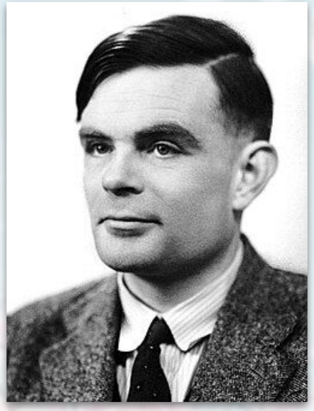
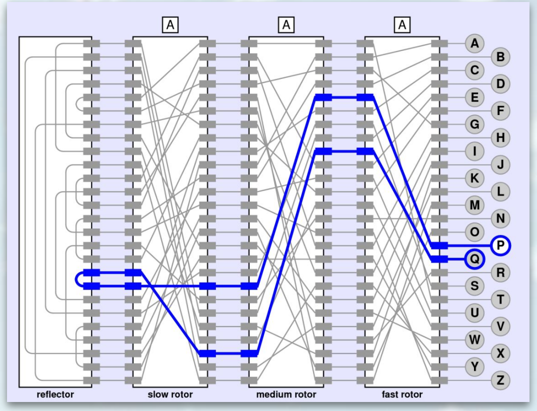
🎞️ Slides at cs106ax.stanford.edu/assignments.html





# YEAH Hours: Assign4

The Enigma Machine ⚙️

CS106AX Autumn 2025 🍁

cs106ax.stanford.edu

Stanford | ENGINEERING
Computer Science

## Welcome to Week 4/5 of 106AX!

🎃 🍁 🟩🟩🟩🟩🟩⬜⬜⬜⬜ ⛄ ⛷️

**Autumn** Week 4/5 **Winter**

*So … how's life y'all*

*Hope that midterms / midterm prep is going well!*

## 🗺️ The Map For Today

1. Welcome back to YEAH hours!
2. Recap of: **objects in JavaScript**
3. Tricky parts / milestones of Enigma

   *See Jerry's lecture for an overall assignment overview!*
4. Questions and office hours!

*Slides adapted from materials by Jonathan Kula, Ryan Eberhardt, CS106 staff*

## Lecture Recap!

⏳ ~5 minutes

Lecture 1 | Programming Paradigms (Stanford)

1.1M views 17 years ago

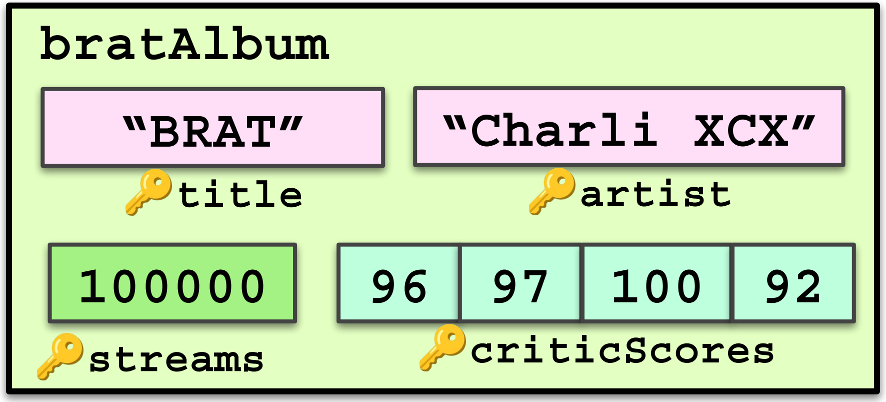
*Jerry Picture of the Week (Youtube Famous 💖 )*

#  Objects in JavaScript

✍ The easiest way to create new aggregates is through **JavaScript Object Notation (JSON).**

🔑 In **JSON,** you specify an object by **listing a sequence of name-value pairs**, enclosed in ***curly braces***. Names are unique; values are not necessarily unique, and can be of any data type.

```
let bratAlbum = {
 title: “BRAT”,
 artist: “Charli XCX”,
 streams: 100000,
 criticScores: [96,...,92]
};
```



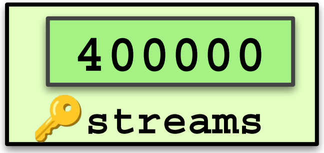
We can **select, add, or modify a key-value pair** of the object via an expression of the form:

🔑 `objectName.keyName` or 🔑 `objectName[keyName]`

```
bratAlbum.streams
```

100000

```
bratAlbum.streams *= 4;
```



# Objects in JavaScript → ✨ a **“vibes-based”** language – we’ll really see what means :)

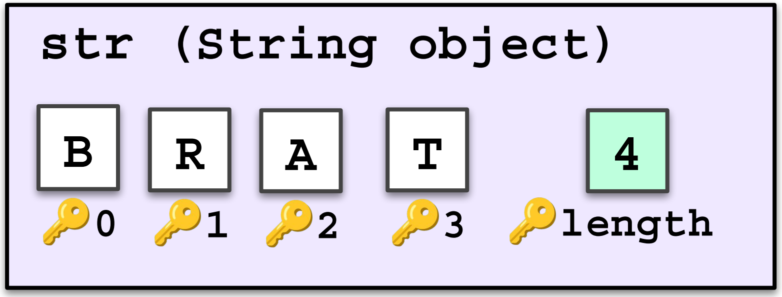
In JavaScript, **nearly everything** is or can be an **object (gallery of key-value pairs)!**

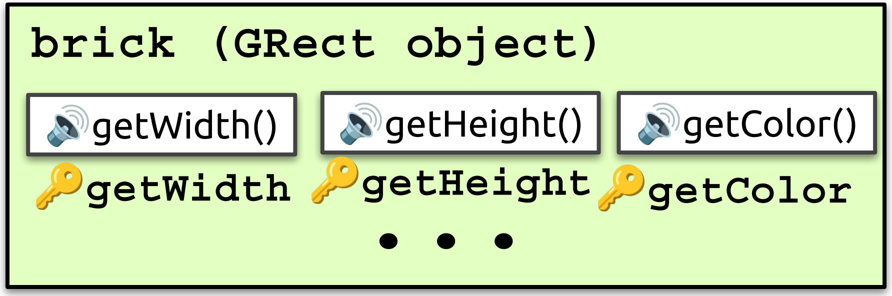
```
let str = “BRAT”;
console.log(str.length);
```
🧵

```
4
```

```
let brick = GRect(0, 0, 100, 100);
console.log(brick.getWidth())
```

```
100
```





🦋 JS temporarily wraps the string primitive into a **String object**, which **allows us to look up the length key** as we would do for an object. Vibes.

**GRect, GOval, etc.** are objects whose keys, e.g., **getWidth, getHeight,** can even have functions 🔊 as values. We can attach our own keys + functions to them, as we’ll see!

## Happy to discuss any questions!

🎮 *Next Up: <u>Trickier bits</u> of Enigma assignment!*

*May be a shorter YEAH "lecture" session than usual as Jerry has wonderfully covered most of the assignment, but extra time for OH!*

# 🎥 *Imitation Game* Movie Screening + OH

Hosted by our wonderful SLs 💖 Andy and Jenny! 💖

"BENEDICT CUMBERBATCH IS OUTSTANDING"
"A SUPERB THRILLER"
"THE BEST BRITISH FILM OF THE YEAR"
"AN INSTANT CLASSIC"
THE TRUE ENIGMA WAS THE MAN WHO CRACKED THE CODE
BENEDICT CUMBERBATCH
KEIRA KNIGHTLEY
THE IMITATION GAME
IN CINEMAS NOVEMBER 14

**ADMIT ONE TICKET**

**Wed, Oct 20th, 7-9PM**

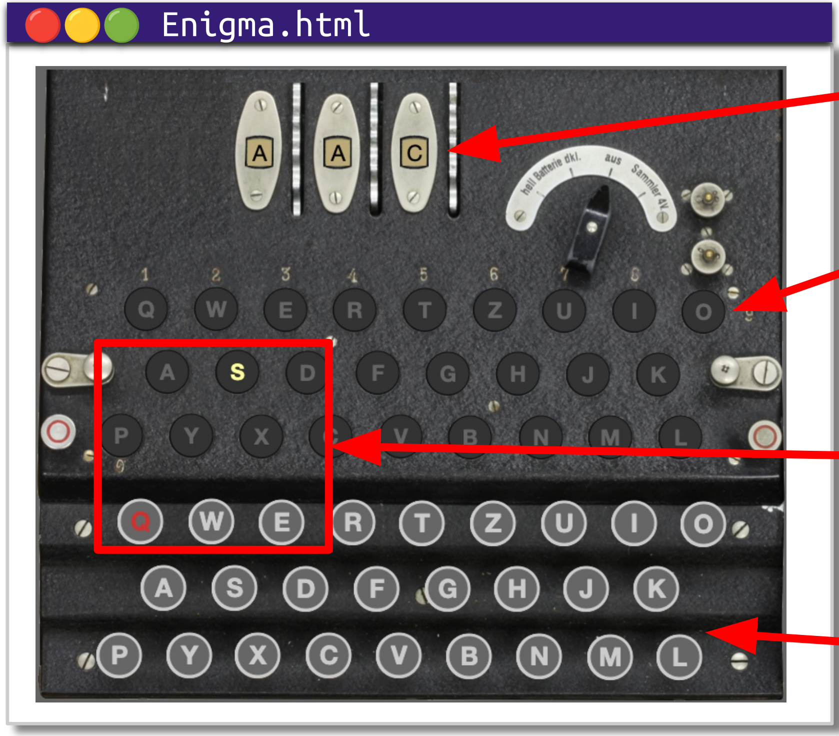
**📚 Storey House**

**Moonlights as extra Assign4 helper hours!**

*Historically accurate? No*
*Engineering / scientifically accurate? Not really*
*Dramatic, entertaining? Absolutely yes (imo)*
*Features the cooler Ben(enedict)? Oh yes yes*

# The Enigma Machine Assignment

✍️ You write a JavaScript simulation of the Enigma Machine, laying interactive GObjects atop a base provided image.



⚙️ Slow, medium, fast **rotors** in that order – whose settings can be manually adjusted by clicking on them

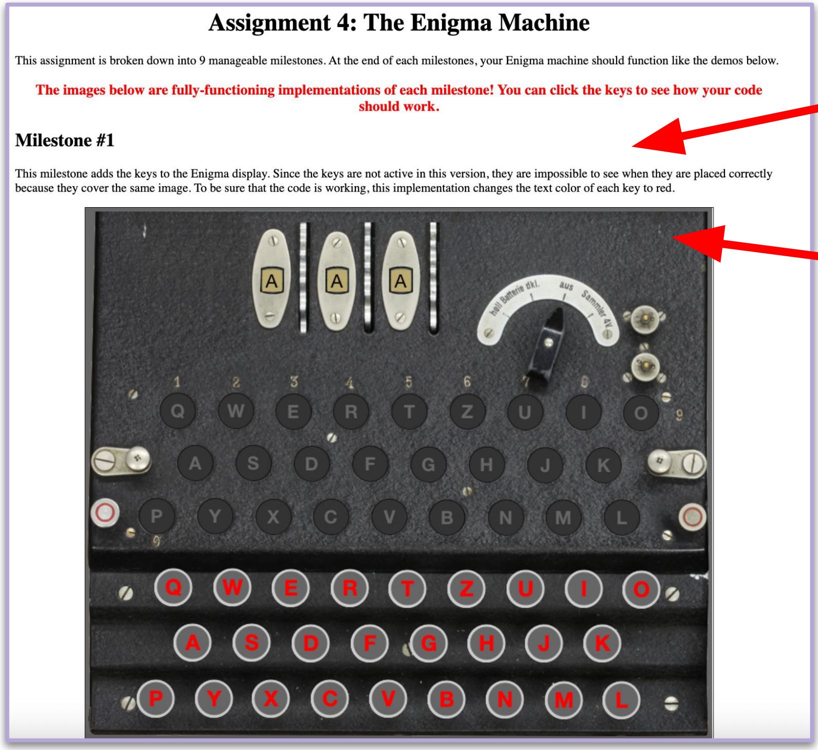
💡 Lamp panel of 26 letters, with locations in **LAMP_LOCATIONS** (*EnigmaConstants.js*)

🧮 Pressing a key down turns it red, and lights up the lamp for the encrypted letter (here, Q→S)

⌨️ Keyboard of 26 letters, with locations in **KEY_LOCATIONS** (*EnigmaConstants.js*)

## 💻 The Enigma Machine: Interactive Demo

✅ As you work through the milestones, you can **compare your Enigma program with a reference demo for each milestone**, or to see how the code should work!



Assignment 4: The Enigma Machine

This assignment is broken down into 9 manageable milestones. At the end of each milestones, your Enigma machine should function like the demos below.

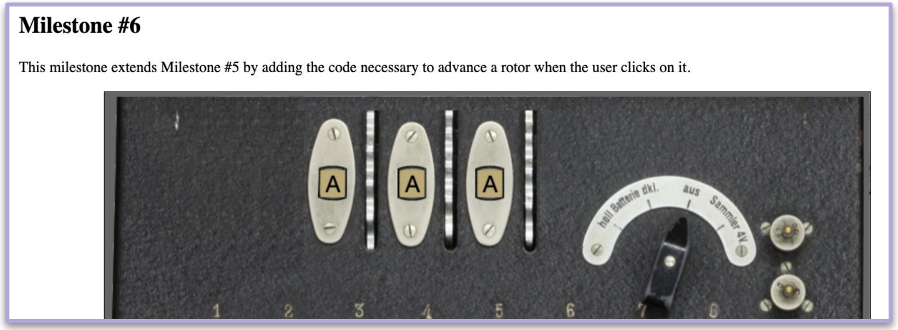
**The images below are fully-functioning implementations of each milestone! You can click the keys to see how your code should work.**

**Milestone #1**

This milestone adds the keys to the Enigma display. Since the keys are not active in this version, they are impossible to see when they are placed correctly because they cover the same image. To be sure that the code is working, this implementation changes the text color of each key to red.

📋 Description / rundown of what each milestone should accomplish

🕹️ Live Enigma interface that you can click around to see how it works!

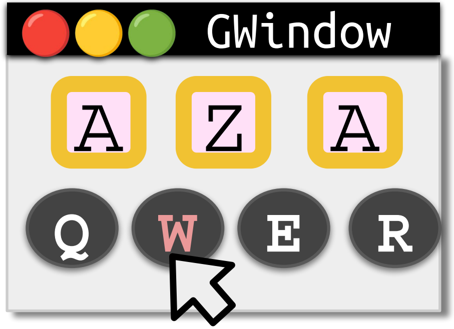


**Milestone #6**

This milestone extends Milestone #5 by adding the code necessary to advance a rotor when the user clicks on it.

# Milestone 2: Making the Keys Interactive

**Problem:** There are many objects on the screen, e.g., different keys, rotors with adjustable settings, that are supposed to do **different things** when I press each of them.



What if I try writing all code under **one event handler** function?

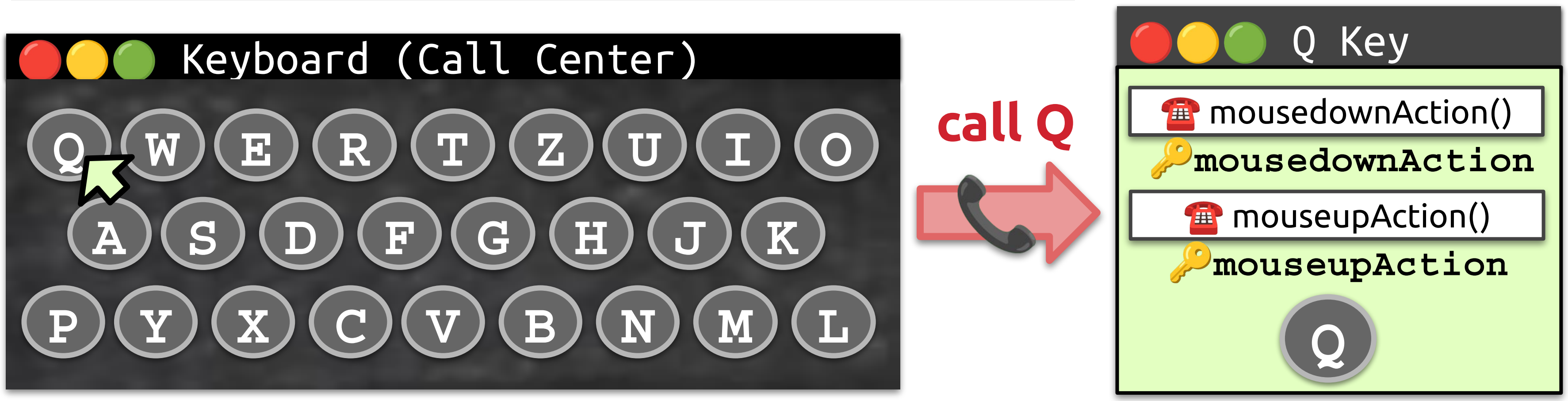
```
function mousedownAction(e){   // pseudocode sketch
      let obj = gw.getElementAt(e.getX(), e.getY());
      if (obj is the Q key) { … code to highlight Q key …
❌    if (obj is the W key) { … code to highlight W key … }
      if (obj is the E key) { … code to highlight E key …
      … one if statement for each of the 26 keys!
}
```

**Solution: Event forwarding** – let the **`GCompound`** object / key handle its own clicks, by bestowing it with its own event-handling functions which we call above

# ☎️ Event Forwarding: Phone Call Metaphor

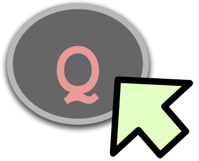
**GWindow Event forwarding**: When creating each key, also install a *phone* (event-handling functions) that responds to *calls* (window mouse events)

✍️ We should add a **mouseupAction**, **mousedownAction** function to **each key object**, to manipulate its GLabel letter.



key. mousedownAction = function mousedownAction(e){ ... }

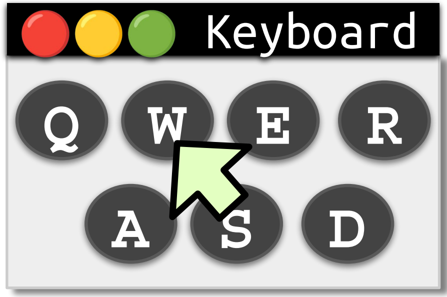
key. mouseupAction = function mouseupAction(e){ ... }



# ☎️ Event Forwarding: Phone Call Metaphor

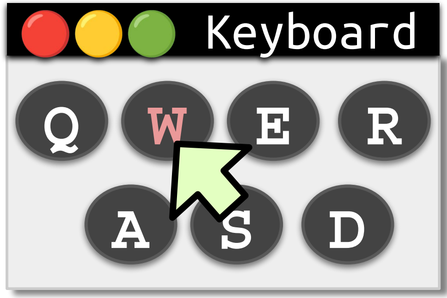
1. When the user presses object, see if it is a key (has a **phone**)

```
function mousedownAction(e){
        let target = gw.getElementAt(e.getX(), e.getY());
        if (target.mousedownAction !== undefined){ // is a key
                target.mousedownAction();
        } // the event-handling is forwarded to the individual key
}
gw.addEventListener(mousedownAction); 🔊
```



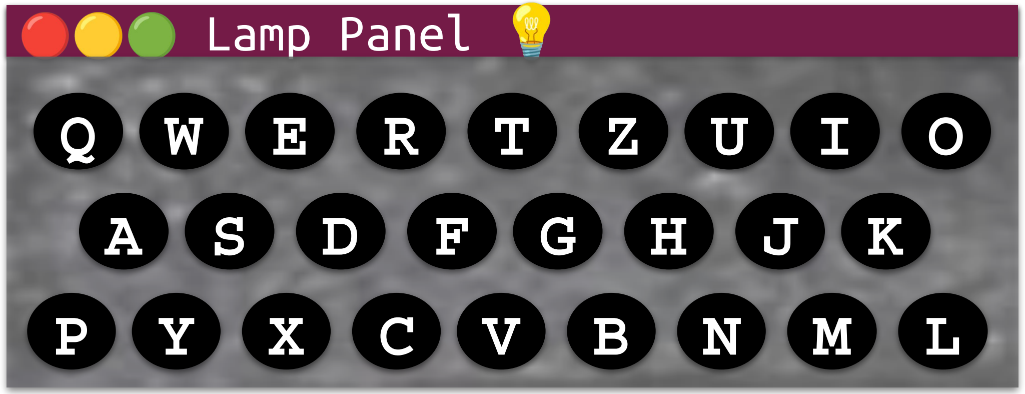
2. If so, dial that specific key's **mousedownAction** function, which 📞 *"answers the phone"* by changing the key color. Event forwarding!

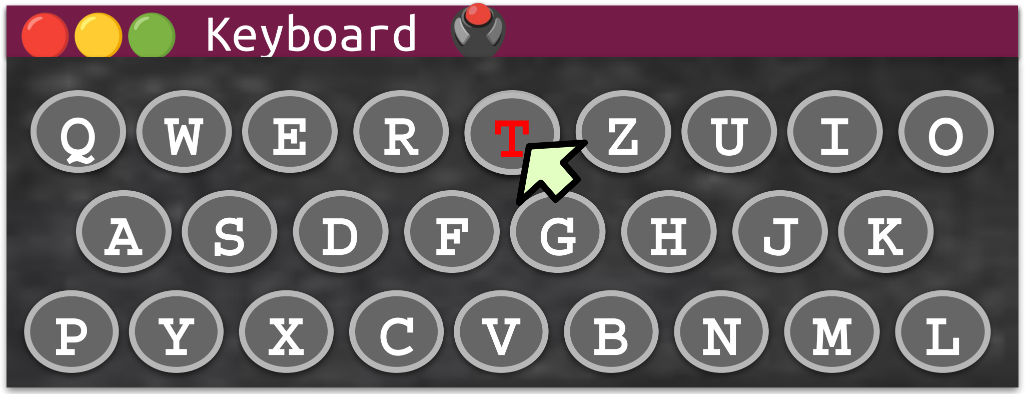
```
key.mousedownAction = function mousedownAction(e){
        // highlights the letter of that specific key as RED
}
```



*Each key should have a function for mouseup as well!*

## 💡 Milestone 4: Wiring Keys to Lamps

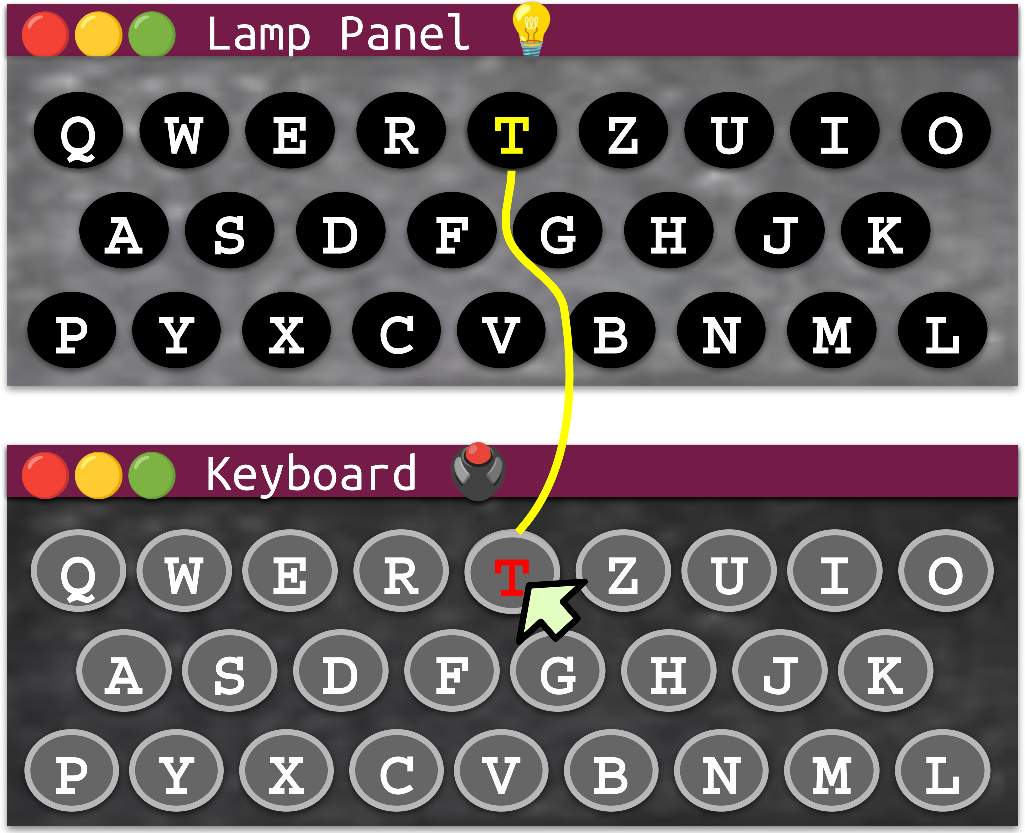




When we mouse-down a key, the **corresponding lamp** should illuminate, e.g., T → T (later lamp will be encrypted letter)

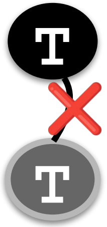
**Problem:** The key objects don't have access to the lamps, can't modify them.

# 💡 Milestone 4: Wiring Keys to Lamps



When we mouse-down a key, the **corresponding lamp** should illuminate, e.g., T → T (later lamp will be encrypted letter)

**Problem:** The key objects don't have access to the lamps, can't modify them.

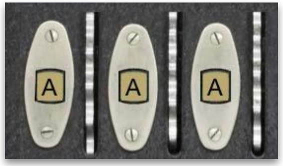


**Solution:** Like Breakout/Wordle, use **outer state variables**! At the top of *runEnigmaSimulation*, we create an **`enigma`** object:

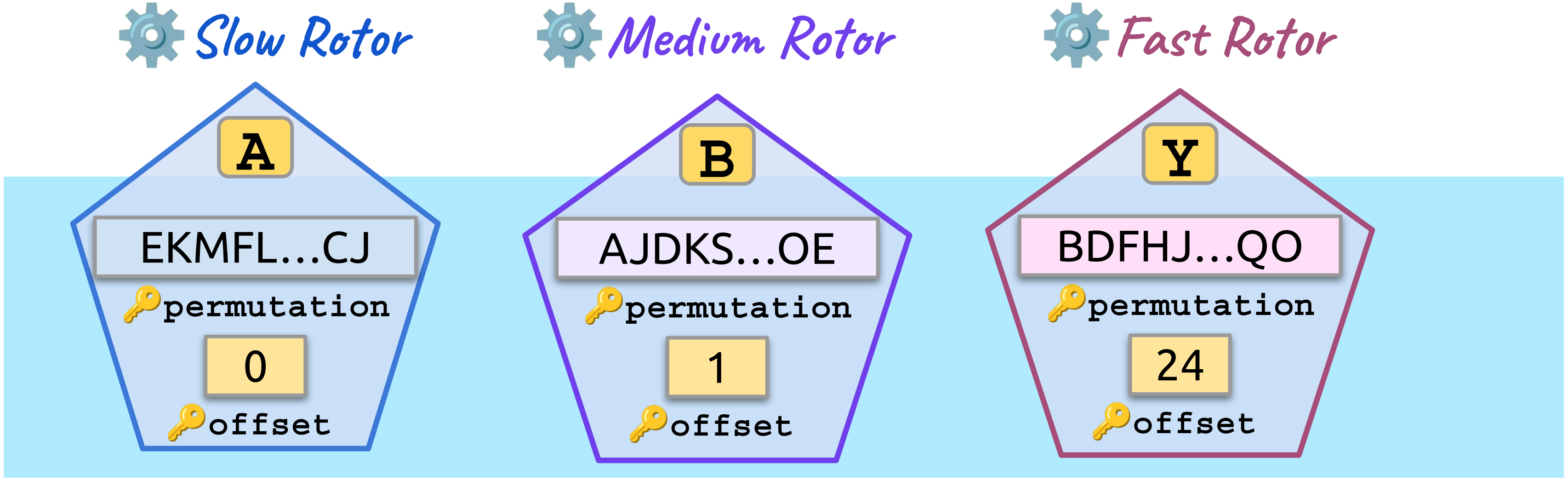
```
let enigma = {
  keys: [], lamps: []
} //arrays of keys, lamps
```

✅ This **`enigma`** object is then passed into each key's event handlers to give access to lamps

# Milestones 5-7: Rotors as Icebergs



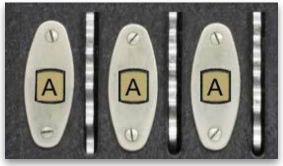
- ❑ On the exterior, a rotor appears merely as a gold square with a letter from A to Z.
- ❑ The true weight of it is **lurking underneath**: its own **fixed permutation string** that —along with its **offset** (A=0,B=1,C=2,...)—determines how it transform letters



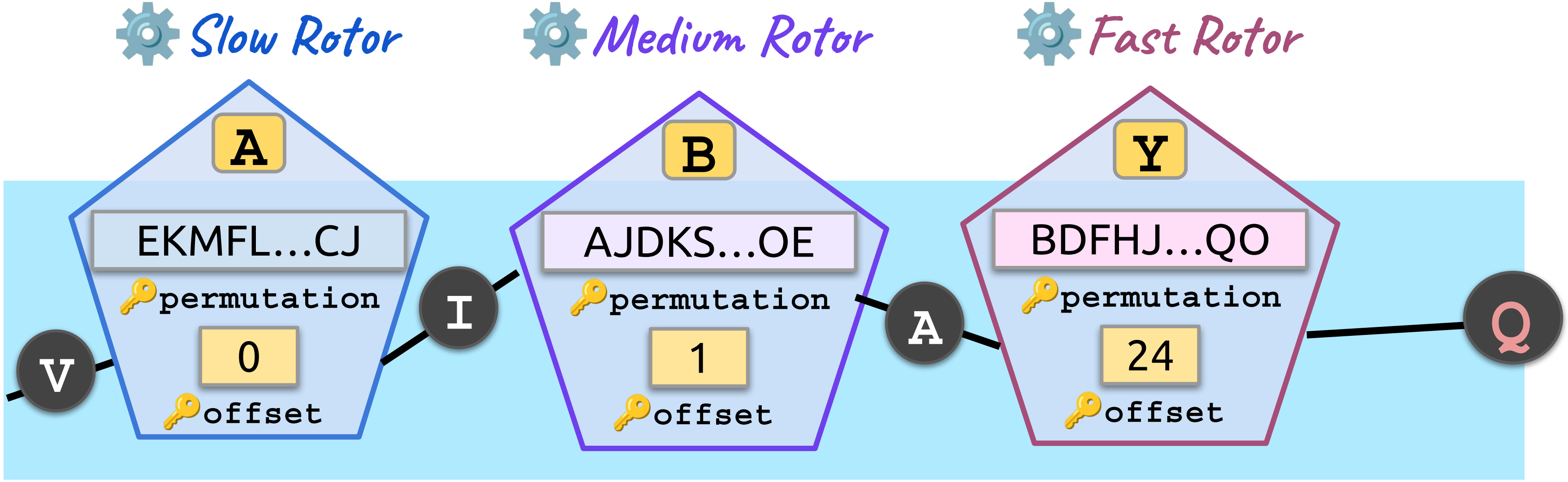
✍ Accordingly, each rotor object should store its own permutation and offset, e.g.,

```
rotor.permutation = ROTOR_PERMUTATIONS[i], rotor.offset=...
```

# Milestones 5-7: Rotors as Icebergs



- ❑ On the exterior, a rotor appears merely as a gold square with a letter from A to Z.
- ❑ The true weight of it is lurking underneath: its own **fixed permutation string** that—along with its **offset** (A=0,B=1,C=2,…)—**determines how it transform letters**



💡 Each rotor applies its **permutation cipher + offset** to the letter to **transform** it!

# Milestone 6: Rotor Click Action

+1

| 1 |
|---|

🔑offset

*The user can manually adjust each offset by **clicking the rotor's letter!***

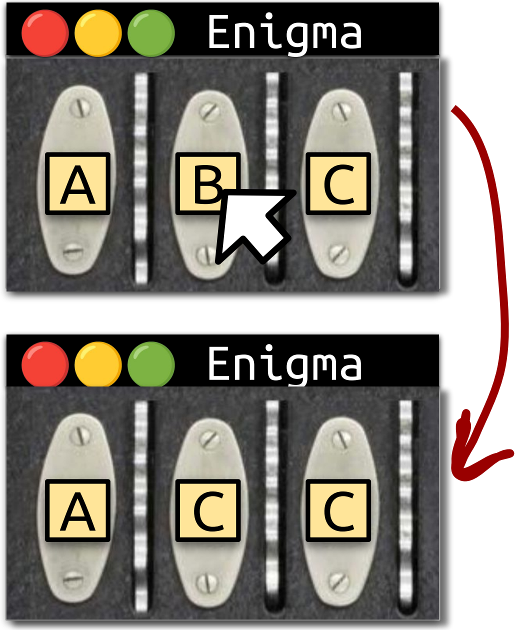
⚙️⚙️⚙️ With multiple rotors, we'll want to perform **event forwarding just like with keys.**

1. When the user presses object, see if it is a rotor

```
function clickAction(e){
        // get object clicked and if it has a clickAction property {
                target.clickAction();
        } // the event-handling is forwarded to that rotor
}
```

2. If so, dial that specific rotor's **clickAction** function, which *"answers the phone"* by advancing the rotor. Event forwarding!

```
rotor.clickAction = function clickAction(e){
        advanceRotor(rotor); // helper function to write: increment
                             offset by 1, change displayed letter accordingly
} // note if offset is 25 (Z), it should wrap back to 0 (A)
```

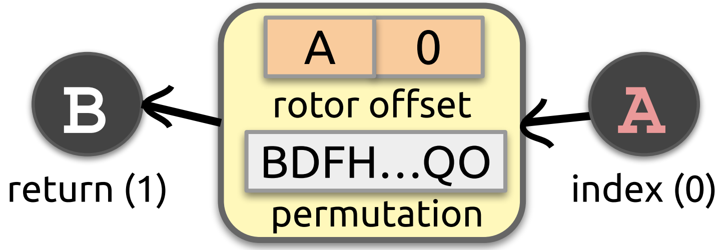


| 0 | 2 | 2 | offset |
|---|---|---|---|

# ⚙️ Milestone 7: Implement one stage of encryption

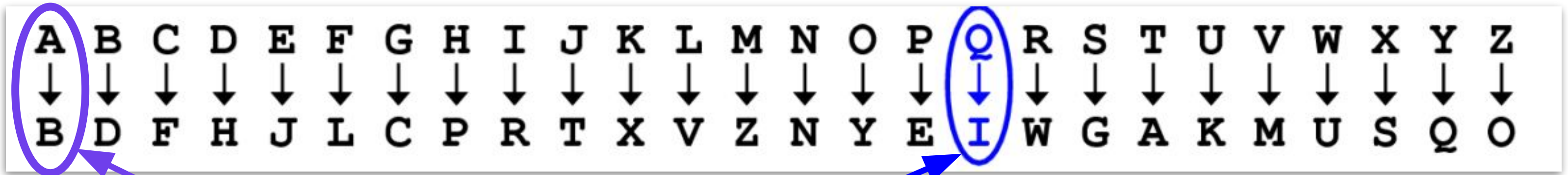
```
applyPermutation(index, permutation, offset)
```

✍️ Life will be a lot easier if you write a general **helper function** that applies a **single permutation cipher** with an offset to a letter / index — instead of trying to code all 7 permutation stages separately.



💡 *Then, for each permutation stage in the full path, you can use the helper function!*

How does a permutation cipher string, e.g., **BDFHJLCPRTXVZNYEIWGAKMUSQO** work?



| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| B | D | F | H | J | L | C | P | R | T | X | V | Z | N | Y | E | I | W | G | A | K | M | U | S | Q | O |

ALPHABET[0], or **“A”**, maps to PERMUTATION[0], or **“B”**

ALPHABET[16], or **“Q”**, maps to PERMUTATION[16], or **“I”**

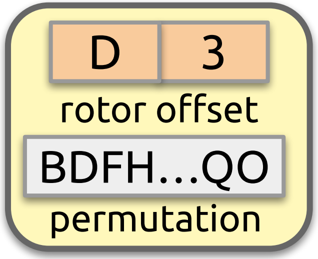
In general, for index 0 to 25, **ALPHABET[i]** → **PERMUTATION[i]**

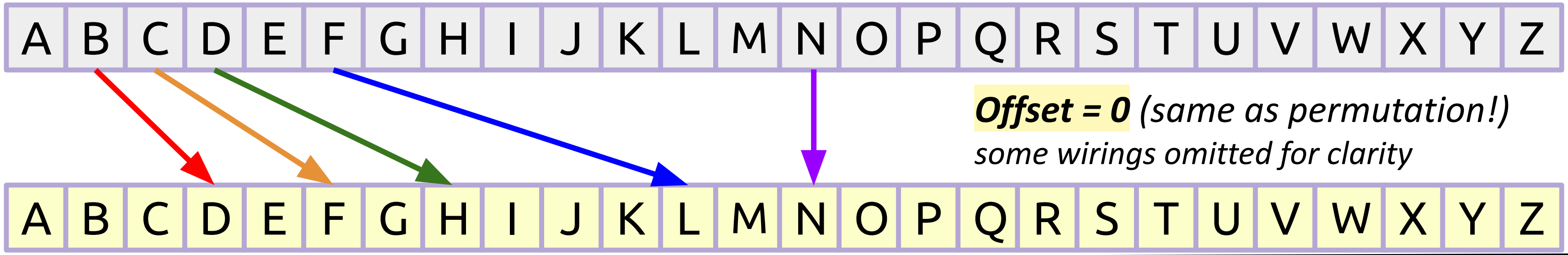
## Milestone 7: Implement one stage of encryption

`applyPermutation(index, permutation, offset)`

For a given permutation, how does the offset come into play?

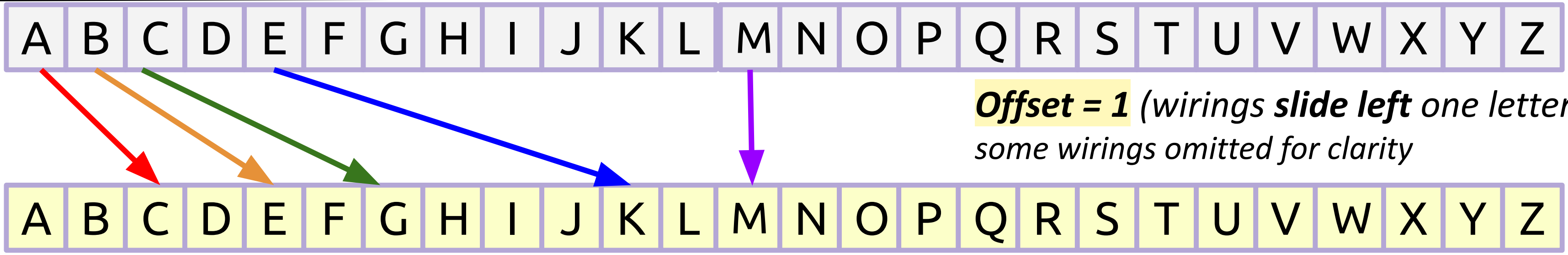
*The offset **slides or rotates the permutation** in cyclical fashion like a Caesar cipher, e.g., if offset = 1, we would use the permutation wirings for the next letter down in the alphabet.*




A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

***Offset = 0** (same as permutation!)*

*some wirings omitted for clarity*


A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

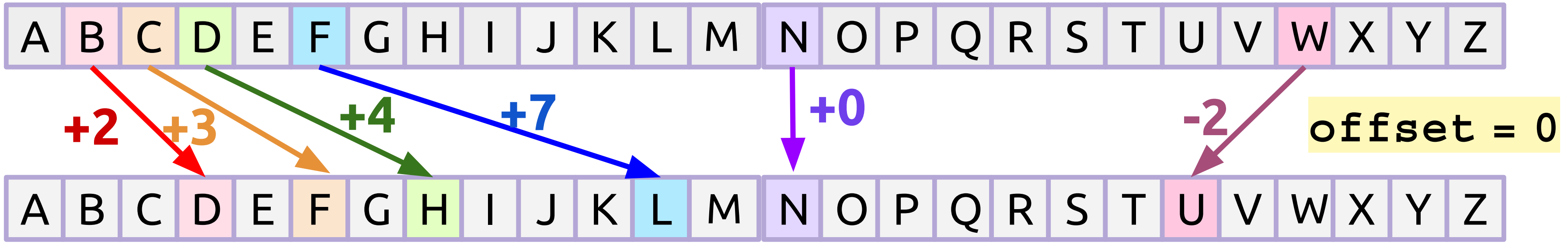
***Offset = 1** (wirings **slide left** one letter)*

*some wirings omitted for clarity*

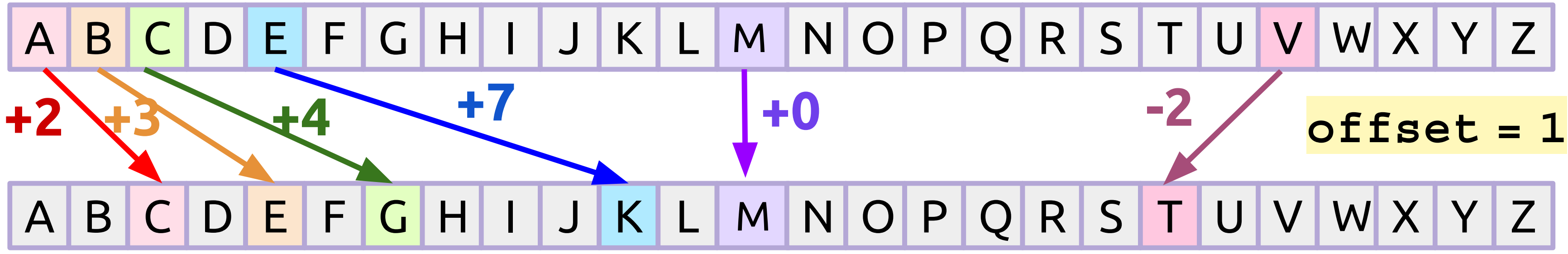
## ⚙️ Milestone 7: apply Permutation(index, permutation, offset)

🏃 We can think of a **permutation as jumps along wires / Harry Potter staircases**, e.g.,

- ❑ A→B (+1), B→D (+2), C→F (+3), D→H (+4), …, W→U (−2), …



A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
+2
+3
+4
+7
+0
-2
offset = 0
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

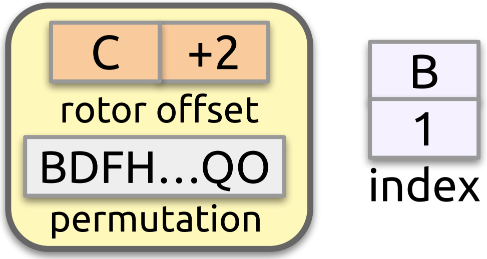


A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
+2
+3
+4
+7
+0
-2
offset = 1
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

- ❑ To get what **A** maps to, with the **offset = 1** we use the **+2 wiring** the next letter or B originally had, so A becomes **C** by jumping **+2 characters**. We need to **look ahead** 🤔
- ❑ Similarly, **B** uses the **+3 wiring** C originally had, so **B**→**E** by leaping **+3 characters.**

# Milestone 7: Suggested Pseudocode

```
function applyPermutation(index, permutation, offset) {
    Compute the index of the letter after shifting it by the offset, wrapping around if necessary.
    Look up the character at that index in the permutation string.
    Return the index of the resulting character after subtracting the offset, wrapping if necessary.
}
```

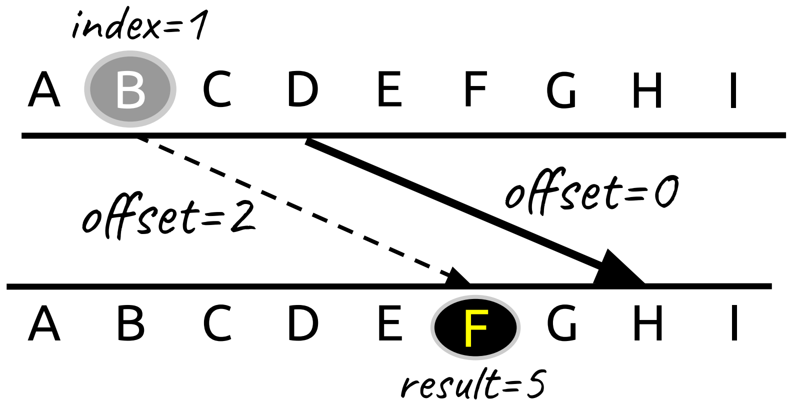


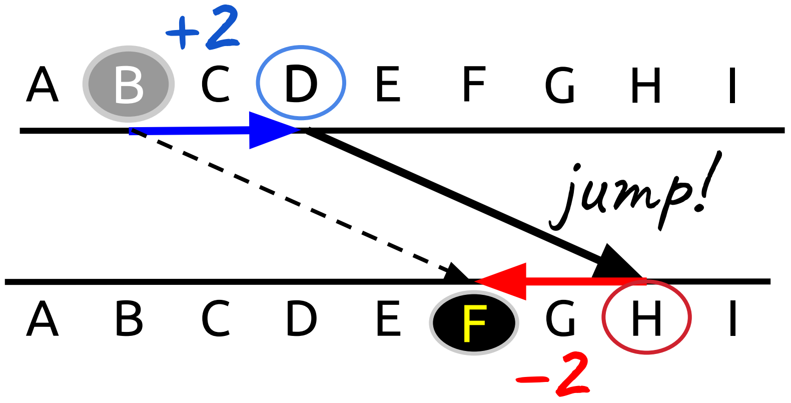
**Key Idea: We can be proactive!** Instead of waiting for the permutation wiring to come down, if say ***offset = 2***, we **already know we'll use the wiring 2 characters up!**

(1) So let's **head there first** by moving up *[offset]* chars!

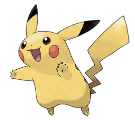
(2) Then jump across the permutation wiring there to the other side

(3) **Move back** *[offset]* chars to account for the initial move.

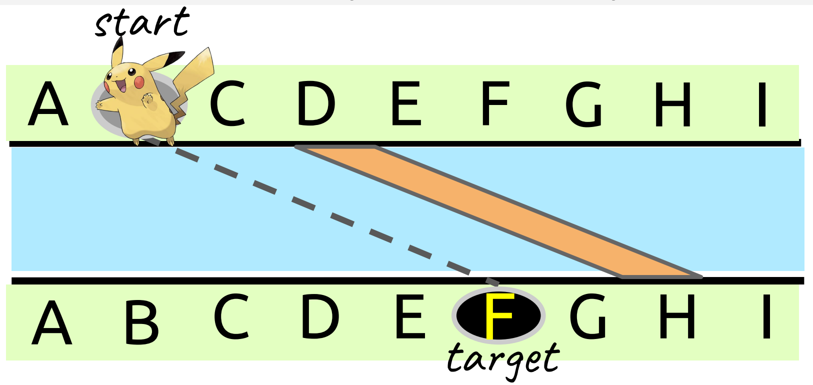




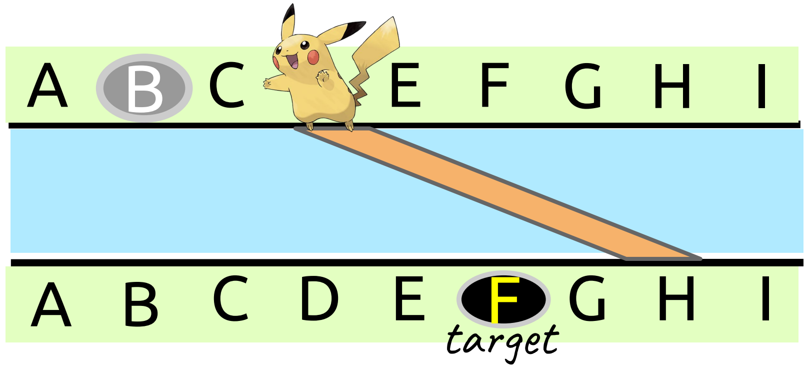
# Milestone 7: applyPermutation story metaphor



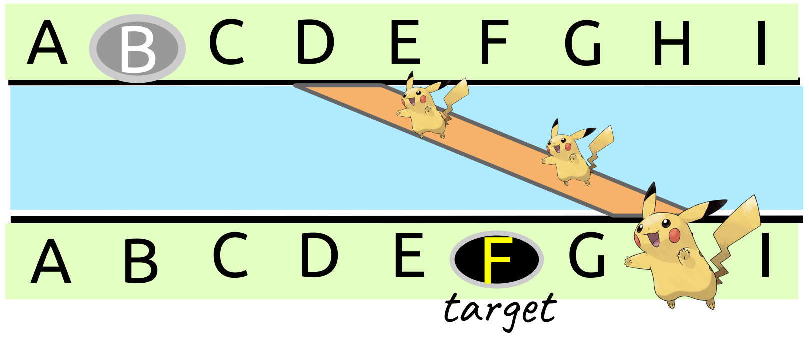
**1.** You're at B and want to cross the river to **F** on the other side, but you can't easily swim across.



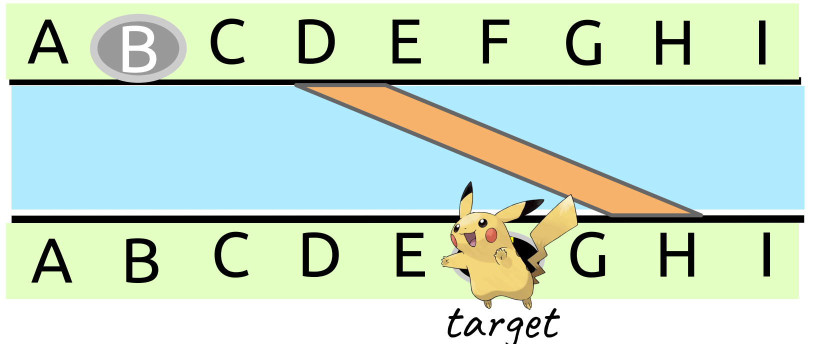
**2.** You walk *offset* = 2 blocks right to where a nearby bridge from **D to H** 🌉 begins.



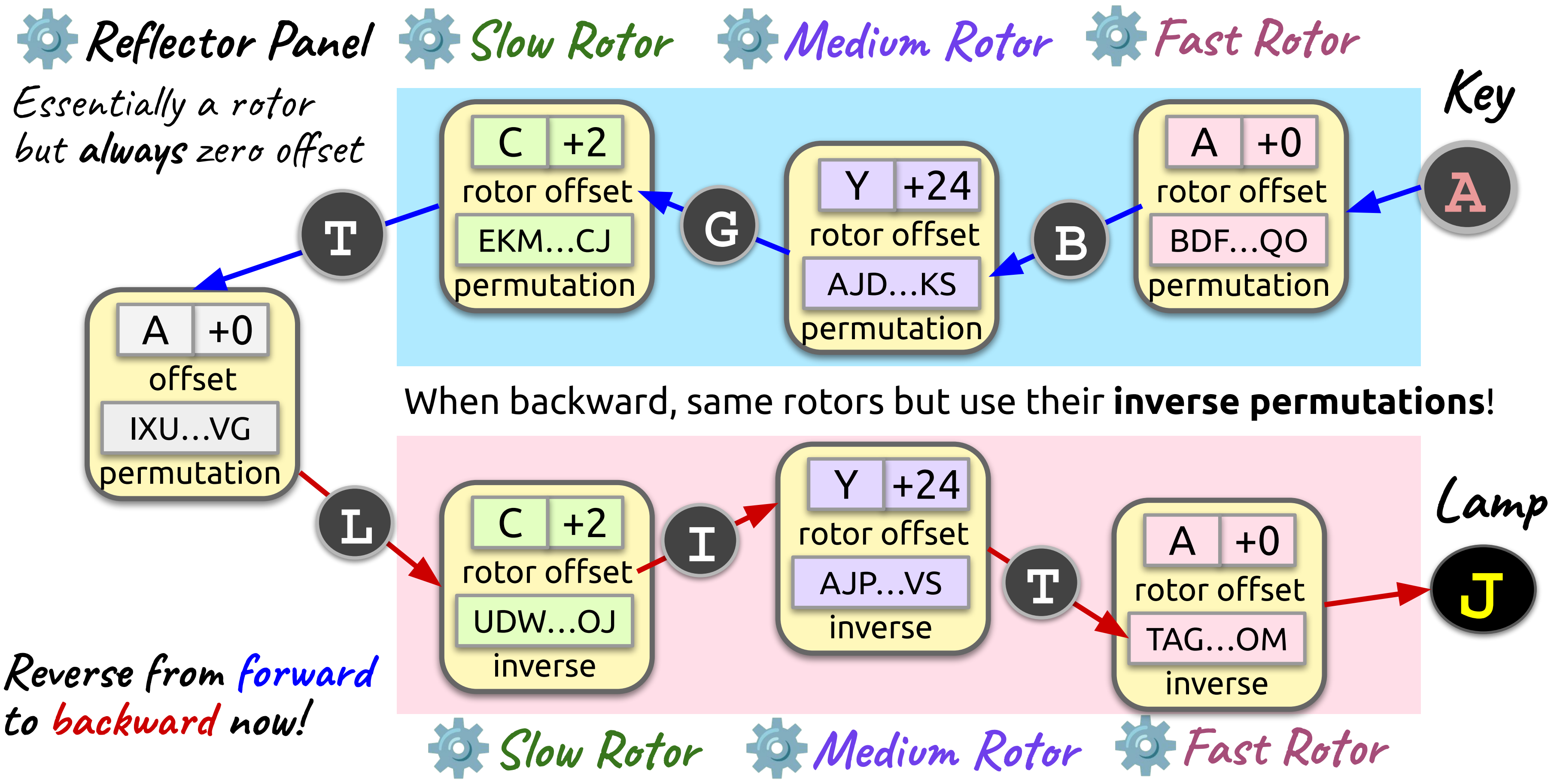
**3.** You cross the bridge from D to **H.**



**4.** You walk back *offset = 2* blocks to reach **F.**



# 🛤️ Milestone 8: Full encryption path from pressed key to lit lamp



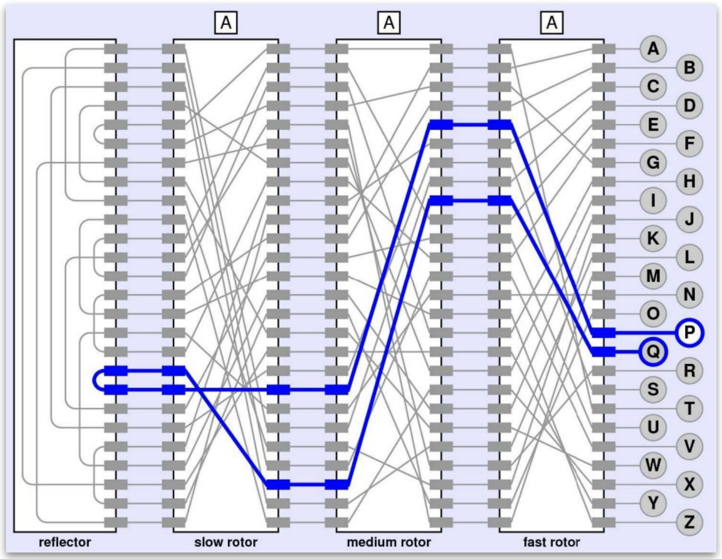
When backward, same rotors but use their **inverse permutations**!

*Reverse from forward to backward now!*

## 🛤️ Milestone 8: Implement full encryption path

💡 *For each permutation in the full path, you can use your earlier* ***helper function****!*

| | |
|---|---|
| 1 | Pass through **fast rotor**, forward |
| 2 | Pass through **medium rotor**, forward |
| 3 | Pass through **slow rotor**, forward |
| 4 | Pass through **reflector panel** |
| 5 | Pass through **slow rotor**, backward |
| 6 | Pass through **medium rotor**, backward |
| 7 | Pass through **fast rotor**, backward |



🪞 *Next up: For backward current, getting the* ***inverse*** *of permutations!*

# 🔄 Milestone 8: Inverse Permutation

To handle the **backward permutations**, you'll need to implement **invertKey():**

- ❑ Takes in a permutation and returns the inverse / reverse permutation also as a string.
- ❑ Inversion is described in detail in the handout. **Test via console.log()!** 📠

permutation="QWERTYUIOPASDFGHJKLZXCVBNM"

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| Q | W | E | R | T | Y | U | I | O | P | A | S | D | F | G | H | J | K | L | Z | X | C | V | B | N | M |

The **inverse permutation** is below (note the 🌀 **reversed** mappings, e.g., **K→A** to **A→K**)

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| K | X | V | M | C | N | O | P | H | Q | R | S | Z | Y | I | J | A | D | L | E | G | W | B | U | F | T |

inverse="KXVMCNOPHQRSZYIJADLEGWBUFT"

# Milestone 8: Reversed Curse Technique

permutation=”QWERTYUIOPASDFGHJKLZXCVBNM”

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| Q | W | E | R | T | Y | U | I | O | P | A | S | D | F | G | H | J | K | L | Z | X | C | V | B | N | M |

Let’s build the inverse permutation letter by letter in **alphabetic order**!

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## Milestone 8: Reversed Curse Technique

permutation="QWERTYUIOPASDFGHJKLZXCVBNM"

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| Q | W | E | R | T | Y | U | I | O | P | A | S | D | F | G | H | J | K | L | Z | X | C | V | B | N | M |

Let's build the inverse permutation letter by letter in **alphabetic order**!

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ↓ | | | | | | | | | | | | | | | | | | | | | | | | | |
| **K** | | | | | | | | | | | | | | | | | | | | | | | | | |

What maps to **A** in the regular permutation?

Searching the permutation / bottom of the table above, we have K $\longrightarrow$ **A** , so then **A** $\longrightarrow$ **K** in the inverse.

# Milestone 8: Reversed Curse Technique

permutation="QWERTYUIOPASDFGHJKLZXCVBNM"

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| Q | W | E | R | T | Y | U | I | O | P | A | S | D | F | G | H | J | K | L | Z | X | C | V | B | N | M |

Let's build the inverse permutation letter by letter in **alphabetic order**!

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ↓ | ↓ | | | | | | | | | | | | | | | | | | | | | | | | |
| K | **X** | | | | | | | | | | | | | | | | | | | | | | | | |

What maps to **B** in the regular permutation?

Searching the permutation / bottom of the table above, we have X $\longrightarrow$ **B** , so then **B** $\longrightarrow$ **X** in the inverse.

# Milestone 8: Reversed Curse Technique

permutation="QWERTYUIOPASDFGHJKLZXCVBNM"

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| Q | W | E | R | T | Y | U | I | O | P | A | S | D | F | G | H | J | K | L | Z | X | C | V | B | N | M |

Let's build the inverse permutation letter by letter in **alphabetic order**!

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ↓ | ↓ | ↓ | | | | | | | | | | | | | | | | | | | | | | | |
| K | X | **V** | | | | | | | | | | | | | | | | | | | | | | | |

What maps to **C** in the regular permutation?

Searching the permutation / bottom of the table above, we have V → **C** , so then **C** → **V** in the inverse.

# Milestone 8: Reversed Curse Technique

permutation="QWERTYUIOPASDFGHJKLZXCVBNM"

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| Q | W | E | R | T | Y | U | I | O | P | A | S | D | F | G | H | J | K | L | Z | X | C | V | B | N | M |

Let's build the inverse permutation letter by letter in **alphabetic order**!

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| K | X | V | M | C | N | O | P | H | Q | R | S | Z | Y | I | J | A | D | L | E | G | W | B | U | F | **T** |

What maps to **Z** in the regular permutation?

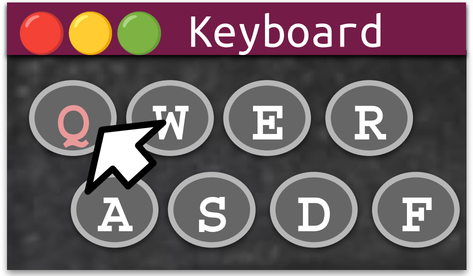
Searching the permutation / bottom of the table above, we have T → **Z**, so then **Z** → **T** in the inverse.

Repeating this for all letters from A to Z – we have our inverse!

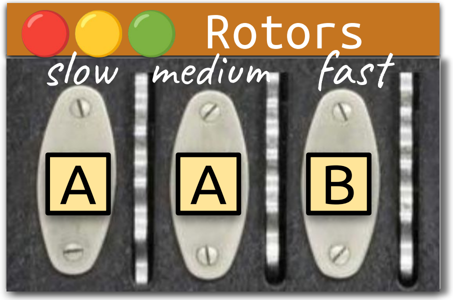
# Milestone 9: Synchronized Rotor Advancing

Lastly, when a key is pressed, the fast rotor should be advanced **prior to encryption**.

1. User presses key

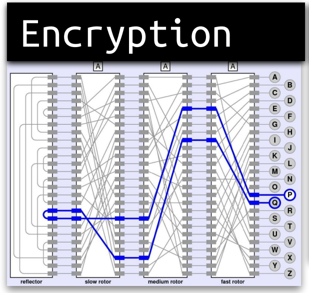


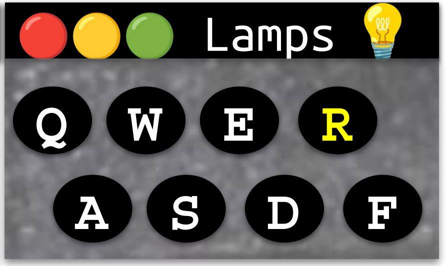
2. Fast rotor advances



originally A A A

3. Encryption occurs on new settings, lights lamp of encrypted letter





**Synchronization:** When the **fast rotor** rolls over from Z to A, the **medium rotor** should be incremented; when the **medium rotor** rolls over, the **slow rotor** should be incremented.

A B Z $\xrightarrow{\text{advance}}$ A C A     A Z Z $\xrightarrow{\text{advance}}$ B A A

Tip: Revise **advanceRotor**() to return a boolean: true if it wrapped back to 0, else false.

## 📠 Final Tips for The Enigma Machine

- ❑ Write and test your code incrementally
  - ❑ **Set a milestone schedule**, knocking them down over the course of the week
  - ❑ Another 15-page handout, so start early!
- ❑ 🧩 **Decompose early!** e.g., if you have 26 keys to generate, it may help to write **createKey(num)**, and similarly for lamps panel
- ❑ 🧮 **Draw out permutation, inversion examples on paper!** This is among the trickiest parts imo
- ❑ Come see us in **office hours** and **film showing**!
  - ❑ Test and test your program, and verify with the **milestones demo** online!

**Sample Milestones Schedule**

**Sun:** Read handout, go to YEAH! :) Get as much of [1]-[3] done as possible
**Mon:** Other work day 🥺
**Tues:** Work on [4]-[6]
**Wed:** Movie night! Work on permutations [7]-[8]
**Thurs:** Pull up to OH for last stretch [9] / group support, extensions? 🙂
**By Fri:** Submit! Relax

## Best wishes on midterms! :)

Please feel free to reach out if you have any questions, e.g., in the office hours following the overview portion of the YEAH session