


THE PROPERTIES AND PROMISES OF UTF-8

by Martin J. Dürst, University of Zurich (1997)

Presented by Marvin Humphrey
for *Papers We Love San Diego*

November 1, 2018


Or...

UTF-8:

What Is All This Ã©Ã¨?!

OVERVIEW

1. Letters as Numbers
2. Rationale for Unicode
3. Problems of UTF-8's competitors
4. UTF-8 solves everything

ASPECTS OF UTF-8

- PRO: ASCII-compatible
- PRO: Not-endian
- PRO: Generally modest space requirements
- PRO: Self-synchronizing
- PRO: Natural sort is code point order
- CON: Variable width
- PRO: Heuristic detectability

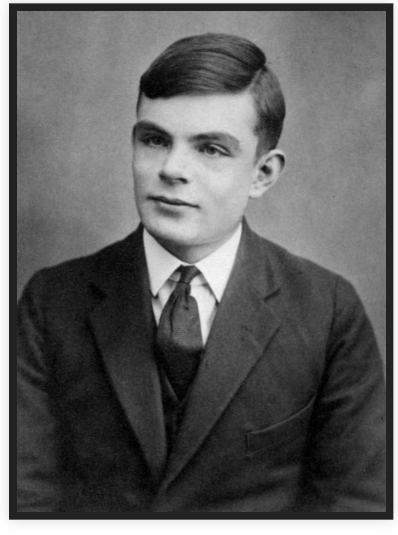
LETTERS AS NUMBERS

$$\texttt{'H'} = 72$$

$$\texttt{'H'}_2 = 01001000$$

ASCII — *American Standard Code for Information Interchange*

LET'S SORT SOME STRINGS



`"Hopper"` `"Turing"`

STRINGS AS NUMBERS

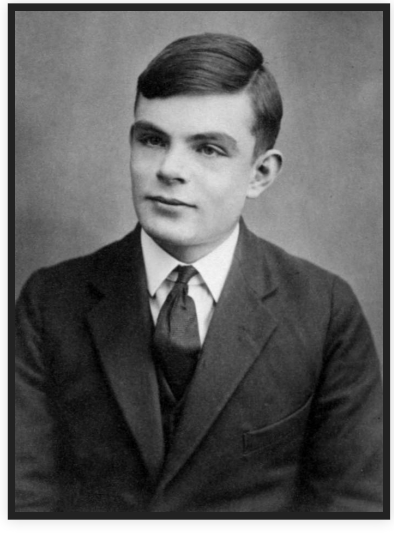
```
H        o        p        p        e        r
01001000 01101111 01110000 01110000 01100101 01110010

T        u        r        i        n        g
01010100 01110101 01110010 01101001 01101110 01100111
```

- "Hopper" = 796434649839 22
- "Turing" = 92863407418983
- "Hopper"$_2$ = 01001000 [...]
- "Turing"$_2$ = 01010100 [...]

LET'S SORT MORE STRINGS

"Hopper"



"Turing"

"Gödel"

```
H        o        p        p        e        r
01001000 01101111 01110000 01110000 01100101 01110010

T        u        r        i        n        g
01010100 01110101 01110010 01101001 01101110 01100111

G        ö        d        e        l
01000111 ???????? 01100100 01100101 01101100
```

ö = ????????

can't be represented using ASCII

But ASCII only defines 2^7 values...
out of 2^8 in a byte.

Solution: ISO-8859-1, a.k.a. *Latin-1*

```
H        o        p        p        e        r
01001000 01101111 01110000 01110000 01100101 01110010

T        u        r        i        n        g
01010100 01110101 01110010 01101001 01101110 01100111

G        ö        d        e        l
01000111 11110110 01100100 01100101 01101100 00000000
```

- Hopper = 796434649839**22**
- Turing = 92863407418983
- ~~Gödel = 309076452716~~
- Gödel = 791235718952**96**

MOAR STRINGS!

"Ле́бедев"

"خوارزمی"

```
H        o        p        p        e        r
01001000 01101111 01110000 01110000 01100101 01110010

T        u        r        i        n        g
01010100 01110101 01110010 01101001 01101110 01100111

G        ö        d        e        l
01000111 11110110 01100100 01100101 01101100

Л        é        б        е        д        е        в
???????? ???????? ???????? ???????? ???????? ???????? ????????

خ        و        ا        ر        ز        م        ى
???????? ???????? ???????? ???????? ???????? ???????? ????????
```

can't fit everything into 8 bits

MULTIPLE 8-BIT ENCODINGS

- "Hopper" — ISO-8859-1 Latin-1
- "Turing" — ISO-8859-1 Latin-1
- "Gödel" — ISO-8859-1 Latin-1
- "Ле́бедев" — ISO-8859-5 Cyrillic
- "خوارزمی" — ISO-8859-6 Arabic

How about a bigger bucket?

UNICODE!

*A **16-bit** universal code.*

- UNIversal — all contemporary languages
- UNIform — fixed-width
- UNIque — unambiguous byte patterns

THE UCS-2 ENCODING

```
H                   o                   p                   ...
00000000 01001000 00000000 01101111 00000000 01110000 ...

T                   u                   r                   ...
00000000 01010100 00000000 01110101 00000000 01110010 ...

G                   ö                   d                   ...
00000000 01000111 00000000 11110110 00000000 01100100 ...

Л                   é                   б                   ...
00000100 00011011 00000100 01010001 00000100 00110001 ...

ز                   م                   ى                   ...
00000110 11001100 00000110 01000101 00000110 00110010 ...
```

(big-endian)

THE UCS-2 ENCODING

```
H                   o                   p                   ...
01001000 00000000 01101111 00000000 01110000 00000000 ...

T                   u                   r                   ...
01010100 00000000 01110101 00000000 01110010 00000000 ...

G                   ö                   d                   ...
01000111 00000000 11110110 00000000 01100100 00000000 ...

Л                   é                   б                   ...
00011011 00000100 01010001 00000100 00110001 00000100 ...

ز                   م                   ى                   ...
11001100 00000110 01000101 00000110 00110010 00000110 ...
```

(little-endian)

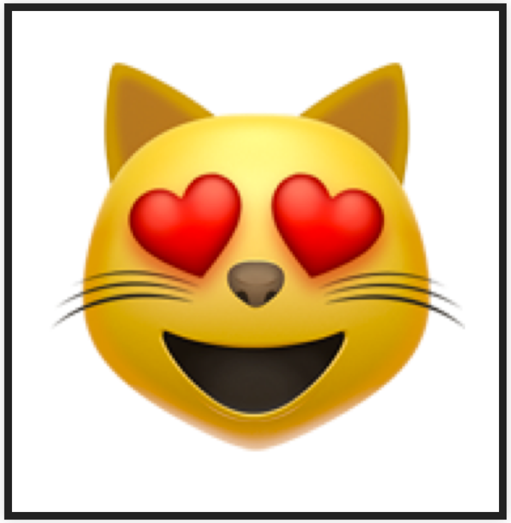
BYTE ORDER MARK

`11111111 11111110`

If you see `11111110 11111111`:
flip every byte pair.

[Introduced in Unicode 1.1 (1993)]

STILL MOAR STRINGS!!!!



"𐎤𐎢𐎽𐎢𐏁"

"😻"

```
H                   o                   p                   ...
00000000 01001000 00000000 01101111 00000000 01110000 ...

[...]

𐌗                   𐌘                   𐌟                   ...
???????? ???????? ???????? ???????? ???????? ???????? ...

😻
???????? ????????
```

can't fit everything into 16 bits

UCS-4 (SAME AS UTF-32)

```
H                                                   o
00000000 00000000 00000000 01001000 00000000 [...]

[...]

ᚲ                                                   ᚷ
00000000 00000001 00000011 10100100 00000000 [...]

😻
00000000 00000001 11110110 00111011
```

UNICODE 2.0

- Published 1996
- **Abandons fixed-width 16-bit**
- Adds *surrogate* mechanism

ASPECTS OF UCS-2

- CON: not ASCII-compatible
- PRO: fixed-width
- PRO: natural sort is code point order
- CON: wastes (some) space
- CON: needs Byte Order Mark
- PRO: self-synchronizing
- CON: **can't express all of Unicode 2.0**

ASPECTS OF UTF-16

- CON: not ASCII-compatible
- CON: variable-width
- CON: natural sort **not** code point order
- CON: wastes (some) space
- CON: needs Byte Order Mark
- PRO: self-synchronizing
- PRO: expresses all of Unicode 2.0

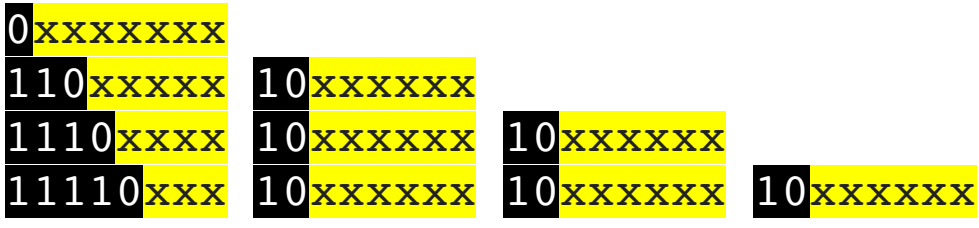
ASPECTS OF UCS-4 / UTF-32

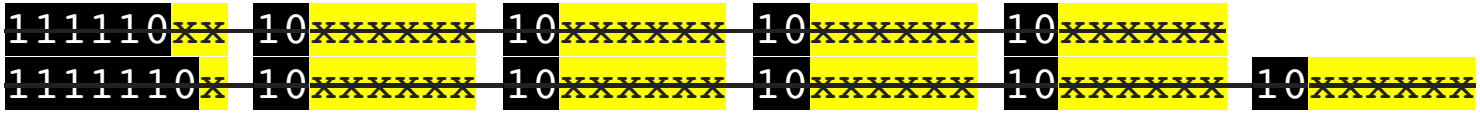
- CON: not ASCII-compatible
- PRO: fixed-width
- PRO: natural sort is code point order
- CON: wastes lots of space
- CON: needs Byte Order Mark
- PRO: expresses all of Unicode 2.0

LET'S GO BACK IN TIME

Scene: A New Jersey Diner in September 1992

UTF-8 BIT PATTERNS






```
11111110 [...] illegal
11111111 [...] illegal
```

- Black — classifier bits
- Yellow — payload bits

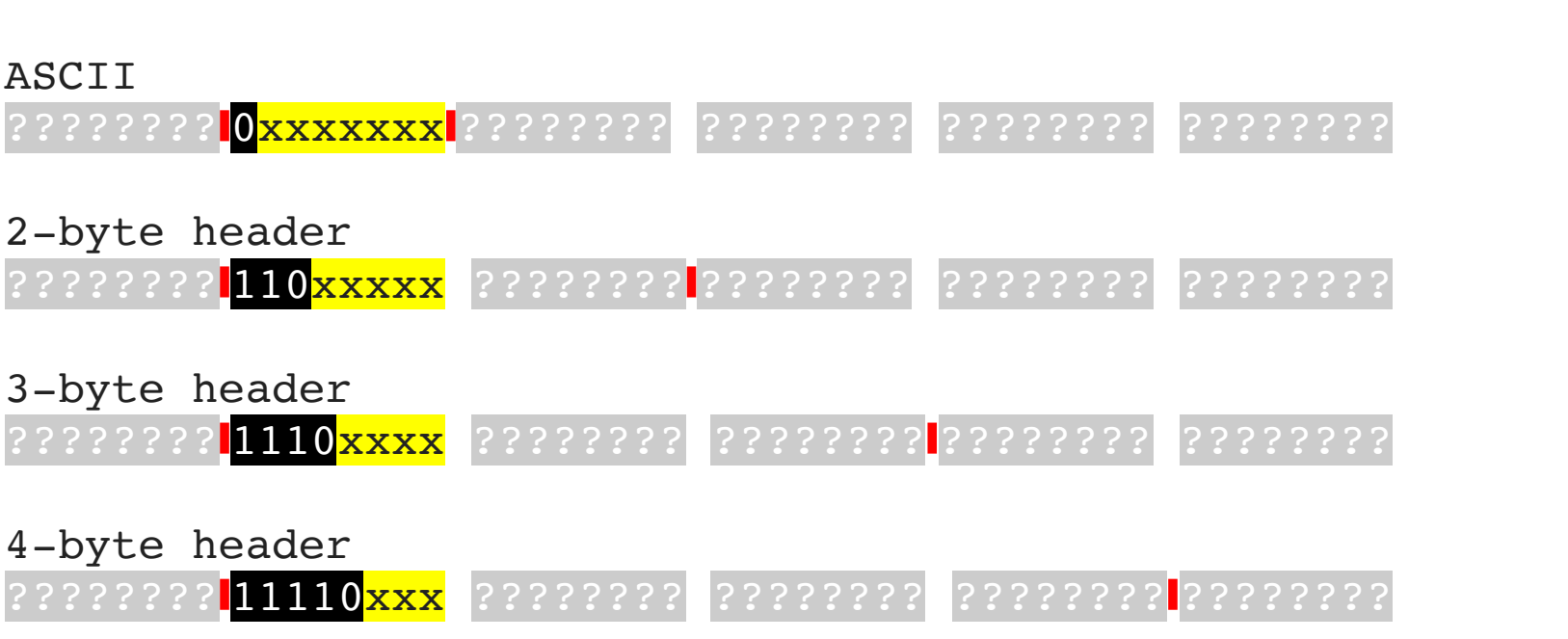
UTF-8 BYTE ROLES

- `0xxxxxxx` — ASCII
- `10xxxxxx` — continuation byte
- `110xxxxx` — header for 2-byte sequence
- `1110xxxx` — header for 3-byte sequence
- `11110xxx` — header for 4-byte sequence
- `111110xx` — illegal (as of 2003)
- `1111110x` — illegal (as of 2003)
- `11111110` — illegal
- `11111111` — illegal

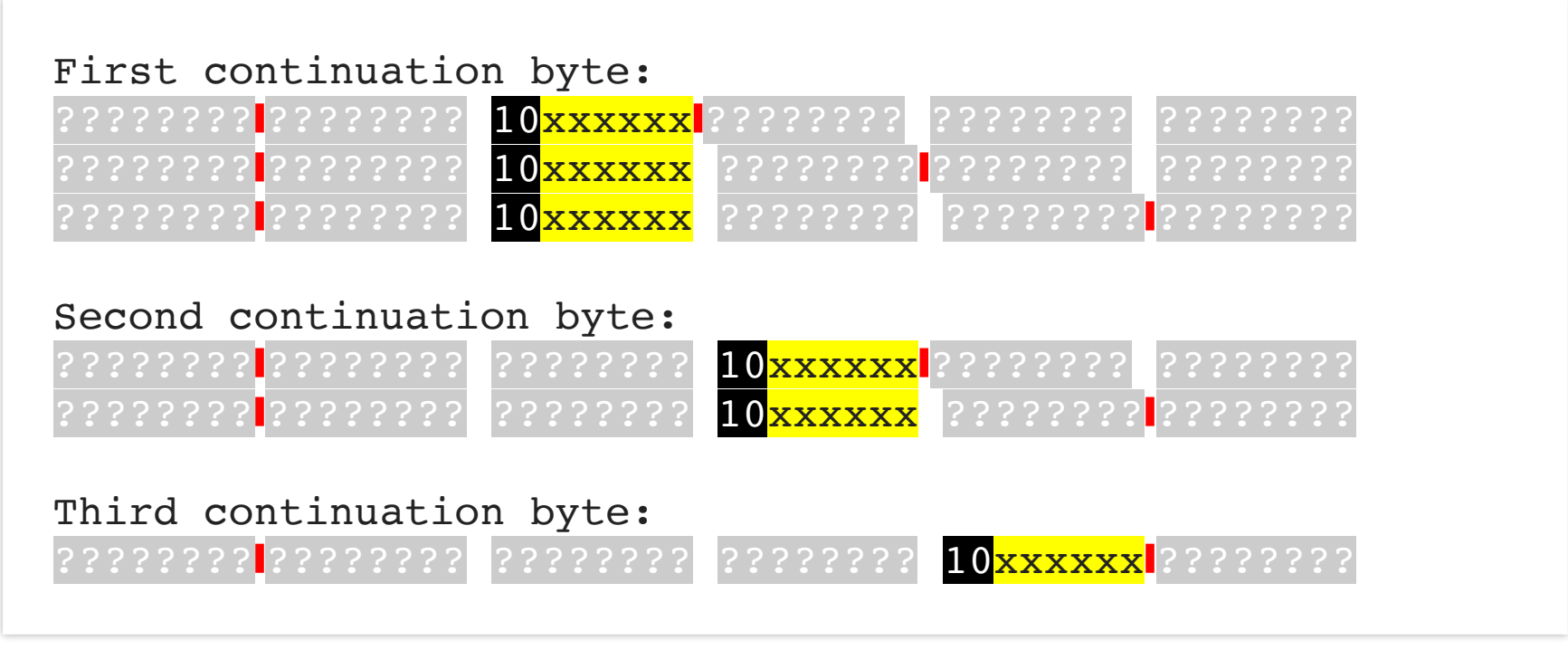
SELF-SYNCHRONIZATION

- boundary detection always possible
- max 3 bytes forward/back

SELF-SYNCH: ASCII/HEADER



SELF-SYNCH: CONTINUATION



ASCII COMPATIBILITY

- UTF-8 in ASCII range: identical to ASCII
- NUL byte always means NUL
- UTF-8 compatible with legacy APIs

ASCII COMPAT COMPARISON

"Hi" as a NUL-terminated string:

ASCII

```
H        i        [NUL]
01001000 01101001 00000000
```

UTF-8

```
H        i        [NUL]
01001000 01101001 00000000
```

UTF-16BE

```
H                 i                   [NUL]
00000000 01001000 00000000 01101001 00000000 00000000
```

UTF-16LE

```
H                 i                   [NUL]
01001000 00000000 01101001 00000000 00000000 00000000
```

NOT-ENDIAN

- No big-endian and little-endian variants
- No Byte Order Mark needed

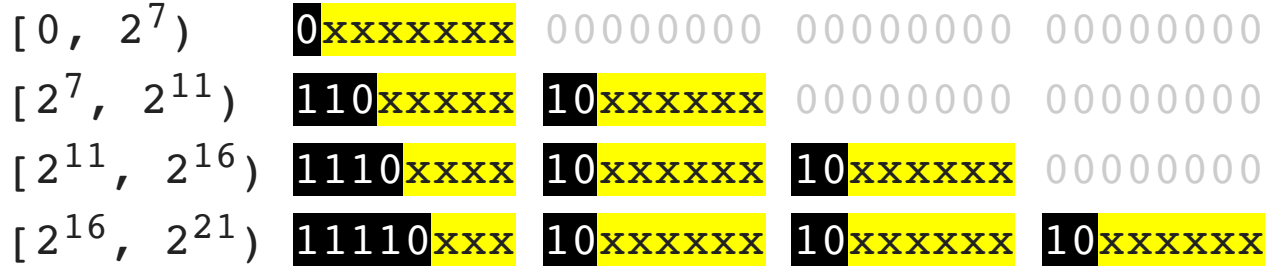
NOT-ENDIAN COMPARISON

UTF-8
Ö
11000011 10110110

UTF-16BE
Ö
00000000 11110110

UTF-16LE
Ö
11110110 00000000

SORTING



- ASCII < 2-byte < 3-byte < 4-byte
- All payloads sort in code point order
- *Therefore, natural sort is code point order!*

SPACE REQUIREMENTS

- By source:
  - ASCII: excellent
  - Cyrillic: same as UTF-16
  - CJK: 1.5x UTF-16
  - Tibetan, Georgian: bad
- HTML with ASCII markup: excellent

UTF-8 wins as web encoding

VARIABLE WIDTH

Accessing *n*th char is *O(n)*, not *O(1)*

A problem for certain programming language idioms

HEURISTIC DETECTABILITY

Q. What Is All This Ã©Ã¨?!

A. "Double-encoded" UTF-8.

- "Ã©" is "é"
- "Ã¨" is "è"
- to a very, very high likelihood!

Similarly, UTF-8 is easy to detect

ASPECTS OF UTF-8

- PRO: ASCII-compatible
- PRO: Not-endian
- PRO: Generally modest space requirements
- PRO: Self-synchronizing
- PRO: Natural sort is code point order
- CON: Variable width
- PRO: Heuristic detectability

UTF-8: OPTIMAL, ELEGANT

- Design seems simple, even obvious
- But competitors riddled with defects
- UTF-7, UTF-1 worse than UTF-16

How Did Ken Thompson pull it off?

Thank you!

Questions?