Fast Multi-Level Locks for Java

Khilan Gudka
Imperial College London

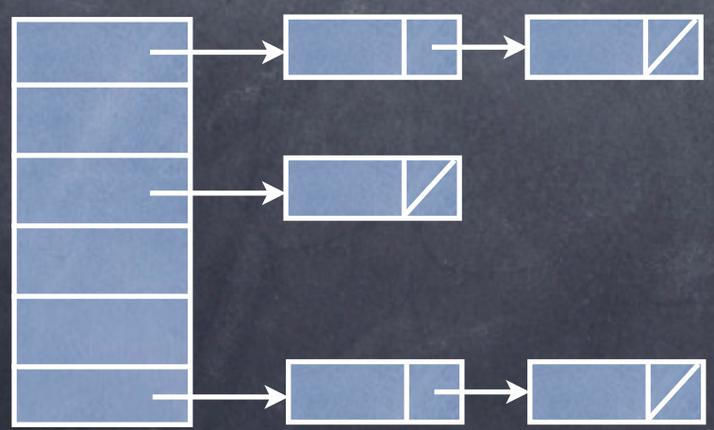
Supervised by
Susan Eisenbach
Sophia Drossopoulou

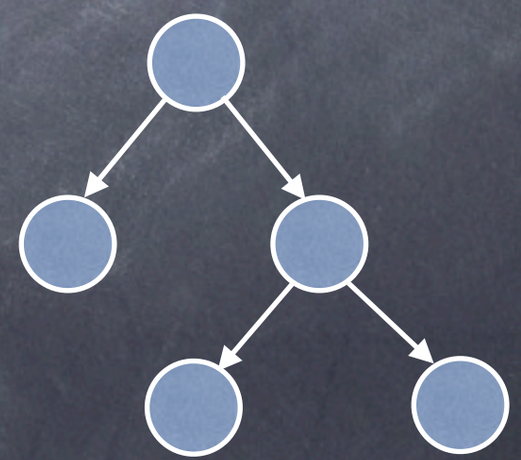
Imperial College London

Microsoft Research

Hierarchical Data Structures

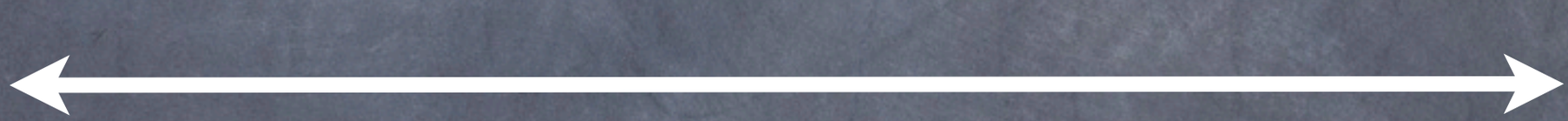
- Databases - tables, rows, cells
- Trees - subtree, leaf
- Hashtables - table, chain, entries





Accesses

- Operations may access differing amounts of data



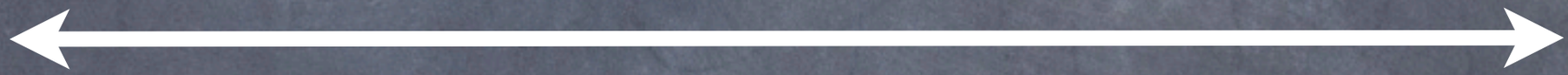
"lots of data" "little data"

- e.g. Tree - access individual leaf nodes vs. all nodes in subtree

Accesses

- Operations may access differing amounts of data

coarse — fine



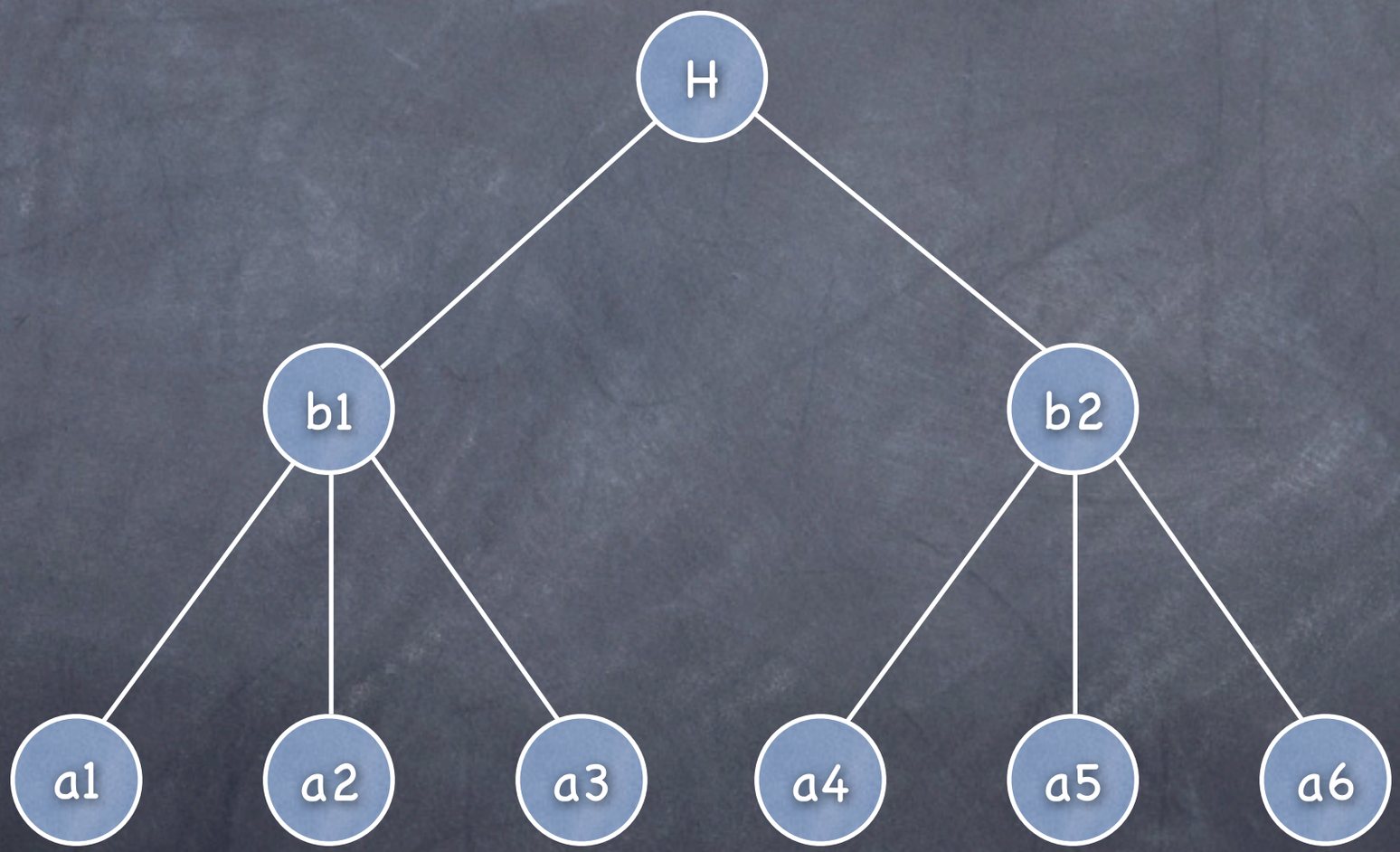
"lots of data" — "little data"

- e.g. Tree - access individual leaf nodes vs. all nodes in subtree

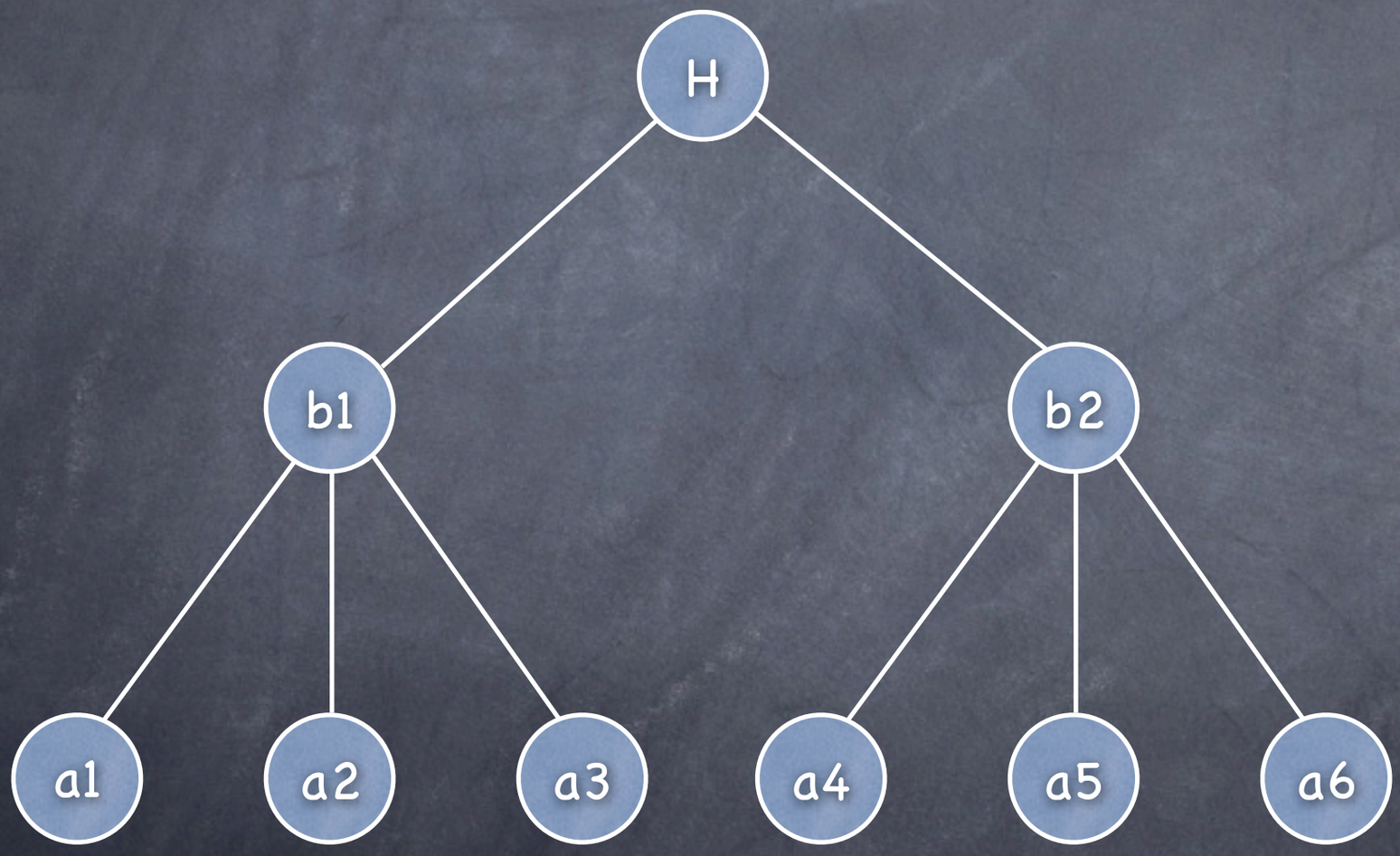
Concurrent Accesses => Concurrency Control

- Lock data before accessing
- Lock granularity - how much data a lock protects
- Trade off between concurrency and overhead
  - fine-grained - more concurrency, higher overhead for coarse accesses
  - coarse-grained - lower overhead, less concurrency for fine accesses

Hierarchical Bank Account Example

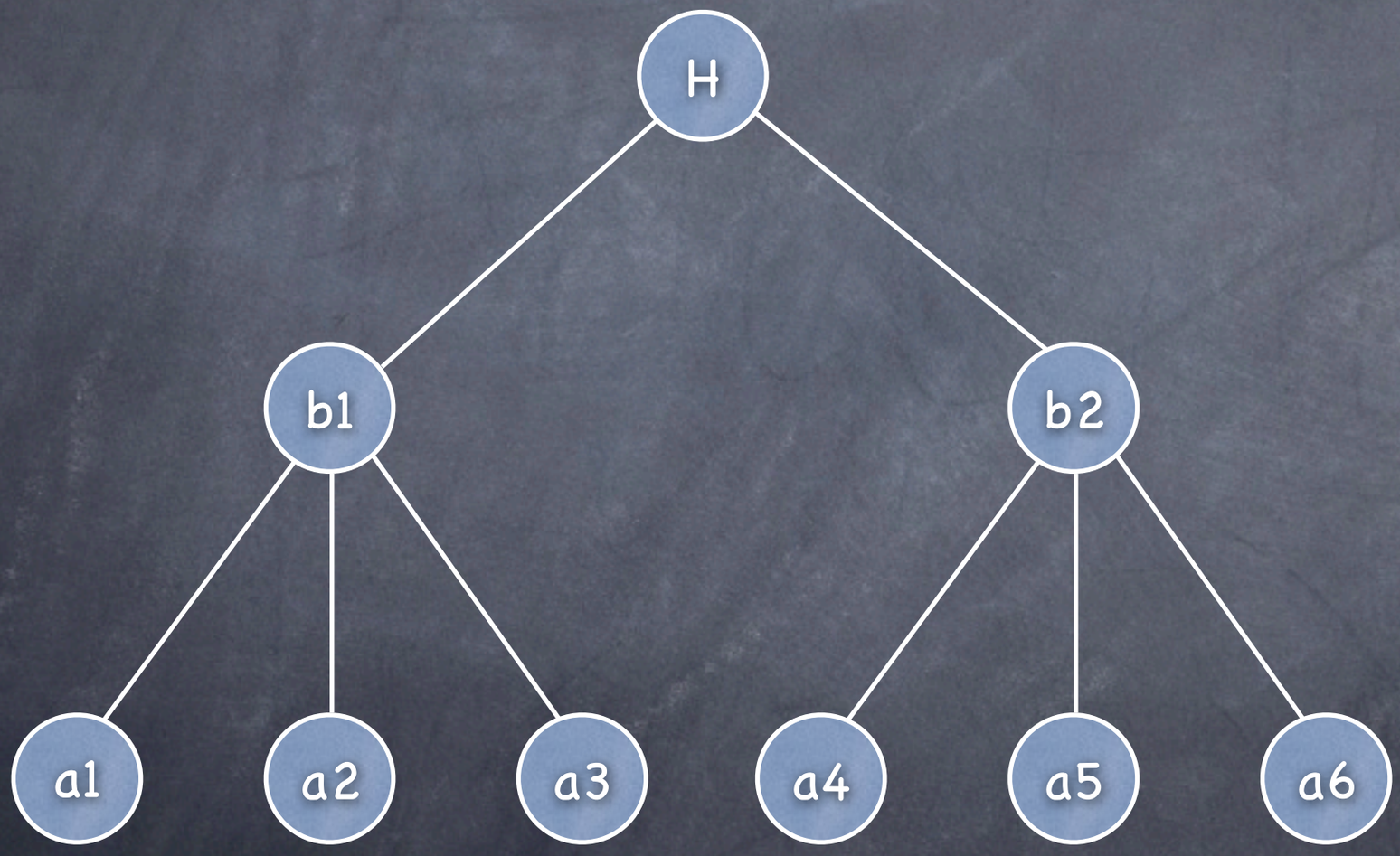


Hierarchical Bank Account Example



- Operations on:
  - Account
  - Branch
  - Whole bank

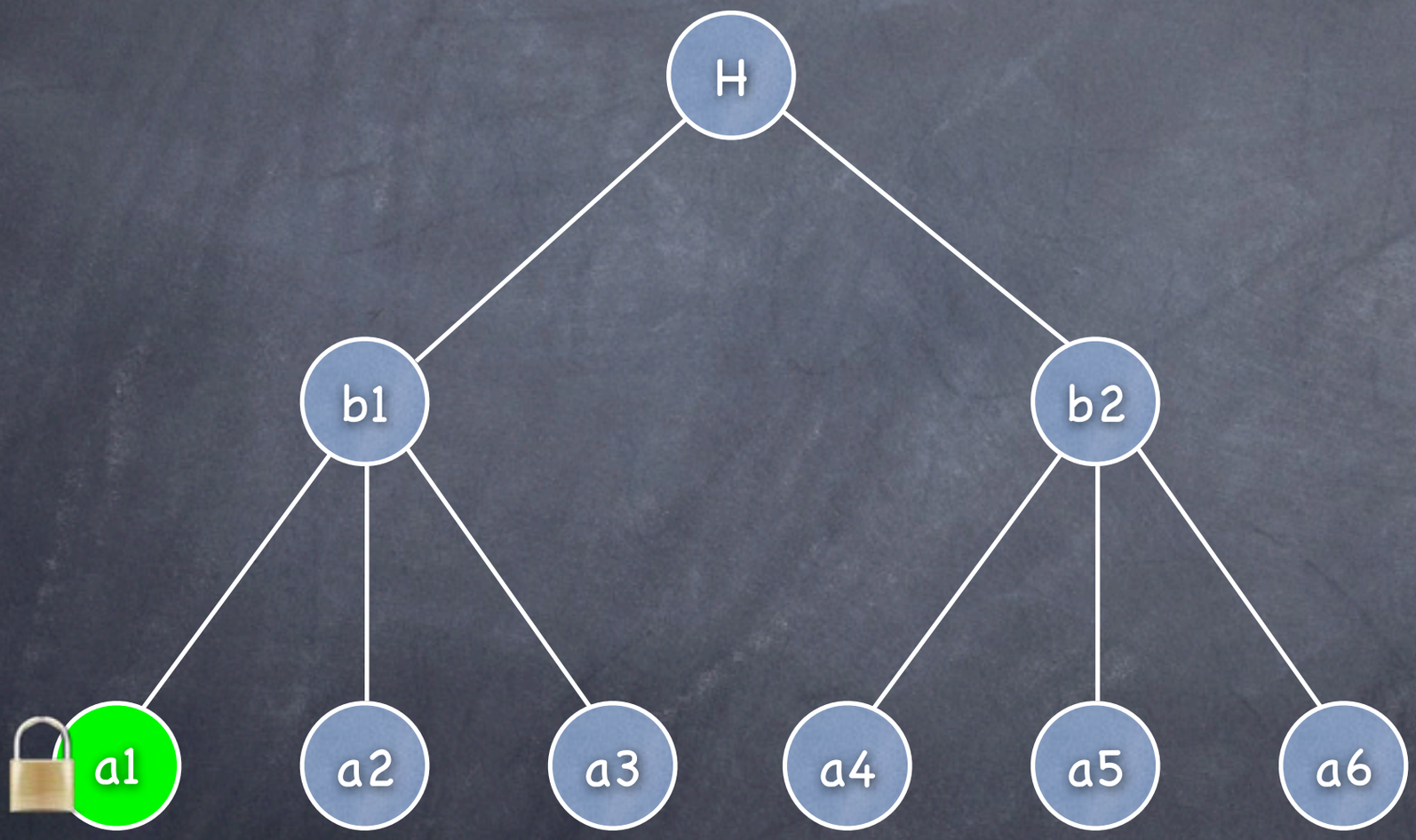
Fine-Grained Accesses



- Operations on:
  - Account
  - Branch
  - Whole bank

Fine-Grained Accesses

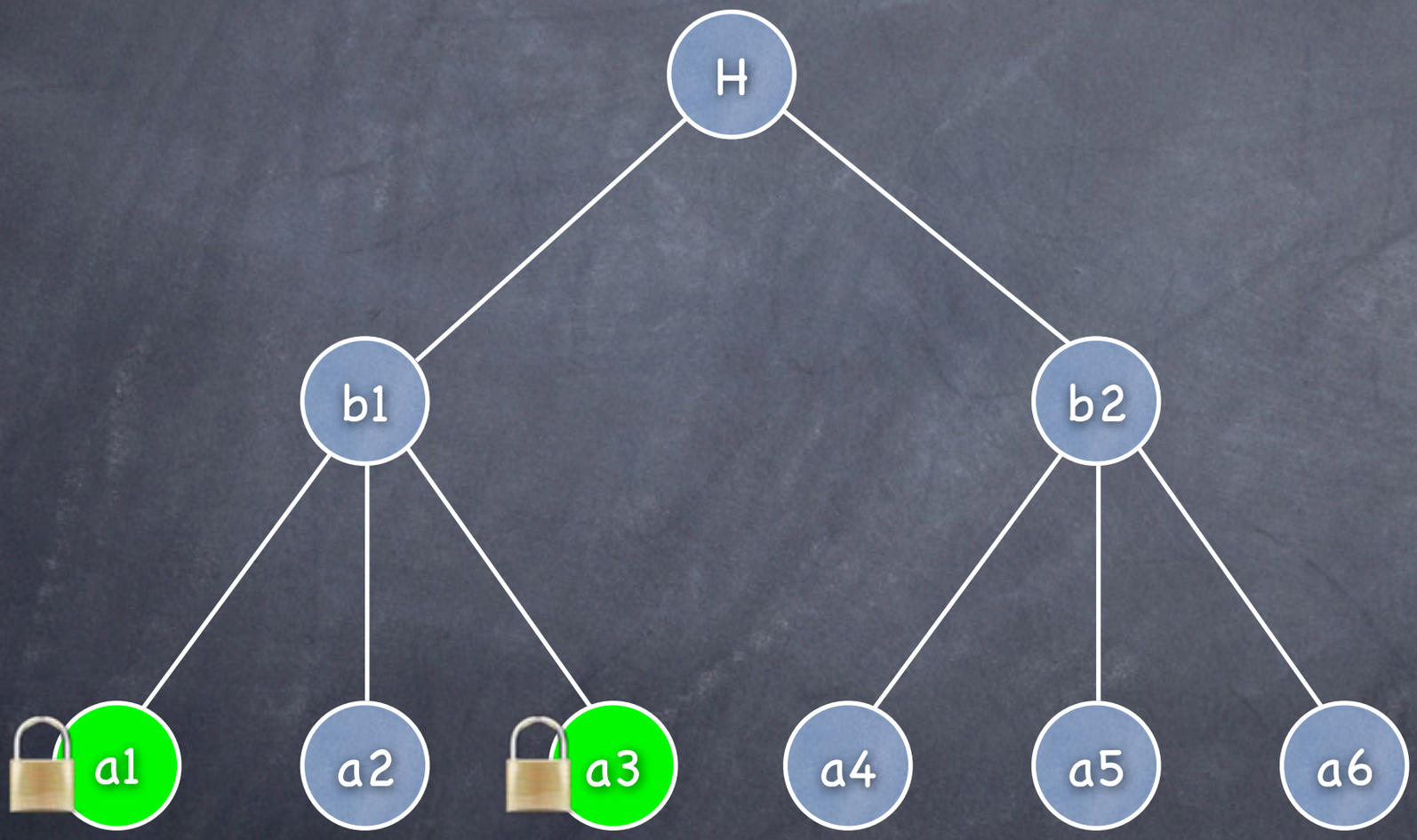
Fine-grained locking => more concurrency



- Operations on:
  - Account
  - Branch
  - Whole bank

Fine-Grained Accesses

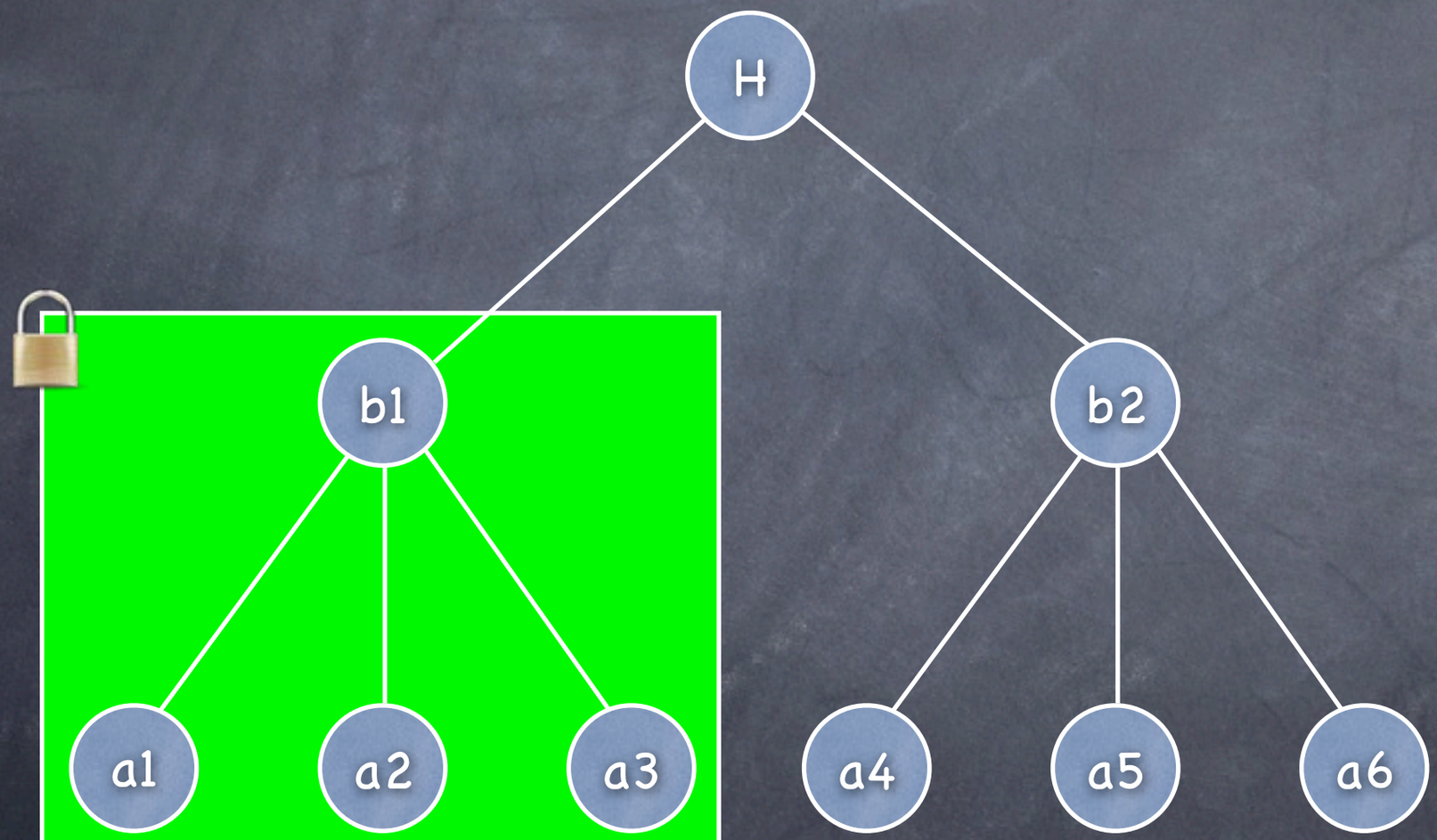
Fine-grained locking => more concurrency



- Operations on:
  - Account
  - Branch
  - Whole bank

Fine-Grained Accesses

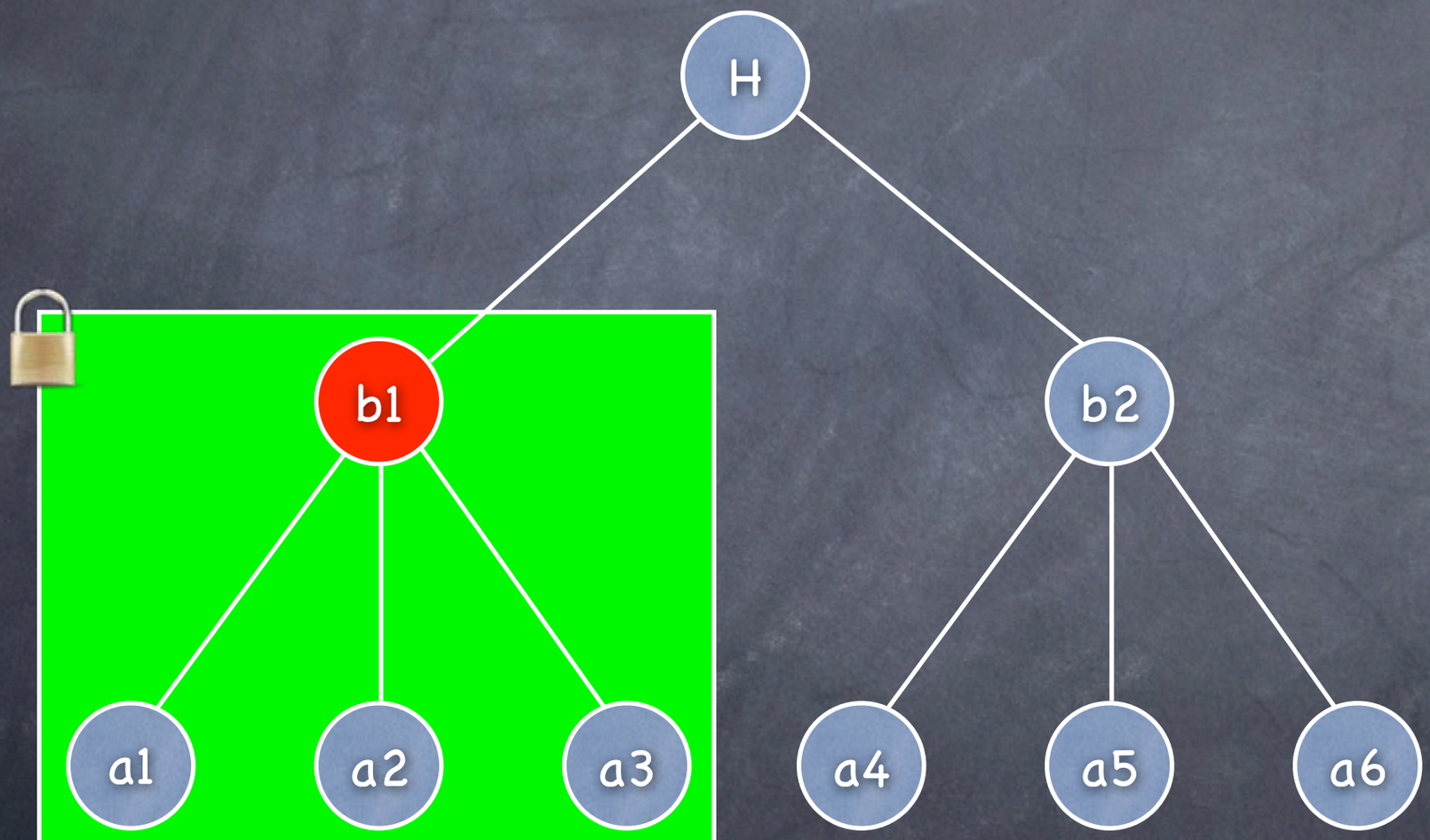
Coarse-grained locking => less concurrency



- Operations on:
- Account
- Branch
- Whole bank

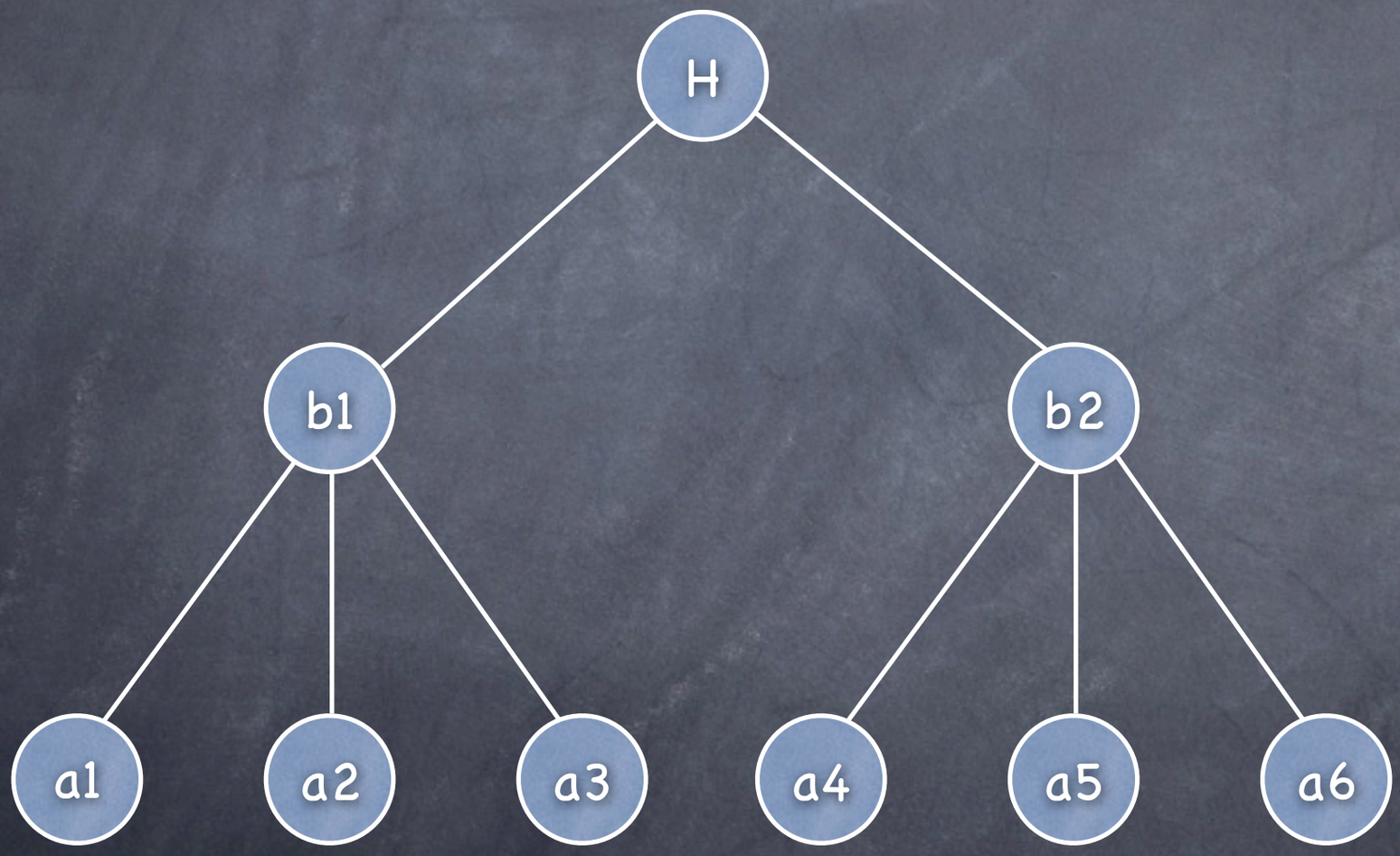
Fine-Grained Accesses

Coarse-grained locking => less concurrency



- Operations on:
- Account
- Branch
- Whole bank

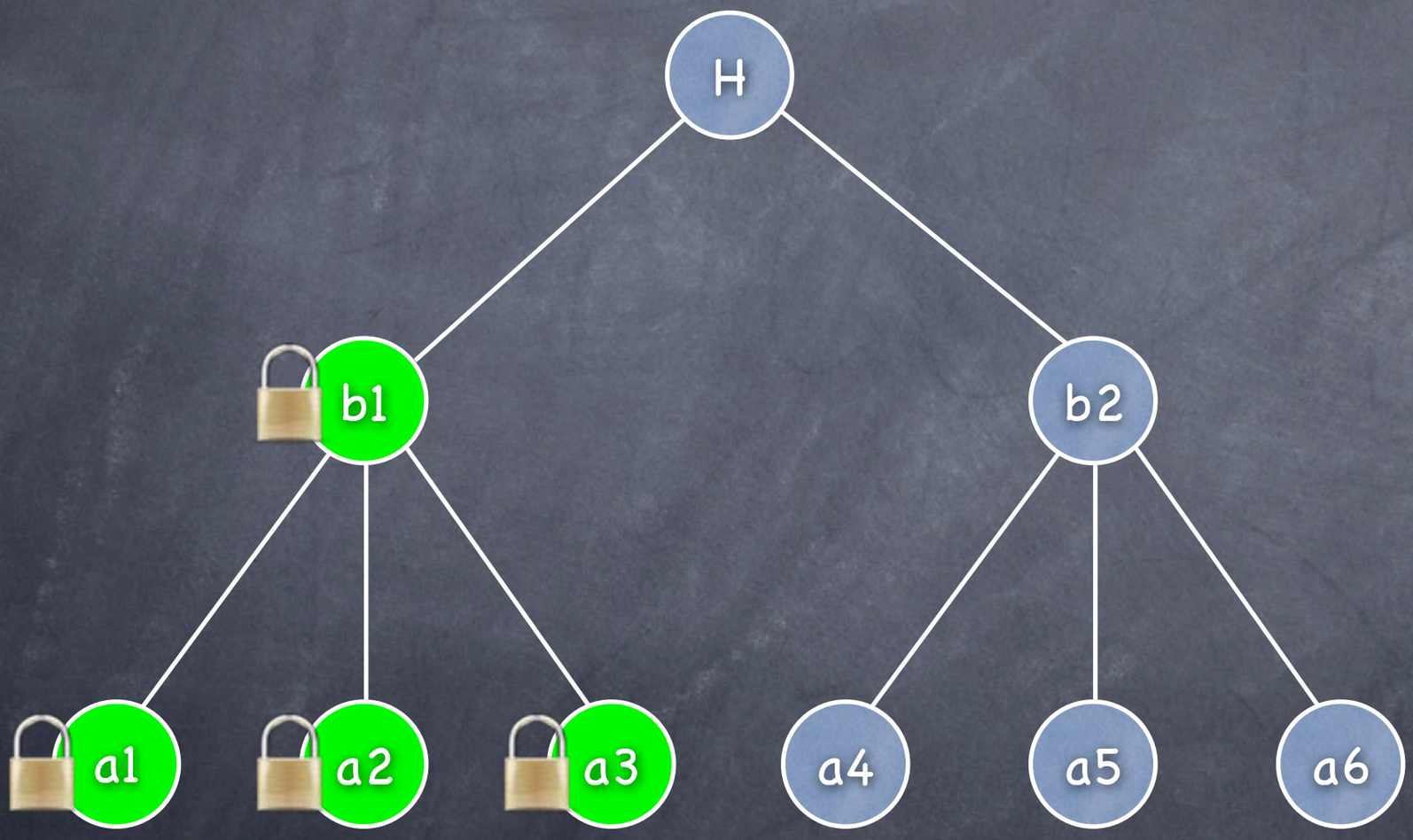
Coarse-Grained Accesses



- Operations on:
  - Account
  - Branch
  - Whole bank

Coarse-Grained Accesses

Fine-grained locking => more overhead

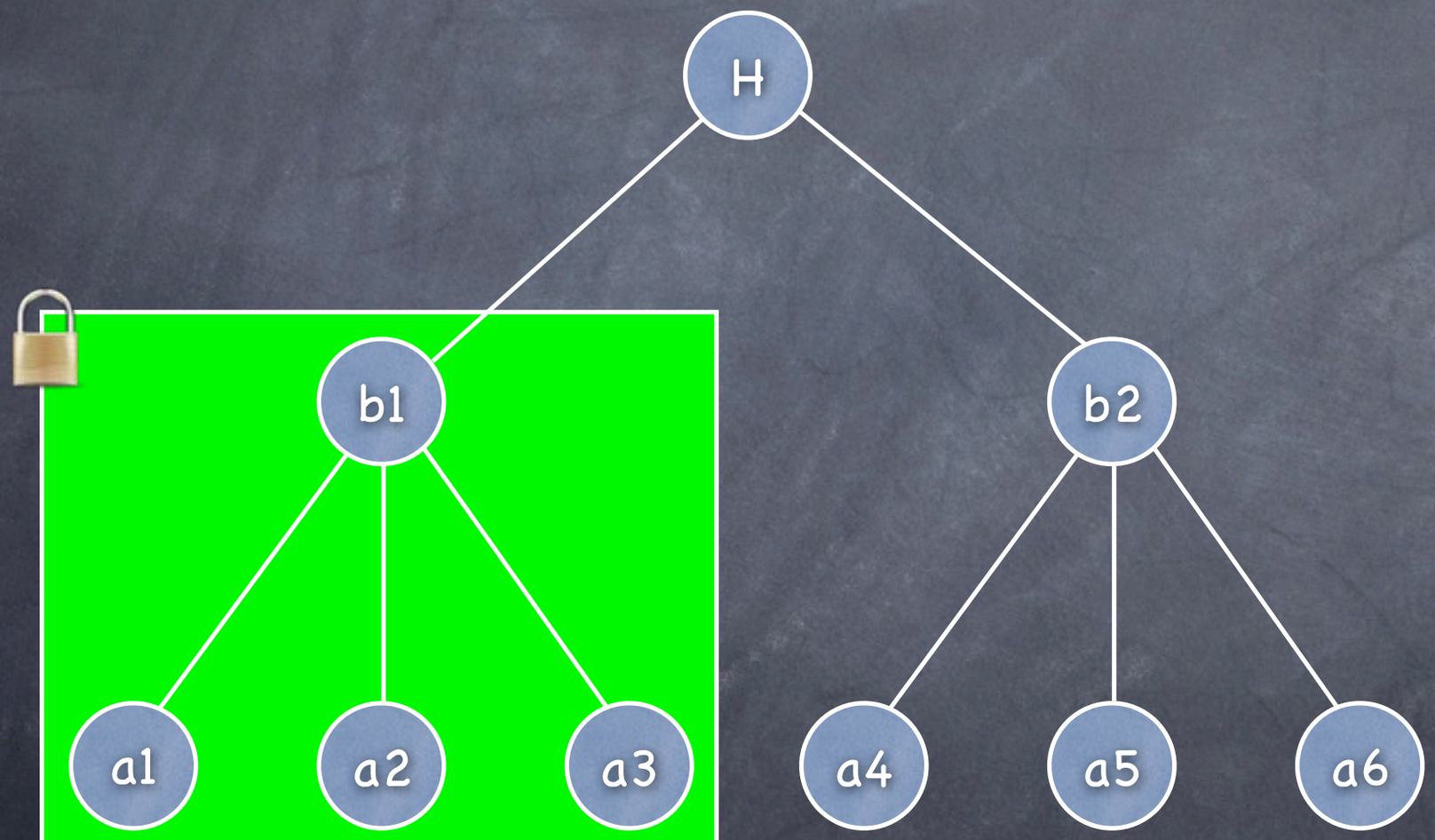


- Operations on:
  - Account
  - Branch
  - Whole bank

4 locks

Coarse-Grained Accesses

Coarse-grained locking => less overhead



- Operations on:
- Account
- Branch
- Whole bank

1 lock

Best of Both Worlds

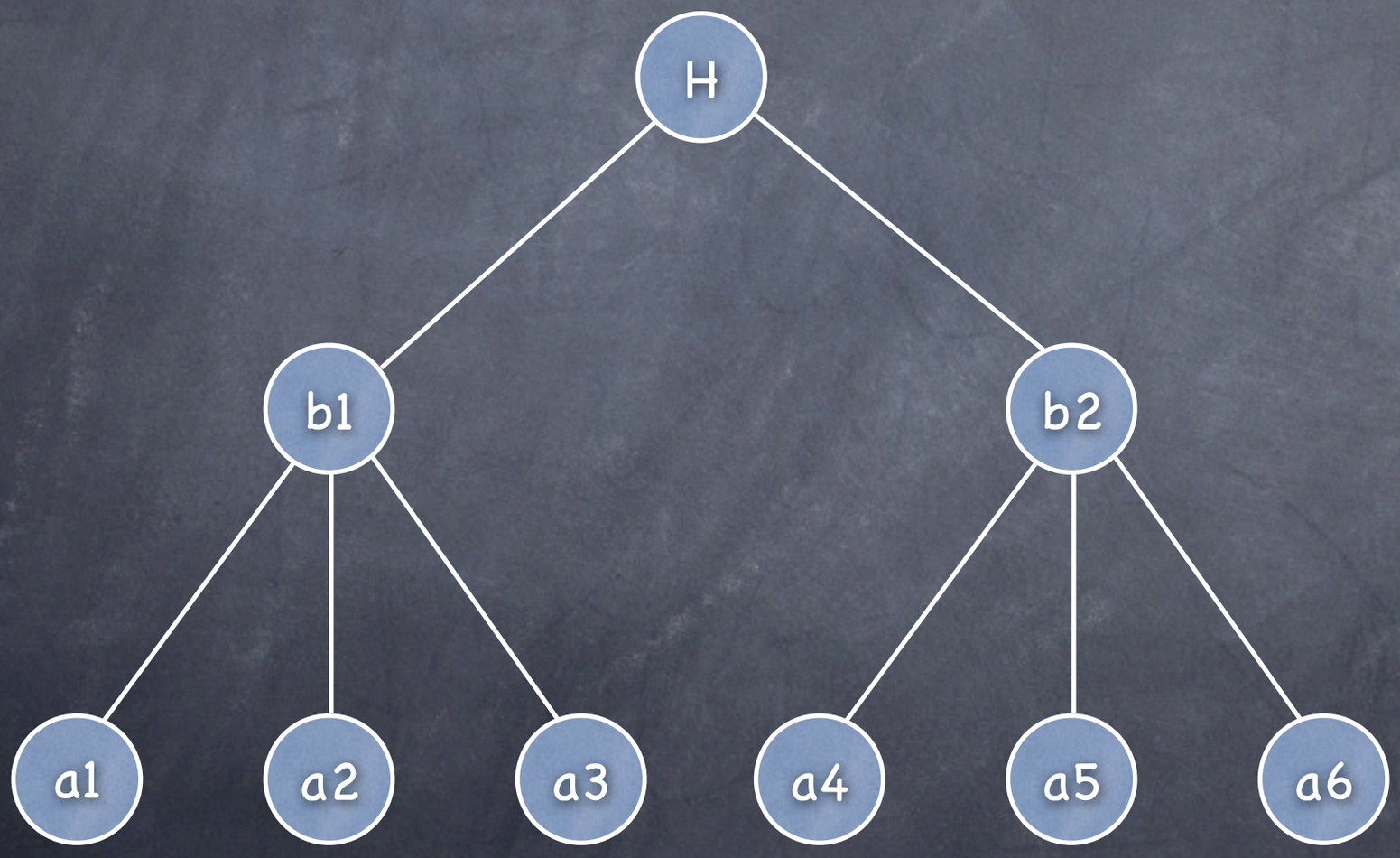
- Workloads access varying amounts of data throughout program's lifetime
- Fine-grained locking when accessing small amounts of data => More concurrency
- Coarse-grained locking when accessing large amounts of data => Low overhead

Multi-Granularity Locks

- Gray et al - "Granularity of Locks in a Shared Data Base"
- Simultaneous locking at differing granularities
- Both coarse-grained and fine-grained locks can be used
- Multi-granularity protocol takes care of their interaction

Multi-Granularity Locks

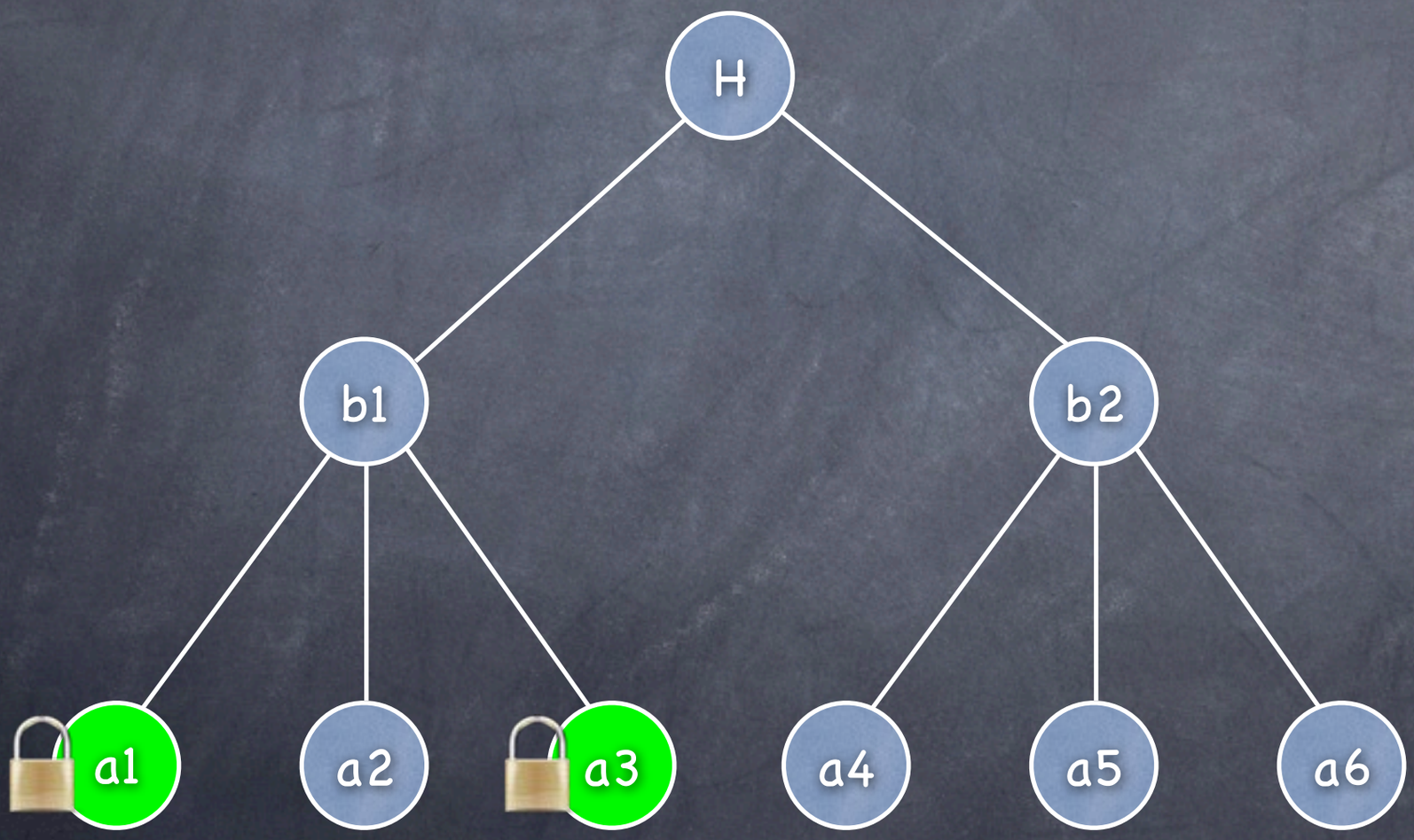
Individual account accesses => fine-grained locks



- Operations on:
  - Account
  - Branch
  - Whole bank

Multi-Granularity Locks

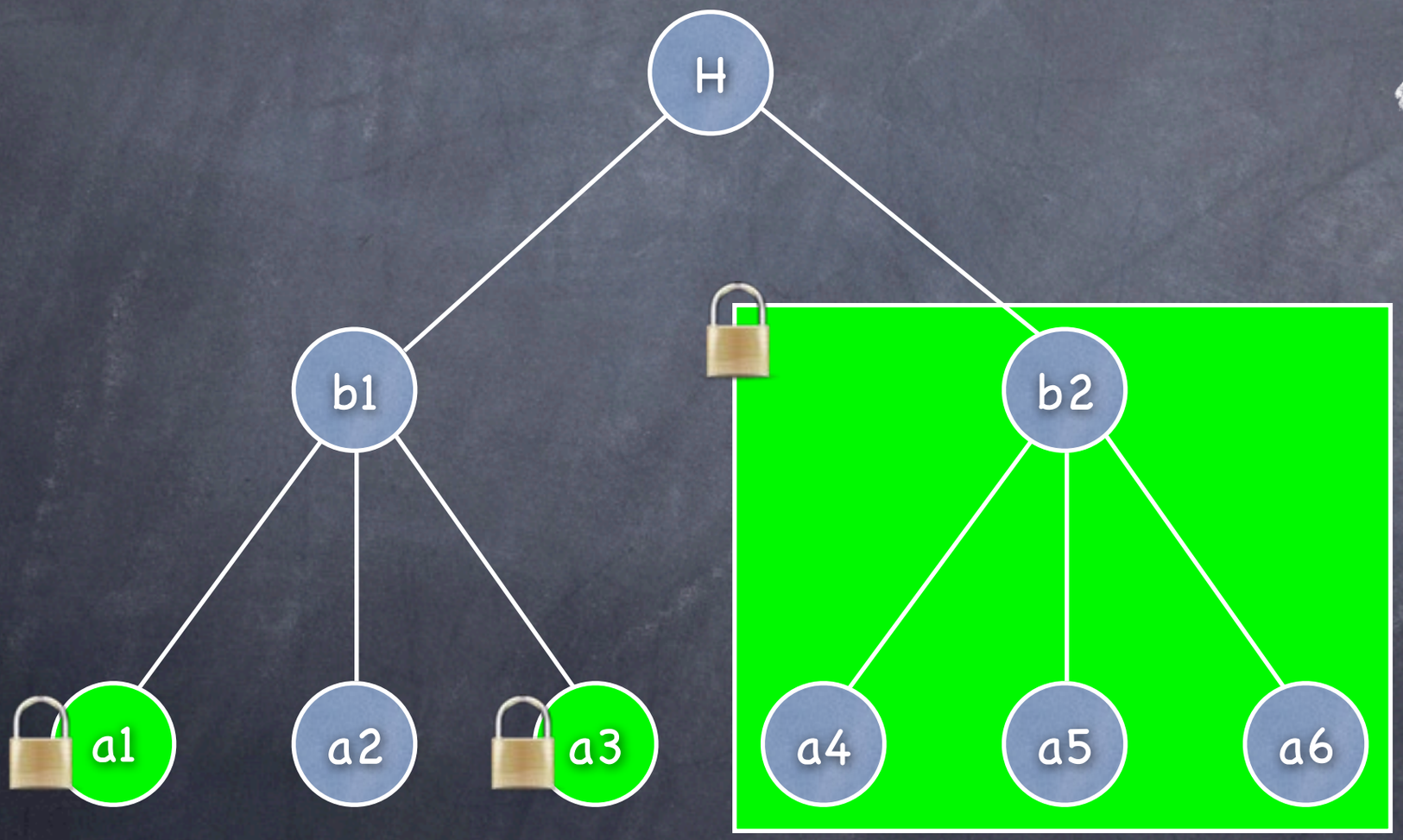
Individual account accesses => fine-grained locks



- Operations on:
  - Account
  - Branch
  - Whole bank

Multi-Granularity Locks

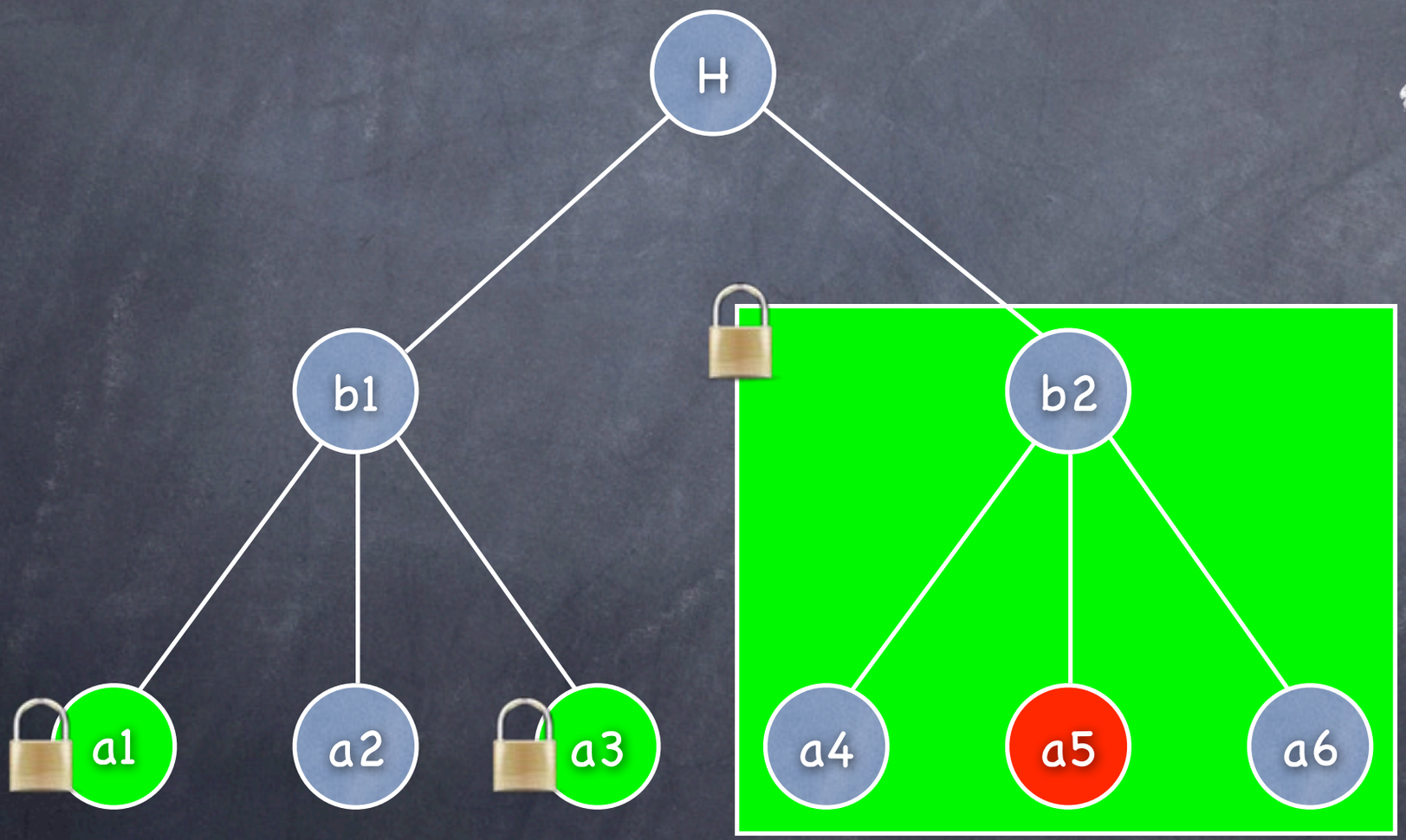
Entire branch access => coarse-grained locks



- Operations on:
  - Account
  - Branch
  - Whole bank

Multi-Granularity Locks

Interaction between coarse- and fine-grained locks

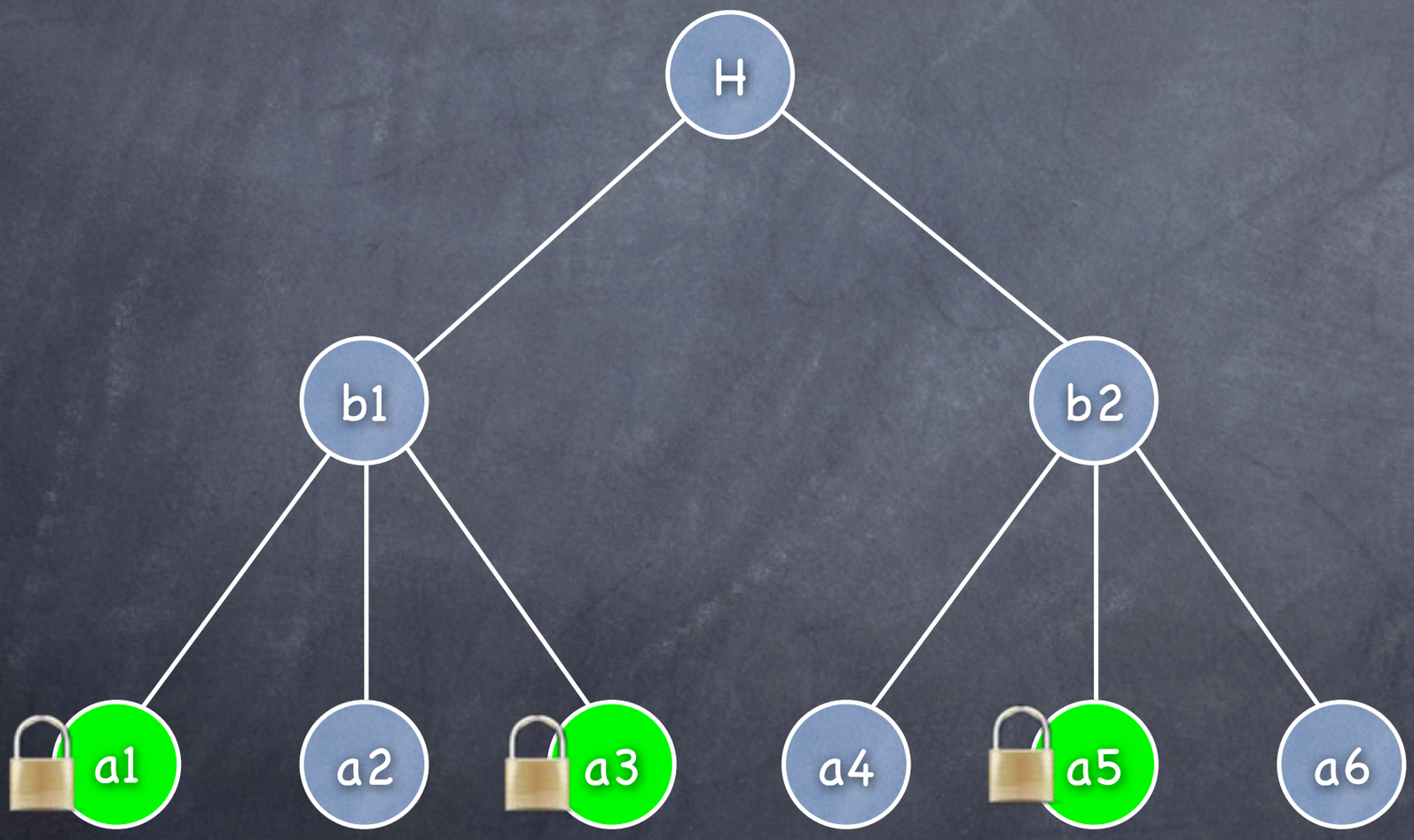


- Operations on:
  - Account
  - Branch
  - Whole bank

Thread wishes to access a5 => has to wait for b2

Multi-Granularity Locks

Interaction between coarse- and fine-grained locks



- Operations on:
  - Account
  - Branch
  - Whole bank

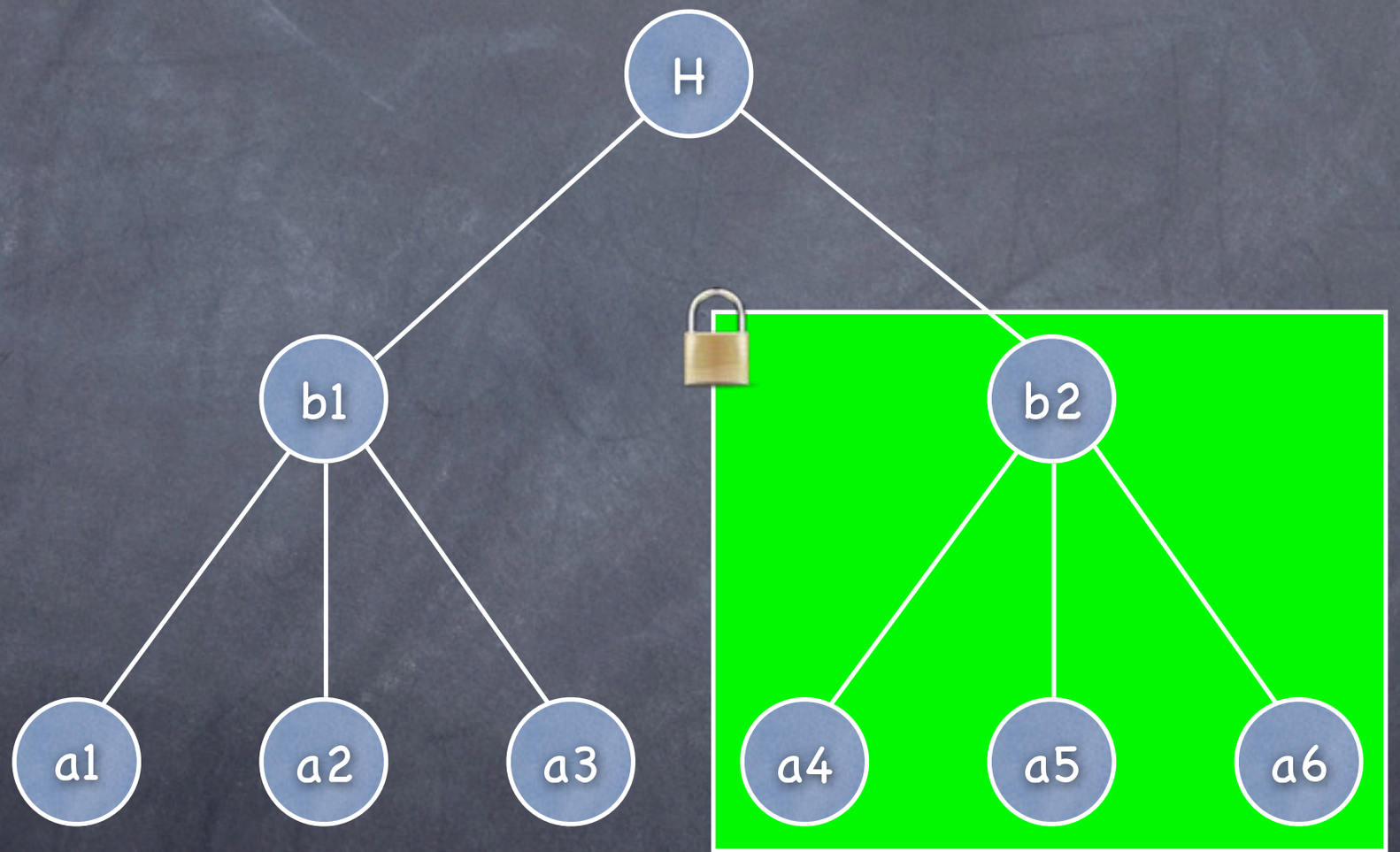
Multi-Granularity Locks

- Account can be locked if branch is not already locked and vice-versa
- Interaction is achieved using "intentional mode" locking

Intentional Mode Locking

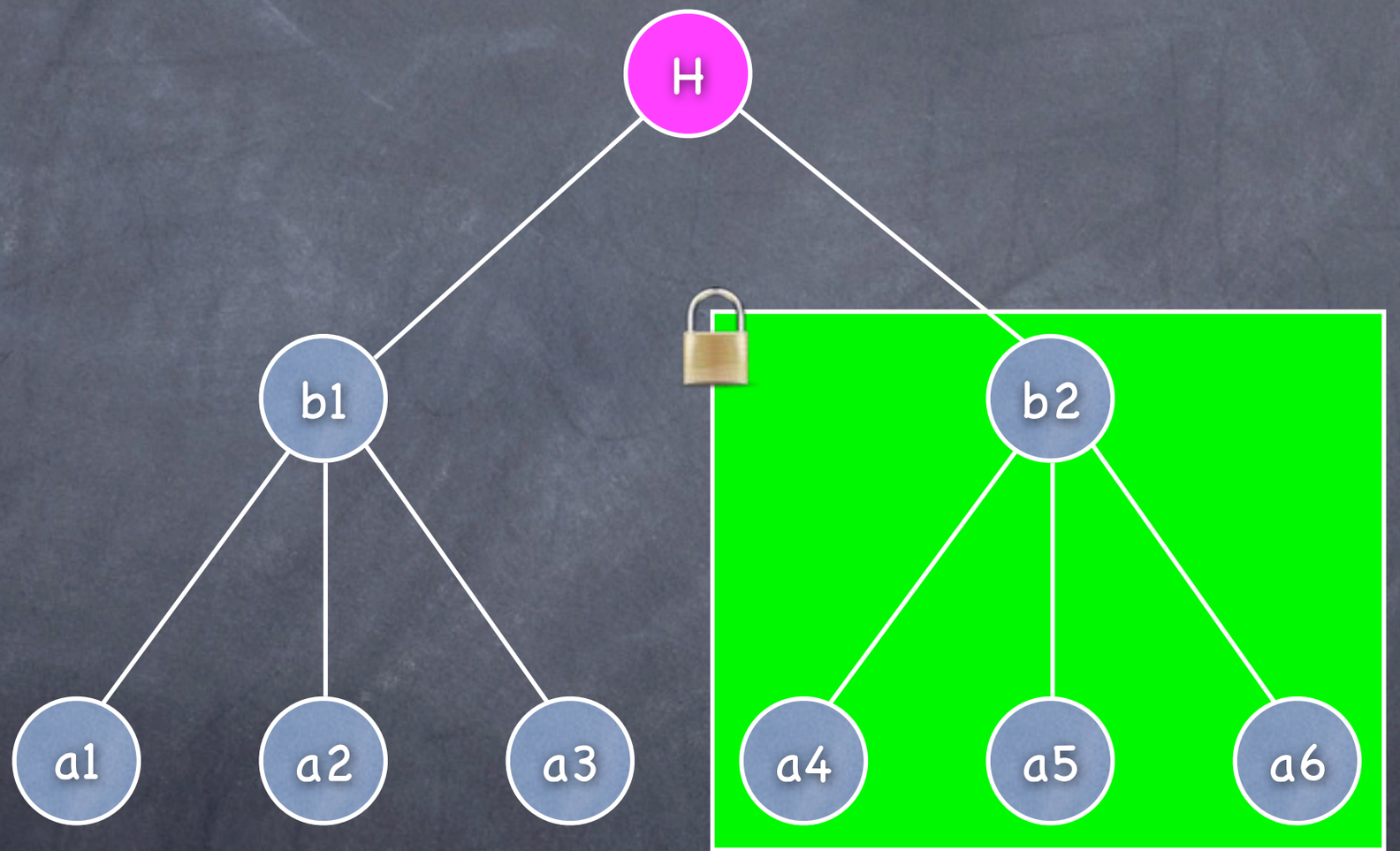
- Before locking a node, lock all ancestors in intentional mode
- "Locking is being performed lower down, is it ok to proceed?"

Intentional Mode Locking



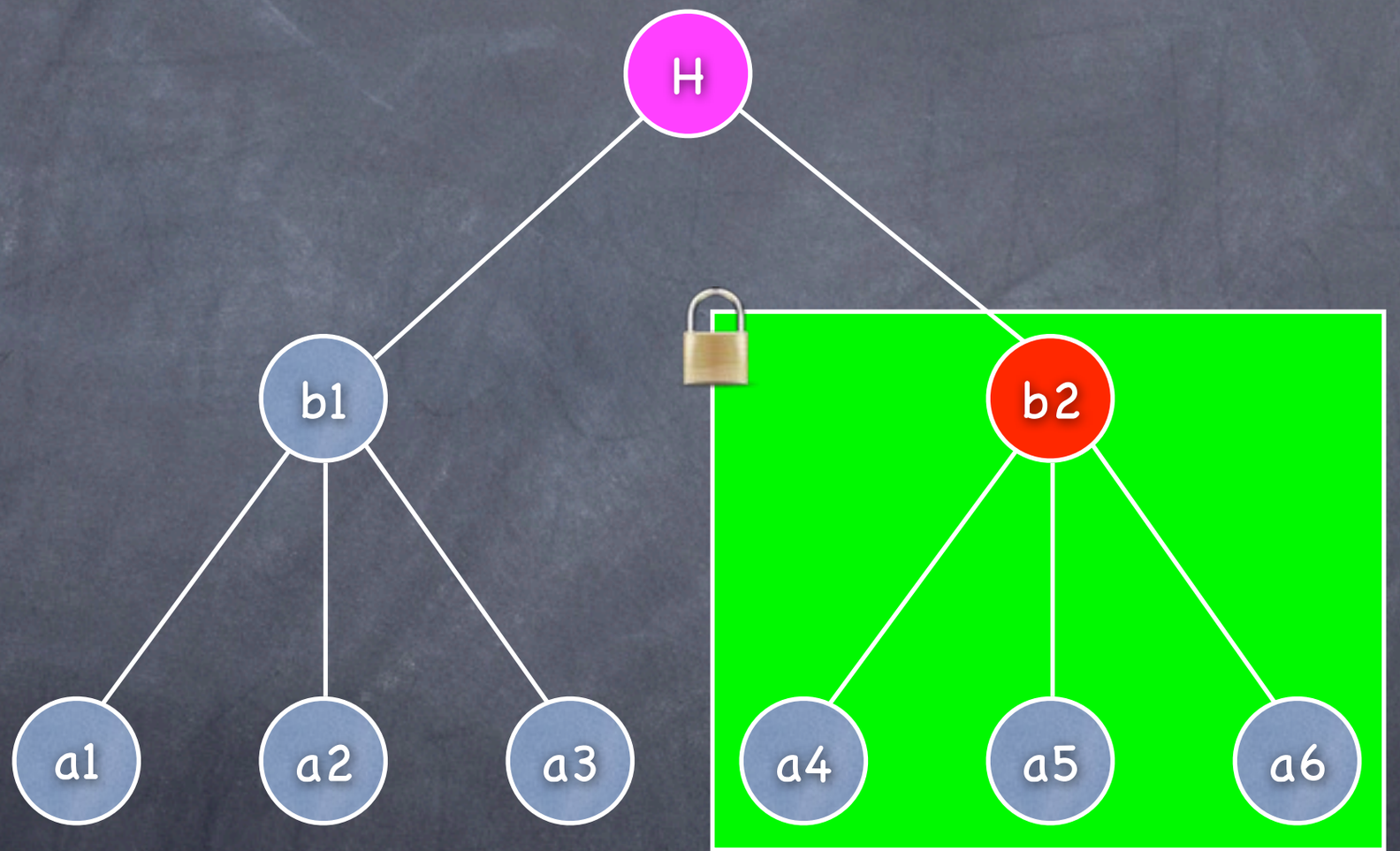
To lock a5, first intentionally acquire H and b2

Intentional Mode Locking



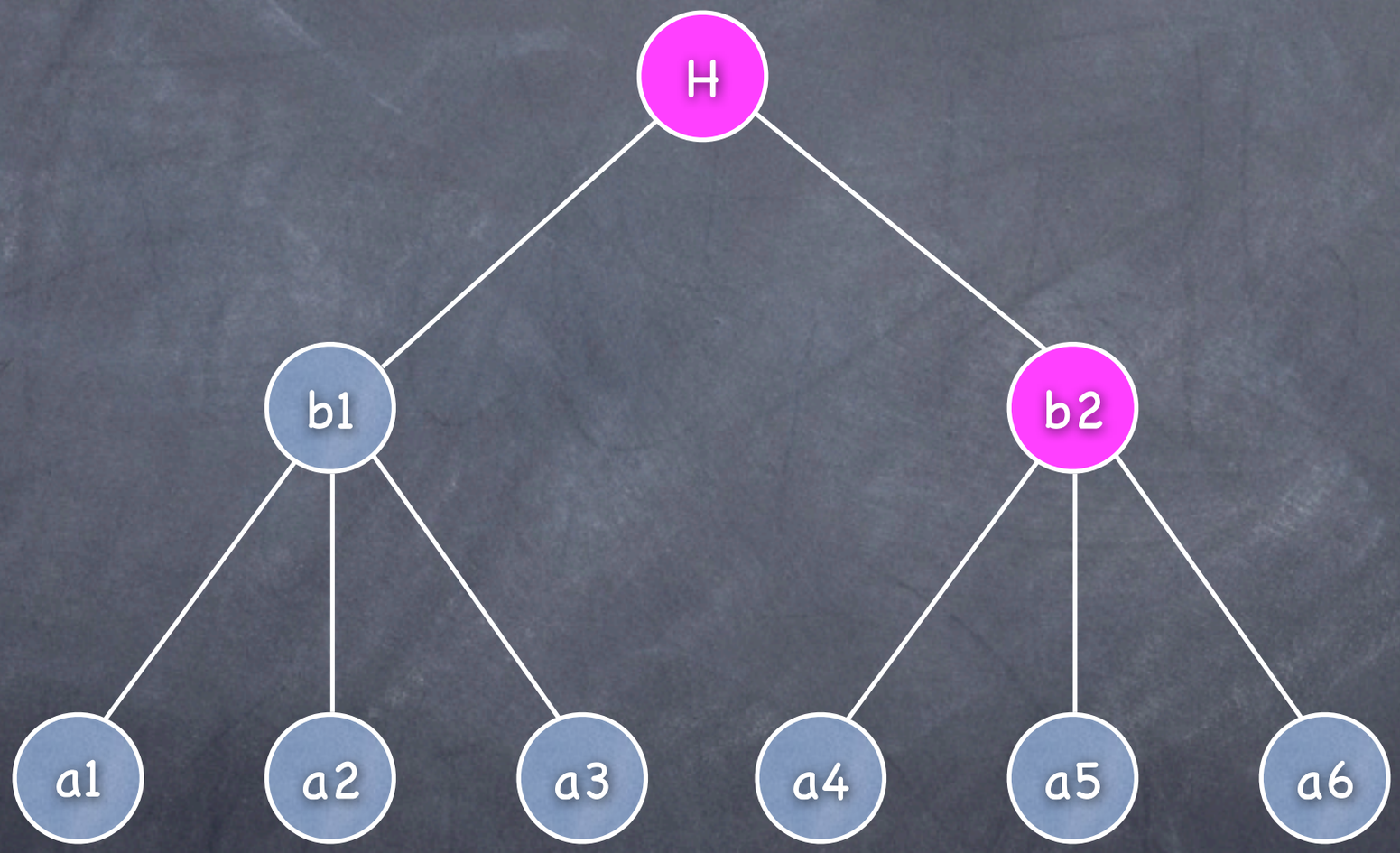
To lock a5, first intentionally acquire H and b2

Intentional Mode Locking



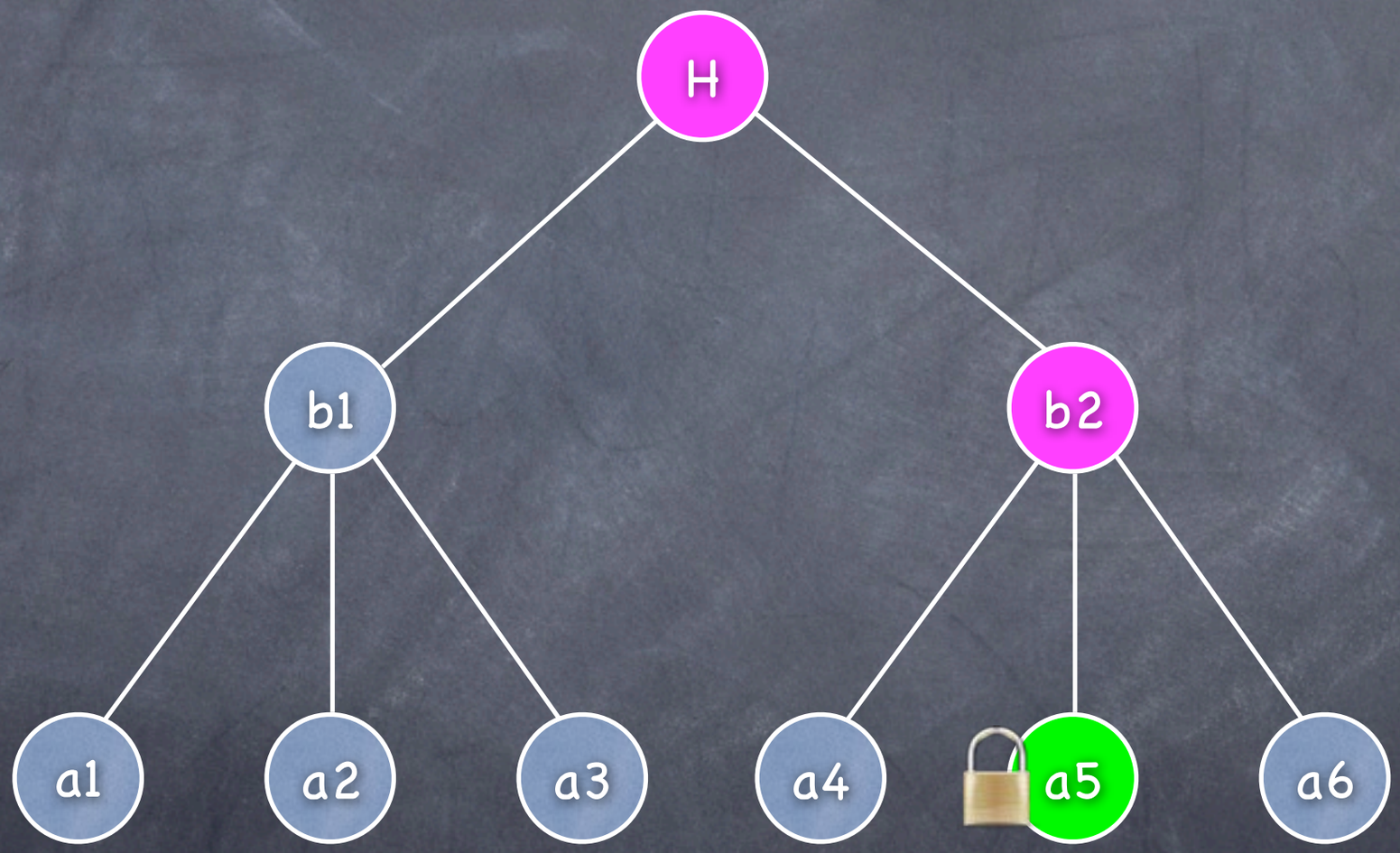
To lock a5, first intentionally acquire H and b2

Intentional Mode Locking



To lock a5, first intentionally acquire H and b2

Intentional Mode Locking



To lock a5, first intentionally acquire H and b2

Implementation

- Used Doug Lea's Synchronizer framework in Java 6 - highly performant
- Lock state represented using 64-bit long
- All state updates performed using CAS
- Queues are non-blocking

Performance Evaluation

- Does multi-granularity locking actually give a performance benefit?
- For which workloads does multi-granularity locking perform well?

Micro-benchmark

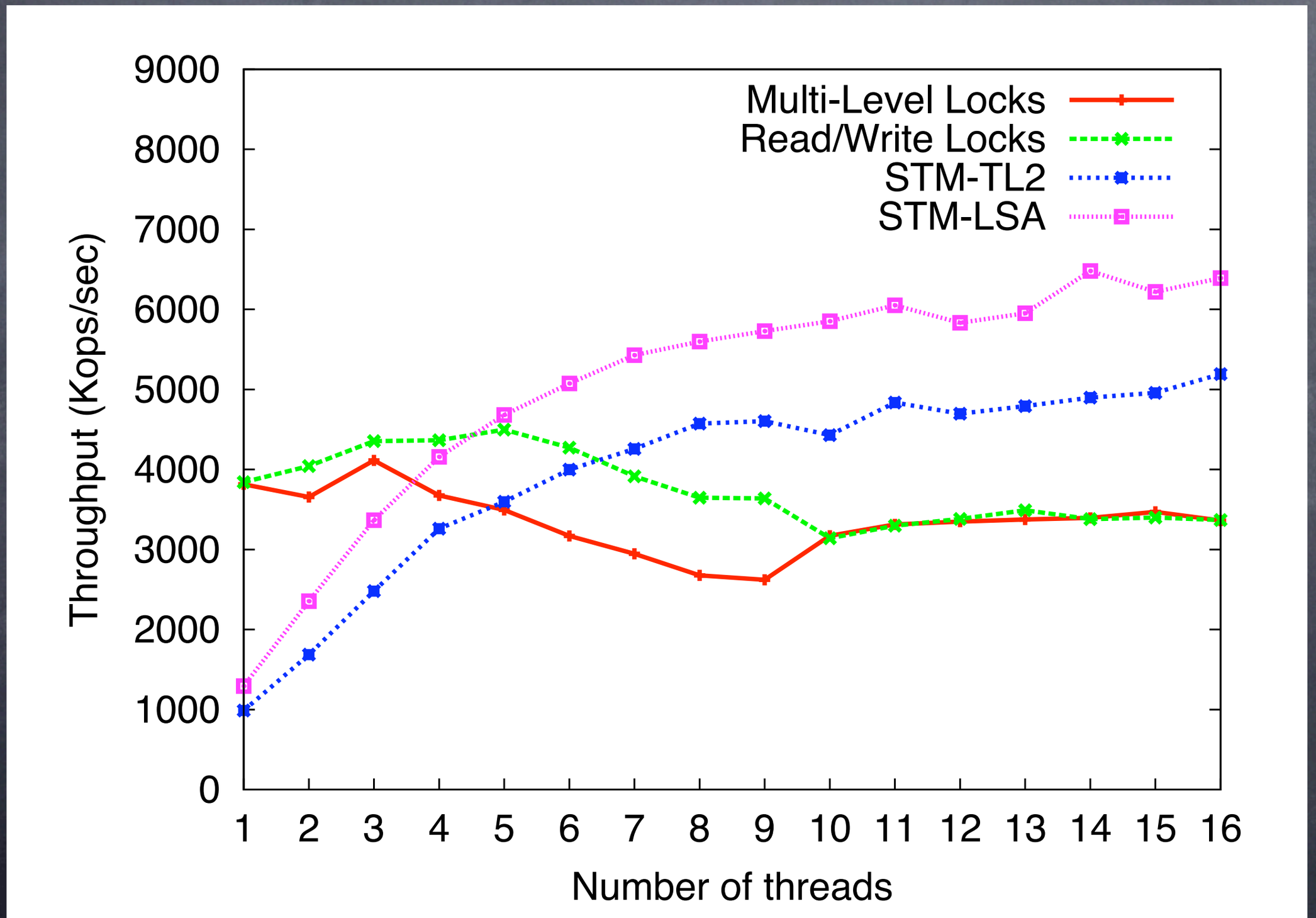
- Hierarchical bank account model with 10 branches each with 10 accounts
- 1 to 16 threads each perform 1,000,000 operations that could be any of the following:
  1. Withdraw from random account
  2. Deposit into random account
  3. Sum balances across random branch
  4. Sum balances across whole bank

Micro-benchmark

- 3 experiments - vary % of each op and measure overall number of ops per sec
- Compare against ReentrantReadWriteLock and Deuce STM (LSA and TL2 algorithms)

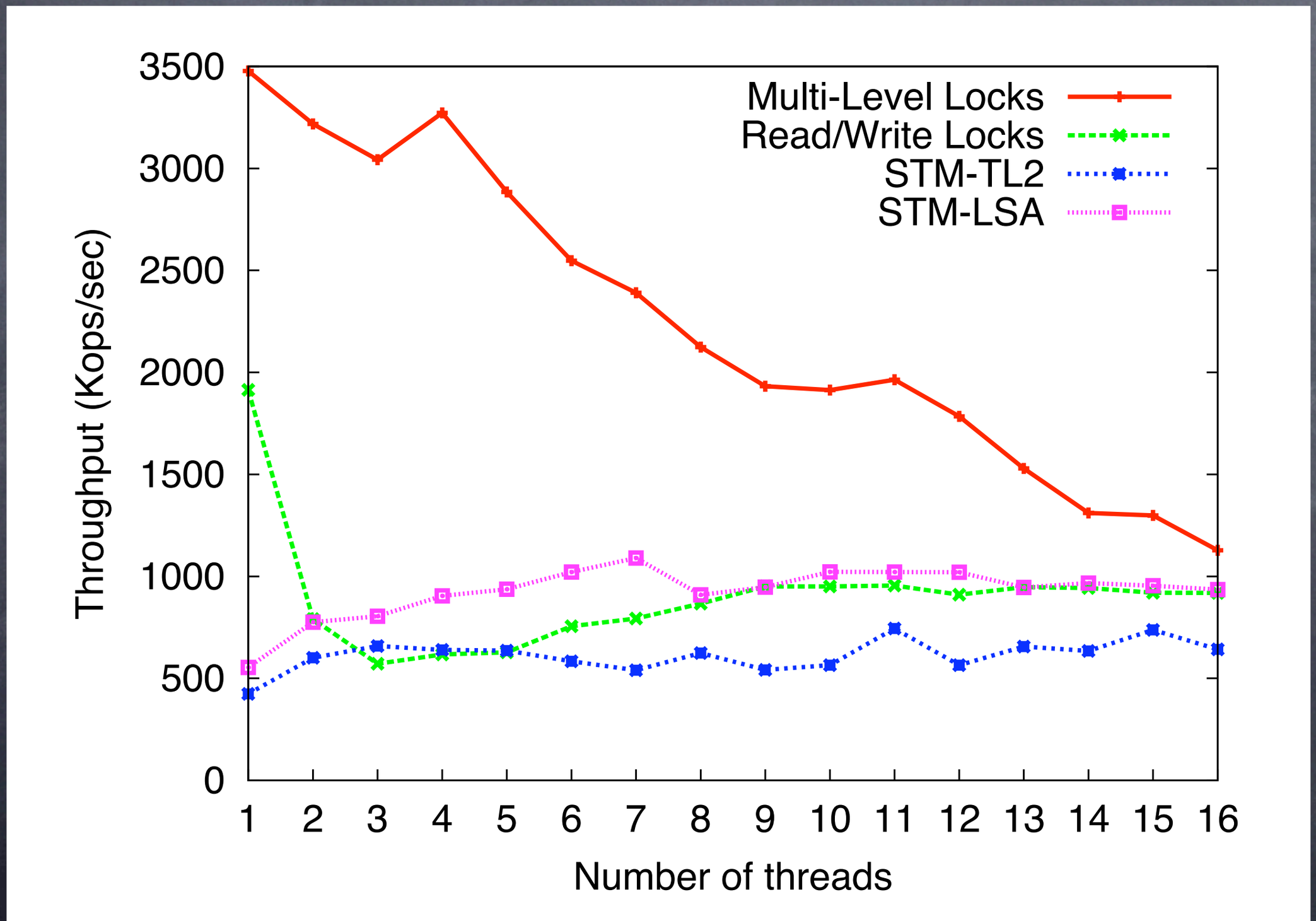
Experiment 1: Fine-Grained

50% withdrawals and 50% deposits



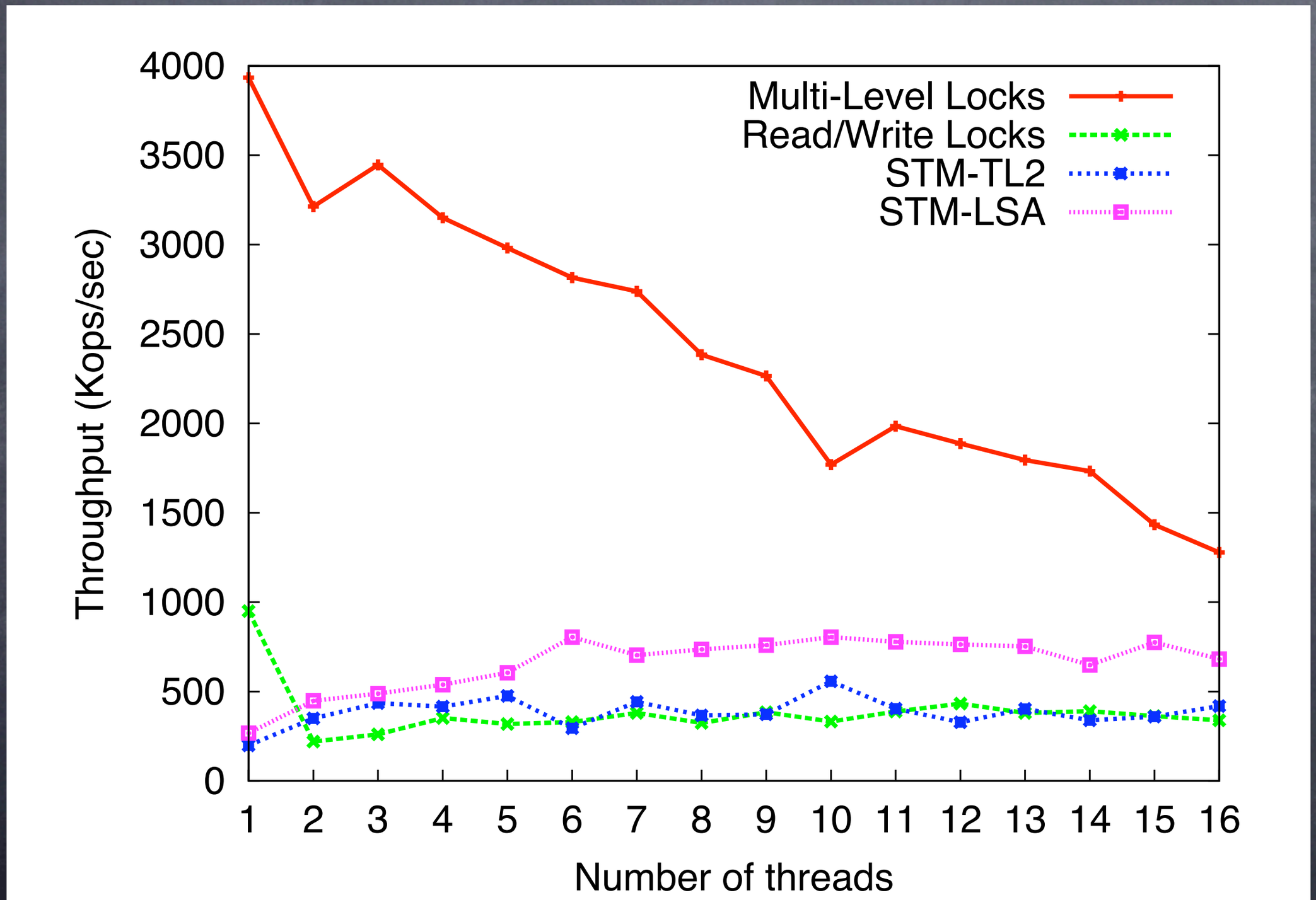
Experiment 2: Medium-Grained

40% with., 40% dep., 10% branch, 10% bank



Experiment 3: Coarse-Grained

20% with., 20% dep., 30% branch, 30% bank



Conclusion

- Fine-grained locking good for small accesses, coarse-grained locking good for large accesses
- Multi-granularity allows different granularities of locks simultaneously
- Results show that multi-level locks can yield better performance for workloads with a mix of coarse- and fine-grained operations