CS564: Assignment 5

HeapFile Manager

Brian (Zhi) Zheng

Part of the slides are borrowed from past TAs in CS564

Overview

1. General Questions about Minirel
2. Review for Buffer Manager
3. HeapFile Manager
4. Logistics

FAQs

- **Return arguments in C**

```
const Status getStudent(const int studentId, Student& studentOutput)
```

FAQs

- **Return arguments in C**

```
const Status getStudent(const int studentId, Student& studentOutput)
```

- **Row Storage vs Column Storage**

  Minirel simply uses row storage. Columns in the same record are consecutive.

FAQs

- **Return arguments in C**

```
const Status getStudent(const int studentId, Student& studentOutput)
```

- **Row Storage vs Column Storage**

  Minirel simply uses row storage. Columns in the same record are consecutive.

- **HeapFile object vs heap file on disk**

  HeapFile is a class whose objects programmatically represent heap files on disk

```
const Status createHeapFile(const string fileName)
```

```
HeapFile::HeapFile(const string & fileName, Status& returnStatus)
```

Review

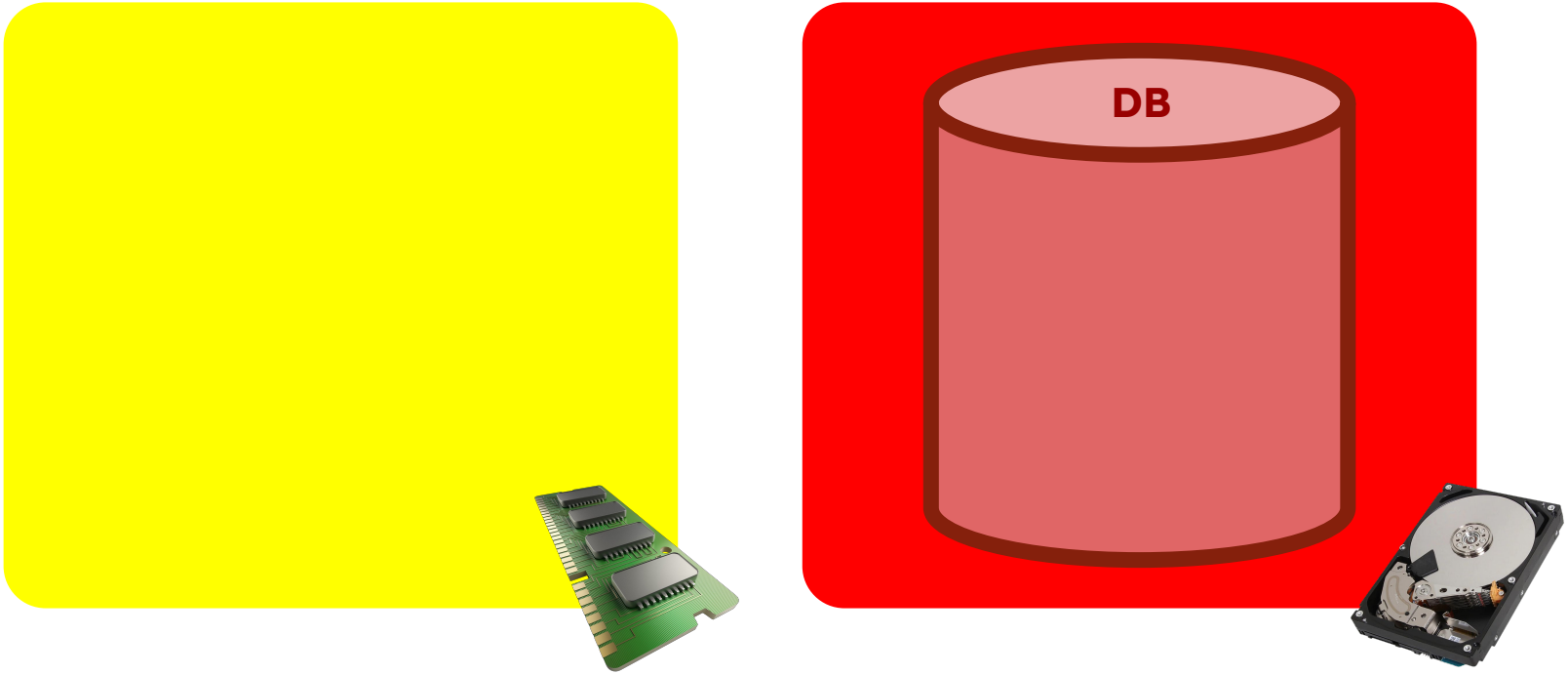
Overview

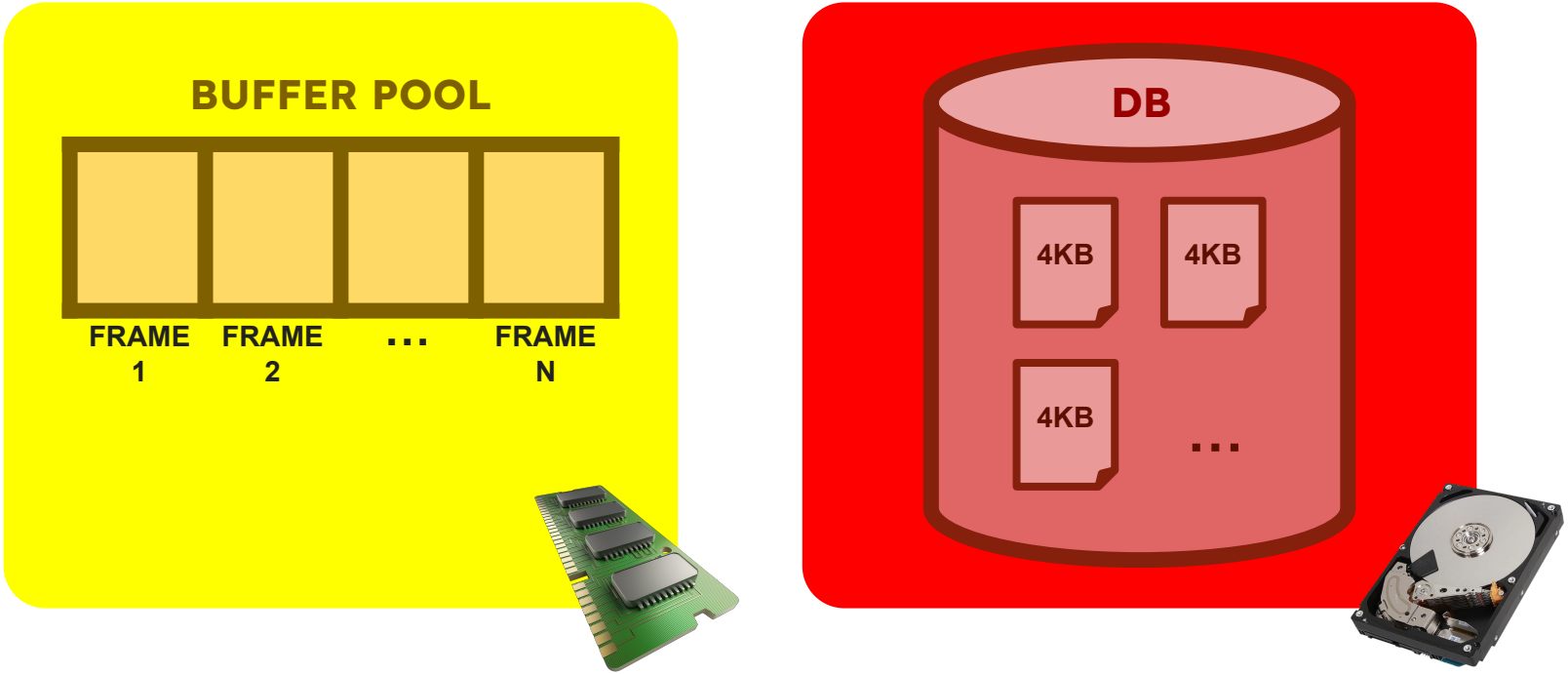
3. Buffer Manager

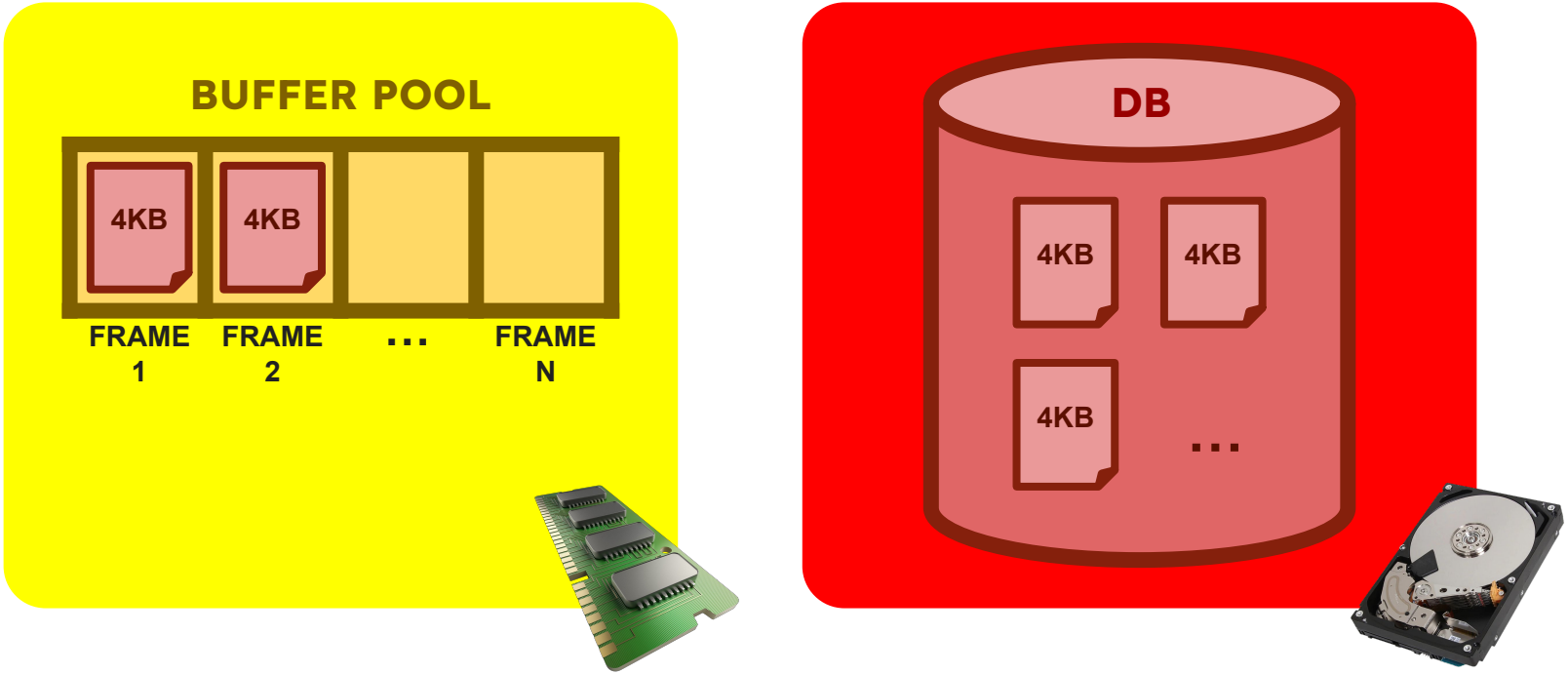
4. HeapFile Manager

5. Front-End and Database Utilities

6. Query and Update Operators


DB


BUFFER POOL
FRAME 1
FRAME 2
...
FRAME N
DB
4KB
4KB
4KB
...


BUFFER POOL
4KB
4KB
FRAME 1
FRAME 2
...
FRAME N
DB
4KB
4KB
4KB
...

What we have so far?

- Allocate page in the buffer & file
  - `const Status BufMgr::allocPage(File* file, int& pageNo, Page*& page)`
- Have a naive way to write to the page
  - `sprintf((char*)page, "test.1 Page %d %7.1f", j[i], (float)j[i]);`
- Flush page associated with the file to disk in clock algorithm
  - `file->writePage(bufTable[clockHand].pageNo, &bufPool[clockHand]);`
  - `lseek() + write()in db.C`

What's missing

- Organized file structure [Logical way to access the data]
- Records in the page?
  - `struct Record {void* data; int length;};`
  - RID!!!!
  - `struct RID{ int pageNo; int slotNo;};`
- How to scan/filter the records?
- How to insert the records?
  - `InsertFileScan->insertRecord(Record, RID&);`
- How to write to the page?
  - `const Status Page::insertRecord(const Record & rec, RID& rid)`

Overview

3. Buffer Manager

4. HeapFile Manager

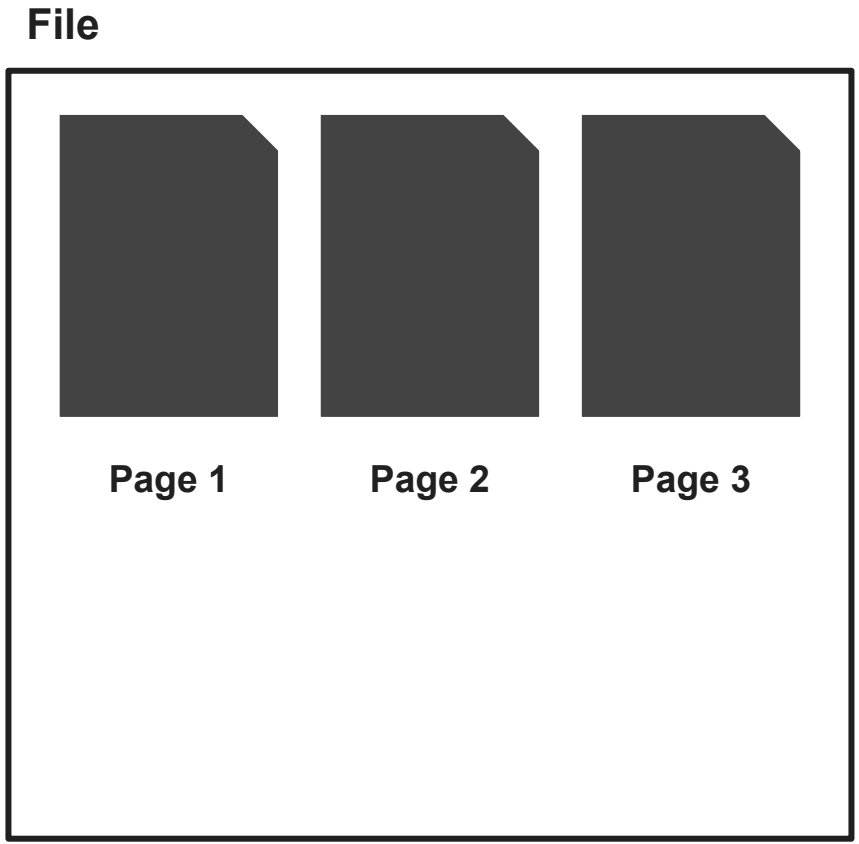
5. Front-End and Database Utilities

6. Query and Update Operator

How data is arranged

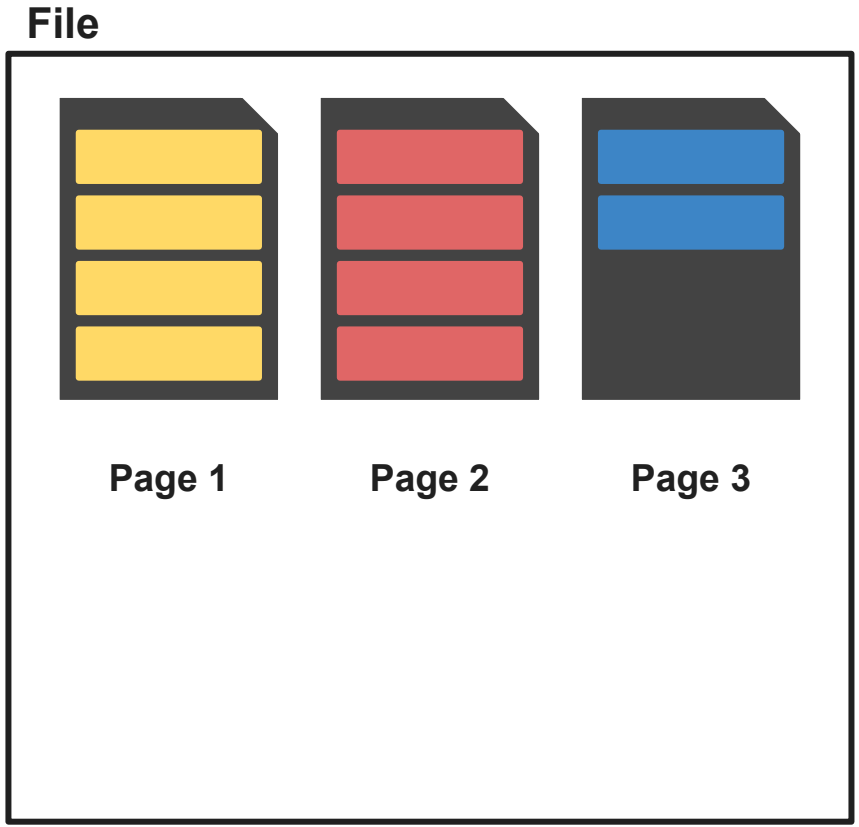
How data is arranged

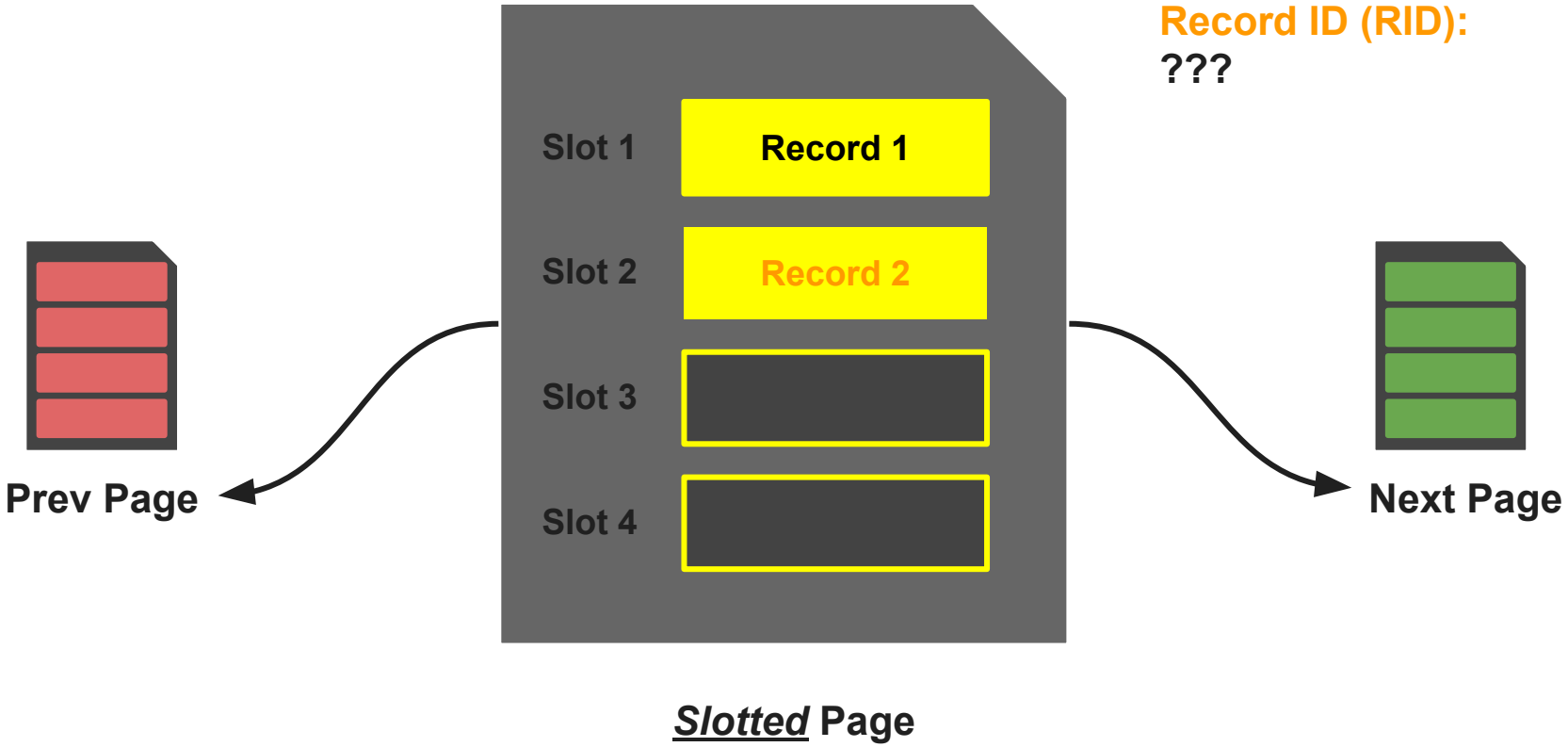
Albert Einstein	1879	Physics
Marie Curie	1876	Chemistry
Charles Darwin	1890	Biology
Galileo Galilei	1564	Astronomy
Stephen Hawking	1942	Physics
Nikola Tesla	1876	Physics
James Watson	1928	Genetics
Ada Lovelace	1907	Computers
Rachel Carson	1928	Physics
Richard Feynman	1918	Physics



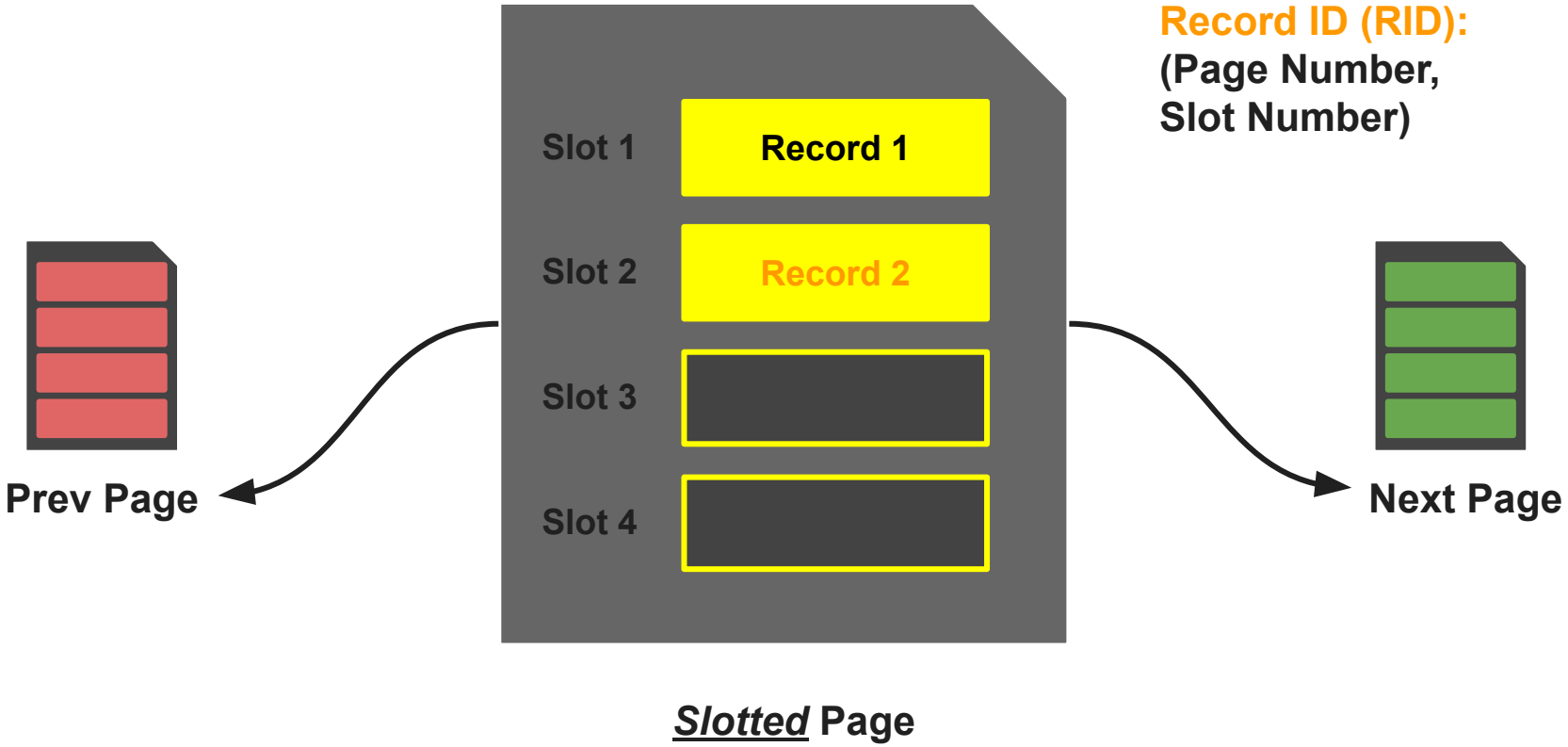
How data is arranged

Albert Einstein	1879	Physics
Marie Curie	1876	Chemistry
Charles Darwin	1890	Biology
Galileo Galilei	1564	Astronomy
Stephen Hawking	1942	Physics
Nikola Tesla	1876	Physics
James Watson	1928	Genetics
Ada Lovelace	1907	Computers
Rachel Carson	1928	Physics
Richard Feynman	1918	Physics





Slotted **Page**



Slotted **Page**

Heapfiles (unrelated to Heap data structure)

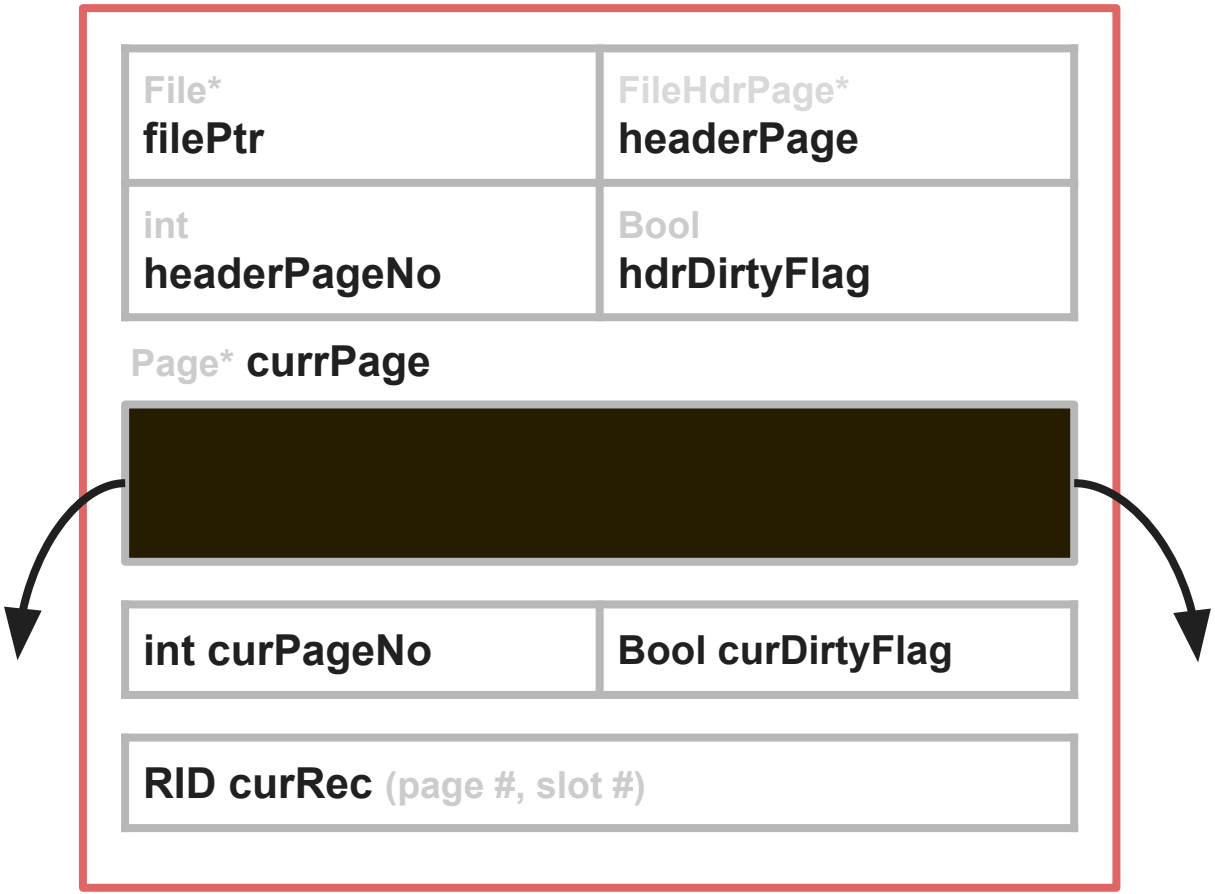
- Unordered set of records
- Heap files can be created and destroyed
- Existing heapfiles can be opened and closed
- Records and pages can be inserted and deleted
- Records are uniquely identified by a record id (RID)
- First page is a special Header-Page

Why HeapFiles?

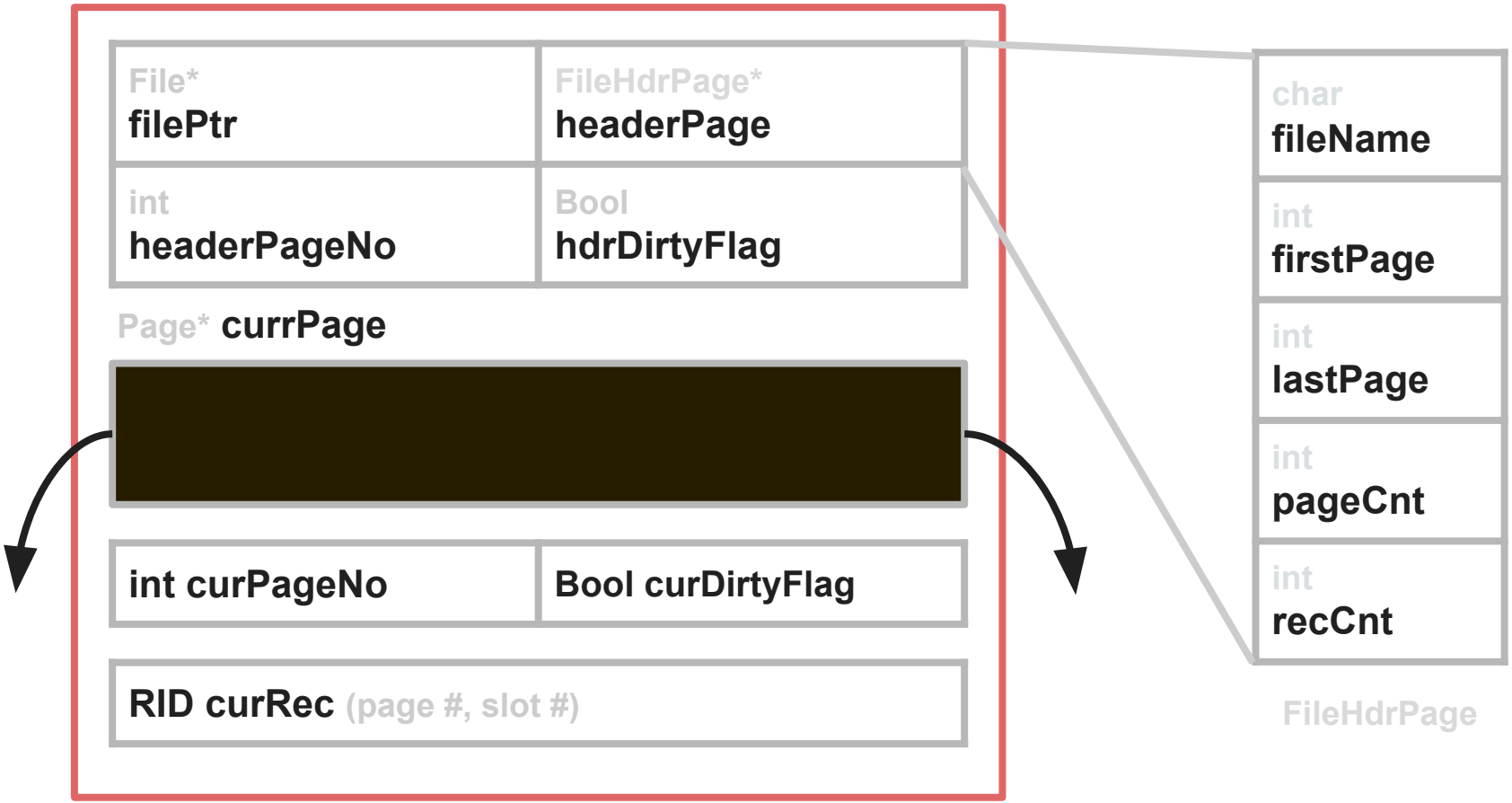
- **Simplicity**
- **Fast Inserts**
- **Space Efficiency**
- **Alternatives:**
  Sorted files, B-Trees, Hash Indexes, ...

Programmatic Representation

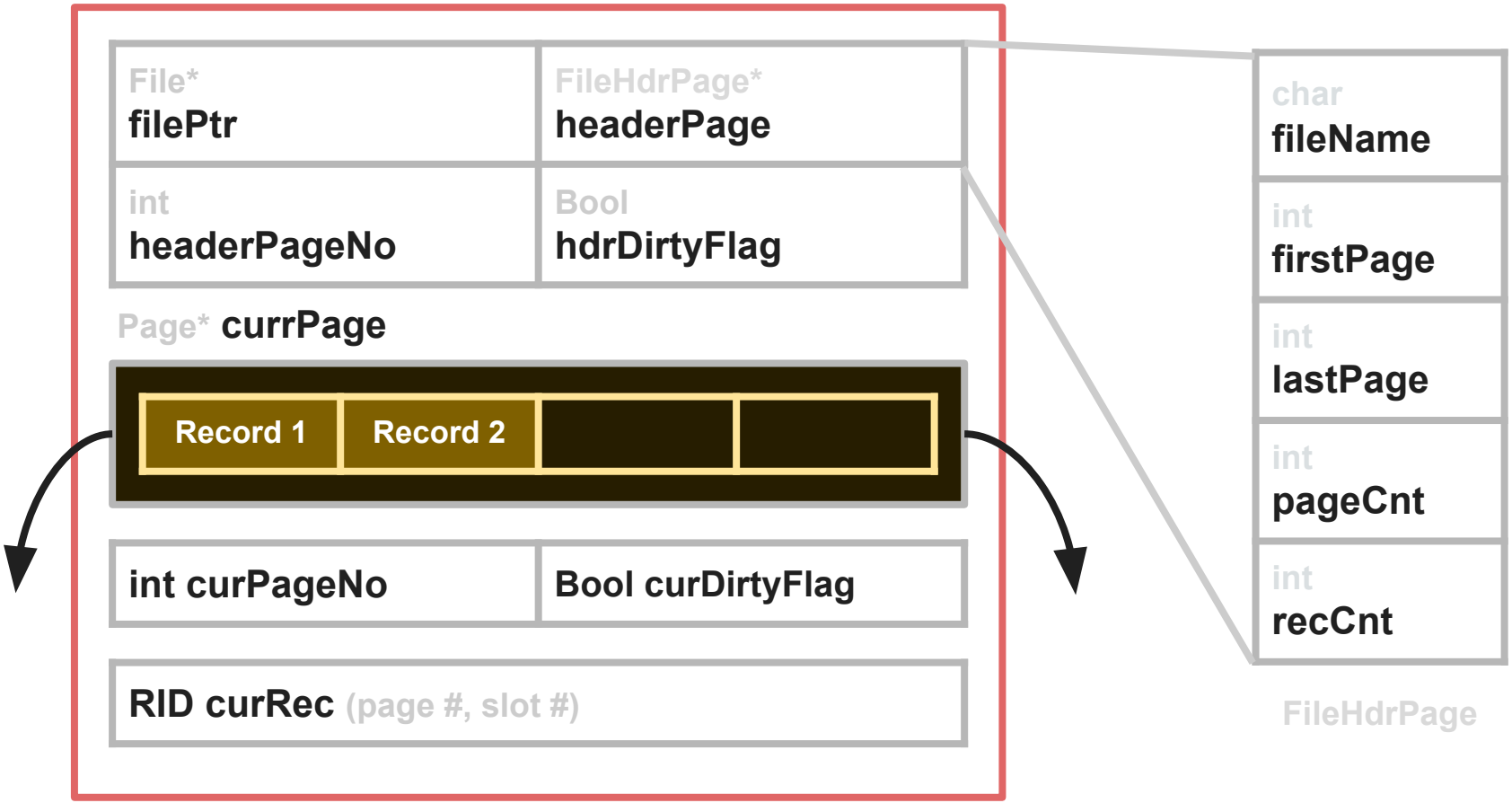
HeapFile Class

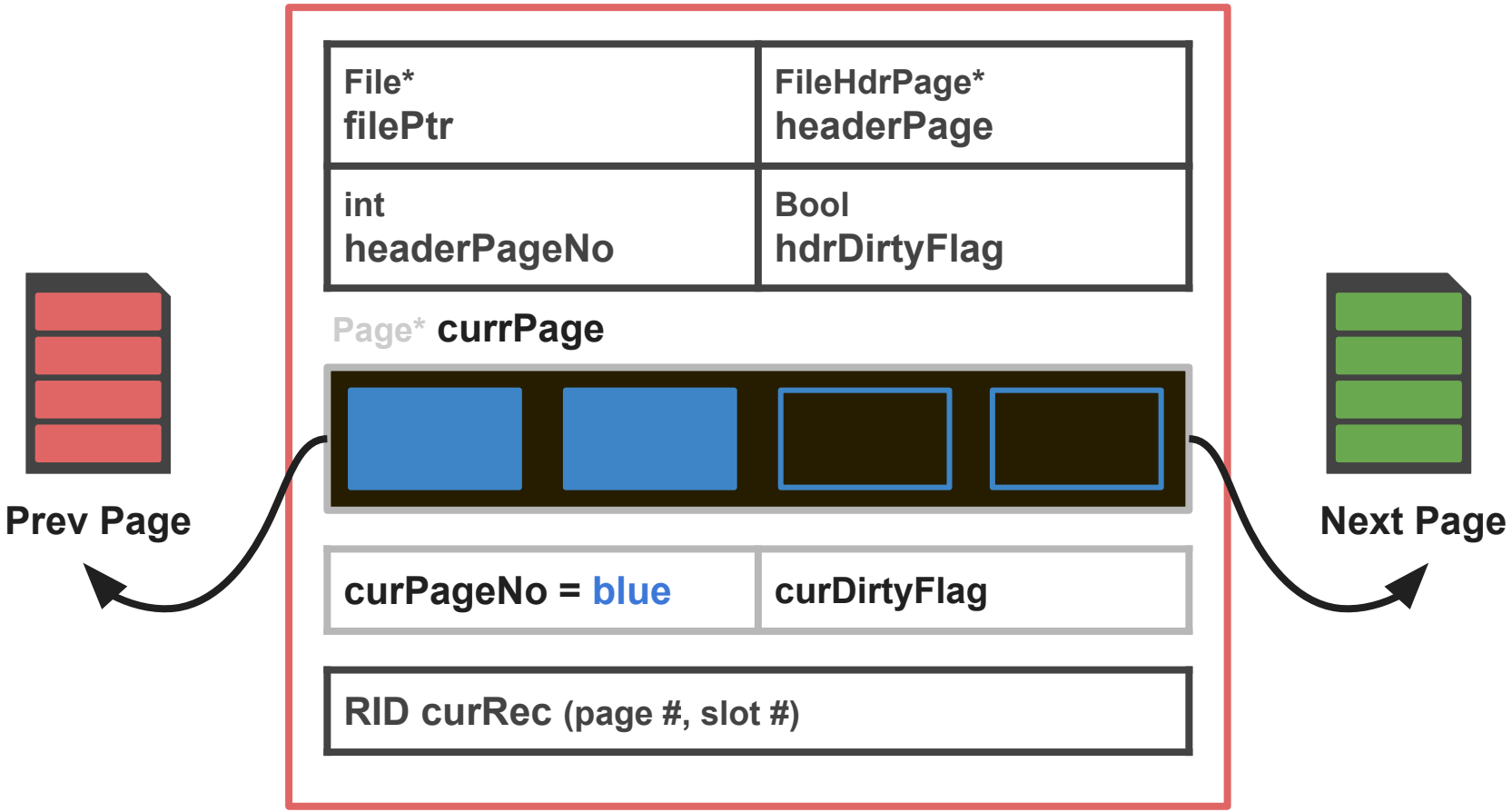


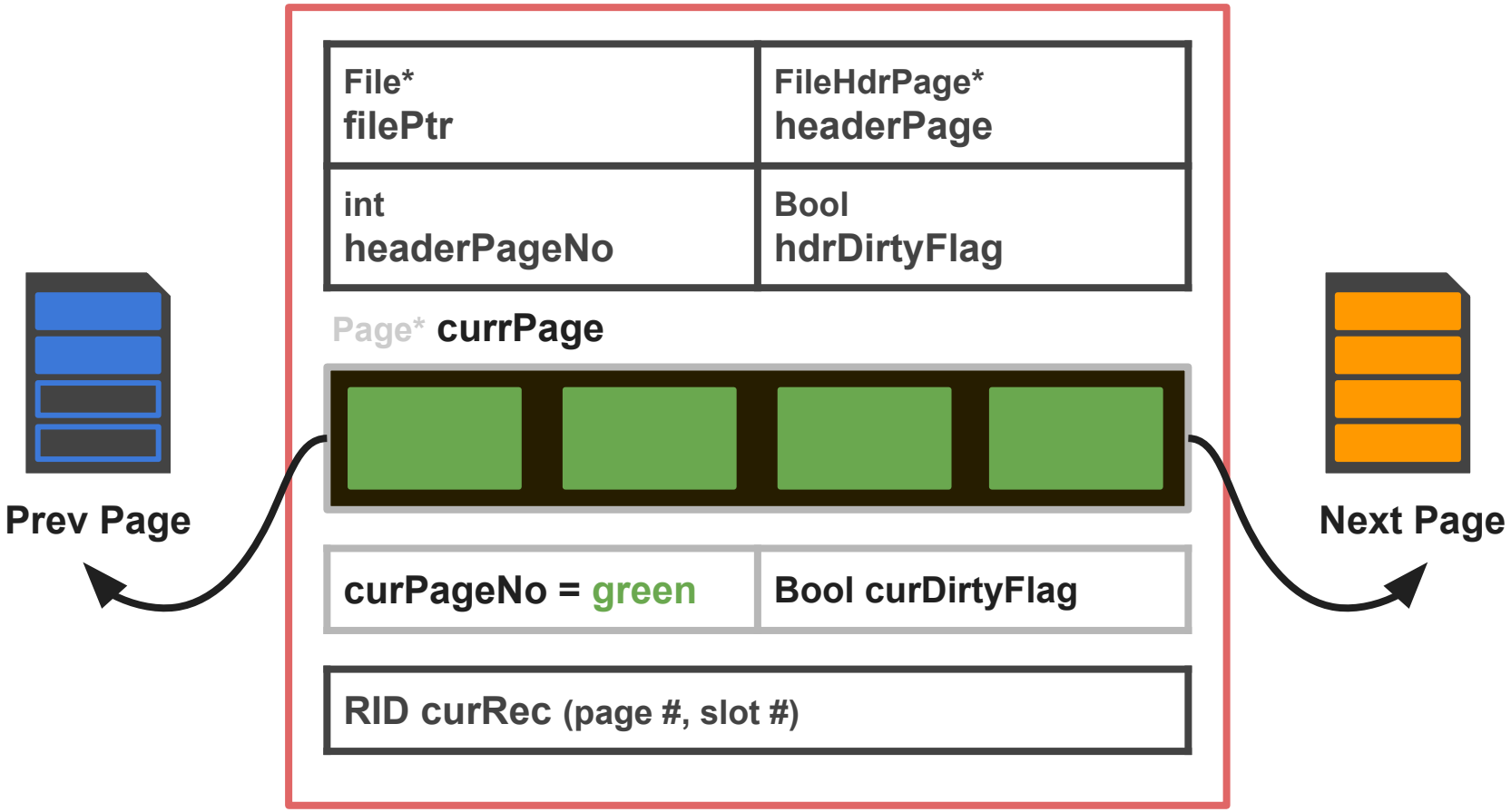
HeapFile Class

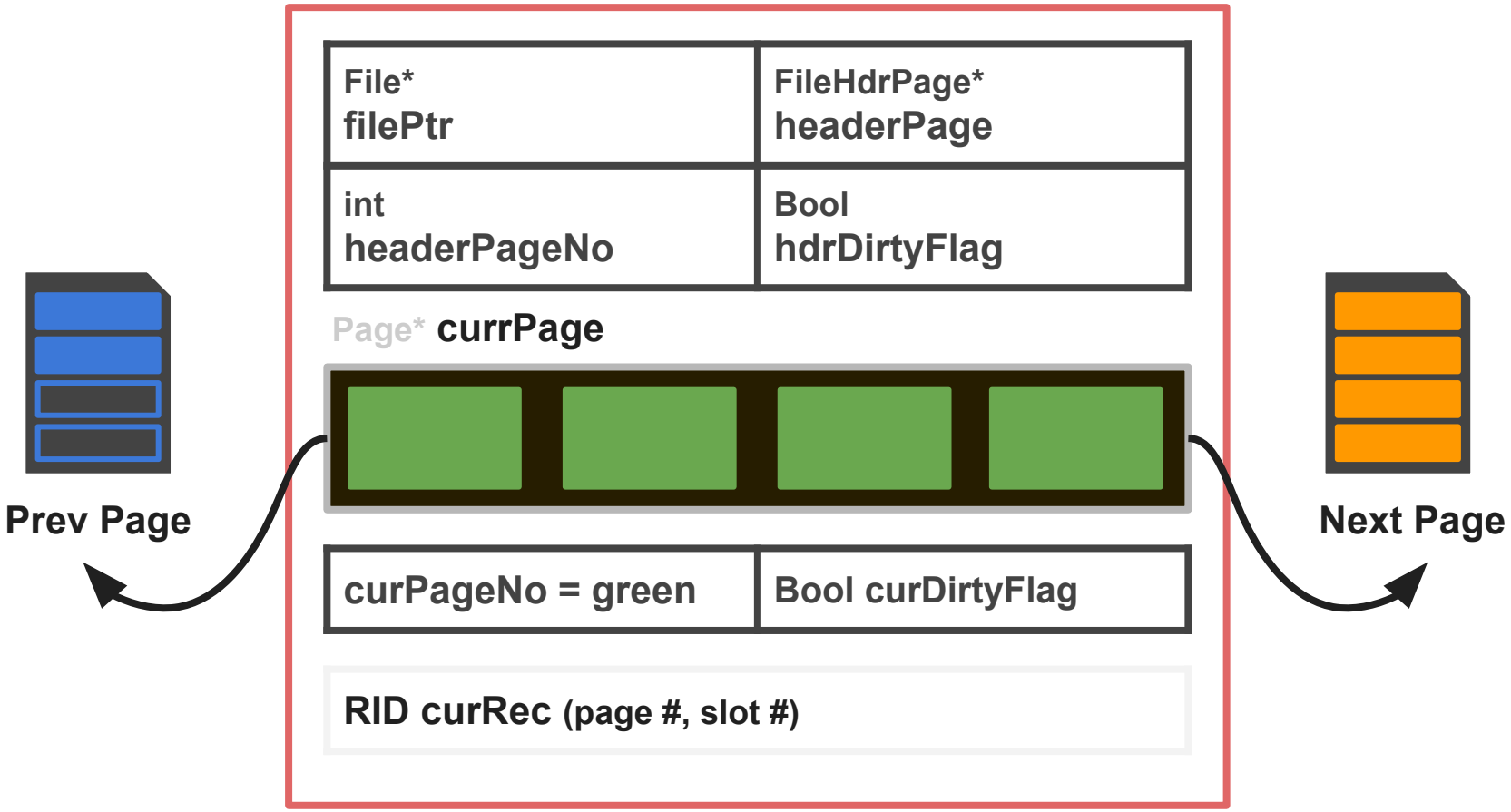


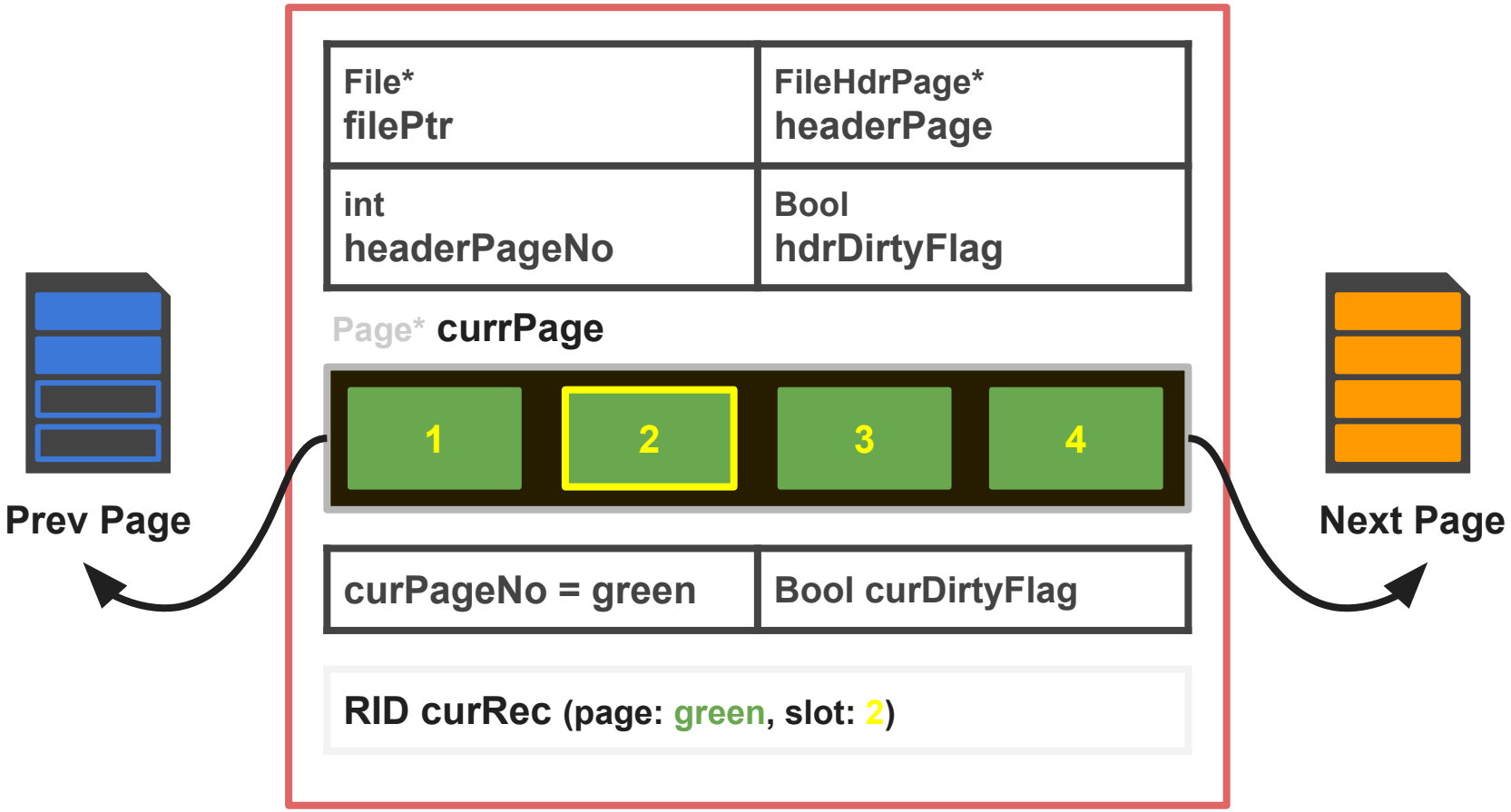
HeapFile Class

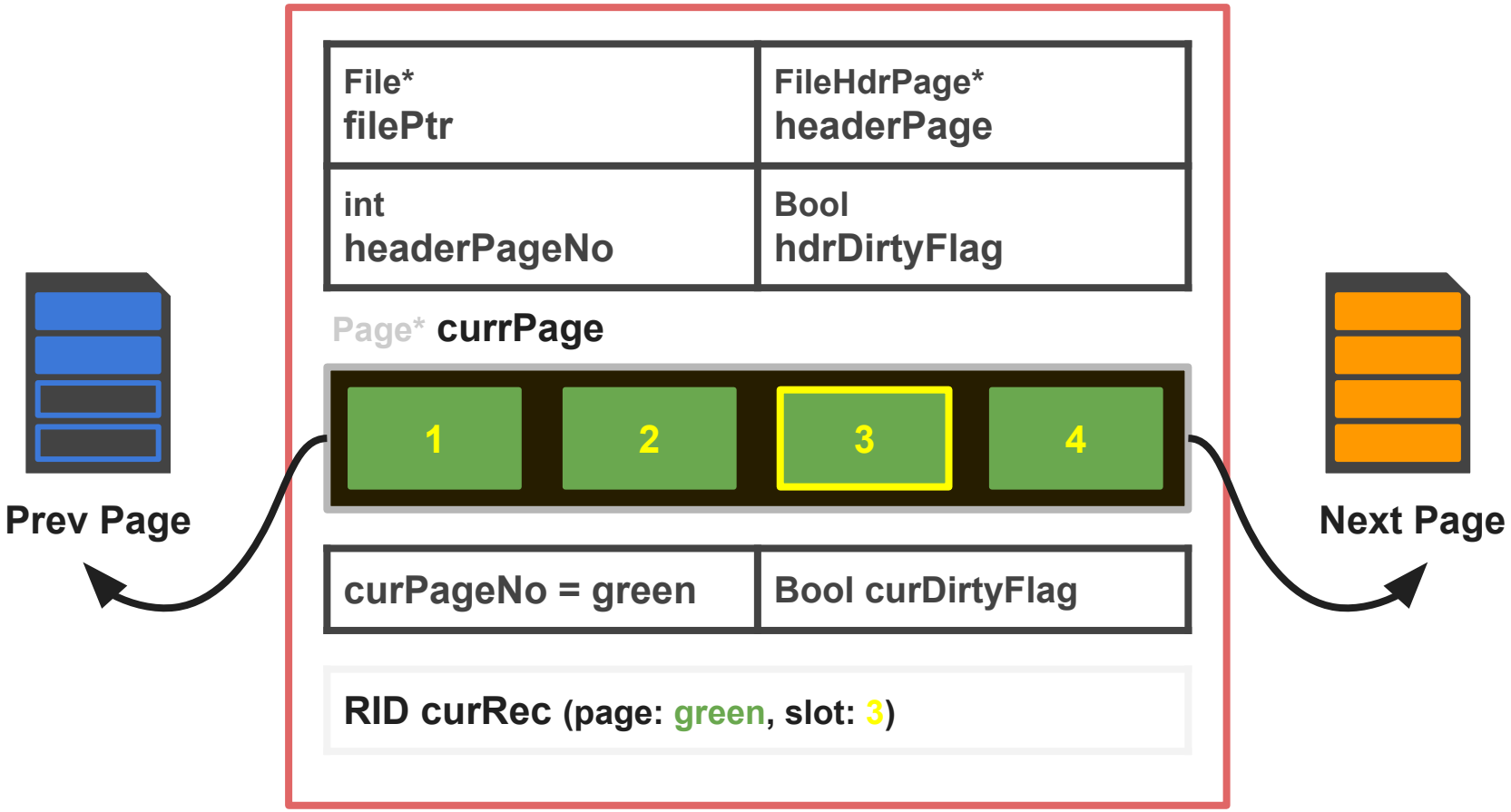


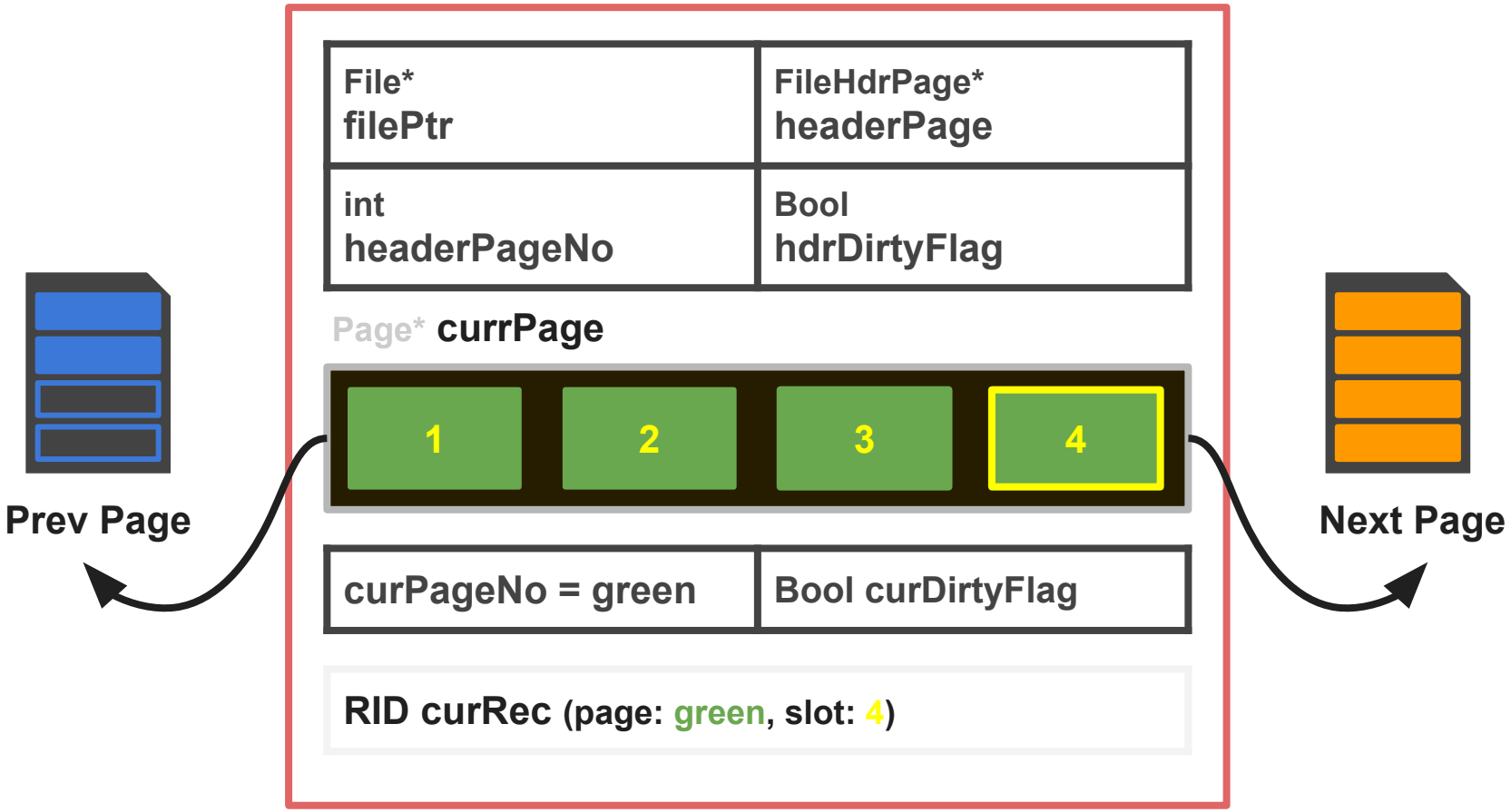

File*
filePtr
FileHdrPage*
headerPage
int
headerPageNo
Bool
hdrDirtyFlag
Page* currPage
curPageNo = blue
curDirtyFlag
RID curRec (page #, slot #)
Prev Page
Next Page

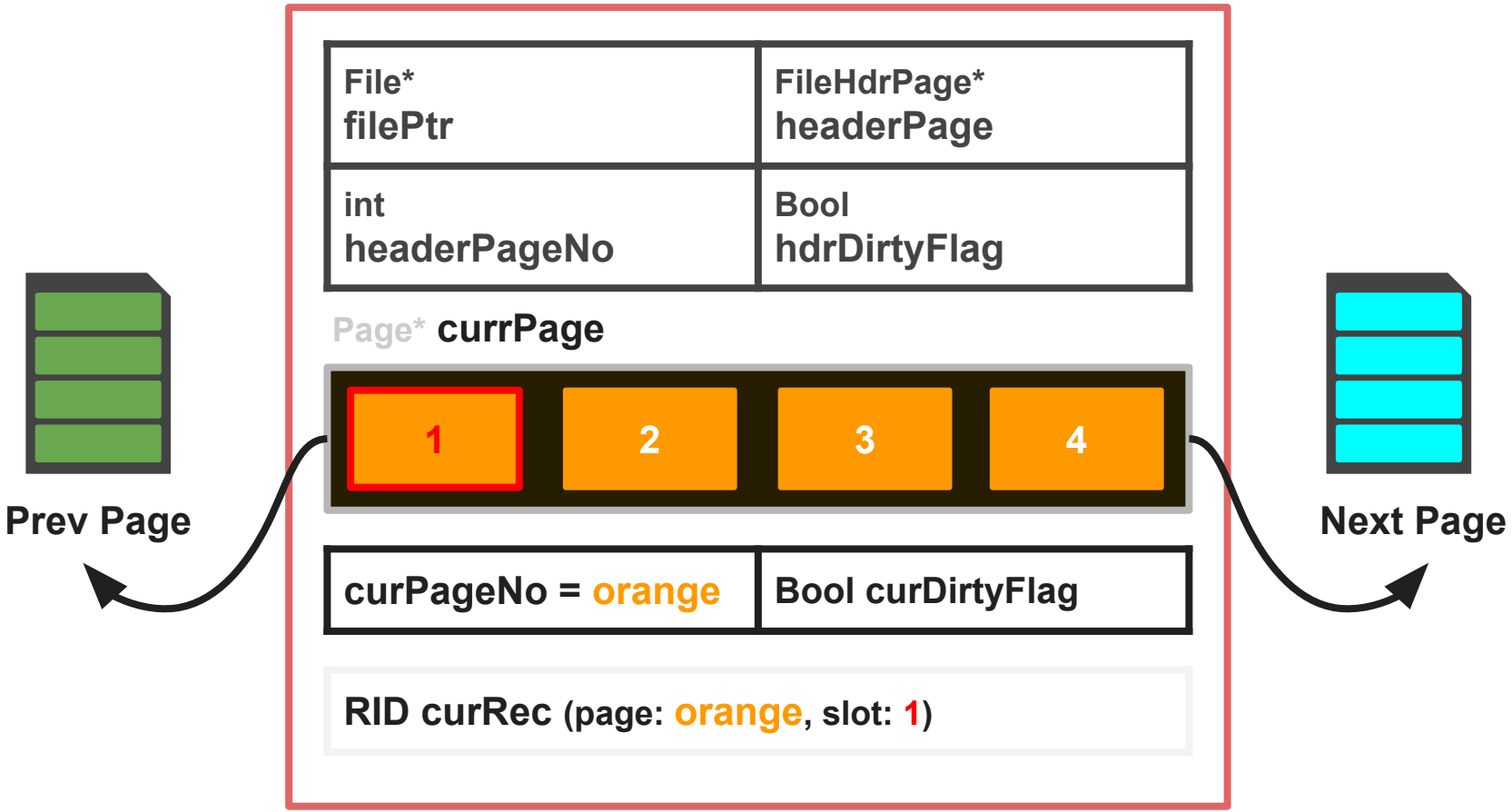

File*
filePtr
FileHdrPage*
headerPage
int
headerPageNo
Bool
hdrDirtyFlag
Page* currPage
Prev Page
Next Page
curPageNo = green
Bool curDirtyFlag
RID curRec (page #, slot #)


File*
filePtr
FileHdrPage*
headerPage
int
headerPageNo
Bool
hdrDirtyFlag
Page* currPage
curPageNo = green
Bool curDirtyFlag
RID curRec (page #, slot #)
Prev Page
Next Page


File*
filePtr
FileHdrPage*
headerPage
int
headerPageNo
Bool
hdrDirtyFlag
Page* currPage
1
2
3
4
curPageNo = green
Bool curDirtyFlag
RID curRec (page: green, slot: 2)
Prev Page
Next Page


File*
filePtr
FileHdrPage*
headerPage
int
headerPageNo
Bool
hdrDirtyFlag
Page* currPage
1
2
3
4
curPageNo = green
Bool curDirtyFlag
RID curRec (page: green, slot: 3)
Prev Page
Next Page


File*
filePtr
FileHdrPage*
headerPage
int
headerPageNo
Bool
hdrDirtyFlag
Page* currPage
1
2
3
4
curPageNo = green
Bool curDirtyFlag
RID curRec (page: green, slot: 4)
Prev Page
Next Page


File*
filePtr
FileHdrPage*
headerPage
int
headerPageNo
Bool
hdrDirtyFlag
Page* currPage
1
2
3
4
curPageNo = orange
Bool curDirtyFlag
RID curRec (page: orange, slot: 1)
Prev Page
Next Page

Methods to implement

1. `createHeapFile("famous_chemists")`
2. `HeapFile()`
3. `scanNext()`
4. `getRecord()`
5. `insertRecord()`

Creating a heap file (just the file, not the Class object)

```
const Status createHeapFile(const string fileName)
```

Creating a heap file (just the file, not the Class object)

```
const Status createHeapFile(const string fileName)
```

1. Check `fileName` doesn't exist

Creating a heap file (just the file, not the Class object)

```
const Status createHeapFile(const string fileName)
```

1. Check `fileName` doesn't exist
2. Create file `db.createFile(fileName)` and open

Creating a heap file (just the file, not the Class object)

```
const Status createHeapFile(const string fileName)
```

1. Check `fileName` doesn't exist
2. Create file `db.createFile(fileName)` and open
3. Allocate a header page using `bufMgr->allocPage`
   a. `FileHdrPage* hdrPage = (FileHdrPage *) newPage;`

Creating a heap file (just the file, not the Class object)

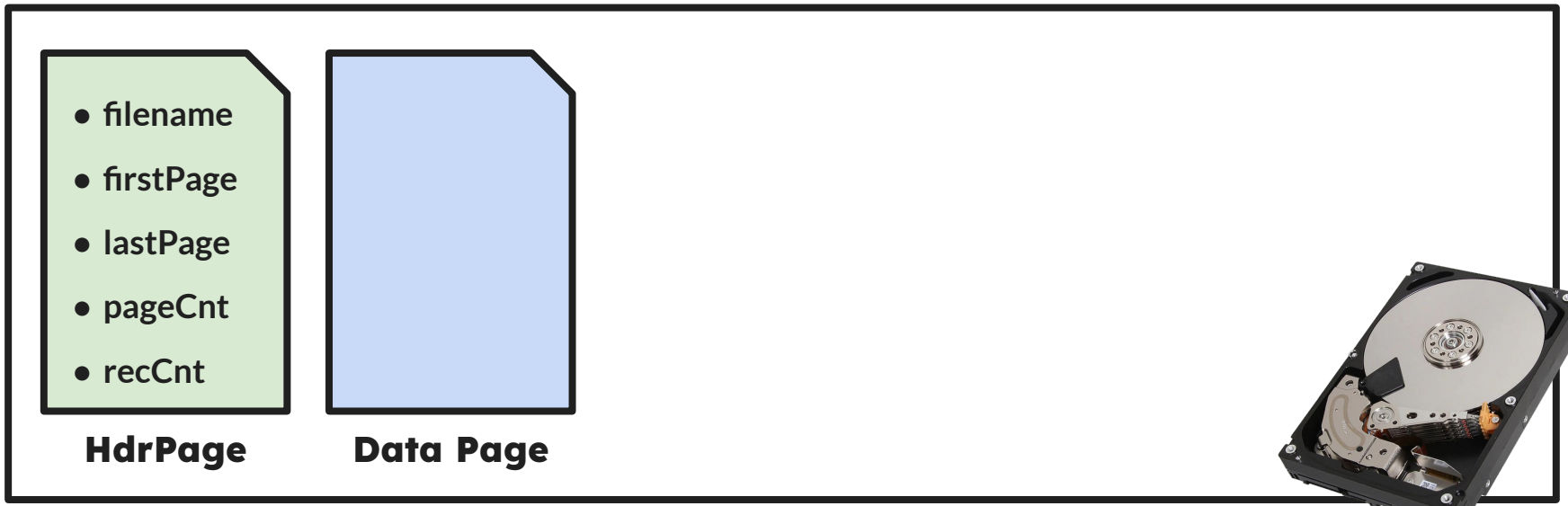
```
const Status createHeapFile(const string fileName)
```

1. Check `fileName` doesn't exist
2. Create file `db.createFile(fileName)` and open
3. Allocate a header page using `bufMgr->allocPage`
   a. `FileHdrPage* hdrPage = (FileHdrPage *) newPage;`
4. Allocate an empty, data page and call `newPage->init(newPageNo);`
5. Unpin pages
6. Flush and close file

Creating a heap file

```
createHeapFile("famous_chemists")
```

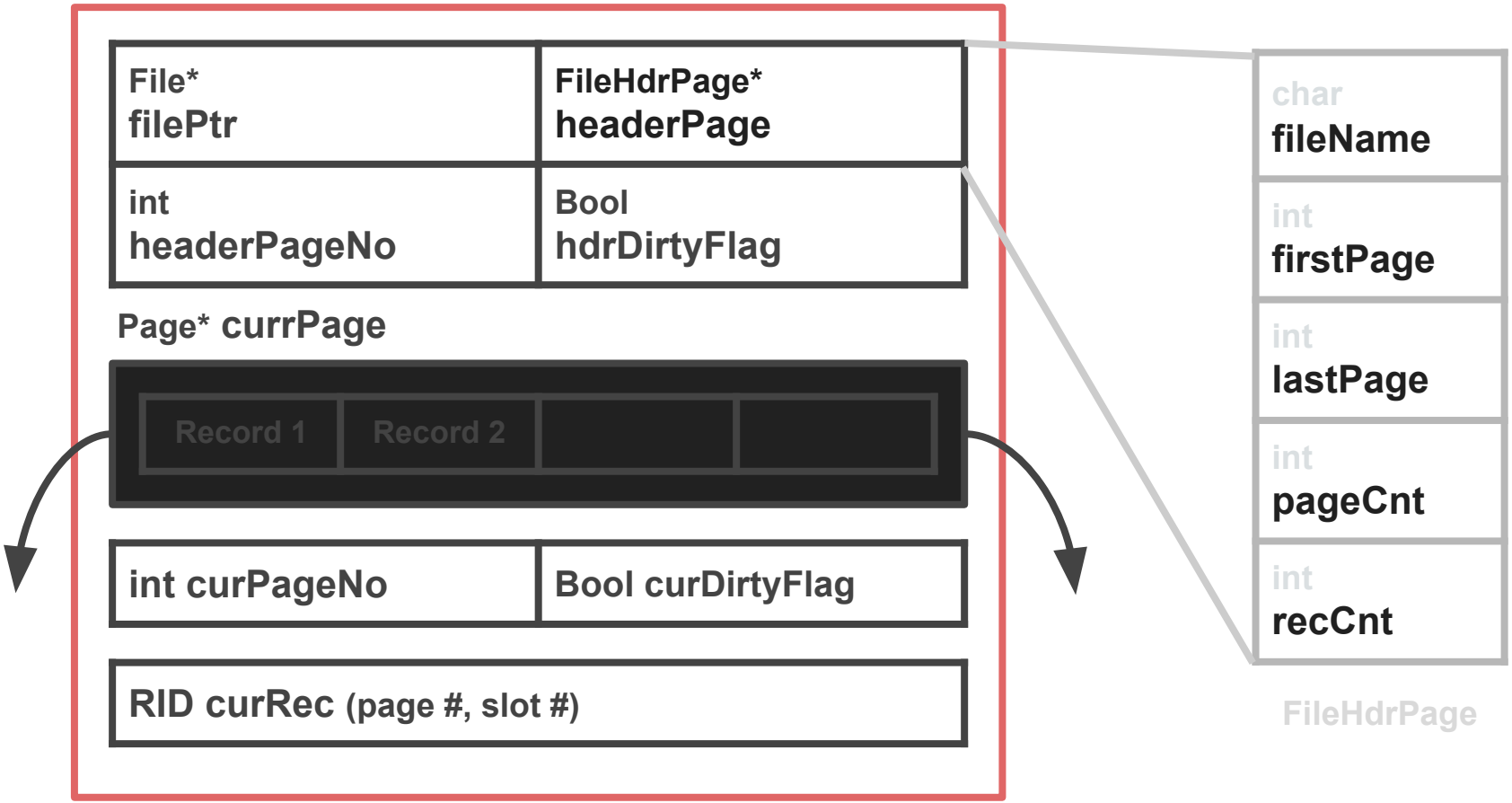
famous_chemists



Constructing a HeapFile object (from a heap file on disk)

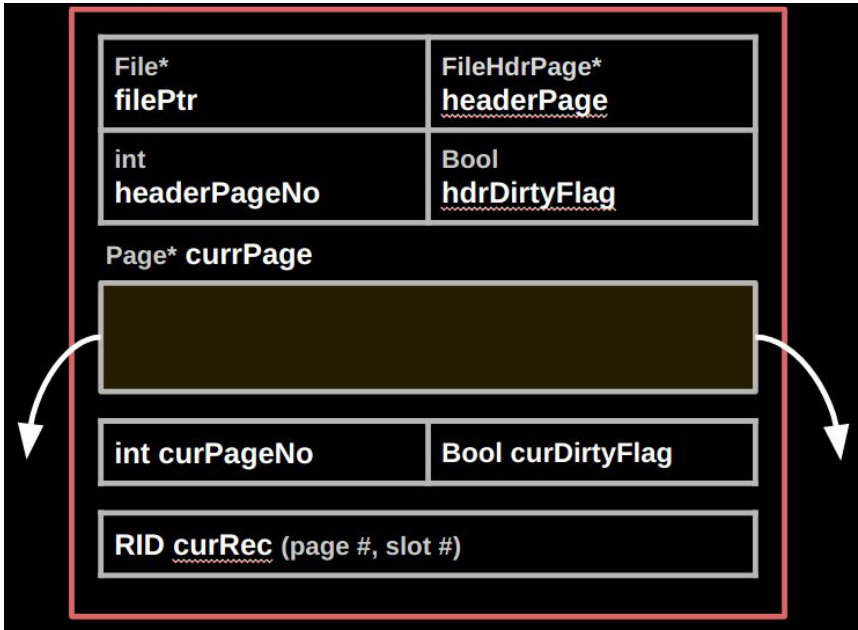
```
HeapFile::HeapFile(const string & fileName, Status& returnStatus)
```

HeapFile


File*
filePtr
FileHdrPage*
headerPage
int
headerPageNo
Bool
hdrDirtyFlag
Page* currPage
Record 1
Record 2
int curPageNo
Bool curDirtyFlag
RID curRec (page #, slot #)
char
fileName
int
firstPage
int
lastPage
int
pageCnt
int
recCnt
FileHdrPage

Constructing a HeapFile object (from a heap file on disk)

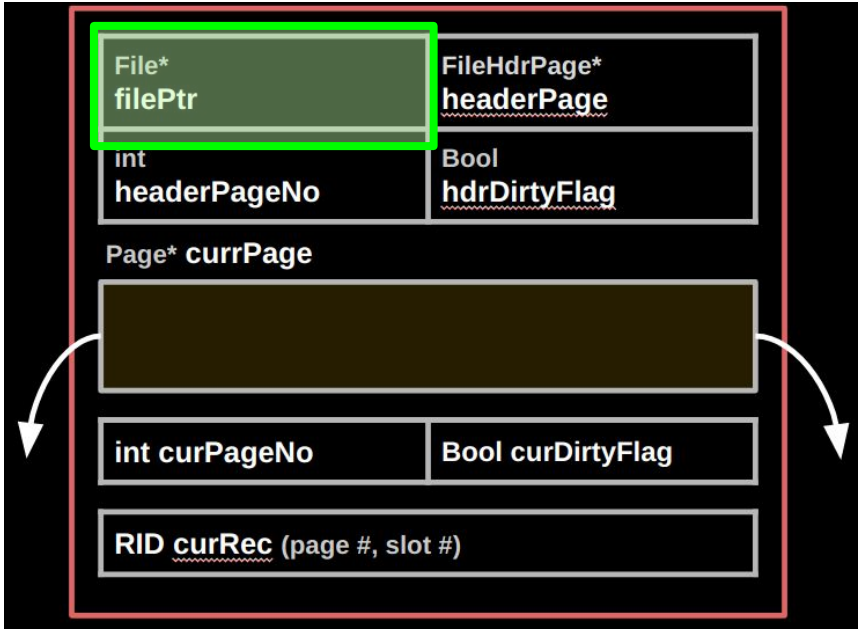
```
HeapFile::HeapFile(const string & fileName, Status& returnStatus)
```



Constructing a HeapFile object (from a heap file on disk)

```
HeapFile::HeapFile(const string & fileName, Status& returnStatus)
```

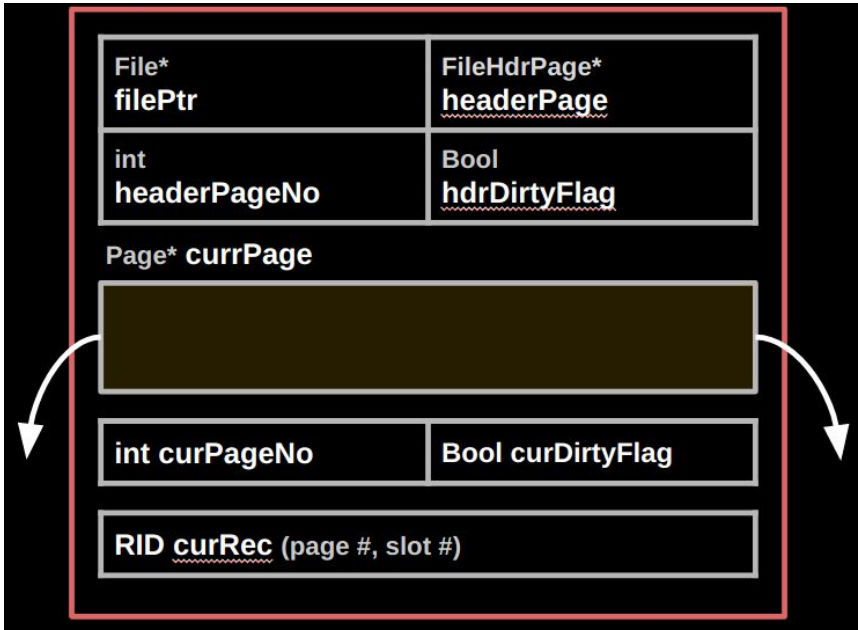
1. Check `fileName` exists and open



Constructing a HeapFile object (from a heap file on disk)

```
HeapFile::HeapFile(const string & fileName, Status& returnStatus)
```

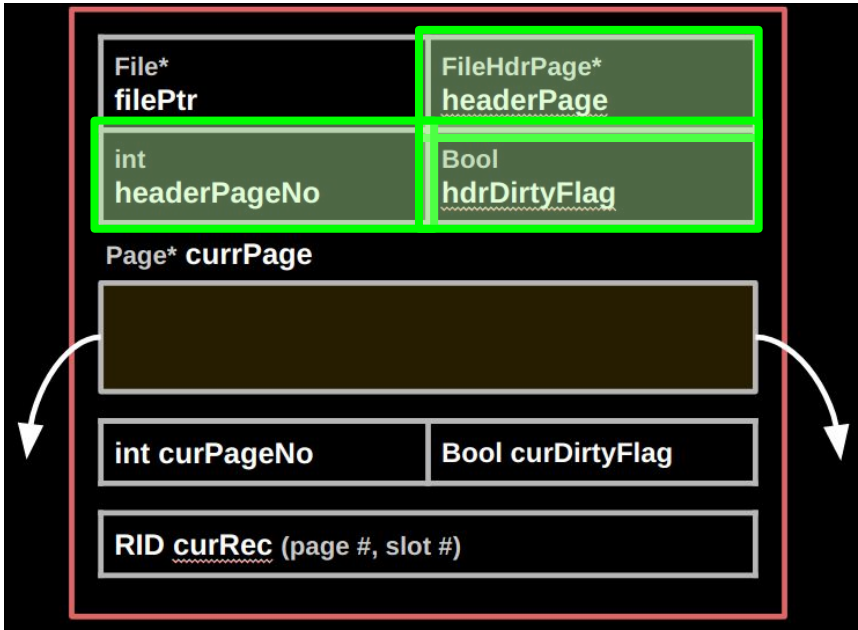
1. Check `fileName` exists and open
2. Assign the header-page with `filePtr->getFirstPage` and ???



Constructing a HeapFile object (from a heap file on disk)

```
HeapFile::HeapFile(const string & fileName, Status& returnStatus)
```

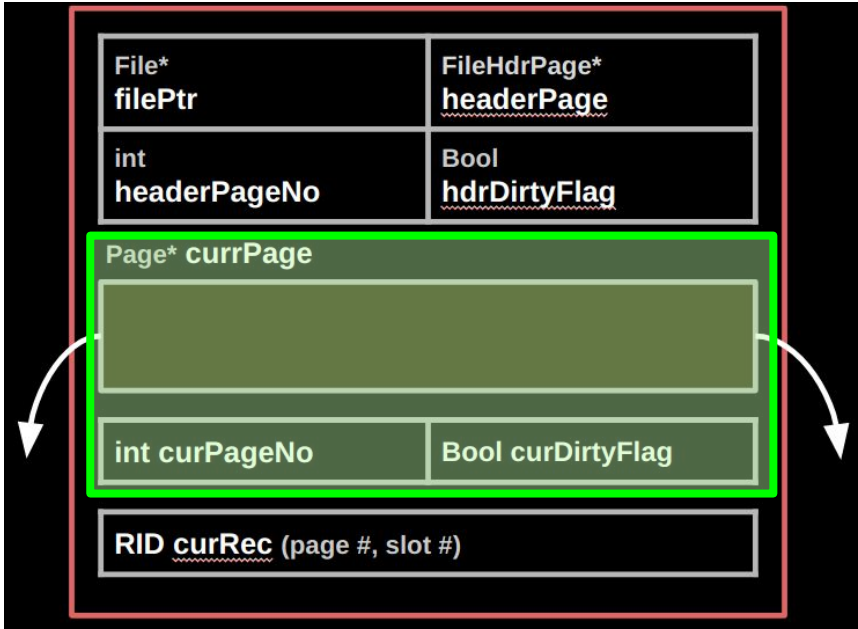
1. Check `fileName` exists and open
2. Assign the header-page with `filePtr->getFirstPage` and `bufMgr->readPage`
   a. `headerPage = (FileHdrPage *) pagePtr;`



Constructing a HeapFile object (from a heap file on disk)

```
HeapFile::HeapFile(const string & fileName, Status& returnStatus)
```

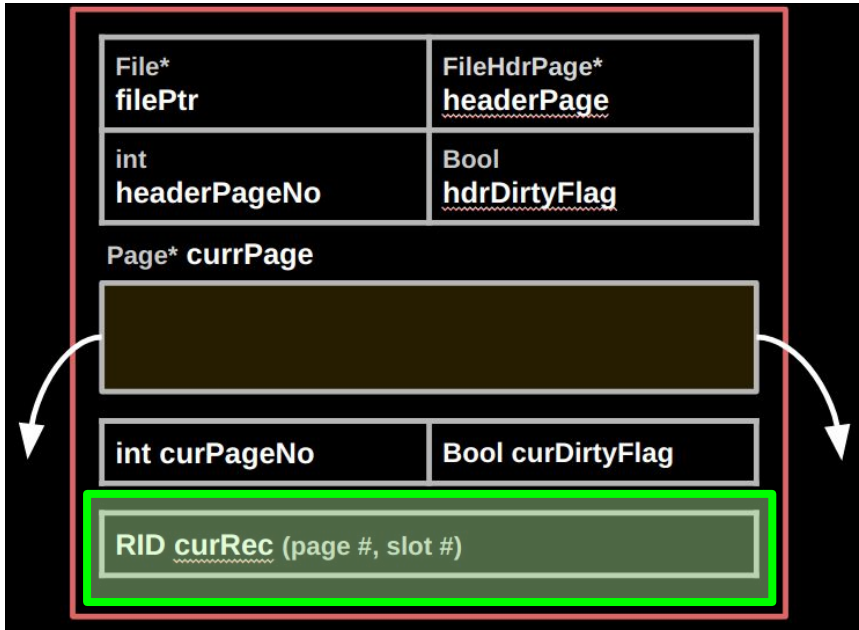
1. Check `fileName` exists and open
2. Assign the header-page with `filePtr->getFirstPage` and `bufMgr->readPage`
   a. `headerPage = (FileHdrPage *) pagePtr;`
3. Read the first data page into *curPage* and *curPageNo*



Constructing a HeapFile object (from a heap file on disk)

```
HeapFile::HeapFile(const string & fileName, Status& returnStatus)
```

1. Check `fileName` exists and open
2. Assign the header-page with `filePtr->getFirstPage` and `bufMgr->readPage`
   a. `headerPage = (FileHdrPage *) pagePtr;`
3. Read the first data page into *curPage* and *curPageNo*
4. `curRec = NULLRID;`



Get a Record (by RID)

```
const Status HeapFile::getRecord(const RID & rid, Record & rec)
```

Get a Record (by RID)

```
const Status HeapFile::getRecord(const RID & rid, Record & rec)
```

1. If on correct page: `curPageNo == rid.pageNo`:
   a. Call *getRecord* on current page (gets record by slot number)
   b. ???

Get a Record (by RID)

```
const Status HeapFile::getRecord(const RID & rid, Record & rec)
```

1. If on correct page: `curPageNo == rid.pageNo`:
   a. Call *getRecord* on current page (gets record by slot number)
   b. Update HeapFile object
   c. Done!

Get a Record (by RID)

```
const Status HeapFile::getRecord(const RID & rid, Record & rec)
```

1. If on correct page: `curPageNo == rid.pageNo`:
   a. Call *getRecord* on current page (gets record by slot number)
   b. Update HeapFile object
   c. Done!
2. Unpin current

Get a Record (by RID)

```
const Status HeapFile::getRecord(const RID & rid, Record & rec)
```

1. If on correct page: `curPageNo == rid.pageNo`:
   a. Call *getRecord* on current page (gets record by slot number)
   b. Update HeapFile object
   c. Done!
2. Unpin current
3. Update HeapFile object
   a. *curPageNo,*
   b. *curDirtyFlag*
   c. *curRec*

Get a Record (by RID)

```
const Status HeapFile::getRecord(const RID & rid, Record & rec)
```

1. If on correct page: `curPageNo == rid.pageNo`:
   a. Call *getRecord* on current page (gets record by slot number)
   b. Update HeapFile object
   c. Done!
2. Unpin current
3. Update HeapFile object
   a. *curPageNo,*
   b. *curDirtyFlag*
   c. *curRec*
4. Read page using curPageNo, then call *getRecord*

Scan Next

```
const Status HeapFileScan::scanNext(RID& outRid)
```

Scan Next

```
const Status HeapFileScan::scanNext(RID& outRid)
```

Purpose:

SELECT * FROM famous_chemists WHERE alias = "Heisenberg"

Name	Alias	Address	Affiliation	DOB
Walter White	Heisenberg	Albuquerque	Los Pollos	Sept 7, 1958

What's missing?

Scan Next

```
const Status HeapFileScan::scanNext(RID& outRid)
```

Purpose:

SELECT * FROM famous_chemists WHERE alias = "Heisenberg"

Name	Alias	Address	Affiliation	DOB
Walter White	Heisenberg	Albuquerque	Los Pollos	Sept 7, 1958

"*filter*" in HeapFile set using *startScan*.

Scan Next

```
const Status HeapFileScan::scanNext(RID& outRid)
```

1. **Valid** current Page
   a. Iterate over pages and records (use *nextPage* and *nextRecord*)
   b. `if (matchRec(rec))` return curRec as outRid
   c. If reached end of file, return EOF

Scan Next

```
const Status HeapFileScan::scanNext(RID& outRid)
```

1. **Valid** current Page
   a. Iterate over pages and records (use *nextPage* and *nextRecord*)
   b. `if (matchRec(rec))` return curRec as outRid
   c. If reached end of file, return FILEEOF

   Before moving to next page, do cleanup

   d. Unpin previous page
   e. Update all relevant HeapFile object fields
   f. Start from first record of new page (*curPage->firstRecord*)

`ENDOFPAGE`

`NORECORDS`

See Page.C

Scan Next

```
const Status HeapFileScan::scanNext(RID& outRid)
```

2. **Invalid** current Page
    a. Start from first page (*headerPage->firstPage*)
    b. Start from first record (*curPage->firstRecord*)
    c. Do steps from previous slide

Insert a Record

```
const Status InsertFileScan::insertRecord(const Record & rec, RID& outRid)
```

Insert a Record

```
const Status InsertFileScan::insertRecord(const Record & rec, RID& outRid)
```

How to find a home for the new record?

Insert a Record

```
const Status InsertFileScan::insertRecord(const Record & rec, RID& outRid)
```

1. Valid current Page
    a. Call *insertRecord* on *curPage*

Insert a Record

```
const Status InsertFileScan::insertRecord(const Record & rec, RID& outRid)
```

1. Valid current Page
    a. Call *insertRecord* on *curPage*
        i. If OK, update header page fields and update *curRec*

Insert a Record

```
const Status InsertFileScan::insertRecord(const Record & rec, RID& outRid)
```

1. Valid current Page
    a. Call *insertRecord* on *curPage*
        i. If OK, update header page fields and update *curRec*
        ii. Else, no-space - insert in a new page at the end
            1. Allocate new page and initialize it
            2. Unpin current page and update to new page
            3. Call *insertRecord* on this new Page
            4. Update header page fields

Insert a Record

```
const Status InsertFileScan::insertRecord(const Record & rec, RID& outRid)
```

2. **Invalid** current Page
   a. Read the last page (Use *hdrPage->lastPage*)
   b. Set last page as current page
   c. Do steps from previous slide

Suggestion

- It will be a good idea to read an overview of the Minirel Page Class
  - https://pages.cs.wisc.edu/~anhai/courses/564/minirel-project/pageclass.htm
- Some common errors
  - Infinite Loop
    - ScanNext() stops in a records satisfying the predicate and always return that records
  - Missing bookkeeping
    - Forget to update HeaderPage (page count, rec count)
    - Forget to update the Heapfile object (cur page, cur rec, dirty bits)
    - Link the data pages incorrectly
      - `newPage->setNextPage(-1); the next page for last page`

Logistics

- You should only edit file heapfile.C.
- Submission: heapfile.C & members.txt
- Testing: We will add your heapfile.C to our code and compile. Make sure that your code compiles and passes all the tests in testfile.c ON CS MACHINES, as we will compile your code using CS machines.
  - How to connect to csl through vscode: https://shawnzhong.com/2019/10/16/remote-ssh-to-cs-lab-with-vscode/
- **No Extensions!!!**

Questions