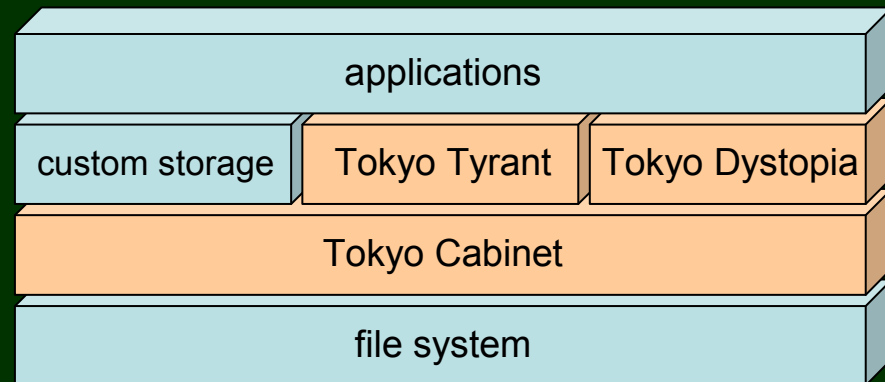


# ***Introduction to Tokyo Products***

**Mikio Hirabayashi**  
***[mikio@users.sourceforge.net](mailto:mikio@users.sourceforge.net)***

# Tokyo Products

- **Tokyo Cabinet**
  - database library
- **Tokyo Tyrant**
  - database server
- **Tokyo Dystopia**
  - full-text search engine



- **open source**
  - released under LGPL
- **powerful, portable, practical**
  - written in the standard C, optimized to POSIX

# **Tokyo Cabinet**

**- database library -**

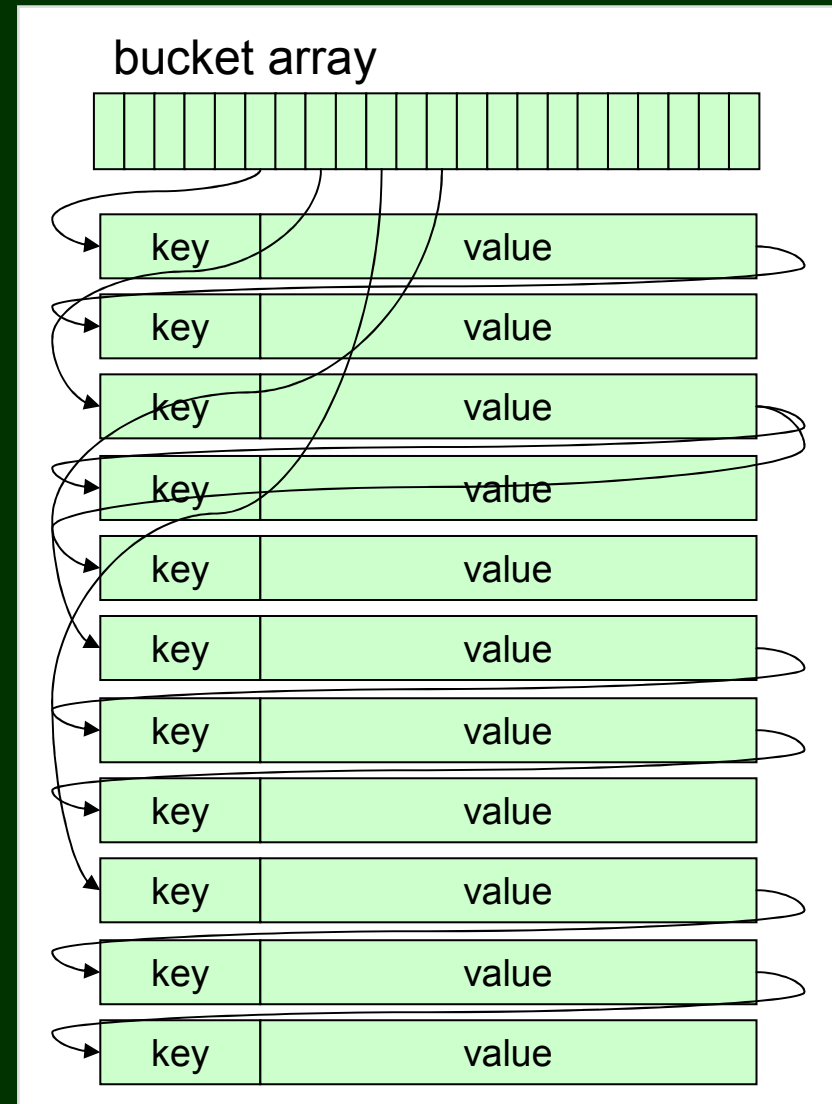
# Features

- **modern implementation of DBM**
  - **key / value database**
    - e.g.) DBM, NDBM, GDBM, TDB, CDB, Berkeley DB
  - **simple library = process embedded**
  - **Successor of QDBM**
    - C99 and POSIX compatible, using Pthread, mmap, etc...
    - Win32 porting is work-in-progress
- **high performance**
  - **insert: 0.4 sec / 1M records (2,500,000 qps)**
  - **search: 0.33 sec / 1M records (3,000,000 qps)**

- **high concurrency**
  - multi-thread safe
  - read/write locking by records
- **high scalability**
  - hash and B+tree structure =  $O(1)$  and  $O(\log N)$
  - no actual limit size of a database file (to 8 exabytes)
- **transaction**
  - write ahead logging and shadow paging
  - ACID properties
- **various APIs**
  - on-memory list/hash/tree
  - file hash/B+tree/array/table
- **script language bindings**
  - Perl, Ruby, Java, Lua, Python, PHP, Haskell, Erlang, etc...

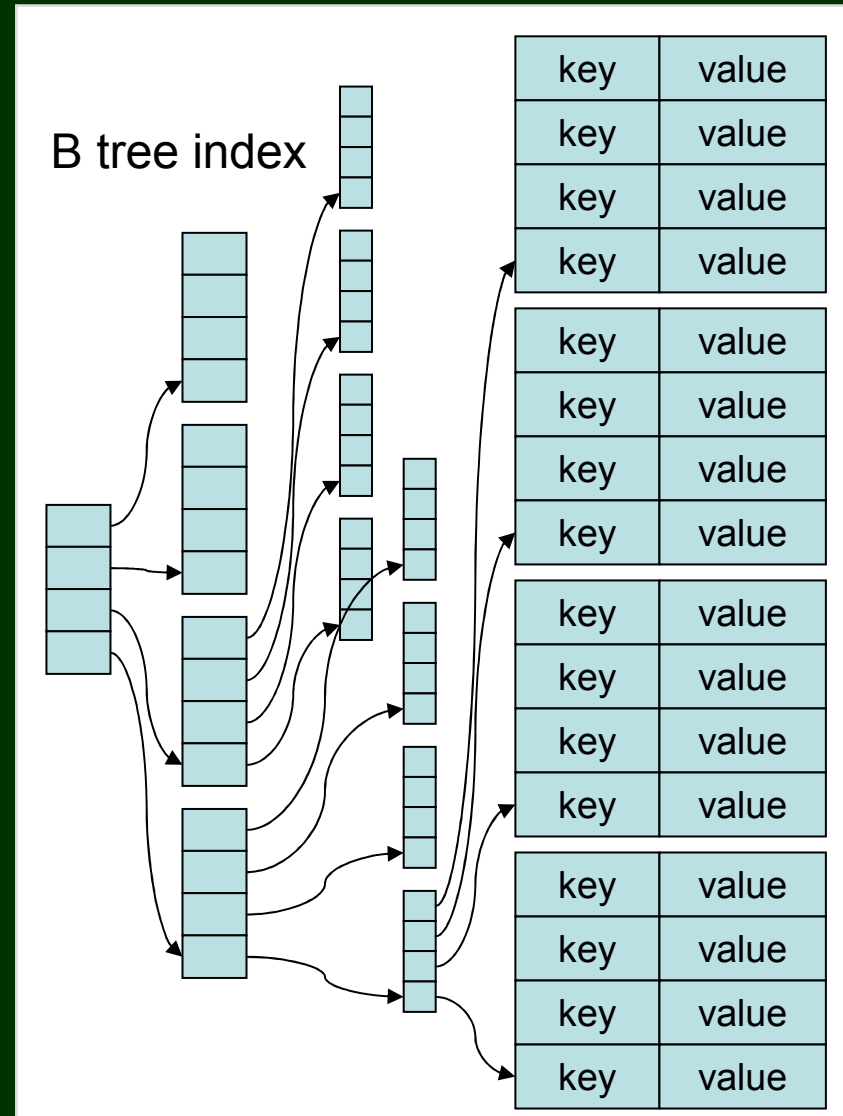
# TCHDB: Hash Database

- **static hashing**
  - $O(1)$  time complexity
- **separate chaining**
  - binary search tree
  - balances by the second hash
- **free block pool**
  - best fit allocation
  - dynamic defragmentation
- **combines mmap and pwrite/pread**
  - saves calling system calls
- **compression**
  - deflate(gzip) / bzip2 / custom



# TCBDB: B+ Tree Database

- **B+ tree**
  - $O(\log N)$  time complexity
- **page caching**
  - LRU removing
  - speculative search
- **stands on hash DB**
  - records pages in hash DB
  - succeeds time and space efficiency
- **custom comparison function**
  - prefix/range matching
- **cursor**
  - jump/next/prev



# TCFDB: Fixed-length Database

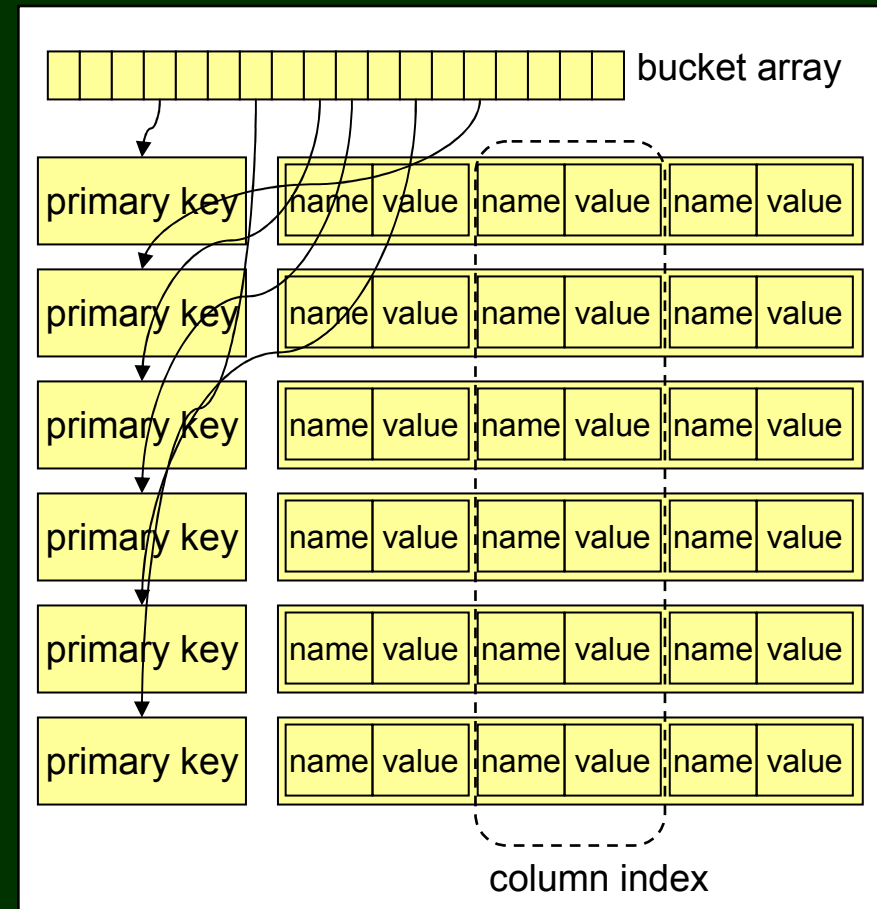
- **array of fixed-length elements**
  - **$O(1)$  time complexity**
  - **natural number keys**
  - **addresses records by multiple of key**
- **most effective**
  - **bulk load by mmap**
  - **no key storage per record**
  - **extremely fast and concurrent**

[illegible]



# TCTDB: Table Database

- **column based**
  - the primary key and named columns
  - stands on hash DB
- **flexible structure**
  - no data scheme, no data type
  - various structure for each record
- **query mechanism**
  - various operators matching column values
  - lexical/decimal orders by column values
- **column indexes**
  - implemented with B+ tree
  - typed as string/number
  - effective search by query optimizer



# On-memory Structures

- **TCXSTR: extensible string**
  - concatenation, formatted allocation
- **TCLIST: array list (dequeue)**
  - random access by index
  - push/pop, unshift/shift, insert/remove
- **TCMAP: map of hash table**
  - insert/remove/search
  - iterator by order of insertion
- **TCTREE: map of ordered tree**
  - insert/remove/search
  - iterator by order of comparison function

# Other Mechanisms

- **abstract database**
  - common interface of 6 schema
    - on-memory hash, on-memory tree
    - file hash, file B+tree, file array, file table
  - decides the concrete scheme in runtime
- **remote database**
  - network interface of the abstract database
  - yes, it's Tokyo Tyrant!
- **miscellaneous utilities**
  - string processing, filesystem operation
  - memory pool, encoding/decoding

# Example Code

```
#include <tcutil.h>
#include <tchdb.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>

int main(int argc, char **argv){

    TCHDB *hdb;
    int ecode;
    char *key, *value;

    /* create the object */
    hdb = tchdbnew();

    /* open the database */
    if(!tchdbopen(hdb, "casket.hdb", HDBOWRITER | HDBOCREAT)){
        ecode = tchdbecode(hdb);
        fprintf(stderr, "open error: %s\n", tchdberrmsg(ecode));
    }

    /* store records */
    if(!tchdbput2(hdb, "foo", "hop") ||
        !tchdbput2(hdb, "bar", "step") ||
        !tchdbput2(hdb, "baz", "jump")){
        ecode = tchdbecode(hdb);
        fprintf(stderr, "put error: %s\n", tchdberrmsg(ecode));
    }

    /* retrieve records */
    value = tchdbget2(hdb, "foo");
    if(value){
        printf("%s\n", value);
        free(value);
    } else {
        ecode = tchdbecode(hdb);
        fprintf(stderr, "get error: %s\n", tchdberrmsg(ecode));
    }
}
```

```
/* traverse records */
tchdbiterinit(hdb);
while((key = tchdbiternext2(hdb)) != NULL){
    value = tchdbget2(hdb, key);
    if(value){
        printf("%s:%s\n", key, value);
        free(value);
    }
    free(key);
}

/* close the database */
if(!tchdbclose(hdb)){
    ecode = tchdbecode(hdb);
    fprintf(stderr, "close error: %s\n", tchdberrmsg(ecode));
}

/* delete the object */
tchdbdel(hdb);

return 0;
}
```

# **Tokyo Tyrant**

**- database server -**

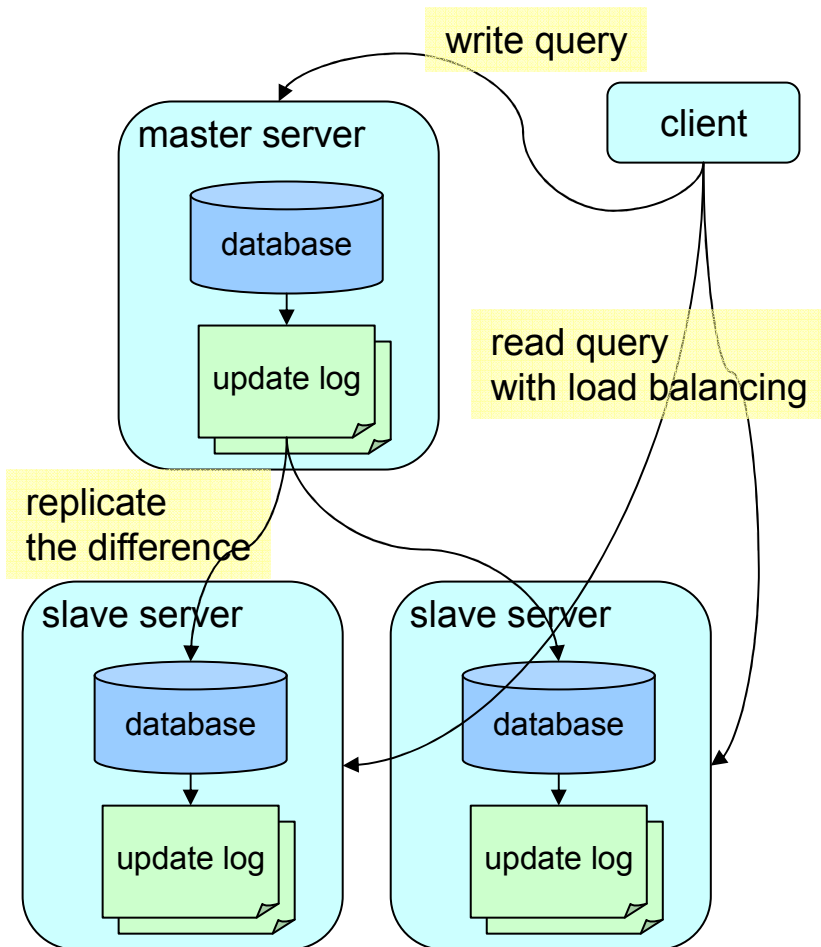
# Features

- **network server of Tokyo Cabinet**
  - client/server model
  - multi applications can access one database
  - effective binary protocol
- **compatible protocols**
  - supports memcached protocol and HTTP
  - available from most popular languages
- **high concurrency/performance**
  - resolves "c10k" with epoll/kqueue/eventports
  - 17.2 sec/1M queries (58,000 qps)

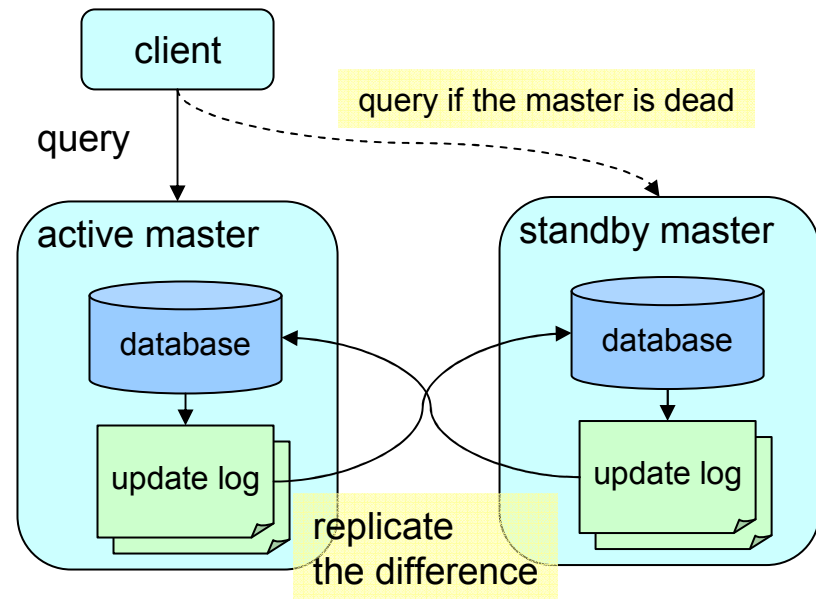
- **high availability**
  - hot backup and update log
  - asynchronous replication between servers
- **various database schema**
  - using the abstract database API of Tokyo Cabinet
- **effective operations**
  - no-reply updating, multi-record retrieval
  - atomic increment
- **Lua extension**
  - defines arbitrary database operations
  - atomic operation by record locking
- **pure script language interfaces**
  - Perl, Ruby, Java, Python, PHP, Erlang, etc...

# Asynchronous Replication

## master and slaves (load balancing)

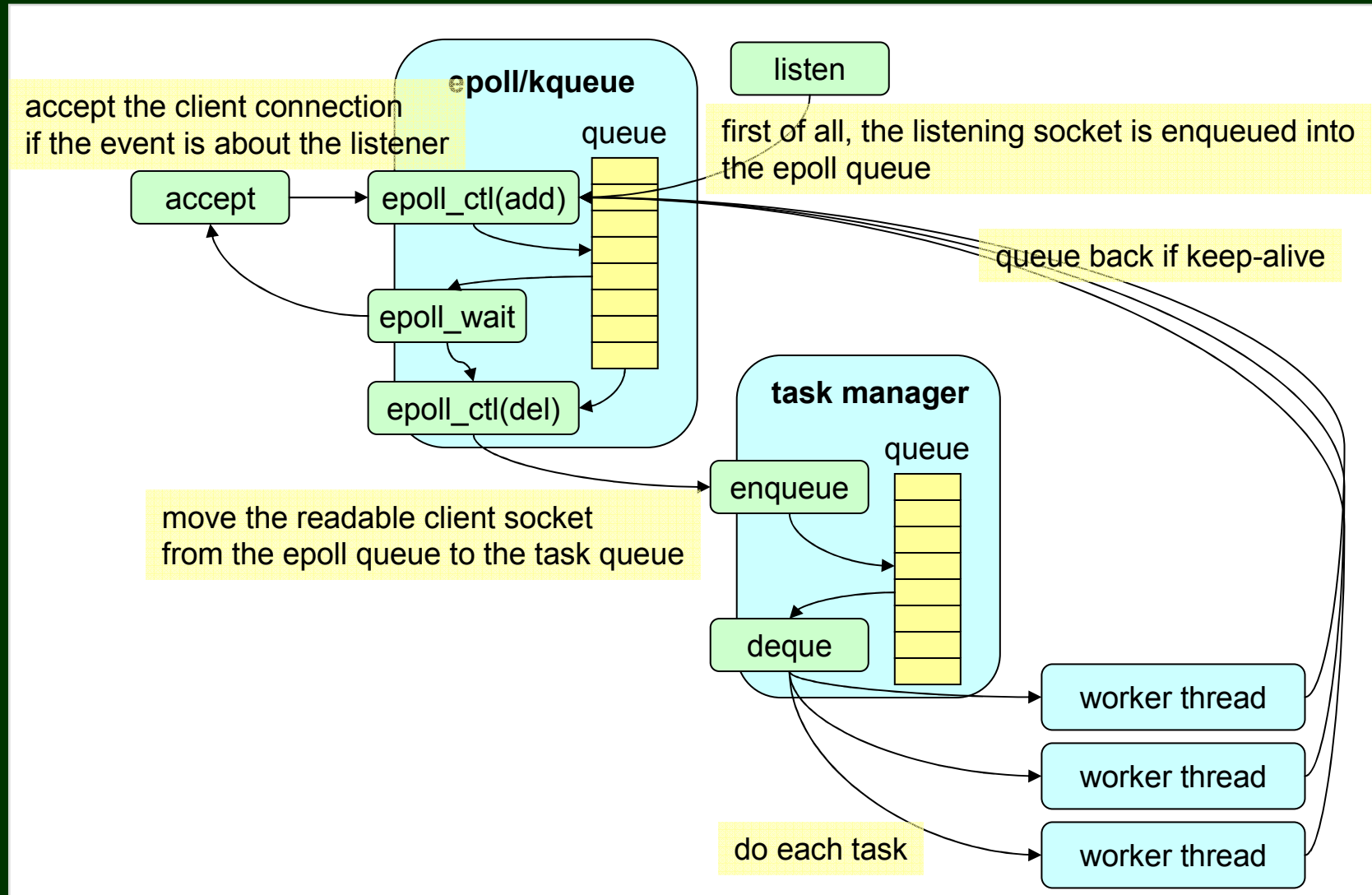


## dual master (fail over)



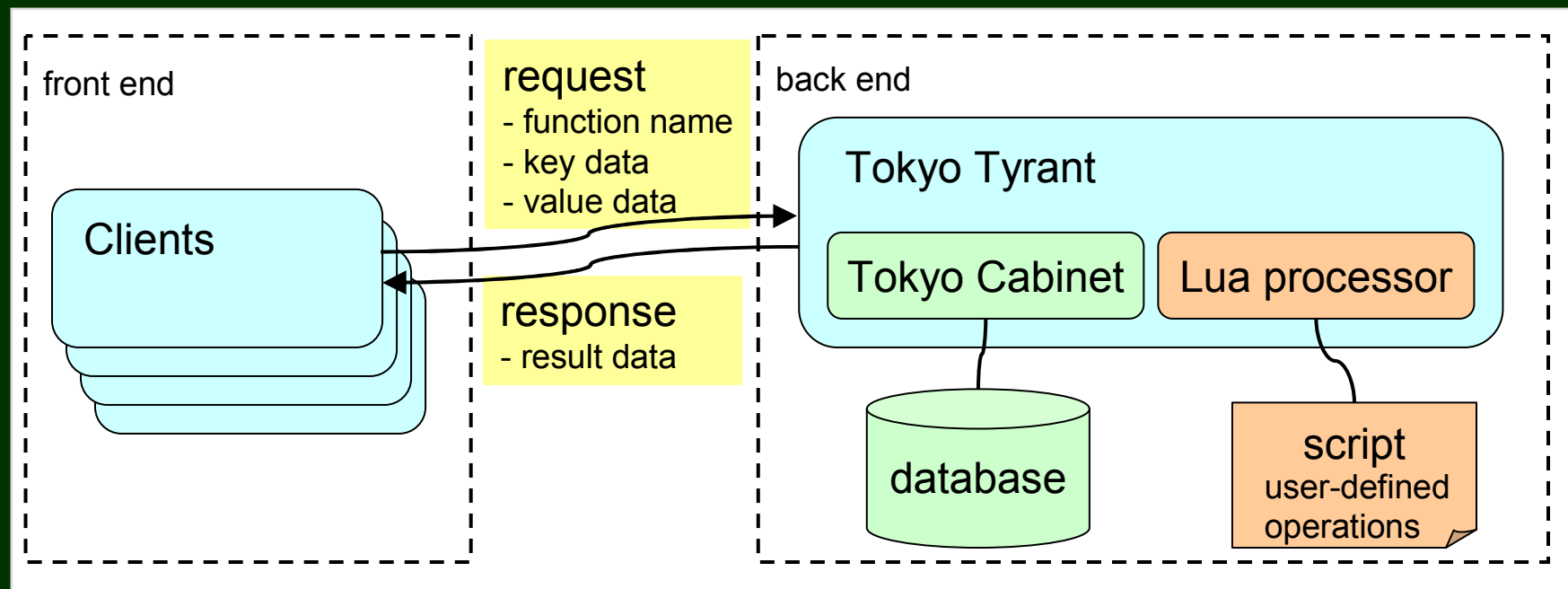


# Thread Pool Model



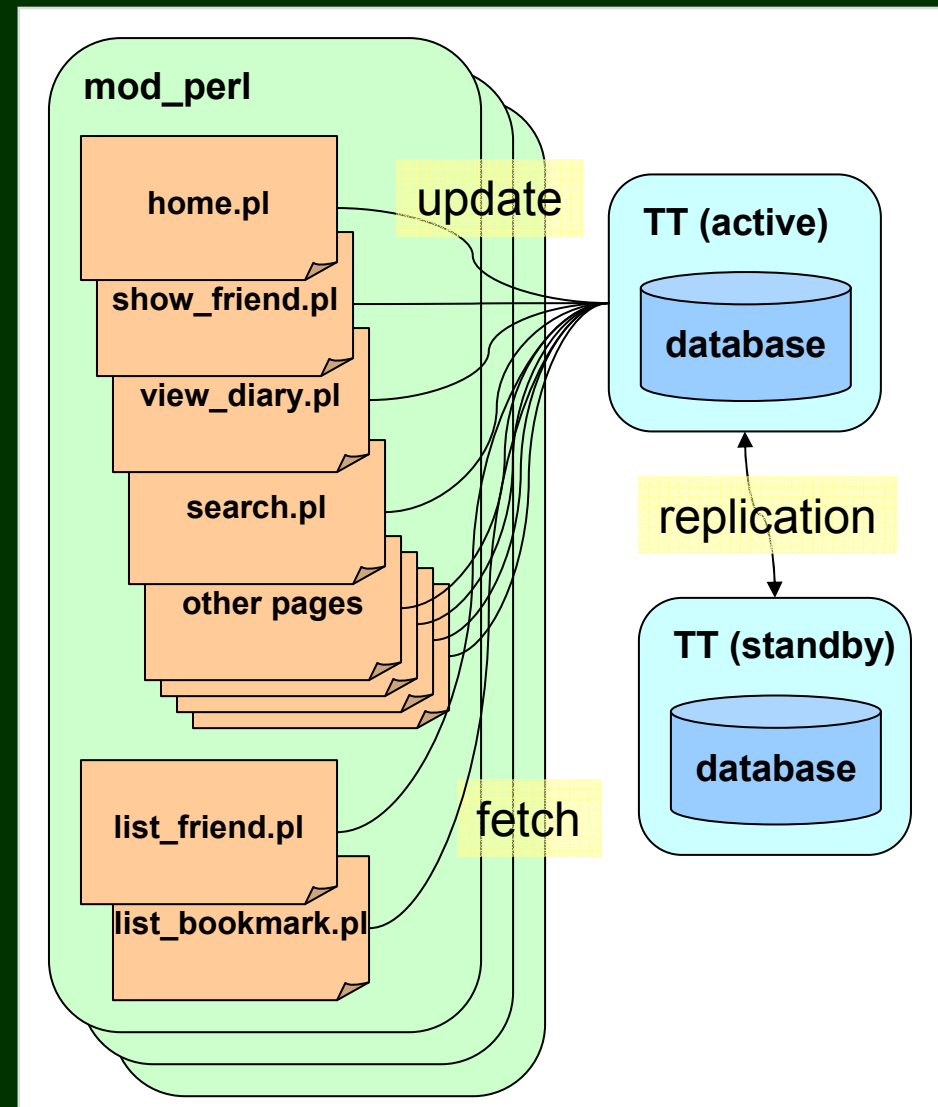
# Lua Extention

- **defines DB operations as Lua functions**
  - clients send the function name and record data
  - the server returns the return value of the function
- **options about atomicity**
  - no locking / record locking / global locking



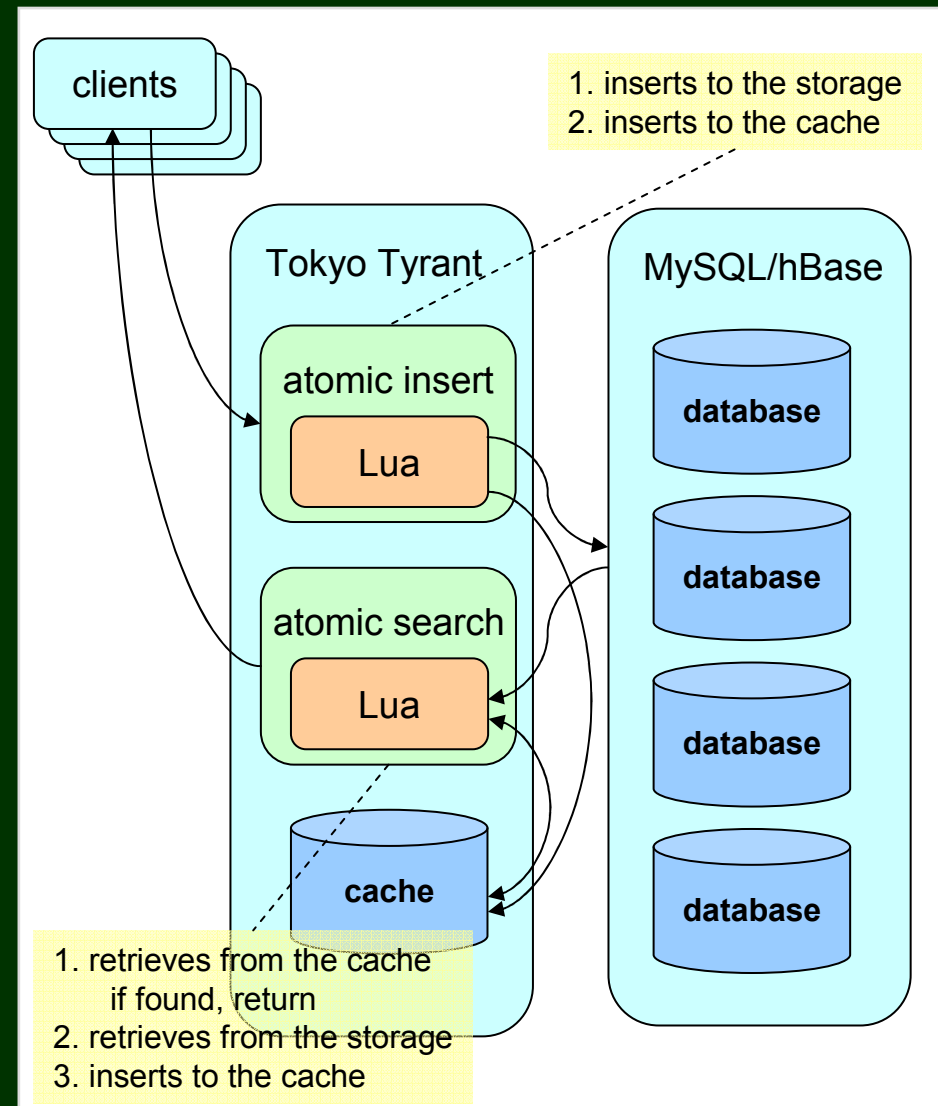
# case: Timestamp DB at mixi.jp

- **20 million records**
  - each record size is 20 bytes
- **more than 10,000 updates per sec.**
  - keeps 10,000 connections
- **dual master replication**
  - each server is only one
- **memcached compatible protocol**
  - reuses existing Perl clients



# case: Cache for Big Storages

- **works as proxy**
  - mediates insert/search
  - write through, read through
- **Lua extension**
  - atomic operation by record locking
  - uses LuaSocket to access the storage
- **proper DB scheme**
  - TCMDB: for generic cache
  - TCNDB: for biased access
  - TCHDB: for large records such as image
  - TCFDB: for small records such as timestamp



# Example Code

```
#include <tcldb.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>

int main(int argc, char **argv){

    TCRDB *rdb;
    int ecode;
    char *value;

    /* create the object */
    rdb = tcldbnew();

    /* connect to the server */
    if(!tcldbopen(rdb, "localhost", 1978)){
        ecode = tcldbdecode(rdb);
        fprintf(stderr, "open error: %s\n", tcldberrmsg(ecode));
    }

    /* store records */
    if(!tcldbput2(rdb, "foo", "hop") ||
        !tcldbput2(rdb, "bar", "step") ||
        !tcldbput2(rdb, "baz", "jump")){
        ecode = tcldbdecode(rdb);
        fprintf(stderr, "put error: %s\n", tcldberrmsg(ecode));
    }

    /* retrieve records */
    value = tcldbget2(rdb, "foo");
    if(value){
        printf("%s\n", value);
        free(value);
    } else {
        ecode = tcldbdecode(rdb);
        fprintf(stderr, "get error: %s\n", tcldberrmsg(ecode));
    }
}
```

```
/* close the connection */
if(!tcldbclose(rdb)){
    ecode = tcldbdecode(rdb);
    fprintf(stderr, "close error: %s\n", tcldberrmsg(ecode));
}

/* delete the object */
tcldbdel(rdb);

return 0;
}
```

# **Tokyo Dystopia**

**– full-text search engine –**

# Features

- **full-text search engine**
  - manages databases of Tokyo Cabinet as an inverted index
- **combines two tokenizers**
  - character N-gram (bi-gram) method
    - perfect recall ratio
  - simple word by outer language processor
    - high accuracy and high performance
- **high performance / scalability**
  - handles more than 10 million documents
  - searches in milliseconds

- **optimized to professional use**

- layered architecture of APIs
- no embedded scoring system
  - to combine outer scoring system
- no text filter, no crawler, no language processor

- **convenient utilities**

- multilingualism with Unicode
- set operations
- phrase matching, prefix matching, suffix matching, and token matching
- command line utilities



# Inverted Index

- **stands on key/value database**
  - **key = token**
    - N-gram or simple word
  - **value = occurrence data (posting list)**
    - list of pairs of document number and offset in the document
- **uses B+ tree database**
  - reduces write operations into the disk device
  - enables common prefix search for tokens
  - delta encoding and deflate compression

**ID:21**      **text:** "abracadabra"

a    - 21:10

ab   - 21:0,21:7

ac   - 21:3

br   - 21:5

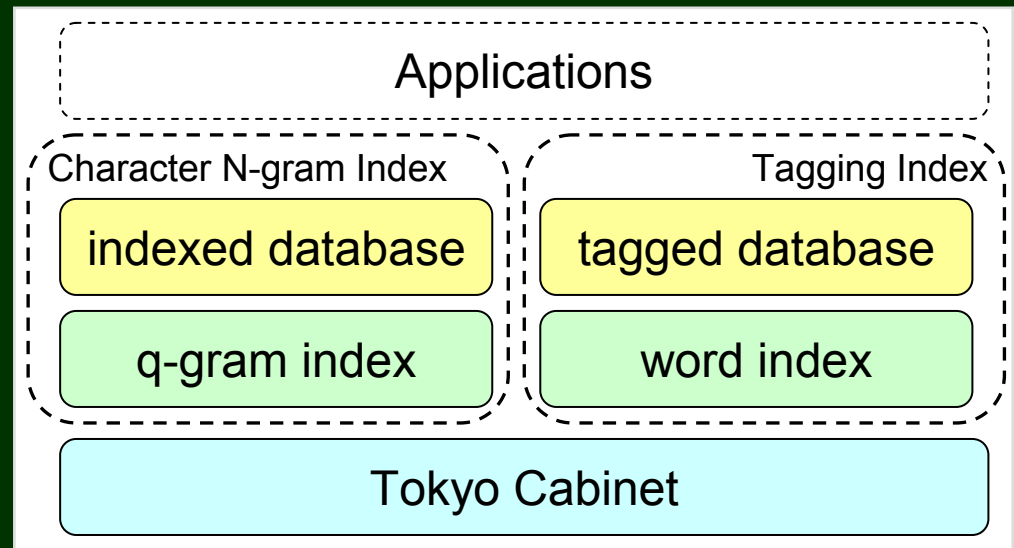
ca   - 21:1, 21:8

da   - 21:4

ra   - 21:2, 21:9

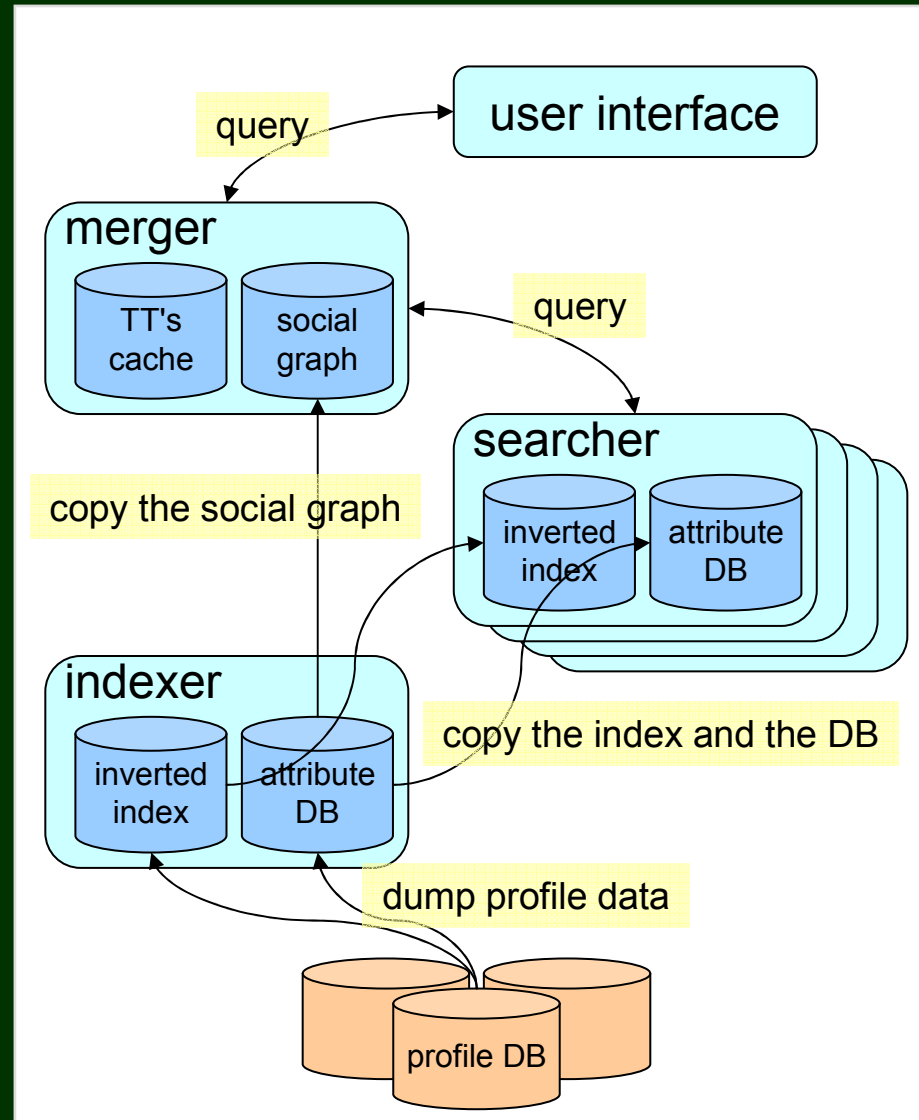
# Layered Architecture

- **character N-gram index**
  - "q-gram index" (only index), and "indexed database"
  - uses embedded tokenizer
- **word index**
  - "word index" (only index), and "tagged index"
  - uses outer tokenizer



# case: friend search at mixi.jp

- **20 million records**
  - each record size is 1K bytes
  - name and self introduction
- **more than 100 qps**
- **attribute narrowing**
  - gender, address, birthday
  - multiple sort orders
- **distributed processing**
  - more than 10 servers
  - indexer, searchers, merger
- **ranking by social graph**
  - the merger scores the result by following the friend links



# Example Code

```
#include <dystopia.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>

int main(int argc, char **argv){
    TCIDB *idb;
    int ecode, rnum, i;
    uint64_t *result;
    char *text;

    /* create the object */
    idb = tcidbnew();

    /* open the database */
    if(!tcidbopen(idb, "casket", IDBOWRITER | IDBOCREAT)){
        ecode = tcidbecode(idb);
        fprintf(stderr, "open error: %s\n", tcidberrmsg(ecode));
    }

    /* store records */
    if(!tcidbput(idb, 1, "George Washington") ||
        !tcidbput(idb, 2, "John Adams") ||
        !tcidbput(idb, 3, "Thomas Jefferson")){
        ecode = tcidbecode(idb);
        fprintf(stderr, "put error: %s\n", tcidberrmsg(ecode));
    }
}
```

```
/* search records */
result = tcidbsearch2(idb, "john || thomas", &rnum);
if(result){
    for(i = 0; i < rnum; i++){
        text = tcidbget(idb, result[i]);
        if(text){
            printf("%d\t%s\n", (int)result[i], text);
            free(text);
        }
    }
    free(result);
} else {
    ecode = tcidbecode(idb);
    fprintf(stderr, "search error: %s\n", tcidberrmsg(ecode));
}

/* close the database */
if(!tcidbclose(idb)){
    ecode = tcidbecode(idb);
    fprintf(stderr, "close error: %s\n", tcidberrmsg(ecode));
}

/* delete the object */
tcidbdel(idb);

return 0;
}
```

***innovating more and yet more...***

***<http://tokyocabinet.sourceforge.net/>***

東京



キャビネット 8192 PiB