Lab 1 bootcamp

Lab1: What is GoDB?

A basic database system implemented in Go

- A simple storage layer, based on Heap Files (Lab 1)
- A buffer pool for caching pages and implementation page-level locking for transactions (Labs 1-3)
- A set of operators (Labs 1 & 2): Scan, Filter, Join, Aggregate, Order By, Project ...
- A SQL parser (Lab 2), which we implement for you
- Simple transactions (Lab 3)
- Previous years we included recovery, B+Trees, and query optimization, but have reduced the labs because this is our second year in Go.
  - Students in 6.5831 may implement one of these for their final project

What is GoDB Missing?

- Focus is on a simple architecture rather than a complete or high-performance implementation
- Only supports fixed length records with strings and ints
- Only supports sequential scan access methods
- No NULLs
- Uses a simple iterator method, so not super efficient

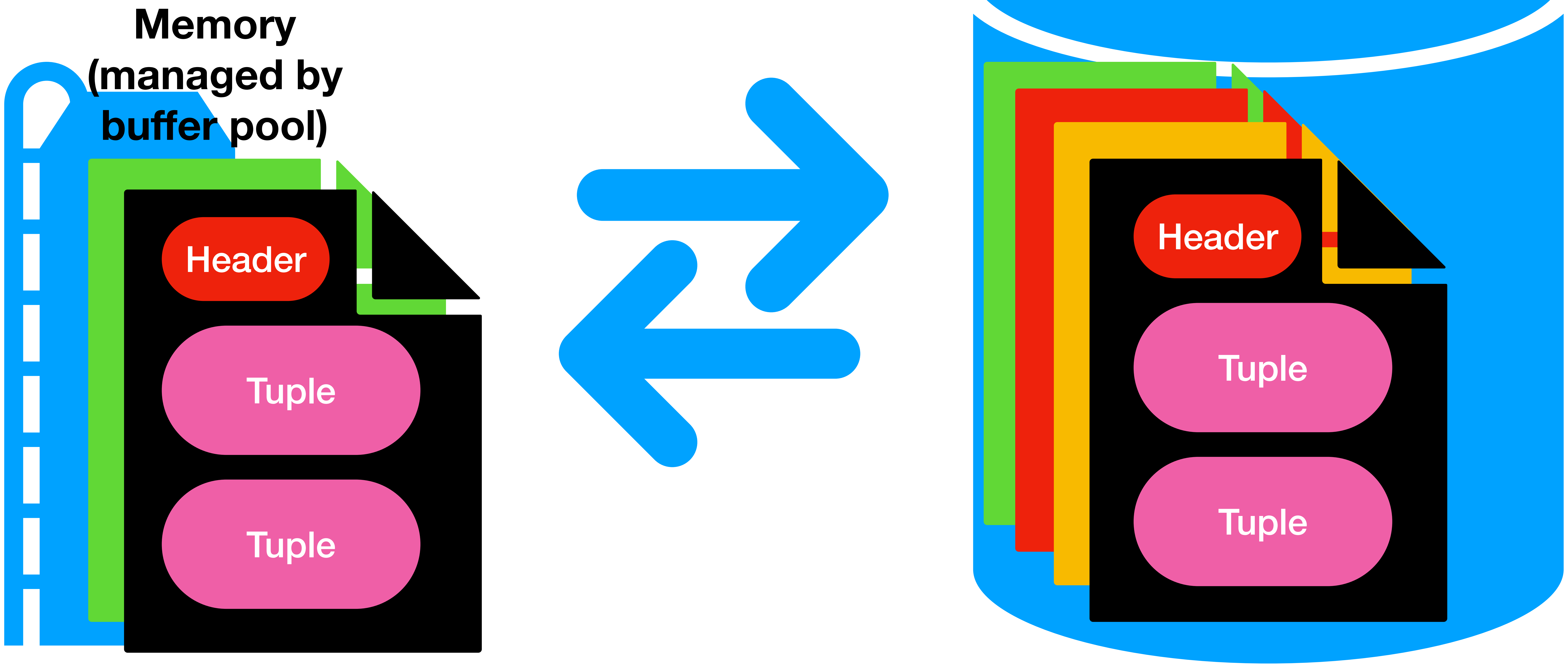
GoDB Storage Layout

- Each table is stored in one file on disk, called a *heap file*
  - Heap files are an unordered collections of records
  - Only way to access records from a heap file is to scan from beginning to end: “Sequential scan” via an iterator
- Each heap file consists of a number of fixed size heap pages
- Each heap page contains a number of fixed size tuples

- Methods in heap_file.go are used to access the contents of the heap file

Goal: Storage

- Physical Layout **Design**



Tuples and Tuple Descriptors

- In a given heap file, each tuple has the same layout
- Layout is specified by a TupleDesc object, which specifies the field names and types in the tuple

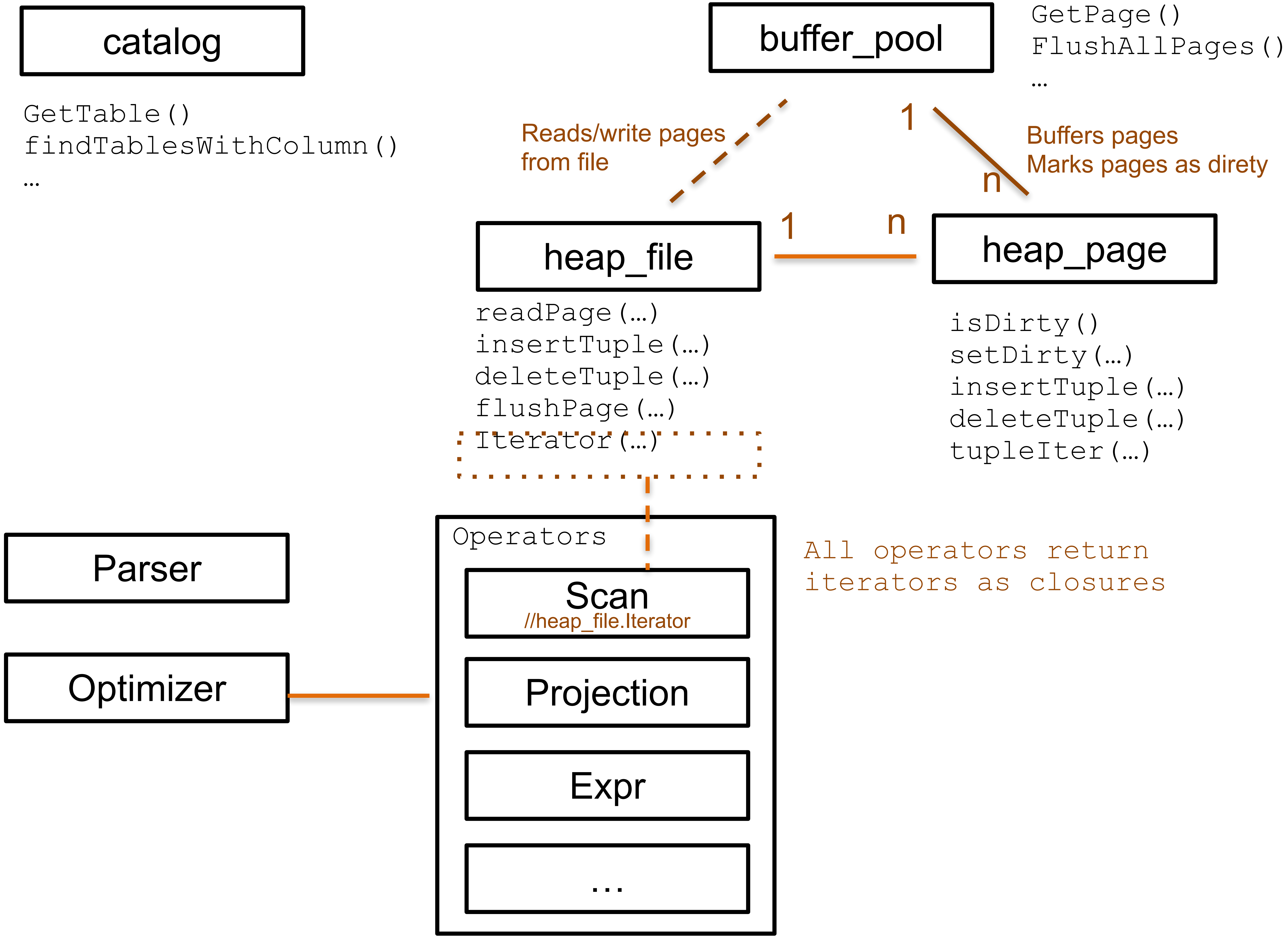
```
// FieldType is the type of a field in a tuple, e.g., its name, table, and [godb.DBType].
// TableQualifier may or may not be an empty string, depending on whether the table
// was specified in the query
type FieldType struct {
    Fname string
    TableQualifier string
    Ftype DBType
}

// TupleDesc is "type" of the tuple, e.g., the field names and types
type TupleDesc struct {
    Fields []FieldType
}
```

Tuples and Tuple Descriptors (cont.)

- Tuple objects contain the values of each record in Fields
- Field is an **interface**, implemented by IntField and StringField
- All ints are 64 bits; all string are StringLength characters, padded with zeros

```
// Tuple represents the contents of a tuple read from a database
// It includes the tuple descriptor, and the value of the fields
type Tuple struct {
    Desc TupleDesc
    Fields []DBValue
    Rid recordID //used to track the page and position this page was read from
}
```

Module Diagram



catalog

GetTable()
findTablesWithColumn()
…

buffer_pool

GetPage()
FlushAllPages()
…

Reads/write pages
from file

1

n

Buffers pages
Marks pages as direty

heap_file

readPage(…)
insertTuple(…)
deleteTuple(…)
flushPage(…)
Iterator(…)

1

n

heap_page

isDirty()
setDirty(…)
insertTuple(…)
deleteTuple(…)
tupleIter(…)

Parser

Optimizer

Operators

Scan
//heap_file.Iterator

Projection

Expr

…

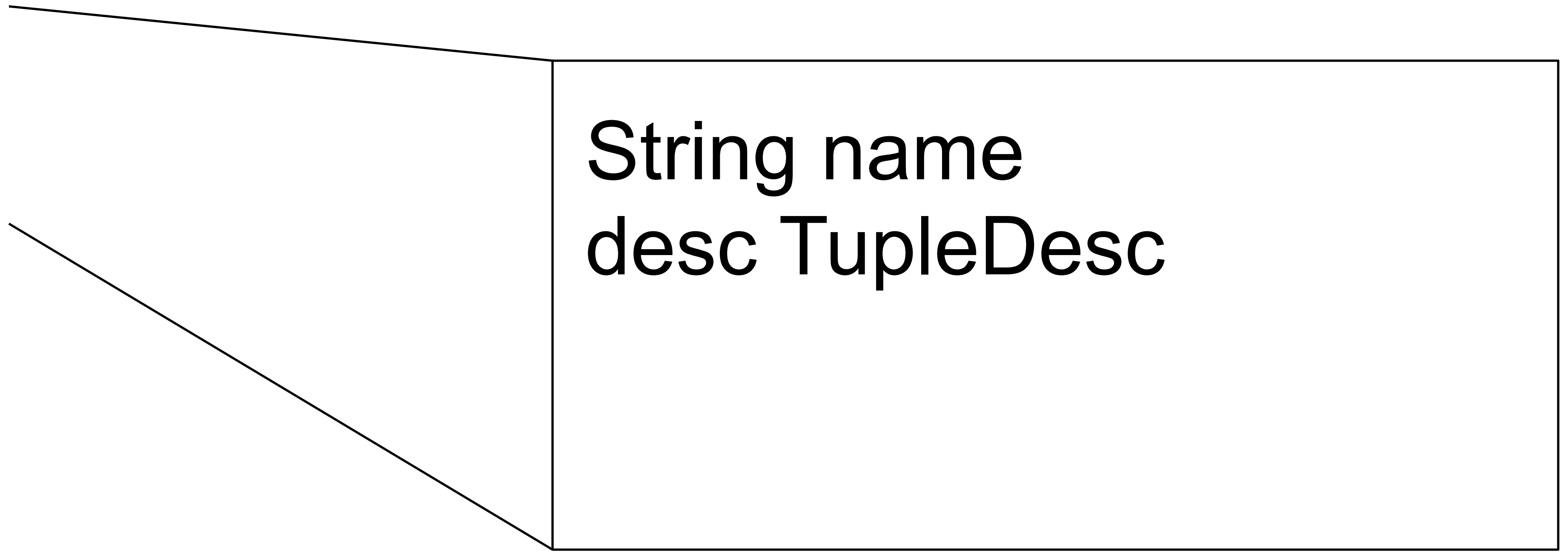
All operators return
iterators as closures

Catalog

Catalog:

Tablename	Table
t1	Table1
t2	Table2

Table:

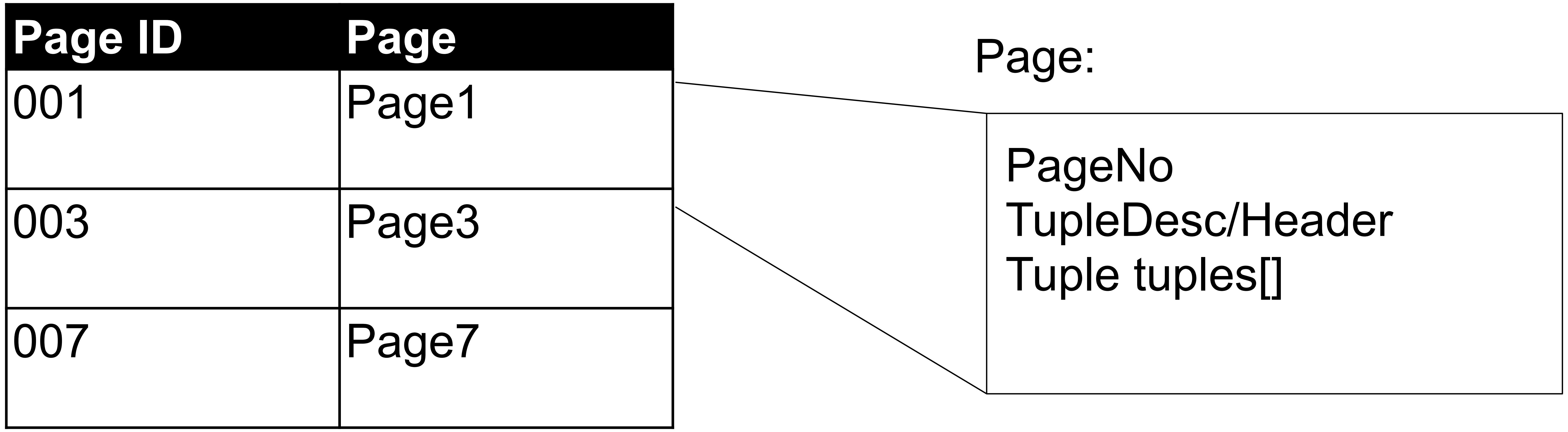


String name
desc TupleDesc

=> Stores a list of all tables in the database

buffer_pool

Buffer Pool:



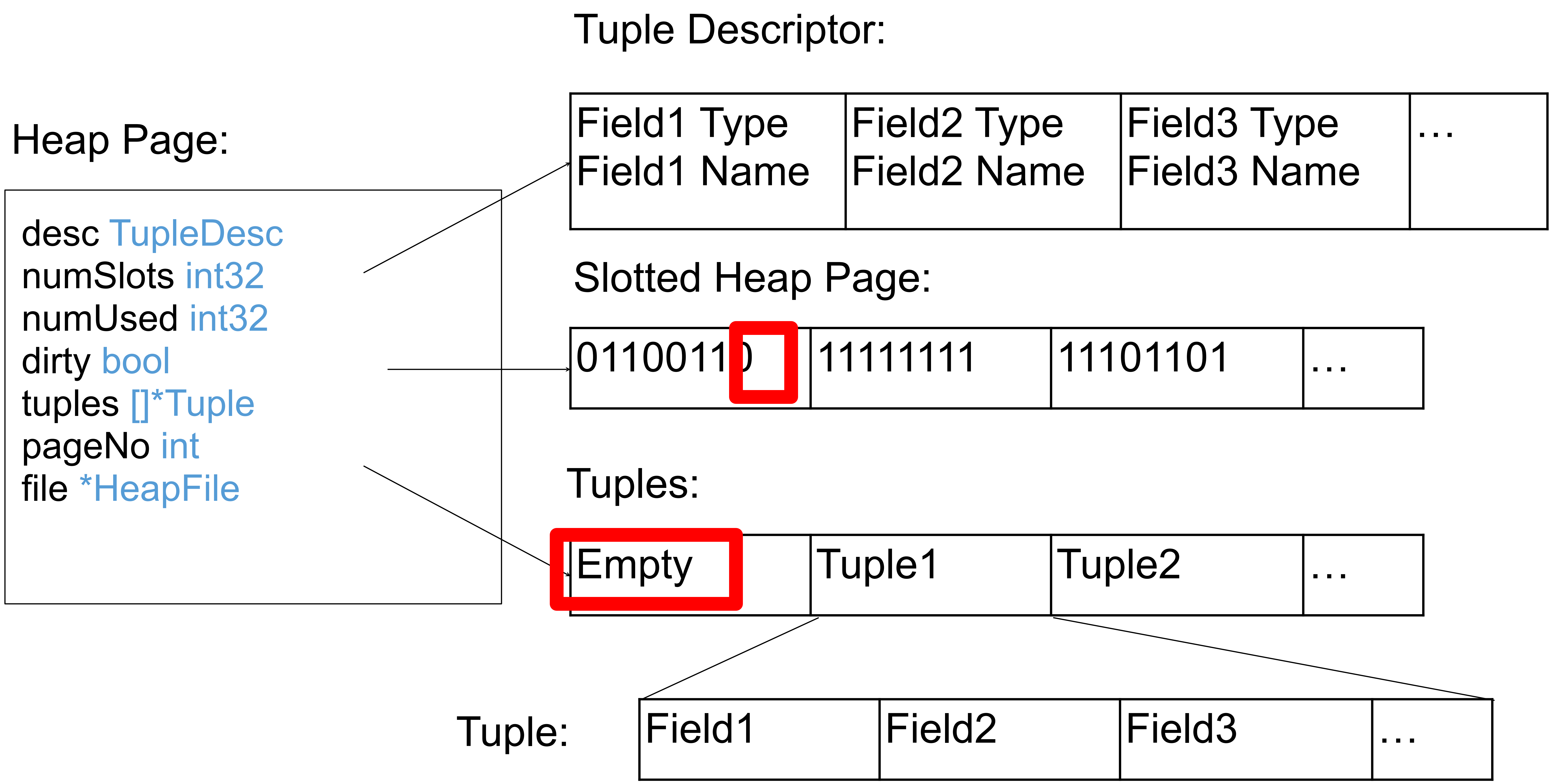
Page ID	Page
001	Page1
003	Page3
007	Page7

Page:

PageNo
TupleDesc/Header
Tuple tuples[]

=> Caches recently accessed database pages in memory
=> Manages read/write locks

heap_page



Heap Page:

desc TupleDesc
numSlots int32
numUsed int32
dirty bool
tuples []*Tuple
pageNo int
file *HeapFile

Tuple Descriptor:

Field1 Type Field1 Name	Field2 Type Field2 Name	Field3 Type Field3 Name	…

Slotted Heap Page:

01100110	11111111	11101101	…

Tuples:

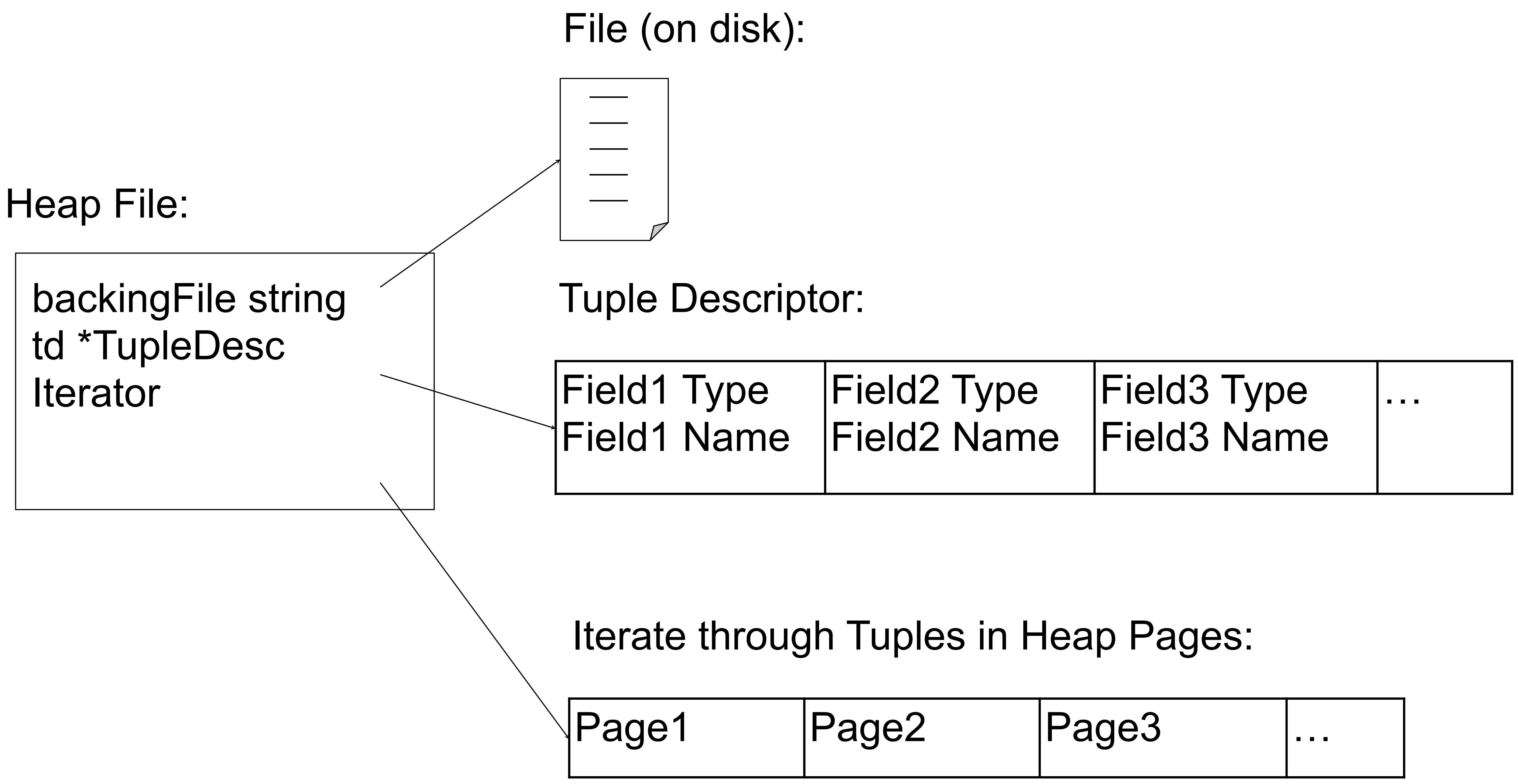
Empty	Tuple1	Tuple2	…

Tuple:

Field1	Field2	Field3	…

Fields and Tuples are Fixed Width!

HeapFile
(Implements DbFile)



File (on disk):

Heap File:

backingFile string
td *TupleDesc
Iterator

Tuple Descriptor:

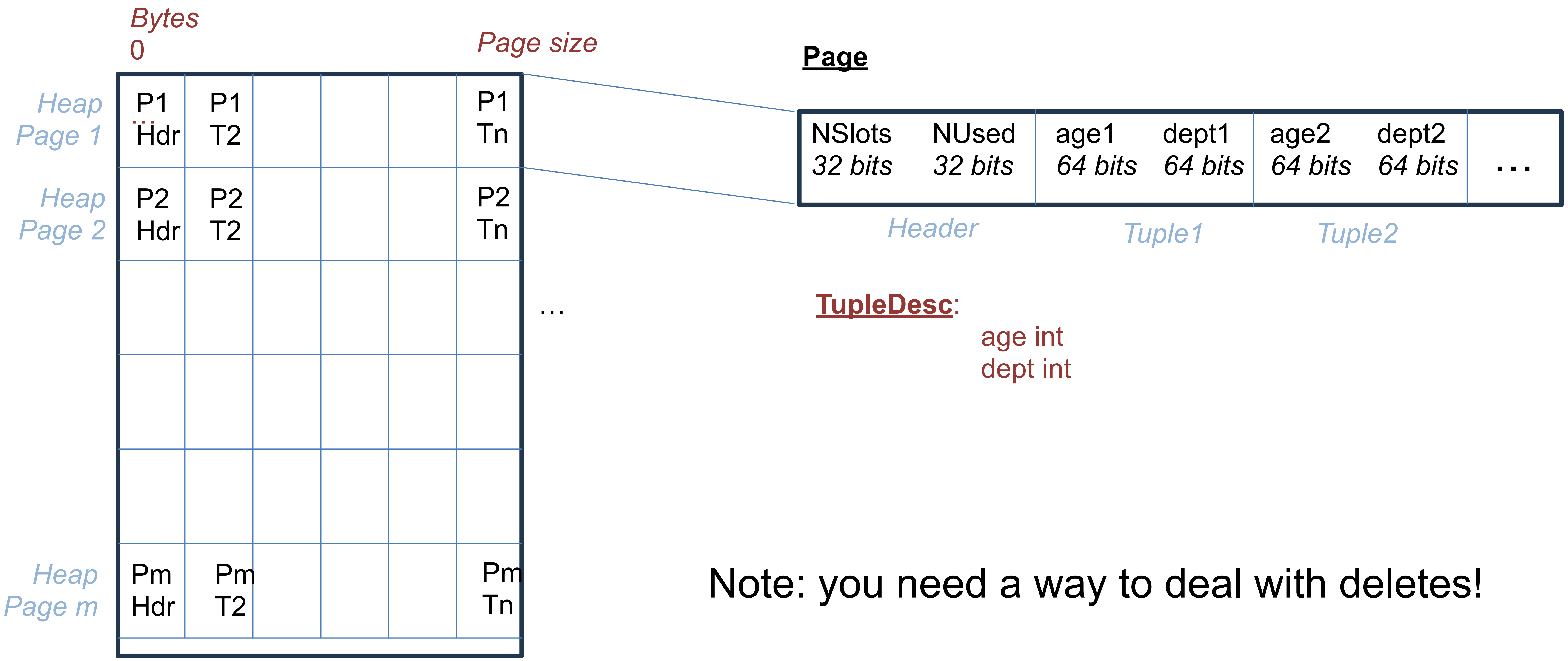
Field1 Type Field1 Name	Field2 Type Field2 Name	Field3 Type Field3 Name	…

Iterate through Tuples in Heap Pages:

Page1	Page2	Page3	…

Storage Layout Diagram

HeapFile (table1)



Bytes 0

Page size

Heap Page 1	P1 Hdr	P1 T2			P1 Tn
Heap Page 2	P2 Hdr	P2 T2			P2 Tn
…					
Heap Page m	Pm Hdr	Pm T2			Pm Tn

Page

Header		Tuple1		Tuple2		
NSlots *32 bits*	NUsed *32 bits*	age1 *64 bits*	dept1 *64 bits*	age2 *64 bits*	dept2 *64 bits*	…

TupleDesc:

age int
dept int

Note: you need a way to deal with deletes!

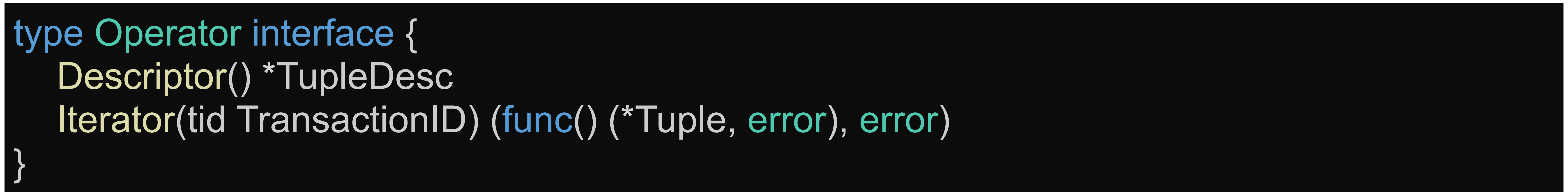
Buffer Pool

- Buffer pool is an in-memory cache of pages
- Allows GoDB to control how much memory is used and support tables larger than memory
- For transactions, will be responsible for implementing page-level locking and two-phase commit (not until lab 3)

- All iterators and operators should use the buffer pool GetPage method to access pages from heap files
- Only the heap file readPage method directly reads data from disk

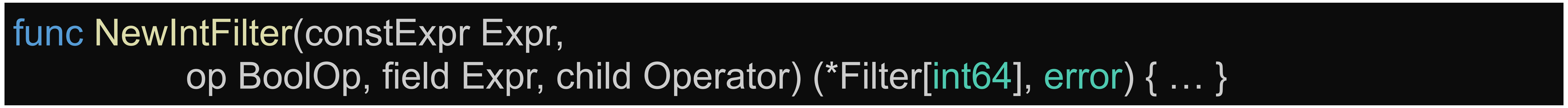
Iterators

- Each database operator (filter, project, join, etc) implements an *Iterator*



```
type Operator interface {
    Descriptor() *TupleDesc
    Iterator(tid TransactionID) (func() (*Tuple, error), error)
}
```

- Iterator() returns a function that iterates through the operator's records
- Most operators take a child operator as a part of their constructor



```
func NewIntFilter(constExpr Expr,
                op BoolOp, field Expr, child Operator) (*Filter[int64], error) { … }
```

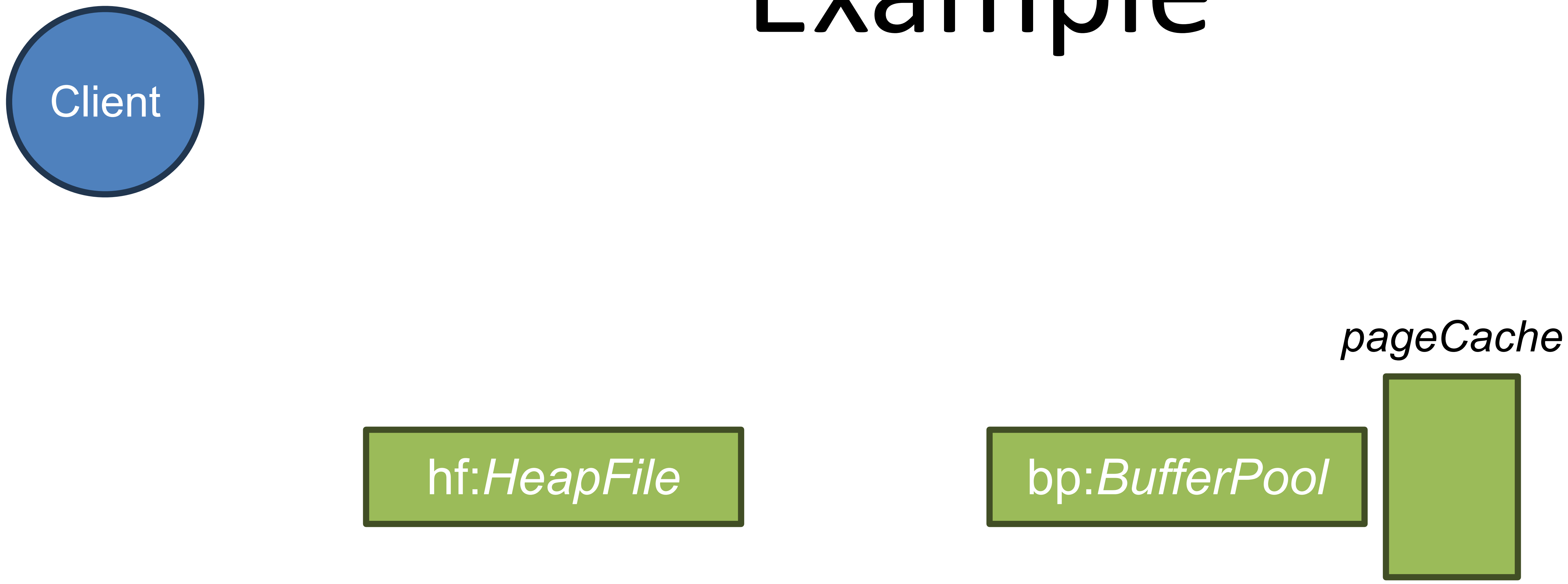
- Heap file Iterator iterates through pages on disk; other operators iterate through their child tuples
  - E.g., filter iterates through child tuples, applied the filter to them, and returns satisfying tuples

Iterator Implementation

- Returns a function that when called returns the next tuple
- Needs to keep state of where it was in its child

```
func (f *Filter[T]) Iterator(tid TransactionID) (func() (*Tuple, error), error) {

    childIter, _ := f.child.Iterator(tid) //childIter is current iterator state
    …
    return func() (*Tuple, error) {
        for {
                // get child tuple from childIter
                // get tuple fields (e.g., using EvalExpr)
                // apply predicate
                // if matches, return tuple
                // else go onto next tuple
        }, _
    }
```

Example

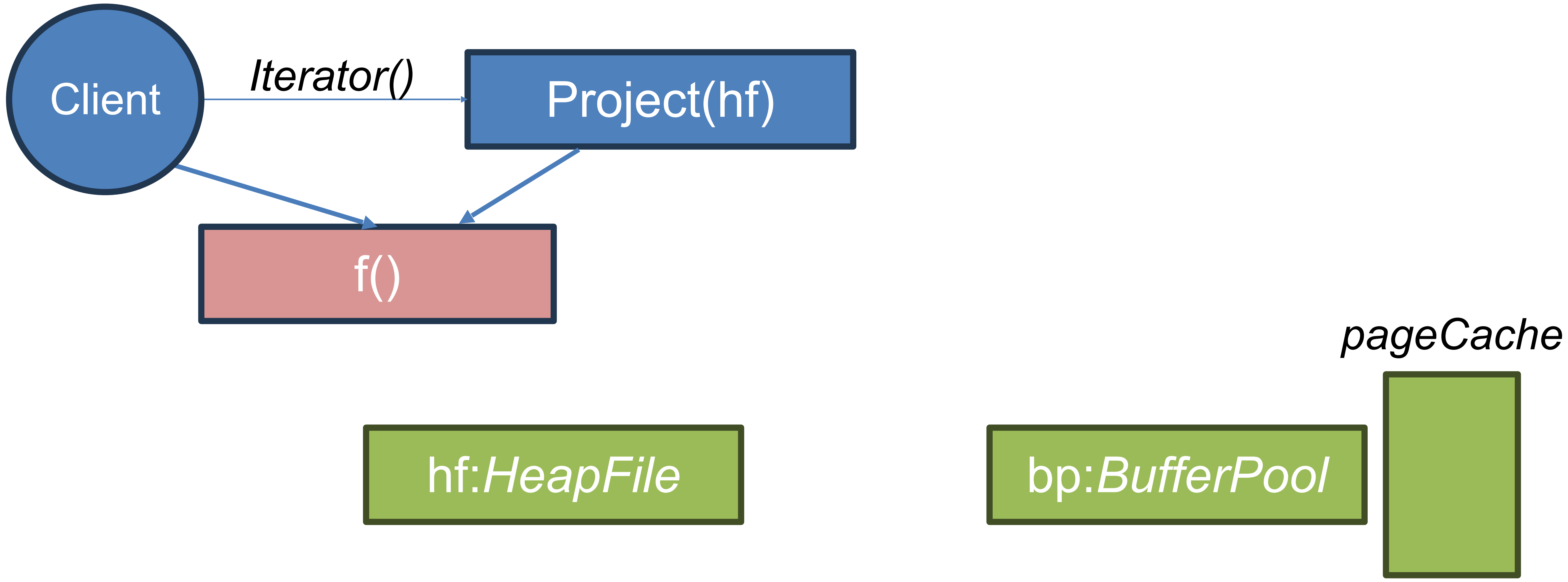


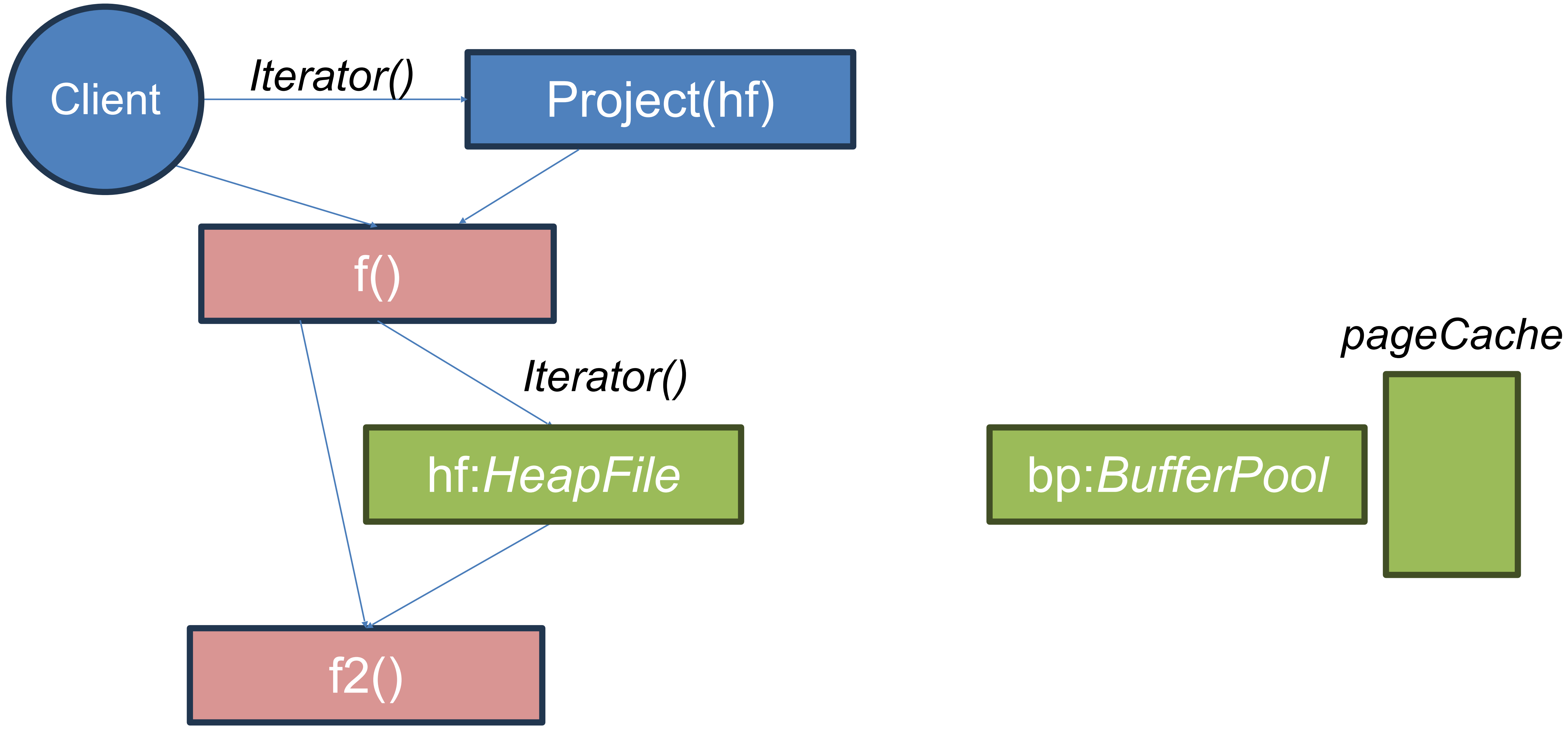
Client

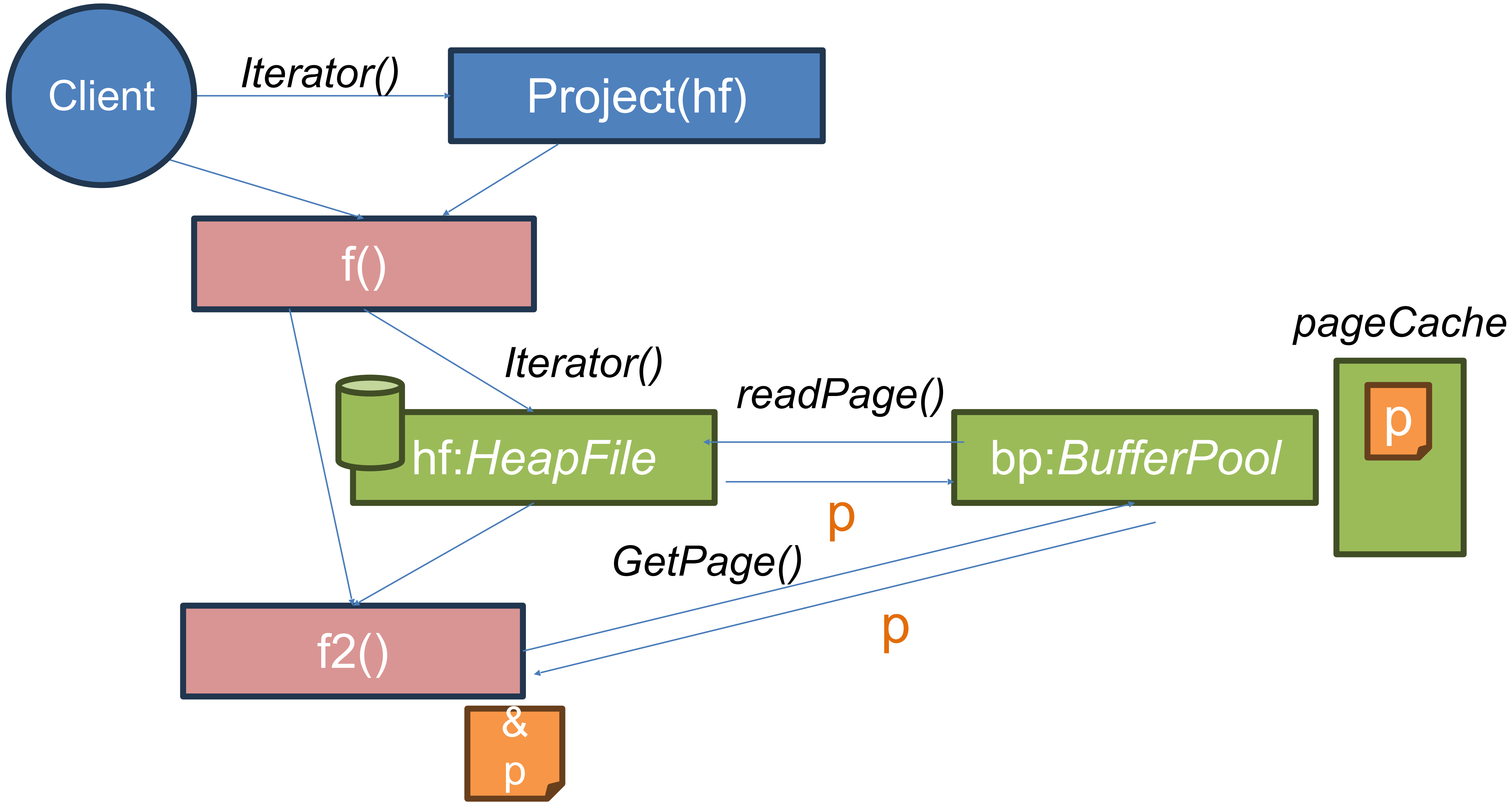
hf:*HeapFile*

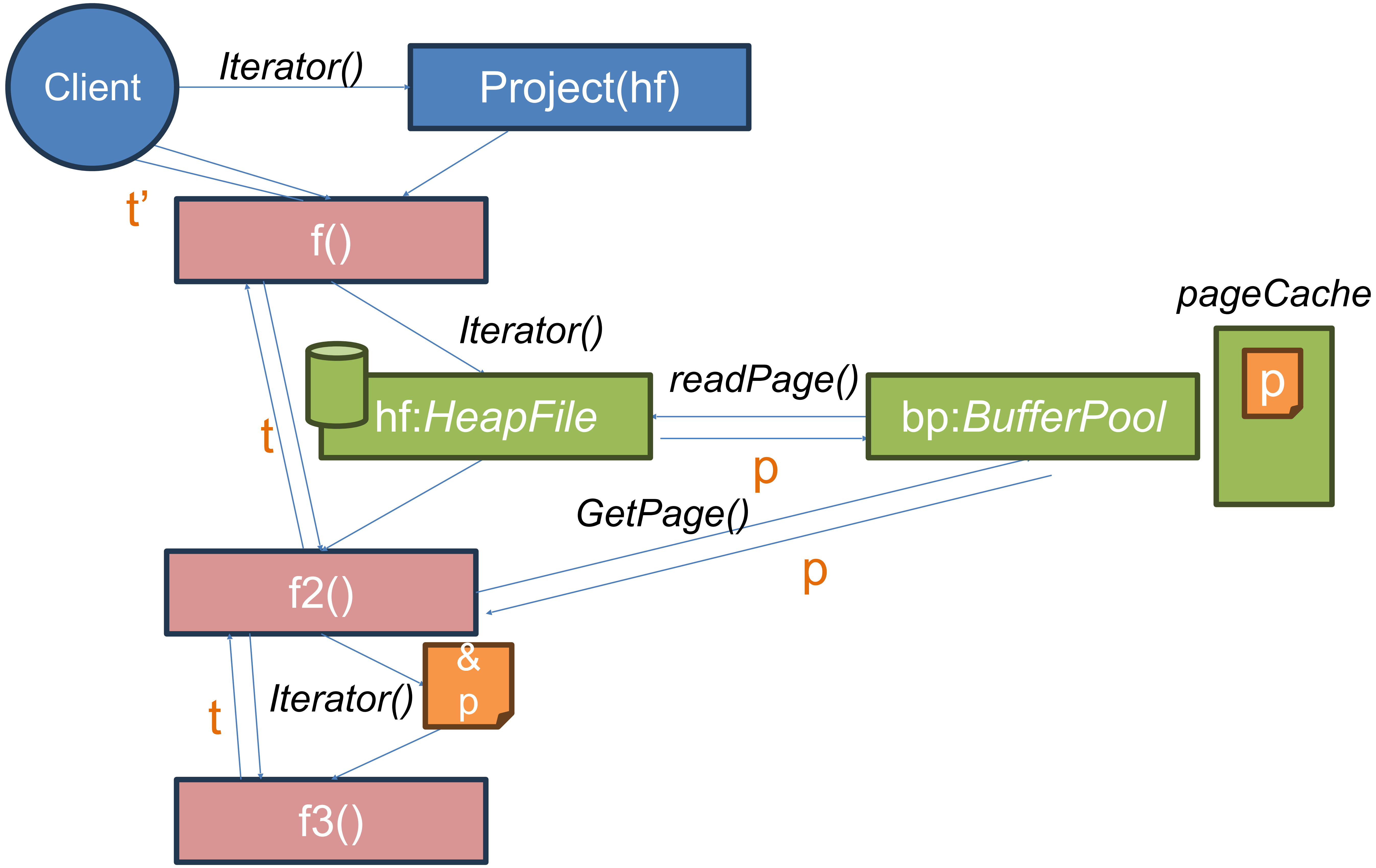
bp:*BufferPool*

pageCache


Client
Iterator()
Project(hf)
f()
pageCache
hf:*HeapFile*
bp:*BufferPool*


Client
Iterator()
Project(hf)
f()
Iterator()
hf:*HeapFile*
f2()
bp:*BufferPool*
pageCache


Client
Iterator()
Project(hf)
f()
Iterator()
hf:*HeapFile*
readPage()
p
bp:*BufferPool*
pageCache
p
GetPage()
p
f2()
&
p

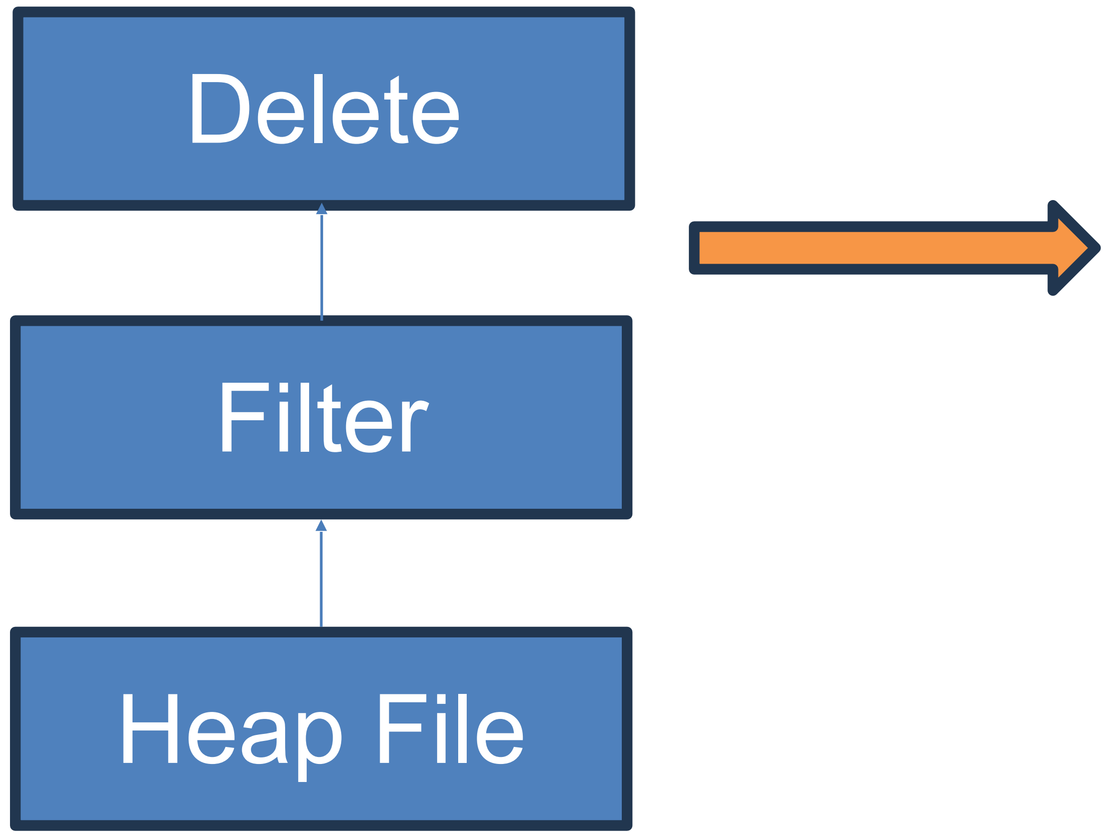

Client
Iterator()
Project(hf)
t'
f()
Iterator()
pageCache
p
t
hf:HeapFile
readPage()
bp:BufferPool
p
GetPage()
p
f2()
&
p
t
Iterator()
f3()

Deleting Records and Rids

- Consider a query like:

  DELETE FROM x WHERE f > 10

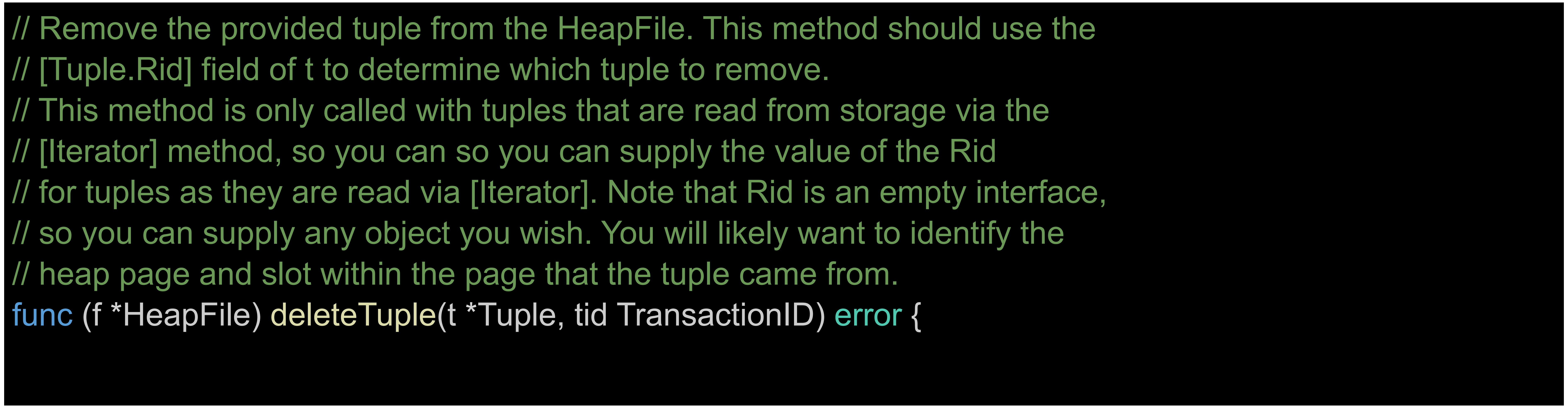
  This is translated into a plan like



Q: How does the delete operator know which records to delete?

A: Each record from the HeapFile is annotated with a *record id* that is used to identify the position of the record in the heap file to be deleted

Deleting Records and Rids



```
// Remove the provided tuple from the HeapFile. This method should use the
// [Tuple.Rid] field of t to determine which tuple to remove.
// This method is only called with tuples that are read from storage via the
// [Iterator] method, so you can so you can supply the value of the Rid
// for tuples as they are read via [Iterator]. Note that Rid is an empty interface,
// so you can supply any object you wish. You will likely want to identify the
// heap page and slot within the page that the tuple came from.
func (f *HeapFile) deleteTuple(t *Tuple, tid TransactionID) error {
```

- deleteTuple will be called by the delete operator
- Using the t.Rid object, you can clear out the position in the heap file containing the record
- Your heapfile implementation supplies the Rid in the iterator, and so you can identify this position however you like
- A standard Rid implementation is a page number and a slot within the page
  - Recall that all pages have the same number of slots

```
func computeFieldSum(fileName string, td TupleDesc, sumField string
) (int, error) {

        //Create buffer pool
        bp := NewBufferPool(10)

        hf, err := NewHeapFile("myfile.dat", &td, bp)
        …
        err = hf.LoadFromCSV(CSVfile, true, ",", false)

        //find the column
        fieldNo, err := findFieldInTd(FieldType{sumField, "", IntType}, &td)

        //Start a transaction -> we will do the implementation in another lab
        tid := NewTID()
        bp.BeginTransaction(tid)
        iter, err := hf.Iterator(tid)

        //Iterate through the tuples and sum them up.
        sum := 0
        for {
                tup, err := iter()
                f := tup.Fields[fieldNo].(IntField)
                sum += int(f.Value)
        }


        bp.CommitTransaction() //commit transaction
        return sum, nil //return the value
}
```

Bytes.Buffer

- https://pkg.go.dev/bytes#Buffer

Golang interface

- https://go.dev/tour/methods/10

Have Fun!

- Start early
- Let us know what you find confusing on Piazza!