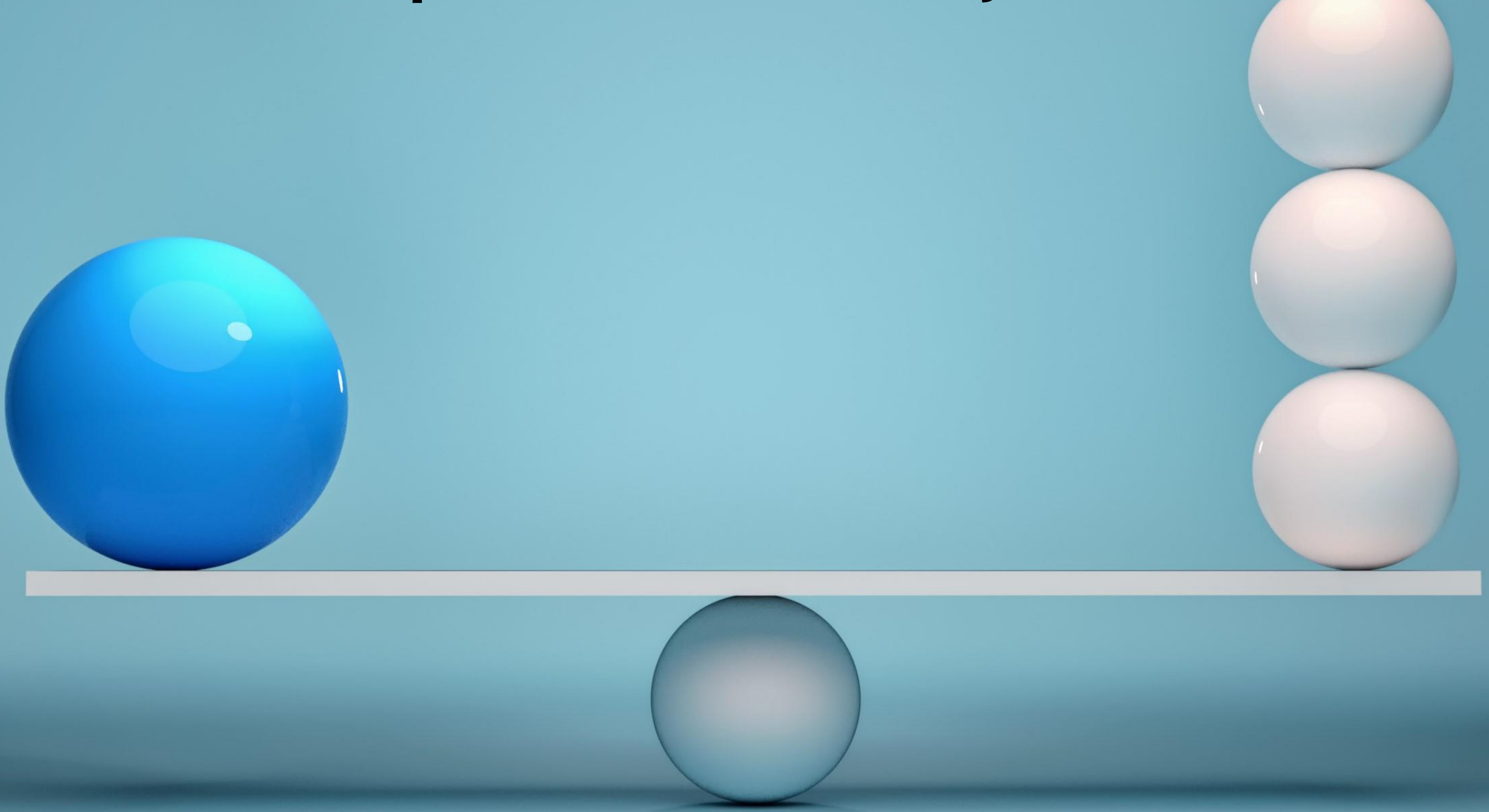
RootishArrayStack

A Space-Efficient Array Stack

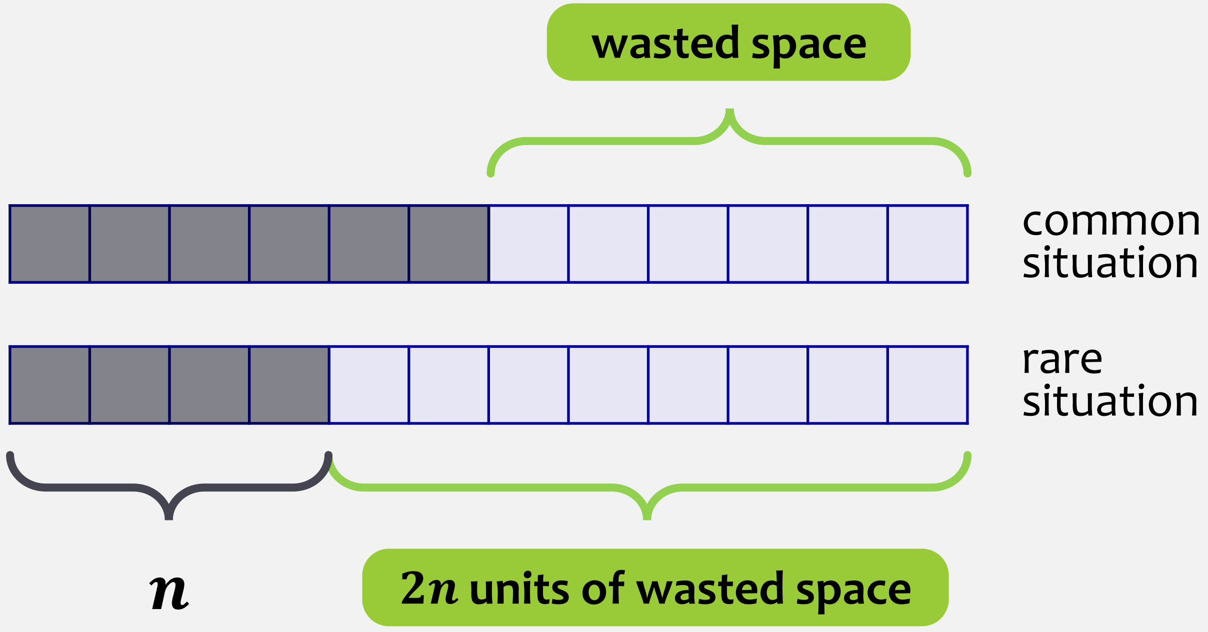


Alina Shaikhet

Wasted Space (memory)

ArrayStack

- immediately after a **resize()** operation:
- **2/3** of the backing array can be empty:



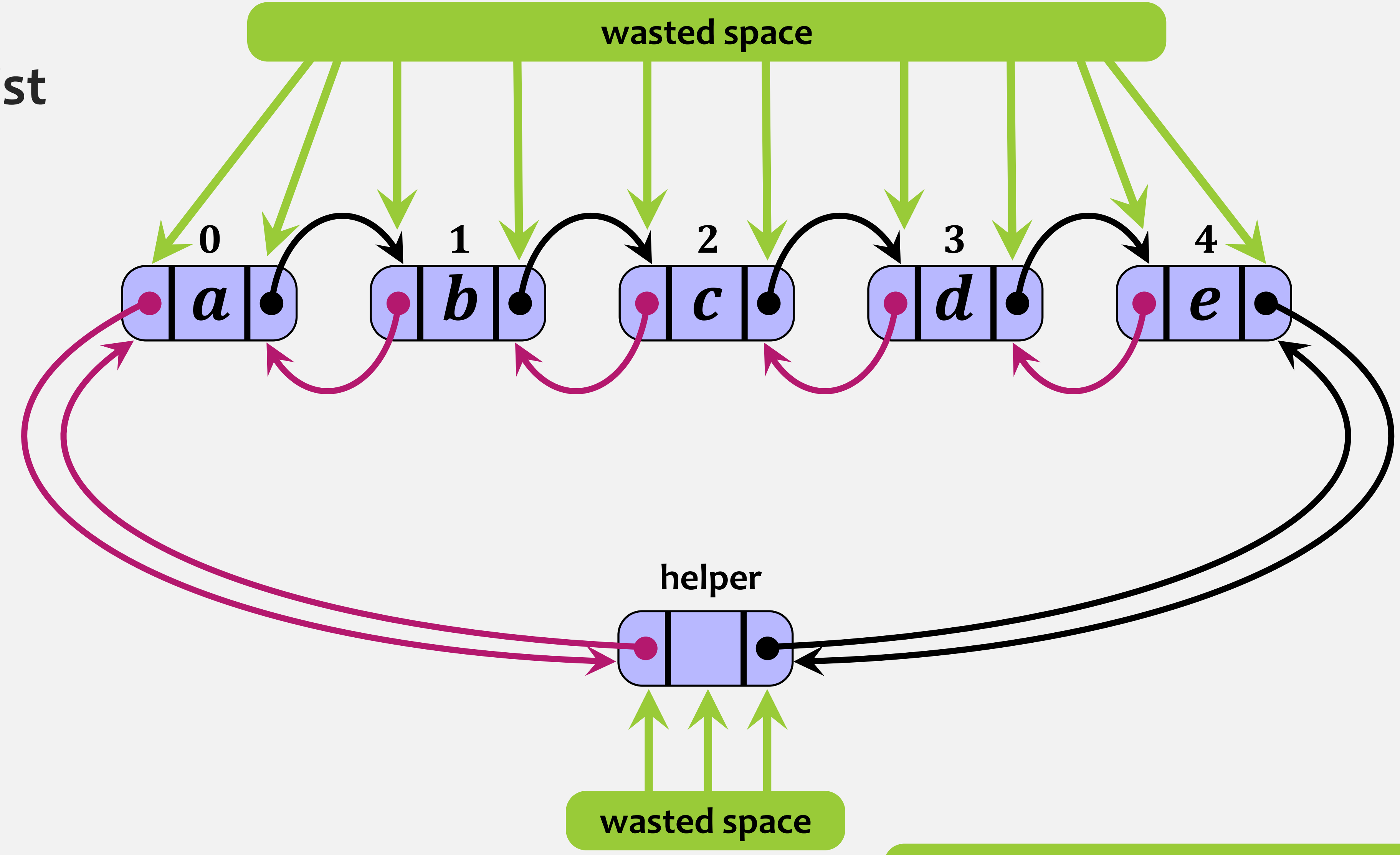
Wasted space (memory) – any memory location not being used to store the only copy of some data item.

one unit of wasted space → T[] a - pointer to the array

one unit of wasted space → int n - list size

Wasted Space (memory)

LInkedList



In total, at least $2n$ units of wasted space

RootishArrayStack

All List implementations, we saw so far, **often** waste $\Omega(n)$ units of space.

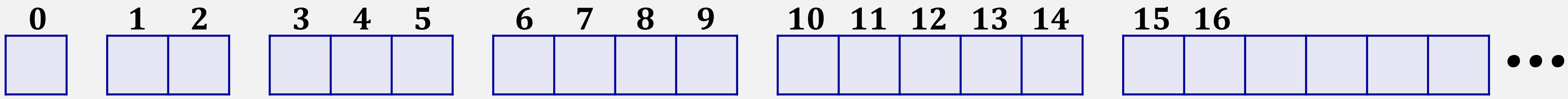
RootishArrayStack is a **List** implementation that wastes only $O(\sqrt{n})$ space.

n	$2n$	$\sqrt{n}$
10000	20000	100
1000000	2000000	1000
100000000	200000000	10000

RootishArrayStack

A **RootishArrayStack** stores its elements in a **List** of $\boldsymbol{r}$ arrays called **blocks.**

$\boldsymbol{r}$ arrays are numbered $\mathbf{0, 1, \ldots, r-1}$.



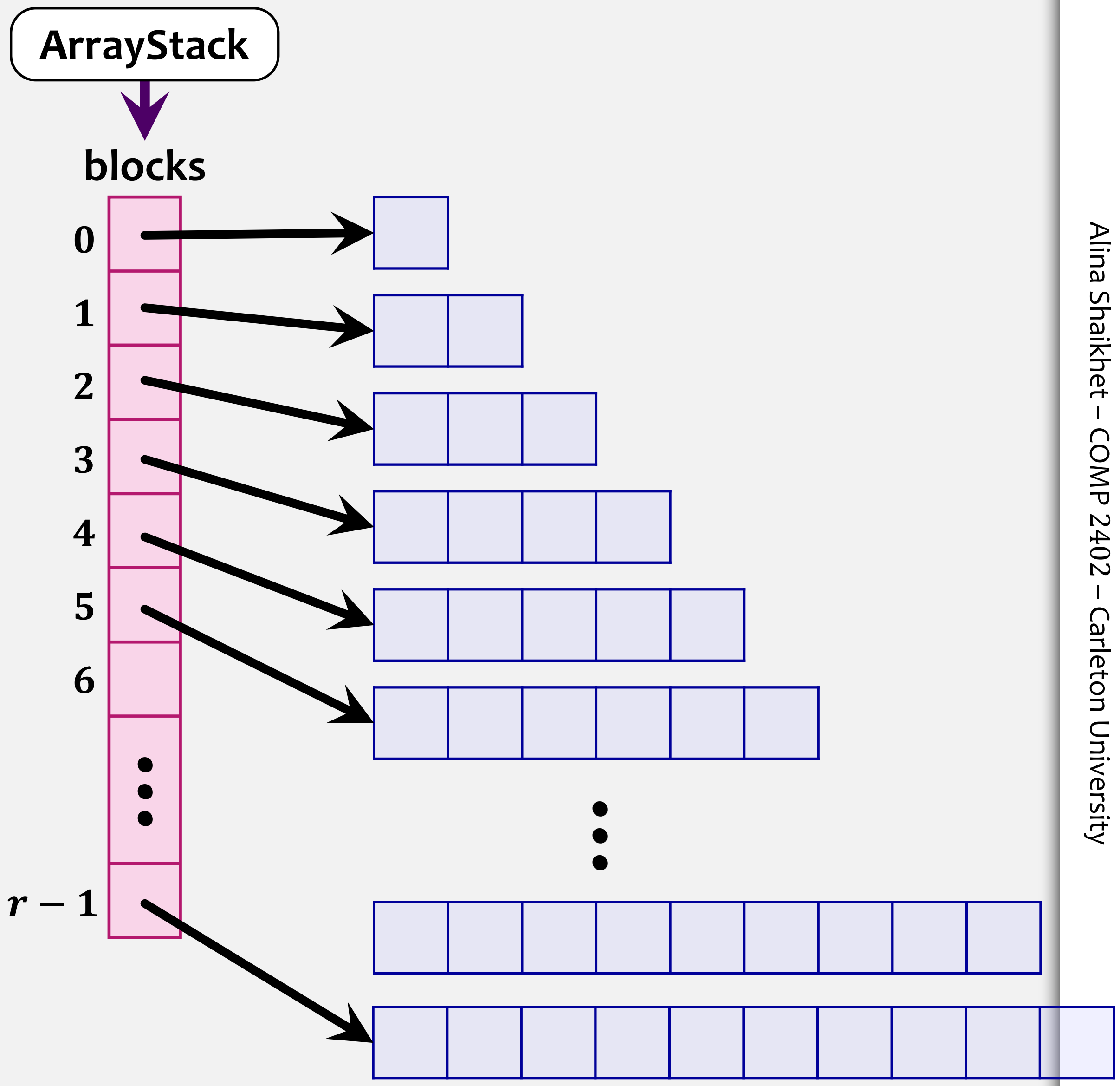
RootishArrayStack

A **RootishArrayStack** stores its elements in a **List** of r arrays called **blocks.**

r arrays are numbered $\mathbf{0, 1, \ldots, r-1}$.

Rule: there is at least one item in the last **two** blocks

All the blocks before the last two are completely full.

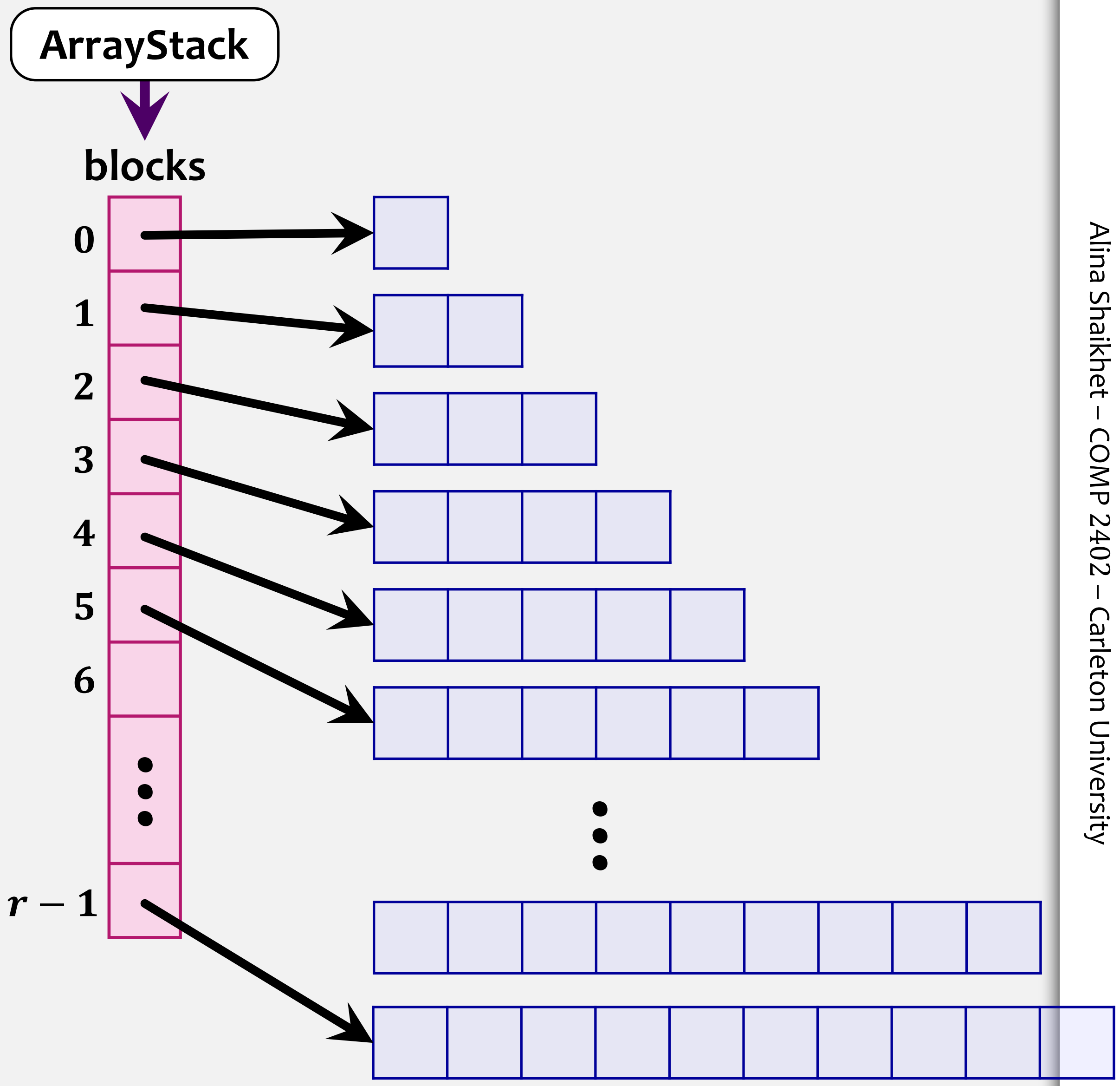


How many list items can we store in DS with r blocks?

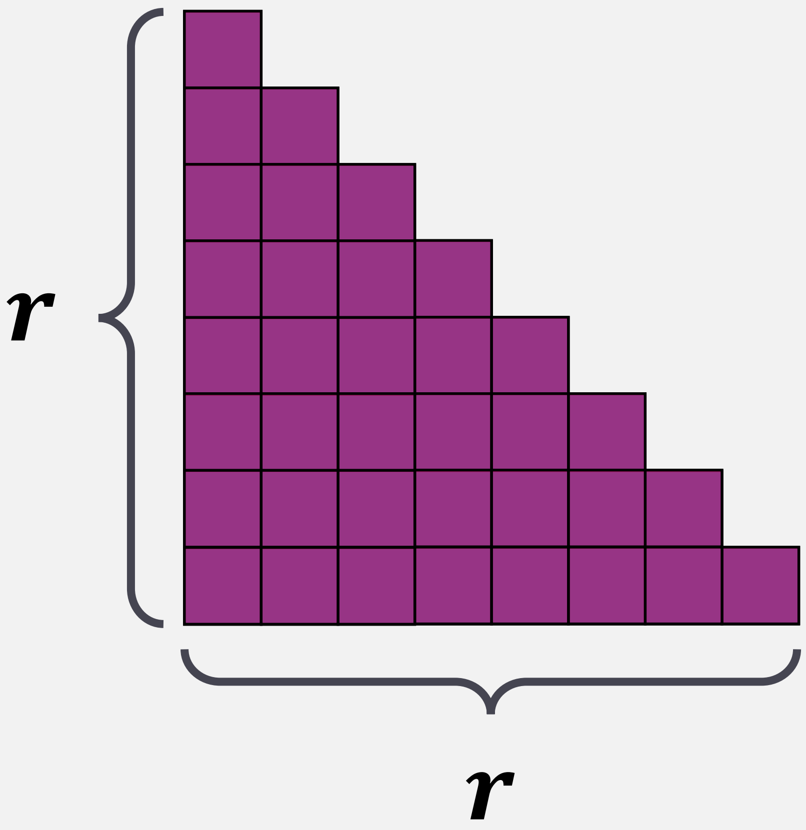
RootishArrayStack

How many list items can we store in this DS with r blocks?

$$1 + 2 + 3 + \cdots + r = \frac{r(r+1)}{2}$$

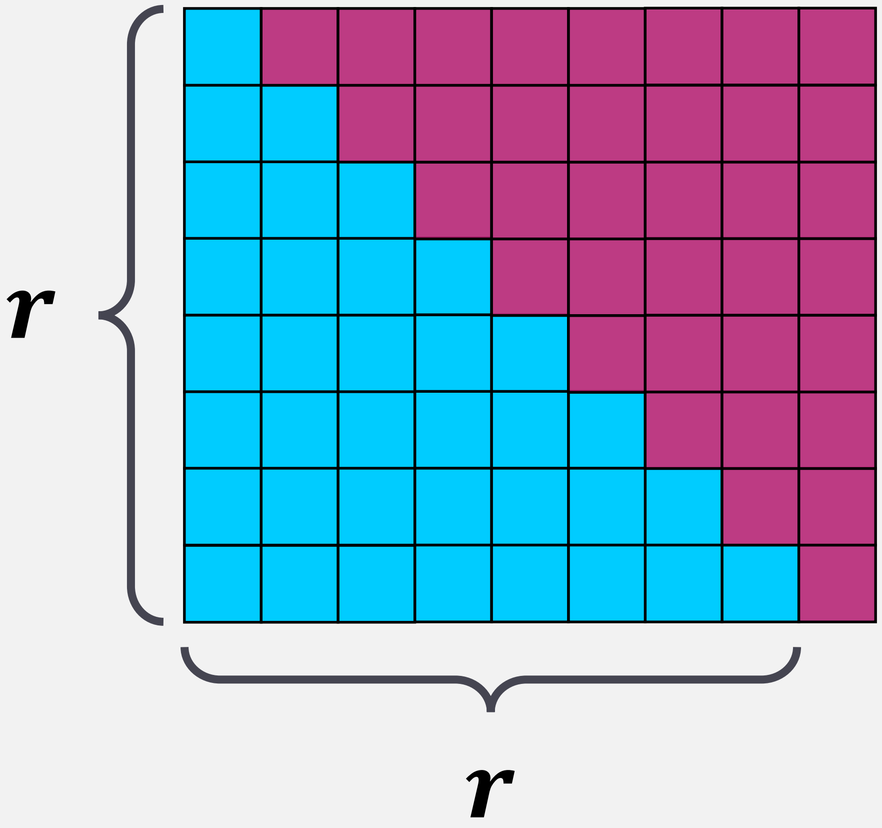


Sum of the first $\boldsymbol{r}$ positive integers



Sum of the first $\boldsymbol{r}$ positive integers

$$\frac{\boldsymbol{r(r+1)}}{\boldsymbol{2}}$$



RootishArrayStack

Assume we have $\boldsymbol{n}$ elements to store.

How many blocks do we need?

Given $\boldsymbol{r}$ blocks we can store $\leq \dfrac{\boldsymbol{r(r+1)}}{\boldsymbol{2}} = \dfrac{\boldsymbol{r^2+r}}{\boldsymbol{2}}$ elements.

Solve for $\boldsymbol{r}$: $\boldsymbol{r^2 + r = 2n}$

RootishArrayStack

$i=8$

What is the index of $\boldsymbol{i}$ within its block?

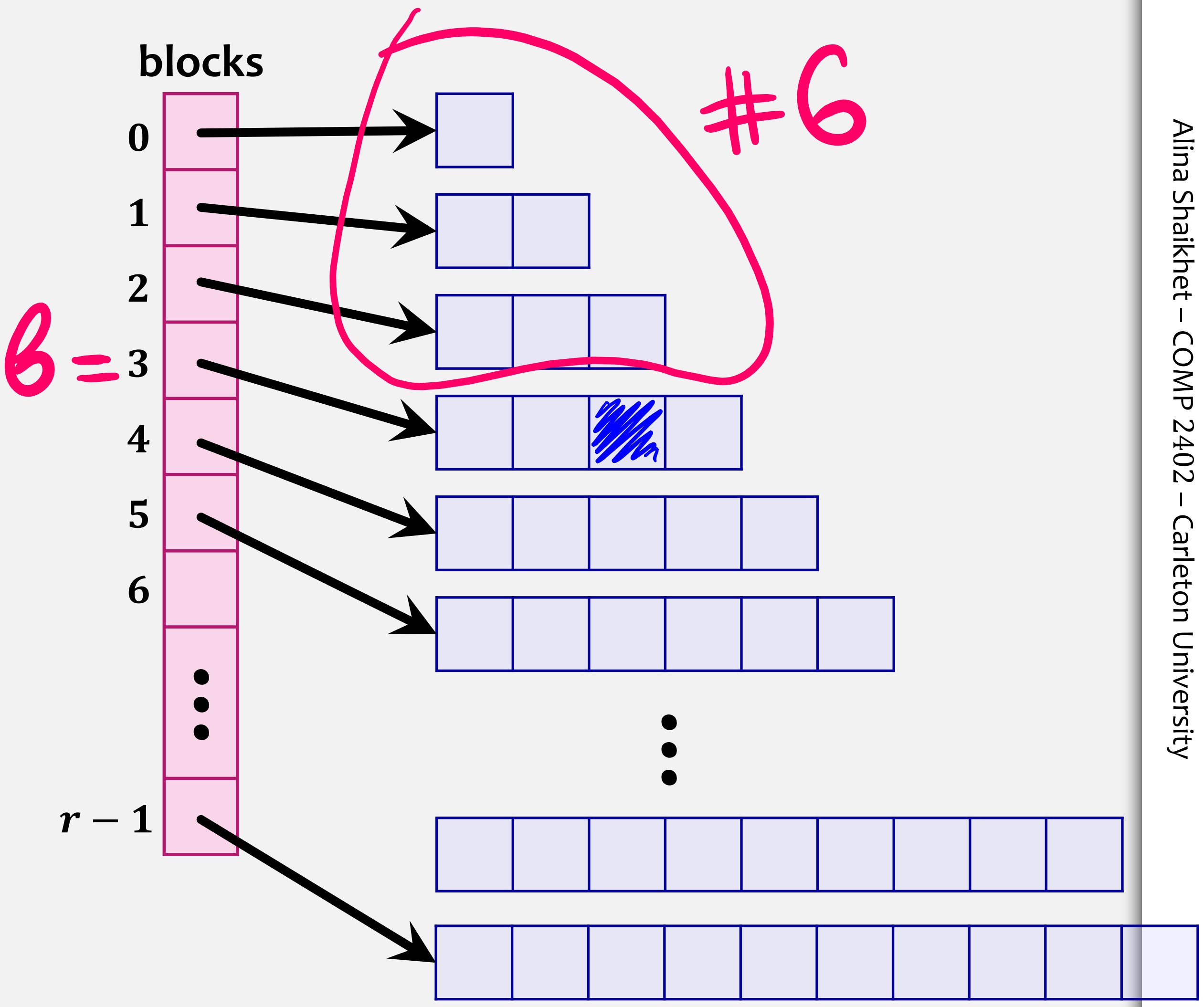
If index $\boldsymbol{i}$ is in block $\boldsymbol{b}$, then the number of elements in blocks $\mathbf{0}, \ldots, \boldsymbol{b}-\mathbf{1}$ is

$$\frac{\boldsymbol{b}(\boldsymbol{b}+\mathbf{1})}{\mathbf{2}}$$

Therefore, $\boldsymbol{i}$ is stored at location

$$\boldsymbol{j}=\boldsymbol{i}-\frac{\boldsymbol{b}(\boldsymbol{b}+\mathbf{1})}{\mathbf{2}}$$

within block $\boldsymbol{b}$.



How do we determine the value of $\boldsymbol{b}$?

RootishArrayStack

How do we determine the value of $\boldsymbol{b}$?

The number of elements that have indices less than or equal to $\boldsymbol{i}$ is $\boldsymbol{i}+\mathbf{1}$.

The number of elements in blocks $\mathbf{0},..,\boldsymbol{b}$ is

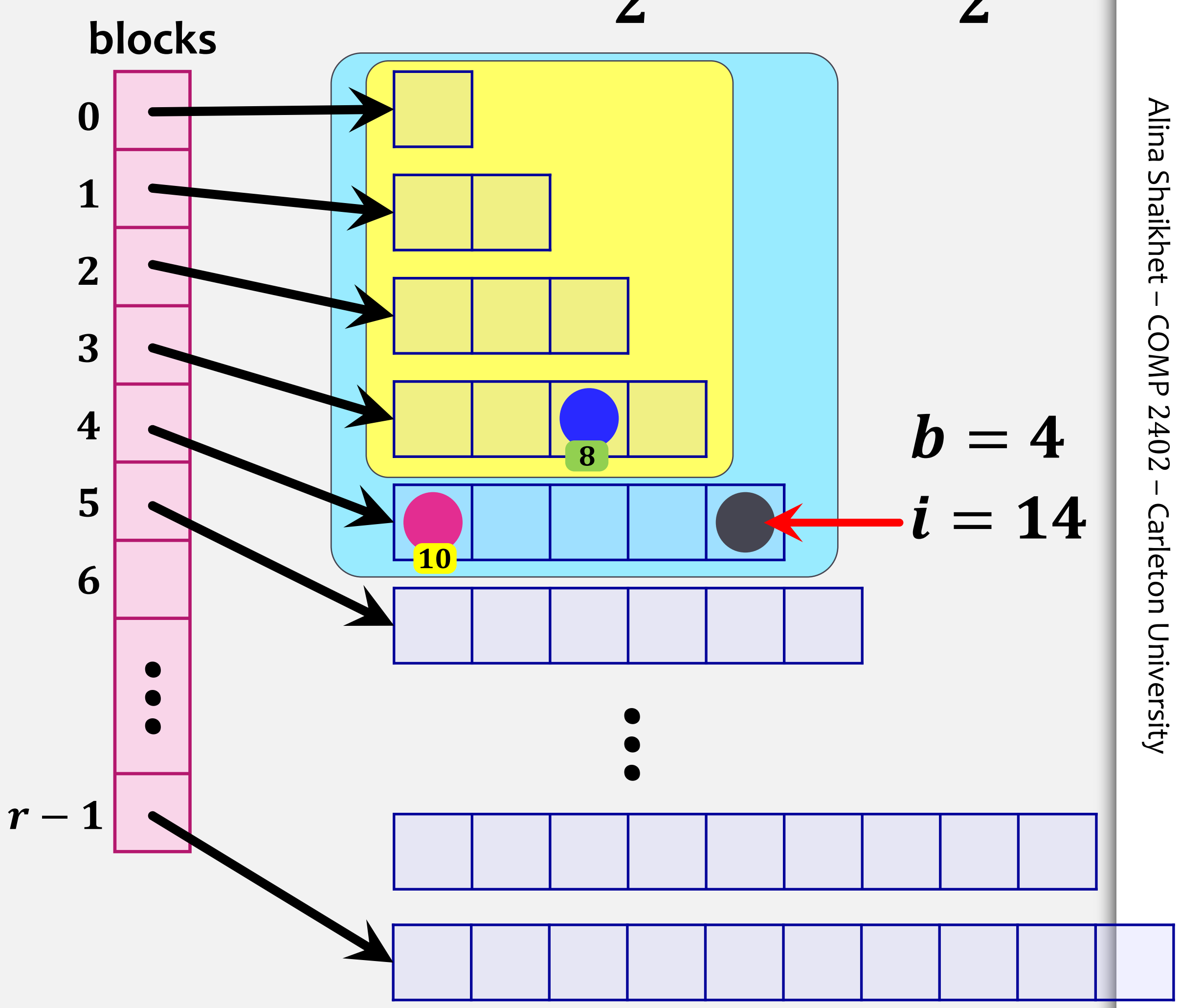
0, ... , 4

$$\frac{(\boldsymbol{b}+\mathbf{1})(\boldsymbol{b}+\mathbf{2})}{\mathbf{2}}$$

Therefore, $\boldsymbol{b}$ is the **smallest integer** such that

$$\frac{(\boldsymbol{b}+\mathbf{1})(\boldsymbol{b}+\mathbf{2})}{\mathbf{2}} \geq \boldsymbol{i}+\mathbf{1}$$

$$\frac{(\boldsymbol{b}+\mathbf{1})(\boldsymbol{b}+\mathbf{2})}{\mathbf{2}} = \frac{\mathbf{5}\cdot\mathbf{6}}{\mathbf{2}} = \mathbf{15}$$



Alina Shaikhet – COMP 2402 – Carleton University

RootishArrayStack

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\frac{(b+1)(b+2)}{2} \geq i + 1$$

$$(b+1)(b+2) \geq 2i + 2$$

$$b^2 + 2b + b + 2 \geq 2i + 2$$

$$b^2 + 3b - 2i \geq 0$$

The quadratic equation $b^2 + 3b - 2i = 0$ has two solutions:

$$b = \frac{-3 + \sqrt{9 + 8i}}{2} \qquad b = \cancel{\frac{-3 - \sqrt{9 + 8i}}{2}} < 0$$

We want the **smallest integer** b such that $b \geq \frac{-3 + \sqrt{9 + 8i}}{2}$

$$b = \left\lceil \frac{-3 + \sqrt{9 + 8i}}{2} \right\rceil$$

int i2b(i):

```
double db = (-3.0 + Math.sqrt(9 + 8*i)) / 2.0;
int b = (int)Math.ceil(db);
return b;
```

get/set operations

T get(i):

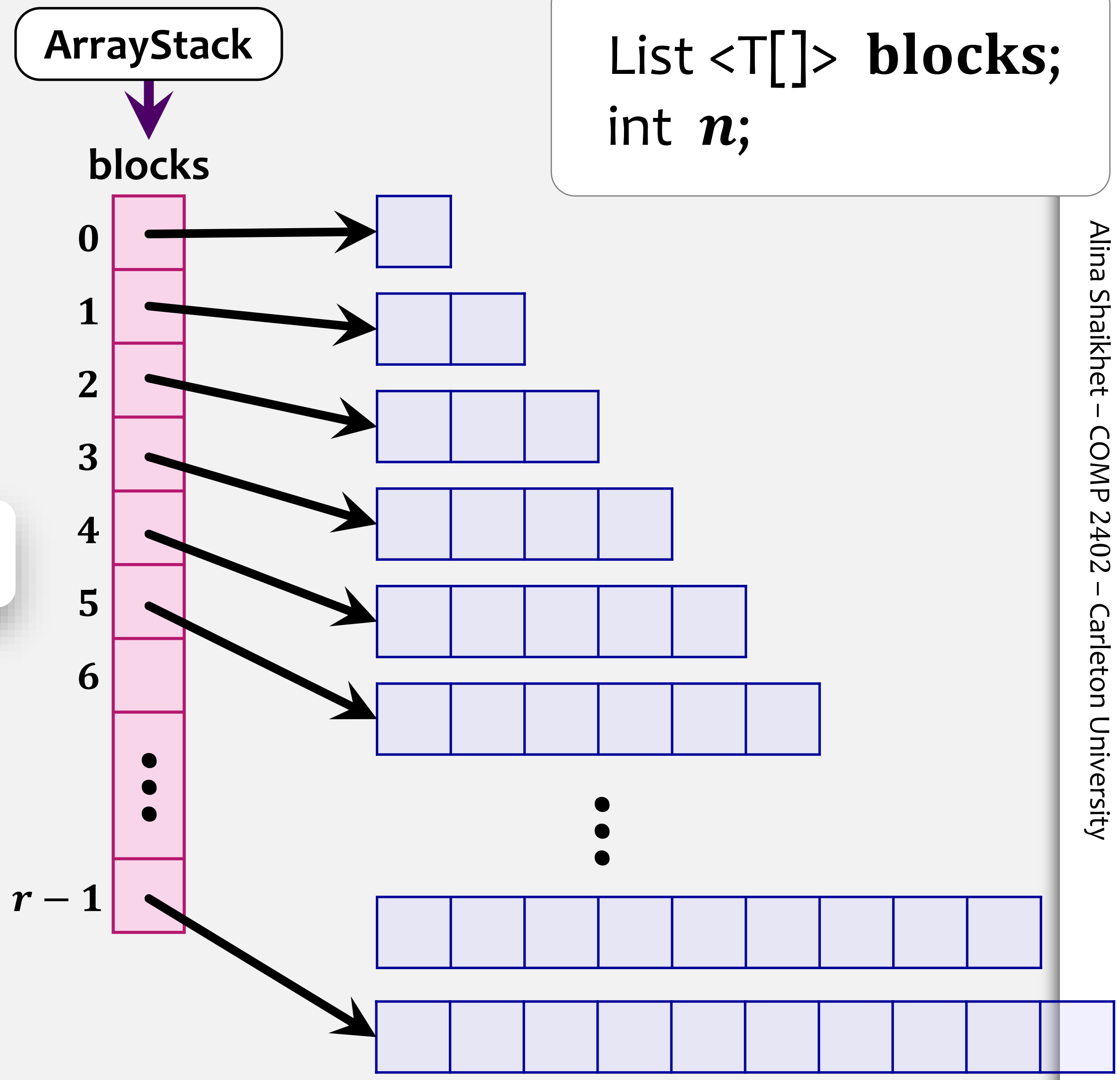
check bounds;
$b = \text{i2b}(i)$;
$j = i - b\,(b+1)/2$;
return $\underbrace{\text{blocks.get}(b)}_{\text{array}}[j]$;

T set(i, x):

check bounds;
$b = \text{i2b}(i)$;
$j = i - b\,(b+1)/2$;
$y = \text{blocks.get}(b)[j]$;
$\text{blocks.get}(b)[j] = x$;
return y;

$O(1)$

List <T[]> **blocks;**
int n;



add operation

void add(x): // append
    check bounds;
    r = blocks.size();
    if $\left(\frac{r(r+1)}{2} < n+1\right)$ then grow();
    $n++$;
    set($n-1, x$);

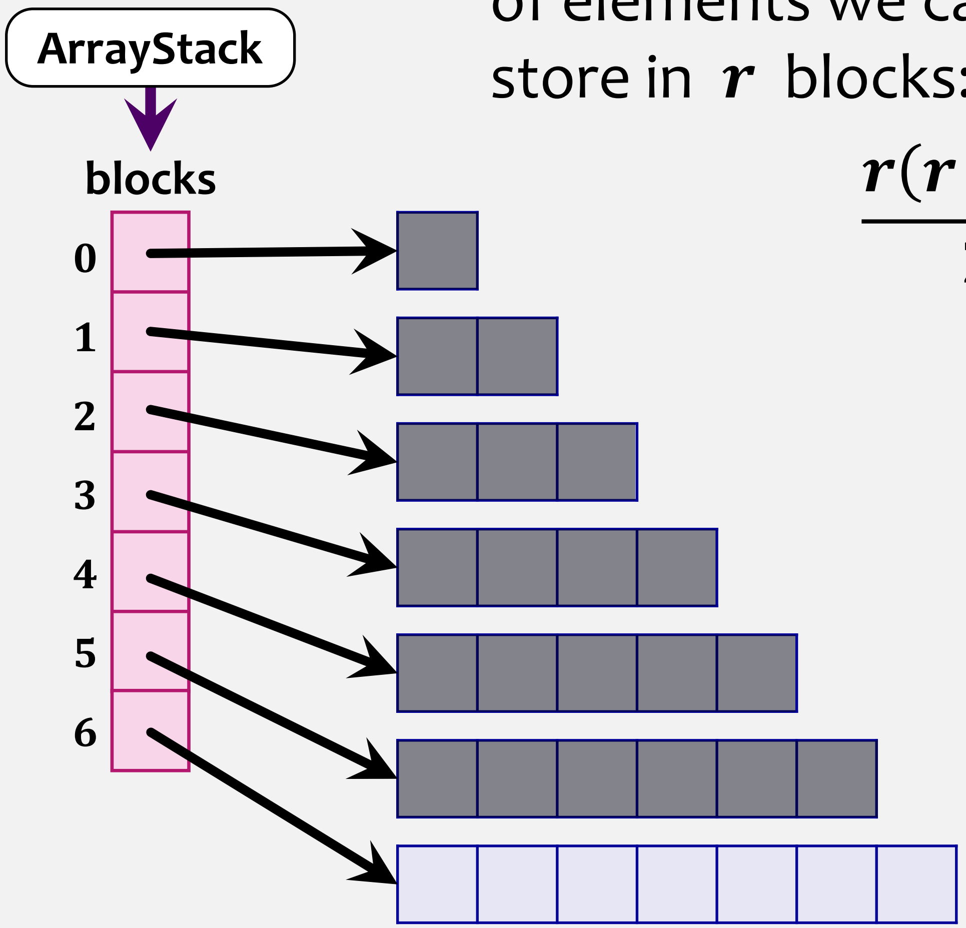
void add(i, x):
    check bounds;
    r = blocks.size();
    if $\left(\frac{r(r+1)}{2} < n+1\right)$ then grow();
    $n++$;
    for ($j = n-1$; $j > i$; $j--$)
        set(j, get($j-1$));
    set(i, x);

shift to the right

$O(1+n-i)$

The maximum number of elements we can store in r blocks:

$$\frac{r(r+1)}{2}$$



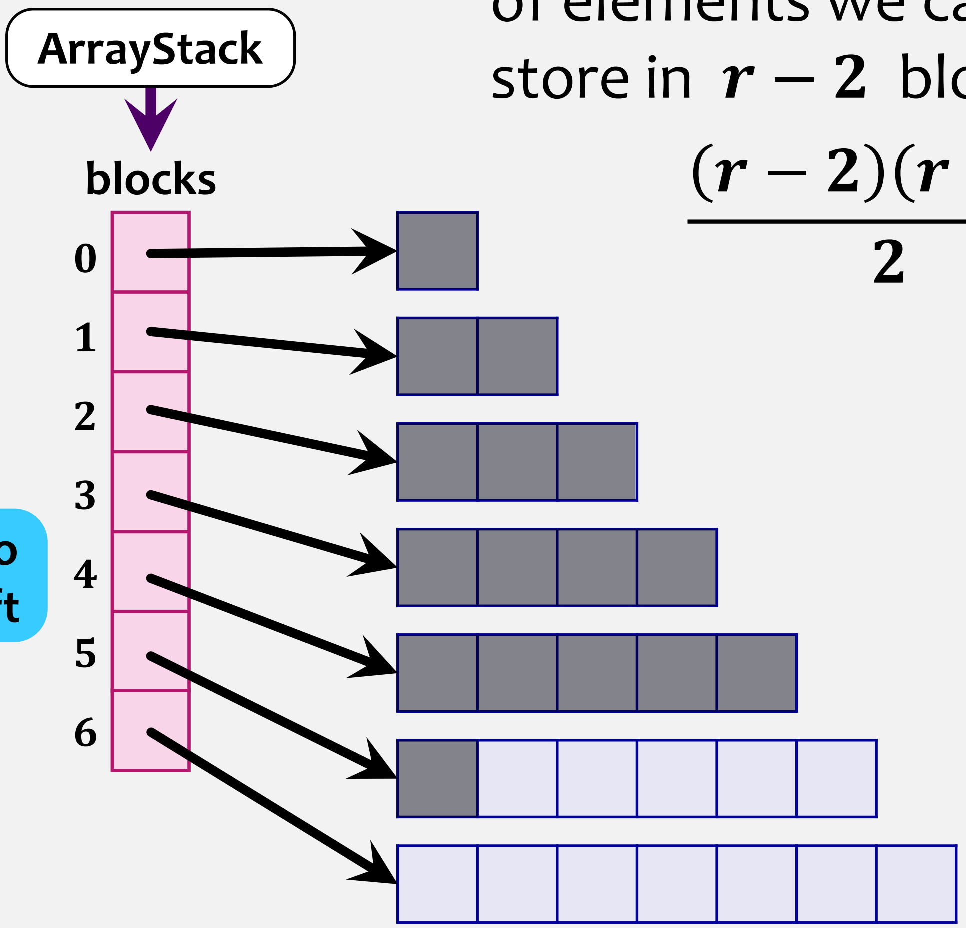
Alina Shaikhet – COMP 2402 – Carleton University

remove operation

T remove(i):

    check bounds;
    $x = \text{get}(i)$;
    for $(j = i;\ j < n - 1;\ j++)$
        $\text{set}(j, \text{get}(j + 1))$;
    $n--$;
    $r = \text{blocks.size}()$;
    if $\left(\frac{(r-2)(r-1)}{2} \geq n\right)$ then shrink();
    return x;

shift to the left



The maximum number of elements we can store in $r - 2$ blocks:

$$\frac{(r-2)(r-1)}{2}$$

$O(1 + n - i)$

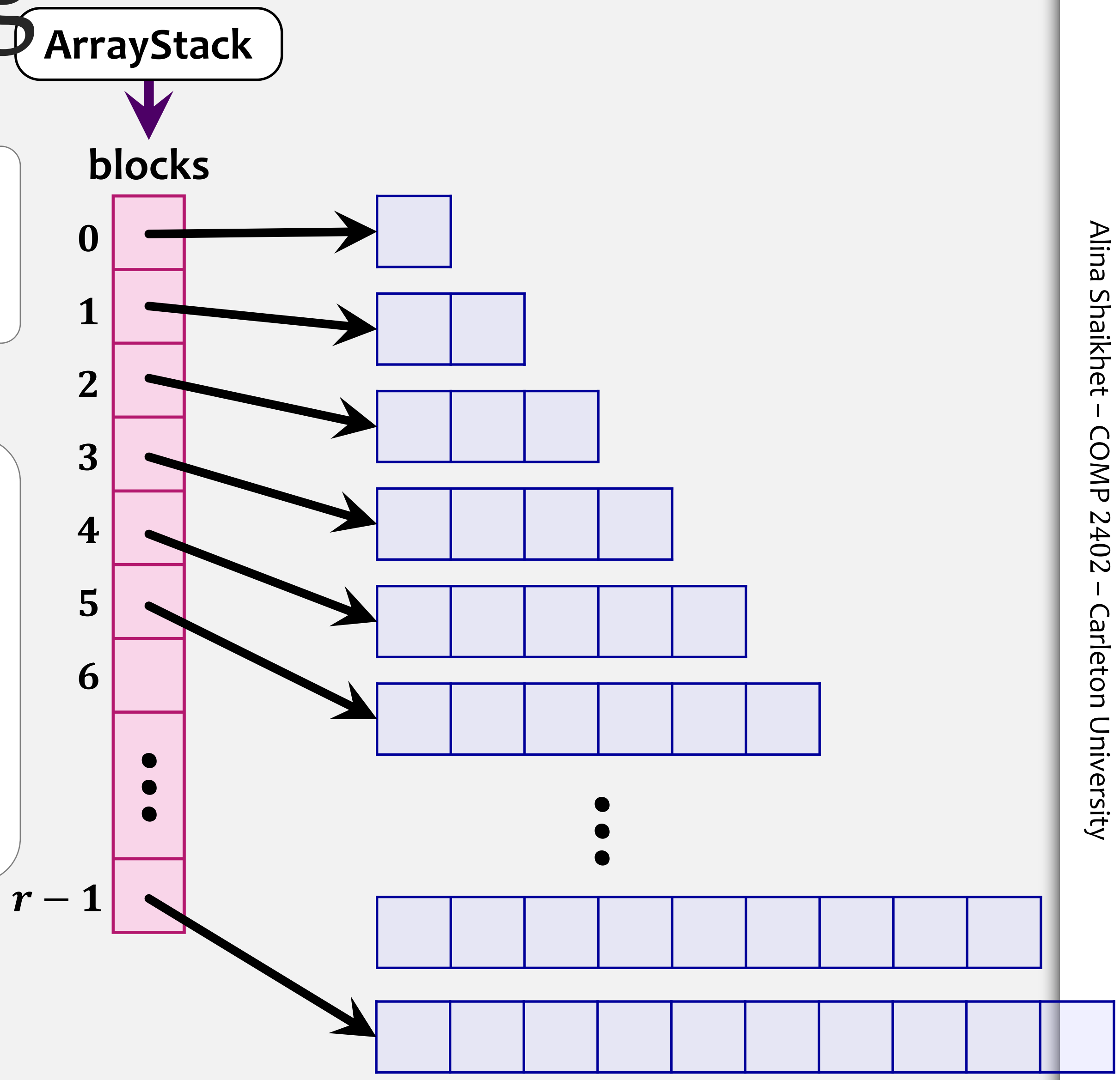
If there are 2 empty blocks at the end, then you shrink

Growing and Shrinking

```
void grow():
      blocks.add(newArray(blocks.size()+1));
```

r = blocks.size()+1

```
void shrink():
      r = blocks.size();
      while (r > 0 && (r−2)(r−1)/2 ≥ n)
            blocks.remove(blocks.size()−1);
            r−−;
```



immediately after a call to grow() or shrink():

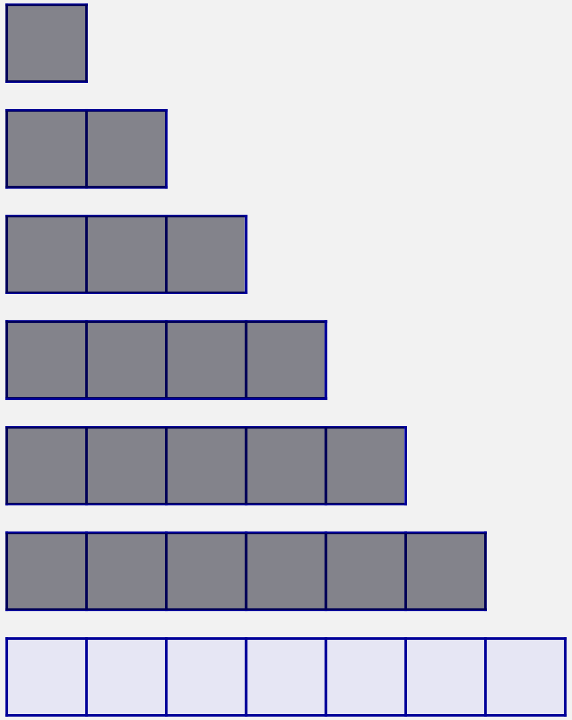
- the final block is completely empty, and
- all other blocks are completely full.

Alina Shaikhet – COMP 2402 – Carleton University

Analysis of growing and shrinking

immediately after a call to grow() or shrink():

- the final block is completely empty, and
- all other blocks are completely full.



Another call to grow() or shrink() will not happen until at least $\boldsymbol{r-1}$ elements have been added or removed.

Despite that grow() and shrink() take $\boldsymbol{O(r)}$ time, this cost can be amortized over at least $\boldsymbol{r-1}$ add$(\boldsymbol{i, x})$ and remove$(\boldsymbol{i})$ operations.

So, the amortized cost of grow() and shrink() is $\boldsymbol{O(1)}$ per operation.

Space Analysis

T[] blocks - pointer to the array
int n - list size

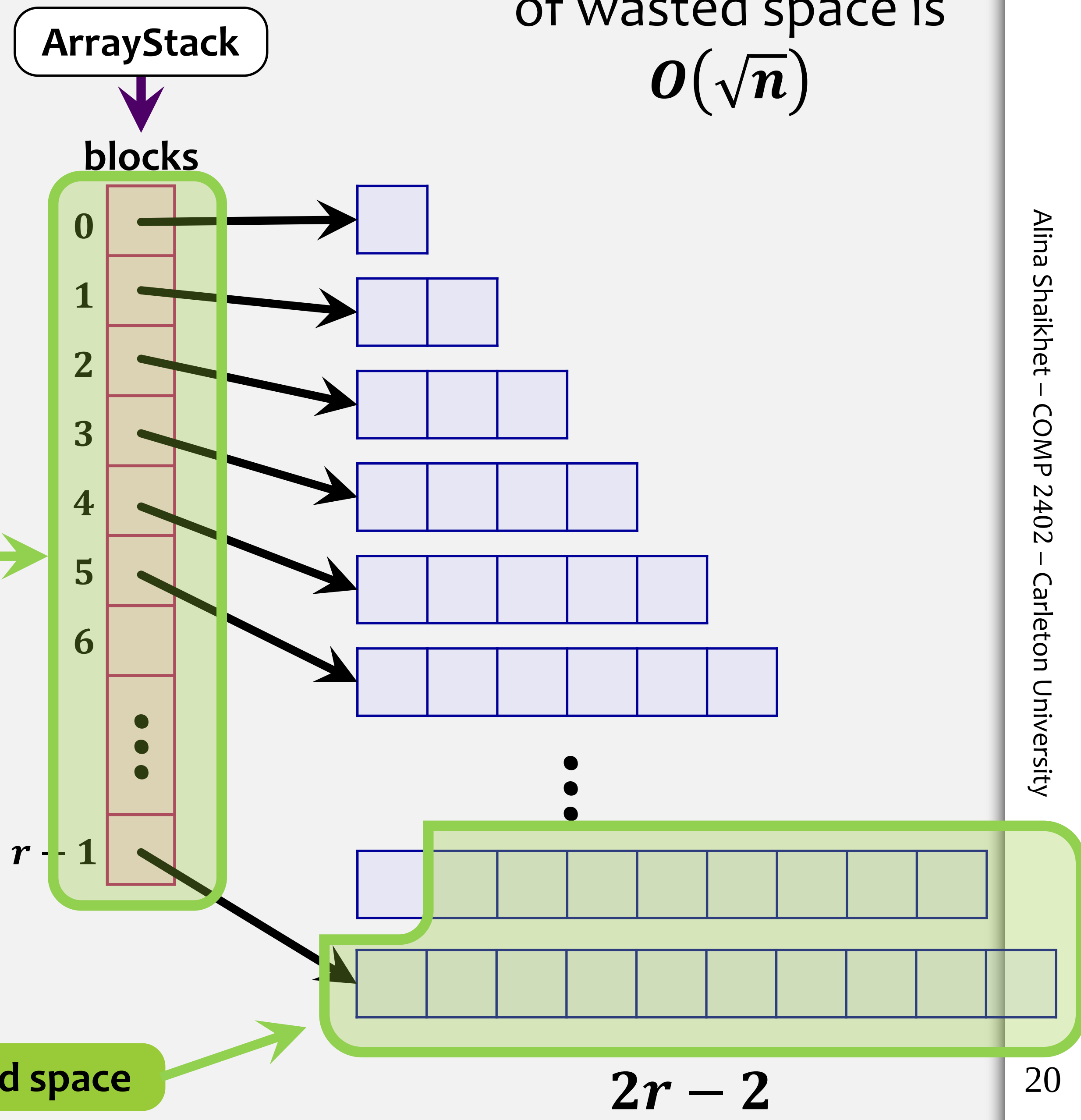
We maintain:

$$\frac{(r-2)(r-1)}{2} \leq n$$

$$r^2 - 3r + 2 \leq 2n$$

$$r \leq \frac{3+\sqrt{1+8n}}{2} = O(\sqrt{n})$$

wasted space



potentially wasted space

the total amount of wasted space is $O(\sqrt{n})$

Theorem 2.5

A **RootishArrayStack** implements the **List** interface. Ignoring the cost of calls to grow() and shrink(), a **RootishArrayStack** supports the operations

- get($\boldsymbol{i}$) and set($\boldsymbol{i}, \boldsymbol{x}$) in $\boldsymbol{O}(\mathbf{1})$ time per operation; and
- add($\boldsymbol{i}, \boldsymbol{x}$) and remove($\boldsymbol{i}$) in $\boldsymbol{O}(\mathbf{1} + \boldsymbol{n} - \boldsymbol{i})$ time per operation.

Furthermore, beginning with an empty **RootishArrayStack**, any sequence of $\boldsymbol{m}$ add($\boldsymbol{i}, \boldsymbol{x}$) and remove($\boldsymbol{i}$) operations results in a total of $\boldsymbol{O}(\boldsymbol{m})$ time spent during all calls to grow() and shrink().

The space (measured in words) used by a **RootishArrayStack** that stores $\boldsymbol{n}$ elements is $\boldsymbol{n} + \boldsymbol{O}(\sqrt{\boldsymbol{n}})$.