LainDB
—— Yet Another Key-Value Store

Present by Zhang Yuning
Stu No. 5140379063

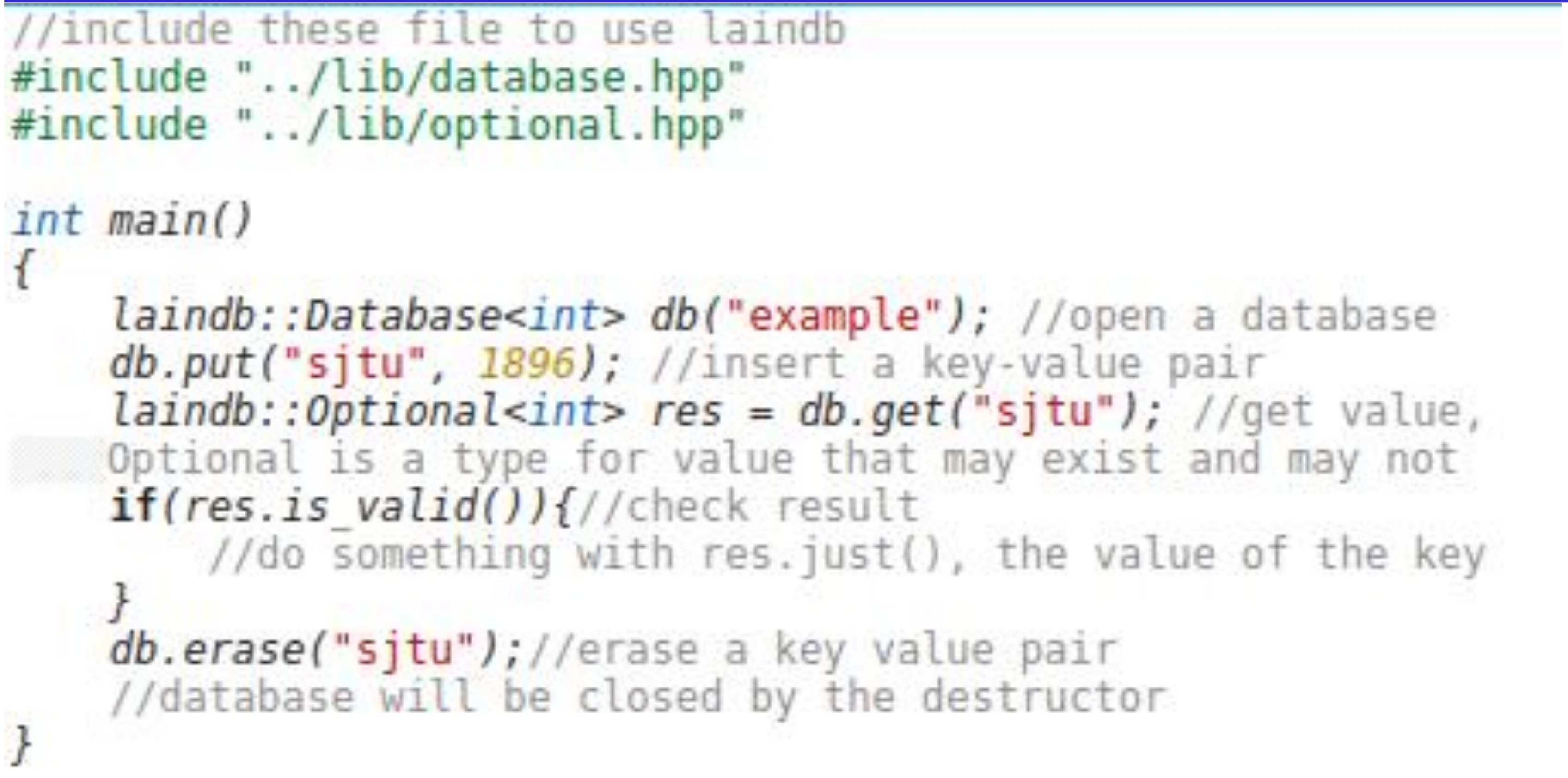
Outline

- Mission
- Example
- API Details
- Architecture
- LainDB in Depth
- Serial Experiments LainDB

Mission

- fast & flexible (at the same time)
- Cross-platform
- minimal interface (easy to use & hard to write unsafe codes)

Example



```
//include these file to use laindb
#include "../lib/database.hpp"
#include "../lib/optional.hpp"

int main()
{
    laindb::Database<int> db("example"); //open a database
    db.put("sjtu", 1896); //insert a key-value pair
    laindb::Optional<int> res = db.get("sjtu"); //get value,
    Optional is a type for value that may exist and may not
    if(res.is_valid()){//check result
        //do something with res.just(), the value of the key
    }
    db.erase("sjtu");//erase a key value pair
    //database will be closed by the destructor
}
```

API Details

```
template <typename Value,
          typename ValueSerializer =
DefaultSerializer<Value>,
          typename Index = BptreeIndex,
          typename Data = DataStore<DefaultAllocator> >
class Database;
```

- flexible – modular design

API Details

Database(const std::string & name, FileMode mode = CREATE);

Optional<Value> get(const char * key);

void put(const char * key, const Value & value);

void erase(const char * key);

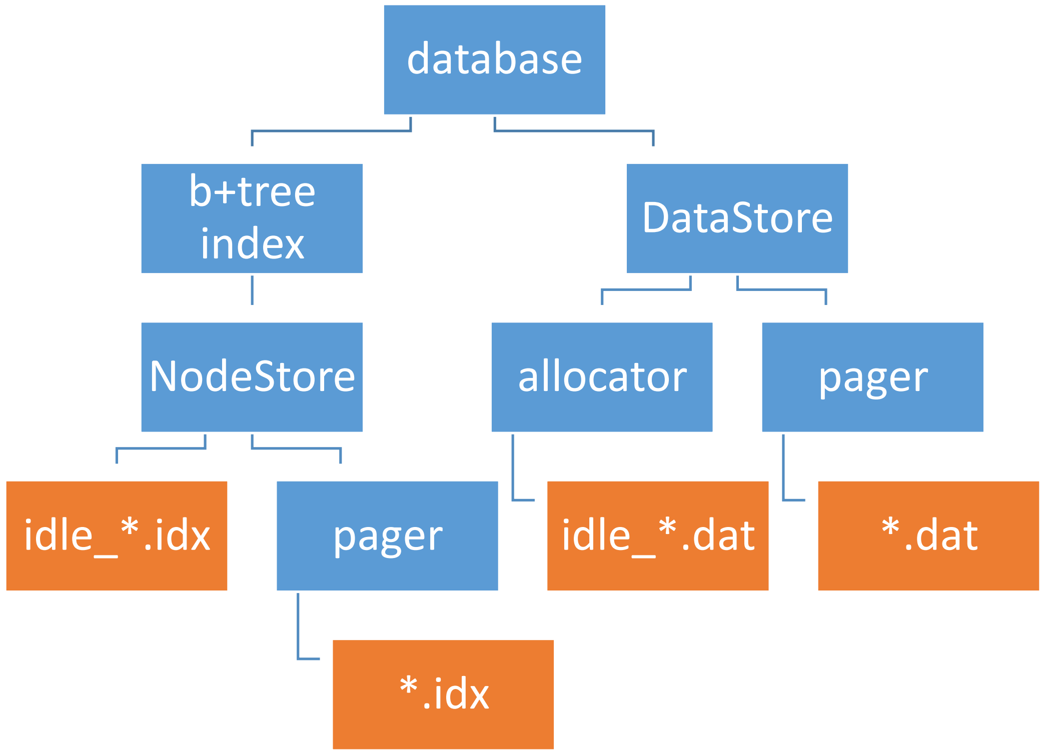
//destructor closes the database

API Details

- Optional<Value> -- force coders to validate
- {Nothing, Just<Value>}
- bool is_valid();
- const T & just();

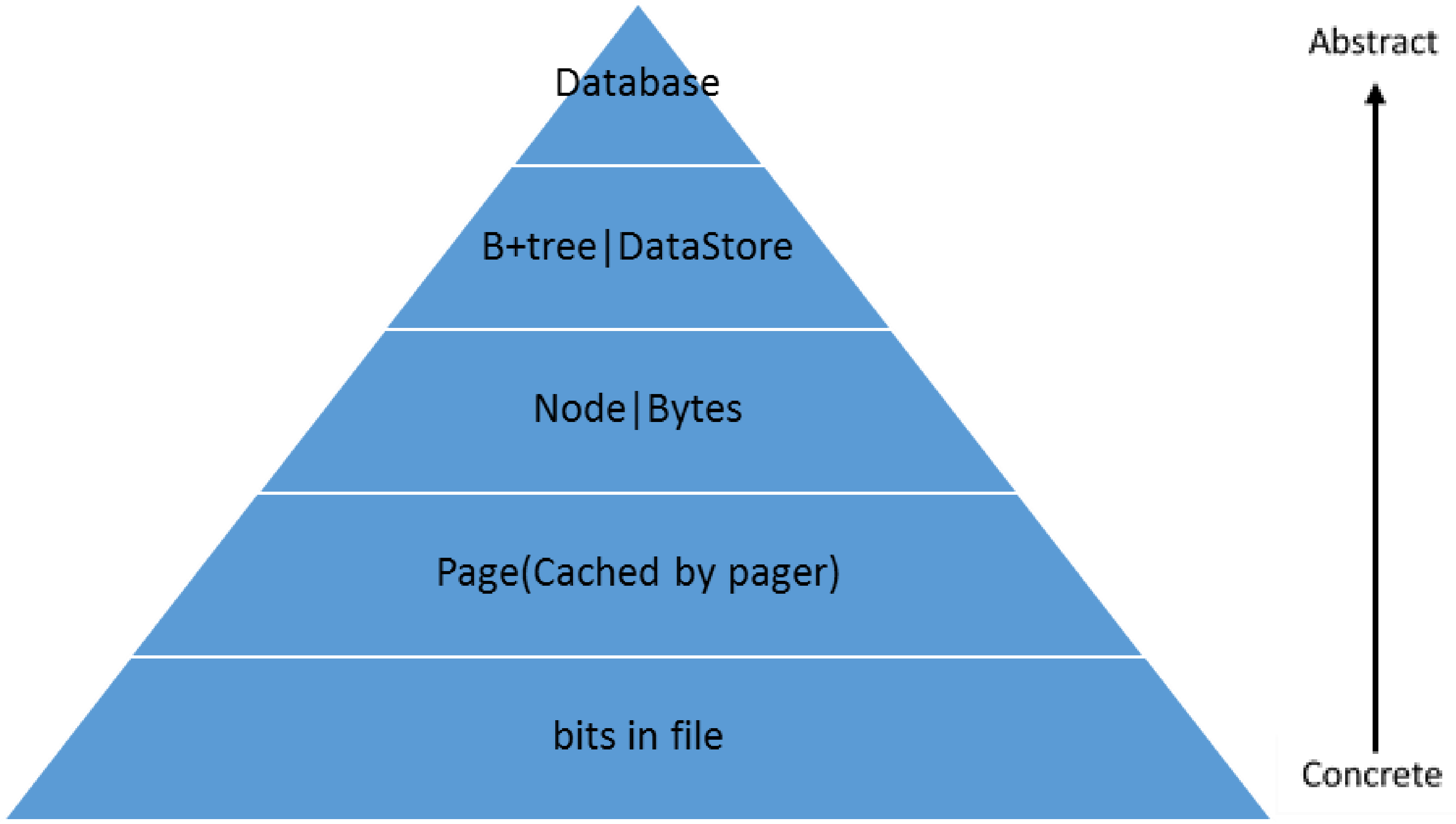
Architecture

Architecture



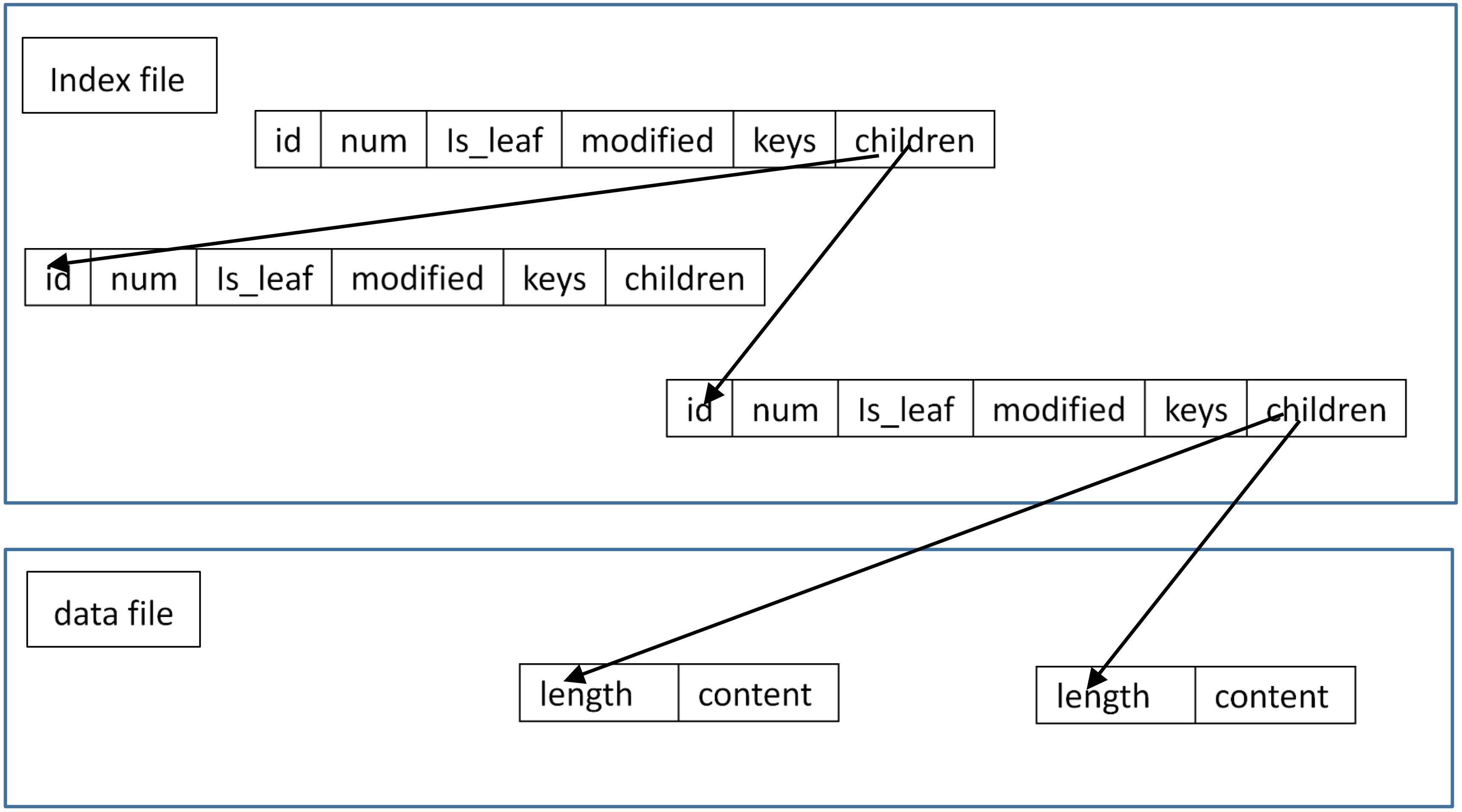
- interact using the interface
- easy to be replaced

Abstractions with Data



LainDB in Depth

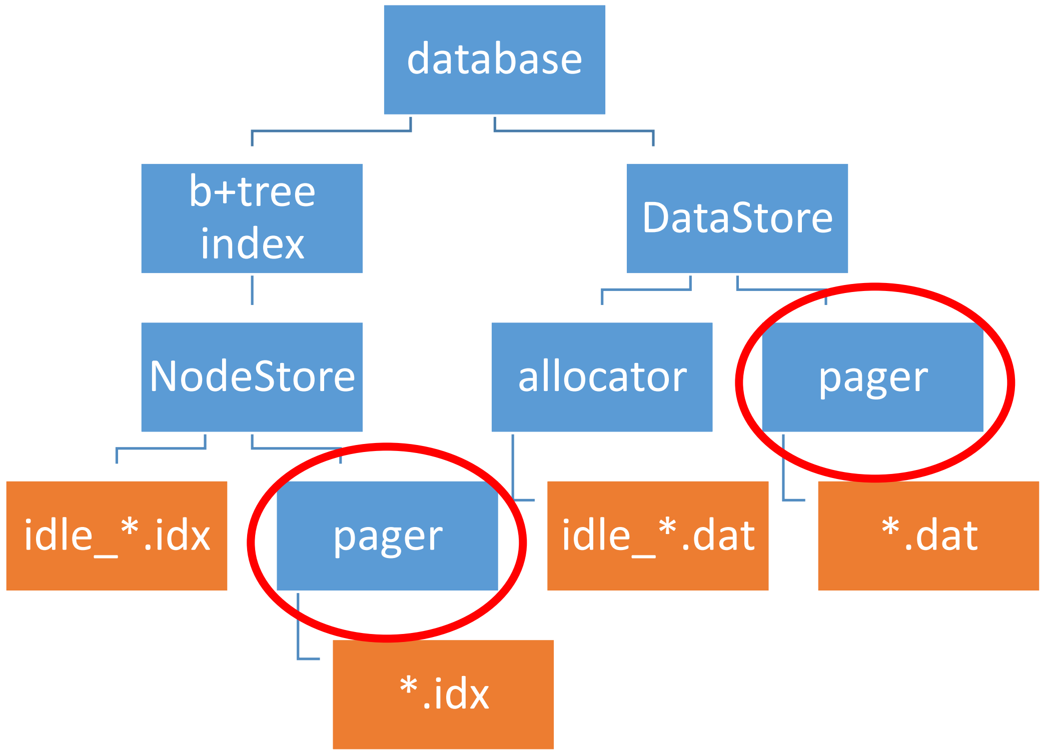
Overview


Index file
id
num
Is_leaf
modified
keys
children
data file
length
content

B+tree

- top-down B+tree
- adjust nodes when going down
- avoids recursion
- Fetch nodes on demand
  -- still fast thanks to the pager

Pager

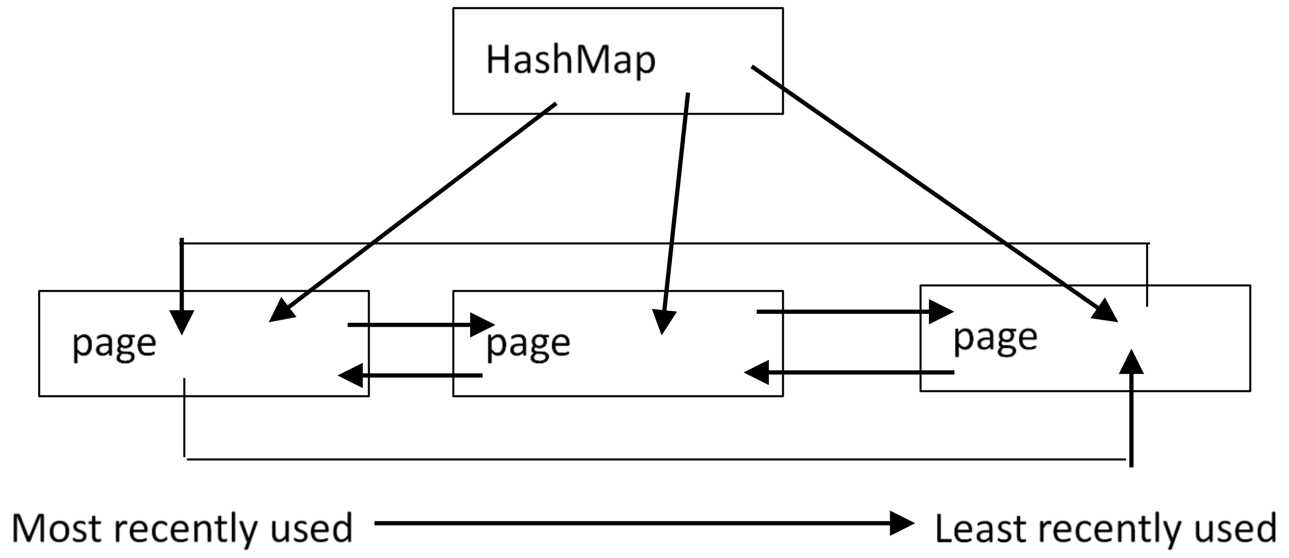

database
b+tree index
DataStore
NodeStore
allocator
pager
idle_*.idx
pager
idle_*.dat
*.dat
*.idx

Pager

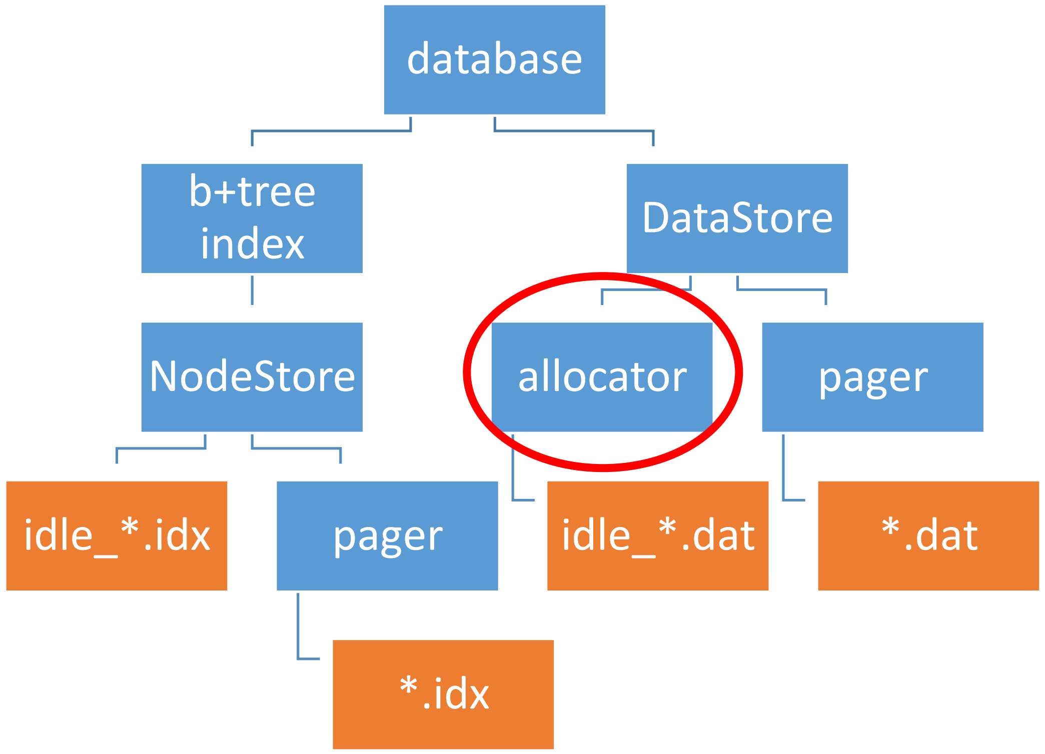
- cache for file
- manages file in unit of page
- uses the write-back method to keep consistency

Pager

- uses LRU(Least Recently Used) policy



Allocator


database
b+tree index
DataStore
NodeStore
allocator
pager
idle_*.idx
pager
idle_*.dat
*.dat
*.idx

Allocator

- Value be allocated by the allocator of the DataStore
- Two kind of allocators: AppendOnlyAllocator and DefaultAllocator

AppendOnlyAllocator

- just append data at the end of the datafile

DefaultAllocator

- best-fit strategy:
- keep deallocated space's information in a ordered linked list

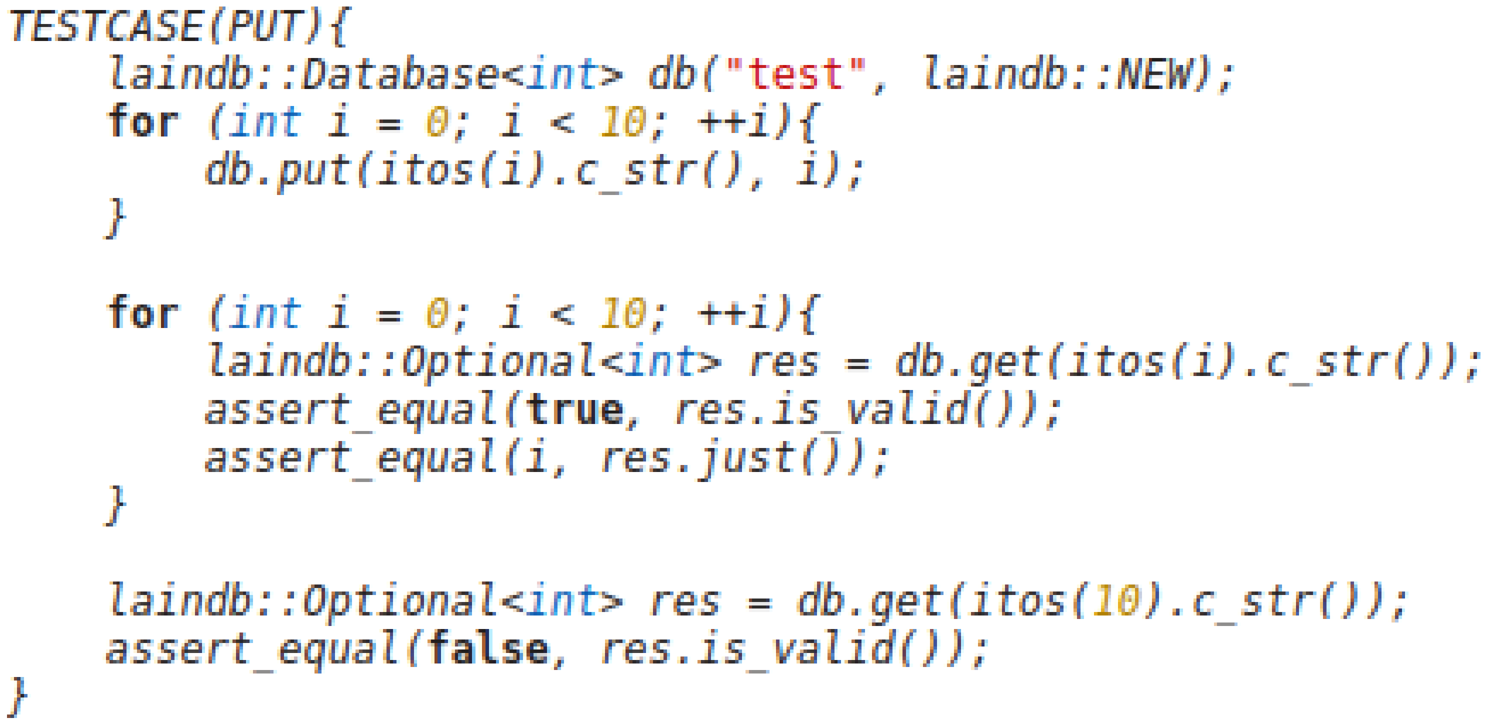
Other tricks

- Assertion at compile time
- Type traits
- ……

Serial Experiments LainDB

Correctness Tests

- simple automatic test framework
- developed with macro and template techniques



```
TESTCASE(PUT){
    laindb::Database<int> db("test", laindb::NEW);
    for (int i = 0; i < 10; ++i){
        db.put(itos(i).c_str(), i);
    }

    for (int i = 0; i < 10; ++i){
        laindb::Optional<int> res = db.get(itos(i).c_str());
        assert_equal(true, res.is_valid());
        assert_equal(i, res.just());
    }

    laindb::Optional<int> res = db.get(itos(10).c_str());
    assert_equal(false, res.is_valid());
}
```

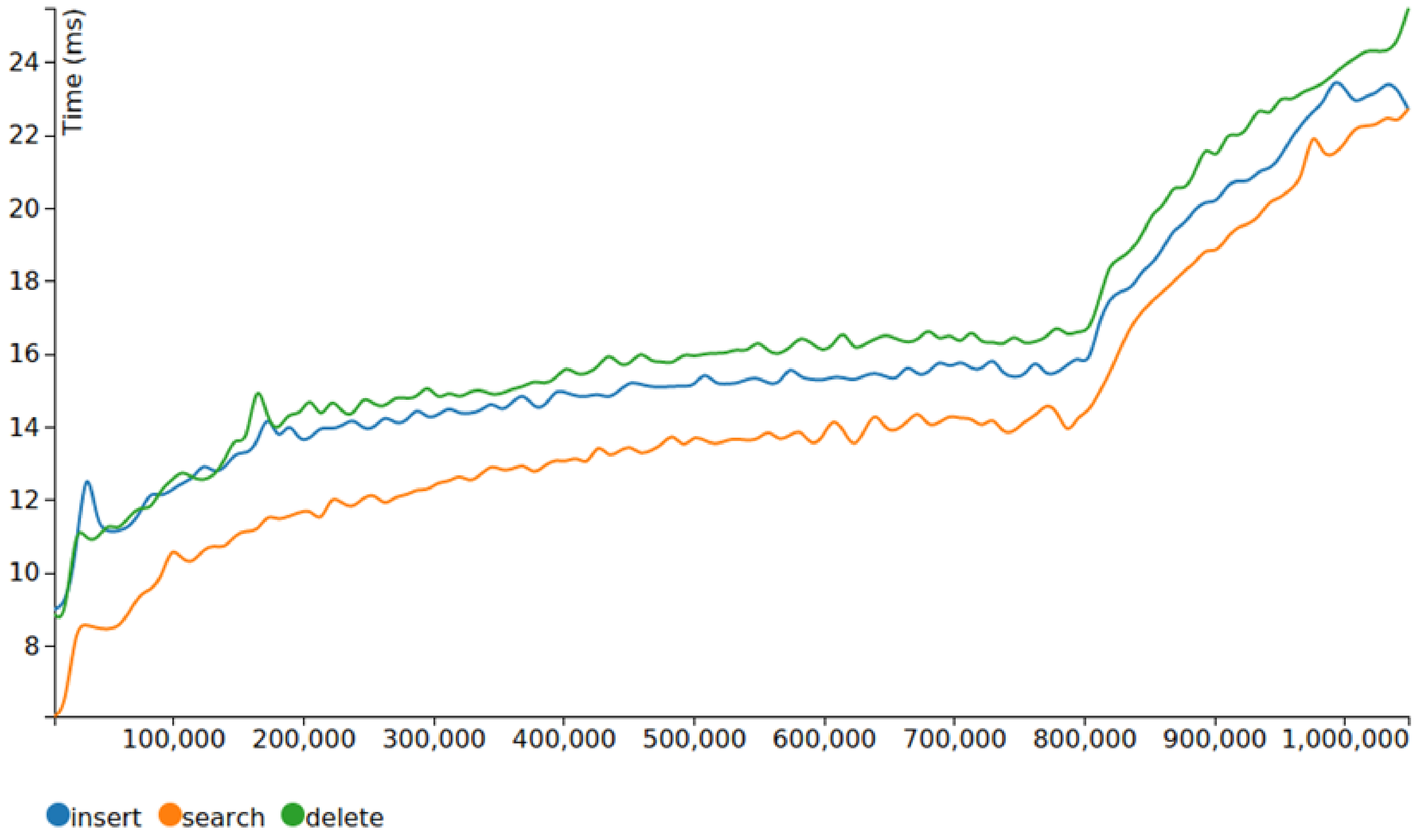
Correctness Tests

- checked by comparing results with the std::map under random operation sequences.

Performance test for insert, search & delete

- Using the algorithm below:
- 1. randomly insert 2^13 entries
- 2. randomly insert 2^13 entries and record time
- 3. randomly search 2^13 entries and record time
- 4. randomly delete 2^13 entries and record time
- Repeat until the number of entries reaches 2^20(1048576).

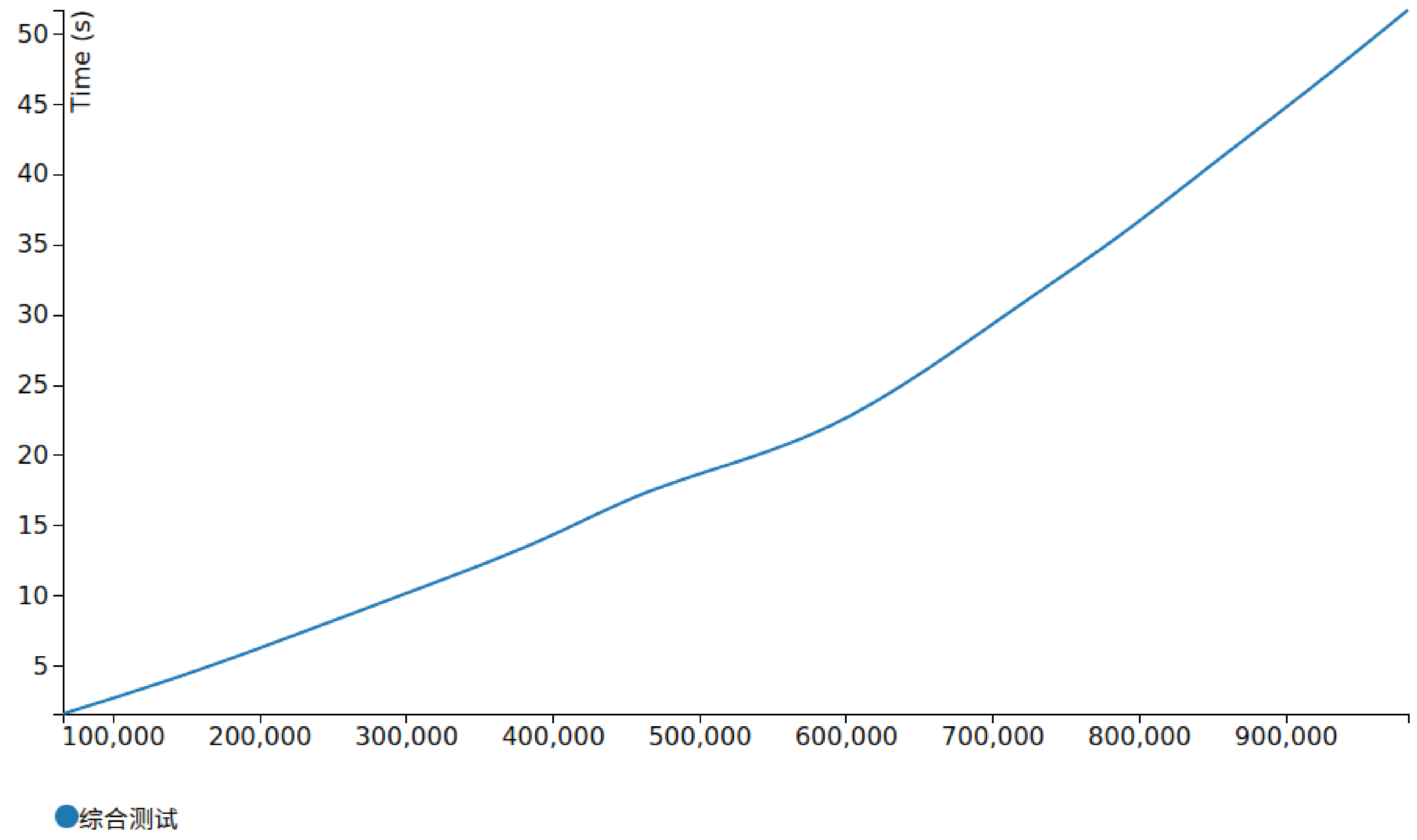
Performance test for insert, search & delete



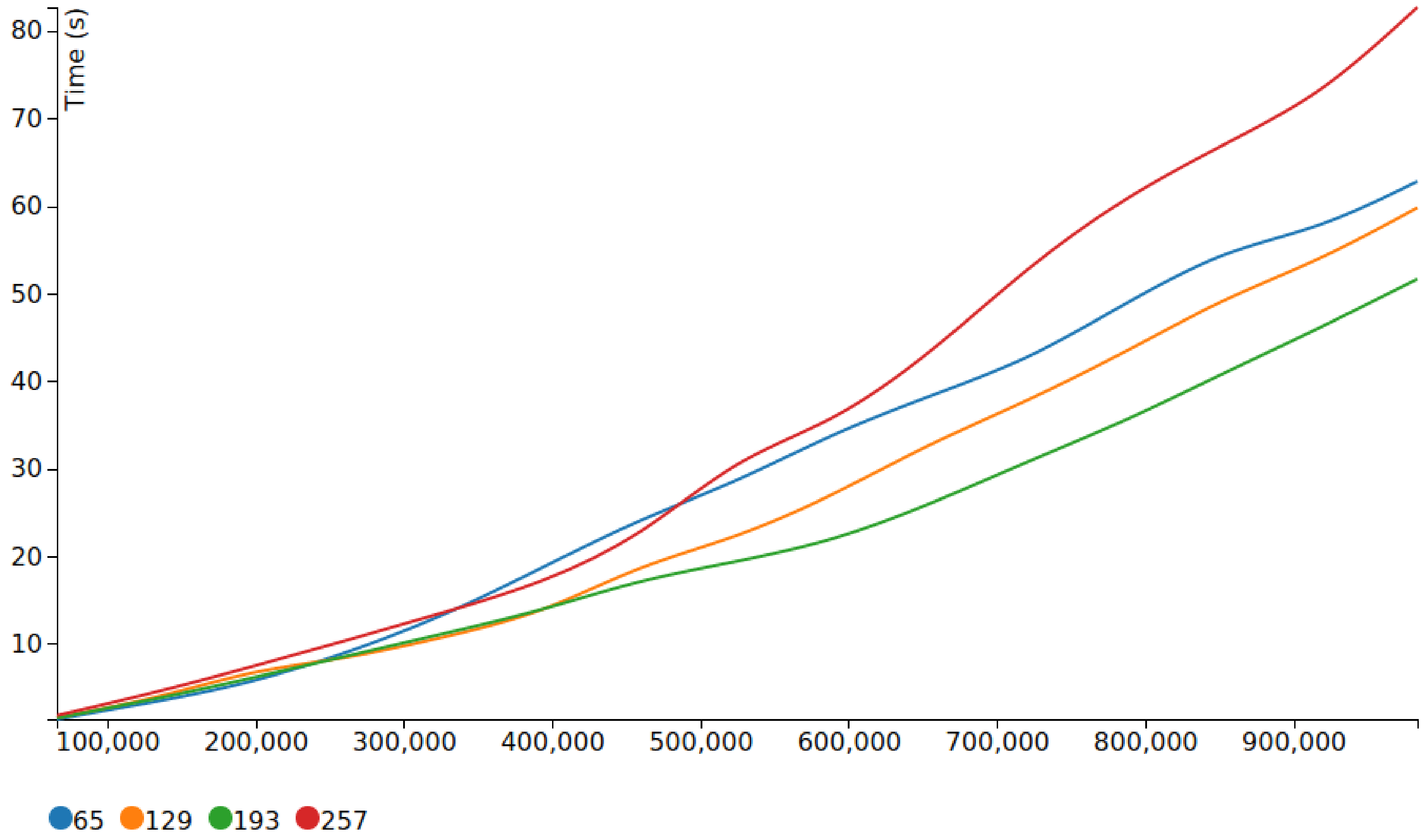
Benchmark

- Using the method from *Advanced Programming in the UNIX Environment.*
- 1. insert NREC entries
- 2. fetch these entries
- 3. loop for 5 * NREC times:
- a. randomly fetch an entry
- b. randomly delete an entry, every 37 times
- c. insert an entry and fetch it, every 11 times
- d. randomly replace an entry, every 17 times
- 4. delete all entries; for each deletion, randomly fetch 10 records.

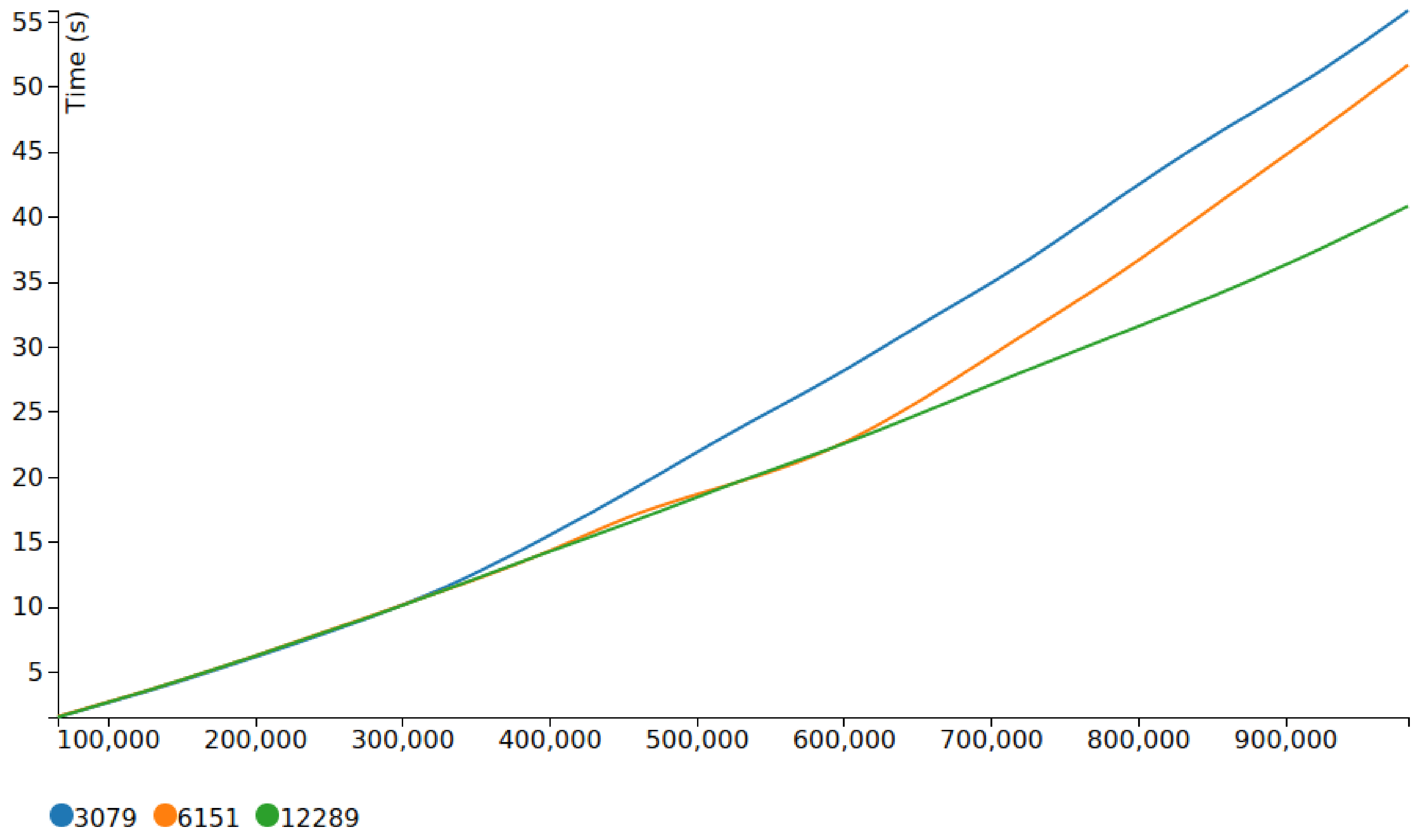
Benchmark


Time (s)
50
45
40
35
30
25
20
15
10
5
100,000
200,000
300,000
400,000
500,000
600,000
700,000
800,000
900,000
● 综合测试

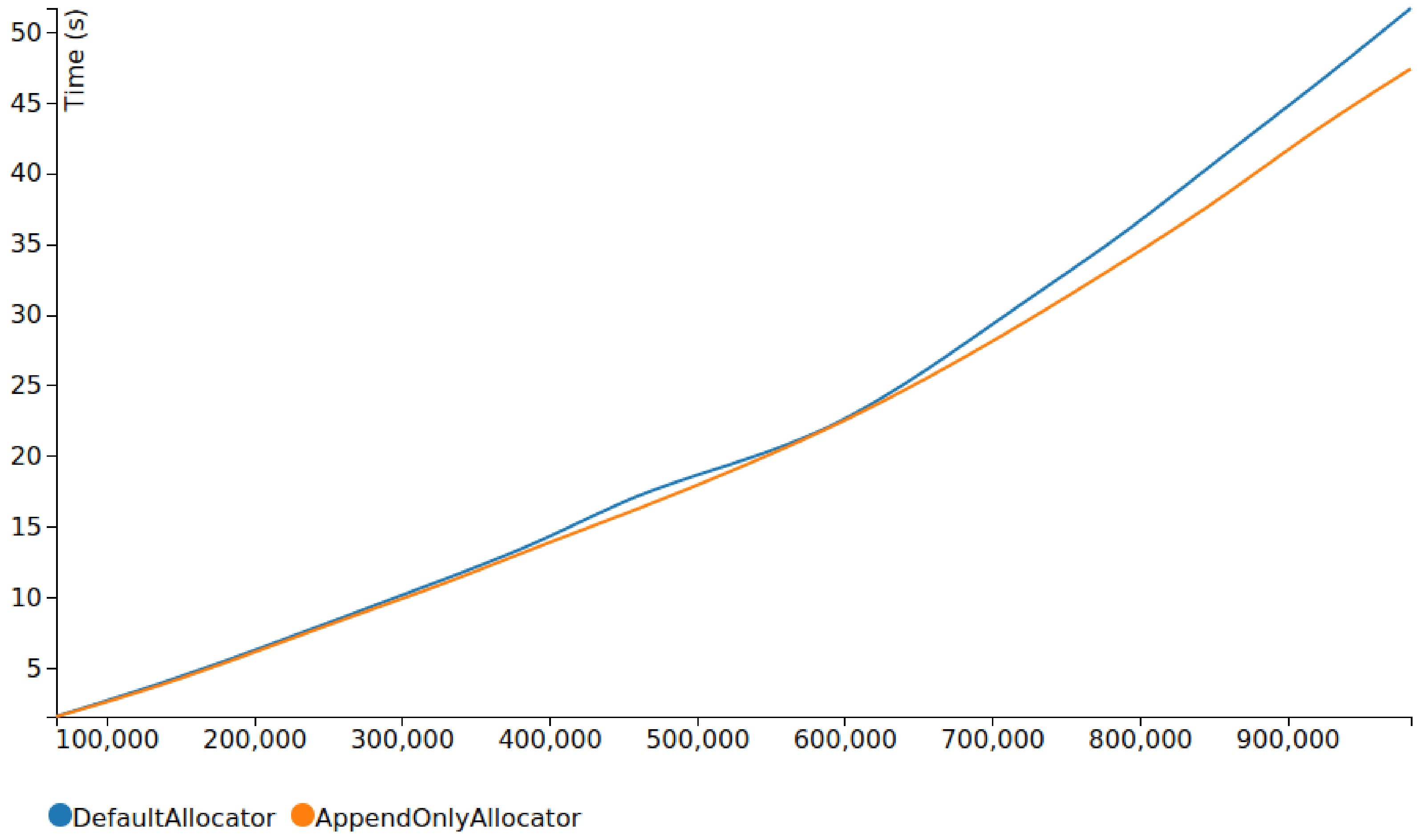
Degree of B+tree



Cache size


Time (s)
55
50
45
40
35
30
25
20
15
10
5
100,000
200,000
300,000
400,000
500,000
600,000
700,000
800,000
900,000
● 3079
● 6151
● 12289

Allocator


Time (s)
50
45
40
35
30
25
20
15
10
5
100,000
200,000
300,000
400,000
500,000
600,000
700,000
800,000
900,000
● DefaultAllocator
● AppendOnlyAllocator

Demonstration

Q&A

Thank you!