CSE 132C : Database System Implementation

PROJECT 1: BADGERDB BUFFER MANAGER IMPLEMENTATION

Discussion : Friday, April 9, 2021

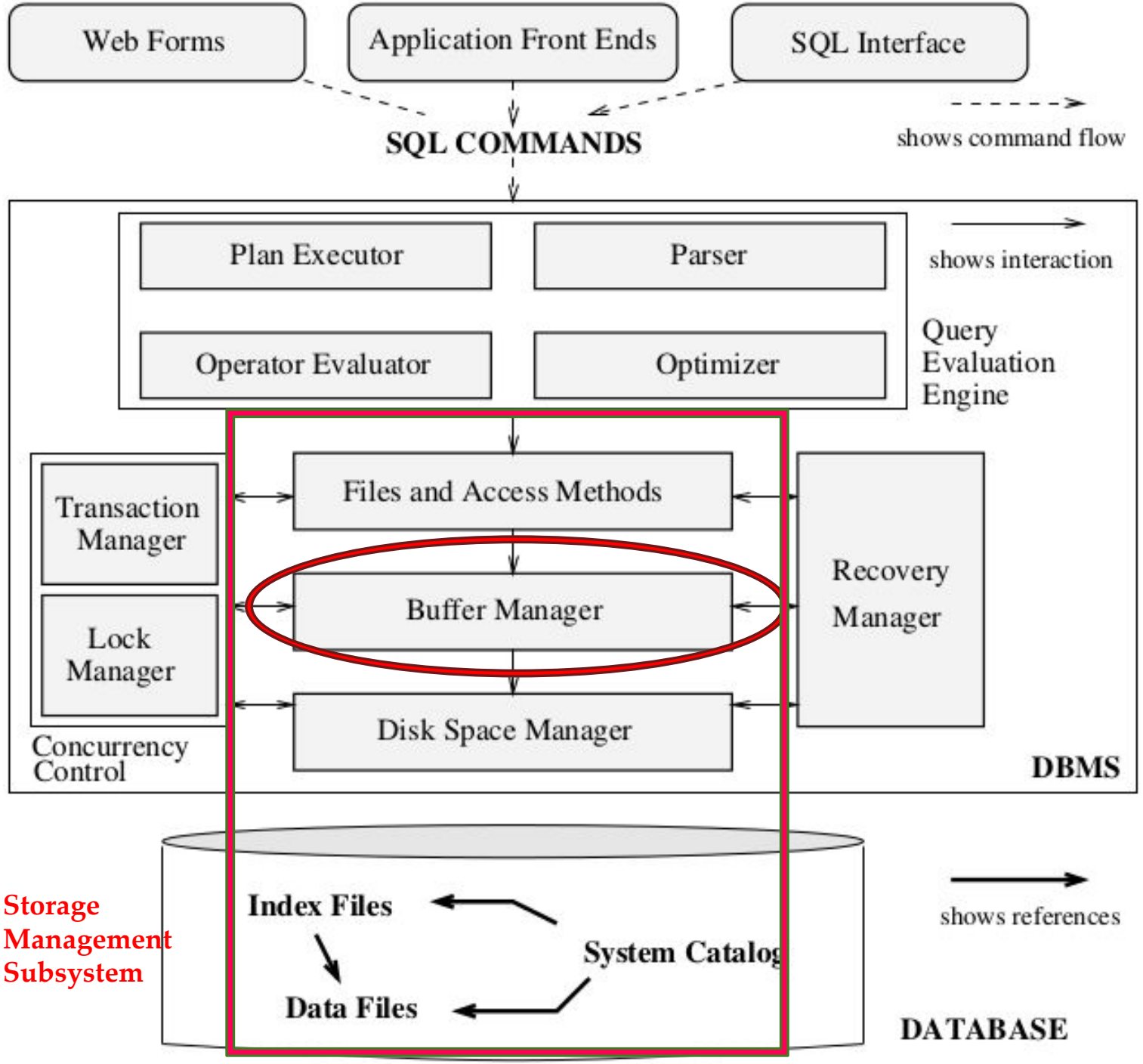
Presenter : Tanay Karve

Slide contribution : Arun Kumar, Apul Jain, Anirudh Singh Shekhawat

Project Assignment

- Build internals of a simple RDBMS
- Project 1 : Buffer Manager
- Project 2 : B+ Tree Indexer(Later in the course)

Buffer Management



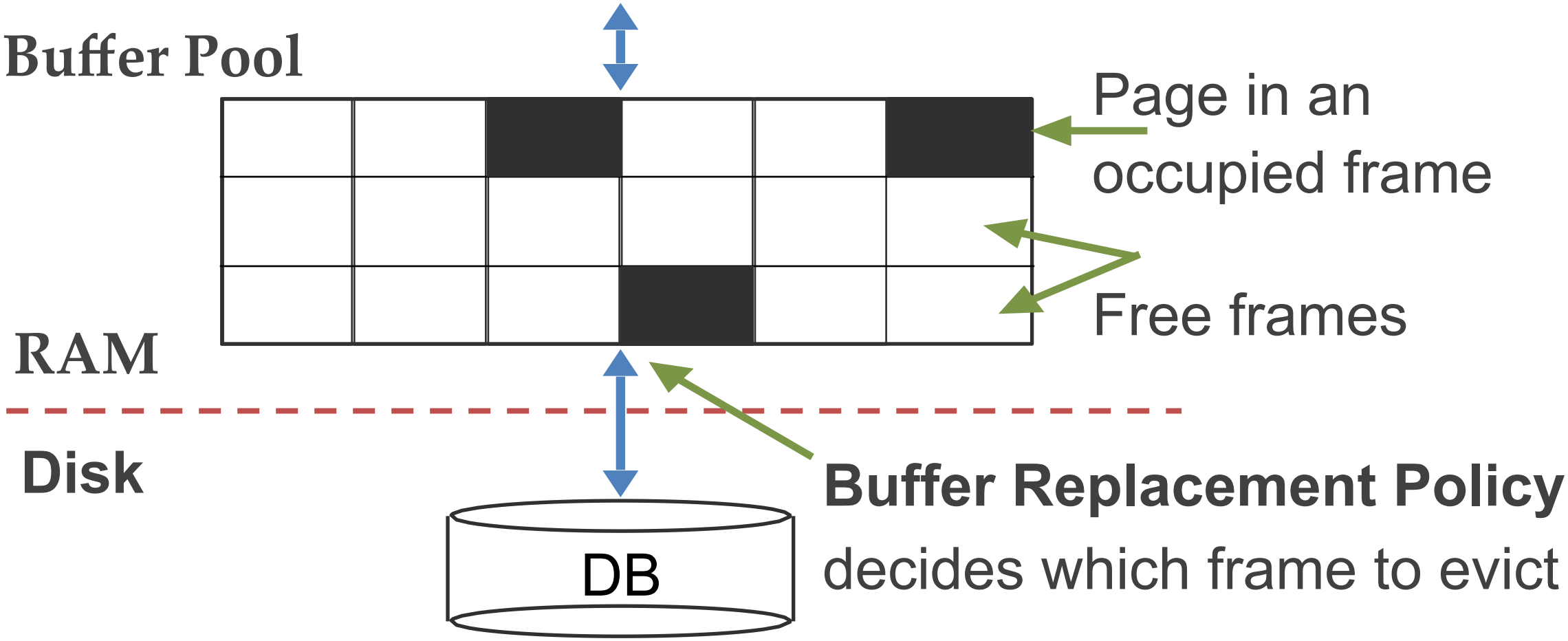
Where to start

- All the auxiliary functionality for your Buffer Manager is already implemented.
  - Create/destroy files
  - Allocate/deallocate pages
  - Read/write pages
  - More!

- The BadgerDB I/O layer handles reading and writes files and the pages within files.
- We provide you data structures to hold the buffer pool and the description of the frames within the buffer pool.
- There is an interface to help you get started with implementing the buffer management algorithm.
- And these slides to help you understand the code base and the algorithm!

Buffer Management

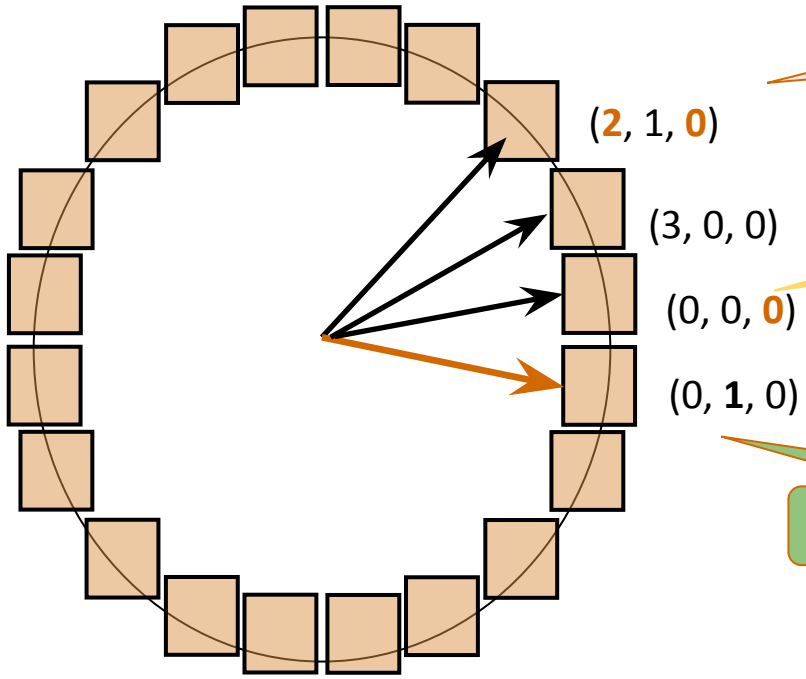
Page Requests from Higher Levels of DBMS



Buffer Management: The Clock Algorithm

Frame:
- pinCnt
- dirty
- refbit
- …

Requested page in Buffer Pool:
- pinCnt++/ Return handle to frame
- Else read page in from disk
  - find space in the buffer pool!



Page in use and dirty

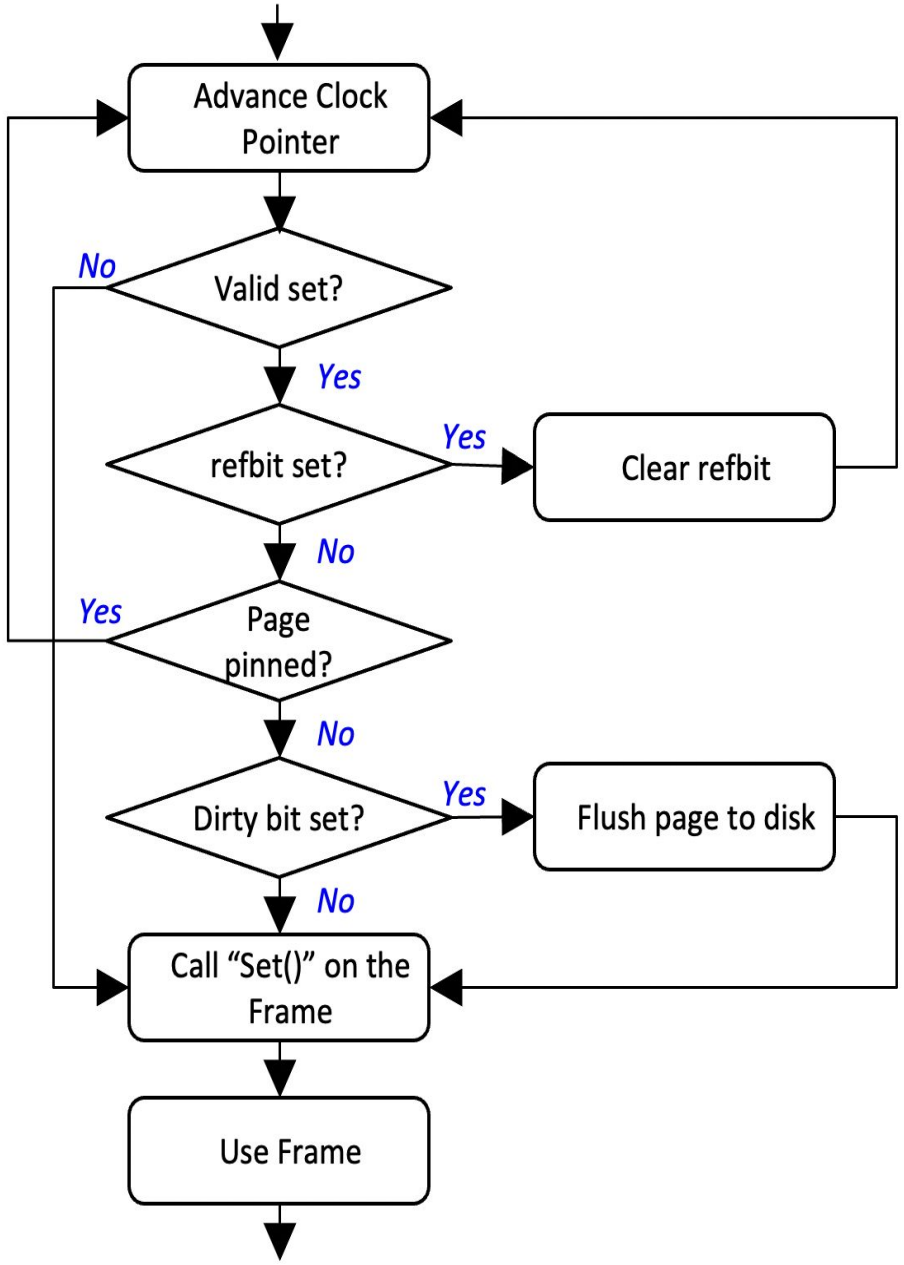
Page in use but not dirty

Page unpinned and now not referenced

Page dirty but not ref/pinned

Use this frame!

Flowchart


Advance Clock Pointer
Valid set?
No
Yes
refbit set?
Yes
Clear refbit
No
Page pinned?
Yes
No
Dirty bit set?
Yes
Flush page to disk
No
Call "Set()" on the Frame
Use Frame

BadgerDB IO Layer -- The File class and Page Class

Page Class

- `Page allocatePage();`
- `void writePage(const Page& new_page);`
- `Page readPage(const PageId page_number) const;`
- `void deletePage(const PageId page_number);`

File Class

- create
- open
- remove
- isOpen
- filename
- More!

The Structure of the Buffer Manager

Three main classes:

- BufMgr
- BufDesc
- BufHashTbl

BufDesc Class

- Used to keep track of the state of each frame
- Buffer is described by four attributes : Dirty Bit, Pin Count, Reference Bit, Valid Bit
- Use the void Set(File* filePtr, PageId pageNum) to initialize the buffer description.
- Use the void Clear() method to reset the buffer description.

```
class BufDesc {
  private:
    File* file;        // pointer to file object
    PageId pageNo;     // page within file
    FrameId frameNo;   // buffer pool frame number
    int pinCnt;             // number of times this page has been pinned
    bool dirty;        // true if dirty; false otherwise
    bool valid;        // true if page is valid
    bool refbit;            // true if this buffer frame been referenced recently

    void Clear();      // initialize buffer frame
    void Set(File* filePtr, PageId pageNum);  //set BufDesc member variable values
    void Print()           //Print values of member variables
    BufDesc();         //Constructor
```

BufHashTbl Class

- Used to keep track of the pages in the buffer pool.
- Maps file and page numbers to buffer pool frames.
- Specifically, provides insert, remove and lookup functionality.
- Implemented using chained bucket hashing.

```
    // insert entry into hash table mapping (file, pageNo) to frameNo
    void insert(const File* file, const int pageNo, const int frameNo);

    // Check if (file,pageNo) is currently in the buffer pool (ie. in
    // the hash table. If so, return the corresponding frame number in
frameNo.
    void lookup(const File* file, const int pageNo, int& frameNo);

    // remove entry obtained by hashing (file,pageNo) from hash table.
    void remove(const File* file, const int pageNo);
```

BufMgr Class

The BufMgr class is the heart of the buffer manager. This (buffer.cpp) is where you write your code for this assignme

```
class BufMgr {
private :
    FrameId clockHand; // clock hand for clock algorithm
    BufHashTbl *hashTable ; // hash table mapping (File,page) to frame number
    BufDesc *bufDescTable ; // BufDesc objects , one per frame
    std::uint32_t numBufs ; // Number of frames in the buffer pool
    BufStats bufStats; // Statistics about buffer pool usage allocate a free frame using the clock algorithm

    void allocBuf(FrameId & frame);
    void advanceClock () ; //Advance clock to next frame in the buffer pool

public :
    Page *bufPool ; // actual buffer pool
    BufMgr( std::uint32_t bufs ) ; // Constructor
    ~BufMgr ( ) ; // Destructor
    void readPage(File* file , const PageId PageNo, Page*& page);
    void unPinPage(File* file , const PageId PageNo, const bool dirty);
    void allocPage(File* file , PageId& PageNo, Page*& page);
    void disposePage( File* file , const PageId pageNo) ;
    void flushFile(const File* file);
};
```

To sum it up

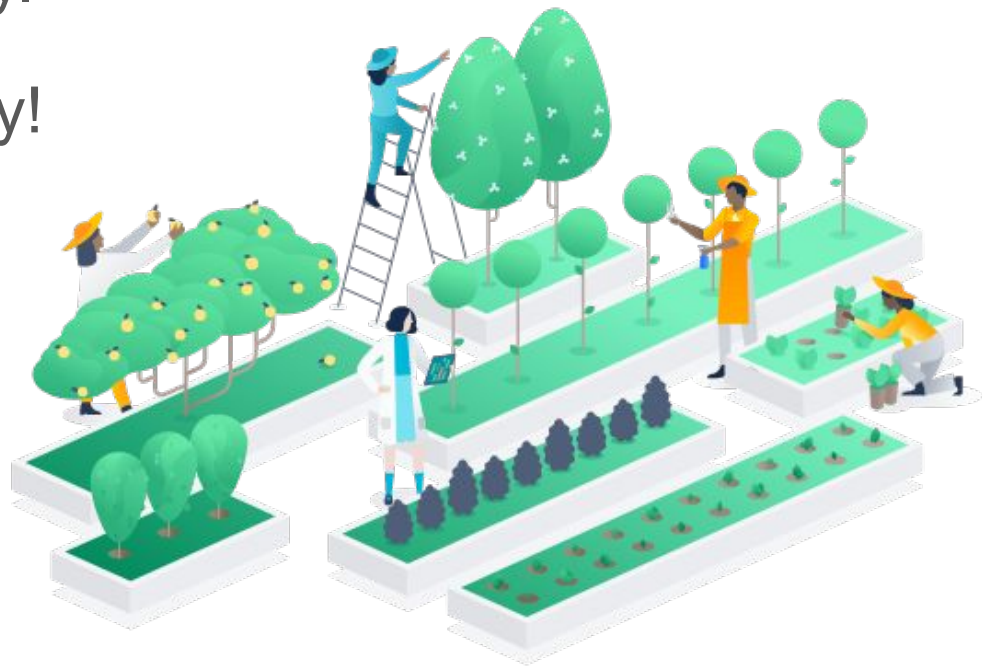
You need to implement:

- `~BufMgr();`
- `void advanceClock();`
- `void allocBuf(FrameId& frame);`
- `void readPage(File *file, const PageId PageNo, Page*& page);`
- `void unPinPage(File *file, const PageID PageNo, const bool dirty)`
- `void allocPage(File * file, PageID & PageNo, Page *& Page)`
- `void disposePage(File * file, const PageId PageNo)`
- `void flushFile(File *file)`

AGILE

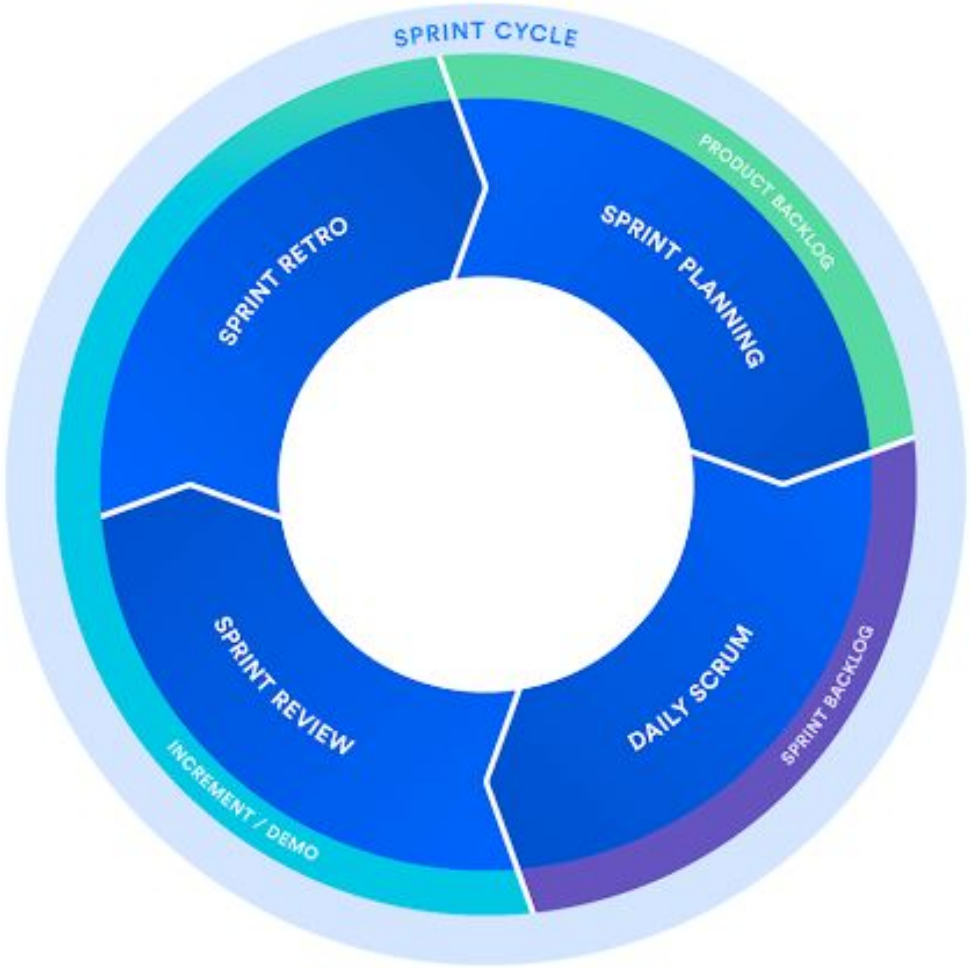
An iterative development approach to managing software development projects that helps finishing deliverables more efficiently.

Most widely used technique in the tech industry!



https://www.atlassian.com/agile

AGILE

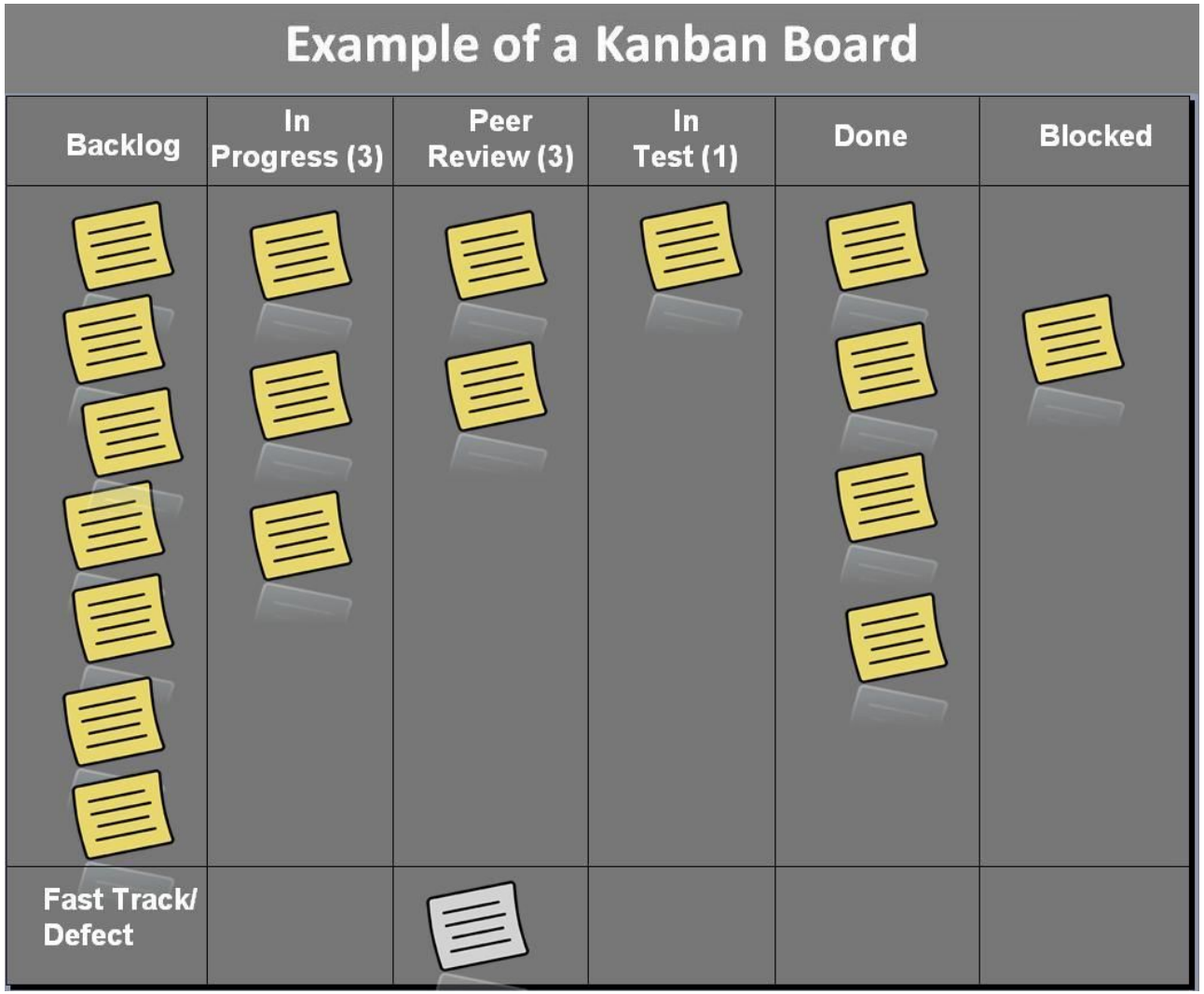


Develop in sprints.

1. Periodic scrum meetings.
2. Plan tasks for the sprint.
3. Code!
4. Review what worked / didn't work.
5. Repeat!

https://www.atlassian.com/agile/scrum

AGILE | Kanban



https://en.wikipedia.org/wiki/Kanban_(development)

Thank you!