**Goal:** Fine-grained sharing  
**Obstacle:** Cost of cross-domains calls

- Unavoidable overhead: context switch  
- You can do between with share memory  
- Context switch is about 200x slower than simple function call

**Least expensive:** message passing between 2 or more machines  
Remote procedure call

SFI’s goal is to all mutually distrust modules to coexist in same address space.  
**SFI**

4GB  
0x02** { Domain 2  
About 24Mb  
0x01** { Domain 1  
0

**Security goal (if d1 does not trust d2):**  
- Integrity of each domains (memory and OS resource)  
- Confidentiality of same  
- (not security goal) How to support cross domain’s procedure calls
Assumptions:

- Load/store architecture
  - Load
  - Store  // we want to make sure store don’t get to the wrong region
  - reg.reg instructions
- wealth of registers

LS architecture usually have 32 registers and don’t know what to do with them

Let’s dedicate a register \( \text{sid} = 0x0200000 \)

```
xor $spr, $sid, $t
shifter $t, 24, $t
jne $t, ERROR
store $p, $r
// attacker can jump directly to load
```

- untrusted party inserts checks
- trusted party verifies checks before executing code
- untrusted code never writes to \( \text{sid} \)
- all loads, stores must use \( \text{pr} \)

```
\( \text{sid} = 0x0200000 \)
$\text{spr}$
mov $p, $\text{spr}$
xor $\text{spr}, \text{sid}, $t$
(shifter $t$), 24, $t$
jnz $t$, ERROR
store $r$, $\text{spr}$
```

- \( \text{spr} \) is checked after each modification
- Untrusted code performs no syscalls
• Each module given any stack and heap

• OS resource must be accessed through trusted modules

| call table  | jmp e1   |
|            | jmp e2   |
| domain 1   | jmp e3   |

Not writable

- Each domain has entry points
- Other domain should only be able to enter at the points, if not, domain can bypass with jmp.
- Also need book keeping

In RISC architecture, jmp is only one instruction

**What if you write to stack and jmp to stack?**
- Stack must be not executable
- What if system can not support it?
  - Separate data and text and check code is unwritable and data is unexecutable

**Other Tricks**
- Optimization
- Virtual memory enable share memory
- Analyze overhead

**Overhead of SFI**
Function call: 0.1 ms
SFI: 1 ms
Context switch: 200 ms