

# LunarBase

-- A real time database engine for managing very large amounts of data



Auth:

[feiben@lunarion.com](mailto:feiben@lunarion.com)

[neo.carmack@lunarion.com](mailto:neo.carmack@lunarion.com)

**Lunarion Consultant Group**

# Application

**Big data era**

**Internet, Internet of Things**

**Use cases as general DB**

**Real-time mode**

**Geo-Spatial search engine**

**Structural, semi-structural search engine**

**Big data market grows from billions to thousands of billions, as Gartner surveyed world-wide. Several important facts worth paying attention to:**

1. 80% of data is unstructured or semi-structured
2. data coming from everywhere, e-commerce, communities, mobile, IOT.
3. Data business covers all the corners of the world: governments, financial, telecom, retail, logistic.....
4. Storage and Analysis requirements grow much more fast than the growth of data itself. Predefined schema, as what RDBMS does, just meets a little portion of this trend.
5. In big data ecosystem, performance, scalability and flexibility are the three key-indexes for all the supporting technologies, including hardwares, softwares, architectures.
6. Big data is just the beginning, digging out trends, precise info, on demand responding are the purpose.

# Application

## Big data era

**applications from big to:**  
precise  
trends  
on-demand

**Software:**  
performance  
flexibility  
scalability

### Big Data Ecosystem

**Data from:**  
internet  
smart devices  
sensor of every thing  
.....

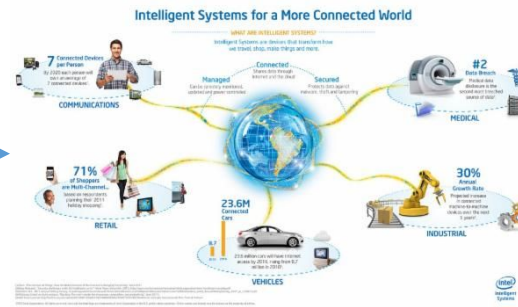
**Hardware:**  
multi-core  
cluster  
cloud

# Application

## Internet, Internet of Things



Traditional data comes from home and office, in front of computers.



Even within a small office, tons of data is generated every moment

But from now on, every thing commits data for analysis, diagnosis, communication. Data may from:

*Mobile phone*

*Auto cars*

*Nano robots inside your body*

*Refrigerator*

*Bed*

*Tooth brush*

*Meeting room projector*

*Coffee cup*

.....

# Application

## Use cases as general DB

**LunarBase is a database for genral purpose, use cases include but not limited to:**

1. Content management for internet and intranet
2. Business for goverments, financial, telecom, retail, logistic.....
3. e-Commerce transaction like shopping, travaling, hotel reservation.....
4. Financial big data analysis including up-selling, fraud detection, e-billing.....
5. Industry and consumer smart device data collection, storage, transmission
6. Social activity data management

.....

# Application

## Real Time mode

LunarBase supports real time analysis by LunarMax module, when its real time mode turned on. Close database, set `rt_mode = on`, and open database again, LunarMax is running.

After a database is created, user opens the configuration file and find the location where to turn on the real time mode under the "real time mode" section:

```
rt_mode = on
rt_analysable = string:name, int:payment, int:age, string:product

# order of how index building.
rt_order = 7

# for float data: 0 for integer, 1 for 0.1, 2 for 0.01, 3 for 0.001, and so on.
rt_precision = 3

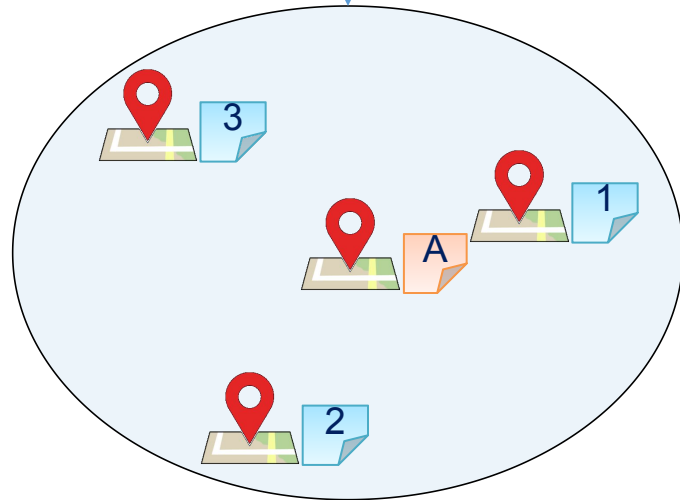
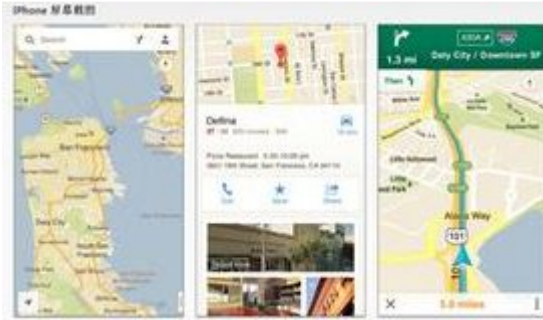
# threads for real time computation.
# Concerning performance, rt_threads+ internal_cache_concurrent_level
# will never exceeds the threads that OS can provide.
# One property with one thread gives the best performance.

rt_threads = 4
```

# Application

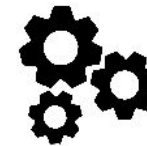
range search

Map is a typical application of Geo-Spatial search



query

select shops where distance < 1000



Map-reduce for  
quick range  
search and sort

Shop 1: distance = 100  
Shop 2: distance = 120  
Shop 3: distance = 120  
.....



# Application

Structural, semi-structural search engine

Conditioned Selection

Empty Conditions

Economic Choice

Budget: 

unlimited

 50,000yuan 100,000yuan 150,000yuan >>

Financial policy : 

unlimited

Down payment 50,000yuan

 ▼ Tenor ▼ Model preferred ▼

Maintenance cost : 

unlimited

 High to low Low to high

Price drop: 

unlimited

 Location ▼ Time ▼ High to low ▼

Oil Consumption : 

unlimited

 High to low Low to high

Model List

High to low Low to high



Brand: Magotan  
Model: 13 1.4T, Tiptronic, Comfort  
Guiding price: 199,800yuan  
Reference final prices: 178,800yuan >>



Brand: Sagitar  
Model: 14 1.6L, Tiptronic, Comfort  
Guiding price: 150,800yuan  
Reference final prices: 144,300yuan >>

Click for more >>

query with **hundreds** of properties

```
select cars from table1, table2, ..., table n
where
size > 100,
and price < 50000,
and color = white,
and performance > 100,
and maintenanceCost = low,
and city = Los Angeles,
and ( brand = BMW or Audi or Benz or Ford or Porsche )
.....
hundreds of properties
```



# Application

## Internet, Internet of Things



RDBMS works  
for old days

Simple tasks  
became **tough** in  
exponentially  
growing data  
volumn

Not  
Only SQL



# Overview

**What is LunarBase**

**Power Consumption VS Data Compression**

**Data Security**

**JSON document model**

**Exact Matching and Range(geo-spatial) Query**

**Concurrent Architecture for multi-core**

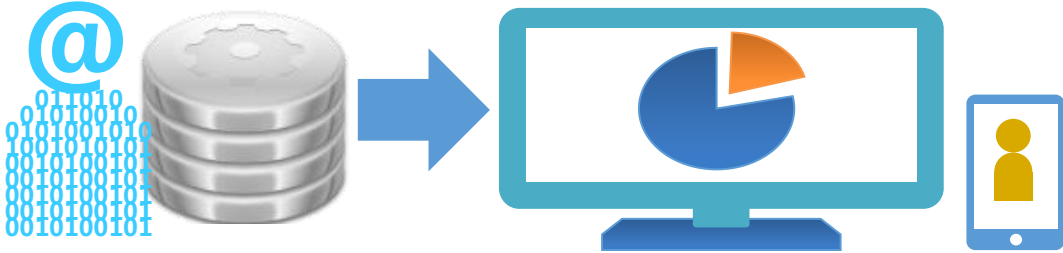
**All in One for lower TCO**

**RoadMap of LunarBase future**

# Overview

## What is LunarBase?

### Any Business Presentation



LunarBase engine targets to a document-oriented database engine, managing 2 billions records for one db instance, where each record has a size limitation to 32k bytes. By simple calculation, 64 TB data for one db can be stored and queried via LunarBase.

LunarBase includes a MMU(Memory Management Unit), administrators are enabled to configure big direct memory to cache hot data for quick access.

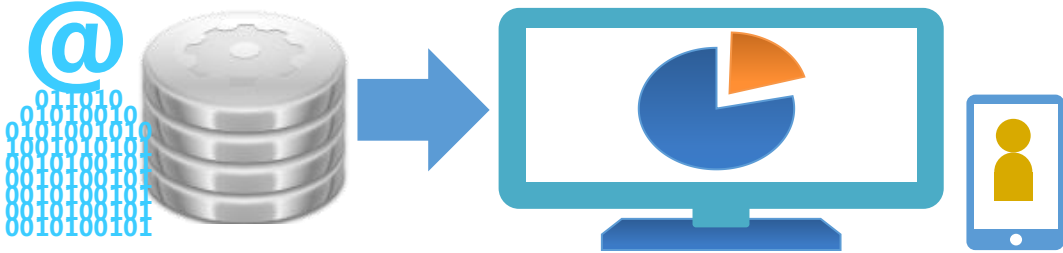
Any incoming data is persisted immediately and then LunarBase put it into cache managed by MMU as a hot data. It is reasonable since the latest data will be visited again soon till it is swapped out of the memory by LRU strategy.

For records stored, user query any property-value pair as a key for those satisfied records. You can send query like "payment=300", "payment=300 AND age=25", or something like this. All basic functionalities are maintained by LunarBase. You just insert records and query them by your keys.

# Overview

## What is LunarBase?

### Any Business Presentation



We publish LunarBase as a doc-oriented NoSQL database, not merely a key-value store or a cache solution. Since a big cache system is self-contained, LunarBase has a low deployment dependency and cost, you don't have to deploy another k-v cache for you hot data anymore.

It is:

- with a persistent DB storage
- with a self-contained big cache system, enabled concurrency, high throughput capability
- Free scalable, document-oriented without predefined schema, as all NoSQL database designed for
- Benefit from the principle of Google Big table, LunarBase has extremely fast retrieval speed
- for multiple purpose. One DB for both data storage and search, and even full text engine with simple word parsing plugins

# Overview

What LunarBase do for your business



**LunarBase as a database engine, it is able to:**

be a full text search engine, with a specified word parser;  
embed to your application as a general purpose database;  
be a standalone database server with network plugin;  
time serial data analysis;

*online Transaction*

*e-Commerce*

*CRM*

*Financial*

*Telecom*

*Retail*

*Smart Device*

*Social Media and  
network*

*CMS*

.....



single DB stores 2 billion records, each has a size limitation of 32K bytes. Hence the total size of one DB is 64TB ;

One server runs several LunarBase Instances, each manages one DB. The number of instances depends on the hardware capability ;

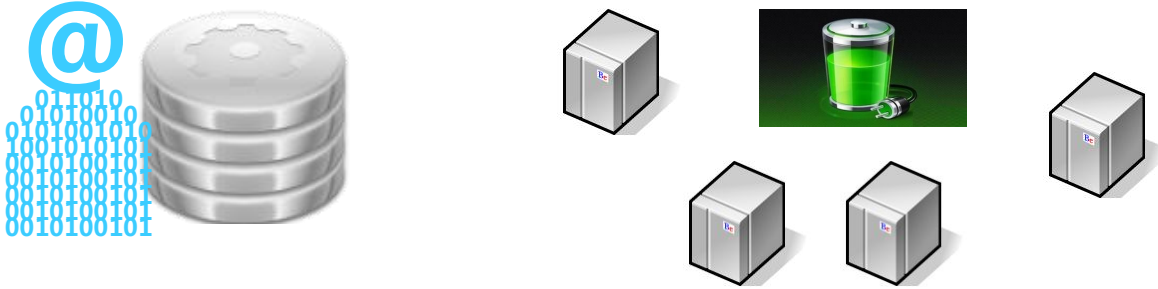
Concurrency support depends on the core of CPU. There is a one-one mapping from internal threads and CPU cores ;

Memory Consumption: LunarBase has an internal MMU, will not produce memory fragments after a long time running. This is an very important feature for any server-end middle ware;

Query speed: on a 4-core X86 server, mechanical HDD, 8G mem, LunarBase responses a query in 10 ms. If the big cache is opened and all hot data stores in it, the query speed of hot data are extremely fast.

# Overview

## Power Consumption of a Cluster



Power consumption is one of the biggest cost in running a data center. LunarBase optimized its key computation modules that are invoked frequently within the engine. These modules includes: hard disk IO, memory alloc/dealloc, FS Block management, Linear Hash storage, geo-hash computation.

The CPU workload is well managed. On the memory side, once LunarBase boots up, with correctly configured memory size, say 16GB, it is running with this constant resource, never grows. This means it does not compete resource with other services running on the same server.

LunarMMU produces no memory fragments no matter how long it is running, which means its alloc/free operation is still in constant time, will never slow down the operation system.



# Overview

## CPU power consume VS data compression



VS



Data compression is a trade off of computation resource and storage occupation. Compression rate of the data depends on the goal of the system design. For example, we need to compress the size of an index, then we compress a 4 bytes integer to a variant length one. Values less than 32K may be stored in two bytes. But the price is the deserialization time, which takes three times of CPU clocks than deserializing a fixed 4 bytes integer. Meanwhile this is a highly frequent internal invocation.

A well known fact is that it is going to be cheaper and cheaper for each GB storage, while the electric charge is going to be expensive. For running a data center, the power consumption is one of the top priorities, since records can be queried billions a day, really no need to compress integer. 1 billion integers only use 4G disk space~~

But other necessary compression is implemented by Lunarbase.



physical  
Memory

Memory  
Buckets

Slab-like  
allocation

LRU Strategy

Memory  
Locker

Concurrency  
Management

For any server-end middle ware, stability is the most important thing. A well designed memory management system can significantly reduce the memory fragments, fasten the alloc/free speed. You really do not want your application alloc a block of memory every time from a busy OS, since it will leads to a failure allocation, proved to be fatal for any application. LunarMMU is designed for such purpose:

- 1) constant time of alloc and free
- 2) has an upper bound of memory consumption
- 3) produce no memory fragments, even after billions of alloc/free

LunarMMU is not a memory system for general purpose. It is for LunarBase and cache only. There is a price payed for the above three advantages. The memory occupation will be little more bigger than that is really needed.

But as the plummet of hard disk price, so is the memory price. This trade off is much worthy.

# Overview

## IO for Linux X86 platform

Memory Mapped File is the most fast mechanism for read and write. Data from user space to the kernel space and then hard disk persistent takes just one copy.

MMAP system call exposed by linux kernel, and it is the only solution for a quick IO procedure. LunarBase kernel invokes MAP for IO, provided in libLunar\_Linux\_X86\_64.so

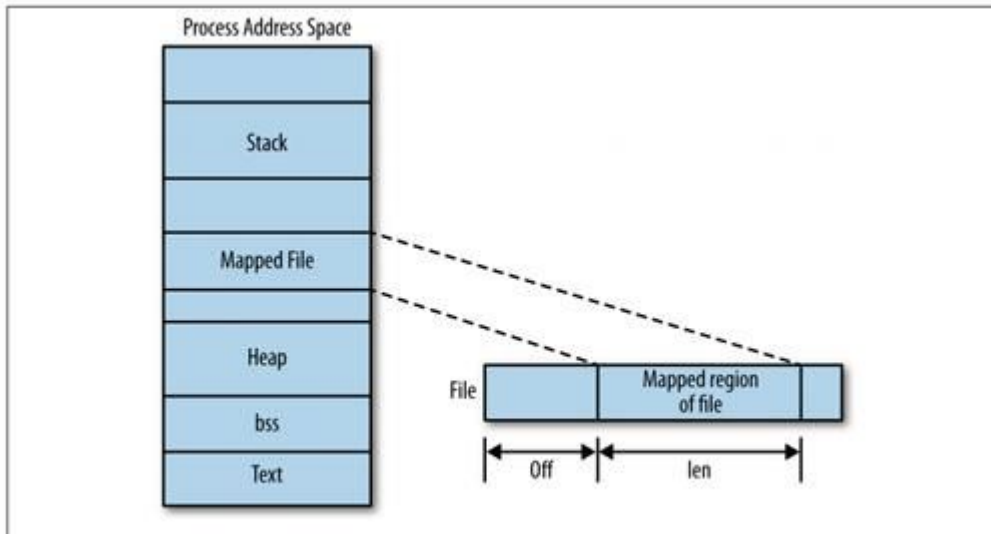


Figure 4-1. Mapping a file into a process's address space

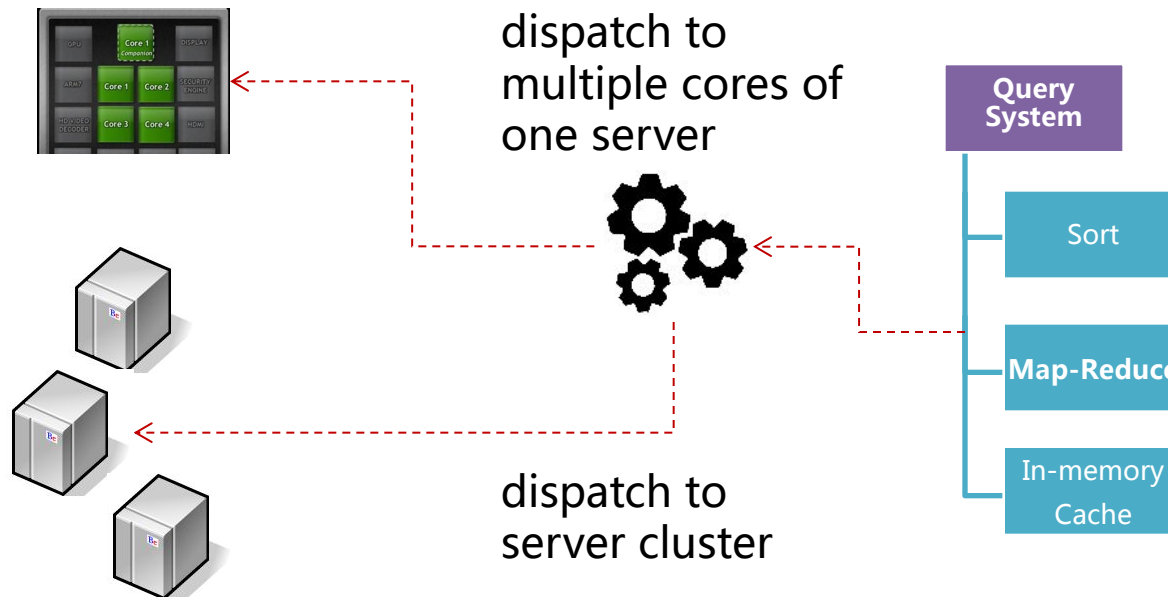
*this picture is from internet for illustration purpose.  
Any violation of copyright please let me know.*

# Overview

## Multi-Core performance

**LunarBase** writes in one thread and reads with multiple threads. Multi-threaded write does no good to the performance for that eventually we flush data into the hard driver, there is a queue waiting; With the big memory solution LunarMMU, the multi-threaded reading is possible, since hot-data are all in memory waiting. Cold data will be swapped out by LRU strategy.

LunarBase engine is an nosql database engine written in java, but its basic IO , file system ( Lunar virtual FS), Memory management unit(MMU) are implemented in C++. Performance is the only reason for our C++ implementation.



# Overview

## Data Security



➤ Log system is first of all we implemented, every command user commits is recorded before execute real database operation:

**succeed@** **insert:** {name=jack, age=30, payment=500, date=20150728};

**succeed@** **insert:** {name=michal, age=25, payment=800, date=20150728};

**succeed@** **insert:** {name=jackson, age=45, degree=master, address=somewhere, payment=2000};

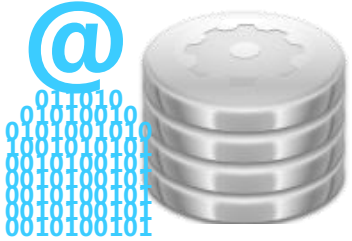
.....

**WAL:** Write a Log first  
when a write committed.

*If any error (system crash, electricity failure...) occurs during writting, LunarBase will check Log first, all the failed commands will be excuted again to make sure the data is succesfully stored;*

*Log system is also the fundamental component for Data migration and backup. LunarBase exposes several API to support these functionalities. User need only to tell LunarBase where the log directory is.*

# Overview



JSON input



**name, age, payment, date** and all these alike are **Properties**, programmers need no further work to tell Lunarbase which one is the key explicitly. Any Property with its value is a key, and searchable with equal performance.

Json document is the primitive unit for data manipulation. Quite like other NoSQL solutions, but not exactly the same.

## DB Insert:

➤ records in simple JSON format:

```
{name=jack, age=30, payment=500, date=20150728};
```

```
{name=michal, age=25, payment=800, date=20150728};
```

```
{name=frank, age=25, payment=1200, date=20150729};
```

```
{name=jackson, age=45, degree=master, address=somewhere, payment=2000};
```

.....

Billions of records in one DB

# Overview

## 水平扩展



➤ Schema Free, all properties is added on demand. As what the example illustrates, name ,age, payment, date appears when new records inserted. In the future, the comming data may includes new properties like address, school, LunarBase recognizes these, and store them automatically:

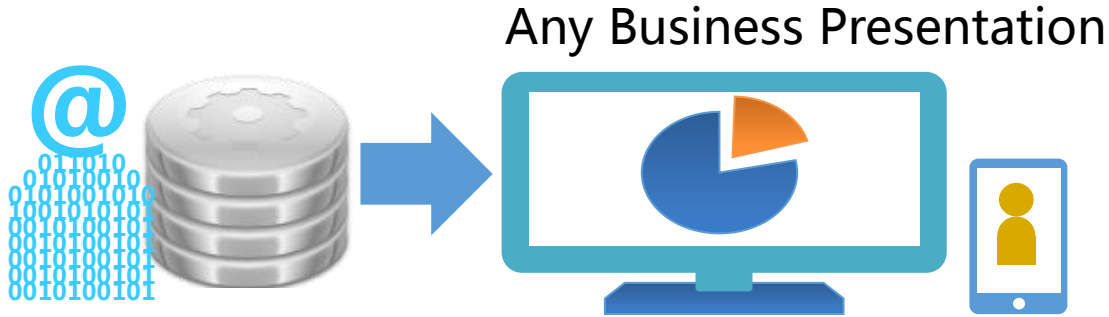
**property@** name, age, payment, date, degree, favorite.....

*Properties of a Data Application is the very first thing need to be clear when start to design and program in old days. But now, the data types variants much more frequently than before, no one has the capability to know the future category a bunch of data belongs to.*

*This is what the industry calls **Scalability** and **Flexibility**, an advantage of NoSQL database.*

# Overview

## Exact Matching and Range Query



We query **any property** with any value, if exists. Never need to specify which one is the key. Simple and Happy programming

### DB Query:

➤ **select records where age=25**, LunarBase returns: ○

docID = 1, Value = {name=michal, **age=25**, payment=800, date=20150728};

docID = 2, Value = {name=frank, **age=25**, payment=1200, date=20150729};

.....

Millions of records matching the query in DB

➤ with Geo-Spatial index, LunarBase supports range search as: **select records where payment > 500 and payment < 2500:**

docID = 1, Value = {name=michal, age=25, **payment**=800, date=20150728};

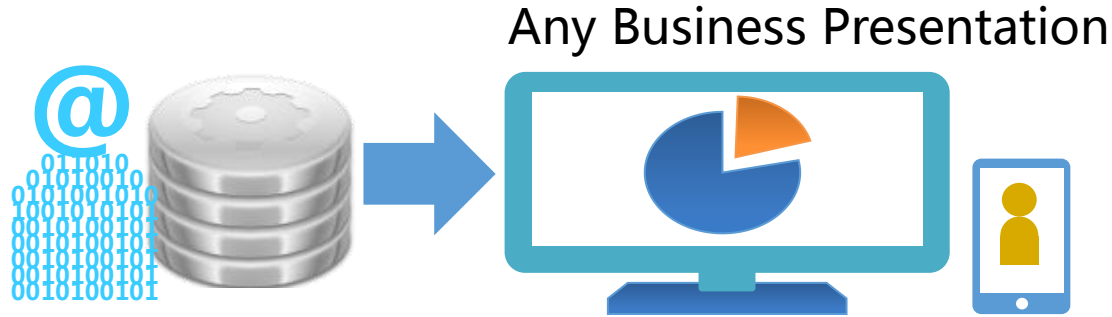
docID = 2, Value = {name=frank, age=25, **payment**=1200, date=20150729};

docID = 3, Value = {name=jackson, age=45, degree=master, address=somewhere, **payment**=2000};



# Overview

## Exact Matching and Range Query



We query **any property** with any value, if exists. Never need to specify which one is the key. Simple and Happy programming

### DB Query:

➤ **select records where age=25**, LunarBase returns: ○

docID = 1, Value = {name=michal, **age=25**, payment=800, date=20150728};

docID = 2, Value = {name=frank, **age=25**, payment=1200, date=20150729};

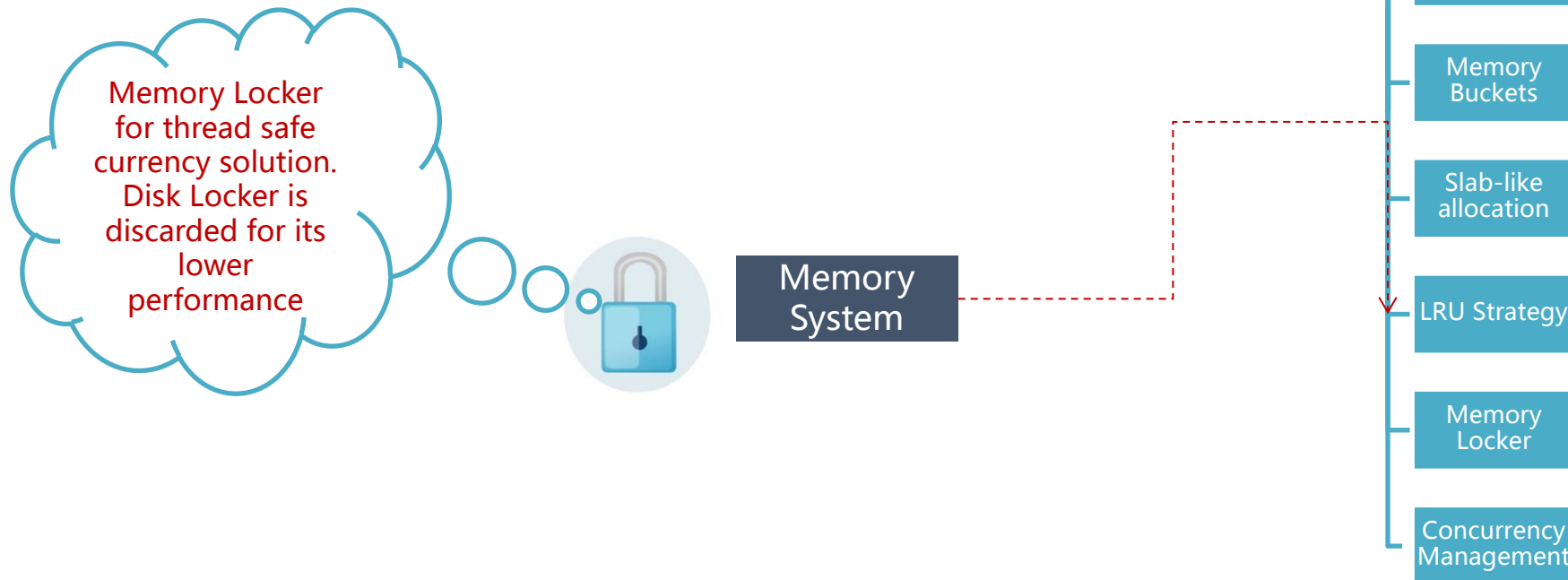
.....

Millions of records matching the query in DB

The query language is quite simple. Under the package of (LunarBaseTutorial)LCG.DBAPI, you will find a bunch of implementation of how to query. In practice, you need to implement your handler(this what we used special term for internal Event-Driven framework) to deal with query results like above. you may order them by the date, payment, or age. Check out how to do it under the package LCG.DB.EventHandler

**LunarBase = persistent storage system + big memory management system for cache:**

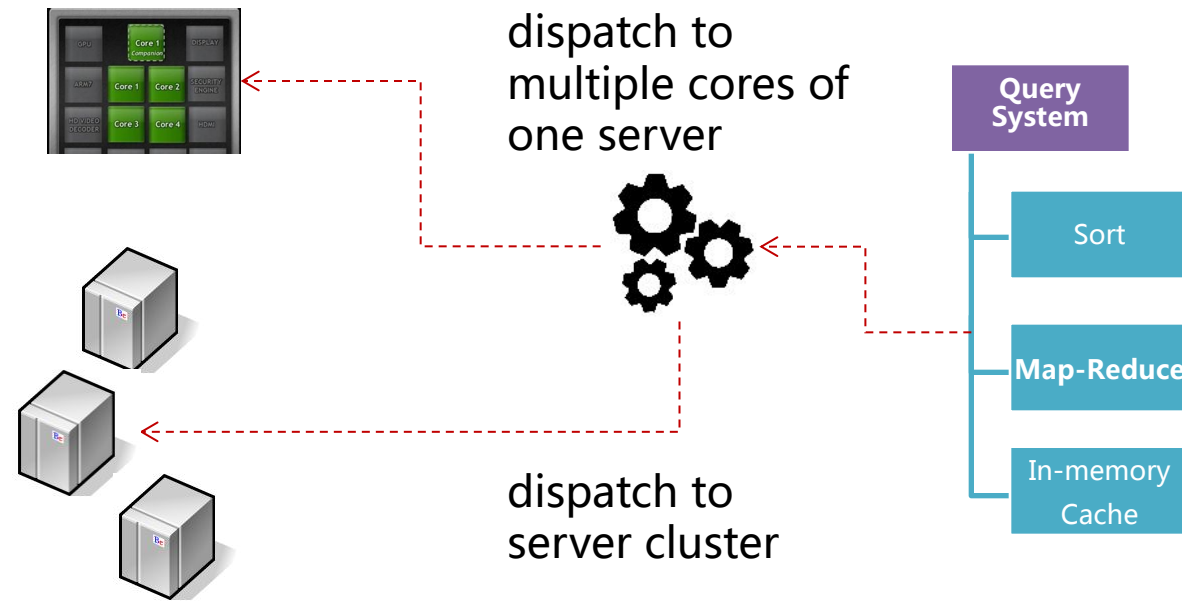
big memory system allocate memory for db usage, and implements concurrent model dealing with data requests from millions of clients as well.



# Overview

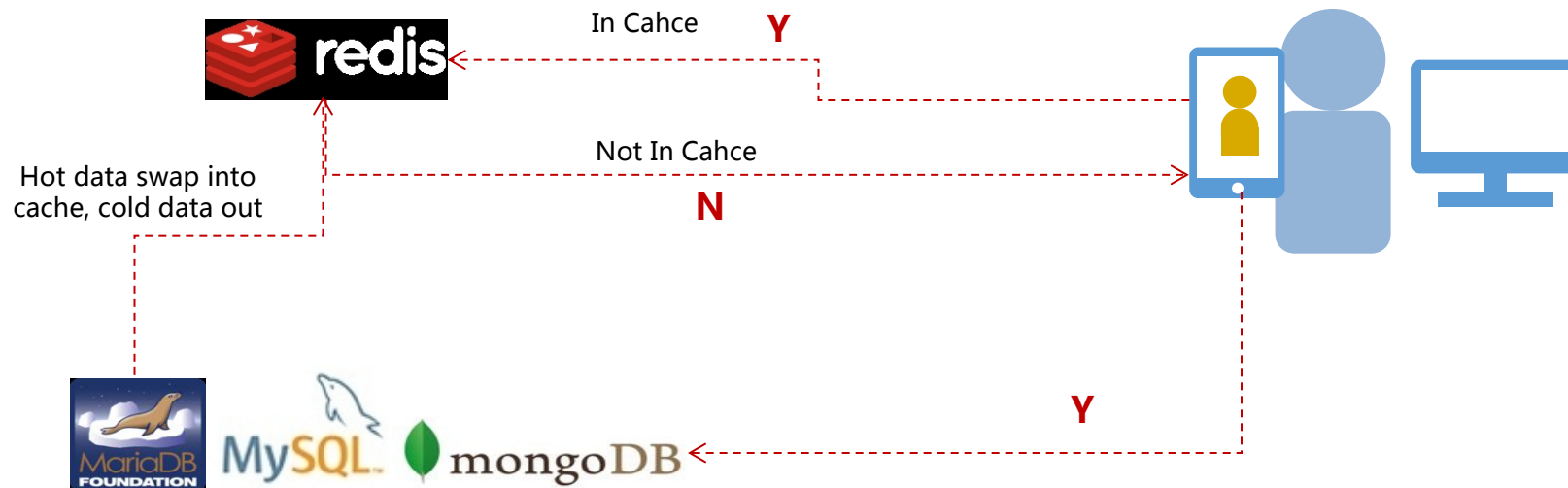
## Concurrent Architecture for multi-core

Map-Reduce is implemented inside LunarBase. Tasks are automatically dispatched to multi-core or several server by task-center.



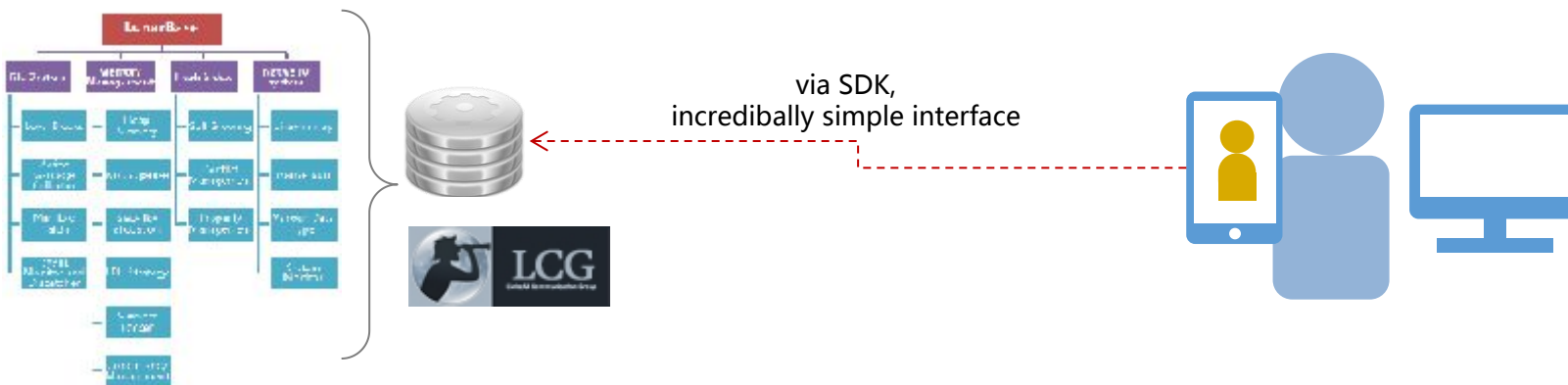
**For high performance, deploy cache systems for a persistent DB storage** is a common solution for various on-line applications. The following combinations are quite familiar:

1. k-v cache like memcached or redis + storage like mongo (or RMDBS like mysql, mariadb ).
2. cluster of memory db + duplicated backup. One fails, another image take place. Discard the persistent solution.



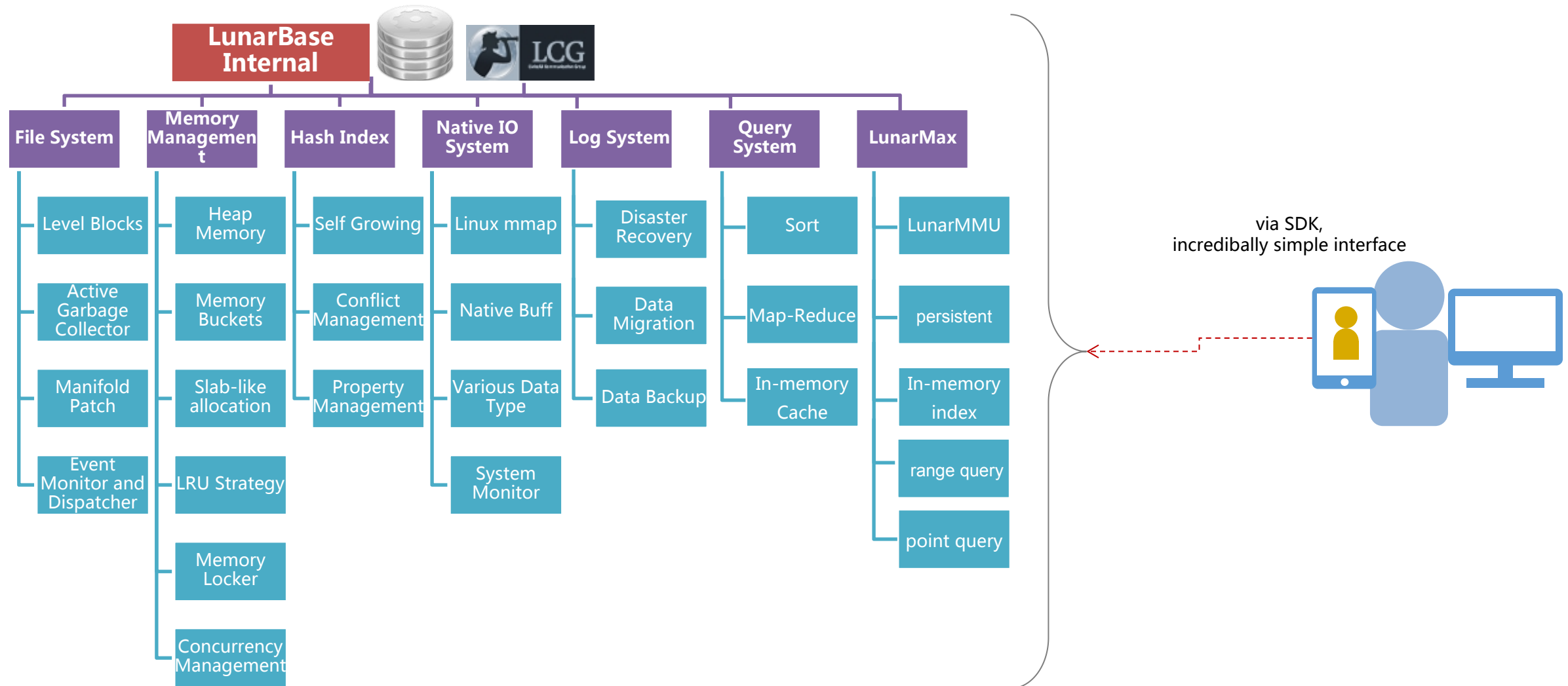
# All in One

1. Automatically manage hot and cold data, greatly simplify application development .
2. Deployment and maintainent became incredibly terse.
3. Therefor the cost decreases.



# Overview

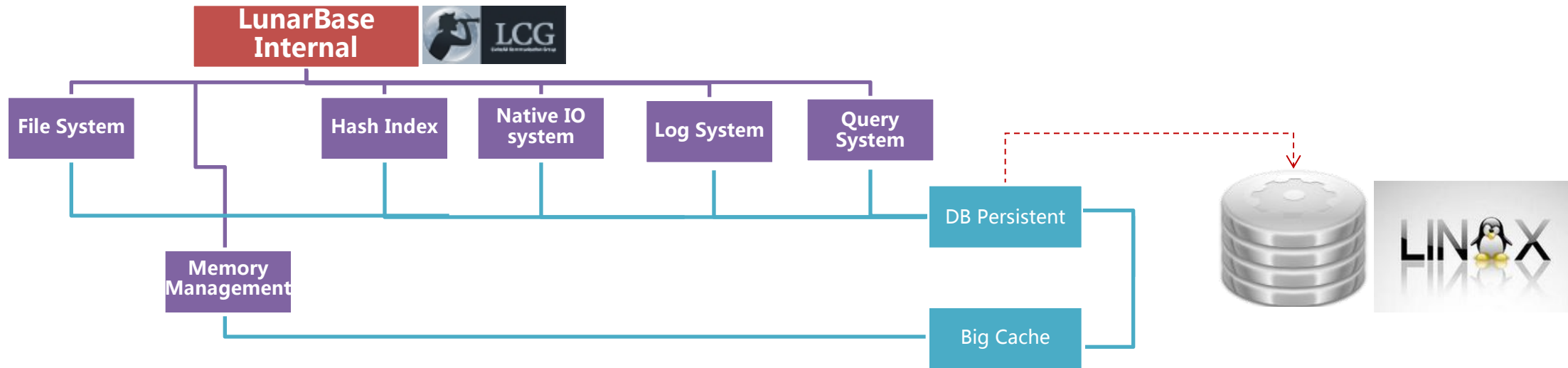
All in One



# Overview

All in one

**LunarBase** simplifies deployment



**LunarBase has the duty to reduce the complexity of database server environment, therefor to reduce the TCO (Total Cost of Ownership) for any company. The dependency list is short:**

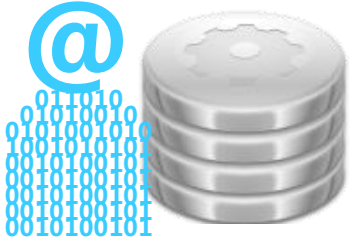
1. OS: Linux on X86 hardware
2. JVM: IBM or Oracle
3. Network Communication Framwork (server version only): Netty





# Overview

All in one



**Embedded** version is published as: jar packages, together with native C/C++ libraries and a bunch of command tools.

**Server** version is published as a DB server, equipped with a client driver for interacting with the LunarBase server. At present, java is the only language supported. Other languages, php, scala, erlang, etc. are in developing.

### **LunarBase is going to be :**

#### 1.0.0 Beta:

geo-spatial search, range search, Auto-Sharding, data migration, disaster recovery

#### 2.0.0 Alpha:

LunarBase Server: running as an independent server, programmers operate LunarBase via a client driver (java driver), simple SQL language support, Interpreter for complex math formular in analysis

#### 3.0.0 Alpha:

LunarBase cluster, nodes management, automatic backup, migration, load balance, disaster recovery among nodes, HA monitor, Cloud Platform based on LunarBase

#### 4.0.0 Alpha:

Cluster management and Monitor system, UI to lower the management cost

#### 5.0.0 Alpha:

Support real time big data processing, this functionality may appear before version 5, Cloud Platform Commercialize

# Architecture

**Blueprint**

**Log System**

**Hash Storage**

**Slab-like Memory Management System**

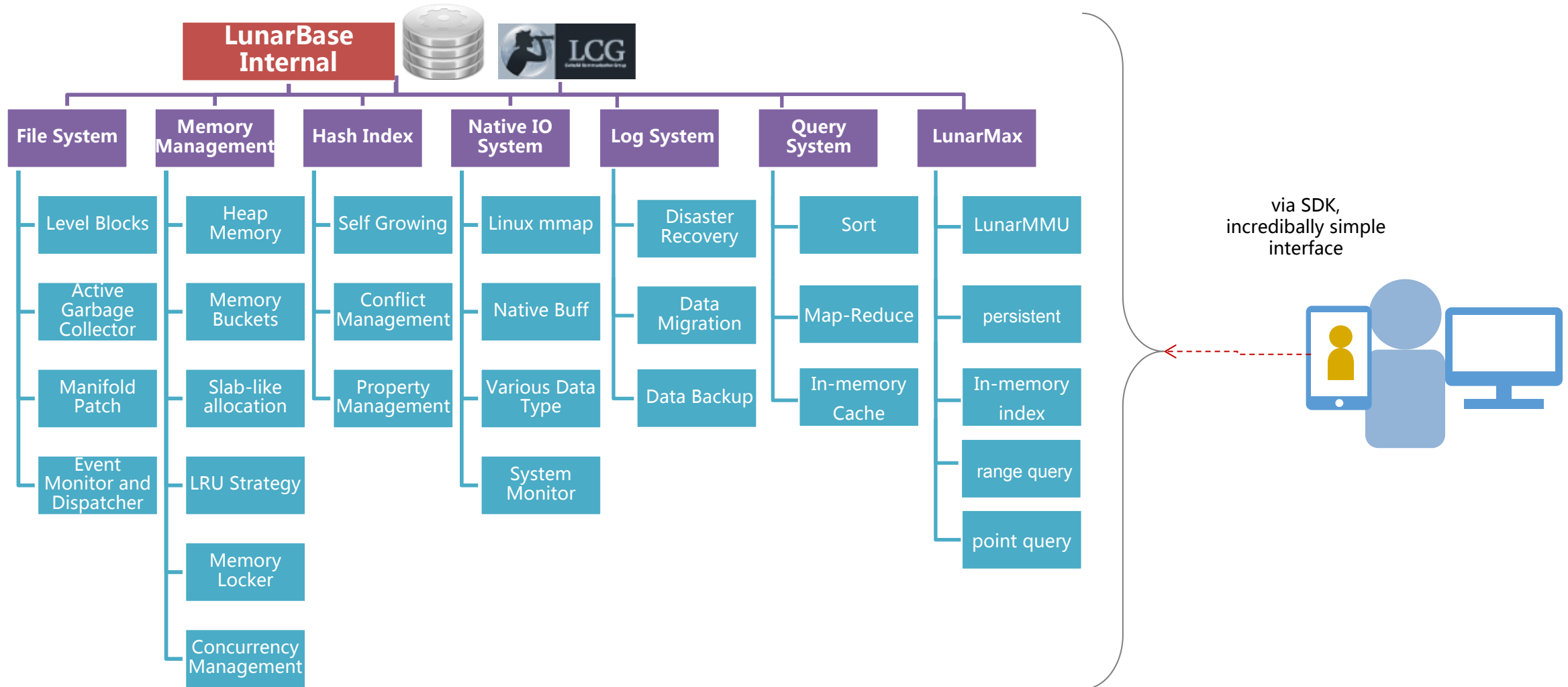
**LunarMax for Real time analysis**

**Lunar File System**

**Active Block Garbo for SSD**

# Architecture

## Blueprint



# Architecture

## Blueprint



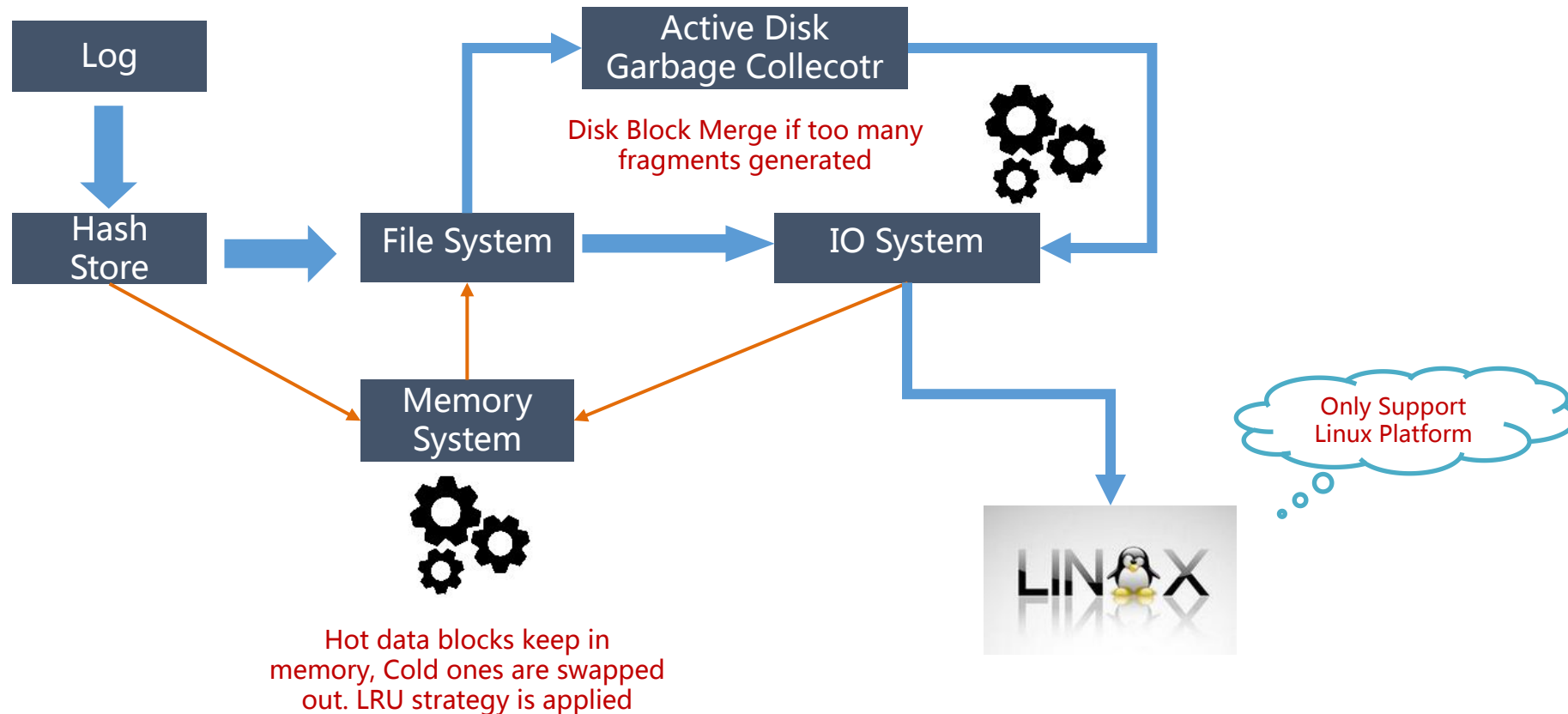
➤ records in simple JSON format:

```
{name=jack, age=30, payment=500, date=20150728};
```

```
{name=michal, age=25, payment=800, date=20150728};
```

```
{name=jackson, age=45, degree=master, address=somewhere, payment=2000};
```

.....



# Architecture

## Blueprint



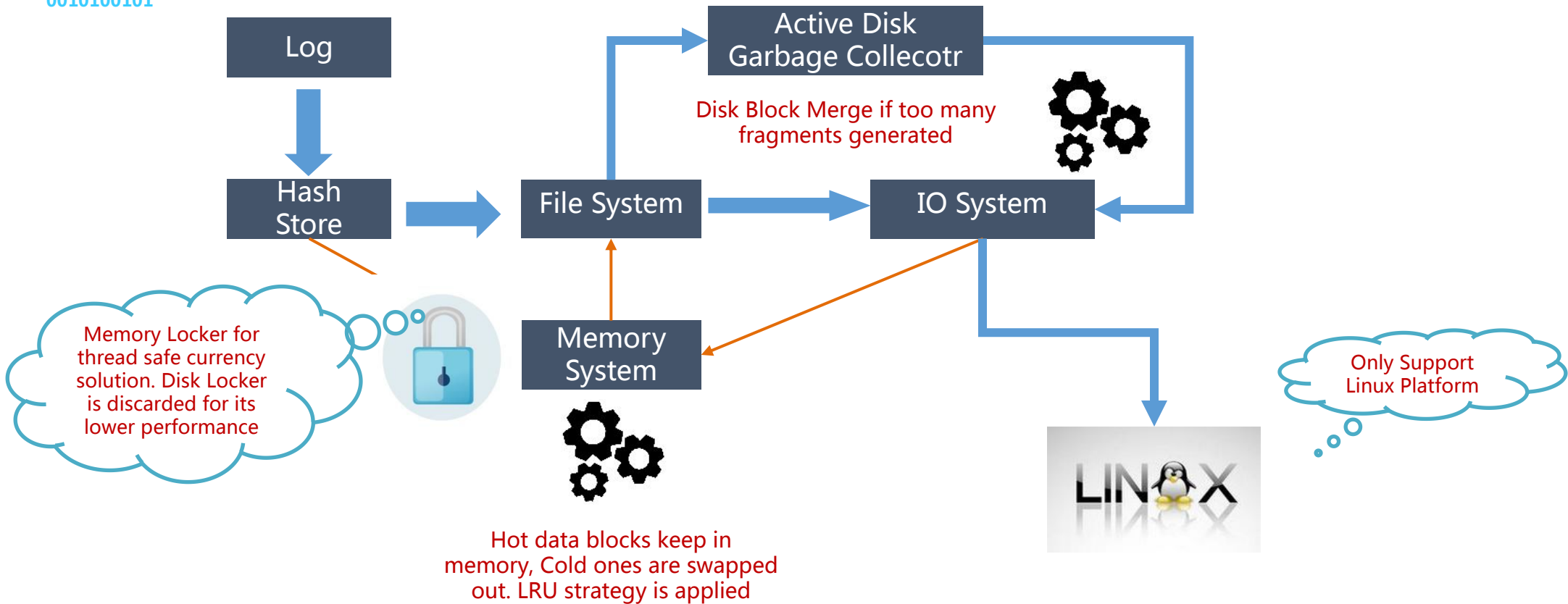
➤ records in simple JSON format:

```
{name=jack, age=30, payment=500, date=20150728};
```

```
{name=michal, age=25, payment=800, date=20150728};
```

```
{name=jackson, age=45, degree=master, address=somewhere, payment=2000};
```

.....



# Architecture

## Log system



➤ Log system is first of all we implemented, every command user commits is recorded before execute real database operation:

**succeed@ insert:** {name=jack, age=30, payment=500, date=20150728};  
**succeed@ insert:** {name=michal, age=25, payment=800, date=20150728};  
**succeed@ insert:** {name=jackson, age=45, degree=master, address=somewhere, payment=2000};  
.....

it is the key component for:  
*disaster recovery,*  
*data migration,*  
*data duplication,*  
*consistent check*  
.....



The virtual file system defines levels of blocks, which is configurable to db administrators:

Level 0: block size 64 bytes

Level 1: block size 512 bytes

Level 2: block size 4096 bytes = 4K

Level 3: block size 32K

Level 4: block size 256K

Level 5: block size 2048K = 2M

Level 6: block size 16M

Total 7 levels are supported. Objects acquire disk spaces from Lunar VFS, and Lunar VFS decides how many blocks is needed and how much of each size. As an example, we assume a requirement of 5K space, Lunar VFS follows a certain rule and allocates:

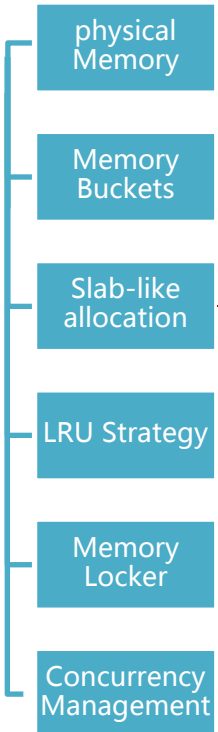
1 4K block + 2 512-bytes blocks.

Of course, they are chained if they belong to one object

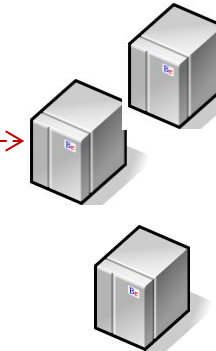


# LunarMax

Manage your billions of small objects



Java claims it can do this for programmers, then we have GC. But in most of cases, in mission critical tasks, GC stops the world. In 64-bit time, a cheap server has several billions of memory, but jvm can only provide you small portion of it. That's a bit waste.



# LunarMax

## In-process off heap storage



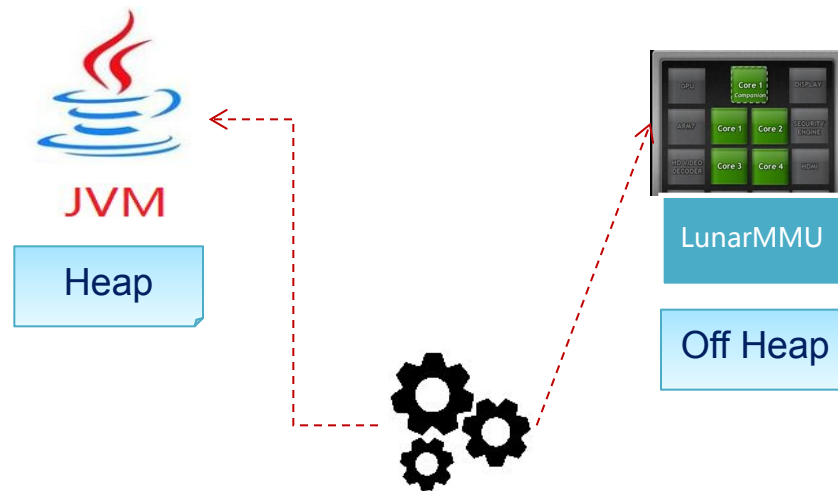
Control off-heap memory via java interfaces in LunarMMU.jar

10~100X hot data on a single server, fully digging the potential of one machine.

cost effective scaling - 1000X faster than disk.

LunarMax supports 2 types of memory:

1. small objects with short lift cycle.
2. Big memory disk that stores data for a long time running.



# Thank You



Lunarion Consultant Group, Co., LTD.