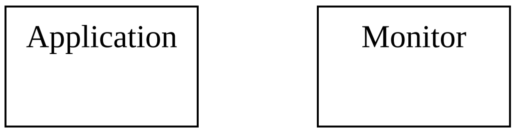
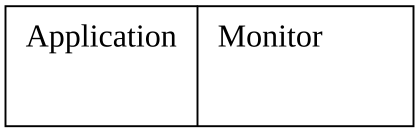
Lecture Notes Submitted by Pranav Moolwaney (03/22/2007)

Inline Reference Monitors:

Till now we have been studying architectures in which the application and the monitors are separate:



In the case of Inline reference monitors, we move the monitor into the application. We will make them share an address space. This is obviously a dangerous proposition and thus we must protect the integrity of the monitor.



Advantages:

- ⇨ Lower overhead
- ⇨ Can monitor more internal states and actions of an application
- ⇨ More powerful than the external monitor

Challenges:

- ⇨ Monitor Integrity
- ⇨ How to do it?
- ⇨ Complete mediation

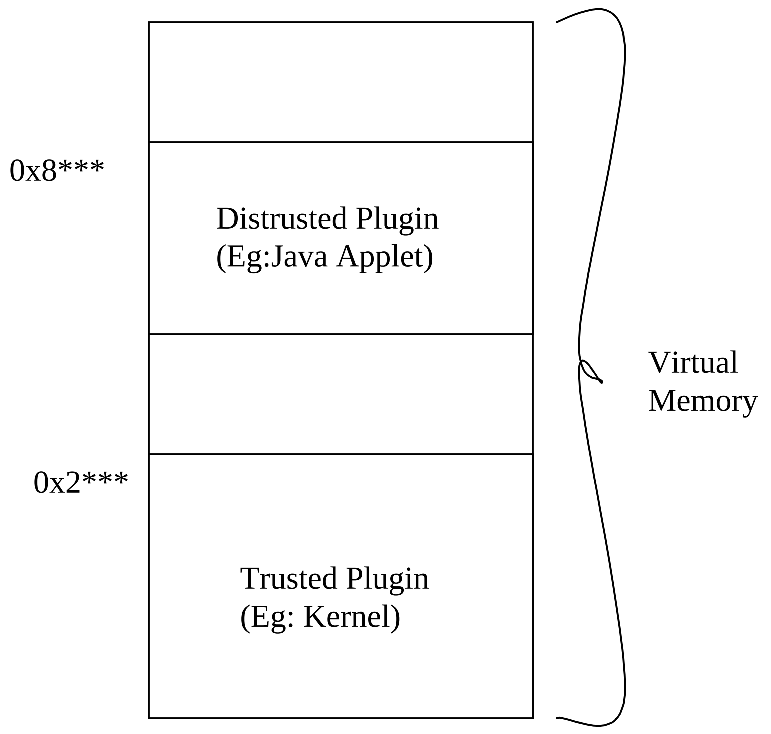
SFI: SOFTWARE FAULT ISOLATION

Scenario:

- ⇨ Distrusted plugin to trusted code.
- ⇨ Lots of RPC Calls.

Goal:

To restrict distrusted code to its own memory.


0x8***
Distrusted Plugin (Eg:Java Applet)
0x2***
Trusted Plugin (Eg: Kernel)
Virtual Memory

<u>Implementation:</u>

We rewrite the memory references:

Original Code

```
ld %r0, %r5.
```

Rewrite (check)

```
mov %r31, %r5
and %r31, %r31, ~mask
cmp %r31, seg_id
jne ABORT
ld %r0, %r5
```

Attack

Jump straight to 'ld'.

Fix:

- ⇨ Dedicate register %r30.
- ⇨ All load/store instructions use %r30
- ⇨ All moves to %r30 followed by check
- ⇨ No signals are allowed.
- ⇨
```
mov %r31, %r5
mov %r31, %r30
and %r30, %r30, ~mask
cmp %r30, seg_id
jne ABORT
ld %r0, %r30
```

Alternative (Forcing):

- ⇨
```
mov %r30, %r5
and %r30, %r30, ~mask
or %r30, %r30, seg_id
ld %r0, %r30
```

Disadvantage of Forcing:

- ⇨ We force a different instruction within the distrusted code to be executed which leads to program crash
- ⇨ We never know the source of the fault.

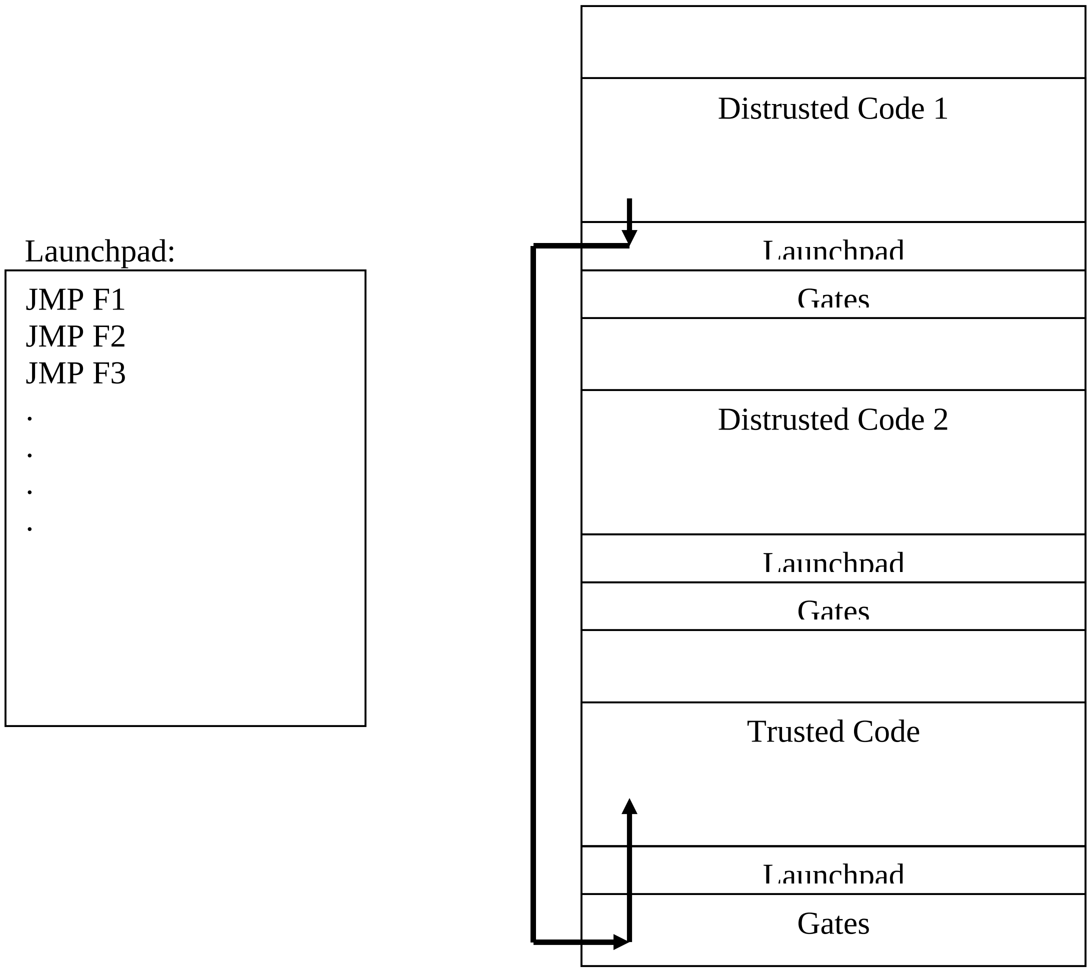
Why Signals are not allowed:

```
mov %r31, %r5
⇨ SIGNAL
mov %r31, %r30
and %r30, %r30, ~mask
cmp %r30, seg_id
```

If the attacker gets really lucky and the signal jumps to the instruction specified above, then the attacker can easily manipulate the check code.

SFI RPC

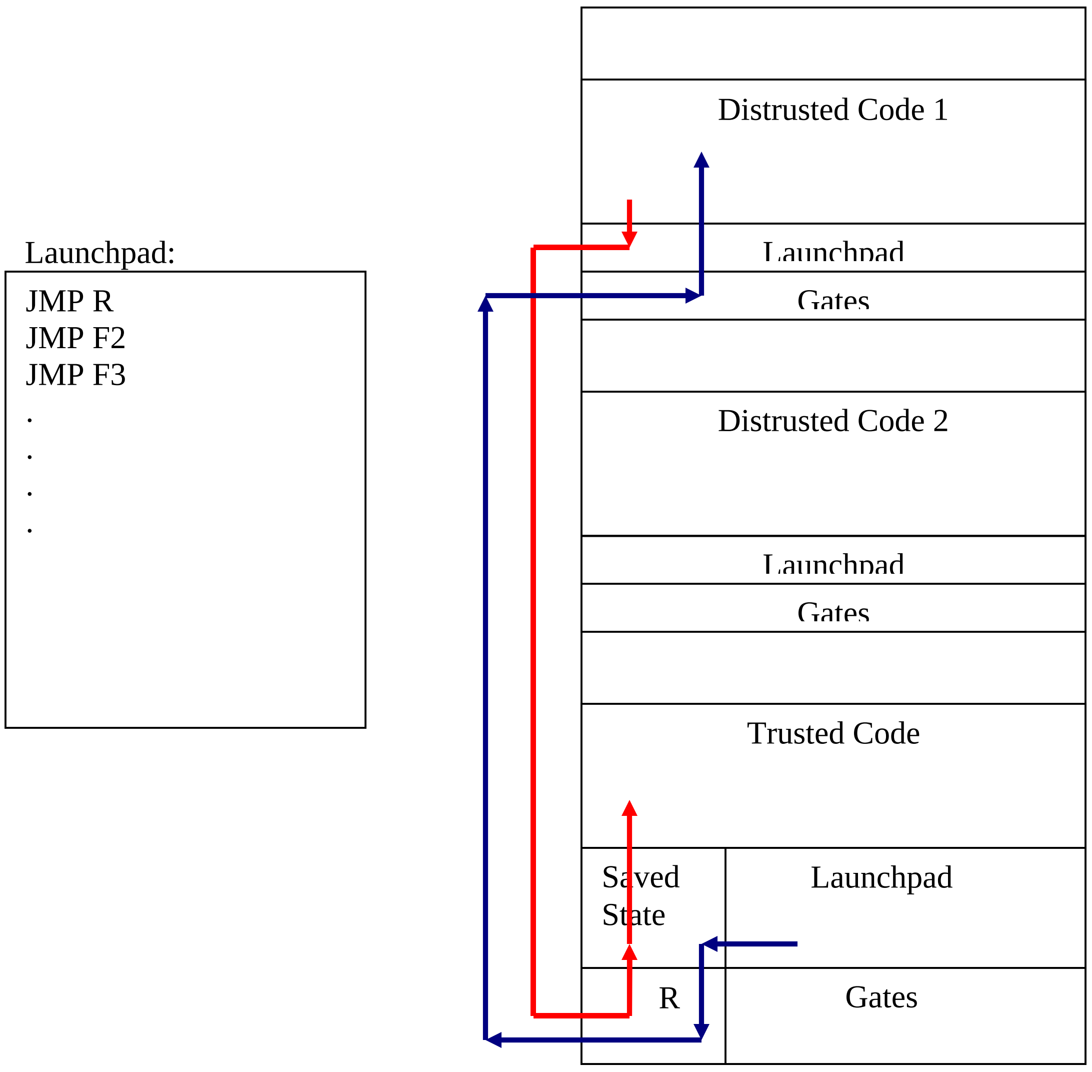
- ⇨ Distrusted module can only jump into its own segment.
- ⇨ Only escape is via launchpad
- ⇨ Only trusted party can control this launchpad
- ⇨ The code in the launchpad jumps to the trusted code.
- ⇨ Distrusted code can only jump to specified locations



- ⇨ The launchpad is a read only code page.
- ⇨ Distrusted code can safely jump anywhere in the launchpad.
- ⇨ Gates are used to move into another segment.
- ⇨ Hence a distrusted instruction takes the following route:
  - o Distrusted Code 1
  - o Launchpad of Distrusted Code 1
  - o Gate of Trusted Code
  - o Reset Register
  - o Inside Trusted Code.

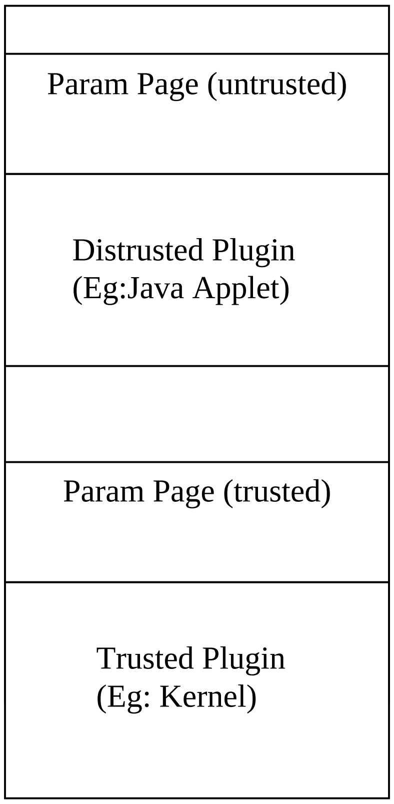
HOW TO RETURN:

We make use of 'R' which stores the return value, which restores the state of the distrusted code.



The launchpad contains JMP R, which is used to jump to the return location stored by 'R".

PARAMETER PASSING



The parameter pages are the same PHYSICAL page. We write distrusted parameters to the trusted parameter page. Since it is the same physical location the distrusted code can now access and execute and write back the results to the parameter page.

SFI RPC	OS RPC
Copy arguments	Copy arguments from caller to kernel and from kernel to callee
2 extra jumps	Save caller state
Dedicated registers	Load callee state
	OS overhead

Performance Results:

OS RPC: 200 microSeconds
SFI RPC: 1 microSeconds
Func Call: 0.1 microSeconds.