descriptor table

0x3000 (descriptor table)

Segment	Base	Limit
0	-	-
1	1000	1000
2	4000	500
3	8000	1000
4	9000	1000

(logical address)

1

segment register (eg %CS)

100

offset register (eg %eip)

+

1100

(linear address)

Segmentation In Sandboxing

- Create segments for each sandbox
- Make segment registers (CS, ES, DS, SS) point to these segments

- Need to ensure that the untrusted code does not modify the segment registers

- Jumping out of a segment: need to change segment registers appropriately

Usage

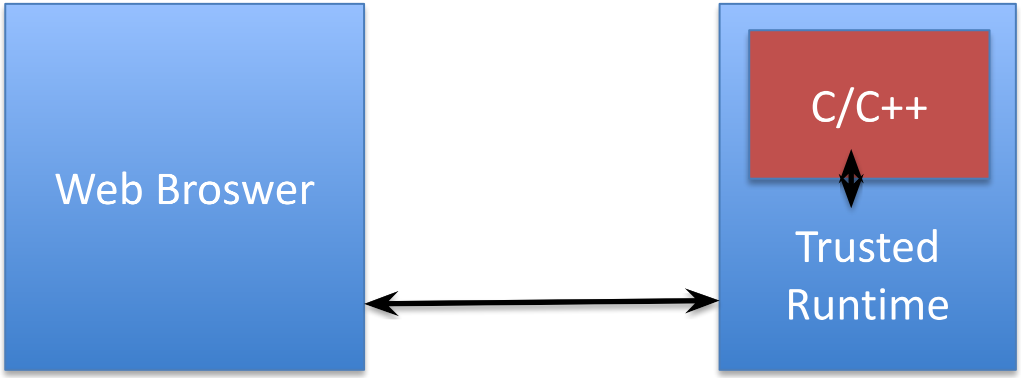
When to use it?

When you have an application with
    multiple tightly linked modules.
    a lot of shared data

If your application does not have these characteristics,
    then hardware based solutions are useful.

Native Client

- Used in Google Chrome till May 2017
- Uses SFI to run C/C++ code in a web browser (with support from Segmentation)
- A trusted environment for operations such as allocating memory, threading, message passing, etc



Native module
always loaded in
The range 0 to 256MB

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/34913.pdf

Native Client Rules

1. Binary not writable
2. Start at mem 64K offset and extend to a max of 256MB
3. Indirect jumps protected by macro instructions
4. Pad memory after code with hlt instructions until page boundary
5. Direct jumps are to valid instructions
6. No instructions that span the 32-byte boundary
7. All instructions reachable by disassembly from the start