Memory Corruption

The (almost) Complete History...

haroon meer - 2010
@haroonmeer | haroon@thinkst.com

thinkst
applied research

Who ?

- haroon meer
- thinkst ?
- some papers, some books, some talks
- academic wannabe

thinkst
applied research

Why?

Why?

twitter

Home Profile Find People Settings Help Sign out

Walking down memory lane, reading old exploits from '99 -- can someone write a history of code exec '95-2009 ?

1:55 AM Nov 20th, 2009 via web

Reply Retweet

halvarflake

thinkst
applied research

Why?

twitter

Home Profile Find People Settings Help Sign out

The ROP discussion is amusing in the sense that our folklore gets republished, and then we are asked "what papers have you published" ? :)

6:28 PM Feb 28th via web

Retweeted by 2 people

Reply Retweet

halvarflake

thinkst applied research

Why?

twitter

Home Profile Find People Settings Help Sign out

Each time your exploit heap spray, someone somewhere think he just pioneer an exploitation technique invented in the 90's

about 20 hours ago via web

Retweeted by 4 people

Reply Retweet

nicowaisman

Nico Waisman

thinkst applied research

Why?

twitter made me do it!

thinkst
applied research

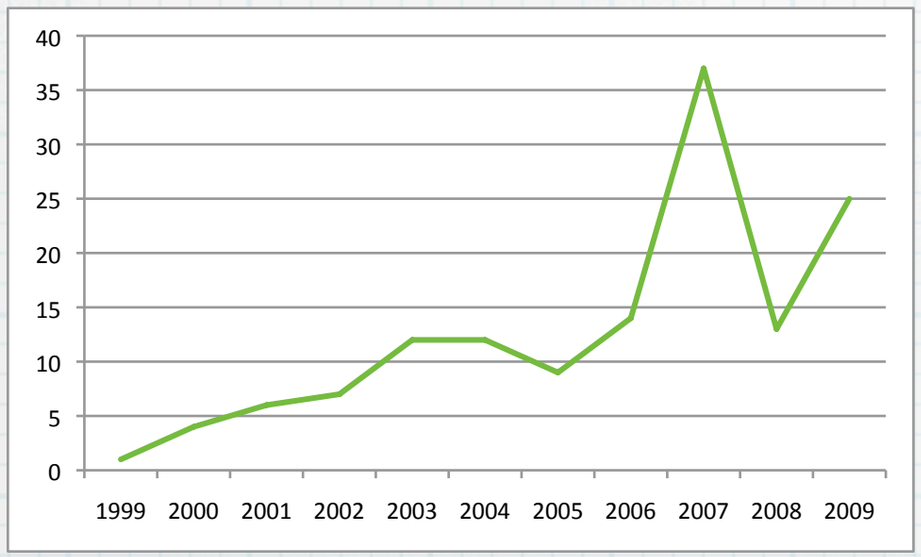
Why?

- de-mystify some of the otherwise mystical
- convince you that Solar Designer was skynet

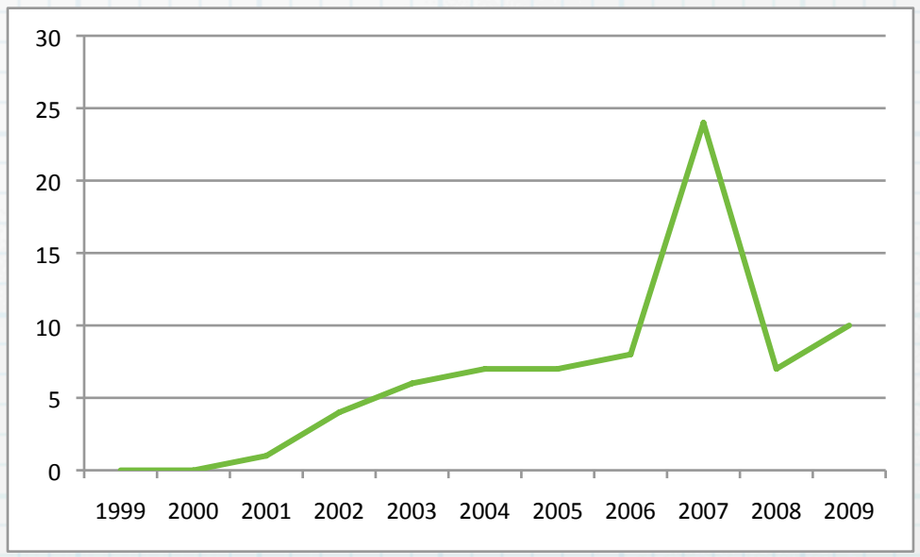
thinkst
applied research

Why?

(Some silly Stats)



Stack : 140

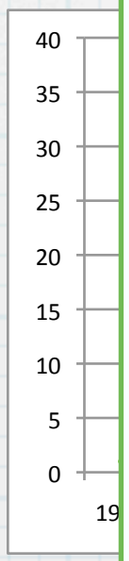


Heap : 74

thinkst
applied research

Why?

Stack spraying is definitely impressive!



thinkst
applied research

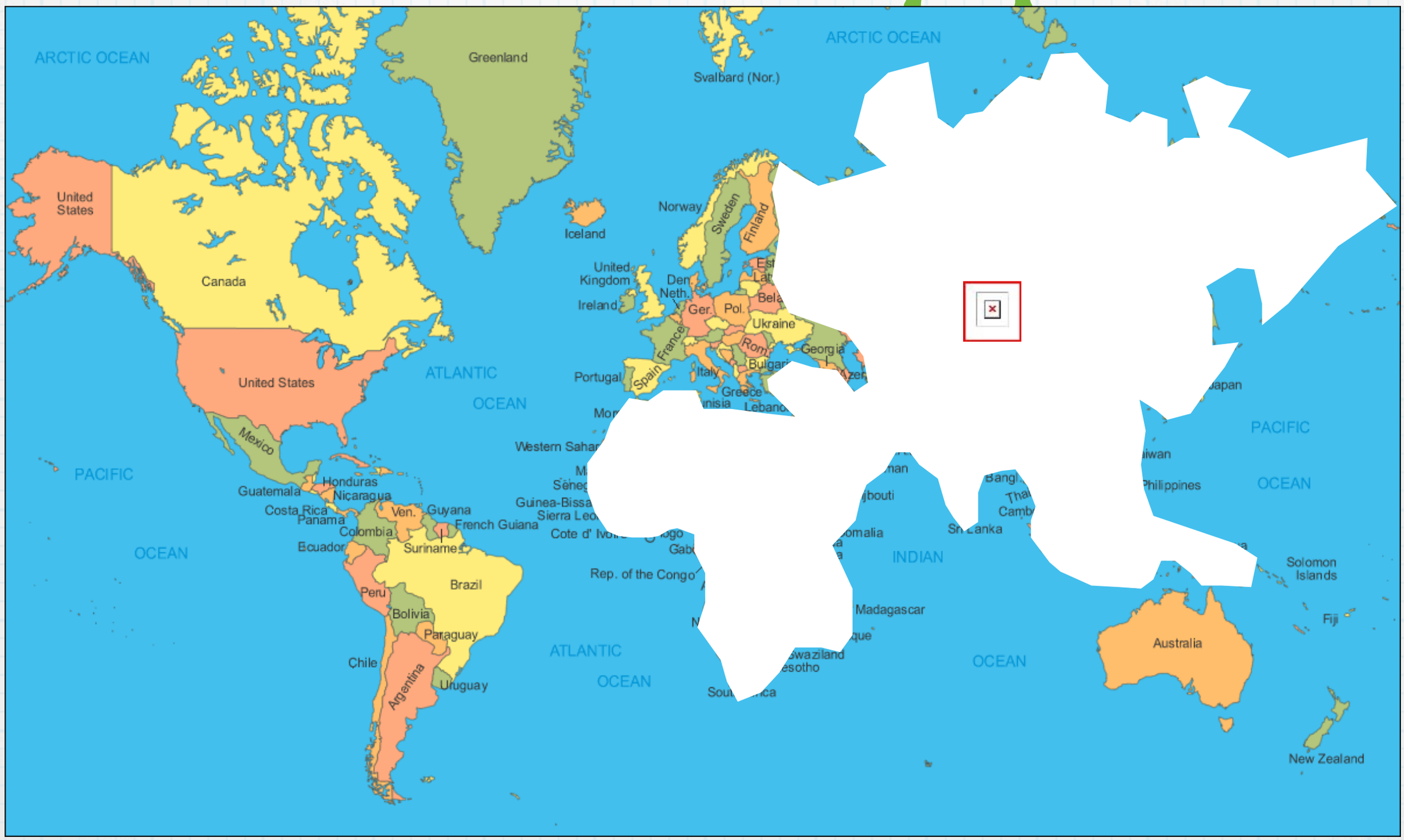
Caveats - Limits

Caveats - Limits

thinkst
applied research

Caveats - Myopia

Caveats - Myopia



thinkst
applied research

Caveats -Compression Ratio

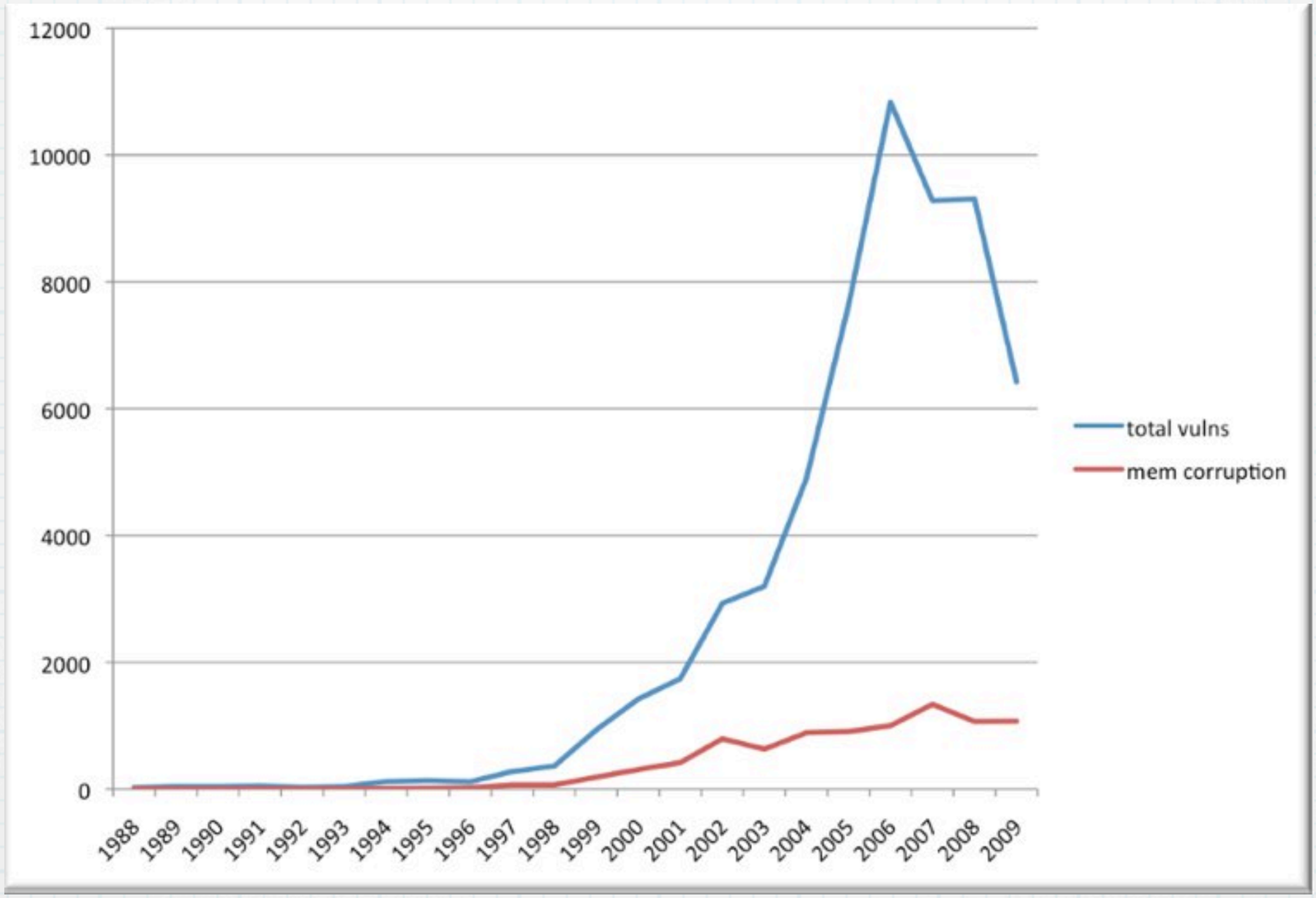
332880 : 1

thinkst
applied research

Disclosure, Bugs and Counts

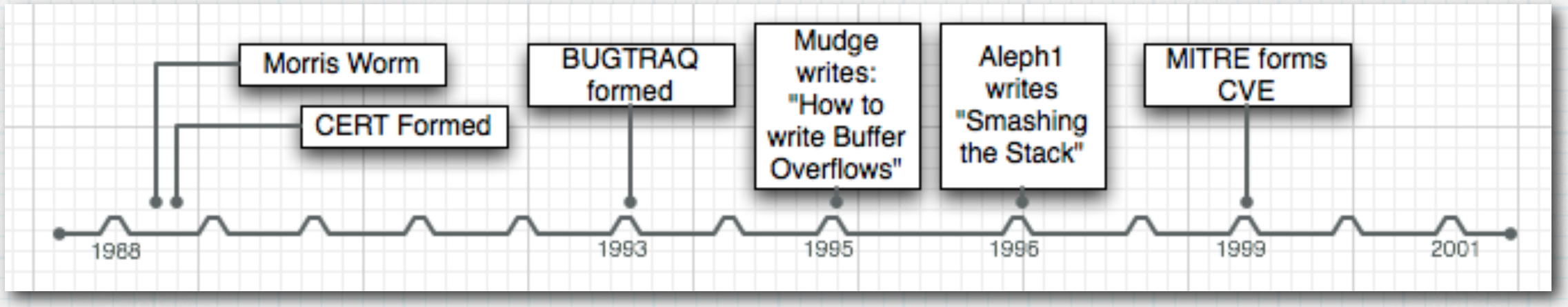
VS.

thinkst
applied research



Disclosed Bugs

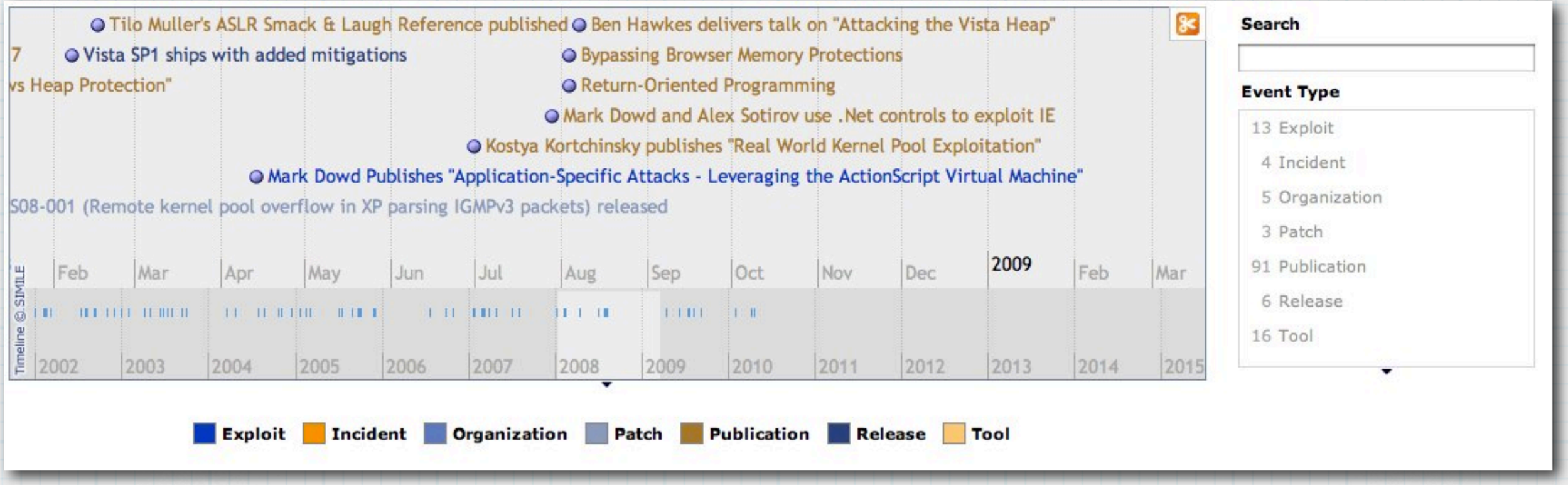
Our Approach



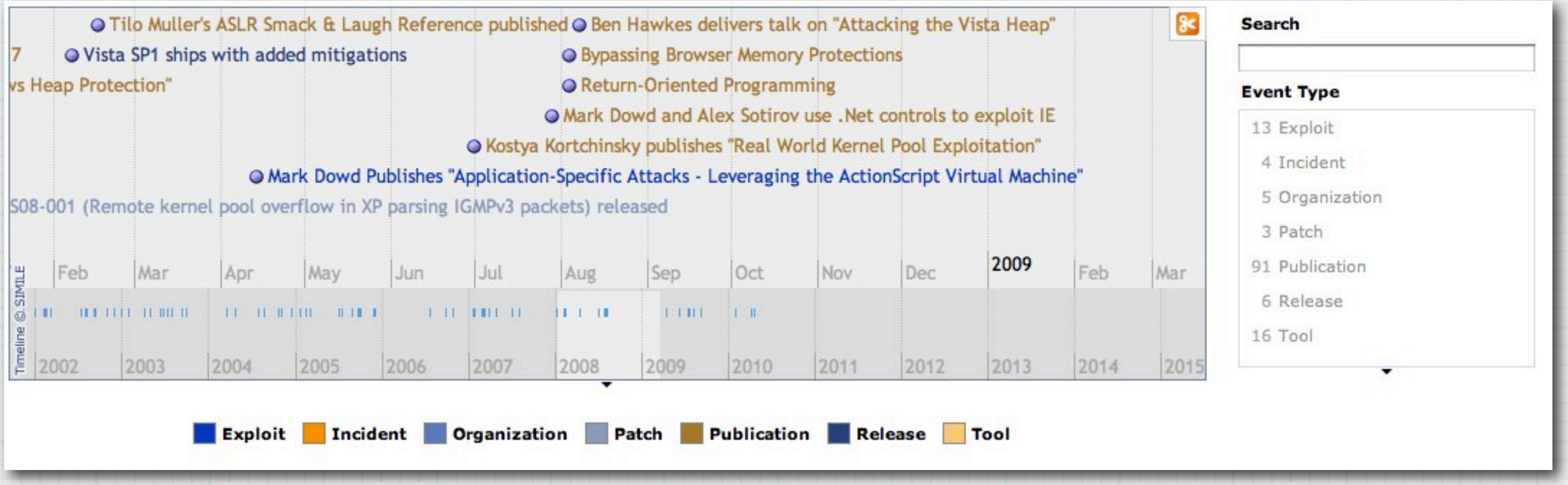
Clearly naive initially

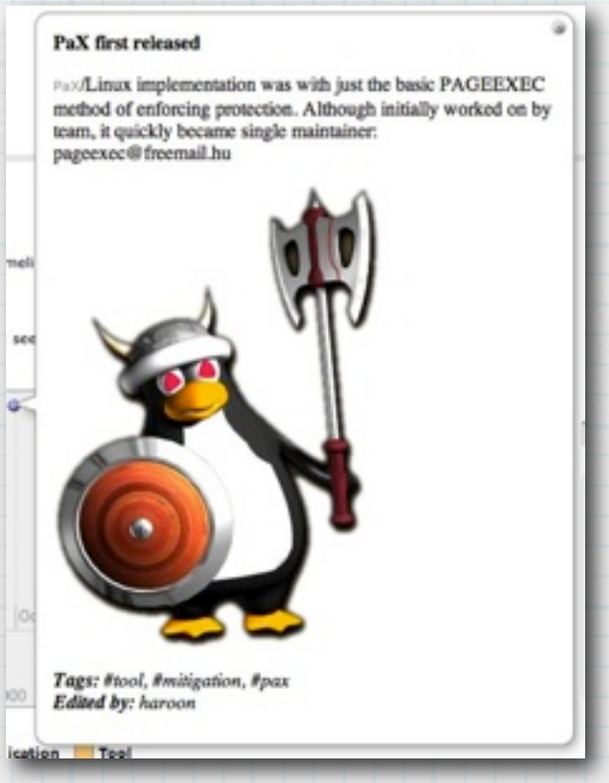
thinkst
applied research

http://ilm.thinkst.com/folklore/



http://ilm.thinkst.com/folklore/



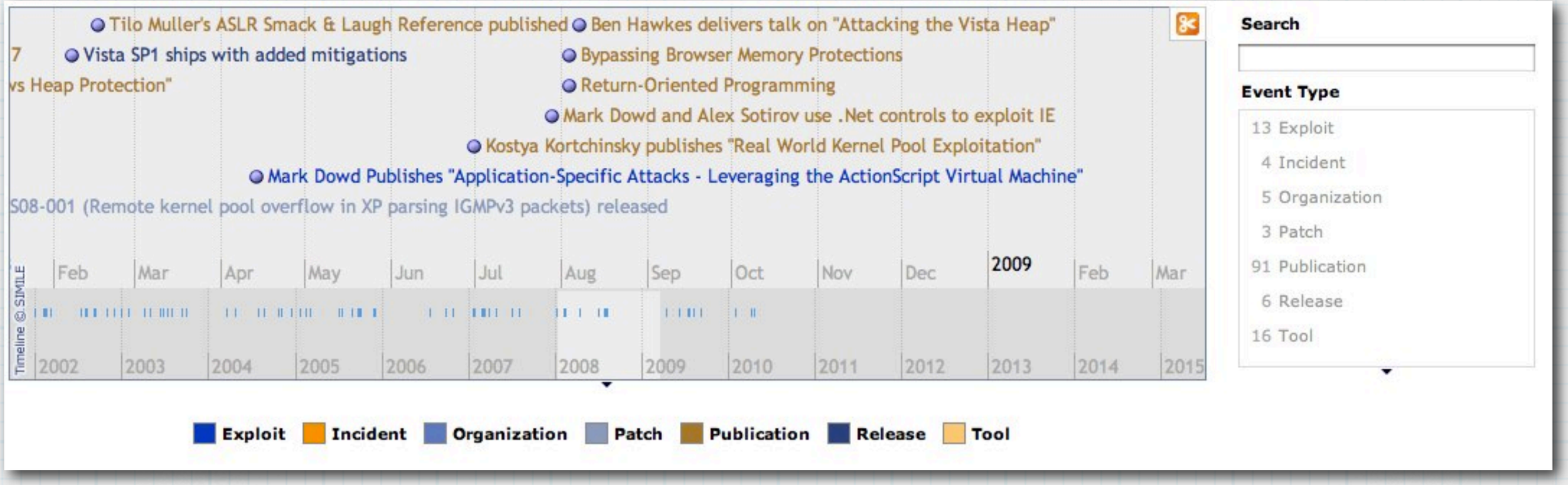


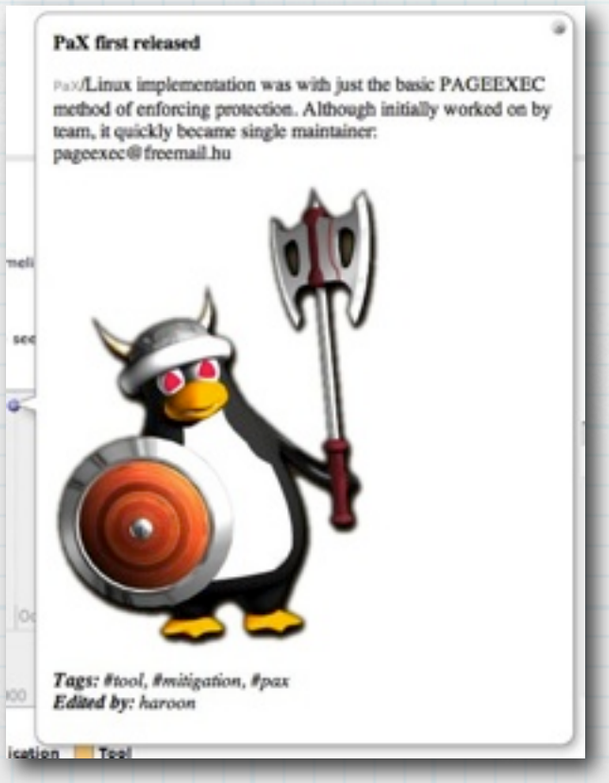
PaX first released

PaX/Linux implementation was with just the basic PAGEEXEC method of enforcing protection. Although initially worked on by team, it quickly became single maintainer: pageexec@freemail.hu

Tags: *#tool, #mitigation, #pax*
Edited by: *haroon*

http://ilm.thinkst.com/folklore/





the paper

(read it)

thinkst

10/20/1995 - **"How to Write Buffer Overflows"**
Although only a private / internal release, Mudge (mudge@l0pht.com) published his document titled "How to write Buffer Overflows".[6]

Written primarily as a set of notes to himself, the document covers both the basics of the overflow and a rough introduction to writing shellcode. The document included an exploit for the syslogd [7] bug made public earlier.

Figure 10
Peiter Zatko (Mudge)

12/3/1995 - **splitvt exploit published**
DaveG and VicM of Avalon Research published an advisory (and explcit) for splitvt on Linux 2-3.x. The vulnerability was due to an unbounded sprintf call, which was exploited by an over long HOME environment variable.

11/8/1996 - **Smashing the Stack Published**

Figure 11
Elias Levy (Aleph One)

Aleph1 published what would become the most referenced paper on memory corruption attacks in Phrack49. [8]
From his introduction:
`smash the stack` [C programming] n. On many C implementations it is possible to corrupt the execution stack by writing past the end of an array declared auto in a routine. Code that does this is said to smash the stack, and can cause return from the routine to jump to a random address. This can produce some of the most insidious data-dependent bugs known to mankind.

1/20/97
Stack Smashing Defenses Discussed
Bugtraq hosts a discussion on defenses against stack smashing

3/21/1997
Superprobe Exploit Published
solar designer overwrites function pointers to hijack execution flow

4/22/1997
DNS Poisoning QID Prediction
CORE and SNI report possible overflows due to bind ignoring MAXHOSTNAMELEN

Figure 12 - Alexander Peslyak (Solar Designer)

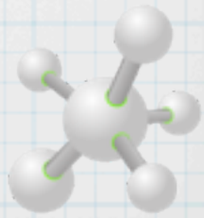
Bypassing the non-exec Stack (ret-2-libc) - 8/10/1997
Solar Designer published the first known return-to-libc attack to overcome his own non-executable stack patch [9].
He demonstrated the technique using the lpr exploit against a Linux system with his non executable stack patch. His patch (for his patch) ensured that shared libraries are mmaped into regions containing a null byte to retard their use with unsafe string functions.

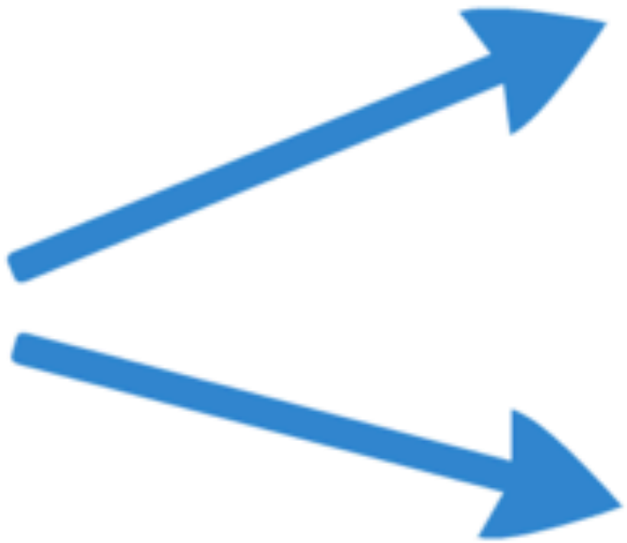
So at the end of this..

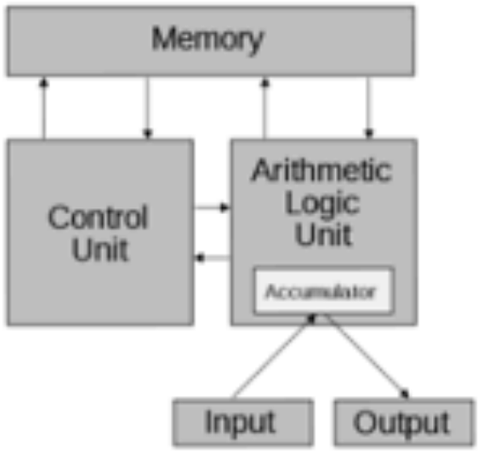
- You wont be able to suddenly use free() to obtain a 4-byte write anything anywhere primitive.
- You will understand what that means.
- You will be able to see:
  - When that was first used;
  - What prevents it's use/abuse today;

Where did it start?

thinkst
applied research






Memory
Control Unit
Arithmetic Logic Unit
Accumulator
Input
Output


ALU
Instructions memory
Control unit
Data memory
I/O

Memory Basics

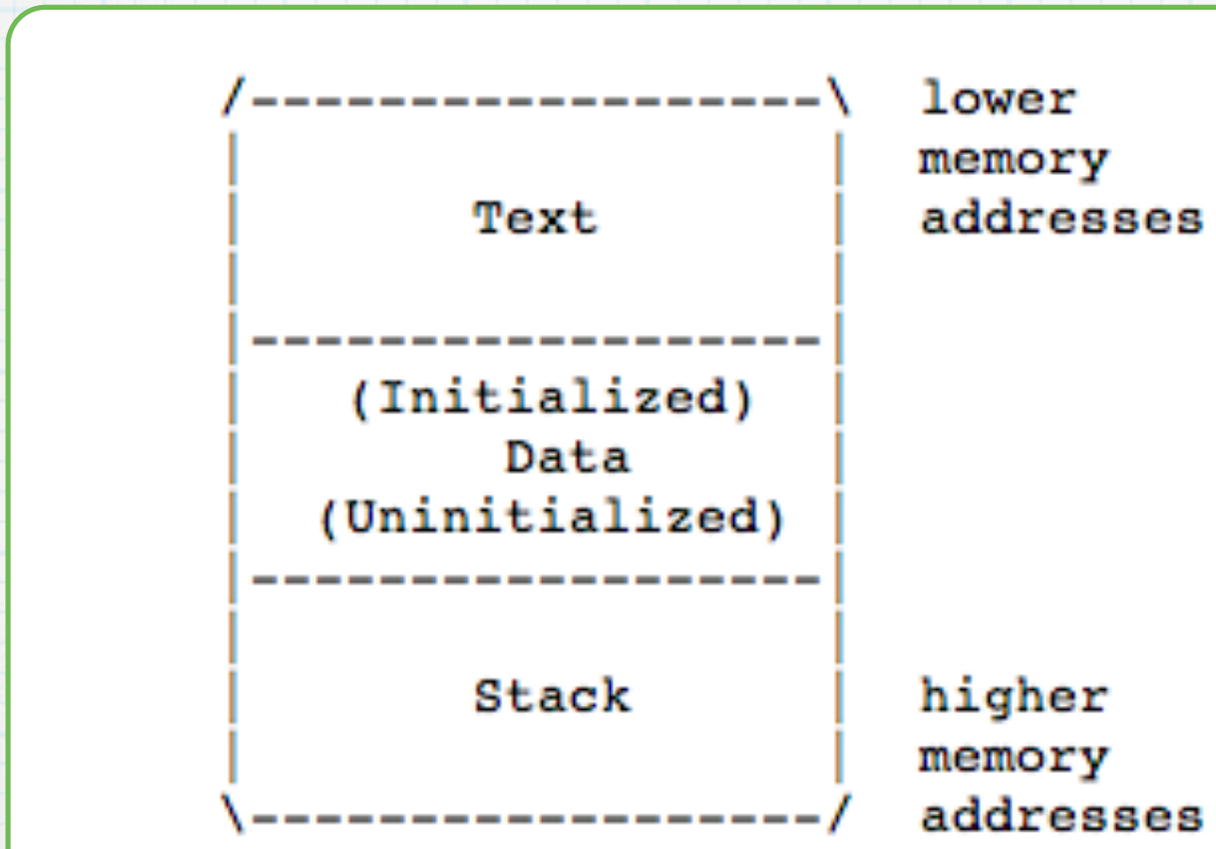
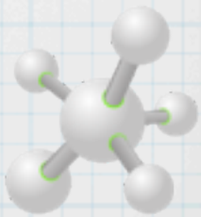
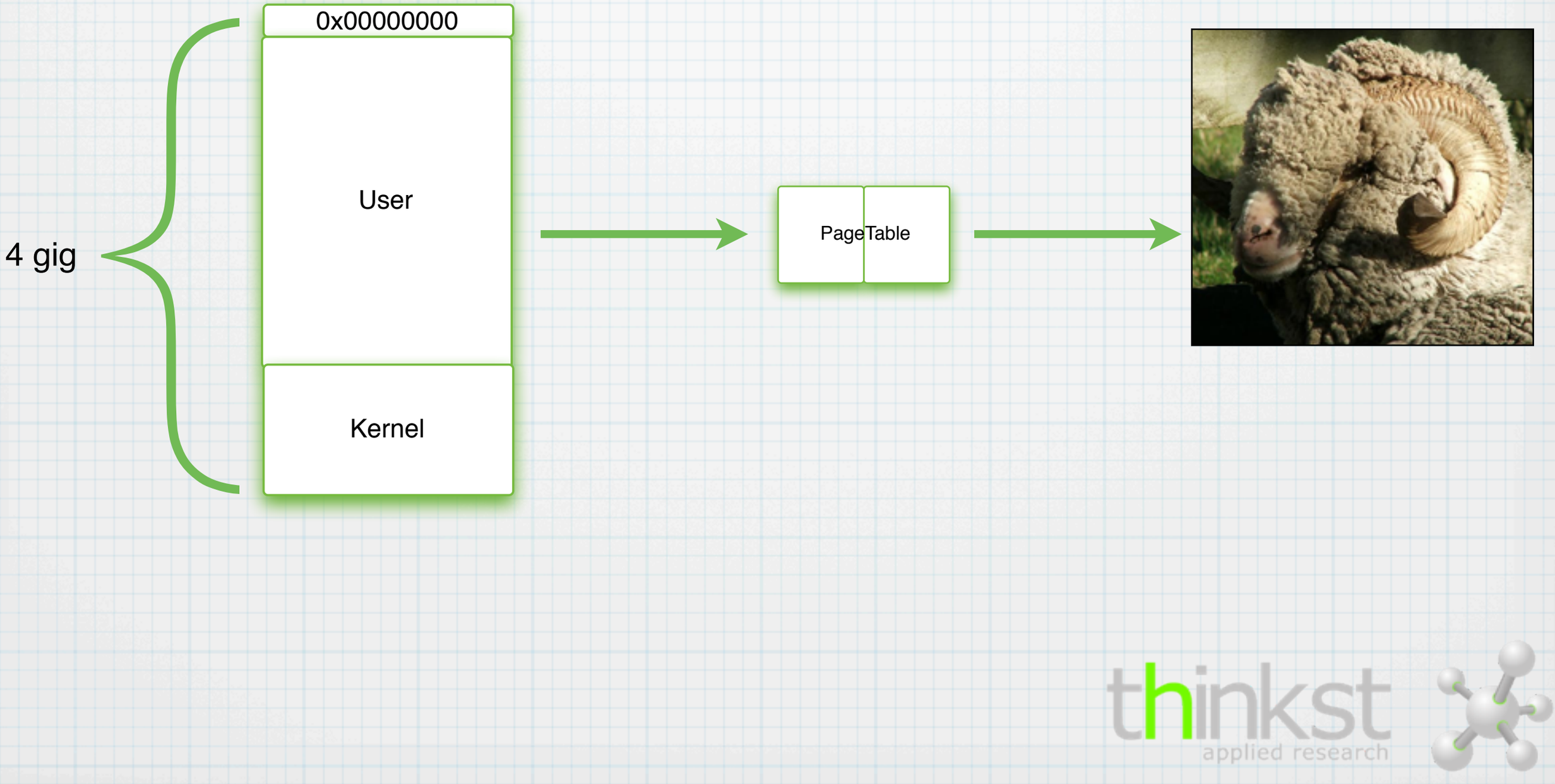


Fig. 1 Process Memory Regions

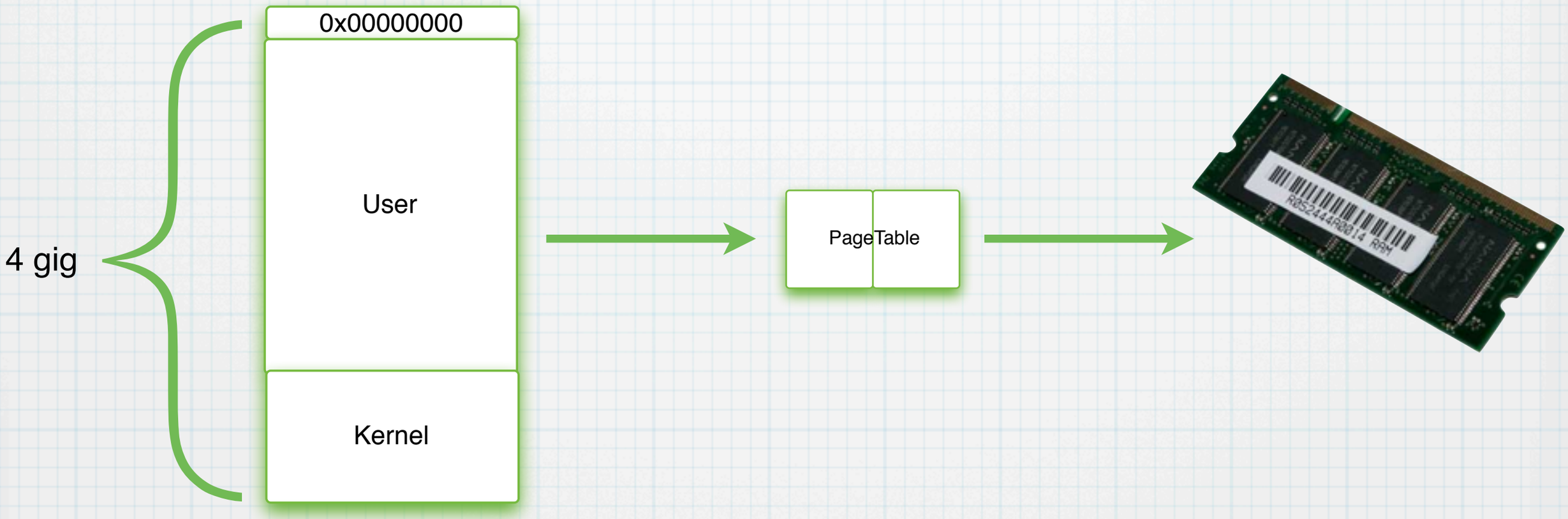
Memory Basics



Memory Basics



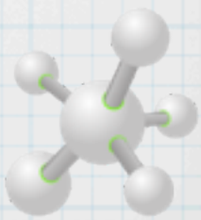
Memory Basics



Multiple Processes

3 gig

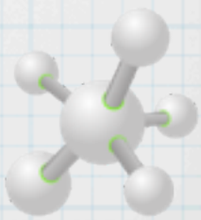
0x00000000	0x00000000	0x00000000
User	User	User
Kernel	Kernel	Kernel



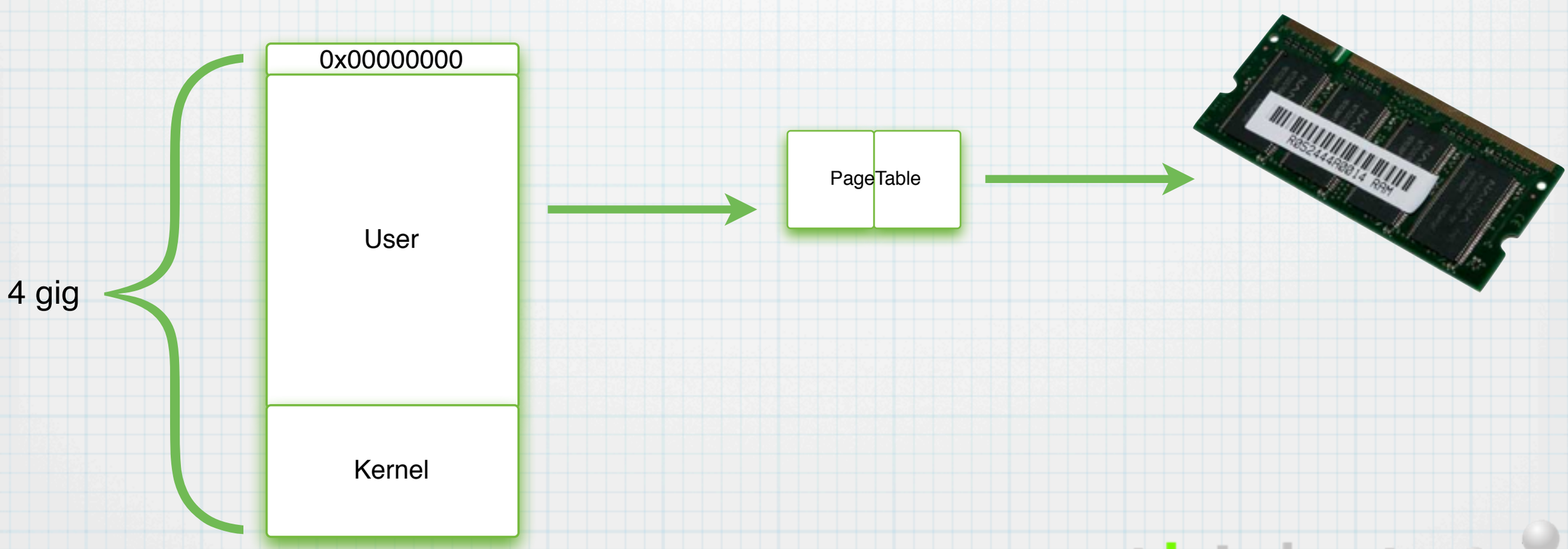
Multiple Processes

3 gig

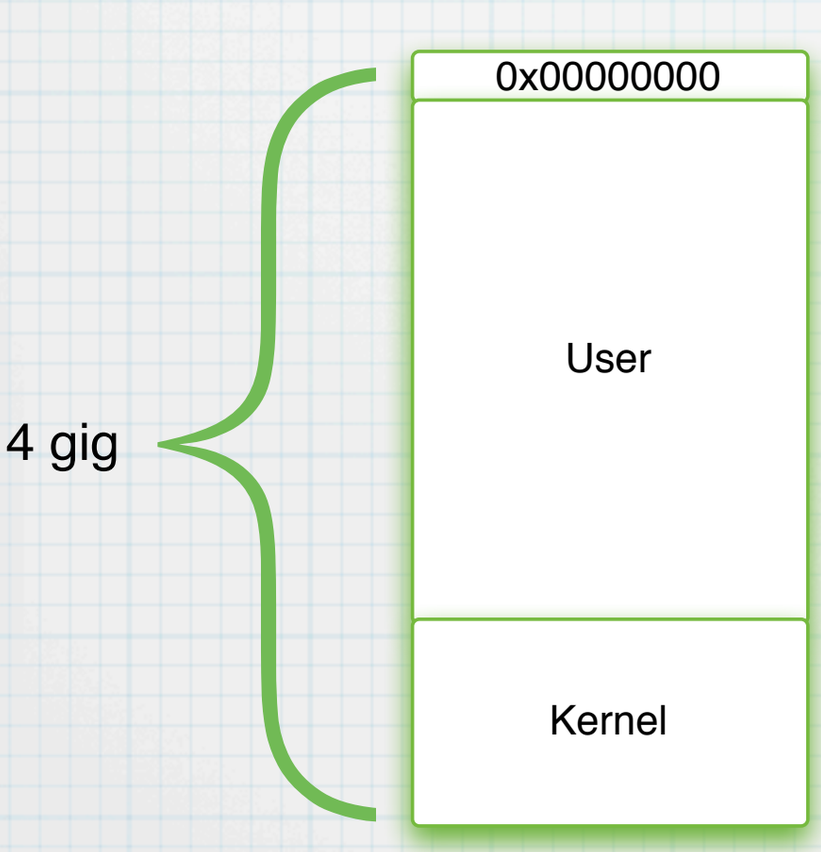
0x00000000	0x00000000	0x00000000
User	User	User
Kernel		



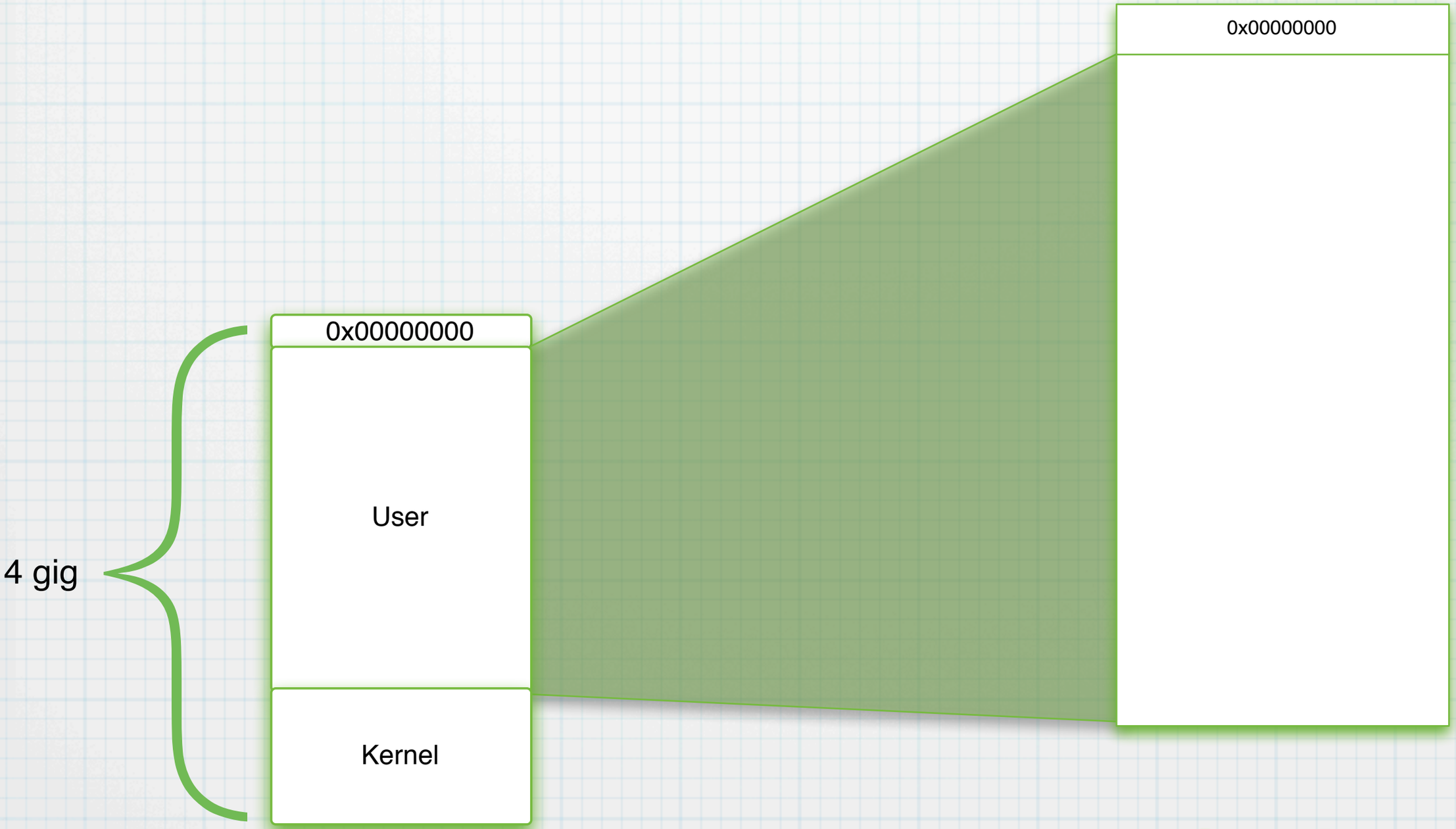
Segments



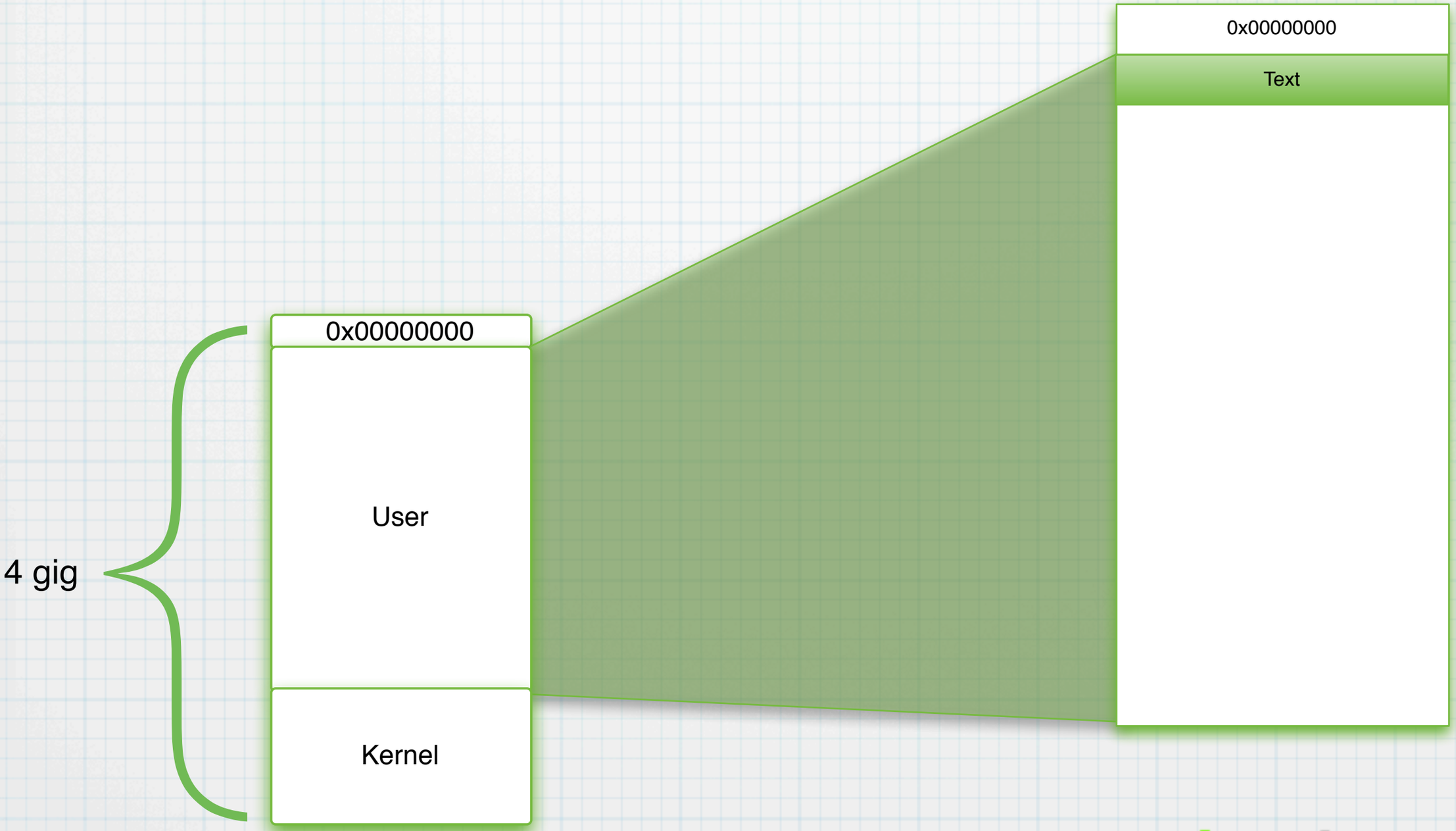
Segments



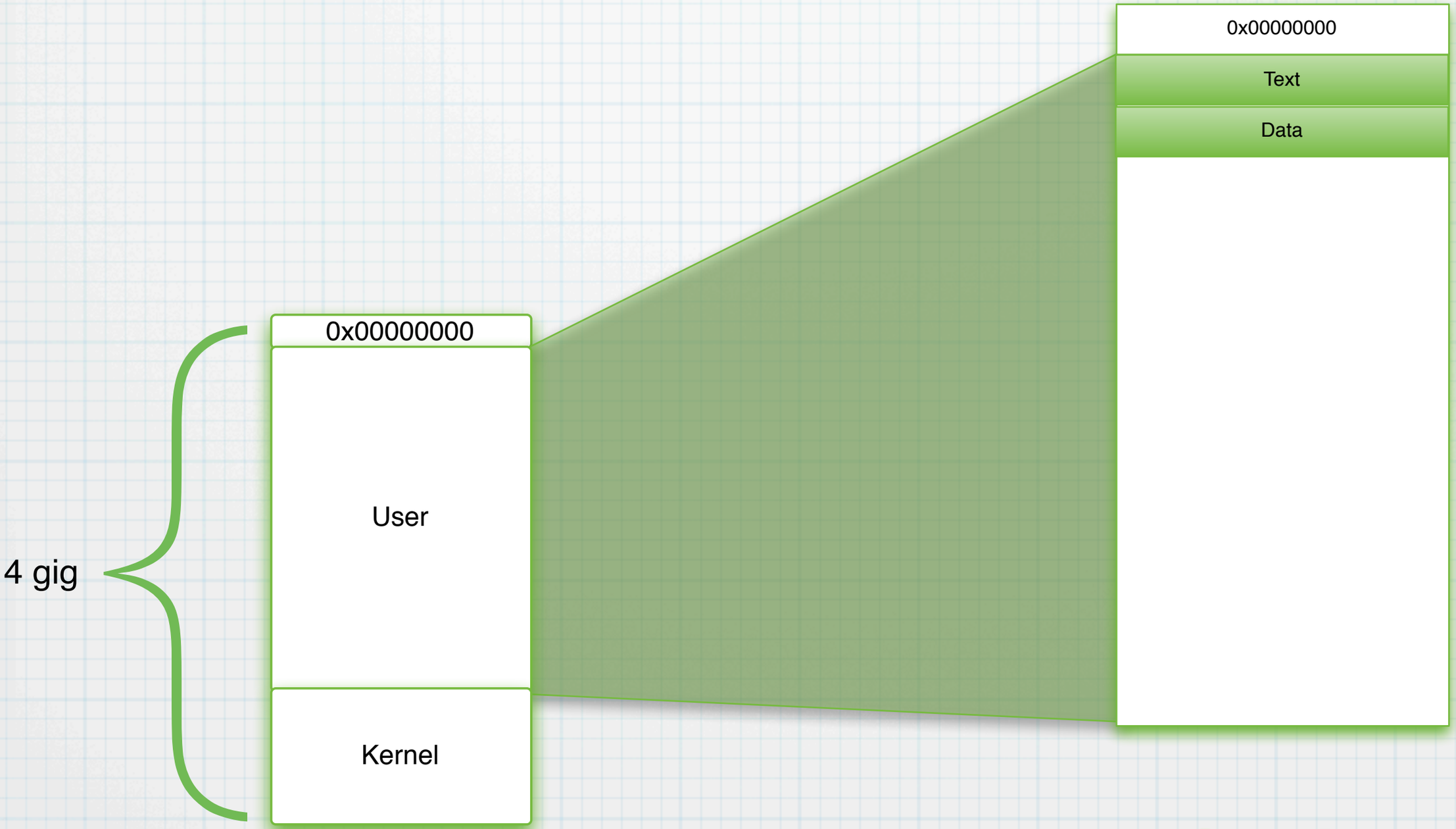
Segments



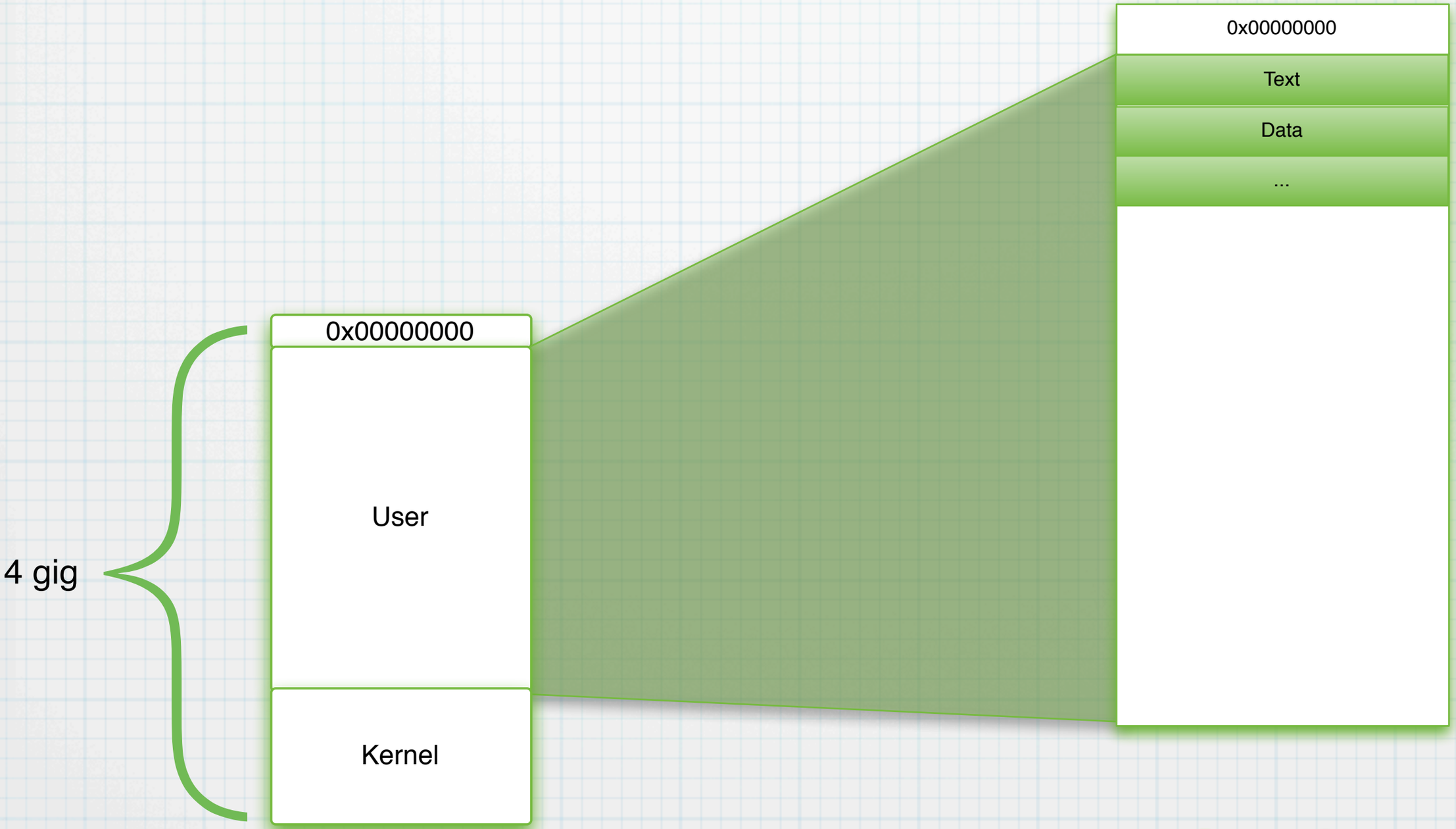
Segments



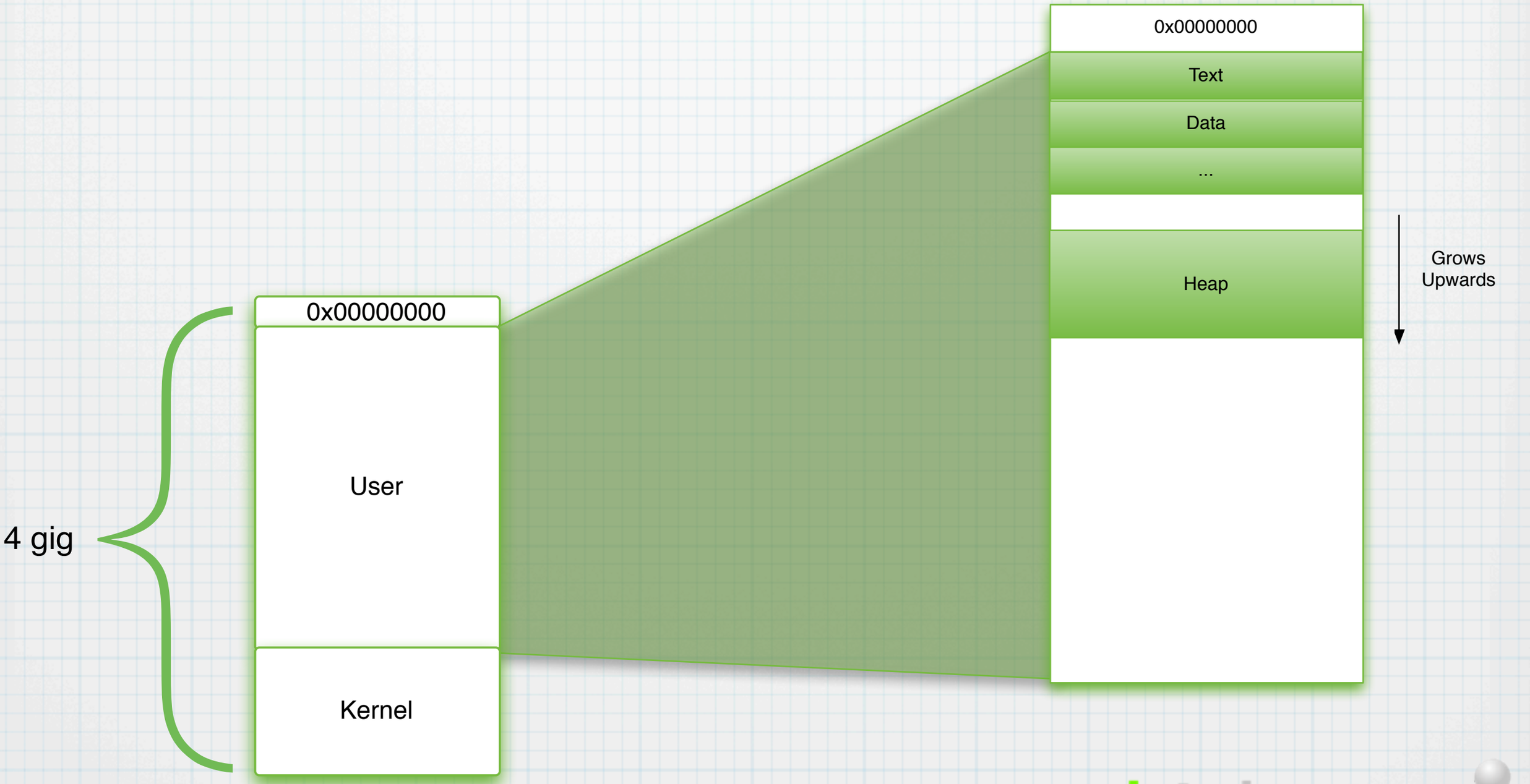
Segments



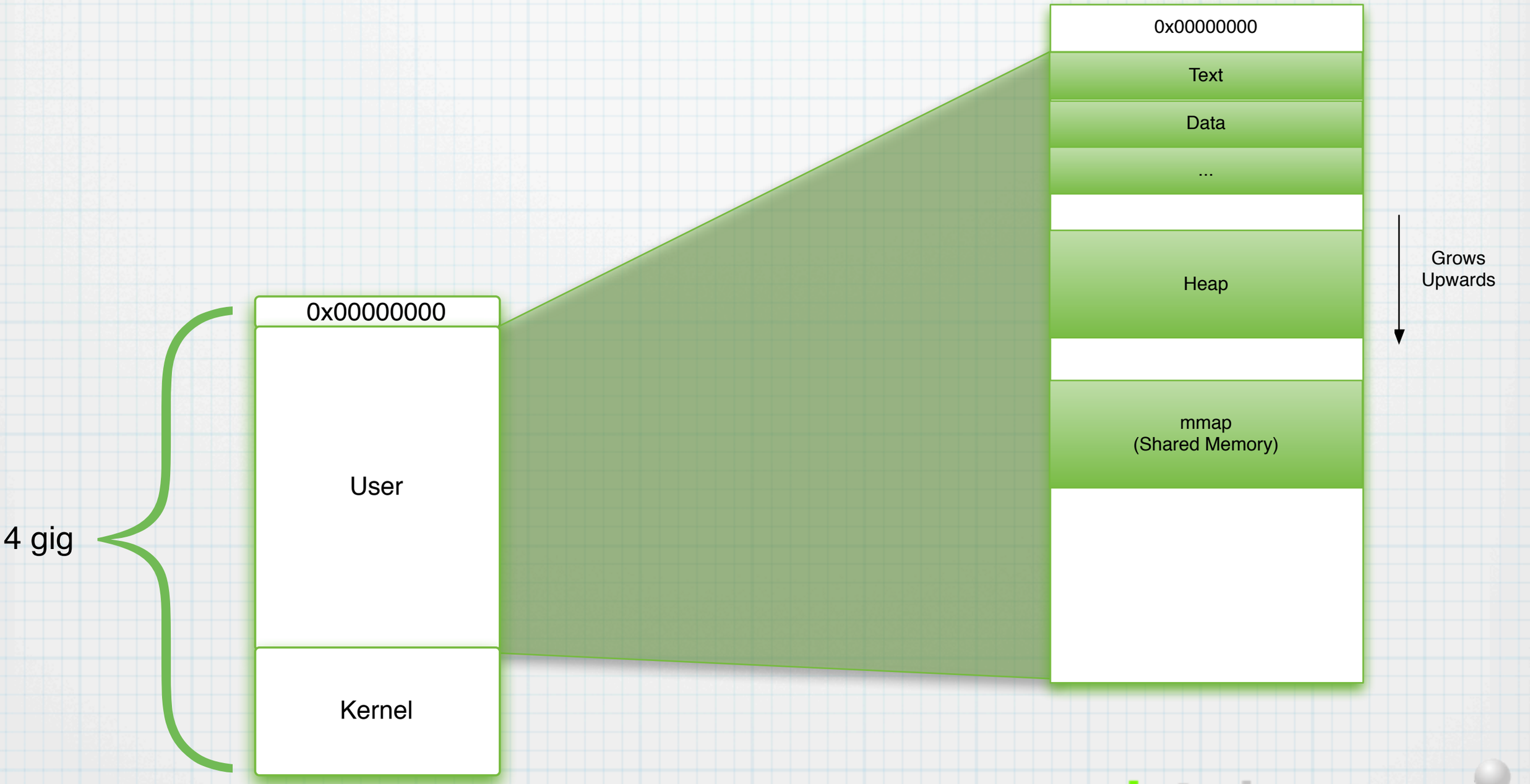
Segments



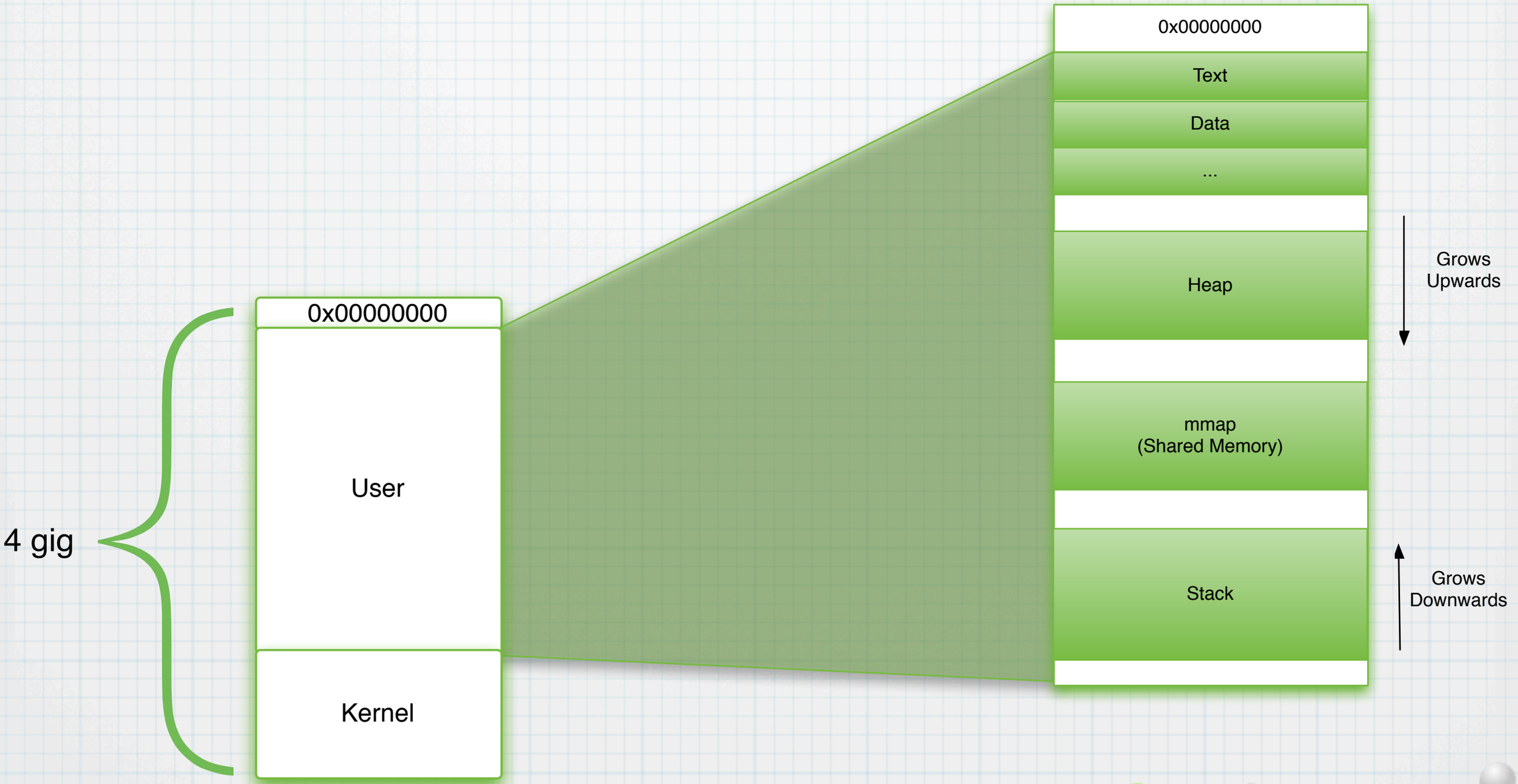
Segments



Segments



Segments



So what is code?

Address	Hex dump	ASCII
0040678F	35 80 2A 41 00 A3 7C 2A 41 00 E8 30 FB FF FF 83	5Ç*A.ú¦*A.₴0ƒ â
0040679F	C4 10 A3 80 2A 41 00 E8 9F B7 FF FF 85 C0 74 65	—▶úÇ*A.₴ƒπ à└te
004067AF	68 79 65 40 00 FF 35 74 2A 41 00 E8 86 FB FF FF	hye@. 5t*A.₴åƒ
004067BF	59 FF D0 83 F8 FF A3 A8 16 41 00 74 48 68 14 02	Y ╨â° úé▂A.tHh¶☻
004067CF	00 00 6A 01 E8 95 F2 FF FF 8B F0 85 F6 59 59 74	..j☺₴ò≥ ï≡à÷YYt
004067DF	34 56 FF 35 A8 16 41 00 FF 35 7C 2A 41 00 E8 53	4V 5é▂A. 5¦*A.₴S
004067EF	FB FF FF 59 FF D0 85 C0 74 1B 6A 00 56 E8 2A FC	ƒ Y ╨à└t←j.V₴*ⁿ
004067FF	FF FF 59 59 FF 15 B0 E0 40 00 83 4E 04 FF 89 06	YY ◙▒∝@.âN♦ ë♠
0040680F	33 C0 40 EB 07 E8 D5 FB FF FF 33 C0 5E 5F C3 55	3└@ş.₴ƒƒ 3└^_├U
0040681F	8B EC 83 EC 10 A1 D8 15 41 00 83 65 F8 00 83 65	ï∞â∞▶í╪◙A.âe°.âe
0040682F	FC 00 53 57 BF 4E E6 40 BB 3B C7 BB 00 00 FF FF	ⁿ.SWΓNμ@╗;╟╗..
0040683F	74 0D 85 C3 74 09 F7 D0 A3 DC 15 41 00 EB 60 56	t.à├t.◙╨ú▄◙A.ŝ'V
0040684F	8D 45 F8 50 FF 15 C4 E0 40 00 8B 75 FC 33 75 F8	ìE°P ◙—∝@.ïuⁿ3u°
0040685F	FF 15 C0 E0 40 00 33 F0 FF 15 B0 E0 40 00 33 F0	◙└∝@.3≡ ◙▒∝@.3≡
0040686F	FF 15 BC E0 40 00 33 F0 8D 45 F0 50 FF 15 B8 E0	◙╝∝@.3≡ìE≡P ◙╕∝
0040687F	40 00 8B 45 F4 33 45 F0 33 F0 3B F7 75 07 BE 4F	@.ïE⌠3E≡3≡;◙u·╛O
0040688F	E6 40 BB EB 0B 85 F3 75 07 8B C6 C1 E0 10 0B F0	μ@╗ŝ∂à≤u·ï├┴∝▶∂≡
0040689F	89 35 D8 15 41 00 F7 D6 89 35 DC 15 41 00 5E 5F	ë5╪◙A.◙╓ë5▄◙A.^_
004068AF	5B C9 C3 8B 44 24 04 A3 84 2A 41 00 C3 FF 74 24	[╔├ïD$♦ú䌆A.├ t$
004068BF	04 FF 15 C8 E0 40 00 33 C0 40 C2 08 00 6A 14 68	♦ ◙╚∝@.3└@┬◘.j¶h
004068CF	30 F8 40 00 E8 18 C3 FF FF 33 FF 89 7D E4 FF 35	0°@.₴↑├ 3 ë}Σ 5

So what is code?

Address	Hex dump	ASCII
0040678F	35 80 2A 41 00 A3 7C 2A 41 00 E8 30 FB FF FF 83	5Ç*A.ú¦*A.₴0ƒ â
0040679F	C4 10 A3 80 2A 41 00 E8 9F B7 FF FF 85 C0 74 65	—▶úÇ*A.₴ƒπ à└te
004067AF	68 79 65 40 00 FF 35 74 2A 41 00 E8 86 FB FF FF	hye@. 5t*A.₴åƒ
004067BF	59 FF D0 83 F8 FF A3 A8 16 41 00 74 48 68 14 02	Y ╨â° úé▂A.tHh¶☻
004067CF	00 00 6A 01 E8 95 F2 FF FF 8B F0 85 F6 59 59 74	..j☺₴ò≥ ï≡à÷YYt
004067DF	34 56 FF 35 A8 16 41 00 FF 35 7C 2A 41 00 E8 53	4V 5é▂A. 5¦*A.₴S
004067EF	FB FF FF 59 FF D0 85 C0 74 1B 6A 00 56 E8 2A FC	ƒ Y ╨à└t←j.V₴*ⁿ
004067FF	FF FF 59 59 FF 15 B0 E0 40 00 83 4E 04 FF 89 06	YY ◙▒∝@.âN♦ ë♠
0040680F	33 C0 40 EB 07 E8 D5 FB FF FF 33 C0 5E 5F C3 55	3└@♂.₴ƒƒ 3└^_├U
0040681F	8B EC 83 EC 10 A1 D8 15 41 00 83 65 F8 00 83 65	ï∞â∞▶í╪§A.âe°.âe
0040682F	FC 00 53 57 BF 4E E6 40 BB 3B C7 BB 00 00 FF FF	ⁿ.SWΓNµ@╗;╟╗..
0040683F	74 0D 85 C3 74 09 F7 D0 A3 DC 15 41 00 EB 60 56	t.à├t.≈╨ú▄§A.♂'V
0040684F	8D 45 F8 50 FF 15 C4 E0 40 00 8B 75 FC 33 75 F8	ìE°P ◙—∝@.ïuⁿ3u°
0040685F	FF 15 C0 E0 40 00 33 F0 FF 15 B0 E0 40 00 33 F0	◙└∝@.3≡ ◙▒∝@.3≡
0040686F	FF 15 BC E0 40 00 33 F0 8D 45 F0 50 FF 15 B8 E0	◙╝∝@.3≡ìE≡P ◙╕∝
0040687F	40 00 8B 45 F4 33 45 F0 33 F0 3B F7 75 07 BE 4F	@.ïE⌠3E≡3≡;≈u•╛O
0040688F	E6 40 BB EB 0B 85 F3 75 07 8B C6 C1 E0 10 0B F0	µ@╗♂∂à≤u•ï╞┴∝▶∂≡
0040689F	89 35 D8 15 41 00 F7 D6 89 35 DC 15 41 00 5E 5F	ë5╪§A.≈╓ë5▄§A.^_
004068AF	5B C9 C3 8B 44 24 04 A3 84 2A 41 00 C3 FF 74 24	[╔├ïD$♦ü䪀A.├ t$
004068BF	04 FF 15 C8 E0 40 00 33 C0 40 C2 08 00 6A 14 68	♦ ◙╚∝@.3└@┬◘.j¶h
004068CF	30 F8 40 00 E8 18 C3 FF FF 33 FF 89 7D E4 FF 35	0°@.₴↑├ 3 ë}Σ 5

So what is code?

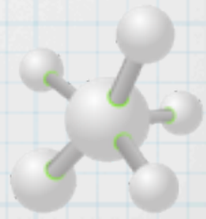
Address	Hex dump	ASCII
0040678F	35 80 2A 41 00 A3 7C 2A 41 00 E8 30 FB FF FF 83	5Ç*A.ú¦*A.Φ0ƒ â
0040679F	C4 10 A3 80 2A 41 00 E8 9F B7 FF FF 85 C0 74 65	−▶üÇ*A.ΦƒŋÿÿàÀte
004067AF	68 79 65 40 00 FF 35 74 2A 41 00 E8 86 FB FF FF	hye@. 5t*A.Φåƒ
004067BF	59 FF D0 83 F8 FF A3 A8 16 41 00 74 48 68 14 02	Y ╨â° úé▄A.tHh¶☻
004067CF	00 00 6A 01 E8 95 F2 FF FF 8B F0 85 F6 59 59 74	..j☺Φò≥ ï≡à÷YYt
004067DF	34 56 FF 35 A8 16 41 00 FF 35 7C 2A 41 00 E8 53	4V 5é▄A. 5¦*A.ΦS
004067EF	FB FF FF 59 FF D0 85 C0 74 1B 6A 00 56 E8 2A FC	ƒ Y ╨àÀt←j.VΦ*ⁿ
004067FF	FF FF 59 59 FF 15 B0 E0 40 00 83 4E 04 FF 89 06	YY ▒▒∝@.âN♦ ë♠
0040680F	33 C0 40 EB 07 E8 D5 FB FF FF 33 C0 5E 5F C3 55	3└@δ.Φƒƒ 3└^_├U
0040681F	8B EC 83 EC 10 A1 D8 15 41 00 83 65 F8 00 83 65	ï∞â∞▶ï╪ΦA.âe°.âe
0040682F	FC 00 53 57 BF 4E E6 40 BB 3B C7 BB 00 00 FF FF	ⁿ.SWΓNµ@╗;╟╗..
0040683F	74 0D 85 C3 74 09 F7 D0 A3 DC 15 41 00 EB 60 56	t.à├t.ª╨ú▄ΦA.δ'V
0040684F	8D 45 F8 50 FF 15 C4 E0 40 00 8B 75 FC 33 75 F8	ìE°P ▒—∝@.ïuⁿ3u°
0040685F	FF 15 C0 E0 40 00 33 F0 FF 15 B0 E0 40 00 33 F0	▒└∝@.3≡ ▒▒∝@.3≡
0040686F	FF 15 BC E0 40 00 33 F0 8D ‘00406814	> E8 D5FBFFFF	CALL 004063EE
0040687F	40 00 8B 45 F4 33 45 F0 33	00406819	> 33C0	XOR EAX,EAX
0040688F	E6 40 BB EB 0B 85 F3 75 07	0040681B	> 5E	POP ESI
0040689F	89 35 D8 15 41 00 F7 D6 89	0040681C	• 5F	POP EDI
004068AF	5B C9 C3 8B 44 24 04 A3 84	0040681D	•└ C3	RETN
004068BF	04 FF 15 C8 E0 40 00 33 C0	0040681E	$┌ 55	PUSH EBP
004068CF	30 F8 40 00 E8 18 C3 FF FF	0040681F	•│ 8BEC	MOV EBP,ESP
		00406821	•│ 83EC 10	SUB ESP,10
		00406824	│ A1 D8154100	MOV EAX,DWORD PTR DS:[4115D8]
		00406829	│ 8365 F8 00	AND DWORD PTR SS:[EBP-8],00000000
		0040682D	│ 8365 FC 00	AND DWORD PTR SS:[EBP-4],00000000
		00406831	│ 53	PUSH EBX
		00406832	│ 57	PUSH EDI
		00406833	│ BF 4EE640BB	MOV EDI,BB40E64E
		00406838	│• 3BC7	CMP EAX,EDI
		0040683A	│• BB 0000FFFF	MOV EBX,FFFF0000
		0040683F	│•v 74 0D	JE SHORT 0040684E
		00406841	│• 85C3	TEST EBX,EAX
		00406843	│•v 74 09	JE SHORT 0040684E
		00406845	│• F7D0	NOT EAX

thinkst applied research

So what is code?

```
Address   Hex dump                                          ASCII
0040678F  35 80 2A 41  00 A3 7C 2A  41 00 E8 30  FB FF FF 83  5Ç*A.ú|*A.Ø0√  â
0040679F  C4 10 A3 80  2A 41 00 E8  9F B7 FF FF  85 C0 74 65  ─▶úÇ*A.Ø ƒπ   à└te
004067AF  68 79 65 40  00 FF 35 74  2A 41 00 E8  86 FB FF FF  hye@. 5t*A.Ø å√
004067BF  59 FF D0 83  F8 FF A3 A8  16 41 00 74  48 68 14 02  Y ╨â°  úí▄A.tHh¶☻
004067CF  00 00 6A 01  E8 95 F2 FF  FF 8B F0 85  F6 59 59 74  ..j☺Ø≥≥  ï≡à÷YYt
004067DF  34 56 FF 35  A8 16 41 00  FF 35 7C 2A  41 00 E8 53  4V 5¿▄A. 5|*A.ØS
004067EF  FB FF FF 59  FF D0 85 C0  74 1B 6A 00  56 E8 2A FC  √  Y ╨à└t←j.VØ*ⁿ
004067FF  FF FF 59 59  FF 15 B0 E0  40 00 83 4E  04 FF 89 06    YY ≥▒∝@.âN♦ ë♠
0040680F  33 C0 40 EB  07 E8 D5 FB  FF FF 33 C0  5E 5F C3 55  3└@ŝ.Ø ƒ√  3└^_├U
0040681F  8B EC 83 EC  10 A1 D8 15  41 00 83 65  F8 00 83 65  ï∞â∞▶í╪§A.âe°.âe
0040682F  FC 00 53 57  BF 4E E6 40  BB 3B C7 BB  00 00 FF FF  ⁿ.SWⁿNµ@╗;╟╗..
0040683F  74 0D 85 C3  74 09 F7 D0  A3 DC 15 41  00 EB 60 56  t.à├t.≈╨ú▄§A.ŝ'V
0040684F  8D 45 F8 50  FF 15 C4 E0  40 00 8B 75  FC 33 75 F8  ìE°P ≥─∝@.ïuⁿ3u°
0040685F  FF 15 C0 E0  40 00 33 F0  FF 15 B0 E0  40 00 33 F0   ≥└∝@.3≡ ≥▒∝@.3≡
0040686F  FF 15 BC E0  40 00 33 F0  8D ‘00406814  > E8 D5FBFFFF  CALL 004063EE
0040687F  40 00 8B 45  F4 33 45 F0  33 00406819  > 33C0          XOR EAX,EAX
0040688F  E6 40 BB EB  0B 85 F3 75  07 0040681B  > 5E            POP ESI
0040689F  89 35 D8 15  41 00 F7 D6  89 0040681C  . 5F            POP EDI
004068AF  5B C9 C3 8B  44 24 04 A3  84 0040681D  . C3            RETN
004068BF  04 FF 15 C8  E0 40 00 33  C0 0040681E  $ 55            PUSH EBP
004068CF  30 F8 40 00  E8 18 C3 FF  FF 0040681F  . 8BEC          MOV EBP,ESP
                                    00406821  . 83EC 10       SUB ESP,10
                                    00406824    A1 D8154100  MOV EAX,DWORD PTR DS:[4115D8]
                                    00406829    8365 F8 00   AND DWORD PTR SS:[EBP-8],00000000
                                    0040682D    8365 FC 00   AND DWORD PTR SS:[EBP-4],00000000
                                    00406831    53           PUSH EBX
                                    00406832    57           PUSH EDI
                                    00406833    BF 4EE640BB  MOV EDI,BB40E64E
                                    00406838  . 3BC7          CMP EAX,EDI
                                    0040683A  . BB 0000FFFF   MOV EBX,FFFF0000
                                    0040683F  .v 74 0D        JE SHORT 0040684E
                                    00406841  . 85C3          TEST EBX,EAX
                                    00406843  .v 74 09        JE SHORT 0040684E
                                    00406845  . F7D0          NOT EAX
```

thinkst
applied research



Is this code?

```
0040681D  L.   C3              RETN
0040681E   $   55              PUSH EBP
0040681F   .   57              PUSH EDI
00406820   .   68 61742069     PUSH 69207461
00406825   .v  73 20           JNB SHORT 00406847
00406827   .v  74 68           JE SHORT 00406891
00406829   .   6973 F8 0083(   IMUL ESI,DWORD PTR DS:[EBX-8],-39A7D00
00406830   .   0053 57         ADD BYTE PTR DS:[EBX+57],DL
00406833   .   BF 4EE640BB     MOV EDI,BB40E64E
00406838   .   3BC7            CMP EAX,EDI
0040683A   .   BB 0000FFFF     MOV EBX,FFFF0000
0040683F   .v  74 0D           JE SHORT 0040684E
00406841   .   85C3            TEST EBX,EAX
00406843   .v  74 09           JE SHORT 0040684E
00406845  [.   F7D0            NOT EAX
00406847  [>   A3 DC154100     MOV DWORD PTR DS:[4115DC],EAX
0040684C  L.v  EB 60           JMP SHORT 004068AE
0040684E   >   56              PUSH ESI
0040684F   .   8D45 F8         LEA EAX,[EBP-8]
00406852   .   50              PUSH EAX
```

Is this code?

```
0040681D  L.   C3              RETN
0040681E   $   55              PUSH EBP
0040681F   .   57              PUSH EDI
00406820   .   68 61742069     PUSH 69207461
00406825   .v  73 20           JNB SHORT 00406847
00406827   .v  74 68           JE SHORT 00406891
00406829   .   6973 F8 0083   IMUL ESI,DWORD PTR DS:[EBX-8],-39A7D00
00406830   .   0053 57         ADD BYTE PTR DS:[EBX+57],DL
00406833   .   BF 4EE640BB     MOV EDI,BB40E64E
00406838   .   3BC7            CMP EAX,EDI
0040683A   .   BB 0000FFFF     MOV EBX,FFFF0000
0040683F   .v  74 0D           JE SHORT 0040684E
00406841   .   85C3            TEST EBX,EAX
00406843   .v  74 09           JE SHORT 0040684E
00406845  [.   F7D0            NOT EAX
00406847  [>   A3 DC154100     MOV DWORD PTR DS:[4115DC],EAX
0040684C  L.v  EB 60           JMP SHORT 004068AE
0040684E   >   56              PUSH ESI
0040684F   .   8D45 F8         LEA EAX,[EBP-8]
00406852   .   50              PUSH EAX
```

Is this code?

```
0040681D  L.   C3                RETN
0040681E   $   55                PUSH EBP
0040681F   .   57                PUSH EDI
00406820   .   68 61742069       PUSH 69207461
00406825   .v  73 20             JNB SHORT 00406847
00406827   .v  74 68             JE SHORT 00406891
00406829   .   6973 F8 0083      IMUL ESI,DWORD PTR DS:[EBX-8],-39A7D00
00406830   .   0053 57            ADD BYTE PTR DS:[EBX+57],DL
00406833   .   BF 4EE640BB       MOV EDI,BB40E64E
00406838   .   3BC7              CMP EAX,EDI
0040683A   .   BB 0000FFFF       MOV EBX,FFFF0000
0040683F   .v  74 0D             JE SHORT 0040684E
00406841   .   85C3              TEST EBX,EAX
00406843   .v  74 09             JE SHORT 0040684F
00406845   [.  F7D0              NOT EAX
00406847   [>  A3 DC154100       MOV DWC
0040684C   [.v EB 60             JMP SHC
0040684E    >  56                PUSH ES
0040684F    .  8D45 F8           LEA EAX
00406852    .  50                PUSH EA
```

Address	Hex dump
0040679F	C4 10 A3 80 2A 41 00 E8 9F B7 FF FF 85 C0 74 65
004067AF	68 79 65 40 00 FF 35 74 2A 41 00 E8 86 FB FF FF
004067BF	59 FF D0 83 F8 FF A3 A8 16 41 00 74 48 68 14 02
004067CF	00 00 6A 01 E8 95 F2 FF FF 8B F0 85 F6 59 59 74
004067DF	34 56 FF 35 A8 16 41 00 FF 35 7C 2A 41 00 E8 53
004067EF	FB FF FF 59 FF D0 85 C0 74 1B 6A 00 56 E8 2A FC
004067FF	FF FF 59 59 FF 15 B0 E0 40 00 83 4E 04 FF 89 06
0040680F	33 C0 40 EB 07 E8 D5 FB FF FF 33 C0 5E 5F C3 55
0040681F	57 68 61 74 20 69 73 20 74 68 69 73 F8 00 83 65
0040682F	FC 00 53 57 BF 4E E6 40 BB 3B C7 BB 00 00 FF FF
0040683F	74 0D 85 C3 74 09 F7 D0 A3 DC 15 41 00 EB 60 56
0040684F	8D 45 F8 50 FF 15 C4 E0 40 00 8B 75 FC 33 75 F8
0040685F	FF 15 C0 E0 40 00 33 F0 FF 15 B0 E0 40 00 33 F0
0040686F	FF 15 BC E0 40 00 33 F0 8D 45 F0 50 FF 15 B8 E0
0040687F	40 00 8B 45 F4 33 45 F0 33 F0 3B F7 75 07 BE 4F
0040688F	E6 40 BB EB 0B 85 F3 75 07 8B C6 C1 E0 10 0B F0
0040689F	89 35 D8 15 41 00 F7 D6 89 35 DC 15 41 00 5E 5F
004068AF	5B C9 C3 8B 44 24 04 A3 84 2A 41 00 C3 FF 74 24
004068BF	04 FF 15 C8 E0 40 00 33 C0 40 C2 08 00 6A 14 68
004068CF	30 F8 40 00 E8 18 C3 FF FF 33 FF 89 7D E4 FF 35
004068DF	84 2A 41 00 E8 5D FA FF FF 59 8B F0 3B F7 75 53

Is this code?

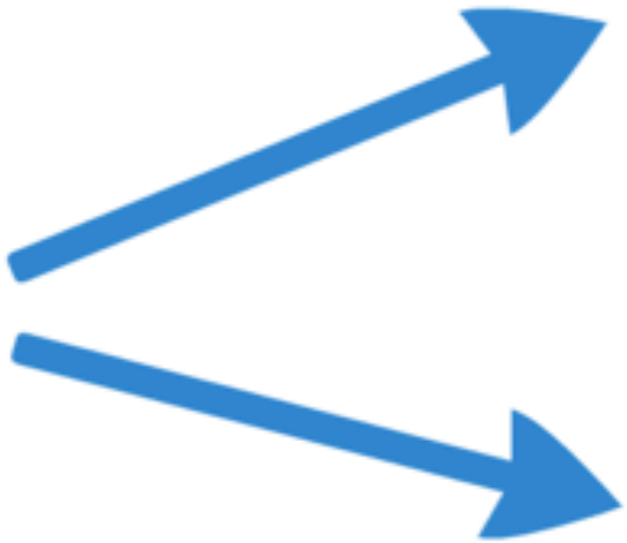
```
0040681D  C3              RETN
0040681E  $ 55            PUSH EBP
0040681F  . 57            PUSH EDI
00406820  . 68 61742069   PUSH 69207461
00406825  .v 73 20        JNB SHORT 00406847
00406827  .v 74 68        JE SHORT 00406891
00406829  . 6973 F8 0083  IMUL ESI,DWORD PTR DS:[EBX-8],-39A7D00
00406830  . 0053 57       ADD BYTE PTR DS:[EBX+57],DL
00406833  . BF 4EE640BB   MOV EDI,BB40E64E
00406838  . 3BC7          CMP EAX,EDI
0040683A  . BB 0000FFFF   MOV EBX,FFFF0000
0040683F  .v 74 0D        JE SHORT 0040684E
00406841  . 85C3          TEST EBX,EAX
00406843  .v 74 09        JE SHORT 0040684F
00406845  . F7D0          NOT EAX
00406847  > A3 DC154100   MOV DWO
0040684C  .v EB 60        JMP SHO
0040684E  > 56            PUSH ES
0040684F  . 8D45 F8       LEA EAX
00406852  . 50            PUSH EA
```

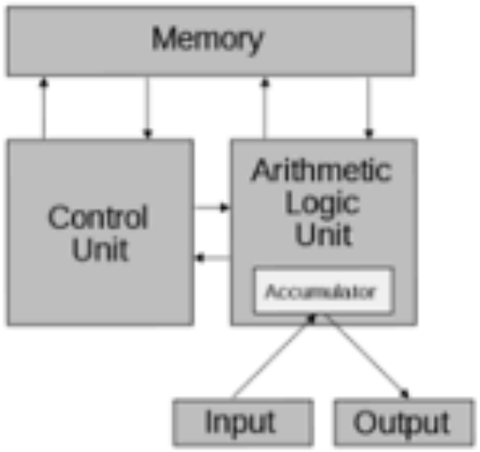
Address	Hex dump
0040679F	C4 10 A3 80 2A 41 00 E8 9F B7 FF FF 85 C0 74 65
004067AF	68 79 65 40 00 FF 35 74 2A 41 00 E8 86 FB FF FF
004067BF	59 FF D0 83 F8 FF A3 A8 16 41 00 74 48 68 14 02
004067CF	00 00 6A 01 E8 95 F2 FF FF 8B F0 85 F6 59 59 74
004067DF	34 56 FF 35 A8 16 41 00 FF 35 7C 2A 41 00 E8 53
004067EF	FB FF FF 59 FF D0 85 C0 74 1B 6A 00 56 E8 2A FC
004067FF	FF FF 59 59 FF 15 B0 E0 40 00 83 4E 04 FF 89 06
0040680F	33 C0 40 EB 07 E8 D5 FB FF FF 33 C0 5E 5F C3 55
0040681F	57 68 61 74 20 69 73 20 74 68 69 73 F8 00 83 65
0040682F	FC 00 53 57 BF 4E E6 40 BB 3B C7 BB 00 00 FF FF
0040683F	74 0D 85 C3 74 09 F7 D0 A3 DC 15 41 00 EB 60 56
0040684F	8D 45 F8 50 FF 15 C4 E0 40 00 8B 75 FC 33 75 F8
0040685F	FF 15 C0 E0 40 00 33 F0 FF 15 B0 E0 40 00 33 F0
0040686F	FF 15 BC E0 40 00 33 F0 8D 45 F0 50 FF 15 B8 E0
0040687F	40 00 8B 45 F4 33 45 F0 33 F0 3B F7 75 07 BE 4F
0040688F	E6 40 BB EB 0B 85 F3 75 07 8B C6 C1 E0 10 0B F0
0040689F	89 35 D8 15 41 00 F7 D6 89 35 DC 15 41 00 5E 5F
004068AF	5B C9 C3 8B 44 24 04 A3 84 2A 41 00 C3 FF 74 24
004068BF	04 FF 15 C8 E0 40 00 33 C0 40 C2 08 00 6A 14 68
004068CF	30 F8 40 00 E8 18 C3 FF FF 33 FF 89 7D E4 FF 35
004068DF	84 2A 41 00 E8 5D FA FF FF 59 8B F0 3B F7 75 53

Is this code?

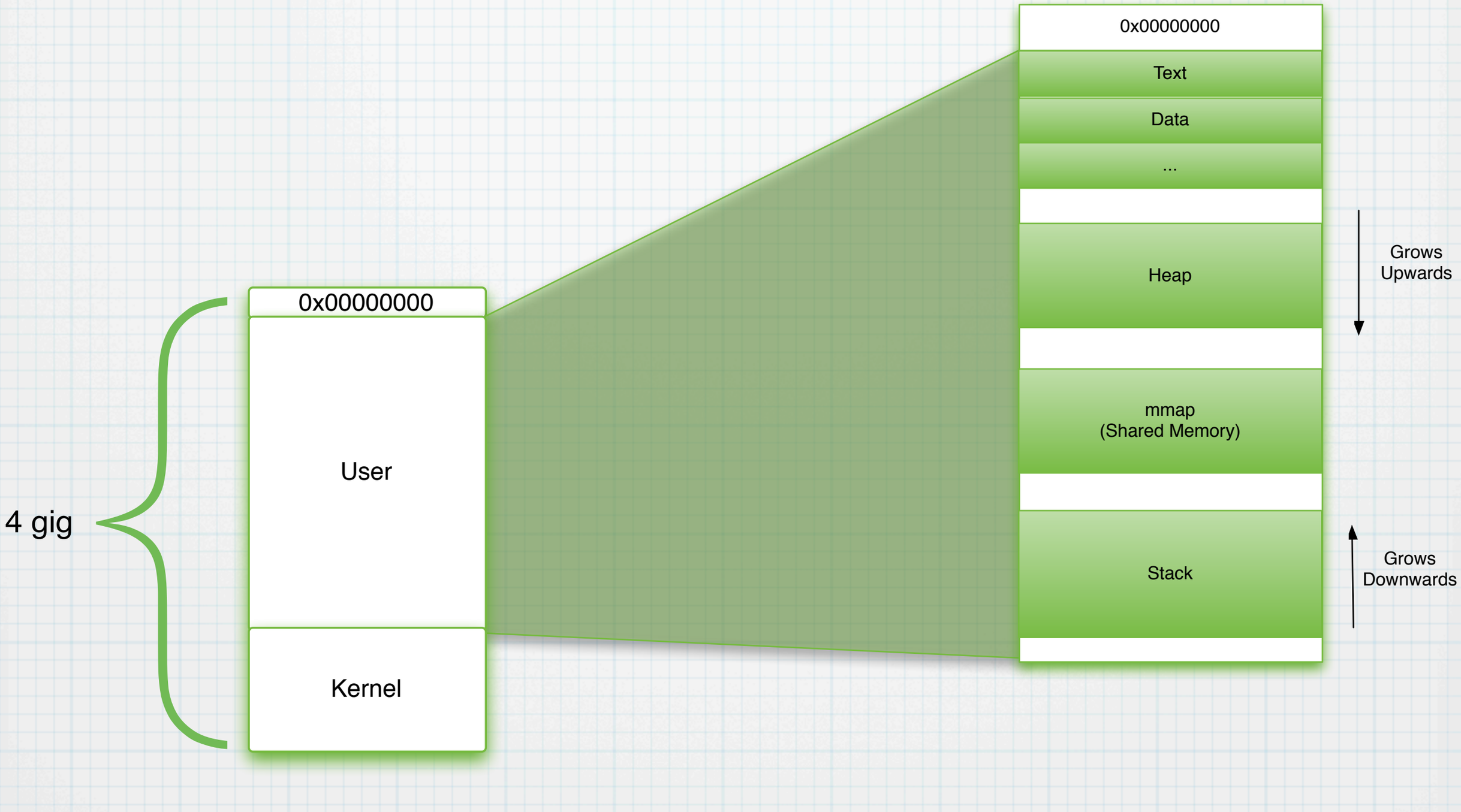
```
0040681D  C3              RETN
0040681E  $ 55            PUSH EBP
0040681F  . 57            PUSH EDI
00406820  . 68 61742069   PUSH 69207461
00406825  .v 73 20        JNB SHORT 00406847
00406827  .v 74 68        JE SHORT 00406891
00406829  . 6973 F8 0083  IMUL ESI,DWORD PTR DS:[EBX-8],-39A7D00
00406830  . 0053 57       ADD BYTE PTR DS:[EBX+57],DL
00406833  . BF 4EE640BB   MOV EDI,BB40E64E
00406838  . 3BC7          CMP EAX,EDI
0040683A  . BB 0000FFFF   MOV EBX,FFFF0000
0040683F  .v 74 0D        JE SHORT 0040684E
00406841  . 85C3          TEST EBX,EAX
00406843  .v 74 09        JE SHORT 0040684F
00406845  . F7D0          NOT EAX
00406847  > A3 DC154100   MOV DWO
0040684C  .v EB 60        JMP SHO
0040684E  > 56            PUSH ES
0040684F  . 8D45 F8       LEA EAX
00406852  . 50            PUSH EA
```

Address	Hex dump	ASCII
0040679F	C4 10 A3 80 2A 41 00 E8 9F B7 FF FF 85 C0 74 65	−▶üÇ*A.$ƒπ à└te
004067AF	68 79 65 40 00 FF 35 74 2A 41 00 E8 86 FB FF FF	hye@. 5t*A.$ªJ
004067BF	59 FF D0 83 F8 FF A3 A8 16 41 00 74 48 68 14 02	Y ╨ªà° ü¿▬A.tHh¶☻
004067CF	00 00 6A 01 E8 95 F2 FF FF 8B F0 85 F6 59 59 74	..j☺$ò≥ ï≡à÷YYt
004067DF	34 56 FF 35 A8 16 41 00 FF 35 7C 2A 41 00 E8 53	4V 5¿▬A. 5!*A.$S
004067EF	FB FF FF 59 FF D0 85 C0 74 1B 6A 00 56 E8 2A FC	√ Y ╨à└t←j.V$*ⁿ
004067FF	FF FF 59 59 FF 15 B0 E0 40 00 83 4E 04 FF 89 06	YY ⌠▒∝@.âN♦ ë♠
0040680F	33 C0 40 EB 07 E8 D5 FB FF FF 33 C0 5E 5F C3 55	3└@ß.$ƒ╒ 3└^_├U
0040681F	57 68 61 74 20 69 73 20 74 68 69 73 F8 00 83 65	What is this°.âe
0040682F	FC 00 53 57 BF 4E E6 40 BB 3B C7 BB 00 00 FF FF	".SW┐Nµ@╗;╟╗..
0040683F	74 0D 85 C3 74 09 F7 D0 A3 DC 15 41 00 EB 60 56	t.à├t.≈╨ü▄⌠A.ß'V
0040684F	8D 45 F8 50 FF 15 C4 E0 40 00 8B 75 FC 33 75 F8	ìE°P ⌠─∝@.ïuⁿ3u°
0040685F	FF 15 C0 E0 40 00 33 F0 FF 15 B0 E0 40 00 33 F0	⌠└∝@.3≡ ⌠▒∝@.3≡
0040686F	FF 15 BC E0 40 00 33 F0 8D 45 F0 50 FF 15 B8 E0	⌠╝∝@.3≡ìE≡P ⌠╕∝
0040687F	40 00 8B 45 F4 33 45 F0 33 F0 3B F7 75 07 BE 4F	@.ïE⌡3E≡3≡;≈u·╛O
0040688F	E6 40 BB EB 0B 85 F3 75 07 8B C6 C1 E0 10 0B F0	µ@╗ß∂à≤u·ï├╨∝▶∂≡
0040689F	89 35 D8 15 41 00 F7 D6 89 35 DC 15 41 00 5E 5F	ë5╪⌠A.≈╓ë5▄⌠A.^_
004068AF	5B C9 C3 8B 44 24 04 A3 84 2A 41 00 C3 FF 74 24	[╔├ïD$♦üä*A.├ t$
004068BF	04 FF 15 C8 E0 40 00 33 C0 40 C2 08 00 6A 14 68	♦ ⌠╚∝@.3└@┬◘.j¶h
004068CF	30 F8 40 00 E8 18 C3 FF FF 33 FF 89 7D E4 FF 35	0°@.$↑├ 3 ë}Σ 5
004068DF	84 2A 41 00 E8 5D FA FF FF 59 8B F0 3B F7 75 53	ä*A.$]· Yï≡;≈uS



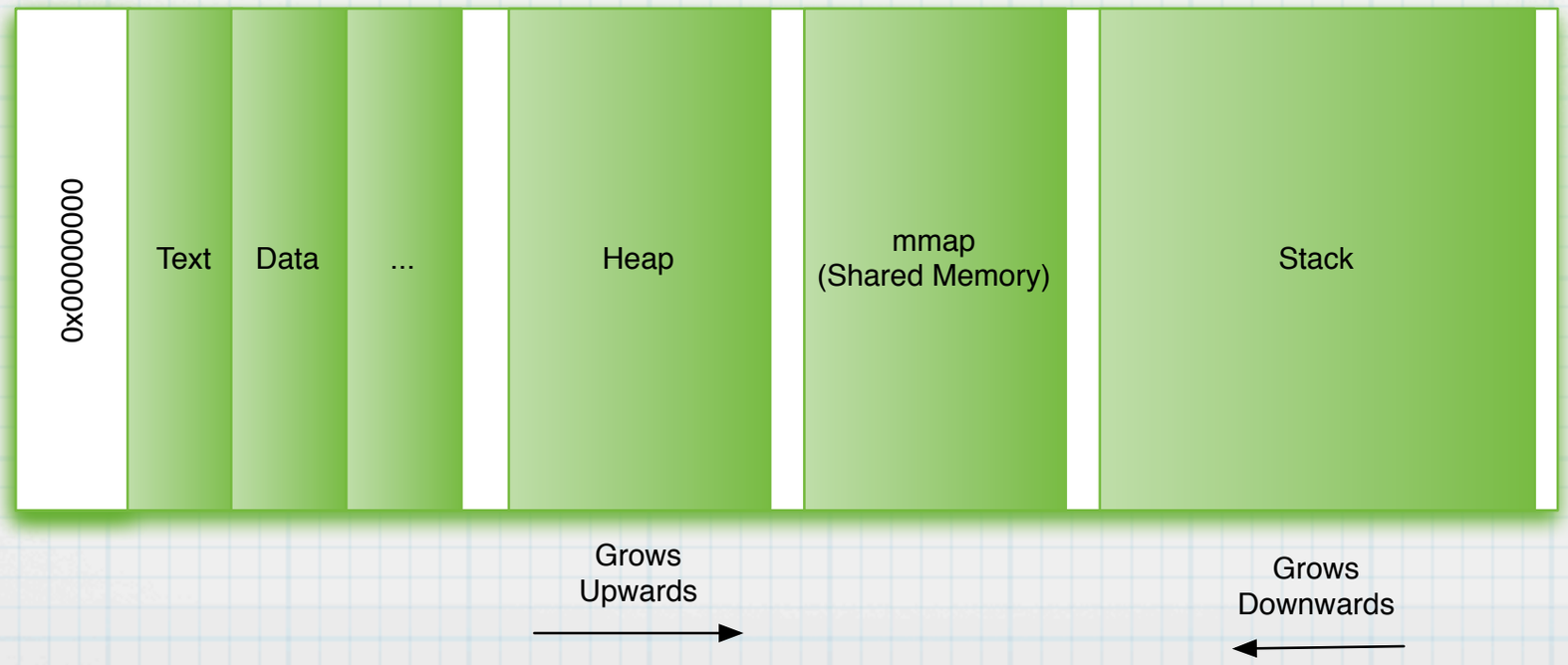

Memory
Control Unit
Arithmetic Logic Unit
Accumulator
Input
Output


ALU
Instructions memory
Control unit
Data memory
I/O



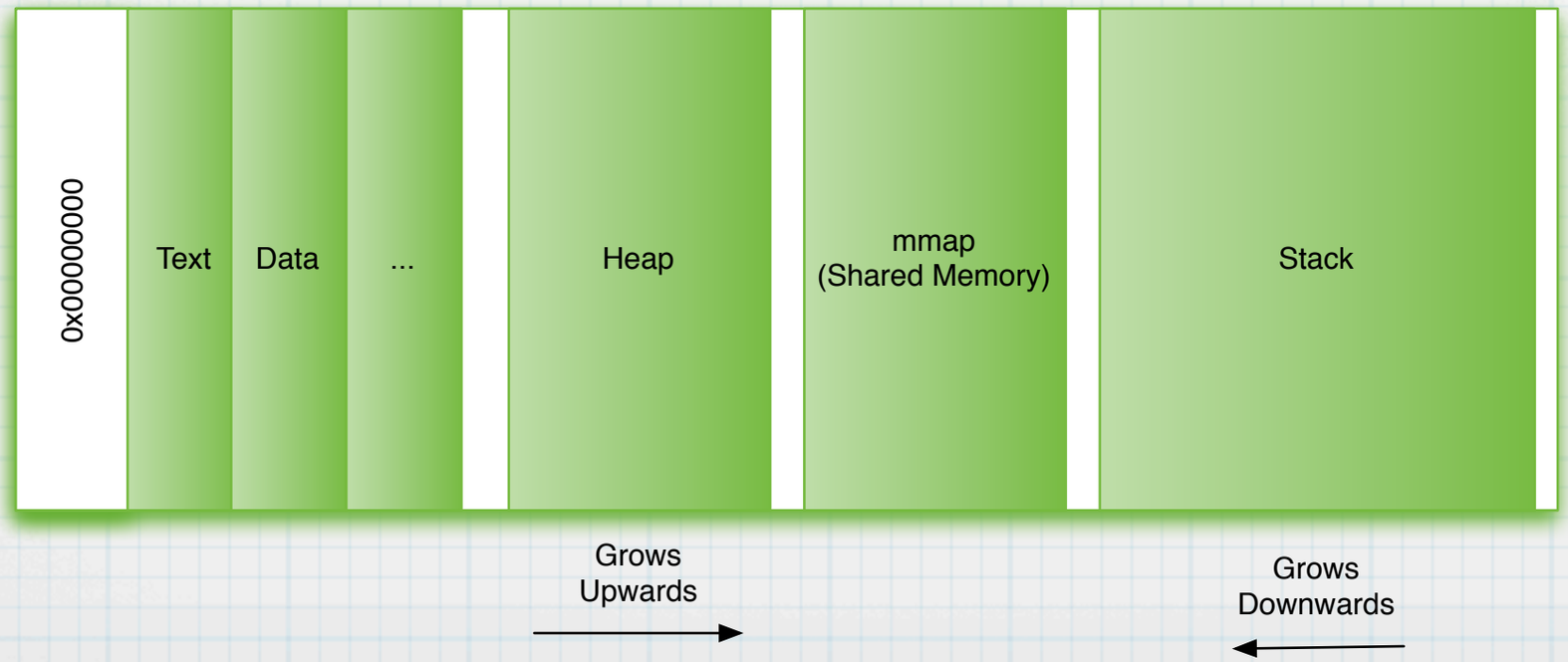
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



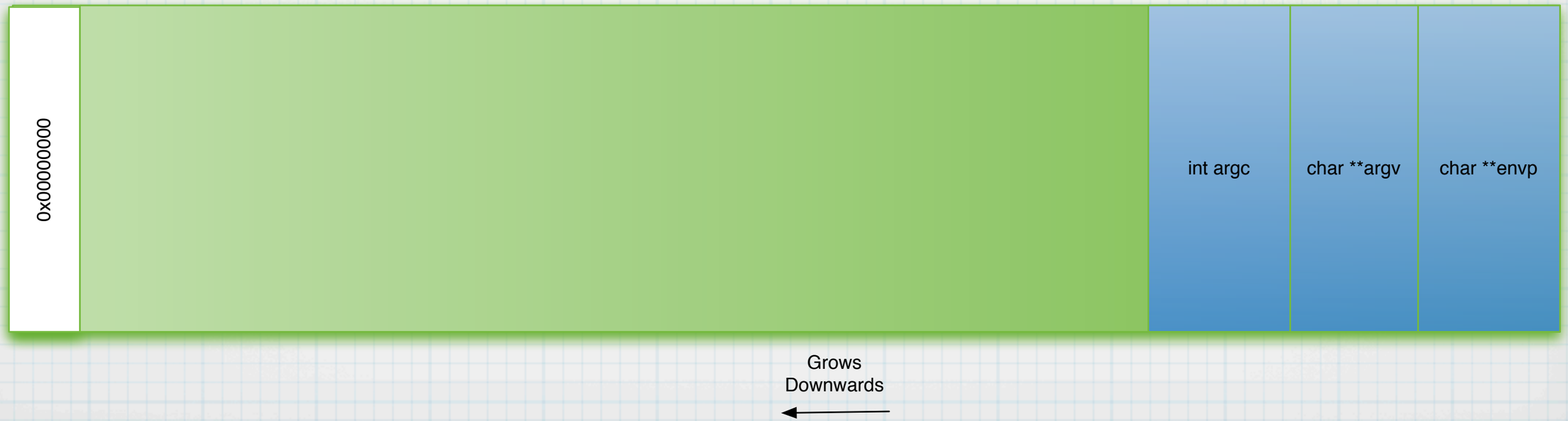
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



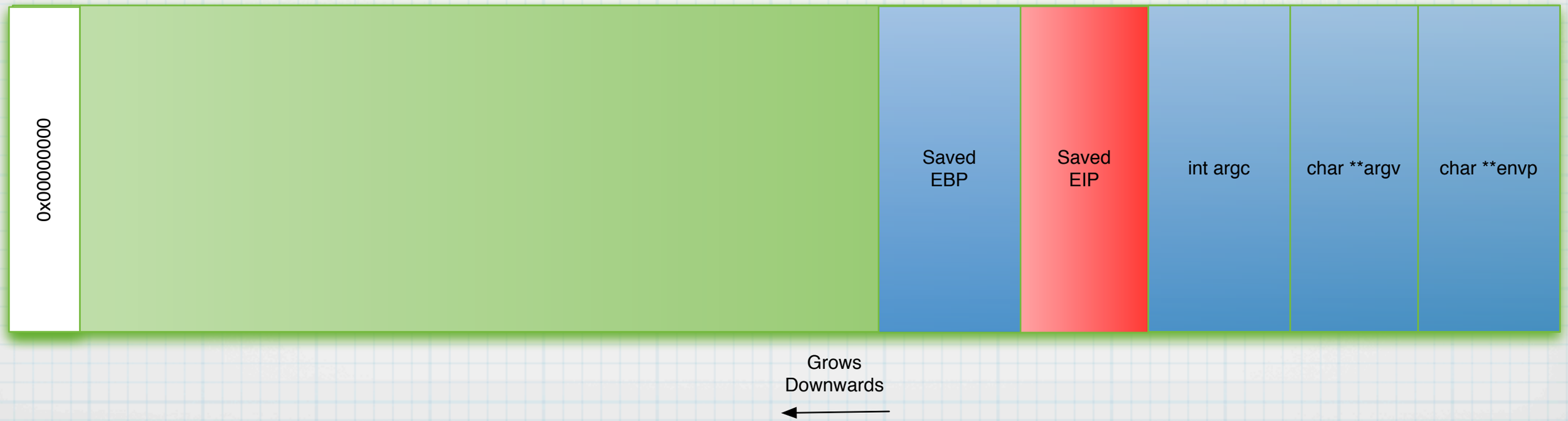
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



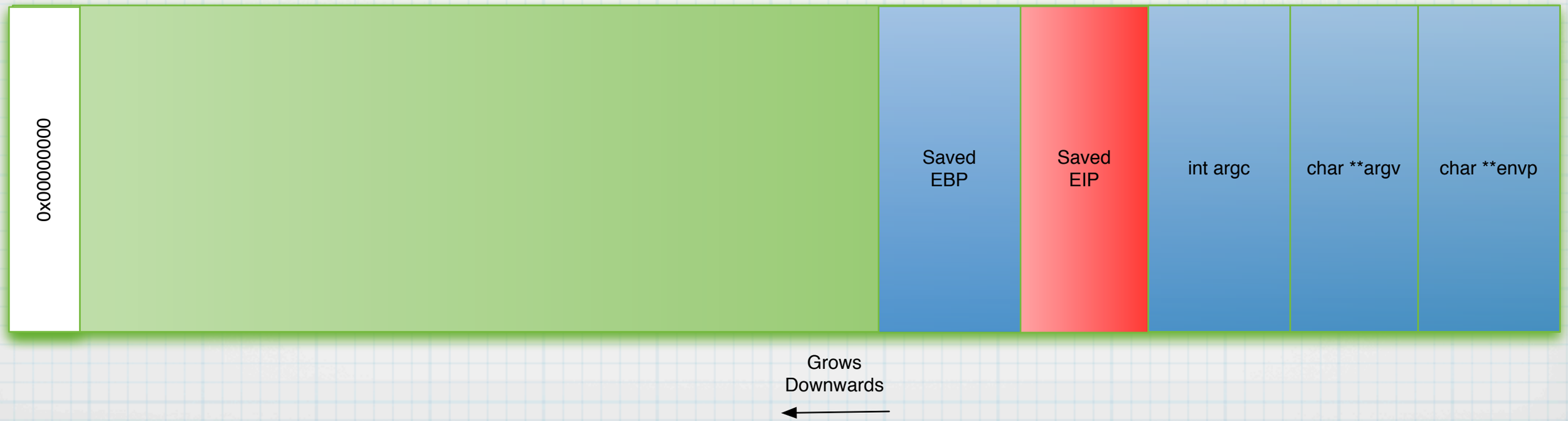
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



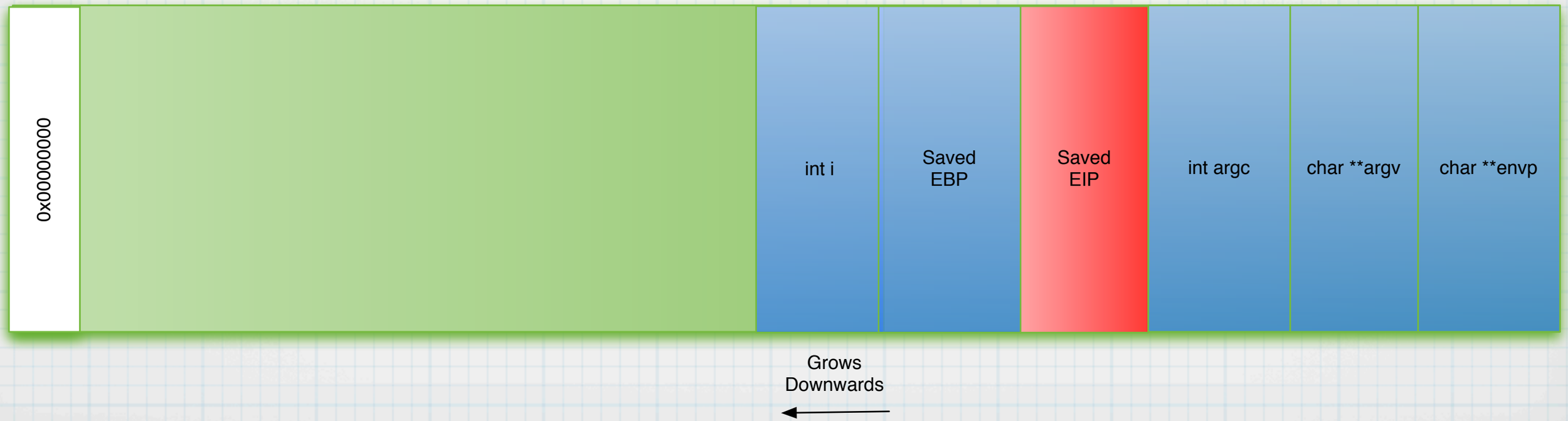
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



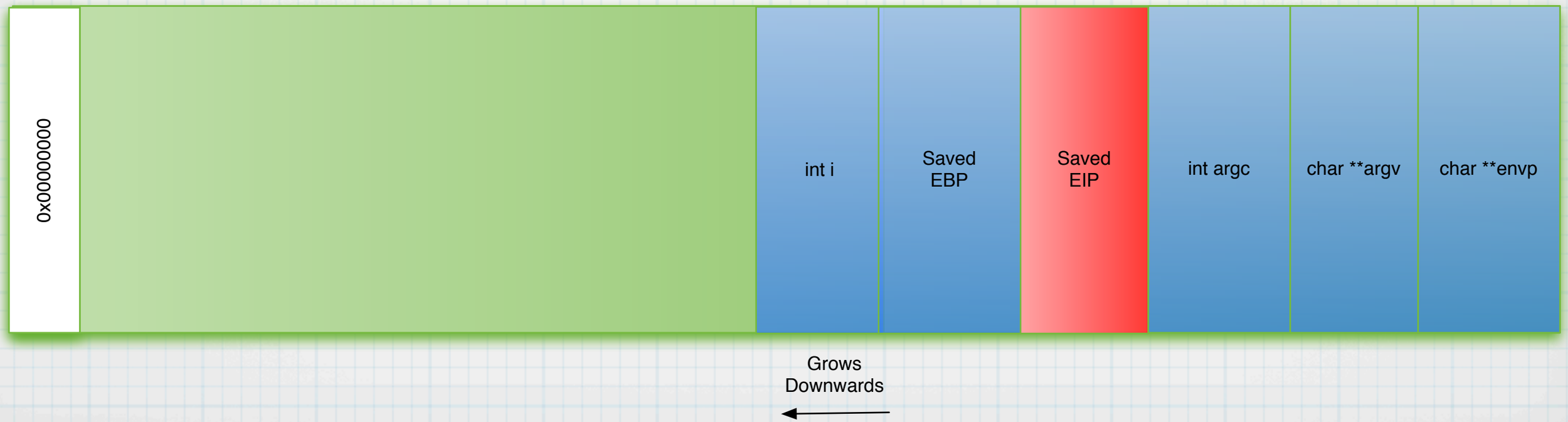
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



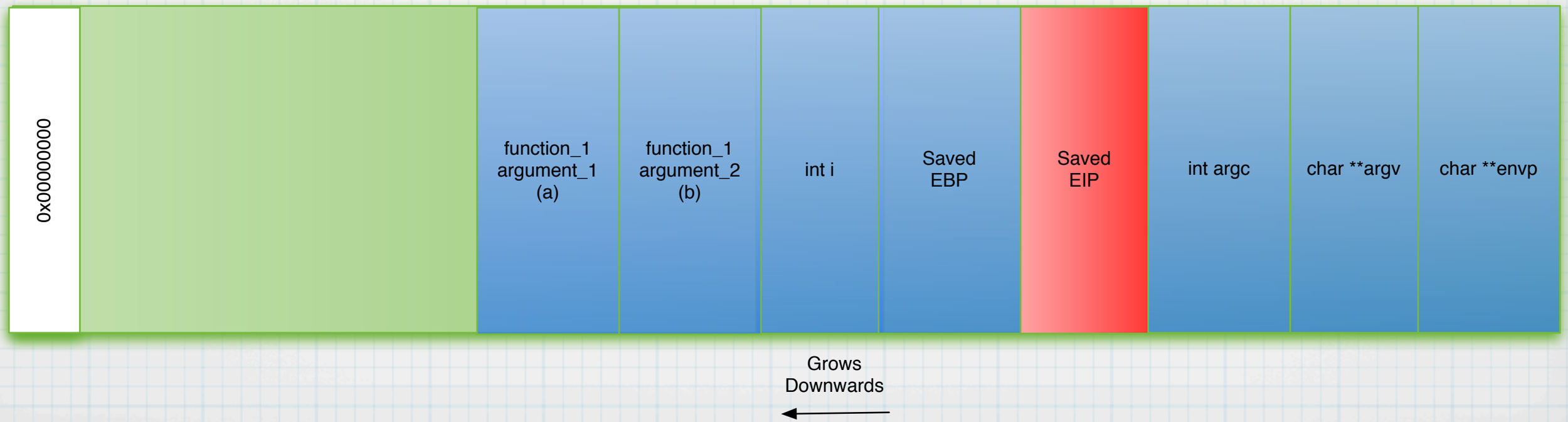
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



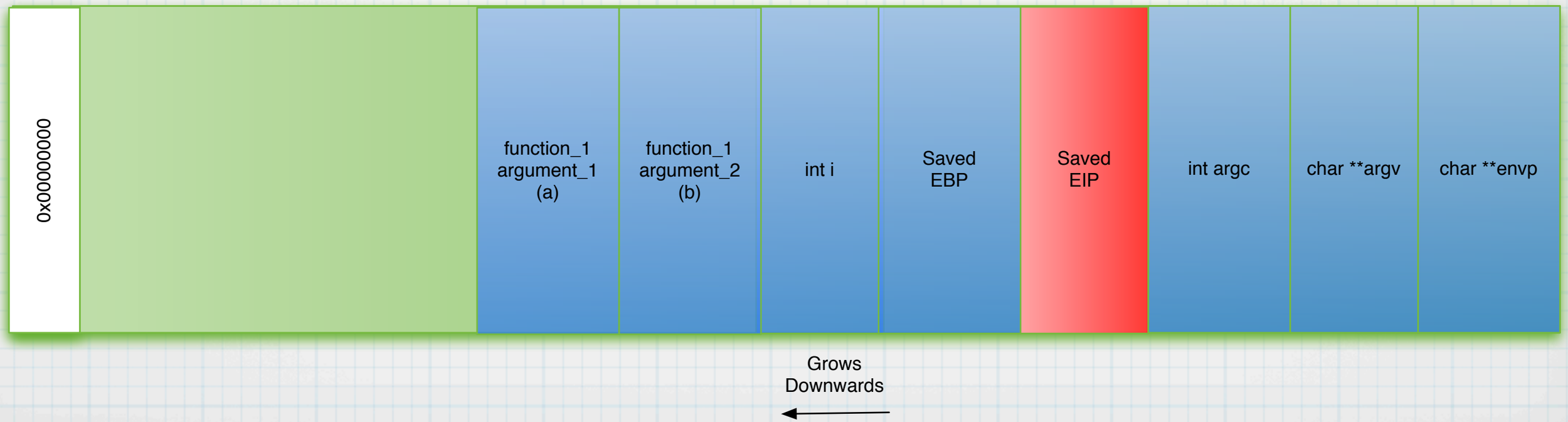
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



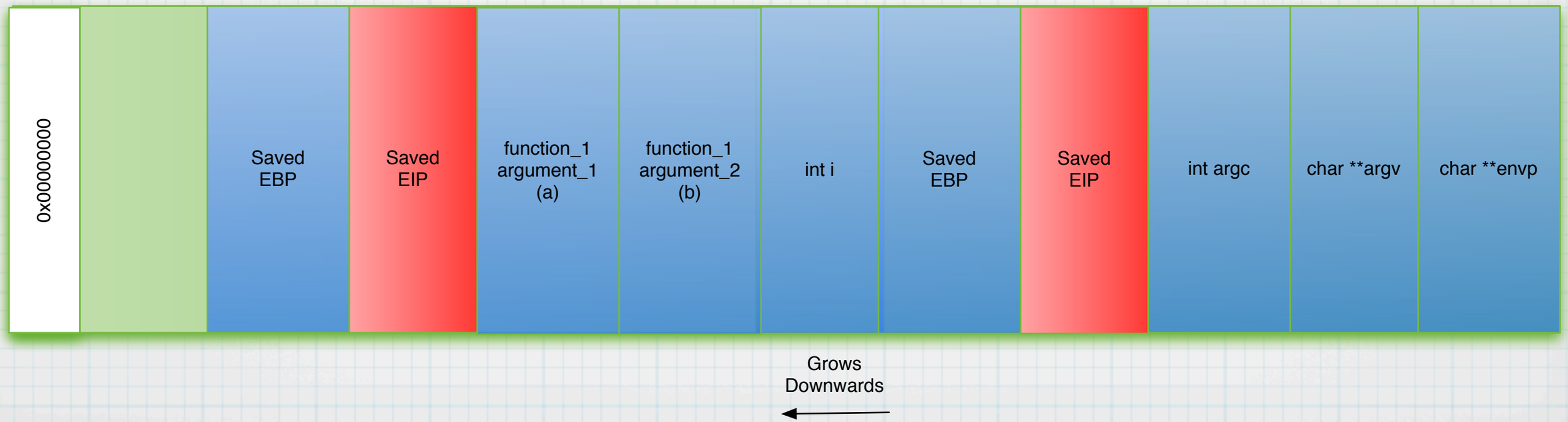
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



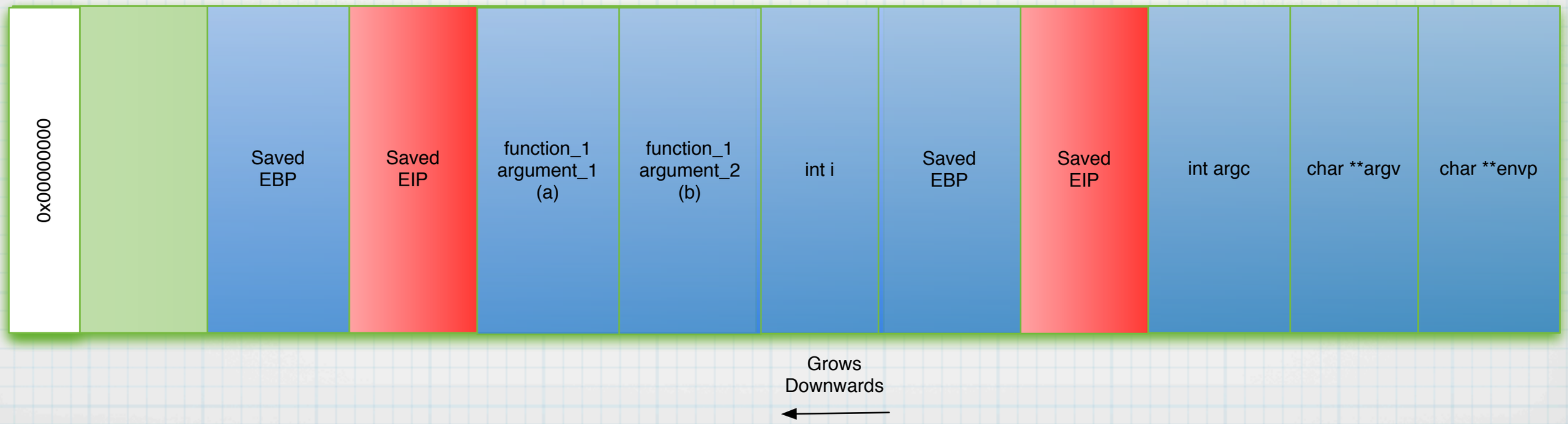
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



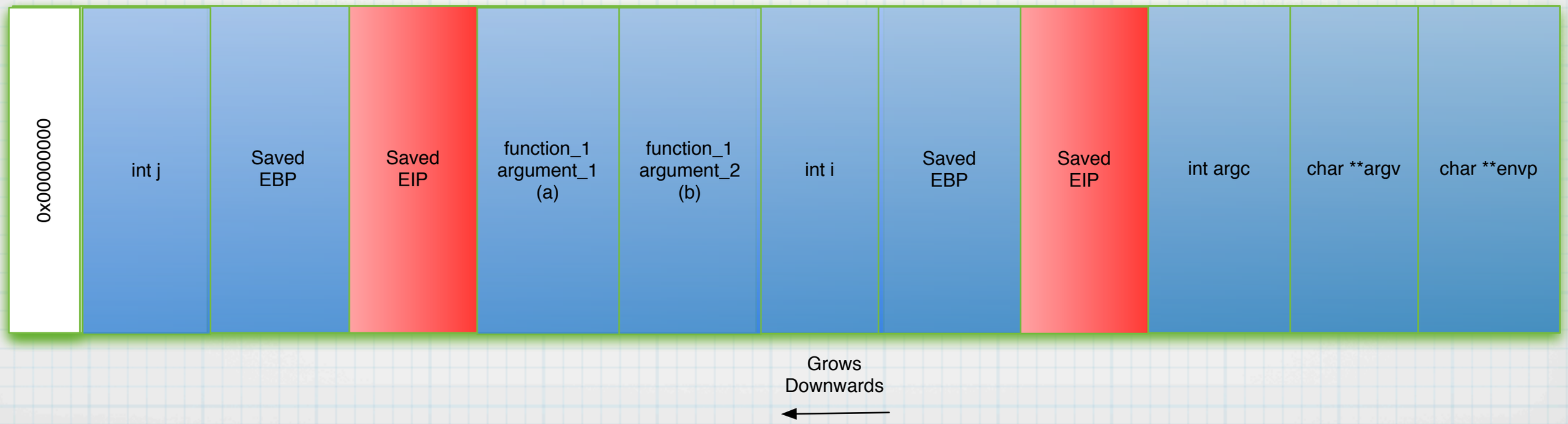
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



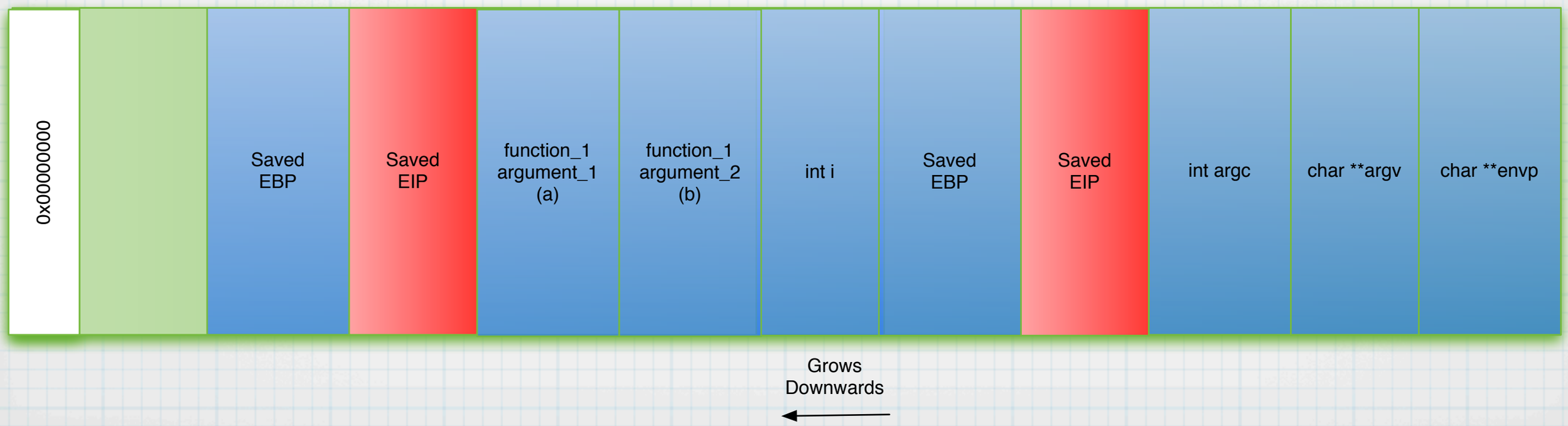
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



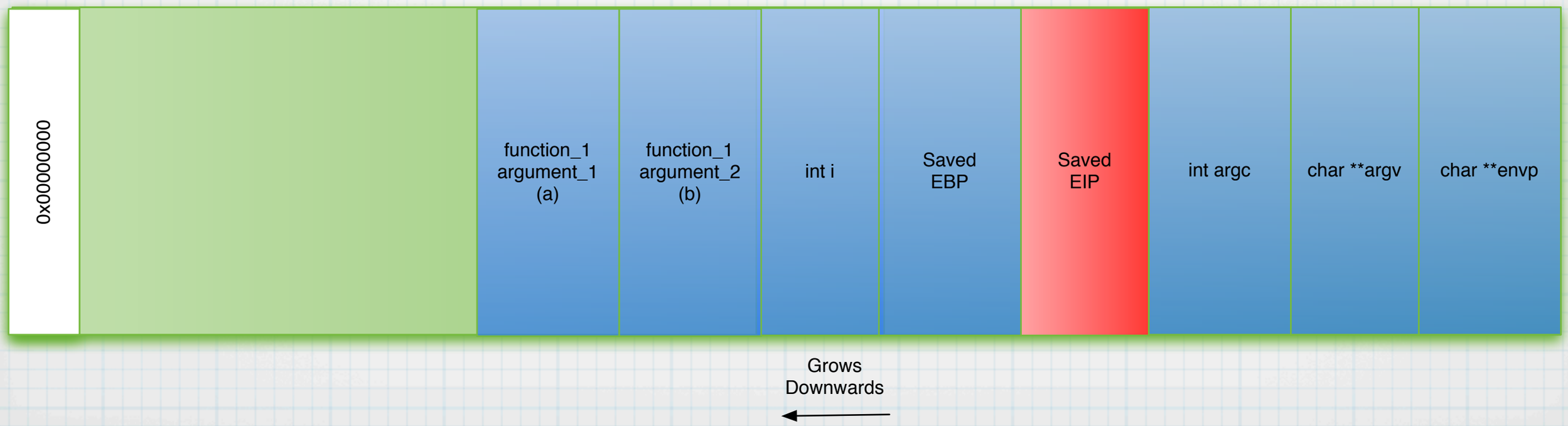
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



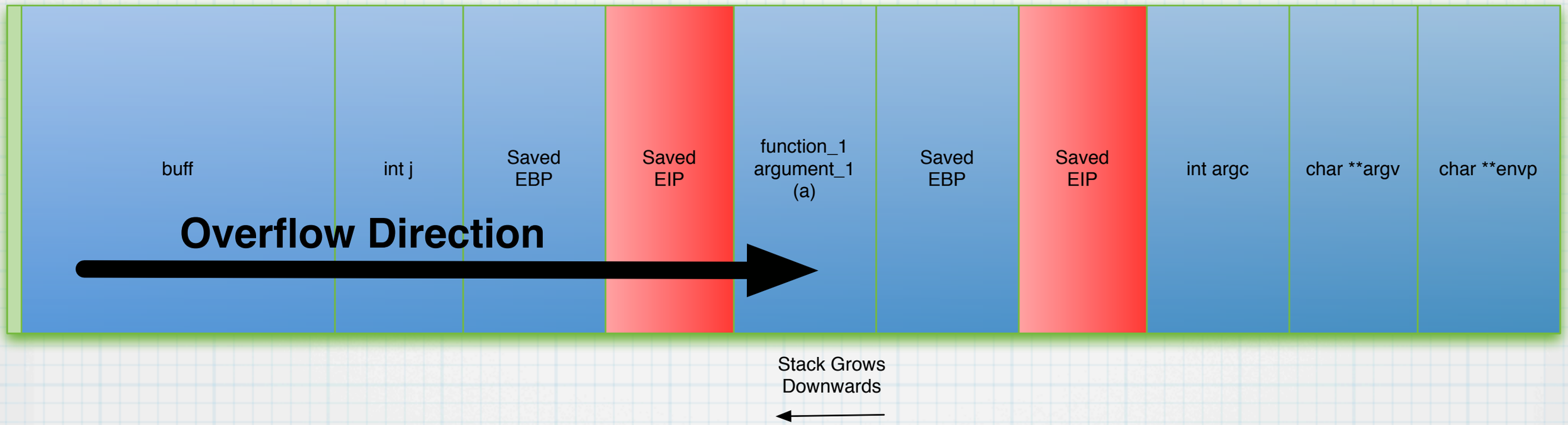
Stack Basics

```
int function_1(int a, int b)
{
   int j;
   // do stuff
   return j;
}

int main(int argc, char **argv, char **envp)
{
   int i;
   i = function_1(1,2);
   printf("answer is %d", i)
   return i;
}
```



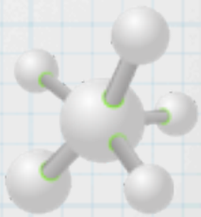
Classic Overflow

Where to go



non-terminated strings

```
strcpy(buf1, buf2);
```

non-terminated strings

non-terminated strings


```
char buf1[4];
strncpy(buf1, buf2, 4);
```

non-terminated strings


```
char buf1[4];
strncpy(buf1, buf2, 4);
```

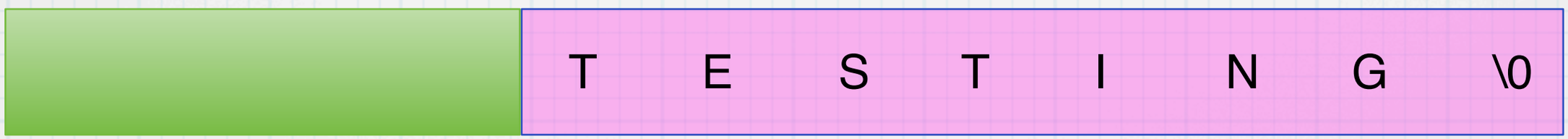
```
The strncpy() function copies at most n characters from s2 into s1.  If s2 is less than n char-
acters long, the remainder of s1 is filled with `\0' characters.  Otherwise, s1 is not termi-
nated.
```

non-terminated strings

```
strcpy(     , buf2);
```

```
char buf1[4];
strncpy(buf1, buf2, 4);
```

> The **strncpy**() function copies at most n characters from s2 into s1. If s2 is less than n characters long, the remainder of s1 is filled with `\0' characters. Otherwise, s1 is not terminated.



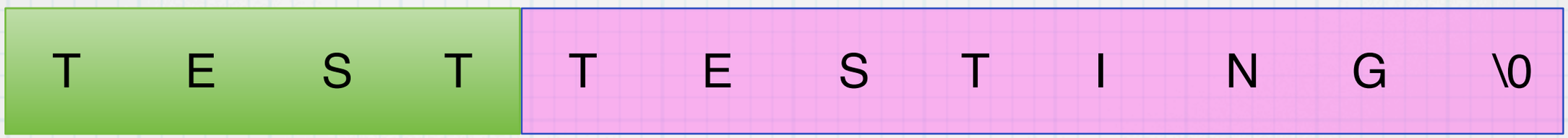
char buf1[4] **char buf2[] = "TESTING"**

non-terminated strings

```
strcpy(     , buf2);
```

```
char buf1[4];
strncpy(buf1, buf2, 4);
```

> The **strncpy**() function copies at most n characters from s2 into s1. If s2 is less than n characters long, the remainder of s1 is filled with `\0' characters. Otherwise, s1 is not terminated.



char buf1[4] **char buf2[] = "TESTING"**

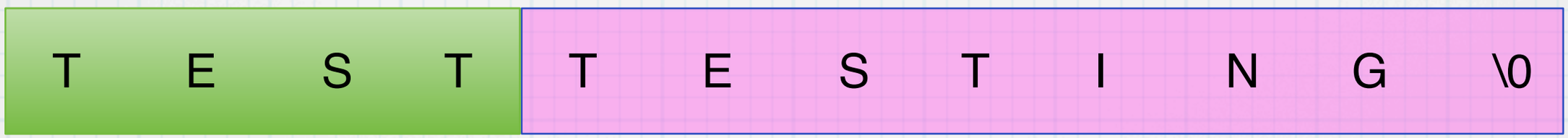
thinkst applied research

non-terminated strings

```
strcpy(     , buf2);
```

```
char buf1[4];
strncpy(buf1, buf2, 4);
```

> The **strncpy**() function copies at most n characters from s2 into s1. If s2 is less than n characters long, the remainder of s1 is filled with `\0' characters. Otherwise, s1 is not terminated.



char buf1[4] **char buf2[] = "TESTING"**

```
printf("buf1 is [%s]\n",buf1);
```

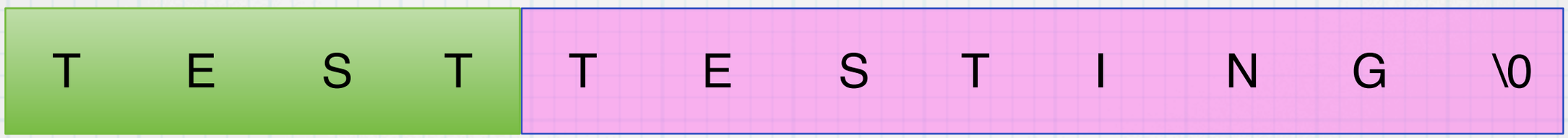
thinkst applied research

non-terminated strings

```
strcpy(     , buf2);
```

```
char buf1[4];
strncpy(buf1, buf2, 4);
```

> The **strncpy**() function copies at most n characters from s2 into s1. If s2 is less than n characters long, the remainder of s1 is filled with `\0' characters. Otherwise, s1 is not terminated.

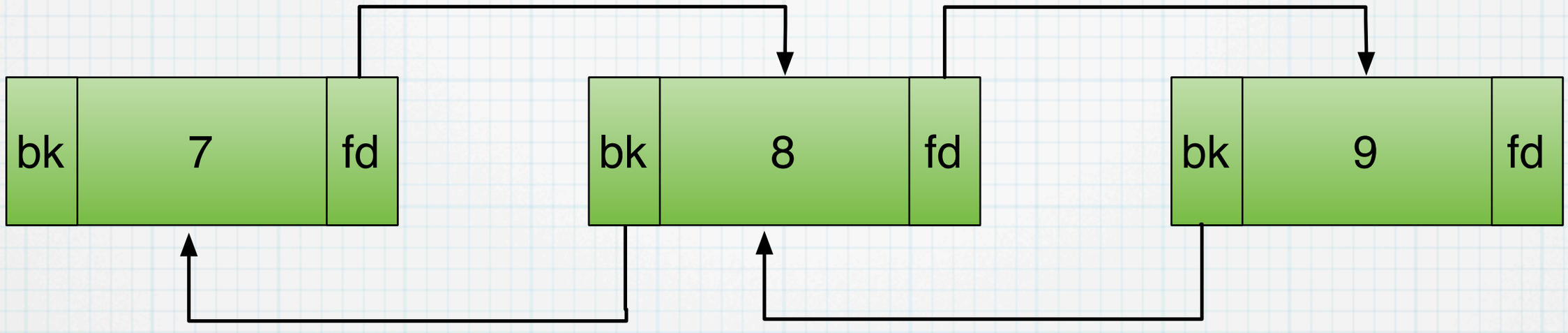


char buf1[4] **char buf2[] = "TESTING"**

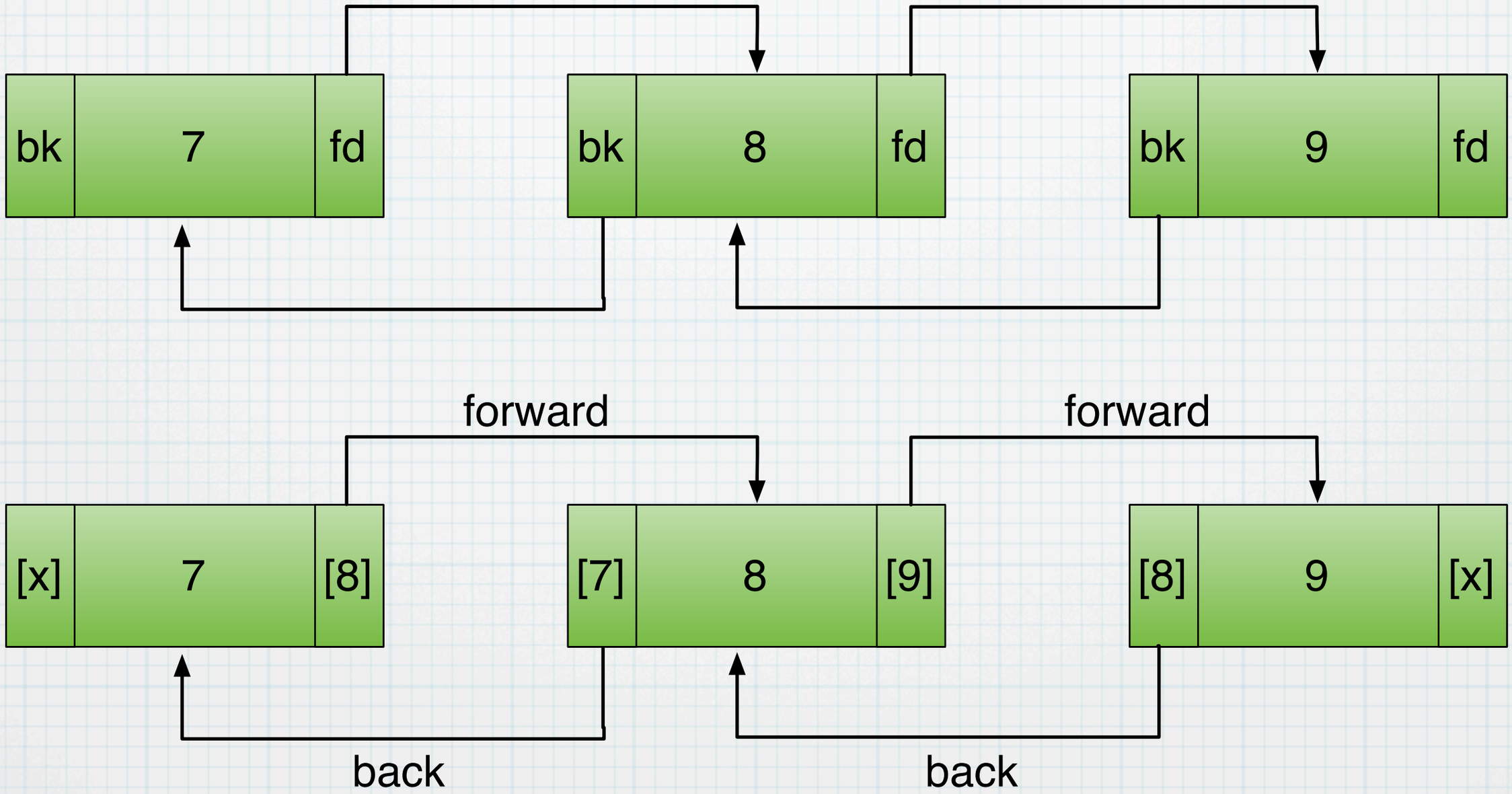
```
printf("buf1 is [%s]\n",buf1);

$ buf1 is [TESTTESTING]
```

thinkst applied research

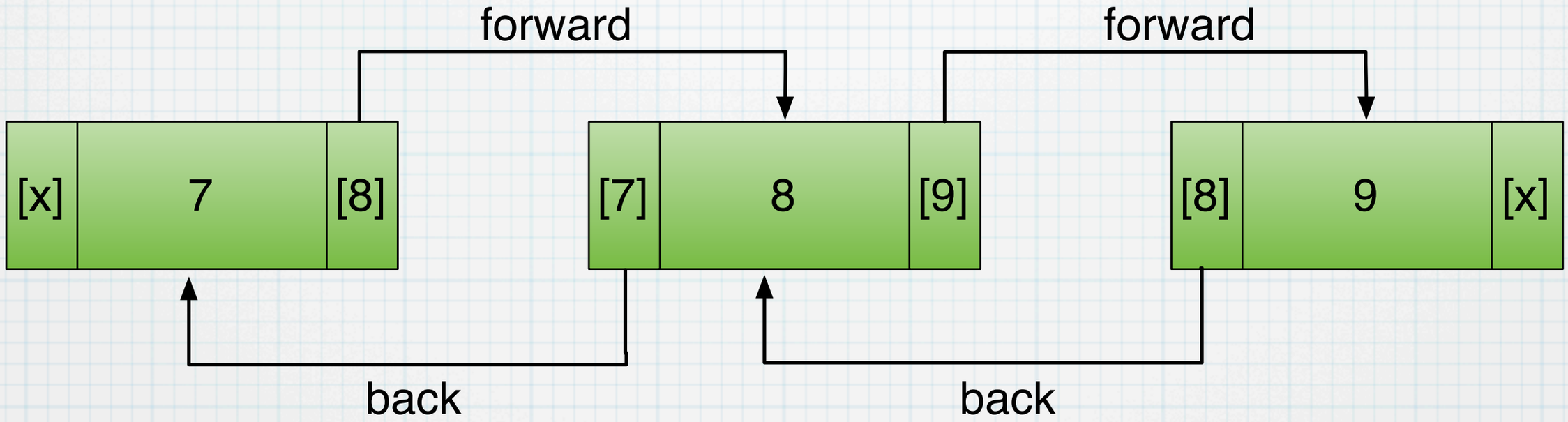
heap-unlink()



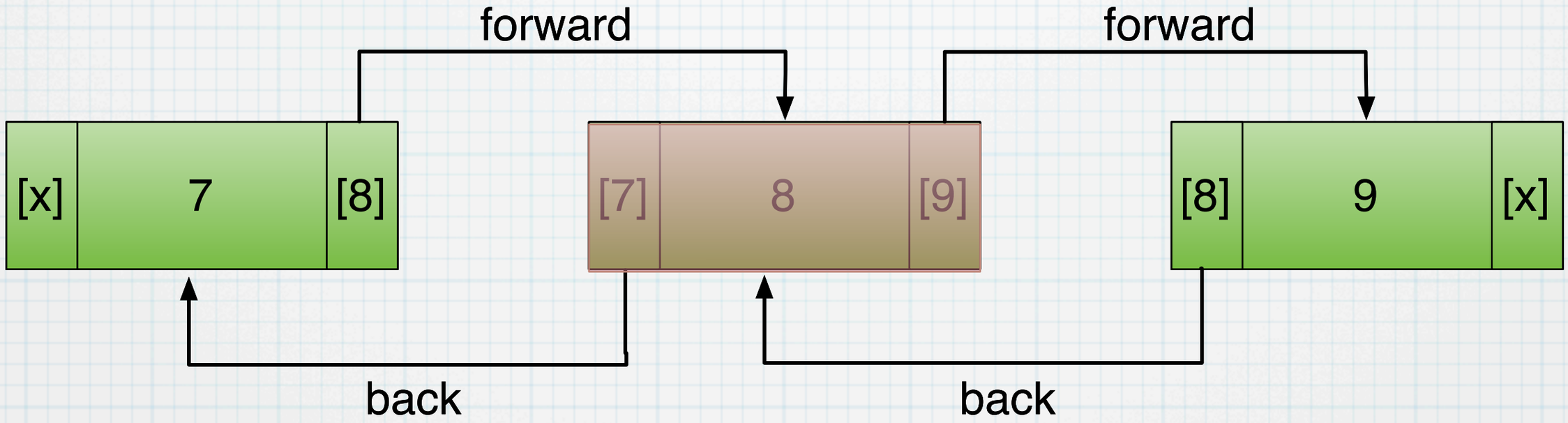
heap-unlink()



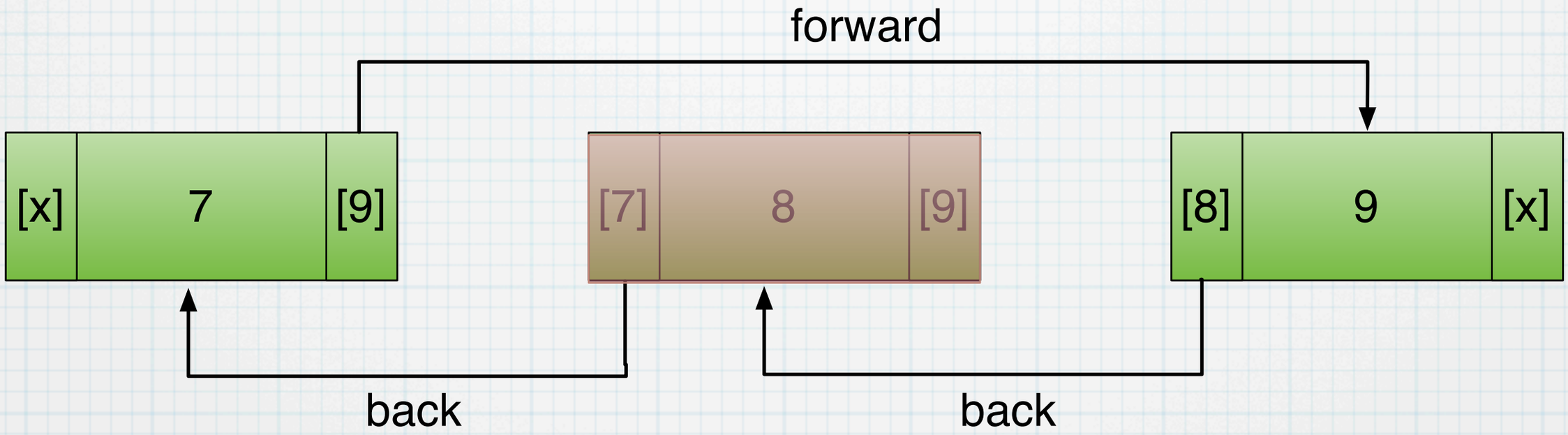
heap-unlink()



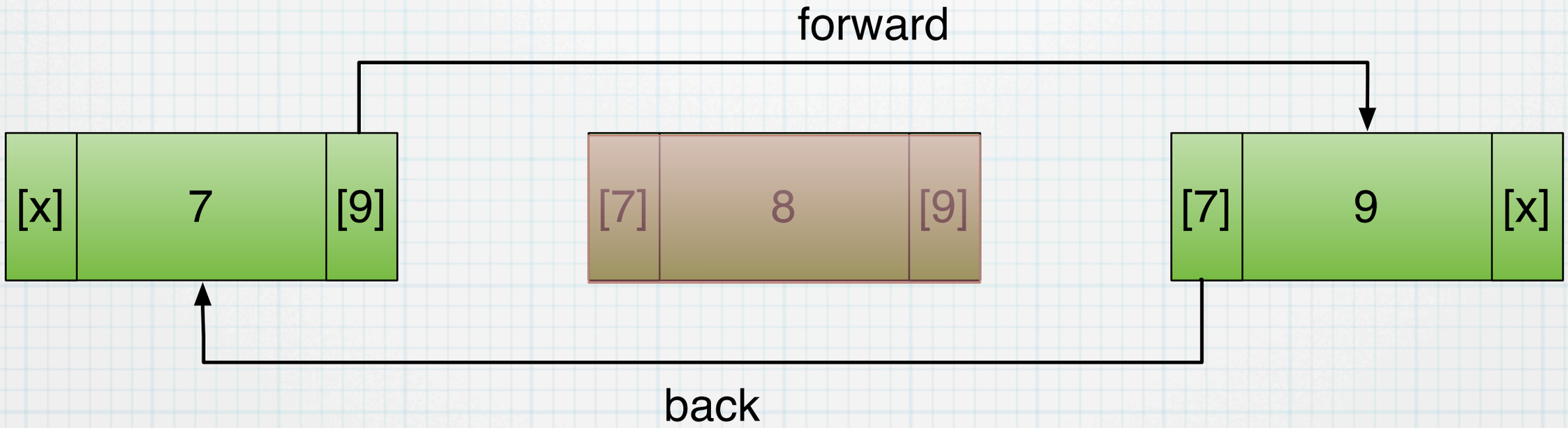
heap-unlink()



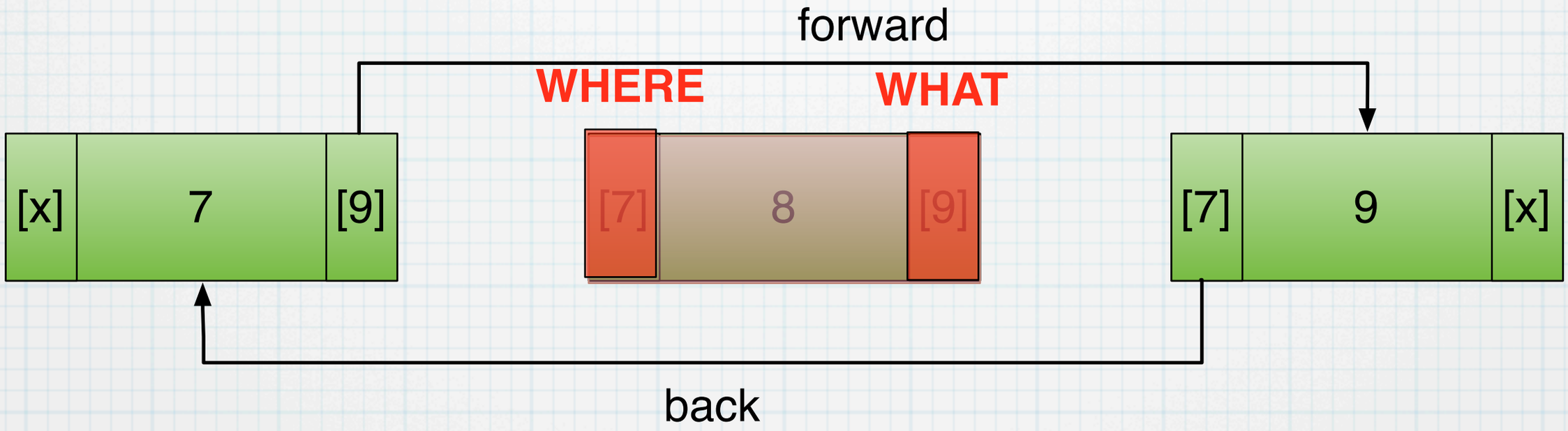
heap-unlink()



heap-unlink()



heap-unlink()



slapper

Client

Server Listens
On Port 443/TCP

Client hello

cipher suite, challenge

Server hello

cipher suite, connection id, certificate

Send client master key

E(server public key, master key), cipher, key argument

Buffer overflow!

Send server verify

E(master key, challenge)

Send client finished

E(master key, connection id)

leaked values

*cipher

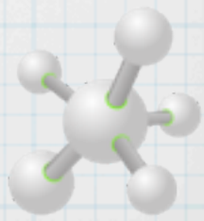
*ciphers

Send server finished

E(master key, session id+leaked information)

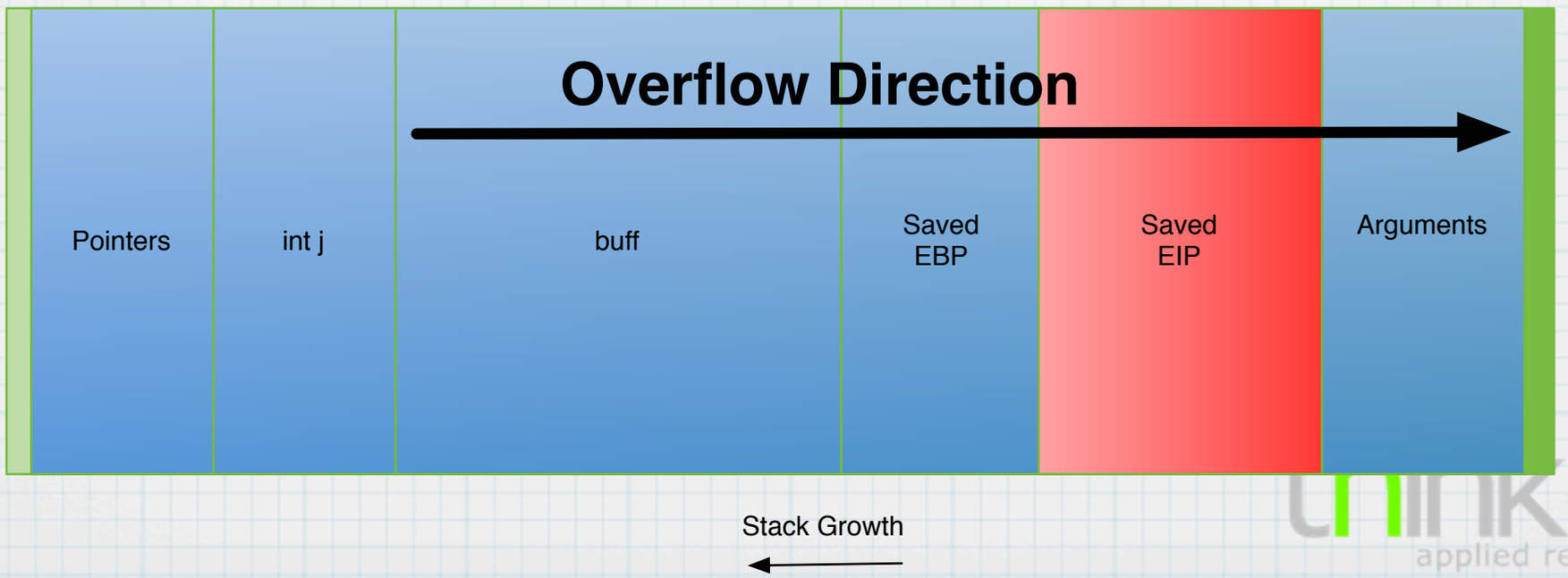
* Peter Szor - analysis of the slapper worm

thinkst applied research



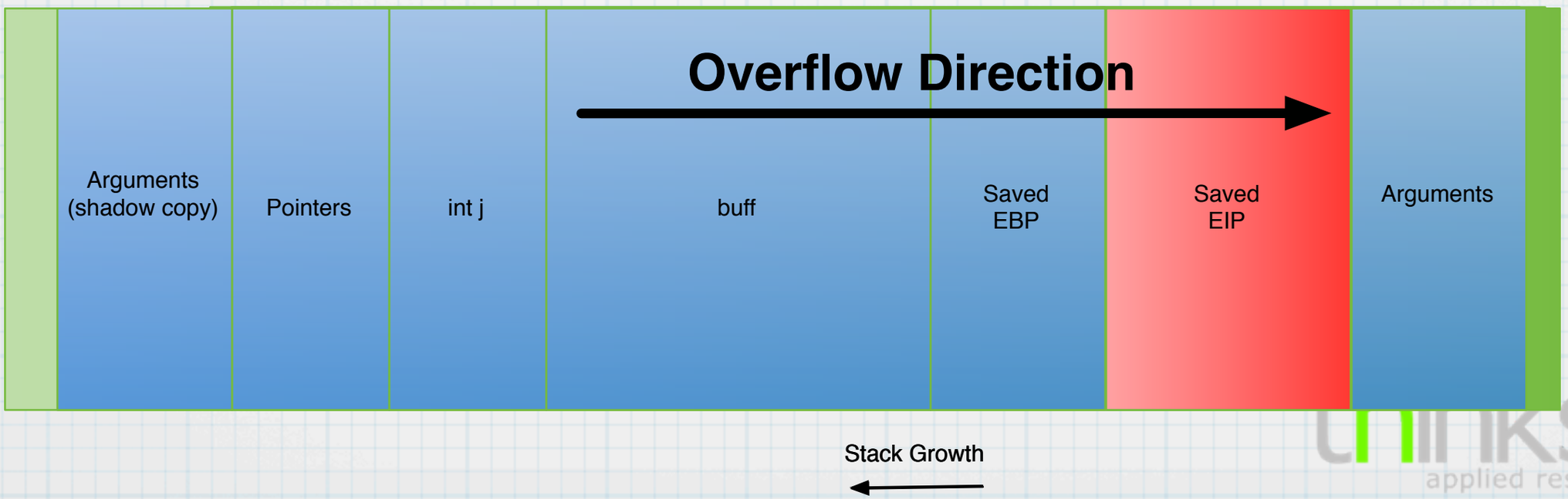
gs vs params

```
int func(char *a, char *b)
{
    char buf[12];
    strcpy(buf, a);
    strcpy(b, buf);
    return 1;
}
```



gs vs params

```
int func(char *a, char *b)
{
    char buf[12];
    strcpy(buf, a);
    strcpy(b, buf);
    return 1;
}
```

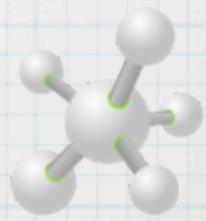


so..

- Everything executable --> DEP
- DEP vs ret-2-libc (ROP)

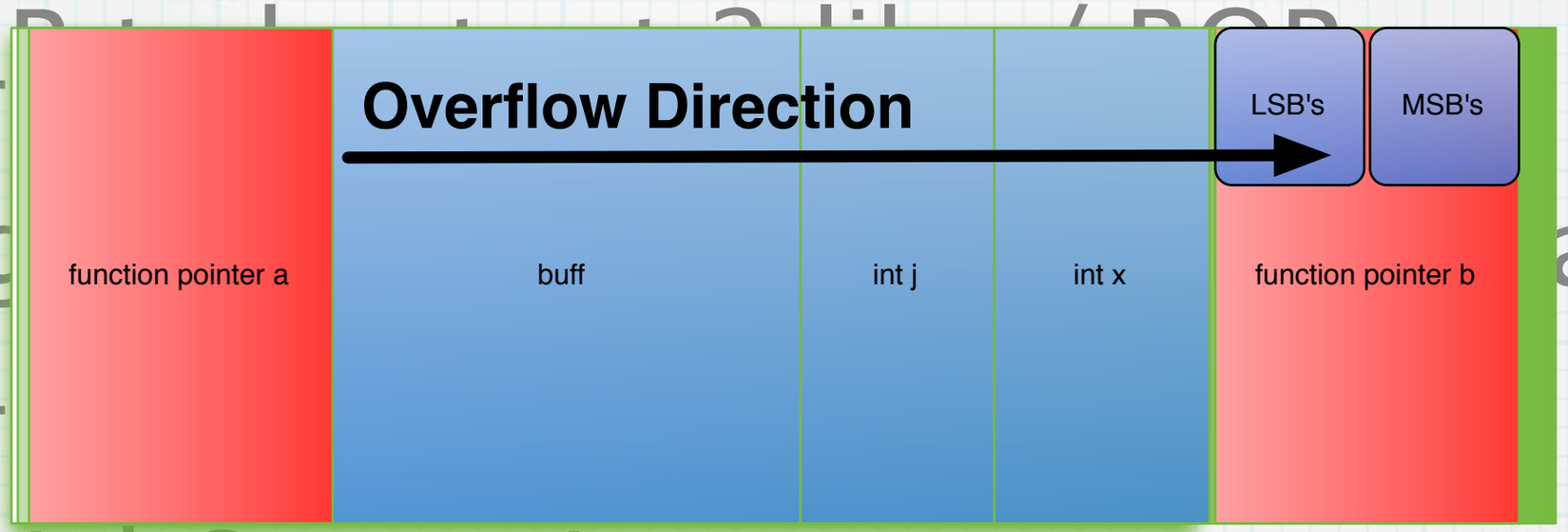
so..

- ASLR to beat ret-2-libc / ROP
- Single leaked / static address beats ASLR
- Partial Overwrites
- App specific..



so..

- ASL
- Sing ats
  ASL
- Partial Overwrites
- App specific..



thinkst
applied research

So..

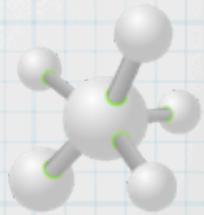
Application	DEP (7)	DEP (XP)	Full ASLR
Flash Player	N/A	N/A	**YES**
Sun Java JRE	no	no	no
Adobe Reader	**YES***	**YES***	no
Mozilla Firefox	**YES**	**YES**	no
Apple Quicktime	no	no	no
VLC Media Player	no	no	no
Apple iTunes	YES	no	no
Google Chrome	**YES**	**YES**	**YES**
Shockwave Player	N/A	N/A	no
OpenOffice.org	no	no	no
Google Picasa	no	no	no
Foxit Reader	no	no	no
Opera	**YES**	**YES**	no
Winamp	no	no	no
RealPlayer	no	no	no
Apple Safari	**YES**	**YES**	no

DEP & ASLR (June 2010)

Secunia - June 2010

- DEP without ASLR
- ASLR without DEP
- without

thinkst
applied research



Conclusions?

- What the ASLR/DEP taketh..
- The rich client side applications giveth back.
- Info. leakage attacks are an area of much research

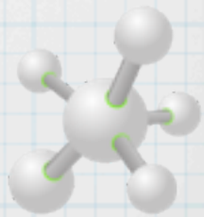
http://ilm.thinkst.com/folklore/

thinkst
applied research

Thanks!

- *Marco Slaviero*
- *Brad Spengler (spender)*
- *PaX Team*
- *Halvar Flake*
- *icesurfer*
- *Nate Lawson*
- *Chris Wysopal*
- *Saumil Shah*
- *Matt Miller*
- *Ollie Whitehouse*
- *Dennis Groves*
- *Ivan Arce*
- *Mario Vilas*
- *Tyler Shields*
- *Dion Blazakis*
- *georgie*
- *Ben Nagy*
- *the Grugq.*
- *Bradley Cowie*
- *Barry Irwin*

thinkst
applied research



Questions ?

http://ilm.thinkst.com/folklore

@haroonmeer
haroon@thinkst.com
http://blog.thinkst.com

thinkst
applied research