* Interesting point regarding the tracking of shared memory modifications. By keeping a copy of a string fetched out of shared memory for longer than the transaction boundary, the ref count on the string might be decremented outside of the transactional operation, leading to deletion of shared memory outside of the transaction. That’s a problem.
* So there is a rule: **any attempt to allocate/deallocate memory within a region that is in use requires a transaction to be within scope.**
* Basic Algorithm:
  + Allocations occur immediately.
  + De-allocations are held until commit to avoid loosing data.
  + When committing:
    - all uncommitted allocations are written with the txn redo log
    - all deallocations are returned to the free pool, and those changes are also written to the redo log at that point.
    - Note: might be able to hold off on writing the deallocation data until the subsequent transactions.
* Example:
  + Txn1 – allocate block1
  + Txn2 – modify region1
  + Txn1 – modify region2
  + Txn2 – allocate block2
  + Txn2 – deallocate block3 (queued)
  + Txn2 – commit:
    - Redo Log: allocation of blocks 1 and 2, mod to region 1.
    - Block 3 returned to free list for use by other txns. That change itself is added to the ‘needs commit’ queue that allocations are also being tracked with.
* The region itself is a shared memory region (non-persistent) which is backed by a separate standard file (not memory mapped.) Alternatively, if mmap is really fast on some system (for some reason), the region could be handled as a set of memory mappings, each a page in size. (Unlikely)
* Shared region of N pages.
* File of M pages. M>>N.

Adjacent shared memory region provides fast TLB-type access to the current location of a page.

Contents (Logical):

* + The LRU ordering of the pages for paging purposes. (Only pages without uncommitted transactions can be paged out.)
  + The address in the shared region of each of the M pages in the database.
  + For each page: The list of transactions which currently have an uncommitted change on that page.

Contents (Physical):

* + “intrusive\_dlist\* [M]” in the shared region – TLB to map page numbers to addresses within the mapped shared region. Array lookup + math.
  + intrusive\_dlist\_node[N] – an array used to identify information about each Intrusive\_dlist\_node contains:
    - The page’s offset in the file.
    - offset in the shared region where the page can be found
    - vector<txn\_id> - the transactions which currently have uncommitted changes on this page.
  + intrusive\_dlist - linked list representing the LRU ordering of pages within the region according to last access times. Holds only those pages which can be paged out. This is used to determine which page to drop during page-in.
  + map<txn\_id,vector<intrusive\_dlist\_node\*> > - map of the pages which currently have uncommitted changes on them for each outstanding transaction in the system. Transactions remember their particular node in this map, so an entry is added upon the first change to a page, and that entry is removed at the end of the transaction, but that is all.