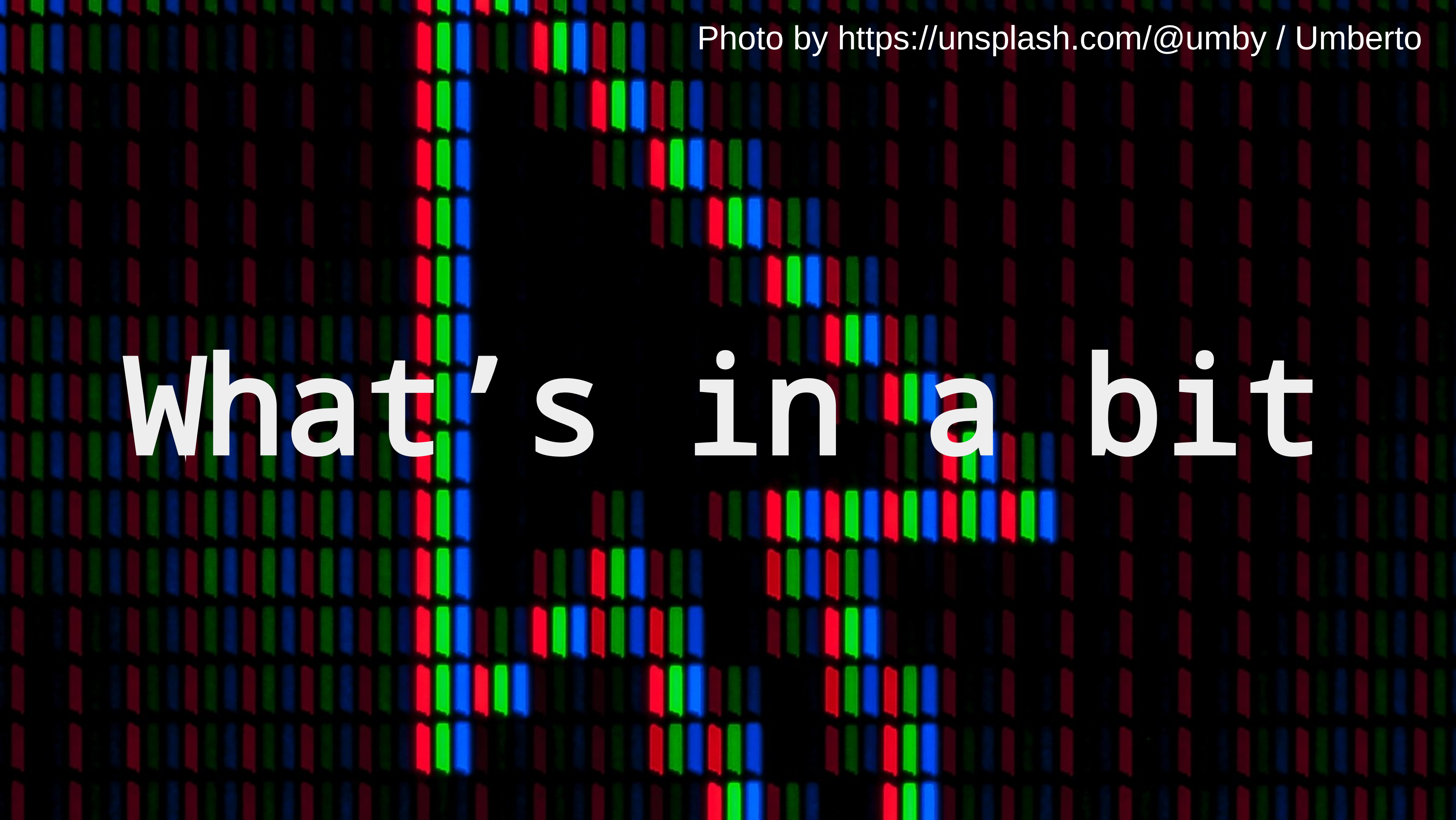


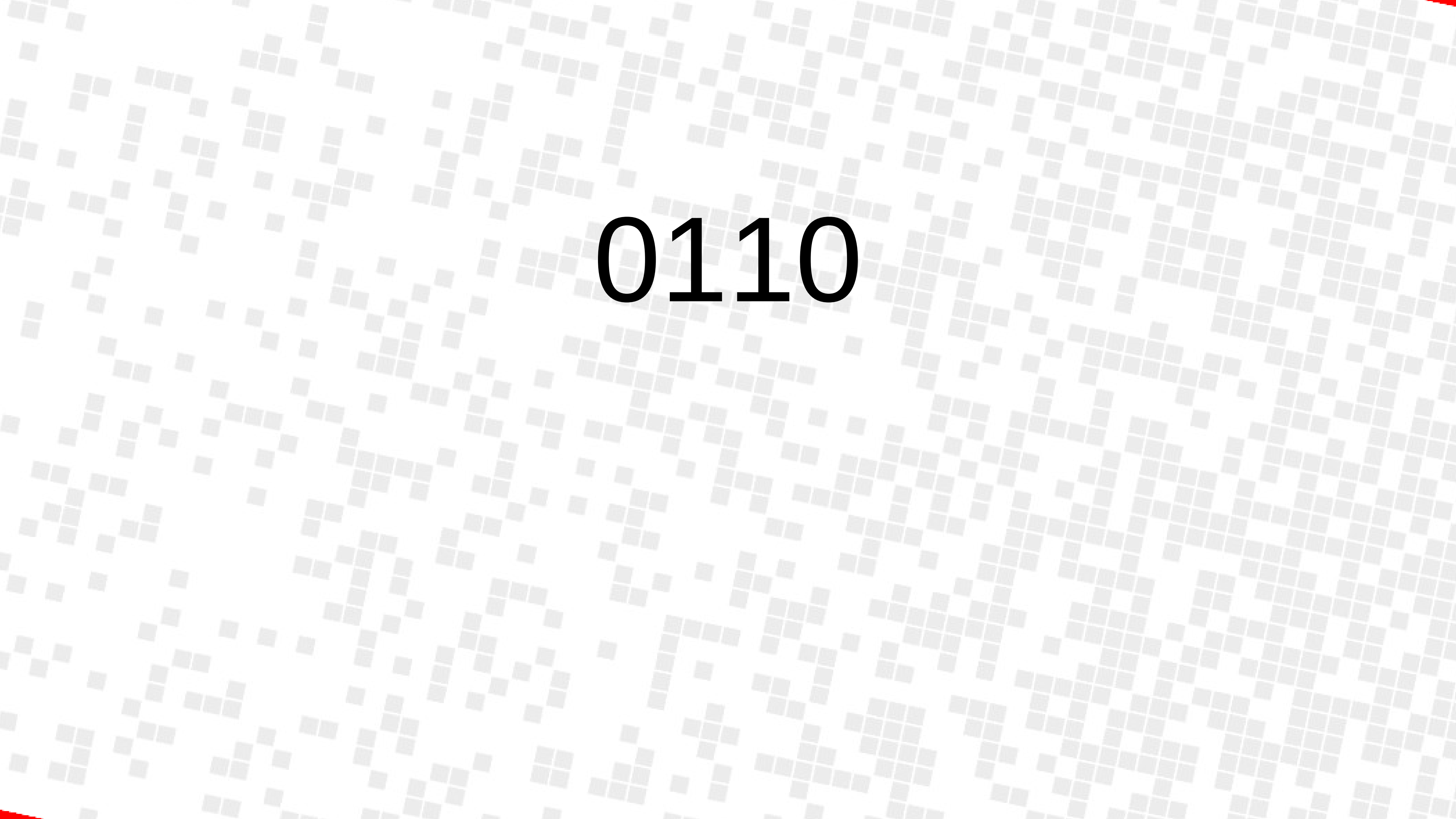
Photo by https://unsplash.com/@umby / Umberto
What's in a bit

Talk overview

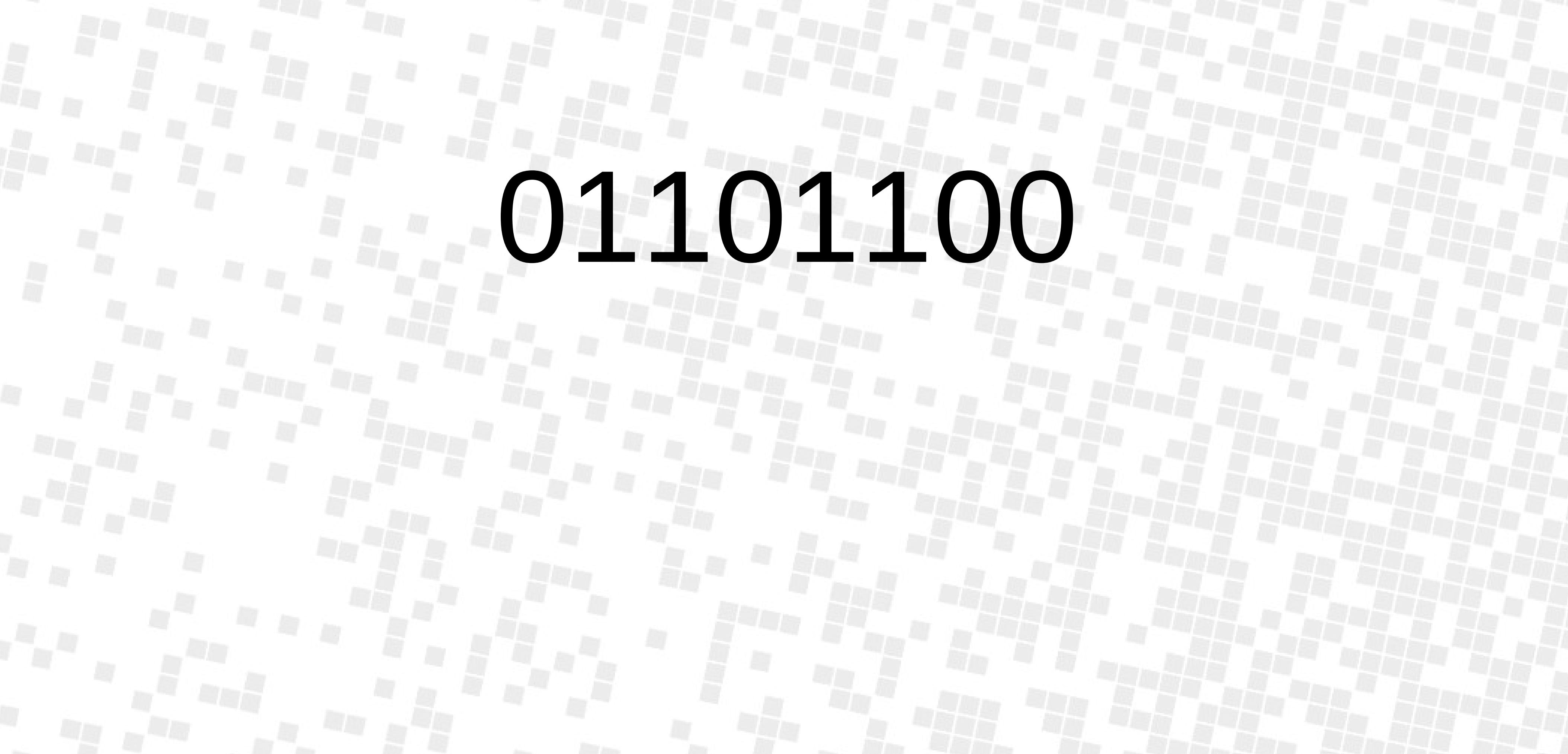
- The basics
- Step 1: Designing a binary file format
- Step 2: Reverse engineering binary files

Talk overview

- **The basics**
- Step 1: Designing a binary file format
- Step 2: Reverse engineering binary files


0110

01101100



01101100

%

01101100

108

01101100

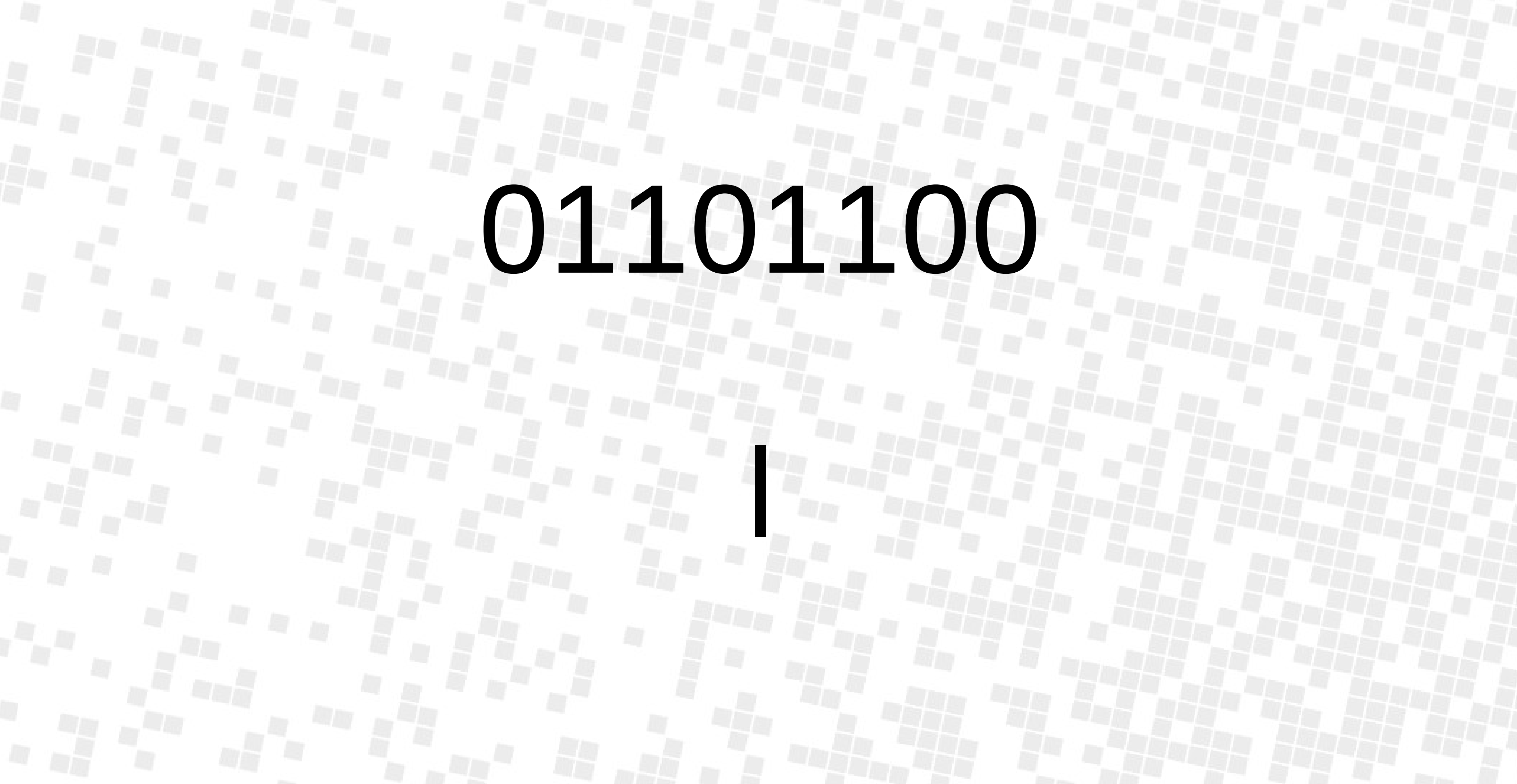
6.75

01101100

112

01101100

160kbit/s, 44.1Khz


01101100
l

Binary data has no inherent meaning

All forms of meaning are ***conventions*** and ***agreements***

Conventions

- Numbers are like decimal numbers
  - $108_{10} == 1 * 10^2 + 0 * 10^1 + 8 * 10^0$
    $== 1 * 100 + 0 * 10 + 8$
  - $01101100_2 == 1*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0$
    $== 2^6 + 2^5 + 2^3 + 2^2$
    $== 64_{10} + 32_{10} + 8_{10} + 4_{10}$
    $== 108_{10}$

Conventions

- What about negative numbers?
  - Sign-magnitude
  - One’s complement
  - Two’s complement
    - Definite choice by all current computers

Conventions

- If the first bit is set, we consider it to represent a negative number
- Just keep subtracting as if it were a regular number

Conventions

0111 == 7
0110 == 6
0101 == 5
0100 == 4
0011 == 3
0010 == 2
0001 == 1
0000 == 0

1111 == 15
1110 == 14
1101 == 13
1100 == 12
1011 == 11
1010 == 10
1001 == 9
1000 == 8

Conventions

0111 == 7
0110 == 6
0101 == 5
0100 == 4
0011 == 3
0010 == 2
0001 == 1
0000 == 0

1111 == -1
1110 == -2
1101 == -3
1100 == -4
1011 == -5
1010 == -6
1001 == -7
1000 == -8

Conventions

0011 == 3
0010 == 2
0001 == 1
0000 == 0
1111 == -1
1110 == -2
1101 == -3
1100 == -4

- $3_{10} + -3_{10} == 0_{10}$
- ```
     0011    ( == 3)
     1101 +  ( == -3)
  (1)0000    ( == 0)
  ```
  ```

Conventions

- Decimals?
  - "Binaries"!
- 6.75_{10}
- 0110.1100_2

$6 * 10^0 + 7 * 10^{-1} + 5 * 10^{-2}$

$2^2 + 2^1 + 2^{-1} + 2^{-2}$

$4 + 2 + ½ + ¼$

$6\ ¾$

Conventions

- Text data is a sequence of identical sized “units”
- Each “unit” represents one letter or action
  - EBCDIC
  - ASCII
  - Unicode UTF-16

Conventions

0100100001100101011011000110110001101111

Hello

Conventions

01001000

01100101

01101100

01101100

01101111

Conventions

01001000	Letter, upper case, 8th
01100101	Letter, lower case, 5th
01101100	Letter, lower case, 12th
01101100	Letter, lower case, 12th
01101111	Letter, lower case, 15th

Conventions

01001000	Letter, upper case, 8th == H
01100101	Letter, lower case, 5th == e
01101100	Letter, lower case, 12th == l
01101100	Letter, lower case, 12th == l
01101111	Letter, lower case, 15th == o

Bigger numbers, use multiple bytes

- Little endian
  - Start from the little end
- 0000000110100100
- 10100100 00000001

- Use a library to read/write the appropriate variant

- Big endian
  - Start from the big end
- 0000000110100100
- 00000001 10100100

- C++20: std::endian

What **else** can you do with 8 bits?

- Index into lookup table with up to 256 entries
- Variable-length encodings
- Run-length encoding
- Backreferences

Variable length encoding

0x01

The value 1 in a variable-length encoding (Protobuf style)

Variable length encoding

0x02

The value 2 in a variable-length encoding (Protobuf style)

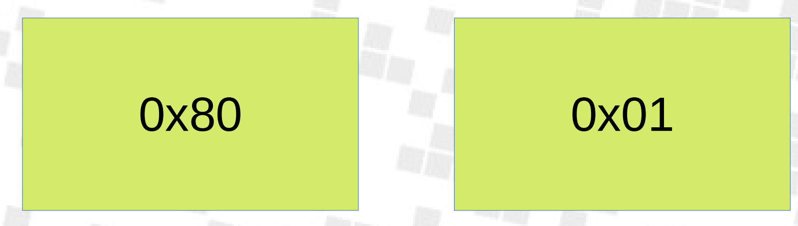
If the top bit is not set, nothing special happens

Variable length encoding

0x7F

The value 127 in a variable-length encoding (Protobuf style)

Variable length encoding



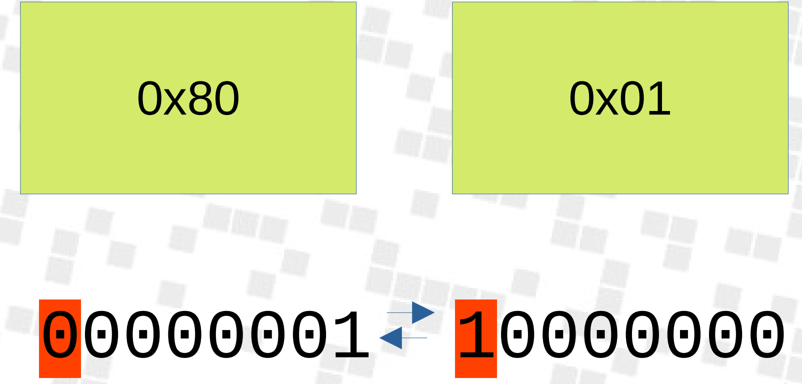
The value 128 in a variable-length encoding (Protobuf style)

Every time the top bit is set, we pull in one extra byte

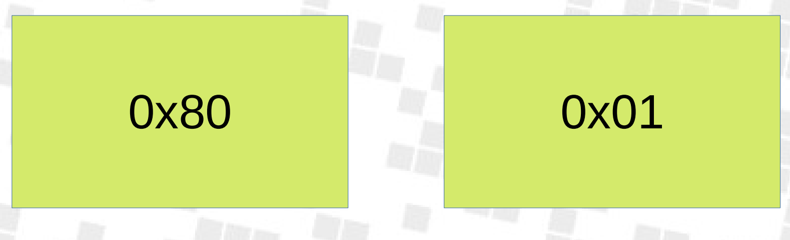
Variable length encoding

10000000 00000001

Variable length encoding

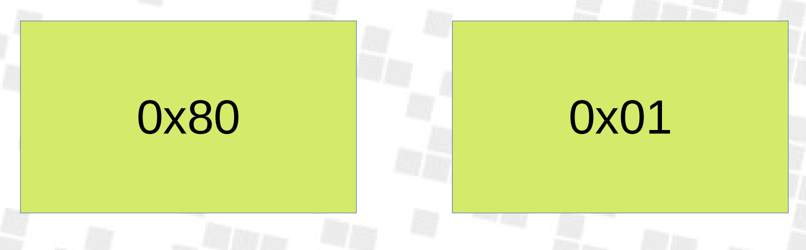


Variable length encoding



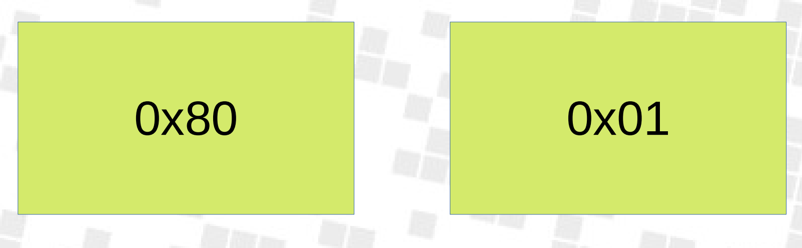
0000001 0000000

Variable length encoding



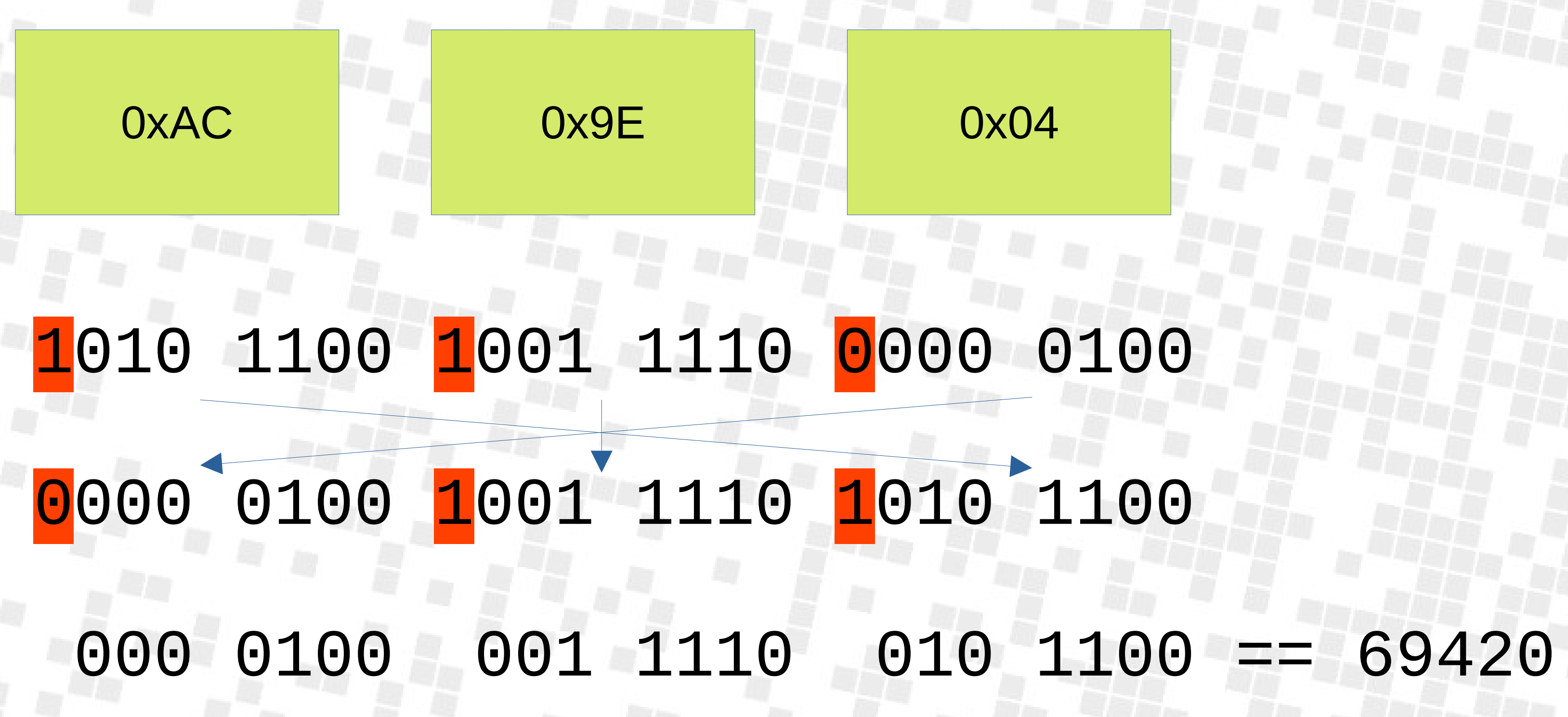
00 0000 1000 0000

Variable length encoding



00 0000 1000 0000 (= 128)

Variable length encoding



Protobuf in detail

- Expanding encoding for values
- Serialized structs
- Each field in the struct is serialized individually
  - First value identifies encoding method and field index number
  - For encoding 2, second value identifies length

Protobuf in detail

```
message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
}
```

Protobuf in detail

```
0A 05 50 65 74 65 72 10 2A 1A 11 64
61 73 63 61 6E 64 78 40 67 6D 61 69
6B 2E 63 6F 6D
```

Protobuf in detail

```
0A 05 50 65 74 65 72

10 2A

1A 11 64 61 73 63 61 6E 64 78 40 67
6D 61 69 6B 2E 63 6F 6D
```

Protobuf in detail

```
0A 05 50 65 74 65 72

0000 1/010 Field 1, encoding 2 == length/data

10 2A

0001 0/000 Field 2, encoding 0 == var-length integer

1A 11 64 61 73 63 61 6E 64 78 40 67 6D 61 69 6B 2E 63
6F 6D

0001 1/010 Field 3, encoding 2 == length/data
```

General purpose compression

- Pure random input: Impossible
- Actual data: very doable

- Base idea: Predict what comes next and only write it down if it is not that.

Let’s try it

Four score and	seven years ago
I have a	dream
Once	upon a time
Tom and	Jerry
Game	Over
Supercali	fragilisticexpialidocious

Compression

Shannon entropy

- Numbers: 2log(range) bits
- English text: ~2.6 bits per character

https://en.wikipedia.org/wiki/Shannon%27s_source_coding_theorem

https://www.princeton.edu/~wbialek/rome/refs/shannon_51.pdf

Compression

Kolmogorov complexity

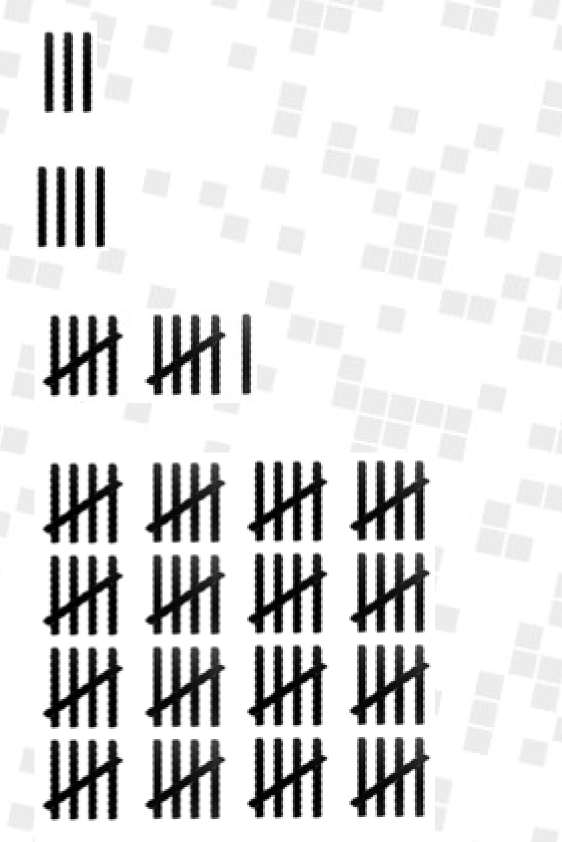
- Same concept from the other direction

- Smallest possible representation of a given amount of data

General compression techniques

- Dictionary building (LZ78, LZW)
- Run-length encoding
- Backreference encoding (LZ77, zlib, gzip, deflate)
  - Can do everything RLE can do, but better
- Data transforms
  - Do not compress, but leave data in more compressible form
- Variable-length codes (Huffman, Arithmetic, Range)

Run-length encoding



- 3
- 4
- 11
- 80

- 6243 ?

Run-length encoding

Photo by Doug Maloney @ Unsplash

Backreference encoding

Baba yetu, yetu uliye
Mbinguni yetu, yetu, amina
Baba yetu, yetu, uliye
Jina lako litukuzwe
Baba yetu, yetu uliye
Mbinguni yetu, yetu, amina
Baba yetu, yetu, uliye
Jina lako litukuzwe

Utupe leo chakula chetu
Tunachohitaji utusamehe
Makosa yetu, hey
Kama nasi tunavyowasamehe
Waliotukosea, usitutie
Katika majaribu, lakini
Utuokoe, na yule, milele na milele
Baba yetu, yetu uliye
Mbinguni yetu, yetu, amina

Baba Yetu, Christopher Tin, Civilization 4 sound track; Swahili translation of The Lord's Prayer.

Backreference encoding

Baba yetu, yetu uliye

Mbinguni yetu, yetu, amina

Baba yetu, yetu, uliye

Jina lako litukuzwe

Baba yetu, yetu uliye

Mbinguni yetu, yetu, amina

Baba yetu, yetu, uliye

Jina lako litukuzwe

Utupe leo chakula chetu

Tunachohitaji utusamehe

Makosa yetu, hey

Kama nasi tunavyowasamehe

Waliotukosea, usitutie

Katika majaribu, lakini

Utuokoe, na yule, milele na milele

Baba yetu, yetu uliye

Mbinguni yetu, yetu, amina

Baba Yetu, Christopher Tin, Civilization 4 sound track; Swahili translation of The Lord's Prayer.

Backreference encoding

Baba yetu, yetu uliye

Mbinguni yetu, yetu, amina

Baba yetu, yetu, uliye

Jina lako litukuzwe

Copy 91 characters from 91 back

Utupe leo chakula chetu

Tunachohitaji utusamehe

Makosa yetu, hey

Kama nasi tunavyowasamehe

Waliotukosea, usitutie

Katika majaribu, lakini

Utuokoe, na yule, milele na milele

Copy 91 characters from 281 back

Baba Yetu, Christopher Tin, Civilization 4 sound track; Swahili translation of The Lord's Prayer.

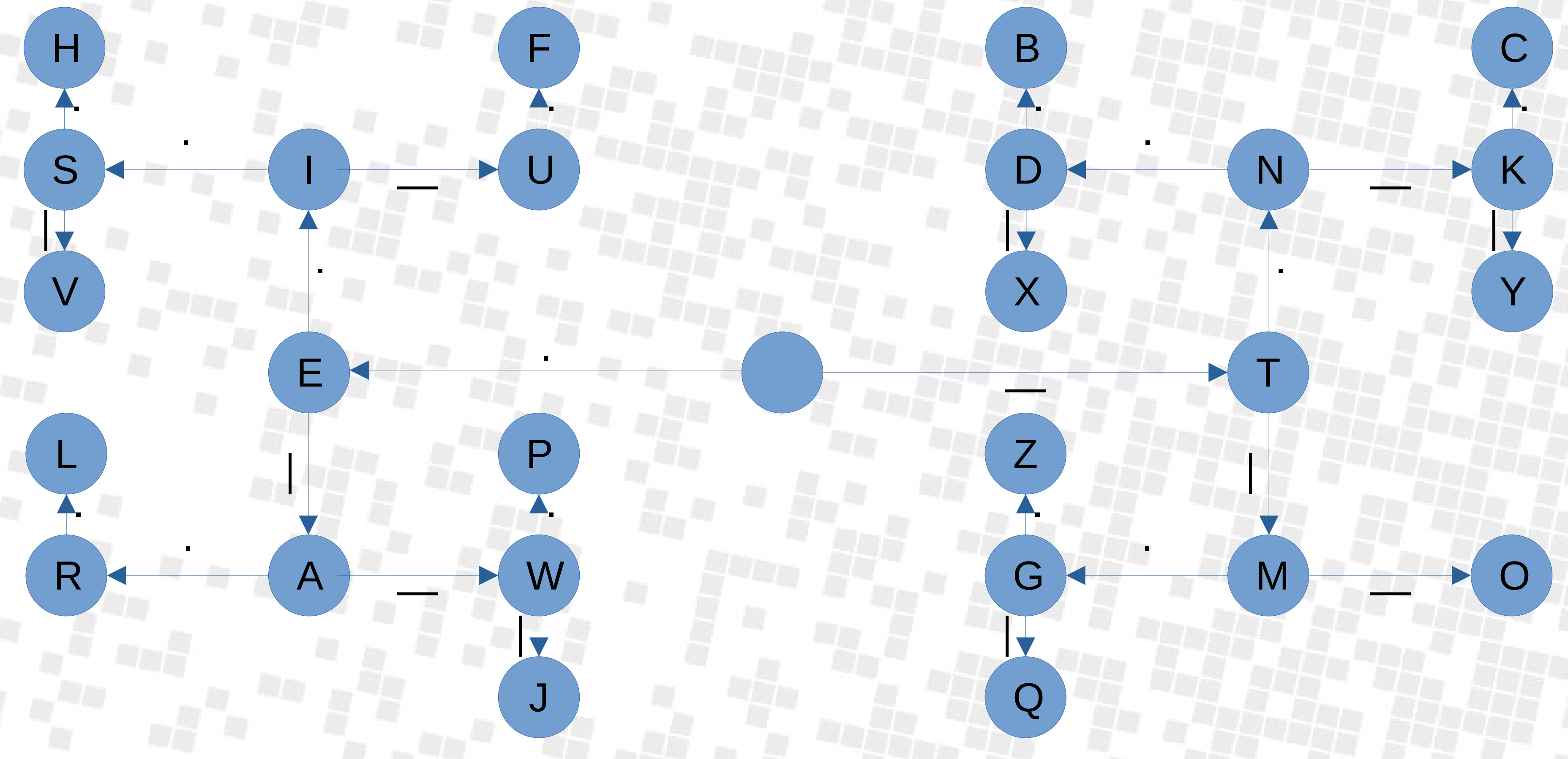
Data transforms

- Use prior knowledge you have to increase predictability of your data
  - Array of Structs to Struct of Arrays
  - Sorting data
  - Delta coding (png)
  - Move-To-Front coding (bzip2)

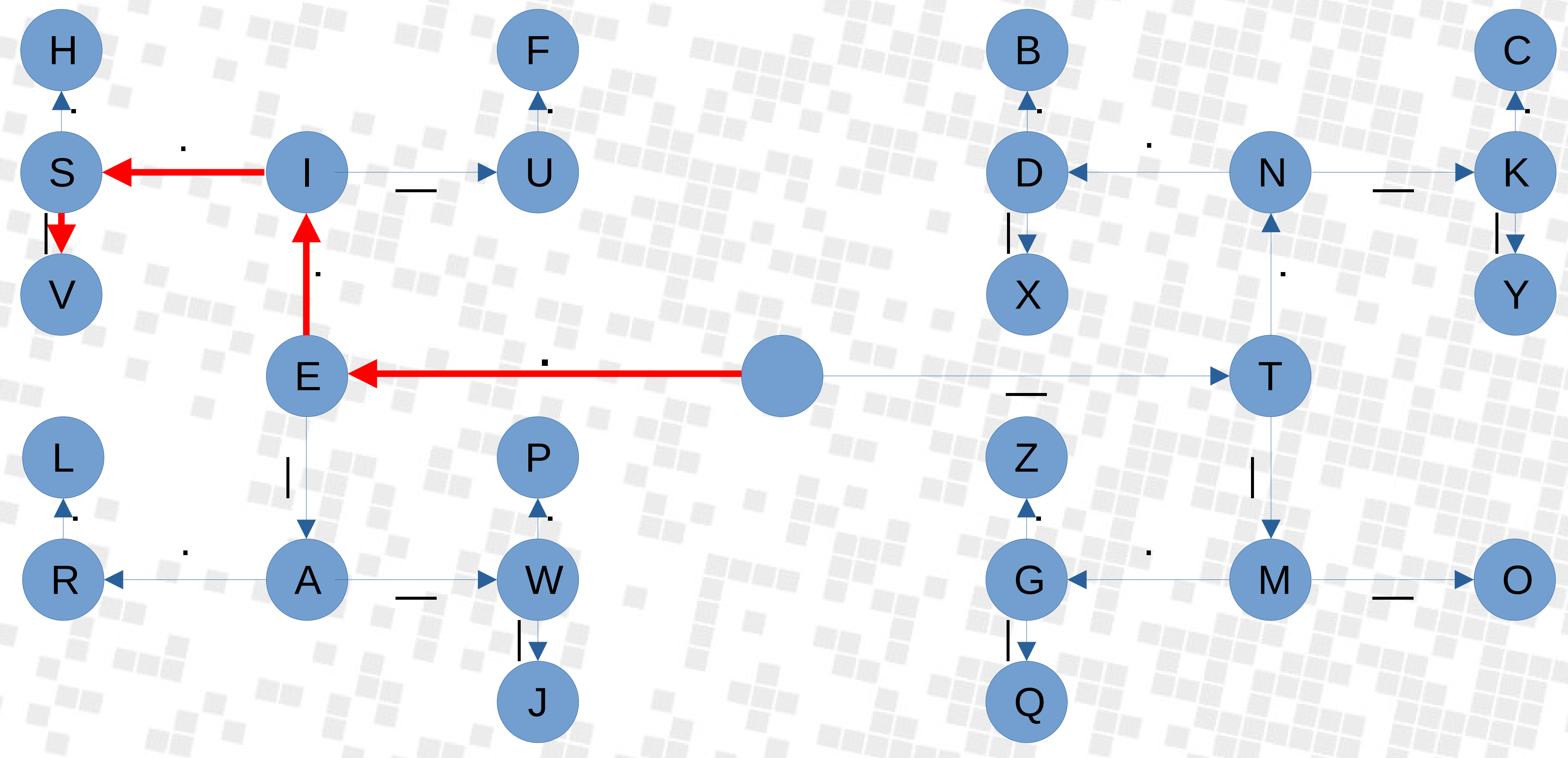
Variable-length codes

... --- -- . / .--. . --- .--. .-.. . / -.-. .- -.
/ .--- ..- ... - / .-. . .- -.. / - ....

Variable-length codes



Variable-length codes



Variable-length codes

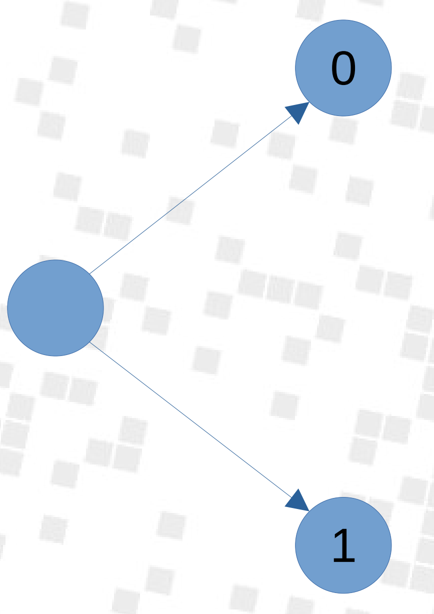
... --- -- . / .--. . --- .--. .-.. . / -.-. .- -.

/ .--- ..- ... - / .-. . .- -.. / - ....

Bit-packing

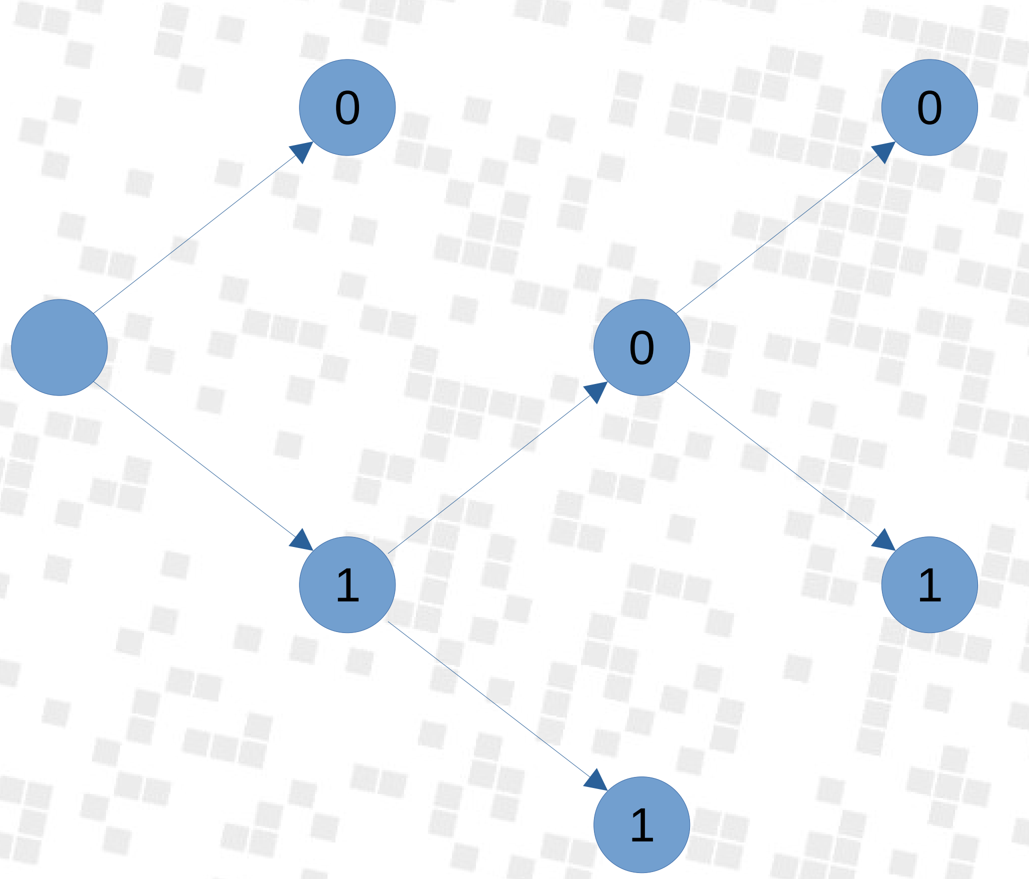
- Instead of assuming bytes or larger units, we represent information with parts of a byte
- Shorter values for more likely things, longer values for less likely things

Variable-length codes



0: Option 1
1: Option 2

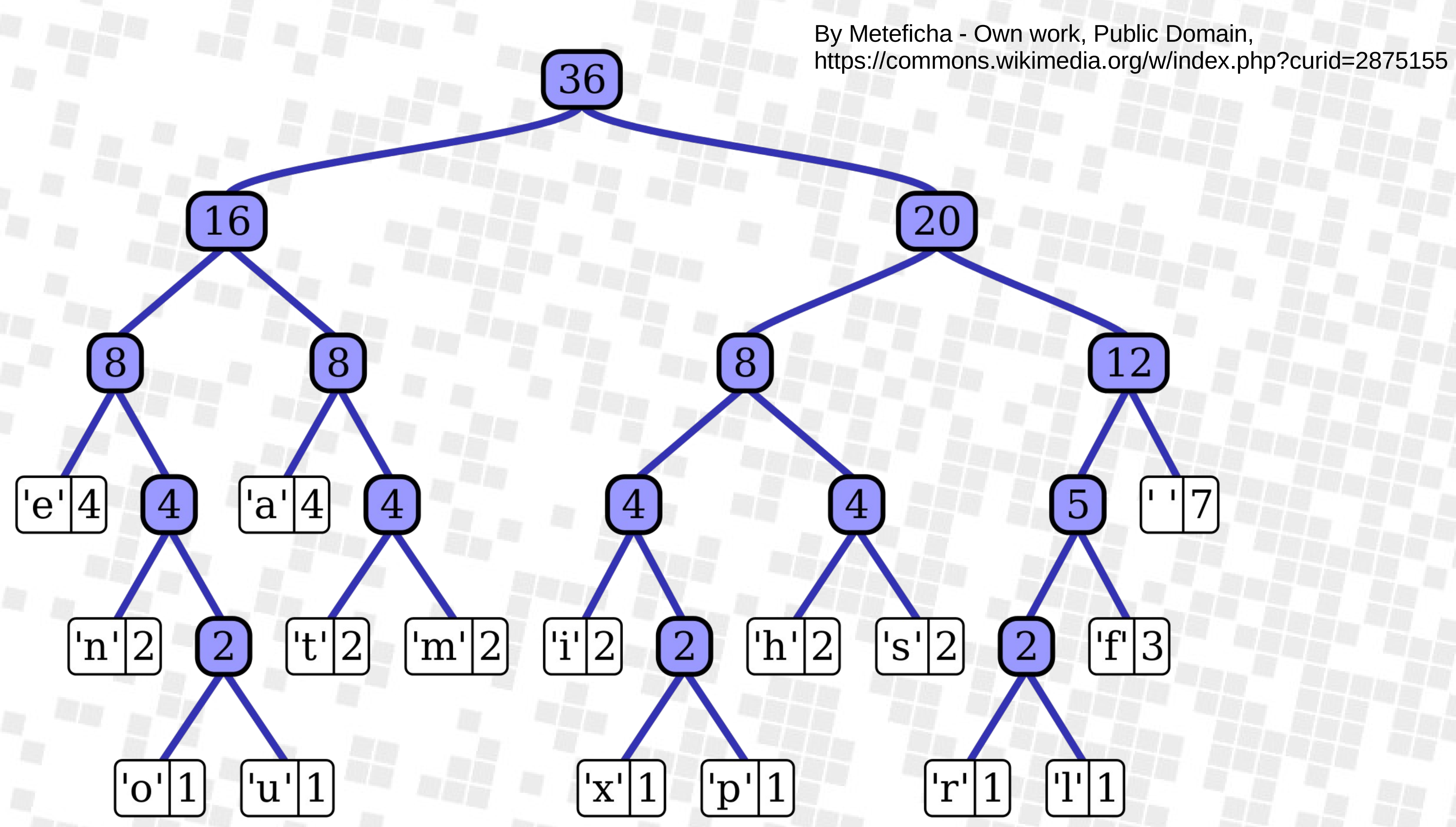
Variable-length codes



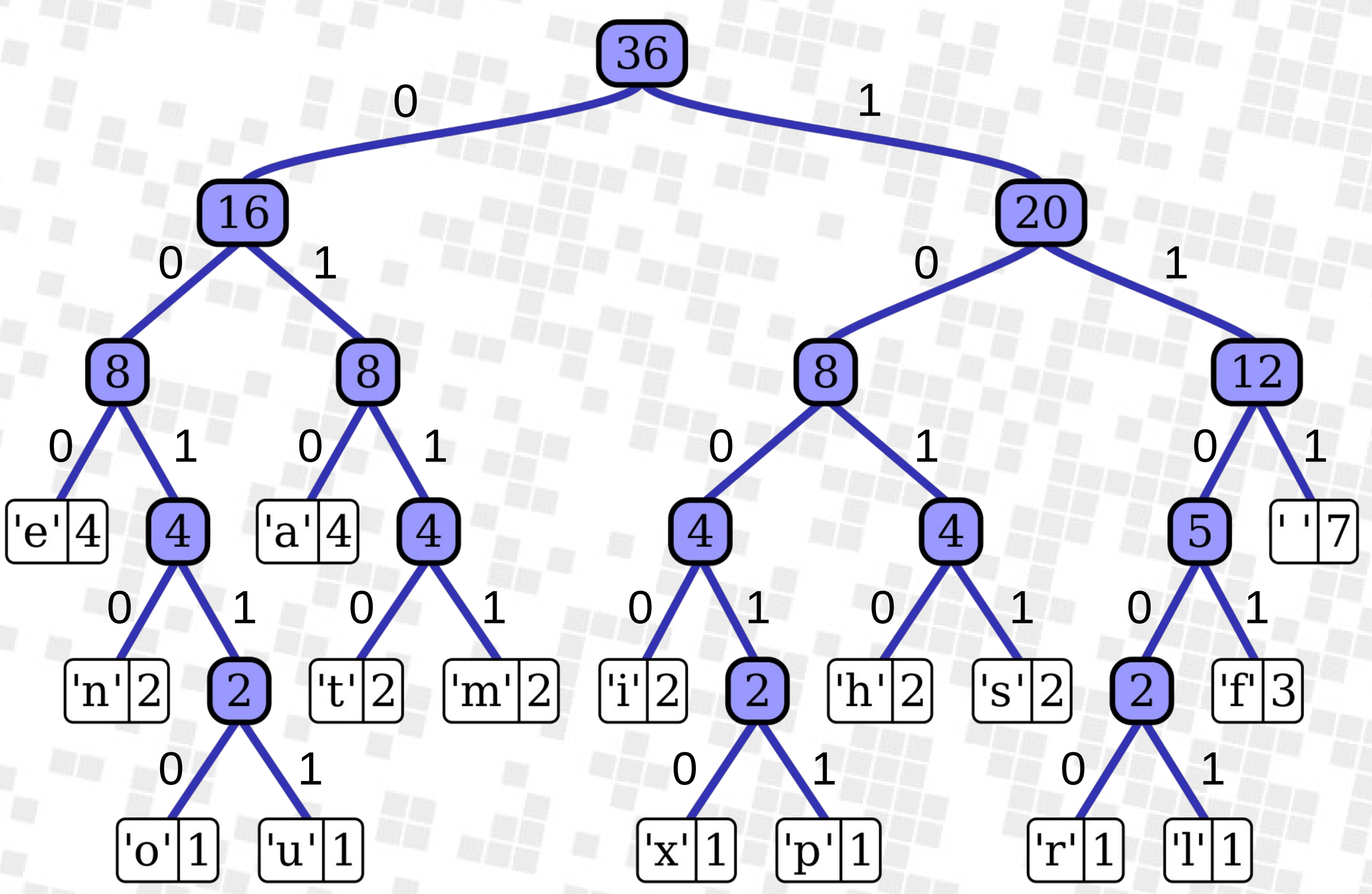
0: Option 1
100: Option 2
101: Option 3
11: Option 4

Huffman coding

By Meteficha - Own work, Public Domain,
https://commons.wikimedia.org/w/index.php?curid=2875155

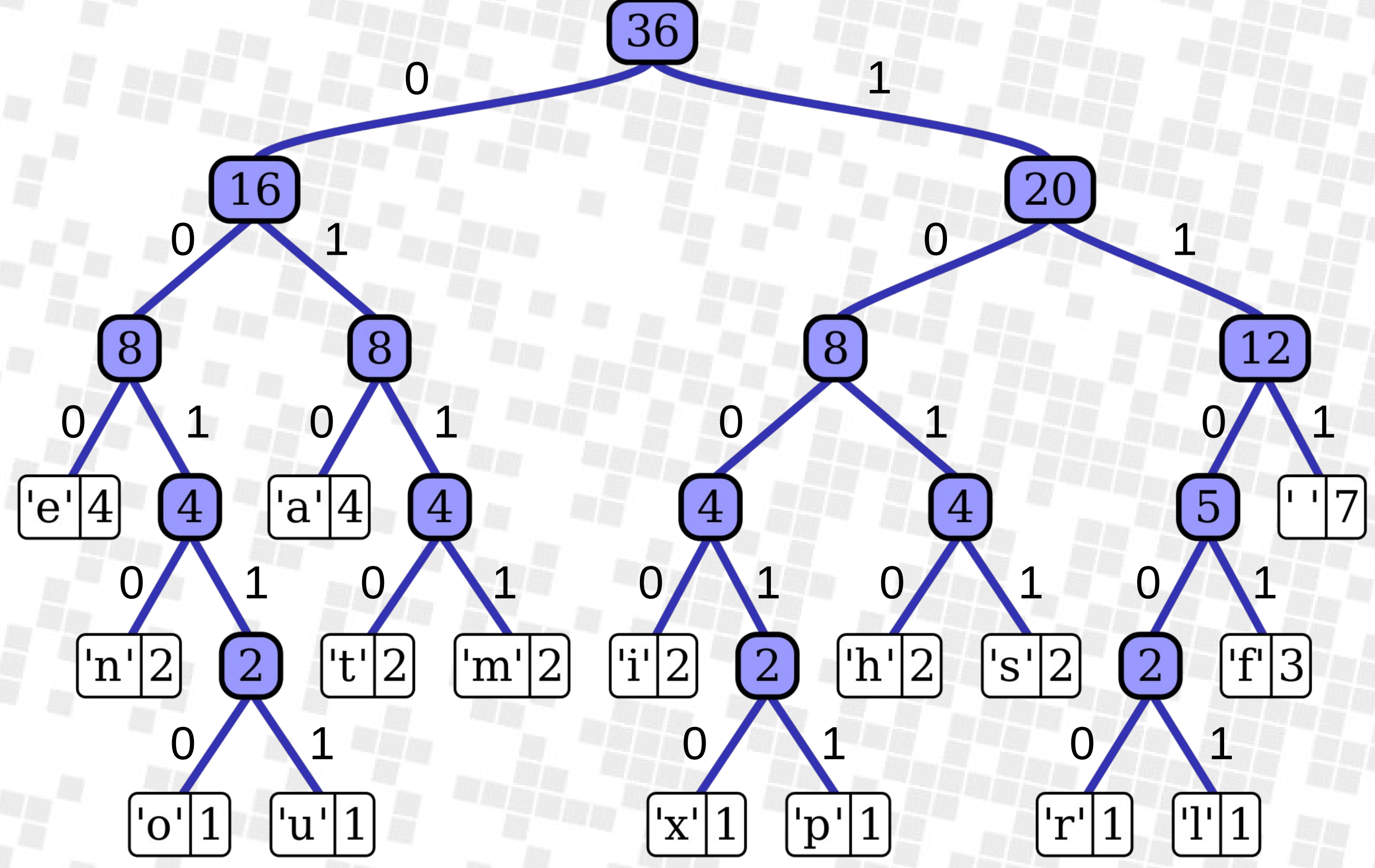


Huffman coding



Huffman coding

Letter	Code
e	000
n	0010
o	00110
u	00111
a	010
t	0110
m	0111
i	1000
x	10010
p	10011
h	1010
s	1011
r	11000
l	11001
f	1101
' '	111



Huffman coding

Letter	Code
e	000
n	0010
o	00110
u	00111
a	010
t	0110
m	0111
i	1000
x	10010
p	10011
h	1010
s	1011
r	11000
l	11001
f	1101
‘ ‘	111

h	e	l	l	o
01101000	01100101	01101100	01101100	01101111

Huffman coding

Letter	Code
e	**000**
n	0010
o	**00110**
u	00111
a	010
t	0110
m	0111
i	1000
x	10010
p	10011
h	**1010**
s	1011
r	11000
l	**11001**
f	1101
‘ ‘	111

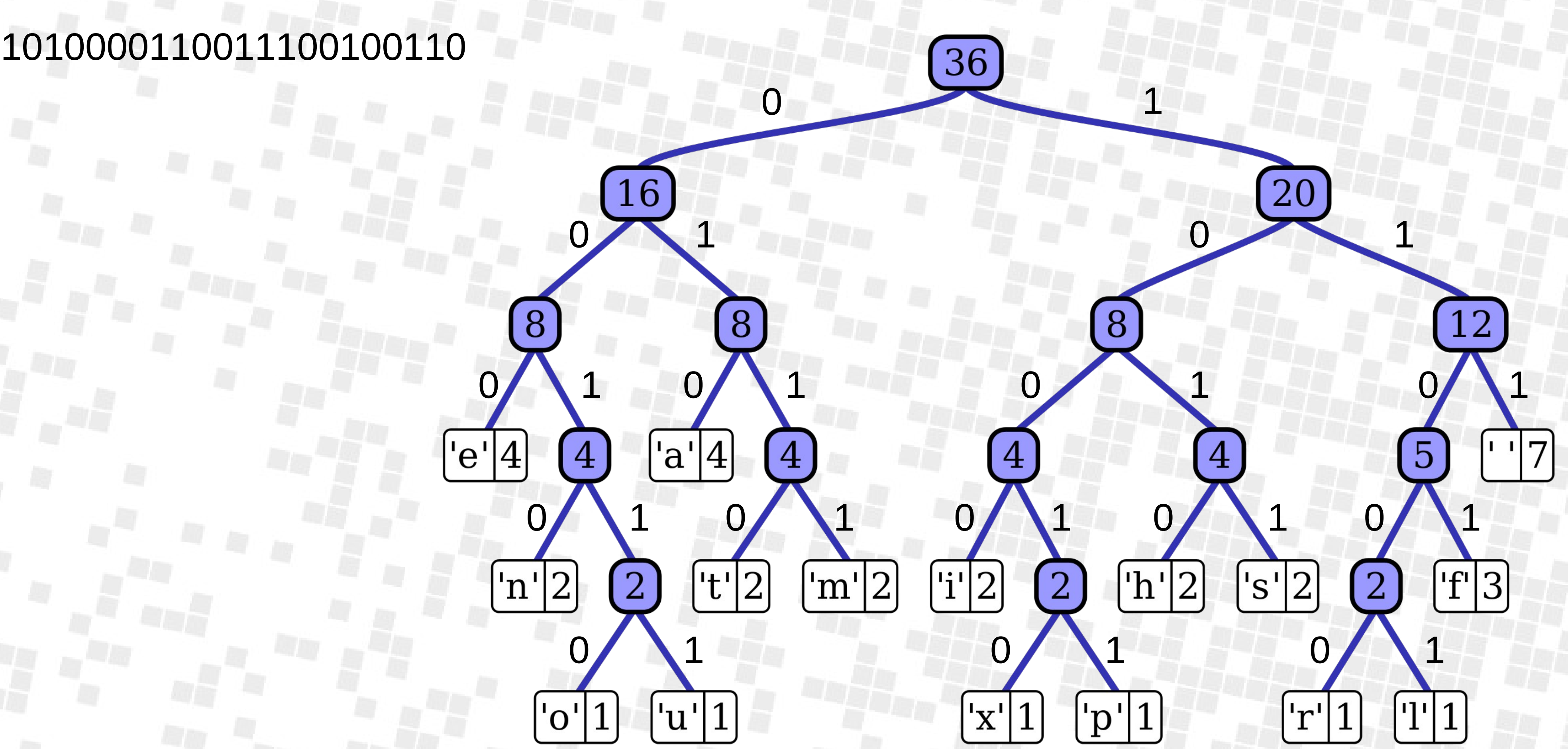
h	e	l	l	o	
01101000	01100101	01101100	01101100	01101111	40 bits
1010	000	11001	11001	00110	22 bits

Huffman coding

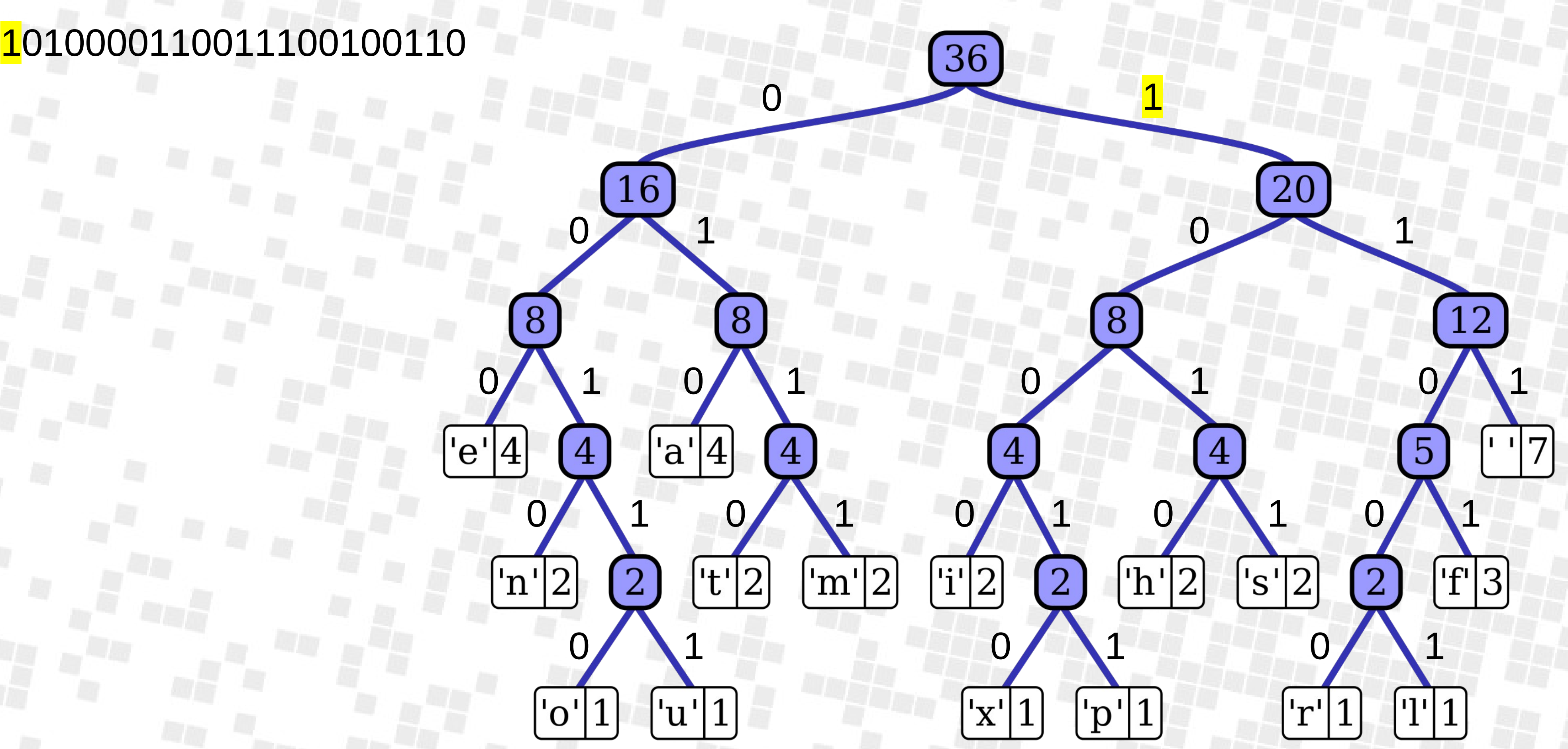
1010000110011100100110

Huffman coding

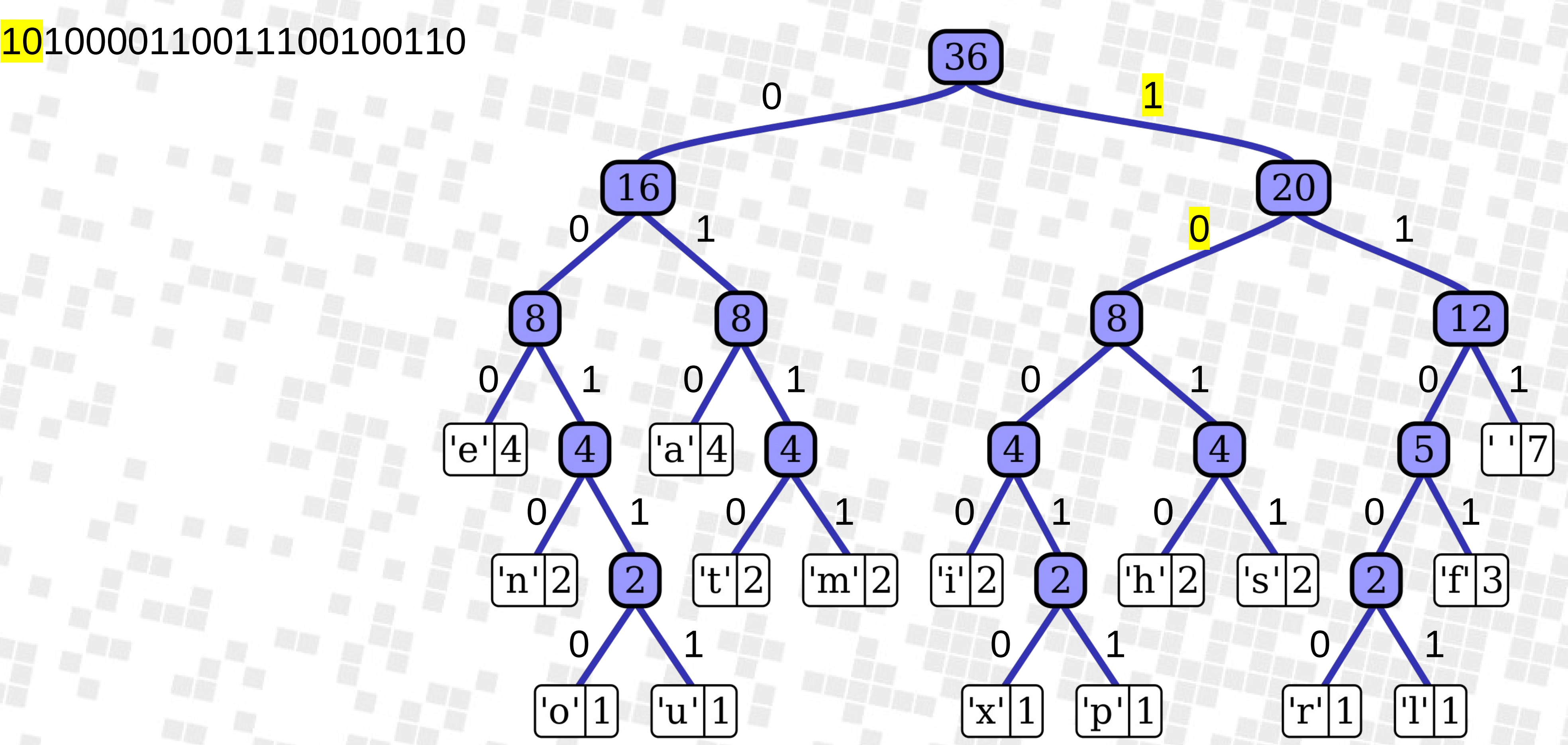
1010000110011100100110



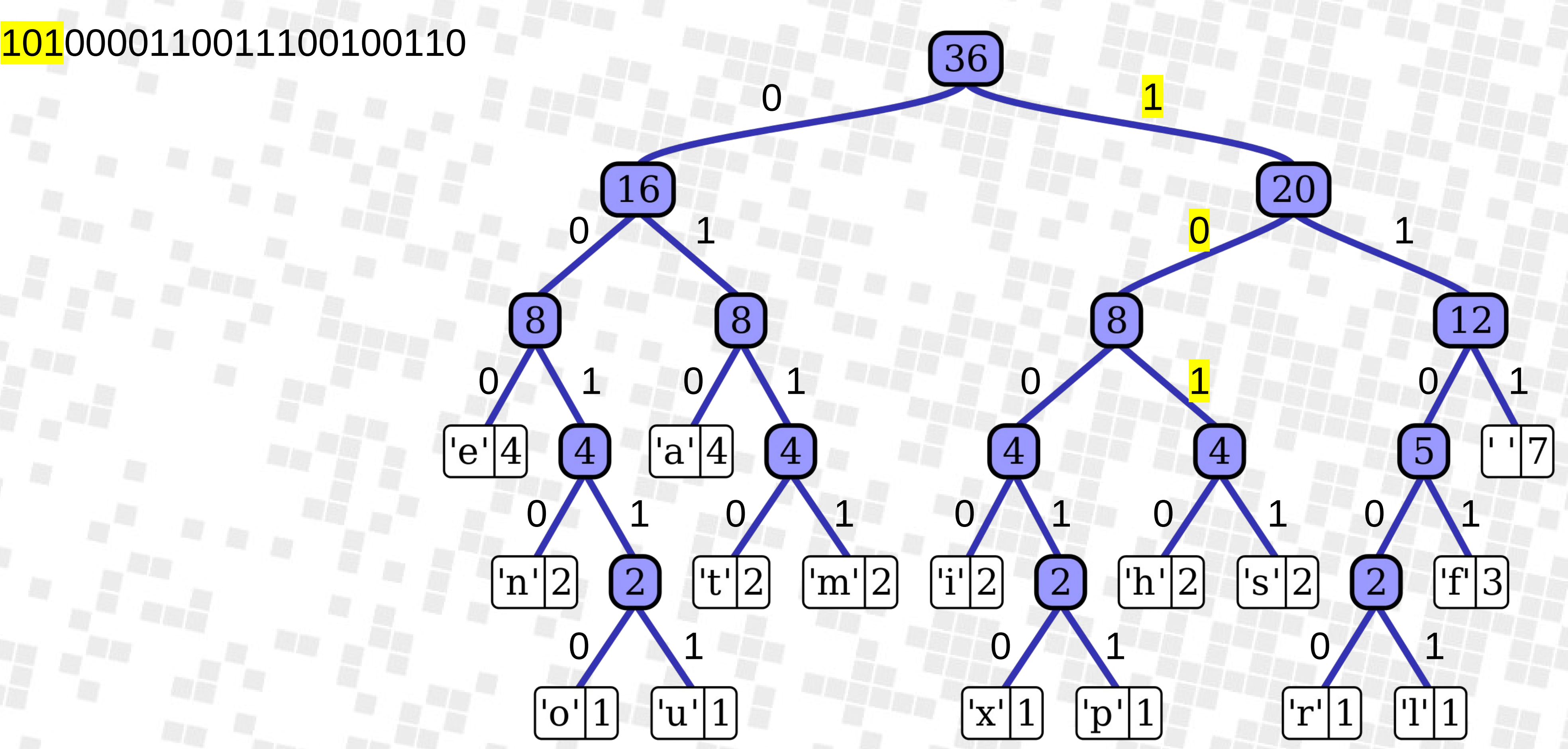
Huffman coding



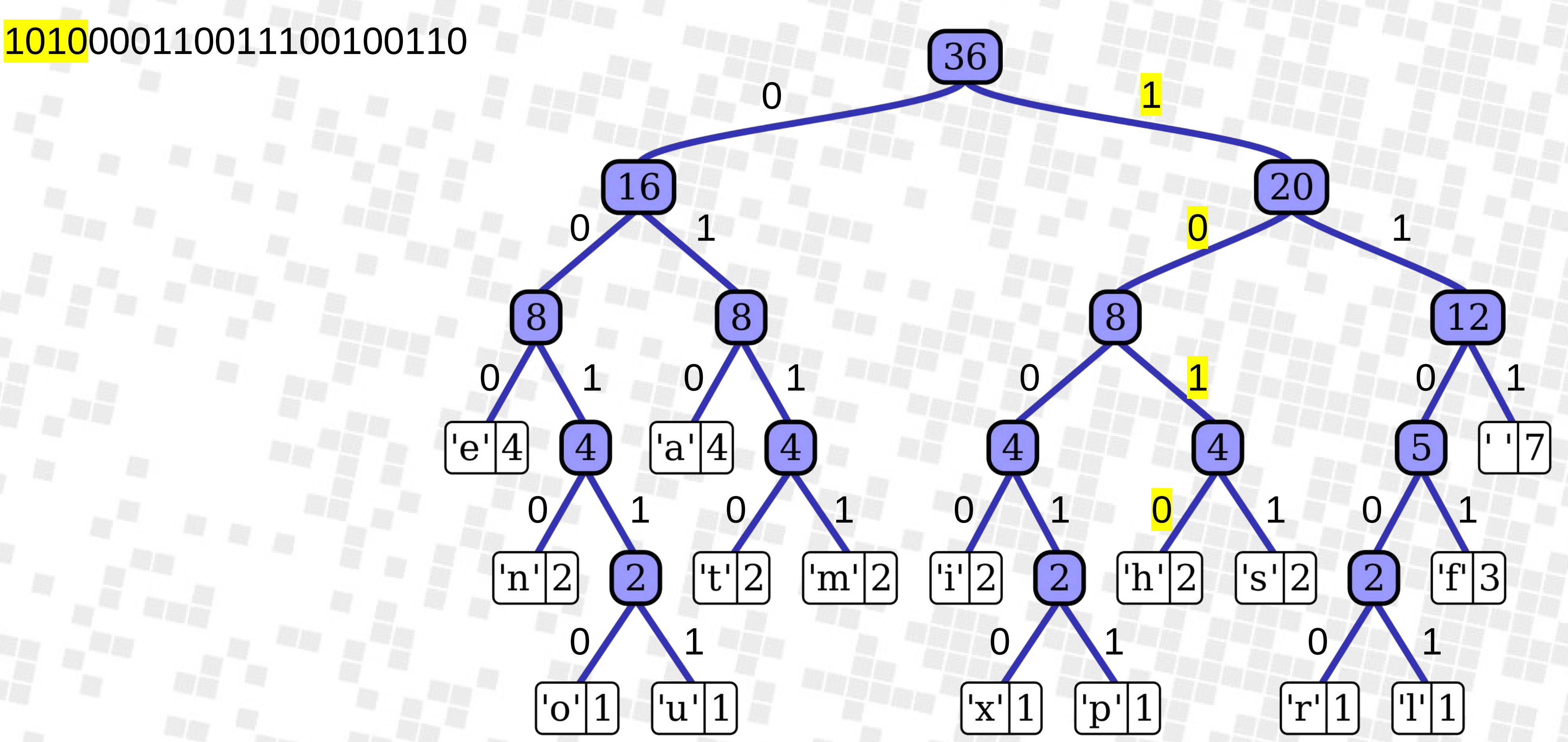
Huffman coding



Huffman coding



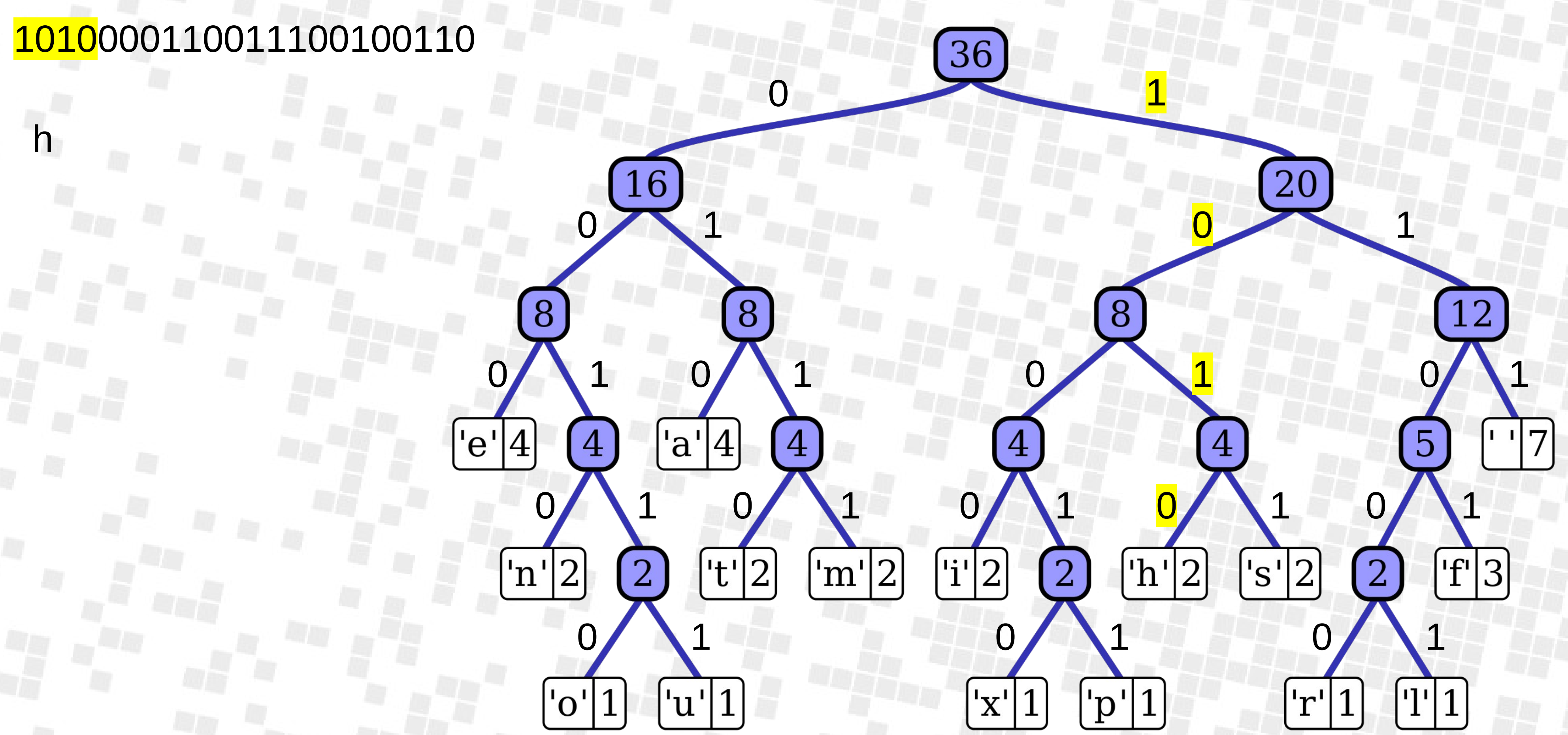
Huffman coding



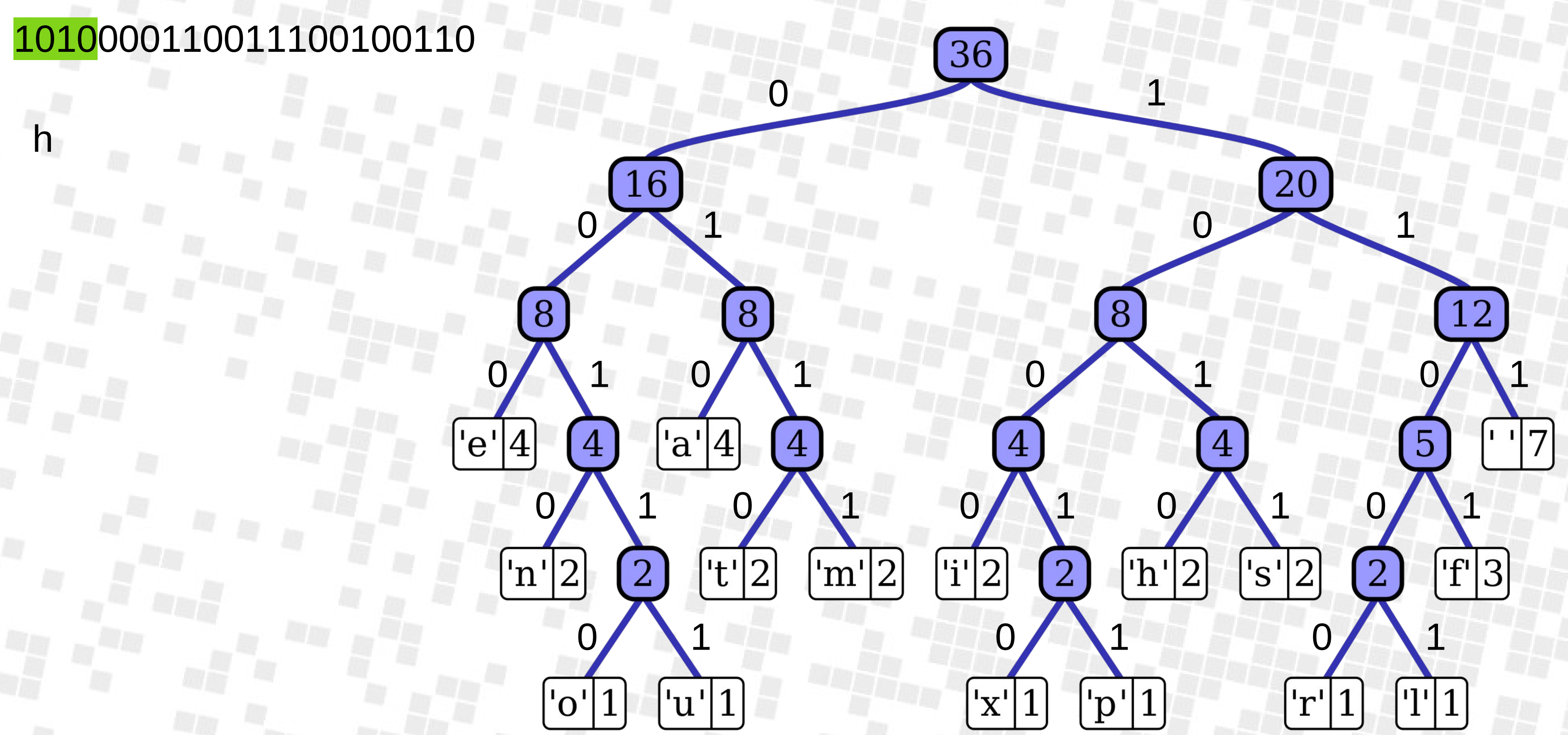
Huffman coding

1010000110011100100110

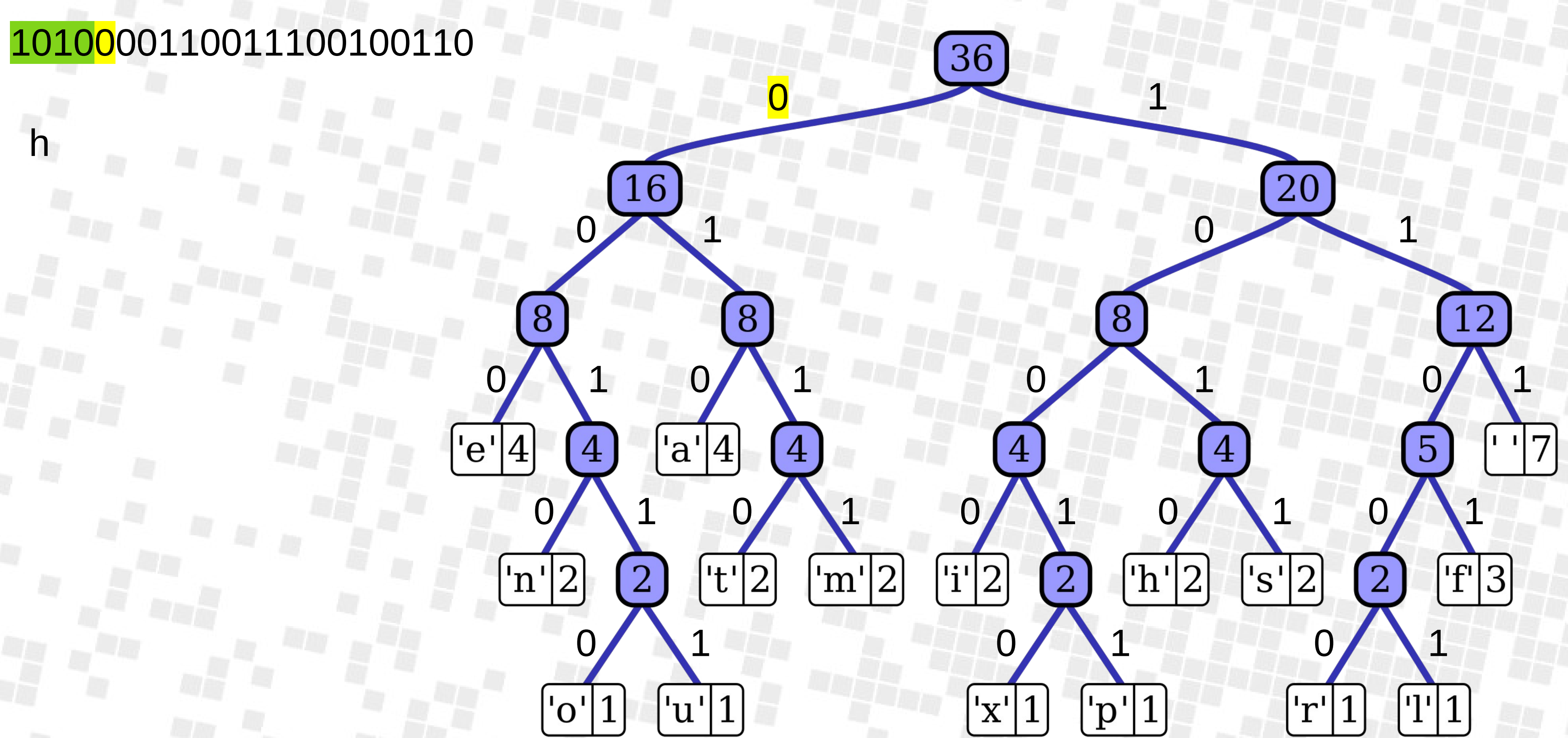
h



Huffman coding



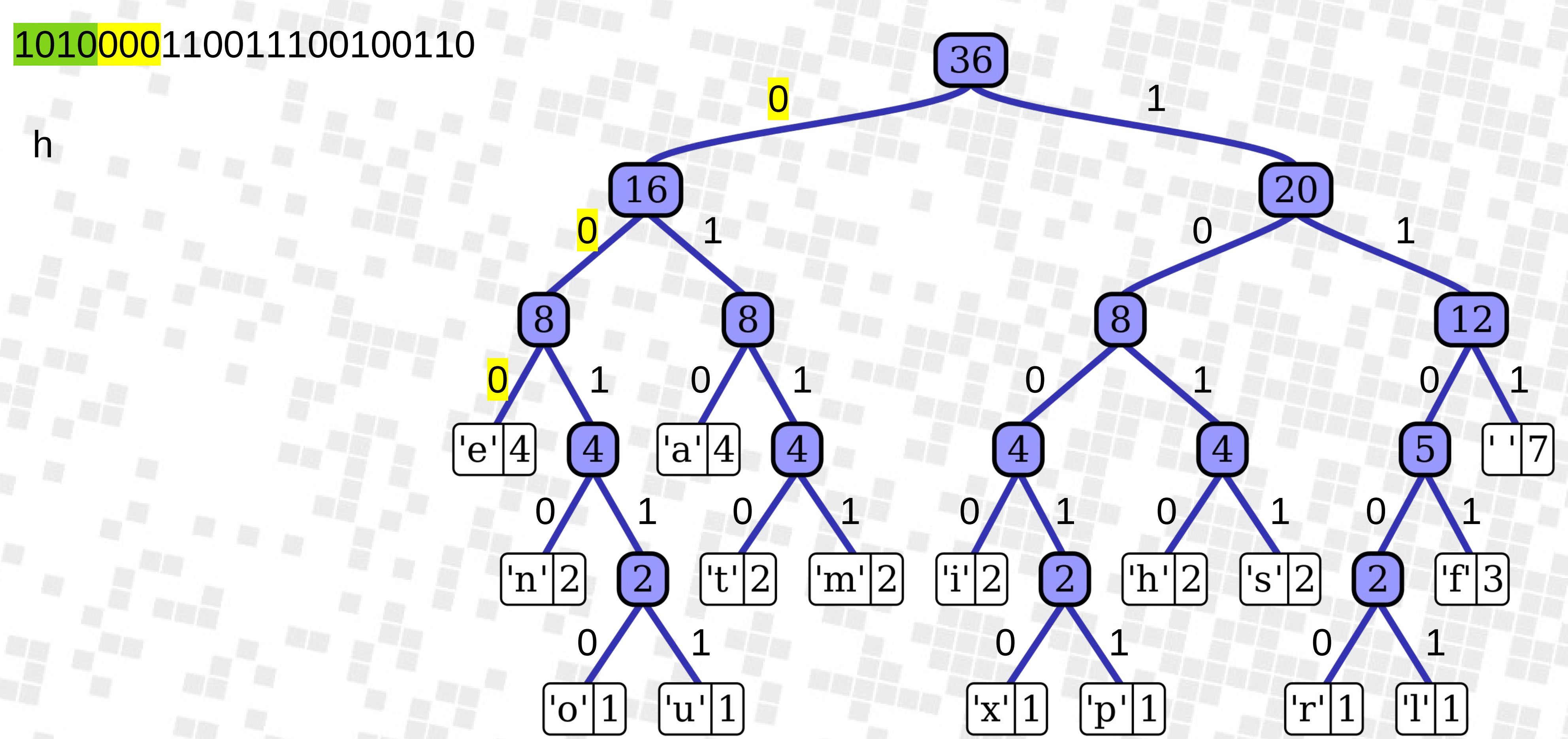
Huffman coding



Huffman coding

1010000110011100100110

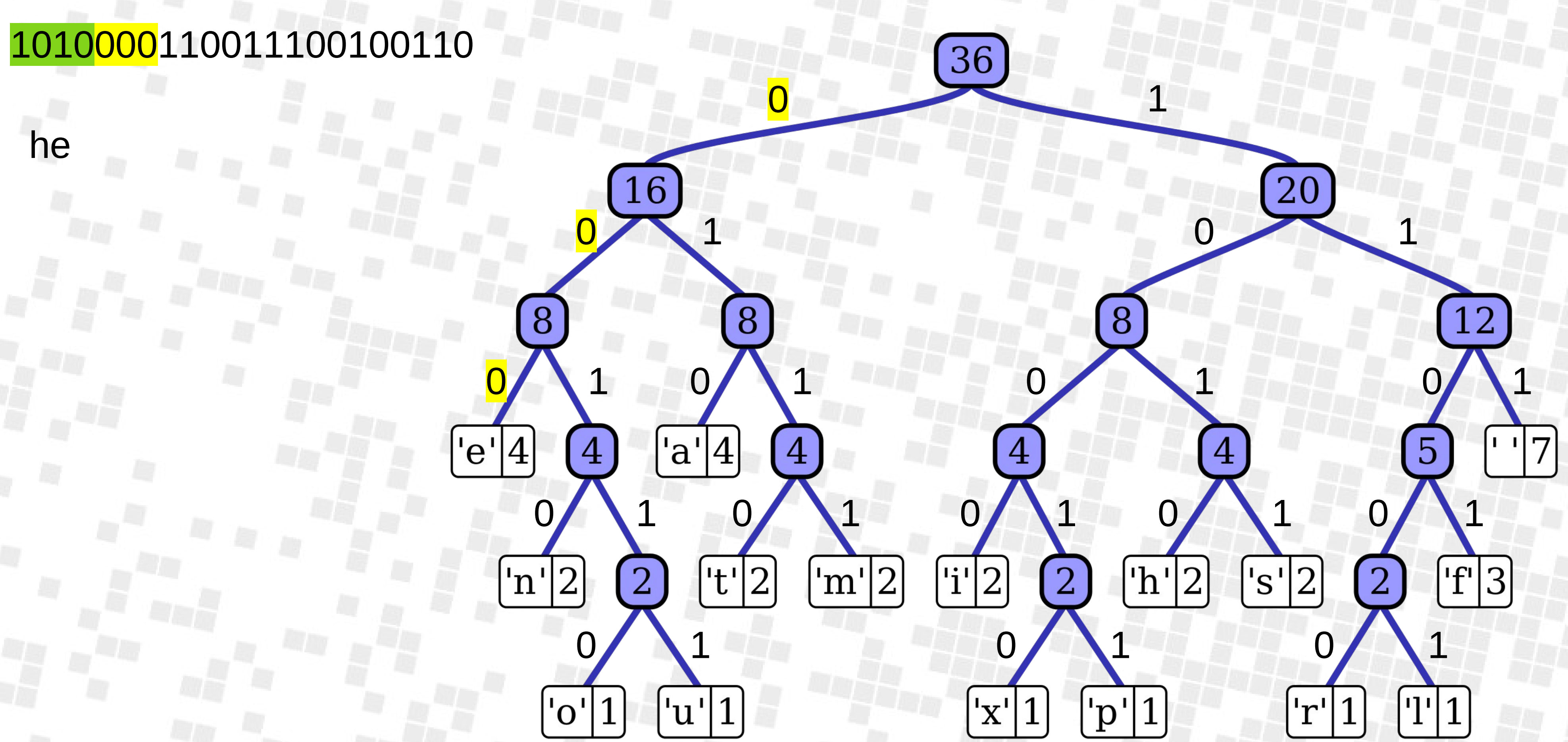
h



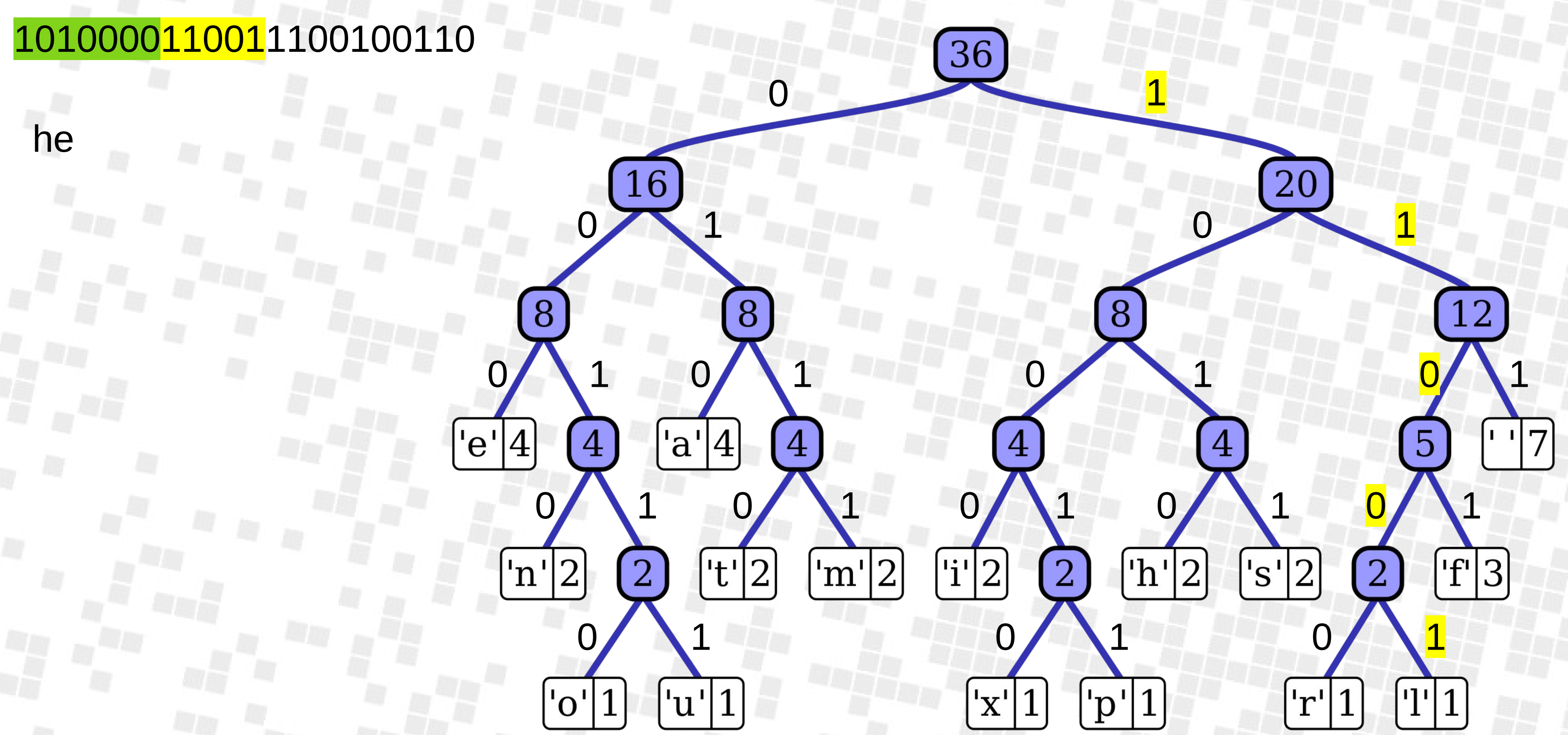
Huffman coding

1010000110011100100110

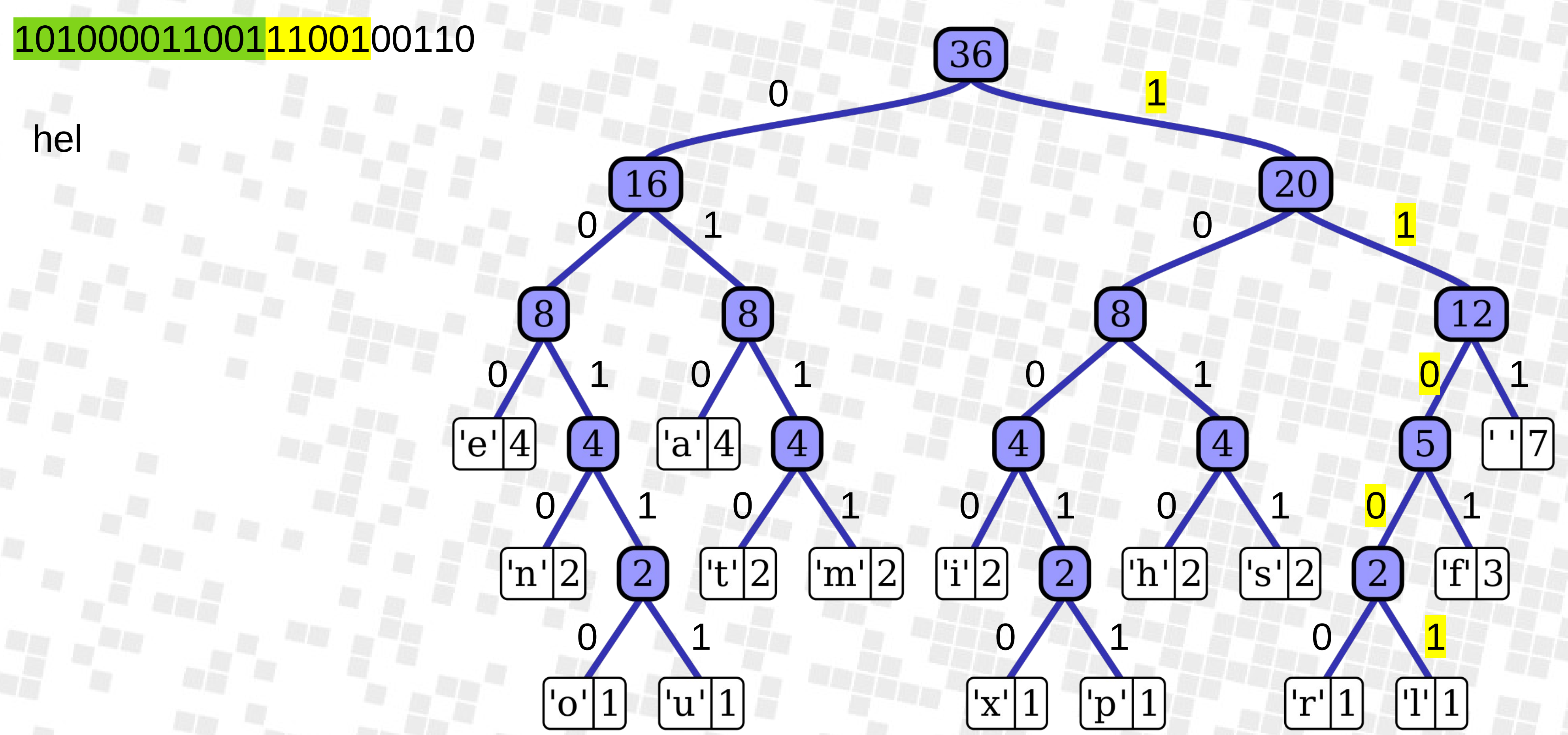
he



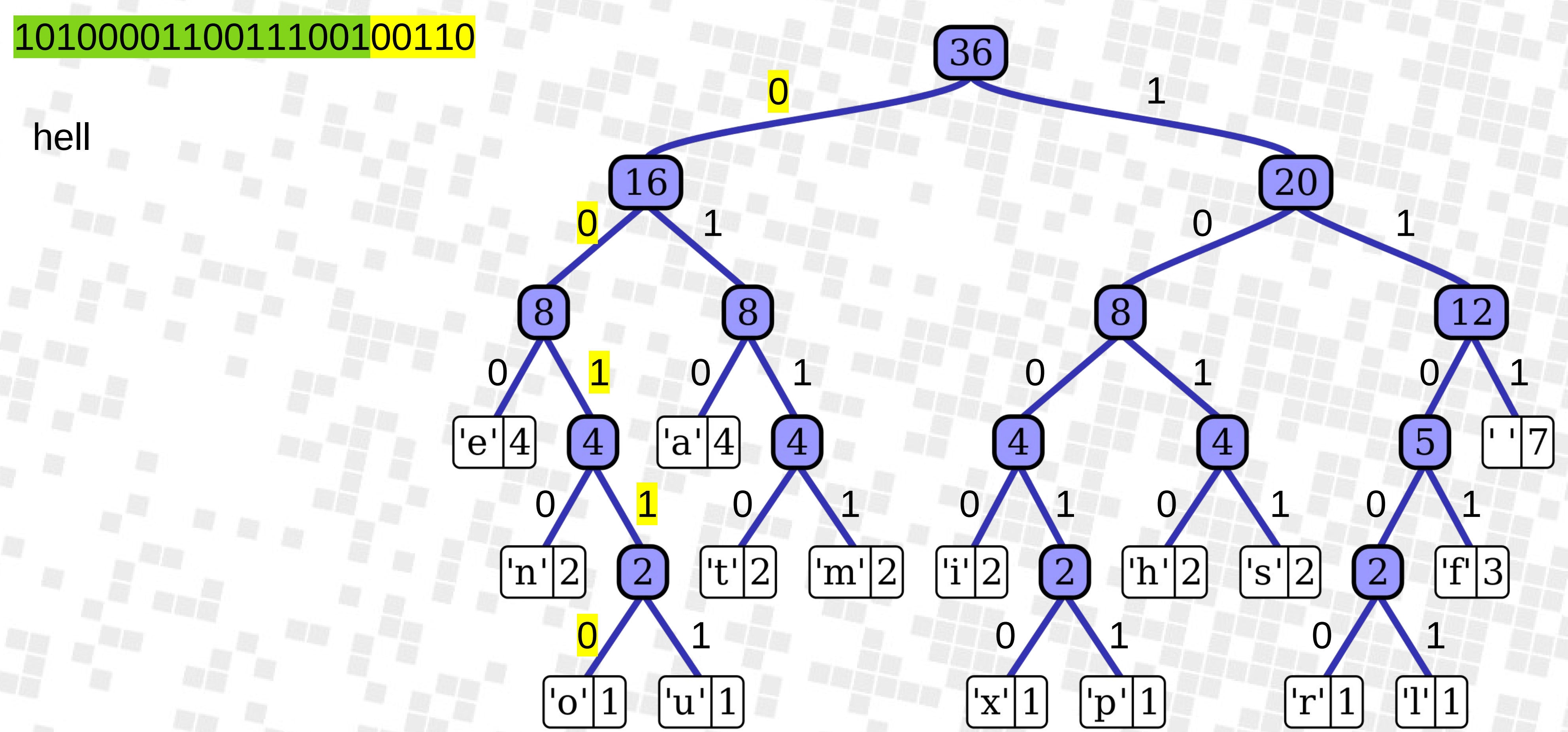
Huffman coding



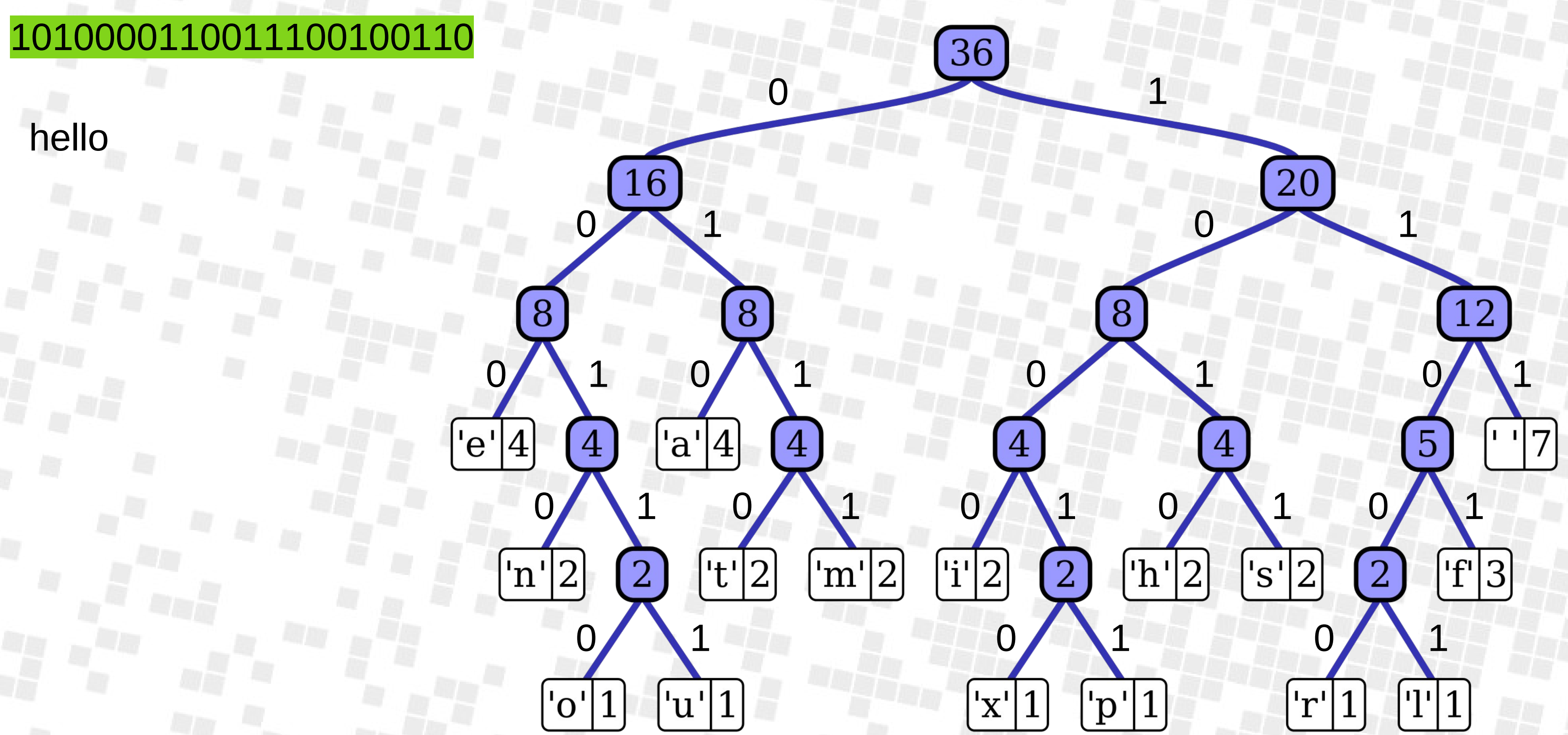
Huffman coding



Huffman coding



Huffman coding



Huffman coding

- General purpose technique
- Implicit table
  - Known input likelyhoods
  - FFT coefficients, text data, XML files
  - Has to accommodate any input values
- Explicit table
  - Unknown input likelyhood
  - Full input available to create table
  - Can leave out values that do not happen to occur

Huffman coding

Letter	Code
e	**000**
n	0010
o	**00110**
u	00111
a	010
t	0110
m	0111
i	1000
x	10010
p	10011
h	**1010**
s	1011
r	11000
l	**11001**
f	1101
‘ ‘	111

h	e	l	l	o	
01101000	01100101	01101100	01101100	01101111	40 bits
1010	000	11001	11001	00110	22 bits

Bytepacking / bitpacking

- Encoding actual data into as few as possible bytes or bits
- Bytepacked is typically easier to write code for
- Bitpacked is more compact
- Sometimes mixed; packing fields into bytes first

Talk overview

- The basics
- **Step 1: Designing a binary file format**
- Step 2: Reverse engineering binary files

Problems with binary file parsing

- Endianness
  - Easy to handle with a library
- Newline conversion
  - Problem that's almost entirely left in the 20th century

Types of binary files

- Flat
- Header-flat
- Header-index-data
- Record based
- Chunked

“Flat” files

- RAW images
- Lookup tables
- Palettes

159	6.0	24	4.0	87	14%	1%
237	9.0	37	4.3	129	8%	1%
262	16.0	24	6.0	337	6%	7%
305	3.7	67	4.3	413	3%	8%
356	16.0	49	3.9	327	7%	16%
375	0.0	94	0.0	50	0%	0%
392	0.2	98	0	38	0%	2%
408	3.2	87	6.5	562	0%	45%
452	25.0	51	4.9	326	2%	22%
518	26.0	65	7	54	12%	6%

Header-flat

- Can often be read as flat data by assuming values from the header
- Can easily be written by hard-coding a useful header
  - BMP, WAV, AVI

Header-index-data

- Archive files
- Basic animations

Record sequence

- “Record” is Header-data
- TAR
- GIF
- PNG

Chunked files

- FBX, Blender files (3D modeling data)
- AVI, WebM, Matroska (Movies/films)
- (XML)
- Protobuf

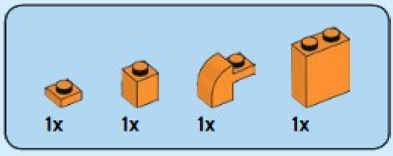
- Like records, but most chunks contain subchunks

How is the file to be used?

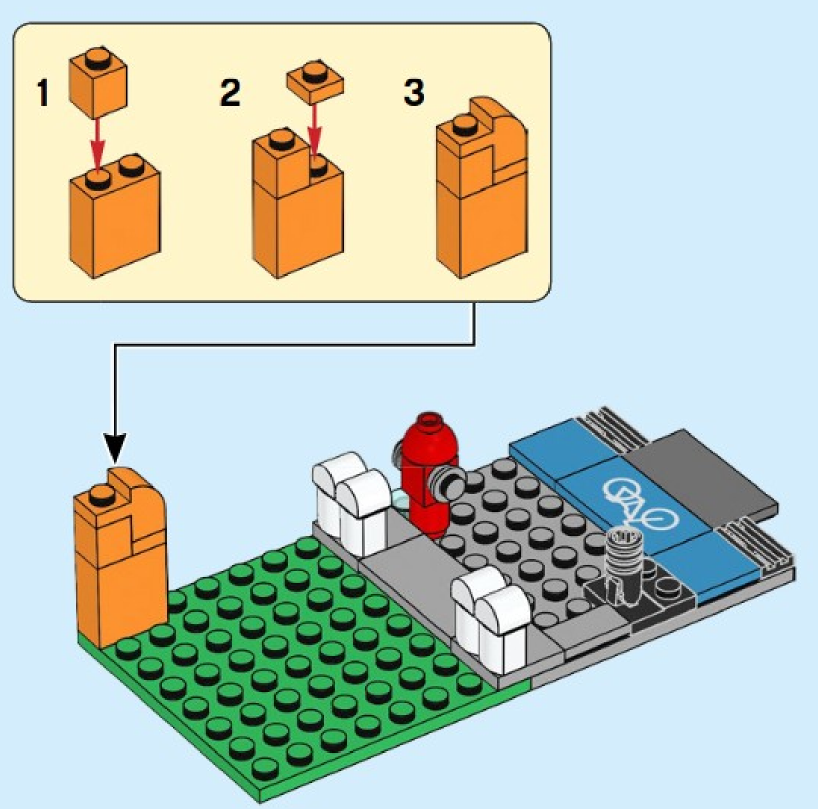
- Data arrives incrementally – streaming
  - DVD, internet video
- Data is huge but we only need parts of it – mappable
  - Databases, filesystems
- We need the whole thing to use it at all – loadable
  - Images, 3D models

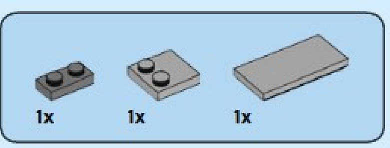
Step 1: Use cases

- What should the data be formatted to do optimally?
  - Data-driven algorithms are often data-speed-bounded
- Data in memory to run some operation is called the “working set”
- We want to make the “working set” for typical operations to be efficiently loadable

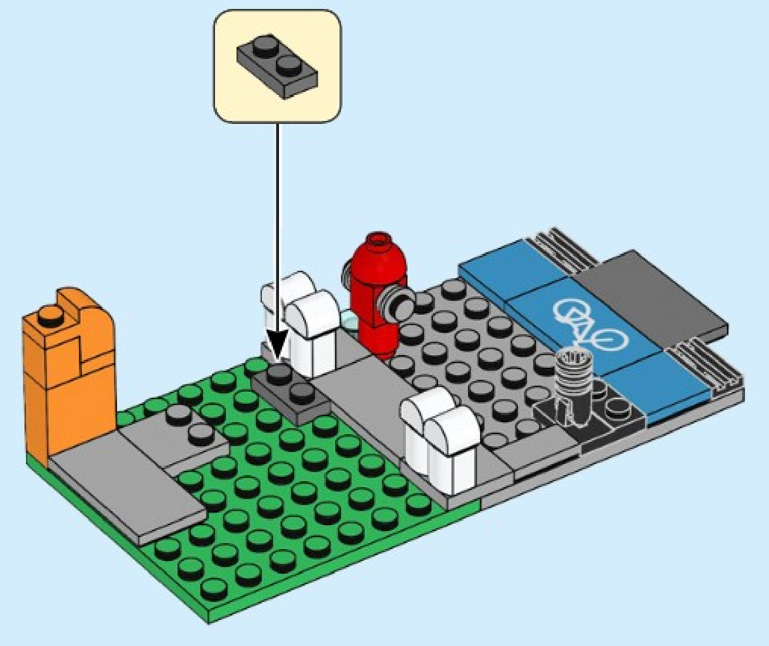


10





11



Caching

- Computers have a fixed amount of cache
- Optimizing for cache size makes the **effective** cache size bigger
  - More useful information in the cache
  - Or fitting all useful information in a smaller (and faster) cache (like L2 over L3).

Caching

- Caching includes main memory
  - Large files
  - Paging (4k – 16k – 64k)
- Caching includes the whole storage subsystem
  - Huge files

Step 1: Use cases

- Video viewing formats store audio along with the video frame they go with
  - Needed at the same time
- Video editing formats have a second image stream
  - Faster feedback while seeking

Step 2: Data model

- Relationships
- Multiplicities
- Sizes
- Counts
- Types

Step 2: Data model

- How much of something?
- Are there useful groupings?
- What indexes?

Step 3: Group bottom-up, index top-down

- Group information that is used together
  - These will share cache lines or pages
- Separate data that is not usually used together
- Store disparate (large) structures separately
- Add an index to find things in large structures
  - No data format can beat not even looking

Step 4: Create specific file format

- Often based on abstractions from pure binary provided by libraries
  - Avro, Protobuf, serialization libraries
- In some cases, creating a separate abstraction can be better
  - Arrays

Step 4: Create specific file format

- Prefer optimized data structures over generic code reuse
  - B+-trees (filesystems, databases)
  - radix trees (text indices)
  - Octree, quadtree, BSP (games, geospatial data)
  - Hash tables (ELF)

Step 4: Create specific file format

- Start the file with a magic value and a version number
  - Error path should be early and quick
- Big-endian, little-endian, endian-agnostic?
  - For directly mapped formats you need to match the endianness of the CPU
- Better option is to create view types

Step 5: Provide access

- Load or memory map the file
  - Consider MAP_LOCKED and MAP_POPULATE for servers
- Load from the file
  - Enough to know that it’s valid
  - Map indexes

Step 5: Provide access

- Provide view-style access to the indexes
  - Know where the data is, but do not load until required
- Depending on access patterns
  - Add caching for long-tail style data (some accessed often, many not accessed, overhead in loading)

Step 6: Iterate the design

- Create measurements to see how well it does in what you want it to do
- Create variations to compare specifics
- Check on target hardware!

Versioning

- Multiple readers
  - One per version
    - Works well, but duplicates lots of code
  - One unified one with tons of if statements
    - Spaghetti

Versioning

- Backward compatible
  - Unified reader with conditionals to handle old versions
  - Most commonly used formats have this
    - BMP, DOC, DOCX, ODT, PNG, JPG, EXE, TAR, ELF, ...

Versioning

- Backward and forward compatible
  - Protobuf – still needs lots of actual work to be used
  - Self-describing files: Avro, Blender
    - Does help if you get a random file without knowing the data model

Cheap
Compact
Fast

Talk overview

- The basics
- Step 1: Designing a binary file format
- **Step 2: Reverse engineering binary files**

Part 3: Reverse engineering a file format

- Why?
  - Many bits of software are used to encode and store data
  - Most of this software turns into abandonware, and the encoded data is unreadable by anyone
  - Happens to computer games a **lot**

CRUSADER
NO REMORSE ™
© 1996 Origin Systems, Inc.
All rights reserved
Press Start Button

Step 0: See if somebody else did this for you already

- An hour of googling can save you a month of work
- But where’s the fun in that?
- For Crusader: No Remorse,
  - ScummVM has an implementation that mostly works
  - Ultima8 (predecessor) is well-documented
  - I used both resources for reverse engineering, mostly from Ultima8 documentation (since it’s much more readable than source code)
    - The games differ enough that everything is slightly off and wrong

Step 1: Use the `file` tool to see if it knows

- Bitmaps!
  - cred.dat: PC bitmap, Windows 3.x format, 640 x 480 x 8, cbSize 308278, bits offset 1078
  - help1.dat: PC bitmap, Windows 3.x format, 640 x 480 x 8, image size 307200, 256 important colors, cbSize 308278, bits offset 1078
  - help2.dat: PC bitmap, Windows 3.x format, 640 x 480 x 8, image size 307200, 256 important colors, cbSize 308278, bits offset 1078
  - help3.dat: PC bitmap, Windows 3.x format, 640 x 480 x 8, image size 307202, cbSize 308280, bits offset 1078

GAME CONTROL

Key	Function
F1	HELP SCREEN
F2	OPTIONS SCREEN
F4	QUICK LOAD
F5	QUICK SAVE
F8	LOAD SCREEN
F9	SAVE SCREEN
+	INCREASE MUSIC VOLUME
-	DECREASE MUSIC VOLUME
R	TARGETING RETICLE ON/OFF
Esc	GAME OR MAIN MENU
Alt X	EXIT TO DOS

PLAYER INTERFACE

Key	Keypad	Function
Space	0	FIRE WEAPON
U	Del	USE INVENTORY ITEM
M		USE MEDIKIT
O	-	CYCLE THROUGH INVENTORY
I		REVERSE THROUGH INVENTORY
W	*	CYCLE THROUGH WEAPONS
Q		REVERSE THROUGH WEAPONS
Ctrl D		DROP WEAPON
S	+	SEARCH / SELECT
Enter		MANIPULATE SELECTED ITEM OR SPEAK TO SELECTED PERSON
G		GRAB EXPOSED ITEMS FOR INVENTORY
B		DETONATE BOMB
E		SHIELD ON/OFF
Z		CENTER CAMERA
Tab or J		JUMP

(▢ = NUMBER KEYPAD)

NEXT QUIT

KEYBOARD MOVEMENT

Key	Function
↑	WALK FORWARD
↓	RETREAT
←	TURN LEFT
→	TURN RIGHT
Shift ↑	RUN
Shift ←	QUICK TURN LEFT
Shift →	QUICK TURN RIGHT
Caps Lock	FOR CONSTANT SHIFT
Ctrl ↑	JUMP
Ctrl ↓	TOGGLE CROUCH
Ctrl ←	ROLL LEFT
Ctrl →	ROLL RIGHT
Alt ↑	ADVANCE
Alt ↓	RETREAT
Alt ←	SIDESTEP LEFT
Alt →	SIDESTEP RIGHT

KEYPAD MOVEMENT

Num Lock OFF

Key	Function
1 THRU 9	ABSOLUTE SCREEN MOVEMENT
5	TOGGLE CROUCH
Shift + KEY	ACCELERATE MOVEMENT
Caps Lock	FOR CONSTANT SHIFT
Ctrl + KEY	ROLL
Alt + KEY	SIDESTEP

Num Lock ON

Key	Function
/	ADVANCE
7	SIDE STEP LEFT
8	WALK FORWARD
9	SIDE STEP RIGHT
4	TURN LEFT
5	TOGGLE CROUCH
6	TURN RIGHT
1	ROLL LEFT
2	RETREAT
3	ROLL RIGHT
Ctrl 8	JUMP
Shift 8	RUN
Shift 4	QUICK TURN LEFT
Shift 6	QUICK TURN RIGHT

EXTENDED KEYS

Key	Function
INSERT	ROLL LEFT
HOME	ADVANCE
PAGE UP	ROLL RIGHT
DELETE	SIDE STEP LEFT
END	RETREAT
PAGE DOWN	SIDE STEP RIGHT

MOUSE MOVEMENT

TURN LEFT

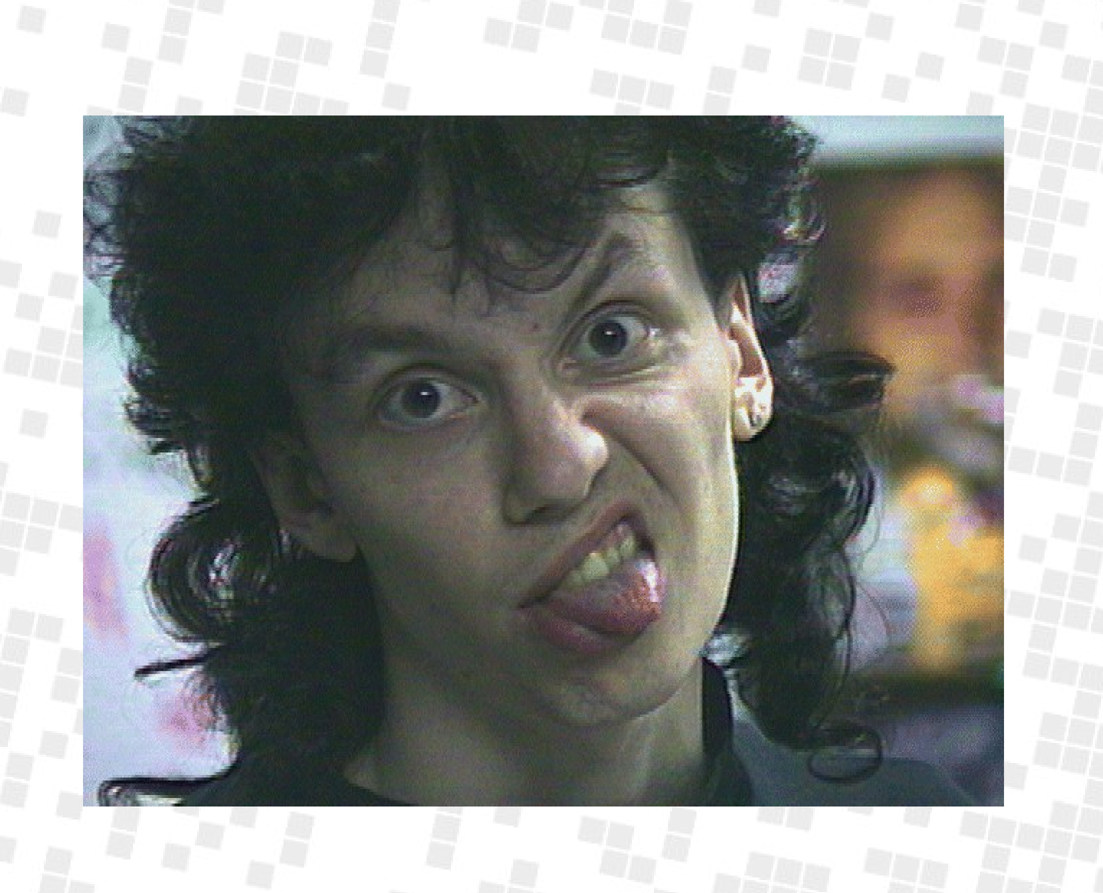
SHOOT

TURN RIGHT

ADVANCE

(▢ = NUMBER KEYPAD)

NEXT QUIT



Step 1: Use the `file` tool to see if it knows

- fixed.dat: DIY-Thermocam raw data (Lepton 2.x), scale -8346-24331, spot sensor temperature 0.000000, unit celsius, color scheme 0, calibration: offset 0.000000, slope 4017143577502862540 8.000000

Step 1: Use the `file` tool to see if it knows

- cred.pal: data
- diff.pal: data
- gamepal.pal: data
- misc2.pal: data
- misc.pal: data
- star.pal: data
- anim.dat: data
- combat.dat: data
- credits.dat: data
- stuff.dat: data
- trig.dat: data
- typeflag.dat: data
- wpnovlay.dat: data
- xformpal.dat: data
- damage.flx: data
- dtable.flx: data
- fonts.flx: data
- glob.flx: data
- gumps.flx: data
- shapes.flx: data

Step 2: Guess from the names

- Gamepal.pal: Game palette.
- Shapes.flx file: Game images? (90’s term used to refer to in-game sprites)
- Typeinfo.dat: Info about types?
- Trig.dat: Trigonometry
- Combat, damage, dtable: Damage and fighting information, I guess?
- Usecode.flx: Useful code? Still, “code” in a data file - interesting
- Glob.flx: Globs??
- Gumps.flx: Gumps?????
- Fixed.dat: uh…. ?
- Stuff.dat: Oh come on now.

Step 2.5: Use `strings` to look through files for text

- Usecode.flx has nearly all in-game strings
  - Looks and feels like executable code
  - Non-x86 – probably an in-game VM
    - Ultima8 has documentation on its usecode
- Credits.dat has the end-game credits

Step 3: Identify file structure

- Header size (if any)
- Index into remainder?
- Fixed-size records?
- Size-prefixed records?
  - Somehow the original software has to be able to read this too, and it's not usually hard-coded in the software.

```
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000010  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000020  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000030  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000040  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000050  1a 1a 00 00 00 0c 00 00  01 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  80 60 00 00 1a 00 00 00  9a 60 00 00 1a 00 00 00  |.`.......`......|
00000090  b4 60 00 00 1a 00 00 00  ce 60 00 00 32 00 00 00  |.`.......`..2...|
```

```
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000010  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000020  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000030  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000040  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000050  1a 1a 00 00 00 0c 00 00  01 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  80 60 00 00 1a 00 00 00  9a 60 00 00 1a 00 00 00  |.`.......`......|
00000090  b4 60 00 00 1a 00 00 00  ce 60 00 00 32 00 00 00  |.`.......`..2...|
```

```
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000010  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000020  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000030  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000040  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................|
00000050  1a 1a 00 00 00 0c 00 00  01 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  80 60 00 00 1a 00 00 00  9a 60 00 00 1a 00 00 00  |.`.......`......|
00000090  b4 60 00 00 1a 00 00 00  ce 60 00 00 32 00 00 00  |.`.......`..2...|
```

```
00000080  80 60 00 00 1a 00 00 00  9a 60 00 00 1a 00 00 00  |.`.......`......|
00000090  b4 60 00 00 1a 00 00 00  ce 60 00 00 32 00 00 00  |.`.......`..2...|

00006050  22 09 04 00 3e 00 00 00  60 09 04 00 62 00 00 00  |"...>...`...b...|
00006060  c2 09 04 00 56 00 00 00  18 0a 04 00 50 00 00 00  |....V.......P...|
00006070  68 0a 04 00 26 00 00 00  8e 0a 04 00 26 00 00 00  |h...&.......&...|
00006080  04 00 7f 7f 00 0f 00 00  ff 7f 00 0f 00 00 7f ff  |................|
00006090  00 0f 00 00 ff ff 00 0f  00 00 04 00 7f 7f 00 2a  |...............*|

0x6080 – 0x80 == 0x6000
0x6000 / 8 == 0xC00
```

```
00006050  22 09 04 00 3e 00 00 00  60 09 04 00 62 00 00 00  |"...>...`...b...|
00006060  c2 09 04 00 56 00 00 00  18 0a 04 00 50 00 00 00  |....V.......P...|
00006070  68 0a 04 00 26 00 00 00  8e 0a 04 00 26 00 00 00  |h...&.......&...|
00006080  04 00 7f 7f 00 0f 00 00  ff 7f 00 0f 00 00 7f ff  |................|
00006090  00 0f 00 00 ff ff 00 0f  00 00 04 00 7f 7f 00 2a  |...............*|

0x6080 – 0x80 == 0x6000
0x6000 / 8 == 0xC00

00000050  1a 1a 00 00 00 0c 00 00  01 00 00 00 00 00 00 00  |................|
```

```
00006070  68 0a 04 00 26 00 00 00  8e 0a 04 00 26 00 00 00  |h...&.......&...|
00006080  04 00 7f 7f 00 0f 00 00  ff 7f 00 0f 00 00 7f ff  |................|
00006090  00 0f 00 00 ff ff 00 0f  00 00 04 00 7f 7f 00 2a  |...............*|

0x40a68 + 0x26 == 0x40a8e
0x40a8e + 0x26 == 0x40ab4

00040a80  00 00 ff ff 00 18 04 01  ff ff 60 19 04 01 06 00  |..........`.....|
00040a90  7f ff 00 0f 00 00 7f 7f  00 0f 00 00 ff 7f 00 0f  |................|
00040aa0  00 00 ff ff 00 0f 00 00  ff 7f 00 18 04 00 ff 7f  |................|
00040ab0  60 19 04 00                                       |`...|
```

FLX file: Archive file

- Now we can decode the *.flx files!
- But actually...

FLX file: Archive file

```
00000000  00 00 00 00 00 20 00 00  00 00 00 00 00 00 00 00  |..... ..........| anim.dat
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................| combat.dat
00000000  21 b4 00 5e 40 44 49 52  45 43 54 45 44 20 42 59  |!..^@DIRECTED BY| credits.dat
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................| fixed.dat
00000000  14 00 57 30 31 00 00 00  00 00 2e 03 00 00 01 00  |..W01...........| stuff.dat
00000000  00 00 00 00 47 06 00 00  8e 0c 00 00 d2 12 00 00  |....G...........| trig.dat
00000000  00 00 00 01 00 00 00 00  01 02 60 60 0c 05 00 04  |..........``....| typeflag.dat
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................| wpnovlay.dat
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................| xformpal.dat
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................| damage.flx
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................| dtable.flx
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................| fonts.flx
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................| glob.flx
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................| gumps.flx
00000000  1a 1a 1a 1a 1a 1a 1a 1a  1a 1a 1a 1a 1a 1a 1a 1a  |................| shapes.flx
```

FLX file: Archive file

- Now we can decode the *.flx files!
- Write tool to extract chunks from FLX files
  - Continue analyzing the chunks found in there

4. Spot patterns for fixed size entries

```
00004530  03 00 20 84 00 00 00 00  00 03 00 20 84 00 00 00  |.. ....... ....|
00004540  00 00 03 00 20 84 00 00  00 00 00 03 00 20 84 00  |.... ....... ..|
00004550  00 00 00 00 03 00 20 84  00 00 00 00 00 03 00 20  |...... ....... |
00004560  84 00 00 00 00 00 03 00  20 84 00 00 00 00 00 03  |....... .......|
00004570  00 20 84 00 00 00 00 00  03 00 20 84 00 00 00 00  |. ....... .....|
00004580  00 03 00 20 84 00 00 00  00 00 03 00 20 84 00 00  |... ....... ...|
00004590  00 00 00 03 00 20 84 00  00 00 00 00 03 00 20 84  |..... ....... .|
000045a0  00 00 00 00 00 03 00 20  84 00 00 00 00 00 03 00  |....... .......|
000045b0  20 84 00 00 00 00 00 03  00 20 84 00 00 00 00 00  | ....... ......|
000045c0  03 00 20 84 00 00 00 00  00 03 00 20 84 00 00 00  |.. ....... ....|
000045d0  00 00 03 00 20 84 00 00  00 00 00 03 00 20 84 00  |.... ....... ..|
000045e0  00 00 00 00 03 00 20 84  00 00 00 00 00 03 00 20  |...... ....... |
000045f0  84 00 00 00 00 00 03 00  20 84 00 00 00 00 00 03  |....... .......|
00004600  00 20 84 00 00 00 00 00  03 00 20 84 00 00 00 00  |. ....... .....|
00004610  00 03 00 20 84 00 00 00  00 00 03 00 20 84 00 00  |... ....... ...|
00004620  00 00 00 03 00 20 84 00  00 00 00 00 03 00 20 84  |..... ....... .|
```

4. Spot patterns for fixed size entries

```
00004530  03 00 20 84 00 00 00 00  00 03 00 20 84 00 00 00  |.. ........ ....|
00004540  00 00 03 00 20 84 00 00  00 00 00 03 00 20 84 00  |.... ........ ..|
00004550  00 00 00 00 03 00 20 84  00 00 00 00 00 03 00 20  |...... ........ |
00004560  84 00 00 00 00 00 03 00  20 84 00 00 00 00 00 03  |........ .......|
00004570  00 20 84 00 00 00 00 00  03 00 20 84 00 00 00 00  |. ........ .....|
00004580  00 03 00 20 84 00 00 00  00 00 03 00 20 84 00 00  |... ........ ...|
00004590  00 00 00 03 00 20 84 00  00 00 00 00 03 00 20 84  |..... ........ .|
000045a0  00 00 00 00 00 03 00 20  84 00 00 00 00 00 03 00  |....... .......|
000045b0  20 84 00 00 00 00 00 03  00 20 84 00 00 00 00 00  | ........ ......|
000045c0  03 00 20 84 00 00 00 00  00 03 00 20 84 00 00 00  |.. ........ ....|
000045d0  00 00 03 00 20 84 00 00  00 00 00 03 00 20 84 00  |.... ........ ..|
000045e0  00 00 00 00 03 00 20 84  00 00 00 00 00 03 00 20  |...... ........ |
000045f0  84 00 00 00 00 00 03 00  20 84 00 00 00 00 00 03  |........ .......|
00004600  00 20 84 00 00 00 00 00  03 00 20 84 00 00 00 00  |. ........ .....|
00004610  00 03 00 20 84 00 00 00  00 00 03 00 20 84 00 00  |... ........ ...|
00004620  00 00 00 03 00 20 84 00  00 00 00 00 03 00 20 84  |..... ........ .|
```

4. Spot patterns for fixed size entries

```
00000000  00 00 00 01 00 00 00 00  01                    |.........|
00000009  02 60 60 0c 05 00 04 32  32                    |.``....22|
00000012  0b 00 00 05 06 10 00 00  00                    |.........|
0000001b  03 00 00 21 00 00 00 00  00                    |...!.....|
00000024  0b 00 c0 84 00 00 00 00  00                    |.........|
0000002d  03 10 c0 84 05 00 00 00  00                    |.........|
00000036  0b 00 00 06 06 00 00 00  00                    |.........|
0000003f  0b 00 20 40 06 00 00 00  00                    |.. @.....|
00000048  0b 00 00 06 06 00 00 00  00                    |.........|
00000051  0b 00 20 40 06 00 00 00  00                    |.. @.....|
0000005a  0b 00 20 08 06 00 00 00  00                    |.. ......|
00000063  0b 00 80 10 00 00 00 00  00                    |.........|
0000006c  0b 00 00 06 06 00 00 00  00                    |.........|
00000075  0b 00 20 40 06 00 00 00  00                    |.. @.....|
0000007e  0b 00 20 10 06 00 00 00  00                    |.. ......|
00000087  03 00 00 42 00 00 00 00  00                    |...B.....|
```

Typeinfo.dat:

- Fixed size entry. 9 bytes each
- File is 18432 bytes
  - Exactly 2048 * 9
- Looks like 2048 entries without a header
  - Shapes.flx also has 2048 entries
- Each entry is bitpacked type data per shape!

Fixed.dat (entry 0)

```
00000000  ff 03 ff 03 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000010  ff 05 ff 03 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000020  ff 07 ff 03 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000030  ff 09 ff 03 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000040  ff 0b ff 03 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000050  ff 0d ff 03 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000060  ff 03 ff 05 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000070  fc 04 10 04 10 e9 01 00  00 00 00 00 00 00 71 1a  |..............q.|
00000080  ff 05 ff 05 00 10 00 00  00 00 8e 0b 00 08 72 1a  |..............r.|
00000090  44 05 44 05 00 88 01 00  00 00 00 00 00 00 ca 1a  |D.D.............|
000000a0  37 05 35 05 08 db 01 00  00 00 0c 00 00 00 00 00  |7.5.............|
000000b0  15 07 0f 04 10 e9 01 01  00 00 00 00 00 00 77 1a  |..............w.|
000000c0  eb 07 e9 04 00 27 05 00  00 04 00 00 00 00 78 1a  |.....'........x.|
000000d0  ff 07 7f 04 00 26 05 00  00 04 00 00 00 00 79 1a  |.....&........y.|
```

Fixed.dat

- No header
- Fixed size entry
  - 16 bytes

Fixed.dat (entry 0)

- Guess a field
- Check the values to see if we can find errors
  - Lather, rinse, repeat. Trial and error.
- Gets us to this

Fixed.dat (entry 0)

```
00000000  ff 03 ff 03 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000010  ff 05 ff 03 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000020  ff 07 ff 03 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000030  ff 09 ff 03 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000040  ff 0b ff 03 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000050  ff 0d ff 03 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000060  ff 03 ff 05 00 10 00 00  00 00 02 00 00 00 00 00  |................|
00000070  fc 04 10 04 10 e9 01 00  00 00 00 00 00 00 71 1a  |..............q.|
00000080  ff 05 ff 05 00 10 00 00  00 00 8e 0b 00 08 72 1a  |..............r.|
00000090  44 05 44 05 00 88 01 00  00 00 00 00 00 00 ca 1a  |D.D.............|
000000a0  37 05 35 05 08 db 01 00  00 00 0c 00 00 00 00 00  |7.5.............|
000000b0  15 07 0f 04 10 e9 01 01  00 00 00 00 00 00 77 1a  |..............w.|
000000c0  eb 07 e9 04 00 27 05 00  00 04 00 00 00 00 78 1a  |.....'........x.|
000000d0  ff 07 7f 04 00 26 05 00  00 04 00 00 00 00 79 1a  |.....&........y.|
```

Fixed.dat

- No header, fixed size entry, 16 bytes
- But by far most of them are type 0x10?

Fixed.dat (entry 0)

```
00010e70  ff 55 ff 49 00 10 00 00  00 00 02 00 00 00 00 00  |.U.I............|
00010e80  ff 57 ff 49 00 10 00 00  00 00 02 00 00 00 00 00  |.W.I............|
00010e90  ff 59 ff 49 00 10 00 00  00 00 f5 0a 00 00 00 00  |.Y.I............|
00010ea0  ff 5b ff 49 00 10 00 00  00 00 fb 0a 00 00 00 00  |.[.I............|
00010eb0  ff 5d ff 49 00 10 00 00  00 00 fc 0a 00 00 00 00  |.].I............|
00010ec0  ff 5f ff 49 00 10 00 00  00 00 fd 0a 00 00 00 00  |._.I............|
00010ed0  ff 71 ff 49 58 10 00 00  00 00 02 00 00 00 00 00  |.q.IX...........|
00010ee0  ff 73 ff 49 58 10 00 00  00 00 02 00 00 00 00 00  |.s.IX...........|
00010fe0  ff 75 ff 49 60 10 00 00  00 00 70 0b 00 00 16 16  |.u.I`.....p.....|
000111a0  ff 77 ff 49 60 10 00 00  00 00 75 0b 00 00 21 0a  |.w.I`.....u...!.|
00011230  ff 79 ff 49 60 10 00 00  00 00 02 00 00 00 00 00  |.y.I`...........|
00011240  ff 35 ff 4b 00 10 00 00  00 00 01 07 64 08 00 00  |.5.K........d...|
00011250  ff 45 ff 4b 00 10 00 00  00 00 02 00 64 08 00 00  |.E.K........d...|
```

Fixed.dat

- No header, fixed size entry, 16 bytes
- But by far most of them are type 0x10?
  - Basic compression
  - 0x10 entries are referring to globs
  - The yellow values indicate which glob

```
00000000  04 00 7f ff 00 50 01 00  ff ff 00 50 01 00 ff 7f  |.....P.....P....|
00000010  00 50 01 00 7f 7f 00 50  01 00                    |.P.....P..|
26 byte file, starts with 2-byte value 0x04

00000000  0a 00 7f 7f 60 14 00 00  ff 7f 60 14 00 00 7f ff  |....`.....`.....|
00000010  60 14 00 00 ff ff 60 14  00 00 07 7f 60 09 00 00  |`.....`.....`...|
00000020  07 ff 60 09 00 00 77 7f  60 09 00 00 77 ff 60 09  |..`...w.`...w.`.|
00000030  00 00 ff 7f 60 09 00 00  ff ff 60 09 00 00        |....`.....`...|
62 byte file, starts with 2-byte value 0x0a

00000000  06 00 ff 7f 00 44 02 00  ff ff 00 44 02 00 7f ff  |.....D.....D....|
00000010  00 3d 02 00 7f 7f 00 3d  02 00 ff ff 00 3d 02 00  |.=.....=.....=..|
00000020  ff 7f 00 3d 02 00                                 |...=..|
38 byte file, starts with 2-byte value 0x06
```

```
00000000  04 00 7f ff 00 50 01 00  ff ff 00 50 01 00 ff 7f  |.....P.....P....|
00000010  00 50 01 00 7f 7f 00 50  01 00                    |.P.....P..|
26 byte file, starts with 2-byte value 0x04, file size == 2 + 6 * 4

00000000  0a 00 7f 7f 60 14 00 00  ff 7f 60 14 00 00 7f ff  |....`.....`.....|
00000010  60 14 00 00 ff ff 60 14  00 00 07 7f 60 09 00 00  |`.....`.....`...|
00000020  07 ff 60 09 00 00 77 7f  60 09 00 00 77 ff 60 09  |..`...w.`...w.`.|
00000030  00 00 ff 7f 60 09 00 00  ff ff 60 09 00 00        |....`.....`...|
62 byte file, starts with 2-byte value 0x0a, file size == 2 + 6 * 10

00000000  06 00 ff 7f 00 44 02 00  ff ff 00 44 02 00 7f ff  |.....D.....D....|
00000010  00 3d 02 00 7f 7f 00 3d  02 00 ff ff 00 3d 02 00  |.=.....=.....=..|
00000020  ff 7f 00 3d 02 00                                 |...=..|
38 byte file, starts with 2-byte value 0x06, file size == 2 + 6 * 6
```

Globs.flx entry:

- Two byte header containing entry count
- 6-byte entries

```
7f ff 00 50 01 00
ff ff 00 50 01 00
ff 7f 00 50 01 00
7f 7f 00 50 01 00

ff 7f 00 44 02 00
ff ff 00 44 02 00
7f ff 00 3d 02 00
7f 7f 00 3d 02 00
ff ff 00 3d 02 00
ff 7f 00 3d 02 00
```

Shapes.flx.452

```
00000000  01 00 01 00 08 00 46 00  00 80 8e 04 00 00 dc 04  |......F.........|
00000010  00 80 9c 04 00 00 80 09  00 80 c6 04 00 00 4e 0e  |..............N.|
00000020  00 80 9c 04 00 00 f2 12  00 80 9c 04 00 00 96 17  |................|
00000030  00 80 d4 04 00 00 72 1c  00 80 f0 04 00 00 6a 21  |......r.......j!|
00000040  00 80 b8 04 00 00 c5 01  00 00 5e c6 9e 00 01 00  |..........^.....|
00000050  00 00 10 00 00 00 68 00  00 00 07 00 00 00 67 00  |......h.......g.|
00000060  00 00 a0 01 00 00 a1 01  00 00 a1 01 00 00 a1 01  |................|
00000070  00 00 a1 01 00 00 a4 01  00 00 a9 01 00 00 b5 01  |................|
```

Shapes.flx.452

```
01 00 01 00 08 00
46 00 00 80 8e 04 00 00     0x46 + 0x48e == 0x4d4
dc 04 00 80 9c 04 00 00     0x4dc + 0x49c == 0x978
80 09 00 80 c6 04 00 00     0x980 + 0x4c6 == 0xe46
4e 0e 00 80 9c 04 00 00     0xe4e + 0x49c == 0x12ea
f2 12 00 80 9c 04 00 00     0x12f2 + 0x49c == 0x178e
96 17 00 80 d4 04 00 00     0x1796 + 0x4d4 == 0x1c6a
72 1c 00 80 f0 04 00 00     0x1c72 + 0x4f0 == 0x2162
6a 21 00 80 b8 04 00 00     0x216a + 0x4b8 == 0x2622
Offset and size?
```

Shapes.flx entry:

- Frame data
- Per frame “subfile”
- Line data in RLE style

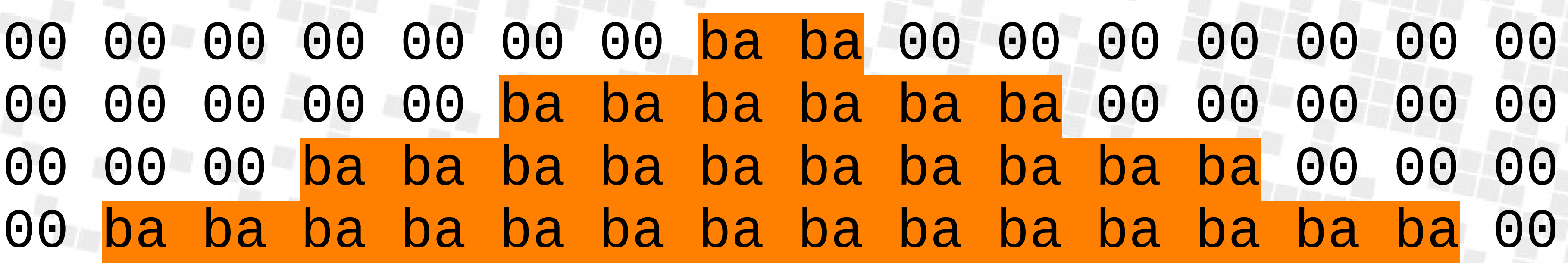
```
00000046  c5 01 00 00 5e c6 9e 00  01 00 00 00 10 00 00 00  |....^...........|
00000056  68 00 00 00 07 00 00 00  67 00 00 00 a0 01 00 00  |h.......g.......|
00000066  a1 01 00 00 a1 01 00 00  a1 01 00 00 a1 01 00 00  |................|

000001f6  c3 02 00 00 cc 02 00 00  d1 02 00 00 07 04 ba ba  |................|
00000206  07 05 0d ba 05 03 15 ba  03 01 1d ba 01 00 04 bb  |................|
00000216  bb 00 1d ba 00 08 bc bc  4d bb 00 19 ba 00 09 bc  |........M.......|
00000226  00 04 4d bb 00 0d ba 00  08 24 ba ba ba 00 0d bc  |..M......$......|
00000236  00 14 4d bb ba ba bb bb  ba ba ba bb 00 0f bc 00  |..M.............|
```

```
00000046  c5 01 00 00 5e c6 9e 00  01 00 00 00 10 00 00 00  |....^...........|
00000056  68 00 00 00 07 00 00 00  67 00 00 00 a0 01 00 00  |h.......g.......|
00000066  a1 01 00 00 a1 01 00 00  a1 01 00 00 a1 01 00 00  |................|

000001f6  c3 02 00 00 cc 02 00 00  d1 02 00 00 07 04 ba ba  |................|
00000206  07 05 0d ba 05 03 15 ba  03 01 1d ba 01 00 04 bb  |................|
00000216  bb 00 1d ba 00 08 bc bc  4d bb 00 19 ba 00 09 bc  |........M.......|
00000226  00 04 4d bb 00 0d ba 00  08 24 ba ba ba 00 0d bc  |..M......$......|
00000236  00 14 4d bb ba ba bb bb  ba ba ba bb 00 0f bc 00  |..M.............|
```

```
07 04 ba ba 07
    Skip 7 pixels, copy 2 values (0xba 0xba), skip 7
05 0d ba 05
    Skip 5 pixels, put 6 copies of 0xba, skip 5
03 15 ba 03
    Skip 3 pixels, put 10 copies of 0xba, skip 3
01 1d ba 01
    Skip 1 pixel, put 14 copies of 0xba, skip 1
```


00 00 00 00 00 00 00 ba ba 00 00 00 00 00 00 00
00 00 00 00 00 ba ba ba ba ba ba 00 00 00 00 00
00 00 00 ba ba ba ba ba ba ba ba ba ba 00 00 00
00 ba ba ba ba ba ba ba ba ba ba ba ba ba ba 00

Let’s draw shapes!

Putting it all together

- Implement the bits that can show you if you went wrong
  - Keep in mind that some games have false-positives
  - Some files may be corrupt but unused
- The more constrained, the better.

Crusader: No Remorse

- Render a whole level
  - Shape drawing
  - x/y/z coordinate details
  - Glob unpacking
  - Typeinfo (up to a point)

CHEESE

Steps beyond this:

- Try to identify headers of known file formats
  - Zlib compressed data is a common one
  - Even inside of other files
    - Can hint at other offsets to find
- Try to identify offset / size pairs, or indexes into the data
  - Write a program to check these

Steps beyond this:

- Find files with the same format, but different data to identify what fields represent
- Try to fully identify small files or byte regions, and then find out how larger files differ
- Look for other file formats from the same era for indications on how people solved similar problems then
  - Many people in the same era will (inadvertently) choose the same kind of solution

Live reverse engineering

- Debugging original executables
- This is illegal in some places
  - if the laws say that doing so for compatibility is legal, you can still get sued
- YMMV
- I choose not to

Thanks to

- Moritz Mertinkat and Gavin Pugh
  - For reverse-engineering Red Alert and Command&Conquer and pulling me into reverse engineering

Thanks to

- Origin, Jason Ely, Eric Willmar and the whole Crusader team
  - For making the awesome game that crashed on my computer, and that pulled me into C++

Thanks to

- Echosector.com
- Stauff for doing most of the work already & implementing it in ScummVM
- Keenan for keeping the site and community running for so long

Questions?

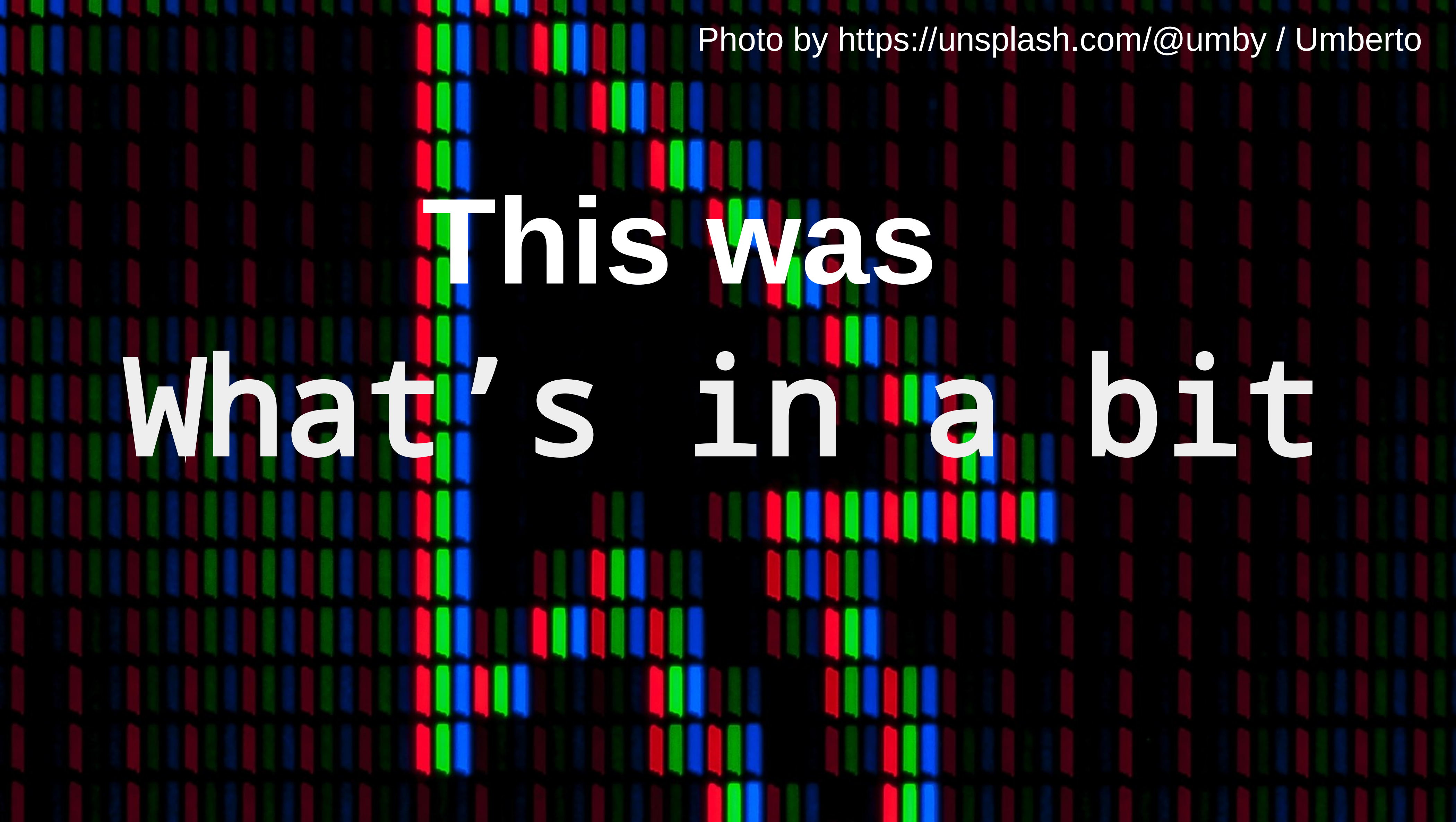


Photo by https://unsplash.com/@umby / Umberto
This was
What’s in a bit