(lab 3) usage of mats

material_list_print(
mats, stdout)

mat = material_
(getbyname(mats,
"blue"))

material_t * mat

```
void
material_list_print(
    list_t *list, FILE *out)
{
    material_t *mat;

    // reset internal list
    iterator
    list_reset(list);
```

```
while (list_not_end(list)) {
    mat =
    (material_t *)
    list_get_data(list);
    material_print(
        mat, out);
    list_next_link(list);
}
```

material_print(material_t
*mat, FILE *out)
{
assert(mat->cookie ==
MAT_COOKIE);

(#define MAT_COOKIE
3245123
in material.h

```
typedef struct material_type
{ int cookie;
  char name[NAME_LEN];
  rgb_t ambient;
  rgb_t diffuse;
  rgb_t specular;
} material_t;
```

like

```
typedef double rgb_t[3];
```

} vec_t in pixel.h

pixel.h:

```
void pix_sum(
        digit_t p1, digit_t p2,
        digit_t p3);
        :
        :
int pix_nonzero(digit_t
                pix);
```

back to material_print:

```
fprintf(out, "material %s\n",
        mat->name);
fprintf(out "{\n");
if (pix_nonzero(mat->
                ambient)) {
    fprintfloat, "    ");
    PIX_print(out, "ambient",
              mat->ambient);
}
```

:

save f: diffuse, specular

fprintf(out, "}\n\n");

}// material_print

```
pix_print( . . . )
{   int i;
fprint(out, "%s", label);
for(i=0; i<3; i++)
   fprintf(out, "%2.0lf ", pix[i]);
fprintf(out, "\n");
}
```

```
int pix_nonzero(digit_t pix)
{
  // as soon as non-zero
     element is found, return 1
        int i;
  for(i=0; i<3; i++)
     if( pix[i] != 0.0 )
        return(1);
  return(0);
```

pix[i] != 0.0

Could be 0.000000000...3

if above will be TRUE

when semantically it

should be false

→ due to limited

precision of digital

architecture

double precision = 0.0000001;

~~if (pix[i] != 0) return (1);~~

if (fabs(pix[i]) > precision)
return (1);

include
<math.h>

note: normally,

if (fabs(x) < precision)

is used to test for

equality to 0.0

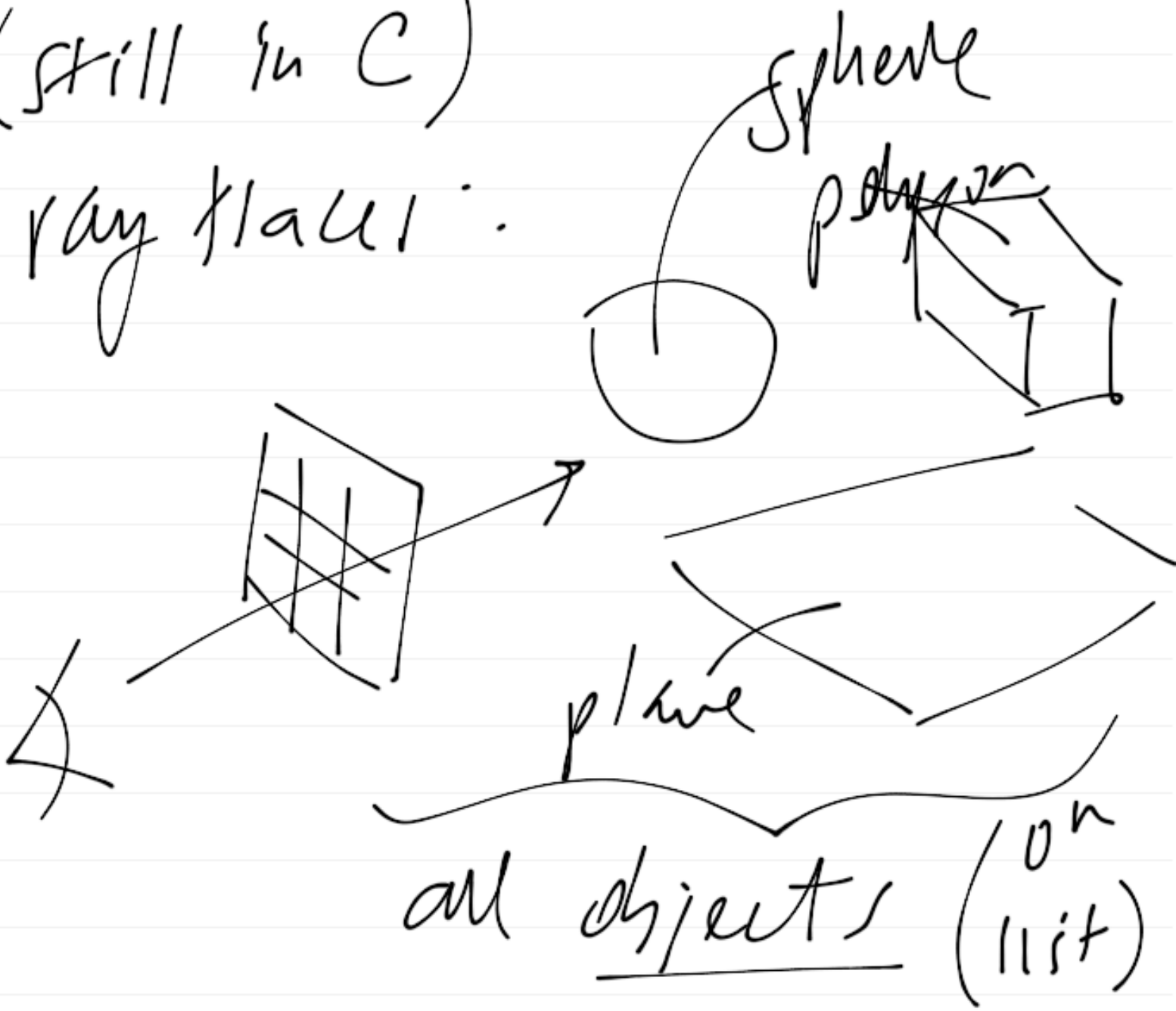
(floats & doubles)

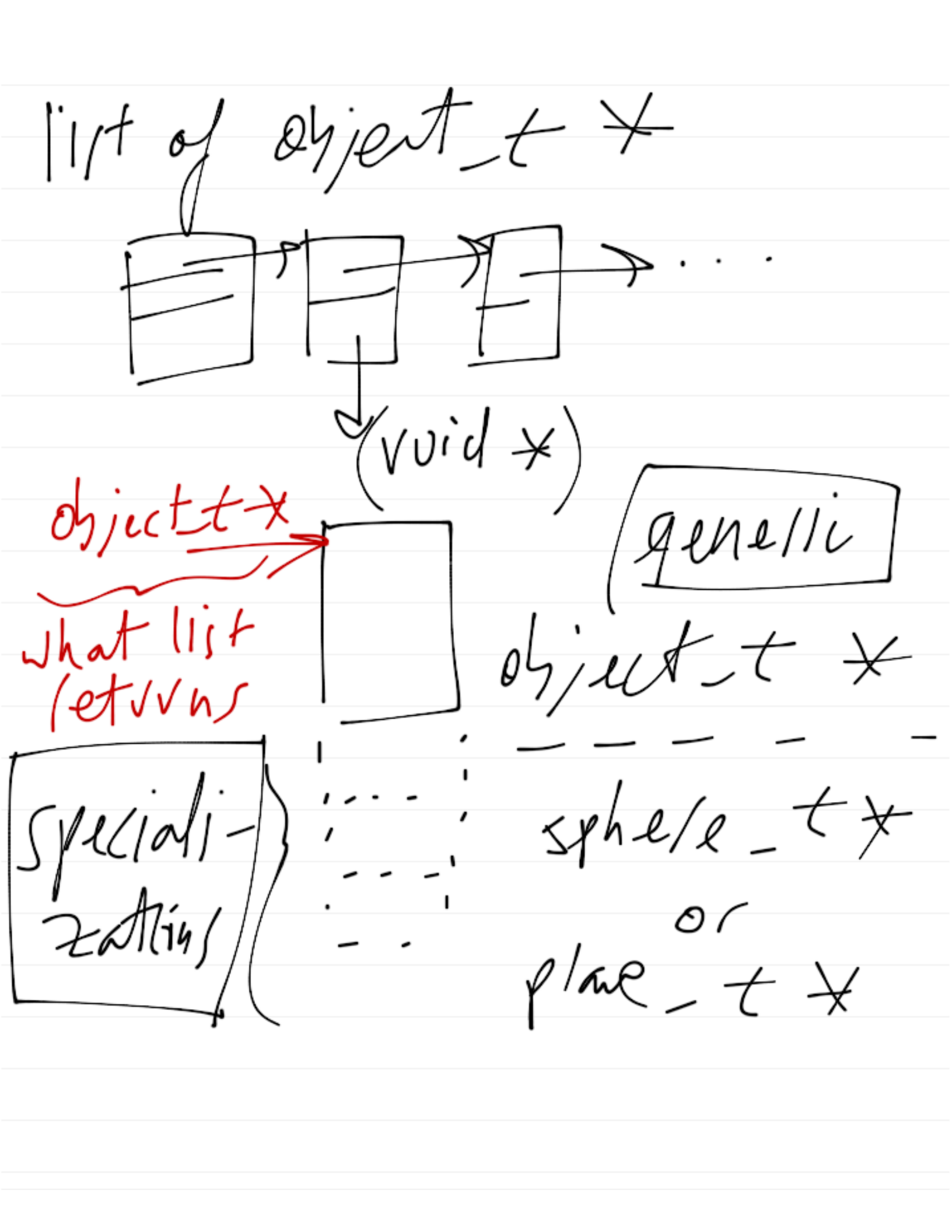
Object-oriented programming

(OOP)

(still in C)

ray tracer.




list of object_t *
(void *)
object_t *
what list returns
generic
object_t *
specializations
sphere_t *
or
plane_t *

```
object_type:
    int cookie; // OBJ_COOKIE
                // 12345678
    char type[NAME_LEN];
        //sphere, plane, label
    char name[NAME_LEN];
        // wall, floor, ...
    material_t *mat;
    vec_t last_hit;
    vec_t last_normal;
```

object_t is generic,

each of sphere_t, planet,

etc. INHERIT

attributes from object_t

⇒ INHERITANCE

(sphere_t is a subclass

derived from object_t

parent class)

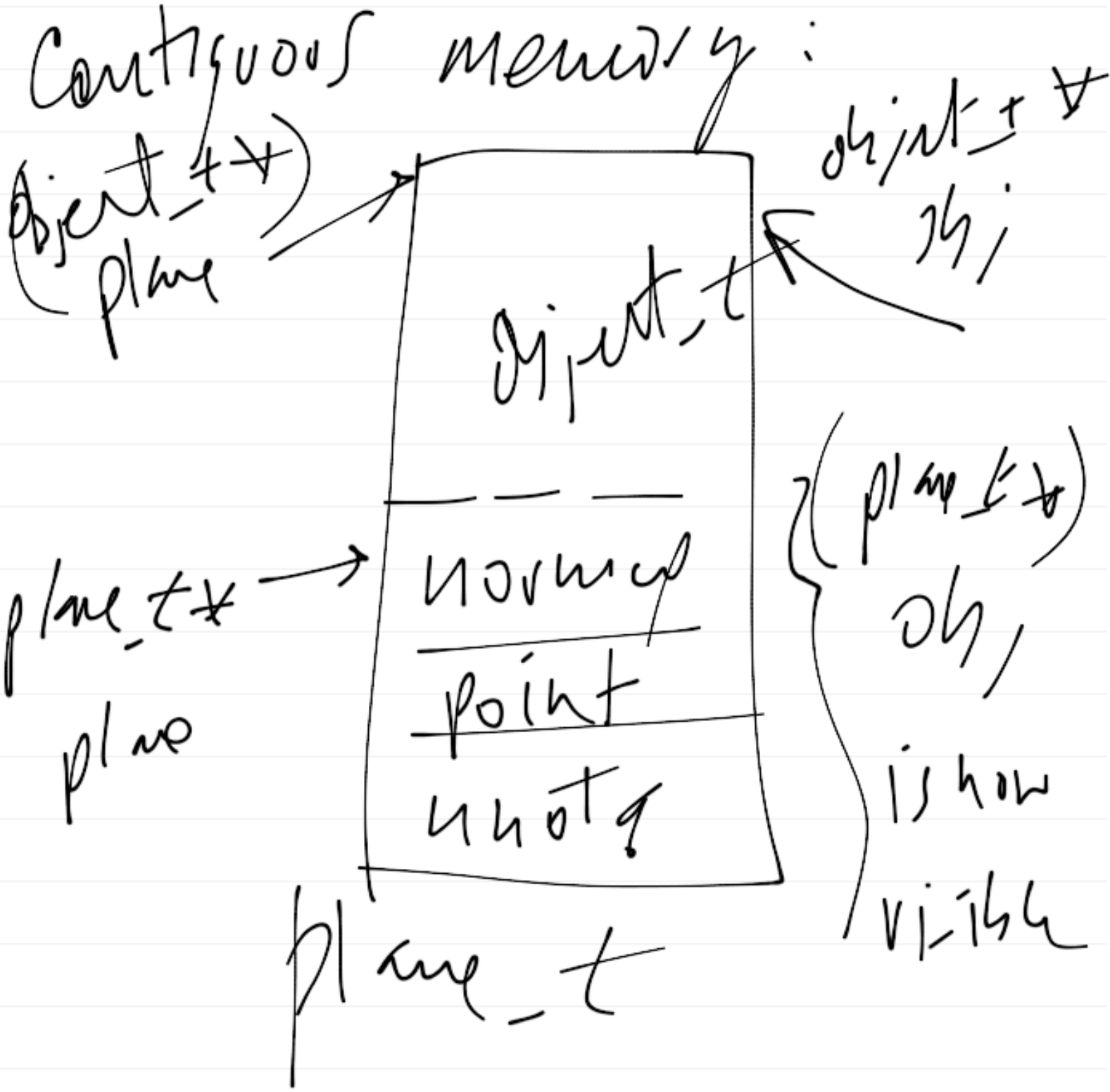
Note:

The notes (cmsystem.pdf) talk about a *pli pointer — don't use this

---

For us, key is for any subclass to include parent typ at top

```
typedef struct plane_type
{
    object_t obj; // parent
    vec_t normal;
    vec_t point;
    double ndotp;
} plane_t;
```


Contiguous memory:
object_t* plane
object_t* obj
object_t
---
normal
point
unot
plane_t*
plane
(plane_t*) obj
is now visible
plane_t

```
place_t *place;
object_t *obj;

    :

read in place
obj = (object_t *)place;
put obj on list
(cast as (void *))
```

each Object also has function pointers that each specialized object defines differently.

POLYMORPHISM

e.g. obj → print()