Roadmap

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
    c.getMPG();
```

Assembly language:

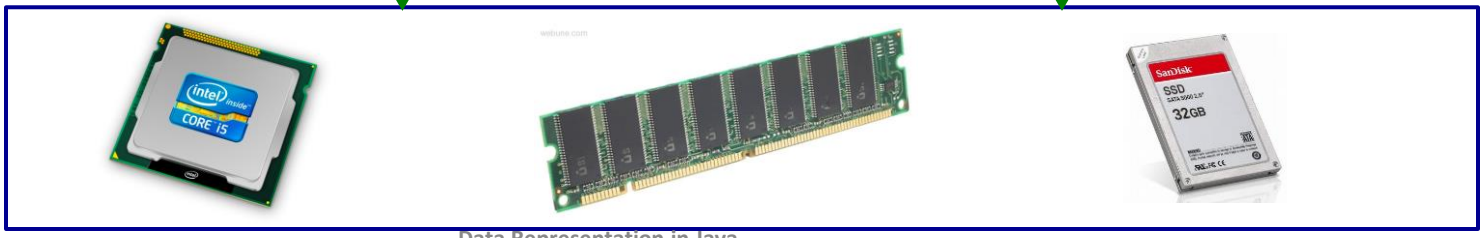
```
get_mpg:
    pushq   %rbp
    movq    %rsp, %rbp
    ...
    popq    %rbp
    ret
```

Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
11000001111110100011111
```

OS:

Computer system:



Memory & data
Integers & floats
Machine code & C
x86 assembly
Procedures & stacks
Arrays & structs
Memory & caches
Processes
Virtual memory
Memory allocation
Java vs. C

Section 11: Comparing Java and C

- Data representations in Java
- Pointers and references
- Method calls
- Virtual machines and runtime environment

- We've learned about the above in C, this section is about how it all works in Java
- But you have a lot more background now, so this tour will be much faster

Meta-point to this lecture

- None of the data representations we are going to talk about are *guaranteed* by Java
- In fact, the language simply provides an *abstraction*
- We can't easily tell how things are really represented
- But it is important to understand *an implementation* of the lower levels – useful in thinking about your program

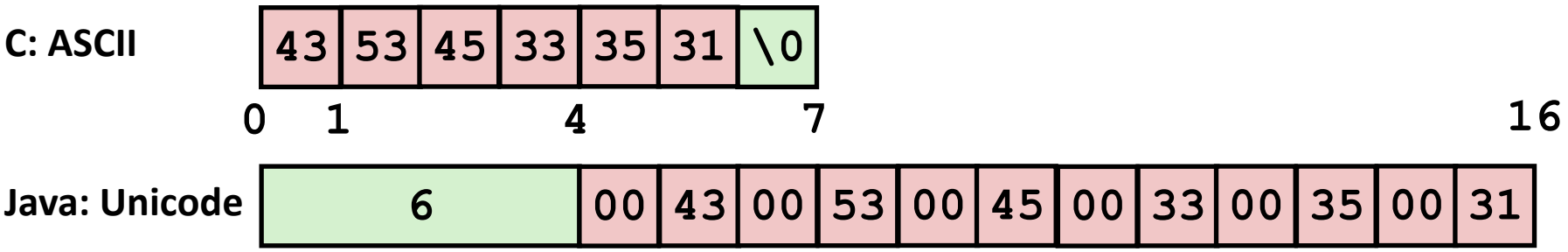
Data in Java

- **Integers, floats, doubles, pointers – same as C**
  - Yes, Java has pointers – they are called ‘references’ – however, Java references are much more constrained than C’s general pointers
- **Null is typically represented as 0**
- **Characters and strings**
- **Arrays**
- **Objects**

Data in Java

- **Characters and strings**
  - Two-byte Unicode instead of ASCII
    - Represents most of the world's alphabets
  - String not bounded by a '\0' (null character)
    - Bounded by hidden length field at beginning of string

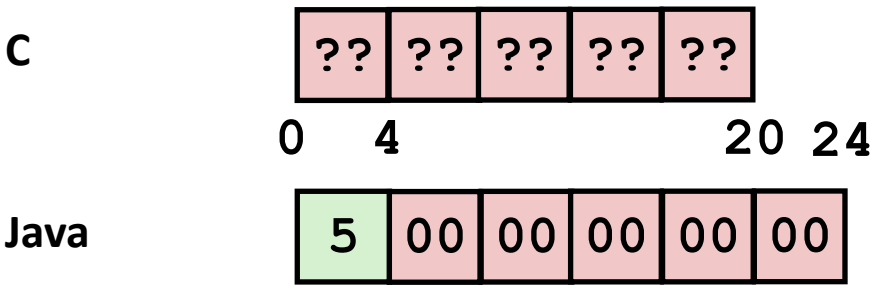
the string 'CSE351':



Data in Java

- **Arrays**
  - Every element initialized to 0
  - Bounds specified in hidden fields at start of array (int – 4 bytes)
    - *array.length* returns value of this field
    - *Hmm, since it has this info, what can it do?*

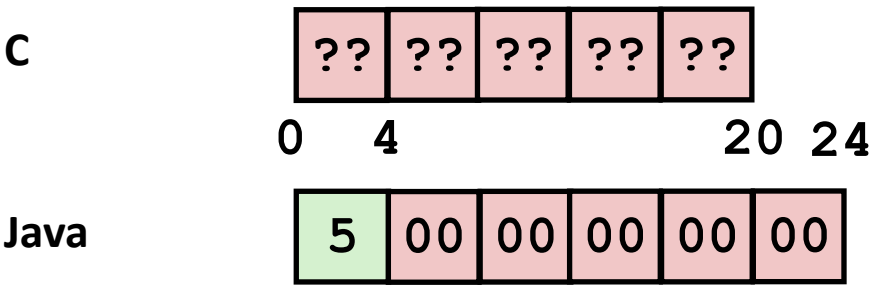
int array[5]:



Data in Java

- **Arrays**
  - Every element initialized to 0
  - Bounds specified in hidden fields at start of array (int – 4 bytes)
    - *array.length* returns value of this field
  - Every access triggers a bounds-check
    - Code is added to ensure the index is within bounds
    - Exception if out-of-bounds

int array[5]:



Data structures (objects) in Java

- **Objects (structs) can only include primitive data types**
  - Include complex data types (arrays, other objects, etc.) using *references*

C

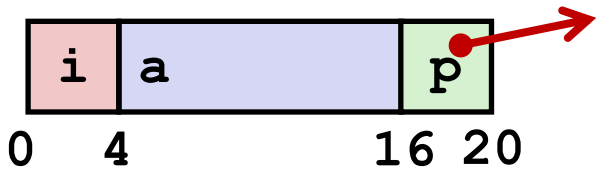
```
struct rec {
  int i;
  int a[3];
  struct rec *p;
};
```

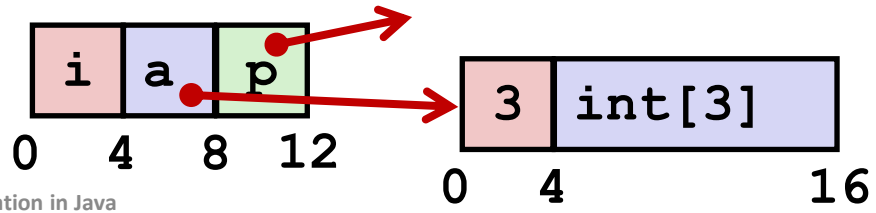
Java

```
class Rec {
  int i;
  int[] a = new int[3];
  Rec p;
…
};
```

```
struct rec *r = malloc(...);
struct rec r2;
r->i = val;
r->a[2] = val;
r->p = &r2;
```

```
r = new Rec;
r2 = new Rec;
r.i = val;
r.a[2] = val;
r.p = r2;
```



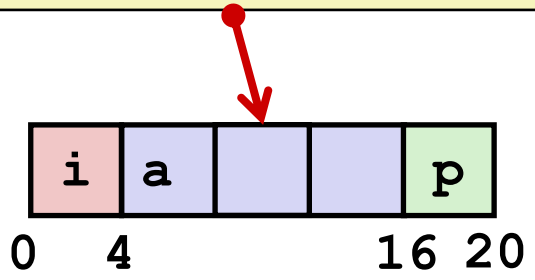


Pointers/References

- **Pointers in C can point to any memory address**
- **References in Java can only point to an object**
  - And only to its first element – not to the middle of it

C

```
struct rec {
  int i;
  int a[3];
  struct rec *p;
};
some_fn(&(r.a[1]))   //ptr
```



Java

```
class Rec {
  int i;
  int[] a = new int[3];
  Rec p;
…
};
some_fn(r.a, 1)   //ref & index
```