2025 Operating System

Lab2. Concurrency

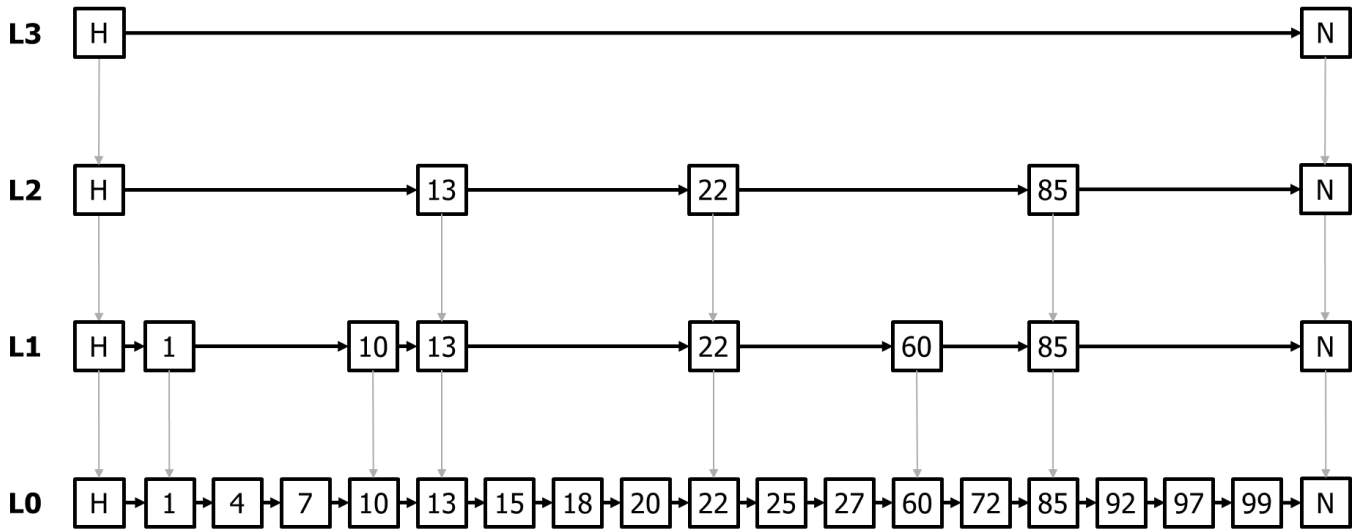
2025.04.28

T.A. 김보승

bskim1102@dankook.ac.kr

Lab2. Concurrency

- Implement Lock
  - Implement the following three versions of the Skip List
    - Without Lock
    - Coarse-grained Lock
    - Fine-grained Lock



Data Structure

- Skip List
  - Data structure that builds **multiple levels of linked lists** on top of a sorted vase list to enable fast search, insertion, and deletion
  - Average time complexity of **O(log N)**
  - Offer a simple alternative to balanced trees with similar performance

  - Structure
    - The bottom level is a regular sorted linked list (level 0)
    - Each higher level contains a subset of the nodes from the level below

Data Structure

- Skip List
  - Pros (Strengths)
    - Fast average performance: O(log N) for search, insert, delete
    - Simple and easy to implement
    - No need for complex rebalancing like in AVL or B+Tree
  - Cons (Weakness)
    - Worst-case performance is O(n)
    - Performance depends on randomness (e.g., coin flips for level height)

Data Structure

- Skip List
  - How it works
    - Search (Lookup)
      - Start at the highest level and move right
      - If the next node is greater than the target, move one level down
      - Repeat until level 0 is reached and the target is found
    - Insert
      - Search for the position to insert
      - Randomly choose the level height for the new node (e.g., coin flipping)
      - Insert the node into each appropriate level
    - Delete (Remove)
      - Locate the node similarly to search
      - Remove the node from each level it appears in

Data Structure

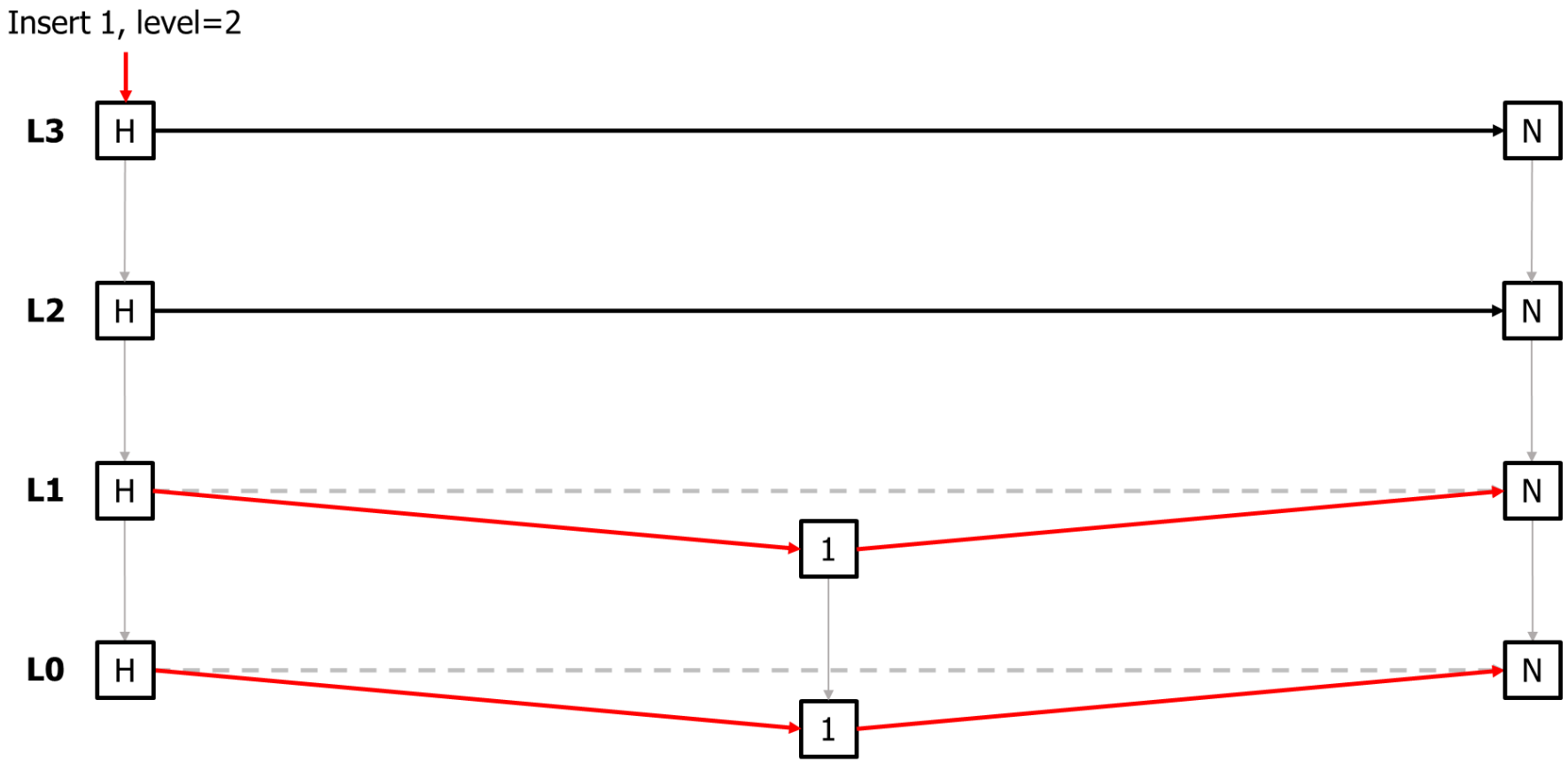
- Skip List - Insert
  - Max Level: 4

Data Structure

- Skip List - Insert

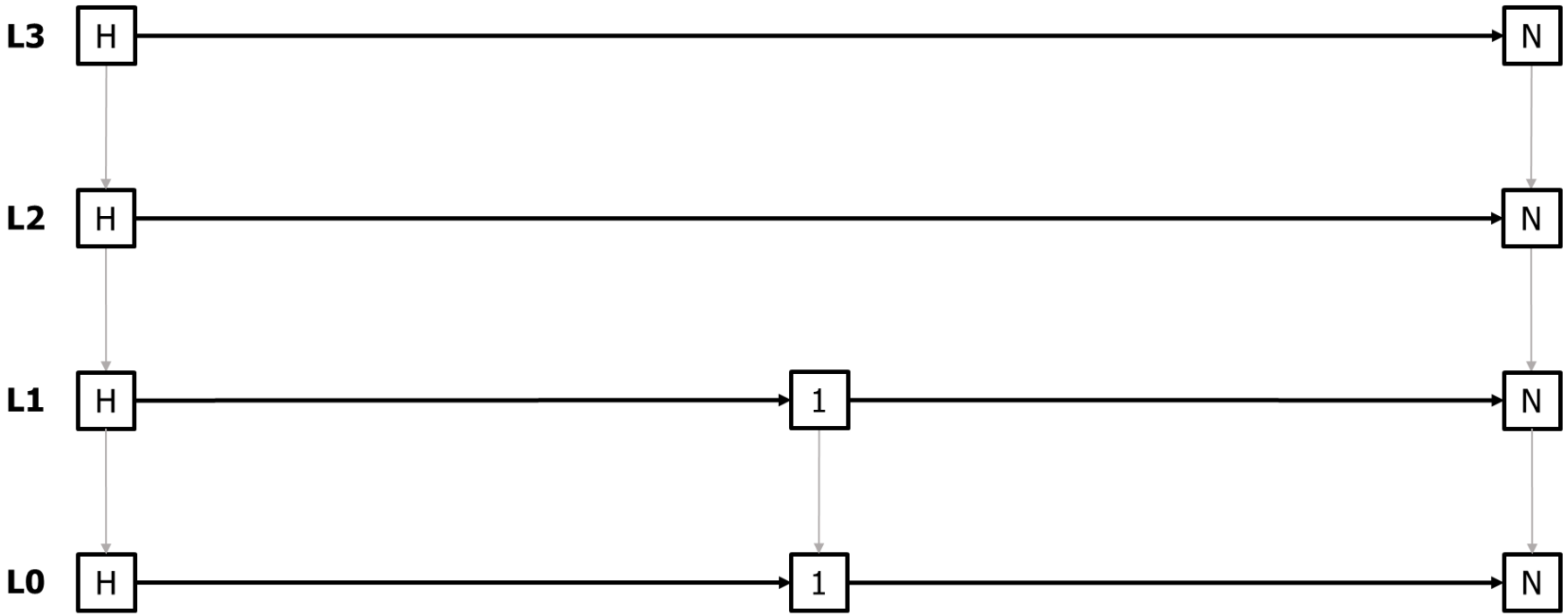
Data Structure

- Skip List - Insert



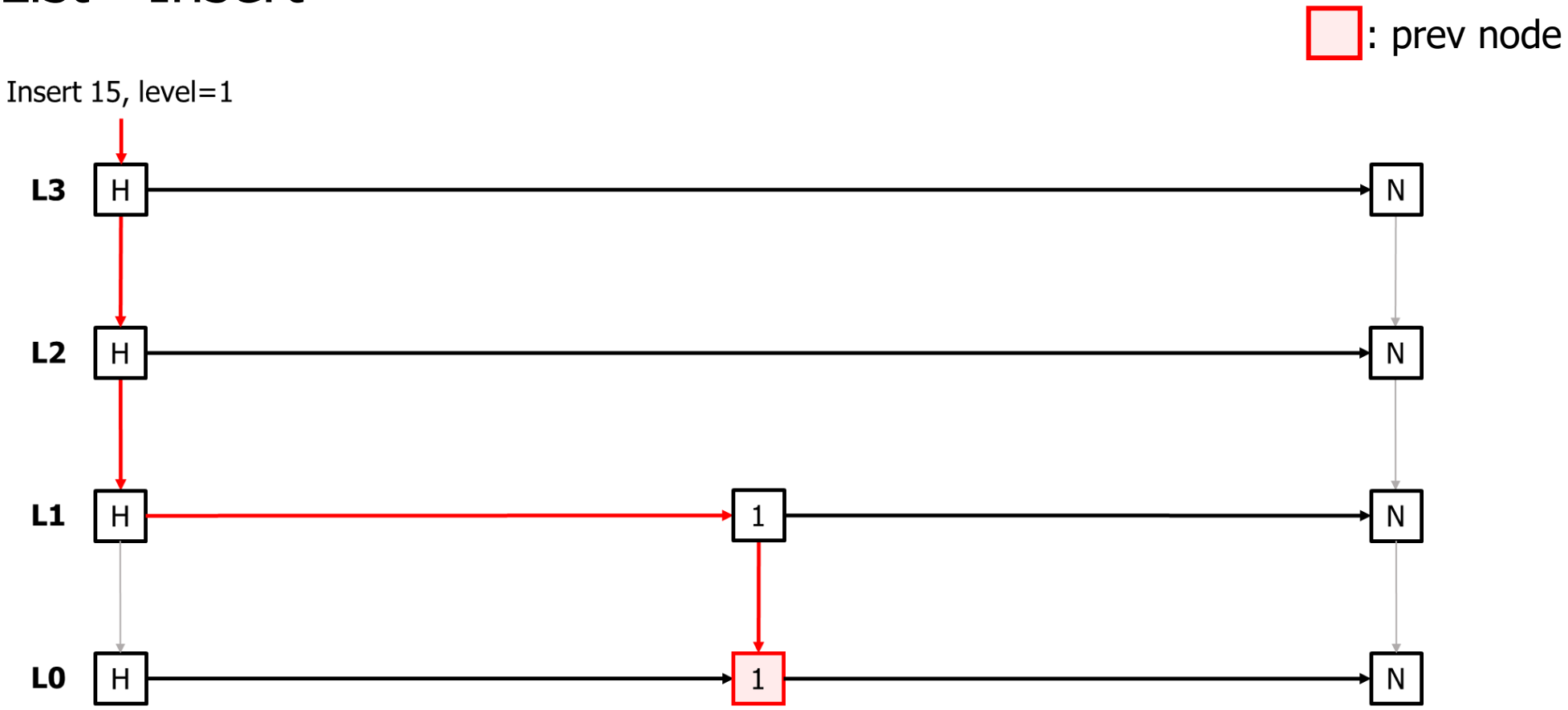
Data Structure

- Skip List - Insert



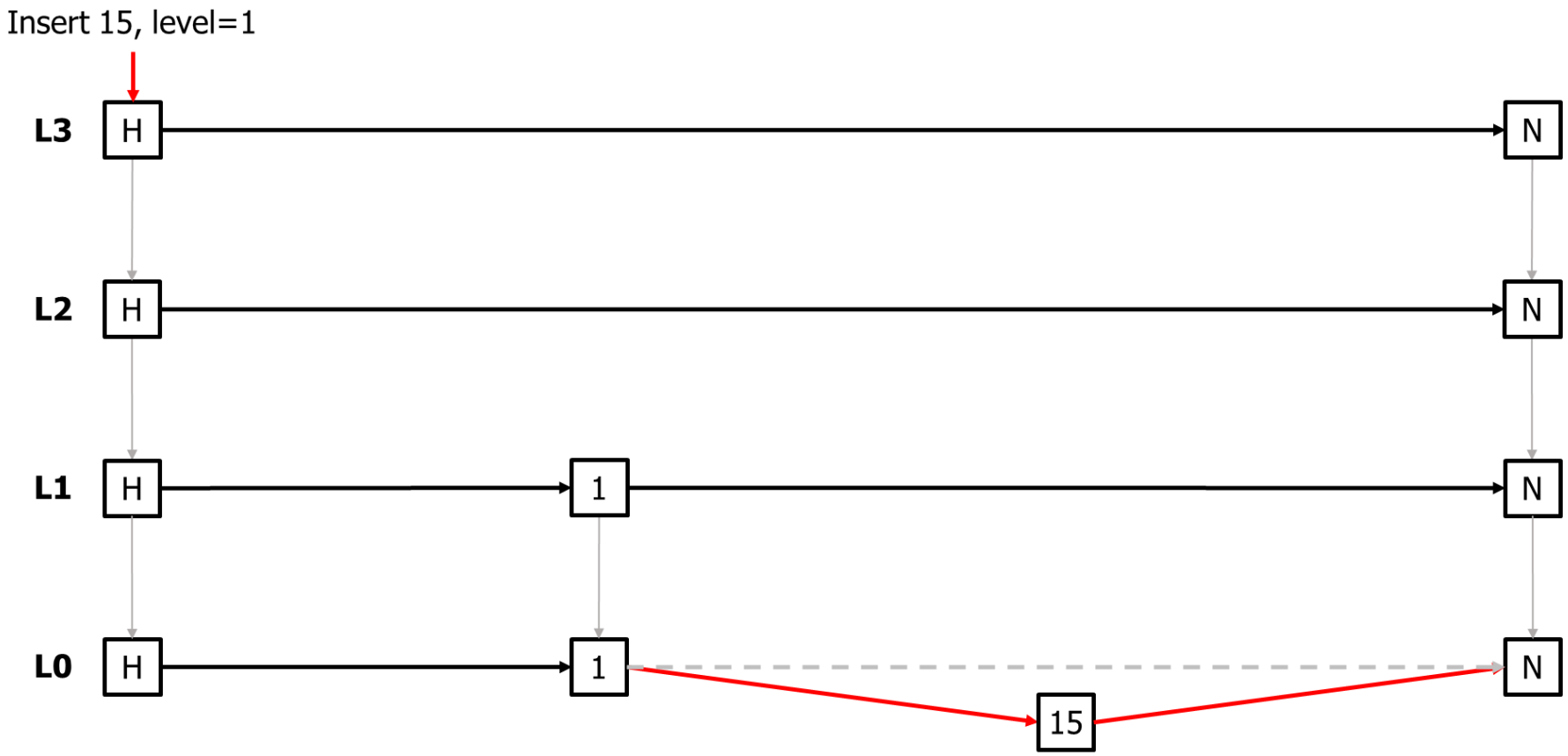
Data Structure

- Skip List - Insert



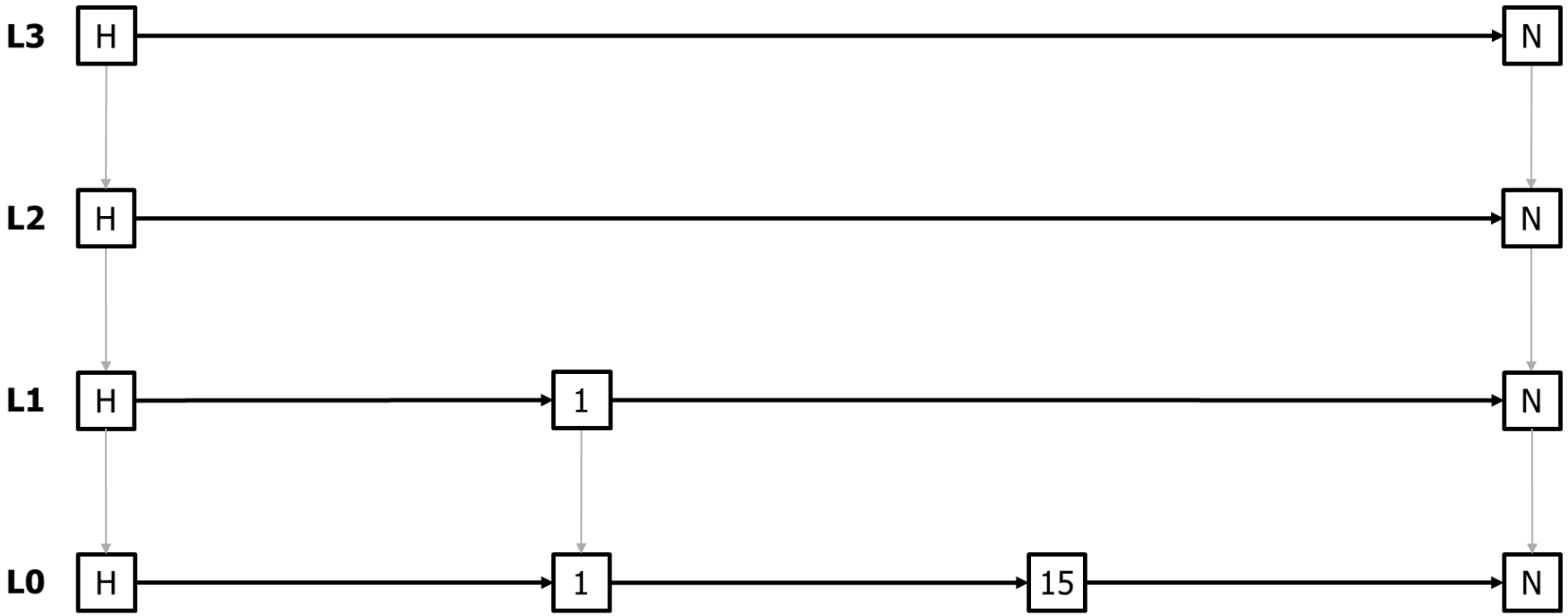
Data Structure

- Skip List - Insert



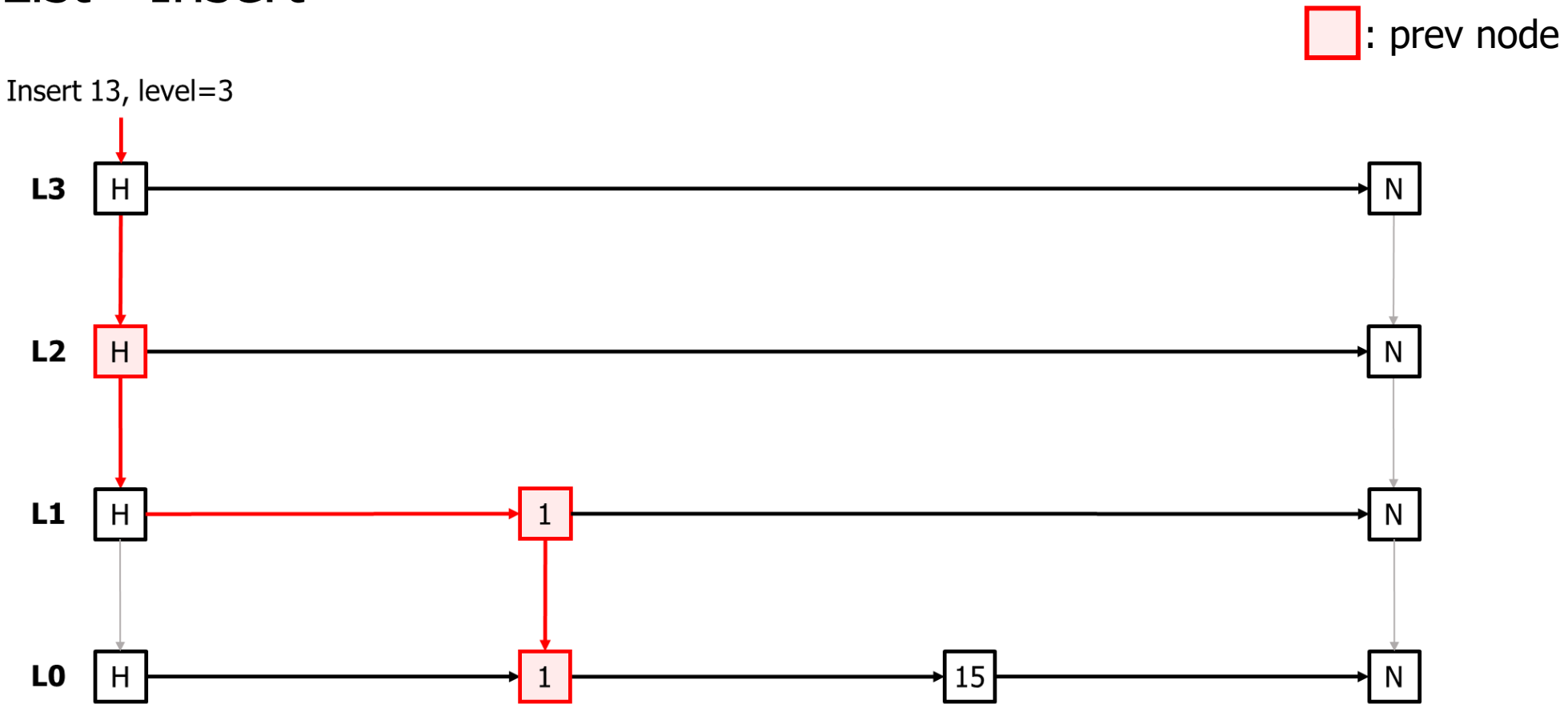
Data Structure

- Skip List - Insert



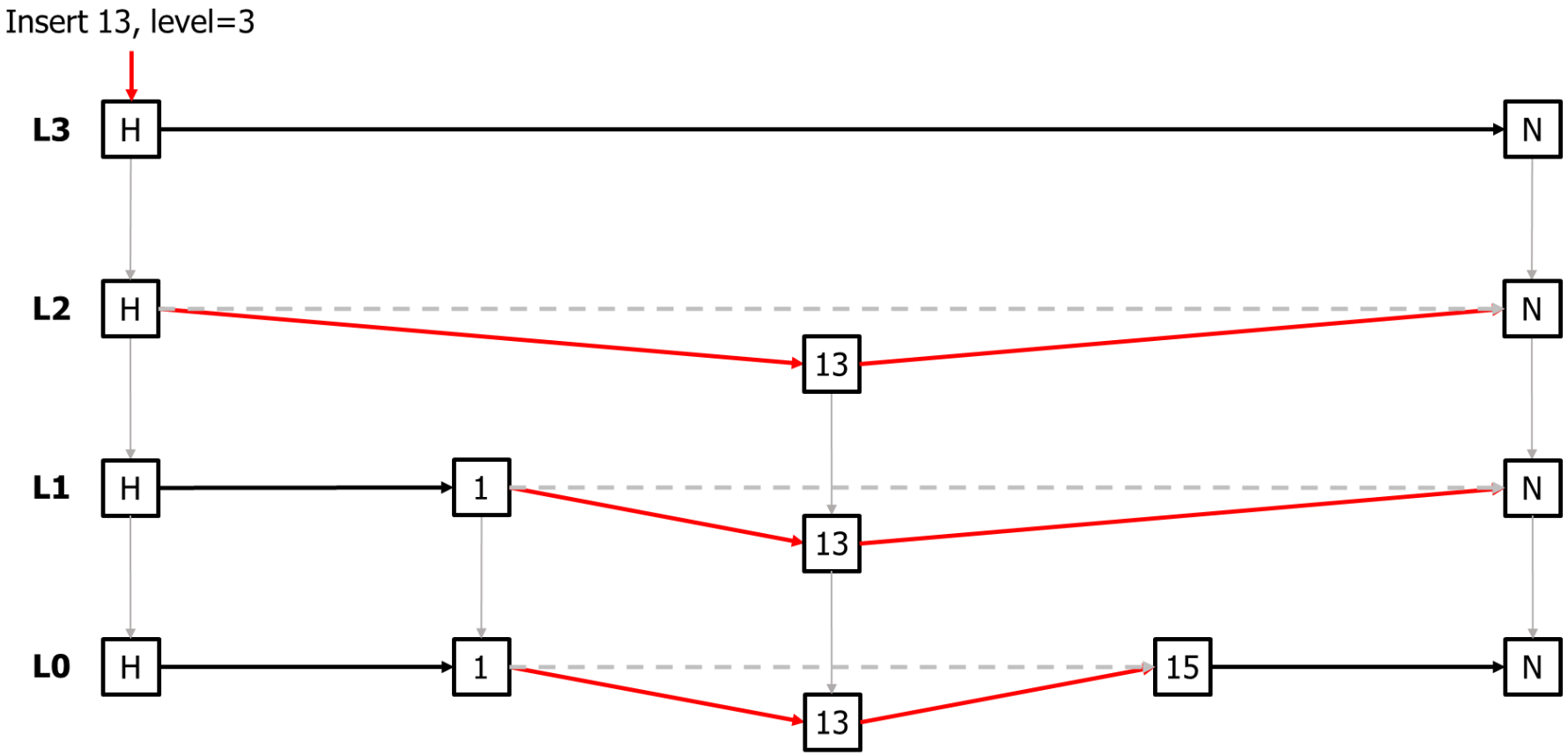
Data Structure

- Skip List - Insert



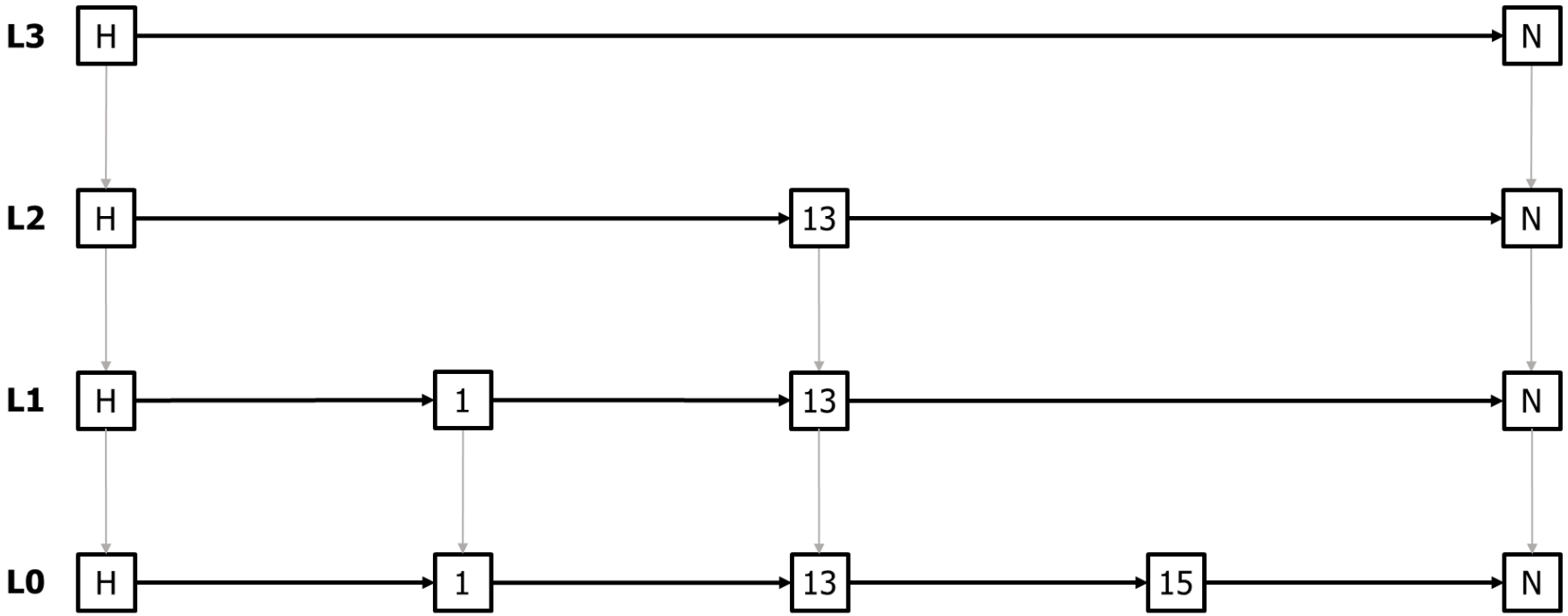
Data Structure

- Skip List - Insert



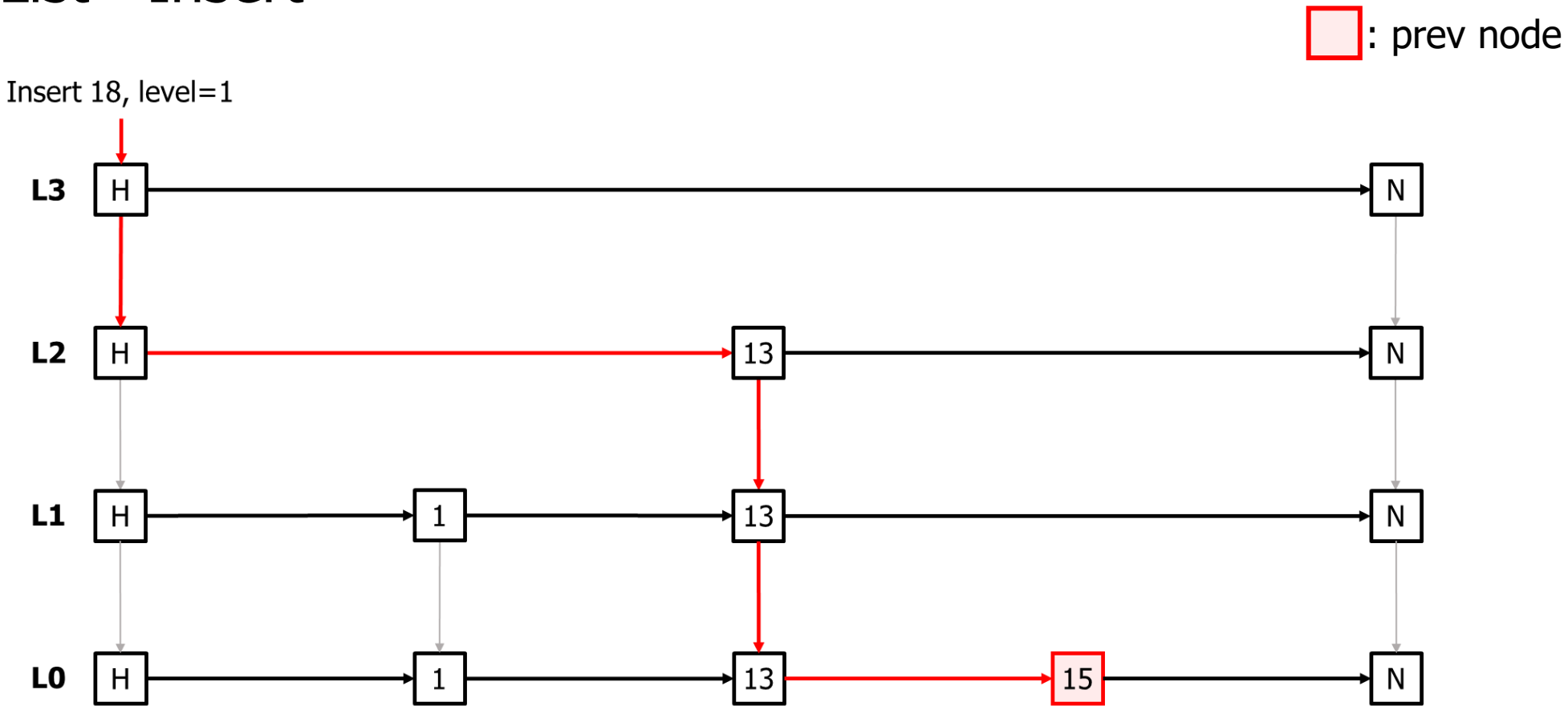
Data Structure

- Skip List - Insert



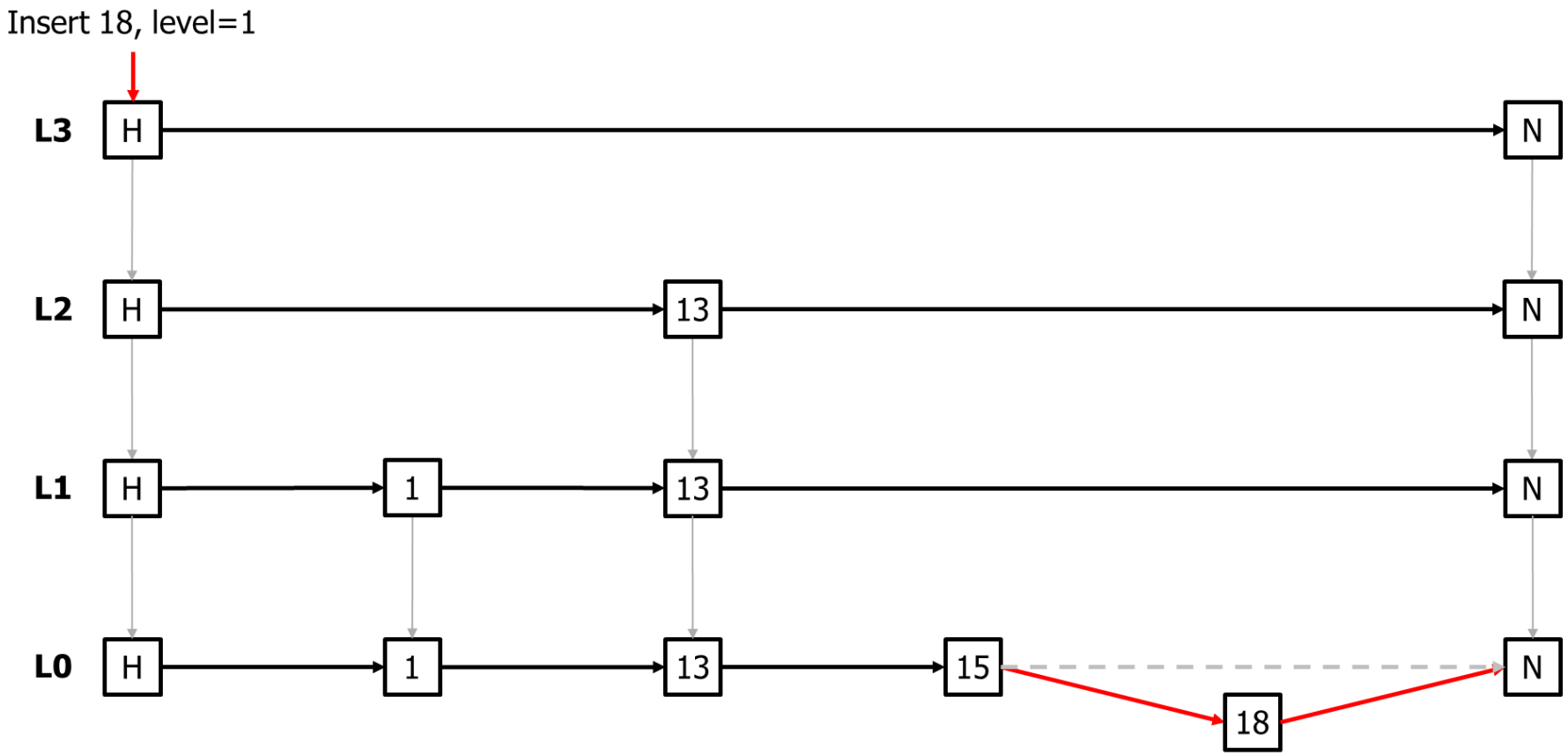
Data Structure

- Skip List - Insert



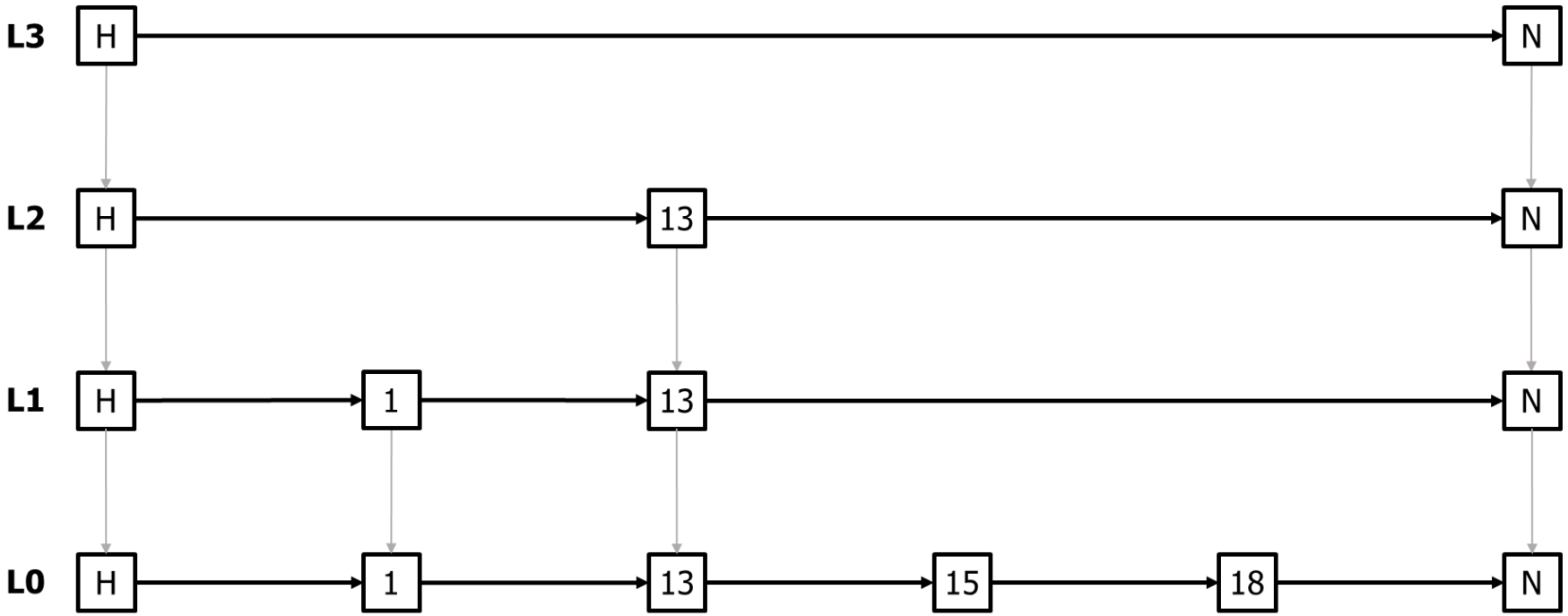
Data Structure

- Skip List - Insert



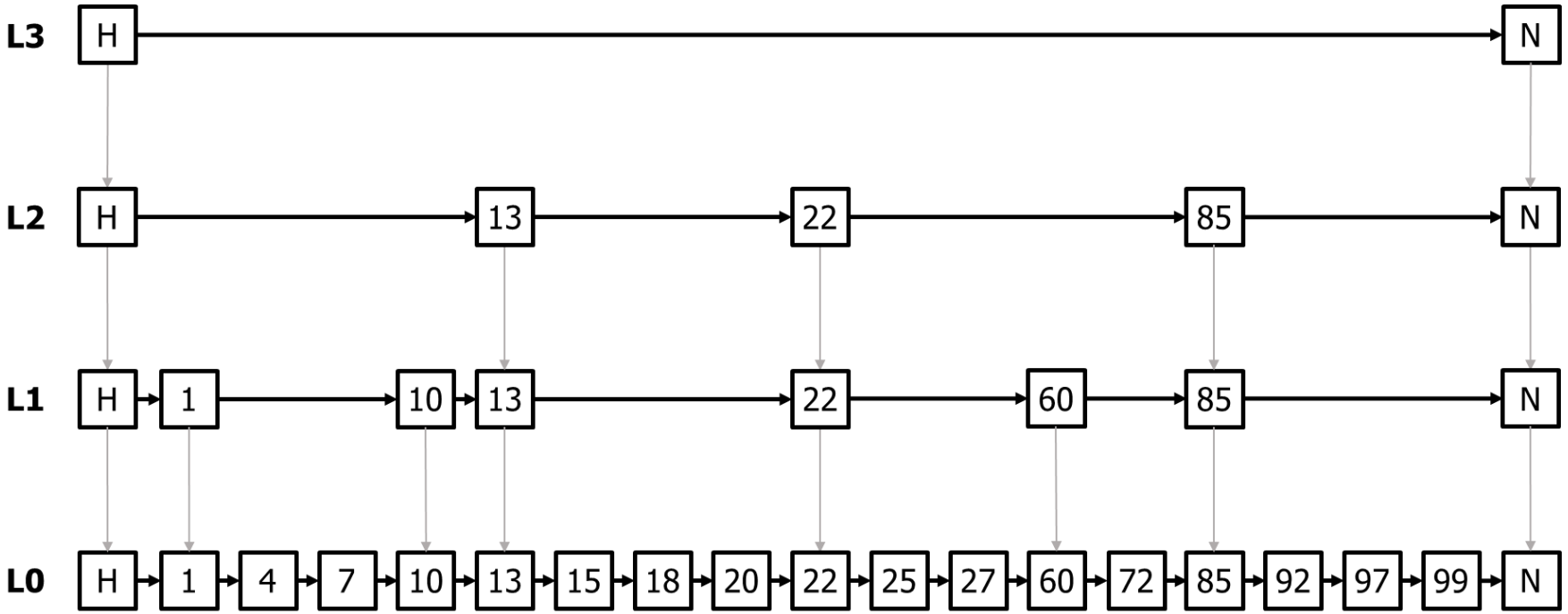
Data Structure

- Skip List - Insert



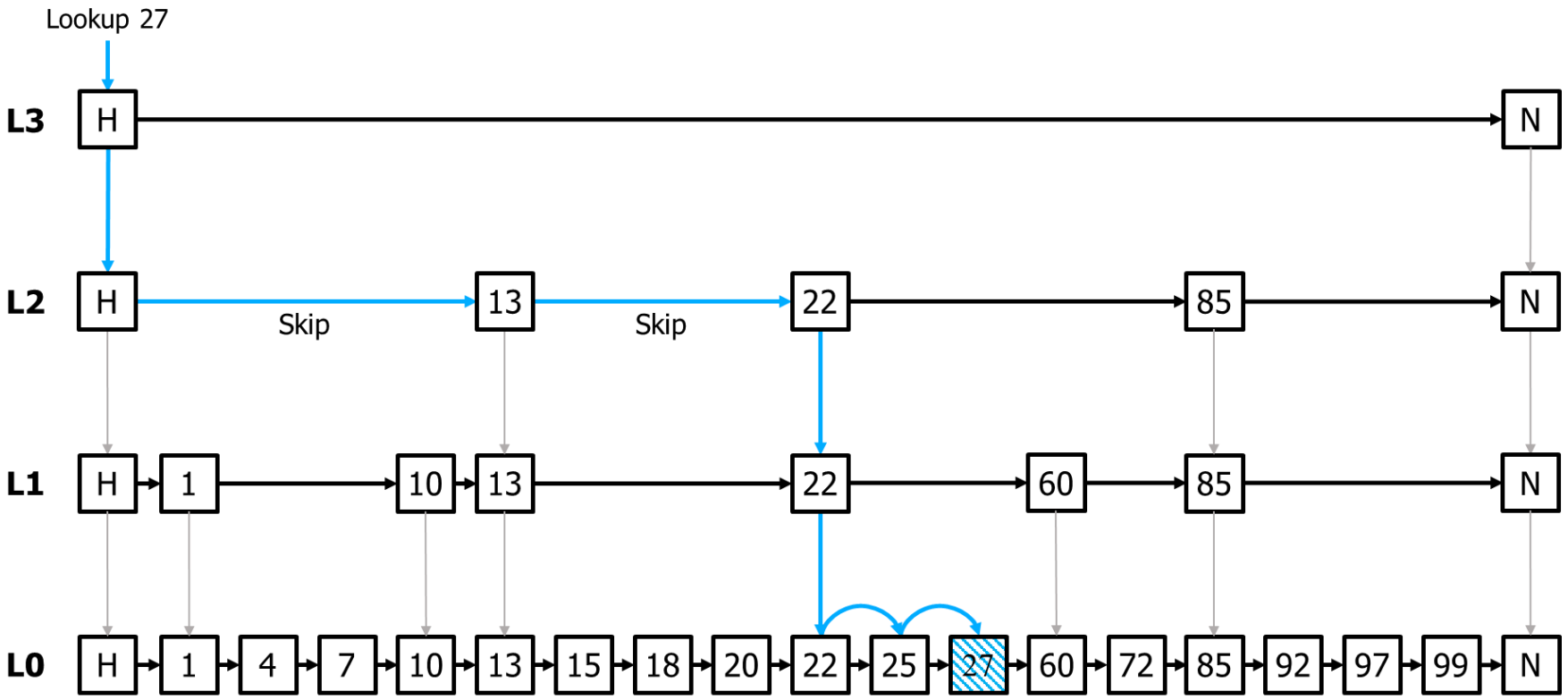
Data Structure

- Skip List - Lookup



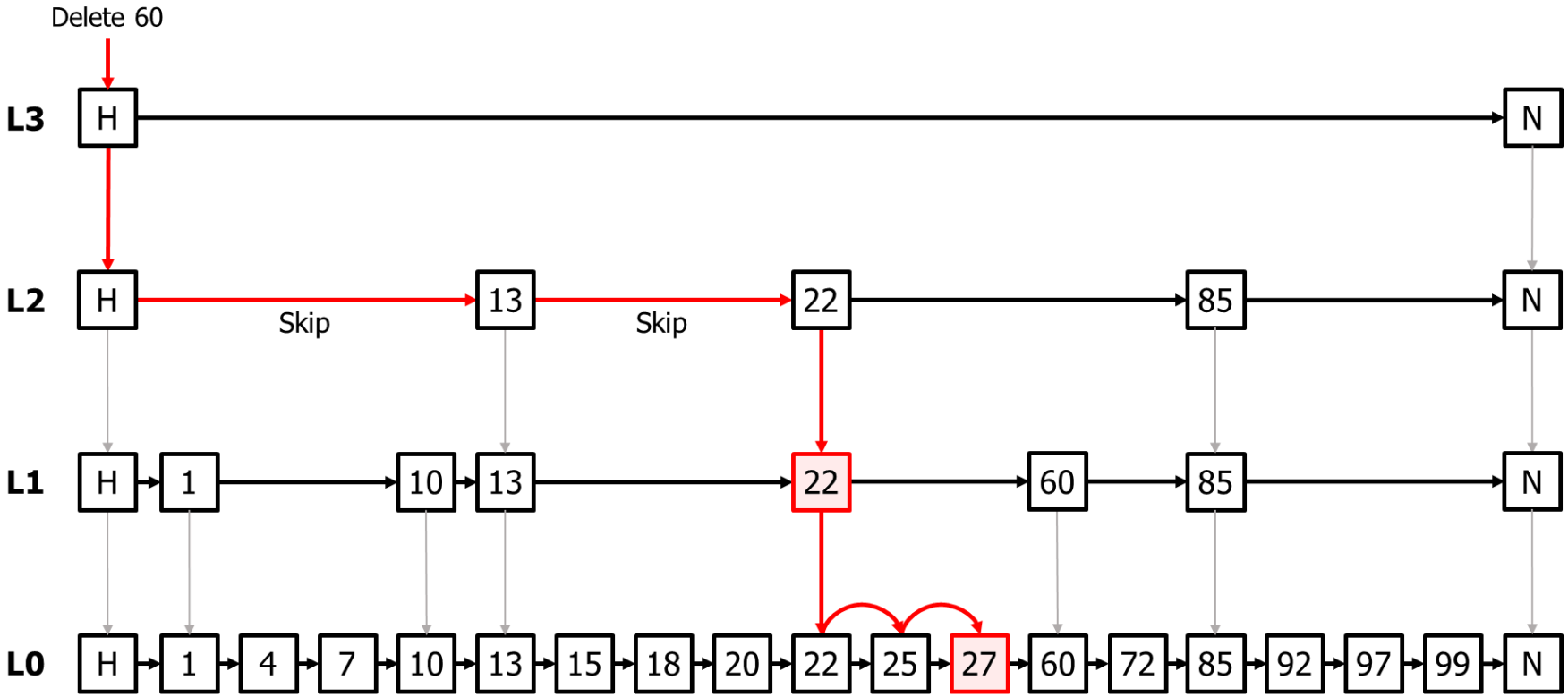
Data Structure

- Skip List - Lookup



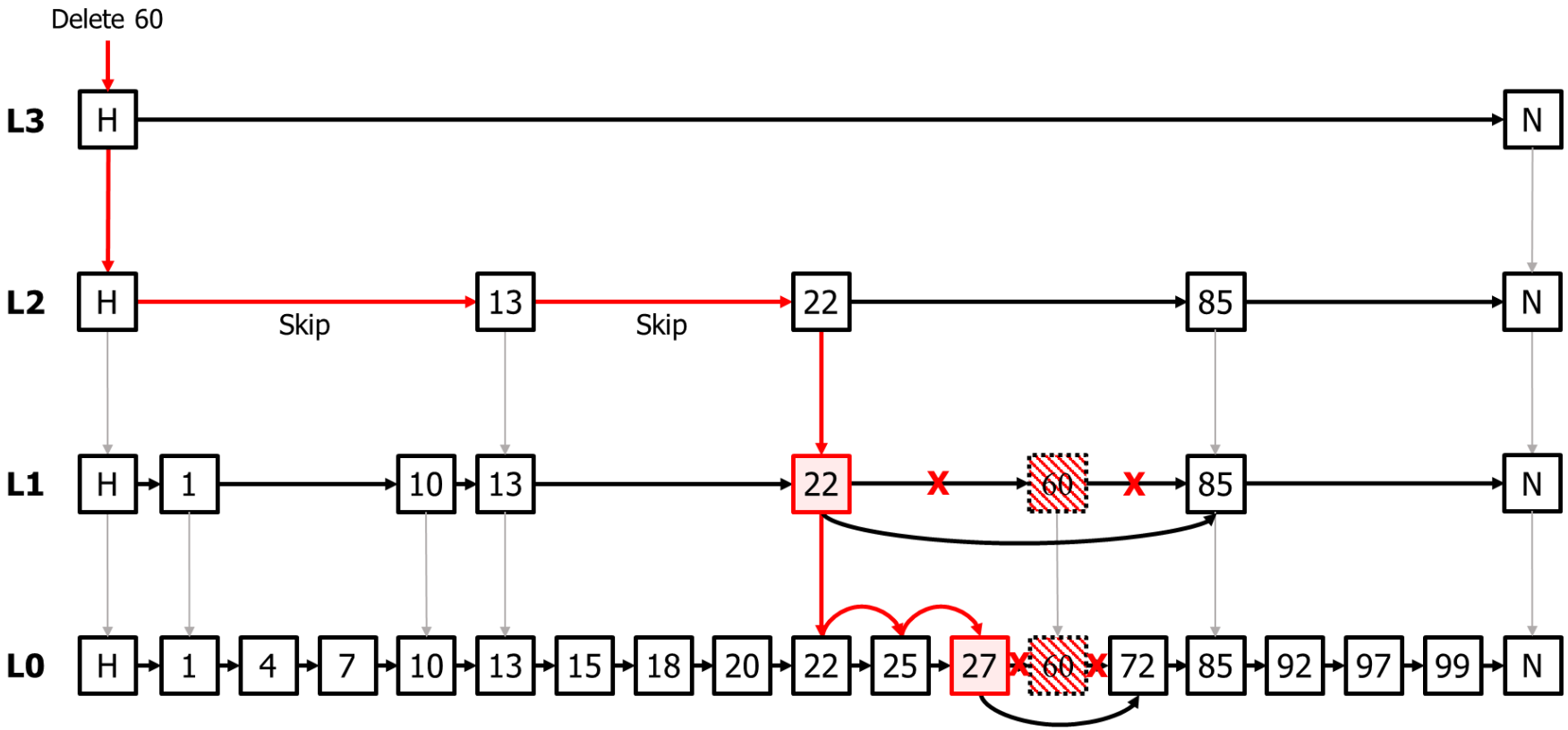
Data Structure

- Skip List - Delete



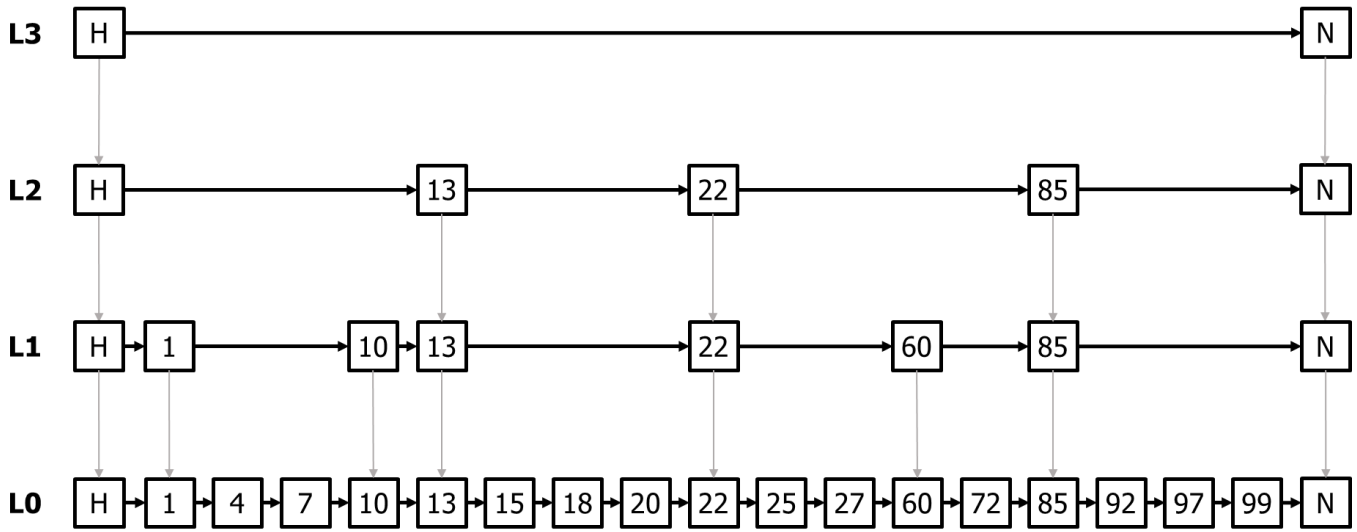
Data Structure

- Skip List - Delete



Lab2. Concurrency

- Lock 구현
  - Skiplist를 다음 3가지 버전으로 구현
    - Without Lock
    - Coarse-grained Lock
    - Fine-grained Lock



Environment Setting

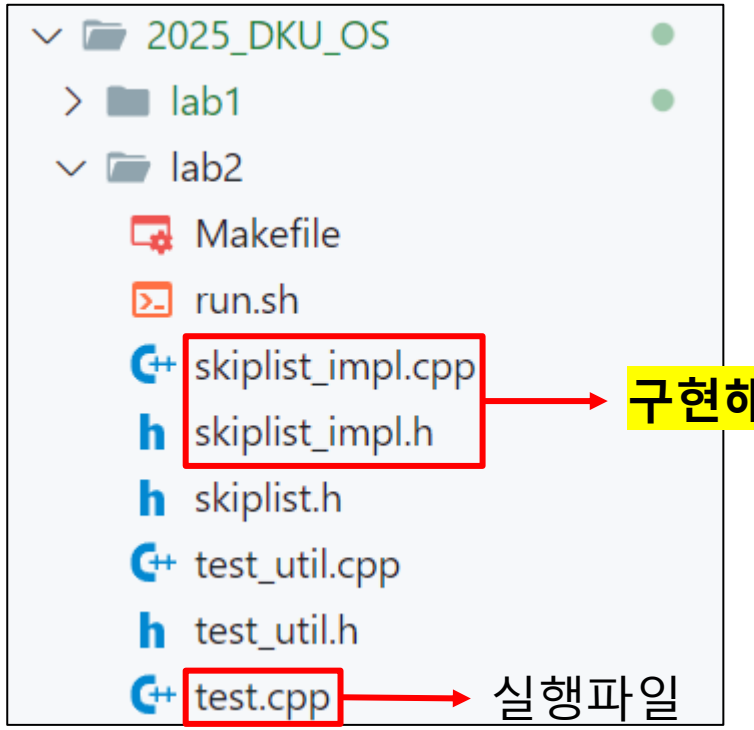
- https://github.com/DKU-EmbeddedSystem-Lab/2025_DKU_OS/
- Ubuntu 사용 (타 OS 사용 X)
- Lab0 매뉴얼을 보고 환경 설정 권장

- Lab 2 환경 구성
  - $ cd 2025_DKU_OS
  - $ git pull origin master
    - 레포지토리 변경 사항 반영
  - $ cd lab2
  - $ run.sh
    - 소스코드 빌드 및 실행

Workload Types

- 유저 request 종류
  - Insert Only
  - Insert + Lookup (50:50)
  - Insert + Lookup + Delete (60:20:20)
- Key & Value
  - 1,000,000개
  - 1 ~ 100,000 사이의 무작위 숫자
- Thread 개수
  - 1, 2, 4, 8 (Without Lock은 스레드 1개만 수행)

Code Flow



구현해야하는 파일

실행파일

<test.cpp>: main 함수 실행

```cpp
int main() {
    ::testing::InitGoogleTest();
    return RUN_ALL_TESTS();
}
```

<test.cpp>: single/multi thread, 워크로드에 따른 테스트 케이스 생성

```cpp
/// @brief SkipListSingleThreadTest Test Case and Parameters
INSTANTIATE_TEST_CASE_P(Default, SkipListSingleThreadTest,
    ::testing::Values(
    // Workload Type, Request Num, Num of Threads
    std::make_tuple(INSERT_ONLY, 1000000, 1),
    std::make_tuple(INSERT_LOOKUP, 1000000, 1),
    std::make_tuple(INSERT_LOOKUP_DELETE, 1000000, 1)
    )
);
```

<test_util.cpp>: 워크로드 생성

```cpp
/// @brief TEST_P 실행 전, 실행 됨
void SkipListTest::SetUp() {
  load_workload();
}
```

<test.cpp>: skiplist 객체 생성

```cpp
/// @brief Test SkipList without lock in single-thread
TEST_P(SkipListSingleThreadTest, Single) {
    skiplist = new SkipList();
}
```

<test_util.cpp>: 실행 및 결과 확인

```cpp
/// @brief TEST_P 실행 후, 실행 됨
void SkipListTest::TearDown() {
  run_workload();
  check_answer();
}
```

Skip list Class

<skiplist.h>: Node structure

```
struct Node {
    int key;          // 키
    int value;        // 밸류
    int upd_cnt;      // 업데이트 횟수
    int level;        // 최대 레벨
    Node** forward; // 레벨 별로 다음 노드를 포인터 배열로 가르킴
};
```

<skiplist.h>: DefaultSkipList class

```
class DefaultSkipList {
    protected:
        int max_level_;     // 최대 레벨
        int current_level_; // 현재 레벨
        float prob_;        // 확률 계수 (0.5)
        Node* header_;      // 헤드 노드

    public:
        DefaultSkipList(int max_level = 16, float prob = 0.5);
        virtual ~DefaultSkipList();
```

<skiplist.h>: DefaultSkipList의 맴버 함수
이 함수들을 상속받아 구현하는 것이 과제

```
/**
 * @brief 삽입 함수 : 인자로 받은 key와 value를 저장한다.
 * 이미 key가 존재하는 경우, 기존 value에 새로운 value를 더하고
 * update count를 1 증가시킨다.
 *
 * @param key
 * @param value
 */
virtual void insert(int key, int value) = 0;

/**
 * @brief 탐색 함수 : 인자로 받은 key의 value를 반환한다.
 * SkipList에 key가 존재하지 않을 경우, 0을 반환한다.
 *
 * @param key
 * @return int
 */
virtual int lookup(int key) = 0;

/**
 * @brief 삭제 함수 : 인자로 받은 key가 저장된 노드를 삭제한다.
 *
 * @param key
 */
virtual void remove(int key) = 0;
```

skiplist_impl.h

- 부모 클래스를 상속받아 사용
  - 필요시 멤버 변수/함수 추가 가능

```
class SkipList : public DefaultSkipList {
    private:

    public:
        // 생성자
        SkipList(int max_level = 16, float prob = 0.5);
        // 소멸자
        ~SkipList();

        void insert(int key, int value) override;
        int lookup(int key) override;
        void remove(int key) override;
};
```

```
class CoarseSkipList : public DefaultSkipList {
    private:
        pthread_mutex_t mutex_lock;

    public:
        // 생성자
        CoarseSkipList(int max_level = 16, float prob = 0.5);
        // 소멸자
        ~CoarseSkipList();

        void insert(int key, int value) override;
        int lookup(int key) override;
        void remove(int key) override;
};
```

```
struct FineNode : public Node {
    pthread_mutex_t lock;
};
class FineSkipList : public DefaultSkipList {
    private:

    public:
        // 생성자
        FineSkipList(int max_level = 16, float prob = 0.5);
        // 소멸자
        ~FineSkipList();

        void insert(int key, int value) override;
        int lookup(int key) override;
        void remove(int key) override;
};
```

skiplist_impl.cpp (Implementation)

- Common
  - 생성자, 소멸자 및 레벨 생성 함수 제공

```
/*
*   DKU Operating System Lab
*        Lab2 (Concurrent Data Structure : SkipList)
*        Student id :
*        Student name :
*        Date :
*/
```

필수!
상단에 인적정보 기입

```
DefaultSkipList::DefaultSkipList(int max_level, float prob) {
    this->max_level_ = max_level;
    this->prob_ = prob;
    this->current_level_ = 0;

    // 헤더 노드 생성
    header_ = new Node();
    header_->key = -1;  // 가장 작은 키 값
    header_->value = -1;
    header_->upd_cnt = 0;
    header_->level = max_level_;

    // 헤더 노드의 forward 배열 초기화
    header_->forward = new Node*[max_level_ + 1];
    for (int i = 0; i <= max_level_; i++) {
        header_->forward[i] = nullptr;
    }

    // 난수 생성을 위한 시드 설정
    srand(time(nullptr));
}
```

```
DefaultSkipList::~DefaultSkipList() {
    Node* current = header_;
    Node* temp;

    while(current) {
        temp = current->forward[0];
        delete[] current->forward;
        delete current;
        current = temp;
    }
}
```

```
int DefaultSkipList::random_level() {
    int level = 0;
    while (((double)rand() / RAND_MAX) < prob_ &&
           level < max_level_) {
        level++;
    }
    return level;
}
```

skiplist_impl.cpp (Implementation)

- SkipList
  - Without Lock 버전, single thread 전용

```
/*
*   DKU Operating System Lab
*       Lab2 (Concurrent Data Structure : SkipList)
*       Student id :
*       Student name :
*       Date :
*/
```

필수!
상단에 인적정보 기입

```
// SkipList 생성자
SkipList::SkipList(int max_level, float prob) : DefaultSkipList(max_level, prob) {}

// SkipList 소멸자
SkipList::~SkipList() {}

void SkipList::insert(int key, int value) {
    // Todo
}

int SkipList::lookup(int key) {
    // Todo
}

void SkipList::remove(int key) {
    // Todo
}
```

→ 구현할 부분

skiplist_impl.cpp (Implementation)

- CoarseSkipList
  - Coarse-grained Lock 버전

```
/*
*   DKU Operating System Lab
*        Lab2 (Concurrent Data Structure : SkipList)
*        Student id :
*        Student name :
*        Date :
*/
```

필수!
상단에 인적정보 기입

```
// CoarseSkipList 생성자
CoarseSkipList::CoarseSkipList(int max_level, float prob) : DefaultSkipList(max_level, prob) {
    pthread_mutex_init(&mutex_lock, nullptr);
}

// CoarseSkipList 소멸자
CoarseSkipList::~CoarseSkipList() {
    pthread_mutex_destroy(&mutex_lock);
}

void CoarseSkipList::insert(int key, int value) {
    // Todo
}

int CoarseSkipList::lookup(int key) {
    // Todo
}

void CoarseSkipList::remove(int key) {
    // Todo
}
```

→ 구현할 부분

skiplist_impl.cpp (Implementation)

```
/*
*   DKU Operating System Lab
*       Lab2 (Concurrent Data Structure : SkipList)
*       Student id :
*       Student name :
*       Date :
*/
```

필수!
상단에 인적정보 기입

- FineSkipList
  - Fine-grained Lock 버전, FineNode 사용
  - 생성자 및 소멸자 새로 정의

```
FineSkipList::FineSkipList(int max_level, float prob) : DefaultSkipList(max_level, prob) {
    // 헤더 노드를 FineNode로 대체
    delete header_;
    header_ = new FineNode();
    header_->key = -1;
    header_->value = -1;
    header_->upd_cnt = 0;
    header_->level = max_level_;

    header_->forward = new Node*[max_level_ + 1];
    for (int i = 0; i <= max_level_; i++) {
        header_->forward[i] = nullptr;
    }

    pthread_mutex_init(&((FineNode*)header_)->lock, nullptr);
}
```

```
FineSkipList::~FineSkipList() {
    Node* current = header_;
    Node* temp;

    while (current) {
        temp = current->forward[0];
        pthread_mutex_destroy(&((FineNode*)current)->lock);
        delete[] current->forward;
        delete current;
        current = temp;
    }
}
```

```
void FineSkipList::insert(int key, int value) {
    // Todo
}

int FineSkipList::lookup(int key) {
    // Todo
}

void FineSkipList::remove(int key) {
    // Todo
}
```

→ 구현할 부분

Results

- 프로그램 실행 결과

```
[==========] Running 27 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 9 tests from Default/SkipListSingleThreadTest
[ RUN      ] Default/SkipListSingleThreadTest.Single/0
Execution time: 275 ms
[       OK ] Default/SkipListSingleThreadTest.Single/0 (579 ms)
[ RUN      ] Default/SkipListSingleThreadTest.Single/1
Execution time: 270 ms
[       OK ] Default/SkipListSingleThreadTest.Single/1 (468 ms)
[ RUN      ] Default/SkipListSingleThreadTest.Single/2
Execution time: 308 ms
[       OK ] Default/SkipListSingleThreadTest.Single/2 (621 ms)
[ RUN      ] Default/SkipListSingleThreadTest.Coarse/0
Execution time: 278 ms
[       OK ] Default/SkipListSingleThreadTest.Coarse/0 (580 ms)
[ RUN      ] Default/SkipListSingleThreadTest.Coarse/1
Execution time: 285 ms
[       OK ] Default/SkipListSingleThreadTest.Coarse/1 (486 ms)
```

Project Requirements

- 요구사항
  - 본 과제에서는 주어진 Lock 적용 범위에 맞춰 Skip list를 구현하는 것을 목표로 한다
    - Without Lock, Coarse-grained Lock, Fine-grained Lock
  - Skip list는 유저의 요청에 따라 insert, lookup, remove 연산을 수행한다
    - Insert: 인자로 받은 key와 value를 저장, 이미 key가 존재하는 경우 기존 value에 새 value를 더하고 update count를 1 증가
    - Lookup: 인자로 받은 key의 value를 반환한다. key가 존재하지 않을 경우, 0을 반환
    - Remove: 인자로 받은 key가 저장된 노드를 삭제

Workload Format

- 입력 (Workload)
  - key의 개수: 1,000,000개
  - key, value의 범위: 1 ≤ key, value ≤ 100,000
  - 첫 번째 요청은 항상 Insert {key=1, value=1}로 설정된다
  - key, value는 랜덤한 값으로 생성된다
  - 1, 2, 4, 8개의 스레드를 사용하여 테스트 진행 (Without Lock 버전은 1개의 스레드만 사용)

Implementation Details

- 구현해야 하는 내용
  - skiplist_impl.cpp에서 SkipList, CoarseSkipList, FineSkipList 클래스를 구현한다
    - 각 클래스의 생성자, 소멸자와 레벨 생성 함수는 기본적으로 제공된다
    - 각 클래스에서 1) insert, 2) lookup, 3) remove 함수를 구현해야 한다
      - 클래스에 적용되는 Lock의 범위에 맞춰 구현을 진행해야 한다
    - 반드시 C++로 구현해야 하고 라이브러리는 C++ STL만 사용 가능하다
    - skiplist_impl.cpp, skiplist_impl.h 외 다른 파일은 수정, 추가, 제출이 불가하다
      - 필요시 맴버 변수/함수 추가 가능

Code Submission

- 양식
  - 제목: os_lab2_학번_이름.cpp
    - Ex) os_lab2_32190617_김보승.cpp + os_lab2_32190617_김보승.h
  - 코드 상단에 작성자 정보 기입
  - 코드 설명하는 주석 달기
  - run.sh로 컴파일이 되고, 정상적으로 실행이 되어야 함
  - C++ 로 작성

Report Guidelines

- 보고서 내용
  - 구현한 소스코드 설명
  - **문제 풀이**
    - 멀티 스레드 환경에서 Skiplist에 접근할 때, 요청 수행 순서를 일관되게 보장하는 방법을 설명하라
    - Fine-grained Skiplist에서는 동일한 키에 대해 서로 다른 스레드에서 동시에 insert와 remove를 요청할 경우, 요청의 처리 순서에 따라 결과가 어떻게 달라질 수 있는지 설명하라
  - Discussion
    - 여러 워크로드에서, lock이 적용되는 범위가 다를 경우, thread의 개수에 따른 실행시간 비교 분석
    - 과제를 진행하며 새롭게 배운 점/ 어려웠던 점
    - 기타 내용 자유롭게 작성

Report Submission

- 양식
  - 제목: os_lab2_학번_이름.pdf
    - Ex) os_lab2_32190617_김보승.pdf
  - 코드 및 터미널 화면 첨부 시 흰색 바탕으로 캡처
    - VS code 사용자: VS Code 테마 변경 방법
    - Linux 터미널: 리눅스 터미널 색상 변경 방법

Grading Criteria

- 구현
  - 실행 결과가 정답과 일치하는가?
    - Multi thread로 실행한 결과와 single thread로 실행한 결과가 동일한지 확인
  - 스레드 수가 증가할 때 Fine-grained 구현의 성능이 Coarse-grained 구현보다 우수한가?
  - 소스코드에 각 줄(or 코드 블록)마다 주석이 적절하게 작성되어 있는가?
  - 주어진 run.sh을 통해 컴파일 및 실행이 정상적으로 동작하는가?
    - 별도의 컴파일 방법 또는 실행 방법을 보고서에 서술하는 것은 인정되지 않음
- 보고서
  - Lock 적용 범위에 따른 성능을 그래프 등의 지표를 활용하여 비교/분석하였는가?
  - 구현한 소스코드에 대한 설명이 충분히 잘 작성되어 있는가?
  - 문제에 대한 해결 방법을 명확히 서술하였는가?

Grading Criteria

- 총점 100점 = 구현 65점 + 보고서 25점 + 양식 10점

- 유의 사항
  - 지각 제출시, 하루에 10% 감점
  - 소스코드 제출 기준 미 준수 시 → 구현 0점 + 형식 점수 0점
    - ex) 소스코드 텍스트로 제출, make/실행 안됨, …
  - 인적사항 주석 미 작성, 파일명/형식 미 준수 시 → 형식 점수 0점

구분	세부사항	점수
구현	Without Lock	20
	Coarse-grained Lock	20
	Fine-grained Lock	25
보고서	구현 내용 설명	10
	문제	5
	Discussion	10
양식	주어진 양식 준수	10

Submission Guideline

- 제출: https://forms.gle/m3TUns69gZUXk3RT7
  - 구글 폼 양식에 맞춰 제출

- 기한: **2025.05.14 23:59**까지

Thank You
Q&A?

2025.04.28

T.A. 김보승

bskim1102@dankook.ac.kr