Last time

(Midterm)

Implementation

- Stacks
- Queues

Linked Lists

Today

Unbounded arrays

Amortized analysis

Next

Hash tables

Toward unbounded arrays

	Unsorted Arrays	Linked Lists
PROS	O(1) access Built-in support	Self-resizing O(1) insertion (given right pointers)
CONS	Fixed size O(n) insertion	O(n) access No built-in support

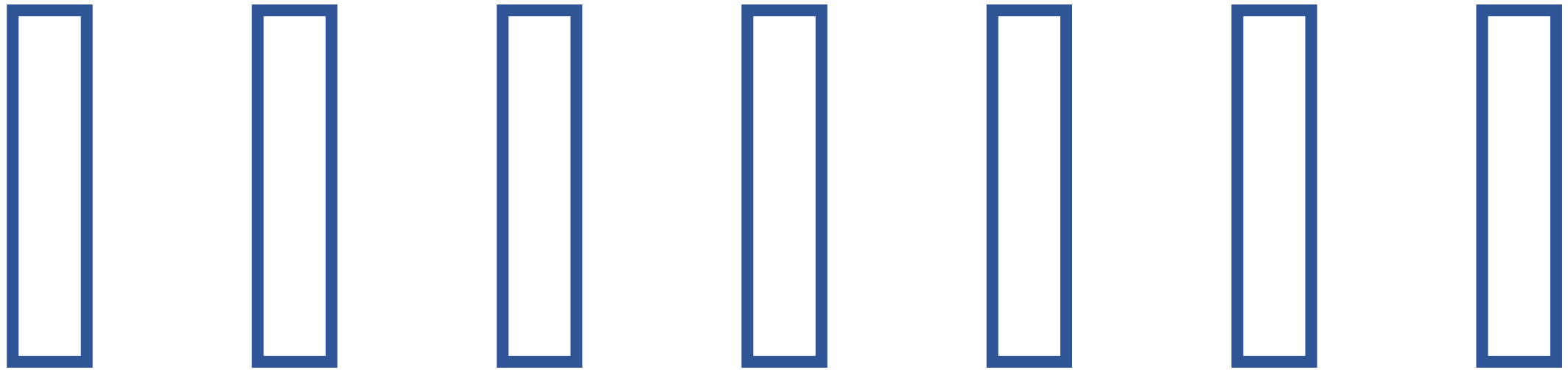
A data structure that combines the best properties of arrays and linked lists

Amortized analysis

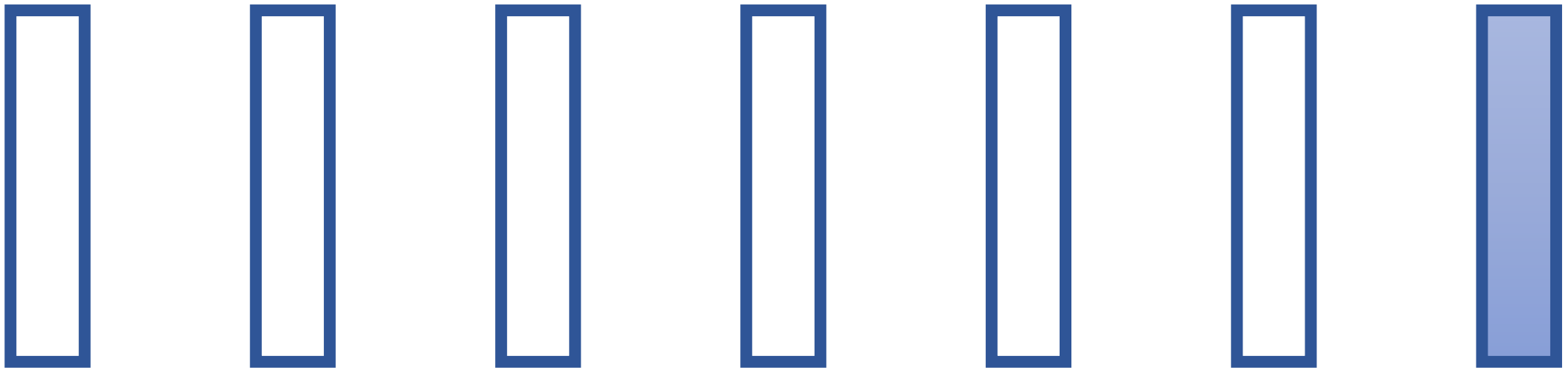
- Average ***frequent cheap*** operations

  with ***infrequent expensive*** operations

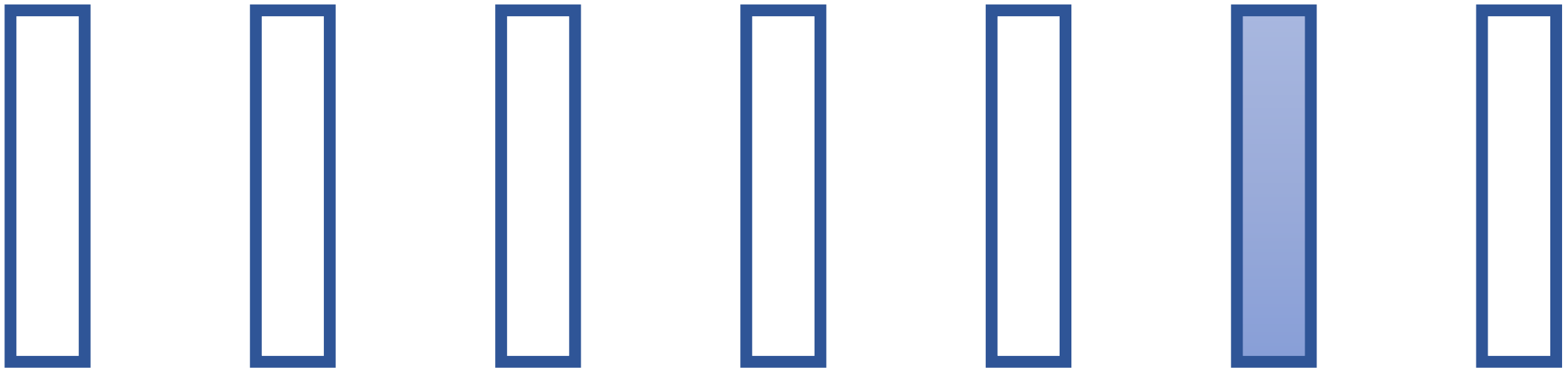
Binary Counter



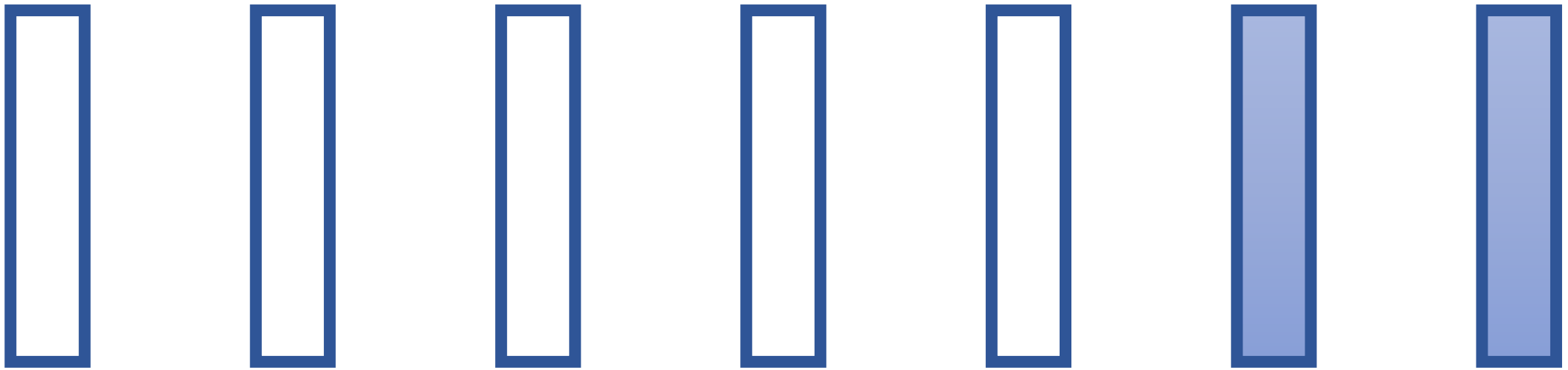
Binary Counter



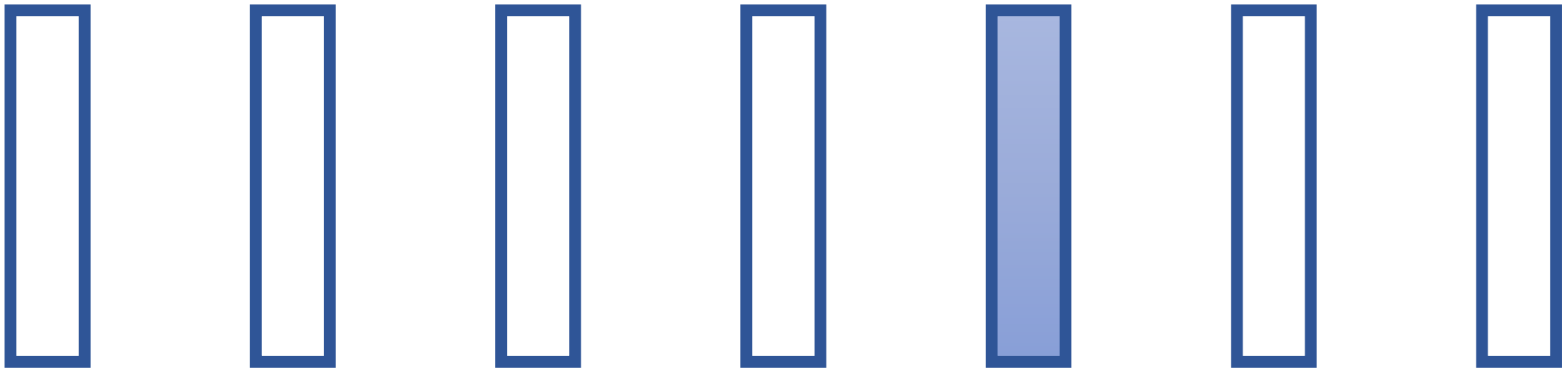
Binary Counter



Bay Bridge Counter



Binary Counter



n-bit counter

Press #	0 0 0 0 0 0	Per press Cost
1	0 0 0 0 0 1	$ 1
2	0 0 0 0 1 0	$ 2
3	0 0 0 0 1 1	$ 1
4	0 0 0 1 0 0	$ 3
5	0 0 0 1 0 1	$ 1
6	0 0 0 1 1 0	$ 2
7	0 0 0 1 1 1	$ 1
8	0 0 1 0 0 0	$ 4

n-bit counter

	Per press
1 0 1 0 1 1	Cost
_ _ _ _ _ _	$?

n-bit counter

$1 / press

Press #	0 0 0 0 0 0	Per press Cost	Total Cost	Total Revenue	Bank Account
1	0 0 0 0 0 1	1	1	1	0
2	0 0 0 0 1 0	2	3	2	-1
3	0 0 0 0 1 1	1	4	3	-1
4	0 0 0 1 0 0	3	7	4	-3
5	0 0 0 1 0 1	1	8	5	-3
6	0 0 0 1 1 0	2	10	6	-4
7	0 0 0 1 1 1	1	11	7	-4
8	0 0 1 0 0 0	4	15	8	-7

n-bit counter

$2 / press

Press #	0 0 0 0 0 0	Per press Cost	Total Cost	Total Revenue	Bank Account
1	0 0 0 0 0 1	1	1	2	1
2	0 0 0 0 1 0	2	3	4	1
3	0 0 0 0 1 1	1	4	6	2
4	0 0 0 1 0 0	3	7	8	1
5	0 0 0 1 0 1	1	8	10	2
6	0 0 0 1 1 0	2	10	12	2
7	0 0 0 1 1 1	1	11	14	3
8	0 0 1 0 0 0	4	15	16	1

n-bit counter

$2 / press

Press #	0 0 0 0 0 0	Per press Cost	Total Cost	Total Revenue	Bank Account
1	0 0 0 0 0 1	1	1	2	1
2	0 0 0 0 1 0	2	3	4	1
3	0 0 0 0 1 1	1	4	6	2
4	0 0 0 1 0 0	3	7	8	1
			8	10	2
			10	12	2
			11	14	3
			15	16	1

Average cost $2, O(1)

Will this keep working?

Can we prove it?

Invariant?

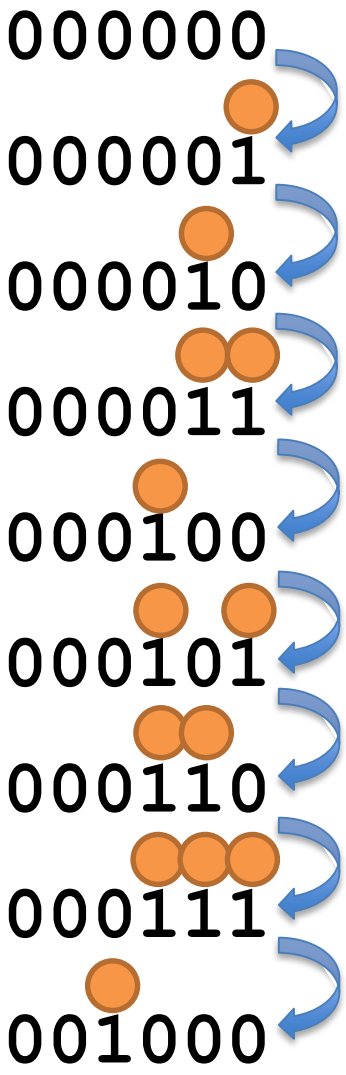
$2 / press

Press #	0 0 0 0 0 0	Per press Cost	Total Cost	Total Revenue	Bank Account
1	0 0 0 0 0 1	1	1	2	1
2	0 0 0 0 1 0	2	3	4	1
3	0 0 0 0 1 1	1	4	6	2
4	0 0 0 1 0 0	3	7	8	1
5	0 0 0 1 0 1	1	8	10	2
6	0 0 0 1 1 0	2	10	12	2
7	0 0 0 1 1 1	1	11	14	3
8	0 0 1 0 0 0	4	15	16	1

n-bit counter

000000				
000001	charge 2 tokens	Pay 1	1 in reserve	
000010	charge 2 tokens	Pay 2	1 in reserve	
000011	charge 2 tokens	Pay 1	2 in reserve	
000100	charge 2 tokens	Pay 3	1 in reserve	
000101	charge 2 tokens	Pay 1	2 in reserve	
000110	charge 2 tokens	Pay 2	2 in reserve	
000111	charge 2 tokens	Pay 1	3 in reserve	
001000	charge 2 tokens	Pay 4	1 in reserve	

n-bit counter



charge 2 tokens	Pay 1	1 in reserve
charge 2 tokens	Pay 2	1 in reserve
charge 2 tokens	Pay 1	2 in reserve
charge 2 tokens	Pay 3	1 in reserve
charge 2 tokens	Pay 1	2 in reserve
charge 2 tokens	Pay 2	2 in reserve
charge 2 tokens	Pay 1	3 in reserve
charge 2 tokens	Pay 4	1 in reserve

Invariant: Preservation

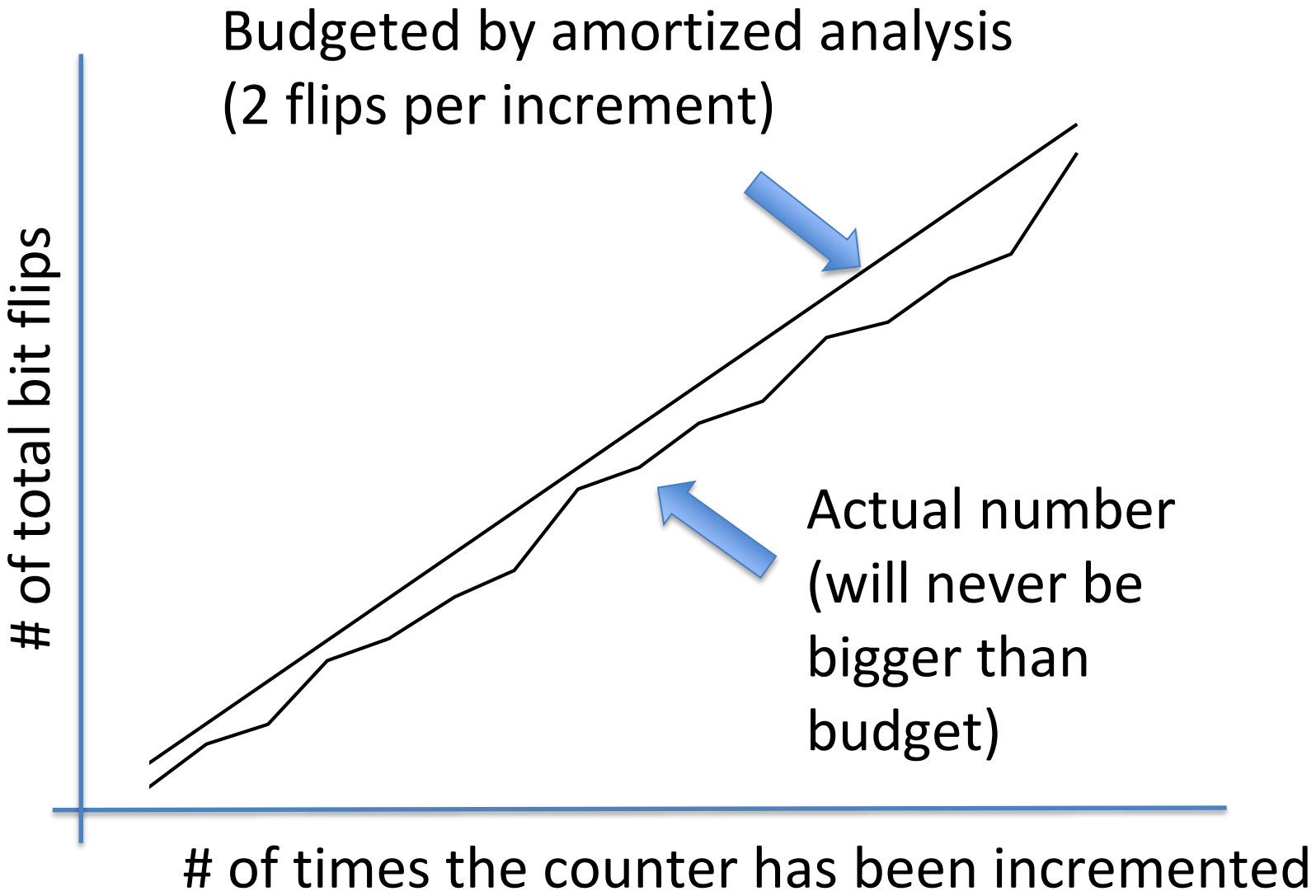
0 1 0 1 0 1 1

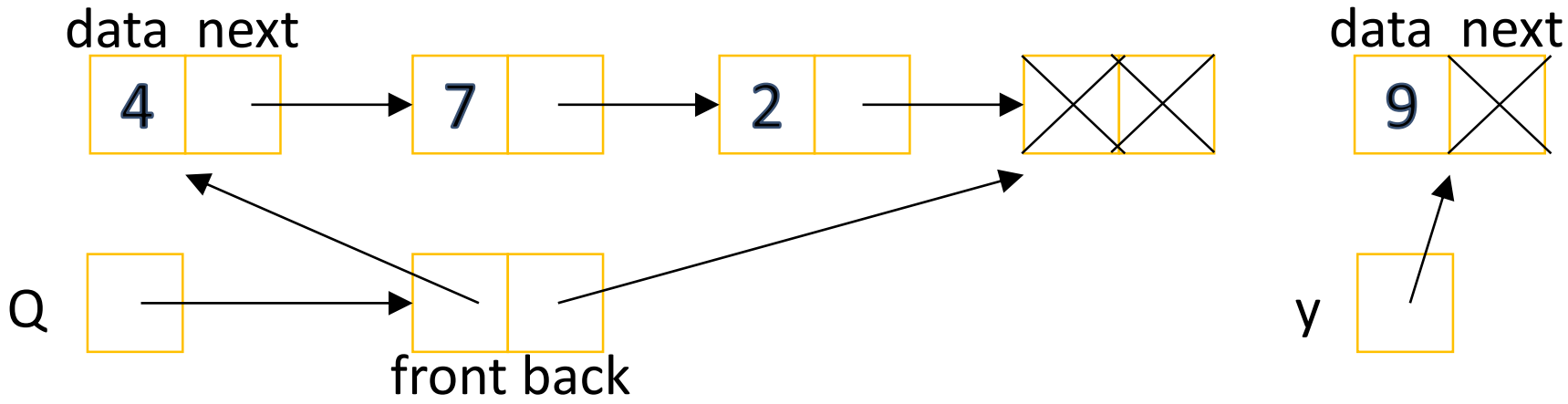
_ _ _ _ _ _ _

1. Assume invariant
   k tokens for k ones

2. Prove preservation
   k’ tokens for k’ ones

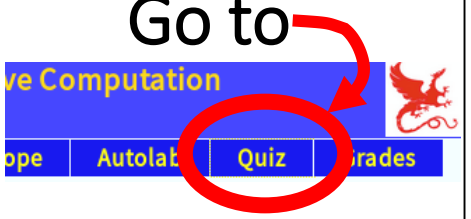
Amortized Analysis





1. If you access `Q->front`, what Boolean expression must be true to ensure safety?
2. What Boolean expression indicates whether this queue is empty?
3. Write the statements in the correct order that will make node `y` the front of this non-empty queue.

Go to



or

cs.cmu.edu/~15122/quiz

4. TRUE or FALSE: In C0, setting `y->next = y` is allowed.
5. If `y` is of type `list*`, the type of `*y` is _____

Unbounded arrays interface

```
// typedef ______* uba_t;

int uba_len(uba_t A)
/*@requires A != NULL;   @*/
/*@ensures \result >= 0; @*/ ;

uba_t uba_new(int size)
/*@requires 0 <= size;                @*/
/*@ensures \result != NULL;           @*/
/*@ensures uba_len(\result) == size; @*/ ;

string uba_get(uba_t A, int i)
/*@requires A != NULL;                 @*/
/*@requires 0 <= i && i < uba_len(A); @*/;

void uba_set(uba_t A, int i, string x)
/*@requires A != NULL;                 @*/
/*@requires 0 <= i && i < uba_len(A); @*/;
```

```
void uba_add(uba_t A, string x)
/*@requires A != NULL; @*/ ;

string uba_rem(uba_t A)
/*@requires A != NULL;       @*/
/*@requires 0 < uba_len(A); @*/ ;
```

new functionality

Unbounded Array

Action

Start with [4, 7]

- add(5)
- rem()
- add(9)

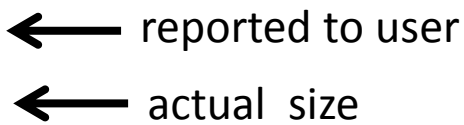
```
struct uba_header {
  int length;

  int[] data;
};
```

User View

Implementation View

Data structure

```
typedef struct uba_header uba;
struct uba_header {
  int size;           // 0 <= size && size < limit
  int limit;          // 0 < limit
  string[] data;      // \length(data) == limit
};
```



Data structure invariant

```
bool is_uba(uba* A) {
  return A != NULL
      && is_array_expected_length(A->data, A->limit)
      && 0 <= A->size
      && A->size < A->limit;
}
```

Resizing the array

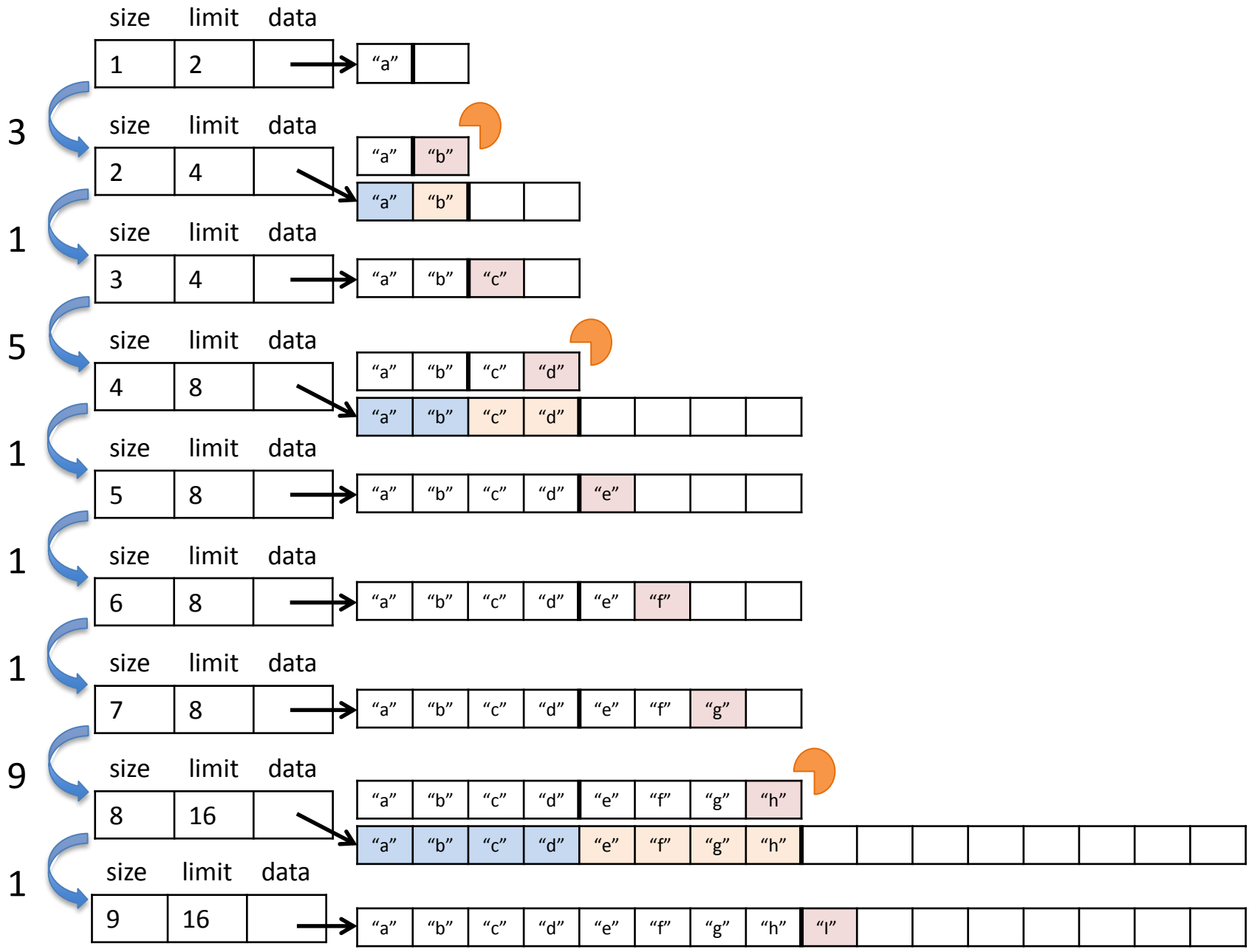
Make `uba_add` have constant amortized cost

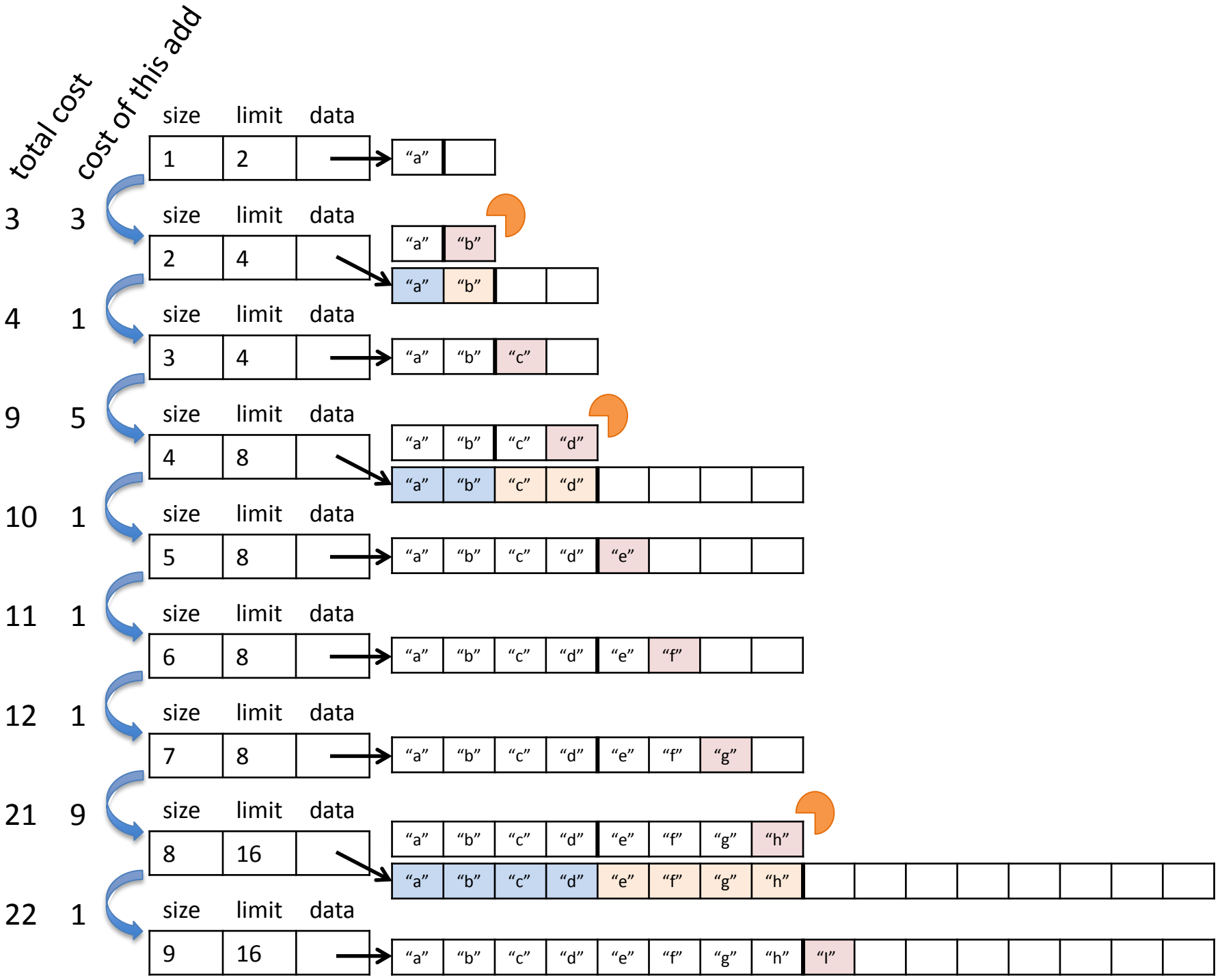
Arrange so that resizing happens rarely

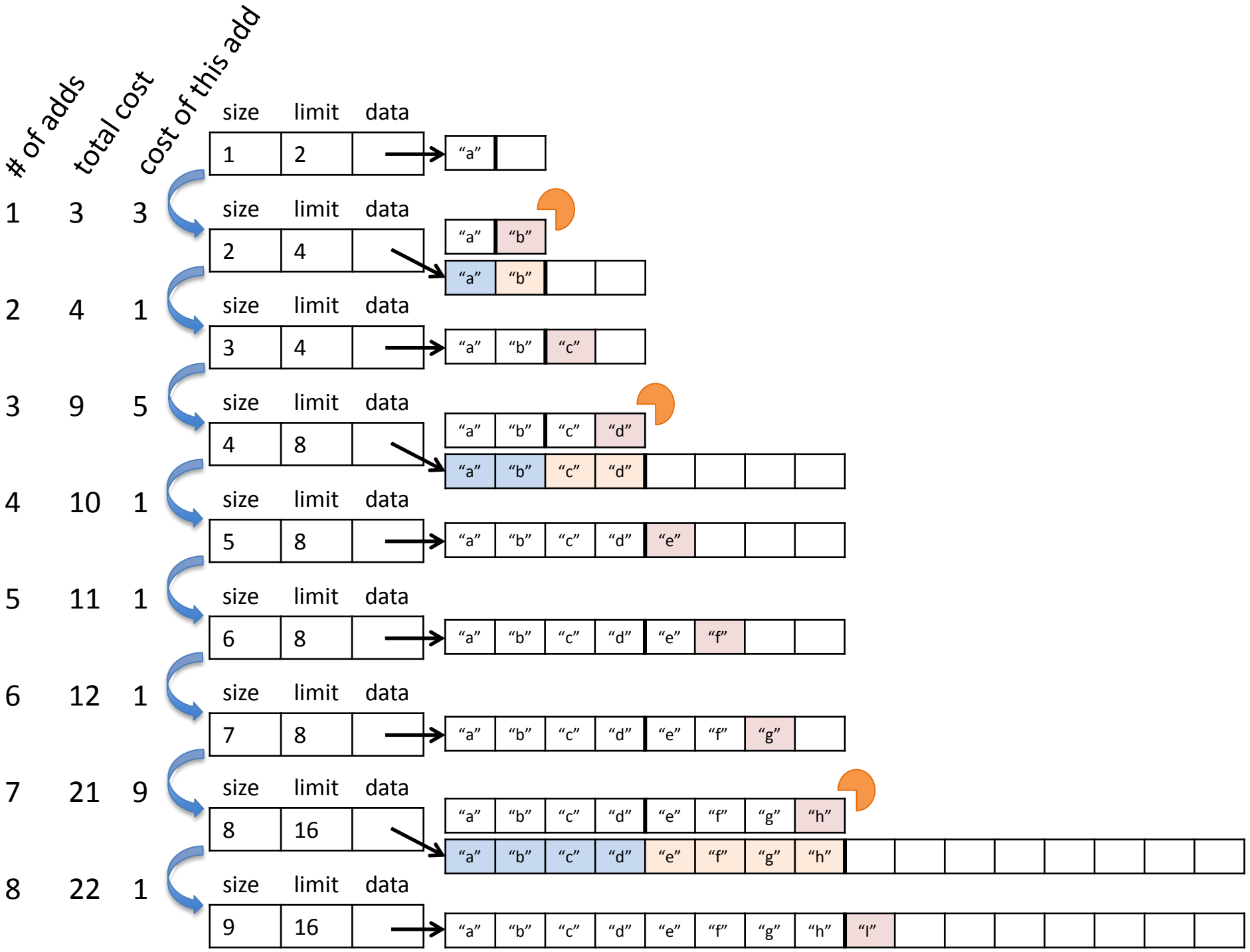
making new array of length limit+1 won't work. Why?

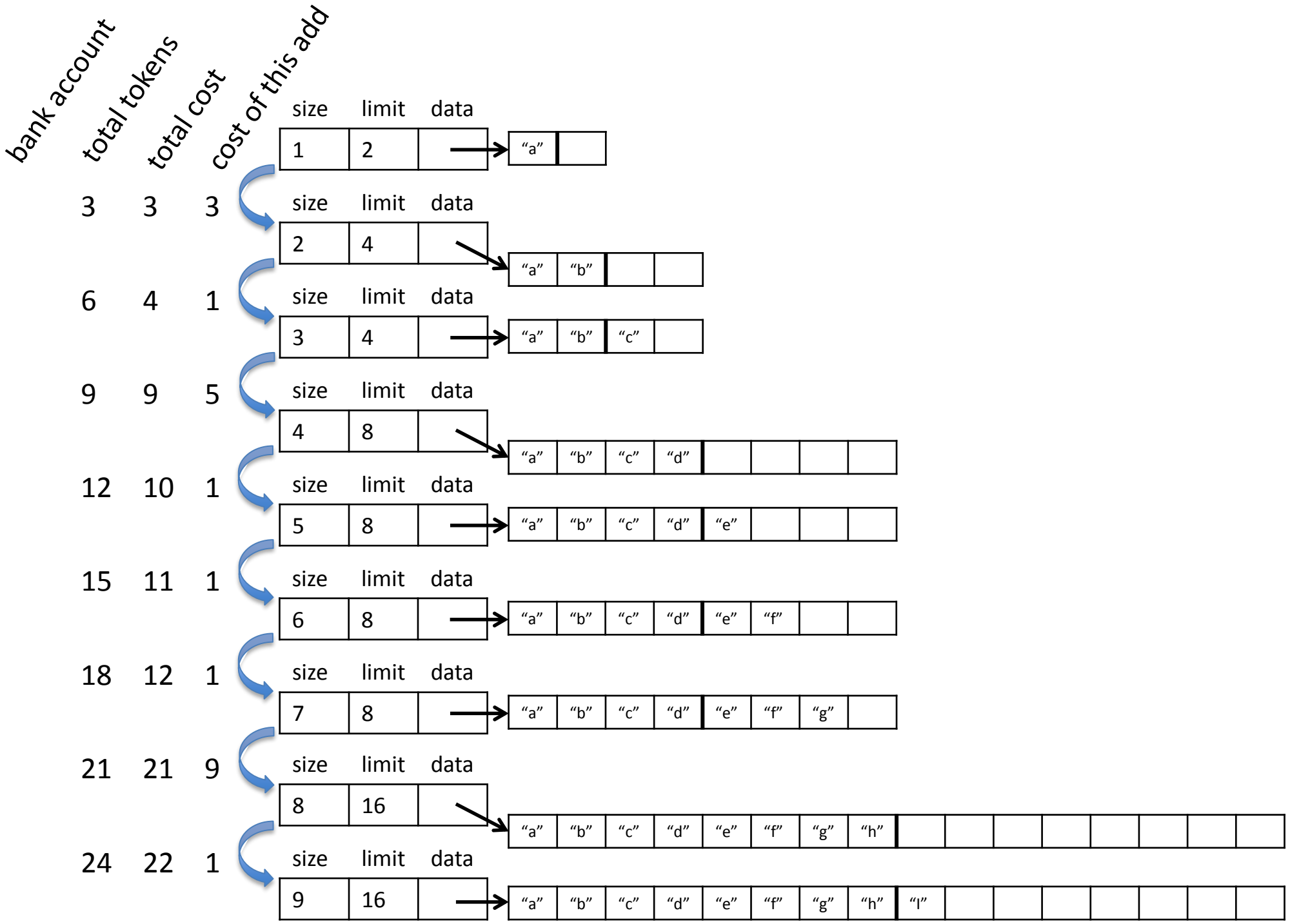
Resize 2x

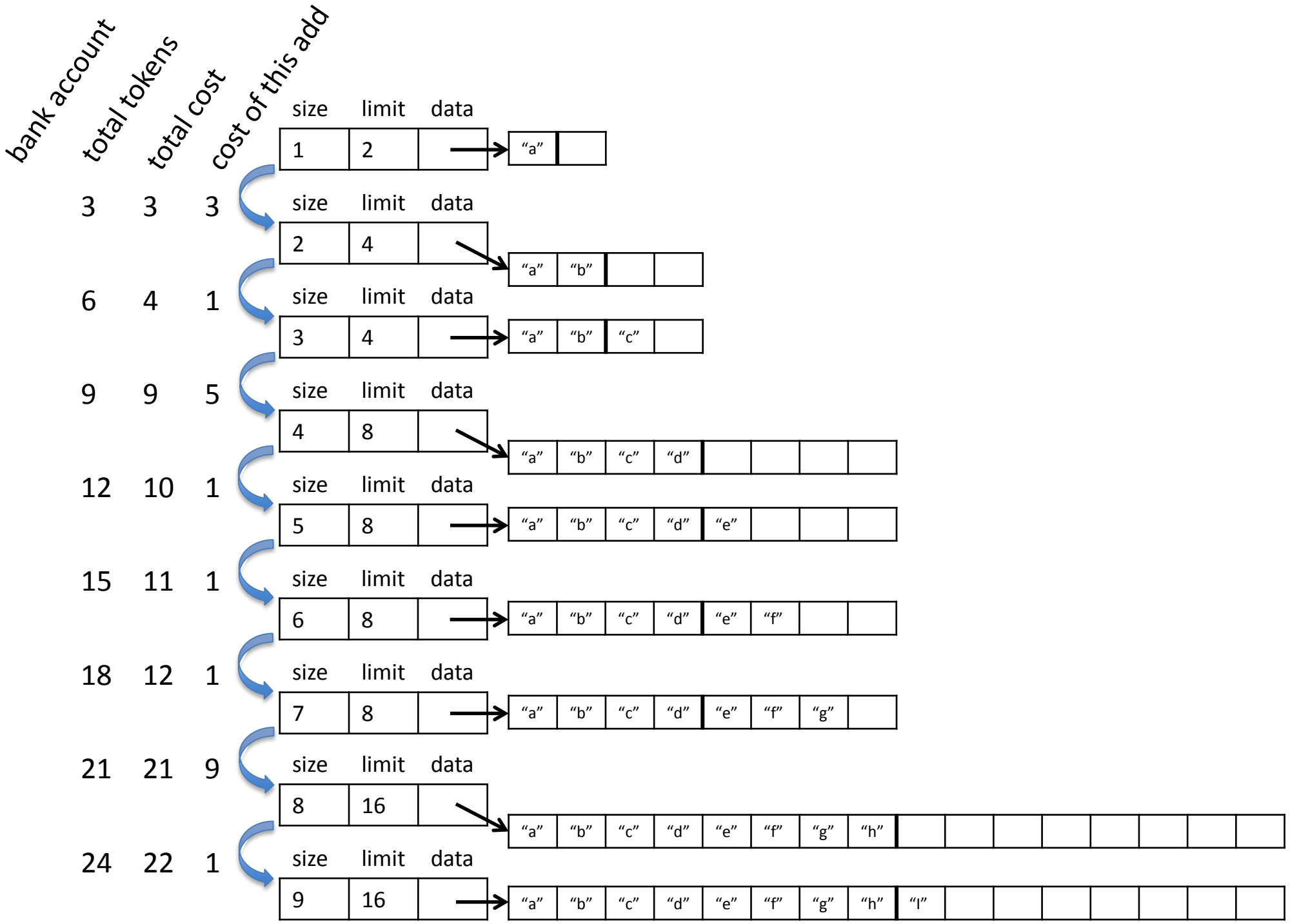
cost of this add

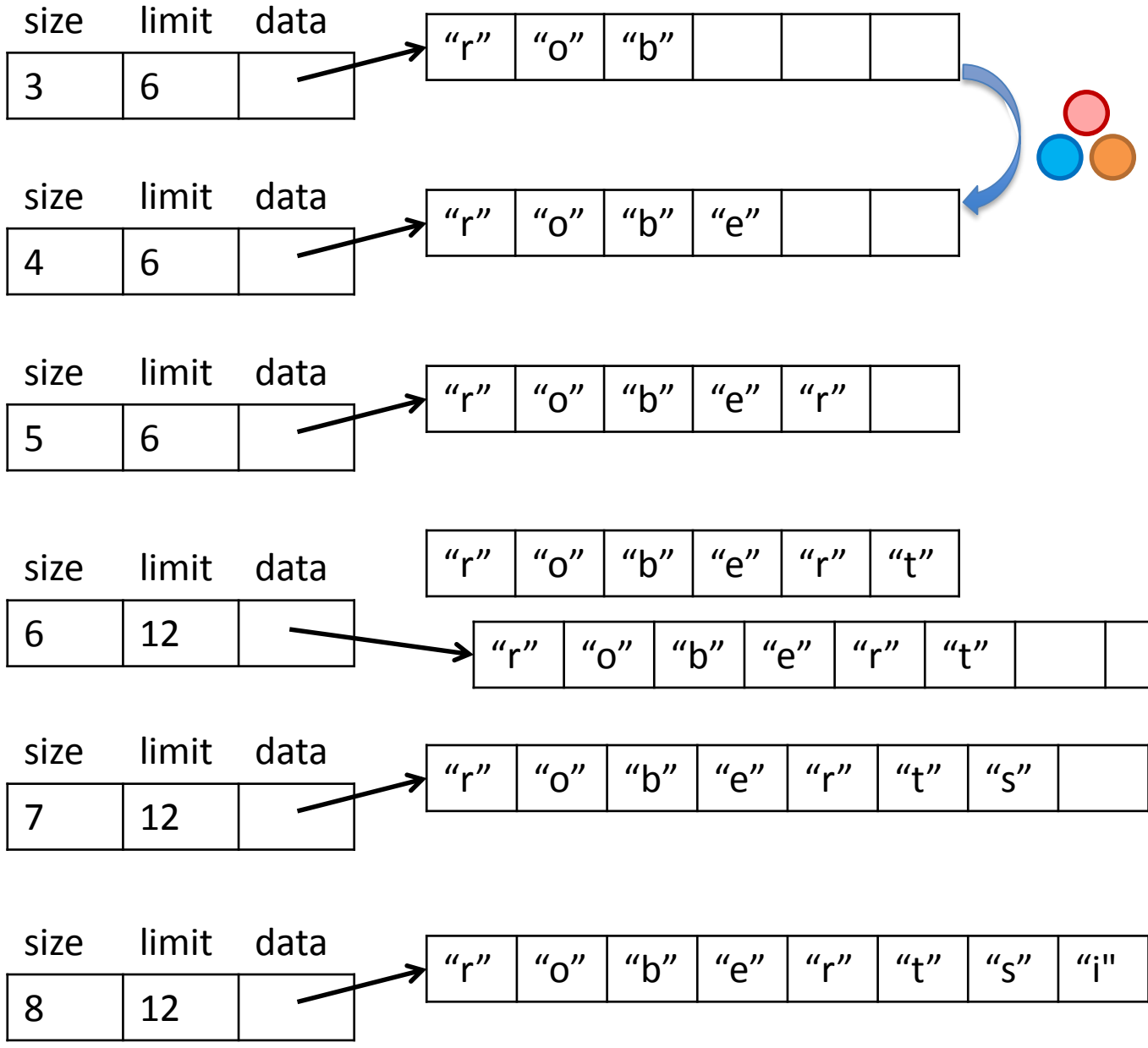



total cost
cost of this add
size limit data
1 2
"a"
3 3
size limit data
2 4
"a" "b"
"a" "b"
4 1
size limit data
3 4
"a" "b" "c"
9 5
size limit data
4 8
"a" "b" "c" "d"
"a" "b" "c" "d"
10 1
size limit data
5 8
"a" "b" "c" "d" "e"
11 1
size limit data
6 8
"a" "b" "c" "d" "e" "f"
12 1
size limit data
7 8
"a" "b" "c" "d" "e" "f" "g"
21 9
size limit data
8 16
"a" "b" "c" "d" "e" "f" "g" "h"
"a" "b" "c" "d" "e" "f" "g" "h"
22 1
size limit data
9 16
"a" "b" "c" "d" "e" "f" "g" "h" "I"

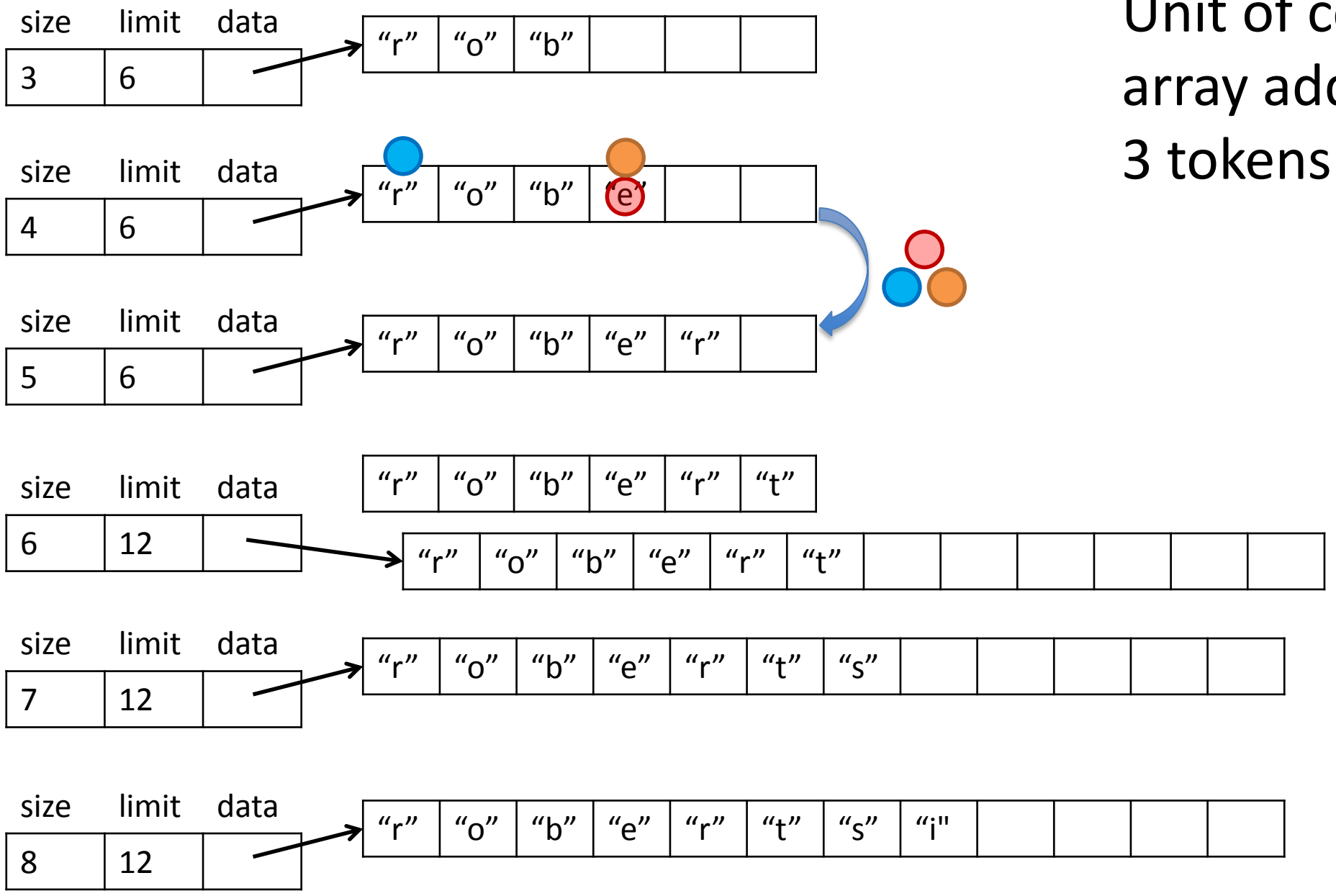

# of adds
total cost
cost of this add
size limit data
1 2
"a"
1 3 3
size limit data
2 4
"a" "b"
"a" "b"
2 4 1
size limit data
3 4
"a" "b" "c"
3 9 5
size limit data
4 8
"a" "b" "c" "d"
"a" "b" "c" "d"
4 10 1
size limit data
5 8
"a" "b" "c" "d" "e"
5 11 1
size limit data
6 8
"a" "b" "c" "d" "e" "f"
6 12 1
size limit data
7 8
"a" "b" "c" "d" "e" "f" "g"
7 21 9
size limit data
8 16
"a" "b" "c" "d" "e" "f" "g" "h"
"a" "b" "c" "d" "e" "f" "g" "h"
8 22 1
size limit data
9 16
"a" "b" "c" "d" "e" "f" "g" "h" "I"


bank account
total tokens
total cost
cost of this add
size limit data
1 2 "a"
3 3 3
size limit data
2 4 "a" "b"
6 4 1
size limit data
3 4 "a" "b" "c"
9 9 5
size limit data
4 8 "a" "b" "c" "d"
12 10 1
size limit data
5 8 "a" "b" "c" "d" "e"
15 11 1
size limit data
6 8 "a" "b" "c" "d" "e" "f"
18 12 1
size limit data
7 8 "a" "b" "c" "d" "e" "f" "g"
21 21 9
size limit data
8 16 "a" "b" "c" "d" "e" "f" "g" "h"
24 22 1
size limit data
9 16 "a" "b" "c" "d" "e" "f" "g" "h" "I"

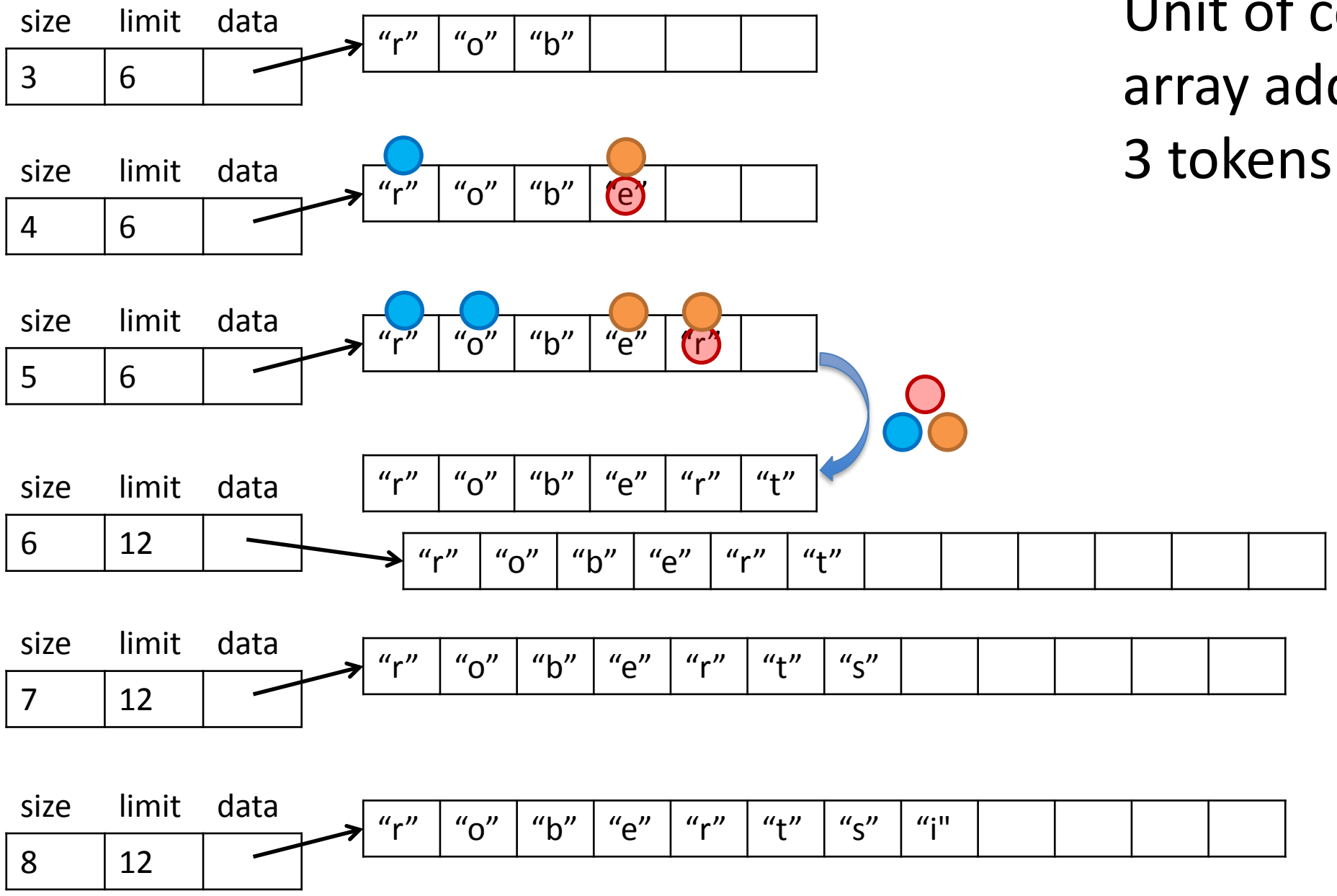

bank account
total tokens
total cost
cost of this add
size limit data
1 2 "a"
3 3 3
size limit data
2 4 "a" "b"
6 4 1
size limit data
3 4 "a" "b" "c"
9 9 5
size limit data
4 8 "a" "b" "c" "d"
12 10 1
size limit data
5 8 "a" "b" "c" "d" "e"
15 11 1
size limit data
6 8 "a" "b" "c" "d" "e" "f"
18 12 1
size limit data
7 8 "a" "b" "c" "d" "e" "f" "g"
21 21 9
size limit data
8 16 "a" "b" "c" "d" "e" "f" "g" "h"
24 22 1
size limit data
9 16 "a" "b" "c" "d" "e" "f" "g" "h" "I"



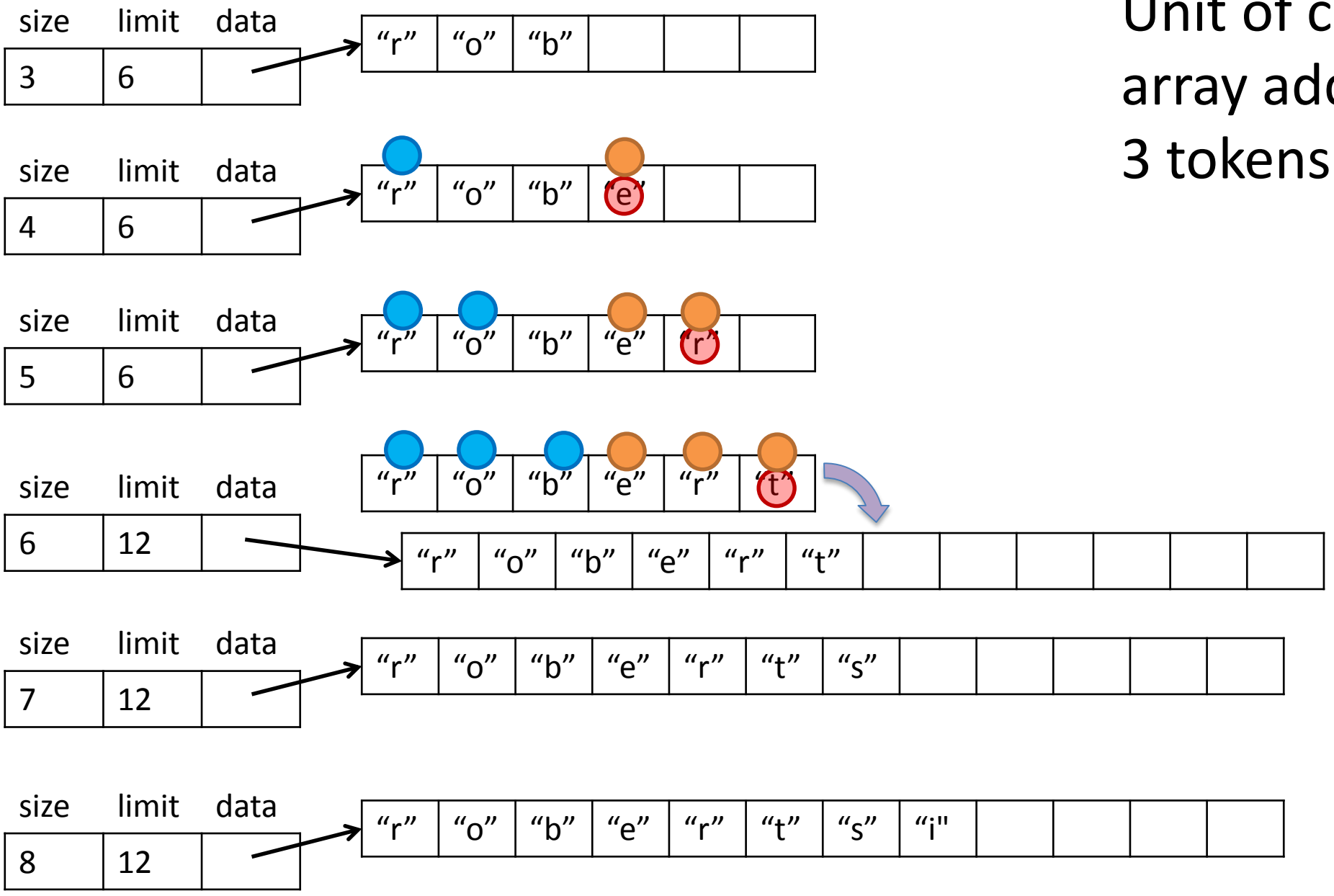
Unit of cost:
array add
3 tokens



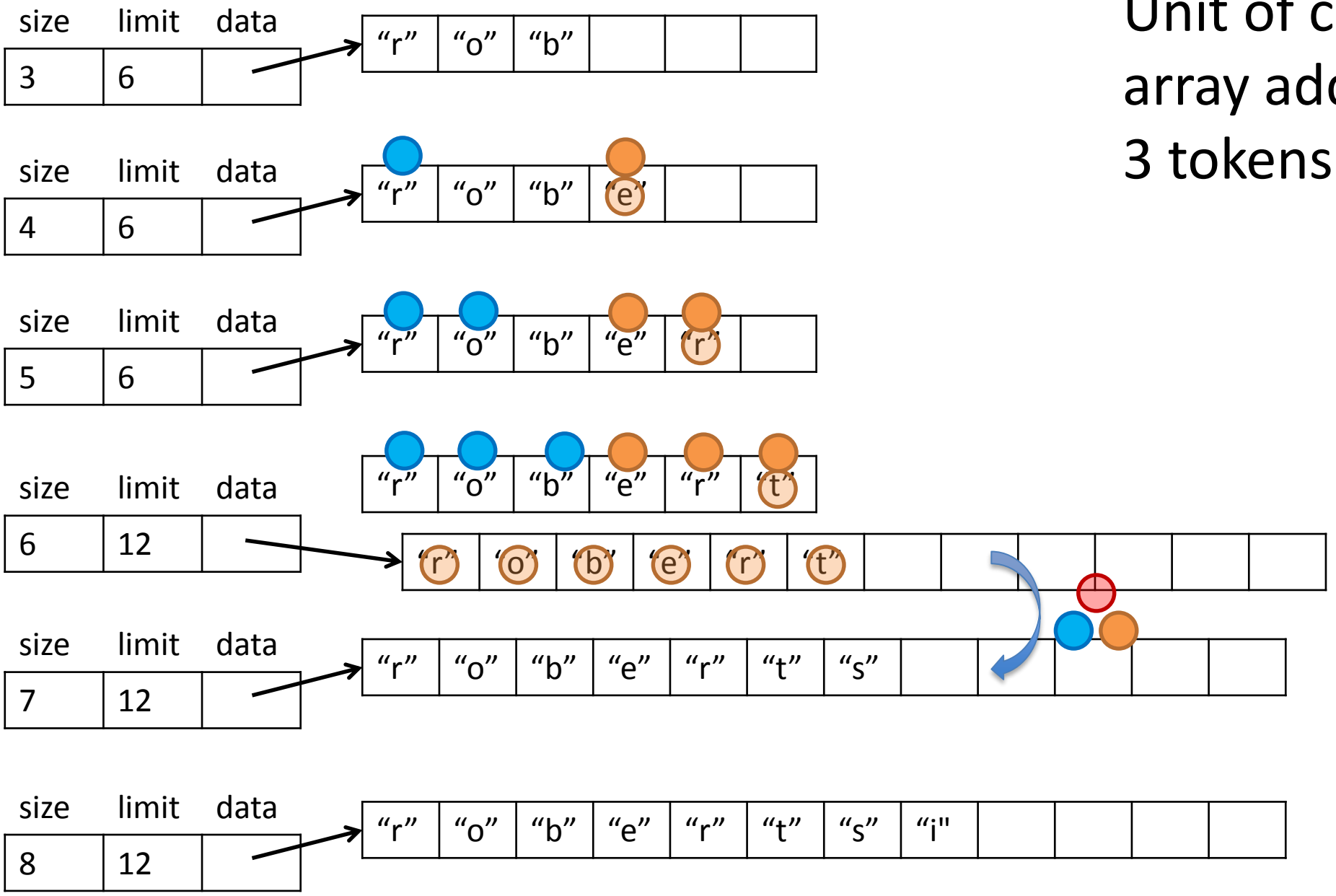
Unit of cost:
array add
3 tokens



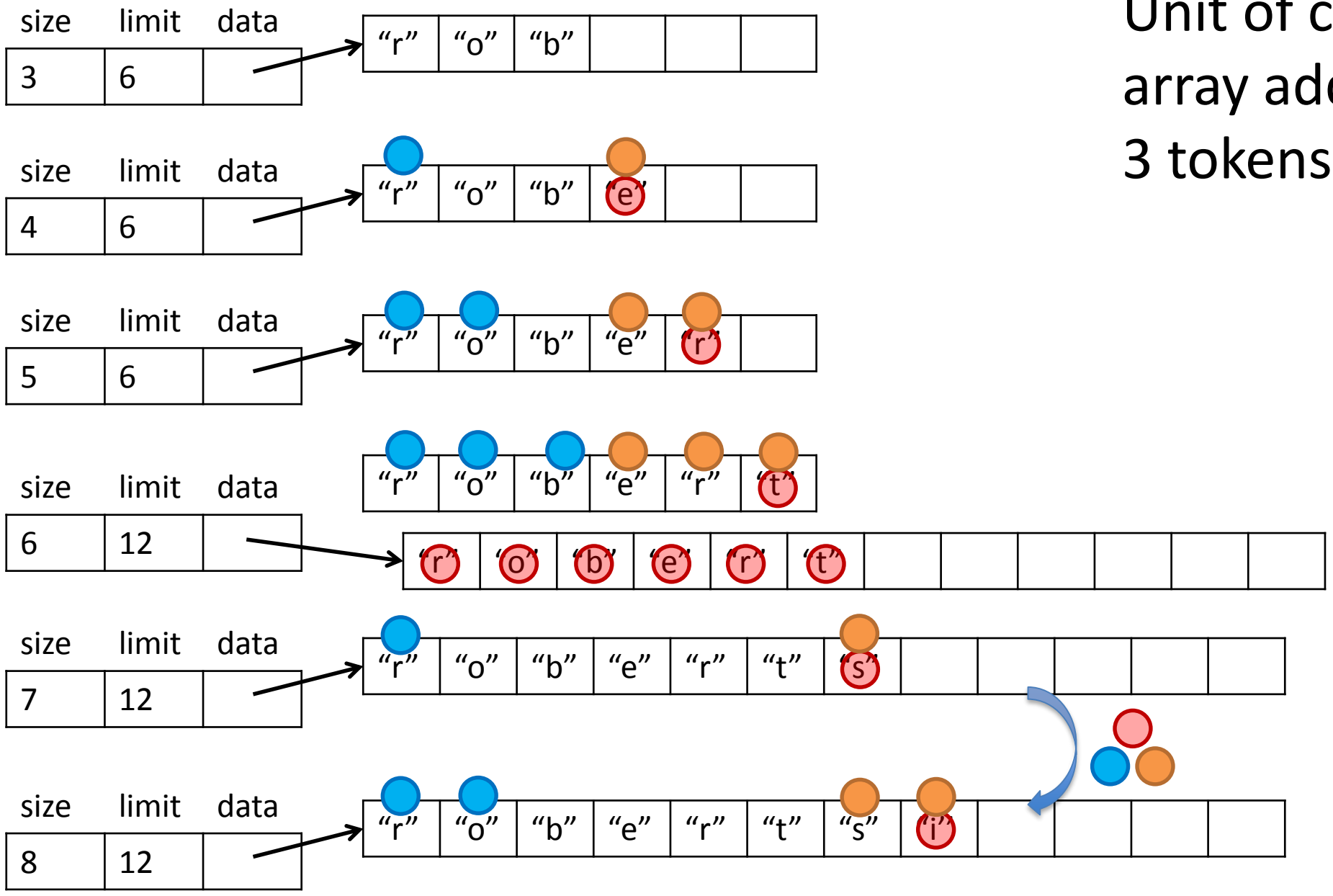
Unit of cost:
array add
3 tokens



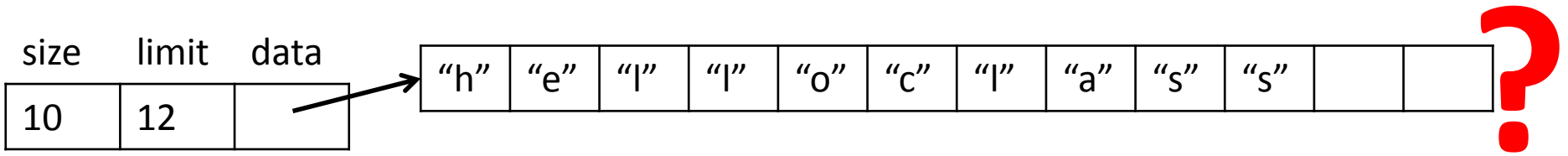
Unit of cost:
array add
3 tokens

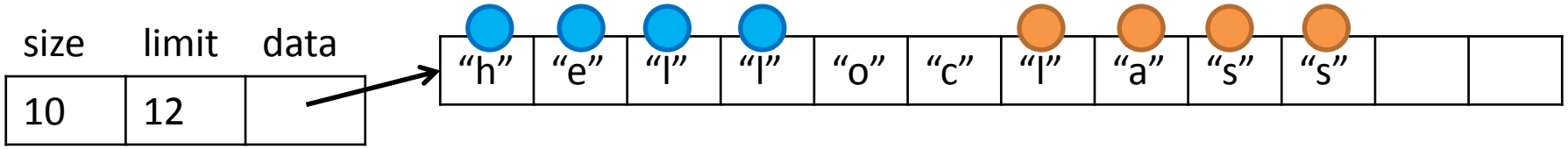


Unit of cost:
array add
3 tokens



Unit of cost:
array add
3 tokens


size
limit
data
10
12
“h”
“e”
“l”
“l”
“o”
“c”
“l”
“a”
“s”
“s”
?


size
limit
data
10
12
“h”
“e”
“l”
“l”
“o”
“c”
“l”
“a”
“s”
“s”