離散数学

スキップリスト
(スキップグラフ)

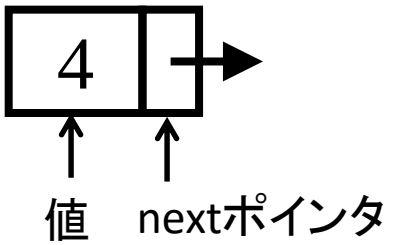
落合 秀也

今日の内容

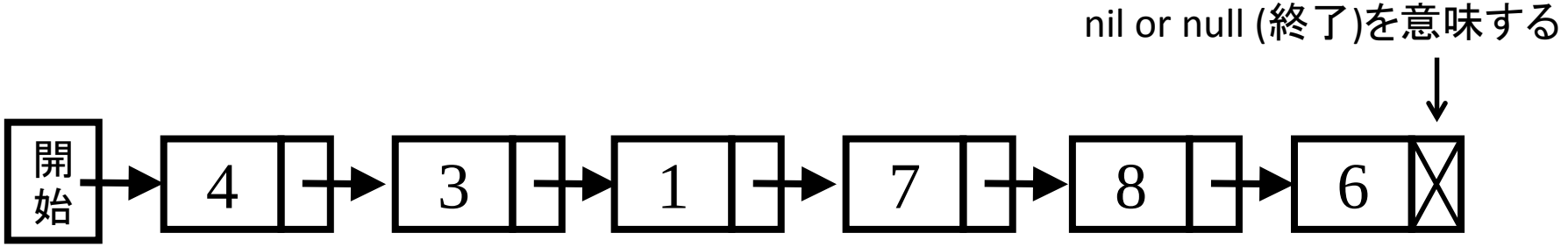
- 連結リスト（復習）
- スキップリスト
- スキップグラフ

連結リスト (復習)

- 「値とnextポインタ」で構成された要素(ノード)を連結して作られる値のリスト




```
struct list_element {
  int value;
  struct list_element* next;
};
```

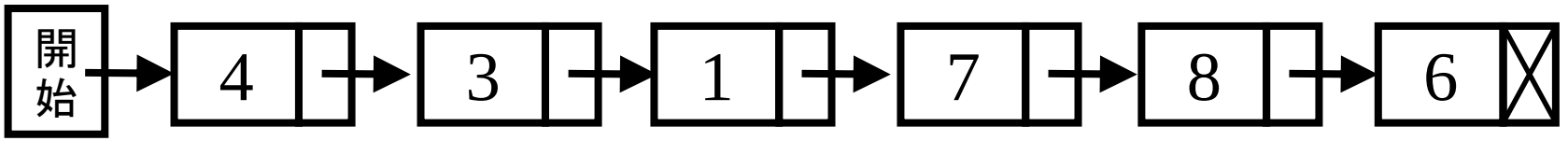


連結リストにおける検索

値xを持つノードを検索(search)する場合

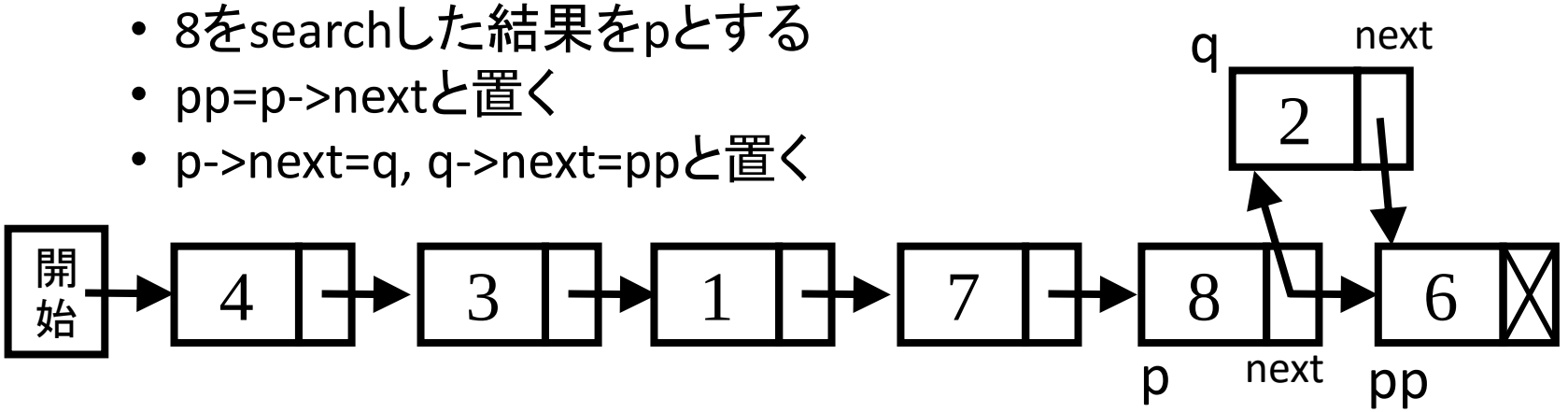
```
struct list_element* search(int x){
   struct list_element* p=start;
   while((p=p->next)!=null){
     if ( p->value == x ){
        return p;
     }
   }
   return null;
}
```

開始点からnextポインタをつたって、そのノードの値がxと一致するまで探索する
(あるいはnextポインタ=nullで終了する)

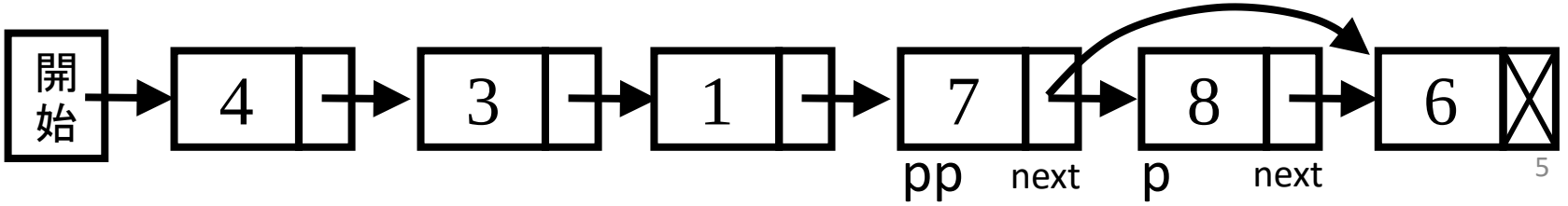


連結リストにおける挿入・削除

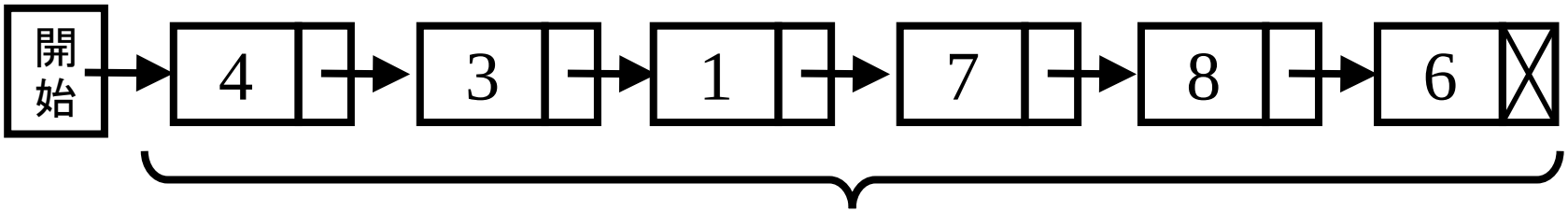
- 8の後ろに2を入れたい場合: insert(8,2)の操作
  - 2へのポインタをqとする
  - 8をsearchした結果をpとする
  - pp=p->nextと置く
  - p->next=q, q->next=ppと置く



- 8を削除したい場合: delete(8)の操作
  - 8をsearchした結果をpとする　(8の前をppとする)
  - pp->next=p->nextと置く



連結リストの計算量



- 検索の場合
  - 検索対象の出現パターンに依存するが、すべて等しい確率で出現すると仮定すると、期待値 n/2回の操作でヒットする。
  - ヒットしない場合: n回の操作が必要

    ➔いずれにしても O(n)
- 挿入・削除の場合
  - 検索したのちに、ポインタの組換え操作を行う: O(n)
  - 先頭にだけ挿入する場合は O(1)

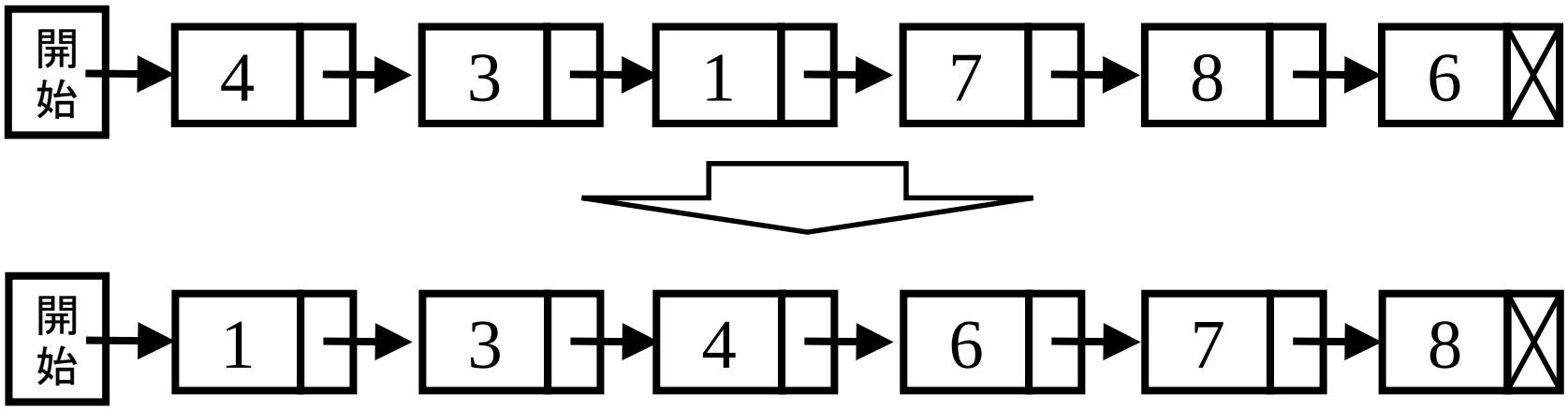
今日の内容

- 連結リスト（復習）
- スキップリスト
- スキップグラフ

連結リストの高速化を考える

- 順序関係が定義される値であれば、順序関係に基づいて、ソートしておくことによって、リストでの検索効率を向上させることができる

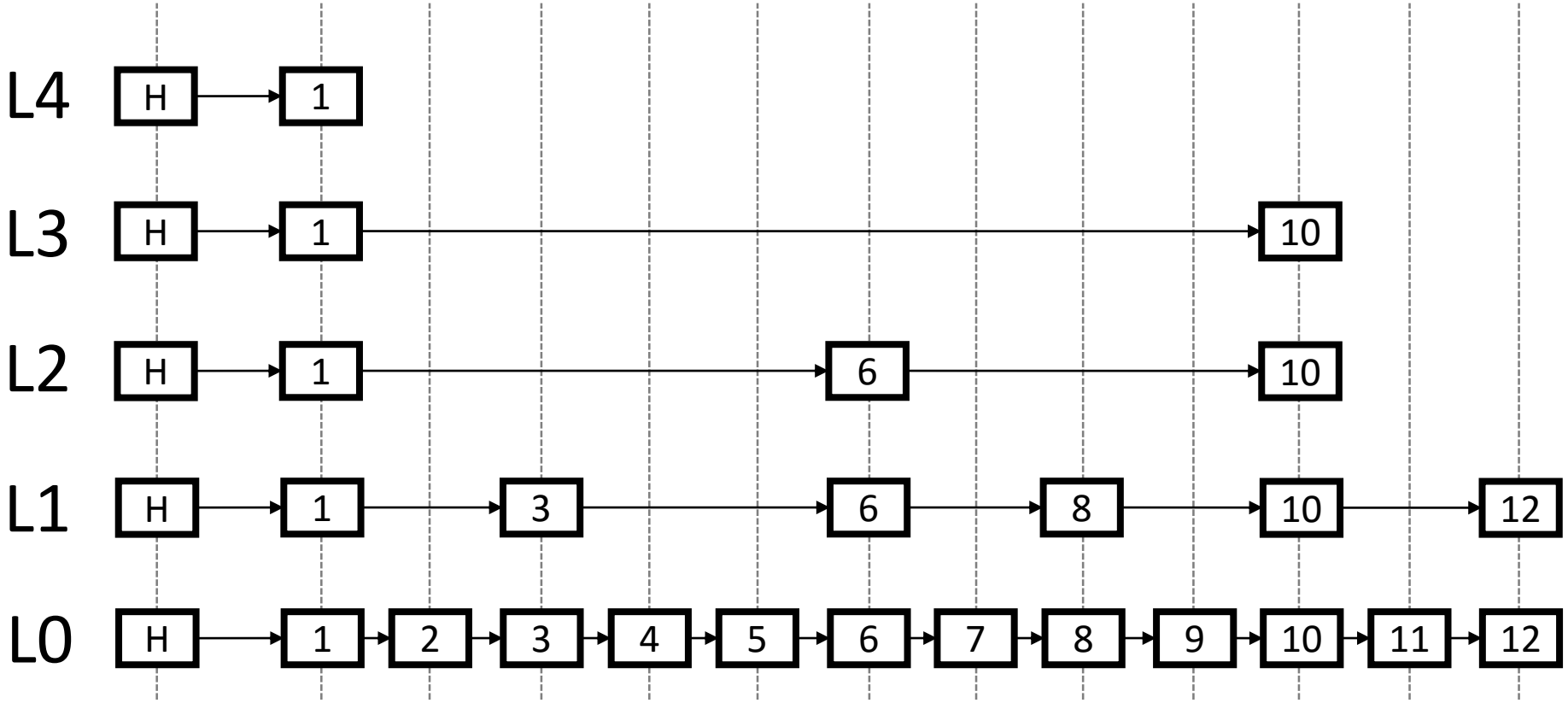
  ➔ 具体的には スキップリスト



昇順にソートされた連結リストの例

スキップリストの高速化原理

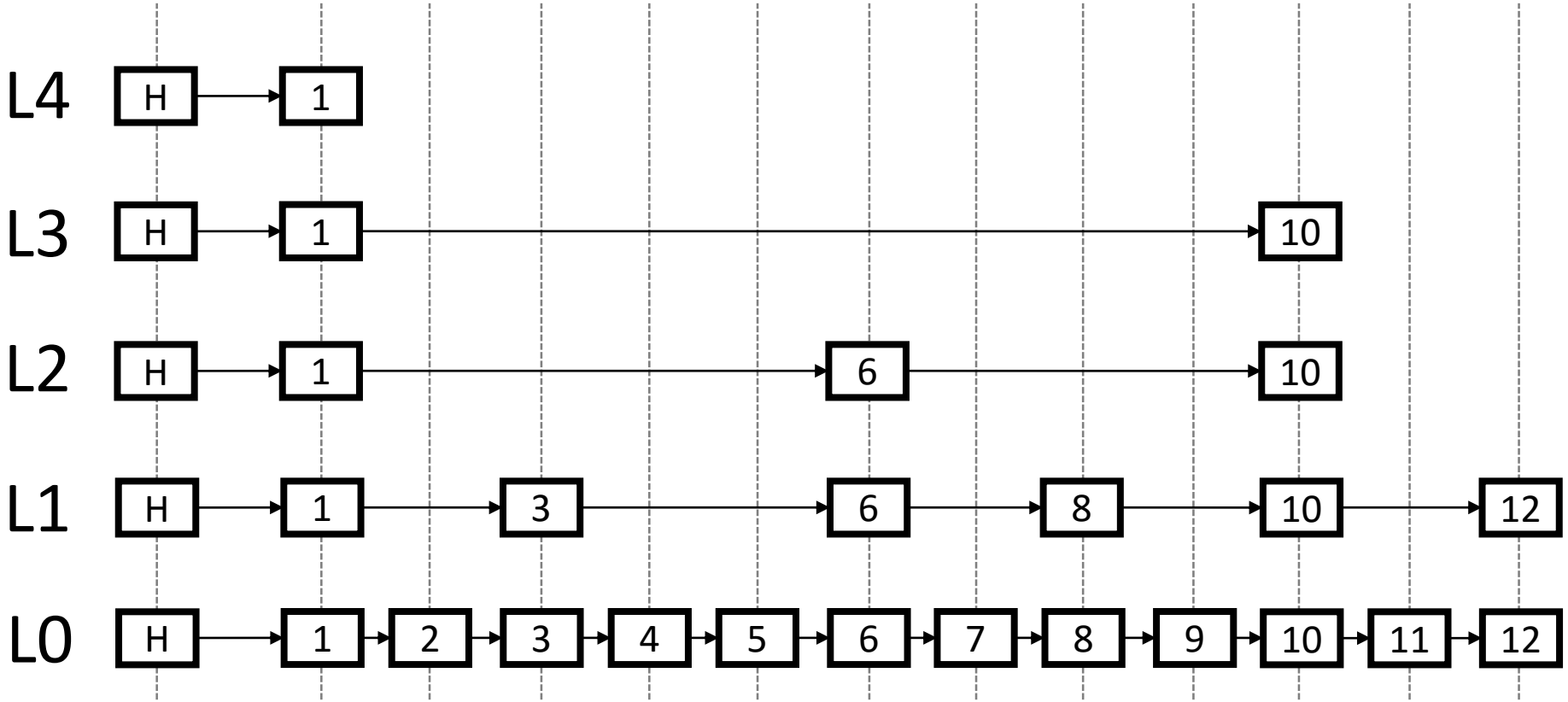
- 通常の連結リストは各駅停車である
- スキップリストでは 快速、準急、急行、特急 を用意する



- 高速化のレベルを L0, L1, L2, L3, … と呼ぶことにする

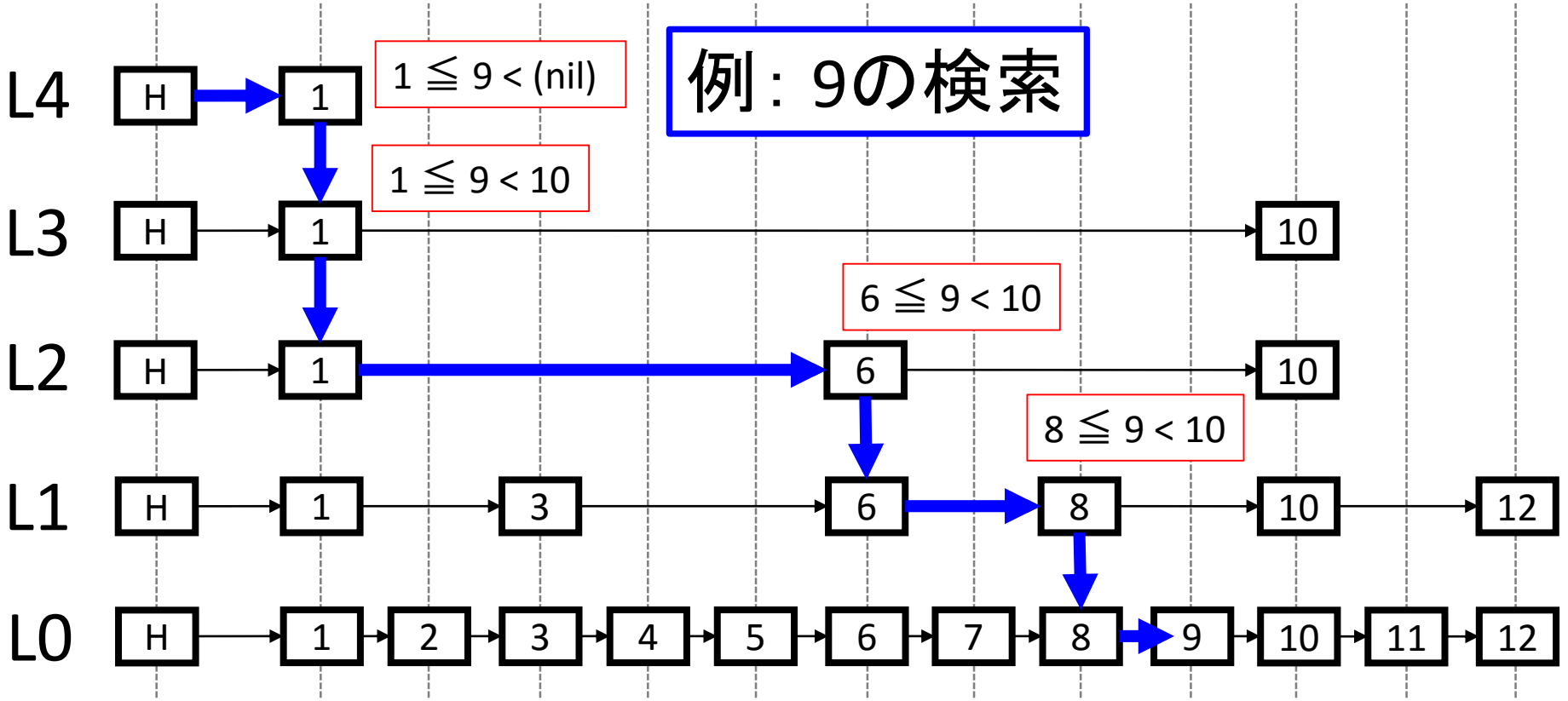
スキップリストの階層の作り方

- スキップリストでは「レベル L_i に属するノードが、確率pで、レベルL_{i+1}に属する」ように階層を作る。pには1/2や1/4などが使われる。
- Headの直後のノードはすべてのレベルに属する
- 平均的に　L_{i+1}に属するノード数 = p × L_iに属するノード数　となる。
- L_0のノード数をnとすると、L_i に属するノード数の期待値は np^i となる。



スキップリストの検索

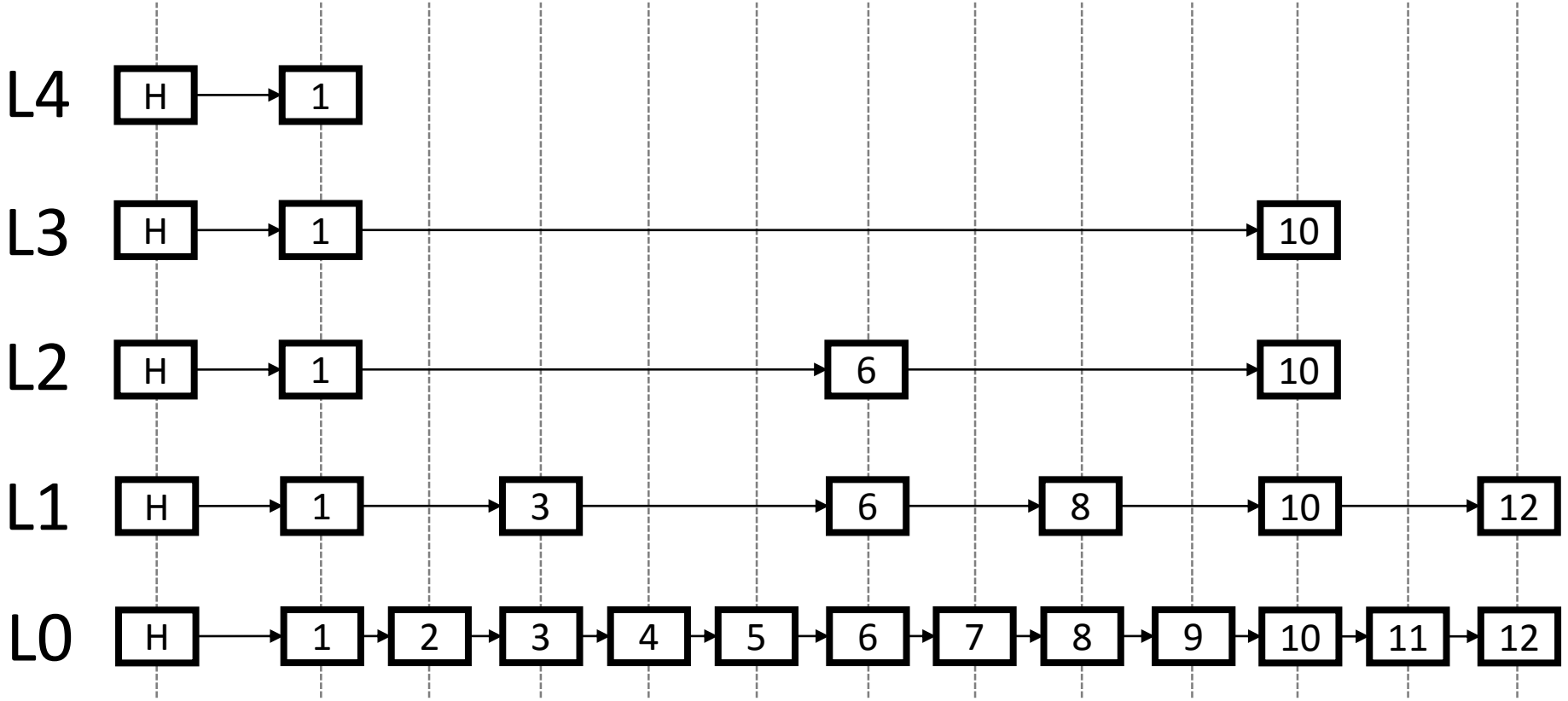
- x を検索する場合を考える
  - 上位の階層から検索を始める
  - L_i で $p_i \leqq x < q_i$ (or nil) となる p_i, q_iを見つける (p_i, q_iはL_iで直前直後の関係にあるものとする)
  - L_{i-1} で p_iから開始し、$p_{i-1} \leqq x < q_{i-1}$となるとなる p_{i-1}, q_{i-1}を見つける (p_{i-1}, q_{i-1}はL_{i-1}で直前直後の関係にあるものとする)
  - 同様の操作を繰り返す (どこかの時点で、x=pとなればそこで終了。あるいはL_0まで検索して見つからなければ終了)



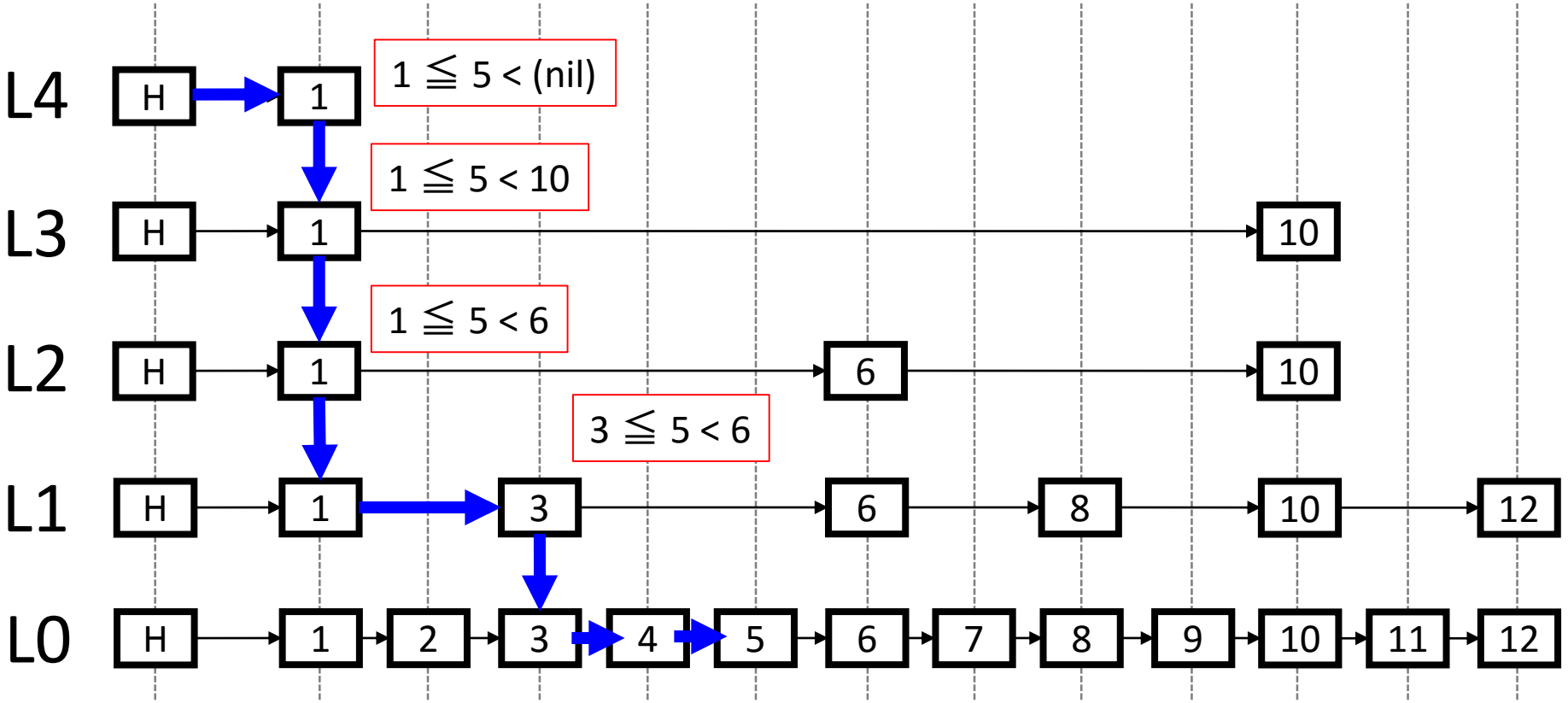
練習

- 以下のスキップリストにおいて
  - 5の検索
  - 7の検索
  - 11の検索
  - 12の検索

を行った際の探索経路をそれぞれ図示せよ (水平移動の操作数も数えてみよ)

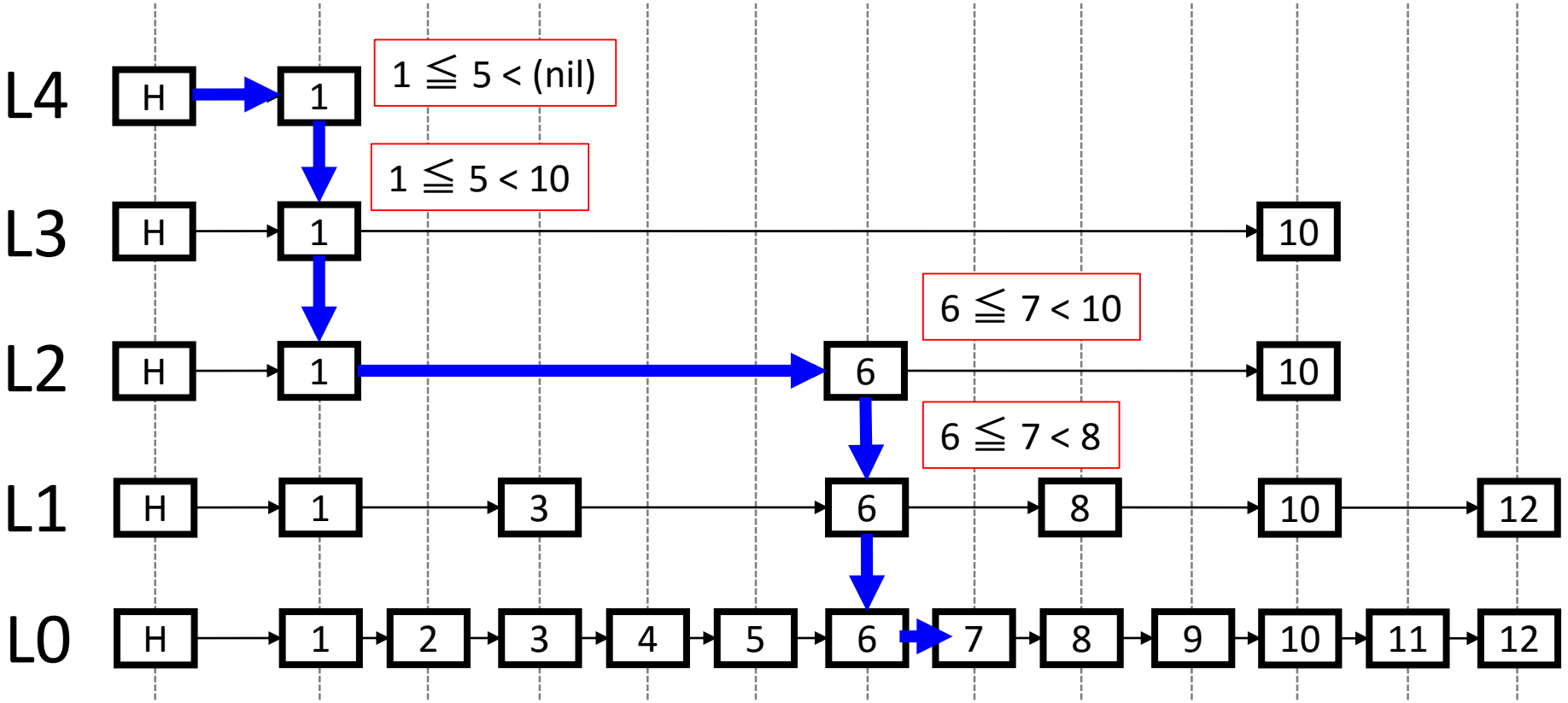


解1/4: 5の検索



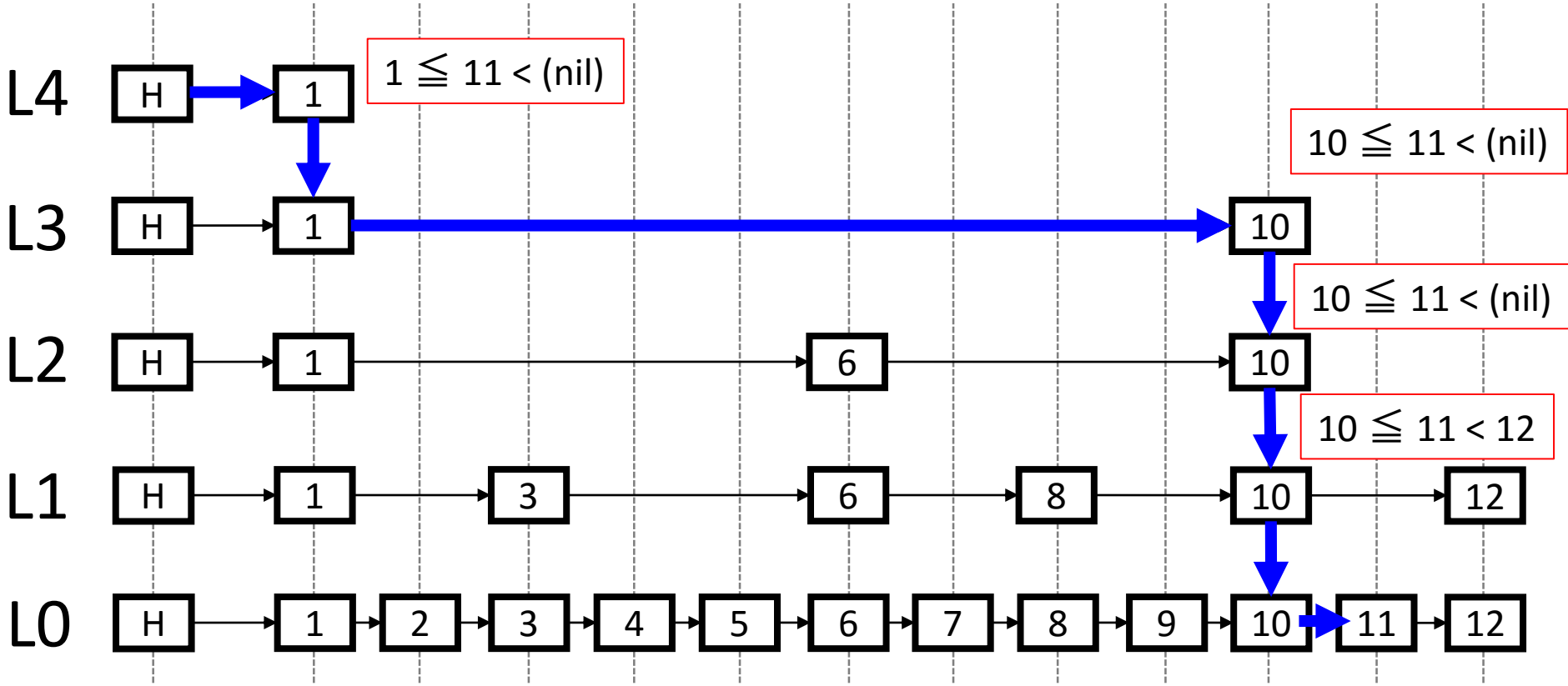
水平移動の操作数: 4回

解2/4: 7の検索



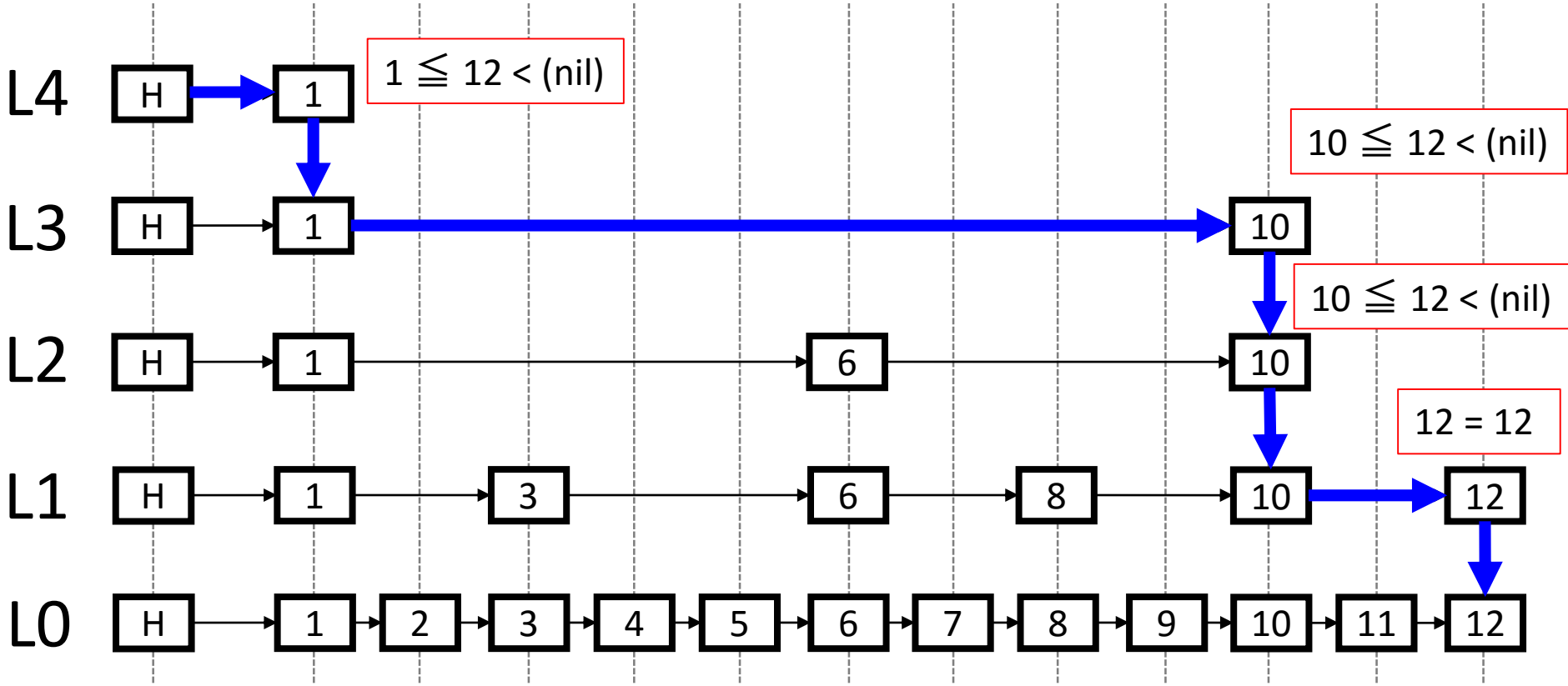
水平移動の操作数: 3回

解3/4: 11の検索



水平移動の操作数: 3回

解4/4: 12の検索



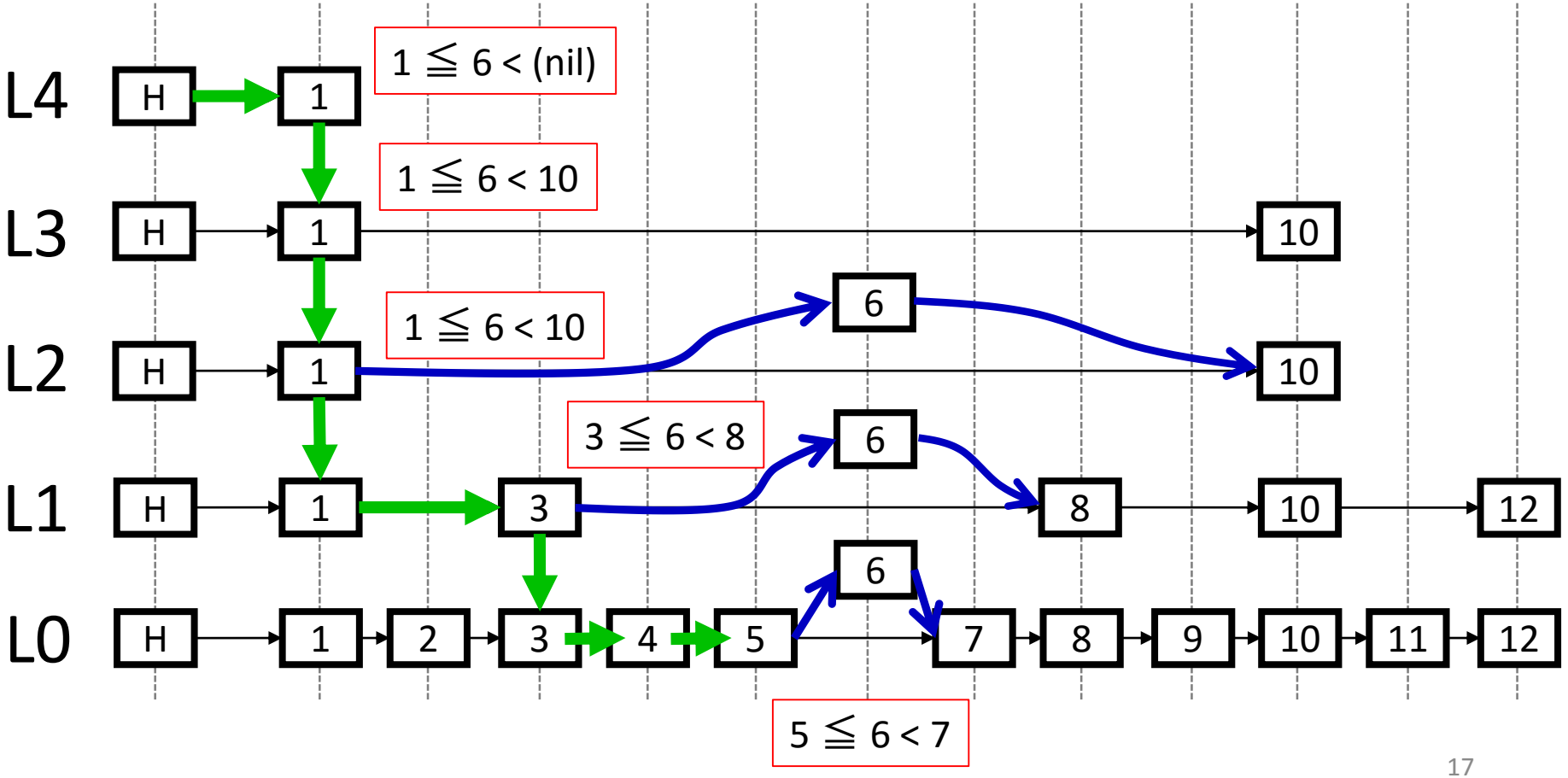
水平移動の操作数: 3回

スキップリストへのノードの挿入

6 を挿入する場合 : insert(6)

(1) 6がどのレベルにまで入るかを乱数を用いて確率的に選ぶ → 以下の例では L_2まで

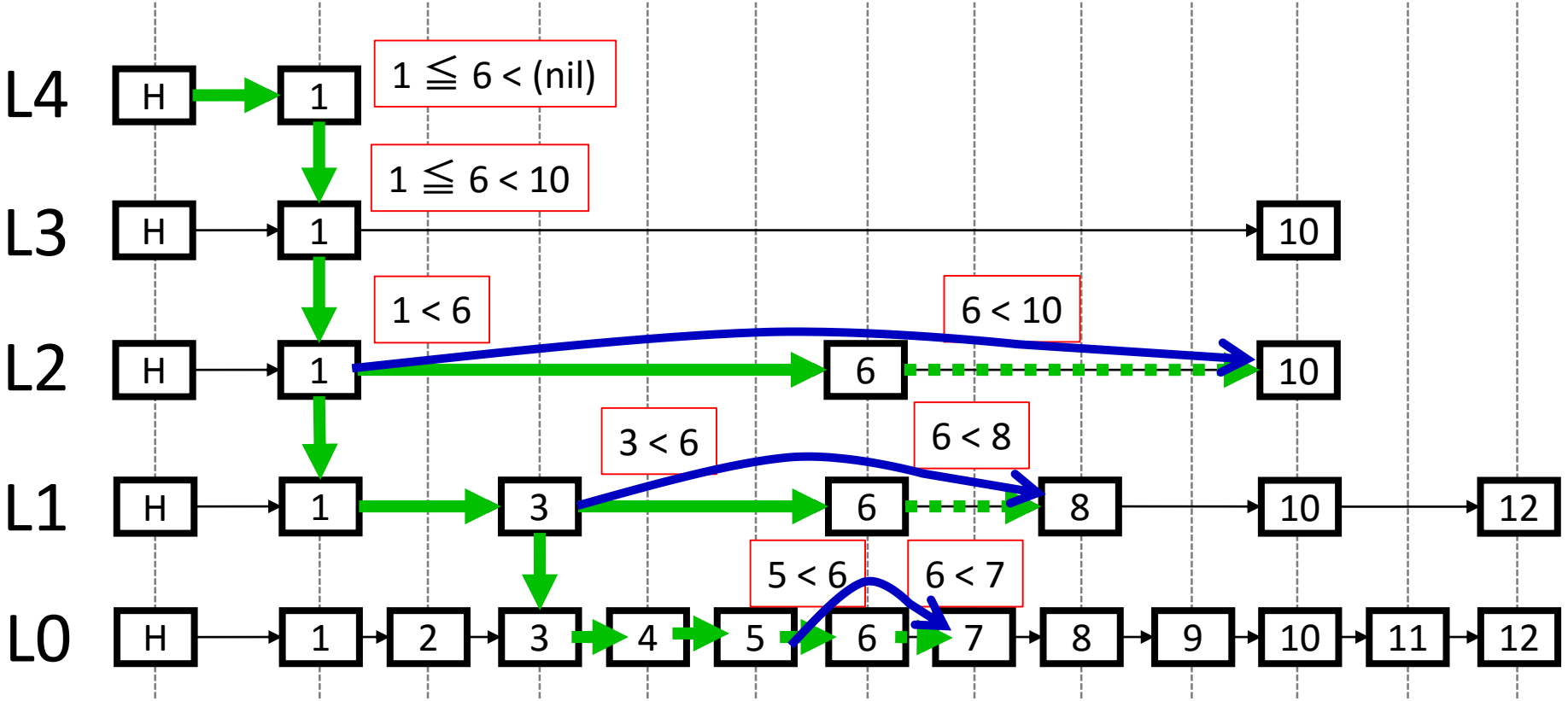
(2) p<6<q となるp, qを該当する各レベルで求め、p.next = &6; 6.next = q とする



スキップリストからのノードの削除

6 を削除する場合 : delete(6)

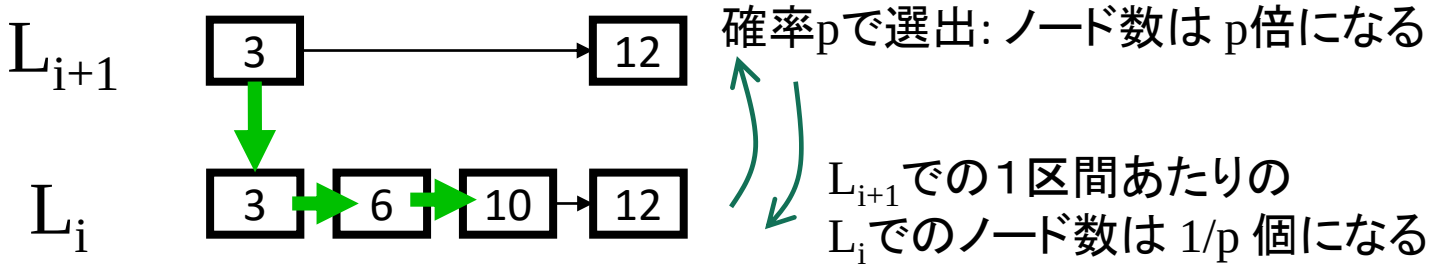
p<6<q となるp, qを該当する各レベルで求め、p.next = q とする



スキップリストの計算量

- スキップリストは、確率的に作られる。そのため、期待値で考えることにする (以下、nはリスト長, pは確率)
  - まず、階層の高さN は $np^N = 1$ より $(1/p)^N = n$

    $\therefore\ N = \log_{1/p} n$
  - 各レベルでのホップ数は、オーダーで 1/p



  - よって、探索の数は
    - $N \times 1/p = (\log_{1/p} n) / p$

- 通常 pは定数なので O(log n) となる

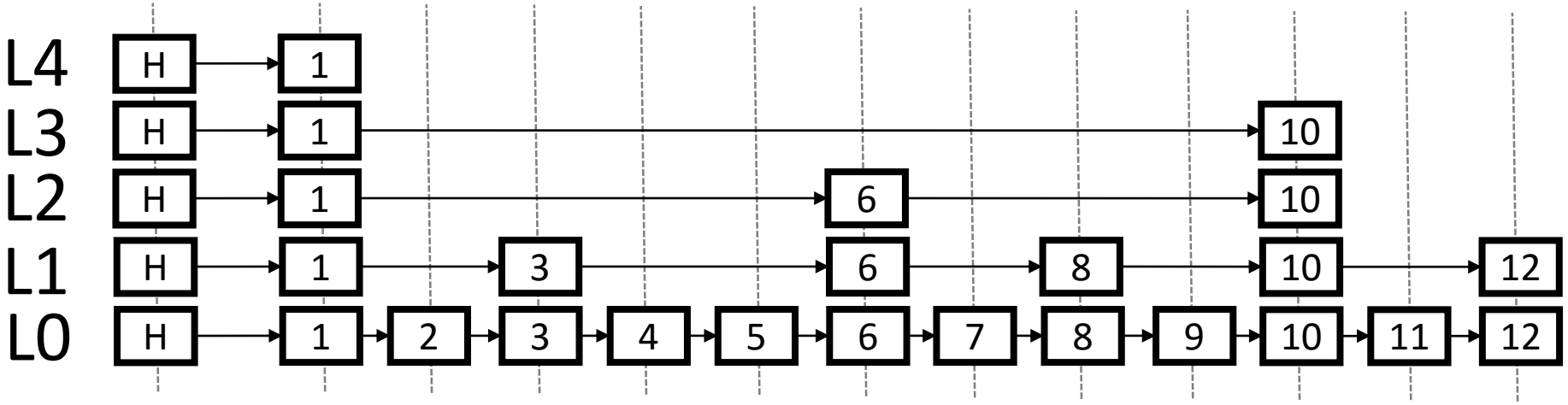
スキップリストの強み

- 各ノードをインターネット上のPCとして考えてみよう

  = 分散環境を考える

- 検索は、広域に渡って行われるが、ノードの追加・削除の操作は、各レベルでの隣接ノードとの関係で行われる

  ⇔ 平衡二分探索木の場合は、広域ロックをする必要がある。



今日の内容

- 連結リスト（復習）
- スキップリスト
- スキップグラフ

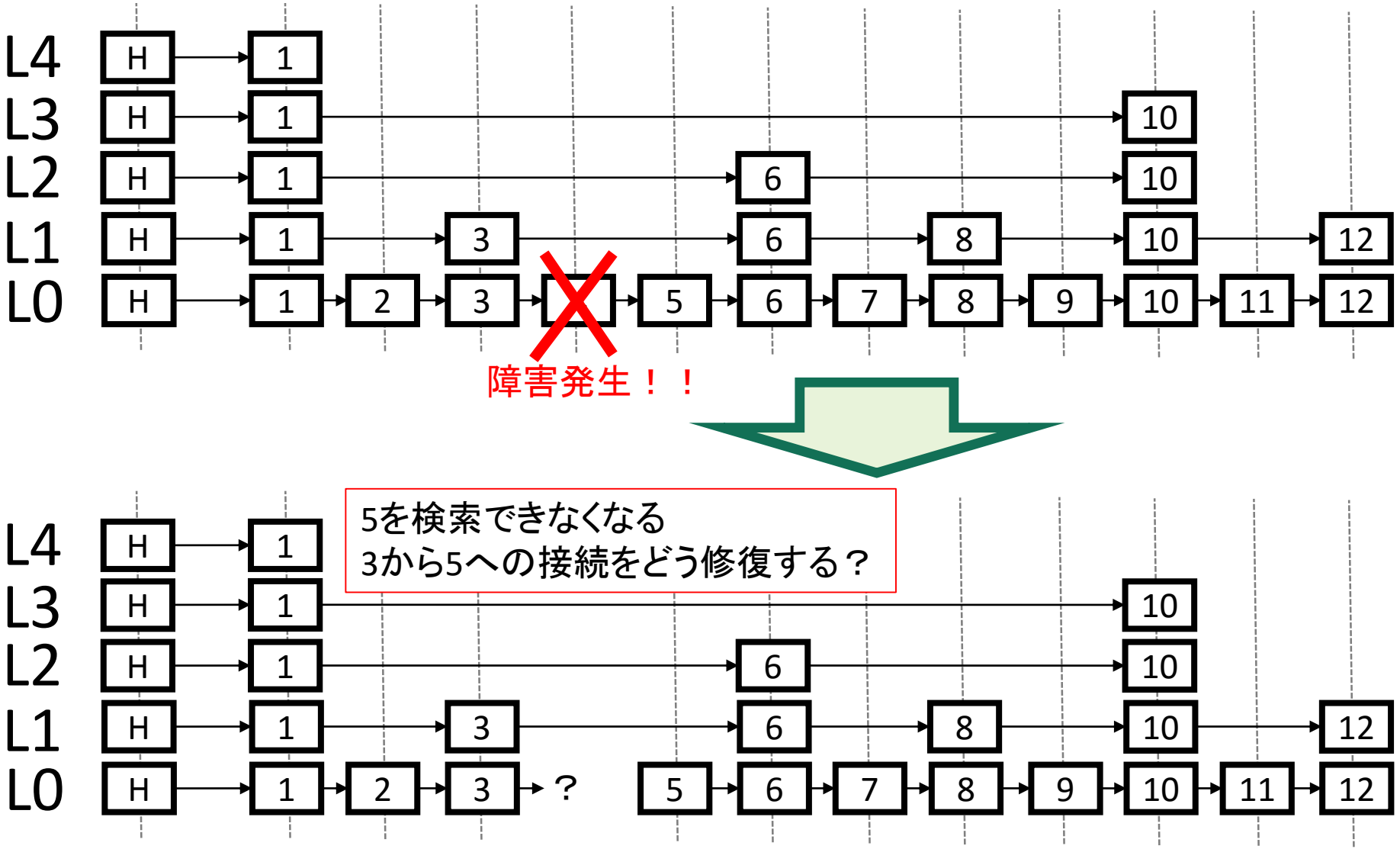
スキップグラフ

- スキップリストをベースに(分散環境への応用を目的に)開発されたデータ構造
  - 完全な自律分散を実現
  - 耐障害性を向上

- スキップグラフの特徴
  - 双方向連結リスト
  - どのノードからでも検索を開始できる
  - 各レベルに複数の双方向連結リストが存在

⇔ スキップリストの特徴
  - 単方向連結リスト。探索開始ノードが決まっていた。各レベルには一つの連結リスト
  - 障害性に強くない (1ノードが消えると、修復が難しい)

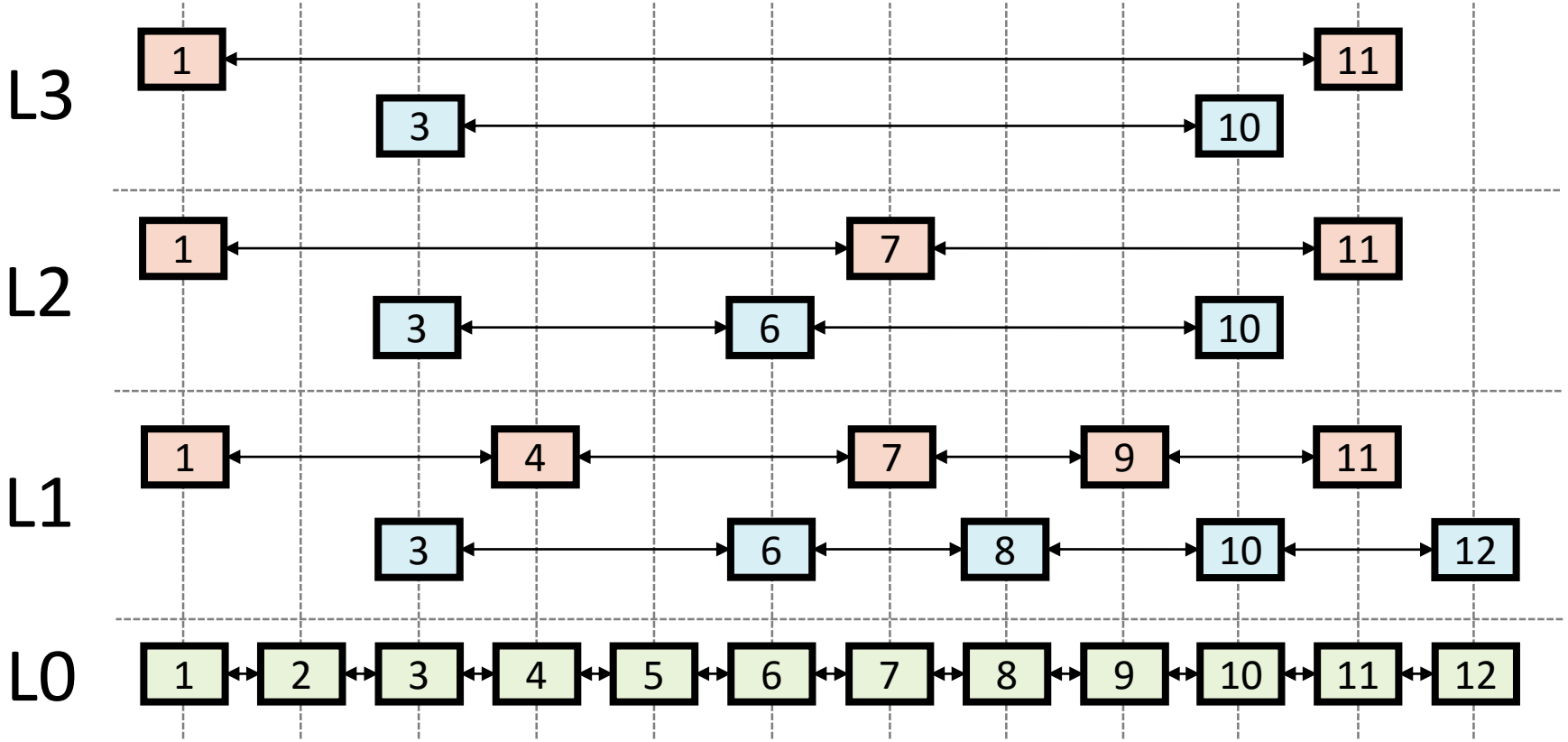
スキップリストの耐障害性



スキップグラフ

- 双方向連結リスト
- どのノードからでも検索を開始できる
- 各レベルに複数の連結リストが存在

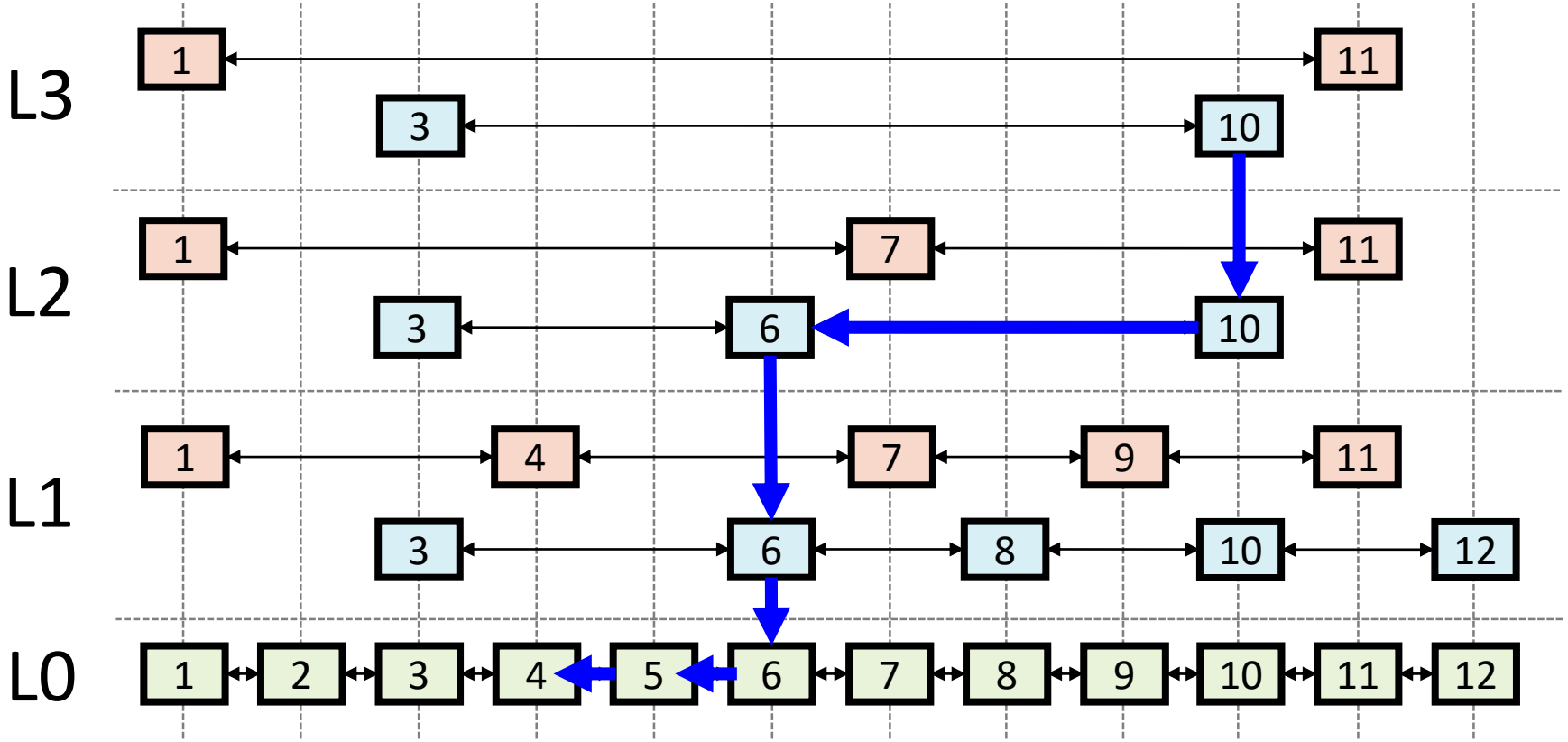
完全に分散
耐障害性の向上



スキップグラフでの検索方法

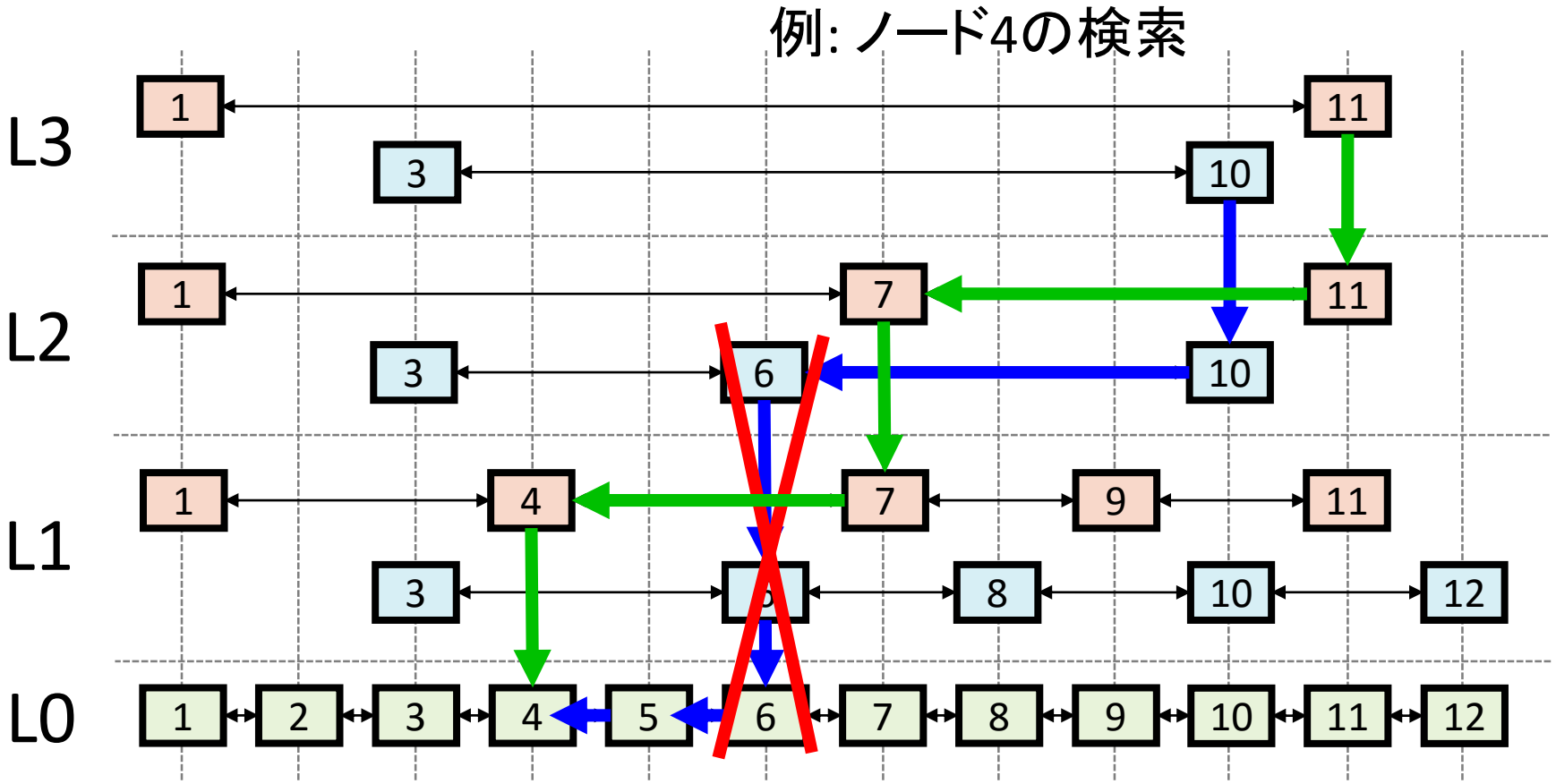
スキップリストと同じように考えれば良い

例: ノード10 から ノード4を検索する場合



スキップグラフの耐障害性

- 特定のノードが消失して、ある探索パスで到達できなくても、別の探索パス(=バックアップパス)が存在すれば、耐障害性がある、と言える



今日の内容

- 連結リスト（復習）
- スキップリスト
- スキップグラフ