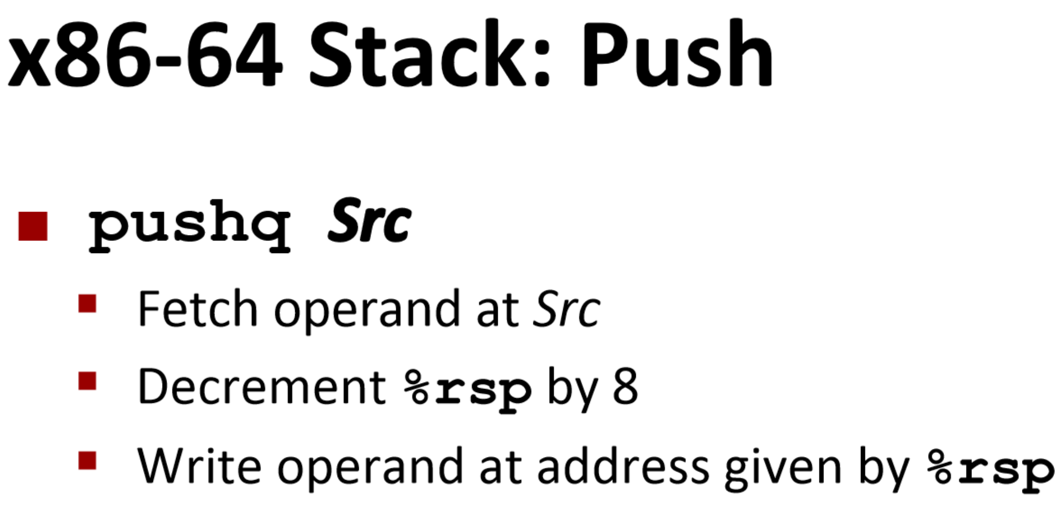
10.18

主要内容

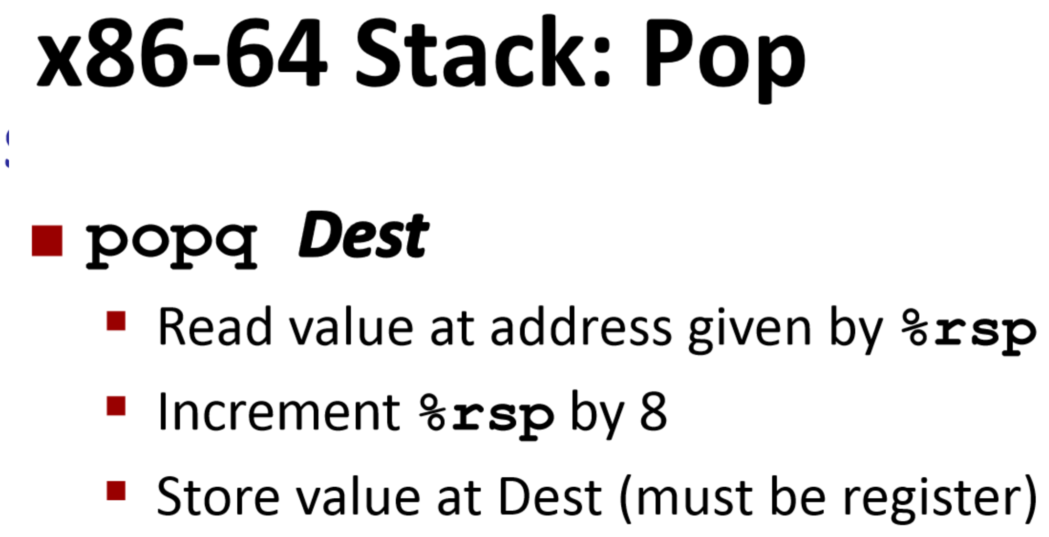
- 进程调用
    - push与pop
- 数据结构的表示
    - 内存对齐
    - 数组的表示和寻址
- 浮点数和向量指令

push和pop



x86-64 Stack: Push

- `pushq` ***Src***
  - Fetch operand at *Src*
  - Decrement `%rsp` by 8
  - Write operand at address given by `%rsp`

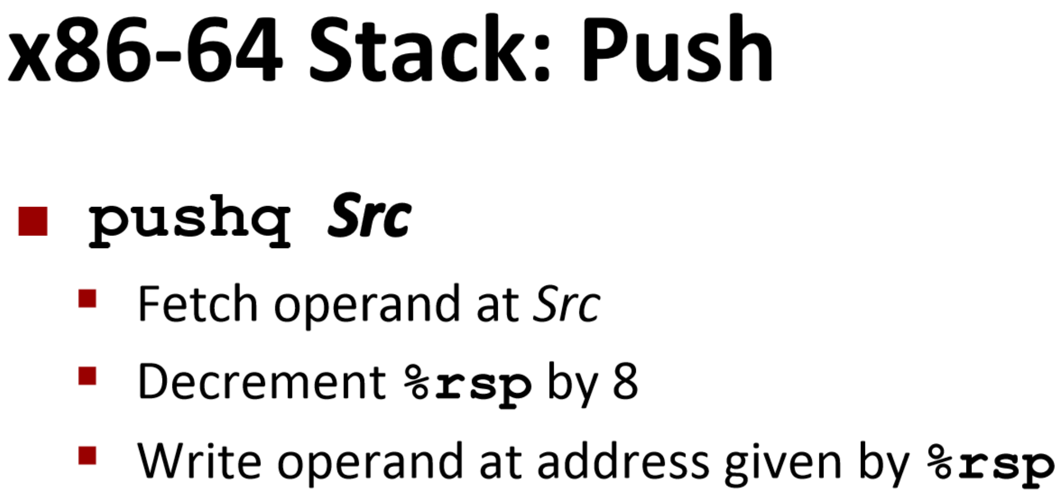


x86-64 Stack: Pop

- `popq` ***Dest***
  - Read value at address given by `%rsp`
  - Increment `%rsp` by 8
  - Store value at Dest (must be register)

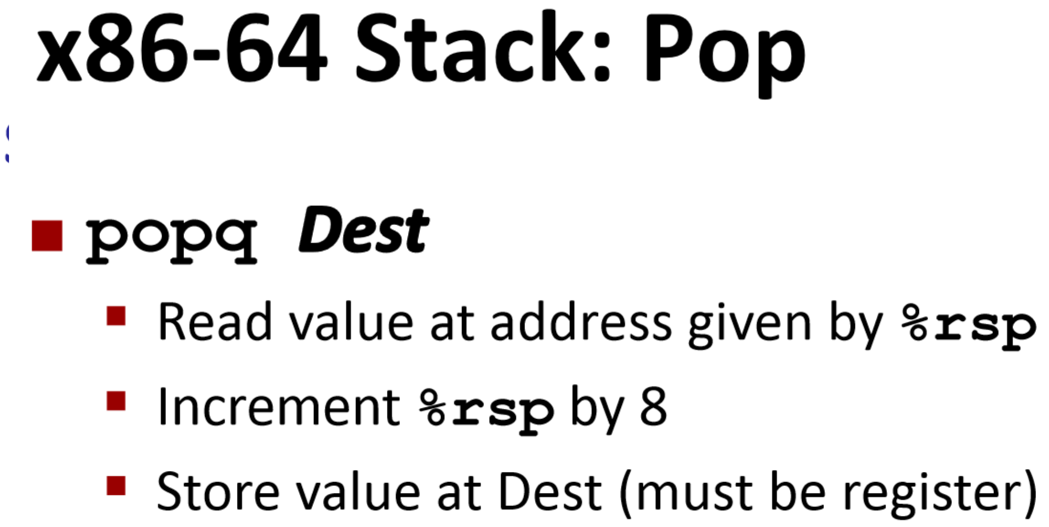
只要记住，%rsp指向的位置是存储了有用的信息的

push和pop



x86-64 Stack: Push

- `pushq` ***Src***
  - Fetch operand at *Src*
  - Decrement `%rsp` by 8
  - Write operand at address given by `%rsp`



x86-64 Stack: Pop

- `popq` ***Dest***
  - Read value at address given by `%rsp`
  - Increment `%rsp` by 8
  - Store value at Dest (must be register)

如果执行

```
pushq %rsp
popq %rsp
```

会怎么样?

数据结构的表示

- 为什么要内存对齐?
- 如果没有对齐，会发生什么?

*内存的最小单元是8bit，但是总线宽度是64bit

→ 但是这又有什么关系呢?

*在后面的向量操作中有用 → 广义上的“对齐”

数据结构的表示

- 数组的寻址模式（今天不说）
- 数组的表示方法
  - Nested数组 → 最简单
  - 如何解析一条形如 int *(*a)(int*, int)[5]; 的语句?
  - 从左往右，遇到括号变方向
  - “指针的数组”和“指向数组的指针”
- 最后还有一些题目

浮点数指令

- XMM寄存器与YMM寄存器
- 浮点数加减乘除指令
  - 没有考过，但还是建议记一下
  - s: single; d: double; ps/d: packed single/double
  - cvt(t): convert (with trunc); unpck: unpack

向量指令

- SSE 和 AVX
- 向量对应Vectorization，即一次操作多个数
  - 就类似向量加法，向量减法
- SIMD：Single Instruction Mutiple Data
- SSE与AVX的使用：大大提升程序性能

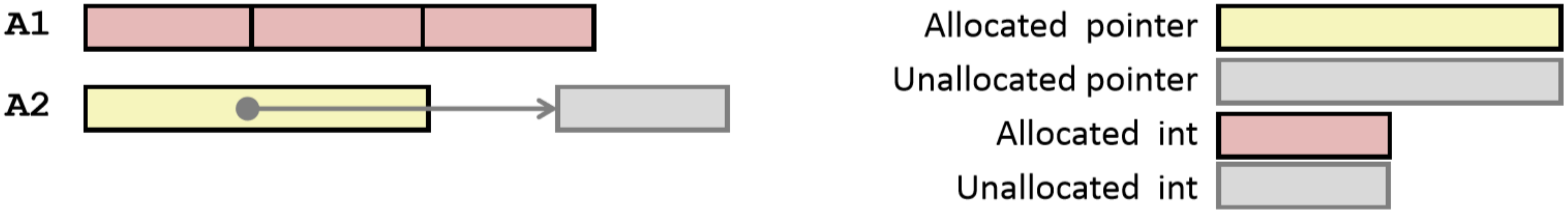
Understanding Pointers & Arrays #1

Decl	An			*An		
	Cmp	Bad	Size	Cmp	Bad	Size
int A1[3]						
int *A2						

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

Understanding Pointers & Arrays #1

Decl	An			*An		
	Cmp	Bad	Size	Cmp	Bad	Size
`int A1[3]`	Y	N	12	Y	N	4
`int *A2`	Y	N	8	Y	Y	4



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

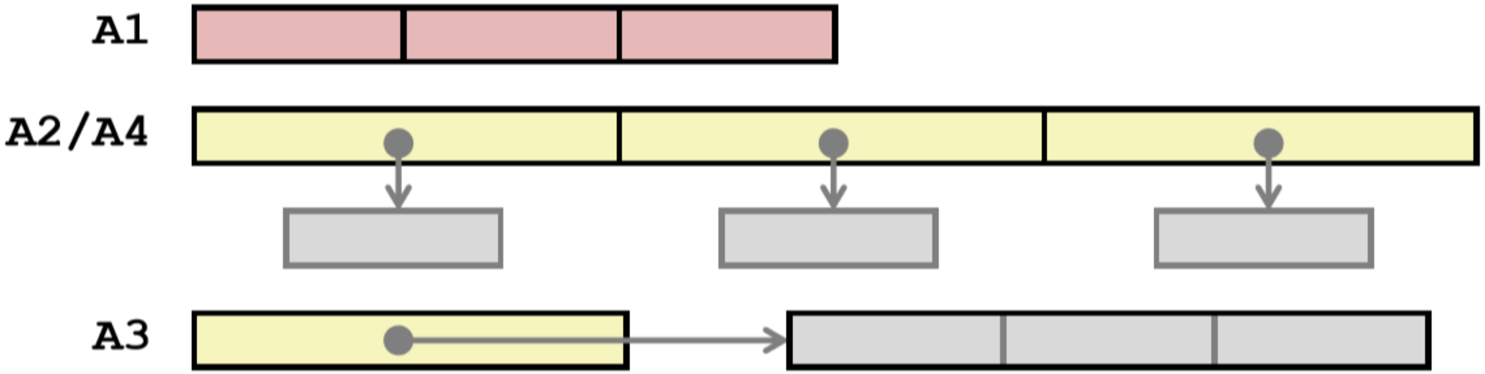
Understanding Pointers & Arrays #2

Decl	`An`			`*An`			`**An`		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
`int A1[3]`									
`int *A2[3]`									
`int (*A3)[3]`									
`int (*A4[3])`									

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

Understanding Pointers & Arrays #2

Decl	An			*An			**An		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
`int A1[3]`	Y	N	12	Y	N	4	N	-	-
`int *A2[3]`	Y	N	24	Y	N	8	Y	Y	4
`int (*A3)[3]`	Y	N	8	Y	Y	12	Y	Y	4
`int (*A4[3])`	Y	N	24	Y	N	8	Y	Y	4



Understanding Pointers & Arrays #3

Decl	An			*An			**An		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
`int A1[3][5]`									
`int *A2[3][5]`									
`int (*A3)[3][5]`									
`int *(A4[3][5])`									
`int (*A5[3])[5]`									

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

Decl	***An		
	Cmp	Bad	Size
`int A1[3][5]`			
`int *A2[3][5]`			
`int (*A3)[3][5]`			
`int *(A4[3][5])`			
`int (*A5[3])[5]`			

Allocated pointer 🟨

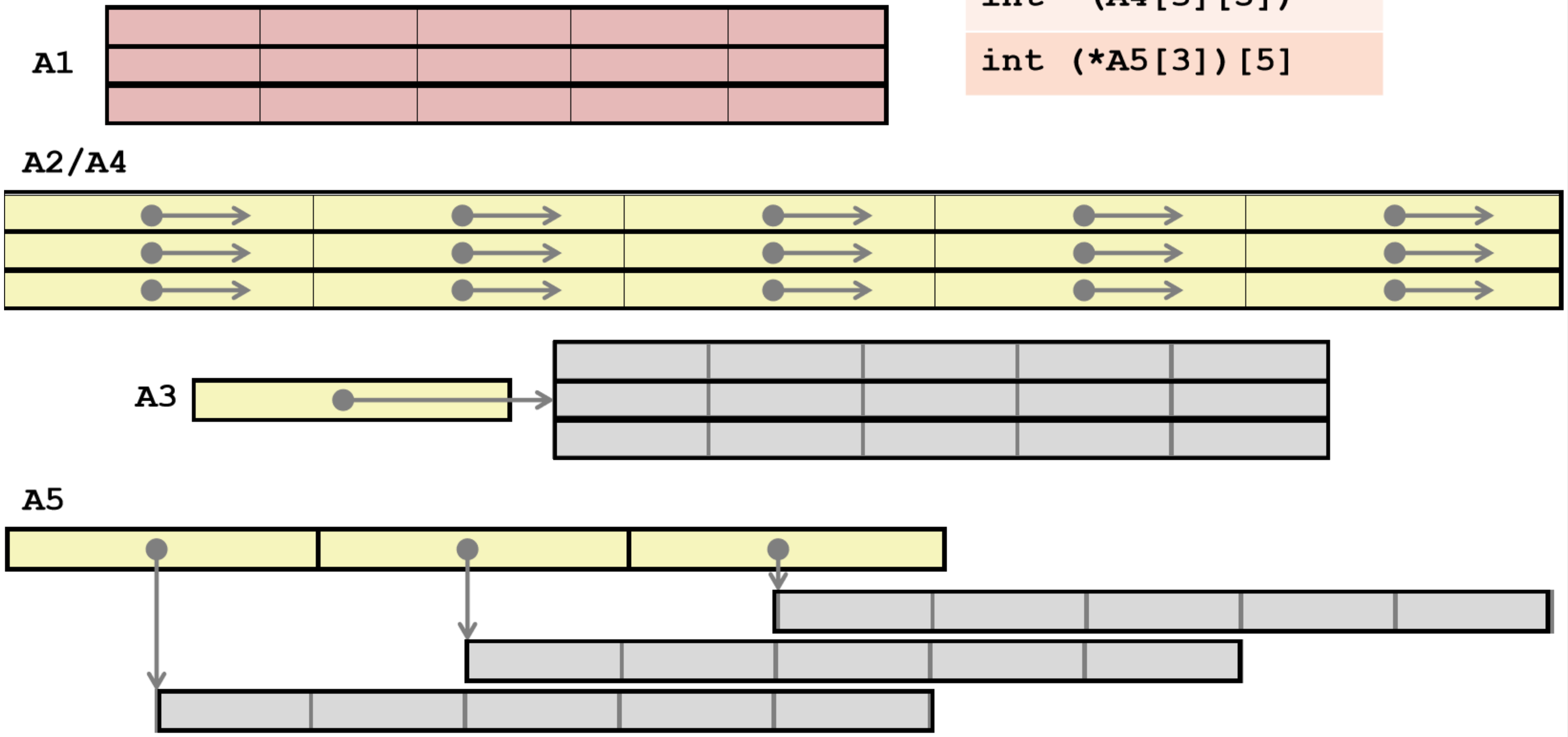
Allocated pointer to unallocated int 🟨 ●→

Unallocated pointer ⬜

Allocated int 🟥

Unallocated int ⬜

Declaration
`int A1[3][5]`
`int *A2[3][5]`
`int (*A3)[3][5]`
`int *(A4[3][5])`
`int (*A5[3])[5]`



Understanding Pointers & Arrays #3

Decl	*An*			*An			**An		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	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		
	Cmp	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