Strings & Runes

01

Text Encoding

02

Runes

03

Strings

Text Encoding

- Textual data in Go uses UTF-8 **encoding**
- Encoding is a way to represent thousands of different symbols using **code pages**
- Code pages are tables which use the first few bytes of data to determine which page to use
  - Each symbol in the code page is called a **code point**

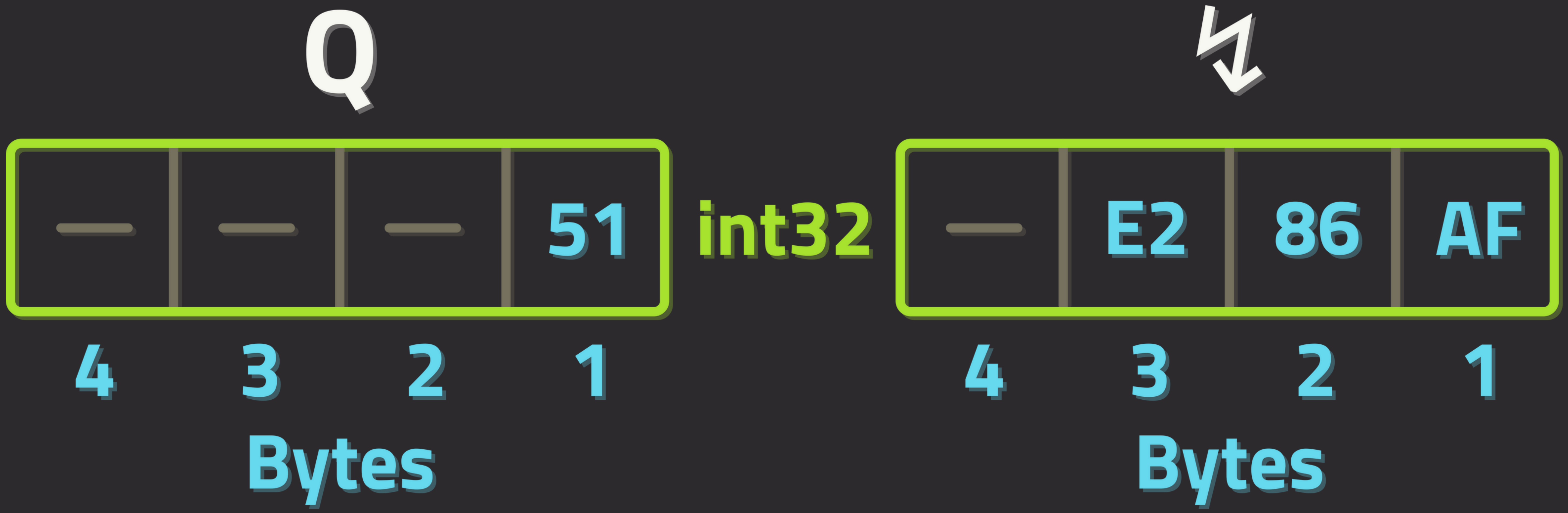
Example Code Page

	0	1	2	3
0	A 00	B 01	C 02	D 03
1	a 10	b 11	c 12	d 13
2	∉ 20	⇒ 21	∀ 22	∃ 23
4F	あ 4F0	い 4F1	う 4F2	え 4F3

Runes

- Text is represented using the **rune** type
  - Similar to **char** in many other programming languages
- **Rune** is an alias for int32 (32-bit integer)
  - Always a number: will print numeric value unless proper formatting is specified
- A rune can represent any symbol
  - Letters, numbers, emoji, etc

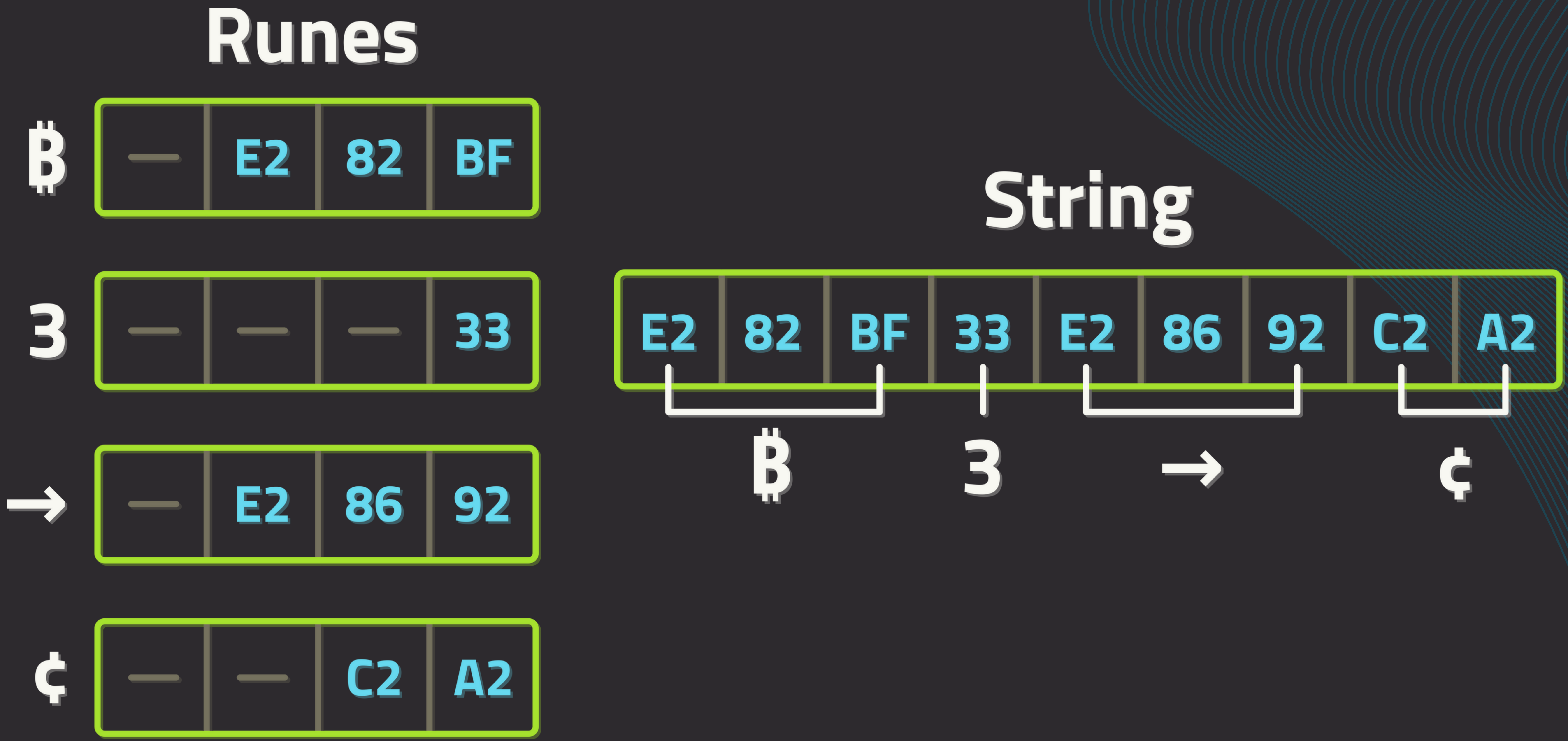
Rune Byte Representation



Strings

- A **string** is the data type for storing multiple runes
- Strings are just an array of **bytes** and a string length
  - There is no null termination with a Go **string**
- When iterating a string, iteration occurs over **bytes**
  - Bytes are **not** symbols
  - Special iteration required to retrieve runes/symbols

String Byte Representation



Creation

- Runes: `'a' 'R' '7' '\n' 'Ω' '₹' '½'`
- Strings: `"Amount is €22\n"`
  `"k"`
- Raw Literal: ``` `Let's code in "Golang!"\n` ```

Recap

- Text in Go is encoded using **UTF-8**
- The **rune** type can represent any individual symbol
  - **rune** is an alias for **int32**
  - They are created using single quotes: '
- The **string** type contains a series of symbols as **bytes**
  - Strings are **not** null terminated
  - They are created using double quotes: "
- Raw literals are created using backticks: `