



Machine-Level Programming IV: Data - pointers

These slides adapted from materials provided by the textbook authors.

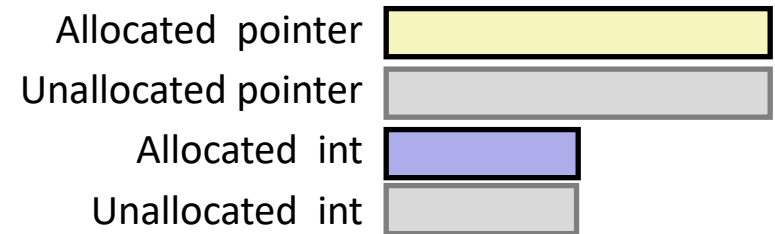
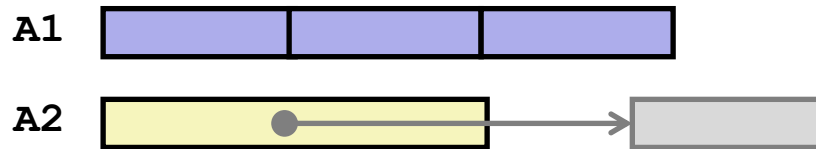
Understanding Pointers & Arrays #1

Decl	<i>An</i>			<i>*An</i>		
	Cmp	Bad	Size	Cmp	Bad	Size
<code>int A1[3]</code>						
<code>int *A2</code>						

- **Cmp: Compiles (Y/N)**
- **Bad: Possible bad pointer reference (Y/N)**
- **Size: Value returned by `sizeof`**

Understanding Pointers & Arrays #1

Decl	A _n			*A _n		
	Cmp	Bad	Size	Cmp	Bad	Size
<code>int A1[3]</code>	Y	N	12	Y	N	4
<code>int *A2</code>	Y	N	8	Y	Y	4



- **Cmp: Compiles (Y/N)**
- **Bad: Possible bad pointer reference (Y/N)**
- **Size: Value returned by `sizeof`**

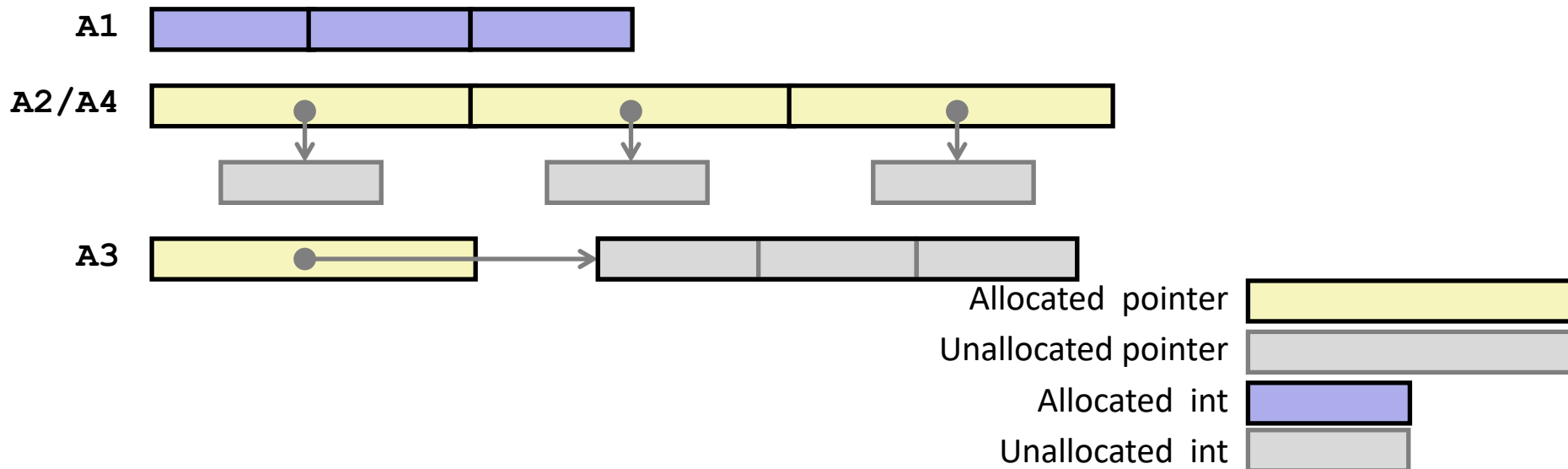
Understanding Pointers & Arrays #2

Decl	<i>An</i>			<i>*An</i>			<i>**An</i>		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
<code>int A1[3]</code>									
<code>int *A2[3]</code>									
<code>int (*A3)[3]</code>									
<code>int (*A4[3])</code>									

- **Cmp: Compiles (Y/N)**
- **Bad: Possible bad pointer reference (Y/N)**
- **Size: Value returned by `sizeof`**

Understanding Pointers & Arrays #2

Decl	<i>A_n</i>			<i>*A_n</i>			<i>**A_n</i>		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
<code>int A1[3]</code>	Y	N	12	Y	N	4	N	–	–
<code>int *A2[3]</code>	Y	N	24	Y	N	8	Y	Y	4
<code>int (*A3)[3]</code>	Y	N	8	Y	Y	12	Y	Y	4
<code>int (*A4[3])</code>	Y	N	24	Y	N	8	Y	Y	4

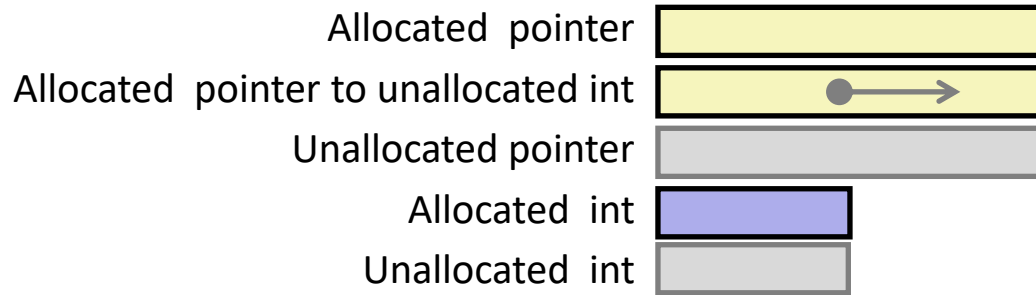


Understanding Pointers & Arrays #3

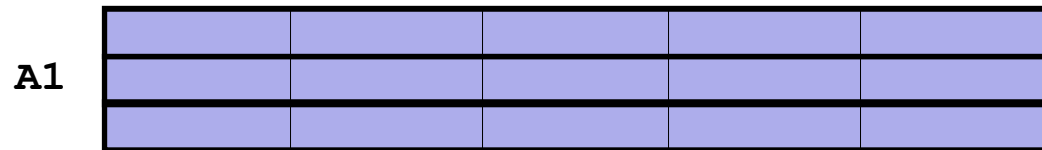
Decl	An			*An			**An		
	Cm p	Bad	Size	Cm p	Bad	Size	Cm p	Bad	Size
<code>int A1[3][5]</code>									
<code>int *A2[3][5]</code>									
<code>int (*A3)[3][5]</code>									
<code>int *(A4[3][5])</code>									
<code>int (*A5[3])[5]</code>									

- **Cmp: Compiles (Y/N)**
- **Bad: Possible bad pointer reference (Y/N)**
- **Size: Value returned by `sizeof`**

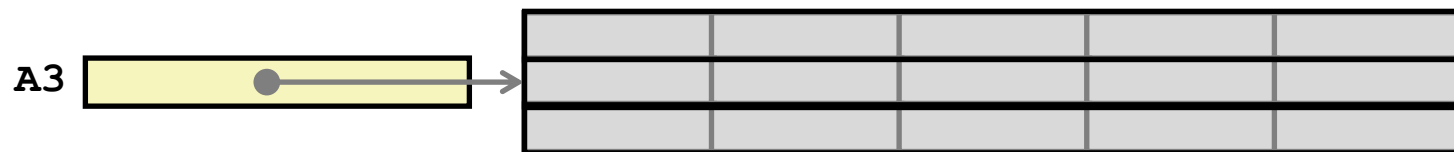
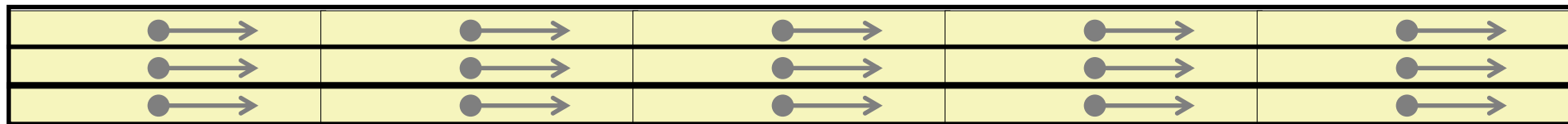
Decl	***An		
	Cm p	Bad	Size
<code>int A1[3][5]</code>			
<code>int *A2[3][5]</code>			
<code>int (*A3)[3][5]</code>			
<code>int *(A4[3][5])</code>			
<code>int (*A5[3])[5]</code>			



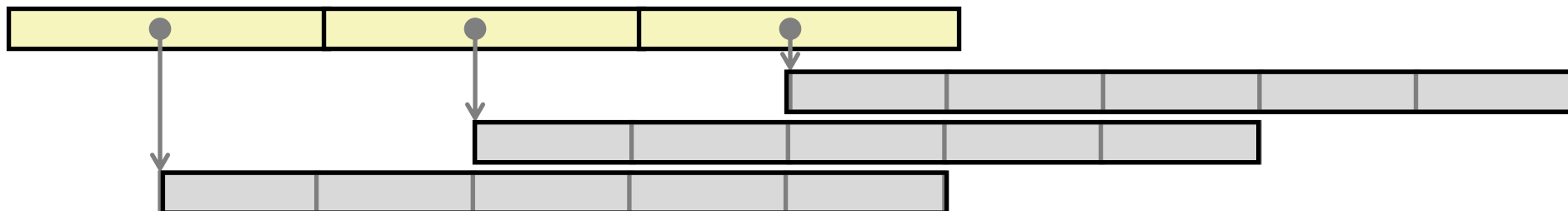
Declaration
<code>int A1[3][5]</code>
<code>int *A2[3][5]</code>
<code>int (*A3)[3][5]</code>
<code>int *(A4[3][5])</code>
<code>int (*A5[3])[5]</code>



A2/A4



A5



Understanding Pointers & Arrays #3

Decl	An			*An			**An		
	Cm p	Bad	Size	Cm p	Bad	Size	Cm p	Bad	Size
int A1[3][5]	Y	N	60	Y	N	20	Y	N	4
int *A2[3][5]	Y	N	120	Y	N	40	Y	N	8
int (*A3)[3][5]	Y	N	8	Y	Y	60	Y	Y	20
int *(A4[3][5])	Y	N	120	Y	N	40	Y	N	8
int (*A5[3])[5]	Y	N	24	Y	N	8	Y	Y	20

- **Cmp: Compiles (Y/N)**
- **Bad: Possible bad pointer reference (Y/N)**
- **Size: Value returned by sizeof**

Decl	***An		
	Cm p	Bad	Size
int A1[3][5]	N	-	-
int *A2[3][5]	Y	Y	4
int (*A3)[3][5]	Y	Y	4
int *(A4[3][5])	Y	Y	4
int (*A5[3])[5]	Y	Y	4

Doctor doctor

- Old joke: it hurts when I do this. So, don't do it.
- Use typedef
 - Typedef int *intptr;
 - Compare “int *foo[5]” to “intptr foo[5]”
- Make use of <http://cdecl.org>
 - You say “void (*f)(int *);”
 - It says “declare f as pointer to function (pointer to int) returning void”

Aside: Programming with SSE3

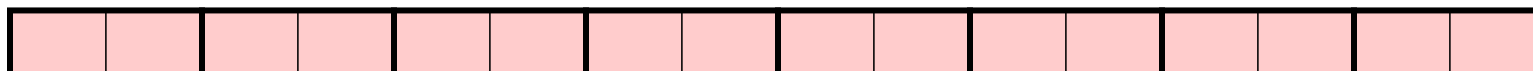
XMM Registers

■ 16 total, each 16 bytes

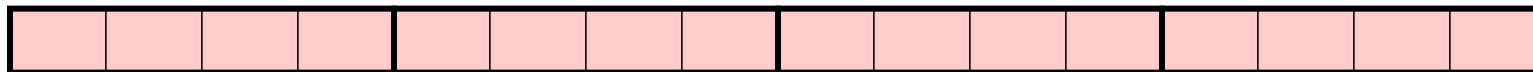
■ 16 single-byte integers



■ 8 16-bit integers



■ 4 32-bit integers



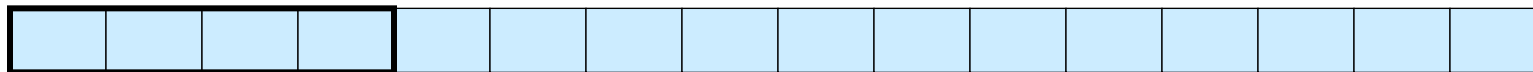
■ 4 single-precision floats



■ 2 double-precision floats



■ 1 single-precision float



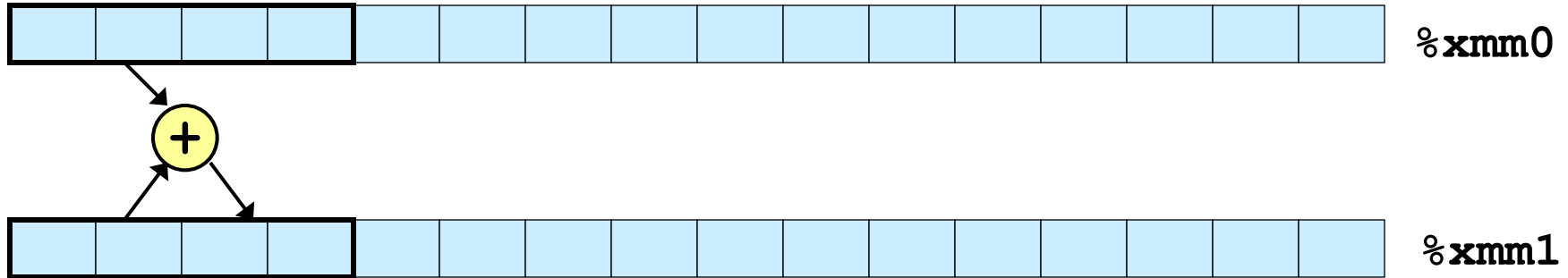
■ 1 double-precision float



Aside: Scalar & SIMD Operations

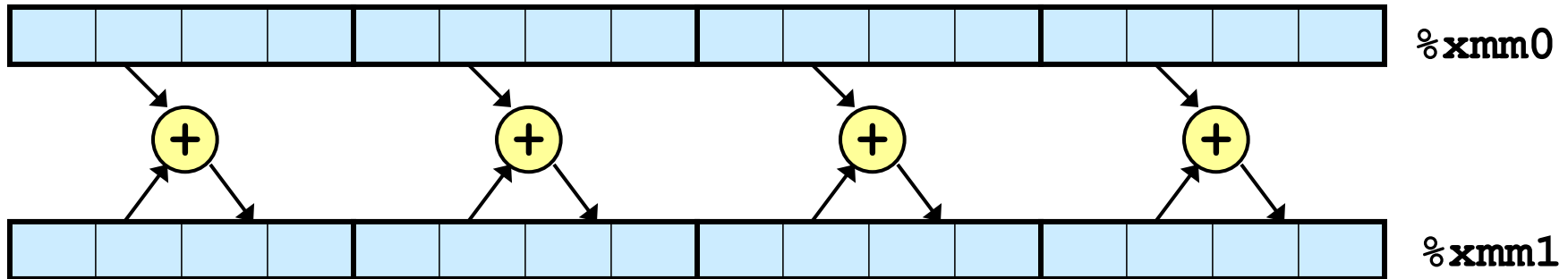
■ Scalar Operations: Single Precision

`addss %xmm0, %xmm1`



■ SIMD Operations: Single Precision

`addps %xmm0, %xmm1`



■ Scalar Operations: Double Precision

`addsd %xmm0, %xmm1`

