Section 11: Comparing Java and C

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

Pointers to fields

- **In C, we have "->" and "." for field selection depending on whether we have a pointer to a struct or a struct**
  - (*r).a is so common it becomes r->a

- **In Java, *all variables are references to objects***
  - We always use r.a notation
  - But really follow reference to r with offset to a, just like C's r->a

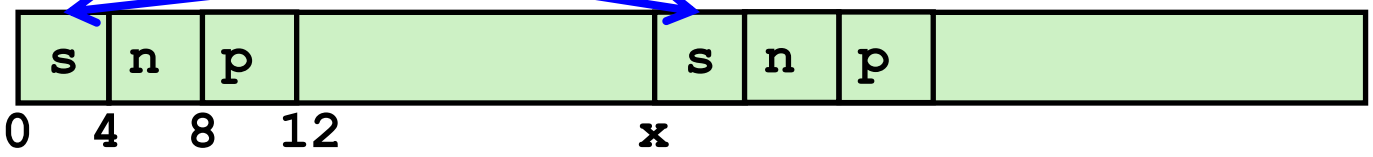
Casting in C

- We can cast any pointer into any other pointer

```
struct BlockInfo {
        int sizeAndTags;
        struct BlockInfo* next;
        struct BlockInfo* prev;
};
typedef struct BlockInfo BlockInfo;
…
int x;
BlockInfo *b;
BlockInfo *newBlock;
…
newBlock = (BlockInfo *) ( (char *) b + x );
…
```

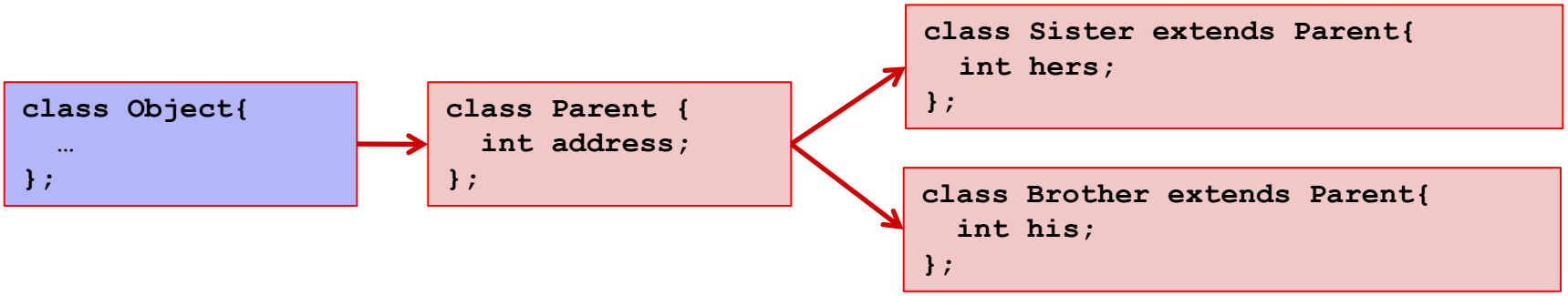
Cast b into char pointer so that you can add byte offset without scaling

Cast back into BlockInfo pointer so you can use it as BlockInfo struct



Casting in Java

- Can only cast compatible object references

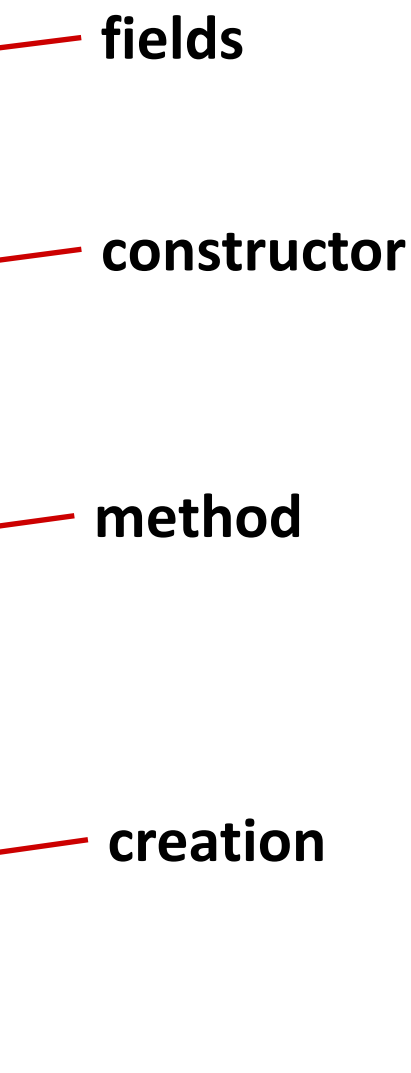



```
// Parent is a super class of Brother and Sister, which are siblings
Parent    a = new Parent();
Sister   xx = new Sister();
Brother  xy = new Brother();
Parent   p1 = new Sister();          // ok, everything needed for Parent
                                     // is also in Sister
Parent   p2 = p1;                    // ok, p1 is already a Parent
Sister  xx2 = new Brother();         // incompatible type – Brother and
                                     // Sisters are siblings
Sister  xx3 = new Parent();          // wrong direction; elements in Sister
                                     // not in Parent (hers)
Brother xy2 = (Brother) a;           // run-time error; Parent does not contain
                                     // all elements in Brother (his)
Sister  xx4 = (Sister) p2;  // ok, p2 started out as Sister
Sister  xx5 = (Sister) xy;  // inconvertible types, xy is Brother
```

How is this implemented / enforced?

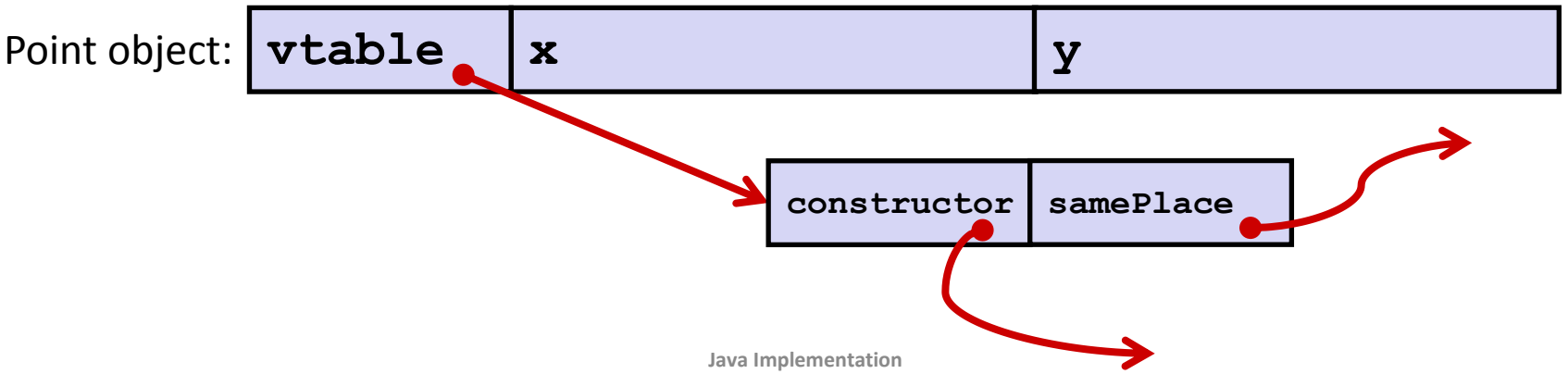
Creating objects in Java

```
class Point {
        double x;
        double y;

Point() {
        x = 0;
        y = 0;
    }

boolean samePlace(Point p) {
        return (x == p.x) && (y == p.y);
    }

}
…
Point newPoint = new Point();
…
```



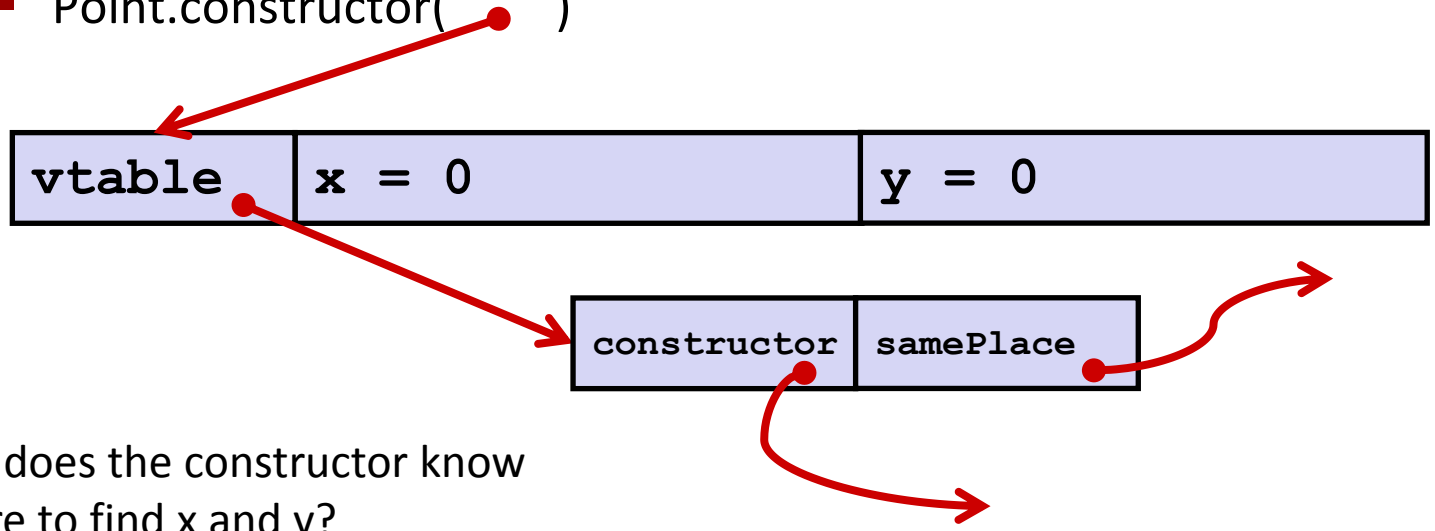
Creating objects in Java

- **"new"**
  - Allocates space for data fields
  - Adds pointer in object to "virtual table" or "vtable" for class
    - vtable is shared across all objects in the class!
    - Includes space for "static fields" and pointers to methods' code
  - Returns reference (pointer) to new object in memory
  - Runs "constructor" method
- **The new object is eventually garbage collected if all references to it are discarded**



Initialization

- newPoint's fields are initialized starting with the vtable pointer to the vtable for this class
- The next step is to call the 'constructor' for this object type
- Constructor code is found using the 'vtable pointer' and passed a pointer to the newly allocated memory area for newPoint so that the constructor can set its x and y to 0
  - Point.constructor( )



How does the constructor know where to find x and y?

Java Methods

- **Methods in Java are just functions (as in C) but with an extra argument: a reference to the object whose method is being called**
  - E.g., newPoint.samePlace calls the samePlace method with a pointer to newPoint (called 'this') and a pointer to the argument, p – in this case, both of these are pointers to objects of type Point
  - Method becomes Point.samePlace(Point this, Point p)
    - return x==p.x && y==p.y; becomes something like:
      return (this->x==p->x) && (this->y==p->y);

Subclassing

```
class PtSubClass extends Point{
  int aNewField;
  boolean samePlace(Point p2) {
     return false;
  }
  void sayHi() {
    System.out.println("hello");
  }
}
```

- **Where does "aNewField" go?**
  - At end of fields of Point – allows easy casting from subclass to parent class!
- **Where does pointer to code for two new methods go?**
  - To override "samePlace", write over old pointer
  - Add new pointer at end of table for new method "sayHi"

Subclassing

```
class PtSubClass extends Point{
   int aNewField;
   boolean samePlace(Point p2) {
      return false;
   }
   void sayHi() {
     System.out.println("hello");
   }
 }
```