GRAB A BYTE

HASHING!

WHAT WE'VE COVERED SO FAR

Prior to Spring Break we covered: Linear and Binary Search. Bubble, Selection, Insertion, Merge and Quick Sort. Since Spring Break we have covered Breadth-First Search and Depth-First Search

WHAT WE ARE COVERING TODAY

Today we are covering Hashing!

WHAT IS HASHING?

Hashing is a technique that maps data to a specific location in memory.

It uses a hash function to turn input into a number

That number is used as an index in a hash table to store or find data very fast!

IMAGINE APARTMENTS

You have 100 residents in an apartment building.

Each resident has an apartment number and needs a mailbox. The mail clerk uses the resident's apartment number to calculate the mailbox number using a hash function

Instead of looking through every mailbox, the clerk jumps to the correct one!

HOW DOES IT WORK?

With hashing, you calculate the index where the item will be in the table and then can search the table by that key. It doesn't search like a normal algorithm.

It goes straight to the desired value!

HOW TO CALCULATE THE INDEX?

A simple way is to calculate the key % the size of the table. Even if the key is String, it will have an ASCII value, so it will always return the remainder.

This means it will always calculate an index within the bounds of the table.

HOW TO … INDEX?

A s… of the
tab… an ASCII
…nder.

This me…s … an …dex within the
…le.

COLLISIONS!!

Sometimes two different keys produce the same index!
This is called a collision

You can handle collisions by storing multiple values in the same spot using a list or finding the next open spot and placing it there.

Lets consider a group of apartments and their residents:

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar

Then lets consider their mailboxes:

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

First we would take the value of their key (Apt) and find the remainder (%) of it based on the size (15)

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

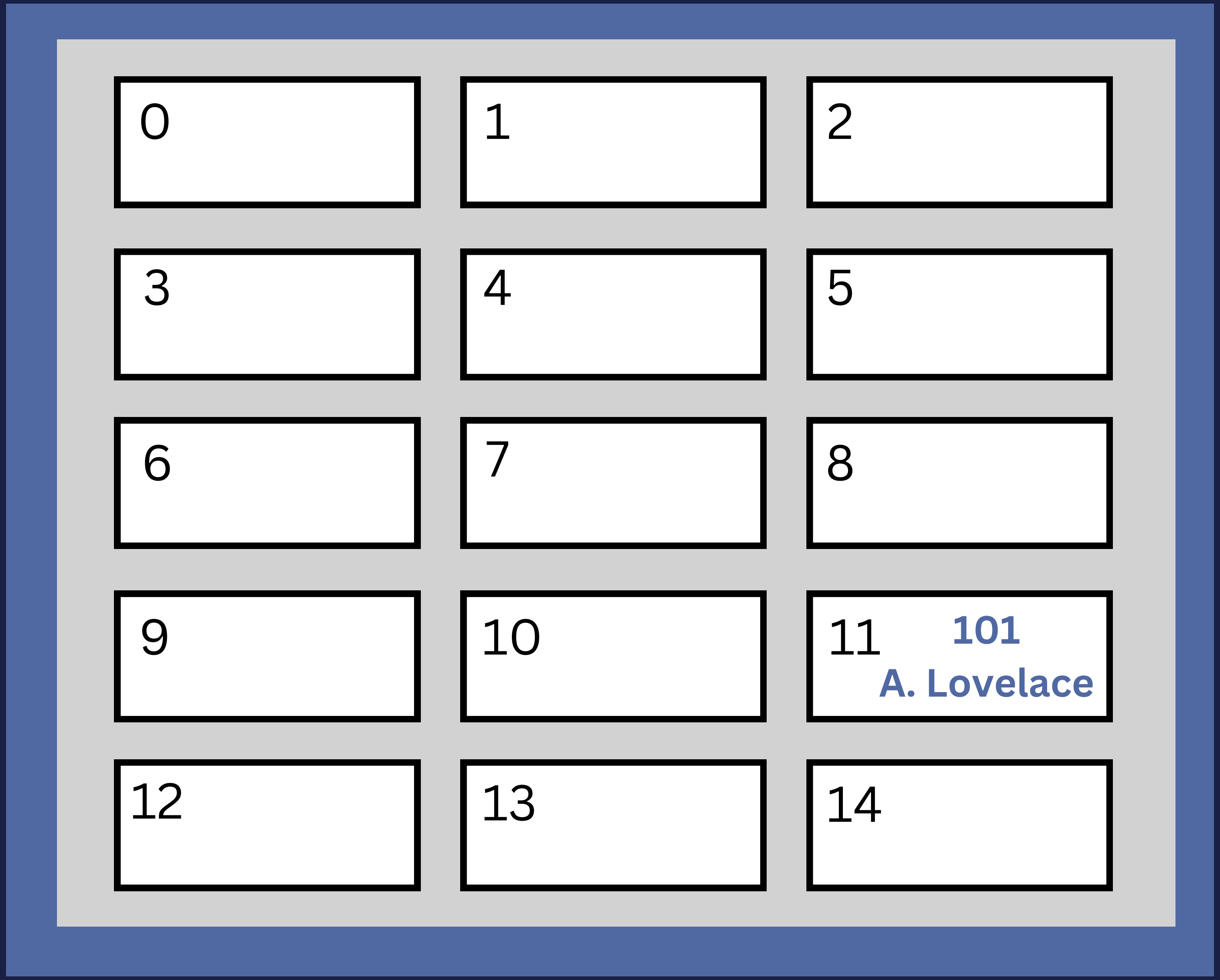
So Ada (101 % 15) would be assigned Mailbox 11

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

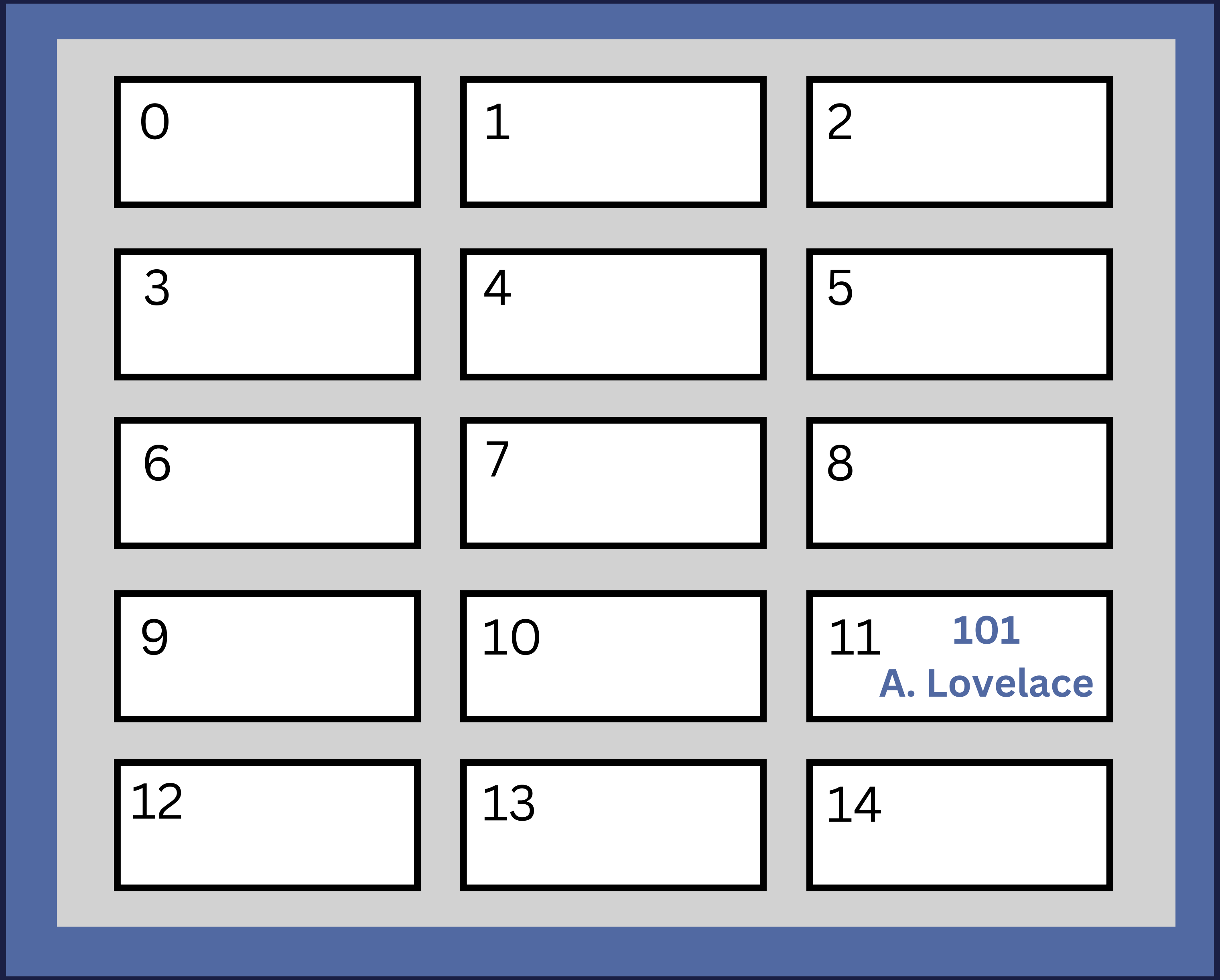
So Ada (101 % 15) would be assigned Mailbox 11

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar



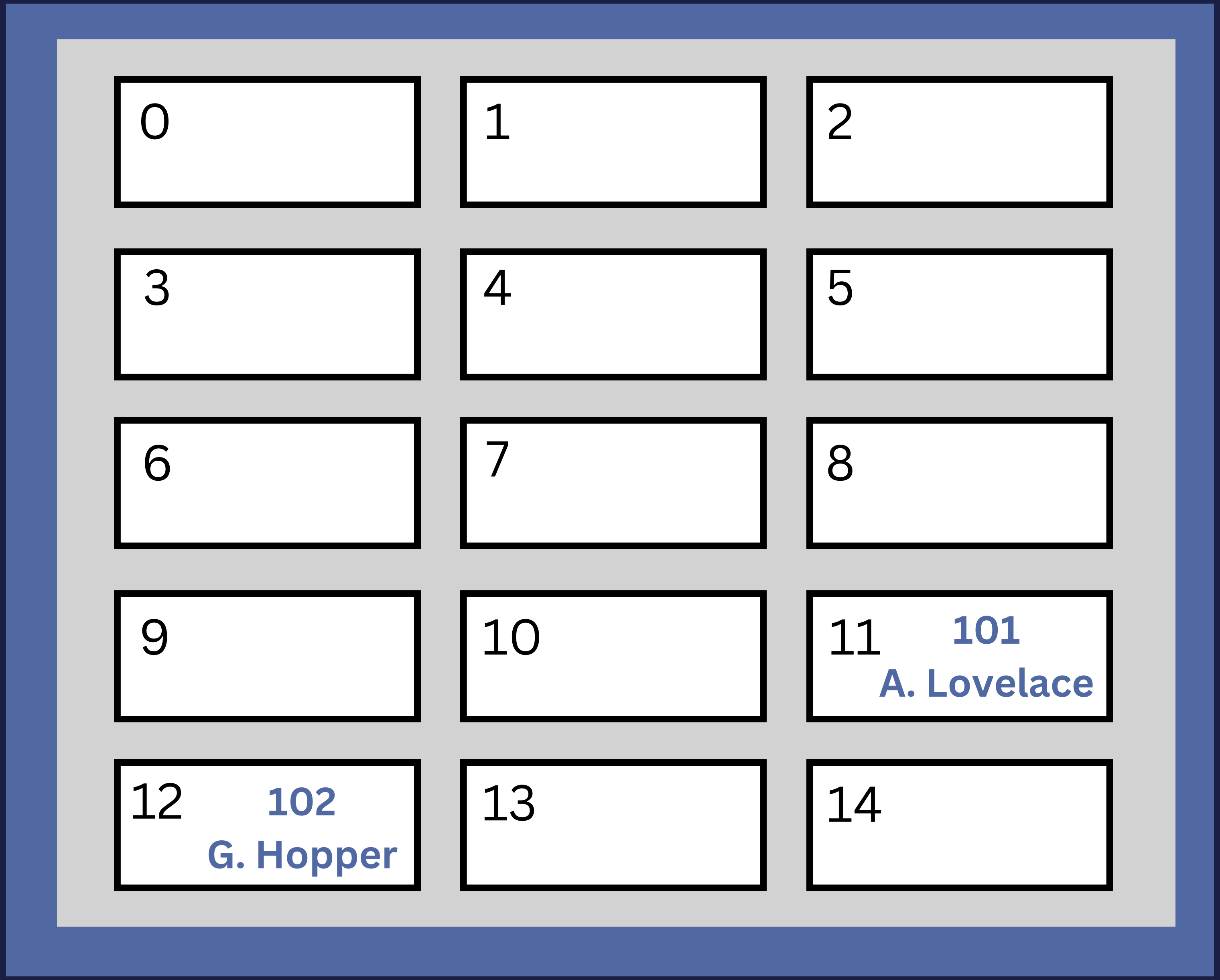
Grace (102 % 15) would be assigned mailbox 12

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar



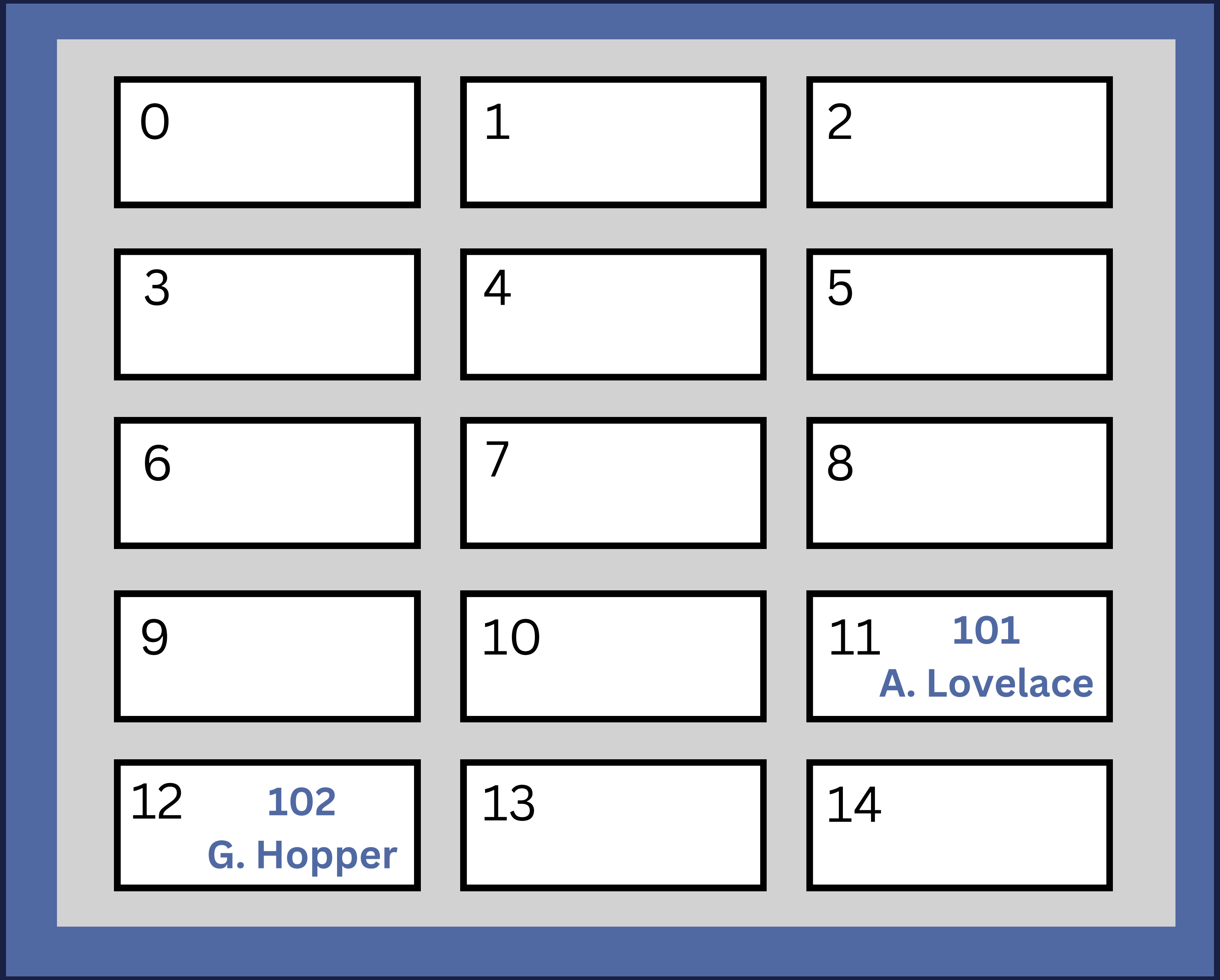
Grace (102 % 15) would be assigned mailbox 12

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar



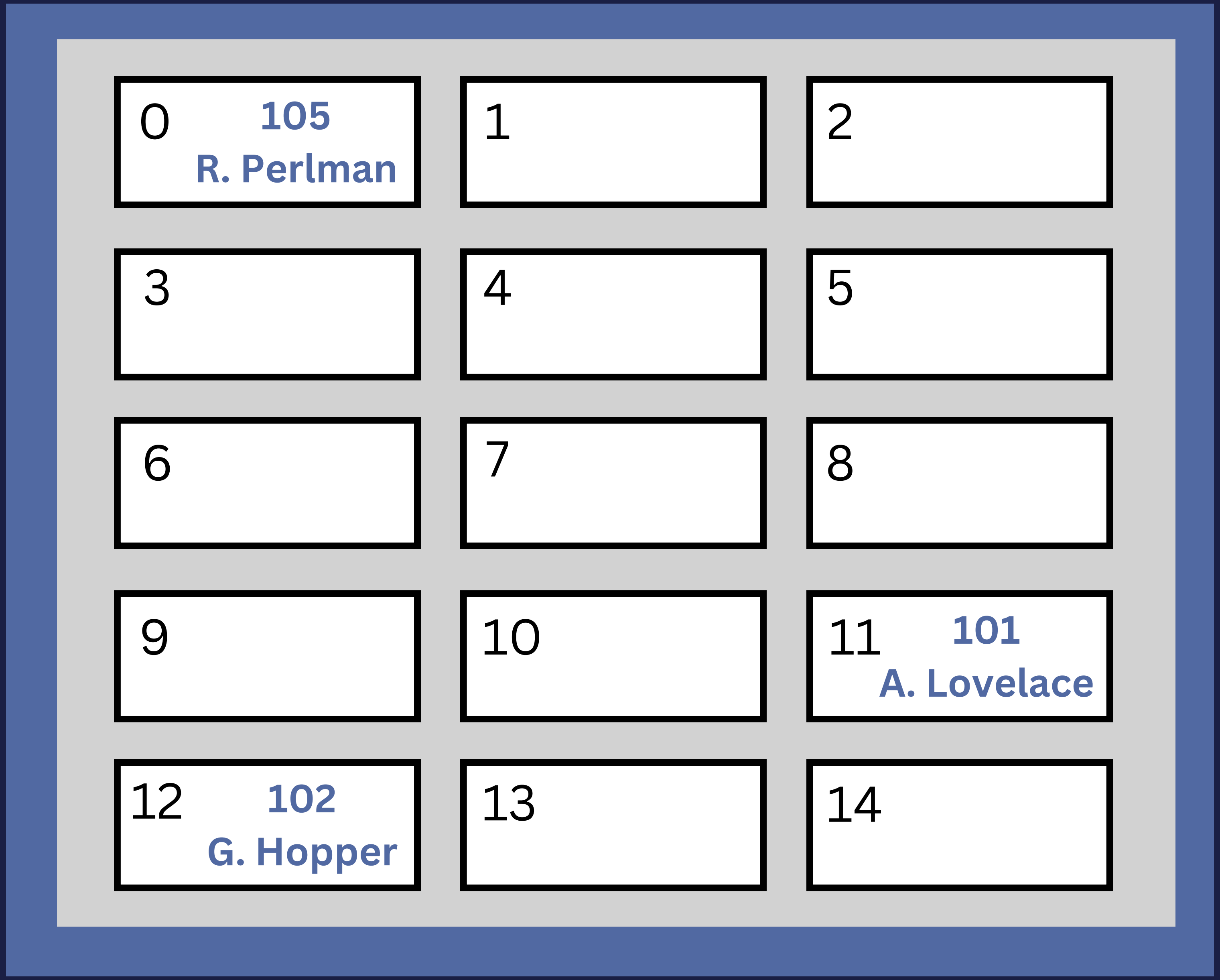
Radia (105 % 15) would be assigned mailbox 0

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar



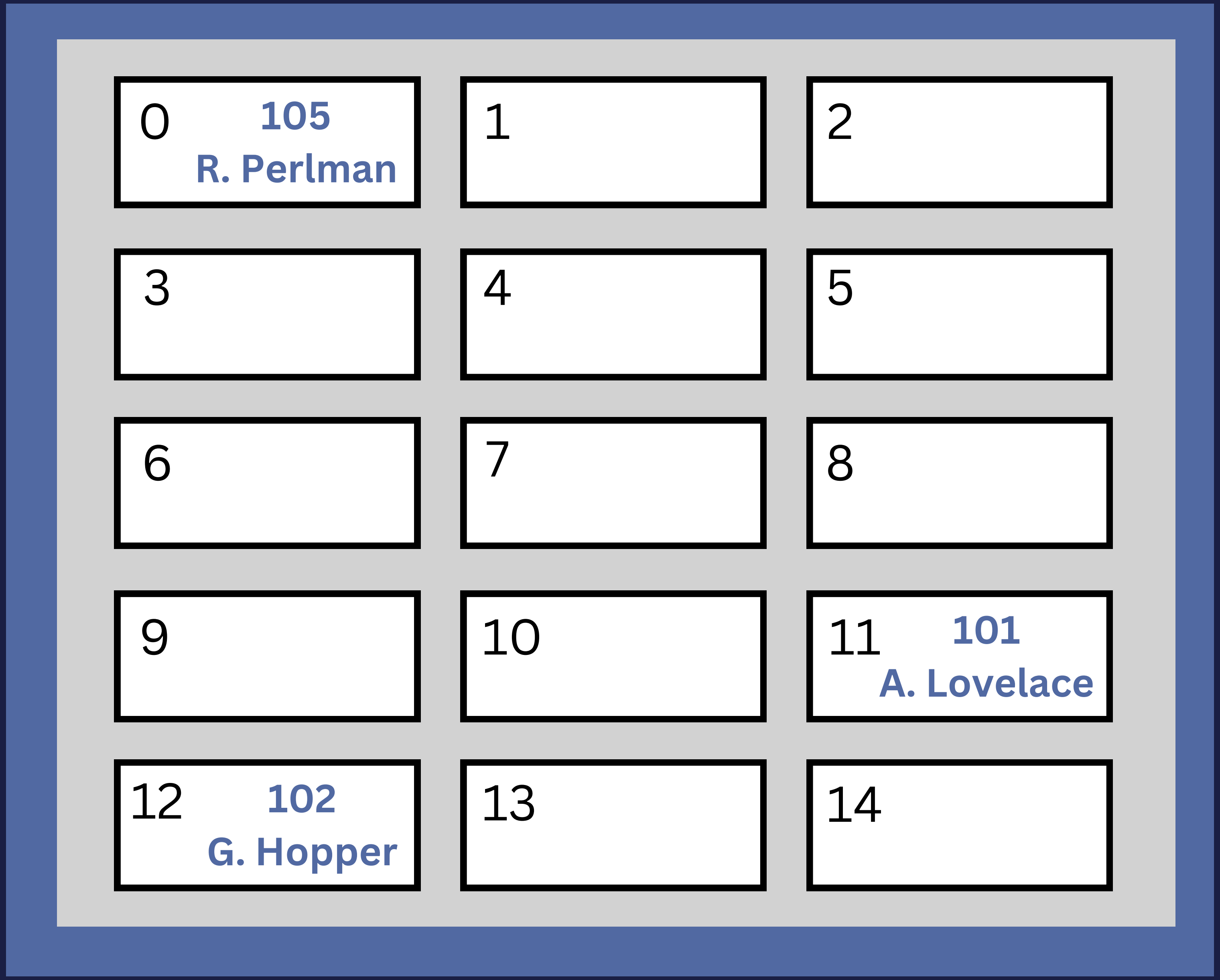
Radia (105 % 15) would be assigned mailbox 0

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar



Katherine (106 % 15) would be assigned mailbox 1

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar



Katherine (106 % 15) would be assigned mailbox 1

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar

0 105 R. Perlman	1 106 K. Johnson	2
3	4	5
6	7	8
9	10	11 101 A. Lovelace
12 102 G. Hopper	13	14

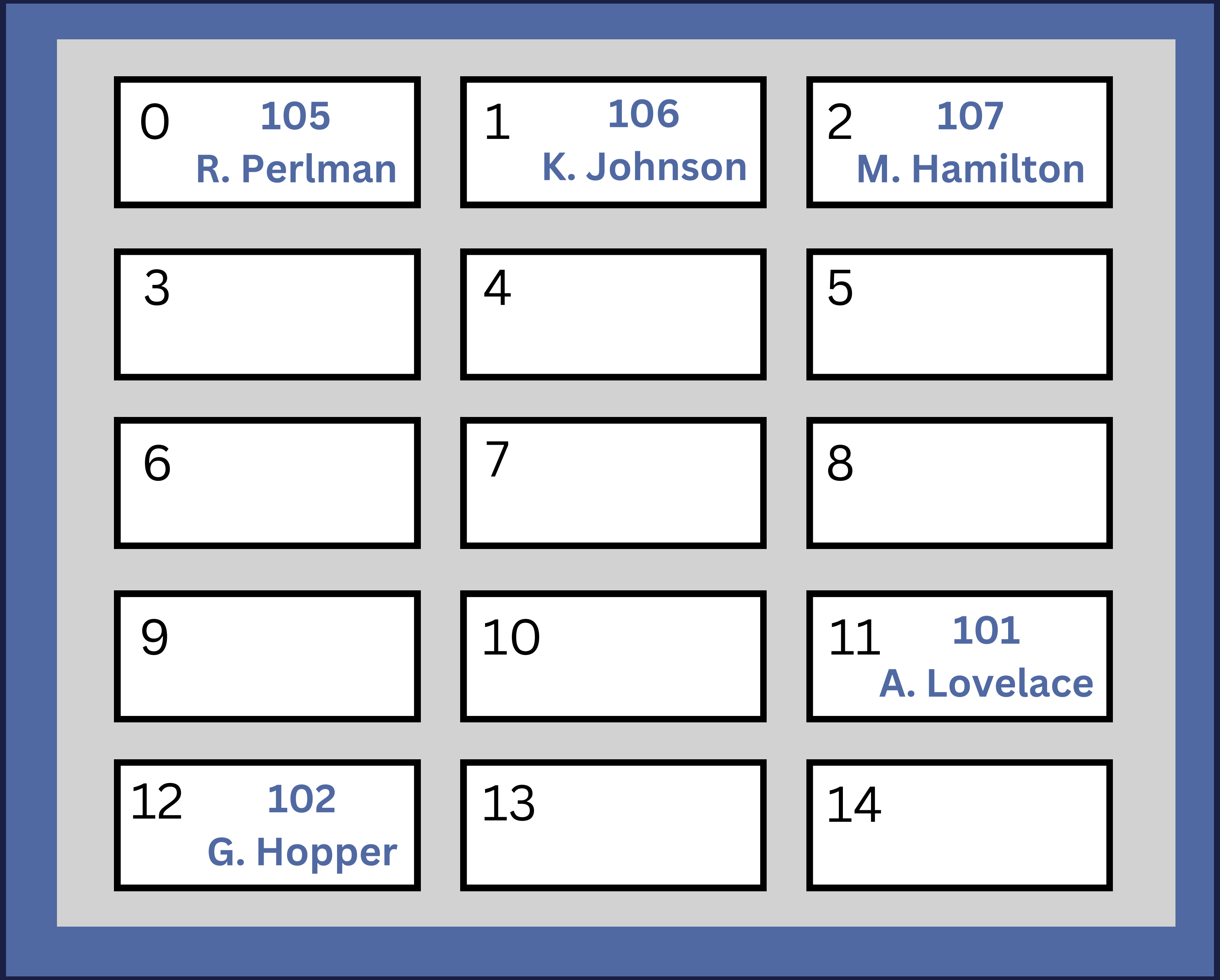
Margaret (107 % 15) would be assigned mailbox 2

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar

0 105 R. Perlman	1 106 K. Johnson	2 107 M. Hamilton
3	4	5
6	7	8
9	10	11 101 A. Lovelace
12 102 G. Hopper	13	14

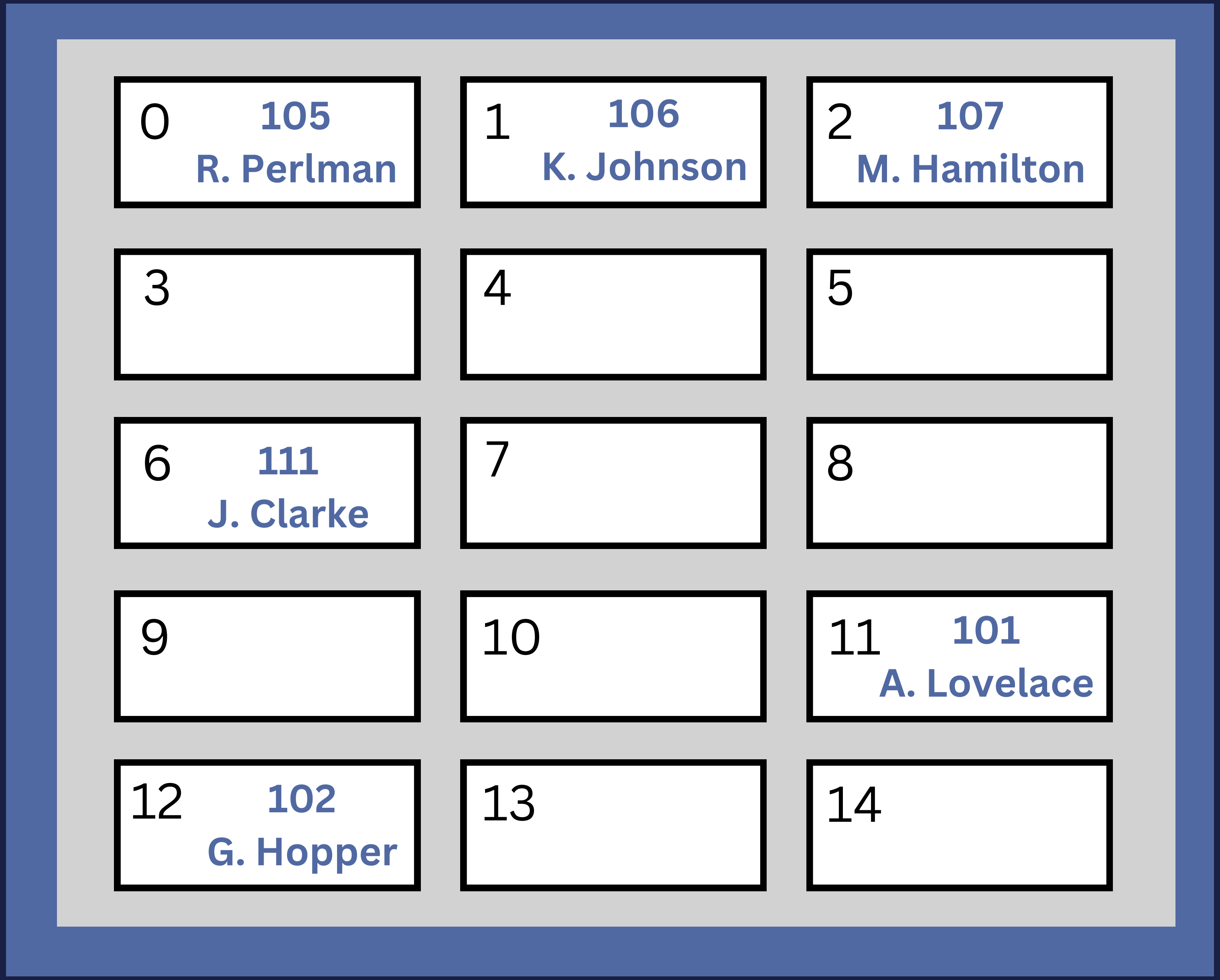
Joan (111 % 15) would be assigned mailbox 6

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar



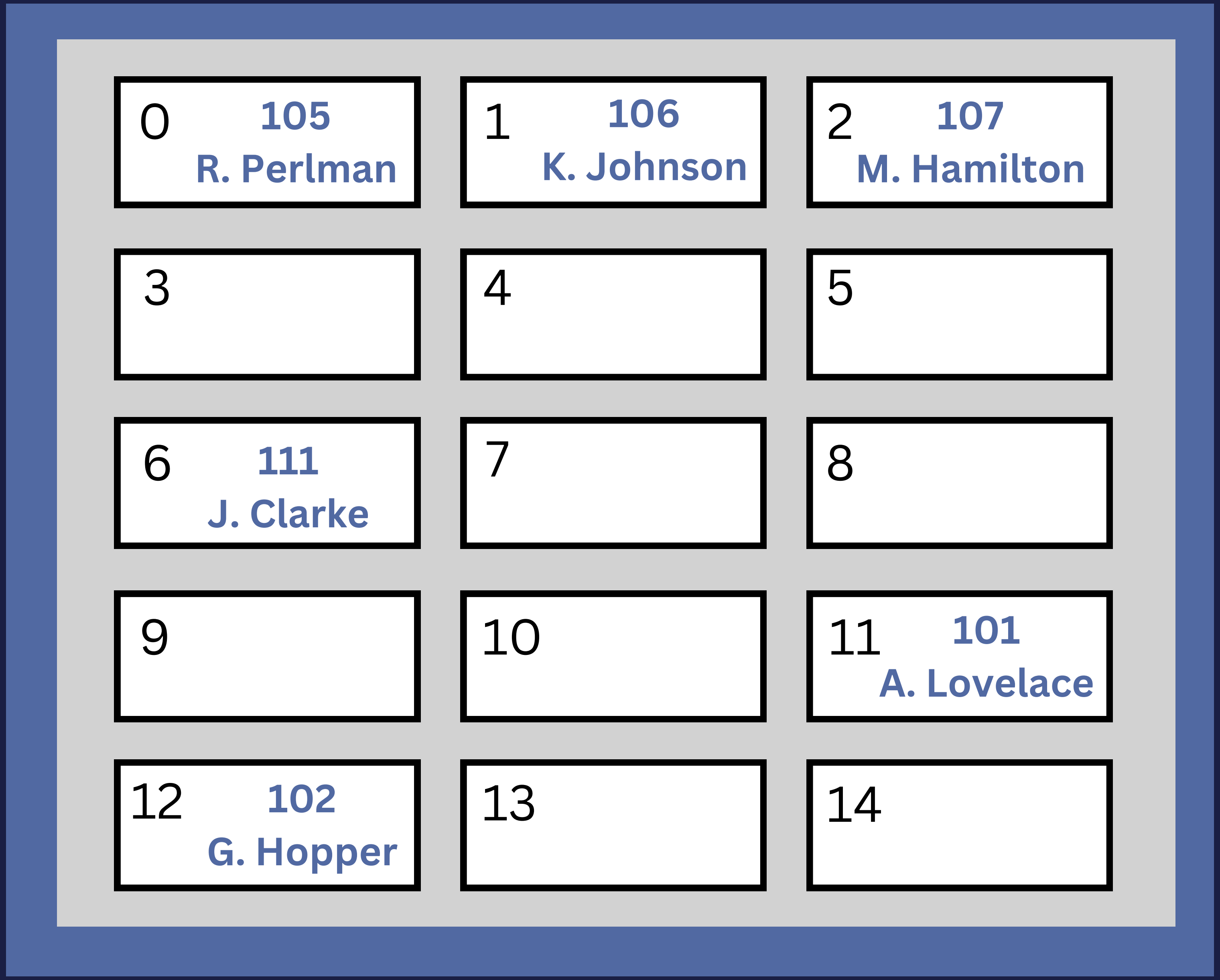
Joan (111 % 15) would be assigned mailbox 6

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar



And Hedy (113 % 15) would be assigned mailbox 8

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar



And Hedy (113 % 15) would be assigned mailbox 8

APT	NAME
101	Ada Lovelace
102	Grace Hopper
105	Radia Perlman
106	Katherine Johnson
107	Margaret Hamilton
111	Joan Clarke
113	Hedy Lamar

0 105 R. Perlman	1 106 K. Johnson	2 107 M. Hamilton
3	4	5
6 111 J. Clarke	7	8 113 H. Lamar
9	10	11 101 A. Lovelace
12 102 G. Hopper	13	14

Now that we have a finished mailbox (hashtable) we can access different mailboxes (keys, values)!

We can:

- Add
- Delete
- Update
- Find

0 105 R. Perlman	1 106 K. Johnson	2 107 M. Hamilton
3	4	5
6 111 J. Clarke	7	8 113 H. Lamar
9	10	11 101 A. Lovelace
12 102 G. Hopper	13	14

THE PSEUDOCODE

```
function hash_function(key, size):
    return hash(key) % size

table_size = size
key = “key”

hashed_value = hash_function(key, table_size)

print(key and hashed_value)
```

EXAMPLES REPLIT AND GITHUB! PLEASE GO TO:

HTTPS://REPLIT.COM/@RIKKIEHRHART/GRABABYTE

HTTPS://GITHUB.COM/RIKKITOMIKOEHRHART/GRABABYTE

O NOTATION!

What is O Notation?

- aka “Big O Notation” is a way to describe how efficient an algorithm is as the size of the input grows.
- It tells us how *long* an algorithm might take or how much *work* it might need to do
- Essentially, how many steps or iterations at the *worst*

Common O Notations:

- Linear Time | O(n) - covered with Linear Search
- Logarithmic Time | O(log n) - covered with Binary Search
- Quadratic Time | O(n^2) - covered with Bubble Sort
- Log-Linear Time | O(n log n) - covered with Merge Sort
- Constant Time | O(1) - covering today!

CONSTANT TIME - O(1)

Constant time means the algorithm takes the same amount of time

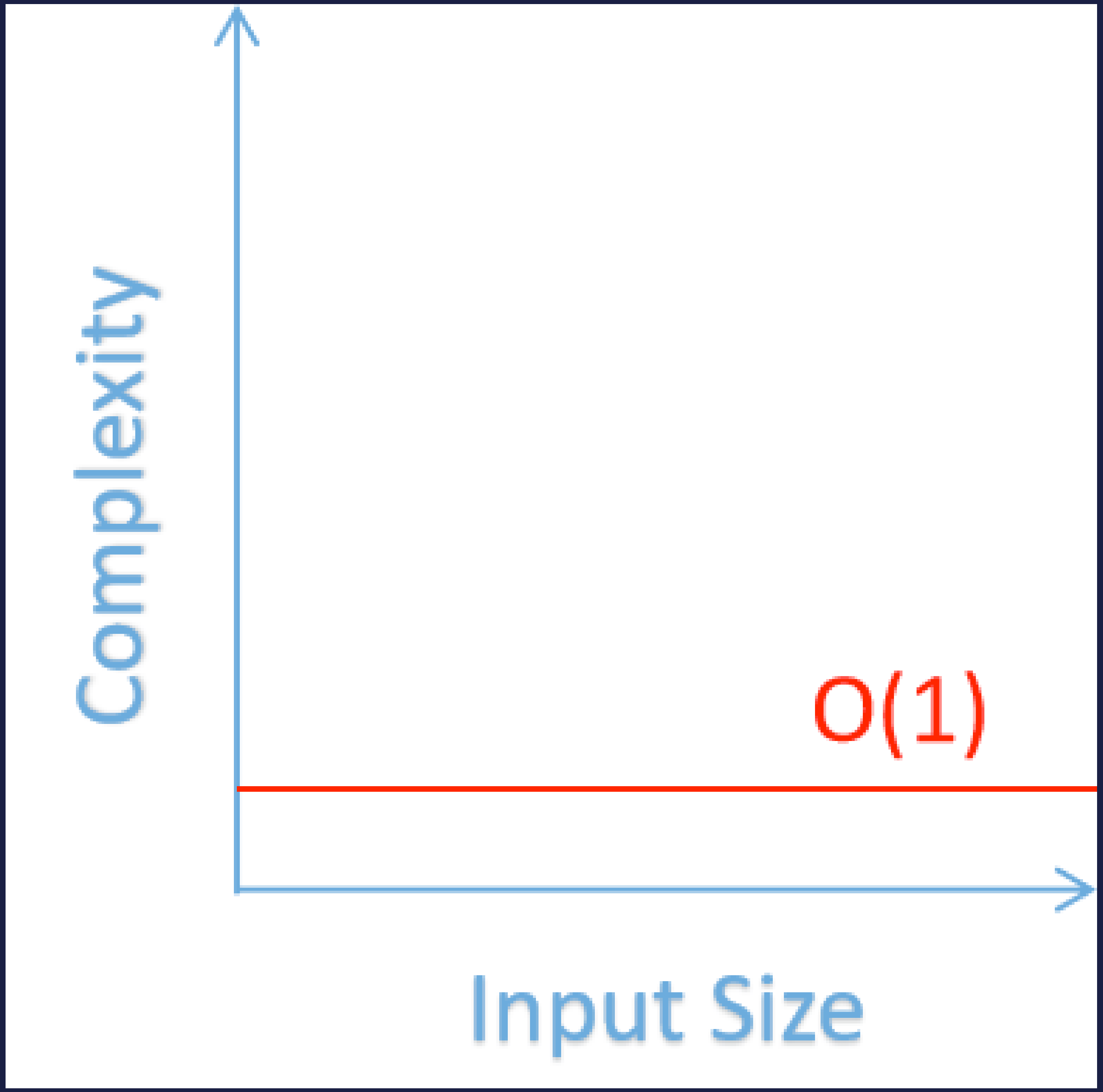
NO MATTER WHAT!

CONSTANT TIME - O(1)

- It doesn't loop through data
- It doesn't grow when the input size grows
- It doesn't matter if it is 5 or 5 million, its still just one step!

Its like driving down your street and going directly to your home. No searching for your house, you know where it is!

CONSTANT TIME - O(1)



UP NEXT

Apr 16 - Dijkstra's Algorithm
Apr 23 - Dynamic Programming (Knapsack Problem)
Apr 30 - Union-Find
May 7 - Kruskal's Algorithm
May 14 - Prim's Algorithm

Questions? - rikki.ehrhart@g.ausitncc.edu

If you'd like the opportunity to run a Grab a Byte algorithm workshop, please let me know!