Drinking from LETHE:

New methods of exploiting and mitigating memory corruption vulnerabilities

Daniel Selifonov

DEF CON 23
August 7, 2015

Show of Hands

1. Have you written programs in C or C++?
2. Have you implemented a classic stack smash exploit?
3. … a return-to-libc or return-oriented-programming exploit?
4. … a return-to-libc or ROP exploit that used memory disclosure or info leaks?

Motivations

- Software is rife with memory corruption vulnerabilities
- Most memory corruption vulnerabilities are directly applicable to code execution exploits
- And there's no end in sight...

Motivations (II)

- Industrialized ecosystem of vulnerability discovery and brokering weaponized exploits
- Little of this discovery process feeds into fixes...

The *other* AFL

Motivations (III)

- State actor (e.g. NSA Tailored Access Operations group) budgets: ≈ $∞
- Bug bounties just drive up prices
- Target supply, not demand for exploits...

The Plan

- Sever the path between vulnerability and (reliable) exploit
- Why do programmers keep hitting this fundamental blindspot?
- Defenses are born in light of attack strategies

Memory Safety

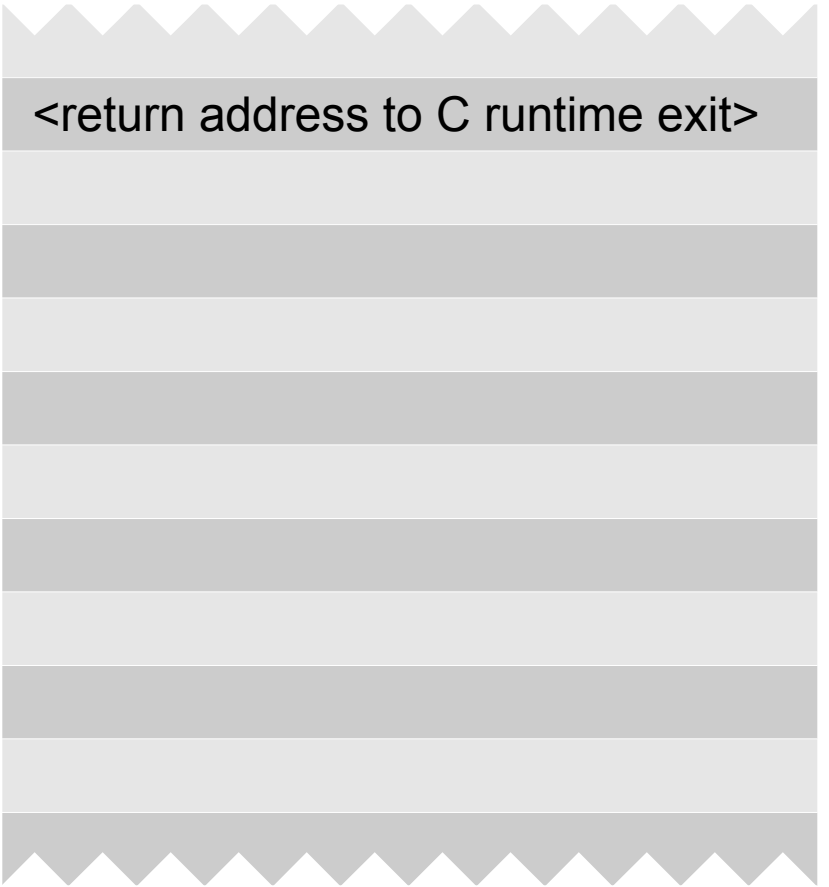
```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```

Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```

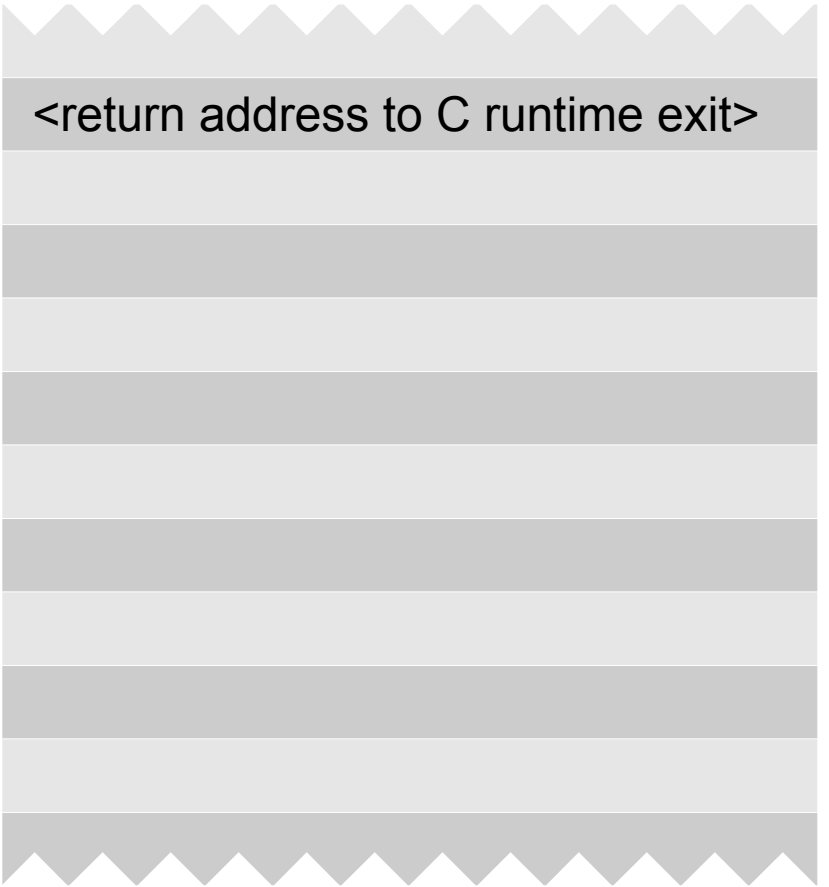
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



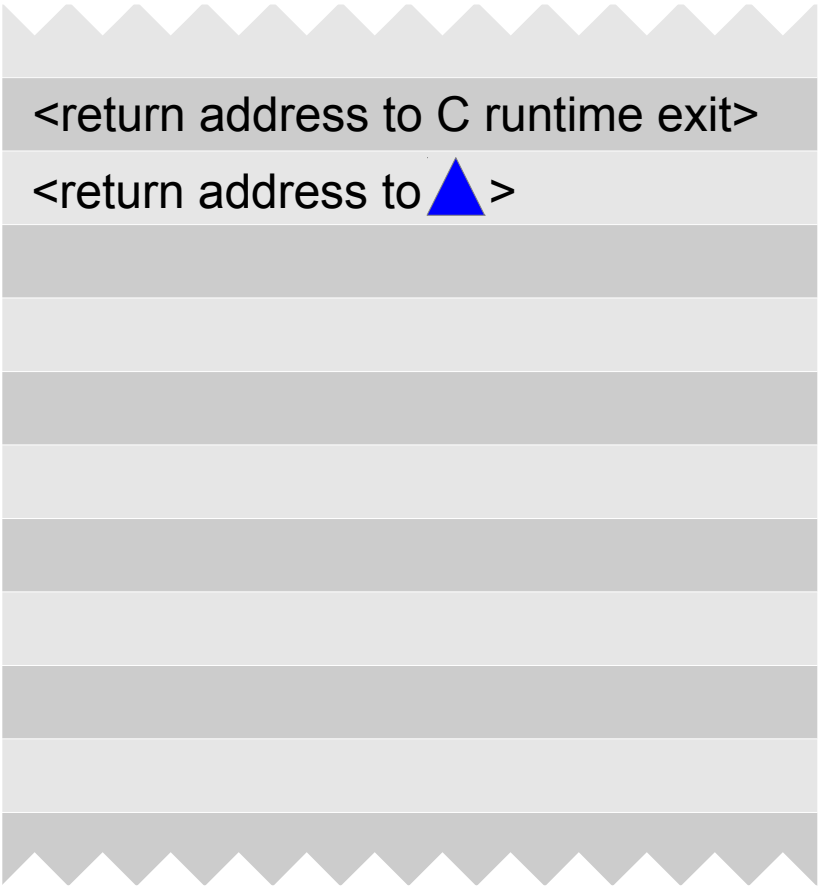
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



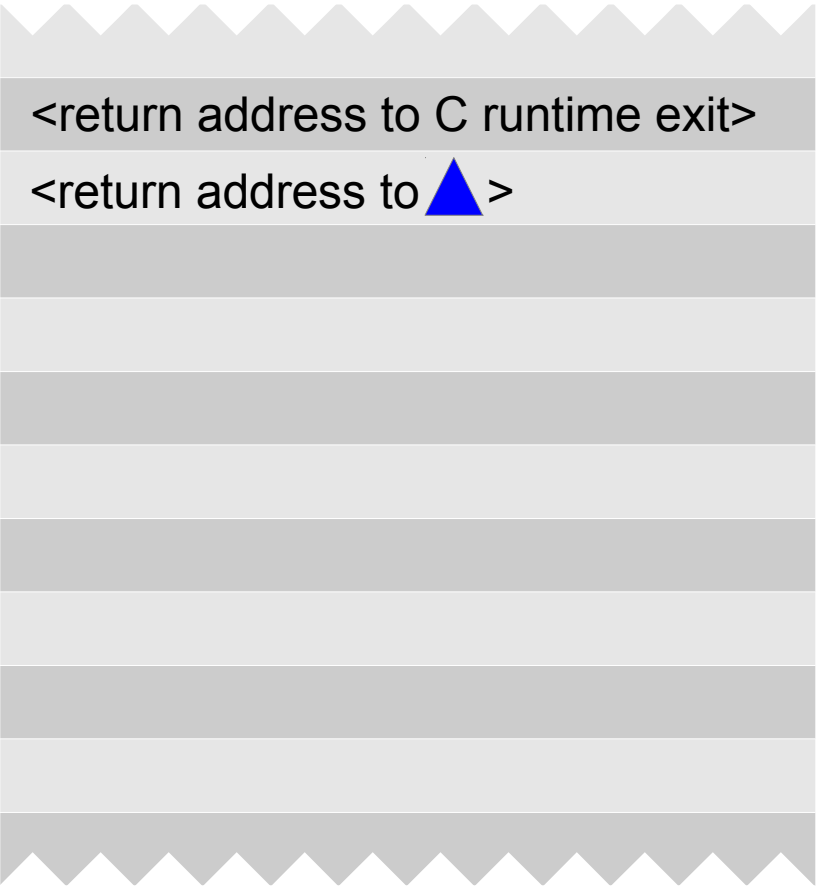
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



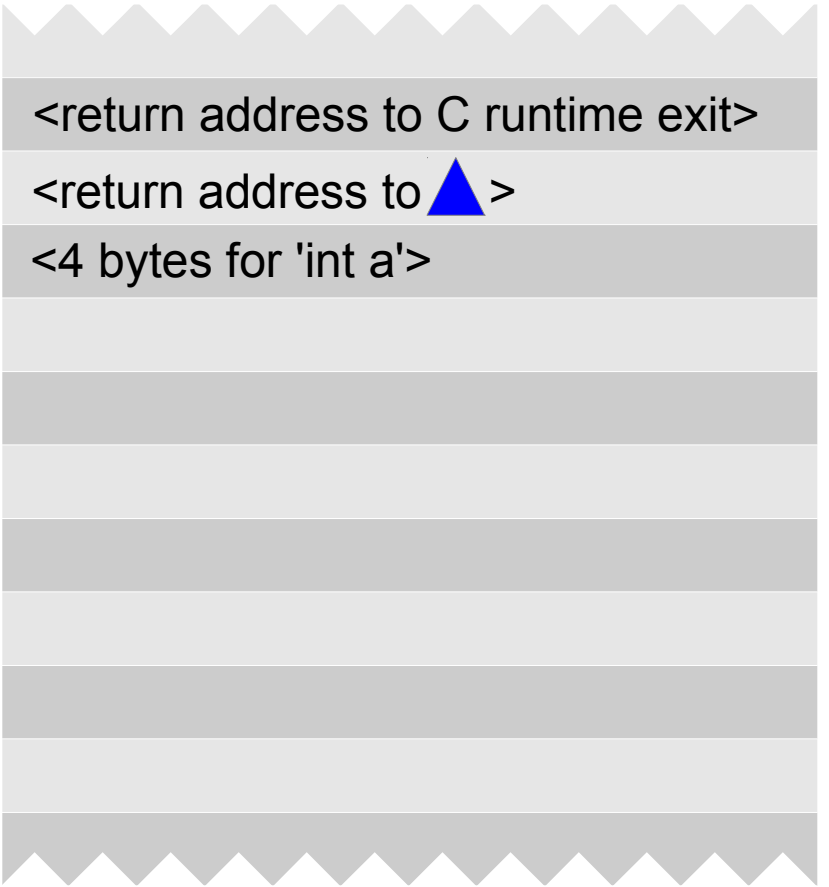
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



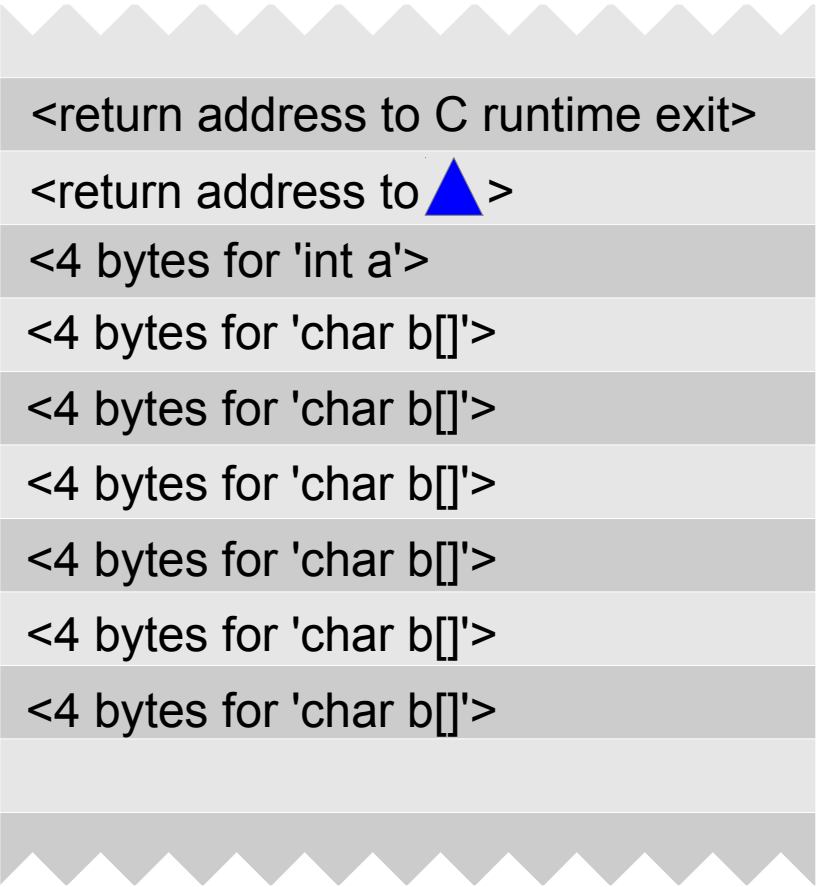
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



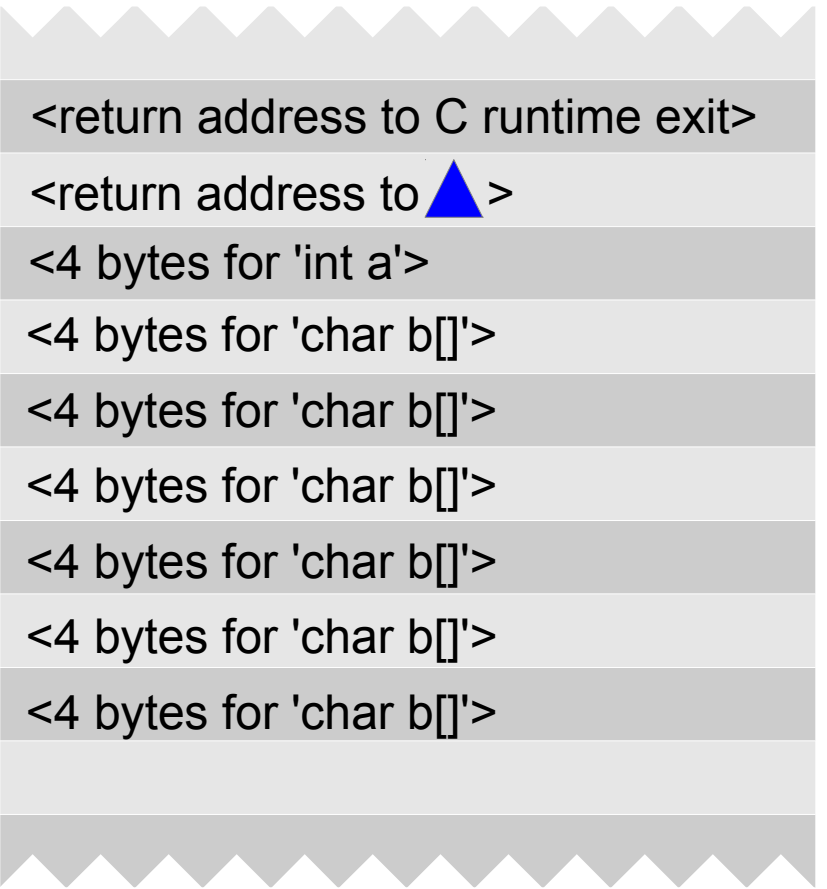
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```

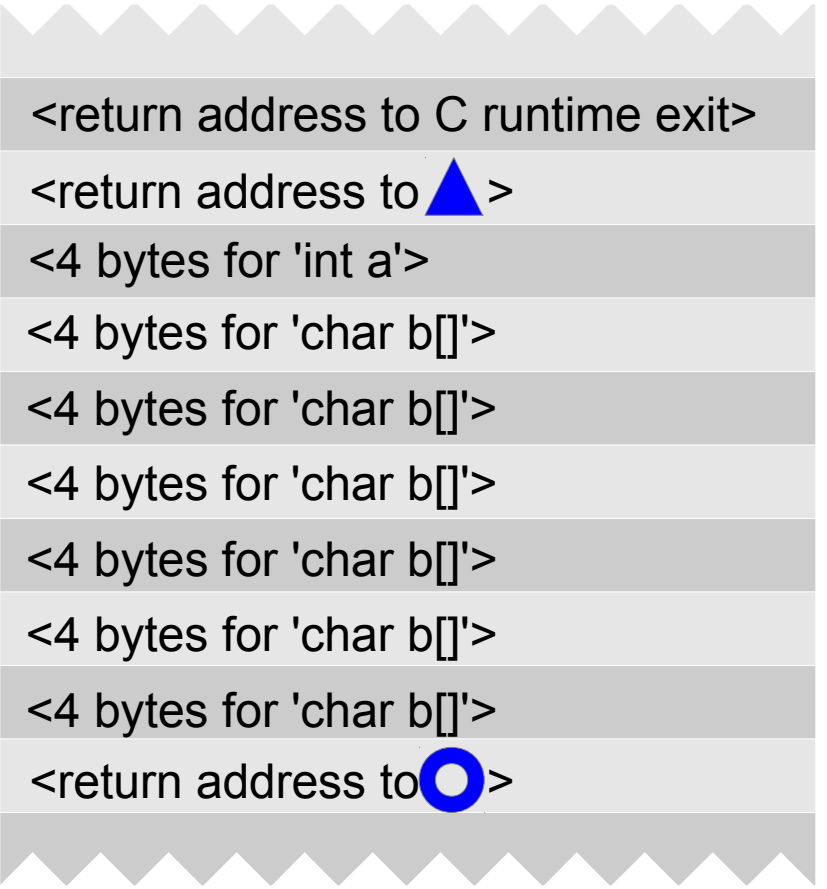
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```

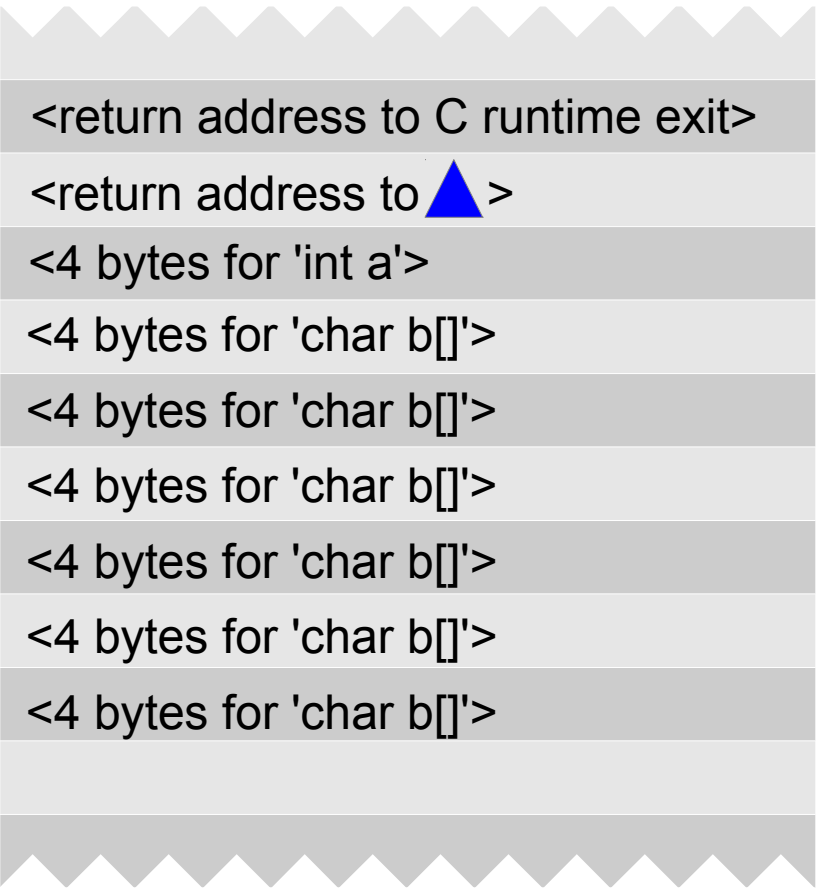
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



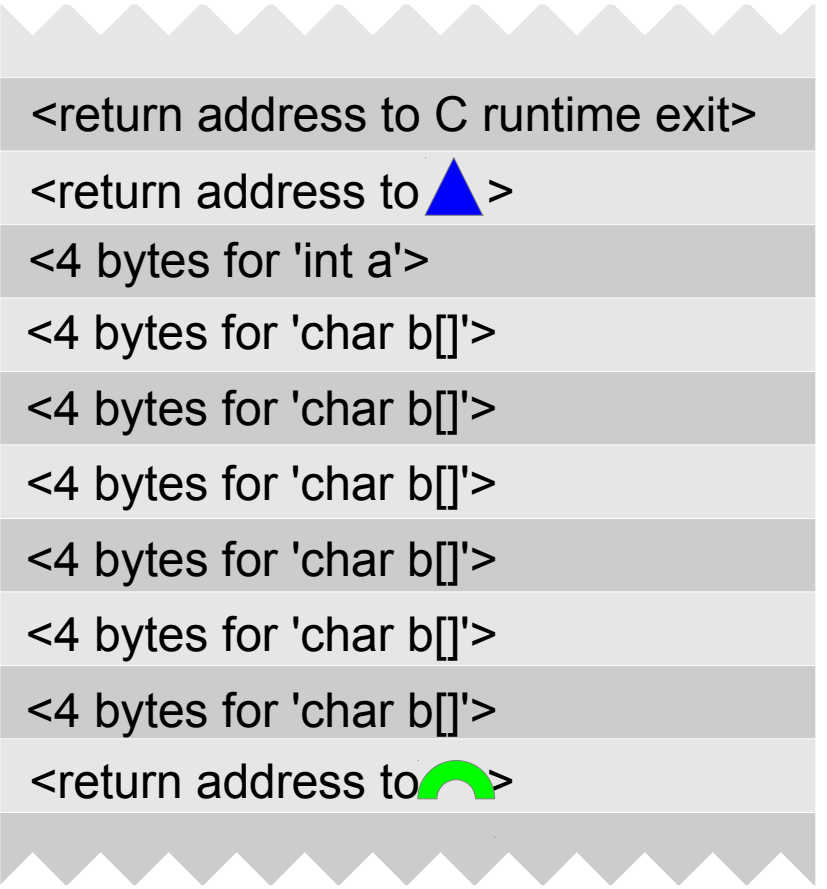
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```

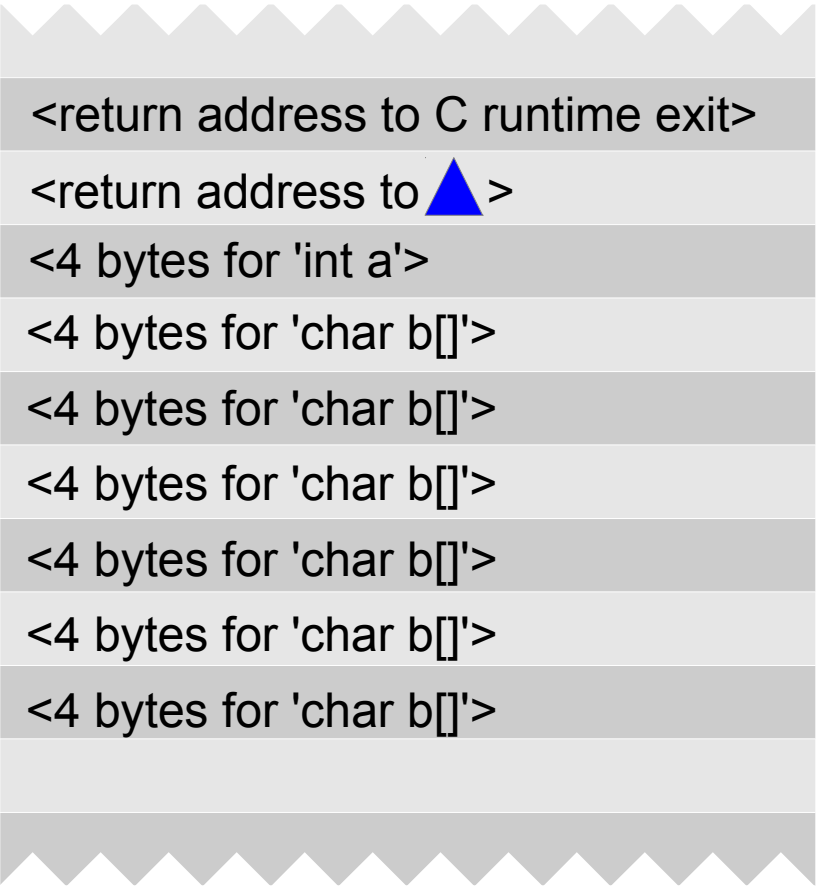
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



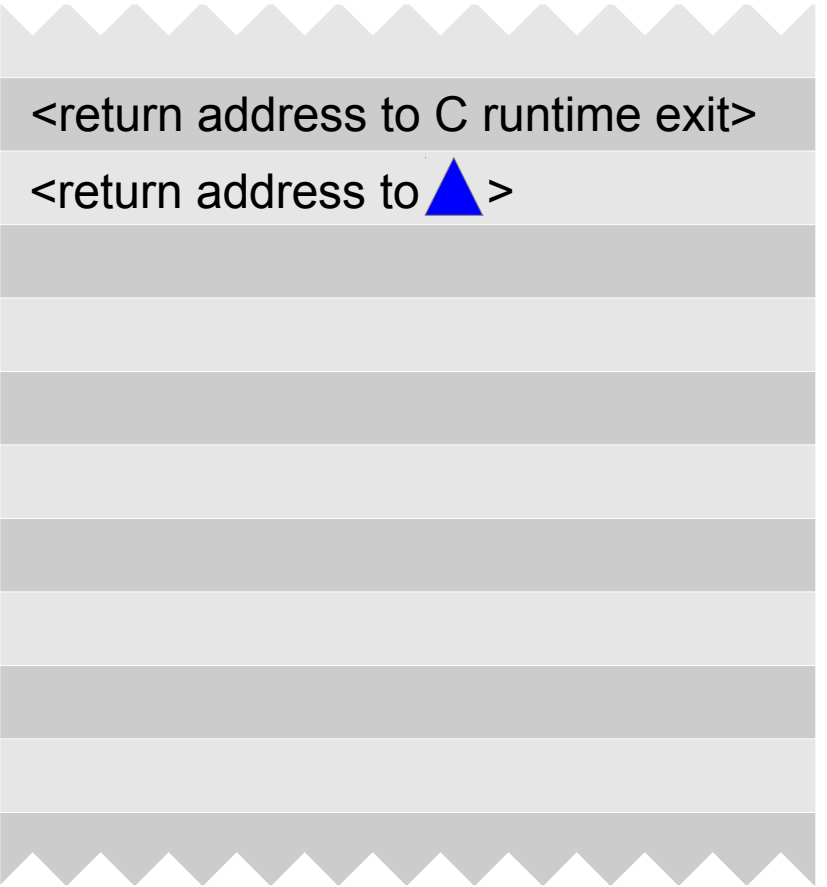
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



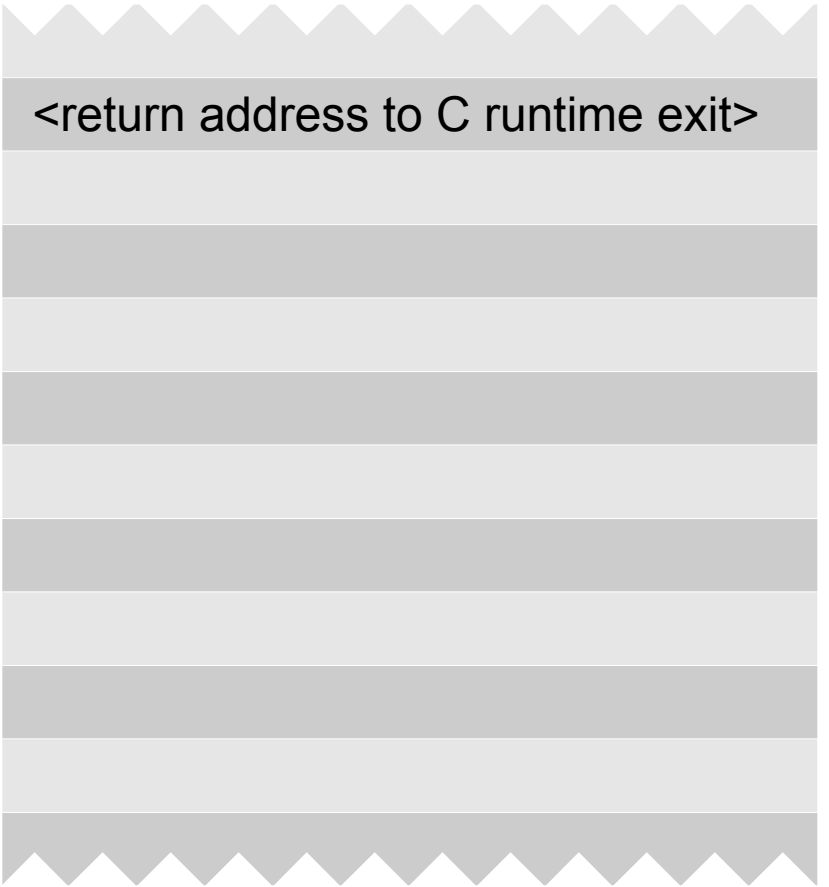
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



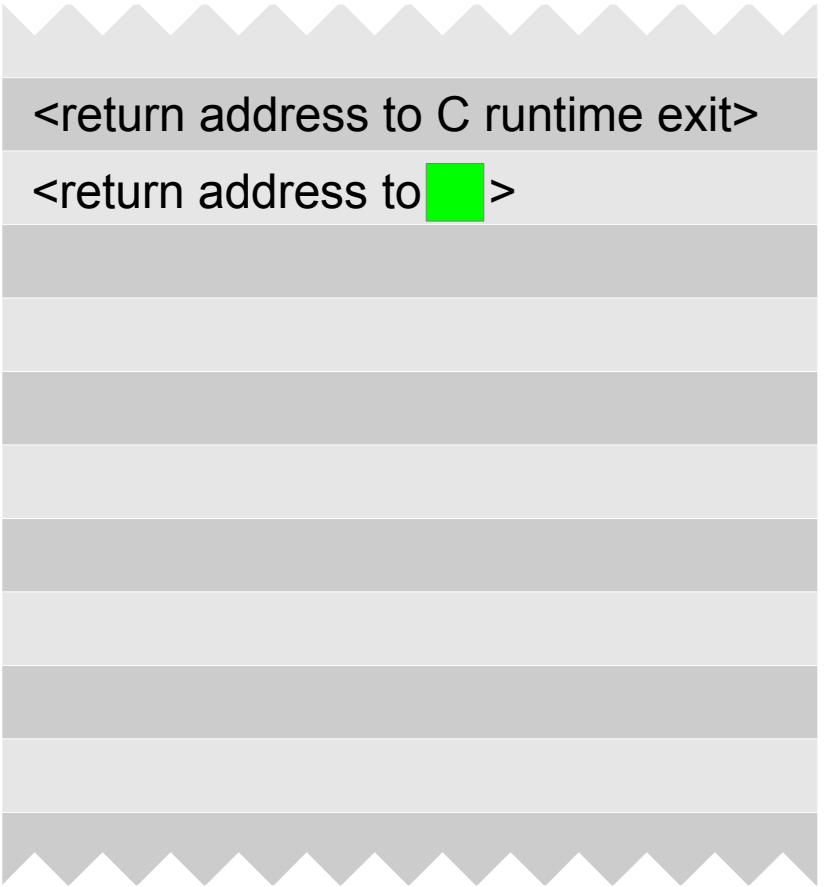
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



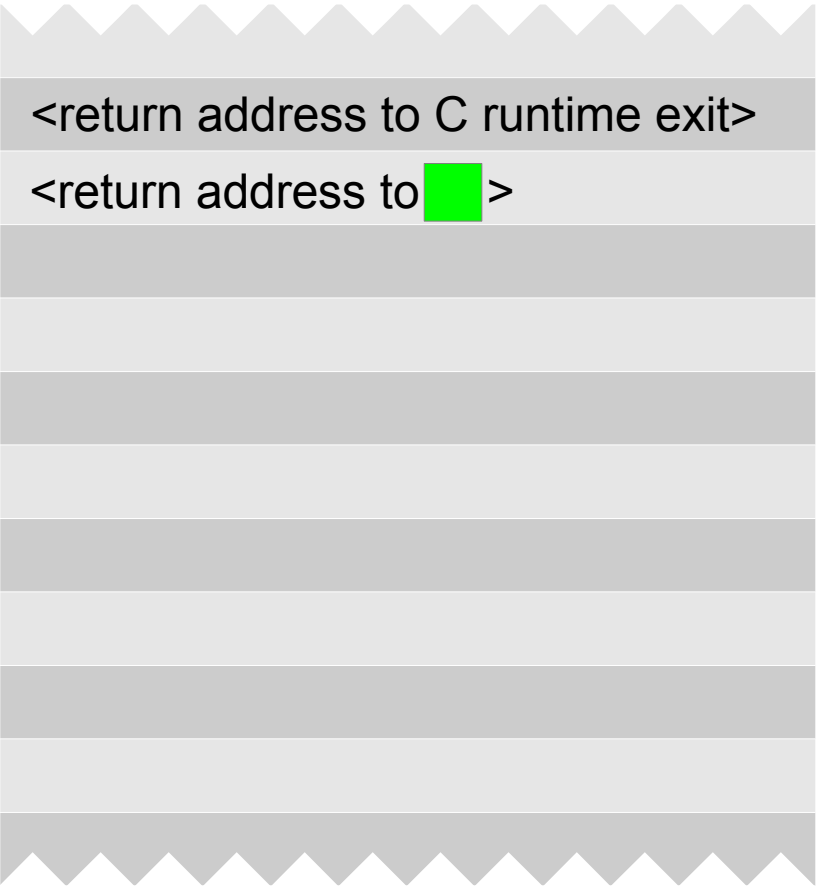
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



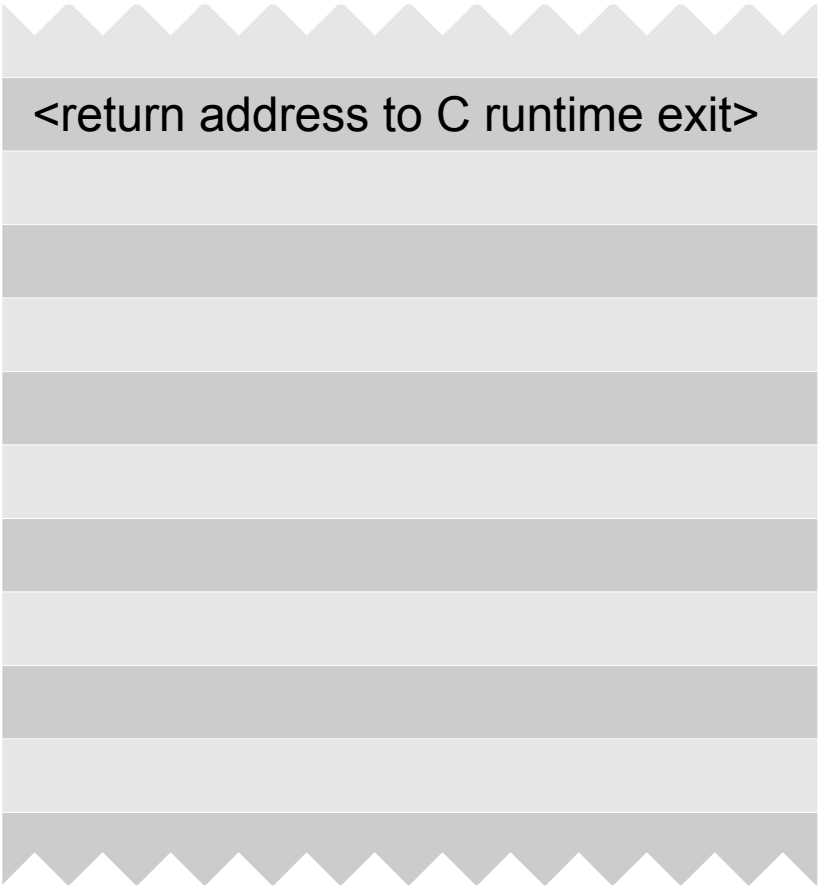
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



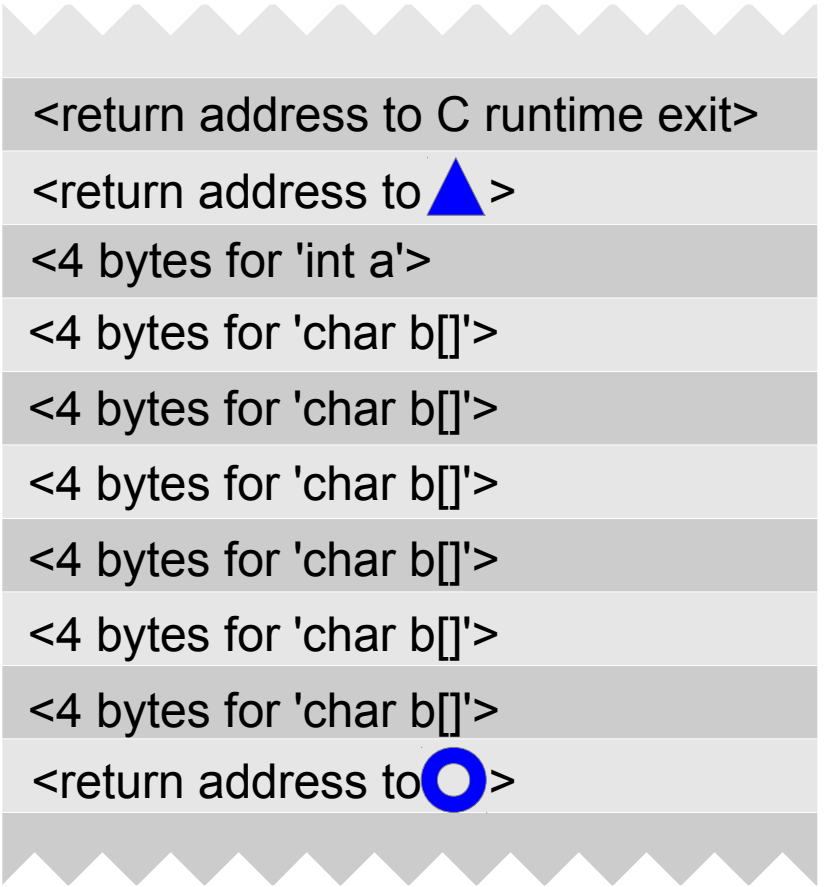
Memory Safety

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```

Part II:
Code Injection

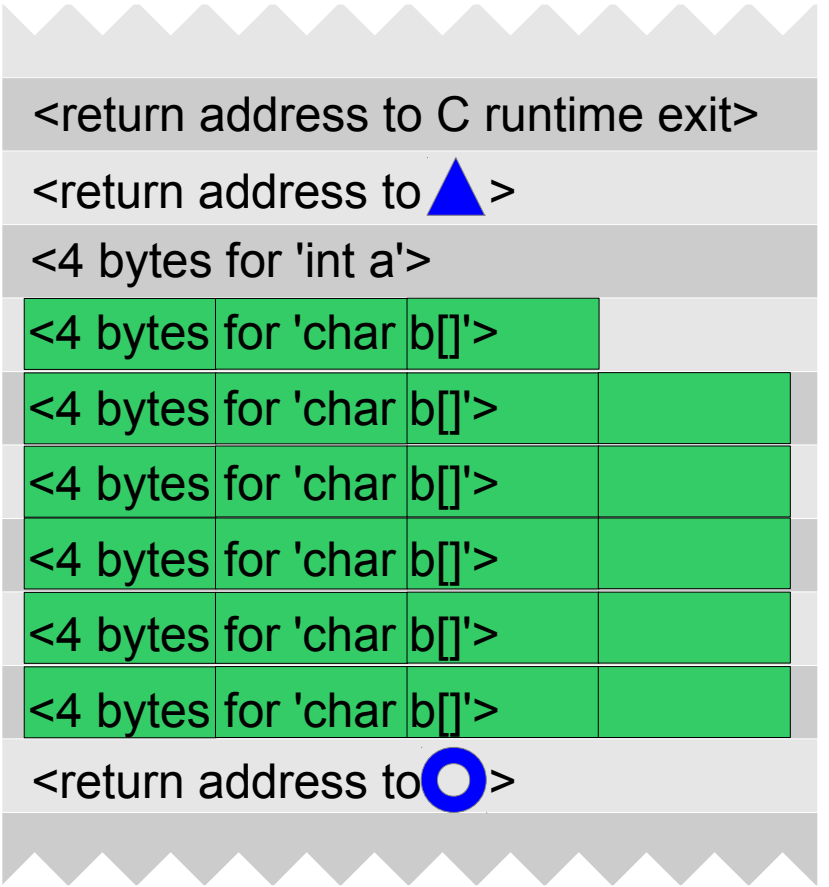
Smashing the Stack (1996)

```
#include <stdio.h>
int main() {
  ▲ foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
  ○ gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



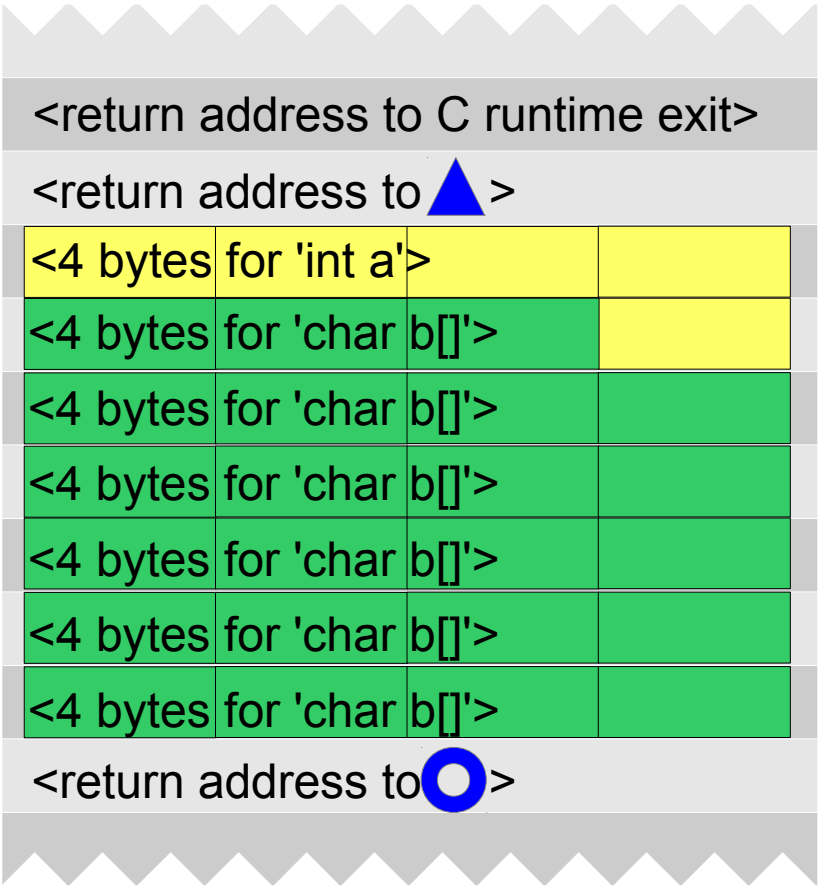
Smashing the Stack (1996)

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



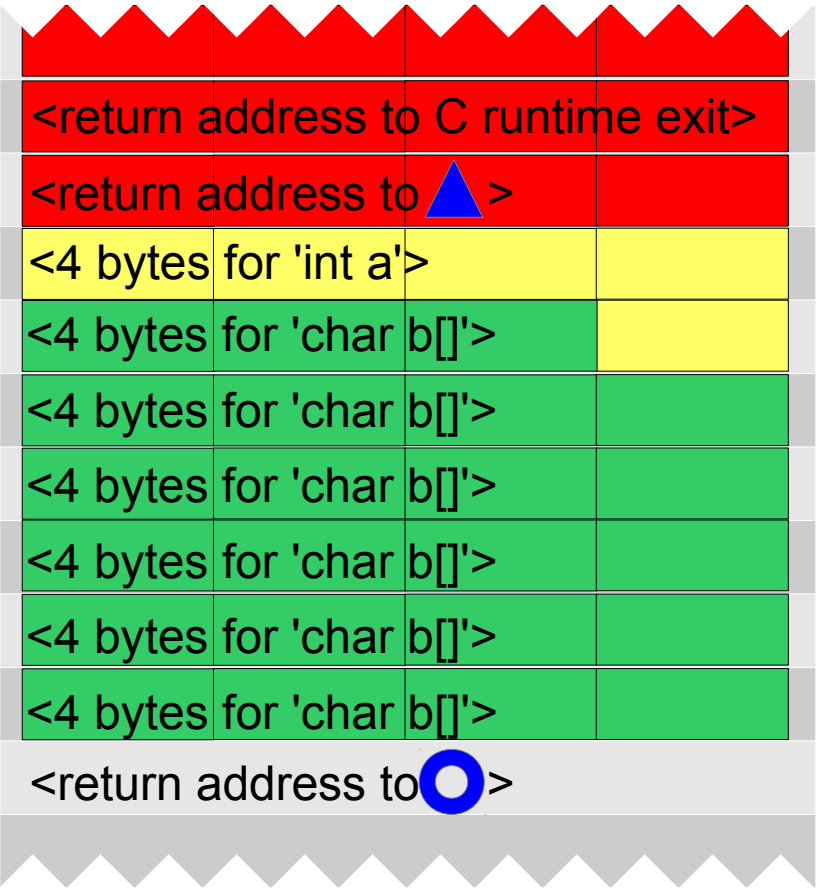
Smashing the Stack (1996)

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



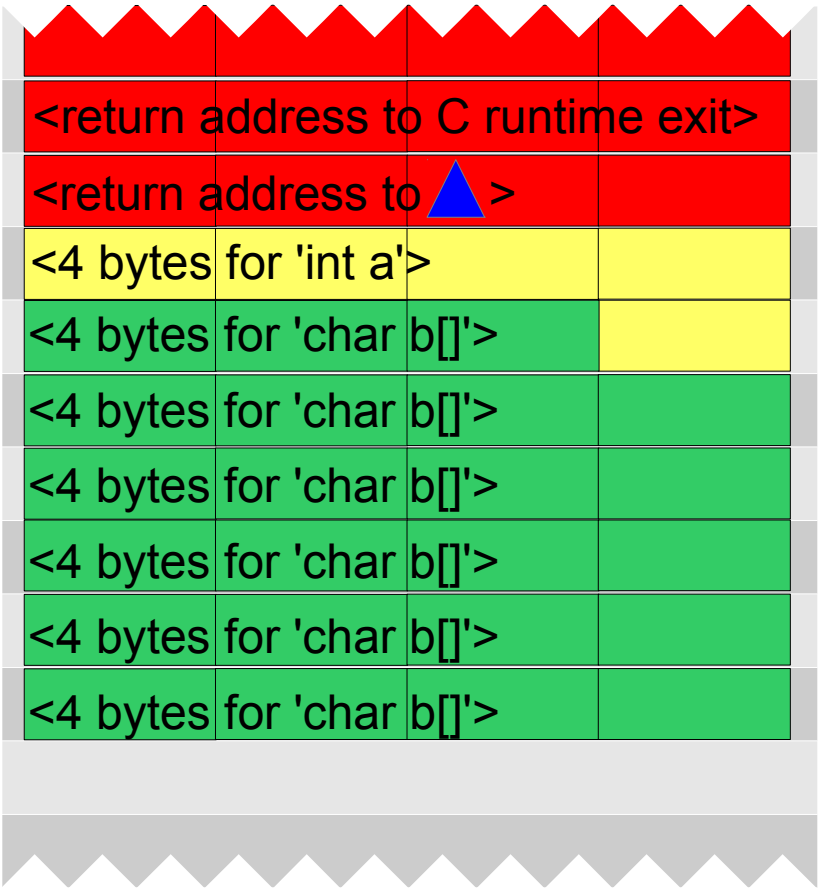
Smashing the Stack (1996)

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



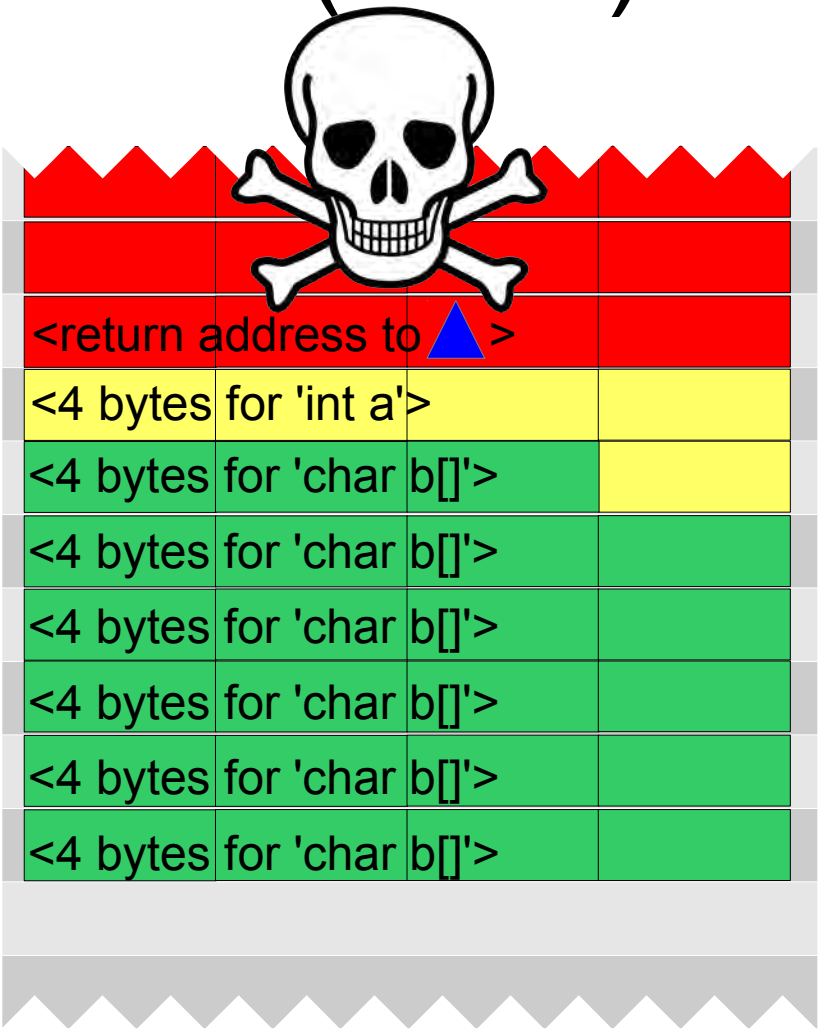
Smashing the Stack (1996)

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



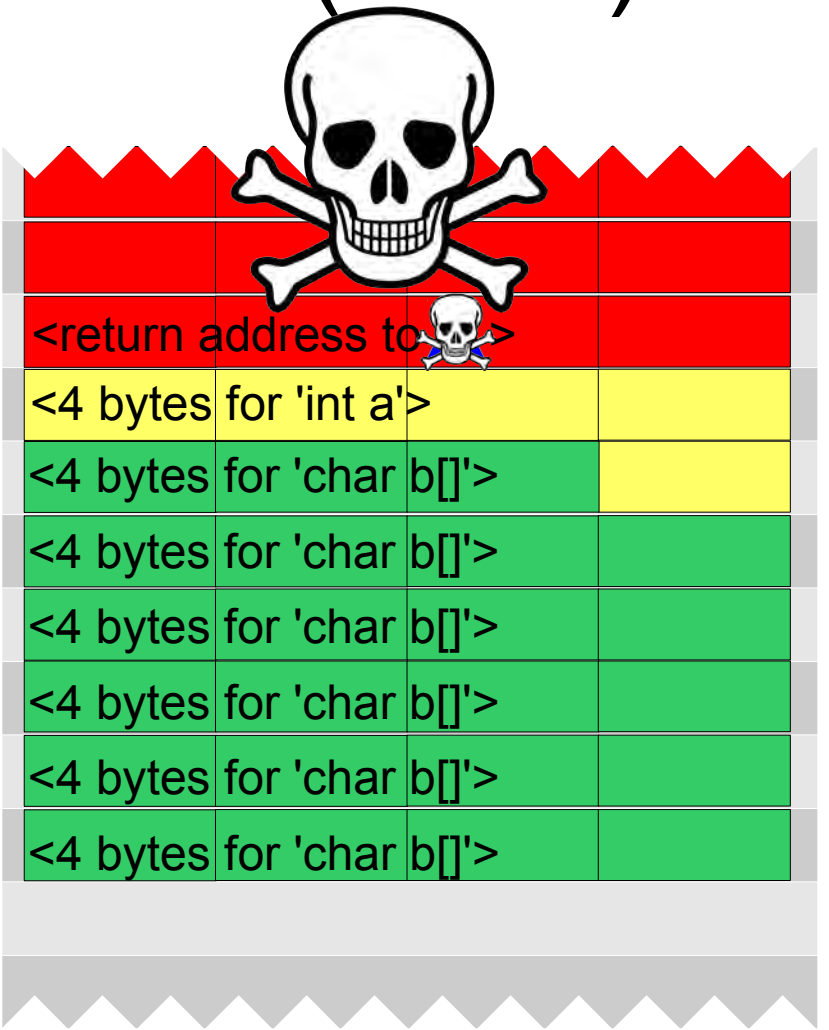
Smashing the Stack (1996)

```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



Smashing the Stack (1996)

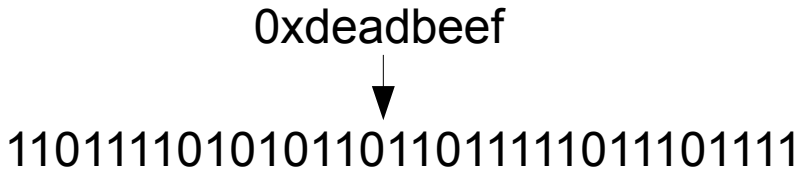
```
#include <stdio.h>
int main() {
    foo();
    bar(11, 12);
    return 0;
}
void foo() {
    int a;
    char b[23];
    gets(b);
    printf("Hey %s!\n",b);
}
int bar(int x, int y) {
    return x + y;
}
```



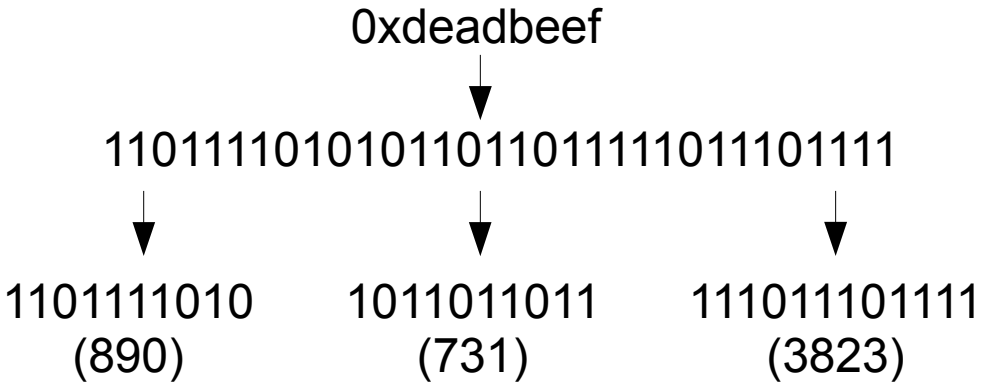
Paging/Virtual Memory

0xdeadbeef

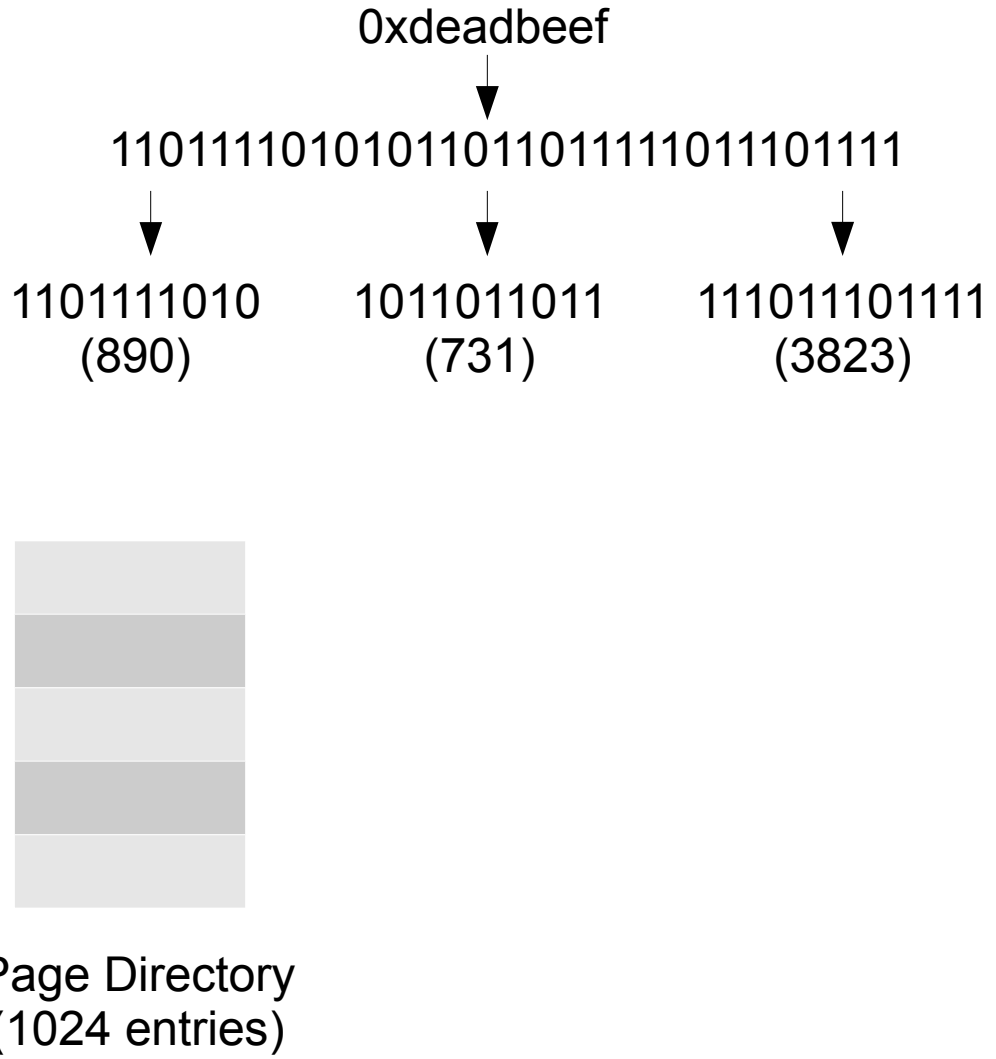
Paging/Virtual Memory



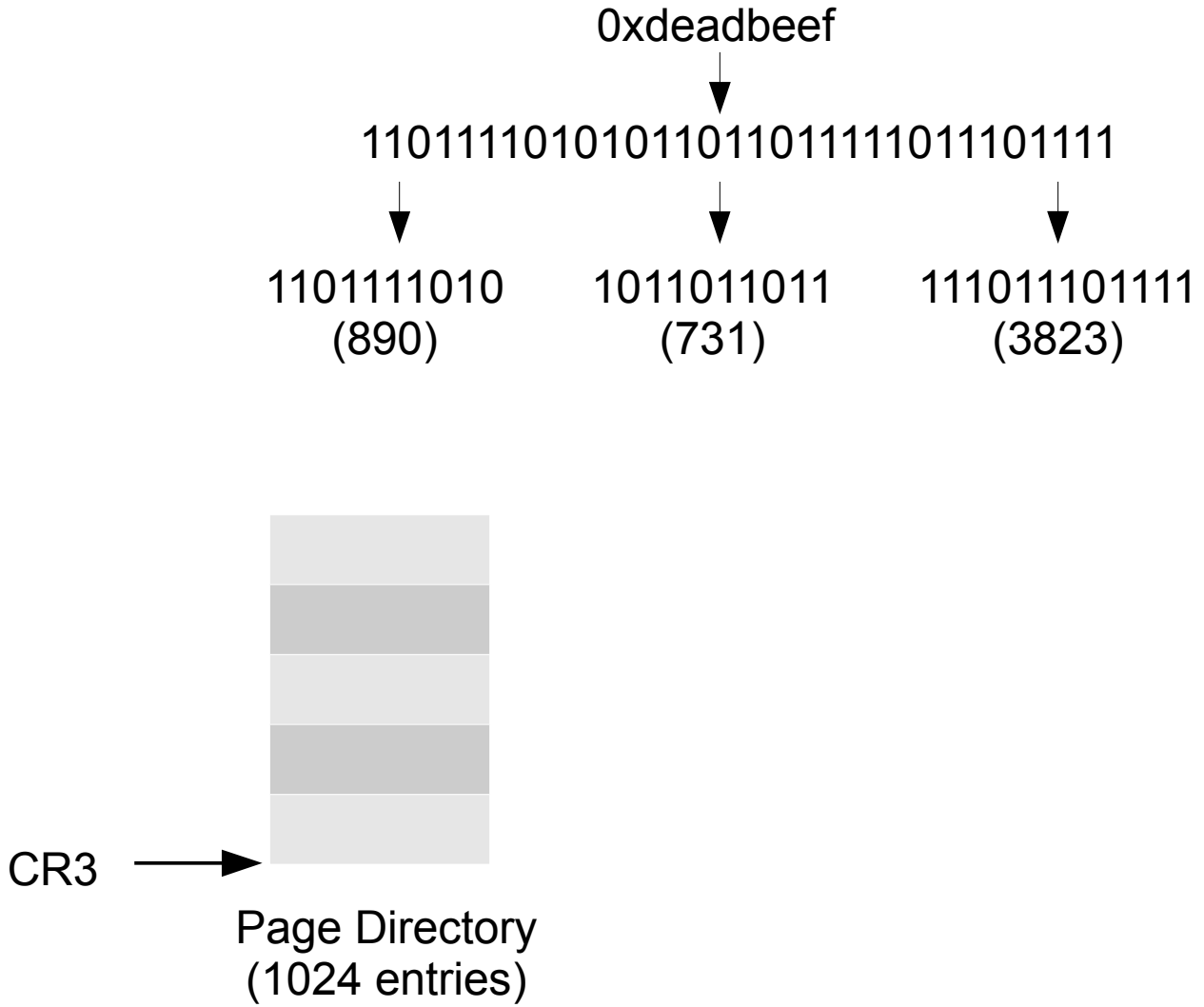
Paging/Virtual Memory



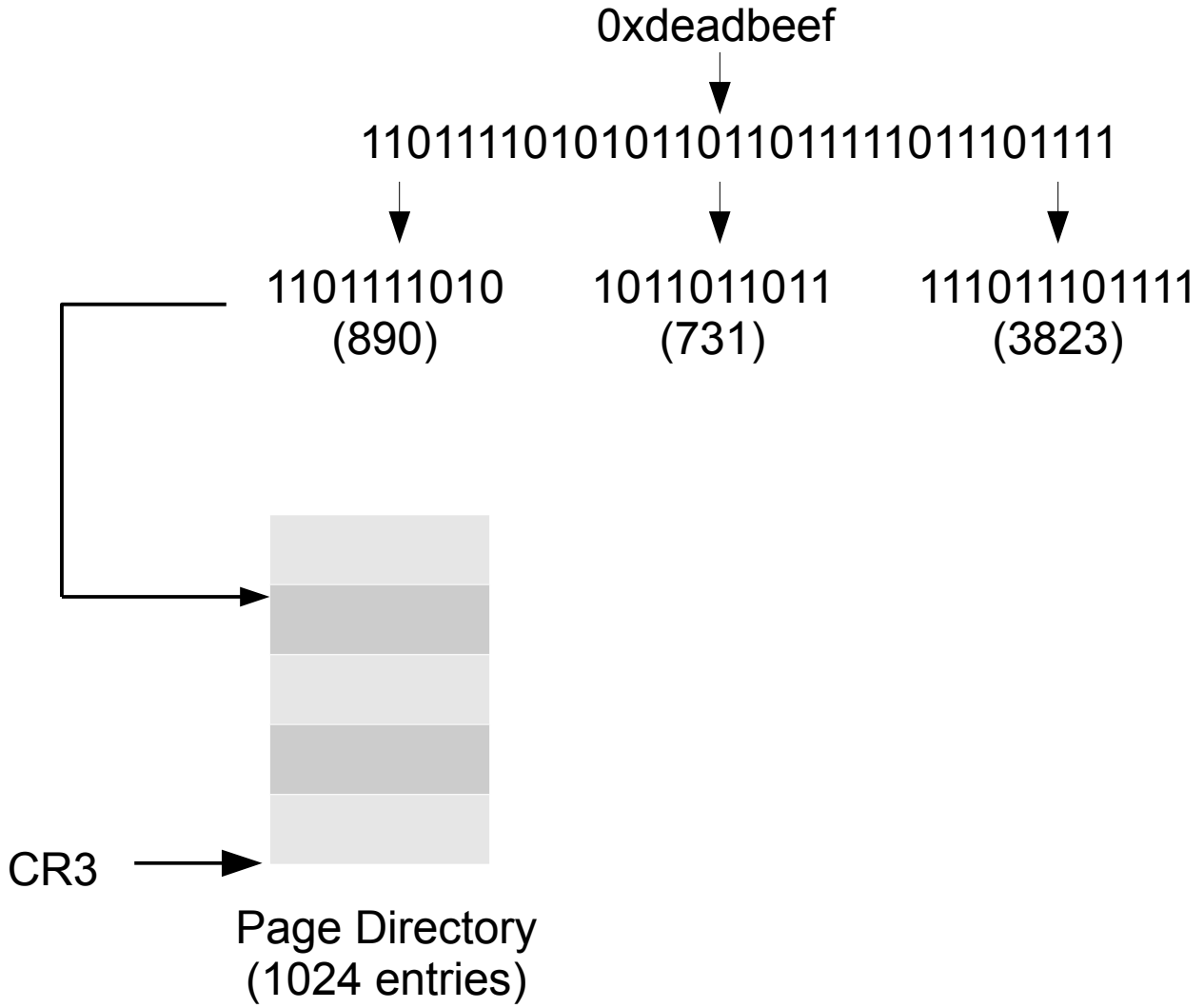
Paging/Virtual Memory



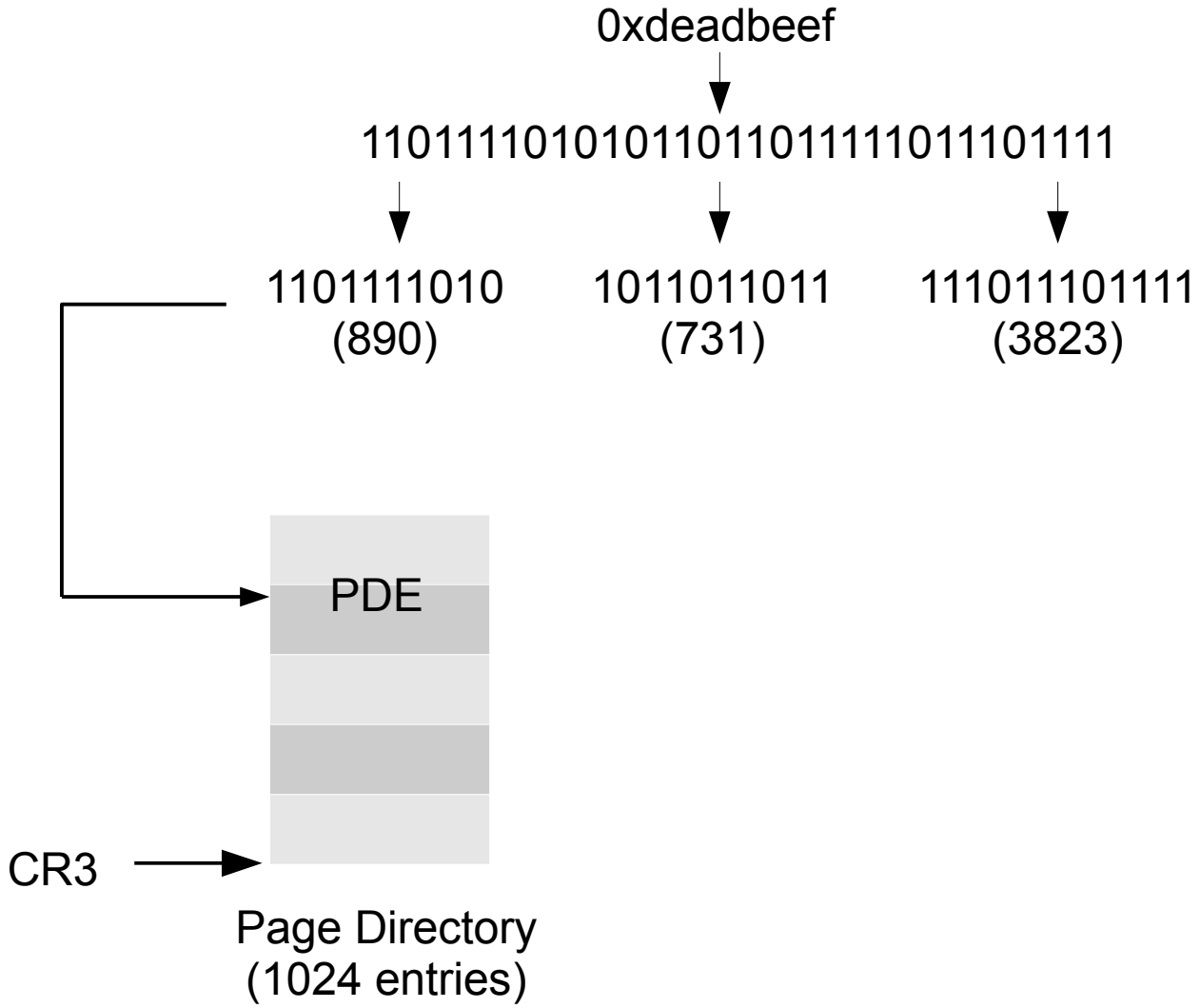
Paging/Virtual Memory



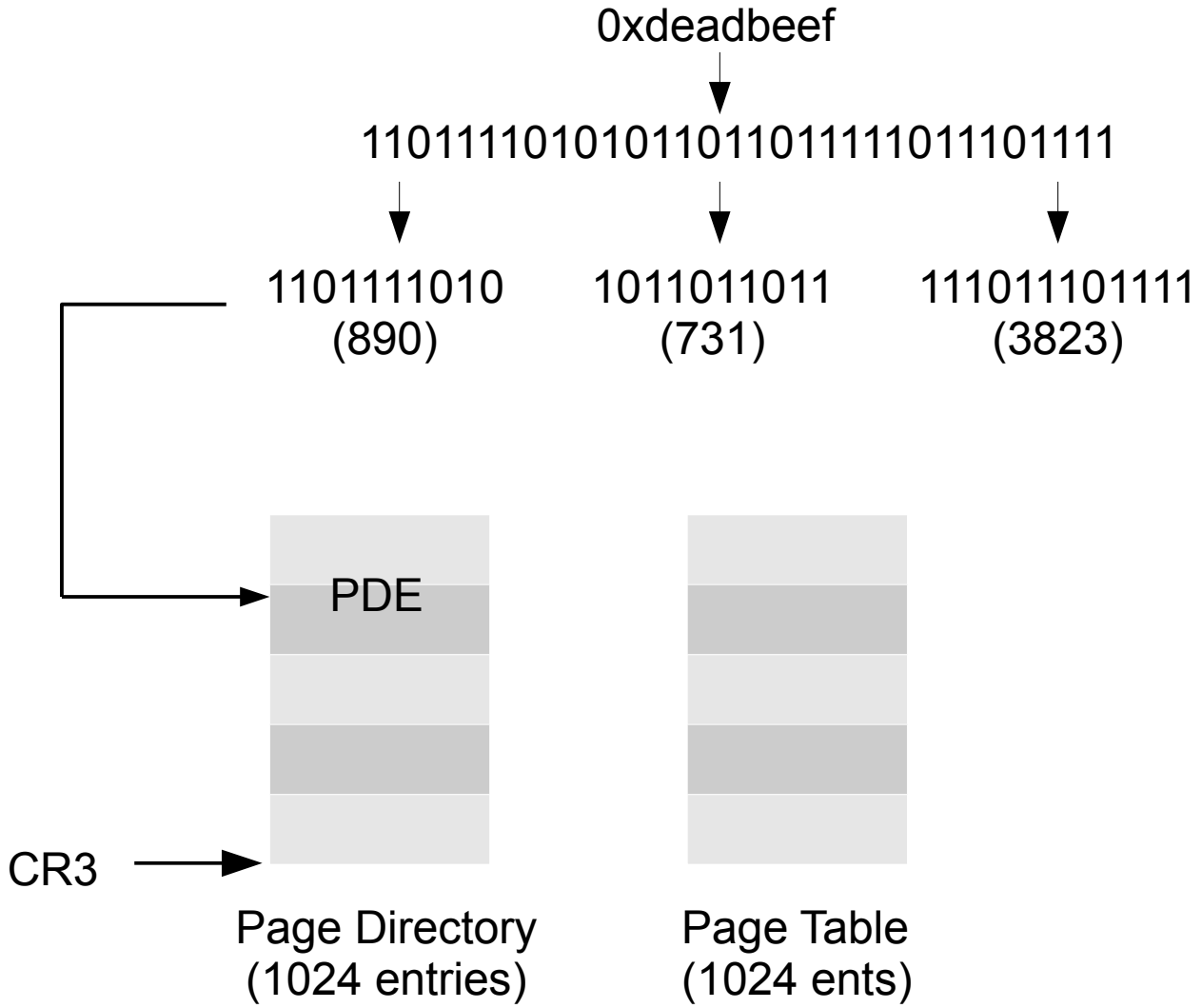
Paging/Virtual Memory



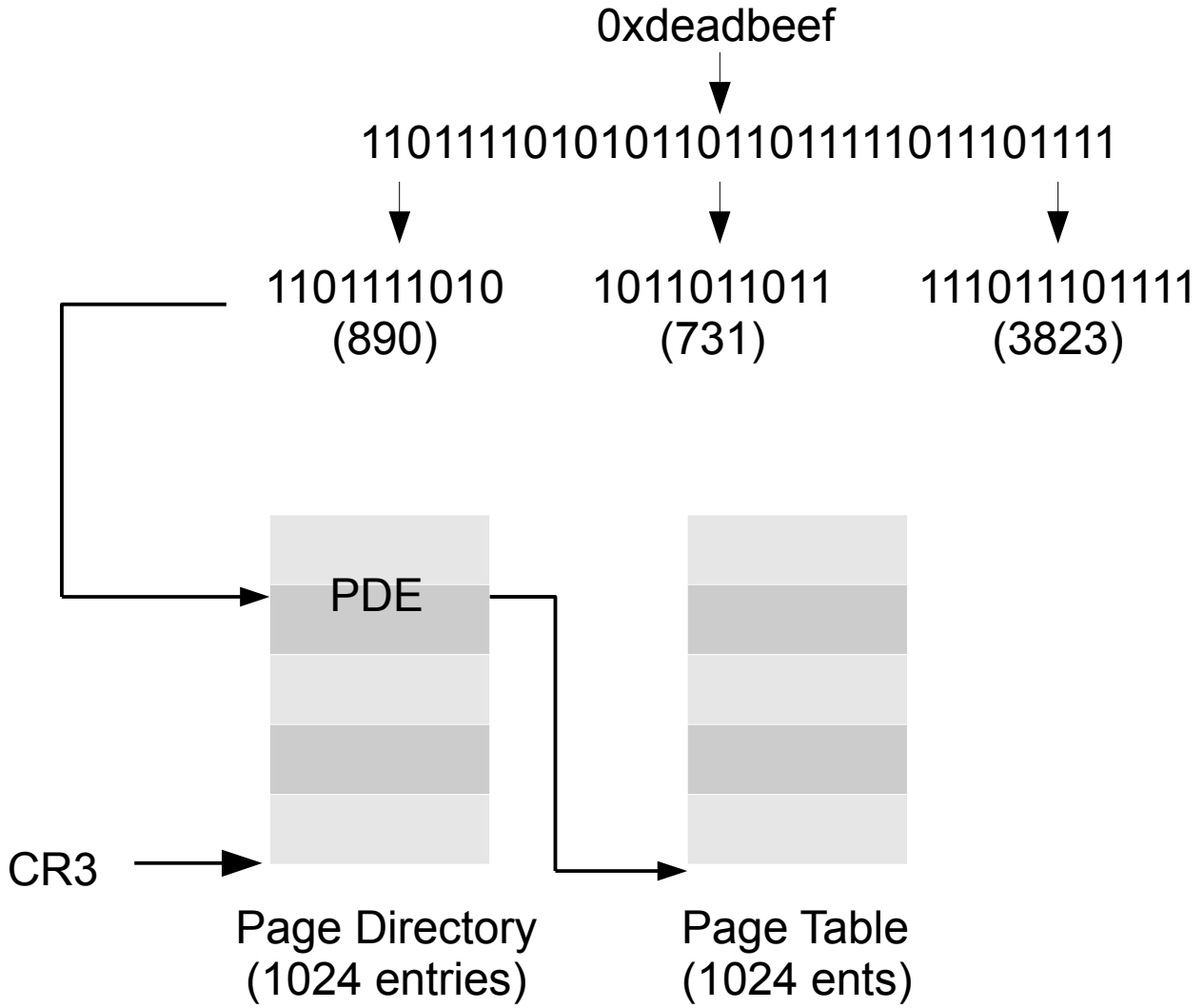
Paging/Virtual Memory



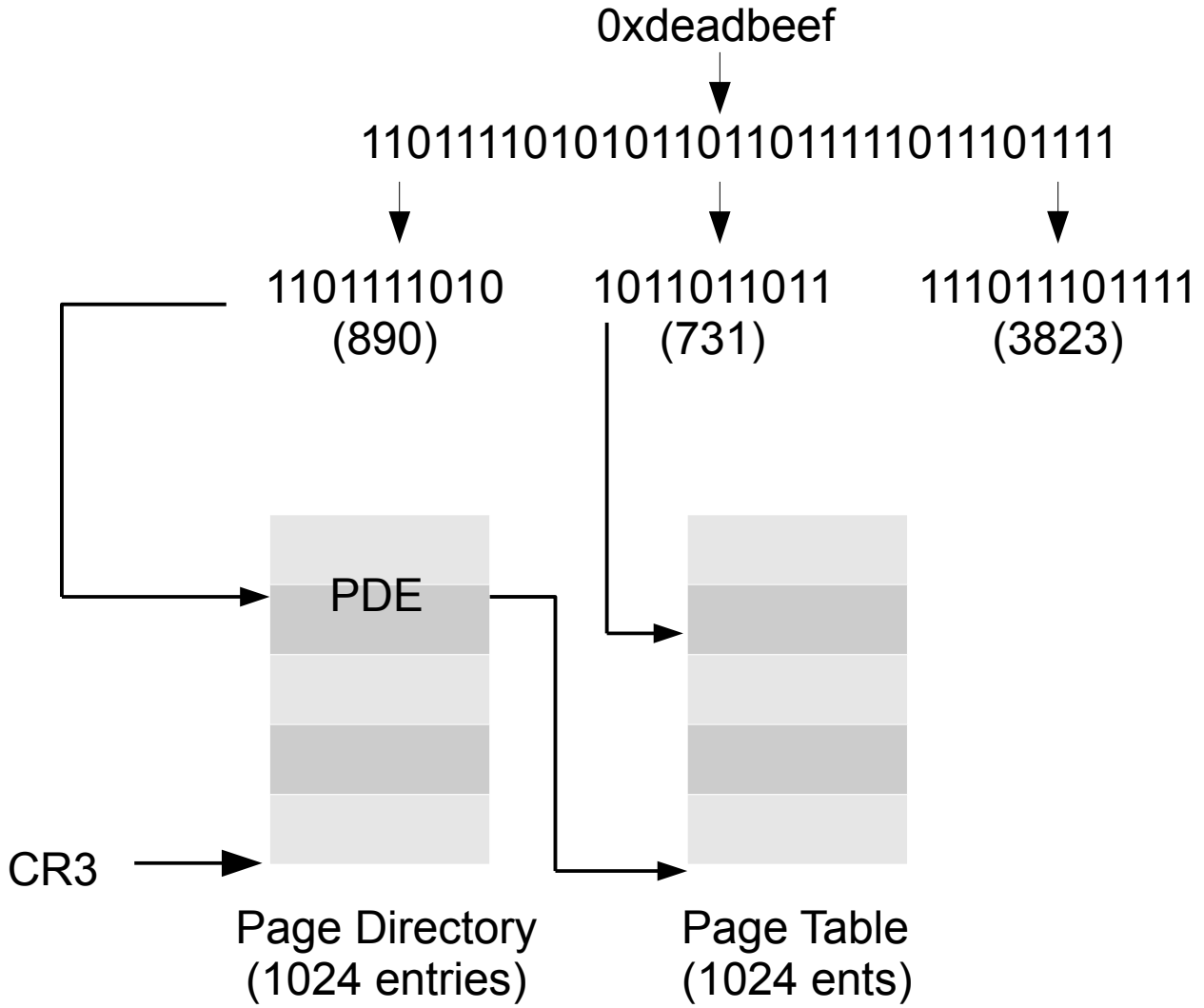
Paging/Virtual Memory



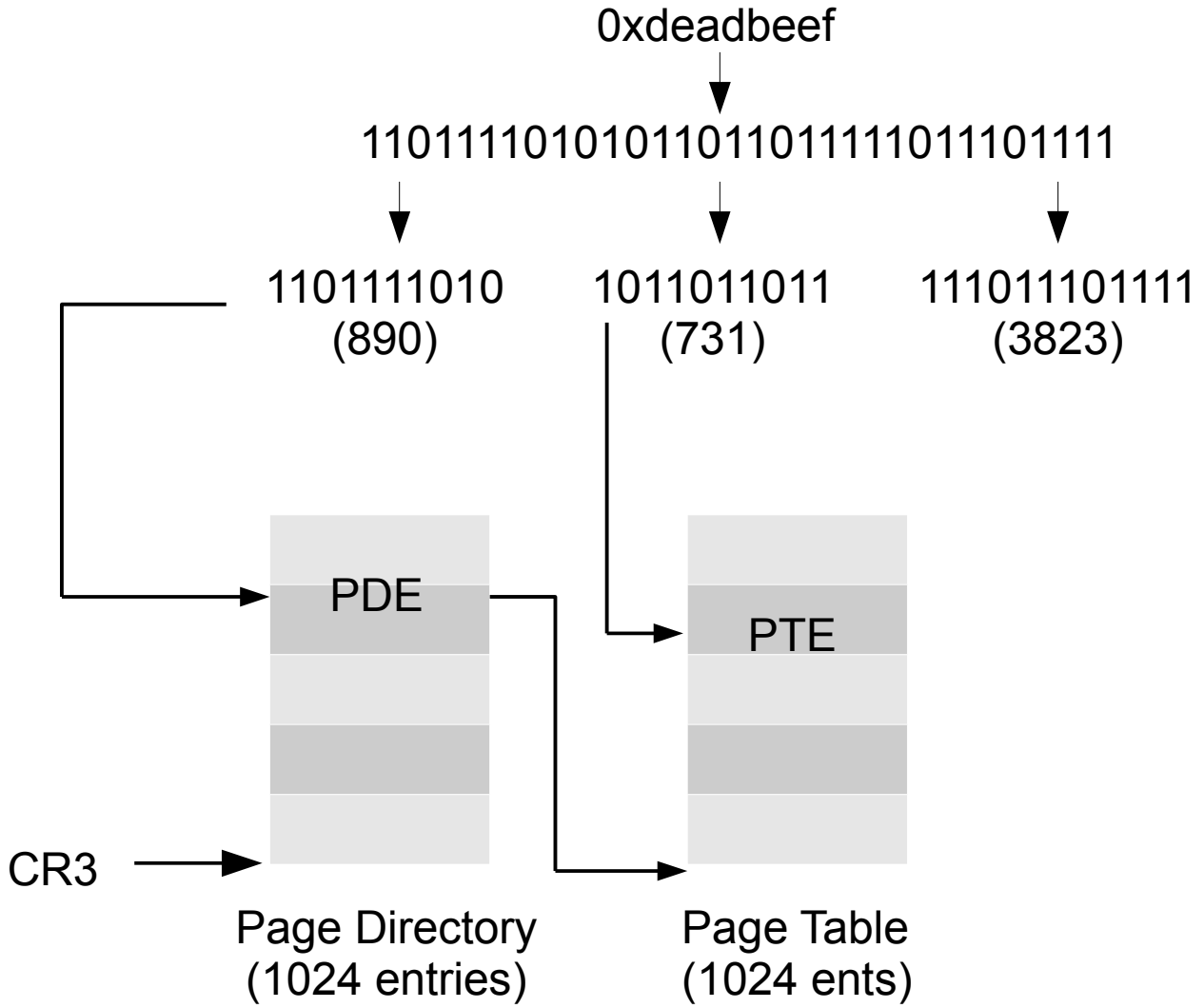
Paging/Virtual Memory



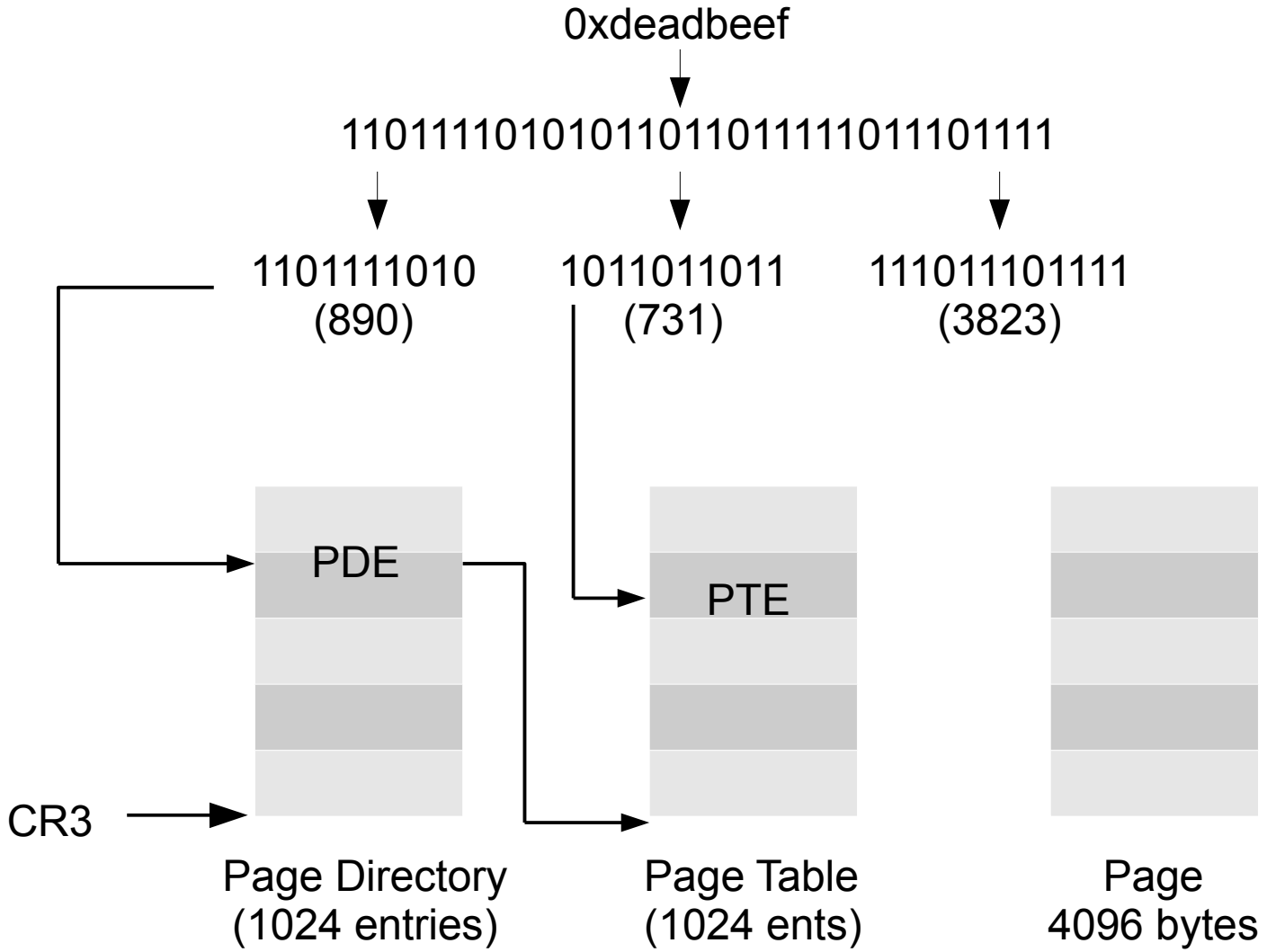
Paging/Virtual Memory



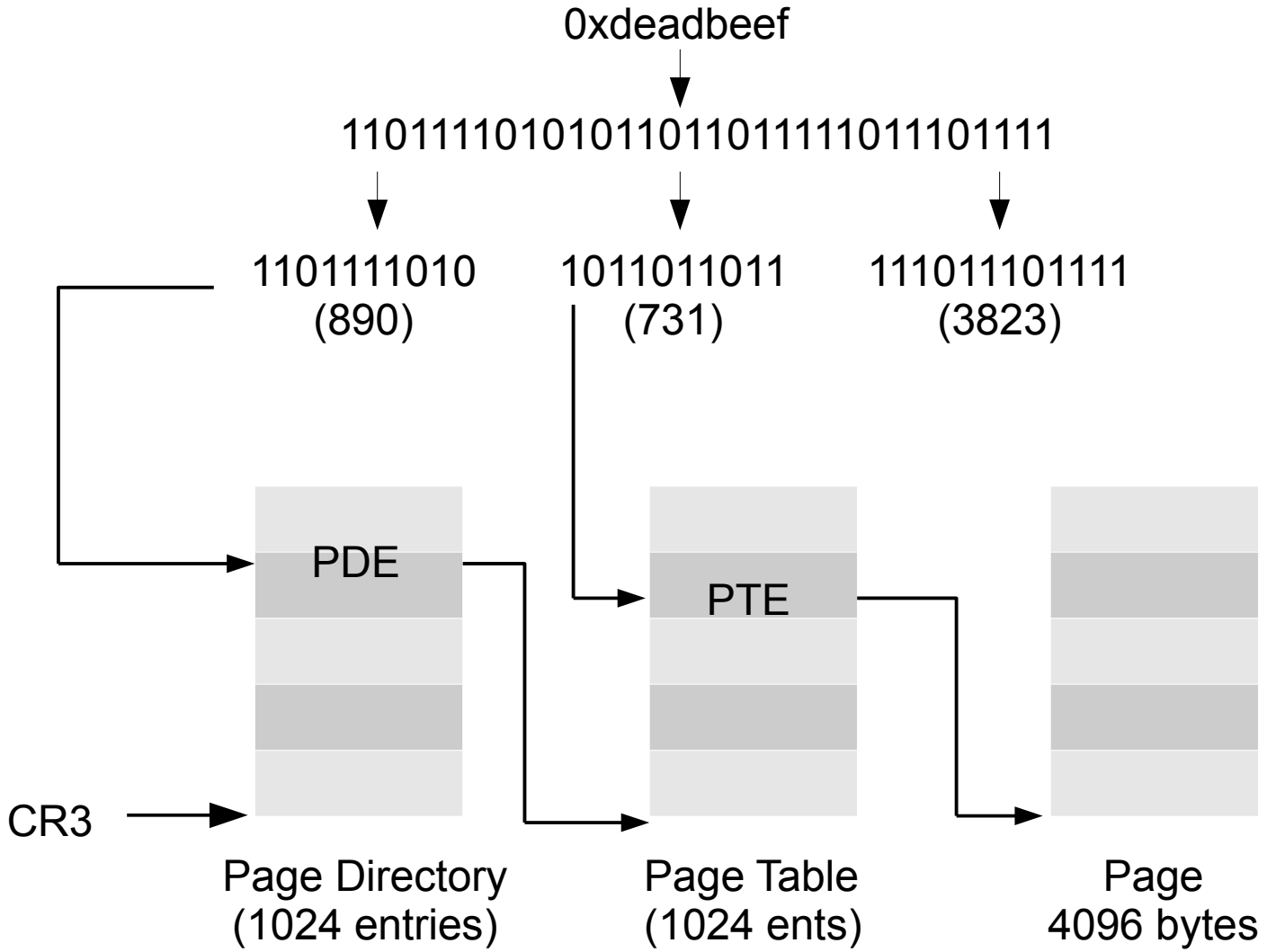
Paging/Virtual Memory



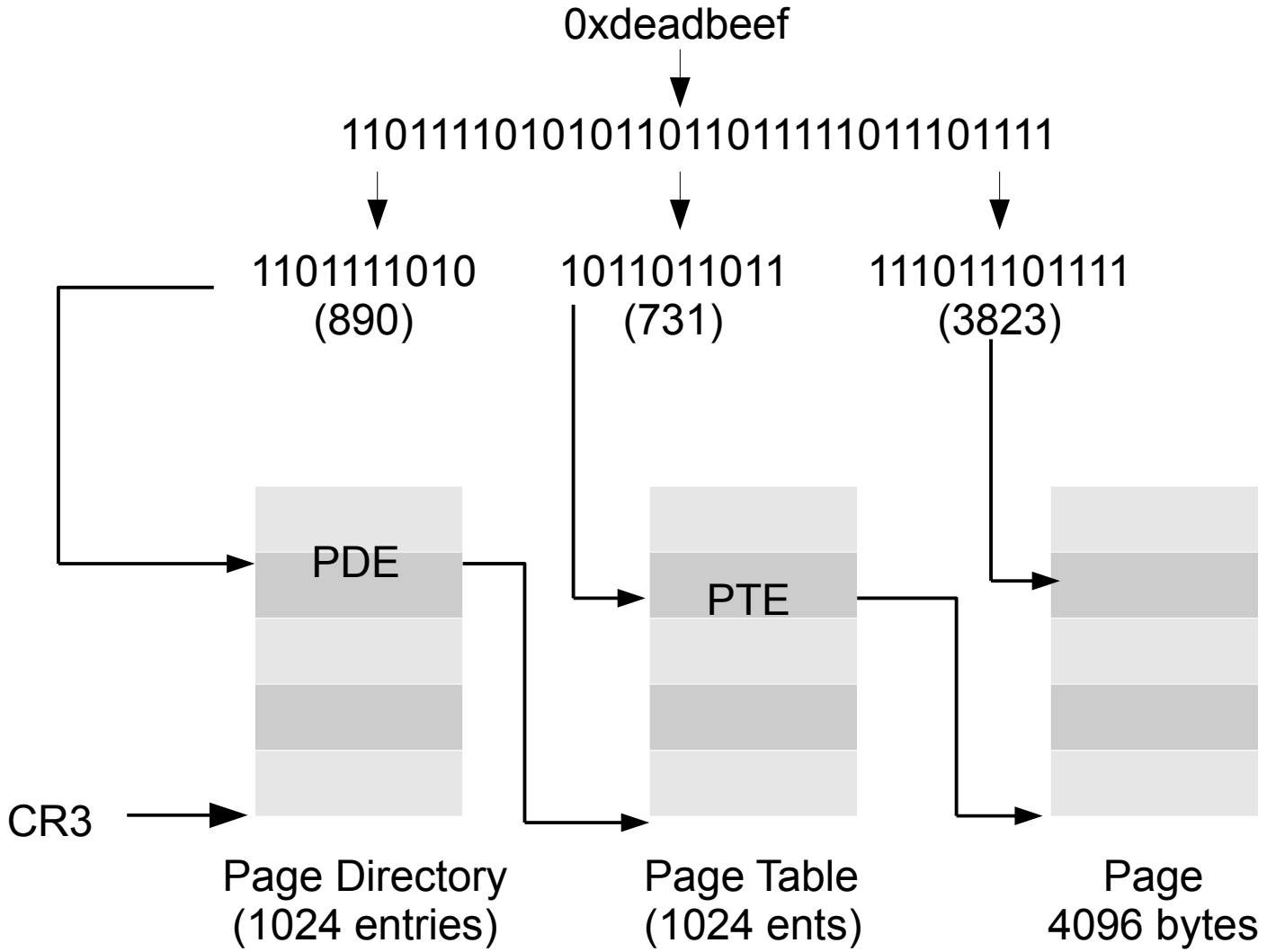
Paging/Virtual Memory



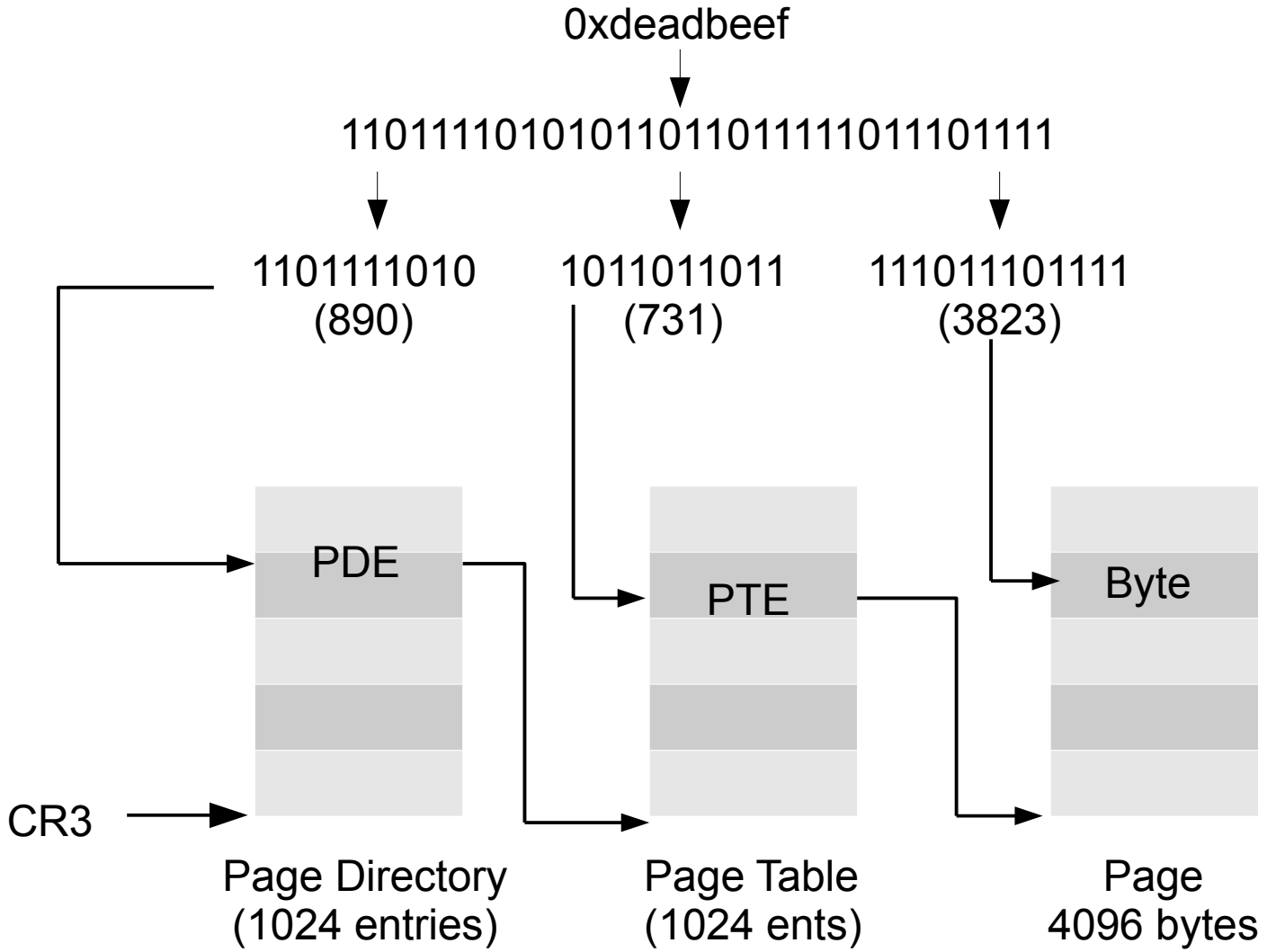
Paging/Virtual Memory



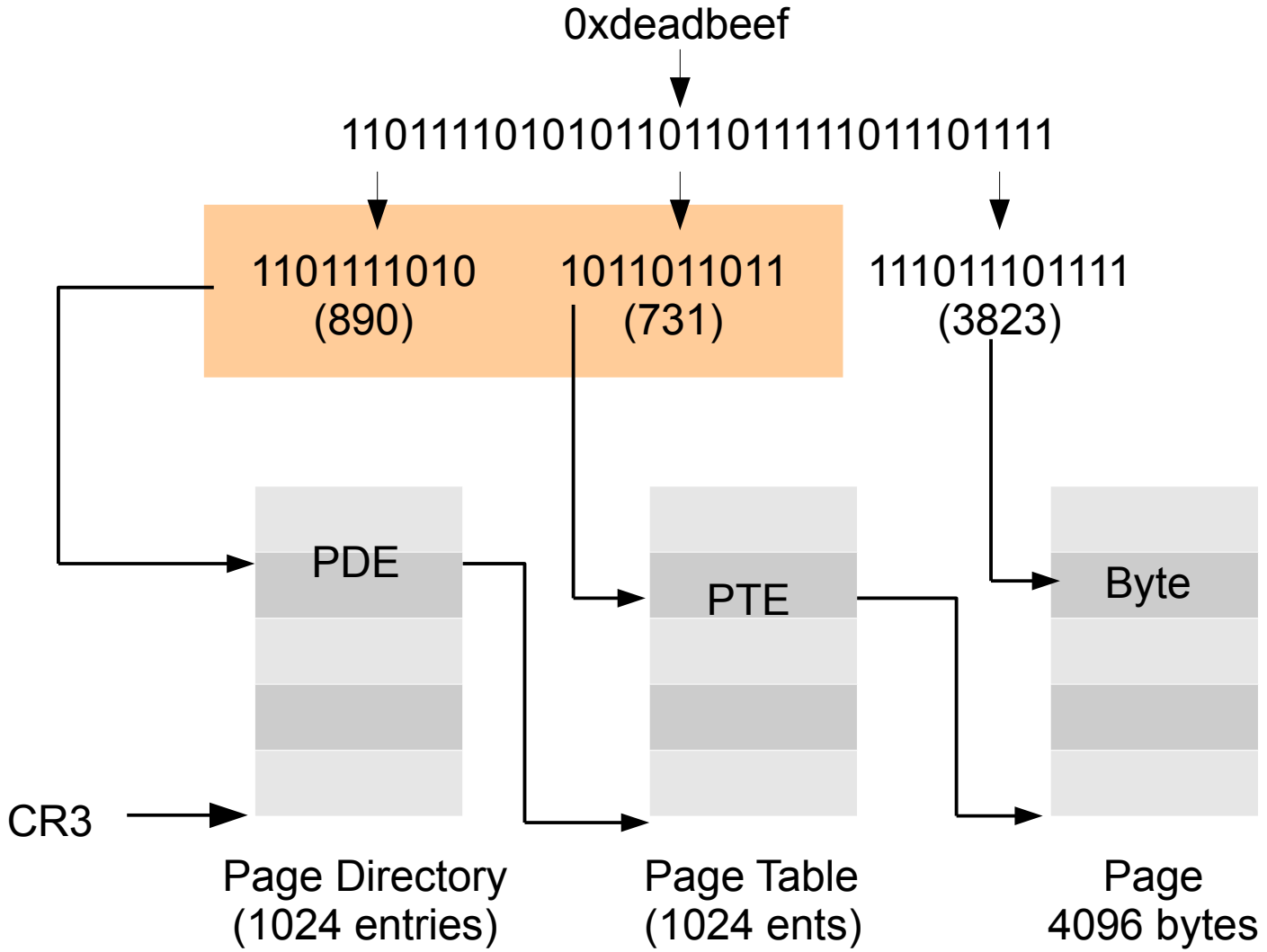
Paging/Virtual Memory



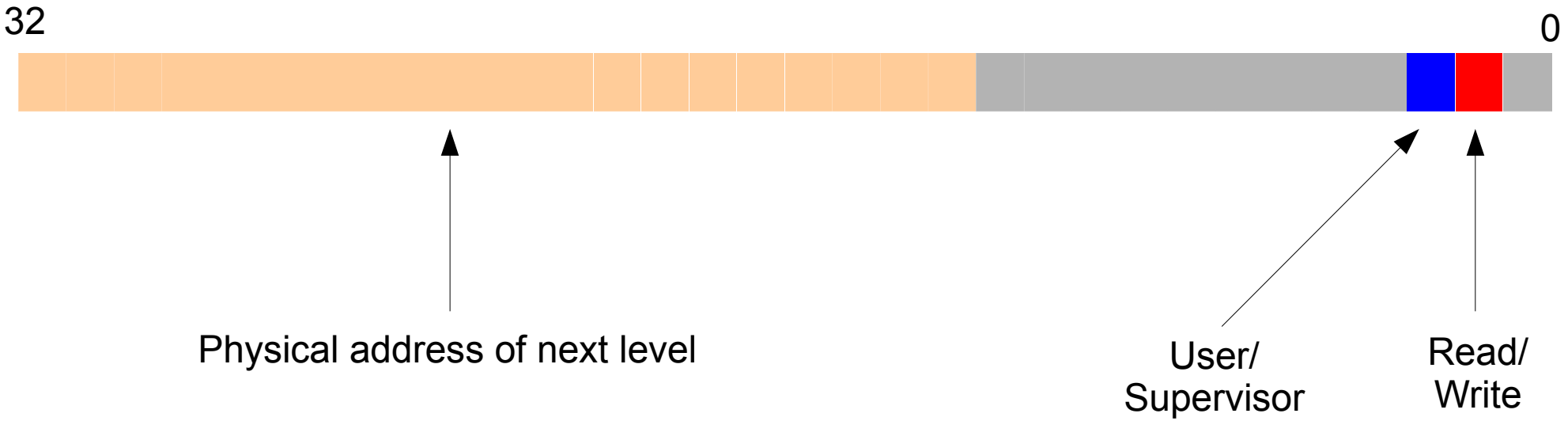
Paging/Virtual Memory



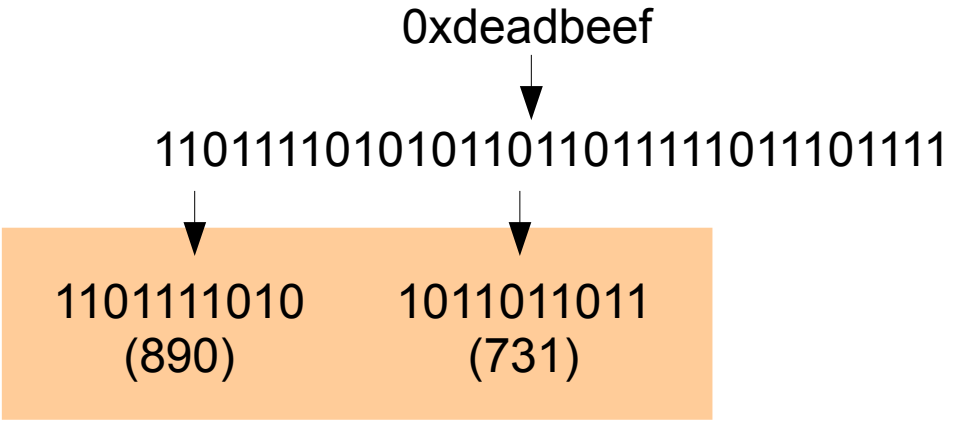
Paging/Virtual Memory



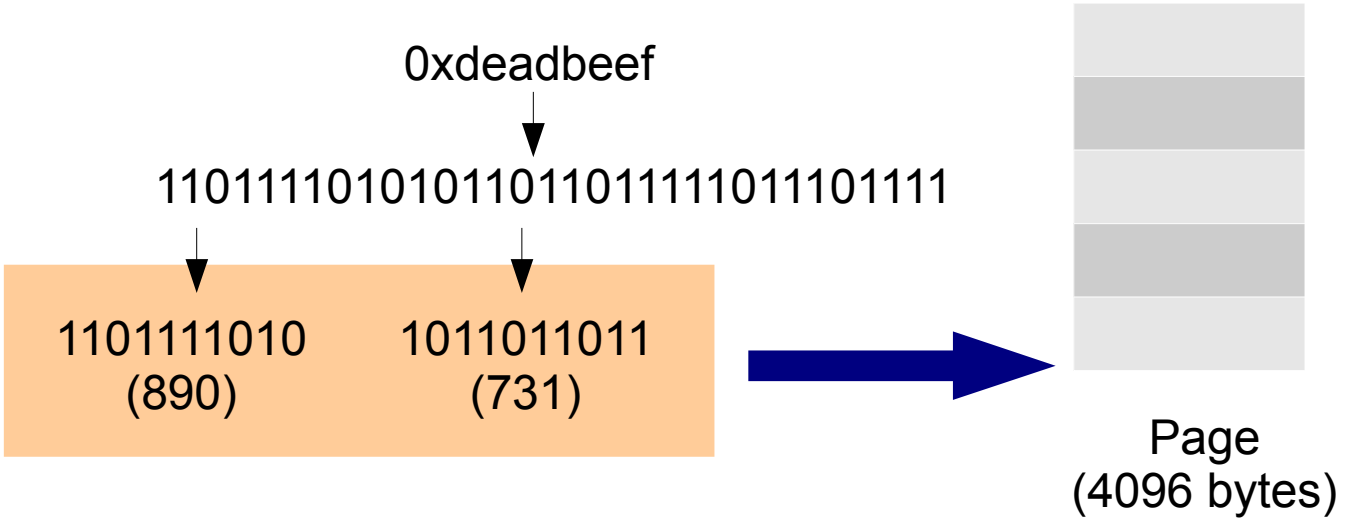
Page Table Entries



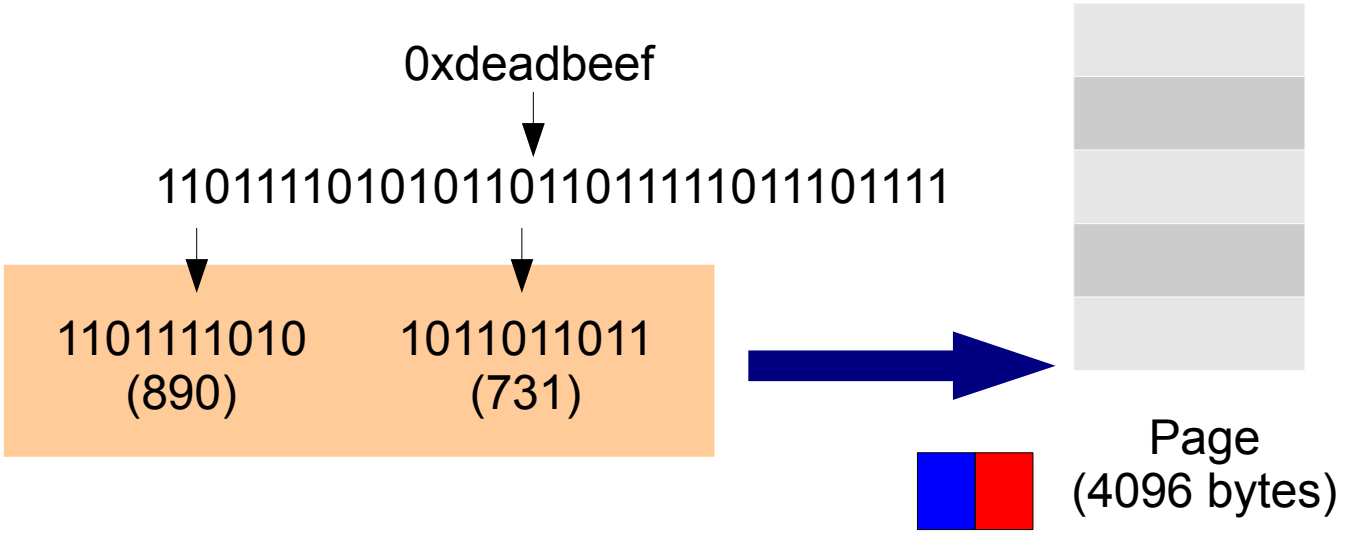
Paging made fast: TLB



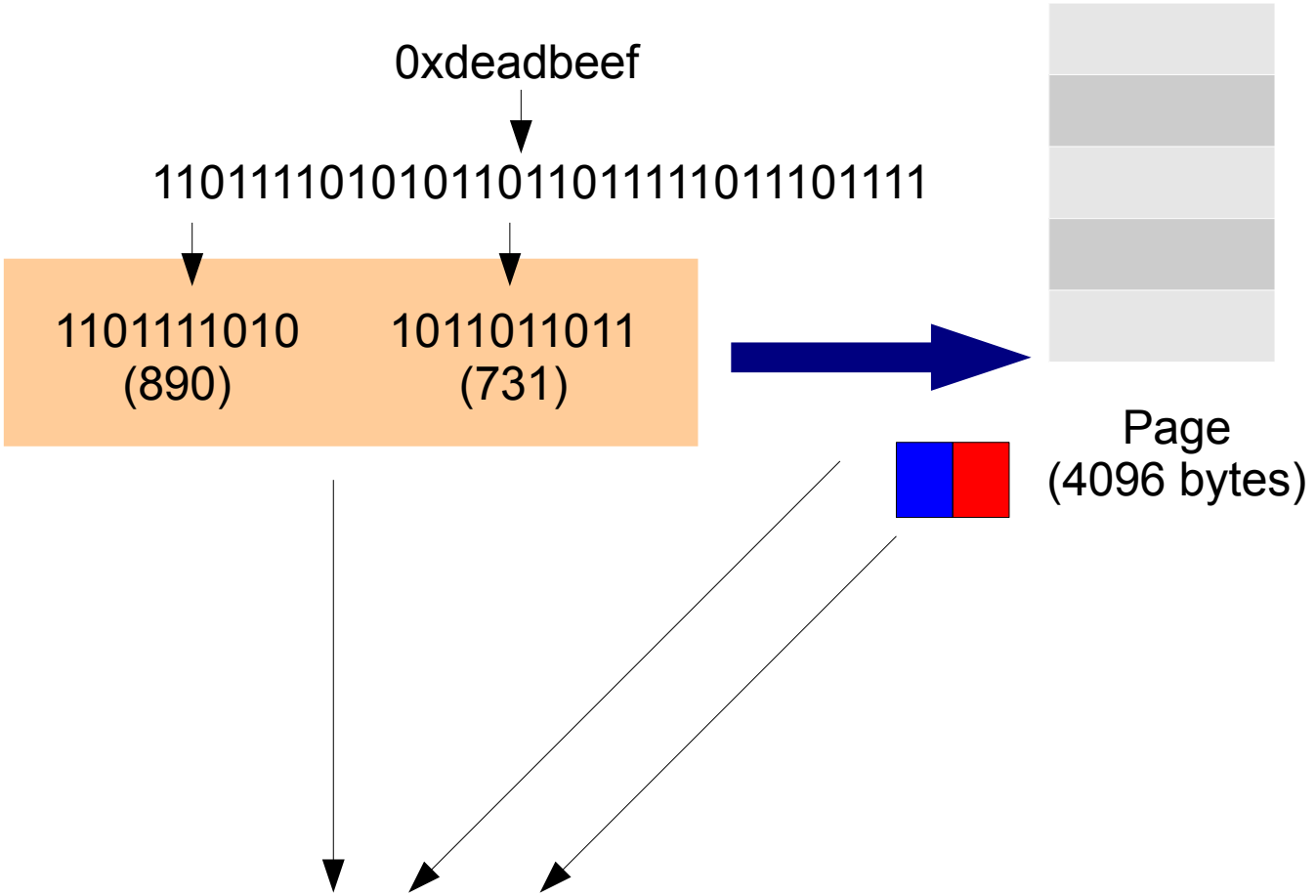
Paging made fast: TLB



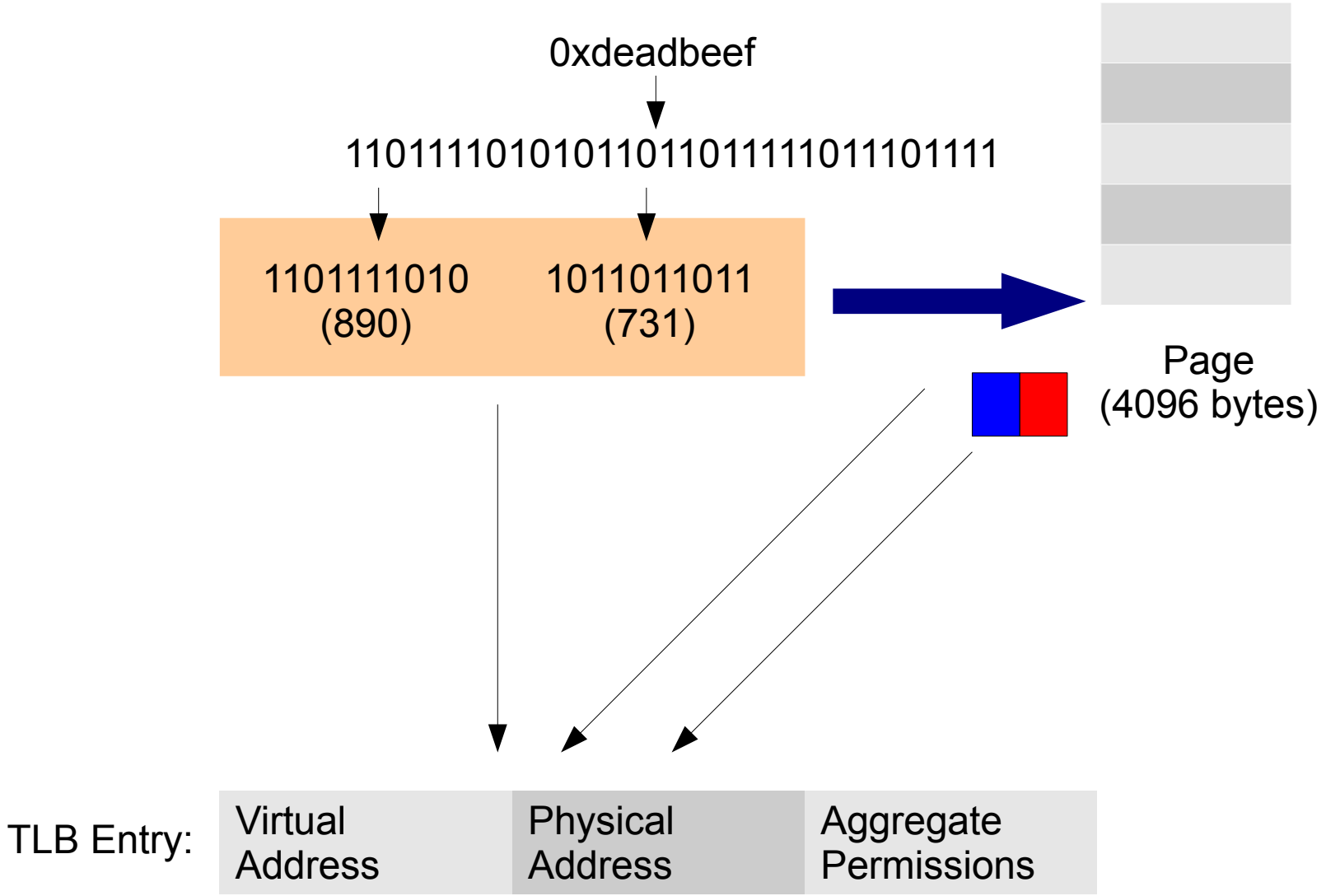
Paging made fast: TLB



Paging made fast: TLB



Paging made fast: TLB



PaX PAGEEXEC (2000)

User/
Supervisor:
Emulates
Non-Exec

Instruction TLB:

Virtual Addr	Physical Addr	Permission

Data TLB:

Virtual Addr	Physical Addr	Permission

Instruction Pointer:

PaX Page Fault Strategy:

```
if (supervisor page &&
    IP on faulting page) {
    Terminate
} else {
    Set user page in PTE
    Prime Data TLB
    Set supervisor page in PTE
}
```

PaX PAGEEXEC (2000)

User/
Supervisor:
Emulates
Non-Exec

Instruction TLB:

Virtual Addr	Physical Addr	Permission

Data TLB:

Virtual Addr	Physical Addr	Permission

Instruction Pointer:

PaX Page Fault Strategy:

```
if (supervisor page &&
    IP on faulting page) {
    Terminate
} else {
    Set user page in PTE
    Prime Data TLB
    Set supervisor page in PTE
}
```

PaX PAGEEXEC (2000)

User/
Supervisor:
Emulates
Non-Exec

Instruction TLB:

Virtual Addr	Physical Addr	Permission

Data TLB:

Virtual Addr	Physical Addr	Permission

Instruction Pointer: 

PaX Page Fault Strategy:

```
if (supervisor page &&
    IP on faulting page) {
    Terminate
} else {
    Set user page in PTE
    Prime Data TLB
    Set supervisor page in PTE
}
```

PaX PAGEEXEC (2000)

User/
Supervisor:
Emulates
Non-Exec

Instruction TLB:

Virtual Addr	Physical Addr	Permission

Data TLB:

Virtual Addr	Physical Addr	Permission

Instruction Pointer: 

PaX Page Fault Strategy:

```
if (supervisor page &&
    IP on faulting page) {
    Terminate
} else {
    Set user page in PTE
    Prime Data TLB
    Set supervisor page in PTE
}
```

PaX PAGEEXEC (2000)

User/
Supervisor:
Emulates
Non-Exec

Instruction TLB:

Virtual Addr	Physical Addr	Permission

Data TLB:

Virtual Addr	Physical Addr	Permission

Instruction Pointer: 

PaX Page Fault Strategy:

```
if (supervisor page &&
    IP on faulting page) {
    Terminate
} else {
    Set user page in PTE
    Prime Data TLB
    Set supervisor page in PTE
}
```

PaX PAGEEXEC (2000)

User/
Supervisor:
Emulates
Non-Exec

Instruction TLB:

Virtual Addr	Physical Addr	Permission

Data TLB:

Virtual Addr	Physical Addr	Permission

Instruction Pointer: 

PaX Page Fault Strategy:

```
if (supervisor page &&
    IP on faulting page) {
    Terminate
} else {
    Set user page in PTE
    Prime Data TLB
    Set supervisor page in PTE
}
```

PaX PAGEEXEC (2000)

User/
Supervisor:
Emulates
Non-Exec

Instruction TLB:

Virtual Addr	Physical Addr	Permission

Data TLB:

Virtual Addr	Physical Addr	Permission
	~	User/~

Instruction Pointer: 

PaX Page Fault Strategy:

```
if (supervisor page &&
    IP on faulting page) {
    Terminate
} else {
    Set user page in PTE
    Prime Data TLB
    Set supervisor page in PTE
}
```

PaX PAGEEXEC (2000)

User/
Supervisor:
Emulates
Non-Exec

Instruction TLB:

Virtual Addr	Physical Addr	Permission

Data TLB:

Virtual Addr	Physical Addr	Permission
	~	User/~

Instruction Pointer: 

PaX Page Fault Strategy:

```
if (supervisor page &&
    IP on faulting page) {
    Terminate
} else {
    Set user page in PTE
    Prime Data TLB
    Set supervisor page in PTE
}
```

PaX PAGEEXEC (2000)

User/
Supervisor:
Emulates
Non-Exec

Instruction TLB:

Virtual Addr	Physical Addr	Permission

Data TLB:

Virtual Addr	Physical Addr	Permission
	~	User/~

Instruction Pointer:

PaX Page Fault Strategy:

```
if (supervisor page &&
    IP on faulting page) {
    Terminate
} else {
    Set user page in PTE
    Prime Data TLB
    Set supervisor page in PTE
}
```

PaX PAGEEXEC (2000)

User/
Supervisor:
Emulates
Non-Exec

Instruction TLB:

Virtual Addr	Physical Addr	Permission

Data TLB:

Virtual Addr	Physical Addr	Permission
	~	User/~

Instruction Pointer:

PaX Page Fault Strategy:

```
if (supervisor page &&
    IP on faulting page) {
    Terminate
} else {
    Set user page in PTE
    Prime Data TLB
    Set supervisor page in PTE
}
```

PaX PAGEEXEC (2000)

User/
Supervisor:
Emulates
Non-Exec

Instruction TLB:

Virtual Addr	Physical Addr	Permission

Data TLB:

Virtual Addr	Physical Addr	Permission
	~	User/~

Instruction Pointer:

PaX Page Fault Strategy:

```
if (supervisor page &&
    IP on faulting page) {
    Terminate
} else {
    Set user page in PTE
    Prime Data TLB
    Set supervisor page in PTE
}
```

PaX PAGEEXEC (2000)

User/
Supervisor:
Emulates
Non-Exec

Instruction TLB:

Virtual Addr	Physical Addr	Permission
	~	User/~

Data TLB:

Virtual Addr	Physical Addr	Permission
	~	User/~

Instruction Pointer:

PaX Page Fault Strategy:

```
if (supervisor page &&
    IP on faulting page) {
    Terminate
} else {
    Set user page in PTE
    Prime Data TLB
    Set supervisor page in PTE
}
```

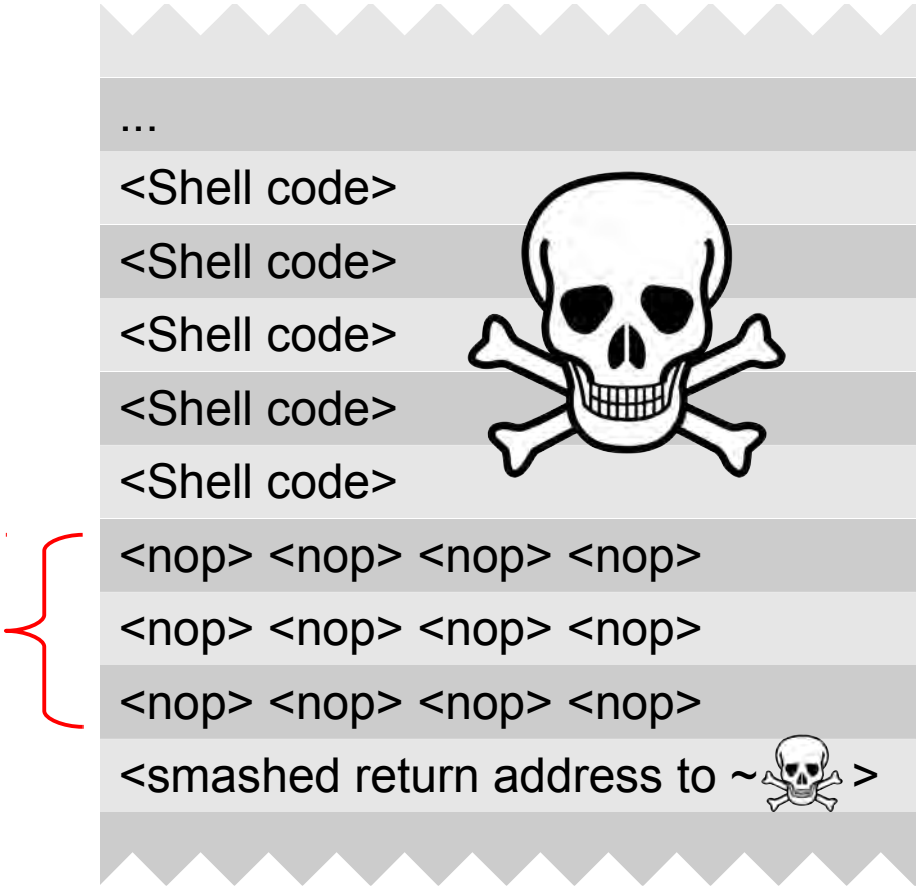
Page Level Permissions

For mapped pages:

	User	Supervisor PaX/NX
Not-Writable	Read/Execute	Read
Writable	Read/Write/Execute	Read/Write

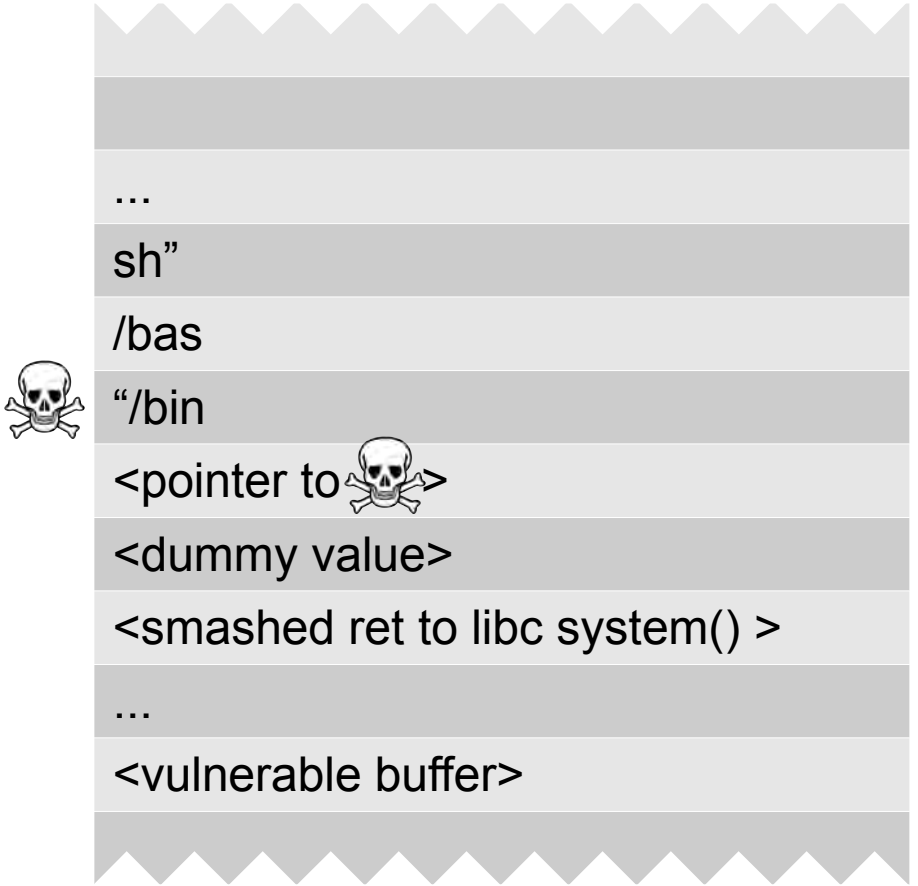
Part III:
Code Reuse

Return to libc (1997)

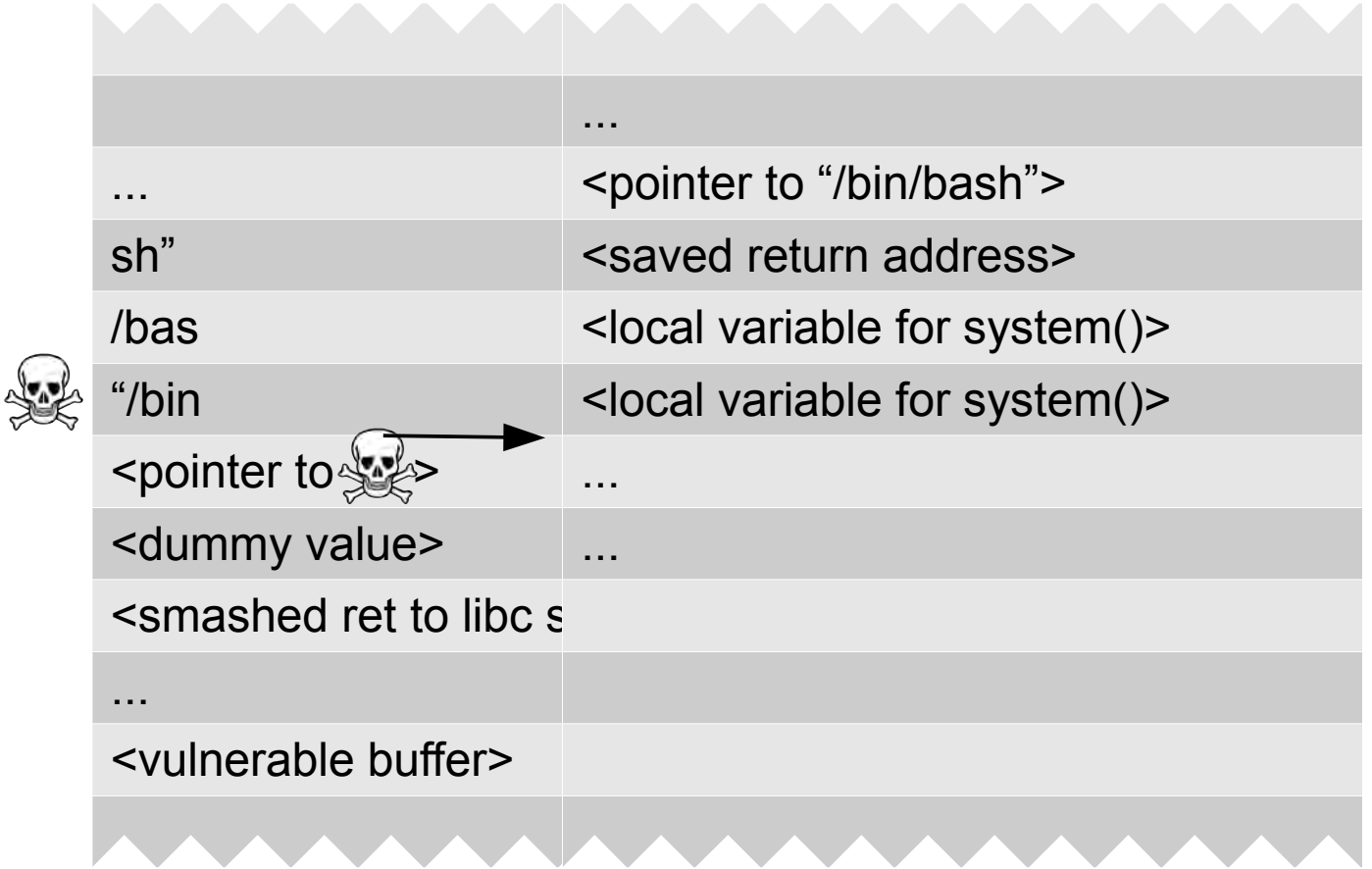


Return to libc (1997)

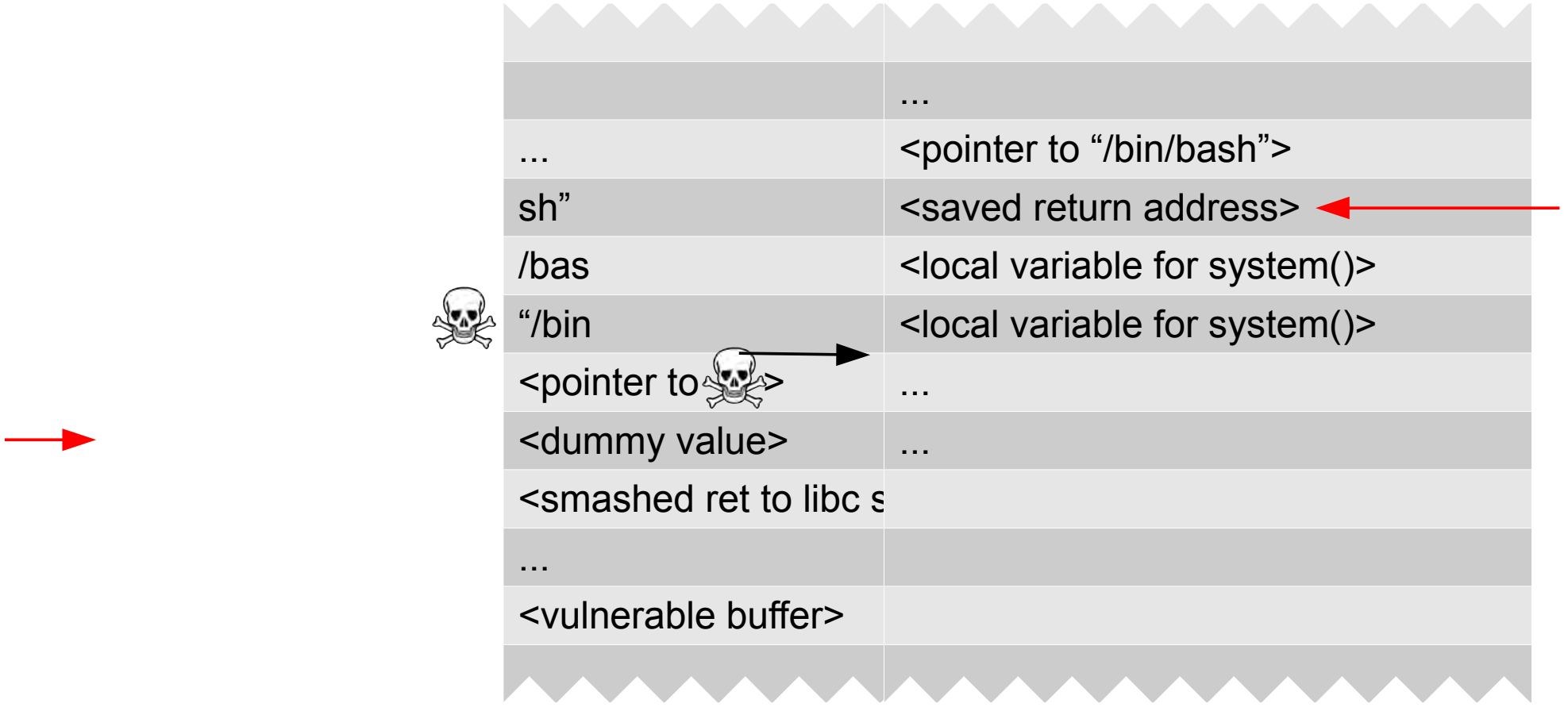
Return to libc (1997)



Return to libc (1997)



Return to libc (1997)



Return Oriented Programming ('07)

```
push eax
ret
```

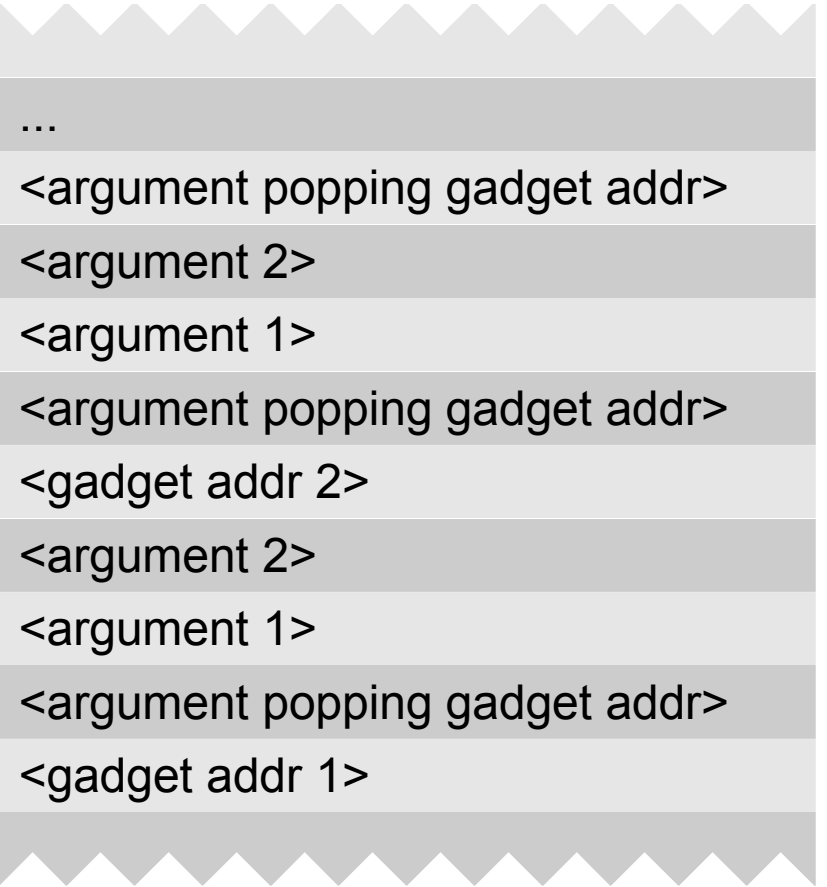
```
pop eax
ret
```

```
pop ebx
ret
```

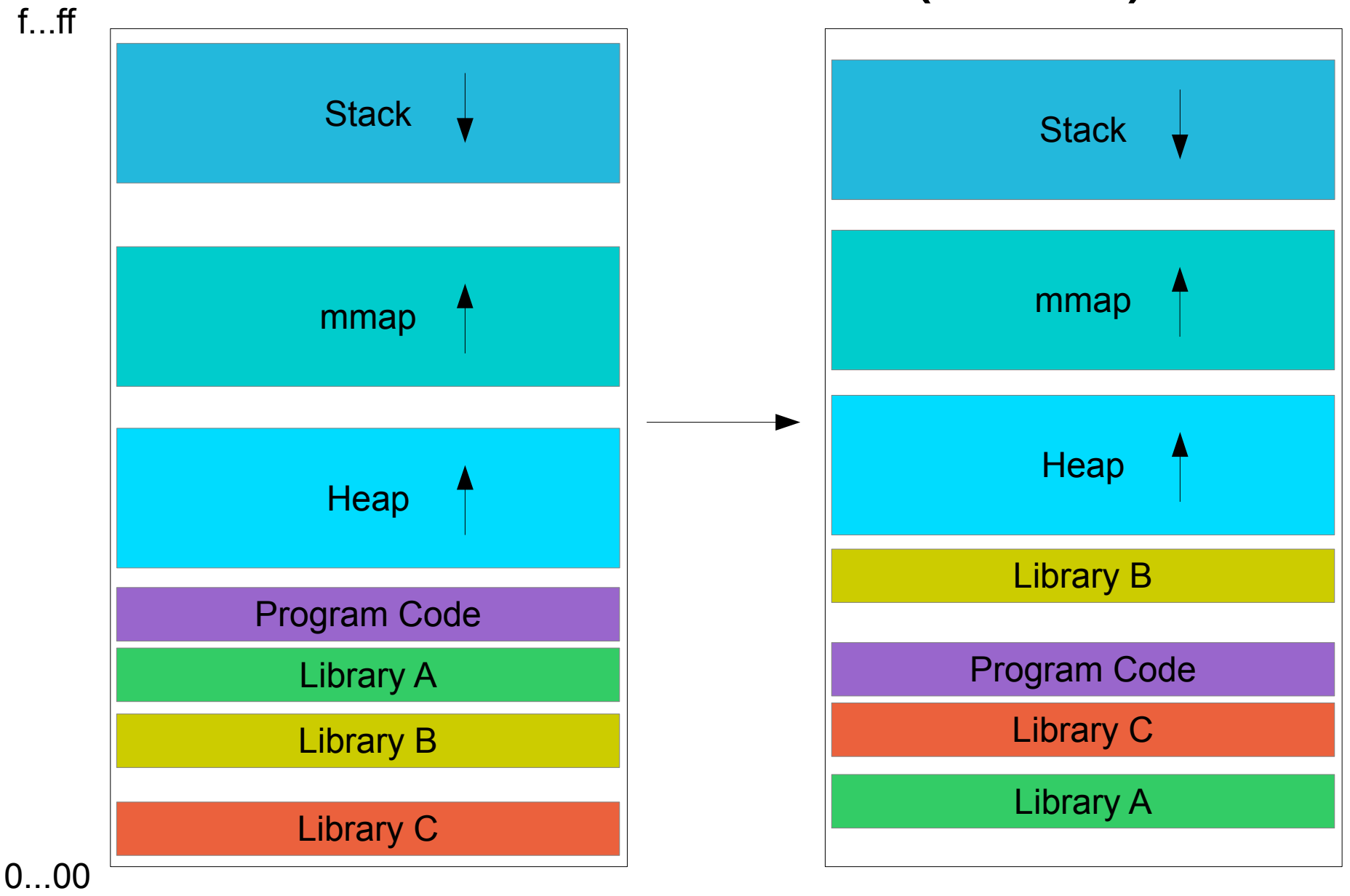
```
mov [ebx],eax
ret
```

```
xchg ebx,esp
ret
```

```
pop edi
pop ebp
ret
```

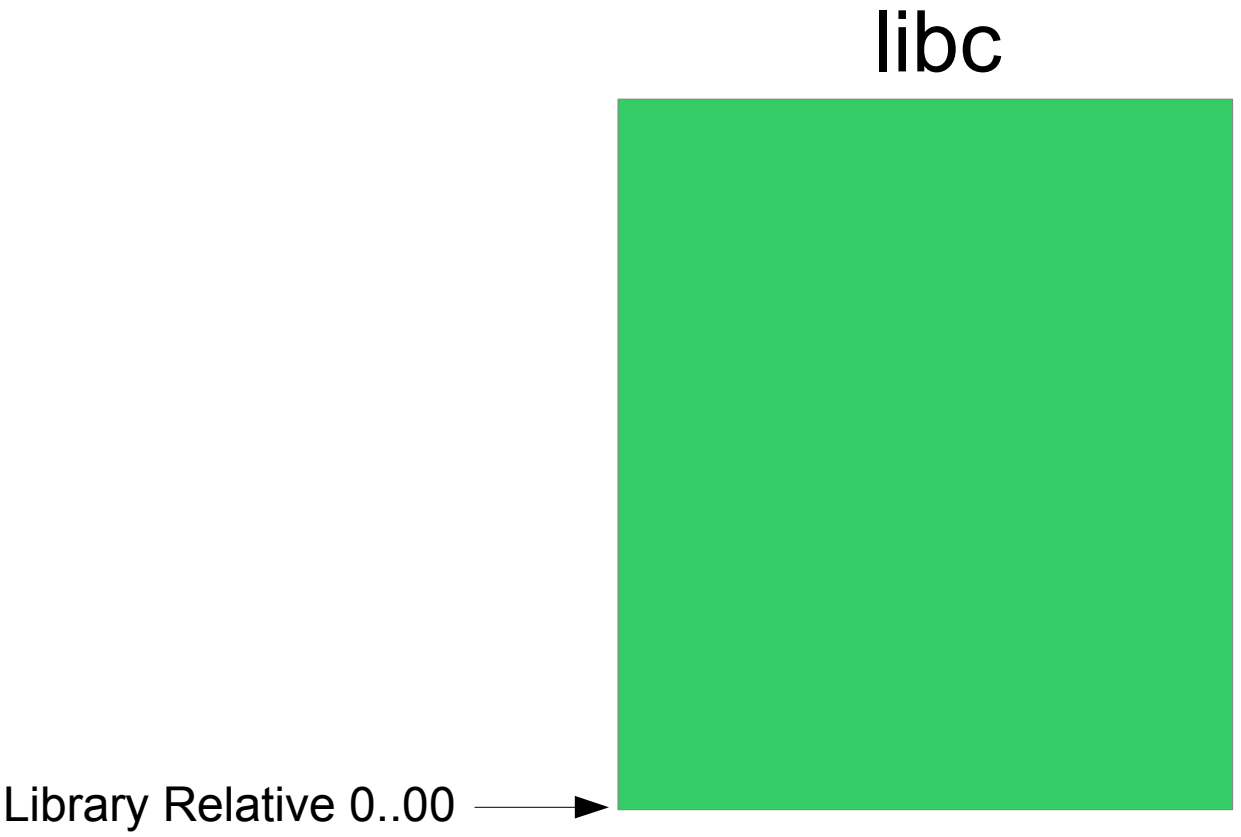


Address Space Layout Randomization (2003)

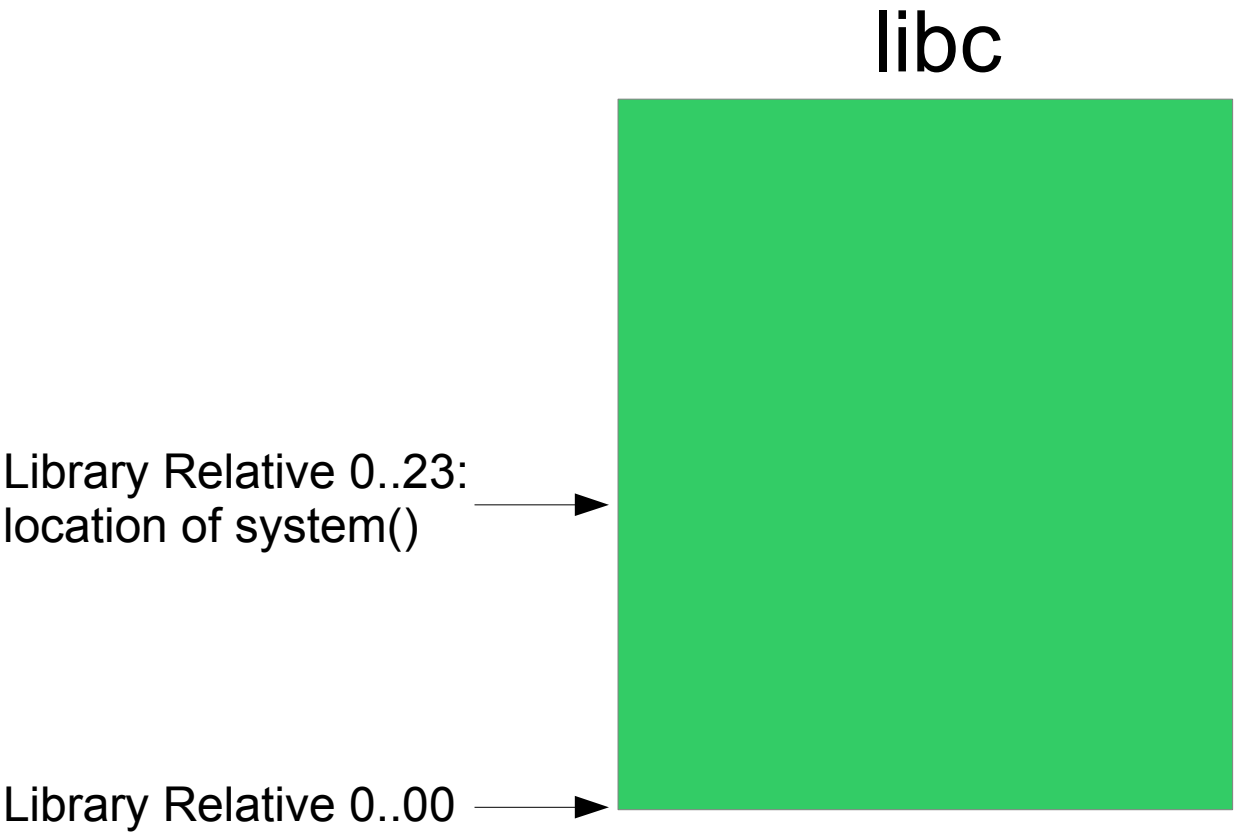


Part IV:
Memory Disclosure
&
Advanced Code Reuse

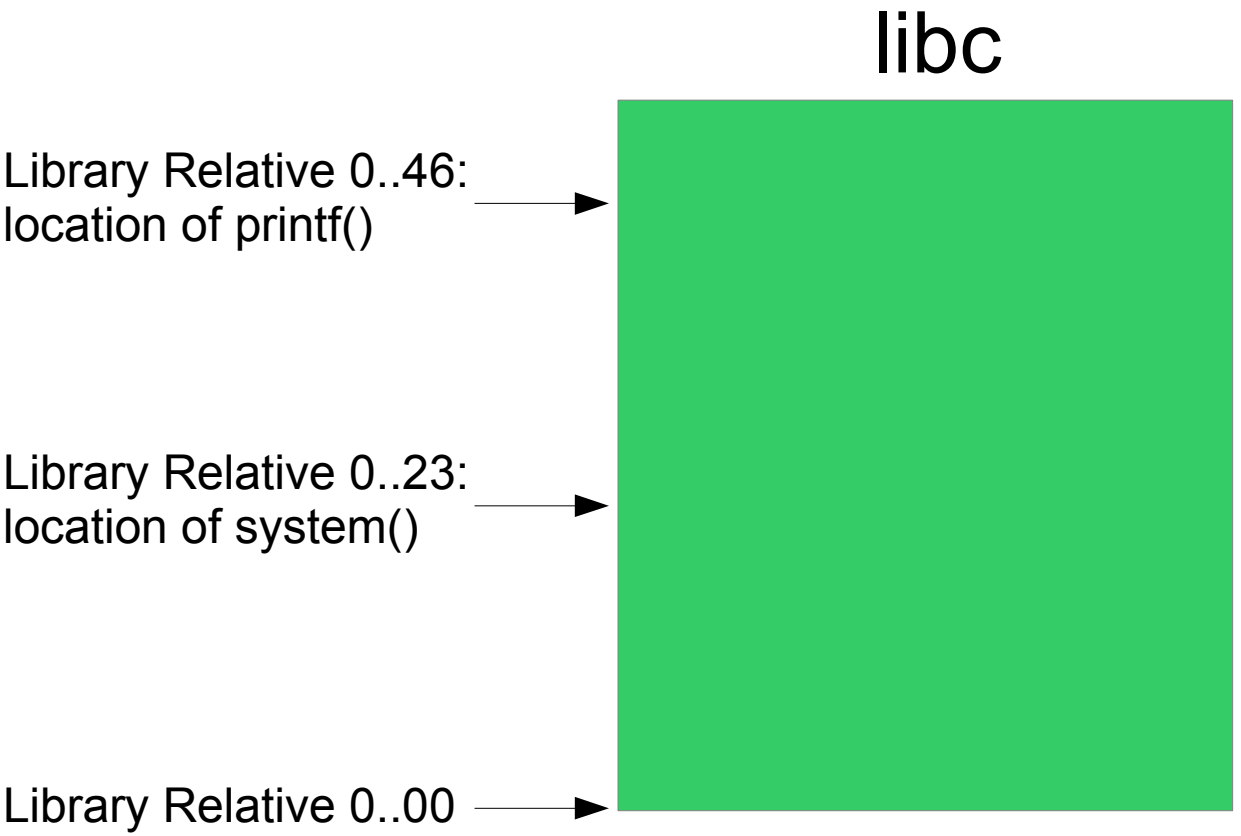
Offset Fix Ups



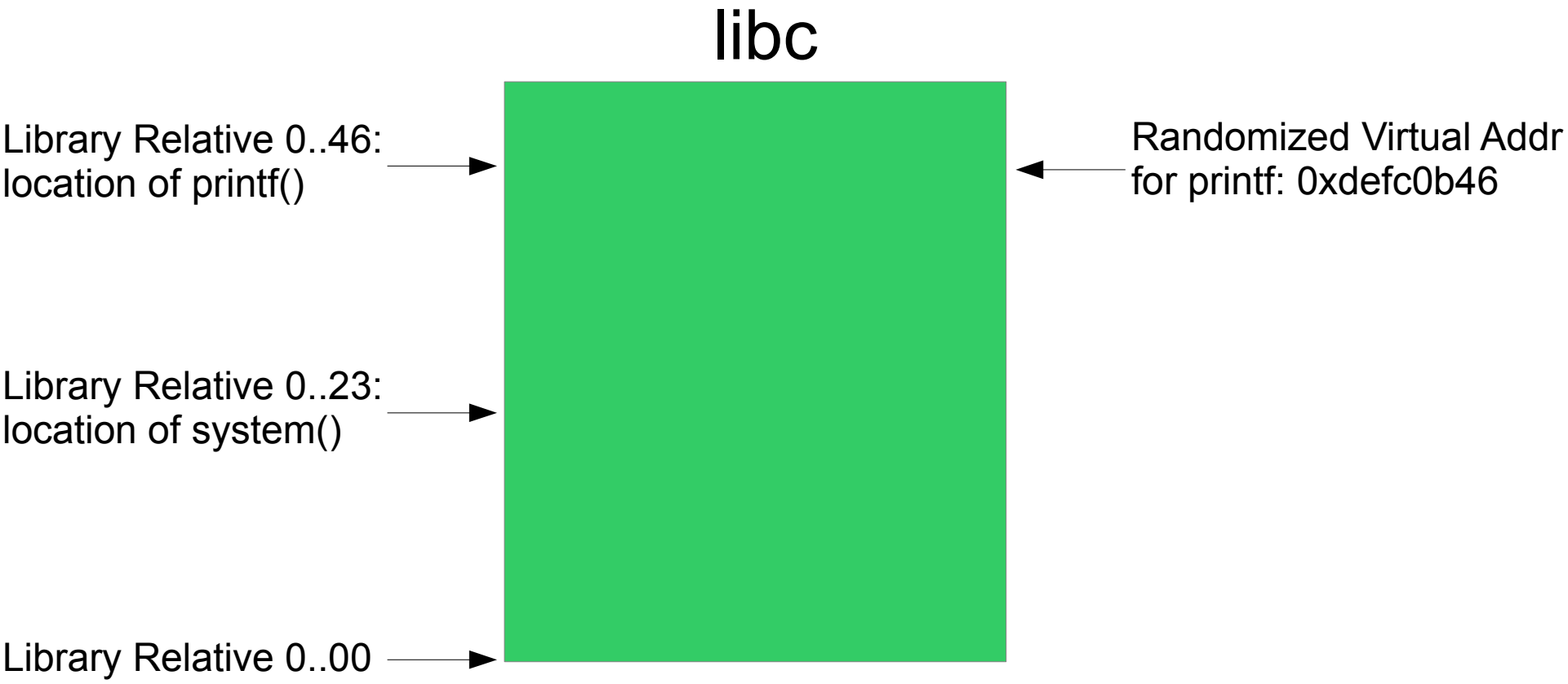
Offset Fix Ups



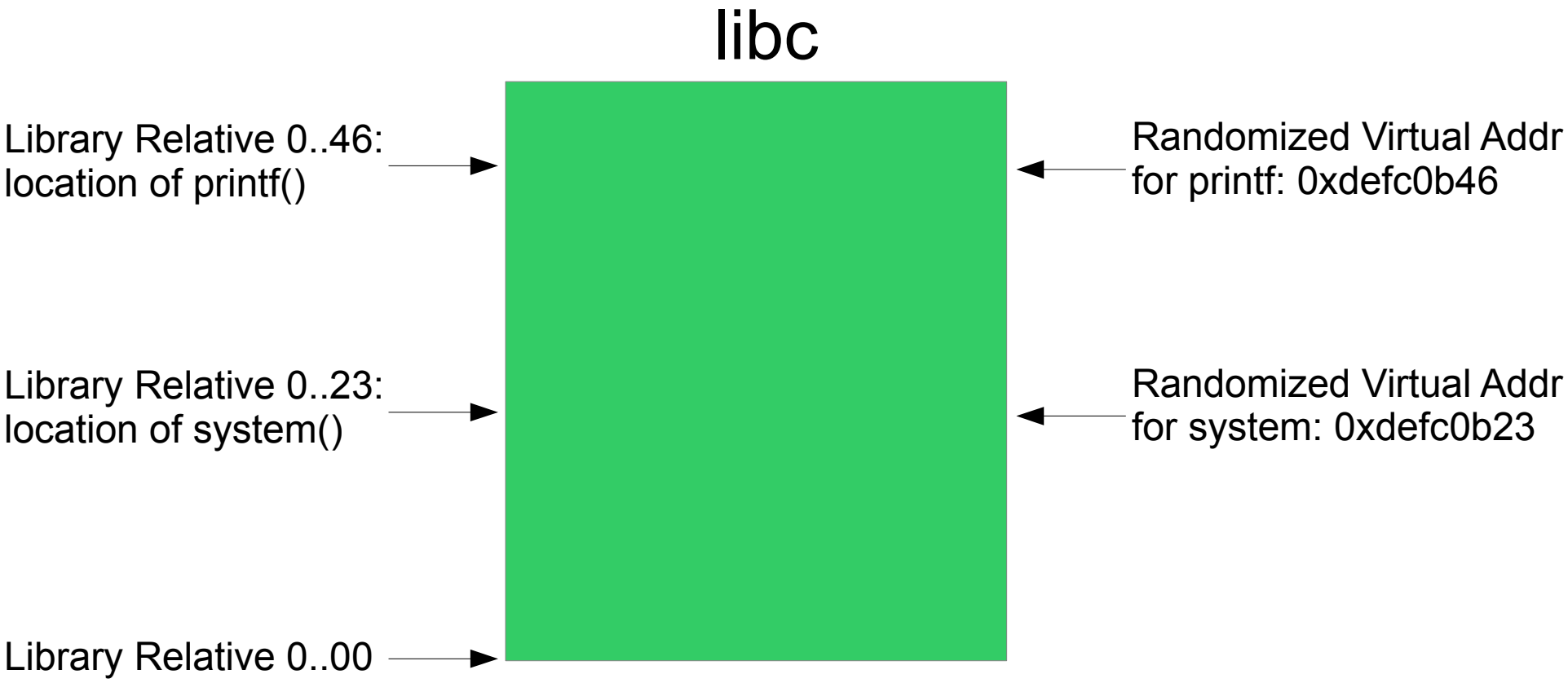
Offset Fix Ups



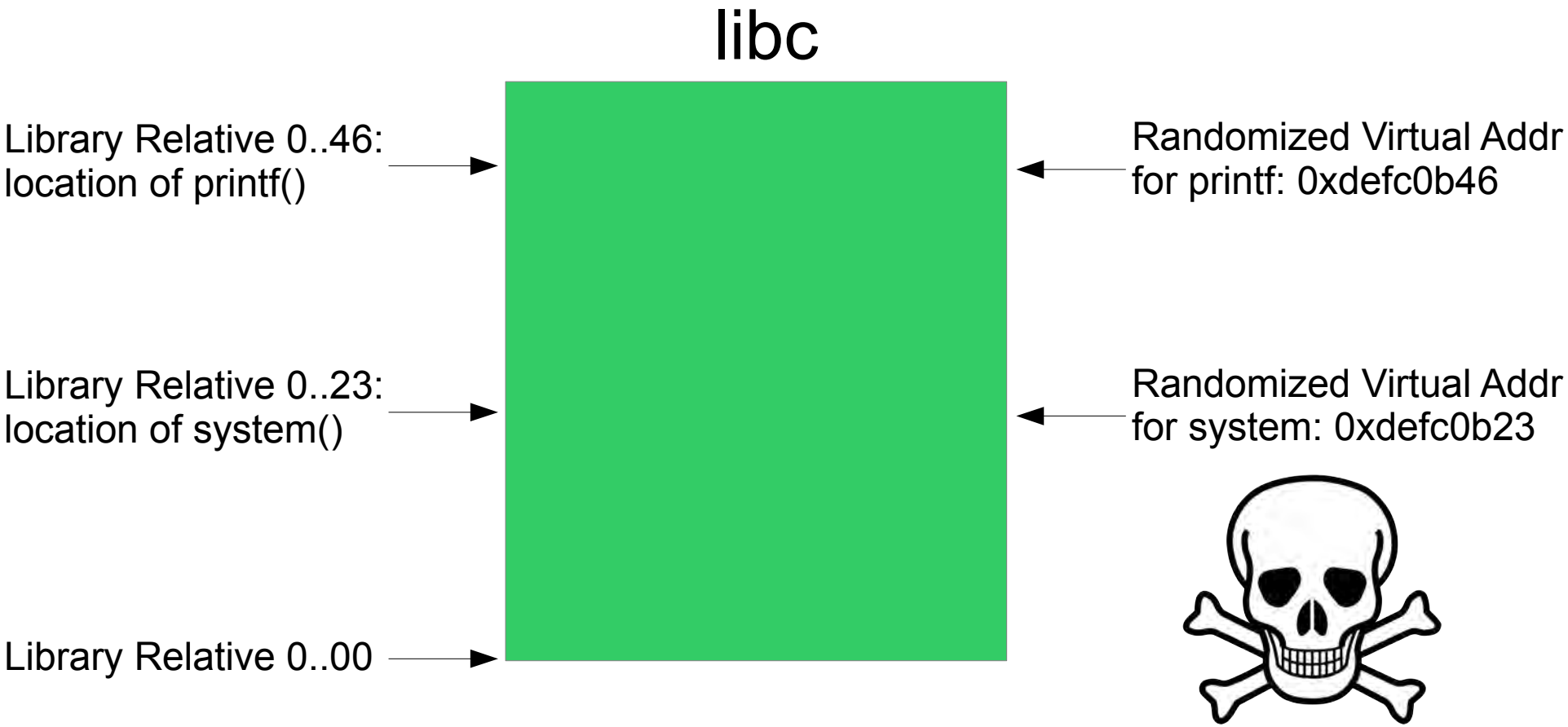
Offset Fix Ups



Offset Fix Ups



Offset Fix Ups



Fine Grained ASLR

- Smashing the Gadgets (2012)

```
mov eax, [ebp-4]
mov ebx, [ebp-8]
add eax, ebx
xor ecx, ecx
push eax
push ebx
push ecx
call foo
```

```
mov edx, [ebp-4]
mov esi, [ebp-8]
add edx, esi
xor edi, edi
push edx
push esi
push edi
call foo
```

- Address Space Layout Permutation (2006)

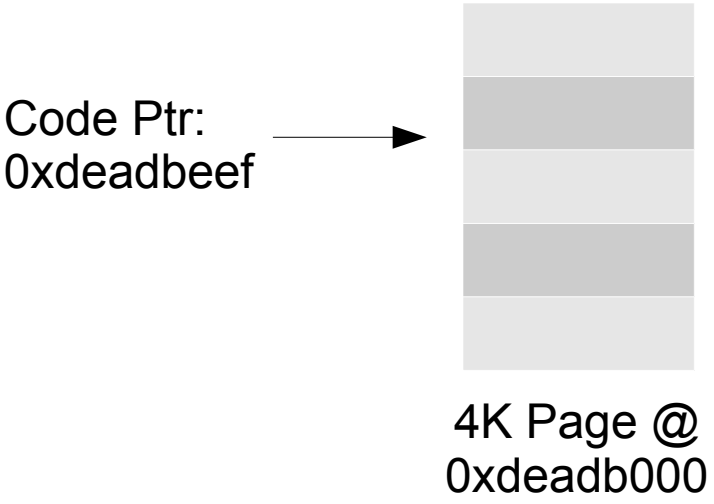
Function level FG-ASLR:

lib-func-a	lib-func-b
lib-func-b	lib-func-f
lib-func-c	lib-func-a
lib-func-d	lib-func-c
lib-func-f	lib-func-d

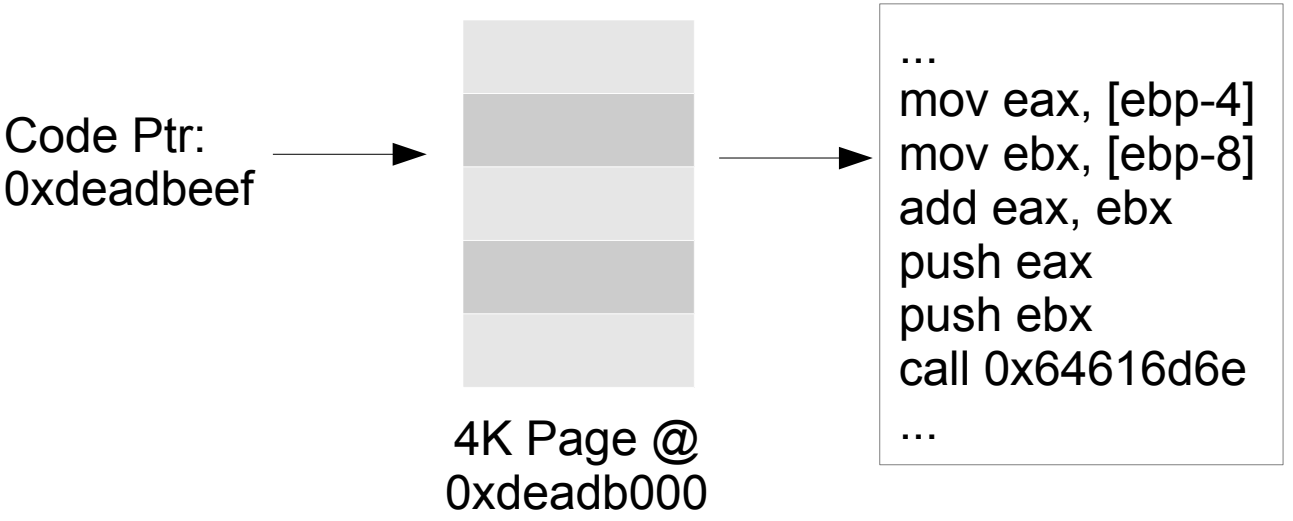
Just-in-Time Code Reuse (2013)

Code Ptr:
0xdeadbeef

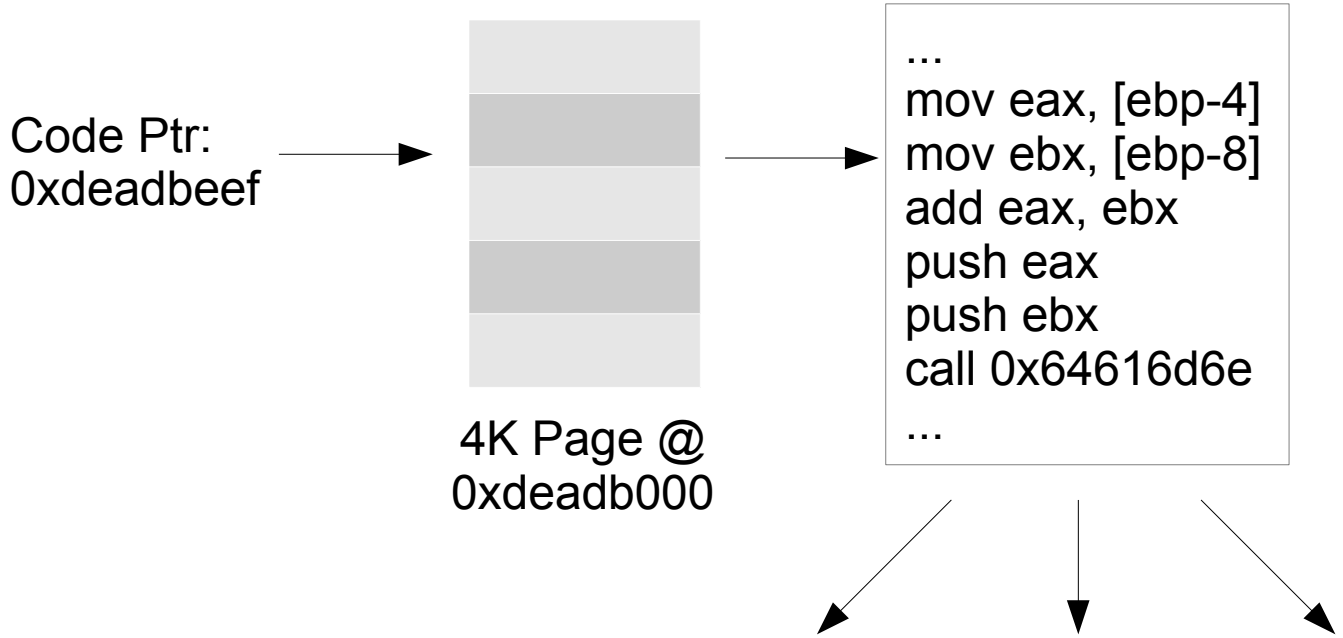
Just-in-Time Code Reuse (2013)



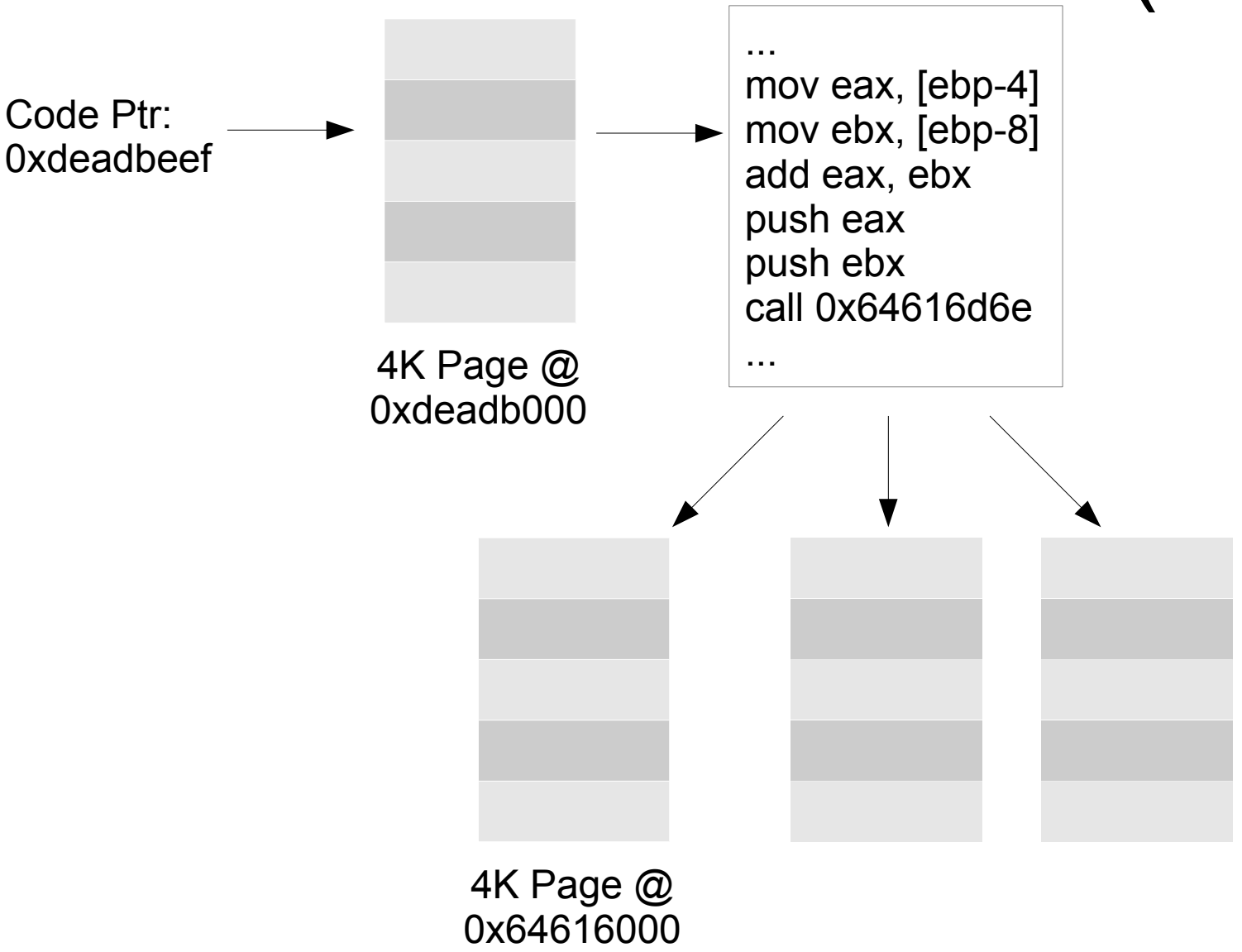
Just-in-Time Code Reuse (2013)



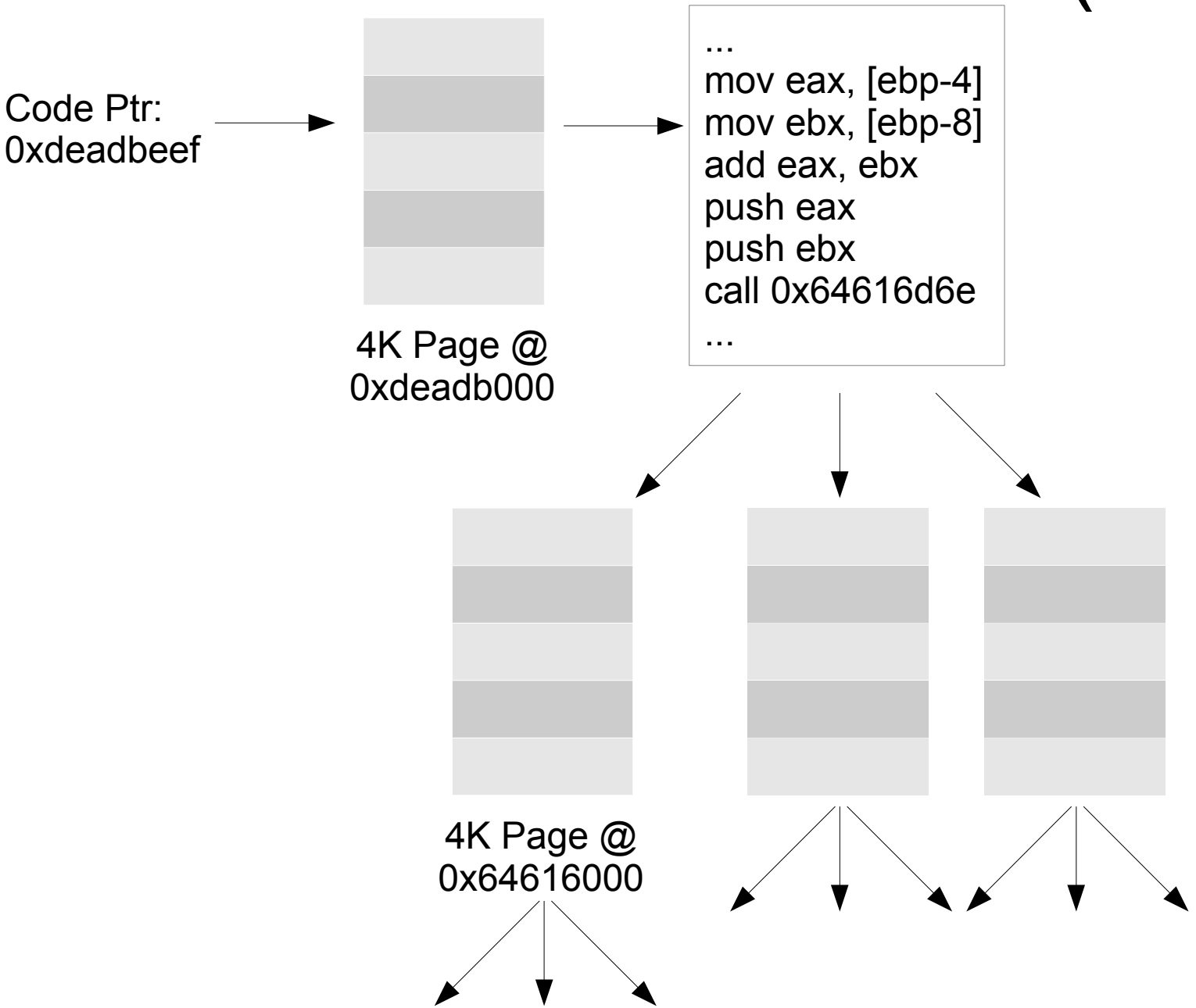
Just-in-Time Code Reuse (2013)



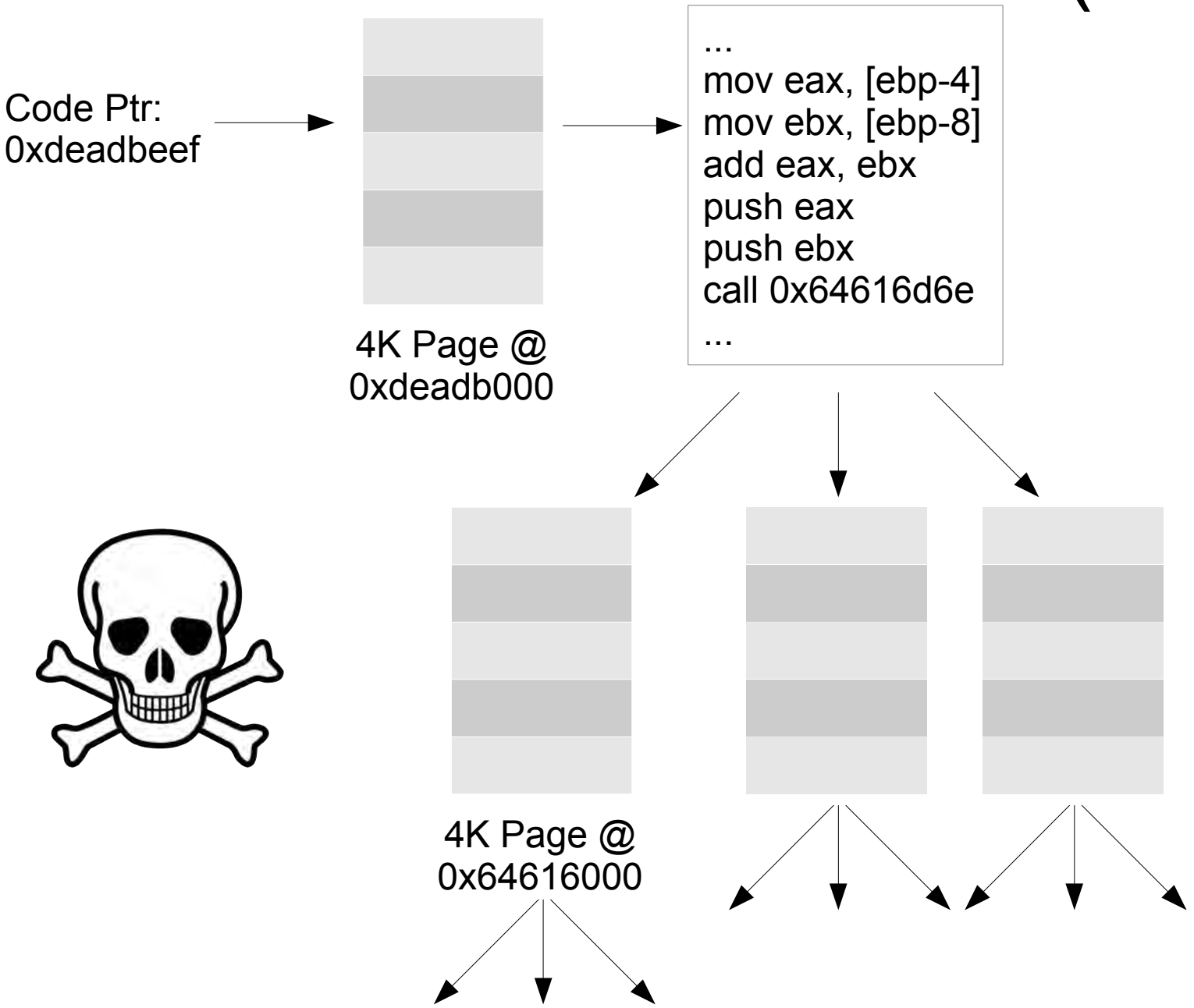
Just-in-Time Code Reuse (2013)



Just-in-Time Code Reuse (2013)



Just-in-Time Code Reuse (2013)



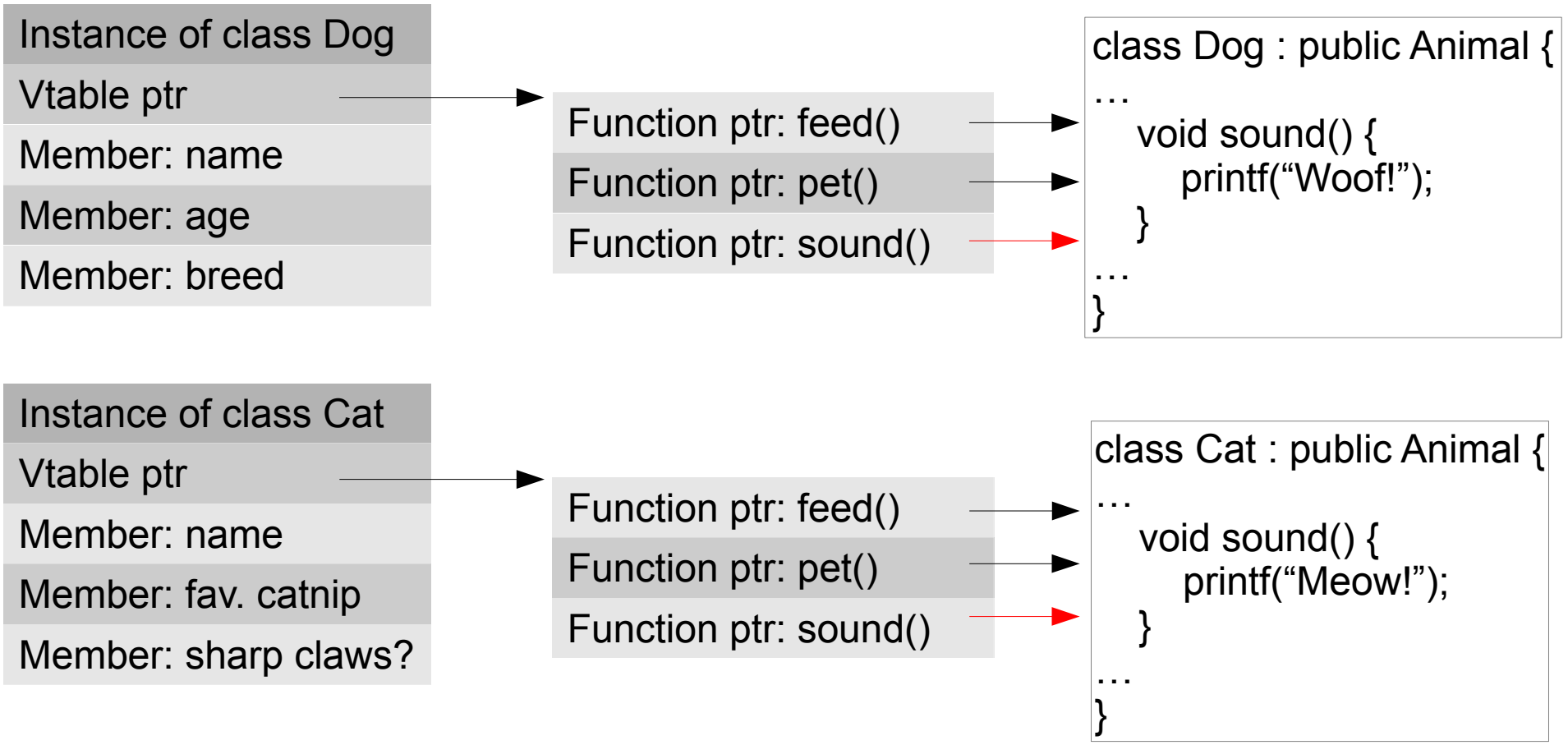
The Value of One Pointer?

Volcano and Hobbit: sold separately.

Part V:
Conceal
&
Forget

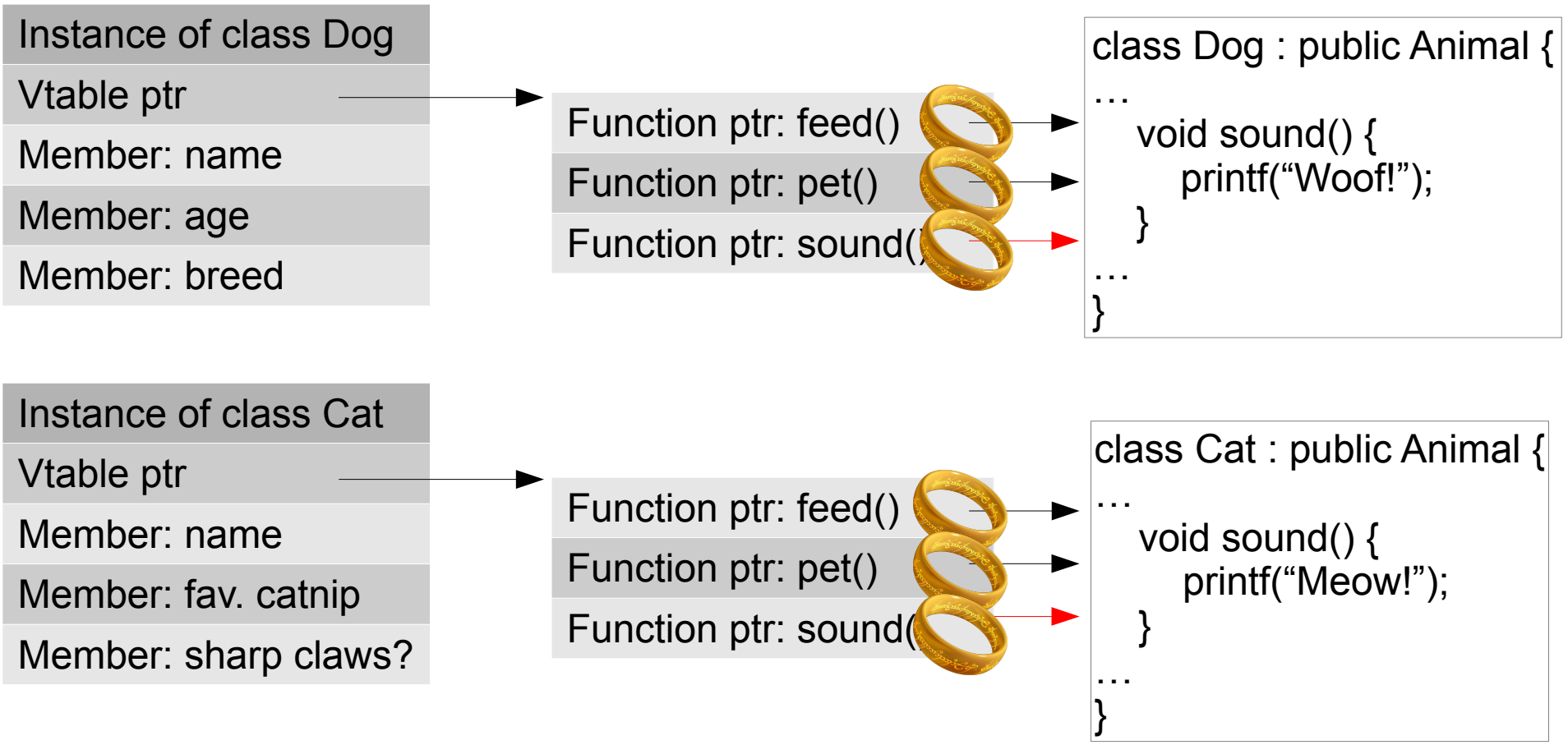
C++ Virtual Function Tables

Animal → Dog, Animal → Cat

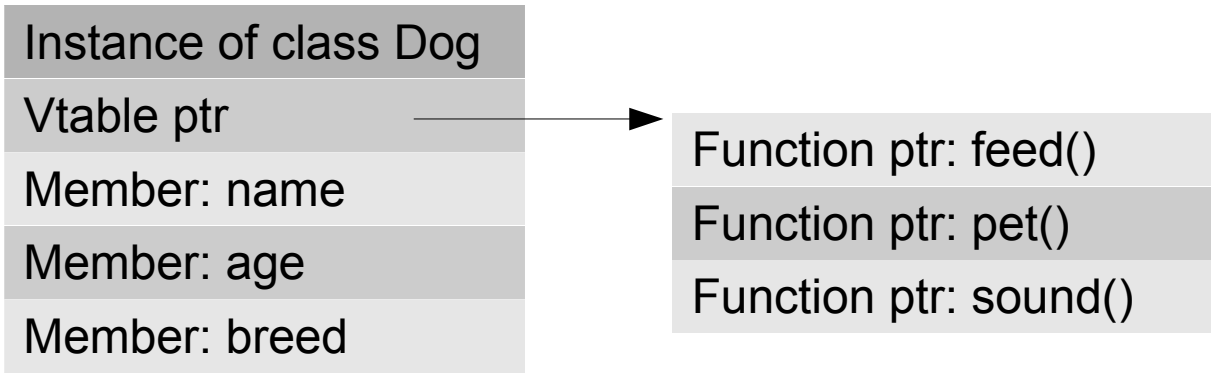


C++ Virtual Function Tables

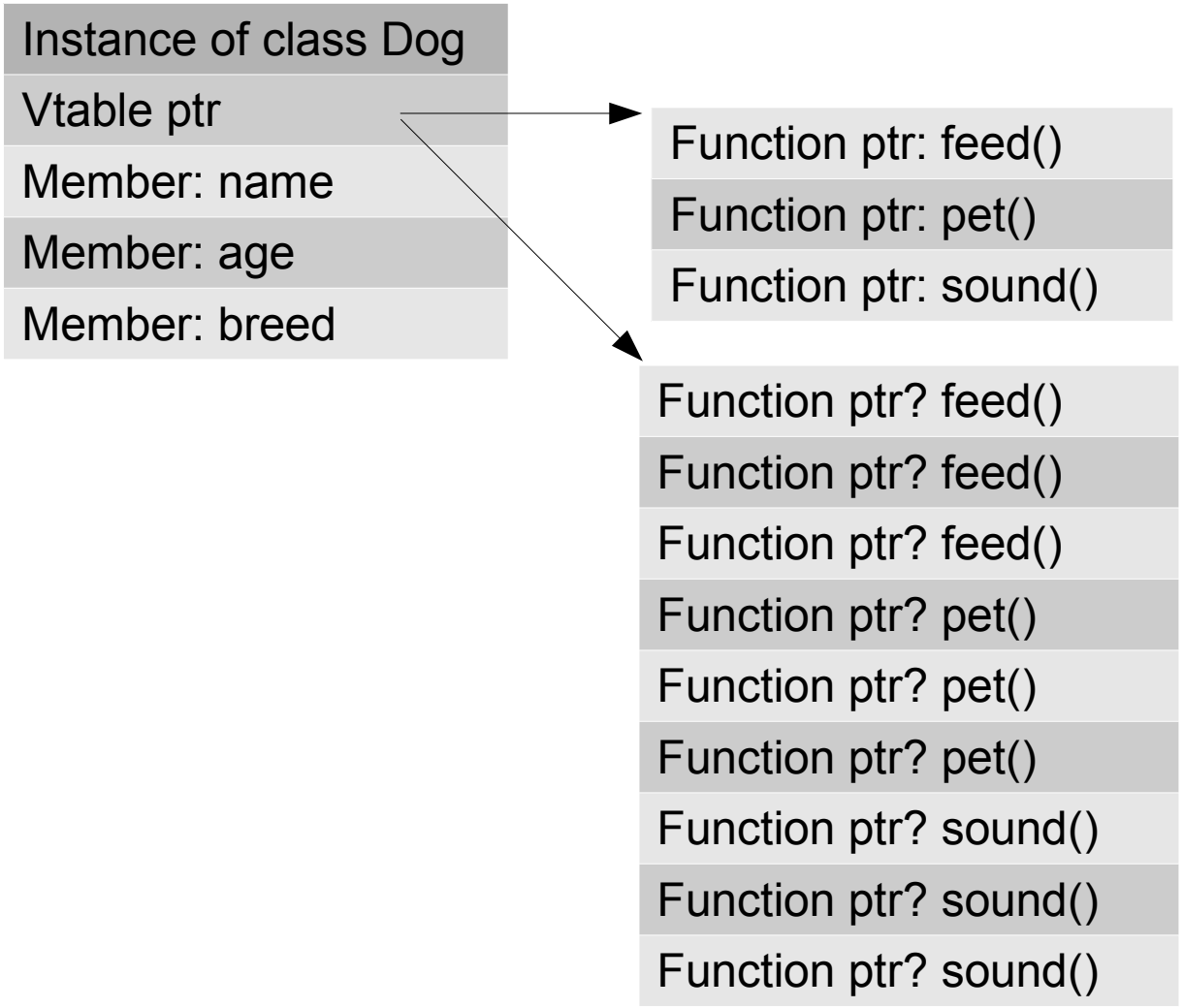
Animal → Dog, Animal → Cat



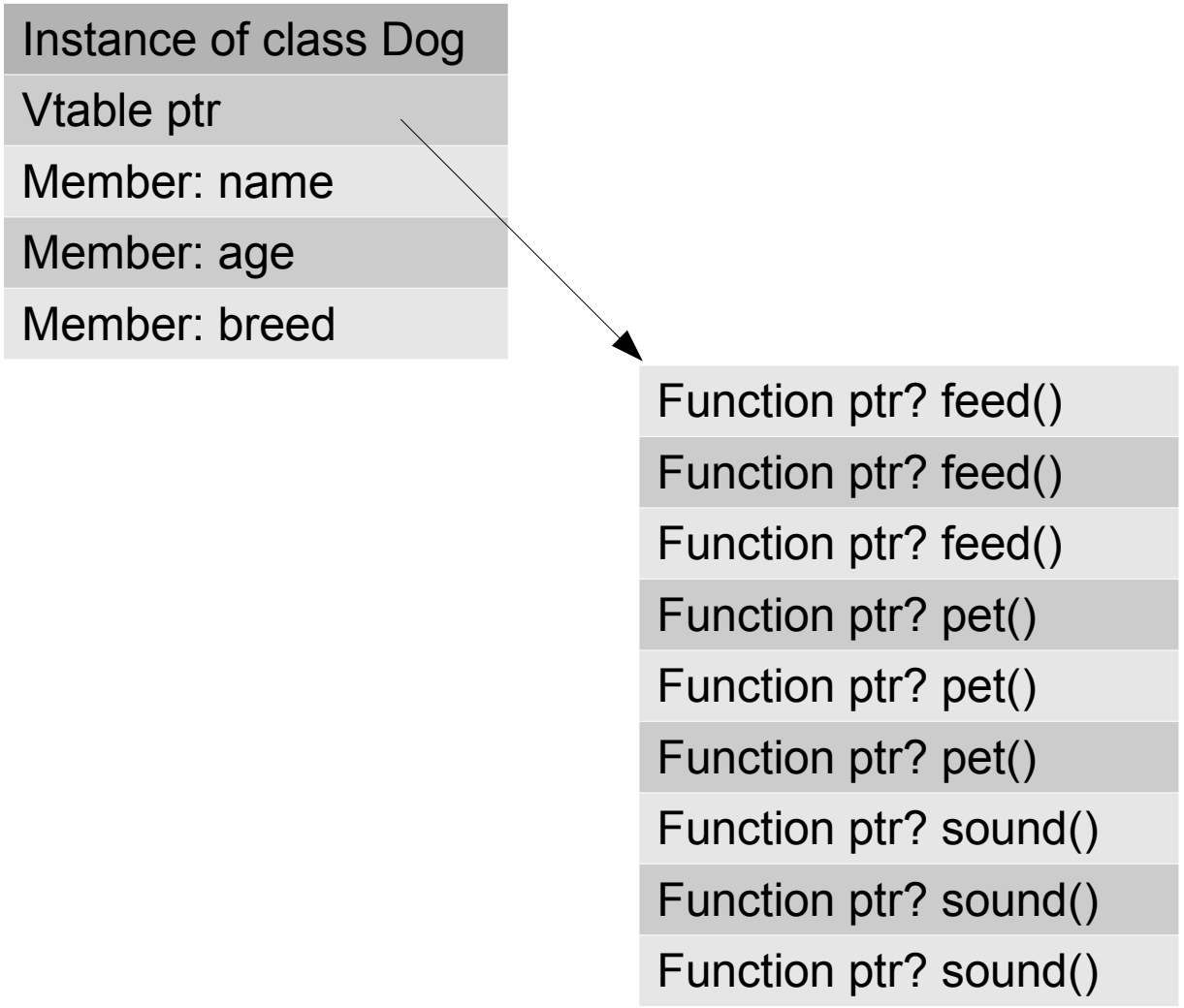
Knights and Knaves



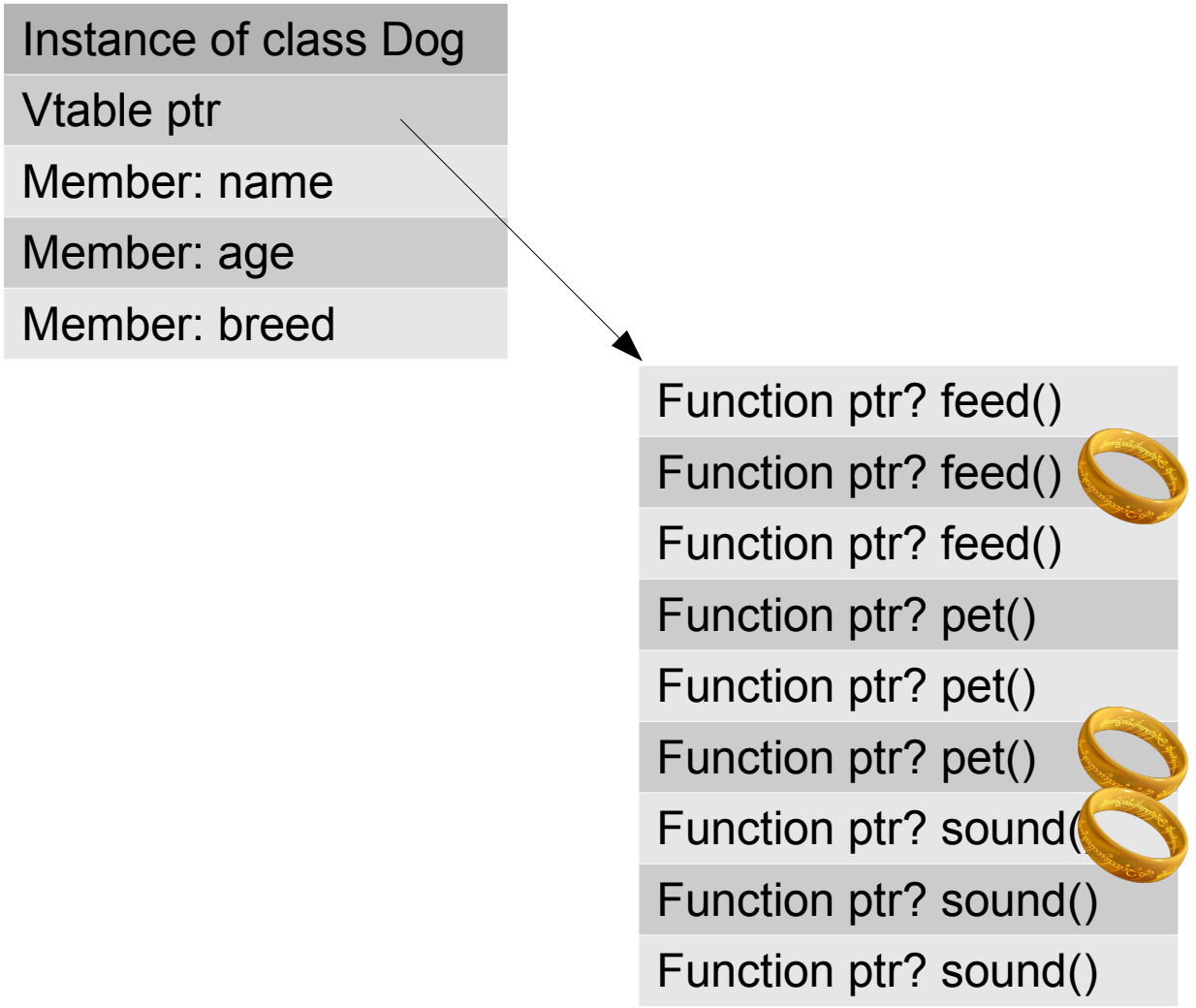
Knights and Knaves



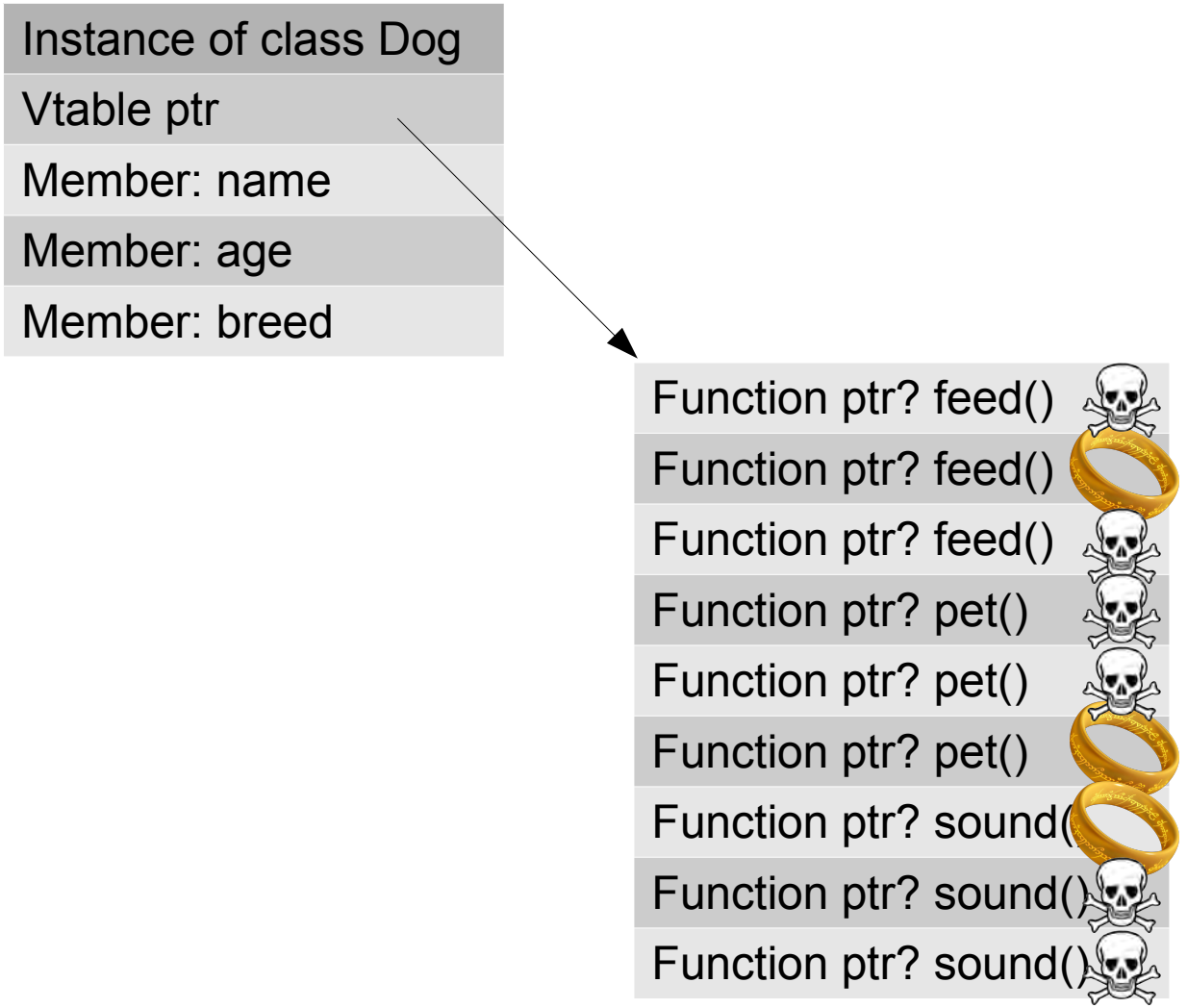
Knights and Knaves



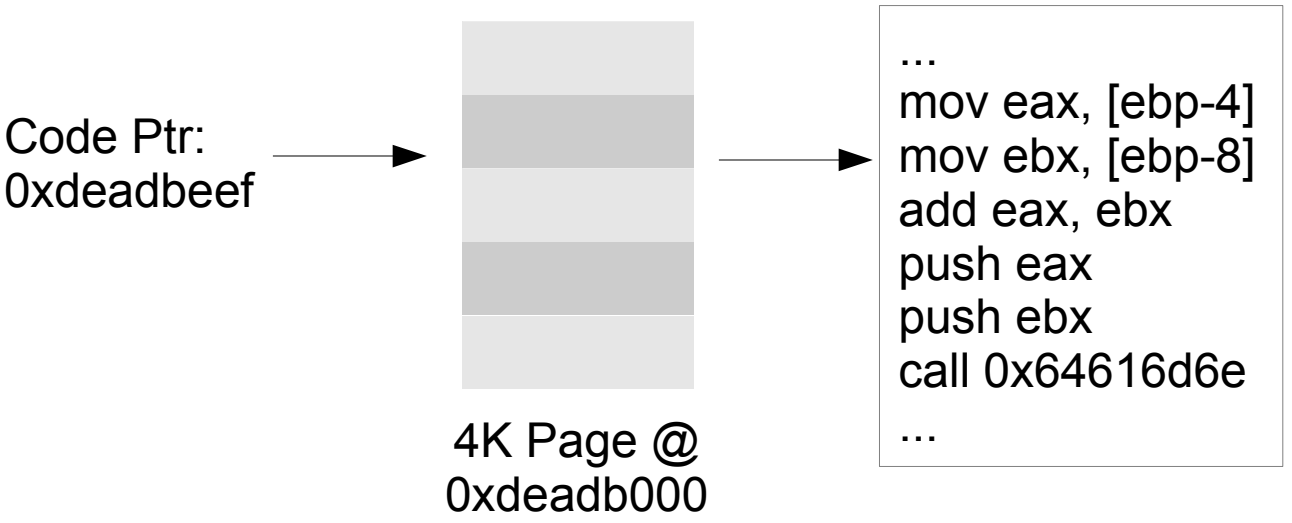
Knights and Knaves



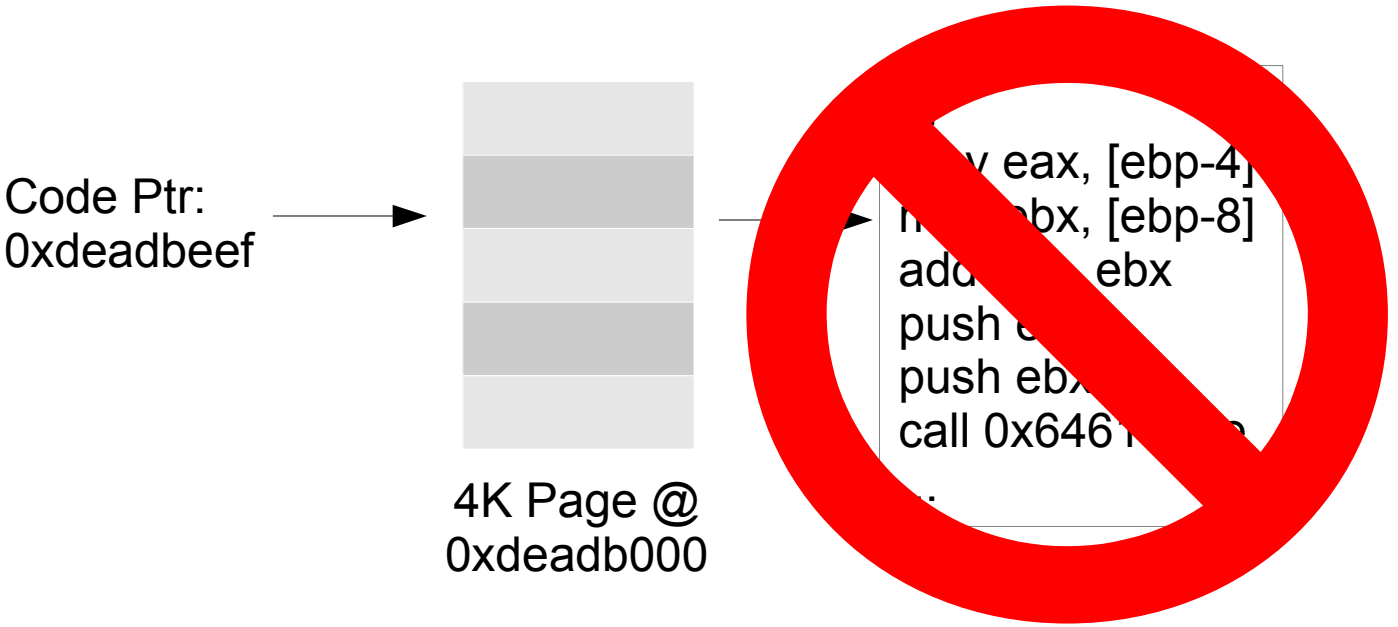
Knights and Knaves



Execute Only Memory



Execute Only Memory



Necessary vs. Sufficient

- Code reuse requires:
  - No ASLR: A priori knowledge of *place*
  - ASLR: A priori knowledge of relative place + runtime discovery of offset
  - FG-ASLR: Runtime discovery of *value* at discovered *place*
- No runtime discovery? No discovery of value or place and no code to reuse:
  - XO-M + FG-ASLR = <3

Elephant in the Room

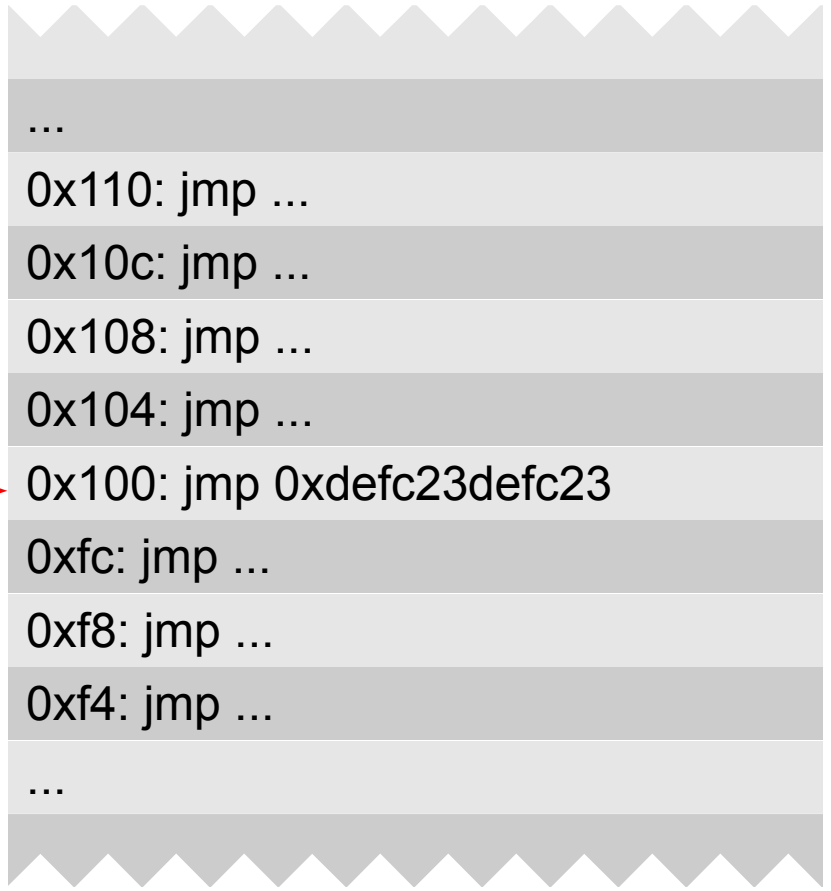
Two words: memory overhead

https://www.flickr.com/photos/mobilestreetlife/4179063482

Blunting the Edge

- Oxymoron (2014)
  - Key idea: call fs:0x100

```
mov eax, [ebp-4]
mov ebx, [ebp-8]
add eax, ebx
xor ecx, ecx
push eax
push ebx
push ecx
call fs:0x100
```



Start of fs segment at random addr...

Xen, Linux, & LLVM

- Xen 4.4 introduced PVH mode (Xen 4.5 → PVH dom0)
  - PVH uses Intel Extended Page Tables for PFN → MFN translations
  - EPT supports explicit R/W/E permissions
- Linux mprotect M_EXECUTE & ~M_READ sets EPT through Xen
  - Xen injects violations into Linux #PF handler
- LLVM for FG-ASLR and execute-only codegen

Part VI:
Closing Thoughts

Takeaways

Non-Writable	Readable	EPT ~R
X	Read/Execute	Execute Only
NX	Read	Nothing

Writable	Readable	EPT ~R
X	Read/Write/Execute	Write/Execute
NX	Read/Write	Write

Takeaways

Non-Writable	Readable	EPT ~R
X	Read/Execute	Execute Only
NX	Read	Nothing

Constant Data

Writable	Readable	EPT ~R
X	Read/Write/Execute	Write/Execute
NX	Read/Write	Write

Takeaways

Non-Writable	Readable	EPT ~R
X	Read/Execute	Execute Only
NX	Read	Nothing

Constant Data

Writable	Readable	EPT ~R
X	Read/Write/Execute	Write/Execute
NX	Read/Write	Write

Stack/Heap/mmap

Takeaways

Non-Writable	Readable	EPT ~R
X	Read/Execute	Execute Only
NX	Read	Nothing

Constant Data

Program/Library Code

Writable	Readable	EPT ~R
X	Read/Write/Execute	Write/Execute
NX	Read/Write	Write

Stack/Heap/mmap

Takeaways

Non-Writable	Readable	EPT ~R
X	~~Read/Execute~~	Execute Only
NX	Read	Nothing

Constant Data

Program/Library Code

Writable	Readable	EPT ~R
X	Read/Write/Execute	Write/Execute
NX	Read/Write	Write

Stack/Heap/mmap

Takeaways

Non-Writable	Readable	EPT ~R
X	~~Read/Execute~~	Execute Only
NX	Read	Nothing

Constant Data

Program/Library Code

Writable	Readable	EPT ~R
X	~~Read/Write/Execute~~	Write/Execute
NX	Read/Write	Write

Stack/Heap/mmap

Takeaways

Non-Writable	Readable	EPT ~R
X	~~Read/Execute~~	Execute Only
NX	Read	~~Nothing~~

Constant Data

Program/Library Code

Writable	Readable	EPT ~R
X	~~Read/Write/Execute~~	~~Write/Execute~~
NX	Read/Write	~~Write~~

Stack/Heap/mmap

FIN

- Code: <TBD>
- White Paper: <TBD>
- Email: ds@thyth.com
- Twitter: @dsThyth
- PGP:
  - 201a 7b59 a15b e5f0 bc37 08d3 bc7f 39b2 dfc0 2d75