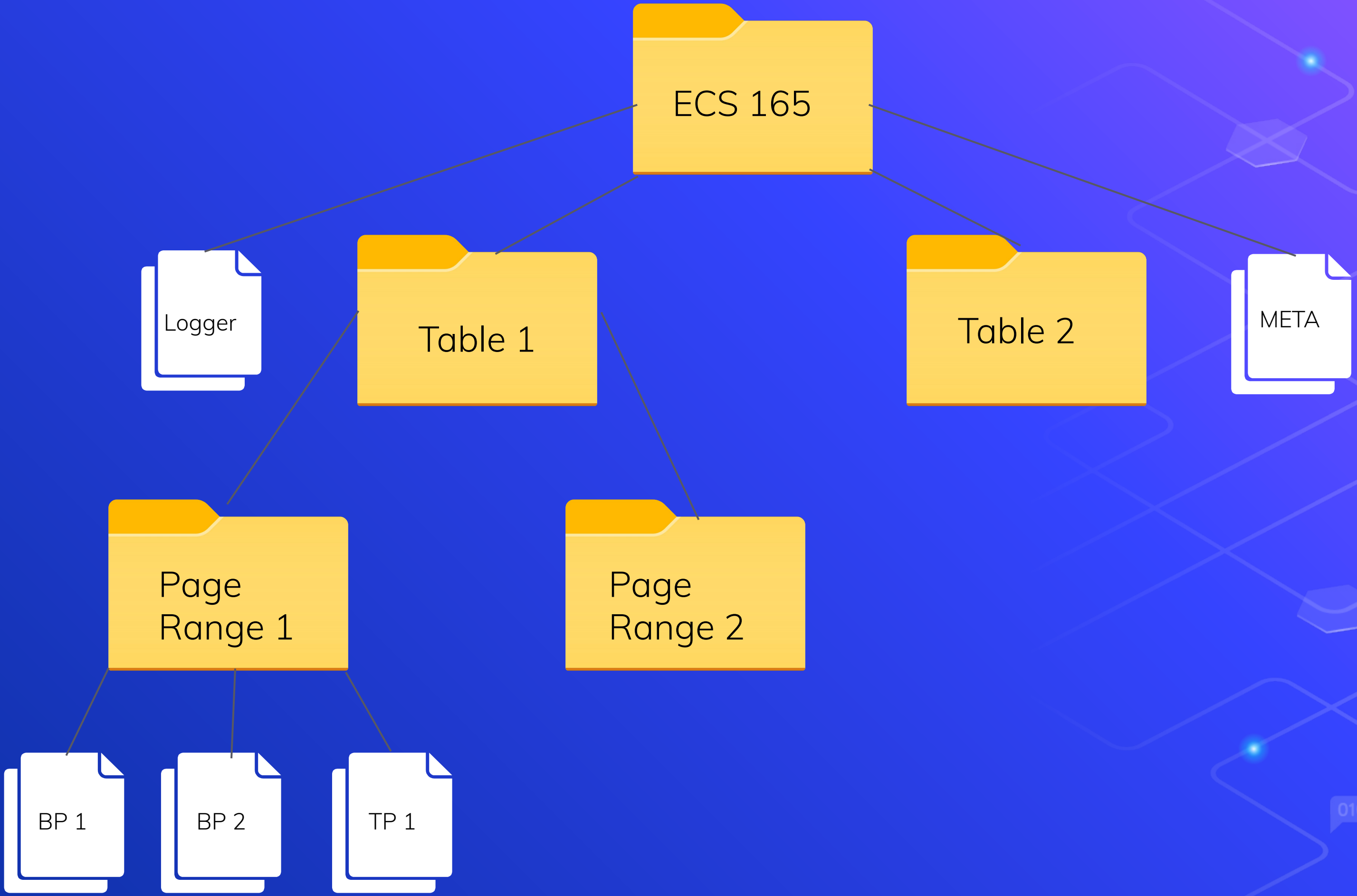
Milestone 3

by The B Team

Architecture

Minor Changes


ECS 165
Logger
Table 1
Table 2
META
Page Range 1
Page Range 2
BP 1
BP 2
TP 1

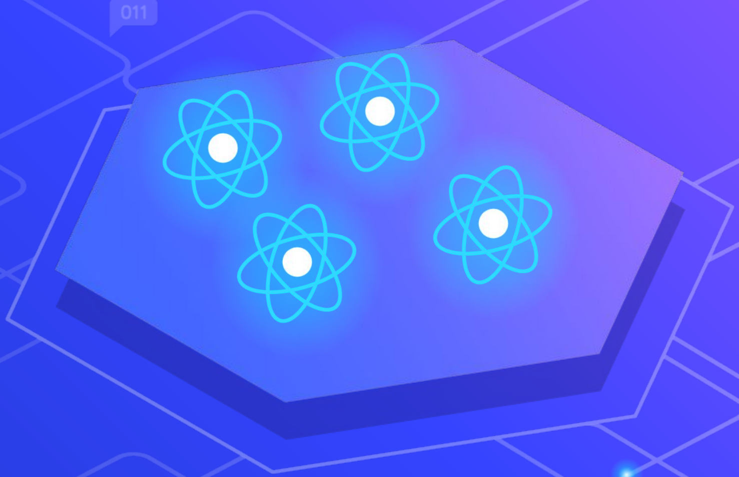
Transactional Semantics

- Atomicity
- Isolation

Goal : Atomicity

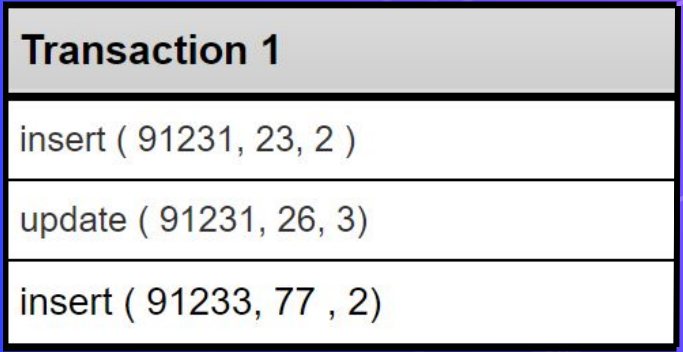
- Single unit transactions
- All or nothing
  - Any fails => abort()
  - No fails => commit()

- Transaction Failure:
  - Query fails



Transaction

- Multiple query calls
- run( ) , abort( ), and commit( )



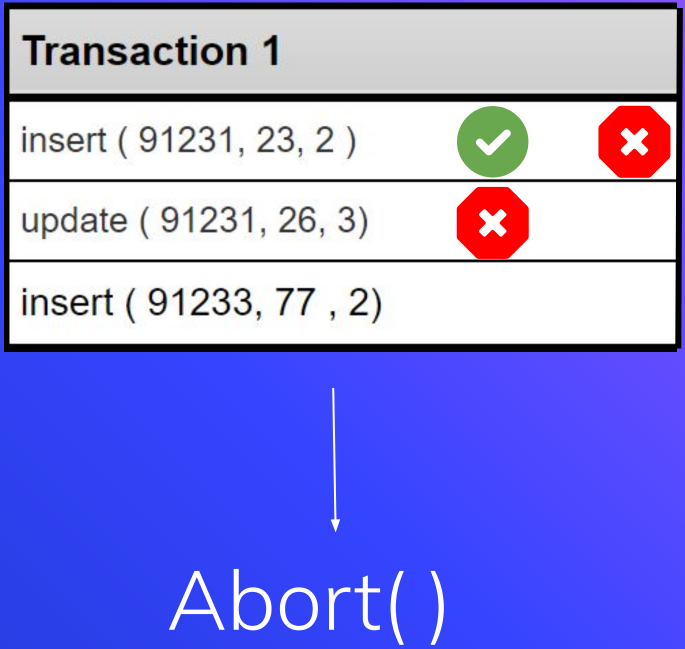
Transaction 1
insert (91231, 23, 2)
update (91231, 26, 3)
insert (91233, 77 , 2)

Transaction Run( )

- Execute in order
- Write to log
- Unsuccessful query -> halt execution
- Finishes without aborting

Transaction Abort( )

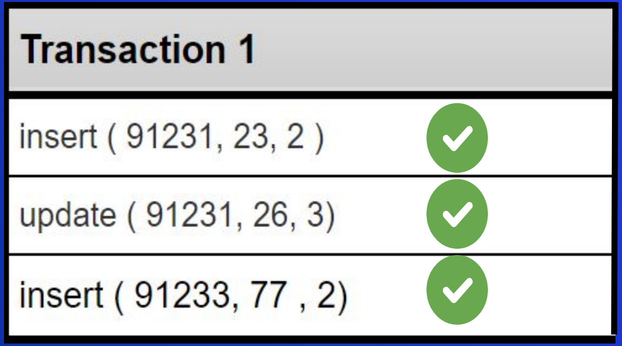
- Reads through the logs
- Abort Inserts deletes that record.
- Abort Updates removes the aborted updates and points to a new tail unaborted values.



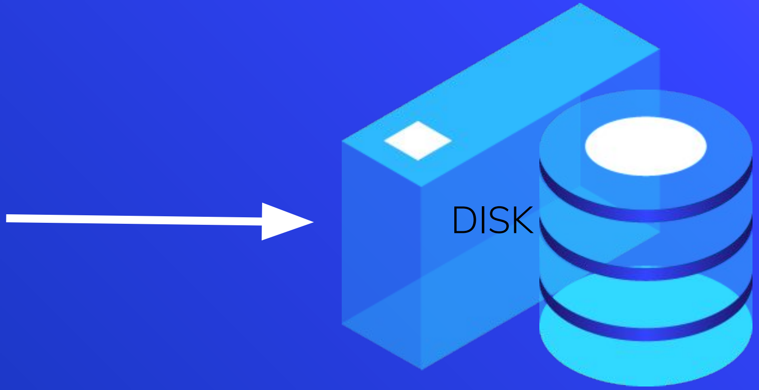
Transaction 1		
insert (91231, 23, 2)	✓	✗
update (91231, 26, 3)	✗	
insert (91233, 77 , 2)		

Transaction Commit( )

- Release all locks for a transaction number
- Write transaction to disk



Transaction 1	
insert (91231, 23, 2)	✓
update (91231, 26, 3)	✓
insert (91233, 77 , 2)	✓



Logger

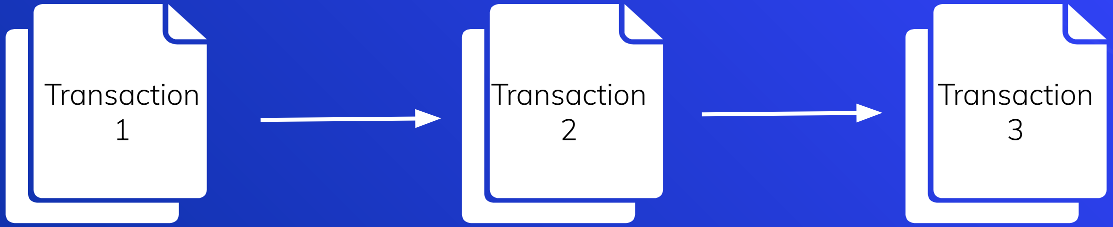
- Log each Query
- Transaction number, query type, key, and columns
- Undo Changes
- DB failure (power failure, etc)

Concurrency Control

- Multithreading

Goal : Isolation

- Concurrent Transactions
- Lock manager
- Treat as sequential

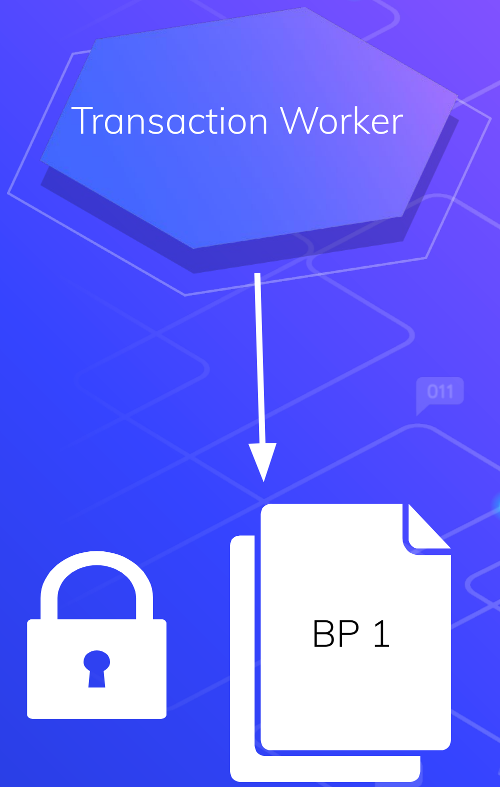


Transaction Worker

- List of transactions
- Run on own thread
- Each thread runs concurrently, wait for all threads to finish before we close

Lock Manager

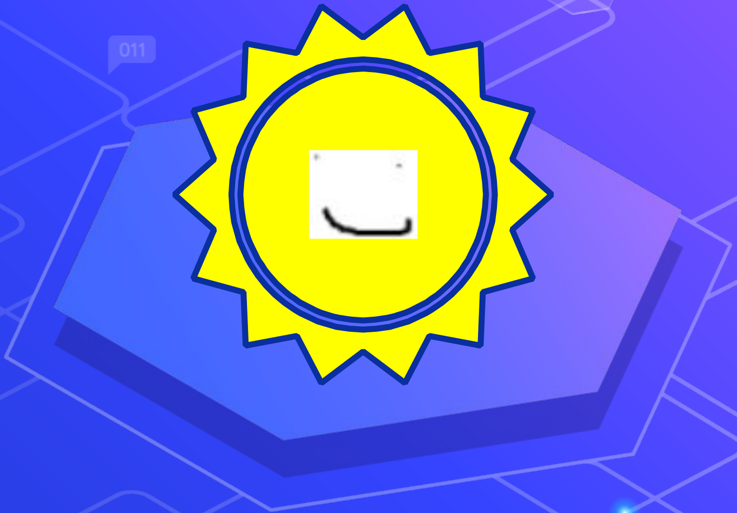
- Read Lock
- Write Lock
- Abort if locked



Latches

- Latch critical points
- Bufferpool locked during read/write operations
- Insert locked

Thank You!



Extra resources

SlidesCarnival icons are editable shapes.

This means that you can:

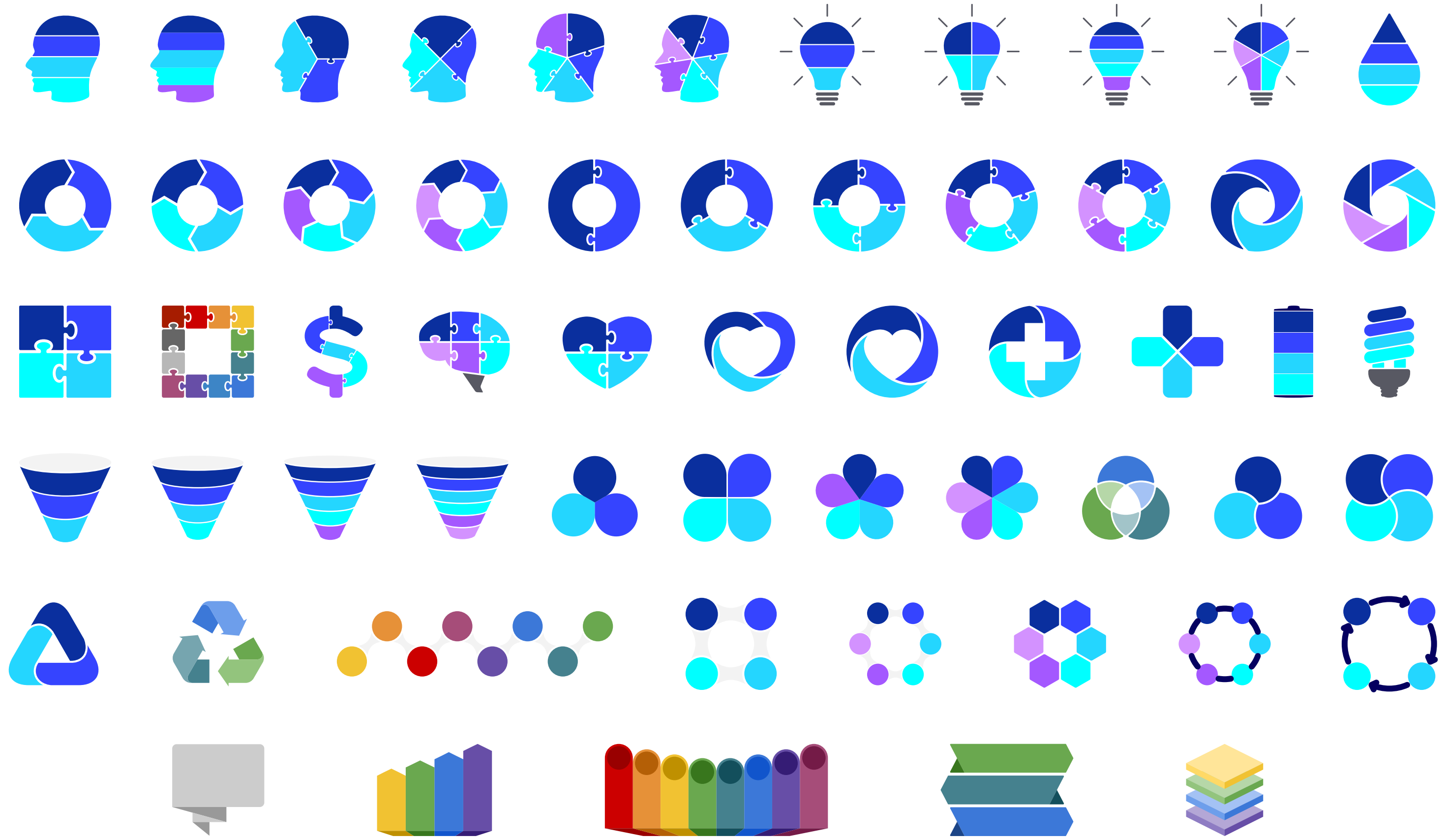
- Resize them without losing quality.
- Change fill color and opacity.
- Change line color, width and style.

Isn't that nice? :)

Examples:

Find more icons at
slidescarnival.com/extra-free-resources-icons-and-maps

Diagrams and infographics



You can also use any emoji as an icon!
And of course it resizes without losing quality.

How? Follow Google instructions
https://twitter.com/googledocs/status/730087240156643328

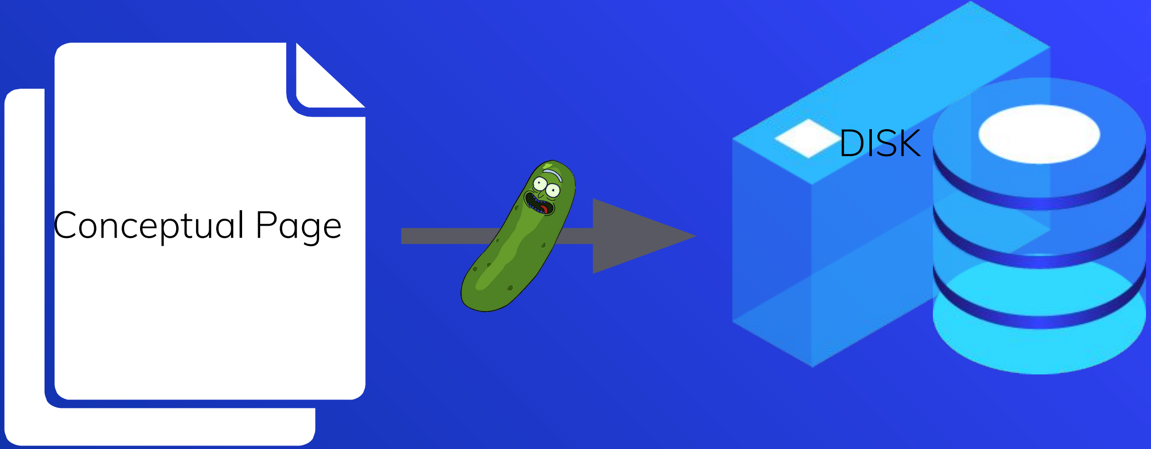
and many more...

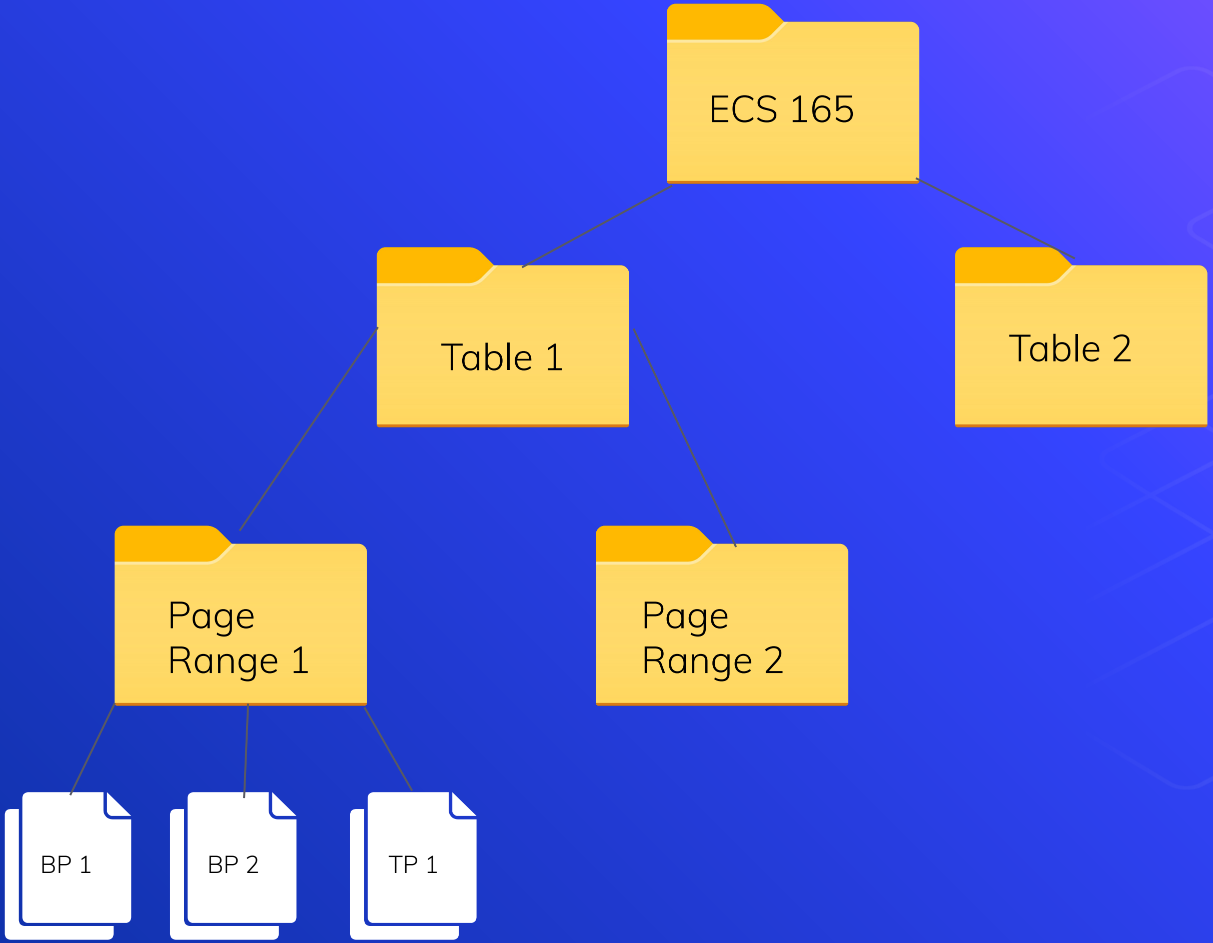
Architectural Changes

Re-done from milestone 1 for efficiency

Writing To Disk

- Write CP object to a file using pickle module




ECS 165
Table 1
Table 2
Page Range 1
Page Range 2
BP 1
BP 2
TP 1

Data Management - Bufferpool

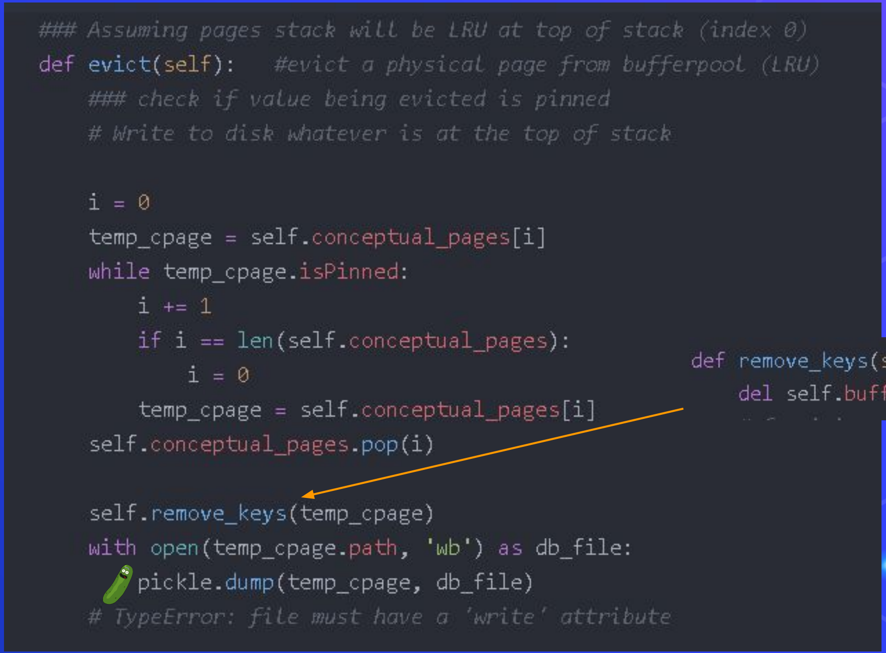
- MetaData
- Conceptual Page granularity
- Manages interactions between Query and Disk

Bufferpool - Metadata

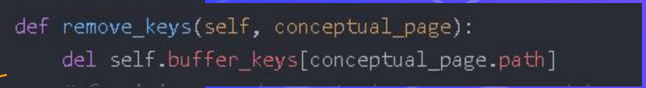
- Maintains key_dict - > Key : Path
- Stores any other dict based on what is indexed
- Keeps track of # PR, BP, RID etc.

Bufferpool- Evict

- LRU Eviction Policy
- Only evicts unpinned pages
- Cleans up by locally stored keys



```
### Assuming pages stack will be LRU at top of stack (index 0)
def evict(self):   #evict a physical page from bufferpool (LRU)
    ### check if value being evicted is pinned
    # Write to disk whatever is at the top of stack

    i = 0
    temp_cpage = self.conceptual_pages[i]
    while temp_cpage.isPinned:
        i += 1
        if i == len(self.conceptual_pages):
            i = 0
        temp_cpage = self.conceptual_pages[i]
    self.conceptual_pages.pop(i)

    self.remove_keys(temp_cpage)
    with open(temp_cpage.path, 'wb') as db_file:
      🥒pickle.dump(temp_cpage, db_file)
    # TypeError: file must have a 'write' attribute
```



```
def remove_keys(self, conceptual_page):
    del self.buffer_keys[conceptual_page.path]
```

Reformatting Query

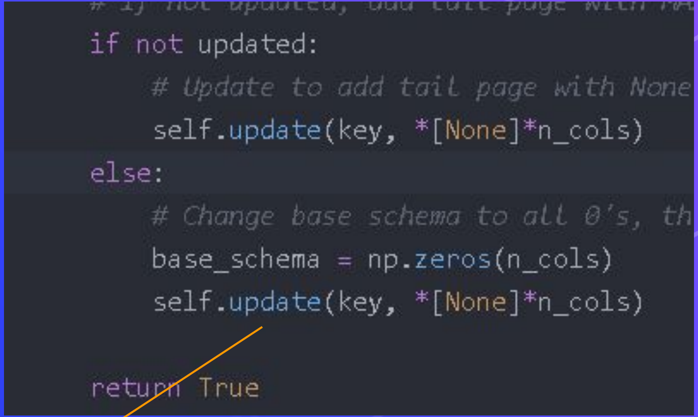
Including bufferpool

New Delete

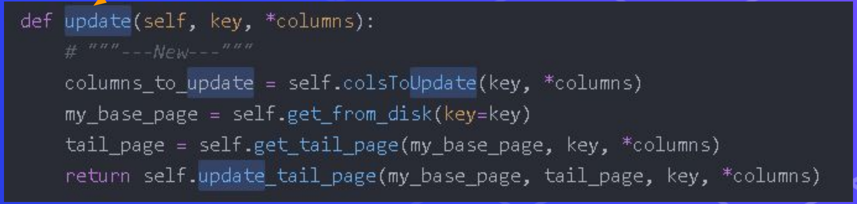
- If not updated before "delete" by adding a tail page with none in columns
- Otherwise: base_schema->-0

  Update page with empty

  Columns
- Update uses bufferpool



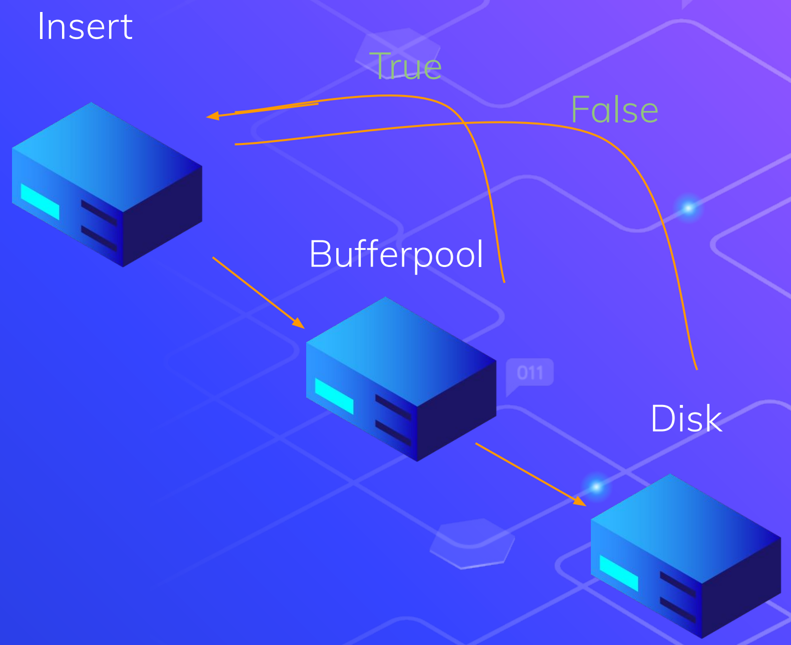
```
    if not updated:
        # Update to add tail page with None
        self.update(key, *[None]*n_cols)
    else:
        # Change base schema to all 0's, th
        base_schema = np.zeros(n_cols)
        self.update(key, *[None]*n_cols)

    return True
```

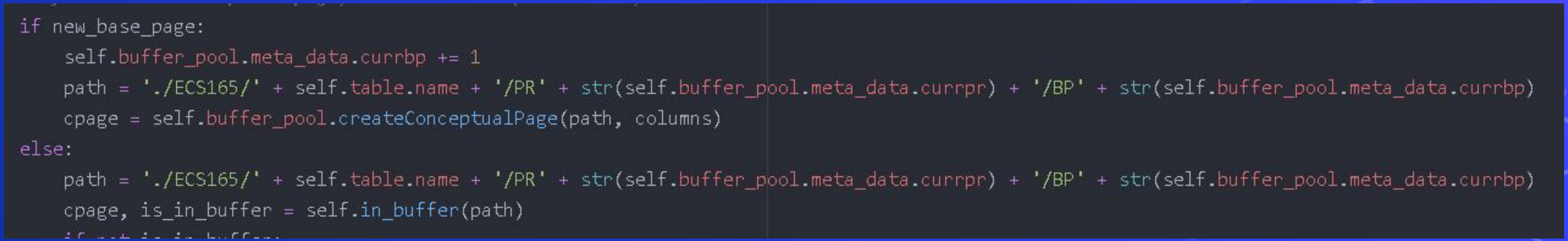


```
def update(self, key, *columns):
    # """---New---"""
    columns_to_update = self.colsToUpdate(key, *columns)
    my_base_page = self.get_from_disk(key=key)
    tail_page = self.get_tail_page(my_base_page, key, *columns)
    return self.update_tail_page(my_base_page, tail_page, key, *columns)
```

New Insert

- Restructured to use bufferpool
- Uses the bufferpool to perform insertions





```
if new_base_page:
    self.buffer_pool.meta_data.currbp += 1
    path = './ECS165/' + self.table.name + '/PR' + str(self.buffer_pool.meta_data.currpr) + '/BP' + str(self.buffer_pool.meta_data.currbp)
    cpage = self.buffer_pool.createConceptualPage(path, columns)
else:
    path = './ECS165/' + self.table.name + '/PR' + str(self.buffer_pool.meta_data.currpr) + '/BP' + str(self.buffer_pool.meta_data.currbp)
    cpage, is_in_buffer = self.in_buffer(path)
```

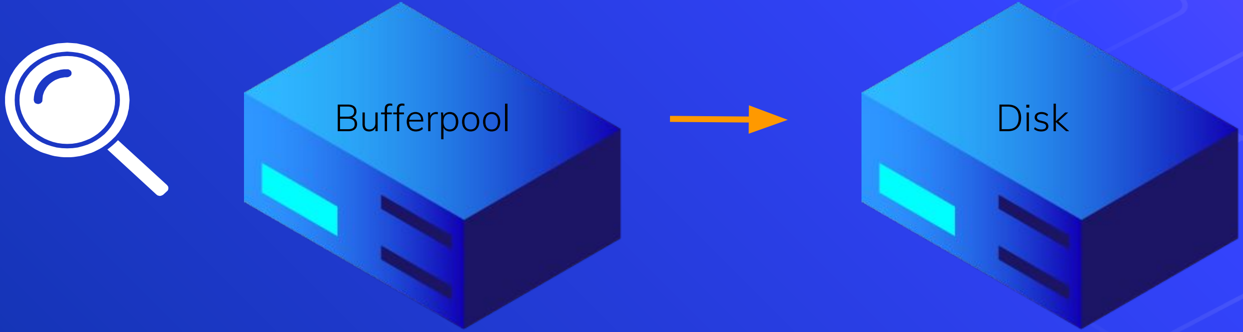
New Select

- If we are selecting based on keys simply use key_dict and same logic as before
- Otherwise we have to utilize secondary indexes if available

Bufferpool
key_dict

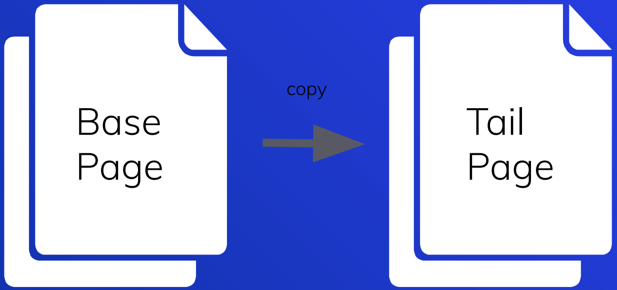
New Update

- Fetch the BP from key
- Find or create the TP associated with given key
- Then update like usual



New Update - Snapshot

- Snapshot upon first update to a specific column in BP
- Copy all query columns from BP into TP

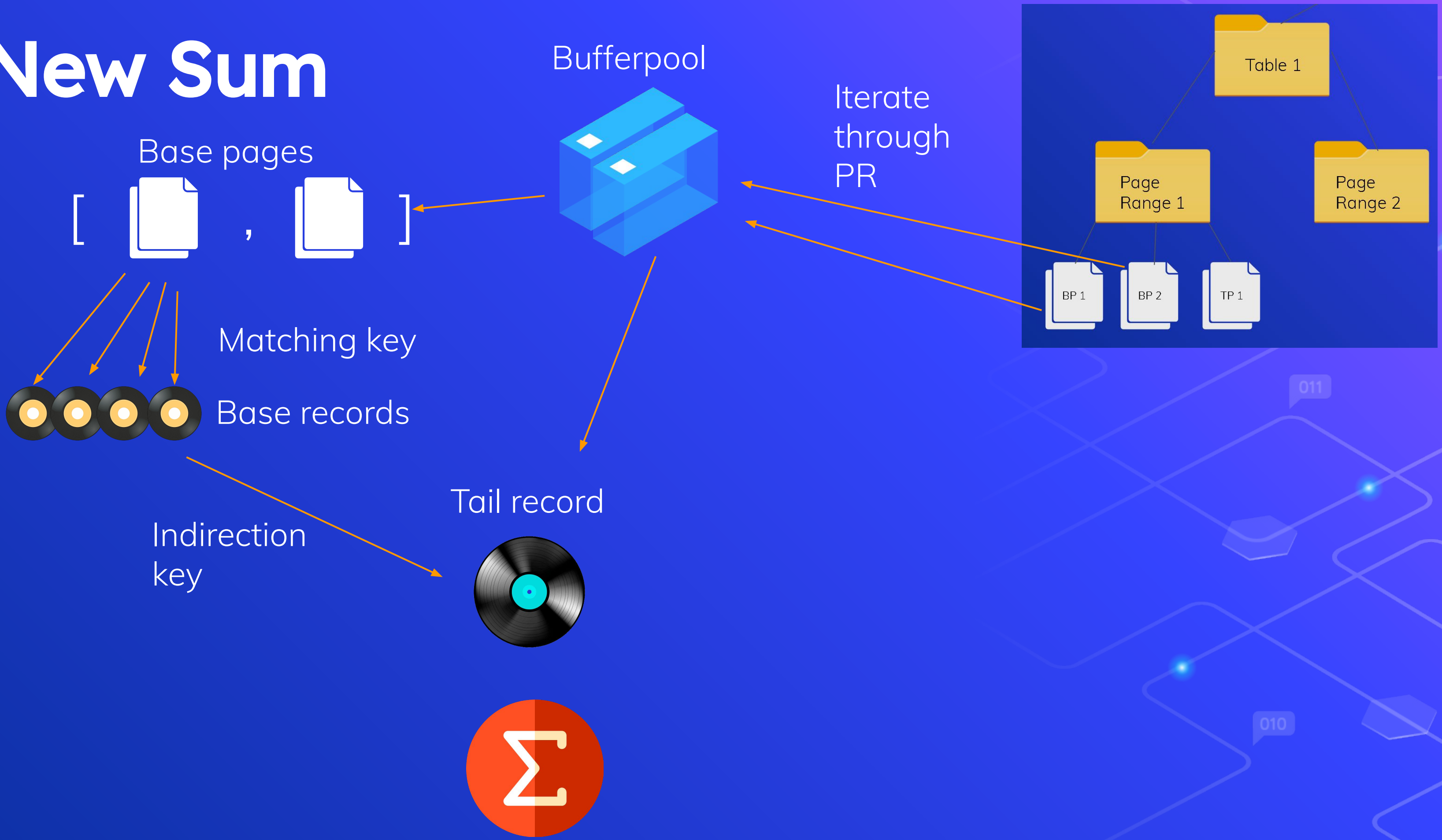


Base Page : [45, 21, 32 , 1, 9]

Update: [17, None, None, 2, None]

Snapshot : [45, max_int, max_int, 1, max_int]

New Sum

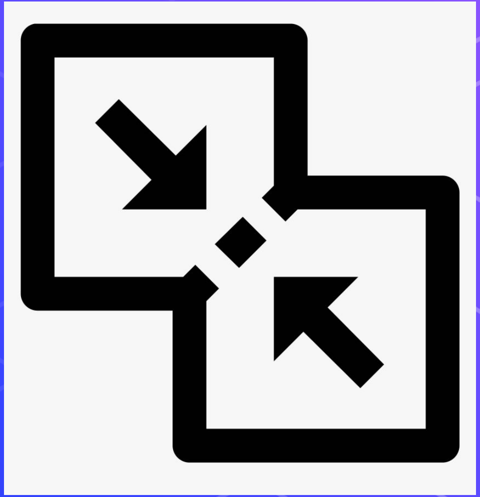

Bufferpool
Base pages
[,]
Iterate through PR
Table 1
Page Range 1
Page Range 2
BP 1
BP 2
TP 1
Matching key
Base records
Tail record
Indirection key
Σ

Data Reorganization

Lazy, Contention Free Merge

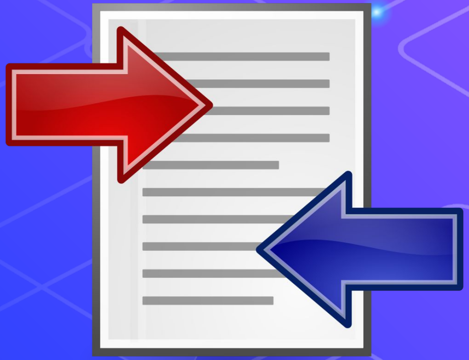
Merge: Criteria

- BP must be read-only 📖
  - Keep track of qualified BPs in bufferpool
  - Qualified BP is if BP is full
- Merge after every 500 updates, check qualified BP array



Merge: Process

- Open new thread using Python’s Threading library and run in background
- Pull base page and associated tail pages into memory
  - Create a copy of the base pages
- Consolidate updates onto copied base pages
- After merge we change the key_dict to point to the new consolidated base pages



Indexing

Create Index & Drop Index

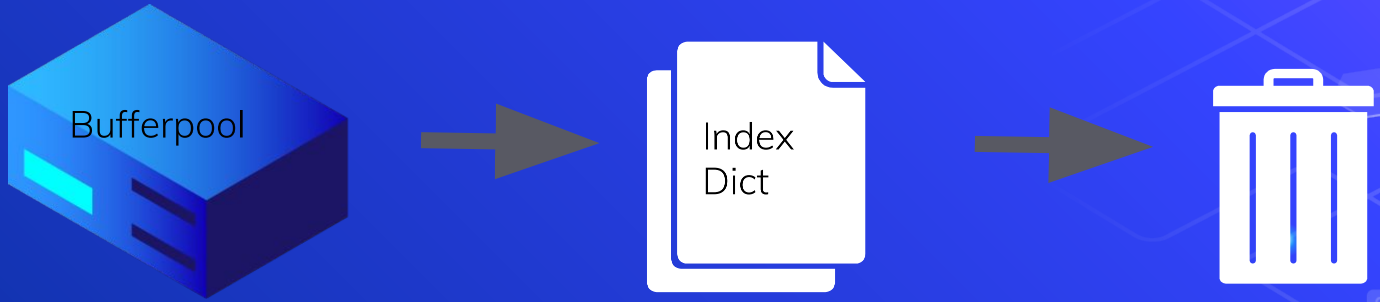
create_index()

- Iterate through base pages in disk
- Iterate through records in base page
    - Value: (base_path, record num)

- Add that value to a index dictionary:
    - (e.g value:[(path_base_i, record_num_j)]

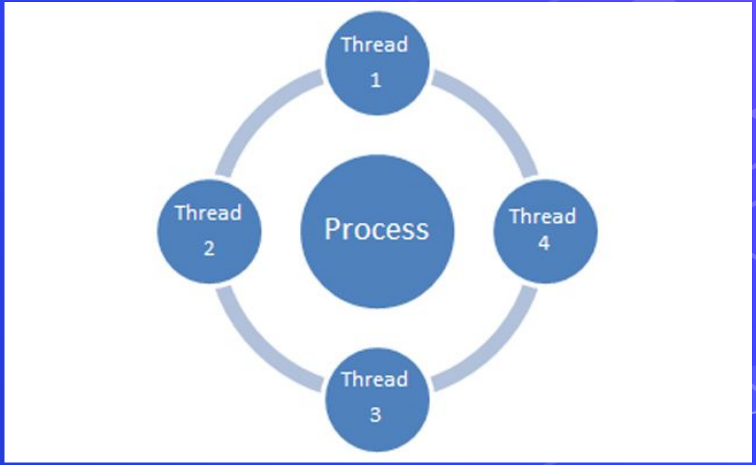
drop_index()

- Deletes the dictionary for a given index



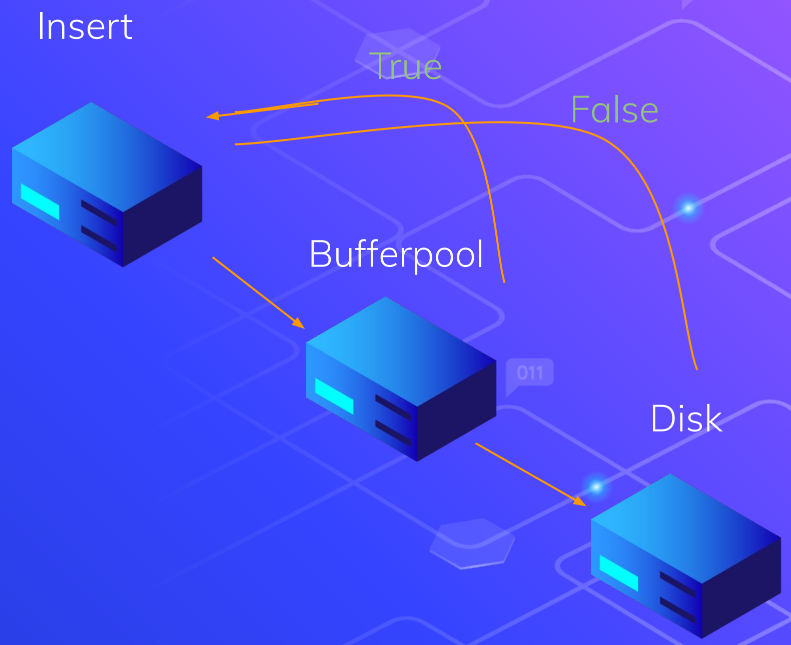
Future Goals & Plans

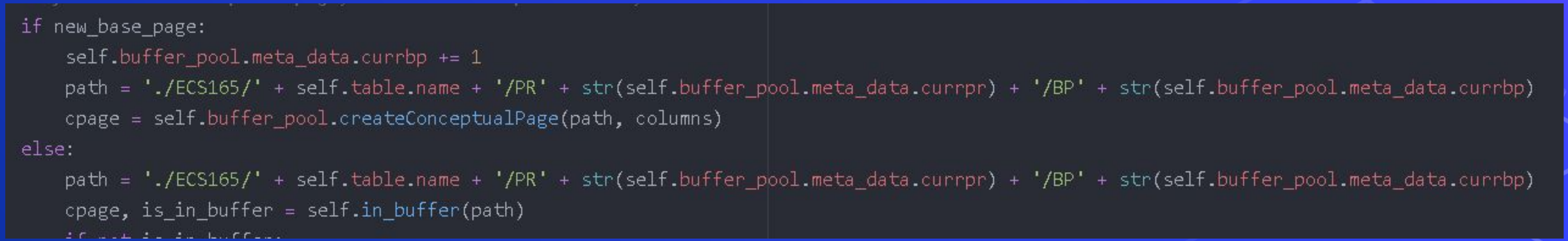
- Implement Multiple Threads for concurrent use of the L-Store



New Insert

- Restructured to use bufferpool
- Uses the bufferpool to perform insertions





```
if new_base_page:
    self.buffer_pool.meta_data.currbp += 1
    path = './ECS165/' + self.table.name + '/PR' + str(self.buffer_pool.meta_data.currpr) + '/BP' + str(self.buffer_pool.meta_data.currbp)
    cpage = self.buffer_pool.createConceptualPage(path, columns)
else:
    path = './ECS165/' + self.table.name + '/PR' + str(self.buffer_pool.meta_data.currpr) + '/BP' + str(self.buffer_pool.meta_data.currbp)
    cpage, is_in_buffer = self.in_buffer(path)
```

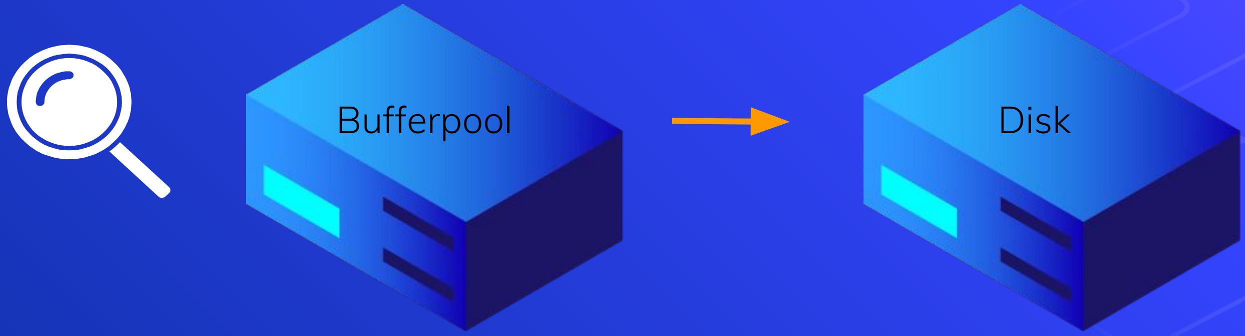
New Select

- If we are selecting based on keys simply use key_dict and same logic as before
- Otherwise we have to utilize secondary indexes if available

Bufferpool
key_dict

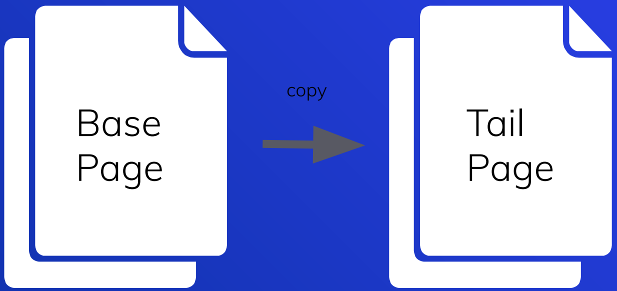
New Update

- Fetch the BP from key
- Find or create the TP associated with given key
- Then update like usual



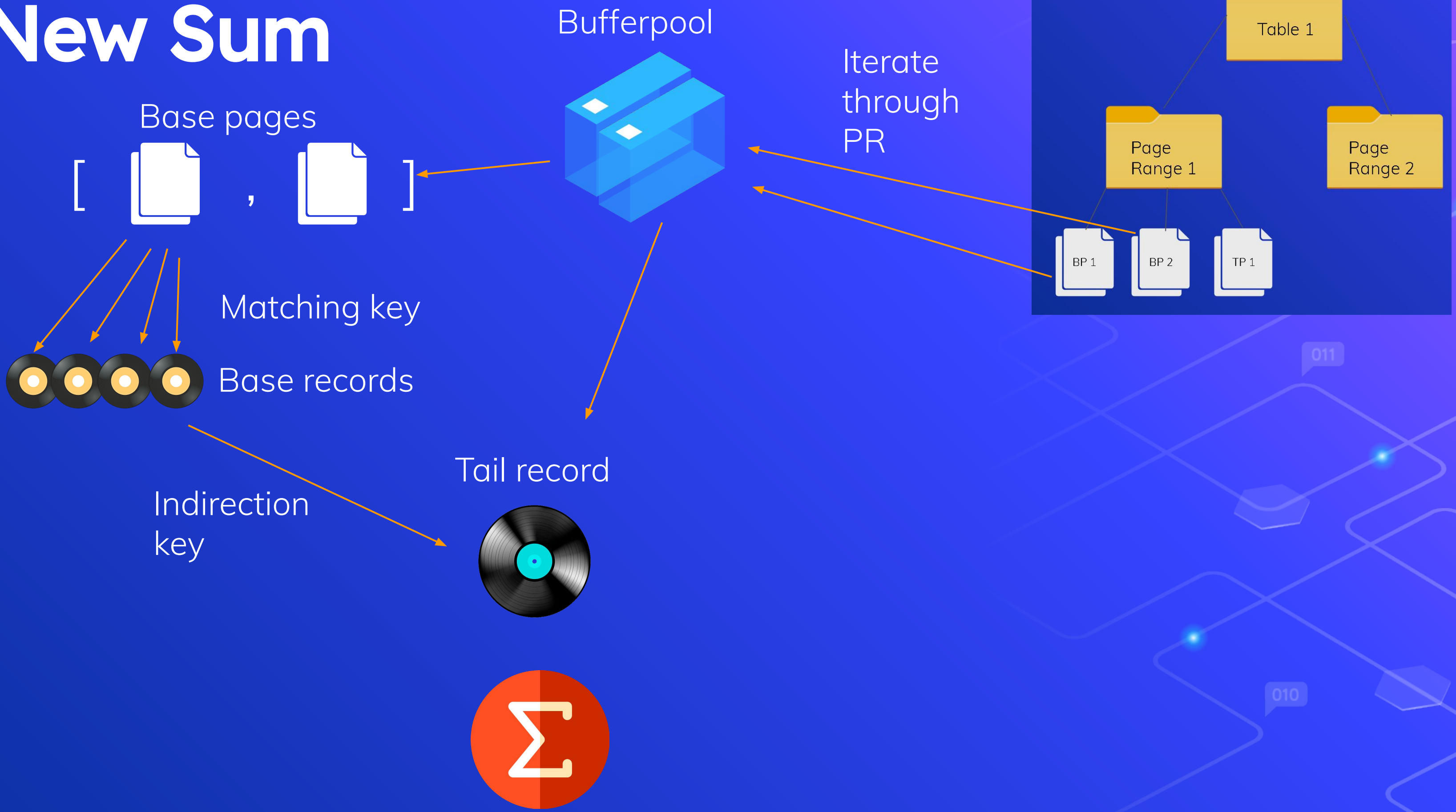
New Update - Snapshot

- Snapshot upon first update to a specific column in BP
- Copy all query columns from BP into TP



Base Page :	[45, 21, 32 , 1, 9]
Update:	[17, None, None, 2, None]
Snapshot :	[45, max_int, max_int, 1, max_int]

New Sum


Bufferpool
Base pages
[,]
Iterate through PR
Table 1
Page Range 1
Page Range 2
BP 1
BP 2
TP 1
Matching key
Base records
Tail record
Indirection key

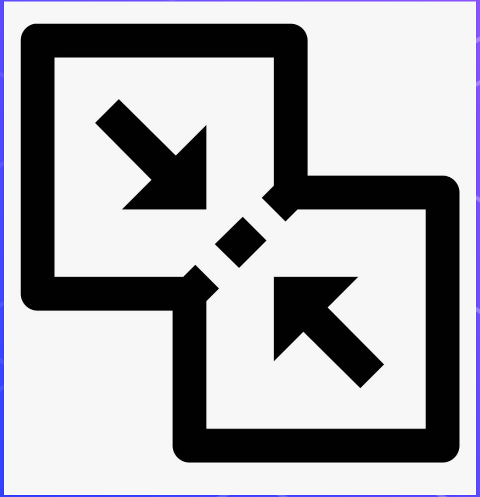
Data Reorganization

Lazy, Contention Free Merge

Merge: Criteria

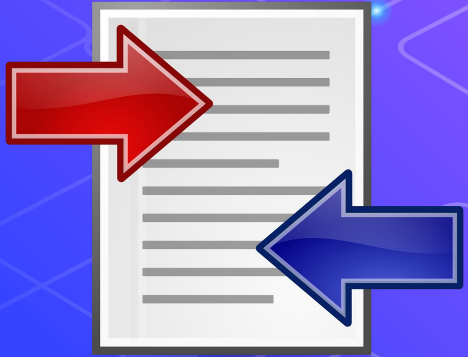
- BP must be read-only 📖
  - Keep track of qualified BPs in bufferpool
  - Qualified BP is if BP is full

- Merge after every 500 updates, check qualified BP array



Merge: Process

- Open new thread using Python's Threading library and run in background
- Pull base page and associated tail pages into memory
  - Create a copy of the base pages
- Consolidate updates onto copied base pages
- After merge we change the key_dict to point to the new consolidated base pages



Indexing

Create Index & Drop Index

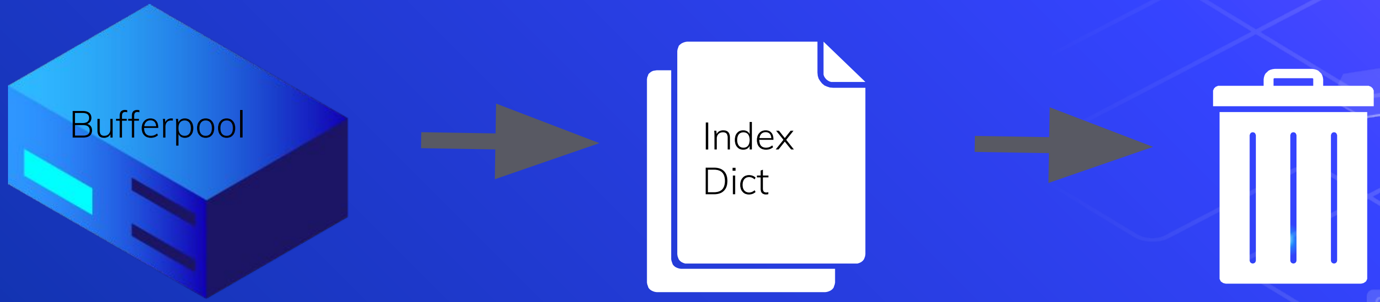
create_index()

- Iterate through base pages in disk
- Iterate through records in base page
    - Value: (base_path, record num)

- Add that value to a index dictionary:
    - (e.g value:[(path_base_i, record_num_j)]

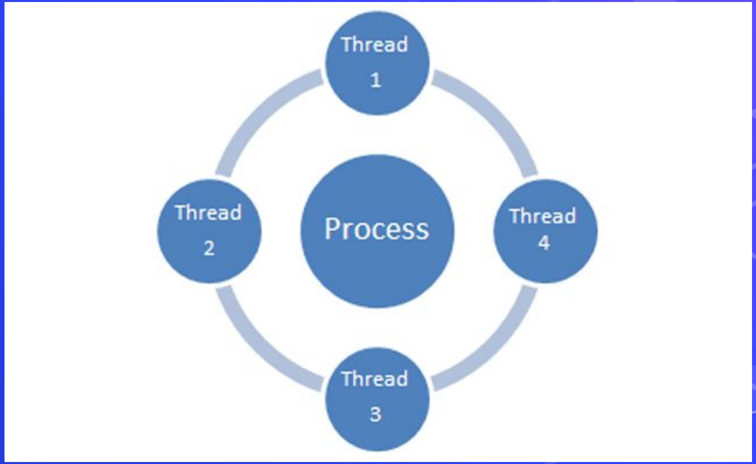
drop_index()

- Deletes the dictionary for a given index



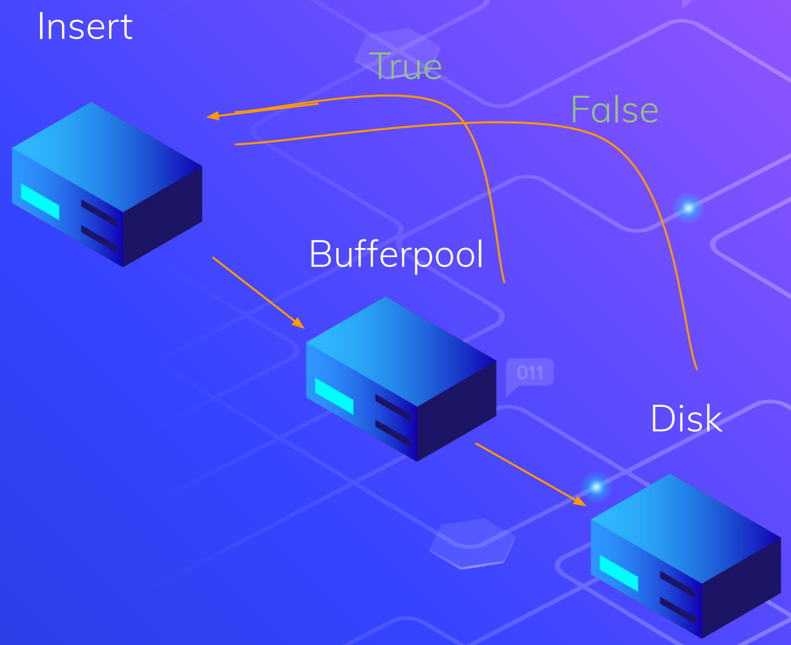
Future Goals & Plans

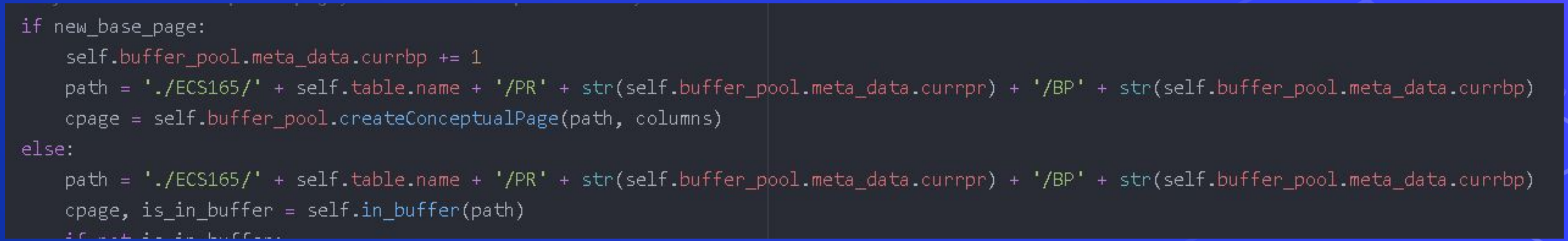
- Implement Multiple Threads for concurrent use of the L-Store



New Insert

- Restructured to use bufferpool
- Uses the bufferpool to perform insertions





```
if new_base_page:
    self.buffer_pool.meta_data.currbp += 1
    path = './ECS165/' + self.table.name + '/PR' + str(self.buffer_pool.meta_data.currpr) + '/BP' + str(self.buffer_pool.meta_data.currbp)
    cpage = self.buffer_pool.createConceptualPage(path, columns)
else:
    path = './ECS165/' + self.table.name + '/PR' + str(self.buffer_pool.meta_data.currpr) + '/BP' + str(self.buffer_pool.meta_data.currbp)
    cpage, is_in_buffer = self.in_buffer(path)
```

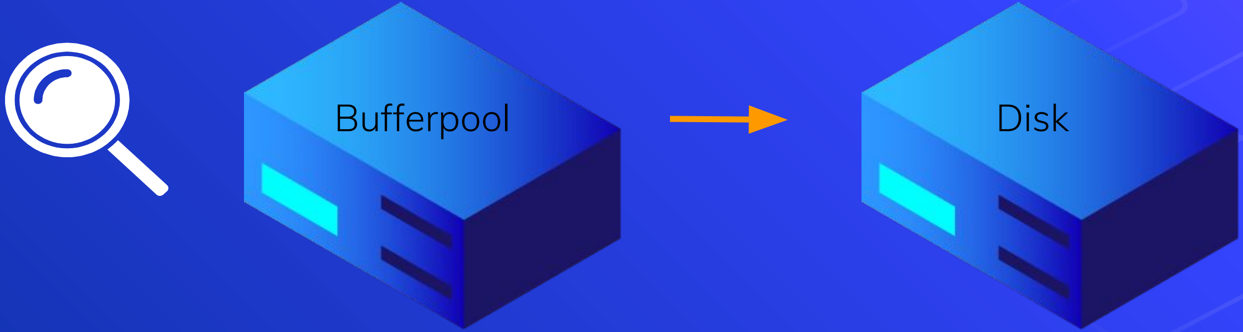
New Select

- If we are selecting based on keys simply use key_dict and same logic as before
- Otherwise we have to utilize secondary indexes if available

Bufferpool
key_dict

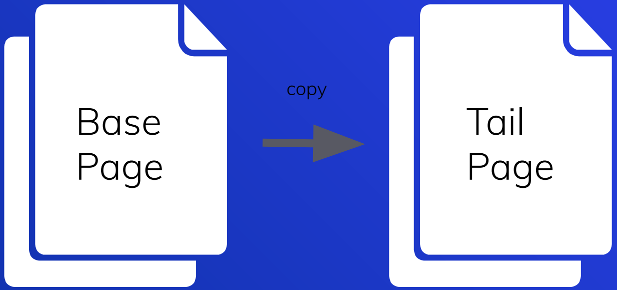
New Update

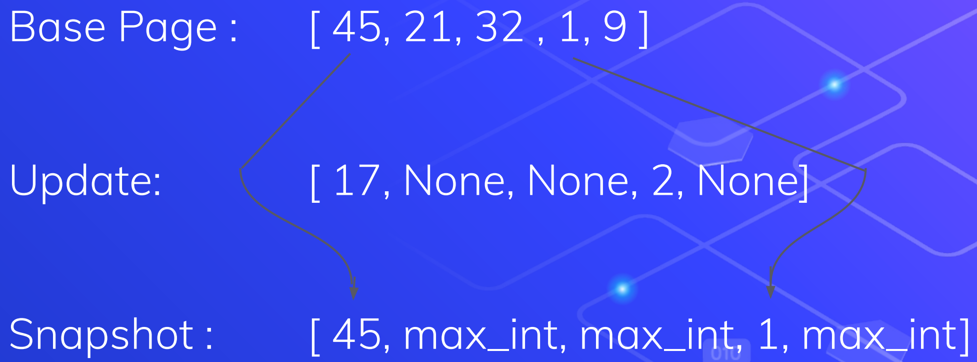
- Fetch the BP from key
- Find or create the TP associated with given key
- Then update like usual



New Update - Snapshot

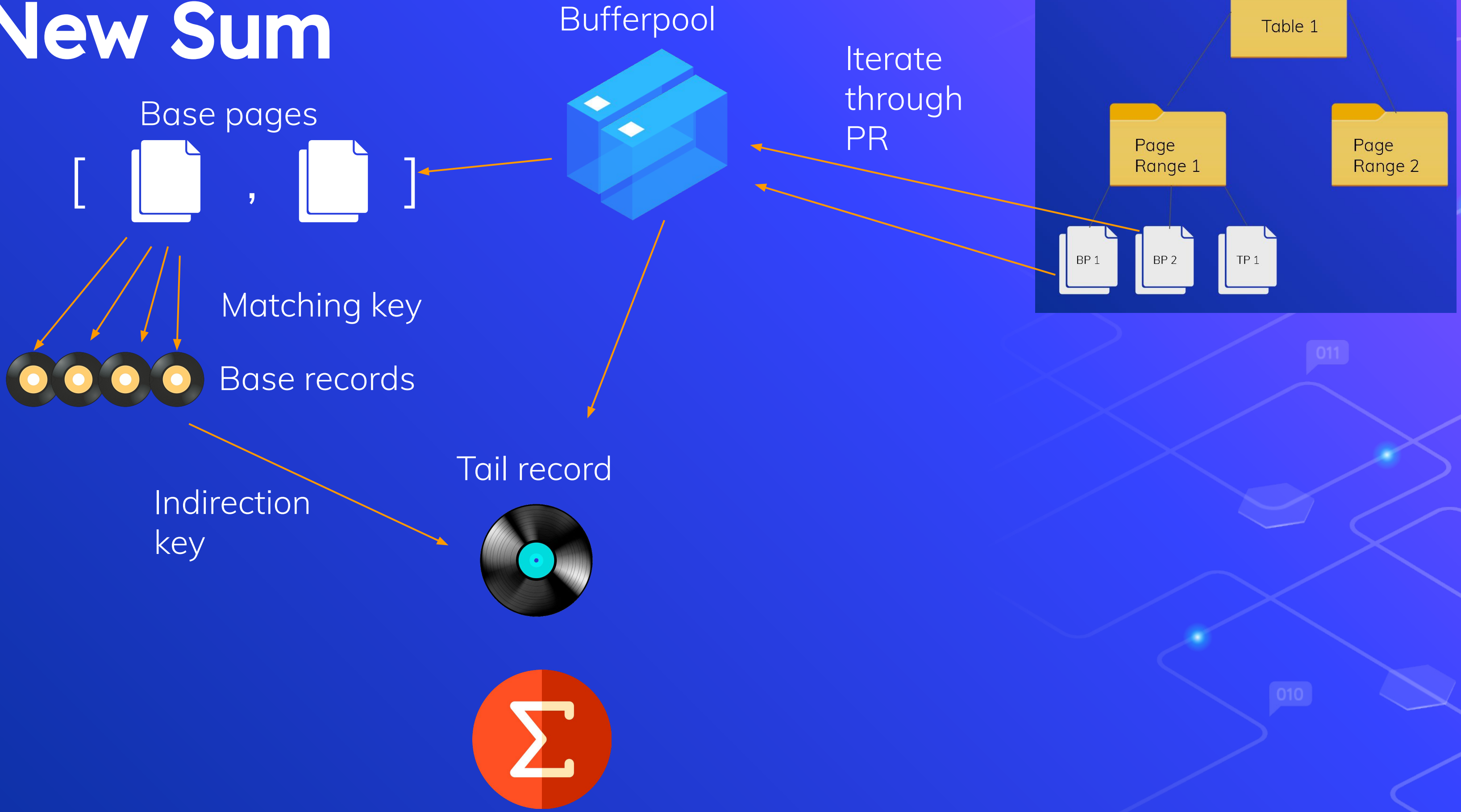
- Snapshot upon first update to a specific column in BP
- Copy all query columns from BP into TP





Base Page :	[45, 21, 32 , 1, 9]
Update:	[17, None, None, 2, None]
Snapshot :	[45, max_int, max_int, 1, max_int]

New Sum


Bufferpool
Base pages
[,]
Iterate through PR
Table 1
Page Range 1
Page Range 2
BP 1
BP 2
TP 1
Matching key
Base records
Tail record
Indirection key

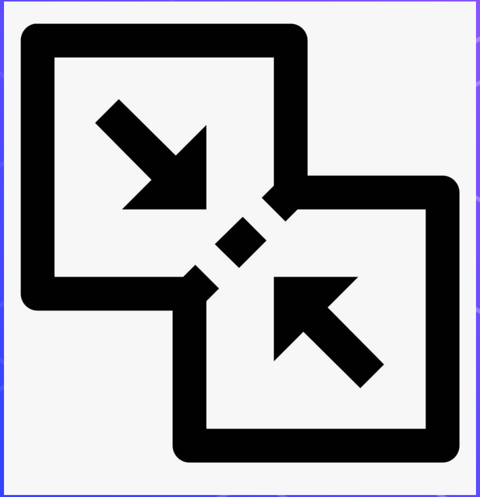
Data Reorganization

Lazy, Contention Free Merge

Merge: Criteria

- BP must be read-only 📖
  - Keep track of qualified BPs in bufferpool
  - Qualified BP is if BP is full

- Merge after every 500 updates, check qualified BP array



Merge: Process

- Open new thread using Python's Threading library and run in background
- Pull base page and associated tail pages into memory
  - Create a copy of the base pages
- Consolidate updates onto copied base pages
- After merge we change the key_dict to point to the new consolidated base pages

Indexing

Create Index & Drop Index

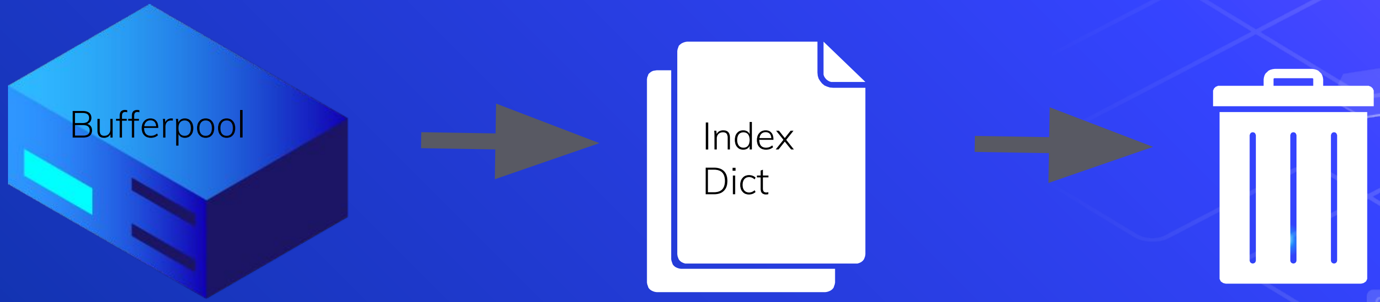
create_index()

- Iterate through base pages in disk
- Iterate through records in base page
    - Value: (base_path, record num)

- Add that value to a index dictionary:
    - (e.g value:[(path_base_i, record_num_j)]

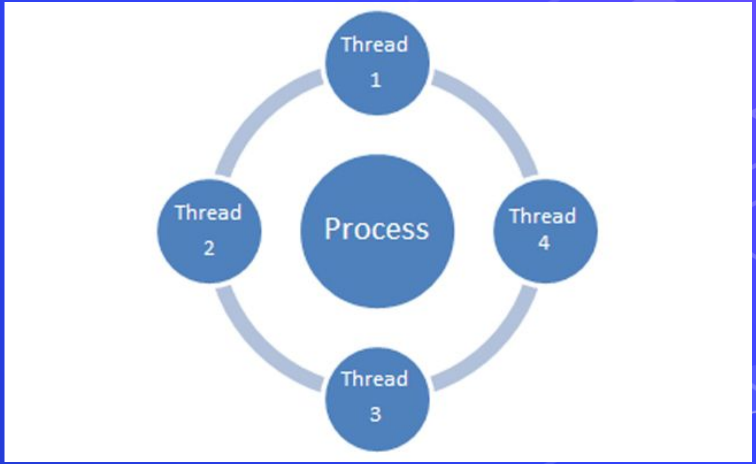
drop_index()

- Deletes the dictionary for a given index



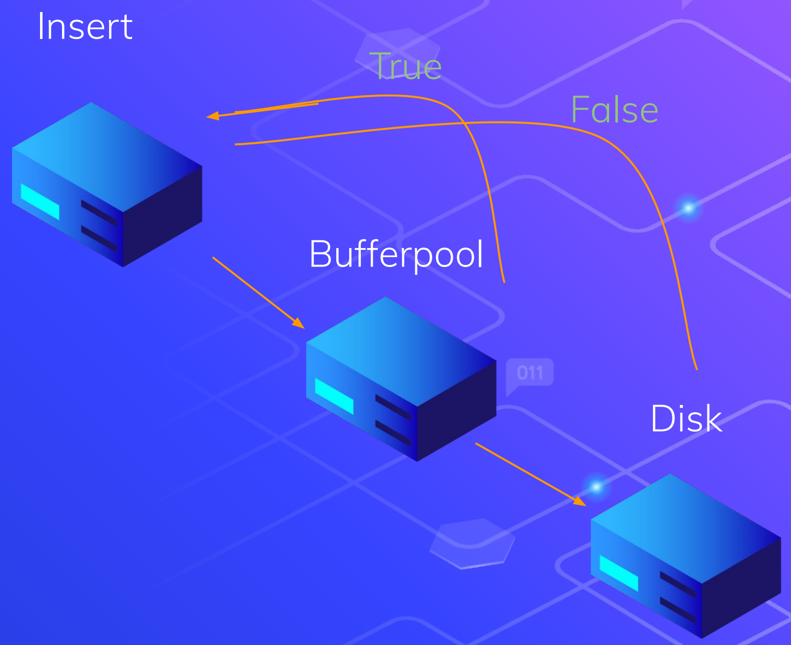
Future Goals & Plans

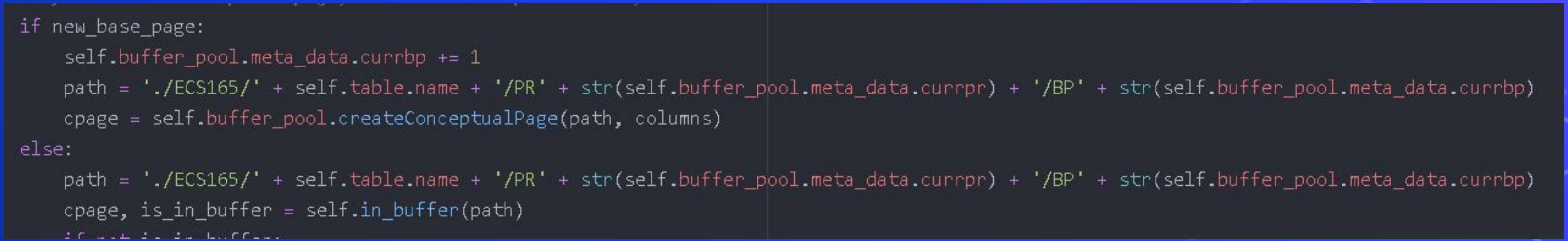
- Implement Multiple Threads for concurrent use of the L-Store



New Insert

- Restructured to use bufferpool
- Uses the bufferpool to perform insertions





```
if new_base_page:
    self.buffer_pool.meta_data.currbp += 1
    path = './ECS165/' + self.table.name + '/PR' + str(self.buffer_pool.meta_data.currpr) + '/BP' + str(self.buffer_pool.meta_data.currbp)
    cpage = self.buffer_pool.createConceptualPage(path, columns)
else:
    path = './ECS165/' + self.table.name + '/PR' + str(self.buffer_pool.meta_data.currpr) + '/BP' + str(self.buffer_pool.meta_data.currbp)
    cpage, is_in_buffer = self.in_buffer(path)
    if not is_in_buffer:
```

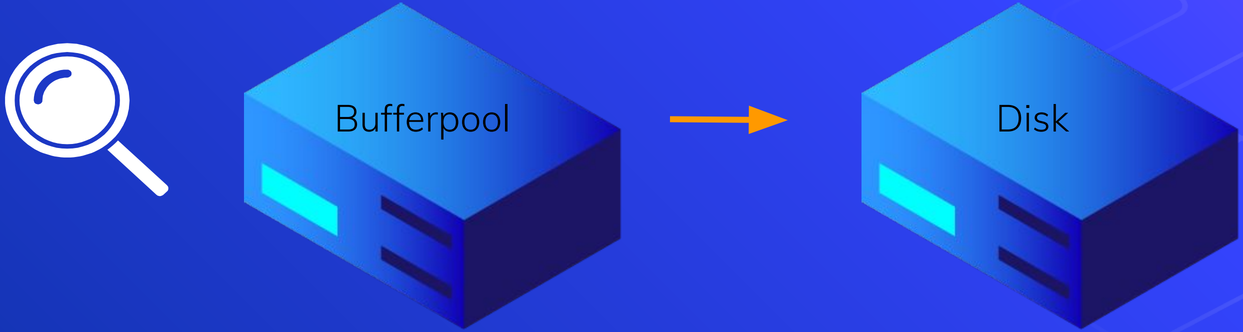
New Select

- If we are selecting based on keys simply use key_dict and same logic as before
- Otherwise we have to utilize secondary indexes if available

Bufferpool
key_dict

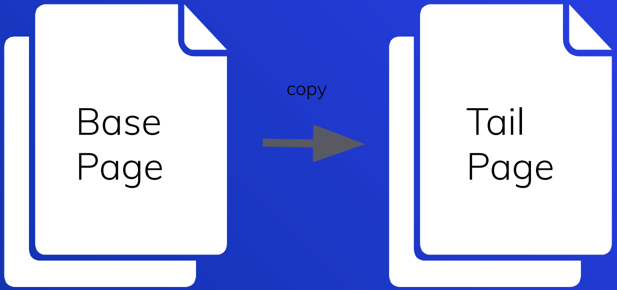
New Update

- Fetch the BP from key
- Find or create the TP associated with given key
- Then update like usual



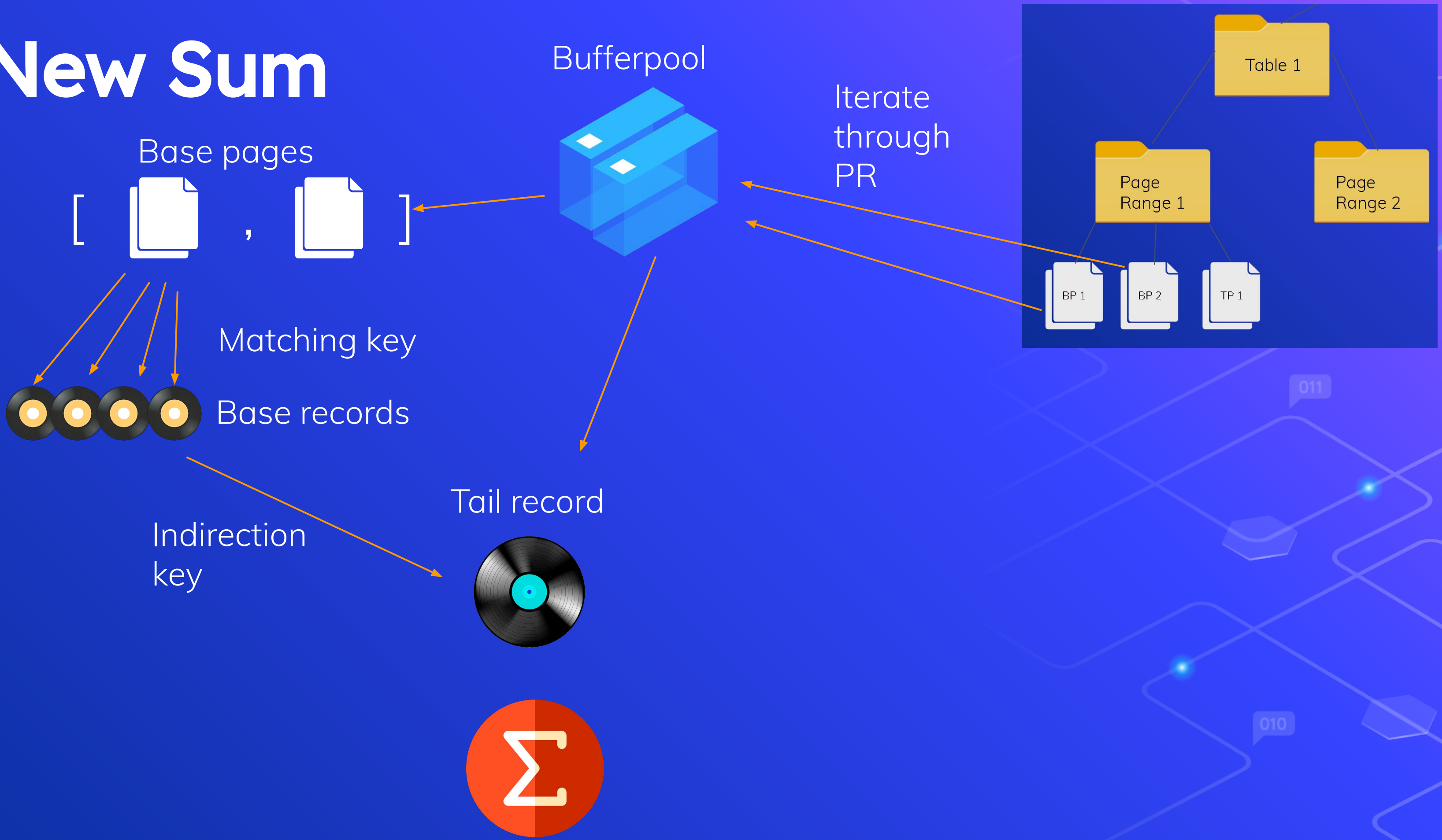
New Update - Snapshot

- Snapshot upon first update to a specific column in BP
- Copy all query columns from BP into TP



Base Page :	[45, 21, 32 , 1, 9]
Update:	[17, None, None, 2, None]
Snapshot :	[45, max_int, max_int, 1, max_int]

New Sum


Bufferpool
Base pages
[,]
Matching key
Base records
Indirection key
Tail record
Iterate through PR
Table 1
Page Range 1
Page Range 2
BP 1
BP 2
TP 1

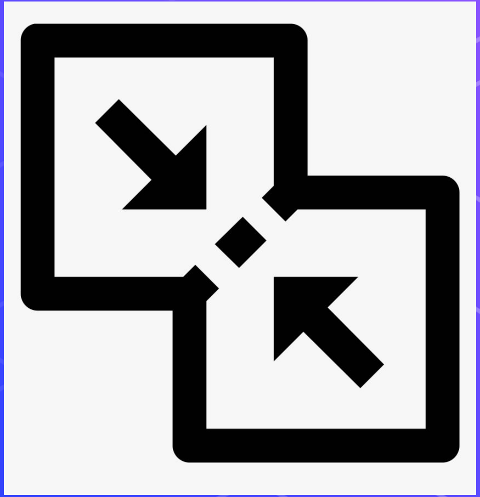
Data Reorganization

Lazy, Contention Free Merge

Merge: Criteria

- BP must be read-only 📖
  - Keep track of qualified BPs in bufferpool
  - Qualified BP is if BP is full

- Merge after every 500 updates, check qualified BP array

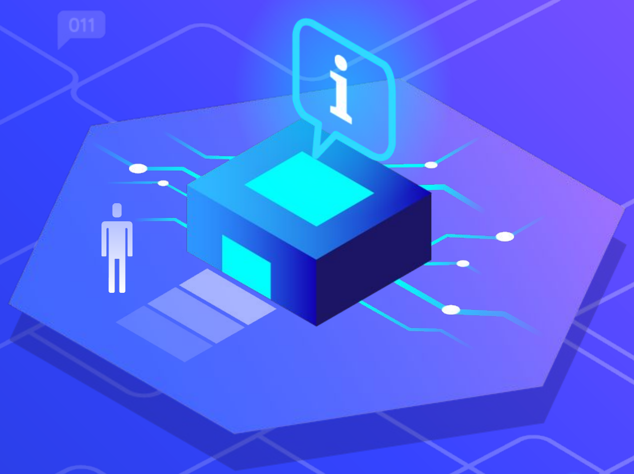


Merge: Process

- Open new thread using Python's Threading library and run in background
- Pull base page and associated tail pages into memory
    - Create a copy of the base pages
- Consolidate updates onto copied base pages
- After merge we change the key_dict to point to the new consolidated base pages

Indexing

Create Index & Drop Index



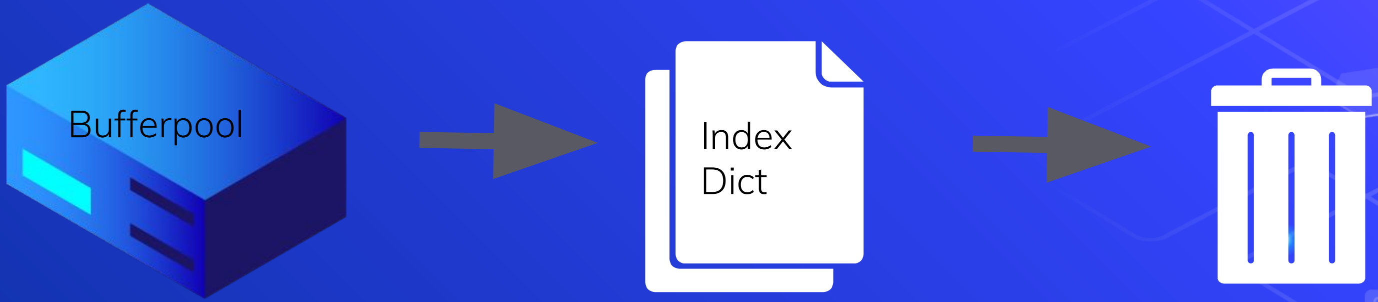
create_index()

- Iterate through base pages in disk
- Iterate through records in base page
    - Value: (base_path, record num)

- Add that value to a index dictionary:
    - (e.g value:[(path_base_i, record_num_j)]

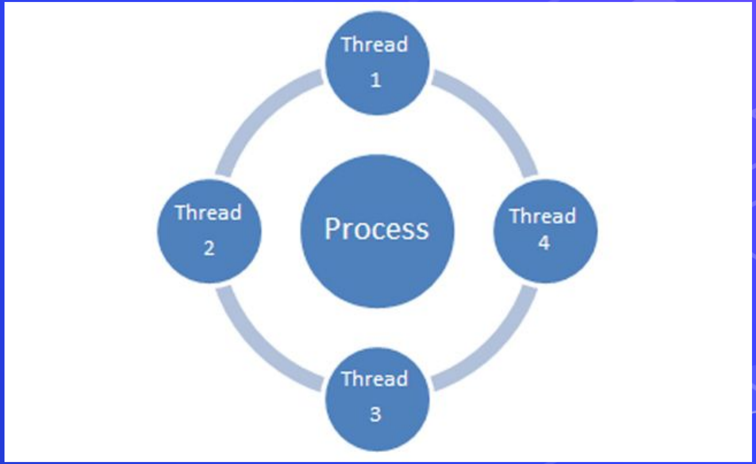
drop_index()

- Deletes the dictionary for a given index



Future Goals & Plans

- Implement Multiple Threads for concurrent use of the L-Store


Thread 1
Thread 2
Process
Thread 4
Thread 3