last time:

fabs(x) < epsilon

in general,

fabs(a - b) < epsilon

$|a-b| < \varepsilon$

precision (0.00001)

test for equality between floating point variables

polymorphism: object_t *

- as we iterate thru objs

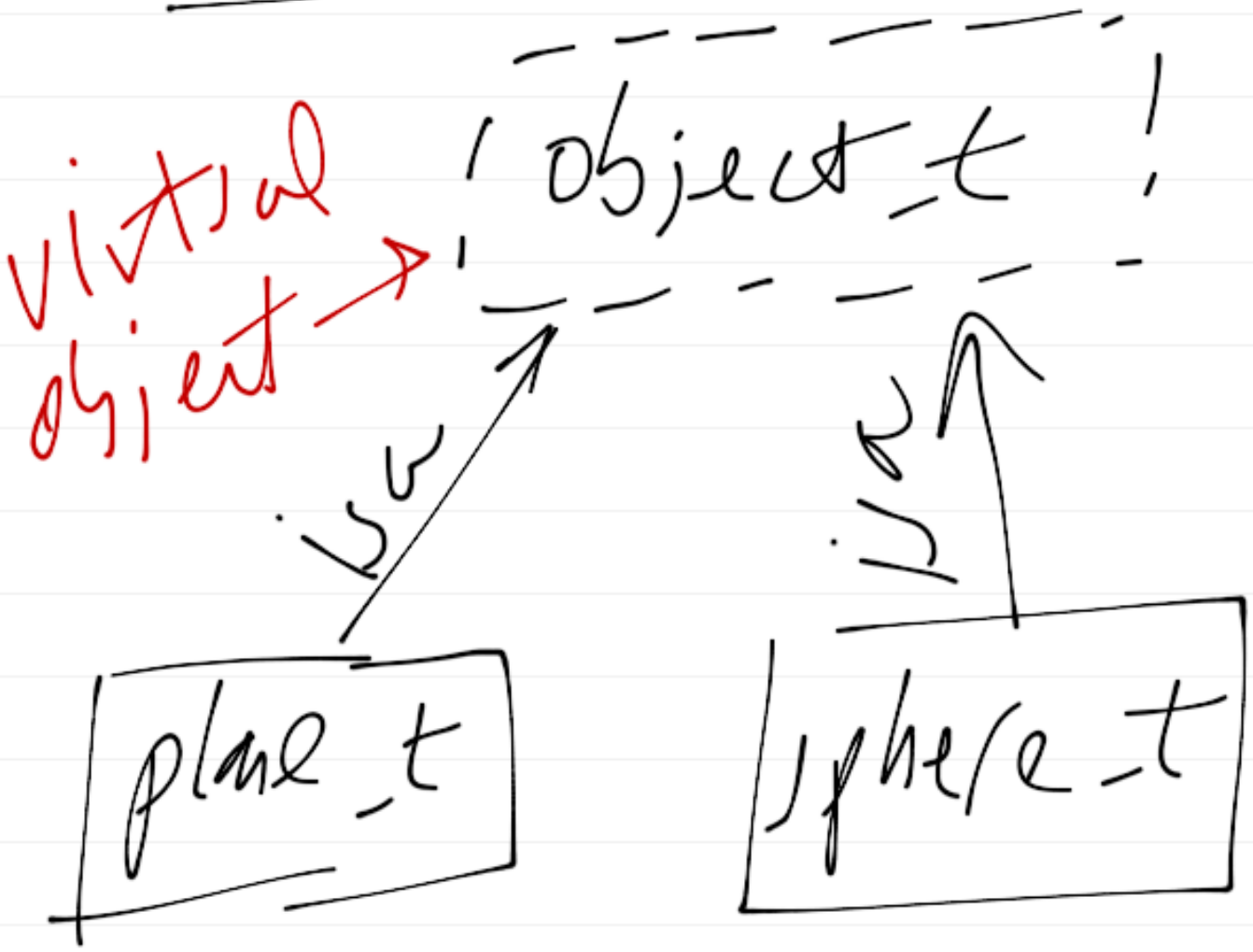
list,

```
object_t * obj;

list_reset(objs);
while(list_not_end(objs)) {
    obj = (object_t *) list_
cast    list_get_data(objs);
    obj->printer(obj, out);
    list_next_int(objs);
}
```

obj → printer (object_t *,
FILE *)

a "virtual method"



Virtual object:

User (programmer) should never (be allowed to)* allocate an object_t

* in C++ this is enforced by compiler

back to obj->printer()

object_t :

printer :

'just prints

obj: type = '{'

& material

but that's it —

incomplete output

object_t :

object_print(. .)

does as above, i.e.,

" plane floor

{

material tag "

---

virtual function printer() :

- calls parent fcn

obj → object_print()

- then prints own stuff

sphere.c:

```
sphere_print (
        object_t *obj,
        FILE *fd)
{
object_print(obj);
sphere_t *sphere =
  (sphere_t *) obj;
// print stuff specific to
sphere (center, radius, "}
```

- recall, we are iterating thru objs list, casting (void *) ptr to (object_t *) ptr

& calling

obj → printer()

how is it that sphere_print() gets called?

obj->printer() is a
function pointer
(pointer to a function)

obj->printer() will point to
plane_print()
or sphere_print()
polymorphic function

Object.h // parent class

```
#define OBJ_COOKIE 12345678
typedef struct object_type
{
  int cookie;
  char type[NAME_LEN];
  char name[ ii ];
  matrix_t *mat;
  vec_t last_hit;
  vec_t last_normal;
```

Actor members

```
void (*printer)(struct
                object_type *,
                FILE *);

void (*ambient)
     (material_t *, dif_t);

same for
     diffuse, specular
} object_t;
```

eventually,

obj → printer

is set to plane_print()

or sphere_print()

depending on which

object type we read in

plane.h :

```
typedef struct plane_type
{
    object_t □obj;      } NOT a pointer
    vec_t    normal;
    vec_t    point;
    double   ndotq;
} plane_t;
```

still in plane.h:

```
void plane_init(FILE*,
        list_t*, list_t**,
        int);
void plane_print(object_t**,
        FILE*);
```

plane.c :

void plane_init(FILE *in,
list_t *objs, list_t *
mats, int attrmax)

↑ ignore

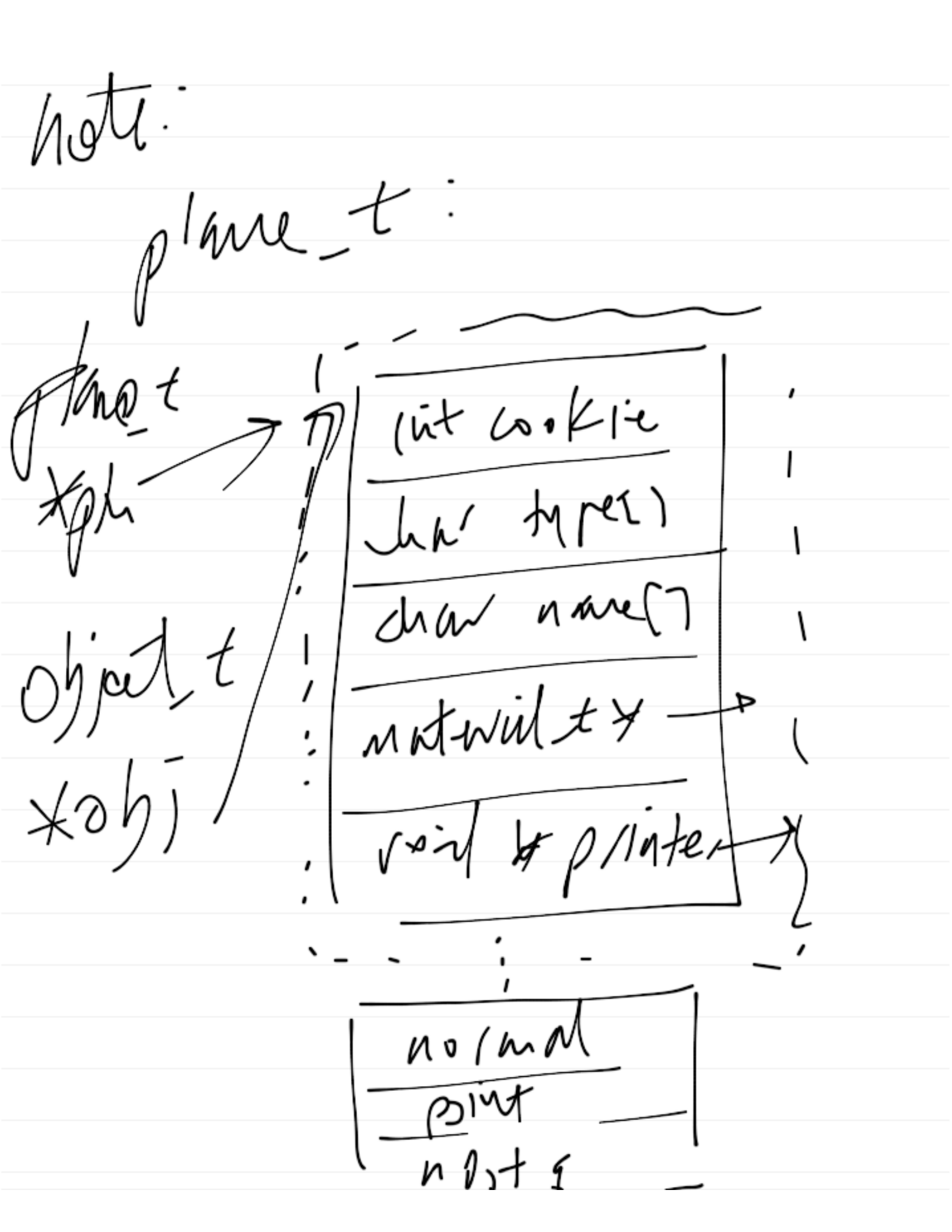
// objective: allocate & init
plane_t * & read in
attributes, & set mat ptr
& put itself on list

```
plane_t *pln;
object_t *obj;
char c, attr_name[NAME_LEN];

// allocate & init plane_t
assert((pln = (plane_t *)
    malloc(sizeof(plane_t))) !=
        NULL);
memset(pln, 0, sizeof(plane_t));
```

// init object (generic structure)

```
object_init( obj =
    (object_t *) pln,
    in, objs, mats);
(let object_t init =
read in object_t portion
of memory
```


note:
plane_t:
plane_t
*ph
object_t
*obj
int cookie
char type[]
char name[]
material_t *
void * printer
normal
print
next

// note: object_init
puts us (plane_t *)
on objs list as a
generic (object_t *)

```
assert((pln=(plane_t*)
  list_get_data(objs))!=
        NULL))
assert(obj->cookie ==
        OBJ_COOKIE);
```

```
// set object type
strcpy(obj->type, "plane");

// overload object's
   // "virtual" functions
obj->printer = plane_print;

// continue parsing file
```

```
while((c=fgetc(in))
    != EOF && c != '}')
  fputc(c, in);
assert((count =
    fscanf(in, "%s",
        attr/name)) == 1);
```

```
if (!strcmp(attrname,
        "normal"))
    vec_read(in, pln->normal);
if (!strcmp(attrname,
        "point"))
    vec_read(in, pln->point);
while ((c=fgetc(in)) != EOF
    && c != '\n');
}
```

```
VEC_UNIT( pln->normal,
    ob[i]->plane_normal);

VEC_UNIT(pln->normal,
        pln->normal);

pln->ndotc = VEC_DOT(
    pln->point, pln->normal);
}  /* plane_init */
```