Compiler-assisted Code Randomization

Hyungjoon Koo Yaohui Chen Long Lu
Vasileios P. Kemerlis Michalis Polychronakis

Stony Brook University
Northeastern University
BROWN

Introduction

❖ The need for fine-grained code randomization

- Code reuse/ROP has been the de facto exploitation technique after the introduction of W^X memory protections
- ASLR provides *insufficient* mitigation
  - ✓ Defeated by information leaks
  - ✓ Fixed relative distances between functions and basic blocks
- **Code randomization** makes gadget locations unpredictable
- The advanced JIT-ROP exploitation technique can bypass fine-grained code randomization
  - ✓ Recent execute-only memory (XOM) protections prevent JIT-ROP
  - ✓ **XOM *relies* on fine-grained code randomization to be effective**

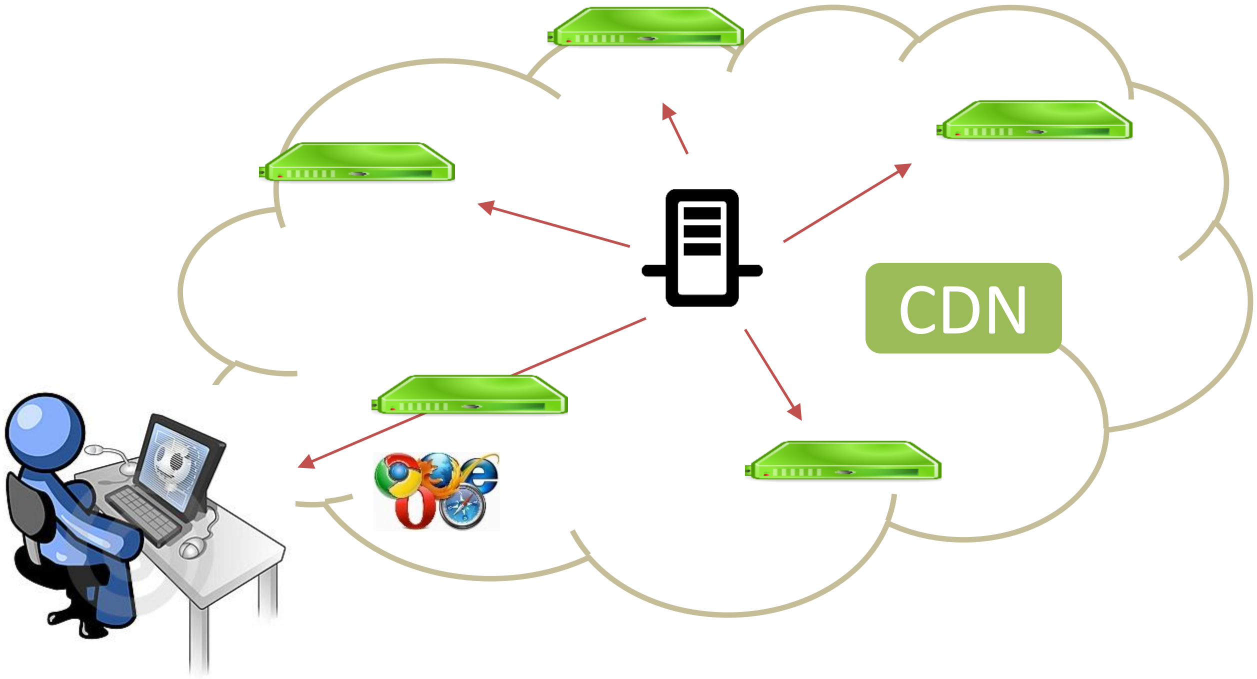
Motivation

- Despite decades of research, code randomization has not seen widespread adoption
  - Diversification by *end users*
    - ✓ Source code level: recompilation
    - ✓ Binary level: static/dynamic binary rewriting
    - ✓ In both cases, the burden is placed on *end users*: responsible for carrying out a complex and cumbersome process
  - Diversification by *software vendors*
    - ✓ Appstores could deliver a randomized variant to each user
    - ✓ Increased cost for generating (compute power) and distributing (no caching/CDNs) randomized copies

Motivation: Key Factors (1/3)

❖ Key factors for making code randomization practical

Transparency	**Software distribution and installation should remain the same**



Motivation: Key Factors (2/3)

- Key factors for making code randomization practical

Transparency	Software distribution and installation should remain the same
Reliability	**Binary rewriting requires ultimate precision**

Static Rewriting
- **Correctness (e.g., indirect transfers)**
- **Incomplete code coverage**

Dynamic Rewriting
- **Performance degradation**
- **Compatibility issues**

Motivation: Key Factors (3/3)

❖ Key factors for making code randomization practical

Transparency	Software distribution and installation should remain the same
Reliability	Binary rewriting requires ultimate precision
Compatibility	**Randomized binaries should remain fully functional**

Software operations
- *Crash reporting*
- *Code signing*
- Patching
- Updating

Code constructs
- Shared object
- Exception handling
- Lazy binding
- Full/partial RELRO
- Compiler optimizations
- Linking time optimizations
- Position independent code
- Control flow integrity

Prior Works (1/2)

❖ Comparison

Research	Needed Information
Efficient Techniques for Comprehensive Protection (USENIX '05)	Source code
G-Free (ACSAC '10)	Source code
ILR (Oakland '12)	Disassembly
Orp: smashing gadgets (Oakland '12)	Disassembly
Binary Stirring (CCS '12)	Disassembly
XIFER: gadge me (CCS '13)	Disassembly, Relocation
Oxymoron (USENIX '14)	Disassembly
Readactor (Oakland '15)	Source code
Shuffler (OSDI '16)	Symbol, Relocation
Selfrando (PETS '16)*	Relocation, Function boundary

Prior Works (2/2)

❖ SoK: Automated software diversity (Oakland '14)

*"Naturally, the research in software diversity can be extended; we point out several promising directions. There is currently a **lack of research on hybrid approaches** combining aspects of compilation and binary rewriting to address practical challenges of current techniques."*

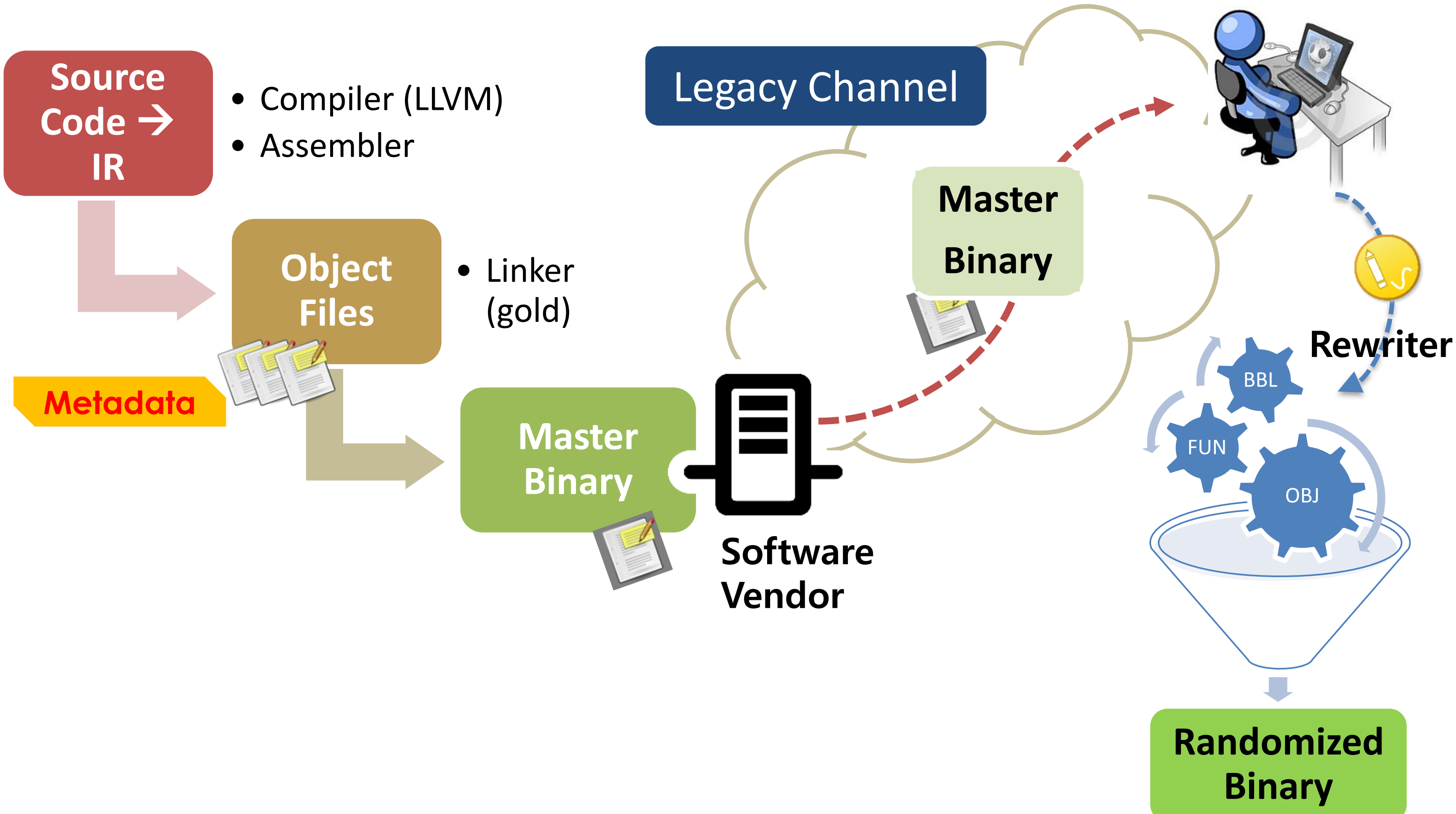
Research Question

- Can we achieve the following *goal*?

Reliably randomize binaries
in a ***transparent*** way,
compatible with existing software

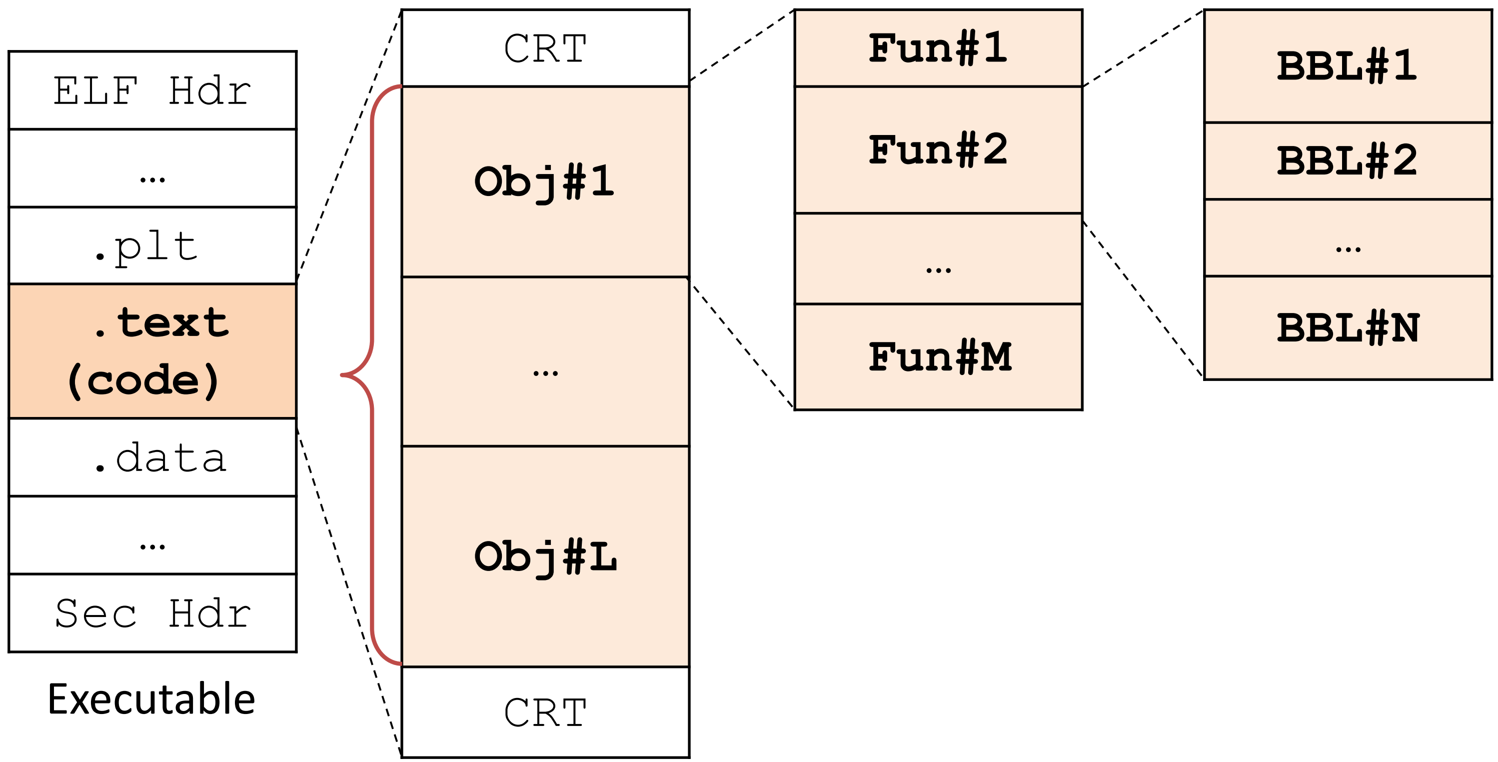
Overview: Compiler-assisted Code Randomization

❖ Compiler-rewriter cooperation



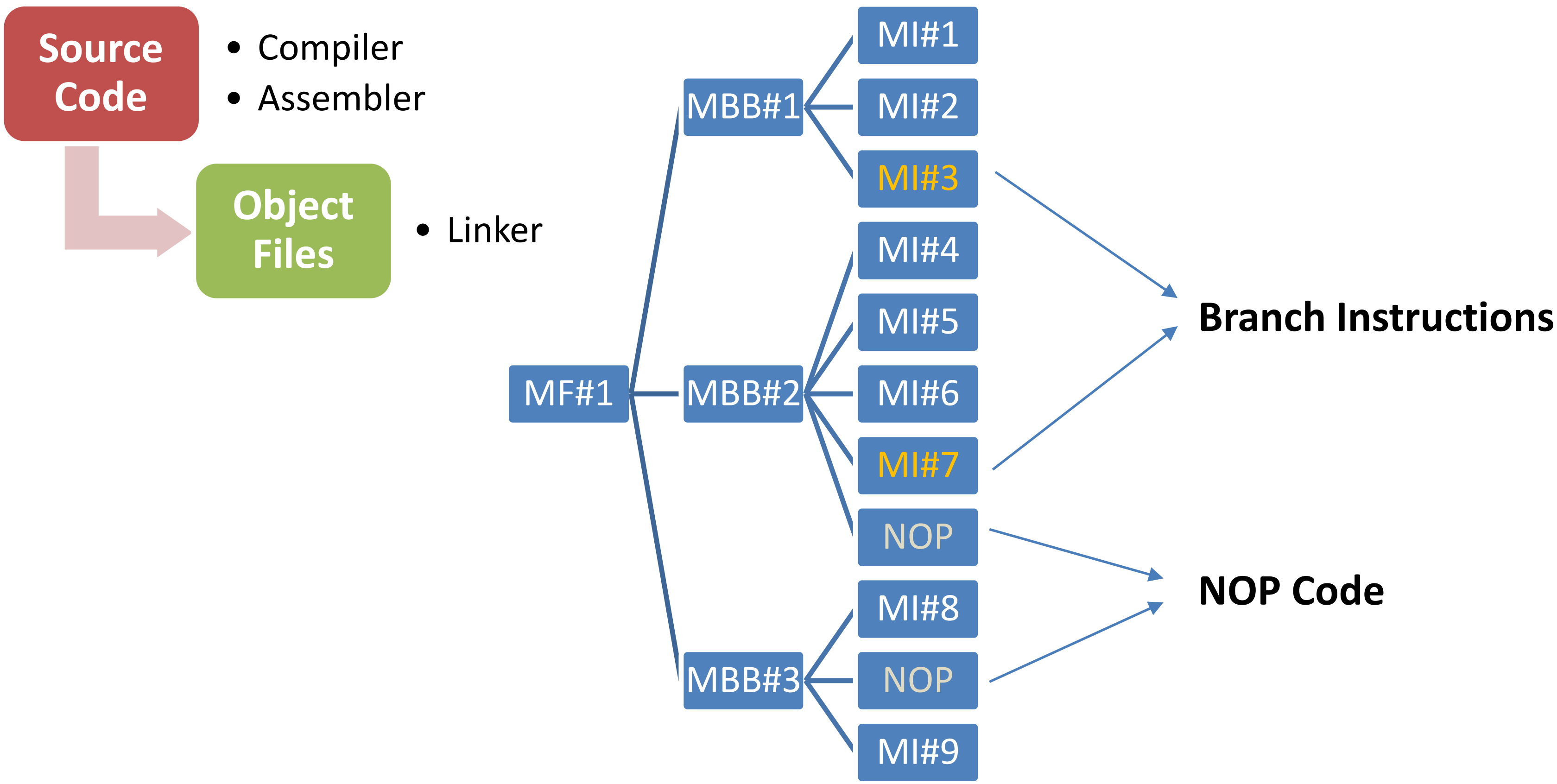
Transformation-assisting Metadata

- Precise object boundaries for transformation



Transformation-assisting Metadata: Code Generation in LLVM Backend (1/2)

❖ *MC Framework* uses an internal hierarchical structure:
Machine Function (MF), Machine Basic Block (MBB), Machine Instruction (MI)

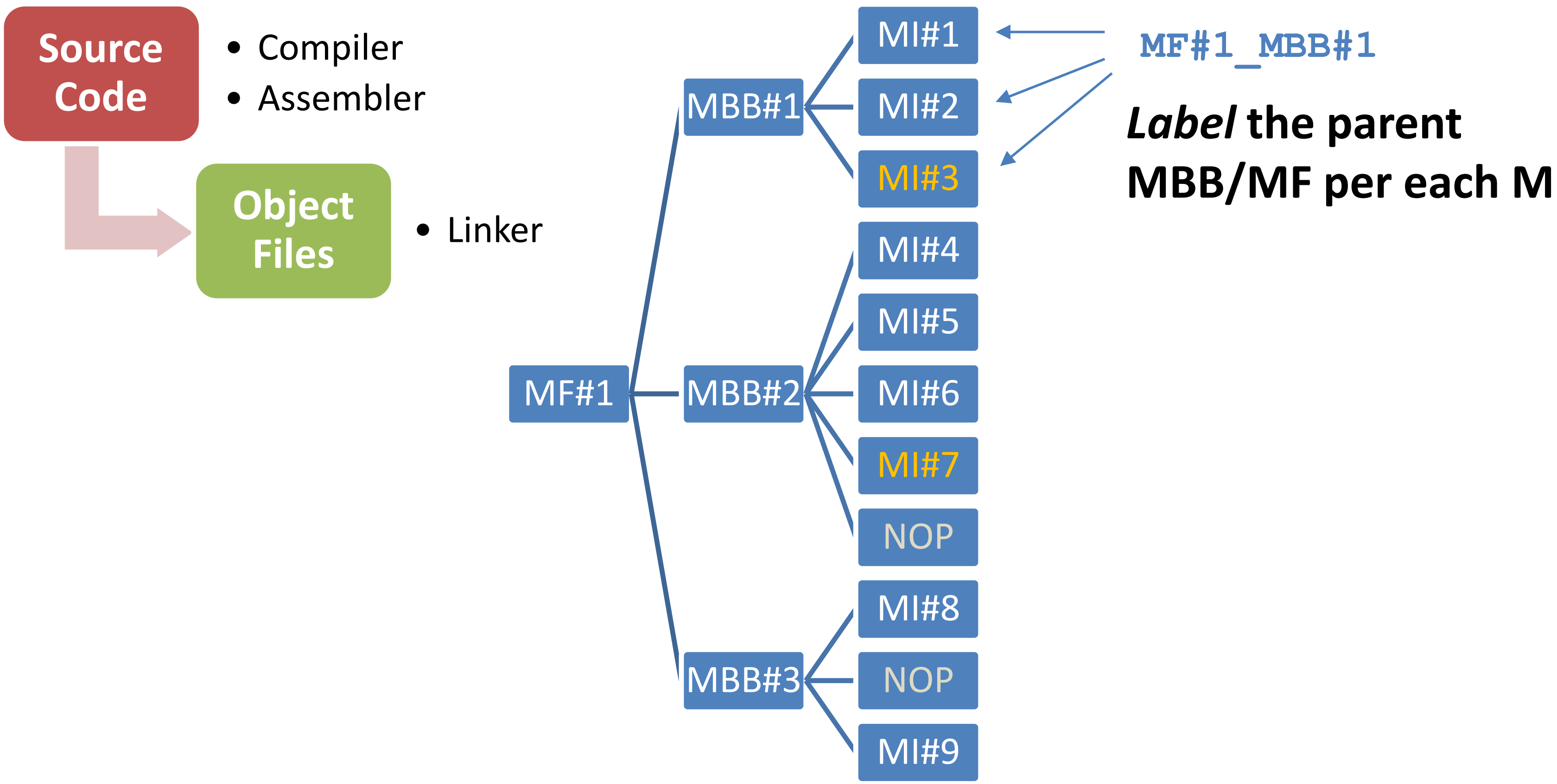


Transformation-assisting Metadata: Code Generation in LLVM Backend (2/2)

❖ *MCAssembler* treats code as a series of fragments:

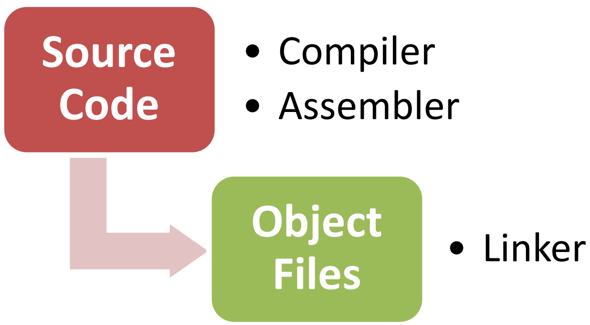
Data Fragment (DF), Relaxable Fragment (RF), Alignment Fragment (AF)

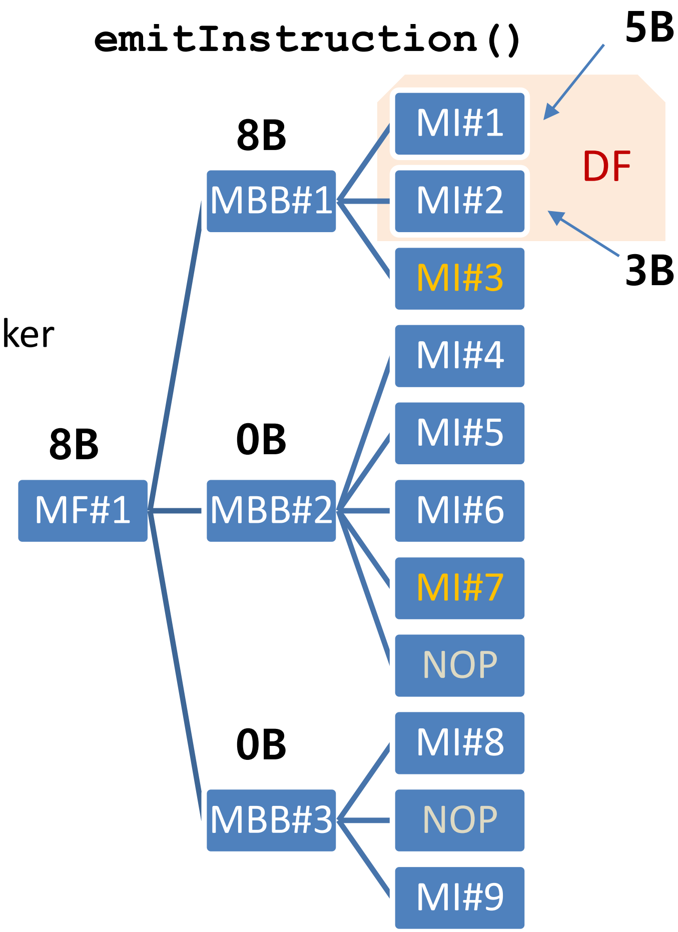
- No *high-level structure (MF or MBB)*



Transformation-assisting Metadata: Tracking Emitted Bytes in the Final Layout (1/3)

- *MCAssembler* treats code as a series of fragments
  - As layout is being determined, both MBB/MF sizes are decided.

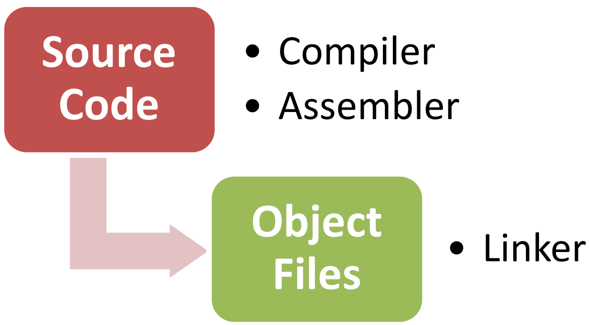


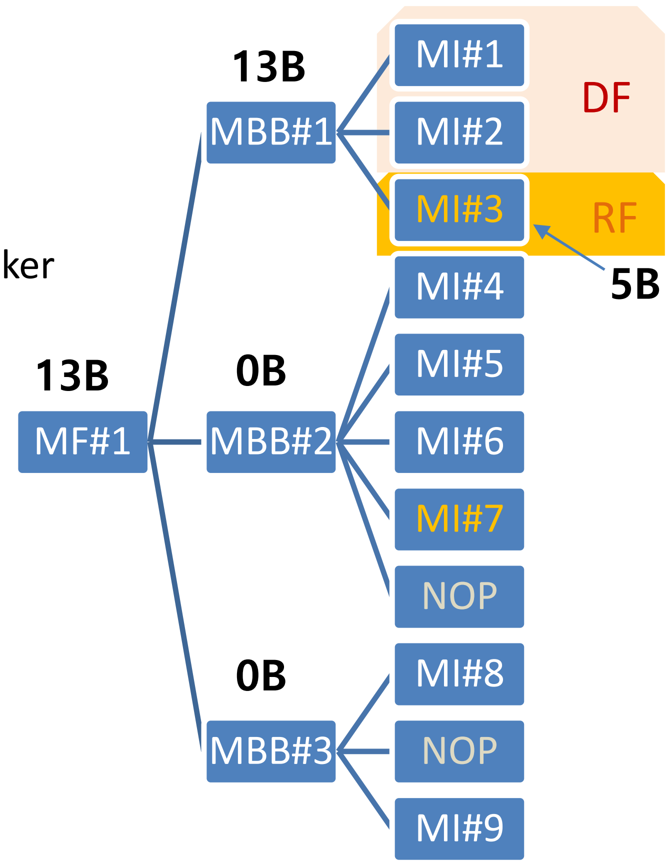


<u>Fragment Types</u>
- DF: Data
- RF: Relaxable
- AF: Alignment

Transformation-assisting Metadata: Tracking Emitted Bytes in the Final Layout (2/3)

- *MCAssembler* treats code as a series of fragments
  - Branch instructions form relaxable fragments (RF).

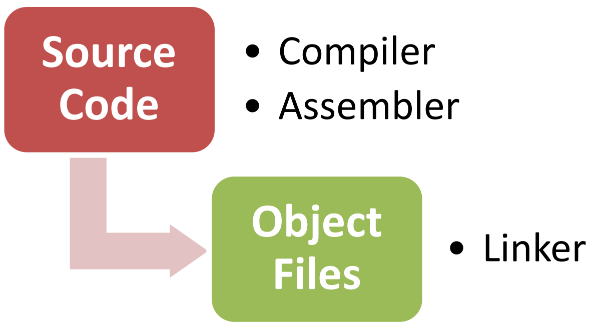


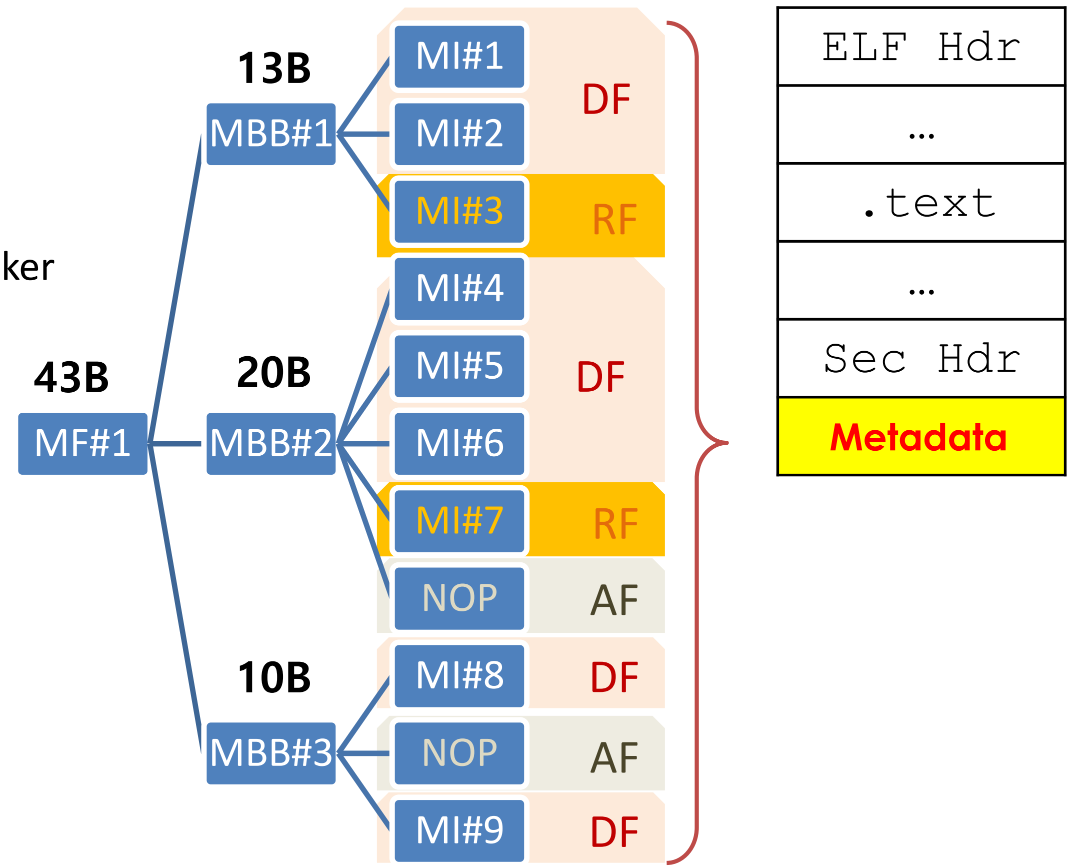


Fragment Types
- DF: Data
- RF: Relaxable
- AF: Alignment

Transformation-assisting Metadata: Tracking Emitted Bytes in the Final Layout (3/3)

- *MCAssembler* treats code as a series of fragments
  - NOP byte(s) are counted as part of MBB or MF in size.





Fragment Types

- DF: Data
- RF: Relaxable
- AF: Alignment

Transformation-assisting Metadata: Fixup Information (1/2)

❖ Fixup information can be resolved:

- At compilation time → *MISSING*
- At link time → relocations in object files
- At load time → relocations in final executable

Byte Code	Instructions	Byte Code
48 89 DF	mov rdi, rbx	48 89 DF
4C 89 F6	mov rsi, r14	4C 89 F6
E8 ***49 43 00 00***	***call someFunc*** ❷	E8 ***8D 30 06 00***
EB ***0D***	***jmp short 0xD*** ❶	EB ***0D***
49 39 1C 24	cmp [mh],ctrl	49 39 1C 24
…	…	…
48 83 C4 08	add rsp, 8	48 83 C4 08

Object File **Final Executable**

Relocation Table for Object File

TYPE	VALUE
R_X86_64_PC32	someFunc-0x4 ❷
...	

Transformation-assisting Metadata: Fixup Information (2/2)

❖ Fixup information relationships

- Set A = {Fixups resolved at compilation time}
- Set B = {Fixups resolved at link time}
- Set C = {Fixups resolved at load time}

A ⊃ B ⊃ C

✓ **Offset from section base**
✓ **Dereferencing size**
✓ **Value is absolute or relative**

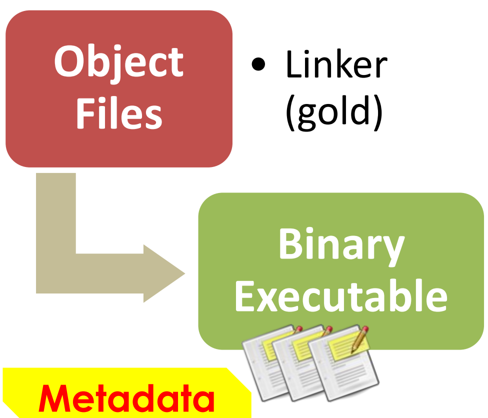
Metadata Summary

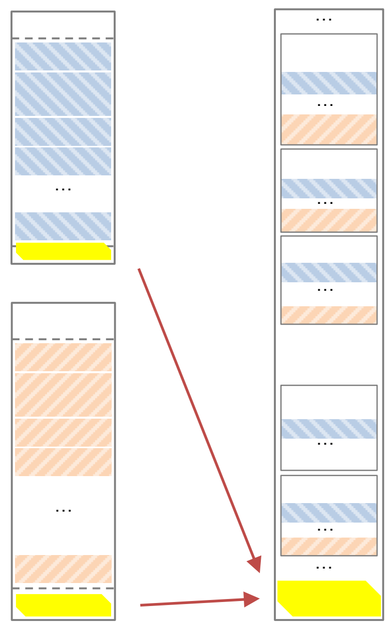
Metadata	Collected Information	Collection time
Layout	Section offset to first object Section offset to `main()` Total code size for randomization	Linking Linking Linking
Basic Block (BBL)	BBL size (in bytes) BBL boundary type (BBL, FUN, OBJ) Fall-through or not Section name that BBL belongs to	Linking Compilation Compilation Compilation
Fixup	Offset from section base Dereference size Absolute or relative Type (c2c, c2d, d2c, d2d) Section name that fixup belongs to	Linking Compilation Compilation Linking Compilation
Jump Table	Size of each jump table entry Number of jump table entries	Compilation Compilation

Metadata Consolidation at Link Time

❖ Linker consolidates per-object metadata

- Constructing the final layout
- Resolving symbols
- Updating relocation information
- ***Merging/adjusting collected metadata from each object file***

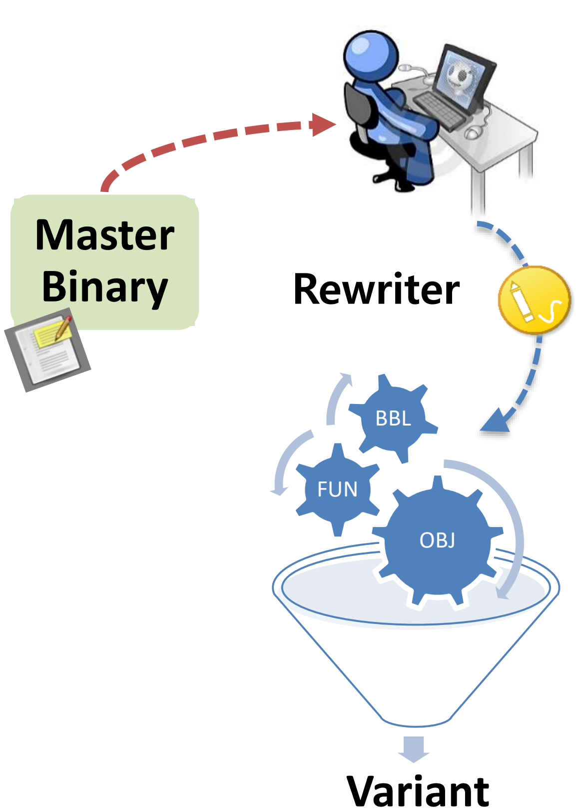


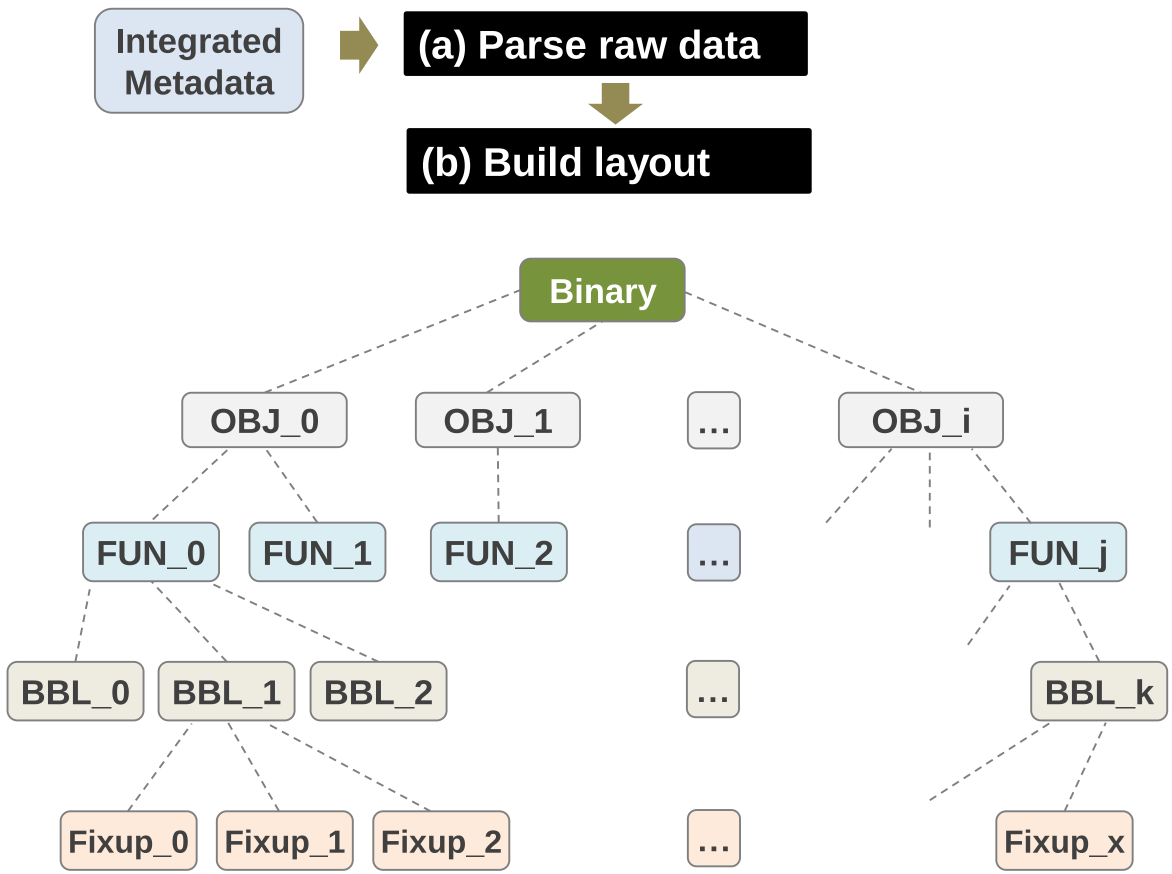


✓ **Layout**
✓ **BBL size**
✓ **Fixup Offset**

Client-side Randomization (1/2)

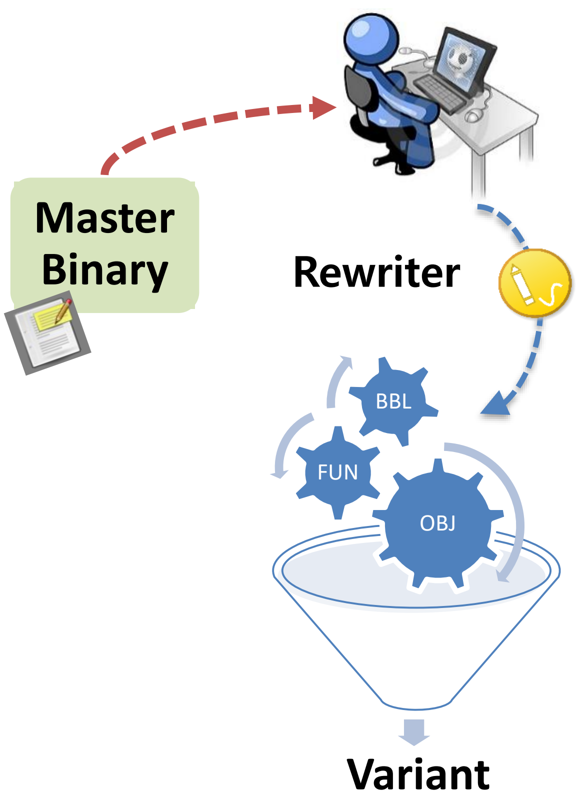
- Binary rewriting at installation time





Client-side Randomization (2/2)

❖ Binary rewriting at installation time



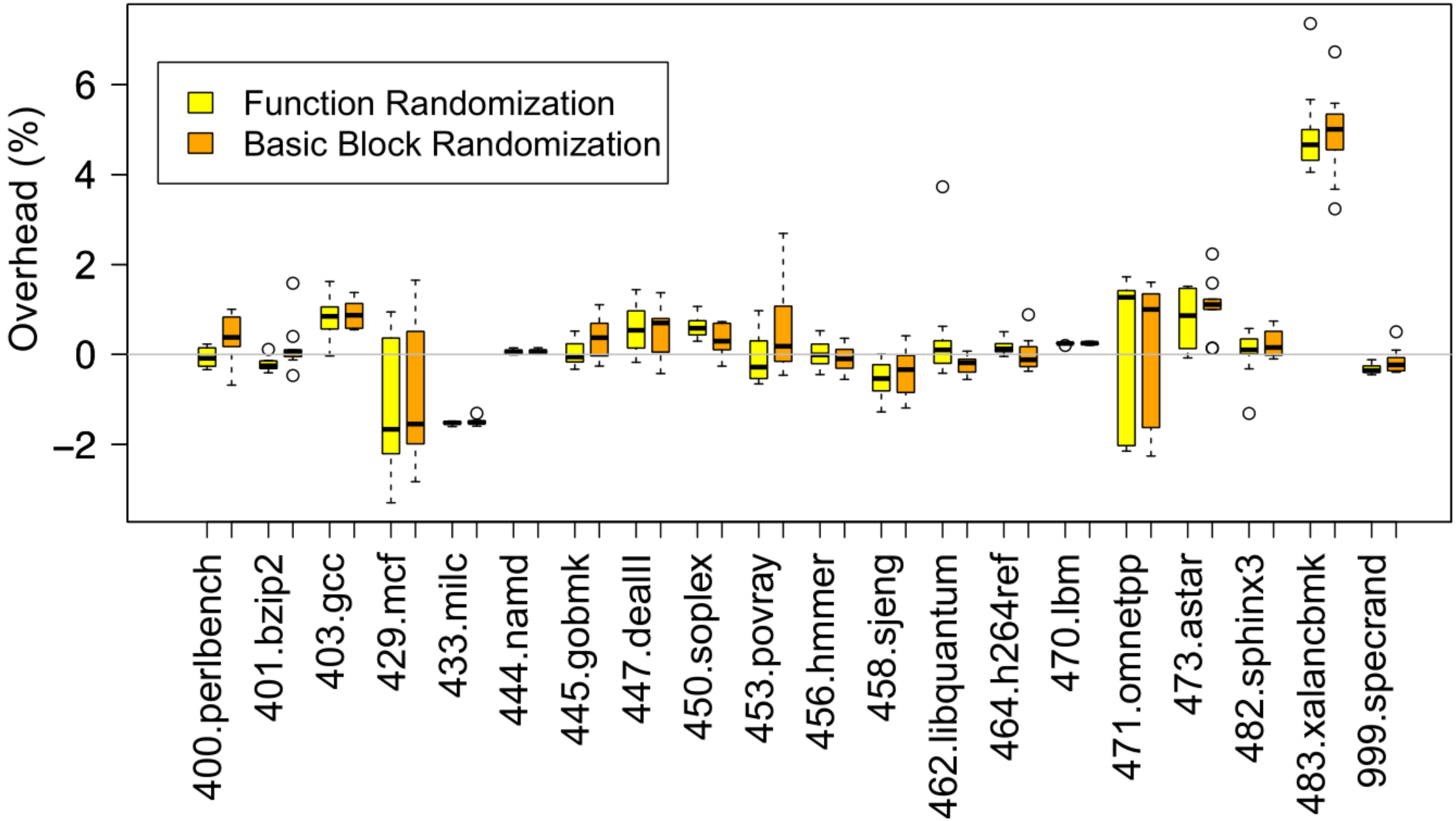
(c) Perform rand.

(d) Rewrite binary → **Output**

.text .data .rodata .data.rel.ro .init_array	Fixup Info Rewriting
.rela.dyn .dynsym (.symtab)	Symbol Info Rewriting
.eh_frame .eh_frame_hdr	Exception Handling Info Rewriting

Evaluation (SPEC2006)

- 0.28% runtime overhead on avg., 11.5% inc. in file size



What we have not talked about

❖ Challenges for enabling robust/practical transformation

- How to handle jump table entries
- Support for various software constructs
  - ✓ Exception handling
  - ✓ Inline assembly
  - ✓ LTO (Linking time optimization)
  - ✓ CFI (Control flow integrity)
- Randomization constraints
- Optimized metadata serialization
- Implementation pitfalls and current limitations of CCR

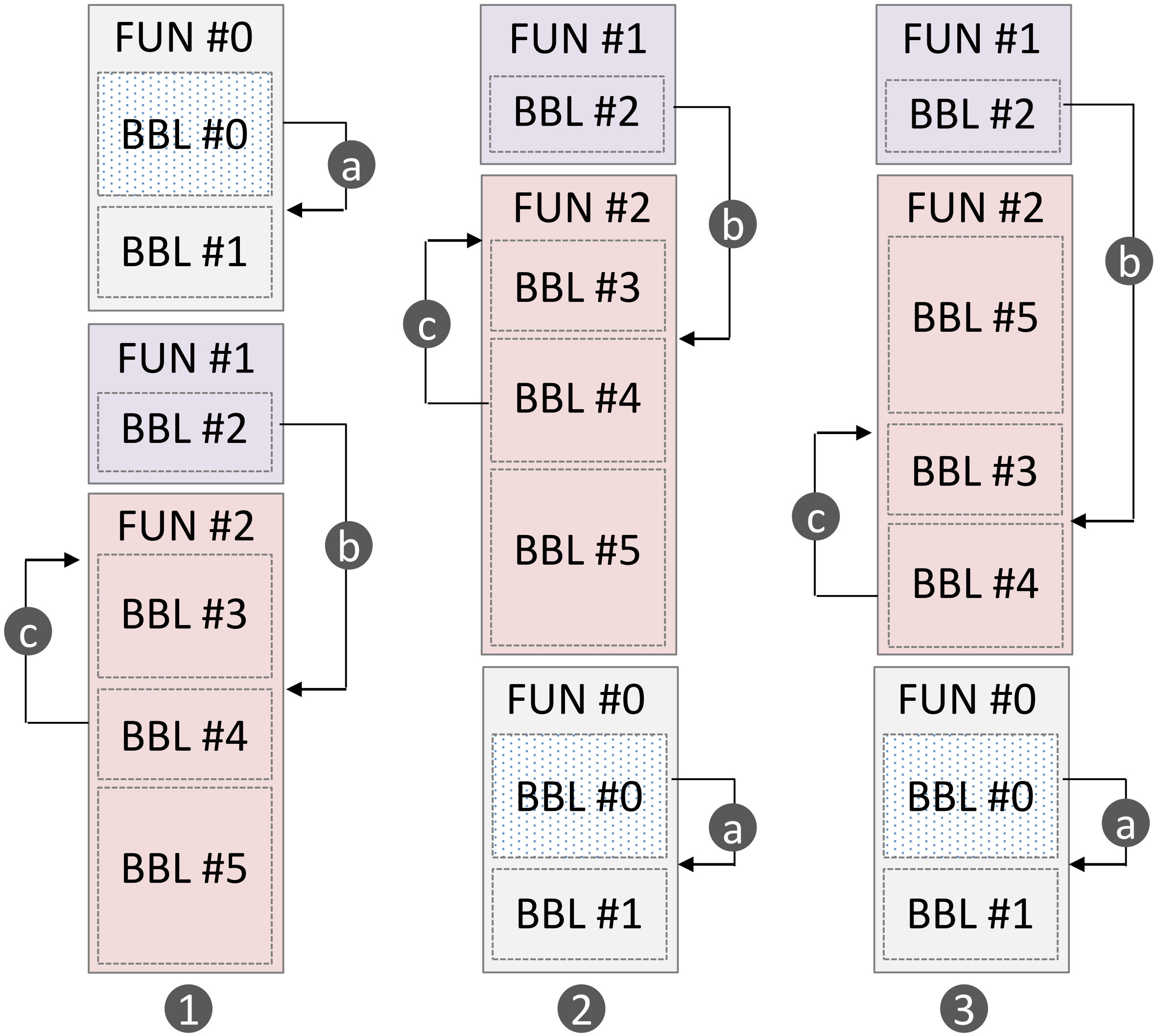
Please read our paper!

Wrap-up

- ***C**ompiler-assisted **C**ode **R**andomization*
  - Function and *basic block* level permutation
  - Facilitated by *transformation-assisting metadata* stored within augmented executables
  - **Transparency, reliability, and compatibility**
  - Integration with Apt package manager

Open-source prototype:
`https://github.com/kevinkoo001/CCR`

Backup: Randomization Constraints



Backup: Jump Table Entry and Metadata

❖ Size of each entry and the # of entries in jump table

Section Name	Compiled *without* PIC/PIE		Compiled *with* PIC/PIE	
	Byte Code	Disassembly	Byte Code	Disassembly
.text	***FF 24 D5 A0 39 4A 00***	jmp qword [rdx*8+0x4A39A0]	***48 8D 05 5E 84 09 00*** 48 63 0C 90 48 01 C1 FF E1 …	lea rax, [rel 0x98465] movsxd rcx, dword [rax+rdx*4] add rcx, rax jmp rcx …
		Code for JTE #1 Code for JTE #0		Code for JTE #1* Code for JTE #0*
.rodata	D2 C0 40 00 00 00 00 00	JT Entry #0(8B) 0x0040C0D2	AB 7B F6 FF	JT Entry #0*(4B) 0xFFF67BAB
	D8 C0 40 00 00 00 00 00 …	JT Entry #1(8B) 0x0040C0D8 …	B1 7B F6 FF …	JT Entry #1*(4B) 0xFFF67BB1 …

Backup: Exception Handling