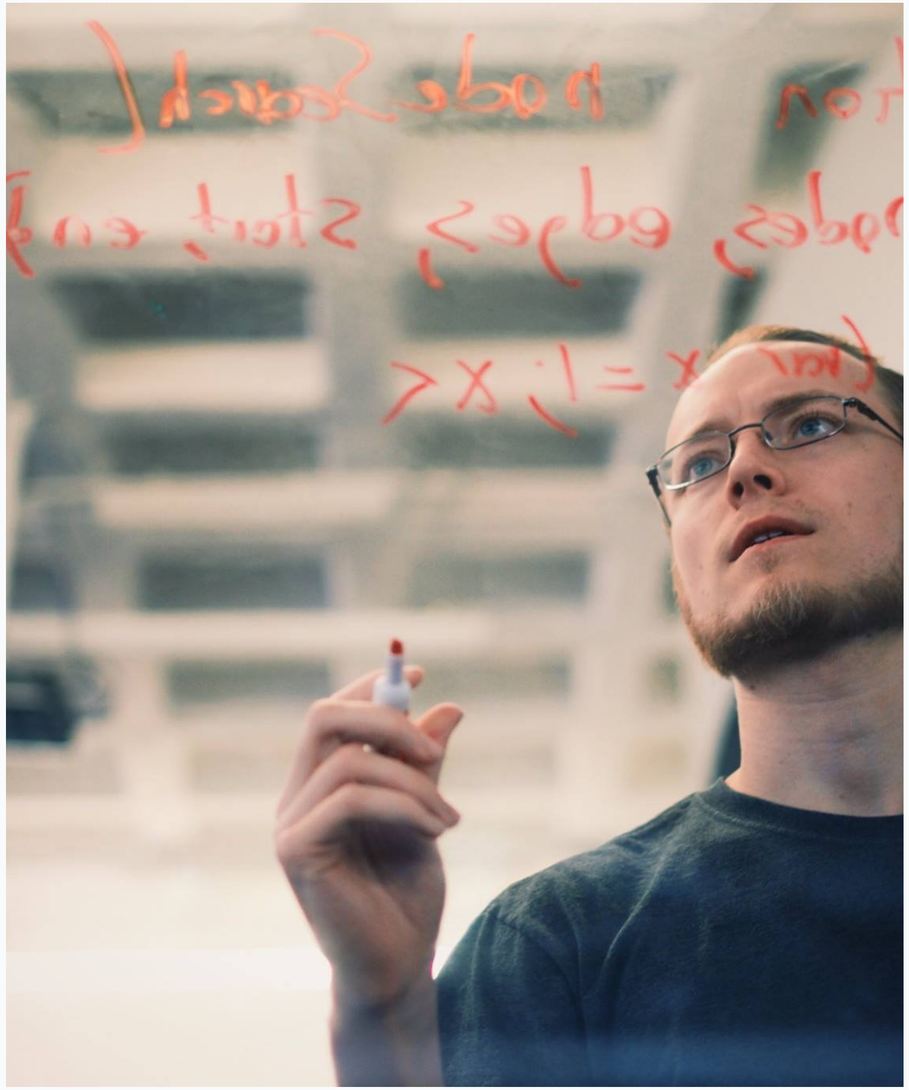
Data Types & Structures

Physical Media, Abstractions, Programs & Performance

"[The files are] in the computer... it's so simple."
MakeAGif.com

(a few) motivations

- Understand what on earth you're doing
  - Demystify foundations
  - Gain insight & perspective
- Recognize problems
- Recognize solutions
- Write fast code
- Interview well



Coming Up

I. Information Theory & Hardware

II. Abstraction & Encoding

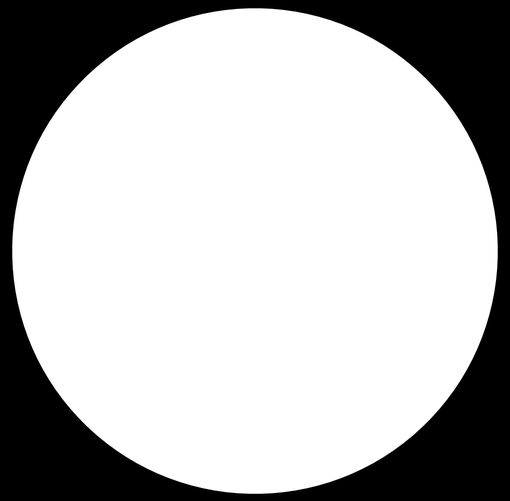
III. Software & APIs

IV. Abstract Data Types & Data Structures

- IV.a. Queues
- IV.b. Linked Lists
- IV.c. Hash Tables
- IV.d. Trees

V. Extras

Part I:
Information Theory & Hardware



T

F

T

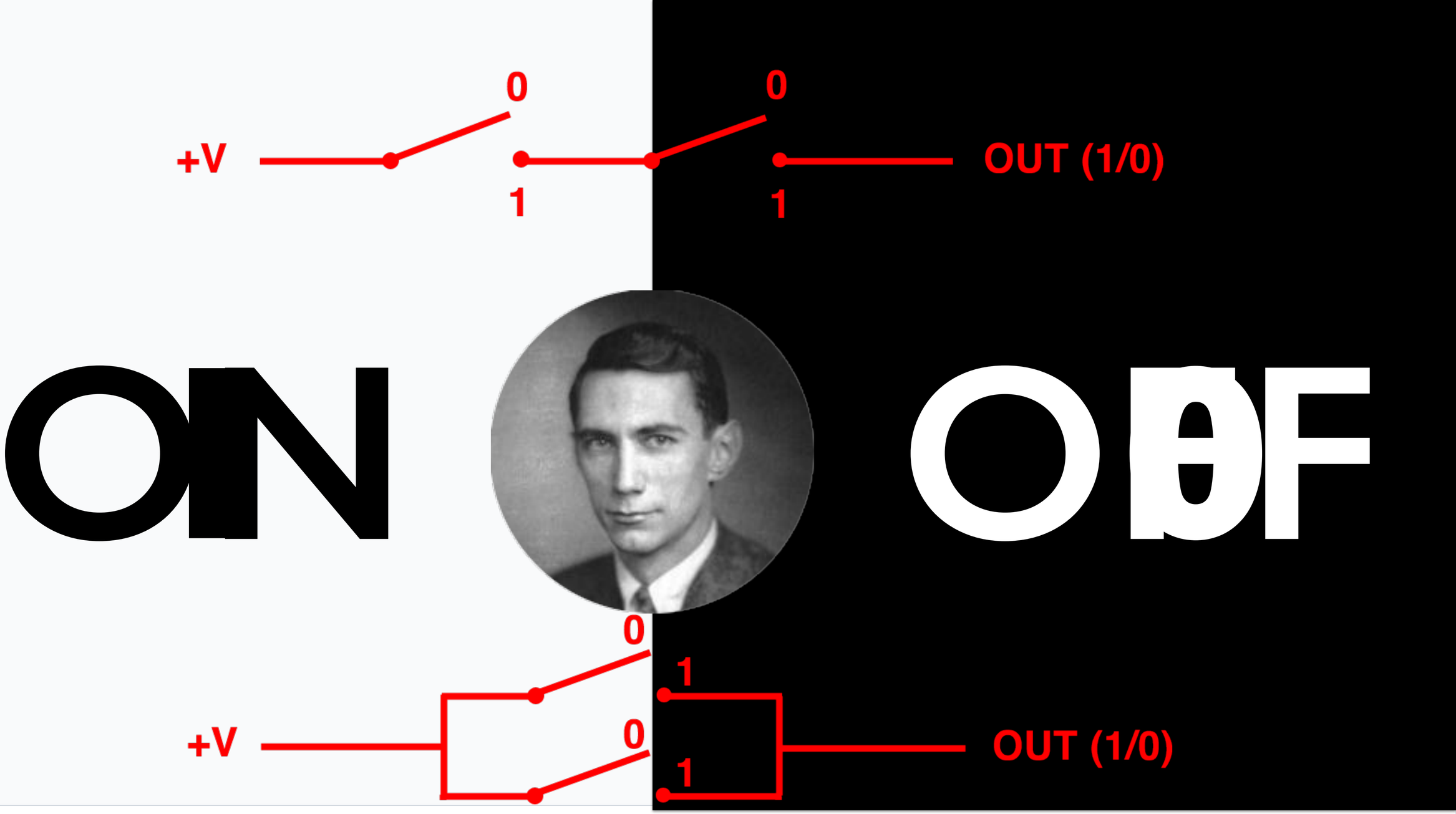
¬0

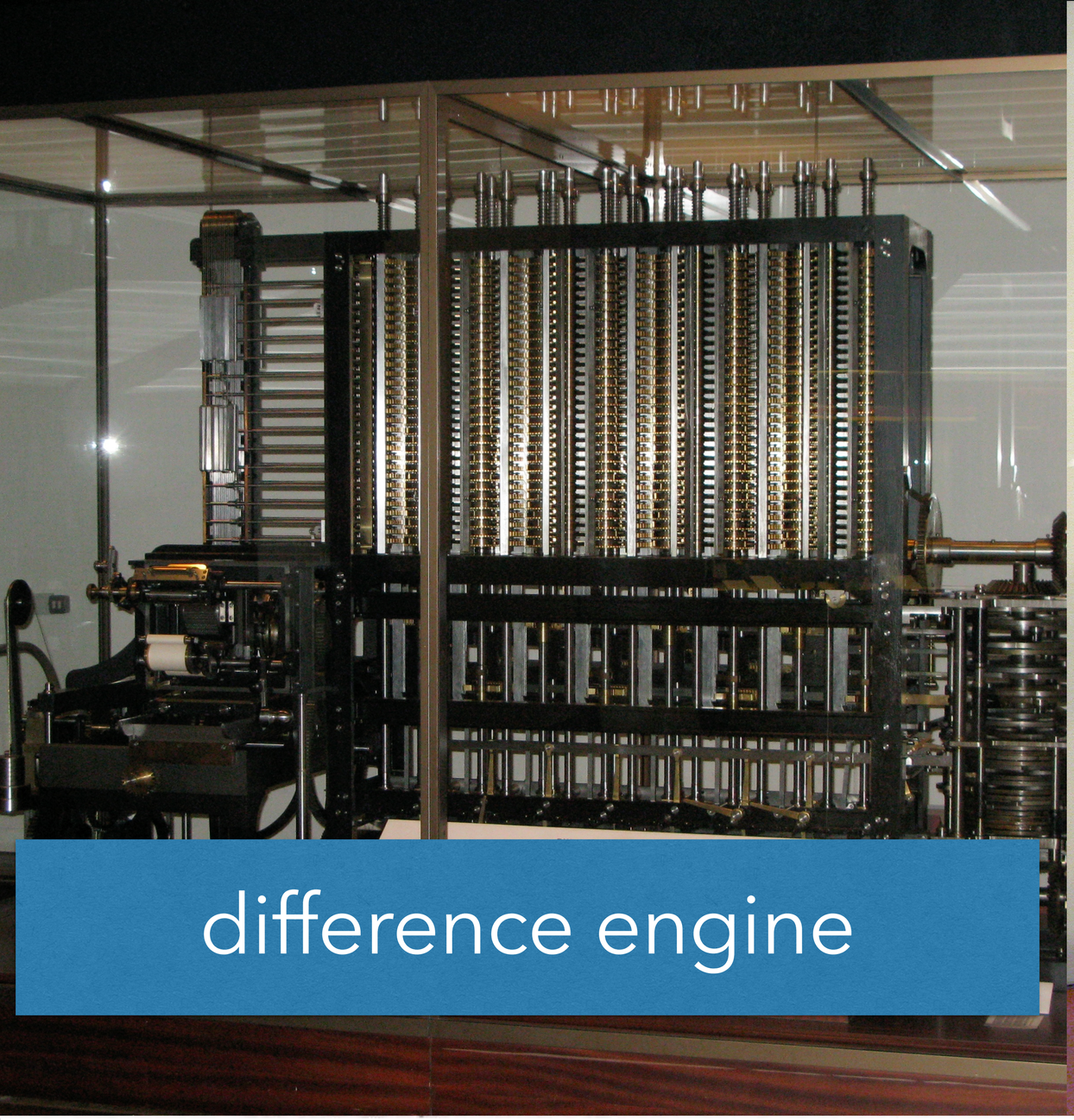
∧

∨

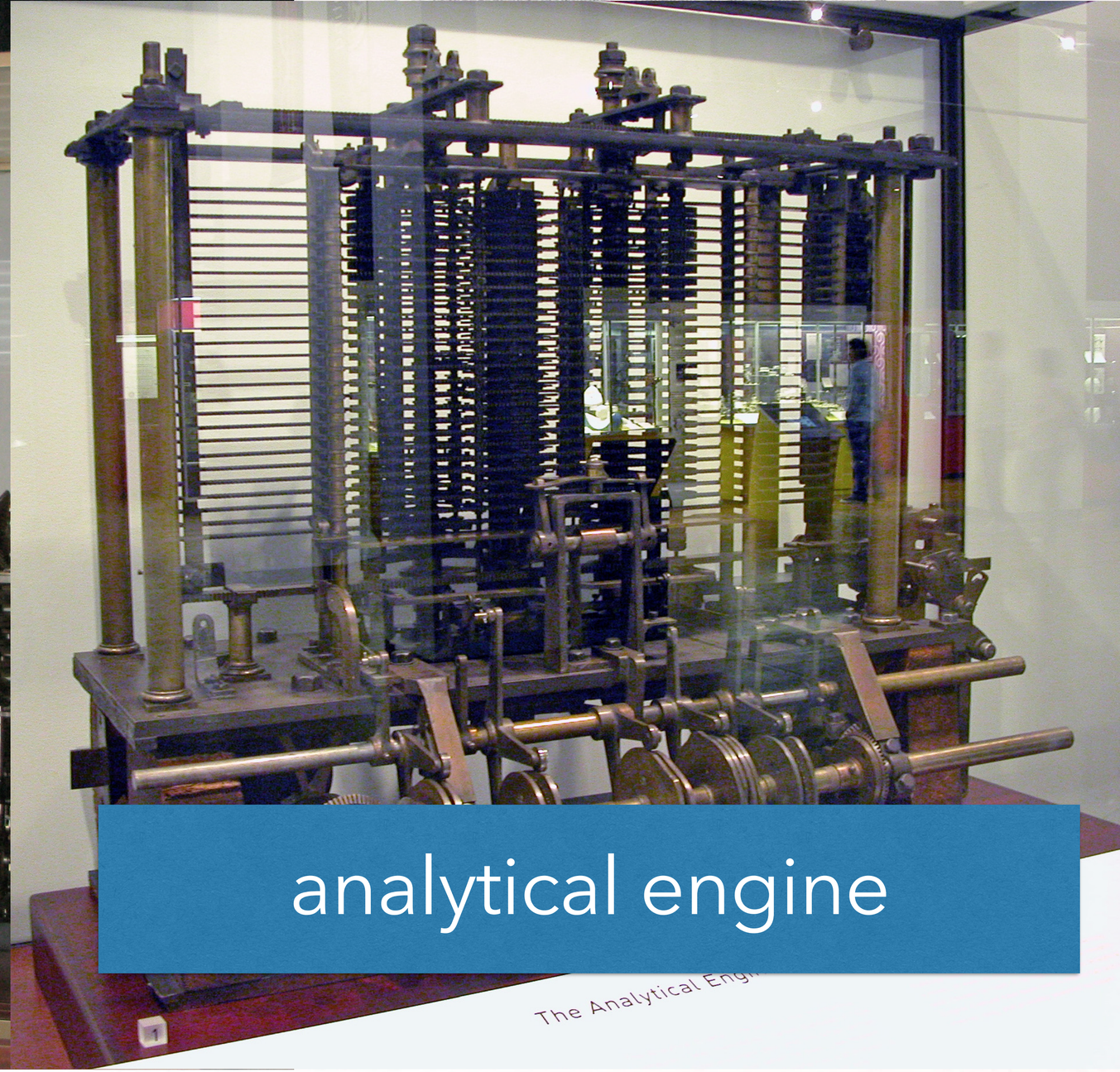
F

¬1

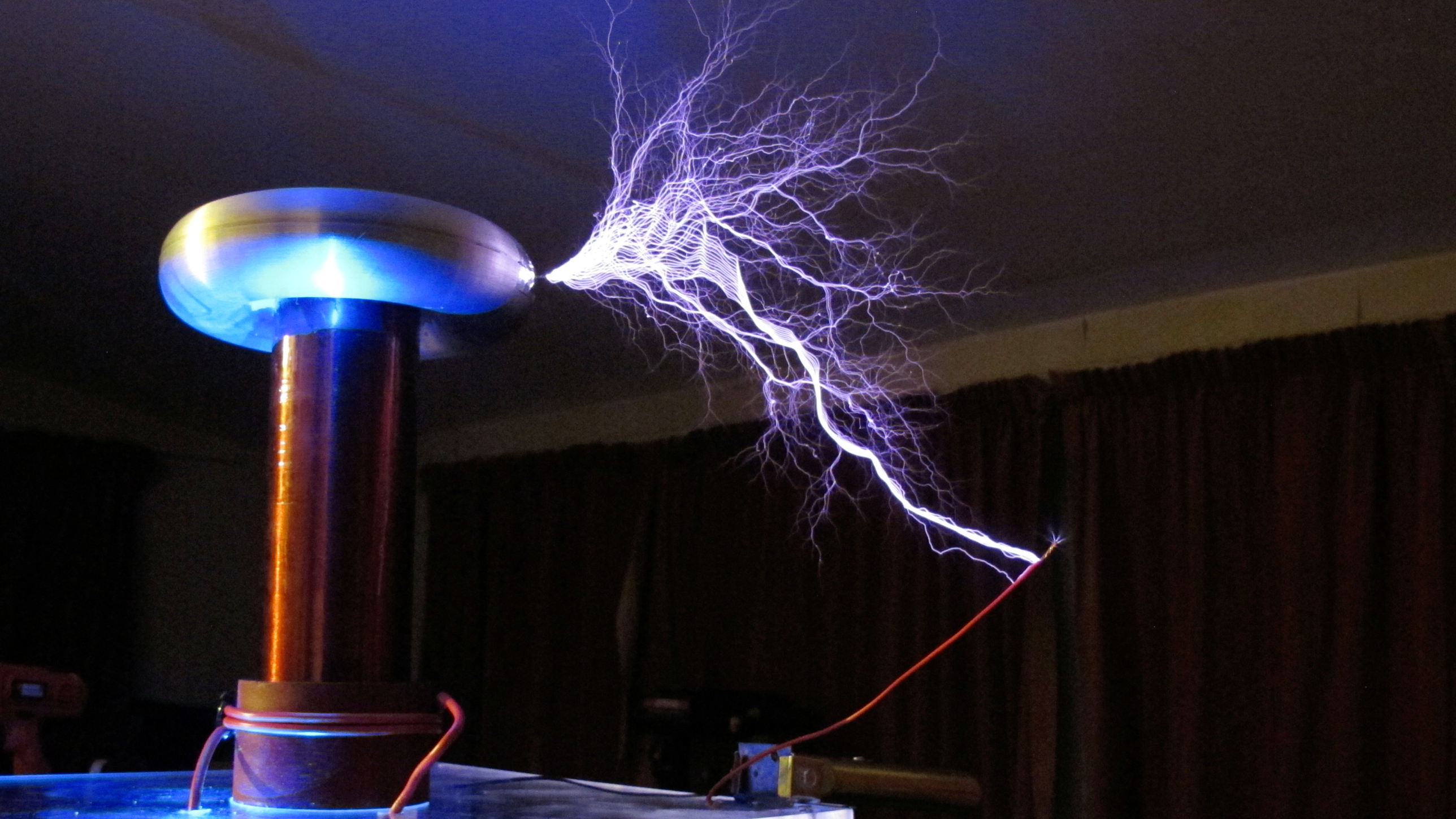

0
+V
1
0
1
OUT (1/0)
ON
OFF
0
1
+V
0
1
OUT (1/0)

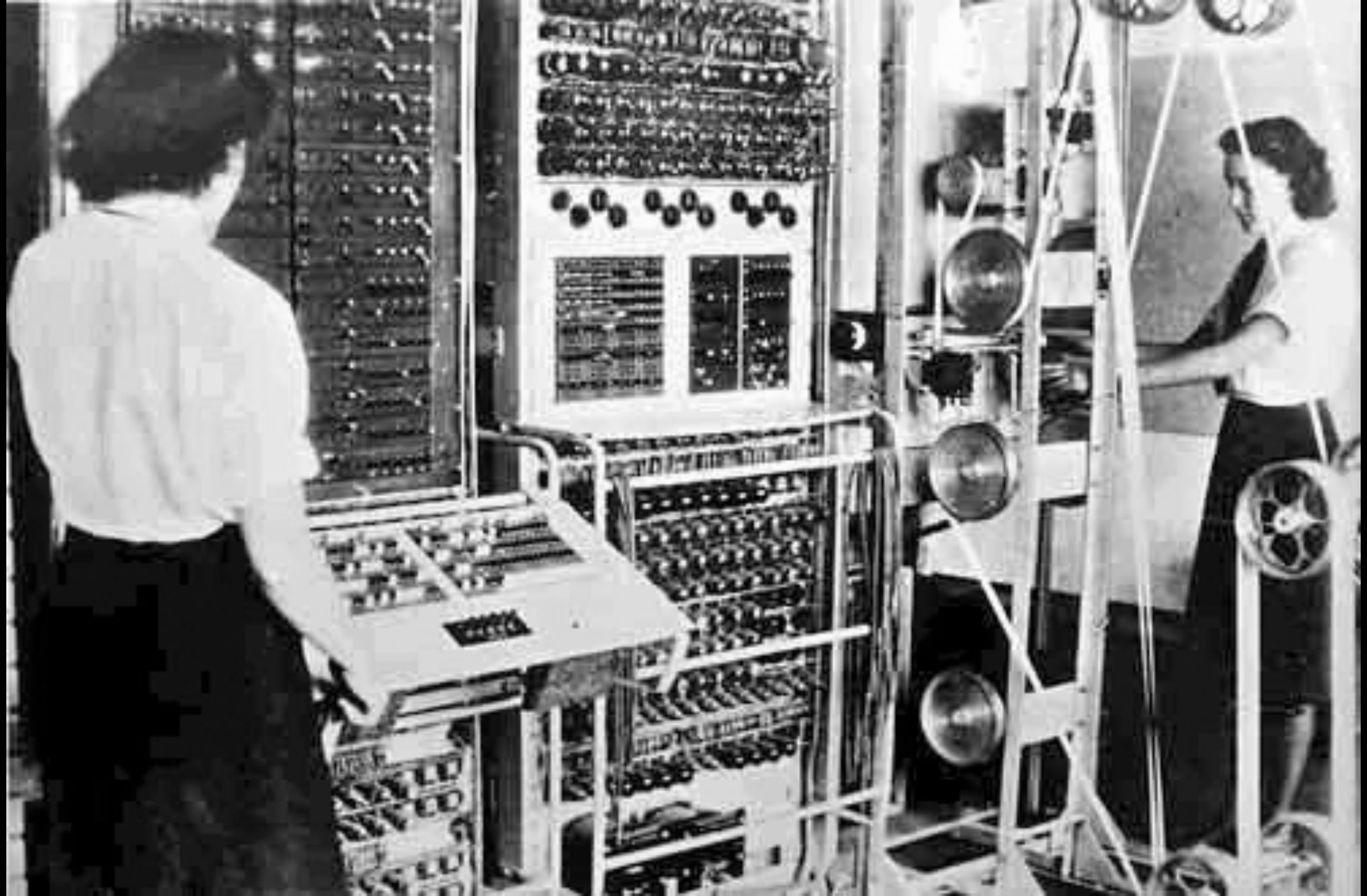


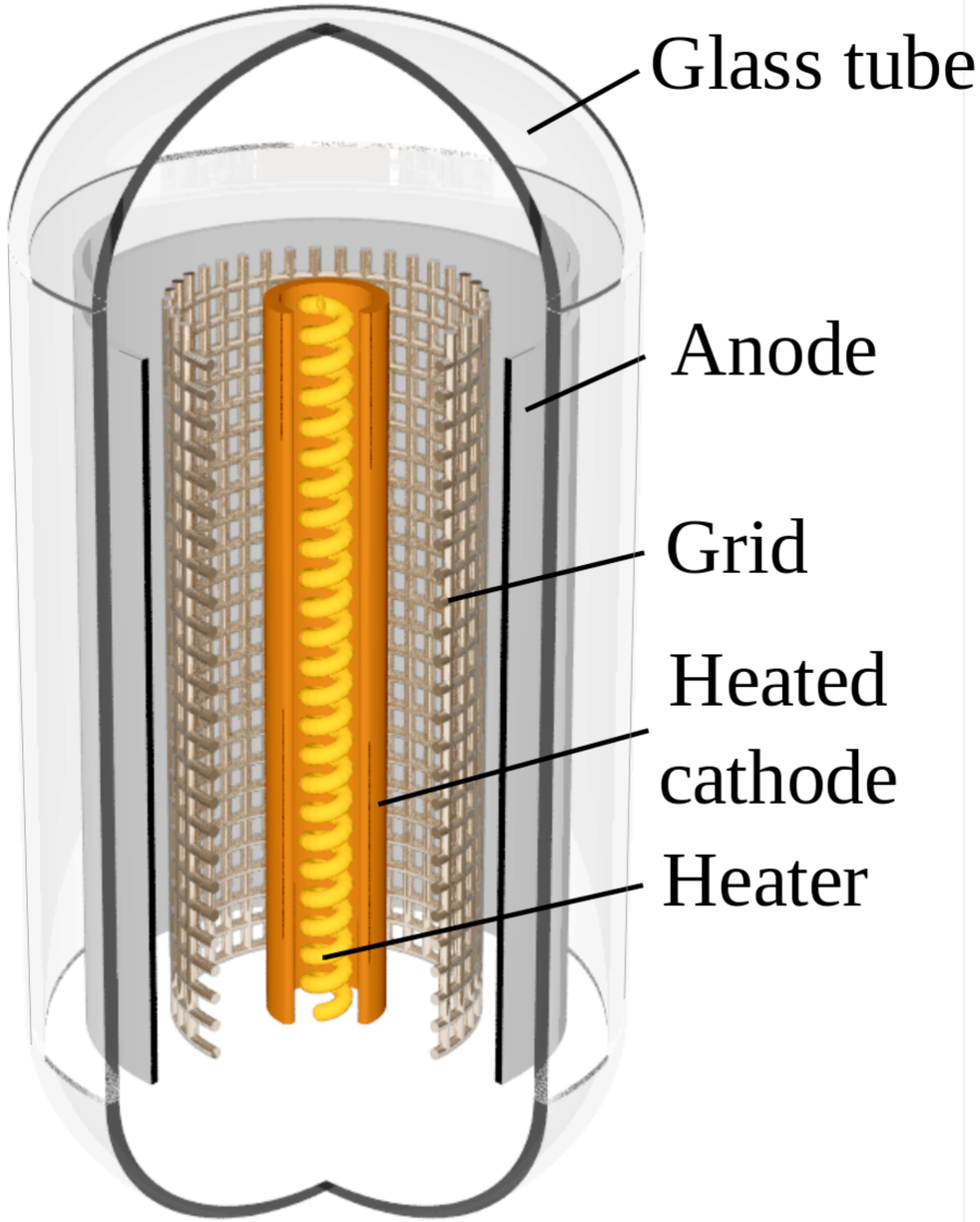
difference engine

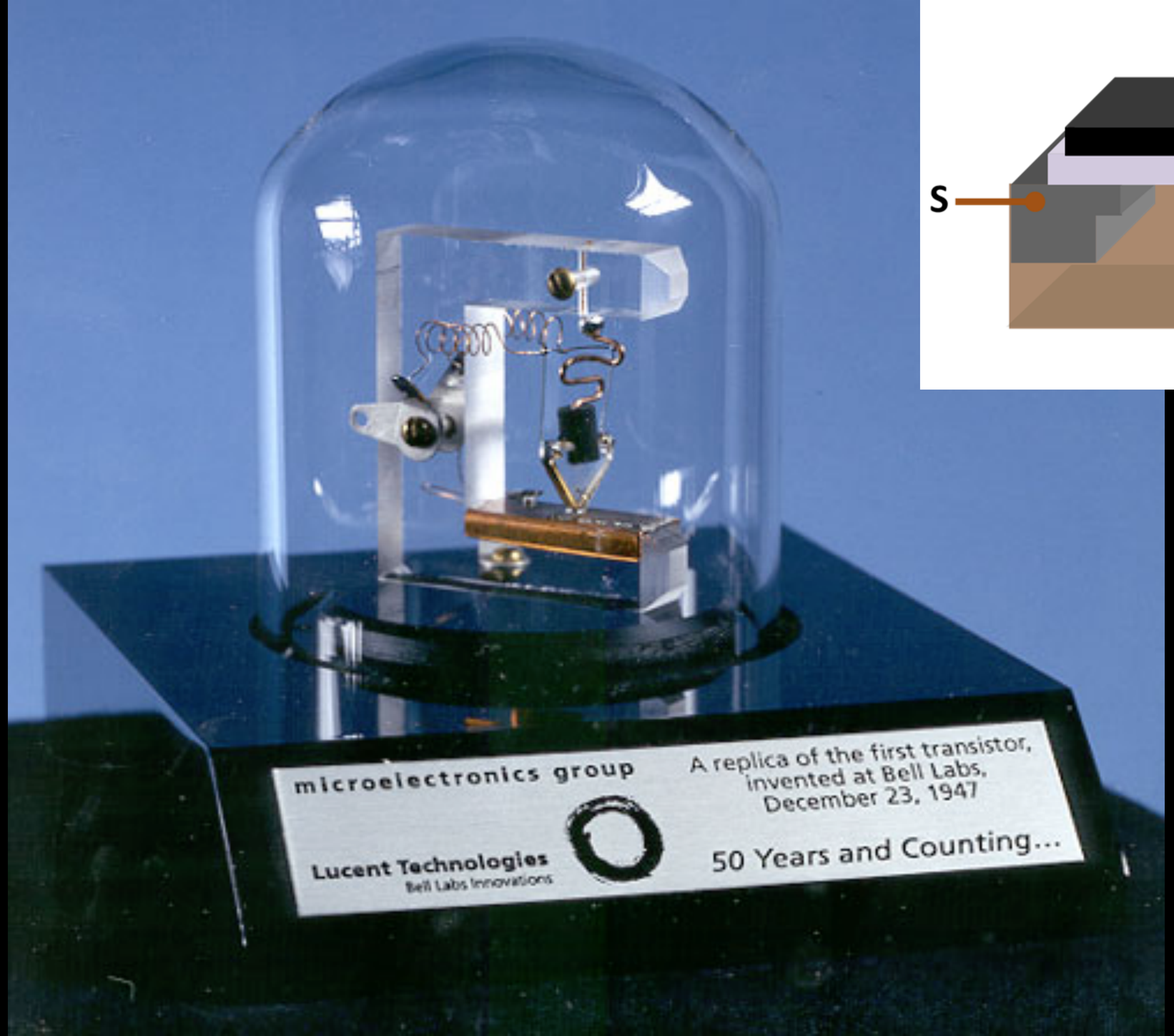


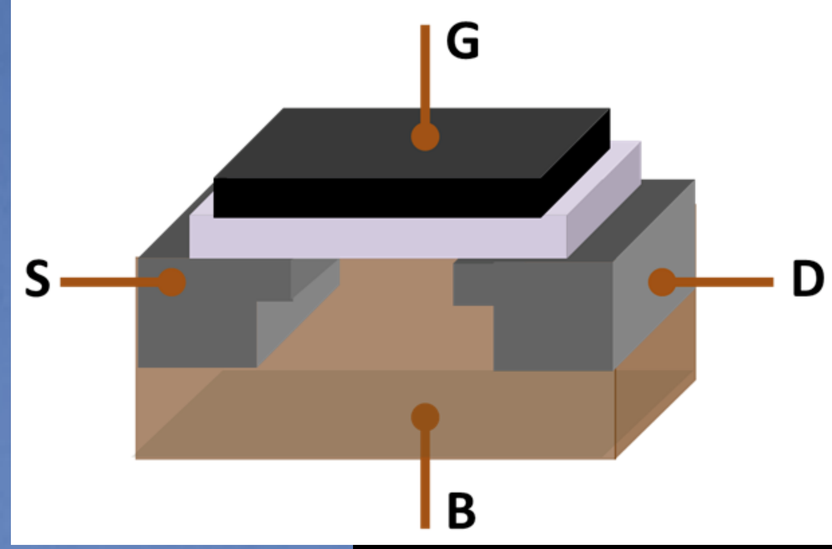
analytical engine

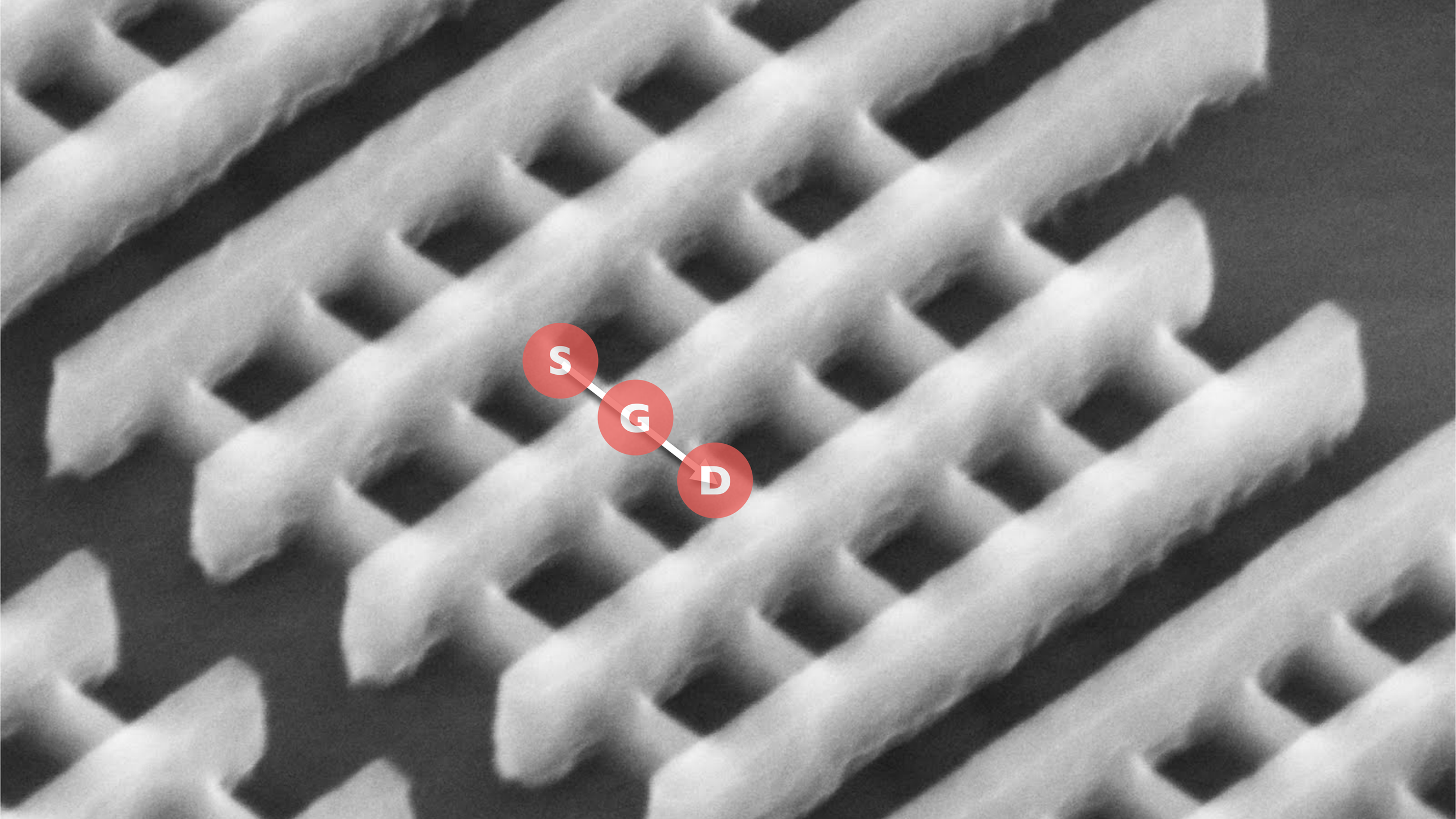




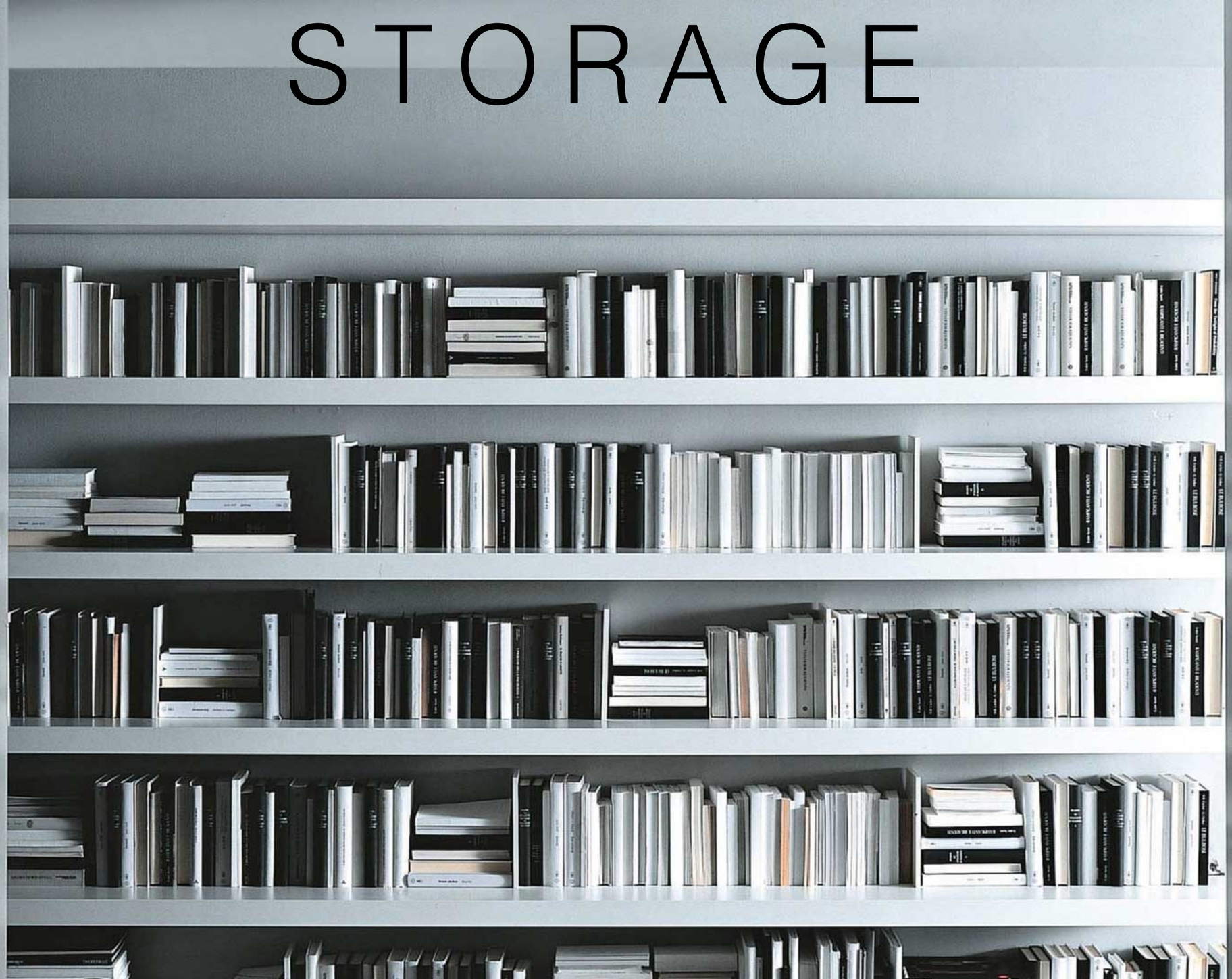

Glass tube
Anode
Grid
Heated cathode
Heater


microelectronics group
A replica of the first transistor, invented at Bell Labs, December 23, 1947
Lucent Technologies
Bell Labs Innovations
50 Years and Counting...

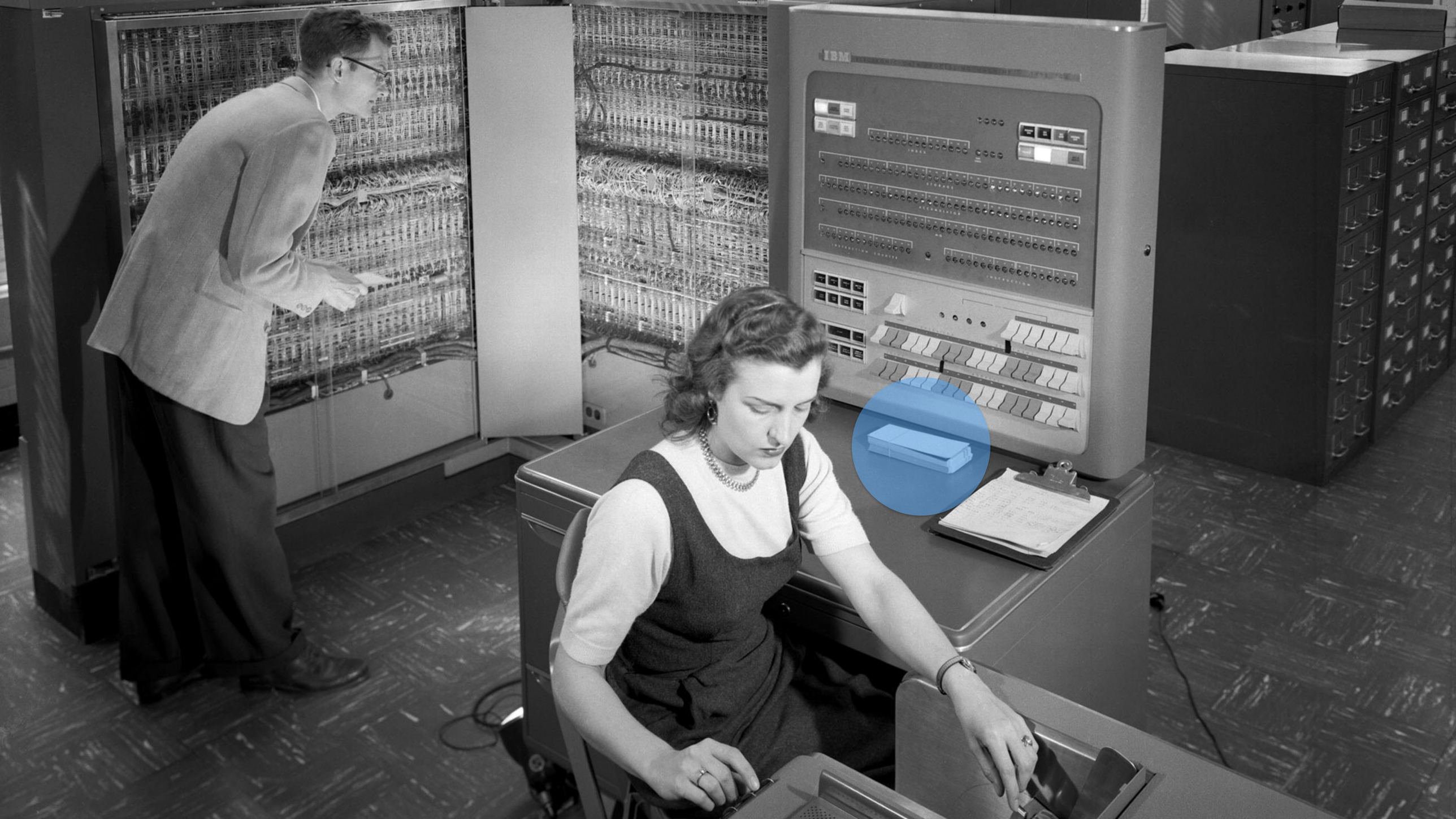

G
S
D
B

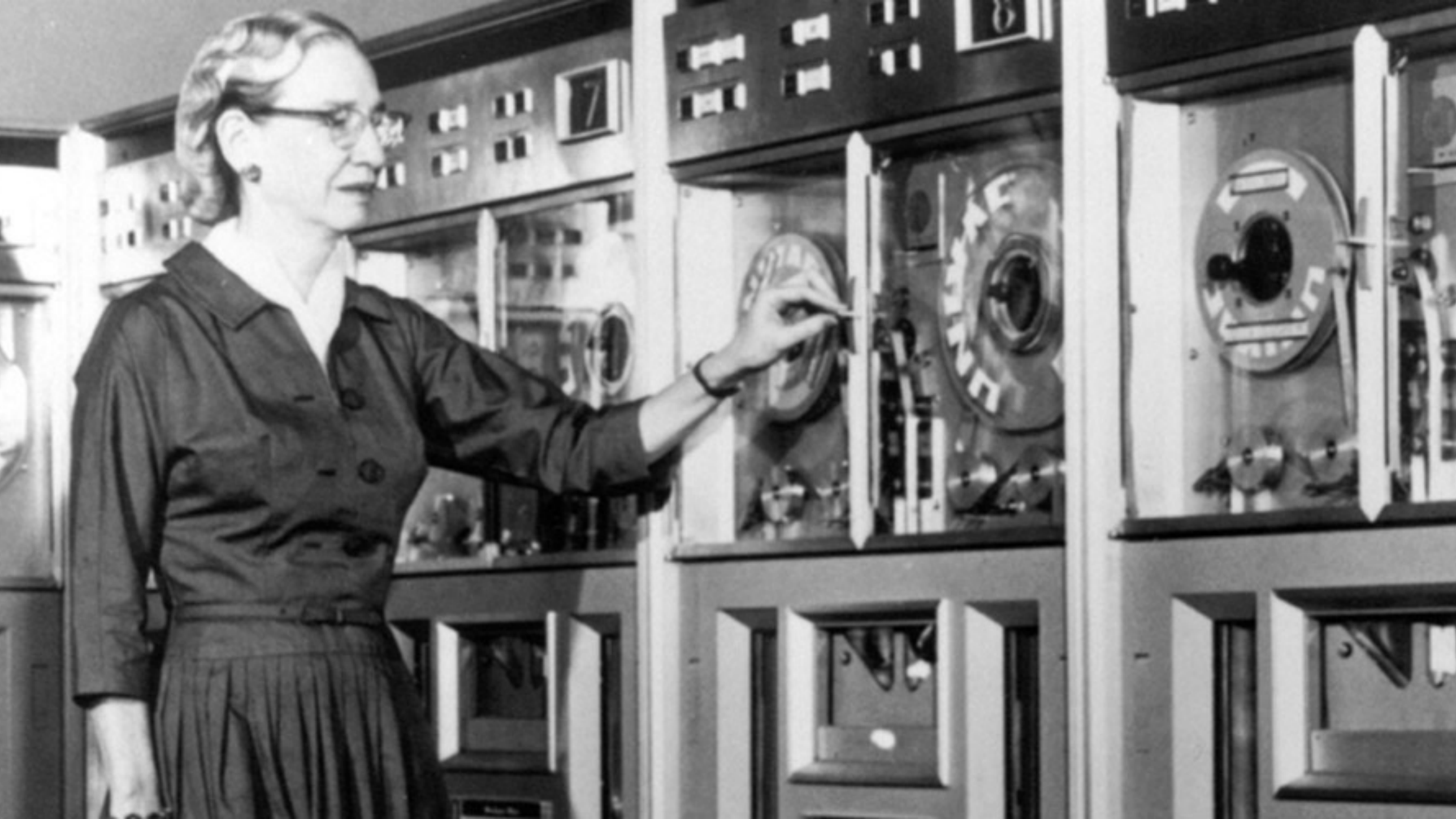

S
G
D

STORAGE

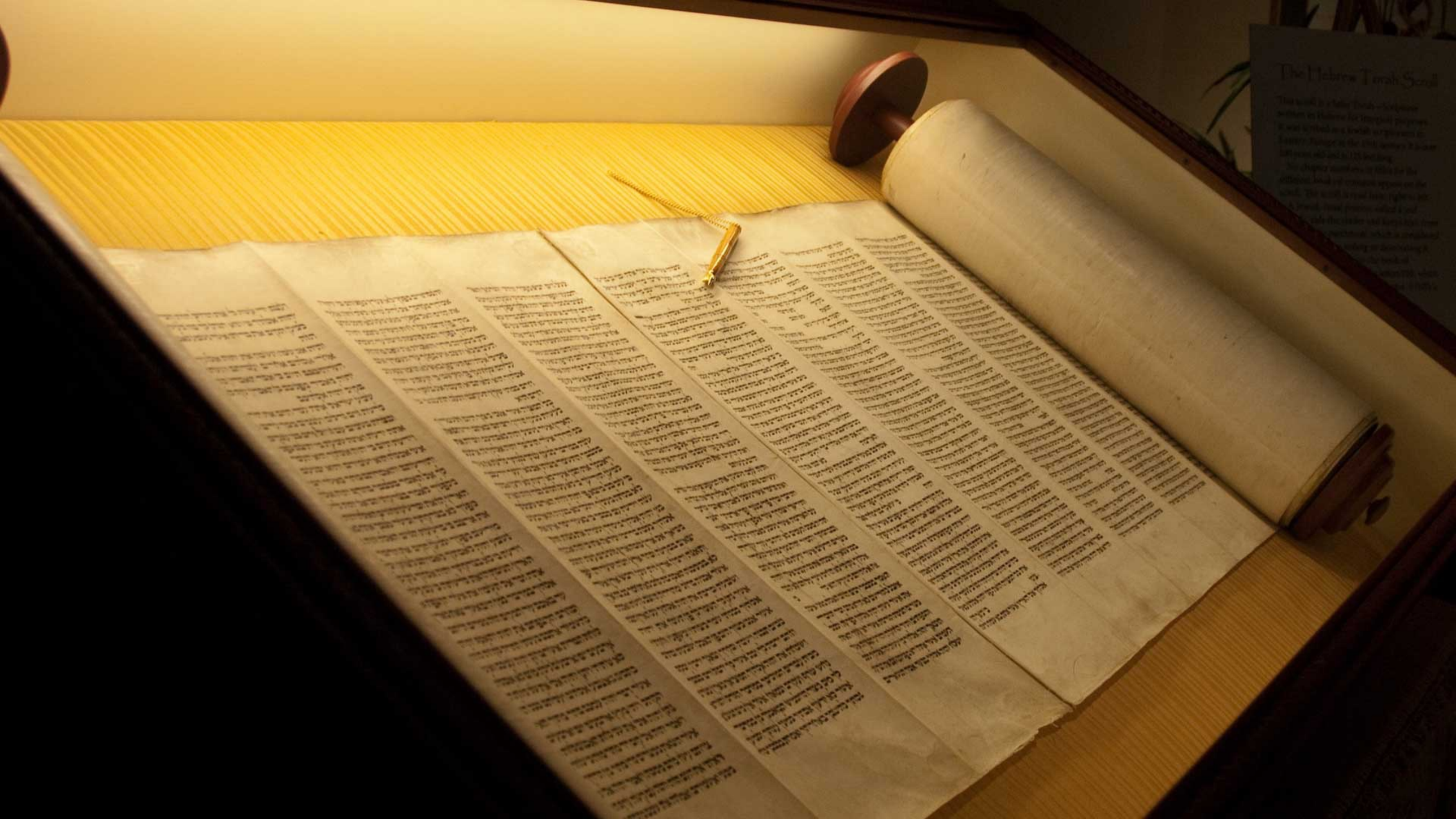


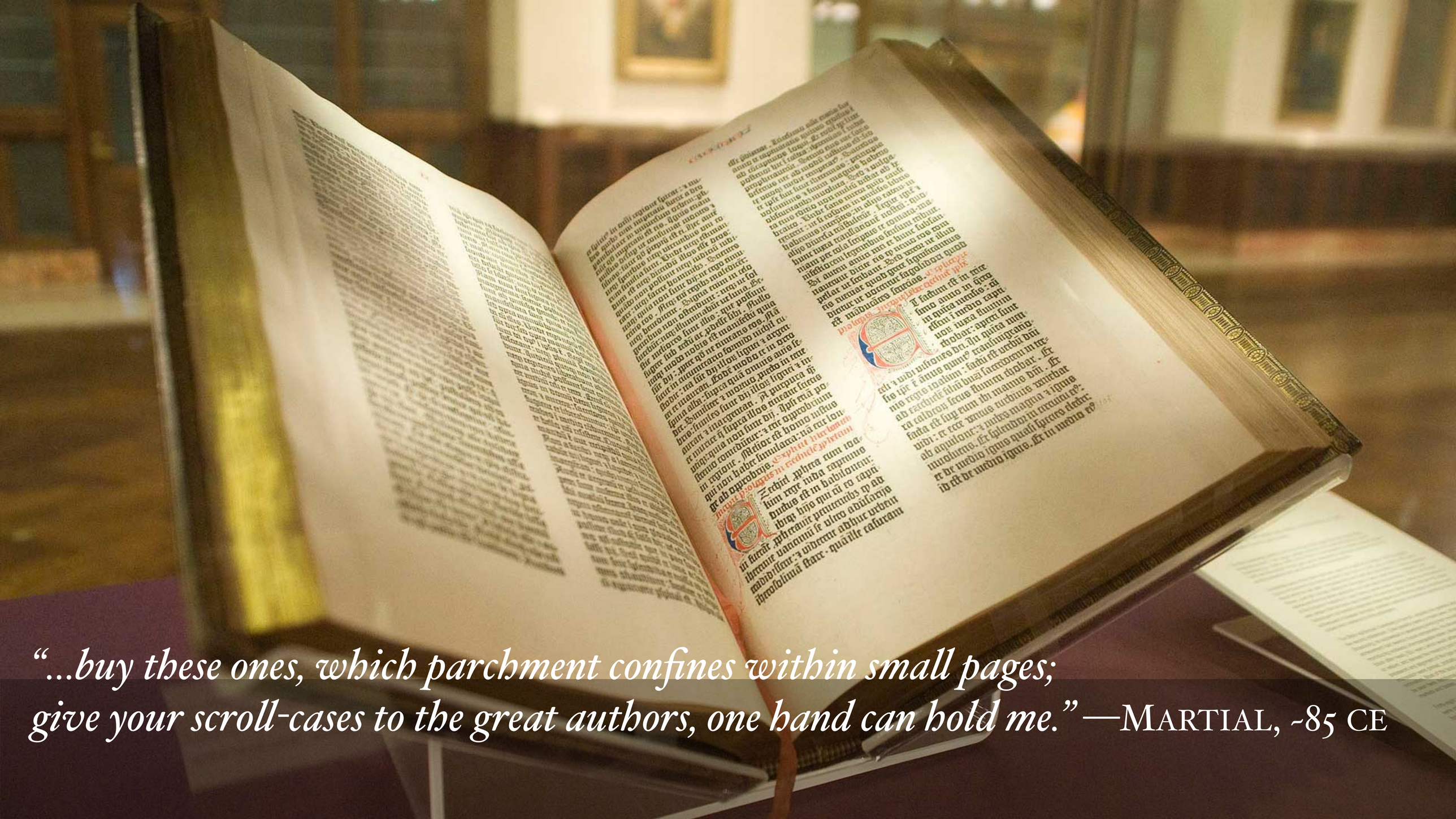
1	1	3	0	2	4	10	On	S	A	C	E	a	c	e	g			EB	SB	Ch	Sy	U	Sh	Hk	Br	Rm
2	2	4	1	3	E	15	Off	IS	B	D	F	b	d	f	h			SY	X	Fp	Cn	R	X	Al	Cg	Kg
3	0	0	0	0	W	20		0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0
A	1	1	1	1	0	25	A	1	1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1
B	2	2	2	2	5	30	B	2	2		2	2	2	2	2	2	2	2	2		2	2	2	2	2	2
C	3	3	3	3	0	3	C	3	3	3		3	3	3	3	3	3	3	3	3		3	3	3	3	3
D	4	4	4	4	1	4	D	4	4	4	4		4	4	4	4	4	4	4	4	4		4	4	4	4
E	5	5	5	5	2	C	E	5	5	5	5	5		5	5	5	5	5	5	5	5	5		5	5	5
F	6	6	6	6	A	D	F	6	6	6	6	6	6		6	6	6	6	6	6	6	6	6		6	6
G	7	7	7	7	B	E	G	7	7	7	7	7	7	7		7	7	7	7	7	7	7	7	7		7
H	8	8	8	8	a	F	H	8	8	8	8	8	8	8	8		8	8	8	8	8	8	8	8	8	
I	9	9	9	9	b	c	I	9	9	9	9	9	9	9	9	9		9	9	9	9	9	9	9	9	9


IBM



STORAGETEK

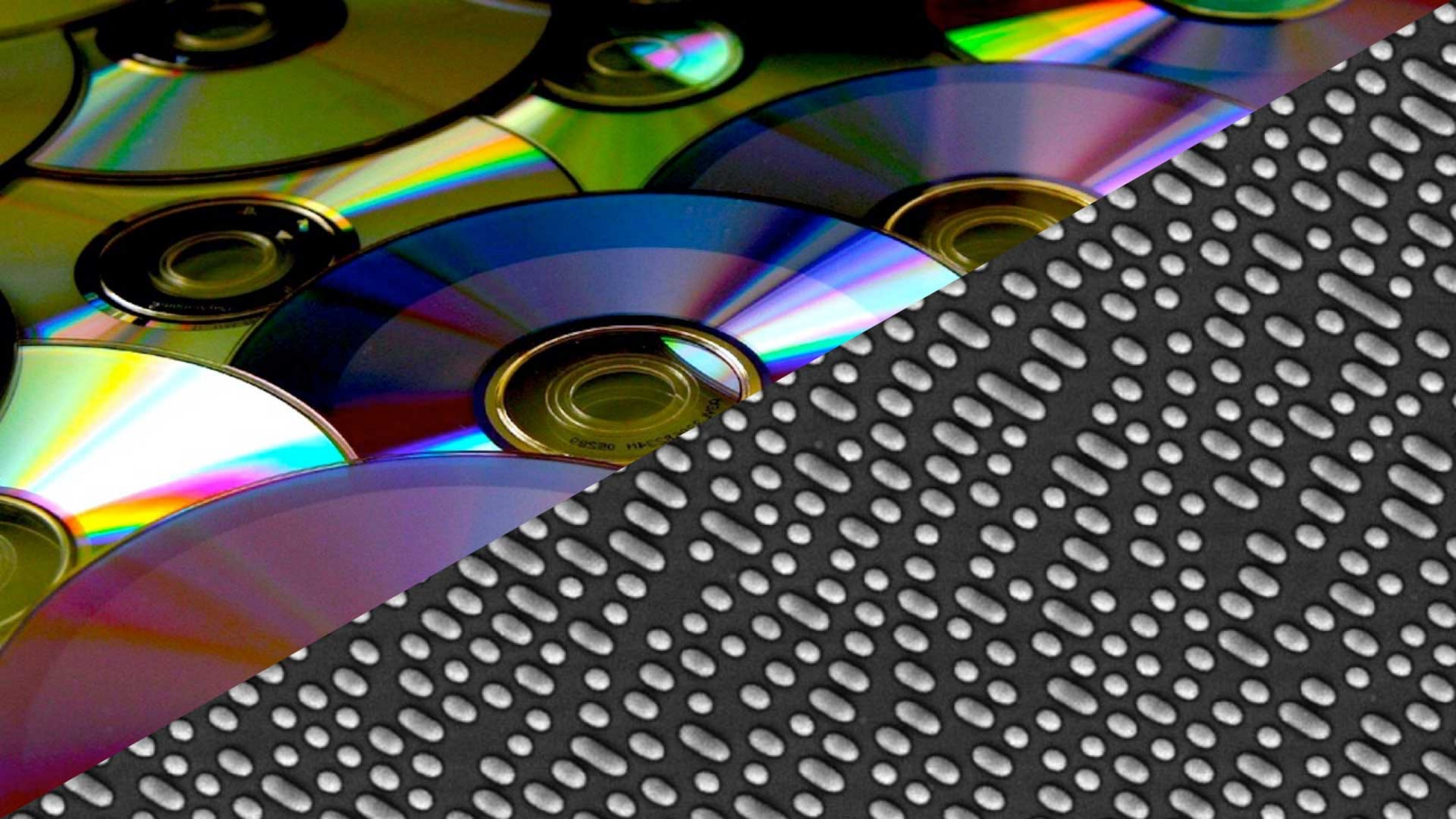




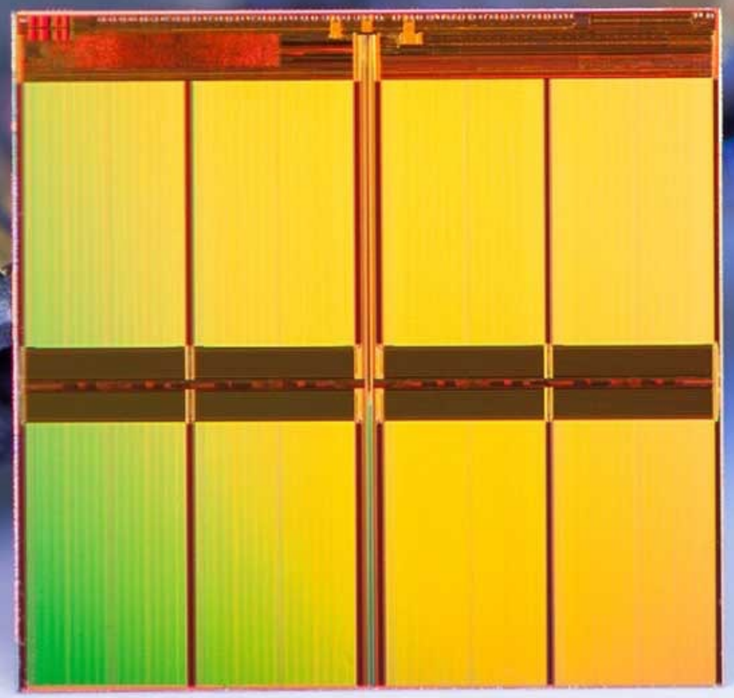
"...buy these ones, which parchment confines within small pages;
give your scroll-cases to the great authors, one hand can hold me." —Martial, -85 CE

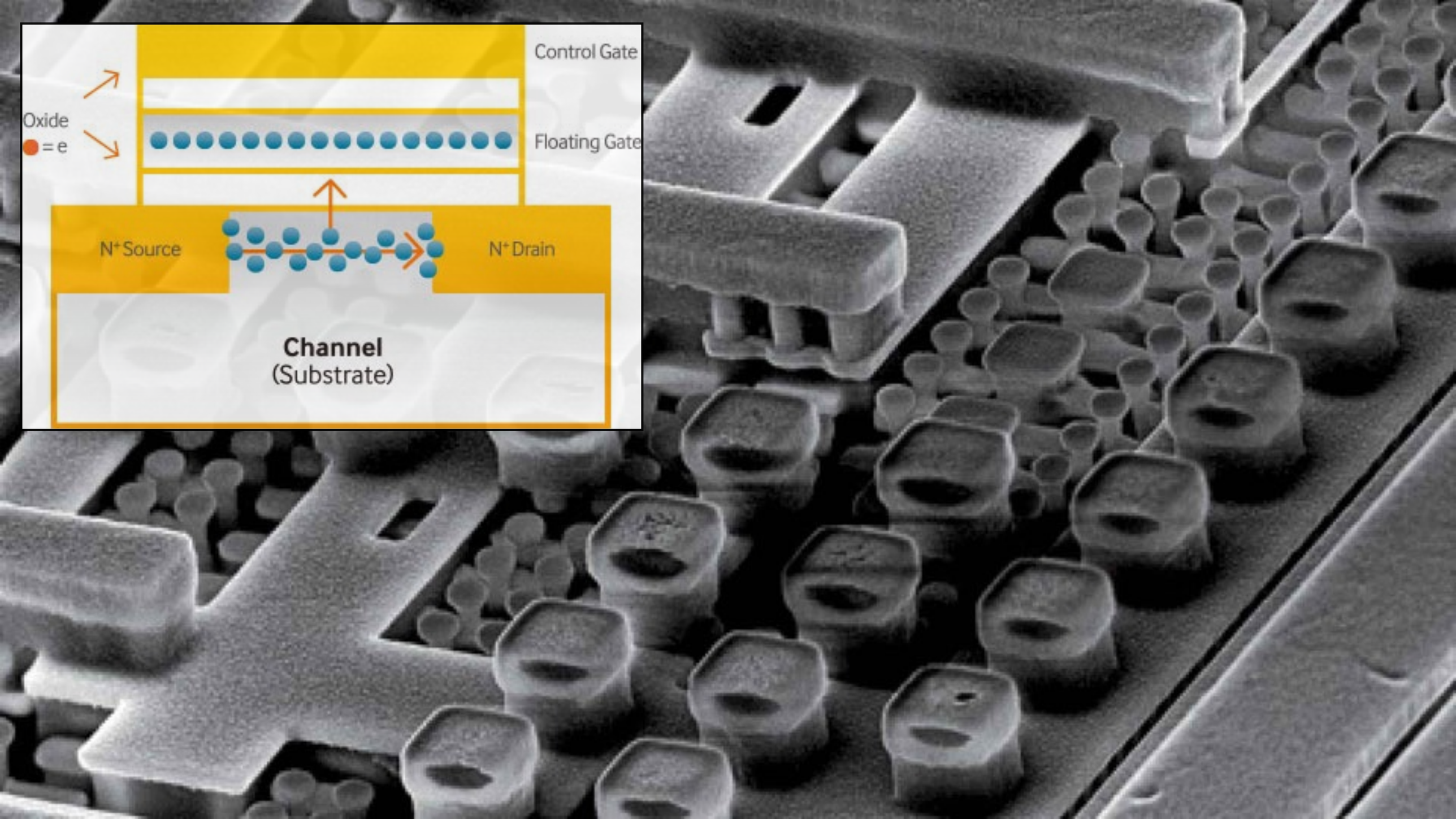
VIDEO LINK

SS/SD
ЗОМ
ДО В БОЛГАРИИ
boedem
FLOPPY DISK
5,25" HD,DS
BASF
WRITE ENABLE
WRITE PROTECT
INDEX



SAMSUNG
Solid State Drive




Control Gate
Oxide
= e
Floating Gate
N+ Source
N+ Drain
Channel
(Substrate)

LONG TERM STORAGE
MEMORY

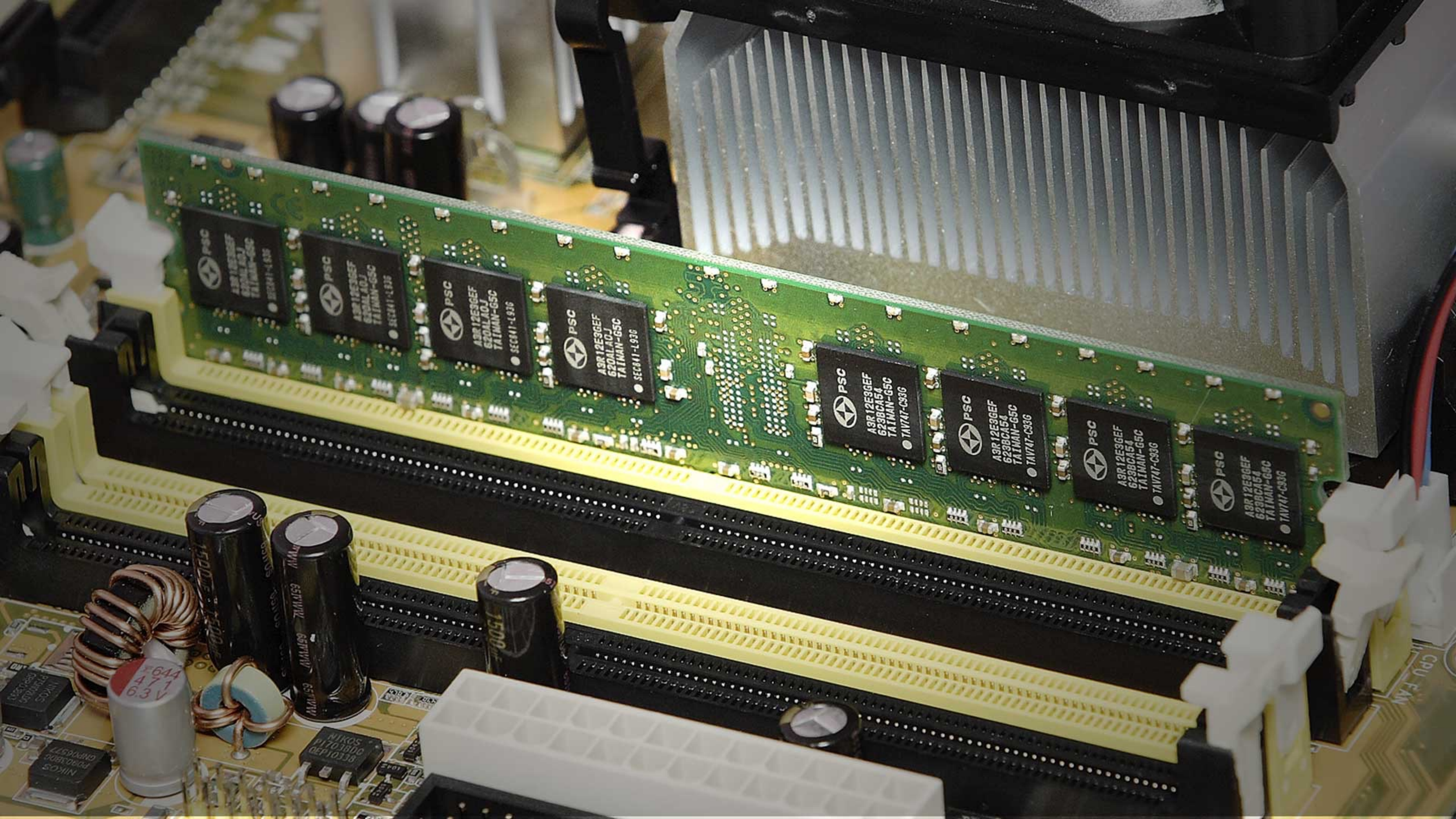
Medium	Typical Size (bytes)	Approx. Access Time
CPU Register	64 bits = 8	one cycle < 1 ns
L1 Cache	64 000	4–5 cycles ≈ 2 ns
L2 Cache	256 000	~12 cycles ≈ 5 ns
RAM	8 000 000 000	70 ns
Solid State Drive	256 000 000 000	25 000 ns
Hard Disk Drive	1 000 000 000 000	12 000 000 ns
Optical Drive	650 000 000	150 000 000 ns
Tape Drive	8 000 000 000 000	60 000 000 000 ns
Network	a big truck?	1000 – 1e8 – inf∞ ns

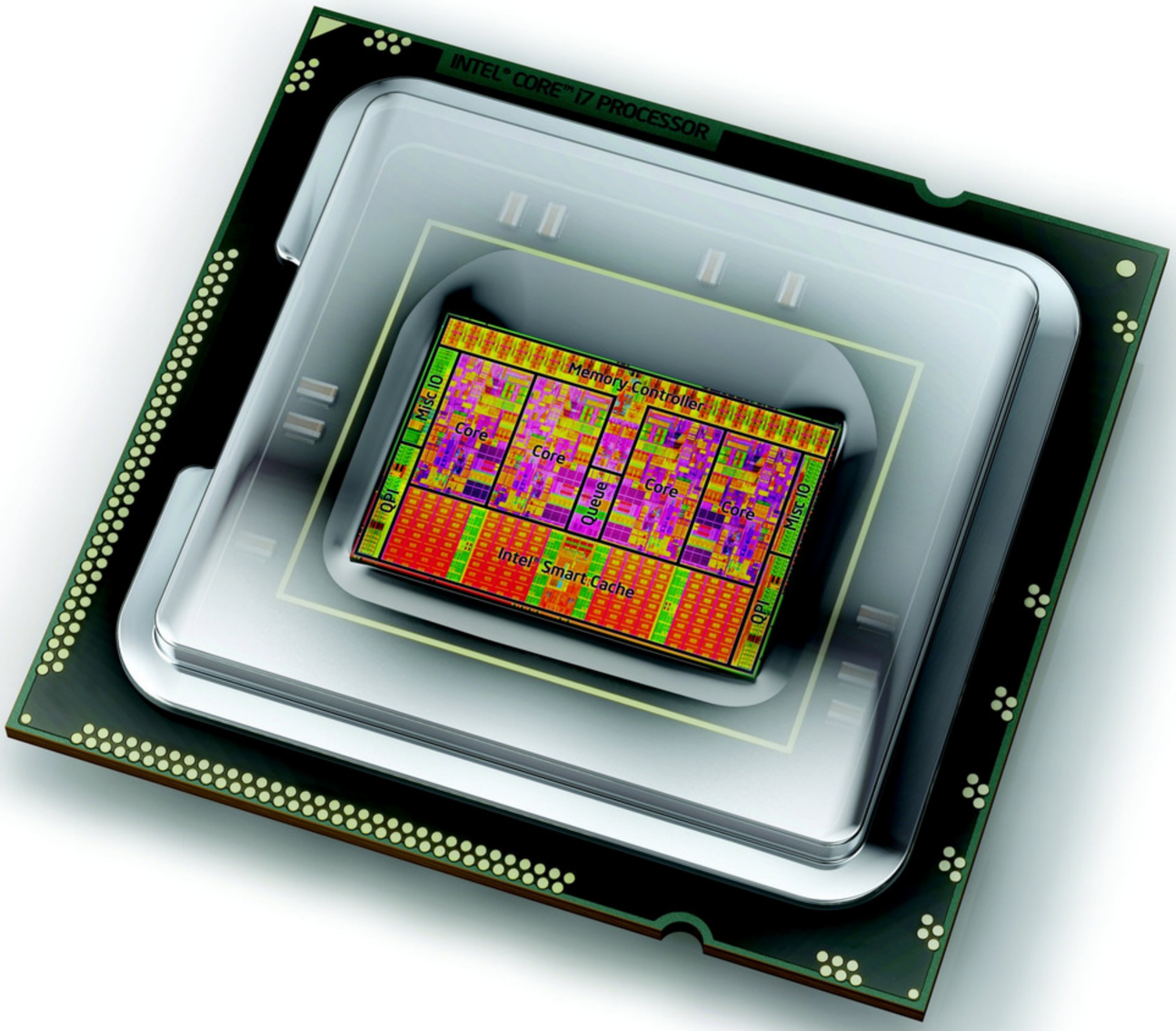
"Memory"
primary
storage
("non-blocking")
volatile

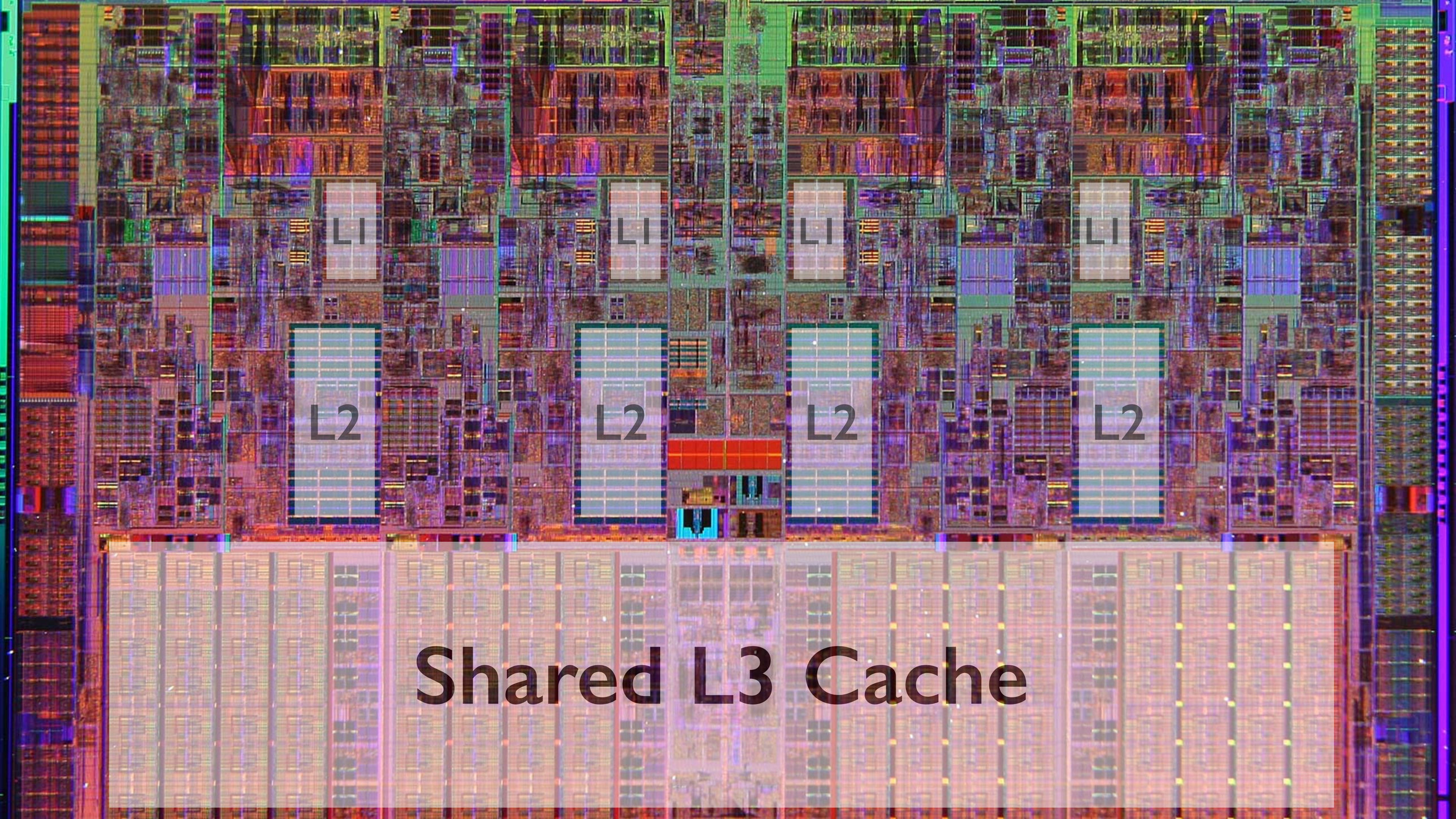
"Storage"
secondary
storage
("blocking")
non-volatile

If a clock cycle were 1 second...

Medium	
CPU Register	1 s (on the paper)
L1 Cache	2s (on the desk)
L2 Cache	5s (in a pile)
RAM	1 minute (in a drawer)
Solid State Drive	7 hours (in the building)
Hard Disk Drive	138 days (library far away)
Optical Drive	5 years (library on Mars)
Tape Drive	2000 years
Network	20 minutes – 3 years – maybe never


PSC
A3R12E3GEF
620ALACJ
TAIWAN-G5C
SEC941-L936
A3R12E3GEF
623BCA54
TAIWAN-G5C
TAV747-C93G
NIKOS
PTA03800
DEF10338
644
471
6.3V


INTEL® CORE™ I7 PROCESSOR
Memory Controller
Misc IO
Core
Core
Core
Core
Queue
QPI
Intel® Smart Cache
Misc IO
QPI


L1
L1
L1
L1
L2
L2
L2
L2
Shared L3 Cache

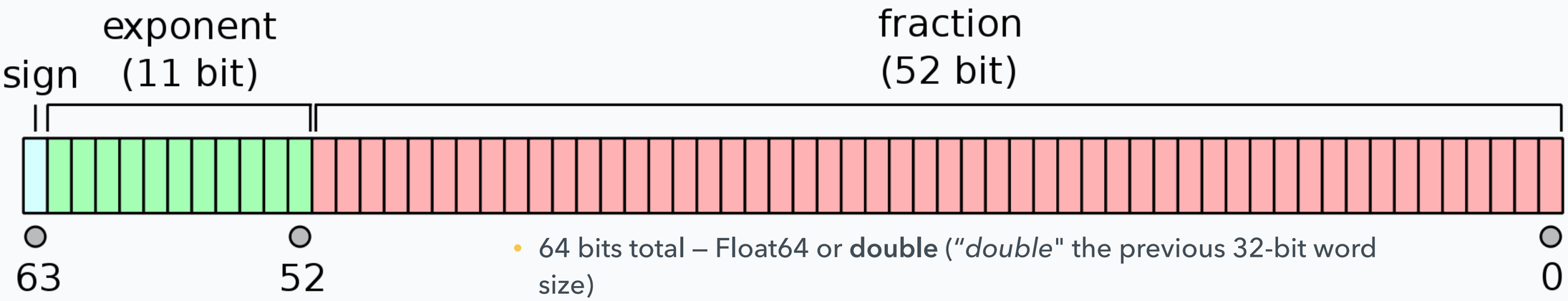
Part II:
Abstraction & Encoding

11101000 00011110 11010100 10010110 01000010
10101111 11000100 10001010 11011101 10011111
11101111 11011111
01101101 There are only 10 types of 00000100
00100000 people in the world: those 10001111
00110010 who understand binary, and 11111111
01110010 those who do not. 01011100
00110111 00110001
11000110 10100011 01101010 10010110 11000111
00001000 10010001 00101011 11011101 11100101
01101110 00101011 00011100 10011111 01101001
01010111 00100111 11111010 11111000 10111011
01111000 11000010 01000110 00100000 01111101

Physical State	OFF	ON	ON	OFF	OFF	ON	ON	OFF
Binary Notation	0	1	1	0	0	1	1	0
Order of Magnitude	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Value	128	64	32	16	8	4	2	1
Applicable Value	0	64	32	0	0	4	2	0
Total Decimal Value	102 = 64 + 32 + 4 + 2							

$-\frac{1}{2}$

IEEE 754: Floating-Point Signed Double



- 64 bits total – Float64 or **double** (“*double*" the previous 32-bit word size)
- 52 bits for the *fraction* of the *significand* (with an implicit “1” bit)
  - Decimal example: in 1.5204 x 10^4, the *fraction* is 5204
- 11 bits for the *exponent* – where the *bicimal point* should *float* to (order of mag.)
  - Many special rules – encoding patterns for ± 0, ± infinity, NaN, etc.
- 1 bit for *sign* (0 positive, 1 negative)

Hexadecimal Notation

Hex. digit	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Dec. notation	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bin. notation	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

- Base-16: uses sixteen different digits per "place" (order)
- To avoid ambiguity: prefaced `0x` or U+ or #, or subscript$_{16}$
- Much easier to convert b2 to/from b16 than to/from b10
- Example: what is `0x2a` (i.e. $2a_{16}$, U+2a) in b10?

someone who came after this.

PLATE I: KUNIMITSU

PAGE K337

Kunimitsu and Kunihiro cast a large shadow in the Kamakura period. In ancient times, the eldest son of an extended family was made the center and all worked together, and at the same time, in the pupil system, obtaining influence by changing to the name of even a famous person was not considered under ordinary circumstances. Also, as for the father-son, teacher-pupil relationship, the continuation of the one kanji of the name was the usual. I can think of the MITSU since Osafune Mitsutada and Nagamitsu, the KUNI of the Rai Mon, and the KUNI in the Sōshū smiths passed on to Kunihiro, the son of Kunimitsu. After Hiromitsu, the names of Hiromasa, Sukehiro, Hirotsugu etc. extended on into the Muromachi period.

The names in the columns, top to bottom, starting with the right hand column, are as follows.

KUNIMUNE, KUNIMITSU, KUNIHIRO, KUNISHI
(YAMASHIRO HASEBE)
UNITSUNA, YUKIMITSU, HIROMITSU
SAMUNE, SADAMUNE, AKIHIRO

line is a conjectural

PAGE K338

BUNPŌ SANNEN SANGATSU HI

PLATE I: BUNPŌ SANNEN SANGATSU HI
PLATE II: KUNIMITSU

According to the writings of the *Unchimyō Collection*, the Bunpō Sannen Kunimitsu at the right became the third generation.

KUNIMITSU UDA

[ŌJI 1362 ETCHŪ] CHŪKOTŌ JŌSAKU 國光

The early Kunimitsu that moved his residence from Yamato no Kuni Uda to Etchū and entered the priesthood was said to be of around Bunpō, but the existing Kunimitsu works are of around the Yoshinochō period. This smith is probably the father of Ōei Kunifusa and Kunimune nado. He signed with a small signature, hamon is sugu.

MEI: UDA KUNIMITSU

KUNIMITSU TANSHŪ JŪ

[ŌJI 1362 TAJIMA] 國光

He is the founder of the Hōjō-ji Ha, and is said to be one of the three disciples of Sōshū Sadamune, but along with the two disciples of Masamune, this is a story that [illegible] within the scope of the actual

MEI:

HELLO WORLD

. . . . . _ . . . _ . . _ _ _ . _ _ _ _ _ . _ . . _ . . _ . .

5
XYZABCDEFGH

ASCII

- American Standard Code for Information Interchange (1960s)
- 7-bit unsigned integers (stored in 8 bits) = 128 chars… +?

Binary	Decimal	Glyph	Binary	Decimal	Glyph
0010 1110	46	.	0011 1010	58	:
0010 1111	47	/	0011 1011	59	;
0011 0000	48	0	0011 1100	60	<
0011 0001	49	1	0011 1101	61	=
0011 0010	50	2	0011 1110	62	>
0011 0011	51	3	0011 1111	63	?
0011 0100	52	4	0100 0000	64	@
0011 0101	53	5	0100 0001	65	A
0011 0110	54	6	0100 0010	66	B
0011 0111	55	7	0100 0011	67	C
0011 1000	56	8	0100 0100	68	D
0011 1001	57	9	0100 0101	69	E

Unicode

- An encoding of every glyph in every human language
- Includes 🦄💩
- Obviously and forever incomplete
  - Unicode 1.0 (1991)
    - 24 writing systems
    - 7,161 code points
  - Unicode 9.0 (2016)
    - 135 writing systems
    - 128,237 code points

UTF-8

Glyph	Unicode Code Point	Code Unit 1: *singleton* or *lead*	Code Unit 2: *trail 1*	Code Unit 3: *trail 2*	Code Unit 4: *trail 3*
女	U+2F25	1110 0010	1011 1100	1010 0101	
❤	U+2764	1110 0010	1001 1101	1010 0100	
J	U+004A	0100 1010			
S	U+0053	0101 0011			
😄	U+1F604	1111 0000	1001 1111	1001 1000	1000 0100

- *Universal Coded Character Set* + *Transformation Format – 8-bit*
- Handles every *code point* of Unicode (1.1M+; ~260K assigned)
- *Variable-width*: one to four 8-bit unsigned integers (*code units*)
- Superset of ASCII – backwards-compatible

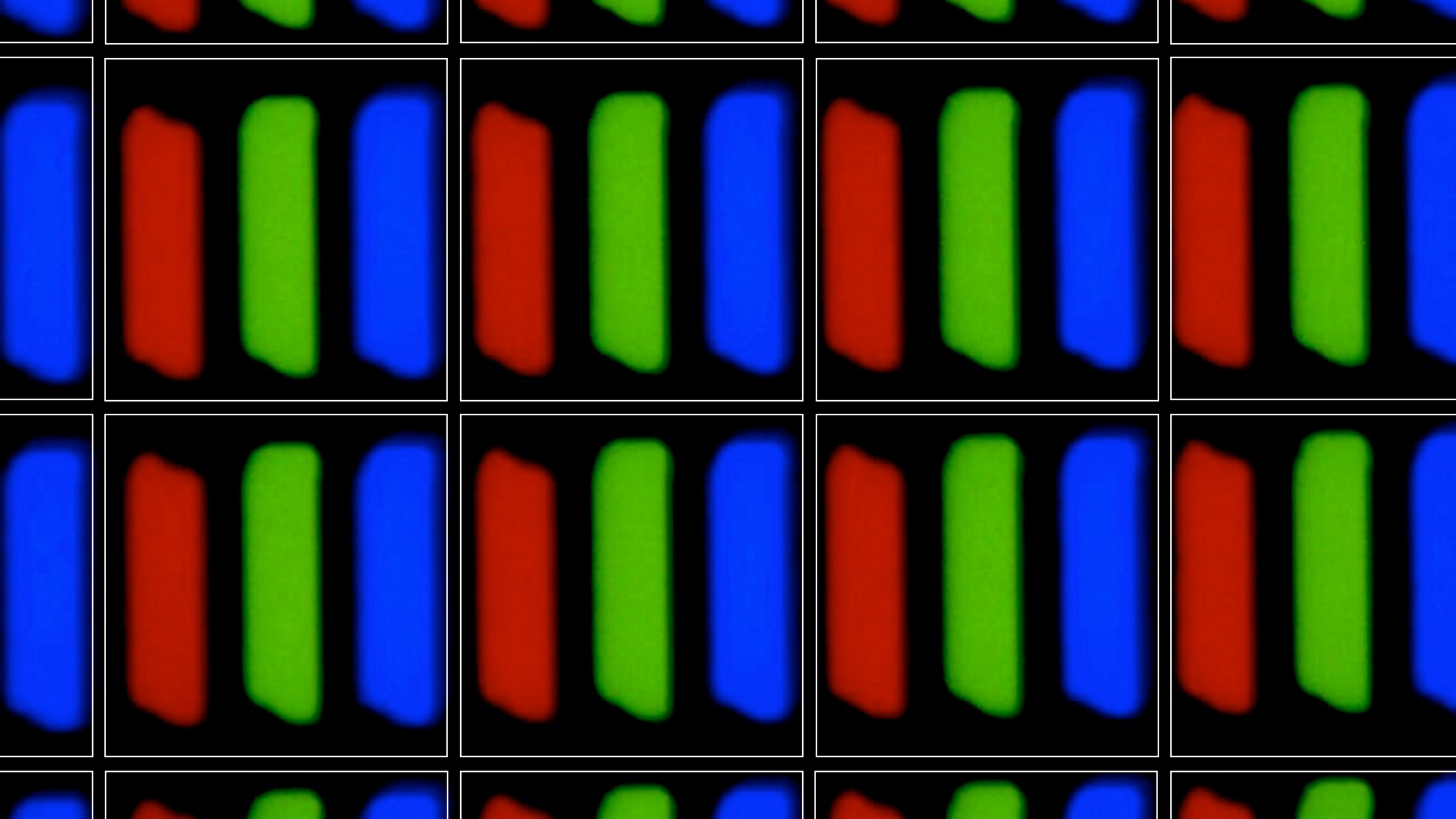
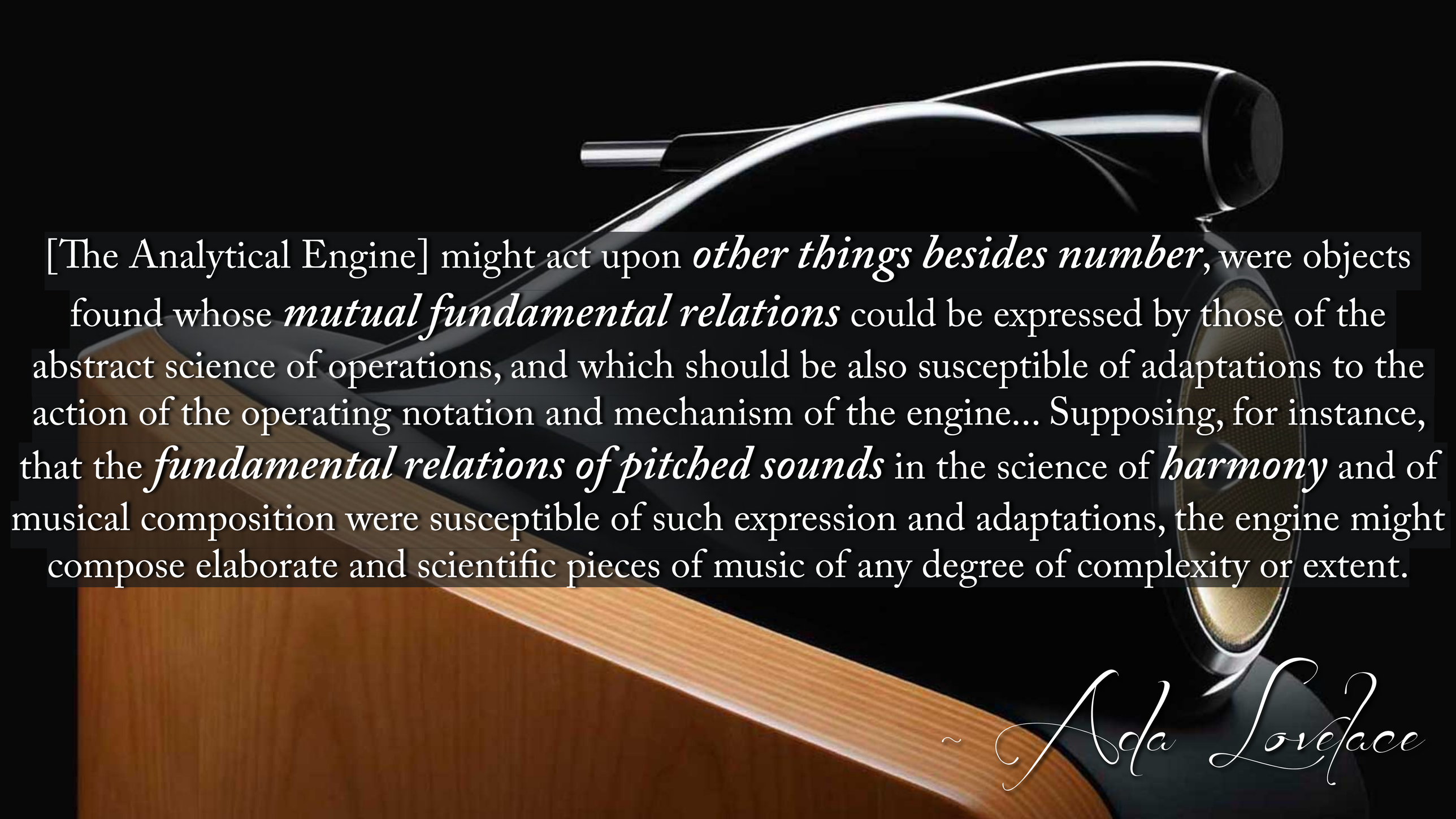


Image Encoding

- Simple *bitmaps* are triads of 8-bit unsigned ints for RGB values
- 8 bits makes 256 possible brightnesses per *channel*
- "24-bit" (8 + 8 + 8) monitors have 256^3 = ~16.8M RGB distinct channel combinations ("colors").
  - Since grayscale means RGB channels have to be the same, that's still only 256 possible B&W values.
- Want translucency? Add 8 bits of opacity information per pixel
  - 8 bits red + 8 bits green + 8 bits blue + 8 bits alpha = 32bpp


[The Analytical Engine] might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine... Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.
~ Ada Lovelace

Audio Encoding

- 16-bit PCM (pulse-code modulation)
  - 44,100 *samples* per second
  - each sample = an *amplitude* measurement with 16 bits of precision
- Different amplitudes over time models an audio wave
- Digital Audio Converter (DAC) converts digital values to analog signal that drives a speaker cone

Part III:
Software & APIs

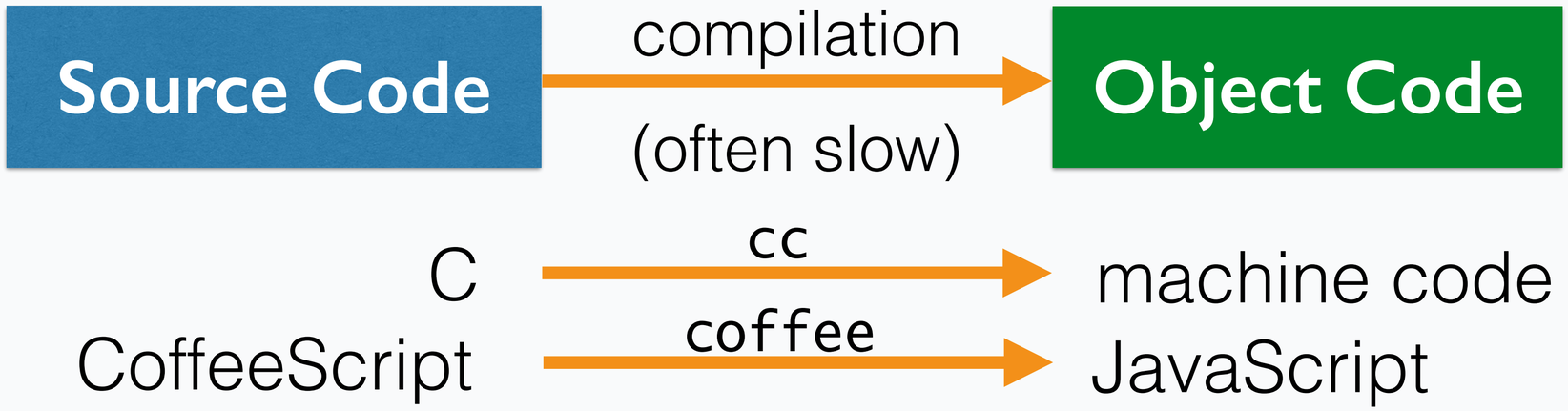
Machine Code & Assembly

- Every CPU has its own *instruction set* – binary codes that trigger the CPU to perform operations on registers.

	op. code	source register	target reg.	destination reg.	shift amount	function type
instruction	000000	00001	00010	00110	00000	100000
hex	0x0	0x1	0x2	0x6	0x0	0x20
decimal	0	1	2	6	0	32
meaning	arithmetic	register 1	register 2	register 6	no offset	addition
English	*"take the value in register 1, add the value in register 2, place the result in register 6"*					
assembly	`add $t6, $t1, $t2`					

Compilers

- First compiler written in 1952 by Grace Hopper, for A-0.
- *"[She] helped teach the machines a language, stopped them from speaking in undecipherable numbers."* — 60 minutes
- Followed by Mark 1, FORTRAN, COBOL, etc.



Languages & Memory Models

- Languages are a *specification* – an *agreement* about how things should work
  - How does data appear to be stored?
  - How can I read and write it?

C vs Javascript – memory models

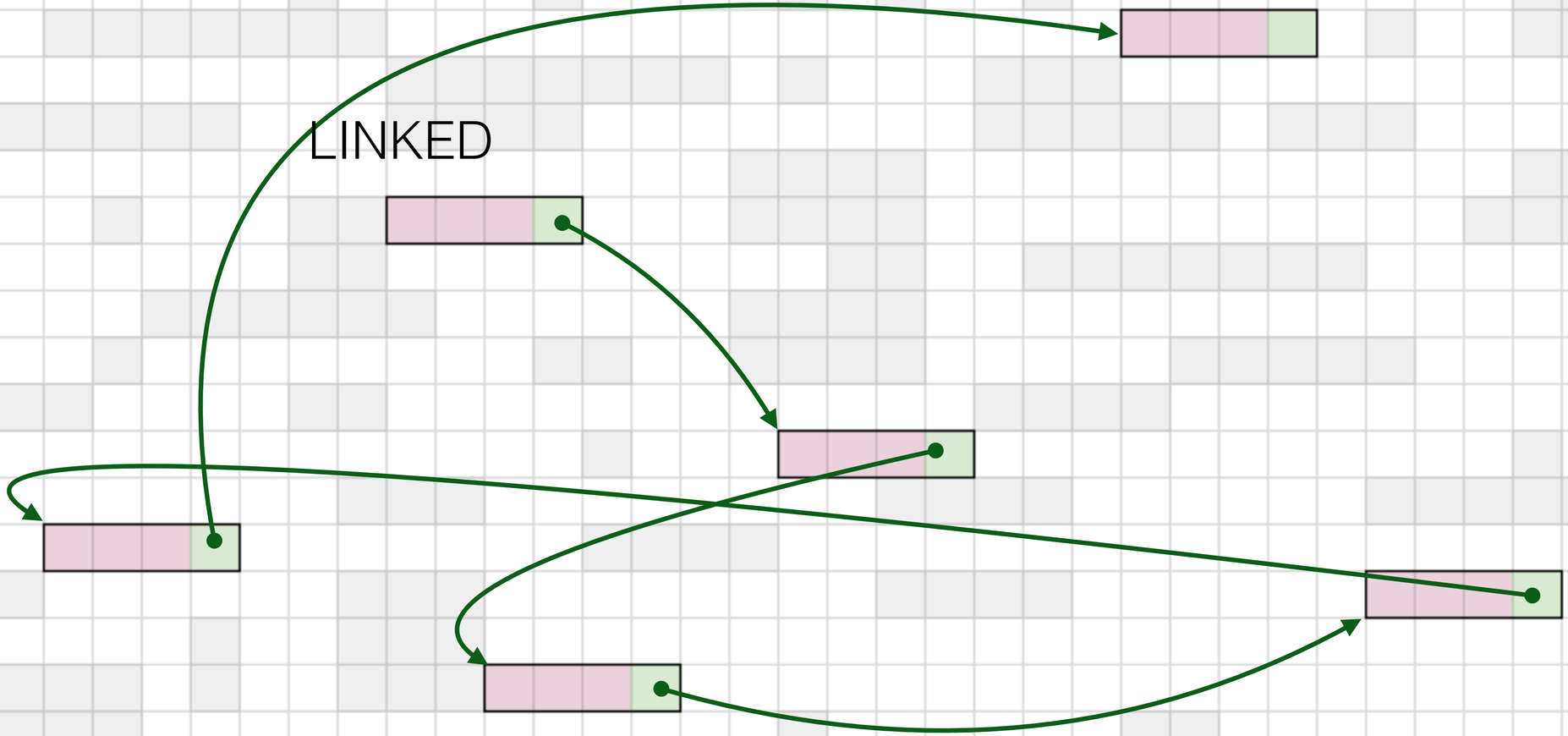
Javascript	C
High level memory model	**Low level memory model**
Memory must be accessed through objects or primitives	Memory addresses can be accessed directly `int *x = 0x1234;  // x points at the address 0x1234`
Arrays are built-in objects, `length` is a property	Arrays are pointers to their first element, length must be tracked separately `int *array = (int[3]) {12, 13, 14};`
Reading array elements out of bounds returns `undefined`	Reading array elements out of bounds returns whatever is at that memory location, may crash
Writing array elements out of bounds extends the array	Writing array elements out of bounds writes to that memory location with undefined results (crash if you're lucky, data corruption if not)

Part IV:
Abstract Data Types
& Data Structures

CONTIGUOUS

MEMORY

LINKED



JS	What happens	Memory
vals = new Int8Array(5)	allocate 5 bytes -> 0x3A6 store 0x3A6 in vals zero 5 bytes starting at 0x3A6	_ _ _ 0 0 0 0 0 _ _ _ _
vals[0] = 9	put 9 at 0x3A6 + 0 = 0x3A6	_ _ _ 9 0 0 0 0 _ _ _ _
vals[3] = 4	put 4 at 0x3A6 + 3 = 0x3A9	_ _ _ 9 0 0 4 0 _ _ _ _
vals[2] // 0	what is at 0x3A6 + 2 = 0x3A8	_ _ _ 9 0 0 4 0 _ _ _ _
vals[3] // 4	what is at 0x3A6 + 3 = 0x3A9	_ _ _ 9 0 0 4 0 _ _ _ _

The Stack ADT

- Collection of elements
- Ordered
- Elements can repeat (not a set)
- Some example operations:
  - Make a new stack
  - Add an element to the stack ("push")
  - Retrieve an element from the list ("pop") - *must be LIFO (Last In, First Out)*
  - Check if stack is empty
  - Look at the top element without removing it ("peek")
  - Clear the stack

The Stack ADT & Array DS

Stack Feature / Operation	Array Implementation with `top`
Collection of elements	Store values at memory addresses
Ordered	Sequential addresses maintain ordering
Create new stack	Initialize a new array
Push onto stack	Insert value at `top` var (next index to use)
Pop off of stack (LIFO)	Use `top` index to return latest value
Check if stack is empty	Return whether `top` variable is 0

ADTs vs DSs

Common Abstract Data Types	(Some) Data Structures
Set	List, Linked Tree, Trie, Hash Table, etc.
List	Linked List, Array
Stack	Array, Linked List
Queue	Linked List, Deque
Map (Associative Array / Dictionary / etc.)	Hash Table, Association List, Red-Black Tree
Graph	Adjacency List, Adjacency Matrix
Tree	Linked Tree, Heap

WHAT ABOUT JS?

Sorta-new concept: Built-In Types

- *ADT* or Abstract Data Type: describes the behavior we want
- *DS* or Data Structure: how to actually implement an ADT
- Built-In Type: the building blocks provided by a given language
  - language specifies an ADT—compiler authors decide what DS to use
  - specification may hint very strongly at an implementation
    "A List's length operation is guaranteed to be O(1)"

JavaScript Primitive (Immutable) Data Types

- `Undefined` · default for unassigned vars / nonexistent props
- `Null` · `typeof` is wrong; Null is a primitive, not "object"
- `Boolean` · could be 1 bit, but may not be *
- `String` · ES: engines use UTF-16 †
- `Number` · ES: 64-bit signed double float ‡
- `Symbol` · ES6: a unique token, can be key for objects

The JavaScript Non-Primitive Data Type

- `Object`
  - Mutable (can change)
  - Collection of *properties* – key-value pairs ("map")
  - Variable assignment copies reference, not value; it "points to" the object
  - Many other types are actually objects. Examples:
    - **Function**
    - **Array**
    - **Typed Array** (ES6-only)
    - **Regular Expression**, **Date**, others
    - Constructed primitives (**String**, **Number**, **Boolean**, etc.)

primitives are immutable

```
let x = 5
let y = x
  x = 8
y.foo = 'hello'
```

```
y.foo  // undefined
x.foo // also undefined
```

x → ~~5~~ 8

y → 5

Implementation detail: V8 treats Number primitives specially. They are always copied. They are guaranteed to be small, so it's ok.

objects aren't (by default)

```
let x = { age: 5 }
    let y = x
    y.age = 8
```

```
x → { age: 8 }
    y ↗
```

QUEUES & LINKED LISTS

The Queue ADT

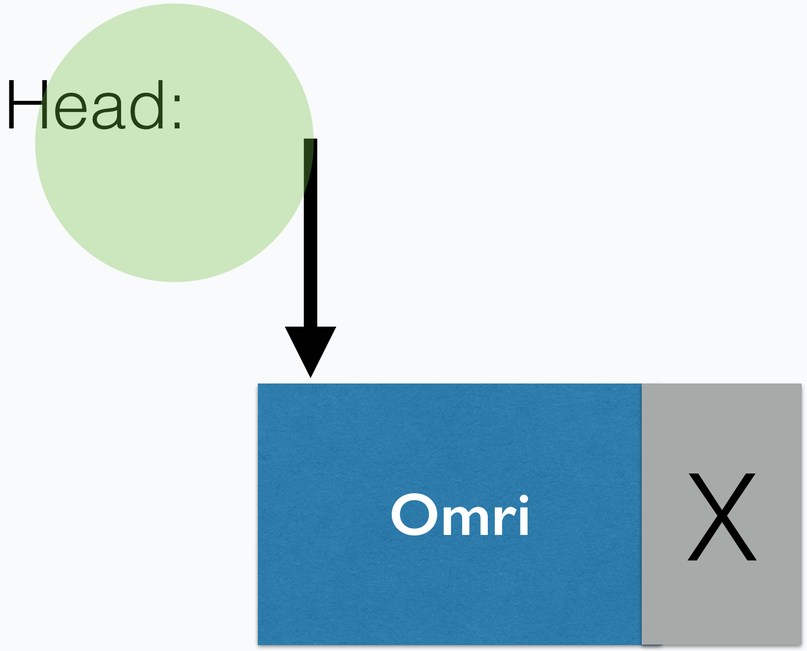
- Like Stack, except FIFO (First In, First Out)
- Collection of elements
- Ordered / sequential
- Operations:
  - Enqueue (add)
  - Dequeue (remove)
  - Peek
  - Clear
  - IsEmpty… etc.

The Linked List DS

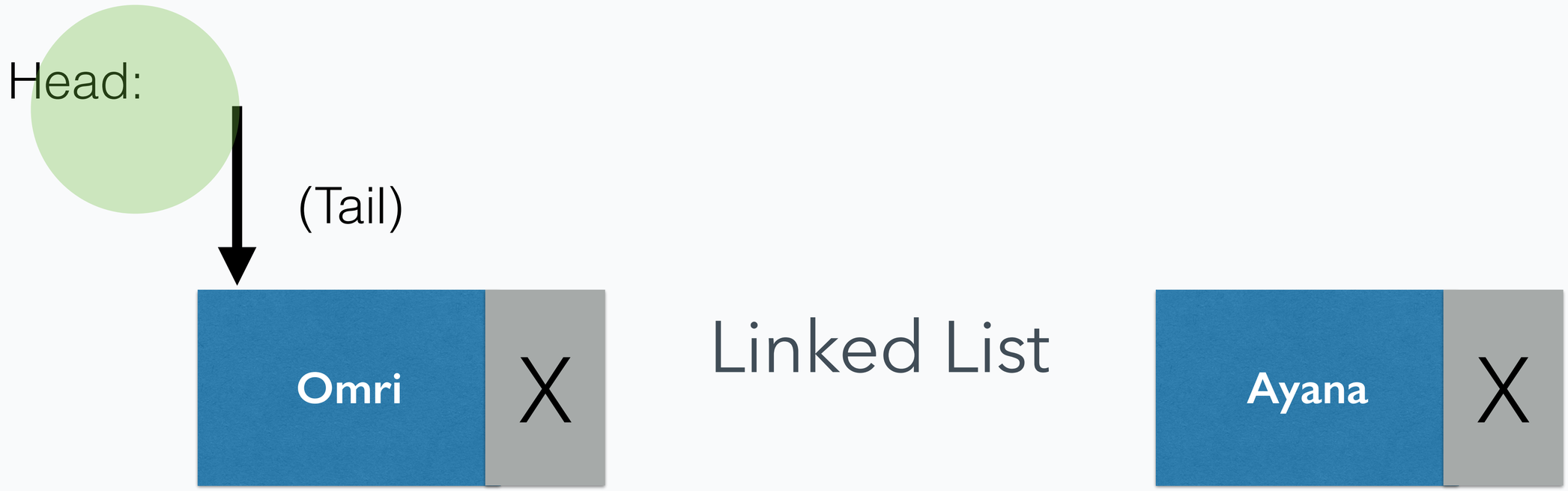
- Data structure used for storing sequences (*list, stack, queue, double ended queue)* ADTs etc.
- Uses *nodes* which encapsulate a value and pointer(s)
- Main entity holds reference(s) to just a head and/or tail node
  - the "handle(s)"
- Each node then *points* to the next and/or previous node
  - "singly-linked" (unidirectional) vs. "doubly-linked" (bidirectional)

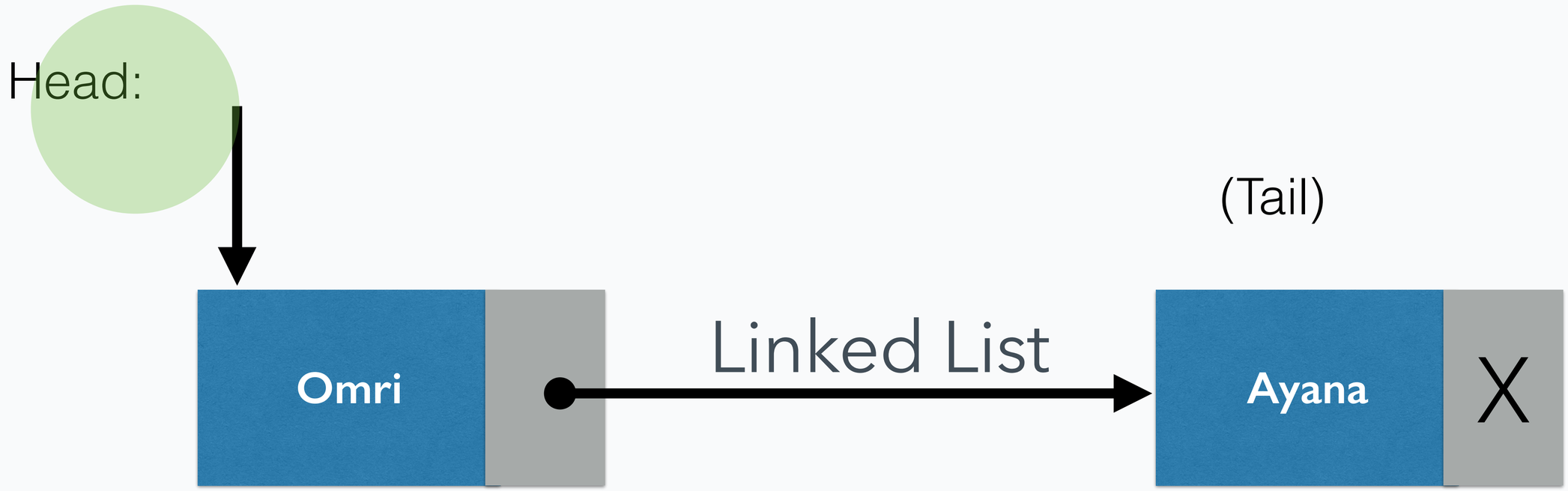
Head:

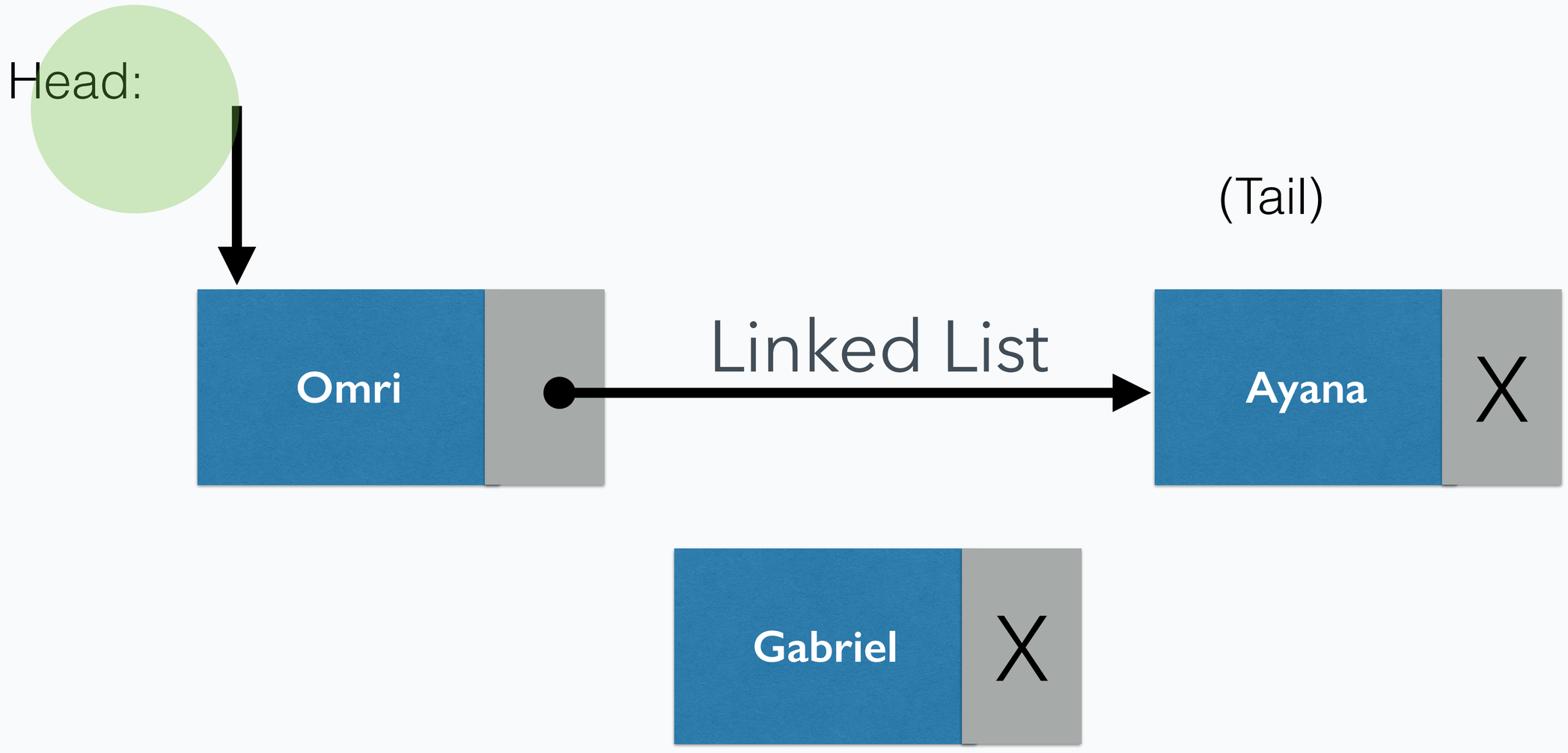
Linked List

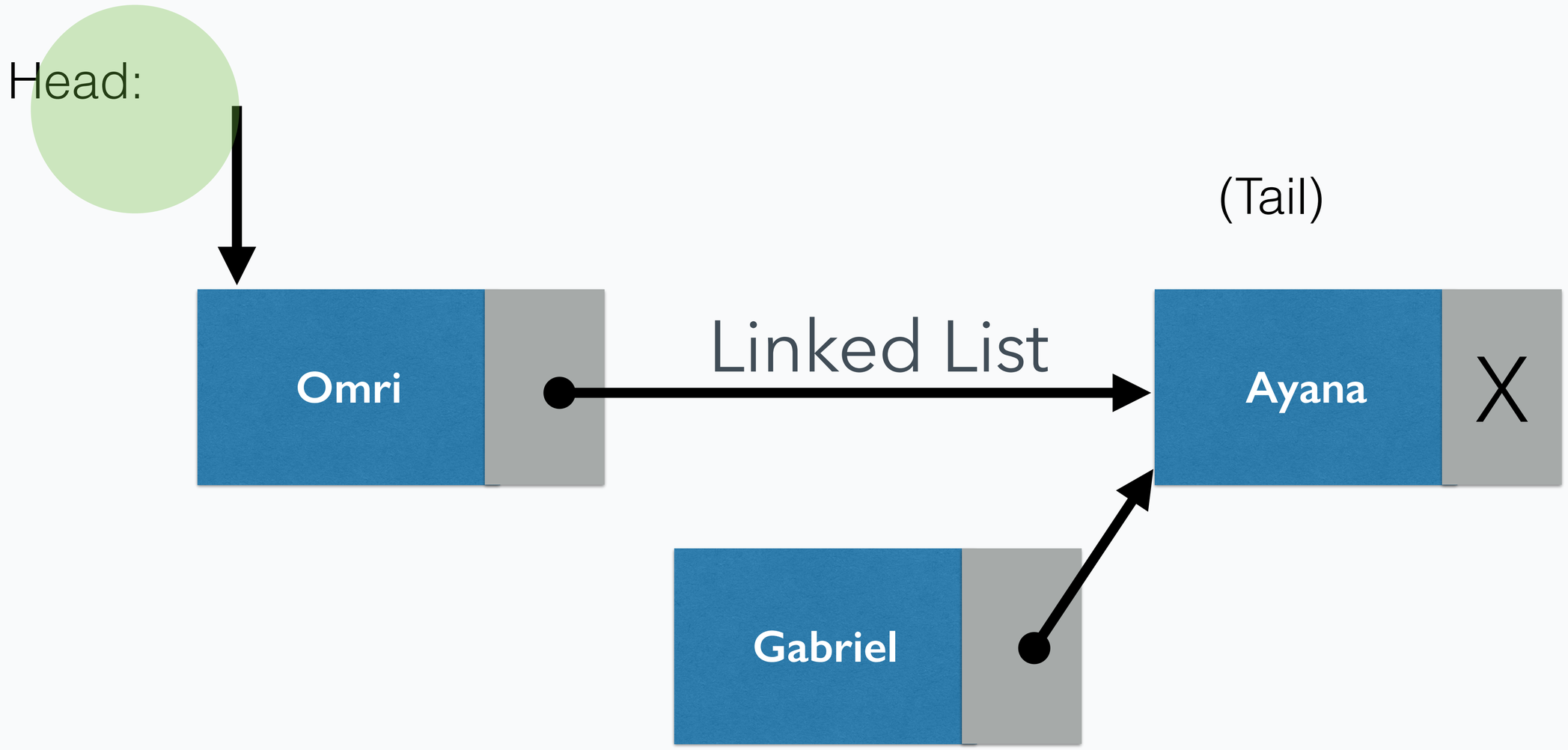


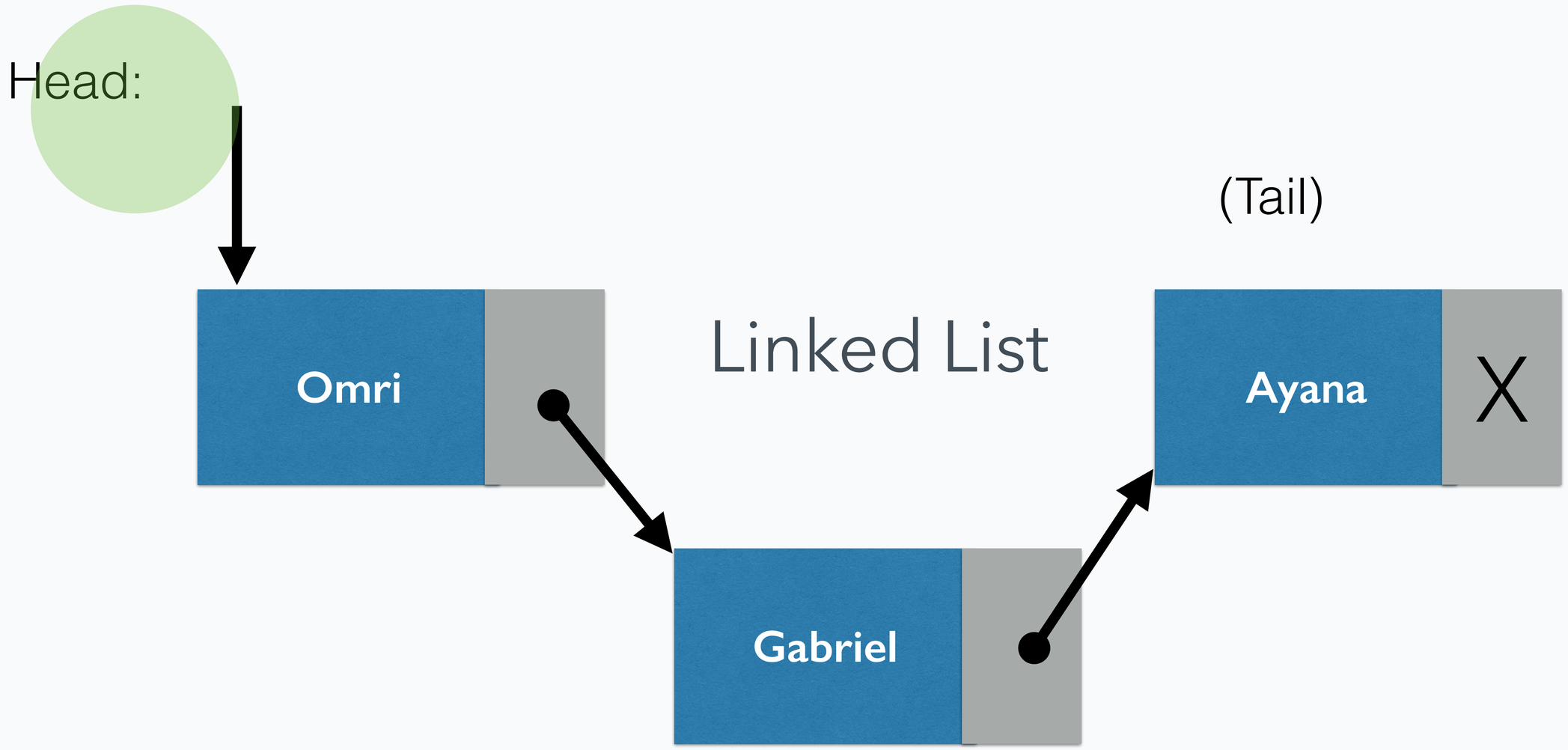
Linked List

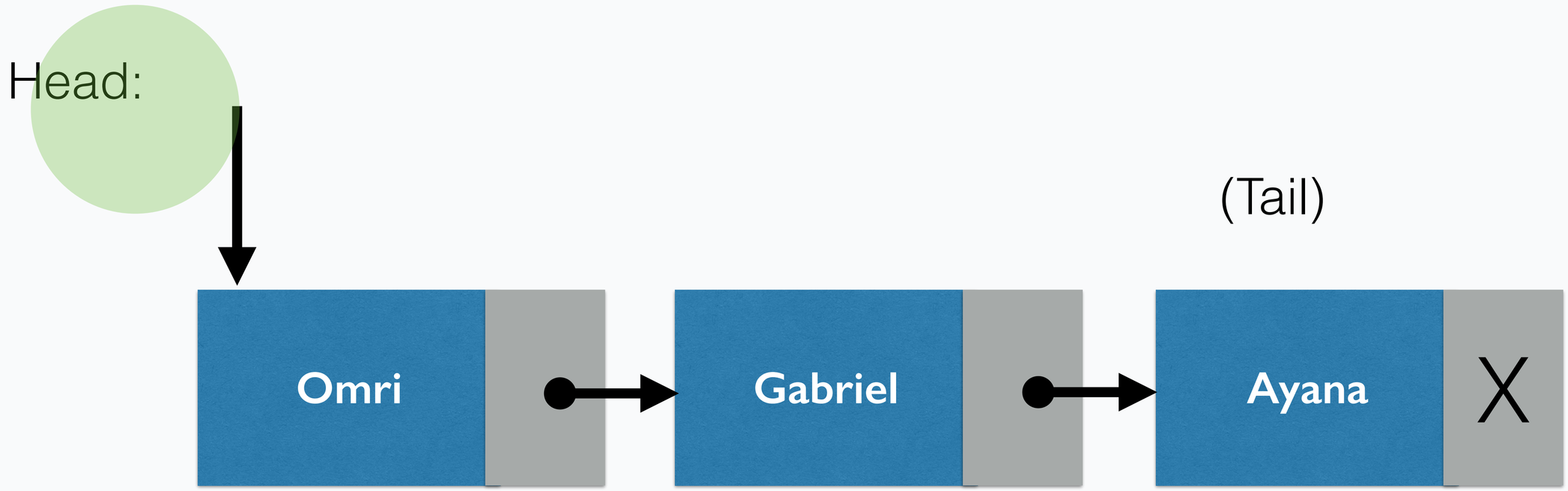

Head:
(Tail)
Omri
X
Linked List
Ayana
X


Head:
(Tail)
Omri
Linked List
Ayana
X

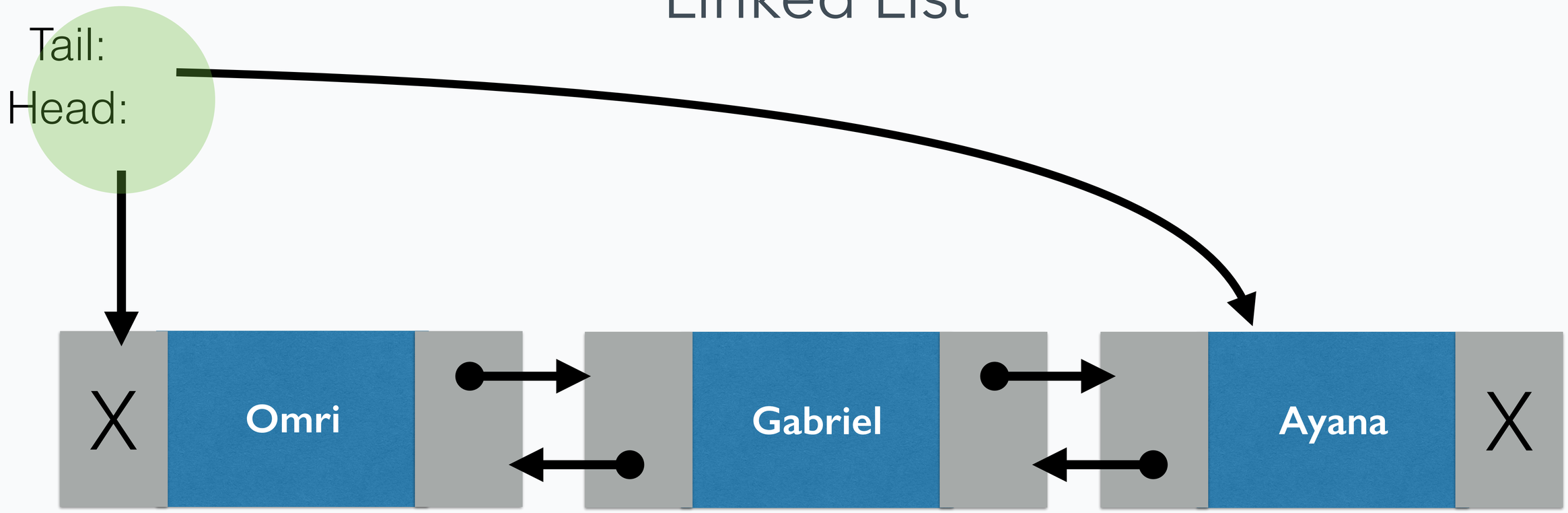

Head:
(Tail)
Omri
Linked List
Ayana
X
Gabriel
X


Head:
(Tail)
Omri
Linked List
Ayana
X
Gabriel


Head:
(Tail)
Omri
Linked List
Ayana
X
Gabriel


Head:
(Tail)
Omri
Gabriel
Ayana
X

Linked List



Operation	Linked List	Typed Array
Reach element in middle	Must crawl though nodes	Constant time
Insert in middle or start	Constant time (if we have ref).	Must move all following elements
Add element to end	With handle, constant time	Constant time
Space per element	Container + element + pointer(s)	Just element!
Total space	Grows as needed	Pre-reserved & limited*
Physical locality	Not likely	Best possible

WORKSHOP