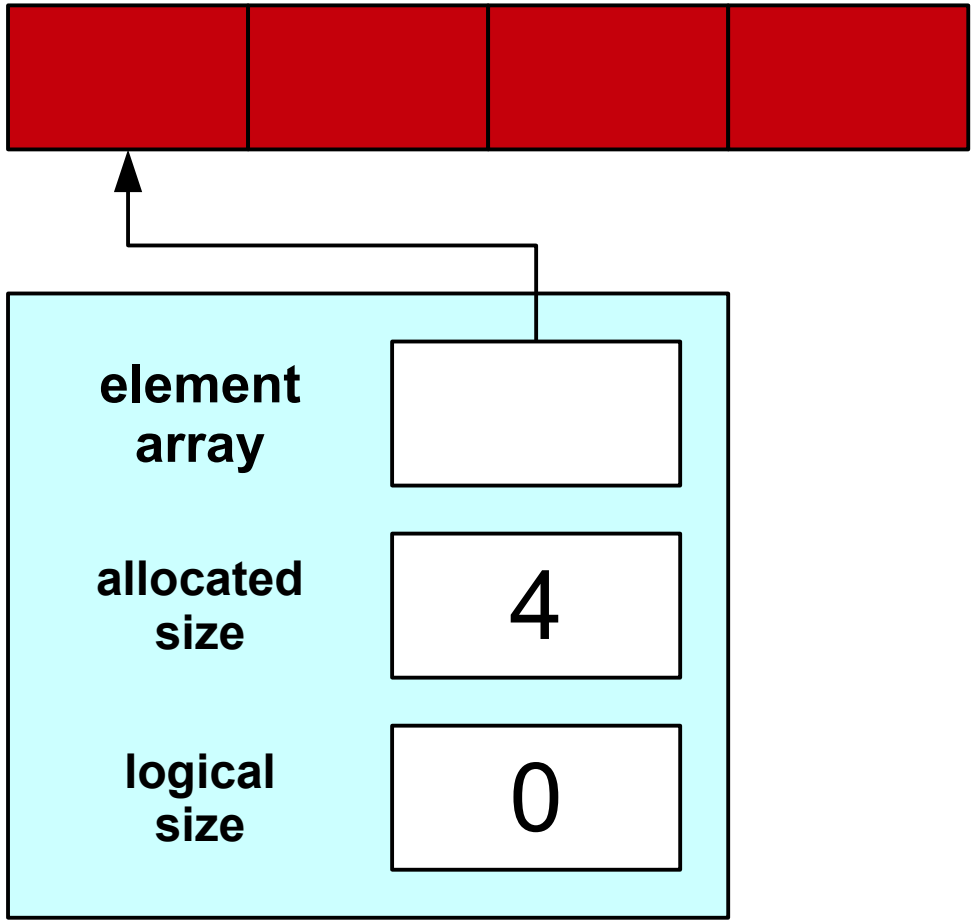
Implementing Abstractions

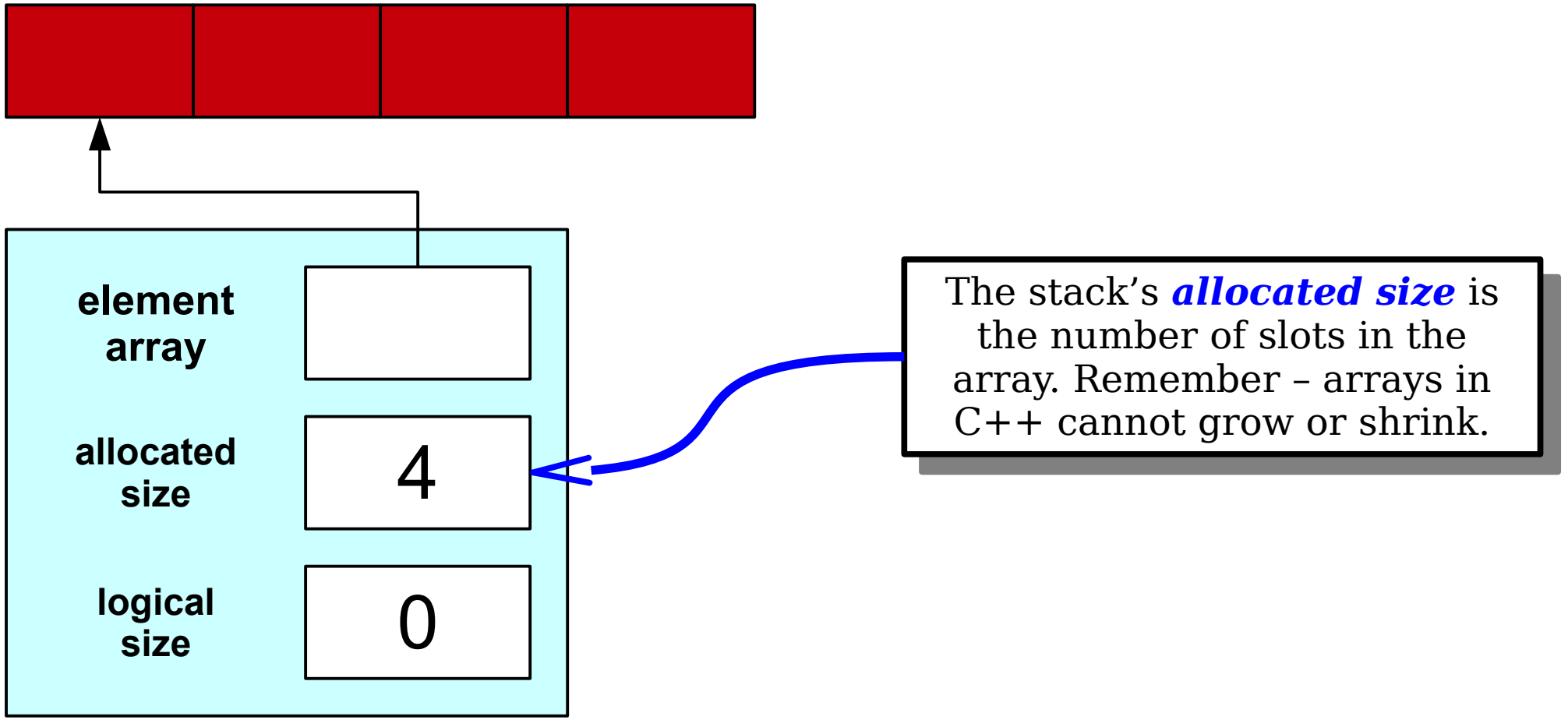
Part Two

Previously, on CS106B...

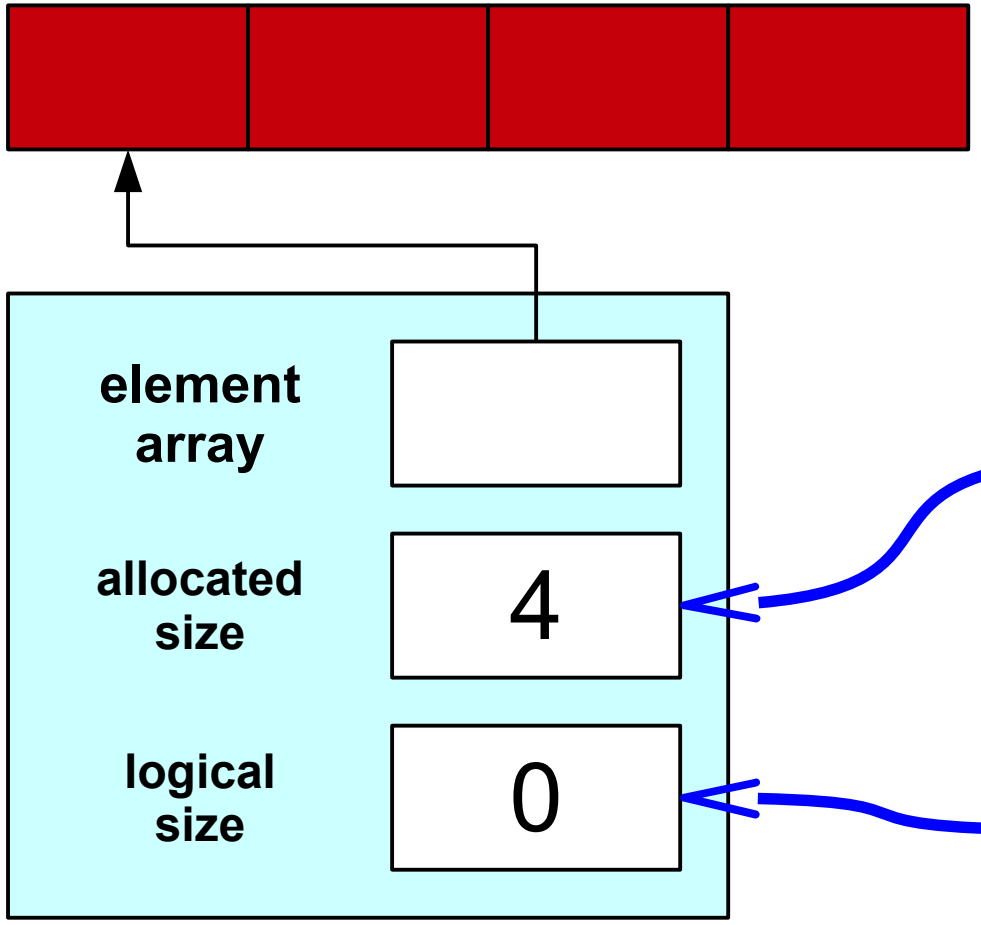
A Bounded Stack



A Bounded Stack



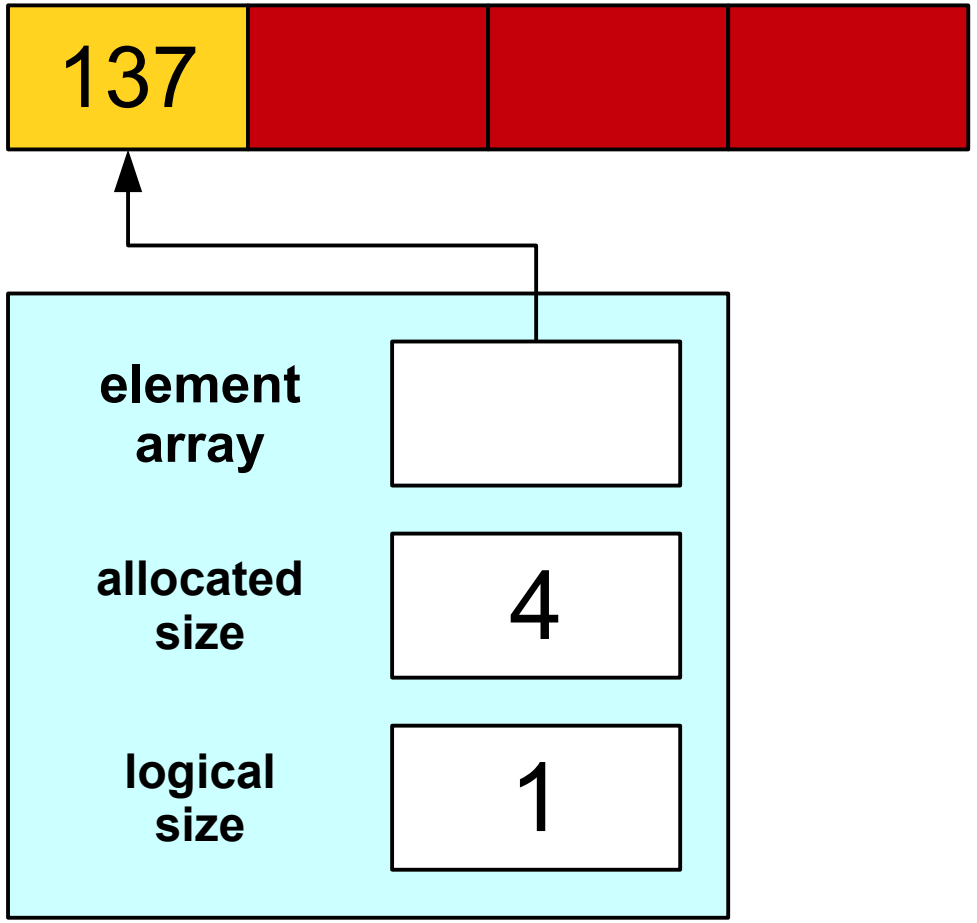
A Bounded Stack



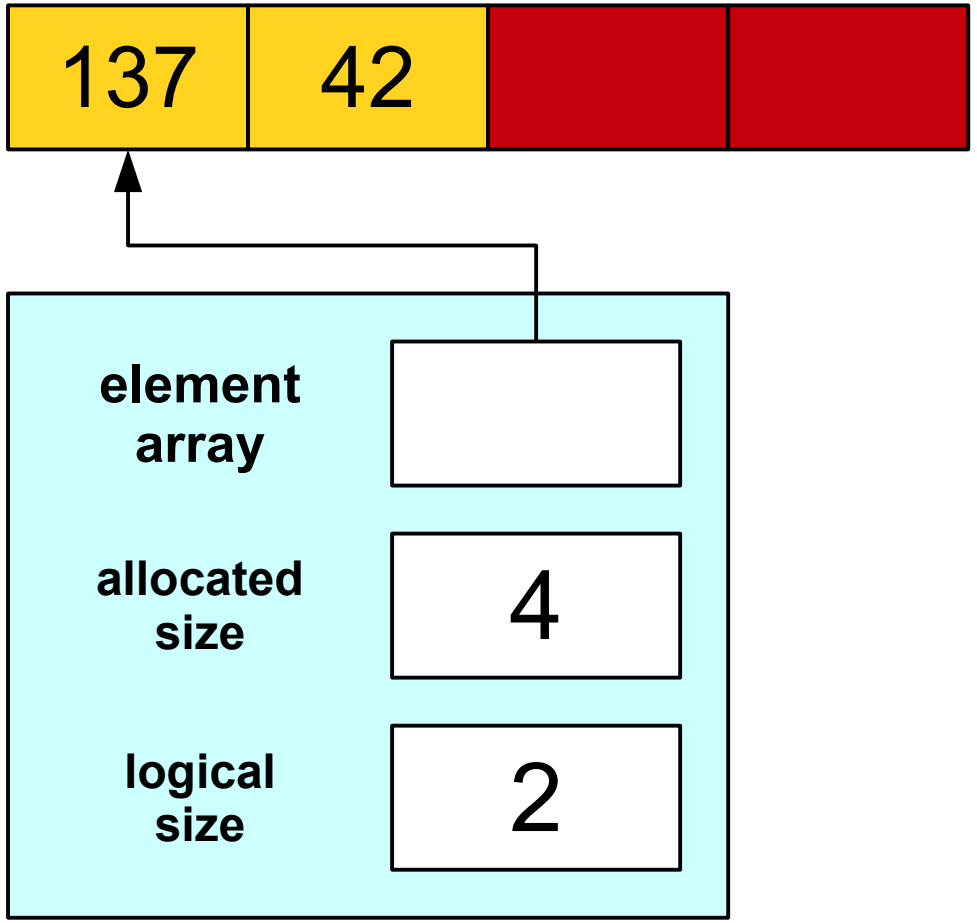
The stack's ***allocated size*** is the number of slots in the array. Remember – arrays in C++ cannot grow or shrink.

The stack's ***logical size*** is the number of elements actually stored in the stack. This lets us track how much space we're actually using.

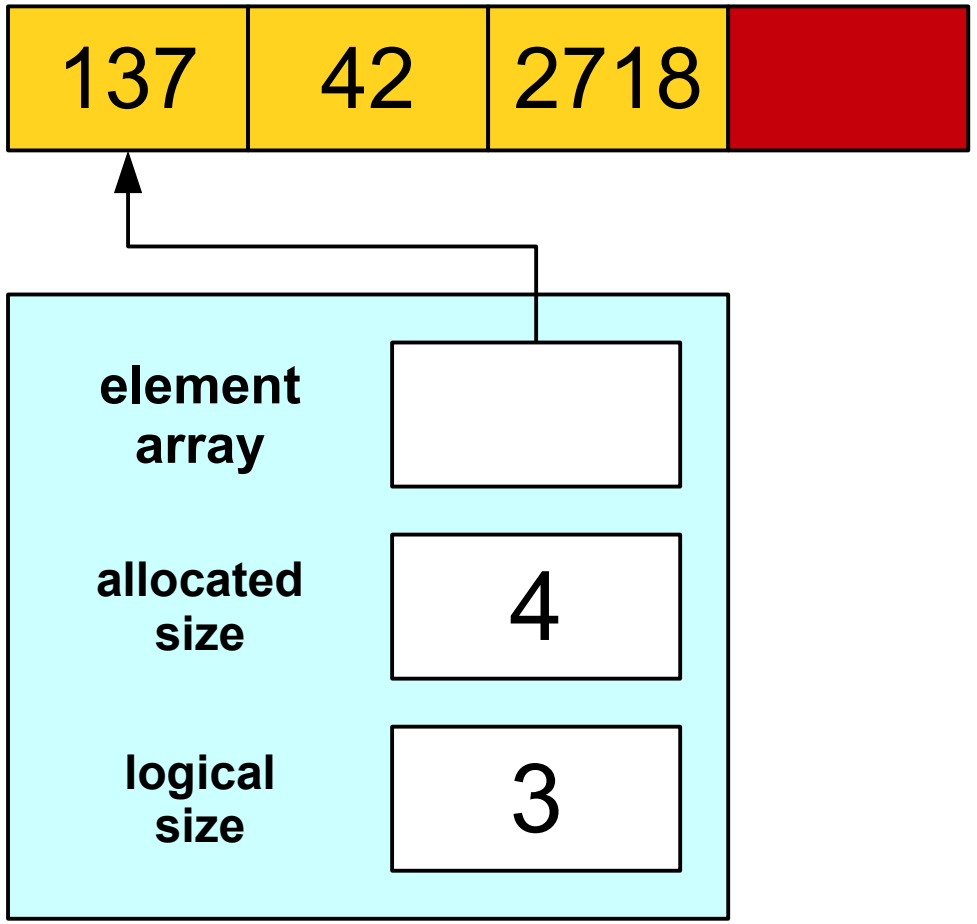
A Bounded Stack



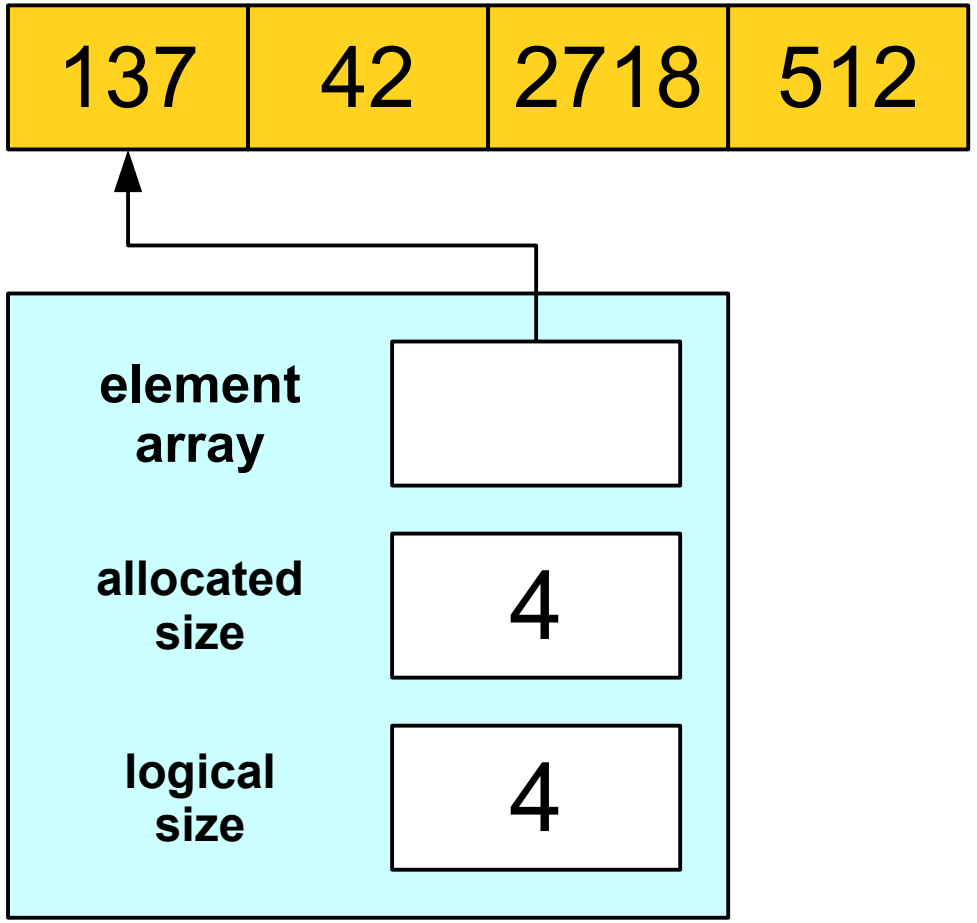
A Bounded Stack



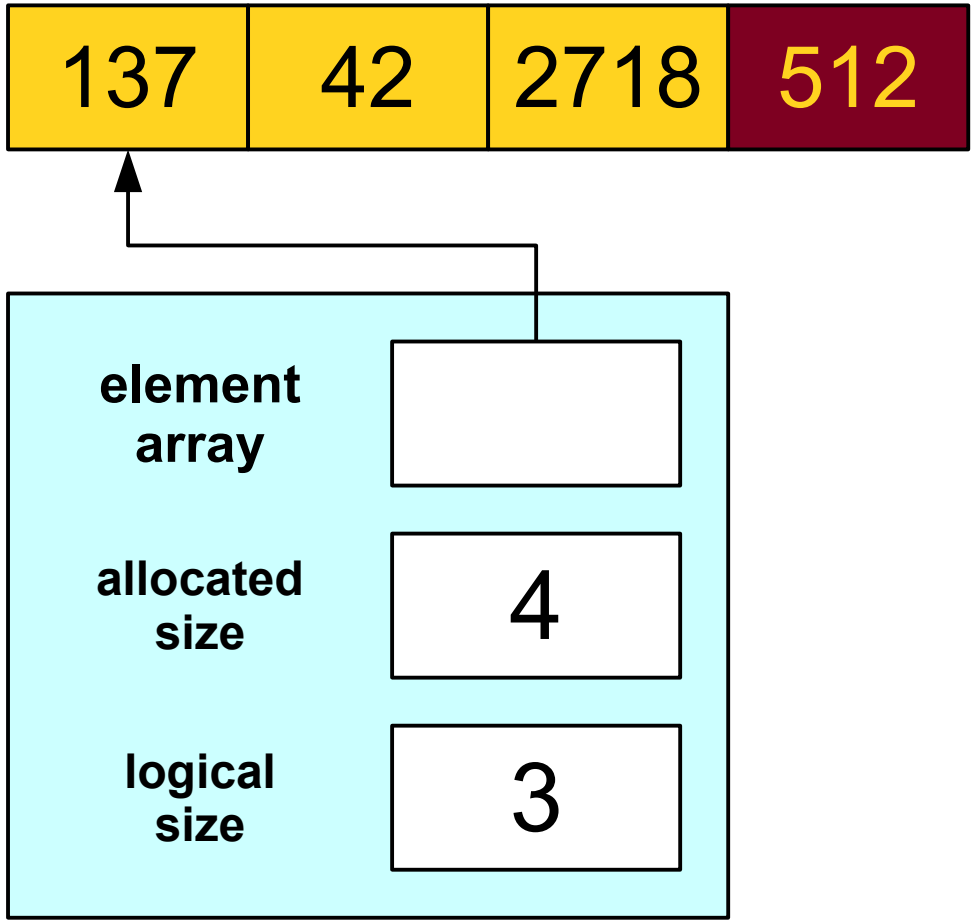
A Bounded Stack



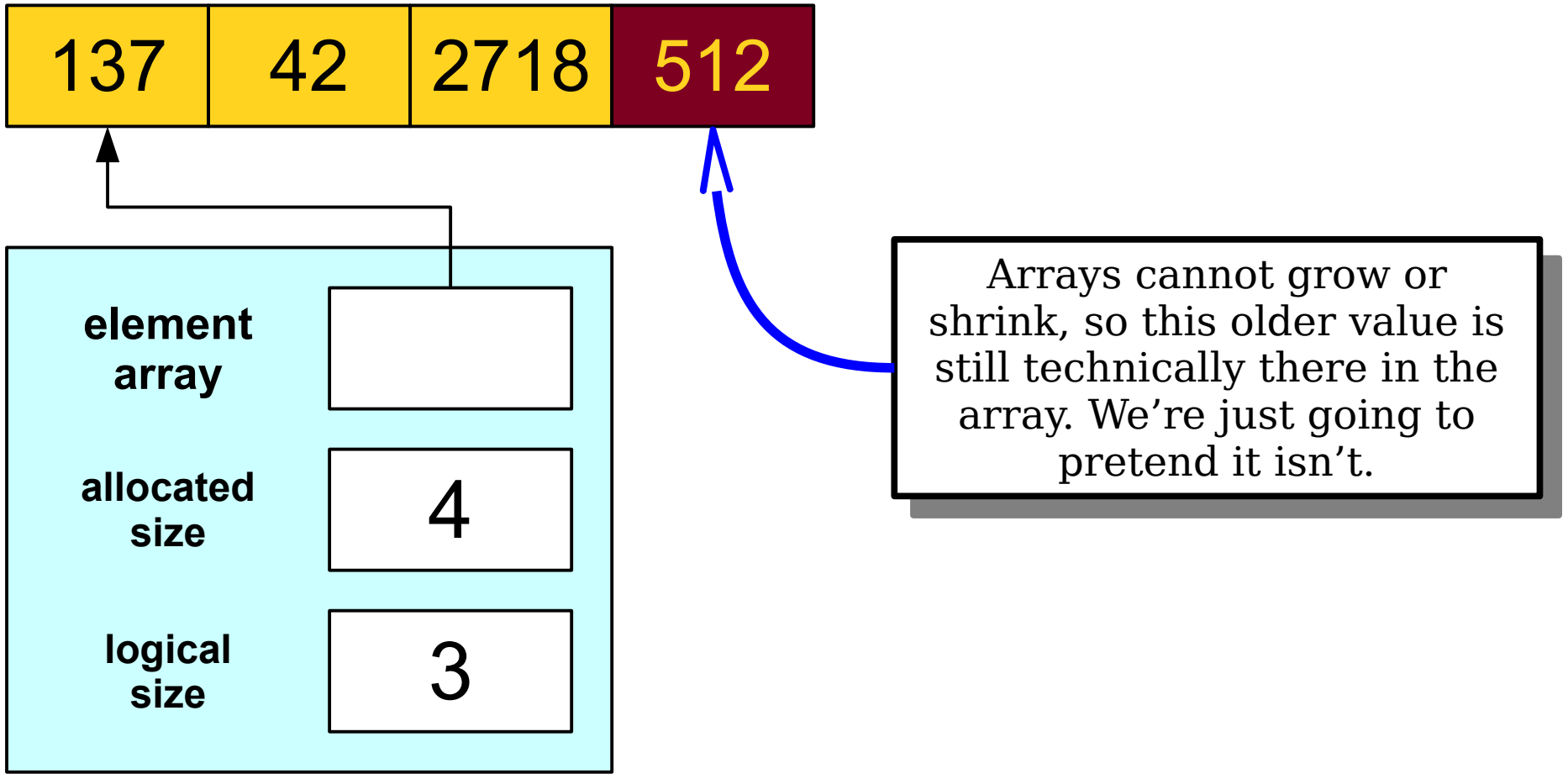
A Bounded Stack



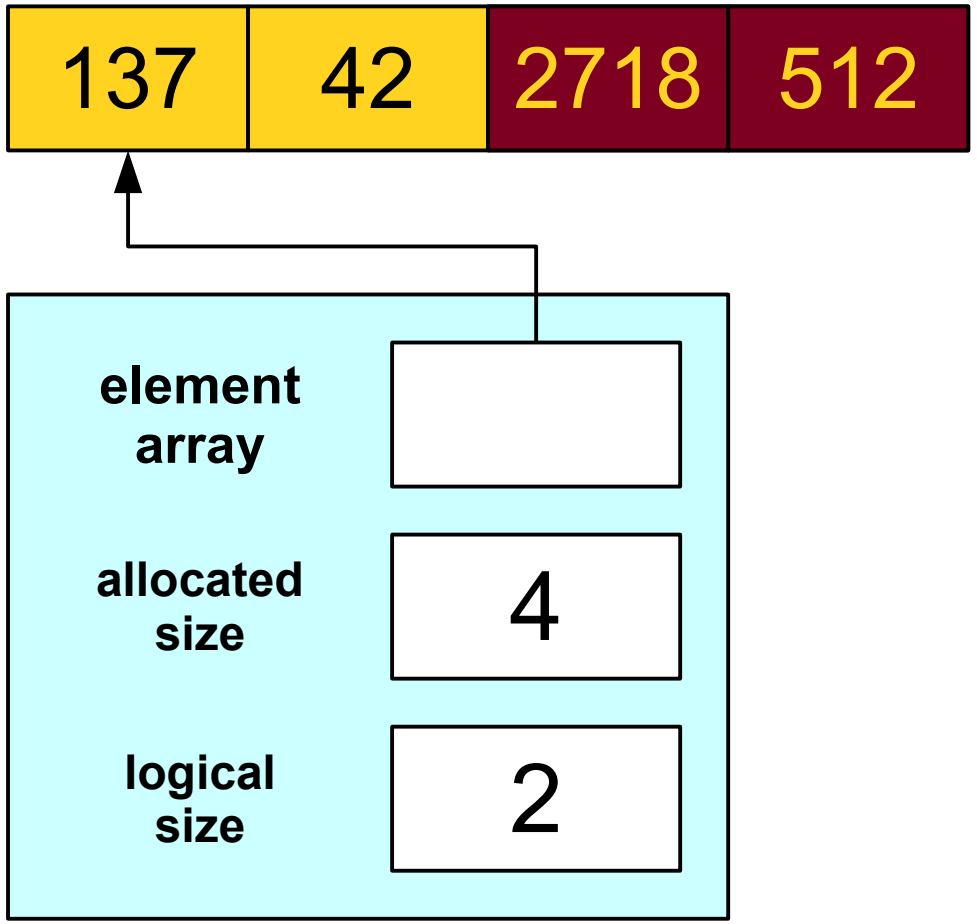
A Bounded Stack



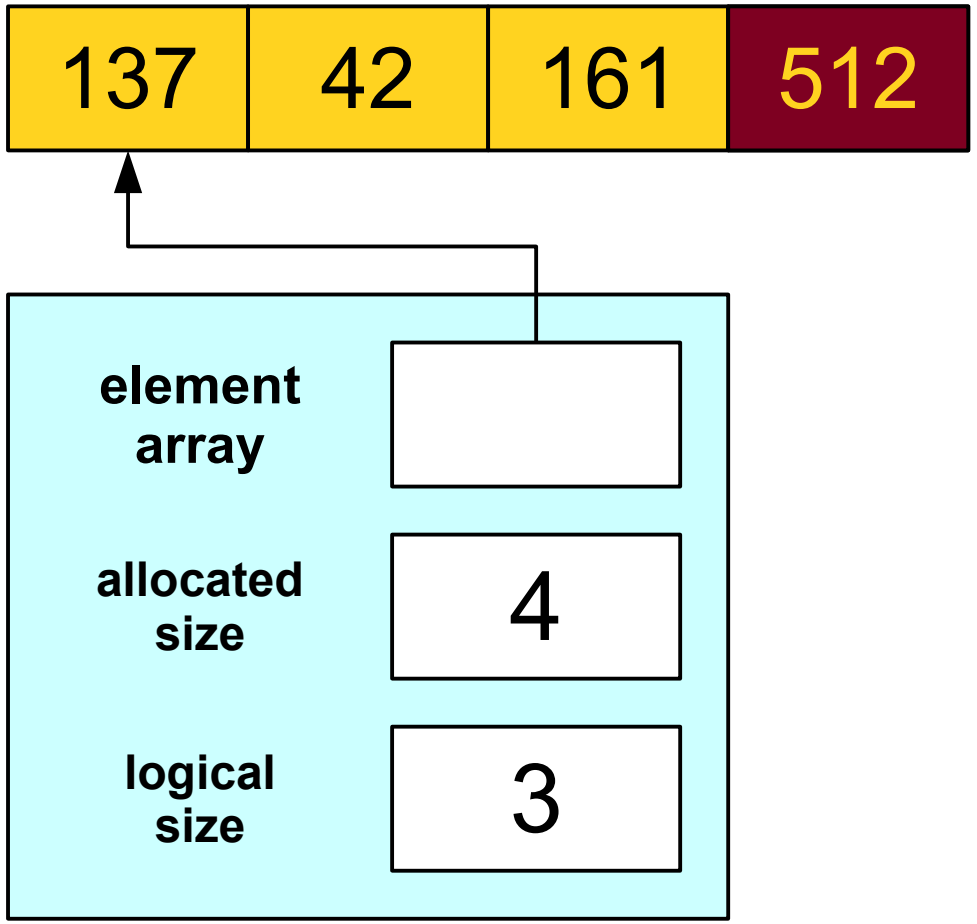
A Bounded Stack



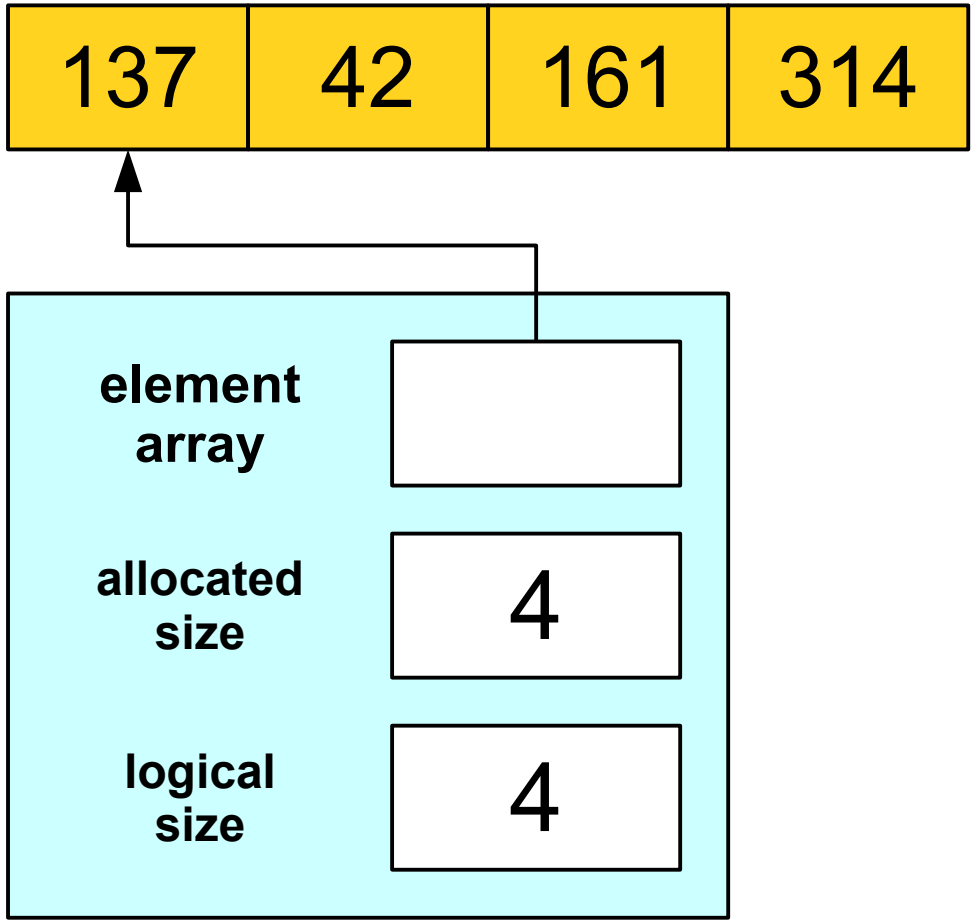
A Bounded Stack



A Bounded Stack



A Bounded Stack

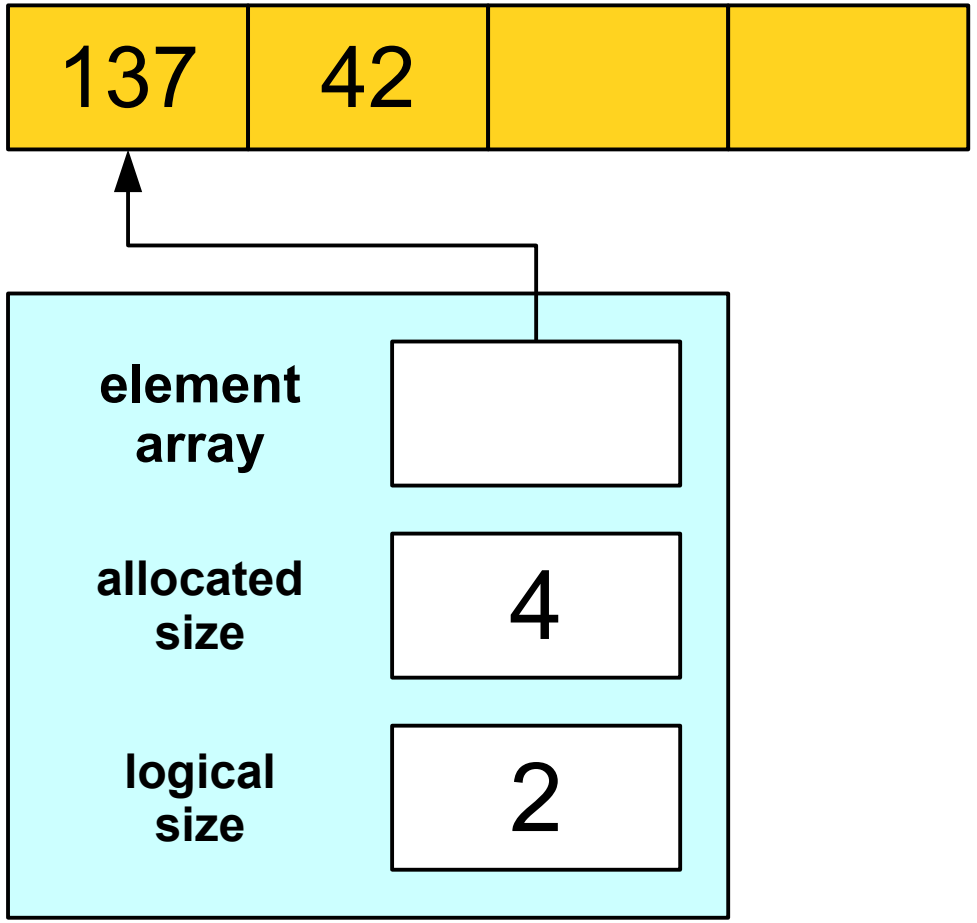


New Stuff!

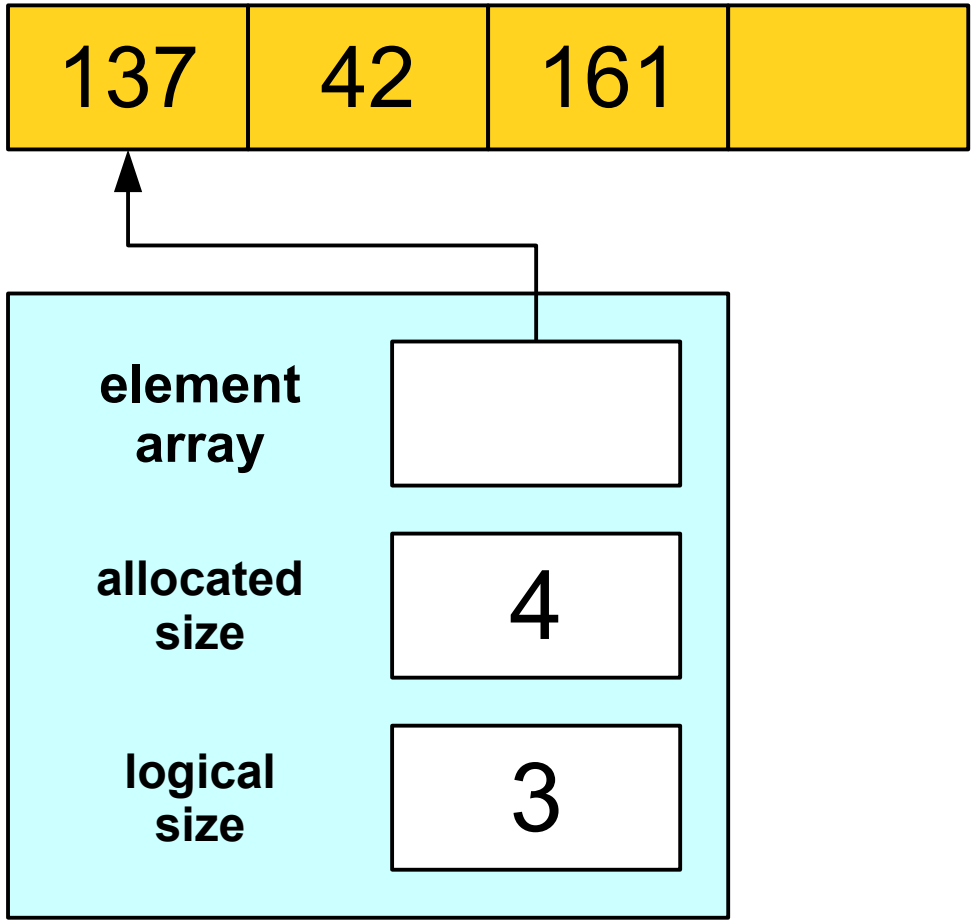
Running out of Space

- Our current implementation very quickly runs out of space to store elements.
- What should we do when this happens?

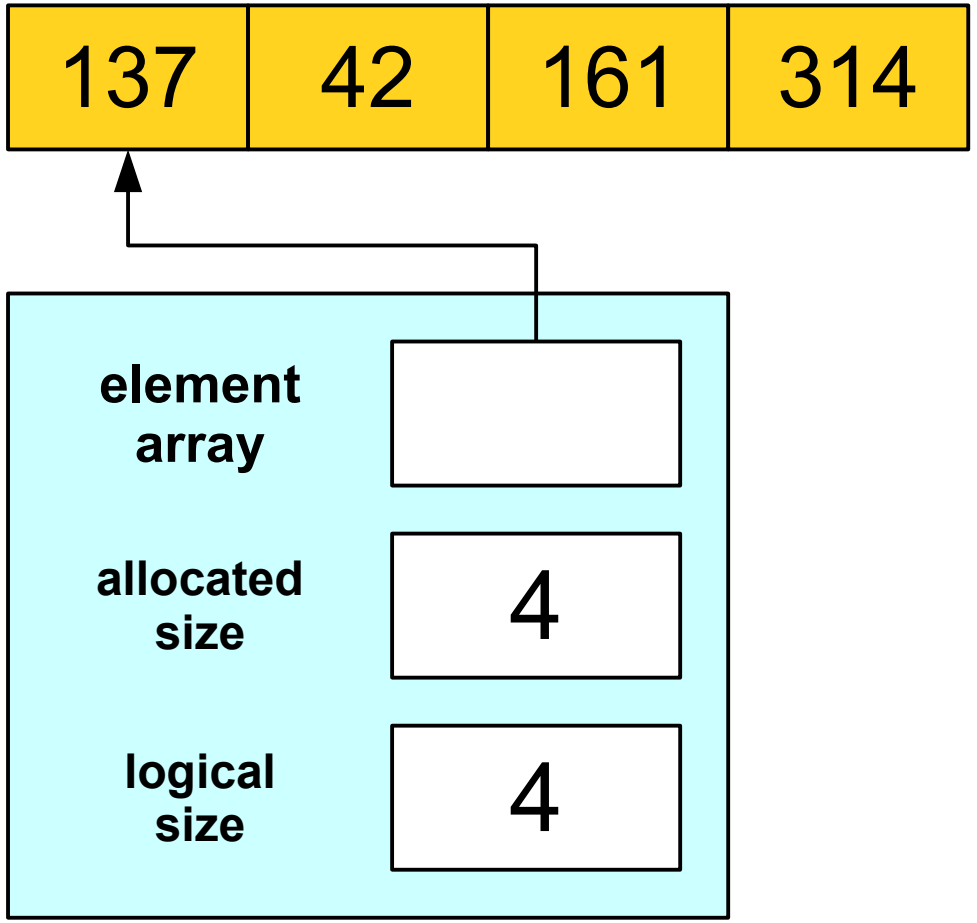
An Initial Idea



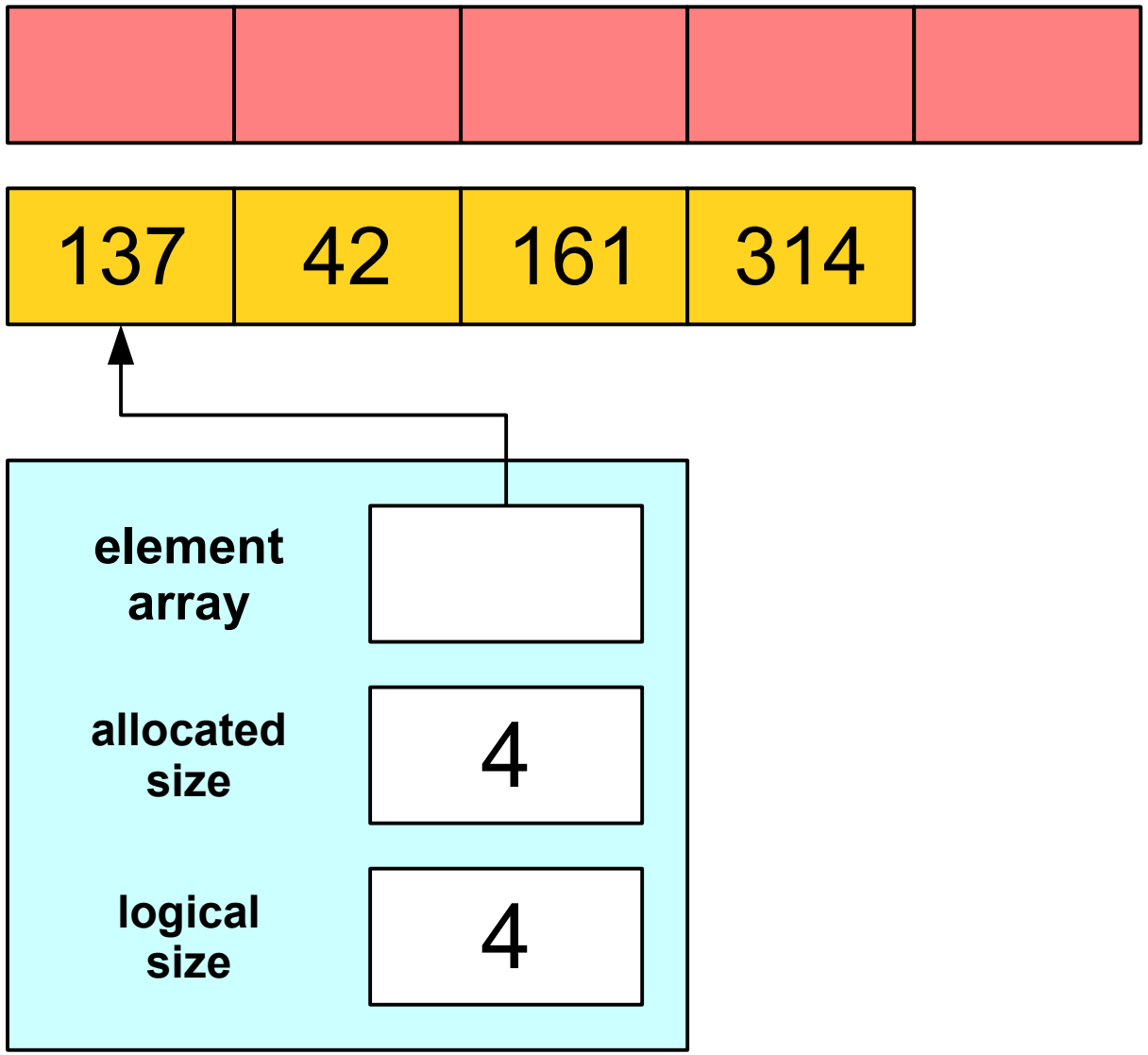
An Initial Idea



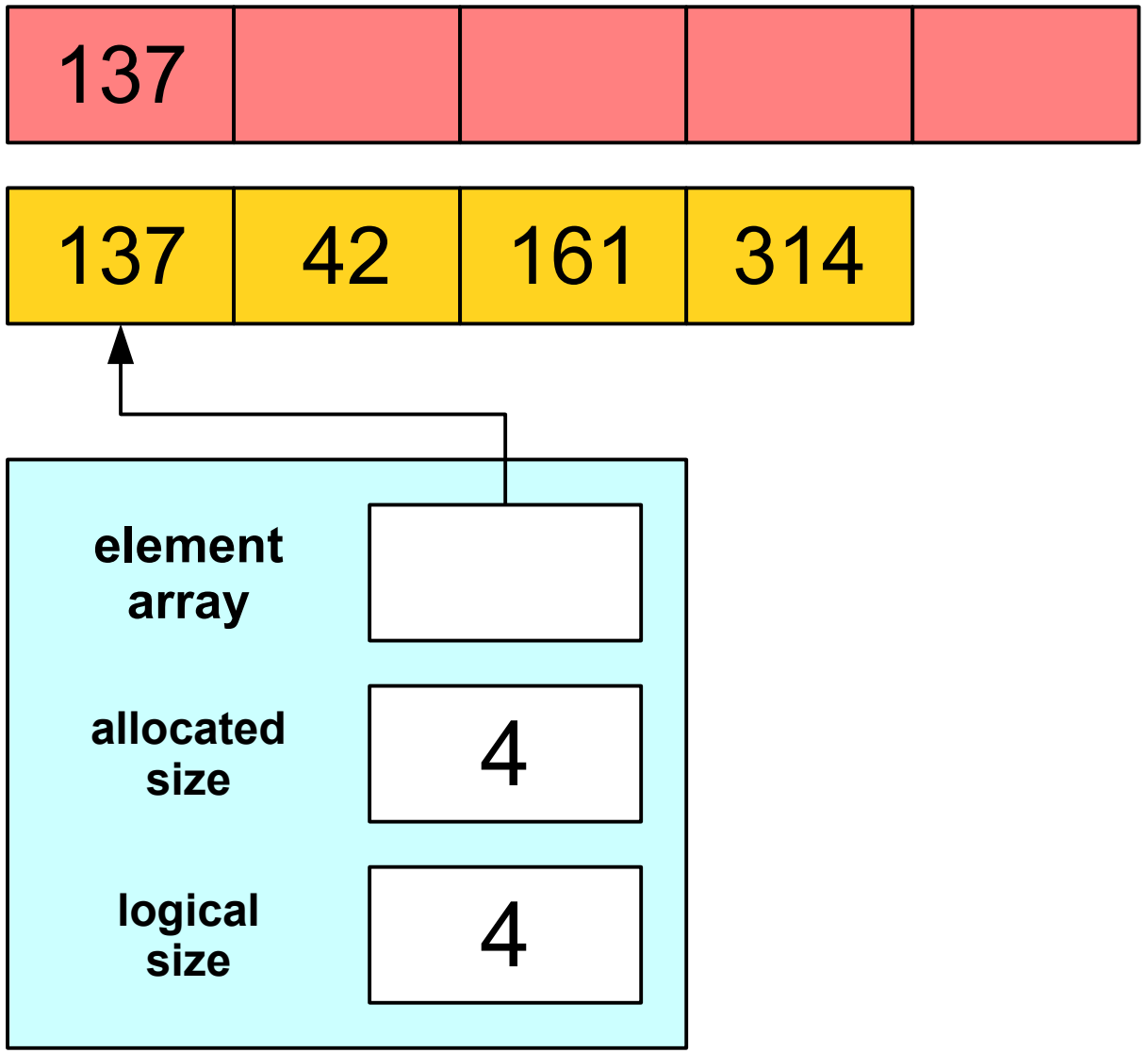
An Initial Idea



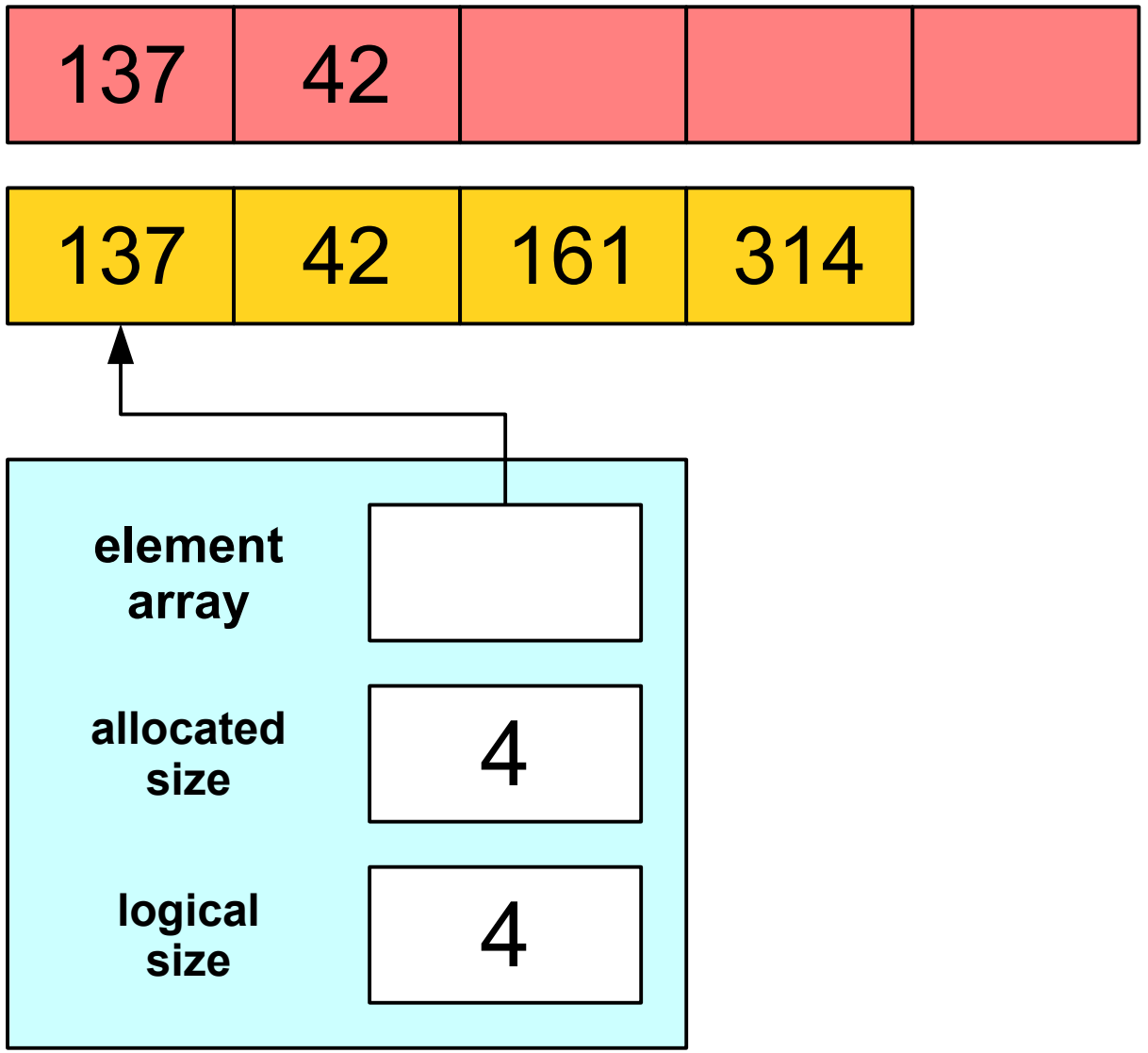
An Initial Idea



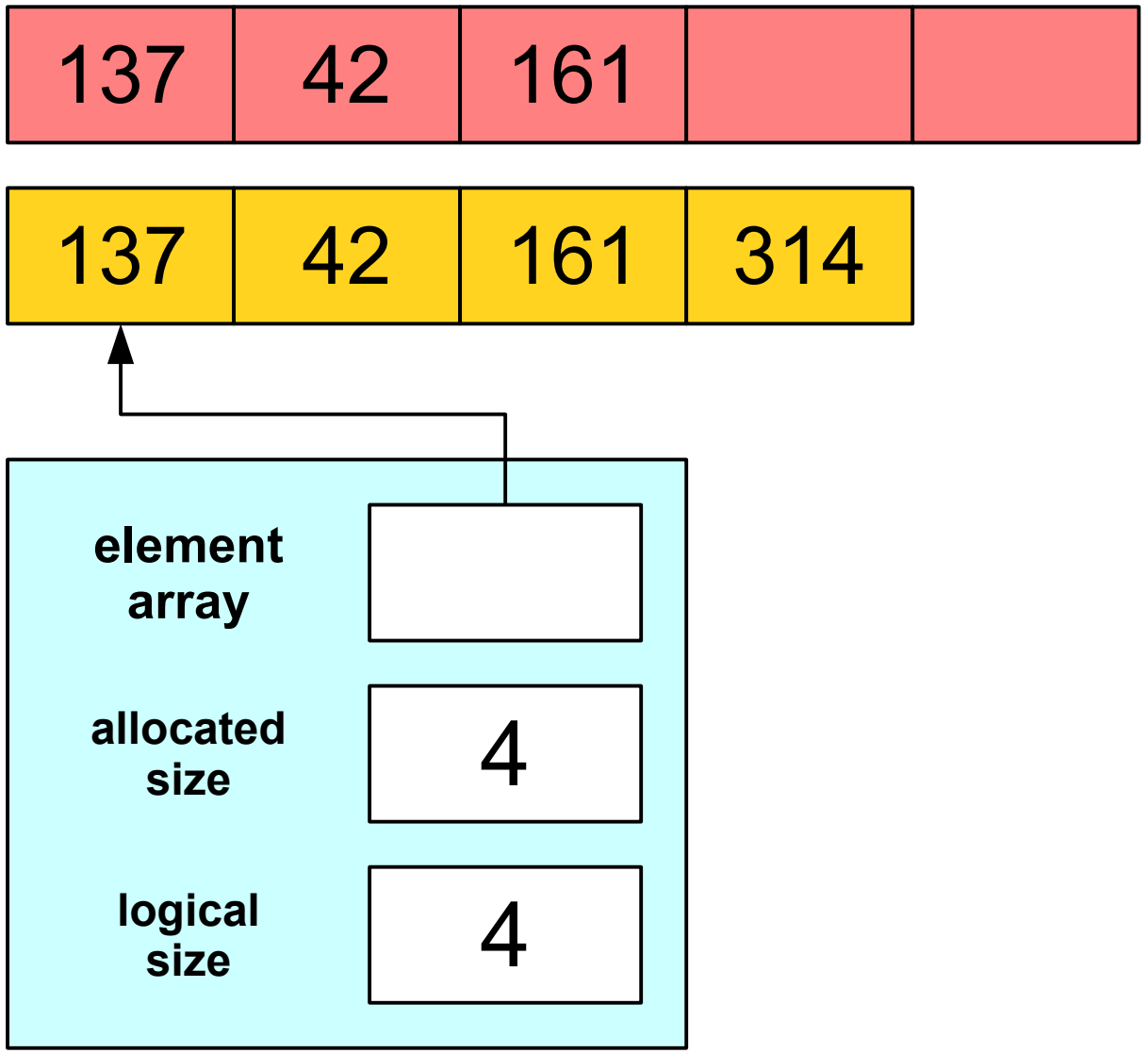
An Initial Idea



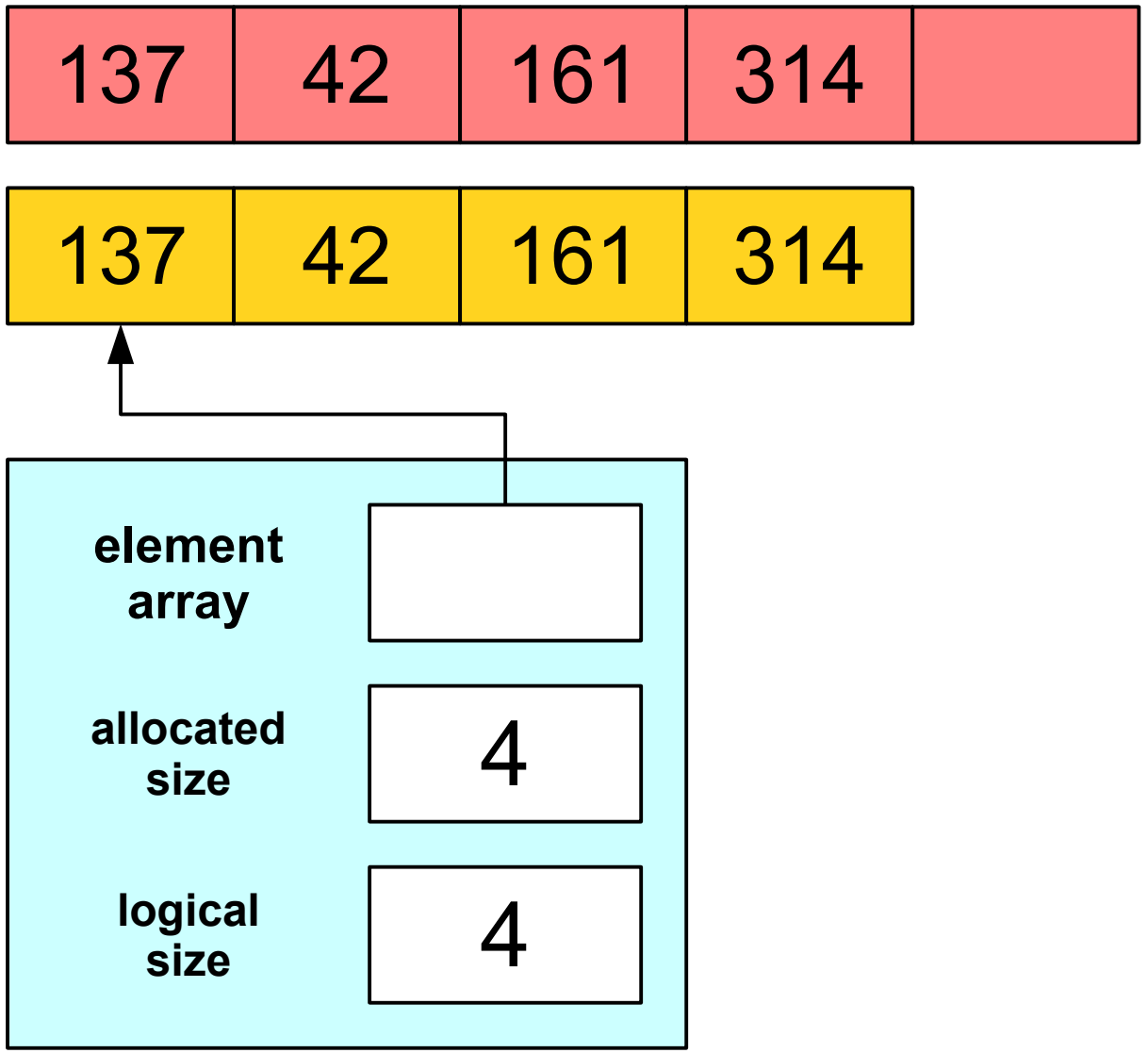
An Initial Idea



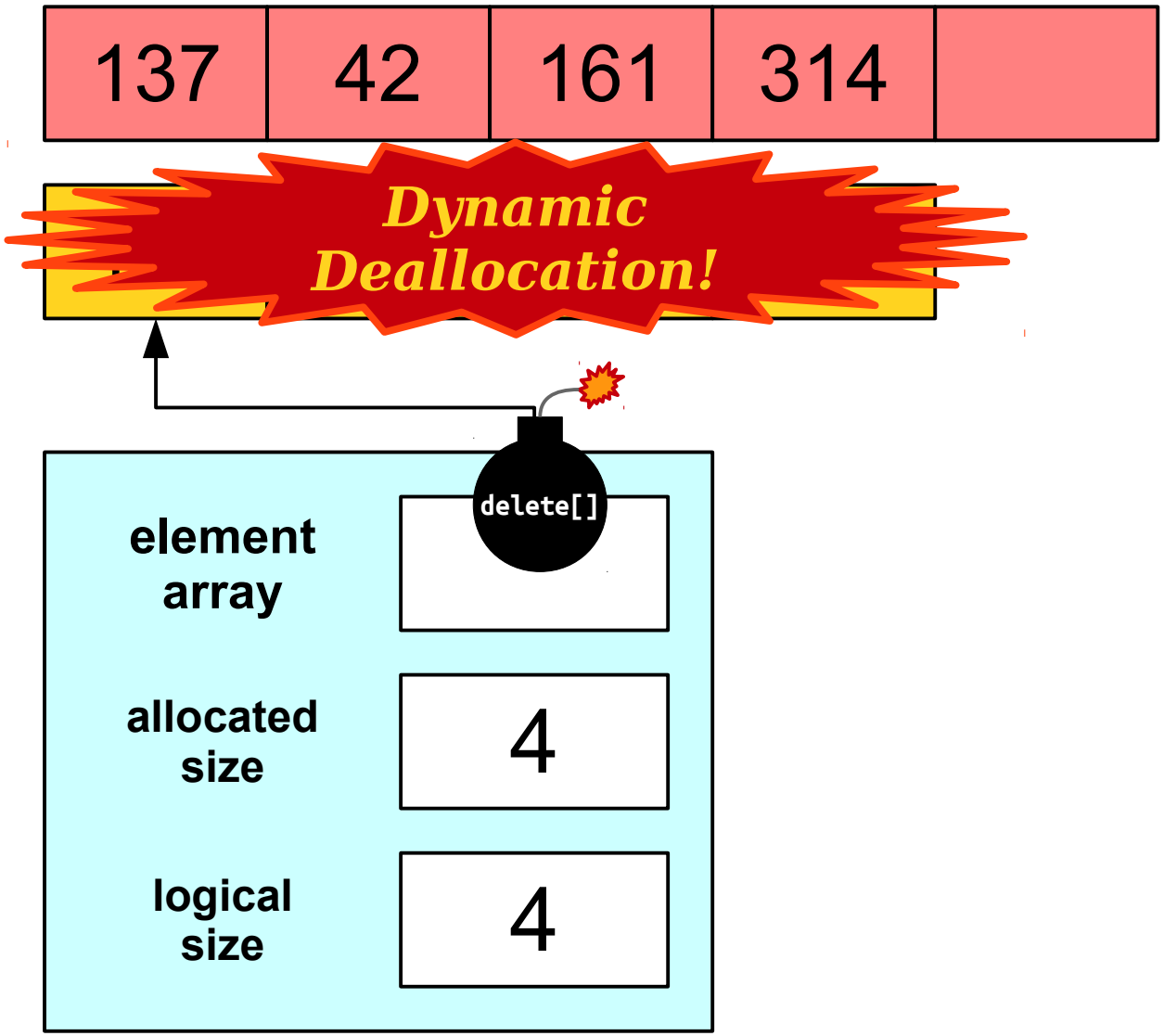
An Initial Idea



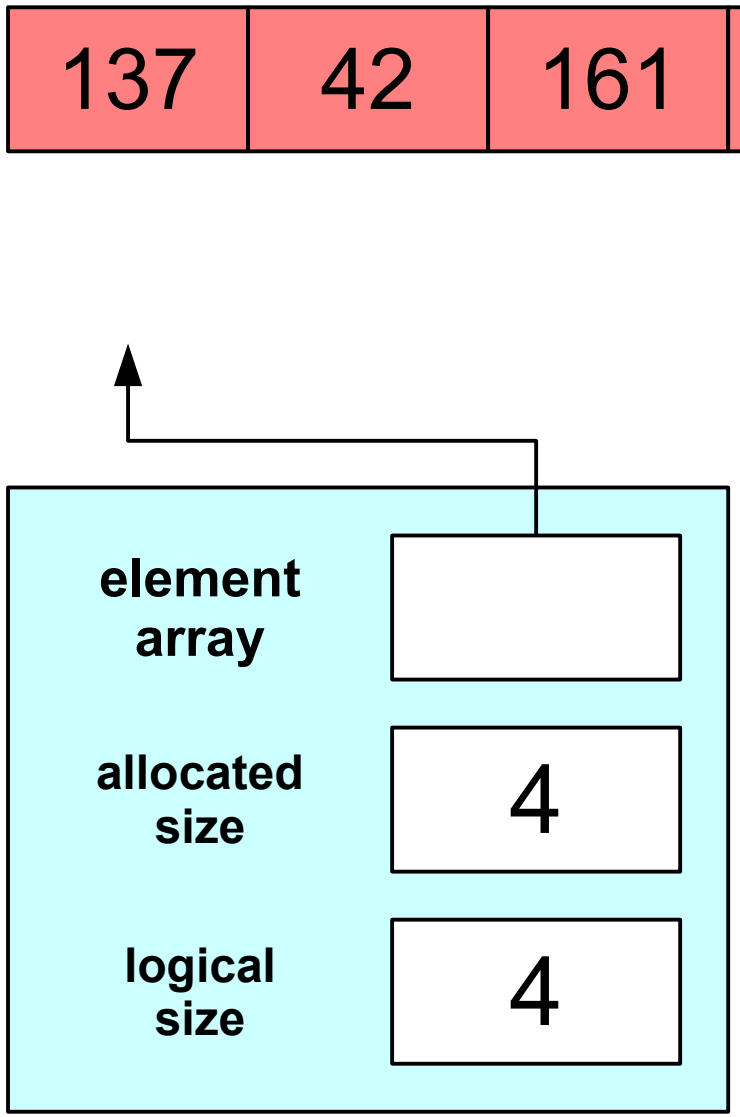
An Initial Idea



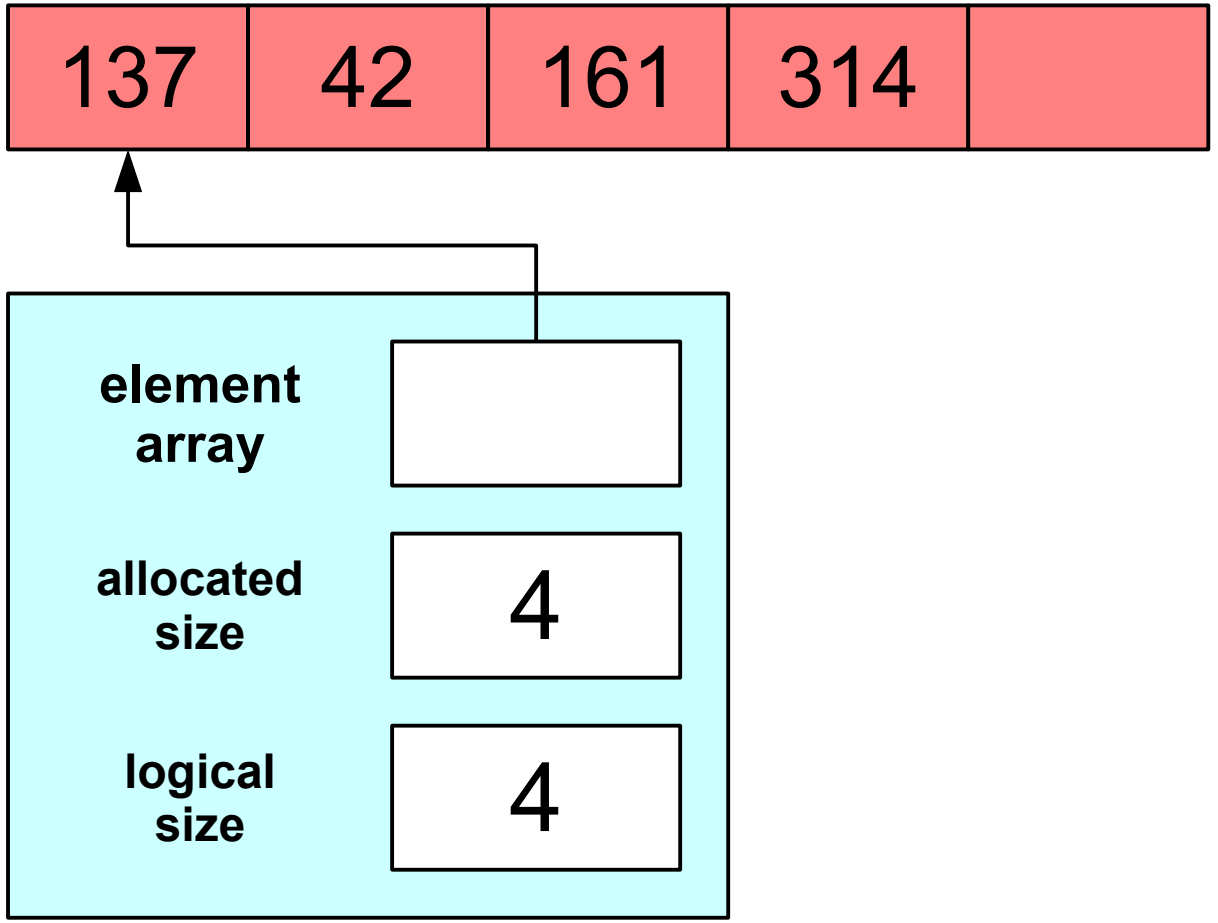
An Initial Idea



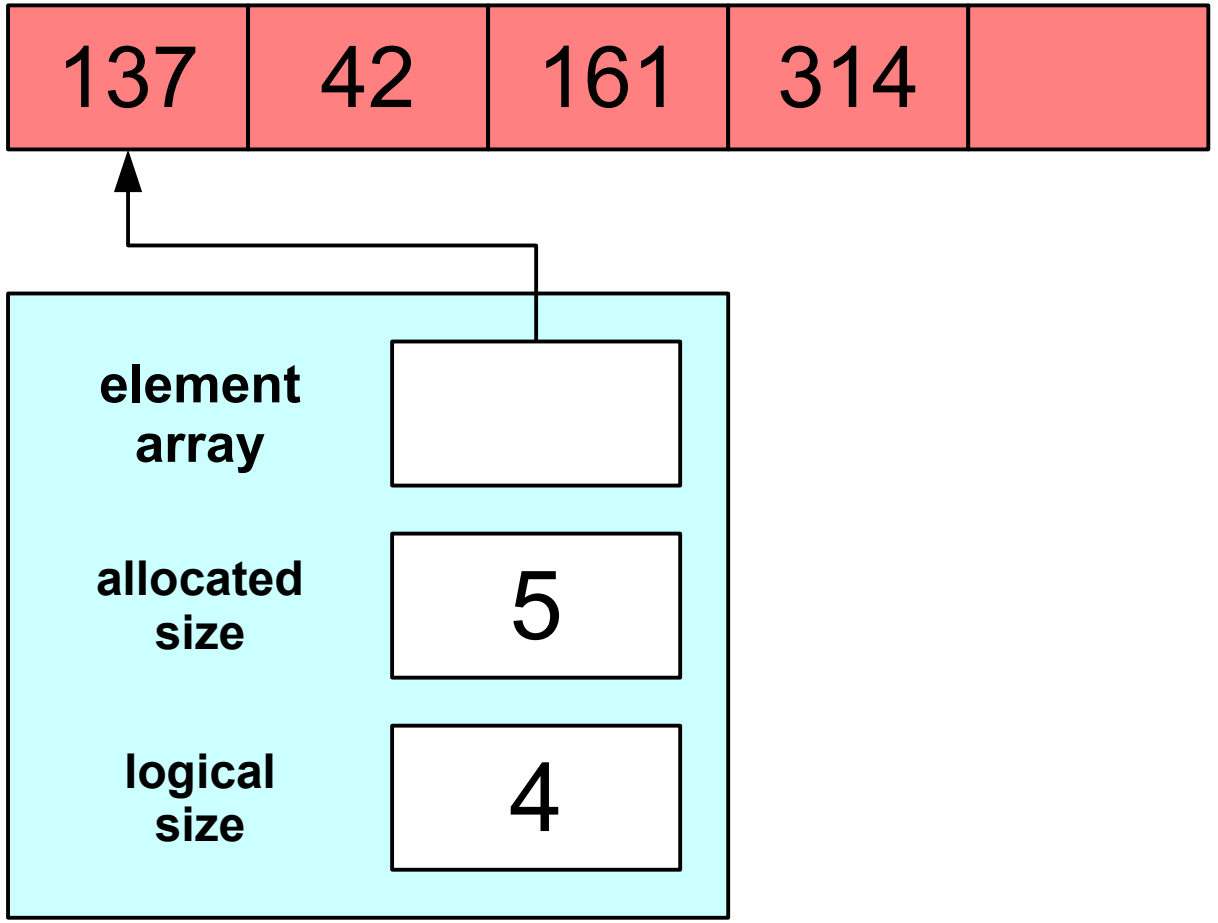
An Initial Idea



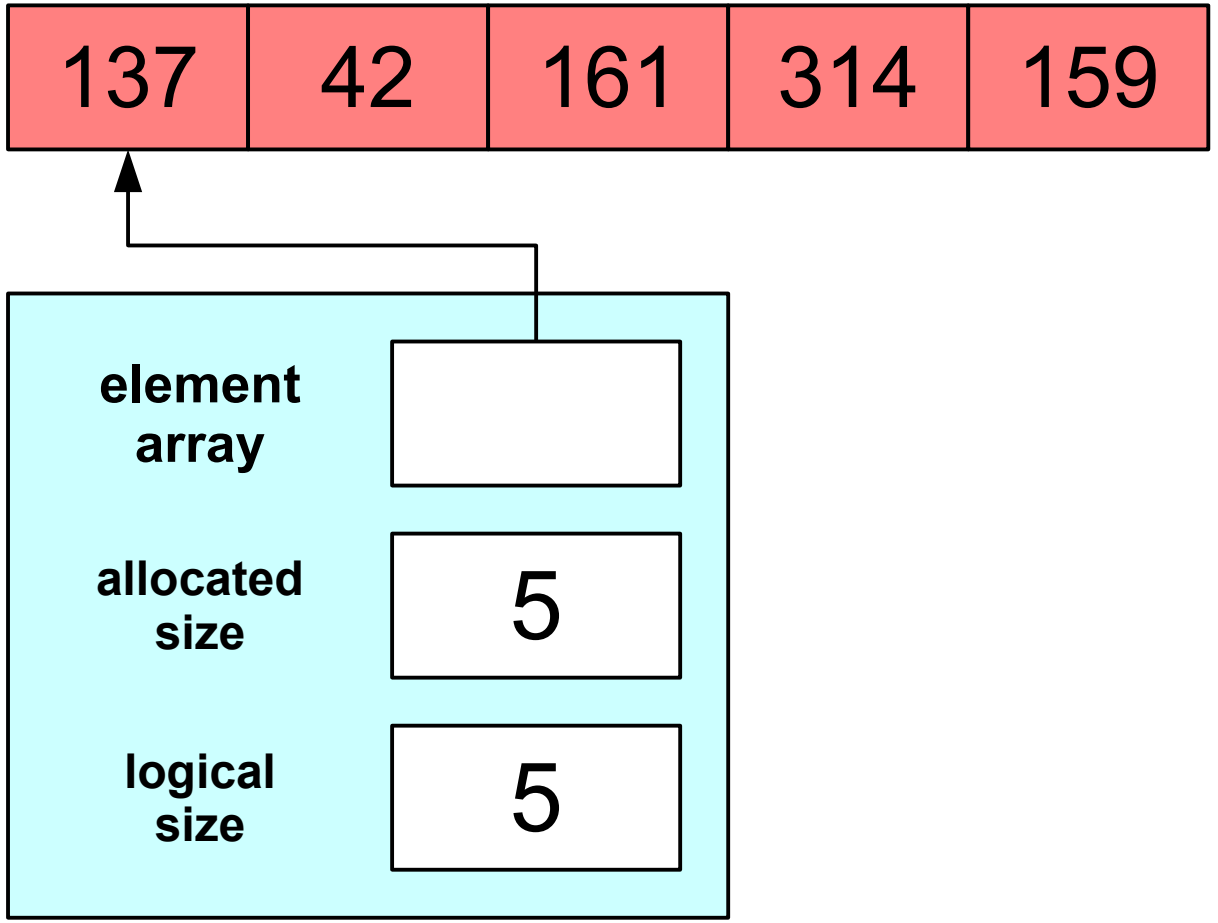
An Initial Idea



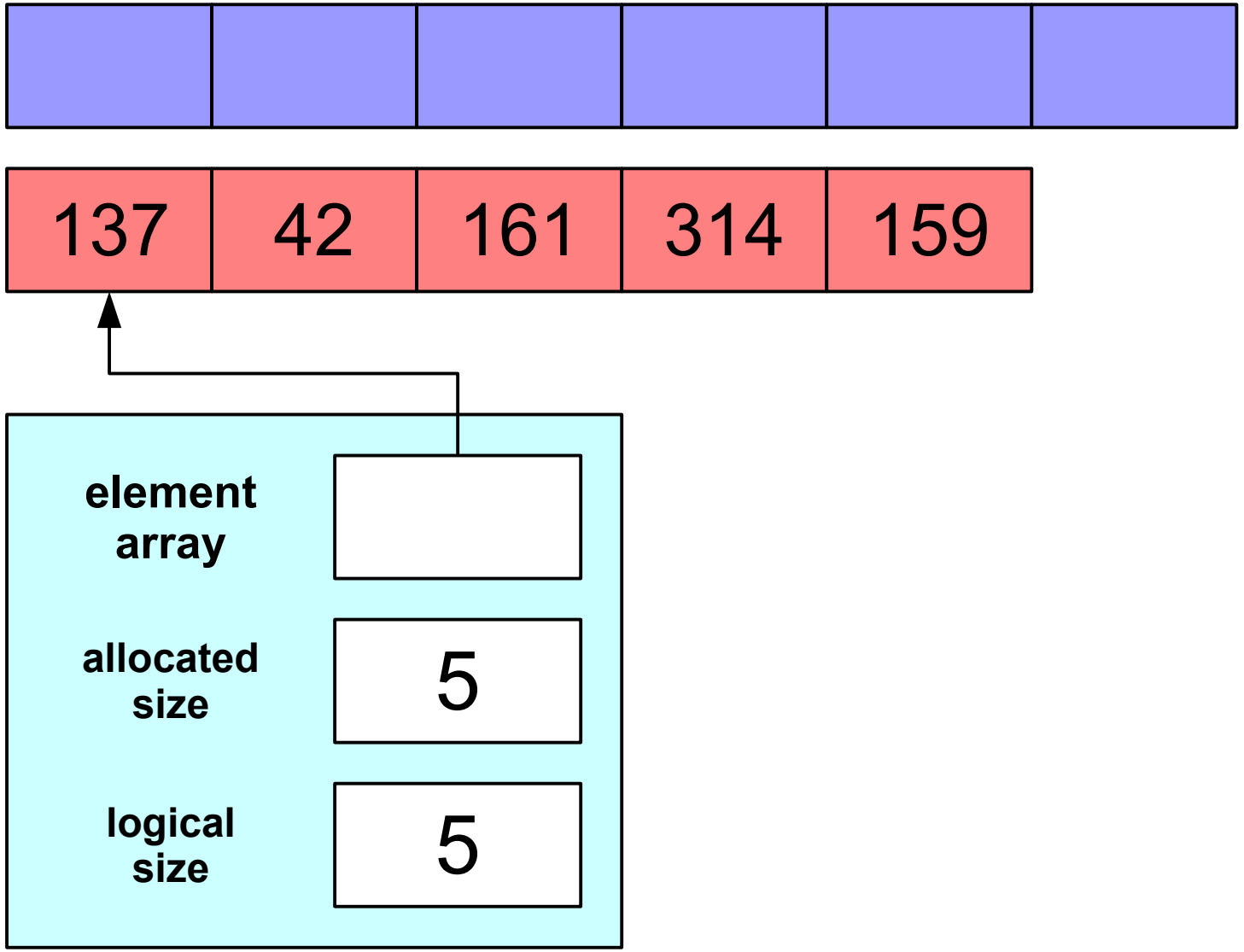
An Initial Idea



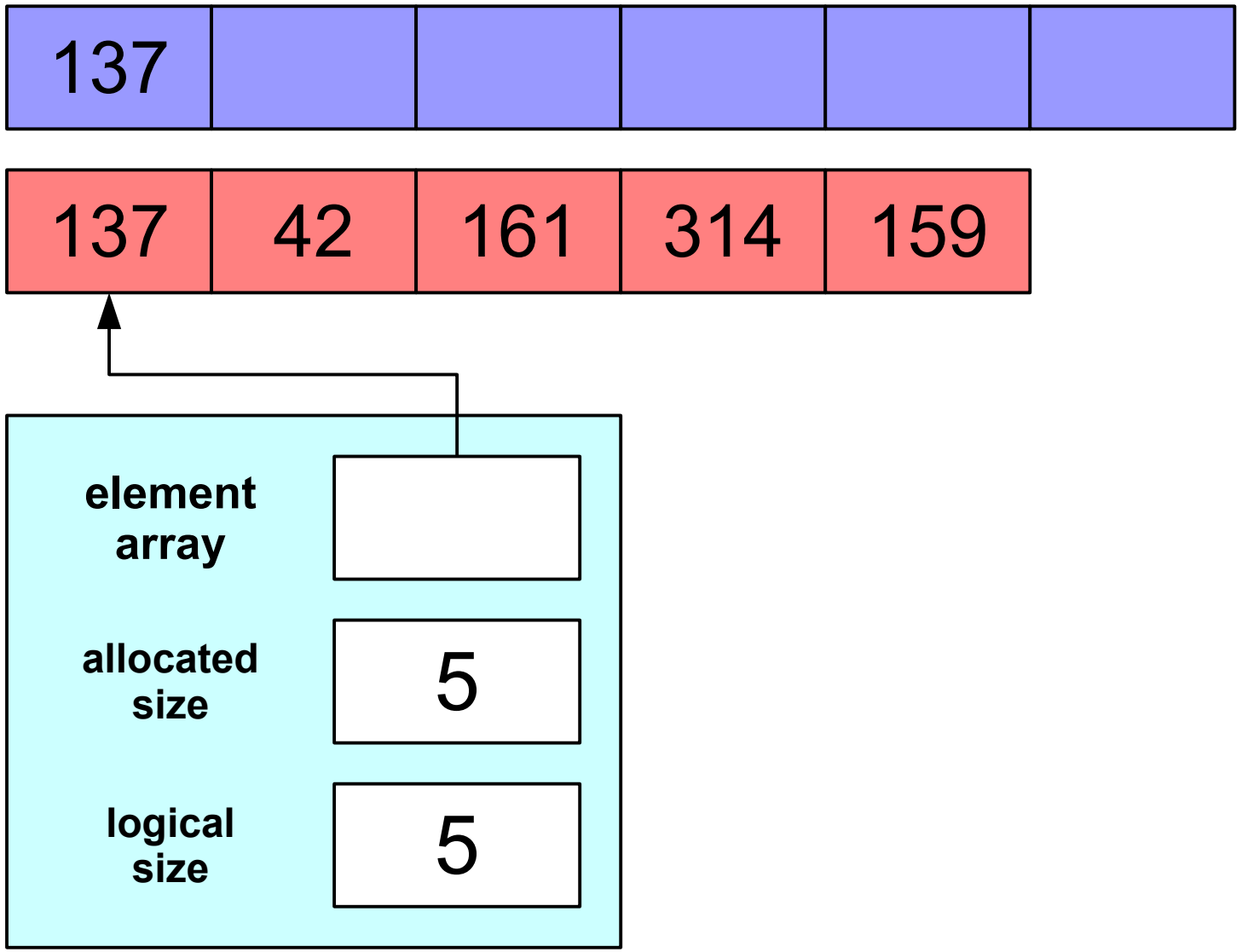
An Initial Idea



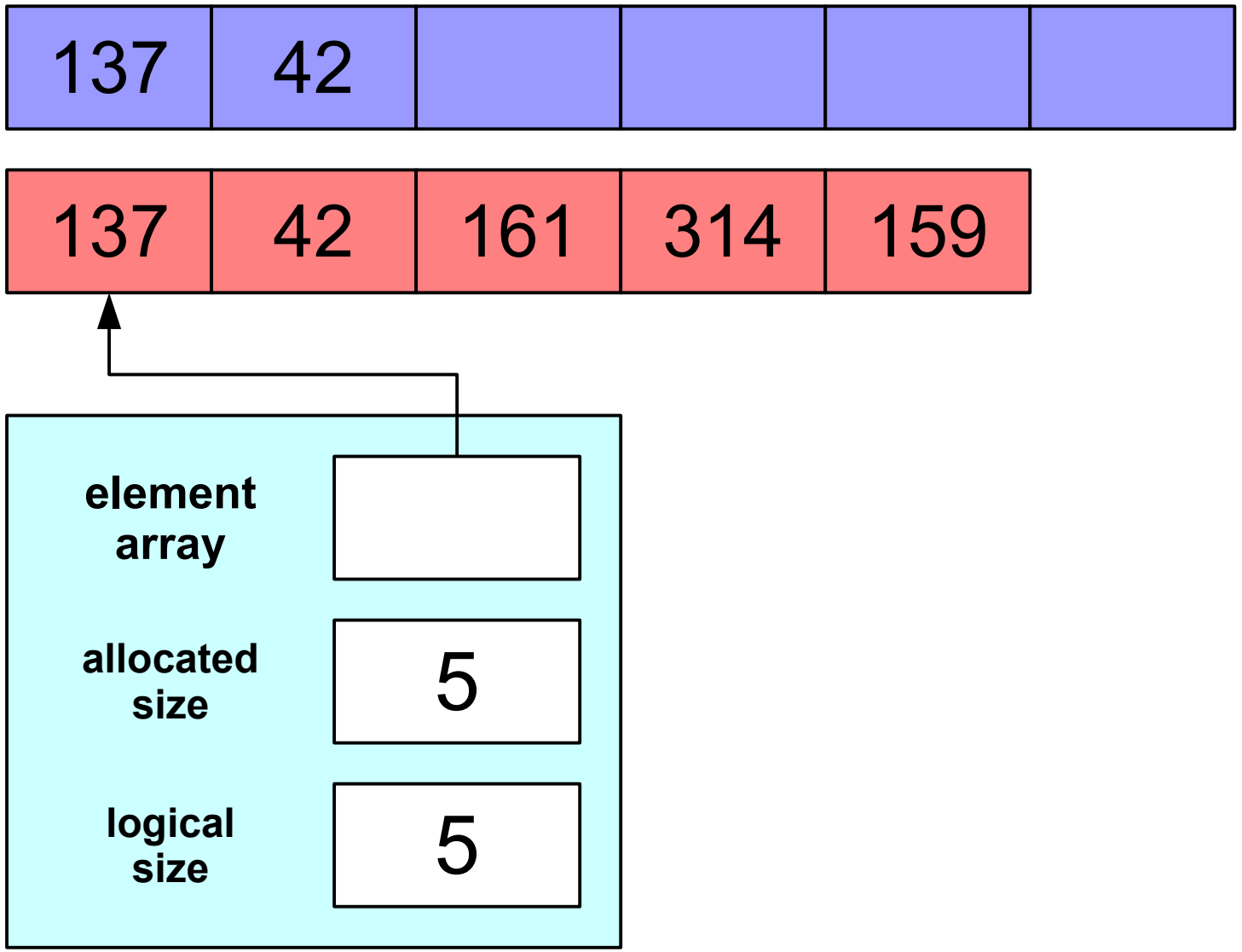
An Initial Idea



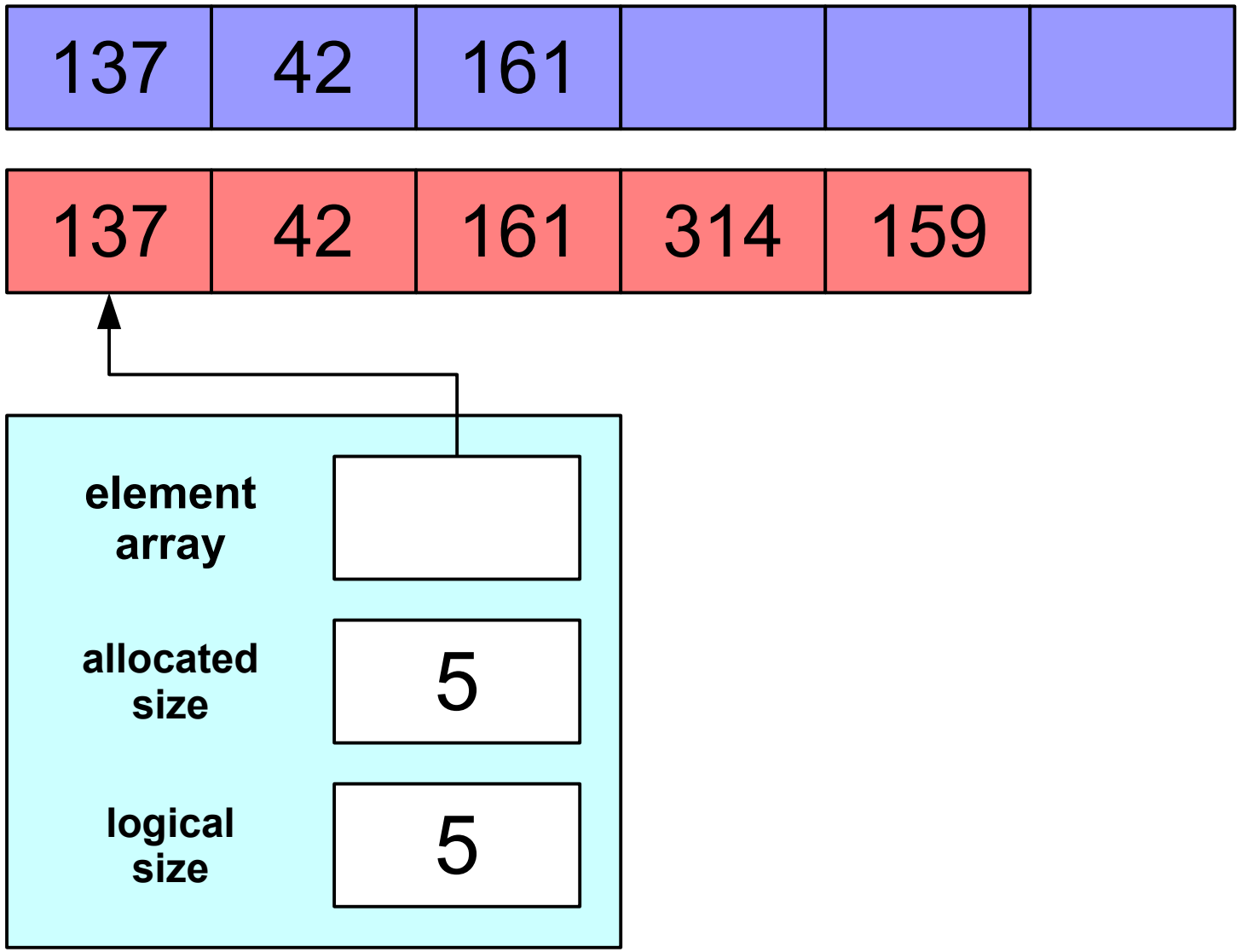
An Initial Idea



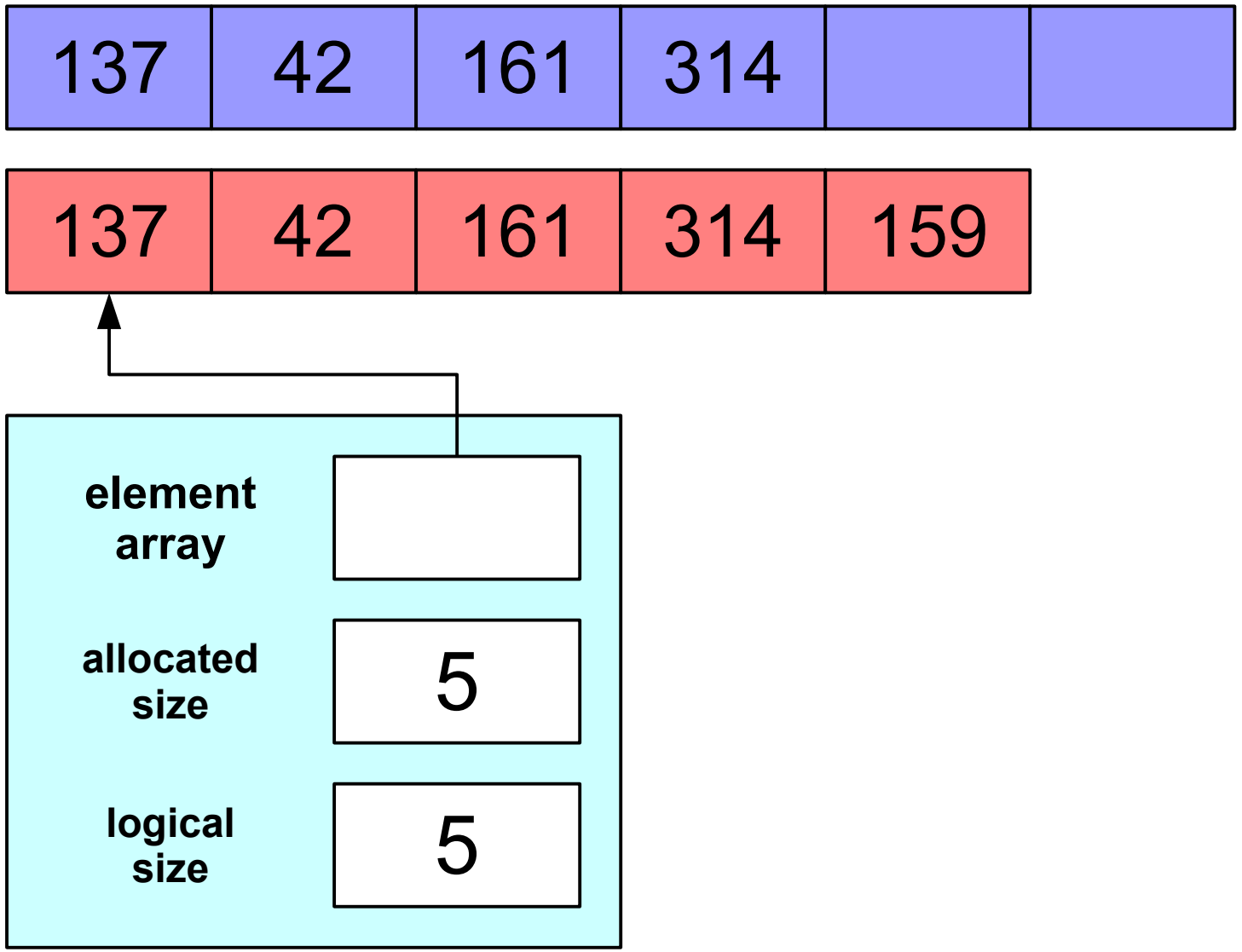
An Initial Idea



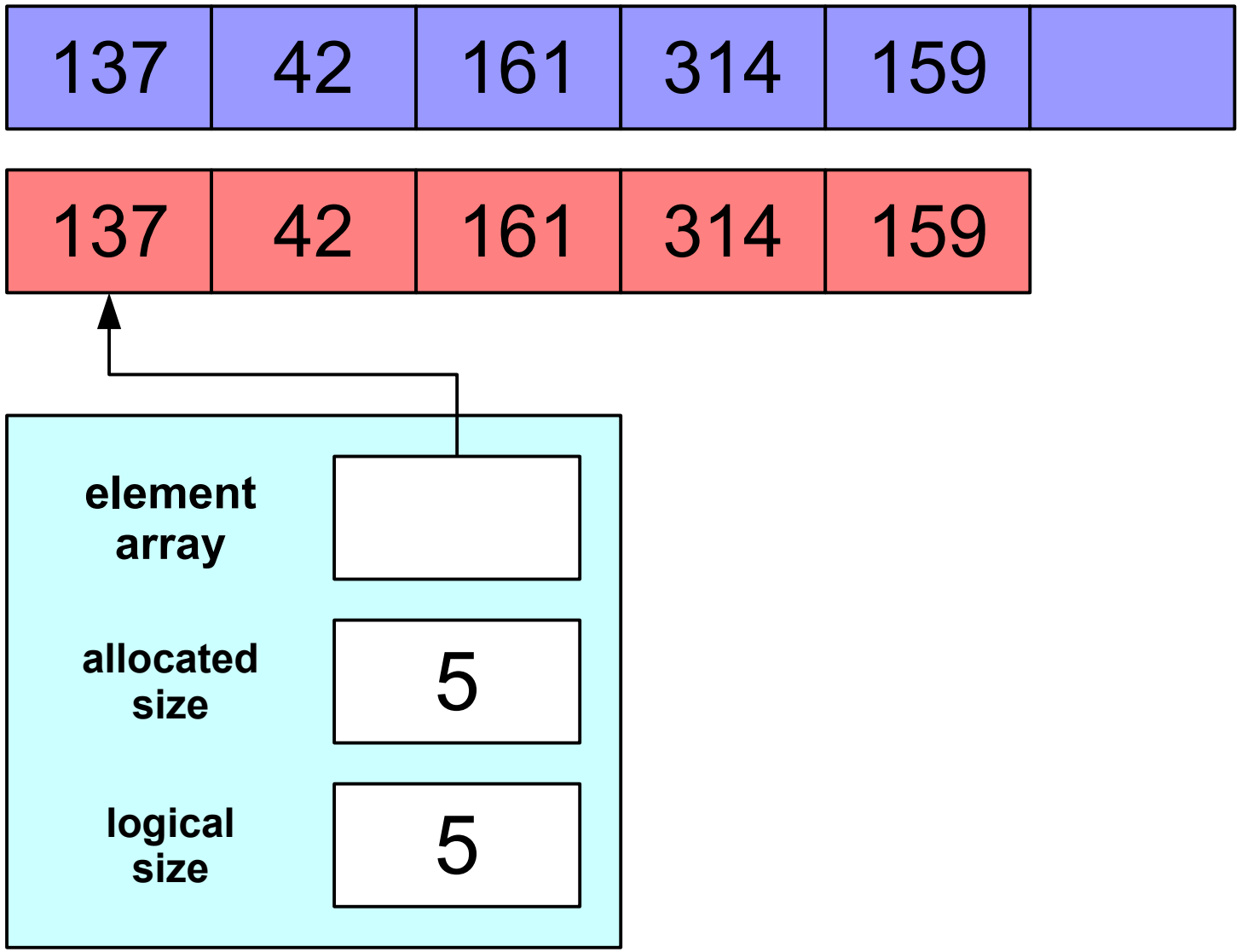
An Initial Idea



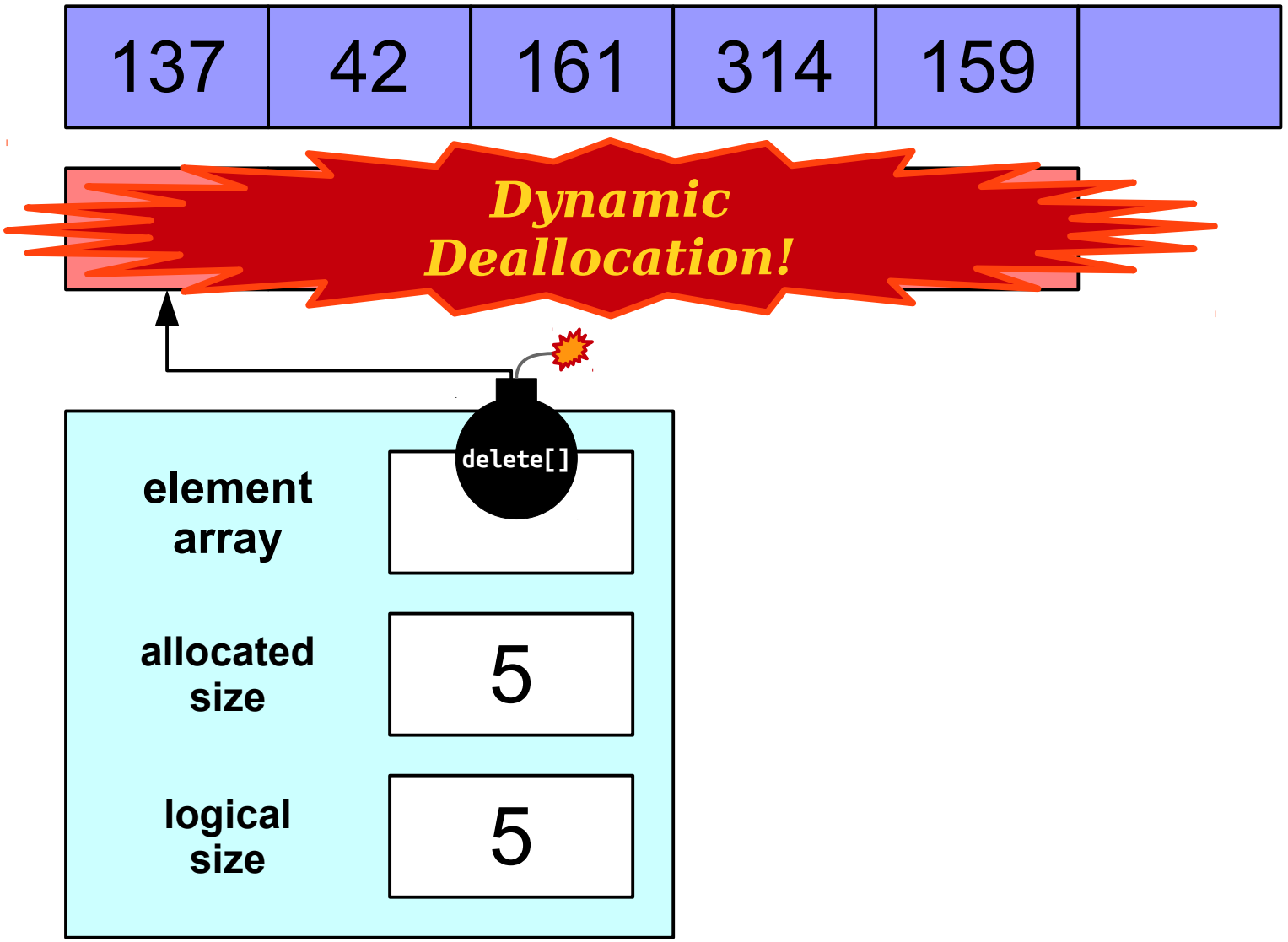
An Initial Idea



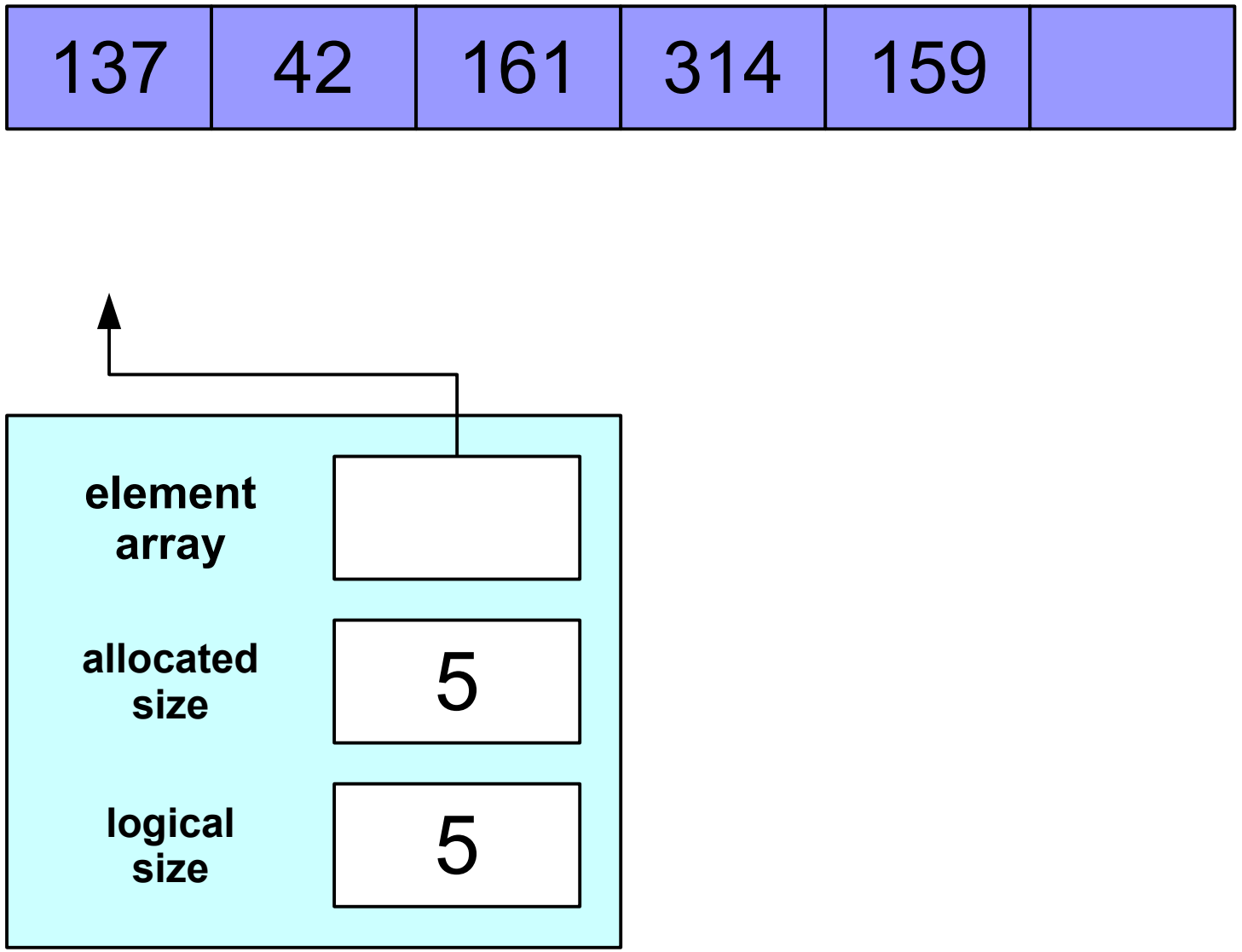
An Initial Idea



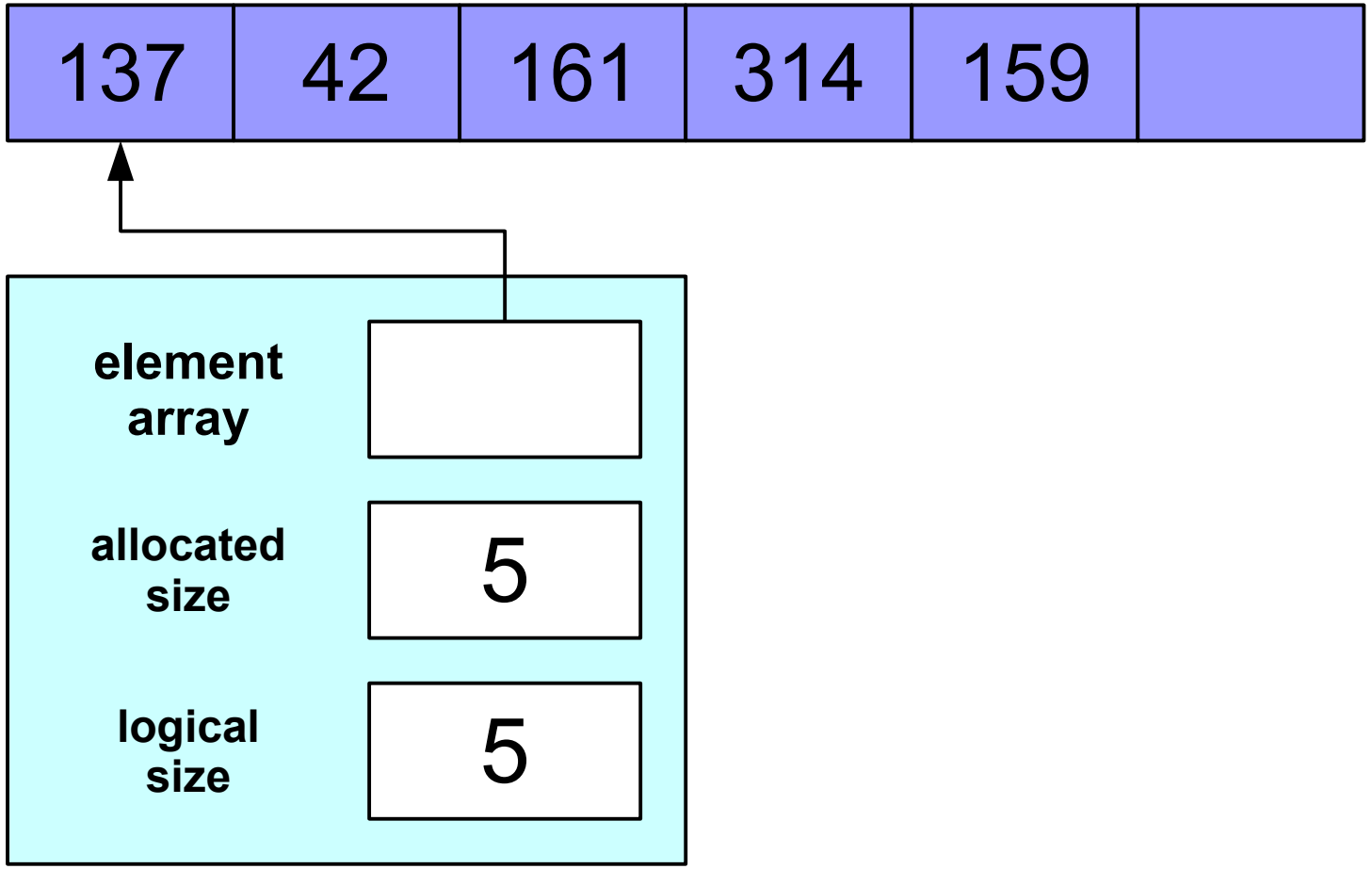
An Initial Idea



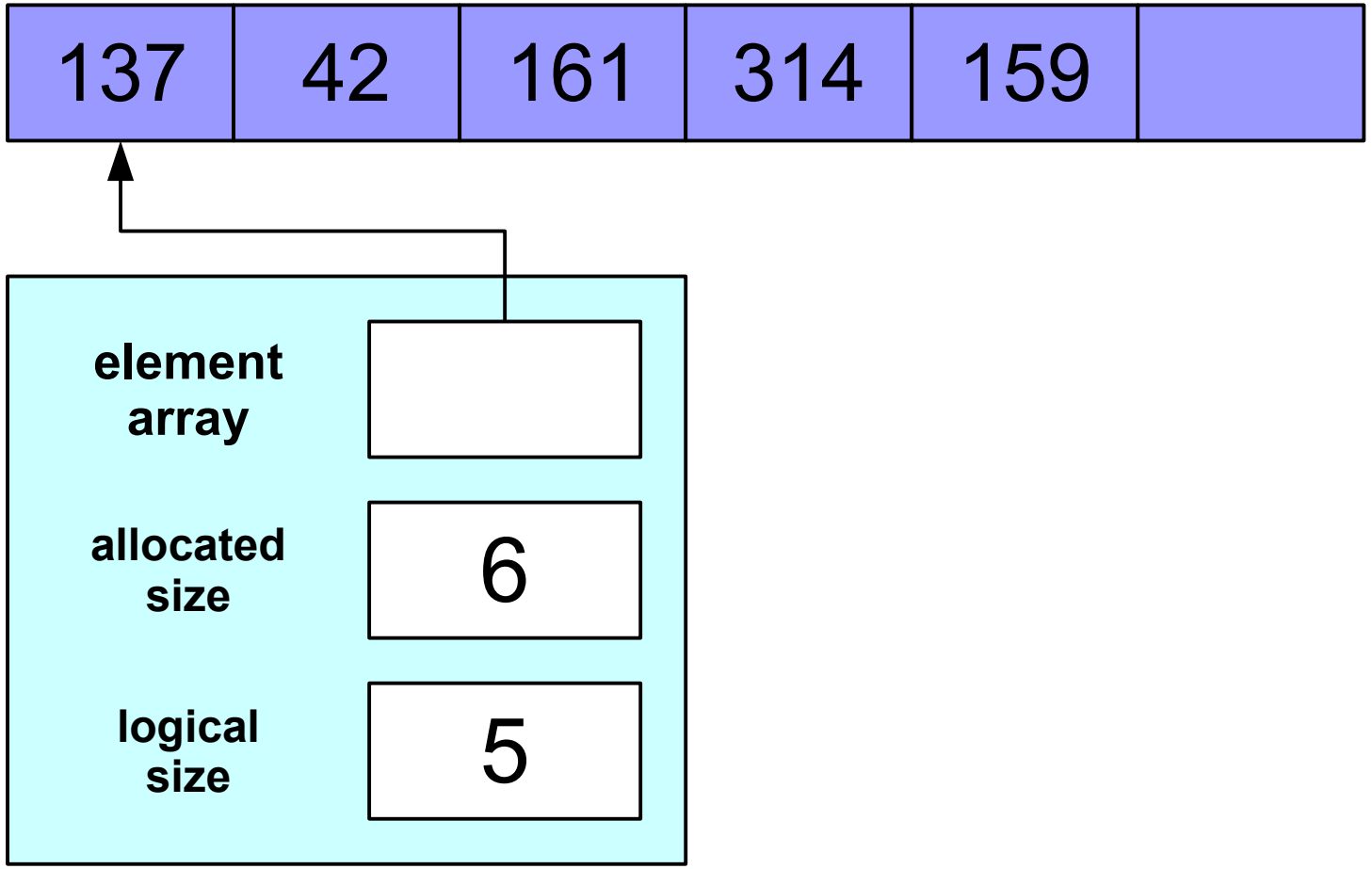
An Initial Idea



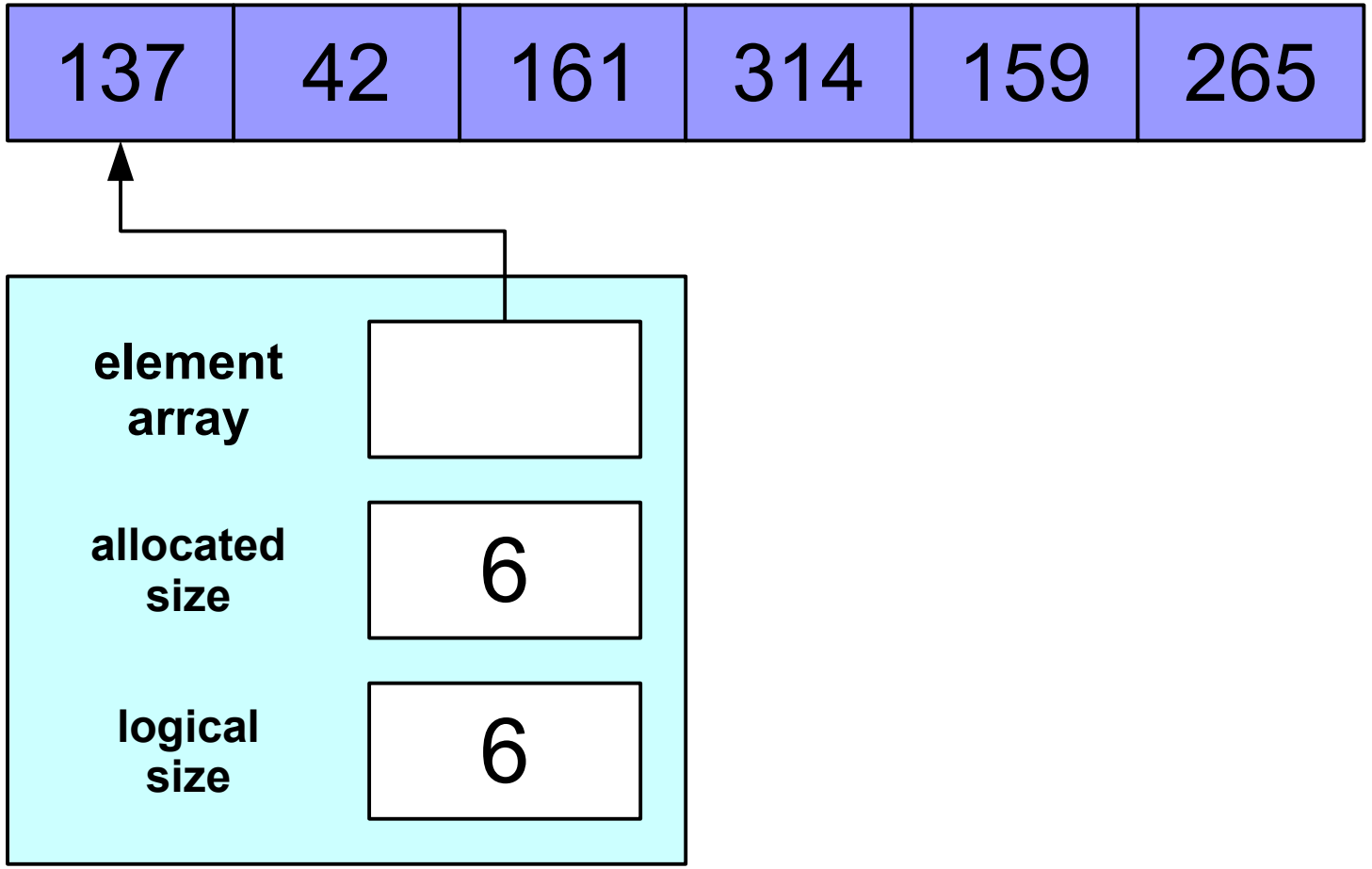
An Initial Idea



An Initial Idea



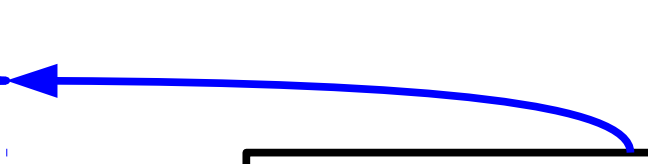
An Initial Idea



Ready... set... grow!

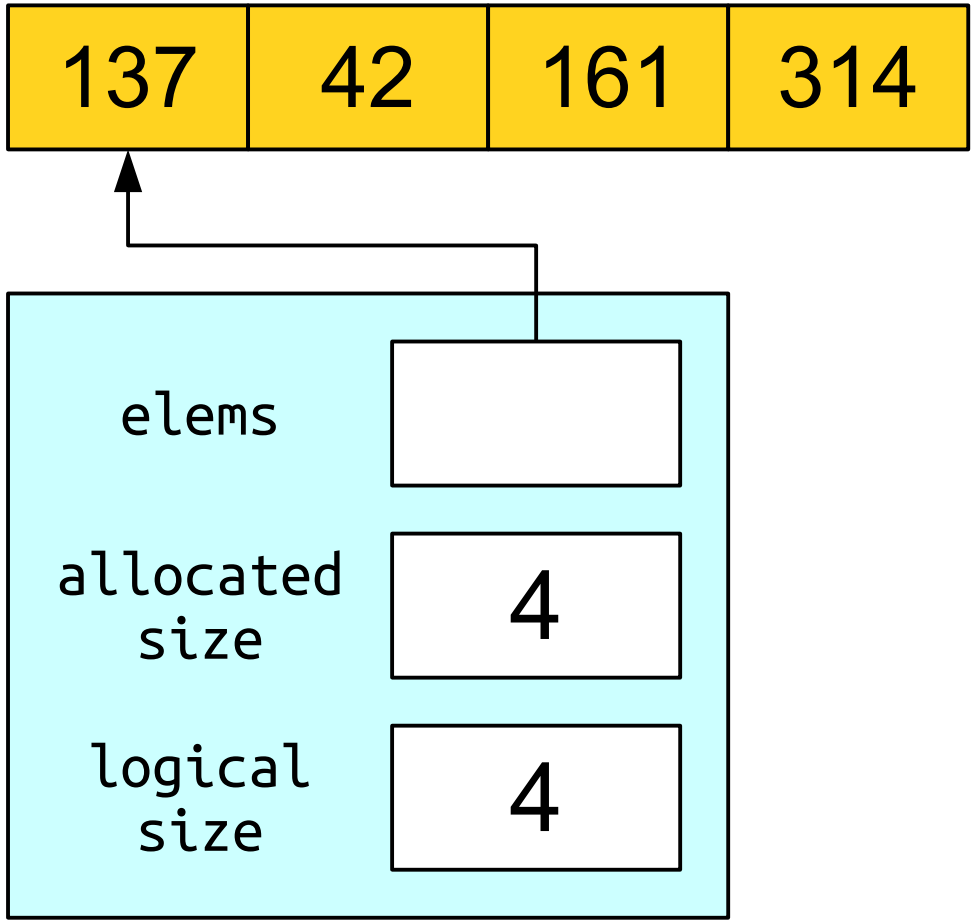
```
class OurStack {
public:
    OurStack();
    ~OurStack();

    void push(int value);
    int  pop();
    int  peek() const;

    int  size() const;
    bool isEmpty() const;

private:


    int* elems;
    int  allocatedSize;
    int  logicalSize;
};
```

```
class OurStack {
public:
    OurStack();
    ~OurStack();

    void push(int value);
    int  pop();
    int  peek() const;

    int  size() const;
    bool isEmpty() const;

private:
    void grow();

    int* elems;
    int  allocatedSize;
    int  logicalSize;
};
```

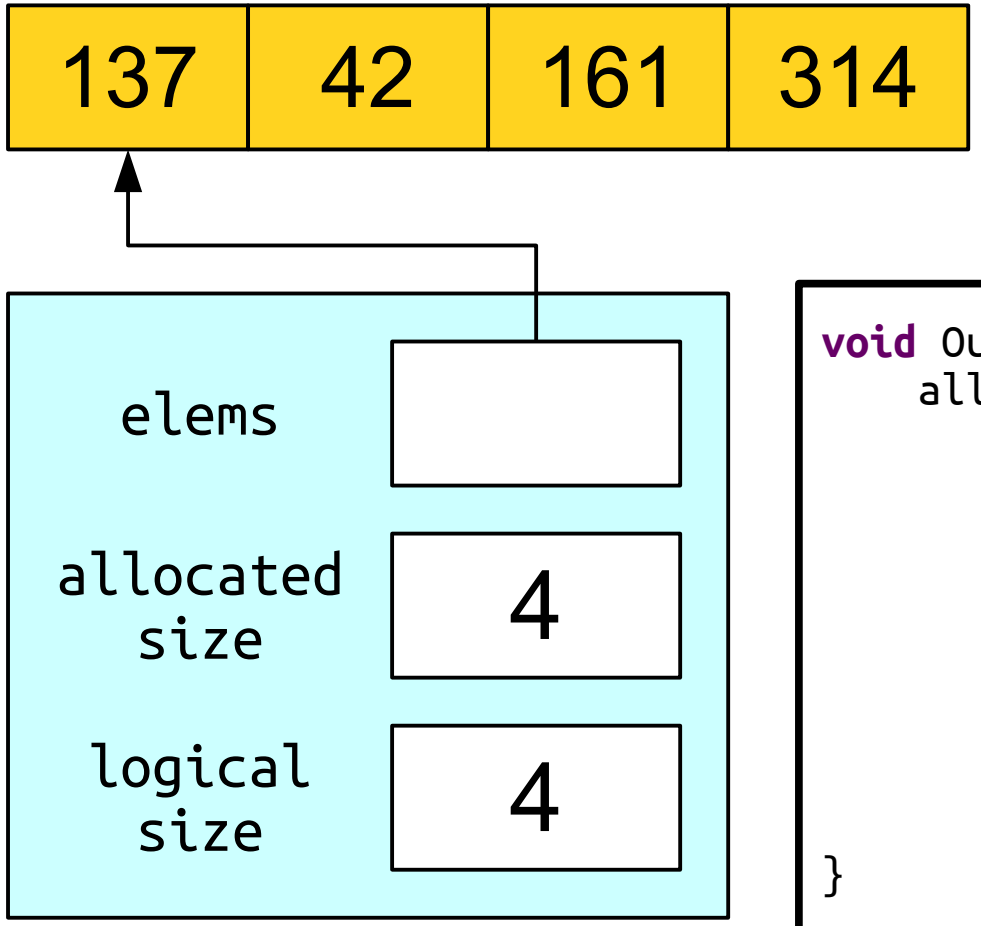


This is a ***private member function***. It's a helper function only the implementation can call.

An Initial Idea

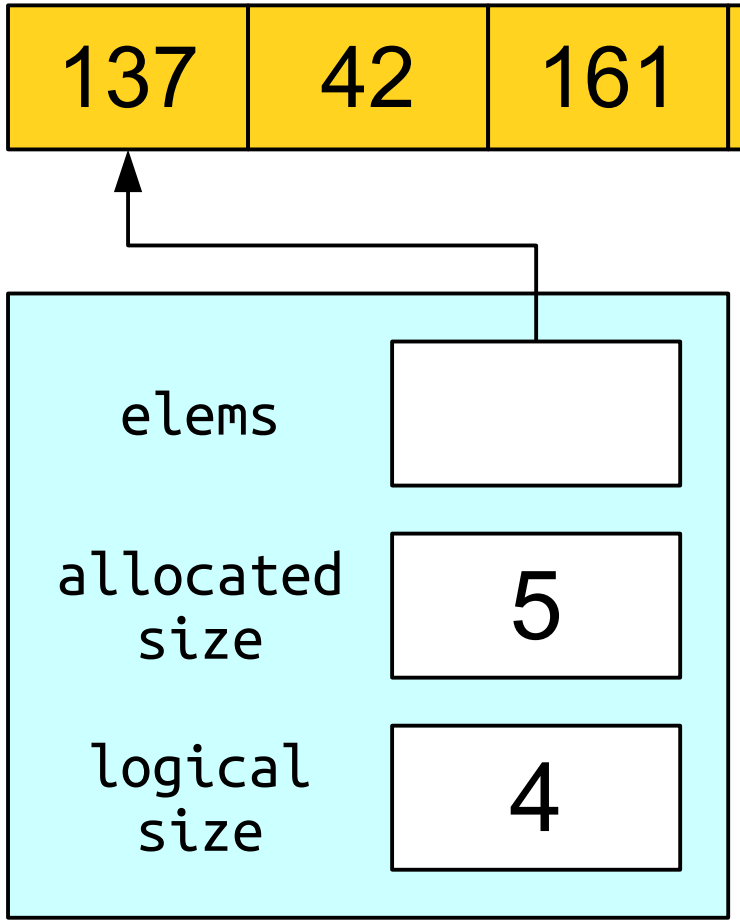


An Initial Idea



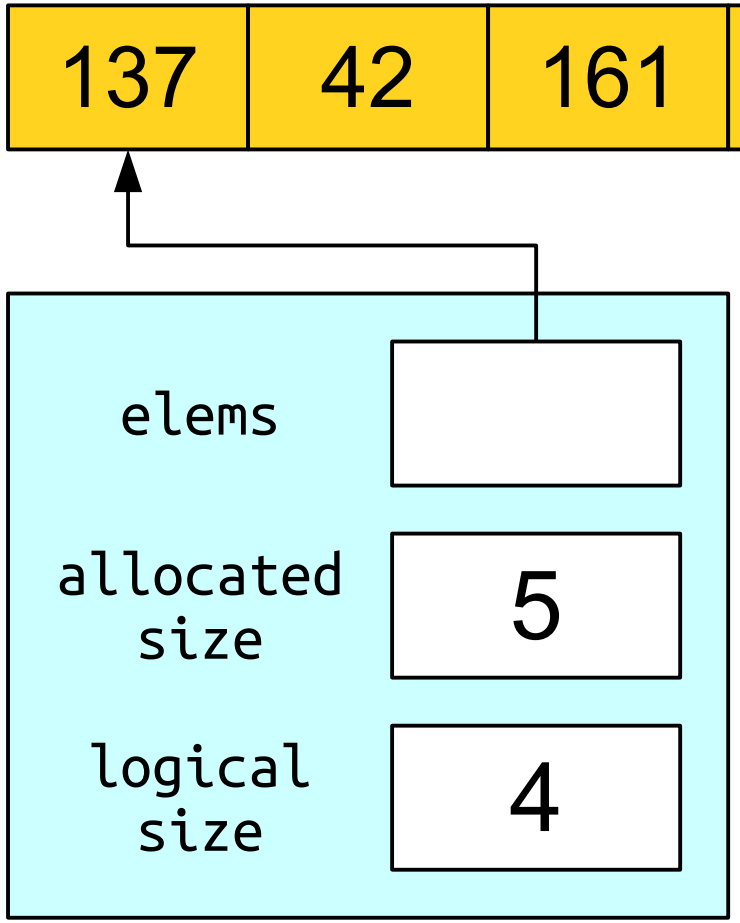

```
void OurStack::grow() {
    allocatedSize++;



}
```

An Initial Idea



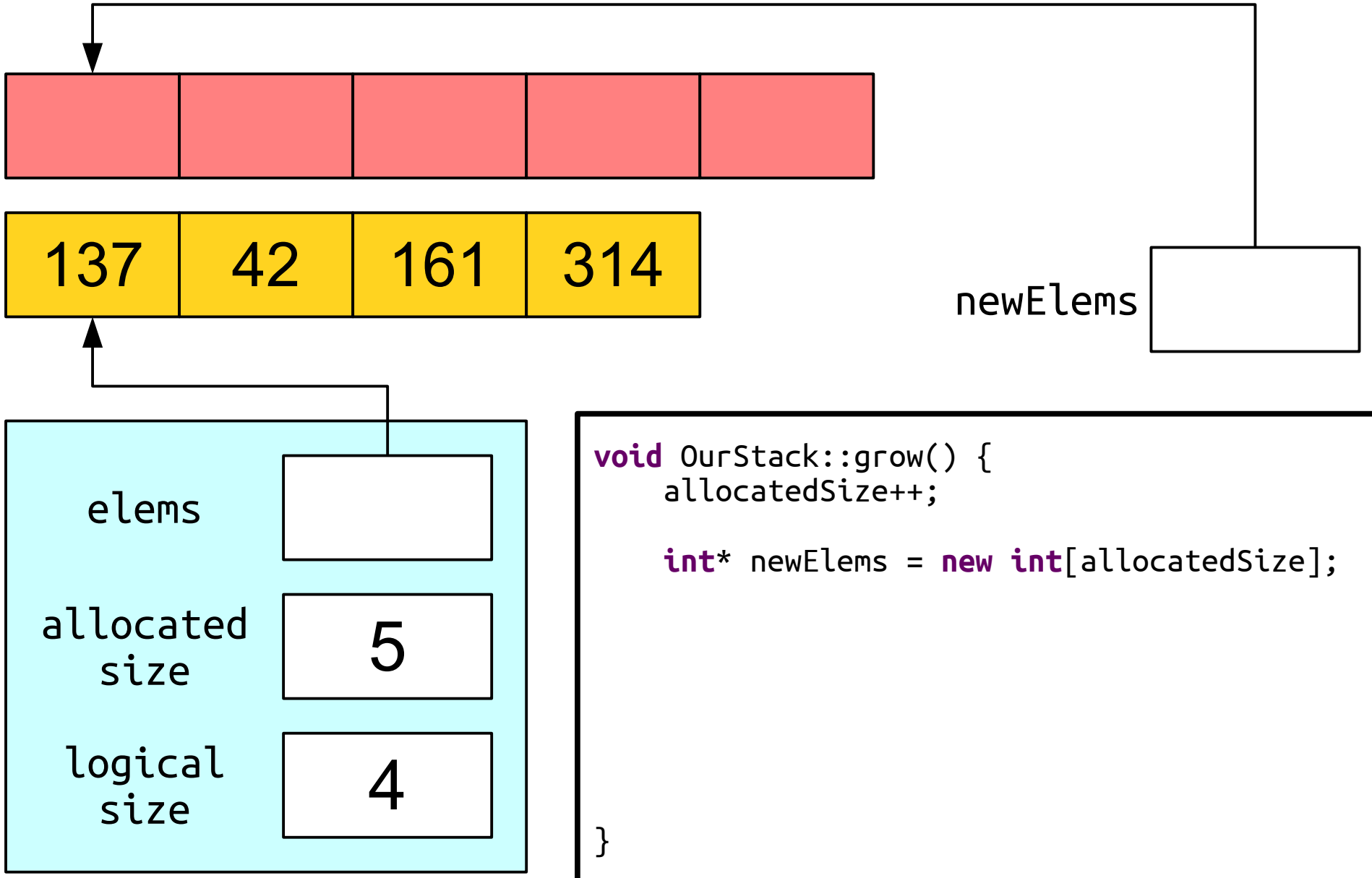

```
void OurStack::grow() {
    allocatedSize++;



}
```

An Initial Idea



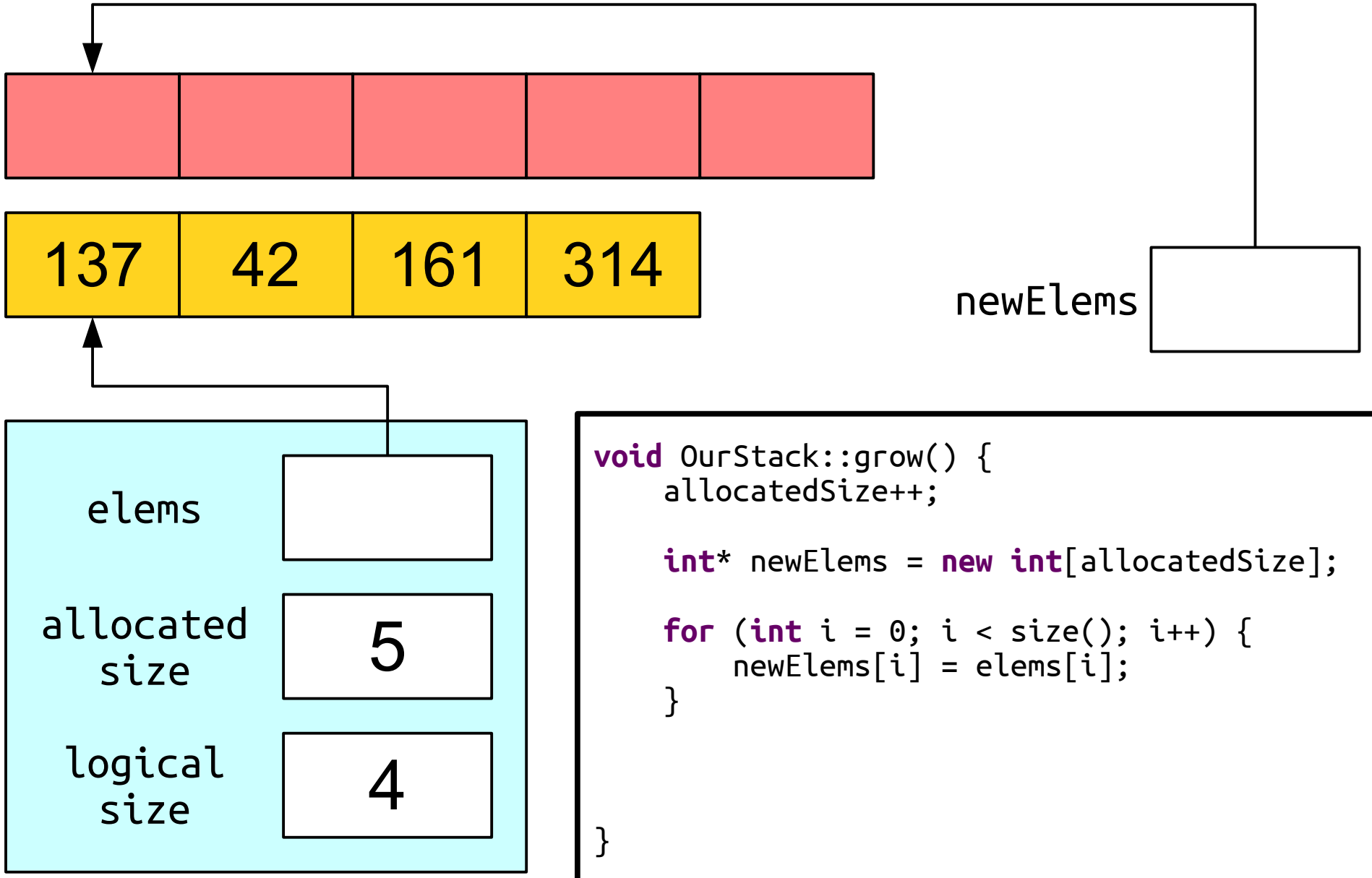

```
void OurStack::grow() {
    allocatedSize++;

    int* newElems = new int[allocatedSize];




}
```

An Initial Idea



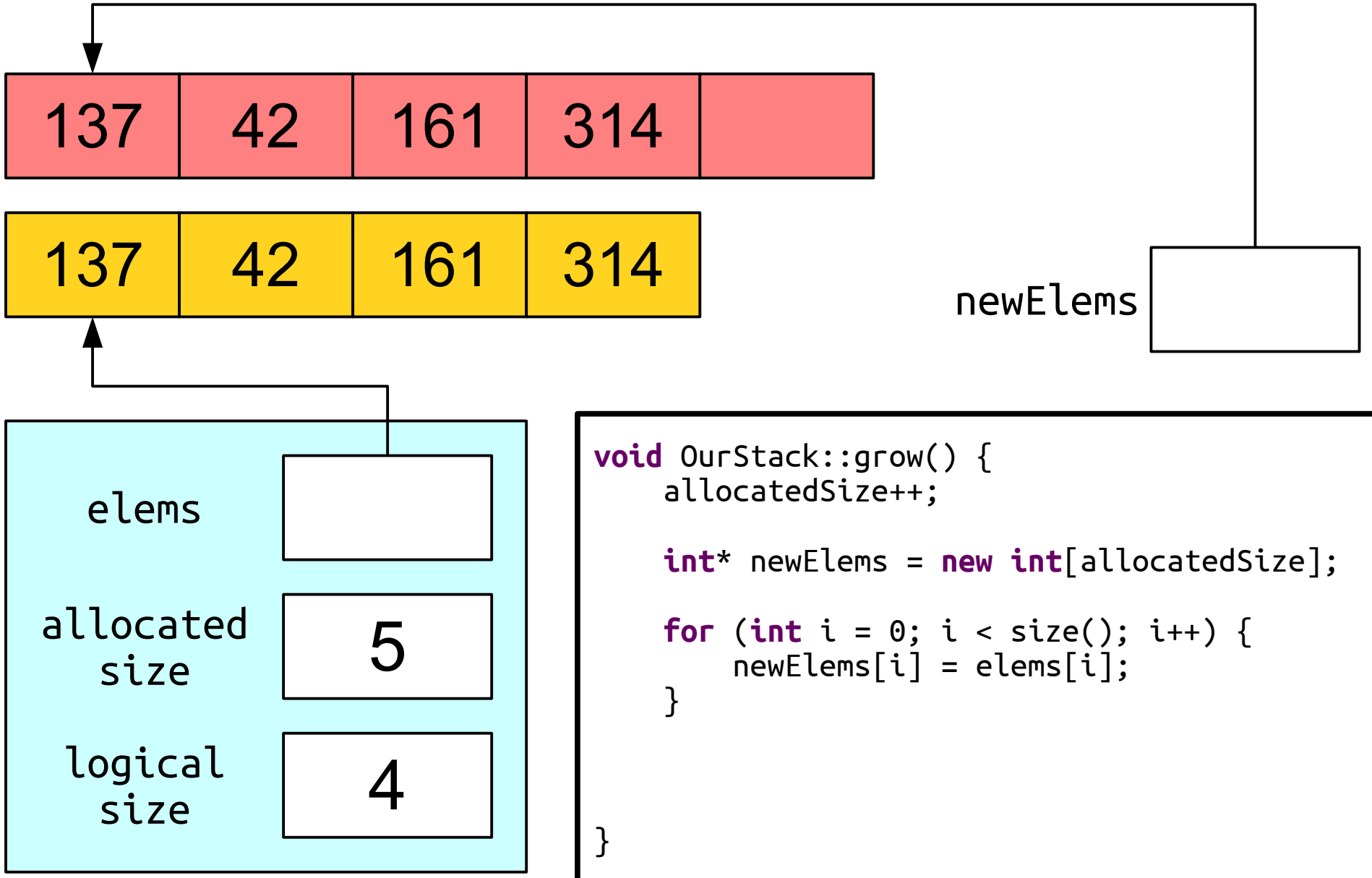

```
void OurStack::grow() {
    allocatedSize++;

    int* newElems = new int[allocatedSize];



}
```

An Initial Idea



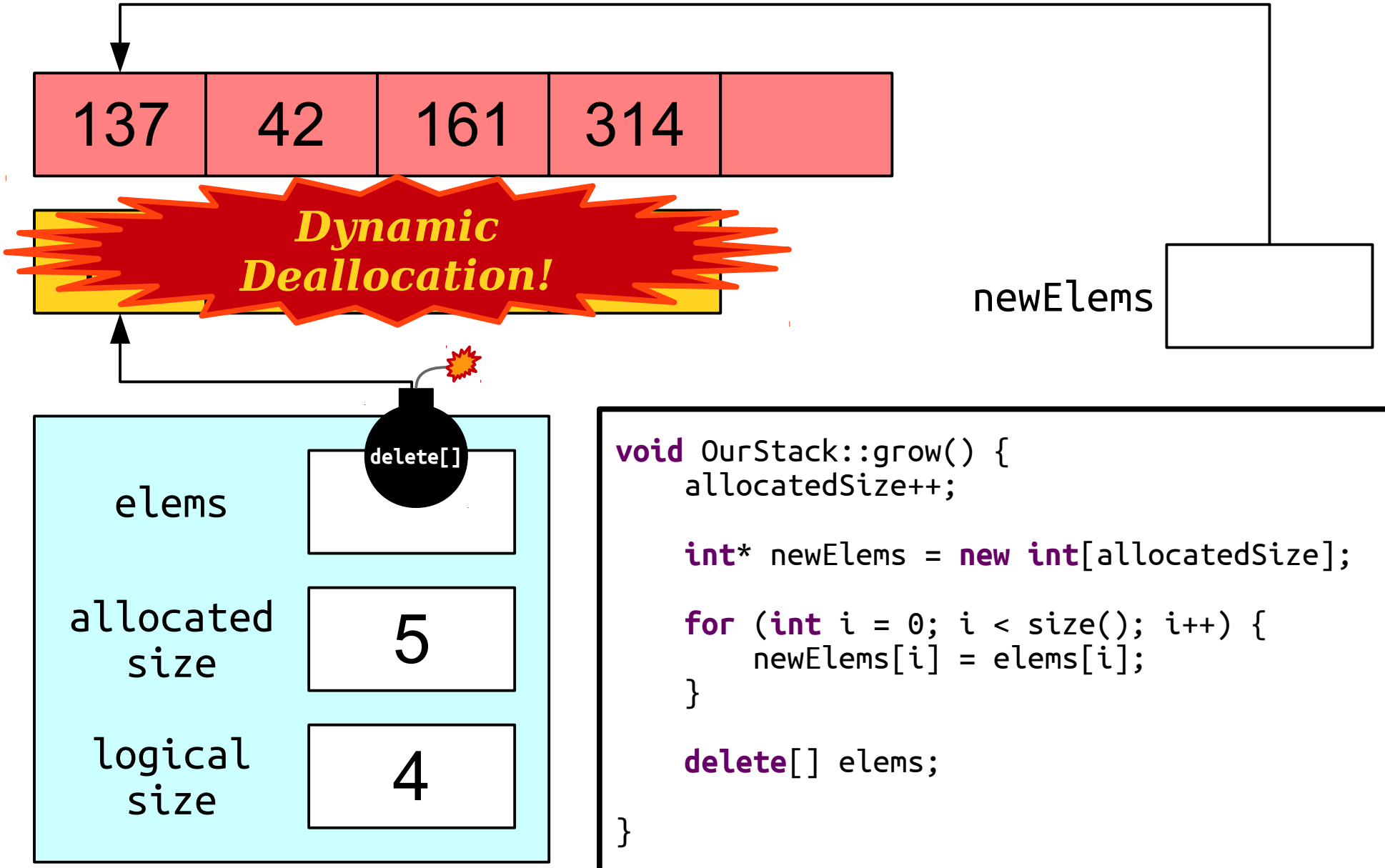

```
void OurStack::grow() {
    allocatedSize++;

    int* newElems = new int[allocatedSize];

    for (int i = 0; i < size(); i++) {
        newElems[i] = elems[i];
    }



}
```

An Initial Idea



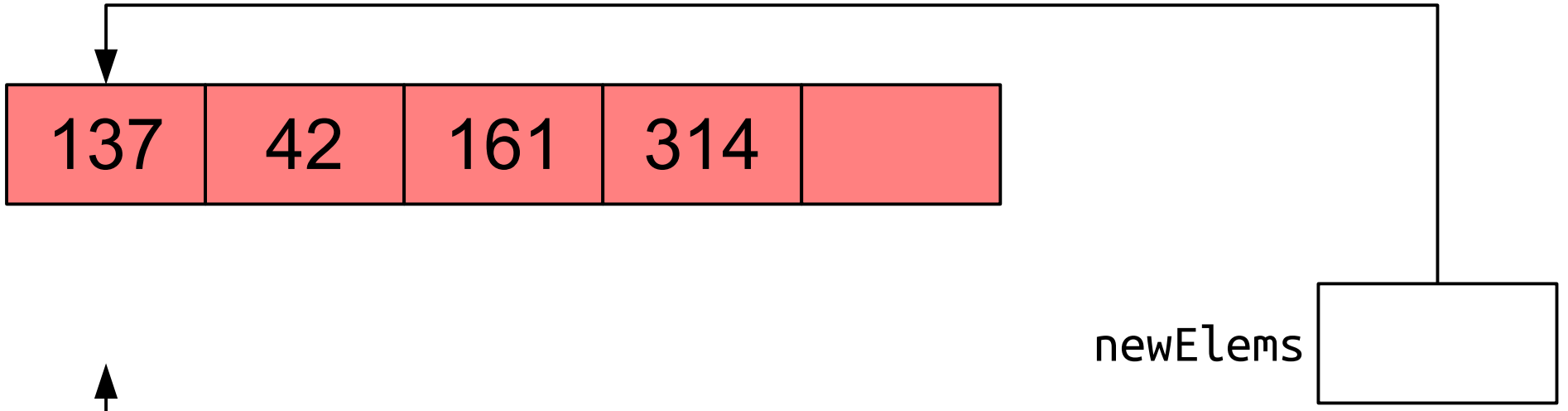

```
void OurStack::grow() {
    allocatedSize++;

    int* newElems = new int[allocatedSize];

    for (int i = 0; i < size(); i++) {
        newElems[i] = elems[i];
    }



}
```

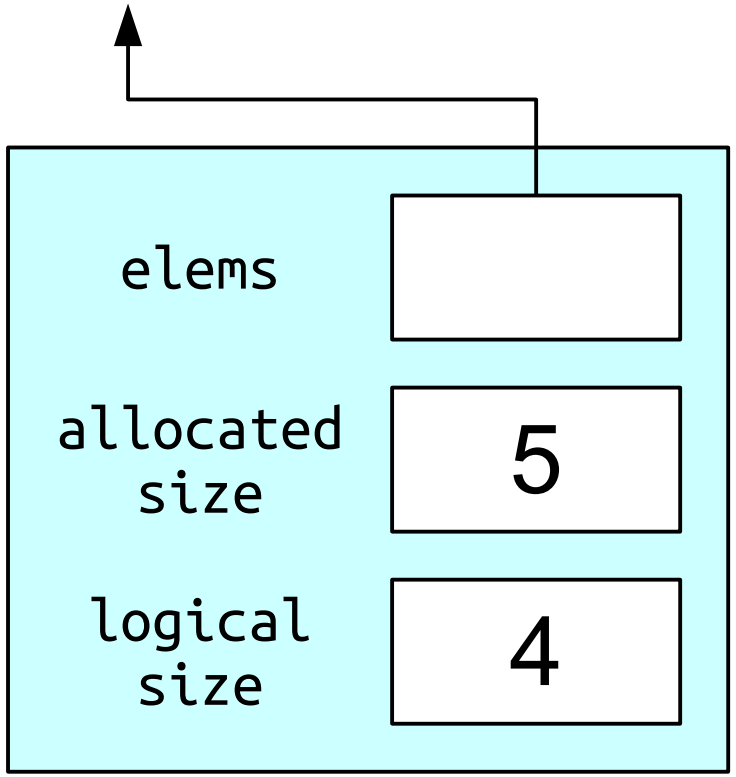
An Initial Idea




```
void OurStack::grow() {
    allocatedSize++;

    int* newElems = new int[allocatedSize];

    for (int i = 0; i < size(); i++) {
        newElems[i] = elems[i];
    }

    delete[] elems;

}
```

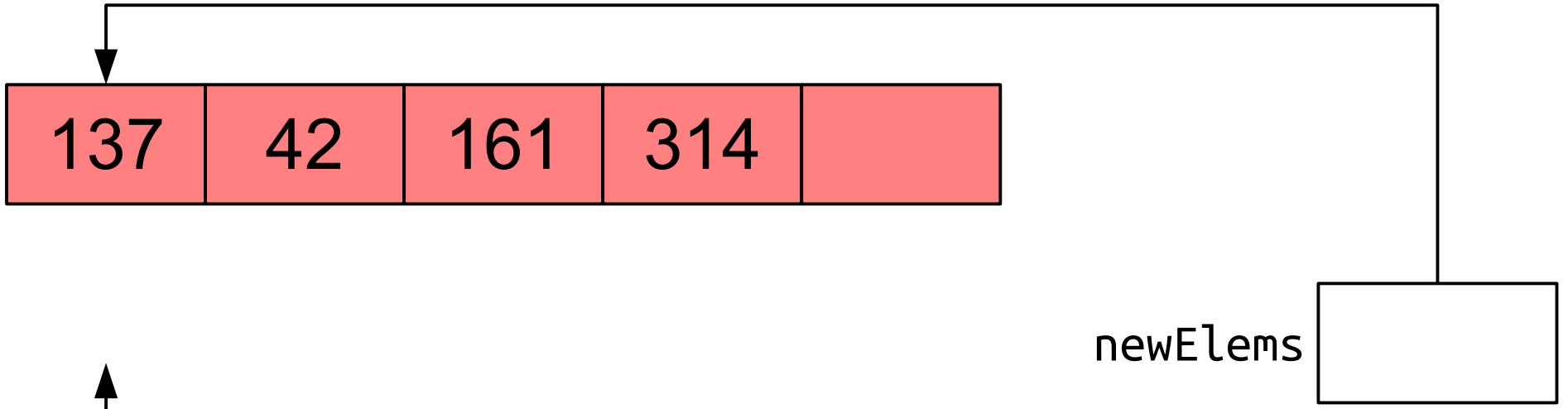
An Initial Idea

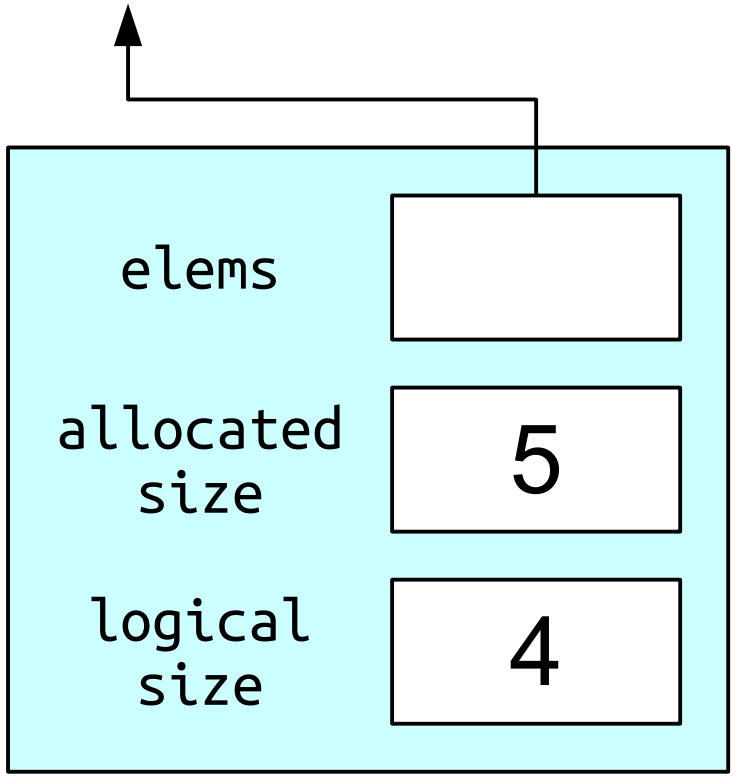





```
void OurStack::grow() {
    allocatedSize++;

    int* newElems = new int[allocatedSize];

    for (int i = 0; i < size(); i++) {
        newElems[i] = elems[i];
    }

    delete[] elems;

}
```

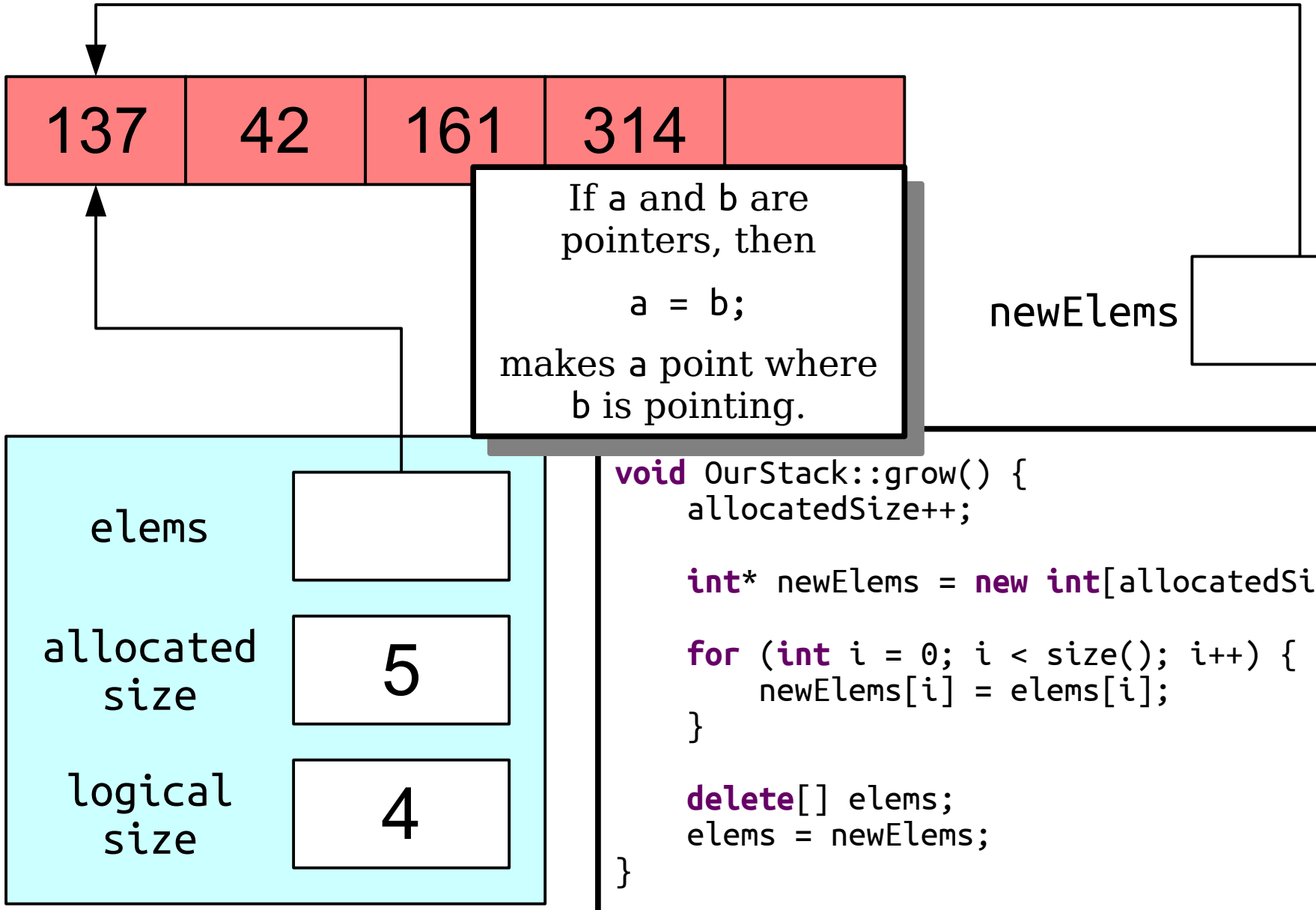
An Initial Idea





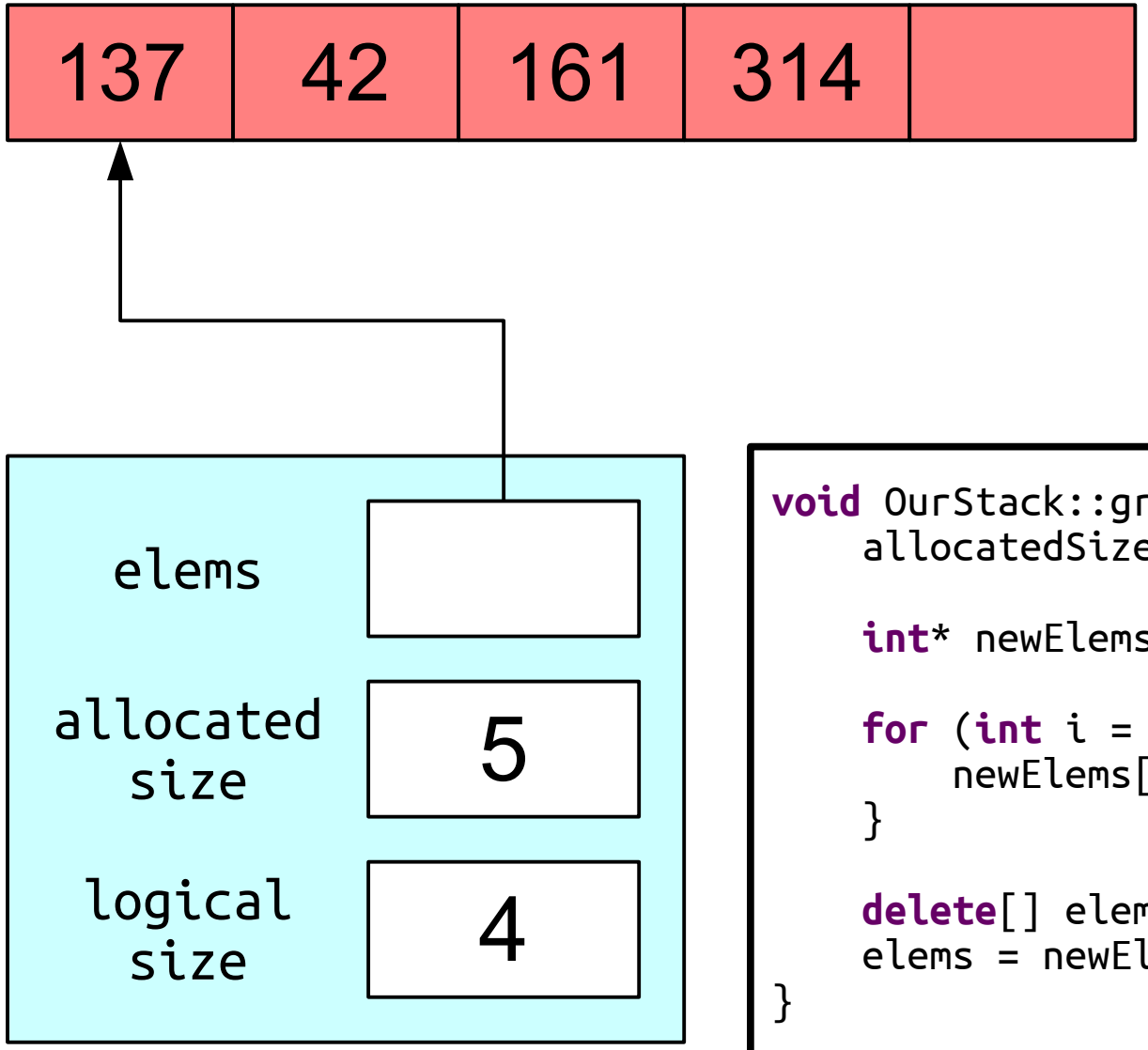

```
void OurStack::grow() {
    allocatedSize++;

    int* newElems = new int[allocatedSize];

    for (int i = 0; i < size(); i++) {
        newElems[i] = elems[i];
    }

    delete[] elems;
    elems = newElems;
}
```

An Initial Idea




```
void OurStack::grow() {
    allocatedSize++;

    int* newElems = new int[allocatedSize];

    for (int i = 0; i < size(); i++) {
        newElems[i] = elems[i];
    }

    delete[] elems;
    elems = newElems;
}
```

An Initial Idea




```
void OurStack::grow() {
    allocatedSize++;

    int* newElems = new int[allocatedSize];

    for (int i = 0; i < size(); i++) {
        newElems[i] = elems[i];
    }

    delete[] elems;
    elems = newElems;
}
```

Analyzing Our Approach

- We now have a working solution, but is it an *efficient* solution?
- Let's analyze the big-O complexity of the five operations.
  - `size`:
  - `isEmpty`:
  - `push`:
  - `pop`:
  - `peek`:

Analyzing Our Approach

- We now have a working solution, but is it an *efficient* solution?
- Let's analyze the big-O complexity of the five operations.
  - `size`: **O(1)**
  - `isEmpty`: **O(1)**
  - `push`:
  - `pop`:
  - `peek`:

Analyzing Our Approach

- We now have a working solution, but is it an *efficient* solution?
- Let's analyze the big-O complexity of the five operations.
  - `size`: **O(1)**
  - `isEmpty`: **O(1)**
  - `push`: **O(*n*)**
  - `pop`:
  - `peek`:

Analyzing Our Approach

- We now have a working solution, but is it an *efficient* solution?
- Let's analyze the big-O complexity of the five operations.
  - `size`: **O(1)**
  - `isEmpty`: **O(1)**
  - `push`: **O(*n*)**
  - `pop`: **O(1)**
  - `peek`: **O(1)**

What This Means

- What is the complexity of pushing n elements and then popping them?

What This Means

- What is the complexity of pushing n elements and then popping them?
- Cost of the pushes:
  - $1 + 2 + 3 + 4 + \ldots + n$

What This Means

- What is the complexity of pushing n elements and then popping them?
- Cost of the pushes:
  - $1 + 2 + 3 + 4 + \ldots + n = \mathbf{O(n^2)}$

What This Means

- What is the complexity of pushing n elements and then popping them?
- Cost of the pushes:
  - $1 + 2 + 3 + 4 + \ldots + n = \mathbf{O(n^2)}$
- Cost of the pops:
  - $1 + 1 + 1 + 1 + \ldots + 1$

What This Means

- What is the complexity of pushing n elements and then popping them?
- Cost of the pushes:
  - $1 + 2 + 3 + 4 + \ldots + n = \mathbf{O(n^2)}$
- Cost of the pops:
  - $1 + 1 + 1 + 1 + \ldots + 1 = \mathbf{O(n)}$

What This Means

- What is the complexity of pushing n elements and then popping them?
- Cost of the pushes:
  - $1 + 2 + 3 + 4 + \ldots + n = \mathbf{O(n^2)}$
- Cost of the pops:
  - $1 + 1 + 1 + 1 + \ldots + 1 = \mathbf{O(n)}$
- Total cost:

What This Means

- What is the complexity of pushing n elements and then popping them?
- Cost of the pushes:
  - $1 + 2 + 3 + 4 + \ldots + n = \mathbf{O(n^2)}$
- Cost of the pops:
  - $1 + 1 + 1 + 1 + \ldots + 1 = \mathbf{O(n)}$
- Total cost: $\mathbf{O(n^2)}$

Validating Our Model

Time-Out for Announcements!

Assignment 4

- Assignment 4 is due on Friday.
- You can use a late day to extend the deadline to Wednesday (there’s no class on Monday), but we don’t recommend this.
  - That will eat into your time for studying for the exam.
  - Topics from Assignment 4 are fair game for the exam.
- YEAH Hours for Assignment 5 will be on Friday at 3:30PM in 380-380Y.

Midterm Exam

- The midterm exam is next ***Tuesday, February 19*** from ***7:00PM – 10:00PM***. Locations are divvied up by last (family) name:
  - A – K: Go to ***Bishop Auditorium***
  - L – Z: Go to ***Hewlett 200***
- It covers topics from Lectures 01 – 12 (up through and including big-O notation) and Assignments 0 – 4.
- The exam is closed-book and limited-note. You may bring one double-sided sheet of 8.5” × 11” of notes to the exam with you.

Midterm Exam

- We will be administering the exam using a software tool called ***BlueBook***.
- Visit the CS106B website, click the “BlueBook” link under the “Resources” tab, then download the BlueBook software.
- This week’s section handout will be done through BlueBook so that you get a chance to test it out.
- Need a laptop for the exam? We can help out with that. Please contact us ASAP so we can make appropriate arrangements.

Practice Midterm Exam

- There's a practice midterm exam up on the course website. It's a minimally-modified version of the exam we gave out in Winter 2017.
- The password is

  `maplesyrup`

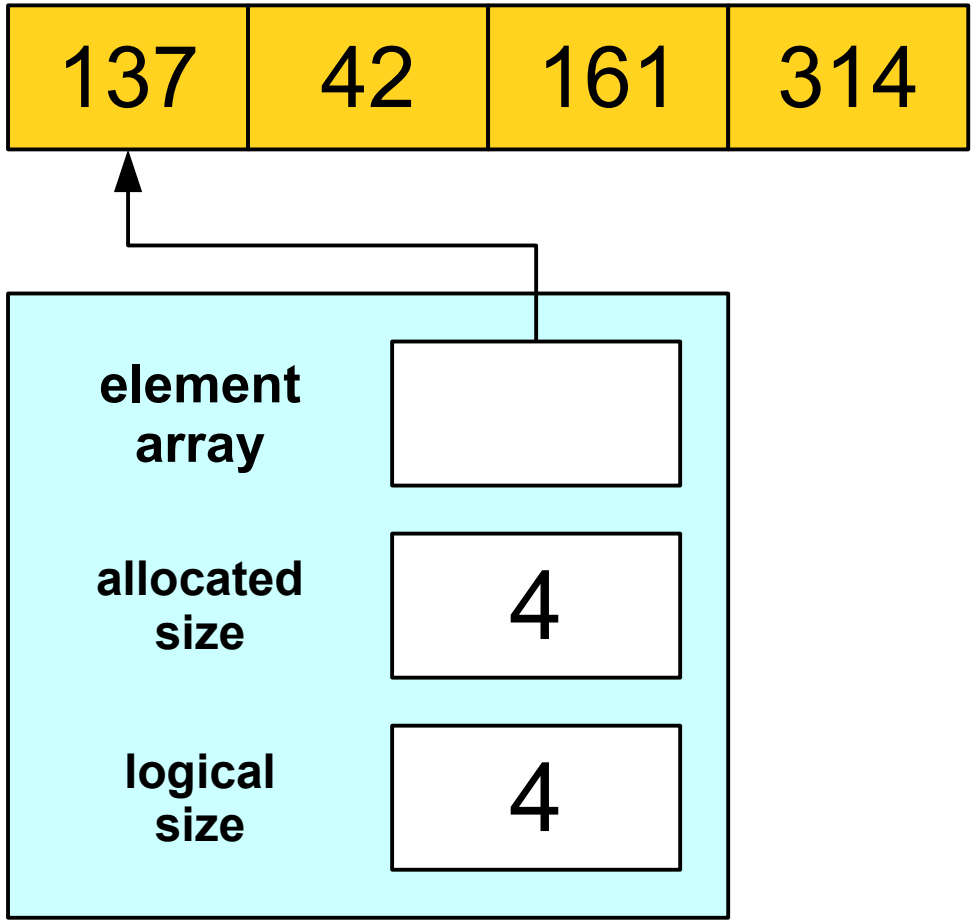
  and you'll see why when you start the exam. 😀

Back to the Stack!

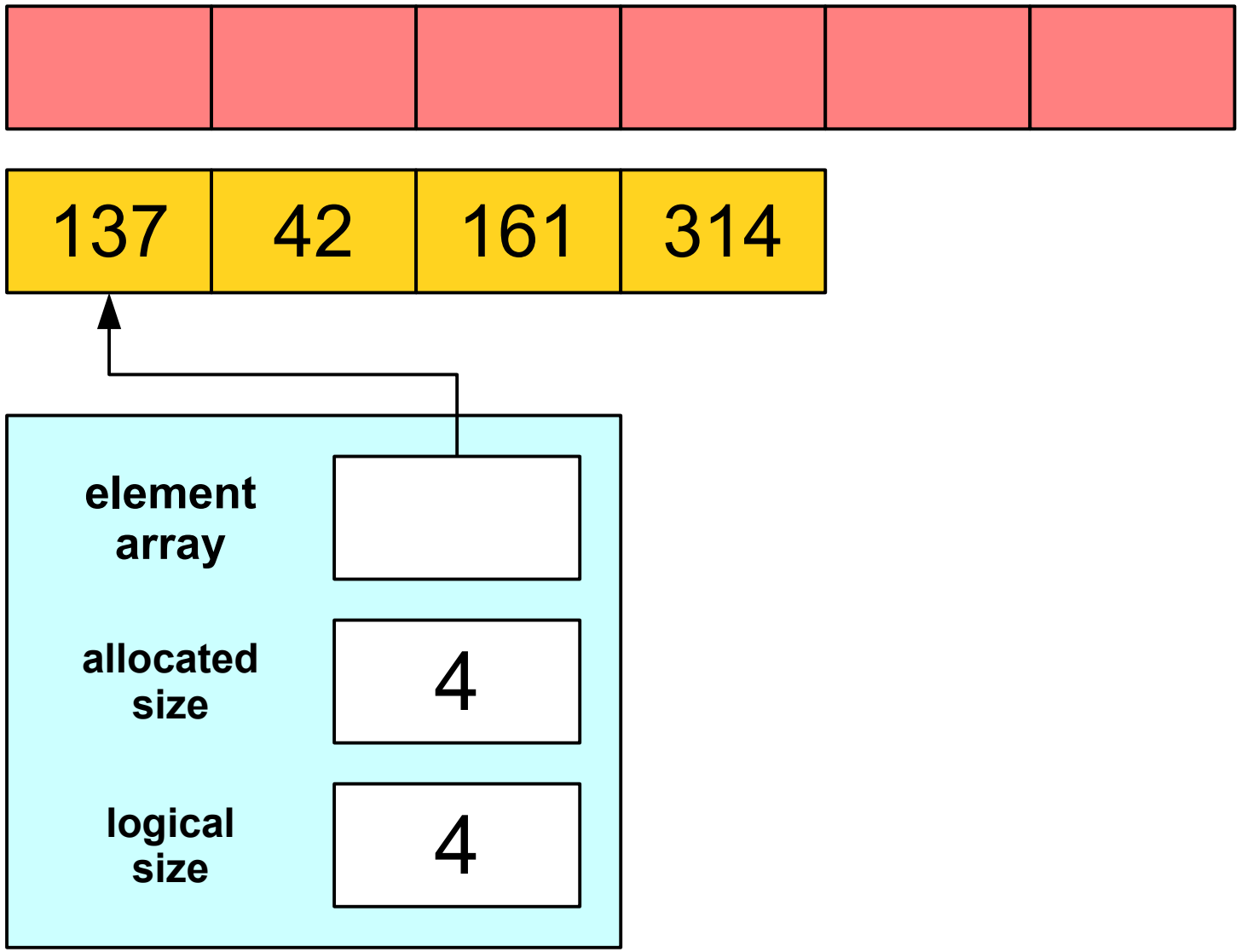
Speeding up the Stack

Key Idea: ***Plan for the Future***

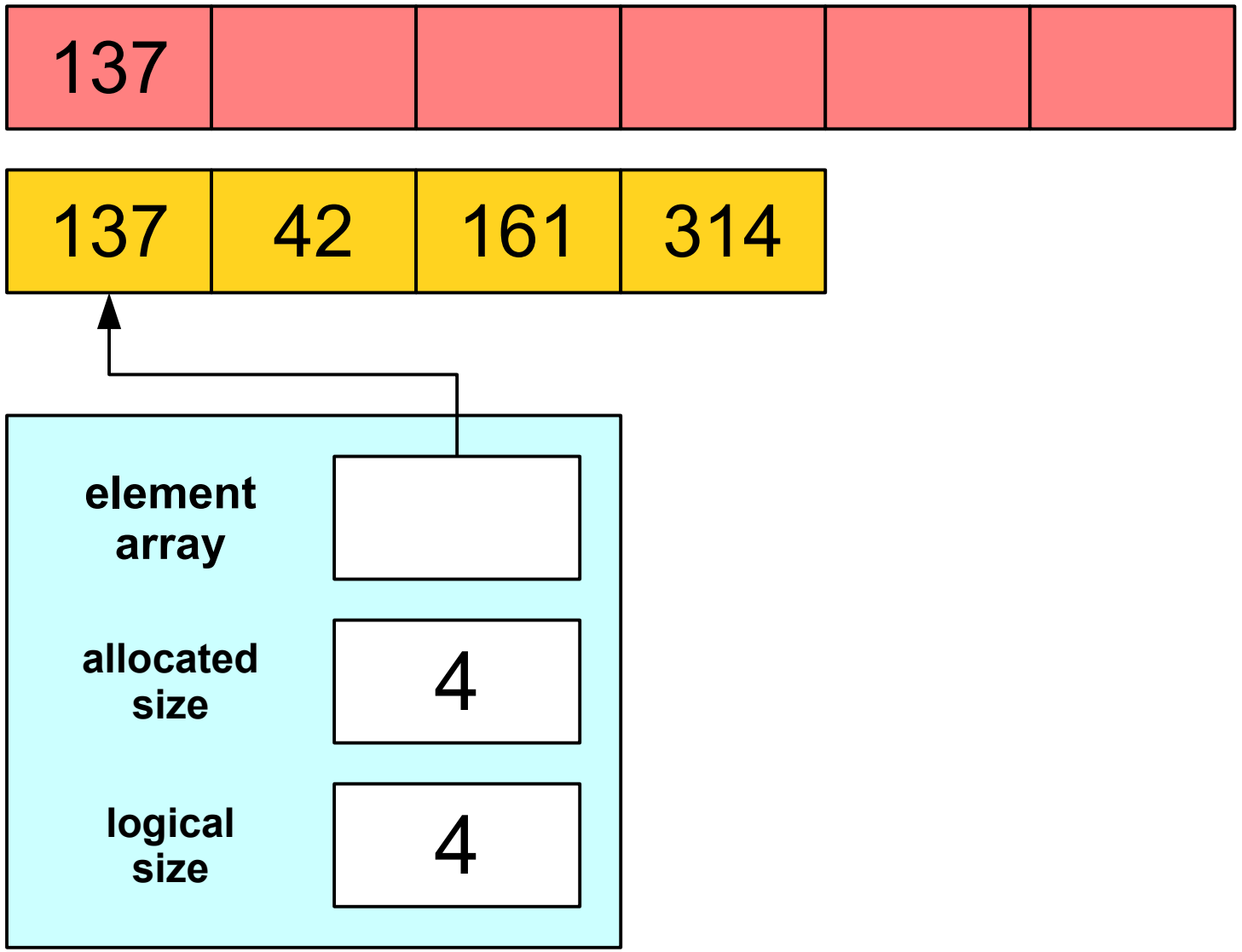
A Better Idea



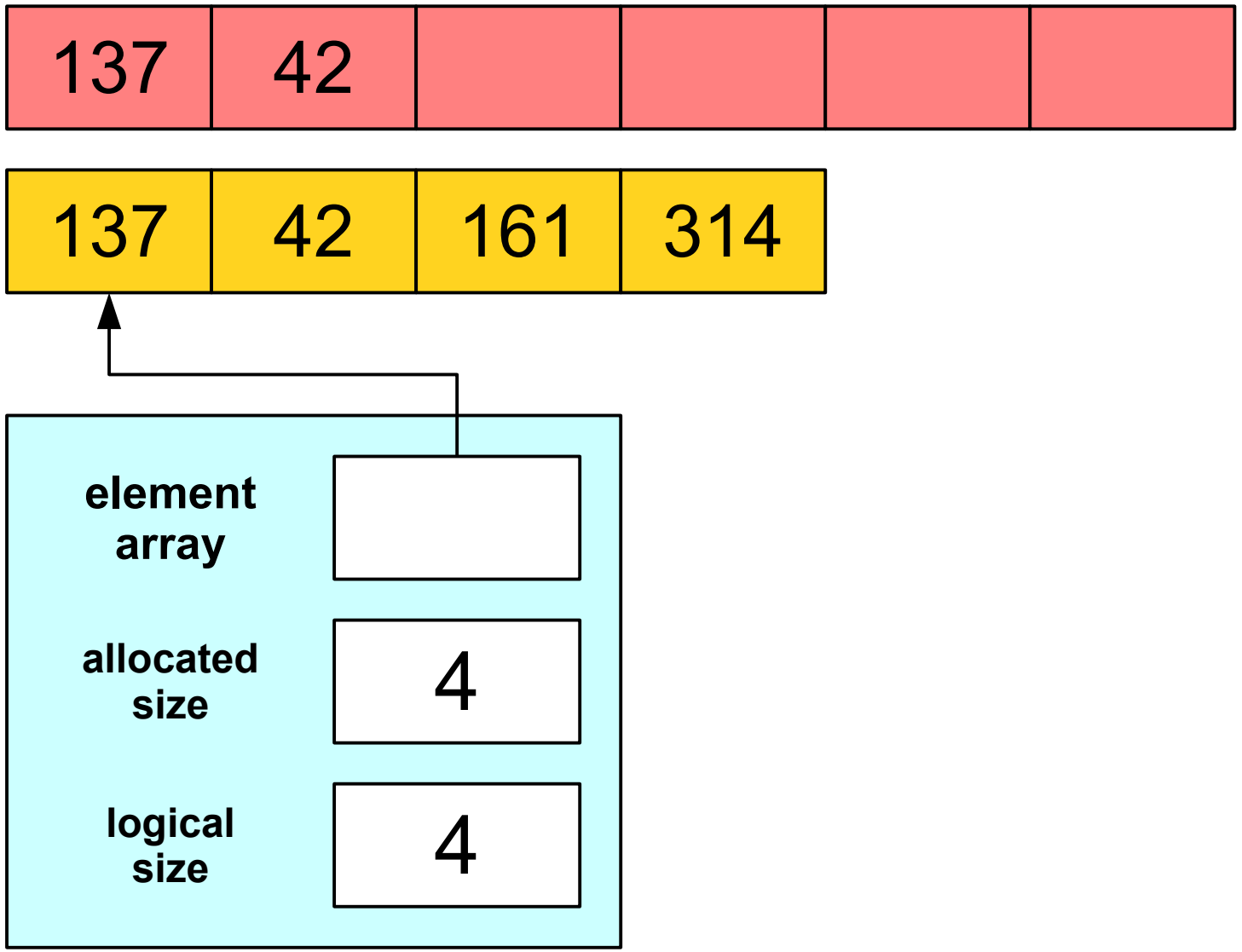
A Better Idea



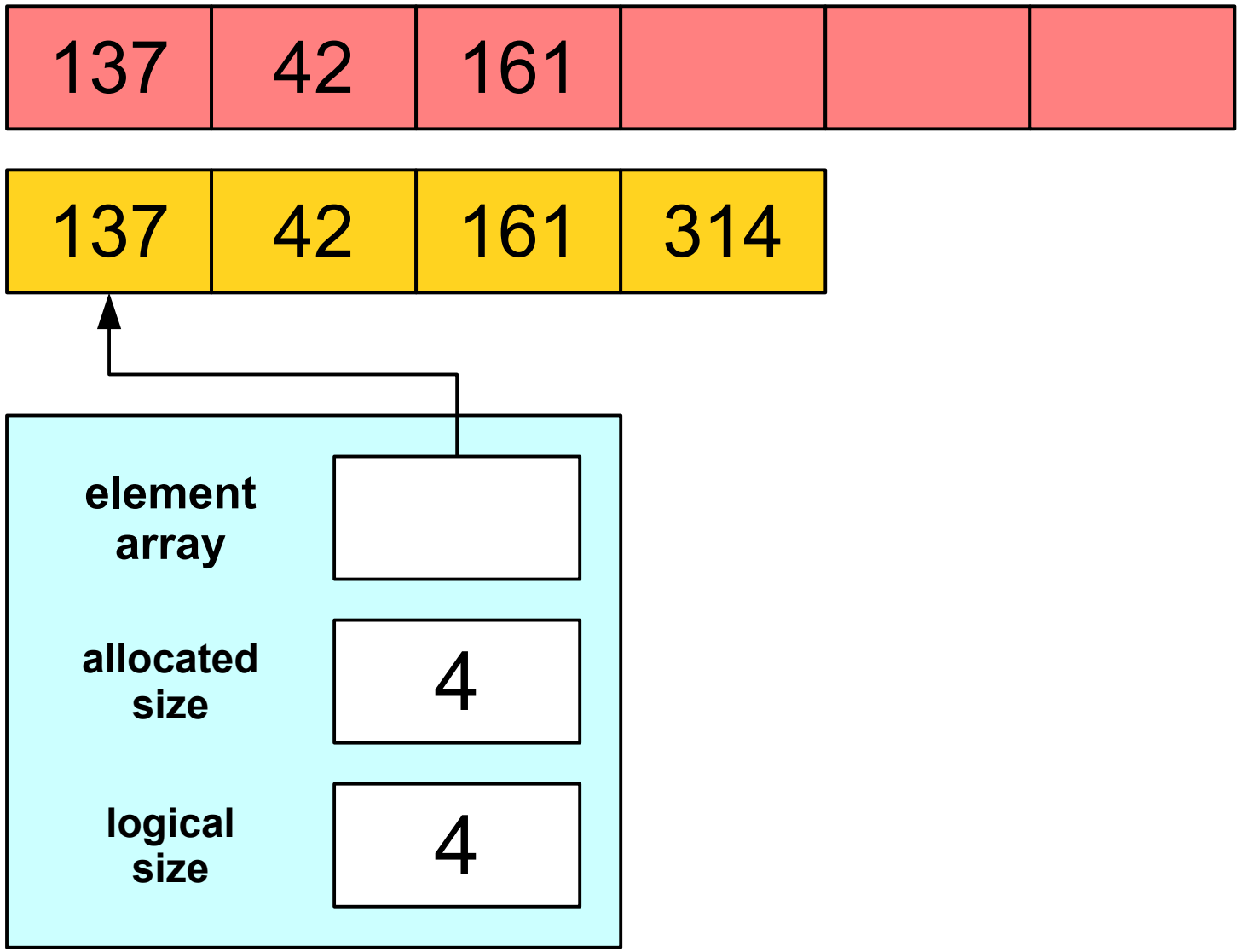
A Better Idea



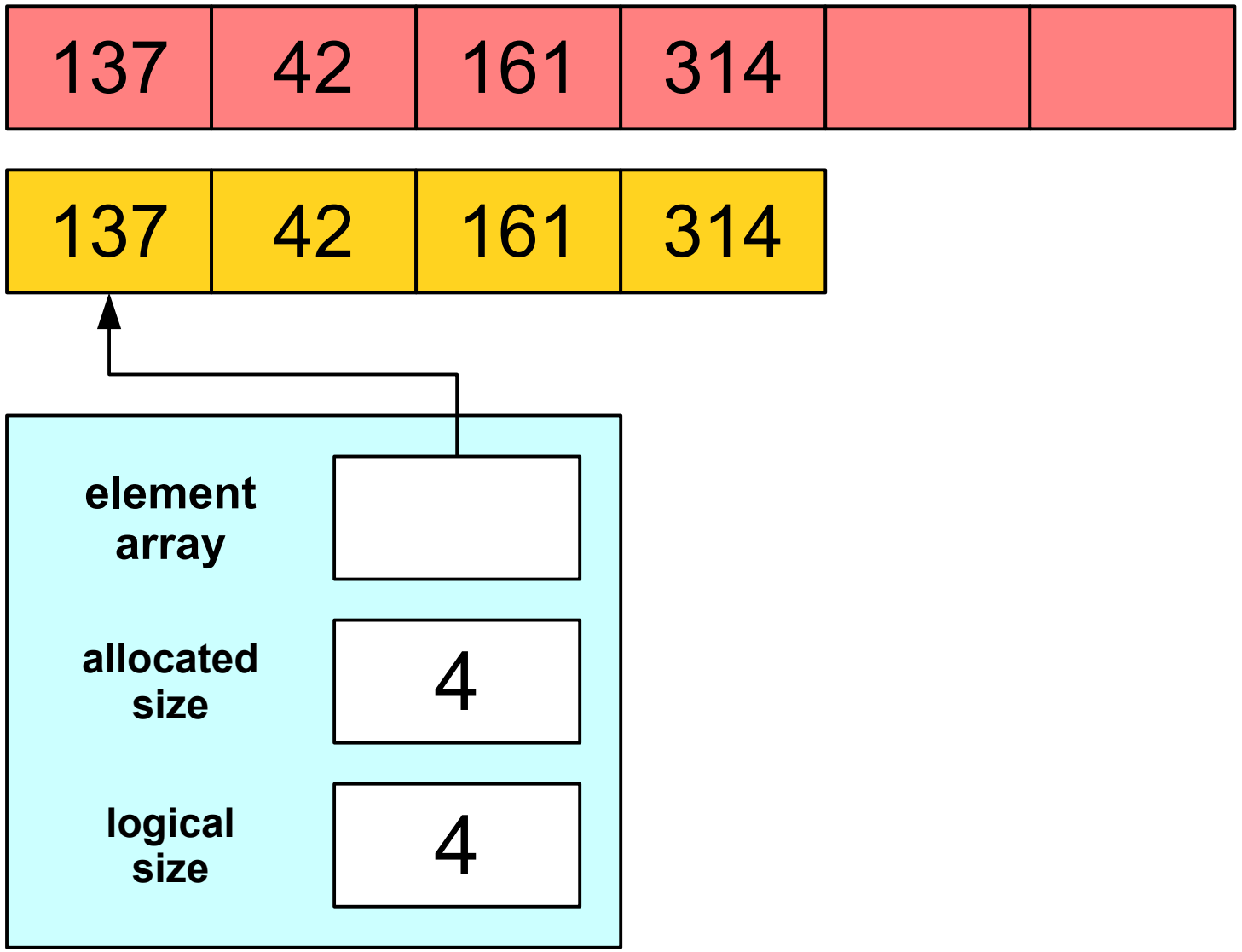
A Better Idea



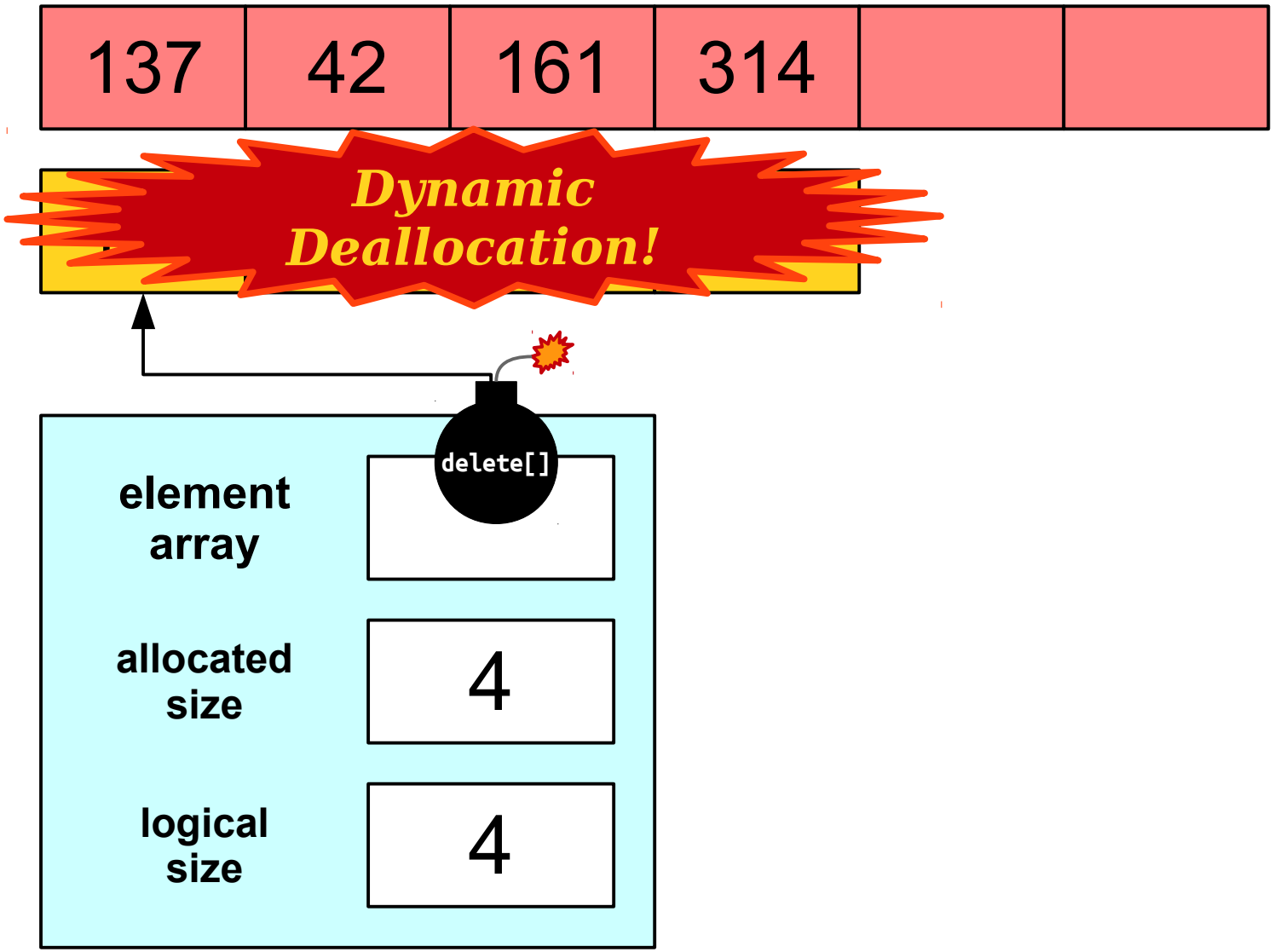
A Better Idea



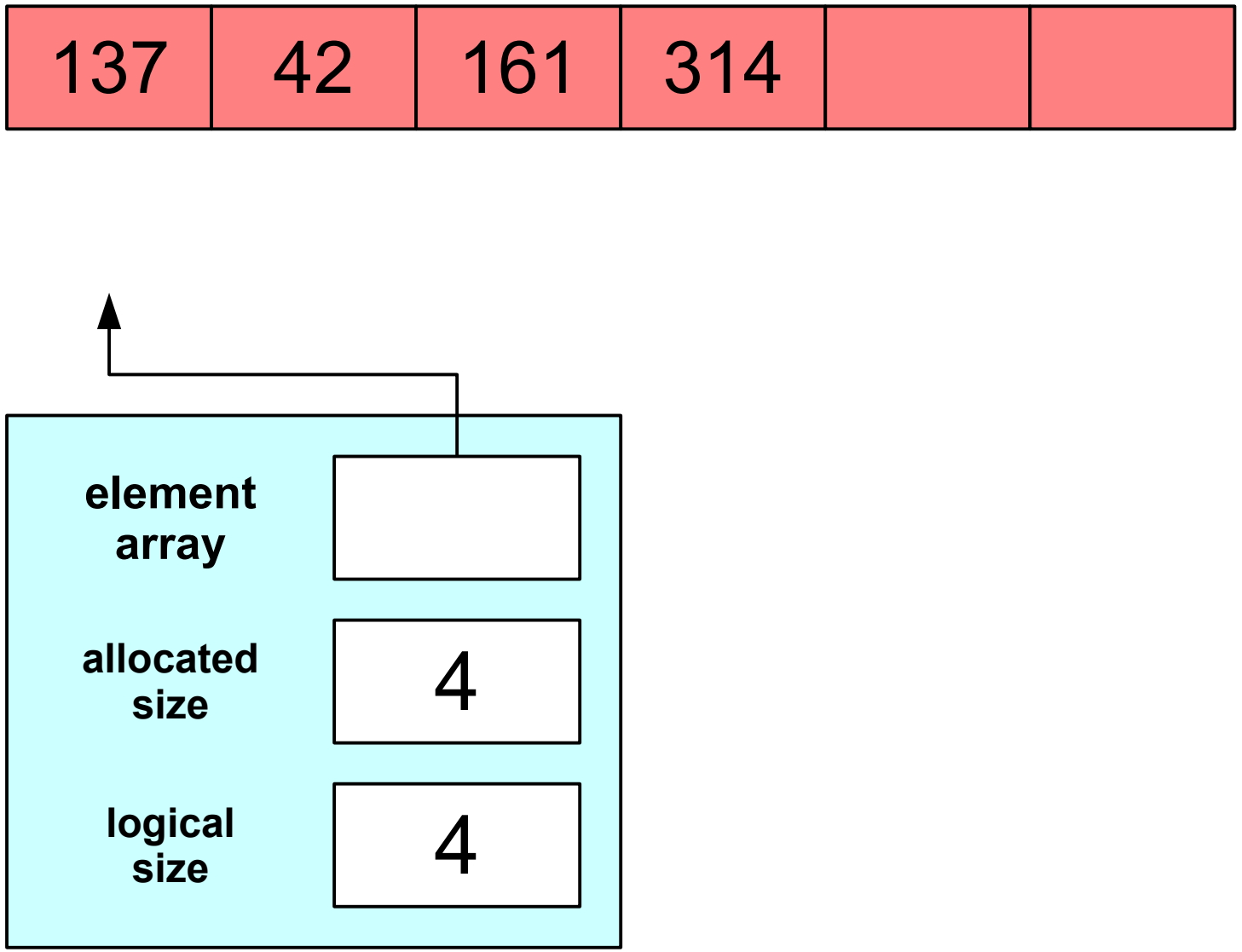
A Better Idea



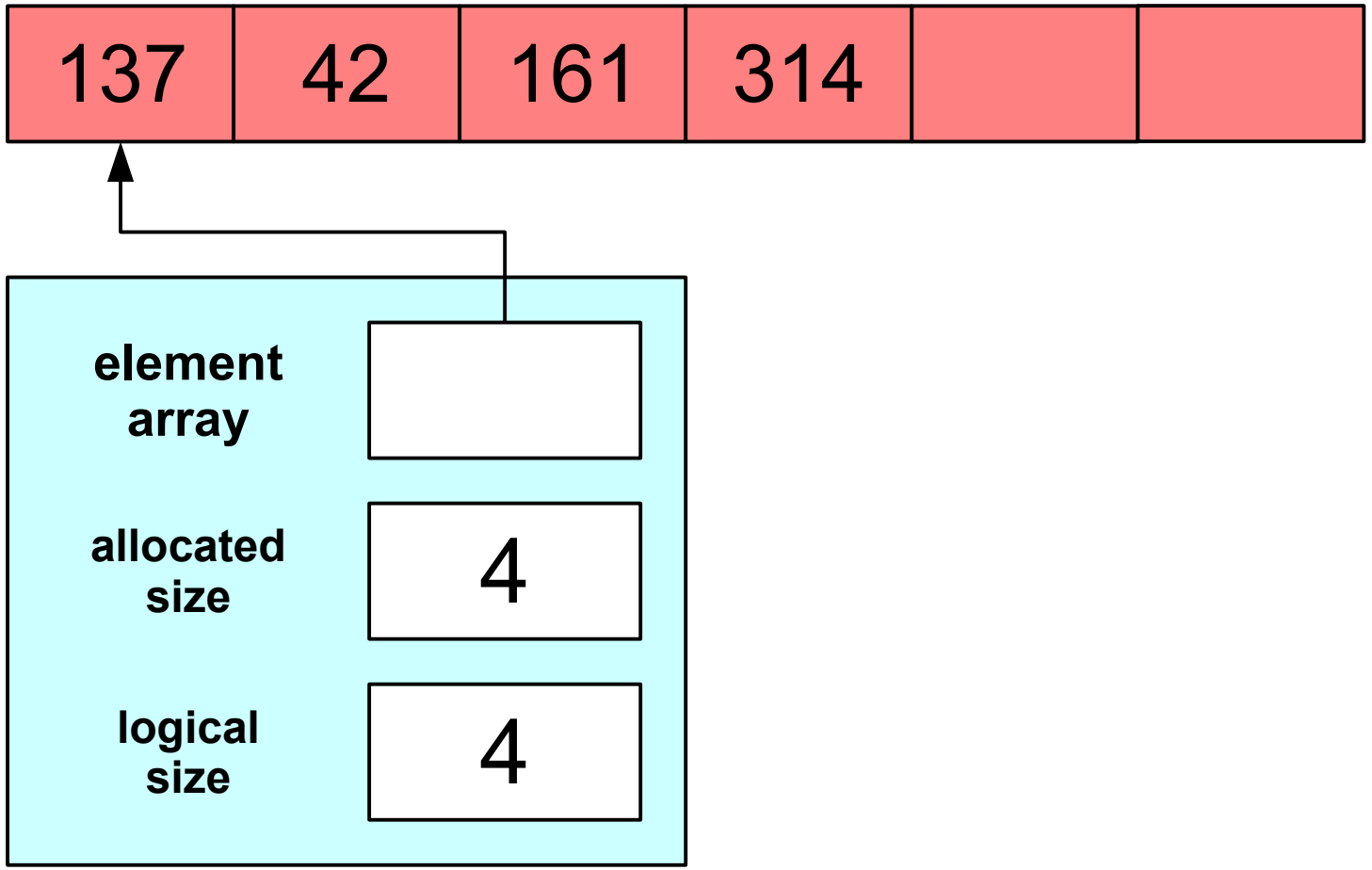
A Better Idea



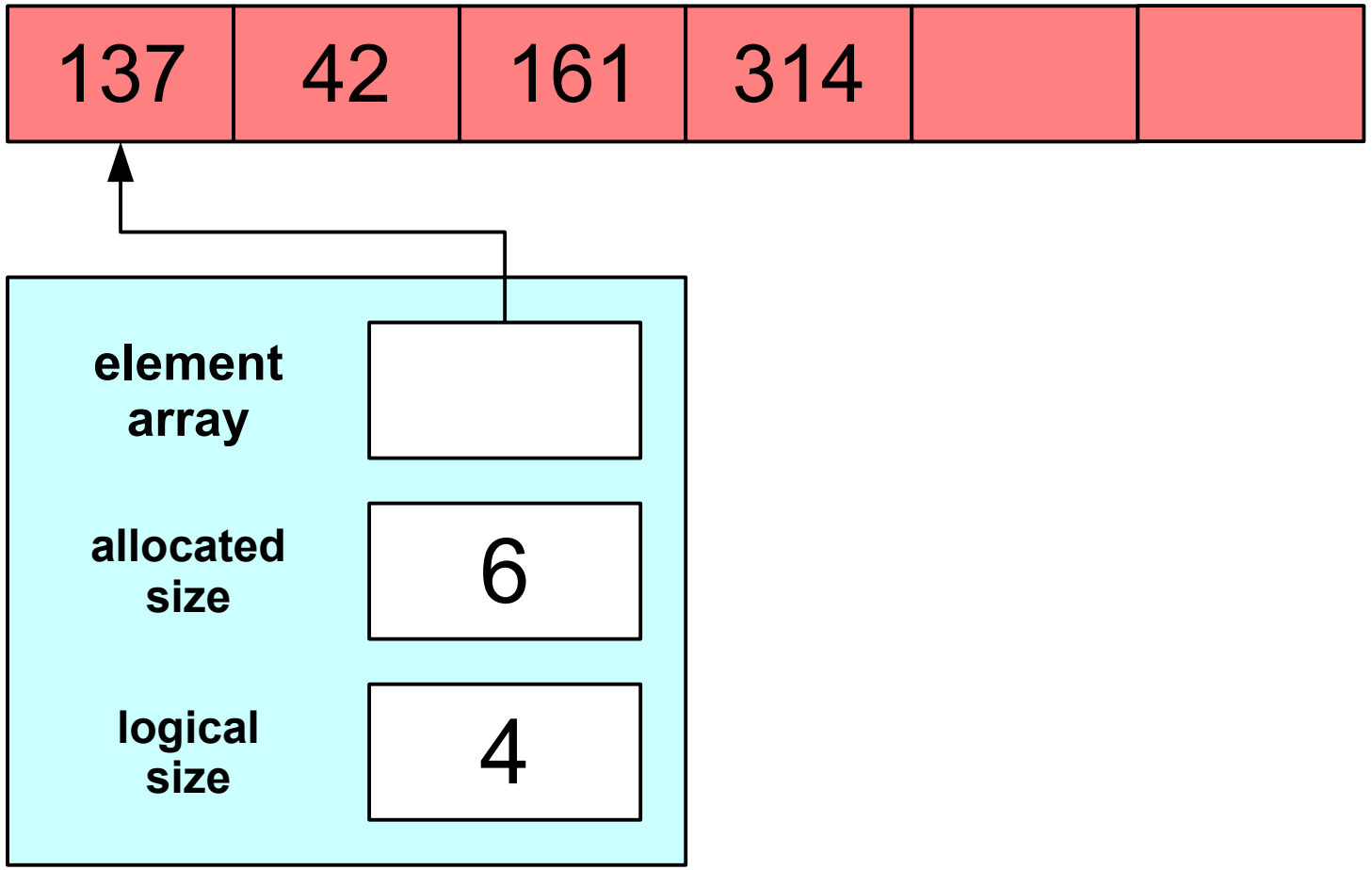
A Better Idea



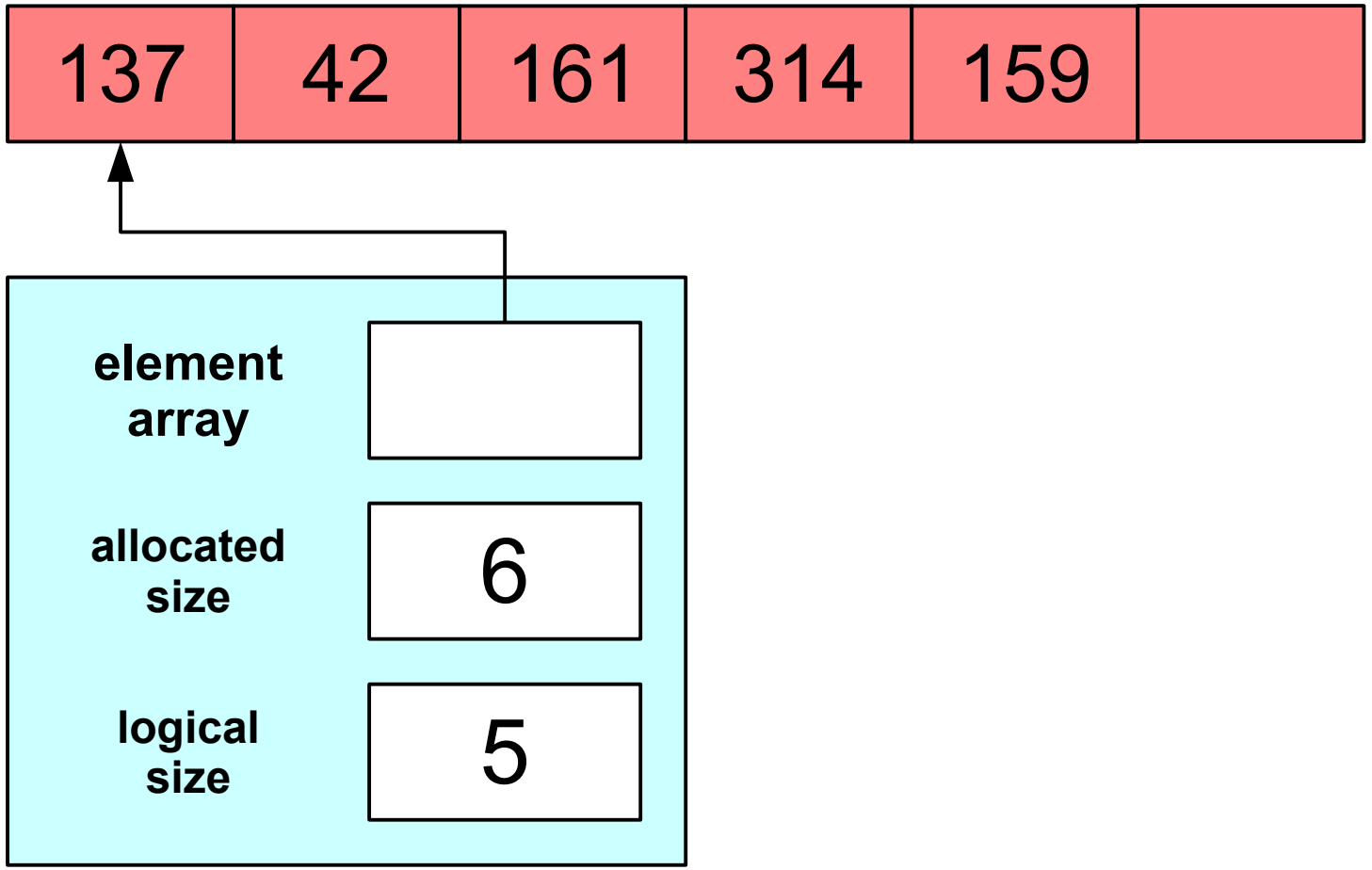
A Better Idea



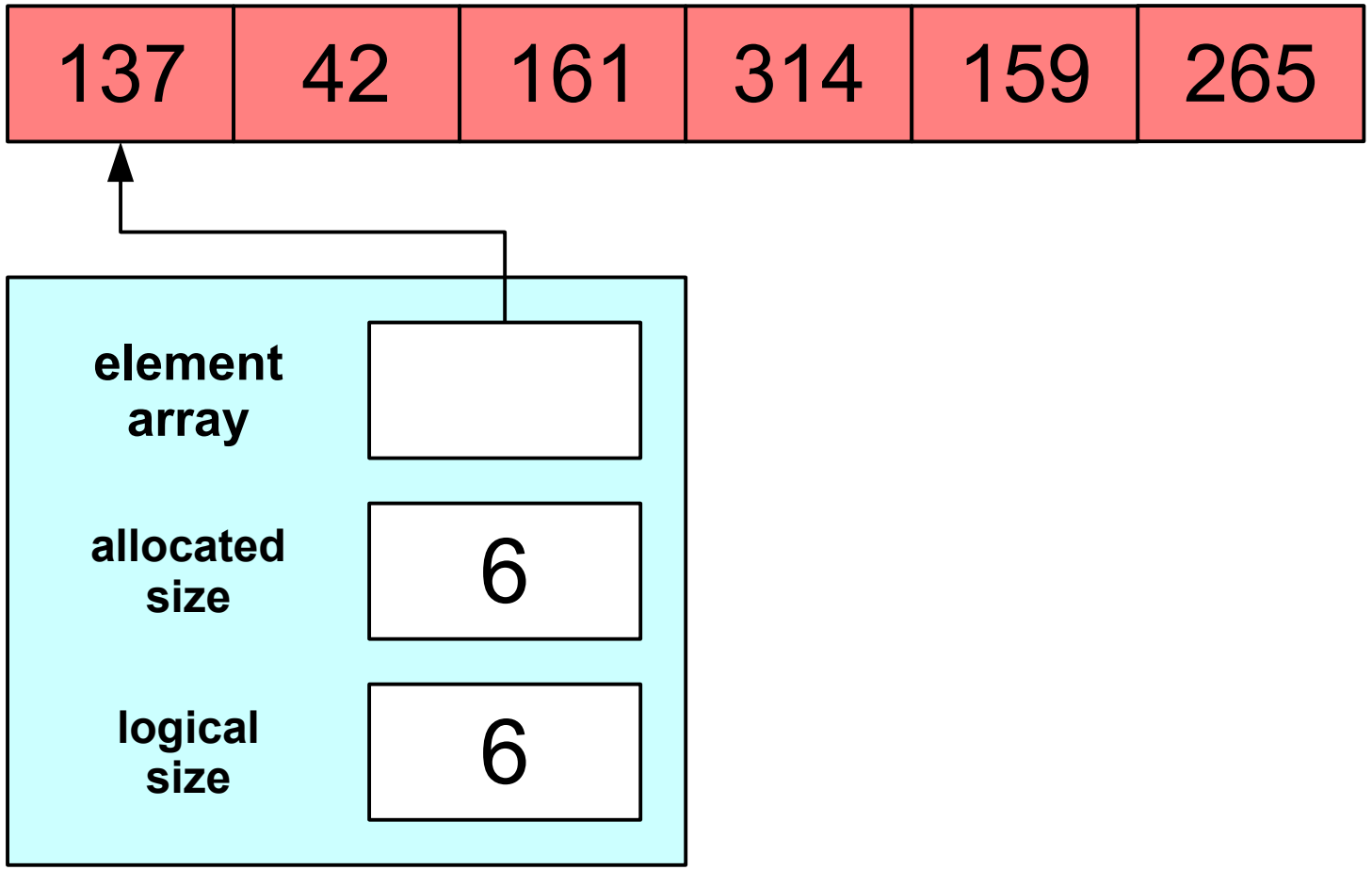
A Better Idea



A Better Idea



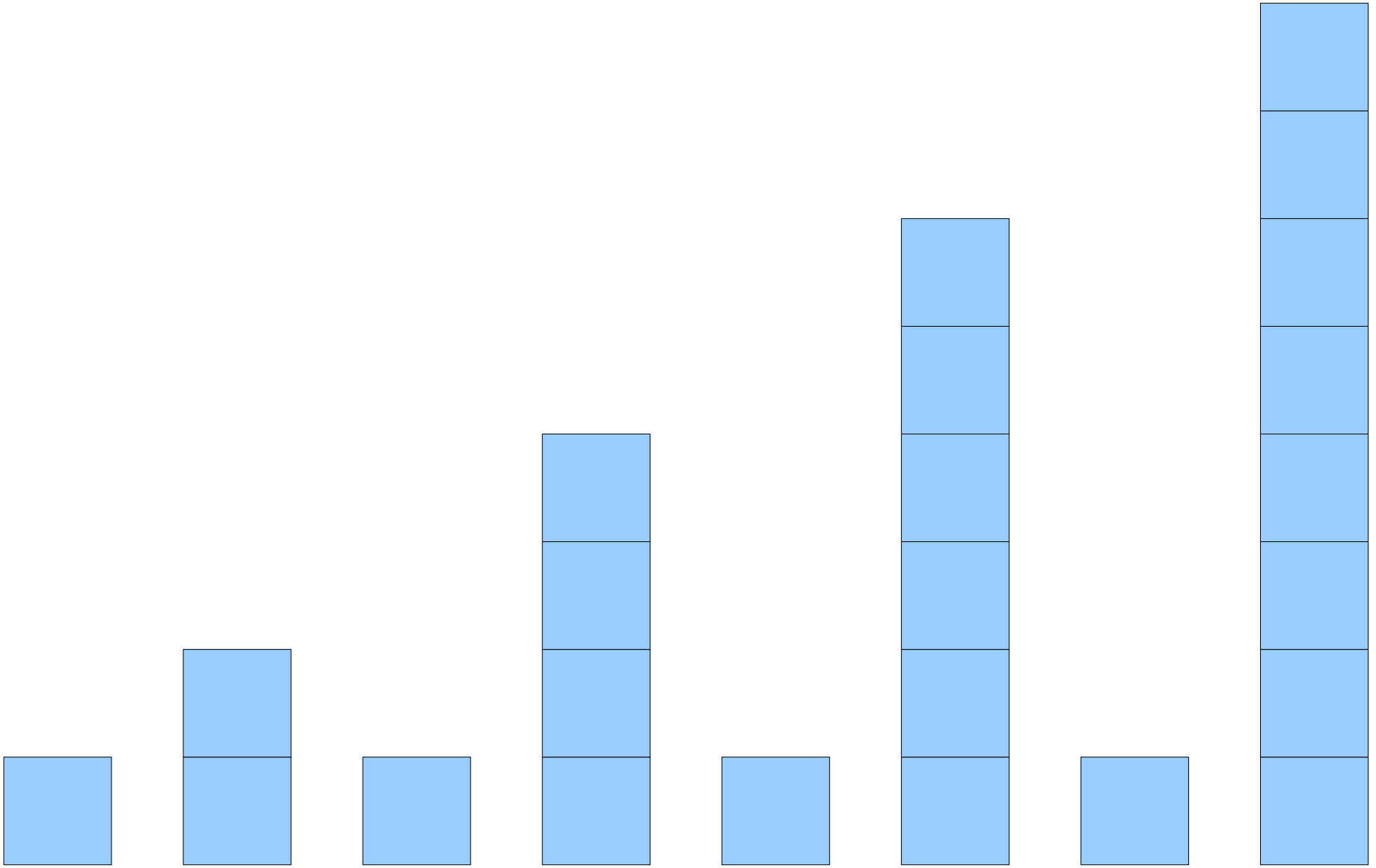
A Better Idea



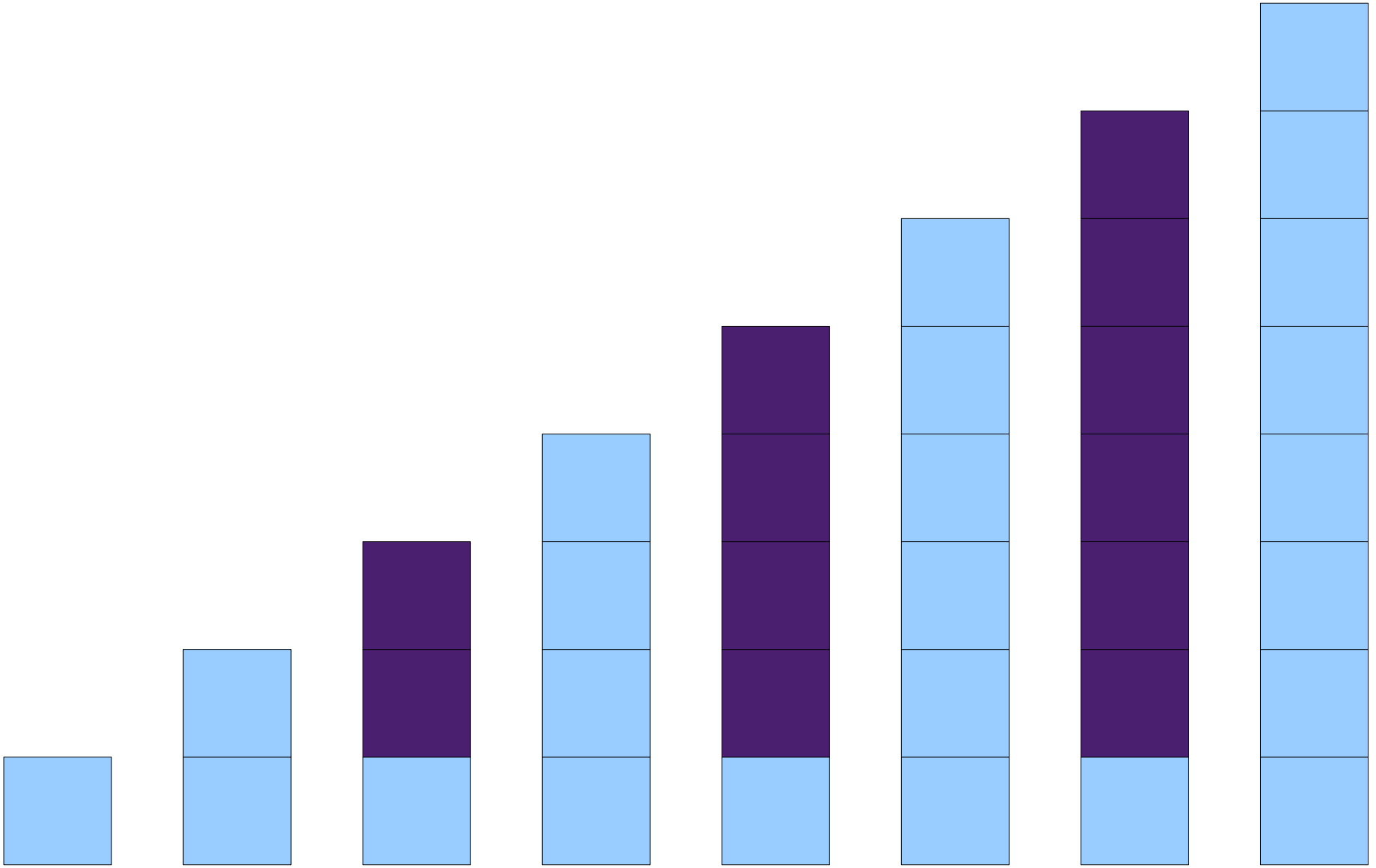
What Just Happened?

- Half of our pushes are now “easy” pushes, and half of our pushes are now “hard” pushes.
- Hard pushes still take time O(n).
- Easy pushes only take time O(1).
- Worst-case is still O(n).
- What about the average case?

Analyzing the Work

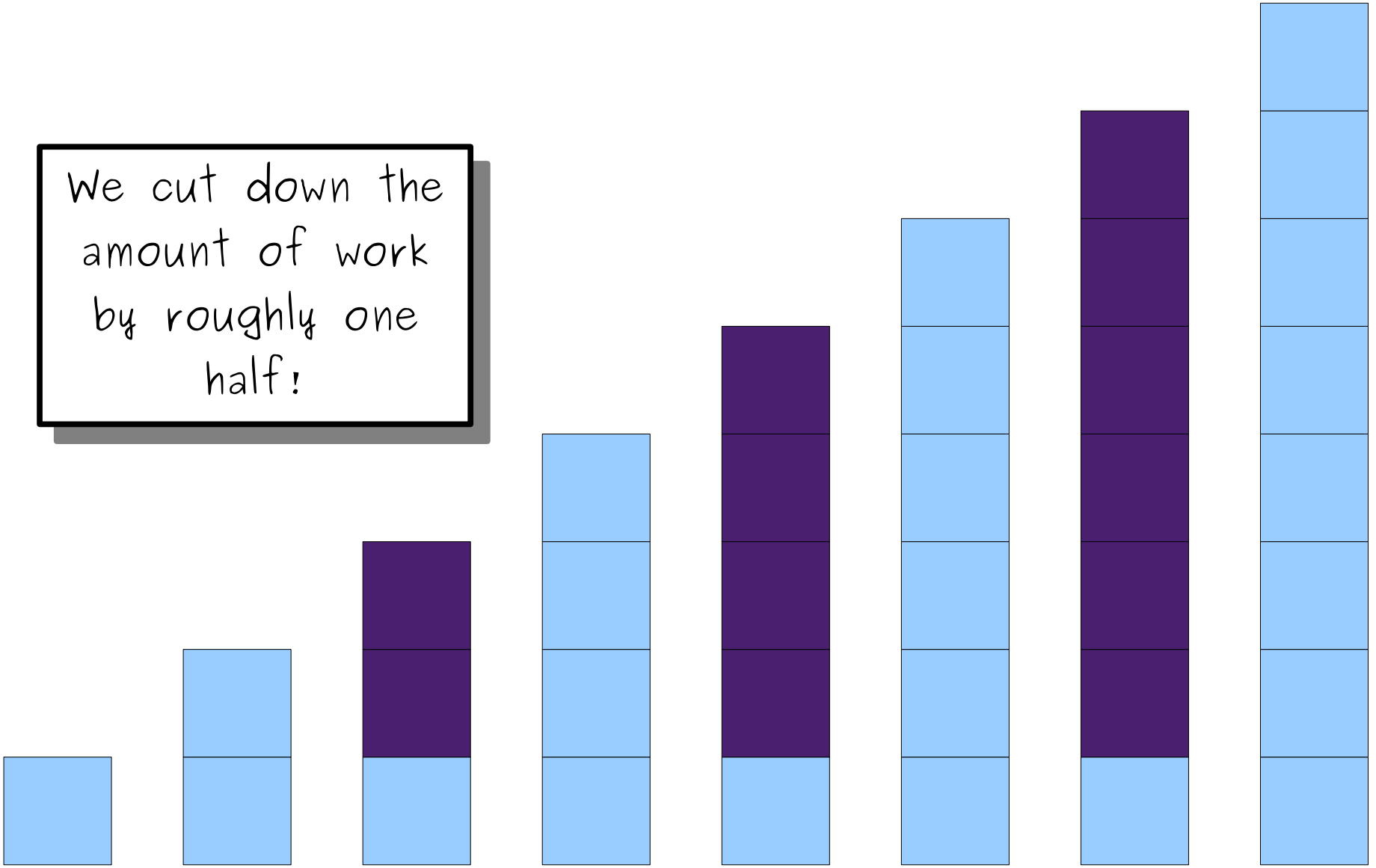


Analyzing the Work



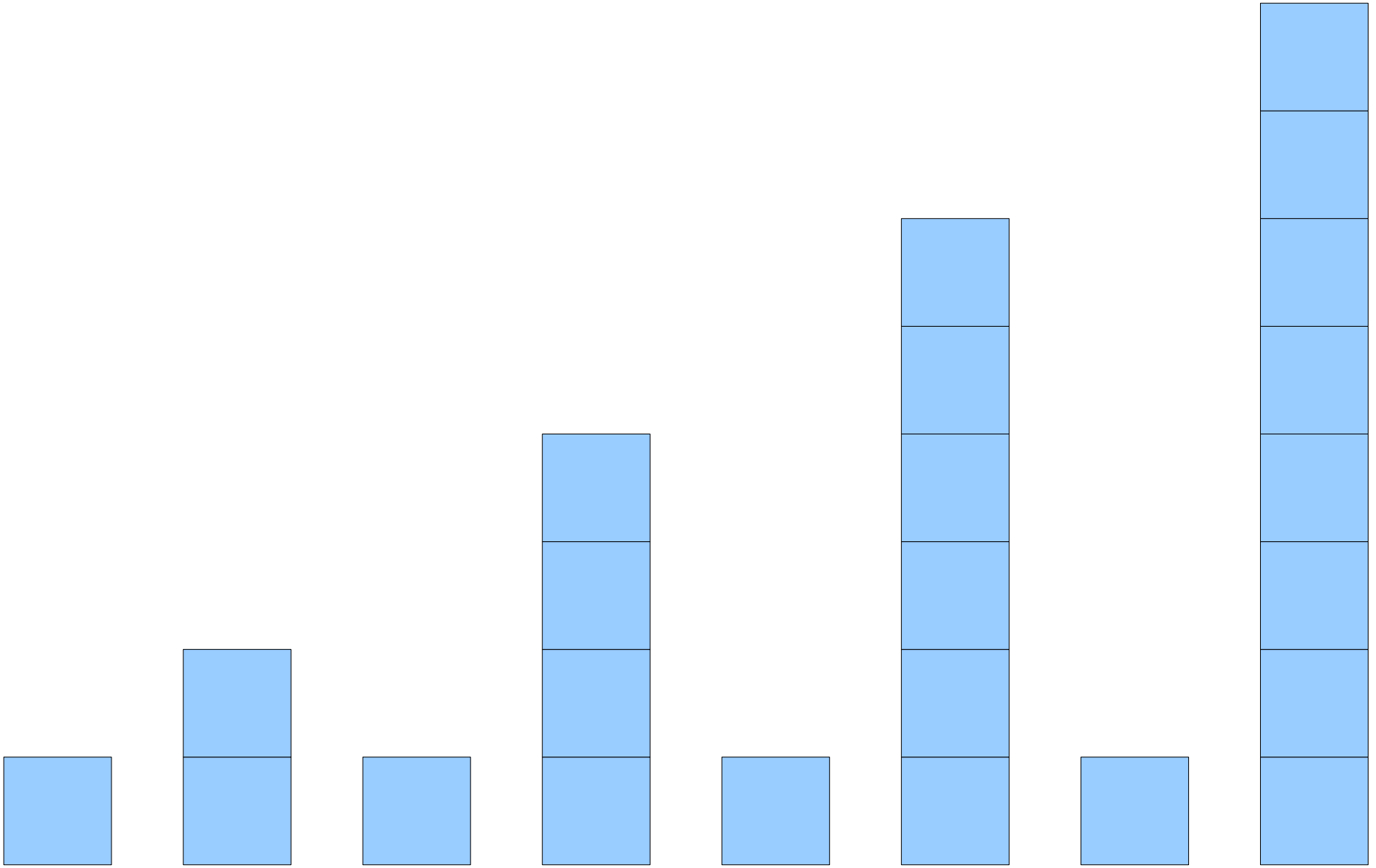
Analyzing the Work

We cut down the amount of work by roughly one half!

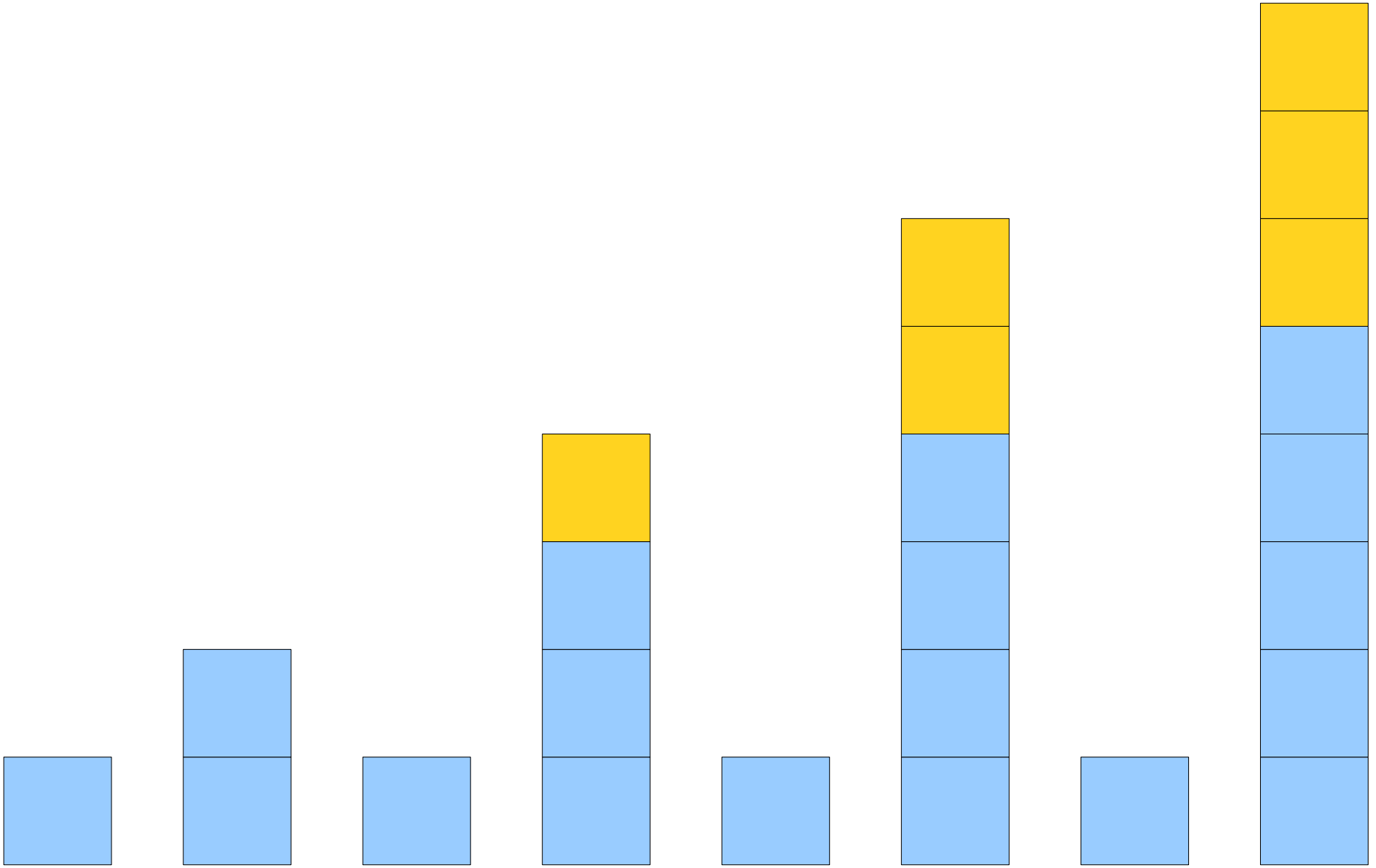


A Different Analysis

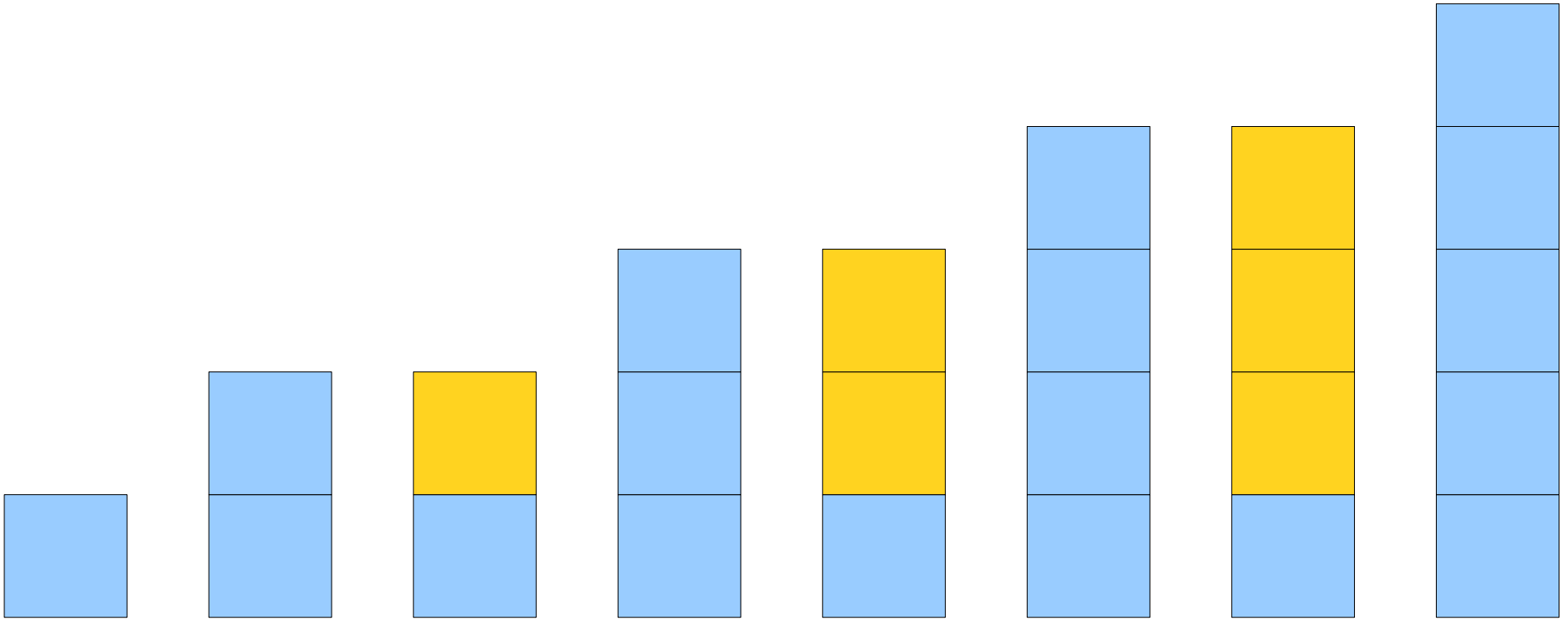
A Different Analysis



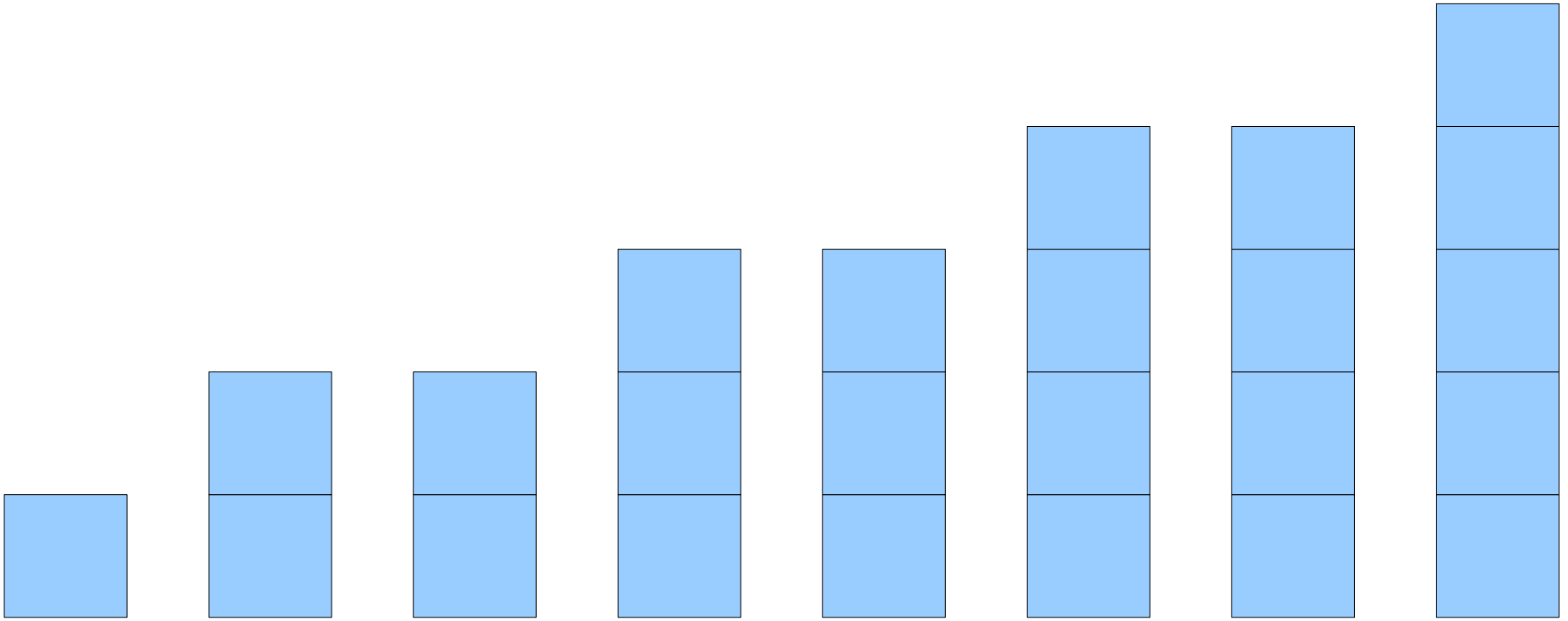
A Different Analysis



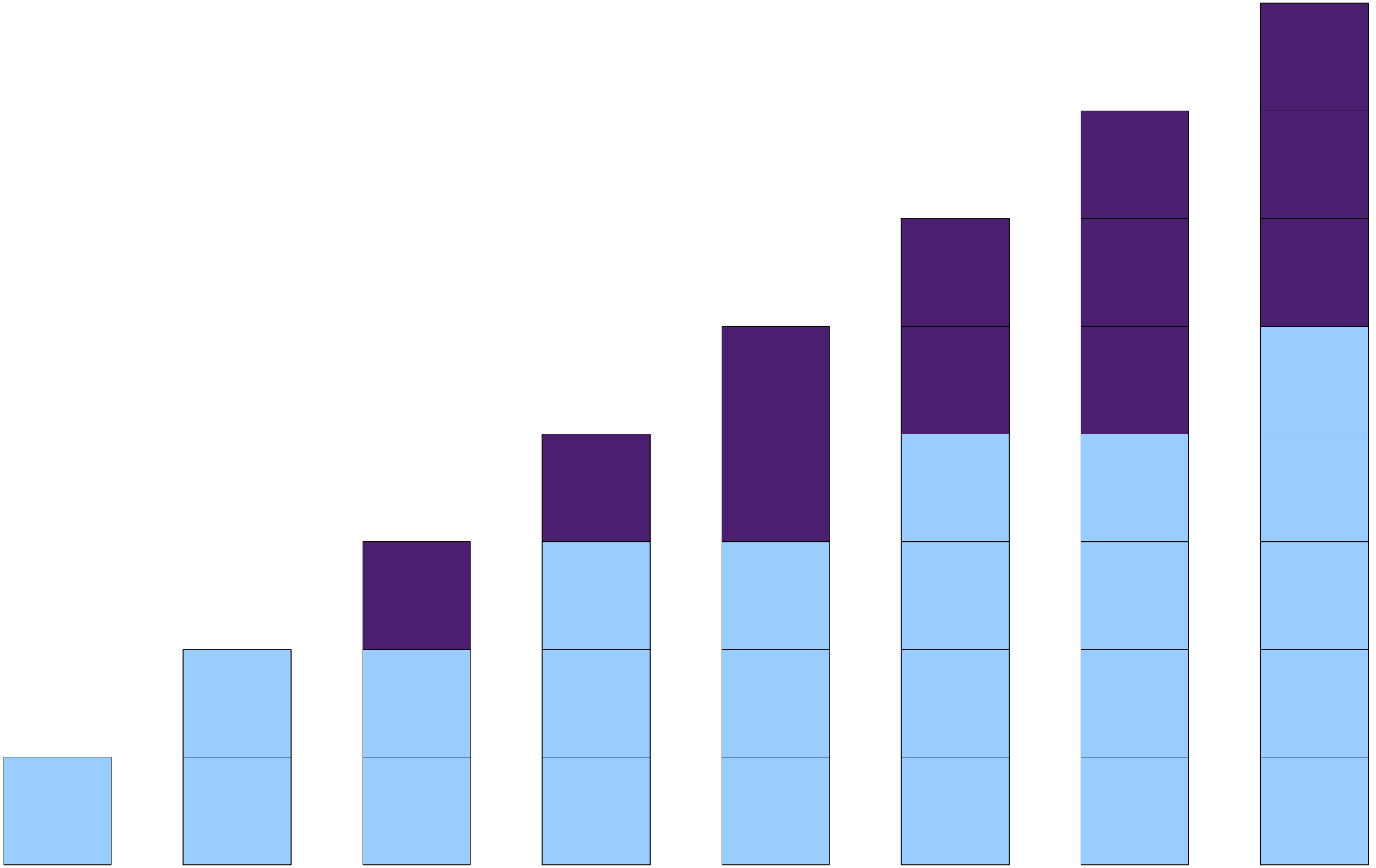
A Different Analysis



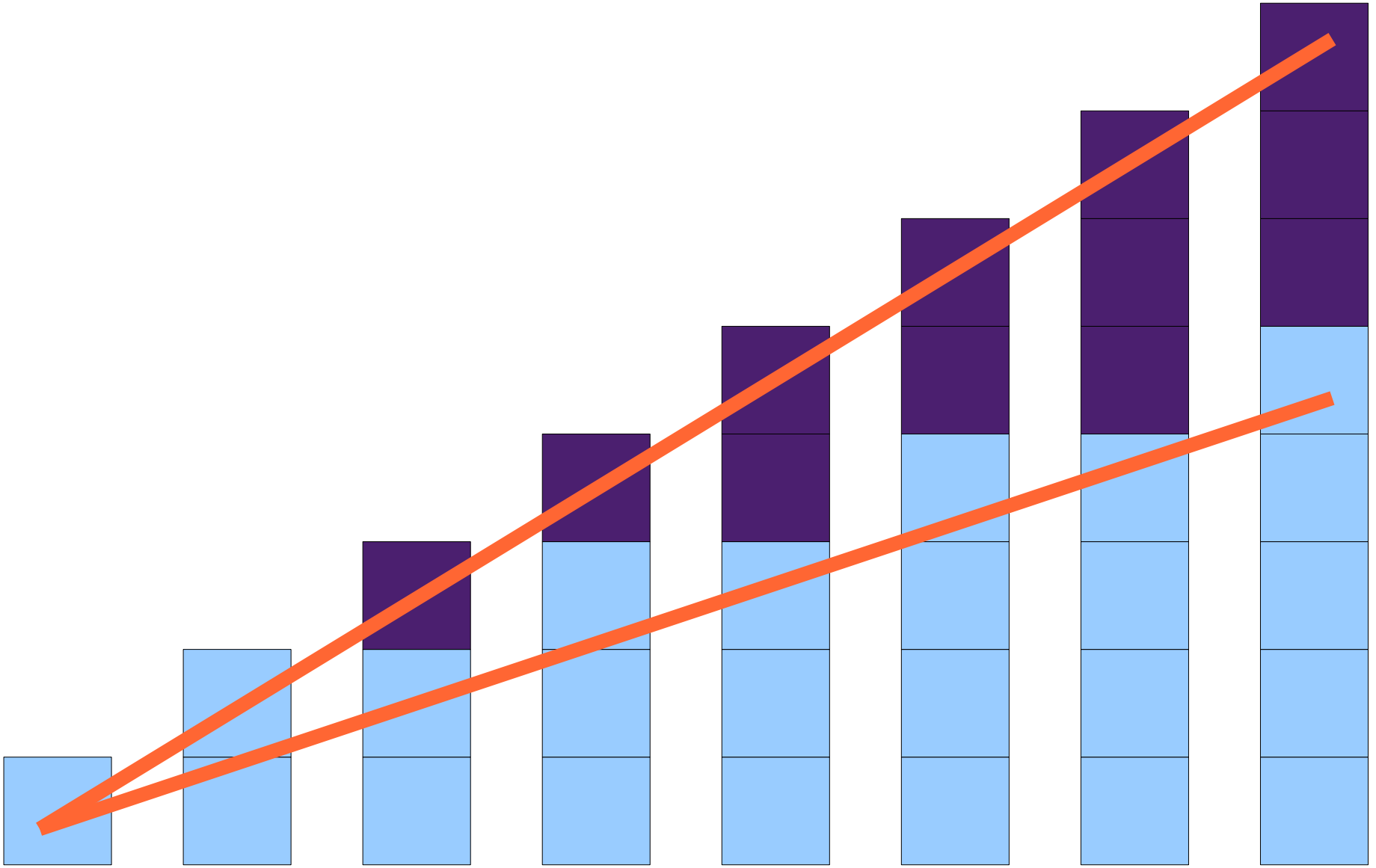
A Different Analysis



A Different Analysis

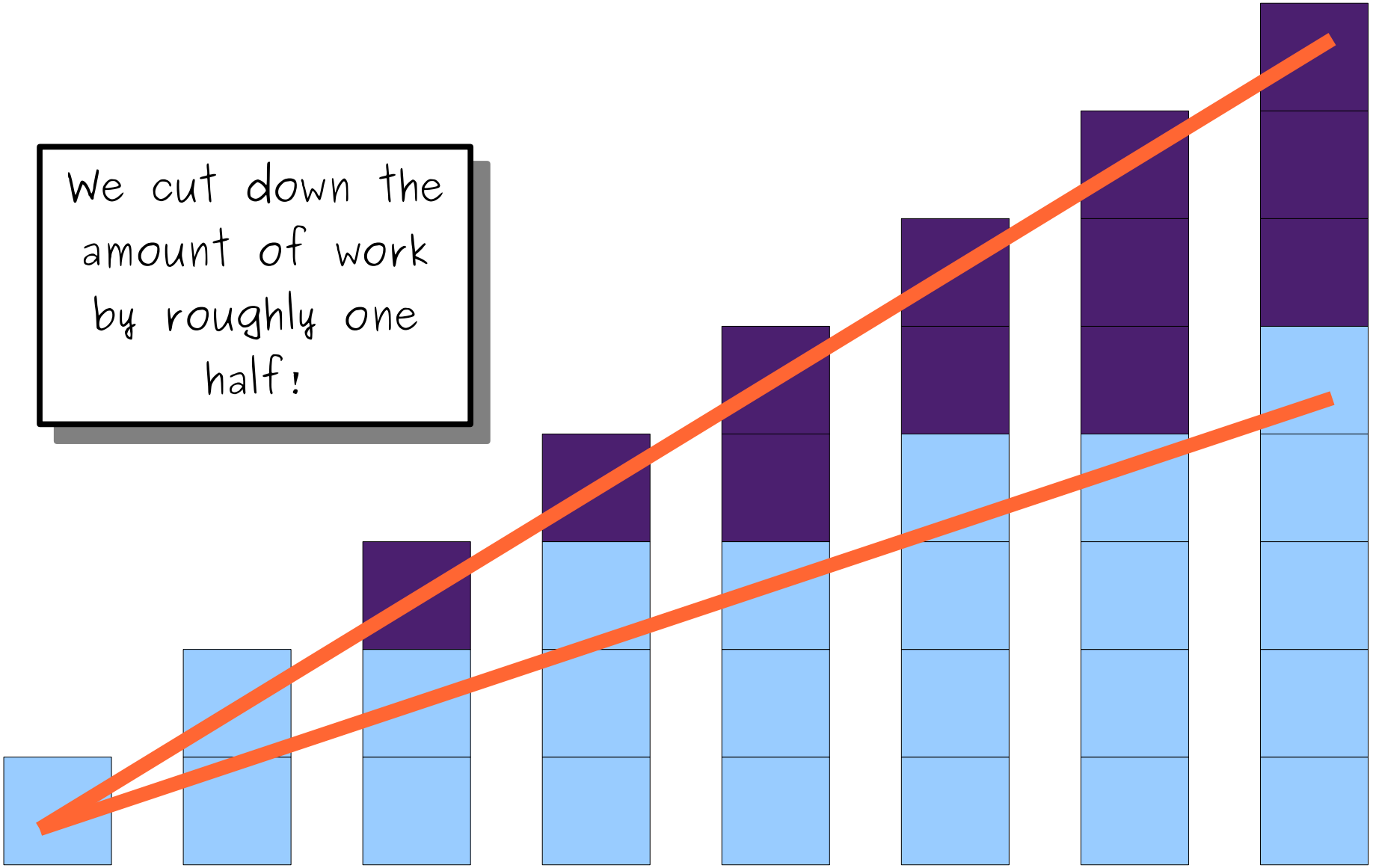


A Different Analysis



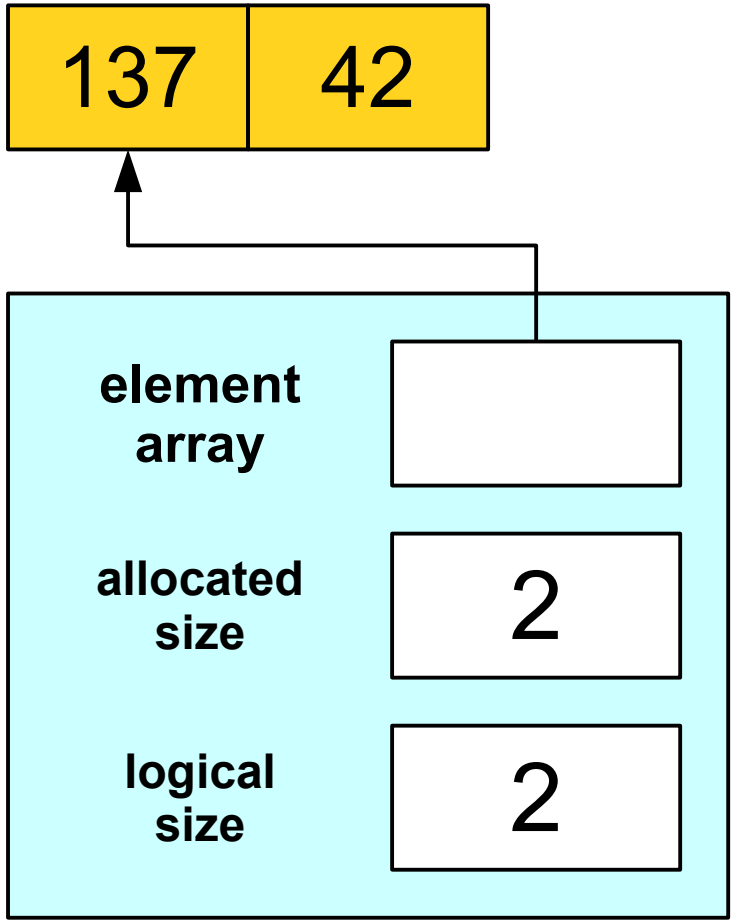
A Different Analysis

We cut down the amount of work by roughly one half!

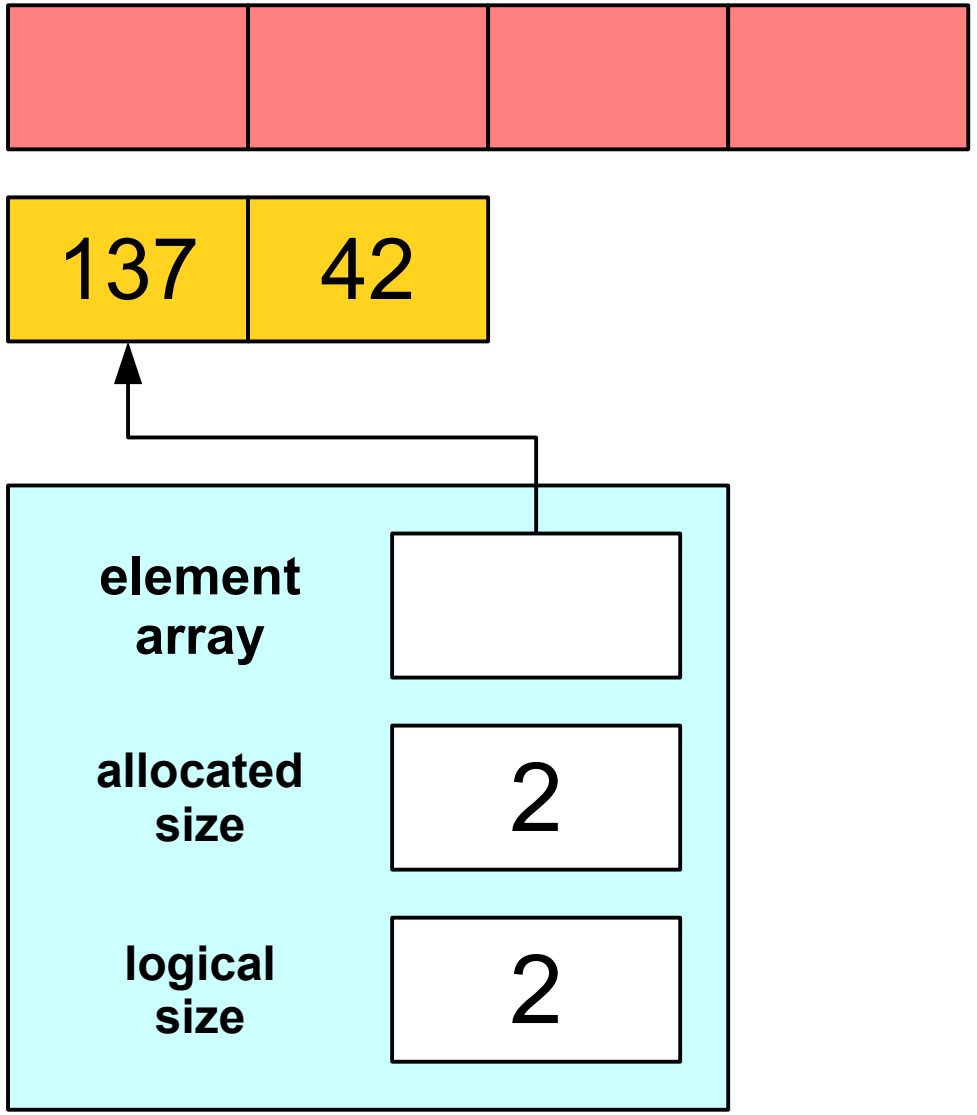


How does it stack up?

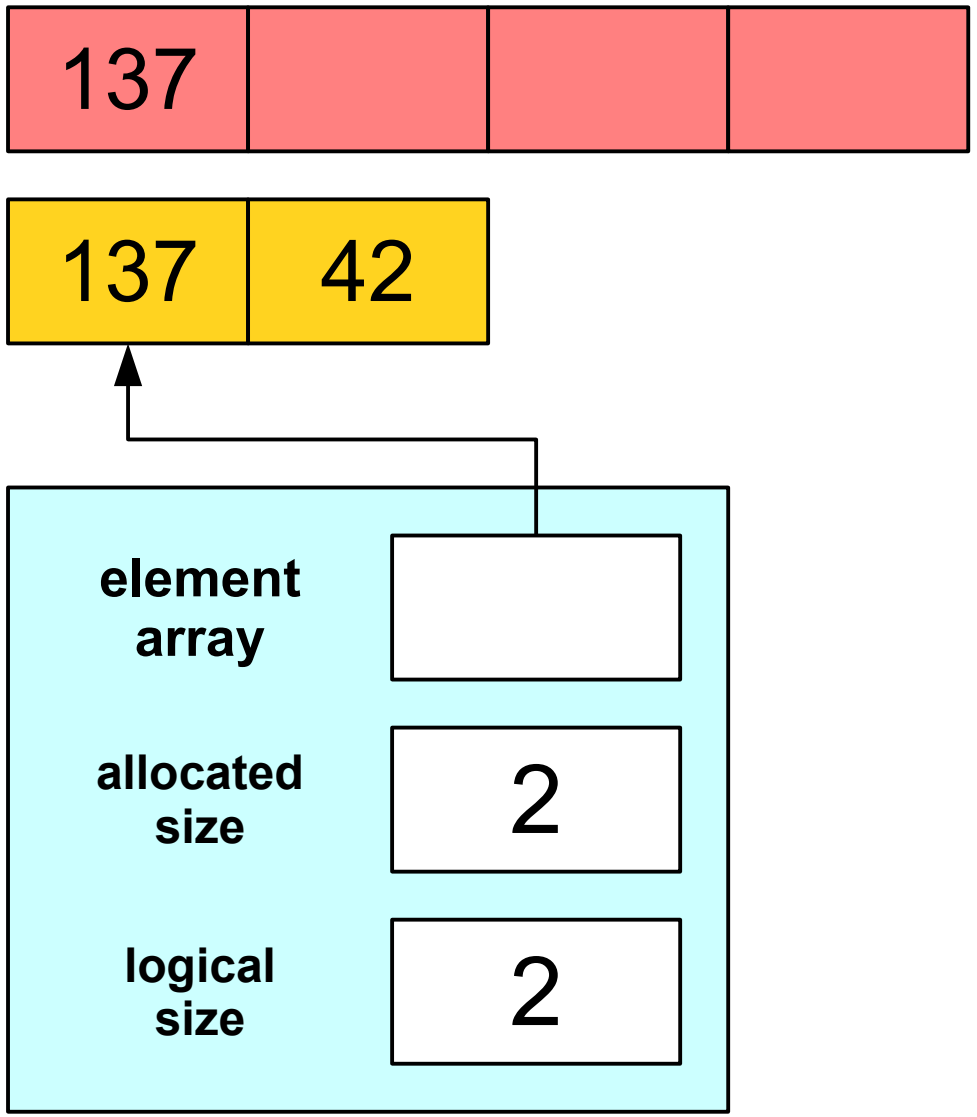
A Much Better Idea



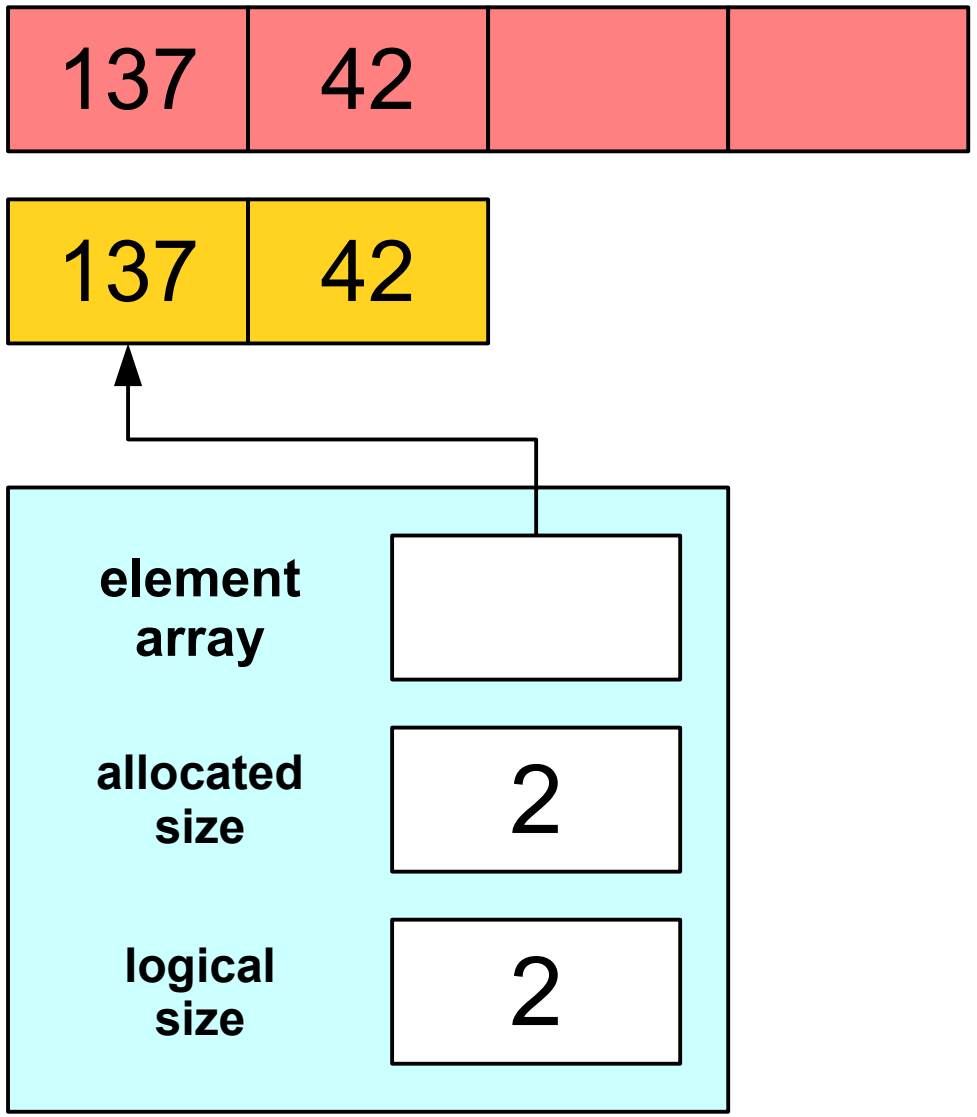
A Much Better Idea



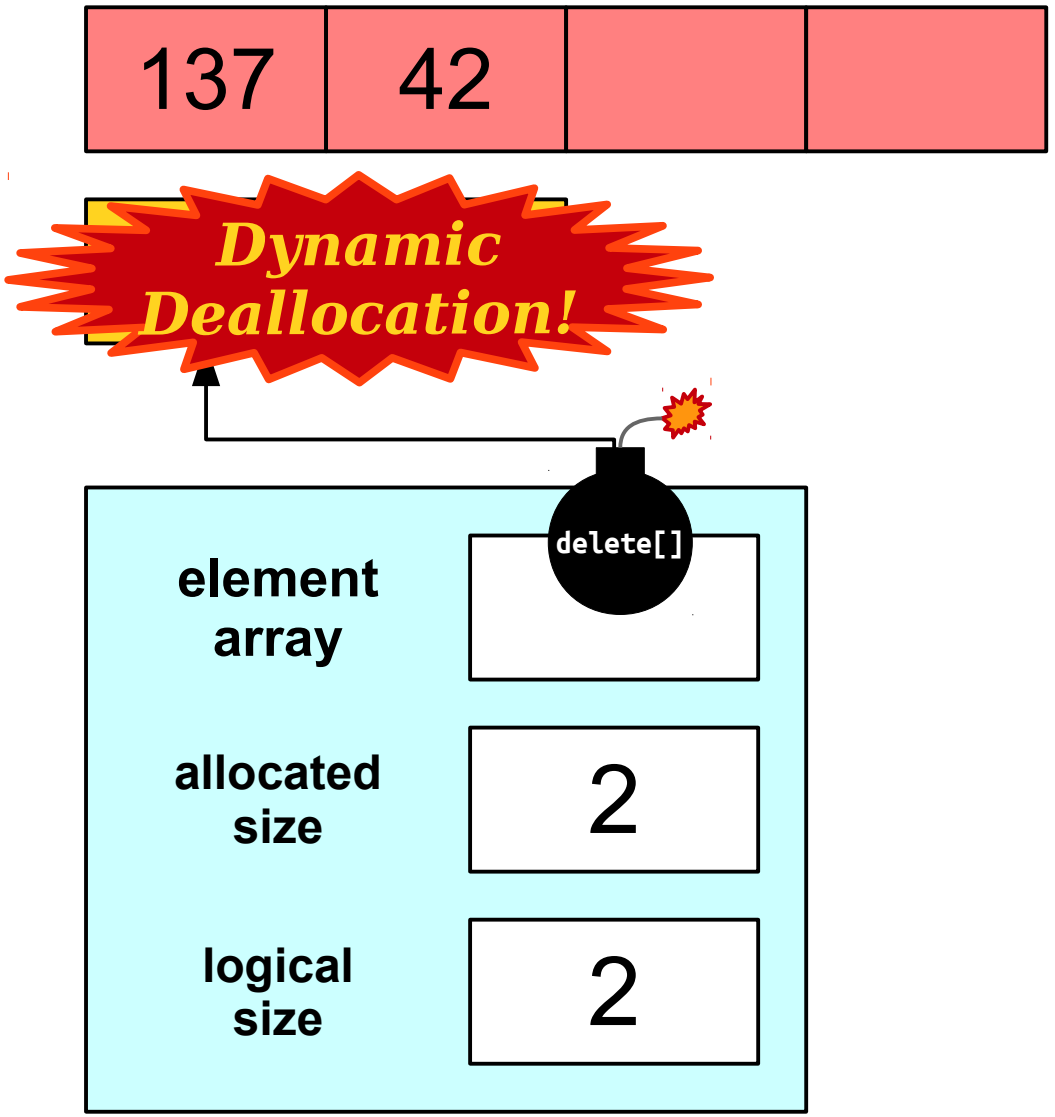
A Much Better Idea



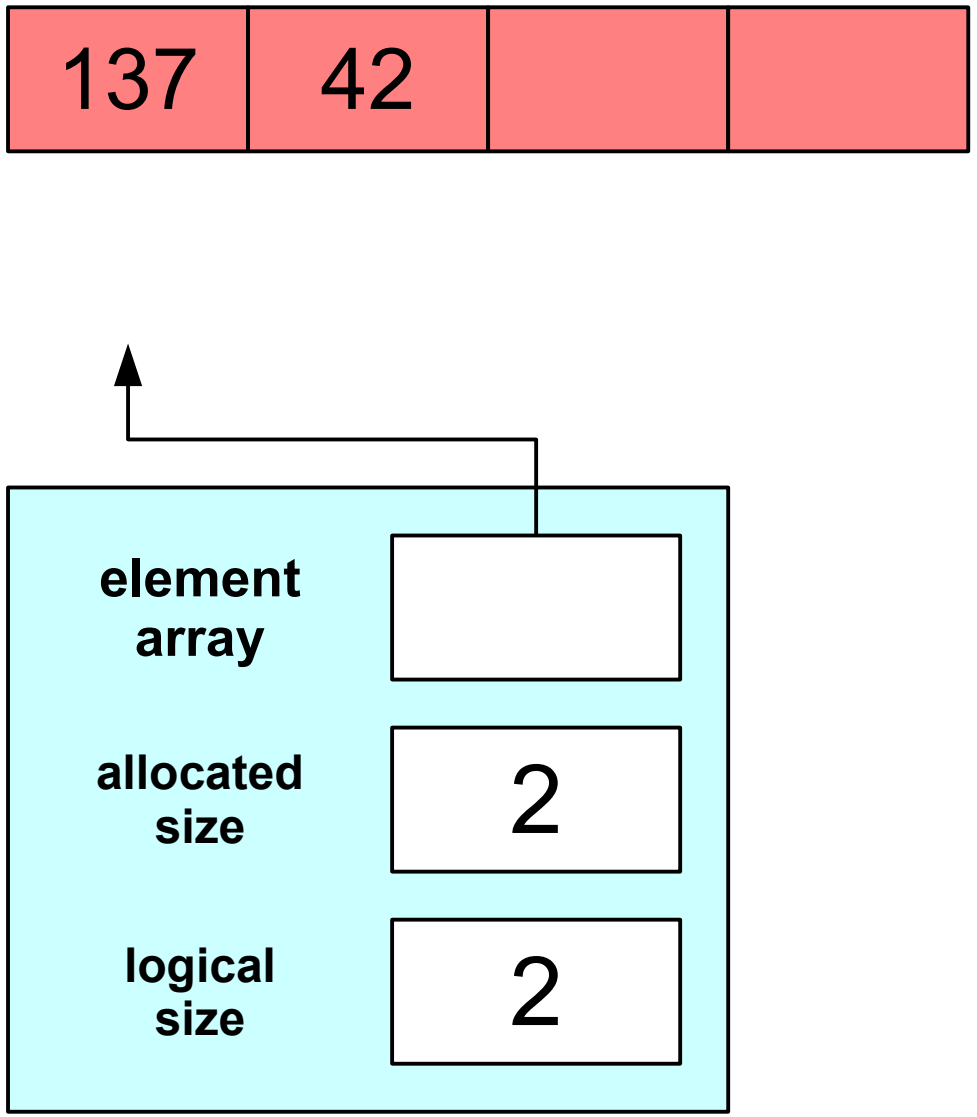
A Much Better Idea



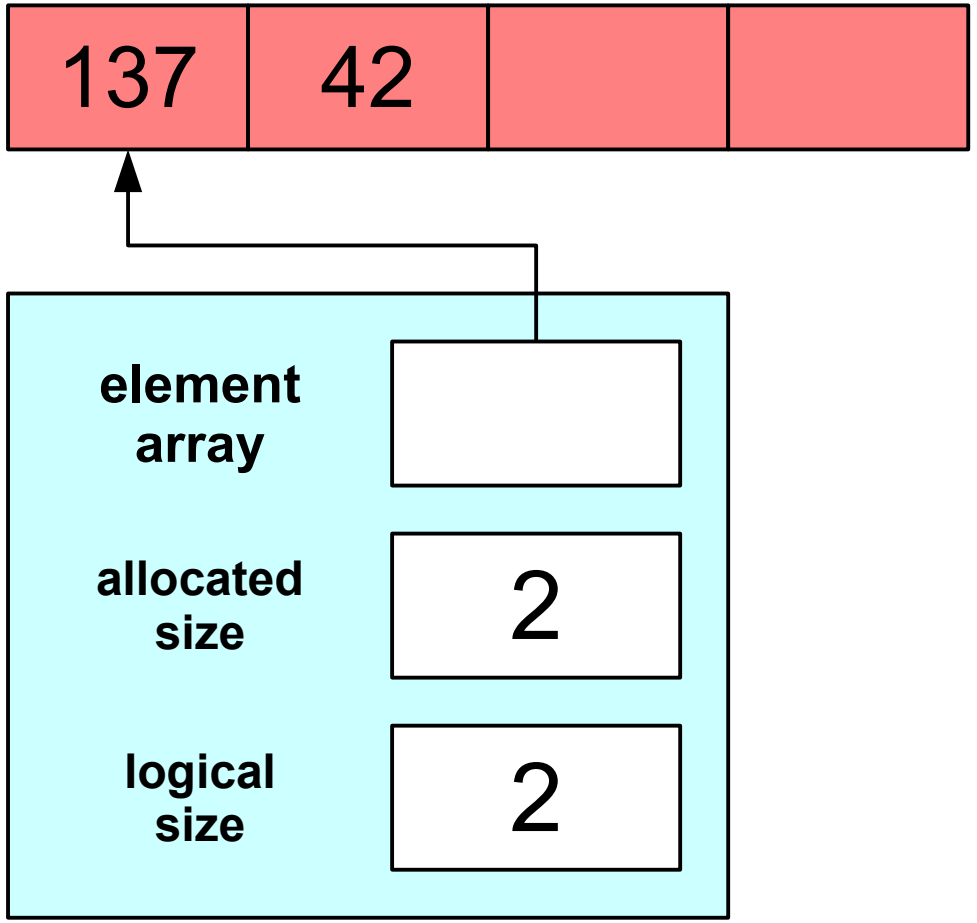
A Much Better Idea



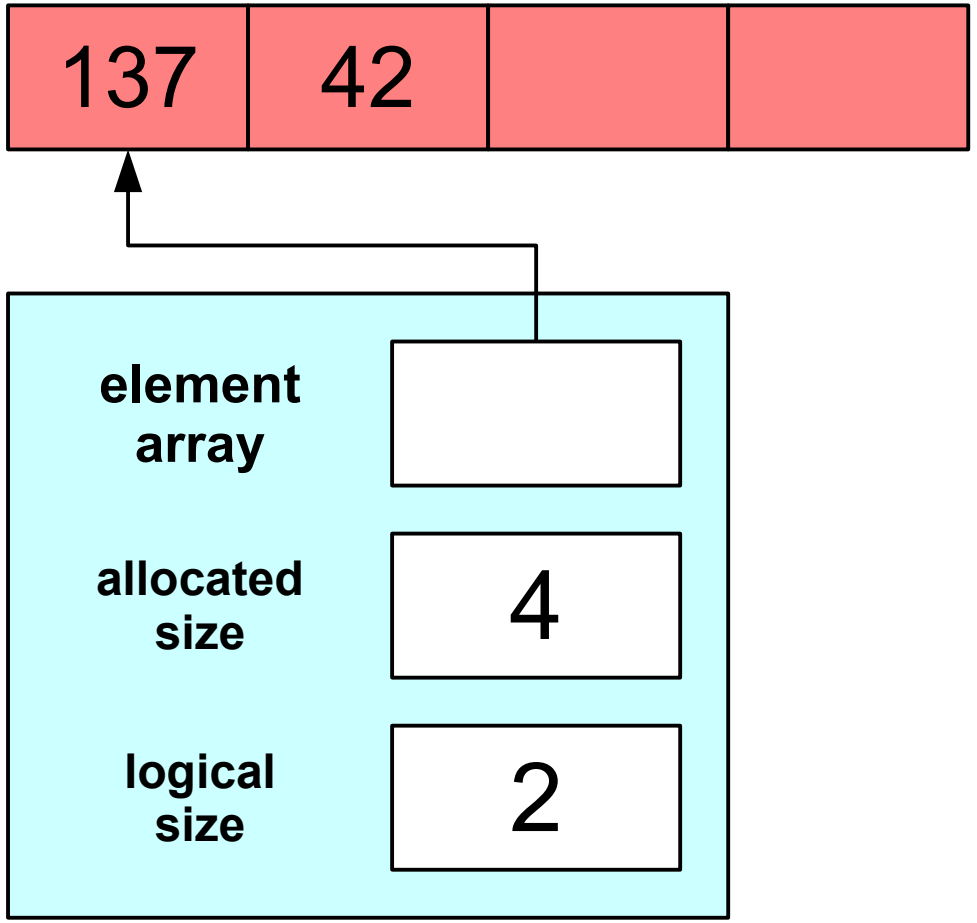
A Much Better Idea



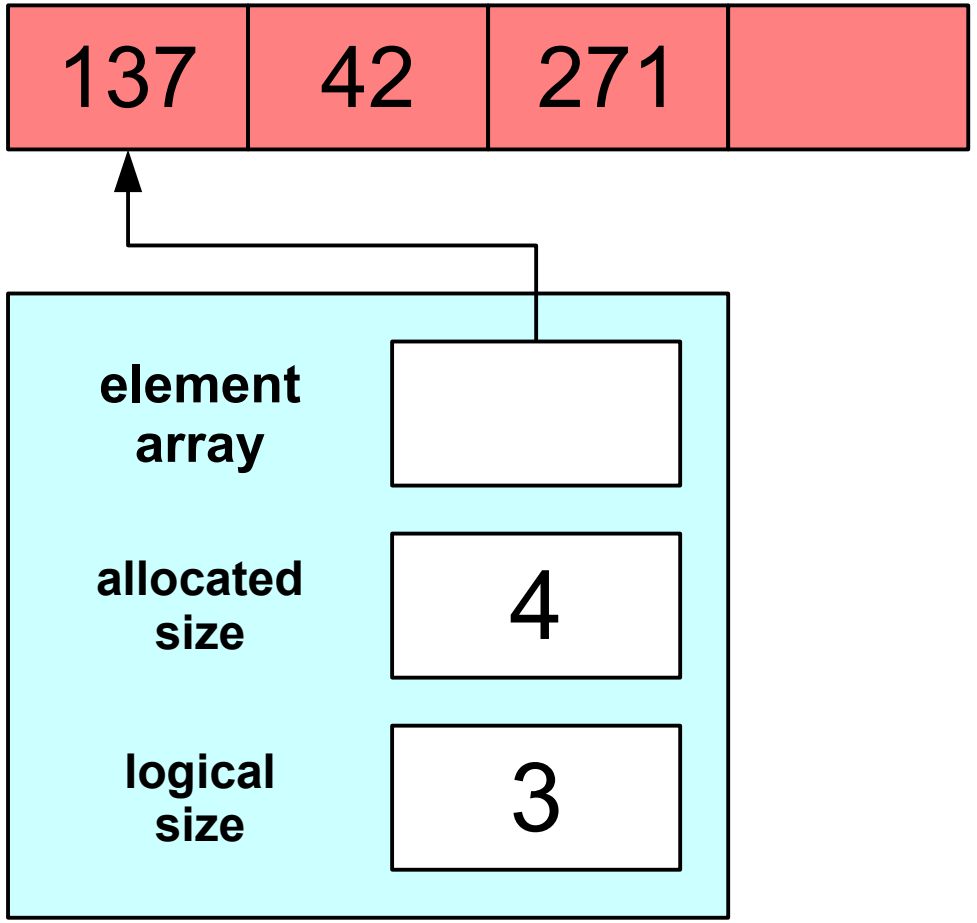
A Much Better Idea



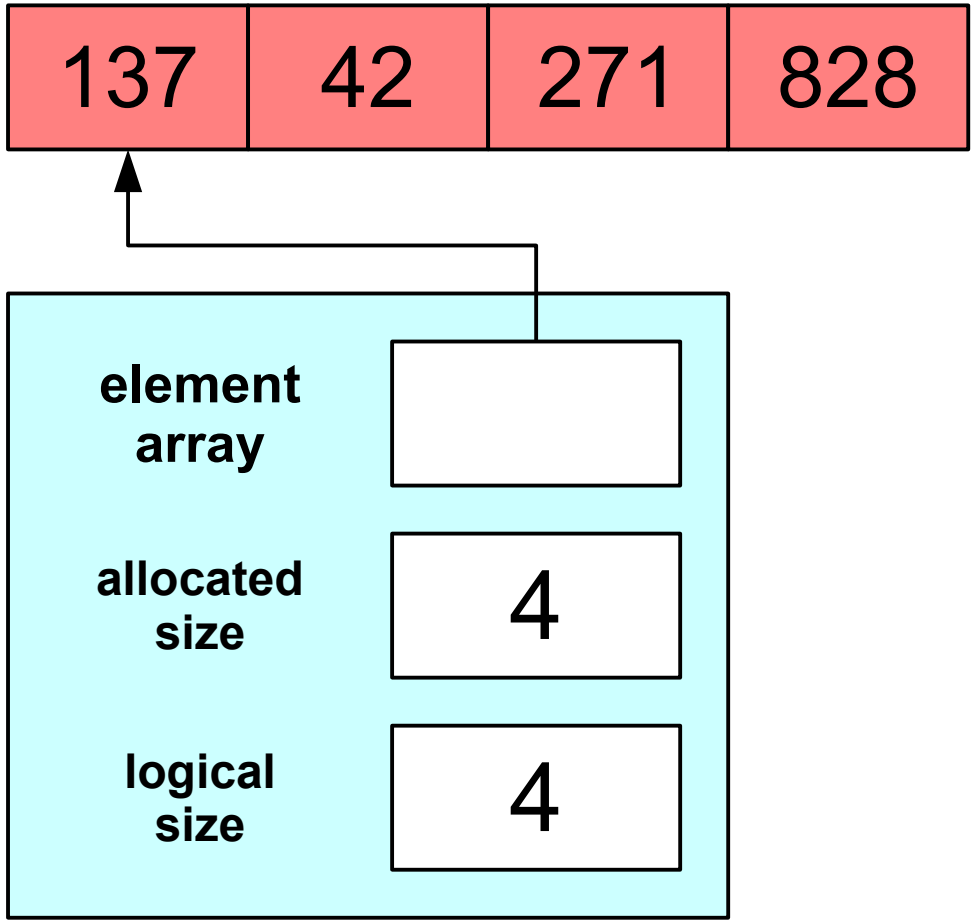
A Much Better Idea



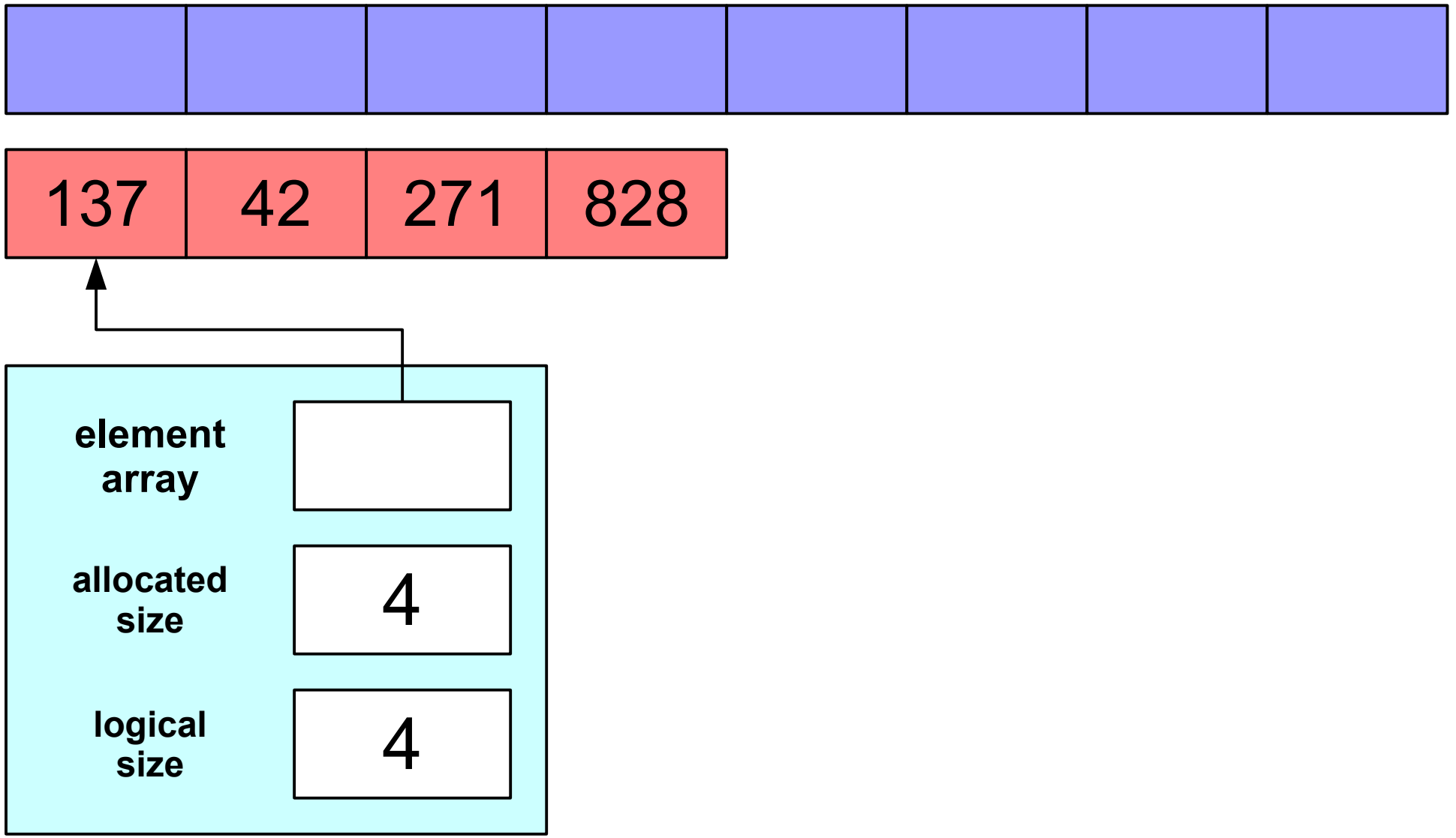
A Much Better Idea



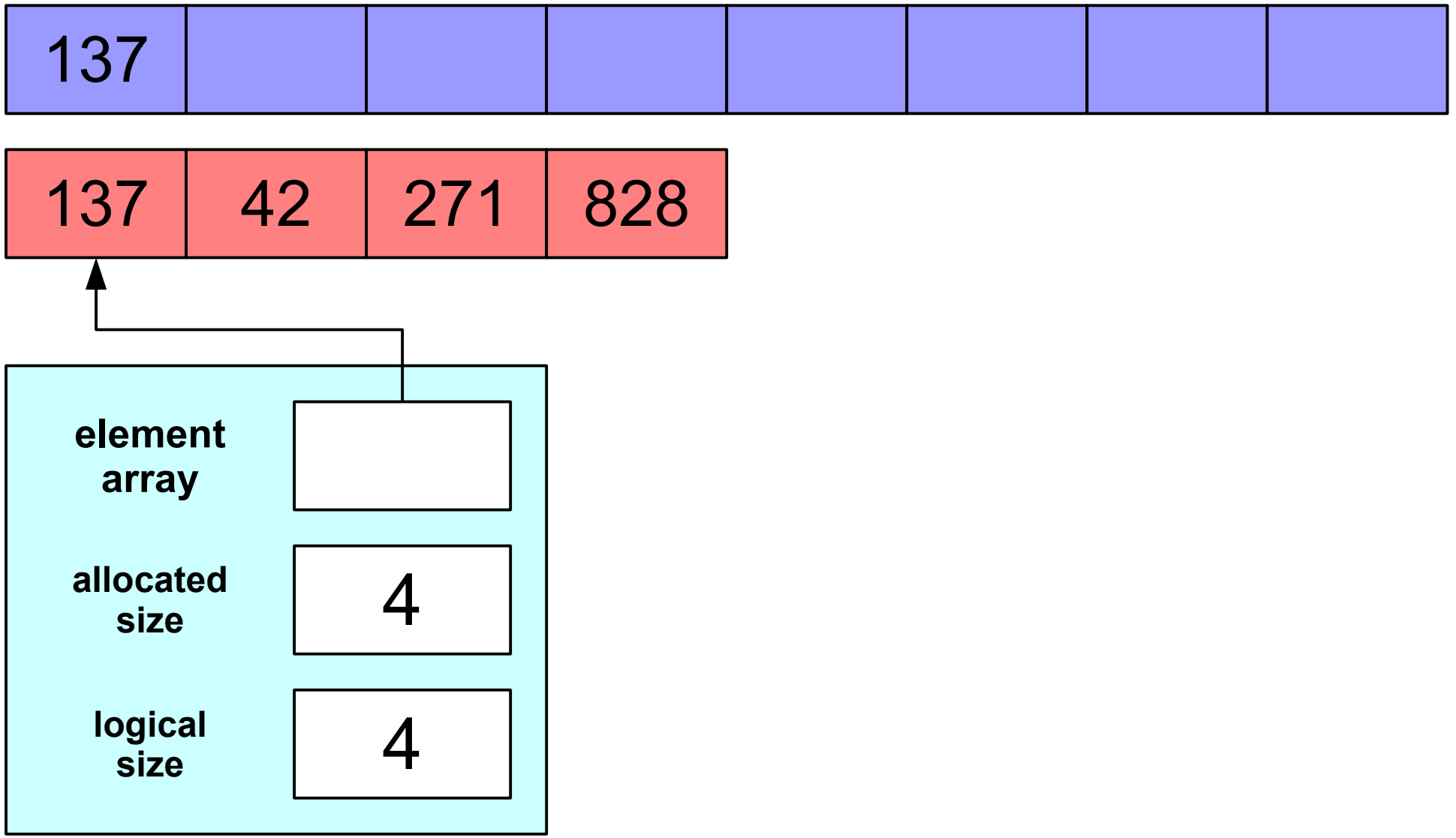
A Much Better Idea



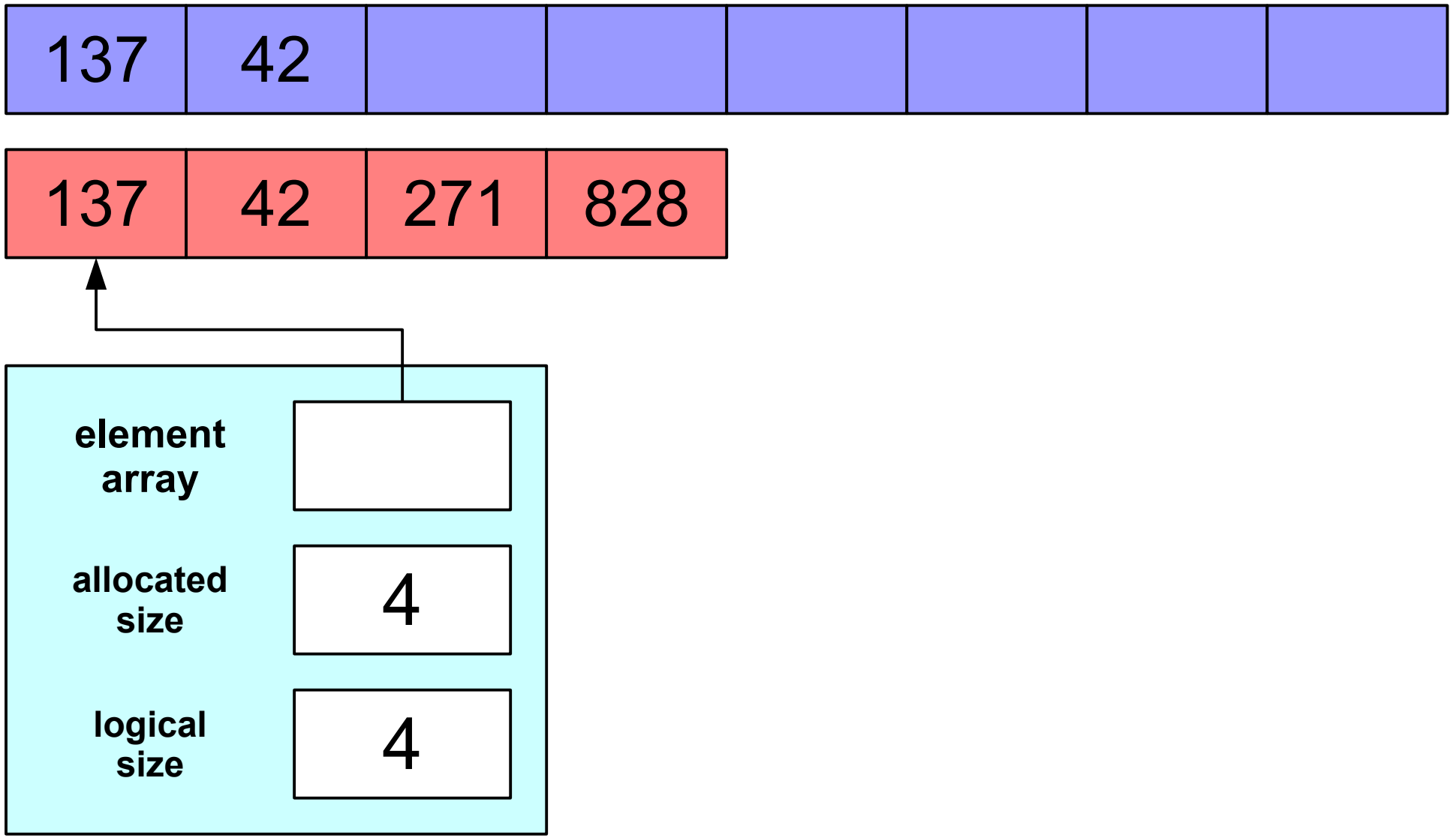
A Much Better Idea



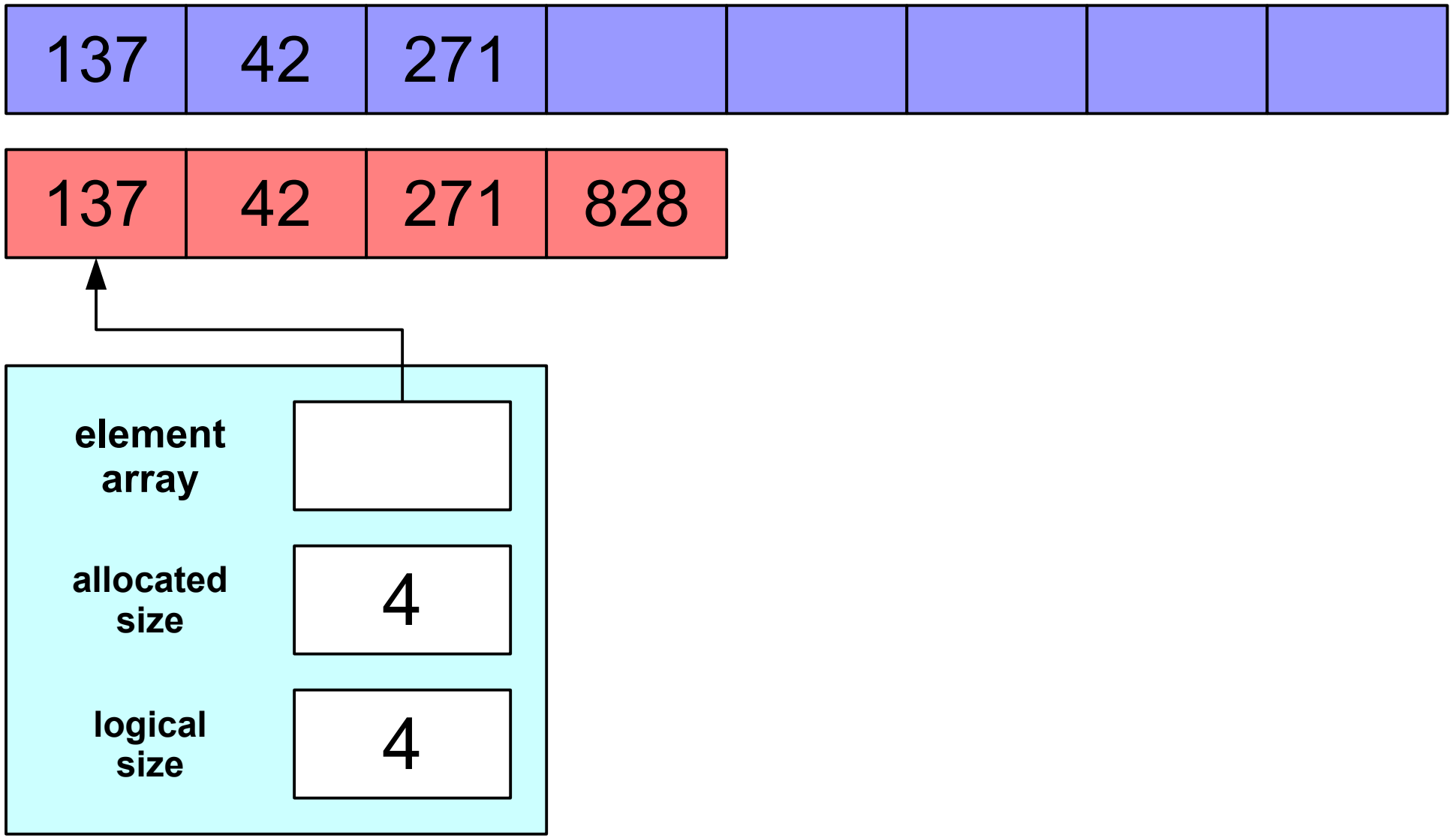
A Much Better Idea



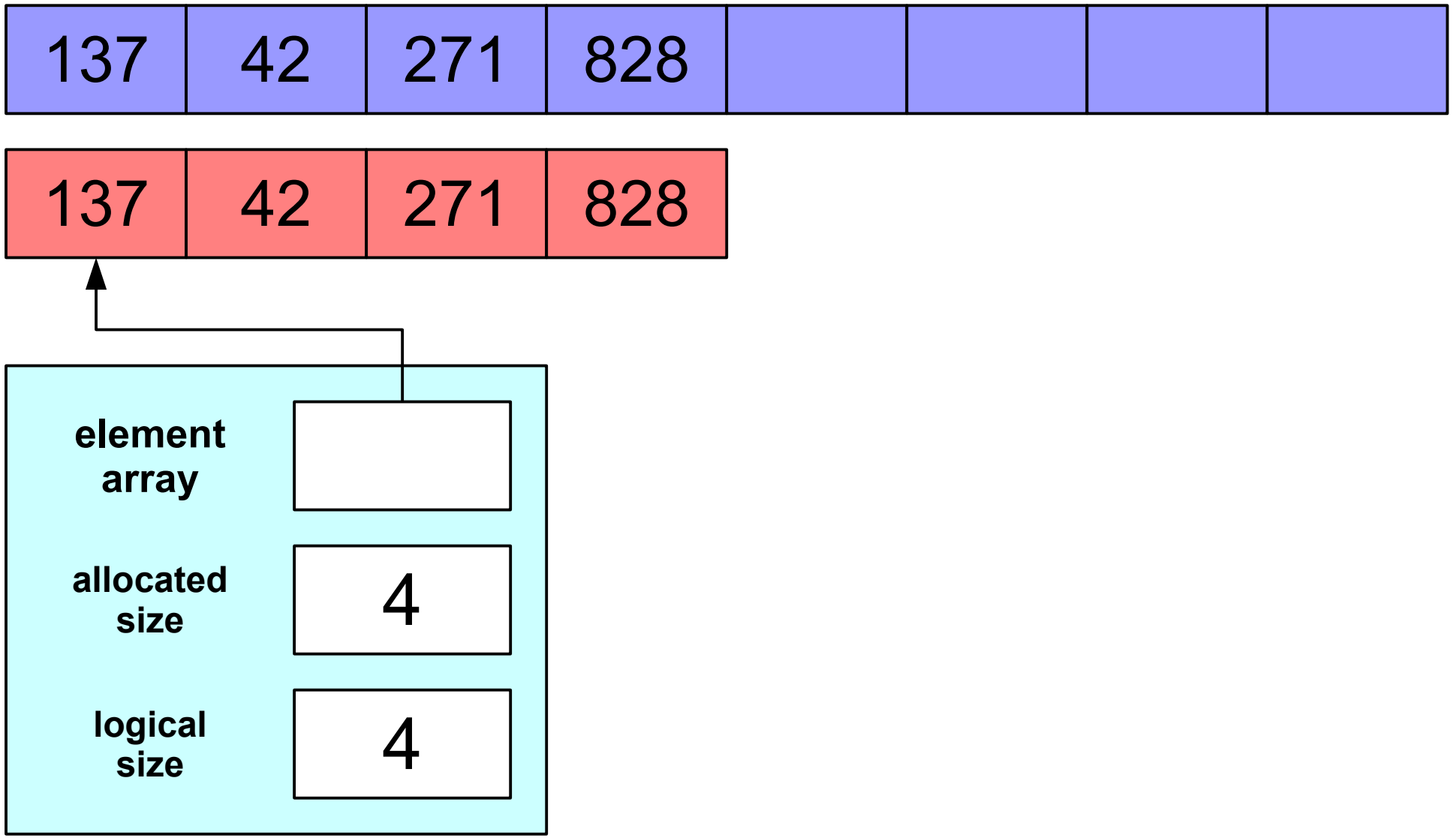
A Much Better Idea



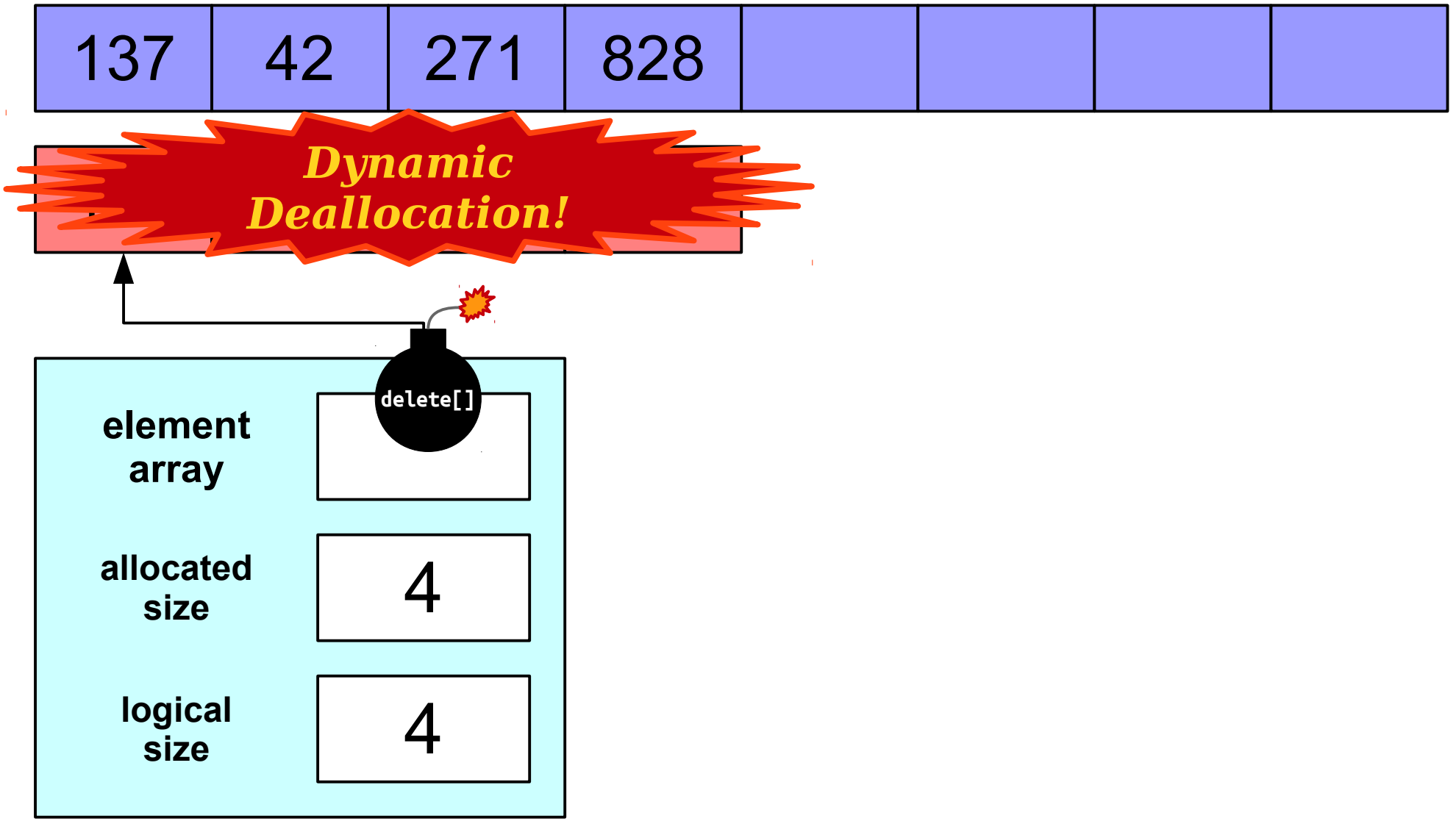
A Much Better Idea



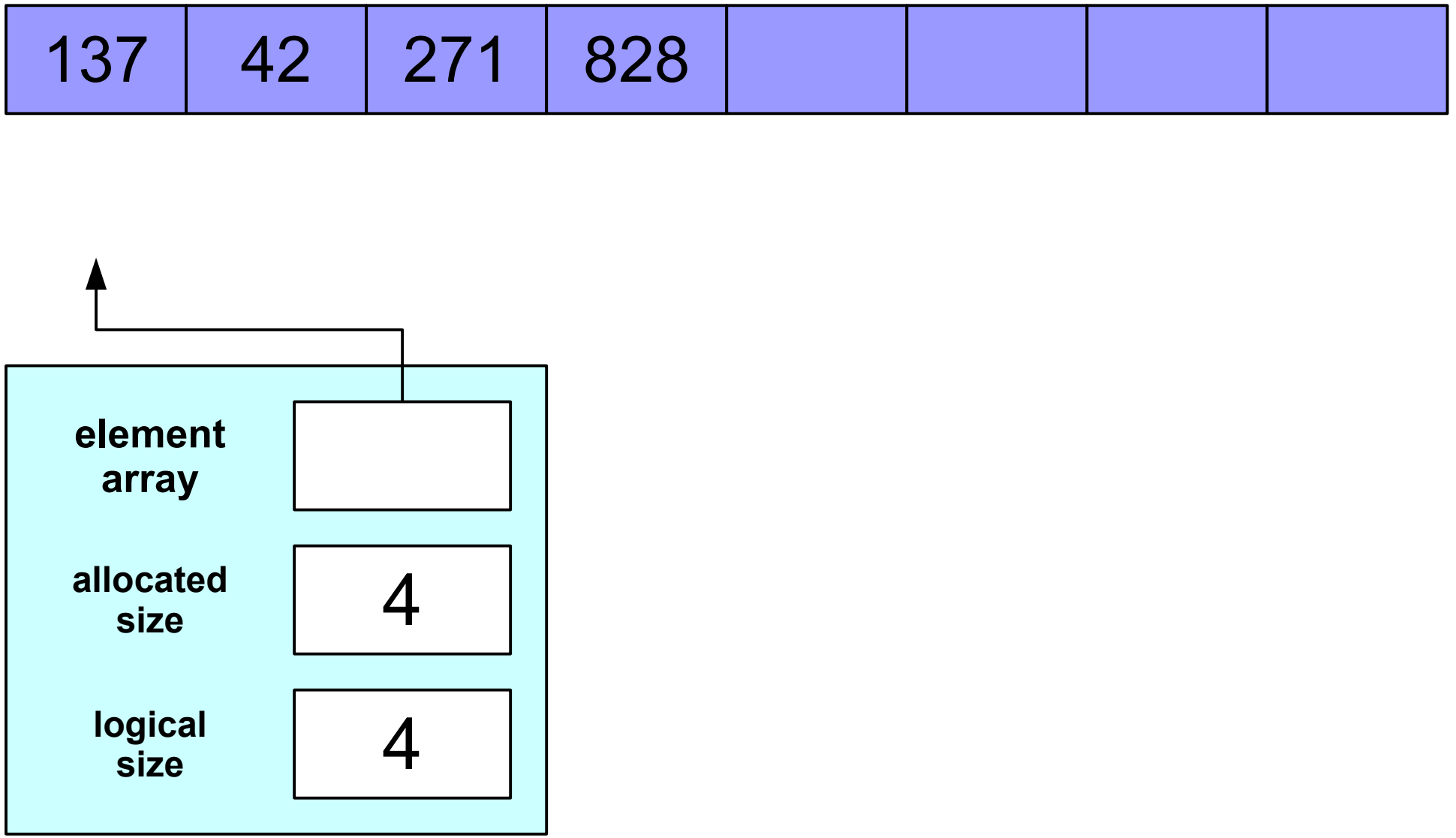
A Much Better Idea



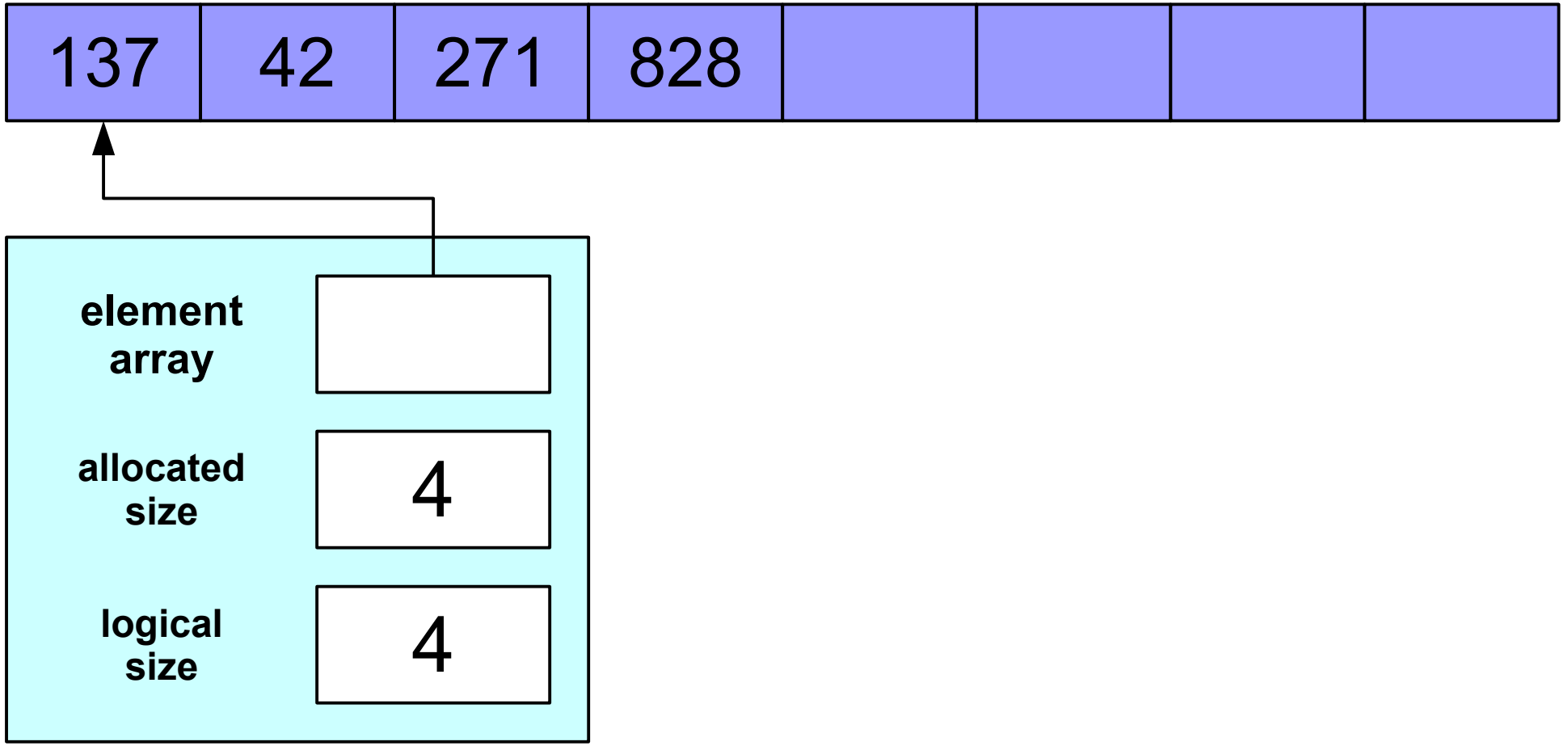
A Much Better Idea



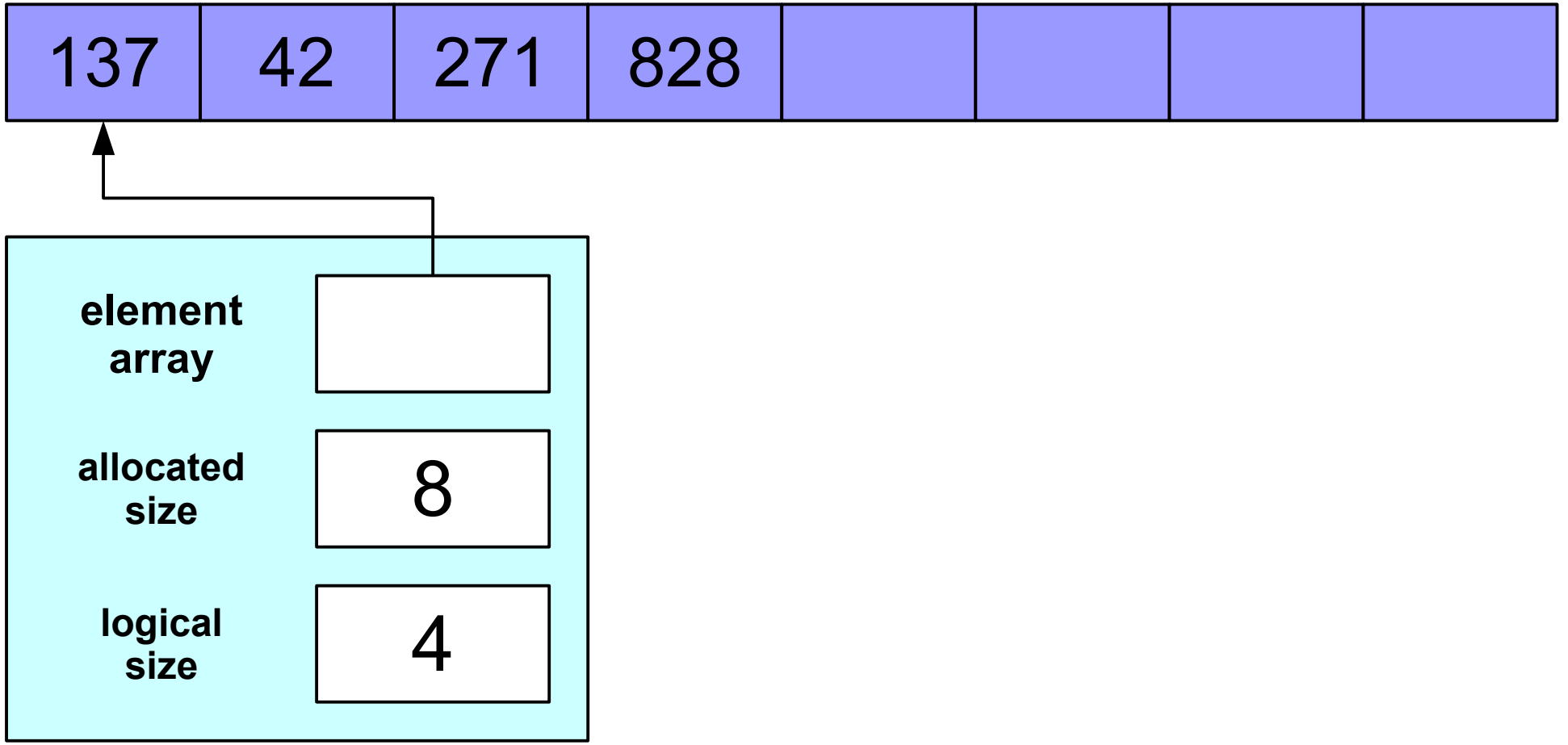
A Much Better Idea



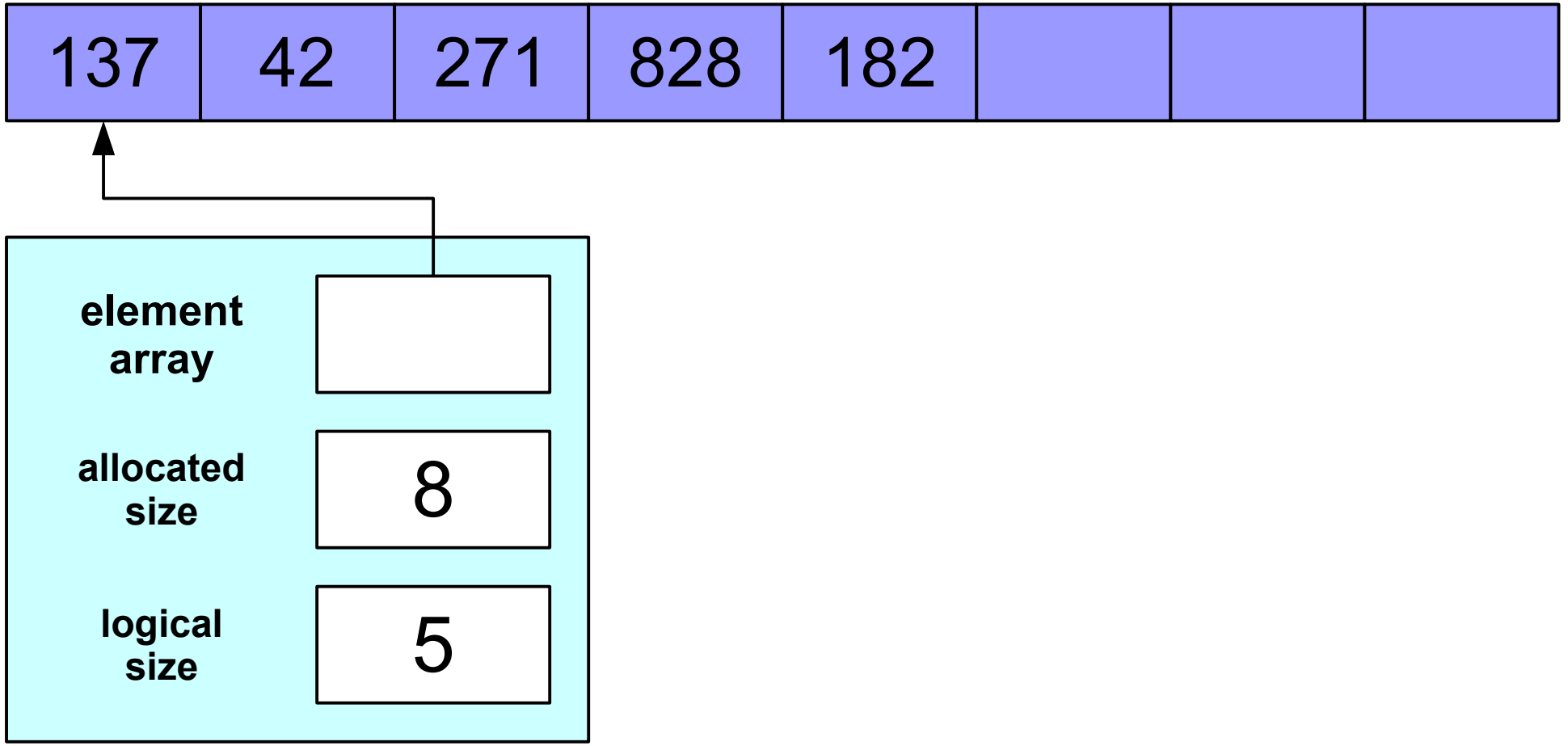
A Much Better Idea



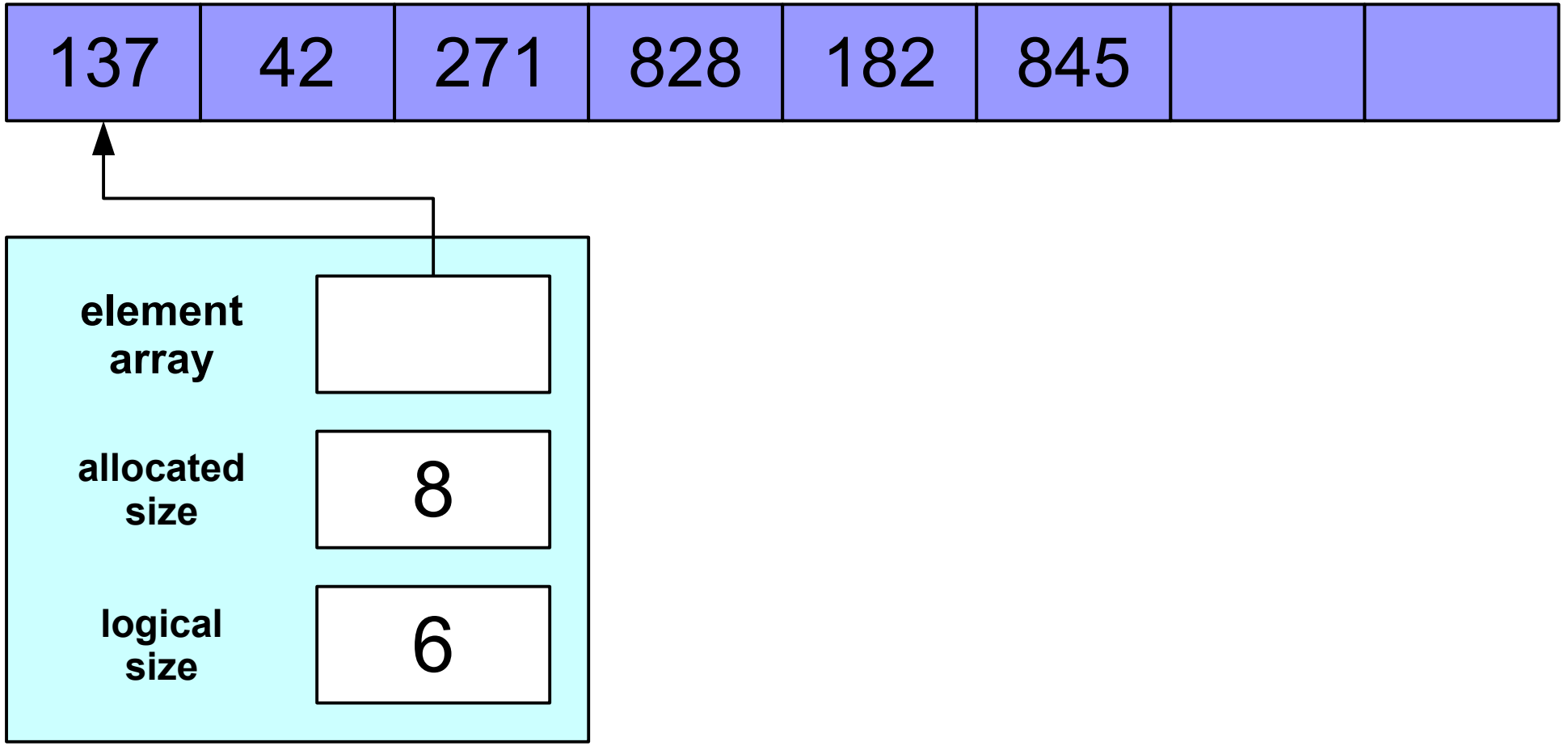
A Much Better Idea



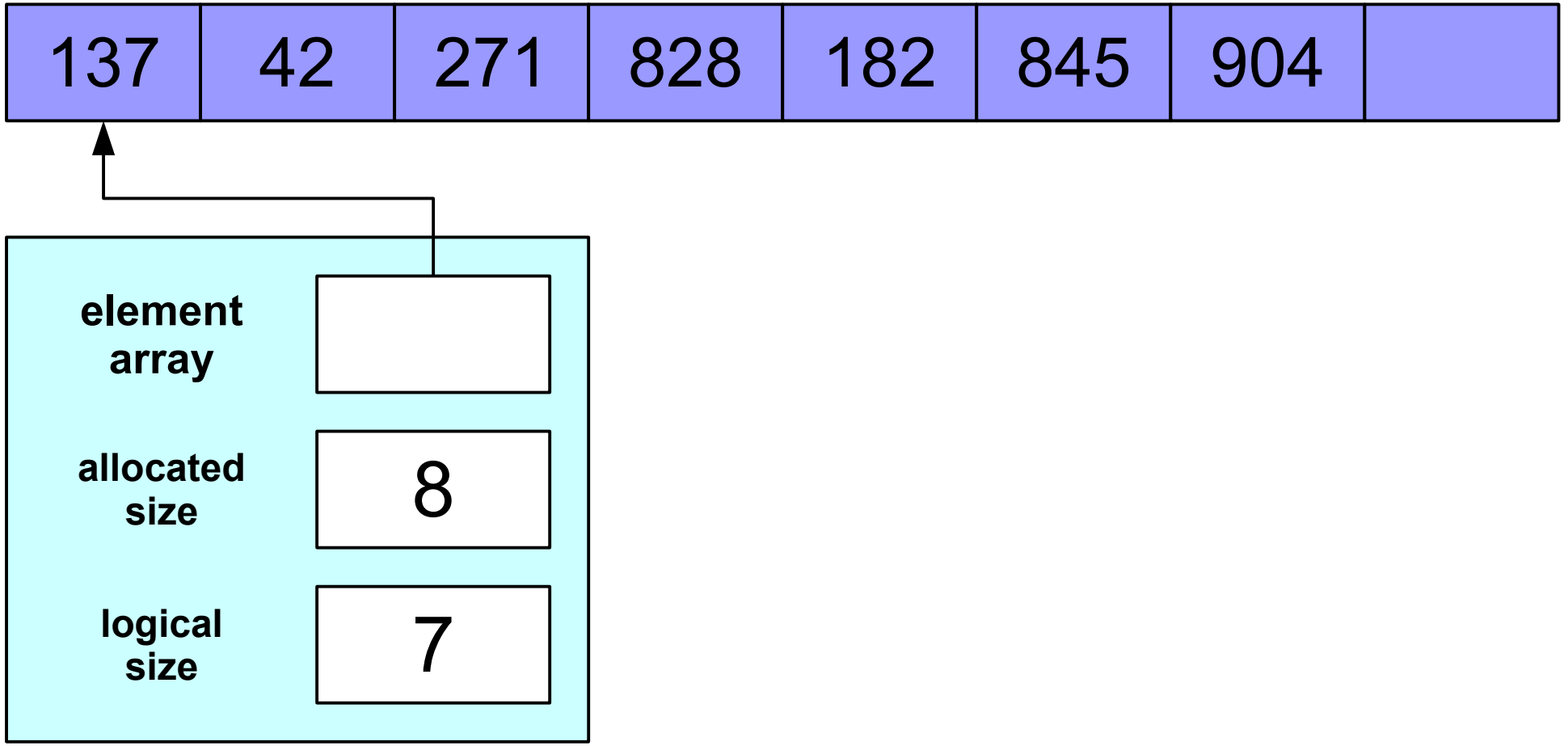
A Much Better Idea



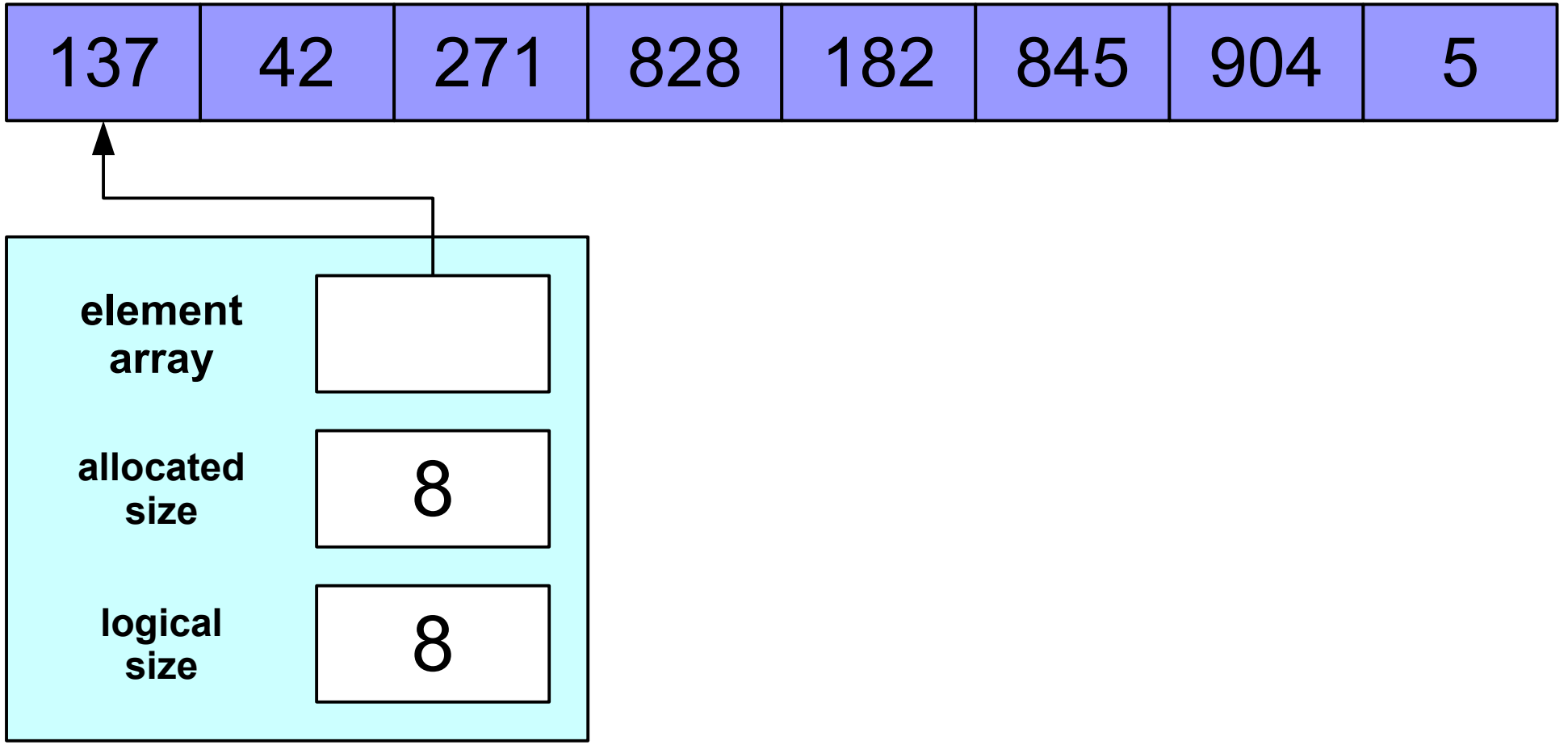
A Much Better Idea



A Much Better Idea



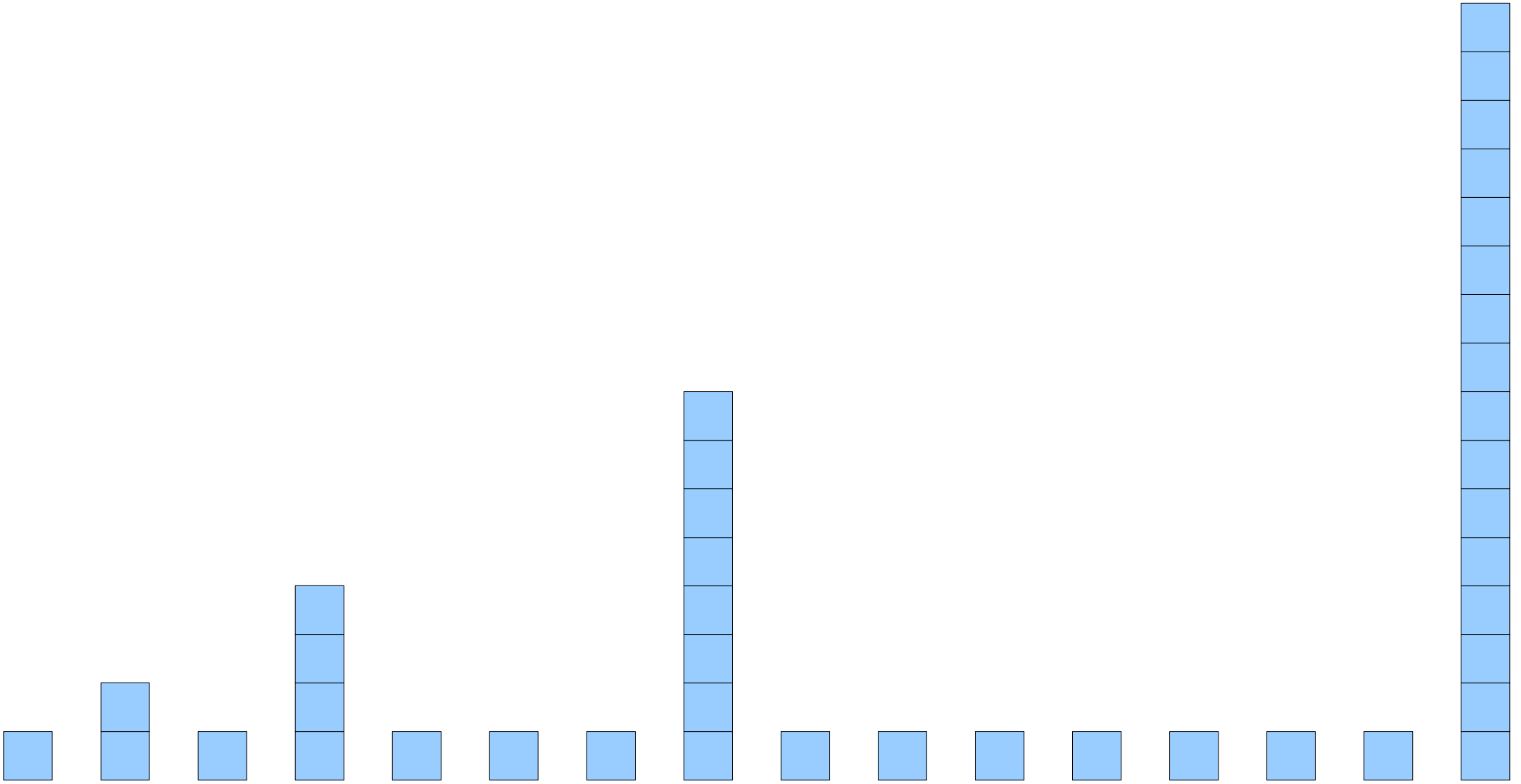
A Much Better Idea



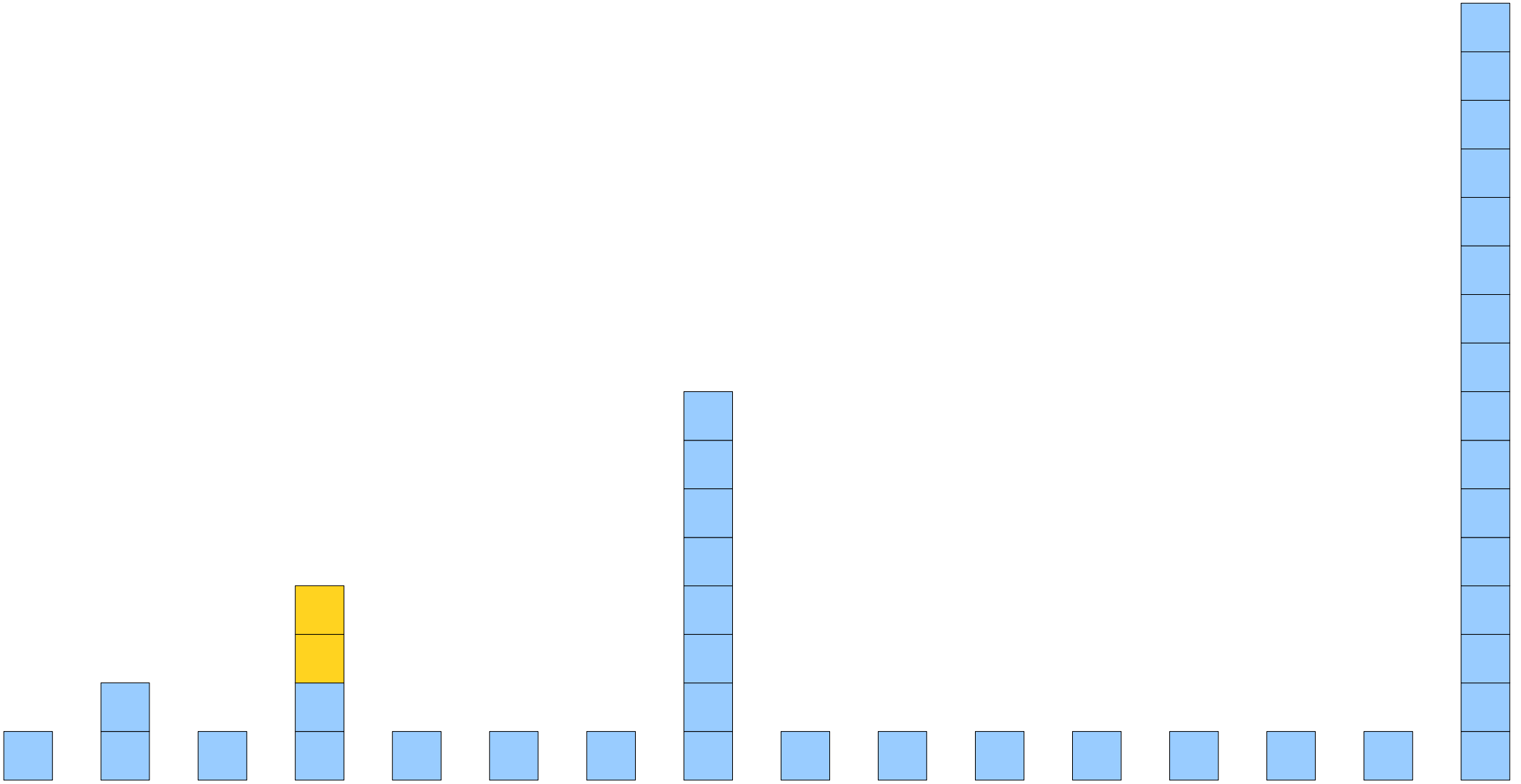
Let's Give it a Try!

How do we analyze this?

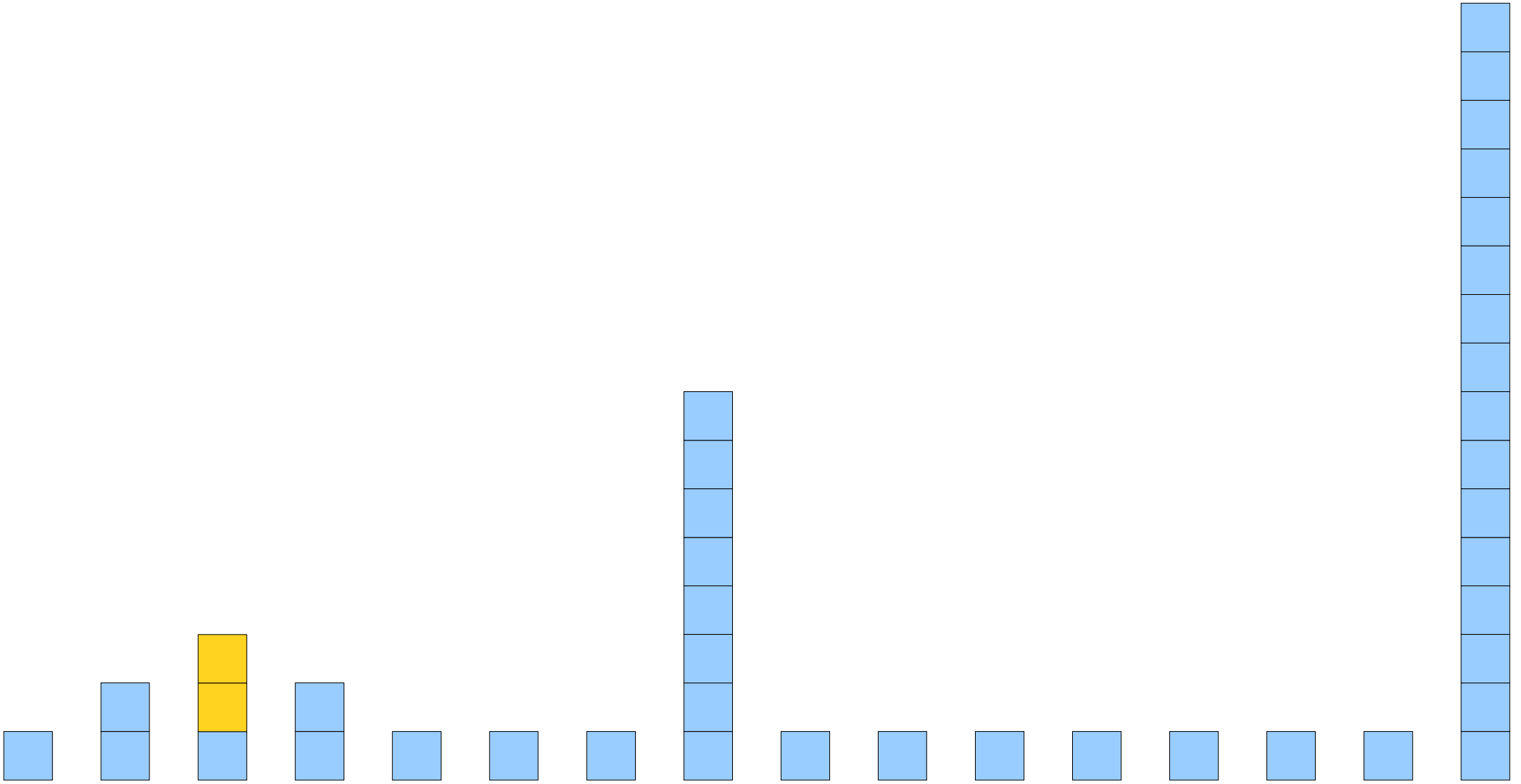
Spreading the Work



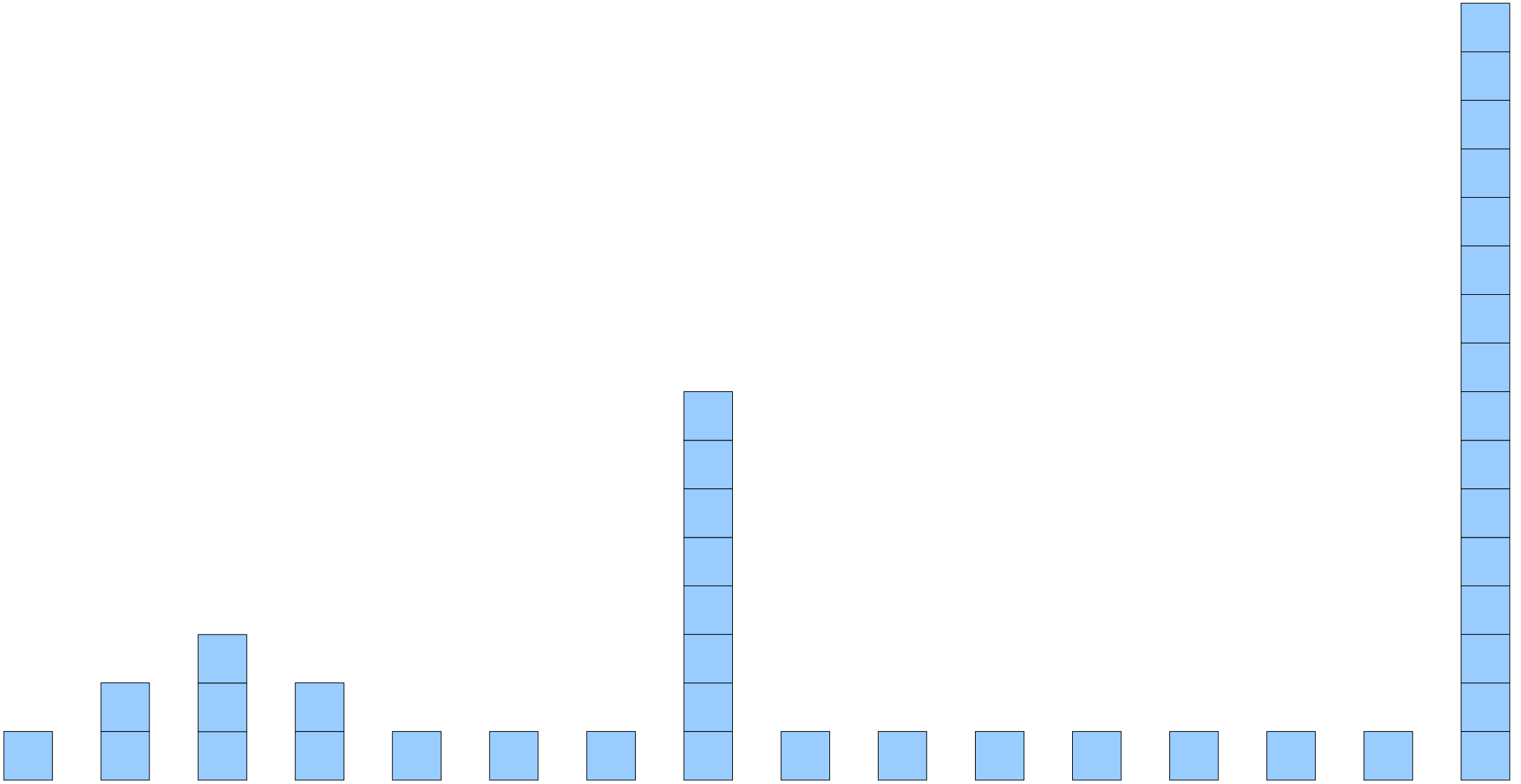
Spreading the Work



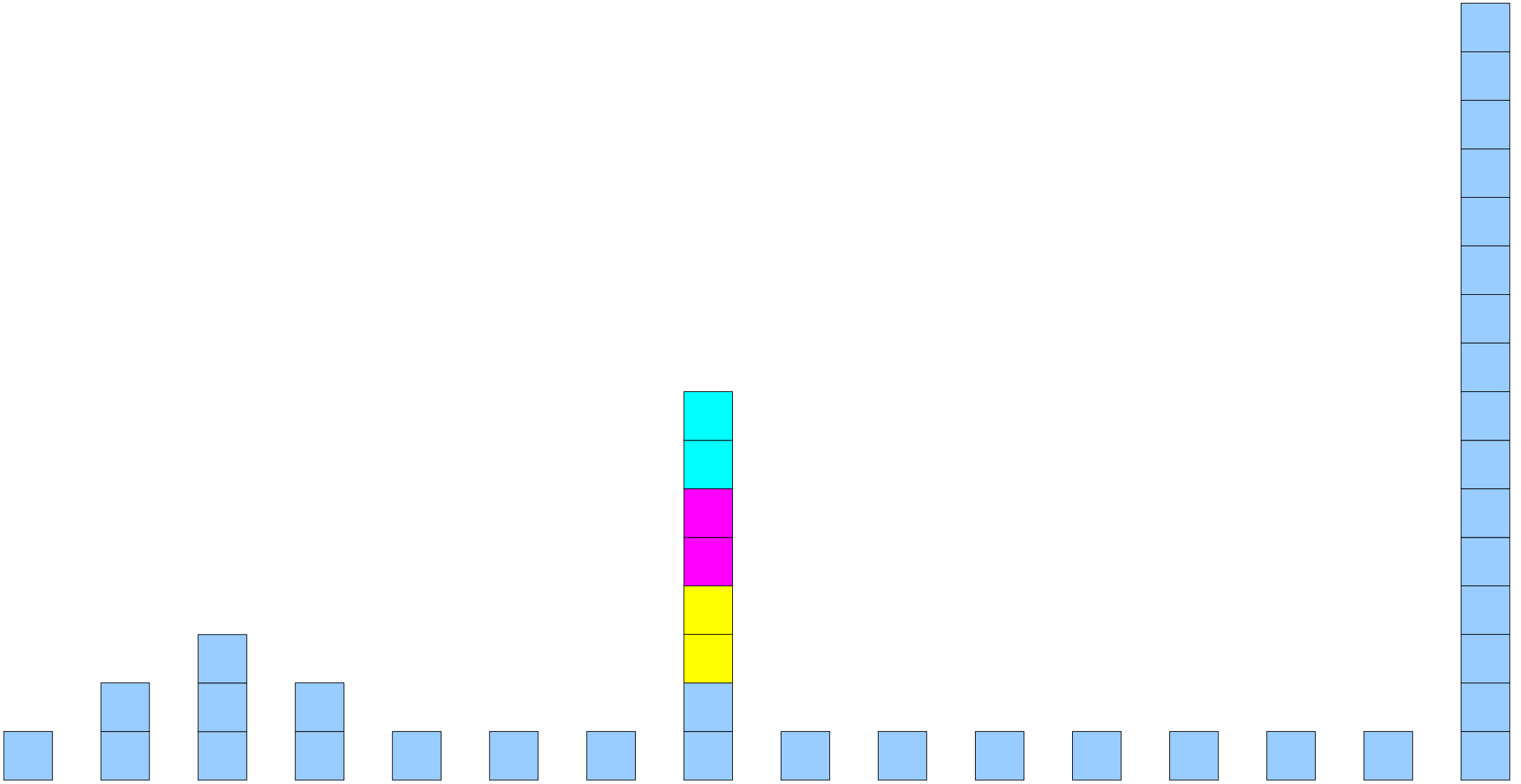
Spreading the Work



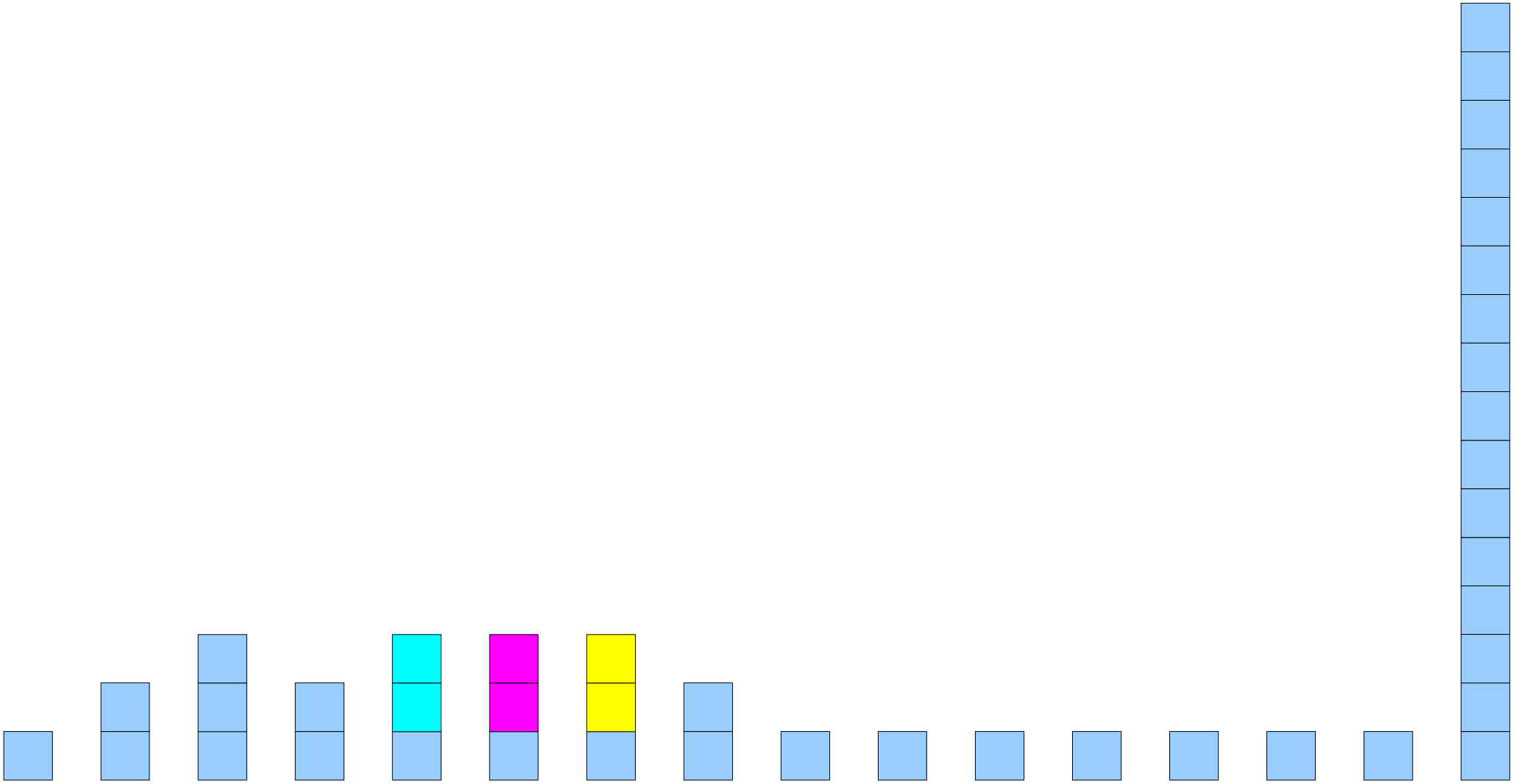
Spreading the Work



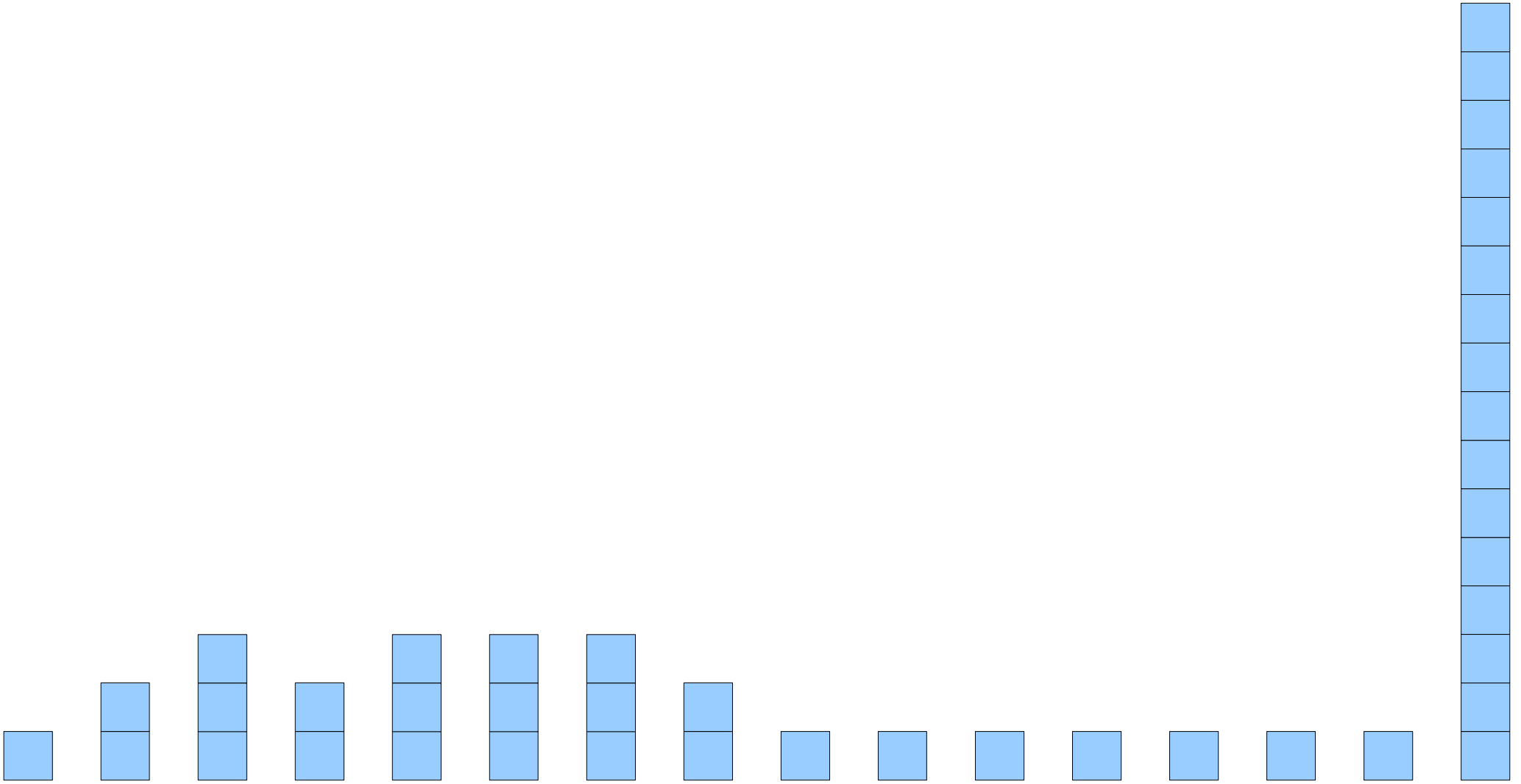
Spreading the Work



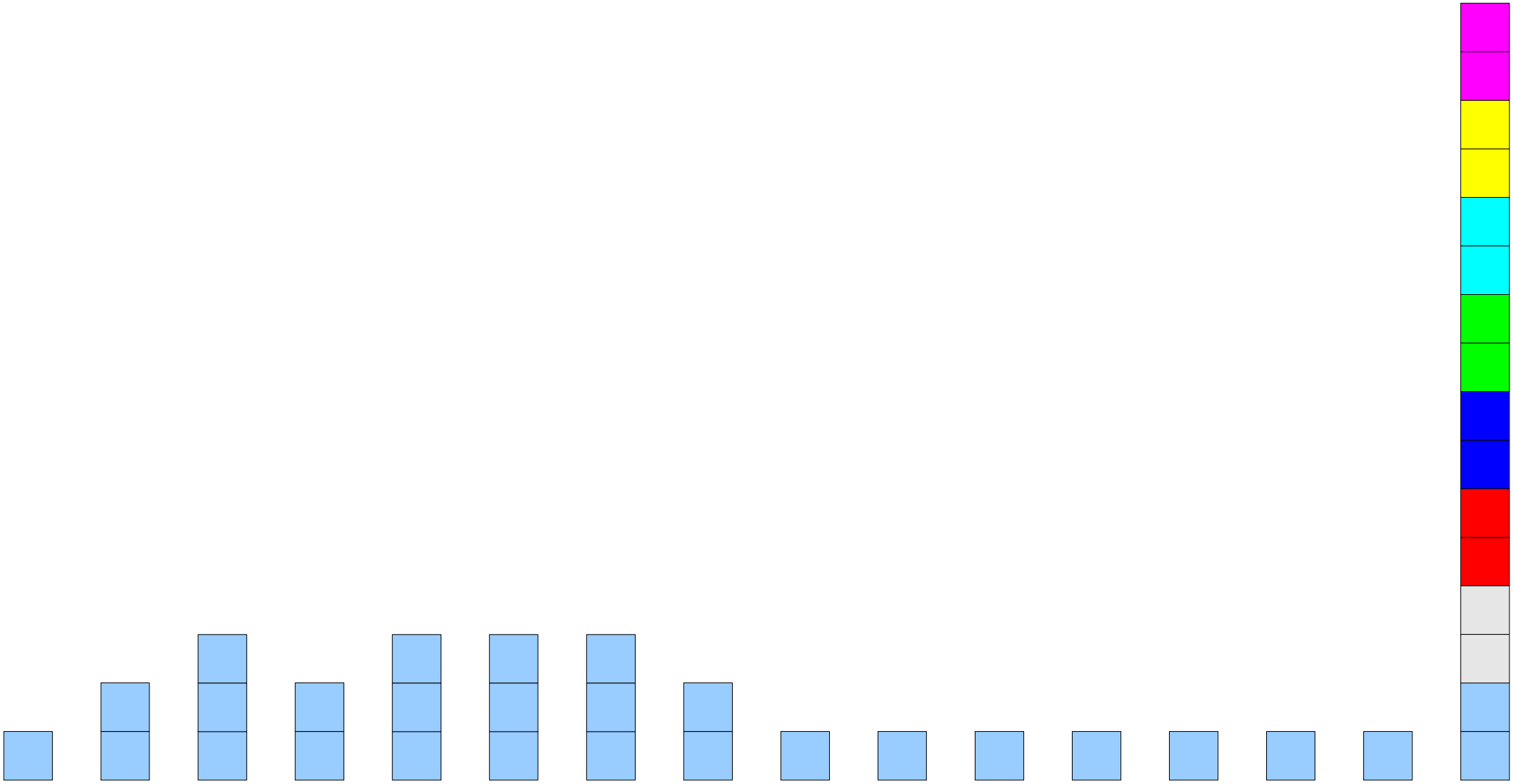
Spreading the Work



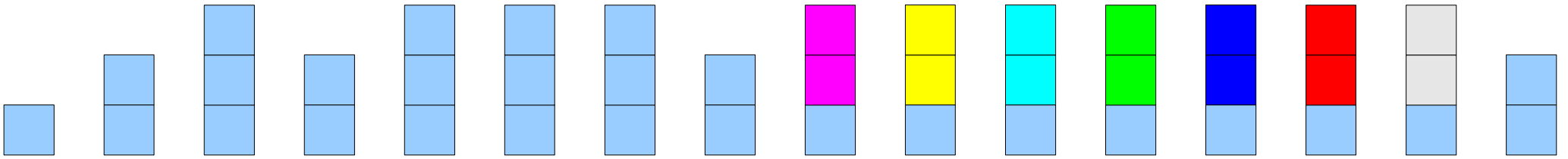
Spreading the Work



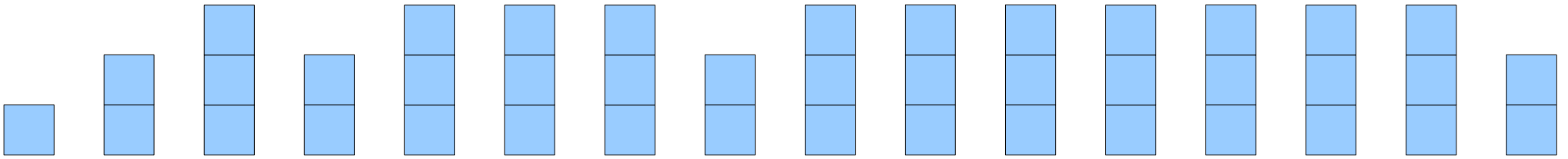
Spreading the Work



Spreading the Work



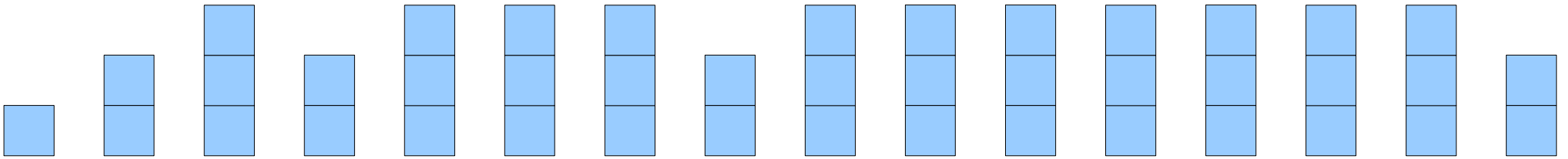
Spreading the Work



Spreading the Work

On average, we do just 3 units of work!

This is O(1) work on average!



Sharing the Burden

- We still have “heavy” pushes taking time O(n) and “light” pushes taking time O(1).
- Worst-case time for a push is O(n).
- Heavy pushes become so rare that the ***average*** time for a push is O(1).
- Can we confirm this?

Amortized Analysis

- The analysis we have just done is called an ***amortized analysis***.
- We reason about the total work done, not the word done per operation.
- In an amortized sense, our implementation of the stack is extremely fast!
- This is one of the most common approaches to implementing `Stack`.

Your Action Items

- ***Download BlueBook***
  - Hopefully you've already done this; if not, please do that soon.
- ***Finish Assignment 4***
  - Need help? Stop by the LaIR!

Next Time

- ***Linked Lists***
  - A different way to represent sequences of elements.
- ***Dynamic Allocation Revisited***
  - What else can we allocate?