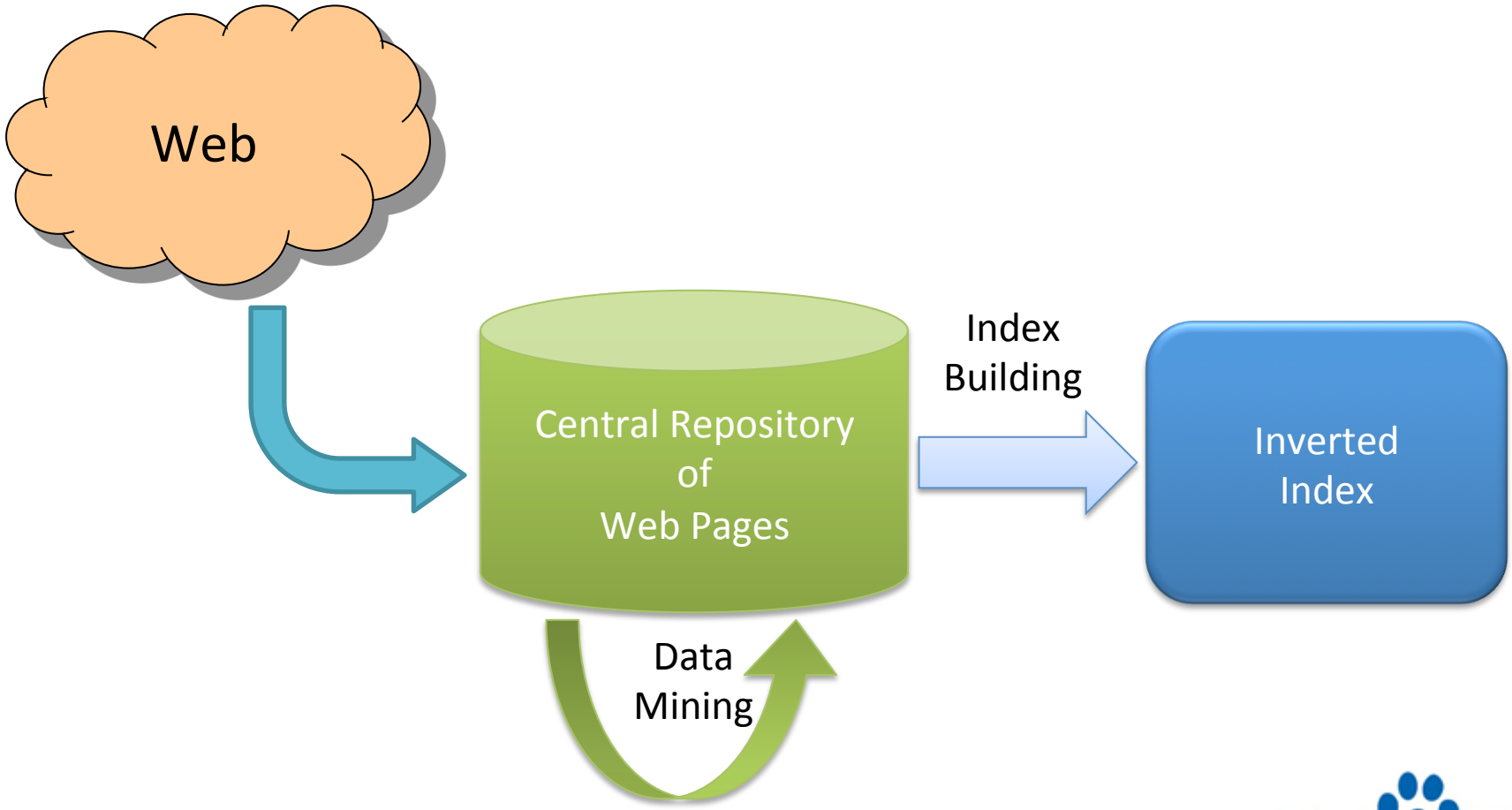
Building Large-Scale Storage Systems: Practices and Experiences

Shiding Lin

LADIS, 2013-11-3

Let's Start from the Search Engine…



The Requirements of the Storage (in 2008)

- Tens of billions of pages
  - 1 page is ~10KB
  - ~100 attributes
- Access pattern (daily)
  - Billions of writes
  - Read all (to generate inverted index and do mining)
  - Update all (to refresh the attributes)
  - Sampling
- Easy to understand and maintain for Ops
  - Fit in current operating process

Baidu 百度

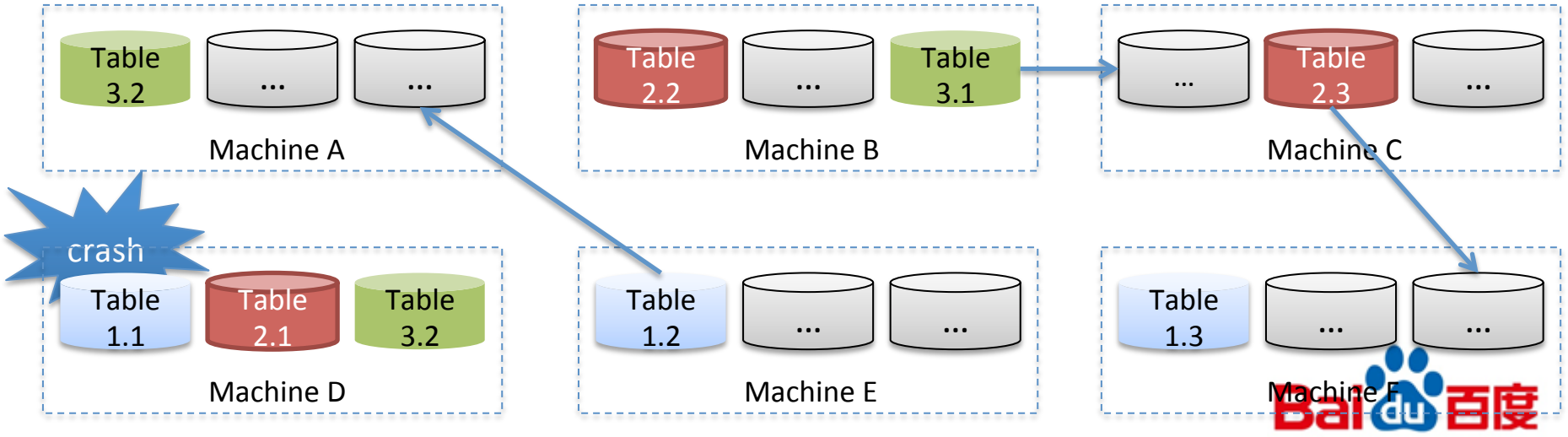
Requirements Drilled Down

- Structured storage
  - Table
  - Flat layout, bit support
  - Insert, update, scan, query, range_query
- First goal: high throughput
  - Optimized for scan
  - Support continuous reads and writes
  - Cost efficient
- Scale: ~1PB to 10 PB
  - Moderately scalable

Baidu 百度

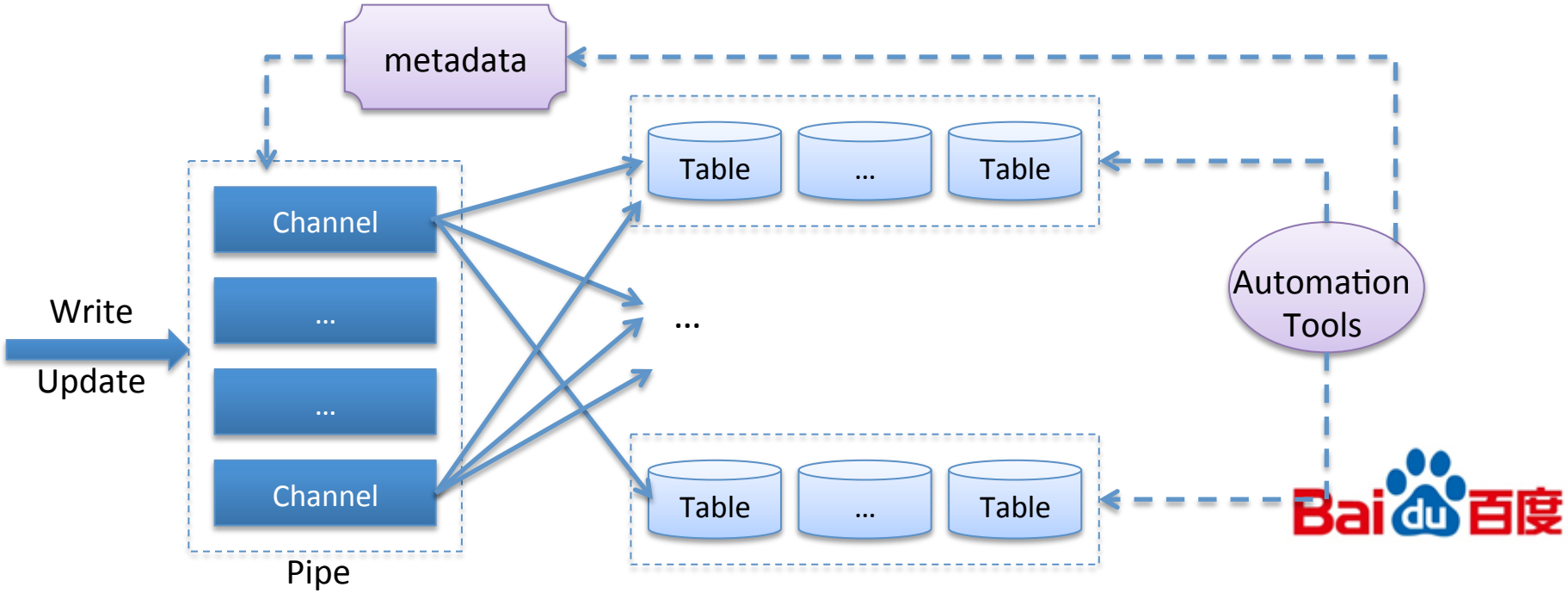
Our Design Choices

- Replicated by tables
  - Table is autonomous, manages local data, knows nothing about its replica
- Pre-defined partition scheme
  - No re-balancing, no re-partitioning
  - Just do re-mapping (from table to machine)



Our Design Choices (cont.)

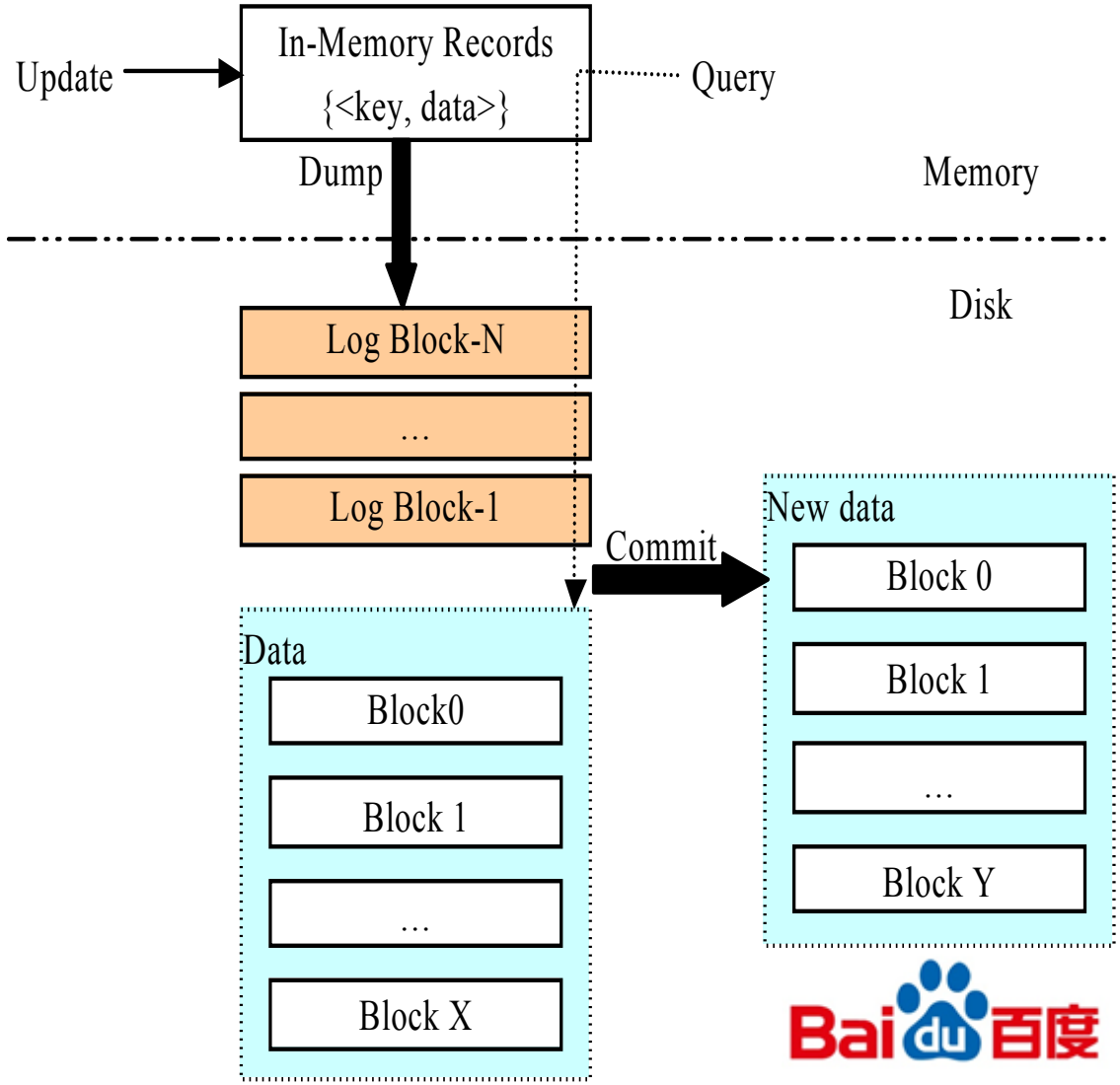
- Make the data flow management explicit
  - Decouple the data flow (buffer) from sink (table)
  - A management layer adjusts the data flow
    - Soft-state metadata service + automation tools (repairing, migration)
  - Provide enough flexibility for substantial structure change
    - Software upgrade, network management, datacenter migration



To Build a High-Throughput Table

Log-based Structure

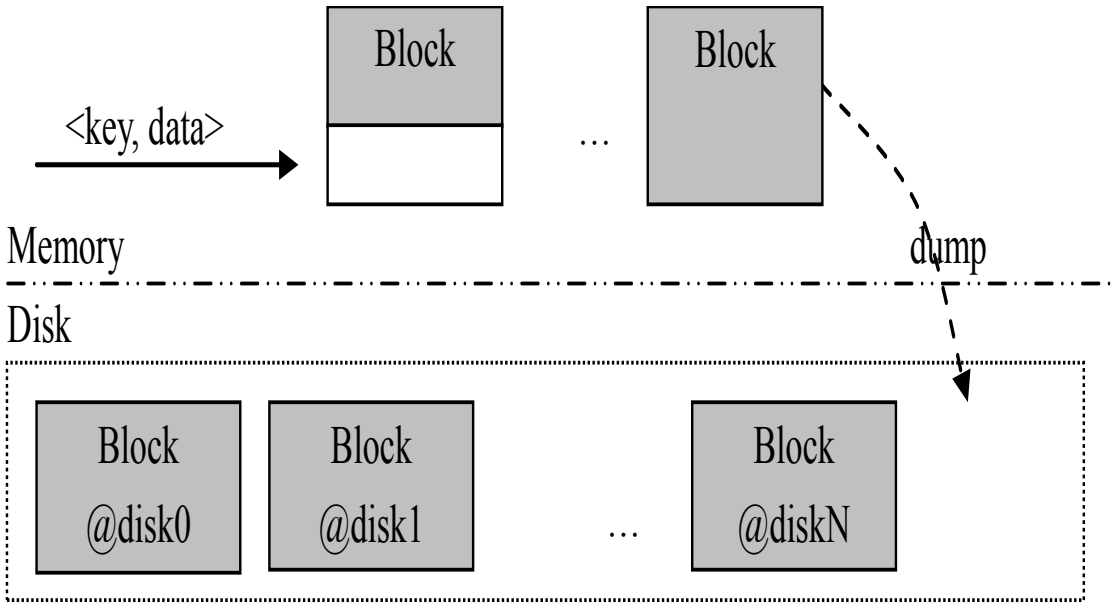
- Block I/O, fixed size
- Read-only blocks,
  - no in-place update
- Batch commit
- Stream R/W



Optimization for Write Throughput

Maximize Parallelism

No RAID, raw disk
Direct I/O, No FS
Multiple write buffers
Disk scheduling
Aggregated B/W: 1GB/s



A Big Virtual Stream by Blocks

Metadata

- Associate blocks to streams (log & data)
- To improve performance (latency & throughput)
  - Managed in memory, periodic dump
- Need to guarantee that the referred blocks be persisted if the metadata is persisted
  - Blocks are dumped asynchronously, bypassing buffer
  - Transactional status update of the blocks
  - Fast recovery from crash

Baidu 百度

Availability Discussion

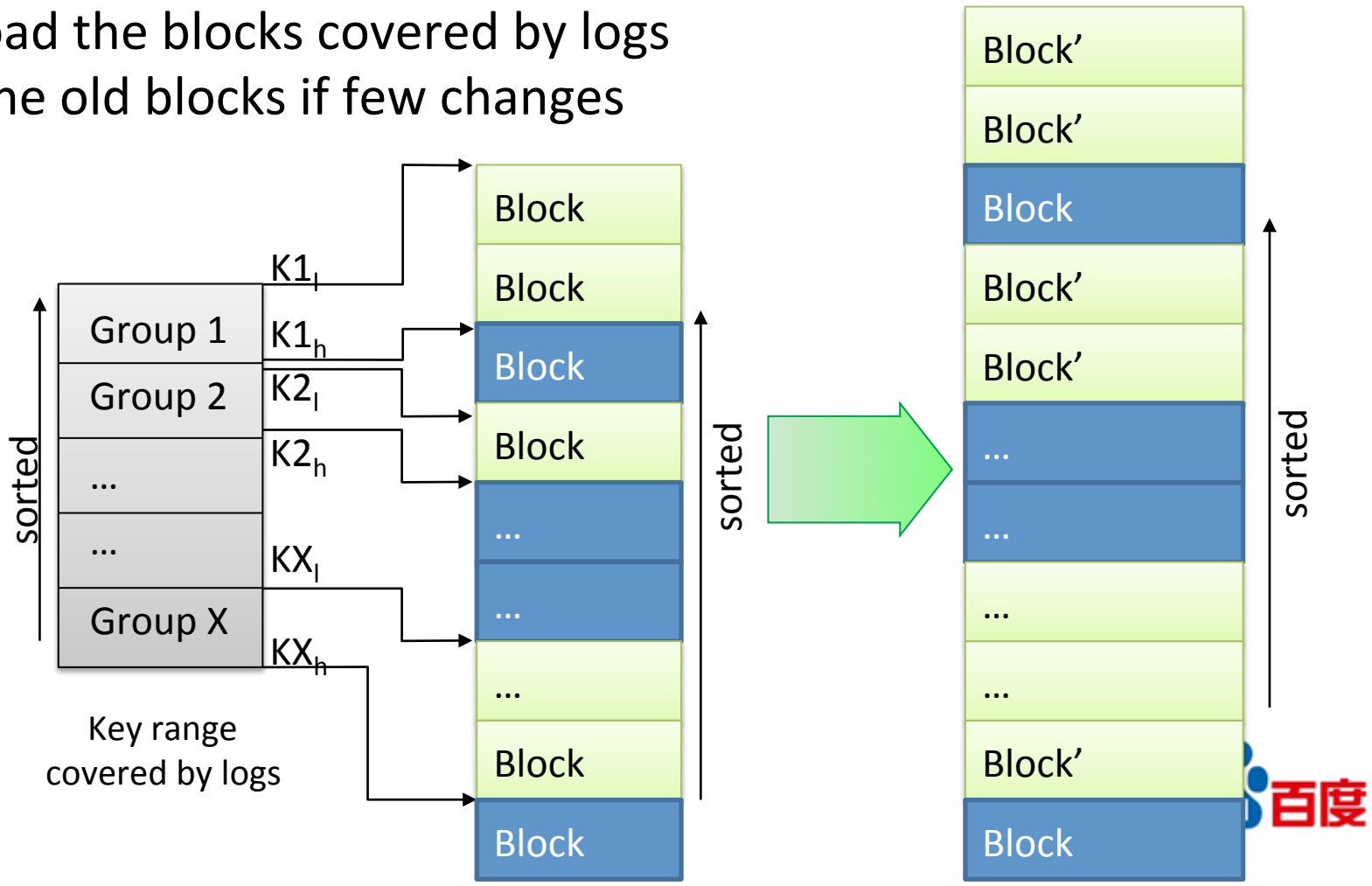
- 89% hardware failures is caused by HDD
  - Transient failure > sector error > disk error
- 3-way memory replication is enough for most applications
  - Significantly improve both latency and throughput
- Fast recovery for a replica is more important
  - Can be killed at any time
  - Become useful as repair destination or migration source

Baidu 百度

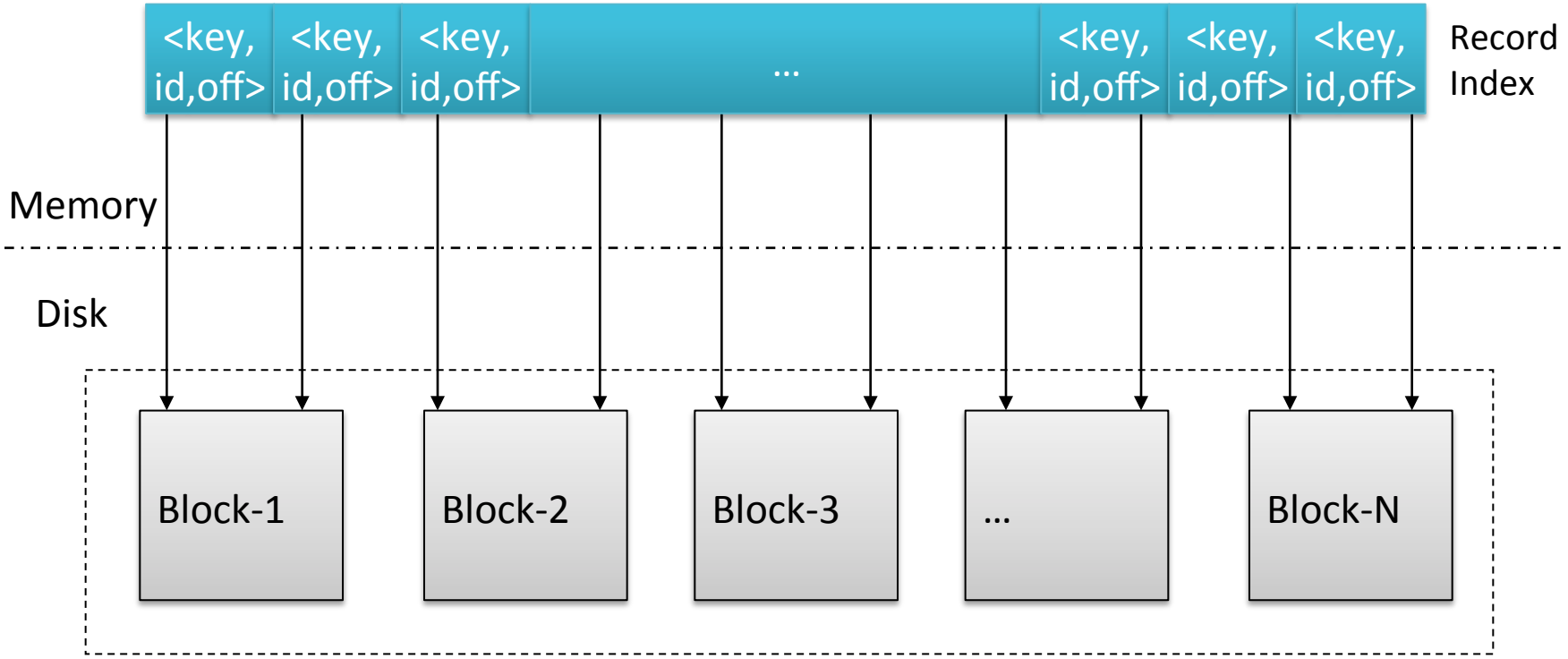
Optimization for Commit

Reduce unnecessary I/O

Represent the sorted data by key range
Only load the blocks covered by logs
Keep the old blocks if few changes



Indexing: to Support Random Query



#Rec: 2 Billion

#Index: 2 Billion * 16 Bytes = 32GB

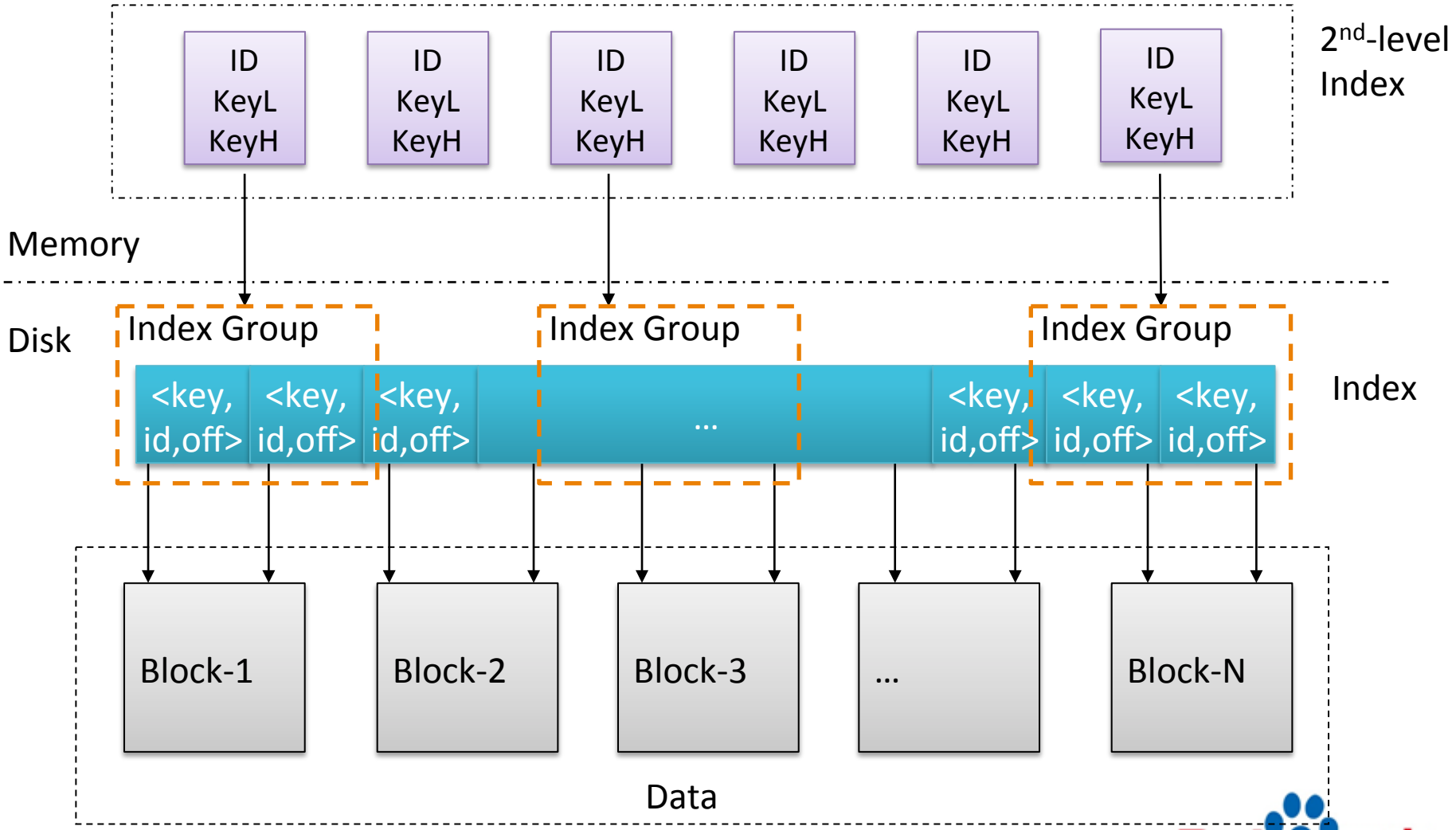
Baidu 百度

Problems

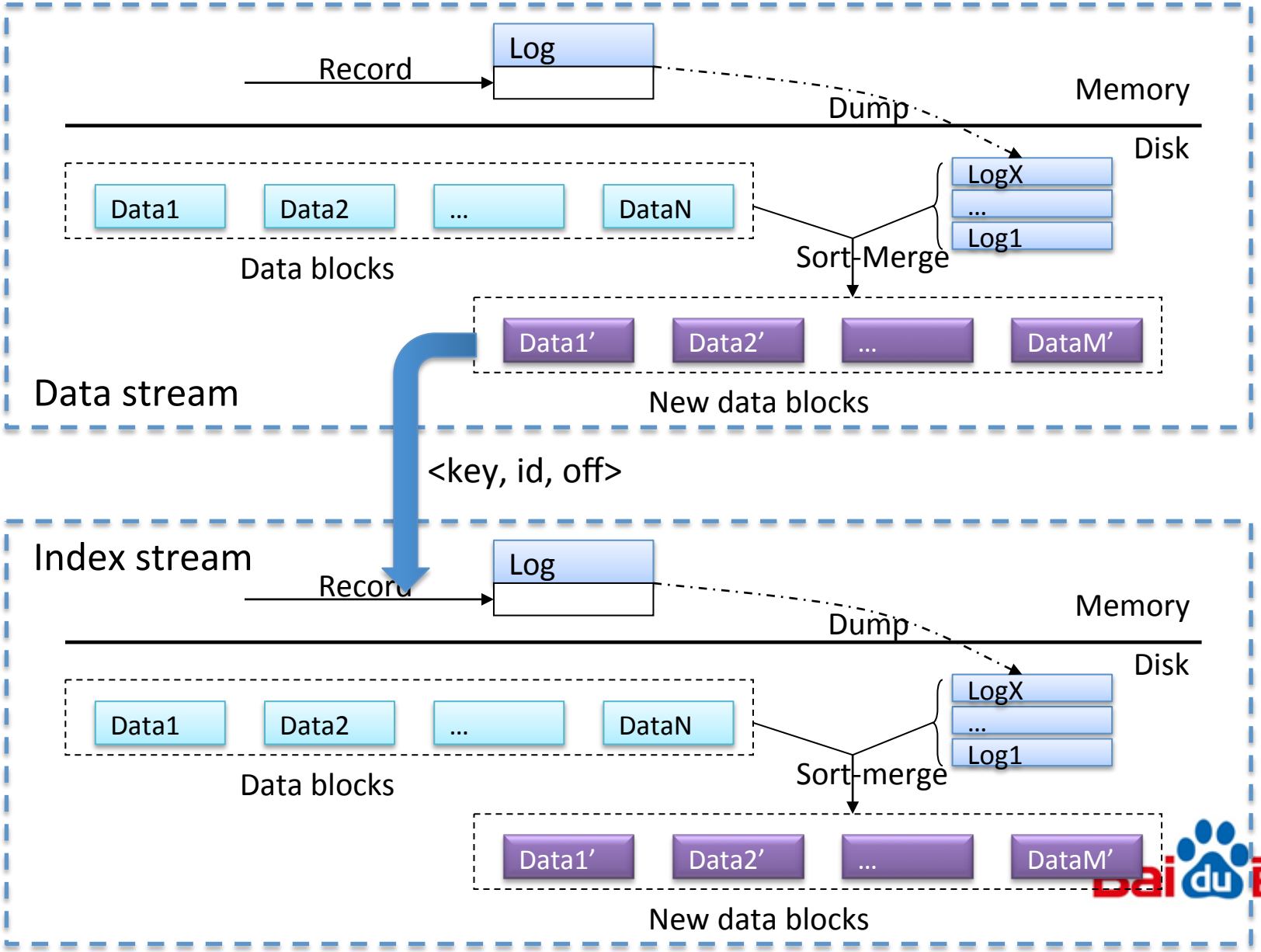
- Data have to be sorted, which limits write throughput
- Indices cost too much memory

Baidu 百度

Multi-level Indexing



How to Generate Index?



Some Observations, 1

- Table = data + index
  - Data-commit triggers index-commit
- 2 random reads to get the record
  - 1 for index_group, 1 for the record

- Index can be managed as the data
- Reuse Sort-Merge to generate index
- Data don’t need to be sorted

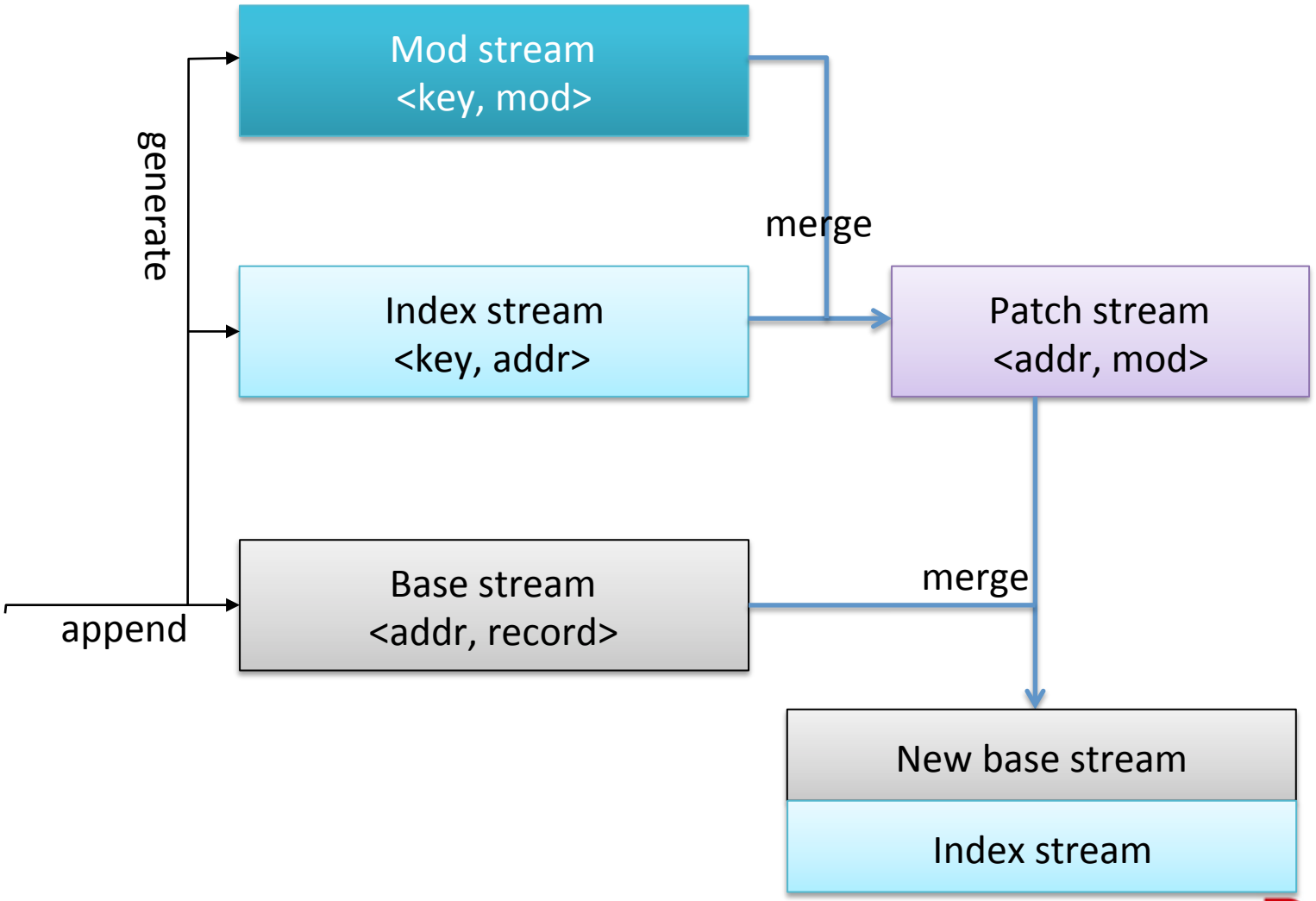
Baidu 百度

If Data is not Sorted…

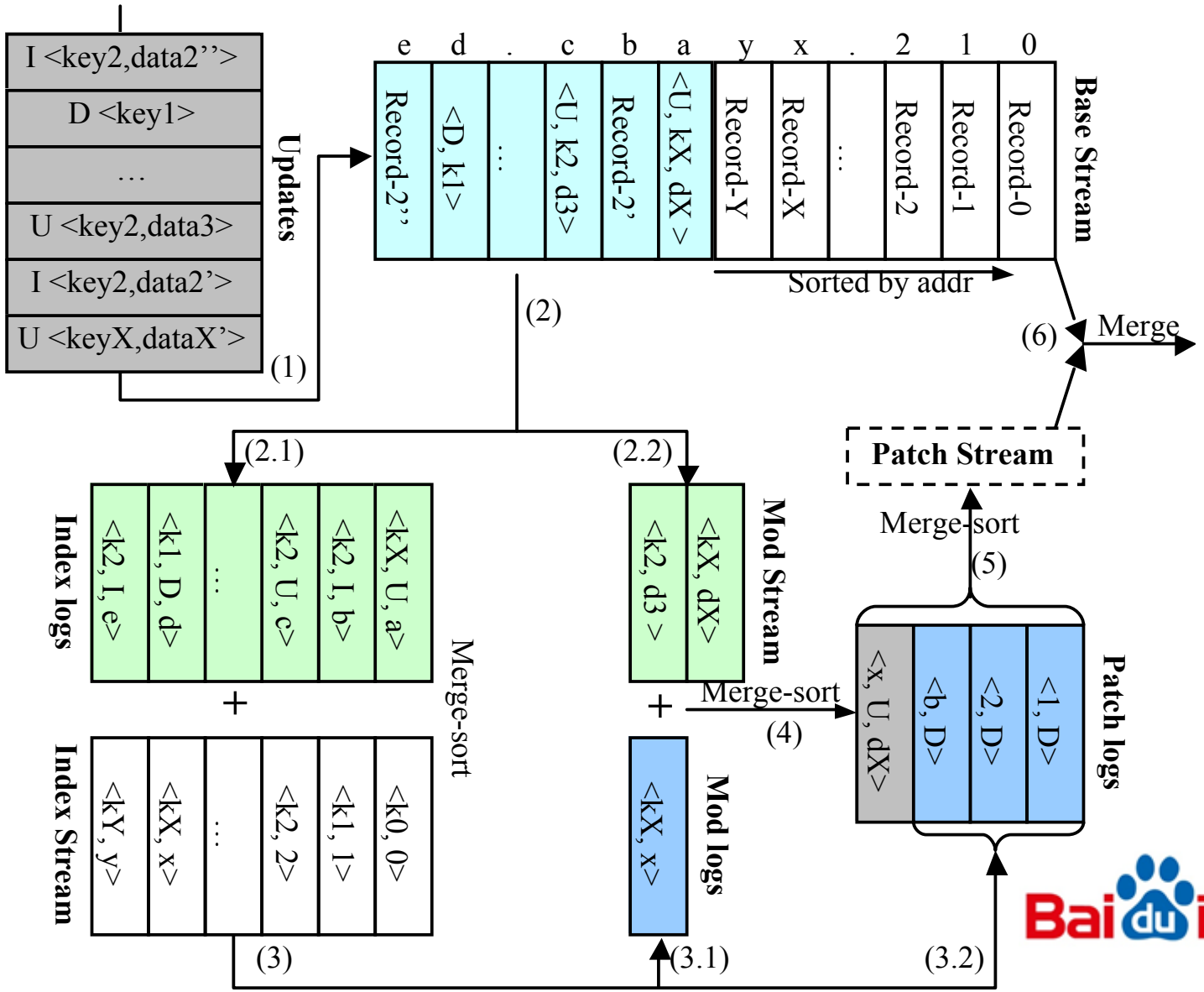
- Insert
  - New? Index: <key, addr>
  - Existed? Update index and delete the old one
- Update
  - Read the old one
  - Incorporate changes and generate a new one
  - Update index and delete the old one
- Data: <addr, record>
- Patch: <addr, changes>

Baidu 百度

Use Affiliate Data to Generate Patch



More Detailed Process



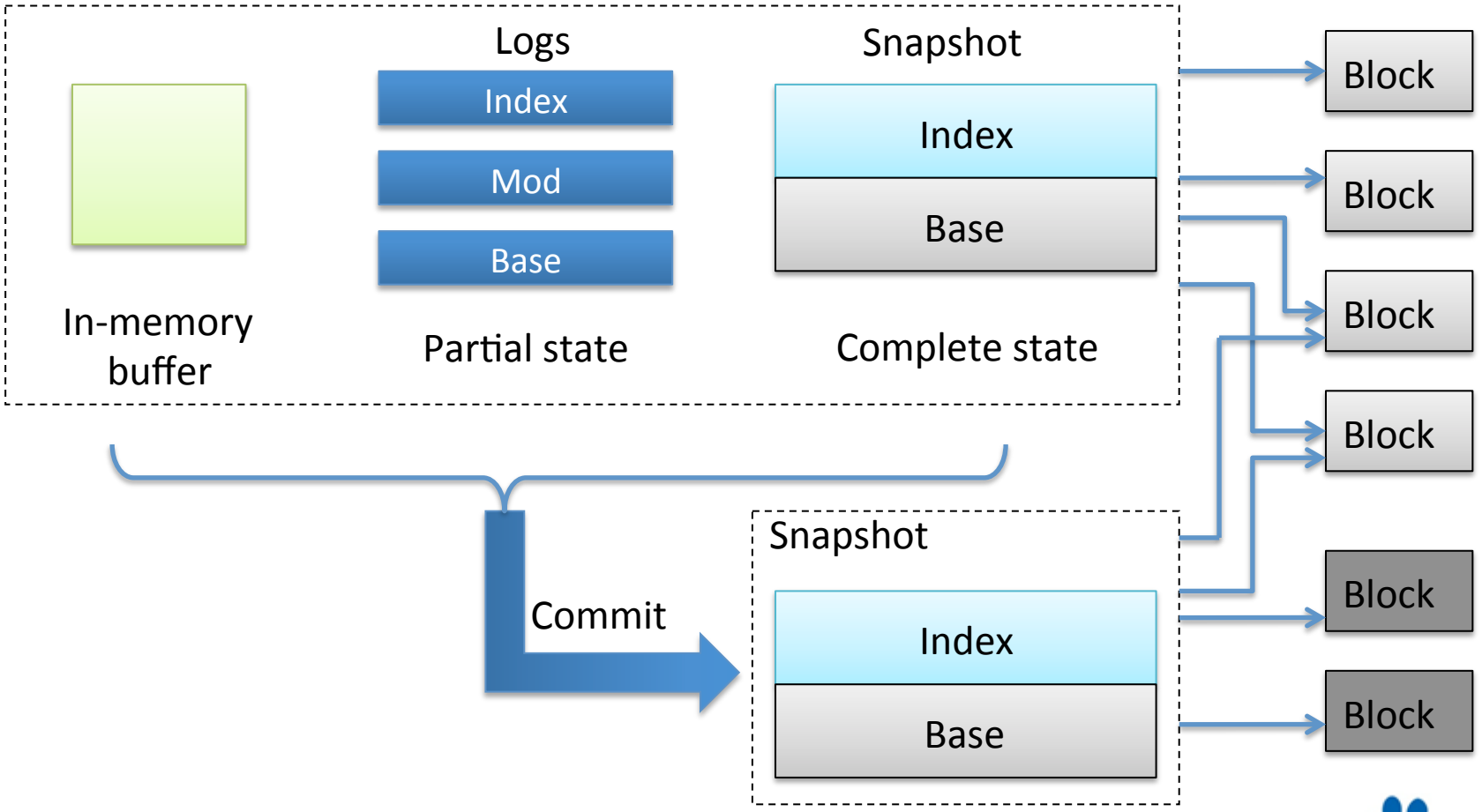
I: Insert
D: Delete
U: Updates

Some Observations, 2

- Table = base + index + mod + patch
  - Affiliate data can be used to help generate table
- Chained commits
  - (index, mod) -> patch -> new (data, index)
  - Sort-Merge + simple scheduling engine
- Optimizations
  - Keep old blocks if few changes
  - Answer scan during commit process
- Snapshots
  - Naturally generated by Commit
  - Share unchanged blocks

Baidu 百度

Logical Organization of a Table

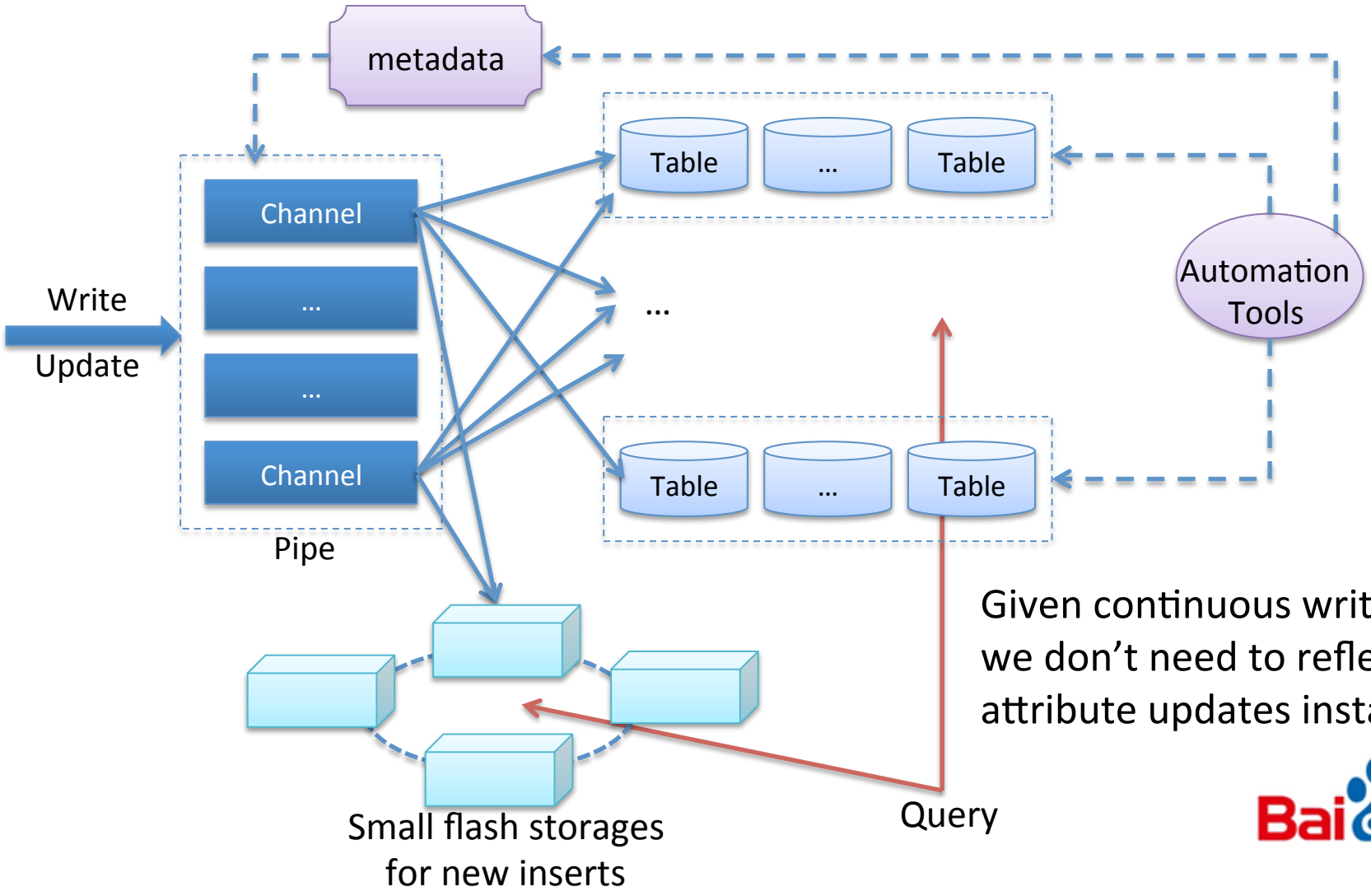


Problems

- How to get the latest results?
  - Too costly to go through all logs
- How to optimize attribute query and dump?
  - Column storage?

Baidu 百度

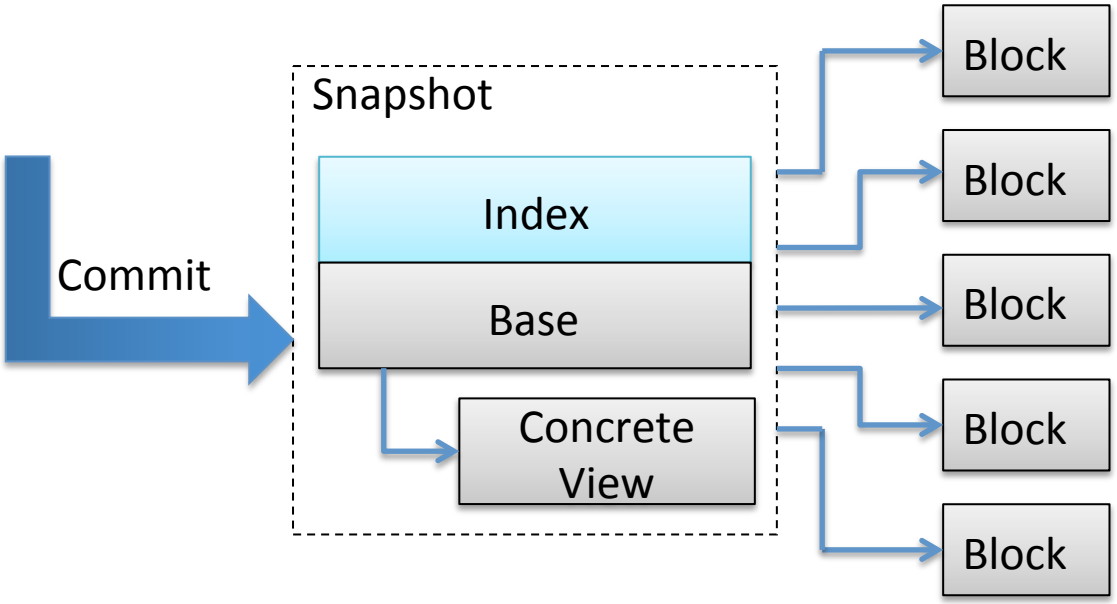
Introduce Other Storages to Support Freshness



Given continuous writes, we don't need to reflect attribute updates instantly.

Concrete View

Duplicated columns
Generated when commit
Speed up attribute query



Rough Performance Results

- Machine model
  - 12 SATA disks, 16 GB memory
- Write
  - Measured by Bytes/s
  - >100 MB/s, network is the bottleneck
- Random query
  - ~400 Q/s
- Commit
  - Aggregated bandwidth: >1GB/s

Baidu 百度

System Building Philosophy: Abstract for Reuse and Simplicity

- Log-structure → Stream (schema-ed, structured)
  - Redefine merge() to support insert/update/delete with schema
- Stream → Table (indexed, unsorted)
  - Sort-Merge is intensively reused
  - Simple scheduling engine can support complicated multi-stage execution flow
- <30K C++ LOC
  - Including async programming library and communication library

Baidu 百度

System Building Philosophy: Convolve Design and Implementation

- Increasing parallelism (cores, disks) in a single machine requires asynchronous programming
  - Streaming blocks + multi-buffering → 1 GB/s
  - Complicate one component, but simplify others
- Implementation could influence design very much
  - Flat and dense record layout
  - Keep blocks with few changes
  - Read more than needed, then filter (attribute query → scan)
- Optimize to the extreme, then relax elsewhere
  - The management layer is written in simple python

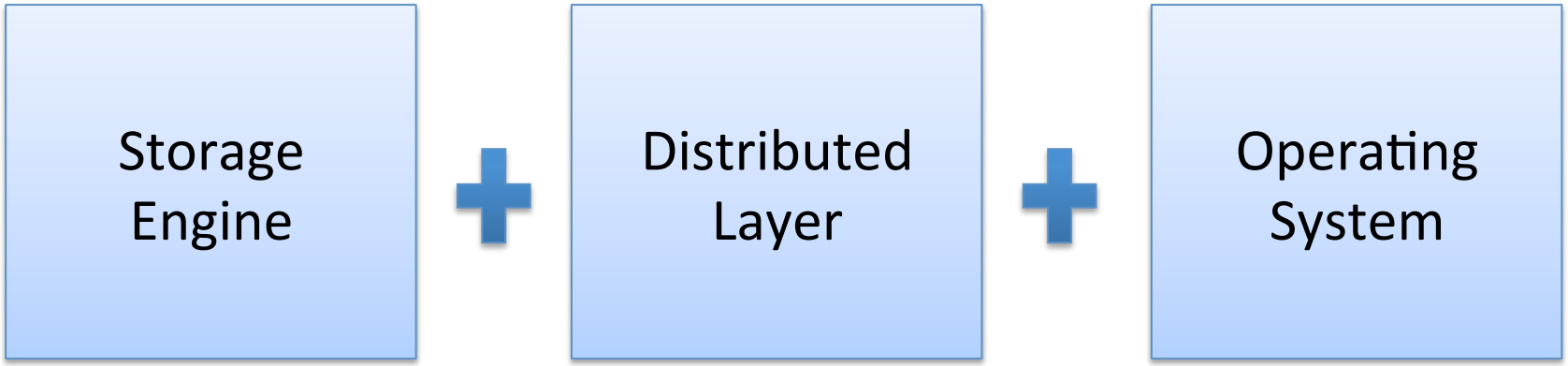
Baidu 百度

System Building Philosophy: Disaggregate and Recompose

- Understand and control the system
  - Fit in existing operating process
- Study the components individually, and optimize the system gradually
  - Data pipe
  - Concrete view
  - Small random storage
- Gain enough recomposability
  - Resolve conflict of requirements by different execution paths without sacrificing performance and generality

Baidu 百度

Storage System from My Perspective



Data organization

Reliability guarantee
Self-manageability

Resource scheduling
Performance guarantee

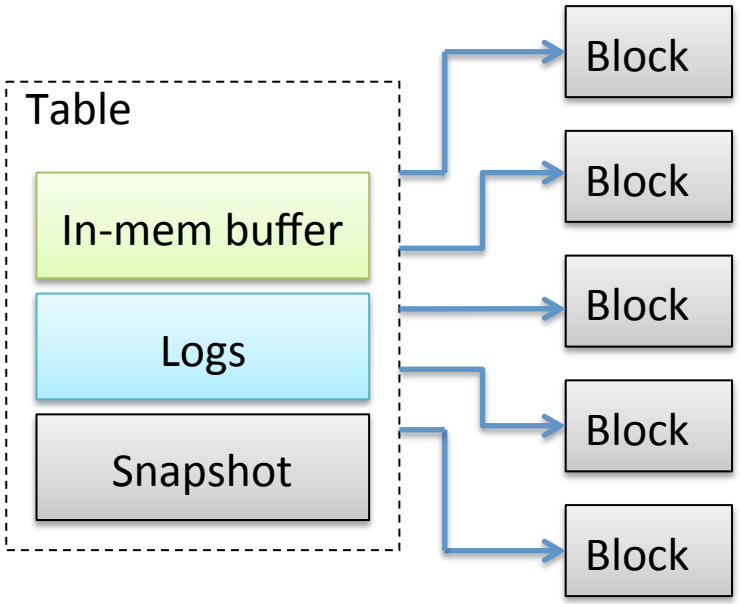
Baidu 百度

Revisit the Design Choices for New Requirements (in 2010)

- Larger scale
  - Repartitioning
- To support online services
  - Freshness and consistency
  - Realtime query by attribute
- To support file, object, pipe
- Better manage-ability
  - Disaggregate too much, need consolidation
  - Tolerate partial failure of disks

Baidu 百度

Logical Replication vs. Physical Replication



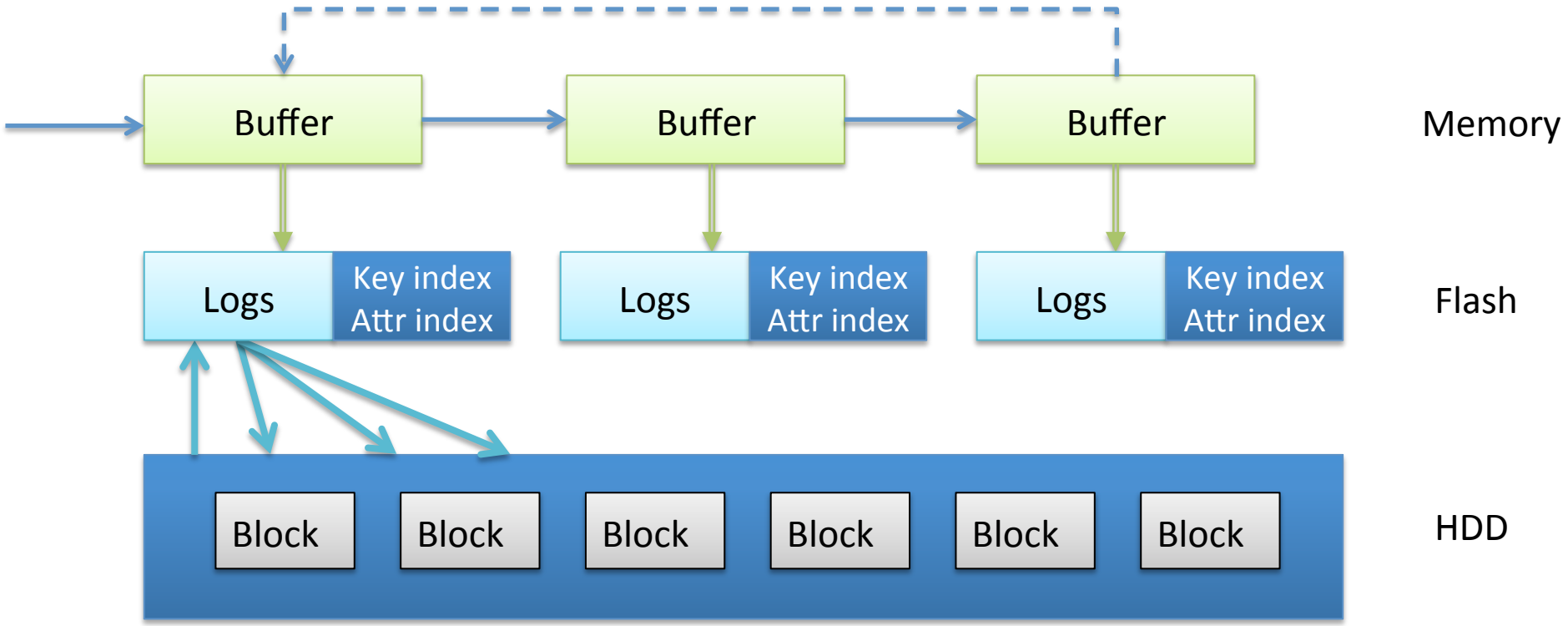
Physical: all identical

Logical: identical in states, different in blocks

	Pros	Cons
Logical	Replicas are less dependent Support record-level repair	3x commit cost Repair whole table even 1 block failure
Physical	Support block-level repair	3x network cost during commit

Baidu 百度

Hybrid Approach



- 3 layers by different storage media
- Buffer and blocks are physically replicated
- Log (partial state) is logically replicated
- Buffer and logs are co-located, but blocks may be not

Baidu 百度

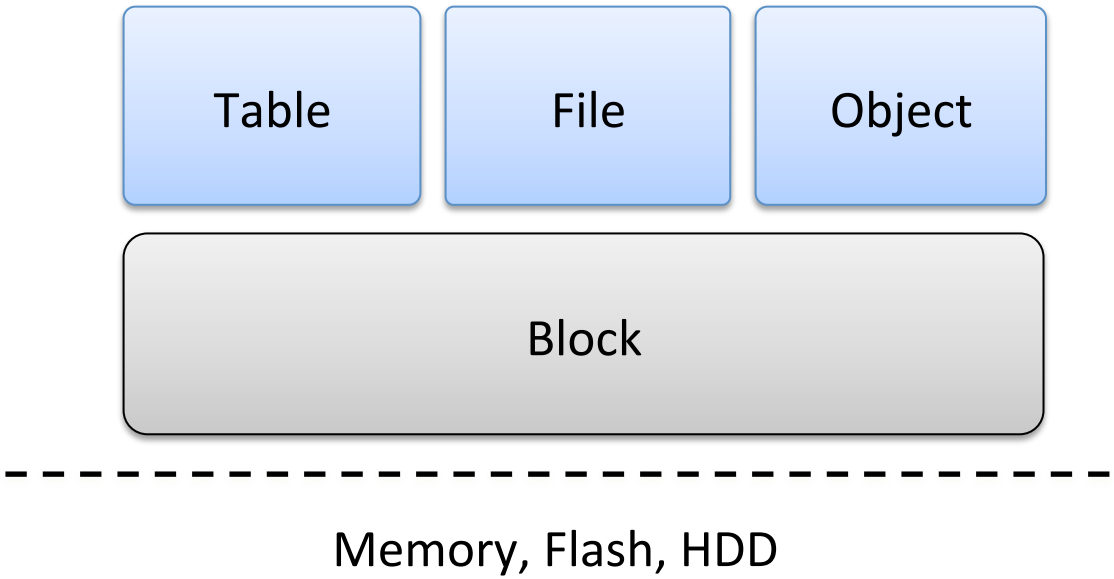
Distributed Centralization

- Partitioned into Slices
- Only primary copy responds to requests
- Slices are distributed and co-located

- Master
  - Only needs to be involved when structure change:
    node failure, repartitioning
  - Can be reconstructed
  - Easy to implement

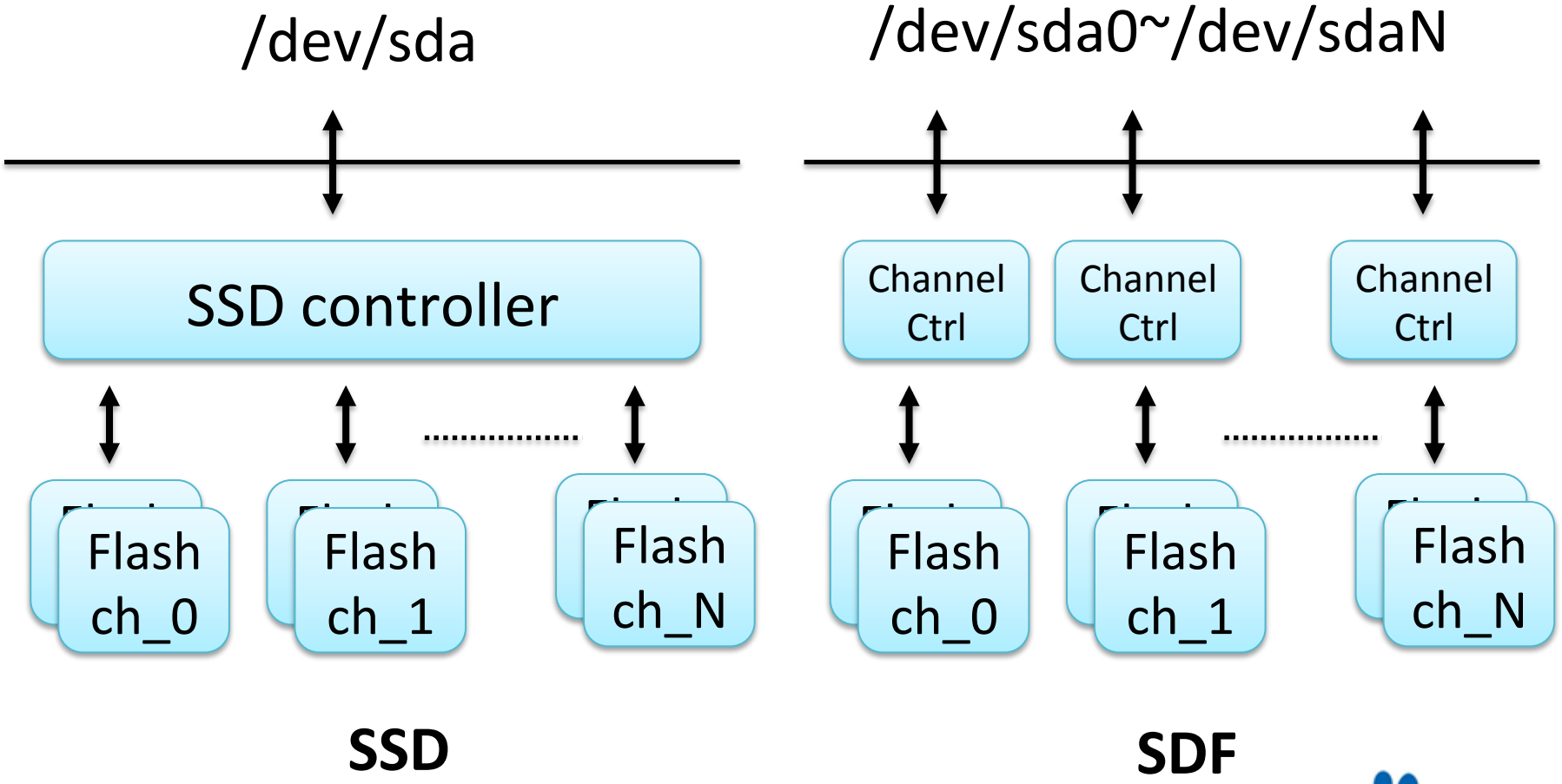
Baidu 百度

One Storage for All Structures



- Decouple storage from interface
- Separate data from schema
- Different forms of logs can represent object/file

Software-Defined Flash



HDD Failure Prediction

- Current results
  - 77M training data
  - 215K features
    - Combination of 25 basic SMART features
  - 98% DR (Detection Rate) + 0.3% FAR (False Alarm Rate)
  - Existing methods based on basic SMART features
    - 56% DR + 0.8% FAR, 52% DR + 0 FAR
- Implication of big training data
  - Much more features without over-fitting
  - Simple linear model
- Indication for system building
  - System is not just software: a lot of resources and stats
  - Collect data for tuning after deployment

Baidu 百度

Thanks You!