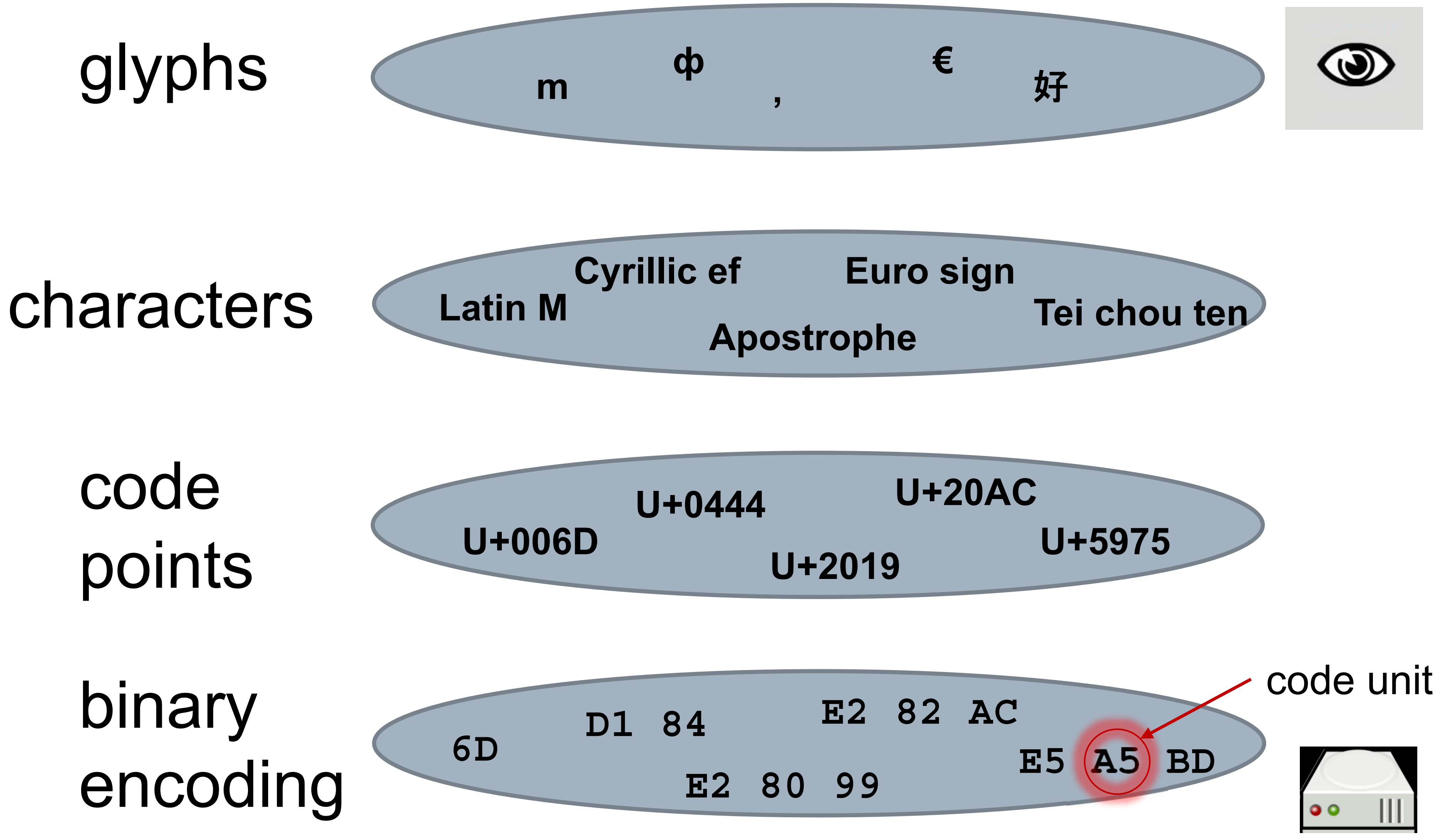
Unicode and UTF-8

Lecture 33

A standard for the discrete representation of written text

The Big Picture



glyphs

m ф ’ € 好

characters

Latin M Cyrillic ef Apostrophe Euro sign Tei chou ten

code points

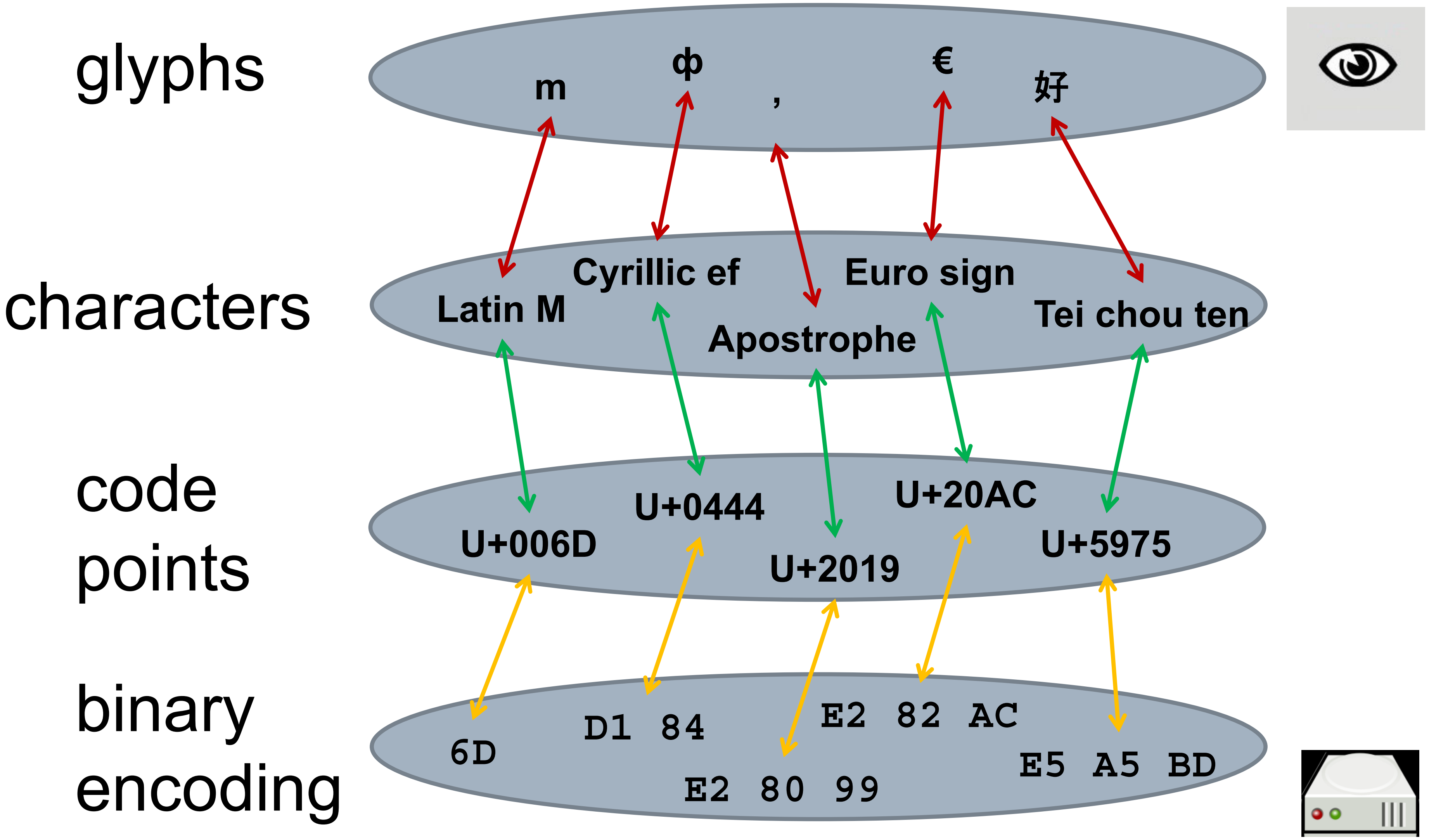
U+006D U+0444 U+2019 U+20AC U+5975

binary encoding

6D D1 84 E2 80 99 E2 82 AC E5 A5 BD

code unit

The Big Picture



glyphs

m ф ’ € 好

characters

Latin M

Cyrillic ef

Apostrophe

Euro sign

Tei chou ten

code points

U+006D

U+0444

U+2019

U+20AC

U+5975

binary encoding

6D

D1 84

E2 80 99

E2 82 AC

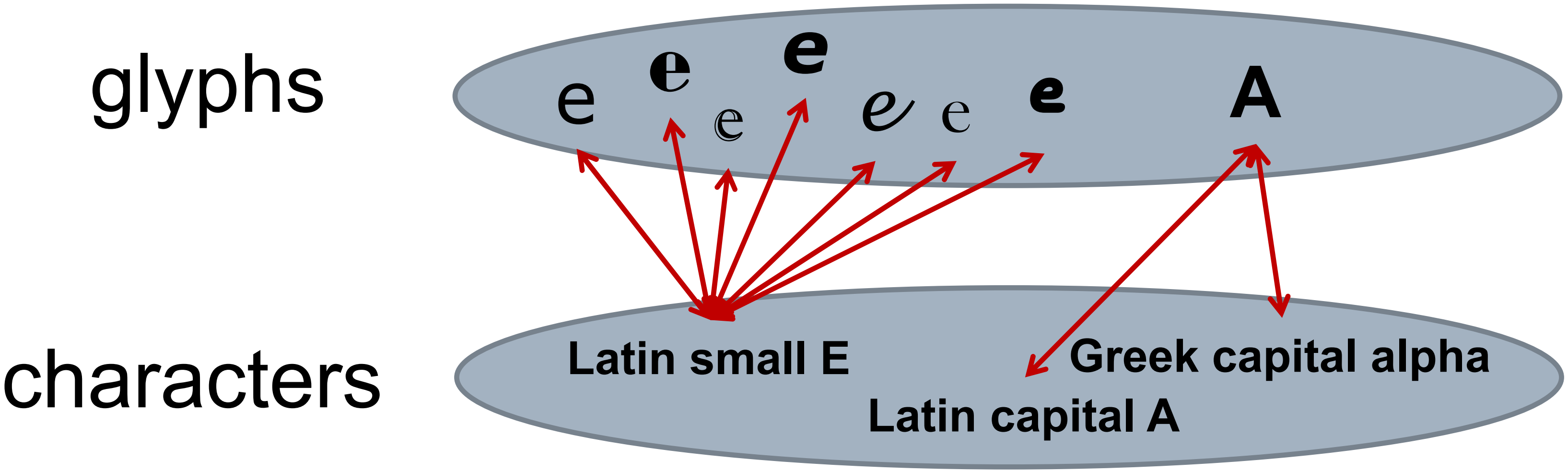
E5 A5 BD

A b C D E F G h I
J K l m N O P
Q R L V
S T U W X y Z

Text: A Sequence of Glyphs

- Glyph: “An individual mark on a written medium that contributes to the meaning of what is written.”
  - See foyer floor in main library
- One *character* can have many *glyphs*
  - Example: Latin E can be e, ***e***, e, **e**, e, **e**, *e*…
- One *glyph* can be different *characters*
  - A is both (capital) Latin A and Greek Alpha
- One unit of text can consist of *multiple* glyphs
  - An accented letter (é) is two glyphs
  - The ligature of f+i (ﬁ) is two glyphs

Glyphs vs Characters



glyphs

characters

Latin small E

Greek capital alpha

Latin capital A

Security Issue

- Visual homograph: Two different characters that look the same
  - Would you click here: www.paypal.com ?

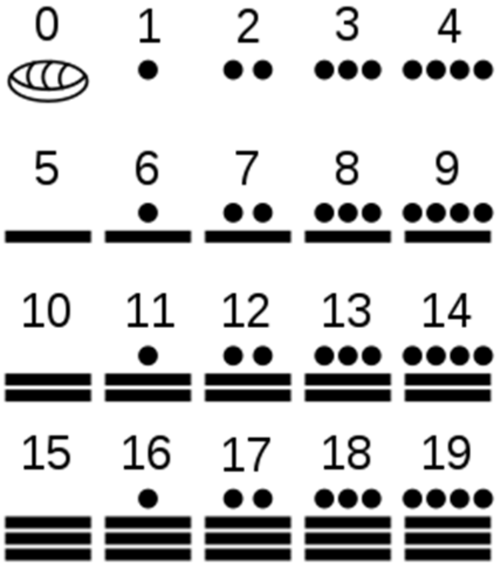
Security Issue

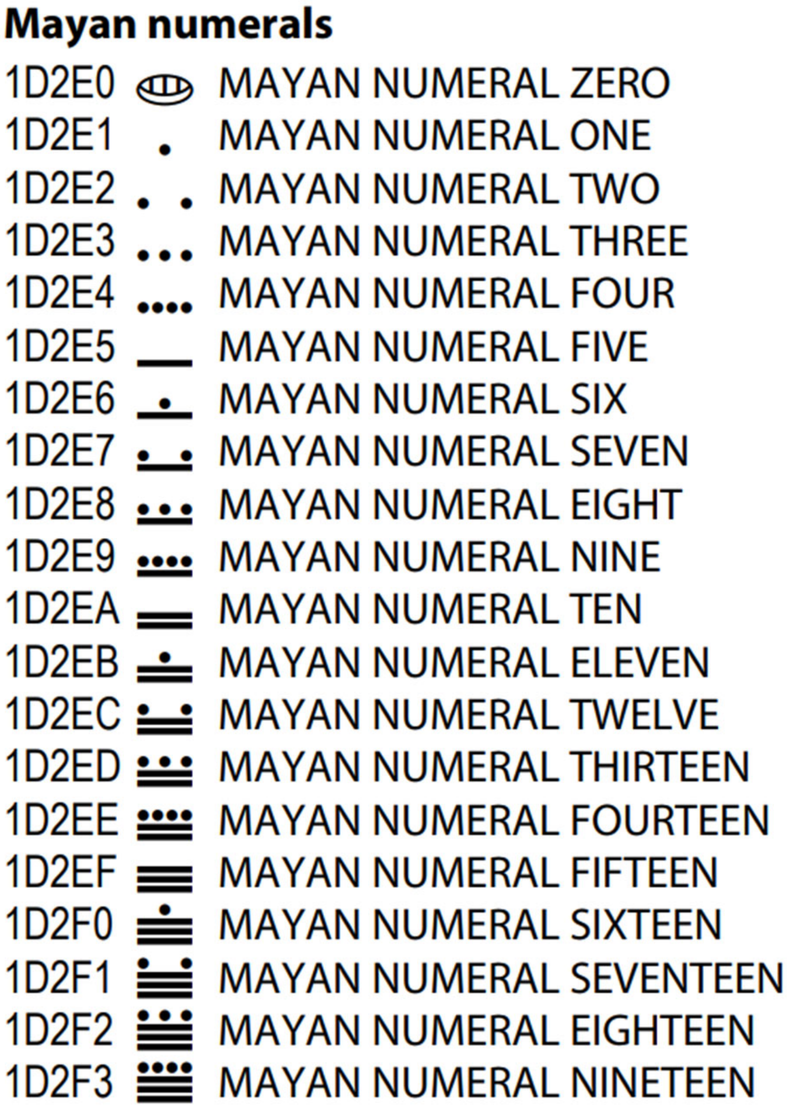
- Visual homograph: Two different characters that look the same
  - Would you click here: www.paypаl.com ?
  - Oops! The second 'a' is actually CYRILLIC SMALL LETTER A
  - This site successfully registered in 2005
- Other examples: combining characters
  - ñ = LATIN SMALL LETTER N WITH TILDE
  - ñ = LATIN SMALL LETTER N + COMBINING TILDE
- "Solution"
  - Heuristics that warn users when languages are mixed and homographs are possible

Unicode Code Points

- Each character is assigned a unique *code point*
- A code point is defined by an integer value, and is also given a name
  - one hundred and nine (109, or 0x6d)
  - LATIN SMALL LETTER M
- Convention: Write code points as U+hex
  - Example: U+006D
- As of March 2020, v13 (see unicode.org):
  - Contains almost 144,000 code points emoji-versions.html#2020
  - Covers 154 scripts (and counting…) unicode.org/charts/

Example Recent Addition (v11)

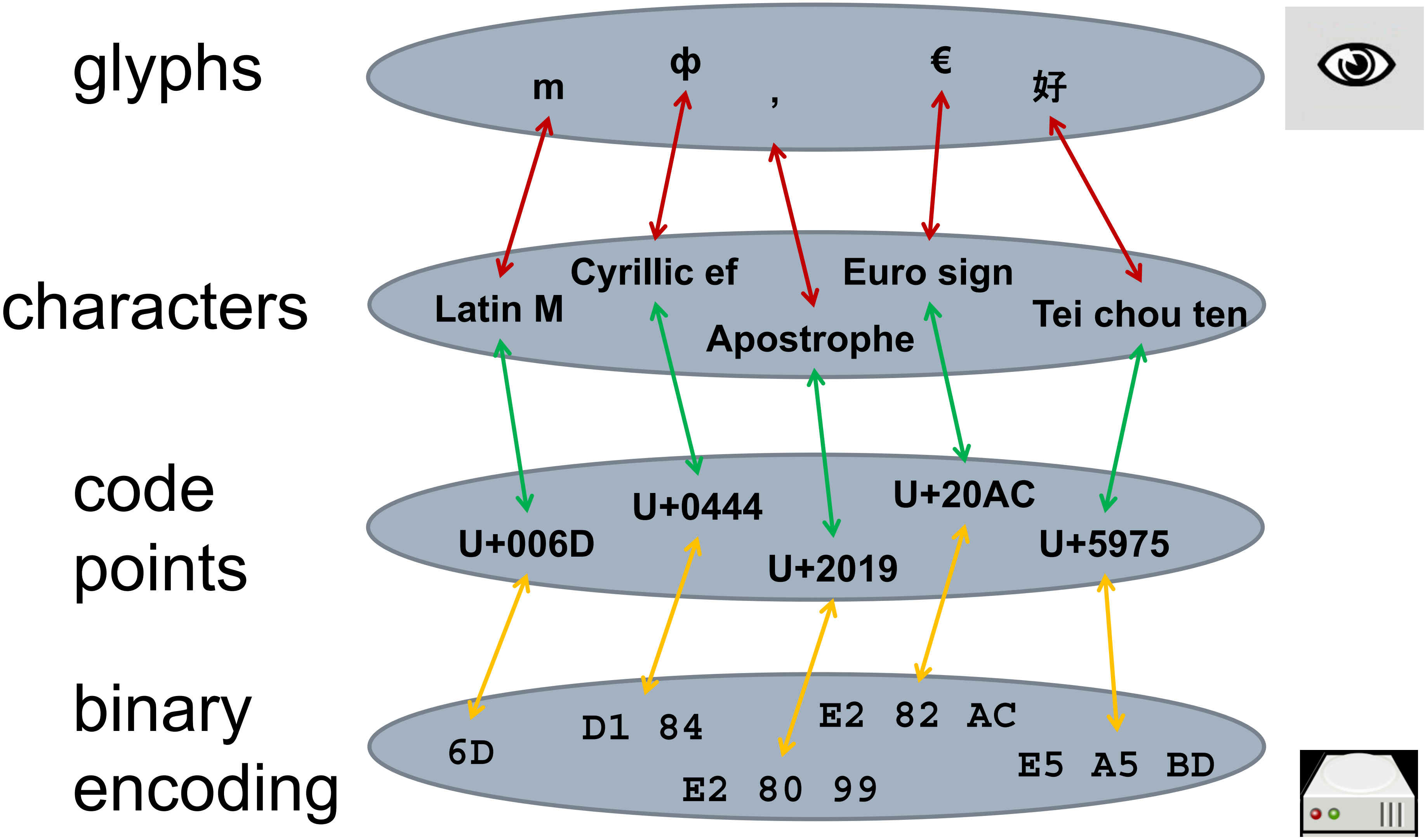




Mayan numerals

1D2E0	𝋠	MAYAN NUMERAL ZERO
1D2E1	𝋡	MAYAN NUMERAL ONE
1D2E2	𝋢	MAYAN NUMERAL TWO
1D2E3	𝋣	MAYAN NUMERAL THREE
1D2E4	𝋤	MAYAN NUMERAL FOUR
1D2E5	𝋥	MAYAN NUMERAL FIVE
1D2E6	𝋦	MAYAN NUMERAL SIX
1D2E7	𝋧	MAYAN NUMERAL SEVEN
1D2E8	𝋨	MAYAN NUMERAL EIGHT
1D2E9	𝋩	MAYAN NUMERAL NINE
1D2EA	𝋪	MAYAN NUMERAL TEN
1D2EB	𝋫	MAYAN NUMERAL ELEVEN
1D2EC	𝋬	MAYAN NUMERAL TWELVE
1D2ED	𝋭	MAYAN NUMERAL THIRTEEN
1D2EE	𝋮	MAYAN NUMERAL FOURTEEN
1D2EF	𝋯	MAYAN NUMERAL FIFTEEN
1D2F0	𝋰	MAYAN NUMERAL SIXTEEN
1D2F1	𝋱	MAYAN NUMERAL SEVENTEEN
1D2F2	𝋲	MAYAN NUMERAL EIGHTEEN
1D2F3	𝋳	MAYAN NUMERAL NINETEEN

Unicode: Mapping to Code Points



glyphs

m ф ’ € 好

characters

Latin M

Cyrillic ef

Apostrophe

Euro sign

Tei chou ten

code points

U+006D

U+0444

U+2019

U+20AC

U+5975

binary encoding

6D

D1 84

E2 80 99

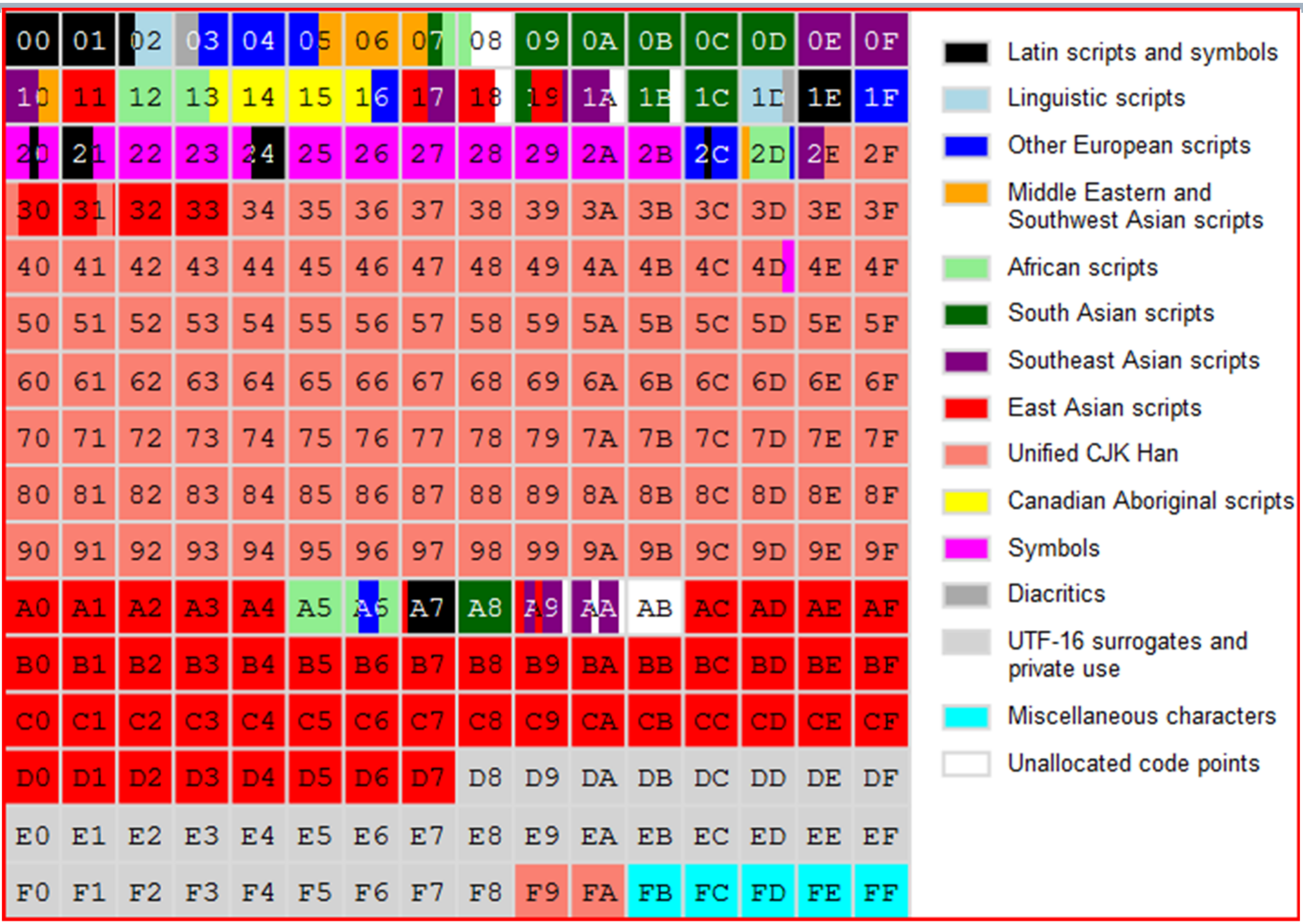
E2 82 AC

E5 A5 BD

Organization

- Code points are grouped into categories
  - Basic Latin, Cyrillic, Arabic, Cherokee, Currency, Mathematical Operators, …
- Standard allows for 17 x 2^{16} code points
  - 0 to 1,114,111 (*i.e.*, > 1 million)
  - U+0000 to U+10FFFF
- Each group of 2^{16} called a *plane*
  - U+nnnnnn, same green ==> same plane
- Plane 0 called *basic multilingual plane* (BMP)
  - Has (practically) everything you could need
  - Convention: code points in BMP written U+nnnn (ie with leading 0's if needed)
  - Others code points written without leading 0's

Basic Multilingual Plane



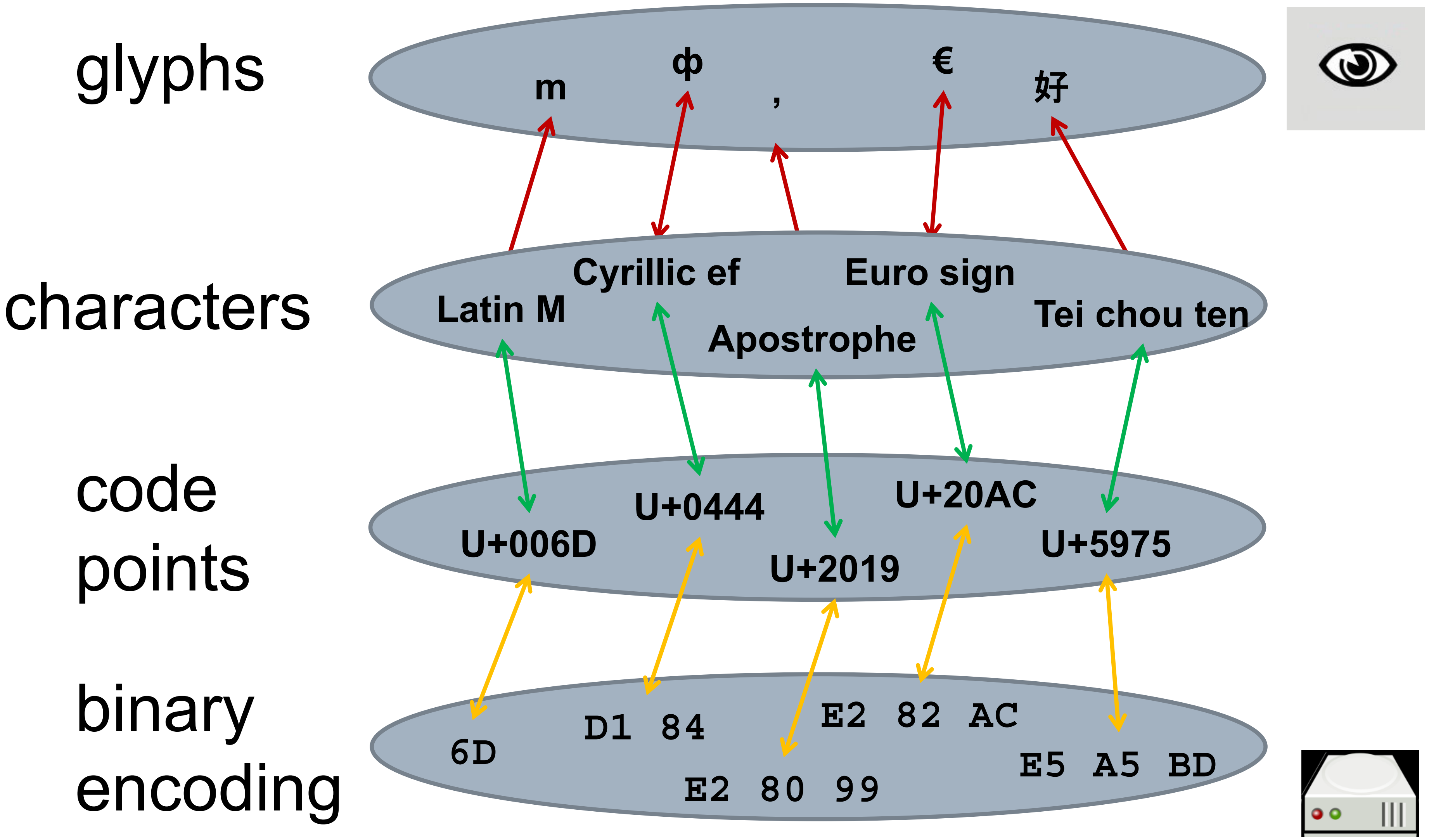
UTF-8

- Encoding of code point (integer) in a sequence of bytes (octets)
  - Standard: all caps, with hyphen (UTF-8)
- Variable length
  - Some code points require 1 octet
  - Others require 2, 3, or 4
- Consequence: Can not infer number of characters from size of file!
- No endian-ness: just a *sequence* of octets

```
D0 BF D1 80 D0 B8 D0 B2 D0 B5 D1 82...
```

- Other encodings might not use 8 bits (more general term: *code unit*)

UTF-8: Code Points & Octets



glyphs	m	ф	’	€	好
characters	Latin M	Cyrillic ef	Apostrophe	Euro sign	Tei chou ten
code points	U+006D	U+0444	U+2019	U+20AC	U+5975
binary encoding	6D	D1 84	E2 80 99	E2 82 AC	E5 A5 BD

UTF-8 Encoding Recipe

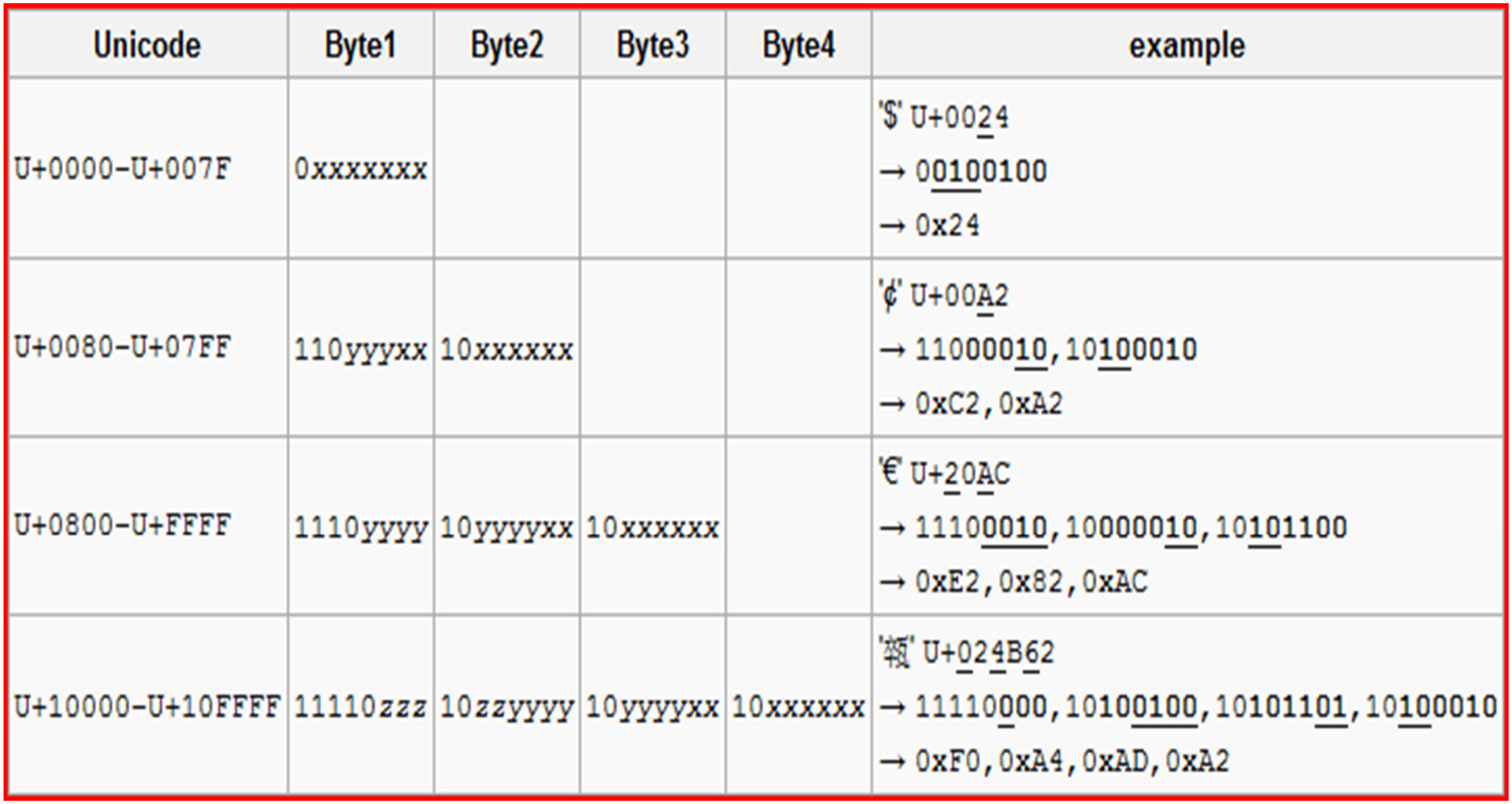
- 1-byte encodings
  - First bit is 0
  - Example: **0**110 1101 (encodes U+006D)
- 2-byte encodings
  - First byte starts with **110**…
  - Second byte starts with **10**…
    - Example: **110**1 0000 **10**11 1111
    - Payload: 110**1 0000** 10**11 1111**
      = 100 0011 1111
      = 0x043F
    - Code point: U+043F
      i.e. п, Cyrillic small letter pe

UTF-8 Encoding Recipe

- Generalization: An encoding of length k:
  - First byte starts with k **1**’s, then **0**
    - Example **1110** 0110 ==> first byte of a 3-byte encoding
  - Subsequent k-1 bytes each start with **10**
  - Remaining bits are payload
- Example: E2 82 AC

  11100010 10000010 10101100
  - Payload: 0x20AC (*i.e.*, U+20AC, €)
- Consequence: Stream is *self-synchronizing*
  - A dropped byte affects only that character

UTF-8 Encoding Summary



Unicode	Byte1	Byte2	Byte3	Byte4	example
U+0000–U+007F	0xxxxxxx				'$' U+0024 → 00100100 → 0x24
U+0080–U+07FF	110yyyxx	10xxxxxx			'¢' U+00A2 → 11000010,10100010 → 0xC2,0xA2
U+0800–U+FFFF	1110yyyy	10yyyyxx	10xxxxxx		'€' U+20AC → 11100010,10000010,10101100 → 0xE2,0x82,0xAC
U+10000–U+10FFFF	11110zzz	10zzyyyy	10yyyyxx	10xxxxxx	'𤭢' U+024B62 → 11110000,10100100,10101101,10100010 → 0xF0,0xA4,0xAD,0xA2

(from wikipedia)

Your Turn

- For the following UTF-8 encoding, what is the corresponding code point(s)?
  - F0 A4 AD A2

- For the following Unicode code point, what is its UTF-8 encoding?
  - U+20AC

Security Issue

- Not all octet sequences are encodings
  - "overlong" encodings are illegal
  - example: C0 AF
    = 1100 0000 1010 1111
    = U+002F (should be encoded 2F)
- Classic security bug (IIS 2001)
  - Should reject URL requests with "../.."
    - Scanned for 2E 2E 2F 2E 2E (in encoding)
  - Accepted "..%c0%af.." (doesn't contain x2F)
    - 2E 2E C0 AF 2E 2E
  - *After* accepting, server *then* decoded
    - 2E 2E C0 AF 2E 2E decoded into "../.."
- Moral: Strings are sequences of "code units"
  - But we think (and see) code points

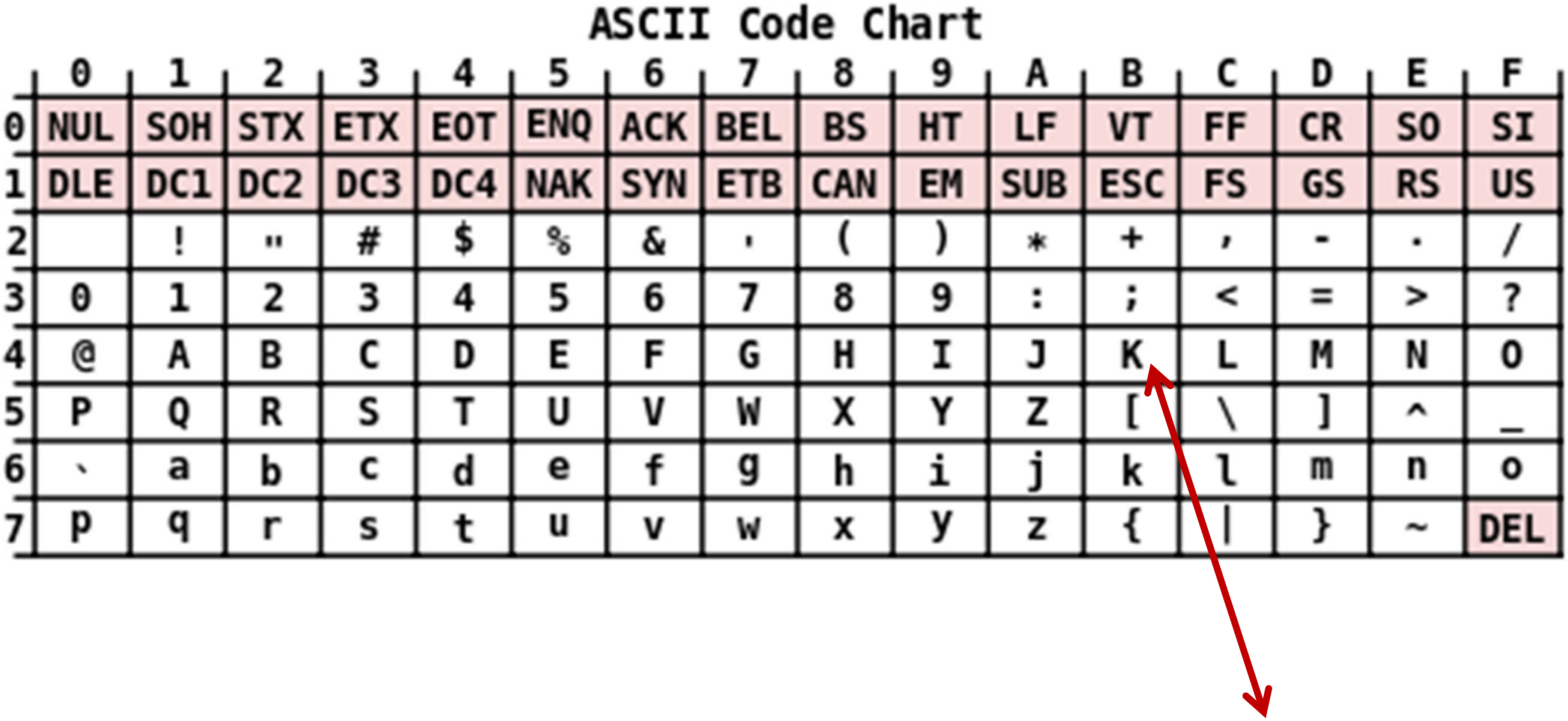
Recall: URL encoding

- Concrete invariant (convention)
  - No space, ;, :, & in representation
  - To represent these characters, use %hh instead (hh is ASCII code in hex)
    - %20 for space
  - Q: What about % in abstract value?
- Recall: *correspondence relation*

Other (Older) Encodings

- In the beginning…
- Character sets were small
  - ASCII: only 128 characters (ie 2^7)
  - 1 byte/character, leading bit always 0
- Globalization means more characters…
  - But 1 byte/character seems fundamental
- Solutions:
  - Use that leading bit!
    - Text data now looks just like binary data
  - Use more than 1 encoding!
    - Must specify data + encoding used

ASCII: 128 Codes

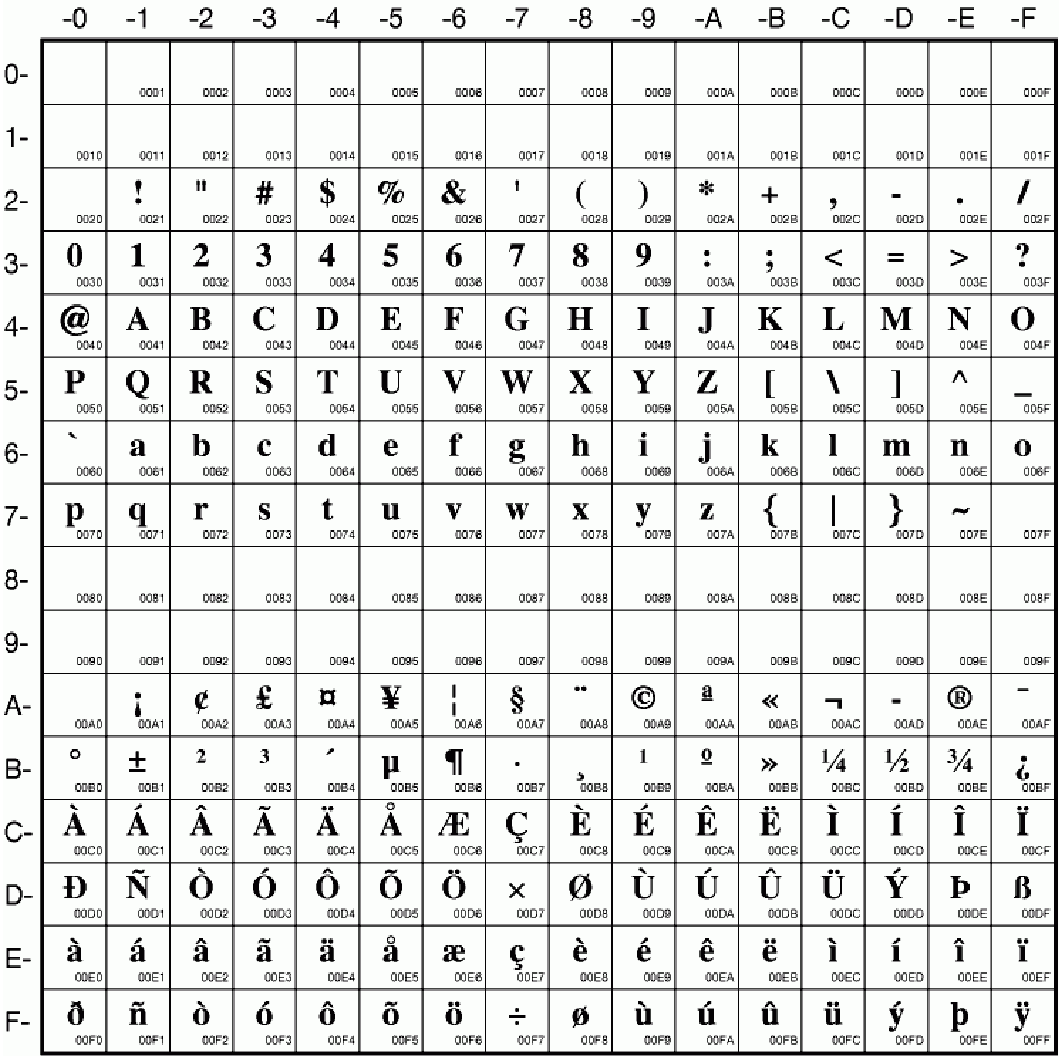


ASCII Code Chart

	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		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
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

4B = Latin capital K

ISO-8859 family (eg -1 Latin)



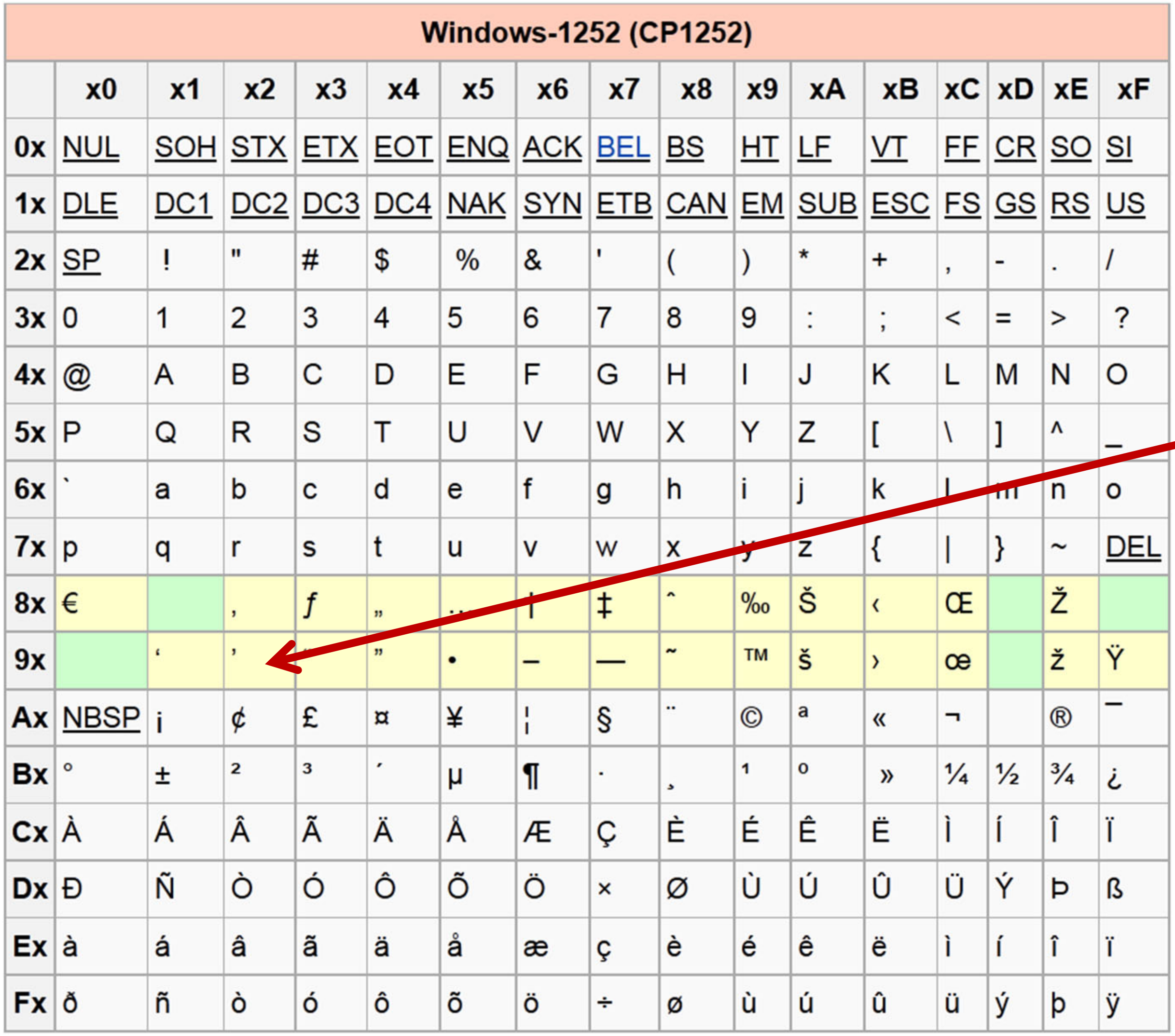
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-		0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
1-	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2-	0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3-	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4-	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5-	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6-	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7-	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	007F
8-	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	008A	008B	008C	008D	008E	008F
9-	0090	0091	0092	0093	0094	0095	0096	0097	0098	0099	009A	009B	009C	009D	009E	009F
A-	00A0	¡ 00A1	¢ 00A2	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	- 00AD	® 00AE	¯ 00AF
B-	° 00B0	± 00B1	² 00B2	³ 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	¹ 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C-	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D-	Ð 00D0	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	Ý 00DD	Þ 00DE	ß 00DF
E-	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F-	ð 00F0	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ý 00FD	þ 00FE	ÿ 00FF

0-7F match ASCII

reserved
(control characters)

A0-FF differ, eg:
-1 "Western"
-2 "East European"
-9 "Turkish

Windows Family (eg 1252 Latin)

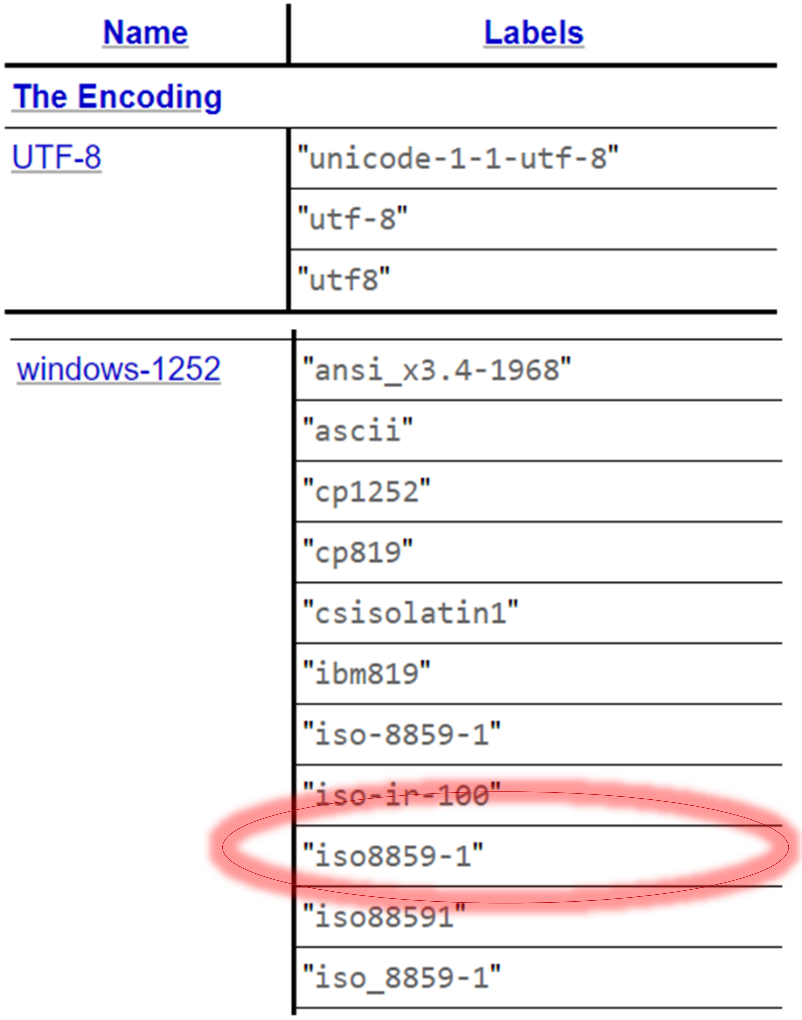


Windows-1252 (CP1252)

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8x	€		‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž	
9x		‘	’	“	”	•	–	—	˜	™	š	›	œ		ž	Ÿ
Ax	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

92 = apostrophe

HTML 5 Standard



Name	Labels
The Encoding	
UTF-8	"unicode-1-1-utf-8"
	"utf-8"
	"utf8"
windows-1252	"ansi_x3.4-1968"
	"ascii"
	"cp1252"
	"cp819"
	"csisolatin1"
	"ibm819"
	"iso-8859-1"
	"iso-ir-100"
	"iso8859-1"
	"iso88591"
	"iso_8859-1"

Early Unicode and UTF-16

- Unicode started as 2^{16} code points
  - The BMP of modern Unicode
  - Bottom 256 code points match ISO-8859-1
- Simple 1:1 encoding (UTF-16)
  - Code point <-> 2-byte code unit (16 bits, 1 word)
  - Simple, but leads to bloat of ASCII text
- Later added code points *outside* of BMP
  - A pair of words (surrogate pairs) carry 20-bit payload split, 10 bits in each word
  - First: **1101 10**xx xxxx xxxx (xD800-DBFF)
  - Second: **1101 11**yy yyyy yyyy (xDC00-DFFF)
- Consequence: U+D800 to U+DFFF became reserved code points in Unicode
  - And now we are stuck with this legacy, even for UTF-8

Demo

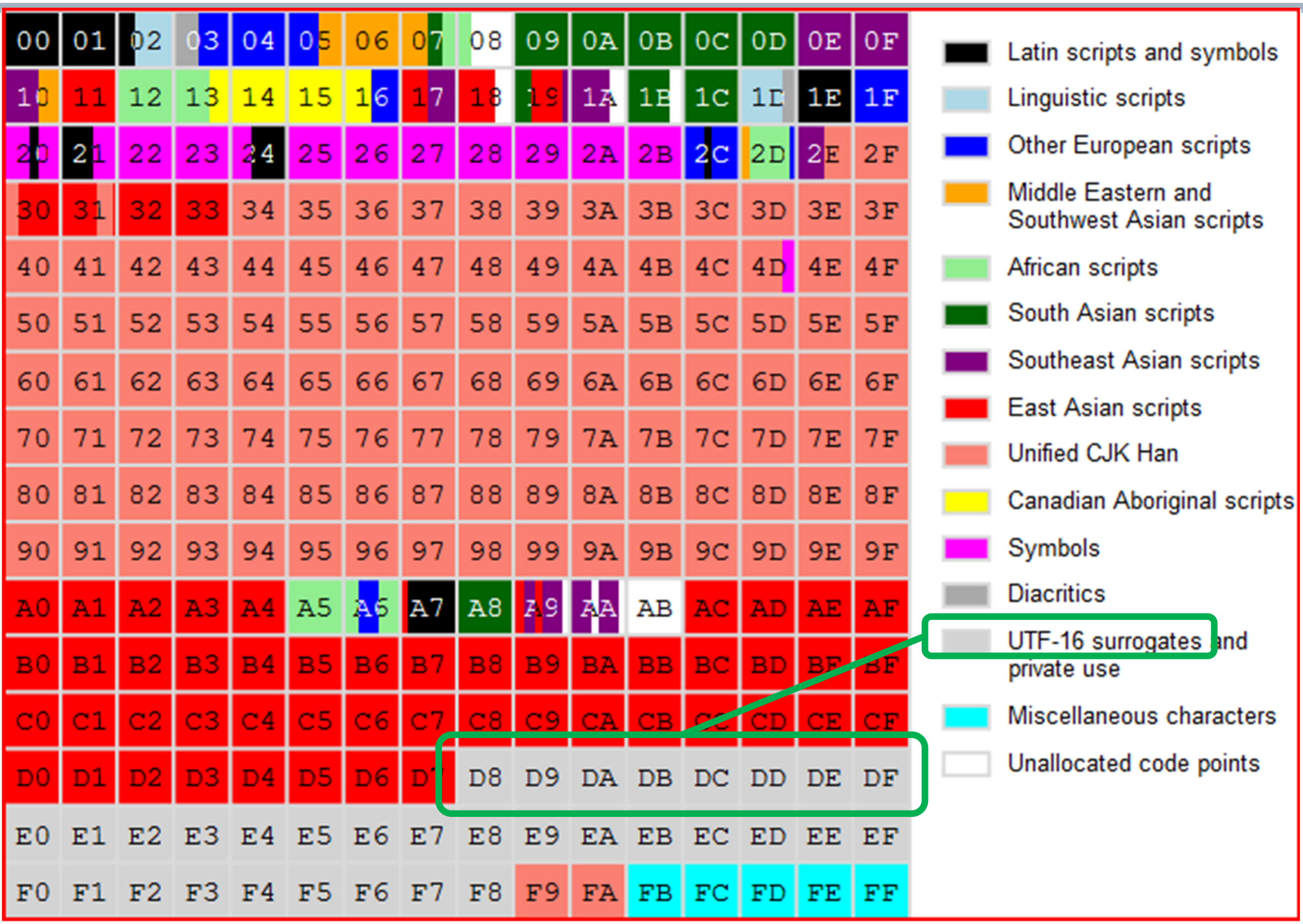
- JavaScript and UTF-16

```
let x = "\u{1f916}" // robot face
x.length
x.charCodeAt(0); x.charCodeAt(1);
x.charAt(0);
x.codePointAt(0);
```

- Ruby and string encodings

```
x = "\u{1f916}”
x.length
x.bytes.map { |b| b.to_s(16) }
x.encoding
x.encode! Encoding::UTF_16
x.bytes.map { |b| b.to_s(16) }
```

Basic Multilingual Plane



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
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

- Latin scripts and symbols
- Linguistic scripts
- Other European scripts
- Middle Eastern and Southwest Asian scripts
- African scripts
- South Asian scripts
- Southeast Asian scripts
- East Asian scripts
- Unified CJK Han
- Canadian Aboriginal scripts
- Symbols
- Diacritics
- UTF-16 surrogates and private use
- Miscellaneous characters
- Unallocated code points

UTF-16 and Endianness

- A multi-byte representation must distinguish between big & little endian
  - Example: 00 25 00 25 00 25
  - "%%%" if LE, "─ ─ ─" if BE
- One solution: Specify encoding in name
  - UTF-16BE or UTF-16LE
- Another solution: require *byte order mark* (BOM) at the start of the file
  - U+FEFF (ZERO WIDTH NO BREAK SPACE)
  - There is *no* U+FFFE code point
  - So FE FF → BigE, while FF FE → LittleE
  - Not considered part of the text

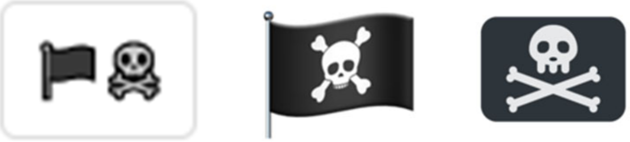
BOM and UTF-8

- Should we add a BOM to the start of UTF-8 files too?
  - UTF-8 encoding of U+FEFF is EF BB BF
- Advantages:
  - Forms magic-number for UTF-8 encoding
- Disadvantages:
  - Not backwards-compatible to ASCII
  - Existing programs may no longer work
  - *E.g.*, In Unix, shebang (#!, *i.e.* 23 21) at *start* of file is significant: file is a script

    ```
    #! /bin/bash
    ```

ZWJ: Zero Width Joiner

- Using U+FEFF as ZWNBSP deprecated
  - Reserved for BOM uses (at start of file)
- Alternative: U+200D (“zwidge”)
- Joined characters may be rendered as a single glyph
  - Co-opted for use with emojis
- Example: (1 “character” in Twitter)
  - U+1F3F4 U+200D U+2620
  - WAVING BLACK FLAG, ZWJ, SKULL AND CROSSBONES



To Ponder

- What is a "text" file? (vs "binary")
  - Given a file, how can you tell which it is?
- A JavaScript program reads in a 5MB file of English prose into a string. How much memory does the string need?
- How many characters does **s** contain?

```
let s = . . . // JavaScript
console.assert (s.length() == 7) // true
```

- Which is better: UTF-8 or UTF-16?
- What's so scary about:

```
..%c0%af..
```

Summary

- Text vs binary
  - In pre-historic times: most significant bit
  - Now: data is data
- Unicode code points
  - Integers U+0000..U+10FFFF
  - BMP: Basic Multilingual Plane
- UTF-8
  - A variable-length, self-synchronizing encoding of unicode code points
  - Backwards compatible with ISO 8859-1, and hence with ASCII too