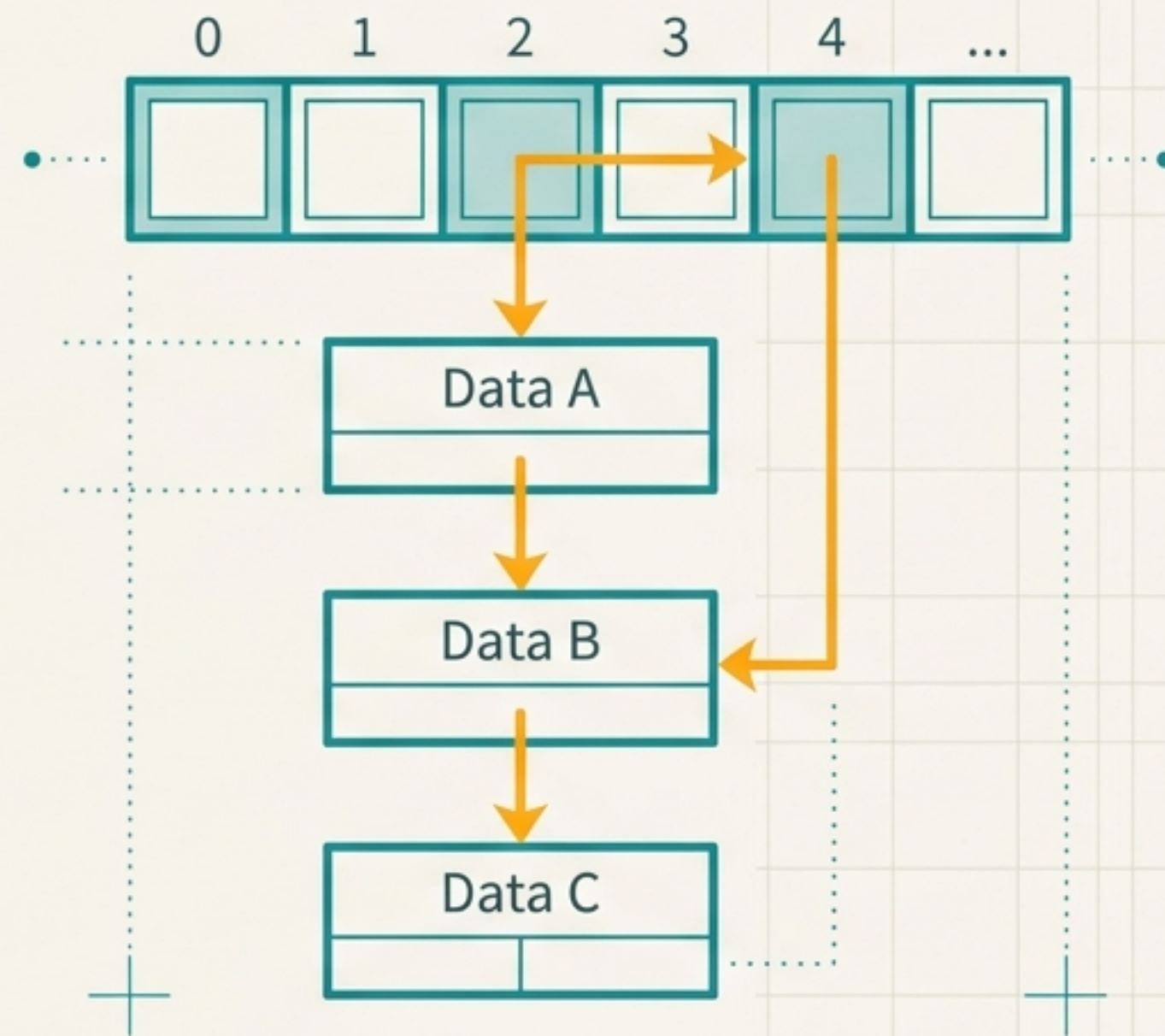


チェイン法（連鎖法）を視覚的に理解する

応用情報技術者試験 過去問マスター

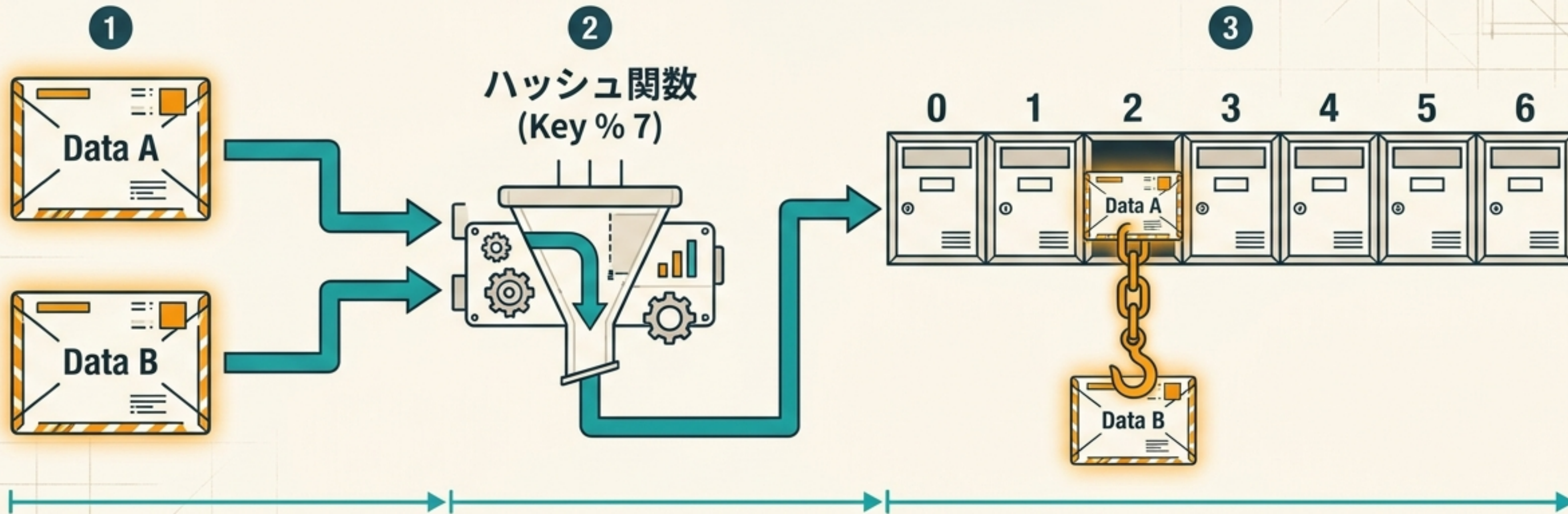
コードを読む前に、データ構造の
「物理的な動き」をイメージしよう。



【台本】 皆さん、こんにちは。今回は応用情報技術者試験の過去問から、初心者がつまずきやすい「チェイン法」のアルゴリズム問題を解き明かします。コードを1行ずつ暗記するのではなく、視覚的なヒントを使ってデータがどう動くのかを理解していきましょう。

ハッシュ法とチェーン法の基本概念

衝突 (Collision) への備え = チェイン法



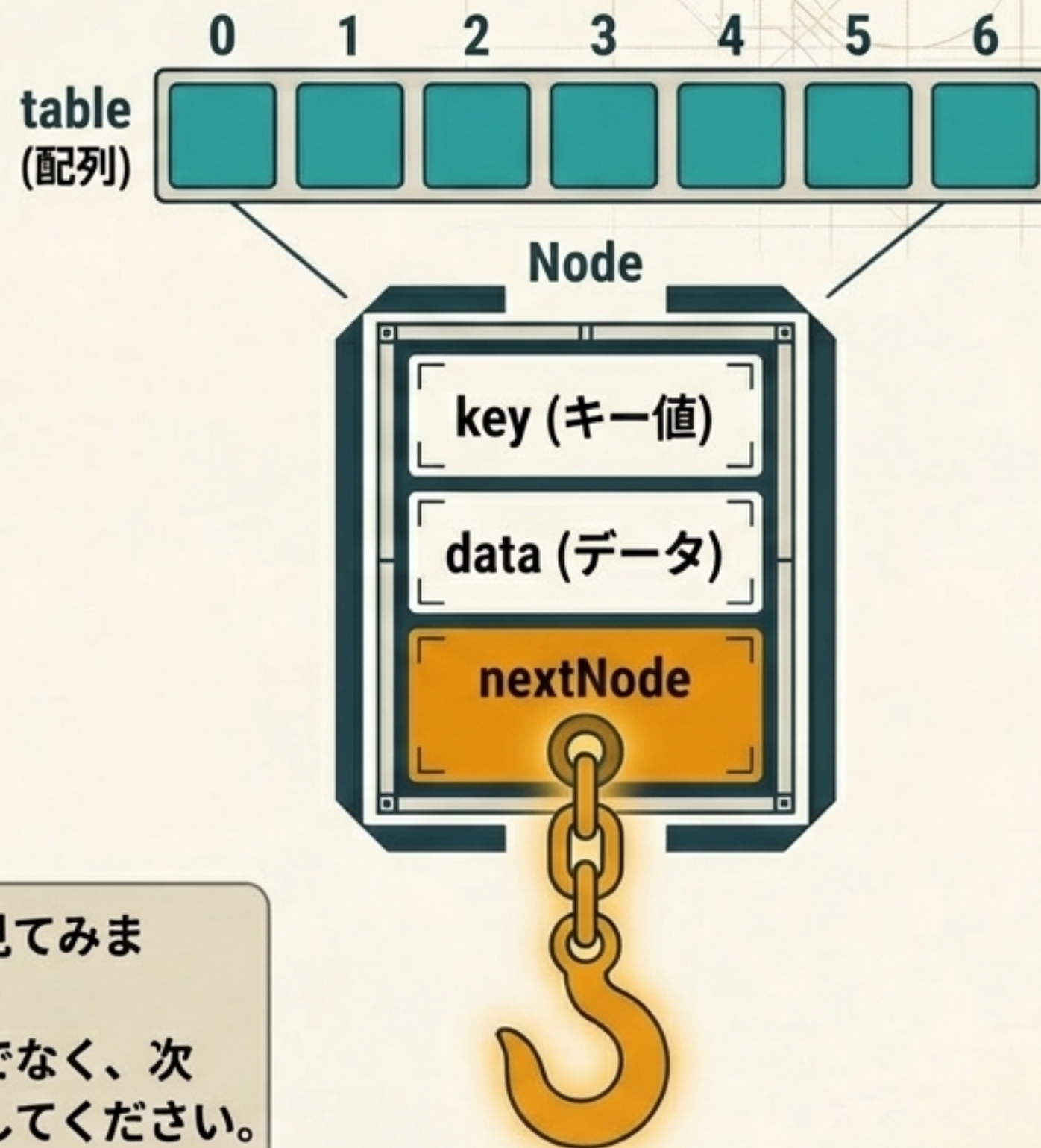
【台本】 まずは基本の確認です。データを効率よくしまうために「ハッシュ関数」という魔法の箱を使います。これによりデータを入れる「配列の添字 (メールボックスの番号)」が決まります。しかし、違うデータなのに同じ番号が出てしまうことがあります。これが衝突です。その時のバックアッププランとして、データを数珠つなぎにするのが「チェーン法」です。

データ構造の解剖図 (Anatomy)

nextNode = 次のデータへ繋ぐための「フック」

名称	種類	内容
Node	構造体	key: キー値 data: データ nextNode: 後続ノードへの参照
table	配列	ノードへの参照を格納する ハッシュ表。
hashValue(key)	関数	キー値を引数とし ハッシュ値を返す。

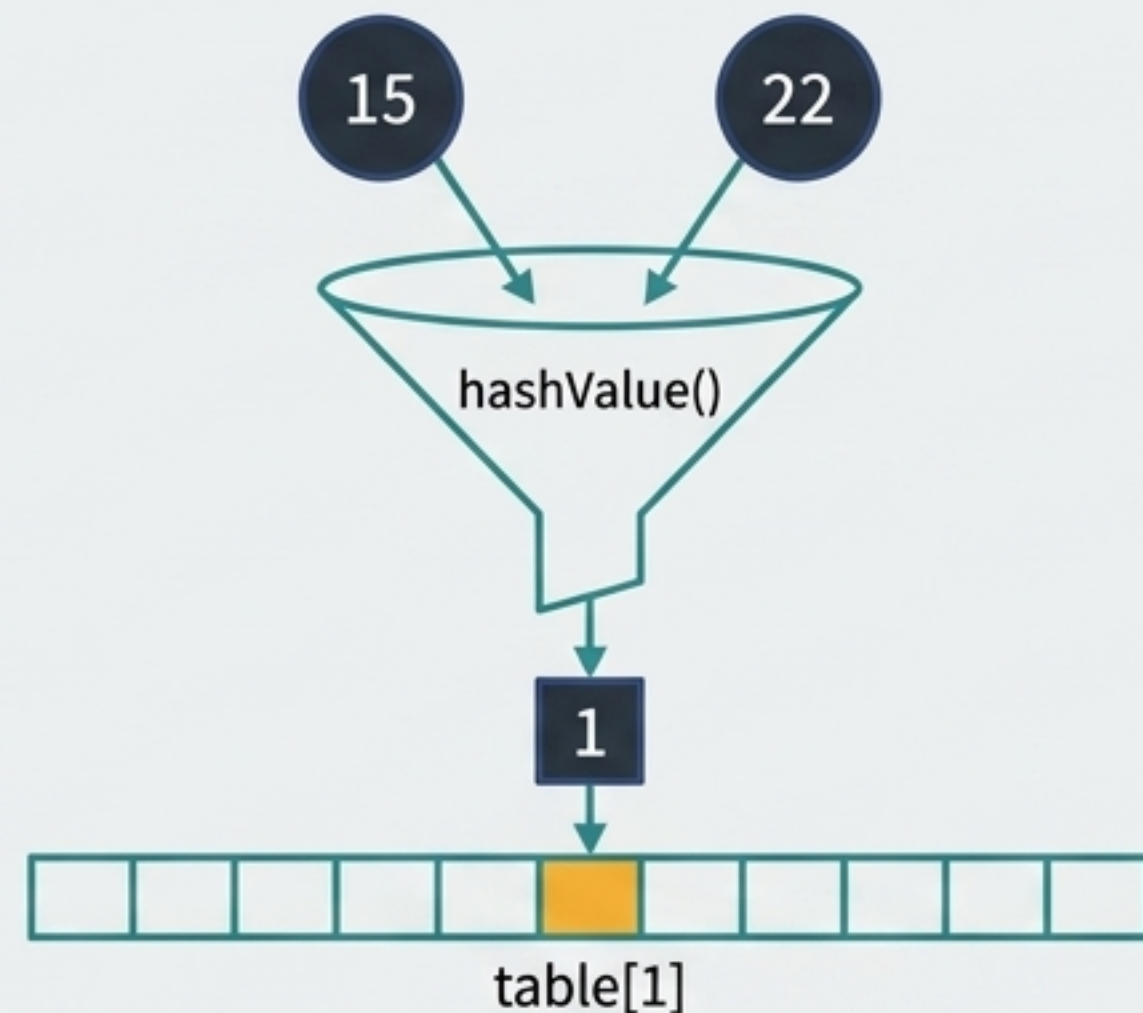
【台本】問題に入る前に、今回使う道具（データ構造）の設計図を見てみましょう。tableはデータの入り口となる配列。そしてNodeが実際のデータの箱です。Nodeにはキーとデータだけでなく、次の箱へ繋がるフックである「nextNode」が含まれていることに注目してください。



Q1: 衝突 (Collision) の定義

設問1: 衝突とはどのような現象か。

条件：“キー”と“ハッシュ関数”という単語を用いる（35字以内）



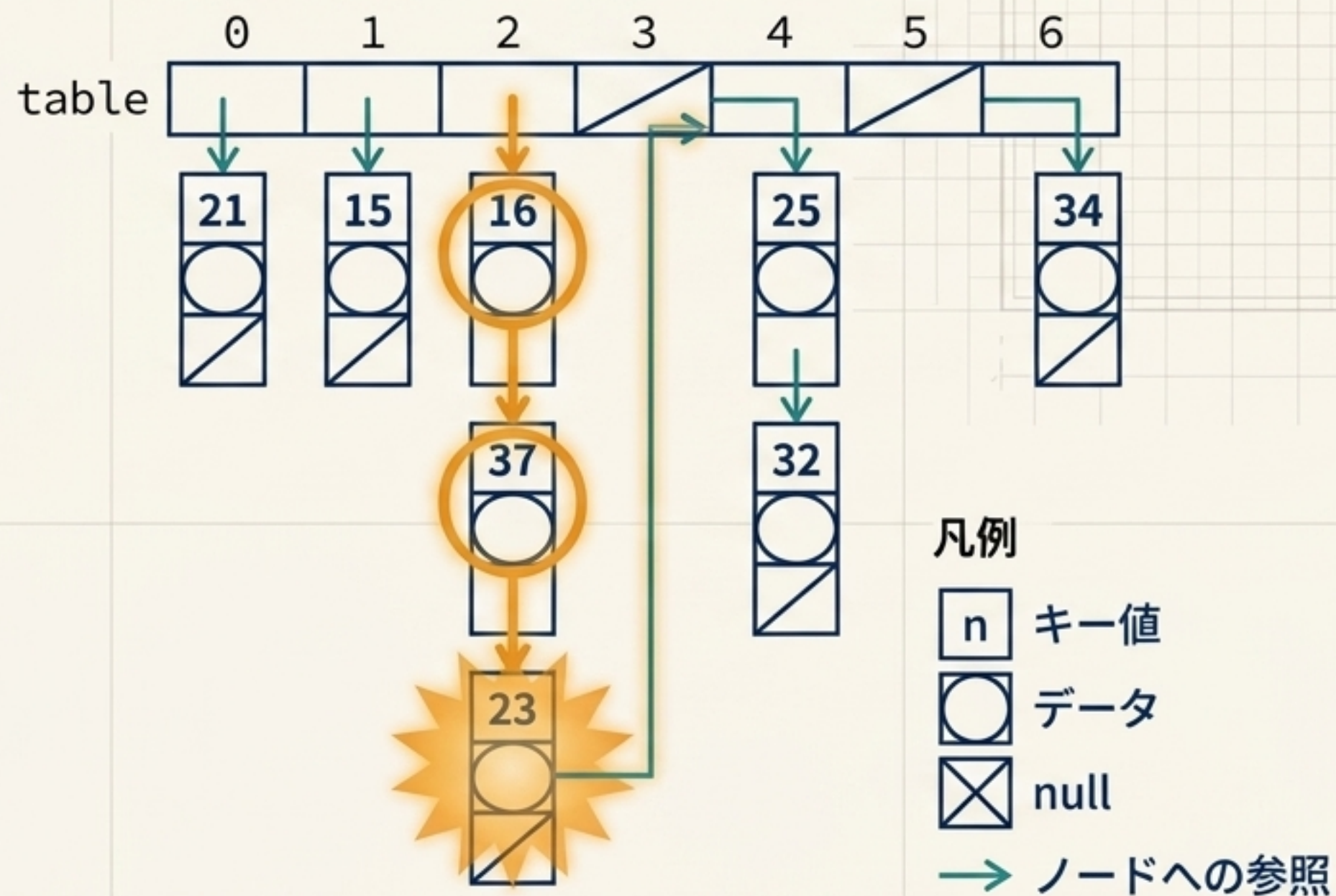
💡 ヒント

異なる「〇〇」が、「〇〇〇〇〇〇〇〇」によって同じ「ハッシュ値」になってしまうこと。

【台本】設問1は衝突の定義です。右の図を見てください。15と22という全く別のキーが、ハッシュ関数を通ると同じ「1」という結果になっていますよね。問題の指定条件である「キー」と「ハッシュ関数」という言葉を使って、この図の状況を文章にしてみましょう。

Q2(1): キー値「23」の探索トレース

- ① スタート地点の計算：
 $23 \div 7 = ?$ 余り？
- ② リストを辿る：
添字[?]の先頭ノードから順に確認する



【台本】設問2の(1)です。プログラムを読む前に、手作業で「23」を探してみましょう。まずはスタート地点の特定です。23を7で割った余りは何でしょうか？それが配列の添字になります。そこからフックを辿って、目的の23が見つかるまでに通過するノードのキー値を順番に書き出せば正解です。

Q2(2): 探索関数 search の解説 [ア]

```
function search(key)
  hash <- hashValue(key);
  node <- table[hash];
  while(node が null でない)
    if( [ ア ] )
      return node.data; // 探索成功
}
```



💡 ヒント

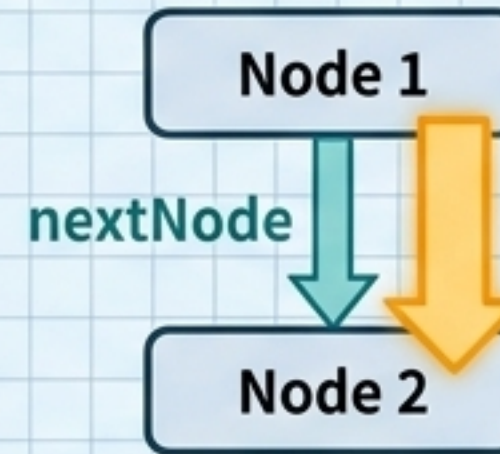
探索が「成功」する条件とは？
今見ているノードのキー (node.key) と、探しているキー (key) がどうなれば良い？

【台本】次はコードの穴埋めです。whileループの中で「探索成功」となりデータを返す条件は何でしょうか。あなたが探しているキーと、今着目している箱（ノード）に書かれたキーがピッタリと一致した時ですよ。それをコードの文法でどう書くか考えてみましょう。

Q2(2): 探索関数 search の解説 [イ]・[ウ]

```
endif  
[ イ ]; // 後続ノードに着目  
endwhile  
return [ ウ ]; // 探索失敗  
endfunction
```

Hint for イ



💡 [イ]のヒント：
ハズレだった場合、現在の着目ノード(node)を、「次のノード」へ進めるには？

Hint for ウ



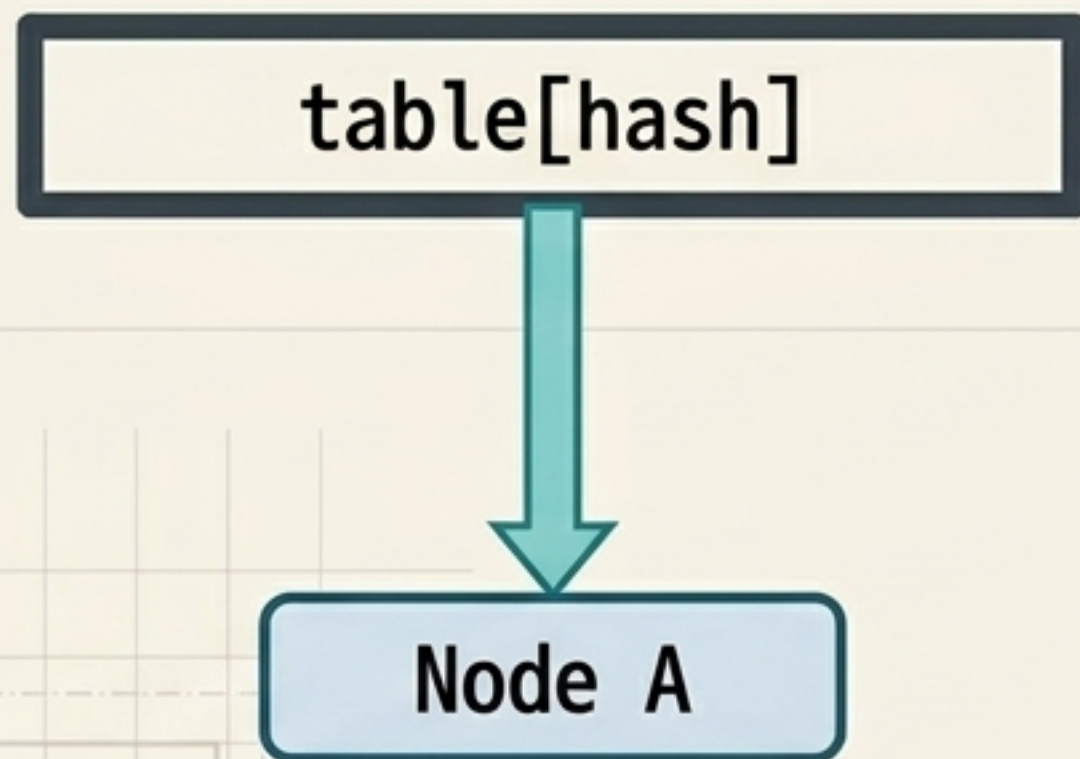
💡 [ウ]のヒント：ループが終わった＝最後まで見つからなかった。失敗の目印として何を返す？

【台本】 [イ]は次のノードへ進む処理です。Node構造体には次の箱への参照が入っている「nextNode」がありましたね。これを使ってnodeの中身を上書きします。[ウ]はリストの最後まで探しても見つからなかった場合です。何もないことを示すプログラミング用語を入れましょう。

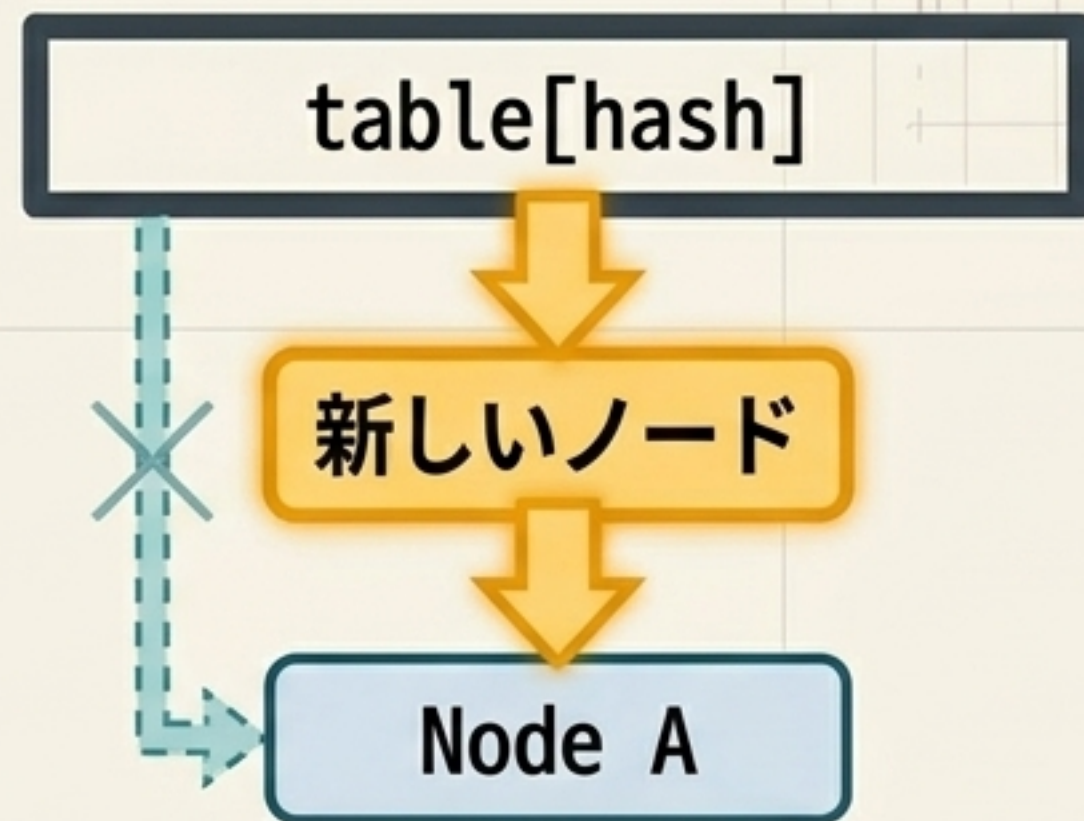
Q3攻略の鍵：新しいノードは「どこ」に追加される？

チェイン法の追加は、効率化のために必ず「リストの先頭」に割り込ませる！

Before



After



【台本】設問3の追加関数に行く前に、物理的な動きを確認します。ここが最大のポイントです！新しいデータをリストに繋ぐとき、わざわざ末尾まで探すのは時間がかかります。一番速いのは、配列のすぐ下、「リストの先頭」に割り込ませることです。この既存の先頭ノードを押し下げる動きをイメージしてください。

Q3: 追加関数 addNode の解説 [エ]・[オ]

```
tempNode <- new Node( [ エ ] ); // ノードを生成  
[ オ ] <- tempNode; // ノードを追加
```

💡 [エ]のヒント：新しいノードの引数 (キー、データ、次のノード)。新しいノードの「次」に来るのは、今まで先頭だったノード(node)であるべきです。

💡 [オ]のヒント：新しい先頭ノード(tempNode)を、ハッシュ表 (配列) の入り口として登録し直そう。

After from previous slide

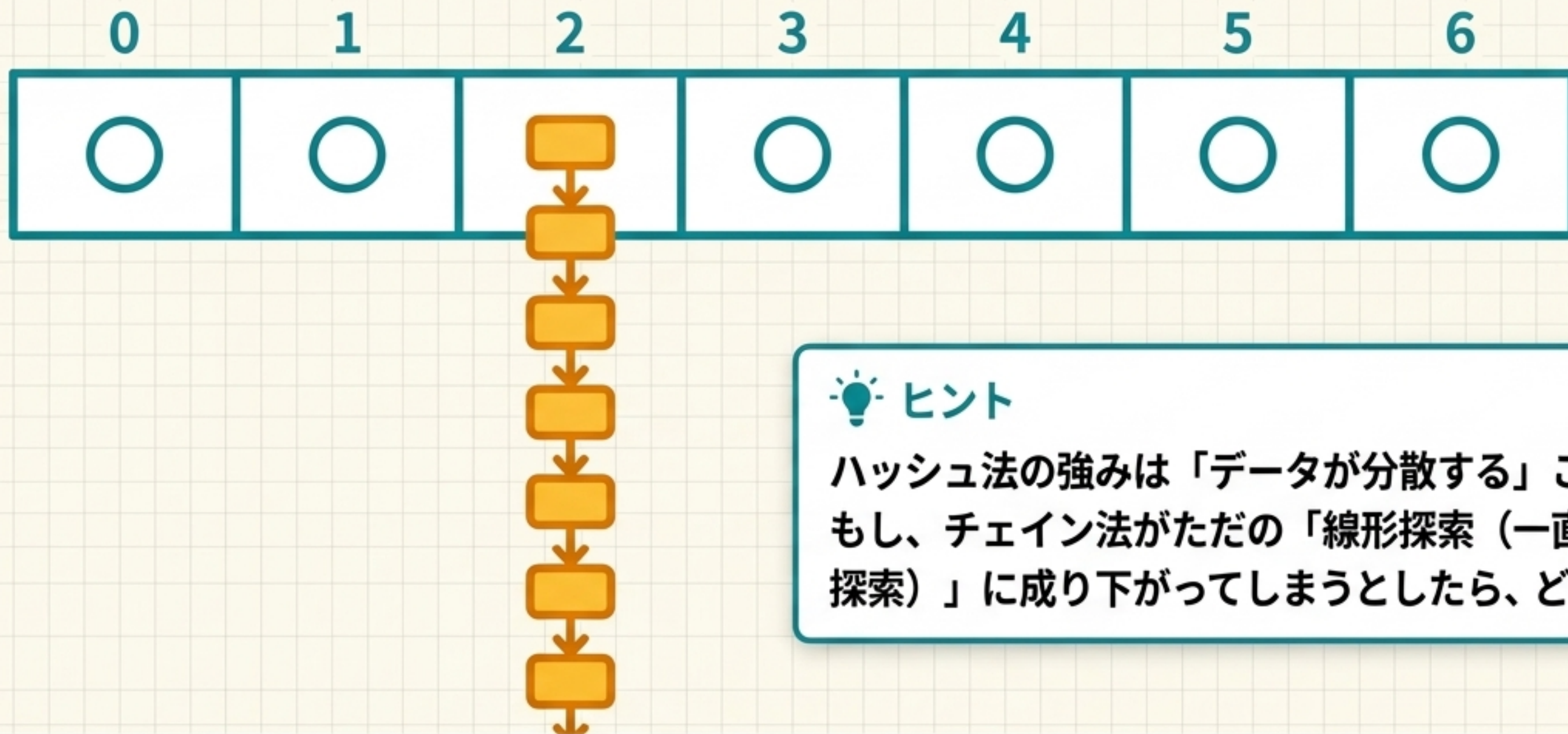
table[hash]

新しいノード

Node A

【台本】先ほどのイメージをコードに当てはめます。[エ]は新しいノードを生成する部分です。引数は3つ。3つ目の「次のノード」には、今まで先頭だったnodeを指定してフックを繋ぎます。そして[オ]で、ハッシュ表の入り口である配列の指し先を、今作ったばかりの tempNode に付け替えます。

Q4(1): ハッシュ法が「最悪」になる瞬間 [カ]



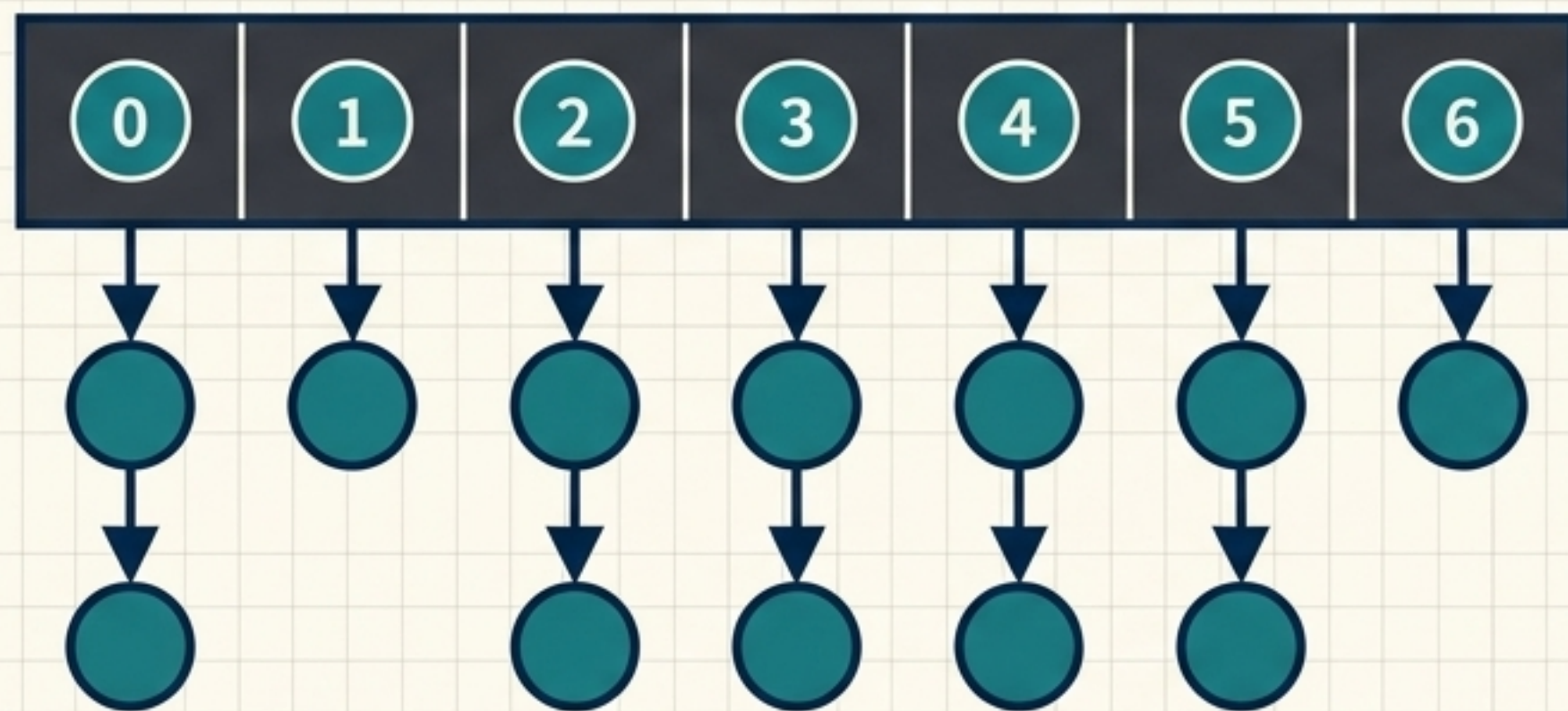
💡 ヒント

ハッシュ法の強みは「データが分散する」こと。
もし、チェイン法がただの「線形探索（一直線の長い探索）」に成り下がってしまったら、どんな状態？

【台本】設問4は計算量、つまりパフォーマンスの問題です。ハッシュ法の最大のメリットはデータが分散することですが、最悪のシナリオは何でしょうか？この図のように、偶然やすべてのデータが同じハッシュ値になってしまい、1つの場所に全データが集中して一直線に並んでしまった場合です。これを25字以内で表現しましょう。

Q4(2): 理想的な状態での計算量 [キ]

データの数(n)が増えても、配列の長さ(m)を十分に大きくすれば、1つのリストの長さ(n/m)は平均して「ほぼ一定」になる。



💡 ヒント

データ量に関わらず、アクセスするノード数が「一定時間」で済む場合のオーダ(O)記法は？

【台本】逆に、配列の長さが十分にあり、関数が優秀であれば、1つのリストにぶら下がるデータの数ほぼ1個か2個で一定になります。つまり、データが1万個に増えても、探す手間は変わりません。データ数に依存せず「一定時間」で終わる場合の O 記法を教えてください。

なぜ「チェイン法」を使うのか？（アルゴリズムの真価）

配列の「検索スピード」と、リストの「追加のしやすさ」のいいとこ取り。
究極のハイブリッド構造。

データ構造	探索	追加/削除
配列	速い (Index指定)	遅い (データがズレる)
線形リスト	遅い ($O(n)$)	速い (フックの付替)
ハッシュ+チェイン法	速い ($O(1)$)	速い (フックの付替)

【台本】最後に、なぜこのアルゴリズムが試験に出るのかを整理しましょう。配列は検索が速いですが追加が苦手。線形リストは追加が速いですが検索が苦手。ハッシュ+チェイン法は、この2つの「いいとこ取り」をした究極のハイブリッド構造だからです。これが、このデータ構造の真価です。

本日のクリア条件

- ✓ 衝突（Collision）の原理を理解した
- ✓ 探索処理のノードの動きをトレースできた
- ✓ 「先頭への追加」のポインタ付け替えをイメージできた
- ✓ チェイン法の計算量とメリットを把握した

コードに迷ったら、頭の中に「箱」と「フック」の図を描こう。

【台本】お疲れ様でした。これで初心者がつまずきやすいチェイン法の過去問を完全攻略しました。ポインタや参照といった難しい言葉が出てきても、今日見た「箱」と「フック」の物理的な動きを頭に思い浮かべれば必ず解けます。本番の試験も応援しています！