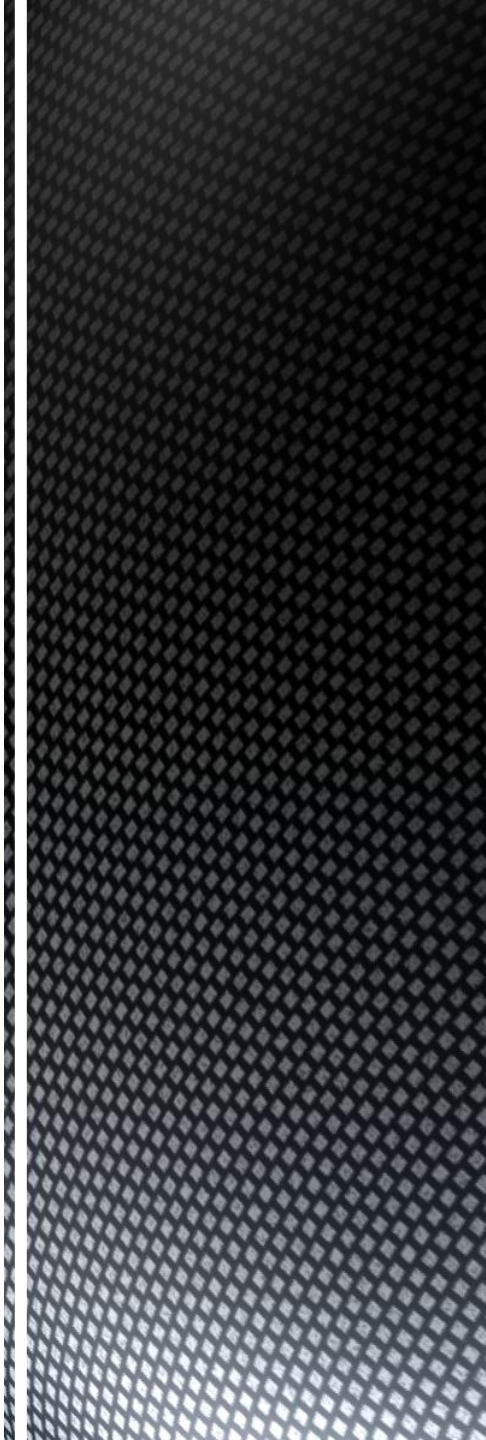
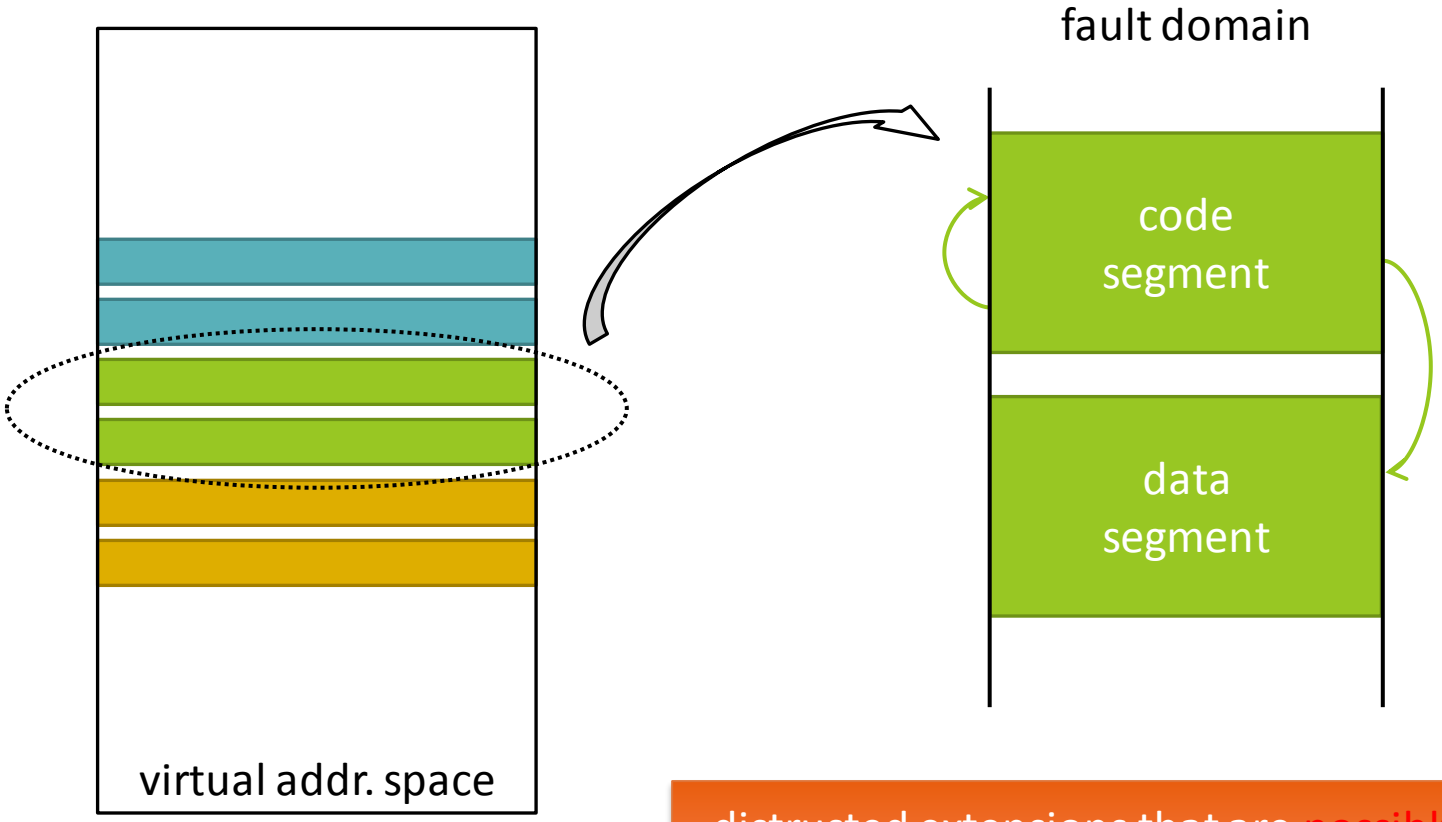
Software-Based Fault Isolation

Jinseong Jeon



software-based fault isolation



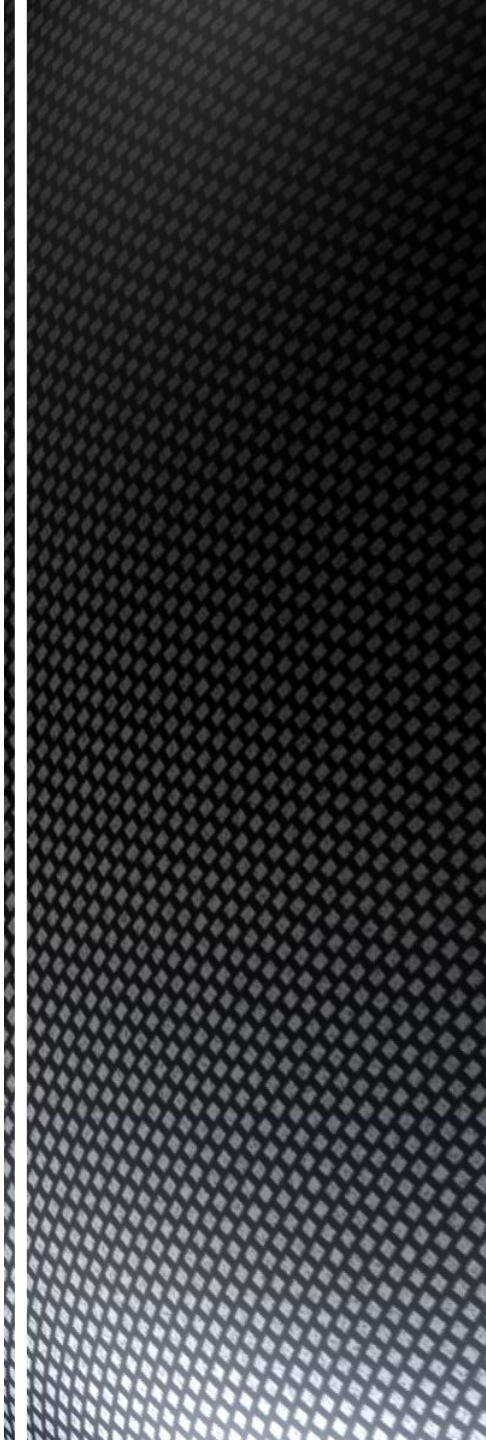
distrusted extensions that are possibly faulty

- Software-based fault isolation is the act of separating ~~something faulty~~.

Efficient Software-Based Fault Isolation

R. Vahbe, S. Lucco, T. E. Anderson, and S. L. Graham

SOSP ’93

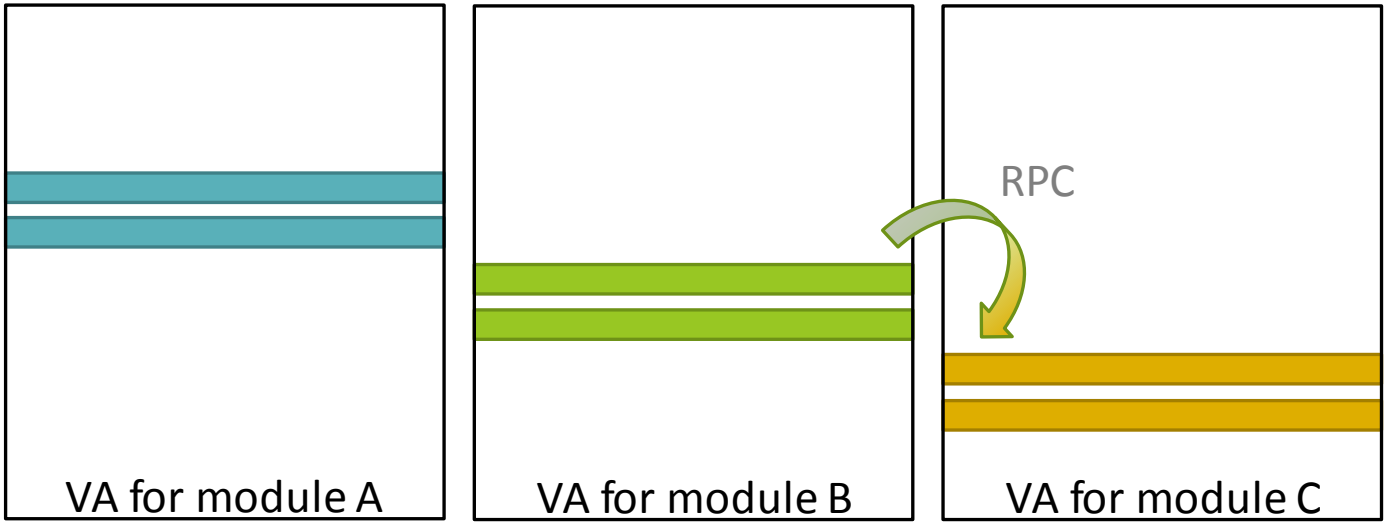


fault isolation?

- need to incorporate independently developed software modules
  - micro-kernel design
    - BSD network packet filter
    - application-specific virtual memory management
    - Active Messages
  - extensible software
    - MS object linking and embedding system
    - Quark Xpress desktop publishing system
  - high I/O processes
    - POSTGRES
- need to prevent faults in extension code from corrupting other codes or permanent data while cooperating

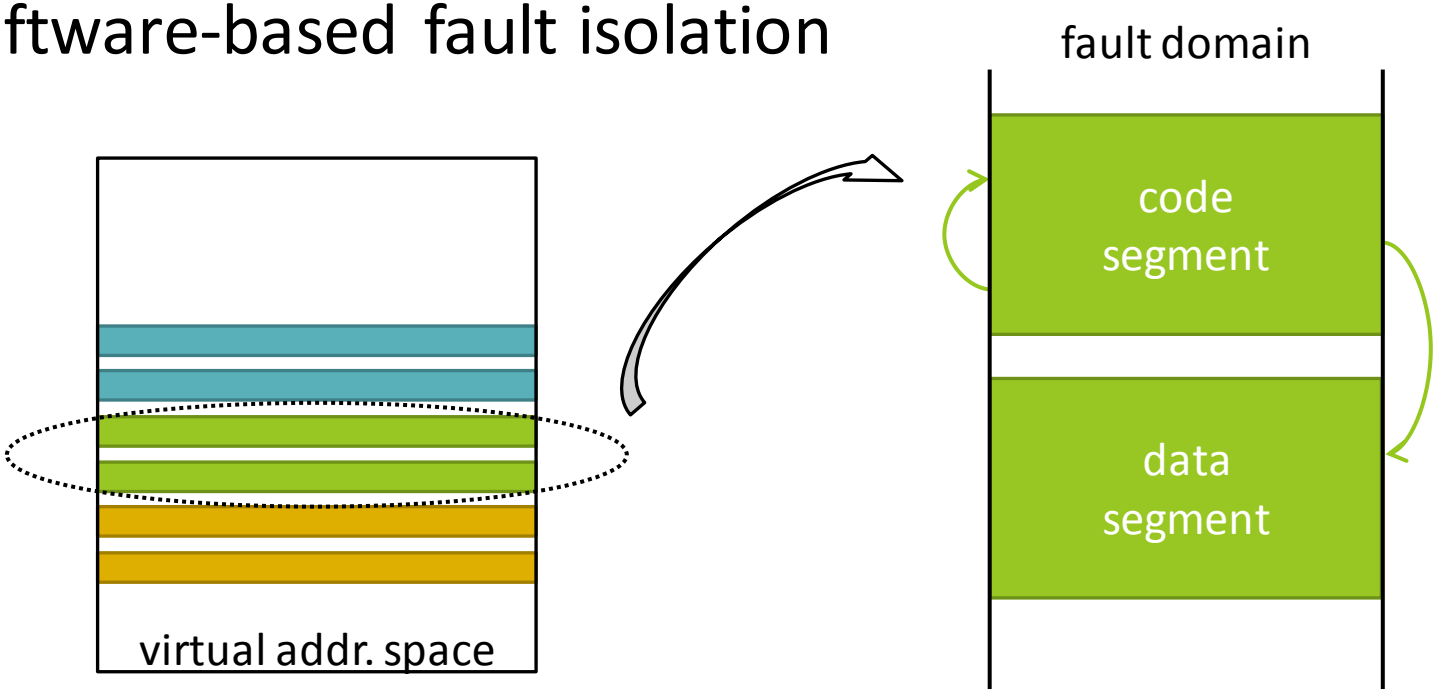
- Hence, fault isolation is an act of separating distrusted extensions.

hardware-based fault isolation



- place each software module in its own address space
- communicate through Remote Procedure Call (RPC)
  - trap into the OS kernel,
  - copying each argument from the caller to the callee,
  - saving and restoring registers,
  - switching hardware address space,
  - trap back to user level.

software-based fault isolation

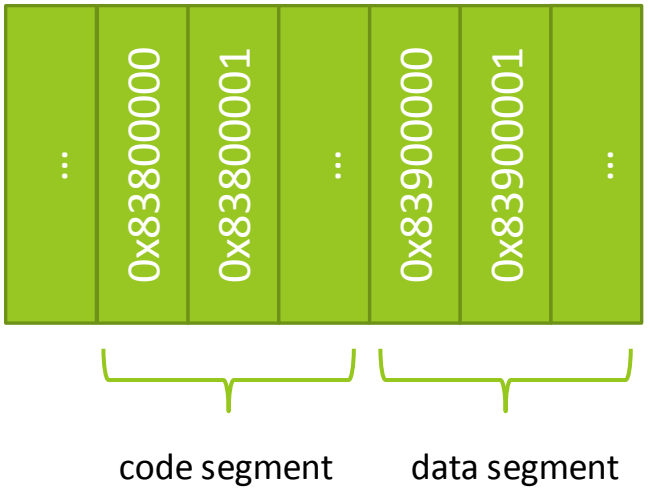


- load extension codes and their data into their own *fault domain*
  - fault domain = code segment + data segment
- enforce security policies that
  - a distrusted module is prohibited from writing or jumping outside its fault domain.
  - i.e. those distrusted modules cannot modify/execute each other's data/code.
  - the only way to do is to use explicit cross fault-domain communication.

possible questions

- how to enforce such security policies?
  - by binary rewriting
- what to rewrite, and how?
  - unsafe instruction
    - that cannot be statically verified to be within the correct segment
  - use dedicated registers
    - segment matching
    - address sandboxing
- how to share process resources and data?
  - trusted arbitration code
  - virtual address aliasing (or, shared segment matching)
- how to communicate with other fault domains?
  - explicit cross-fault-domain RPC interface
  - stub and jump table

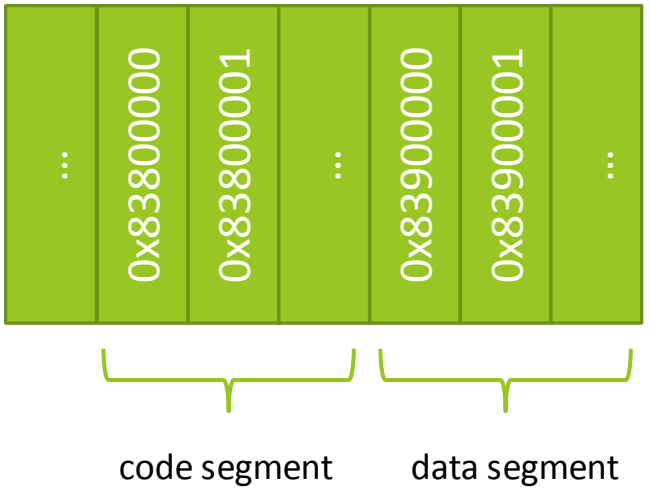
segment matching



- fault domain
  - = code segment + data segment
  - shares a unique pattern of upper bits, “segment identifier”
- insert checking code before every unsafe instruction
  - indirect jumps or stores, i.e. via registers of which value is determined at runtime
- pseudo code

```
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
```

address sandboxing

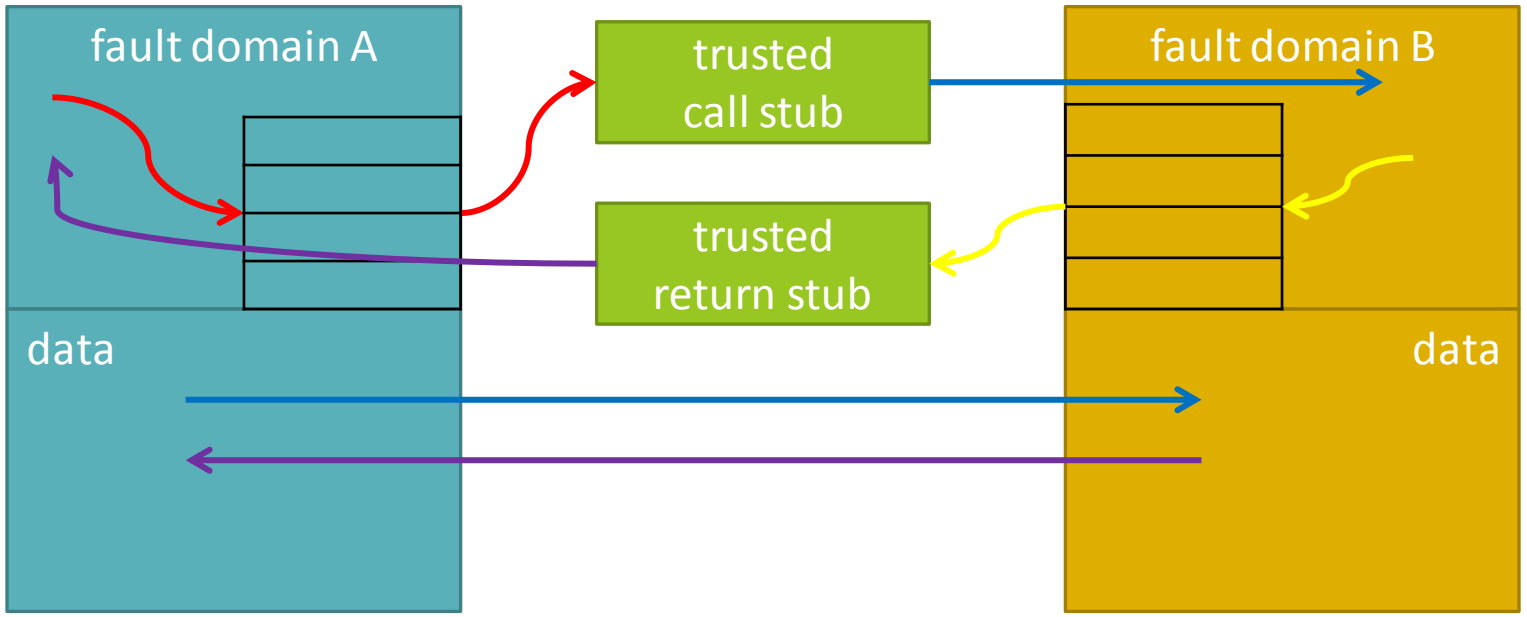


- instead of checking,
  just setting the upper bits
  to the correct segment identifier
- in the section “Ensure, don’t check”
  at the next paper,
  - check = segment matching
  - ensure = address sandboxing

- pseudo code

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

cross fault domain communication



- trusted stubs to handle RPC
  - for each pair of fault domains
  - stub: copy arguments, re/store registers, switch the exe. stack, validate dedicated regs but! no traps or address space switching (thus, cheaper than HW RPC)
- jump tables to transfer control
  - consists of jump instructions of which target address is legal, outside the domain