
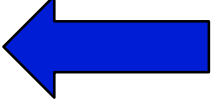


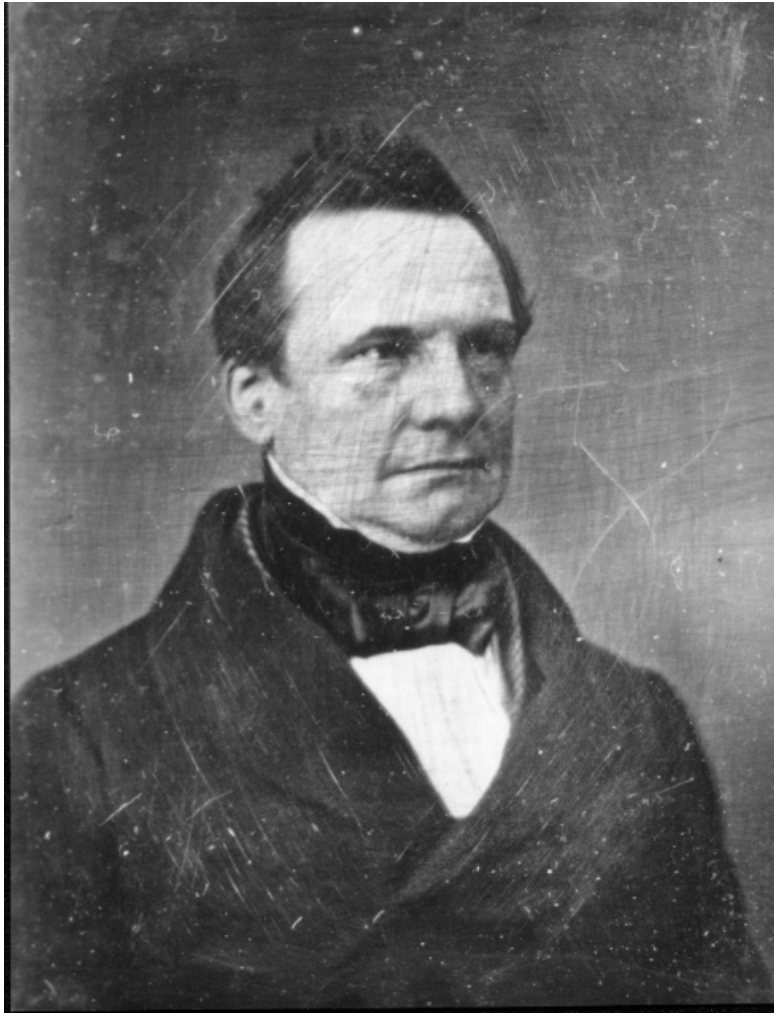
# Summer School Digital Tools for Humanists

Pisa – June 13-22 2023

## Refresher on Computer Fundamentals and Networking

- History of computers 
- Architecture of a computer 
- Data representation within a computer
- Computer networks and the Internet
- The Semantic Web

# Early visions



Charles Babbage  
1791-1871

Professor of  
Mathematics,  
Cambridge University,  
1827-1839

# Babbage's engines

- *Difference Engine*      1823
- *Analytic Engine*      1833
  - The forerunner of modern digital computer

## *Technology*

- mechanical gears, Jacquard's loom (1801), simple calculators

## *Programming*

- Ada Lovelace

## *Application*

- Mathematical Tables – Astronomy
- Nautical Tables – Navy

# Use of punched paper tape



# The organ grinder



# Early experiments 100 years later

- Z1 machine (Konrad Zuse, , private entrepreneur, 1936-1941)
- ABC (Atanasoff-Berry Computer, Iowa State University, 1937-1942)
- Mark I (Howard Aiken, MIT, 1937-1941)

1942 Second World War

- Built in 1944 in IBM Endicott laboratories
  - Howard Aiken – Professor of Physics at Harvard
  - Essentially mechanical but had some electro-magnetically controlled relays and gears
  - Weighed *5 tons* and had *750,000* components
  - A synchronizing clock that beat every *0.015* seconds (66KHz)

## Performance:

0.3 seconds for addition

6 seconds for multiplication

1 minute for a sine calculation

WW-2 Effort

*Broke down once a week!*

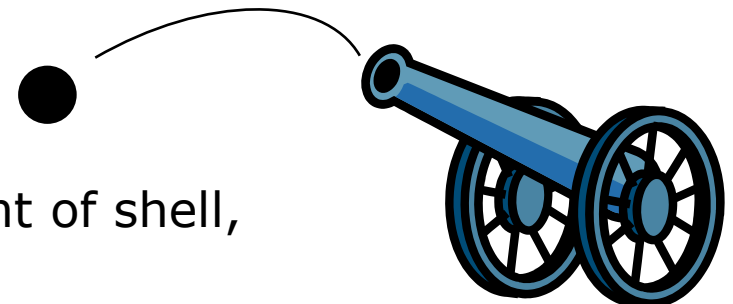


- Inspired by Atanasoff and Berry, Eckert and Mauchly designed and built ENIAC (1943-45) at the University of Pennsylvania
- The first, completely electronic, operational, general-purpose analytical calculator!
  - 30 tons, 72 square meters, 200KW
- Performance
  - Read in 120 cards per minute
  - Addition took 200  $\mu$ s, Division 6 ms
  - 1000 times faster than Mark I
- Not very reliable!

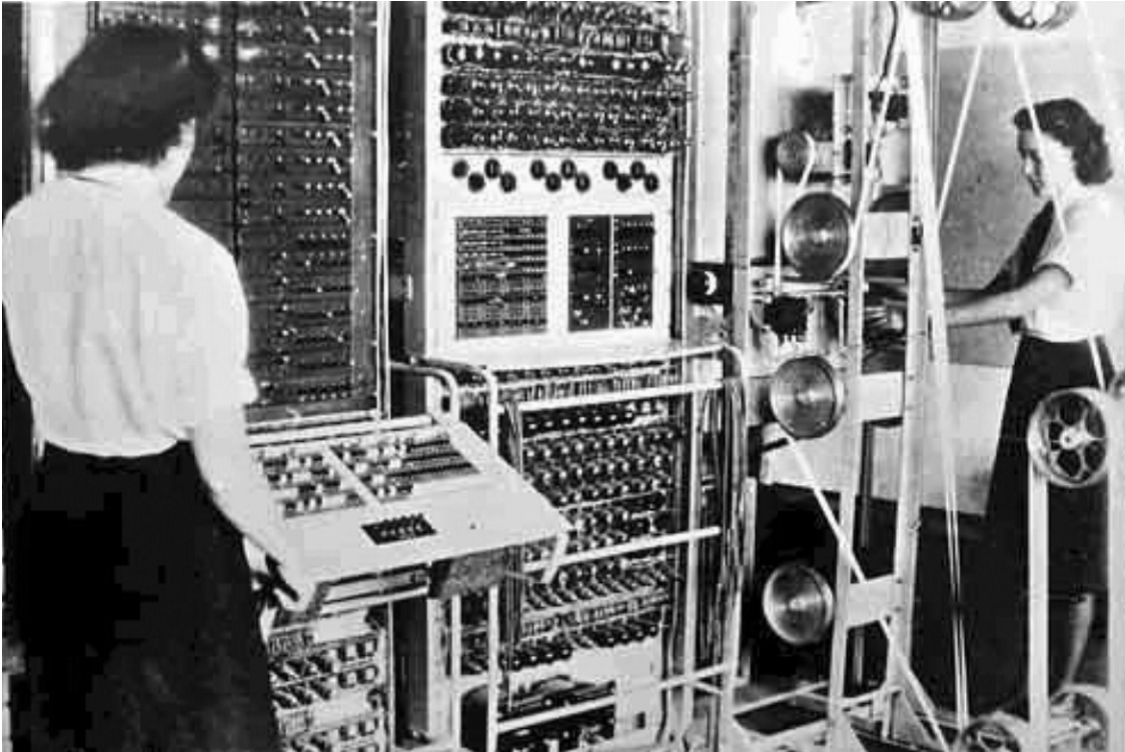
WW-2 Effort

*Application:* Ballistic calculations

angle = f (location, tail wind, cross wind,  
air density, temperature, weight of shell,  
propellant charge, ... )



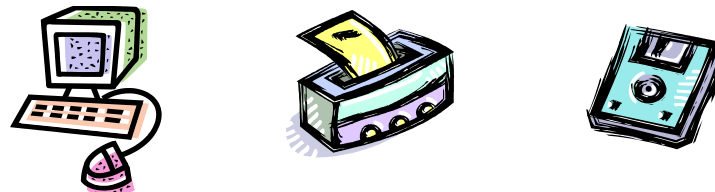
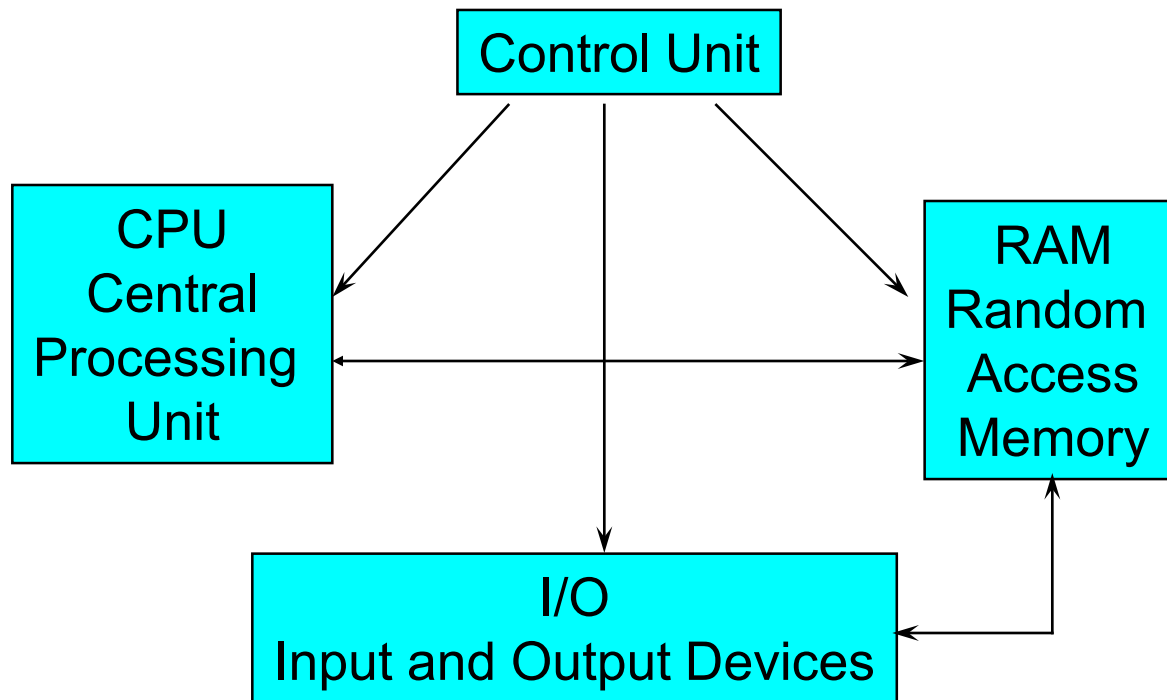
# Colossus



Colossus (derived in 1943 from Mark1 and Mark2) was used in London during the second World War to decipher encrypted German messages (Enigma machine)

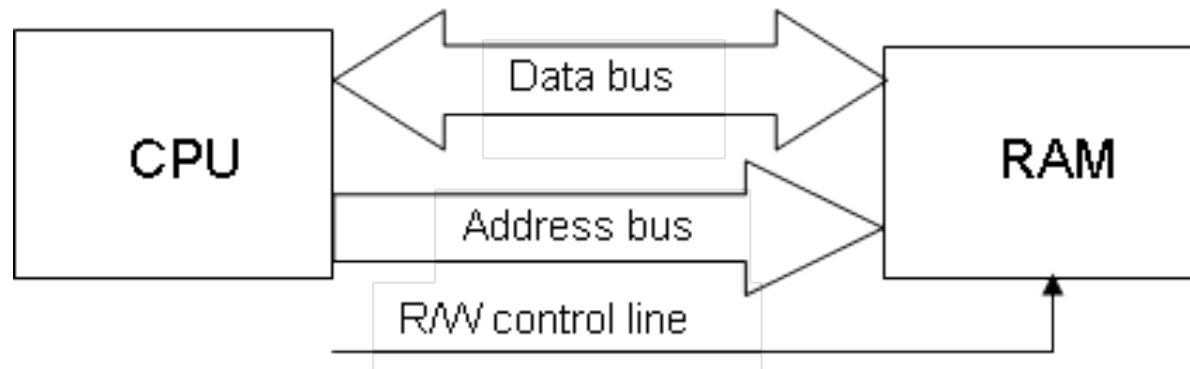
- ENIAC's programming system was external
  - Sequences of instructions were executed independently of the results of the calculation
  - Human intervention required to take instructions “out of order”
- Eckert, Mauchly, John von Neumann and others designed EDVAC (1944-1945) to solve this problem
  - Solution was the *stored program computer*
    - ⇒ “*program can be manipulated as data*”
- *First Draft of a report on EDVAC* was published in 1945, but just had von Neumann's signature
- In 1973 the court of Minneapolis attributed the honor of *inventing the computer* to John Atanasoff

# Von Neuman architecture



# Random Access Memory

- The RAM is a linear array of “cells”, usually called “words”. The words are numbered from 0 to N, and this number is the “address” of the word
- In order to read/write a word from/into a memory cell, the CPU has to provide its address on the “address bus”
- A “control line” tells the memory whether it is a read or write operation
- In a read operation the memory will provide on the “data bus” the content of the memory cell at the address provided on the “address bus”
- In a write operation the memory will store the data provided on the “data bus” into the memory cell at the address provided on the “address bus”, overwriting previous content



# Program execution

- The RAM contains both the program (machine instructions) and the data
- The basic model is “sequential execution”
  - each instruction is extracted from memory (in sequence) and executed
- Basic execution cycle
  - Fetch instruction (from memory) at location indicated by LC
  - Increment Location Counter (to point to the next instruction)
  - Bring instruction to CPU
  - Execute instruction
    - Fetch operand from memory (if needed)
    - Execute operation
    - Store result
      - in “registers” (temporary memory)
      - in memory (RAM)

- The Control Unit, the RAM, the CPU and all the physical components in a computer act on electrical signals and on devices that (basically) can be in only one of two possible states
- The two states are conventionally indicated as “zero” and “one” (0 and 1), and usually correspond to two voltage levels
- The consequence is that all the data within a computer (or in order to be processed by a computer) has to be represented in **binary notation**, i.e. with a sequence of 0s and 1s, **called bits**

# Evolution of computer technology

- First Generation  
mechanical/electromechanical
- Second Generation  
vacuum tubes
- Third Generation  
discrete transistors (solid state devices)  
SSI, MSI, LSI integrated circuits
- Fourth Generation  
VLSI integrated circuits

VLSI = Very Large Scale Integration



# Evolution of computer components

- Computer technology
  - CPU on integrated chips
    - From KHz to MHz to GHz
  - Random Access Memories
    - RAM – from KB to GB
  - External memories
    - Tapes, hard disks, floppy disks
    - Memory sticks
    - CDs
    - DVDs
    - from MB to GB to TB to PB to EB

# Size of digital information

1000	k	kilo
1000 <sup>2</sup>	M	mega
1000 <sup>3</sup>	G	giga
1000 <sup>4</sup>	T	tera
1000 <sup>5</sup>	P	peta
1000 <sup>6</sup>	E	exa
1000 <sup>7</sup>	Z	zetta
1000 <sup>8</sup>	Y	yotta

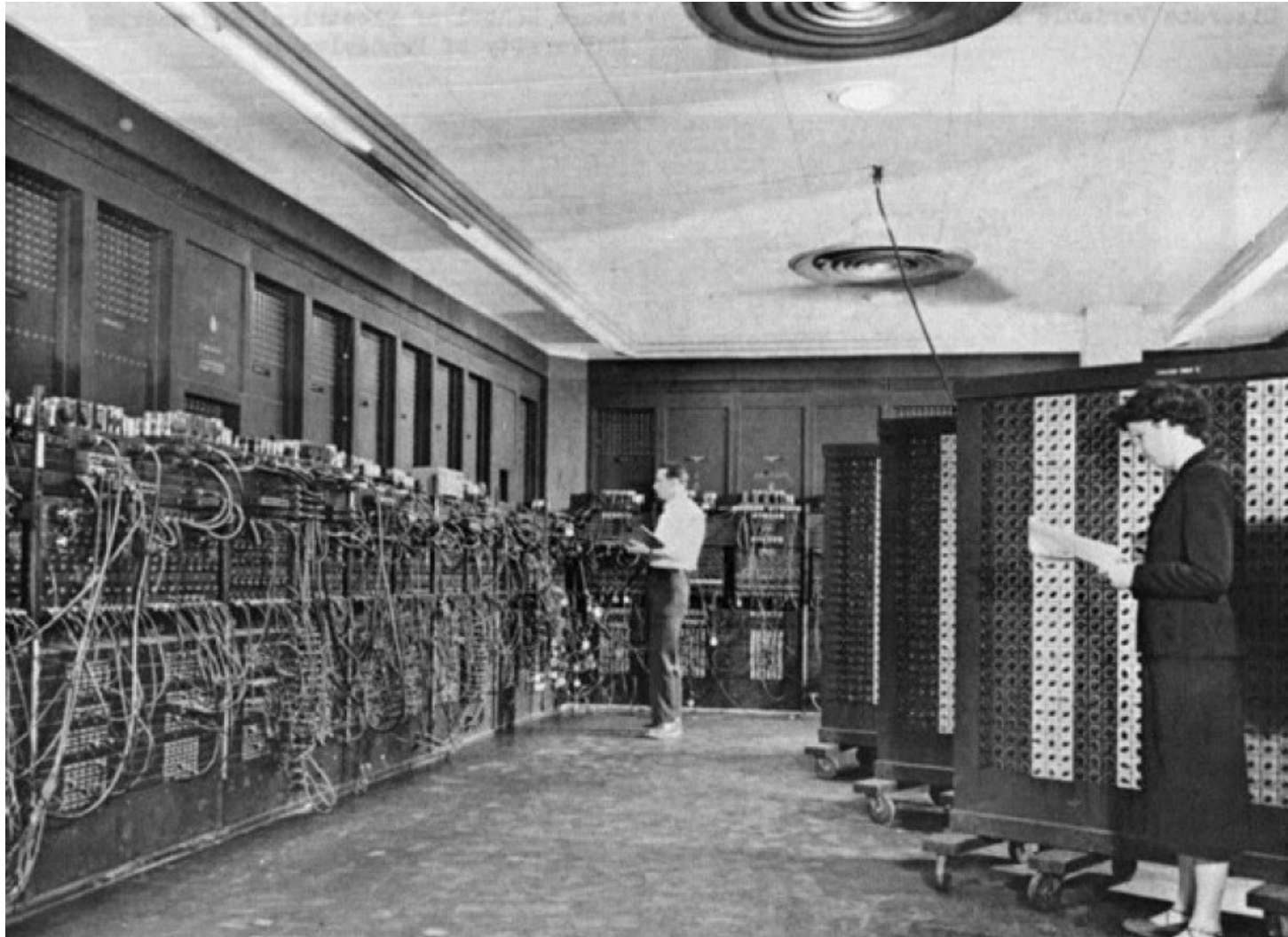
- Operating systems
  - Multi user
  - Multi tasking (e.g. Windows)
- Applications
  - Client-Server (e.g. the Web)
  - Multimedia
- Communication (networking)

- Military applications in early 40s
- Scientific/research applications in late 40s
- Commercial applications appear in early 50s
- Monopoly of IBM starts with 650, 701, 702
- Monopoly of IBM continues with 7070, 7090 and the 360 series, starting the “mainframe era” (in the 60s)
- Arrival of the “minicomputers” in the 70s
- Arrival of the PC in the 80s
- Arrival of the Internet in the 90s
- Arrival of the Web in the 90s

# Harvard Mark I, 1943

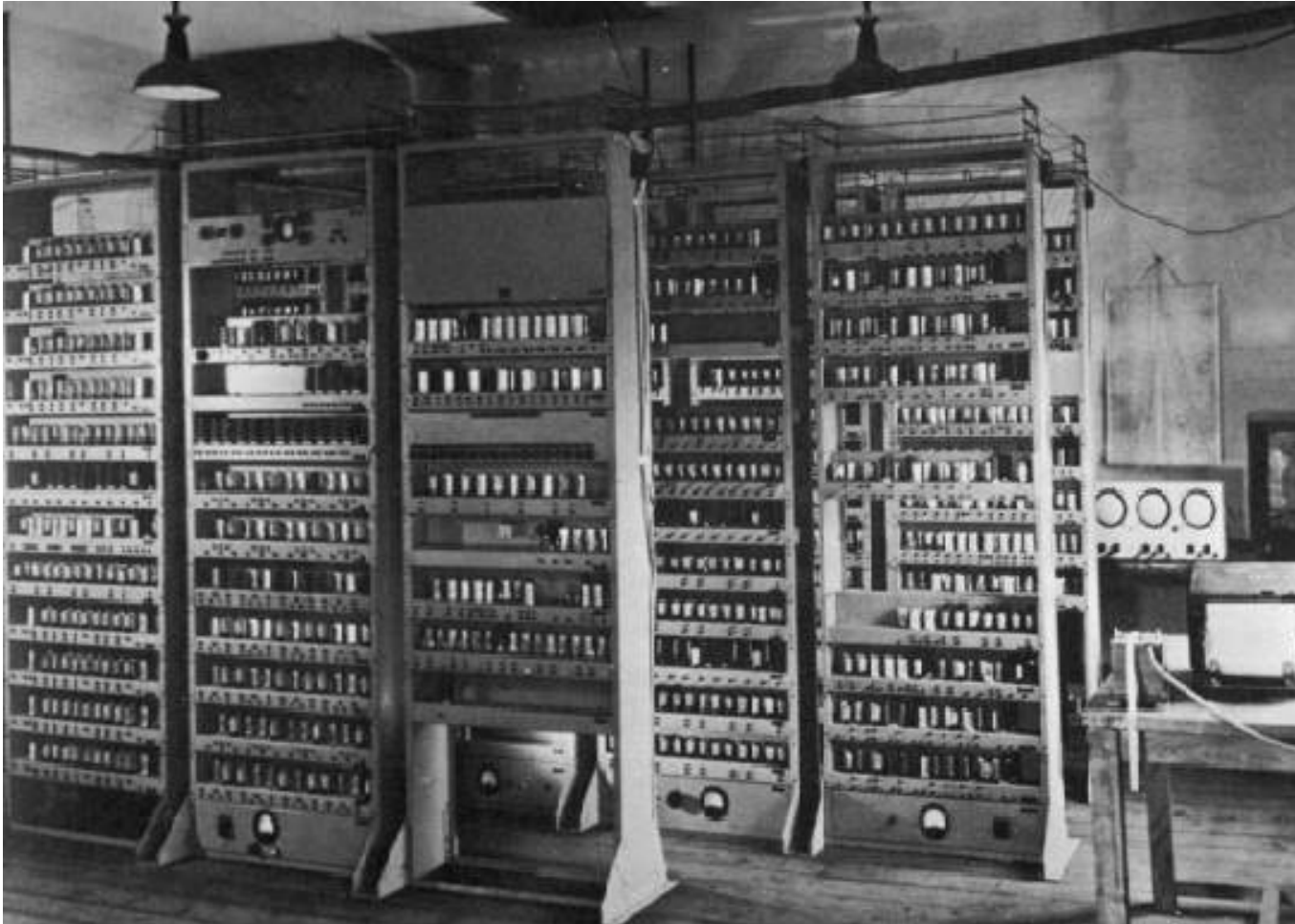


# ENIAC - Electronic Numerical Integrator And Computer (1945)



# EDSAC - Electronic Delay Storage Automatic Calculator

## EDSAC, University of Cambridge, UK, 1949



# A “mainframe” in the 60’





# A “mainframe” in the 70’



**Photograph: Dominic Hart/NASA Ames**

# Minicomputers (in the '70)



UNIPI SUMMER SCHOOL DIGITAL TOOLS 2023



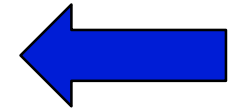
Vittore Casarosa – University of Pisa and ISTI-CNR

# Early PCs (in the '80)




## Refresher on Computer Fundamentals and Networking

- History of computers
- Architecture of a computer
- Data representation within a computer
- Computer networks and the Internet
- The Semantic Web



- The Control Unit, the RAM, the CPU and all the physical components in a computer act on electrical signals and on devices that (basically) can be in only one of two possible states
- The two states are conventionally indicated as “zero” and “one” (0 and 1), and usually correspond to two voltage levels
- The consequence is that all the data within a computer (or in order to be processed by a computer) has to be represented in **binary notation**, i.e. with a sequence of 0s and 1s, **called bits**

# Representation of information within a computer

- Numbers 
- Text (characters and ideograms)
- Documents
- Images
- Video
- Audio

Positional notation in base 10

Ten different symbols are needed for the digits (0,1,2,3,4,5,6,7,8,9)

The “weight” of each digit is a power of 10 (the base) and depends on its position in the number

$$10^0=1$$

$$10^1=10$$

$$10^2=100$$

$$10^3=1000$$

$$10^4=10000$$

<b>3</b>	<b>4</b>	<b>7</b>
----------	----------	----------

$$3 \times 10^2 + 4 \times 10^1 + 7 \times 10^0 = 347$$

Positional notation in base 8

Eight different symbols are needed for the digits (0,1,2,3,4,5,6,7)

The “weight” of each digit is a power of 8 (the base) and depends on its position in the number

$$8^0=1$$

$$8^1=8$$

$$8^2=64$$

$$8^3=512$$

$$8^4=4096$$

<b>3</b>	<b>4</b>	<b>7</b>
----------	----------	----------

$$3 \times 8^2 + 4 \times 8^1 + 7 \times 8^0$$

$$192 + 32 + 7 = 231$$



## Positional notation in base 16

Sixteen different symbols are needed for the digits (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)

The “weight” of each digit is a power of 16 (the base) and depends on its position in the number

$$16^0=1$$

$$16^1=16$$

$$16^2=256$$

$$16^3=4096$$

$$16^4=65536$$

<b>3</b>	<b>B</b>	<b>F</b>
----------	----------	----------

$$3 \times 16^2 + B \times 16^1 + F \times 16^0$$

$$3 \times 256 + 11 \times 16 + 15 \times 1$$

$$768 + 176 + 15 = 959$$

## Positional notation in base 2

Two different symbols are needed for the digits (0,1)

The “weight” of each digit is a power of 2 (the base) and depends on its position in the number

$$2^0=1$$

$$2^1=2$$

$$2^2=4$$

$$2^3=8$$

$$2^4=16$$

$$2^5=32$$

$$2^6=64$$

$$2^7=128$$

$$2^8=256$$

1		0		1		1
---	--	---	--	---	--	---

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

$$1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$$

$$8 + 0 + 2 + 1 = 11$$

# Powers of 2

$2^0=1$	$2^9=512$		
$2^1=2$	<b><math>2^{10}=1024</math></b>	<b>1K</b>	
$2^2=4$	$2^{11}=2048$	2K	
$2^3=8$	$2^{12}=4096$	4K	
$2^4=16$	$2^{13}=8192$	8K	
$2^5=32$	$2^{14}=16384$	16K	
$2^6=64$	$2^{15}=32768$	32K	
$2^7=128$	<b><math>2^{16}=65356</math></b>	<b>64K</b>	
<b><math>2^8=256</math></b>	.....		
	<b><math>2^{20}=1.048.576</math></b>	<b>1024K</b>	<b>1M</b>
	<b><math>2^{30}=1.073.741.824</math></b>	<b>1024M</b>	<b>1G</b>
	<b><math>2^{32}=4.271.406.736</math></b>	<b>4096M</b>	<b>4G</b>

# Binary and hexadecimal numbers

$2^0=1$   
 $2^1=2$   
 $2^2=4$   
 $2^3=8$   
 $2^4=16$   
 $2^5=32$   
 $2^6=64$   
 $2^7=128$   
 $2^8=256$

0000=0	1000=8
0001=1	1001=9
0010=2	1010=10 A
0011=3	1011=11 B
0100=4	1100=12 C
0101=5	1101=13 D
0110=6	1110=14 E
0111=7	1111=15 F
	10000=16 10

decimal and hexadecimal

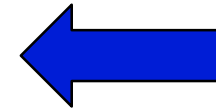
decimal

hexadecimal

0101|1011 can be represented  
in hexadecimal as 5B

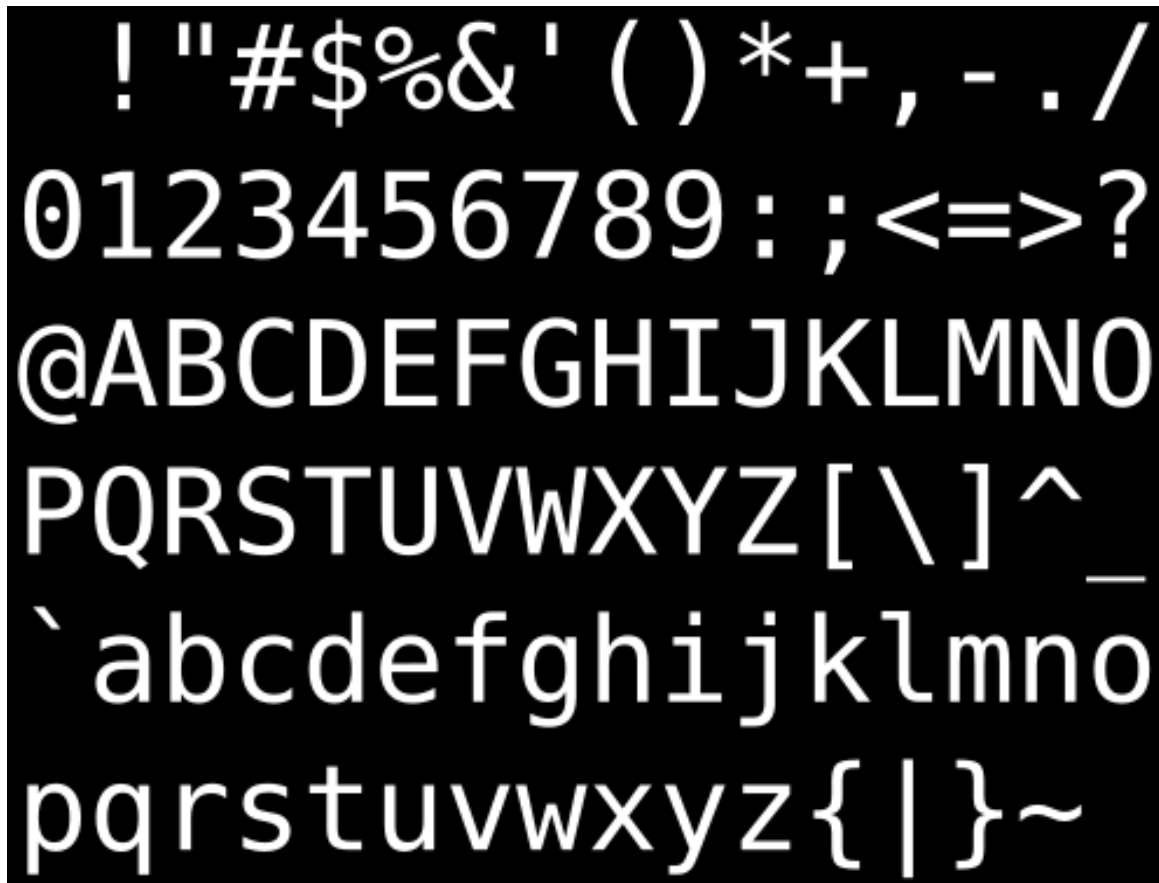
# Representation of information within a computer

- Numbers
- Text (characters and ideograms)
- Documents
- Images
- Video
- Audio



- The most simple way to represent (alphanumeric) characters (and symbols) within a computer is to associate a character (a symbol) with a number, defining a “coding table”
- How many bits are needed to represent the Latin alphabet ?

# The ASCII table (7 bits)



! " # \$ % & ' ( ) \* + , - . /  
0 1 2 3 4 5 6 7 8 9 : ; < = > ?  
@ A B C D E F G H I J K L M N O  
P Q R S T U V W X Y Z [ \ ] ^ \_  
` a b c d e f g h i j k l m n o  
p q r s t u v w x y z { | } ~

The 95  
printable  
ASCII  
characters,  
numbered  
from 32 to  
126 (decimal)

33 control  
characters

# ASCII table (7 bits)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□



# ASCII 7-bits character set

Last 4 bits

## 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

First 3 bits

- ASCII 7 bits (late fifties)
  - American Standard Code for Information Interchange
  - 7 bits for 128 characters (Latin alphabet, numbers, punctuation, control characters)
- EBCDIC (early sixties)
  - Extended Binary Code Decimal Interchange Code
  - 8 bits; defined by IBM in early sixties, still used and supported on many computers
- ASCII 8 bits (ISO 8859-xx) extends original ASCII to 8 bits to include accented letters and non Latin alphabets (e.g. Greek, Russian)
- UNICODE or ISO-10646 (1993)
  - Merged efforts of the Unicode Consortium and ISO
  - UNiversal CODE still evolving
  - It incorporates all(?) the pre-existing representation standards
  - Basic rule: round trip compatibility
    - Side effect is multiple representations for the same character

- Developed by ISO (International Organization for Standardization)
- There are 16 different tables coding characters with 8 bit
- Each table includes ASCII (7 bits) in the lower part and other characters in the upper part for a total of 191 characters and 32 control codes
- It is also known as ISO-Latin-xx (includes all the characters of the “Latin alphabet”)

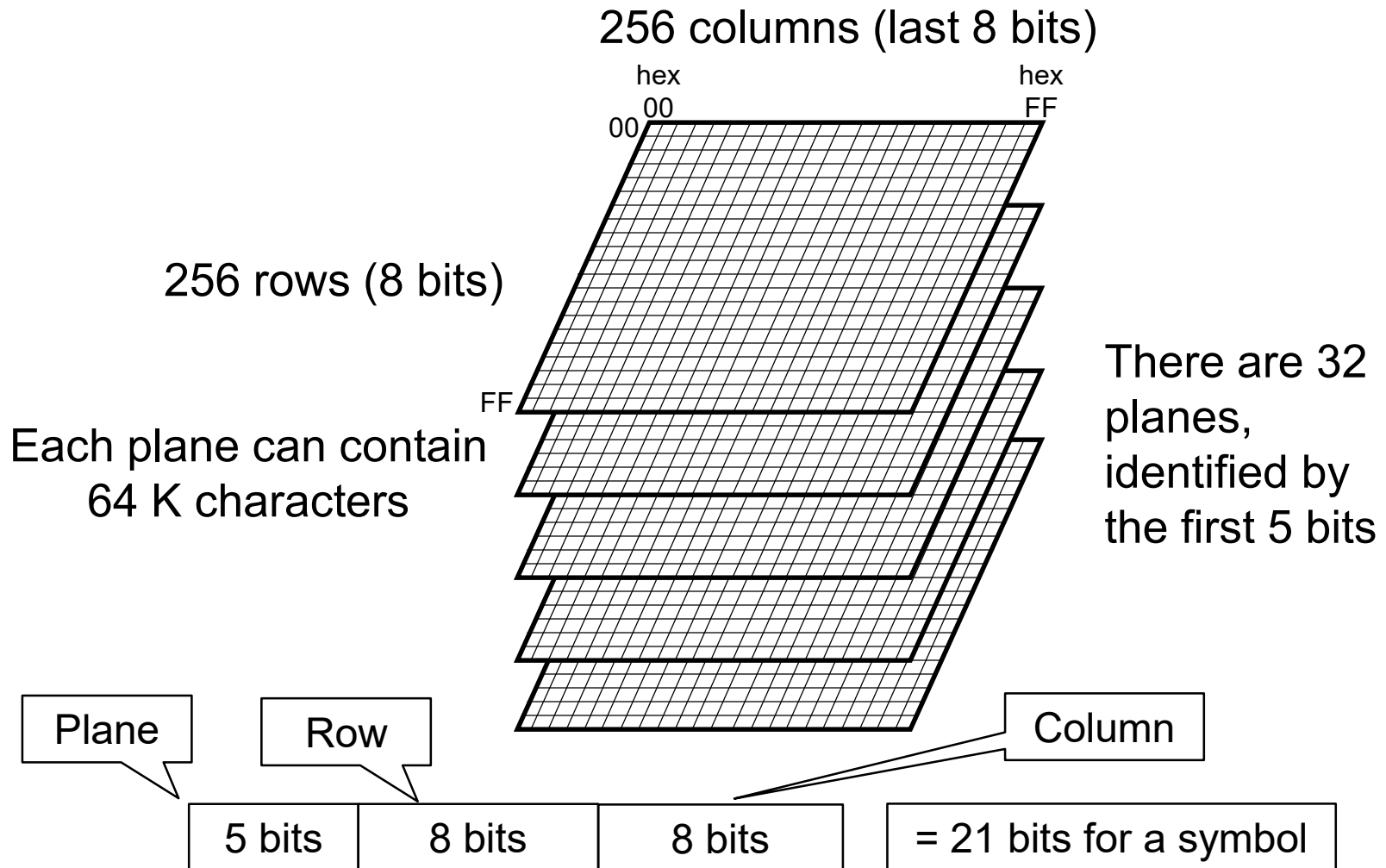
- 8859-1 Latin-1 Western European languages
- 8859-2 Latin-2 Central European languages
- 8859-3 Latin-3 South European languages
- 8859-4 Latin-4 North European languages
- 8859-5 Latin/Cyrillic Slavic languages
- 8859-6 Latin/Arabic Arabic language
- 8859-7 Latin/Greek modern Greek alphabet
- 8859-8 Latin/Hebrew modern Hebrew alphabet
- 8859-9 Latin-5 Turkish language (similar to 8859-1)
- 8859-10 Latin-6 Nordic languages (rearrangement of Latin-4)
- 8859-11 Latin/Thai Thai language
- 8859-12 Latin/Devanagari Devanagari language (abandoned in 1997)
- 8859-13 Latin-7 Baltic Rim languages
- 8859-14 Latin-8 Celtic languages
- 8859-15 Latin-9 Revision of 8859-1
- 8859-16 Latin-10 South-Eastern European languages

- ASCII (late fifties)
  - American Standard Code for Information Interchange
  - 7 bits for 128 characters (Latin alphabet, numbers, punctuation, control characters)
- EBCDIC (early sixties)
  - Extended Binary Code Decimal Interchange Code
  - 8 bits; defined by IBM in early sixties, still used and supported on many computers
- ISO 8859-1 extends ASCII to 8 bits (accented letters, non Latin characters)
- UNICODE or ISO-10646 (1993)
  - Merged efforts of the Unicode Consortium and ISO
  - UNiversal CODE still evolving
  - It incorporates all(?) the pre-existing representation standards
  - Basic rule: round trip compatibility
    - Side effect is multiple representations for the same character

- In Unicode, the word “character” refers to the notion of the abstract form of a “letter”, in a very broad sense
  - a letter of an alphabet
  - a mark on a page
  - a symbol (in a language)
- A “glyph” is a particular rendition of a character (or composite character). The same Unicode character can be rendered by many glyphs
  - Character “a” in 12-point Helvetica, or
  - Character “a” in 16-point Times
- In Unicode each “character” has a name and a numeric value (called “code point”), indicated by U+hex value.  
For example, the letter “G” has:
  - Unicode name: “LATIN CAPITAL LETTER G”
  - Unicode value: U+0047 (see ASCII codes)

- The Unicode standard has specified (and assigned values to) about 96.000 characters
- Representing Unicode characters (code points) in binary
  - 32 bits in ISO-10646
  - 21 bits in the Unicode Consortium
- In the 21 bit address space, we can take the last 16 bits to address a “plane” of 64K characters (256 rows by 256 columns)
- The first five bits can then identify one of the 32 possible planes
- Only 6 planes defined as of today, of which only 4 are actually “filled”
- Plane 0, the Basic Multilingual Plane, contains most of the characters used (as of today) by most of the languages present in the Web

# The planes of Unicode





# Beginning of BMP

in this table each “column” represents 16 characters

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	C0 Controls		<u>Basic Latin</u>						C1 Controls		<u>Latin 1 Supplement</u>					
01	<u>Latin Extended-A</u>						<u>Latin Extended-B</u>									
02	<u>Latin Extended-B</u>			<u>IPA Extensions</u>						<u>Spacing Modifiers</u>						
03	<u>Combining Diacritics</u>						<u>Greek</u>									
04	<u>Cyrillic</u>															
05	<u>Cyrillic Sup.</u>		<u>Armenian</u>						<u>Hebrew</u>							
06	<u>Arabic</u>															
07	<u>Syriac</u>			<u>Arabic Sup.</u>			<u>Thaana</u>			<u>N'Ko</u>						
08	<u>(Samaritan)</u>			<u>(Mandaic)</u>		???	???	???	???	ﻻArabic Extended-A?						
09	<u>Devanagari</u>						<u>Bengali</u>									
0A	<u>Gurmukhi</u>						<u>Gujarati</u>									
0B	<u>Oriya</u>						<u>Tamil</u>									
0C	<u>Telugu</u>						<u>Kannada</u>									
0D	<u>Malayalam</u>						<u>Sinhala</u>									
0E	<u>Thai</u>						<u>Lao</u>									
0F	<u>Tibetan</u>															
10	<u>Myanmar</u>						<u>Georgian</u>									

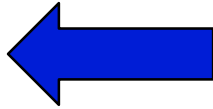
# Unicode encoding

- UTF-32 (fixed length, four bytes)
  - UTF stands for “UCS Transformation Format” (UCS stands for “Unicode Character Set”)
  - UTF-32BE and UTF-32LE have a “byte order mark” to indicate “endianness”
- UTF-16 (variable length, two bytes or four bytes)
  - All characters in the BMP represented by two bytes
  - The 21 bits of the characters outside of the BMP are divided in two parts of 11 and 10 bits; to each part is added an offset to bring it in the “surrogate zone” of the BMP (low surrogate at D800 and high surrogate at DC800)
  - in other words, they are represented as two characters in the BMP
  - UTF-16BE and UTF-16LE to indicate “endianness”
- UTF-8 (variable length, 1 to 4 bytes)
  - Characters in the 7-bit ASCII represented by one byte
  - Variable length encoding (2, 3 or 4 bytes) for all other characters

**Table 4.3 Encoding the Unicode character set as UTF-8.**

Unicode value	21-bit binary code	UTF-8 code			
U+00000000 – U+0000007F	0000000000000000wwwwwww	0wwwwwww			
U+00000080 – U+000007FF	0000000000wwwwxxxxxx	110wwww	10xxxxxx		
U+00000800 – U+0000FFFF	00000wwwwxxxxxyyyyyy	1110www	10xxxxxx	10yyyyyy	
U+00010000 – U+001FFFFF	wwwxxxxxyyyyyyzzzzzz	11110ww	10xxxxxx	10yyyyyy	10zzzzzz

# Representation of information within a computer

- Numbers
- Text (characters and ideograms)
- Documents 
- Images
- Video
- Audio

- Plain text

- No information about structure
- Different representation for line breaks

Knowledge of internal representation needed to extract text

- Windows represent a new line with the sequence “carriage return” followed by “line feed”
- Unix and Apple/Mac represent a new line with “line feed” only

- Page description languages

- PostScript
- PDF – Portable Document Format

- Word processors (text editors)

- RTF – Rich Text Format
- Microsoft Word
- LaTeX

Text editors

Editing of the contents

Editing of the format

- Mark-up languages

- WYSIWYG

(What You See Is What You Get)

- First commercially available page description language (Adobe 1985)
- It is a real programming language (variables, procedures, etc.) and a PostScript document is actually a “PostScript program”
- A page description comprises a number of graphical drawing instructions, including those that draw letters in a specific font in a specific size
  - Type-1 (Adobe) fonts versus TrueType (Apple)
- The document can be printed (or displayed) by having a “PostScript interpreter” executing the program
- The “abstract” PostScript description is converted to a matrix of dots (“rasterization” or “rendering”)
- PostScript initially designed for printing
  - Photo typesetters resolution up to 12000 dpi (dots per inch)
- PostScripts documents in a Digital Library
  - Extraction of text not always immediate
  - Digital Library must have a PostScript interpreter

- Successor to PostScript, to include good support for displays
- No longer a real programming language
- It defines an overall structure for a pdf document
  - Header, objects, cross-references, trailer
- Support for interactive display
  - Hierarchically structured content
  - Random access to pages
  - Navigation within a document
  - Support of hyperlinks
  - Support of “searchable images”
  - Limited editing capabilities
  - pdf/A “standard” format for “archiving”

- **.odt** – a text document
- **.ods** – a spreadsheet file
- **.odp** – a presentation file
- **.odg** – an illustration or graphic

Internal format is a zip-compressed XML document  
Versions of the OpenDocument Format are  
available for free download and use

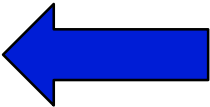


- Widely used in the scientific and mathematical communities
- Based on TeX, defined in the late seventies by Don Knuth, to overcome the limitations of the typesetters available at the time
- LaTeX documents are expressed in plain text, to expose all the details of the internal representation
  - Any text editor on any platform can be used to compose LaTeX document
  - Converted to a page description language (typically PostScript or PDF) to get the formatted document
- Simple document structure
  - Preamble to set the defaults and the global features
  - Structured (sections and subsections) document content
- Highly customizable with “external packages”
- Text extraction not so immediate
  - A single document may occupy several files
  - Possibility of “too much” customization

# Proprietary format Word (.doc, .docx)

- Last published specification is that of Word 97
  - Many changes since then
- Internal binary format (more fast and more compact with respect to the other formats)
- Abstract document structure similar to rtf documents
- More rich in functionality, and therefore more complicated
- The “Fast Save” option does not preserve the order of the text
  - Edits are appended at the end of the document
- For text extraction the best alternative is to save in some open format, such as odt, rtf, html

# Representation of information within a computer

- Numbers
- Text (characters and ideograms)
- Documents
- Images 
- Video
- Audio

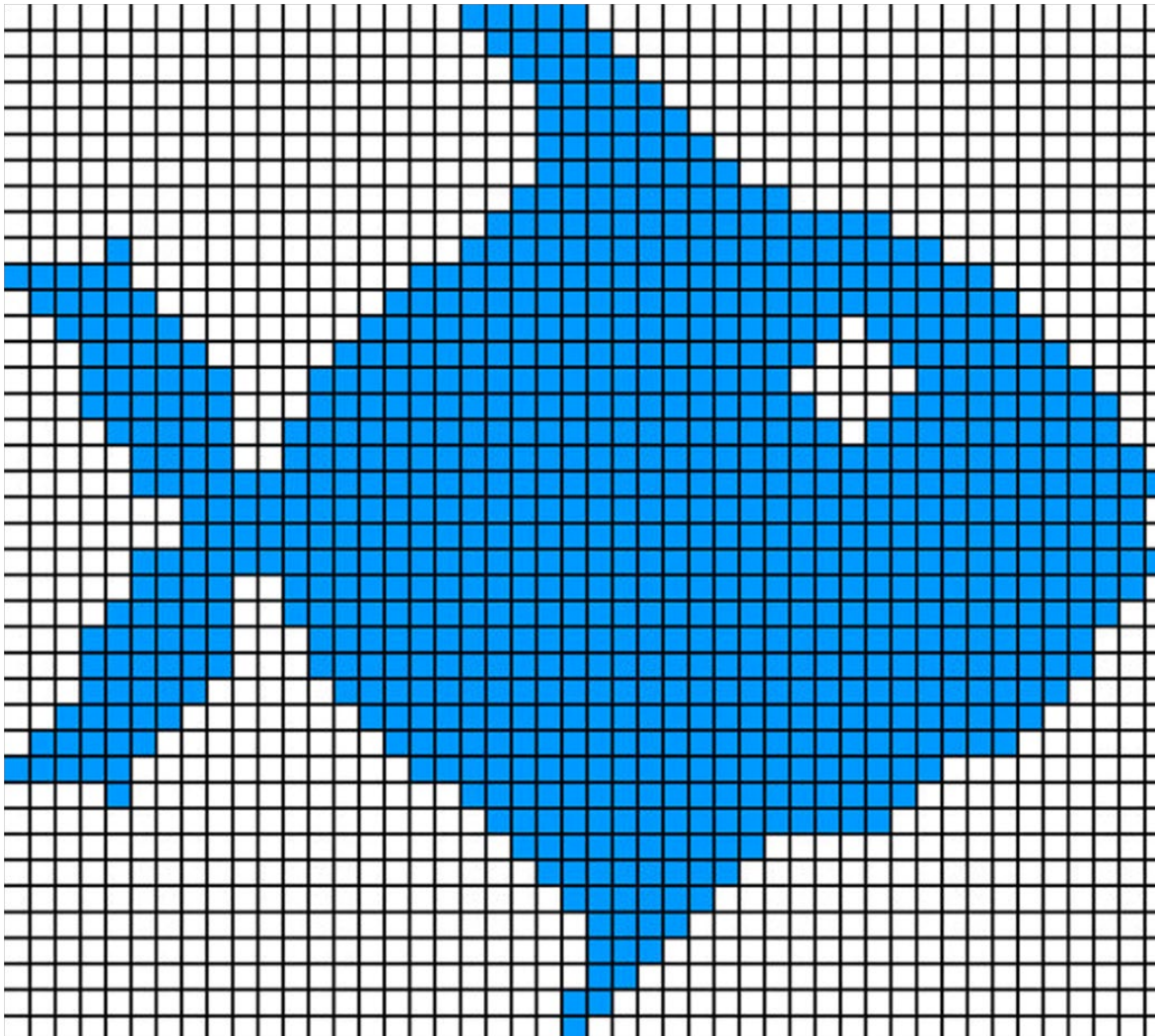


Welcome to image  
representation and  
compression



- Vector formats (geometric description)
  - Main advantage is scalability
    - Postscript
    - PDF
    - SVG (Scalable Vector Graphics)
    - SWF (ShockWave Flash)
      - vector-based images, plus audio, video and interactivity
      - Flash player obsolete since end of 2020
- Raster formats (array of “picture elements”, called “pixels”)

# Picture elements (pixels)



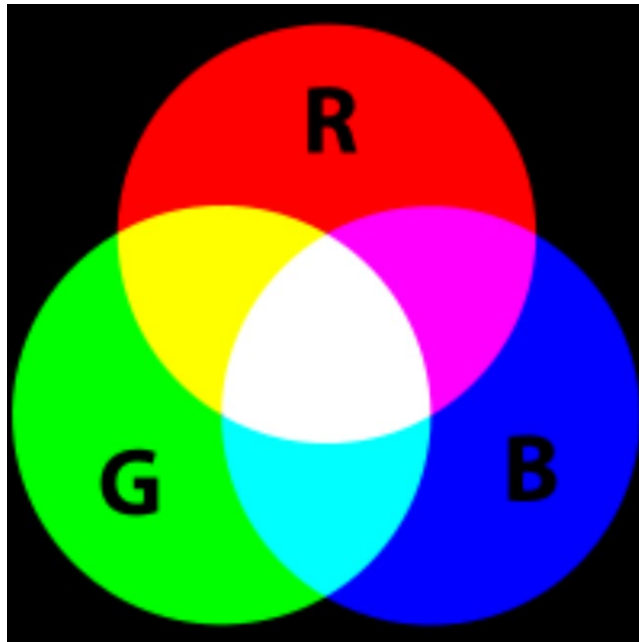
A pixel must be small enough so that its color can be considered uniform for the whole pixel. Inside the computer, a pixel is represented with a number representing its color.

- In raster format an image (picture) is represented by a matrix of “pixels”
- A first measure of the quality of a picture is given by the number of pixels, which can be measured in different ways
- Total number of pixels, as in digital cameras and phones
  - from 3-5 MegaPixels to 30-50 MegaPixels
- Number of rows and columns of the matrix, like in TV or PC screens (columns by rows)
  - HDTV 1920x1080, 4K TV 3840 x 2160,
  - PC screen 1024x768, 1280x1024, 1920x1080
- Number of pixels in 1 inch (2,54 cm), called “dpi” (dots per inch), in scanners and printers
  - 200-4800 dpi most common ranges

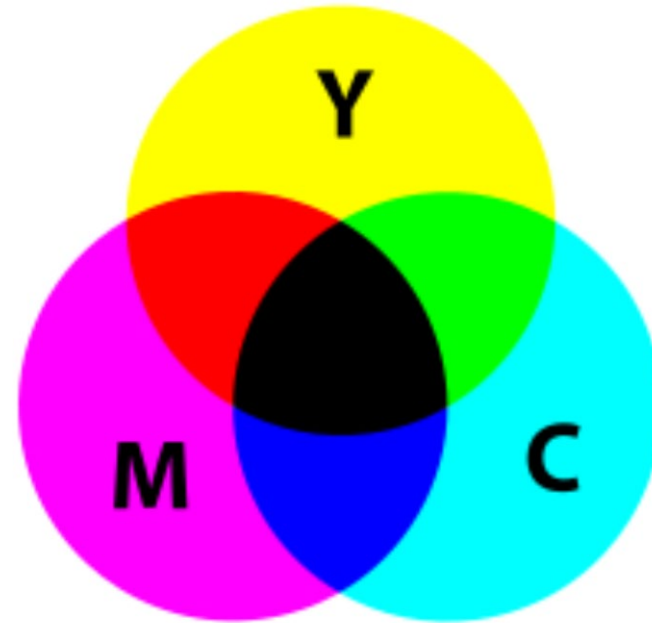
- In raster format an image (picture) is represented by a matrix of “pixels”
- The quality of a picture is determined also by the number of bits used to represent one pixel (called depth)
  - 1 bit for black and white
  - 8-16 bits for grey scale (most common ranges)
  - 12-48 bits for color images (most common ranges)
- Colors are represented by three numbers, one for each “color component”
- Big file sizes for (uncompressed color) pictures
  - For example, one color page scanned at 600 dpi is about 100 MB



# RGB and CMY color components



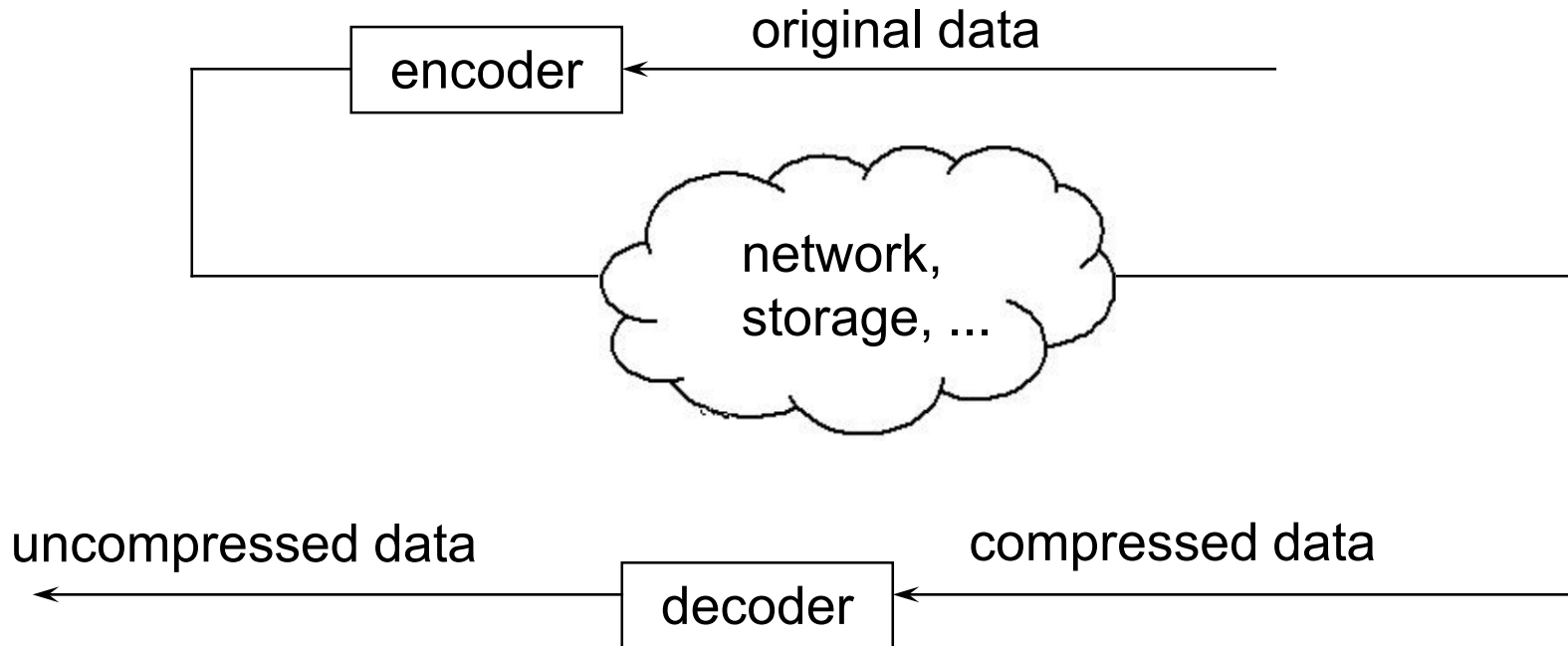
Additive color mixing



Subtractive color mixing

Big file sizes for raw pictures - Compression is needed

- Lossless compression
  - G3, G4, JBIG (fax)
  - GIF, PNG (simple graphics)
- Lossy compression
  - JPEG (all kind of images)
- BMP, RAW (sensor output), DNG (Digital Negative), etc.
- Tagged Image File Format (image container)
  - TIFF
- International Image Interoperability Framework
  - IIF



**lossless compression:** the uncompressed data is identical (bit by bit) to the original data

**lossy compression:** the uncompressed data contains less “information” than the original data

# Lossless compression techniques

- There are two main classes of lossless data compression methods
  - Symbol-wise encoding (Huffman)
  - Dictionary encoding (LZV)
- Symbolwise encoding
  - The basic idea is that the most frequent symbols can be coded with fewer bits (short codewords) than the less frequent symbols (long codewords)
  - Coders work by taking one symbol at the time from the input string, and coding it with a codeword whose length depends on the frequency (probability) of the symbol in the given alphabet
  - One of the most common symbol encoders is the Huffman coding
- Dictionary coding
  - The basic idea is to replace a sequence of symbols in the input string with an “index” in a dictionary (list) of “phrases”
  - .zip, .rar, .tar, etc

# Frequency distribution of the English letters

A	0.0856	0.1304	E	.
B	0.0139	0.1045	T	-
C	0.0279	0.0856	A	.-
D	0.0378	0.0797	O	---
E	0.1304	0.0707	N	-. .
F	0.0289	0.0677	R	.- .
G	0.0199	0.0627	I	..
H	0.0528	0.0607	S	... .
I	0.0627	0.0528	H	....
J	0.0013	0.0378	D	-..
K	0.0042	0.0339	L	-. . .
L	0.0339	0.0289	F	..- .
M	0.0249	0.0279	C	-.- .
N	0.0707	0.0249	M	--
O	0.0797	0.0249	U	..-
P	0.0199	0.0199	G	-- .
Q	0.0012	0.0199	Y	-.-.-
R	0.0677	0.0199	P	.-.- .
S	0.0607	0.0149	W	.-.-
T	0.1045	0.0139	B	-... .
U	0.0249	0.0092	V	...-
V	0.0092	0.0042	K	-. -
W	0.0149	0.0017	X	-...-
X	0.0017	0.0013	J	.-.-.-
Y	0.0199	0.0012	Q	--.-
Z	0.0008	0.0008	Z	--..

The Morse  
alphabet  
(1840)


# Lossless compression techniques

- There are two main classes of lossless data compression methods
  - Symbol-wise encoding
  - Dictionary encoding
- Symbolwise encoding
  - The basic idea is that the most frequent symbols can be coded with less bits (short codewords) than the less frequent symbols (long codewords)
  - Symbol coders work by taking one symbol at the time from the input string, and coding it with a codeword whose length depends on the frequency (probability) of the symbol in the given alphabet
  - One of the most common symbol encoders is the Huffman coding
- Dictionary coding
  - The basic idea is to replace a sequence of symbols in the input string with an “index” in a dictionary (list) of “phrases”
  - .zip
  - .rar
  - .tar

# Lempel Ziv 77 coding (1/4)

Repeat occurrences of character sequences?  
— replace them with a pointer back to the first occurrence

to • be • or • not • to • be , • that • is • ...



Pointer is <position,length>

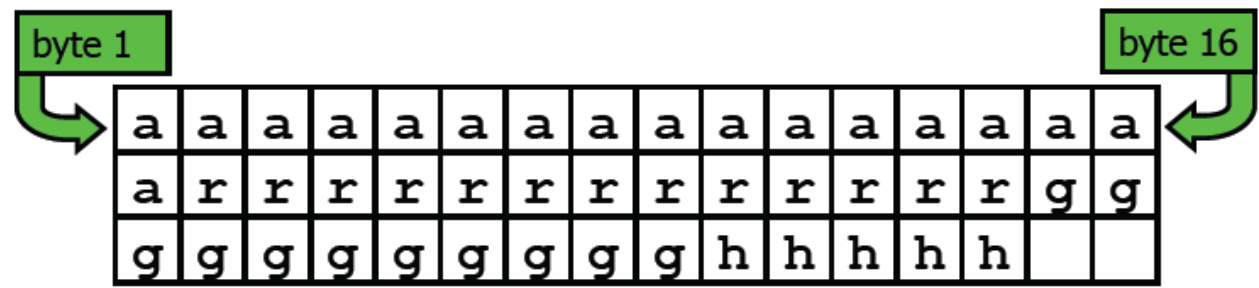
to • be • or • not • <1,5> , • that • is • ...

In both are <256, pointer fits in 2 bytes

# Lempel Ziv 77 coding (2/4)

The more regular the input, the better the compression

aaaaaaaaaaaaaaaaarrrrrrrr  
 rrrrrggggggggggghhhhh



a	1	16	r	18	12	g	31	10	h	42	4				



# Dictionary built while encoding and decoding

\_phrase 0 = NUL  
 phrase 1 = SOH  
 phrase 2 = STX  
 phrase 3 = ETX  
 phrase 4 = EOT  
 phrase 5 = ENQ  
 phrase 6 = ACK  
 phrase 7 = BEL  
 phrase 8 = BS  
 phrase 9 = HT  
 phrase 10 = LF  
 phrase 11 = VT  
 phrase 12 = FF  
 phrase 13 = CR

Initial vocabulary

.....  
 phrase 97 = a  
 phrase 98 = b  
 phrase 99 = c  
 phrase 100 = d  
 phrase 101 = e  
 phrase 102 = f  
 .....  
 phrase 123 = {  
 phrase 124 = |  
 phrase 125 = }  
 phrase 126 = ~  
 phrase 127 = DEL

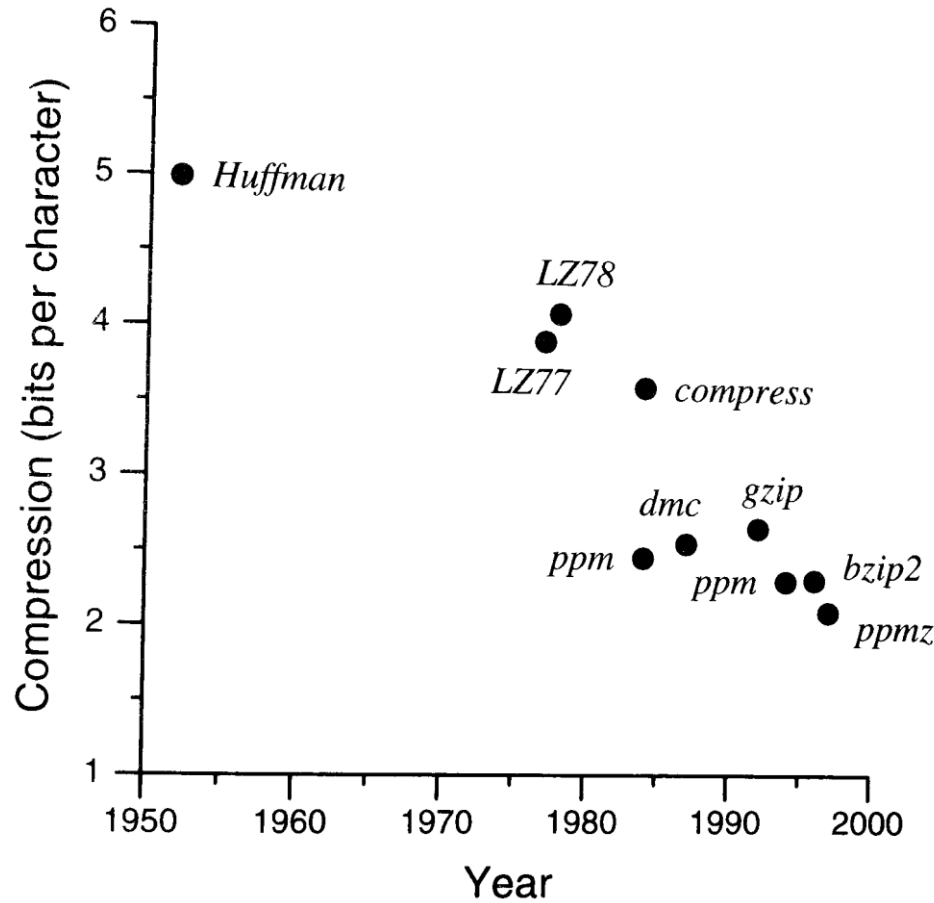
.....  
 phrase 128 = ab  
 phrase 129 = ba  
 phrase 130 = aa  
 phrase 131 = aba  
 phrase 132 = abb  
 phrase 133 = baa  
 phrase 134 = abaa

new phrases added as the string is being encoded

a b a ab ab ba aba abaa

97 98 97 128 128 129 131 134

# Comparison of lossless compression methods



- Lossless compression
  - G3, G4, JBIG (fax)
  - GIF, PNG (simple graphics)
- Lossy compression
  - JPEG (all kind of images)
- BMP, RAW (sensor output), DNG (Digital Negative), etc.
- Tagged Image File Format (image container)
  - TIFF
- International Image Interoperability Framework
  - IIIF

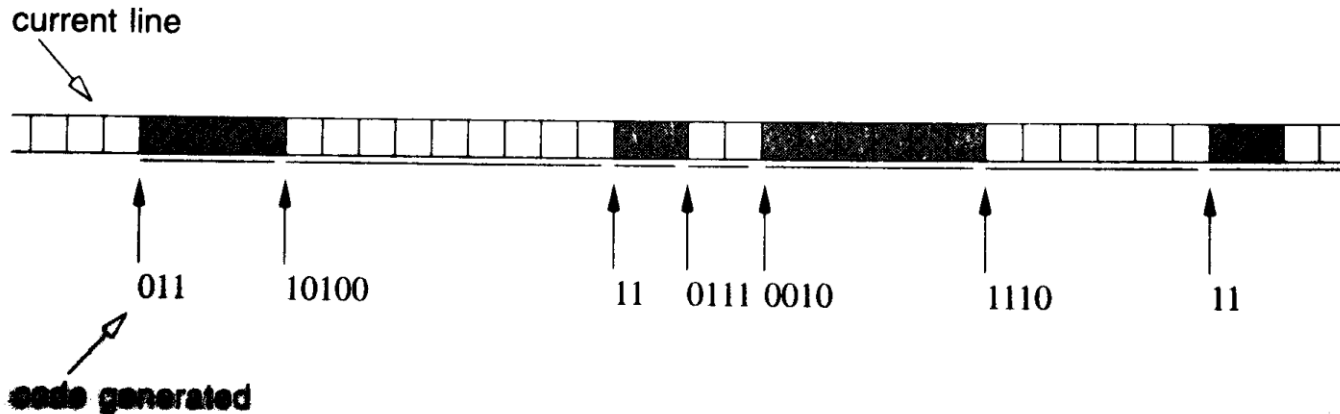
# Lossless compression

## G3, G4, JBIG

- CCITT standard (since late seventies) for fax
  - Comite' Consultatif Internationale de Telegraphie et de Telephonie, part of ITU – International Telecommunications Union
- Specifies resolution
  - 200 x 100 dpi (standard) or 200 x 200 dpi (high resolution)
- Basically bi-level documents (black and white), even if G4 includes also provisions for optional greyscale and color images
- A one-page A4 document contains 1728x1188 pixels (bits), which is about 2 MB of data (too much to be sent over telephone lines, especially at that time)
- G3 specifies two coding (compression) methods.
  - One-dimensional (each line treated separately)
  - Two-dimensional (called READ, exploits coherence between successive scan lines)
- G4 and JBIG are more recent versions of the standard, which allow a much better compression

It is basically a Huffman coding, with pre-set probabilities of the different “run lengths”, i.e. the number of consecutive pixels either black or white

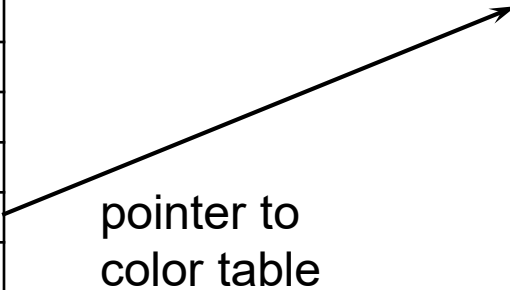
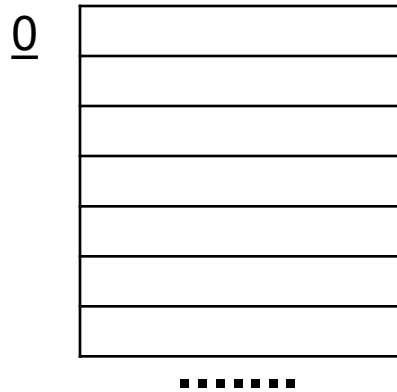
code table run length	color of run	
	white	black
0	00110101	0000110111
1	000111	010
2	0111	11
3	1000	10
4	1011	011
5	1100	0011
6	1110	0010
7	1111	00011
8	10011	000101
9	10100	000100
...	...	...



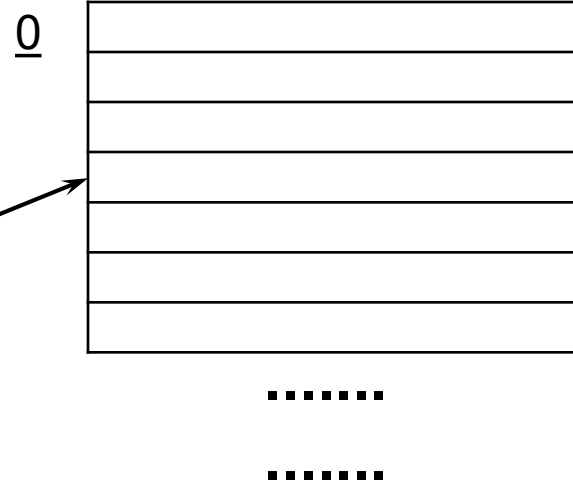
- Lossless compression
  - G3, G4, JBIG (fax)
  - GIF, PNG (simple graphics)
- Lossy compression
  - JPEG (all kind of images)
- BMP, RAW (sensor output), DNG (Digital Negative), etc.
- Tagged Image File Format (image container)
  - TIFF
- International Image Interoperability Framework
  - IIIF

# Pixel representation in GIF

image - 8 bits/pixel  
sequence of rows



color table  
24-36-48 bits



# GIF and PNG

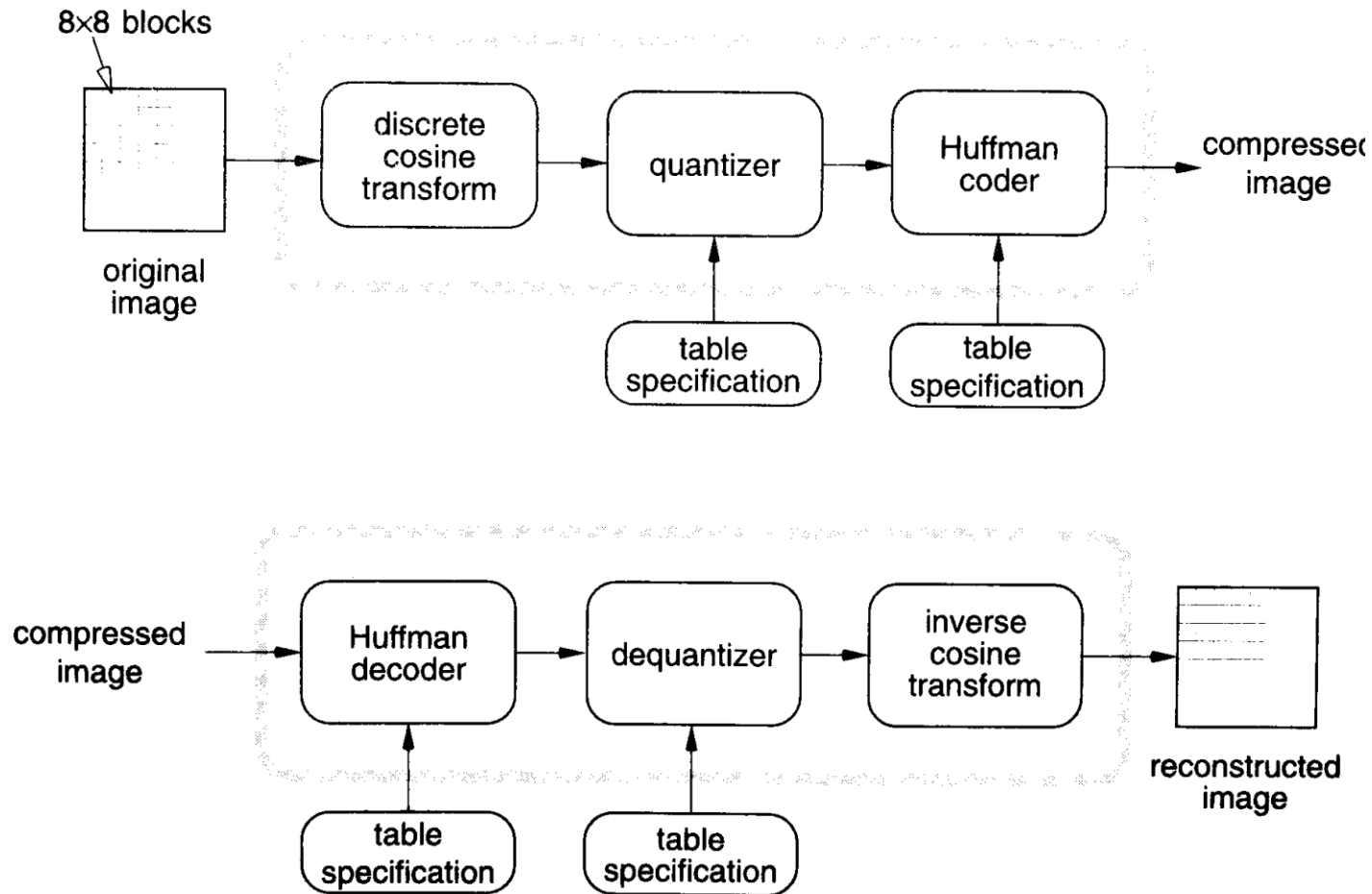
- GIF – Graphics Interchange Format, is probably the most used “lossless” compression format for images (late eighties)
- Each file may contain several images (it supports animation)
- In an image, each pixel is represented by 8 bits (or less), and the value is an index in a color table, which can be included in the file (if not included, a standard color table is used)
- The color table has 256 entries, therefore a GIF image can have a “palette” of at most 256 colors (which is much less than the colors actually in the picture)
- The pixel index values are compressed using the LZW (zip) method
- The LZW coded information is divided in blocks, preceded by a header with a byte count, so it is possible to skip over images without decompressing them
- PNG (Portable Network Graphics) is essentially the same, and was defined some years later to avoid the use of the “proprietary” LZW compression algorithm
  - PNG uses “public domain” *gzip* or *deflate* methods
  - It incorporates also several improvements over GIF



- Lossless compression
  - G3, G4, JBIG
  - GIF, PNG
- Lossy compression
  - JPEG
- BMP, RAW (sensor output), DNG (Digital Negative), etc.
- Tagged Image File Format (image container)
  - TIFF
- International Image Interoperability Framework
  - IIIF

- For grayscale and color images, lossless compression still results in “too many bits”
- Lossy compression methods take advantage from the fact that the human eye is less sensitive to small greyscale or color variation in an image
- JPEG - Joint Photographic Experts Group and Joint Binary Image Group, part of CCITT and ISO
- JPEG can compress down to about one bit per pixel (starting with 8-48 bits per pixel) still having excellent image quality
  - Not very good for fax-like images
  - Not very good for sharp edges and sharp changes in color
- The encoding and decoding process is done on an 8x8 block of pixels (separately for each color component)

# JPEG encoding and decoding



pixel values

$$\begin{bmatrix} 154 & 123 & 123 & 123 & 123 & 123 & 123 & 136 \\ 192 & 180 & 136 & 154 & 154 & 154 & 136 & 110 \\ 254 & 198 & 154 & 154 & 180 & 154 & 123 & 123 \\ 239 & 180 & 136 & 180 & 180 & 166 & 123 & 123 \\ 180 & 154 & 136 & 167 & 166 & 149 & 136 & 136 \\ 128 & 136 & 123 & 136 & 154 & 180 & 198 & 154 \\ 123 & 105 & 110 & 149 & 136 & 136 & 180 & 166 \\ 110 & 136 & 123 & 123 & 123 & 136 & 154 & 136 \end{bmatrix}$$

DCT coefficients

$$\begin{bmatrix} 162.3 & 40.6 & 20.0 & 72.3 & 30.3 & 12.5 & -19.7 & -11.5 \\ 30.5 & 108.4 & 10.5 & 32.3 & 27.7 & -15.5 & 18.4 & -2.0 \\ -94.1 & -60.1 & 12.3 & -43.4 & -31.3 & 6.1 & -3.3 & 7.1 \\ -38.6 & -83.4 & -5.4 & -22.2 & -13.5 & 15.5 & -1.3 & 3.5 \\ -31.3 & 17.9 & -5.5 & -12.4 & 14.3 & -6.0 & 11.5 & -6.0 \\ -0.9 & -11.8 & 12.8 & 0.2 & 28.1 & 12.6 & 8.4 & 2.9 \\ 4.6 & -2.4 & 12.2 & 6.6 & -18.7 & -12.8 & 7.7 & 12.0 \\ -10.0 & 11.2 & 7.8 & -16.3 & 21.5 & 0.0 & 5.9 & 10.7 \end{bmatrix}$$


Discrete Cosine Transform

# The “lossy step”

$$\begin{bmatrix} 162.3 & 40.6 & 20.0 & 72.3 & 30.3 & 12.5 & -19.7 & -11.5 \\ 30.5 & 108.4 & 10.5 & 32.3 & 27.7 & -15.5 & 18.4 & -2.0 \\ -94.1 & -60.1 & 12.3 & -43.4 & -31.3 & 6.1 & -3.3 & 7.1 \\ -38.6 & -83.4 & -5.4 & -22.2 & -13.5 & 15.5 & -1.3 & 3.5 \\ -31.3 & 17.9 & -5.5 & -12.4 & 14.3 & -6.0 & 11.5 & -6.0 \\ -0.9 & -11.8 & 12.8 & 0.2 & 28.1 & 12.6 & 8.4 & 2.9 \\ 4.6 & -2.4 & 12.2 & 6.6 & -18.7 & -12.8 & 7.7 & 12.0 \\ -10.0 & 11.2 & 7.8 & -16.3 & 21.5 & 0.0 & 5.9 & 10.7 \end{bmatrix}$$

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

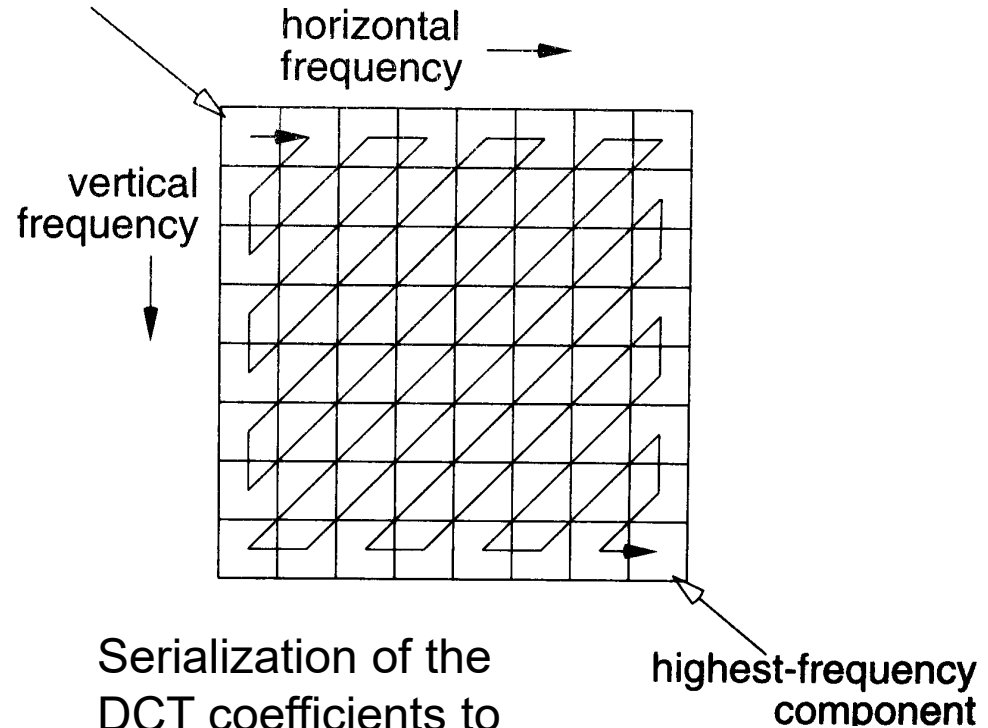
$$\begin{bmatrix} 10 & 4 & 2 & 5 & 1 & 0 & 0 & 0 \\ 3 & 9 & 1 & 2 & 1 & 0 & 0 & 0 \\ -7 & -5 & 1 & -2 & -1 & 0 & 0 & 0 \\ -3 & -5 & 0 & -1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

divide DCT coefficients by  $Q_{50}$   
 quantization matrix, round to  
 nearest integer and get this  
 result

$$\begin{bmatrix} 10 & 4 & 2 & 5 & 1 & 0 & 0 & 0 \\ 3 & 9 & 1 & 2 & 1 & 0 & 0 & 0 \\ -7 & -5 & 1 & -2 & -1 & 0 & 0 & 0 \\ -3 & -5 & 0 & -1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

DCT coefficients after quantization  $Q_{50}$   
The DCT coefficients have been divided by the quantization matrix and then rounded to nearest integer

DC component



Serialization of the DCT coefficients to maximize run-lengths of zeros and therefore take advantage of Huffman coding

# JPEG dequantization

$$\begin{bmatrix} 10 & 4 & 2 & 5 & 1 & 0 & 0 & 0 \\ 3 & 9 & 1 & 2 & 1 & 0 & 0 & 0 \\ -7 & -5 & 1 & -2 & -1 & 0 & 0 & 0 \\ -3 & -5 & 0 & -1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 160 & 44 & 20 & 80 & 24 & 0 & 0 & 0 \\ 36 & 108 & 14 & 38 & 26 & 0 & 0 & 0 \\ -98 & -65 & 16 & -48 & -40 & 0 & 0 & 0 \\ -42 & -85 & 0 & -29 & 0 & 0 & 0 & 0 \\ -36 & 22 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

DCT coefficients after  
quantization  $Q_{50}$

In between there is  
the Huffman coding  
and decoding

DCT coefficients  
dequantized

# Inverse DCT

$$\begin{bmatrix} 160 & 44 & 20 & 80 & 24 & 0 & 0 & 0 \\ 36 & 108 & 14 & 38 & 26 & 0 & 0 & 0 \\ -98 & -65 & 16 & -48 & -40 & 0 & 0 & 0 \\ -42 & -85 & 0 & -29 & 0 & 0 & 0 & 0 \\ -36 & 22 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 149 & 134 & 119 & 116 & 121 & 126 & 127 & 128 \\ 204 & 168 & 140 & 144 & 155 & 150 & 135 & 125 \\ 253 & 195 & 155 & 166 & 183 & 165 & 131 & 111 \\ 245 & 185 & 148 & 166 & 184 & 160 & 124 & 107 \\ 188 & 149 & 132 & 155 & 172 & 159 & 141 & 136 \\ 132 & 123 & 125 & 143 & 160 & 166 & 168 & 171 \\ 109 & 119 & 126 & 128 & 139 & 158 & 168 & 166 \\ 111 & 127 & 127 & 114 & 118 & 141 & 147 & 135 \end{bmatrix}$$



Inverse of Discrete Cosine Transform



# Comparison with original values

149	134	119	116	121	126	127	128
204	168	140	144	155	150	135	125
253	195	155	166	183	165	131	111
245	185	148	166	184	160	124	107
188	149	132	155	172	159	141	136
132	123	125	143	160	166	168	171
109	119	126	128	139	158	168	166
111	127	127	114	118	141	147	135

pixel values after Inverse Cosine Transform

154	123	123	123	123	123	123	136
192	180	136	154	154	154	136	110
254	198	154	154	180	154	123	123
239	180	136	180	180	166	123	123
180	154	136	167	166	149	136	136
128	136	123	136	154	180	198	154
123	105	110	149	136	136	180	166
110	136	123	123	123	136	154	136

original pixel values

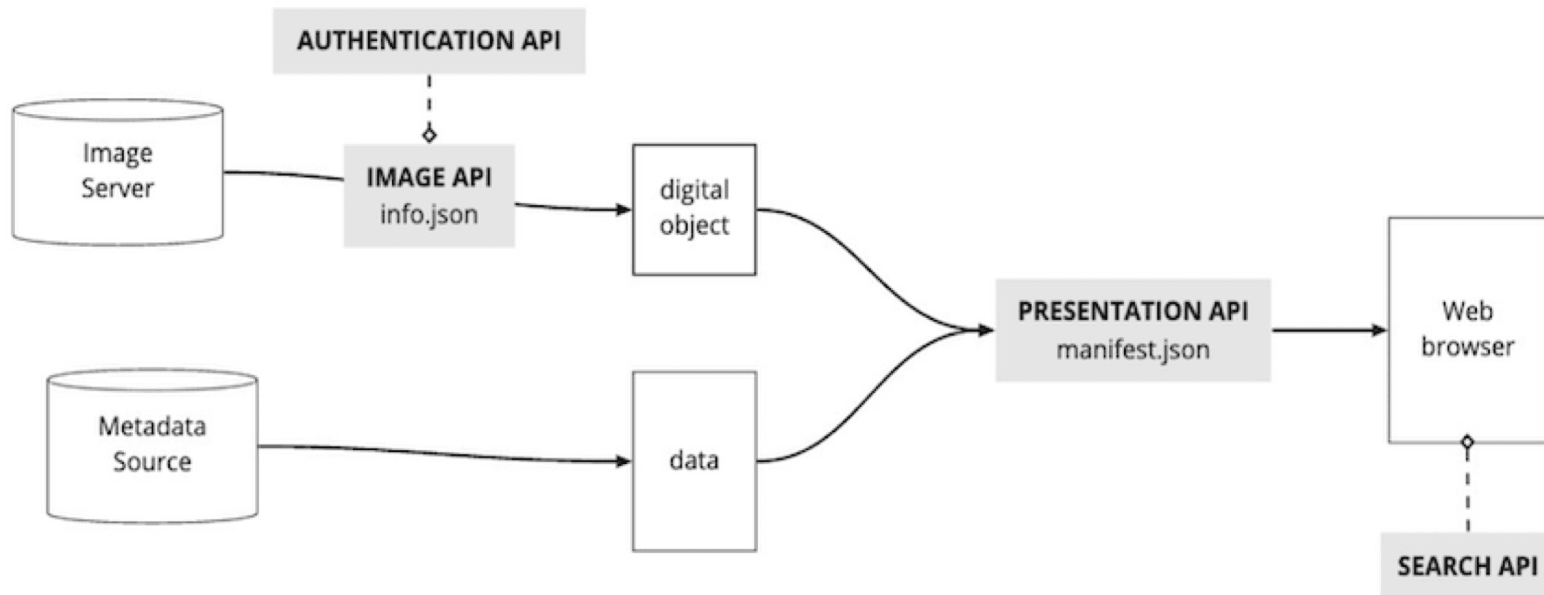
- Lossless compression
  - G3, G4, JBIG
  - GIF, PNG
- Lossy compression
  - JPEG
- BMP, RAW (sensor output), DNG (Digital Negative), etc.
- Tagged Image File Format (image container)
  - TIFF
- International Image Interoperability Framework
  - IIF

- Tagged Image File Format – file format that includes extensive facilities for **descriptive metadata**
  - note that TIFF tags are not the same thing as XML tags
- Owned by Adobe, but public domain (no licensing)
- Large number of options
  - Problems of backward compatibility
  - Problems of interoperability  
(Thousands of Incompatible File Formats 😊)
- Can include (and describe) four types of images
  - bilevel (black and white), greyscale, palette-color, full-color
- Support of different color spaces
- Support of different compression methods
- Much used in digital libraries and archiving

## International Image Interoperability Framework

Delivery

Viewing

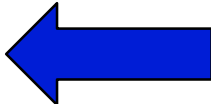


Just an example: <https://digi.vatlib.it/>

# IIF Framework (based on JSON)

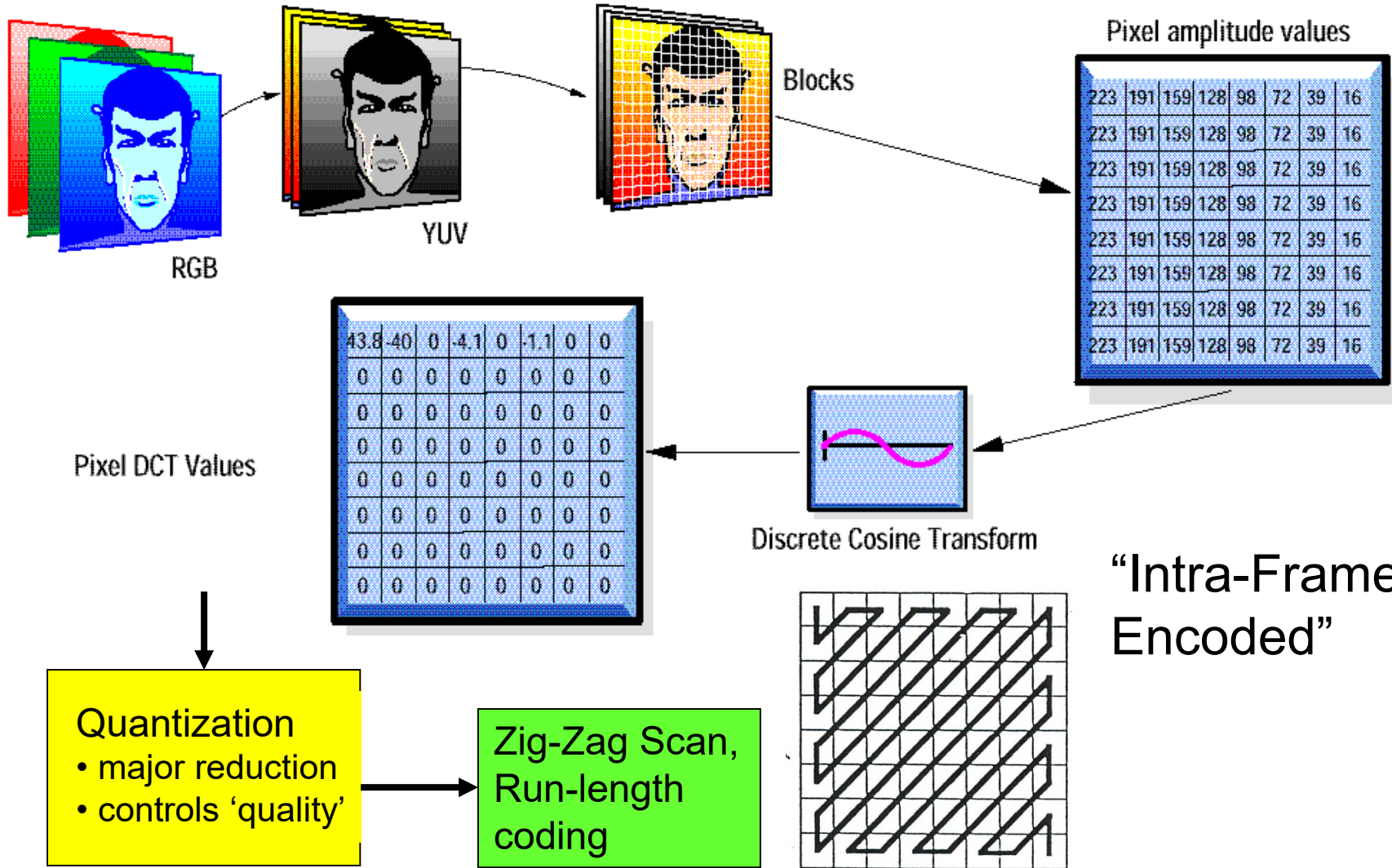
- The Image API defines how image servers deliver image pixels to a viewer
- The Presentation API attaches basic metadata and structure to digital objects, to be shown by IIF-compliant viewers
- The Authentication API defines where or by who your objects can be viewed
- The Search API allows users to search within any text associated with an object
- The Content State API provides a way of linking directly to a particular region and zoom level of a IIF resource
- The Change Discovery API is a tool to describe the new publication of and updates to digital objects

# Representation of information within a computer

- Numbers
- Text (characters and ideograms)
- Documents
- Images
- Video 
- Audio

- Sequence of *frames* (still images) displayed with a given frequency
  - NTSC 30 f/s, PAL 25 f/s, HDTV 60 f/s
- Resolution of each frame depend on quality and video standard
  - 720x480 NTSC, 768x576 PAL, 1920x1080 HDTV, 3840x2160 UltraHD, 4096x2160 4K
- Uncompressed video requires “lots of bits”
  - e.g.  $1920 \times 1080 \times 30 \times 24 = \sim 1,5 \text{ GB/sec}$
- It is possible to obtain very high compression rates
  - Spatial redundancy (within each frame, JPEG-like)
  - Temporal redundancy (across frames)

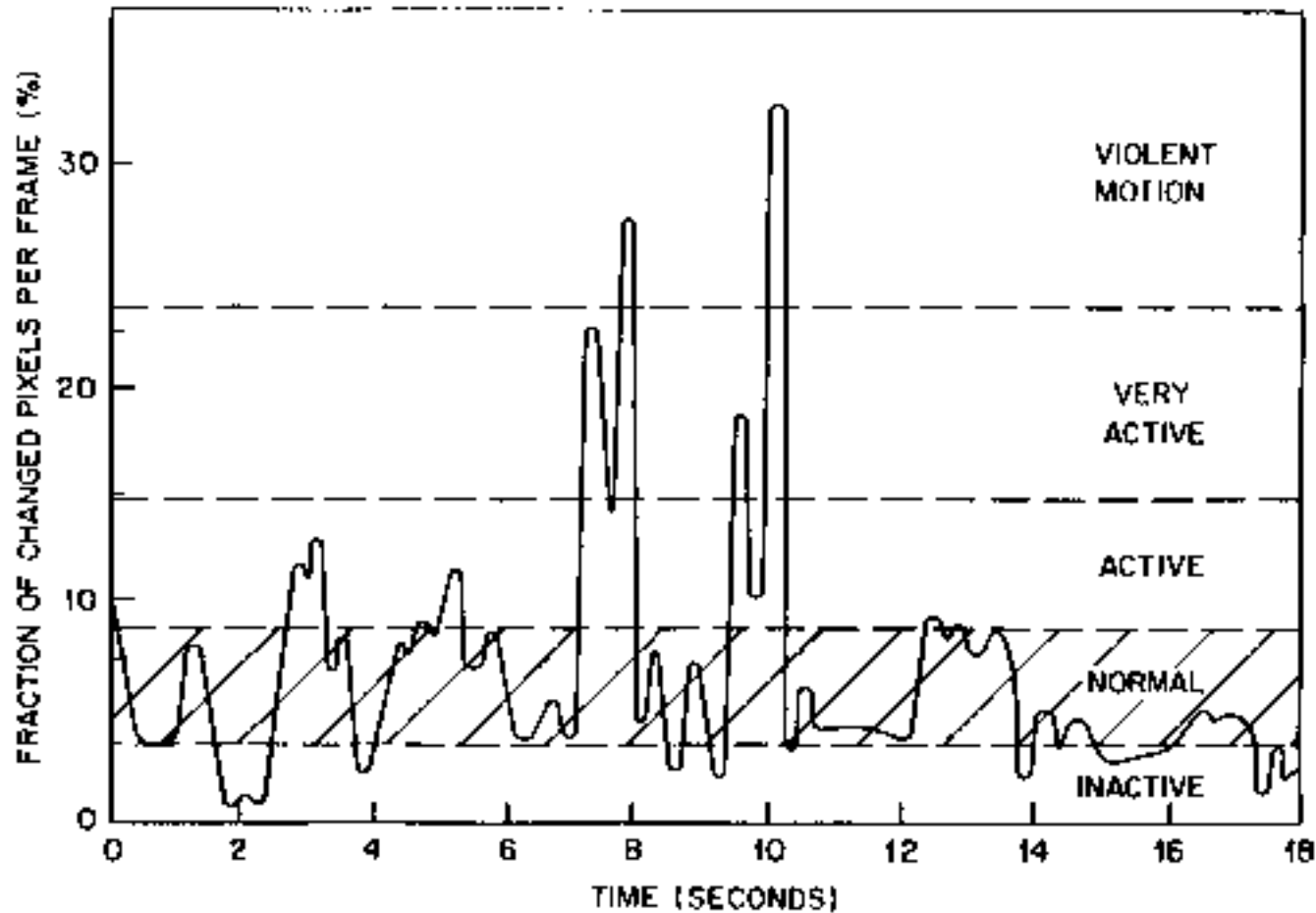
# Spatial Redundancy Reduction (DCT)



“Intra-Frame Encoded”

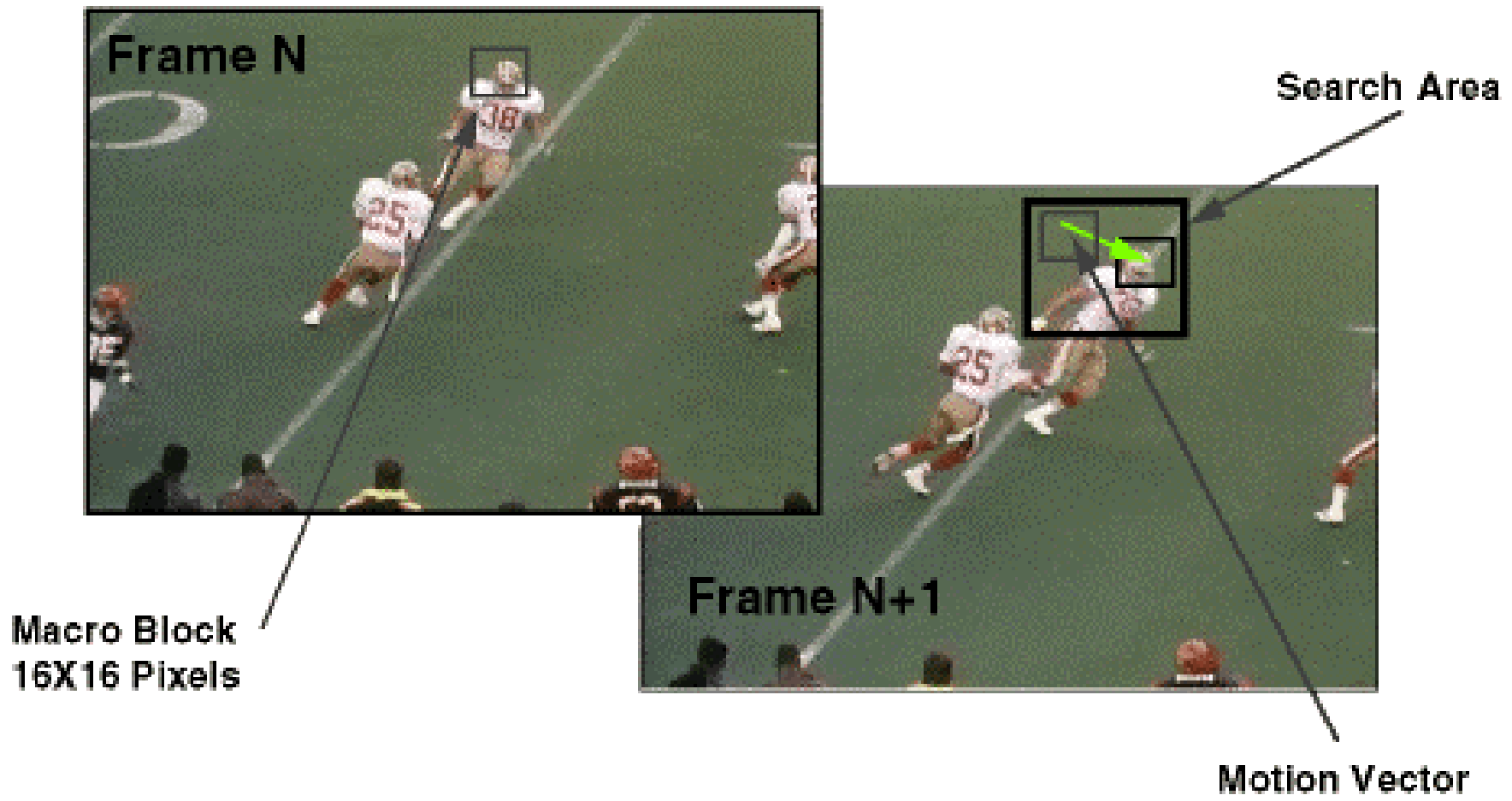


# Temporal Activity



“Talking Head”

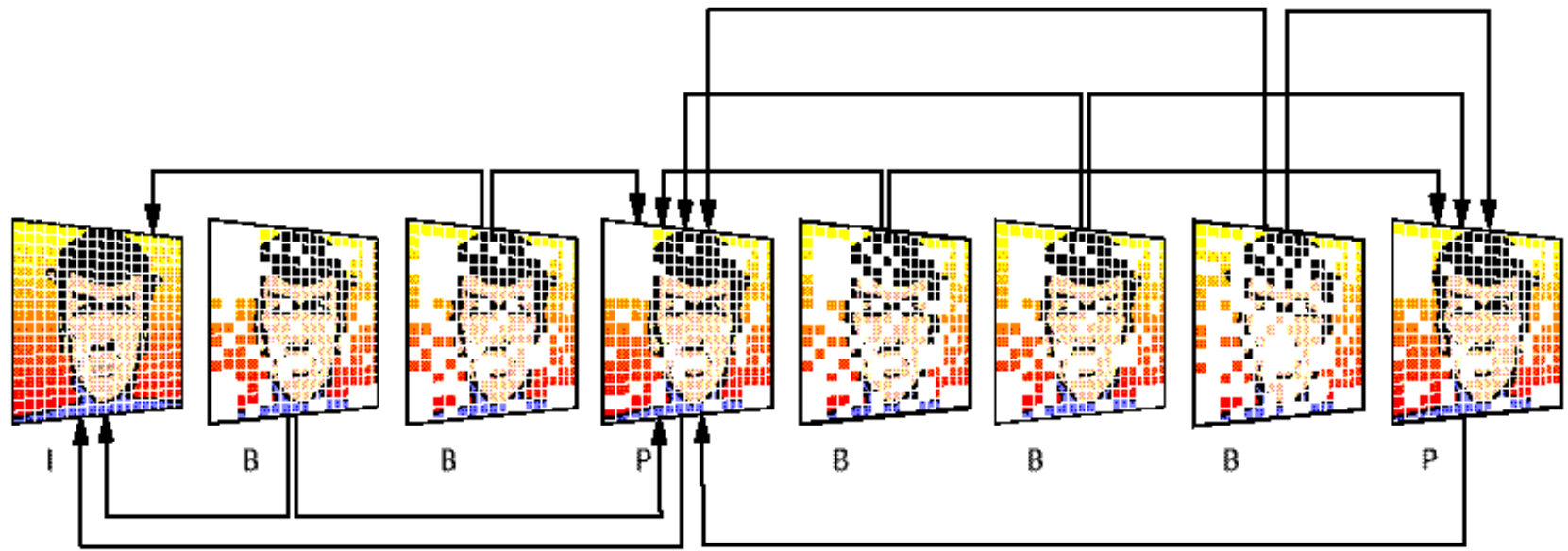
# Temporal Redundancy Reduction (motion vectors)



# Types of frames in compression

- MPEG uses three types of frames for video coding (compressing)
  - I frames: intra-frame coding
    - Coded without reference to other frames
    - Moderate compression (DCT, JPEG-like)
    - Access points for random access
  - P frames: predictive-coded frames
    - Coded with reference to **previous** I or P frames
  - B frames: bi-directionally predictive coded
    - Coded with reference to **previous and future** I and P frames
    - Highest compression rates

# Temporal Redundancy Reduction



- *I* frames are independently encoded
- *P* frames are based on previous *I* and *P* frames
- *B* frames are based on previous and following *I* and *P* frames

## *Type Size Compression*

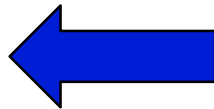
---

I	18	KB	7:1
P	6	KB	20:1
B	2.5	KB	50:1
Avg	4.8	KB	27:1

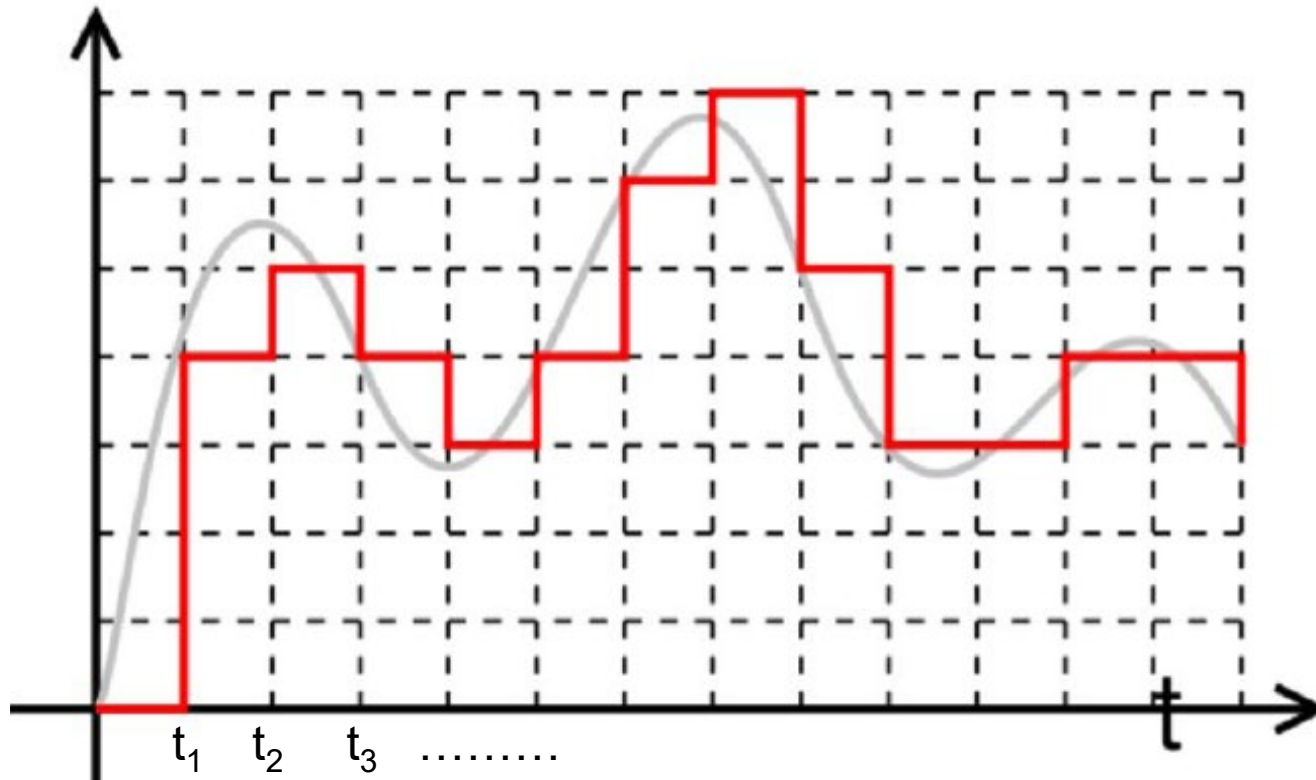
---

# Representation of information within a computer

- Numbers
- Text (characters and ideograms)
- Documents
- Images
- Video
- Audio



# Digitization of audio (analog) signals



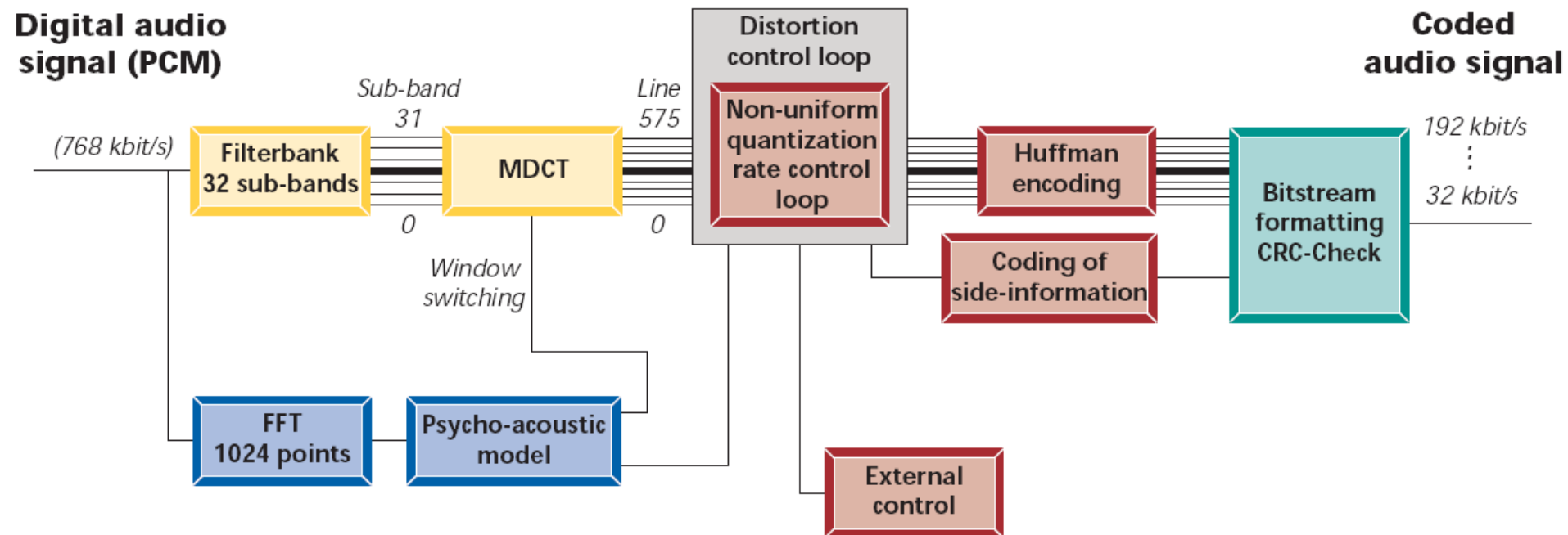
The signal is sampled (the intensity is measured) at fixed time intervals and the “curve” is replaced by a sequence of numbers

# Digitization of audio (analog) signals

- The audio signal is a vibration that hits the human ear
- The (audio) signal can be considered as the sum of a series of sinusoidal signals, each with increasing frequency (Fourier theorem)
- The human ear can hear “vibrations” that go from 15-20 times per second (Hertz) to 20000 Hertz (20 KHz)
- Sampling rate should be at least the double of the highest frequency in the signal that we want to maintain (Shannon theorem)
- Sampling at 44 KHz will “keep” all the frequencies up to 20 KHz
- Transforming the “sequence of samples” back into an audio signal, the human ear will not detect that this signal has “lost” the higher frequencies



- MPEG-1 defines three different schemes (called *layers*) for compressing audio
- All layers support sampling rates of 32, 44.1 and 48 kHz
- Each sample 8-16 bits
- MP3 is MPEG-1 Layer 3



- A muxer (abbreviation of multiplexer) is a “container” file that can contain several video and audio streams, compressed with codecs
  - Common file formats are AVI, DIVx, FLV, MKV, MOV, MP4, OGG, VOB, WMV, 3GPP
- A codec (abbreviation of coder/decoder) is a “system” (a series of algorithms) to compress video and audio streams
  - Common video codecs are HuffYUV, FLV1, HEVC, Mpeg2, xvid4, x264, H264, H265
  - Common audio codecs are AAC, AC3, MP3, PCM, Vorbis