CS 181AG
Lecture 9

Prefix Lookup Cont.

Arthi Padmanabhan

Sep 28, 2022

Vehicular Networking

- Discussion

Last Time

- Goals of prefix lookup: 1) low lookup time 2) low memory usage 3) fast insertion
- Tries can help us find longest matching prefix, but looking one bit a time has worst case lookup of 32 ☹
- Looking multiple bits at a time (stride > 1) can help, but then we need to expand prefixes to fit the stride
  - Good for lookup time, bad for memory

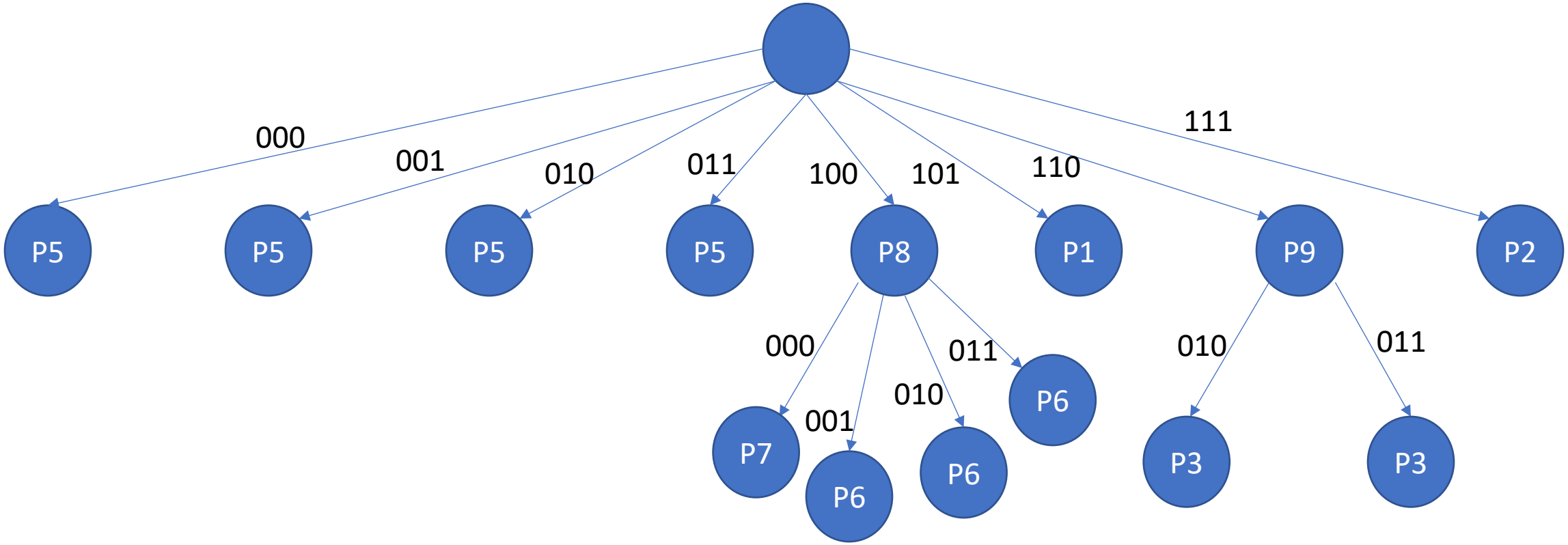
Today

- How can we reduce the memory of multibit tries while keeping the benefit of faster lookup?
- What are some non-trie options for prefix lookup?

Transformed database to multiples of 3

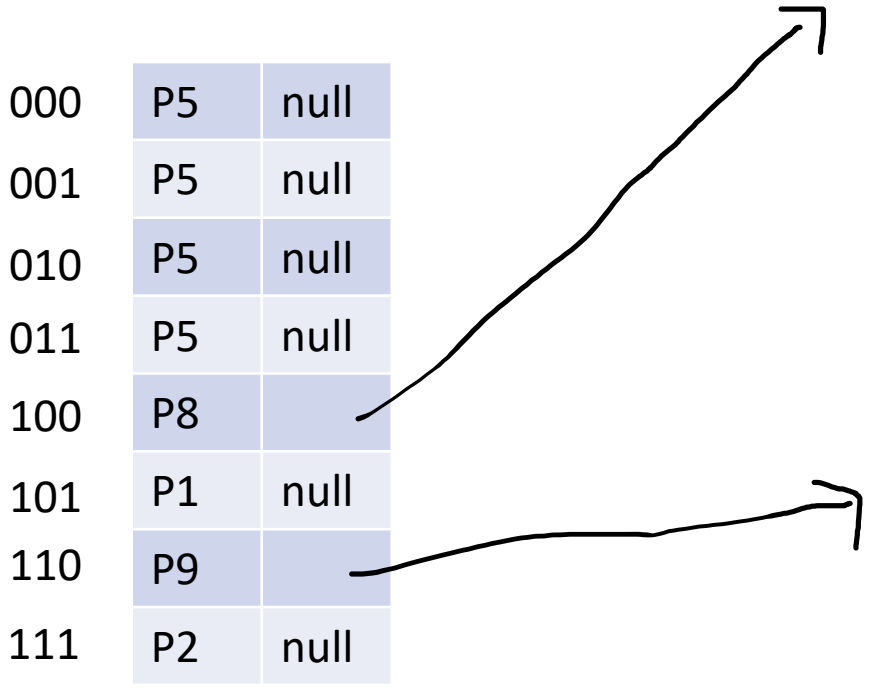
Interface	Prefix	
P1	101*	→ 101*
P2	111*	→ 111*
P3	11001*	→ 110010*, 110011*
P4	1*	→ ~~100*~~, ~~101*~~, ~~110*, 111*~~
P5	0*	→ 000*, 001*, 010*, 011*
P6	1000*	→ ~~100000*,~~ 100001*, 100010*, 100011*
P7	100000*	→ 100000*
P8	100*	→ 100*
P9	110*	→ 110*

Trie formed from expanded prefixes



Multibit Tries in Memory

- Each node contains a list of pairings: <Prefix, Pointer>



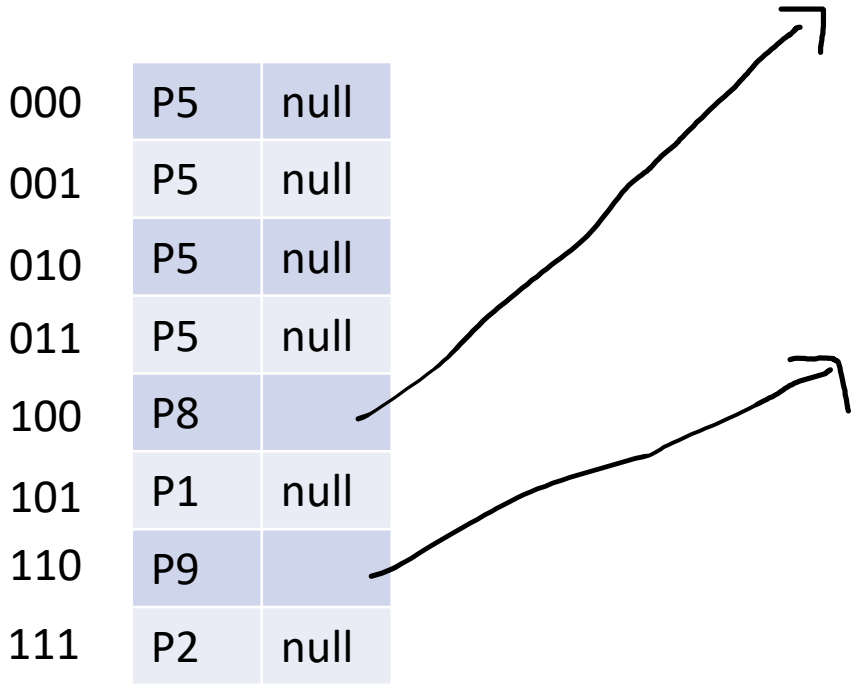
000	P5	null
001	P5	null
010	P5	null
011	P5	null
100	P8	
101	P1	null
110	P9	
111	P2	null

000	P7	null
001	P6	null
010	P6	null
011	P6	null
100		null
101		null
110		null
111		null

000		null
001		null
010	P3	null
011	P3	null
100		null
101		null
110		null
111		null

Variable Stride Length

- Using different stride lengths within the same level



000	P5	null
001	P5	null
010	P5	null
011	P5	null
100	P8	
101	P1	null
110	P9	
111	P2	null

000	P7	null
001	P6	null
010	P6	null
011	P6	null
100		null
101		null
110		null
111		null

00		null
01	P3	null
10		null
11		null

Optimal Stride Length

- For each table, what is the minimum stride length we could use while keeping all information? Assume for now we can only use one table

000	P6
001	P6
010	
011	
100	
101	
110	P3
111	P3

000	P6
001	P6
010	
011	
100	
101	
110	P3
111	

00	P4
01	P4
10	
11	

0000	P6
0001	P6
0010	P6
0011	P6
0100	P3
0101	P3
0110	P2
0111	P2
1000	
1001	
1010	
1011	
1100	P7
1101	P7
1110	P7
1111	P7

Optimal Stride Length

- For each table, what is the minimum stride length we could use while keeping all information? Assume for now we can only use one table

00	P6
01	
10	
11	P3

000	P6
001	P6
010	
011	
100	
101	
110	P3
111	

0	P4
1	

000	P6
001	P6
010	P3
011	P2
100	
101	
110	P7
111	P7

Memory Usage of Tries

- We can save some memory by using variable length strides instead of fixed
- Can we save more?
  - New characteristic when using prefix expansion: more contiguous memory locations carrying same prefix

Lulea-Compressed Tries

- Use bit array and value array to compress each node
- P5 P5 P5 P5 P8 P9 P1 P2 becomes:
  - Bit array: 10001111
  - Value array: P5, P8, P9, P1, P2
- P7 P6 P6 P6 - - - - becomes:
  - Bit array: 11001000
  - Value array: P7, P6, –
- - P3 - - becomes
  - Bit array: 1110
  - Value array: -, P3, -

Separate pointer array?

- Is it possible to encode these tables such that we only keep a single bit array and value array each? How do we include pointers?

000	P7	null
001	P6	null
010	P6	null
011	P6	null
100		null
101		null
110		null
111		null

000	P5	null
001	P5	null
010	P5	null
011	P5	null
100	P8	
101	P1	null
110	P9	
111	P2	null

ptr1

ptr2

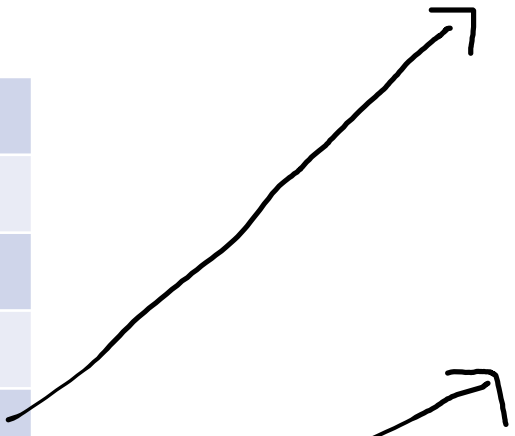
00		null
01	P3	null
10		null
11		null

Leaf Pushing

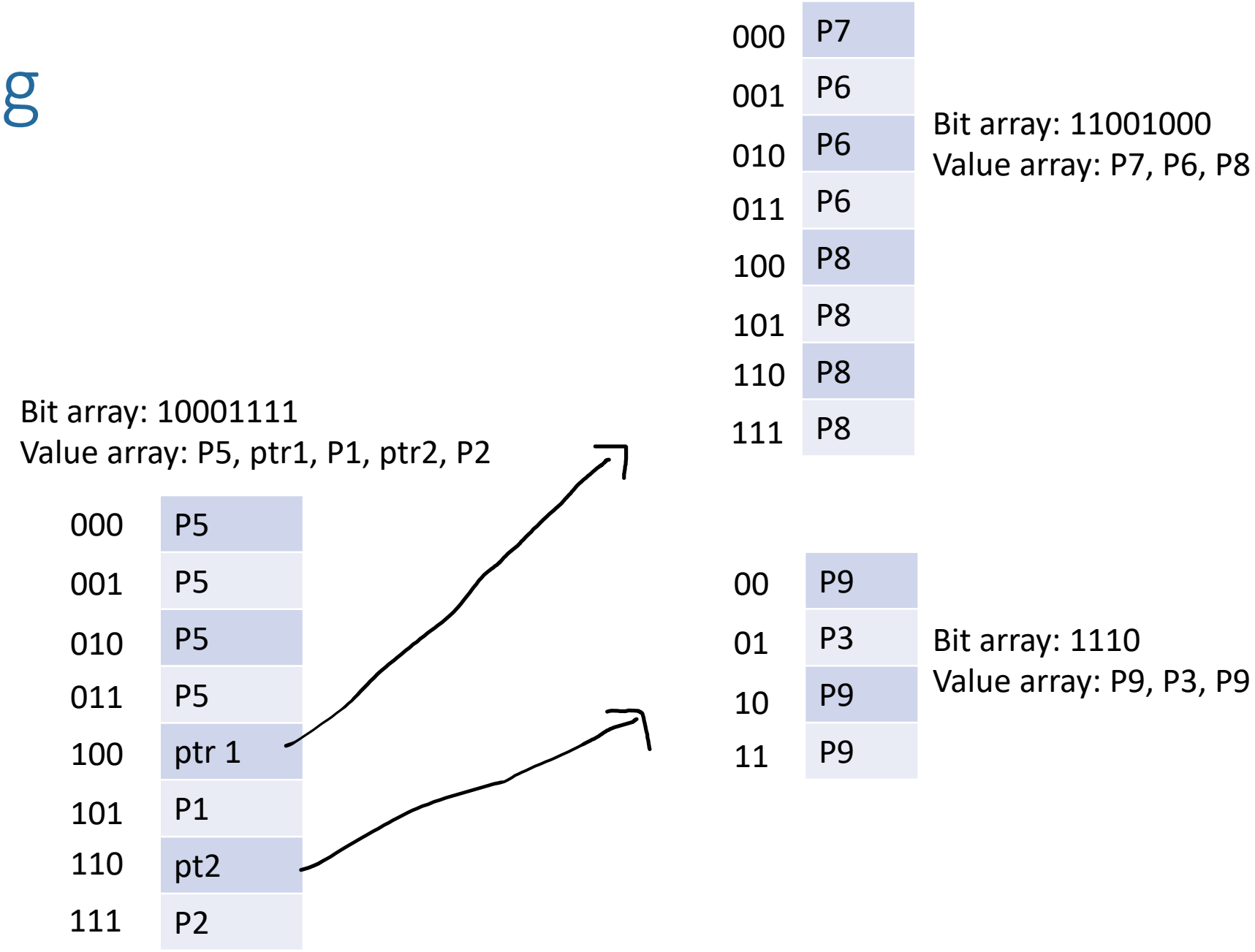
000	P7
001	P6
010	P6
011	P6
100	P8
101	P8
110	P8
111	P8

000	P5
001	P5
010	P5
011	P5
100	ptr 1
101	P1
110	pt2
111	P2

00	P9
01	P3
10	P9
11	P9

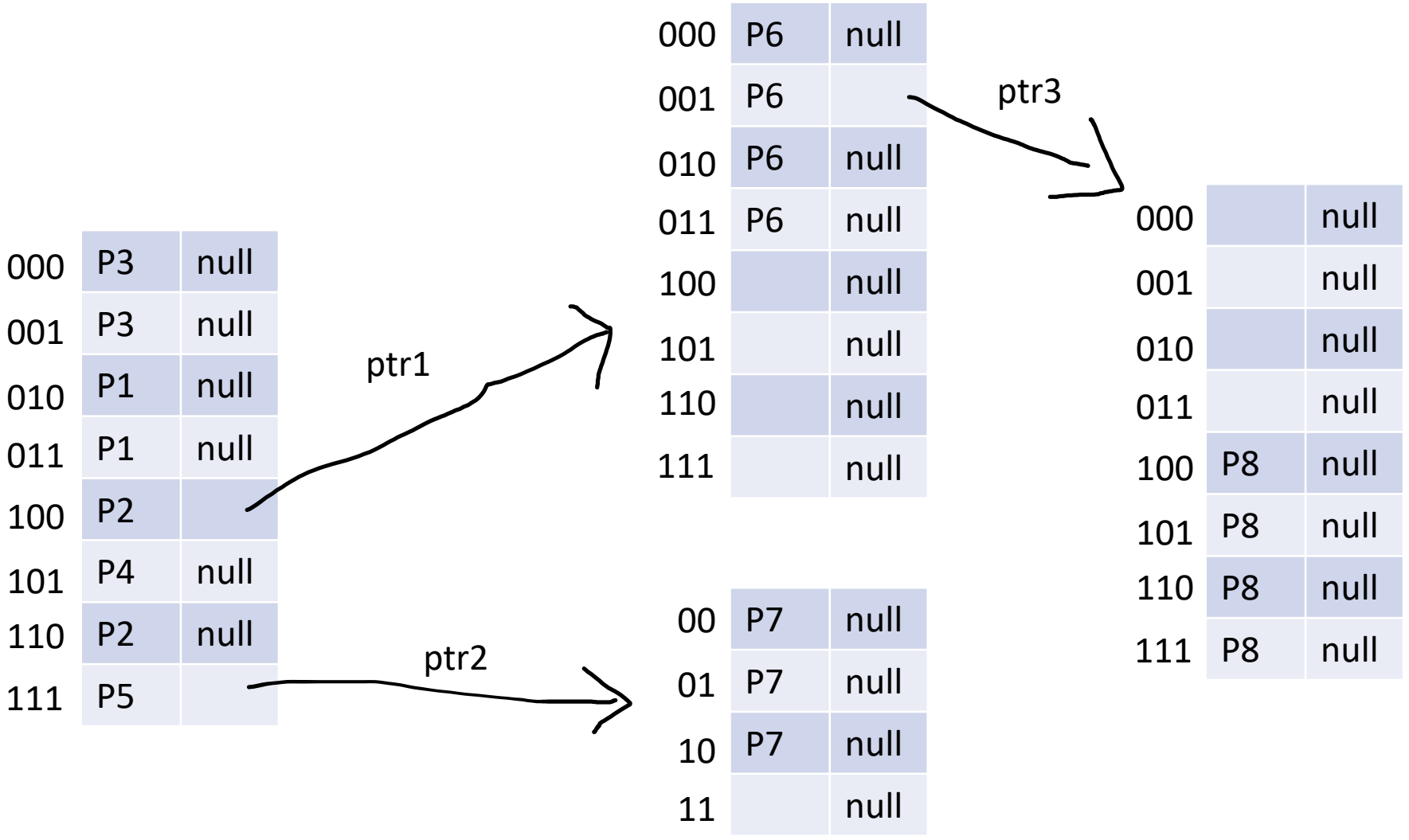


Leaf Pushing



Practice: Compress these nodes

- For each node, write the bit array and value array



Compression Properties

- Original version of Lulea proposed using fixed stride lengths of 16-8-8
- Stride of 16 -> array of length 2^16
- Longer stride -> more contiguous array slots with the same value -> more opportunities for compression

Lookup Using Compressed Tries

- Steps for lookup:
  - Index into bit array and count the number of 1s so far
  - Use that number to index into value array
- With 2^16 length bit array, very slow!

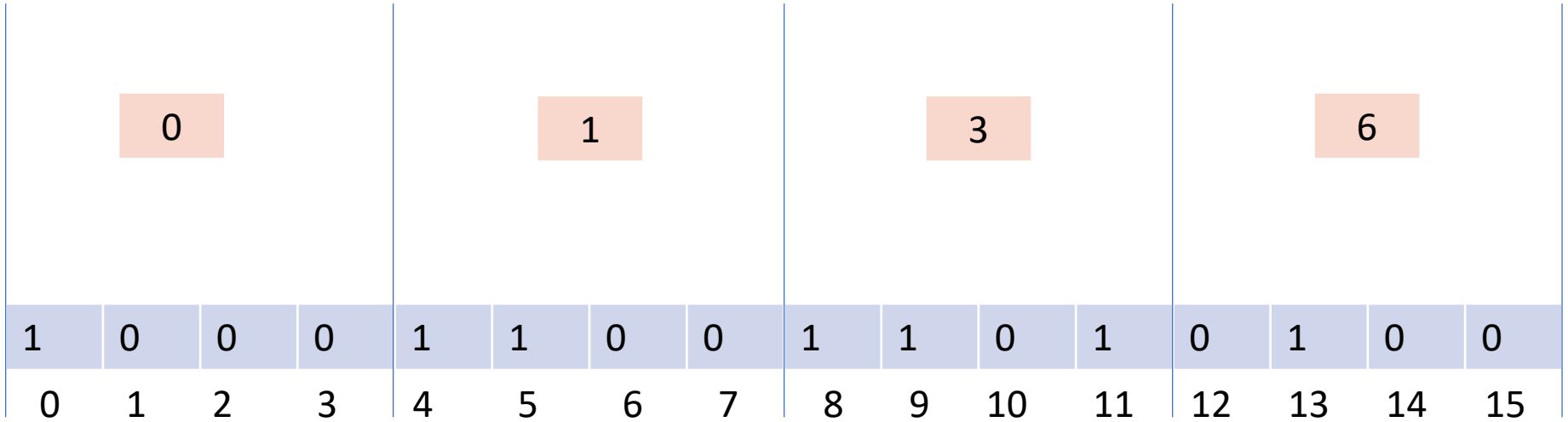
Speed up Bit Array Counting

- Consider this bit array of length 16.
  - Worst-case number of lookups (assume one bit at a time)?
- Can we lower worst-case number of lookups to count the 1s?
- Assume the following:
  - You can use more memory, but you don't want the extra memory to exceed 25% of the bit array length
- Hint: you can do any precomputation you'd like

1	0	0	0	1	1	0	0	1	1	0	1	0	1	0	0

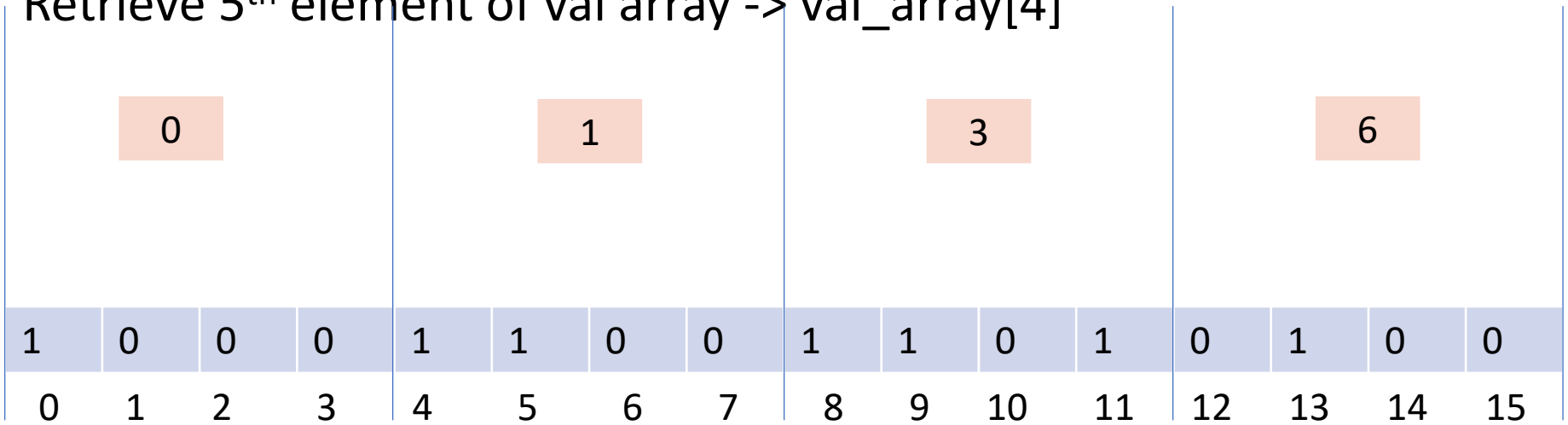
Speed up Bit Array Counting

- Divide bitmap into fixed-size chunks, precompute the number of 1s in each chunk
- Summary number corresponding to chunk X keeps Count(X-1): the total number of 1s encountered *before* this chunk



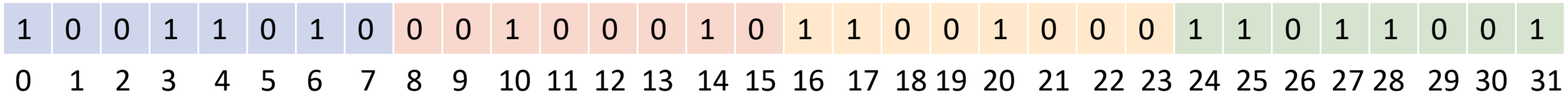
Speed up Bit Array Counting

- To access element at index 10:
  - Floor(index / chunk size) = 10/4 = 2
  - Index 2: 3 1s *before* this chunk; within this chunk, 2 1s up to and including index 10
  - Total = 3 +2 = 5 1s
  - Retrieve 5th element of val array -> val_array[4]



Practice

- Given the following bitmap, and value array:
  - Write the summary table
  - What values should be returned when accessing the element at index:
    - 3
    - 4
    - 11
    - 28



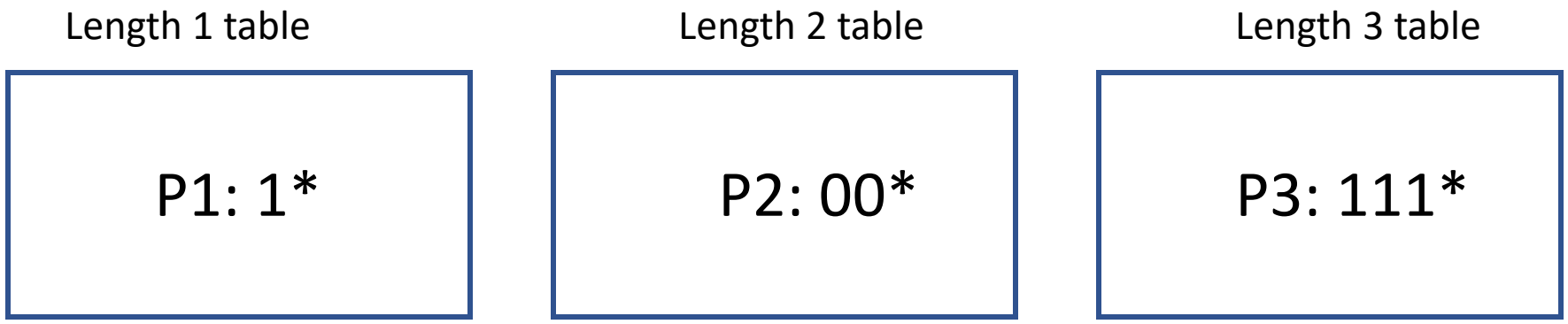
P1	ptr1	P1	P4	P2	P3	P2	P5	P6	P7	ptr2	P6	ptr3	P8

Number of lookups?

- One for summary table
- One for counting within chunk (hardware helps!)
- One for value array

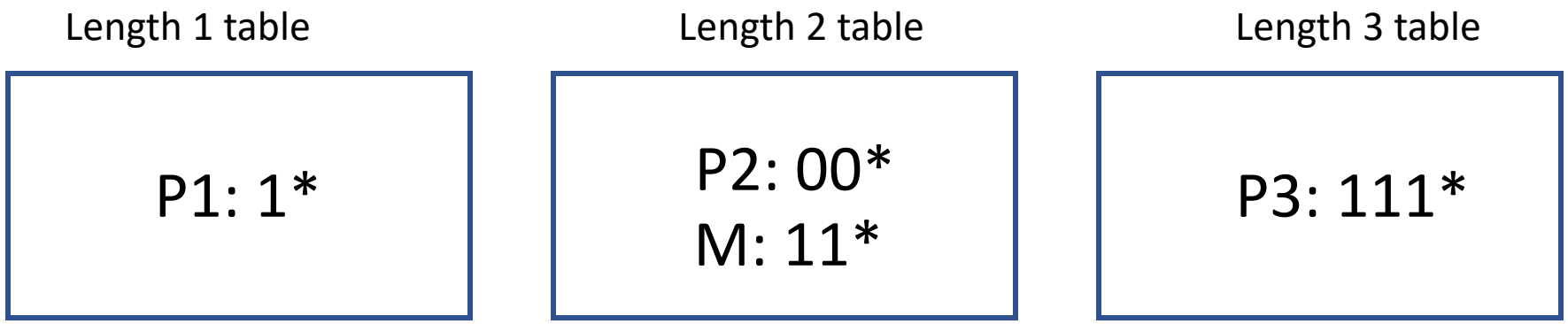
Non-Trie Options

- Binary search on prefix lengths



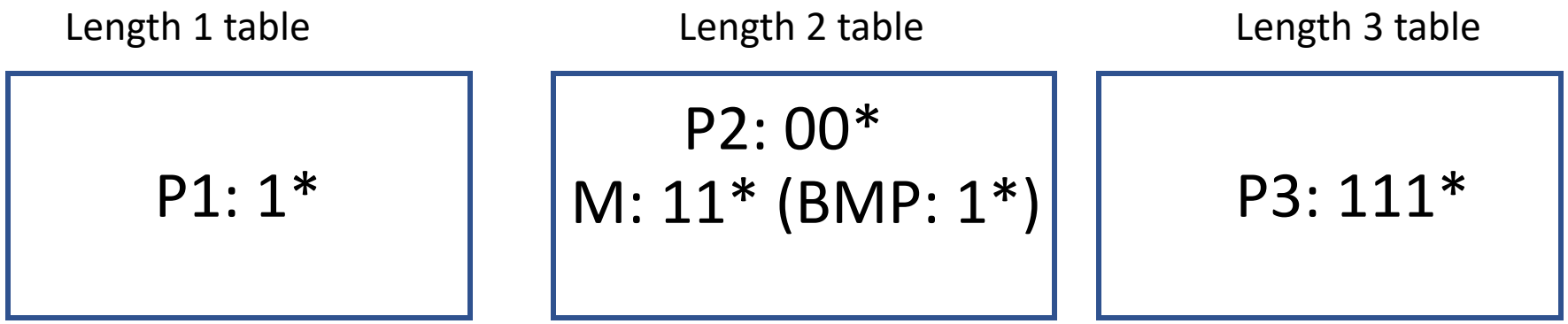
Non-Trie Options

- Binary search on prefix lengths



Non-Trie Options

- Binary search on prefix lengths



Practice

- Separate by prefix lengths and add marker for each element in the right half of the table
- If prefix already exists in shorter-length table, you don't need to add a marker

Interface	Prefix
P1	101*
P2	111*
P3	11001*
P4	1*
P5	0*
P6	1000*
P7	100000*
P8	100*
P9	110*

Next Time

- One more prefix lookup algorithms
- What if we need to look at packet header info beyond just destination, e.g., for firewalls?