NOT SCARY BINARY

WHO IS HAL POMERANZ?

Professional computer wrangler since 1985

Independent consultant since 1997

Digital forensics, incident response, expert witness

Don't tell anybody but my degree is in Math (minor in Comp Sci)

hrpomeranz@gmail.com

@hal_pomeranz@infosec.exchange

WHO NEEDS BINARY?

Programmers/Exploit Developers

Malware reverse engineers

Forensic Analysts

Threat Hunters

SOC Analysts

System and Network Admins

WHY DO COMPUTERS USE BINARY?

In the history of computing, information has been represented by:

- Vacuum tubes (On/Off)
- Mechanical relays (Open/Closed)
- Punch cards (Punched/Not Punched)
- Magnets (Polarity Positive/Negative)
- Transistors (Voltage High/Low)

It is natural for computers to use a system that tracks two states

WHAT DID WE LEARN IN GRADE SCHOOL?

$$\begin{tikzcd}
1 \arrow[d] & 3 \arrow[d] & 3 \arrow[d] & 7 \arrow[d] \\
\text{"thousands"} \arrow[d] & \text{"hundreds"} \arrow[d] & \text{"tens"} \arrow[d] & \text{"ones"} \arrow[d] \\
10^3 & 10^2 & 10^1 & 10^0
\end{tikzcd}$$

BINARY FOLLOWS THE SAME PATTERN

$$\begin{tikzcd}
1 \arrow[d] & 0 \arrow[d] & 1 \arrow[d] & 1 \arrow[d] \\
2^3 \arrow[d] & 2^2 \arrow[d] & 2^1 \arrow[d] & 2^0 \arrow[d] \\
\text{"eights"} & \text{"fours"} & \text{"twos"} & \text{"ones"}
\end{tikzcd}$$

EXAMPLE – CONVERT BINARY TO DECIMAL

$$\begin{tikzcd} 1 \arrow[d] & & 0 \arrow[d] & & 1 \arrow[d] & & 1 \arrow[d] \\ 8 & + & 0 & + & 2 & + & 1 \end{tikzcd}$$

$$\boxed{= 11}$$

GO BIG OR GO HOME

$$\begin{tikzcd}
1 \arrow[d] & & 0 \arrow[d] & & 1 \arrow[d] & & 1 \arrow[d] & & 1 \arrow[d] & & 1 \arrow[d] & & 0 \arrow[d] & & 1 \arrow[d] \\
128 & + & 0 & + & 32 & + & 16 & + & 8 & + & 4 & + & 0 & + & 1
\end{tikzcd}$$

$$\boxed{= 189}$$

CONVERT 243 TO BINARY

		What's left?
Is it bigger than or equal to 128?	YES!	115
Is it bigger than or equal to 64?	YES!	51
Is it bigger than or equal to 32?	YES!	19
Is it bigger than or equal to 16?	YES!	3
Is it bigger than or equal to 8?	NO!	3
Is it bigger than or equal to 4?	NO!	3
Is it bigger than or equal to 2?	YES!	1

1	1	1	1	0	0	1	1
128	64	32	16	8	4	2	1

NOT SCARY PRACTICE #1

YOU CAN DO THIS!

HOW BIG CAN YOU GET?

# Bits	Max Value	AKA
2	1 + 2 = 3	4 – 1 = 2^2 - 1
3	1 + 2 + 4 = 7	8 – 1 = 2^3 - 1
4	1 + 2 + 4 + 8 = 15	2^4 - 1
…		
8	255	2^8 - 1
…		
16	65,535	2^{16} - 1
…		
32	4,294,967,295	2^{32} - 1

SUPPOSE I WANT TO BE NEGATIVE?

Two's complement is used to represent negative values

The biggest bit is treated as *negative*, all other bits are added to it

These are called *signed* values (as opposed to *unsigned*)

	1		0		1		1	
Unsigned	8	+	0	+	2	+	1	= 11
Signed	-8	+	0	+	2	+	1	= -5

WHAT'S YOUR SIGN?

	Unsigned	Signed
0000	0	0
1000	8	-8
0010	2	2
0111	7	7
1110	14	-2
1111	15	-1

Most negative value possible: -2^{4-1}

Biggest positive value possible: $2^{4-1}-1$

Always -1, no matter how many bits

KNOW YOUR LIMITS

Given *N* bits:

Unsigned values range from 0 … $(2^N$-1)

Signed values range from -2^{N-1} … $(2^{N-1}$-1)

# Bits	Unsigned	Signed
4	0 … 15	-8 … 7
8	0 … 255	-128 … 127
16	0 … 65,535	-32,768 … 32,767
32	0 … 4,294,967,295	-2,147,483,648 … 2,147,483,647

NOT SCARY PRACTICE #2

SIGN ME UP!

WHAT THE HEX?

Hexadecimal is base 16 notation– each digit ranges from 0 … 15

We use letters for the digits from 10 … 15

10 = A	13 = D
11 = B	14 = E
12 = C	15 = F

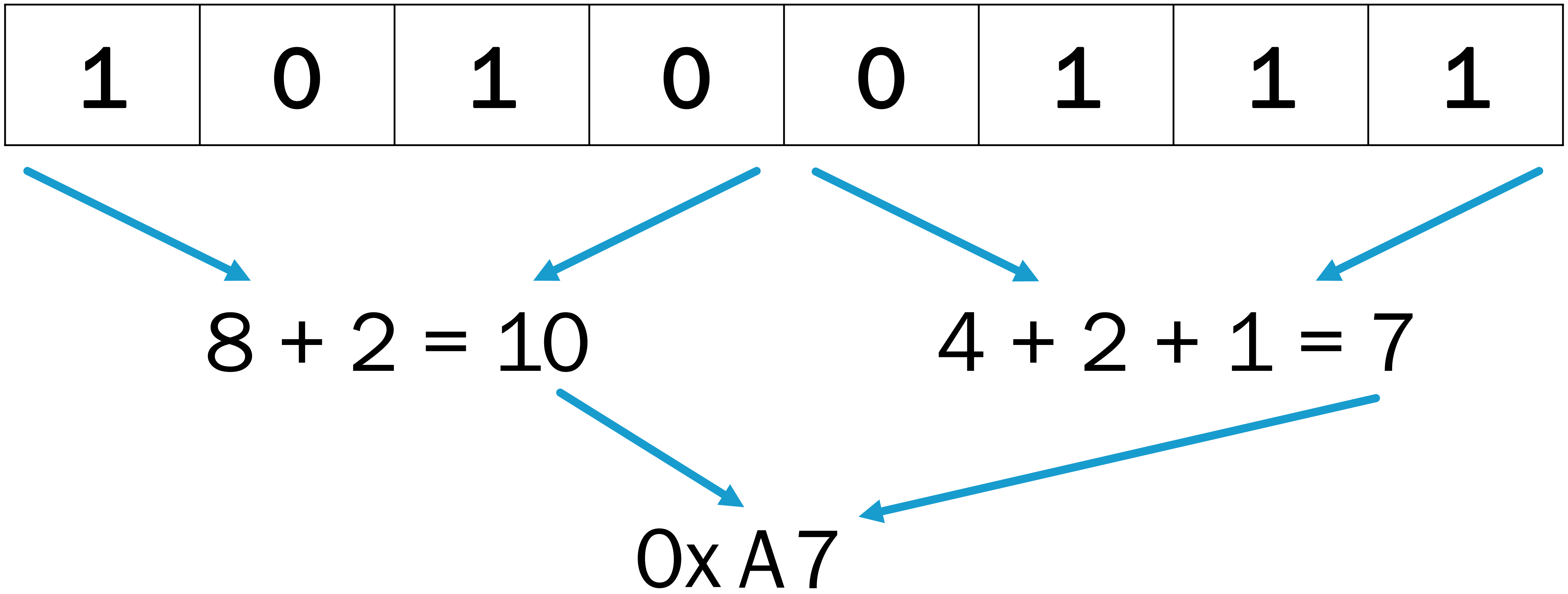
I THOUGHT THIS WAS A COURSE ON BINARY?

What else ranges from 0 … 15? Four-bit values!

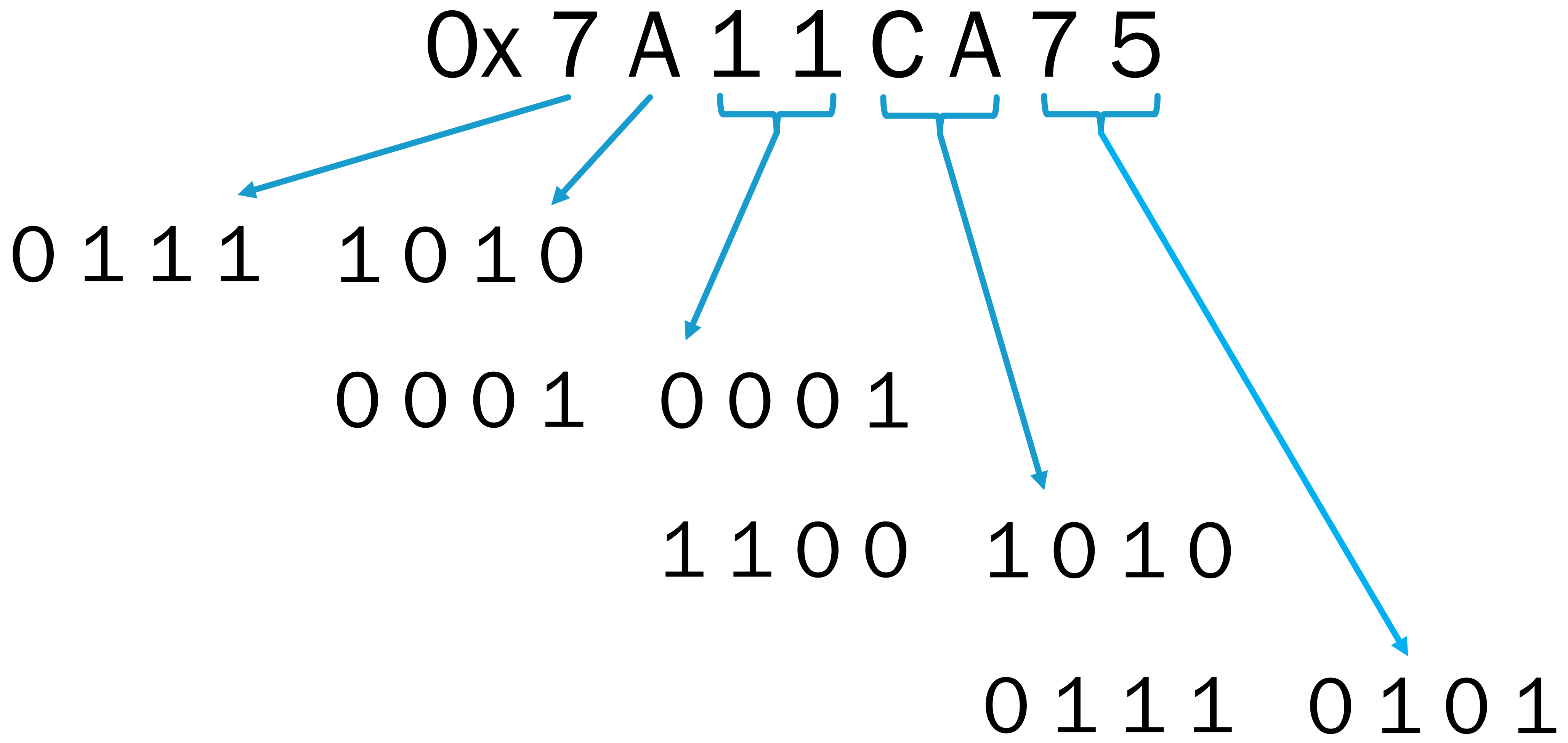
Each byte can be represented as two four-bit values (nybbles)

We can use two hex digits for a more compact byte representation

BINARY TO HEX



AND BACK AGAIN



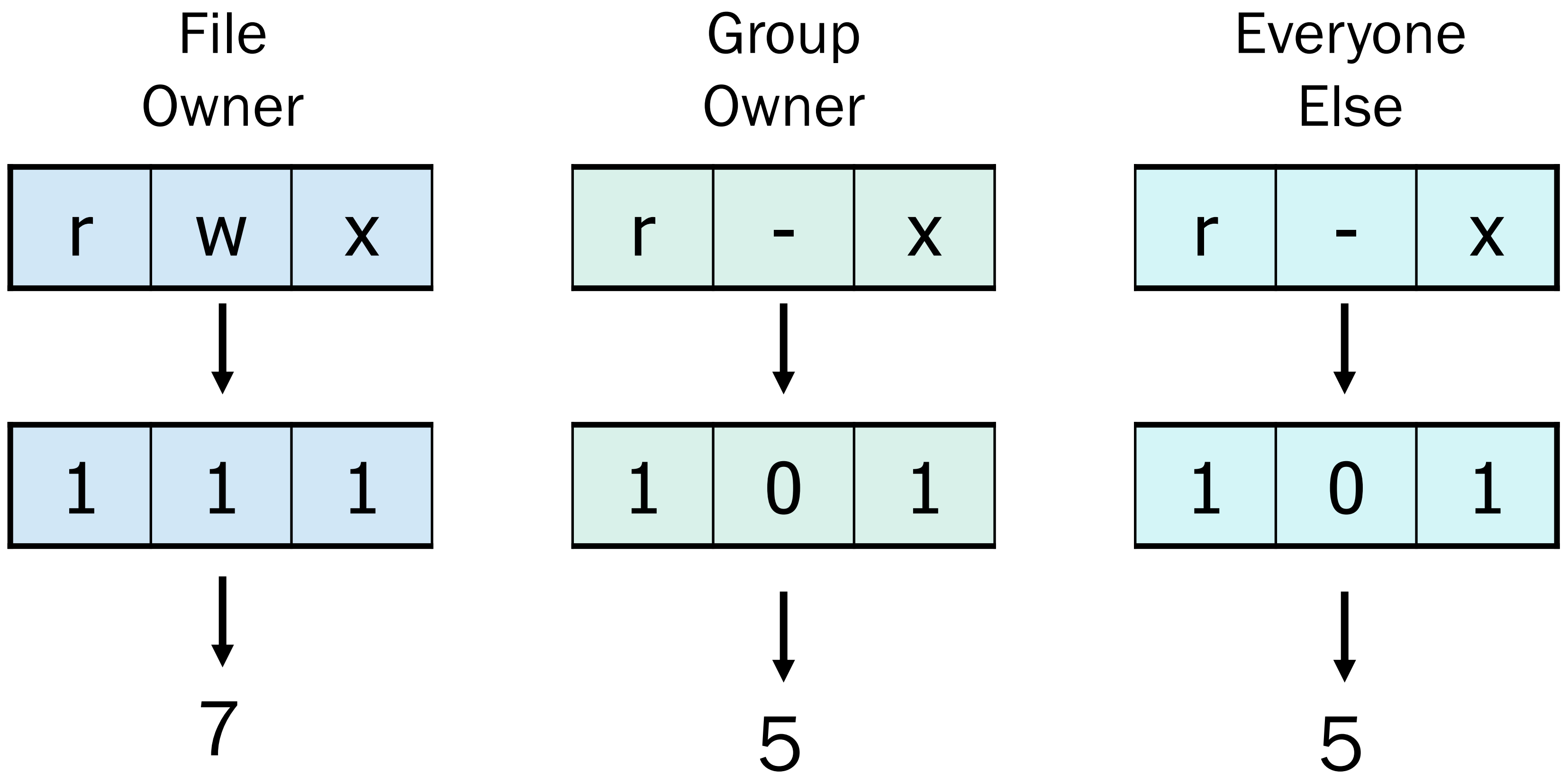
BINARY'S POORER COUSIN

Octal is base 8 notation– each digit ranges from 0 … 7

Each octal digit can represent three bits

Only ever comes up in one use case

LINUX/UNIX FILE PERMISSIONS



NOT SCARY PRACTICE #3

BE SURE TO COVER ALL YOUR BASES

WHY IS A BYTE EIGHT BITS?

You need at least 7 bits for English letters, numbers, and punctuation

Eight is the next higher power of two

Also a bunch of history from the 1960s and 1970s:

- AT&T used 8-bit encoding for early digital telephony
- Mainframes with 16-bit processors became common
- IBM introduces EBCDIC
- 8/16/32-bit microprocessors happen (1972/1976/1979)

WORDS, WORDS, WORDS

Word size originally meant the data width of a given CPU

16-bit computers were around a loooooong time

These days, (assembly) programmers talk about:

- *Bytes* – 8-bit values
- *Words* – 16-bit values
- *DWords* – 32-bit values
- *QWords* – 64-bit values

BACK TO GRADE SCHOOL

Numbers are commonly written left to right

The bigger numbers are the digits to the left and smaller on the right

"Big end" "Little end"

Decimal: 2,047,986,293

Hex: 0x7A 11 CA 75

"Big end" "Little end"

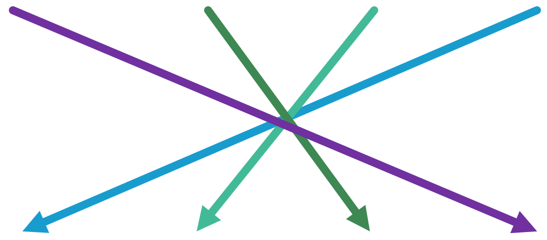
COMPUTERS ARE WEIRD

Humans generally use *big endian* ordering for numbers

Intel-family processors use *little endian*– byte order is reversed!

1 2 3 4

Big endian: 0x 7A 11 CA 75



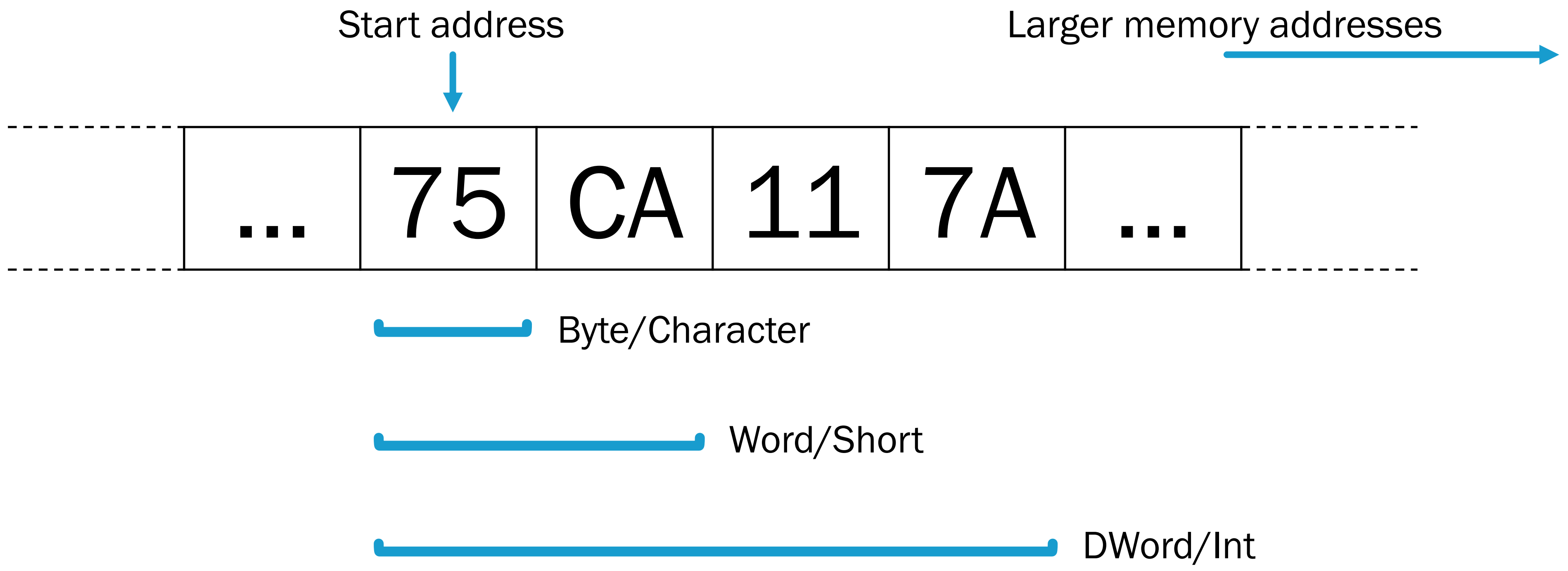
Little endian: 0x 75 CA 11 7A

4 3 2 1

YOU MANIACS! WHY?!

Little endian makes choosing correct data size easier

Also easier to determine even vs. odd



HOW DO YOU KNOW?

Uses little endian

- Intel-family processors
- Most file systems on Intel hosts
- PCAP files created on Intel

Uses big endian

- Sparc, MIPS, … CPUs
- XFS file systems (always)
- Packets on the wire

NOT SCARY PRACTICE #4

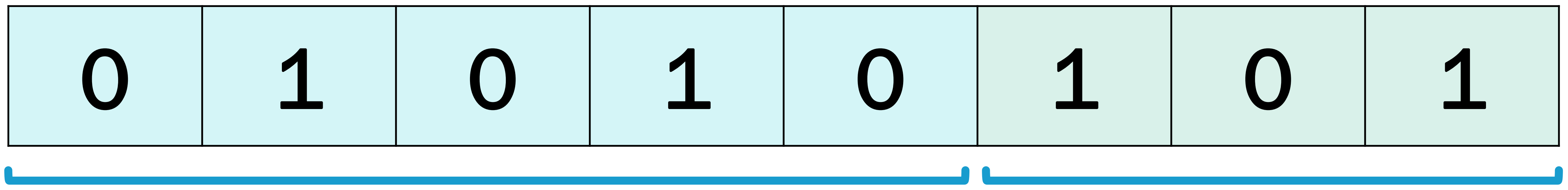
WHICH END IS UP?

PROGRAMMERS ARE ~~WEIRD~~ THRIFTY

Programmers sometimes pack multiple fields into the same byte(s)

This comes from a time when memory/disk were much more limited

Unix Syslog packs a *facility number* and a *priority value* into a single byte



Facility in upper 5 bits
(10 = "authpriv")

Priority in lower 3 bits
(5 = "notice")

PLAYING THE FIELDS

Suppose I want to use the facility or priority value?

We need to get each field into a byte by itself

This requires us to use *masking* and *shifting* – binary arithmetic!

THIS IS NOT AS TERRIBLE AS IT SOUNDS!

MASK ON/MASK OFF

&	0	1
0	0	0
1	0	1

Bitwise AND ("&") is like multiplication for single bits

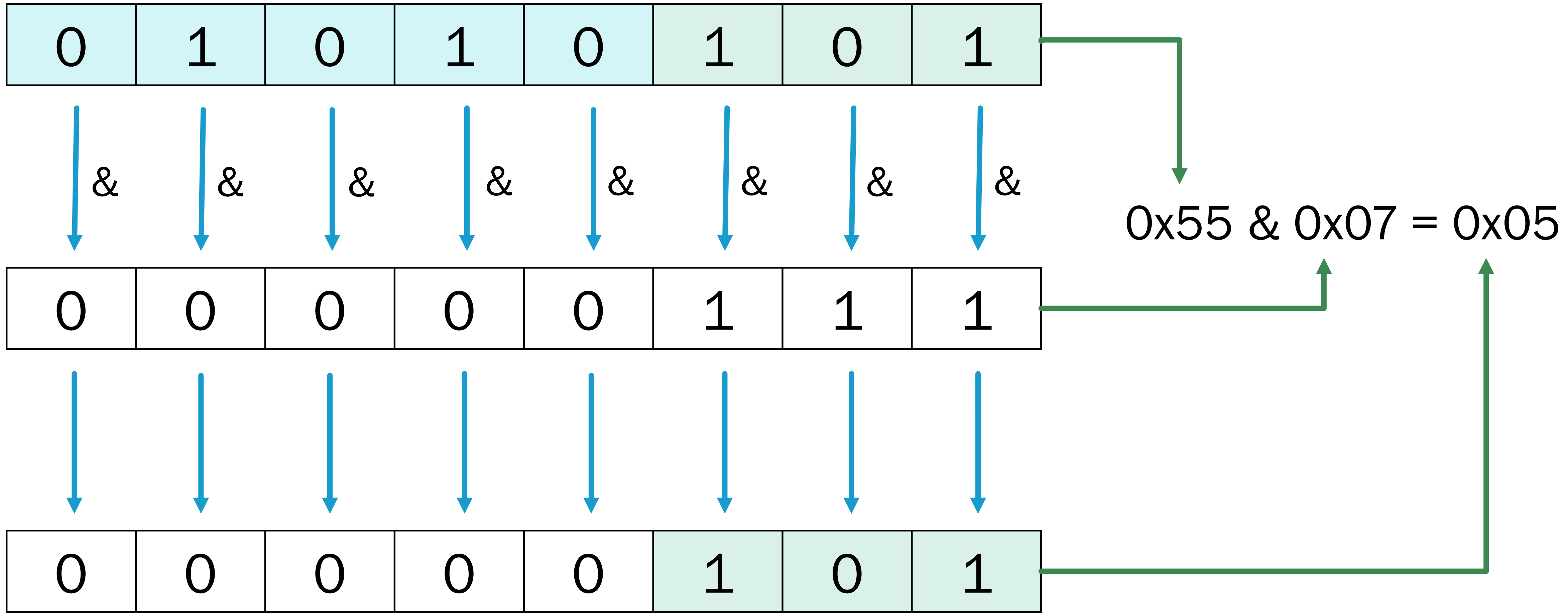
Unless both bits are one, the result is zero

A *mask* is multiple bits that keep/suppress bits you want/don't want

The "all-ones" mask retains bits you want

Use an "all-zeroes" mask to eliminate bits you don't want

WHAT IS BEHIND THE MASK?

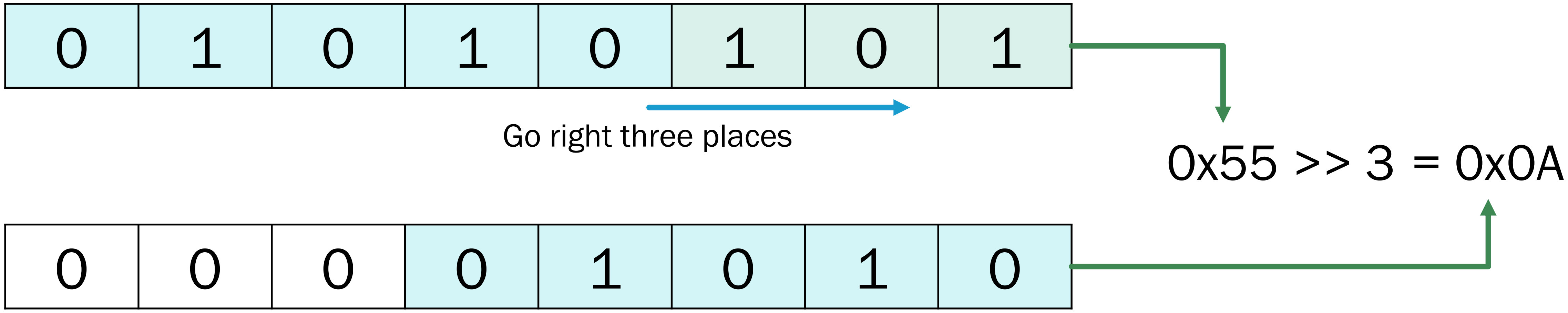


A MOVING STORY

Now I want to use the facility value (upper five bits)!

We need to move those five bits three places to the right

Programmers say, "*Shift right three*" (">> 3")



GOING BACK THE OTHER WAY

Create the original packed byte with facility 0x0A and priority 0x05:

0	0	0	0	1	0	1	0	0x0A
0	1	0	1	0	0	0	0	0x0A << 3
0	1	0	1	0	1	0	1	0x0A << 3 + 0x05

NOT SCARY PRACTICE #5

SHIFTY PEOPLE WITH MASKS

THANK YOU!

Thanks for participating!

Any final questions?

hrpomeranz@gmail.com

@hal_pomeranz@infosec.exchange