Computer Architecture and Organization

CS 115

Lecture 3

Instructor: **Gerald John M. Sotto**

Last Updated: September 10, 2025

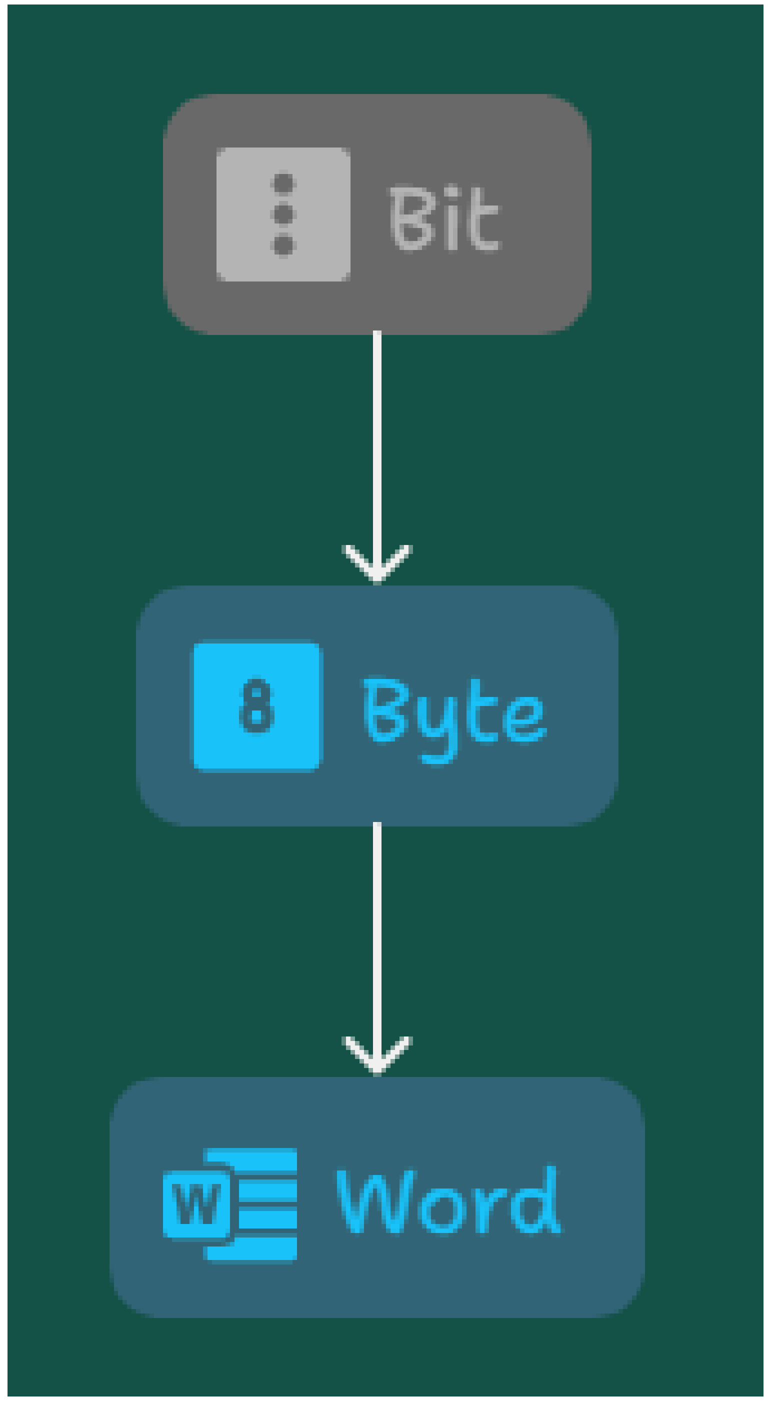
Table of Contents

Unit II: Machine Level Representation of Data

- Bits, bytes, and words
- Numeric data representation and number bases
- Representation of non-numeric data (character codes, graphical data)

Bits, bytes, and words

Bit, Bytes, Word

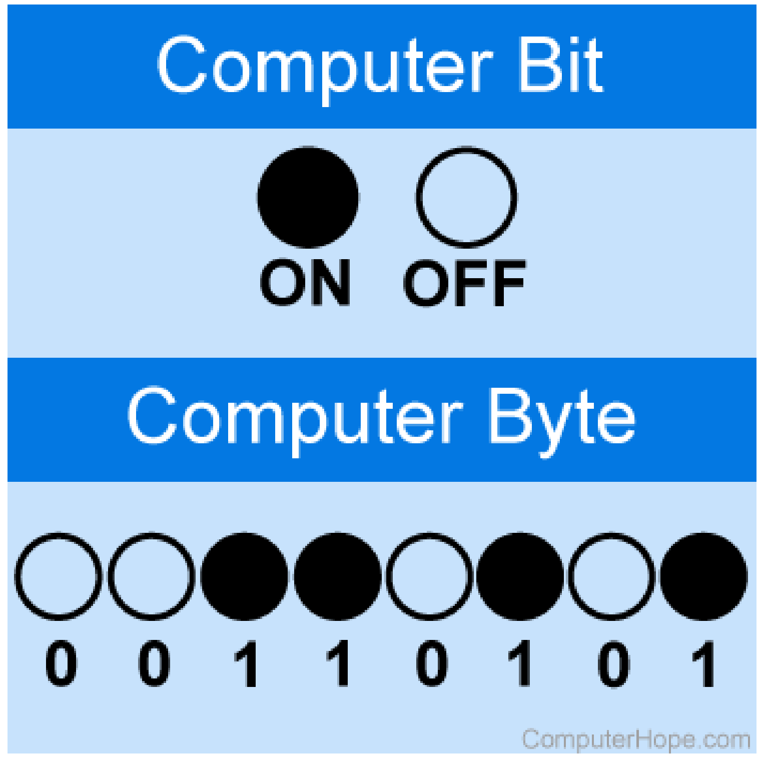


Bit, Bytes, Word

- A bit (short for binary digit)
- Smallest unit of data and can have only one of two values: 0 or 1
- Think of it as a single light switch that is either on or off. This is the foundation of all digital logic.

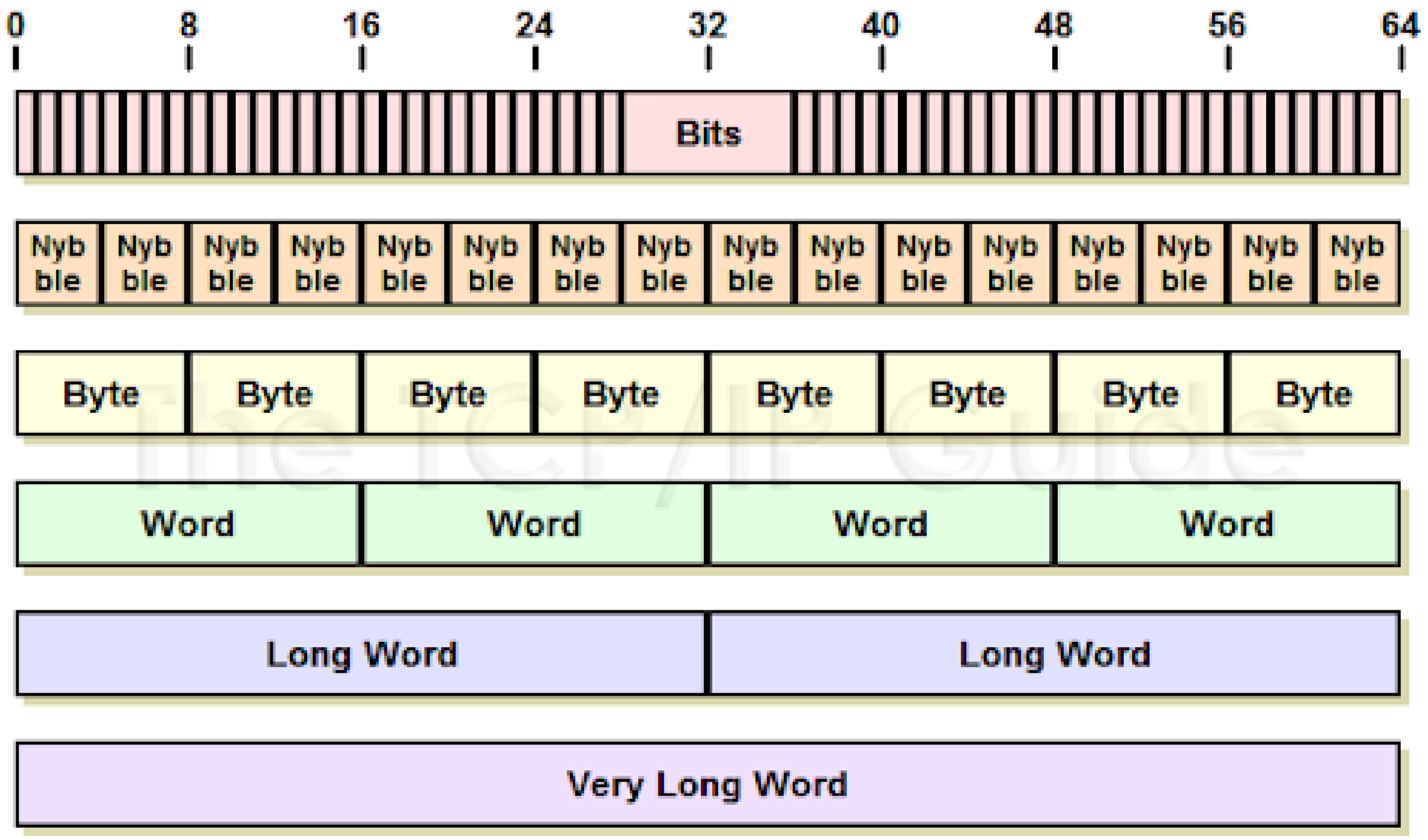
Bit, Bytes, Word

- A byte is a group of eight bits.
- This is the standard fundamental unit of storage in most modern computer systems.
- For example, a single character, like the letter 'A', is typically stored as a byte. The reason we use a byte is that with eight bits, we can represent 28=256 different combinations, which is enough to encode all the standard characters (letters, numbers, symbols) we need.



Bit, Bytes, Word

- A **word is the natural unit of data** used by a particular processor design.
- The size of a word is specific to the computer's architecture.
- For example, a 32-bit processor has a word size of 32 bits (4 bytes), while a 64-bit processor has a word size of 64 bits (8 bytes). This is a crucial concept because the word size determines how much data the CPU can process at a time. The CPU's registers, which are its temporary storage locations, are typically the size of a word.



Bit, Bytes, Word

Relevance: These terms are the language of computer hardware. They're essential for understanding memory addressing, data transfer rates, and processor capabilities. If a professor asks you what a 64-bit processor means, you'll know it refers to its word size, not just a marketing term.

Numeric Data Representation and Number Bases

Numeric Data Representation and Number Bases

- This is the foundation of how computers perform calculations. All arithmetic operations within the ALU are done in **binary**. Understanding number bases is crucial for debugging low-level code and interpreting memory dumps.

Binary (Base-2)

- **Binary to Decimal**
- Binary (Base–2): Uses only digits 0 and 1. Each position represents a power of 2
- $101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
  - $= 4 + 0 + 1$
  - $= 5_{10}$
- $101101_2 = 1x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0$
  - $= (1 \times 32) + (0 \times 16) + (1 \times 8) + (1 \times 4) + (0 \times 2) + (1 \times 1)$
  - $32 + 0 + 8 + 4 + 0 + 1$
  - $= 45_{10}$

Binary (Base-2)

Binary to Decimal Bit Value Table

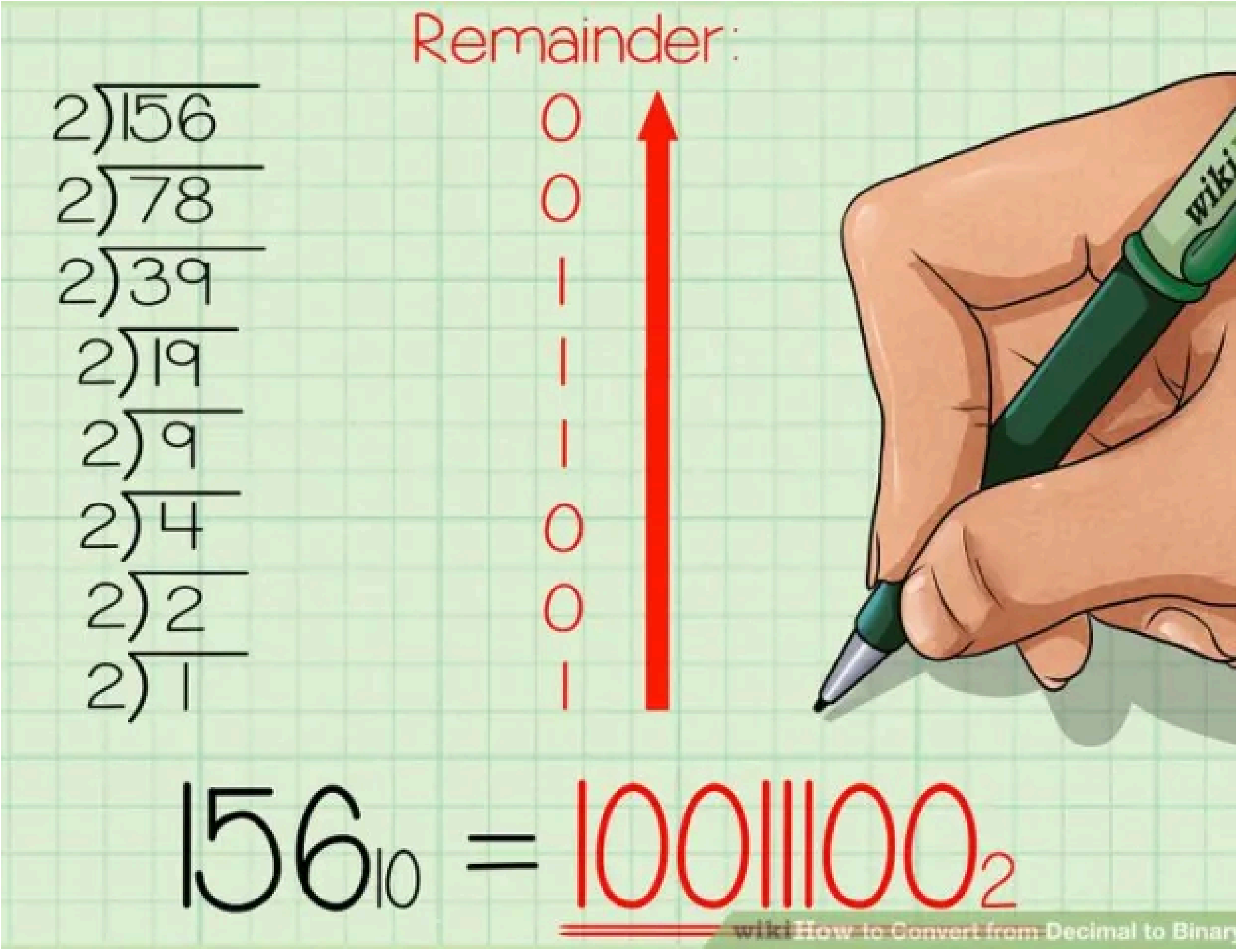
(8 Bits in a Byte)

128	64	32	16	8	4	2	1
1 (On) + 128	1 (On) +64	1 (On) +32	1 (On) +16	1 (On) +8	1 (On) +4	1 (On) +2	1 (On) +1

255

Binary (Base-2)

- **Decimal to Binary**



Octal (Base-8)

- The octal number system is a base-8 system, using **digits 0 through 7**. Its main advantage lies in its relationship with the binary system (base-2), as one octal digit can represent exactly three binary digits.

Octal (Base-8)

- This makes it a **useful shorthand for representing long binary** numbers, especially in older computing contexts. For example, the binary number 110101 can be easily grouped into sets of three digits from the right: 110 and 101. Converting each group to its decimal equivalent gives us 6 and 5, respectively, which results in the octal number 65.
- **Binary to Octal**

$$110101_2$$

110 101

Octal (Base-8)

110101_2

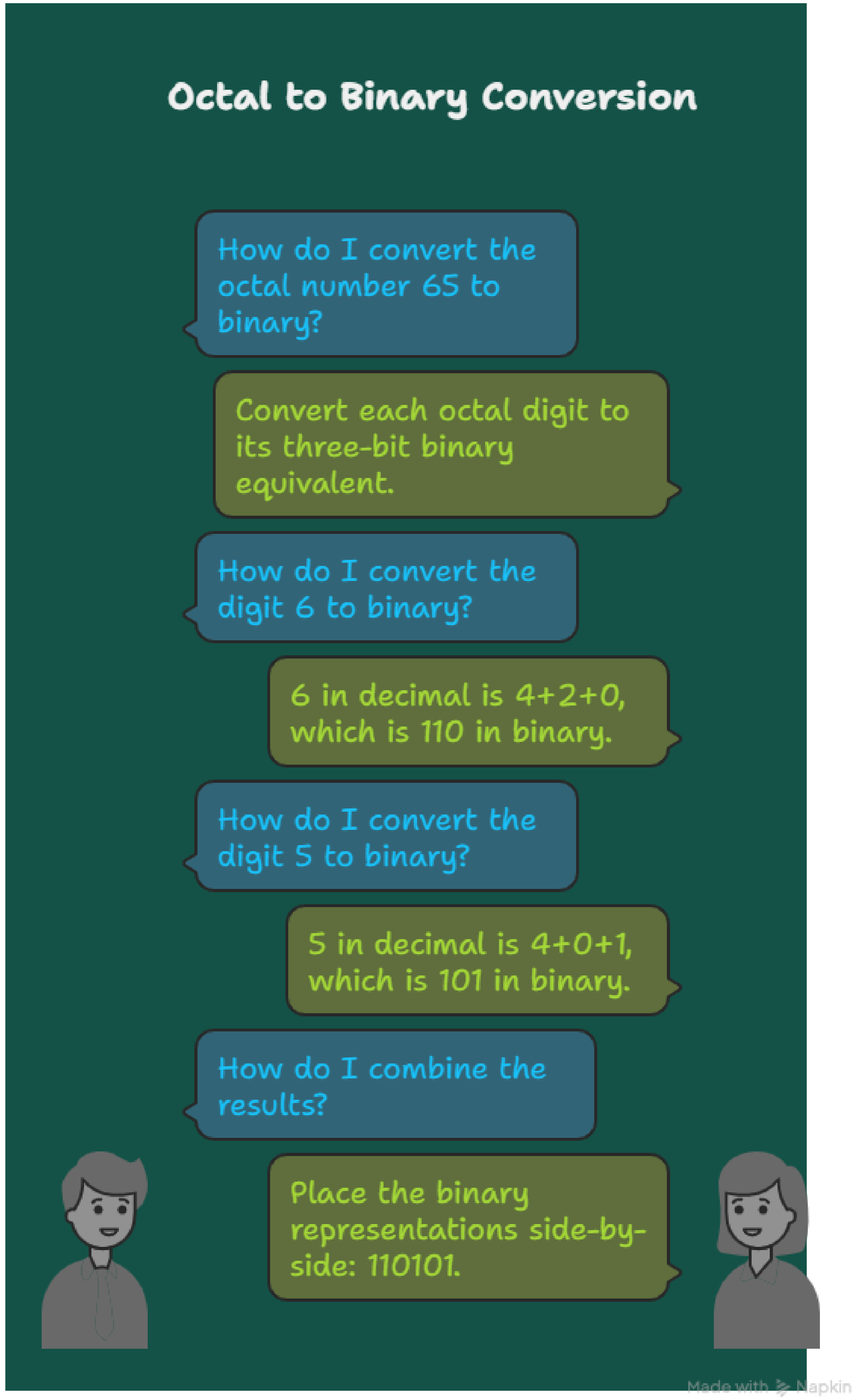
110 101

- For the first group on the right, 101:
  - $(1\times2^2)+(0\times2^1)+(1\times2^0)$
  - $(1\times4)+(0\times2)+(1\times1)$
  - $4+0+1=5$, So, 101 in binary is 5 in octal.
- For the second group, 110:
  - $(1\times2^2)+(1\times2^1)+(0\times2^0)$
  - $(1\times4)+(1\times2)+(0\times1)$
  - $4+2+0=6$
  - So, 110 in binary is 6 in octal.

Therefore, the binary number **110101** is equivalent to **65** in octal.

Octal (Base-8)

- **Octal to Binary**
  - Convert the first digit (6):
  - Convert the second digit (5):
  - Combine the results:



Hexadecimal Number System (Base 16)

- The hexadecimal number system is a **base-16 system** that **uses sixteen digits: 0-9** and **A, B, C, D, E, F.** The letters A through F represent the decimal values 10 through 15.
- Hexadecimal, or "hex," is the most widely used shorthand for binary in modern computing. This is because a single hex digit can represent exactly four bits (2^4=16).
- This aligns perfectly with the common byte size of 8 bits, which can be represented by exactly two hex digits. Hex is essential for low-level programming, debugging, and network communication.

Hexadecimal Number System (Base 16)

- You'll see it everywhere: representing memory addresses, color codes in web design **(e.g., #FF5733)**, and MAC addresses. It makes long binary strings much more human-readable.

- Let's take the same binary string as before: 110001010_2.
- We group the bits in fours from the right: $1\ 1000\ 1010_2$. The leading 1 gets an extra three zeros to make a full group of four: $0001\ 1000\ 1010_2$.
- Converting each group to its hex equivalent: 1 8 A.
- So, $110001010_2 = 18A_{16}$.

Representation of Non-numeric Data

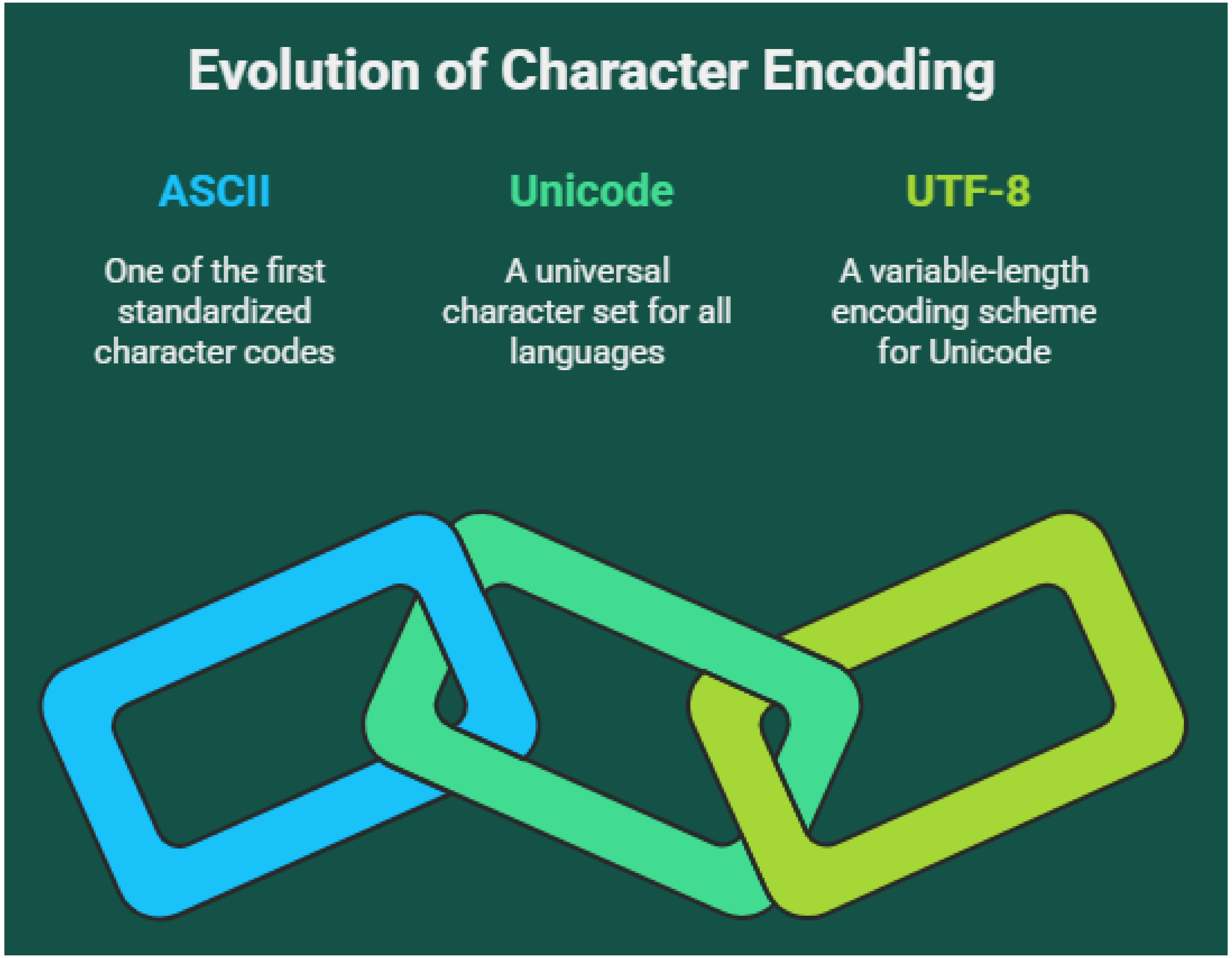
Character Codes

Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

Character Codes

- Character codes are systems that **assign a unique number to each character.** These systems create a **standardized mapping**, allowing computers to represent and exchange text data consistently.
- When you **type a letter on a keyboard**, the hardware doesn't send the letter itself. Instead, it sends the character's numerical code. The computer's software then uses this code to display the correct character on the screen.

Character Codes



Character Codes

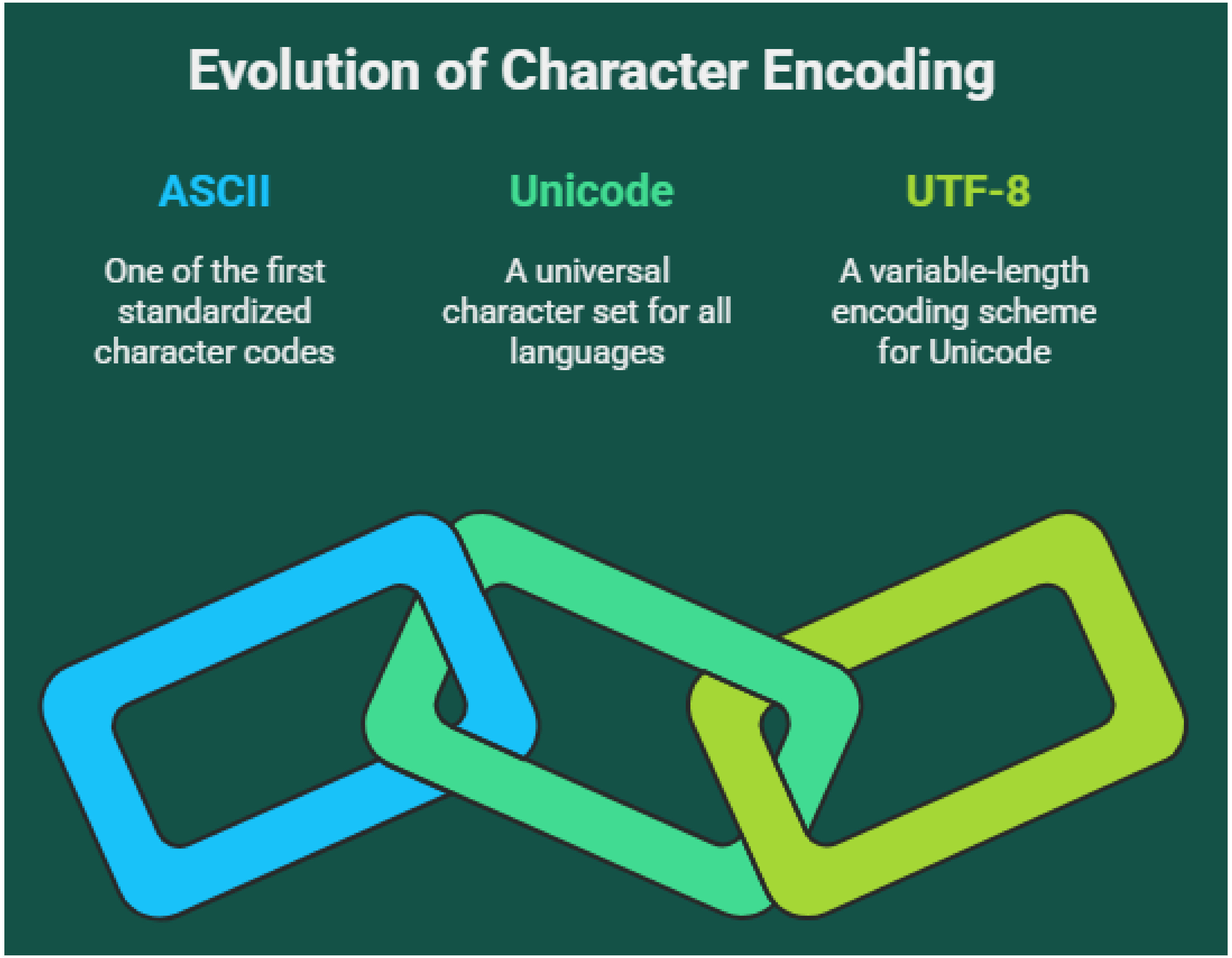
1. **ASCII:** American Standard Code for Information Interchange
   a. Development: **Standardized in 1963**, ASCII was one of the earliest widely adopted character codes.
   b. Capacity: ASCII **uses 7 bits** to represent characters. This gives it a capacity to hold **2^7=128 unique** values. These values are enough to encode English letters (both uppercase and lowercase), numbers, punctuation marks, and some control characters.
   c. Usage: Due to its limited capacity, **ASCII is only suitable for the English language** and has been largely superseded by more modern standards.

Character Codes

1. **ASCII:** American Standard Code for Information Interchange

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SPC	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Character Codes



Made with Napkin

Character Codes

2. Unicode: American Standard Code for Information Interchange

a. Development: Work began in the late **1980s**, with the first version released in 1991.

b. Capacity: Unlike ASCII, **Unicode is a very large character set.** It can use up to 21 bits to represent a character, giving it a theoretical capacity of over a million code points (**221=2,097,152**). This is enough to encode every character from every major language, including a huge variety of symbols, mathematical notations, and emojis.

Character Codes

2. Unicode: it's a portmanteau of "universal" and "code".

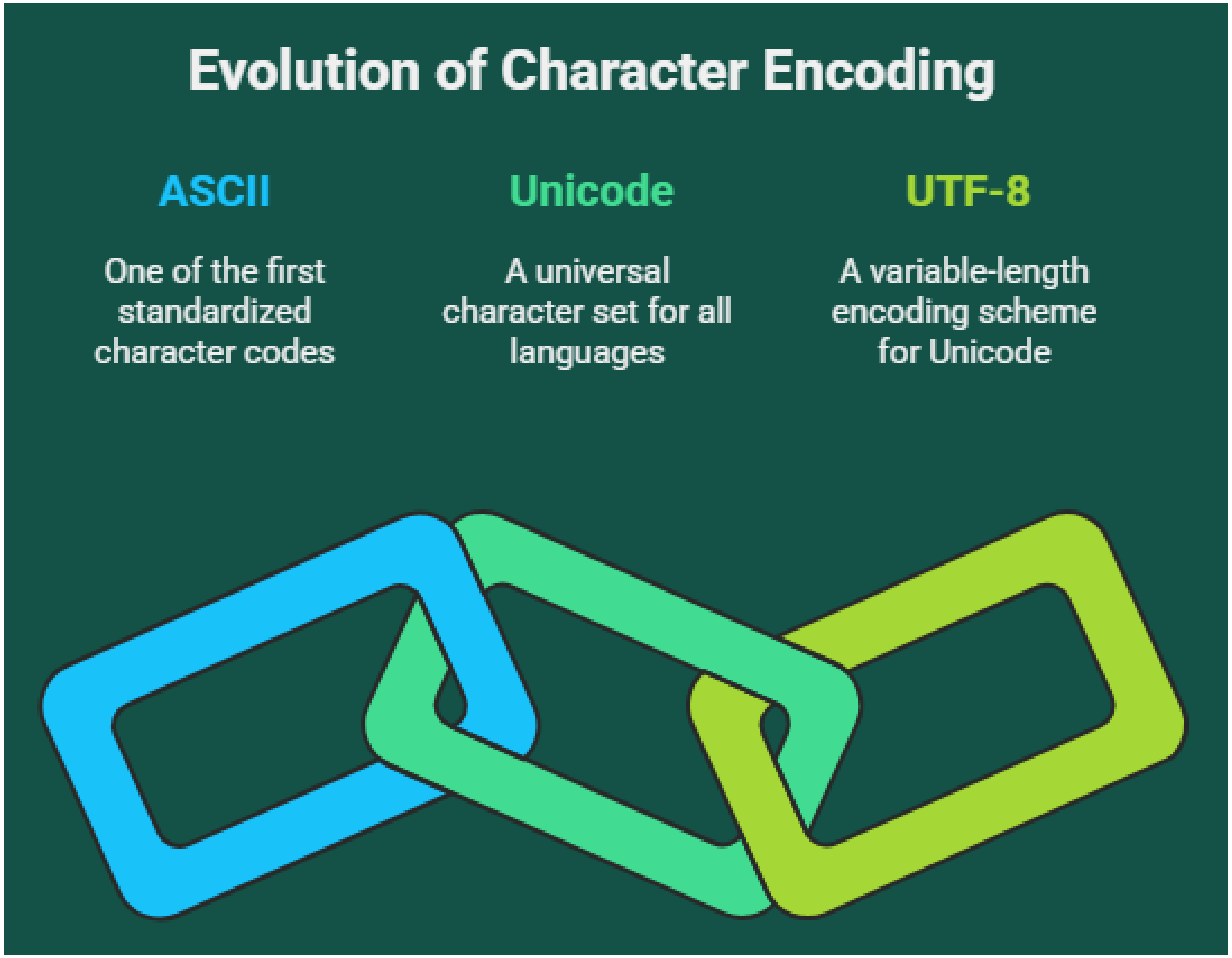
- Unicode is a universal standard that aims to encompass all the world's writing systems. While it defines the code points for characters, **it does not specify how they are stored as bytes.** That's the job of the various UTF encodings.
- Unicode itself is a massive catalog of every character and symbol, assigning each one a unique number called a code point. The reason for Unicode having separate encodings like UTF-8, UTF-16, and UTF-32 is to **provide a balance between efficiency, compatibility, and universality**.

Character Codes

Unicode Table

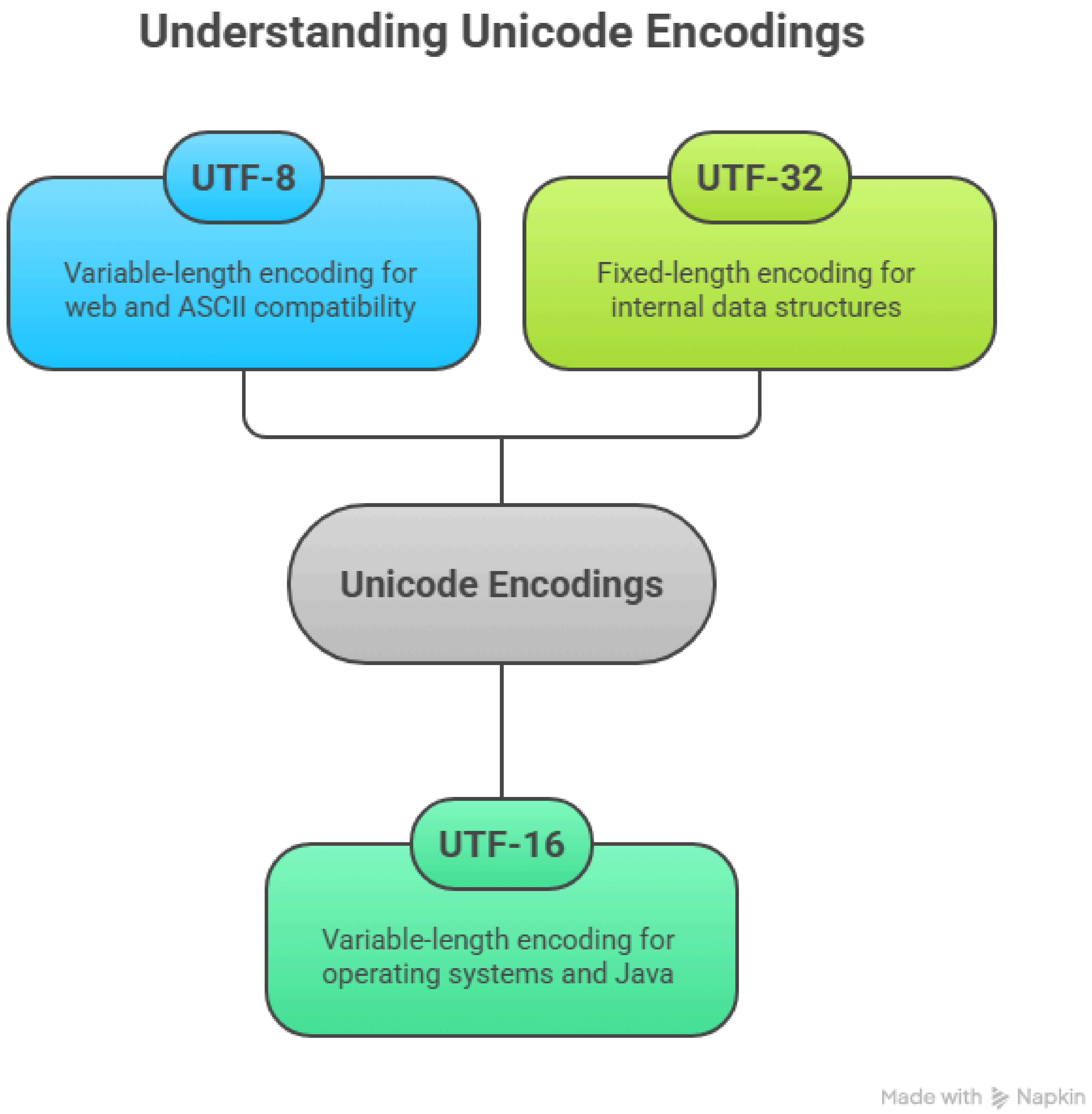
	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	
0000																																	Symbols
0020		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	Number
0040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	Alphabet
0060	`	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	{	|	}	~		
0080	€		‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž			‘	’	“	”	•	–	—	˜	™	š	›	œ		ž	Ÿ	
00A0		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿	
00C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	Latin
00E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	
0100	Ā	ā	Ă	ă	Ą	ą	Ć	ć	Ĉ	ĉ	Ċ	ċ	Č	č	Ď	ď	Đ	đ	Ē	ē	Ĕ	ĕ	Ė	ė	Ę	ę	Ě	ě	Ĝ	ĝ	Ğ	ğ	
0120	Ġ	ġ	Ģ	ģ	Ĥ	ĥ	Ħ	ħ	Ĩ	ĩ	Ī	ī	Ĭ	ĭ	Į	į	İ	ı	Ĳ	ĳ	Ĵ	ĵ	Ķ	ķ	ĸ	Ĺ	ĺ	Ļ	ļ	Ľ	ľ	Ŀ	
0140	ŀ	Ł	ł	Ń	ń	Ņ	ņ	Ň	ň	ŉ	Ŋ	ŋ	Ō	ō	Ŏ	ŏ	Ő	ő	Œ	œ	Ŕ	ŕ	Ŗ	ŗ	Ř	ř	Ś	ś	Ŝ	ŝ	Ş	ş	
0160	Š	š	Ţ	ţ	Ť	ť	Ŧ	ŧ	Ũ	ũ	Ū	ū	Ŭ	ŭ	Ů	ů	Ű	ű	Ų	ų	Ŵ	ŵ	Ŷ	ŷ	Ÿ	Ź	ź	Ż	ż	Ž	ž	ſ	
0180	ƀ	Ɓ	Ƃ	ƃ	Ƅ	ƅ	Ɔ	Ƈ	ƈ	Ɖ	Ɗ	Ƌ	ƌ	ƍ	Ǝ	Ə	Ɛ	Ƒ	ƒ	Ɠ	Ɣ	ƕ	Ɩ	Ɨ	Ƙ	ƙ	ƚ	ƛ	Ɯ	Ɲ	ƞ	Ɵ	
01A0	Ơ	ơ	Ƣ	ƣ	Ƥ	ƥ	Ʀ	Ƨ	ƨ	Ʃ	ƪ	ƫ	Ƭ	ƭ	Ʈ	Ư	ư	Ʊ	Ʋ	Ƴ	ƴ	Ƶ	ƶ	Ʒ	Ƹ	ƹ	ƺ	ƻ	Ƽ	ƽ	ƾ	ƿ	
01C0	ǀ	ǁ	ǂ	ǃ	Ǆ	ǅ	ǆ	Ǉ	ǈ	ǉ	Ǌ	ǋ	ǌ	Ǎ	ǎ	Ǐ	ǐ	Ǒ	ǒ	Ǔ	ǔ	Ǖ	ǖ	Ǘ	ǘ	Ǚ	ǚ	Ǜ	ǜ	ǝ	Ǟ	ǟ	
01E0	Ǡ	ǡ	Ǣ	ǣ	Ǥ	ǥ	Ǧ	ǧ	Ǩ	ǩ	Ǫ	ǫ	Ǭ	ǭ	Ǯ	ǯ	ǰ	Ǳ	ǲ	ǳ	Ǵ	ǵ	Ƕ	Ƿ	Ǹ	ǹ	Ǻ	ǻ	Ǽ	ǽ	Ǿ	ǿ	
0200	Ȁ	ȁ	Ȃ	ȃ	Ȅ	ȅ	Ȇ	ȇ	Ȉ	ȉ	Ȋ	ȋ	Ȍ	ȍ	Ȏ	ȏ	Ȑ	ȑ	Ȓ	ȓ	Ȕ	ȕ	Ȗ	ȗ	Ș	ș	Ț	ț	Ȝ	ȝ	Ȟ	ȟ	
0220	Ƞ	ȡ	Ȣ	ȣ	Ȥ	ȥ	Ȧ	ȧ	Ȩ	ȩ	Ȫ	ȫ	Ȭ	ȭ	Ȯ	ȯ	Ȱ	ȱ	Ȳ	ȳ	ȴ	ȵ	ȶ	ȷ	ȸ	ȹ	Ⱥ	Ȼ	ȼ	Ƚ	Ⱦ	ȿ	

Character Codes



Character Codes

3. UTF and its variants: (Unicode Transformative Format)



Character Codes

3. UTF and its variants: (Unicode Transformative Format)

UTF-8

- It's a variable-length encoding, meaning it uses a different number of bytes to store a character depending on its code point.
- **How it works:**
  - Characters from the original ASCII set (U+0000 to U+007F) are stored using a single byte. This is why **it's backward compatible with ASCII**, a major reason for its popularity.
  - Characters outside of the ASCII range, such as Chinese characters or emojis, are stored using two to four bytes.
- **Example:** The character 'A' (U+0041) is represented in UTF-8 as a single byte: 01000001_2. The Chinese character '中' (U+4E2D) requires three bytes: 11100100_2 10111000_2 10101101_2. This design makes it very efficient for languages that primarily use ASCII characters while still supporting a full range of other languages.

Character Codes

3. UTF and its variants: (Unicode Transformative Format)

UTF-16

- UTF-16 is a **variable-length encoding** that uses at least two bytes (16 bits) per character.
- **How it works:** It was a popular choice for some operating systems (like Windows) and programming languages (like Java) because **it could represent most common characters with a single 16-bit unit.** For rarer characters, it uses a four-byte "surrogate pair."
- **Example:** The character 'A' (U+0041) is represented as two bytes: $00000000\ 01000001_2$. The Chinese character '中' (U+4E2D) is represented as 0100111000101101_2.

Character Codes

3. UTF and its variants: (Unicode Transformative Format)

UTF-16

- UTF-32 is a fixed-length encoding that uses four bytes (32 bits) for every character, regardless of its code point.
- **How it works:** This makes it very fast for a computer to find a specific character in a string, as every character takes up the same amount of space. However, it's very inefficient in terms of memory usage, as every ASCII character that could be stored in one byte now takes up four.
- **Example:** The character 'A' (U+0041) is represented as four bytes: $00000000\ 00000000\ 00000000\ 01000001_2$.

End of Presentation

Questions…?