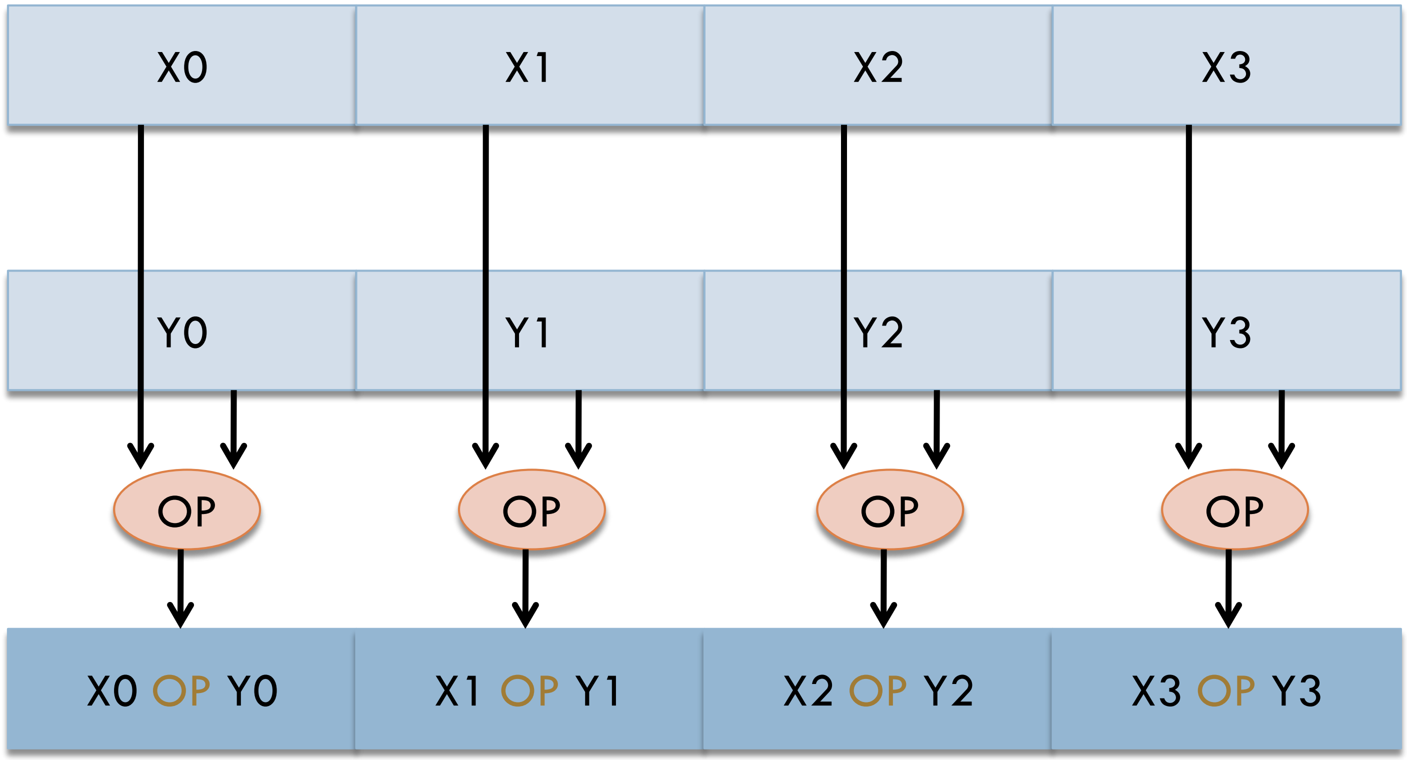
Implementing Database Operations Using SIMD Instructions
By: Jingren Zhou, Kenneth A. Ross

Presented by: Ioan Stefanovici

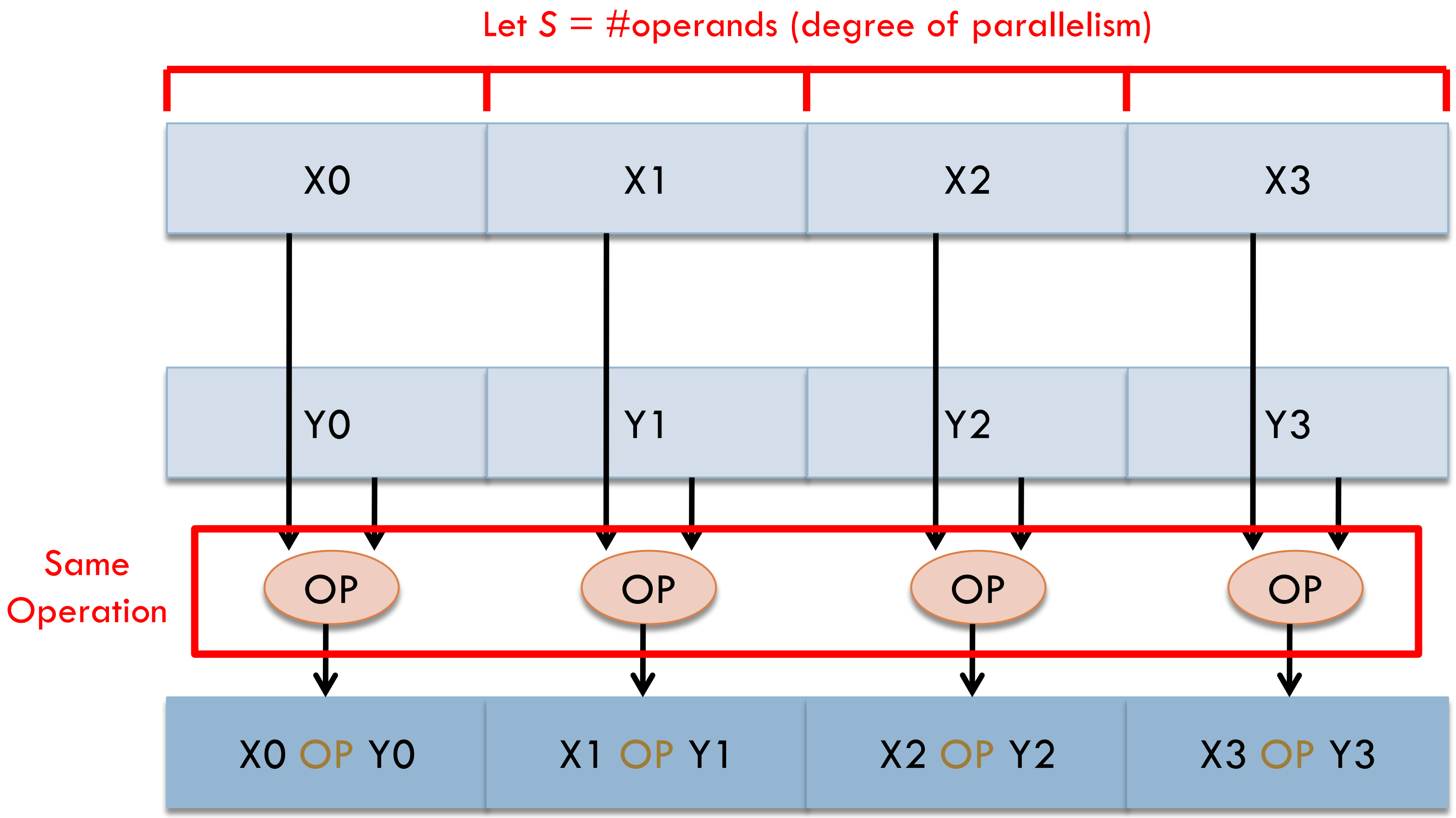
The Problem

- Databases have become bottlenecked on CPU and memory performance
- Need to fully utilize available architectures' features to maximize performance
  - Cache performance
    - e.g.: cache-conscious B^+ trees, PAX, etc.
  - Proposal: use SIMD instructions

Single-Instruction, Multiple-Data (SIMD)

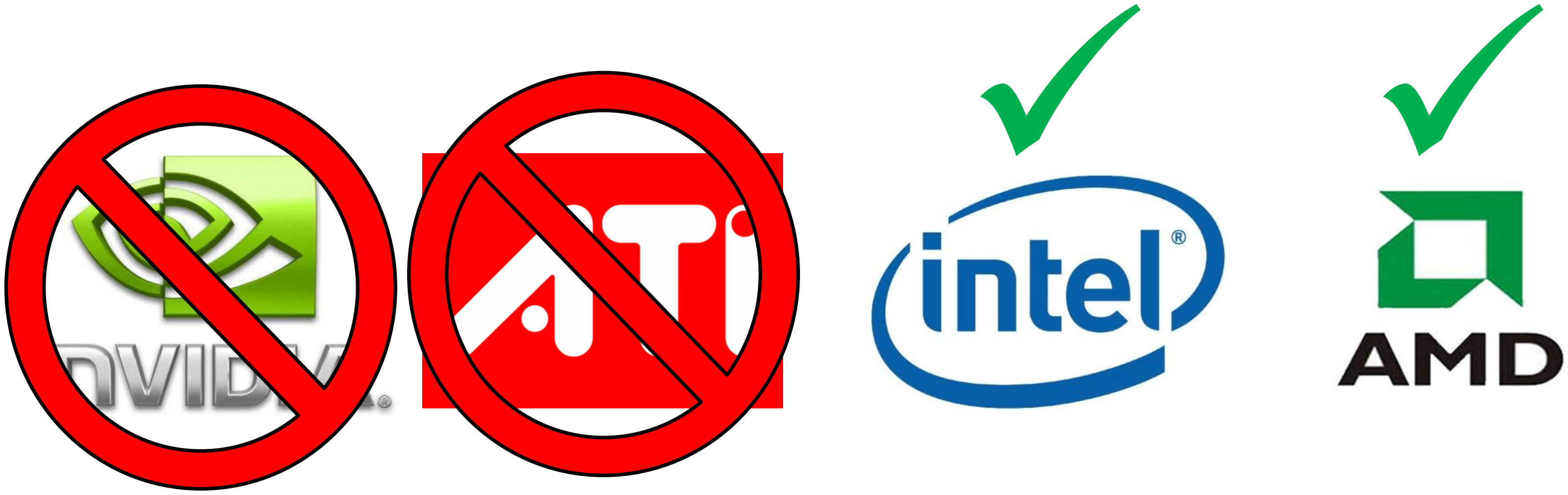


Single-Instruction, Multiple-Data (SIMD)



Single-Instruction, Multiple-Data (SIMD)

- Focus



- Goal
  - Achieve speed-ups close to (or higher!) than *S* (the degree of parallelization)

Outline

- Motivation & Problem Statement
- SIMD Instructions and Implementation Details
- Algorithm Improvements:
  - Scan algorithms
  - Index traversals
  - Join algorithms

A few points…

- Compiler auto-parallelization is difficult

  → Explicit use of SIMD instructions

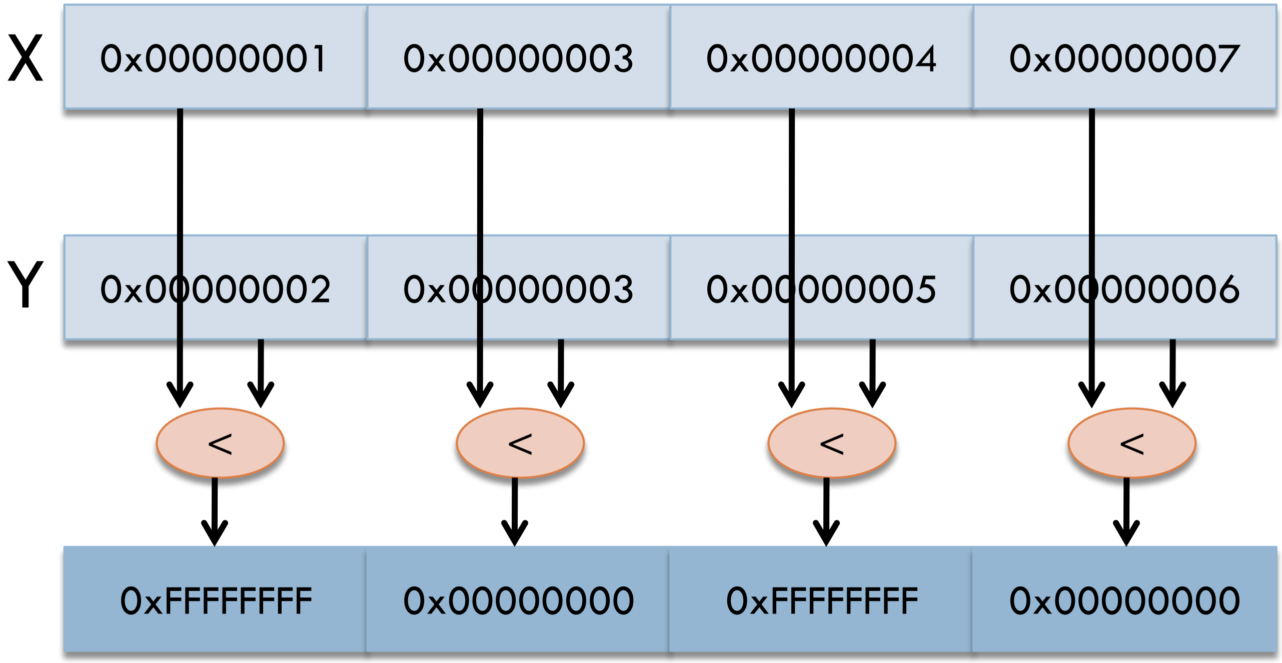
- SIMD data alignment

  → Column-oriented storage

- Targets
  - Scan-like operations
  - Index traversals
  - Join algorithms

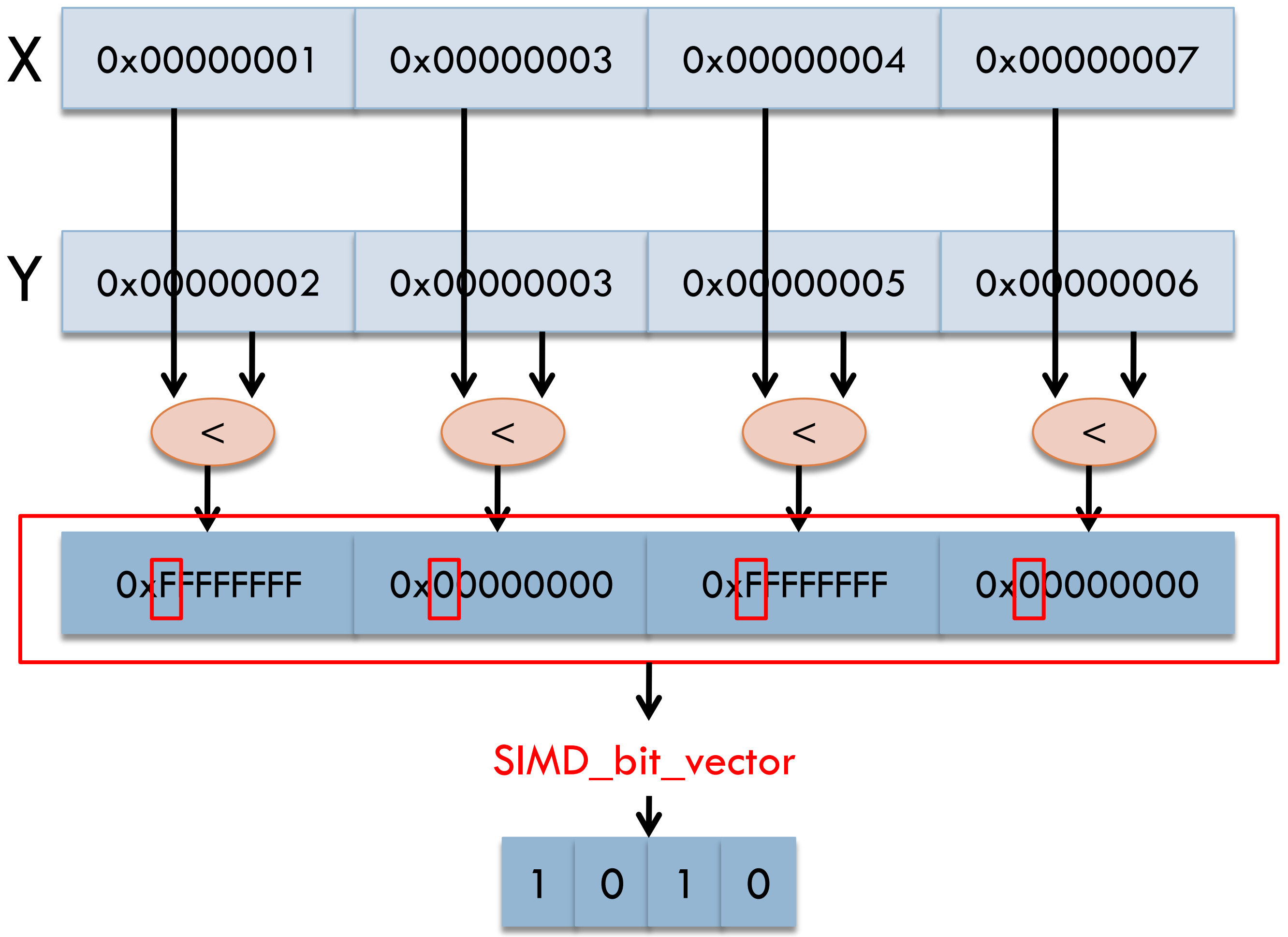
Comparison Result Example

- Want to perform: X < Y



Comparison Result Example

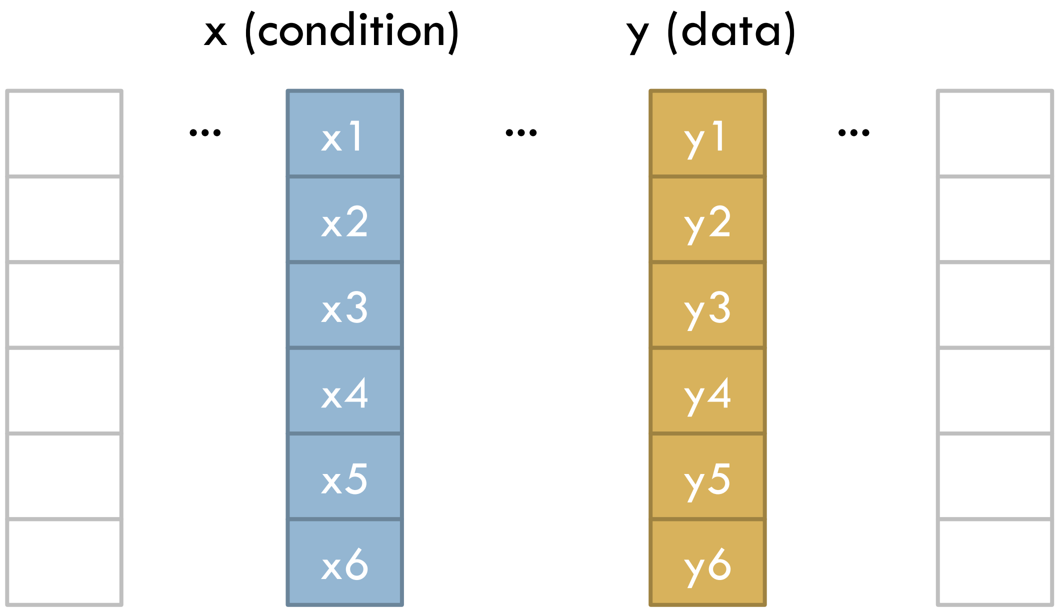
- Want to perform: X < Y



Scan

- Typical scan:

```
for i = 1 to N{
   if (condition(x[i])) then
      process1(y[i]);
   else
      process2(y[i]);
}
```

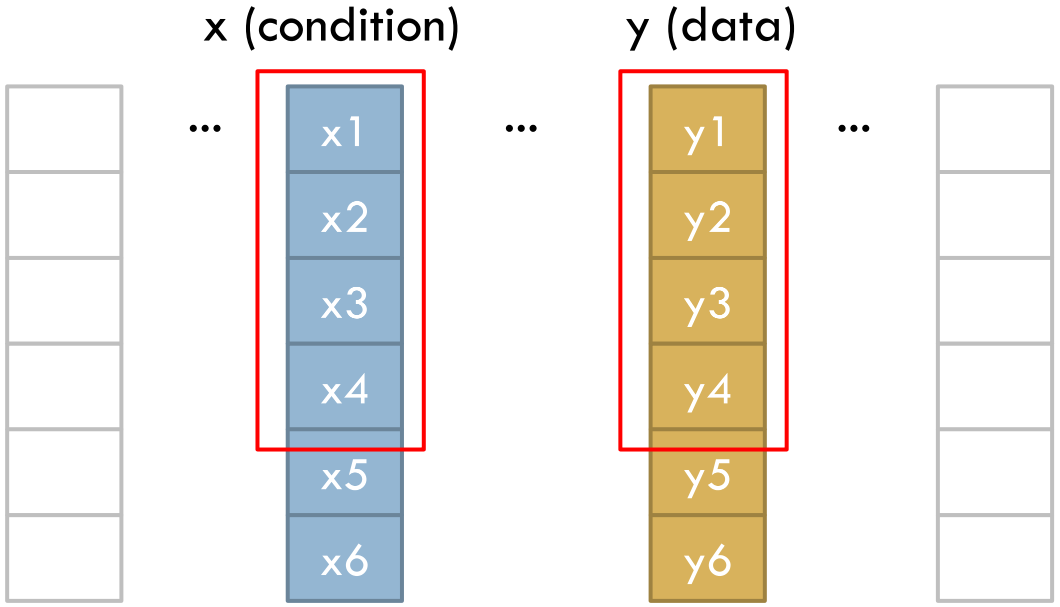


SIMD Scan

- Typical SIMD scan:

```
for i = 1 to N step S {
   Mask[1..S] = SIMD_condition(x[i..i+S-1]);
   SIMD_Process(Mask[1..S], y[i..i+S-1]);
}
```

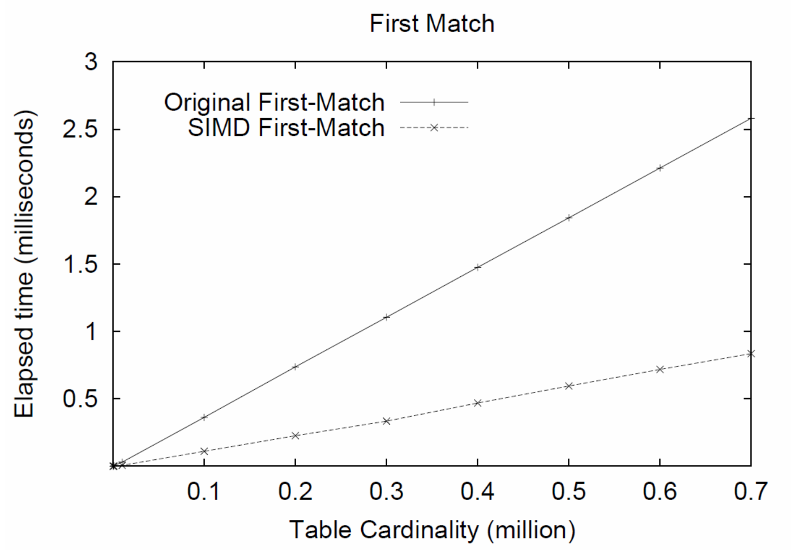
For S=4



Scan: Return First Match

- SIMD Return First Match

```
SIMD_Process(mask[1..S], y[1..S]){
   V = SIMD_bit_vector(mask);
   /* V = number between 0 and 2^S-1 */
   if (V != 0){
      for j = 1 to S
         if ( (V >> (S-j)) & 1 ) /* jth bit */
            { result = y[j]; return; }}
}
```



Scan: Return All Matches

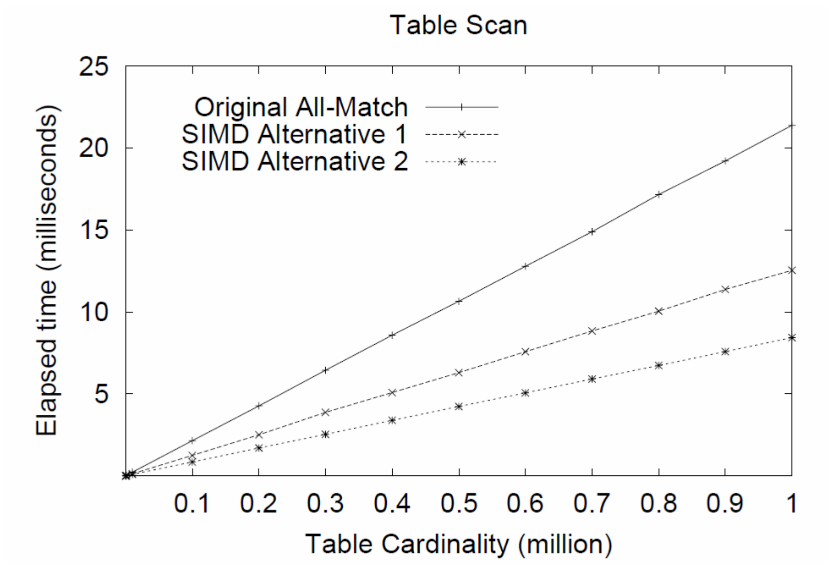
- SIMD All Matches Alternative 1

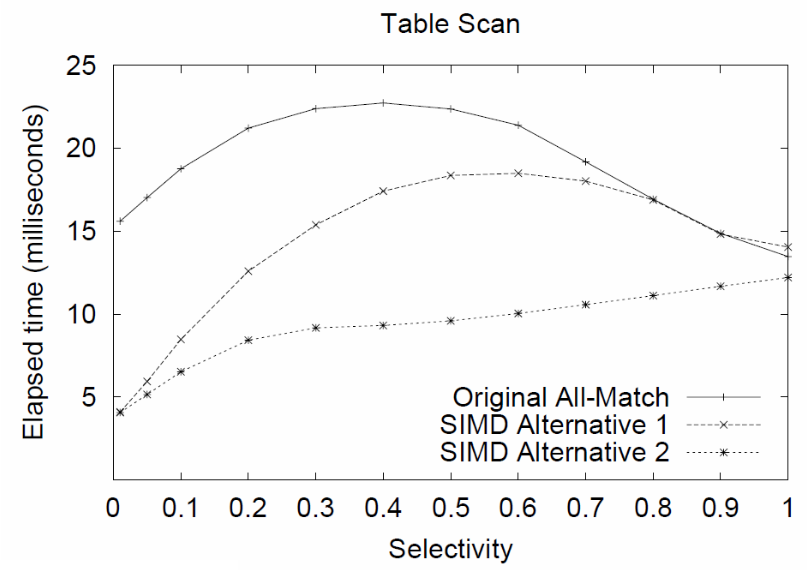
```
SIMD_Process(mask[1..S], y[1..S]){
   V = SIMD_bit_vector(mask);
   /* V = number between 0 and 2^S-1 */
   if (V != 0){
      for j = 1 to S
         if ( (V >> (S-j)) & 1 ) /* jth bit */
            { result[pos++] = y[j]; }
}
```

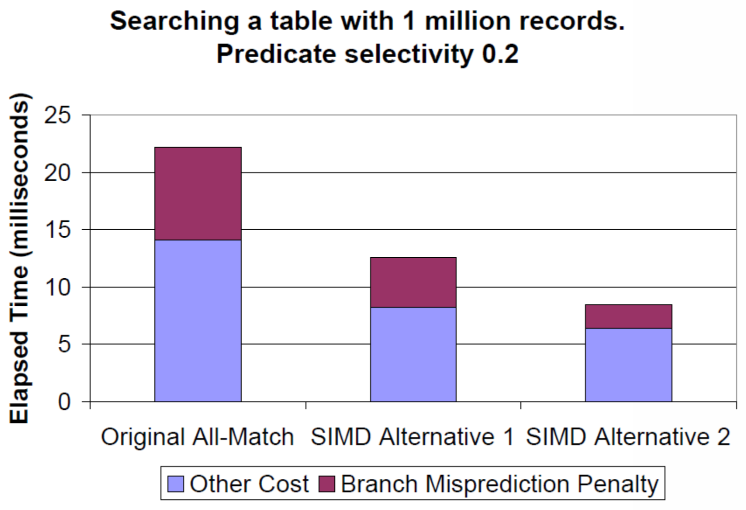
- SIMD All Matches Alternative 2

```
SIMD_Process(mask[1..S], y[1..S]){
   V = SIMD_bit_vector(mask);
   /* V = number between 0 and 2^S-1 */
   if (V != 0){
      for j = 1 to S
         tmp = (V >> (S-j)) & 1 /* jth bit */
         result[pos] = y[j];
         pos += tmp; } }
}
```

Scan: Return All Matches Performance

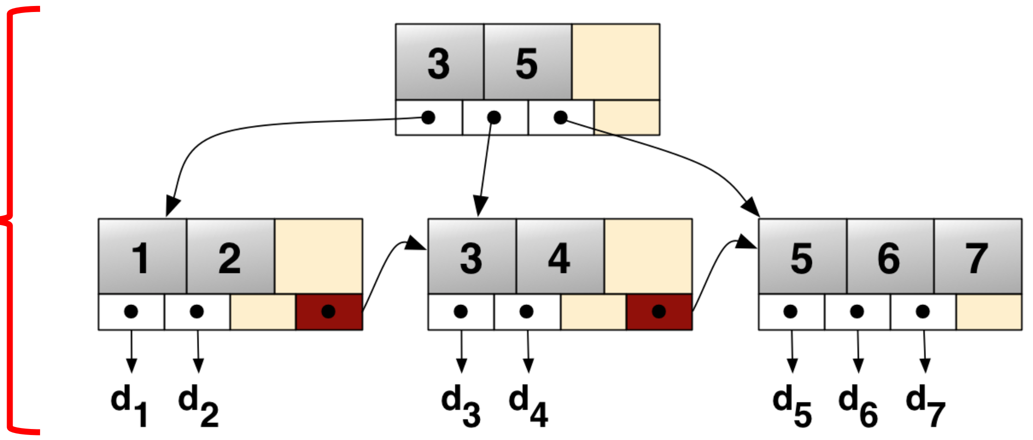




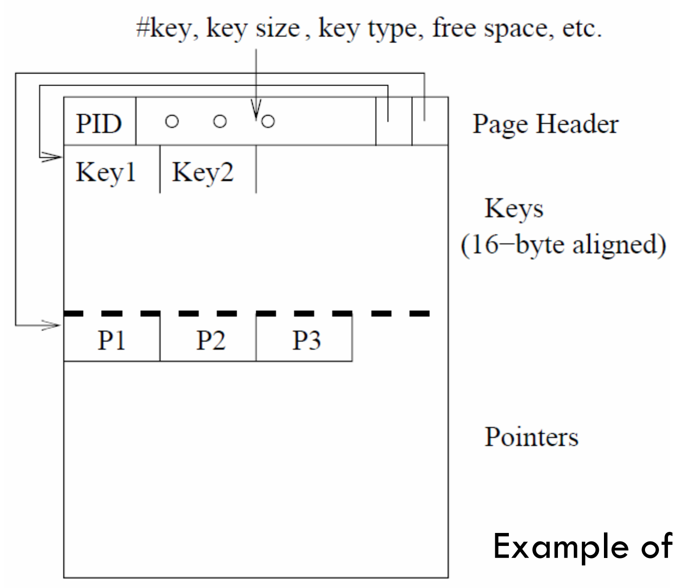


Index Structures (B^+ trees)

$\log_2 (n)$
Height



(Source: Wikipedia)



Example of a B+ -tree internal node

Internal Node Search

- 5 Ways to Search
  - Binary Search (SISD)
  - SIMD Binary Search
  - SIMD Sequential Search 1
  - SIMD Sequential Search 2
  - Hybrid Search

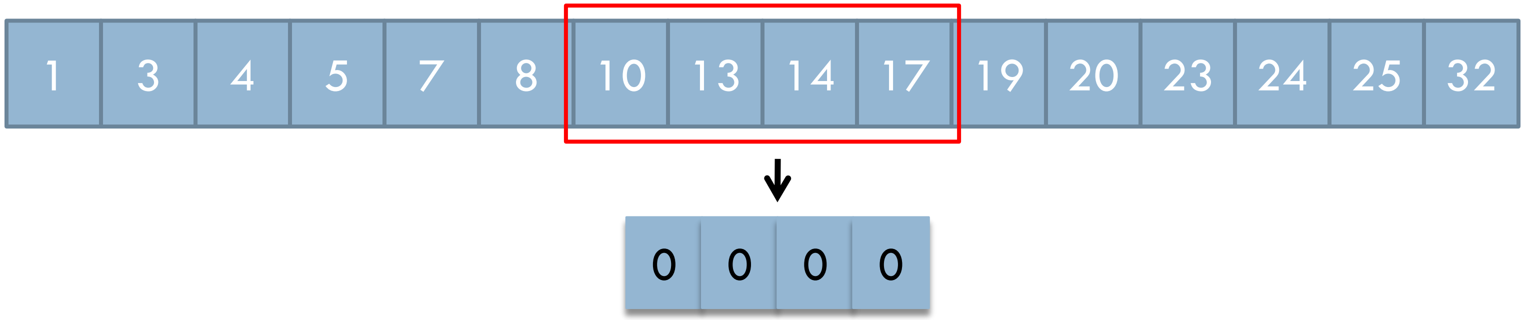
Internal Node Search

- Naive SIMD Binary Search (looking for “4”)

1	3	4	5	7	8	10	13	14	17	19	20	23	24	25	32

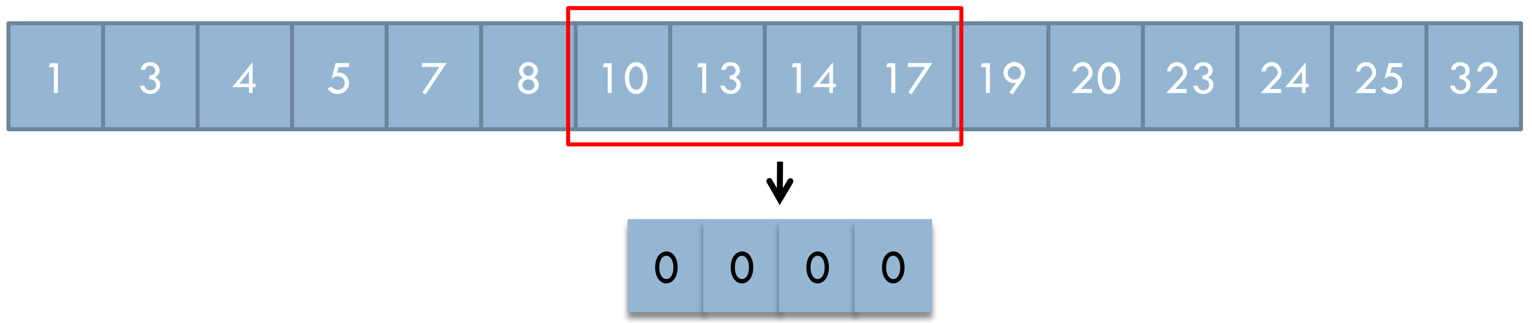
Internal Node Search

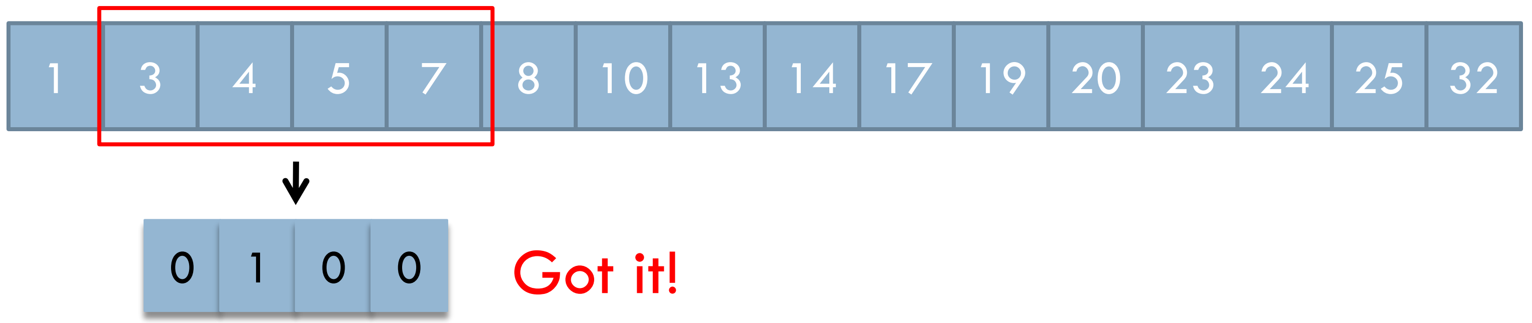
- Naive SIMD Binary Search (looking for "4")



Internal Node Search

- Naive SIMD Binary Search (looking for “4”)





Got it!

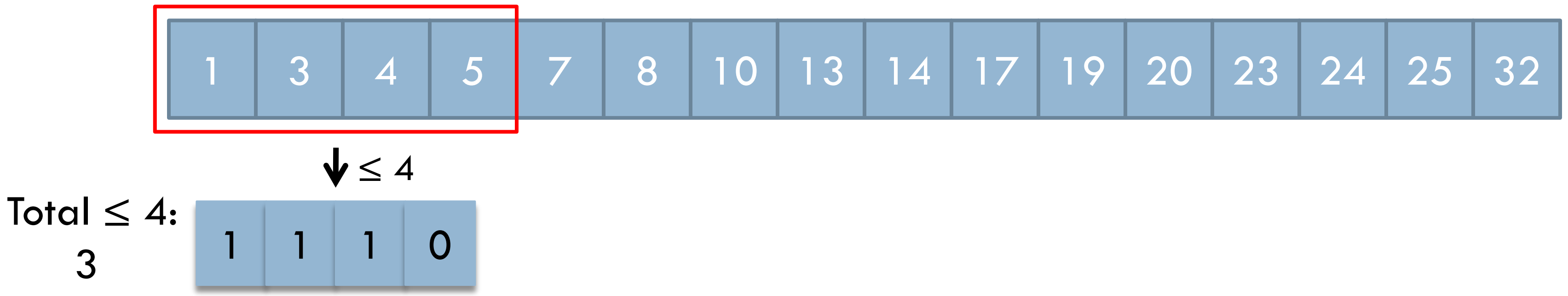
Internal Node Search

- SIMD Sequential Search 1 (looking for "4")

1	3	4	5	7	8	10	13	14	17	19	20	23	24	25	32

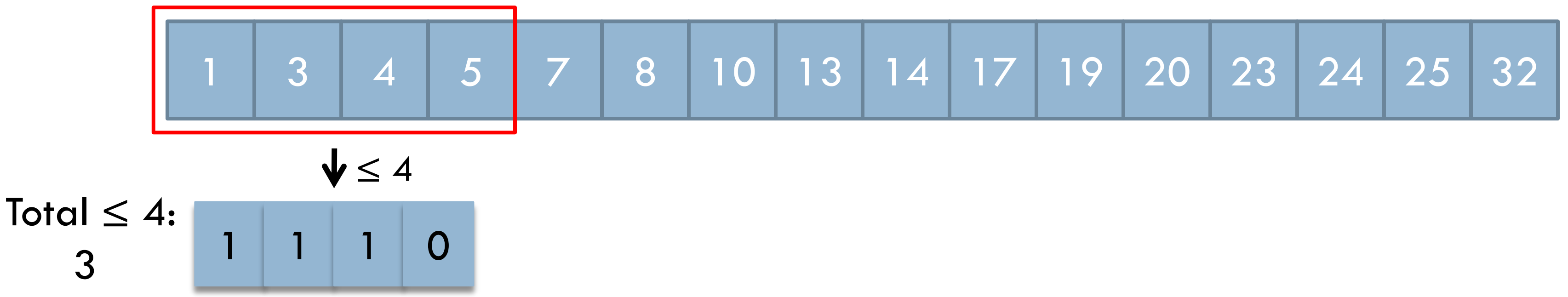
Internal Node Search

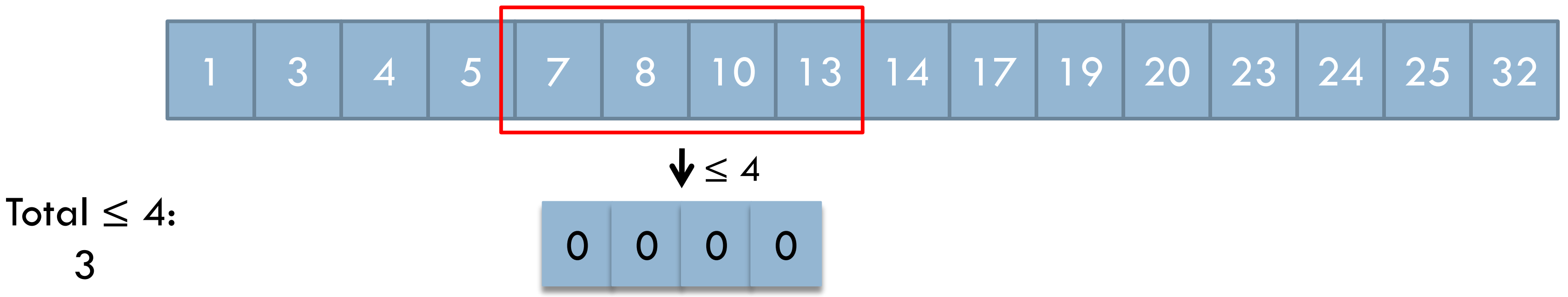
- SIMD Sequential Search 1 (looking for "4")



Internal Node Search

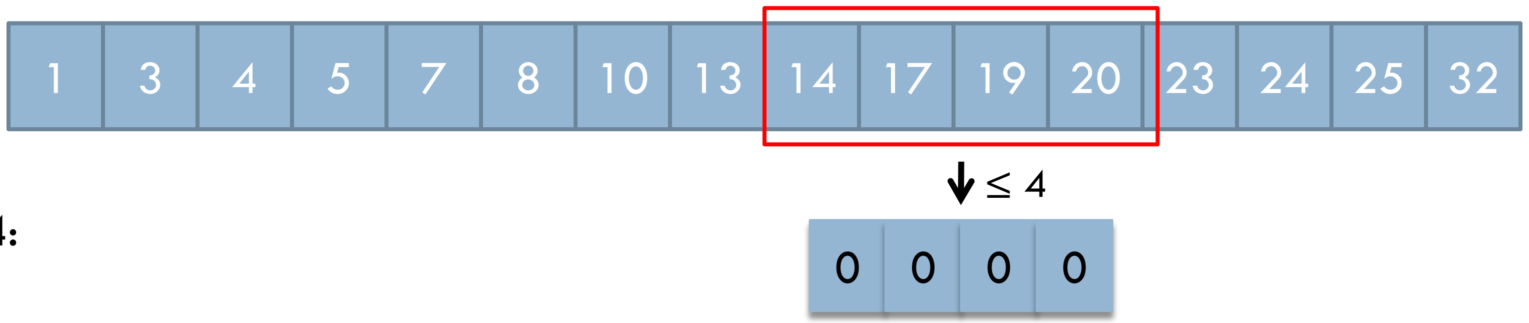
- SIMD Sequential Search 1 (looking for "4")





Internal Node Search

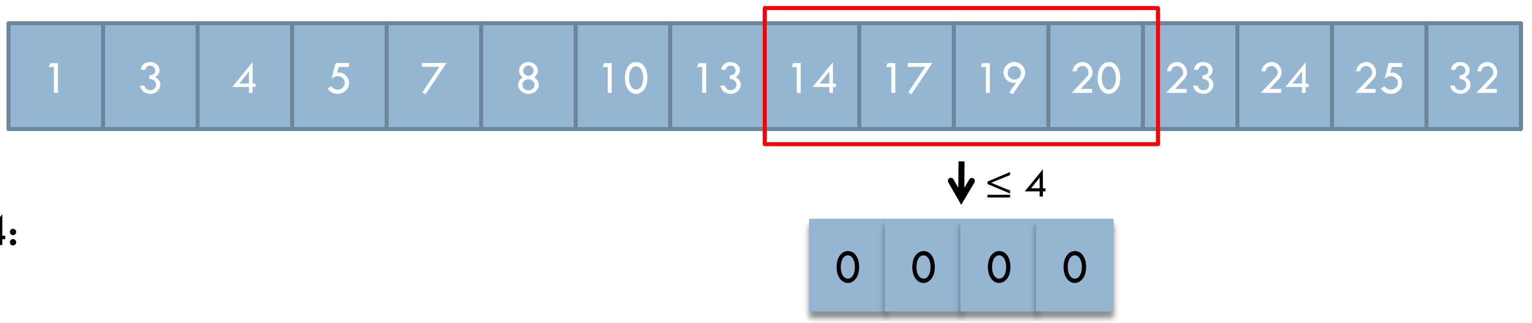
- SIMD Sequential Search 1 (looking for "4")



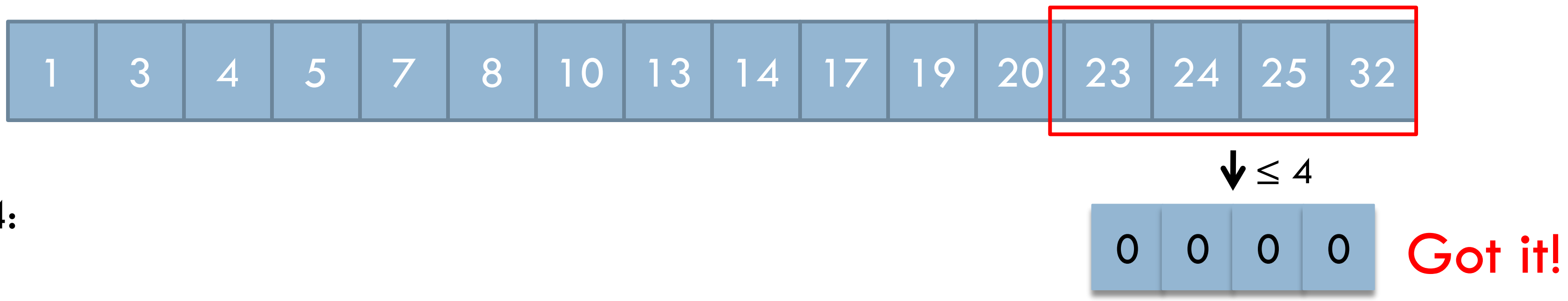
Total ≤ 4:
3

Internal Node Search

- SIMD Sequential Search 1 (looking for "4")



Total ≤ 4:
3



Total ≤ 4:
3

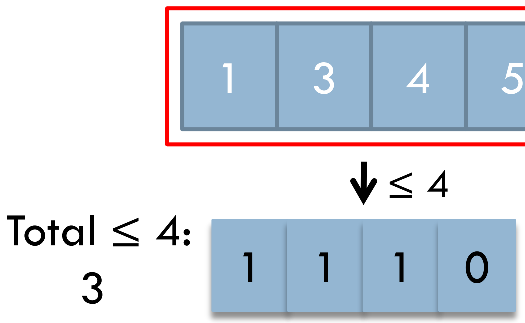
Internal Node Search

- SIMD Sequential Search 2 (looking for “4”)

1	3	4	5	7	8	10	13	14	17	19	20	23	24	25	32

Internal Node Search

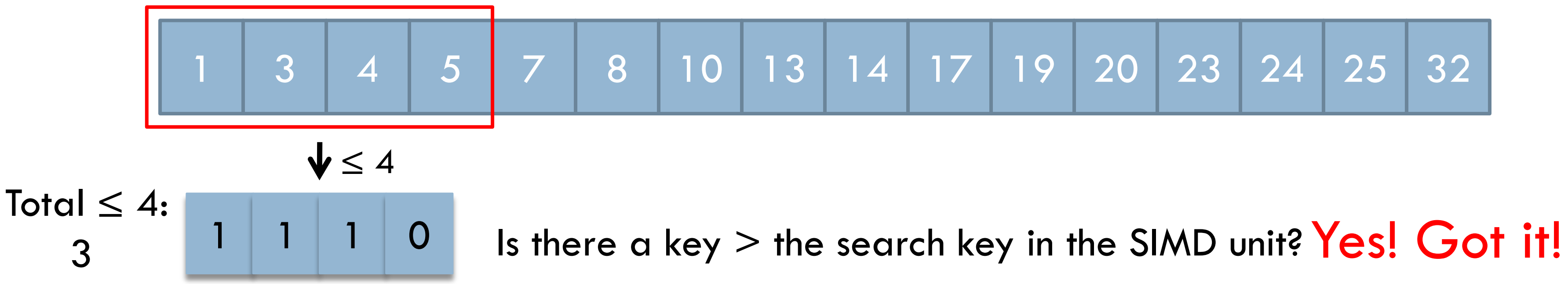
- SIMD Sequential Search 2 (looking for "4")



Is there a key > the search key in the SIMD unit? Yes! Got it!

Internal Node Search

- SIMD Sequential Search 2 (looking for “4”)



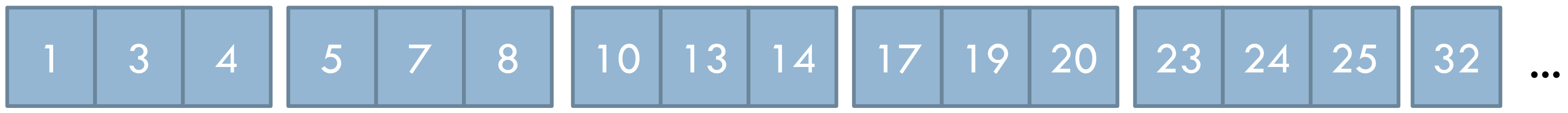
Is there a key > the search key in the SIMD unit? Yes! Got it!

- Pro: processes fewer keys (50% fewer on average)
- Con: extra conditional test

Internal Node Search

- Hybrid Search (looking for "4")

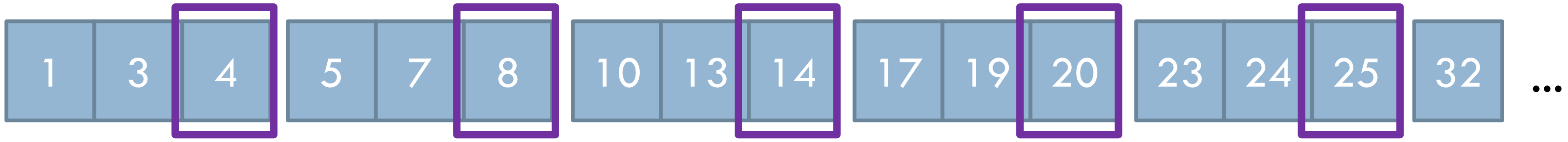
Pick some L (say L = 3)



Internal Node Search

- Hybrid Search (looking for "4")

Pick some L (say L = 3)

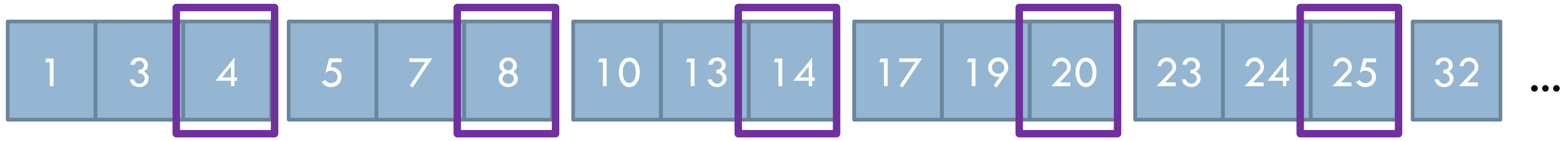


Binary Search on last element of each "segment"

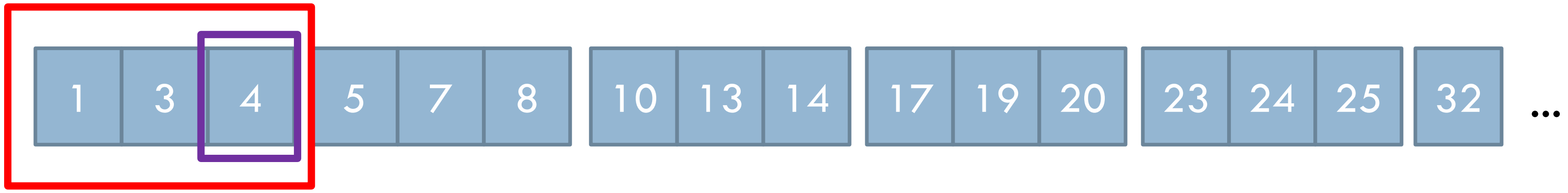
Internal Node Search

- Hybrid Search (looking for "4")

Pick some L (say L = 3)

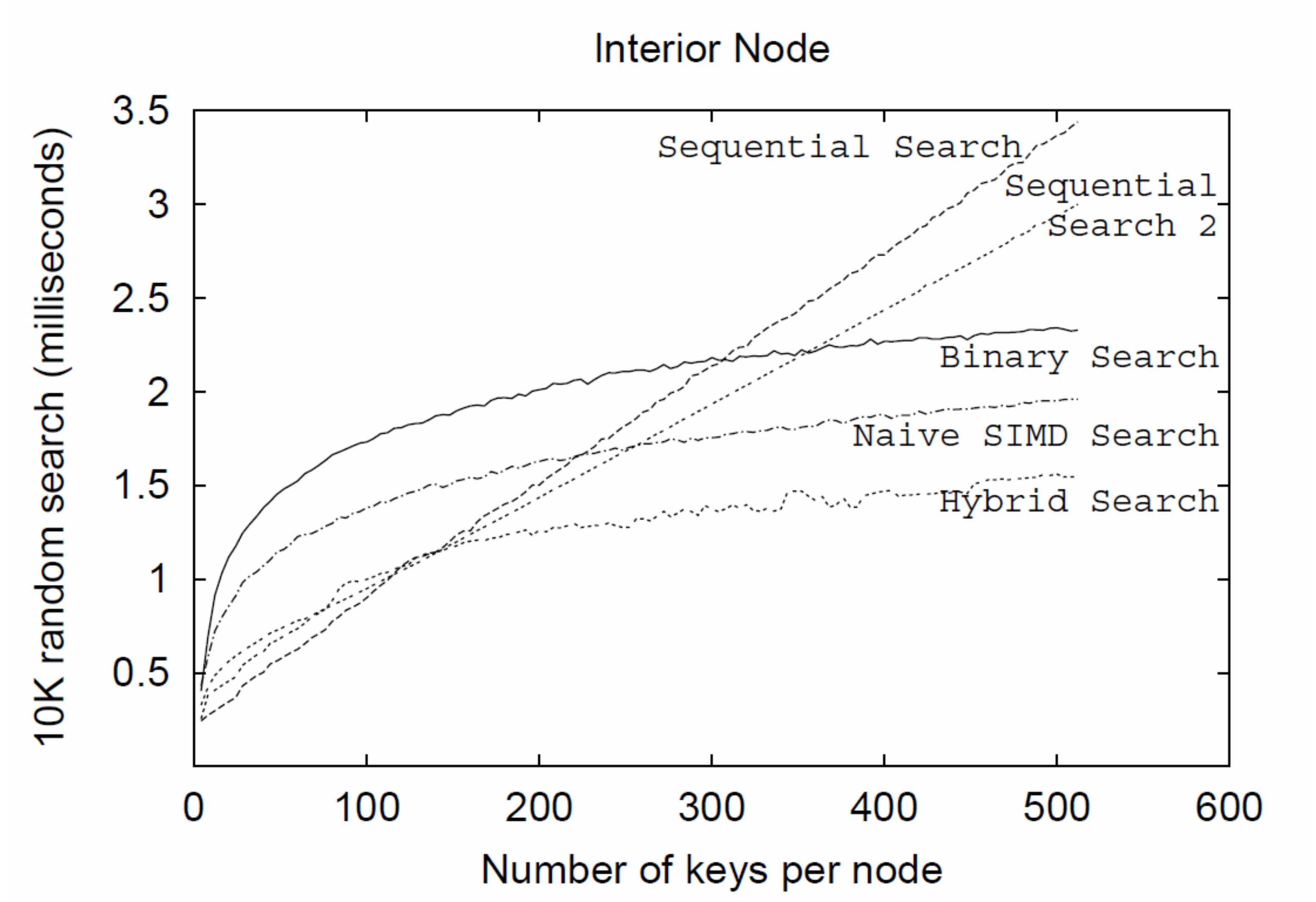


Binary Search on last element of each "segment"

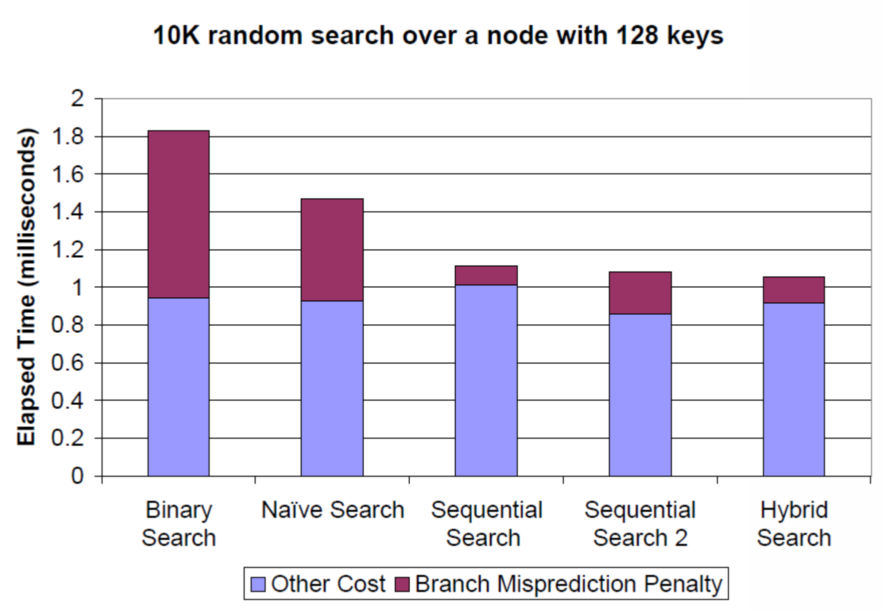


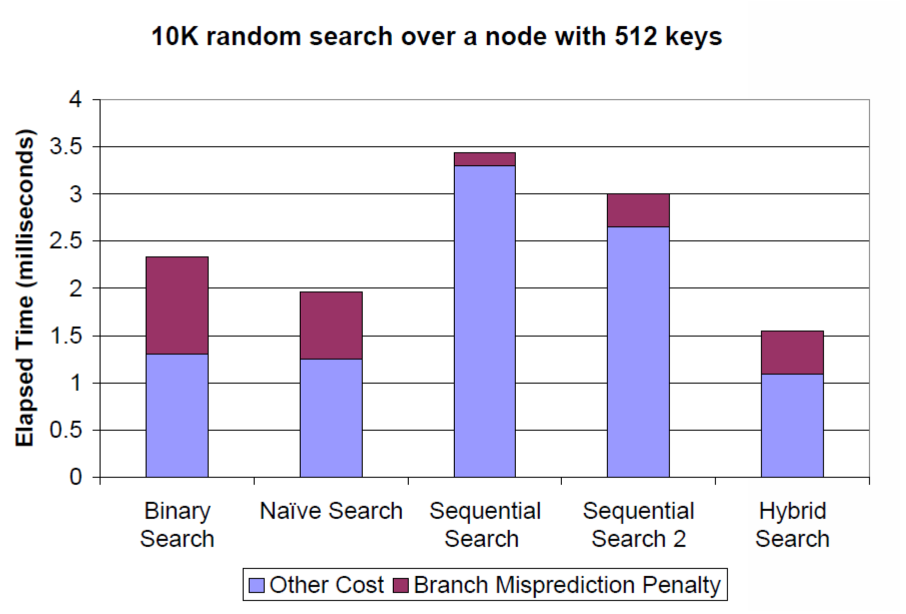
Sequential SIMD scan inside the correct segment

Internal Node Search Performance



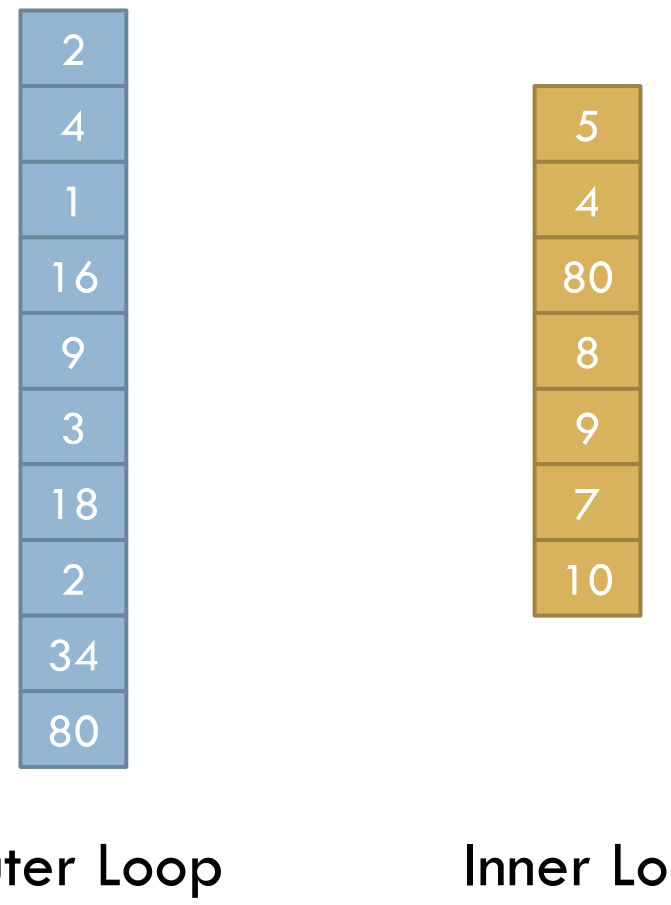
Internal Node Search – Branch Misprediction





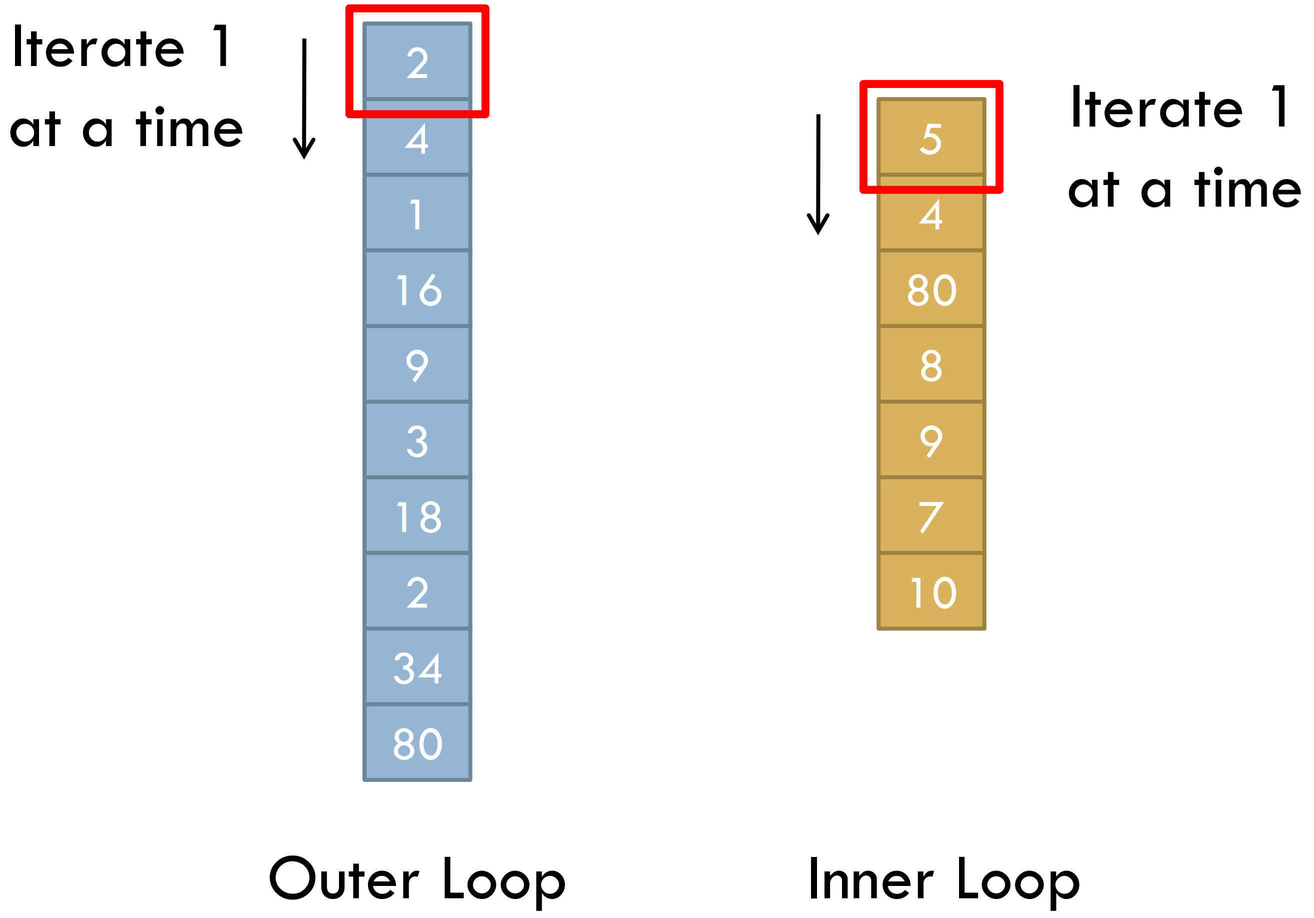
Nested Loop Join – O(n^2)

- Nested Loop



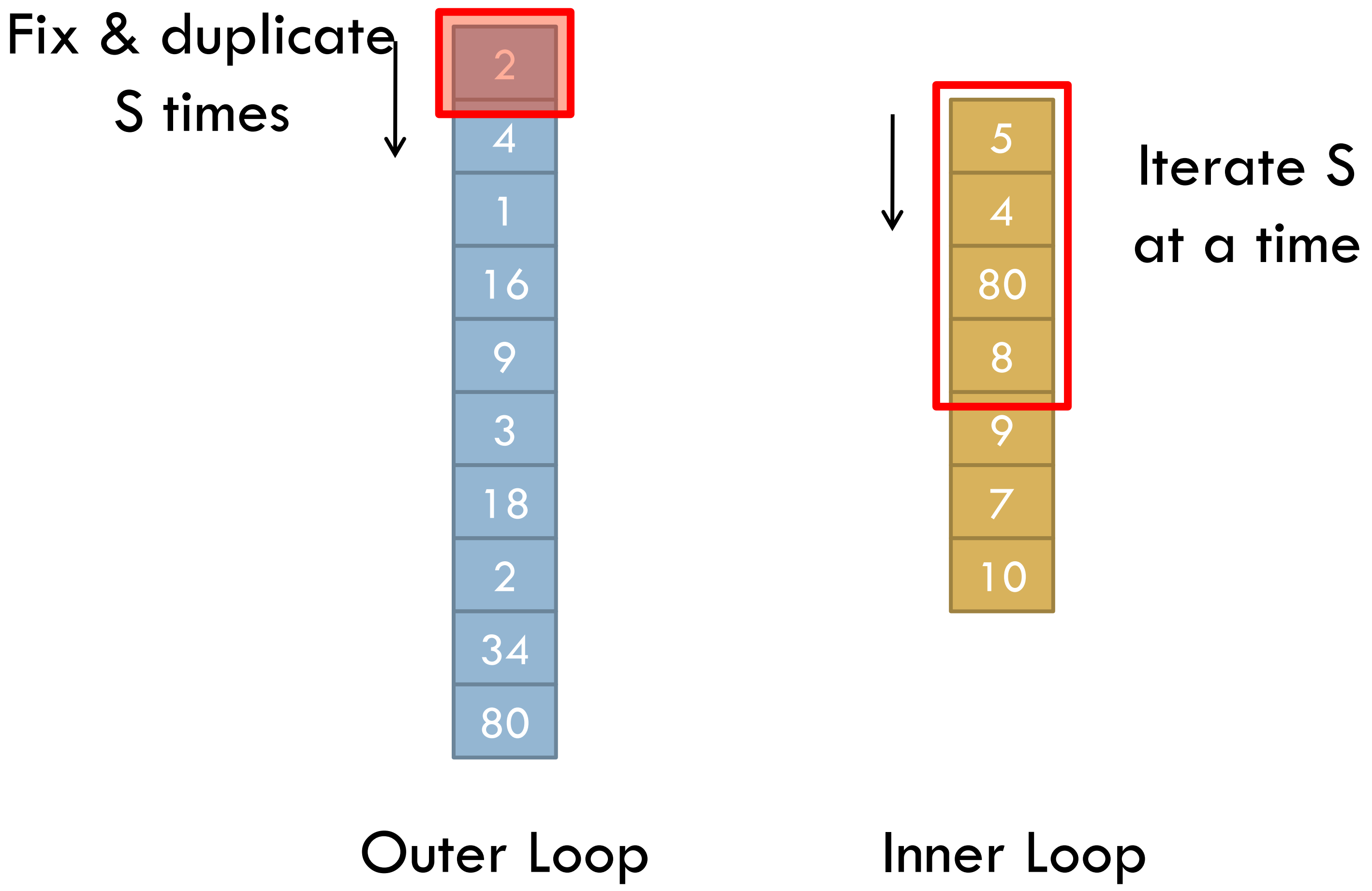
Nested Loop Join – O(n^2)

- SISD Algorithm



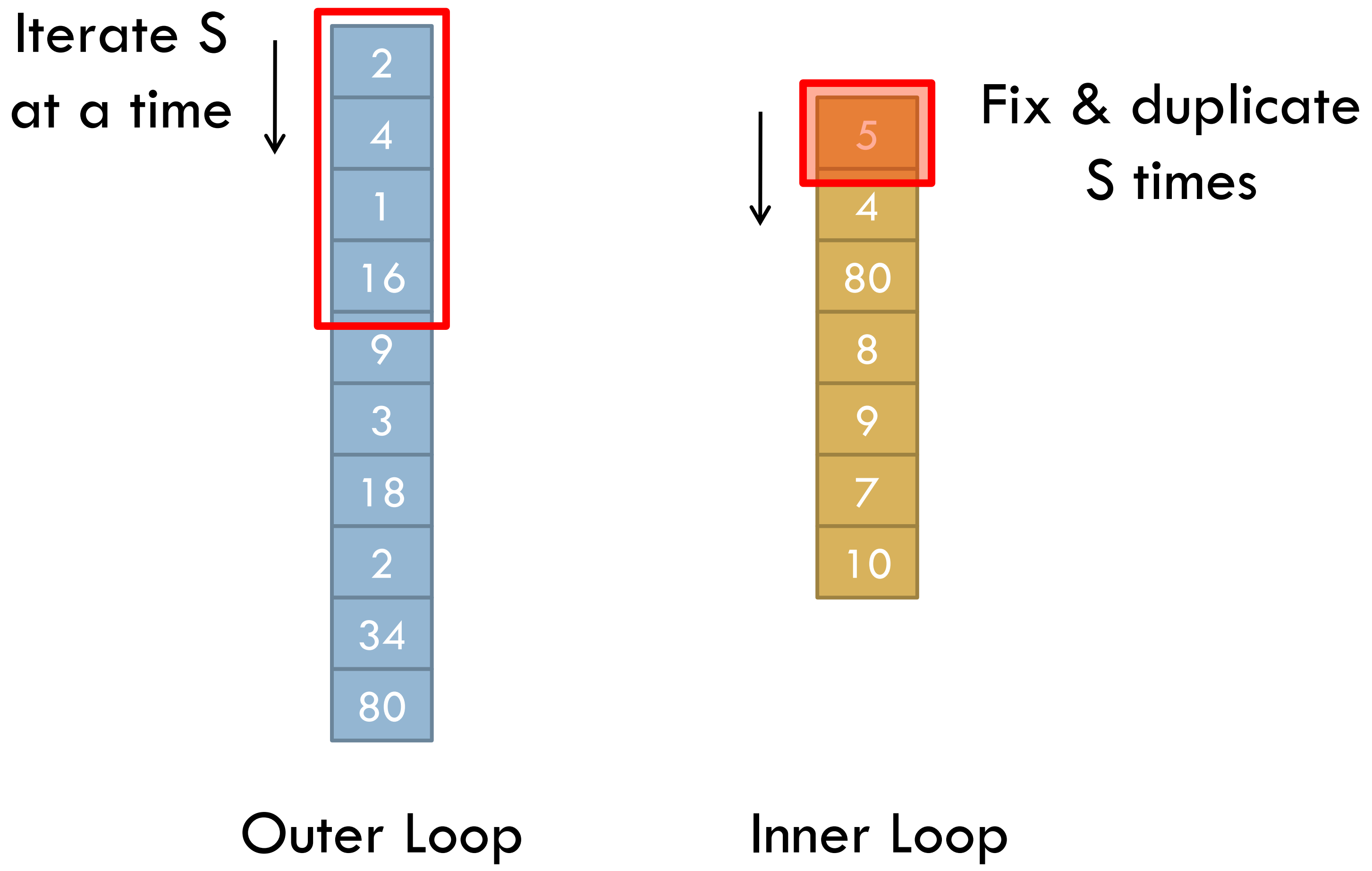
Nested Loop Join – O(n^2)

- SIMD Duplicate-Outer



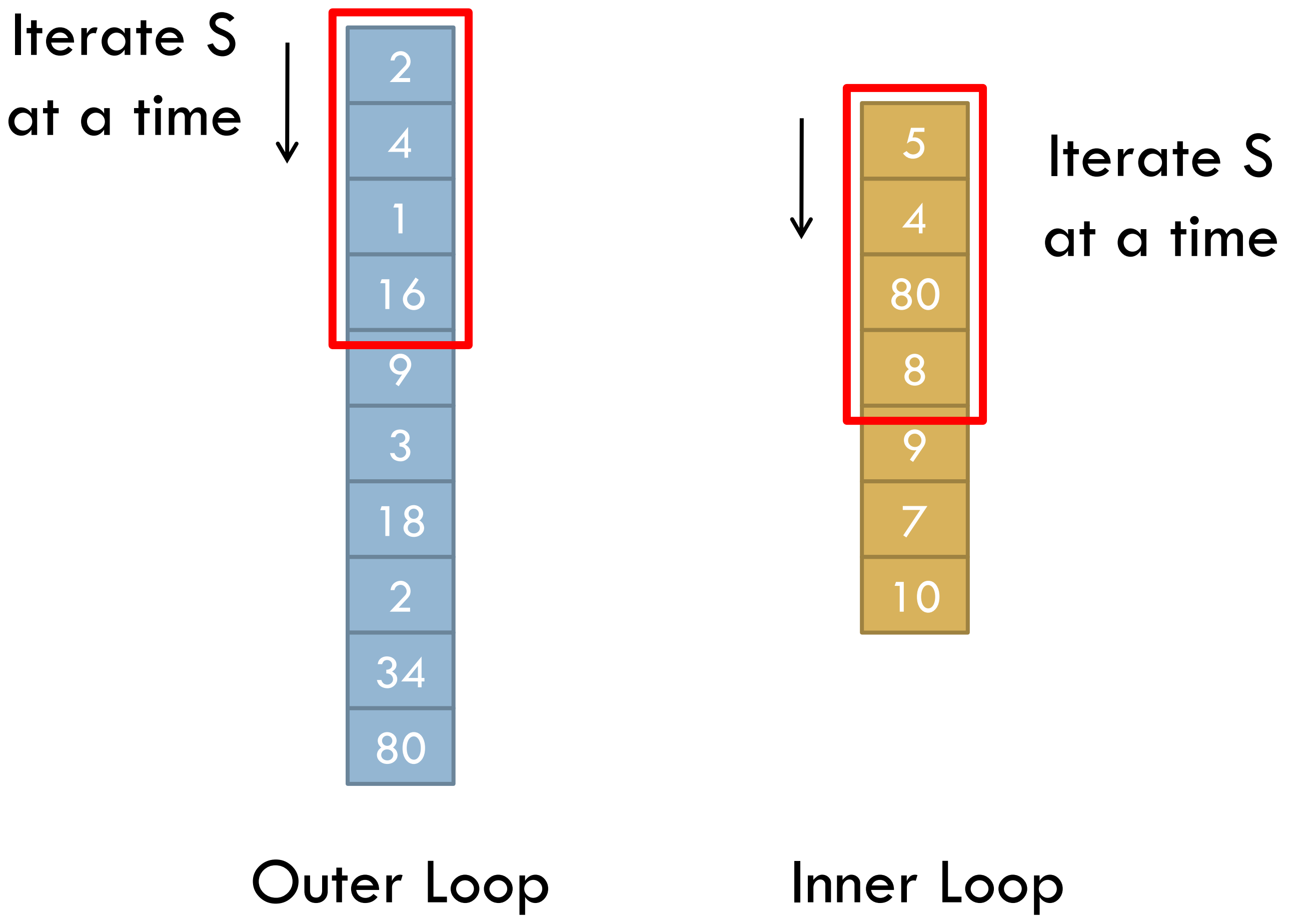
Nested Loop Join – O(n^2)

- SIMD Duplicate-Inner



Nested Loop Join – O(n^2)

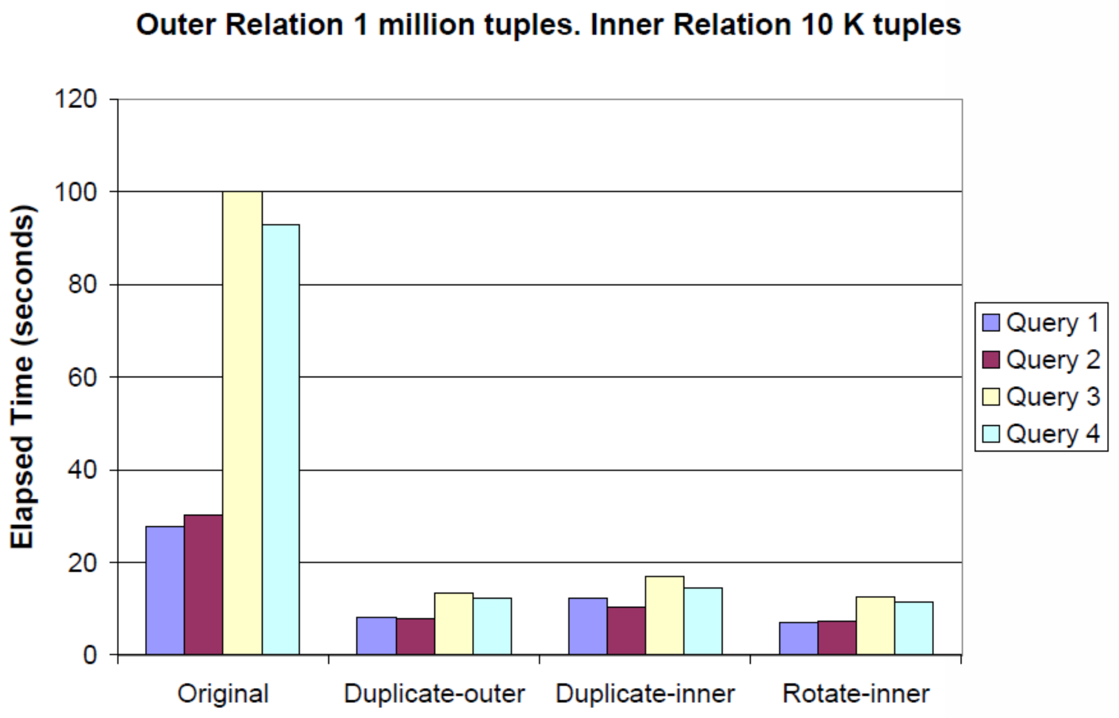
- SIMD Rotate-Inner (Rotate & Compare S times)



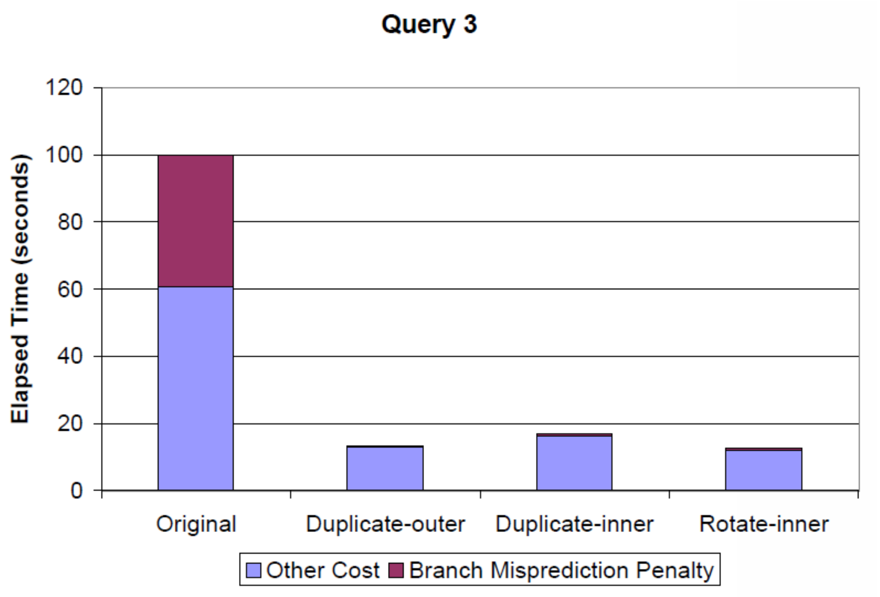
Nested Loop Join – Performance

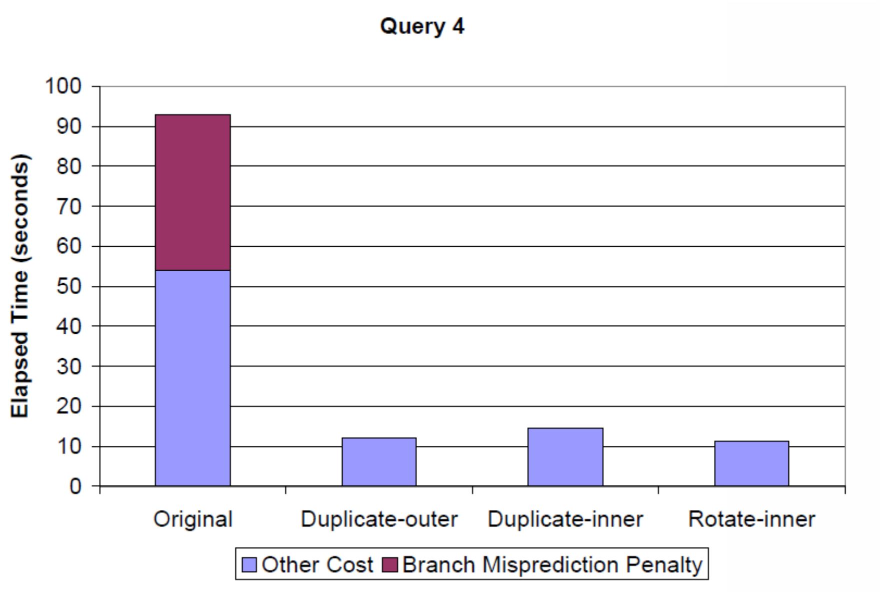
- Queries

```
Q1. SELECT ... FROM R, S WHERE R.Key = S.Key (integer)
Q2. SELECT ... FROM R, S WHERE R.Key = S.Key (floating-point)
Q3. SELECT ... FROM R, S WHERE R.Key < S.Key < 1.01 * R.Key
Q4. SELECT ... FROM R, S WHERE R.Key < S.Key < R.Key + 5
```



Nested Loop Join Branch Misprediction





Conclusion

- Thank you!

?

Questions