

VanillaCore Walkthrough

Part 4

Introduction to Databases

DataLab

CS, NTHU

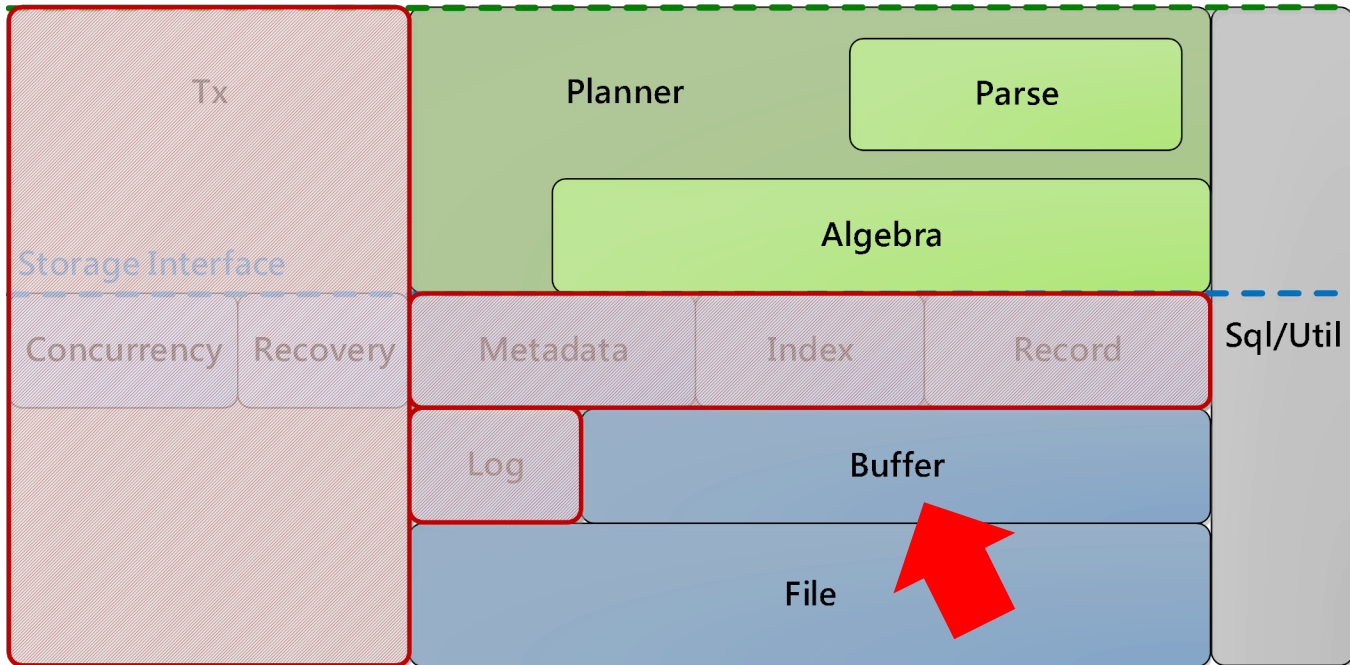
Today's Focus

VanillaDB

JDBC/SP Interface (at Client Side)



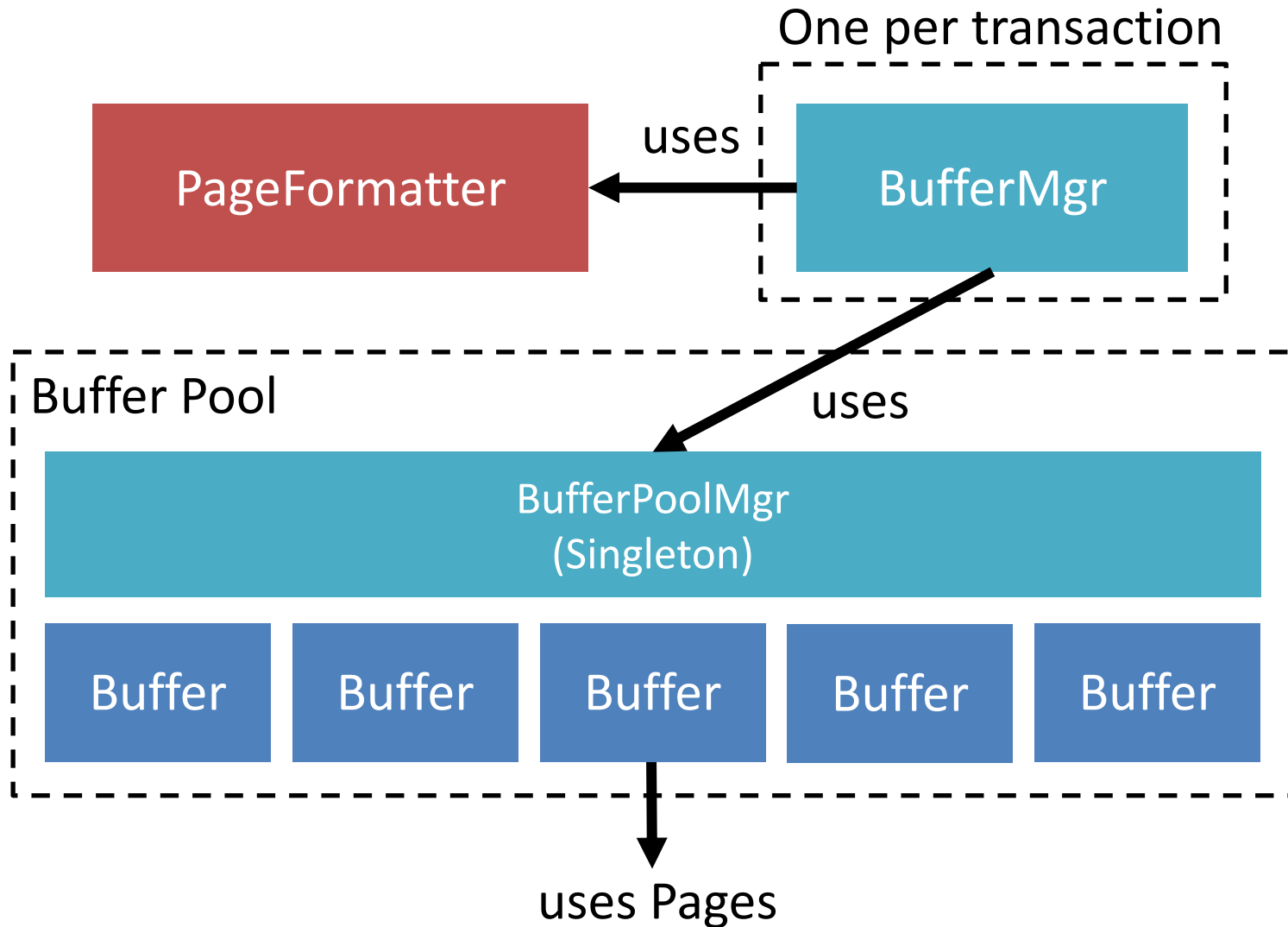
Query Interface



The Mission

- The `buffer` package caches blocks of files in order to reduce I/O.

buffer Package



Functionality

- Main Components
 - `Buffer`: a memory space for caching a block
 - `BufferPoolMgr`: manages the buffer pool
 - `BufferMgr`: provides the access to buffers and manages the pinned buffers for each transaction
 - A transaction waits here if it could not pin any buffer.
 - `PageFormatter`: formats a fresh block

Functionality

- Main Components
 - Buffer: a memory space for caching a block
 - BufferPoolMgr: manages the buffer pool
 - BufferMgr: provides the access to buffers and manages the pinned buffers for each transaction
 - A transaction waits here if it could not pin any buffer.
 - PageFormatter: formats a fresh block

Buffer

- Wraps a page and stores
 - ID of the holding block
 - Pin count
 - Modified information
 - Log information
- ***Supports WAL***
 - `setVal()` requires an LSN
 - Must be preceded by `LogMgr.append()`
 - `flush()` calls `LogMgr.flush(maxLsn)`

Buffer
<pre>~ Buffer() <<synchronized>> + getVal(offset : int, type : Type) : Constant <<synchronized>> + setVal(offset : int, val : Constant , txnum : long, lsn : long) <<synchronized>> + block() : BlockId <<synchronized>> ~ flush() <<synchronized>> ~ pin() <<synchronized>> ~ unpin() <<synchronized>> ~ isPinned() : boolean <<synchronized>> ~ isModifiedBy(txNum : long) : boolean <<synchronized>> ~ assignToBlock(b : BlockId) <<synchronized>> ~ assignToNew (filename : String, fmtr : PageFormatter)</pre>

Functionality

- **Main Components**
 - `Buffer`: a memory space for caching a block
 - `BufferPoolMgr`: manages the buffer pool
 - `BufferMgr`: provides the access to buffers and manages the pinned buffers for each transaction
 - A transaction waits here if it could not pin any buffer.
 - `PageFormatter`: formats a fresh block

BufferPoolMgr

BufferPoolMgr
<pre>~ BufferPoolMgr(numbuffs : int) <<synchronized>> ~ flushAll() <<synchronized>> ~ flushAll(txnum : long) <<synchronized>> ~ pin(blk : BlockId) : boolean <<synchronized>> ~ pinNew(filename : String, fmtr : PageFormatter) : Buffer <<synchronized>> ~ unpin(buffs : Buffer[]) <<synchronized>> ~ available() : int</pre>

BufferPoolMgr

- Singleton
- Finds a hit for a `pin()`
- Implements the **clock** replacement strategy
- The `pin()` ***returns null immediately*** if there's no candidate buffer
 - Then, the `BufferMgr` make the calling thread waiting and retrying later

Functionality

- Main Components
 - Buffer: a memory space for caching a block
 - BufferPoolMgr: manages the buffer pool
 - BufferMgr: provides the access to buffers and manages the pinned buffers for each transaction
 - A transaction waits here if it could not pin any buffer.
 - PageFormatter: formats a fresh block

BufferMgr

BufferMgr : TransactionLifecycleListener

<<final>> # BUFFER_POOL_SIZE : int

- + BufferMgr()
- + onTxCommit(tx : Transaction)
- + onTxRollback(tx : Transaction)
- + onTxEndStatement(tx : Transaction)
- + pin(blk : BlockId)
- + pinNew(filename : String, fmtr : PageFormatter) : Buffer
- + unpin(buff : Buffer)
- + flushAll()
- + flushAll(txNum)
- + available() : int

BufferMgr

- Created when constructing a transaction
- A `BufferMgr` manages the pinned buffers and the pinning counts of a transaction
- `BufferMgr.pin()` makes the calling thread to wait if there's no candidate buffer for replacement
 - How?

Java `wait()` and `notifyAll()` Methods

- In Java, every object has a waiting list
 - `obj.wait(timeout)` puts the caller thread into the waiting list of `obj`
- The thread will be removed from the list and ready for execution in two conditions:
 - Another thread call `obj.notifyAll()`
 - Timeout elapsed

Java `wait()` and `notifyAll()` Methods

- If...
 1. `obj.wait()` is surrounded by a synchronized block, and
 2. there are multiple threads in `obj`'s waiting list,
- Then when `notifyAll()` is called, **all** waiting threads will compete on the lock to enter the synchronized block
 - **No** FIFO guarantee which thread will be notified first, and which will acquire the lock first
 - Only one thread wins the lock, others **blocked** until the winner releases the lock

BufferMgr

- `pin()`: if `BufferPoolMgr` returns null, put the current thread into `BufferPoolMgr`'s waiting list

```
buff = bufferPool.pin(blk);  
while (buff == null && !waitingTooLong(timestamp)) {  
    bufferPool.wait(MAX_TIME);  
    buff = bufferPool.pin(blk);  
}
```

- `unpin(buff)`: notify all threads in `BufferPoolMgr`'s waiting list
 - Only one thread will pin successfully due to the synchronization


```

public Buffer pin(BlockId blk) {
    synchronized (bufferPool) {
        PinnedBuffer pinnedBuff = pinnedBuffers.get(blk);
        if (pinnedBuff != null) {
            pinnedBuff.pinnedCount++;
            return pinnedBuff.buffer;
        }
        if (pinnedBuffers.size() == BUFFER_POOL_SIZE)
            throw new BufferAbortException();
        try {
            Buffer buff;
            long timestamp = System.currentTimeMillis();
            buff = bufferPool.pin(blk);
            if (buff == null) {
                waitingThreads.add(Thread.currentThread());
                while (buff == null && !waitingTooLong(timestamp)) {
                    bufferPool.wait(MAX_TIME);
                    if (waitingThreads.get(0).equals(Thread.currentThread()))
                        buff = bufferPool.pin(blk);
                }
                waitingThreads.remove(Thread.currentThread());
                bufferPool.notifyAll();
            }
            if (buff == null) {
                repin();
                buff = pin(blk);
            } else {
                pinnedBuffers.put(buff.block(), new PinnedBuffer(buff));
            }
            return buff;
        } catch (InterruptedException e) {
            throw new BufferAbortException();
        }
    }
}

```

```

public Buffer pin(BlockId blk) {
    synchronized (bufferPool) {
        PinnedBuffer pinnedBuff = pinnedBuffers.get(blk);
        if (pinnedBuff != null) {
            pinnedBuff.pinnedCount++;
            return pinnedBuff.buffer;
        }
        if (pinnedBuffers.size() == BUFFER_POOL_SIZE)
            throw new BufferAbortException();
        try {
            Buffer buff;
            long timestamp = System.currentTimeMillis();
            buff = bufferPool.pin(blk);
            if (buff == null) {
                waitingThreads.add(Thread.currentThread());
                while (buff == null && !waitingTooLong(timestamp)) {
                    bufferPool.wait(MAX_TIME);
                    if (waitingThreads.get(0).equals(Thread.currentThread()))
                        buff = bufferPool.pin(blk);
                }
                waitingThreads.remove(Thread.currentThread());
                bufferPool.notifyAll();
            }
            if (buff == null) {
                repin();
                buff = pin(blk);
            } else {
                pinnedBuffers.put(buff.block(), new PinnedBuffer(buff));
            }
            return buff;
        } catch (InterruptedException e) {
            throw new BufferAbortException();
        }
    }
}

```

Synchronize on the buffer pool (singleton)

```

public Buffer pin(BlockId blk) {
    synchronized (bufferPool) {
        PinnedBuffer pinnedBuff = pinnedBuffers.get(blk);
        if (pinnedBuff != null) {
            pinnedBuff.pinnedCount++;
            return pinnedBuff.buffer;
        }
        if (pinnedBuffers.size() == BUFFER_POOL_SIZE)
            throw new BufferAbortException();
        try {
            Buffer buff;
            long timestamp = System.currentTimeMillis();
            buff = bufferPool.pin(blk);
            if (buff == null) {
                waitingThreads.add(Thread.currentThread());
                while (buff == null && !waitingTooLong(timestamp)) {
                    bufferPool.wait(MAX_TIME);
                    if (waitingThreads.get(0).equals(Thread.currentThread()))
                        buff = bufferPool.pin(blk);
                }
                waitingThreads.remove(Thread.currentThread());
                bufferPool.notifyAll();
            }
            if (buff == null) {
                repin();
                buff = pin(blk);
            } else {
                pinnedBuffers.put(buff.block(), new PinnedBuffer(buff));
            }
            return buff;
        } catch (InterruptedException e) {
            throw new BufferAbortException();
        }
    }
}

```

Find the given block from the pinned buffers of this transaction

```

public Buffer pin(BlockId blk) {
    synchronized (bufferPool) {
        PinnedBuffer pinnedBuff = pinnedBuffers.get(blk);
        if (pinnedBuff != null) {
            pinnedBuff.pinnedCount++;
            return pinnedBuff.buffer;
        }
        if (pinnedBuffers.size() == BUFFER_POOL_SIZE)
            throw new BufferAbortException();
        try {
            Buffer buff;
            long timestamp = System.currentTimeMillis();
            buff = bufferPool.pin(blk);
            if (buff == null) {
                waitingThreads.add(Thread.currentThread());
                while (buff == null && !waitingTooLong(timestamp)) {
                    bufferPool.wait(MAX_TIME);
                    if (waitingThreads.get(0).equals(Thread.currentThread()))
                        buff = bufferPool.pin(blk);
                }
                waitingThreads.remove(Thread.currentThread());
                bufferPool.notifyAll();
            }
            if (buff == null) {
                repin();
                buff = pin(blk);
            } else {
                pinnedBuffers.put(buff.block(), new PinnedBuffer(buff));
            }
            return buff;
        } catch (InterruptedException e) {
            throw new BufferAbortException();
        }
    }
}

```

Pins the requested block

Add the buffer to the pinned list of this transaction

```

public Buffer pin(BlockId blk) {
    synchronized (bufferPool) {
        PinnedBuffer pinnedBuff = pinnedBuffers.get(blk);
        if (pinnedBuff != null) {
            pinnedBuff.pinnedCount++;
            return pinnedBuff.buffer;
        }
        if (pinnedBuffers.size() == BUFFER_POOL_SIZE)
            throw new BufferAbortException();
        try {
            Buffer buff;
            long timestamp = System.currentTimeMillis();
            buff = bufferPool.pin(blk);
            if (buff == null) {
                waitingThreads.add(Thread.currentThread());
                while (buff == null && !waitingTooLong(timestamp)) {
                    bufferPool.wait(MAX_TIME);
                    if (waitingThreads.get(0).equals(Thread.currentThread()))
                        buff = bufferPool.pin(blk);
                }
                waitingThreads.remove(Thread.currentThread());
                bufferPool.notifyAll();
            }
            if (buff == null) {
                repin();
                buff = pin(blk);
            } else {
                pinnedBuffers.put(buff.block(), new PinnedBuffer(buff));
            }
            return buff;
        } catch (InterruptedException e) {
            throw new BufferAbortException();
        }
    }
}

```

*If there was not any available buffer,
make the thread waiting*

The thread in the head of the list can pin

Wake up other thread again

```

public Buffer pin(BlockId blk) {
    synchronized (bufferPool) {
        PinnedBuffer pinnedBuff = pinnedBuffers.get(blk);
        if (pinnedBuff != null) {
            pinnedBuff.pinnedCount++;
            return pinnedBuff.buffer;
        }
        if (pinnedBuffers.size() == BUFFER_POOL_SIZE)
            throw new BufferAbortException();
        try {
            Buffer buff;
            long timestamp = System.currentTimeMillis();
            buff = bufferPool.pin(blk);
            if (buff == null) {
                waitingThreads.add(Thread.currentThread());
                while (buff == null && !waitingTooLong(timestamp)) {
                    bufferPool.wait(MAX_TIME);
                    if (waitingThreads.get(0).equals(Thread.currentThread()))
                        buff = bufferPool.pin(blk);
                }
                waitingThreads.remove(Thread.currentThread());
                bufferPool.notifyAll();
            }
            if (buff == null) {
                repin();
                buff = pin(blk);
            } else {
                pinnedBuffers.put(buff.block(), new PinnedBuffer(buff));
            }
            return buff;
        } catch (InterruptedException e) {
            throw new BufferAbortException();
        }
    }
}

```

**Waiting too long? There might be deadlock.
Re-pin all blocks**

```

public Buffer pin(BlockId blk) {
    synchronized (bufferPool) {
        PinnedBuffer pinnedBuff = pinnedBuffers.get(blk);
        if (pinnedBuff != null) {
            pinnedBuff.pinnedCount++;
            return pinnedBuff.buffer;
        }
        if (pinnedBuffers.size() == BUFFER_POOL_SIZE)
            throw new BufferAbortException();
        try {
            Buffer buff;
            long timestamp = System.currentTimeMillis();
            buff = bufferPool.pin(blk);
            if (buff == null) {
                waitingThreads.add(Thread.currentThread());
                while (buff == null && !waitingTooLong(timestamp)) {
                    bufferPool.wait(MAX_TIME);
                    if (waitingThreads.get(0).equals(Thread.currentThread()))
                        buff = bufferPool.pin(blk);
                }
                waitingThreads.remove(Thread.currentThread());
                bufferPool.notifyAll();
            }
            if (buff == null) {
                repin();
                buff = pin(blk);
            } else {
                pinnedBuffers.put(buff.block(), new PinnedBuffer(buff));
            }
            return buff;
        } catch (InterruptedException e) {
            throw new BufferAbortException();
        }
    }
}

```

Self-deadlock: throw exception

BufferMgr vs. BufferPoolMgr

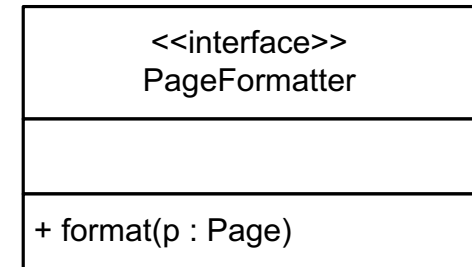
- Each transaction has its own BufferMgr, but there is only one BufferPoolMgr
- Responsibility
 - BufferPoolMgr manages the buffer pool
 - BufferMgr handles waiting for pinning and manages pinned buffers for each transaction

Functionality

- **Main Components**
 - `Buffer`: a memory space for caching a block
 - `BufferPoolMgr`: manages the buffer pool
 - `BufferMgr`: provides the access to buffers and manages the pinned buffers for each transaction
 - A transaction waits here if it could not pin any buffer.
 - `PageFormatter`: **formats a fresh block**

PageFormatter

- The `pinNew(fmtr)` method of `BufferMgr` appends a new block to a file
- `PageFormatter` initializes the block
 - To be extended in packages (`storage.record` and `storage.index.btree`) where the semantics of records are defined



```
class ZeroIntFormatter implements PageFormatter {  
    public void format(Page p) {  
        Constant zero = new IntegerConstant(0);  
        int recsize = Page.size(zero);  
        for (int i = 0; i + recsize <= Page.BLOCK_SIZE; i += recsize)  
            p.setVal(i, zero);  
    }  
}
```