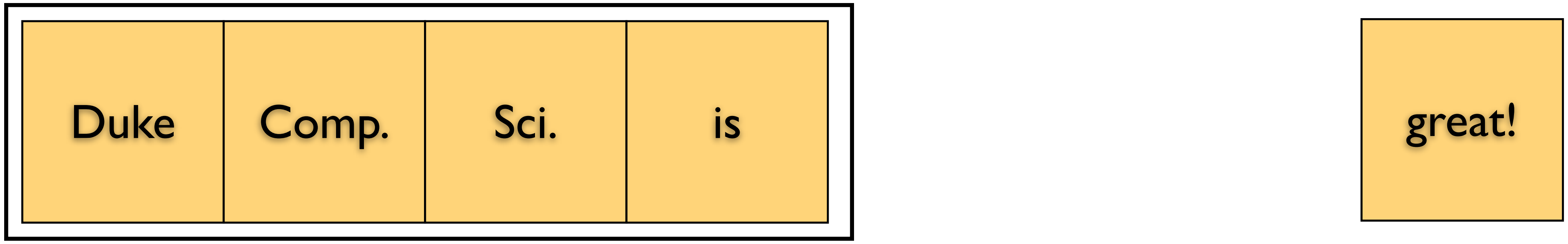
Amortized Analysis

(or: How ArrayLists work)

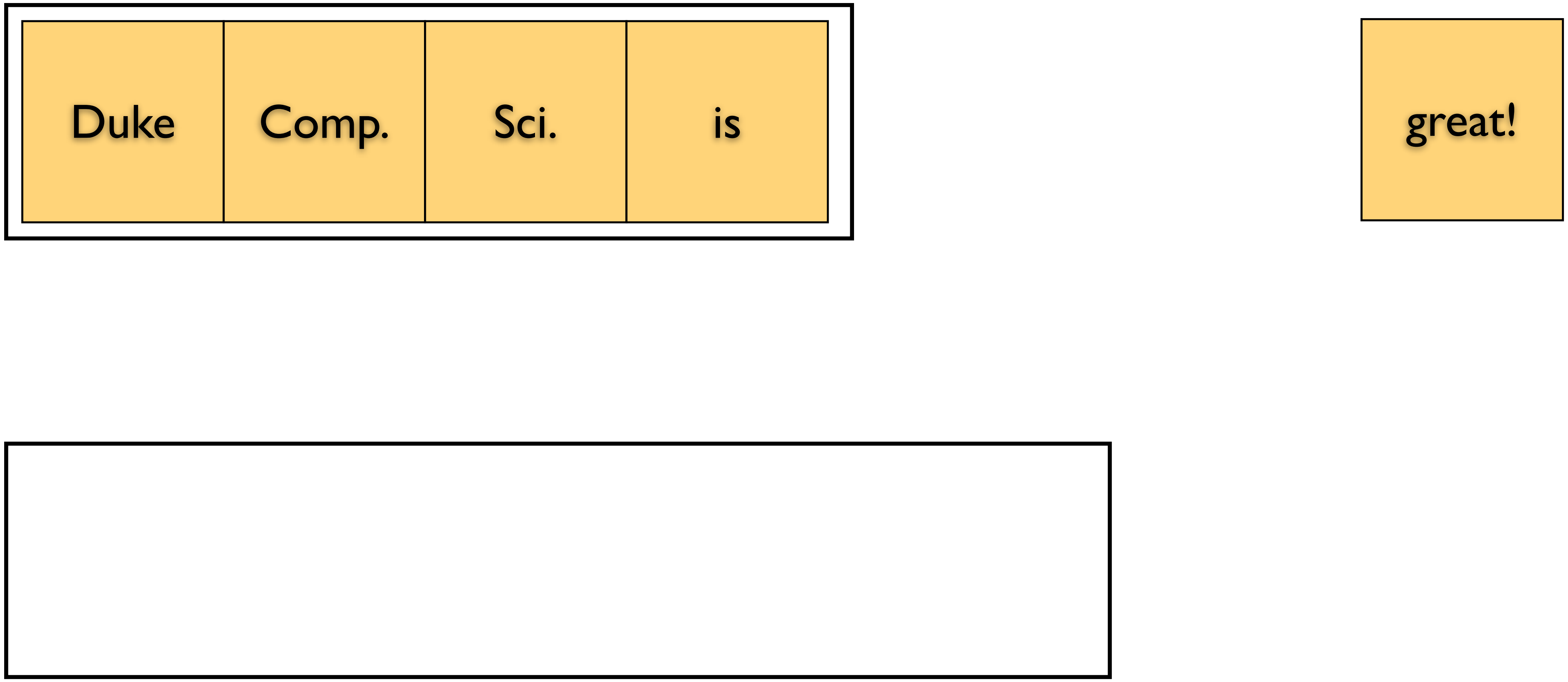
Recall ExpandingArray



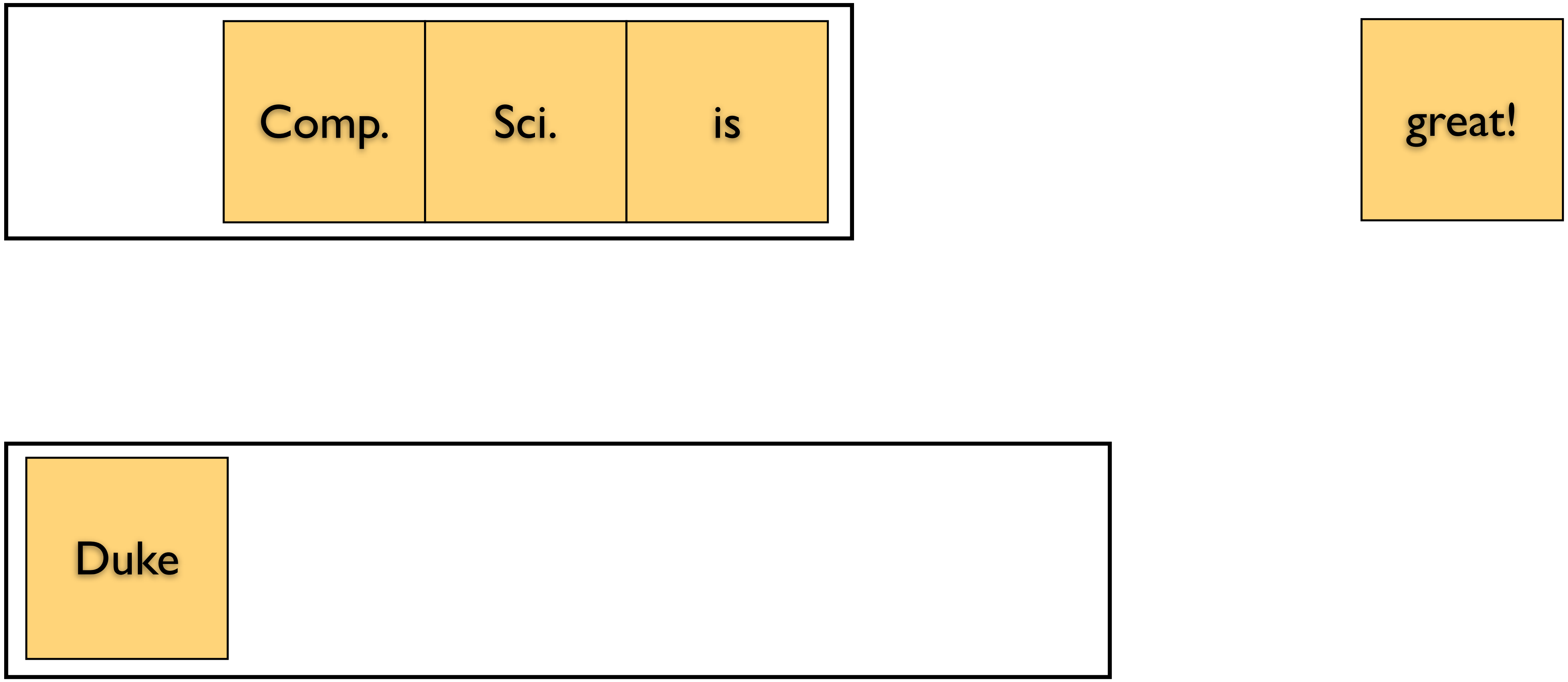
Duke | Comp. | Sci. | is

great!

Recall ExpandingArray



Recall ExpandingArray

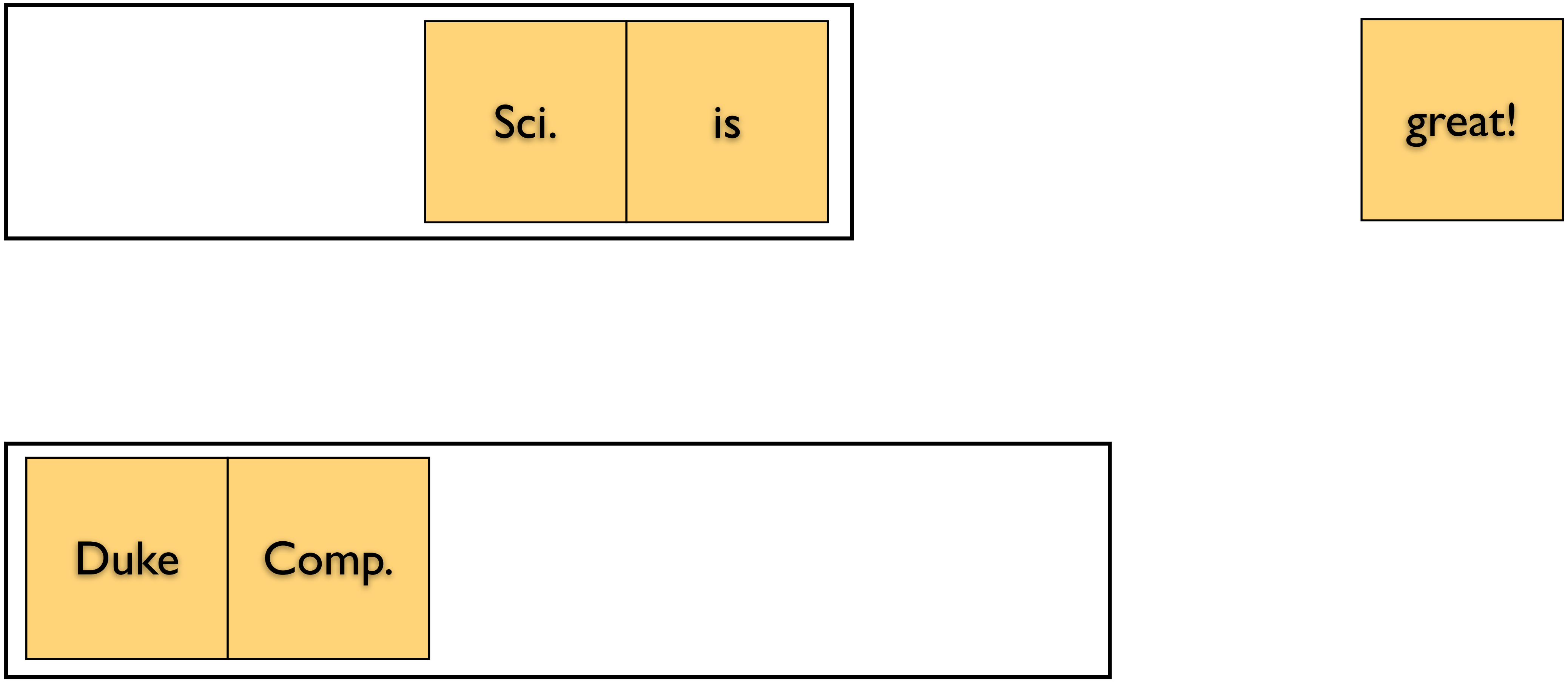


Comp. Sci. is

great!

Duke

Recall ExpandingArray



Sci.

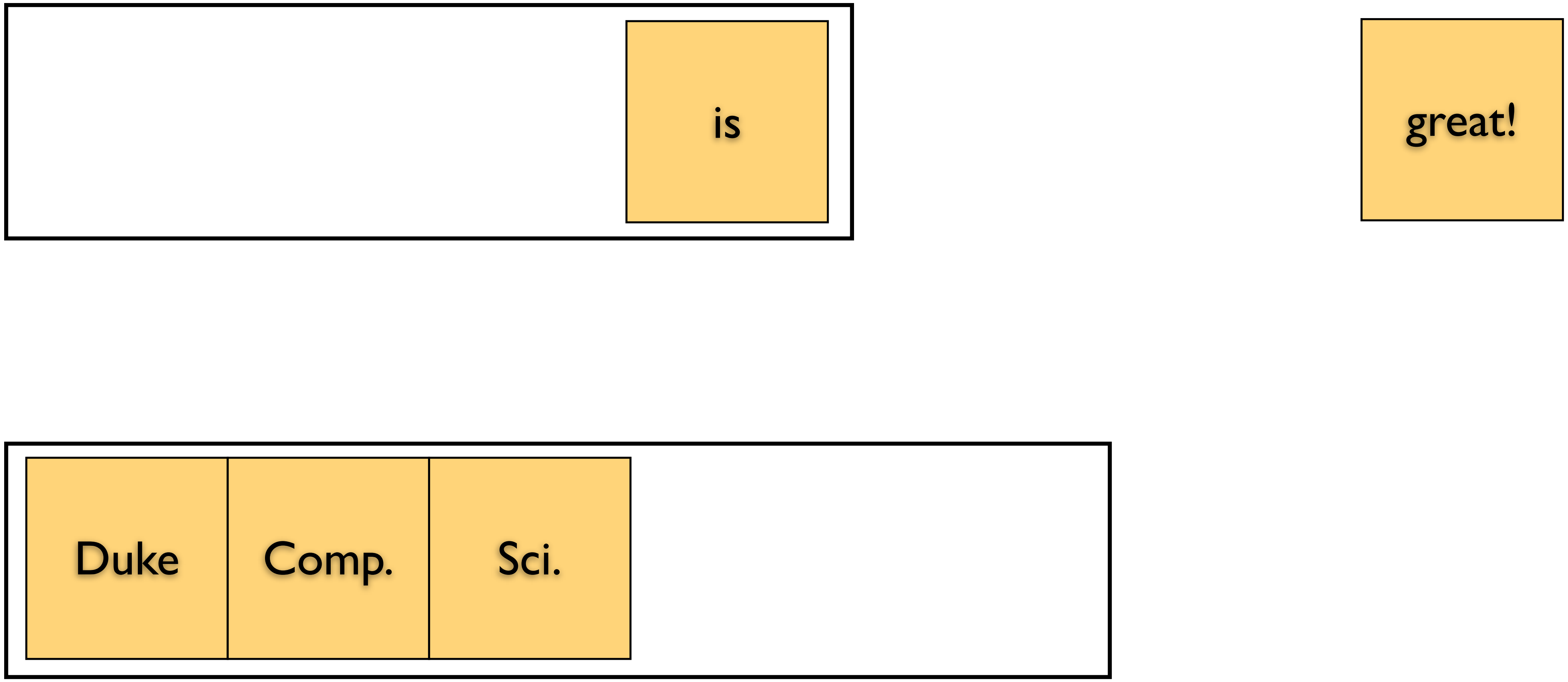
is

great!

Duke

Comp.

Recall ExpandingArray



is

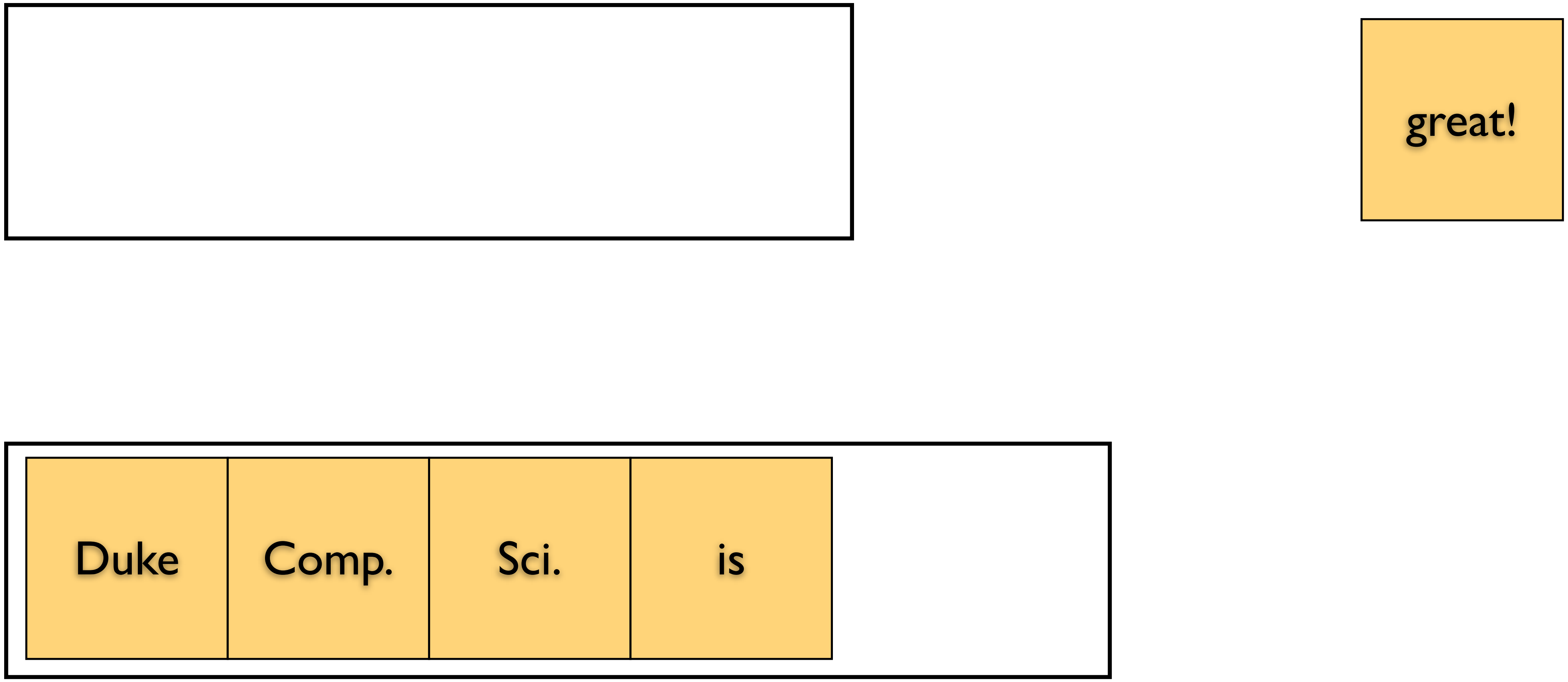
great!

Duke

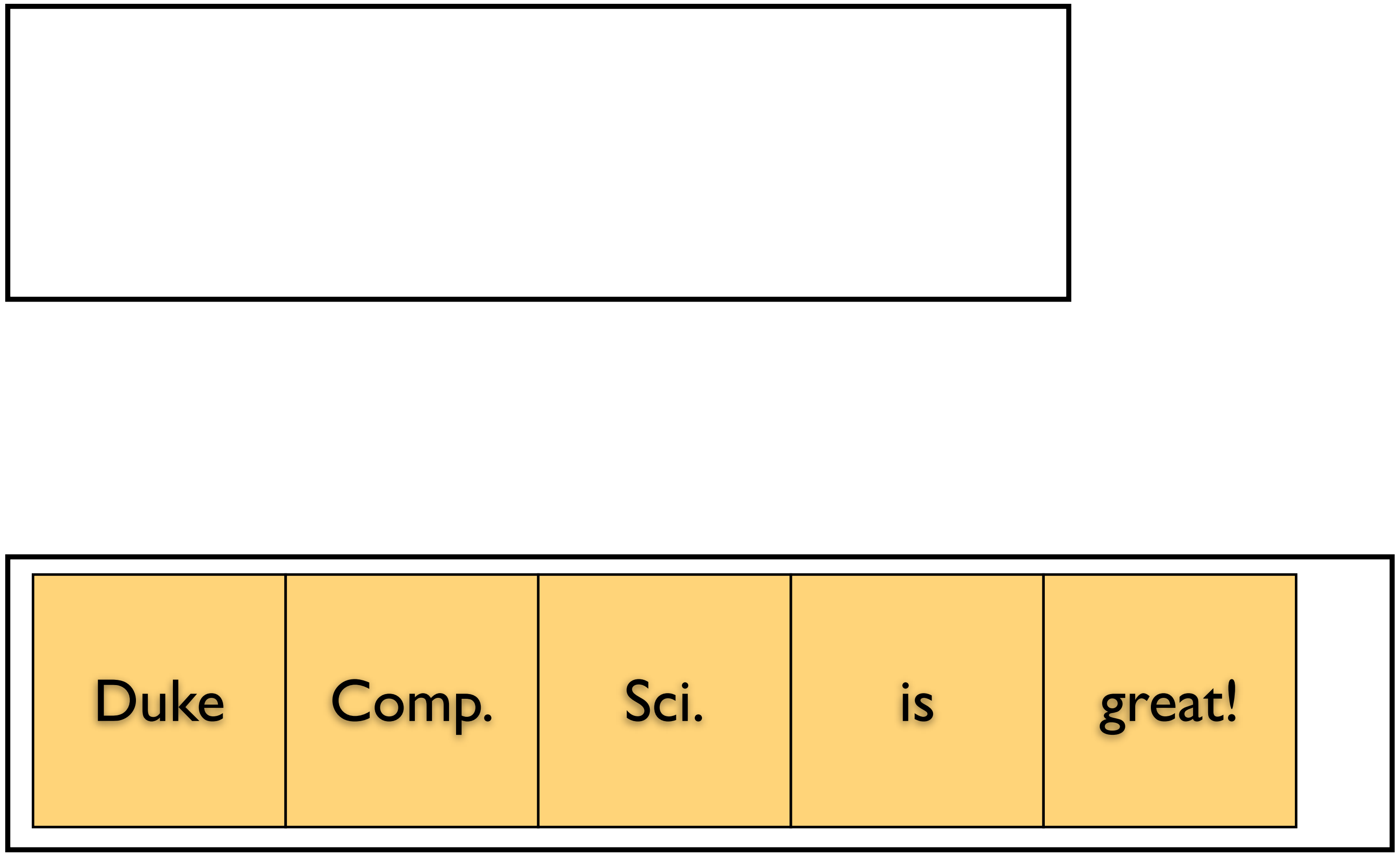
Comp.

Sci.

Recall ExpandingArray



Recall ExpandingArray



Duke	Comp.	Sci.	is	great!

Adding one each time leads to $$\frac{n(n+1)}{2} \in O(n^2)$$

Why not Linked Lists?

Why not Linked Lists?

`ExpandingArray`

.get(): $O(1)$ *You couldn't hope for better!*

.add(): $O(n)$ *Which means $O(n^2)$ for n operations...*

Linked List

Re: DNA: Good at splicing, too!

.get(): $O(n)$ *Which means $O(n^2)$ for n operations...*

.add(): $O(1)$ *Best it can be!*

What we want:

.get(): $O(1)$ *Best it can be!*

.add(): $O(1)$ *Best it can be!*

It can be done!

ArrayList & StringBuilder, for example

Also: StringBuffer, C++'s vector, and Python's list. *Not* Matlab's array.

It can be done!

ArrayList & StringBuilder, for example

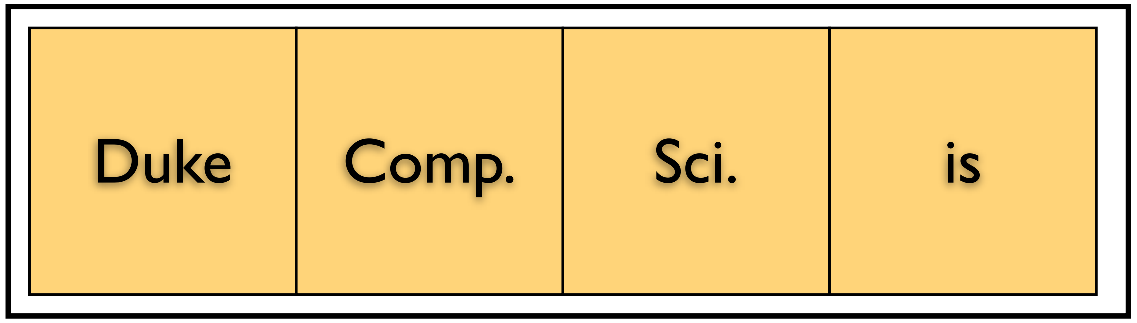
What we want:

.get(): $O(1)$ *Best it can be!*

.add(): $O(1)$ *Best it can be!*

Also: StringBuffer, C++'s vector, and Python's list. *Not* Matlab's array.

Backed by an array!



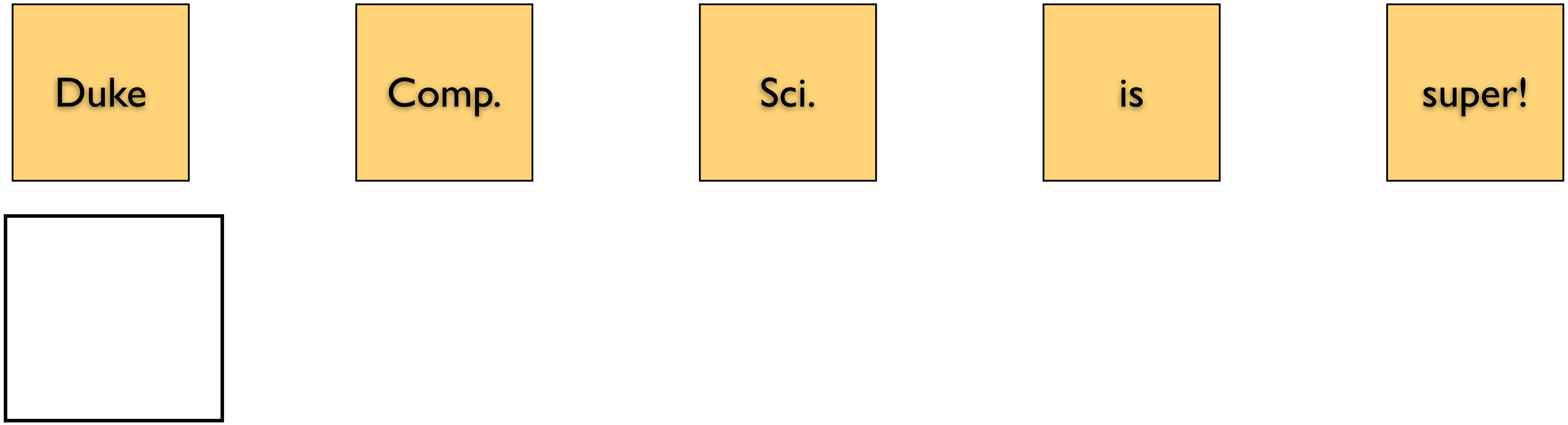
Duke	Comp.	Sci.	is

What we want:

.get(): *O(1)* *Best it can be!*

.add(): *O(...)* *...?*

Add more than one?

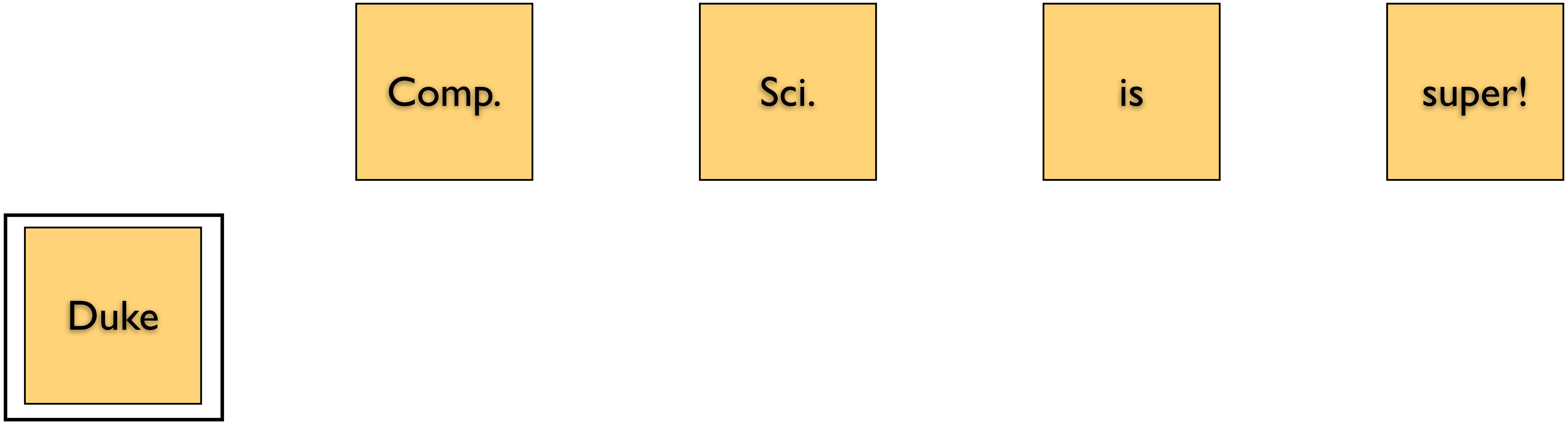


Duke Comp. Sci. is super!

Operations: 0

Adds: 0

Add more than one?



Comp.

Sci.

is

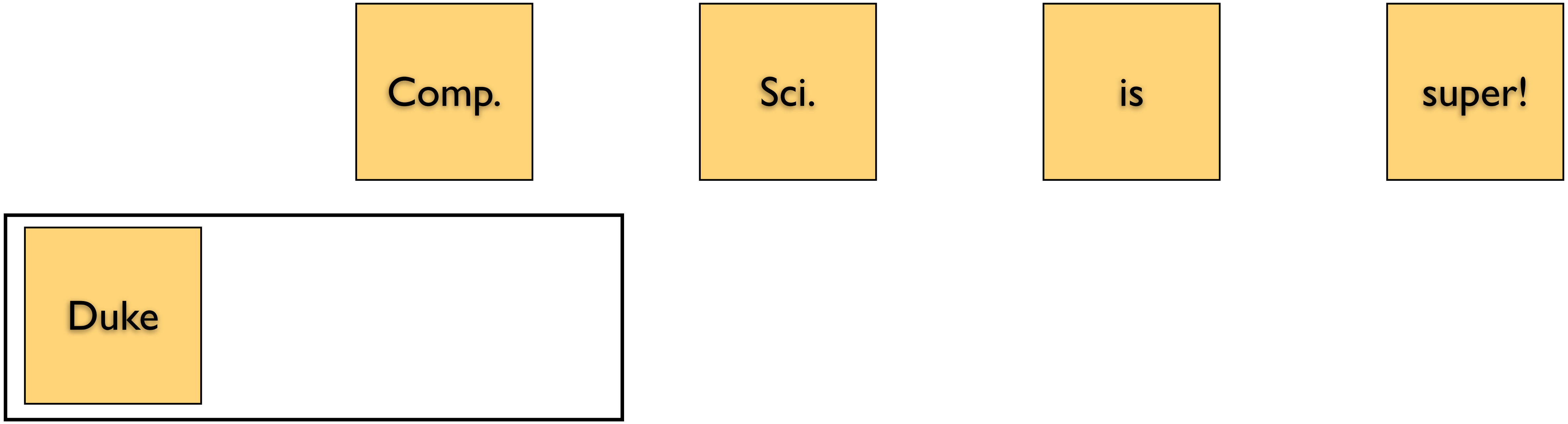
super!

Duke

Operations: 1

Adds: 1

Add more than one?



Comp.

Sci.

is

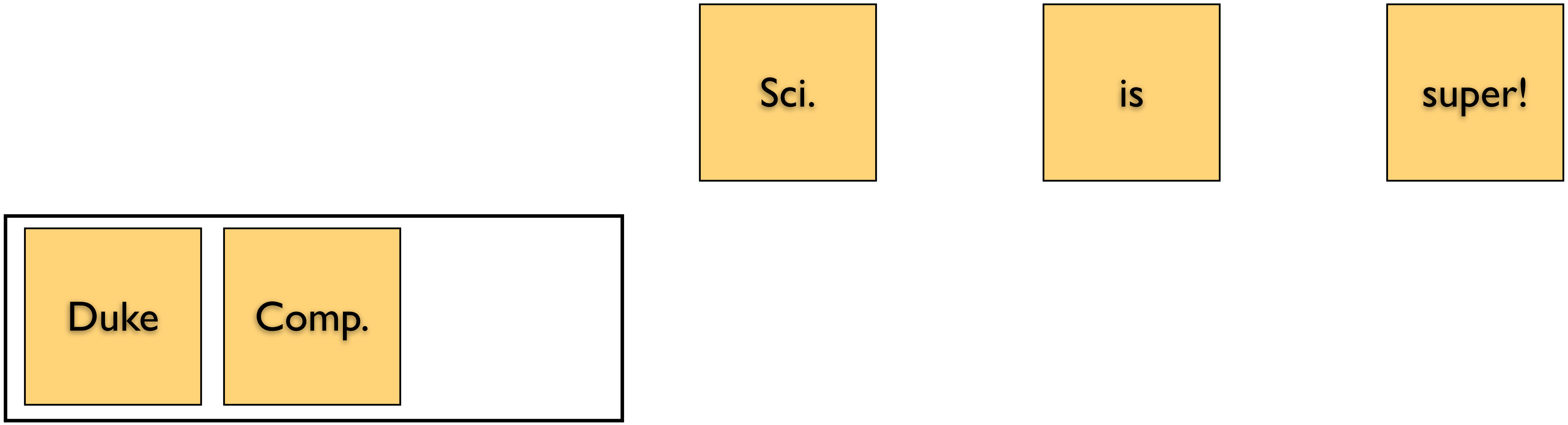
super!

Duke

Operations: 2

Adds: 1

Add more than one?



Sci.

is

super!

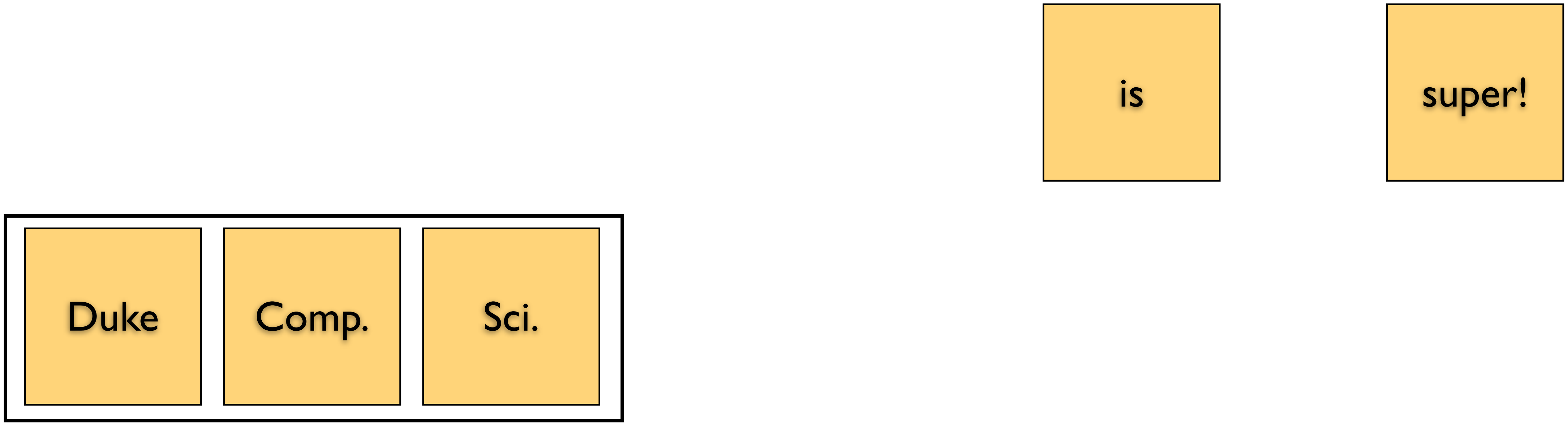
Duke

Comp.

Operations: 3

Adds: 2

Add more than one?



is

super!

Duke

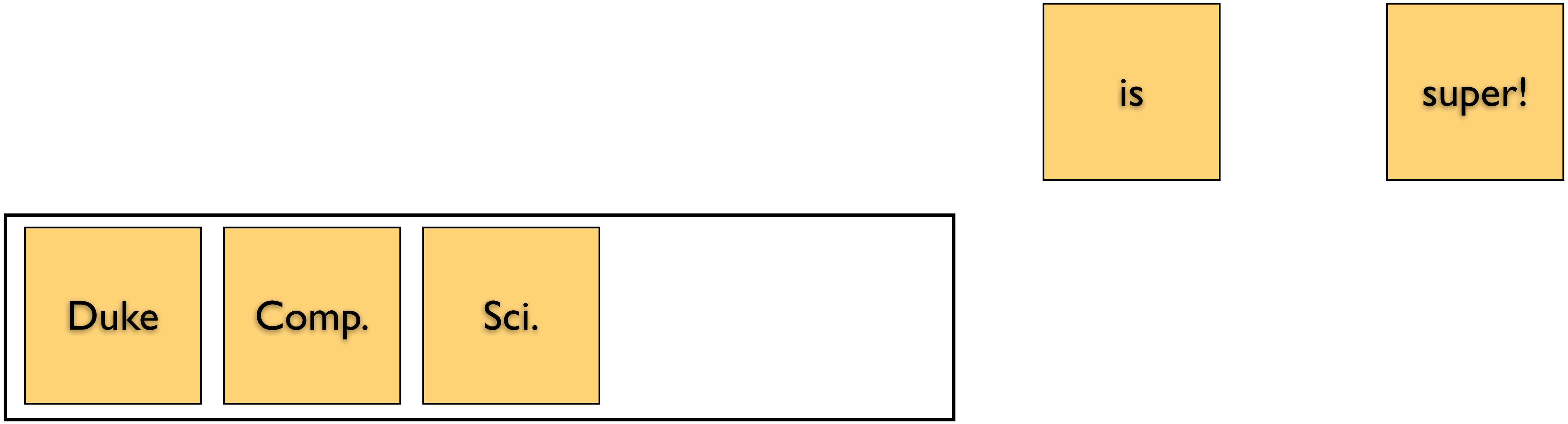
Comp.

Sci.

Operations: 4

Adds: 3

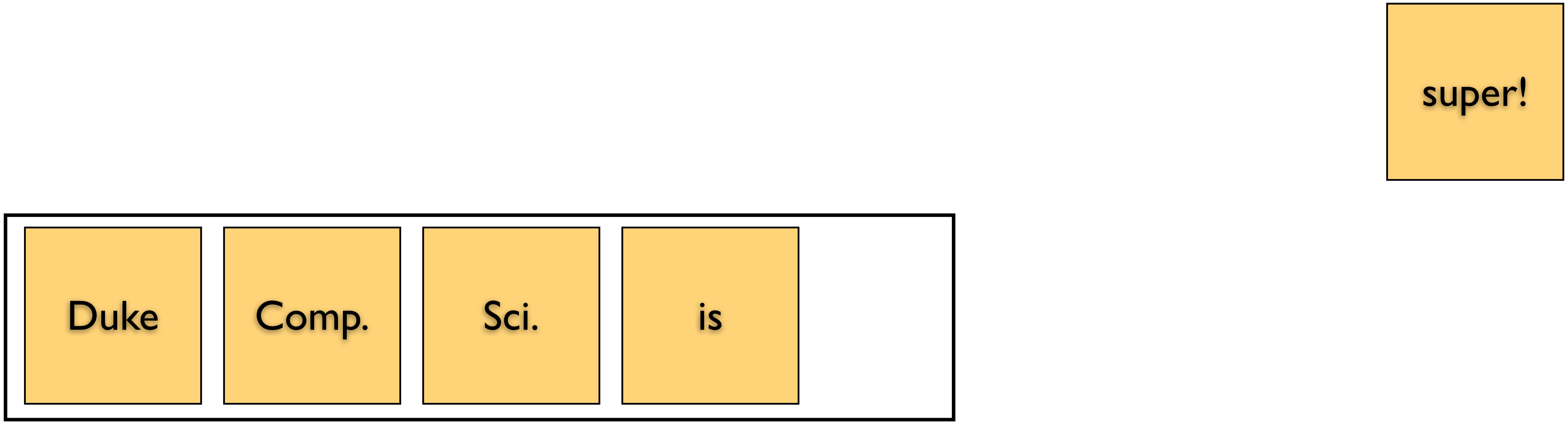
Add more than one?



Operations: 7

Adds: 3

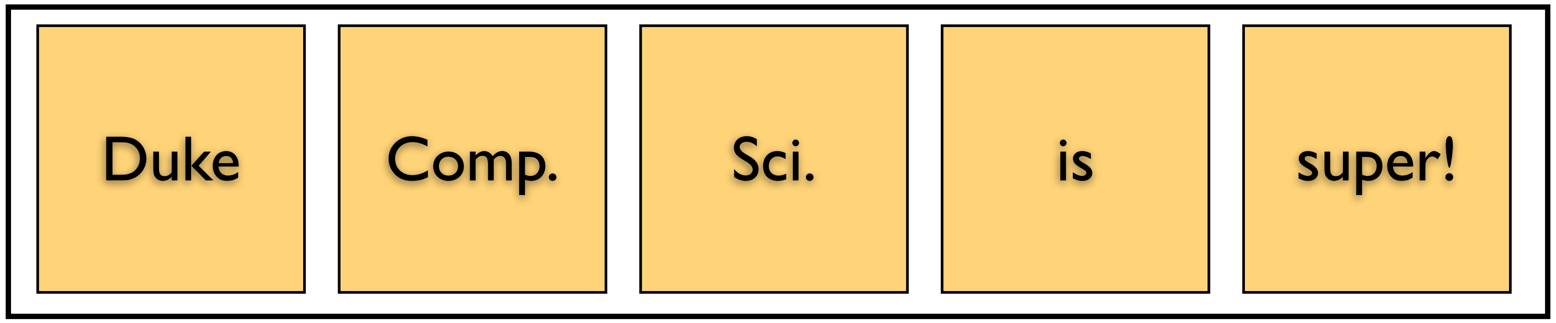
Add more than one?



Operations: 8

Adds: 4

Add more than one?

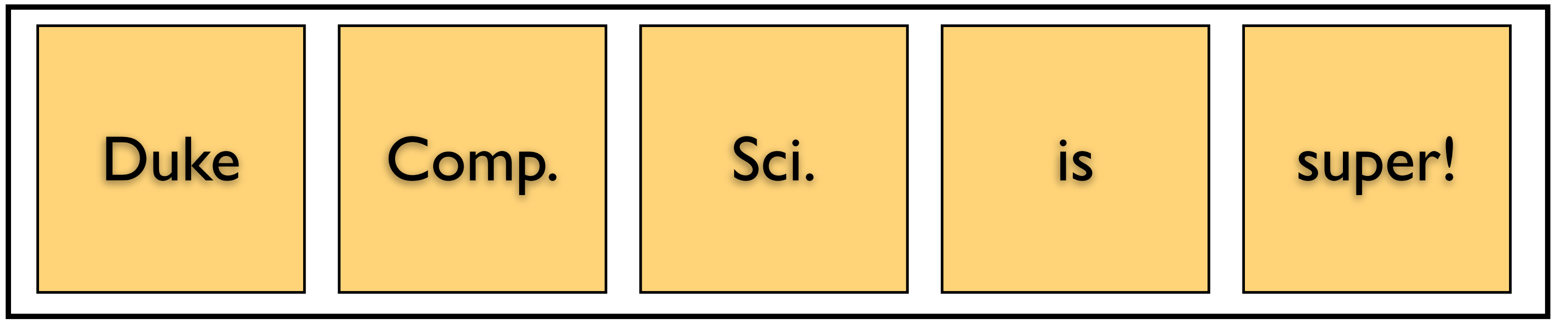


Duke	Comp.	Sci.	is	super!

Operations: 9

Adds: 5

Add more than one?



Duke	Comp.	Sci.	is	super!

Operations: 9

Adds: 5

http://goo.gl/eLp8l

Adding *n* elements, expanding by *k*

O(1) most of the time

O(n) sometimes

$$\sum_{i=1}^{n} \left[1 + \frac{i}{k} \right]$$

Adding *n* elements, expanding by *k*

O(1) most of the time

O(n) sometimes

$$\sum_{i=1}^{n}\left[1+\frac{i}{k}\right]=n+\sum_{i=1}^{n}\frac{i}{k}=n+\frac{1}{k}\sum_{i=1}^{n}i$$

Adding *n* elements, expanding by *k*

O(1) most of the time

O(n) sometimes

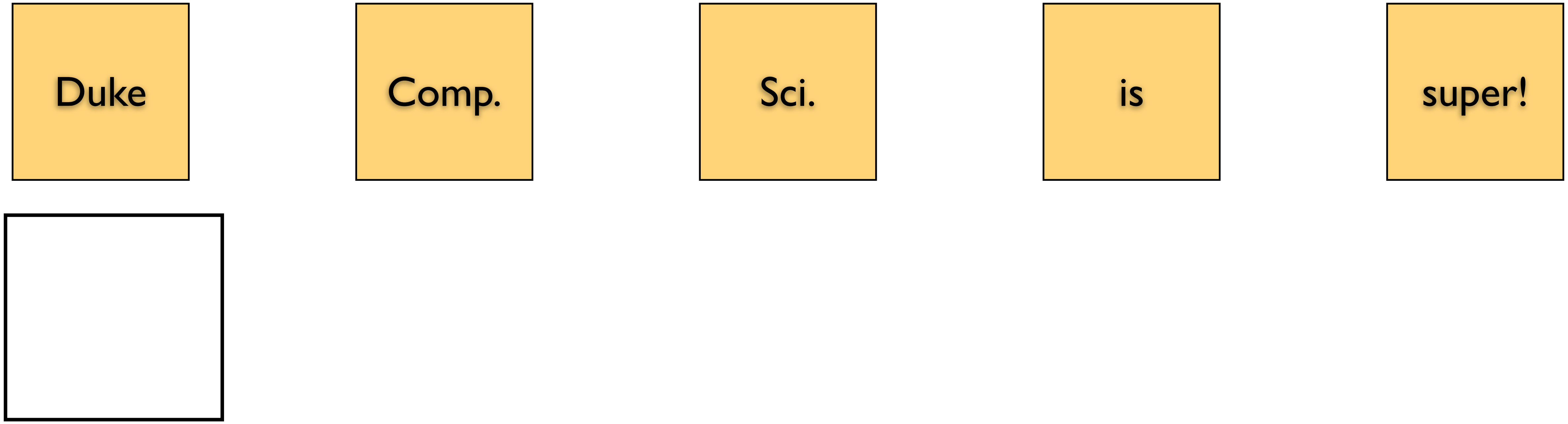
Note the worst-case behavior still dominates!

$$\sum_{i=1}^{n}\left[1+\frac{i}{k}\right]=n+\sum_{i=1}^{n}\frac{i}{k}=n+\frac{1}{k}\sum_{i=1}^{n}i$$

$$\in O(n^2)$$

Uh-oh...

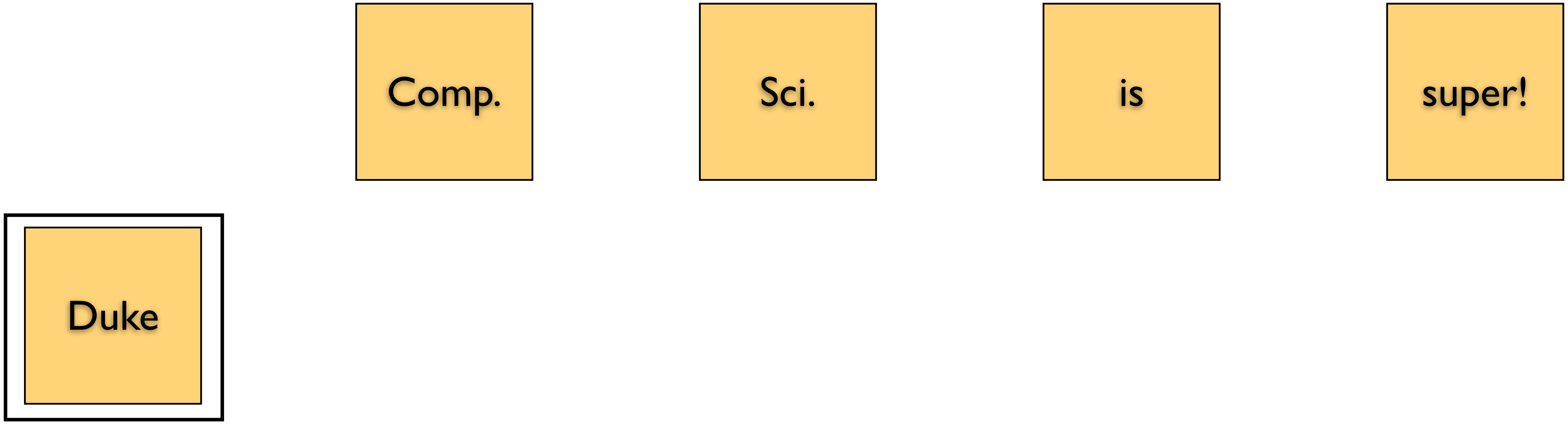
Adding didn’t work...



Operations: 0

Adds: 0

Adding didn't work...



Comp.

Sci.

is

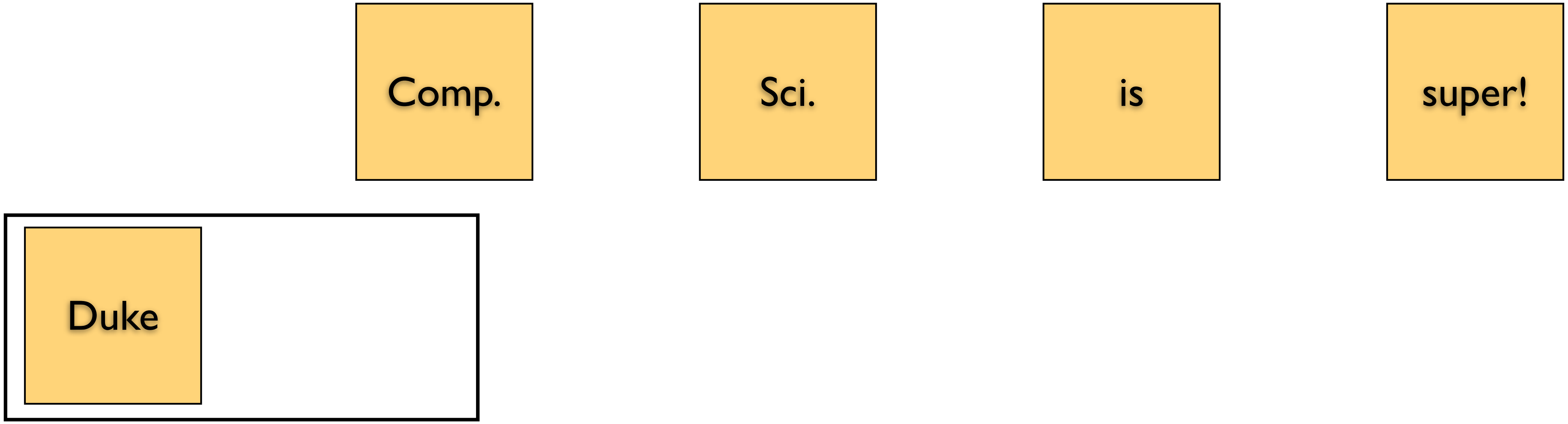
super!

Duke

Operations: 1

Adds: 1

Adding didn't work...



Comp.

Sci.

is

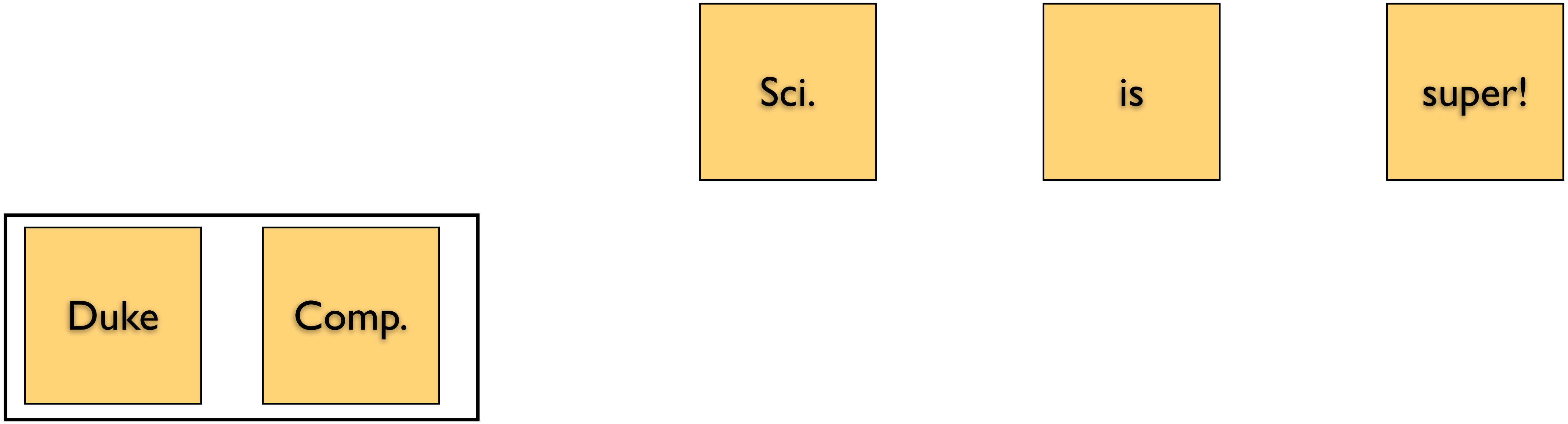
super!

Duke

Operations: 2

Adds: 1

Adding didn’t work...



Sci.

is

super!

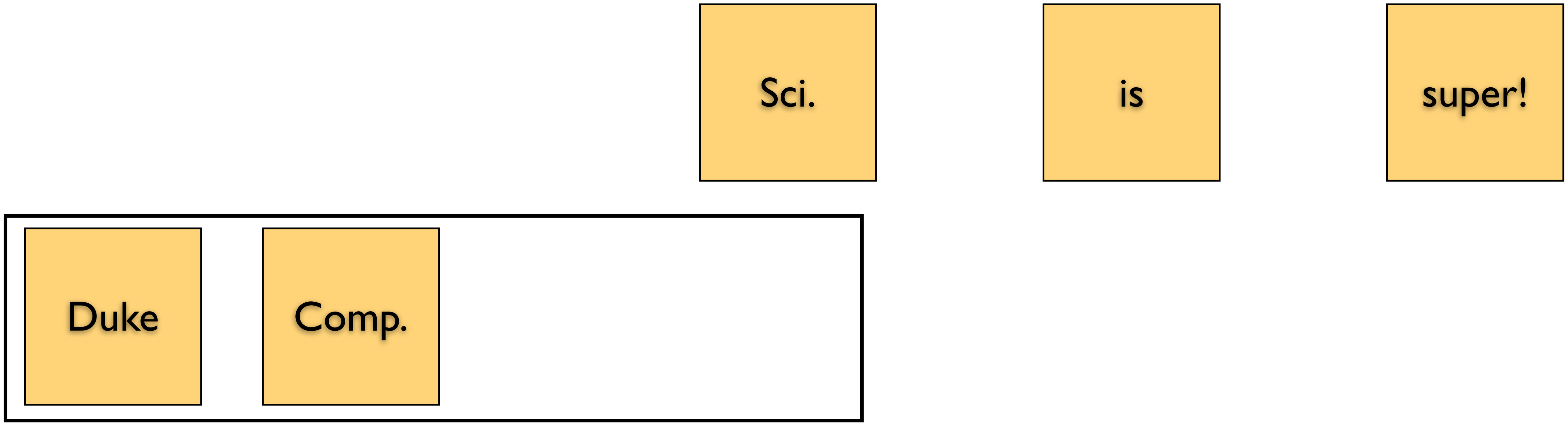
Duke

Comp.

Operations: 3

Adds: 2

Adding didn't work...



Sci.

is

super!

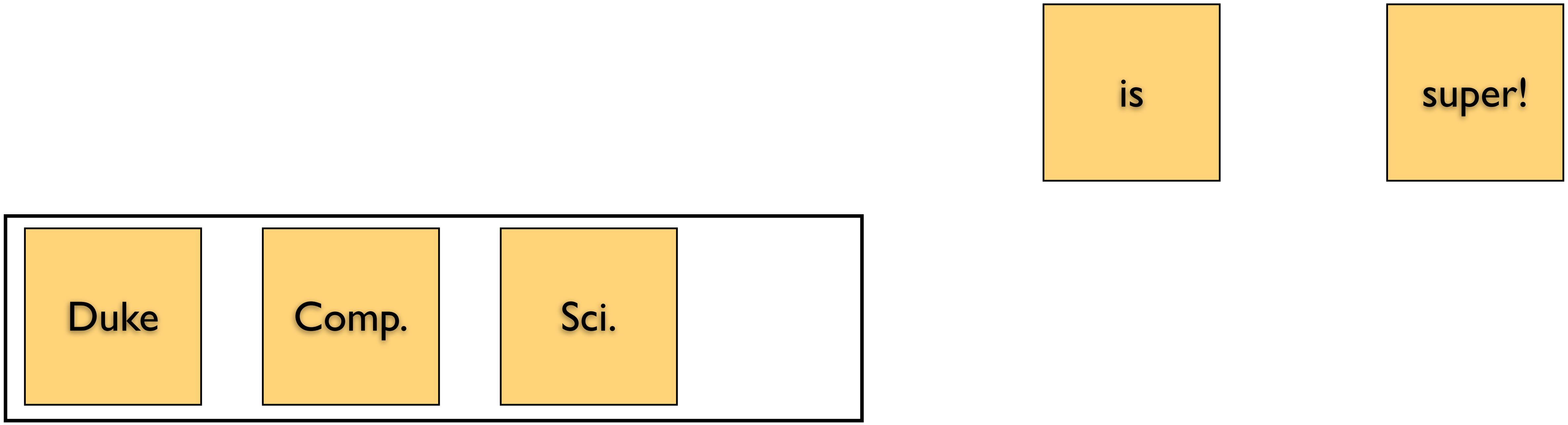
Duke

Comp.

Operations: 5

Adds: 2

Adding didn’t work...



is

super!

Duke

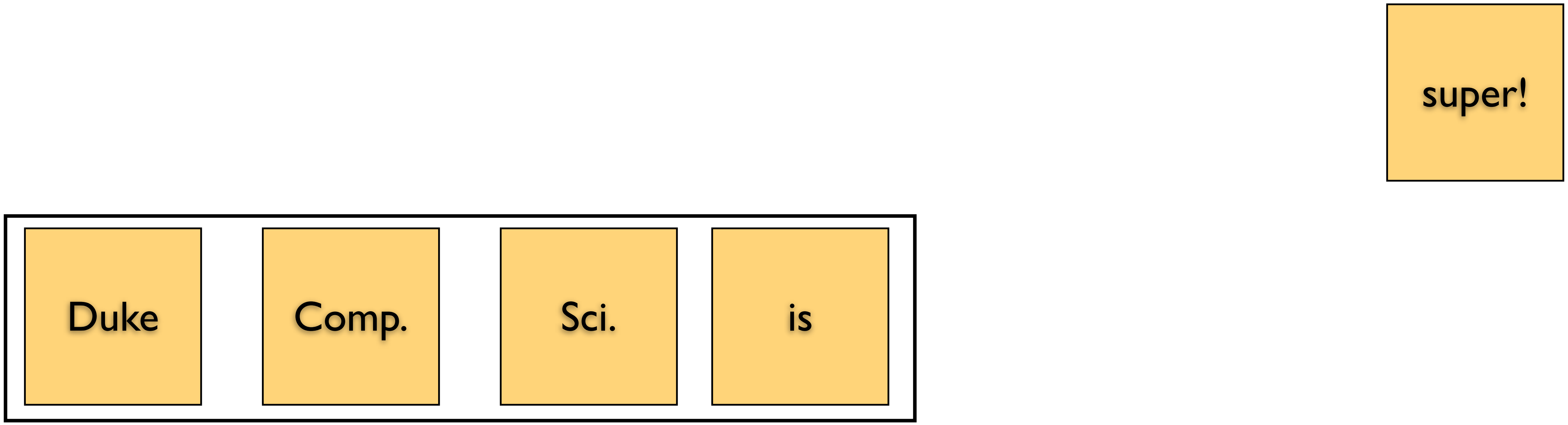
Comp.

Sci.

Operations: 6

Adds: 3

Adding didn’t work...



super!

Duke

Comp.

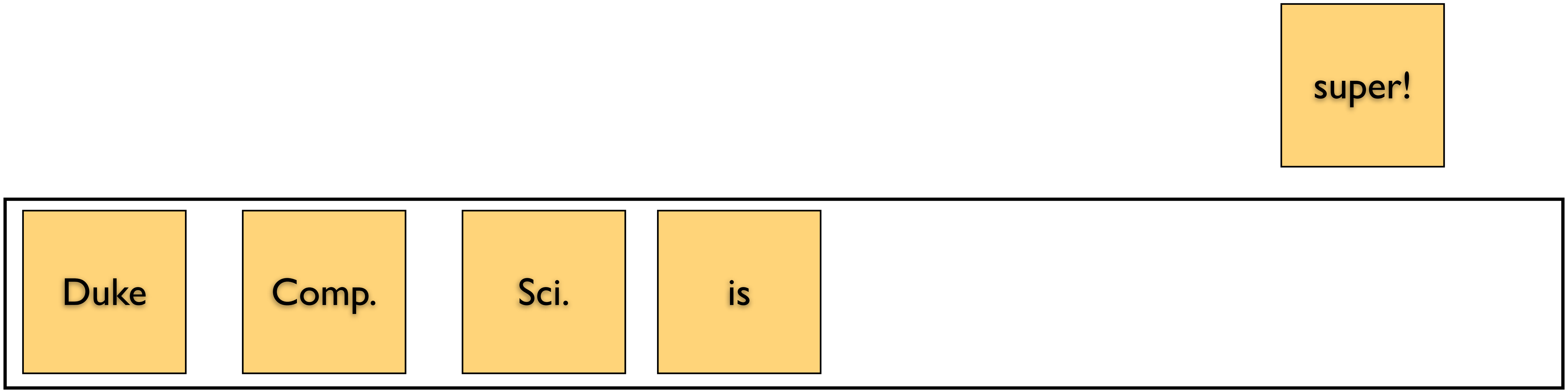
Sci.

is

Operations: 7

Adds: 4

Adding didn’t work...



super!

Duke

Comp.

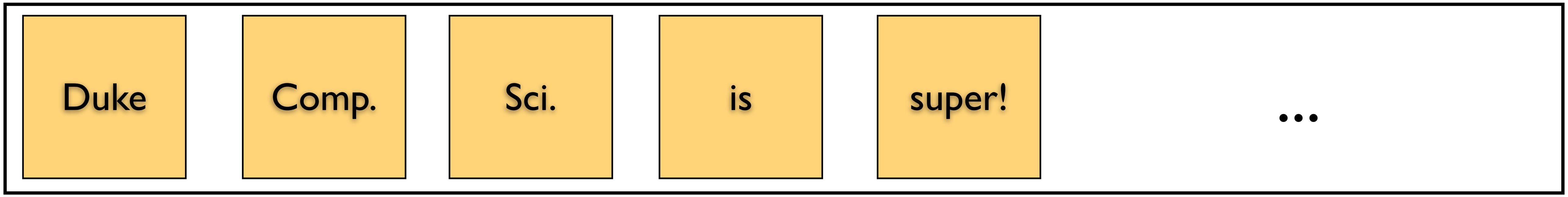
Sci.

is

Operations: 11

Adds: 4

Adding didn't work...

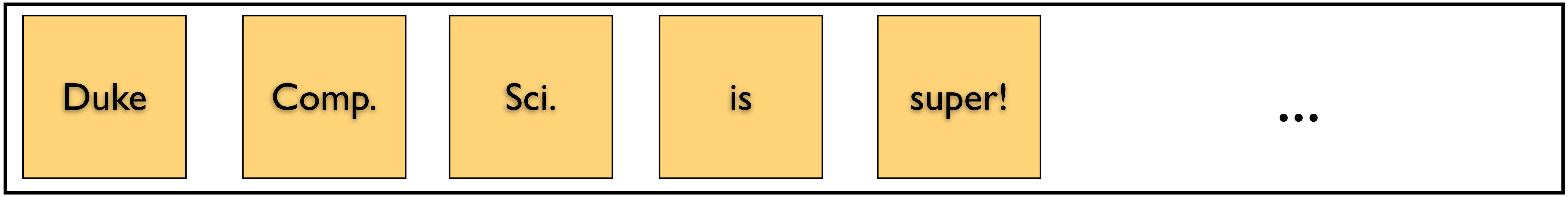


Duke | Comp. | Sci. | is | super! | ...

Operations: 12

Adds: 5

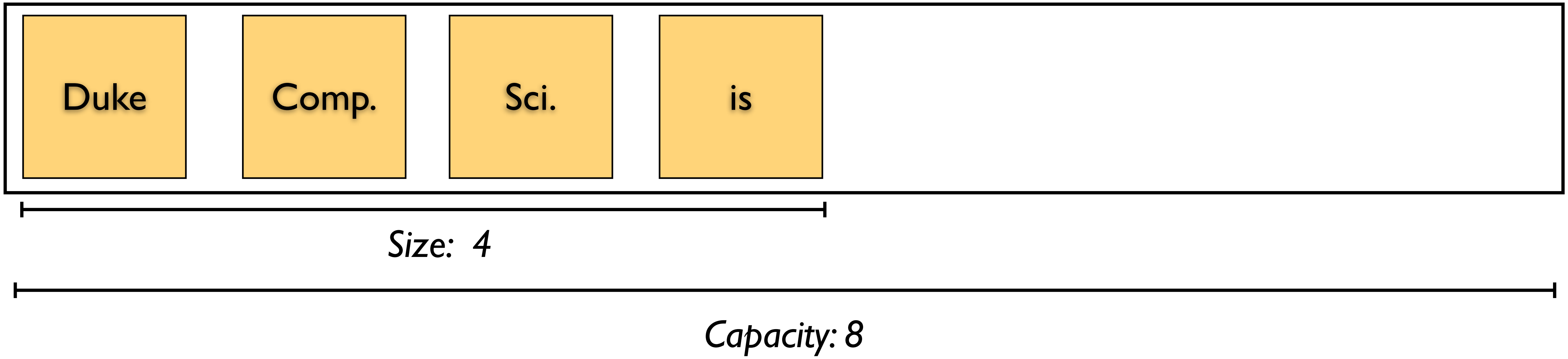
Adding didn’t work...



Duke Comp. Sci. is super! ...

How expensive is this strategy?

How expensive is this strategy?



Duke

Comp.

Sci.

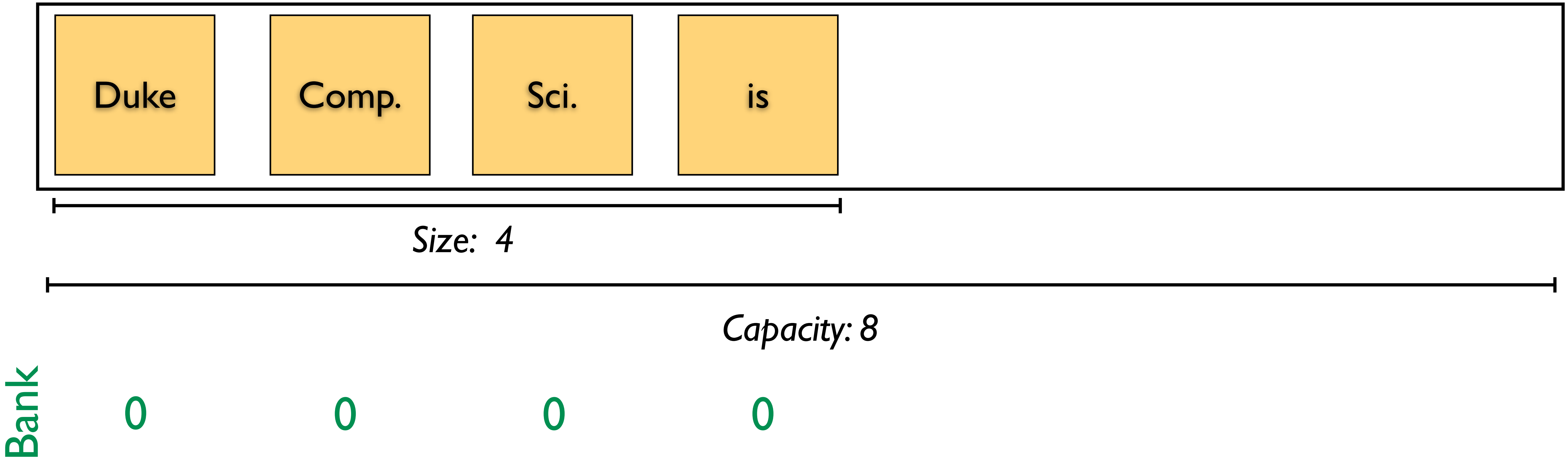
is

Size: 4

Capacity: 8

Suppose you’ve just doubled your array.

How expensive is this strategy?



Duke Comp. Sci. is

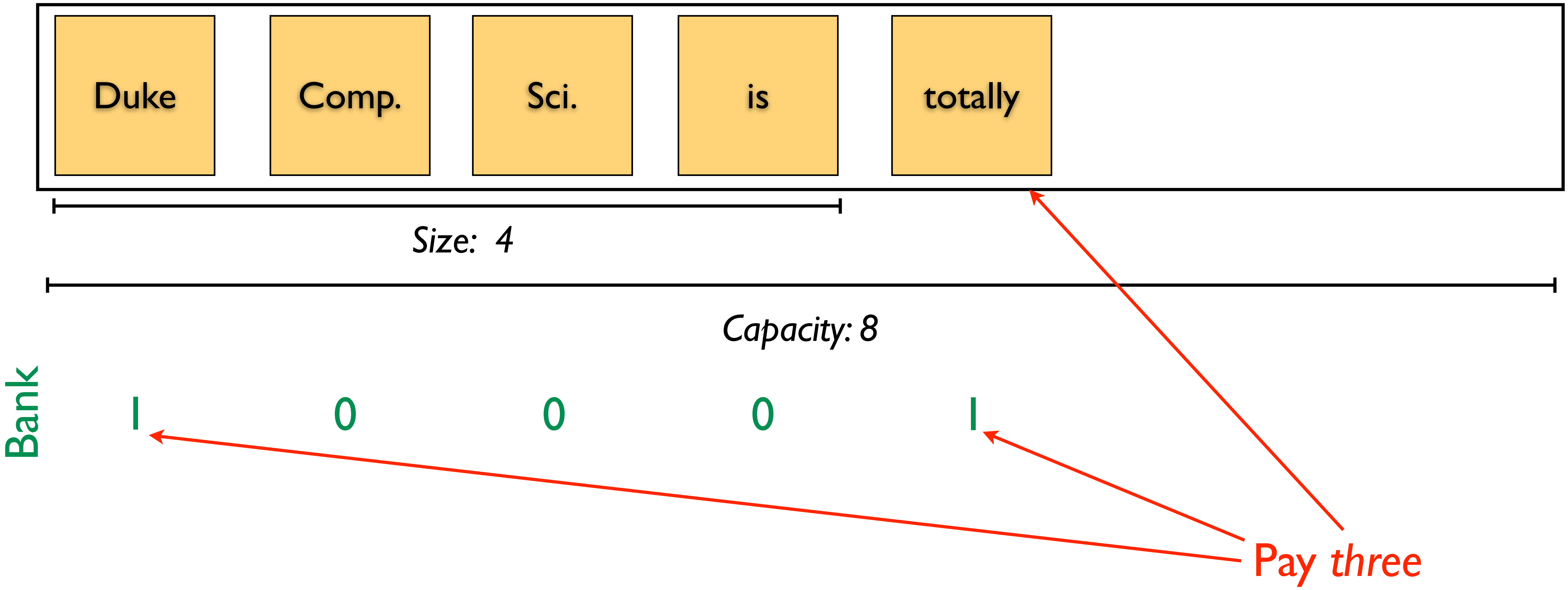
Size: 4

Capacity: 8

Bank

0 0 0 0

How expensive is this strategy?



Duke

Comp.

Sci.

is

totally

Size: 4

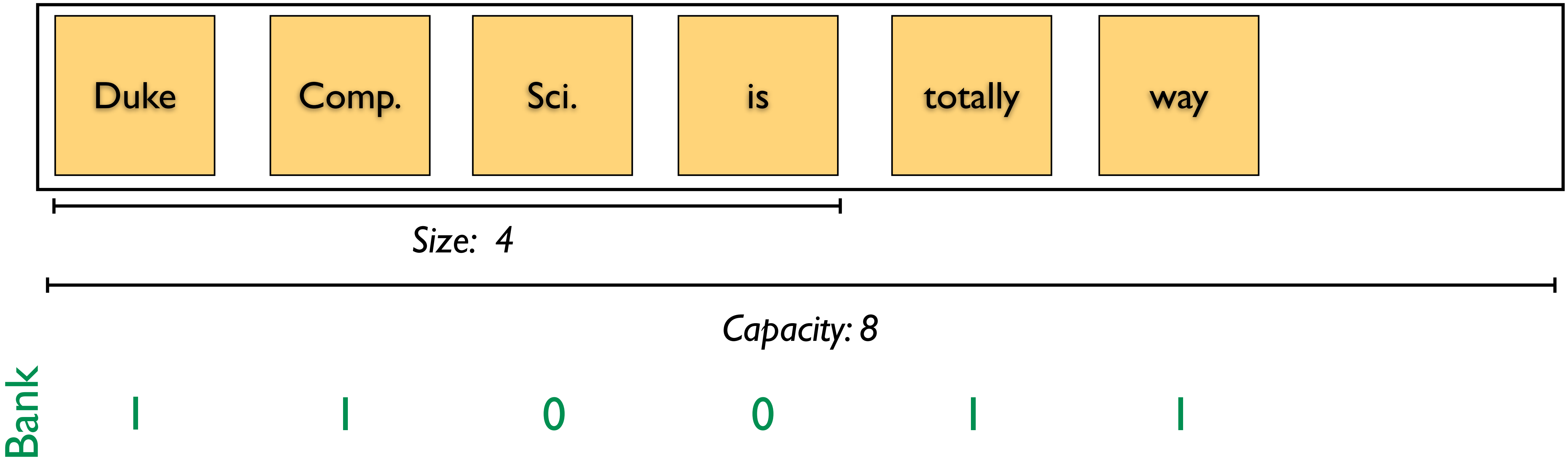
Capacity: 8

Bank

1 0 0 0 1

Pay *three*

How expensive is this strategy?



Duke Comp. Sci. is totally way

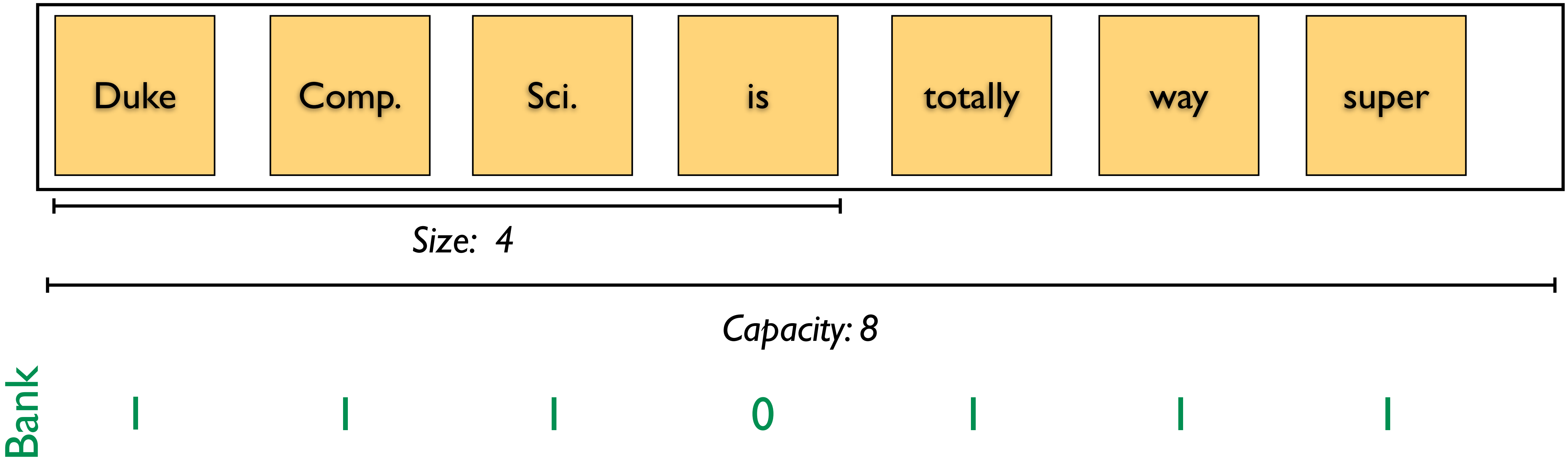
Size: 4

Capacity: 8

Bank

1 1 0 0 1 1

How expensive is this strategy?



Duke Comp. Sci. is totally way super

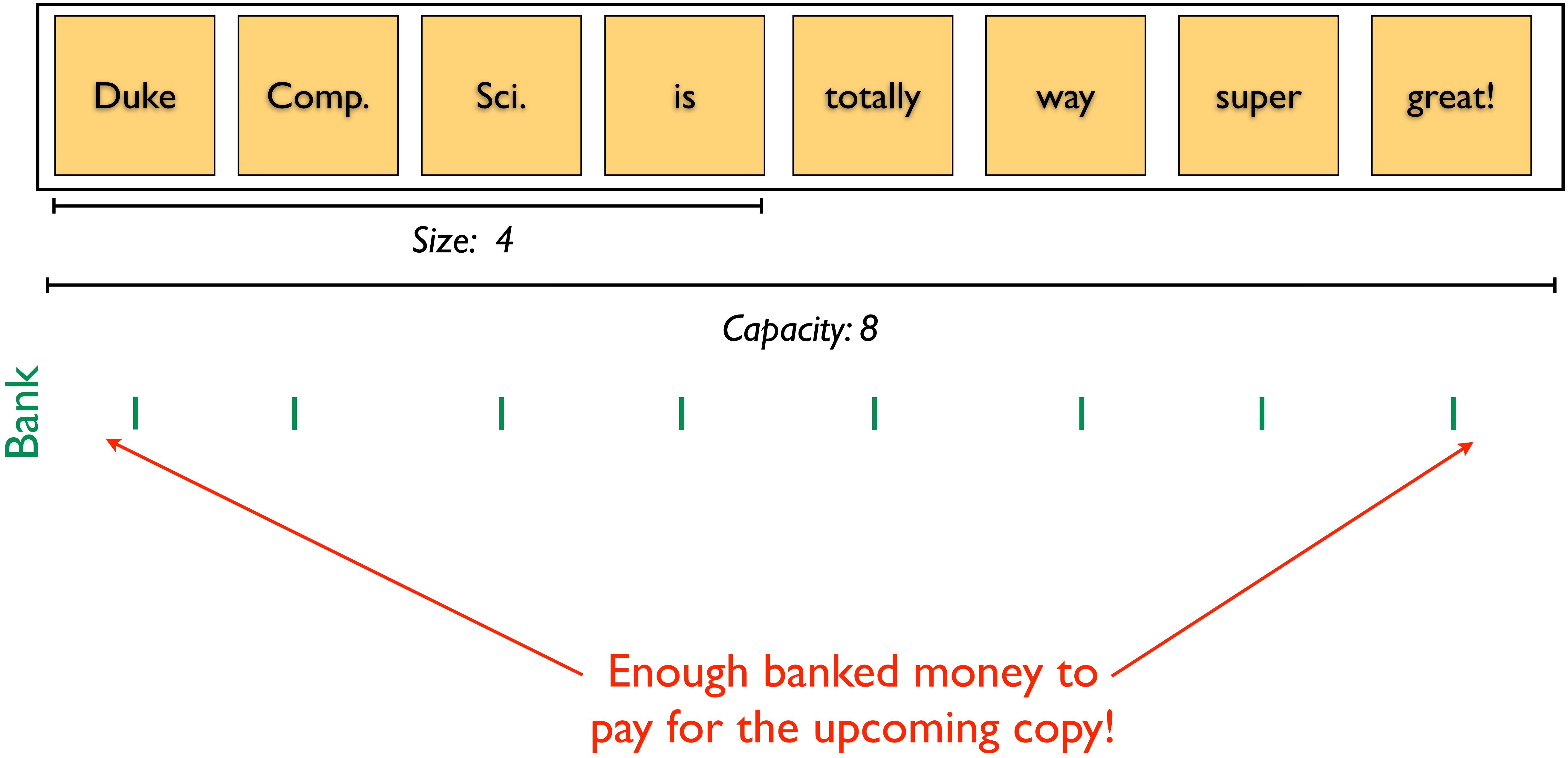
Size: 4

Capacity: 8

Bank

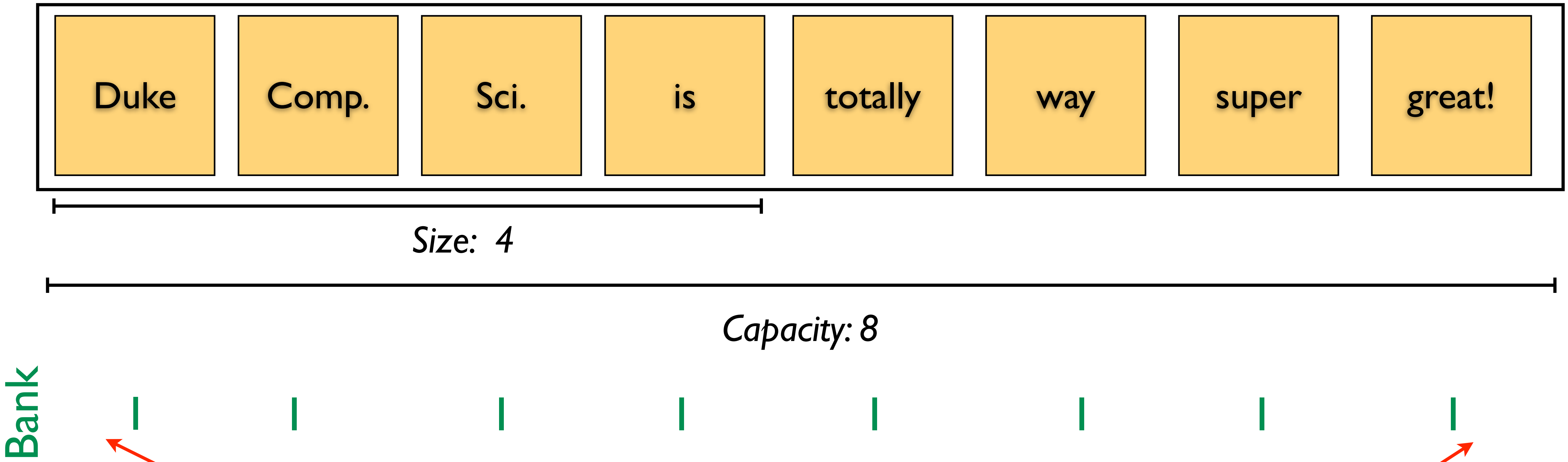
1 1 1 0 1 1 1

How expensive is this strategy?



Enough banked money to pay for the upcoming copy!

How expensive is this strategy?



Enough banked money to pay for the upcoming copy!

Key fact: add() costs a constant *amount of money!*

add has *constant amortized cost.*

Amortization facts

Amortized analysis deals with the cost of *n* operations, not the cost of one operation.

“N calls to add cost *O*(*n*) total.”

“One call to add might be *O*(*n*), too.”

Almost always good enough. So-called “realtime” applications are the exception.

The bank isn’t part of the data structure (no data is stored). It’s just an analytical tool.

A stack in two queues!

A stack in two queues!

Snarf Oct22InClass

http://goo.gl/4o5oN