支付宝 Alipay.com 知托付

OceanBase核心数据结构
In Memory B+ Tree

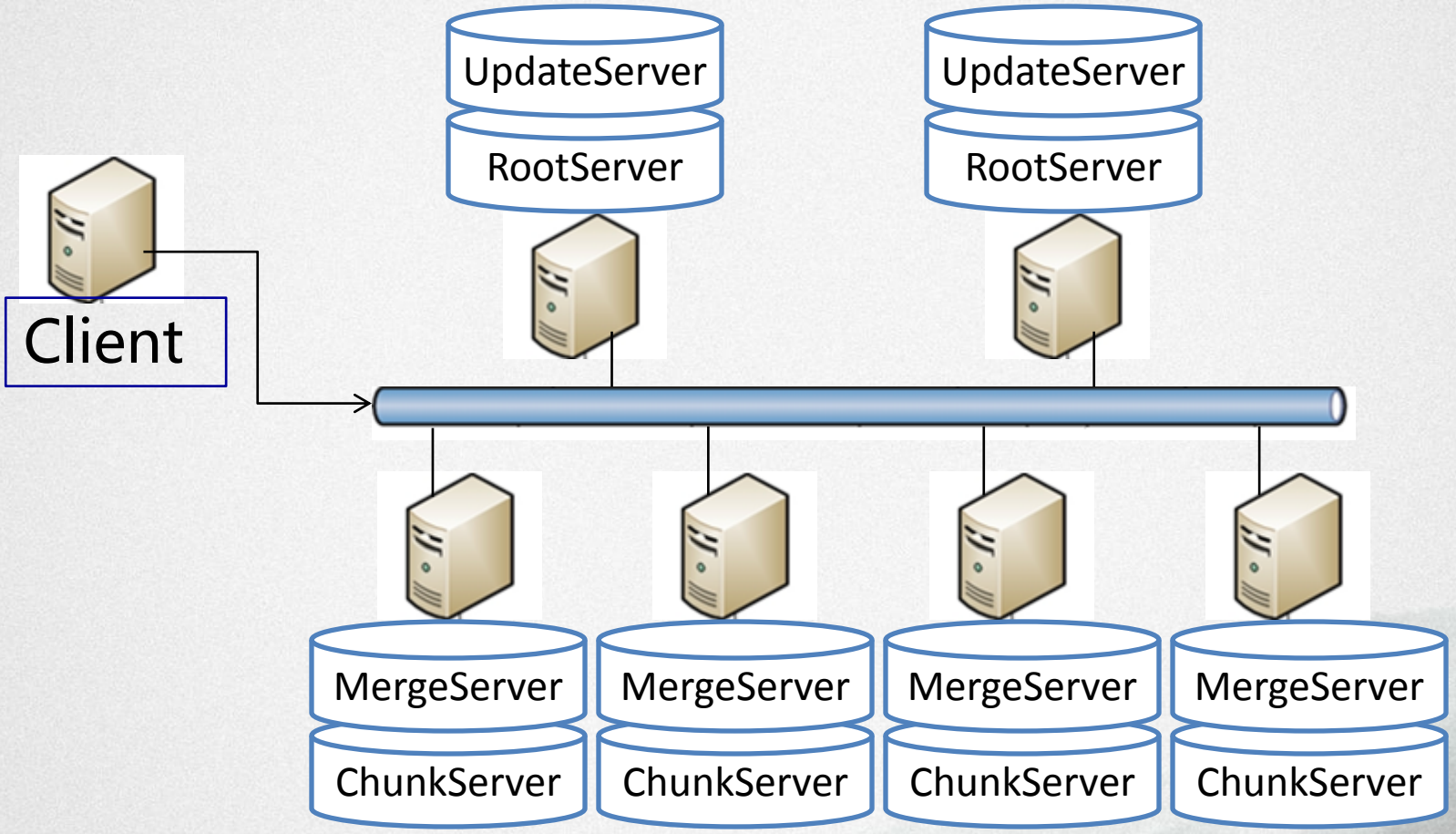
颜然/韩富晟

2013.1

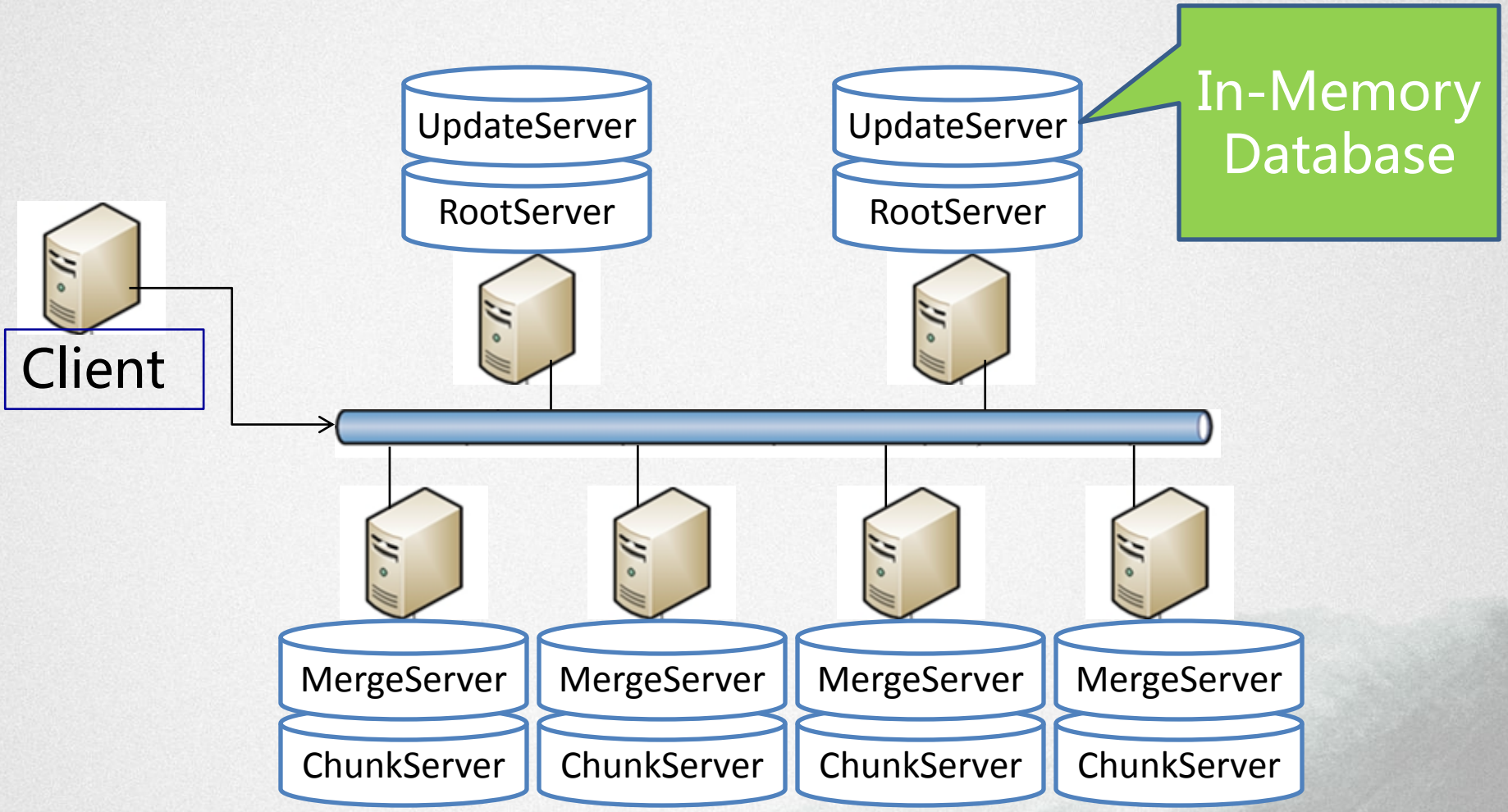
Agenda

- Motivation
- Evolution
- Algorithm
- Performance Evaluation

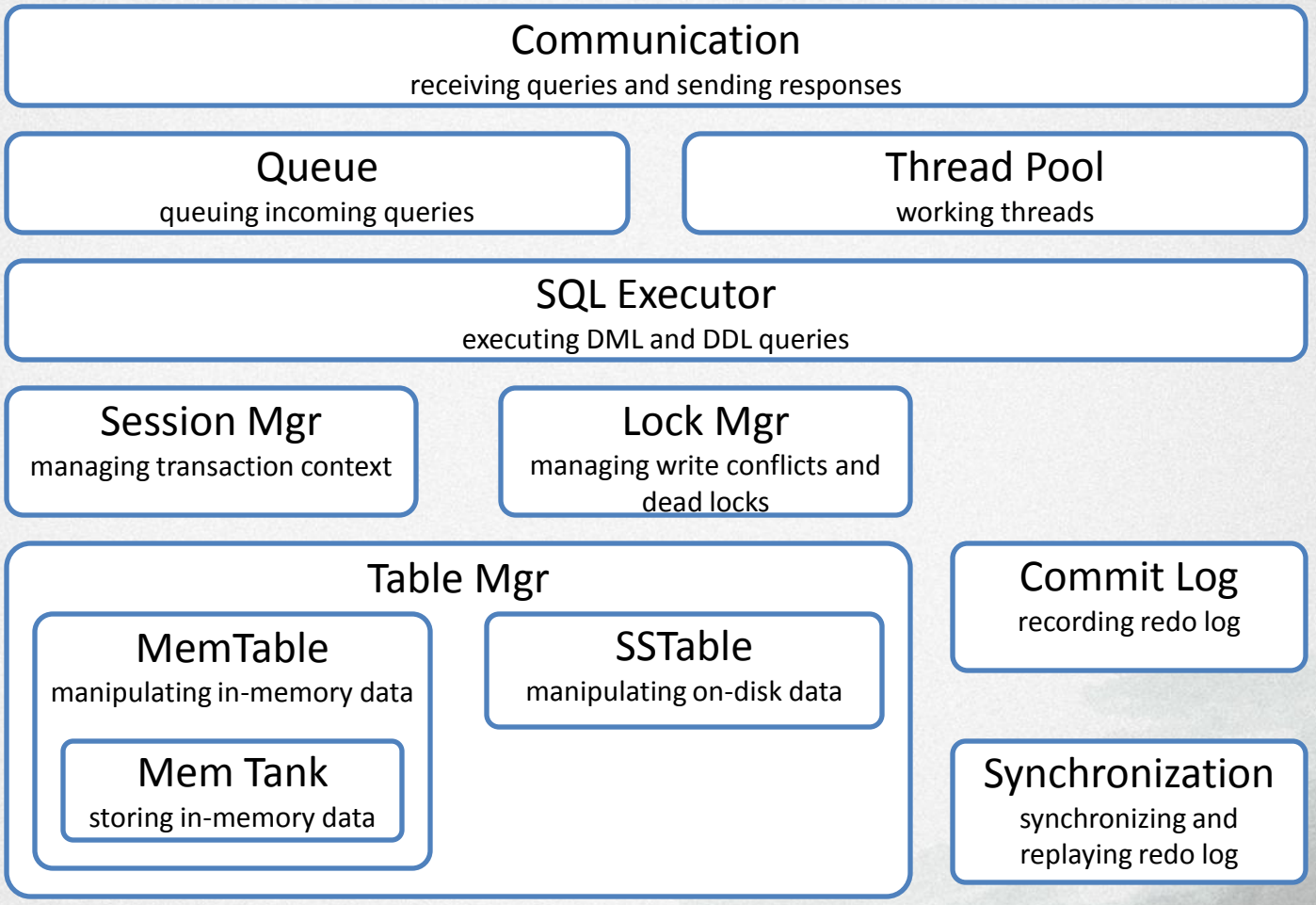
OceanBase Architecture



OceanBase Architecture

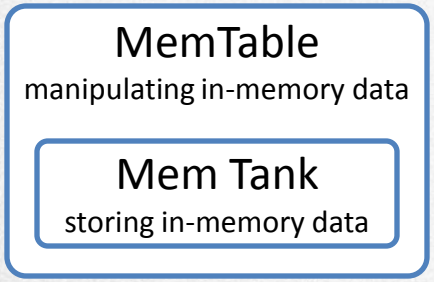


UpdateServer Architecture



UpdateServer MemTable

- Operations provided:
  - Insert data
  - Update data
  - Delete
  - Point query
  - Range query
- All operations are done in memory



MemTable Indexes

- MemTable employ two kinds of indexes
  - Hash: used for point query
  - B+ Tree: used for range query
- Indexes build on primary keys
- Insert operations add new index entries into both structures

B+ Tree Index Requirement

- Read operation is more frequent than write
- Read should not be blocked by write
- High performance required

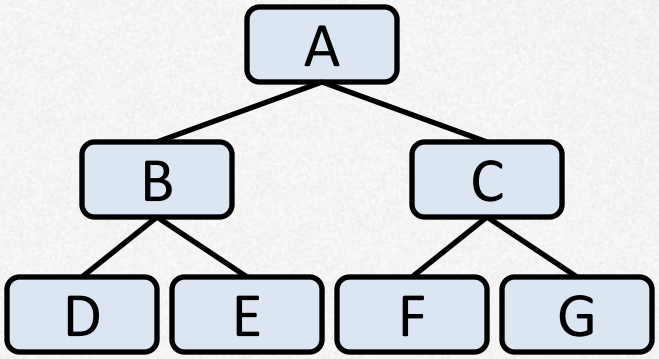
B+ Tree Evolution

- OceanBase version 0.1 ~ 0.3:
  - Support only single-thread modification
  - Copying a path from root to leaf when doing modification
  - Support a kind of MVCC

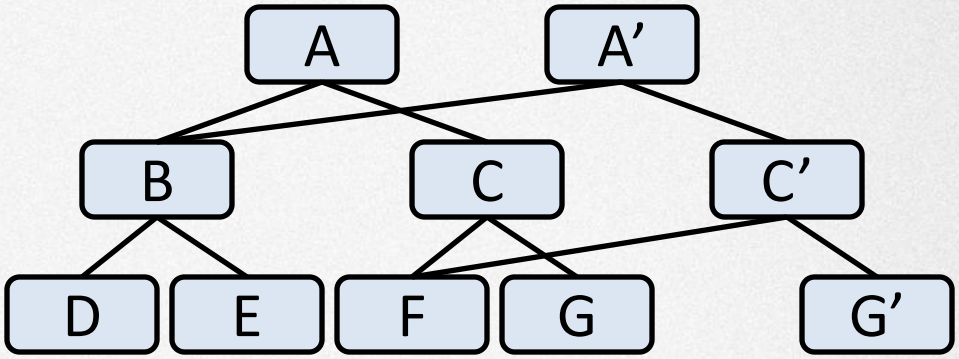
B+ Tree Evolution (cont.)

- OceanBase version 0.4:
  - Support concurrent modification
  - Copying as fewer nodes as possible
  - Still no conflicts between read and write
  - Find-grained locking for write
  - Embedding keys into tree nodes for improved cache utilization
  - No MVCC support, a separate MVCC module is used

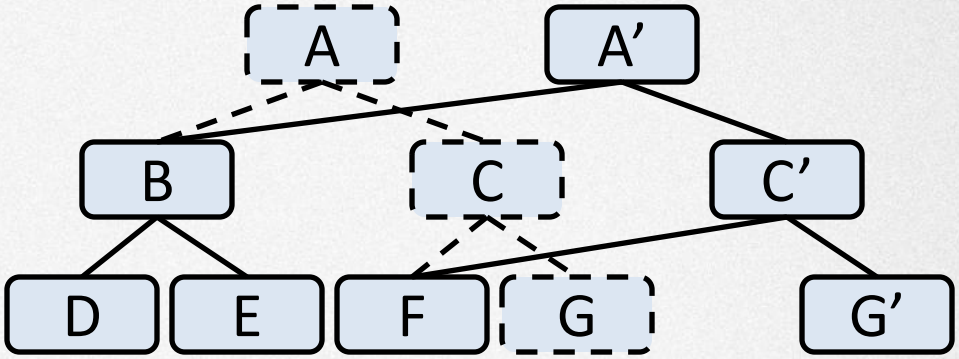
1st gen B+ Tree



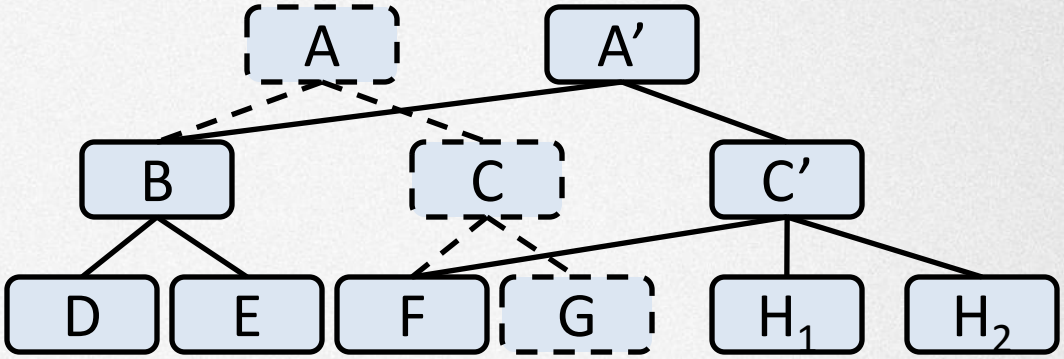
1st gen B+ Tree



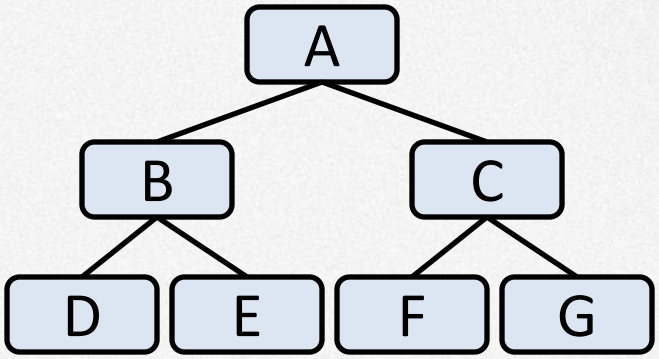
1st gen B+ Tree



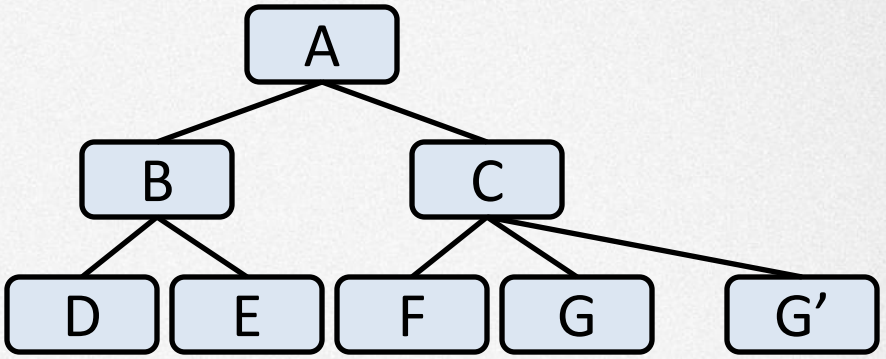
1st gen B+ Tree



2nd gen B+ Tree

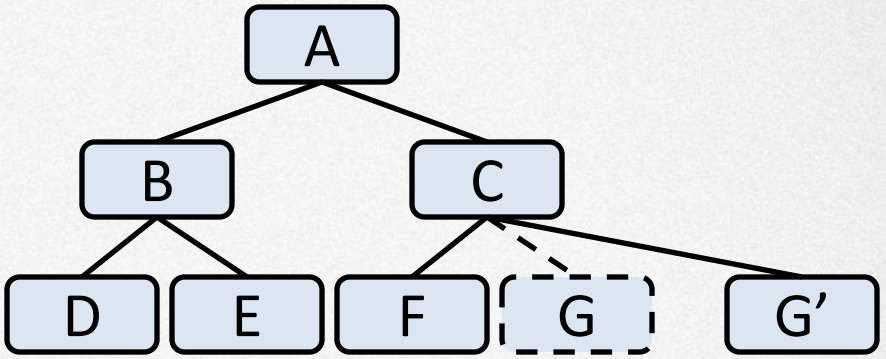


2nd gen B+ Tree



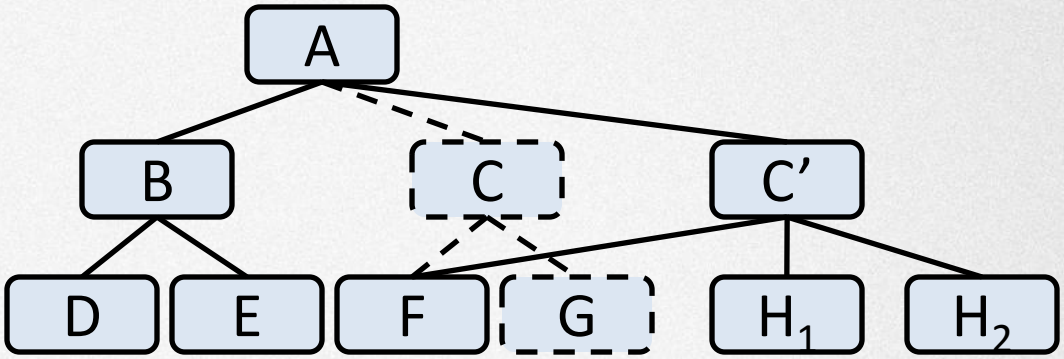
- Copying node only when keys are modified
- Using atomic CAS If just replacing pointer

2nd gen B+ Tree



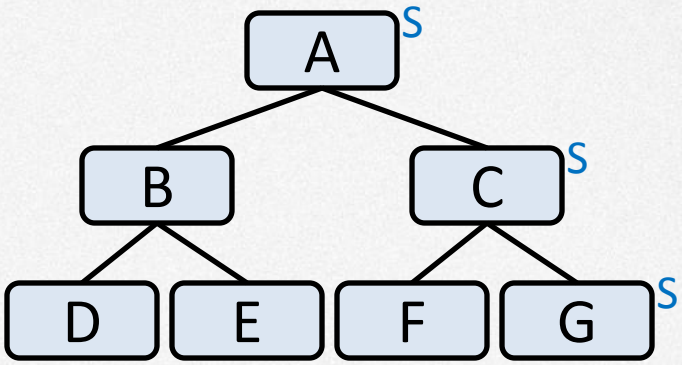
- Copying node only when keys are modified
- Using atomic CAS If just replacing pointer

2nd gen B+ Tree



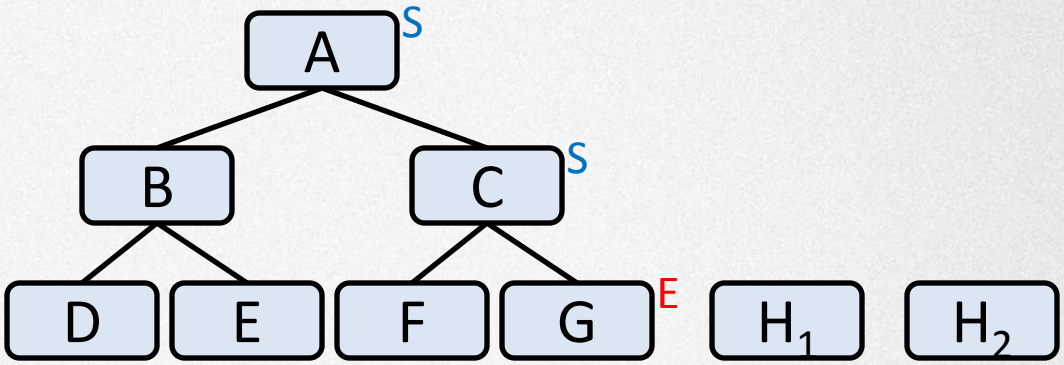
- Copying node only when keys are modified
- Using atomic CAS If just replacing pointer

2nd gen B+ Tree: Locking



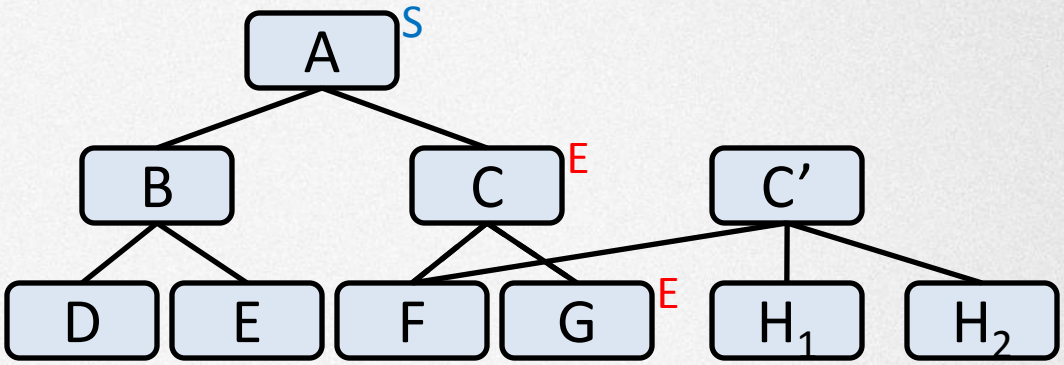
- Nodes on the path are added shared locks
- Node which keys are modified is added exclusive locks just before modification
- If two or more threads add exclusive locks on the same node, only one succeeds and others fail and retry

2nd gen B+ Tree: Locking



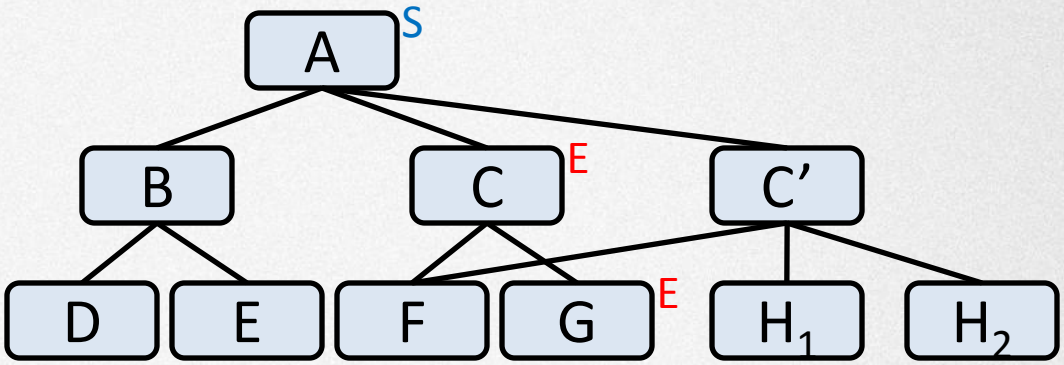
- Nodes on the path are added shared locks
- Node which keys are modified is added exclusive locks just before modification
- If two or more threads add exclusive locks on the same node, only one succeeds and others fail and retry

2nd gen B+ Tree: Locking



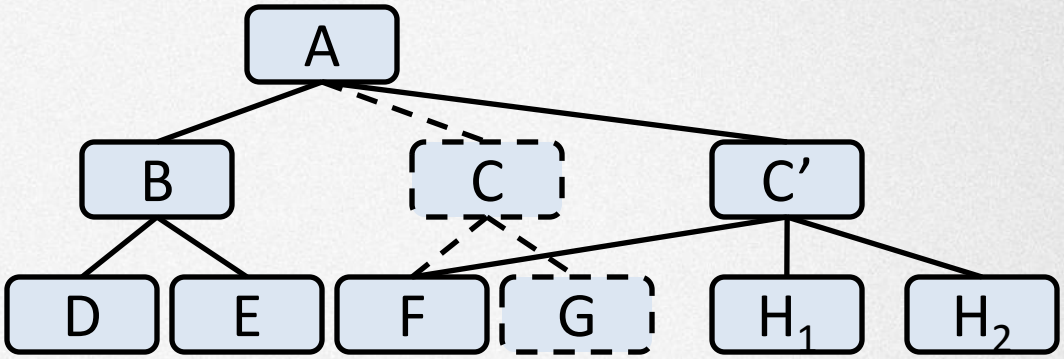
- Nodes on the path are added shared locks
- Node which keys are modified is added exclusive locks just before modification
- If two or more threads add exclusive locks on the same node, only one succeeds and others fail and retry

2nd gen B+ Tree: Locking



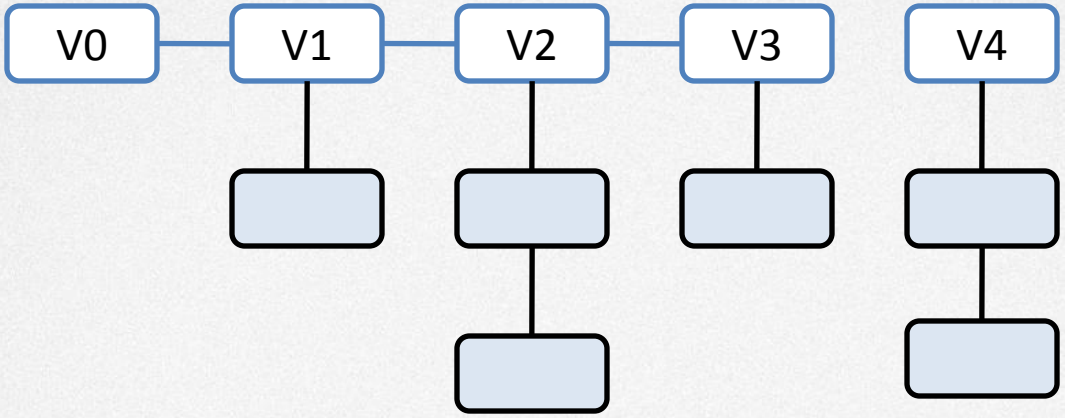
- Nodes on the path are added shared locks
- Node which keys are modified is added exclusive locks just before modification
- If two or more threads add exclusive locks on the same node, only one succeeds and others fail and retry

2nd gen B+ Tree: Locking



- Nodes on the path are added shared locks
- Node which keys are modified is added exclusive locks just before modification
- If two or more threads add exclusive locks on the same node, only one succeeds and others fail and retry

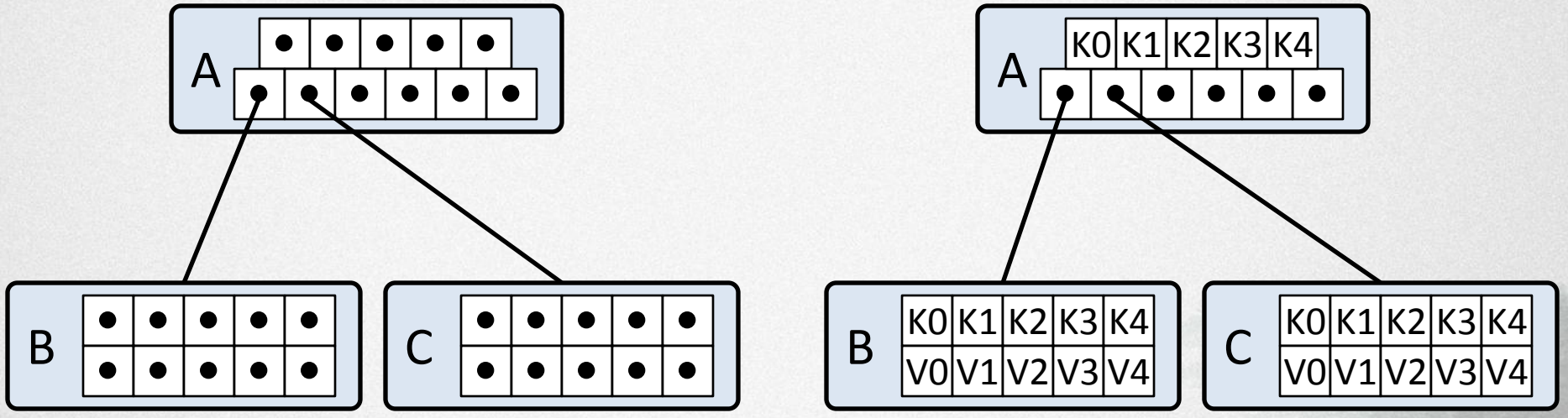
2nd gen B+ Tree: Memory Recycle



- The entire B+ Tree maintains a version. Each time modification happens, the version is incremented
- Reference count is applied to the version
- All memory needing recycle are assign to each version
- The oldest version with zero reference count can be recycled

Performance Improvement

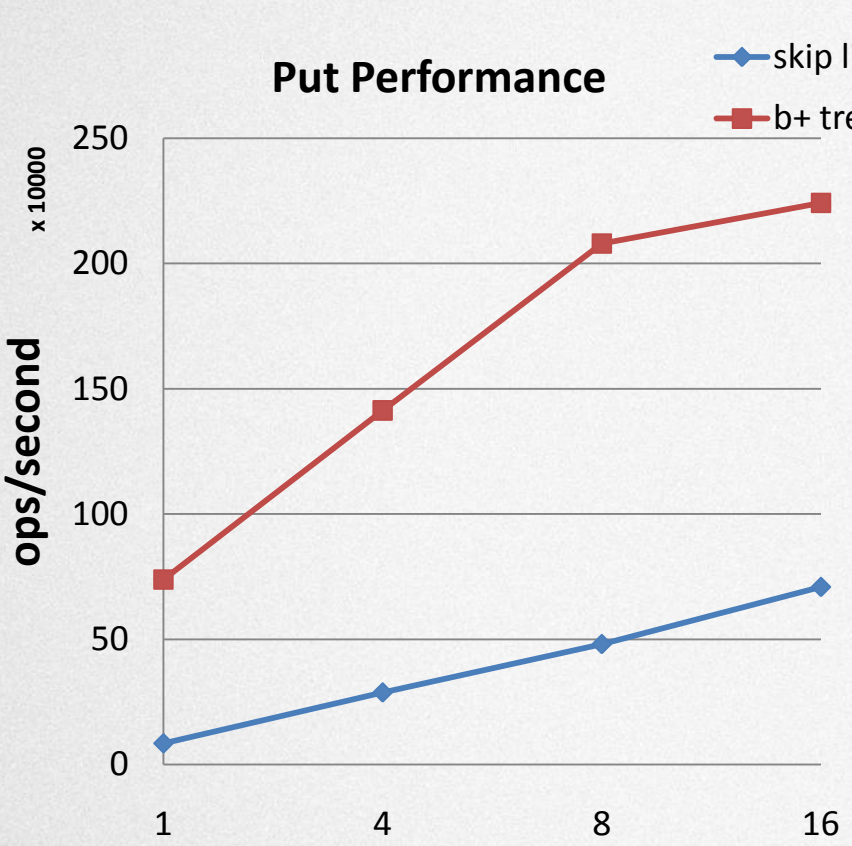
- Better cache utilization
  - Put keys directly into B+ Tree node

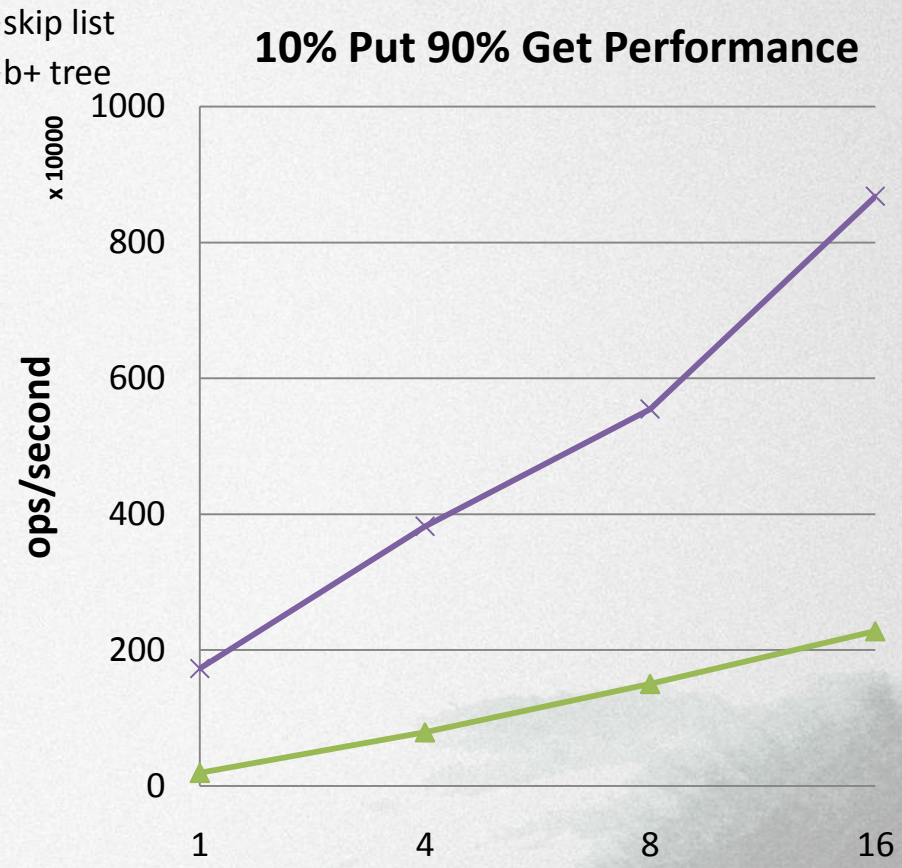


Comparison with other structures

- Compare to other structures with range query support
  - In Memory B+ Tree
  - AVL Tree
  - Skip List
  - T-tree
- All perform search and insert operations in O(log n)
- B+ Tree is better in cache utilization, but waste memory

Performance Evaluation





Q&A

- Thanks
- 颜然/@韩富晟
- yanran.hfs@alipay.com