南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

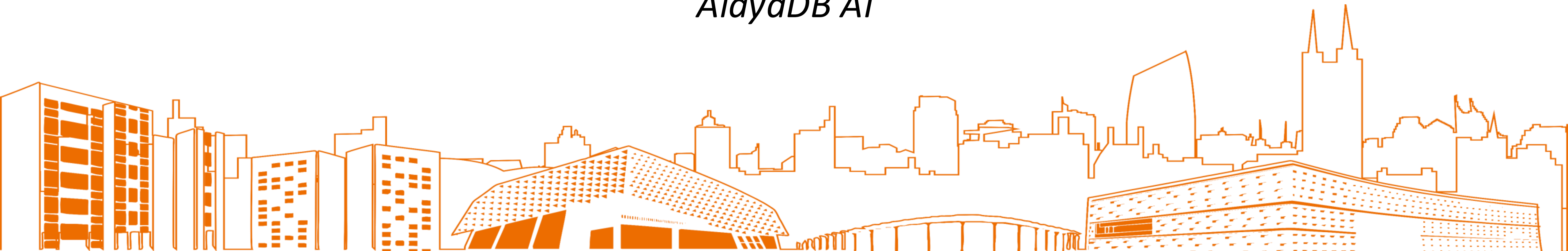
Accelerating Merkle Patricia Trie with GPU

Yangshen Deng*
dengys2022@mail.sustech.edu.cn

Muxi Yan*
yanmx2022@mail.sustech.edu.cn

Bo Tang
tangb3@sustech.edu.cn

Southern University of Science and Technology
AlayaDB AI



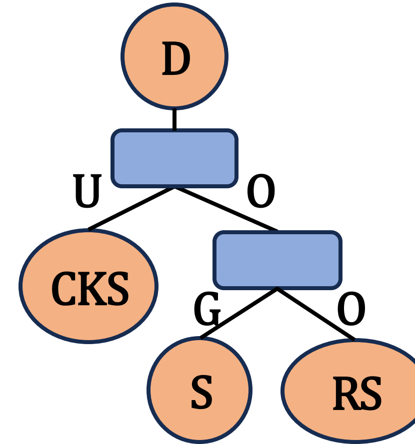
Overview

- **Motivation**
- Challenges
- Solutions
- Results
- Takeaways

Merkle Patricia Trie (MPT)

A KV index for immutable data systems

- **Blockchain:** Ethereum, FISCO BCOS, Quorum, etc.
- **Verifiable database:** Alibaba LedgerDB, Spitz, GSSE, etc.



Structure

- **Patricia trie (radix trie)**

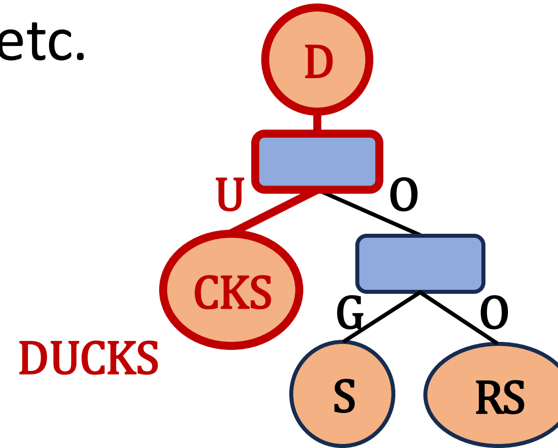
Merkle Patricia Trie (MPT)

A KV index for immutable data systems

- **Blockchain:** Ethereum, FISCO BCOS, Quorum, etc.
- **Verifiable database:** Alibaba LedgerDB, Spitz, GSSE, etc.

Structure

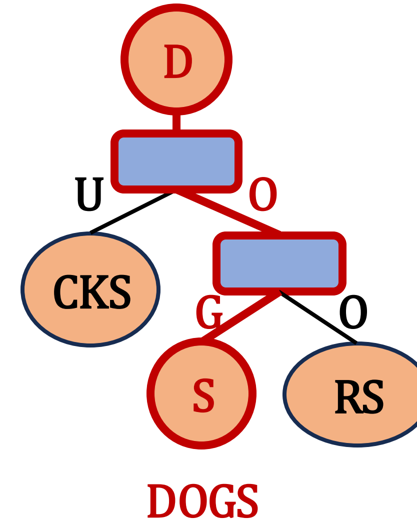
- **Patricia trie (radix trie)**



Merkle Patricia Trie (MPT)

A KV index for immutable data systems

- **Blockchain:** Ethereum, FISCO BCOS, Quorum, etc.
- **Verifiable database:** Alibaba LedgerDB, Spitz, GSSE, etc.



Structure

- **Patricia trie (radix trie)**

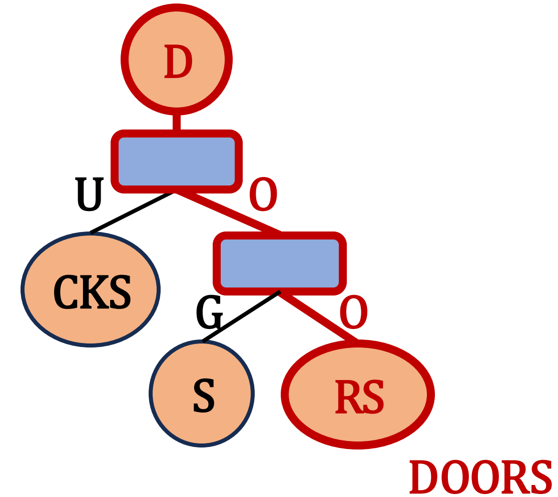
Merkle Patricia Trie (MPT)

A KV index for immutable data systems

- **Blockchain:** Ethereum, FISCO BCOS, Quorum, etc.
- **Verifiable database:** Alibaba LedgerDB, Spitz, GSSE, etc.

Structure

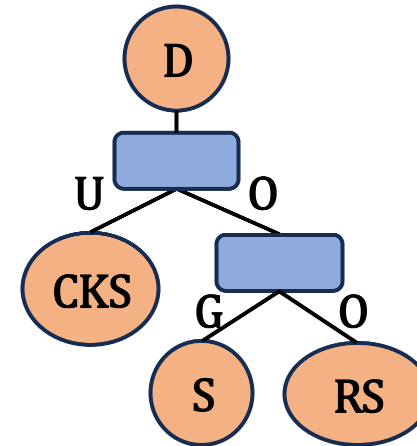
- **Patricia trie (radix trie)**



Merkle Patricia Trie (MPT)

A KV index for immutable data systems

- **Blockchain:** Ethereum, FISCO BCOS, Quorum, etc.
- **Verifiable database:** Alibaba LedgerDB, Spitz, GSSE, etc.



Structure

- **Patricia trie (radix trie)**
- **Cryptographical hashing (merkle tree)**

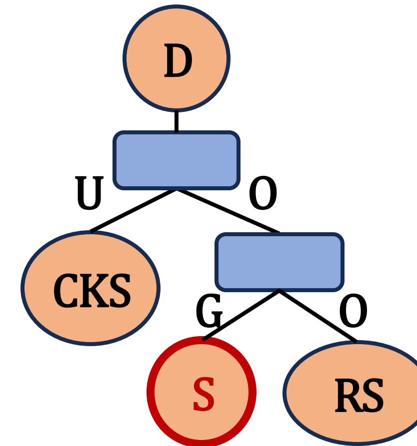
Merkle Patricia Trie (MPT)

A KV index for immutable data systems

- **Blockchain:** Ethereum, FISCO BCOS, Quorum, etc.
- **Verifiable database:** Alibaba LedgerDB, Spitz, GSSE, etc.

Structure

- **Patricia trie (radix trie)**
- **Cryptographical hashing (merkle tree)**

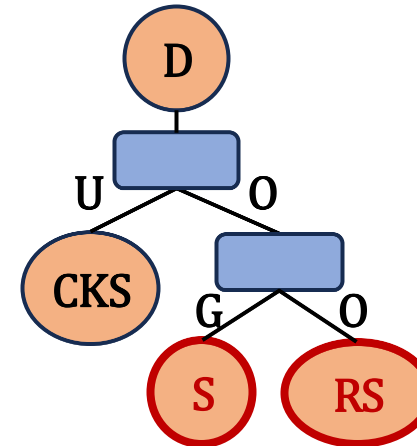


$$Root\ hash = \quad H(\mathbf{S})$$

Merkle Patricia Trie (MPT)

A KV index for immutable data systems

- **Blockchain:** Ethereum, FISCO BCOS, Quorum, etc.
- **Verifiable database:** Alibaba LedgerDB, Spitz, GSSE, etc.



Structure

- **Patricia trie (radix trie)**
- **Cryptographical hashing (merkle tree)**

$Root\ hash =$ $H(\mathbf{S})$ $H(\mathbf{RS})$

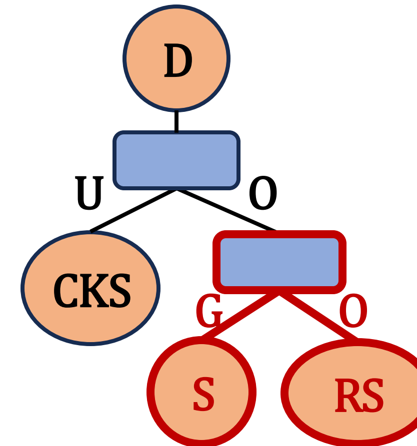
Merkle Patricia Trie (MPT)

A KV index for immutable data systems

- **Blockchain:** Ethereum, FISCO BCOS, Quorum, etc.
- **Verifiable database:** Alibaba LedgerDB, Spitz, GSSE, etc.

Structure

- **Patricia trie (radix trie)**
- **Cryptographical hashing (merkle tree)**

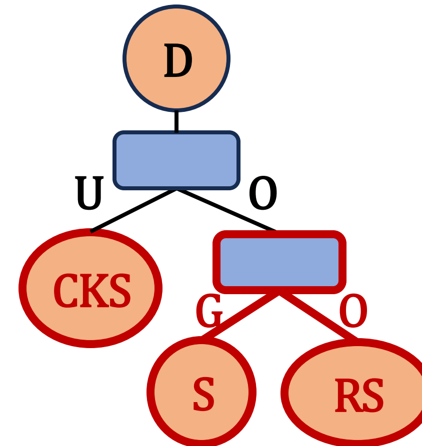


$Root\ hash = H(\mathbf{G}, H(\mathbf{S}), \mathbf{O}, H(\mathbf{RS}))$

Merkle Patricia Trie (MPT)

A KV index for immutable data systems

- **Blockchain:** Ethereum, FISCO BCOS, Quorum, etc.
- **Verifiable database:** Alibaba LedgerDB, Spitz, GSSE, etc.



Structure

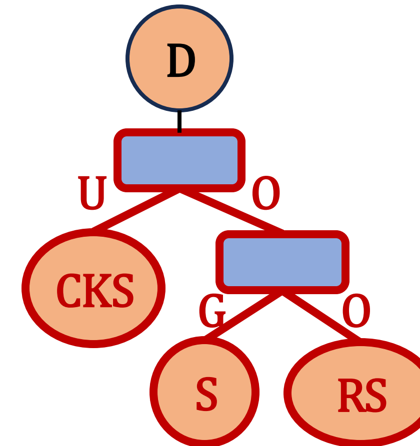
- **Patricia trie (radix trie)**
- **Cryptographical hashing (merkle tree)**

$$Root\ hash = \quad H(\boldsymbol{CKS}) \quad H(\mathbf{G}, H(\mathbf{S}), \mathbf{O}, H(\mathbf{RS}))$$

Merkle Patricia Trie (MPT)

A KV index for immutable data systems

- **Blockchain:** Ethereum, FISCO BCOS, Quorum, etc.
- **Verifiable database:** Alibaba LedgerDB, Spitz, GSSE, etc.



Structure

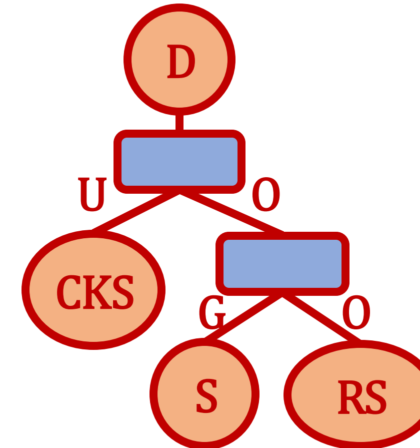
- **Patricia trie (radix trie)**
- **Cryptographical hashing (merkle tree)**

$$Root\ hash = \quad H(\boldsymbol{U}, H(\boldsymbol{CKS}), \boldsymbol{O}, H(\mathbf{G}, H(\mathbf{S}), \mathbf{O}, H(\mathbf{RS})))$$

Merkle Patricia Trie (MPT)

A KV index for immutable data systems

- **Blockchain:** Ethereum, FISCO BCOS, Quorum, etc.
- **Verifiable database:** Alibaba LedgerDB, Spitz, GSSE, etc.



Structure

- **Patricia trie (radix trie)**
- **Cryptographical hashing (merkle tree)**

$$Root\ hash = H(\mathbf{D}, H(\boldsymbol{U}, H(\boldsymbol{CKS}), \boldsymbol{O}, H(\mathbf{G}, H(\mathbf{S}), \mathbf{O}, H(\mathbf{RS}))))$$

Immutable Databases: e.g., Blockchain

Txn 1
Txn 2
Txn 3
Txn 4

Batch Transaction Processing

update k1, v1
update k2, v2
update k3, v3
update k4, v4
update k5, v5
update k6, v6

Batch Update Cryptographical Index

Block
Txn 1
Txn 2
Txn 3
Txn 4
Root Hash

read

write

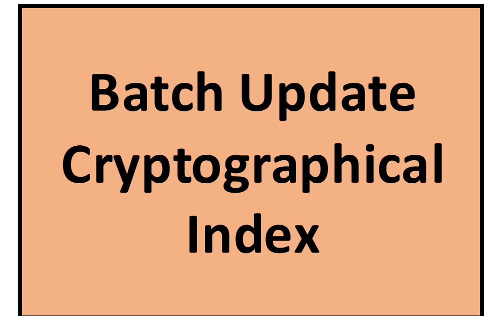
MPT

return hash

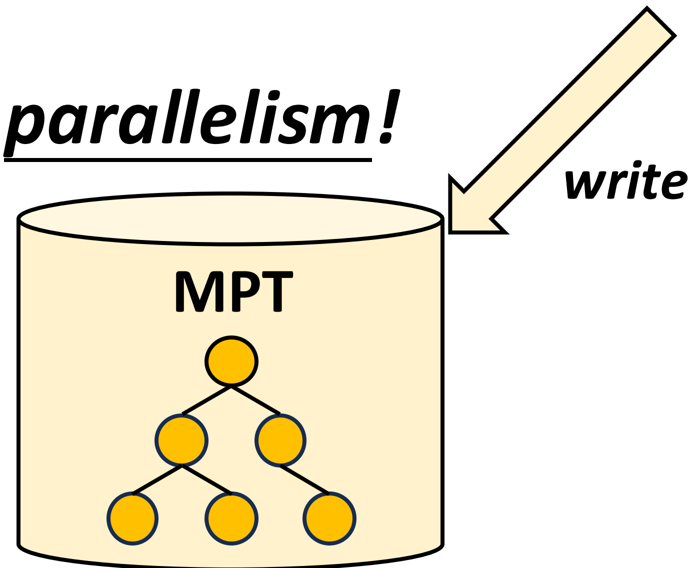
Accelerating MPT with GPU

The bottleneck: MPT update

- Heavy writes to index structure
  Observation 1: The writes are batched
- Heavy hash computation
  Observation 2: Compute-sensitive and easy to parallelize



GPU processes in batch with massive parallelism!



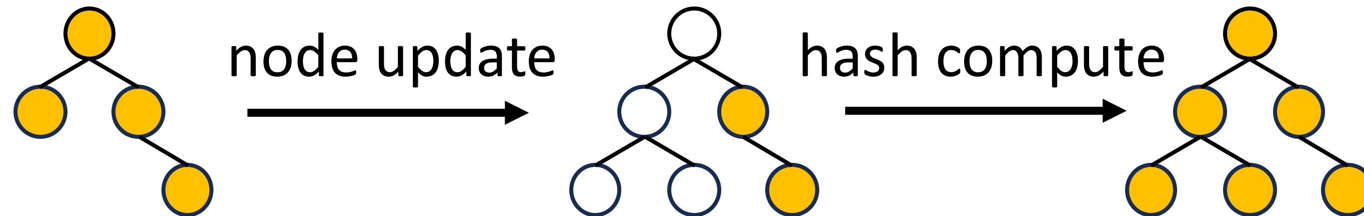
Overview

- Motivation
- **Challenges**
- Solutions
- Results
- Takeaways

Challenges: How to parallelize?

Separate into two phases

- **Node update:** concurrent modifications to index structure.
- **Hash compute:** read-write dependency in concurrent hash updates.



Overview

- Motivation
- Challenges
- **Solutions**
- Results
- Takeaways

Solution

Node Update

- PhaseNU
- LockNU

Hash Compute

- PhaseHC

Solution

DBGroup

Node Update

- PhaseNU (lock-free)
- LockNU (lock-base)

Hash Compute

- PhaseHC

Solution

Node Update

- **PhaseNU**
- LockNU

Hash Compute

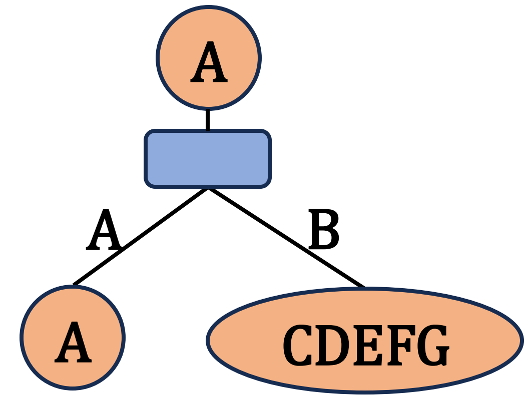
- PhaseHC

PhaseNU

DBGroup

Key Idea: Proactively expanding nodes and eliminating locks.

Without PhaseNU:



ABBB

ABCCC

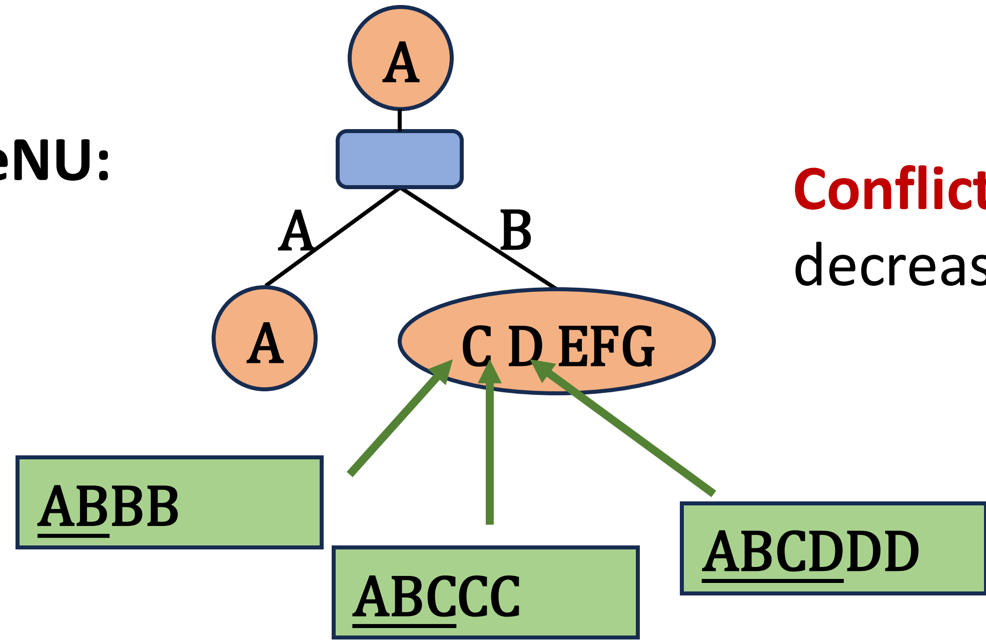
ABCDDD

PhaseNU

DBGroup

Key Idea: Proactively expanding nodes and eliminating locks.

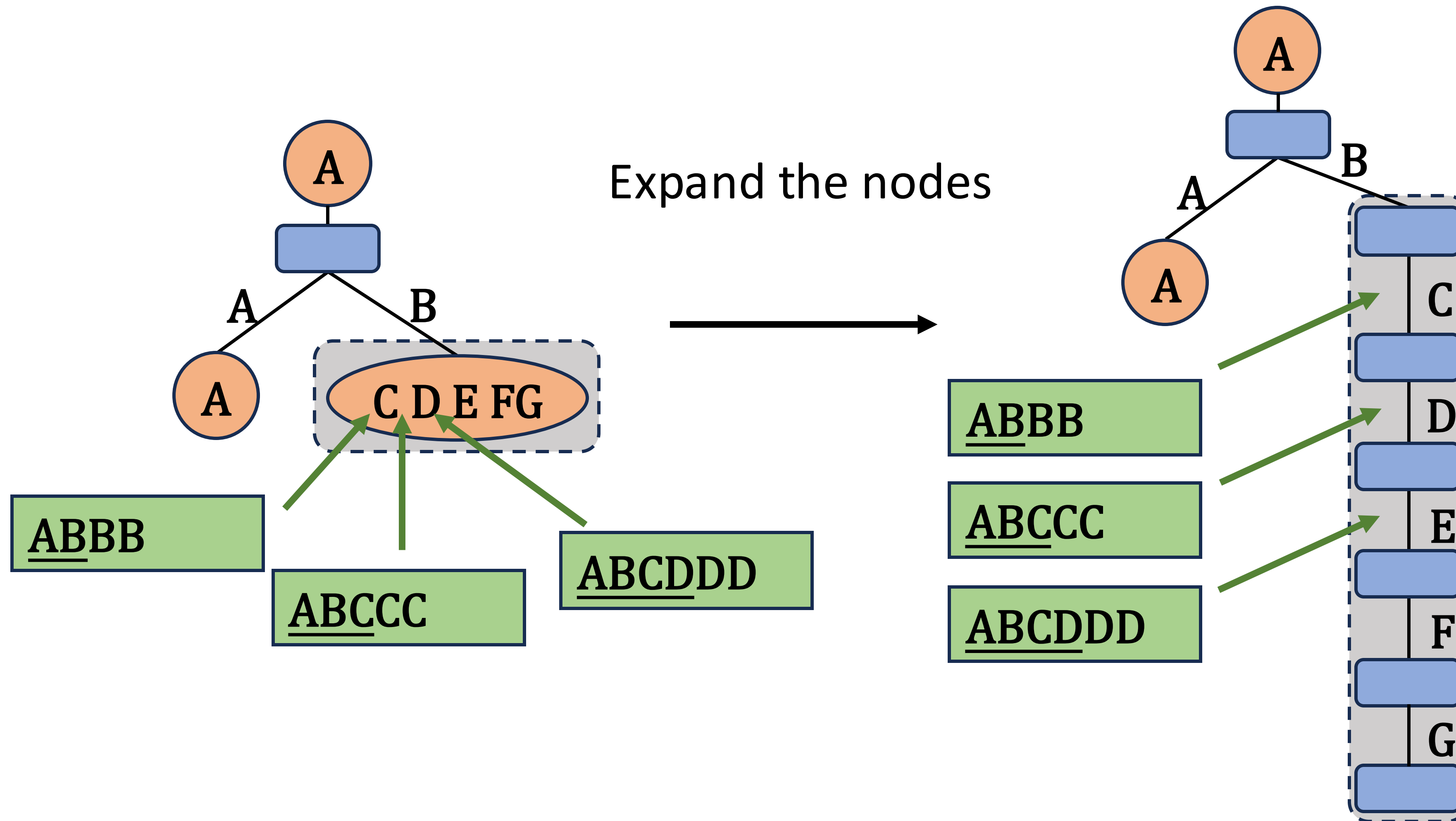
Without PhaseNU:



Conflict: parallelism degree decreases from 3 to 1

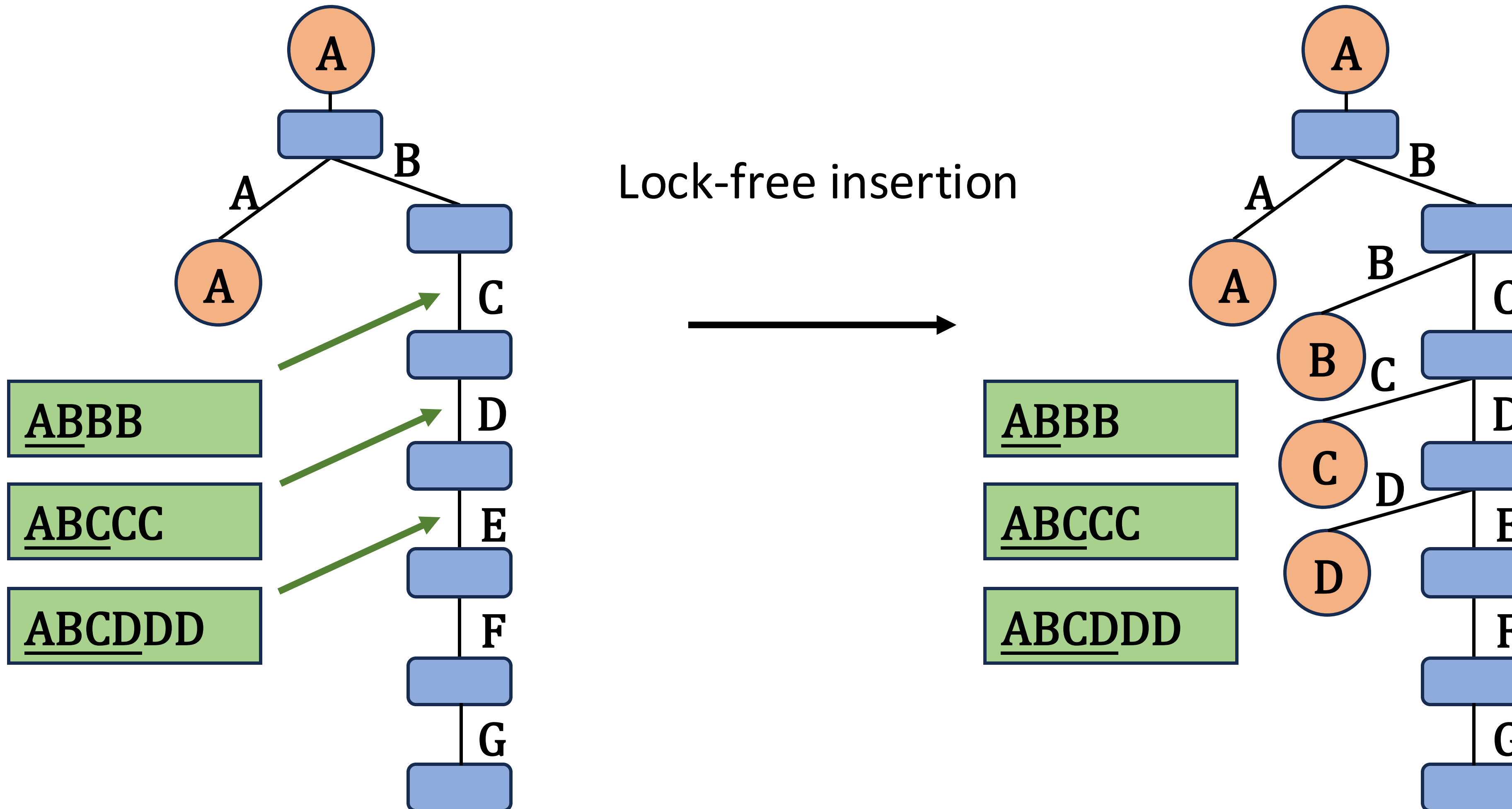
PhaseNU: Phase 1 – Expanding Phase

Key Idea: Proactively expanding nodes and eliminating locks.



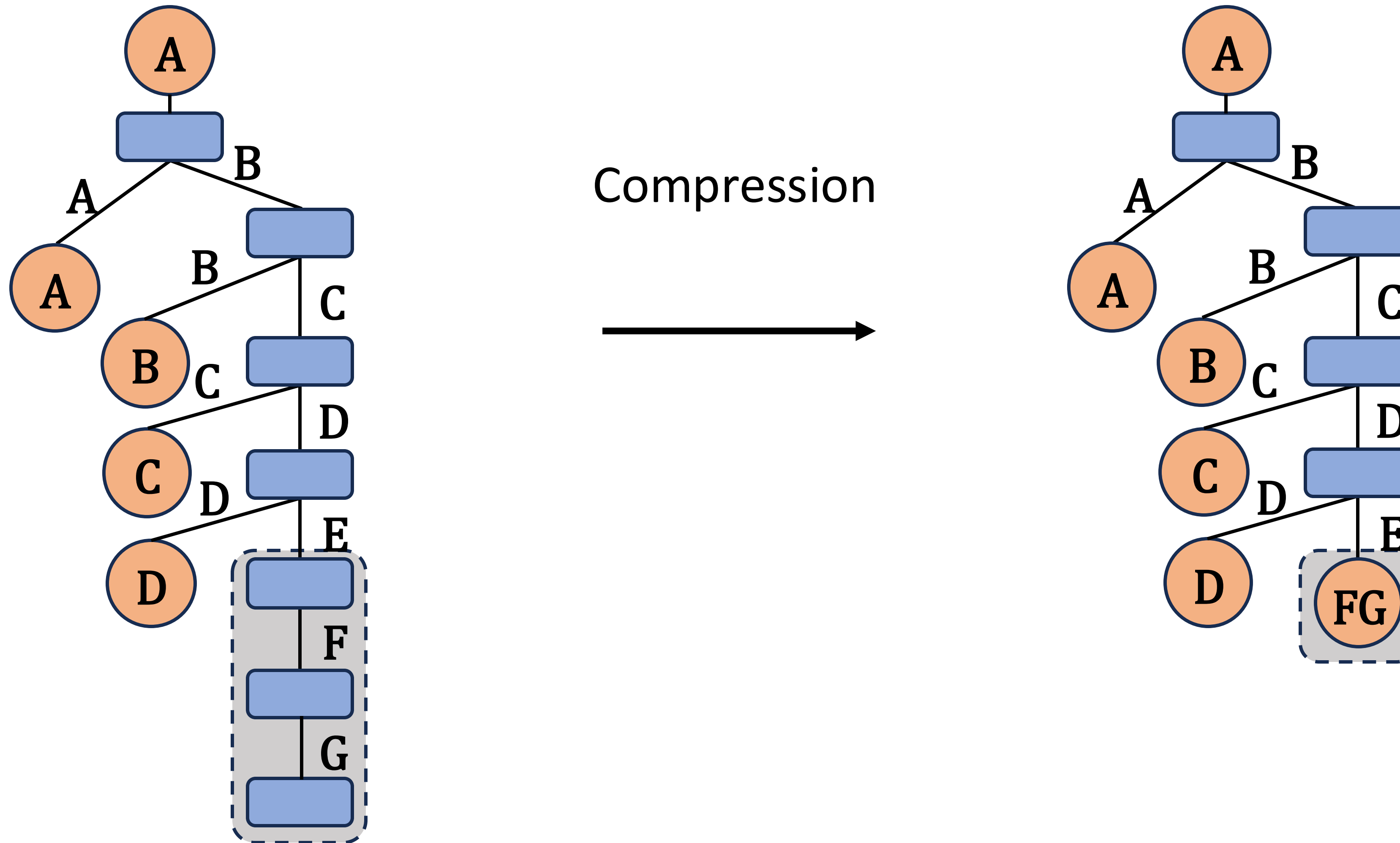
PhaseNU: Phase 2 – Inserting Phase

Key Idea: Proactively expanding nodes and eliminating locks.



PhaseNU: Phase 3 – Compressing Phase

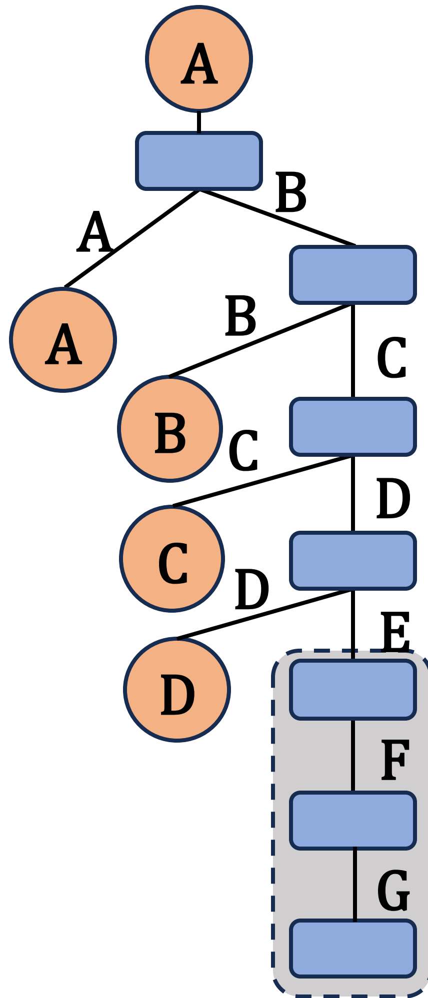
Key Idea: Proactively expanding nodes and eliminating locks.



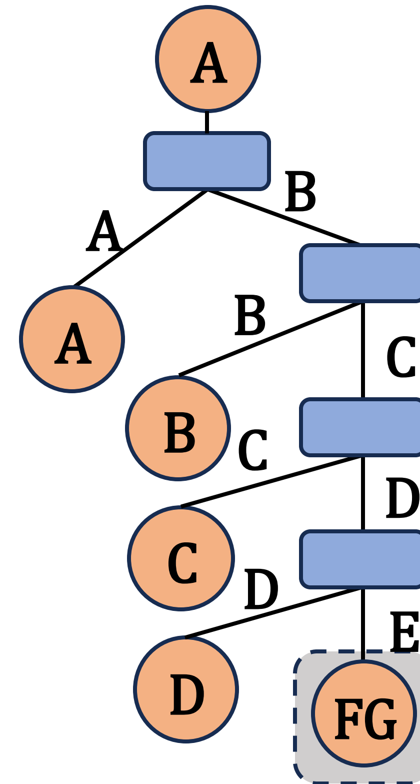
PhaseNU

DBGroup

Limitation: Complexity increases with the length of key.



Compression



Solution

Node Update

- PhaseNU
- **LockNU**

Hash Compute

- PhaseHC

LockNU

DBGroup

Key Idea: Optimistic lock coupling, inspired by Vicktor Leis.[1]

[1] Viktor Leis, Florian Scheibner, Alfons Kemper, and Thomas Neumann. 2016. The ART of practical synchronization. In DaMoN ’16.

LockNU

DBGroup

Key Idea: Optimistic lock coupling, inspired by Vicktor Leis.[1]

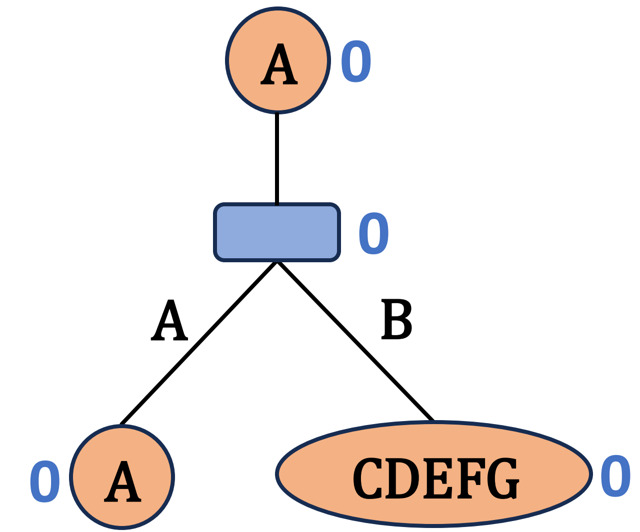
- Read with **optimistic read lock**
  - Get the **version** when lock
  - Verify the **version** when unlock
- Write with **pessimistic write lock**

[1] Viktor Leis, Florian Scheibner, Alfons Kemper, and Thomas Neumann. 2016. The ART of practical synchronization. In DaMoN '16.

LockNU: An Example of Inserting ABCDEEE

Key Idea: Optimistic lock coupling.

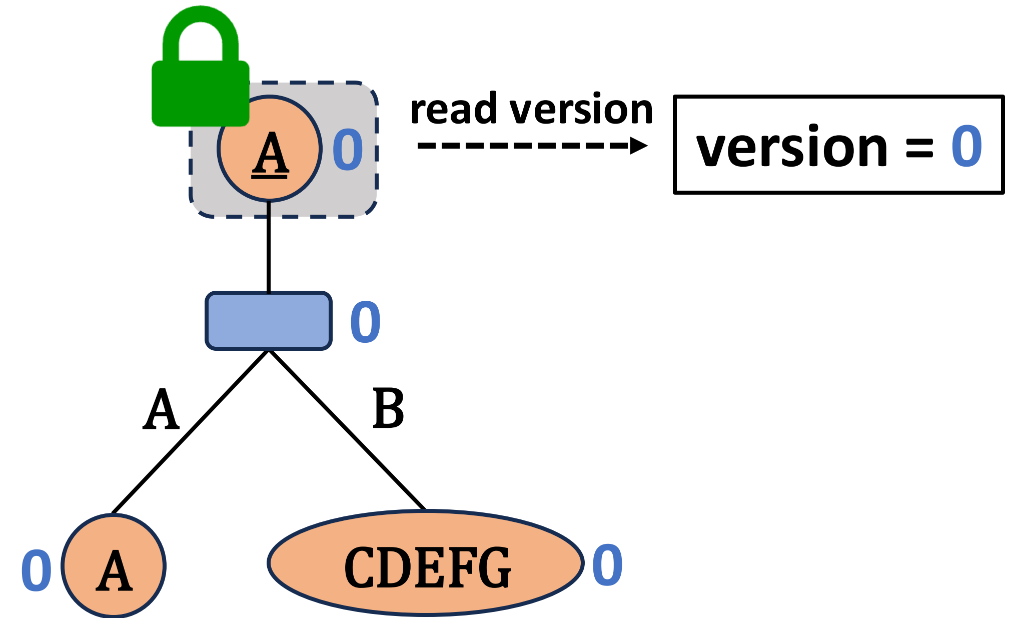
- Read with **optimistic read lock**
  - Get the **version** when lock
  - Verify the **version** when unlock
- Write with **pessimistic write lock**



LockNU: An Example of Inserting ABCDEEE

Key Idea: Optimistic lock coupling.

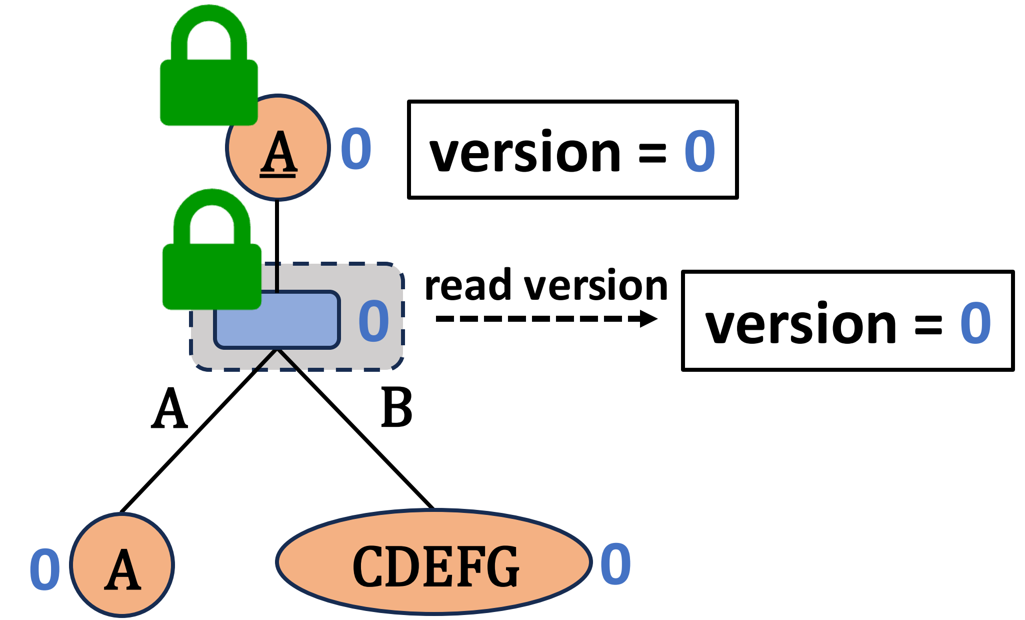
- Read with **optimistic read lock**
  - Get the **version** when lock
  - Verify the **version** when unlock
- Write with **pessimistic write lock**



LockNU: An Example of Inserting ABCDEEE

Key Idea: Optimistic lock coupling.

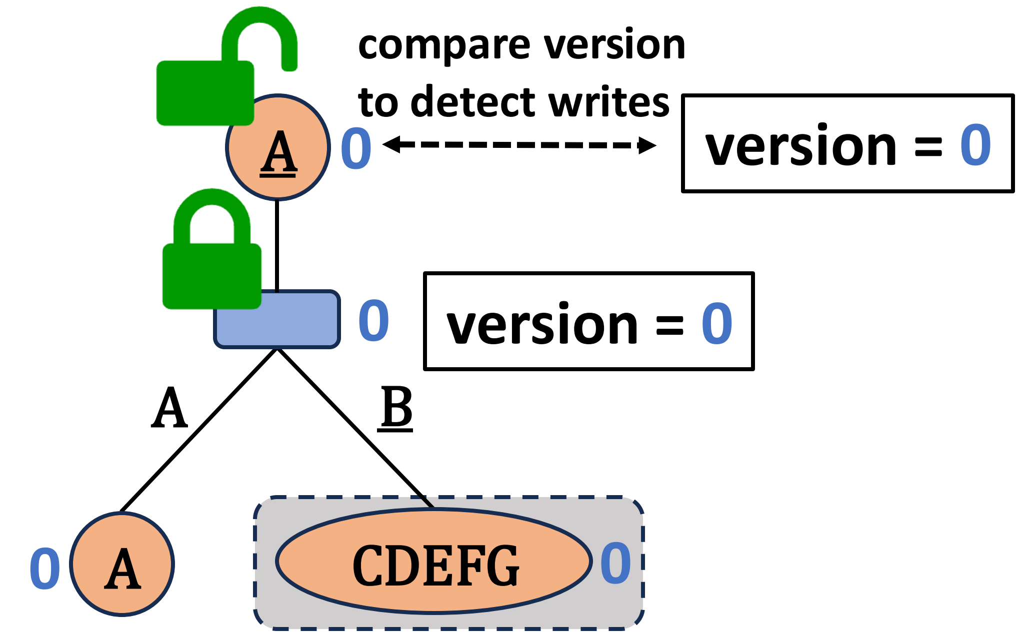
- Read with **optimistic read lock**
  - Get the **version** when lock
  - Verify the **version** when unlock
- Write with **pessimistic write lock**



LockNU: An Example of Inserting ABCDEEE

Key Idea: Optimistic lock coupling.

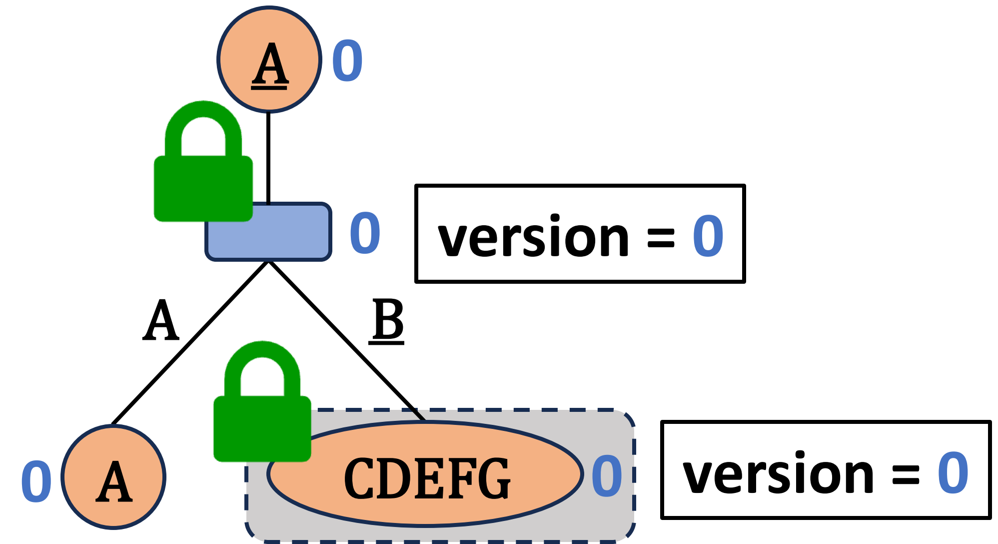
- Read with **optimistic read lock**
  - Get the **version** when lock
  - Verify the **version** when unlock
- Write with **pessimistic write lock**



LockNU: An Example of Inserting ABCDEEE

Key Idea: Optimistic lock coupling.

- Read with **optimistic read lock**
  - Get the **version** when lock
  - Verify the **version** when unlock
- Write with **pessimistic write lock**

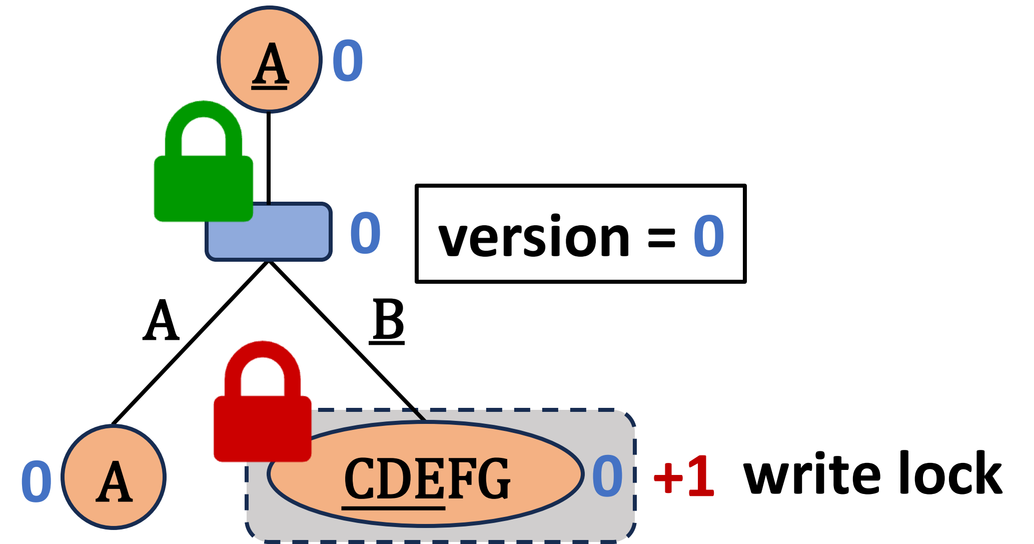


LockNU: An Example of Inserting ABCDEEE

DBGroup

Key Idea: Optimistic lock coupling.

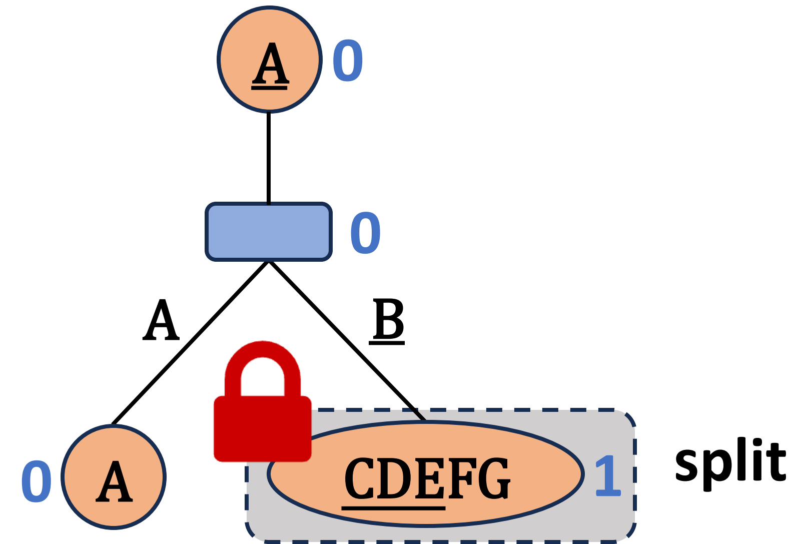
- Read with **optimistic read lock**
  - Get the **version** when lock
  - Verify the **version** when unlock
- Write with **pessimistic write lock**



LockNU: An Example of Inserting ABCDEEE

Key Idea: Optimistic lock coupling.

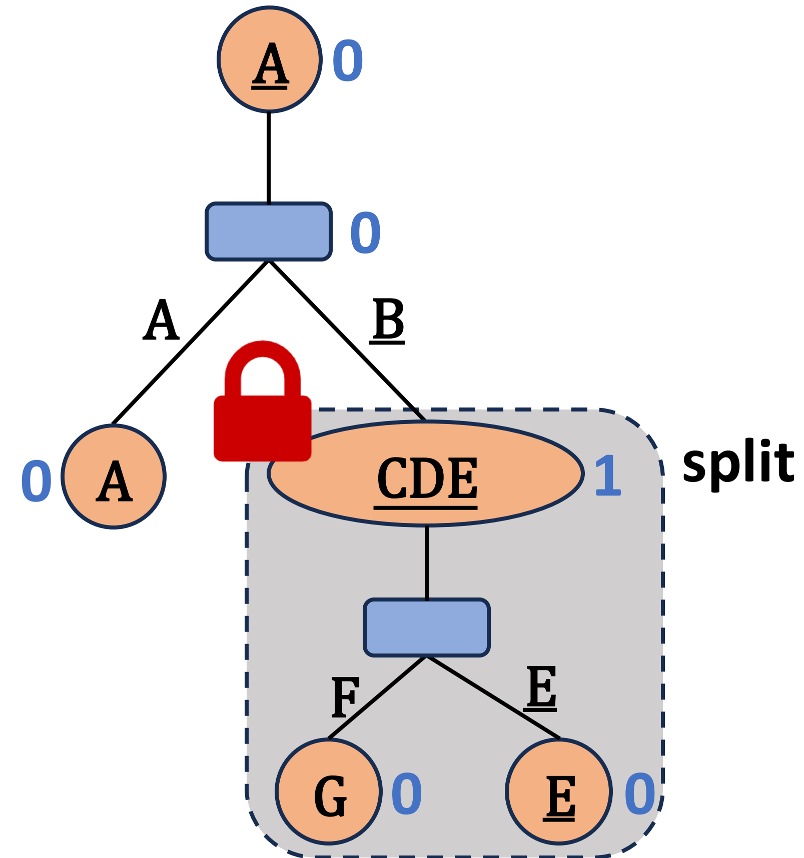
- Read with **optimistic read lock**
  - Get the **version** when lock
  - Verify the **version** when unlock
- Write with **pessimistic write lock**



LockNU: An Example of Inserting ABCDEEE

Key Idea: Optimistic lock coupling.

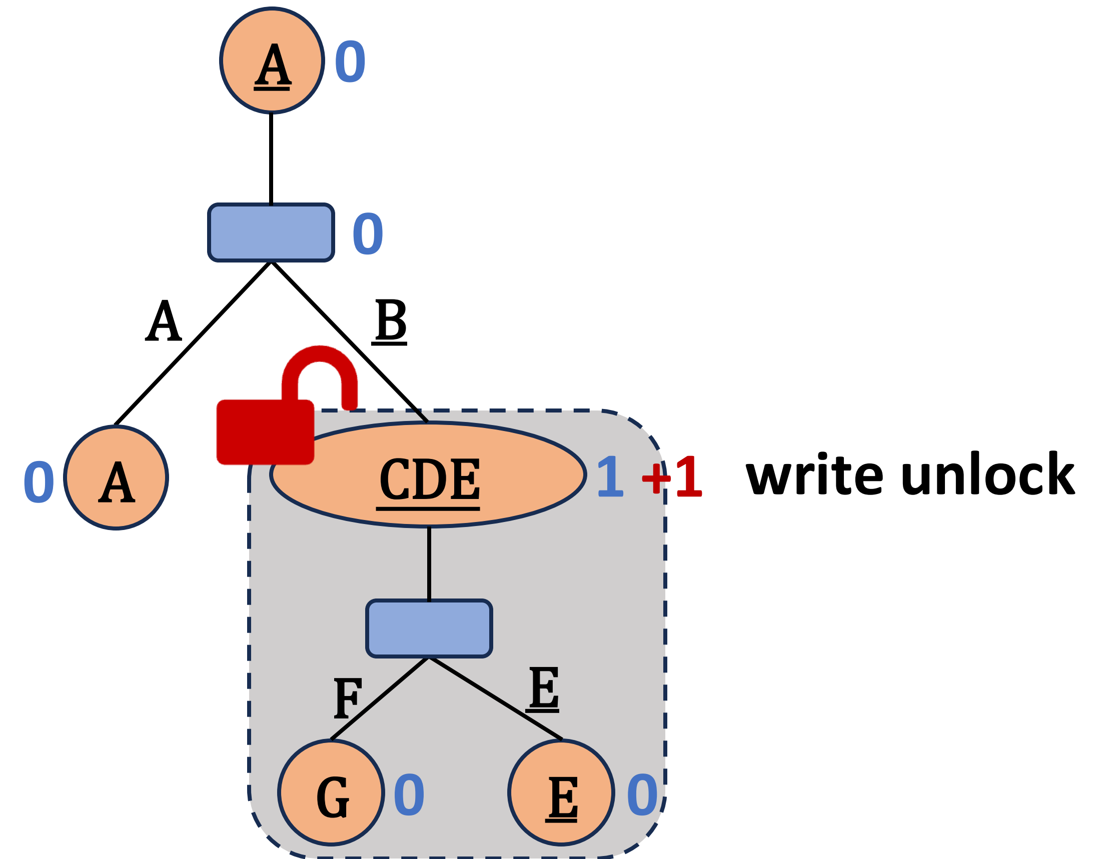
- Read with **optimistic read lock**
  - Get the **version** when lock
  - Verify the **version** when unlock
- Write with **pessimistic write lock**



LockNU: An Example of Inserting ABCDEEE

Key Idea: Optimistic lock coupling.

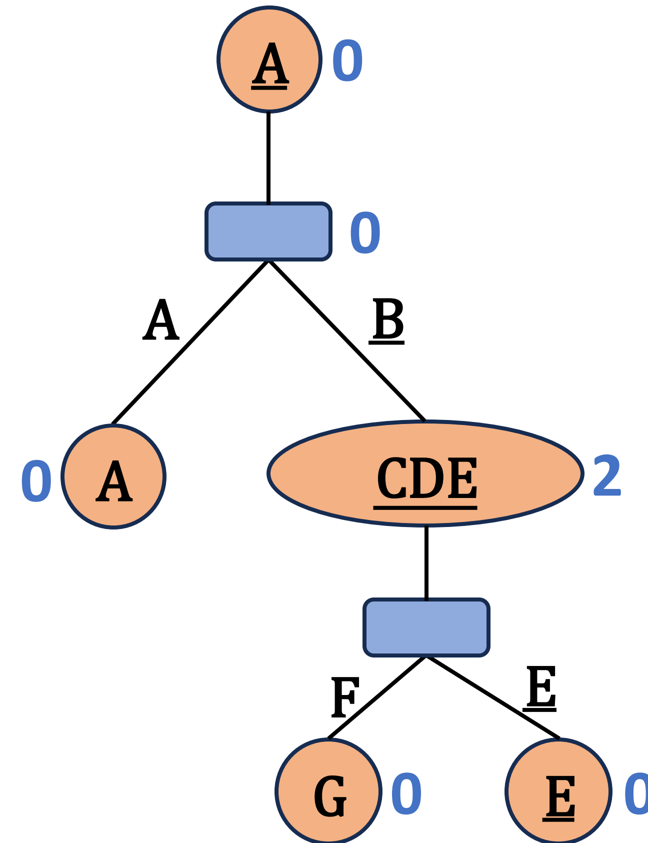
- Read with **optimistic read lock**
  - Get the **version** when lock
  - Verify the **version** when unlock
- Write with **pessimistic write lock**



LockNU: An Example of Inserting ABCDEEE

Key Idea: Optimistic lock coupling.

- Read with **optimistic read lock**
  - Get the **version** when lock
  - Verify the **version** when unlock
- Write with **pessimistic write lock**



Solution

Node Update

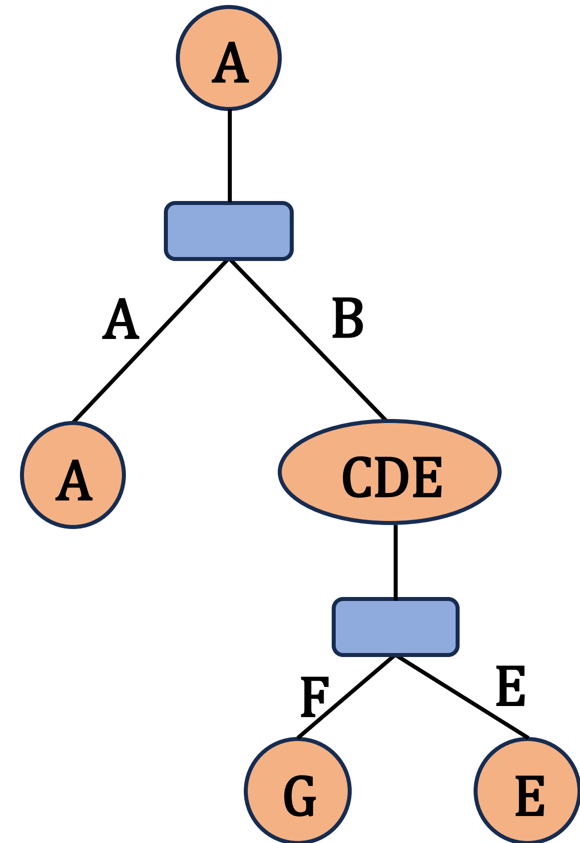
- PhaseNU
- LockNU

Hash Compute

- **PhaseHC**

PhaseHC: Phase 1 – Marking Phase

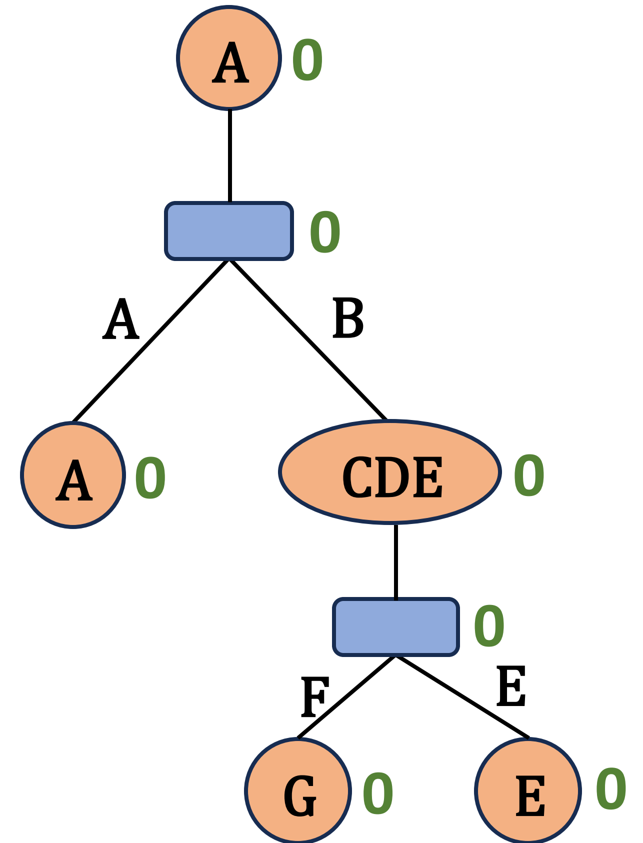
Key Idea: Modeling the hash computation into a **dependency** graph.



PhaseHC: Phase 1 – Marking Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

Dependency counter: **how many hash values it needs?**

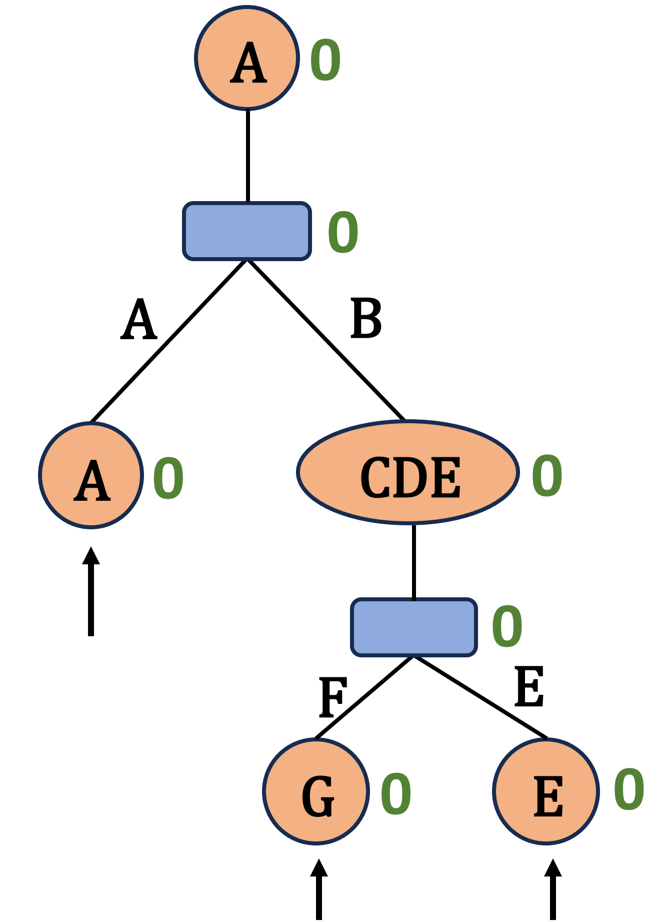


PhaseHC: Phase 1 – Marking Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

Dependency counter: how many hash values it needs?

- Each thread starts from a newly inserted leaf.

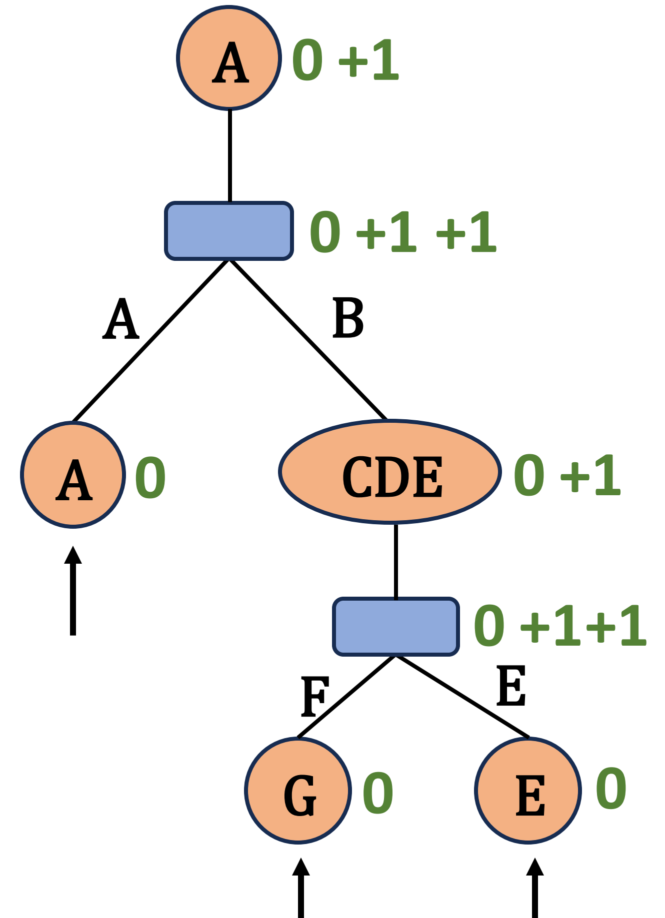


PhaseHC: Phase 1 – Marking Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

Dependency counter: how many hash values it needs?

- Each thread starts from a newly inserted leaf.
- Increment counters in the path. (+1)

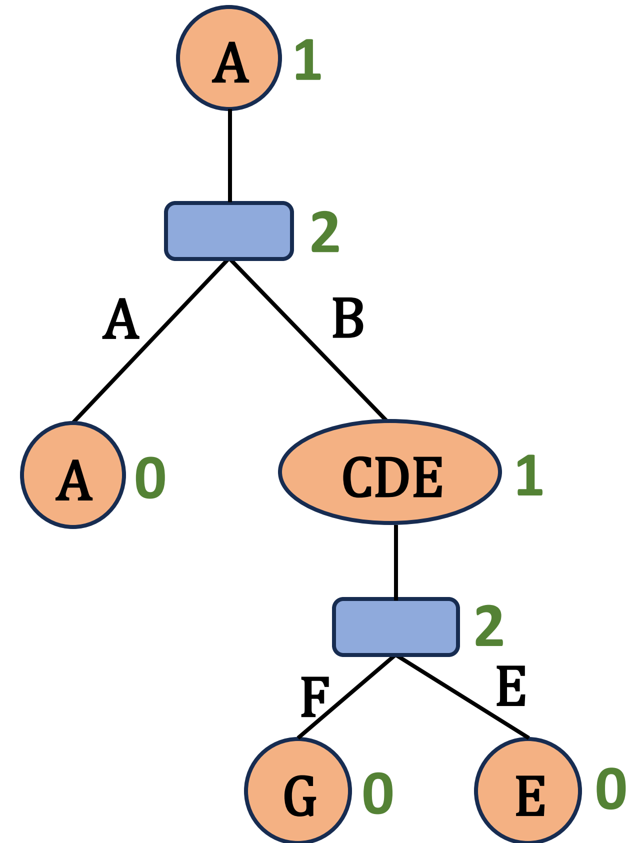


PhaseHC: Phase 1 – Marking Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

Dependency counter: how many hash values it needs?

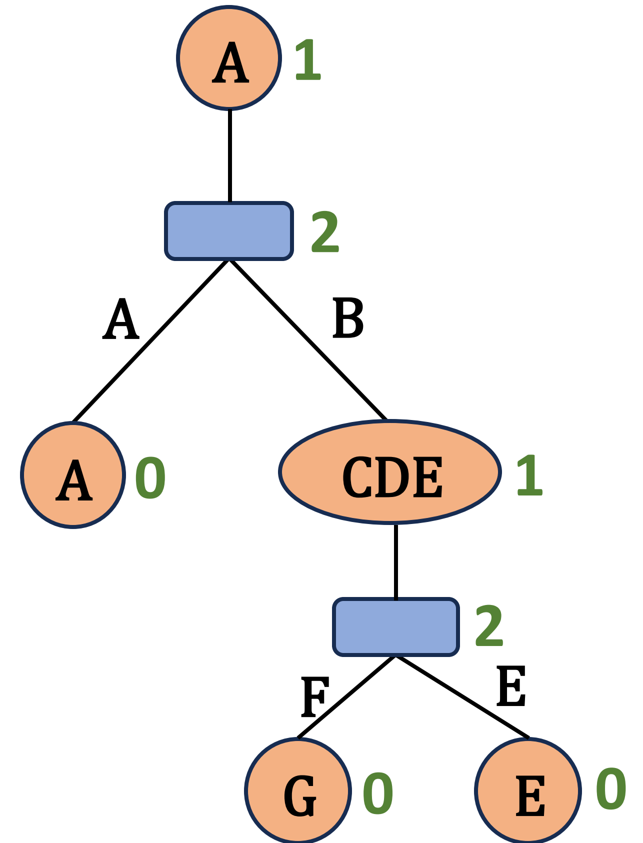
- Each thread starts from a newly inserted leaf.
- Increment counters in the path. (**+1**)



PhaseHC: Phase 2 – Computing Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

The last one arrives takes the job.

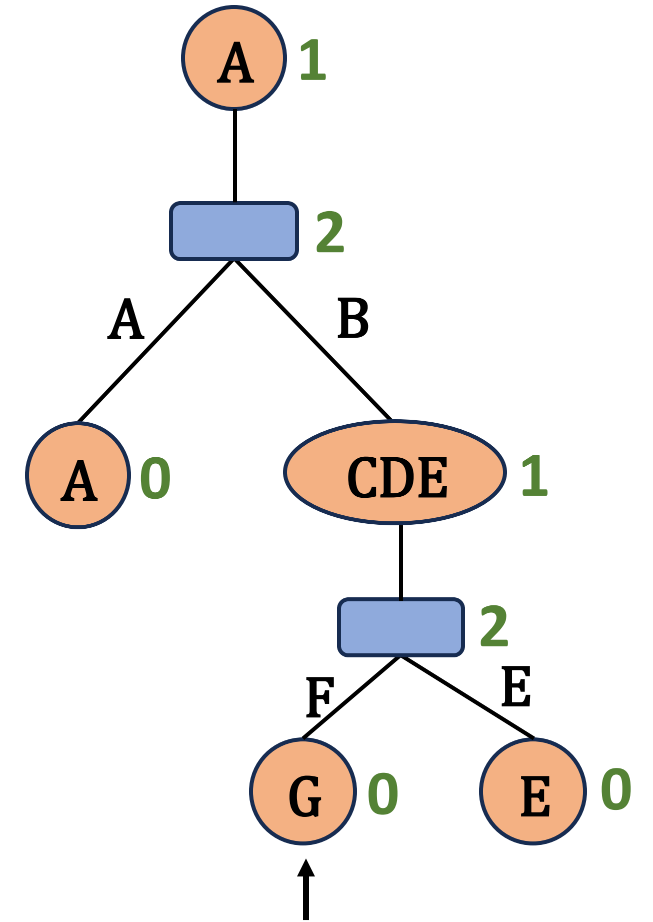


PhaseHC: Phase 2 – Computing Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

The last one arrives takes the job.

- Each thread starts from a newly inserted leaf.

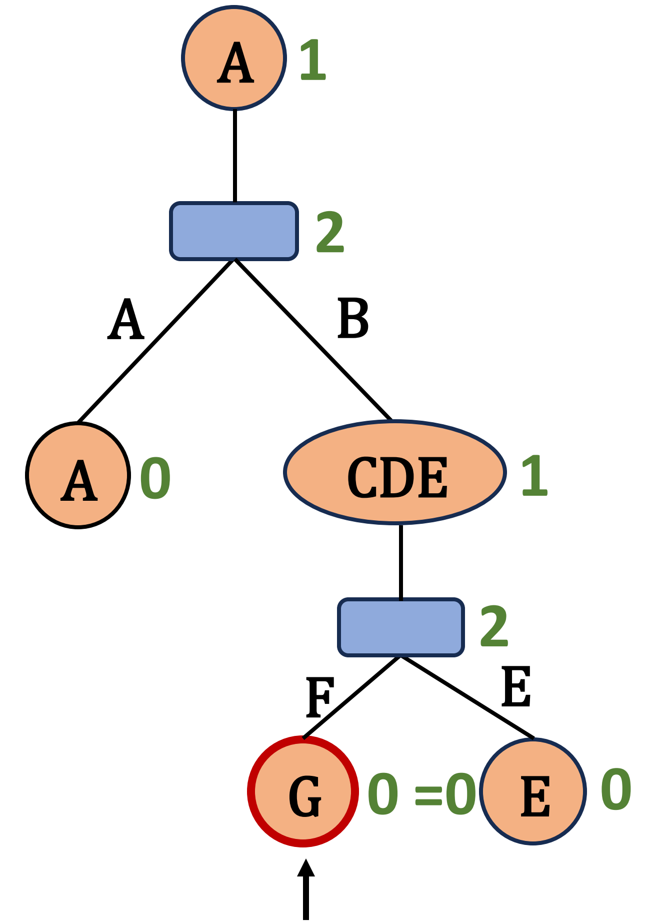


PhaseHC: Phase 2 – Computing Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

The last one arrives takes the job.

- Each thread starts from a newly inserted leaf.
- Proceeds when dependencies are resolved. (=0)

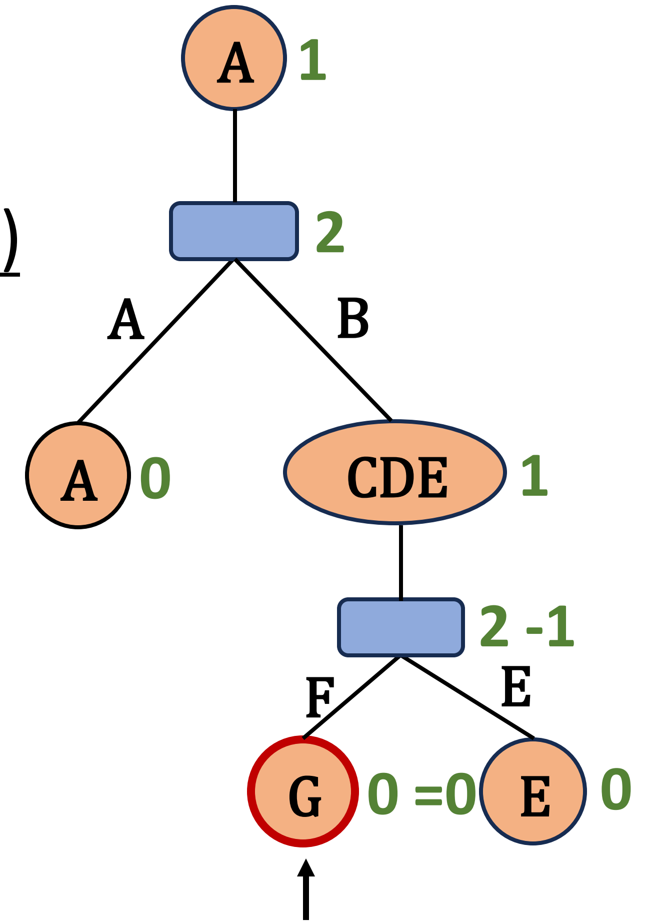


PhaseHC: Phase 2 – Computing Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

The last one arrives takes the job.

- Each thread starts from a newly inserted leaf.
- Proceeds when dependencies are resolved. (=0)
- Compute hash value and decrement the counter (-1)

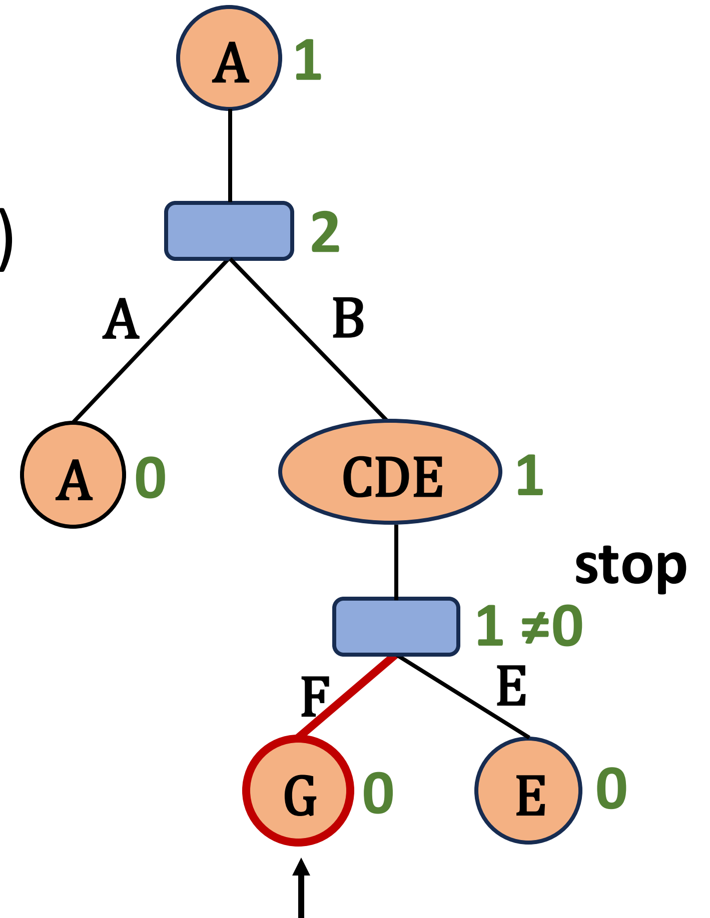


PhaseHC: Phase 2 – Computing Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

The last one arrives takes the job.

- Each thread starts from a newly inserted leaf.
- Proceeds when dependencies are resolved. (=0)
- Compute hash value and decrement the counter (-1)

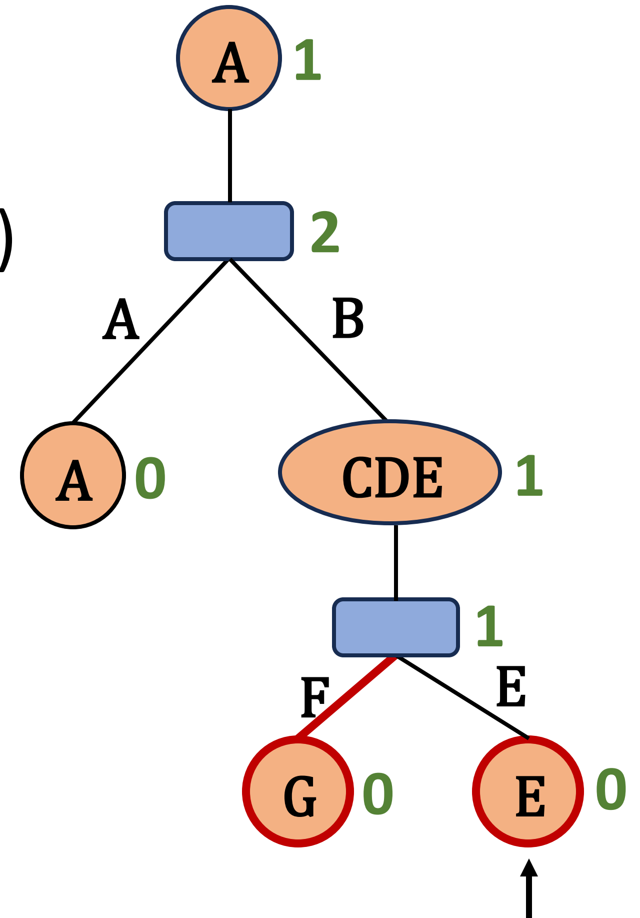


PhaseHC: Phase 2 – Computing Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

The last one arrives takes the job.

- Each thread starts from a newly inserted leaf.
- Proceeds when dependencies are resolved. (=0)
- Compute hash value and decrement the counter (-1)

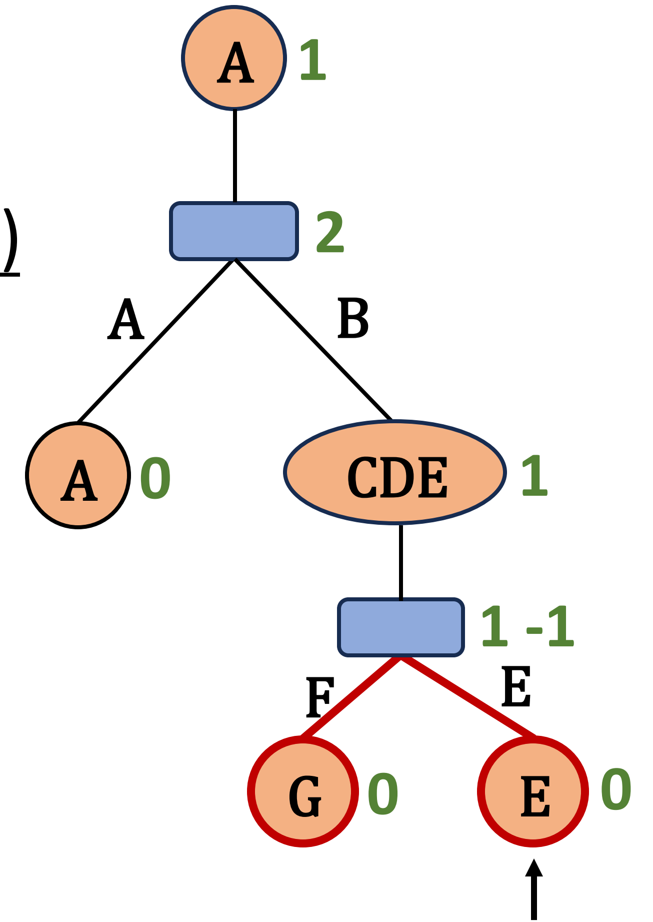


PhaseHC: Phase 2 – Computing Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

The last one arrives takes the job.

- Each thread starts from a newly inserted leaf.
- Proceeds when dependencies are resolved. (=0)
- Compute hash value and decrement the counter (-1)

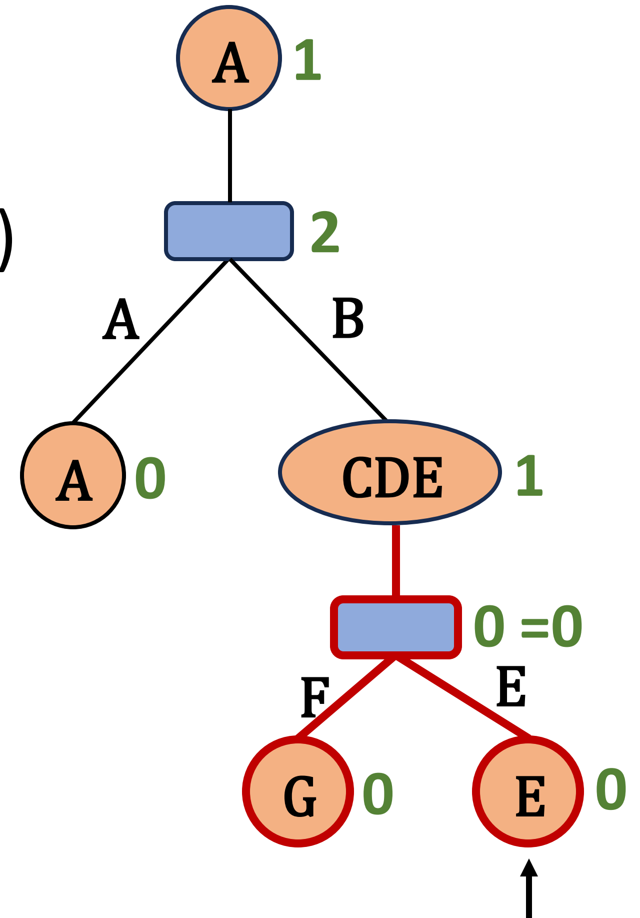


PhaseHC: Phase 2 – Computing Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

The last one arrives takes the job.

- Each thread starts from a newly inserted leaf.
- Proceeds when dependencies are resolved. (=0)
- Compute hash value and decrement the counter (-1)

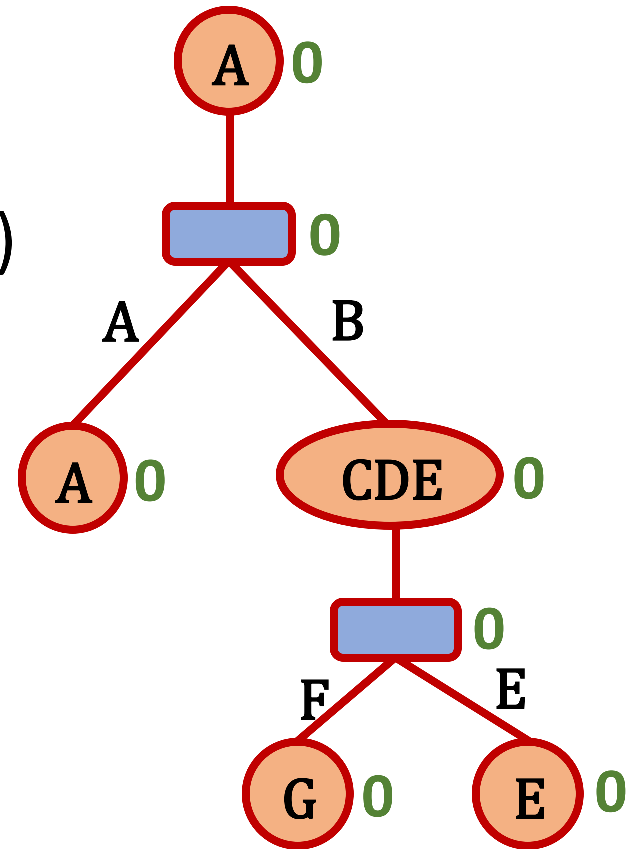


PhaseHC: Phase 2 – Computing Phase

Key Idea: Modeling the hash computation into a **dependency** graph.

The last one arrives takes the job.

- Each thread starts from a newly inserted leaf.
- Proceeds when dependencies are resolved. (=0)
- Compute hash value and decrement the counter (-1)



Overview

- Motivation
- Challenges
- Solutions
- **Results**
- Takeaways

Experiment Setup

- **GPU:** NVIDIA V100
- **CPU:** Intel Xeon Gold 6230R (2 sockets x 26 cores x 2 threads)
- **Baseline system:** Go-Ethereum (Geth)

Index Benchmark: Node Update

GethNU PhaseNU LockNU SpinLC RestartLC

Throughput (qps)

10^5 10^6 10^7 10^8

5 40 160 320 640

Ethereum transactions

10 80 320 640 1280

YCSB

2.5 20 80 160 320

Wiki

Data volume ($x10^3$)

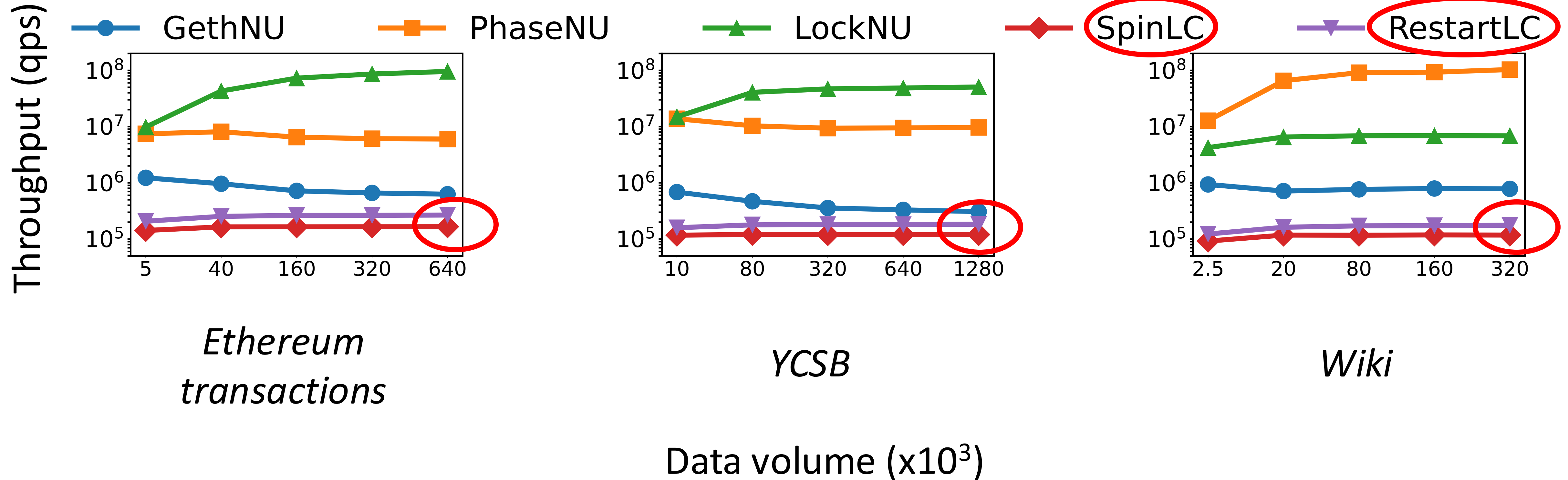
Index Benchmark: Node Update

- **Up to 165x speed up.**
- **LockNU scales better with long keys.**

LockNU or PhaseNU? See our paper for the data-driven decision model!

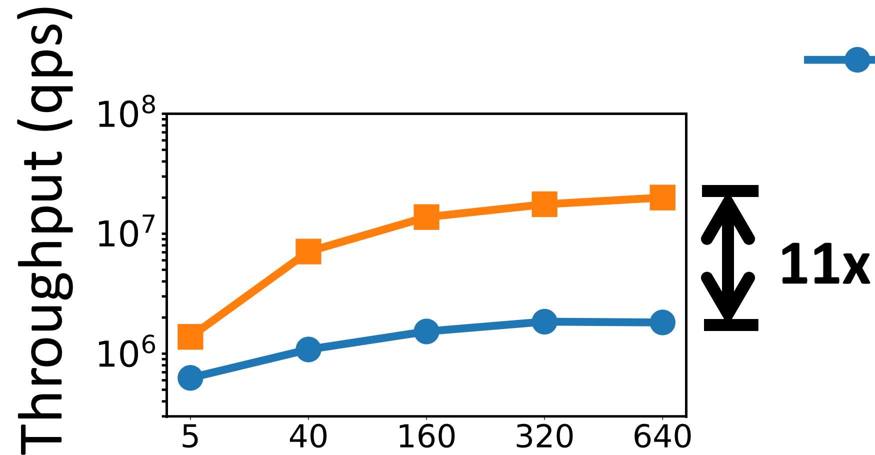
Index Benchmark: Node Update

- **Traditional lock coupling methods do not scale in GPU.**

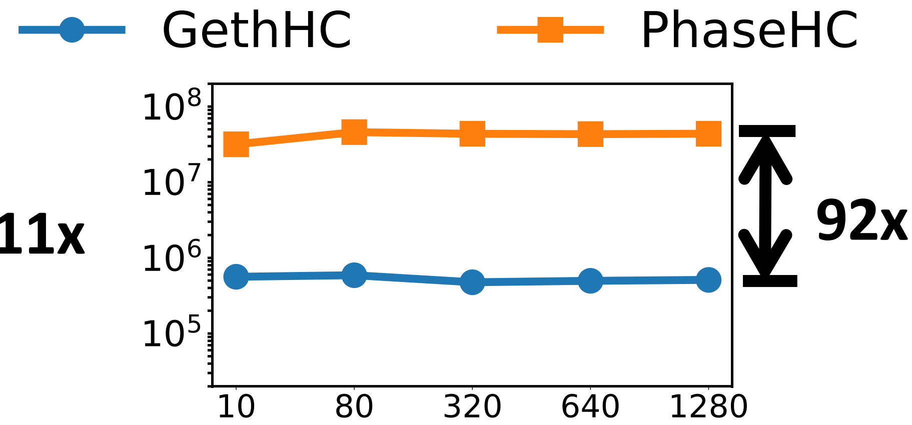


Index Benchmark: Hash Compute

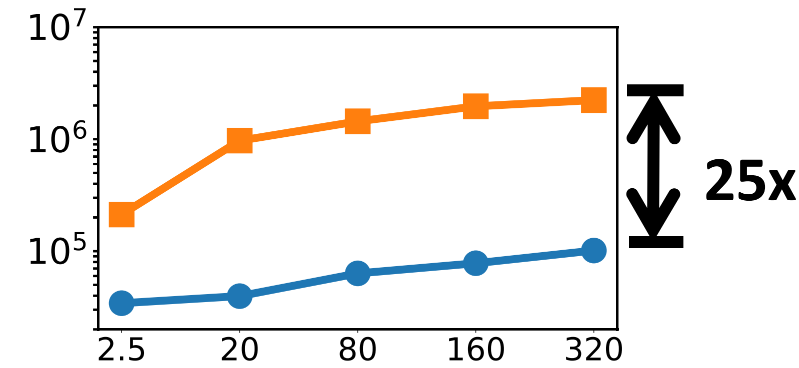
- **Up to 92x speedup.**



Ethereum transactions



YCSB



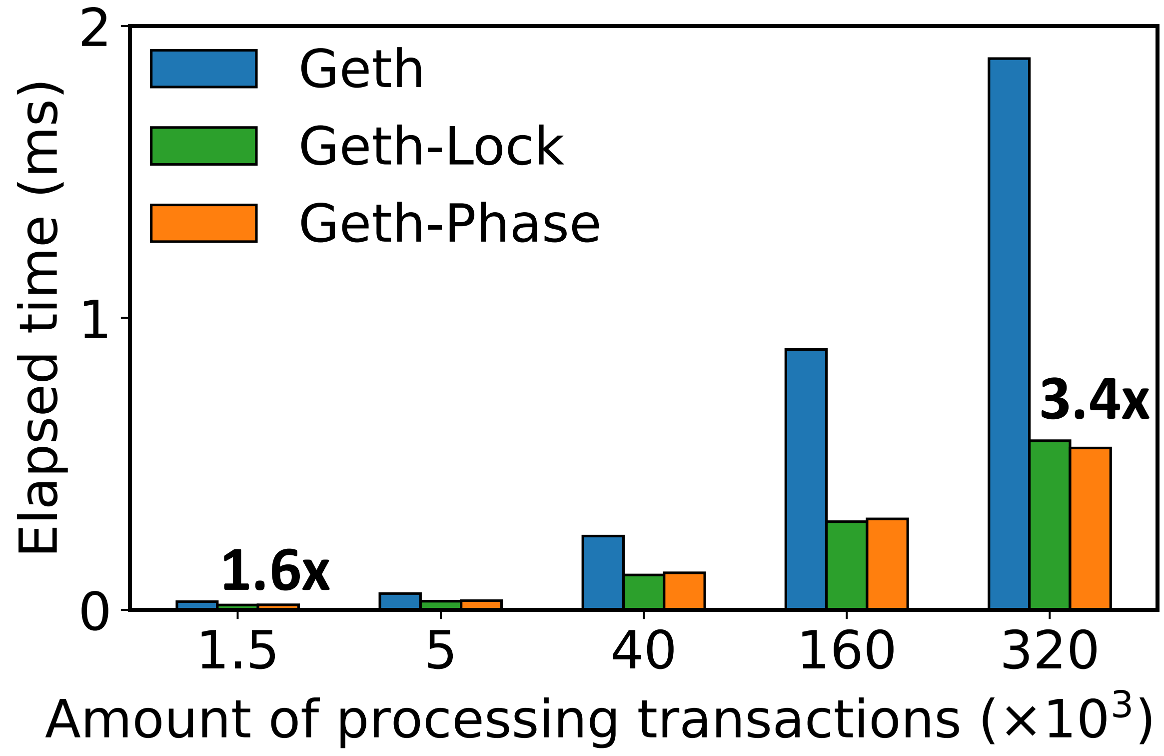
Wiki

Data volume ($\times 10^3$)

Integrating Into Real Systems

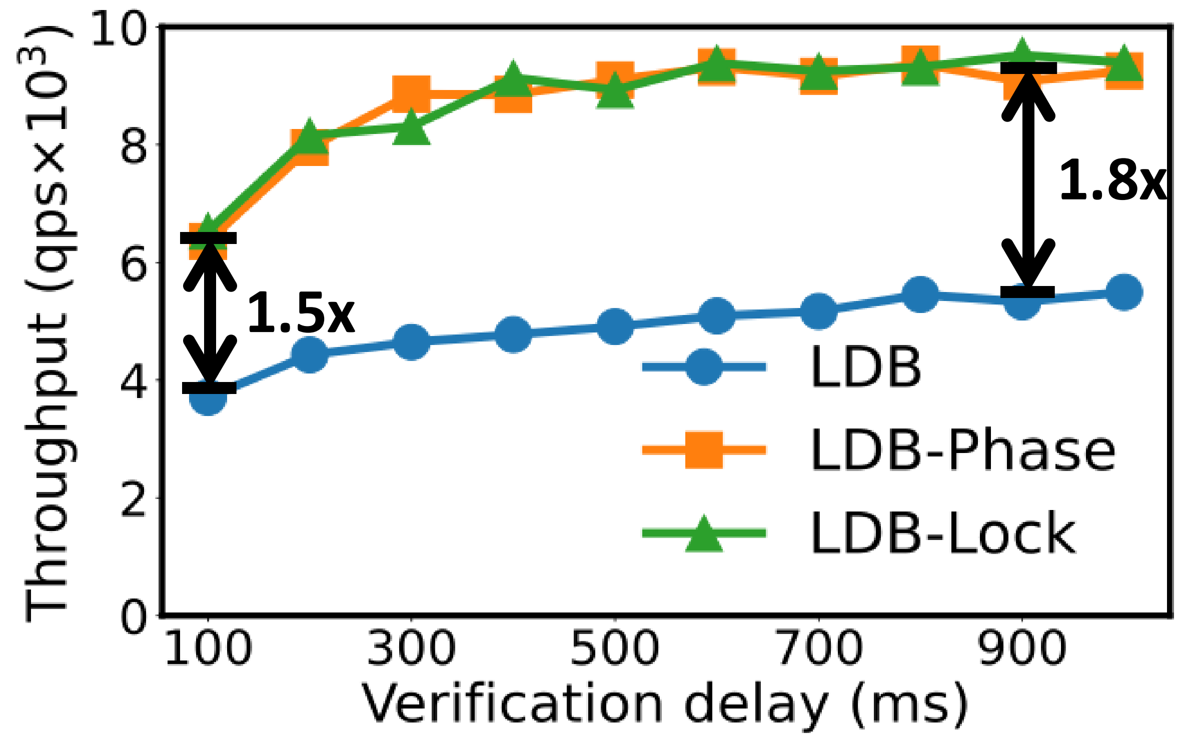
Go-Ethereum

Time to generate a block



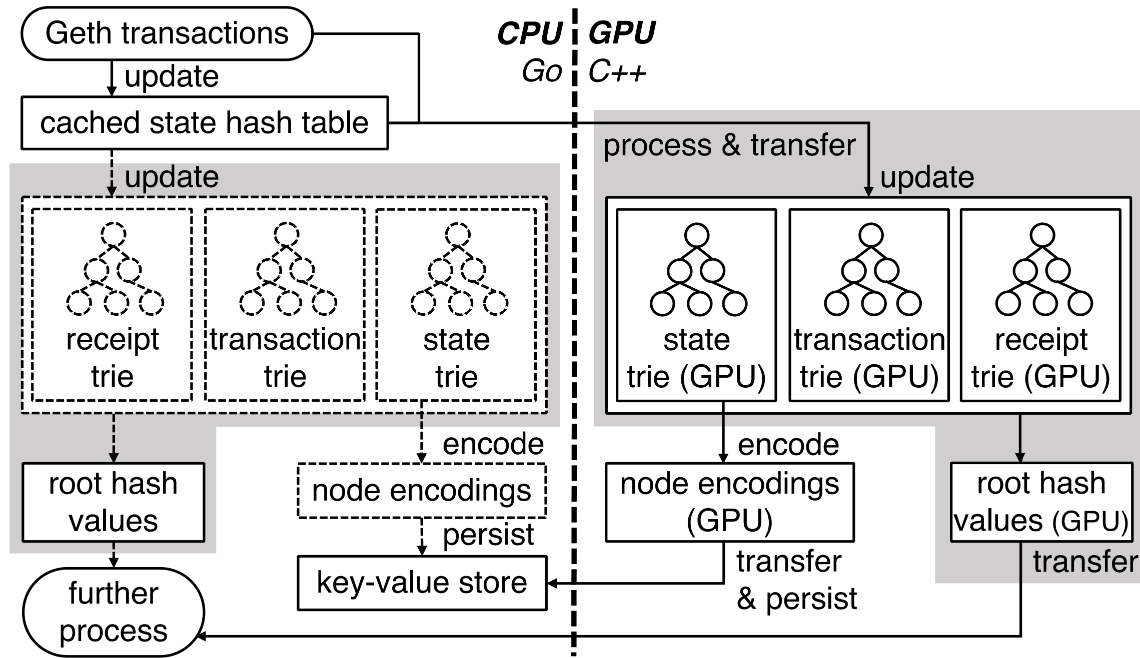
LedgerDB

End-to-end throughput

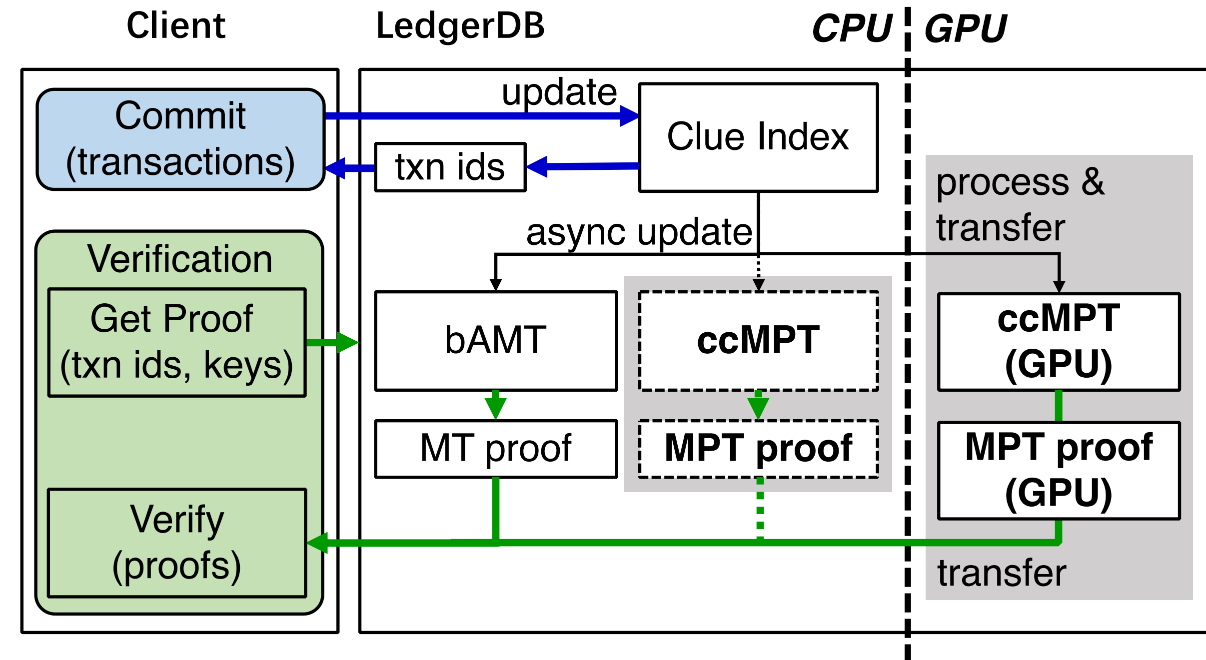


Integrating Into Real Systems

Go-Ethereum



LedgerDB



Overview

- Motivation
- Challenges
- Solution
- Results
- **Takeaways**

Takeaways and Opportunities

DBGroup

Takeaways

- Updating MPT on the GPU is **100+** times faster than on the CPU.
- Offloading MPT to GPU can improve the system's throughput.
- Lock-base and lock-free methods are suitable for different input distributions.

Thank you

Email: dbgroup@sustech.edu.cn

Code: https://github.com/DBGroup-SUSTech/GPUDB-Prefetch