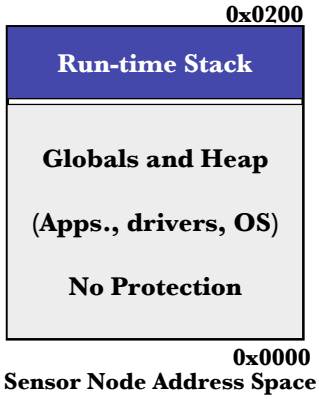
NESL Networked & Embedded Systems Laboratory

System for coarse-grained memory protection in embedded processors

Ram Kumar, Akhilesh Singhania
Mani Srivastava
Electrical Engineering
Annual Research Review

Memory Corruption



- Single address space CPU
- Shared by apps., drivers and OS
- Most bugs in deployed systems come from memory corruption
- Corrupted nodes trigger network-wide failures

Memory protection is an enabling technology for building robust embedded software

Why is Memory Protection hard ?

- No MMU in embedded micro-controllers
- MMU hardware requires lot of RAM
- Increases area and power consumption

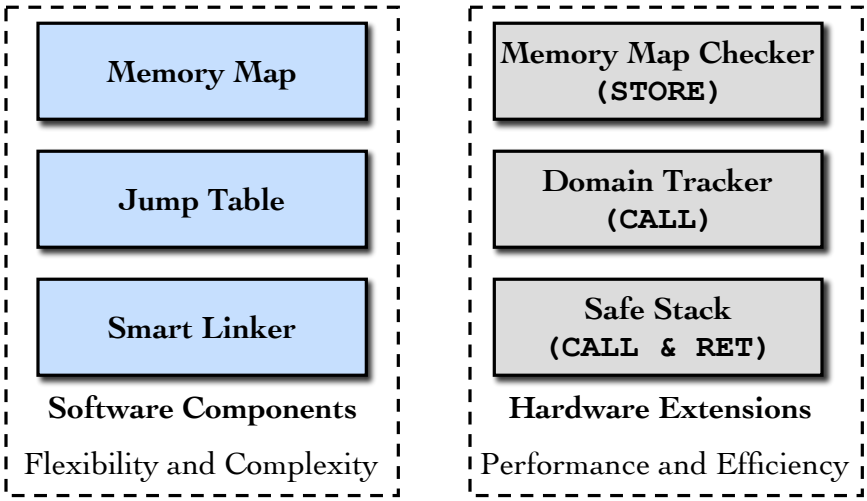
Core	MMU	Cache	Area (mm^2) 0.13u Tech.	mW per MHz
ARM7-TDMI	No	No	0.25	0.1
ARM720T	Yes	8 Kb	2.40 (~10x)	0.2 (~2x)

Software-based Approaches

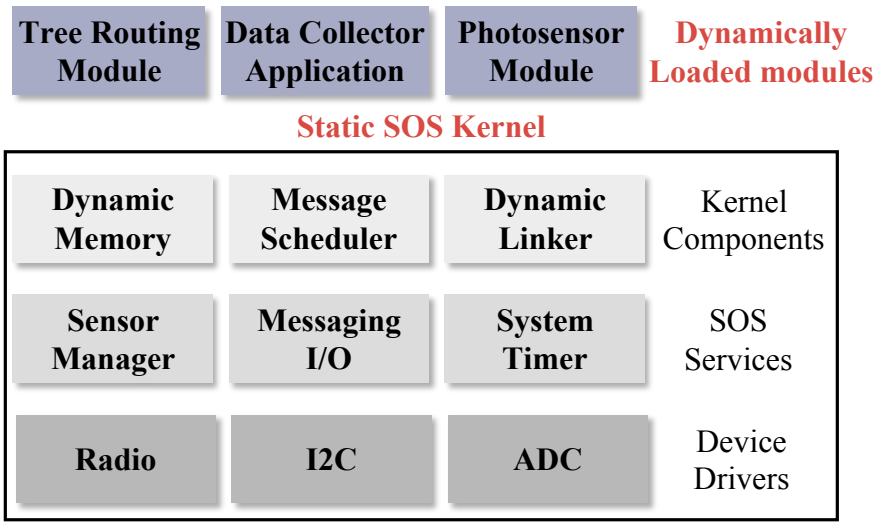
- Software-based Fault Isolation (Sandbox)
  - Inline checks validate memory accesses
  - Modify code through compiler or binary rewrite
- Application Specific Virtual Machine
  - Interpreted code is safe
  - Domain specific extensions are efficient
- Type-safe language
  - Language semantics prevent illegal accesses
  - Fine-grained memory protection
  - Ccured - Retrofit C to make it type-safe

System Overview

Hardware Software Co-Design



SOS Operating System



Design Goals

- Provide coarse-grained memory protection
  - Protect OS from applications
  - Protect applications from one another
- Targeted for resource constrained processors
  - Low RAM and ROM usage
  - Minimize area of protection logic
- Practical system
  - Customizable hardware
  - No modifications to instruction set architecture

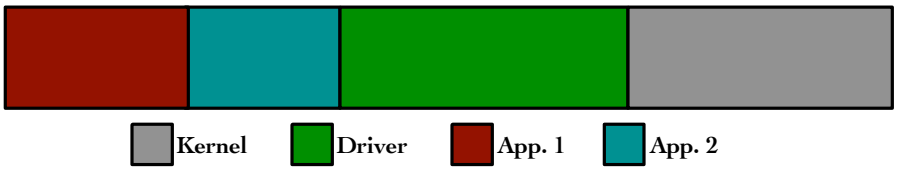
Outline

- Introduction
- **System Components**
  - **Protection Domains**
  - Memory Map Manager
  - Control Flow Manager
- Evaluation

Protection Domains

Data RAM - Non contiguous partitions

Program FLASH - Contiguous partitions

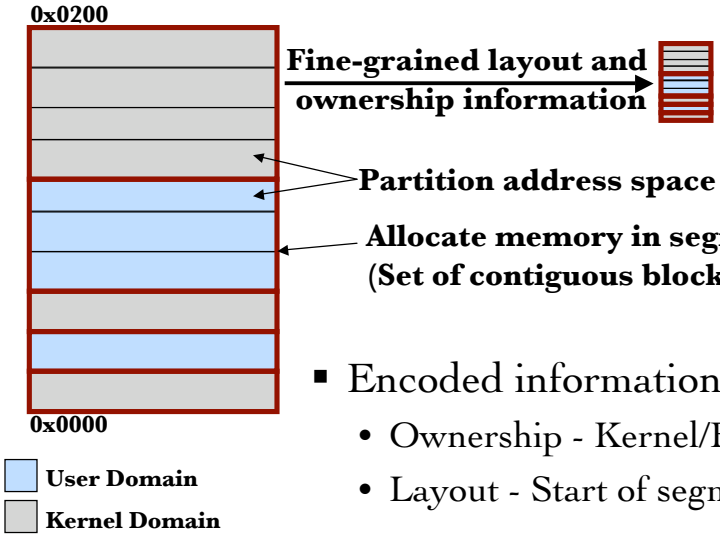


- Domains are logical partitions of address space
  - Data memory write access restricted to single domain
  - Control flow transfer across domains are checked

Outline

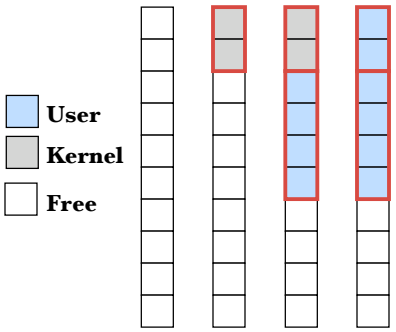
- Introduction
- **System Components**
  - Protection Domains
  - **Memory Map Manager**
  - Control Flow Manager
- Evaluation

Memory Map



- Encoded information per block
  - Ownership - Kernel/Free or User
  - Layout - Start of segment bit

Memmap in Action: User-Kernel Protection




```
pA = malloc(KERN, 16);
pB = malloc(USER, 30);
mem_chown(pA, USER);
```

- Blocks Size on Mica2 - 8 bytes
- Efficiently encoded using 2 bits per block
  - 00 - Free / Start of Kernel Allocated Segment
  - 01 - Later portion of Kernel Allocated Segment
  - 10 - Start of User Allocated Segment
  - 11 - Later Portion of User Allocated Segment

Memmap API

`memmap_set(Blk_ID, Num_blk, Dom_ID)`

- Updates Memory Map
  - Blk_ID: ID of starting block in a segment
  - Num_blk: Number of blocks in a segment
  - Dom_ID: Domain ID of owner (e.g. USER / KERN)

`Dom_ID = memmap_get(Blk_ID)`

- Returns domain ID of owner for a memory block

API accessible only from trusted domain (e.g. Kernel)

- Property verified before loading

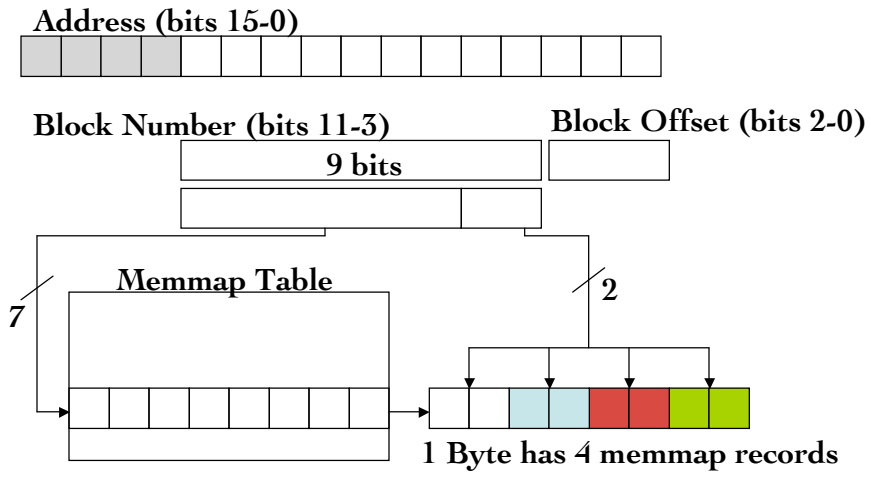
Using memory map for protection

- Protection Model
  - Write access to a block is granted only to its owner
- Systems using memory map need to ensure:
  1. Ownership information in memory map is current
  2. Only block owner can free/transfer ownership
  3. Single trusted domain has access to memory map API
  4. Store memory map in protected memory
- Easy to incorporate into existing systems
  - Modify dynamic memory allocator - malloc, free
  - Track function calls that pass memory from one domain to other
  - Changes to SOS Kernel ~ 1%
    - 103 lines in SOS memory manager
    - 12720 lines in kernel

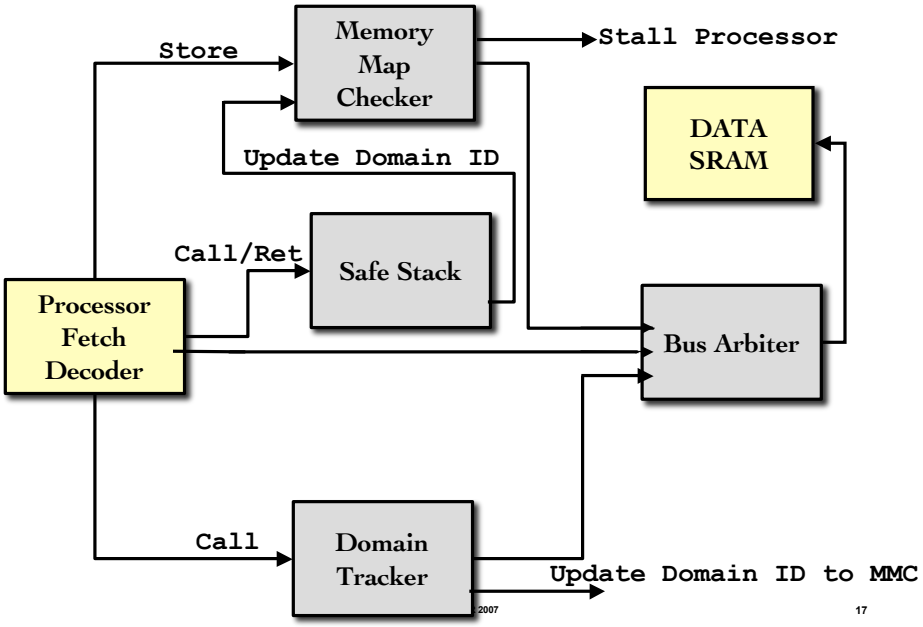
Memmap Checker

- Enforce a protection model
- Checker invoked before EVERY write access
- Protection Model
  - Write access to block granted only to owner
- Checker Operations
  - Lookup memory map based on write address
  - Verify current executing domain is block owner

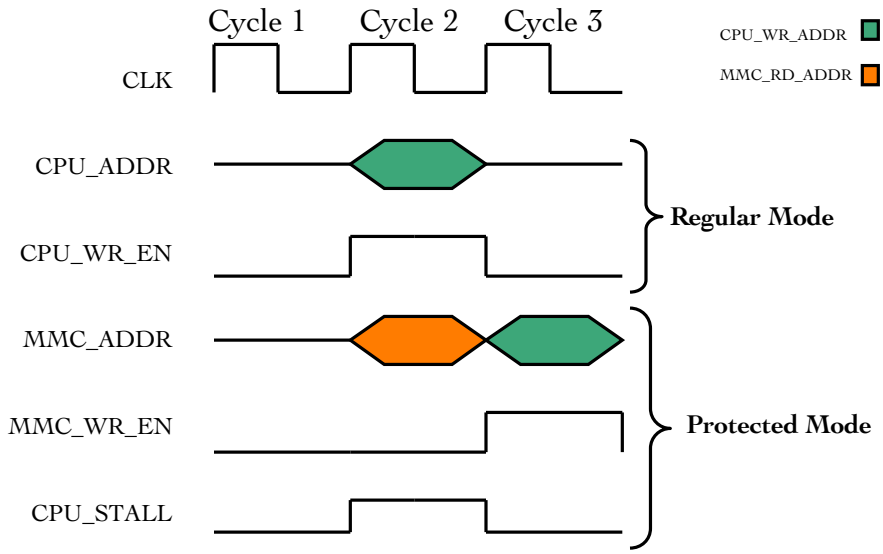
Address → Memory Map Lookup



Block Diagram of Extensions



MMC Store Operation



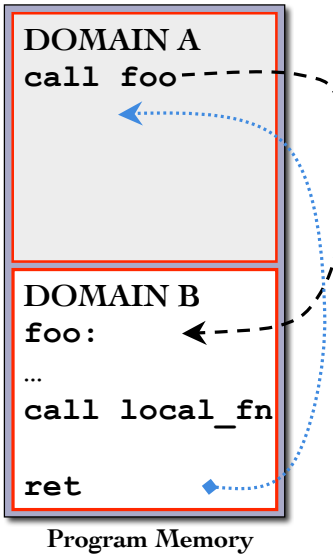
Outline

- Introduction
- **System Components**
  - Protection Domains
  - Memory Map Manager
  - **Control Flow Manager**
- Evaluation

What about Control Flow ?

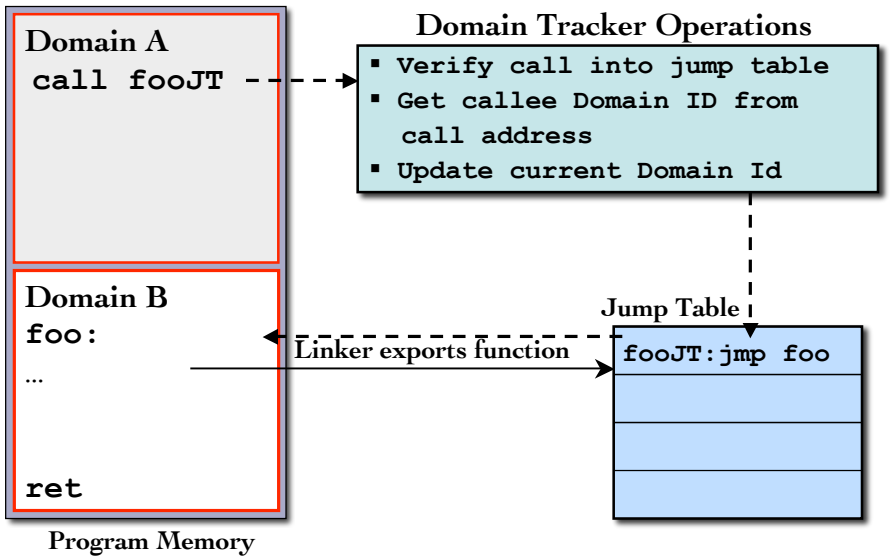
- State within domain can become corrupt
  - Memory map protects one domain from other
- Function pointers in data memory
  - Calls to arbitrary locations in code memory
- Return Address on Stack
  - Single stack for entire system
  - Returns to arbitrary locations in code memory

Control Flow Manager



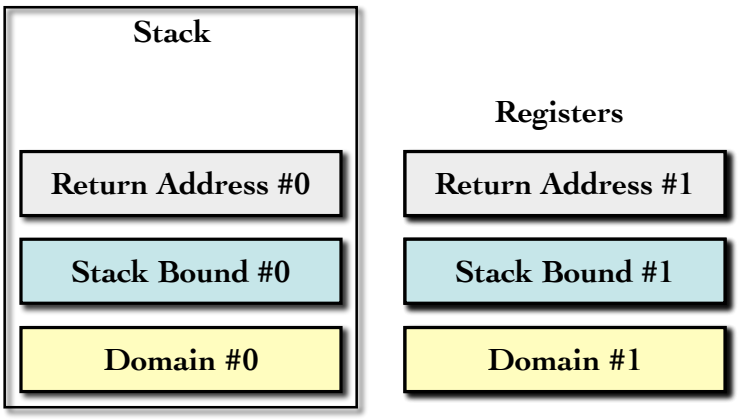
- Ensure control flow integrity
  - Control flow enters domain at designated entry point
  - Control flow leaves domain to correct return address
- Track current active domain
  - Required for memmap checker
- Require Binary Modularity
  - Program memory is partitioned
  - Only one domain per partition

Domain Tracker

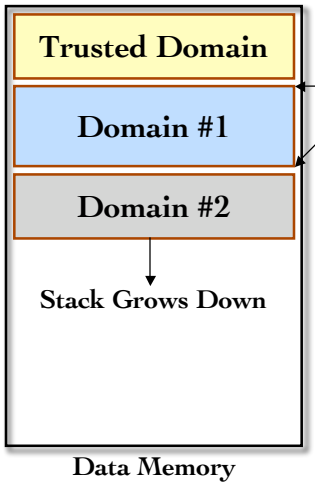


Cross Domain Call Operation

Stack required for chaining cross domain calls



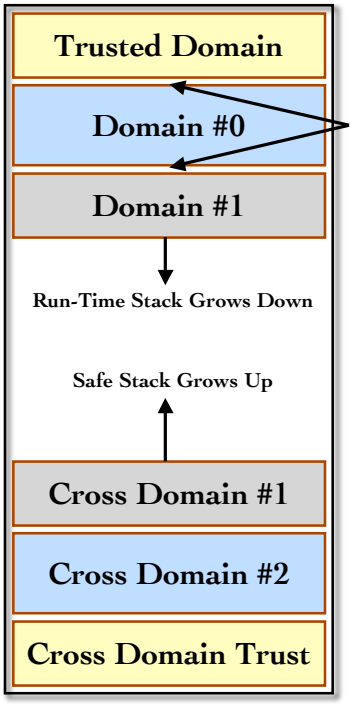
Stack Protection



Single stack shared by all domains

Stack Bounds

- Stack bound set at cross domain calls and returns
- Protection Model
  - No writes beyond latest stack bound
  - Limits corruption to current stack frame
- Enforced by memmap_checker
  - Check all write address



Safe Stack

Stack Bounds

- Maintain callback information for return instruction
- Protect run-time stack from corruption
- Store cross domain call frames

Outline

- Introduction
- System Components
  - Protection Domains
  - Memory Map Manager
  - Control Flow Manager
- **Evaluation**

Hardware Performance: Clock Cycles

Instruction	Normal Execution	Modified Execution
Store	2	3
Call	4	9
Return	4	9

- Modified Store is Protected Store
- Modified Call is Cross Domain Call
- Modified Return is Cross Domain Return

Hardware Performance: Area overhead

HW Component	Ext. Gate Count	Orig. Gate Count
AVR Core	22498	16419
Processor	6783	6685
MMC	2284	N/A
Safe Stack	1749	N/A
Domain Tracker	541	N/A

Software Performance: Clock cycles

Function name	Normal	Protected
Malloc	343	610
Free	138	425
Change_own	55	365

Software Performance: Memory utilization

SW Component	PROM (B)	RAM (B)
Dynamic Memory	1204	2054
Memory Map	422	256
Jump Table	2048	0

Conclusion

- Hw-Sw Co-Desgin for Memory Protection
  - Enabling technology for reliable embedded software systems
  - Combine flexibility of software with efficiency of hardware
- Building blocks for memory protection
  - Memory map manager
  - Control flow manager
- Practical system with widespread applications
  - Low resource utilization
  - Minimal performance overhead
  - Binary compatible with existing software and tool-chains
  - Useful for embedded operating systems

Thank You !
https://projects.nesl.ucla.edu/projects/umpu

NESL

Electrical Engineering Research Review
February 2007