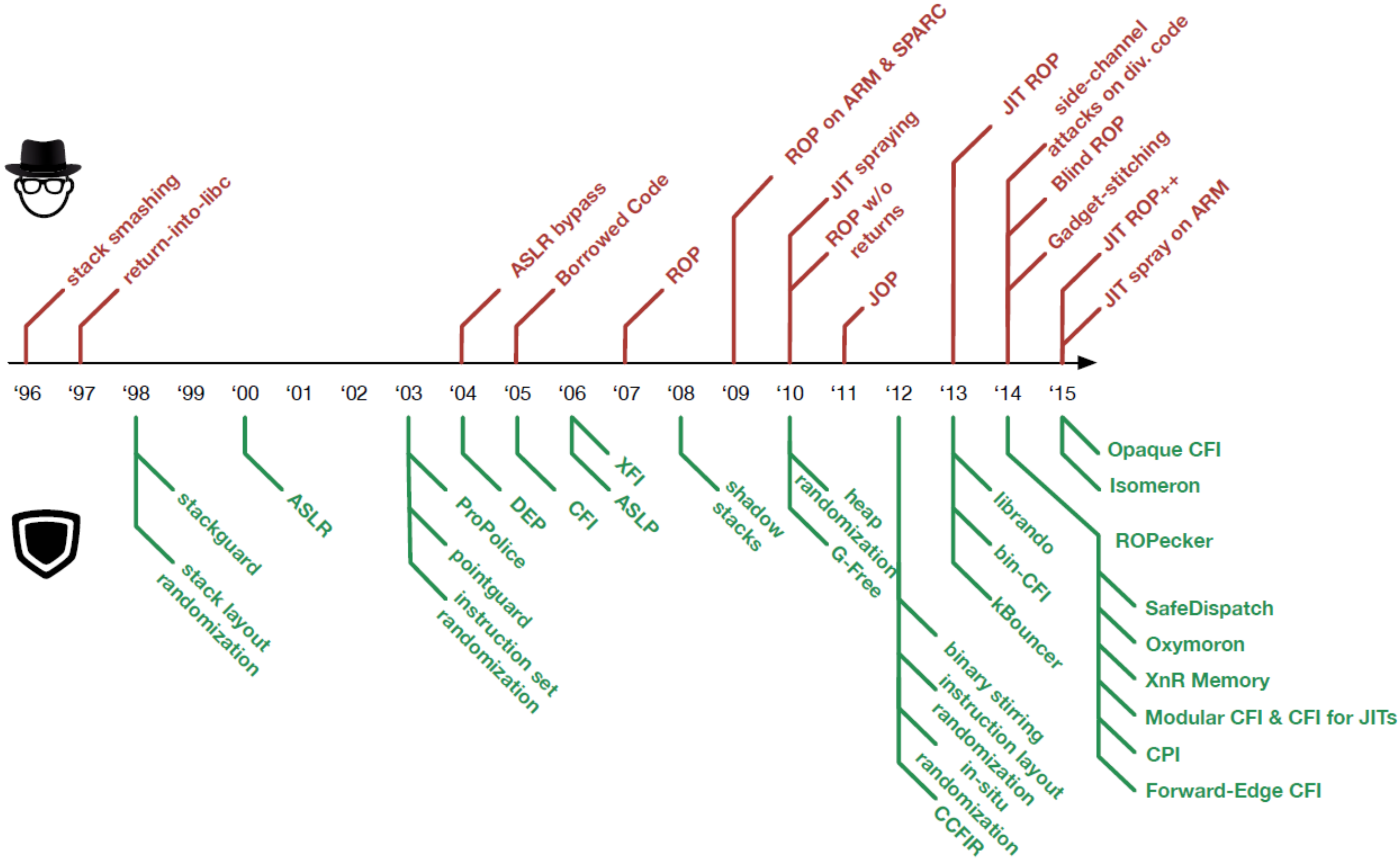
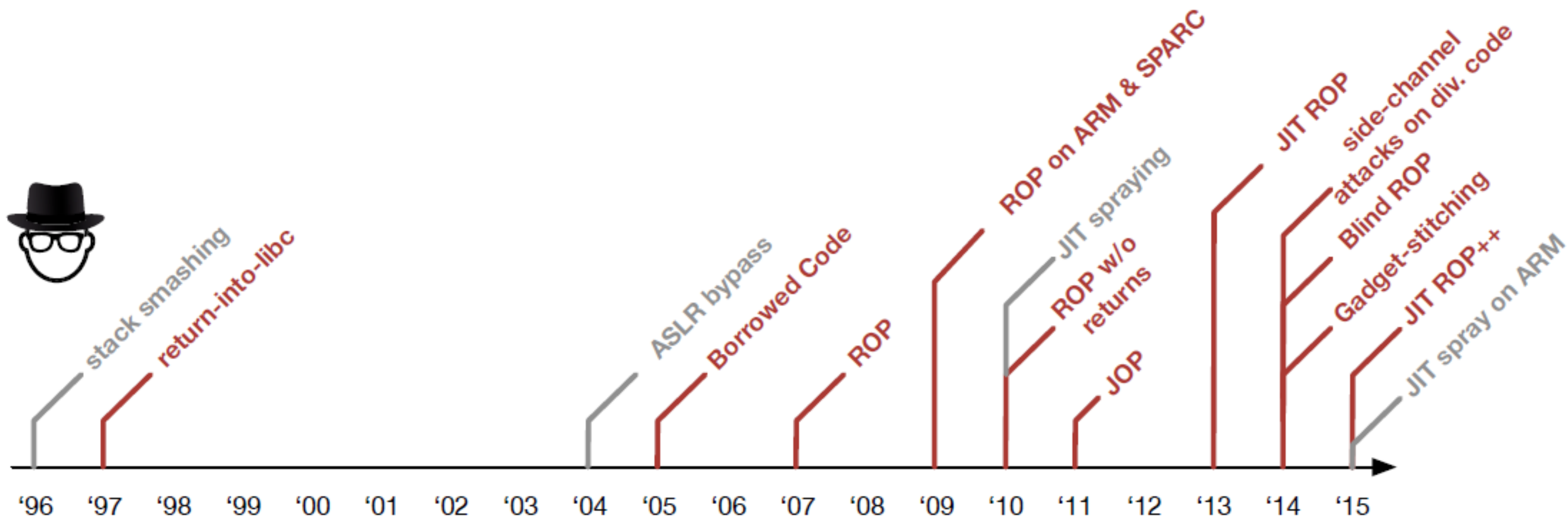
Opaque Control-Flow Integrity

Vishwath Mohan[¥], Per Larsen[§], Stefan Brunthaler[§], Kevin W. Hamlen[¥], Michael Franz[§]

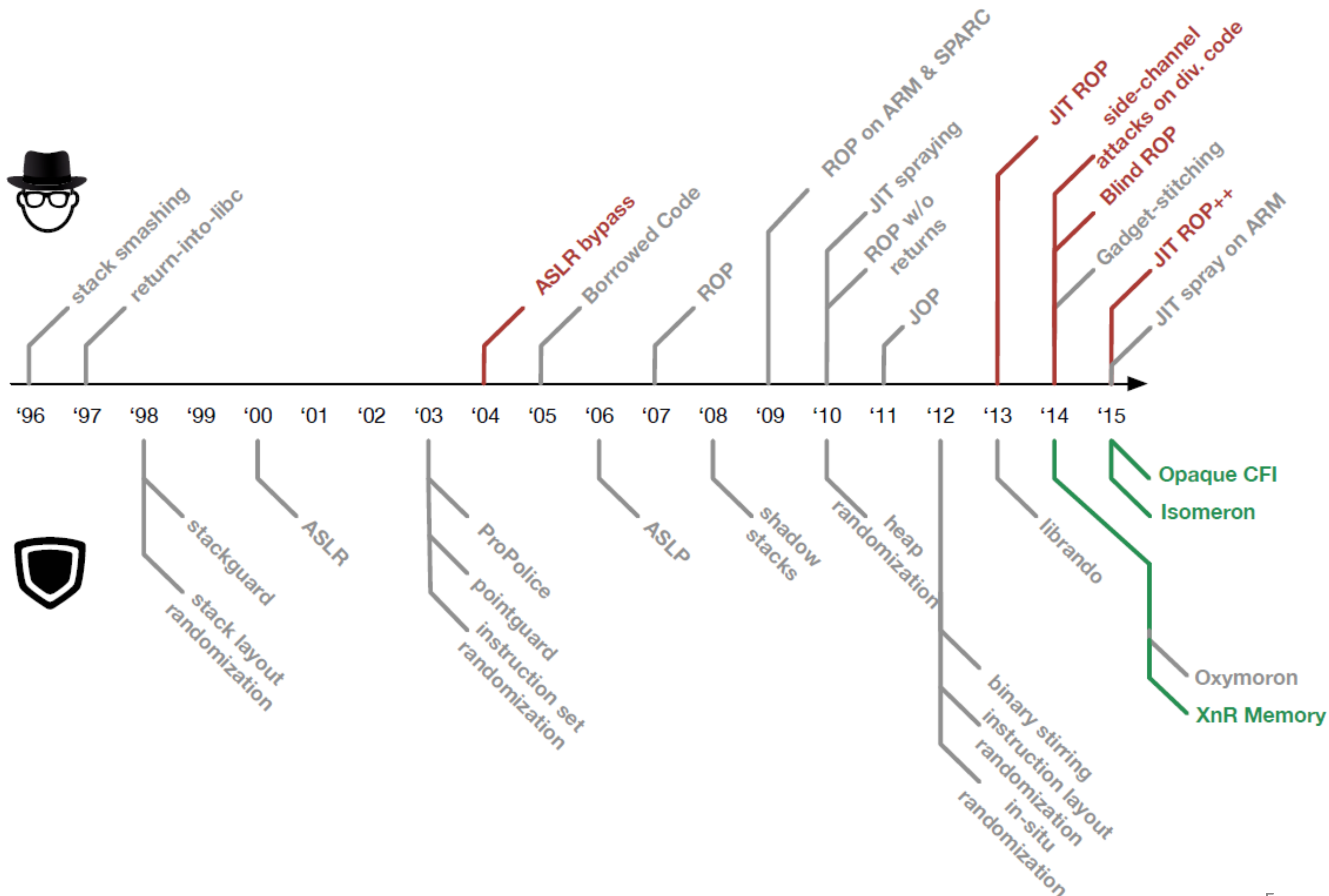
The University of Texas at Dallas[¥] University of California, Irvine[§]

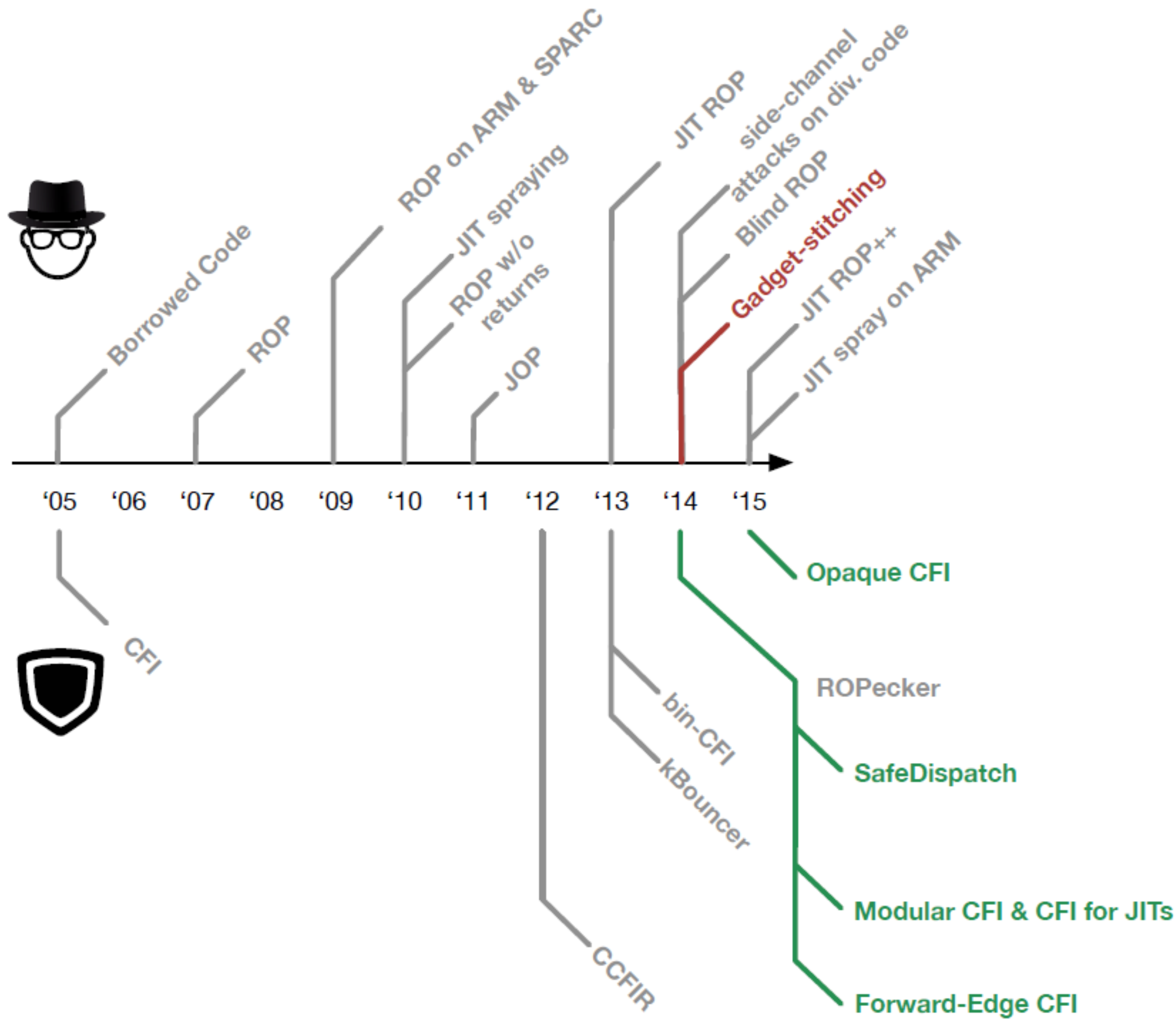

stack smashing
return-into-libc
ASLR bypass
Borrowed Code
ROP
ROP on ARM & SPARC
JIT spraying
ROP w/o returns
JOP
JIT ROP
side-channel attacks on div. code
Blind ROP
Gadget-stitching
JIT ROP++
JIT spray on ARM
'96
'97
'98
'99
'00
'01
'02
'03
'04
'05
'06
'07
'08
'09
'10
'11
'12
'13
'14
'15
stackguard
stack layout randomization
ASLR
ProPolice
pointguard
instruction set randomization
DEP
CFI
XFI
ASLP
shadow stacks
heap randomization
G-Free
binary stirring
instruction layout randomization
in-situ randomization
CCFIR
librando
bin-CFI
kBouncer
ROPecker
SafeDispatch
Oxymoron
XnR Memory
Modular CFI & CFI for JITs
CPI
Forward-Edge CFI
Opaque CFI
Isomeron


stack smashing
return-into-libc
ASLR bypass
Borrowed Code
ROP
ROP on ARM & SPARC
JIT spraying
ROP w/o returns
JOP
JIT ROP
side-channel attacks on div. code
Blind ROP
Gadget-stitching
JIT ROP++
JIT spray on ARM
'96
'97
'98
'99
'00
'01
'02
'03
'04
'05
'06
'07
'08
'09
'10
'11
'12
'13
'14
'15

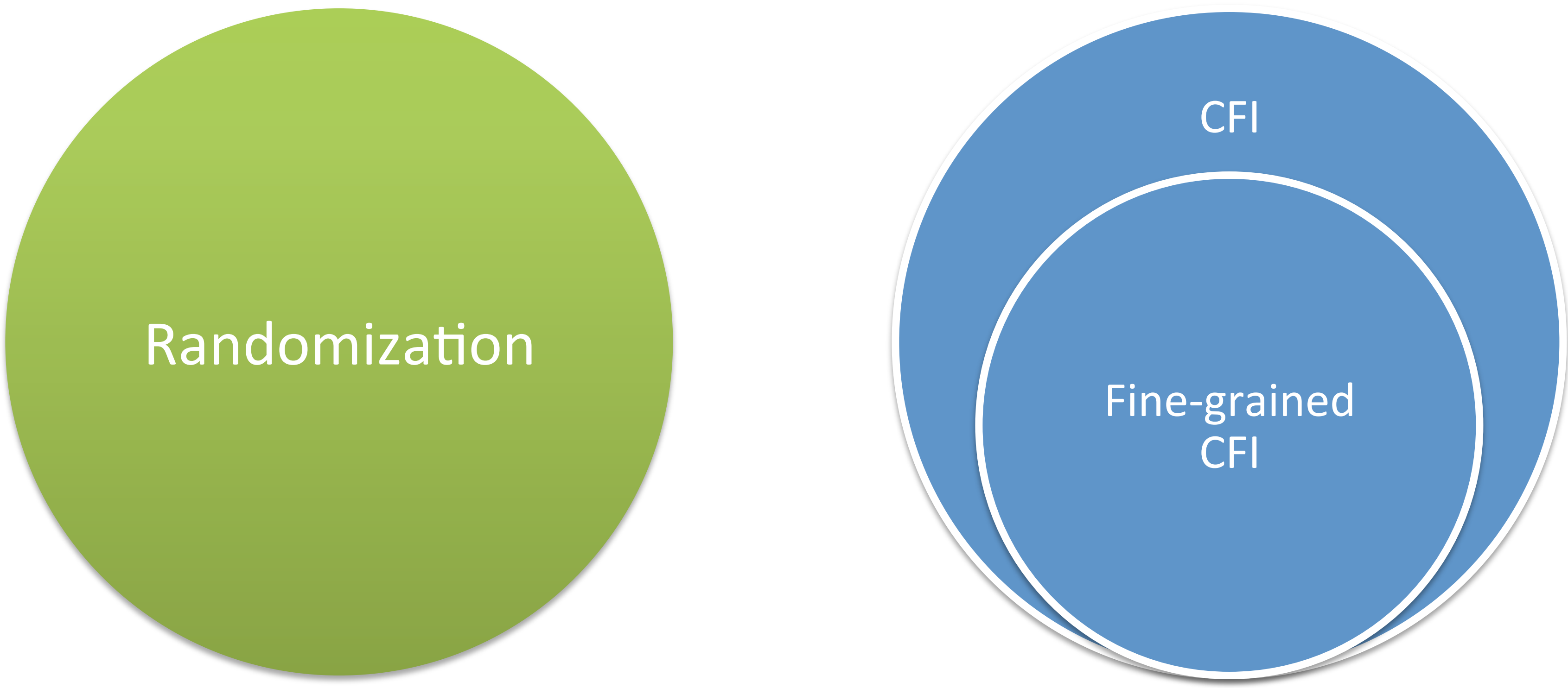
Code Reuse Attacks

- Needs
  - Location of code
  - Hijack control-flow
- Defensive options
  - Randomization
  - Control Flow Integrity

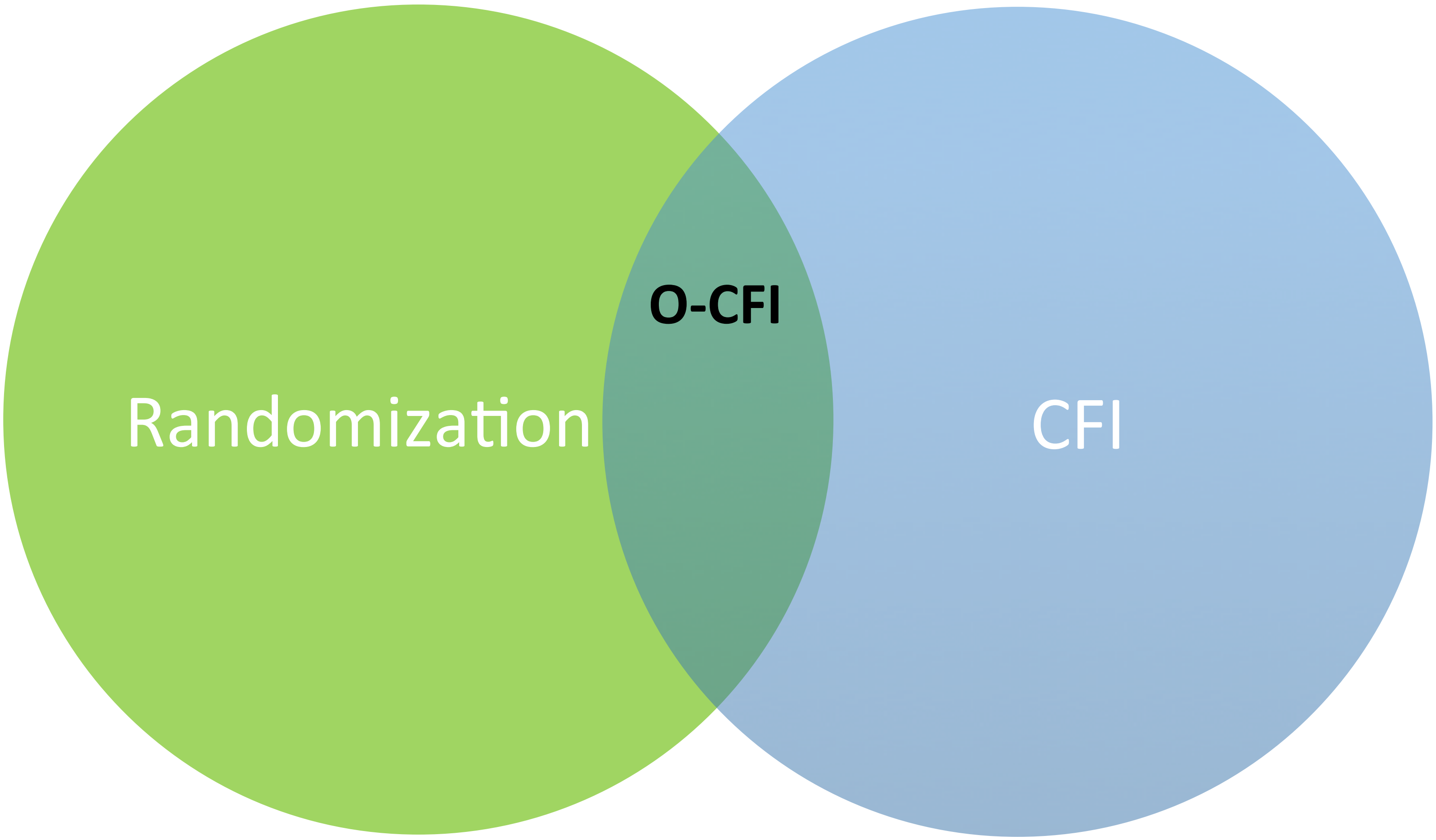

stack smashing
return-into-libc
ASLR bypass
Borrowed Code
ROP
ROP on ARM & SPARC
JIT spraying
ROP w/o returns
JOP
JIT ROP
side-channel attacks on div. code
Blind ROP
Gadget-stitching
JIT ROP++
JIT spray on ARM
'96
'97
'98
'99
'00
'01
'02
'03
'04
'05
'06
'07
'08
'09
'10
'11
'12
'13
'14
'15
stackguard
stack layout randomization
ASLR
ProPolice
pointguard
instruction set randomization
ASLP
shadow stacks
heap randomization
binary stirring
instruction layout randomization
in-situ randomization
librando
Opaque CFI
Isomeron
Oxymoron
XnR Memory

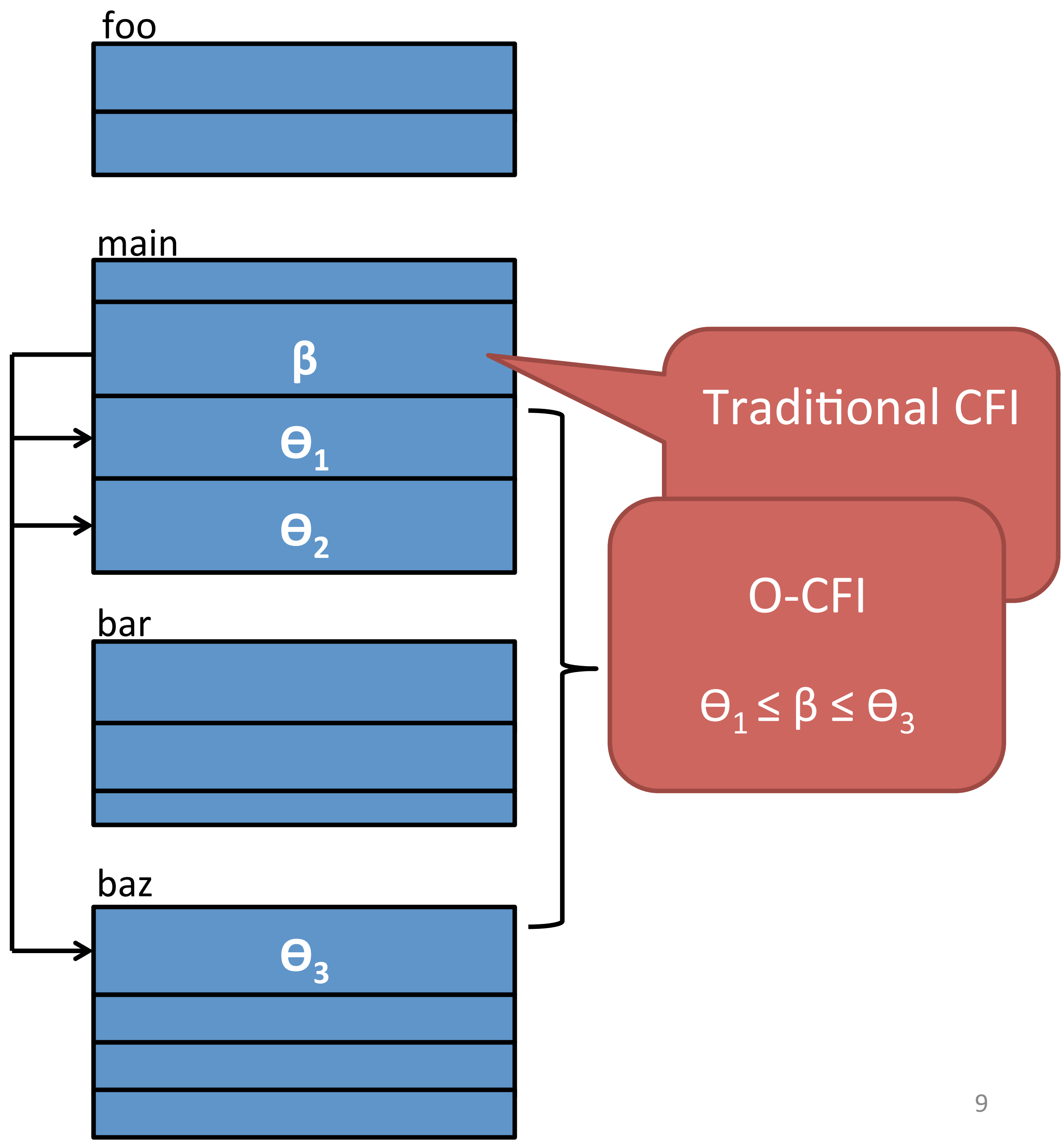

Borrowed Code
ROP
ROP on ARM & SPARC
JIT spraying
ROP w/o returns
JOP
JIT ROP
side-channel attacks on div. code
Blind ROP
Gadget-stitching
JIT ROP++
JIT spray on ARM
'05
'06
'07
'08
'09
'10
'11
'12
'13
'14
'15
CFI
CCFIR
bin-CFI
kBouncer
Opaque CFI
ROPecker
SafeDispatch
Modular CFI & CFI for JITs
Forward-Edge CFI

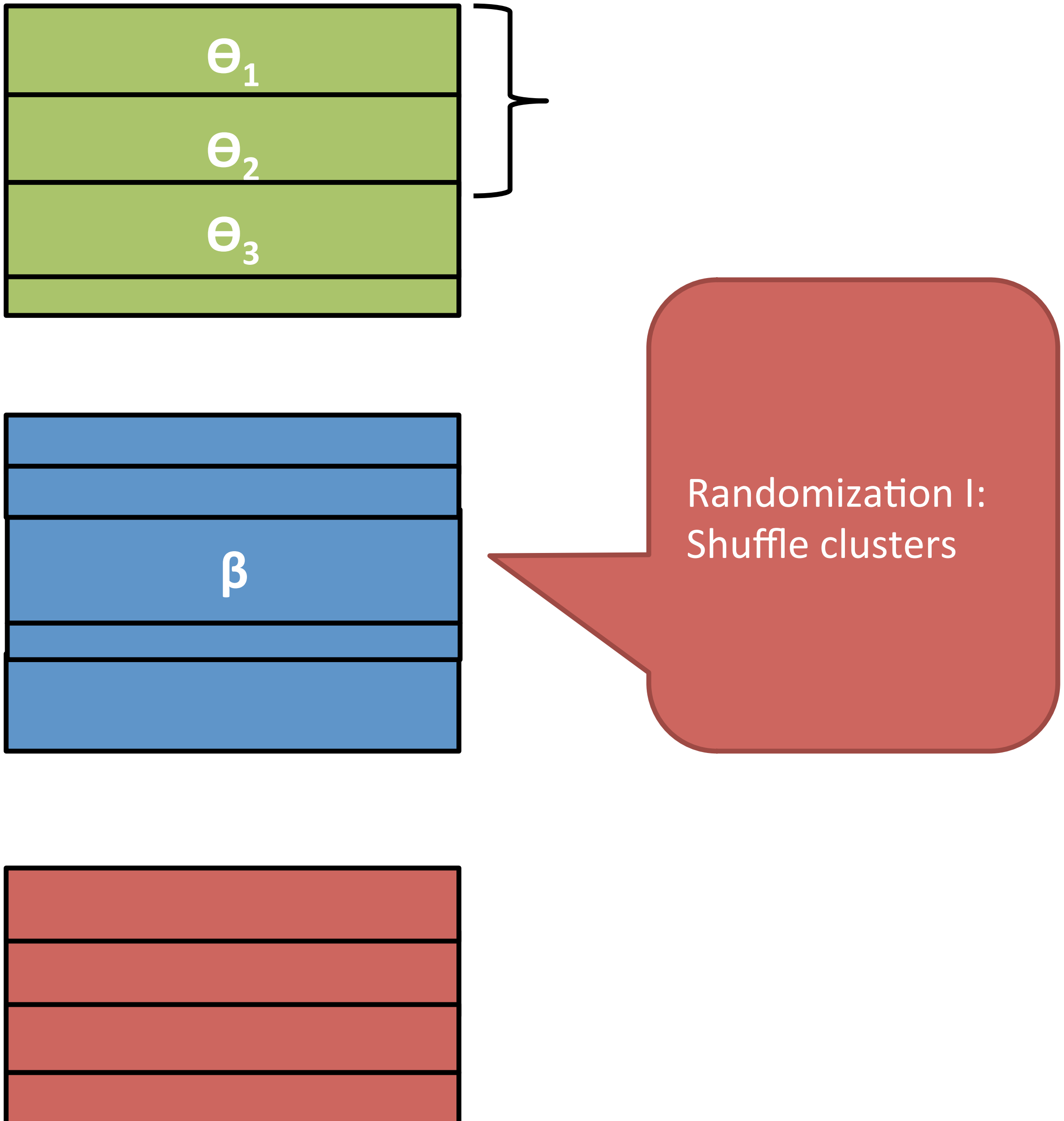
Where does that leave us?

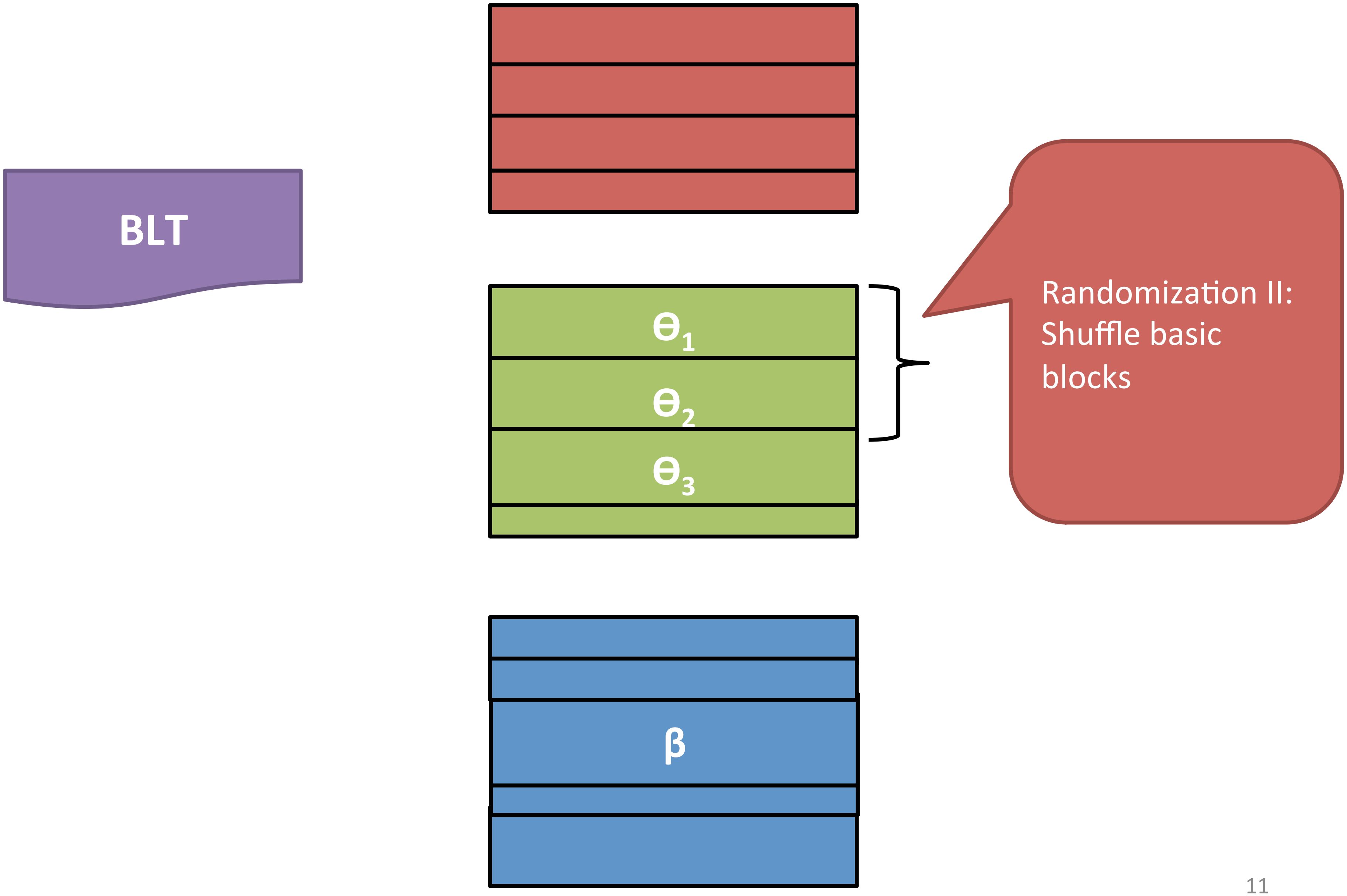


Opaque Control-Flow Integrity




foo
main
β
Θ1
Θ2
bar
baz
Θ3
Traditional CFI
O-CFI
Θ1 ≤ β ≤ Θ3


θ1
θ2
θ3
β
Randomization I:
Shuffle clusters


BLT
θ1
θ2
θ3
β
Randomization II: Shuffle basic blocks

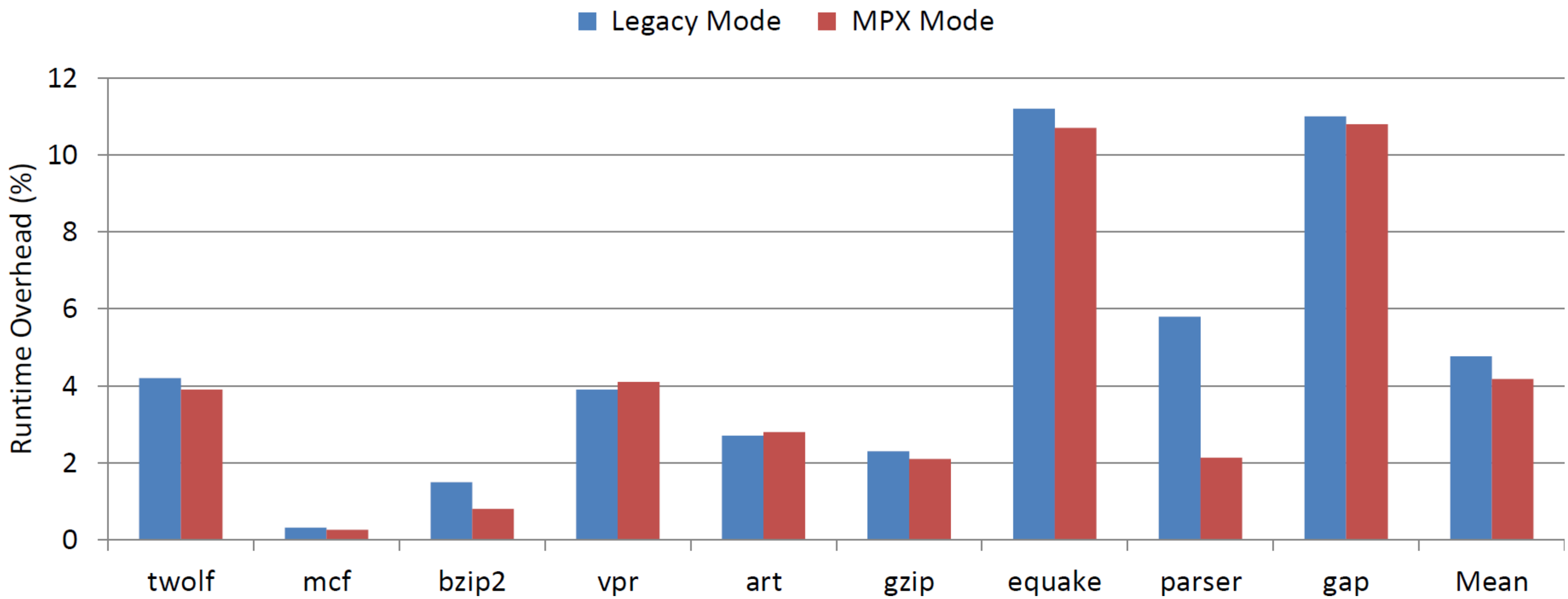
Accelerated Bounds Checks

- MPX mode on supported chipsets

Syntax	Description
`bndmov bnd, m64`	Move upper and lower bound from `m64` to bound register `bnd`.
`bndcl bnd, r/m32`	Generate a #BR if `r/m32` is less than the lower bound in `bnd`.
`bndcu bnd, r/m32`	Generate a #BR is `r/m32` is higher than the upper bound in `bnd`.

- Legacy mode as fallback

Performance



Legacy mode overhead 4.7%

MPX mode overhead ≈ 4.2%

Security

- When the CFI policy is opaque

Gadget Chain Size	Chance(%)
2	2.0
3	0.8
4	0.01
5	-

Security

- When the CFI policy is not opaque
  - Expressed as CSP
  - Attempted constructing `VirtualAlloc` payload
  - Across Mona/custom tool, no payload found

Conclusion

- Coarse-grained CFI with randomization
  - Advantages of both
- Effective against state-of-the-art exploits
  - JIT-ROP, BROP, Gadget stitching
- Efficient
  - 4.7% overhead in legacy mode

Thank you

Extra Resources

Optimizing Guards

- Actual guard implementation
  - PittSFIeld inspired guards
  - Want minimal chunk size
  - Comparison instructions rather large (~ 7 bytes)
- How efficient can we be?

Optimizing Guards

Description	Original Code	Rewritten Code (MPX-mode)	Rewritten Code (Legacy-mode)
Indirect Branches	call/jmp r/[m]	1: mov [esp-4], eax 2: mov eax, r/[m] 3: cmp byte ptr [eax], 0xF4 4: cmovz eax, [eax+1] — chunk boundary — 5: bndmov bnd1, gs:[branch_id] 6: bndcu bnd1, eax 7: jmp 9 — chunk boundary — 8: xor eax, eax 9: and al, align_mask 10: bndcl bnd1, eax 11: xchg eax, [esp-4] 12: call/jmp [esp-4]	1: push ecx 2: push eax 3: mov eax, r/[m] 4: cmp byte ptr [eax], 0xF4 5: cmovz eax, [eax+1] — chunk boundary — 6: mov ecx, branch_id 7: cmp eax, gs:[ecx] 8: jb 10 9: cmp gs:[ecx+4], eax boundary —— 10: jbe abort — chunk 11: and al, align_mask 12: xchg eax, [esp] 13: pop ecx 14: pop ecx 15: call/jmp [esp-8]
Returns	ret ⟨n⟩	— chunk boundary — 1: xchg eax, [esp] 2: and al, align_mask 3: bndmov bnd1, gs:[branch_id] 4: jmp 6 — chunk boundary — 5: xor eax, eax 6: bndcu bnd1, eax 7: bndcl bnd1, eax 8: xchg eax, [esp] 9: ret ⟨n⟩	— chunk boundary — 1: xchg eax, [esp] 2: cmp eax, gs:[branch_id] 3: jb 9 boundary —— 4: and al, align_mask —— chunk 5: cmp eax, gs:[branch_id + 4] 6: jae 9 7: xchg eax, [esp] 8: ret ⟨n⟩ — chunk boundary — 9: jmp abort

Coarse Grained Insecurity

		CFI [1]	bin-CFI [50]	CCFIR [49]	kBouncer [33]	ROPecker [7]	ROPGuard [16]	EMET [30]
DeMott [12]	Feb 2014							☹
Göktaş et al. [18]	May 2014	☹	☹	☹				
Davi et al. [11]	Aug 2014		☹		☹	☹	☹	☹
Göktaş et al. [19]	Aug 2014				☹	☹		
Carlini and Wagner [6]	Aug 2014				☹	☹		