

Pointer Swizzling (refer to the readings)

In order to speed up the access along interobject references, the persistent pointers in the form of unique object identifiers (OIDs) are transformed (swizzled) into main memory pointers (addresses).

- Resident Object Table (ROT)
completely resident in main memory.
- Persistent Object Table (POT):
It is only needed when logical object identifiers are used.
partially resident in main memory.

See Fig 22.1

```
myCuboid.v1.x;  
myCuboid.mat.specWeight;
```

22.2 *The Goal of Pointer Swizzling*

- The basic idea of **pointer swizzling** is to materialise the address of main memory resident persistent objects in order to avoid the lookup in the table ROT to localize the object each time it is accessed.
- Pointer swizzling converts database objects from an external (persistent, disk address) form into an internal (main memory address) form.
- Unswizzling converts an internal form into an external form.

22.3 *Classification of Pointers Swizzling*

- In place/Copy: In the same page/Copy to a separate object buffer.
- Eager/Lazy
- Direct/Indirect: Through a “descriptor” or not.

22.4 In Place and Copy Swizzling

- The secondary storage is organized in pages.
- Objects are mapped onto pages.
- An object must have to fit on one page.
- Pages are the transport units between secondary and the main memories.

Two buffers or one buffer

- A page buffer only: No extra copy costs between two buffer.
 - Objects whose pointers are swizzled remain on their original place.
 - All pages that contain modified objects have to be written back to secondary storage.
 - Unmodified objects have to be unswizzled if at least one object on the page is modified.
- A page buffer and an Object buffer:
 - Incur copy costs. But, it is better if only a few objects are needed.
 - No need to unswizzle unmodified objects.
 - Only the swizzled pointers contained in modified objects have to be unswizzled.

22.5 *Eager and Lazy Swizzling*

- Eager Swizzling
 - All the pointers in main memory are swizzled.
 - When an object is loaded from disk, the object is “scanned through” and all pointers the object contains are immediately swizzled.
- Lazy Swizzling
 - Swizzle pointer only on demand.
 - Have to handle two different kinds of pointers at run time: swizzled and non-swizzled pointers.
Need to check the state of pointer each time an object is accessed.

22.6 *Direct and Indirect Swizzling*

Direct Swizzling

- A directly swizzled pointer contains the main memory address of the object it references to.
- When an object is displaced, all the directly swizzled pointers that reference the displaced object need to be unswizzled.
- Reverse reference list (RRL) is needed. It can be very costly if the degree of sharing of an object is very high. For example, an attribute of an object is assigned to a new value.

See Figure 22.5

- In case of eager direct swizzling, “snowball effect”.

Indirect Swizzling

- A swizzled pointer materialises the address of a descriptor — a placeholder of the actual object.
 - If the object is not in main–memory, its descriptor is marked as invalid.
 - If the object is displaced, the swizzled pointers that reference the object need not be unswizzled.
- A descriptor keeps a counter counting the number of indirectly swizzled pointers pointing to the descriptor. Maintaining this counter is much cheaper than maintaining the RRL.
- Induce an additional overhead” descriptor, invalid or not, etc.