The Effect of Instruction Padding on SFI Overhead

Navid Emamdoost
navid@cs.umn.edu

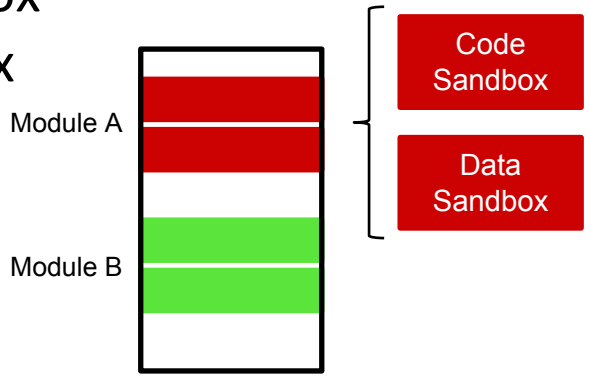
Stephen McCamant
mccamant@cs.umn.edu

UNIVERSITY OF MINNESOTA
Driven to Discover℠

Software-based Fault Isolation

UNIVERSITY OF MINNESOTA

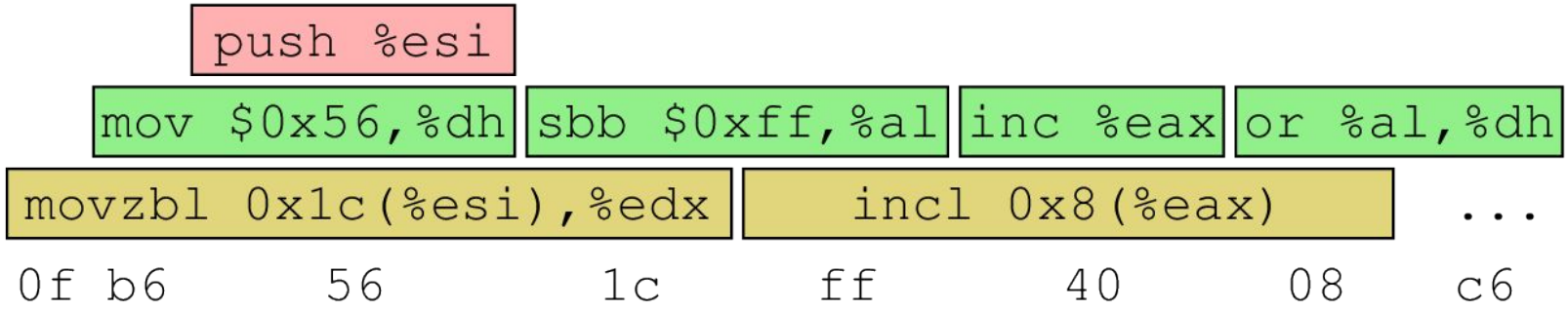
- Software-based isolation
  - Incorporate an untrusted module
  - All modules in same virtual address
  - Provide efficient communication
- Security Policy:
  - No code is executed outside the sandbox
  - No data is changed outside the sandbox



CISC Architectures

UNIVERSITY OF MINNESOTA

- Variable length instructions
- Processor can jump to any byte
- Hard to make hidden instructions safe
- Solution: Instruction Alignment



Our Focus

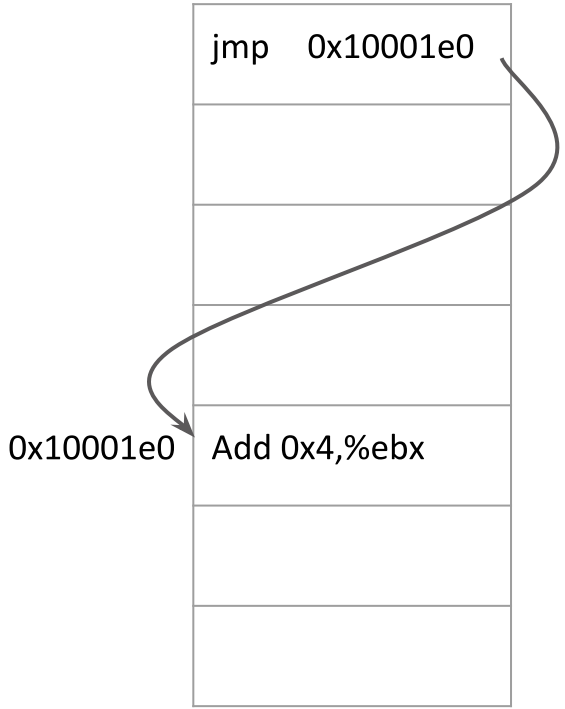
UNIVERSITY OF MINNESOTA

- Improving current isolation mechanism performance
  - Guaranteeing same level of security
- Native Client
  - Software-based Fault Isolation implementation for CISC architectures
  - Instruction padding to enforce address layout invariant
  - Padding imposes runtime overhead
  - Changed padding scheme
  - Updated validator

Unsafe Instructions

University of Minnesota

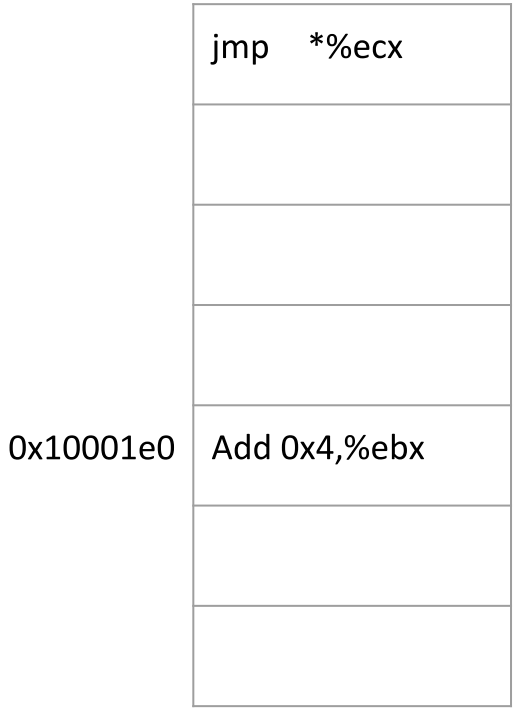
- Jump or store instructions
  - Change Control flow
  - Change data
- Addressing issue
  - jmp 0x10001e0
  - mov %eax,0x11020028



Unsafe Instructions

University of Minnesota

- Jump or store instructions
  - Change Control flow
  - Change data
- Addressing issue
  - jmp 0x10001e0
  - jmp *%ecx
  - mov %eax,0x11020028
  - mov $0x1b80,(%ecx)



Alignment in NaCl

UNIVERSITY OF MINNESOTA

- Divide memory into 32-byte *Bundles*
- Target of indirect jump is placed at the beginning of bundles
- No instruction is allowed to cross bundle boundary

```
                                ...
1060451:      83 c8 01              or      $0x1,%eax
1060454:      88 41 28              mov     %al,0x28(%ecx)
1060457:      8b 0e                 mov     (%esi),%ecx
1060459:      8b 51 04              mov     0x4(%ecx),%edx
106045c:      85 d2                 test    %edx,%edx
106045e:      66 90                 xchg    %ax,%ax

1060460:      0f 88 9a 01 00 00     js      1060600
1060466:      8b 01                 mov     (%ecx),%eax
1060468:      8d 3c 95 00 00 00 00  lea     0x0(,%edx,4),%edi
106046f:      89 d6                 mov     %edx,%esi
1060471:      83 ea 01              sub     $0x1,%edx
1060474:      83 e6 07              and     $0x7,%esi
1060477:      83 fa ff              cmp     $0xffffffff,%edx
106047a:      8d b6 00 00 00 00     lea     0x0(%esi),%esi

1060480:      c7 04 38 84 06 03 11  movl    $0x11030684,(%eax,%edi,1)
1060487:      8d 47 fc              lea     -0x4(%edi),%eax
106048a:      0f 84 70 01 00 00     je      1060600
                                ...
```

Alignment in NaCl

UNIVERSITY OF MINNESOTA

- Divide memory into 32-byte *Bundles*
- Call instructions placed at the end of bundles
  - For the sake of return address alignment

```
10ee700:      55                      push    %ebp
10ee701:      89 e5                   mov     %esp,%ebp
10ee703:      83 ec 18                sub     $0x18,%esp
10ee706:      c7 04 24 2c 5e 01 11    movl    $0x11015e2c,(%esp)
10ee70d:      8d b4 26 00 00 00 00    lea     0x0(%esi,%eiz,1),%esi
10ee714:      8d bc 27 00 00 00 00    lea     0x0(%edi,%eiz,1),%edi
10ee71b:      e8 a0 72 f5 ff          call    10459c0
10ee720:      89 ec                   mov     %ebp,%esp
10ee722:      5d                      pop     %ebp
10ee723:      59                      pop     %ecx
10ee724:      83 e1 e0                and     $0xffffffe0,%ecx
10ee727:      ff e1                   jmp     *%ecx
                                      …
```

Padding vs Performance

- Inserting artificial alignments
  - Necessary for protecting against hidden instructions
  - Extra instructions (NOP)
  - Hurt CPU cache and pre-fetch
- Impose performance overhead

CBI NaCl

- Change NaCl padding scheme
  - Less NOPs
  - To decrease performance overhead

- Multipass Validator
  - We must guarantee sandboxing policy enforcement
  - Appropriate changes in validator

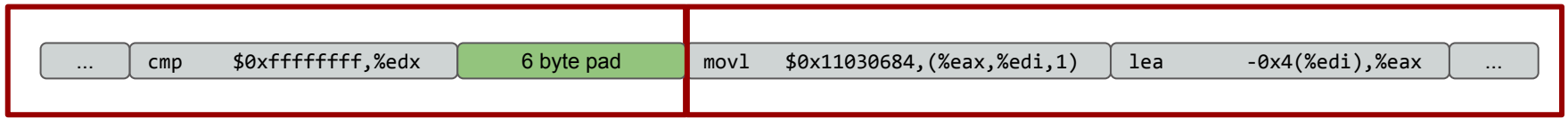
Types of Padding

- Indirect jump target
  - Will be placed at the next bundle start
- Call instruction
  - Will be placed at the end of the bundle
- Cross bundle instruction
  - Will be pushed to the start of next bundle

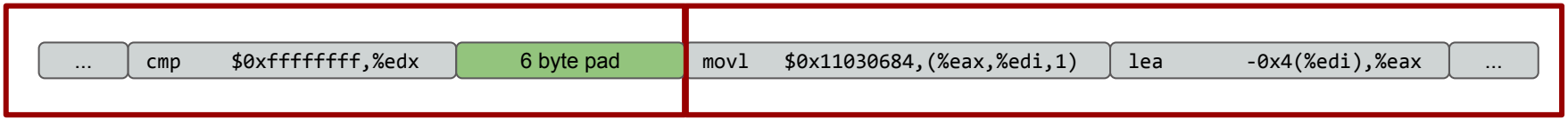
Types of Padding

UNIVERSITY OF MINNESOTA

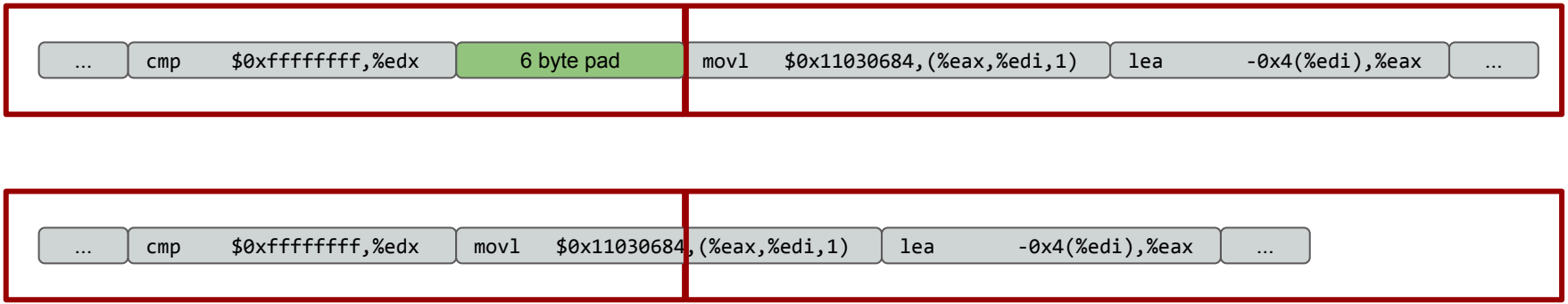
- Indirect jump target
  - Will be placed at the next bundle start
- Call instruction
  - Will be placed at the end of the bundle
- Cross bundle instruction
  - Will be pushed to the start of next bundle
  - Conservative



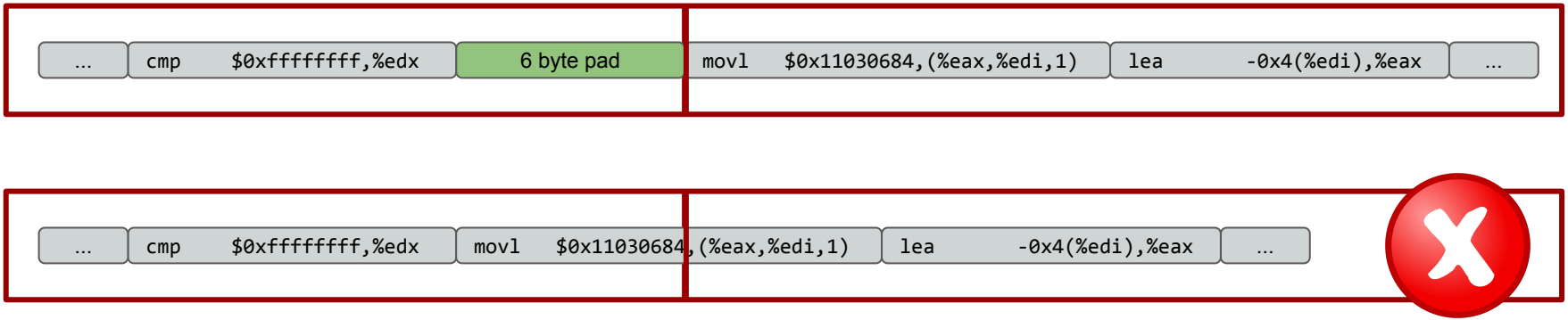
Pad Removal (simplified)



Pad Removal (simplified)

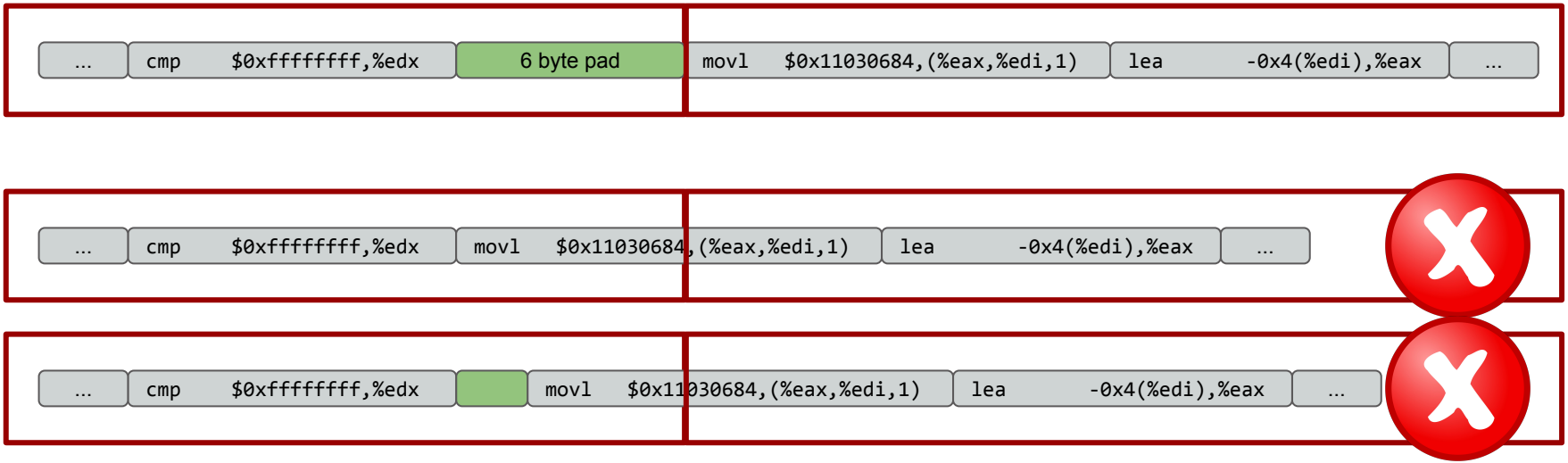


Pad Removal (simplified)

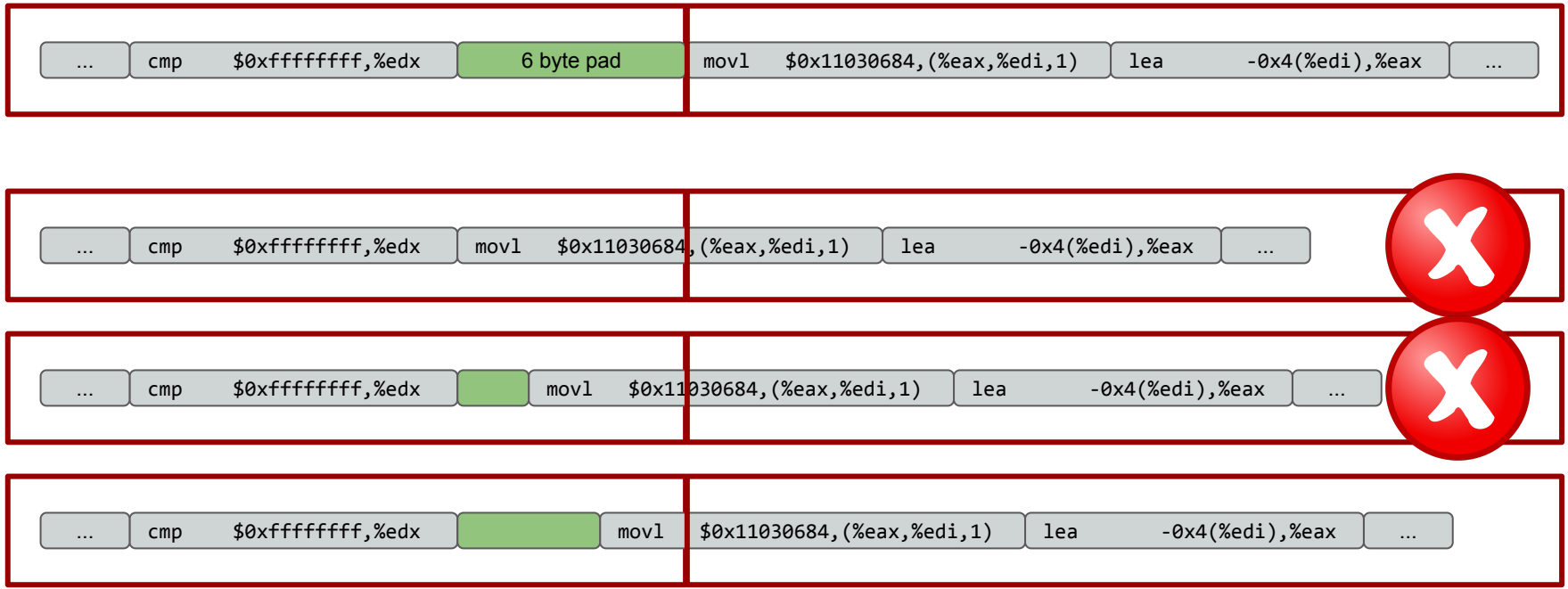


Pad Removal (simplified)

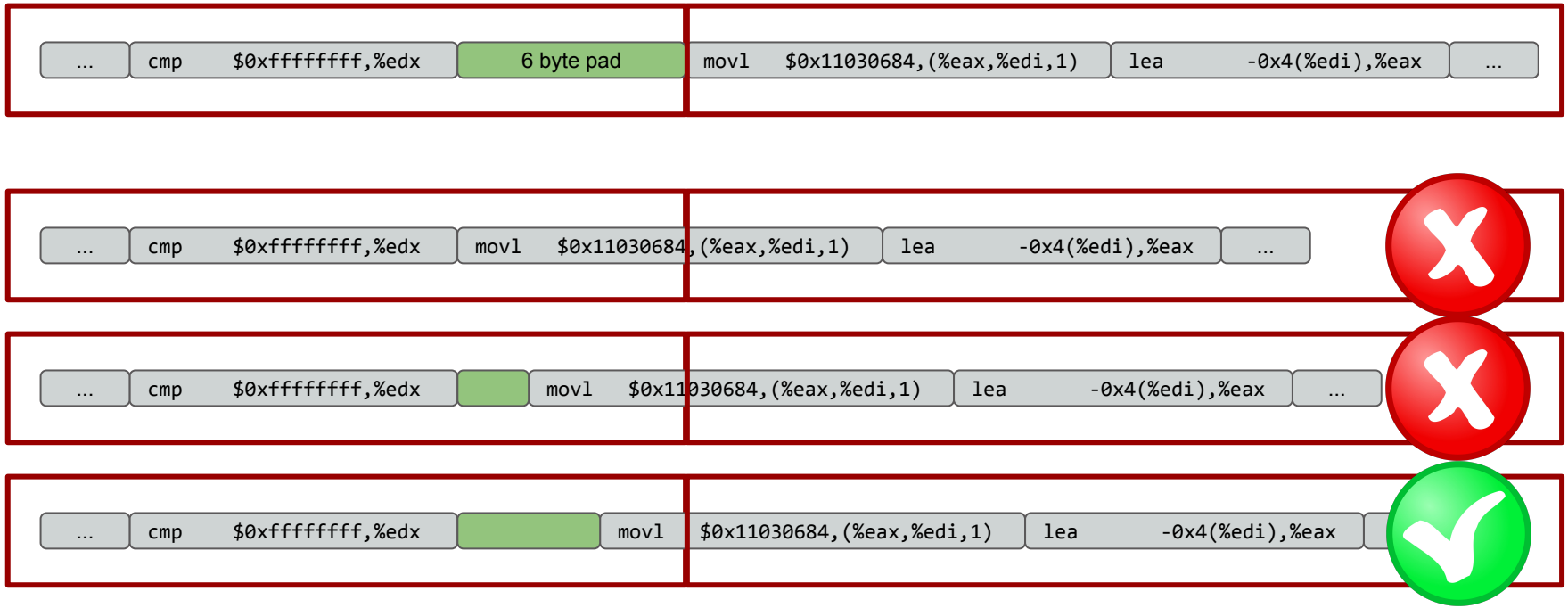
Pad Removal (simplified)



Pad Removal (simplified)



Pad Removal (simplified)



NaCl Validator

- One pass: from the start to the end of code
- At each address checks the instruction
- For indirect branch
  - Checks presence of masking instruction
- For direct branch
  - Check the target is a valid instruction

Multipass Validator

UNIVERSITY OF MINNESOTA

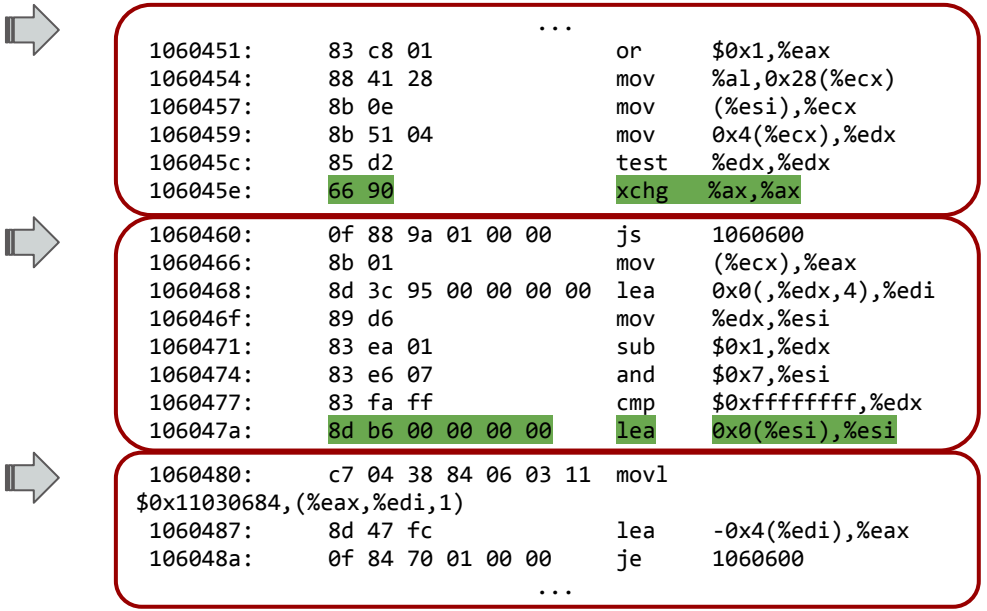
- Challenge: Cross-Bundle Instructions

```
                                ...
 1060451:      83 c8 01              or      $0x1,%eax
 1060454:      88 41 28              mov     %al,0x28(%ecx)
 1060457:      8b 0e                 mov     (%esi),%ecx
 1060459:      8b 51 04              mov     0x4(%ecx),%edx
 106045c:      85 d2                 test    %edx,%edx
 106045e:      66 90                 xchg    %ax,%ax
 1060460:      0f 88 9a 01 00 00     js      1060600
 1060466:      8b 01                 mov     (%ecx),%eax
 1060468:      8d 3c 95 00 00 00 00  lea     0x0(,%edx,4),%edi
 106046f:      89 d6                 mov     %edx,%esi
 1060471:      83 ea 01              sub     $0x1,%edx
 1060474:      83 e6 07              and     $0x7,%esi
 1060477:      83 fa ff              cmp     $0xffffffff,%edx
 106047a:      8d b6 00 00 00 00     lea     0x0(%esi),%esi
 1060480:      c7 04 38 84 06 03 11  movl
$0x11030684,(%eax,%edi,1)
 1060487:      8d 47 fc              lea     -0x4(%edi),%eax
 106048a:      0f 84 70 01 00 00     je      1060600
                                ...
```

Multipass Validator

UNIVERSITY OF MINNESOTA

- Challenge: Cross-Bundle Instructions
- Multipass: start validation from every crossing point
  - Bundle start
  - Proceed to next if reached an already validated instruction



Multipass Validator Correctness

- NaCl sandboxing policy relies on:
  - Segment registers are not changed
  - Bytes in code segment are not altered after validation
- Multipass validator inherits NaCl validator instruction filtering
- Validation from every bundle start ensures all possible instructions are checked

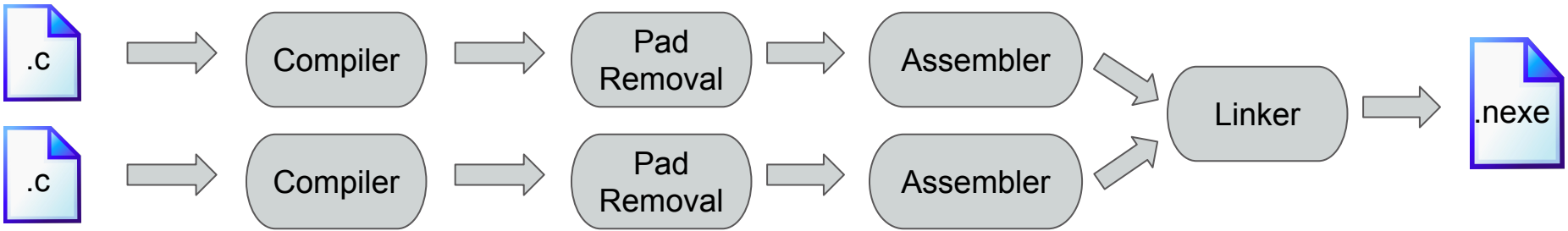
Proof of Correctness

- Providing a formal proof for multipass validator correctness
  - Based on proof provided in RockSalt paper (PLDI'12)
  - Using Coq theorem prover
- RockSalt proof modeled a single-pass validator
- Modification to model multi-pass validator
  - Loops modeled as recursive functions
  - Inserted a recursive function to model outer loop
    - Starts validation from every bundle start
  - Reuse single-pass model as inner loop

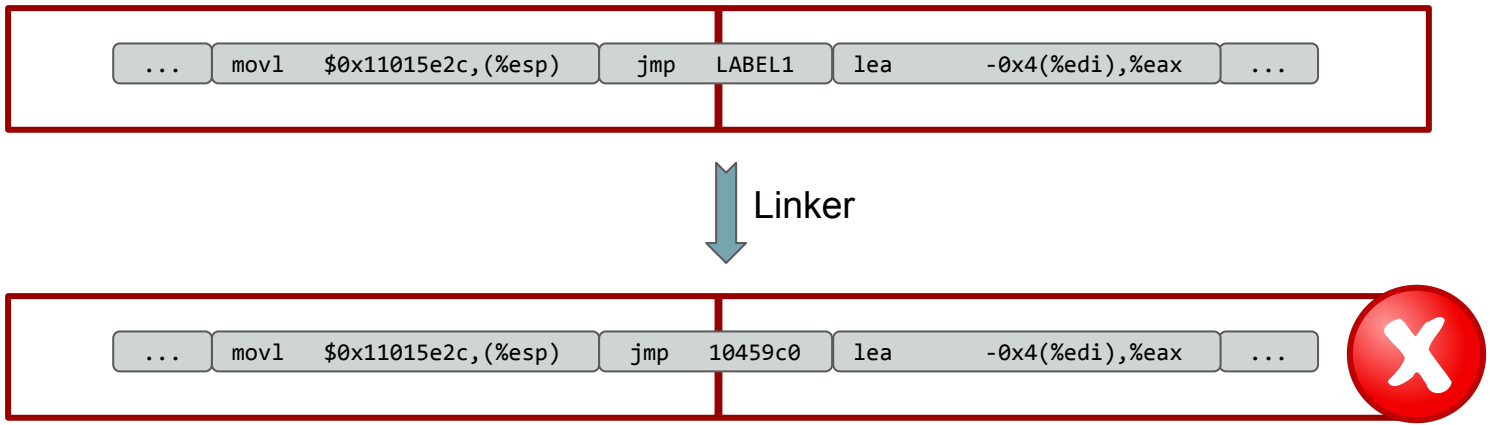
Separate Compilation

UNIVERSITY OF MINNESOTA

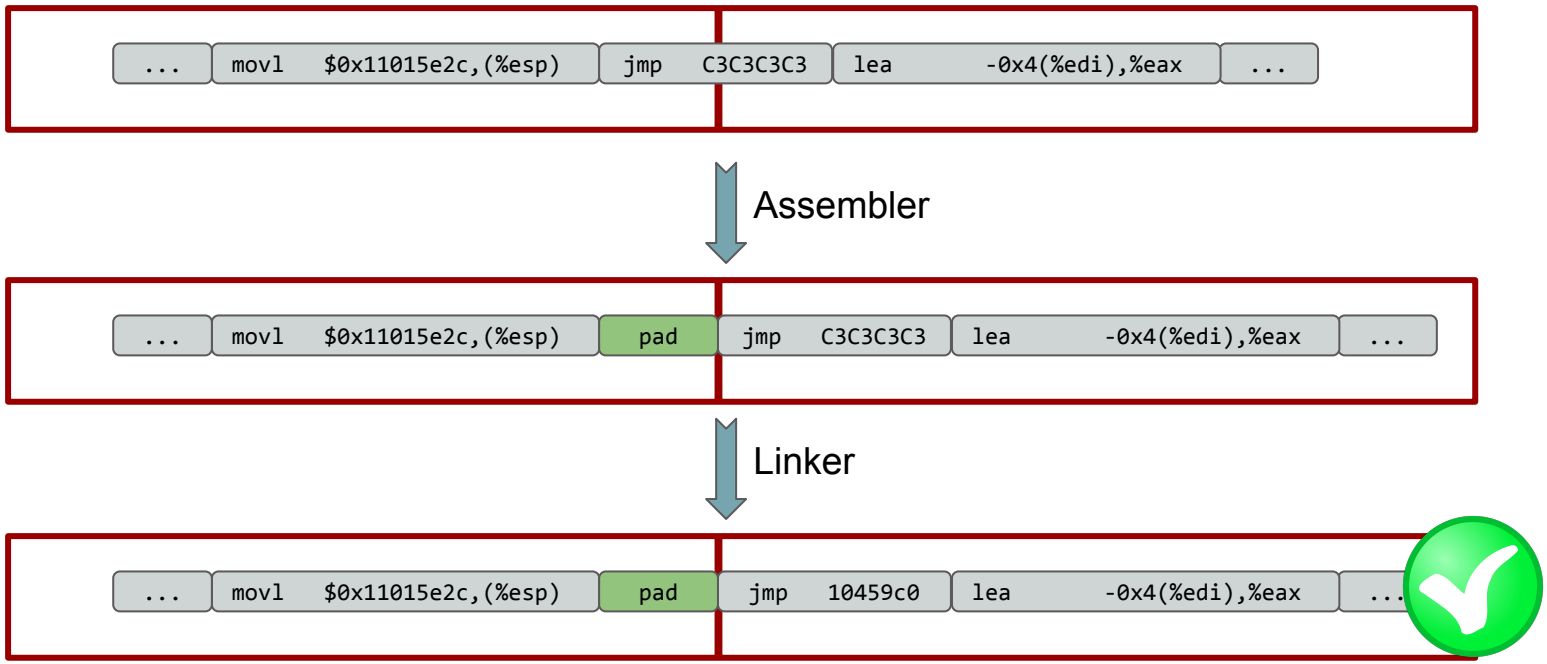
- Process each source file separately
  - Decide about the paddings to be removed
  - Assemble them into object files (using modified *GAS*)
  - Then link them together



Relocations Problem



Relocations Problem



More on compilation

- For some benchmarks *0xc3* filtering was not enough
  - *Vortex* and *Crafty* from SPEC2000
  - *gcc*, *gobmk*, and *h264ref* from SPEC2006
- Reason: special values in relocation
  - Change the interpreted instruction length
- Caused the final binary fail validation
  - Even though with valid intermediate object files

Example

- In nexe file:

```
66 89 81 9f 19 02 11 7f 99 e9 54 71 ff ff
```

- In object file:

```
66 89 81 c3 c3 c3 c3 7f 99 e9 54 71 ff ff
```

Example

UNIVERSITY OF MINNESOTA

- In nexe file:

```
66 89 81 9f 19 02 11 7f 99 e9 54 71 ff ff
```

```
104c53e:      66 89 81 9f 19 02 11    mov    %ax,0x1102199f(%ecx)
104c545:      7f 99                   jg     104c4e0 <Option+0x8ea0>
104c547:      e9 54 71 ff ff          jmp    10436a0 <Option+0x60>
```

- In object file:

```
66 89 81 c3 c3 c3 c3 7f 99 e9 54 71 ff ff
```

```
1009a5e:      66 89 81 c3 c3 c3 c3    mov    %ax,-0x3c3c3c3d(%ecx)
1009a65:      7f 99                   jg     1009a00 <Option+0x8ea0>
1009a67:      e9 54 71 ff ff          jmp    1000bc0 <Option+0x60>
```

Example

UNIVERSITY OF MINNESOTA

- In nexe file:

```
66 89 81 9f 19 02 11 7f 99 e9 54 71 ff ff
```

```
104c540:  81 9f 19 02 11 7f 99      sbbl   $0x7154e999,0x7f110219(%edi)
104c547:  e9 54 71
104c54a:  ff                        (bad)
104c54b:  ff                        .byte 0xff
```

- In object file:

```
66 89 81 c3 c3 c3 c3 7f 99 e9 54 71 ff ff
```

```
1009a60:  81 c3 c3 c3 c3 7f         add    $0x7fc3c3c3,%ebx
1009a66:  99                        cltd
1009a67:  e9 54 71 ff ff            jmp    0xffff7160
```

Example

UNIVERSITY OF MINNESOTA

- In nexe file:

```
66 89 81 9f 19 02 11 7f 99 e9 54 71 ff ff
```

```
104c540:  81 9f 19 02 11 7f 99      sbbl   $0x7154e999,0x7f110219(%edi)
104c547:  e9 54 71
104c54a:  ff                        (bad)
104c54b:  ff                        .byte 0xff
```

- In object file:

```
66 89 81 c3 c3 c3 c3 7f 99 e9 54 71 ff ff
```

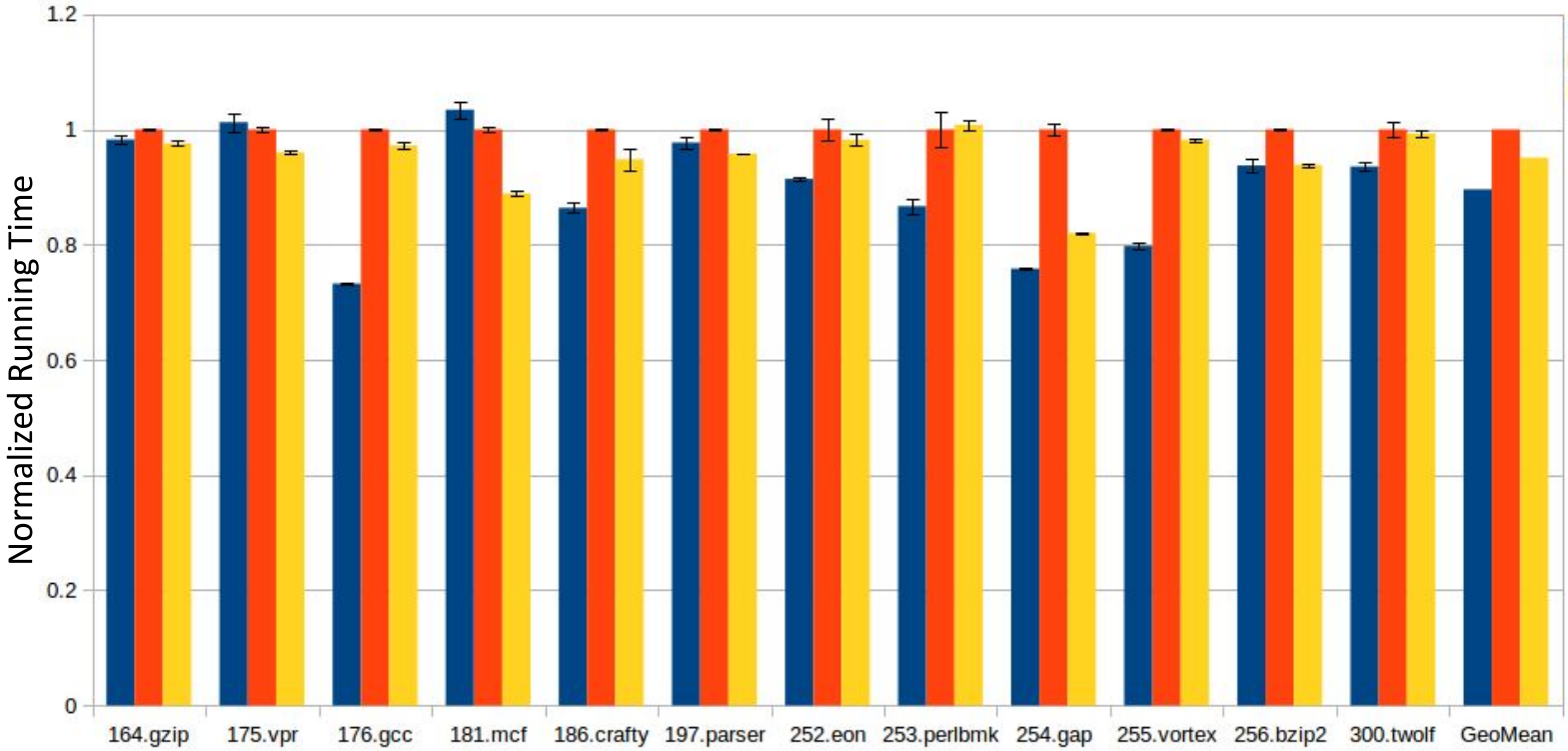
```
1009a60:  81 c3 c3 c3 c3 7f         add    $0x7fc3c3c3,%ebx
1009a66:  99                        cltd
1009a67:  e9 54 71 ff ff            jmp    0xffff7160
```

- We filter *9f c3 c3 c3*, as well

Performance Evaluation

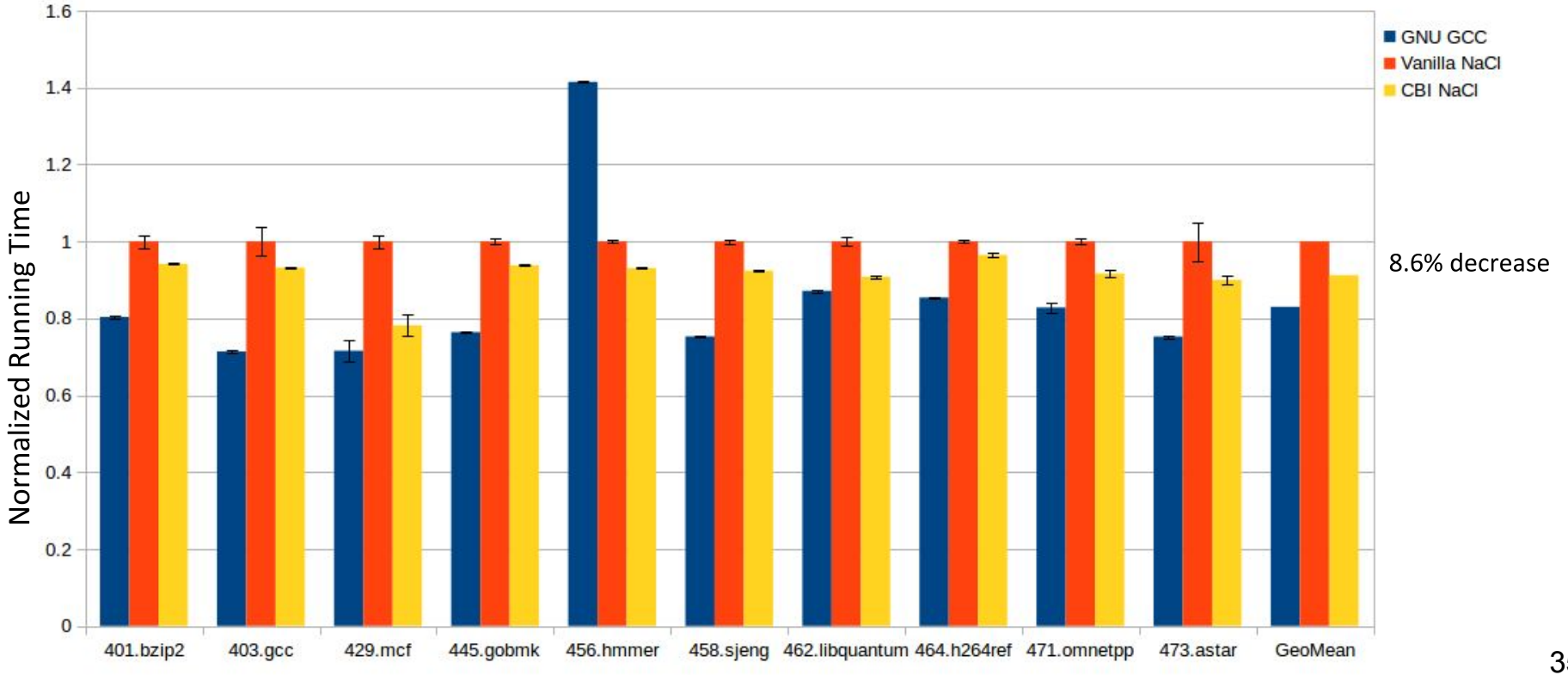
- SPEC2000 and SPEC2006 benchmarks
- Built binaries using CBI NaCl build chain
  - SPEC2000: all 12 benchmarks built successfully
  - SPEC2006: 10 benchmarks were built successfully
    - One did not build even with GNU g++ 4.4.3
    - One fails to be executed in the NaCl
- Run for 11 times
- Discarding first executions

SPEC2000



4.7% decrease

SPEC2006

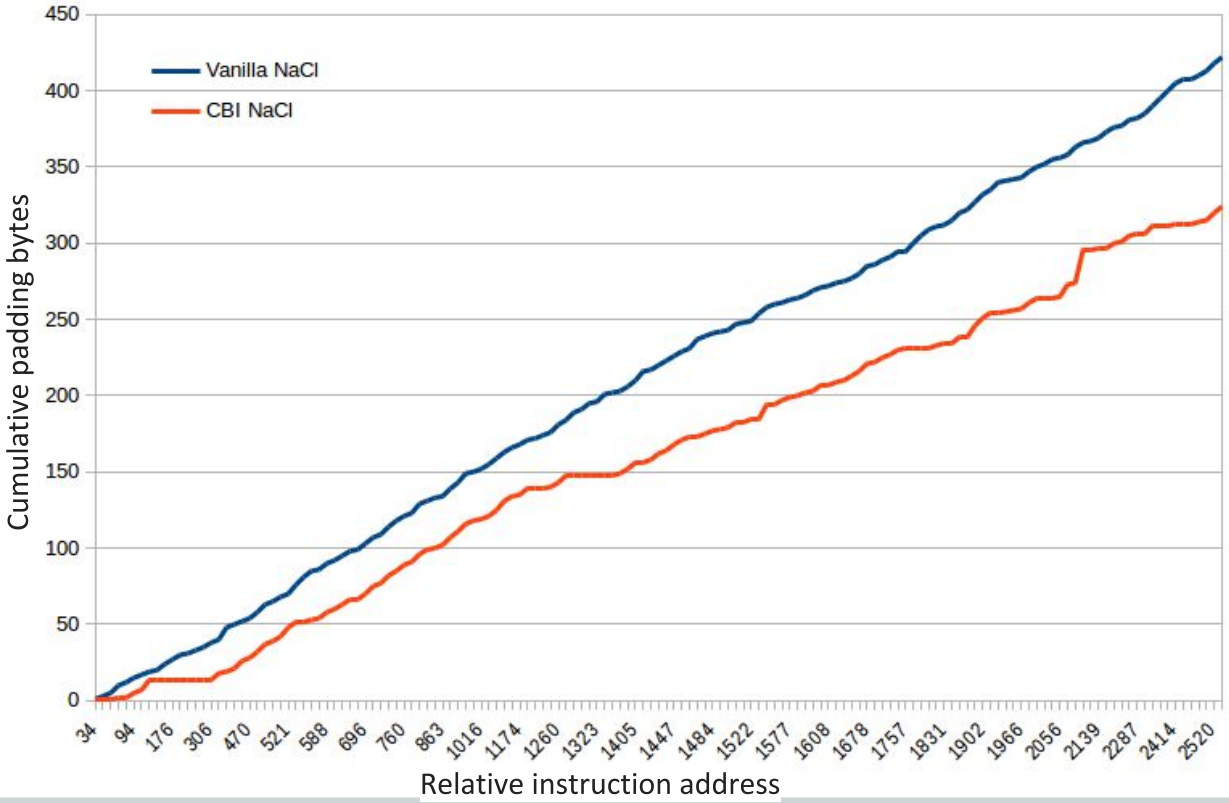


8.6% decrease

Number of instructions

- We are removing NOPs
- Decrease in the number of instructions executed
  - 1.3% for SPEC2000 (collected via Valgrind)
  - 15% for SPEC2006 (collected via OProfile)
- We were able to remove more NOPs from SPEC2006
  - Less number of NOPs executed yields better performance

Pad Removal Success



On x64

- CBI NaCl is also applicable to x64
- x64 validation rules are more restrictive
  - No address segmentation
  - Restricted set of addressing are allowed
  - As a result: fewer overlapping instructions allowed
- CBI NaCl on x64
  - Rarely able to remove padding
  - Less than 0.5% performance improvement

Future Work

- Replacing the greedy pad removal with a dynamic programming one
- Instruction re-ordering
  - To avoid branch prediction disruption
  - Reuse padded space to place actual instruction

Conclusion

- Effect of instruction level padding on SFI performance
- Change Native Client padding scheme
- Change NaCl build chain
  - Generate valid binaries
- Update validator appropriately and prove the correctness
- More than 8% performance improvement on average
- Project artifacts at:
  - `https://www-users.cs.umn.edu/~emamd001/cbi-nacl.html`

Thank you for your attention
Q/A