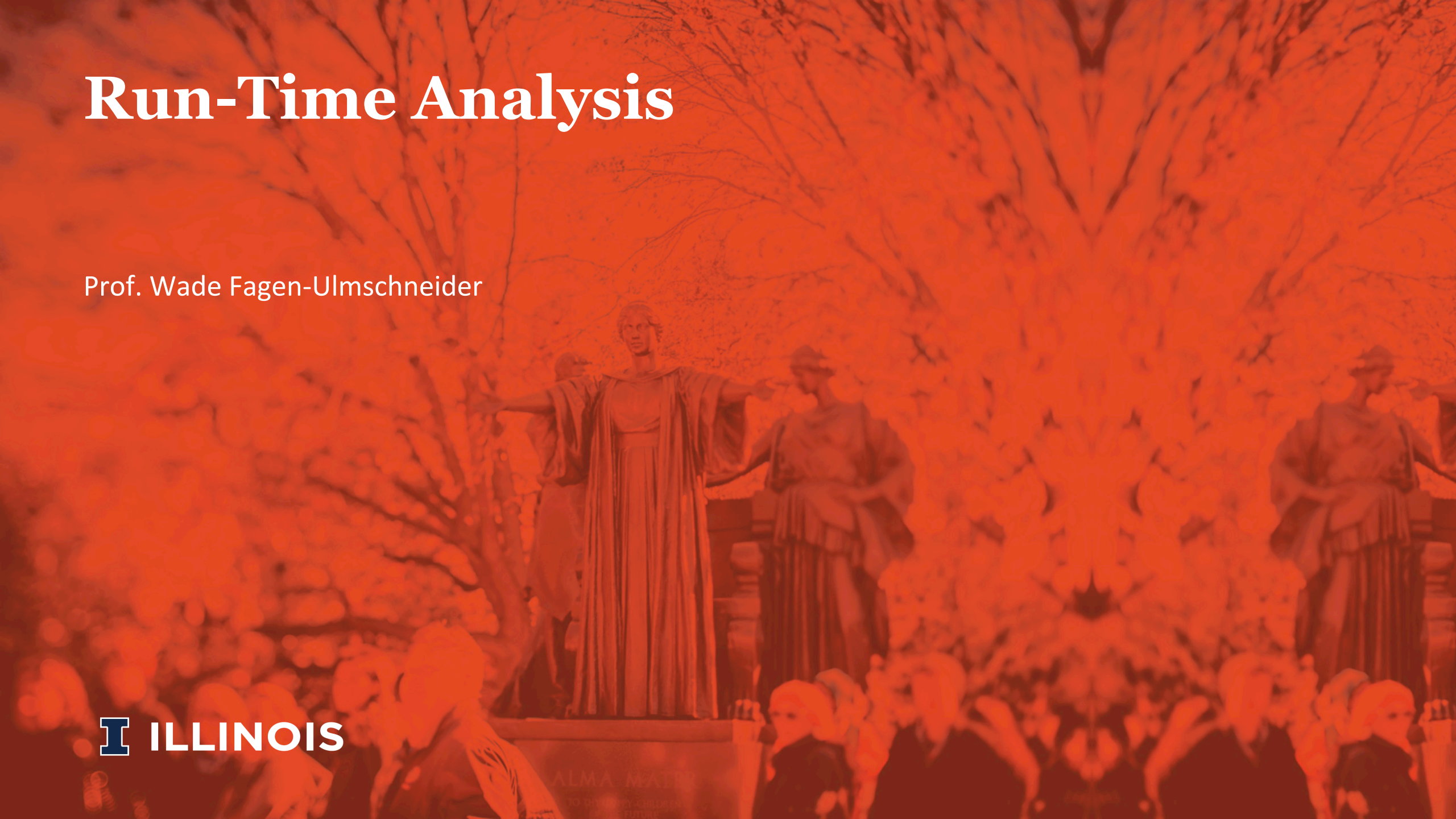
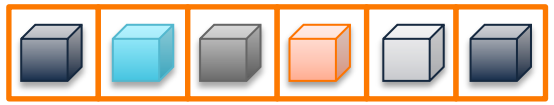
Run-Time Analysis

Prof. Wade Fagen-Ulmschneider

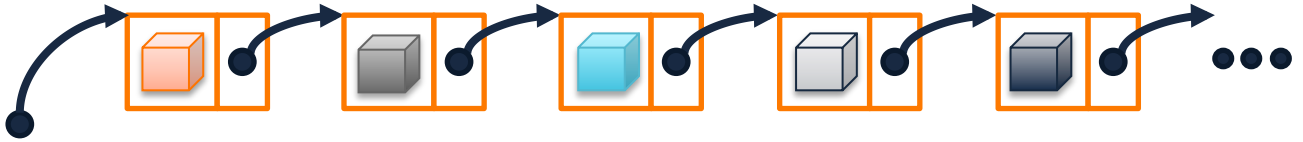


Run-time Analysis allows us to formalize a method of comparing the speed of an algorithm as the size of input grows.

Array

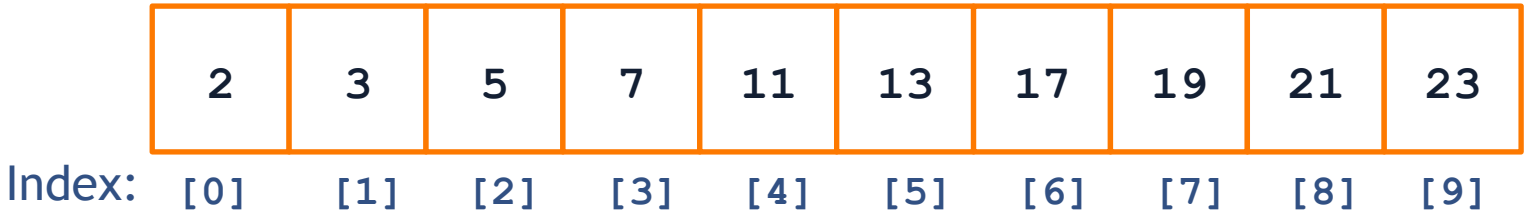


List



Example:

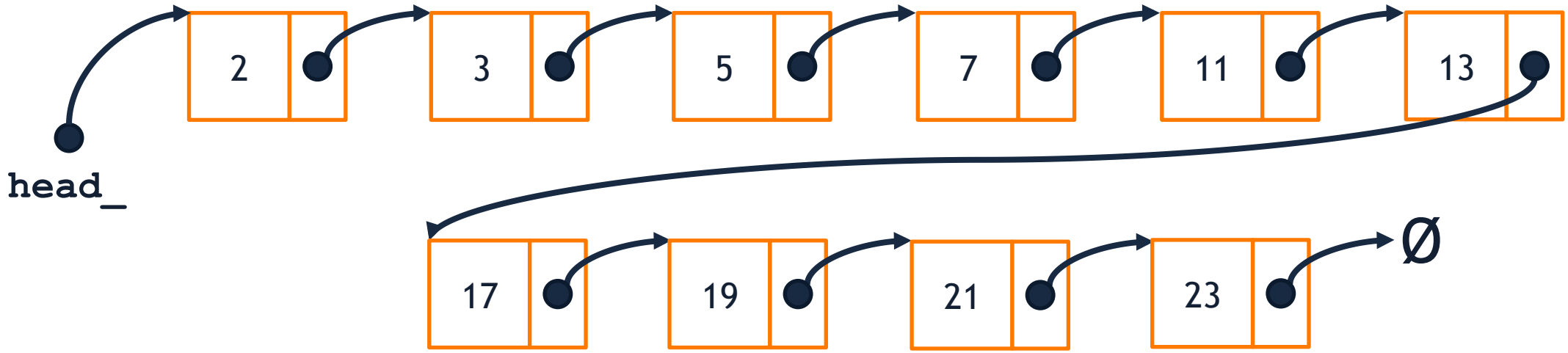
Objective: Access a given index.



Calculate offset (= sizeof(data type) * index) to find index, so it is O(1)

Example:

Objective: Access a given index.



Need to travel through the list to find index, so it is O(n)

Objective: Access a given index:

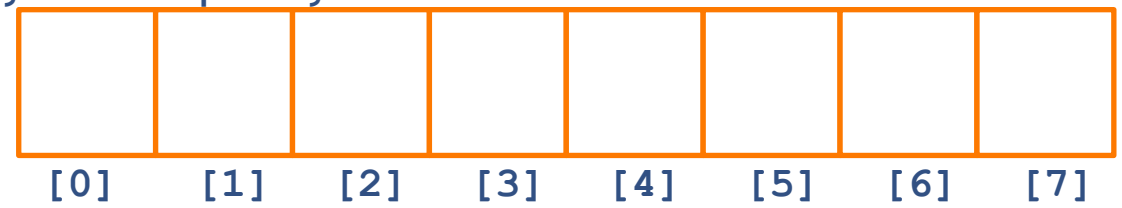
	Array	Linked List
Access [3]		
Access [4285]		
Access [1250000]		
Based on n pieces of data:		

Array Resize Strategy

Objective: Resize an array:

Array with capacity=6:

Array with capacity=8:

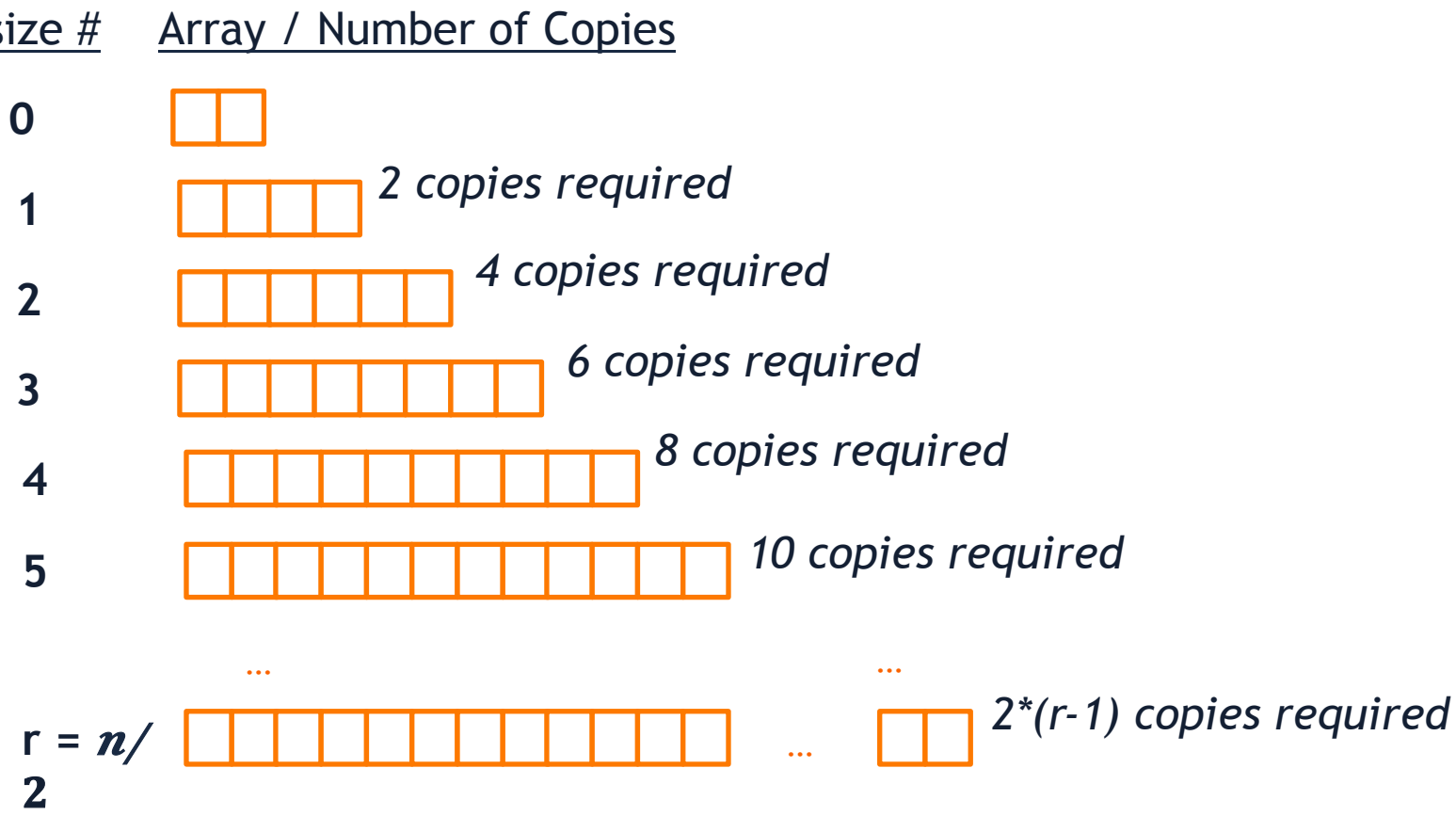


Strategy #1: When the array is full, add two to the capacity.

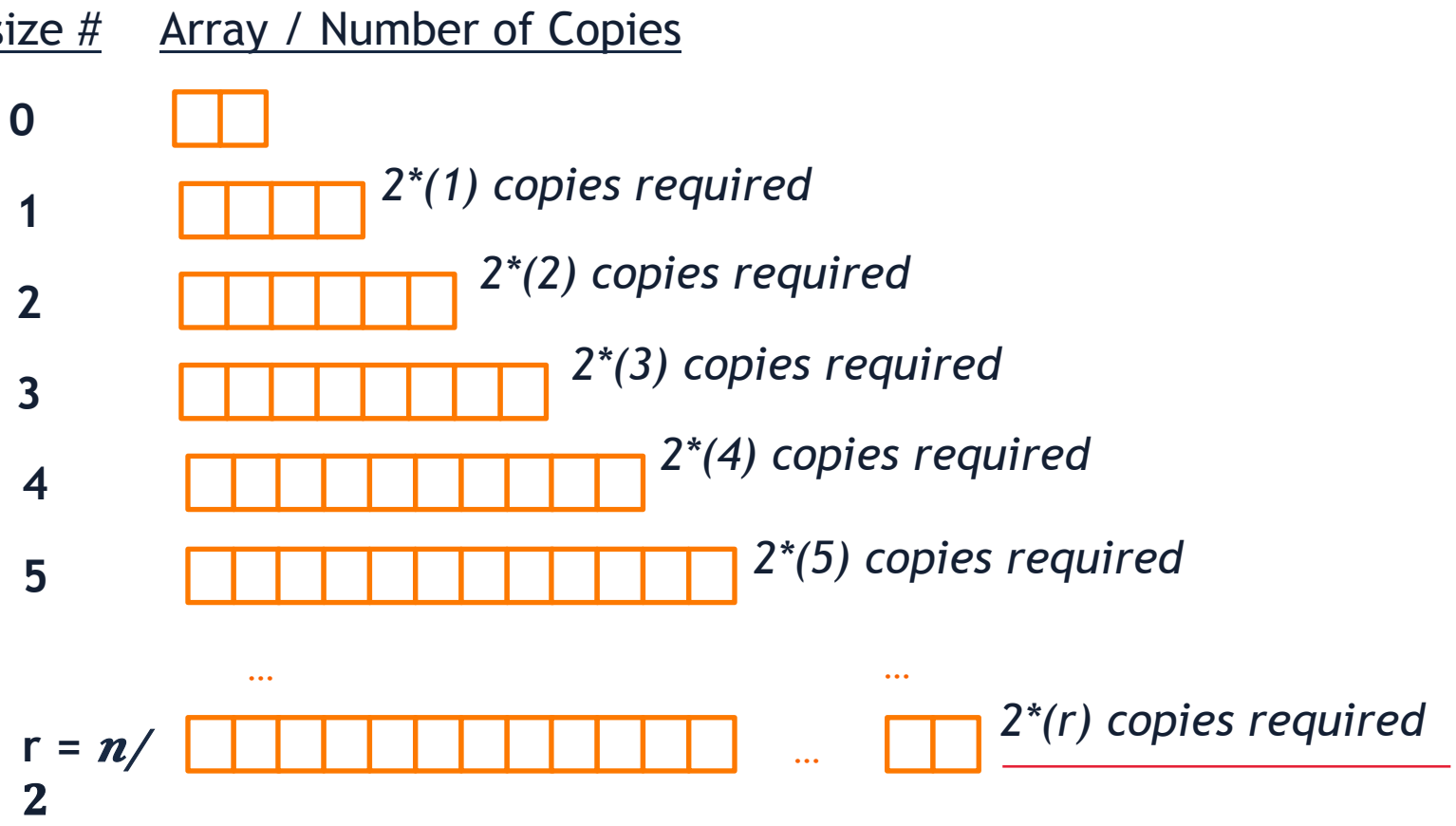
Resize #	Array / Number of Copies
0	
1	
2	
3	
4	
5	
...	
r	 ...

...where r = $n/2$

Strategy #1: When the array is full, add two to the capacity.

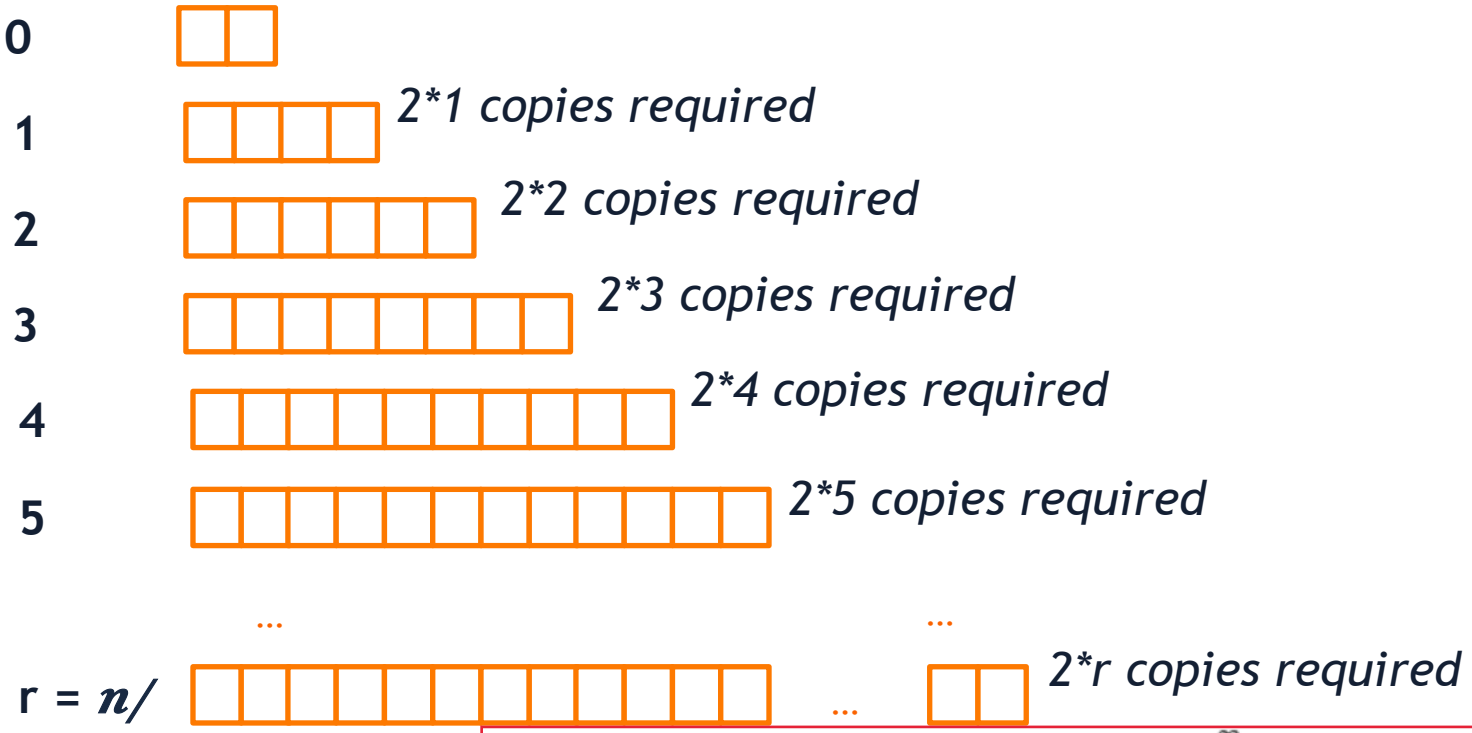


Strategy #1: When the array is full, add two to the capacity.



Strategy #1: When the array is full, add two to the capacity.

Resize # Array / Number of Copies



Sum of all required copies: $\sum_{k=1}^{r}(2k) = 2 \cdot \left(\frac{r(r+1)}{2}\right) = r^2 + r$

Strategy #1: When the array is full, add two to the capacity.

- We have found two equations:
  - Total number of copies: $r^2 + r$
  - Relationship between r and n: $r = n/2$

- Therefore, the total number of copies made:

$$r^2 + r = (n/2)^2 + n/2 = n^2 + 2n/4 = O(n^2)$$

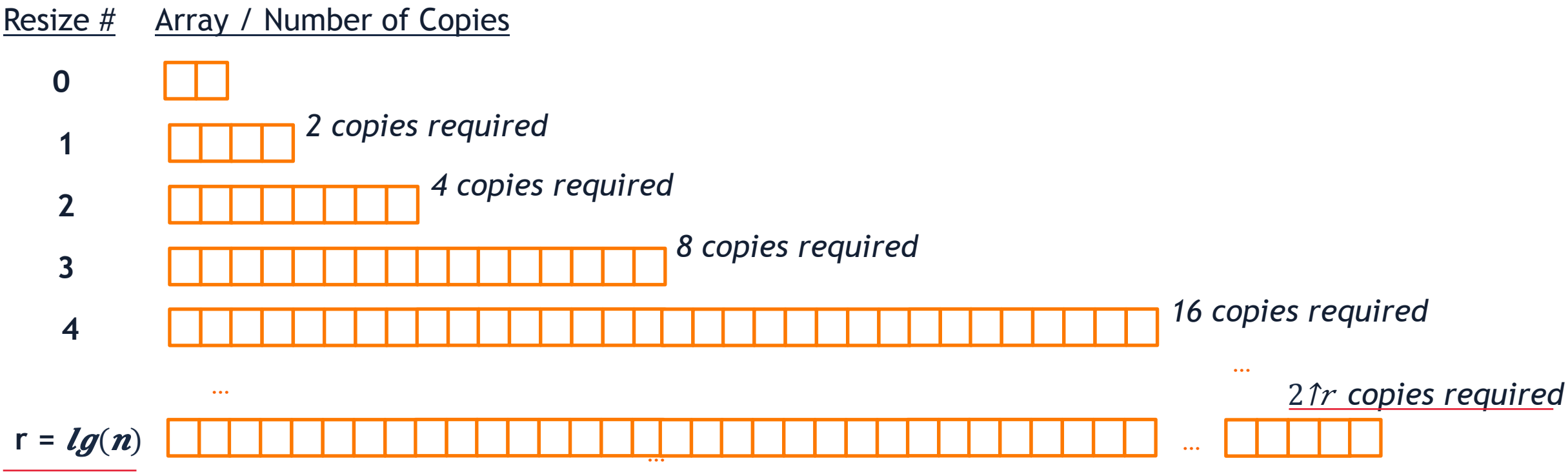
Substituting (n/2) in place of r

Expanding the exponent and simplifying

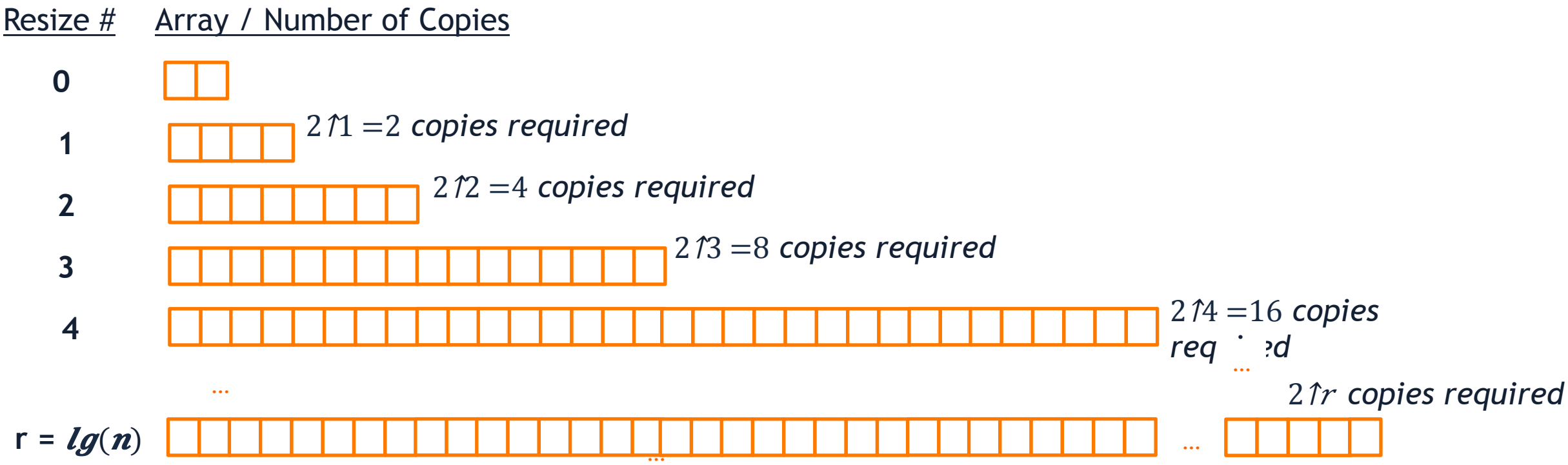
Using Big-O notation

O(n^2)

Strategy #2: When the array is full, double the capacity.



Strategy #2: When the array is full, double the capacity.



Sum of all required copies: $\sum_{k=1}^{r} 2^k = 2*(2^r - 1)$

Solving the summation

Strategy #2: When the array is full, double the capacity.

- We have found two equations:
  - Total number of copies: $2*(2\uparrow r-1)$
  - Relationship between r and n: $r=\lg(n)$

- Therefore, the total number of copies made:

$2(2\uparrow \mathrm{r}-1)$ = $2(2\uparrow \lg(n)-1)$ = $2(n-1)$ = $O(n)$

Substituting lg(n) in place of r

$2^{\lg(n)}$ simplifies to n

Using Big-O notation

Array Resize Strategy

Objective: Resize an array

- Strategy #1 (grow array by +2 each time):
  - Total work done by adding n elements: $O(n^2)$

- Strategy #2 (double array each time):
  - Total work done by adding n elements: $O(n)$
  - Average work per element: $O(n) / n == O(1)$*

* Amortized running time; some operations take longer.

Objective: Access a given index:

	Array	Linked List
Access [3]	1 formula	Visits 4 ListNodes
Access [4285]	1 formula	Visits 4,285 ListNodes
Access [1250000]	1 formula	Visits 1,250,000 ListNodes
Based on n pieces of data:	1 formula	Visits up to n ListNodes

Run-time Analysis

- Run-time Analysis allows us to formalize a method of comparing the speed of an algorithm as the size of input grows.

- We summarize the runtime in “Big-O notation”, leaving only the term that dominates the growth:
  - O(1), constant time
  - O(n), linear time
  - $O(n^2)$, polynomial time