The Bright Side of the Moon

Exploring Novel Techniques for Bypassing Call Stack Analysis

bsi.

$> whoami

Alessandro Magnosi

KlezVirus

Principal Security Consultant

Red Teamer, Malware Developer, Source Code Reviewer

Offensive Security OSS Developer

Author of Inceptor, and several other tools

Bug Bounty Hunter

Hunting bugs for fun and a little profit

Introduction

Definitions and Windows Internals Primer

Thread Stack Spoofing Approaches

History and previous research on the topic

Moonwalking the Windows Stack

Developing a dynamic call stack spoofer

Detecting Spoofed Frames

Designing detection strategies for moonwalking

Towards the future

Additional research avenues and where to find them

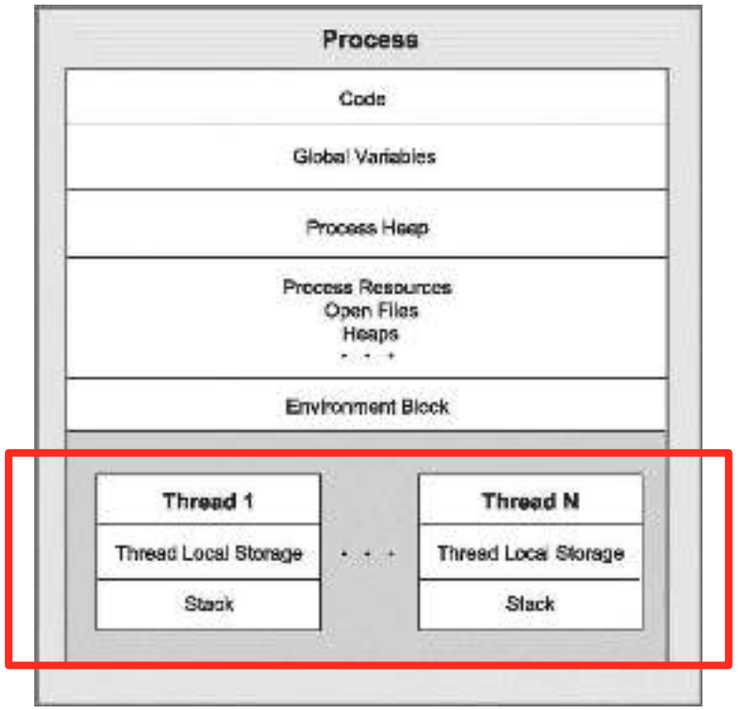
bsi.

Introduction

Definitions and Windows Internals Primer

Why “Thread Call Stack Spoofing”?

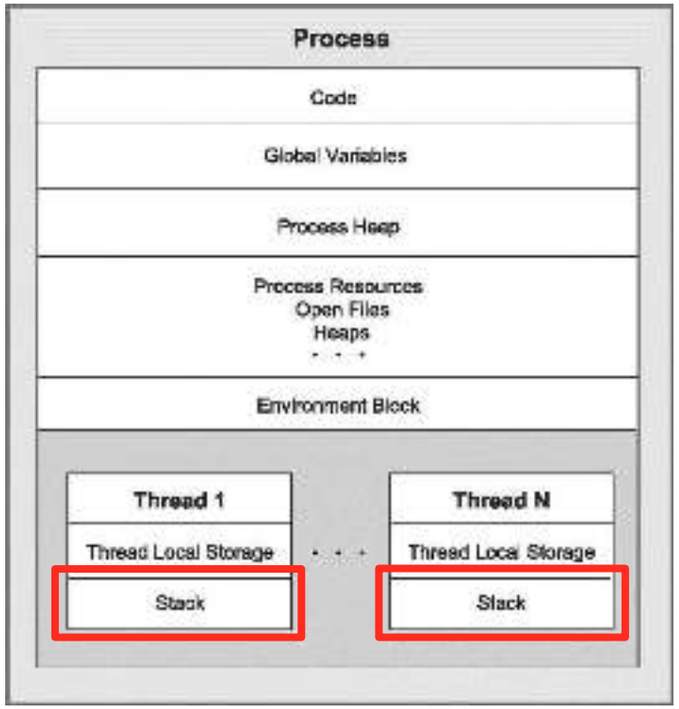
Thread



bsi.

Why “Thread Call Stack Spoofing”?

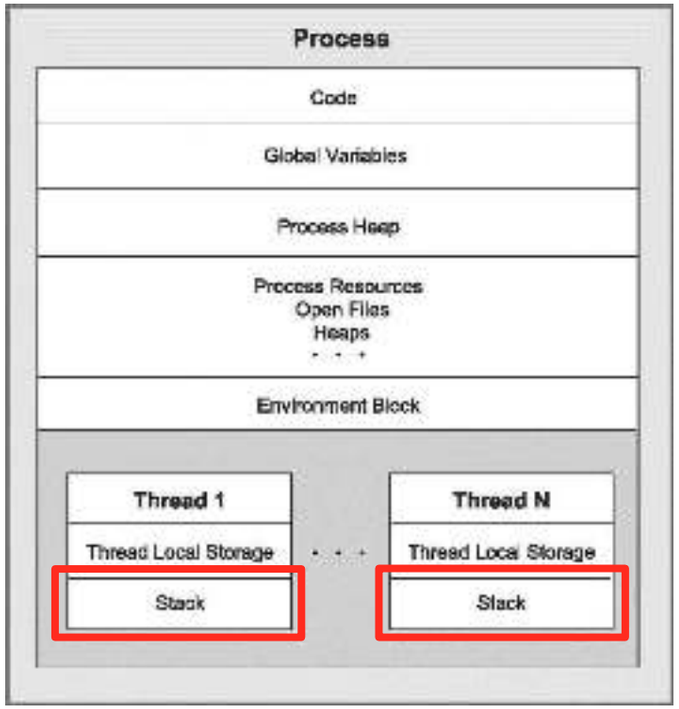
Thread



bsi.

Why “Thread Call Stack Spoofing”?

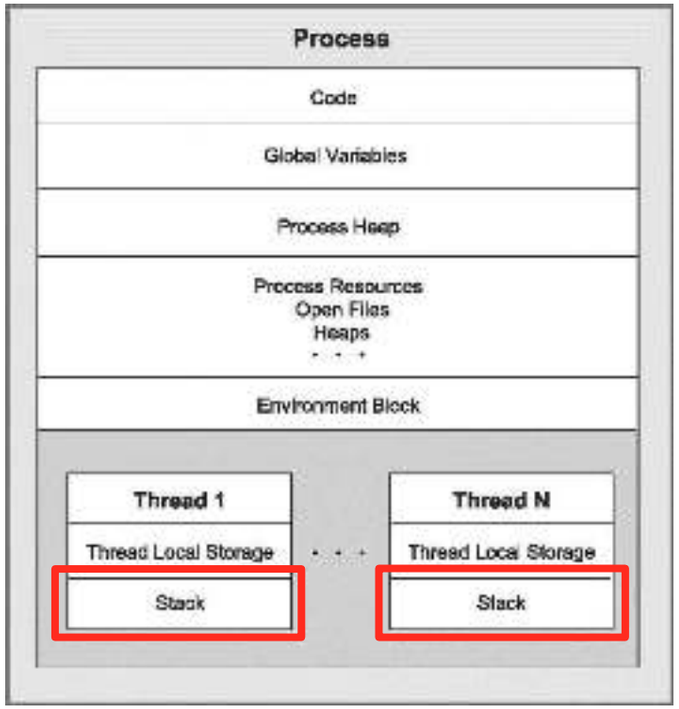
Thread Call Stack



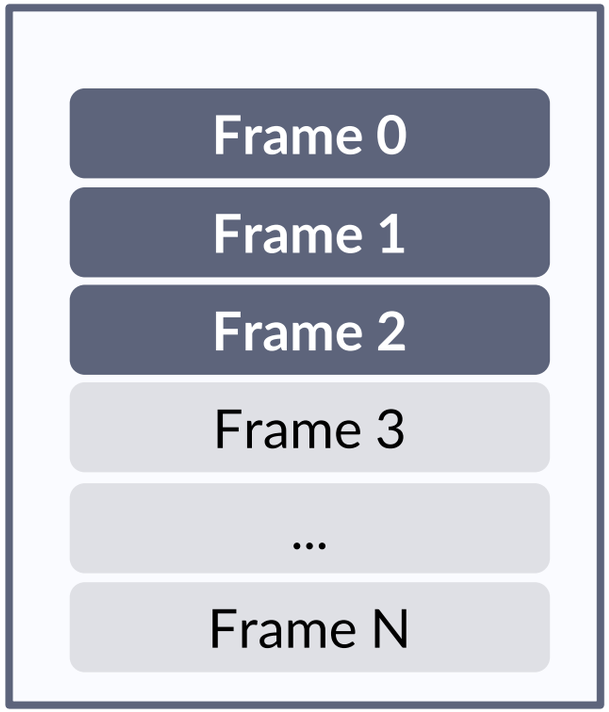
bsi.

Why “Thread Call Stack Spoofing”?

Thread



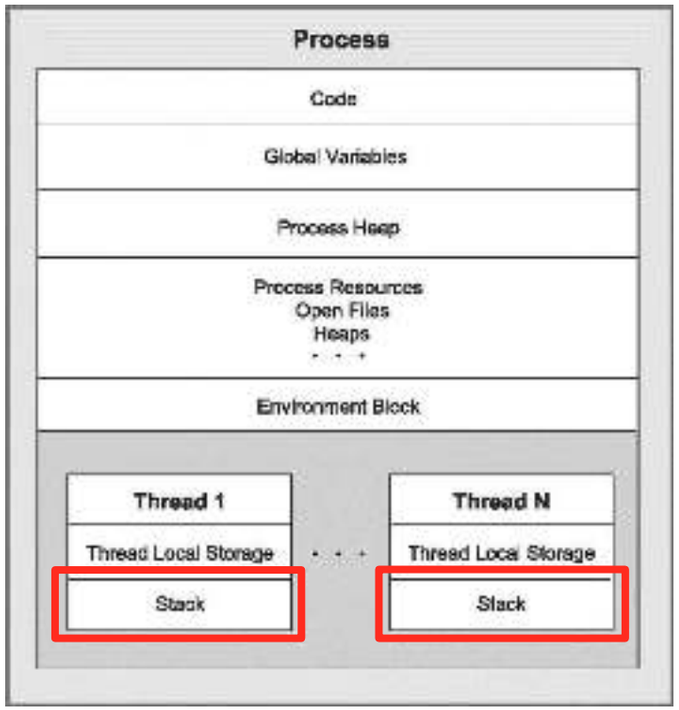
Call Stack

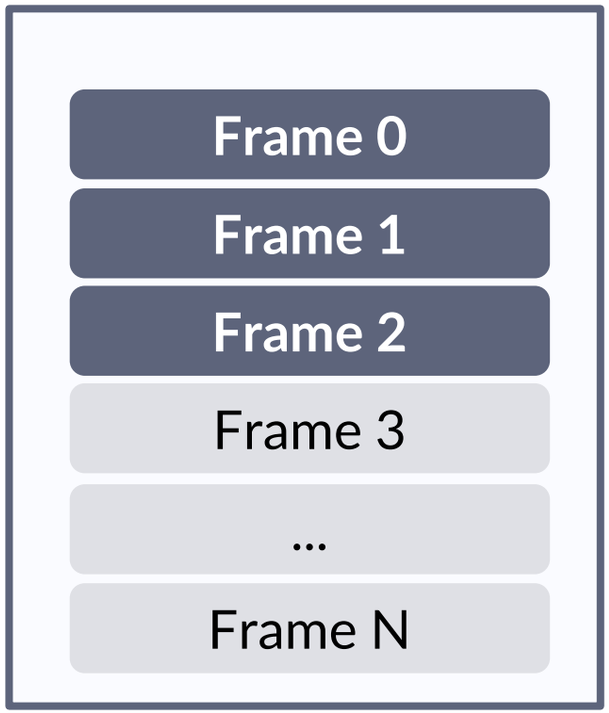


bsi.

Why “Thread Call Stack Spoofing”?

Thread Call Stack Spoofing

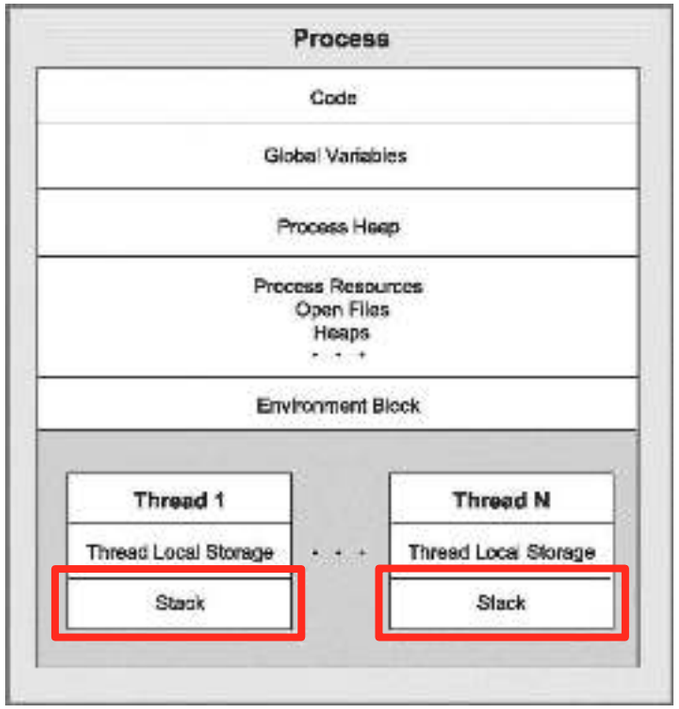




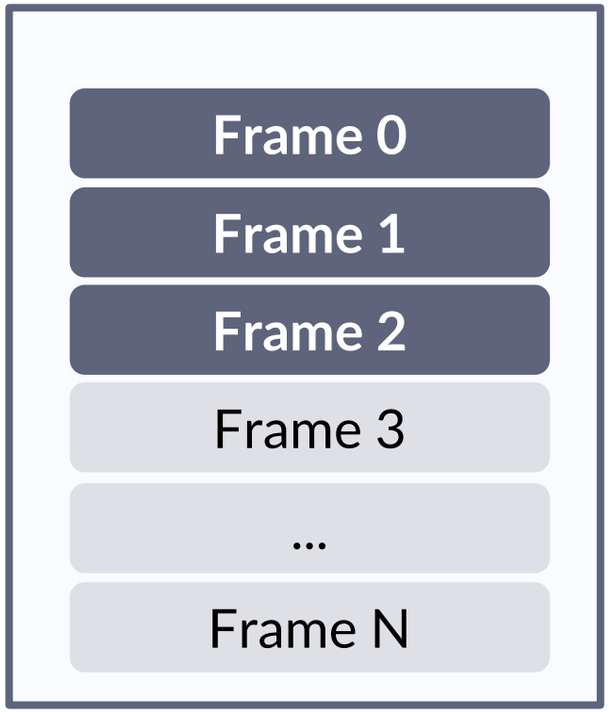
bsi.

Why “Thread Call Stack Spoofing”?

Thread



Call Stack



Spoofing

bsi.

Thread Call Stack

```
00, win32u.dll!NtUserGetMessage+0x14
01, user32.dll!GetMessageW+0x2a
02, notepad++.exe!SetLibraryProperty+0x19ae92
03, notepad++.exe!GetNameSpace+0x105802
04, kernel32.dll!BaseThreadInitThunk+0x1d
05, ntdll.dll!RtlUserThreadStart+0x28
```

```
00, win32u.dll!NtUserGetMessage+0x14
01, user32.dll!GetMessageW+0x2a
02, Notepad.exe+0x734c6
03, Notepad.exe+0x18236
04, Notepad.exe+0x40e70
05, Notepad.exe+0x7b55e
06, kernel32.dll!BaseThreadInitThunk+0x1d
07, ntdll.dll!RtlUserThreadStart+0x28
```

```
00, ntdll.dll!NtDeviceIoControlFile+0x14
01, KernelBase.dll!GetConsoleScreenBufferInfoEx+0x229
02, KernelBase.dll!ReadConsoleW+0x1c5
03, KernelBase.dll!ReadConsoleW+0x1e
04, cmd.exe+0x2bf87
05, cmd.exe+0x13b11
06, cmd.exe+0x1305a
07, cmd.exe+0x12d66
08, cmd.exe+0x12ad2
09, cmd.exe+0x25b2d
10, cmd.exe+0x1f876
11, kernel32.dll!BaseThreadInitThunk+0x1d
12, ntdll.dll!RtlUserThreadStart+0x28
```

```
00, ntdll.dll!NtWaitForSingleObject+0x14
01, KernelBase.dll!DeviceIoControl+0x86
02, kernel32.dll!DeviceIoControl+0x81
03, conhost.exe+0x16490
04, conhost.exe+0x94a6
05, kernel32.dll!BaseThreadInitThunk+0x1d
06, ntdll.dll!RtlUserThreadStart+0x28
```

bsi.

Thread Call Stack

```
00, win32u.dll!NtUserGetMessage+0x14
01, user32.dll!GetMessageW+0x2a
02, notepad++.exe!SetLibraryProperty+0x19ae92
03, notepad++.exe!GetNameSpace+0x105802
04, kernel32.dll!BaseThreadInitThunk+0x1d
05, ntdll.dll!RtlUserThreadStart+0x28
```

```
00, win32u.dll!NtUserGetMessage+0x14
01, user32.dll!GetMessageW+0x2a
02, Notepad.exe+0x734c6
03, Notepad.exe+0x18236
04, Notepad.exe+0x40e70
05, Notepad.exe+0x7b55e
06, kernel32.dll!BaseThreadInitThunk+0x1d
07, ntdll.dll!RtlUserThreadStart+0x28
```

```
00, ntdll.dll!NtDeviceIoControlFile+0x14
01, KernelBase.dll!GetConsoleScreenBufferInfoEx+0x229
02, KernelBase.dll!ReadConsoleW+0x1c5
03, KernelBase.dll!ReadConsoleW+0x1e
04, cmd.exe+0x2bf87
05, cmd.exe+0x13b11
06, cmd.exe+0x1305a
07, cmd.exe+0x12d66
08, cmd.exe+0x12ad2
09, cmd.exe+0x25b2d
10, cmd.exe+0x1f876
11, kernel32.dll!BaseThreadInitThunk+0x1d
12, ntdll.dll!RtlUserThreadStart+0x28
```

```
00, ntdll.dll!NtWaitForSingleObject+0x14
01, KernelBase.dll!DeviceIoControl+0x86
02, kernel32.dll!DeviceIoControl+0x81
03, conhost.exe+0x16490
04, conhost.exe+0x94a6
05, kernel32.dll!BaseThreadInitThunk+0x1d
06, ntdll.dll!RtlUserThreadStart+0x28
```

bsi.

Thread Call Stack

```
00, win32u.dll!NtUserGetMessage+0x14
01, user32.dll!GetMessageW+0x2a
02, notepad++.exe!SetLibraryProperty+0x19ae92
03, notepad++.exe!GetNameSpace+0x105802
04, kernel32.dll!BaseThreadInitThunk+0x1d
05, ntdll.dll!RtlUserThreadStart+0x28
```

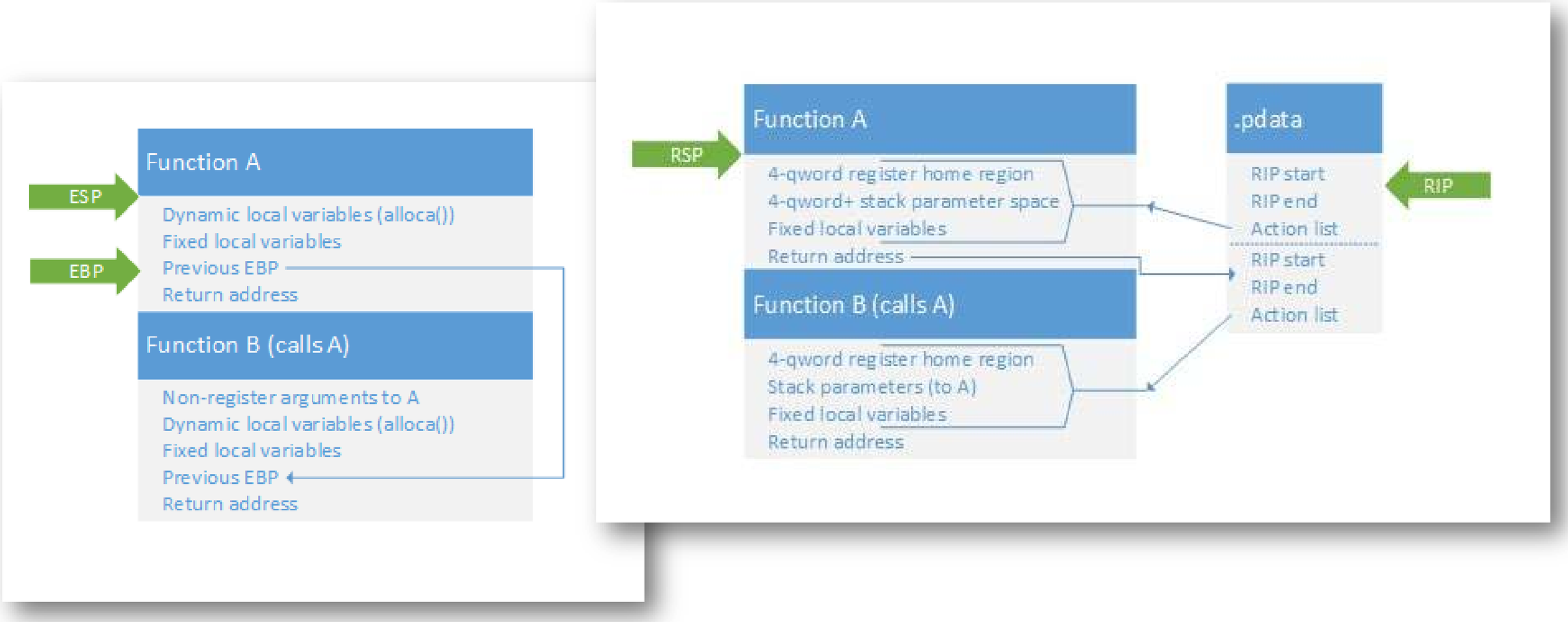
```
00, win32u.dll!NtUserGetMessage+0x14
01, user32.dll!GetMessageW+0x2a
02, Notepad.exe+0x734c6
03, Notepad.exe+0x18236
04, Notepad.exe+0x40e70
05, Notepad.exe+0x7b55e
06, kernel32.dll!BaseThreadInitThunk+0x1d
07, ntdll.dll!RtlUserThreadStart+0x28
```

```
00, ntdll.dll!NtDeviceIoControlFile+0x14
01, KernelBase.dll!GetConsoleScreenBufferInfoEx+0x229
02, KernelBase.dll!ReadConsoleW+0x1c5
03, KernelBase.dll!ReadConsoleW+0x1e
04, cmd.exe+0x2bf87
05, cmd.exe+0x13b11
06, cmd.exe+0x1305a
07, cmd.exe+0x12d66
08, cmd.exe+0x12ad2
09, cmd.exe+0x25b2d
10, cmd.exe+0x1f876
11, kernel32.dll!BaseThreadInitThunk+0x1d
12, ntdll.dll!RtlUserThreadStart+0x28
```

```
00, ntdll.dll!NtWaitForSingleObject+0x14
01, KernelBase.dll!DeviceIoControl+0x86
02, kernel32.dll!DeviceIoControl+0x81
03, conhost.exe+0x16490
04, conhost.exe+0x94a6
05, kernel32.dll!BaseThreadInitThunk+0x1d
06, ntdll.dll!RtlUserThreadStart+0x28
```

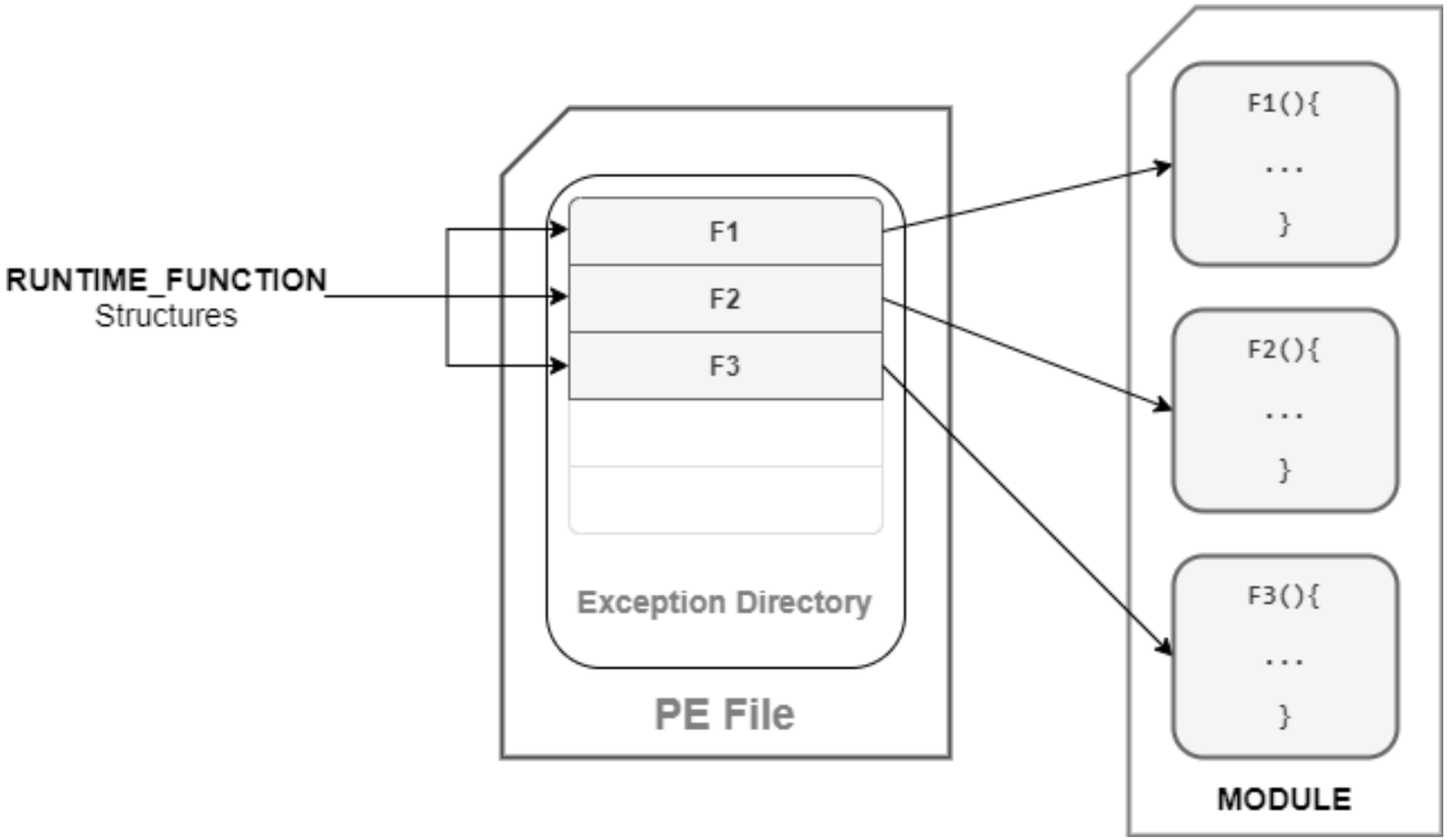
bsi.

Windows Calling Convention



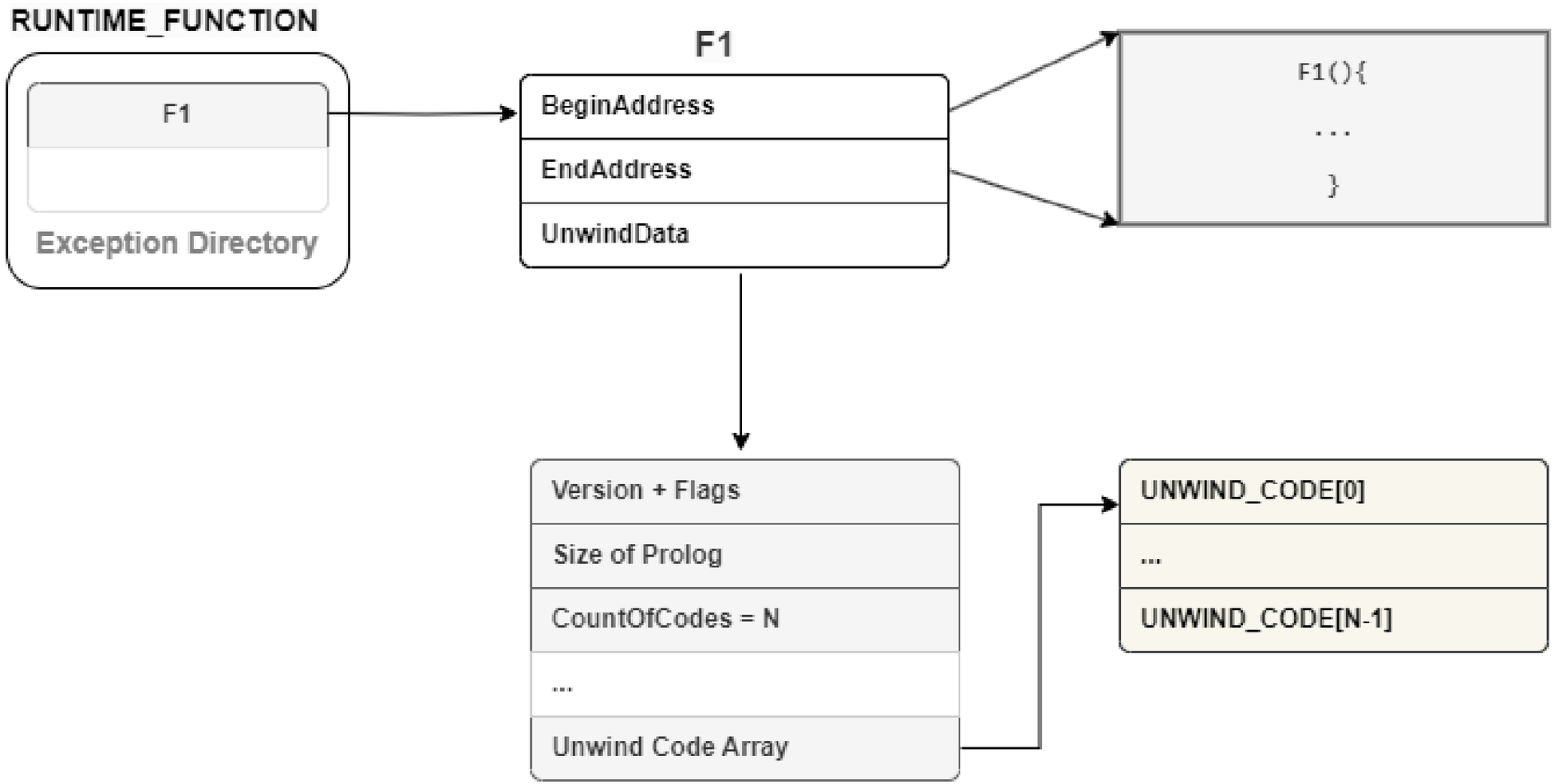
https://www.leviathansecurity.com/media/use-of-windows-exception-handling-metadata

Exceptions and Unwinding



https://codemachine.com/articles/x64_deep_dive.html

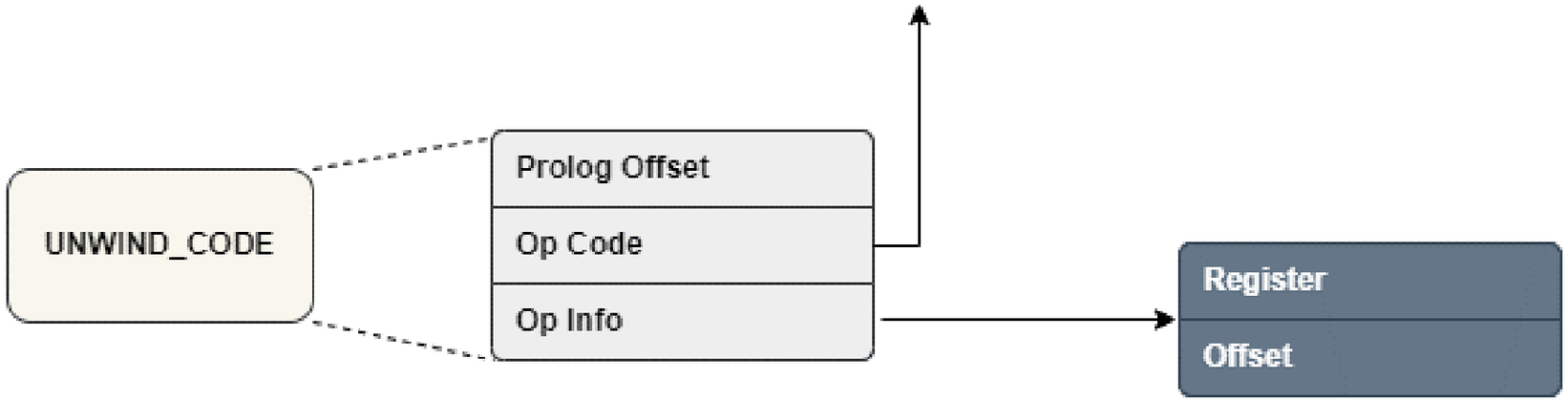
Exceptions and Unwinding



https://codemachine.com/articles/x64_deep_dive.html

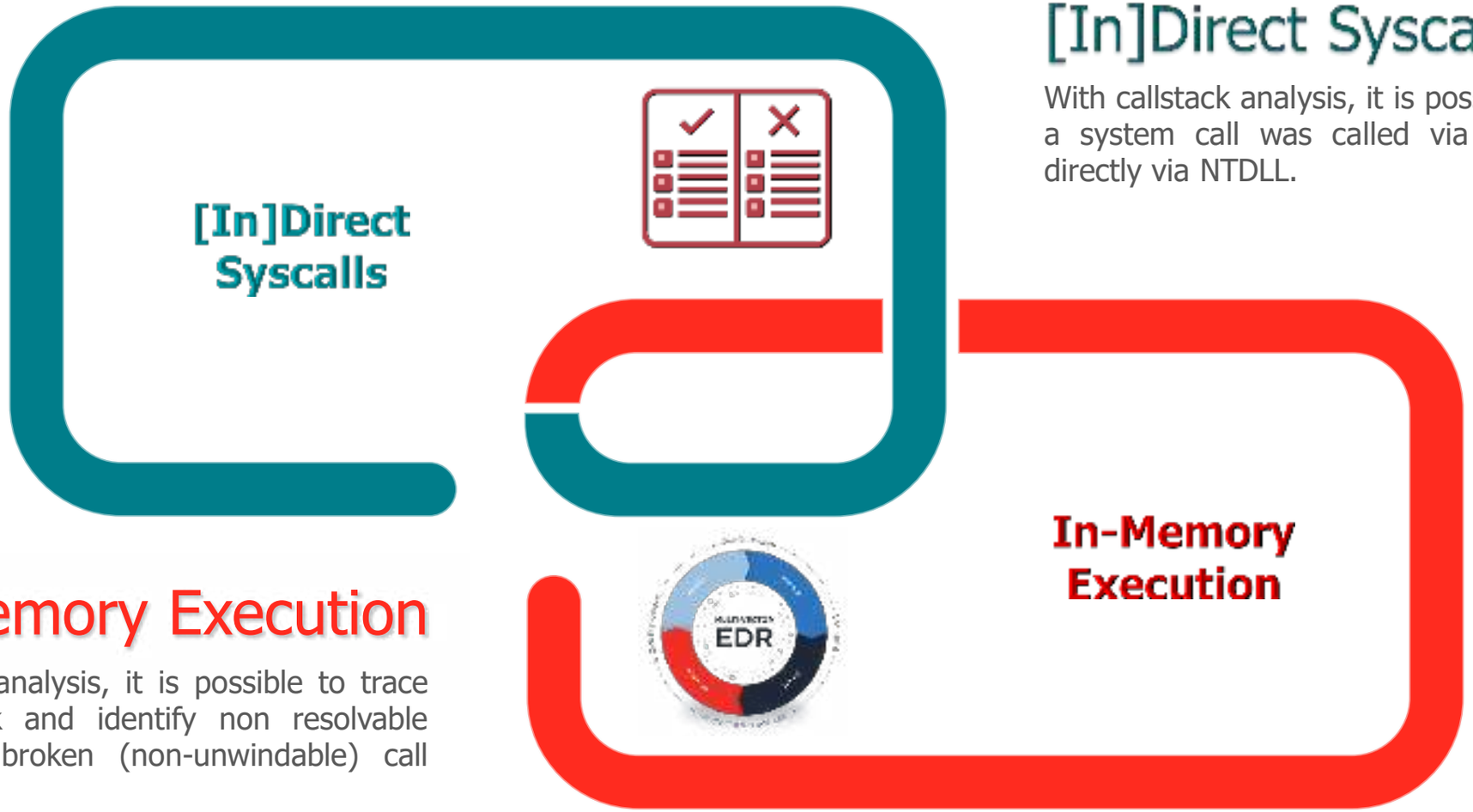
Exceptions and Unwinding

Impact Frame Size	*Do not Impact Frame Size*
UWOP_PUSH_NONVOL	UWOP_SET_FPREG
UWOP_ALLOC_SMALL	UWOP_SAVE_NONVOL
UWOP_ALLOC_LARGE	UWOP_SAVE_NONVOL_FAR
UWOP_PUSH_MACHFRAME	UWOP_SAVE_XMM128_FAR
	UWOP_SAVE_XMM128



https://codemachine.com/articles/x64_deep_dive.html

Call stack-based detection



[In]Direct Syscalls

With callstack analysis, it is possible to check if a system call was called via Kernelbase or directly via NTDLL.

In-Memory Execution

With callstack analysis, it is possible to trace execution back and identify non resolvable modules and broken (non-unwindable) call stacks.

In-Memory Execution

Normal Call Stack

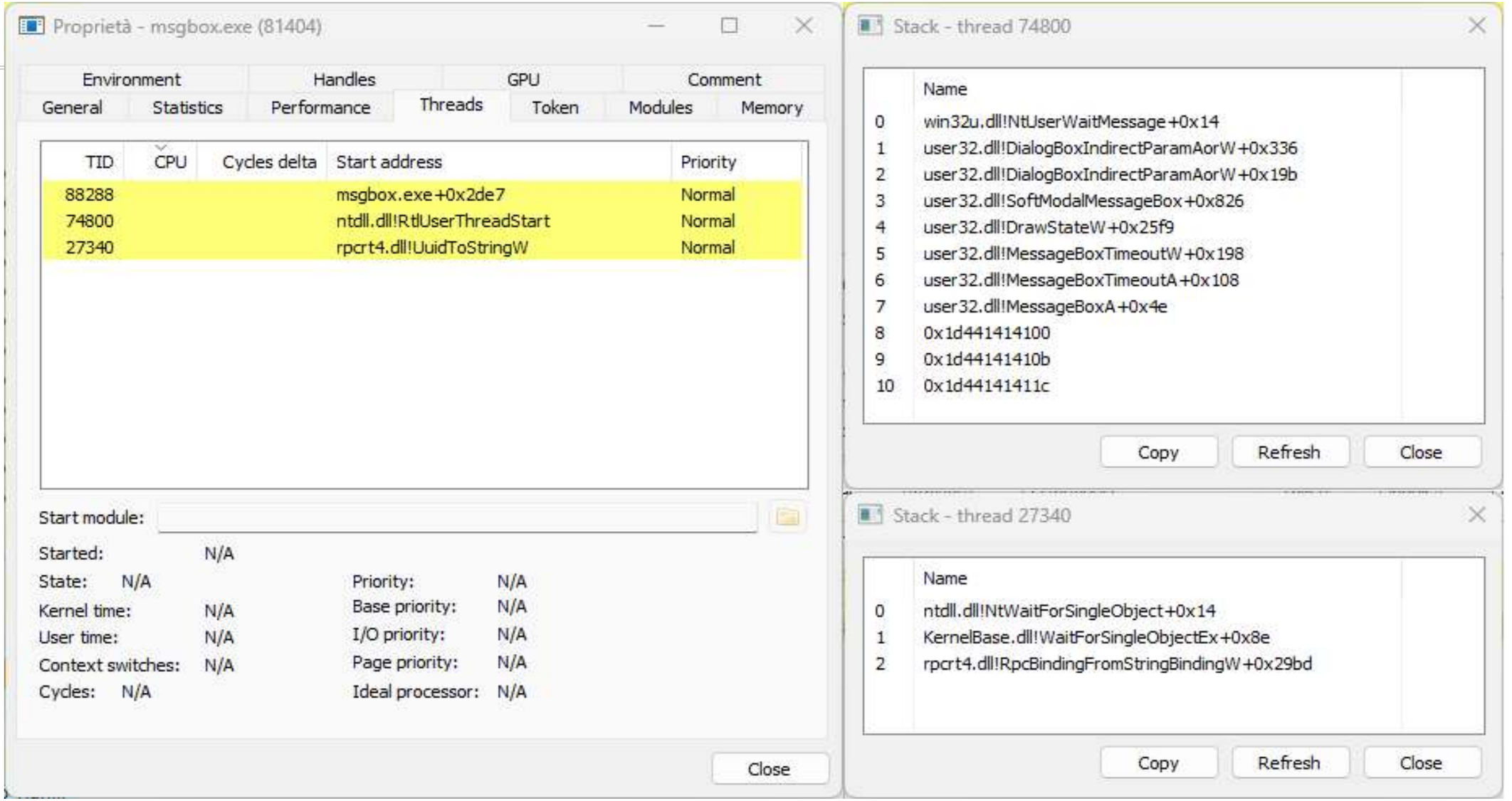
```
0, win32u.dll!NtUserWaitMessage+0x12
1, user32.dll!DialogBoxIndirectParamAorW+0x336
2, user32.dll!DialogBoxIndirectParamAorW+0x19b
3, user32.dll!SoftModalMessageBox+0x826
4, user32.dll!DrawStateW+0x25f9
5, user32.dll!MessageBoxTimeoutW+0x198
6, user32.dll!MessageBoxTimeoutA+0x108
7, user32.dll!MessageBoxA+0x4e
8, AbsentMoon.exe!main+0xed
9, AbsentMoon.exe!__scrt_common_main_seh+0x10c
10, kernel32.dll!BaseThreadInitThunk+0x1d
11, ntdll.dll!RtlUserThreadStart+0x28
```

In-Memory Code Call Stack

```
0, win32u.dll!NtUserWaitMessage+0x12
1, user32.dll!DialogBoxIndirectParamAorW+0x336
2, user32.dll!DialogBoxIndirectParamAorW+0x19b
3, user32.dll!SoftModalMessageBox+0x826
4, user32.dll!DrawStateW+0x25f9
5, user32.dll!MessageBoxTimeoutW+0x198
6, user32.dll!MessageBoxTimeoutA+0x108
7, user32.dll!MessageBoxA+0x4e
8, 0x0000000000000000
```

bsi.

In-Memory Execution - Stomping



[In]Direct System Call Execution

WINAPI Call Stack

```
0, ntdll.dll!NtAllocateVirtualMemory+0x13
1, KernelBase.dll!VirtualAllocExNuma+0x5c
2, KernelBase.dll!VirtualAllocEx+0x16
3, AbsentMoon.exe!main+0x5a
4, AbsentMoon.exe!__scrt_common_main_seh+0x10c
5, kernel32.dll!BaseThreadInitThunk+0x1d
6, ntdll.dll!RtlUserThreadStart+0x28
```

NTAPI Call Stack

```
0, ntdll.dll!NtAllocateVirtualMemory+0x13


1, AbsentMoon.exe!main+0xc4
2, AbsentMoon.exe!__scrt_common_main_seh+0x10c
3, kernel32.dll!BaseThreadInitThunk+0x1d
4, ntdll.dll!RtlUserThreadStart+0x28
```

bsi.

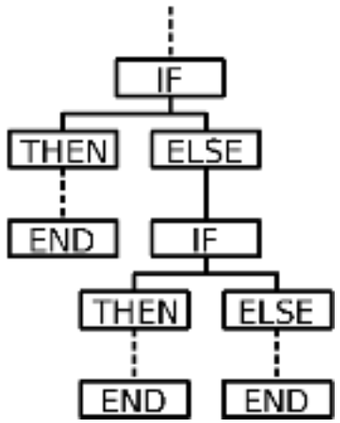
Detection Timings

Periodic

Every period of T seconds all threads may be scanned and their call stack analysed

Conditional

The EDR might scan all threads in a WAIT state to check for sleeping implants

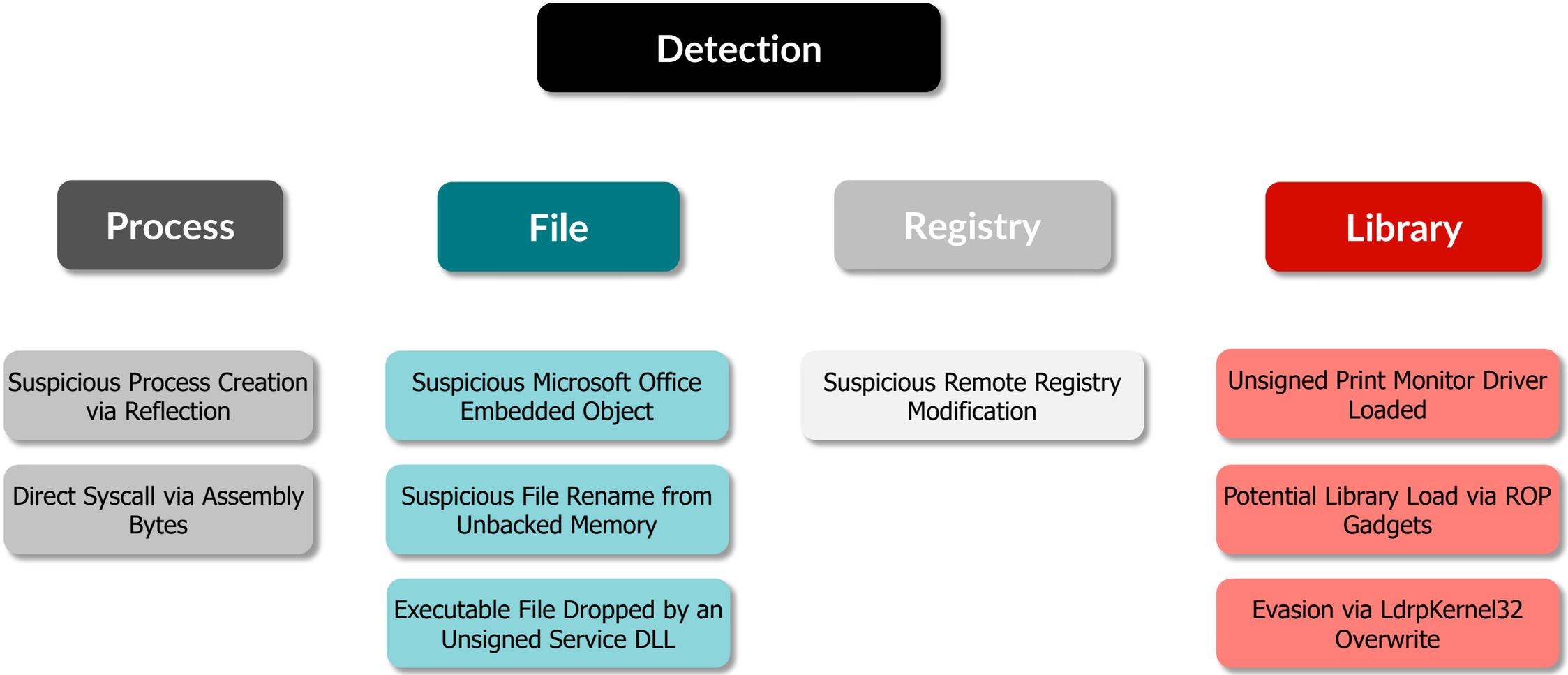


Via Hooking

Every time a specific API or System Call gets called, the call stack is analysed

bsi.

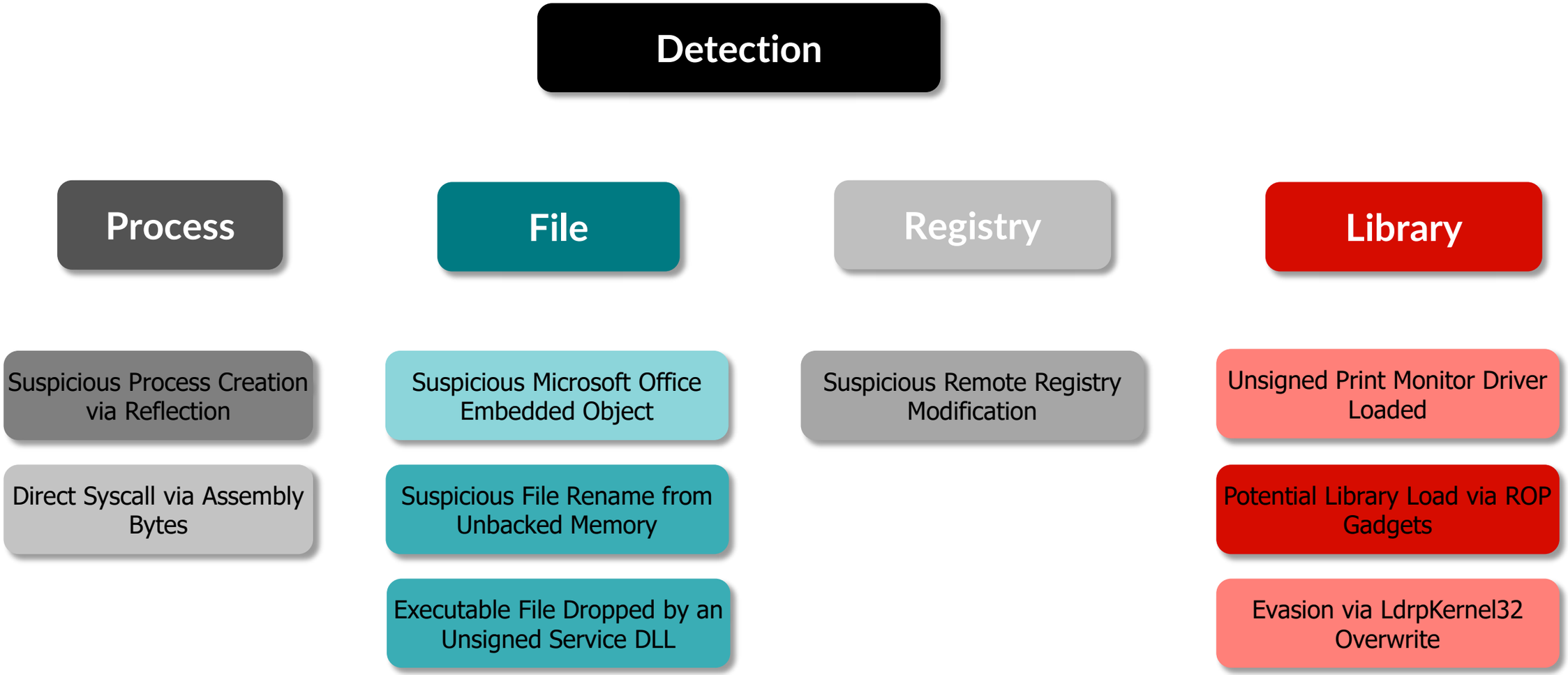
Call stack enriched detection – Elastic EDR



bsi.

https://www.elastic.co/security-labs/peeling-back-the-curtain-with-call-stacks

Call stack enriched detection – Elastic EDR



bsi.

https://www.elastic.co/security-labs/peeling-back-the-curtain-with-call-stacks

bsi.

Thread Stack Spoofing Approaches

History and previous research
on the topic

bsi.

Previous Research

Stack Truncation

This is an extension of return address spoofing, which ensure to zero out the return address of the caller frame

Stack Crafting

Craft the thread call stack artificially, to mimic other legitimate threads when invoking a specific API

Stack Cloning

Uses an external mechanism (timers, APC queues) to hide the thread stack of the in-memory injected code by cloning a legitimate thread stack

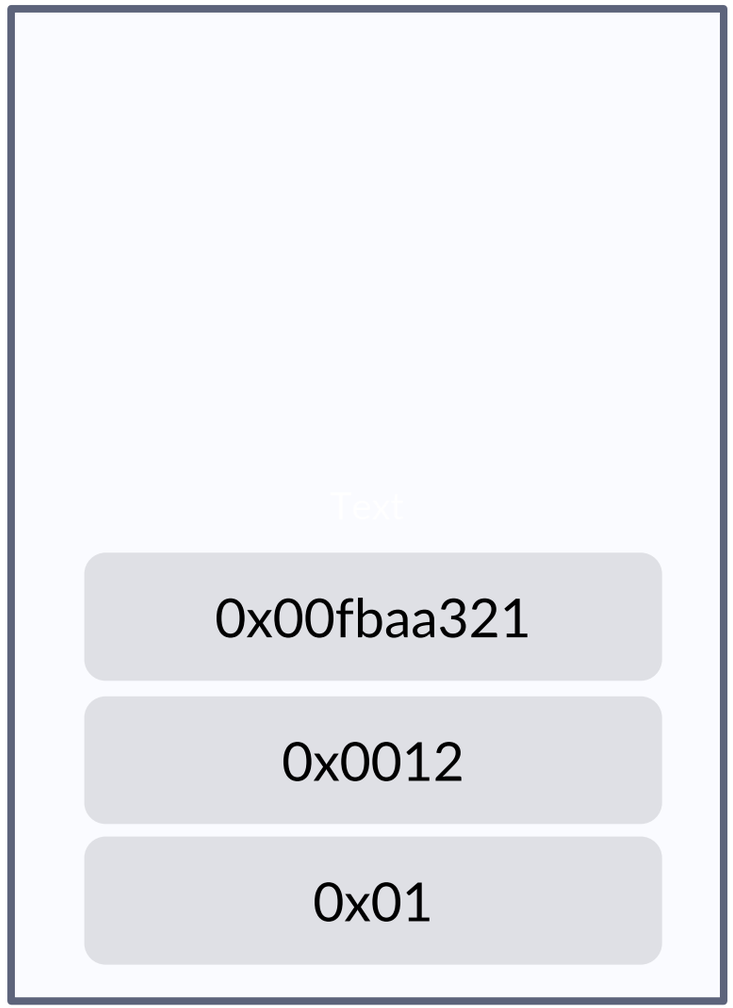
Stack Hiding

Uses Fibers to hide the thread stack of the in-memory injected code

bsi.

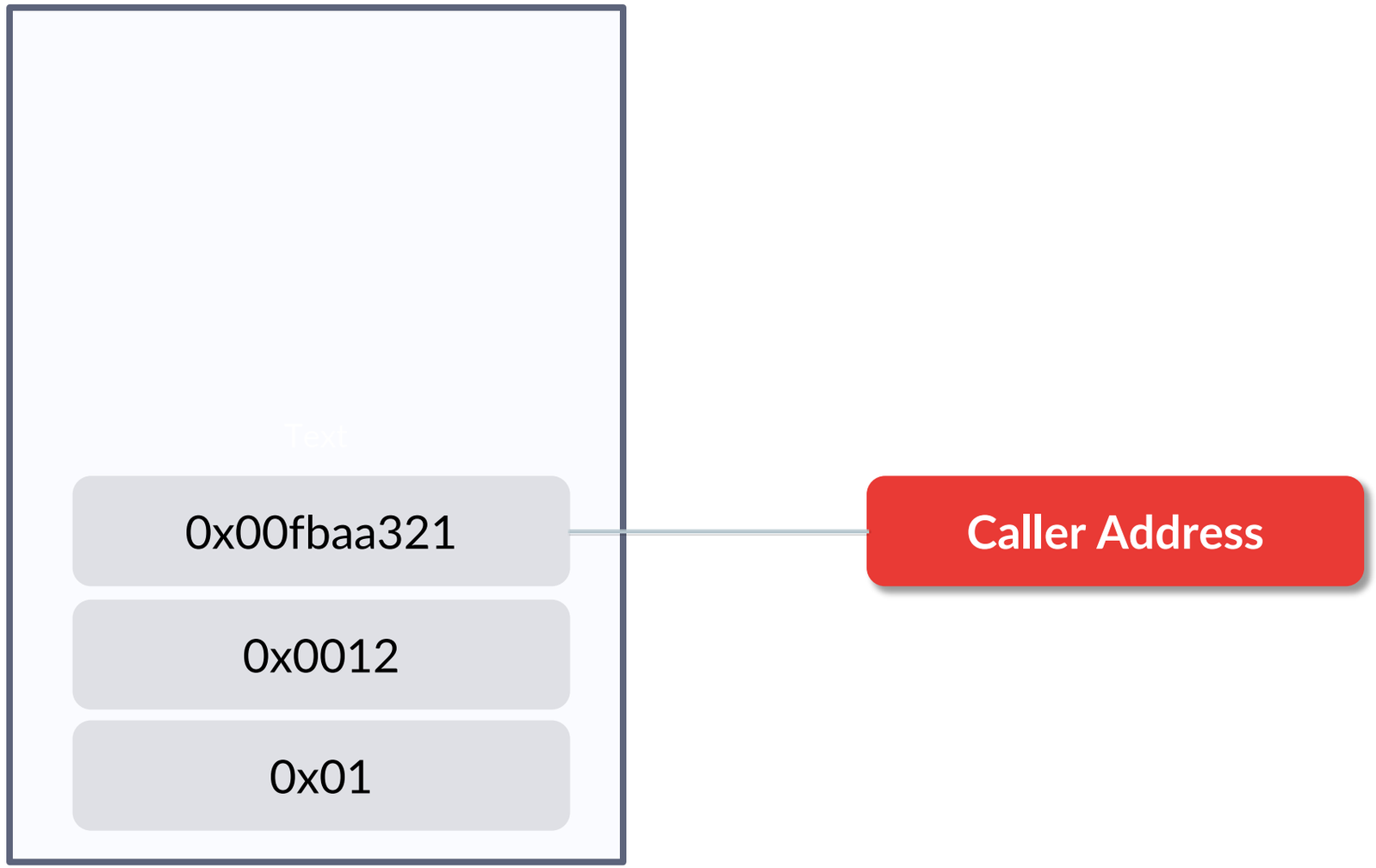
https://github.com/mgeeky/ThreadStackSpoofer

Stack Truncation



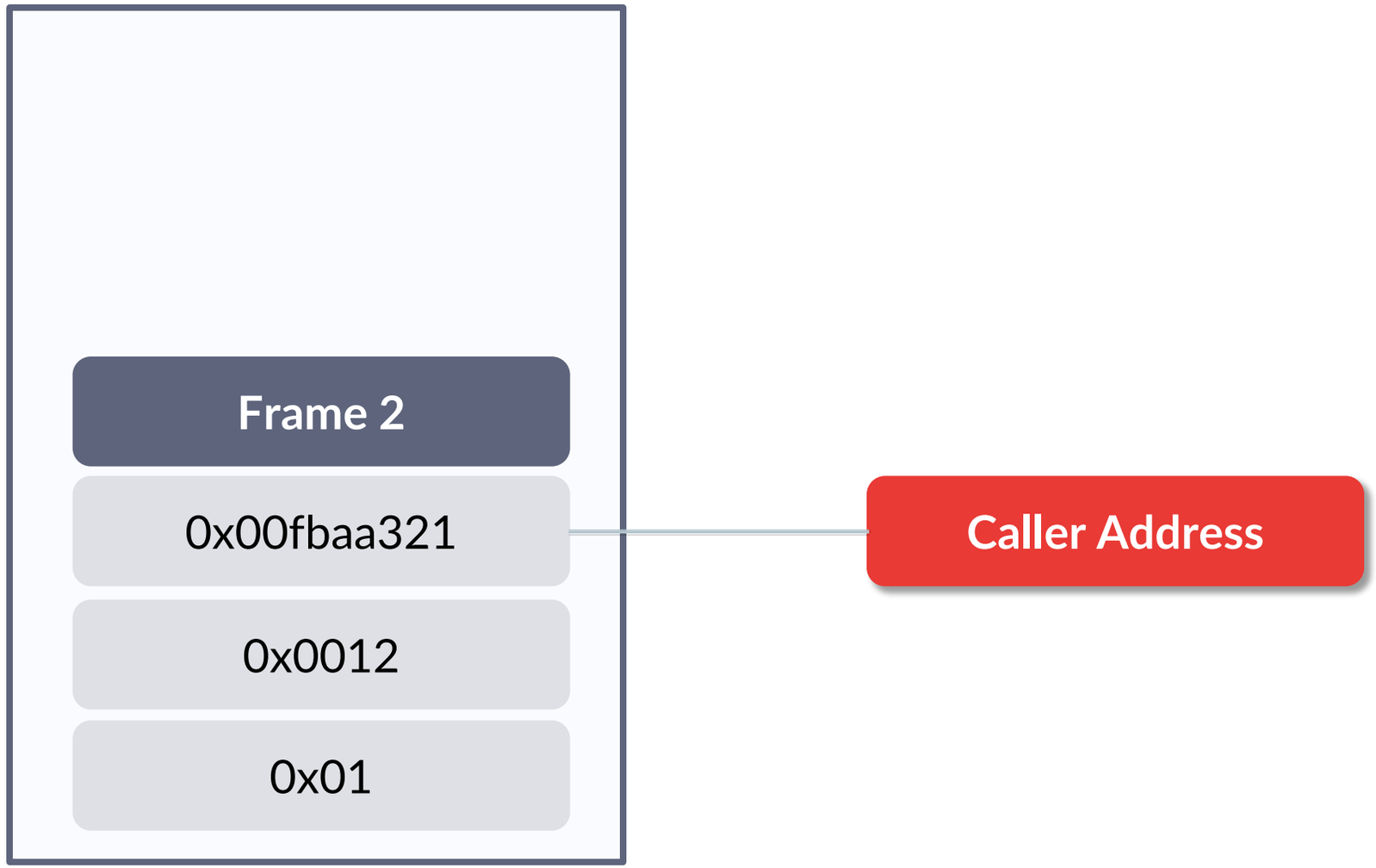
bsi.

Stack Truncation



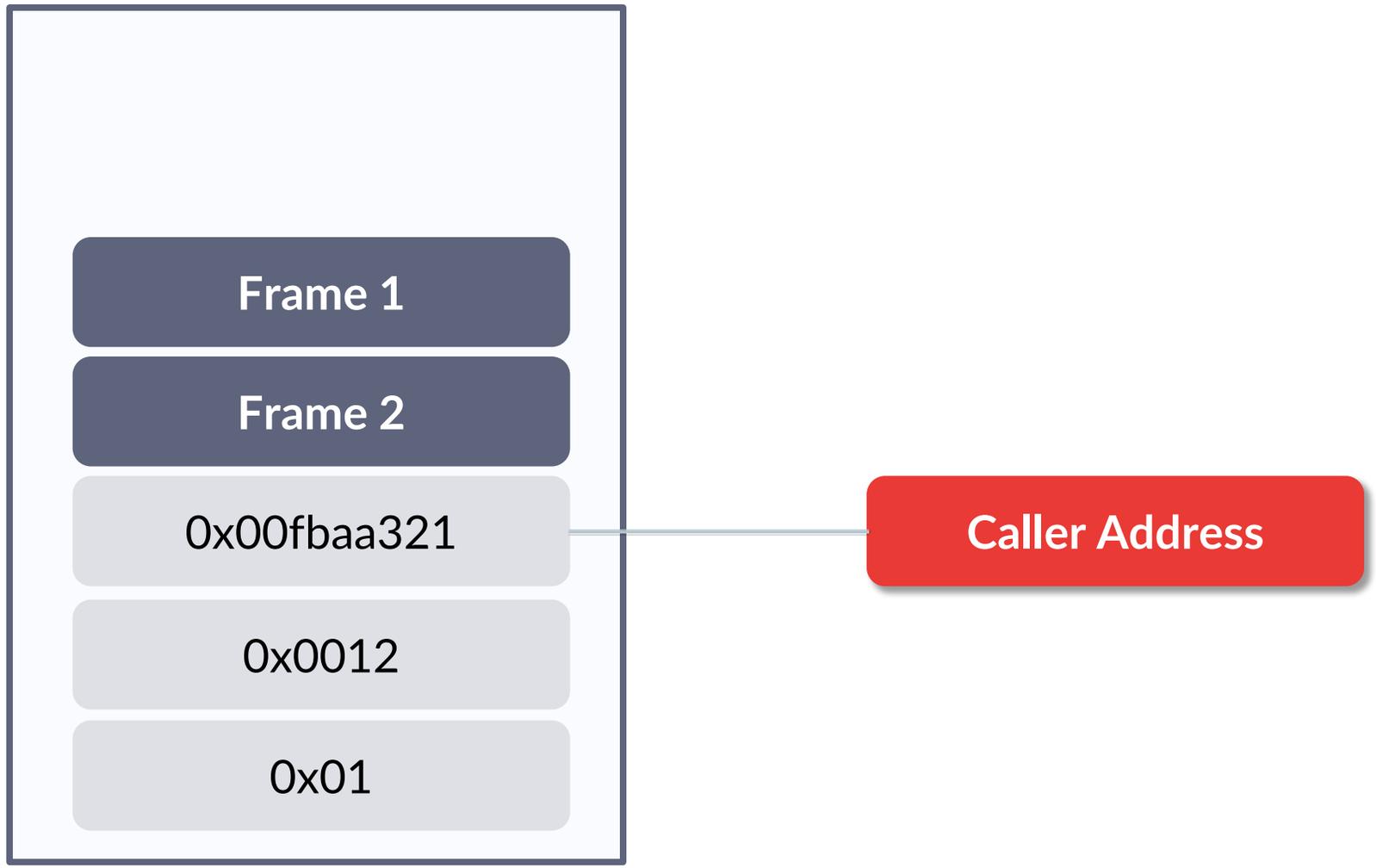
bsi.

Stack Truncation



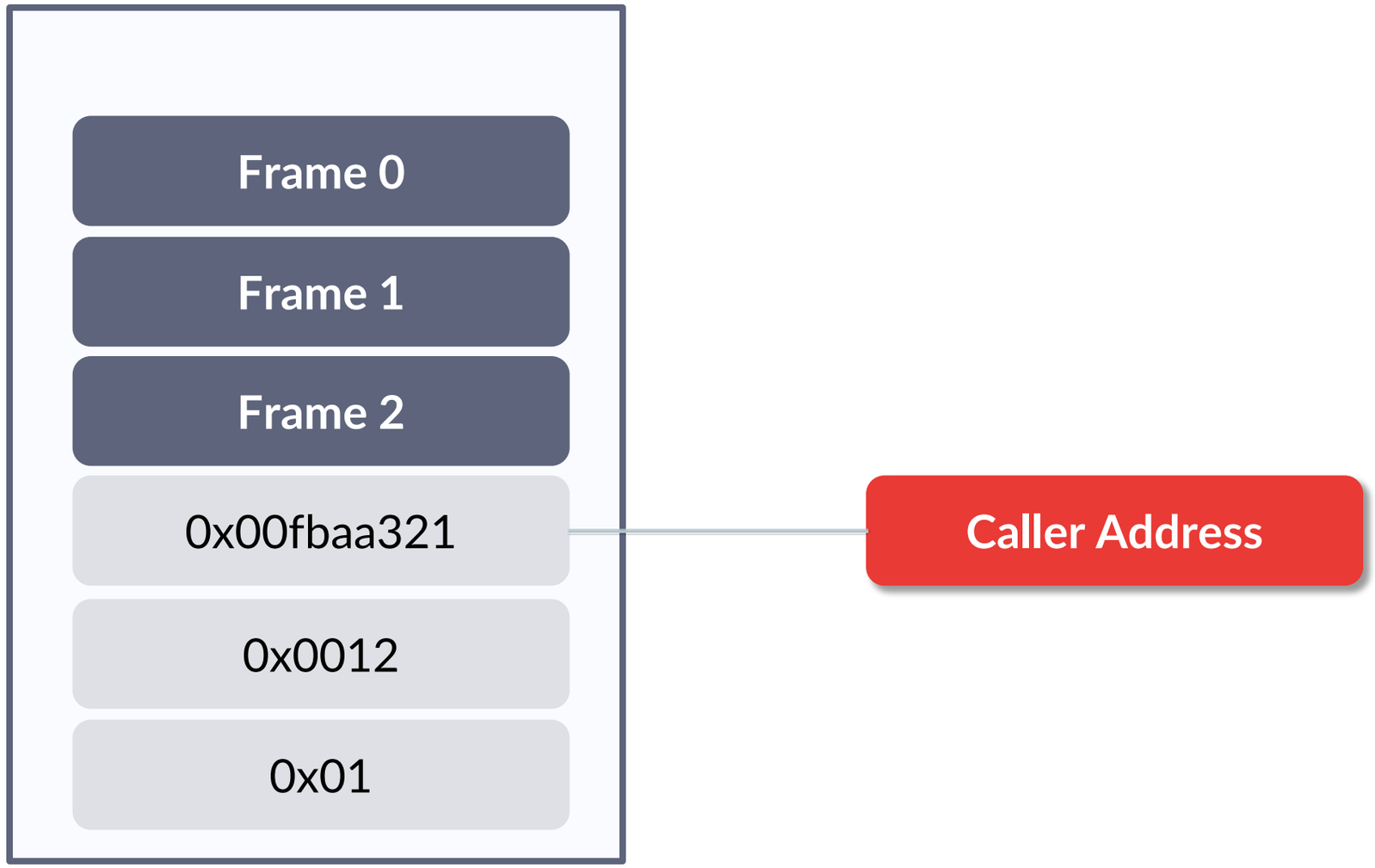
bsi.

Stack Truncation



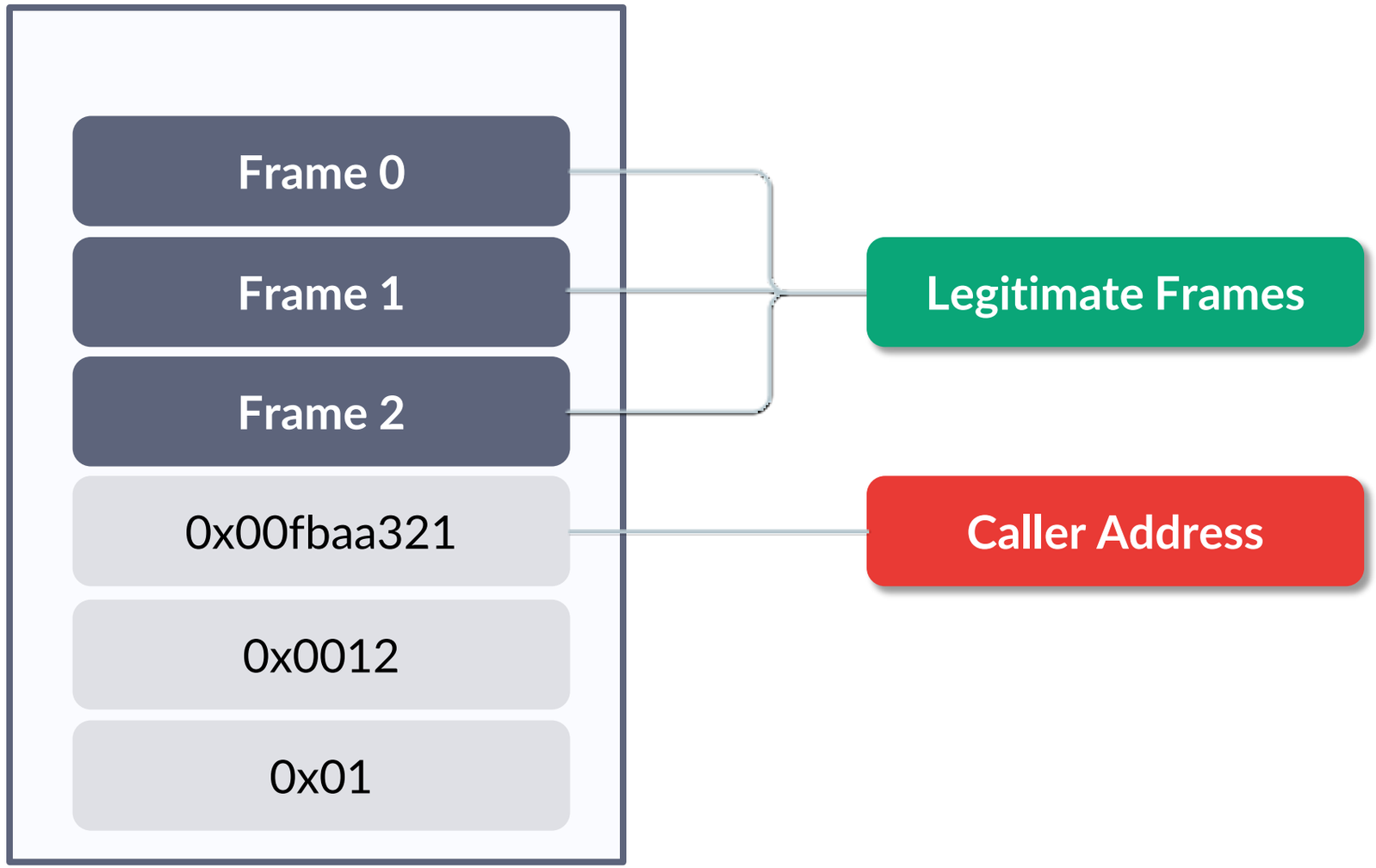
bsi.

Stack Truncation



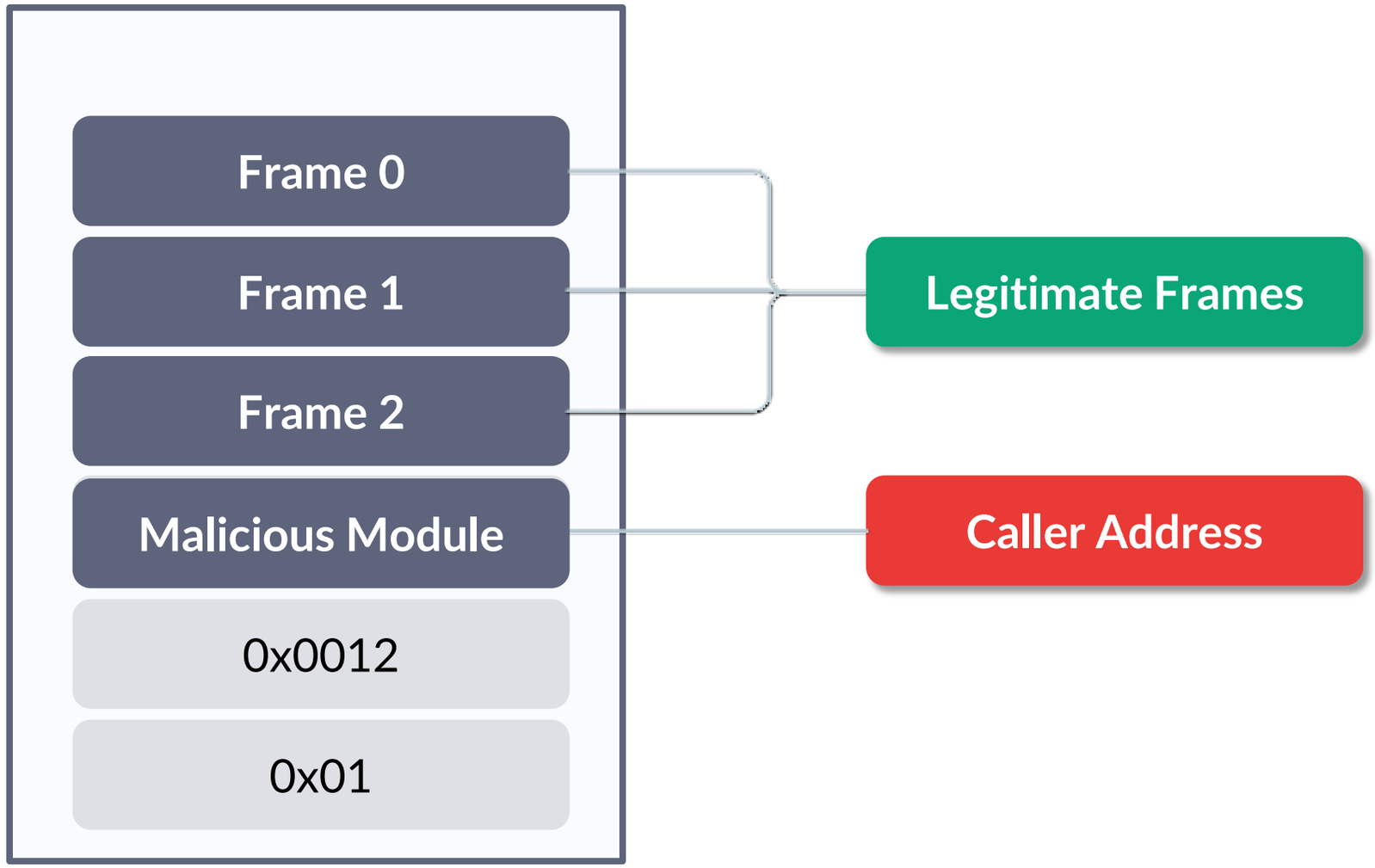
bsi.

Stack Truncation



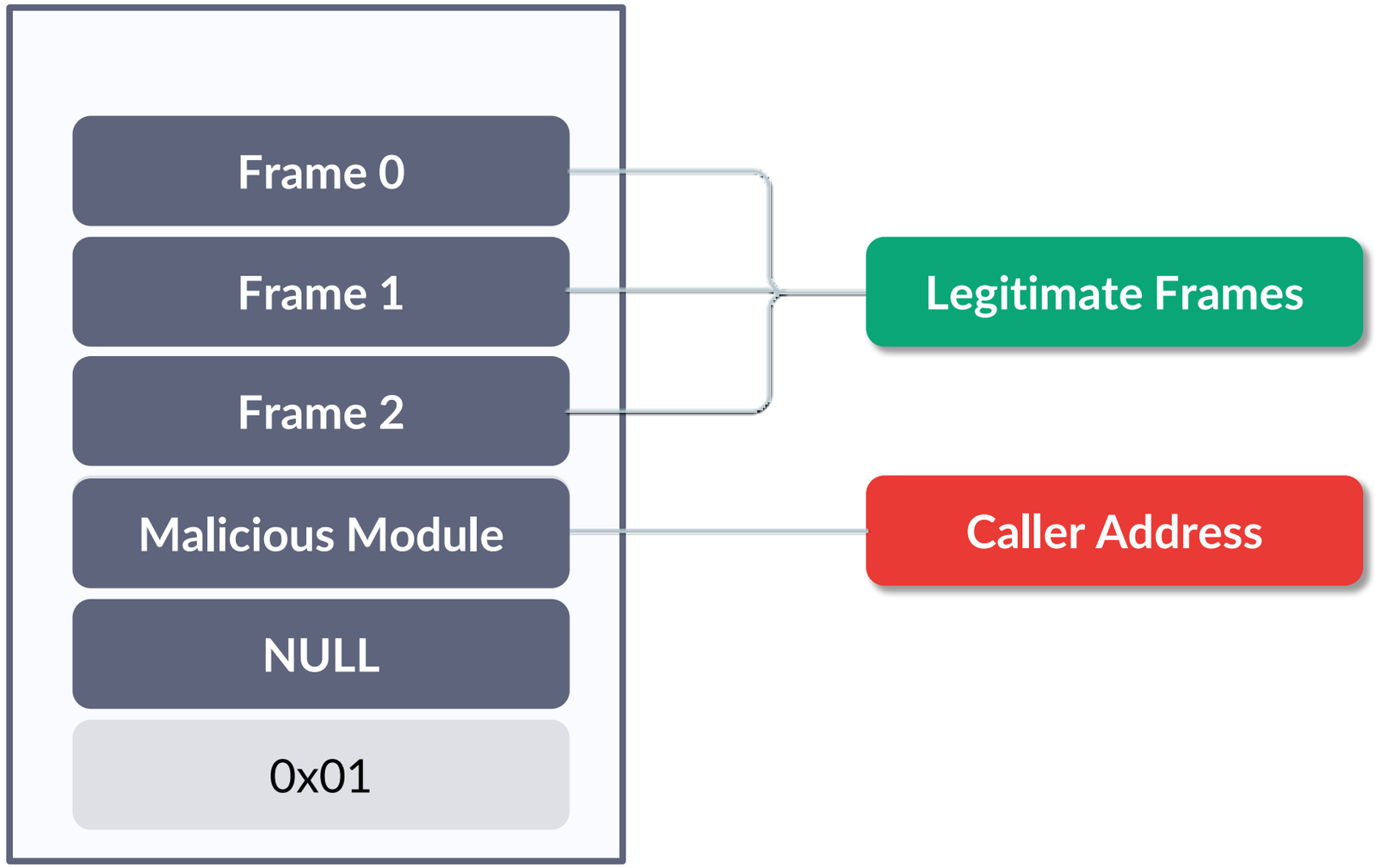
bsi.

Stack Truncation



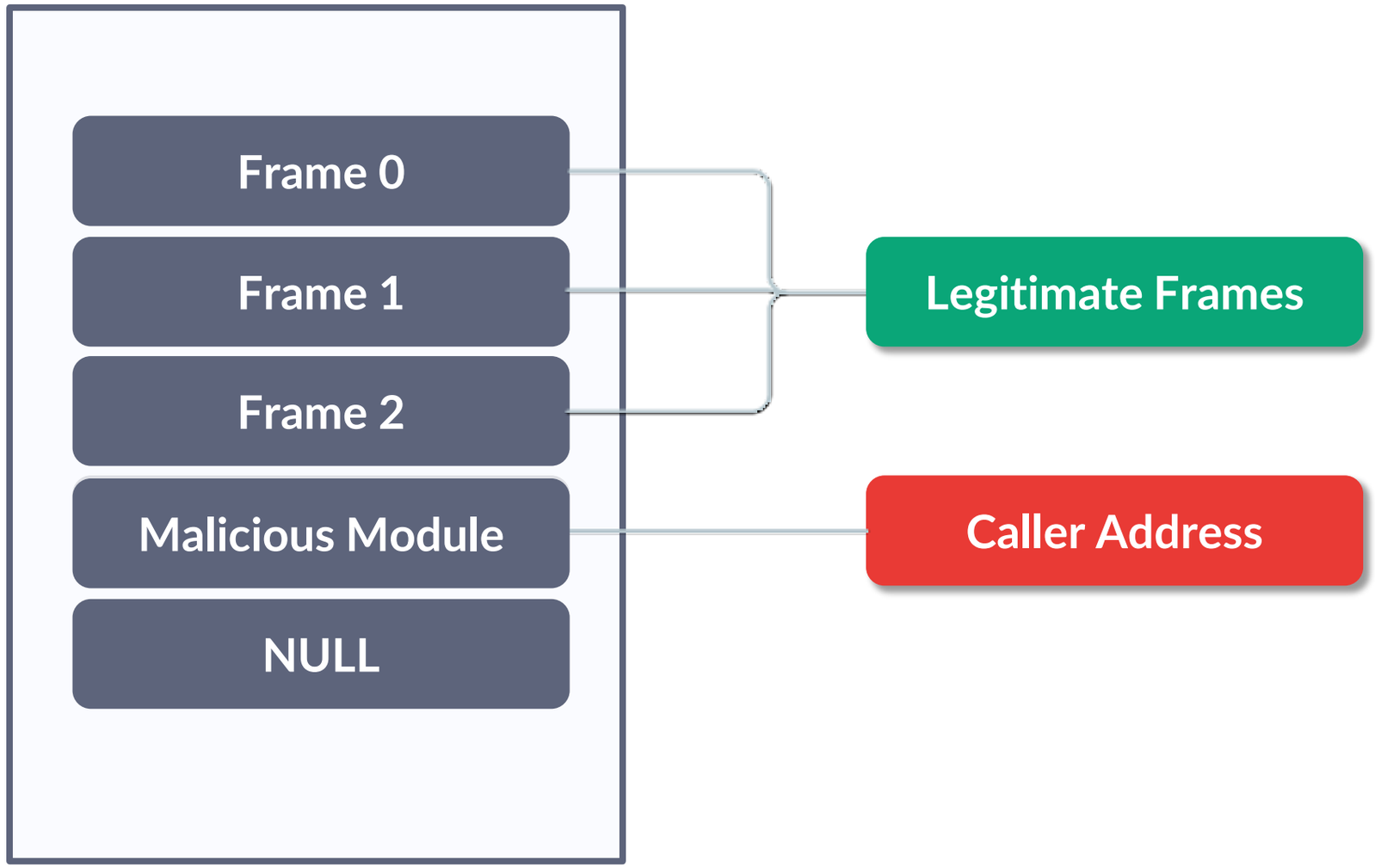
bsi.

Stack Truncation



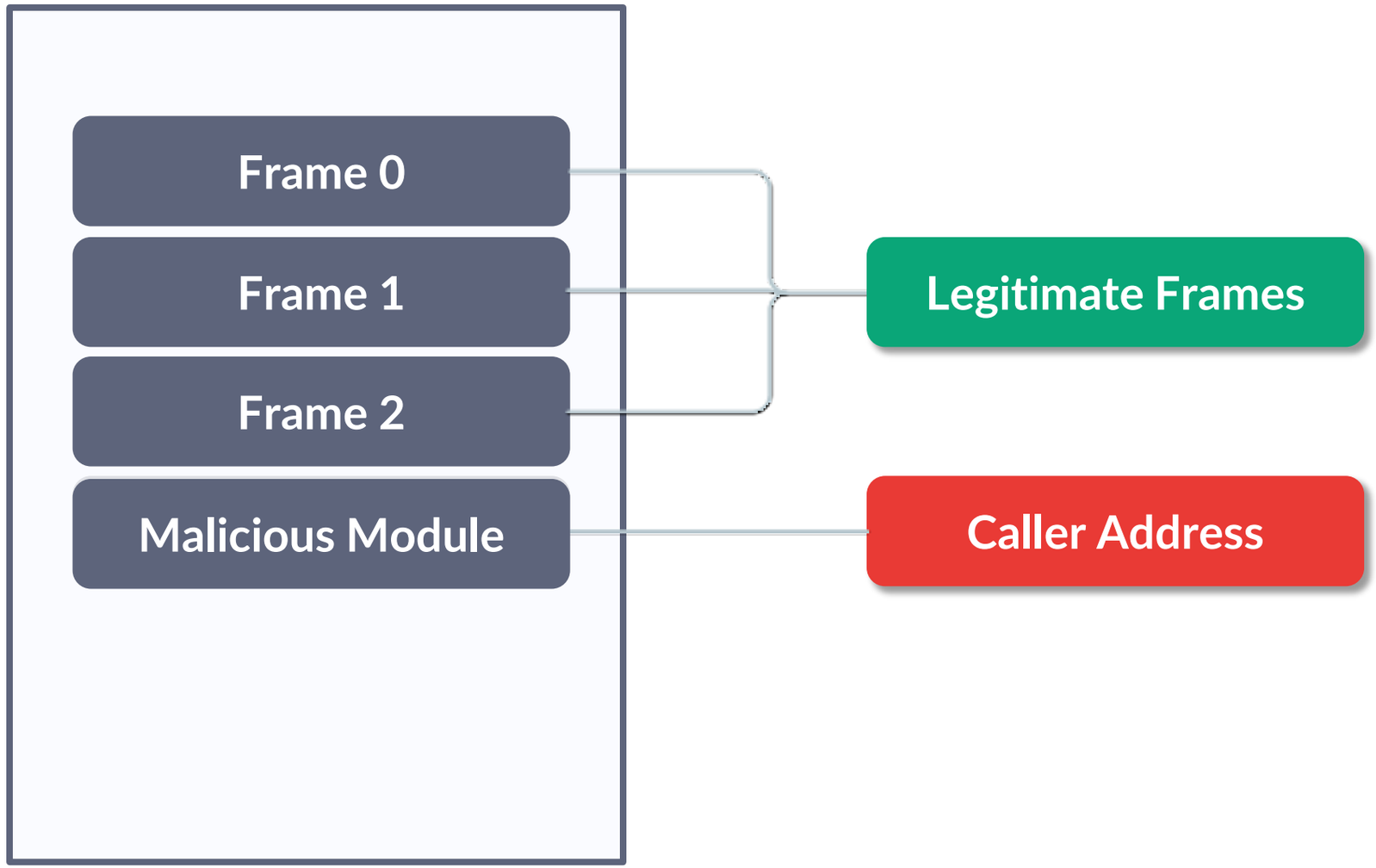
bsi.

Stack Truncation



bsi.

Stack Truncation



bsi.

Stack Truncation

Stack - thread 34356

#	Name	Stack address	Return address	Frame address
0	ntoskrnl.exe!KiDeliverApc+0x1b0			
1	ntoskrnl.exe!KiSwapThread+0x827			
2	ntoskrnl.exe!KiCommitThreadWait+0x14f			
3	ntoskrnl.exe!KeDelayExecutionThread+0x122			
4	ntoskrnl.exe!NtDelayExecution+0x5f			
5	ntoskrnl.exe!KiSystemServiceCopyEnd+0x25			
6	ntdll.dll!NtDelayExecution+0x14	0x88da5ffa98	0x7ffeb65795be	0x88da5ffa90
7	KernelBase.dll!SleepEx+0x9e	0x88da5ffaa0	0x22d6bd5bd51	0x88da5ffb30
8	0x22d6bd5bd51	0x88da5ffb40	0x1388	0x88da5ffb38
9	0x1388	0x88da5ffb48	0x22d00000000	0x88da5ffb40
10	0x22d00000000	0x88da5ffb50	0x1b0001c00000bb	0x88da5ffb48
11	0x1b0001c00000bb	0x88da5ffb58		0x88da5ffb50

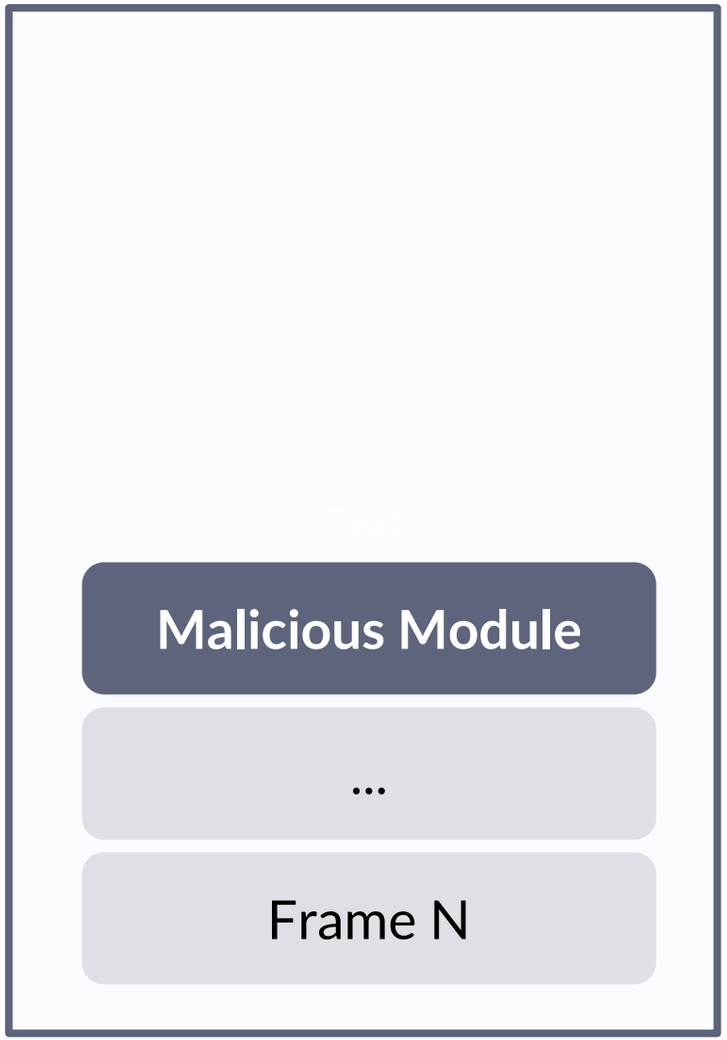
Stack - thread 45956

#	Name	Stack address	Frame address	Return address
0	ntoskrnl.exe!KiDeliverApc+0x1b0			
1	ntoskrnl.exe!KiSwapThread+0x827			
2	ntoskrnl.exe!KiCommitThreadWait+0x14f			
3	ntoskrnl.exe!KeDelayExecutionThread+0x122			
4	ntoskrnl.exe!NtDelayExecution+0x5f			
5	ntoskrnl.exe!KiSystemServiceCopyEnd+0x25			
6	ntdll.dll!NtDelayExecution+0x14	0x3211ff4d8	0x3211ff4d0	0x7ffeb65795be
7	KernelBase.dll!SleepEx+0x9e	0x3211ff4e0	0x3211ff570	0x7ff79a49125c
8	ThreadStackSpoofer.exe!MySleep+0x5c	0x3211ff580	0x3211ff5d0	

bsi.

https://github.com/mgeeky/ThreadStackSpoofer

Stack Crafting

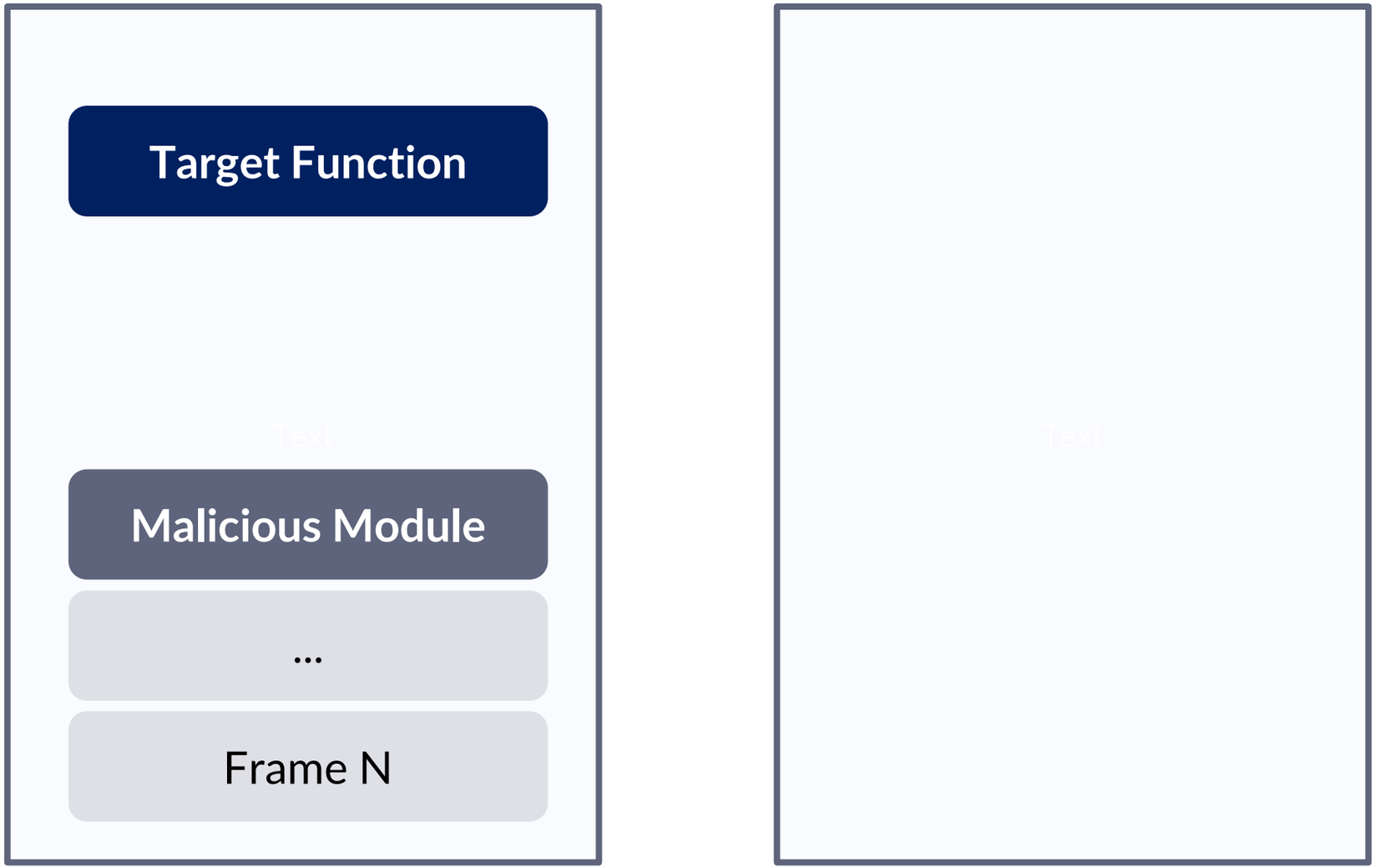


bsi.

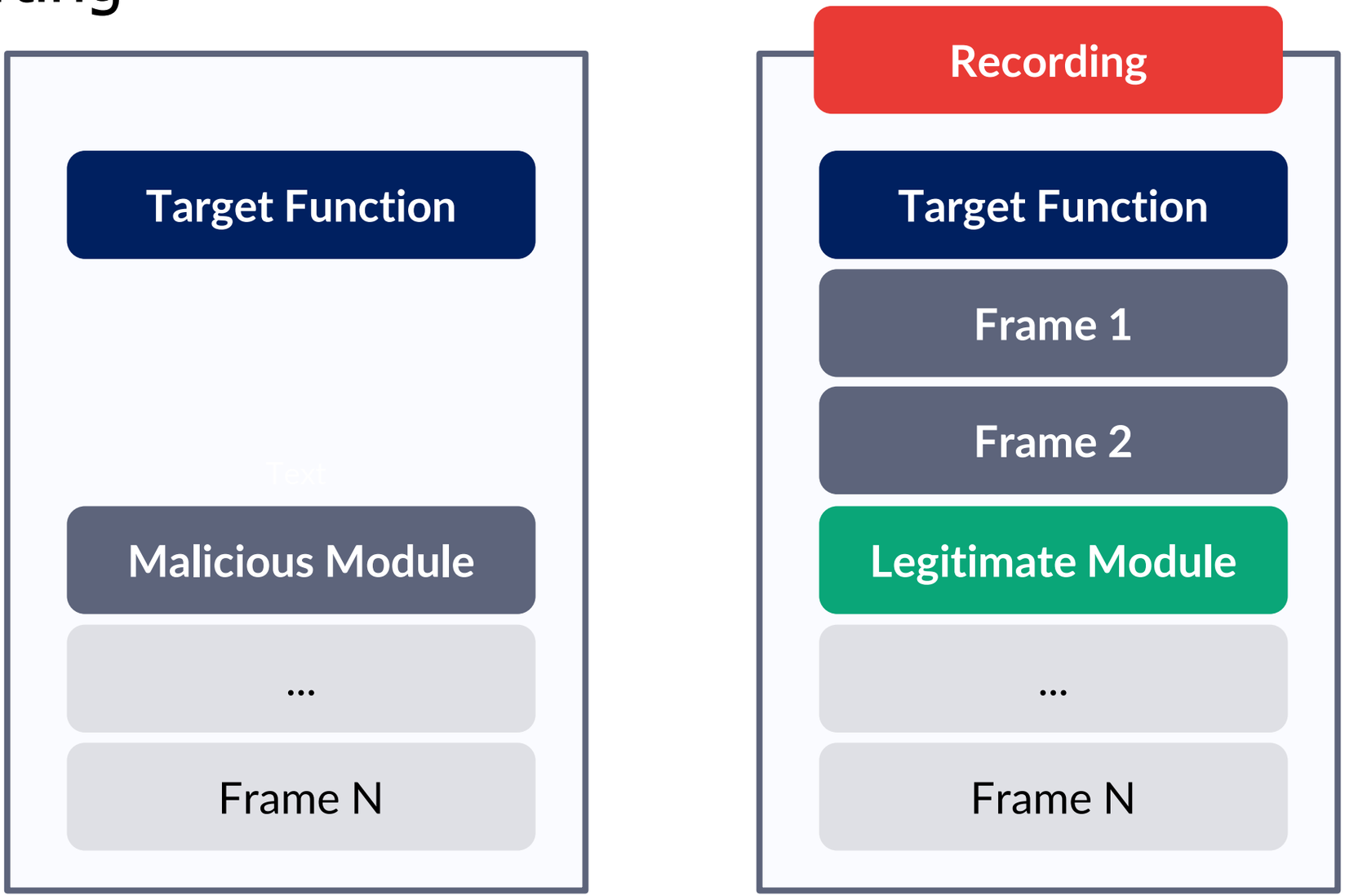
Stack Crafting

bsi.

Stack Crafting

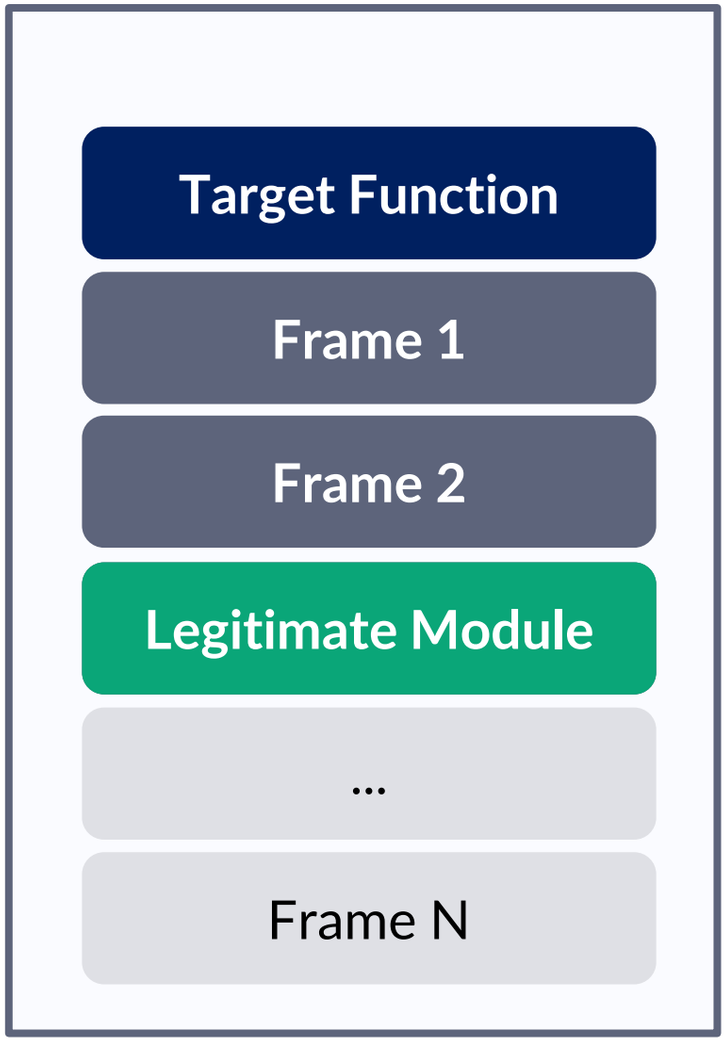


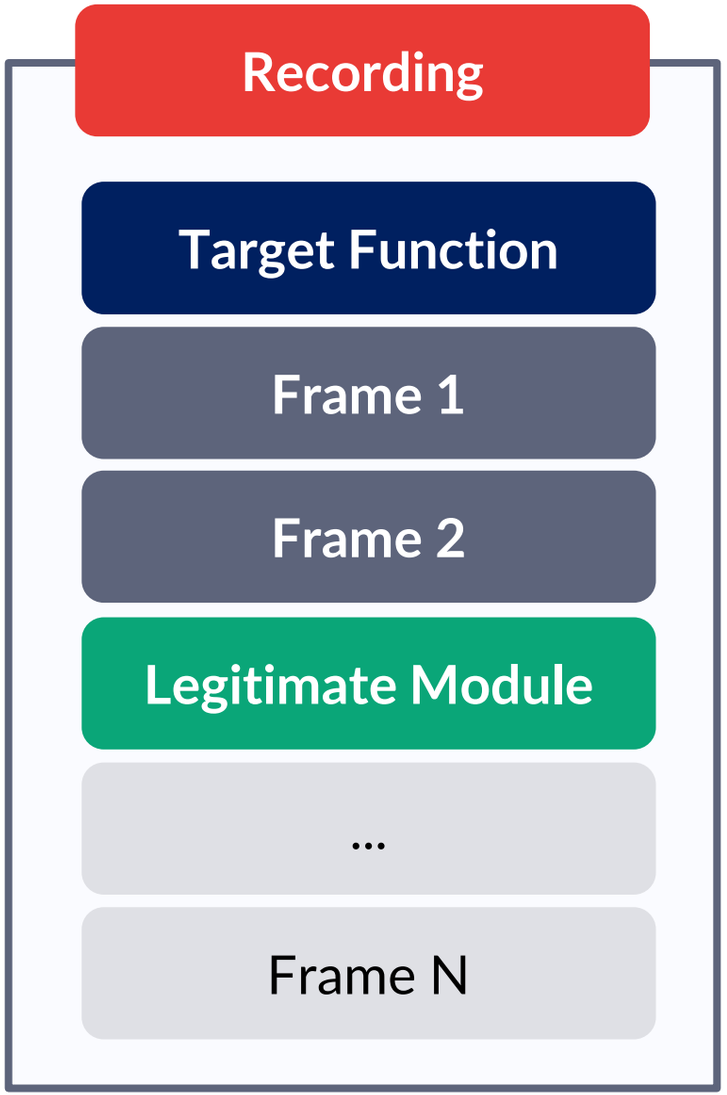
Stack Crafting



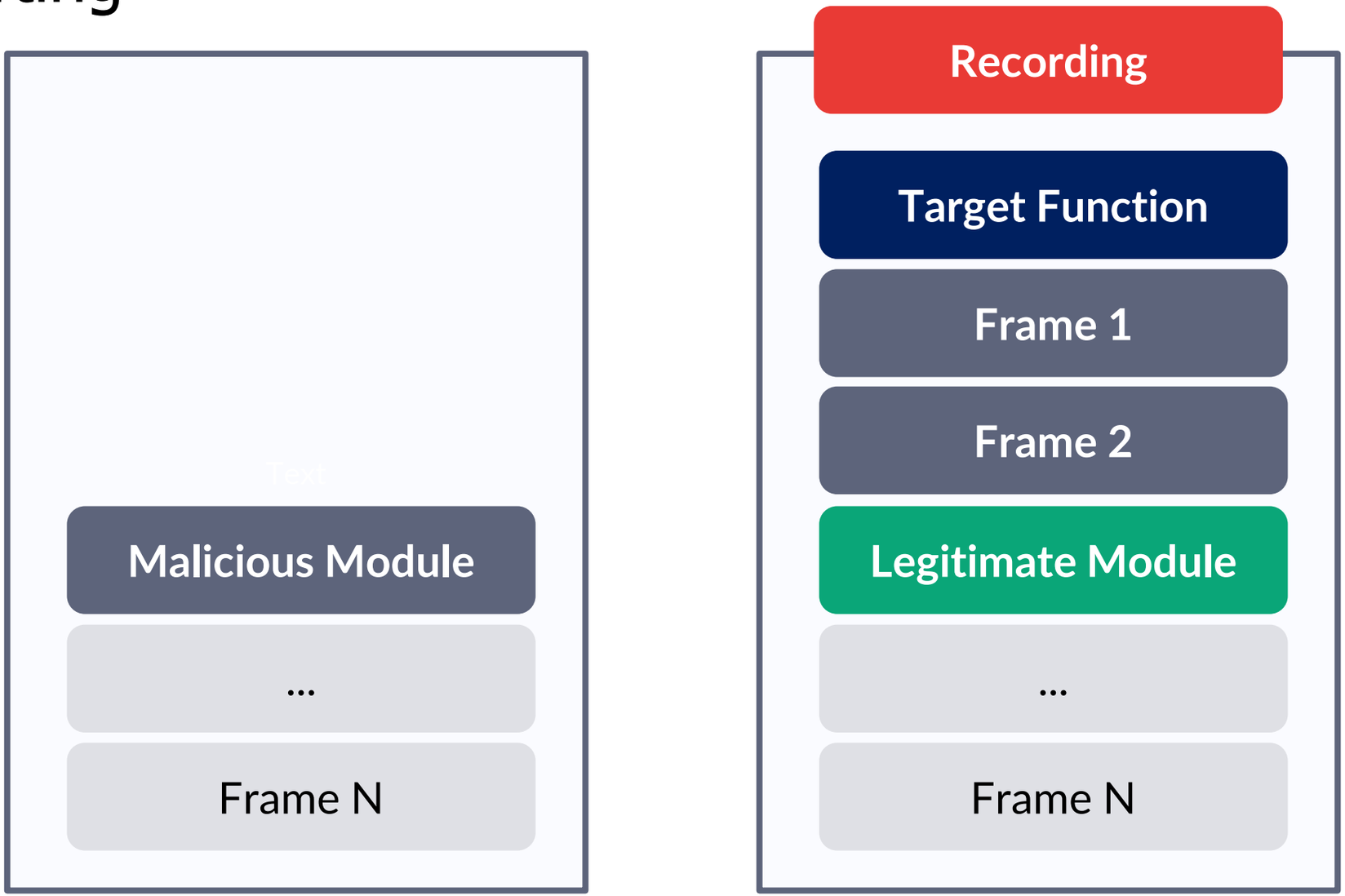
bsi.

Stack Crafting



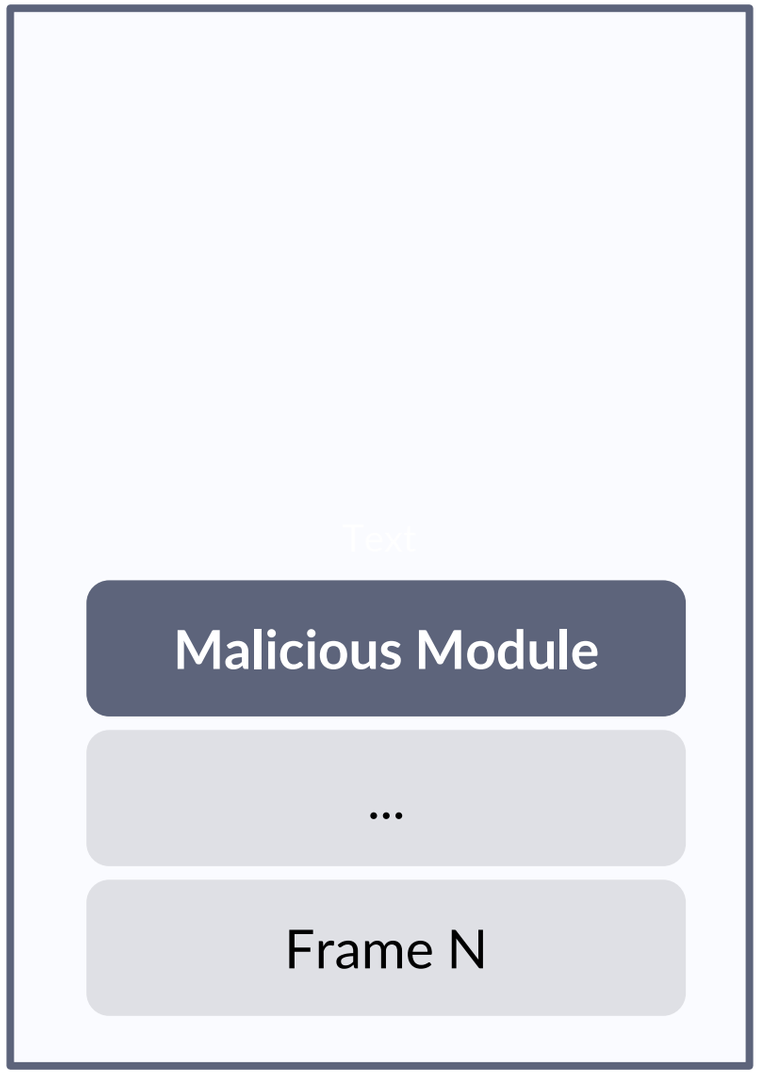


Stack Crafting



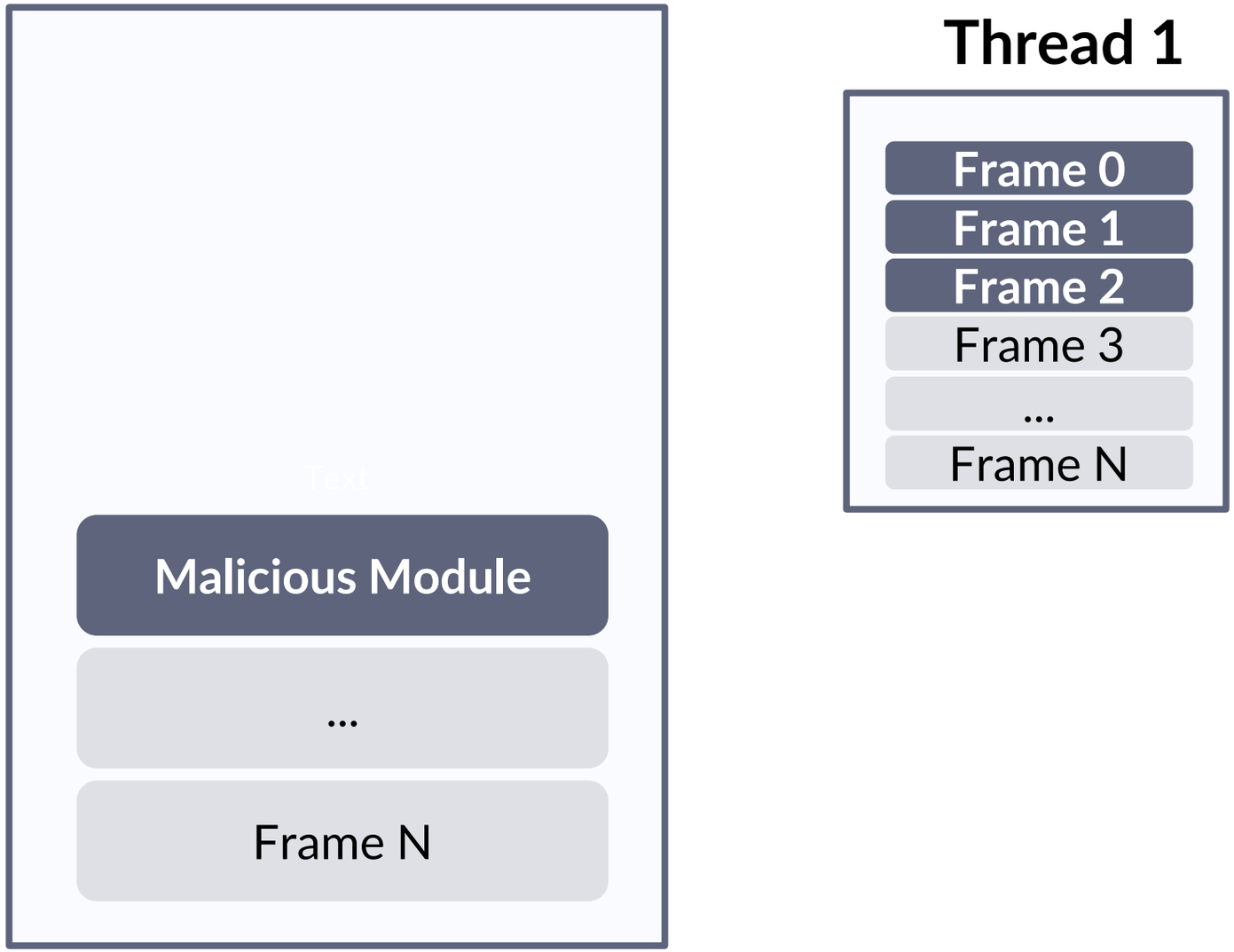
bsi.

Stack Cloning



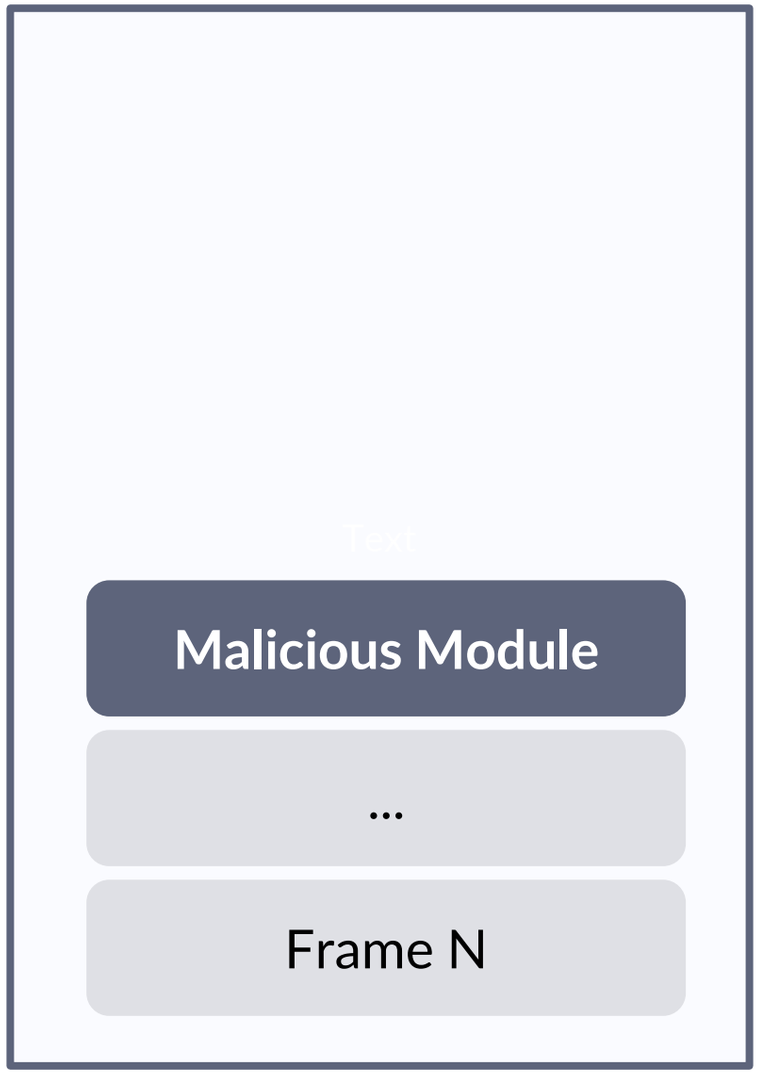
bsi.

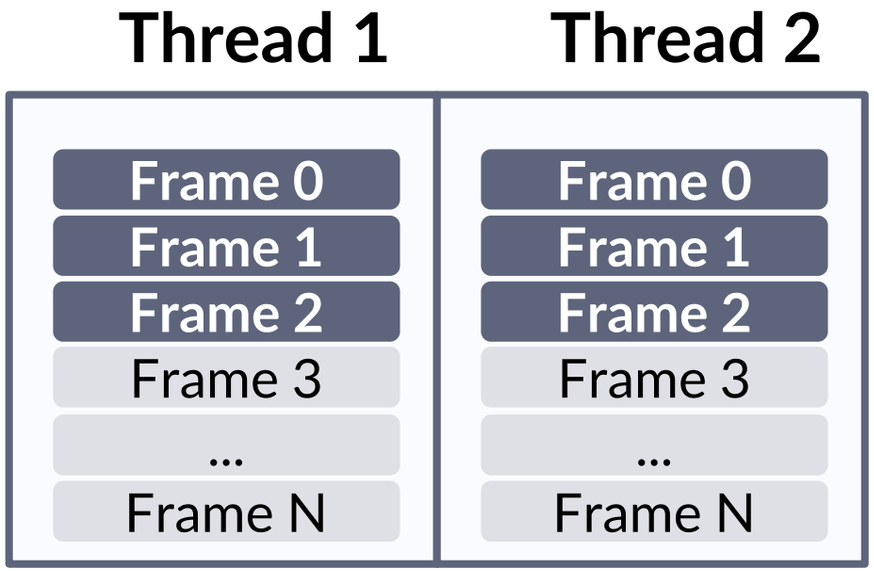
Stack Cloning



bsi.

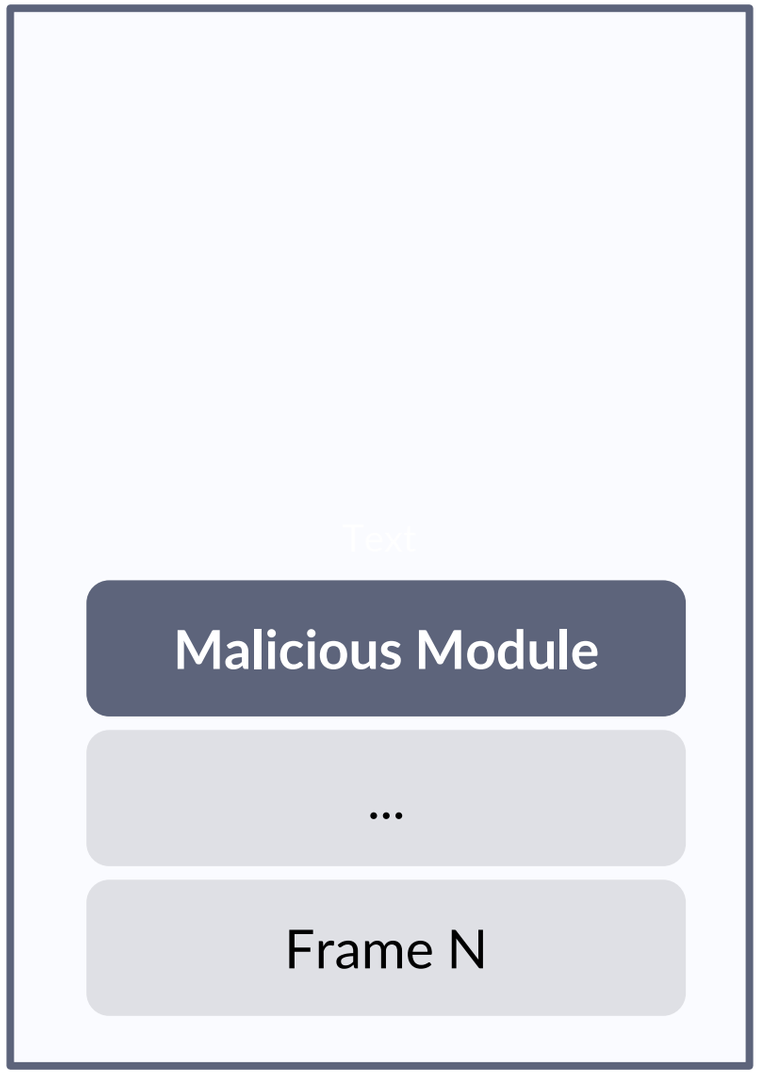
Stack Cloning

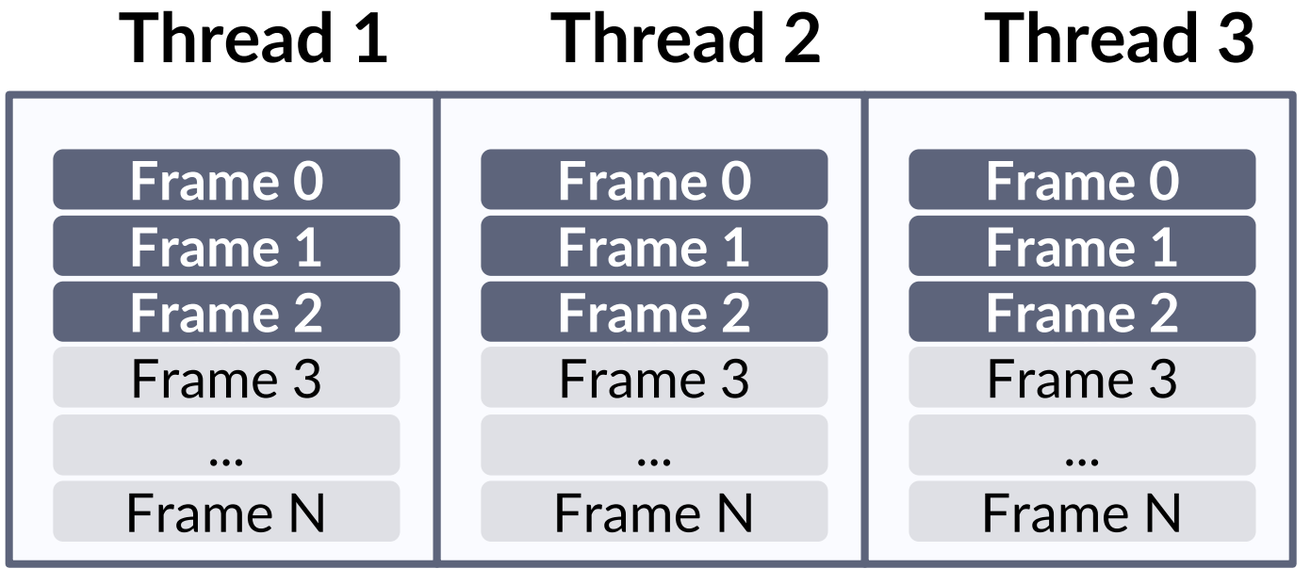




bsi.

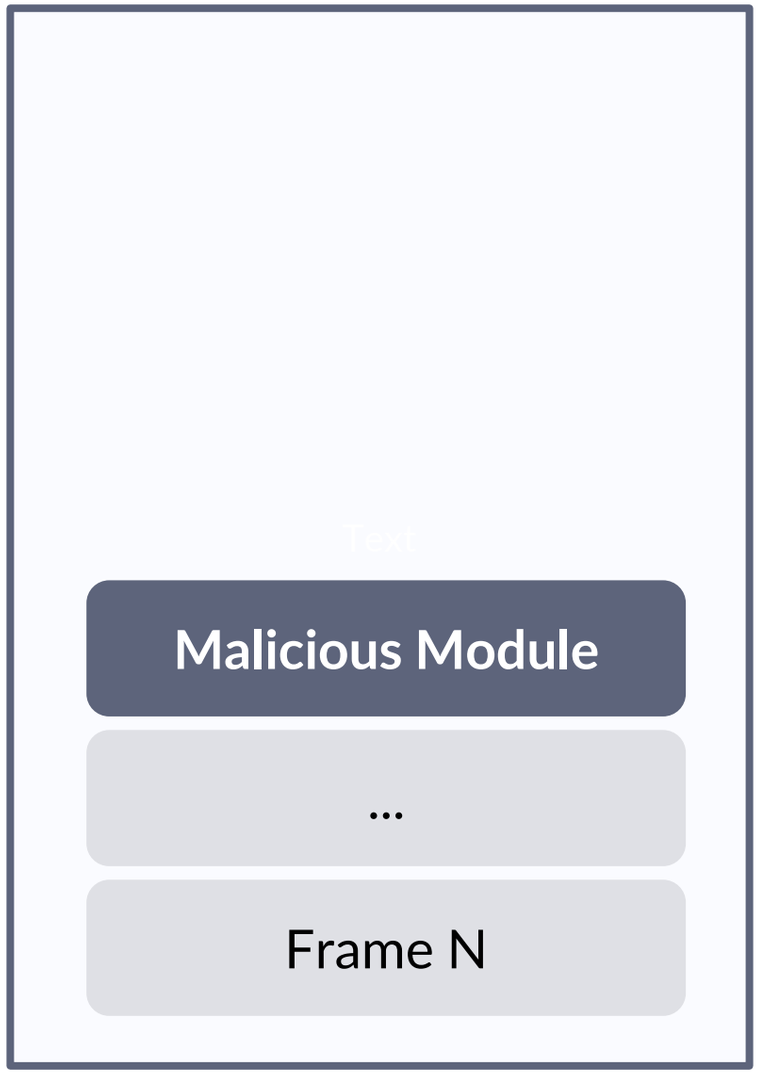
Stack Cloning

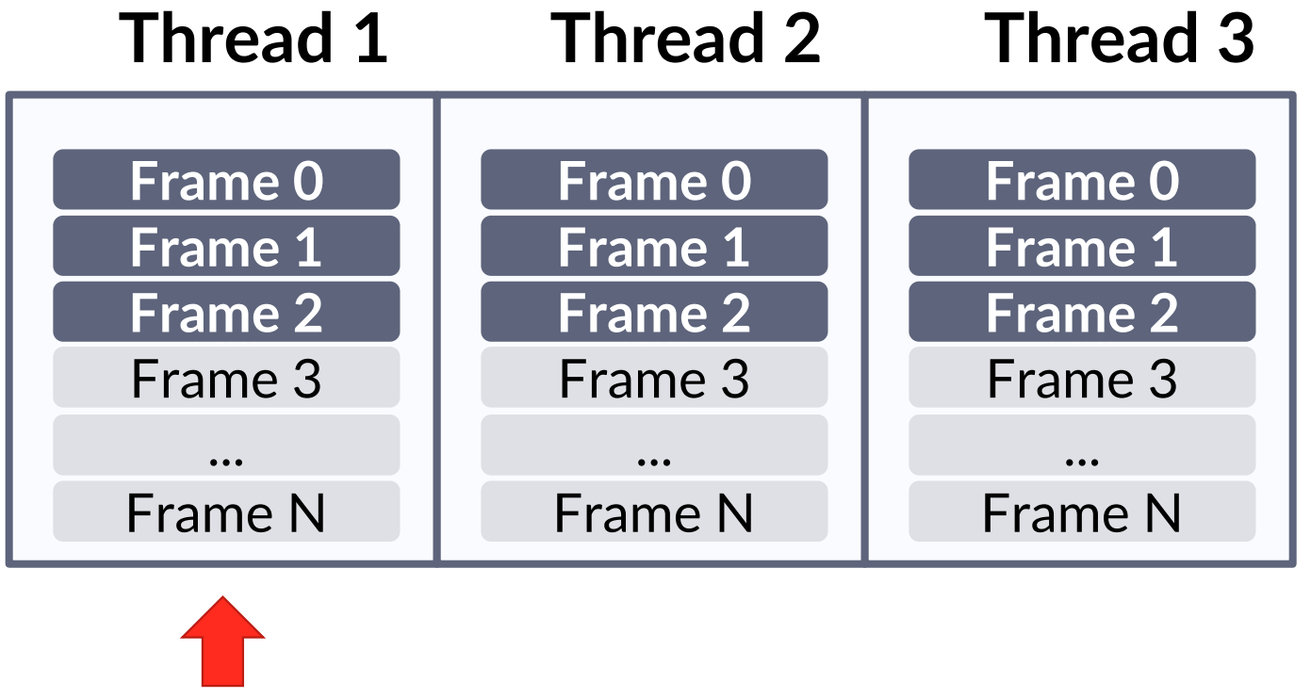




bsi.

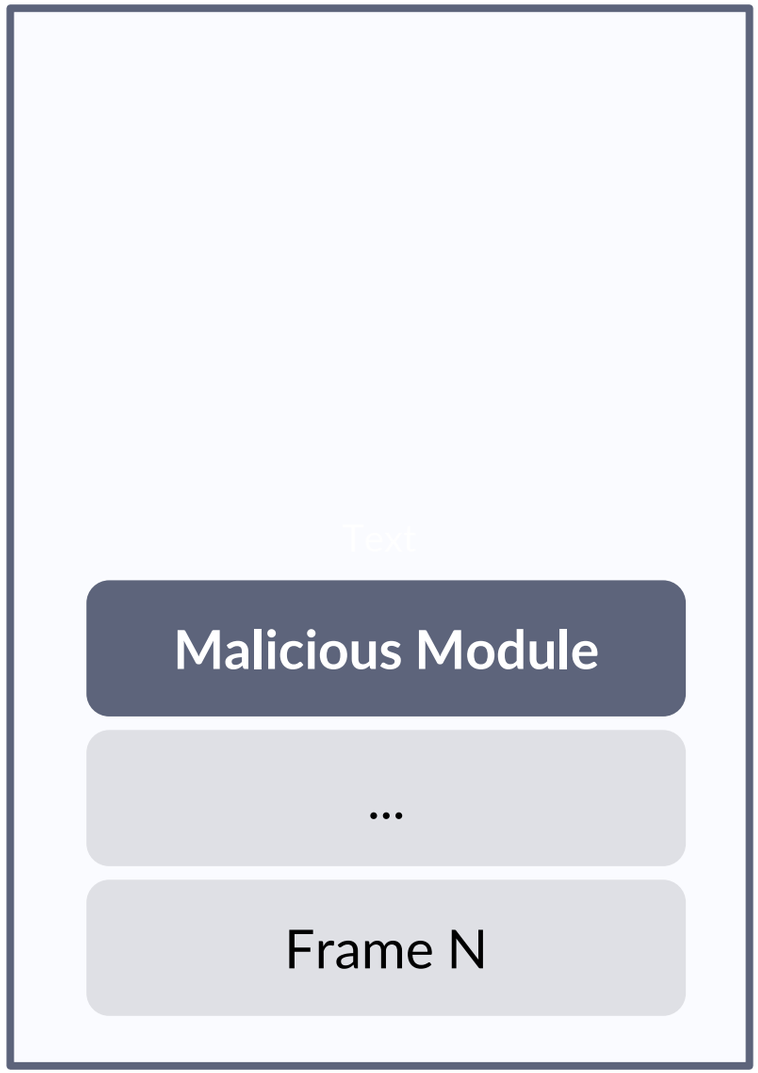
Stack Cloning

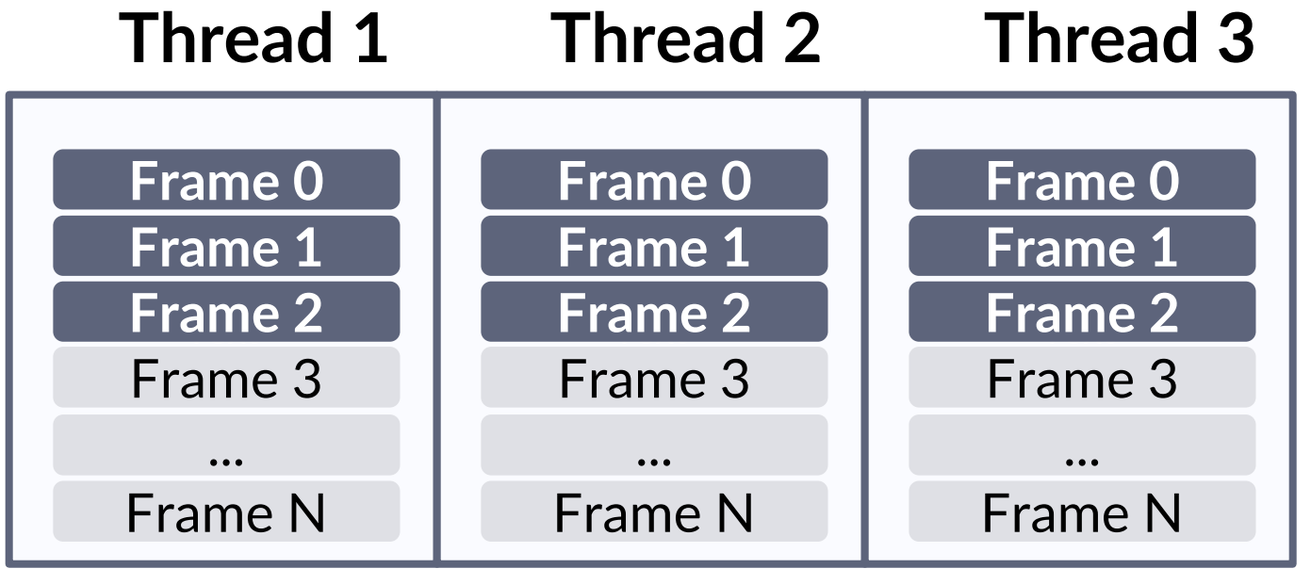




bsi.

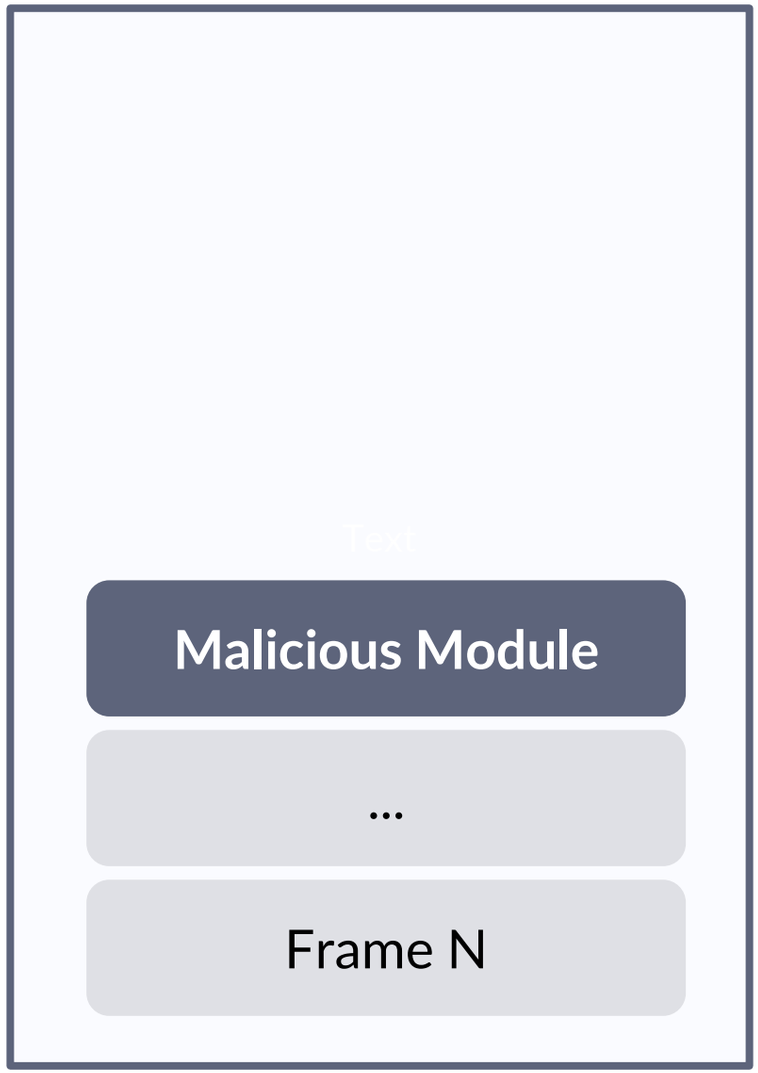
Stack Cloning

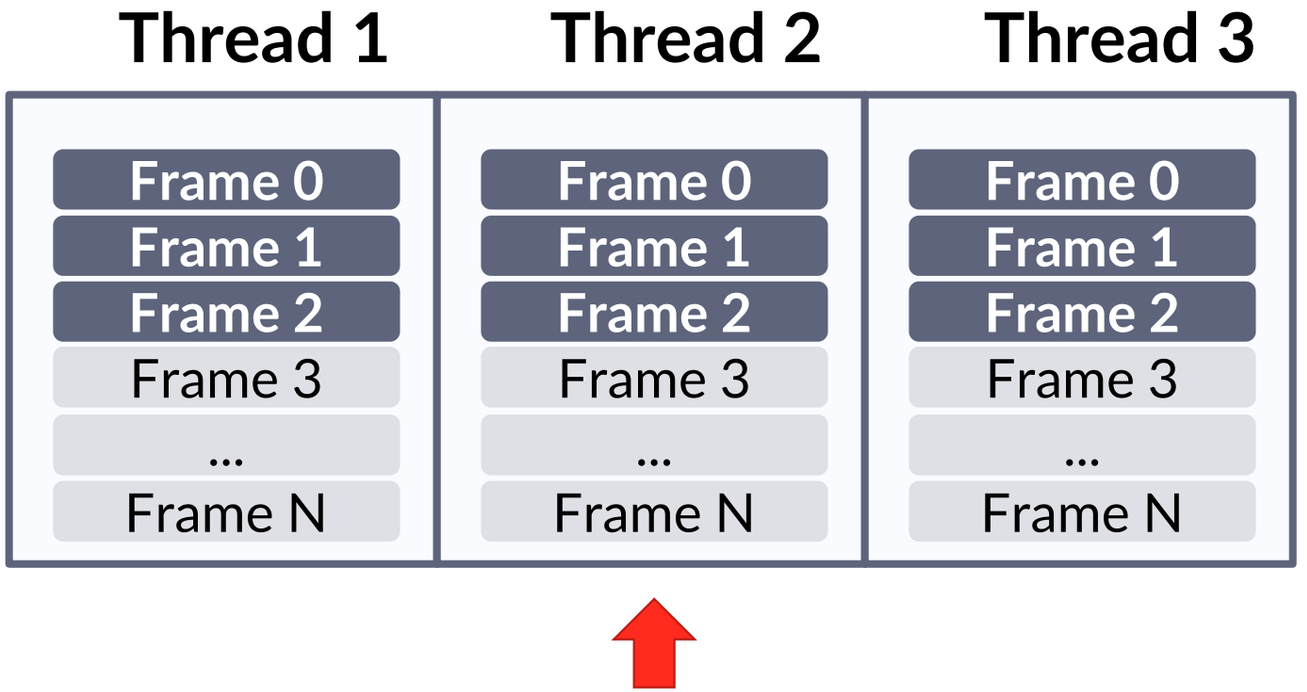




bsi.

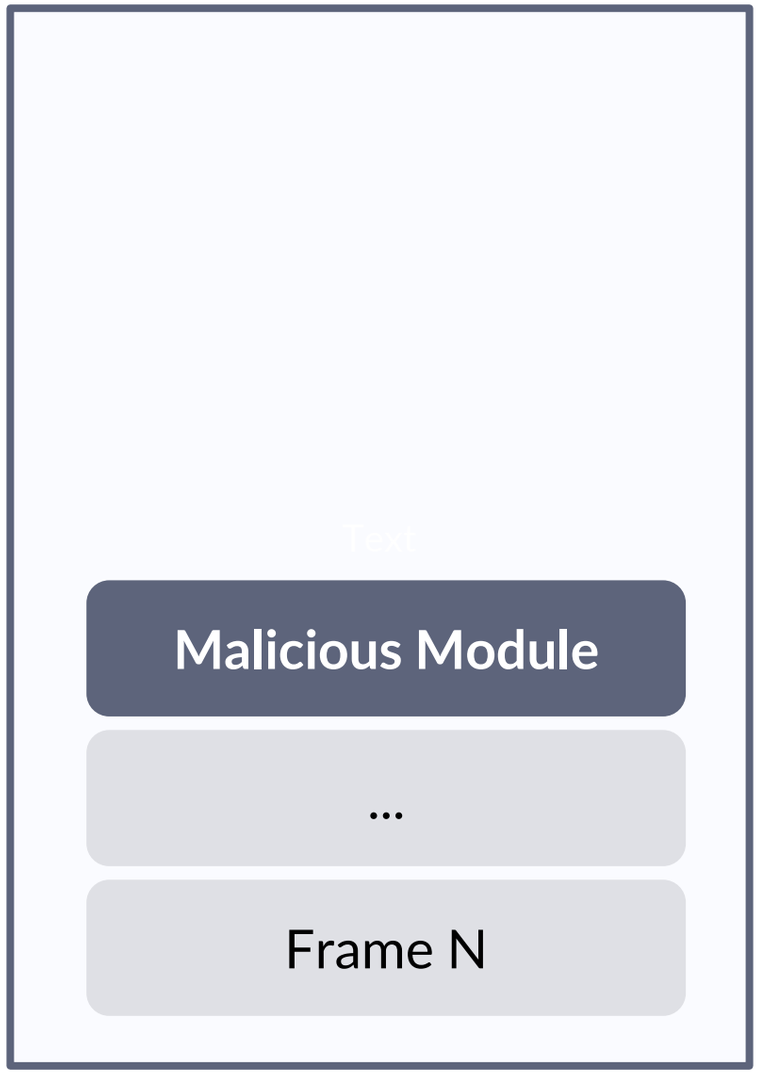
Stack Cloning

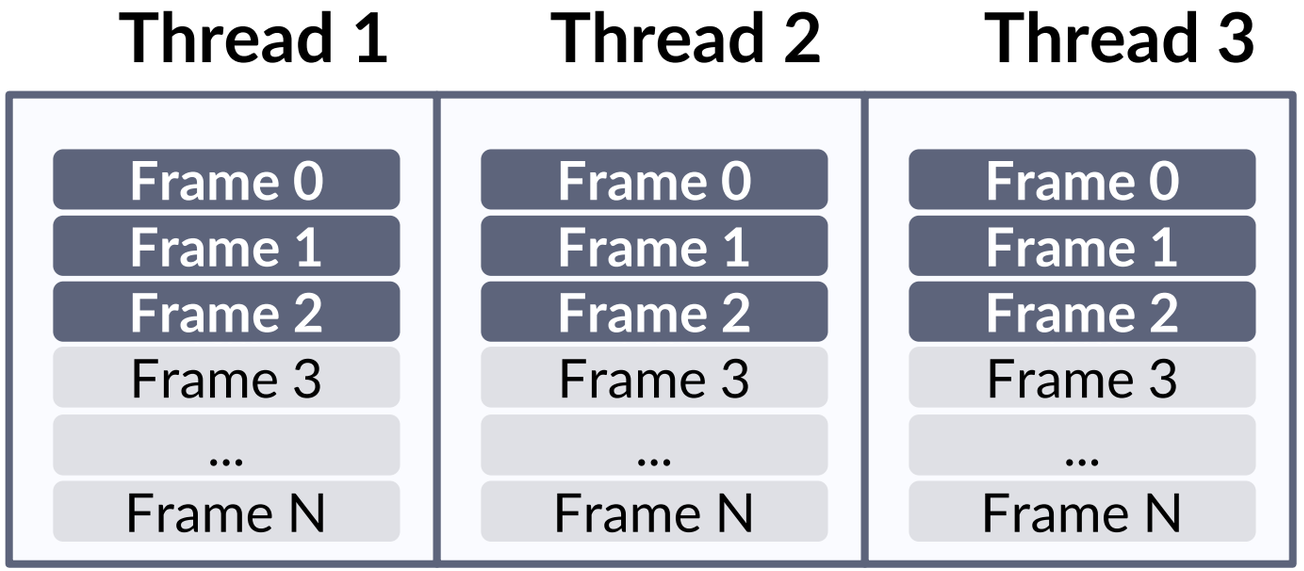




bsi.

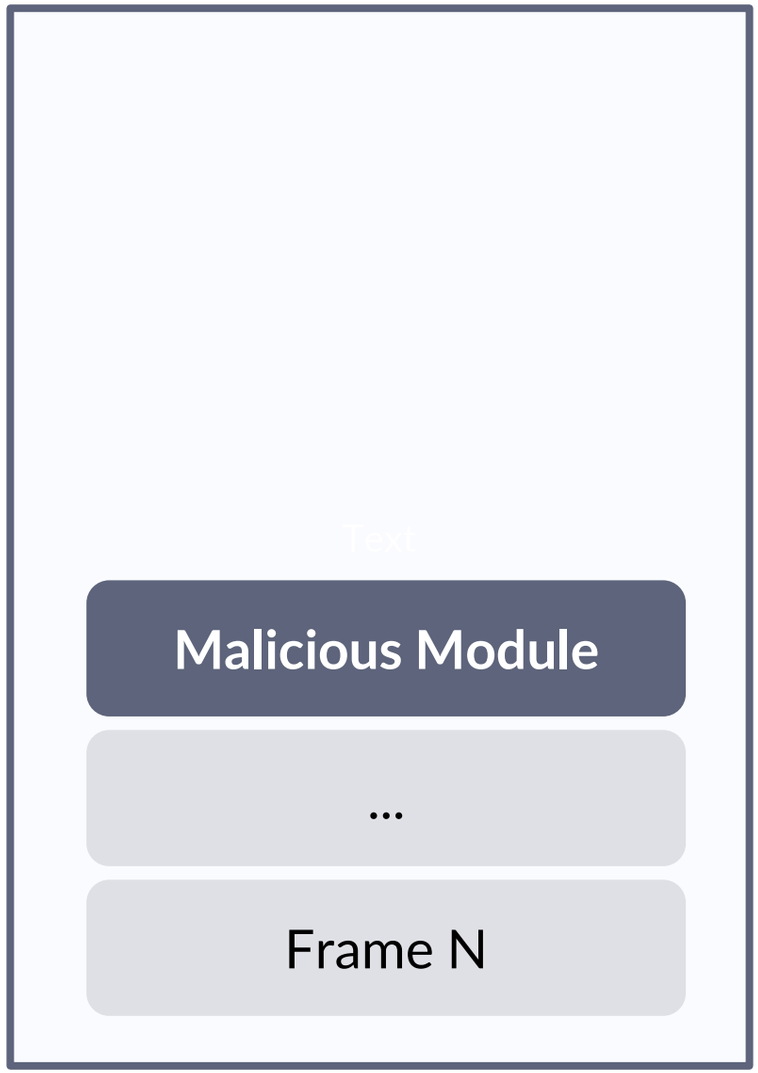
Stack Cloning

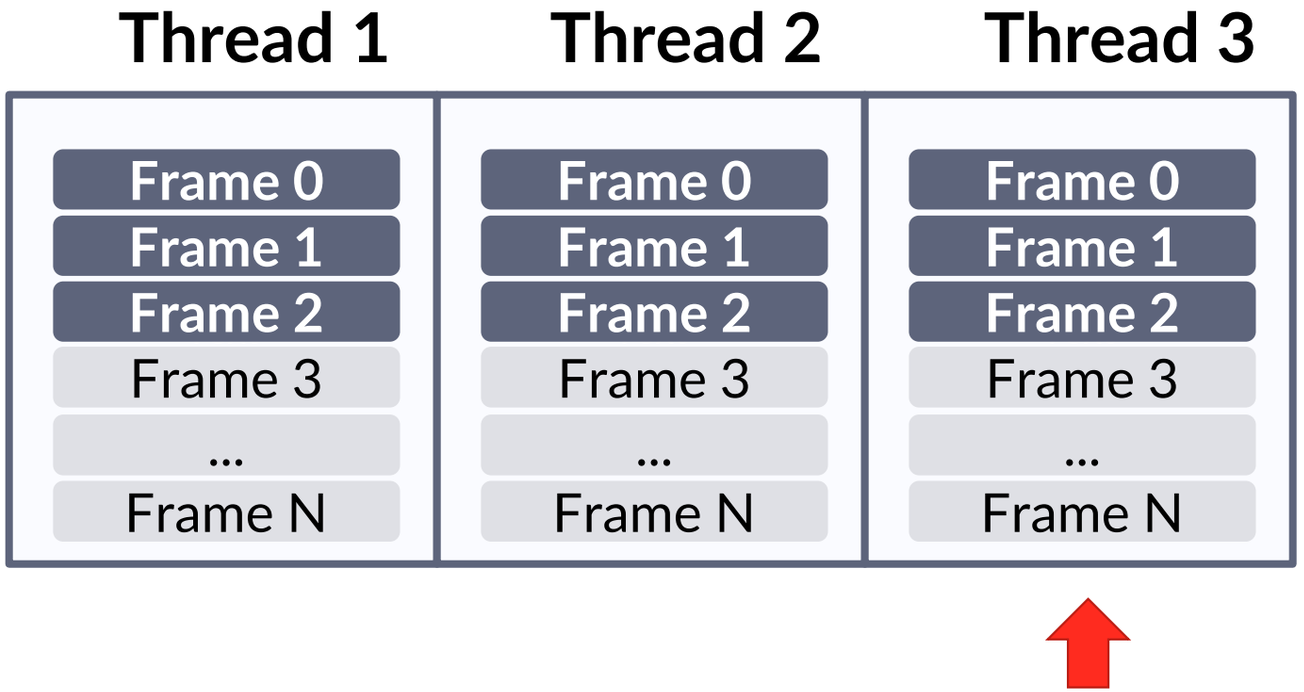




bsi.

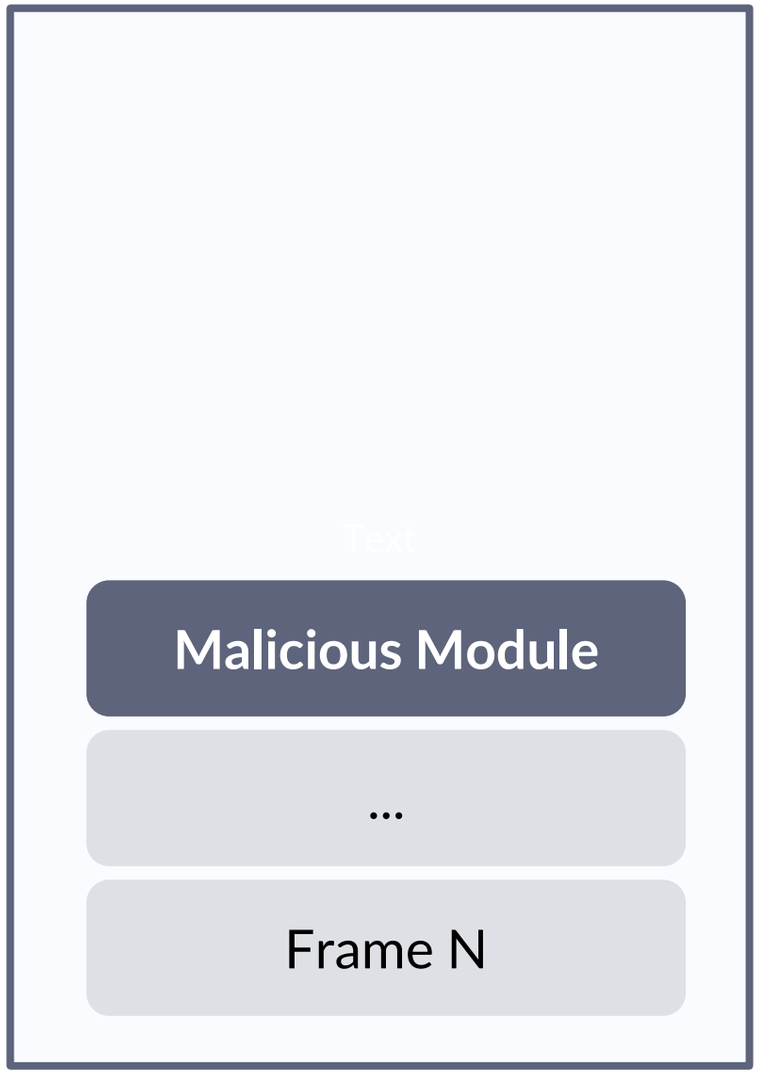
Stack Cloning

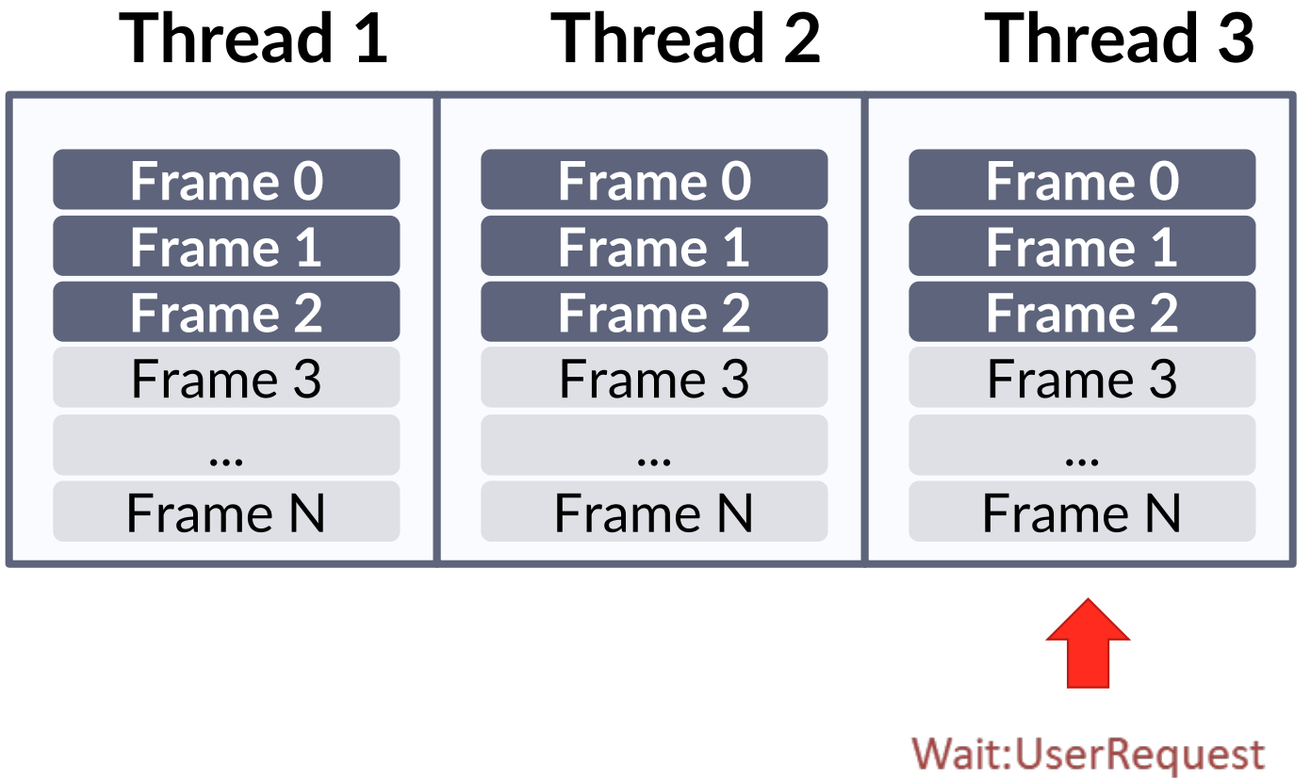




bsi.

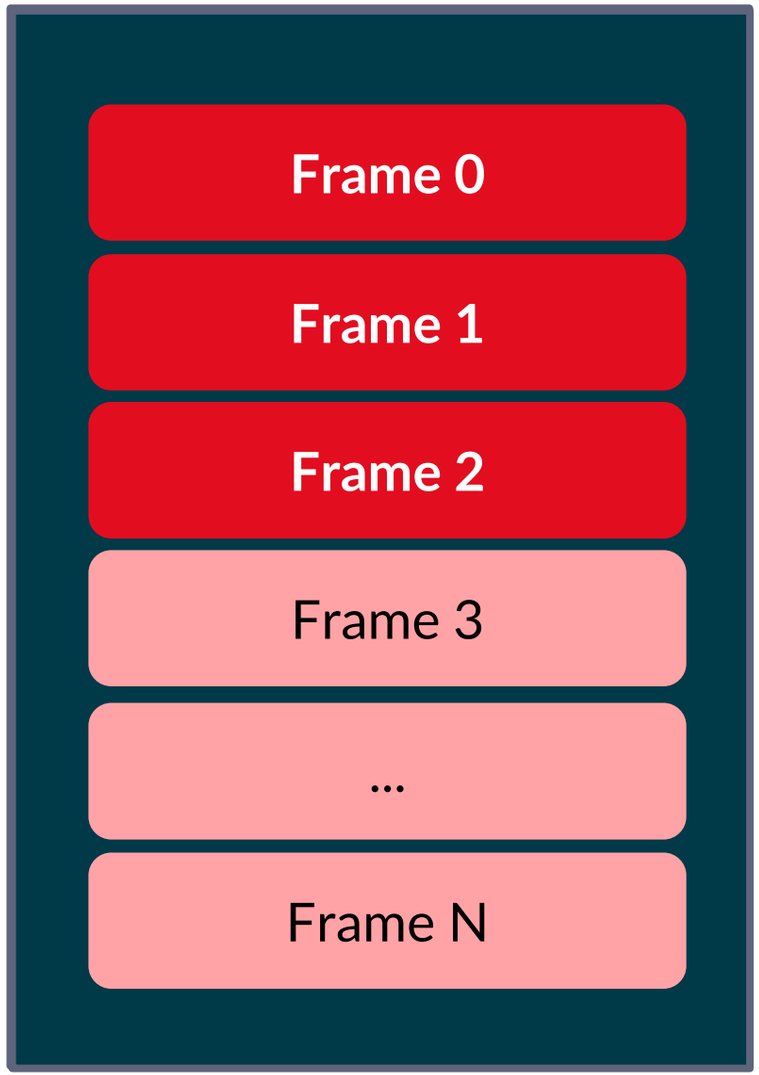
Stack Cloning

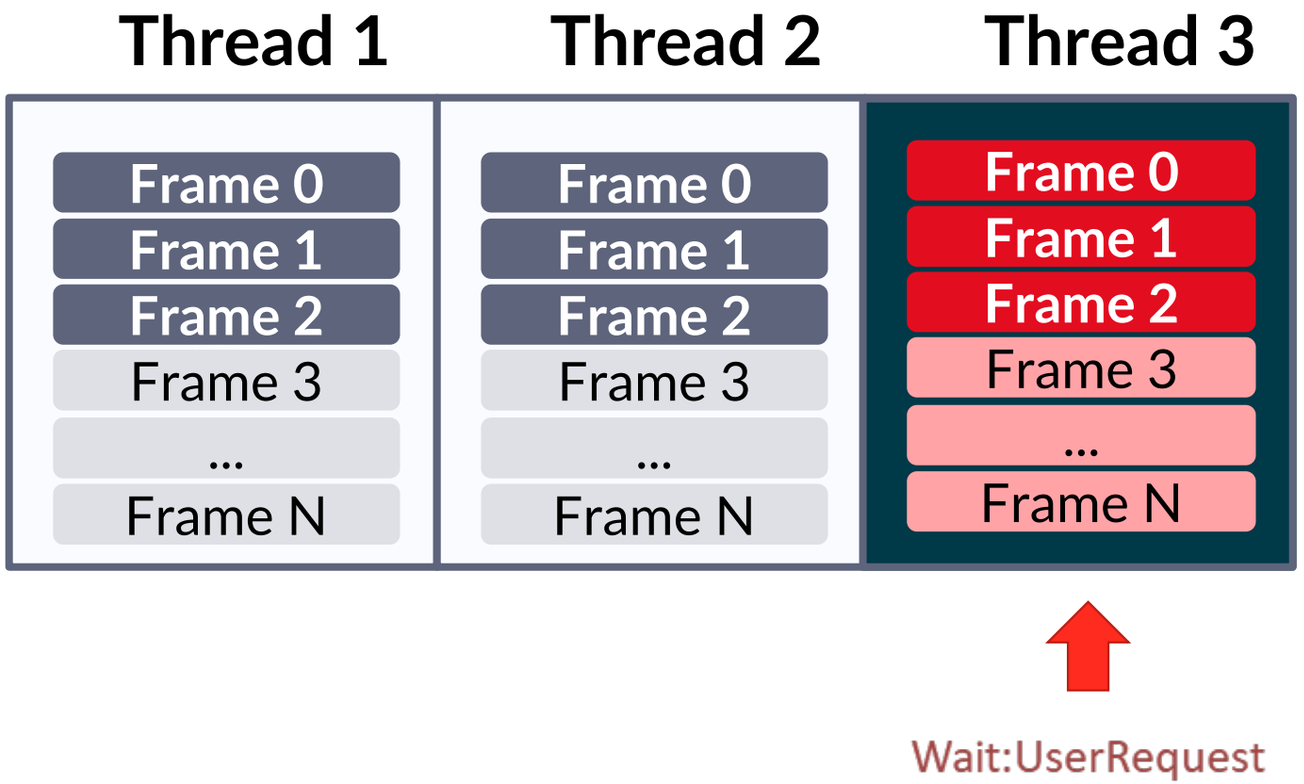




bsi.

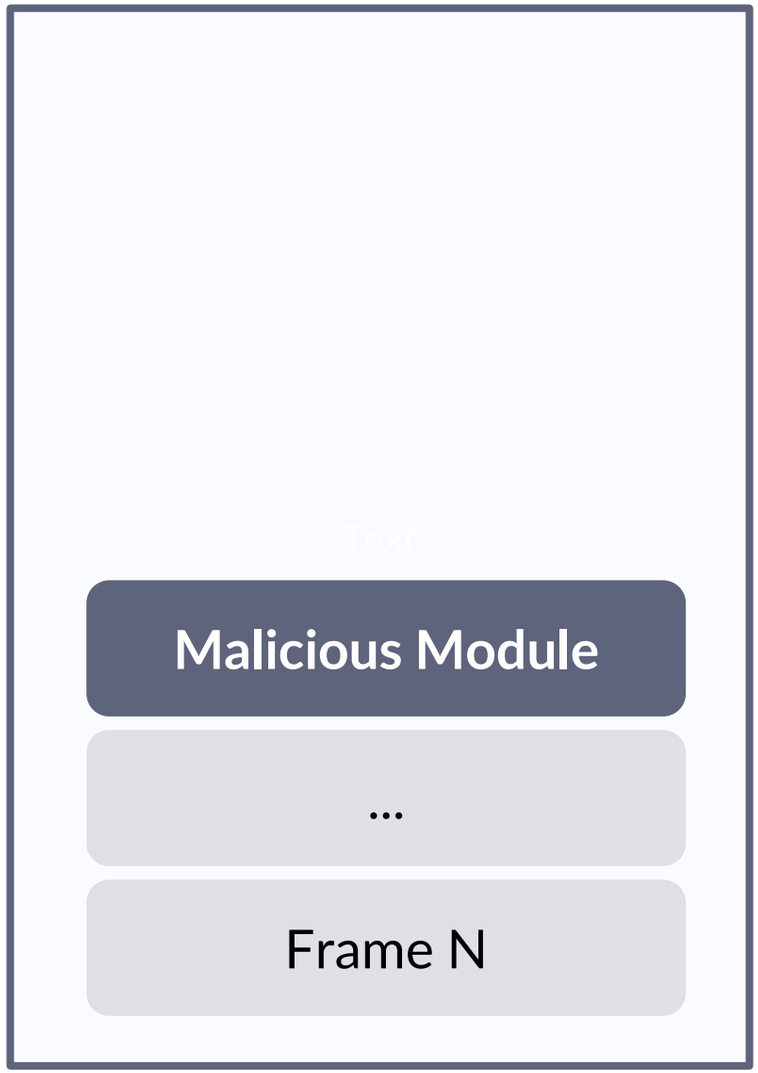
Stack Cloning





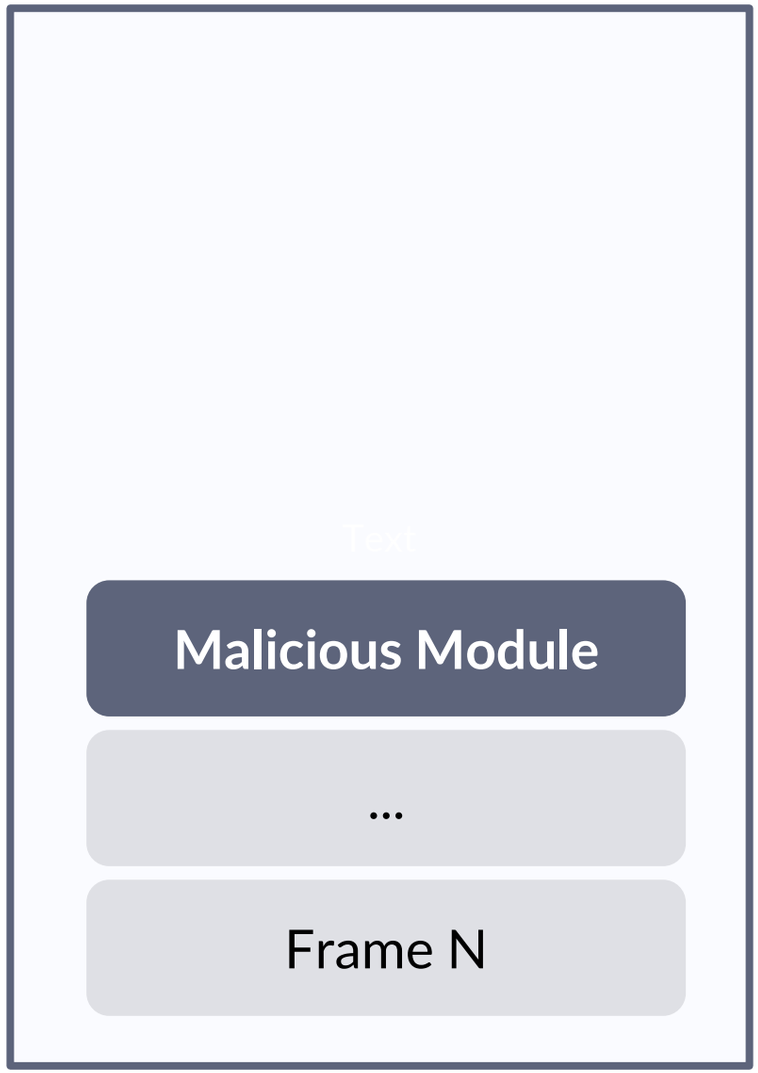
bsi.

Stack Hiding



bsi.

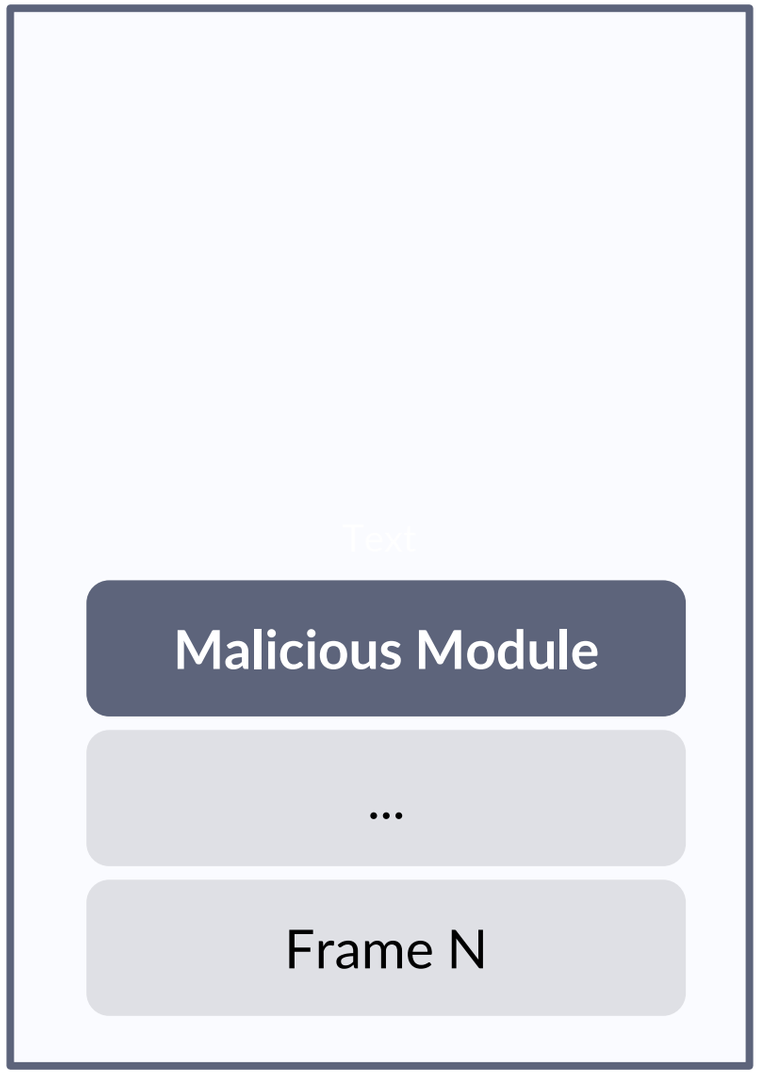
Stack Hiding

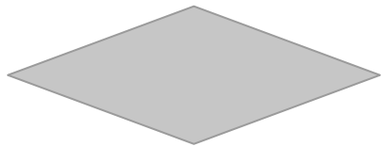


ConvertThreadToFiber(x)

bsi.

Stack Hiding

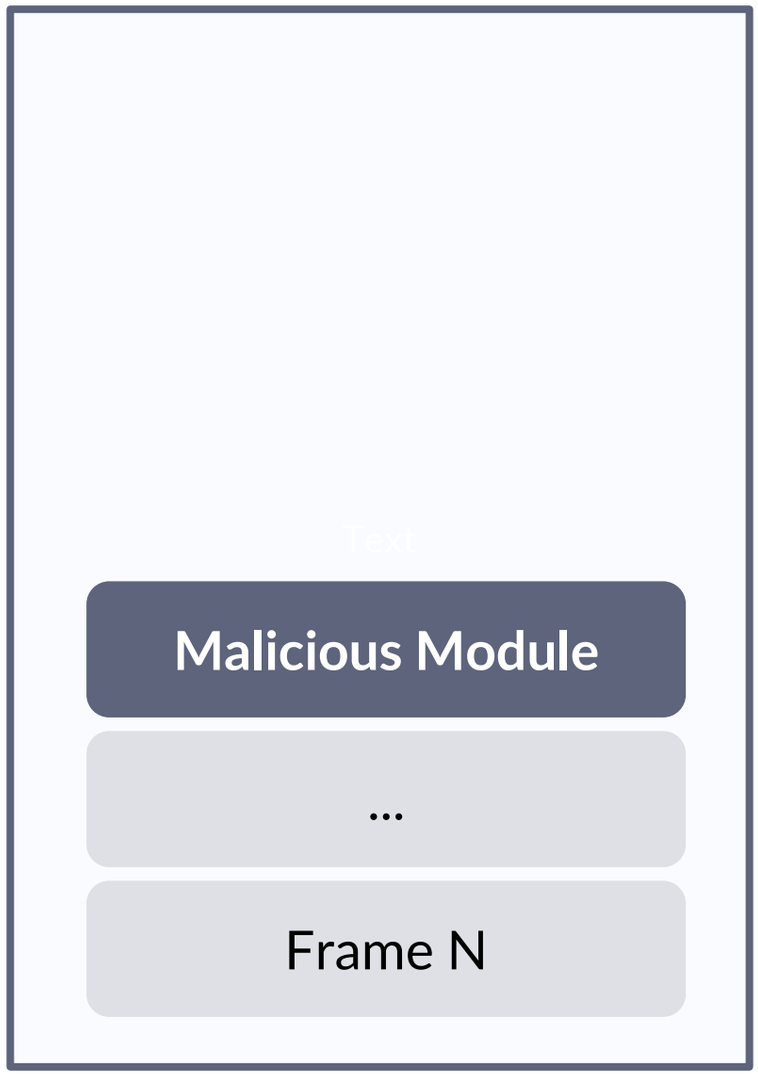


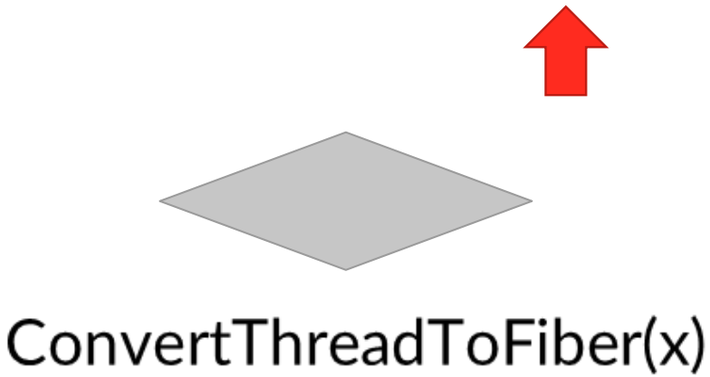


ConvertThreadToFiber(x)

bsi.

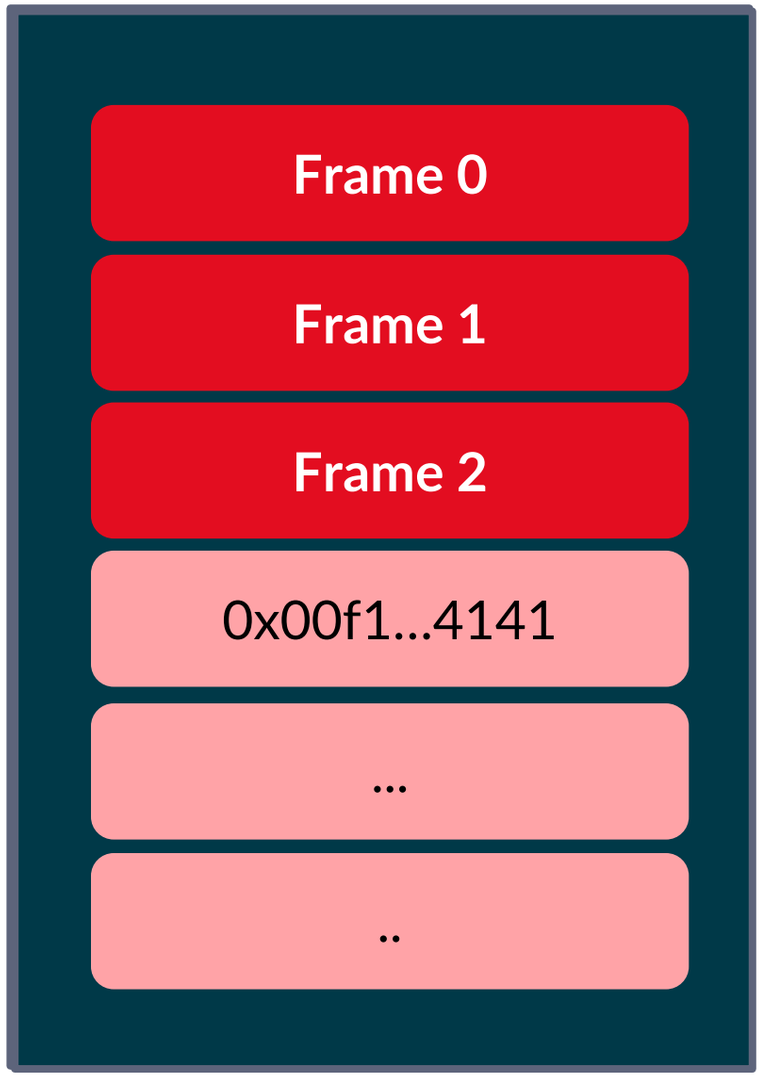
Stack Hiding



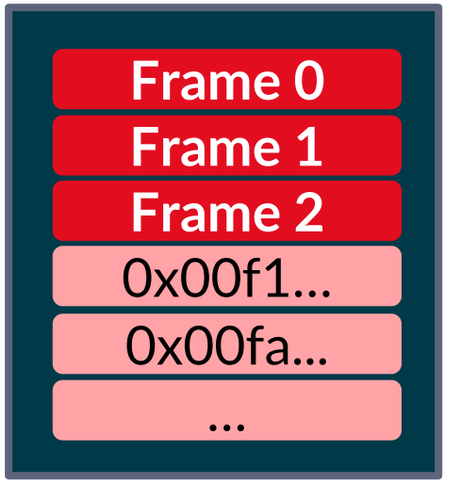


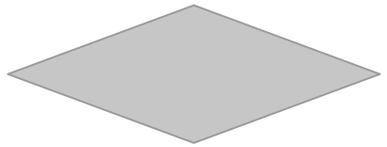
bsi.

Stack Hiding



Evil Fiber

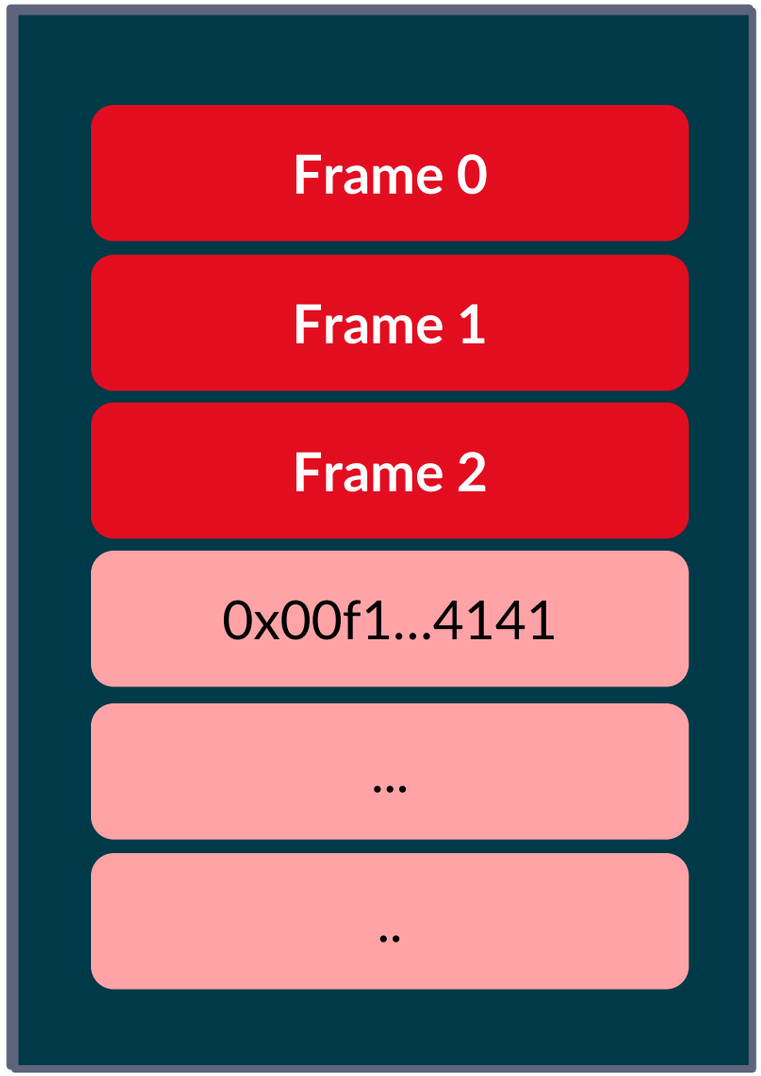




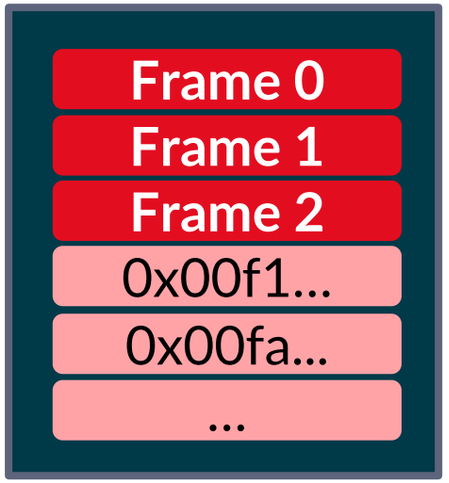
ConvertThreadToFiber(x)

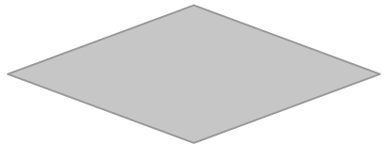
bsi.

Stack Hiding



Evil Fiber

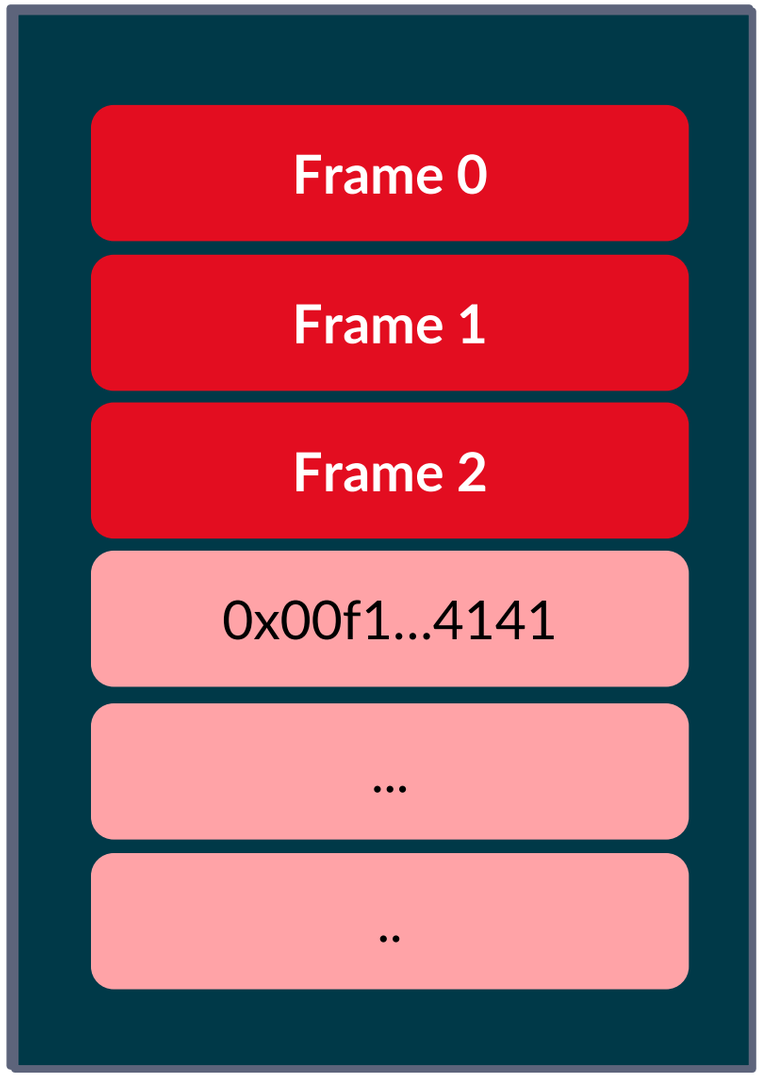




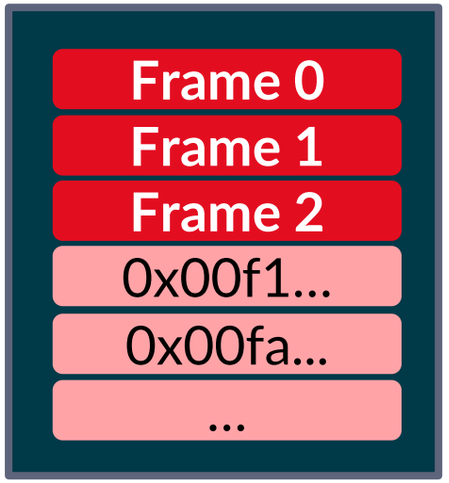
CreateFiber(x, y, z)

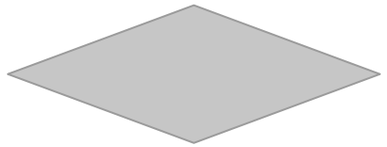
bsi.

Stack Hiding



Evil Fiber

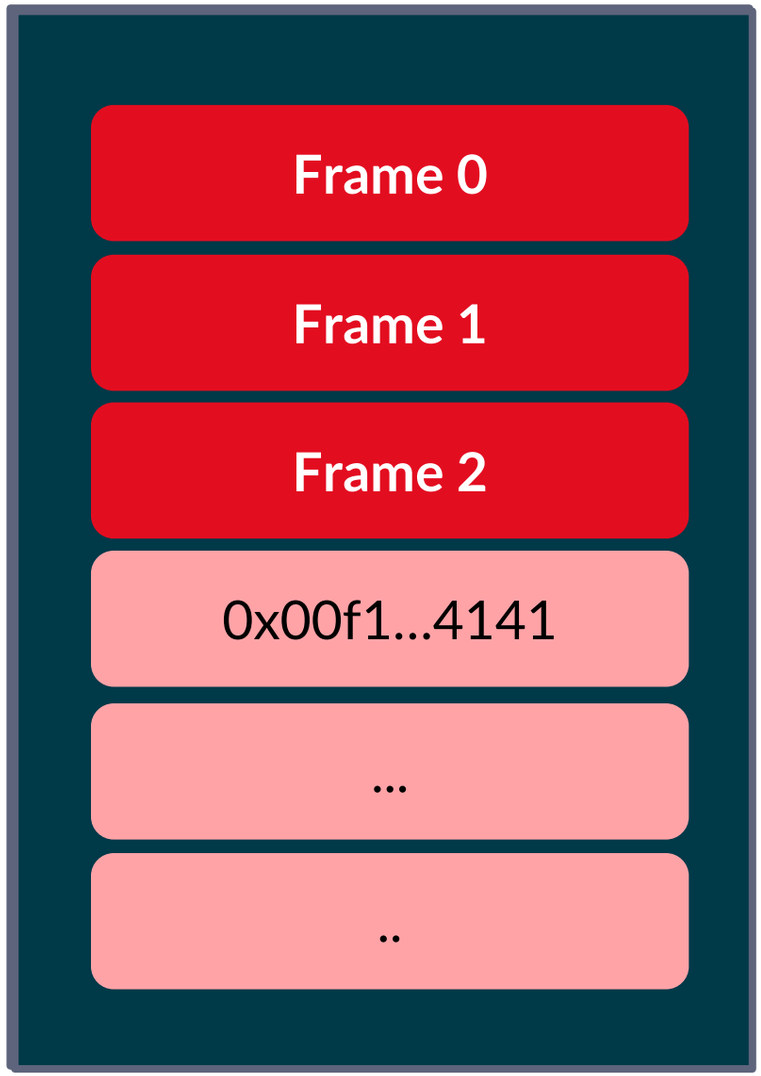


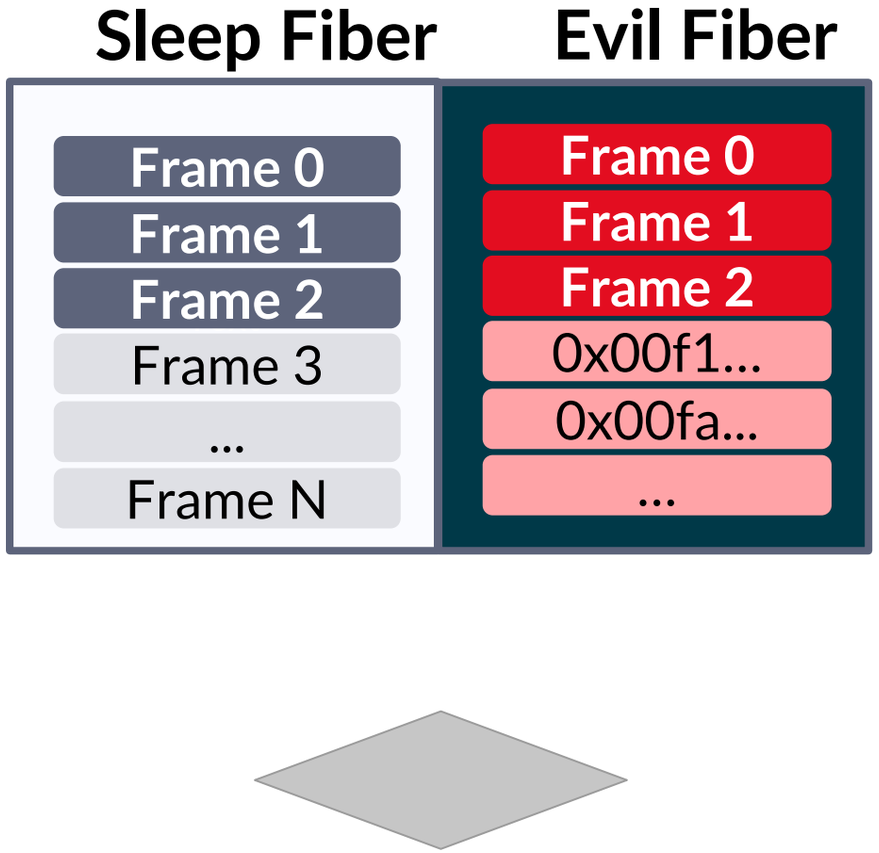


CreateFiber(x, y, z)

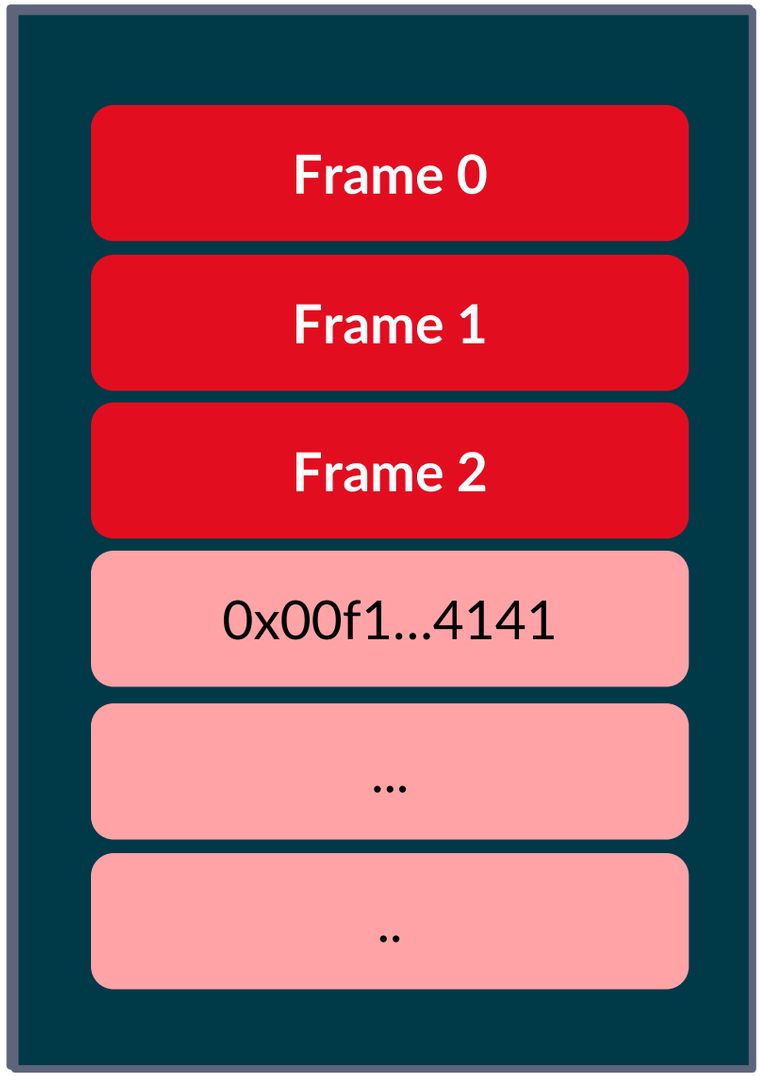
bsi.

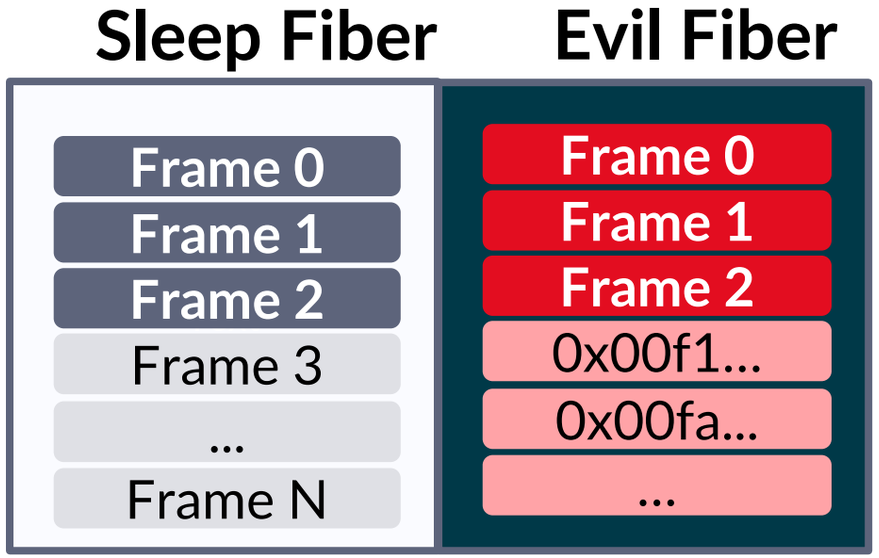
Stack Hiding

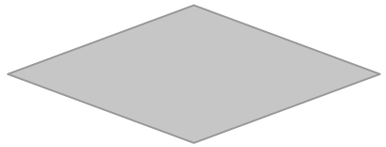




Stack Hiding



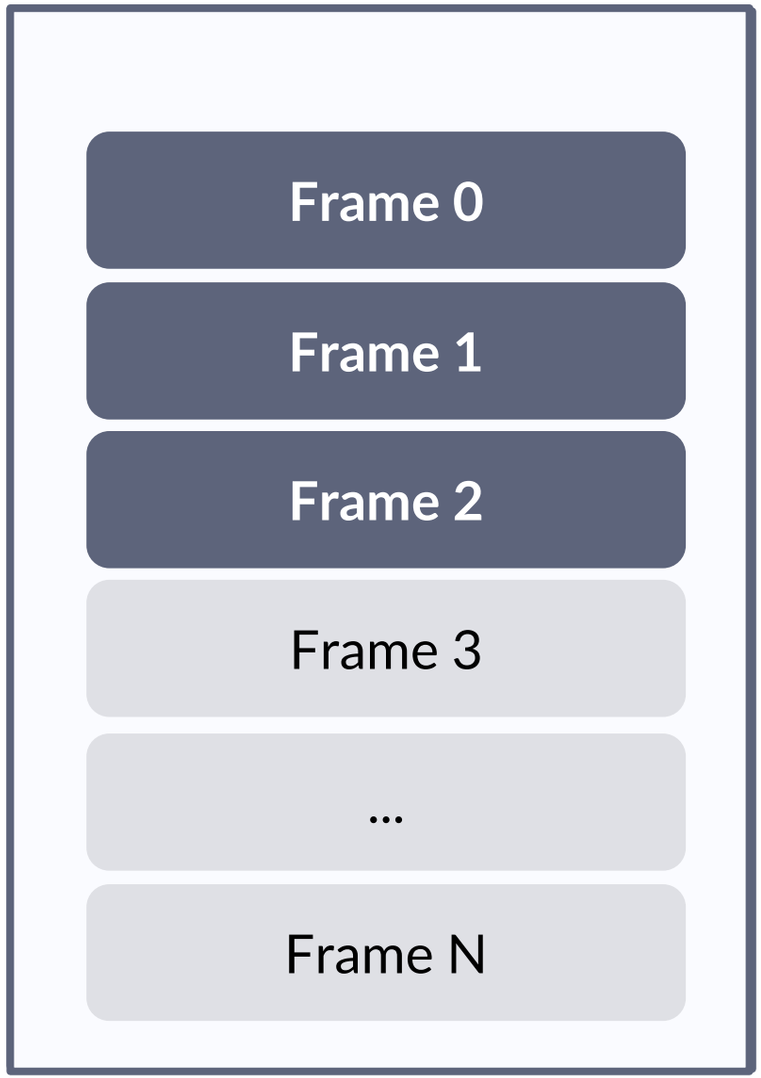


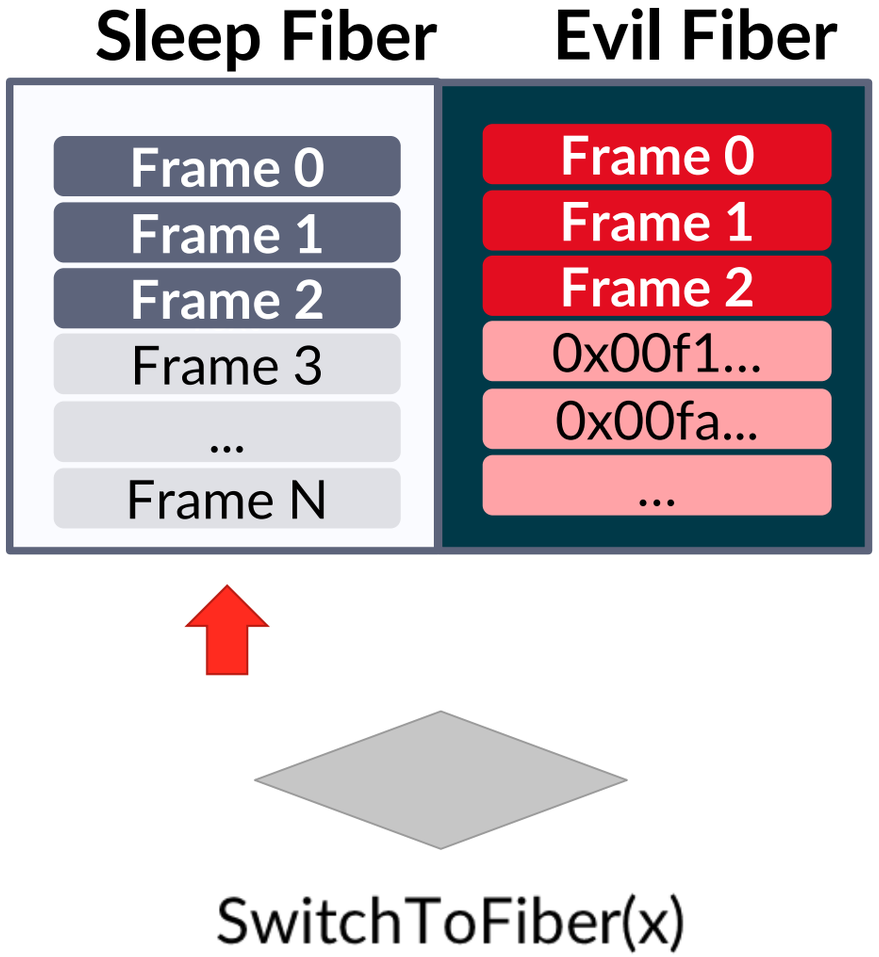


SwitchToFiber(x)

bsi.

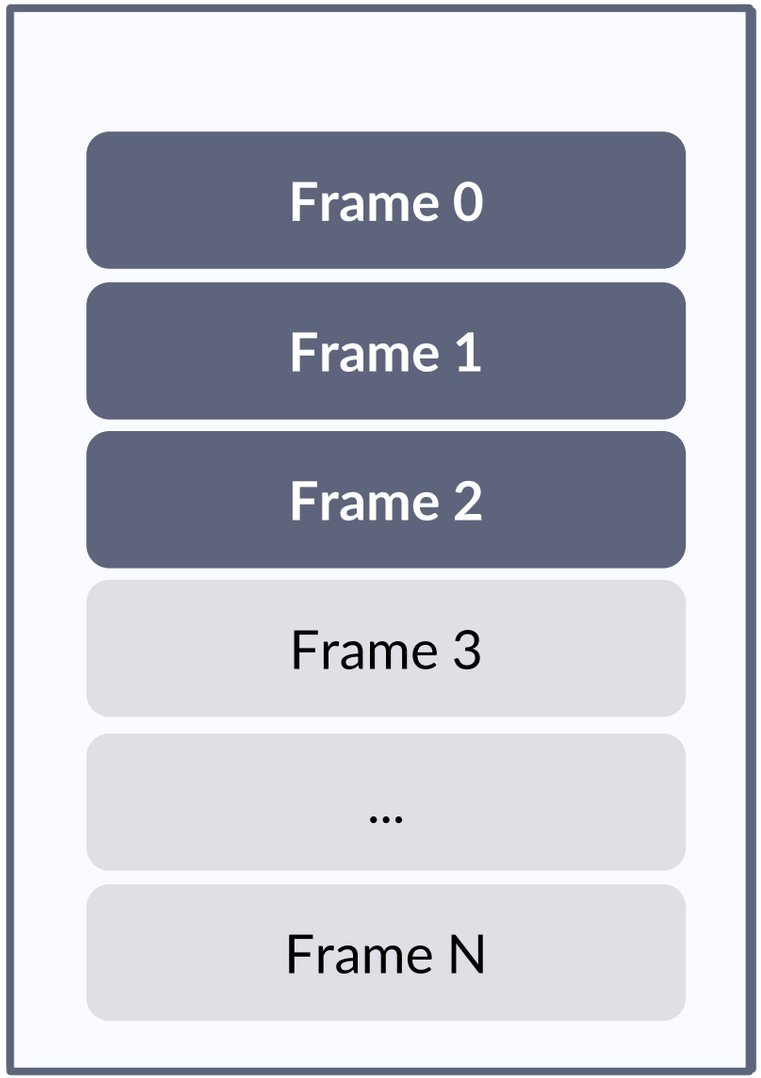
Stack Hiding

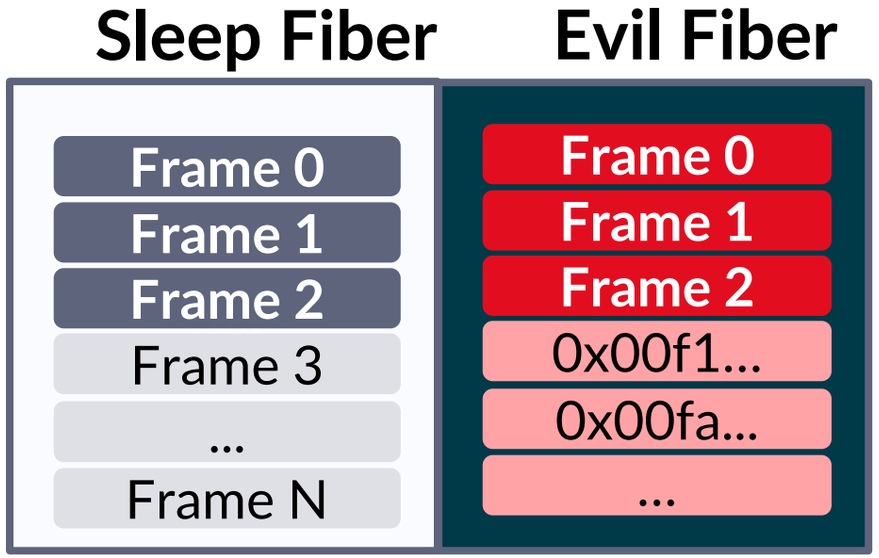


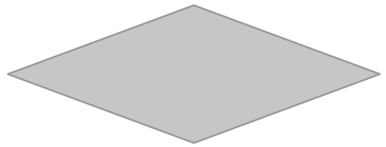


bsi.

Stack Hiding



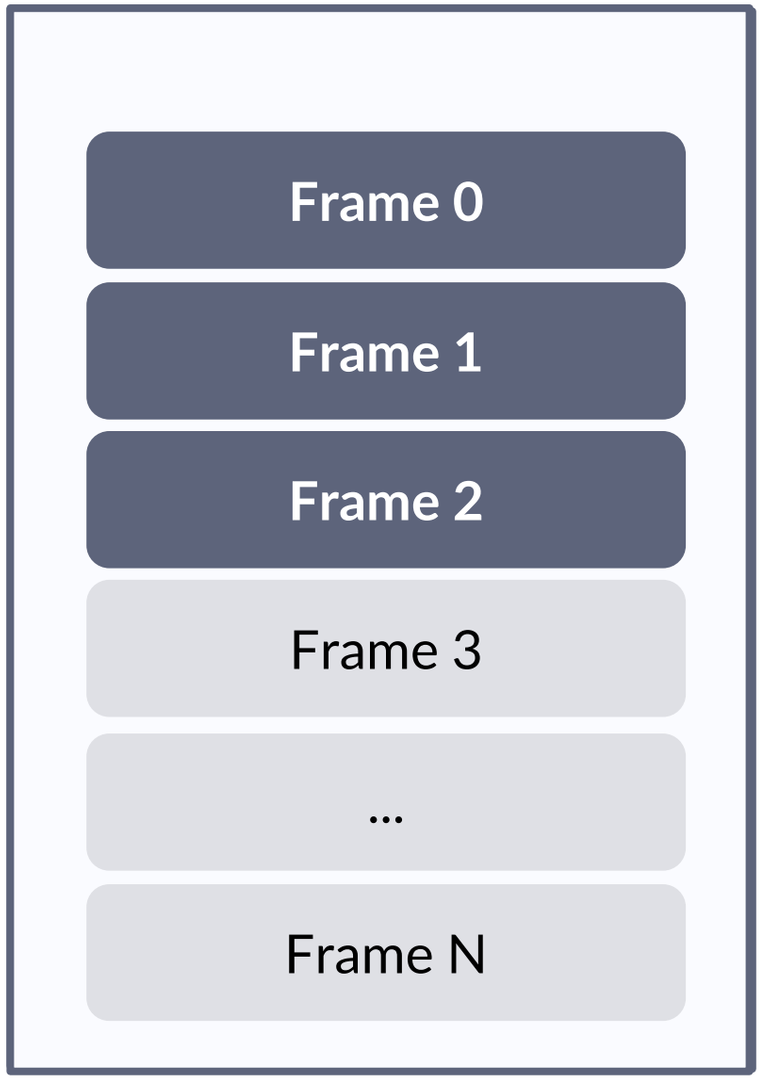


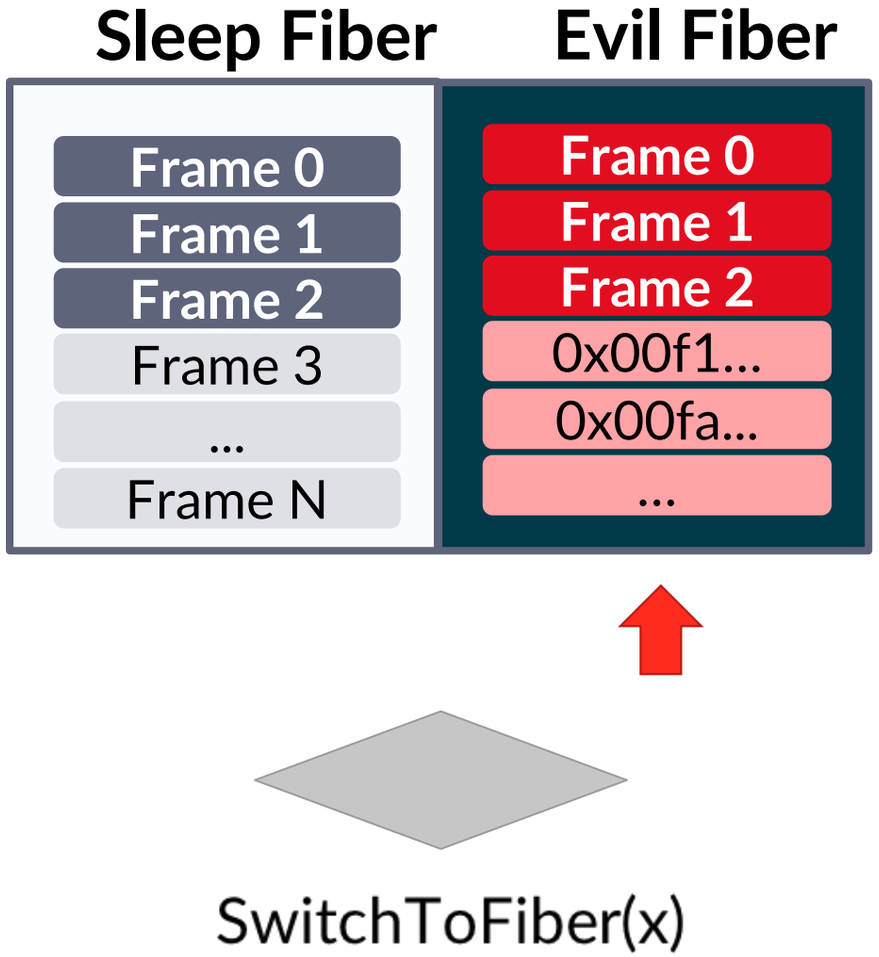


SwitchToFiber(x)

bsi.

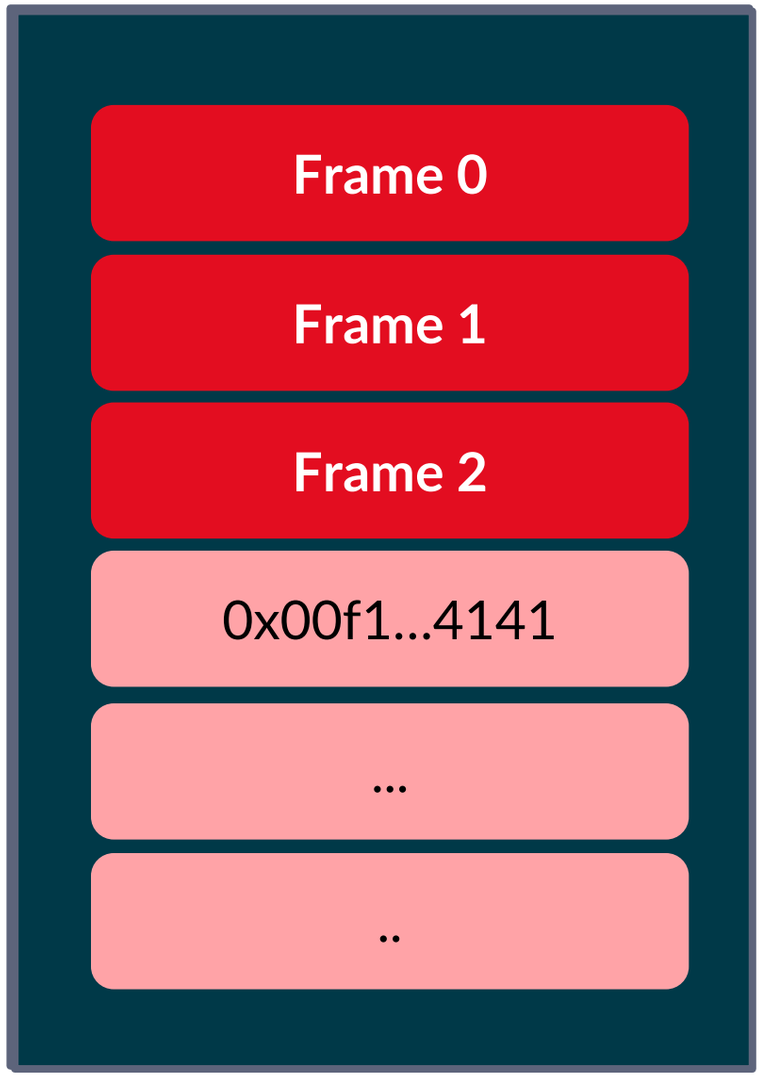
Stack Hiding

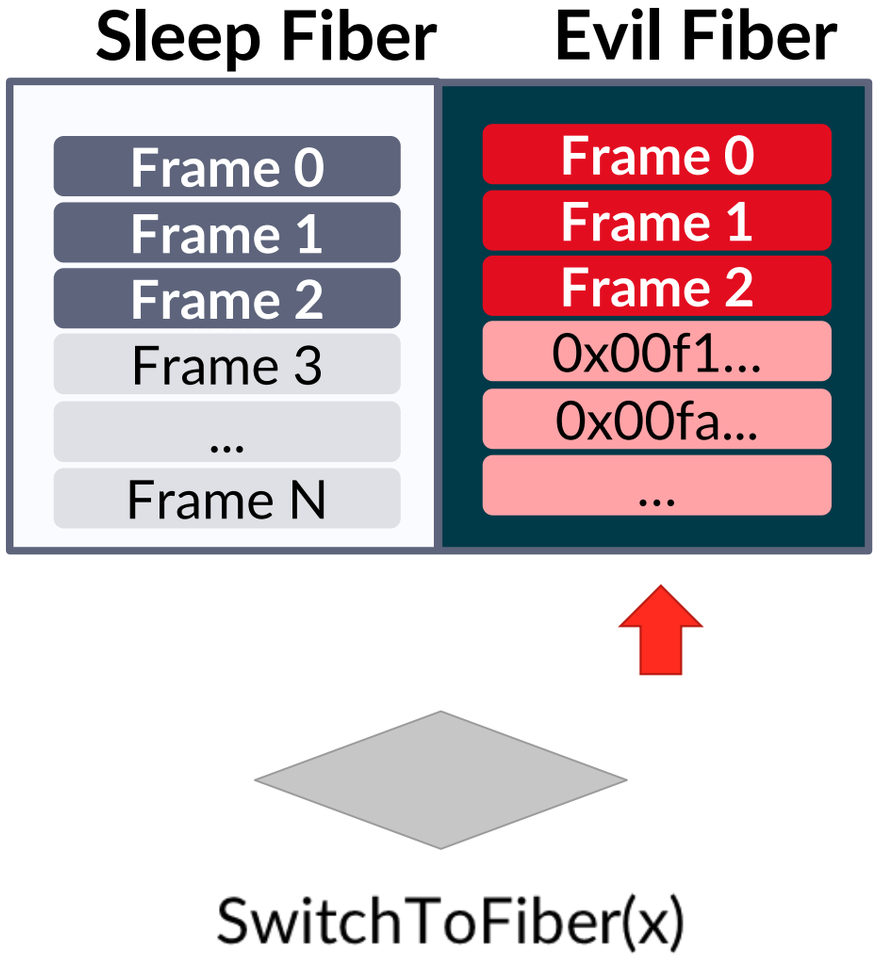




bsi.

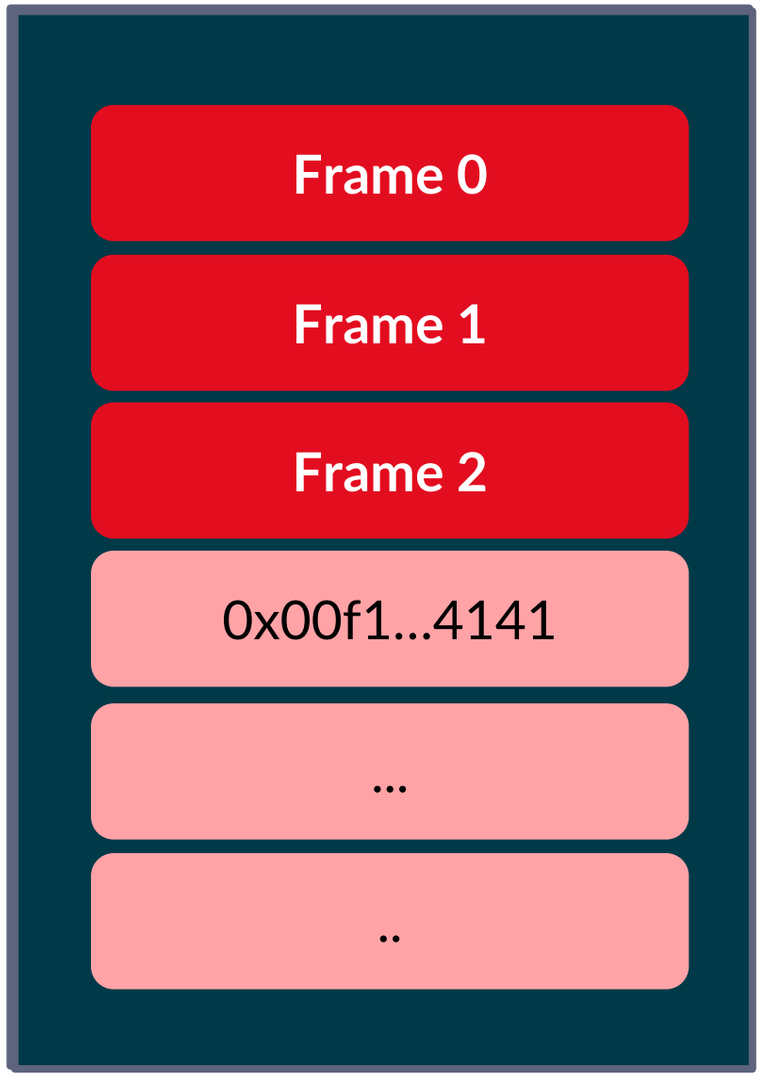
Stack Hiding

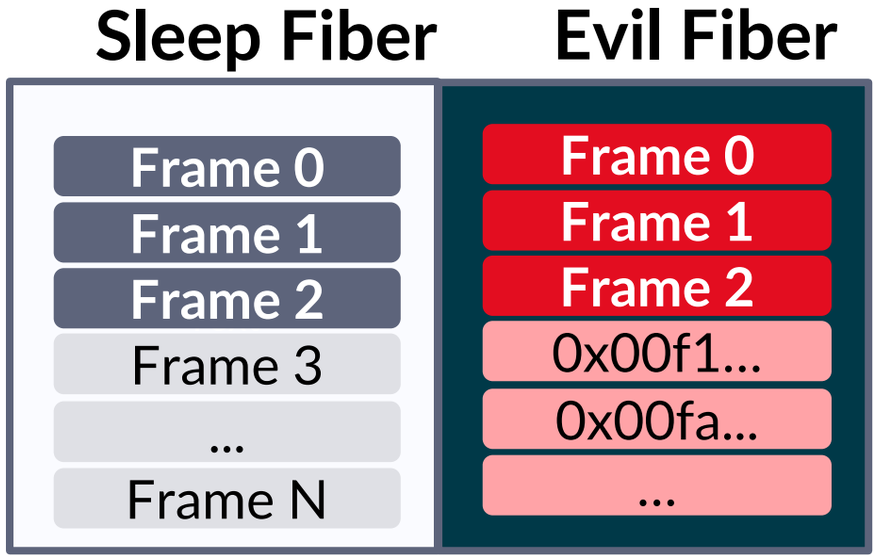


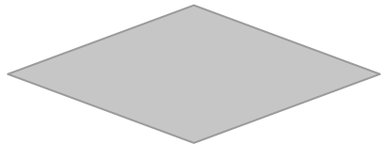


bsi.

Stack Hiding



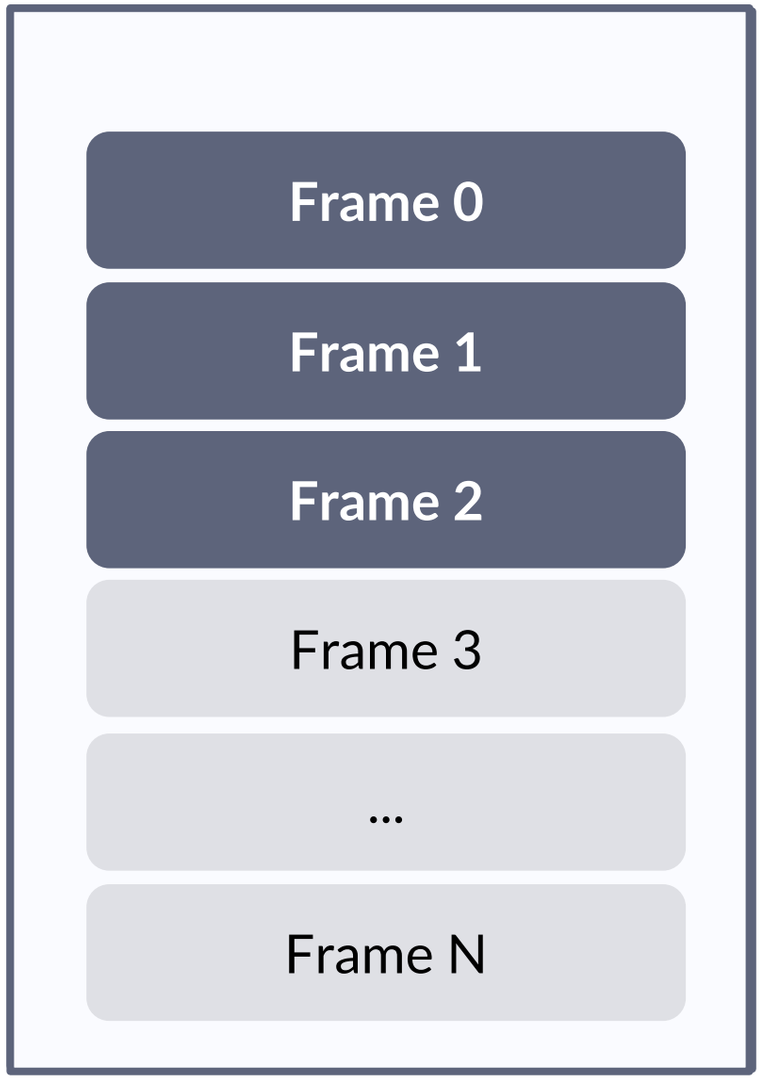


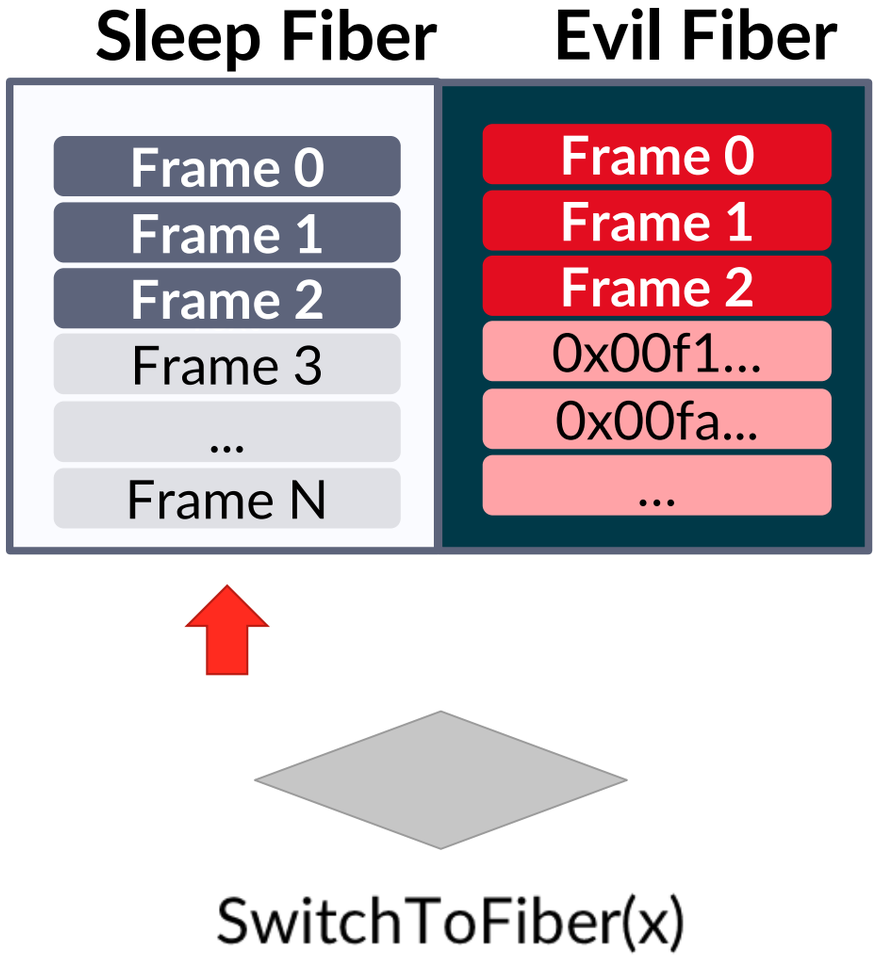


SwitchToFiber(x)

bsi.

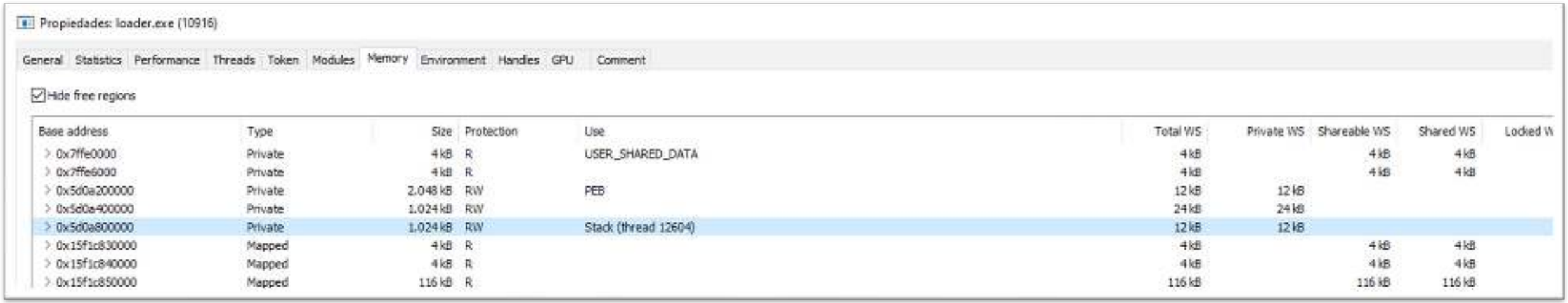
Stack Hiding





bsi.

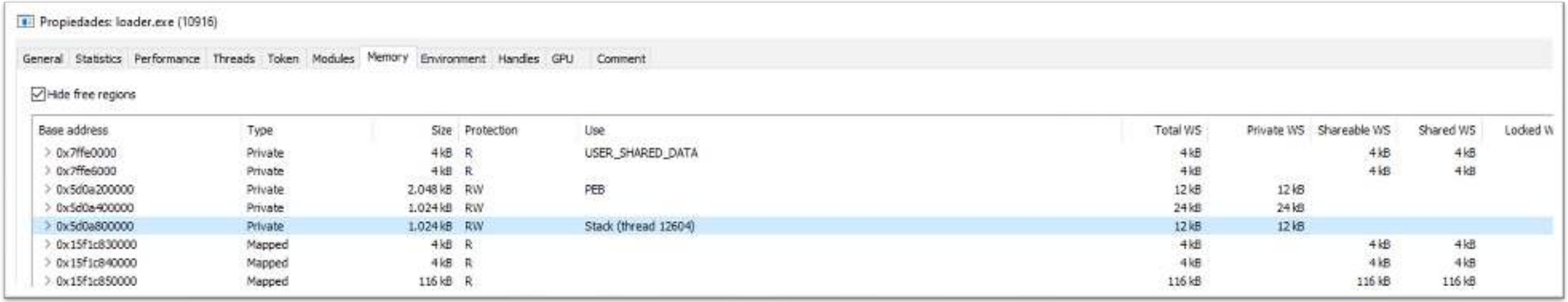
Stack Hiding

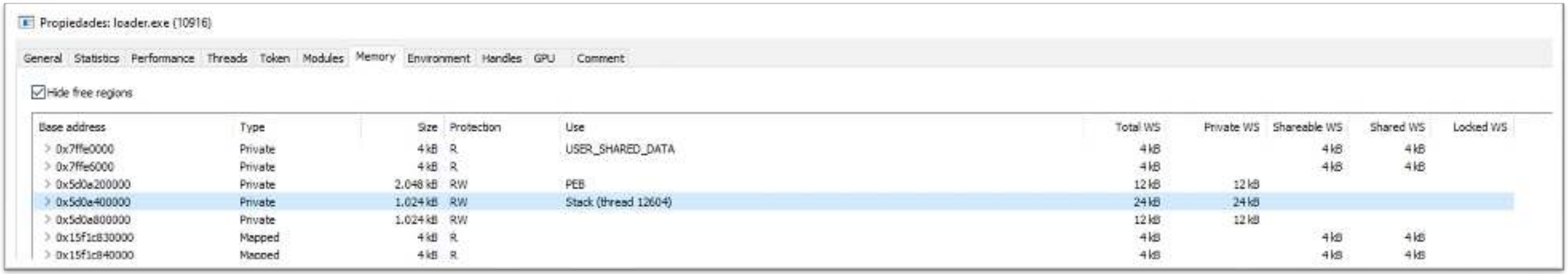


bsi.

https://github.com/Kudaes/Fiber

Stack Hiding

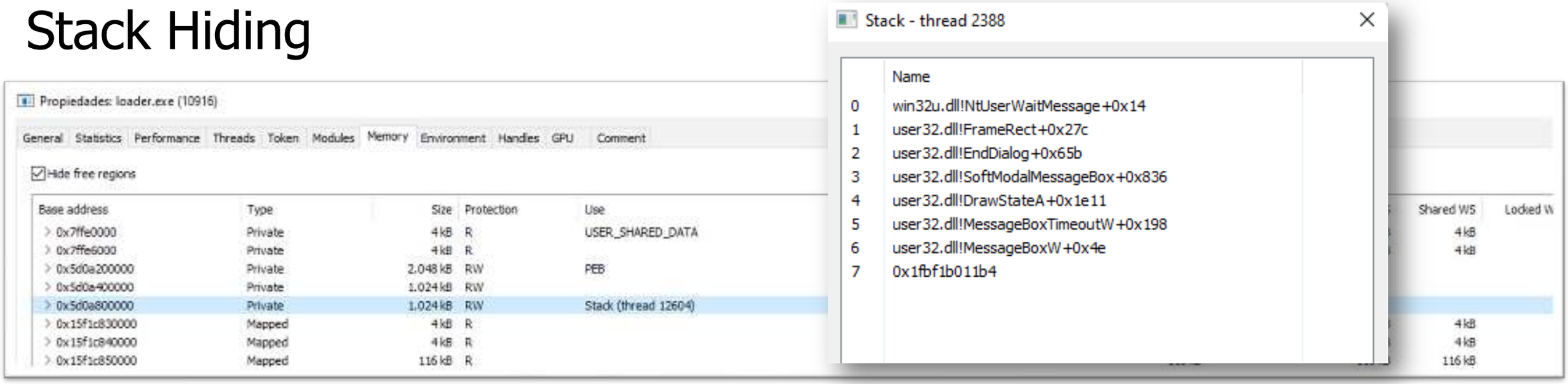


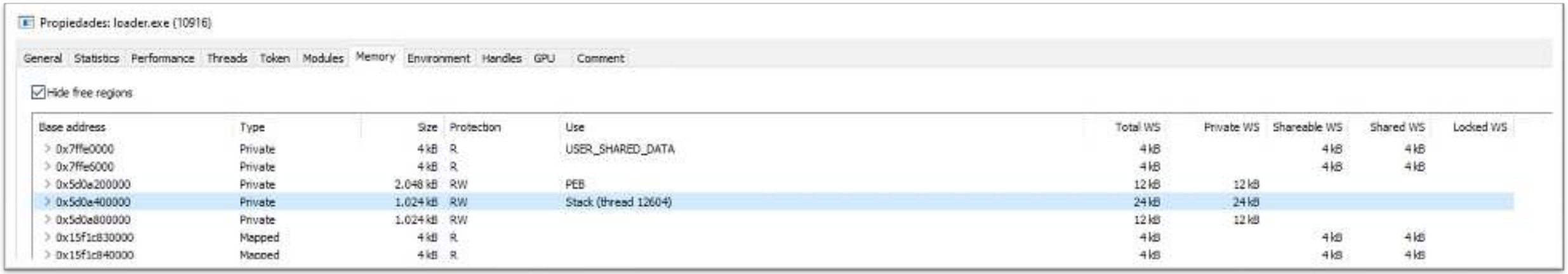


bsi.

https://github.com/Kudaes/Fiber

Stack Hiding

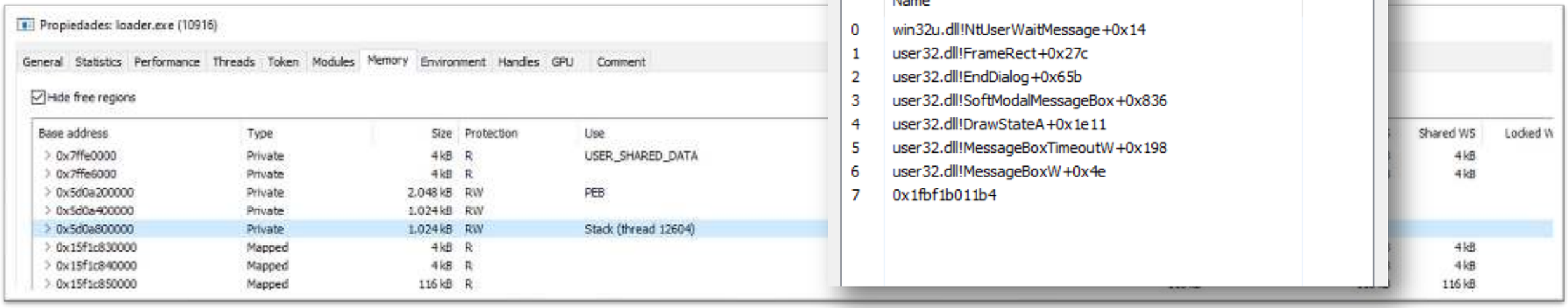


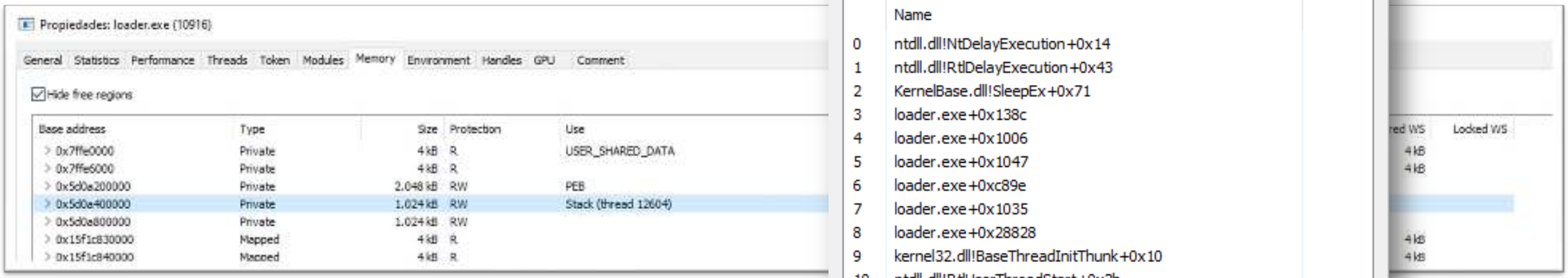


bsi.

https://github.com/Kudaes/Fiber

Stack Hiding





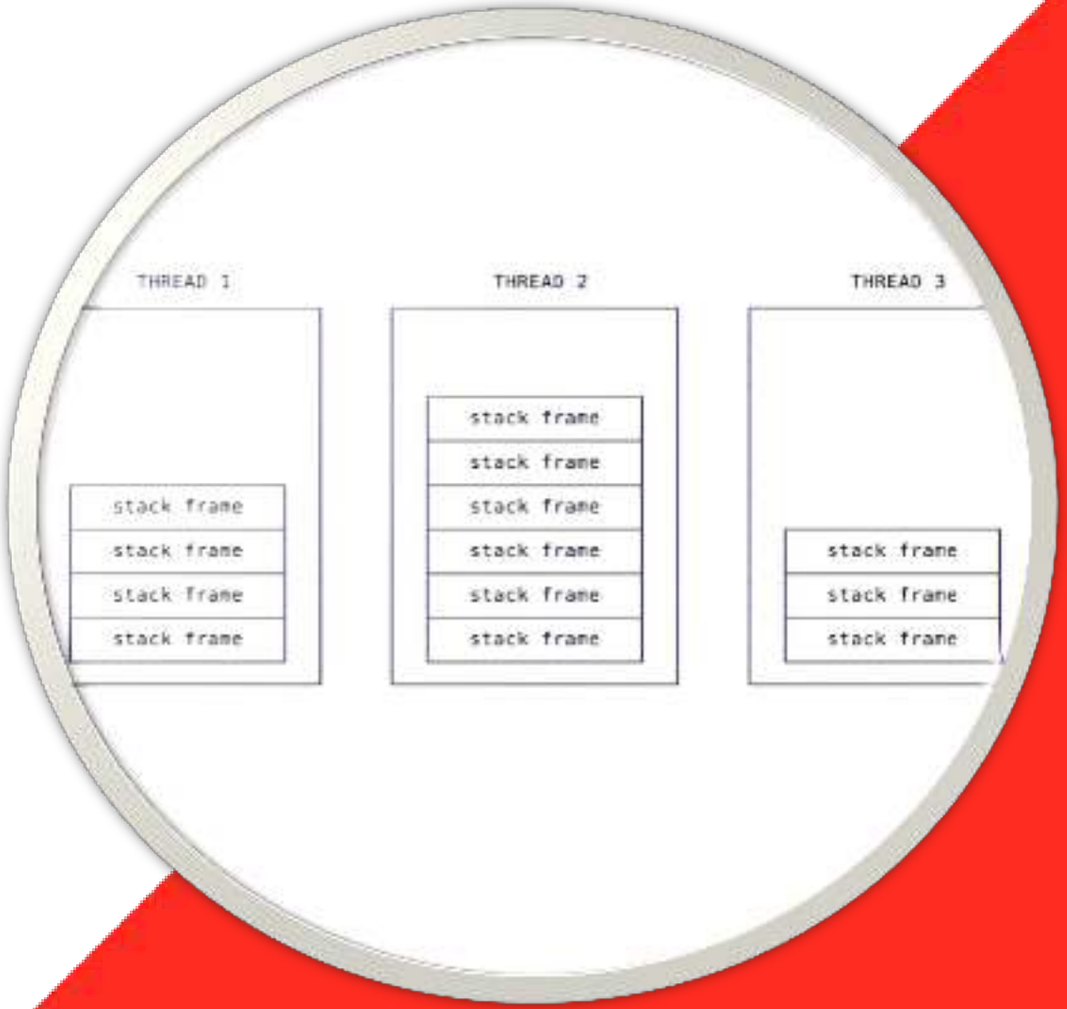
bsi.

https://github.com/Kudaes/Fiber

bsi.

Moonwalking the Windows Stack

Developing a dynamic call stack spoofer



bsi.

Stack Moonwalk

1

Hide Caller

This technique was initially developed to conceal the main caller module from the call stack, preseriving its unwindability when executing in-memory code

2

Obfuscate Stack

Stack Moonwalk obfuscates the stack, making it difficult to understand how a specific API/System Call was invoked

3

Auto Restore

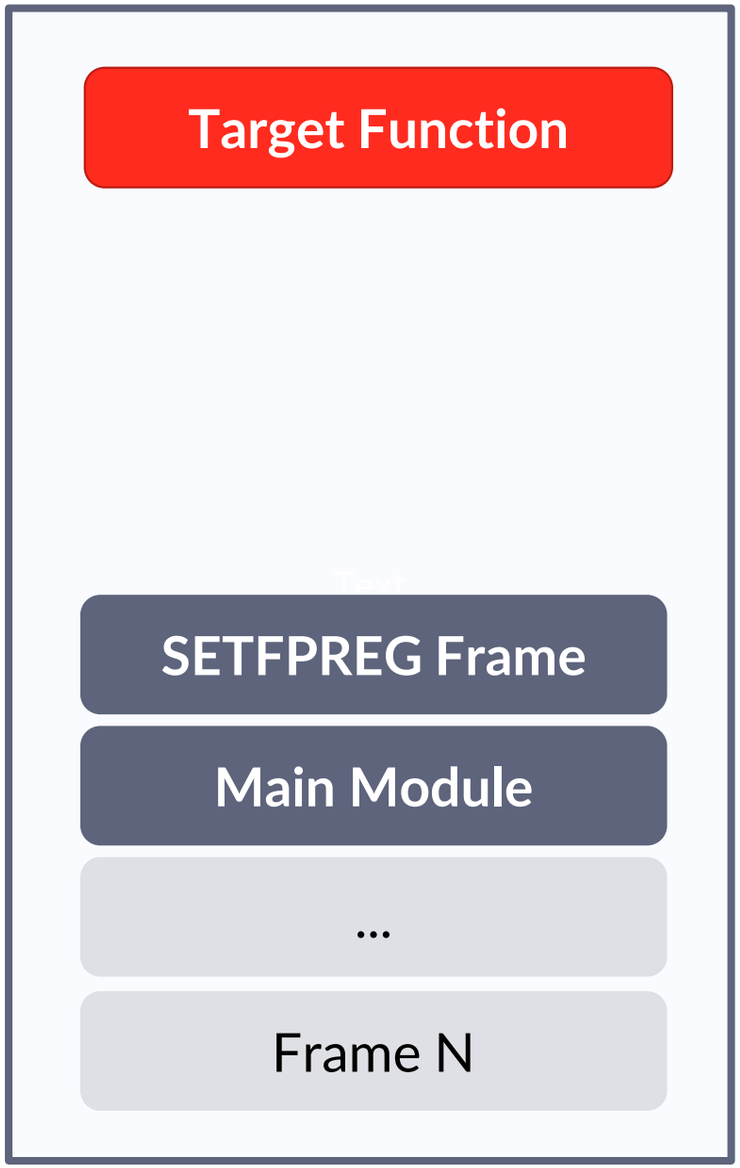
The technique does not use any external mechanism to restore the stack on return, it does use a ROP paradigm instead

bsi.

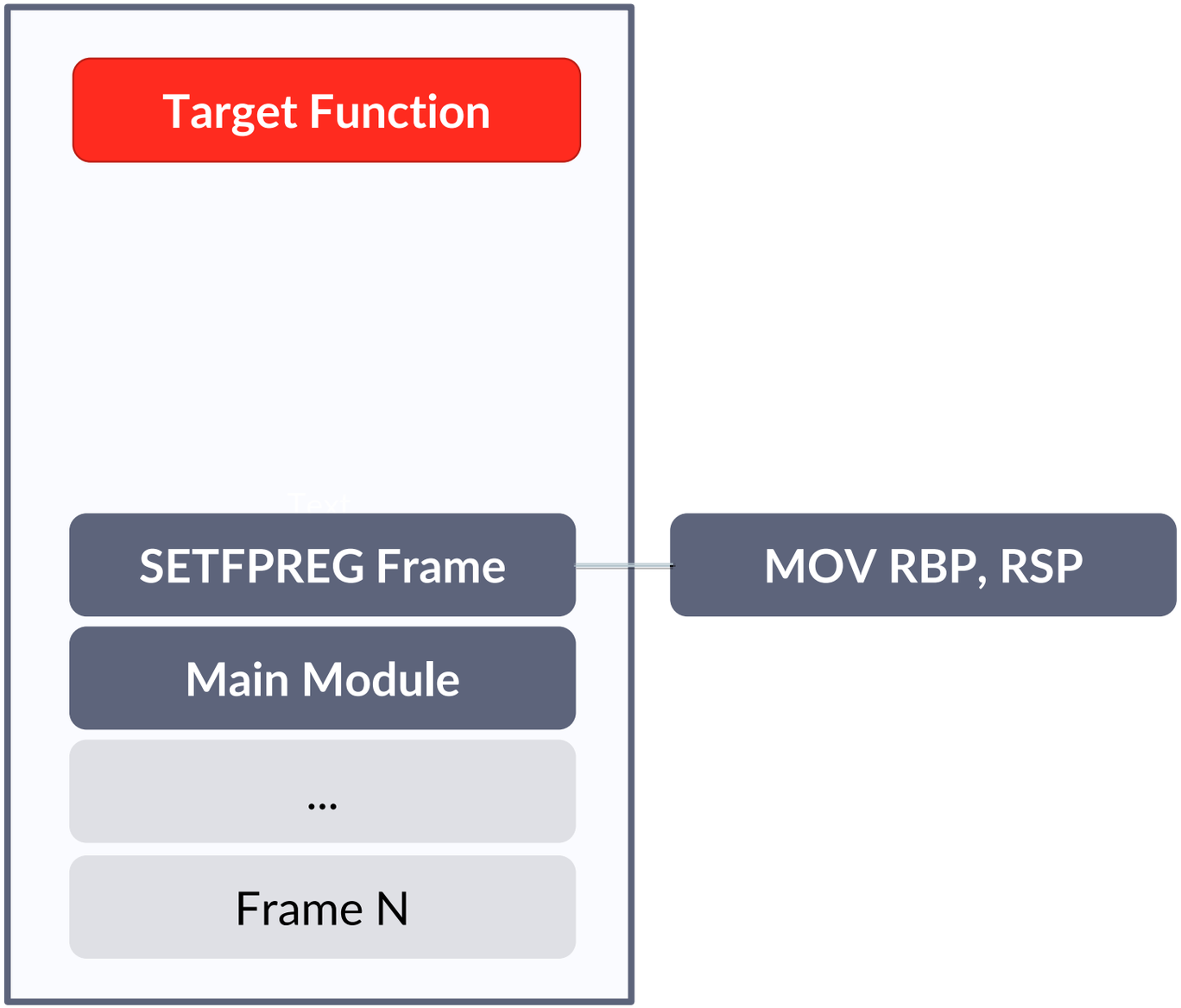
Full Moon (Walk)

Full Moon (Walk)

Full Moon (Walk)

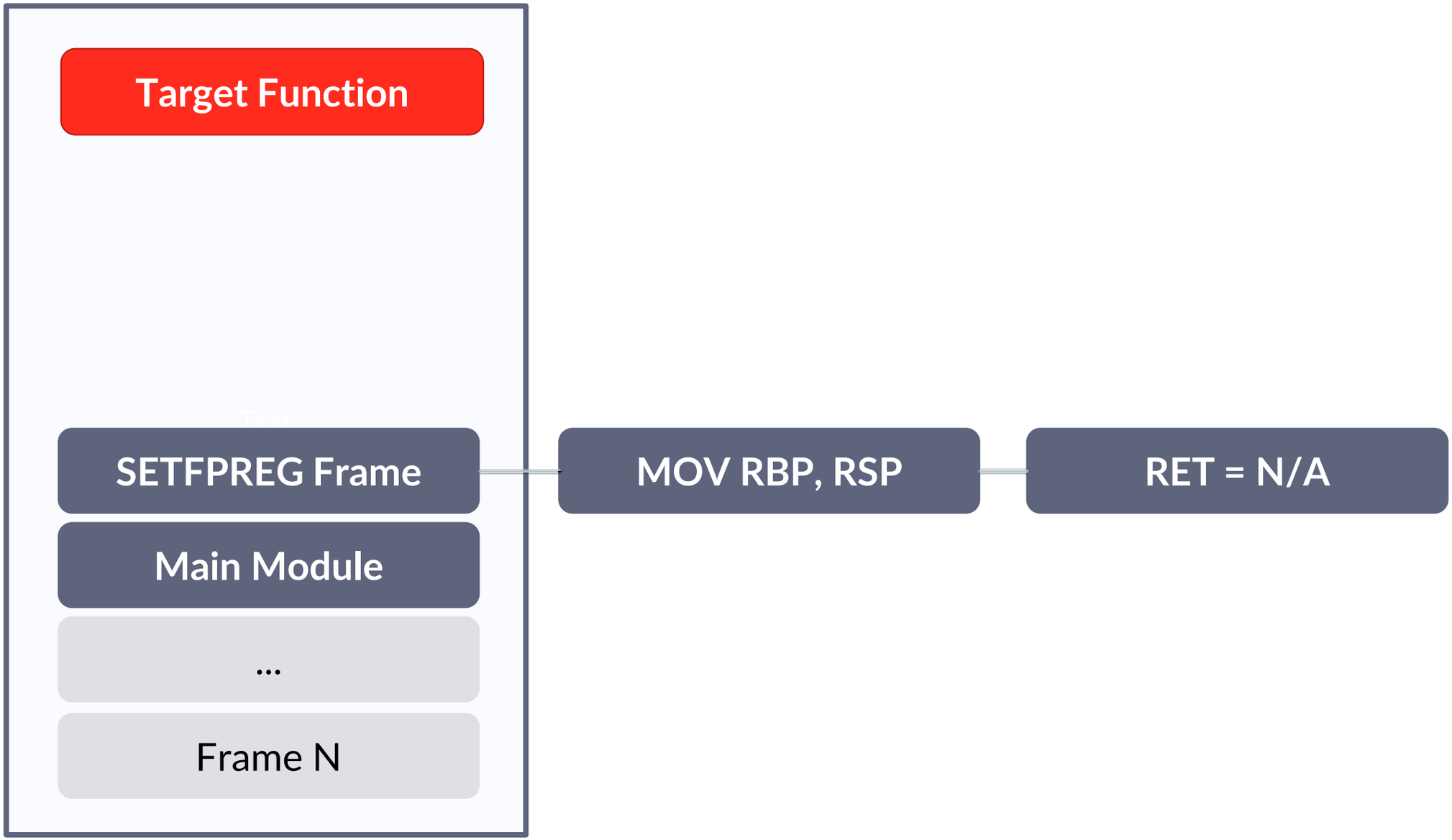


Full Moon (Walk)

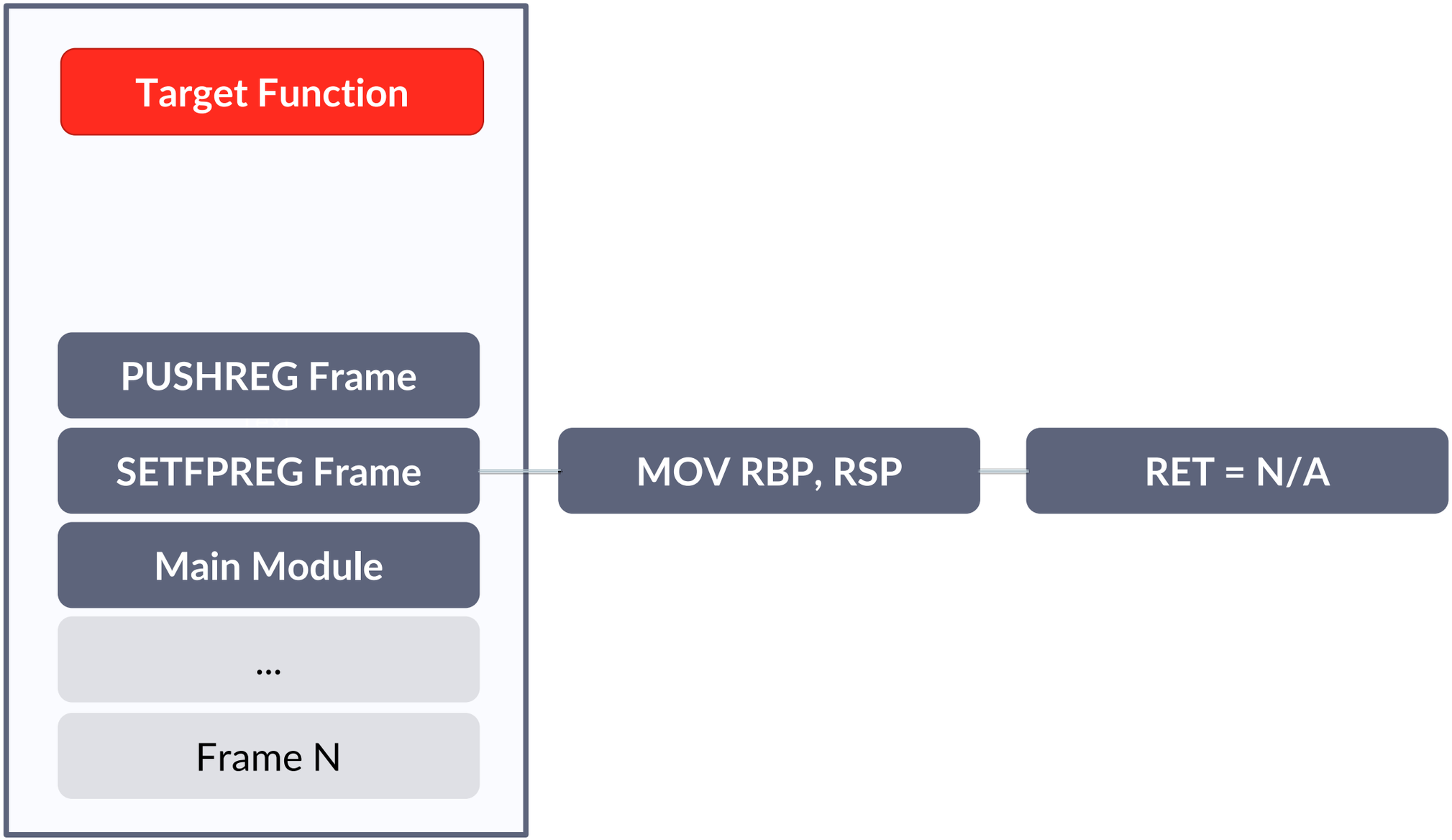


bsi.

Full Moon (Walk)

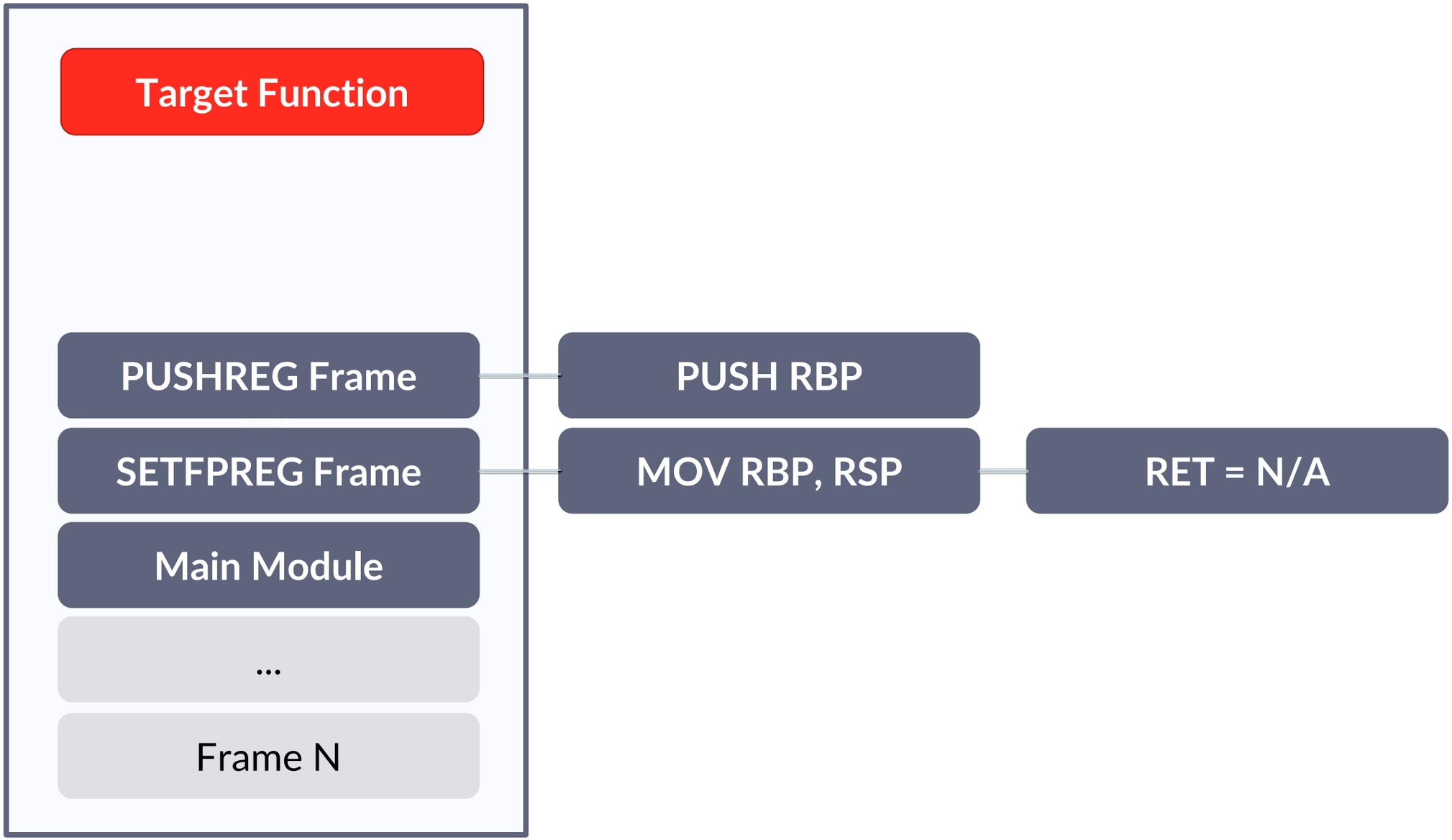


Full Moon (Walk)



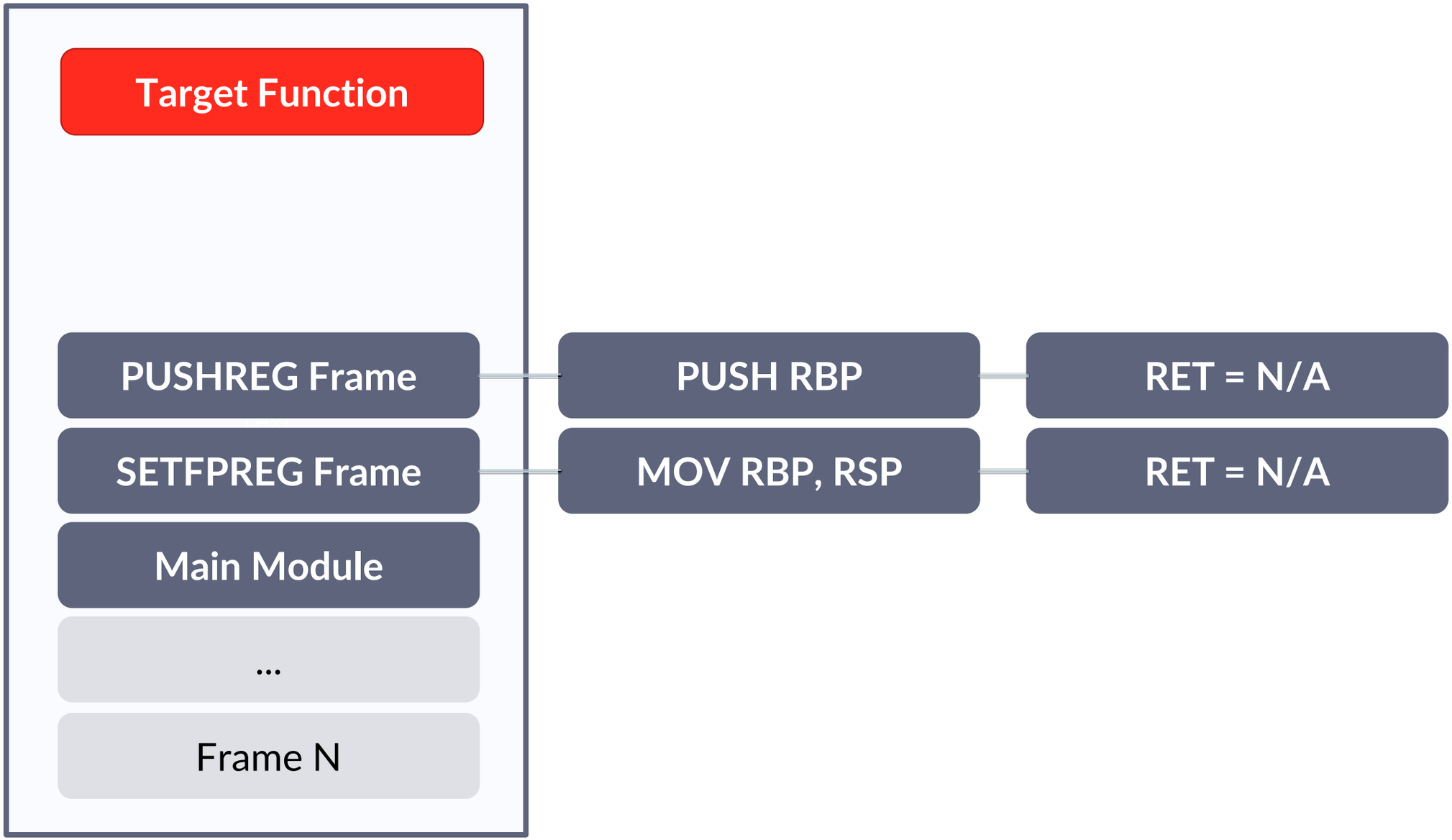
bsi.

Full Moon (Walk)



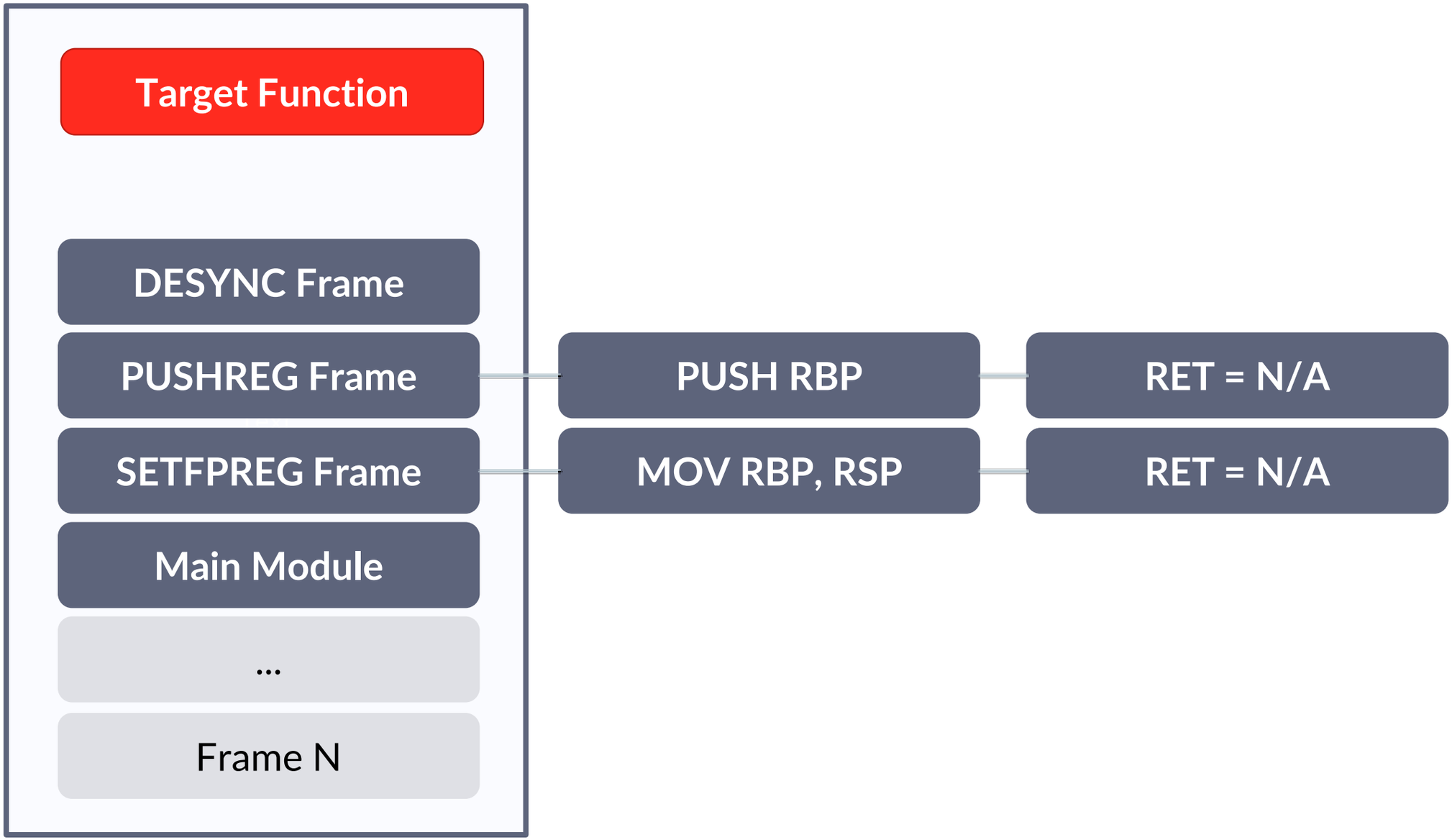
bsi.

Full Moon (Walk)



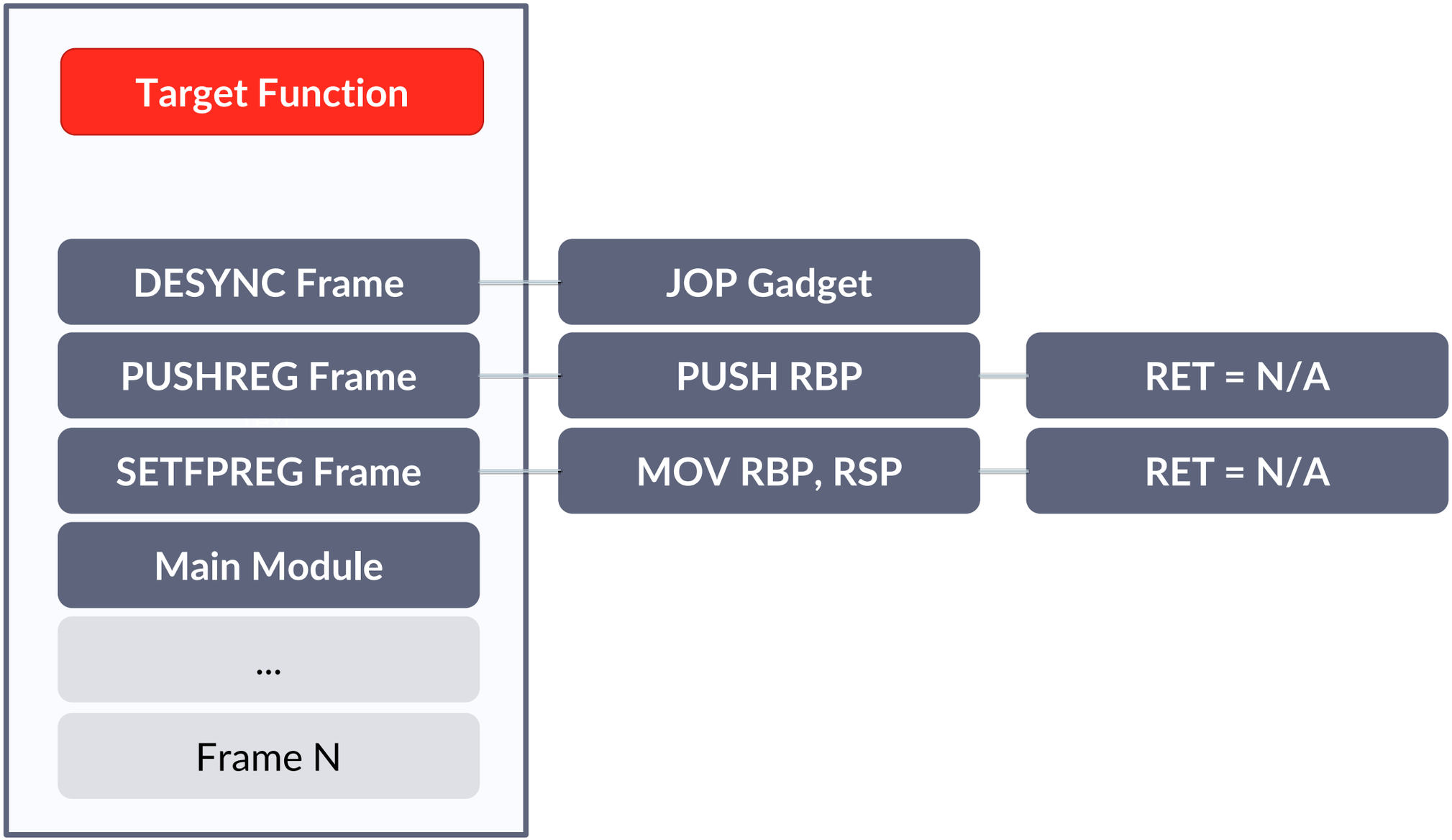
bsi.

Full Moon (Walk)



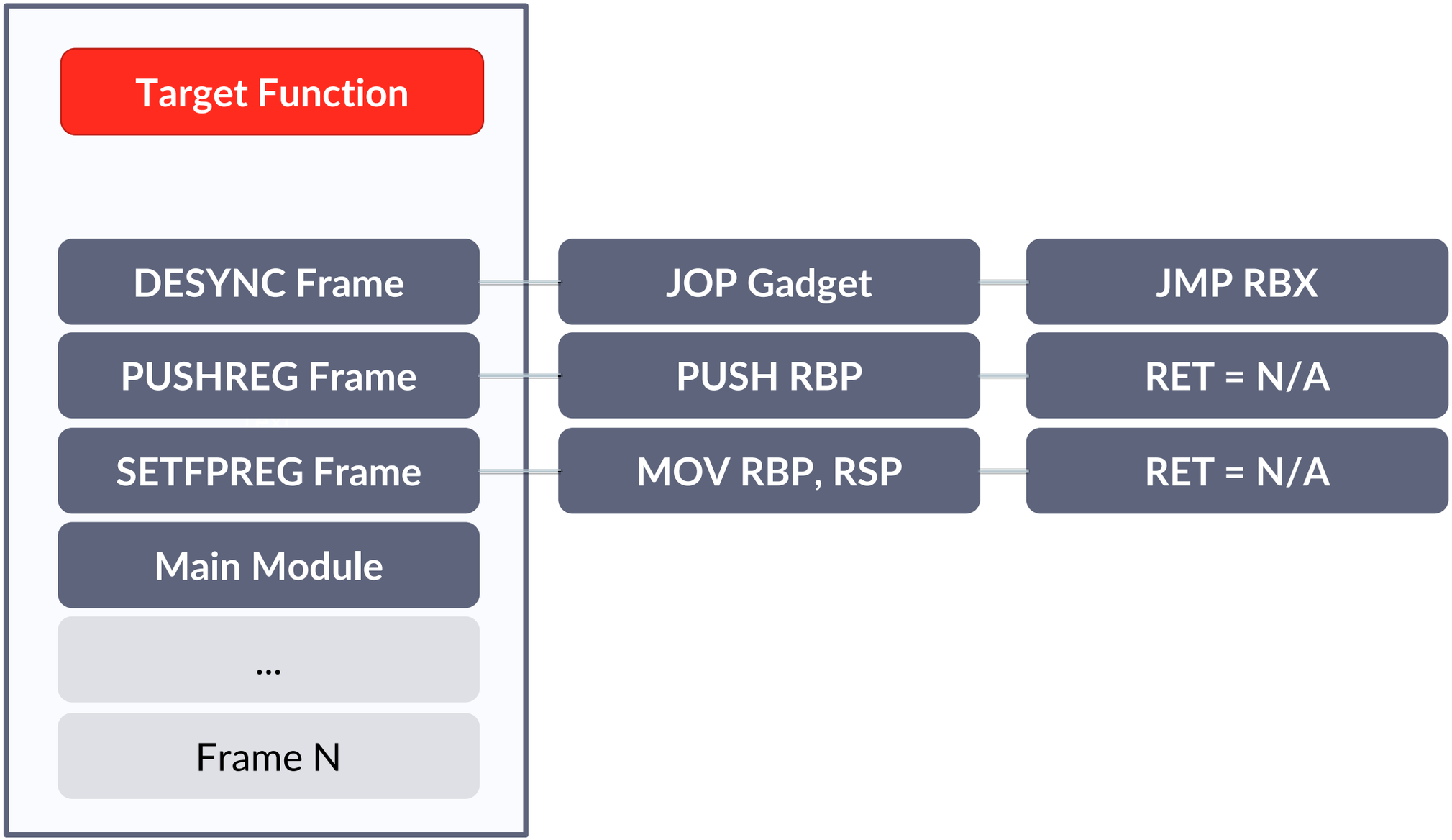
bsi.

Full Moon (Walk)



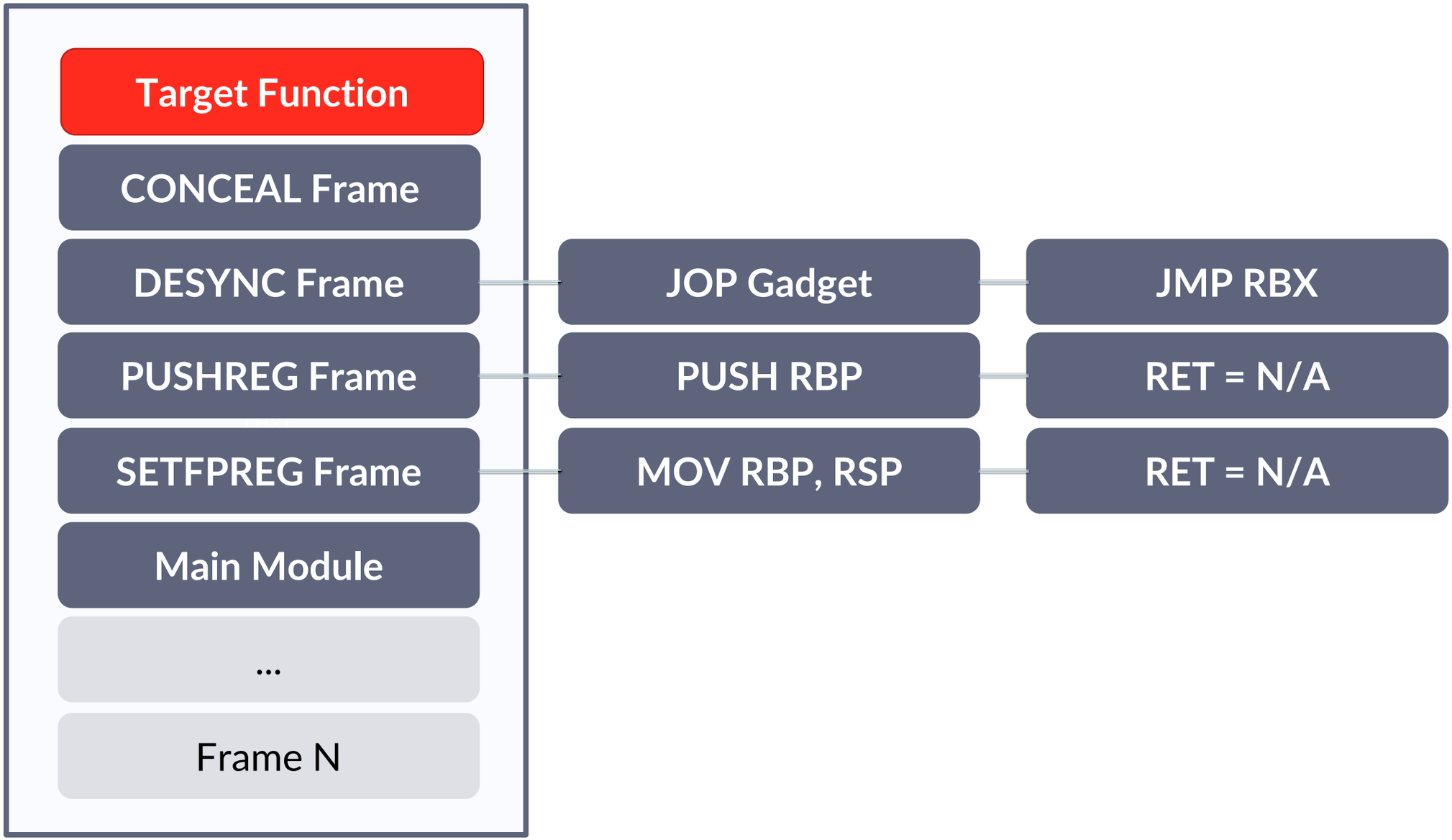
bsi.

Full Moon (Walk)



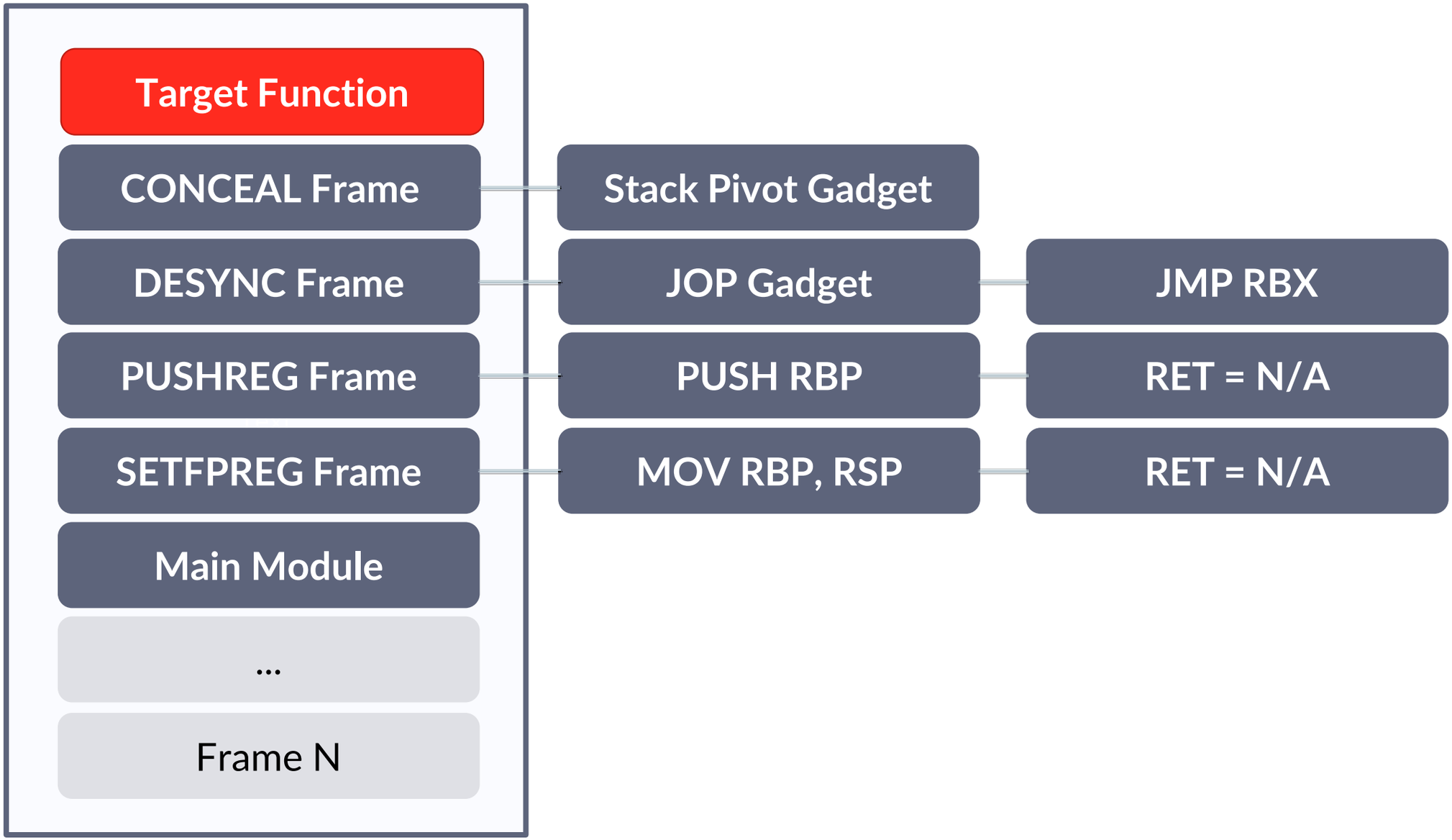
bsi.

Full Moon (Walk)



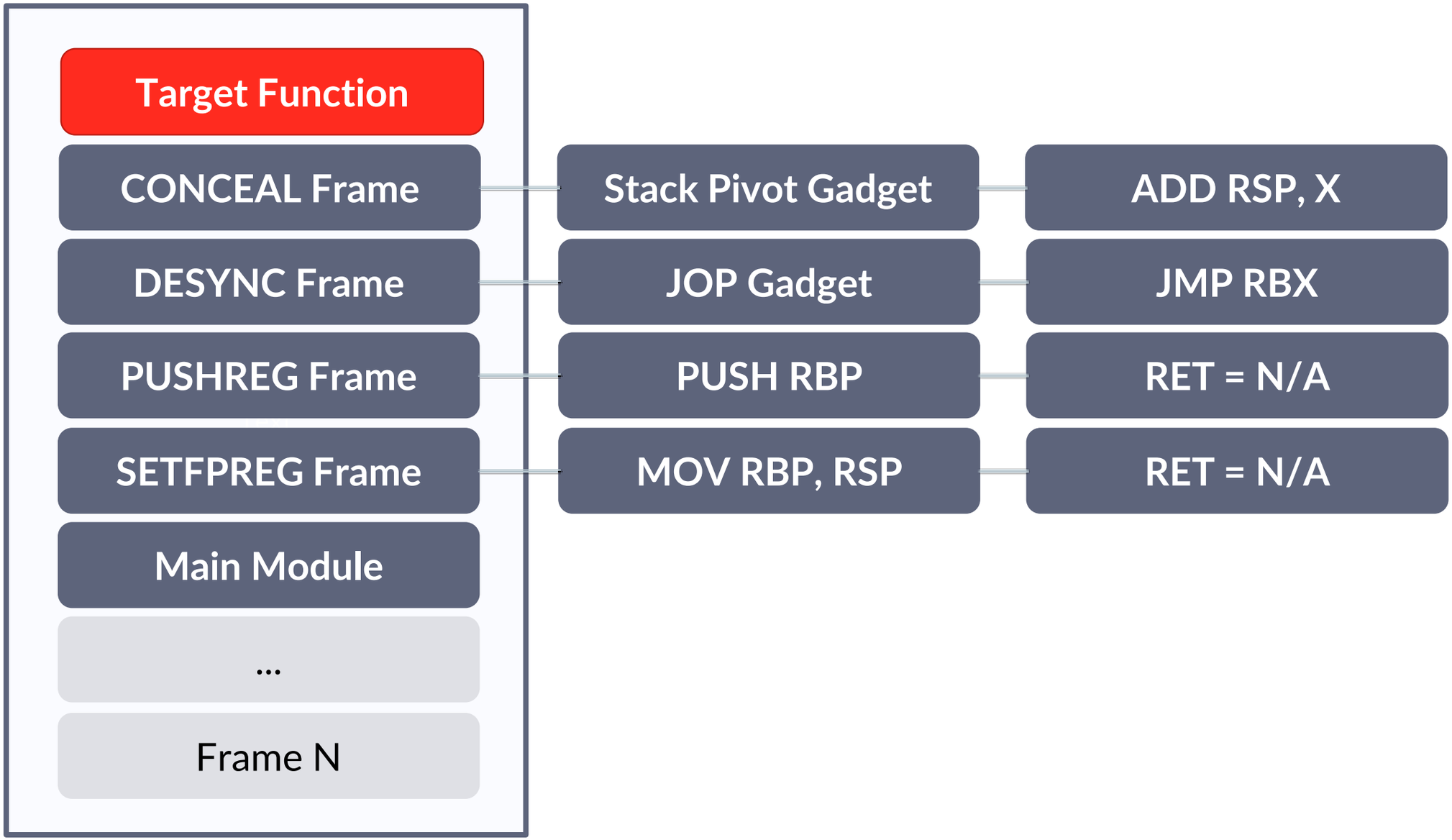
bsi.

Full Moon (Walk)



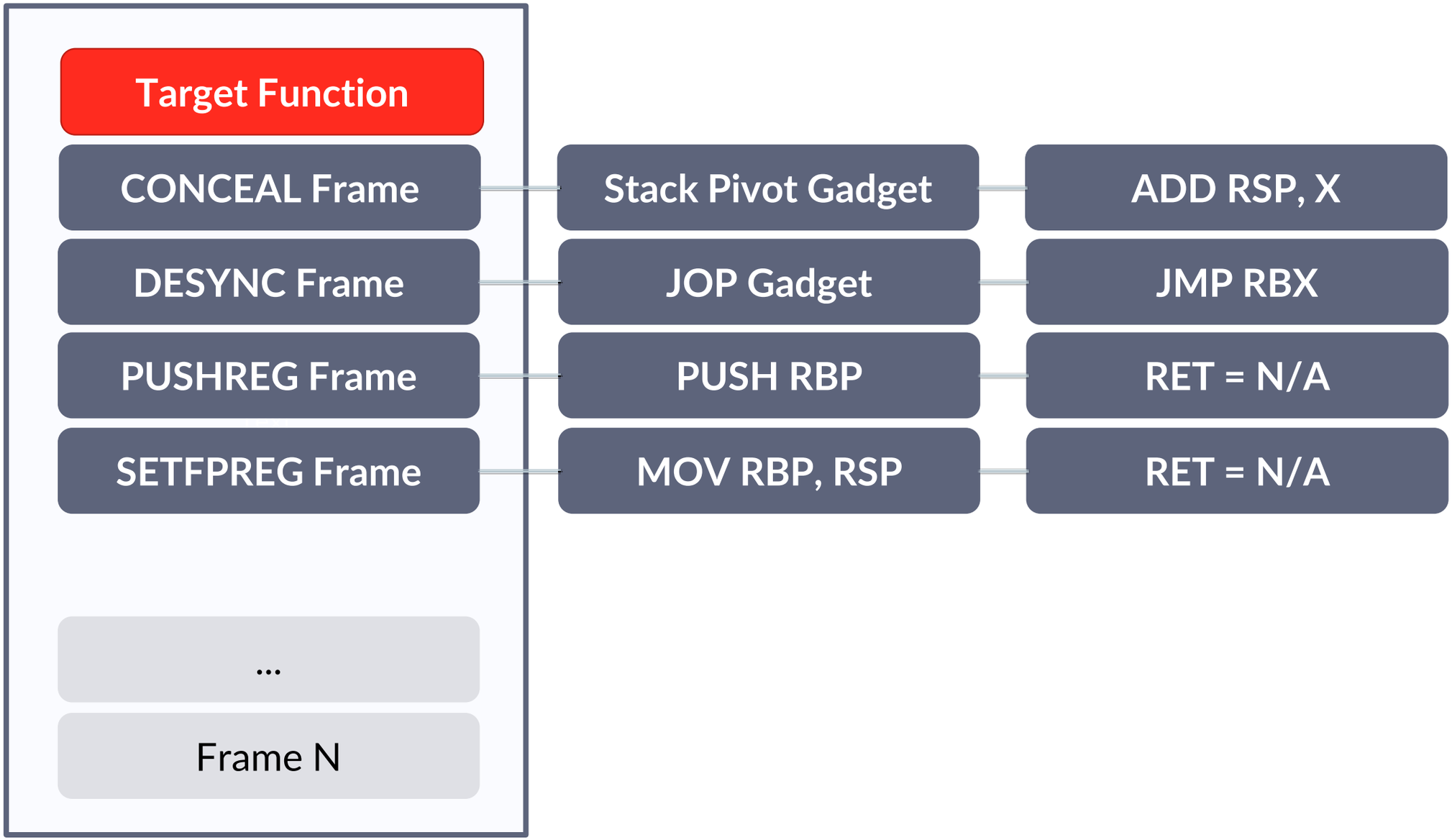
bsi.

Full Moon (Walk)



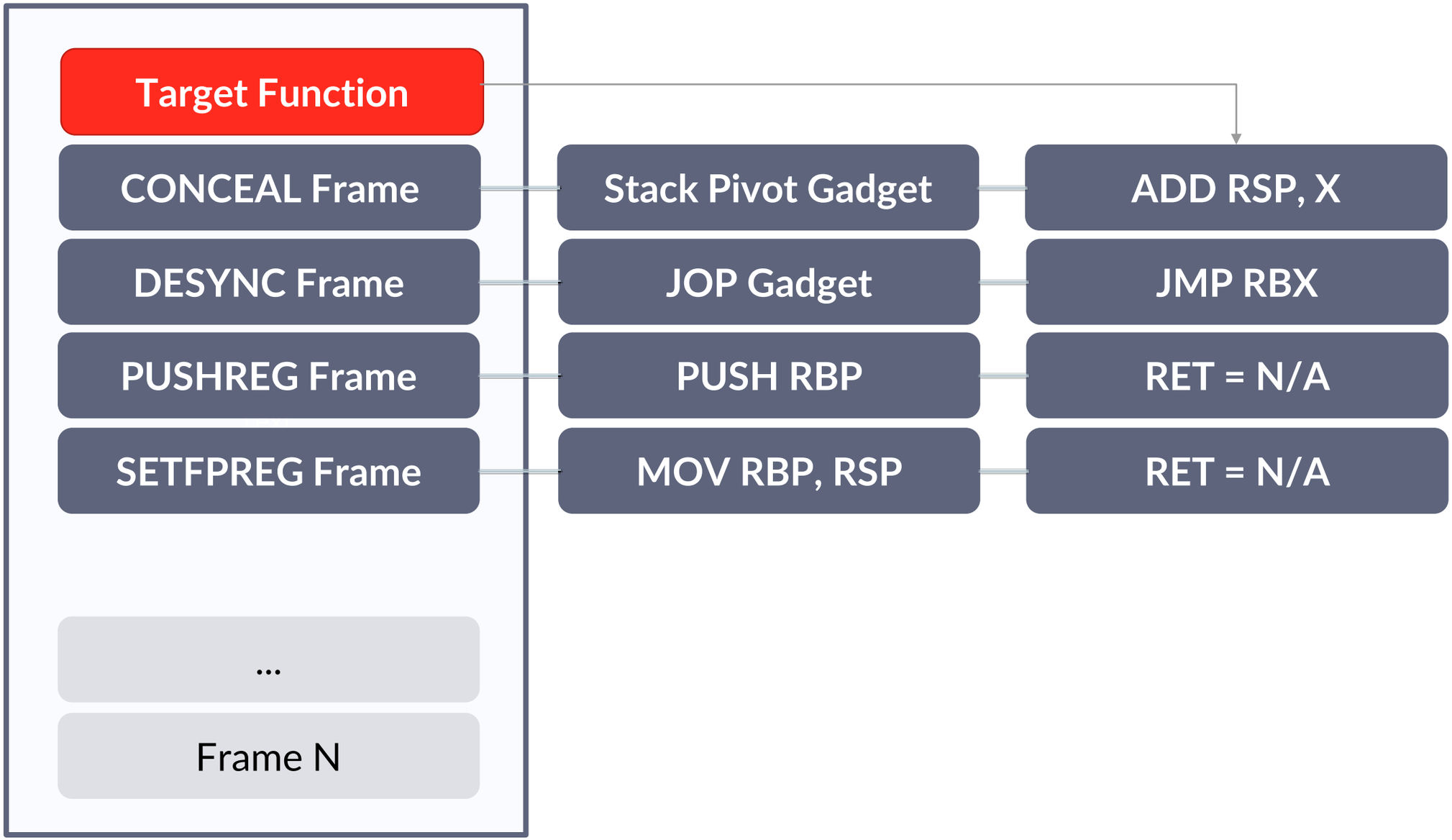
bsi.

Full Moon (Walk)



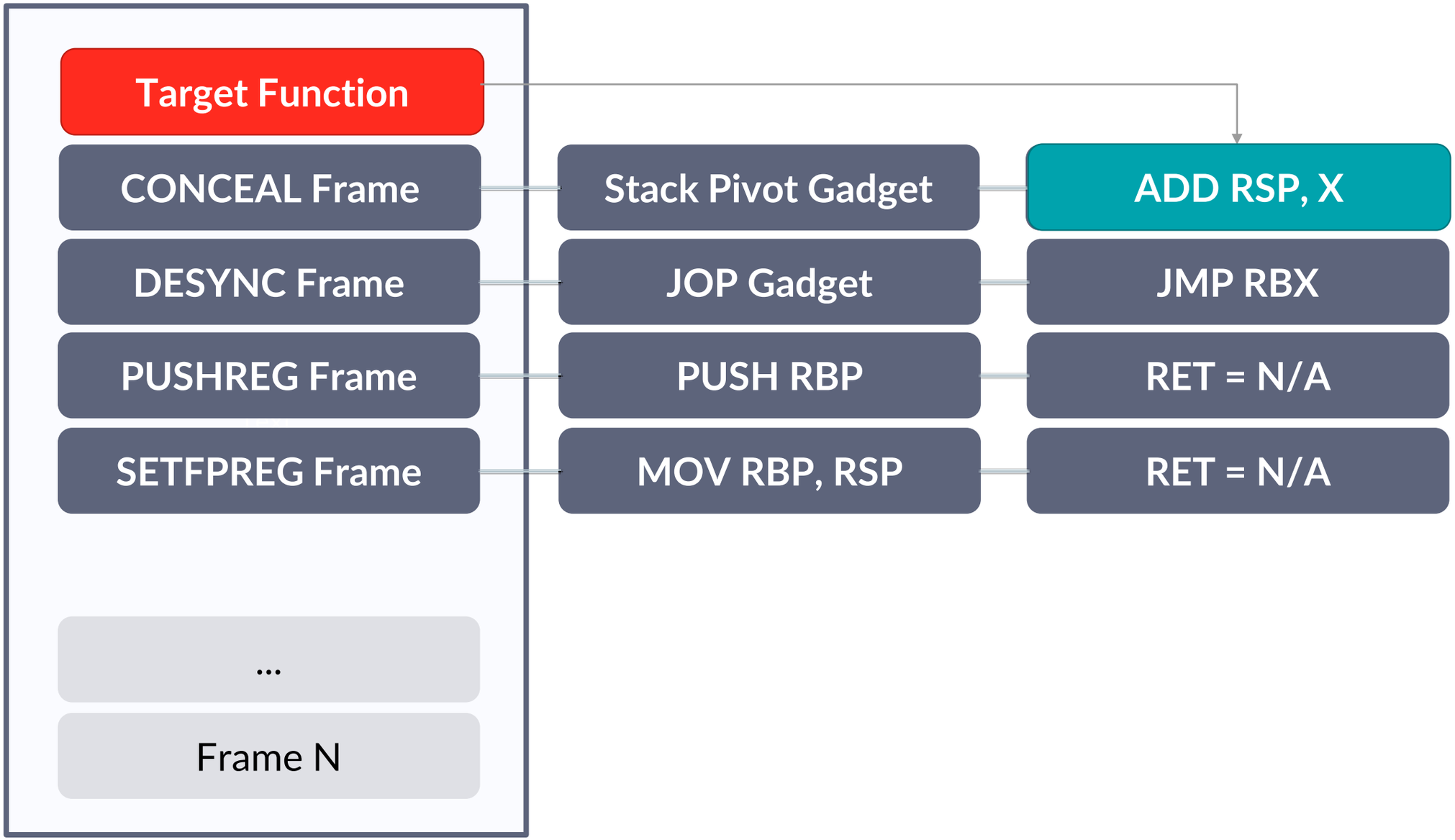
bsi.

Full Moon (Walk)



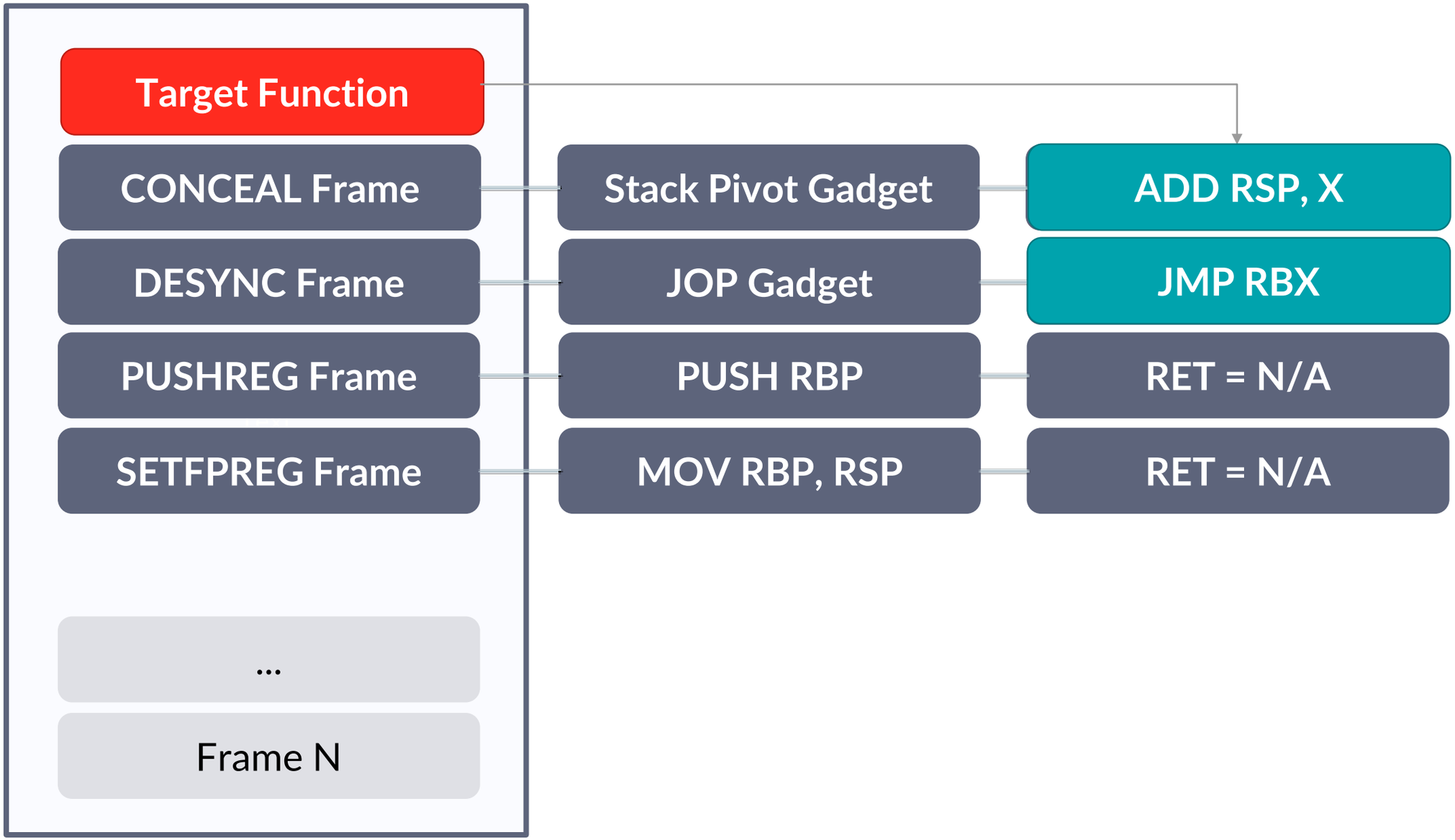
bsi.

Full Moon (Walk)



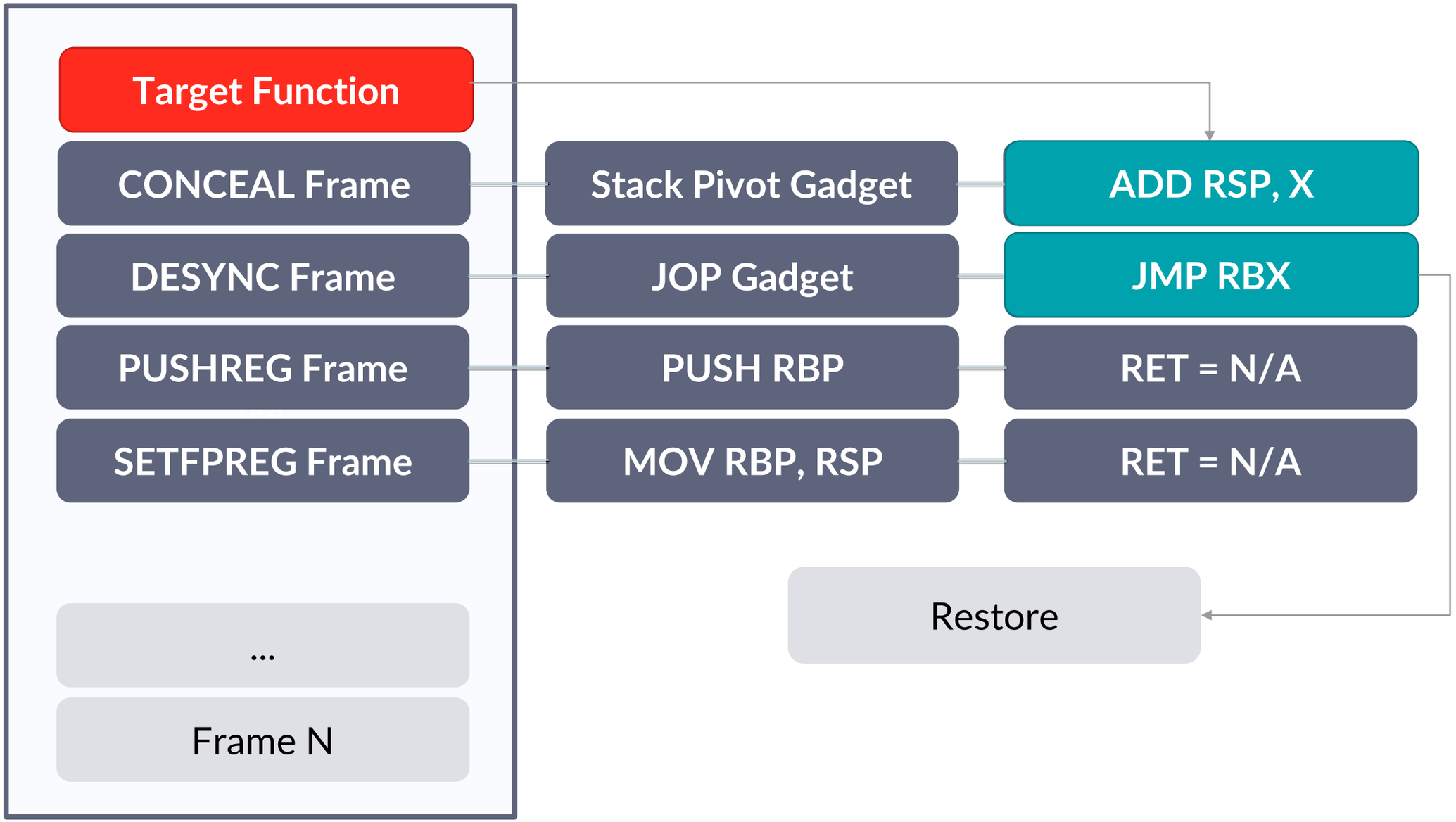
bsi.

Full Moon (Walk)



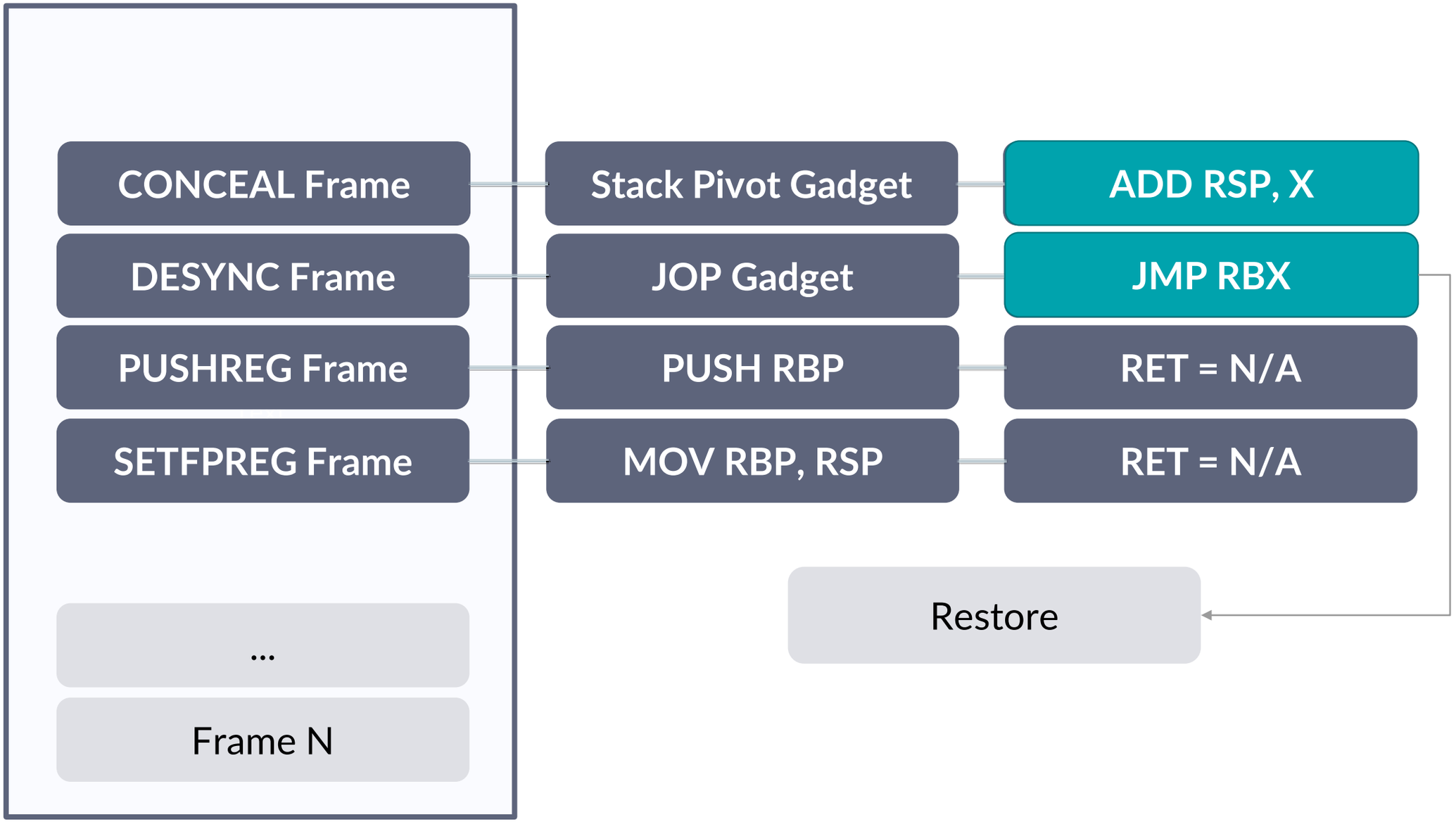
bsi.

Full Moon (Walk)



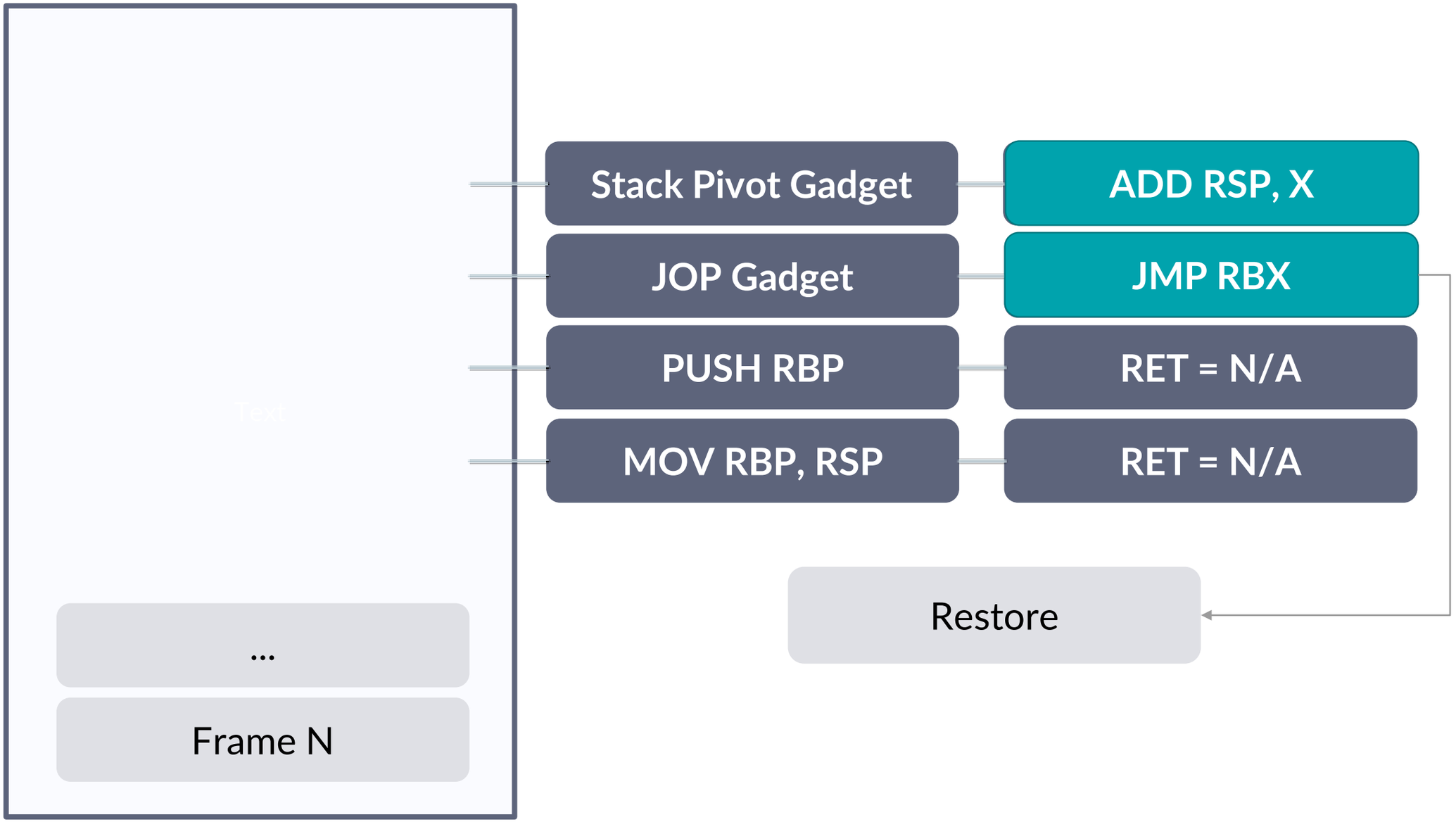
bsi.

Full Moon (Walk)



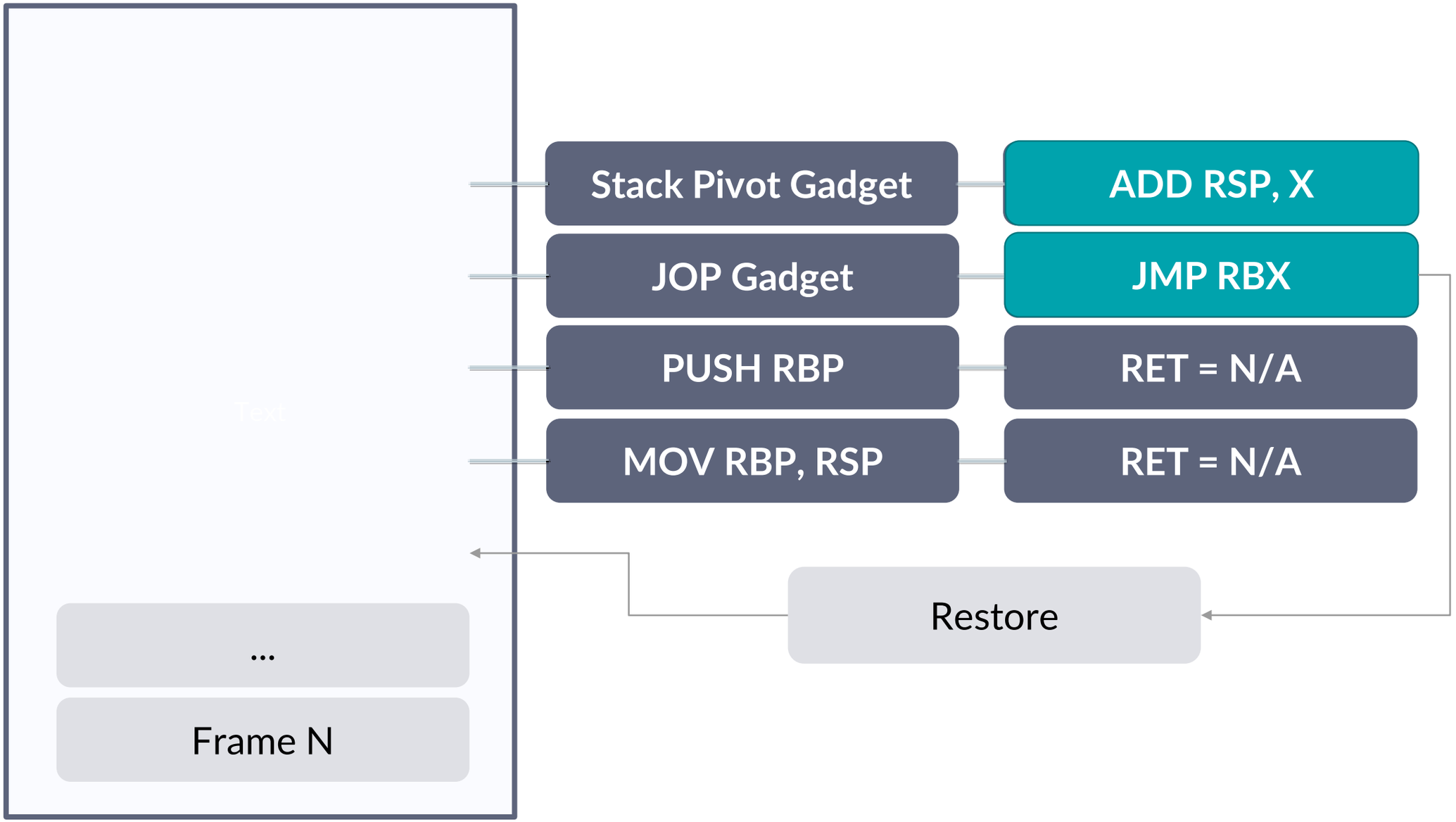
bsi.

Full Moon (Walk)



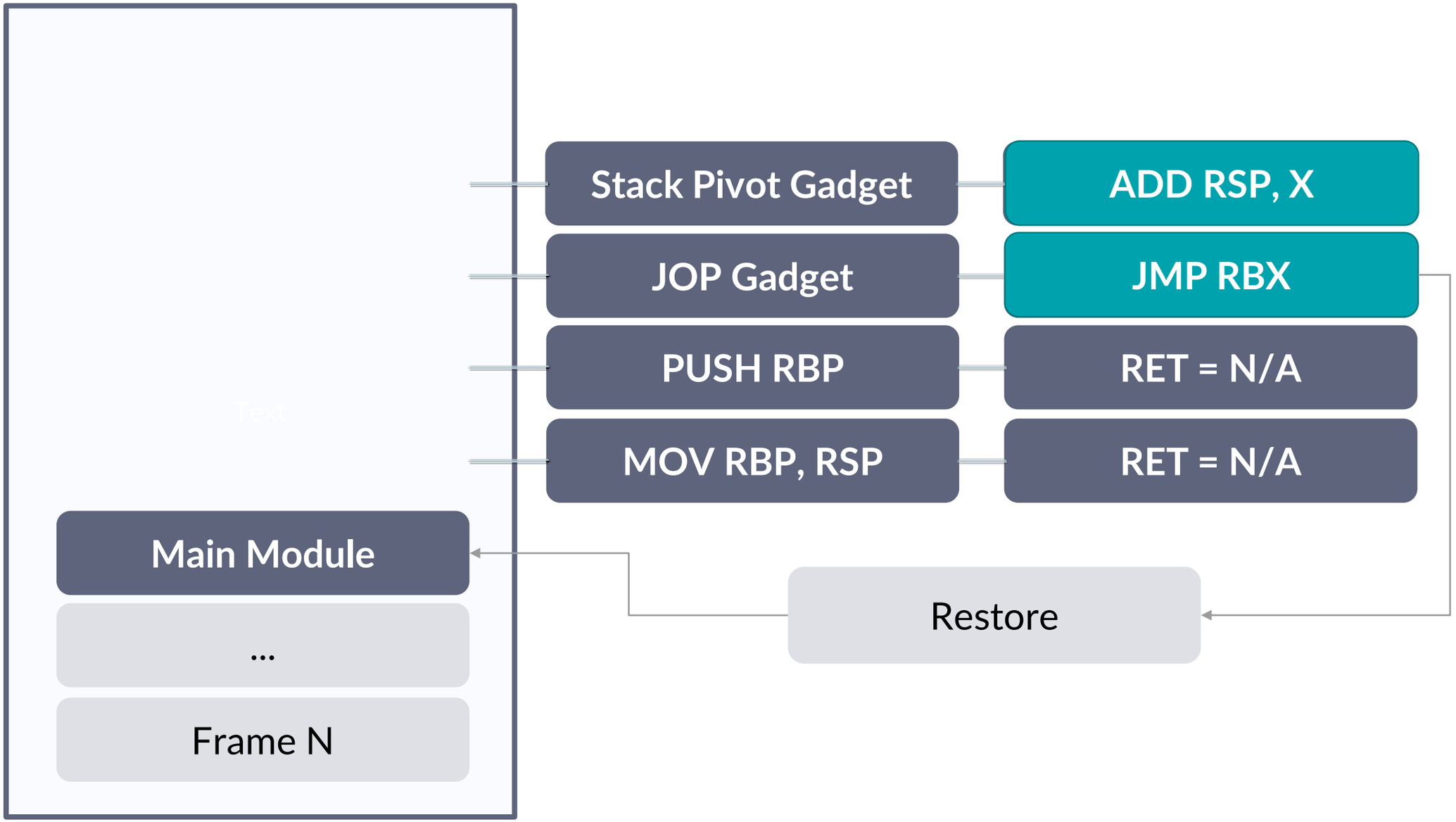
bsi.

Full Moon (Walk)



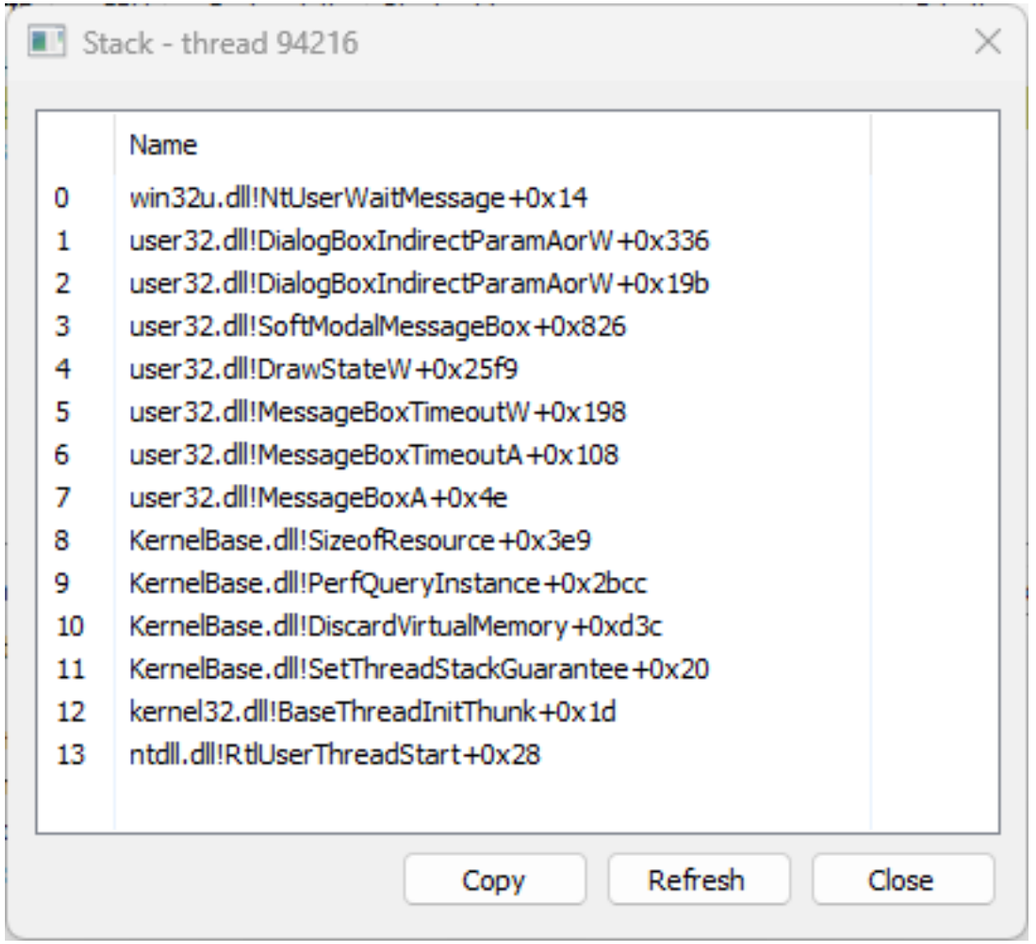
bsi.

Full Moon (Walk)

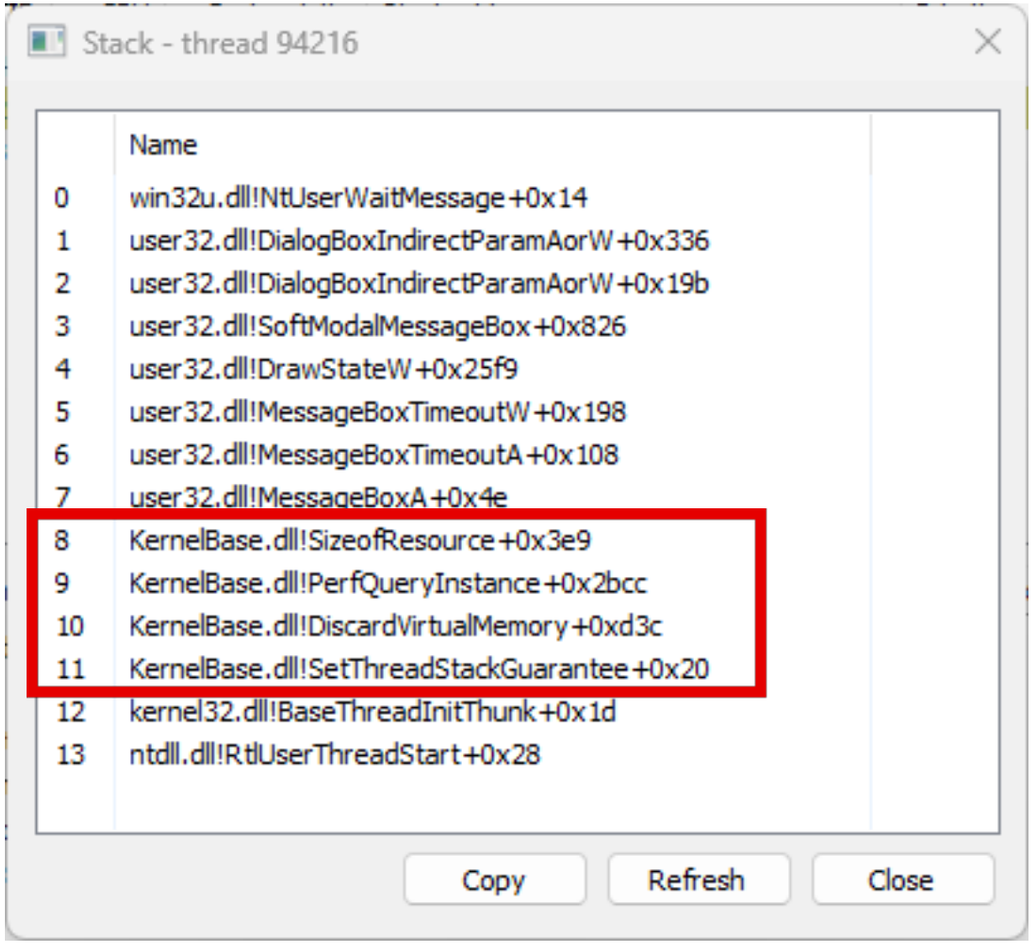


bsi.

Full Moon (Walk) - Frames



Full Moon (Walk) - Frames



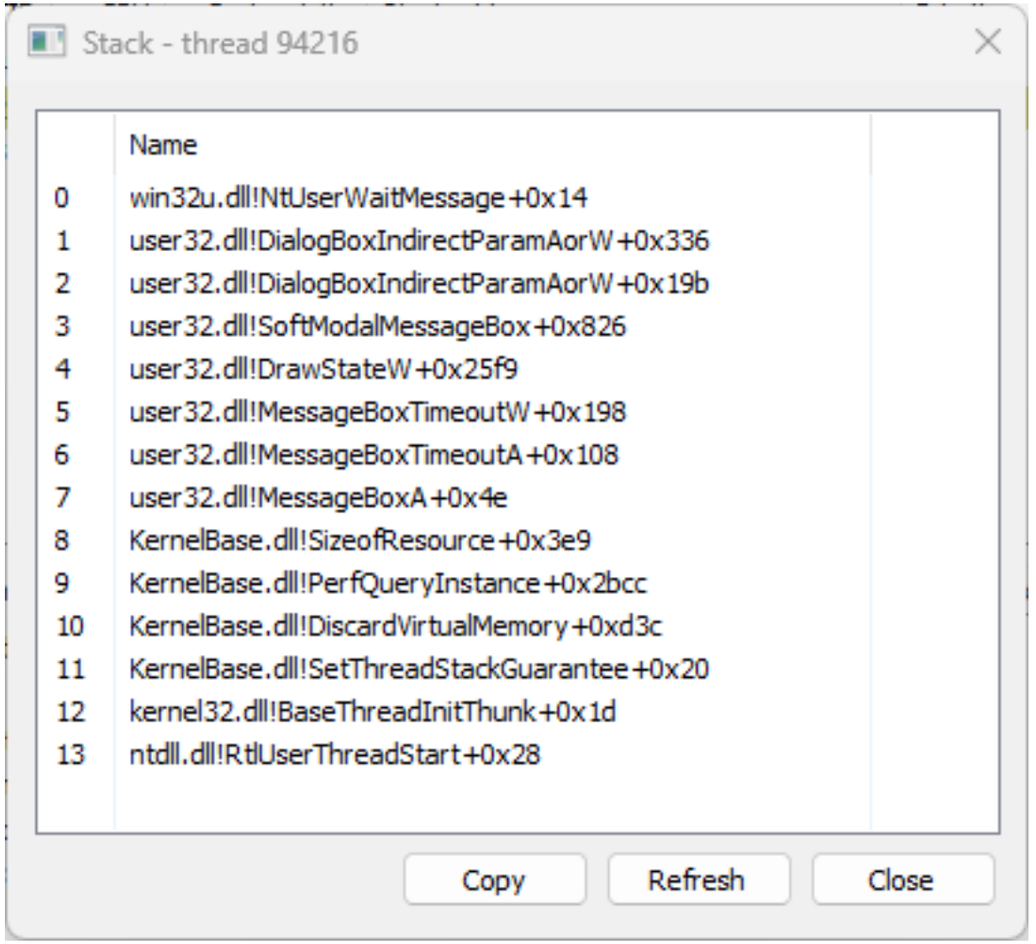
bsi.

Detecting Spoofed Frames

Designing valid detection strategies for moonwalking

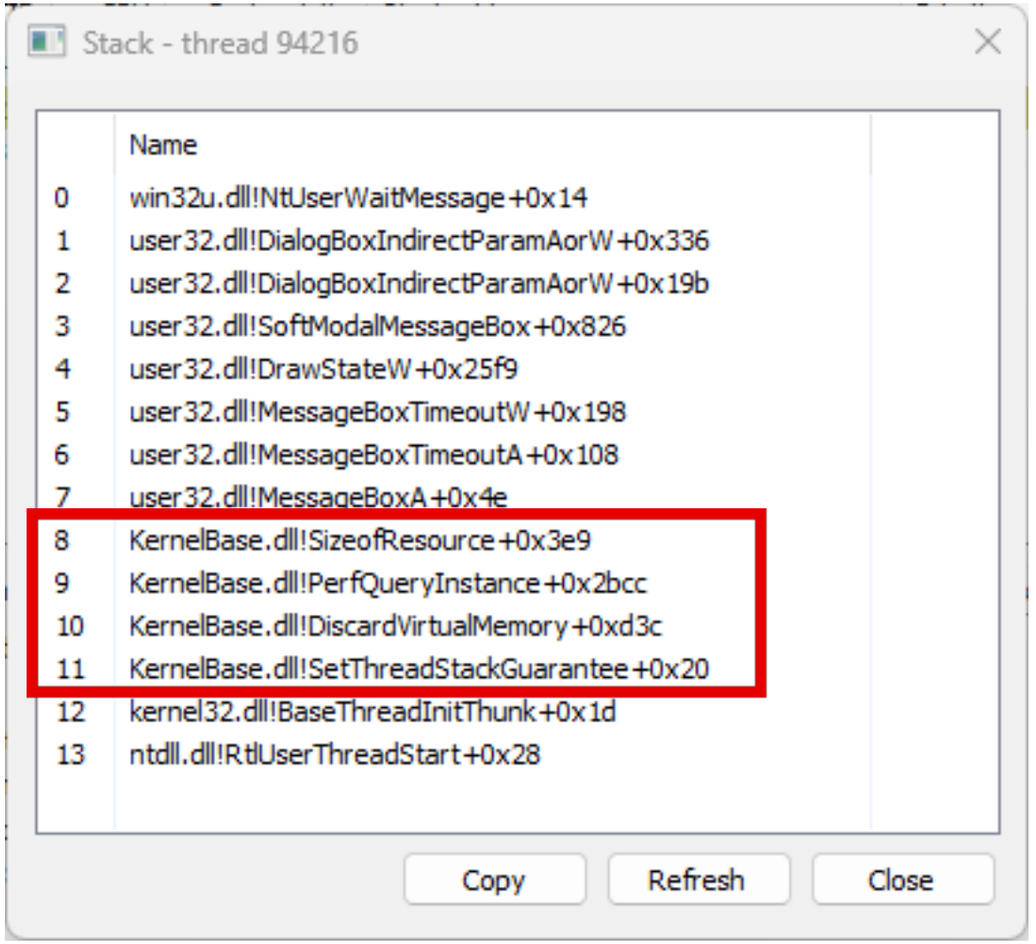
bsi.

Full Moon (Walk) - Frames



bsi.

Full Moon (Walk) - Frames



Eclipse


0:000> u user32!MessageBoxTimeoutA+0x108 - 5 L1
USER32!MessageBoxTimeoutA+0x103:
00007ffe`1961c723 e868000000 call USER32!MessageBoxTimeoutW (00007ffe`1961c790)
Legitimate Frame
0, win32u.dll!NtUserWaitMessage+0x14
1, user32.dll!FrameRect+0x27c
2, user32.dll!EndDialog+0x65b
3, user32.dll!SoftModalMessageBox+0x836
4, user32.dll!DrawStateA+0x1e11
5, user32.dll!MessageBoxTimeoutW+0x198
6, user32.dll!MessageBoxTimeoutA+0x108
7, user32.dll!MessageBoxA+0x4e
8, KernelBase.dll!CreatePrivateObjectSecurityWithMultipleInheritance+0x58
9, KernelBase.dll!OpenStateExplicit+0x39d
10, KernelBase.dll!DiscardVirtualMemory+0x39
11, KernelBase.dll!DiscardVirtualMemory+0x362
12, kernel32.dll!BaseThreadInitThunk+0x10
13, ntdll.dll!RtlUserThreadStart+0x2b
Malicious Frame
0:000> u KernelBase!OpenStateExplicit+0x39d - 5 L1
KERNELBASE!ReservedForLocalUse::MashThePublisherc
00007ffe`18b20198 884ada mov byte ptr [rdx-26h],cl

Eclipse

Expected CALL Instructions

Instruction	Opcode	Length
CALL MODR/M Rex.W 1	48 FF cd	0x7
CALL MODR/M Rex.W 0	FF cd	0x6
CALL REL32	E8 cd	0x5
CALL R64	[41] FF cb	0x2 - 0x3
SYSTEM CALL	0F 05	0x2

Possible CALL instructions observable before a frame return address

bsi.

Eclipse - Algorithm

Algorithm 2: Eclipse detection algorithm

Result: True if $Eclipse(f) = 0 \forall f \in s$
Input : s: Stack, current: Frame, strict: Boolean
Output: alarm: Integer

```
1  while s is not empty do
2      frame ← s.pop()
3      prev_s ← prev_call_instruction(frame)
4      retn_s ← instruction_on_return(frame)
5      if frame is not backed by a file on disk then
6          return REFLECTIVE_INJECTION_DETECTED
7      else if ret_s == DESYNC GADGET then
8          return STACK_MOONWALK_DETECTED
9      else if prev_s! = CALL then
10         return CALL_MISMATCH_DETECTED
11     else
12         address ← extract(prev_s)
           if address does not call current ∧ strict then
13             return CALL_WRONG_ADDRESS_DETECTED
14         else
15             current ← frame
16         end
17     end
18     return NO_ALERT;
19
```

Eclipse - Execution

```
Function BaseThreadInitThunk found. Stack size: 0x28 - Address: 0x7ffef88b2690
Function RtlUserThreadStart found. Stack size: 0x78 - Address: 0x7ffef988aa40
Runtime Function Table Size: 102312
Runtime Function Table Last Index: 8526
RT Function Table Range: 0x7FFEF7418000 - 0x7FFEF7430FA8
Return address: 0x244C8FFBD8
Address of Function to spoof: 0x7FFEF8176DB0

  ----------------------------------
Breaking at: 74
First Frame FP: 0x7FFEF70C54B4
First Frame stack size: 0x58
Return address: 0x7FFEF70C0160
Breaking at: 2
Second Frame FP: 0x7FFEF70C1470
Second Frame stack size: 0x148
Return address: 0x7FFEF70C0520
PUSH RBP offset: 0x140
Breaking at: 632
Gadget Address: 0x7FFEF70E4BCB
JMP [RBX] Frame Stack size: 0x168
Breaking at: 15
Gadget Address: 0x7FFEF70C245F
ADD RSP, X Frame Stack size: 0x38
```

Eclipse - Execution

#	Name
0	win32u.dll!NtUserWaitMessage+0x14
1	user32.dll!DialogBox2+0x172
2	user32.dll!InternalDialogBox+0x127
3	user32.dll!SoftModalMessageBox+0x826
4	user32.dll!MessageBoxWorker+0x341
5	user32.dll!MessageBoxTimeoutW+0x198
6	user32.dll!MessageBoxTimeoutA+0x108
7	user32.dll!MessageBoxA+0x4e
8	KernelBase.dll!wistd::function<long __cdecl(unsigned short *
9	KernelBase.dll!CreateFileInternal+0x34b
10	KernelBase.dll!PssQuerySnapshot+0x25
11	KernelBase.dll!QueryProcessMachine+0x82
12	kernel32.dll!BaseThreadInitThunk+0x1d
13	ntdll.dll!RtlUserThreadStart+0x28

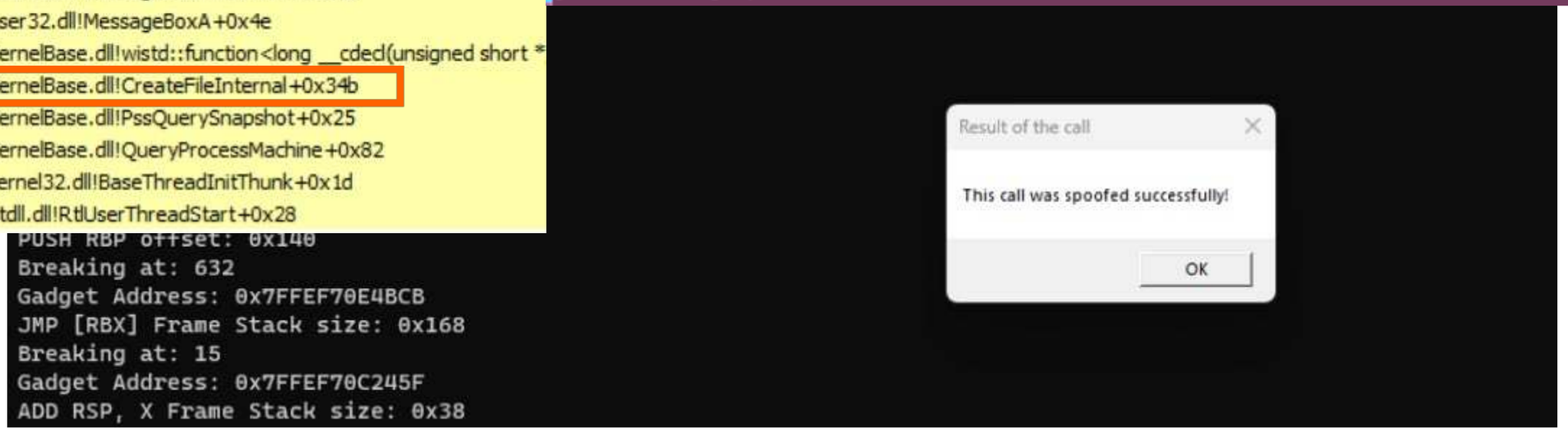
```
Analyzing ThreadID: 4436

Analyzed Function Frame: CreateFileW
 PC Address: 0x00007FFEF70E4BCB
 Stack Address: 0x000000244C8FF820
 Return Address: 0x00007FFEF70C1495
[X] Possibly spoofed return address observed!  There is a false caller observed here.
[!] This alert was generated because a call was not observed before the return address
 UNLESS this is a JIT process (such as a C Sharp Process).  Then FP's are expected.
[Information] Opcode observed at return address is: 0xF1
```

```
PUSH RBP offset: 0x140
Breaking at: 632
Gadget Address: 0x7FFEF70E4BCB
JMP [RBX] Frame Stack size: 0x168
Breaking at: 15
Gadget Address: 0x7FFEF70C245F
ADD RSP, X Frame Stack size: 0x38
```

Eclipse - Execution

#	Name
0	win32u.dll!NtUserWaitMessage+0x14
1	user32.dll!DialogBox2+0x172
2	user32.dll!InternalDialogBox+0x127
3	user32.dll!SoftModalMessageBox+0x826
4	user32.dll!MessageBoxWorker+0x341
5	user32.dll!MessageBoxTimeoutW+0x198
6	user32.dll!MessageBoxTimeoutA+0x108
7	user32.dll!MessageBoxA+0x4e
8	KernelBase.dll!wistd::function<long __cdecl(unsigned short *
9	KernelBase.dll!CreateFileInternal+0x34b
10	KernelBase.dll!PssQuerySnapshot+0x25
11	KernelBase.dll!QueryProcessMachine+0x82
12	kernel32.dll!BaseThreadInitThunk+0x1d
13	ntdll.dll!RtlUserThreadStart+0x28

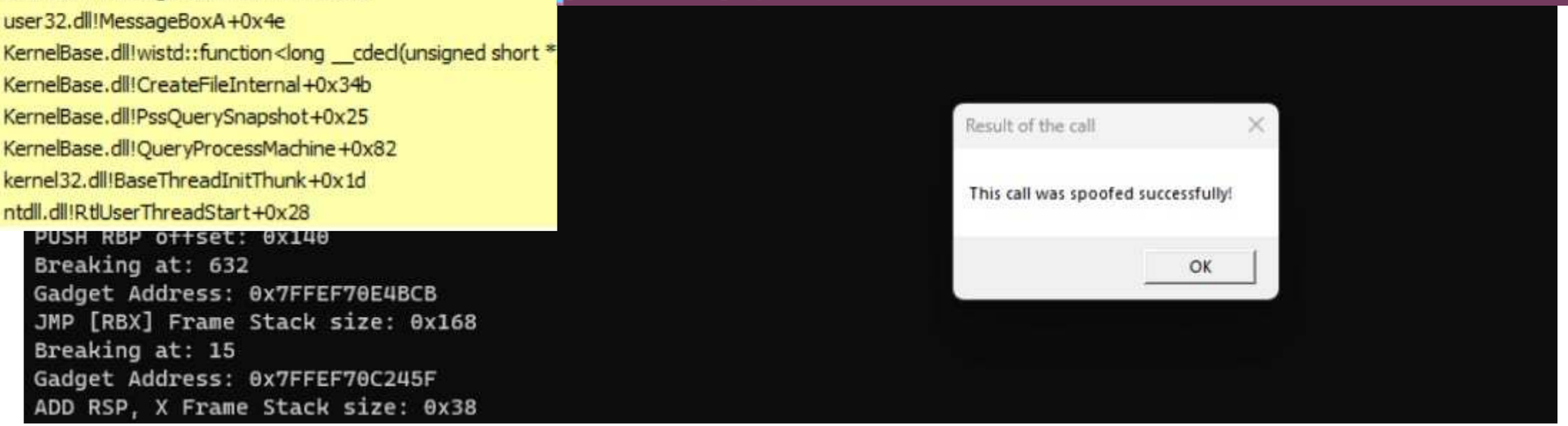
```
Analyzing ThreadID: 4436

Analyzed Function Frame: CreateFileW
 PC Address: 0x00007FFEF70E4BCB
 Stack Address: 0x000000244C8FF820
 Return Address: 0x00007FFEF70C1495
[X] Possibly spoofed return address observed!  There is a false caller observed here.
[!] This alert was generated because a call was not observed before the return address
 UNLESS this is a JIT process (such as a C Sharp Process).  Then FP's are expected.
[Information] Opcode observed at return address is: 0xF1
```



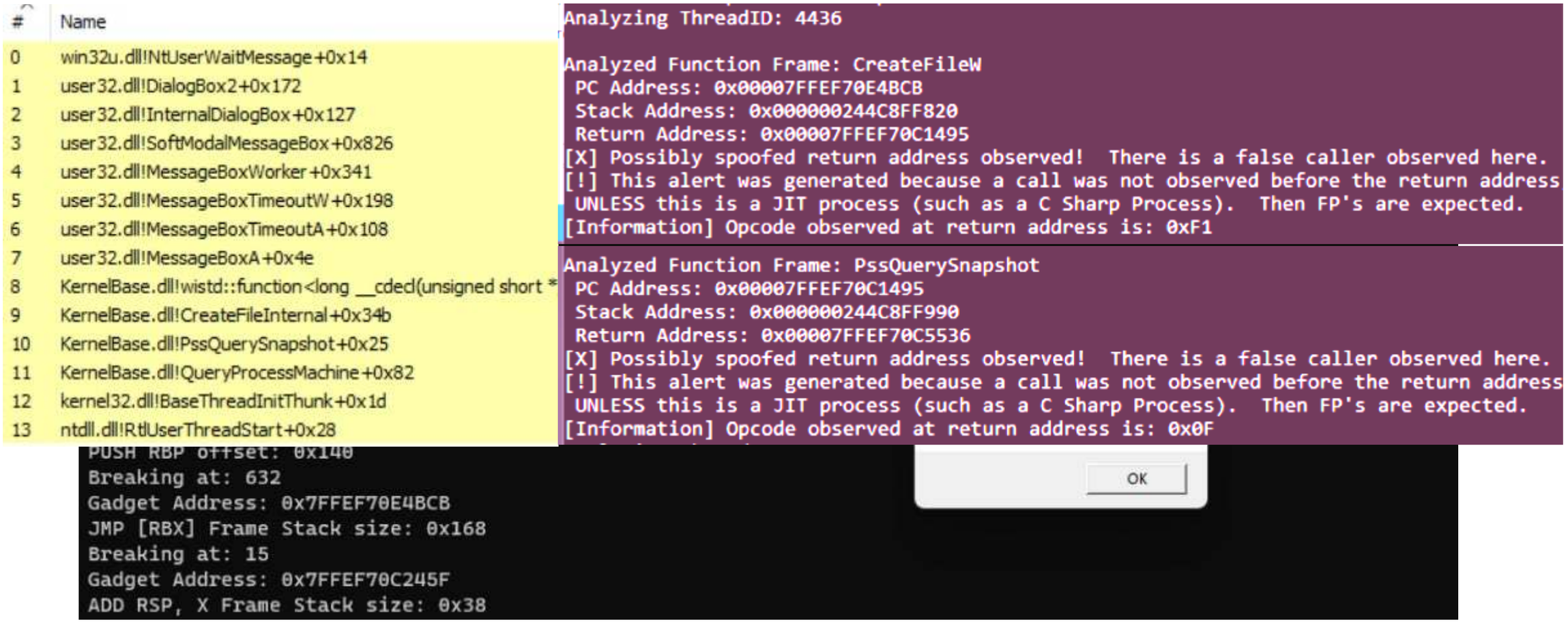
Eclipse - Execution

#	Name
0	win32u.dll!NtUserWaitMessage+0x14
1	user32.dll!DialogBox2+0x172
2	user32.dll!InternalDialogBox+0x127
3	user32.dll!SoftModalMessageBox+0x826
4	user32.dll!MessageBoxWorker+0x341
5	user32.dll!MessageBoxTimeoutW+0x198
6	user32.dll!MessageBoxTimeoutA+0x108
7	user32.dll!MessageBoxA+0x4e
8	KernelBase.dll!wistd::function<long __cdecl(unsigned short *
9	KernelBase.dll!CreateFileInternal+0x34b
10	KernelBase.dll!PssQuerySnapshot+0x25
11	KernelBase.dll!QueryProcessMachine+0x82
12	kernel32.dll!BaseThreadInitThunk+0x1d
13	ntdll.dll!RtlUserThreadStart+0x28

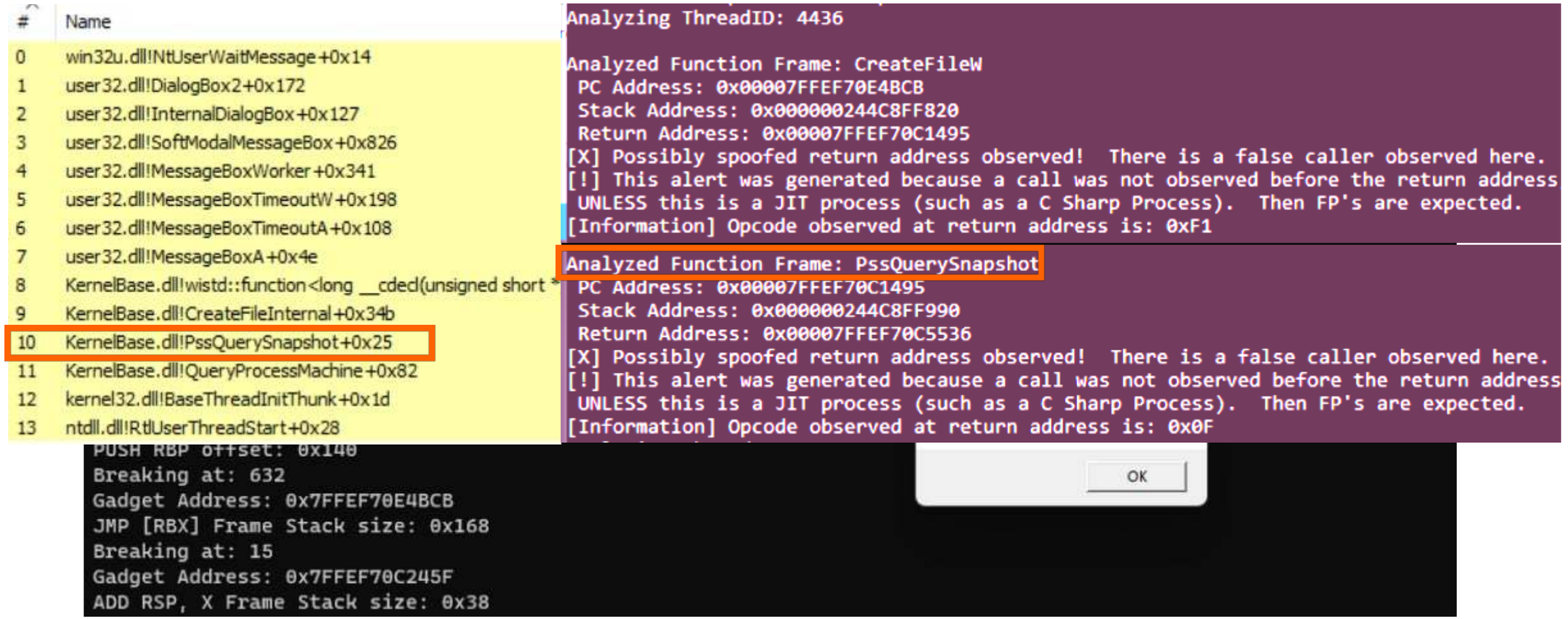
```
Analyzing ThreadID: 4436

Analyzed Function Frame: CreateFileW
 PC Address: 0x00007FFEF70E4BCB
 Stack Address: 0x000000244C8FF820
 Return Address: 0x00007FFEF70C1495
[X] Possibly spoofed return address observed!  There is a false caller observed here.
[!] This alert was generated because a call was not observed before the return address
 UNLESS this is a JIT process (such as a C Sharp Process).  Then FP's are expected.
[Information] Opcode observed at return address is: 0xF1
```



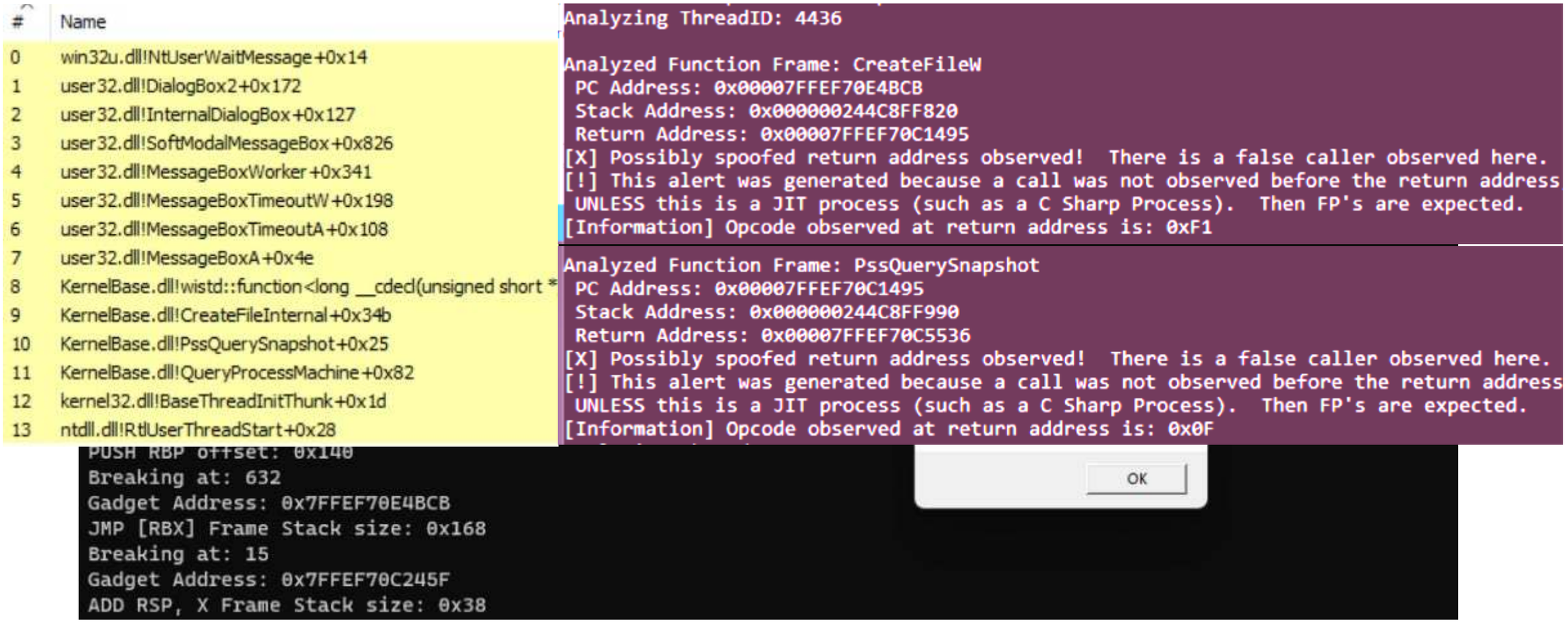
Eclipse - Execution



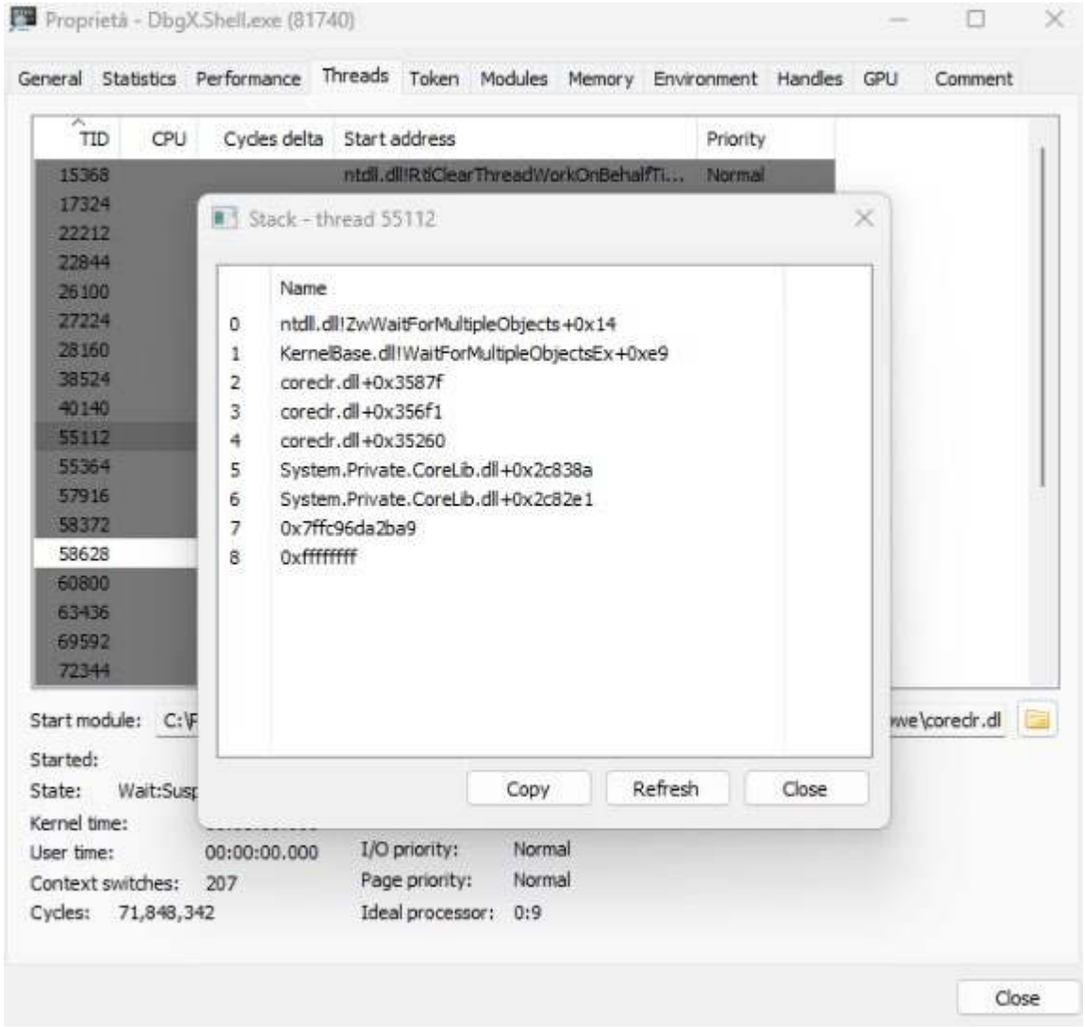
Eclipse - Execution



Eclipse - Execution



Eclipse – False Positives



Eclipse – False Positives

Machine	Processes Count	Alerts	False Positive Rate
Windows 10 Pro (std)	190	2	1.05%
Windows 10 Pro (dev)	350	15	4.29%
Windows 11 Enterprise (std)	176	4	2.27%
Windows 11 Enterprise (dev)	330	12	3.64%
Windows Server 2019	160	5	3.13%

bsi.

Eclipse – False Positives

Machine	Processes Count	Alerts	False Positive Rate
Windows 10 Pro (std)	190	2	1.05%
Windows 10 Pro (dev)	350	15	4.29%
Windows 11 Enterprise (std)	176	4	2.27%
Windows 11 Enterprise (dev)	330	12	3.64%
Windows Server 2019	160	5	3.13%

bsi.

Eclipse – False Positives

Machine	Processes Count	Alerts	False Positive Rate
Windows 10 Pro (std)	190	2	1.05%
Windows 10 Pro (dev)	350	15	4.29%
Windows 11 Enterprise (std)	176	4	2.27%
Windows 11 Enterprise (dev)	330	12	3.64%
Windows Server 2019	160	5	3.13%

bsi.

Eclipse – False Positives

Machine	Processes Count	Alerts	False Positive Rate
Windows 10 Pro (std)	190	2	1.05%
Windows 10 Pro (dev)	350	15	4.29%
Windows 11 Enterprise (std)	176	4	2.27%
Windows 11 Enterprise (dev)	330	12	3.64%
Windows Server 2019	160	5	3.13%

bsi.

Towards the future

Additional research avenues and where to find them

Key Takeaways

01 Public POC:
https://github.com/klezVirus/SilentMoonwalk

02 This POC is designed to obfuscate the stack at runtime

03 For sleep obfuscation an approach like Stack Hiding or Stack Cloning is preferred

04 It is possible to use other JOP/COP/ROP gadgets

05 It is possible to create chains of different length by inserting stack pivot frames

06 It is difficult but not impossible to create completely valid call stacks

bsi.

Unanswered Questions

1. How would I use this technique within an implant?
2. Is it possible to integrate the technique with other advanced evasion techniques?
3. Is it possible to bypass Eclipse detection strategy?
4. What about Intel CET/AMD Shadow Stack? Would it prevent the technique?

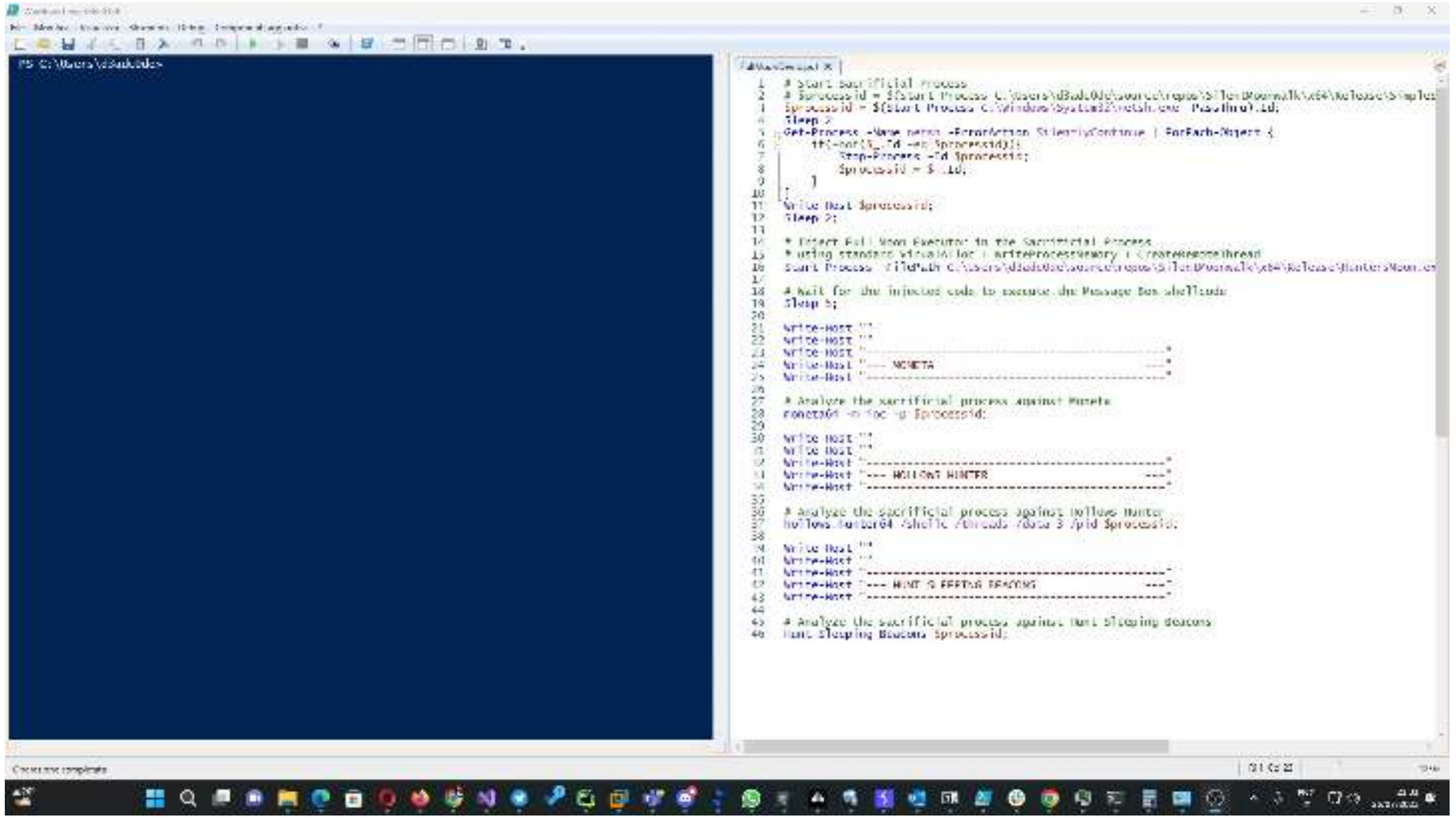
NO HAT 2023

COMPUTER SECURITY CONFERENCE
21ST OF OCTOBER 2023, BERGAMO - ITALY

bsi.

Sneak Peak

https://www.youtube.com/watch?v=7jW558VW0nk




Thanks!