

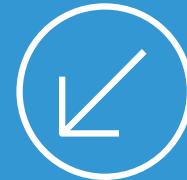


# Quantum Computing PQC – Post Quantum Crypto

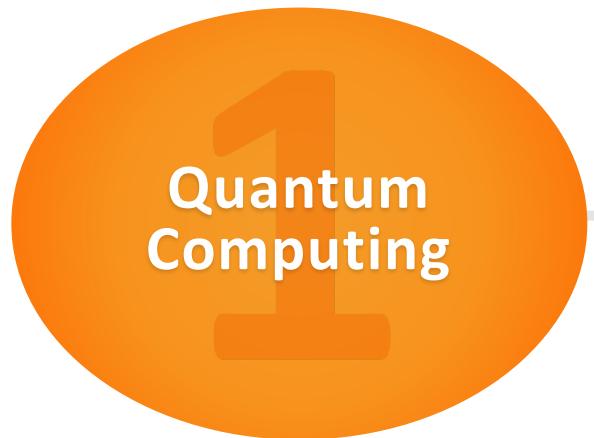
Cristian TOMA

IT | Cybersecurity Master Program – ISM  
Department of Economic Informatics & Cybernetics – DICE | DEIC

[www.ism.ase.ro](http://www.ism.ase.ro) | [www.dice.ase.ro](http://www.dice.ase.ro) | [cristian.toma@ie.ase.ro](mailto:cristian.toma@ie.ase.ro)



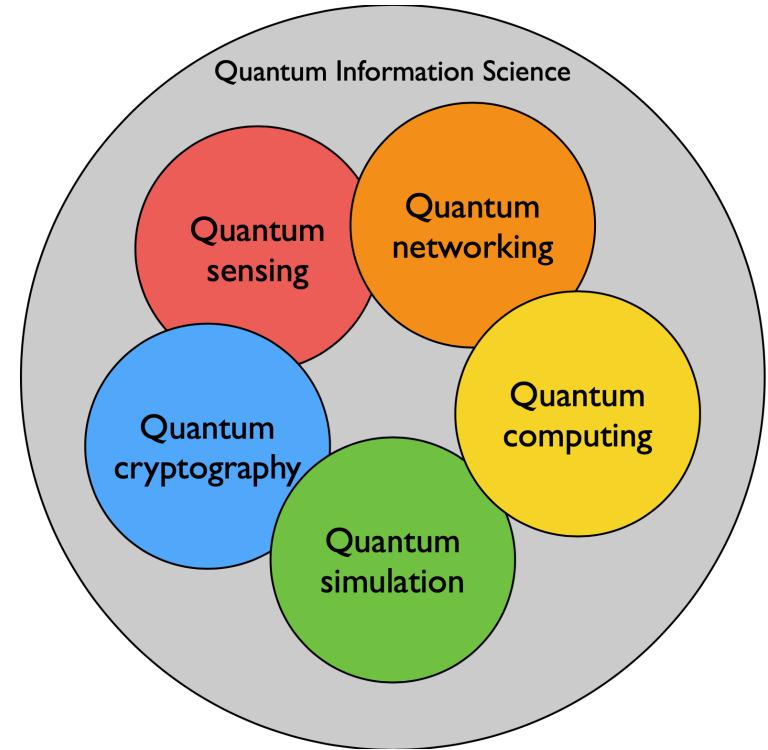
# Agenda for the Presentation



[www.ism.ase.ro](http://www.ism.ase.ro)



[www.dice.ase.ro](http://www.dice.ase.ro)



Math Recap – Linear algebra, Tensor Product, Electronics Digital Circuits and Gates - Classic Bits & Gates, Qubits & Quantum Gates, Superposition, Bell State - Entanglement, Teleportation, Quantum Algorithms, BB84, QKD, Deutsch-Jozsa, Grover, Shor, ...

# Quantum Computing Overview



**Quantum computing is a computing paradigm that uses quantum mechanical properties (superposition, entanglement, interference...) of matter, in order to do calculations**

Quantum is dealing with subatomic particles.

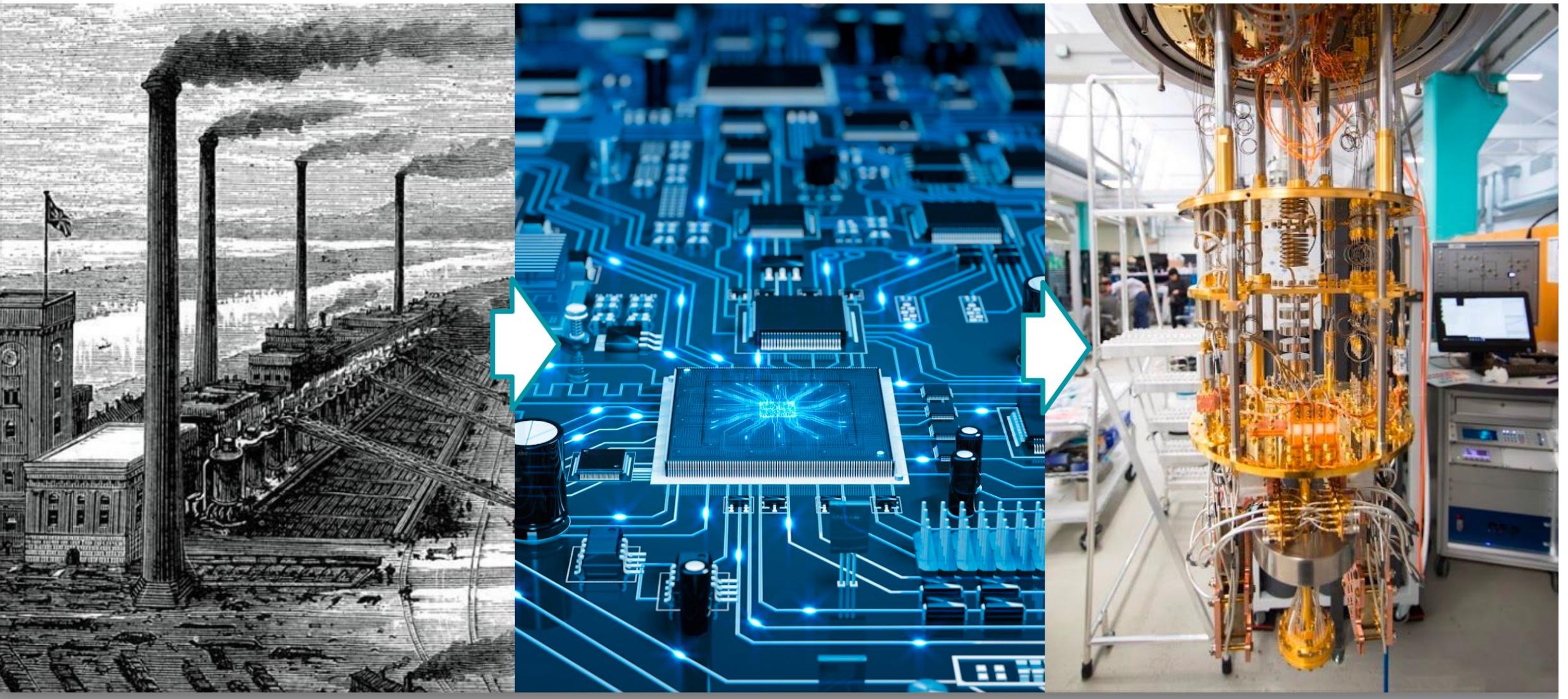
Quantum Computing may resolve problems which in classic approach require exponential time.

## ***Motivation - Why are we doing this lecture?***

- At the leading edge of a new technology, discipline, and industry
- A different paradigm – different programming and algorithms approach
- A great way to challenge yourself to think about computation in a totally new way
- A way to learn “just enough” quantum mechanics and physics
- ***Understanding better PQC – Post Quantum Cryptography in order to forecast the challenges and the risks for current digital signatures used in Blockchain, E-Payment/ Cards/Secure Elements, TLS/SSL, ...***
- A journey!

# Why program a quantum computer?

New power | New opportunity | **Fundamental curiosity**



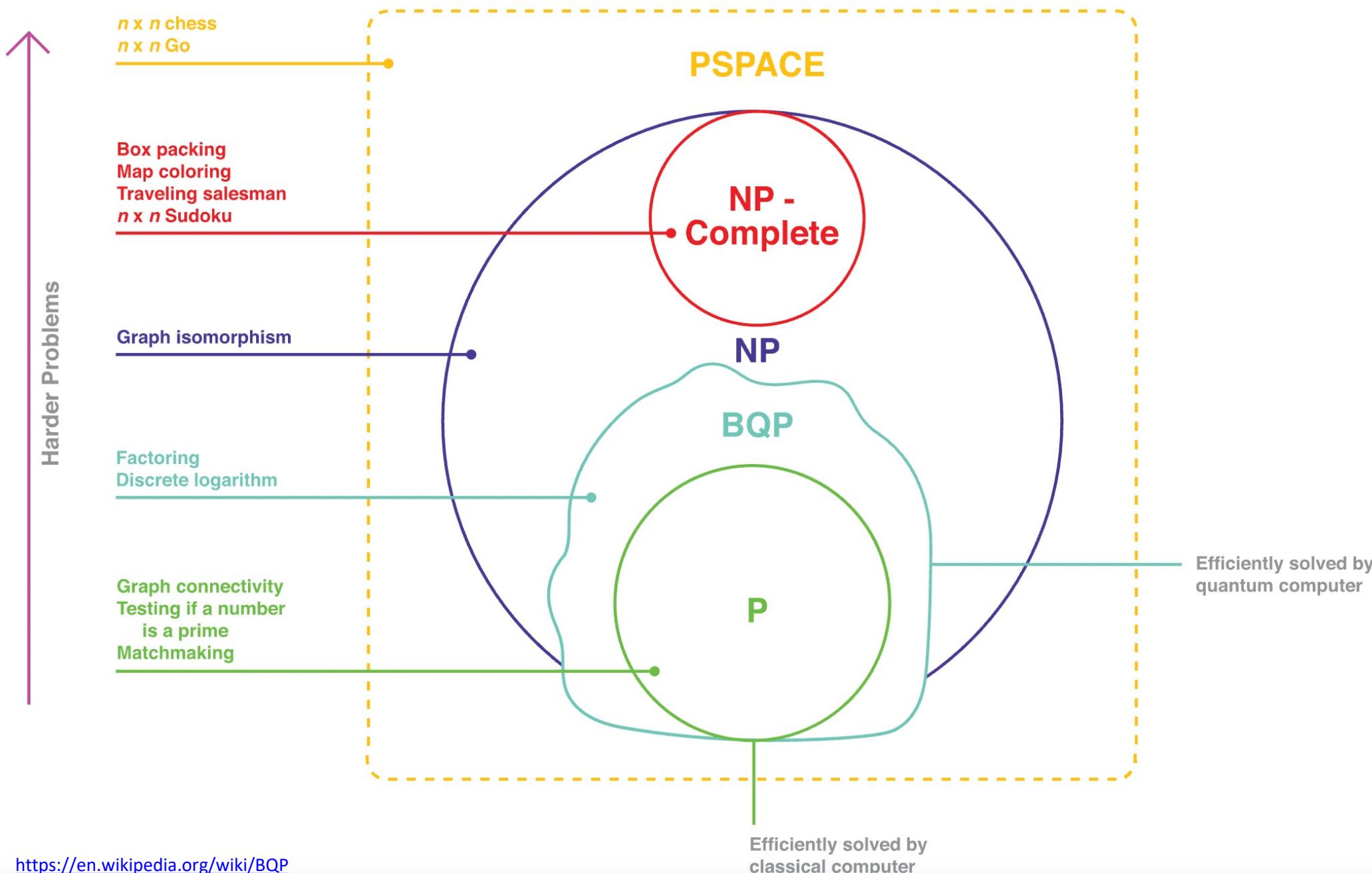
# Quantum computing – Type of Problems

$2^{32} = 4294967296$  rice pcs  
=>  $4294967 \text{ cm}^3$  of rice  
=>  $4.29 \text{ m}^3$  of rice  
=>  
Weight = Density( $798.60 \text{ kg/m}^3$ ) \*  $4 \text{ m}^3$   
=> Weight =  $3.42 \text{ tons}$   
 $\Rightarrow 3.42 * 440 \text{ USD} = 1504.8 \text{ USD}$   
 $\Rightarrow - \text{ one more line } 1504 \text{ USD} * 2^8 = 385024 \text{ USD}$   
 $\Rightarrow - \text{ one more line } 385024 \text{ USD} * 2^8 = 24641536 \text{ USD}$   
 $\Rightarrow - \text{ one more line } 24641536 \text{ USD} * 2^8 = 6'308'233'216 \text{ USD}$

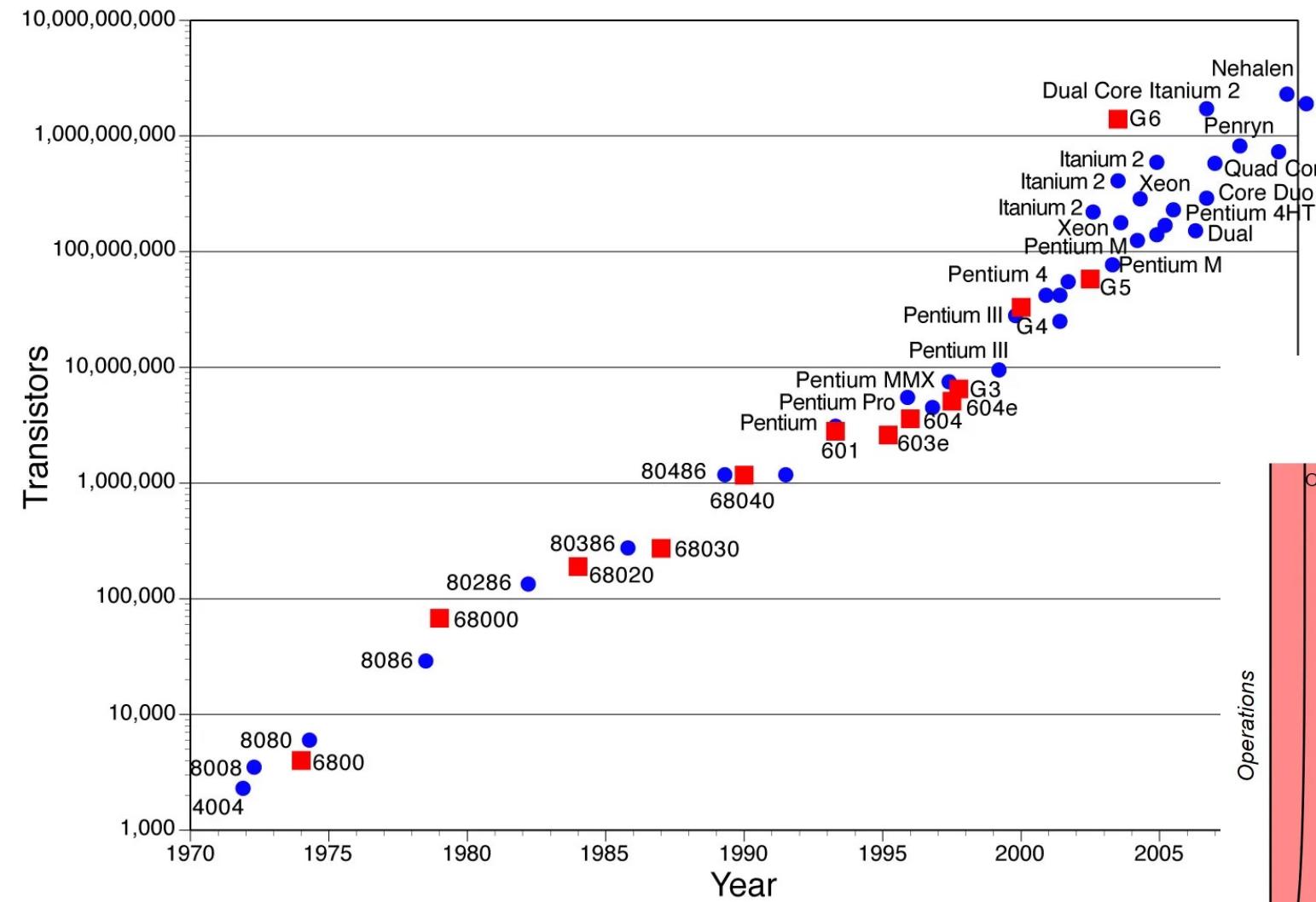
**EASY (Prime Numbers Product):**  
 $4813 * 6257 = 30114941$

# HARD (Prime Factorization): $30114941 = A * B$

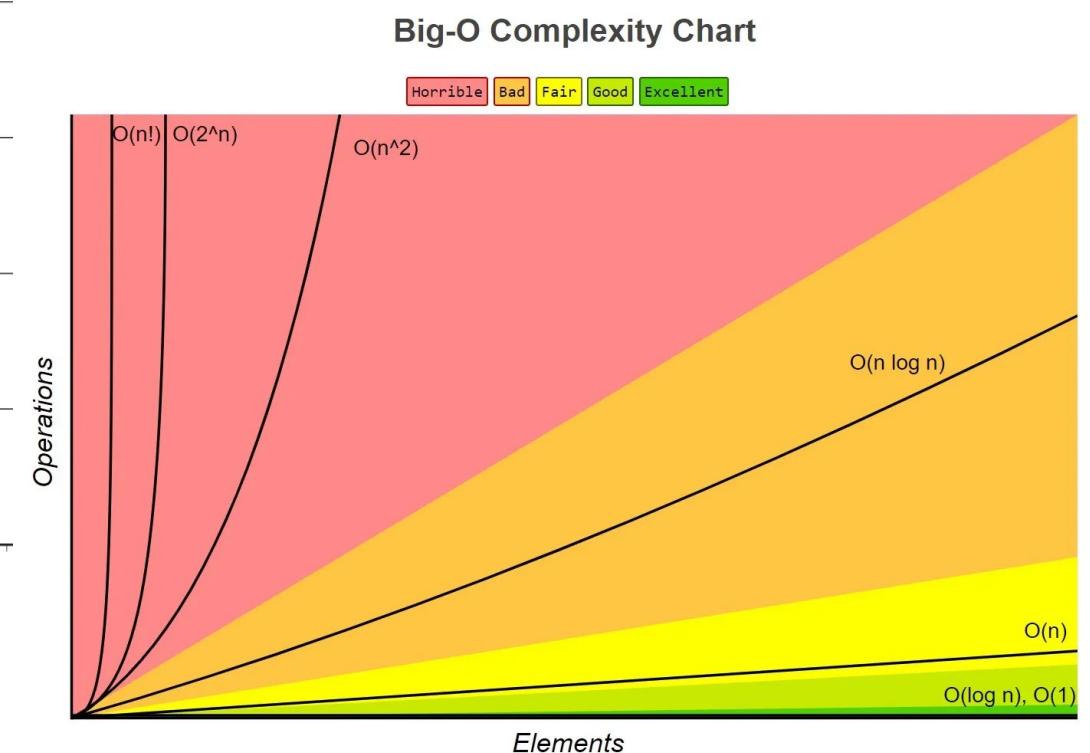
## Quantum computing – Type of Problems



## Standard Computing Complexity – “Big O”



<https://medium.com/@bilalaamir/p-vs-np-problem-in-a-nutshell-dbf08133bec5>



# Why build a quantum computer?

New power | New opportunity | Fundamental curiosity

## Machine Learning

- > Development of new training sets and algorithms
- > Classification and sampling of large data sets



## Supply Chain Optimization

- > Forecast and optimize for future inventory demand
- > NP-hard scheduling and logistics map into quantum applications



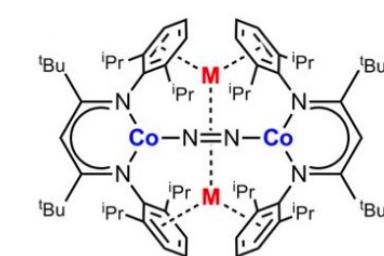
## Robotic Manufacturing

- > Reduce manufacturing time and cost
- > Maps to a Traveling Salesman Problem addressable by quantum constrained optimization



## Computational Materials Science

- > Design of better catalysts for batteries
- > Quantum algorithms for calculating electronic structure



## Alternative Energy Research

- > Efficiently convert atmospheric CO<sub>2</sub> to methanol
- > Powered by existing hybrid quantum-classical algorithms + machine learning



What isn't on here: breaking RSA with Shor's algorithm

# Quantum hardware development is accelerating

Plotted by number of qubits in development

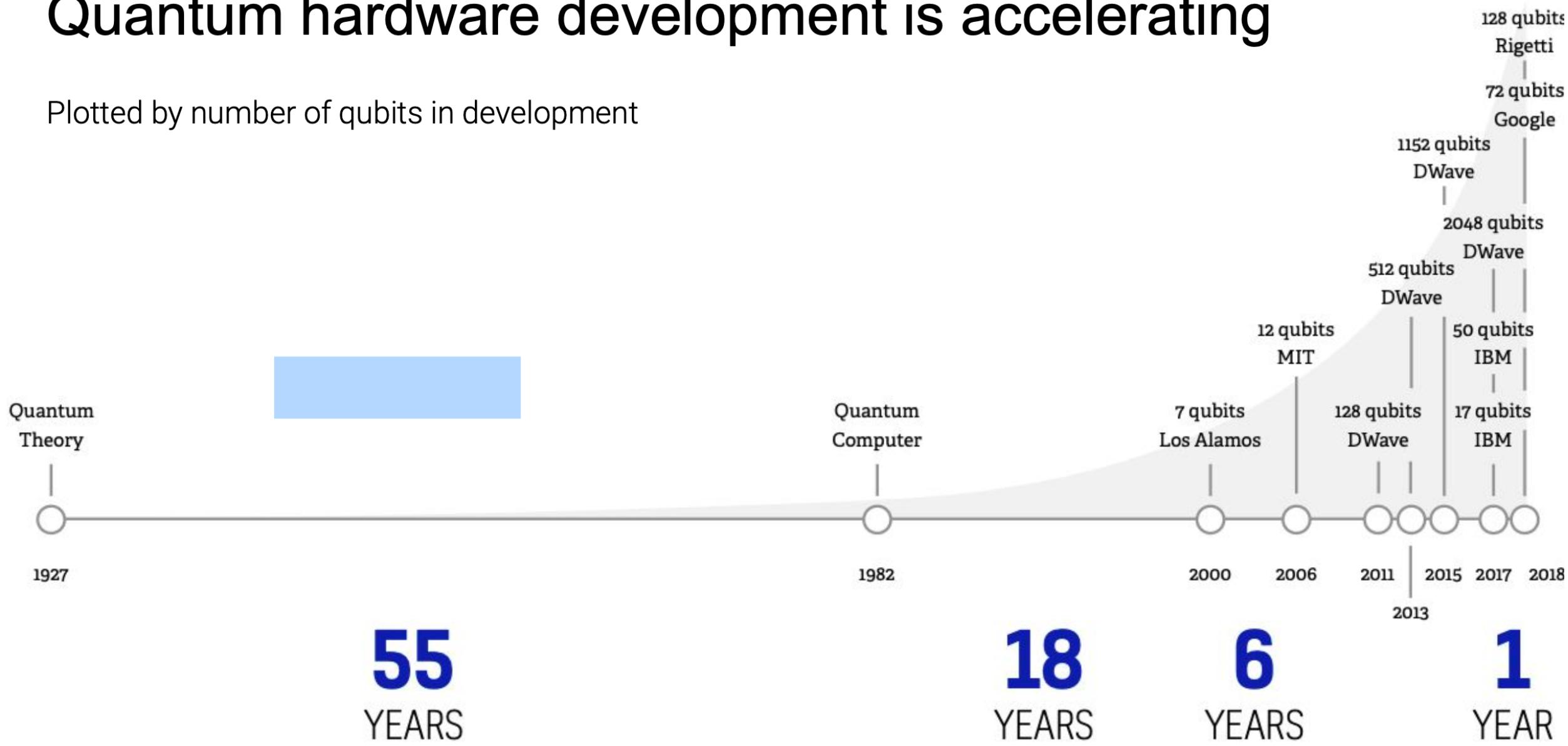
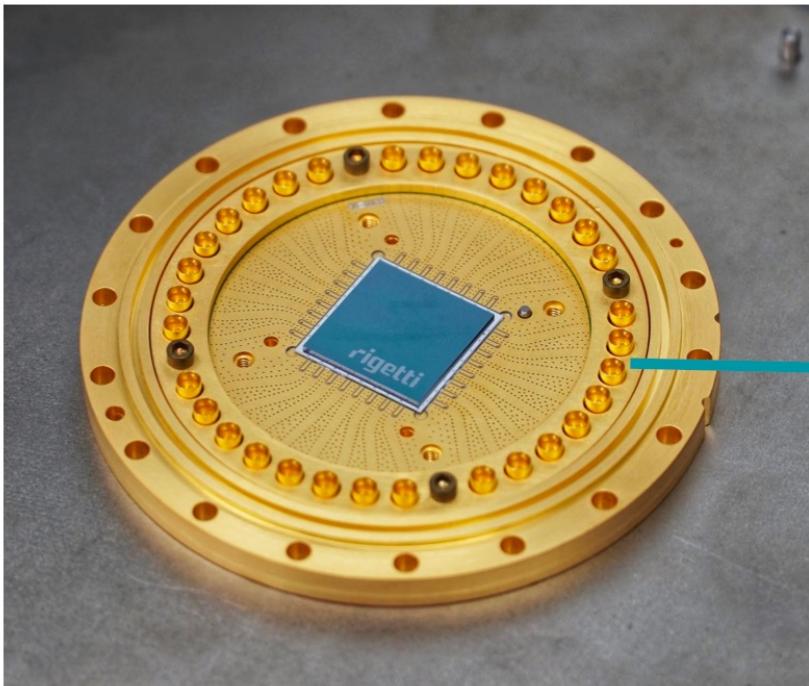


Image: Strangeworks

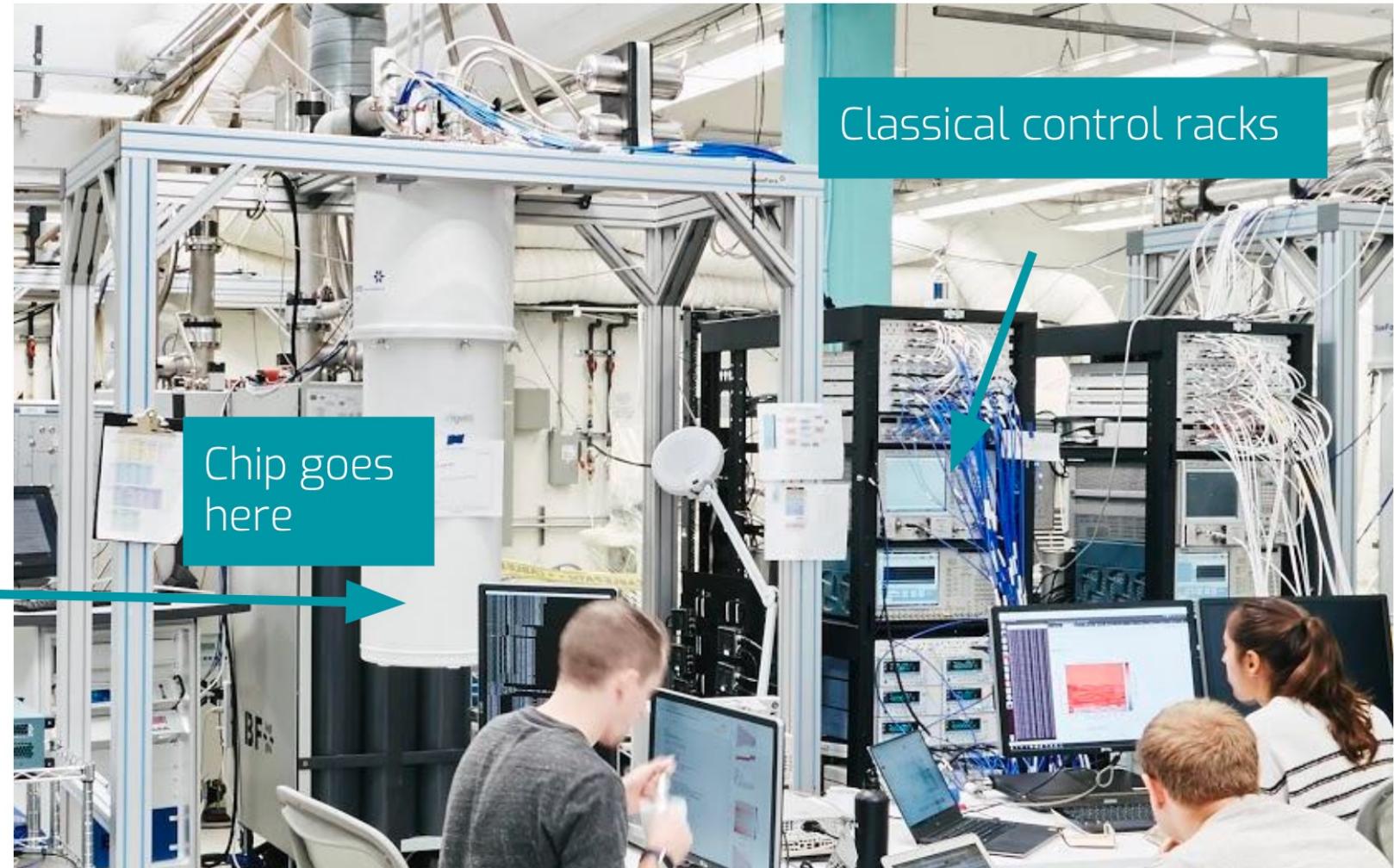
# How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

Quantum computers have  
quantum processor(s) and  
classical processors

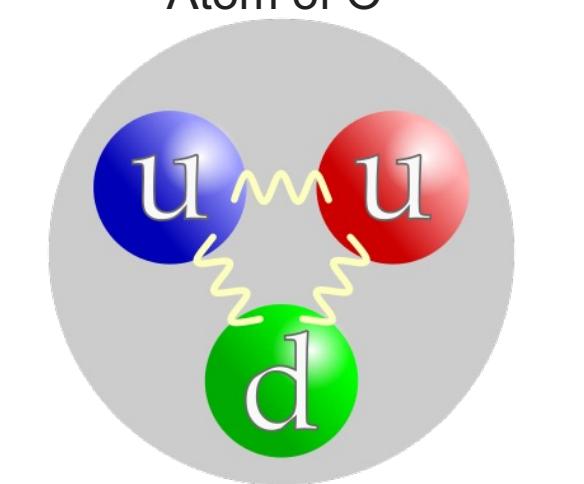
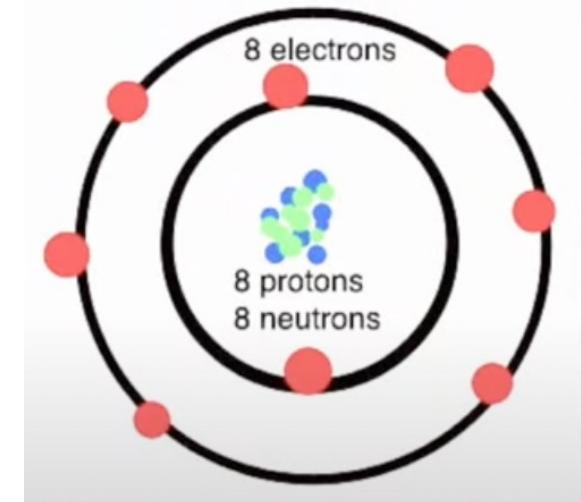
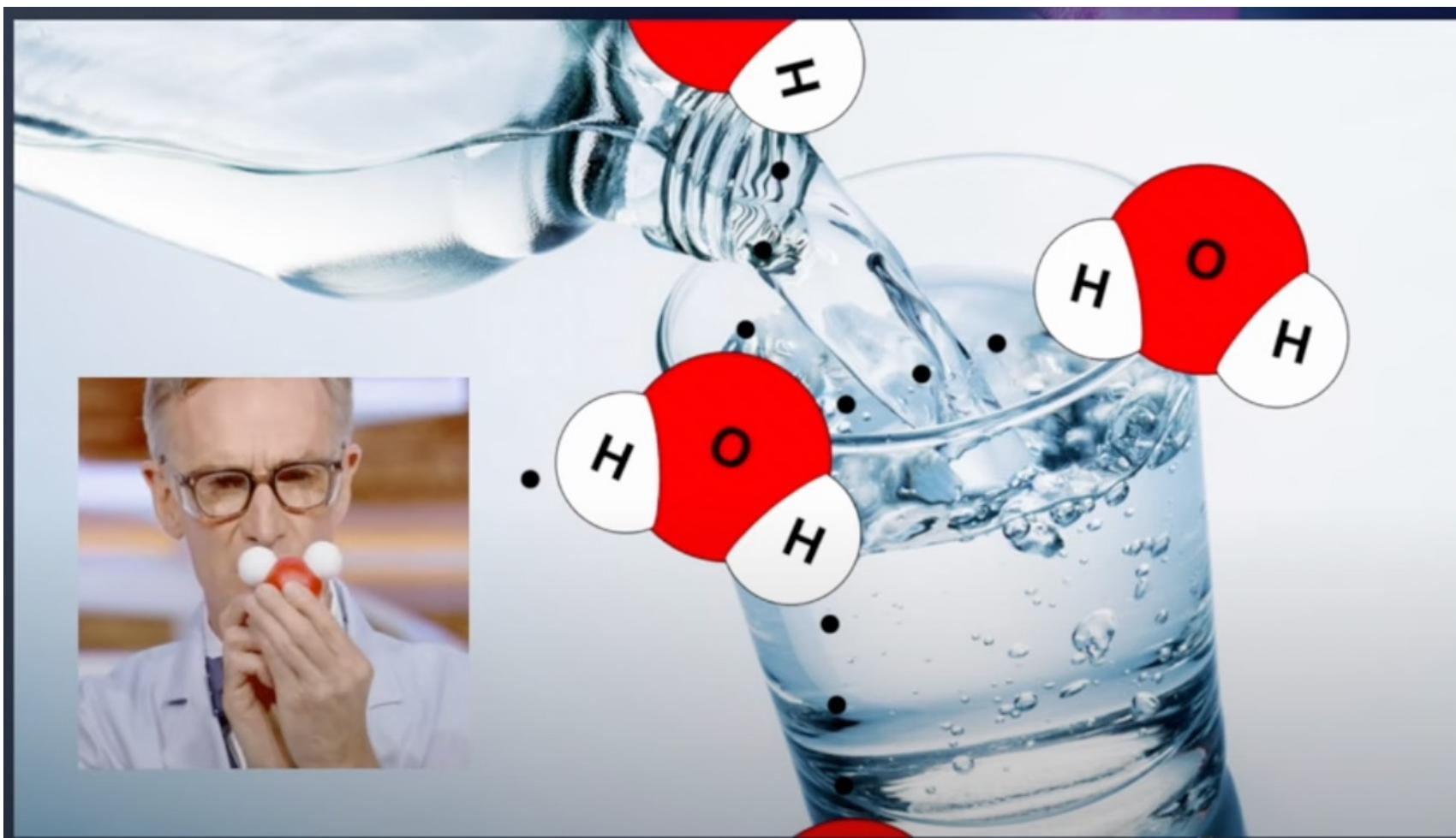


Quantum processor



Full quantum computing system

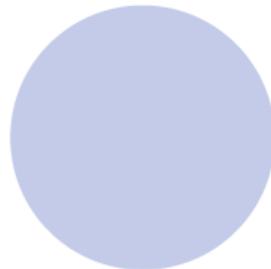
## Quantum computing – Molecule H<sub>2</sub>O



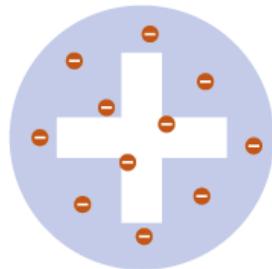
# A HISTORY OF THE ATOM: THEORIES AND MODELS

How have our ideas about atoms changed over the years? This graphic looks at atomic models and how they developed.

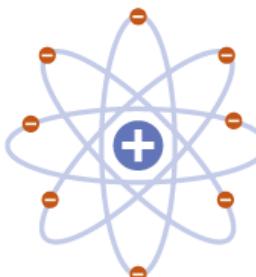
SOLID SPHERE MODEL



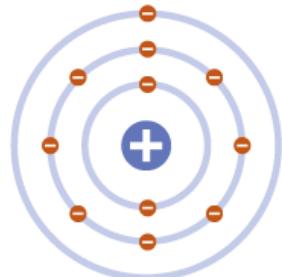
PLUM PUDDING MODEL



NUCLEAR MODEL



PLANETARY MODEL



QUANTUM MODEL



JOHN DALTON



1803

Dalton drew upon the Ancient Greek idea of atoms (the word 'atom' comes from the Greek 'atomos' meaning indivisible). His theory stated that atoms are indivisible, those of a given element are identical, and compounds are combinations of different types of atoms.

- + RECOGNISED ATOMS OF A PARTICULAR ELEMENT DIFFER FROM OTHER ELEMENTS

- ATOMS AREN'T INDIVISIBLE - THEY'RE COMPOSED FROM SUBATOMIC PARTICLES

J.J. THOMSON



1904

Thomson discovered electrons (which he called 'corpuscles') in atoms in 1897, for which he won a Nobel Prize. He subsequently produced the 'plum pudding' model of the atom. It shows the atom as composed of electrons scattered throughout a spherical cloud of positive charge.

- + RECOGNISED ELECTRONS AS COMPONENTS OF ATOMS

- NO NUCLEUS; DIDN'T EXPLAIN LATER EXPERIMENTAL OBSERVATIONS

ERNEST RUTHERFORD



1911

Rutherford fired positively charged alpha particles at a thin sheet of gold foil. Most passed through with little deflection, but some deflected at large angles. This was only possible if the atom was mostly empty space, with the positive charge concentrated in the centre: the nucleus.

- + REALISED POSITIVE CHARGE WAS LOCALISED IN THE NUCLEUS OF AN ATOM

- DID NOT EXPLAIN WHY ELECTRONS REMAIN IN ORBIT AROUND THE NUCLEUS

NIELS BOHR



1913

Bohr modified Rutherford's model of the atom by stating that electrons moved around the nucleus in orbits of fixed sizes and energies. Electron energy in this model was quantised; electrons could not occupy values of energy between the fixed energy levels.

- + PROPOSED STABLE ELECTRON ORBITS; EXPLAINED THE EMISSION SPECTRA OF SOME ELEMENTS

- MOVING ELECTRONS SHOULD EMIT ENERGY AND COLLAPSE INTO THE NUCLEUS; MODEL DID NOT WORK WELL FOR HEAVIER ATOMS

ERWIN SCHRÖDINGER



1926

Schrödinger stated that electrons do not move in set paths around the nucleus, but in waves. It is impossible to know the exact location of the electrons; instead, we have 'clouds of probability' called orbitals, in which we are more likely to find an electron.

- + SHOWS ELECTRONS DON'T MOVE AROUND THE NUCLEUS IN ORBITS, BUT IN CLOUDS WHERE THEIR POSITION IS UNCERTAIN

- + STILL WIDELY ACCEPTED AS THE MOST ACCURATE MODEL OF THE ATOM



© COMPOUND INTEREST 2016 - WWW.COMPOUNDCHEM.COM | Twitter: @compoundchem | Facebook: www.facebook.com/compoundchem

This graphic is shared under a Creative Commons Attribution-NonCommercial-NoDerivatives licence.



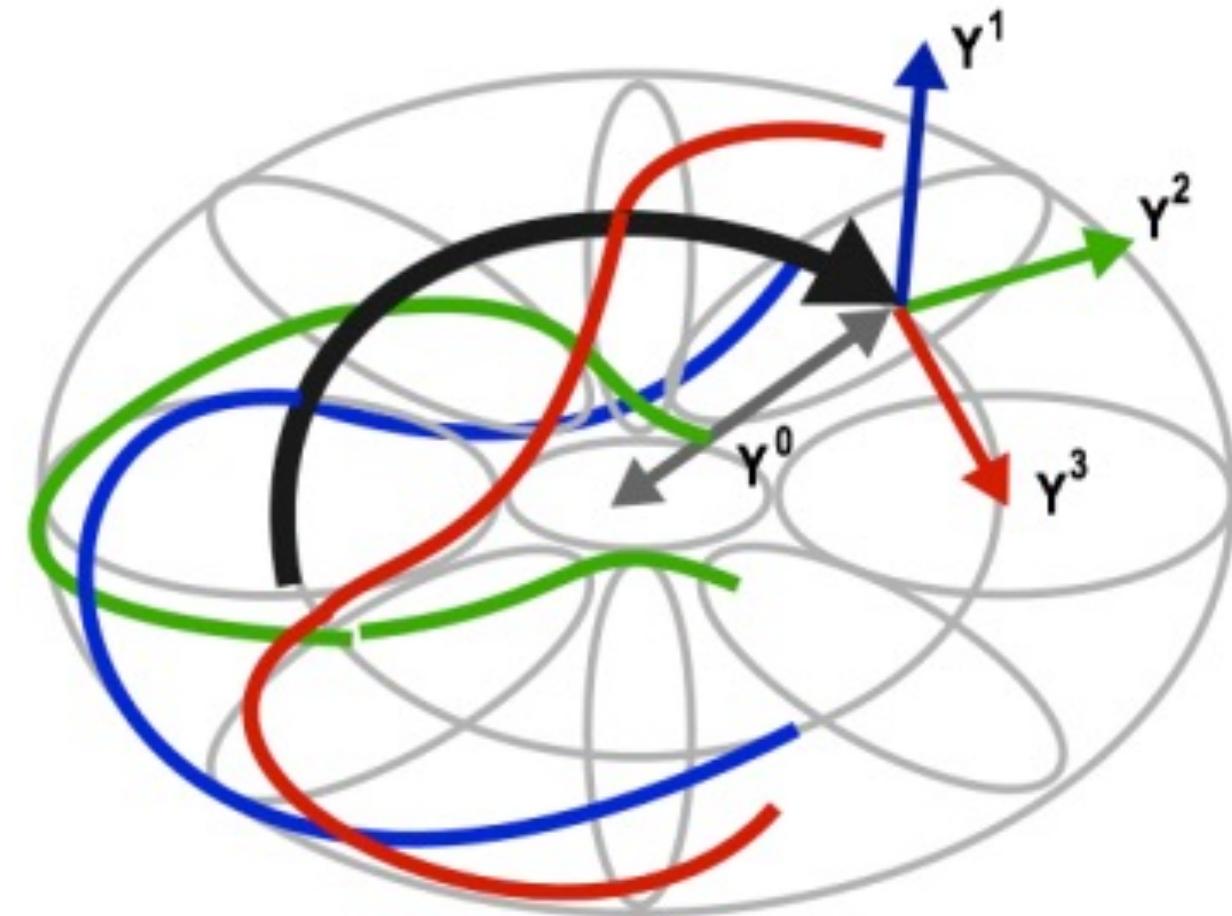
# Atoms new model

## Toroidal Resolution of the Dirac Wave Function

The suitability of the Dirac wave function to represent fermions (spin 1/2 particles) has been well established in Quantum Mechanics.

The Dirac wave function can be resolved using Clifford Algebra as single toroidal structure/s with oscillating "out of phase" wave forms. Refer to the image on the right.

The Subtle Atomics model identifies protons & neutrons as "triple toroidal" structures comprised of meson resonances.



S. Brink., 1st March 2019

## Atoms new model

What might imaginary helium ( $iHe$ ) look like? The mass of  $iH$  is equal to the mass of one electron plus one proton while the mass of helium is equal to the mass of two electrons plus two protons plus two neutrons, or four times that of  $iH$ . But the shape of  $iHe$  is more like a dumbbell than a sphere. Figure 2 shows two protons and two neutrons at the center of opposing electronic envelopes. Due to like charges each electron has trouble violating the others envelope, and as they must maintain nearly the velocity of light, they seem to be everywhere in their own envelope at once. One must imagine these envelopes as three dimensional coming out of the page as well as below, more like an apple eaten down to the core!

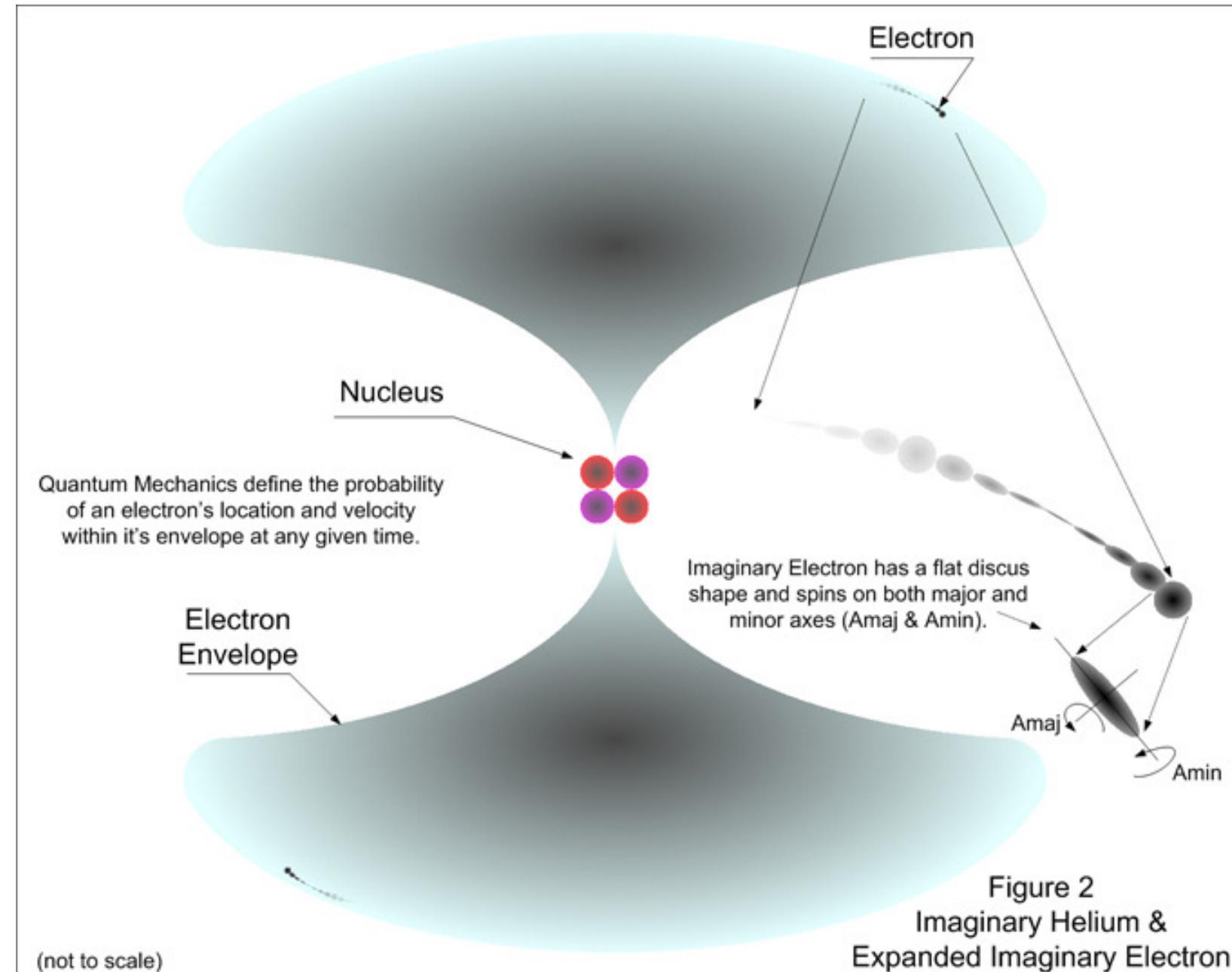
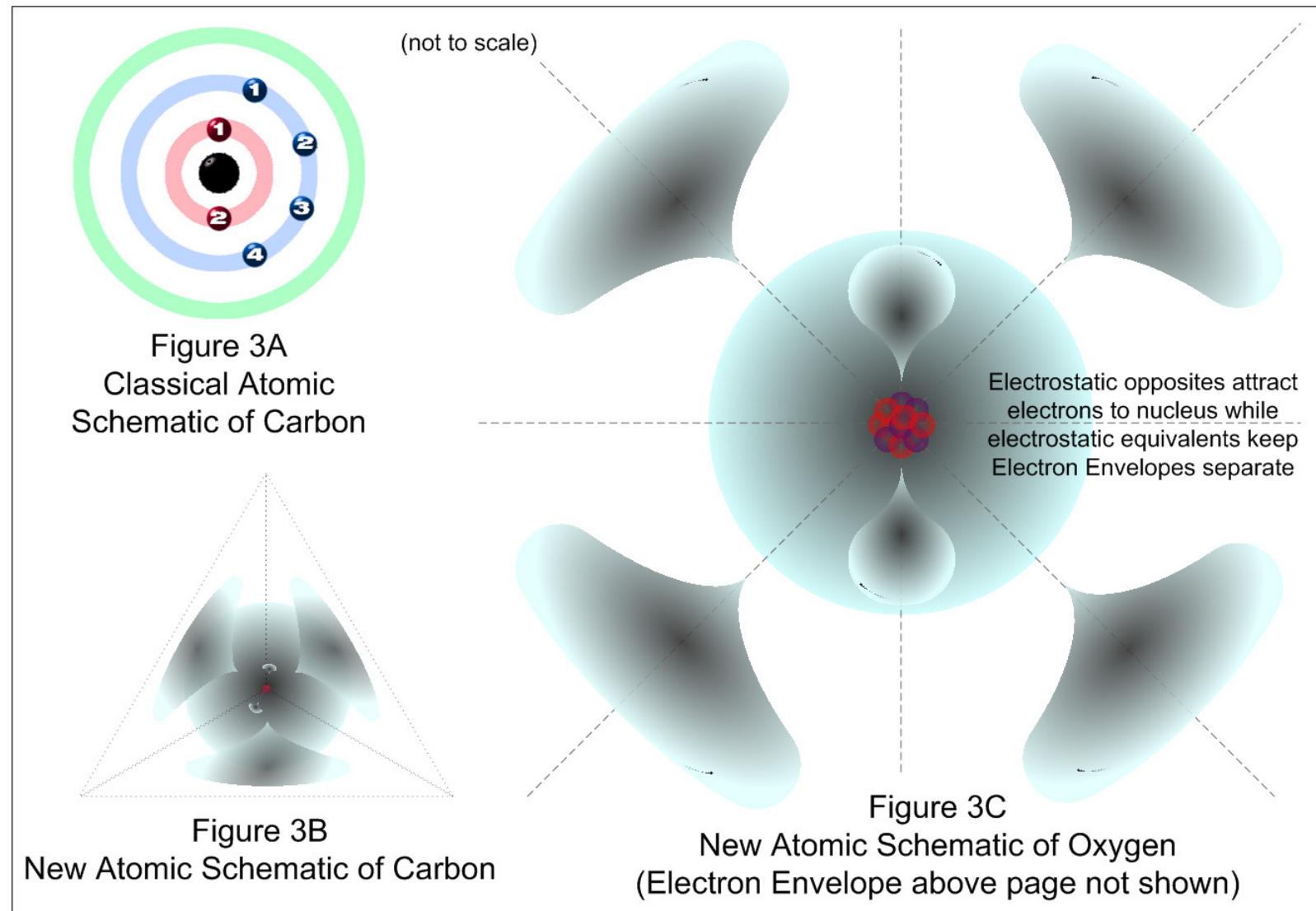


Figure 2  
Imaginary Helium &  
Expanded Imaginary Electron

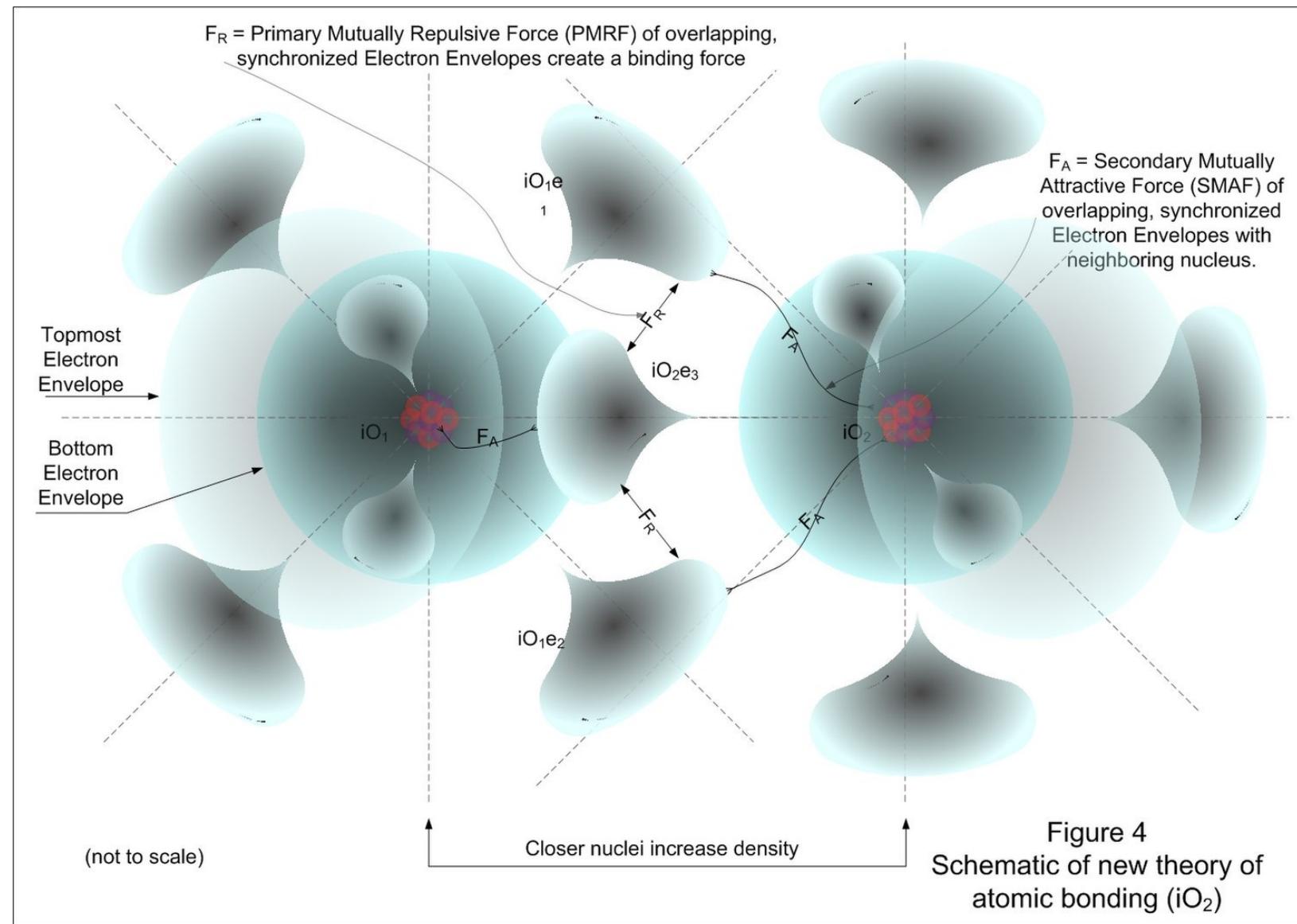
# Atoms new model

Figure 3A shows a typical middle school rendering of the carbon atom. Such a view is inaccurate and makes nuclear electro-mechanics more difficult to comprehend. The imaginary carbon (iC, Figure 3B) and imaginary oxygen (iO, Figure 3C) is more realistic and will lead to a much more intuitive understanding of chemical bonding (the dotted lines are for visual reference only). Performing the "thought experiment" of imagining the push-pull of electromagnetic forces upon these various particles, one can "see" how these subatomic components hold together. The two electrons in the innermost shell of Figure 3C are compelled to "fall" towards the nucleus, repel one another, and maintain near light speed. The four electrons in the outer shell are in a similar predicament, rapidly organization into synchronized equilibrium. This equilibrium is what defines substances and gives rise to their crystallization characteristics and chemical properties. Note that the Figure 3B model of carbon is an equilateral tetrahedron (a triangle in 3 dimensions). Triangles are among the strongest geometric forms, and equilateral triangles are the strongest triangles. Equilateral tetrahedrons are the strongest 3D shapes, and perfect for turning soft carbon or graphite to that hardest of natural substances... Diamonds! The more one learns about Nature, the more one can appreciate the connections from algebra to zoology. Nature can be wonderfully logical!



# Atoms new model

Noting that the space between all components of iO would be orders of magnitude greater than this page can show, Figure 4 shows schematically how two iO could bond. Assume that for whatever reasons that electron  $iO_2e_2$  has come well into the influence of  $iO_1e_1$  and  $iO_1e_2$  as shown. We say they are sharing their shells. Since electrons repel each other electromagnetically atom  $iO_1$  and atom  $iO_2$  are now coupled together by the force labeled  $F_R$ . Additional bonding force ( $F_A$ ) is supplied by the displaced of  $iO_1e_1$  and  $iO_1e_2$  being attracted to  $iO_2$ 's nucleus. This is much like linking your knuckles together and attempting to pull your hands apart. But if electrons repel how could they get into such a configuration to begin with? Either by pressure & heat, or catalytically, or electromagnetically. If the pressure and heat is sufficient, the electron envelopes deform allowing the outer shells to slip amongst one another. This is how the sun and deep earth turn carbon to diamond. If a catalyst has the correct properties, it can pull the electron envelopes apart creating a "hole" for another envelope to slip into, freeing the catalyst to slip away for another reaction. This is how catalytic converters convert automobile emissions to less toxic substances. Electromagnetically, electron envelopes may be pulled apart creating a "hole" for coupling. This is how "buckyballs" are created in an [electric arc](#).



Quantum Mechanics ... are weird, hard to understand, invisible – e.g. Light Duality: particle (photons) and wave (energy)

FILTER #1



FILTER #2



sunlight is

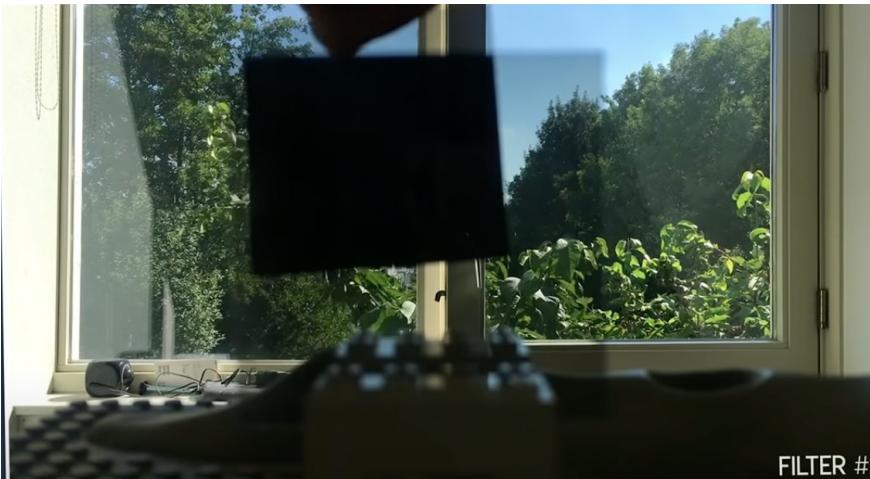
**RANDOMLY POLARISED**  
with waves in  
**ALL DIRECTIONS**



\*



FILTER #1

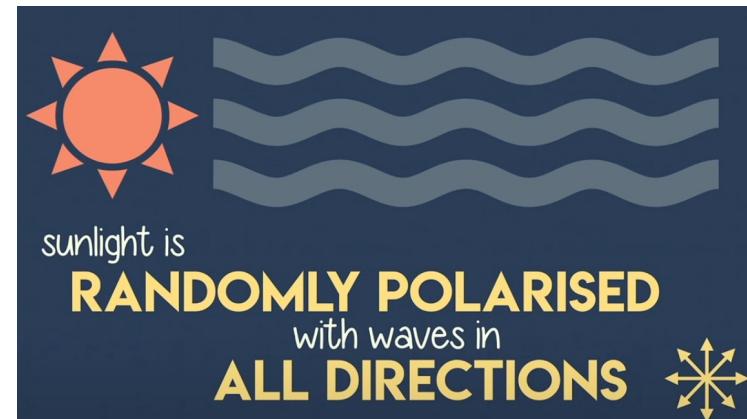
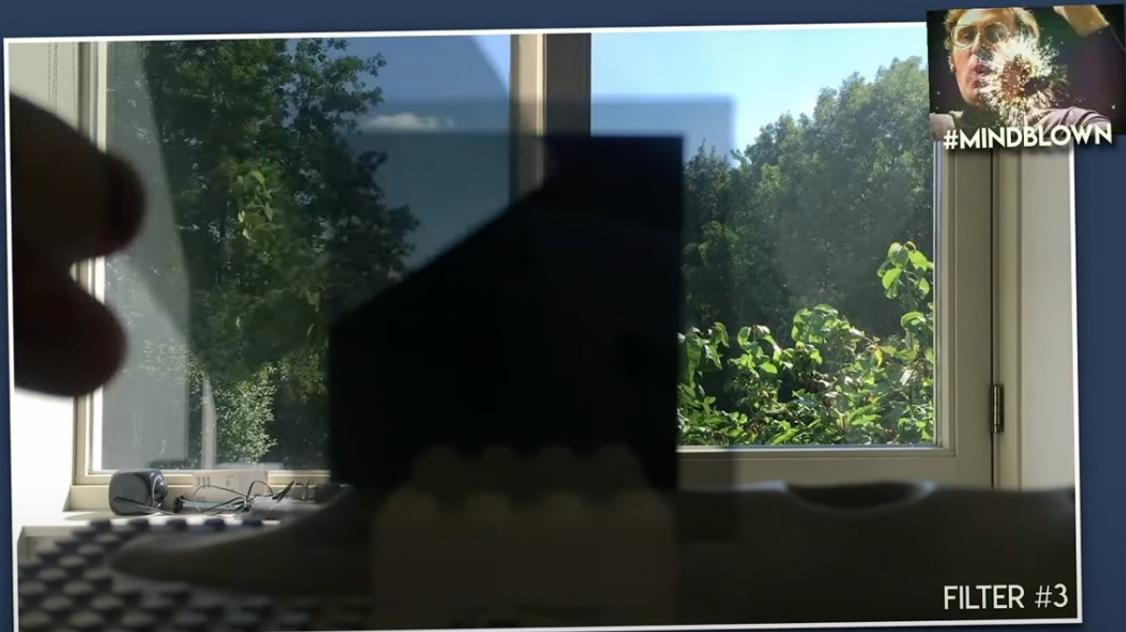
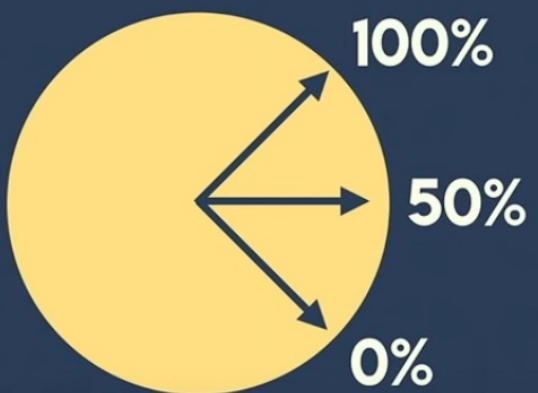
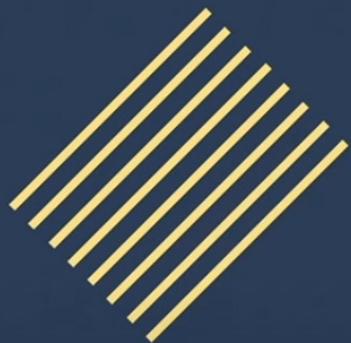


FILTER #2

A diagram illustrating the addition of light waves. It shows two sets of arrows representing waves from different directions. The top set, labeled '+', has four arrows pointing in different directions and is associated with '50% LIGHT'. The bottom set, also labeled '+', has four arrows pointing in different directions and is associated with '0% LIGHT DARKNESS...'. A double-headed arrow between the two sets indicates they are being added together.

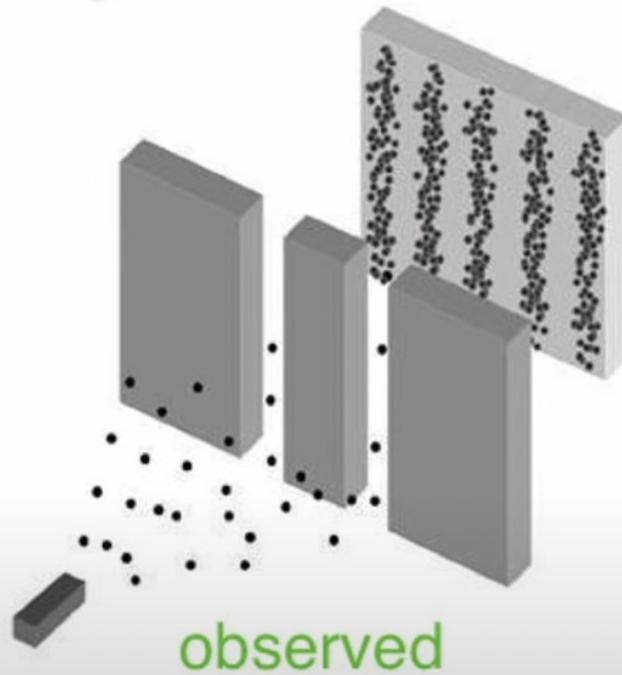
Quantum Mechanics ... are weird, hard to understand, invisible – e.g. Light Duality: particle (photons) and wave (energy)

## MIDDLE FILTER



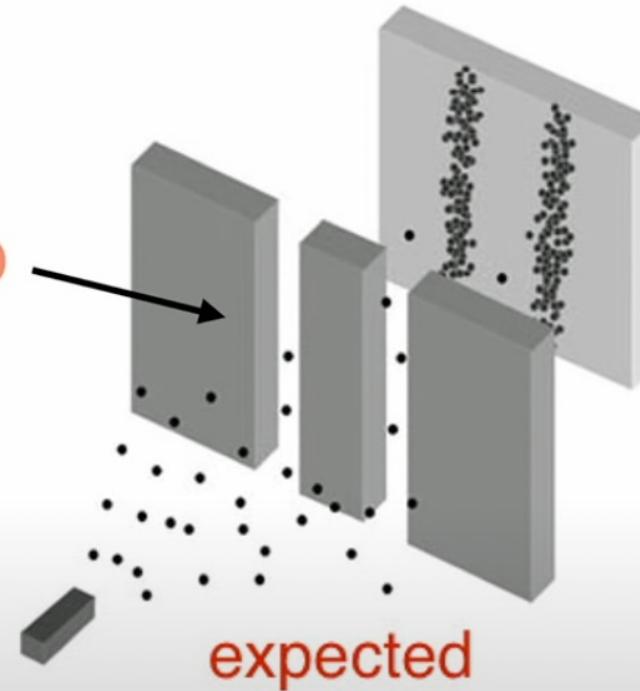
Quantum Mechanics ... are weird, hard to understand, invisible – e.g. Light Duality: particle (photons) and wave (energy)

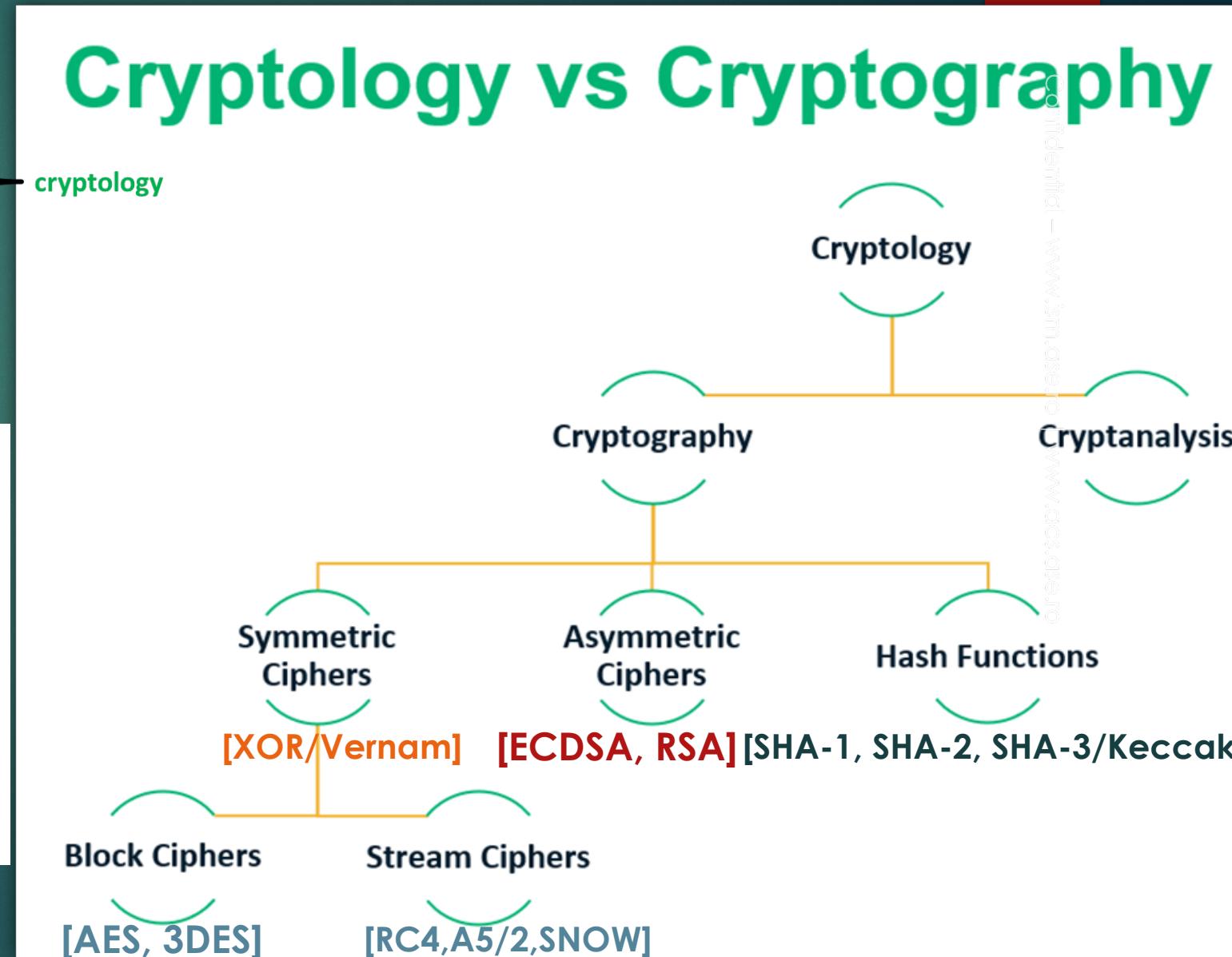
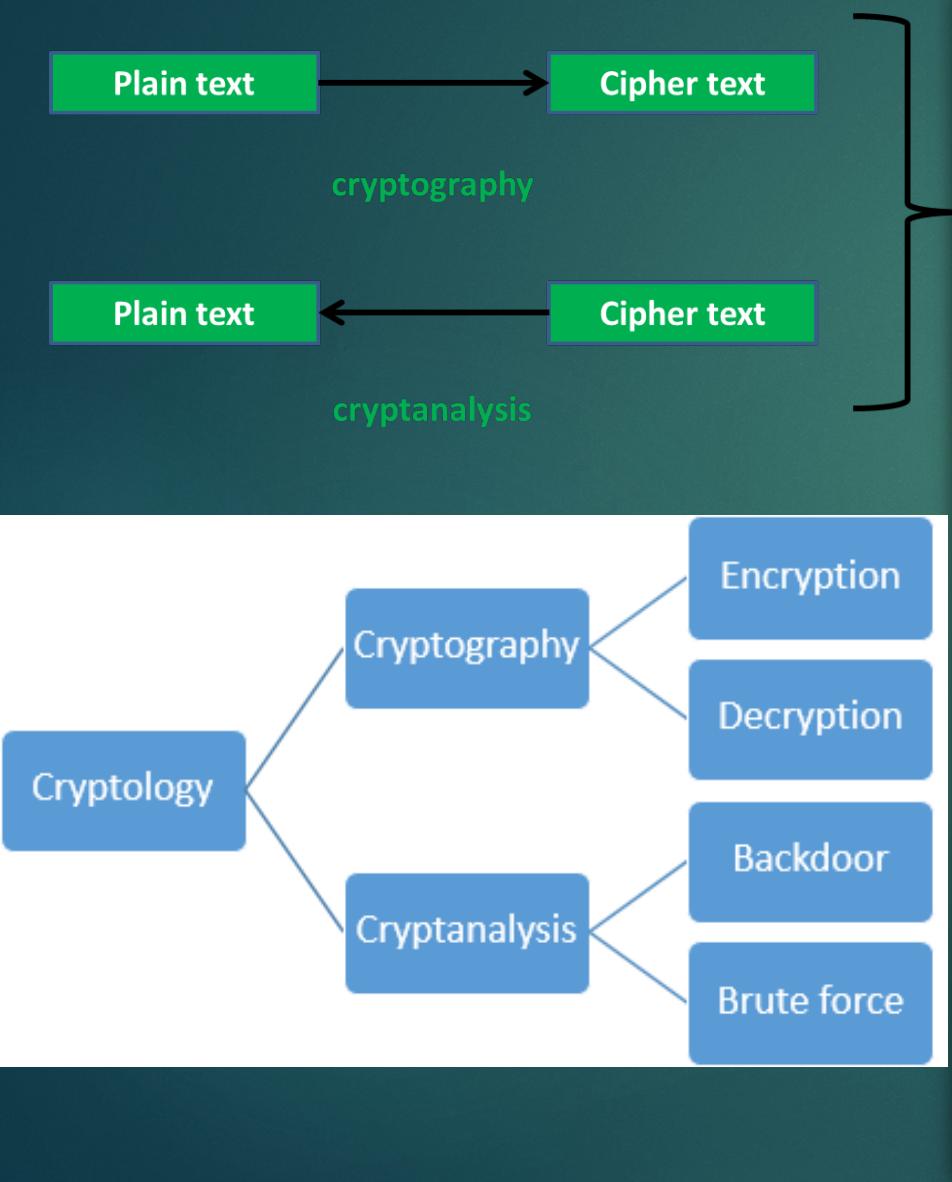
## DOUBLE SLIT EXPERIMENT



## DOUBLE SLIT EXPERIMENT

WHEN  
MEASURED





# Crypto – XOR with symmetric key

XOR = Exclusive or or exclusive disjunction is a logical operation that outputs true only when inputs differ (one is true, the other is false).

AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

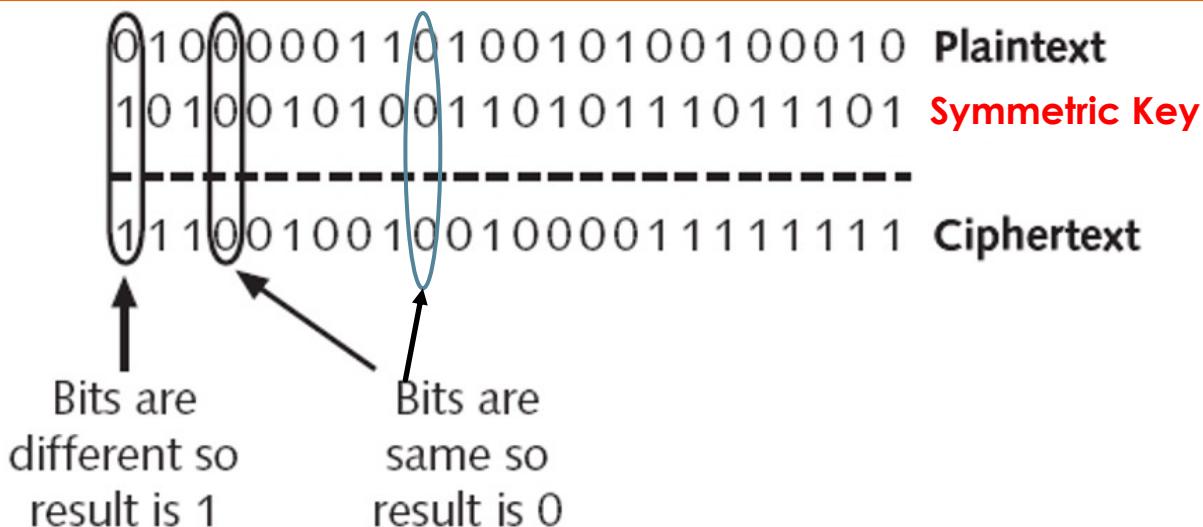
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NOT Truth Table

A	B
0	1
1	0



Exclusive disjunction essentially means 'either one, but not both nor none'. In other words, the statement is true if and only if one is true and the other is false. For example, if two horses are racing, then one of the two will win the race, but not both of them.

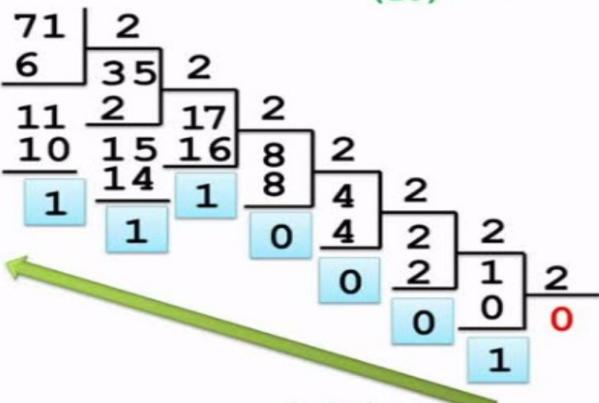
# Crypto – XOR with symmetric key

23

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr		
0	0 000	NUL	(null)	32	20 040	&#32;	Space		64	40 100	&#64;	Ø	96	60 140	&#96;	`	128	80 140	&#96;	~
1	1 001	SOH	(start of heading)	33	21 041	&#33;	!	!	65	41 101	&#65;	A	97	61 141	&#97;	a	192	81 141	&#97;	ä
2	2 002	STX	(start of text)	34	22 042	&#34;	"	"	66	42 102	&#66;	B	98	62 142	&#98;	b	224	82 142	&#98;	ß
3	3 003	ETX	(end of text)	35	23 043	&#35;	#	#	67	43 103	&#67;	C	99	63 143	&#99;	c	256	83 143	&#99;	ç
4	4 004	EOT	(end of transmission)	36	24 044	&#36;	\$	\$	68	44 104	&#68;	D	100	64 144	&#100;	d	288	84 144	&#100;	¤
5	5 005	ENQ	(enquiry)	37	25 045	&#37;	%	%	69	45 105	&#69;	E	101	65 145	&#101;	e	320	85 145	&#101;	€
6	6 006	ACK	(acknowledge)	38	26 046	&#38;	&	&	70	46 106	&#70;	F	102	66 146	&#102;	f	352	86 146	&#102;	ƒ
7	7 007	BEL	(bell)	39	27 047	&#39;	'	'	71	47 107	&#71;	G	103	67 147	&#103;	g	384	87 147	&#103;	ѓ
8	8 010	BS	(backspace)	40	28 050	&#40;	{	{	72	48 110	&#72;	H	104	68 150	&#104;	h	416	88 150	&#104;	߱
9	9 011	TAB	(horizontal tab)	41	29 051	&#41;	)	)	73	49 111	&#73;	I	105	69 151	&#105;	i	448	89 151	&#105;	߳
10	A 012	LF	(NL line feed, new line)	42	2A 052	&#42;	*	*	74	4A 112	&#74;	J	106	6A 152	&#106;	j	480	90 152	&#106;	߲
11	B 013	VT	(vertical tab)	43	2B 053	&#43;	+	+	75	4B 113	&#75;	K	107	6B 153	&#107;	k	512	91 153	&#107;	ߴ
12	C 014	FF	(NP form feed, new page)	44	2C 054	&#44;	,	,	76	4C 114	&#76;	L	108	6C 154	&#108;	l	544	92 154	&#108;	ߵ
13	D 015	CR	(carriage return)	45	2D 055	&#45;	-	-	77	4D 115	&#77;	M	109	6D 155	&#109;	m	576	93 155	&#109;	߶
14	E 016	SO	(shift out)	46	2E 056	&#46;	.	.	78	4E 116	&#78;	N	110	6E 156	&#110;	n	608	94 156	&#110;	߷
15	F 017	SI	(shift in)	47	2F 057	&#47;	/	/	79	4F 117	&#79;	O	111	6F 157	&#111;	o	640	95 157	&#111;	߸
16	10 020	DLE	(data link escape)	48	30 060	&#48;	0	0	80	50 120	&#80;	P	112	70 160	&#112;	p	672	96 160	&#112;	߹
17	11 021	DC1	(device control 1)	49	31 061	&#49;	1	1	81	51 121	&#81;	Q	113	71 161	&#113;	q	704	97 161	&#113;	߻
18	12 022	DC2	(device control 2)	50	32 062	&#50;	2	2	82	52 122	&#82;	R	114	72 162	&#114;	r	736	98 162	&#114;	߻
19	13 023	DC3	(device control 3)	51	33 063	&#51;	3	3	83	53 123	&#83;	S	115	73 163	&#115;	s	768	99 163	&#115;	߻
20	14 024	DC4	(device control 4)	52	34 064	&#52;	4	4	84	54 124	&#84;	T	116	74 164	&#116;	t	800	100 164	&#116;	߻
21	15 025	NAK	(negative acknowledgement)	53	35 065	&#53;	5	5	85	55 125	&#85;	U	117	75 165	&#117;	u	832	101 165	&#117;	߻
22	16 026	SYN	(synchronous idle)	54	36 066	&#54;	6	6	86	56 126	&#86;	V	118	76 166	&#118;	v	864	102 166	&#118;	߻
23	17 027	ETB	(end of trans. block)	55	37 067	&#55;	7	7	87	57 127	&#87;	W	119	77 167	&#119;	w	900	103 167	&#119;	߻
24	18 030	CAN	(cancel)	56	38 070	&#56;	8	8	88	58 130	&#88;	X	120	78 170	&#120;	x	932	104 170	&#120;	߻
25	19 031	EM	(end of medium)	57	39 071	&#57;	9	9	89	59 131	&#89;	Y	121	79 171	&#121;	y	964	105 171	&#121;	߻
26	1A 032	SUB	(substitute)	58	3A 072	&#58;	:	:	90	5A 132	&#90;	Z	122	7A 172	&#122;	z	1000	106 172	&#122;	߻
27	1B 033	ESC	(escape)	59	3B 073	&#59;	;	;	91	5B 133	&#91;	[	123	7B 173	&#123;	{	1032	107 173	&#123;	߻
28	1C 034	FS	(file separator)	60	3C 074	&#60;	<	<	92	5C 134	&#92;	\	124	7C 174	&#124;		1064	108 174	&#124;	߻
29	1D 035	GS	(group separator)	61	3D 075	&#61;	=	=	93	5D 135	&#93;	]	125	7D 175	&#125;	}	1100	109 175	&#125;	߻
30	1E 036	RS	(record separator)	62	3E 076	&#62;	>	>	94	5E 136	&#94;	^	126	7E 176	&#126;	~	1132	110 176	&#126;	߻
31	1F 037	US	(unit separator)	63	3F 077	&#63;	?	?	95	5F 137	&#95;	_	127	7F 177	&#127;	DEL	1164	111 177	&#127;	߻

Source: [www.LookupTables.com](http://www.LookupTables.com)

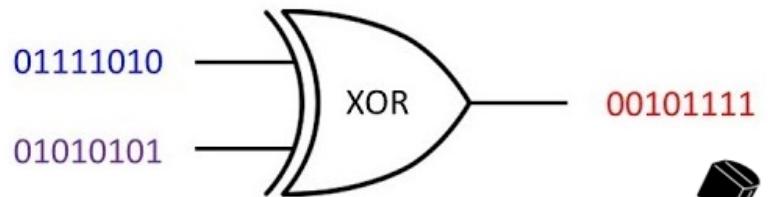
$$71_{(10)} = 1000111_{(2)}$$



kalytheo

# Crypto – XOR with symmetric key

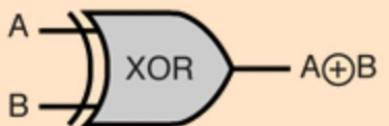
## ENCRYPTION



## DECRYPTION



## Exclusive OR Gate



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

The output is high when either of inputs A or B is high, but not if both A and B are high.

# Electronics 4 Beginners – the YouTube 3D videos are NOT necessary correct, but they are “good enough” to understand!!!

What is Electricity: <https://www.youtube.com/watch?v=Dx3RpXdJw2k>

What is Electricity: <https://www.youtube.com/watch?v=ru032Mfsfig&t=501s>

Voltage (min 1:00 batteries): <https://www.youtube.com/watch?v=wmaobOy8r6I>

Electric Potential – Voltage: <https://www.youtube.com/watch?v=-Rb9guSEeVE>

What is Ampere?: <https://www.youtube.com/watch?v=xr68GxkiWFM>

Current / I (min 1:11): <https://www.youtube.com/watch?v=tbxwAIA-10k>

Ohm's Law: <https://www.youtube.com/watch?v=G3H5IKoWPpY> ( $I = U/R$ )

Capacitor: [https://www.youtube.com/watch?v=f\\_MZNsEgyQw](https://www.youtube.com/watch?v=f_MZNsEgyQw)

TRANSISTOR (PHY + CHEMISTRY): <https://www.youtube.com/watch?v=lcrBqCFLHIY>

Logical Gates: <https://www.youtube.com/watch?v=sTu3LwpF6XI>

AND Gate: <https://www.youtube.com/watch?v=aZrTrMiy0iY>

AC (Alternating Current) vs. DC (Direct Current): <https://www.youtube.com/watch?v=bt6A5HjtNso>

Big Misconception about Electricity: <https://www.youtube.com/watch?v=bHIhgxav9LY>

...

## Students:

Devoxx QBits Entanglement: <https://www.youtube.com/watch?v=fFW36-CGkPY>

<https://github.com/royvanrijn/Quantastic/tree/master/src/main/java/com/royvanrijn/quantum>

<https://introcs.cs.princeton.edu/java/32class/Complex.java.html>

Devoxx DNA Encoding: <https://www.youtube.com/watch?v=NsfuBSsF1Fk&t=1295s>

FPGA for designing controllers VHDL | Verilog: <https://www.fpga4student.com/p/vhdl-project.html>

| <https://www.fpga4student.com/2017/06/Verilog-code-for-ALU.html>

...

An atom is a particle of matter that uniquely defines a chemical element. An atom consists of a central nucleus that is surrounded by one or more negatively charged electrons. The nucleus is positively charged and contains one or more relatively heavy particles known as protons and neutrons. Silicon (Si) has 4 electrons in the outermost orbit shell, but 14 in total (2, 8, 4). Phosphorus (P) has 5 electrons in the outermost shell, but 15 (2, 8, 5) in total and Boron (B) has 3 electrons in the outermost shell, but 5 in total (2,3).

An ion = any atom or group of atoms that bears one or more positive or negative electrical charges. Positively charged ions are called cations; negatively charged ions, anions.

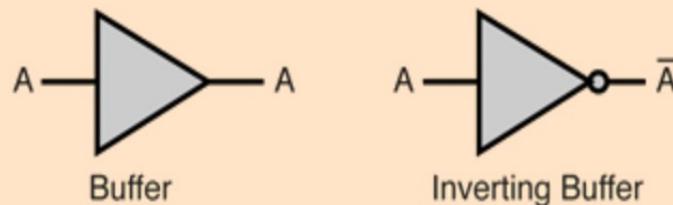
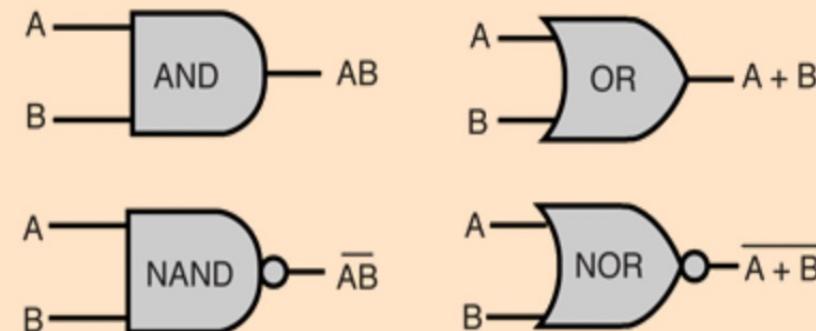
Photons and electrons are two of the basic quantum-mechanical particles but they have completely different properties. Photon is a type of elementary particle which acts as a carrier of energy, but the electron is a subatomic particle which occurs in all the atoms. | A photon can spontaneously degenerate into a particle with mass and its antiparticle in a process known as pair production. In this process, the energy of the photon is completely transformed into the mass of the two particles. For example, a photon can turn into an electron and an anti-electron.

<https://byjus.com/physics/electrons-and-photons/> | <https://wtamu.edu/~cbaird/sq/2013/09/06/can-one-bit-of-light-bounce-off-another-bit-of-light/> | <https://www.techtarget.com/searchcio/definition/atom> | <https://www.britannica.com/science/ion-physics>

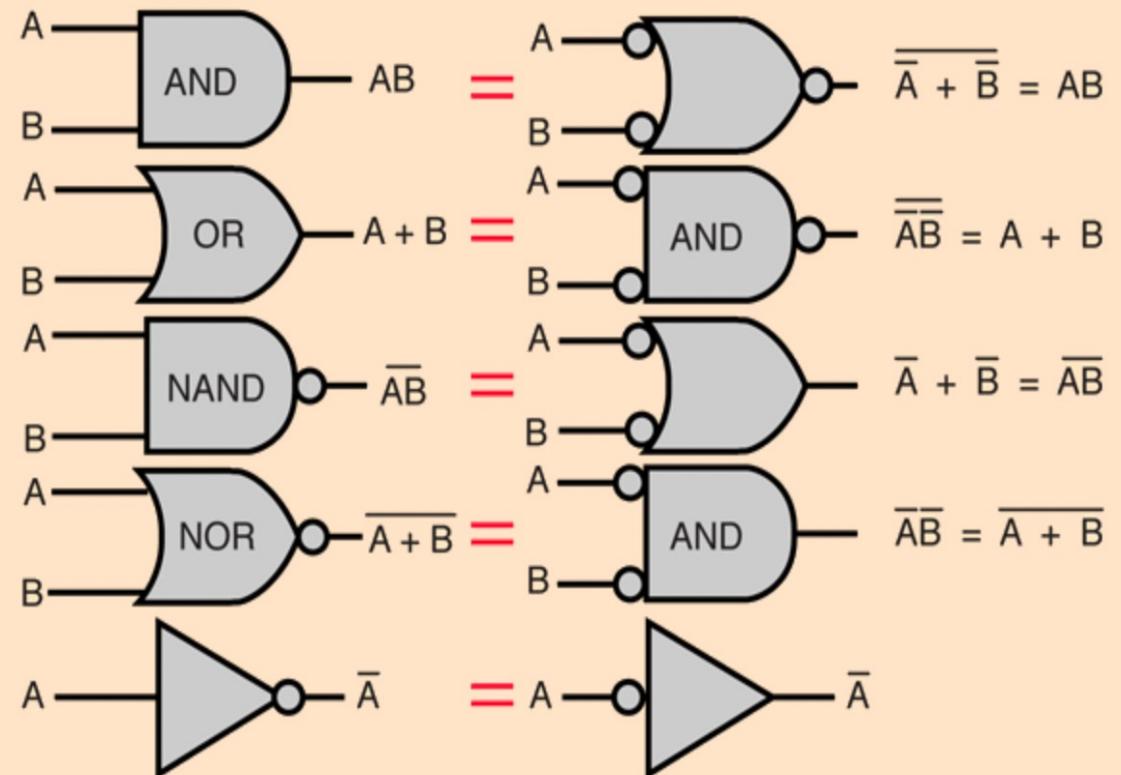
<https://www.youtube.com/watch?v=IcrBqCFLHIY>

## Basic Gates

This is an active graphic. Click on any of the gates for further information.



## Negative Logic Gates



Each of the basic [gates](#) has a negative logic equivalent as shown. The equivalence is shown by the application of [DeMorgan's theorem](#). It amounts to changing AND's to OR's or vice versa and inverting all input and output lines compared to the implementation in gates shown at left.

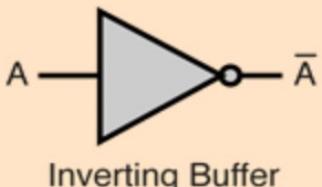
Symbols

Negation or inversion

Negation or inversion

Exclusive OR

## Inverting Buffer



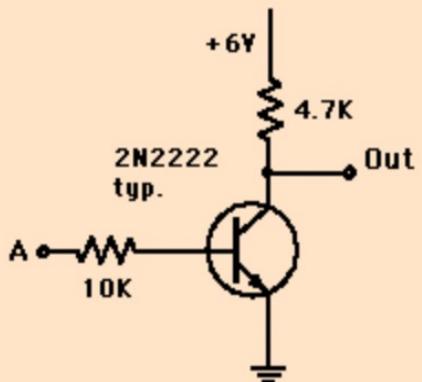
In	Out
0	1
1	0

The inverting buffer is a single-input device which produces the state opposite the input. If the input is high, the output is low and vice versa.

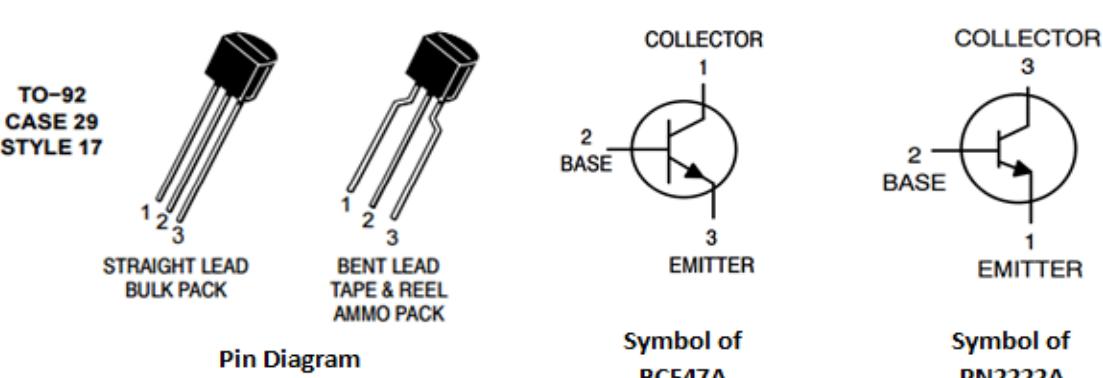
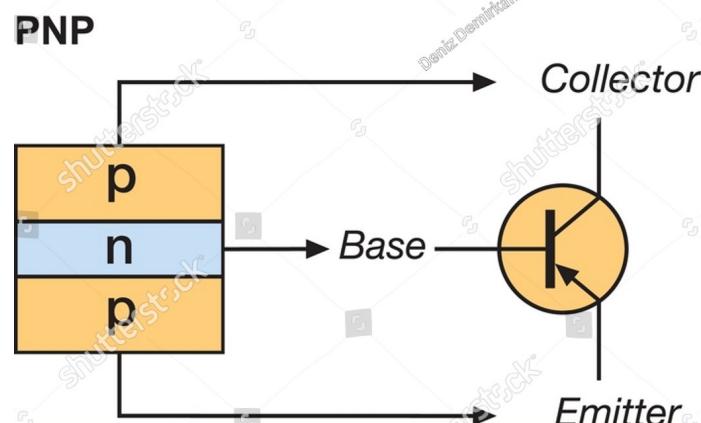
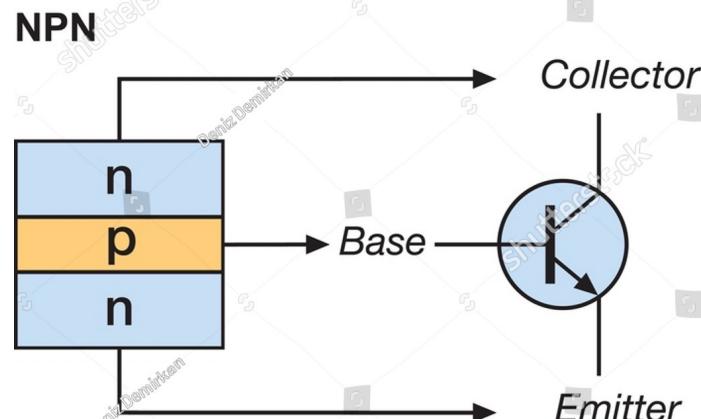
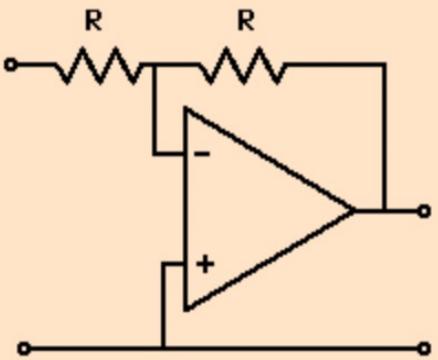
This device is commonly referred to as just an inverter.

## Making an Inverting Buffer

A [transistor switch](#) with collector resistor can serve as an [inverting buffer](#).



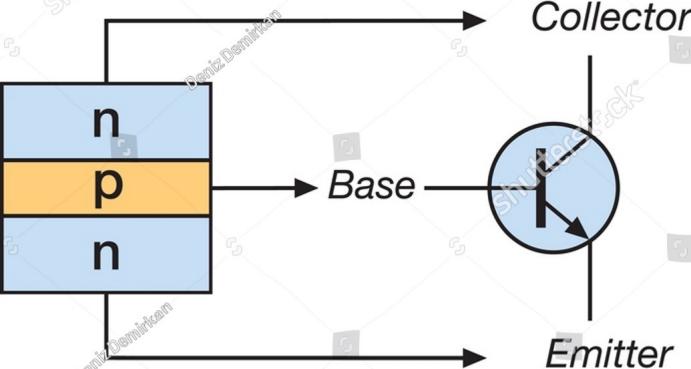
An op-amp [inverting amplifier](#) with a gain of one serves as an inverting buffer.



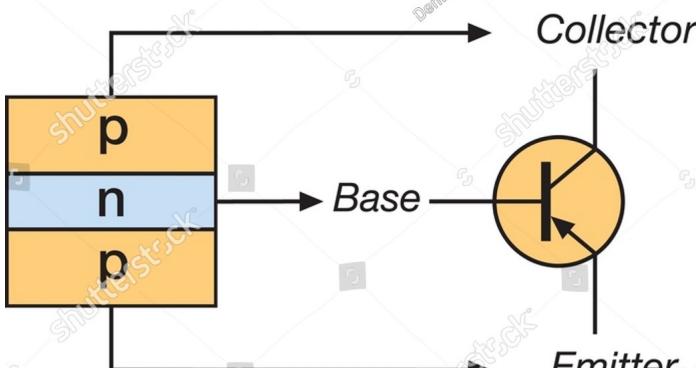
## Electronics & Classical Bits + Gates – Recap

### Bipolar Junction Transistors

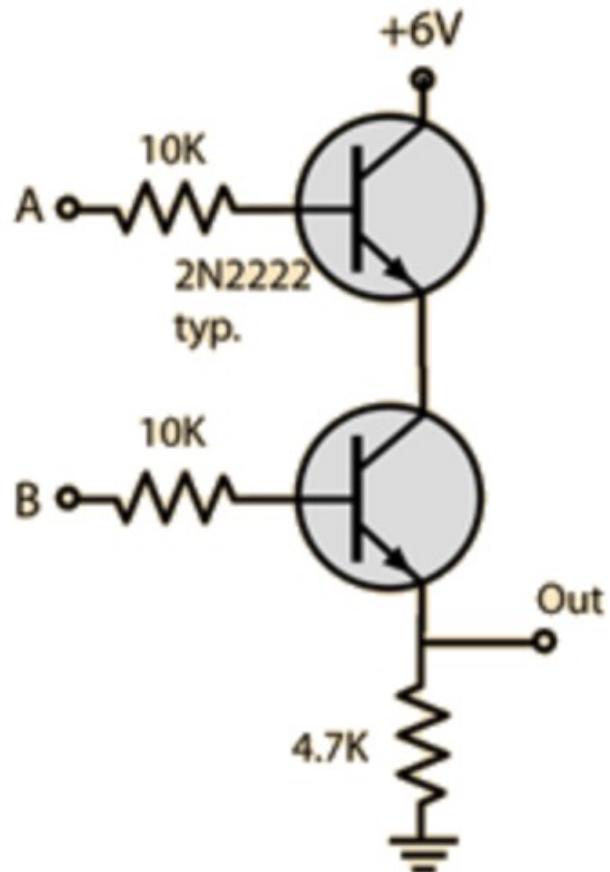
NPN



PNP



<input checked="" type="checkbox"/>	<b>AND Gate</b>
<input type="checkbox"/>	<b>OR Gate</b>
<input type="checkbox"/>	<b>NAND Gate</b>
<input type="checkbox"/>	<b>NOR Gate</b>
<input type="checkbox"/>	<b>Double Transistor</b>
<input type="checkbox"/>	<b>NOR Gate Single Transistor</b>



The use of transistors for the construction of logic gates depends upon their utility as fast switches. When the base-emitter diode is turned on enough to be driven into saturation, the collector voltage with respect to the emitter may be near zero and can be used to construct gates for the TTL logic family. For the AND logic, the transistors are in series and both transistors must be in the conducting state to drive the output high.

<http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/trangate.html#c1>

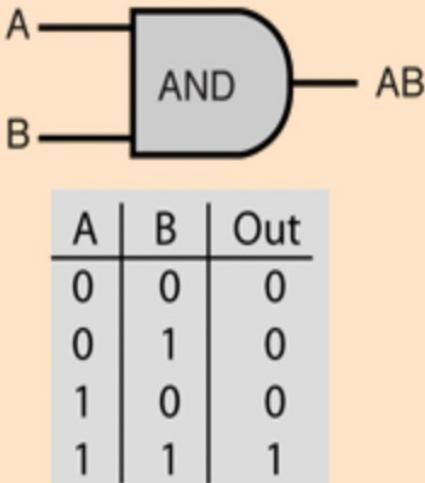
<http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/nand.html#c2>

# Making an AND Gate

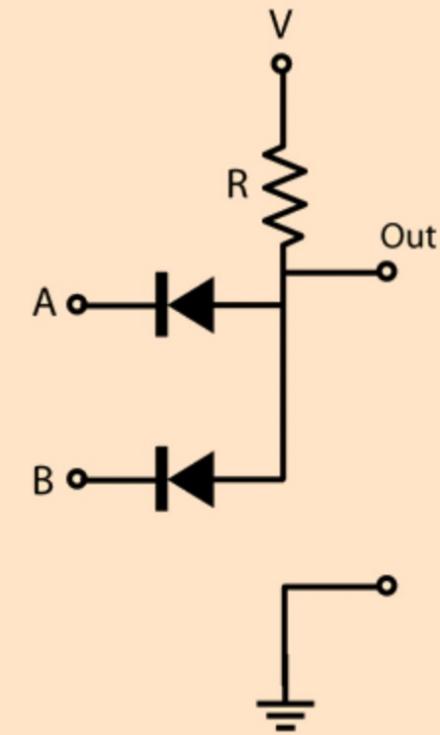
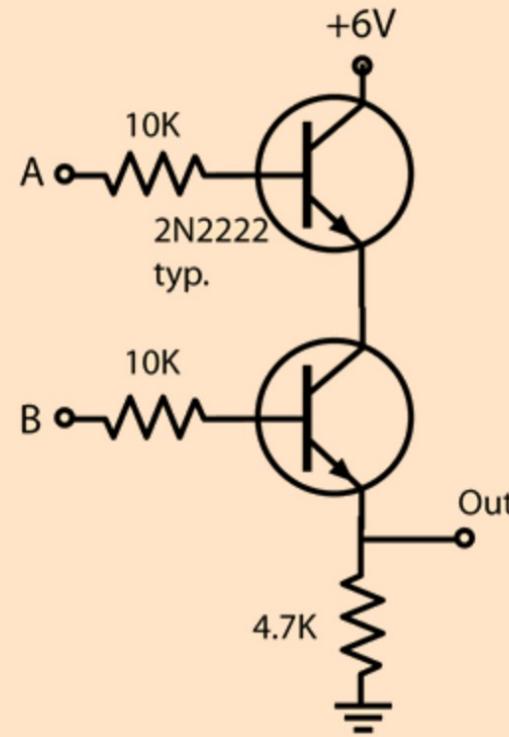
These are ways to make an AND gate from transistor switches or diodes.

## AND Gate

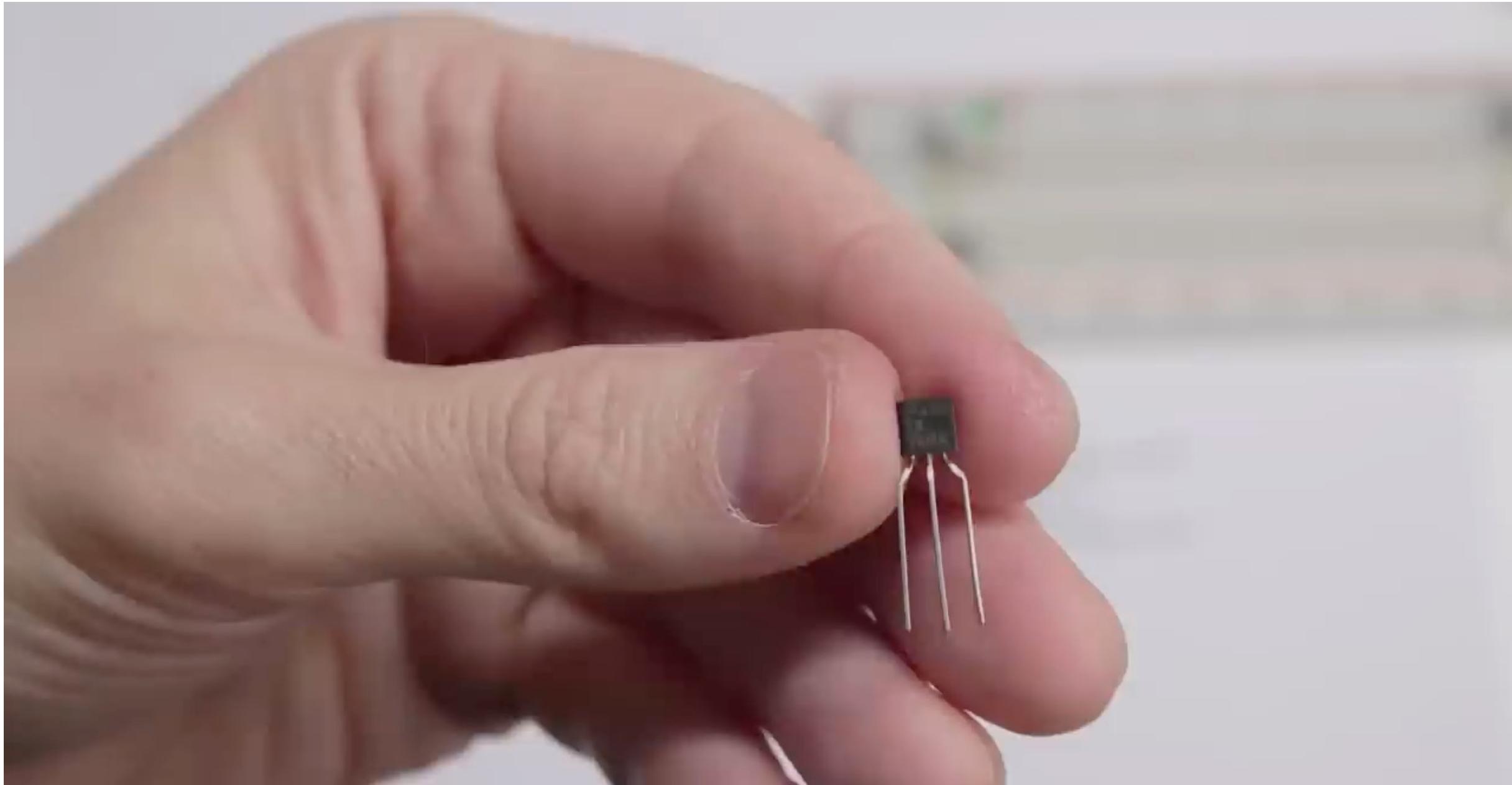
The output is high only when both inputs A and B are high.



The *AND* operation will be signified by  $AB$ . Other notations for it are  $A \wedge B$  and  $A \cap B$ , called the intersection of A and B.



## Electronics & Classical Bits + Gates – Recap 4 bits ALU (Arithmetic Logic Unit)



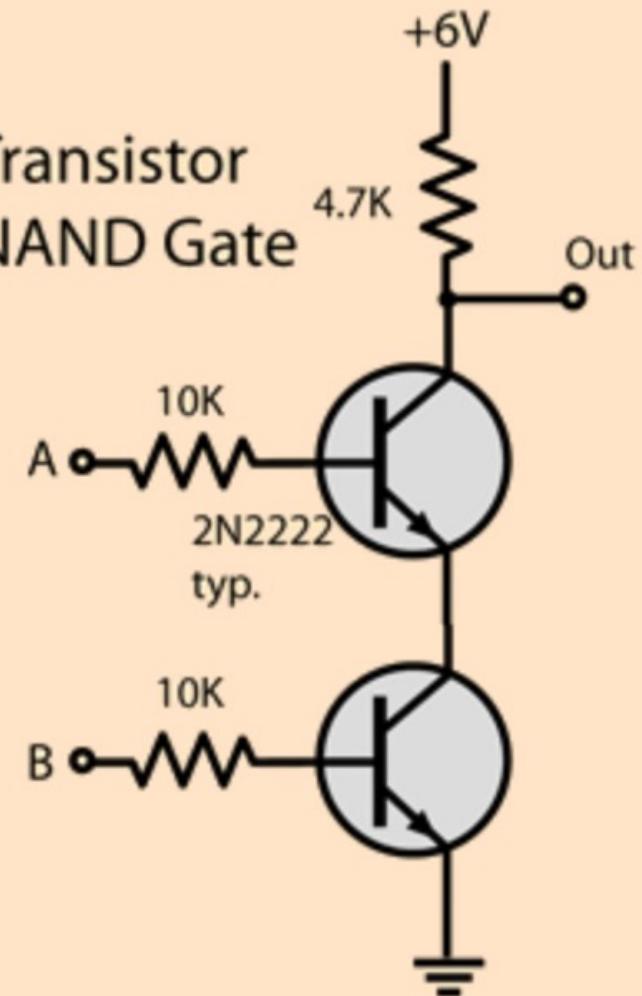
# NAND Gate

The output is high when either of inputs A or B is high, or if neither is high. In other words, it is normally high, going low only if both A and B are high.



The NAND gate and the [NOR gate](#) can be said to be universal gates since [combinations](#) of them can be used to accomplish any of the [basic operations](#) and can thus produce an [inverter](#), an [OR gate](#) or an [AND gate](#). The non-inverting gates do not have this versatility since they can't produce an invert.

Transistor  
NAND Gate

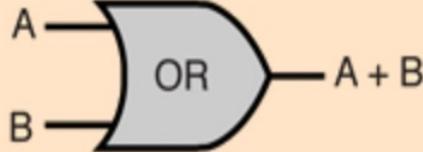


# Making an OR Gate

These are ways to make an OR gate from transistor switches or diodes.

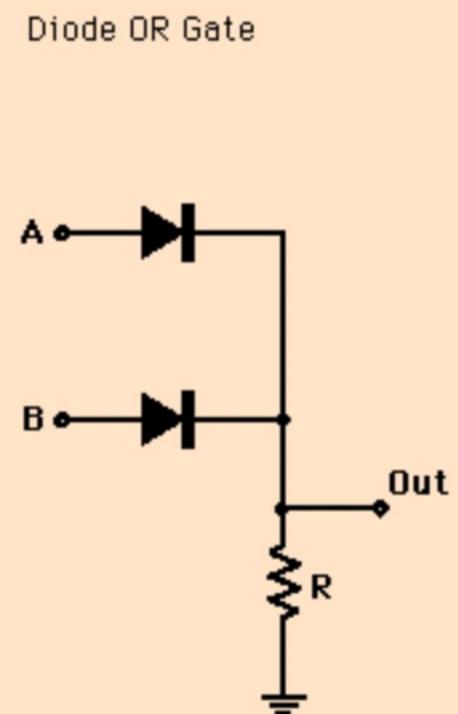
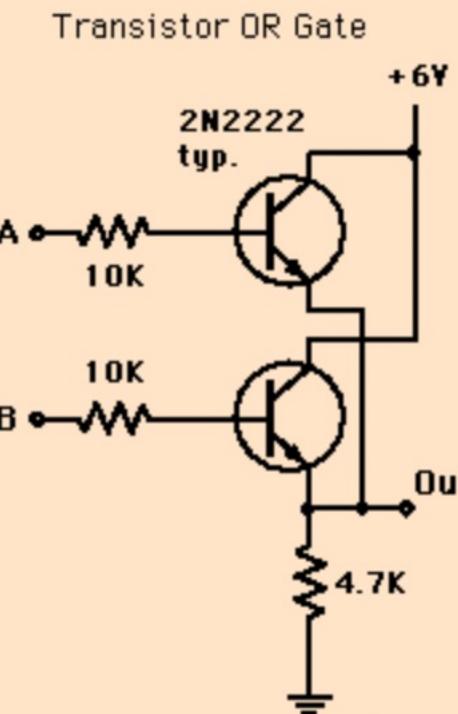
## OR Gate

The output is high when either or both of inputs A or B is high. This is logically different from the exclusive OR



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

The OR operation will be signified by  $A+B$ . Other notations for it are  $A \vee B$  and  $A \cup B$ , called the union of A and B.



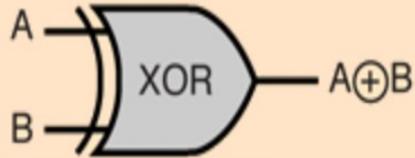
Logically, the exclusive OR (XOR) operation can be seen as either of the following operations:

$$1. A \oplus B = A\bar{B} + B\bar{A} \quad A \text{ AND NOT } B \text{ OR } B \text{ AND NOT } A$$

$$2. A \oplus B = (A + B)(\bar{A}\bar{B}) \quad A \text{ OR } B \text{ AND NOT } A \text{ AND NOT } B$$

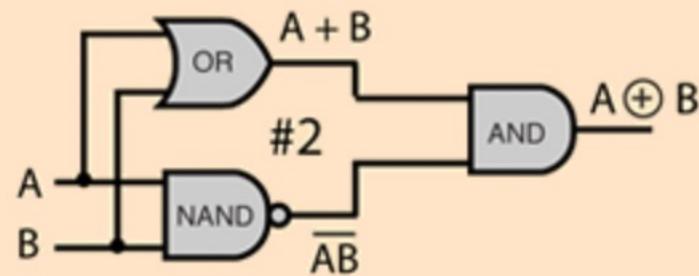
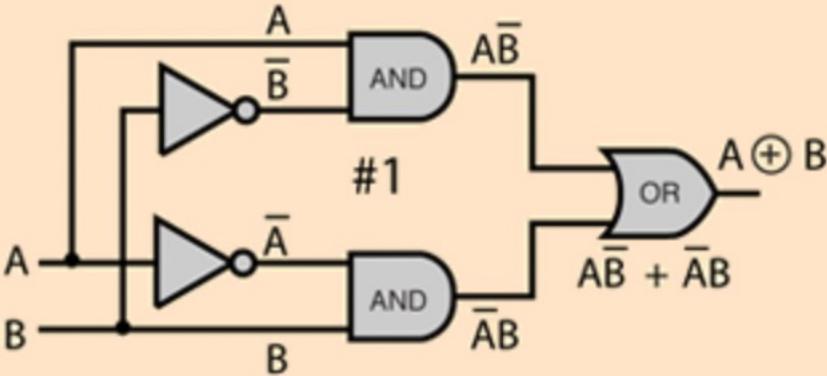
which can be implemented by the gate arrangements shown. They can also be implemented using NAND gates only.

## Exclusive OR Gate

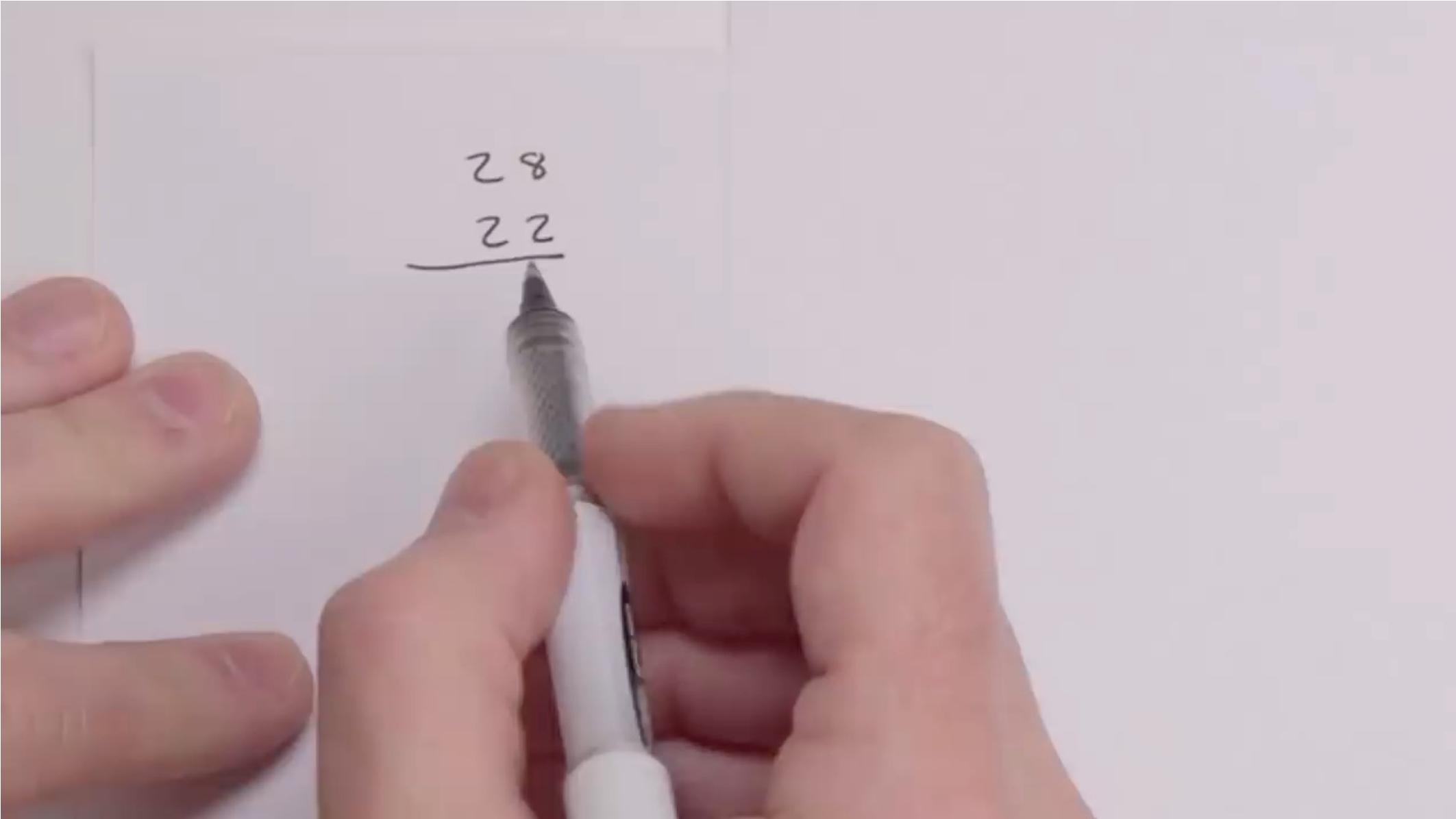


A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

The output is high when either of inputs A or B is high, but not if both A and B are high.

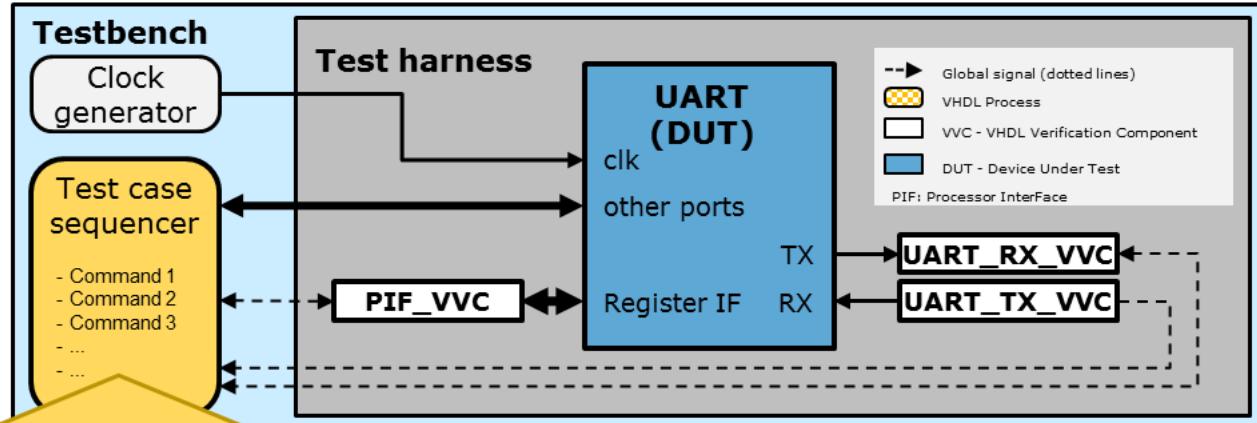


## Electronics & Classical Bits + Gates – Recap 4 bits ALU (Arithmetic Logic Unit)



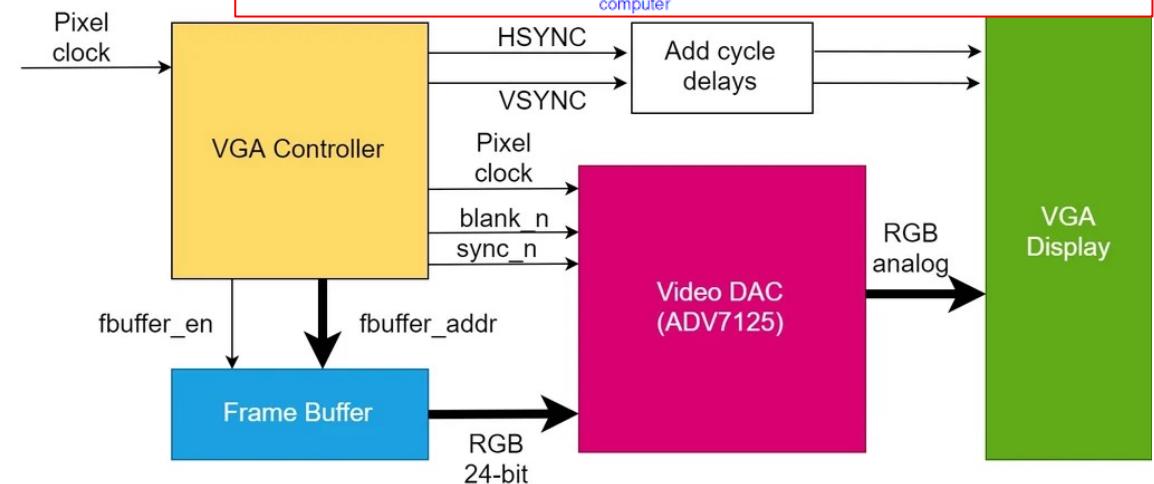
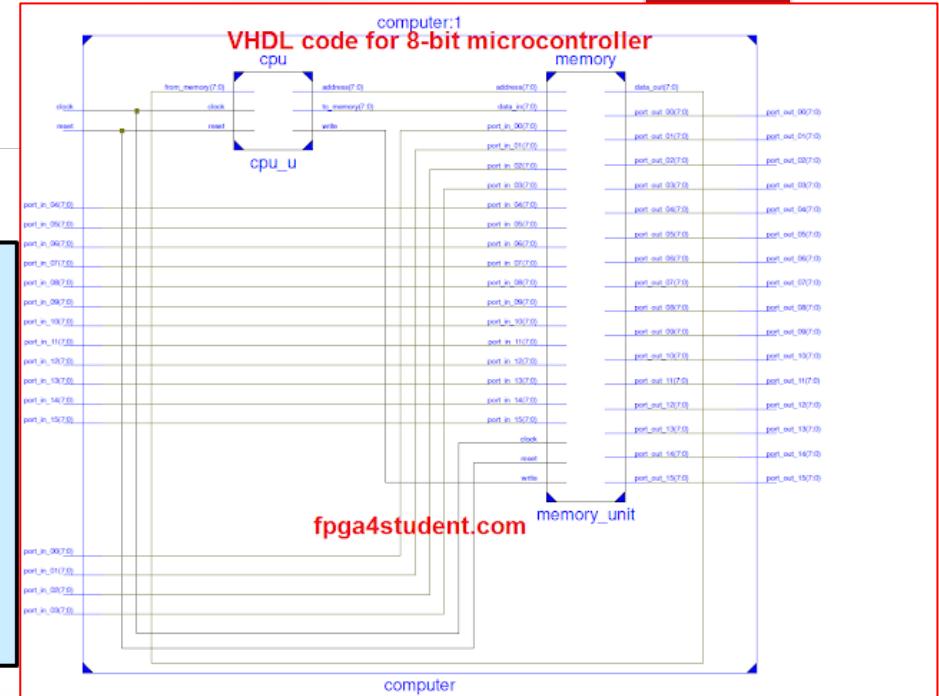
<https://eater.net/8bit/schematics> | <https://www.youtube.com/watch?v=wvJc9CZcvBc>

# UVVM - Universal VHDL Verification Methodology



**Test case sequencer commands to access all interfaces above**

```
pif_write( PIF_VVC,      C_ADDR_UART_TX_DATA, x"A1");
uart_receive( UART_RX_VVC, x"A1");
uart_transmit(UART_TX_VVC, x"4C");
await_value(uart_rx_empty, '0', 0 ns, C_UART_FRAME_TIME);
pif_check(   PIF_VVC,      C_ADDR_UART_RX_DATA, x"4c");
```



# Atoms and subatomic particles

In [physics](#), a **subatomic particle** is a [particle](#) smaller than an [atom](#).<sup>[1]</sup> According to the [Standard Model of particle physics](#), a subatomic particle can be either a [composite particle](#), which is composed of other particles (for example, a [proton](#), [neutron](#), or [meson](#)), or an [elementary particle](#), which is not composed of other particles (for example, an [electron](#), [photon](#), or [muon](#)).<sup>[2]</sup> [Particle physics](#) and [nuclear physics](#) study these particles and how they interact.<sup>[3]</sup>

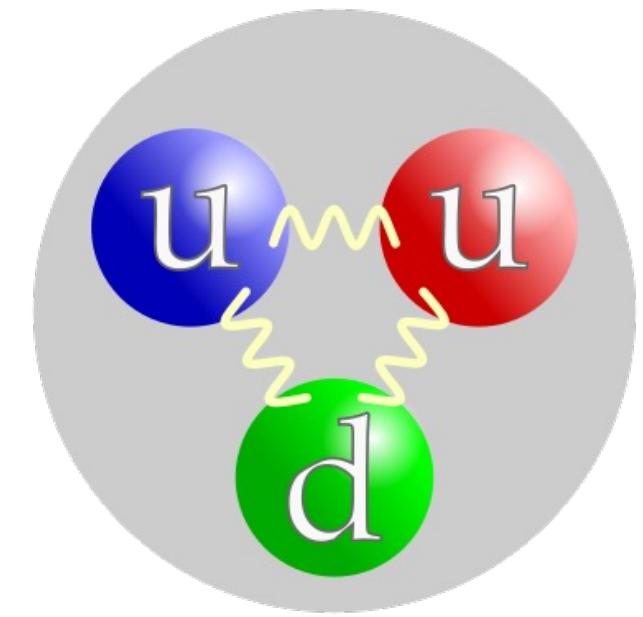
Experiments show that light could behave like a [stream of particles](#) (called [photons](#)) as well as exhibiting wave-like properties. This led to the concept of [wave–particle duality](#) to reflect that quantum-scale *particles* behave like both particles and waves; they are sometimes called **wavicles** to reflect this.<sup>[4]</sup>

Another concept, the [uncertainty principle](#), states that some of their properties taken together, such as their simultaneous [position](#) and [momentum](#), cannot be measured exactly.<sup>[5]</sup> The wave–particle duality has been shown to apply not only to photons but to more massive particles as well.<sup>[6]</sup>

Interactions of particles in the framework of [quantum field theory](#) are understood as creation and annihilation of [quanta](#) of corresponding [fundamental interactions](#). This blends particle physics with [field theory](#).

Even among [particle physicists](#), the exact definition of a particle has diverse descriptions. These professional attempts at the definition of a particle include:<sup>[7]</sup>

- A particle is a [collapsed wave function](#)
- A particle is a [quantum excitation of a field](#)
- A particle is an [irreducible representation of the Poincaré group](#)
- A particle is an observed thing



A composite particle [proton](#) is made of two [up quark](#) and one [down quark](#), which are [elementary particles](#)

# Atoms and subatomic particles

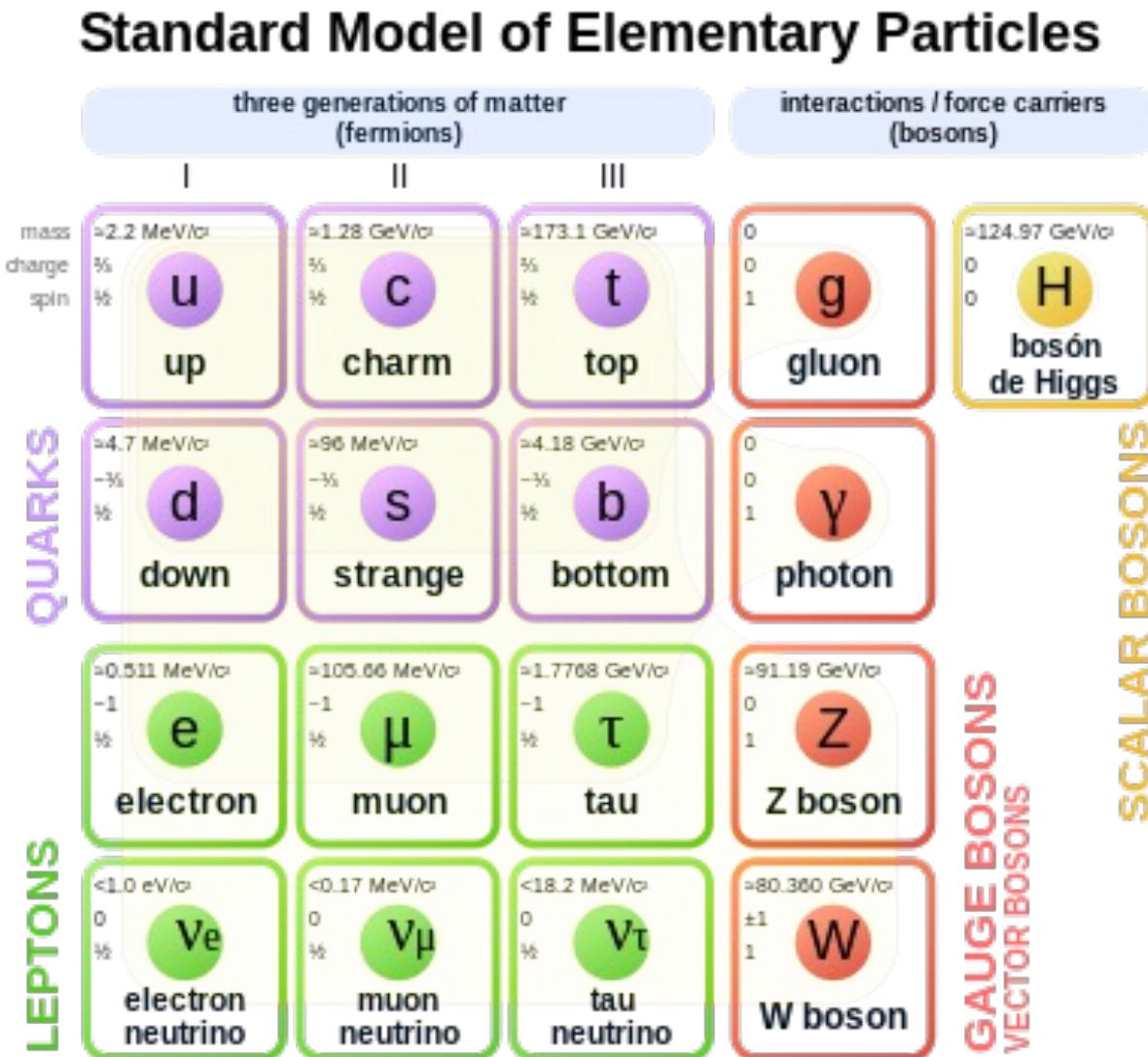
Subatomic particles are either "elementary", i.e. not made of multiple other particles, or "composite" and made of more than one elementary particle bound together.

The elementary particles of the Standard Model are:<sup>[8]</sup>

- Six "flavors" of quarks: up, down, strange, charm, bottom, and top;
  - Six types of leptons: electron, electron neutrino, muon, muon neutrino, tau, tau neutrino;
  - Twelve gauge bosons (force carriers): the photon of electromagnetism, the three W and Z bosons of the weak force, and the eight gluons of the strong force;
  - The Higgs boson.

All of these have now been discovered by experiments, with the latest being the top quark (1995), tau neutrino (2000), and Higgs boson (2012).

Various [extensions of the Standard Model](#) predict the existence of an elementary [graviton](#) particle and [many other elementary particles](#), but none have been discovered as of 2021.



## The Standard Model classification of particles

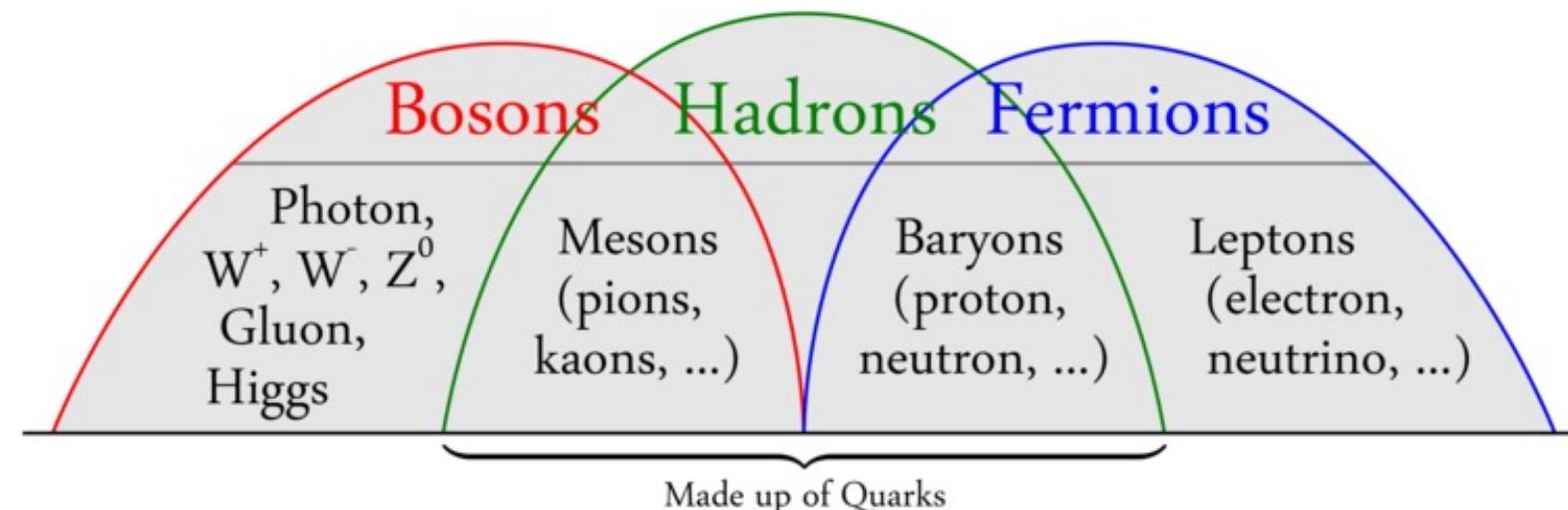
# Atoms and subatomic particles

Any subatomic particle, like any particle in the [three-dimensional space](#) that obeys the [laws](#) of [quantum mechanics](#), can be either a [boson](#) (with integer [spin](#)) or a [fermion](#) (with odd half-integer spin).

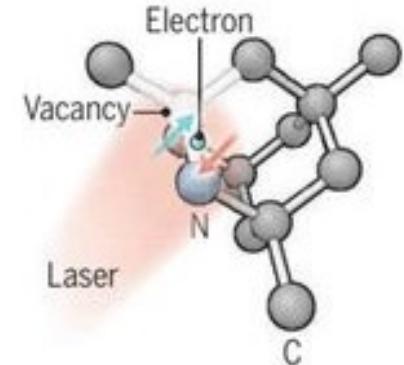
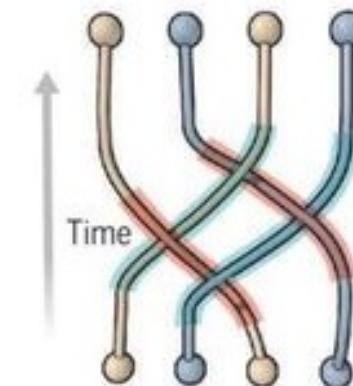
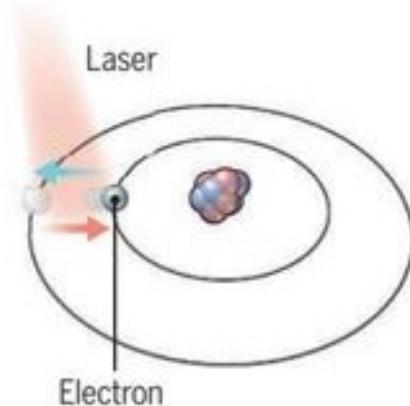
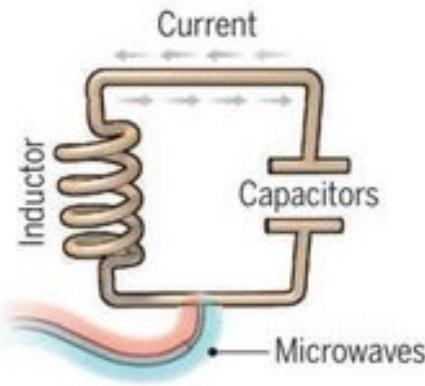
In the Standard Model, all the elementary fermions have spin 1/2, and are divided into the [quarks](#) which carry [color charge](#) and therefore feel the strong interaction, and the [leptons](#) which do not. The elementary bosons comprise the [gauge bosons](#) (photon, W and Z, gluons) with spin 1, while the [Higgs boson](#) is the only elementary particle with spin zero.

The hypothetical [graviton](#) is required theoretically to have spin 2, but is not part of the Standard Model. Some extensions such as [supersymmetry](#) predict additional elementary particles with spin 3/2, but none have been discovered as of 2021.

Due to the laws for spin of composite particles, the baryons (3 quarks) have spin either 1/2 or 3/2 and are therefore fermions; the mesons (2 quarks) have integer spin of either 0 or 1 and are therefore bosons.



# Technologies to build quantum computers



## Superconducting loops

### Company support

Google, IBM, Quantum Circuits

## Trapped ions

ionQ

## Silicon quantum dots

Intel

## Topological qubits

Microsoft,  
Bell Labs

## Diamond vacancies

Quantum Diamond  
Technologies

### Pros

Fast working. Build on existing semiconductor industry.

Very stable. Highest achieved gate fidelities.

Stable. Build on existing semiconductor industry.

Greatly reduce errors.

Can operate at room temperature.

### Cons

Collapse easily and must be kept cold.

Slow operation. Many lasers are needed.

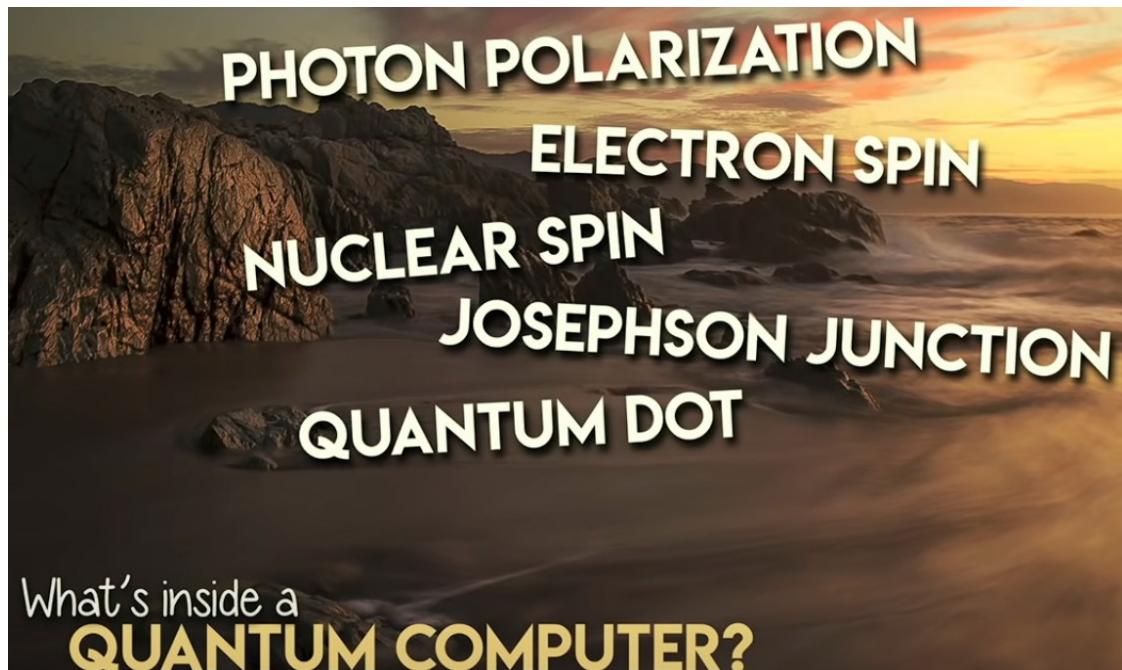
Only a few entangled. Must be kept cold.

Existence not yet confirmed.

Difficult to entangle.

Image credits: Graphic by C. Bickle/Science data by Gabriel Popkin

# Technologies to build quantum computers



# IBM Q

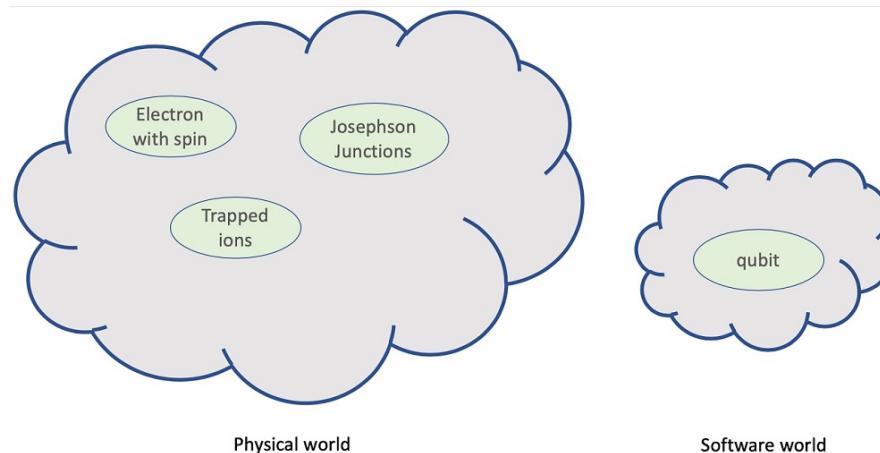
Elias F. Combarro – University of Oviedo, CERN  
December 2020, “A Practical Introduction to Quantum Computing: From Qubits to Quantum Machine Learning and Beyond”

IBM Quantum System One is the first circuit-based commercial quantum computer, introduced by IBM in January 2019. This integrated quantum computing system is housed in a 2.7 by 2.7 by 2.7 meters airtight glass cube that maintains a controlled physical environment. [Wikipedia](#)

<https://www.ibm.com/quantum>

<https://www.ibm.com/quantum/tools>

<https://quantum-computing.ibm.com/composer/files/new>



# Technologies to interact with quantum computers – quantum bits (qubits), gates and circuits

<https://qiskit.org/> | <https://en.wikipedia.org/wiki/OpenQASM> | <https://github.com/openqasm/openqasm/tree/OpenQASM2.x> | <https://github.com/openqasm/openqasm>

Switch to Composer

Backend: Custom Topology Experiment Units: 3 ⓘ

Simulate

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg c0[1];
creg c1[1];
creg c2[1];

gate post q { }
u3(0.3,0.2,0.1) q[0];
h q[1];
cx q[1],q[2];
barrier q;
cx q[0],q[1];
h q[0];
measure q[0] -> c0[0];
measure q[1] -> c1[0];
if(c0==1) z q[2];
if(c1==1) x q[2];
post q[2];
measure q[2] -> c2[0];
```

Import QASM Download QASM

Select a provider below and explore code examples of how to use it with Qiskit:

- IBM Quantum Learn more ↗
- IonQ
- IQM
- MQT DDSIM
- NVIDIA cuStateVec
- QC Ware Forge
- QuaC
- Quantinuum
- Rigetti
- Strangeworks

Install

```
pip install qiskit
pip install qiskit-ibm-provider qiskit-ibm-runtime
```

Build and run

Transpile Sample a Bell State Run VQE

```
from qiskit_ibm_provider import IBMProvider
# Get the API token in https://quantum-computing.ibm.com/account
provider = IBMProvider(token="MY_IBM_QUANTUM_TOKEN")
backend = provider.get_backend("ibm_nairobi")
```

```
# Build circuit
from qiskit.circuit.library import QuantumVolume
circuit = QuantumVolume(5)
```

```
# Transpile circuit
from qiskit import transpile
transpiled_circuit = transpile(circuit, backend)
transpiled_circuit.draw()
```

<https://quantum-computing.ibm.com/>

<https://quantum-computing.ibm.com/composer/>

<https://quantum-computing.ibm.com/docs/>

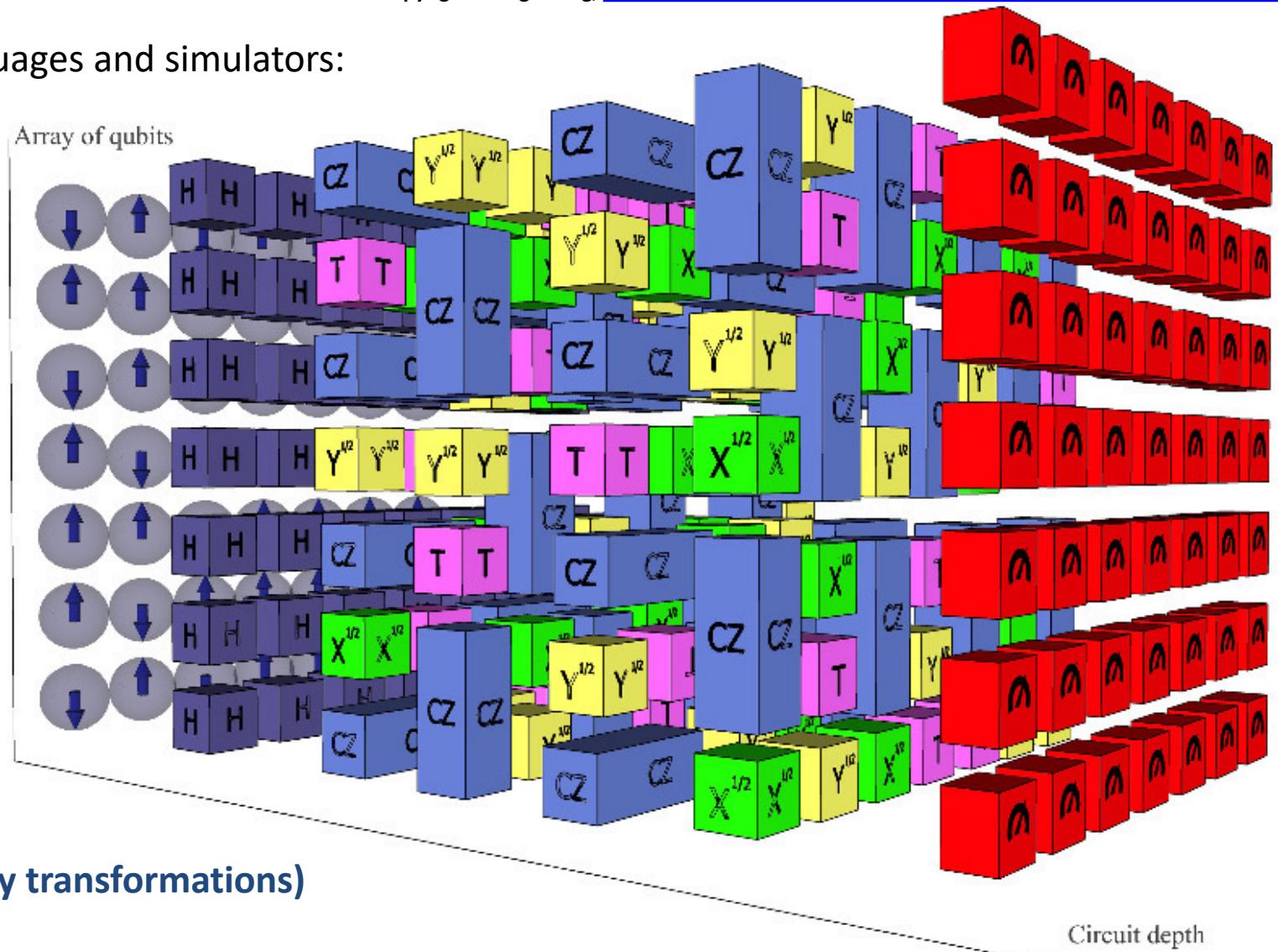
# Technologies to interact with quantum computers – quantum bits (qubits), gates and circuits

Copyright: Google Blog, <https://ai.googleblog.com/2018/05/the-question-of-quantum-supremacy.html>

- Different frameworks, programming languages and simulators:

- QASM 2.0 | 3.0 -> Qiskit**
- Qiskit (IBM)*
- Cirq (Google)
- Forest/pyquil (Rigetti)
- Q# (Microsoft)
- Ocean (D-Wave)
- Java Strange**
- ...

- Most of them for:**
  - quantum circuit specification**
- Every computation has three elements:**
  - data, operations and results**
- In quantum circuits:**
  - Data = qubits**
  - Operations = quantum gates (unitary transformations)**
  - Results = measurements**



<https://qiskit.org/> | <https://en.wikipedia.org/wiki/OpenQASM> | <https://github.com/openqasm/openqasm/tree/OpenQASM2.x> | <https://github.com/openqasm/openqasm>  
<https://github.com/redfx-quantum/strange> | <https://github.com/johanvos/quantumjava> | <https://github.com/topics/quantum> | <https://github.com/royvanrijn/Quantastic> |  
<https://github.com/critoma/quantum>

In mathematics, the greatest common divisor (GCD) of two or more integers, which are not all zero, is the largest positive integer that divides each of the integers. For two integers  $x, y$ , the greatest common divisor of  $x$  and  $y$  is denoted  $\text{gcd}(x, y)$ . For example, the GCD of 8 and 12 is 4, that is,  $\text{gcd}(8, 12)=4$ . In the name "greatest common divisor", the adjective "greatest" may be replaced by "highest", and the word "divisor" may be replaced by "factor", so that other names include highest common factor (hcf). Historically, other names for the same concept have included greatest common measure.

### GCD of Two Numbers

Let us see the steps given below to learn how to find the GCD of two numbers.

- **Step 1:** Write the divisors of the number 'a'.
- **Step 2:** Write the divisors of the number 'b'.
- **Step 3:** List the common divisors of 'a' and 'b'.
- **Step 4:** Now find the divisor which is the highest among the common divisors.

**Example:** Find the greatest common divisor of 13 and 48.

**Solution:** We will use the following steps to find the greatest common divisor of (13, 48).

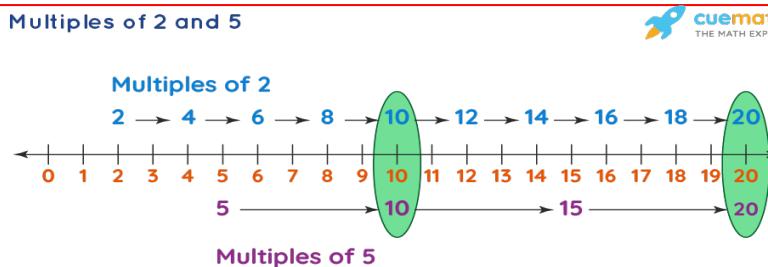
Divisors of 13 = 1, and 13.

Divisors of 48 = 1, 2, 3, 4, 6, 8, 12, 16, 24 and 48.

The common divisor of 13 and 48 is 1.

The greatest common divisor of 13 and 48 is 1.

**Therefore,  $\text{GCD}(13, 48) = 1$ .**



### Finding Greatest Common Divisor by LCM Method

As per the LCM Method for the greatest common divisor, the GCD of two positive integers (a, b) can be calculated, by using the following formula:

$$\text{GCD}(a, b) = \frac{(a \times b)}{\text{LCM}(a, b)}$$

### How to Calculate the GCD using LCM?

Let us see how to calculate the GCD of (a, b) using the LCM method:

- **Step 1:** Find the product of a and b.
- **Step 2:** Find the Least Common Multiple (LCM - <https://www.cuemath.com/numbers/lcm-least-common-multiple/>) of a and b.
- **Step 3:** Divide the product of the numbers by the LCM of the numbers.
- **Step 4:** The obtained value after division is the greatest common divisor of (a, b).

**Example:** Find the greatest common divisor of 15 and 70 using the LCM method.

**Solution:** The greatest common divisor of 15 and 70 can be calculated as follows:

- The product of 15 and 70 is given as,  $15 \times 70$
- The LCM of (15, 70) is 210.
- We know that  $\text{GCD}(a, b) = (a \times b) / \text{LCM}(a, b)$
- $\text{GCD}(15, 70) = (15 \times 70) / 210 = 5$ .

**The greatest common divisor of (15, 70) is 5.**

The Euclidean algorithm is a way to find the greatest common divisor of two positive integers. GCD of two numbers is the largest number that divides both of them. A simple way to find GCD is to factorize both numbers and multiply common prime factors.

$$\begin{aligned} 36 &= 2 \times 2 \times 3 \times 3 \\ 60 &= 2 \times 2 \times 3 \times 5 \end{aligned}$$

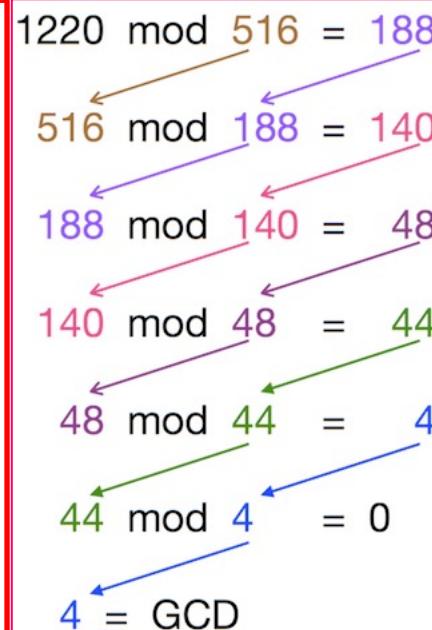
**GCD = Multiplication of common factors**  
 $= 2 \times 2 \times 3$   
 $= 12$

$60 \bmod 36 = 24$

$36 \bmod 24 = 12$

$24 \bmod 12 = 0$

**12 = GCD**



### Problem Solution

1. Take two numbers as input.
2. Find the greater of the two numbers.
3. Keep on dividing the greater number by the smaller number until remainder becomes 0.
4. When the remainder becomes 0 print the smaller number i.e., the denominator as the output.
5. Exit.

```
void main()
{
    int num1, num2, gcd, remainder, numerator, denominator;

    printf("Enter two numbers\n");
    scanf("%d %d", &num1, &num2);
    if (num1 > num2)
    {
        numerator = num1;
        denominator = num2;
    }
    else
    {
        numerator = num2;
        denominator = num1;
    }
    remainder = numerator % denominator;
    while (remainder != 0)
    {
        numerator = denominator;
        denominator = remainder;
        remainder = numerator % denominator;
    }
    gcd = denominator;
    printf("GCD of %d and %d = %d\n", num1, num2, gcd);
}
```

$$A = \begin{pmatrix} 1 & 0 \\ -3 & 4 \end{pmatrix} \quad x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

$$y_0 = a_{00}x_0 + a_{01}x_1 = 1 * 1 + 0 * 2 = 1$$

$$y_1 = a_{10}x_0 + a_{11}x_1 = -3 * 1 + 4 * 2 = 5$$

$$y = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$$

$$\mathbf{C} = \begin{pmatrix} 0 & -2 \\ -3 & 6 \end{pmatrix} A = \begin{pmatrix} 1 & 2 \\ 0 & -3 \end{pmatrix} B = \begin{pmatrix} -2 & 2 \\ 1 & -2 \end{pmatrix}$$

$$\begin{bmatrix} c_{00} & c_{10} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & b_{00} & b_{01} \\ a_{10} & a_{11} & b_{10} & b_{11} \end{bmatrix}$$

$$c_{00} = a_{00}b_{00} + a_{01}b_{10} = 1 * (-2) + 2 * 1 = 0$$

MS Quantum Presentation

# Review: matrix multiplication

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} w & x \\ y & z \end{pmatrix} = \begin{pmatrix} aw + by & ax + bz \\ cw + dy & cx + dz \end{pmatrix}$$

$$a = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \quad b = \begin{bmatrix} -1 \\ 0 \\ 2 \end{bmatrix}$$

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \otimes \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} = \begin{pmatrix} a_{00}[B] & a_{01}[B] \\ a_{10}[B] & a_{11}[B] \end{pmatrix}$$

$$a \otimes b = \begin{bmatrix} 2 & \begin{bmatrix} -1 \\ 0 \\ 2 \\ -1 \end{bmatrix} \\ 3 & \begin{bmatrix} 0 \\ 2 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ 4 \\ -3 \\ 0 \\ 6 \end{bmatrix}$$

$$= \begin{pmatrix} a_{00}b_{00} & a_{00}b_{01} & a_{01}b_{00} & a_{01}b_{01} \\ a_{00}b_{10} & a_{00}b_{11} & a_{01}b_{10} & a_{01}b_{11} \\ a_{10}b_{00} & a_{10}b_{01} & a_{11}b_{00} & a_{11}b_{01} \\ a_{10}b_{10} & a_{10}b_{11} & a_{11}b_{10} & a_{11}b_{11} \end{pmatrix}$$

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}, \quad B = \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix},$$

respectively, then the tensor product of these two matrices is

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \otimes \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{1,2} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \\ a_{2,1} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{2,2} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{bmatrix}.$$

The resultant rank is at most 4, and thus the resultant dimension is 4. Note that *rank* here denotes the [tensor rank](#) i.e. the number of requisite indices (while the [matrix rank](#) counts the number of degrees of freedom in the resulting array). Note  $\text{Tr } A \otimes B = \text{Tr } A \times \text{Tr } B$ .

A [dyadic product](#) is the special case of the tensor product between two vectors of the same dimension.

MS Quantum Presentation

# Review: tensor product of vectors

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \otimes \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} \\ x_1 \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} x_0y_0 \\ x_0y_1 \\ x_1y_0 \\ x_1y_1 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} \otimes \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ 6 \\ 8 \end{pmatrix}$$

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \otimes \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} \otimes \begin{pmatrix} z_0 \\ z_1 \end{pmatrix} = \begin{pmatrix} x_0y_0z_0 \\ x_0y_0z_1 \\ x_0y_1z_0 \\ x_0y_1z_1 \\ x_1y_0z_0 \\ x_1y_0z_1 \\ x_1y_1z_0 \\ x_1y_1z_1 \end{pmatrix}$$

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Eigenvalues and eigenvectors are often introduced to students in the context of linear algebra courses focused on matrices.<sup>[23][24]</sup> Furthermore, linear transformations over a finite-dimensional vector space can be represented using matrices,<sup>[3][4]</sup> which is especially common in numerical and computational applications.<sup>[25]</sup>

Consider  $n$ -dimensional vectors that are formed as a list of  $n$  scalars, such as the three-dimensional vectors

$$\mathbf{x} = \begin{bmatrix} 1 \\ -3 \\ 4 \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} -20 \\ 60 \\ -80 \end{bmatrix}.$$

These vectors are said to be **scalar multiples** of each other, or **parallel** or **collinear**, if there is a scalar  $\lambda$  such that

$$\mathbf{x} = \lambda \mathbf{y}.$$

In this case  $\lambda = -\frac{1}{20}$ .

Now consider the linear transformation of  $n$ -dimensional vectors defined by an  $n$  by  $n$  matrix  $A$ ,

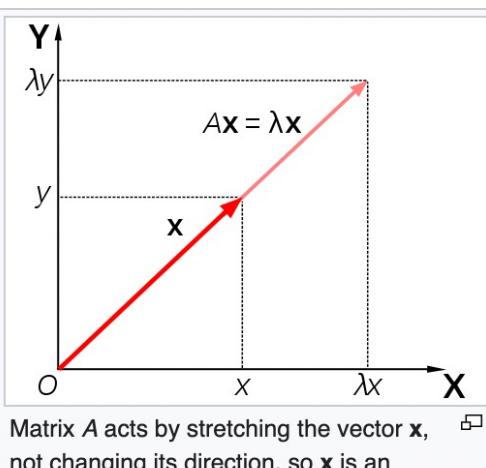
$$A\mathbf{v} = \mathbf{w},$$

or

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

where, for each row,

$$w_i = A_{i1}v_1 + A_{i2}v_2 + \cdots + A_{in}v_n = \sum_{j=1}^n A_{ij}v_j.$$



## Math – Recap Linear Algebra, Tensor, **Eigenvectors & Eigenvalues**

If it occurs that  $v$  and  $w$  are scalar multiples, that is if:

$$A\mathbf{v} = \mathbf{w} = \lambda\mathbf{v}, \quad (1)$$

then  $\mathbf{v}$  is an **eigenvector** of the linear transformation  $A$  and the scale factor  $\lambda$  is the **eigenvalue** corresponding to that eigenvector. Equation (1) is the **eigenvalue equation** for the matrix  $A$ .

Equation (1) can be stated equivalently as:

$$(A - \lambda I)\mathbf{v} = \mathbf{0}, \quad (2)$$

where  $I$  is the  $n$  by  $n$  **identity matrix** and  $\mathbf{0}$  is the zero vector.

## Mathematics – Recap Root, Log/Ln, Complex Numbers, Euler's number e, Polar vs Cartesian, Trigonometry, ....

The roots and logs look at the relation  $a^b=c$  and ask "*If I know one of the numbers on the left (and the number on the right), then what's the other number?*" ... When you know  $b$  (and  $c$ ) and seek  $a$ , you get what we call "the  $b$ -th root of  $c$ "; when you know  $a$  (and  $c$ ) and seek  $b$ , you get what we call "the base- $a$  logarithm of  $c$ ".

It might be best to answer with some examples.

- The number  $\log_7 23$  answers the question  $7^? = 23$ .
- The number  $\sqrt[7]{23}$  answers the question  $?^7 = 23$ .

As you can see, these are different questions.

Some further points:

- When we write  $\log$ , this is short for  $\log_{10}$ . So the number  $\log 23$  answers the question  $10^? = 23$ .
- When we write  $\sqrt{ }$ , this is short for  $\sqrt[2]{ }$ . So the number  $\sqrt{23}$  answers the question  $?^2 = 23$ .

## Mathematics – Recap Log/Ln, Complex Numbers, Euler's number e, Polar vs Cartesian, Trigonometry, ....

In **mathematics**, the logarithm is the **inverse function to exponentiation**. That means the logarithm of a number  $x$  to the **base  $b$**  is the **exponent** to which  $b$  must be raised, to produce  $x$ . For example, since  $1000 = 10^3$ , the **logarithm base 10** of 1000 is 3, or  $\log_{10}(1000) = 3$ . The logarithm of  $x$  to **base  $b$**  is denoted as  $\log_b(x)$ , or without parentheses,  $\log_b x$ , or even without the explicit base,  $\log x$ , when no confusion is possible, or when the base does not matter such as in **big O notation**.

The logarithm base 10 is called the **decimal** or **common logarithm** and is commonly used in science and engineering.

The **natural logarithm** has the number  $e \approx 2.718$  as its base; its use is widespread in mathematics and **physics**, because of its very simple **derivative**.

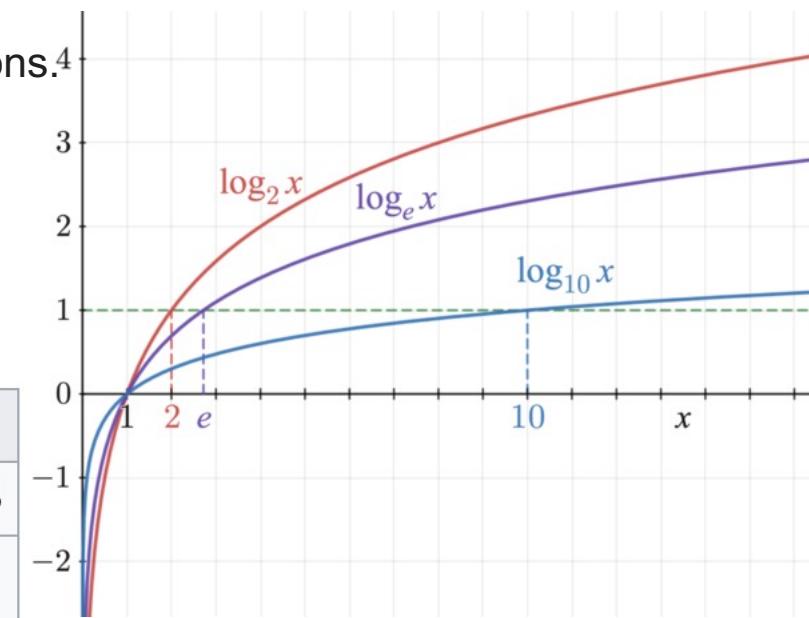
The **binary logarithm** uses base 2 and is frequently used in **computer science**.

Logarithms were introduced by **John Napier** in 1614 as a means of simplifying calculations.<sup>4</sup>

- $\log_2 16 = 4$ , since  $2^4 = 2 \times 2 \times 2 \times 2 = 16$ .
- Logarithms can also be negative:  $\log_2 \frac{1}{2} = -1$  since  $2^{-1} = \frac{1}{2^1} = \frac{1}{2}$ .
- $\log_{10} 150$  is approximately 2.176, which lies between 2 and 3, just as 150 lies between  $10^2 = 100$  and  $10^3 = 1000$ .
- For any base  $b$ ,  $\log_b b = 1$  and  $\log_b 1 = 0$ , since  $b^1 = b$  and  $b^0 = 1$ , respectively.

	<b>Formula</b>	<b>Example</b>
Product	$\log_b(xy) = \log_b x + \log_b y$	$\log_3 243 = \log_3(9 \cdot 27) = \log_3 9 + \log_3 27 = 2 + 3 = 5$
Quotient	$\log_b \frac{x}{y} = \log_b x - \log_b y$	$\log_2 16 = \log_2 \frac{64}{4} = \log_2 64 - \log_2 4 = 6 - 2 = 4$
Power	$\log_b(x^p) = p \log_b x$	$\log_2 64 = \log_2(2^6) = 6 \log_2 2 = 6$
Root	$\log_b \sqrt[p]{x} = \frac{\log_b x}{p}$	$\log_{10} \sqrt{1000} = \frac{1}{2} \log_{10} 1000 = \frac{3}{2} = 1.5$

$$\log_b x = \frac{\log_k x}{\log_k b}. \quad \log_b x = \frac{\log_{10} x}{\log_{10} b} = \frac{\log_e x}{\log_e b}.$$

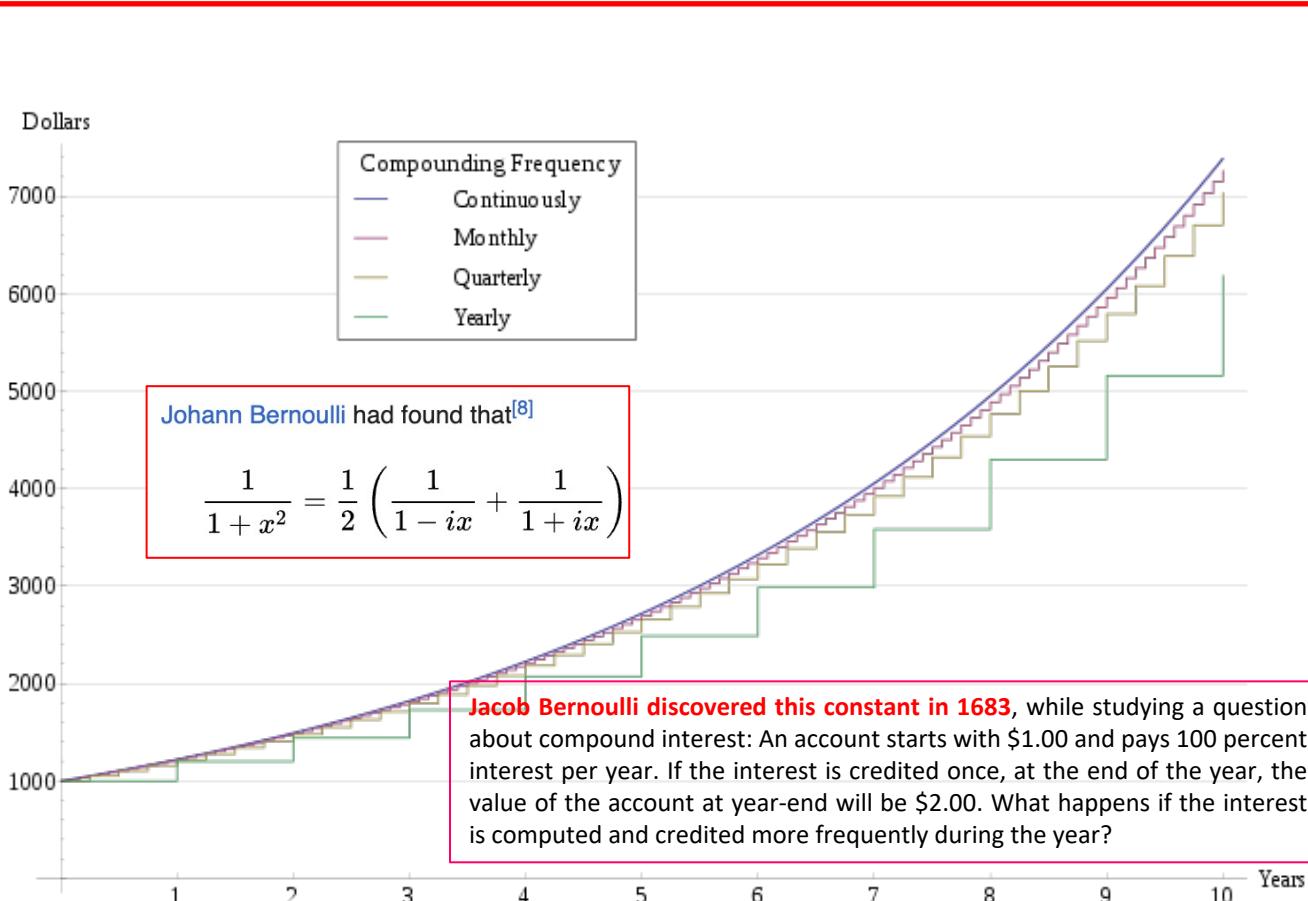


Plots of logarithm functions, with three commonly used bases. The special points  $\log_b b = 1$  are indicated by dotted lines, and all curves intersect in  $\log_b 1 = 0$ .

## Mathematics – Recap Log/Ln, Complex Numbers, Euler's number e, Polar vs Cartesian, Trigonometry, ....

The number **e**, also known as **Euler's number**, is a mathematical constant approximately equal to **2.71828...** that can be characterized in many ways. It is the base of natural logarithms. It is the limit of  $(1 + 1/n)^n$  as  $n$  approaches infinity, an expression that arises in the study of compound interest. It can also be calculated as the sum of the infinite series.

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots, \quad e = \sum_{n=0}^{\infty} \frac{1}{n!} = 1 + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \dots. \quad e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n \quad e = \lim_{t \rightarrow 0} (1+t)^{\frac{1}{t}} = \int_1^e \frac{1}{t} dt = 1.$$



Number of known decimal digits of *e*

Date	Decimal digits	Computation performed by
1690	1	Jacob Bernoulli <sup>[8]</sup>
1714	13	Roger Cotes <sup>[32]</sup>
1748	23	Leonhard Euler <sup>[33]</sup>
1853	137	William Shanks <sup>[34]</sup>
1871	205	William Shanks <sup>[35]</sup>
1884	346	J. Marcus Boorman <sup>[36]</sup>
1949	2,010	John von Neumann (on the ENIAC)
1961	100,265	Daniel Shanks and John Wrench <sup>[37]</sup>
1978	116,000	Steve Wozniak on the Apple II <sup>[38]</sup>

The effect of earning 20% annual interest on an initial \$1,000 investment at various compounding frequencies. The limiting curve on top is the graph  $y = 1000e^{0.2t}$ , where  $y$  is in dollars,  $t$  in years, and  $0.2 = 20\%$ .

Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

## An Intuitive Guide To Exponential Functions & e | Intuitive Guide to Angles, Degrees and Radians

Math books and [Wikipedia](#) describe **e** using obtuse jargon:

***The mathematical constant e is the base of the natural logarithm.***

And when you look up the **natural logarithm** you get:

“The **natural logarithm**, formerly known as the **hyperbolic logarithm**, is the **logarithm to the base e**, where **e** is an irrational constant approximately equal to 2.718281828459.”

**“Nice” circular reference there.**

### e is NOT Just a Number

Describing e as “a constant approximately 2.71828...” is like calling **pi** “an irrational number, approximately equal to 3.1415...”. Sure, it’s true, but you completely missed the point.

**Pi ( $\pi$ ) is the ratio between circumference and diameter shared by all circles.** It is a fundamental ratio inherent in all circles and therefore impacts any calculation of circumference, area, volume, and surface area for circles, spheres, cylinders, and so on. **Pi** is important and shows all circles are related, not to mention the trigonometric functions derived from circles (sin, cos, tan).

**e is the base rate of growth shared by all continually growing processes.** **e** lets you take a simple growth rate (where all change happens at the end of the year) and find the impact of compound, continuous growth, where every nanosecond (or faster) you are growing just a little bit.

**e shows up whenever systems grow exponentially and continuously: population, radioactive decay, interest calculations, and more. Even jagged systems that don't grow smoothly can be approximated by e.**

Just like every number can be considered a scaled version of 1 (the base unit), every circle can be considered a scaled version of the unit circle (radius 1), and every rate of growth can be considered a scaled version of **e** (unit growth, perfectly compounded).

So **e** is not an obscure, seemingly random number.

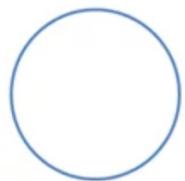
**e represents the idea that all continually growing systems are scaled versions of a common rate.**

Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

## An Intuitive Guide To Exponential Functions & e | Intuitive Guide to Angles, Degrees and Radians

Calculus is similarly enlightening. Don't these formulas seem related in some way?

### Circle and Sphere Fun Facts



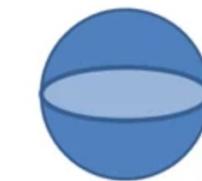
$$2\pi r$$



$$\pi r^2$$



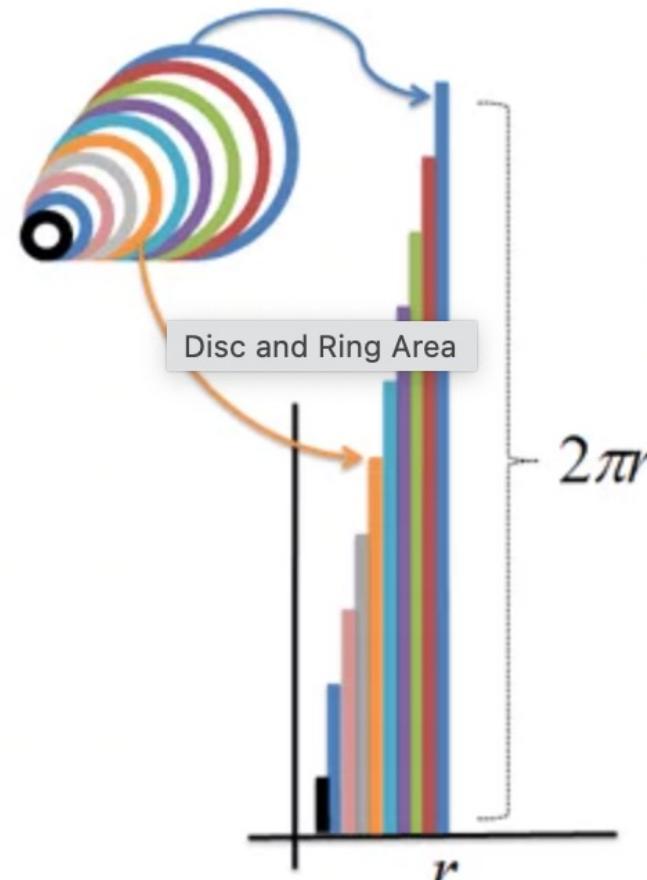
$$4\pi r^2$$



$$\frac{4}{3}\pi r^3$$

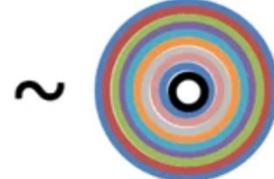
### Unroll the Rings

ALWAYS: Length of circle (Lc) / Diameter (2r) = Pi ( $\pi$ )



$$\frac{1}{2}(r)(2\pi r) = \pi r^2$$

### Dissecting a Circle



Disc

Rings

Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

## An Intuitive Guide To Exponential Functions & e | Intuitive Guide to Angles, Degrees and Radians

### Understanding Exponential Growth – please revisit the example with the chess board from the beginning

Let's start by looking at a basic system that **doubles** after an amount of time. For example,

- Bacteria can split and “doubles” every 24 hours
- We get twice as many noodles when we fold them in half.
- Your money doubles every year if you get 100% return (lucky!)

And it looks like in the pic.

Splitting in two or doubling is a very common progression. Sure, we can triple or quadruple, but doubling is convenient, so hang with me here.

Mathematically, if we have  $x$  splits then we get  $2^x$  times as much stuff than when we started. With 1 split we have  $2^1$  or 2 times as much. With 4 splits we have  $2^4 = 16$  times as much. As a general formula:

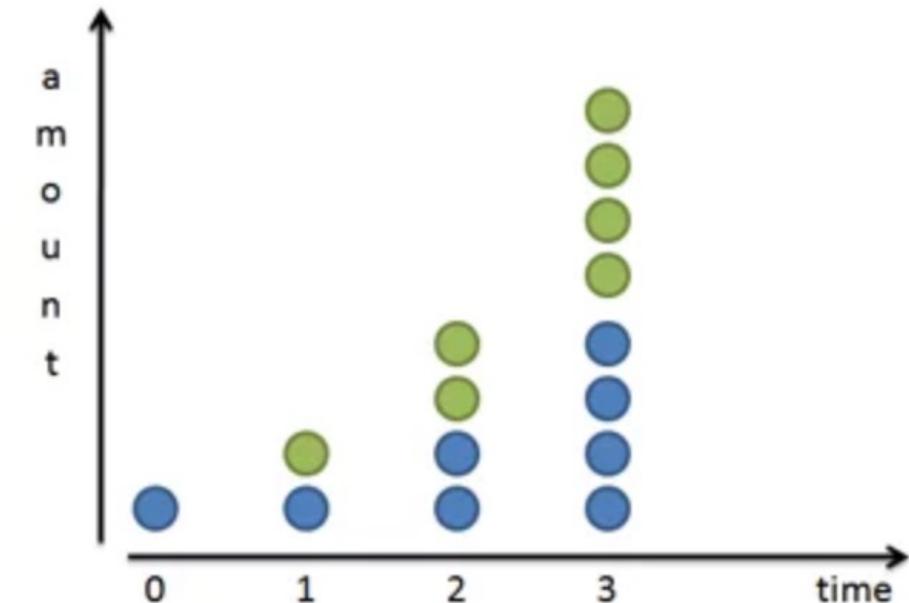
$$\text{growth} = 2^x$$

Said another way, doubling is 100% growth. We can rewrite our formula like this:

$$\text{growth} = (1 + 100\%)^x$$

It's the same equation, but we separate 2 into what it really is: the original value (1) plus 100%. Clever, eh?

<https://betterexplained.com/articles/an-intuitive-guide-to-exponential-functions-e/> | <https://betterexplained.com/articles/intuitive-guide-to-angles-degrees-and-radians/>



Of course, we can substitute any number (50%, 25%, 200%) for 100% and get the growth formula for that new rate. So the general formula for  $x$  periods of return is:

$$\text{growth} = (1 + \text{return})^x$$

This just means we use our rate of return,  $(1 + \text{return})$ , “ $x$ ” times.

Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

## An Intuitive Guide To Exponential Functions & e | Intuitive Guide to Angles, Degrees and Radians

### A Closer Look

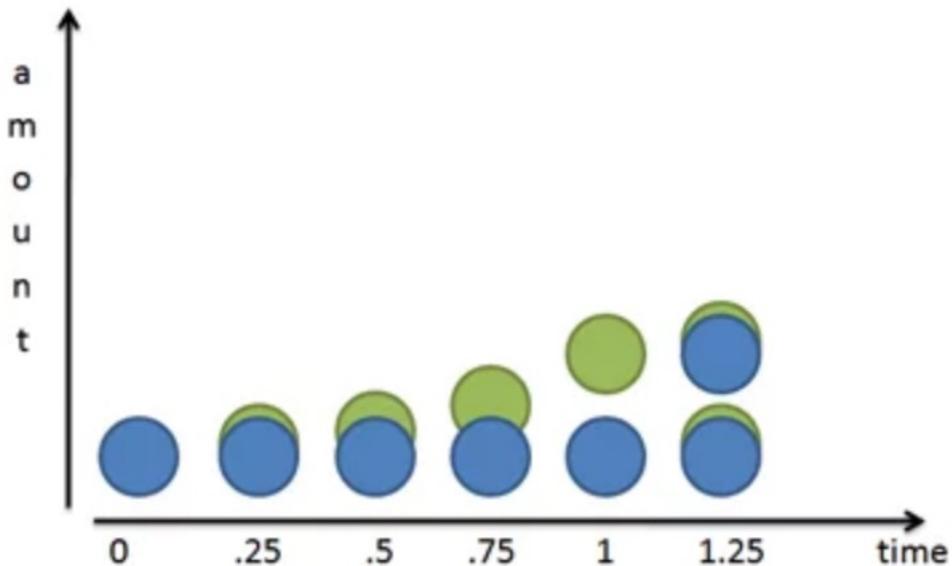
Our formula assumes growth happens in discrete steps. Our bacteria are waiting, waiting, and then **boom**, they double at the very last minute after 24h. Our interest earnings magically appear at the 1 year mark. Based on the formula above, growth is punctuated and happens instantly. The green dots suddenly appear.

The world isn't always like this. If we zoom in, we see that our bacterial friends split over time:

Mr. Green doesn't just show up: he slowly grows out of Mr. Blue. After 1 unit of time (24 hours in our case), Mr. Green is complete. He then becomes a mature blue cell and can create new green cells of his own.

### *Does this information change our equation?*

**Nope.** In the bacteria case, the half-formed green cells still can't do anything until they are fully grown and separated from their blue parents. The equation still holds.

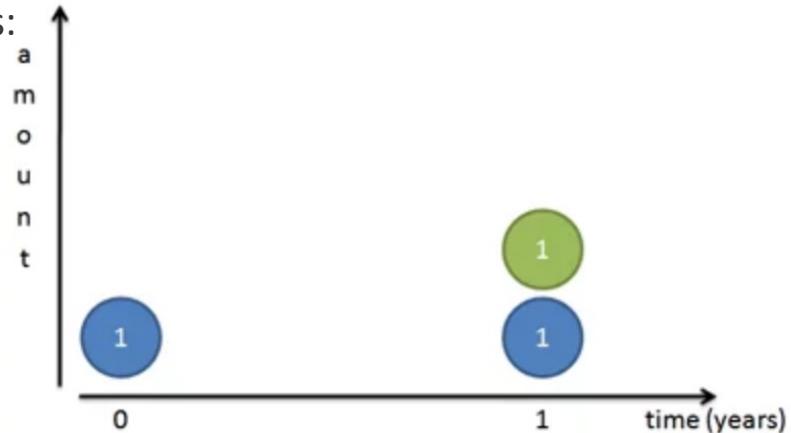


Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

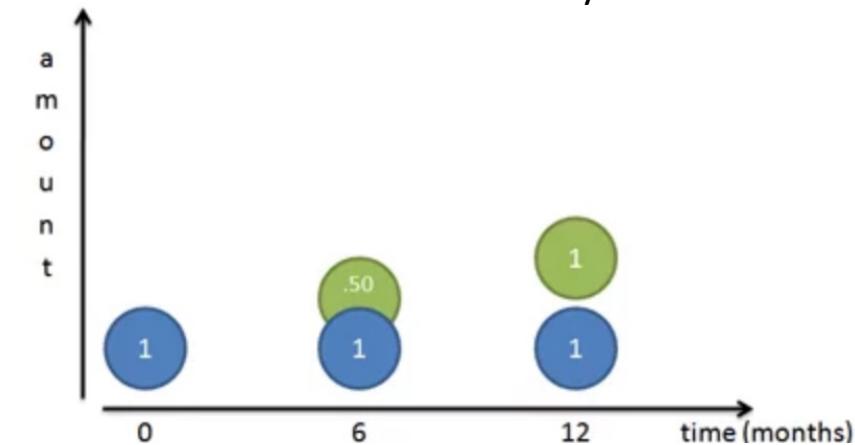
## An Intuitive Guide To Exponential Functions & e | Intuitive Guide to Angles, Degrees and Radians

### Money Changes Everything

1. But money is different. As soon as we earn a penny of interest, that penny can start earning micro-pennies of its own. We don't need to wait until we earn a complete dollar in interest — fresh money doesn't need to mature. Based on our *old formula*, interest growth looks like this:



2. But again, this isn't quite right: all the interest appears on the last day. Let's zoom in and split the year into two chunks. We earn 100% interest every year, or 50% every 6 months. So, we earn 50 cents the first 6 months and another 50 cents in the last half of the year.



3. But this still isn't right! Sure, our original dollar (Mr. Blue) earns a dollar over the course of a year. But after 6 months we had a 50-cent piece, ready to go, that we neglected! That 50 cents could have earned money on its own.

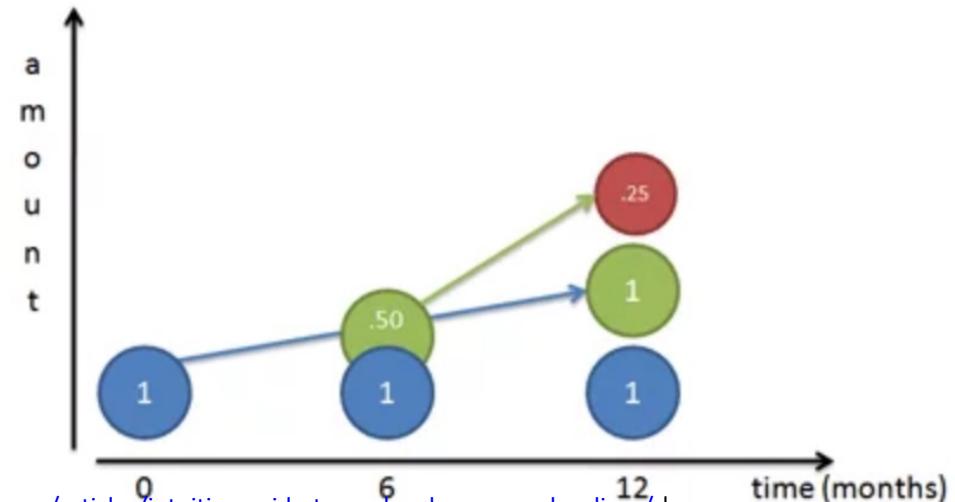
Because our rate is 50% per half year, that 50 cents would have earned 25 cents (50% times 50 cents). At the end of 1 year we'd have

- Our original dollar (Mr. Blue)
- The dollar Mr. Blue made (Mr. Green)
- The 25 cents Mr. Green made (Mr. Red)

$$\text{growth} = (1 + 100\%/2)^2 = 2.25$$

Giving us a total of \$2.25. We gained \$1.25 from our initial dollar, even better than doubling!

Let's turn our return into a formula. The growth of two half-periods of 50% is.



Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

## An Intuitive Guide To Exponential Functions & e | Intuitive Guide to Angles, Degrees and Radians

### Diving into Compound Growth

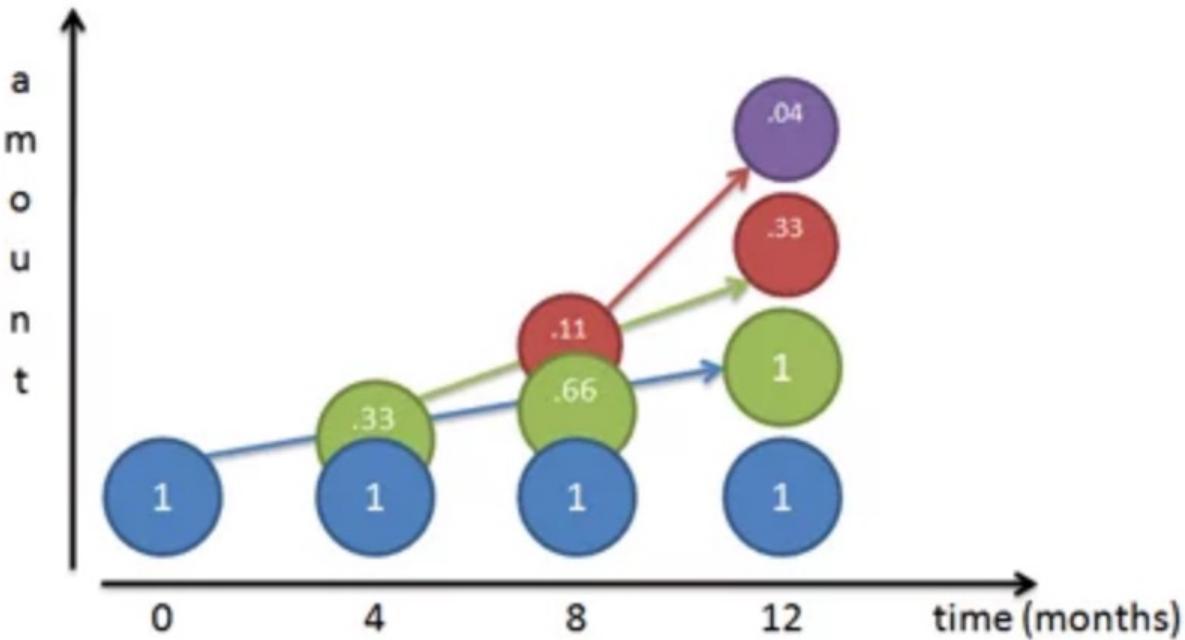
It's time to step it up a notch. Instead of splitting growth into two periods of 50% increase, let's split it into 3 segments of 33% growth. Who says we have to wait for 6 months before we start getting interest? Let's get more granular in our counting.

Charting our growth for 3 compounded periods gives a funny picture:

Think of each color as shoveling money upwards towards the other colors (its children), at 33% per period:

- **Month 0:** We start with Mr. Blue at \$1.
- **Month 4:** Mr. Blue has earned 1/3 dollar on himself, and creates Mr. Green, shoveling along 33 cents.
- **Month 8:** Mr. Blue earns another 33 cents and gives it to Mr. Green, bringing Mr. Green up to 66 cents. Mr. Green has actually earned 33% on his previous value, creating 11 cents ( $33\% * 33$  cents). This 11 cents becomes Mr. Red.
- **Month 12:** Things get a bit crazy. Mr. Blue earns another 33 cents and shovels it to Mr. Green, bringing Mr. Green to a full dollar. Mr. Green earns 33% return on his Month 8 value (66 cents), earning 22 cents. This 22 cents gets added to Mr. Red, who now totals 33 cents. And Mr. Red, who started at 11 cents, has earned 4 cents ( $33\% * .11$ ) on his own, creating Mr. Purple.

Phew! The final value after 12 months is:  $1 + 1 + .33 + .04$  or about 2.37.



We get a formula by using 3 periods in our growth equation:

$$\text{growth} = (1 + 100\%/3)^3 = 2.37037\dots$$

We earned \$1.37, even better than the \$1.25 we got last time!

Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

## An Intuitive Guide To Exponential Functions & e | Intuitive Guide to Angles, Degrees and Radians

### Can We Get Infinite Money?

Why not take even shorter time periods? How about every month, day, hour, or even nanosecond? Will our returns skyrocket?

Our return gets better, but only to a point. Try using different numbers of n in our magic formula to see our total return:

n	$(1 + 1/n)^n$
1	2
2	2.25
3	2.37
5	2.488
10	2.5937
100	2.7048
1,000	2.7169
10,000	2.71814
100,000	2.718268
1,000,000	2.7182804
...	

### Computing $e^x$

try a few values of n

n 1000000

estimate 2.71828047

true value 2.71828183

accuracy 99.99994999 %

POWERED BY **INSTACALC**

<https://instacalc.com/50446>

The numbers get bigger and converge around 2.718. Hey... wait a minute... that looks like e!

Yowza. In geeky math terms, e is **defined** to be that rate of growth if we continually compound 100% return on smaller and smaller time periods:

$$\text{perfect compound growth} = e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n$$

This limit appears to converge, and there [are proofs](#) to that effect. But as you can see, as we take finer time periods the total return stays around 2.718.

**E (Compound Interest Definition) = 2.7182...**

## E (Compound Interest Definition)

$$e = \lim_{n \rightarrow \infty} \left( 1 + \frac{1}{n} \right)^{1 \cdot n}$$

The base for continuous growth is  
the unit quantity earning unit interest for unit time,  
compounded as fast as possible

### Plain English

- **What is e?** A constant (2.718...) representing *continuous unit growth*: the unit quantity (1.0), continuously growing the unit rate (100%), for unit time (1 period).
- **Why's e special?** All circles are the unit circle, scaled up. All continuously growing systems are  $e^{\{rt\}}$ , scaled to some rate and time.
- **When should I use e?** Use  $e^{\{rt\}}$  for things that change constantly (radioactive decay, populations). For growth based on discrete intervals (interest payments, combinatorics),  $(1 + \text{rate})^{\{\text{time}\}}$  is a better model.

# The faces of $e$

**E (Compound Interest Definition) = 2.7182...**

**Definition 1: Compound Interest Perfectly**

Discovered circa 1700

$$\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n$$

Max compounded growth

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots$$

Infinite Series

$e$

$$\ln(a) = \int_1^a \frac{1}{x} dx$$

$$\int_1^e \frac{1}{t} dt = 1$$

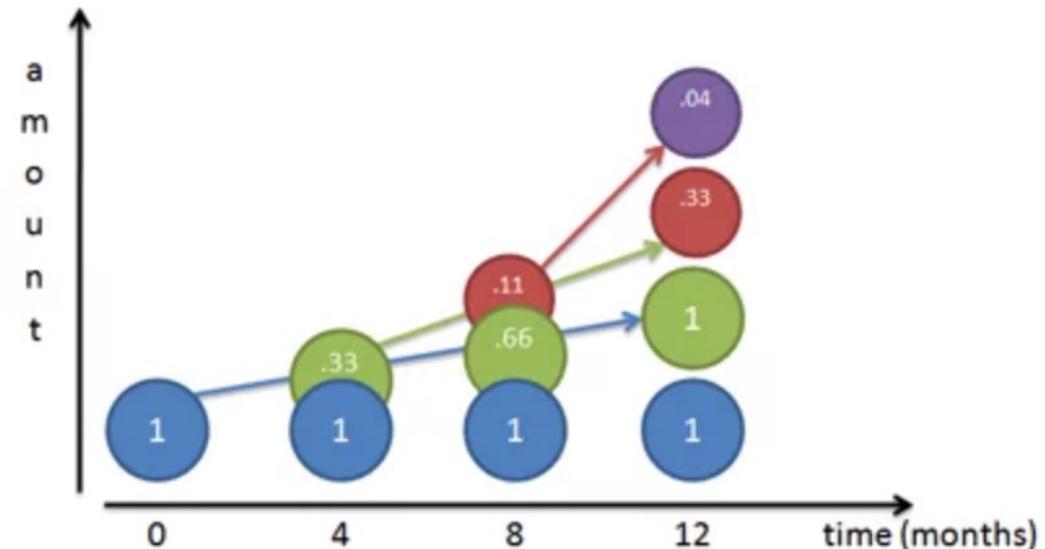
Inverse of  $\ln(x)$

$$\frac{d}{dx} e^x = e^x.$$

Derivative is exponent

$$e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^{1 \cdot n}$$

The base for continuous growth is the unit quantity earning unit interest for unit time, compounded as fast as possible



# The faces of $e$

**E (Compound Interest Definition) = 2.7182...**

**Definition 2: Track Each Interest Contribution**

Discovered circa 1700

$$\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n$$

Max compounded growth

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots$$

Infinite Series

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

The base for continuous growth is the unit quantity earning unit interest plus the interest on the interest

plus the interest on the interest on the interest and so on

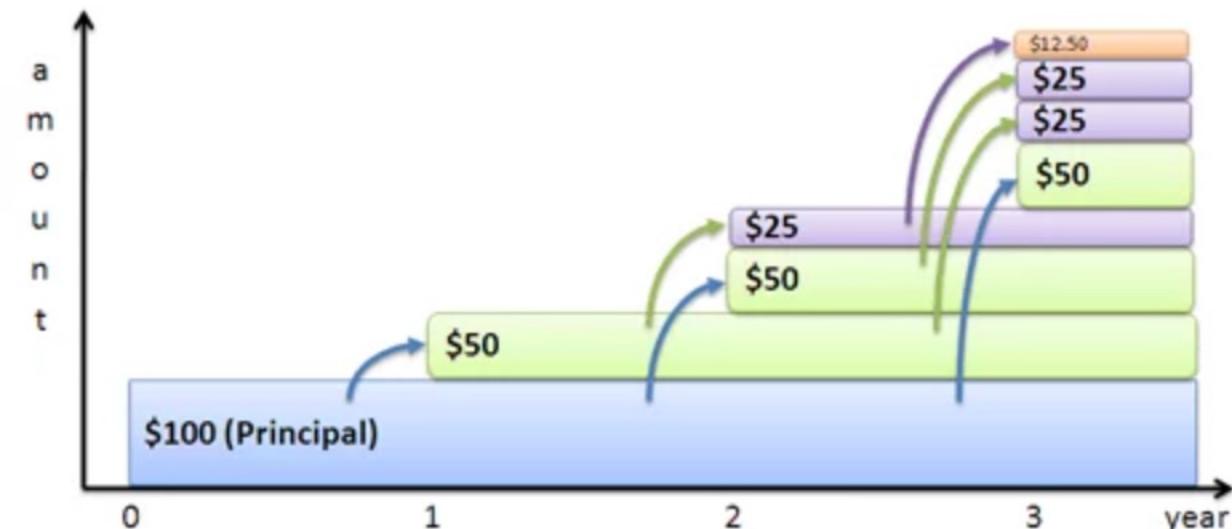
$$\ln(a) = \int_1^a \frac{1}{x} dx$$

$$\int_1^e \frac{1}{t} dt = 1$$

Inverse of  $\ln(x)$

Derivative is exponent

$$\frac{d}{dx} e^x = e^x.$$



Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

## A Visual, Intuitive Guide to Imaginary Numbers & Intuitive Guide to Angles, Degrees and Radians

Fun Fact	Negative Numbers (-x)	Complex Numbers (a +bi)
Invented to answer	"What is $3 - 4$ ?"	"What is $\sqrt{-1}$ ?"
Strange because...	<i>How can you have less than nothing?</i>	<i>How can you take the square root of less than nothing?</i>
Intuitive meaning	"Opposite"	"Rotation"
Considered absurd until	1700s	Today ☺
Multiplication cycle [& general pattern]	$1, -1, 1, -1\dots$ $X, -X, X, -X\dots$	$1, i, -1, -i\dots$ $X, Y, -X, -Y\dots$
Use in coordinates	Go backwards from origin	Rotate around origin
Measure size with	Absolute value $\sqrt{(-x)^2}$	Pythagorean Theorem: $\sqrt{a^2 + b^2}$

### Really Understanding Negative Numbers

Negative numbers aren't easy. Imagine you're a European mathematician in the 1700s. You have 3 and 4, and know you can write  $4 - 3 = 1$ . Simple.

But what about  $3 - 4$ ? What, exactly, does that mean? How can you take 4 cows from 3? *How could you have less than nothing?*

Negatives were considered absurd, something that "darkened the very whole doctrines of the equations" ([Francis Maseres, 1759](#)). Yet today, it'd be absurd to think negatives aren't logical or useful. Try asking your teacher whether negatives corrupt the very foundations of math.

What happened? We invented a *theoretical number that had useful properties*. Negatives aren't something we can touch or hold, but they describe certain relationships well (like debt). It was a useful fiction.

Rather than saying "I owe you 30" and reading words to see if I'm up or down, I can write "-30" and know it means I'm in the hole. If I earn money and pay my debts  $(-30 + 100 = 70)$ , I can record the transaction easily. I have +70 afterwards, which means I'm in the clear.

Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

## A Visual, Intuitive Guide to Imaginary Numbers & Intuitive Guide to Angles, Degrees and Radians

### Visual Understanding of Negative and Complex Numbers

As we saw last time, the equation  $x^2 = 9$  really means:

$$1 \cdot x^2 = 9$$

or

$$1 \cdot x \cdot x = 9$$

What transformation  $x$ , when applied twice, turns 1 to 9?

The two answers are “ $x = 3$ ” and “ $x = -3$ ”: That is, you can “scale by” 3 or “scale by 3 and flip” (flipping or taking the opposite is one interpretation of multiplying by a negative).

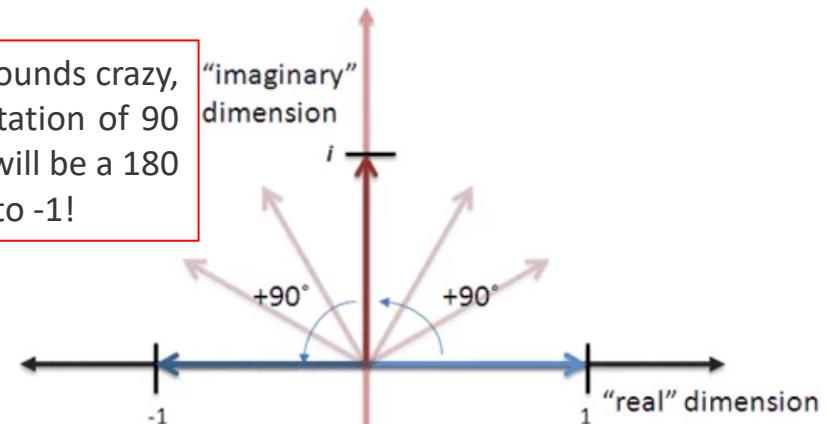
Now let's think about  $x^2 = -1$ , which is really

$$1 \cdot x \cdot x = -1$$

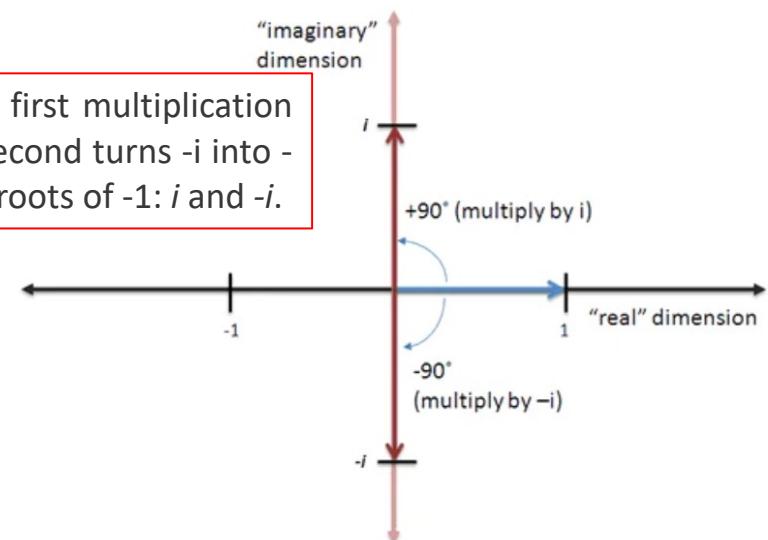
What transformation  $x$ , when applied twice, turns 1 into -1? Hrm.

- We can't multiply by a positive twice, because the result stays positive
- We can't multiply by a negative twice, because the result will flip back to positive on the second multiplication

### Rotate 1 to -1



### Positive & Negative Rotation



If we multiply by  $-i$  twice, the first multiplication would turn 1 into  $-i$ , and the second turns  $-i$  into  $-1$ . So there's really two square roots of  $-1$ :  $i$  and  $-i$ .

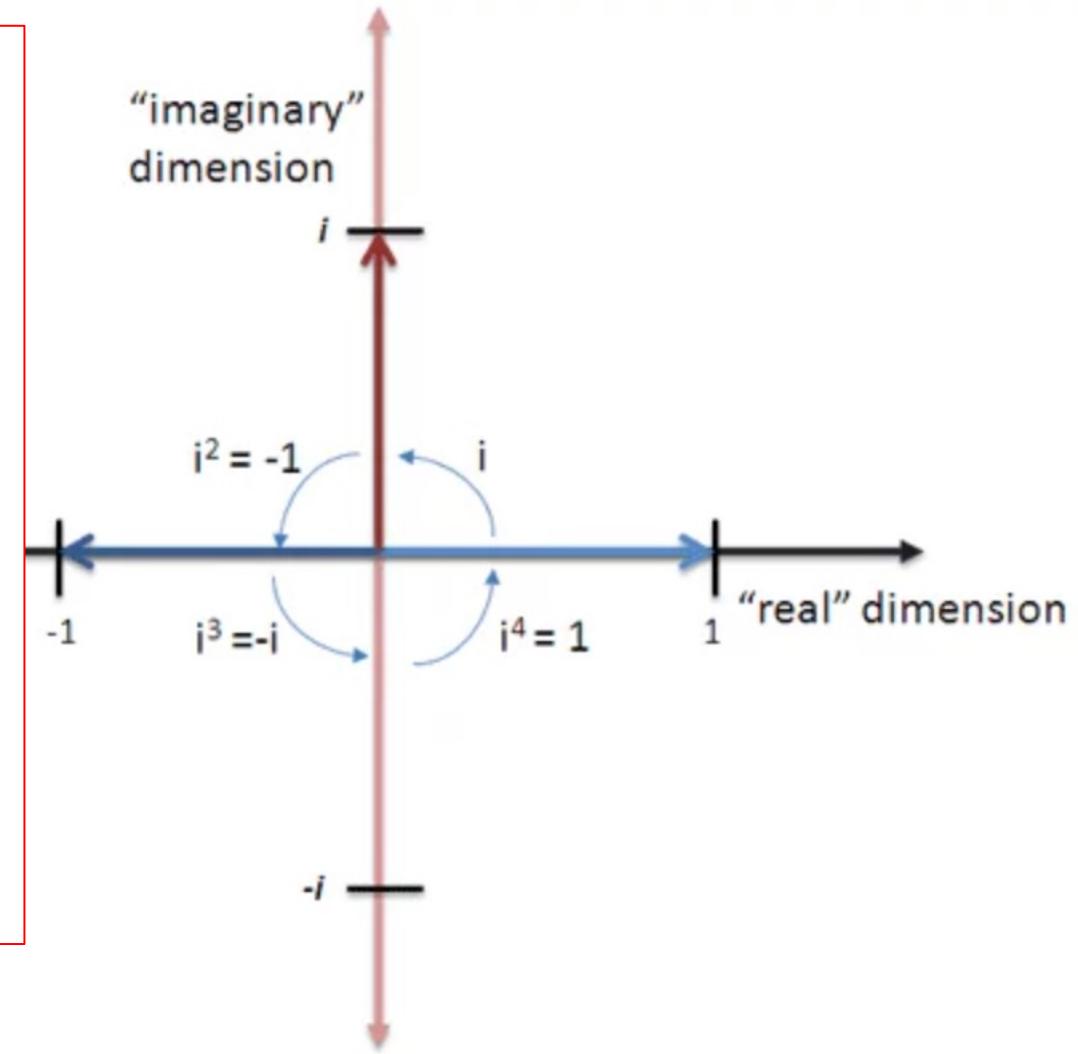
## A Visual, Intuitive Guide to Imaginary Numbers & Intuitive Guide to Angles, Degrees and Radians

Ok. Now what happens if we keep multiplying by  $i$ ?

$$1, i, i^2, i^3, i^4, i^5$$

Very funny. Let's reduce this a bit:

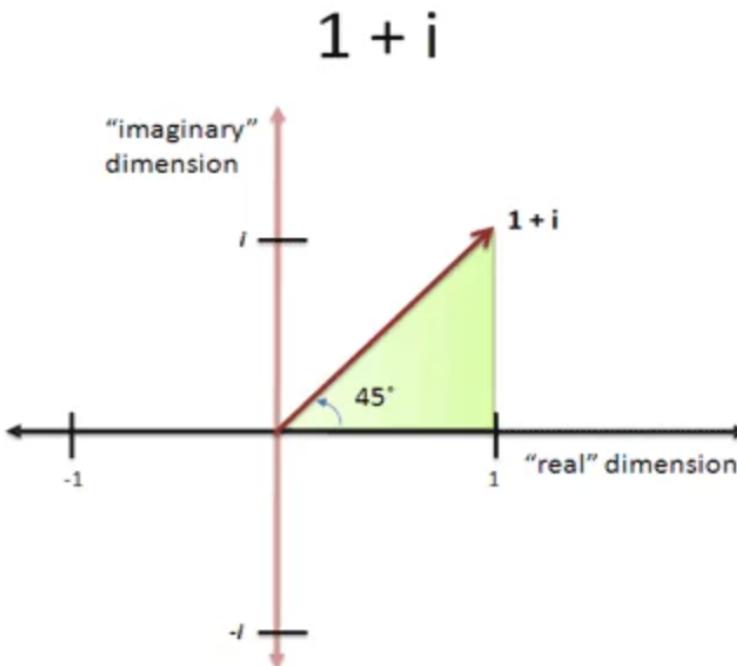
- $1 = 1$  (No questions here)
- $i = i$  (Can't do much)
- $i^2 = -1$  (That's what  $i$  is all about)
- $i^3 = (i \cdot i) \cdot i = -1 \cdot i = -i$  (Ah, 3 rotations counter-clockwise = 1 rotation clockwise. Neat.)
- $i^4 = (i \cdot i) \cdot (i \cdot i) = -1 \cdot -1 = 1$  (4 rotations bring us "full circle")
- $i^5 = i^4 \cdot i = 1 \cdot i = i$  (Here we go again...)



## Understanding Complex Numbers

There's another detail to cover: can a number be both "real" and "imaginary"?

You bet. Who says we have to rotate the entire 90 degrees? If we keep 1 foot in the "real" dimension and another in the imaginary one, it looks like this:



We're at a 45 degree angle, with equal parts in the real and imaginary ( $1 + i$ ). It's

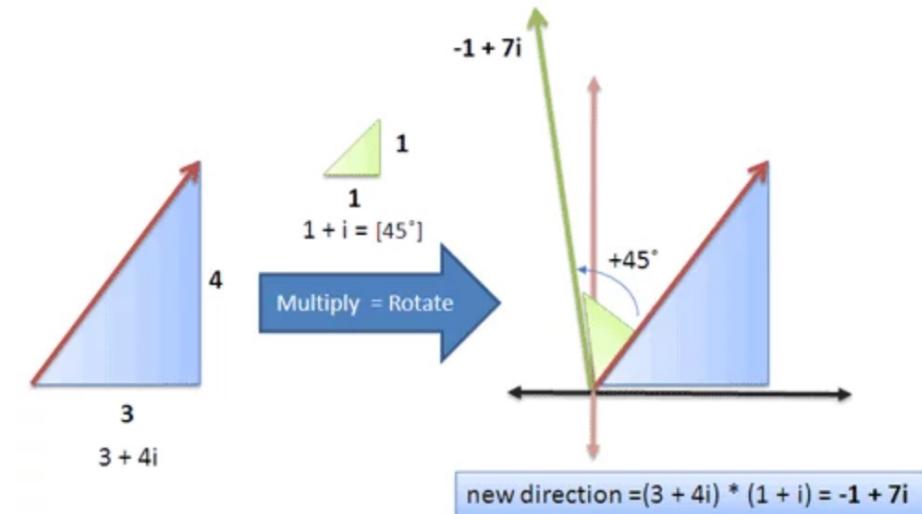
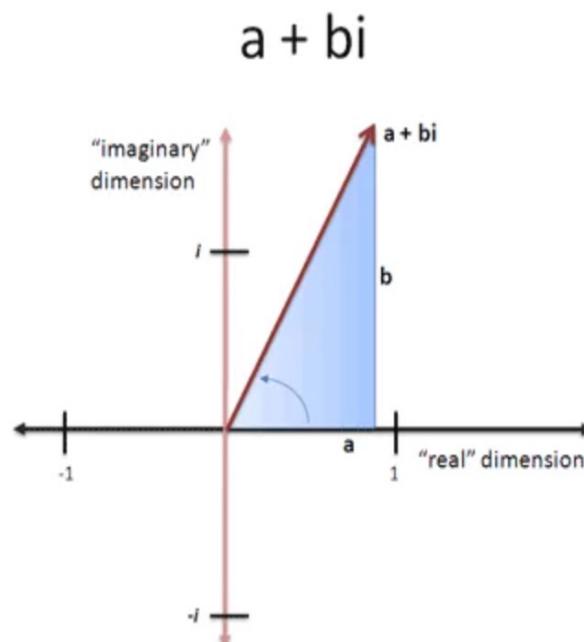
# A Visual, Intuitive Guide to Imaginary Numbers & Intuitive Guide to Angles, Degrees and Radians

## Applying Complex Numbers

In fact, we can pick any combination of real and imaginary numbers and make a triangle. The angle becomes the “angle of rotation”. A complex number is the fancy name for numbers with both real and imaginary parts. They’re written  $a + bi$ , where

- $a$  is the real part
- $b$  is the imaginary part

### Complex Number Addition

$$S = (2+2i) + (2 - 2i) = (4 + 0i)$$


Here's the idea:

- Original heading: 3 units East, 4 units North =  $3 + 4i$
- Rotate counter-clockwise by 45 degrees = multiply by  $1 + i$ . (Here's why multiplication, not addition, performs the rotation.)

If we multiply them together we get:

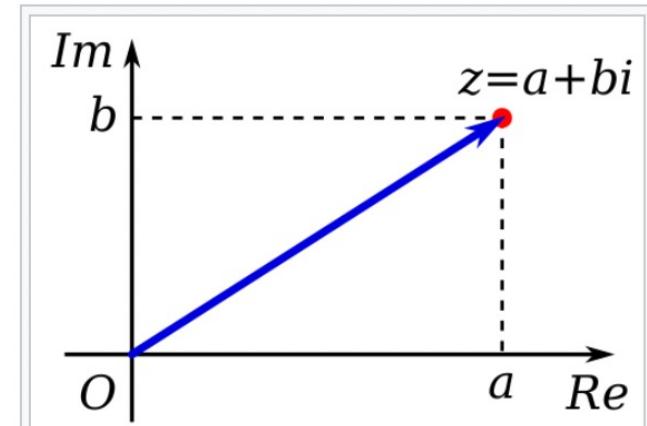
$$\begin{aligned}(3 + 4i) \cdot (1 + i) &= 3 + 3i + 4i + 4i^2 \\&= 3 + 7i + 4(-1) \\&= -1 + 7i\end{aligned}$$

## Mathematics – Recap Log/Ln, Complex Numbers, Euler's number e, Polar vs Cartesian, Trigonometry, ....

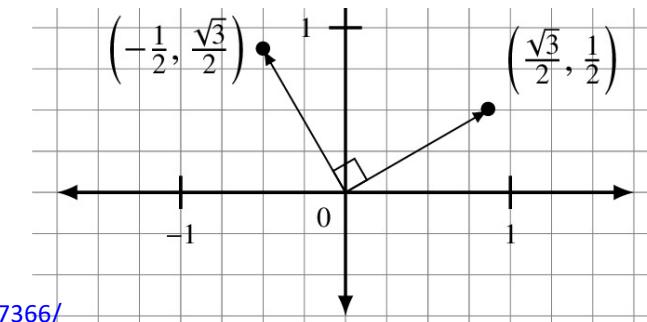
In mathematics, a **complex number** is an element of a **number system** that extends the **real numbers** with a specific element denoted  $i$ , called the **imaginary unit** and satisfying the **equation**  $i^2 = -1$ ; every complex number can be expressed in the form  $a + bi$ , where  $a$  and  $b$  are real numbers. Because no real number satisfies the above equation,  $i$  was called an **imaginary number** by René Descartes. For the complex number  $a + bi$ ,  $a$  is called the **real part**, and  $b$  is called the **imaginary part**. The set of complex numbers is denoted by either of the symbols  $\mathbb{C}$  or  $\mathbf{C}$ . Despite the historical nomenclature "imaginary", complex numbers are regarded in the **mathematical sciences** as just as "real" as the real numbers and are fundamental in many aspects of the scientific description of the natural world.<sup>[1][a]</sup>

Complex numbers allow solutions to all **polynomial equations**, even those that have no solutions in real numbers. More precisely, the **fundamental theorem of algebra** asserts that every non-constant polynomial equation with real or complex coefficients has a solution which is a complex number. For example, the equation  $(x + 1)^2 = -9$  has no real solution, since the square of a real number cannot be negative, but has the two nonreal complex solutions  $-1 + 3i$  and  $-1 - 3i$ .

Addition, subtraction and multiplication of complex numbers can be naturally defined by using the rule  $i^2 = -1$  combined with the **associative**, **commutative**, and **distributive laws**. Every nonzero complex number has a **multiplicative inverse**. This makes the complex numbers a **field** that has the real numbers as a subfield. The complex numbers also form a **real vector space** of dimension two, with  $\{1, i\}$  as a **standard basis**.



A complex number can be visually represented as a pair of numbers  $(a, b)$  forming a vector on a diagram called an **Argand diagram**, representing the **complex plane**.  $Re$  is the real axis,  $Im$  is the imaginary axis, and  $i$  is the "**imaginary unit**", that satisfies  $i^2 = -1$ .

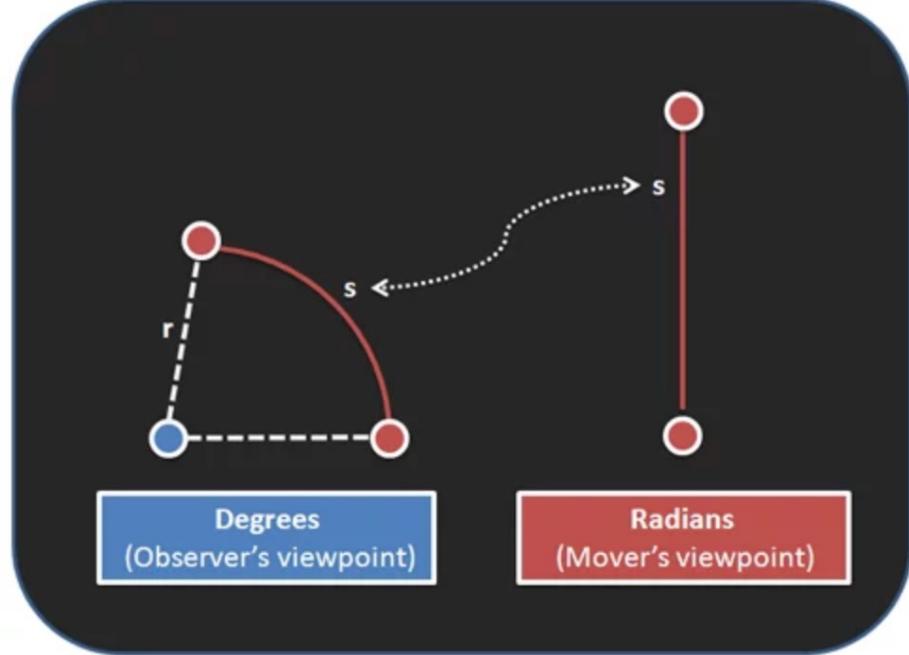


Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

## A Visual, Intuitive Guide to Imaginary Numbers & Intuitive Guide to Angles, Degrees and Radians

### Radians: The Unselfish Choice

Much of physics (and life!) involves leaving your reference frame and seeing things from another's viewpoint. Instead of wondering how far we tilted our heads, **consider how far the other person moved**.



Degrees measure angles by how far we tilted our heads. Radians measure angles by **distance traveled**.

But absolute distance isn't that useful, since going 10 miles is a different number of laps depending on the track. So we divide by radius to get a normalized angle:

$$\text{Radian} = \frac{\text{distance traveled}}{\text{radius}}$$

You'll often see this as

$$\theta = \frac{s}{r}$$

or angle in radians ( $\theta$ ) is arc length ( $s$ ) divided by radius ( $r$ ).

A circle has 360 degrees or  $2\pi$  radians — going all the way around is  $2 * \pi * r / r$ . So a radian is about  $360 / (2 * \pi)$  or 57.3 degrees.

Now don't be like me, memorizing this thinking "Great, another unit. 57.3 degrees is so weird." Because it is weird when you're still thinking about you!

Moving 1 radian (unit) is a perfectly normal distance to travel. Put another way, our idea of a "clean, 90 degree angle" means the mover goes a very *unclean*  $\pi/2$  units. Think about it — "Hey Bill, can you run 90 degrees for me? What's that? Oh, yeah, that'd be  $\pi/2$  miles from your point of view." The strangeness goes both ways.

Radians are the empathetic way to do math — a shift from away from head tilting and towards the mover's perspective.

## A Visual, Intuitive Guide to Imaginary Numbers & Intuitive Guide to Angles, Degrees and Radians

### Using Radians

I'm still getting used to thinking in radians. But we encounter the concept of "mover's distance" quite a bit:

- We use “rotations per minute” not “degrees per second” when measuring certain rotational speeds. This is a shift towards the mover’s reference point (“How many laps has it gone?”) and away from an arbitrary degree measure.
- When a satellite orbits the Earth, we understand its speed in “miles per hour”, not “degrees per hour”. Now divide by the distance to the satellite and you get the orbital speed in radians per hour.
- Sine, that wonderful function, is defined in terms of *radians* as

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \dots$$

## A Visual, Intuitive Guide to Imaginary Numbers & Intuitive Guide to Angles, Degrees and Radians

### Radian Example 1: Wheels of the Bus

Let's try a real example: you have a bus with wheels of radius 2 meters (it's a monster truck bus). I'll say how fast the wheels are turning and you say how fast the bus is moving. Ready?

"The wheels are turning 2000 degrees per second". You'd think:

- Ok, the wheels are going 2000 degrees per second. That means it's turning  $2000/360$  or  $5$  and  $5/9$ ths rotations per second. Circumference =  $2 * \pi * r$ , so it's moving, um,  $2 * 3.14 * 5$  and  $5/9$ ths... where's my [calculator](#)... Ok. Now imagine a car with wheels of radius 2 meters (also a monster). "The car wheels are turning 6 radians per second". You'd think:
  - Radians are distance along a unit circle — we just scale by the real radius to see how far we've gone.  $6 * 2 = 12$  meters per second. Next question.

Wow -- the car was easier to figure out than the bus! No crazy formulas, no [Pi](#) floating around — just *multiply* to convert rotational speed to linear speed. All because radians speak in terms of the mover.

The reverse is easy too. Suppose you're cruising 90 feet per second on the highway (60 miles per hour) on your 24" inch rims (radius 1 foot). How fast are the wheels turning?

Well, 90 feet per second / 1 foot radius = 90 radians per second.

## A Visual, Intuitive Guide to Imaginary Numbers & Intuitive Guide to Angles, Degrees and Radians

### Radian Example 2: $\sin(x)$

Time for a beefier example. Calculus is [about many things](#), and one is what happens when numbers get really big or really small.

Choose a number of degrees ( $x$ ). and put  $\sin(x)$  into your calculator:

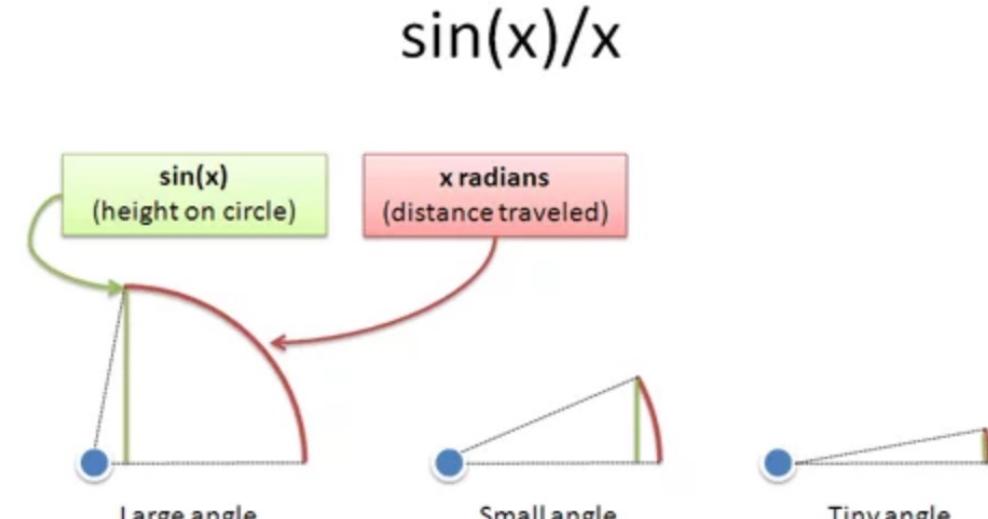


When you make  $x$  small, like .01,  $\sin(x)$  gets small as well. And the ratio of  $\sin(x)/x$  seems to be about .017 — what does that mean? Even stranger, what does it mean to multiply or divide by a degree? Can you have square or cubic degrees?

Radians to the rescue! Knowing they refer to distance traveled (they're not just a ratio!), we can interpret the equation this way:

- $x$  is how far you traveled along a circle
- $\sin(x)$  is how high on the circle you are

So  $\sin(x)/x$  is the ratio of how high you are to how far you've gone: the amount of energy that went in an “upward” direction. If you move vertically, that ratio is 100%. If you move horizontally, that ratio is 0%.



When something moves a tiny amount, such as 0 to 1 degree from our perspective, it's basically going straight up. If you go an even smaller amount, from 0 to .00001 degrees, it's *really* going straight up. The distance traveled ( $x$ ) is very close to the height ( $\sin(x)$ ). As  $x$  shrinks, the ratio gets closer to 100% — more motion is straight up. Radians help us see, intuitively, why  $\sin(x)/x$  approaches 1 as  $x$  gets tiny. We're just nudging along a tiny amount in a vertical direction. By the way, this also explains why  $\sin(x) \sim x$  for small numbers. Sure, you can rigorously *prove* this using calculus, but the radian intuition helps you *understand* it.

Remember, these relationships only work when measuring angles with radians. With degrees, you're comparing your height on a circle ( $\sin(x)$ ) with how far some observer tilted their head ( $x$  degrees), and it gets ugly fast.

Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

## A Visual, Intuitive Guide to Imaginary Numbers & Intuitive Guide to Angles, Degrees and Radians

### So What's the Point?

Degrees have their place: in our own lives, we're the focal point and want to see how things affect us.

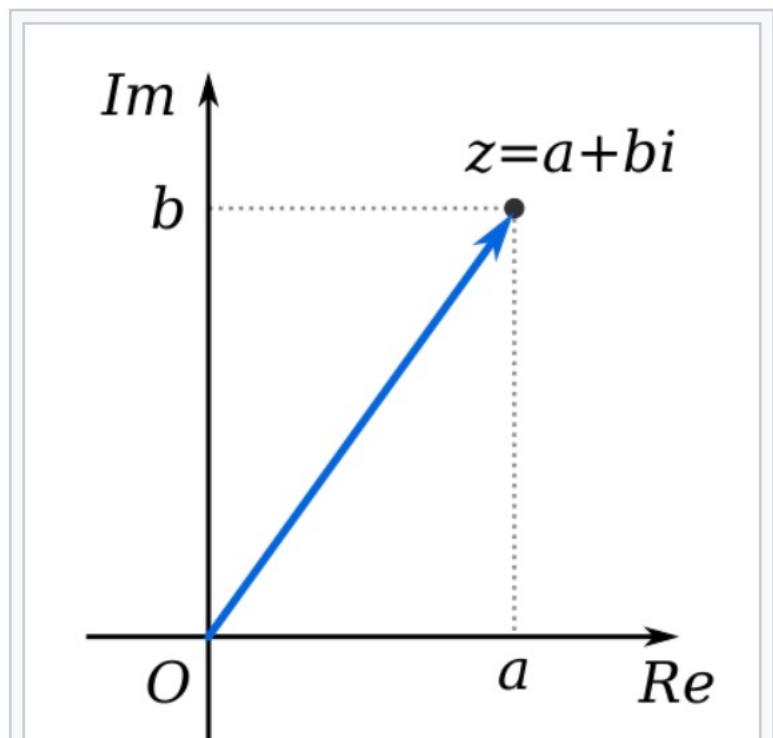
*How much do I tilt my telescope, spin my snowboard, or turn my steering wheel?*

With natural laws, we're an observer describing the motion of others. Radians are about them, not us. PLEASE realize that:

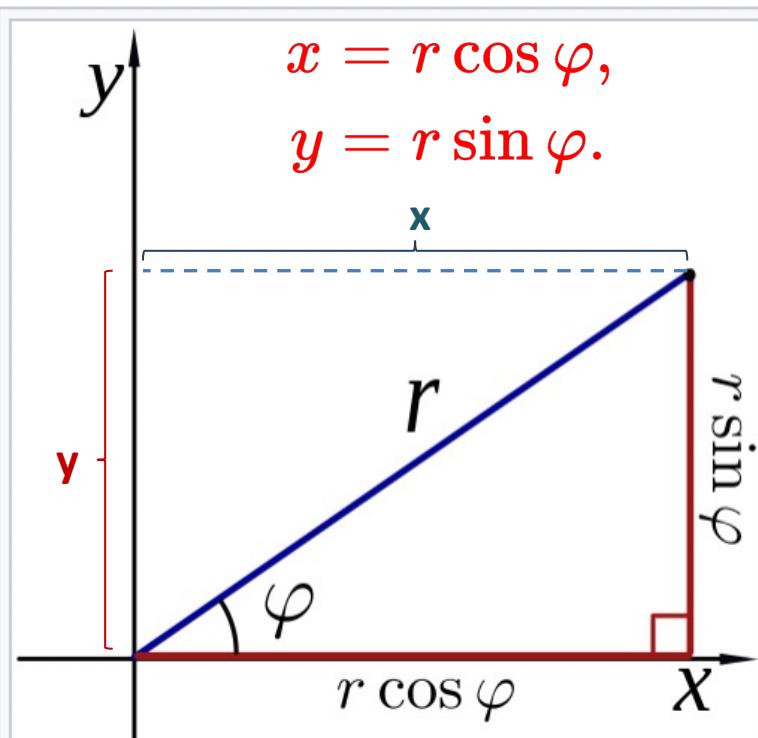
- Degrees are *arbitrary* because they're based on the sun (365 days  $\sim$  360 degrees), but they are *backwards* because they are from the observer's perspective.
- Because radians are in terms of the mover, equations "click into place". Converting rotational to linear speed is easy, and ideas like  $\sin(x)/x$  make sense.

Even angles can be seen from more than one viewpoint, and understanding radians makes math and physics equations more intuitive. Happy math.

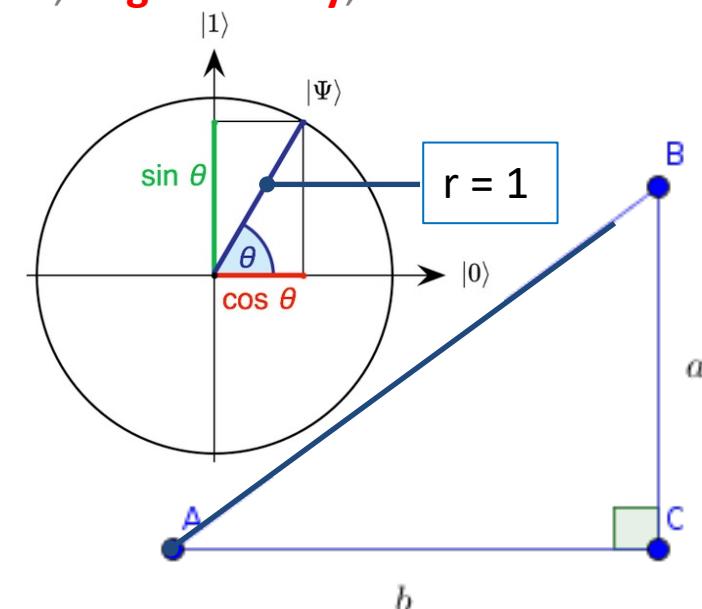
## Mathematics – Recap Log/Ln, Complex Numbers, Euler's number e, Polar vs Cartesian, Trigonometry, ....



A complex number  $z$ , as a point (black) and its position vector (blue)



A diagram illustrating the relationship between polar and Cartesian coordinates.



$$\sin A = \frac{a}{c} \quad \cos A = \frac{b}{c} \quad \tan A = \frac{a}{b}$$

$$\csc A = \frac{c}{a} \quad \sec A = \frac{c}{b} \quad \cot A = \frac{b}{a}$$

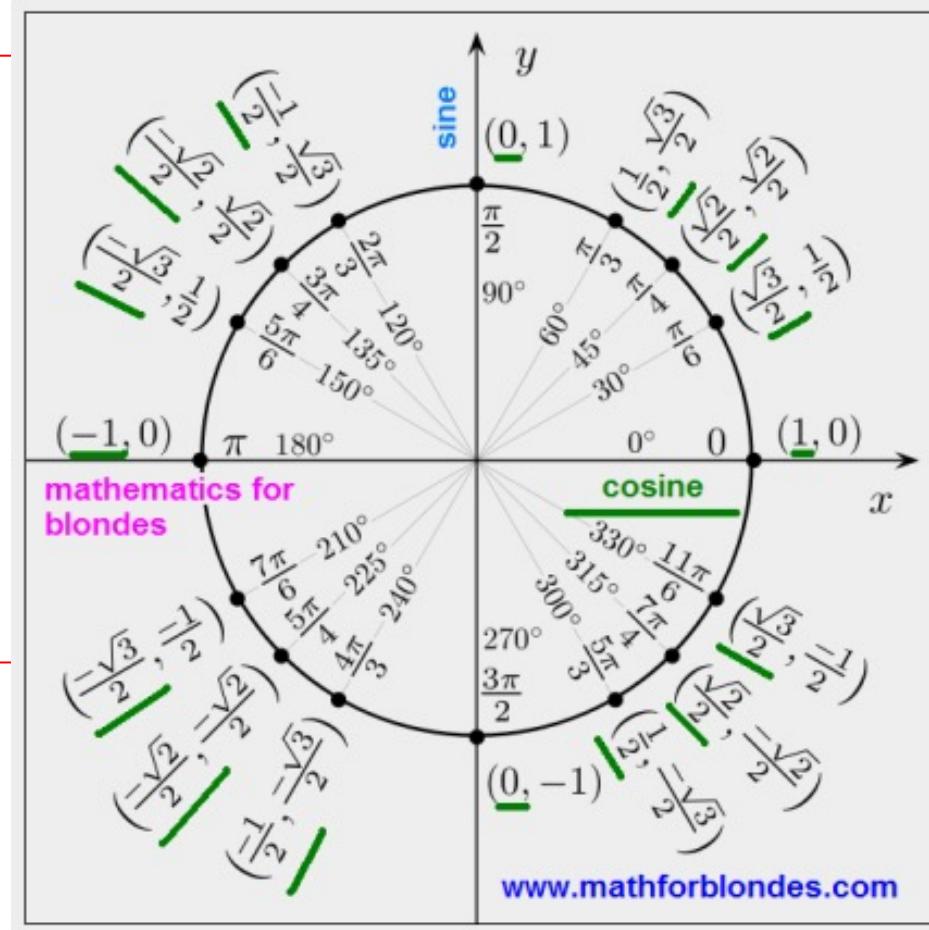
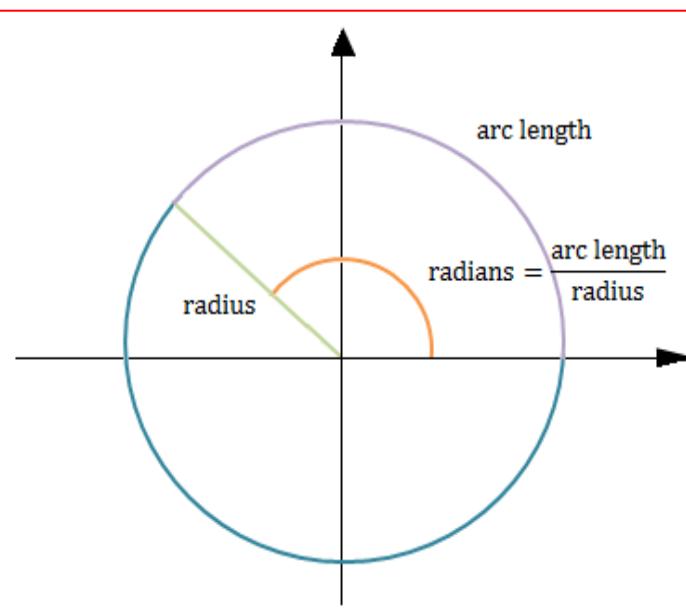
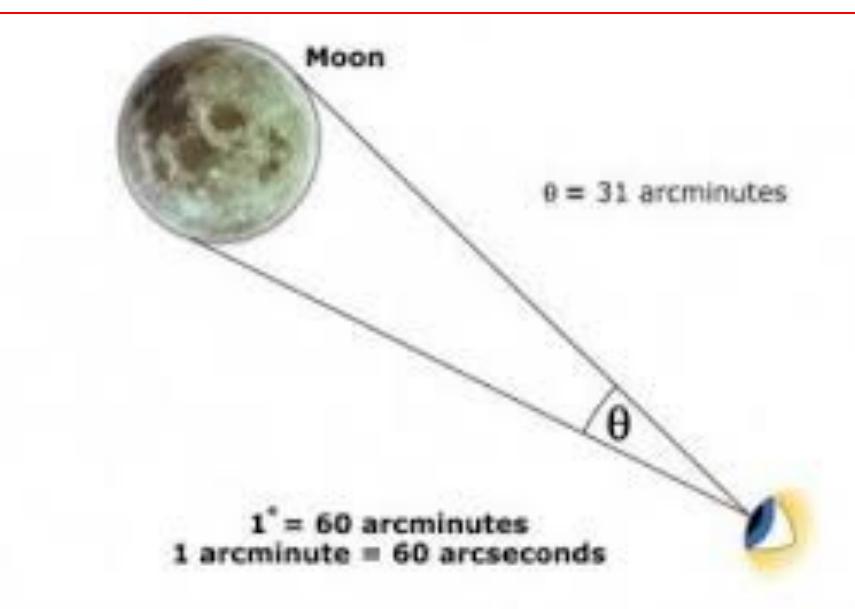
**Pythagoras:**  $r^2 = x^2 + y^2$  |  $\sin = y / r \Rightarrow y = r * \sin$  |  $\cos = x / r \Rightarrow x = r * \cos \Rightarrow (\sin^2 + \cos^2) = 1 \Leftrightarrow (a/c)^2 + (b/c)^2 = c^2/c^2 = 1$

$\pi = (\text{Length of circle} = L) / (\text{Diameter of circle} = 2 * r) \Rightarrow \pi = L/2r \Rightarrow L = 2\pi r$  where  $\pi = 22/7 = 3.141592653589793238462643383279502\dots$

Radian measure is a unit used to measure angles. As you know, there are  $360^\circ$  in a circle. If you divide that by  $2\pi$ , you get  $57.3^\circ$  per radian.

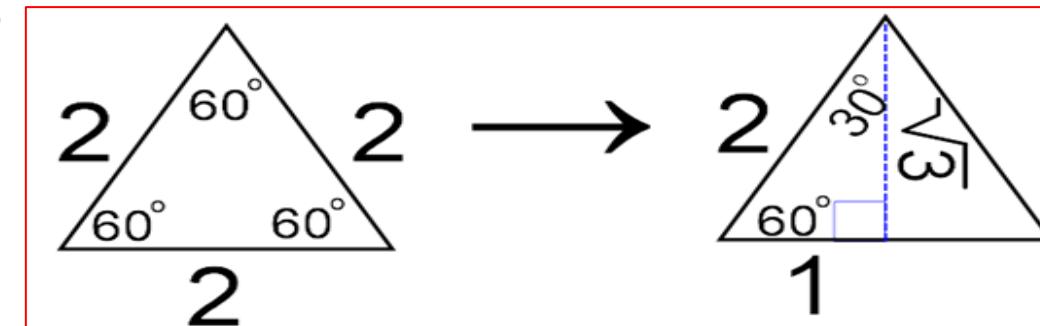
**Radians  $\times 180/\pi = Degrees$  |  $\pi = 180^\circ$**

# Mathematics – Recap Log/Ln, Complex Numbers, Euler's number e, Polar vs Cartesian, Trigonometry, ....

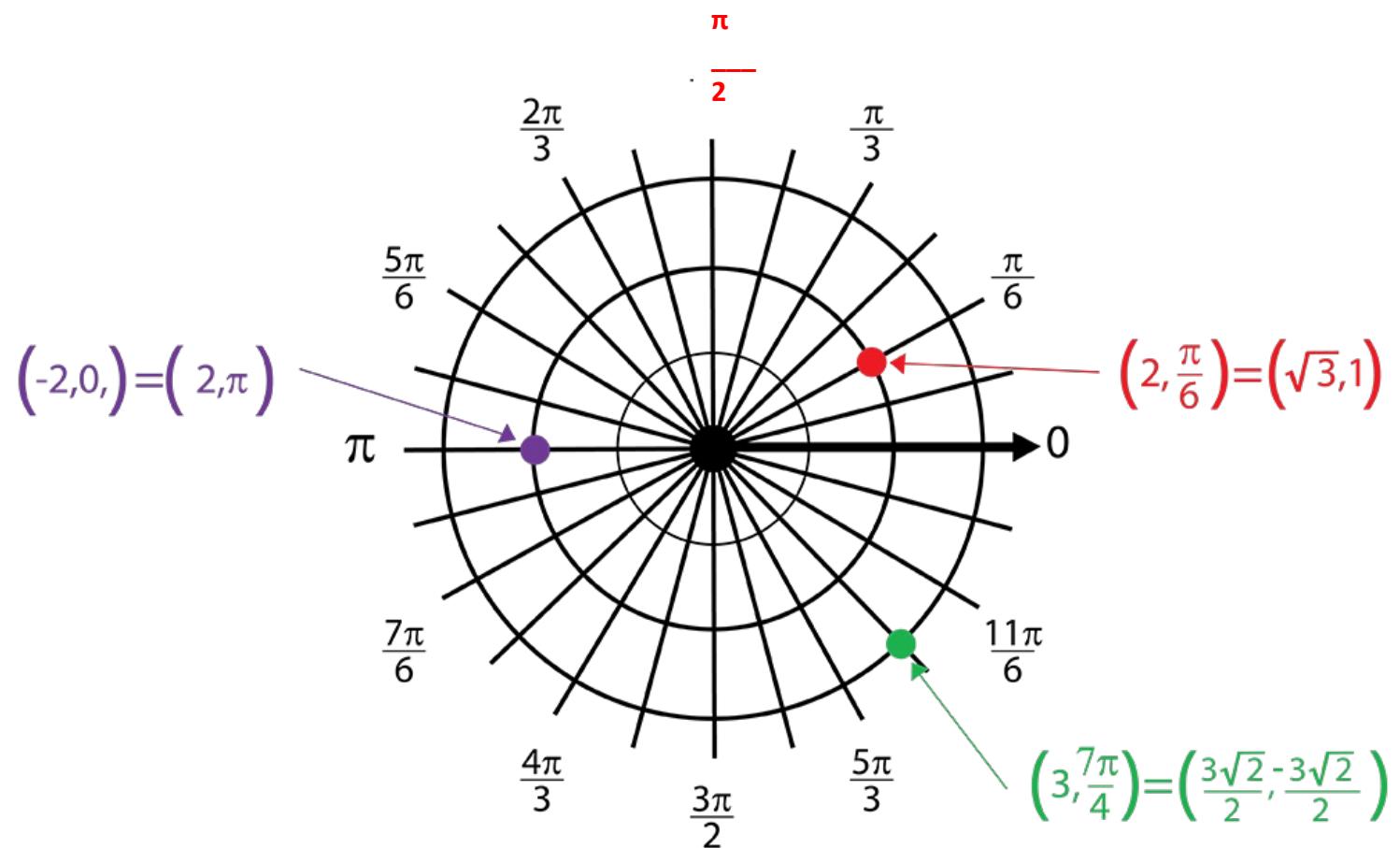


An arc minute is equal to  $1/60^{\text{th}}$  of one degree, and an arc second is equal to  $1/60^{\text{th}}$  of an arc minute. It is used in applications that involve very small angles, such as astronomy, optics, ophthalmology, optometry, and navigation.

	0	$30^\circ (\pi/6)$	$45^\circ (\pi/4)$	$60^\circ (\pi/3)$	$90^\circ (\pi/2)$	$\pi = 180^\circ$
Sin	0	$\frac{1}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$	1	
Cos	1	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{1}{2}$	0	
Tan	0	$\frac{\sqrt{3}}{3}$	1	$\sqrt{3}$	Doesn't exist	



## Mathematics – Recap Log/Ln, Complex Numbers, Euler's number e, Polar vs Cartesian, **Trigonometry**, ....



We are very familiar with the Cartesian (Rectangular) Coordinate System, which are directed horizontal and vertical distances from the x and y axes. In **Polar Coordinates**, a point in the plane is determined by its distance (radius) from the origin, now called the Pole, and the angle theta, in radians, between the line from the origin to the point and the x-axis, which is now called the Polar Axis.

The reason why we need a new coordinate system is that in some situations the Cartesian coordinate system can be quite difficult to implement (i.e., too difficult to differentiate or integrate). These instances arise when we are dealing with circular, cylindrical, or spherical symmetry.

## Intuitive Understanding Of Euler's Formula

Euler's identity seems baffling:

$$e^{i\pi} = -1$$

It emerges from a more general formula:

$$e^{ix} = \cos(x) + i \sin(x)$$

Yowza -- we're relating an *imaginary exponent* to sine and cosine! And somehow plugging in pi gives -1? Could this ever be intuitive?

Not according to 1800s mathematician Benjamin Peirce:

*It is absolutely paradoxical; we cannot understand it, and we don't know what it means, but we have proved it, and therefore we know it must be the truth.*

**Euler's formula describes two equivalent ways to move in a circle.**

That's it? This stunning equation is about spinning around? Yes -- and we can understand it by building on a few analogies:

- Starting at the number 1, see multiplication as a **transformation** that changes the number:  $1 \cdot e^{i\pi}$
- Regular **exponential growth** continuously *increases* 1 by some rate for some time period; **imaginary** exponential growth continuously *rotates* 1 for some time period
- Growing for "pi" units of time means going pi **radians** around a circle
- Therefore,  $e^{i\pi}$  means starting at 1 and rotating pi (halfway around a circle) to get to -1

That's the high-level view, let's dive into the details. By the way, if someone tries to impress you with  $e^{i\pi} = -1$ , ask them about  $i$  to the  $i$ -th power. If they can't think it through, Euler's formula is still a magic spell to them.

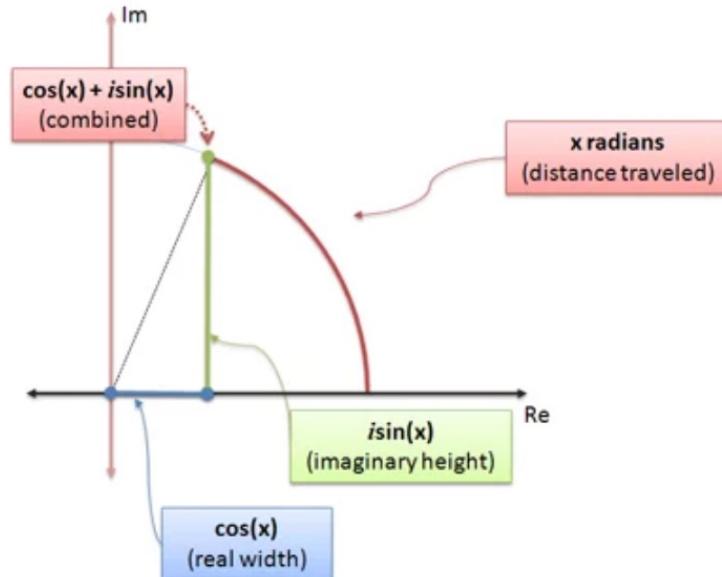
## Understanding $\cos(x) + i * \sin(x)$

The equals sign is overloaded. Sometimes we mean "set one thing to another" (like  $x = 3$ ) and others we mean "these two things describe the same concept" (like  $\sqrt{-1} = i$ ).

Euler's formula is the latter: it gives two formulas which explain how to move in a circle. If we examine circular motion using trig, and travel  $x$  radians:

A circle with  $r = 1$ , and  $x$  is the angle in radians from the origin O

### Traversing A Circle



- $\cos(x)$  is the x-coordinate (horizontal distance)
- $\sin(x)$  is the y-coordinate (vertical distance)

Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

### The Nitty Gritty Details

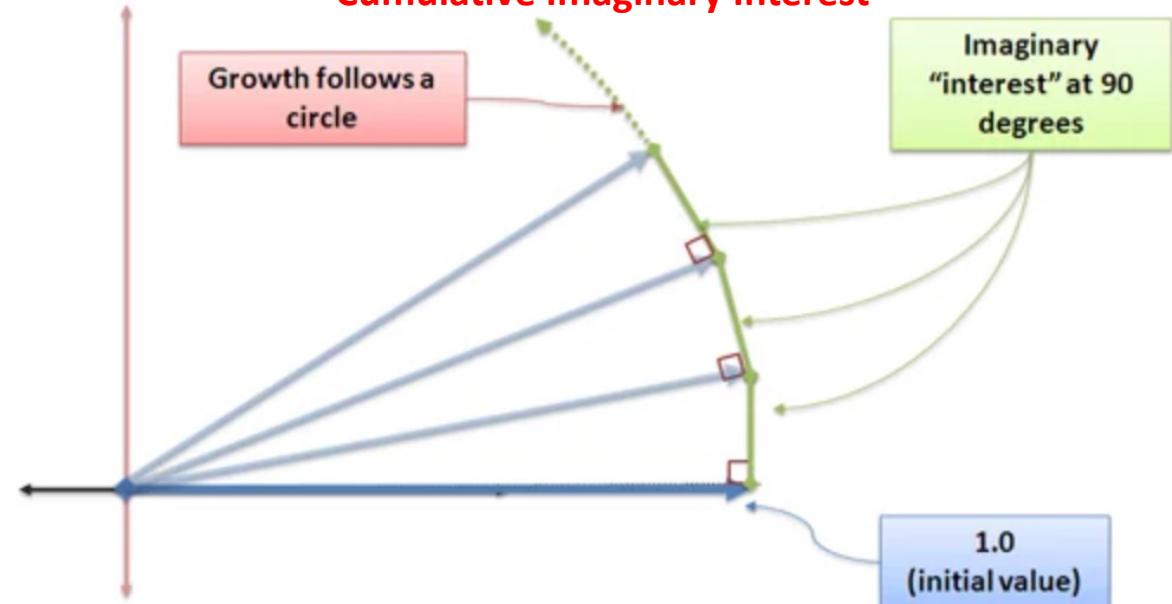
Let's take a closer look. Remember this definition of  $e$ :

$$e = e^{100\%} = \lim_{n \rightarrow \infty} \left(1 + \frac{100\%}{n}\right)^n$$

That  $\frac{100\%}{n}$  represents the portion interest we earned in each microscopic period. We assumed the interest rate was 100% in the real dimension -- but what if it were 100% in the imaginary direction?

$$e^{100\% \cdot i} = \lim_{n \rightarrow \infty} \left(1 + \frac{100\% \cdot i}{n}\right)^n$$

### Cumulative Imaginary Interest



## Mathematics – Recap Log/Ln, Complex Numbers, Euler's formula/number e, Polar vs Cartesian, Trigonometry, ....

Example:  $e^i$

Where's the x? Ah, it's just 1. Intuitively, without breaking out a calculator, we know that this means "travel 1 radian along the unit circle". In my head, I see "e" trying to grow 1 at 100% all in the same direction, but  $i$  keeps moving the ball and forces "1" to grow along the edge of a circle:

$$e^i = \cos(1) + i \cdot \sin(1) = .5403 + .8415i$$

Not the prettiest number, but there it is. Remember to put your calculator in radian mode when punching this in.

A complex growth rate like  $(a + bi)$  is a mix of real and imaginary growth. The real part  $a$ , means "grow at 100% for  $a$  seconds" and the imaginary part  $b$  means "rotate for  $b$  seconds". Remember, rotations don't get the benefit of compounding since you keep 'pushing' in a different direction -- rotation adds up linearly.

With this in mind, we can represent any point on any sized circle using  $(a+bi)$ ! The radius is  $e^a$  and the angle is determined by  $e^{bi}$ . It's like putting the number in the expand-o-tron for two cycles: once to grow it to the right size ( $a$  seconds), another time to rotate it to the right angle ( $b$  seconds). Or, you could rotate it first and then grow!

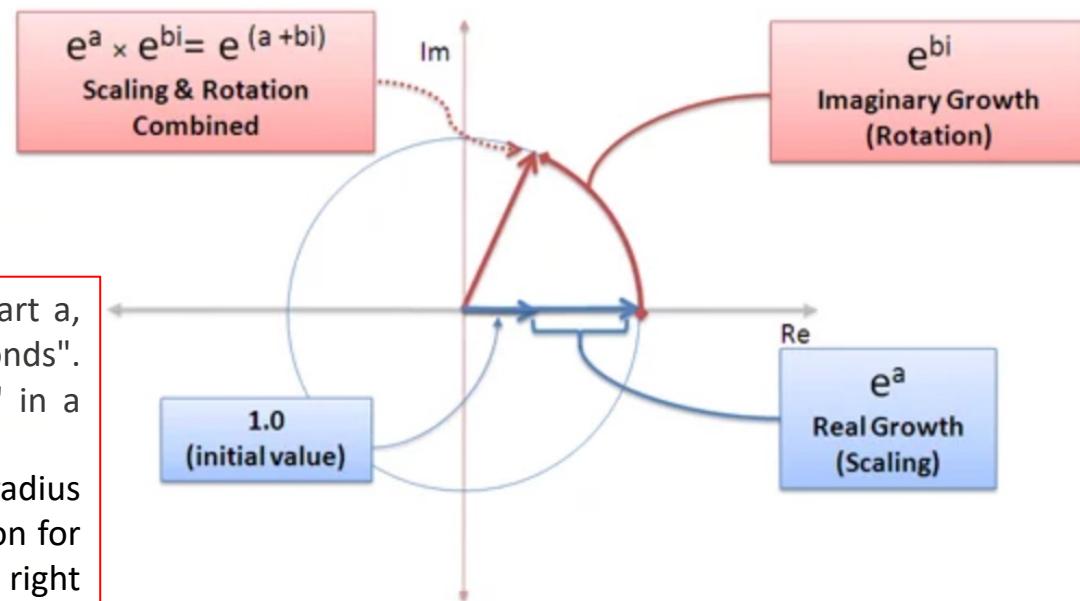
Let's say we want to know the growth amount to get to  $6 + 8i$ . This is really asking for the natural log of an imaginary number: how do we grow  $e$  to get  $(6 + 8i)$ ?

- Radius: How big of a circle do we need? The magnitude is  $\sqrt{6^2 + 8^2} = \sqrt{100} = 10$ . Which means we need to grow for  $\ln(10) = 2.3$  seconds to reach that amount.
- Amount to rotate: What's the angle of that point? We can use arctan to figure it out:  $\arctan(8/6) = 53$  degrees =  $.93$  radian.
- Combine the result:  $\ln(6+8i) = 2.3 + .93i$

## Complex Growth

We can have real and imaginary growth at the same time: the real portion scales us up, and the imaginary part rotates us around:

## Real & Imaginary Growth



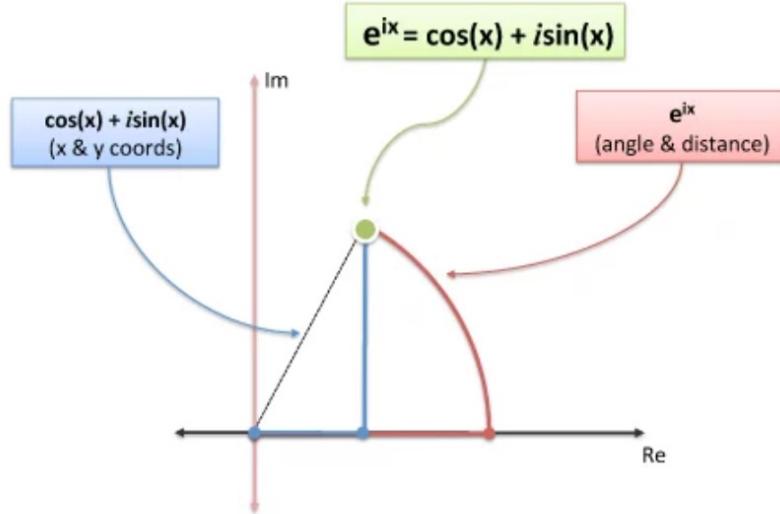
That is, we can reach the random point  $(6 + 8i)$  if we use  $e^{2.3+.93i}$

## Why Is This Useful?

Euler's formula gives us another way to describe motion in a circle. But we could already do that with sine and cosine -- what's so special?

It's all about perspective. Sine and cosine describe motion in terms of a *grid*, plotting out horizontal and vertical coordinates.

## Two Paths, Same Result



Euler's formula uses polar coordinates -- what's your angle and distance? Again, it's two ways to describe motion:

- Grid system: Go 3 units east and 4 units north
- Polar coordinates: Go 5 units at an angle of 53.13 degrees

<https://betterexplained.com/articles/intuitive-understanding-of-eulers-formula/>

## Mathematics – Recap Log/Ln, Complex Numbers, **Euler's formula/number e**, Polar vs Cartesian, Trigonometry, ....

Depending on the problem, polar or rectangular coordinates are more useful. Euler's formula lets us convert between the two to use the best tool for the job. Also, because  $e^{ix}$  can be converted to sine and cosine, we can rewrite formulas in trig as variations on  $e$ , which comes in very handy (no need to memorize  $\sin(a+b)$ ). And it's beautiful that every number, real or complex, is a variation of  $e$ .

But utility, schmutility: the most important result is the realization that baffling equations can become intuitive with the right analogies. Don't let beautiful equations like Euler's formula remain a magic spell -- build on the analogies you know to see the insights inside the equation.

**Happy math!**

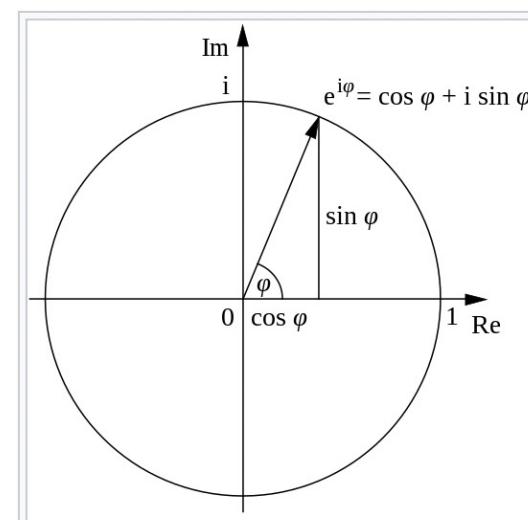
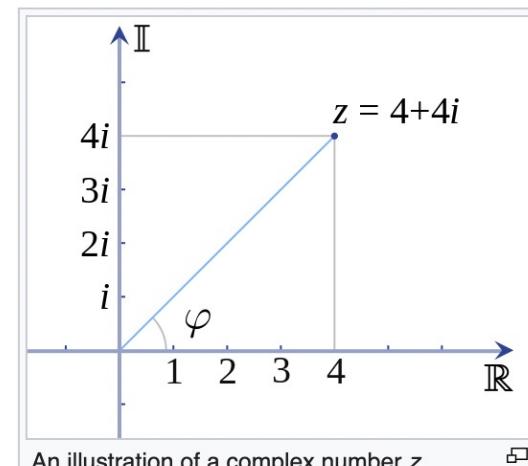
## Mathematics – Recap Log/Ln, Complex Numbers, Euler's number e, Polar vs Cartesian, Trigonometry, ....

For complex  $z$ , Using the [ratio test](#), it is possible to show that this [power series](#) has an infinite [radius of convergence](#) and so defines  $e^z$  for all complex  $z$ :

$$e^z = 1 + \frac{z}{1!} + \frac{z^2}{2!} + \frac{z^3}{3!} + \cdots = \sum_{n=0}^{\infty} \frac{z^n}{n!}. \quad e^z = \lim_{n \rightarrow \infty} \left(1 + \frac{z}{n}\right)^n.$$

$$\begin{aligned} i^0 &= 1, & i^1 &= i, & i^2 &= -1, & i^3 &= -i, \\ i^4 &= 1, & i^5 &= i, & i^6 &= -1, & i^7 &= -i \\ \vdots & & \vdots & & \vdots & & \vdots \end{aligned}$$

$$\begin{aligned} e^{ix} &= 1 + ix + \frac{(ix)^2}{2!} + \frac{(ix)^3}{3!} + \frac{(ix)^4}{4!} + \frac{(ix)^5}{5!} + \frac{(ix)^6}{6!} + \frac{(ix)^7}{7!} + \frac{(ix)^8}{8!} + \cdots \\ &= 1 + ix - \frac{x^2}{2!} - \frac{ix^3}{3!} + \frac{x^4}{4!} + \frac{ix^5}{5!} - \frac{x^6}{6!} - \frac{ix^7}{7!} + \frac{x^8}{8!} + \cdots \\ &= \left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots\right) + i \left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots\right) \\ &= \cos x + i \sin x, \quad \text{if and only if, } x \text{ is radians} \end{aligned}$$



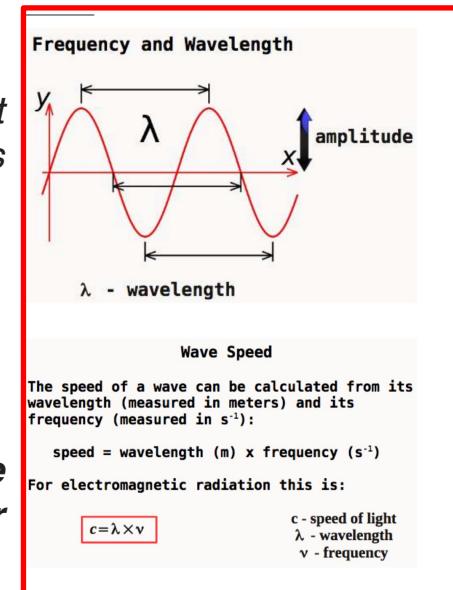
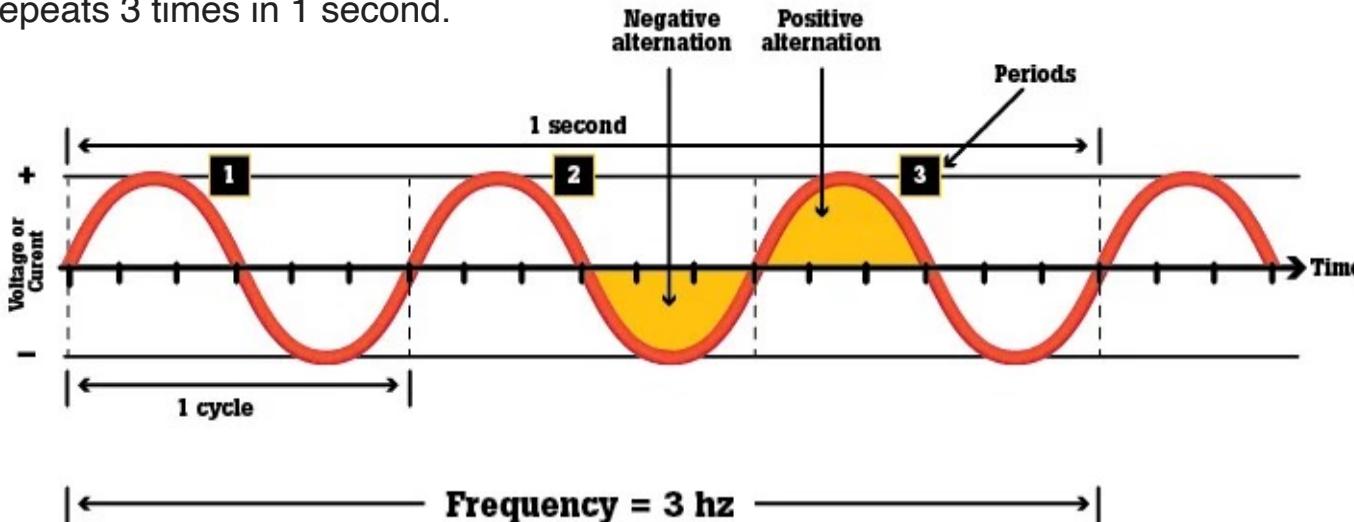
**Wiki:** “The hertz is the unit of frequency in the International System of Units, equivalent to one event per second.  
... ⇔ 1 Hz is equal to one cycle per second.”

Alternating current (ac) frequency is the number of cycles per second in an ac sine wave. Frequency is the rate at which current changes direction per second. It is measured in hertz (Hz), an international unit of measure where 1 hertz is equal to 1 cycle per second.

- Hertz (Hz) = One hertz is equal to one cycle per second.
- Cycle = One complete wave of alternating current or voltage.
- Alternation = One half of a cycle.
- Period = The time required to produce one complete cycle of a waveform.

**At its most basic, frequency is how often something repeats. In the case of electrical current, frequency is the number of times a sine wave repeats, or completes, a positive-to-negative cycle. The more cycles that occur per second, the higher the frequency.**

**Example:** If an alternating current is said to have a frequency of 3 Hz (see diagram below), that indicates its waveform repeats 3 times in 1 second.



Frequency is typically used to describe electrical equipment operation. Below are some common frequency ranges:

- Power line frequency (normally 50 Hz or 60 Hz).
- Variable-frequency drives, which normally use a 1-20 kilohertz (kHz) carrier frequency.
- Audio frequency range: 15 Hz to 20 kHz (the range of human hearing).
- Radio frequency: 30-300 kHz.
- Low frequency: 300 kHz to 3 megahertz (MHz).
- Medium frequency: 3-30 MHz.
- High frequency: 30-300 MHz.

\* In terms of sound waves, 1 Hertz (Hz) equals one vibration per second. Human voice averages a frequency band of 1000 Hertz, or 1 kilohertz (kHz), but the 'normal' hearing frequency range of a healthy young person is about 20 to 20,000Hz.

## Mathematics – Recap Log/Ln, Euler's formula, Polar vs Cartesian, Trigonometry, Frequency, Fourier Transformation, ....

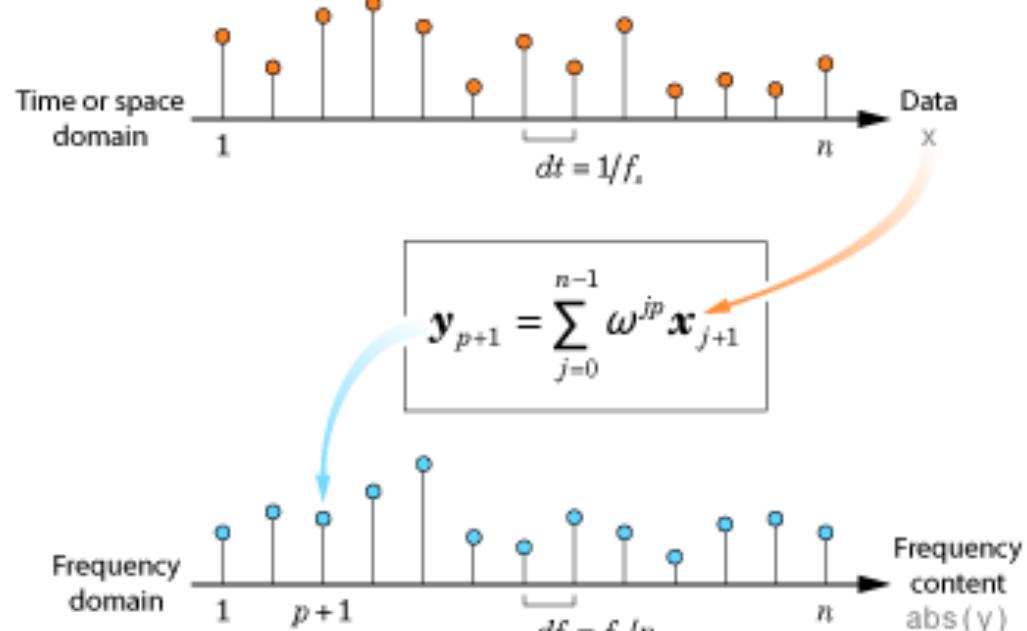
**Transforms and filters are tools for processing and analyzing discrete data**, and are commonly *used in signal processing applications* and computational mathematics. **When data is represented as a function of time or space, the Fourier transform decomposes the data into frequency components.** The [fft](#) function uses a fast Fourier transform algorithm (Augustine Louis Cauchy, published in 1827) that reduces its computational cost compared to other direct implementations. For a more detailed introduction to Fourier analysis, see [Fourier Transforms](#). The [conv](#) and [filter](#) functions are also useful tools for modifying the amplitude or phase of input data using a transfer function.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}$$

**DIRECT**

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i2\pi kn/N}$$

**INVERSE**



**Discrete Fourier Transform (DFT)**

<a href="#">fft</a>	Fast Fourier transform
<a href="#">fft2</a>	2-D fast Fourier transform
<a href="#">fftn</a>	N-D fast Fourier transform
<a href="#">nufft</a>	Nonuniform fast Fourier transform ( <i>Since R2020a</i> )
<a href="#">nufftn</a>	N-D nonuniform fast Fourier transform ( <i>Since R2020a</i> )
<a href="#">fftshift</a>	Shift zero-frequency component to center of spectrum
<a href="#">fftw</a>	Define method for determining FFT algorithm
<a href="#">ifft</a>	Inverse fast Fourier transform
<a href="#">ifft2</a>	2-D inverse fast Fourier transform
<a href="#">ifftn</a>	Multidimensional inverse fast Fourier transform
<a href="#">ifftshift</a>	Inverse zero-frequency shift
<a href="#">nextpow2</a>	Exponent of next higher power of 2
<a href="#">interpft</a>	1-D interpolation (FFT method)

## Discrete Fourier Transform (DFT)

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}$$

This example demonstrates how to apply the DFT to a sequence of length  $N = 4$  and the input vector

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 - i \\ -i \\ -1 + 2i \end{pmatrix}.$$

Here's the "math English" version:

**The Fourier Transform** takes a time-based pattern, measures every possible cycle, and returns the overall "cycle recipe" (the amplitude, offset, & rotation speed for every cycle that was found).

Calculating the DFT of  $\mathbf{x}$  using [Eq.1](#)

$$X_0 = e^{-i2\pi0\cdot0/4} \cdot 1 + e^{-i2\pi0\cdot1/4} \cdot (2 - i) + e^{-i2\pi0\cdot2/4} \cdot (-i) + e^{-i2\pi0\cdot3/4} \cdot (-1 + 2i) = 2$$

$$X_1 = e^{-i2\pi1\cdot0/4} \cdot 1 + e^{-i2\pi1\cdot1/4} \cdot (2 - i) + e^{-i2\pi1\cdot2/4} \cdot (-i) + e^{-i2\pi1\cdot3/4} \cdot (-1 + 2i) = -2 - 2i$$

$$X_2 = e^{-i2\pi2\cdot0/4} \cdot 1 + e^{-i2\pi2\cdot1/4} \cdot (2 - i) + e^{-i2\pi2\cdot2/4} \cdot (-i) + e^{-i2\pi2\cdot3/4} \cdot (-1 + 2i) = -2i$$

$$X_3 = e^{-i2\pi3\cdot0/4} \cdot 1 + e^{-i2\pi3\cdot1/4} \cdot (2 - i) + e^{-i2\pi3\cdot2/4} \cdot (-i) + e^{-i2\pi3\cdot3/4} \cdot (-1 + 2i) = 4 + 4i$$

results in

$$\mathbf{X} = \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} 2 \\ -2 - 2i \\ -2i \\ 4 + 4i \end{pmatrix}.$$

The Fourier Transform is one of deepest insights ever made. Unfortunately, the meaning is buried within dense equations:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}$$

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i2\pi kn/N}$$

Yikes. Rather than jumping into the symbols, let's experience the key idea firsthand. Here's a plain-English metaphor:

- **What does the Fourier Transform do?** Given a smoothie, it finds the recipe.
- **How?** Run the smoothie through filters to extract each ingredient.
- **Why?** Recipes are easier to analyze, compare, and modify than the smoothie itself.
- **How do we get the smoothie back?** Blend the ingredients.

Here's the "math English" version of the above:

- The Fourier Transform takes a time-based pattern, measures every possible cycle, and returns the overall "cycle recipe" (the amplitude, offset, & rotation speed for every cycle that was found).

# Smoothie to Recipe

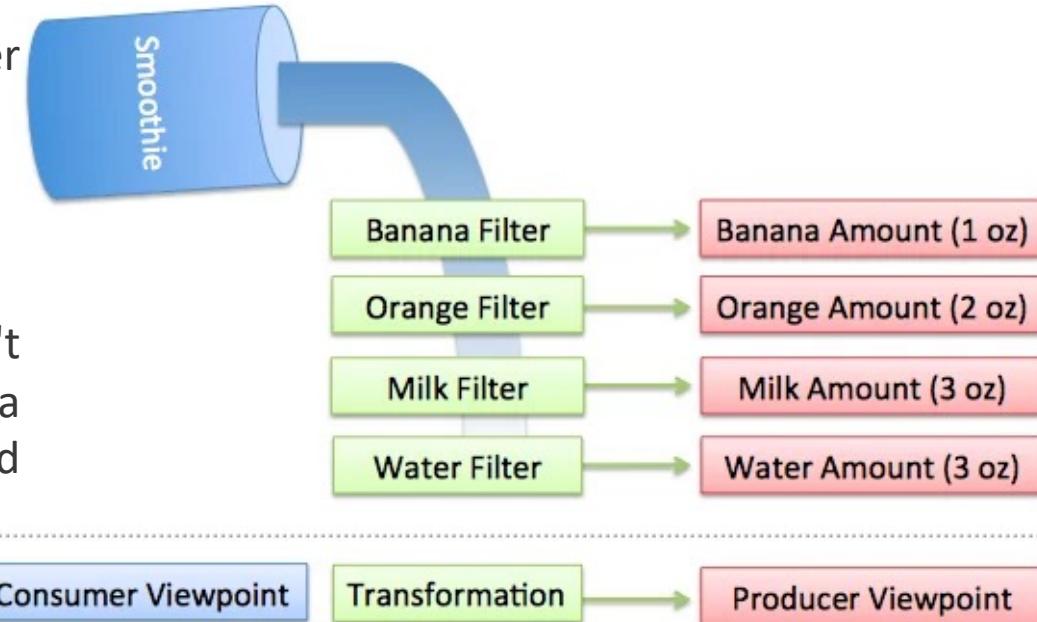
A math transformation is a change of perspective.

The Fourier Transform changes our perspective from consumer to producer, turning ***What do I have?*** Into => ***How was it made?***

***In other words: given a smoothie, let's find the recipe.***

Why? Well, recipes are great descriptions of drinks. You wouldn't share a drop-by-drop analysis, you'd say "I had an orange/banana smoothie". A recipe is more easily categorized, compared, and modified than the object itself.

***So, ... given a smoothie, how do we find the recipe?***



Well, imagine you had a few **filters** lying around:

- Pour through the "banana" filter. 1 oz of bananas are extracted.
- Pour through the "orange" filter. 2 oz of oranges.
- Pour through the "milk" filter. 3 oz of milk.
- Pour through the "water" filter. 3 oz of water.

We can reverse-engineer the recipe by filtering each ingredient. The catch?

- **Filters must be independent.** The banana filter needs to capture bananas, and nothing else. Adding more oranges should never affect the banana reading.
- **Filters must be complete.** We won't get the real recipe if we leave out a filter ("There were mangoes too!"). Our collection of filters must catch every possible ingredient.
- **Ingredients must be combine-able.** Smoothies can be separated and re-combined without issue (A cookie? Not so much. Who wants crumbs?). The ingredients, when separated and combined in any order, must make the same result.

## See The World As Cycles

The Fourier Transform takes a specific viewpoint: **What if any signal could be filtered into a bunch of circular paths?**

Whoa. This concept is mind-blowing, and poor Joseph Fourier had his idea rejected at first. (*Really Joe, even a staircase pattern can be made from circles?*) And despite decades of debate in the math community, we expect students to internalize the idea without issue. Ugh. Let's walk through the intuition.

**The Fourier Transform** finds the recipe for a signal, like our smoothie process:

- Start with a time-based signal
- Apply filters to measure each possible "circular ingredient"
- Collect the full recipe, listing the amount of each "circular ingredient"

Stop. Here's where most tutorials excitedly throw engineering applications at your face. Don't get scared; think of the examples as "Wow, we're finally seeing the source code (DNA) behind previously confusing ideas".

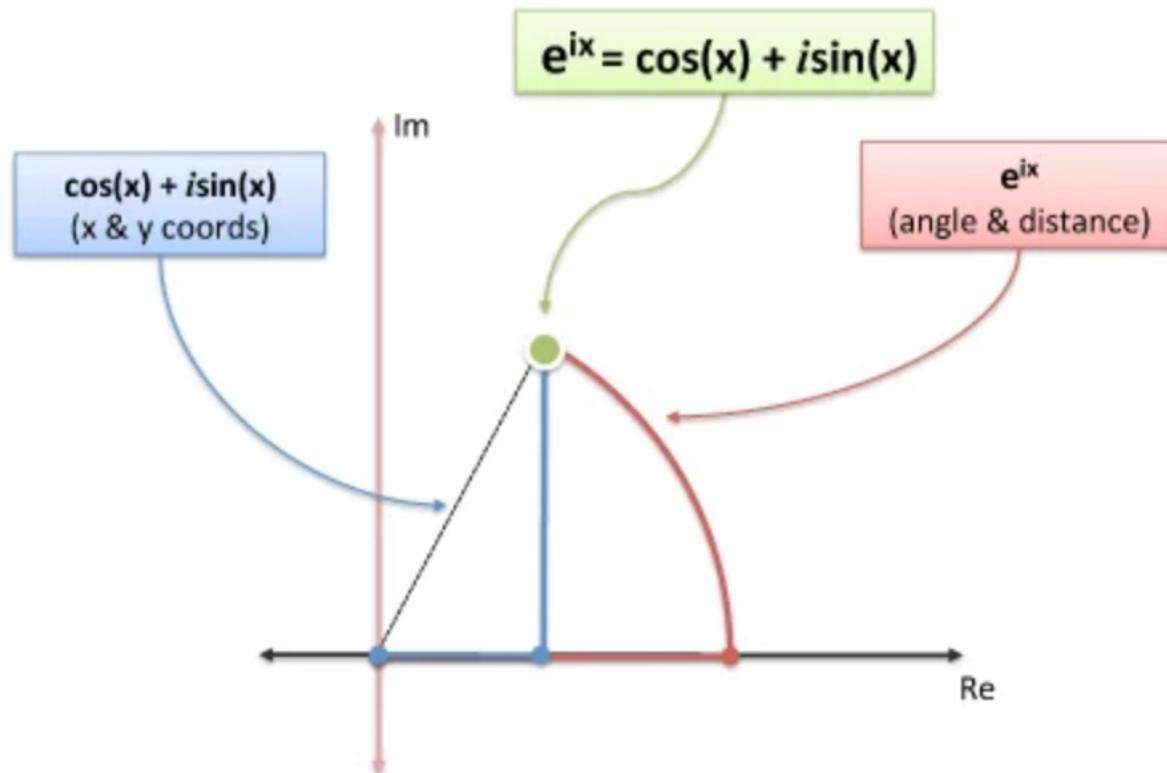
- *If earthquake vibrations can be separated into "ingredients" (vibrations of different speeds & amplitudes), buildings can be designed to avoid interacting with the strongest ones.*
- *If sound waves can be separated into ingredients (bass and treble frequencies), we can boost the parts we care about, and hide the ones we don't. The crackle of random noise can be removed. Maybe similar "sound recipes" can be compared (music recognition services compare recipes, not the raw audio clips).*
- *If computer data/files content can be represented with oscillating patterns, perhaps the least-important ones can be ignored. This "lossy compression" can drastically shrink file sizes (and why JPEG and MP3 files are much smaller than raw .bmp or .wav files).*
- *If a radio wave is our signal, we can use filters to listen to a particular channel. In the smoothie world, imagine each person paid attention to a different ingredient: Adam looks for apples and Bob looks for water.*

*The Fourier Transform is useful in engineering, sure, but it's a metaphor about finding the root causes behind an observed effect.*

Must we use imaginary exponents to move in a circle? Nope. But it's convenient and compact. And sure, we can describe our path as coordinated motion in two dimensions (real and imaginary), but don't forget the big picture: we're just moving in a circle.

**PLAY!!! – <https://planetcalc.com/7543/>**

## Two Paths, Same Result



### Following Circular Paths

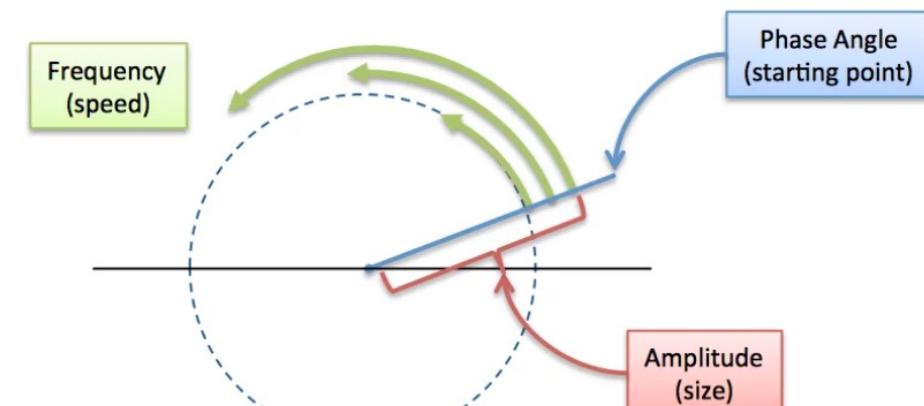
Let's say we're chatting on the phone and, like usual, I want us to draw the same circle simultaneously. (*You promised!*) What should I say?

- How big is the circle? (Amplitude, i.e. size of radius)
- How fast do we draw it? (Frequency. 1 circle/second is a frequency of 1 Hertz (Hz) or  $2\pi$  radians/sec)

• Where do we start? (Phase angle, where 0 degrees is the x-axis)

I could say "2-inch radius, start at 45 degrees, 1 circle per second, go!". After half a second, we should each be pointing to: starting point + amount traveled =  $45 + 180 = 225$  degrees (on a 2-inch circle).

## Describing A Circular Path



Every circular path needs a size, speed, and starting angle (amplitude/frequency/phase). We can even combine paths: imagine tiny motorcars, driving in circles at different speeds.

The combined position of *all the cycles* is our signal, just like the combined flavor of *all the ingredients* is our smoothie.

Here's a simulation of a basic circular path:

<https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/>

The magnitude of each cycle is listed in order, starting at 0Hz.

Cycles [0 1] means

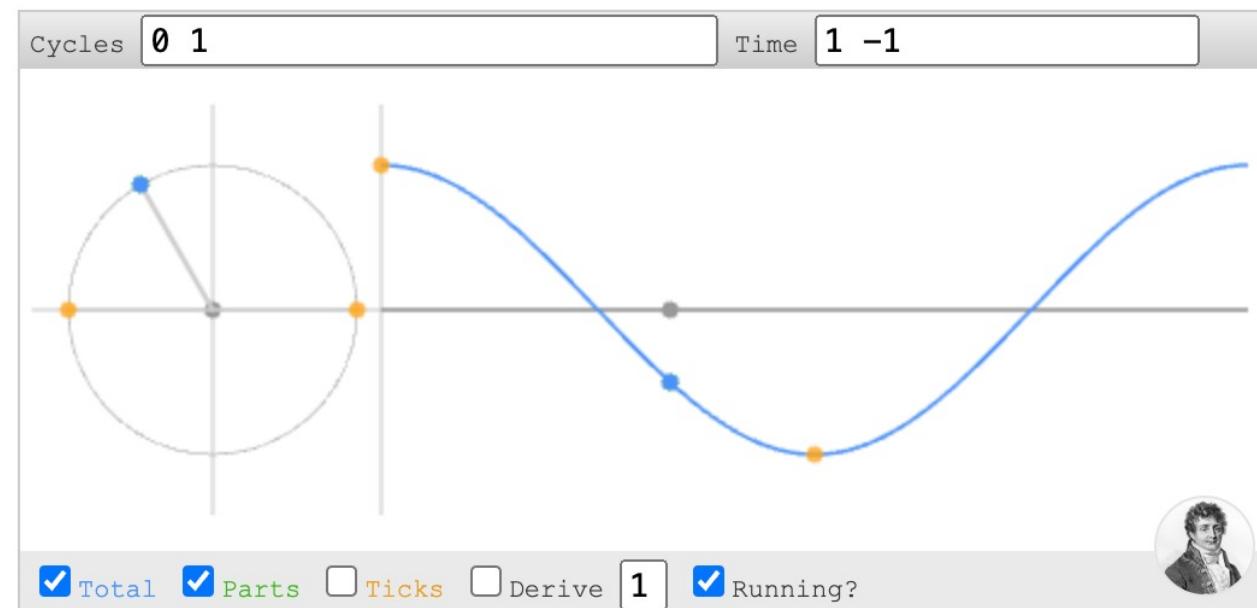
- 0 amplitude for the 0Hz cycle (0Hz = a constant cycle, stuck on the x-axis at zero degrees)
- 1 amplitude for the 1Hz cycle (completes 1 cycle per time interval)

**Now the tricky part:**

• The blue graph measures the *real part of the cycle*. Another lovely math confusion: the real axis of the circle, which is usually horizontal, has its magnitude shown on the vertical axis. You can mentally rotate the circle 90 degrees if you like.

• The time points are spaced at the fastest frequency. A 1Hz signal needs 2 time points for a start and stop (a single data point doesn't have a frequency). The time values [1 -1] shows the amplitude at these equally-spaced intervals.

[0 1] is a pure 1Hz cycle.



Now let's add a 2Hz cycle to the mix. [0 1 1] means "Nothing at 0Hz, 1Hz of amplitude 1, 2Hz of amplitude 1":

[https://planetcalc.com/7543/ - PLAY!!!](https://planetcalc.com/7543/)

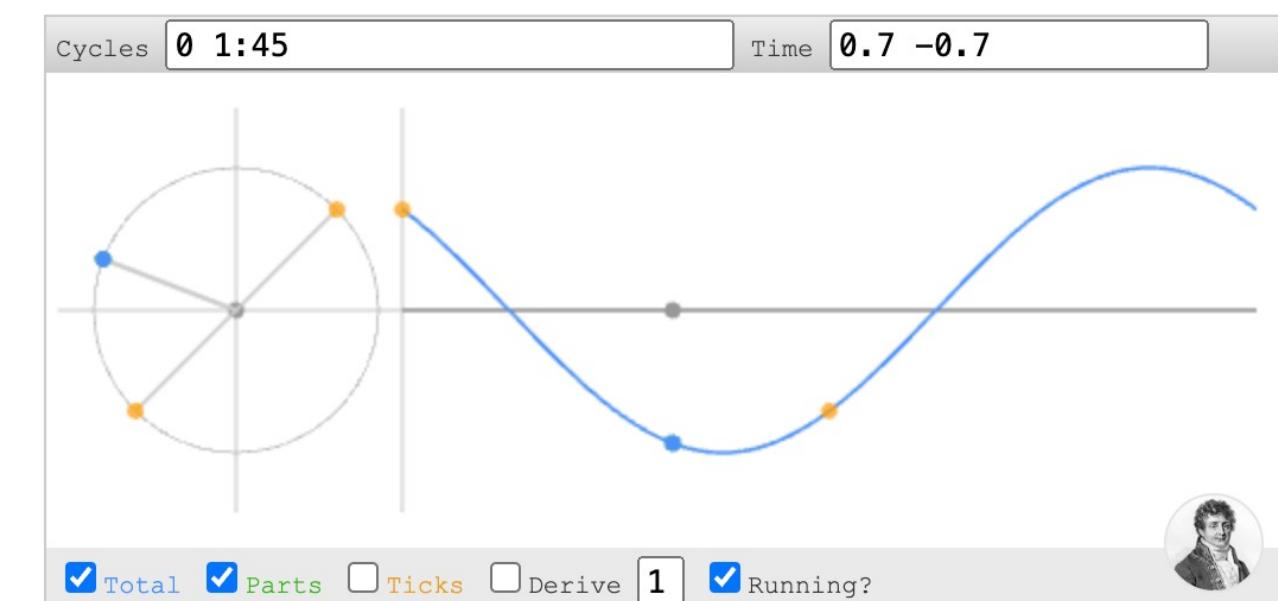
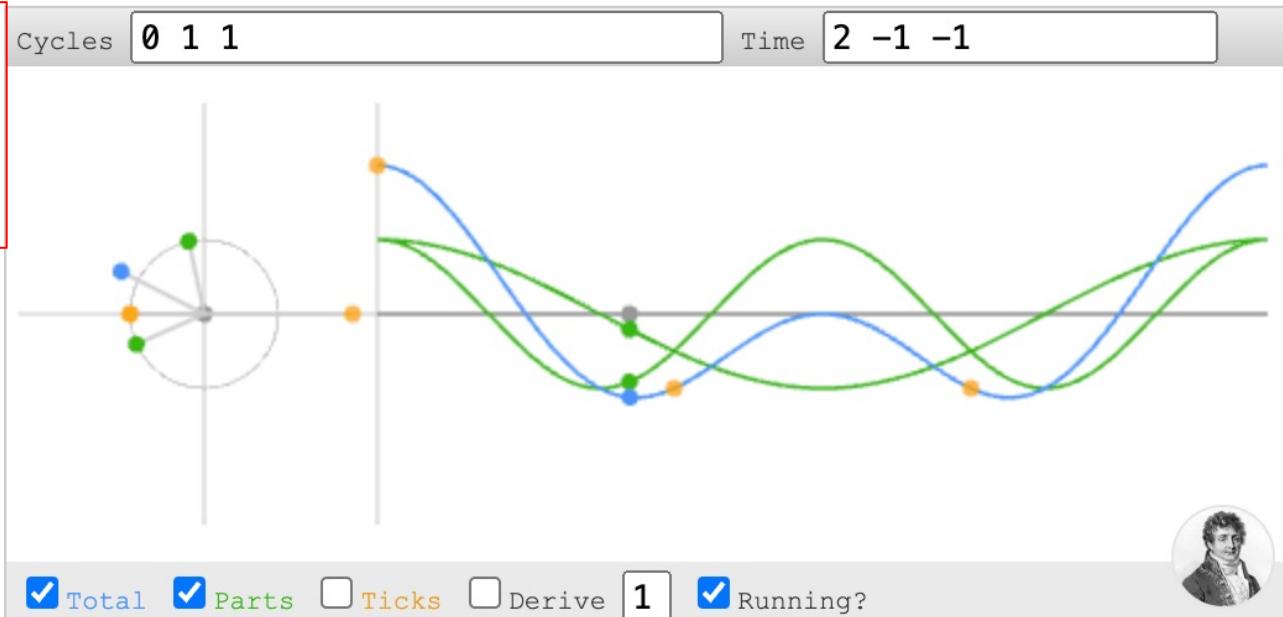
<https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/>

Whoa! The little motorcars are getting wild: the green lines are the 1Hz and 2Hz cycles, and the blue line is the combined result. Try toggling the green checkbox to see the final result clearly. The combined "flavor" is a sway that starts at the max and dips low for the rest of the interval.

The yellow dots are when we actually measure the signal. With 3 cycles defined (0Hz, 1Hz, 2Hz), each dot is 1/3 of the way through the signal. In this case, cycles [0 1 1] generate the time values [2 -1 -1], which starts at the max (2) and dips low (-1).

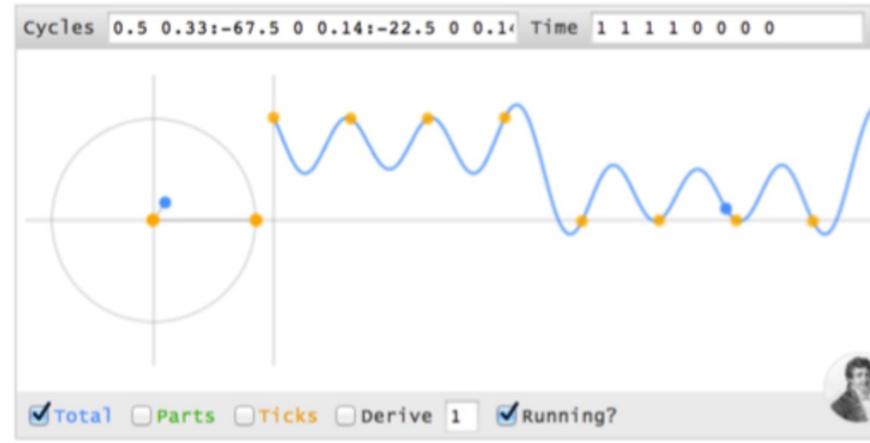
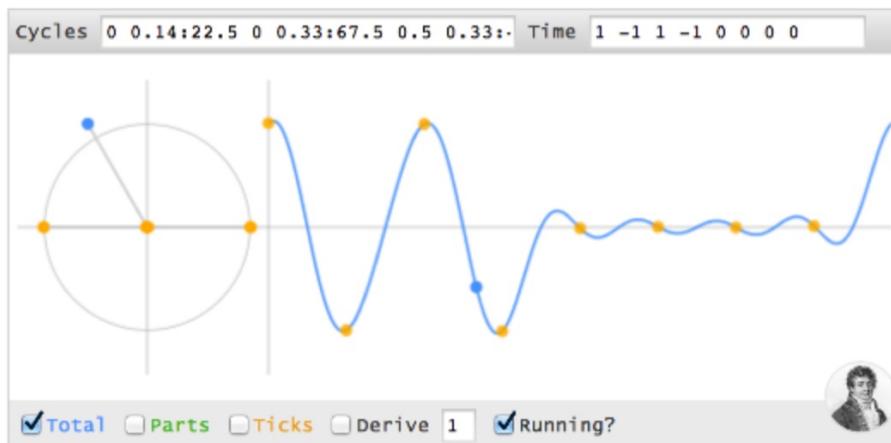
Oh! We can't forget phase, the starting angle! Use magnitude:angle to set the phase. So [0 1:45] is a 1Hz cycle that starts at 45 degrees.

This is a shifted version of [0 1]. On the time side we get [.7 -.7] instead of [1 -1], because our cycle isn't exactly lined up with our measuring intervals, which are still at the halfway point (this could be desired!).



## The Fourier Transform finds the set of cycle speeds, amplitudes and phases to match any time signal.

Our signal becomes an abstract notion that we consider as "observations in the time domain" or "ingredients in the frequency domain". Enough talk: try it out! In the simulator, type any time or cycle pattern you'd like to see. If it's time points, you'll get a collection of cycles (that combine into a "wave") that matches your desired points.



But... doesn't the combined wave have strange values between the yellow time intervals? Sure. But who's to say whether a signal travels in straight lines, or curves, or zips into other dimensions when we aren't measuring it? It behaves exactly as we need at the equally-spaced moments we asked for.

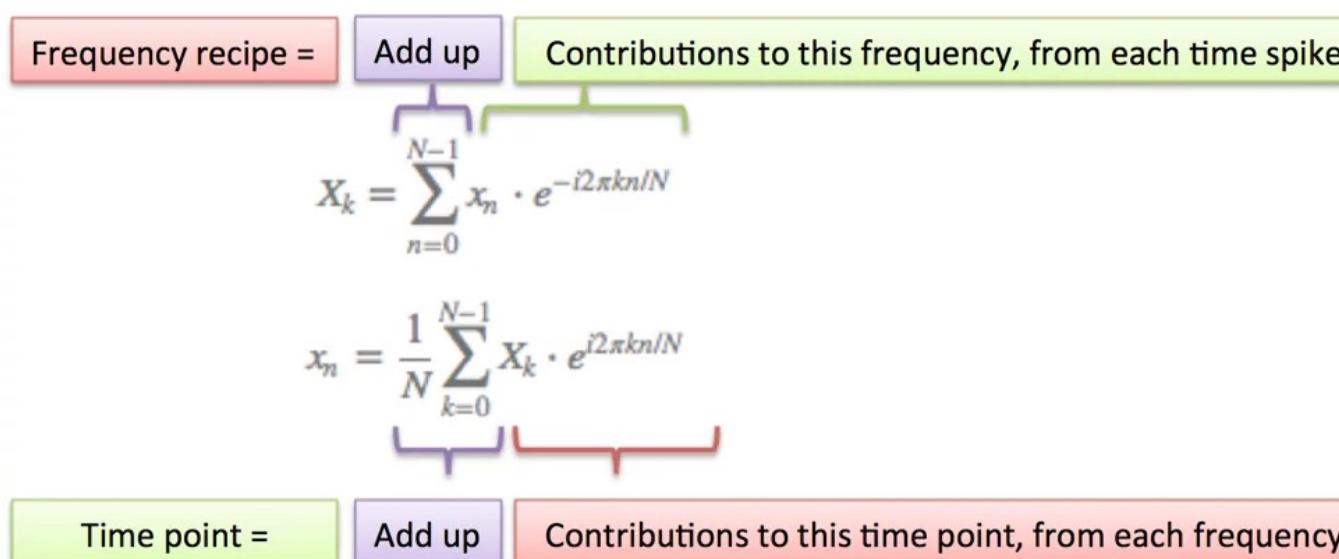
## Discovering The Full Transform

The big insight: our signal is just a bunch of time spikes! If we merge the recipes for each time spike, we should get the recipe for the full signal. The Fourier Transform builds the recipe frequency-by-frequency:

- Separate the full signal (a b c d) into "time spikes": (a 0 0 0) (0 b 0 0) (0 0 c 0) (0 0 0 d)
- For any frequency (like 2Hz), the *tentative* recipe is "a/4 + b/4 + c/4 + d/4" (the amplitude of each spike is split among all frequencies)
- Wait! We need to offset each spike with a phase delay (the angle for a "1 second delay" depends on the frequency).
- Actual recipe for a frequency = a/4 (no offset) + b/4 (1 second offset) + c/4 (2 second offset) + d/4 (3 second offset).

We can then loop through every frequency to get the full transform.

Here's the conversion from "math English" to full math:



- N = number of time samples we have
- n = current sample we're considering (0 .. N-1)
- $x_n$  = value of the signal at time n
- k = current frequency we're considering (0 Hertz up to N-1 Hertz)
- $X_k$  = amount of frequency k in the signal (amplitude and phase, a complex number)
- The  $1/N$  factor is usually moved to the *reverse transform* (going from frequencies back to time). This is allowed, though I prefer  $1/N$  in the forward transform since it gives the *actual* sizes for the time spikes. You can get wild and even use  $1/\sqrt{N}$  on both transforms (going forward and back creates the  $1/N$  factor).
- $n/N$  is the percent of the time we've gone through.  $2 * \pi * k$  is our speed in radians / sec.  $e^{-ix}$  is our backwards-moving circular path. The combination is how far we've moved, for this speed and time.
- The raw equations for the Fourier Transform just say "add the complex numbers". Many programming languages cannot handle complex numbers directly, so you convert everything to rectangular coordinates and add those.

## Mathematics – Recap Log/Ln, **Euler's formula**, Polar vs Cartesian, Trigonometry, ....

From Wikipedia, the free encyclopedia

*This article is about Euler's formula in complex analysis. For other uses, see [List of things named after Leonhard Euler § Formulae](#).*

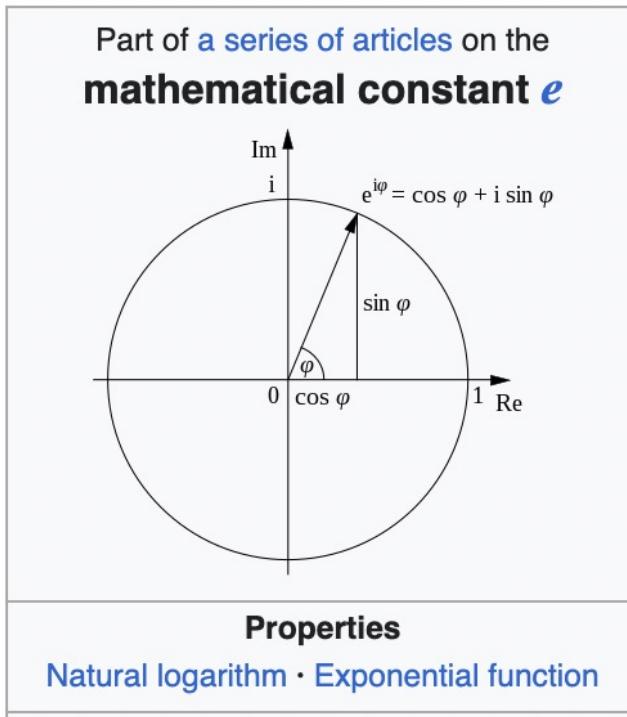
**Euler's formula**, named after [Leonhard Euler](#), is a [mathematical formula](#) in [complex analysis](#) that establishes the fundamental relationship between the [trigonometric functions](#) and the [complex exponential function](#). Euler's formula states that for any [real number](#)  $x$ :

$$e^{ix} = \cos x + i \sin x,$$

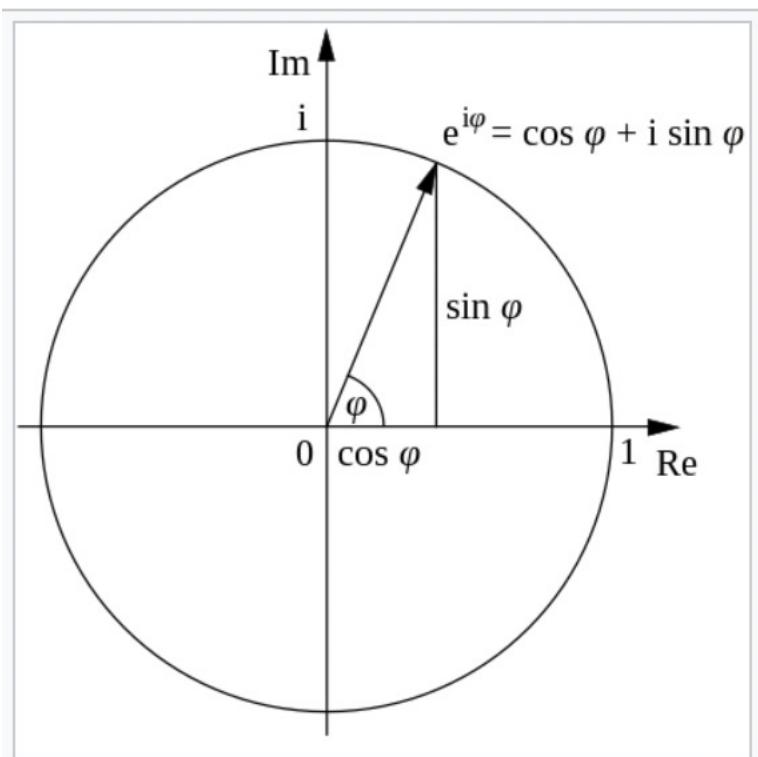
where  $e$  is the [base of the natural logarithm](#),  $i$  is the [imaginary unit](#), and  $\cos$  and  $\sin$  are the [trigonometric functions cosine](#) and [sine](#) respectively. This complex exponential function is sometimes denoted [cis](#)  $x$  ("cosine plus  $i$  sine"). The formula is still valid if  $x$  is a [complex number](#), and so some authors refer to the more general complex version as Euler's formula.<sup>[1]</sup>

Euler's formula is ubiquitous in mathematics, physics, and engineering. The physicist [Richard Feynman](#) called the equation "our jewel" and "the most remarkable formula in mathematics".<sup>[2]</sup>

When  $x = \pi$ , Euler's formula may be rewritten as  $e^{i\pi} + 1 = 0$  or  $e^{i\pi} = -1$ , which is known as [Euler's identity](#).

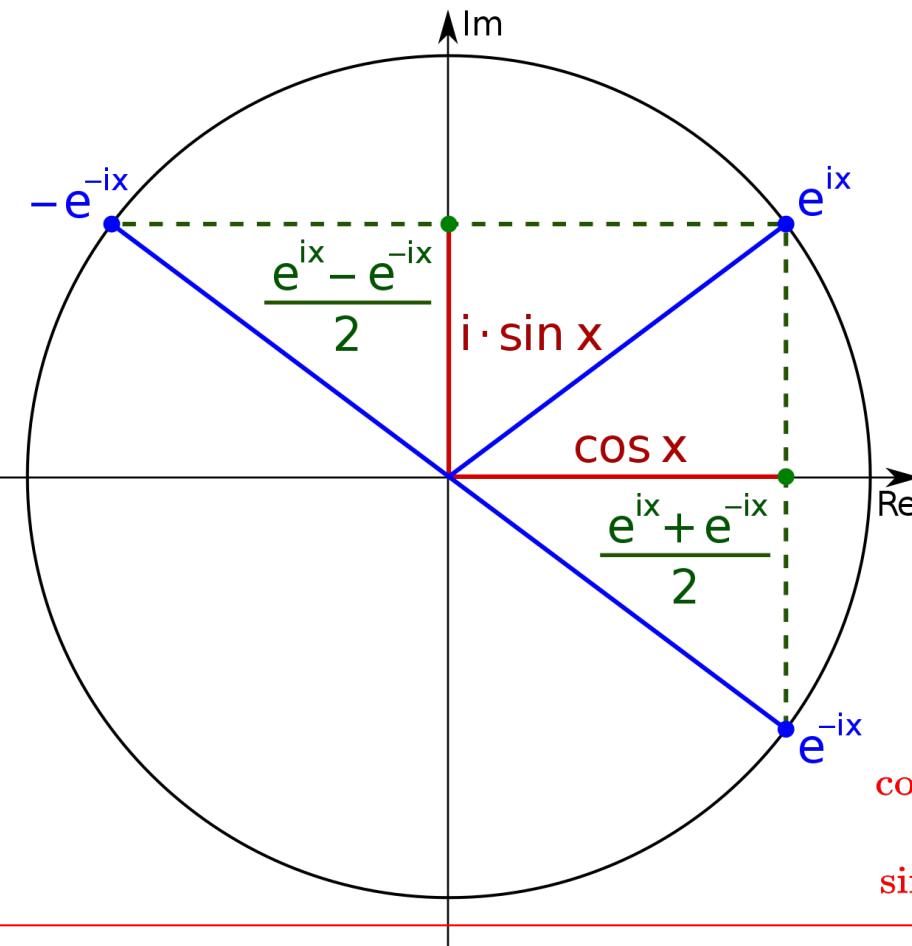


## Mathematics – Recap Log/Ln, Complex Numbers, Euler's formula, Polar vs Cartesian, Trigonometry, ....



Euler's formula

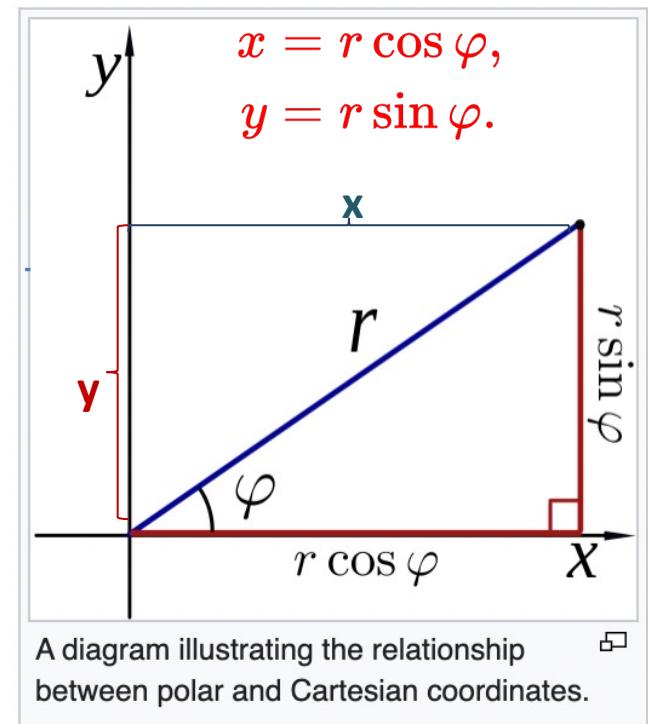
$e^{i\varphi} = \cos \varphi + i \sin \varphi$  illustrated in the complex plane.



where

- $x = \operatorname{Re} z$  is the real part,
- $y = \operatorname{Im} z$  is the imaginary part,
- $r = |z| = \sqrt{x^2 + y^2}$  is the magnitude of  $z$  and
- $\varphi = \arg z = \operatorname{atan2}(y, x)$ .

$\varphi$  is the argument of  $z$ , i.e., the angle between the  $x$  axis and the vector  $z$  measured counter-clock-wise in radians, which is defined up to addition of  $2\pi$ .



$$\cos x = \operatorname{Re}(e^{ix}) = \frac{e^{ix} + e^{-ix}}{2},$$

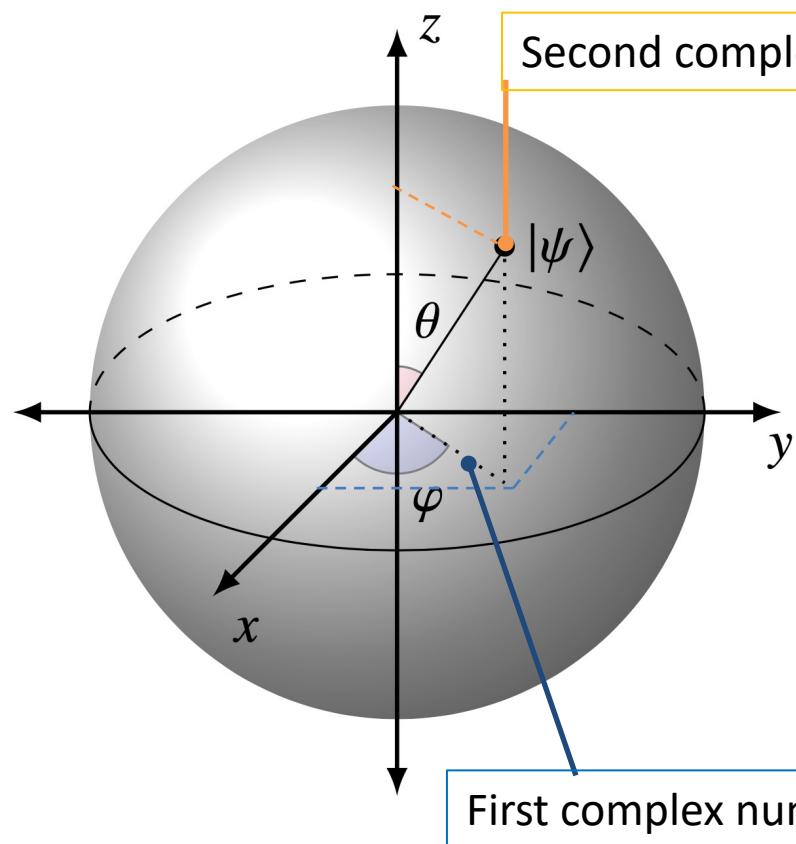
$$\sin x = \operatorname{Im}(e^{ix}) = \frac{e^{ix} - e^{-ix}}{2i}.$$

$$z = x + iy = |z|(\cos \varphi + i \sin \varphi) = re^{i\varphi},$$

$$\bar{z} = x - iy = |z|(\cos \varphi - i \sin \varphi) = re^{-i\varphi},$$

# Quantum Computing: Qubit – Bloch Sphere

[https://www.sharetechnote.com/html/QC/QuantumComputing\\_BlochSphere.html](https://www.sharetechnote.com/html/QC/QuantumComputing_BlochSphere.html)



For a single qubit, we have a unit sphere in  $\mathbb{C}^2$  with the quantum state  $a|0\rangle + b|1\rangle$  such that  $|a|^2 + |b|^2 = 1$ . The state can be re-written as  $|\cos \theta|^2 + |\sin \theta|^2 = 1$ , or  $|a|^2 = \cos^2 \theta$  and  $|b|^2 = \sin^2 \theta$ . Note:  $|a|^2$  is the probability of measuring  $|0\rangle$  and  $|b|^2$  is the probability of measuring  $|1\rangle$ .

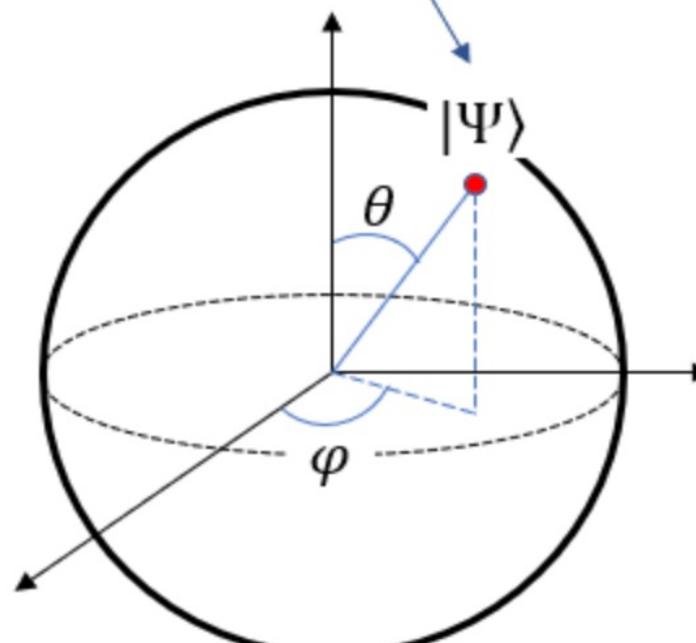
[https://en.wikipedia.org/wiki/Quantum\\_logic\\_gate](https://en.wikipedia.org/wiki/Quantum_logic_gate)

## Definition of Bloch Sphere

Simply put, Bloch Sphere is a way to represent a qubit states in a 3D spherical coordinates. As you know, Spherical Coordinates is a system defined by R (radius), theta and phi. In Bloch sphere, R is fixed to 1. Only Phi and Theta are variable. The qbit state function in the Bloch Sphere is defined as follows.

The absolute value (magnitude) of this term is always 1 regardless of the value  $\varphi$ . (i.e, the magnitude of  $\alpha$  and  $\beta$  is determined by  $\theta$  only)

$$|\Psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi} \sin\left(\frac{\theta}{2}\right)|1\rangle$$



$$0 \leq \theta \leq \pi$$

$$0 \leq \varphi \leq 2\pi$$

## Quantum Computing: Qubit – Bloch Sphere

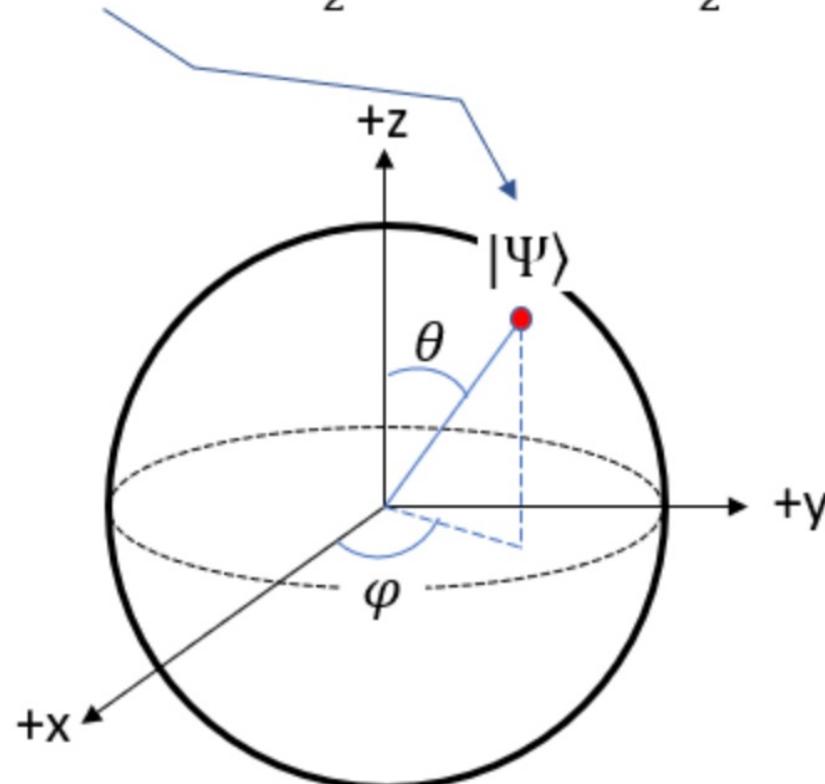
[https://www.sharetechnote.com/html/QC/QuantumComputing\\_BlochSphere.html](https://www.sharetechnote.com/html/QC/QuantumComputing_BlochSphere.html)

In case not familiar with Euler form of the equation and for easy calculation, we can rewrite the above formula as the equation [2] shown below.

$$|\Psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + (\sin(\varphi) + i \cos(\varphi)) \cdot \sin\left(\frac{\theta}{2}\right)|1\rangle \quad \text{--- [2]}$$

||

$$|\Psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi} \sin\left(\frac{\theta}{2}\right)|1\rangle \quad \text{--- [1]}$$



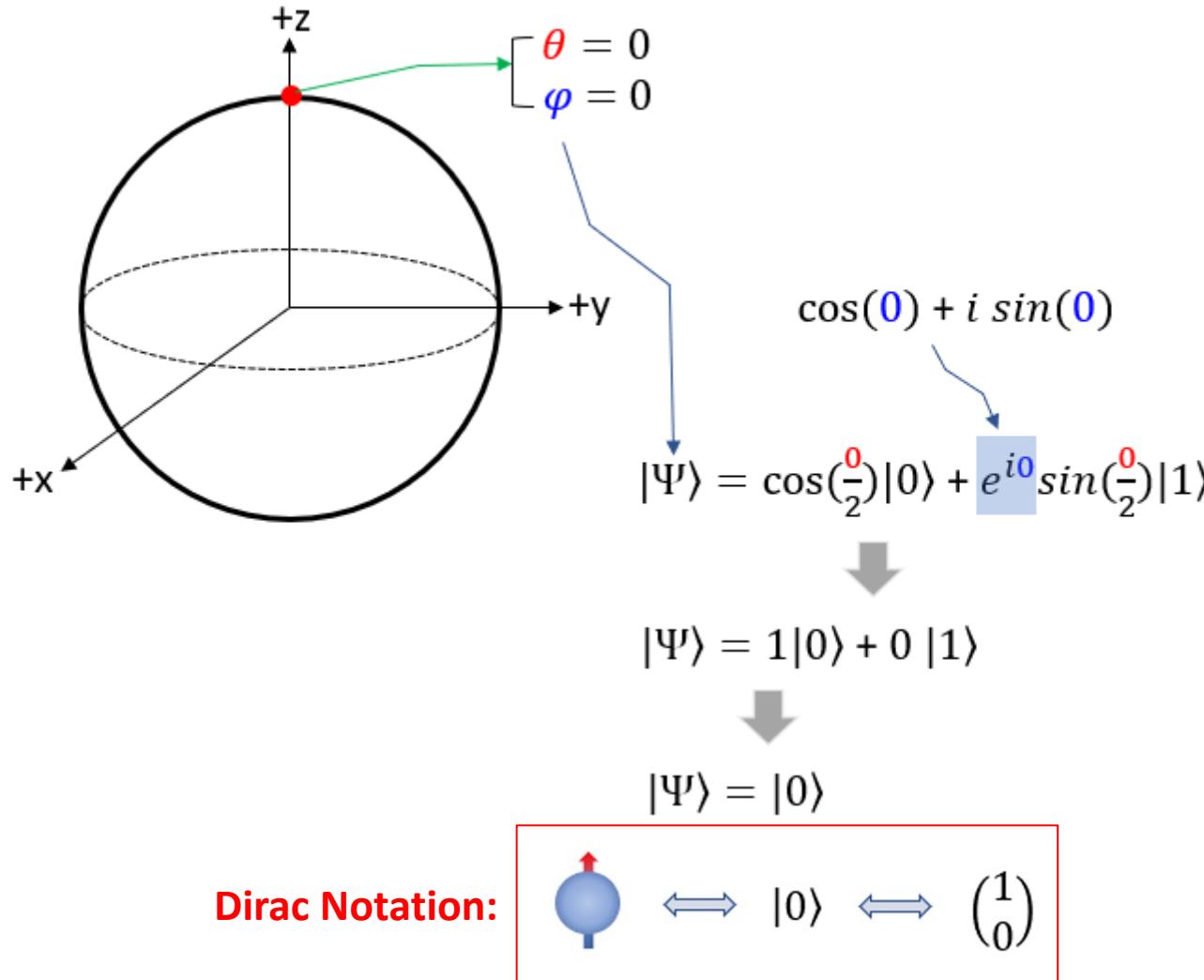
Even though the qbit state function on Bloch sphere looks a little bit complicated, it is still in the form of probability for two qbit state  $|0\rangle$  and  $|1\rangle$ . The probability of  $|0\rangle$  and  $|1\rangle$  (alpha and beta) is transformed into a little bit complicated form in Bloch sphere as shown below. Here you would notice that the probability of  $|1\rangle$  can be expressed in complex number.

$$\begin{aligned} |\Psi\rangle &= \boxed{\alpha}|0\rangle + \boxed{\beta}|1\rangle \\ |\Psi\rangle &= \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi} \sin\left(\frac{\theta}{2}\right)|1\rangle \end{aligned}$$

# Quantum Computing: Qubit – Bloch Sphere

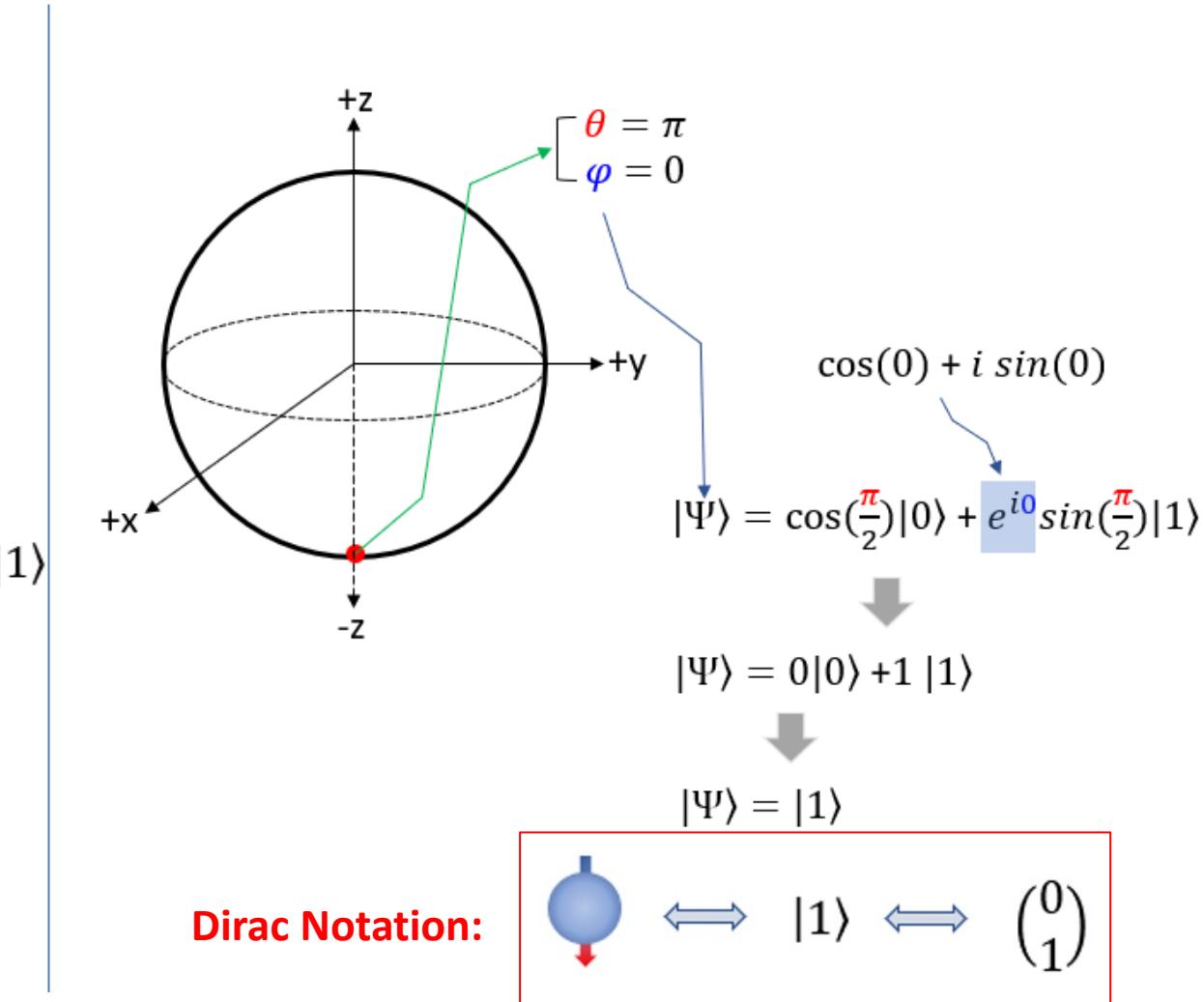
## Bloch Vectors on each Axis

Let's take the point on the circumference that crosses the  $+z$  axis. In spherical coordinate, this point is where  $\theta = 0$  and  $\phi = 0$ . If you plug these values into Bloch sphere state function you will get  $|0\rangle$  as shown below.

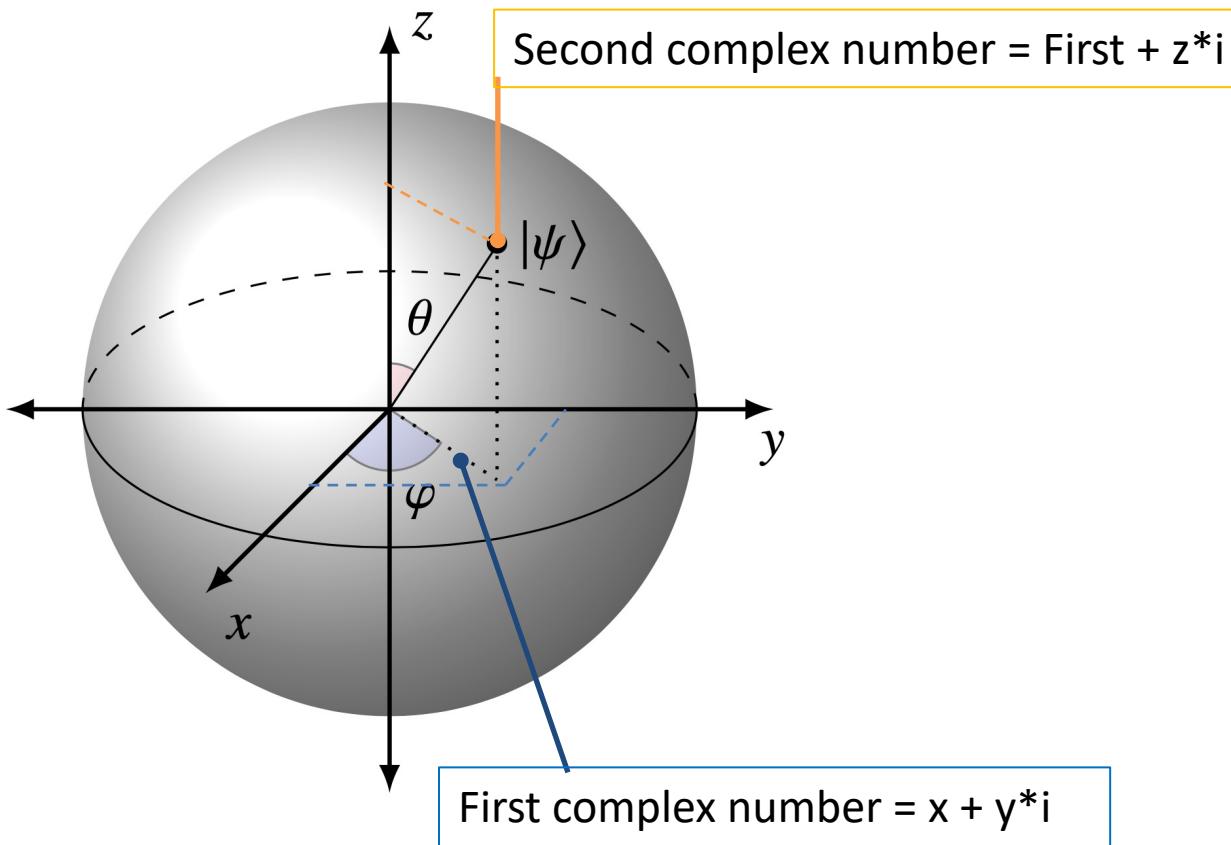


[https://www.sharetechnote.com/html/QC/QuantumComputing\\_BlochSphere.html](https://www.sharetechnote.com/html/QC/QuantumComputing_BlochSphere.html)

Now Let's take the point on the circumference that crosses the  $-z$  axis. In spherical coordinate, this point is where  $\theta = \pi$  and  $\phi = 0$ . If you plug these values into Bloch sphere state function you will get  $|1\rangle$  as shown below.



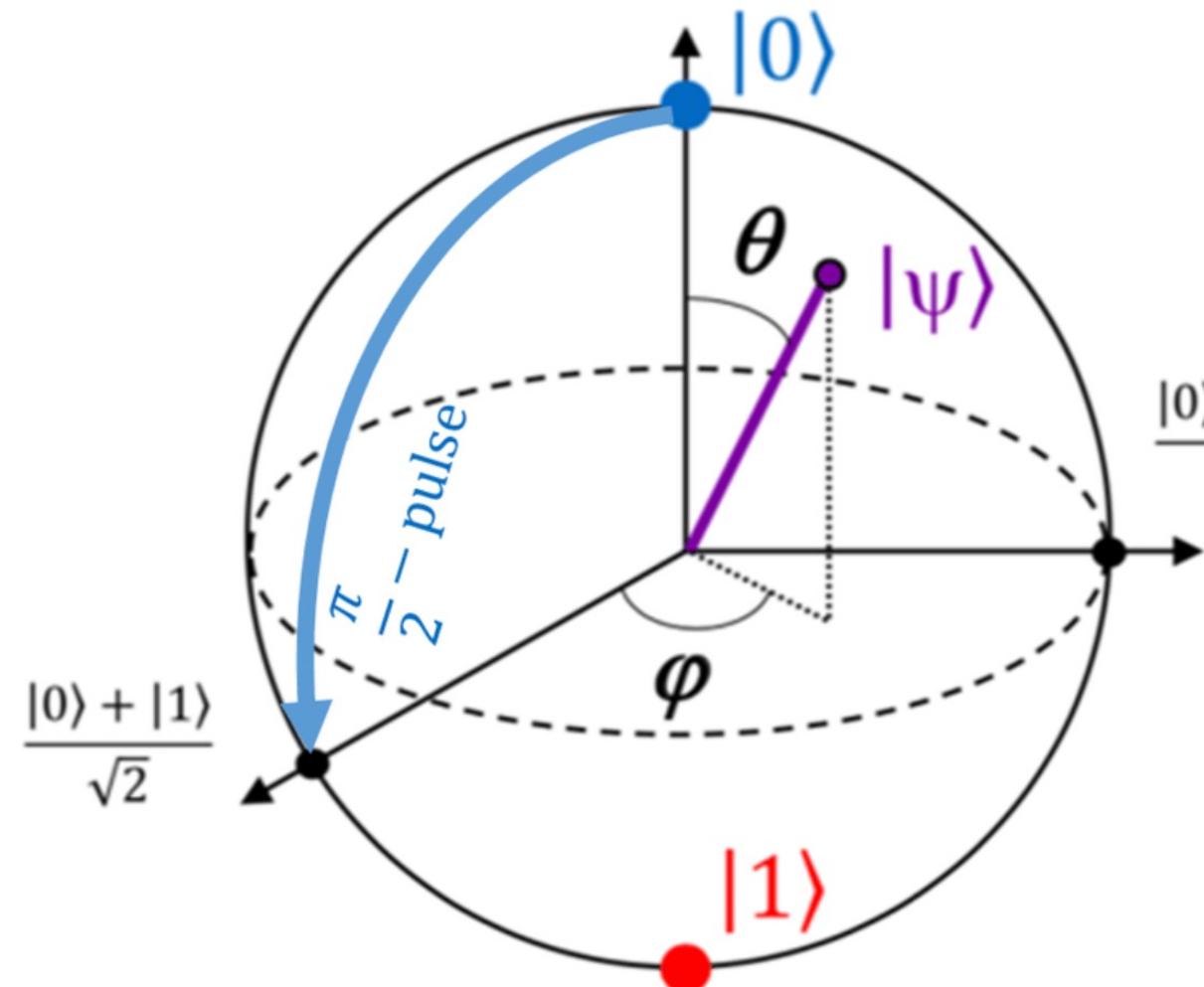
## Quantum Computing: Qubit – Bloch Sphere



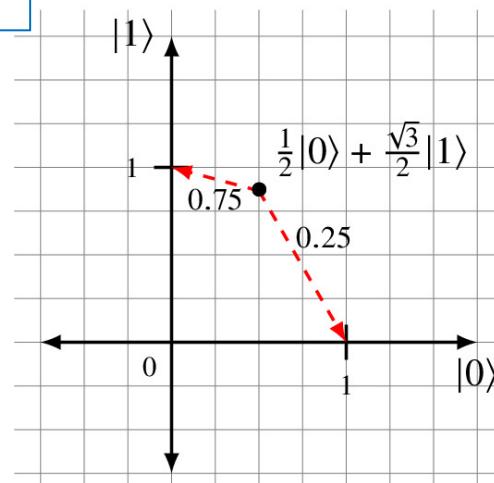
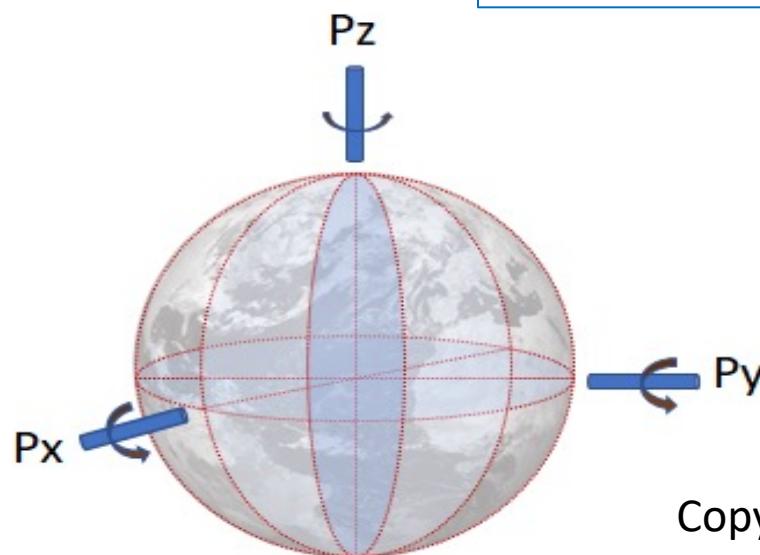
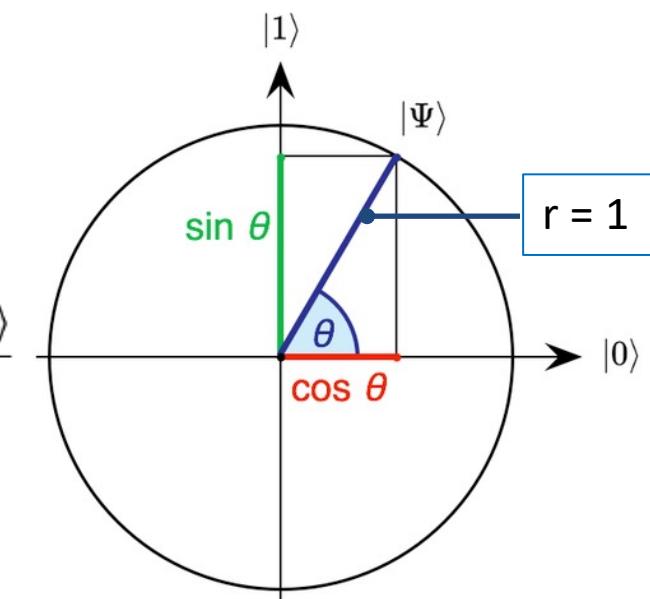
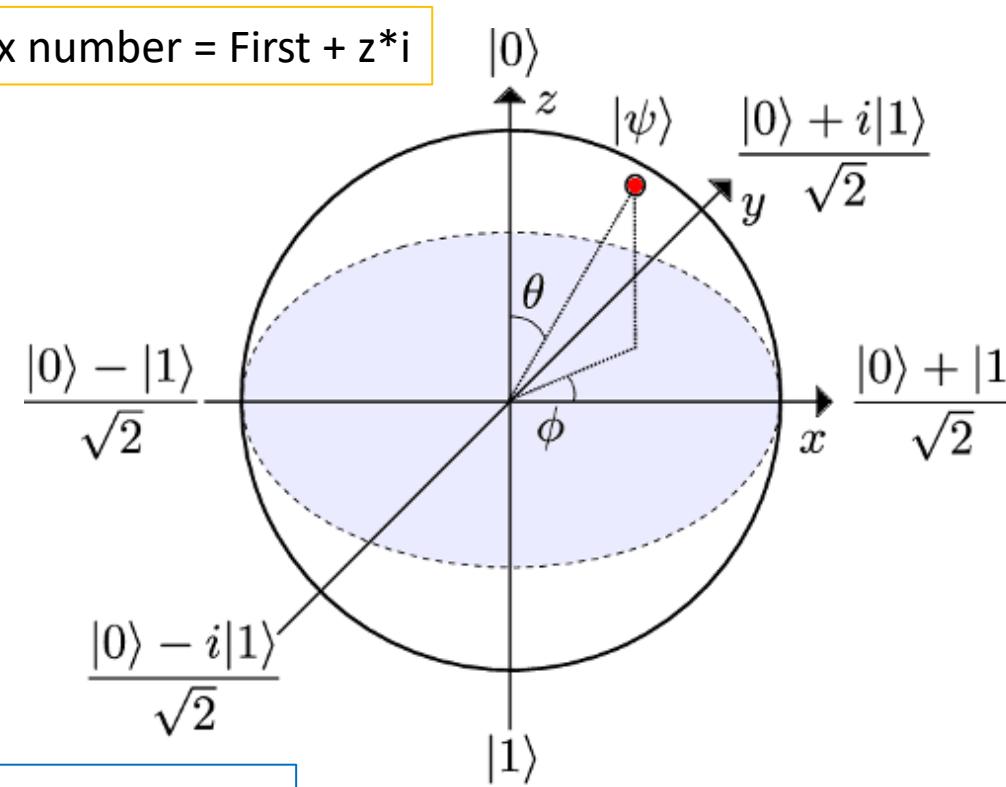
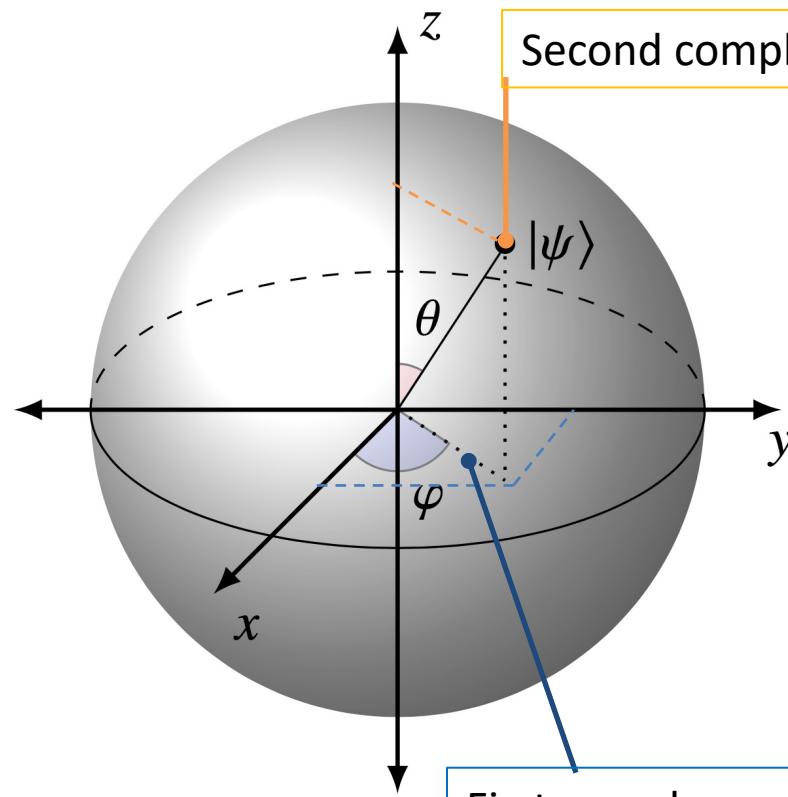
For a single qubit, we have a unit sphere in  $\mathbb{C}^2$  with the quantum state  $a|0\rangle + b|1\rangle$  such that  $|a|^2 + |b|^2 = 1$ . The state can be re-written as  $|\cos \theta|^2 + |\sin \theta|^2 = 1$ , or  $|a|^2 = \cos^2 \theta$  and  $|b|^2 = \sin^2 \theta$ .

Note:  $|a|^2$  is the probability of measuring  $|0\rangle$  and  $|b|^2$  is the probability of measuring  $|1\rangle$ .

[https://en.wikipedia.org/wiki/Quantum\\_logic\\_gate](https://en.wikipedia.org/wiki/Quantum_logic_gate)



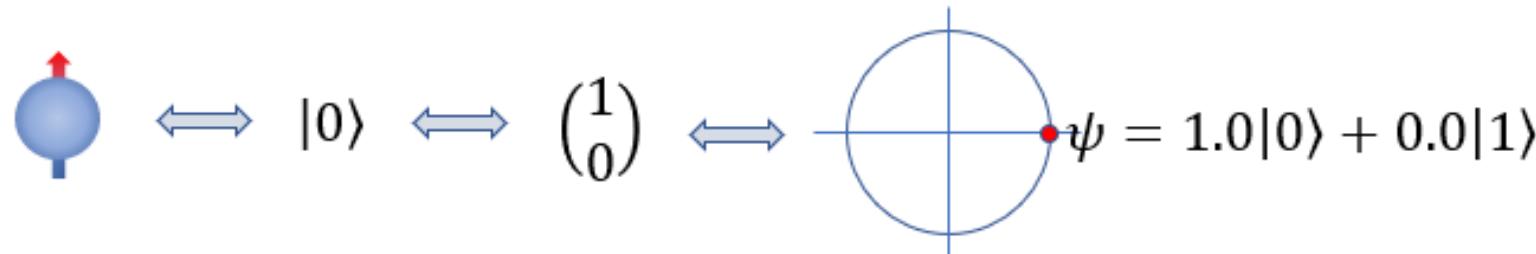
The Bloch sphere provides a useful means of visualizing the state of a single qubit and operations on it. Any point on this sphere represents a linear combination of the 0 and 1 states with complex coefficients. It 'rotates' a qubit from the 0-state to a superposition state.



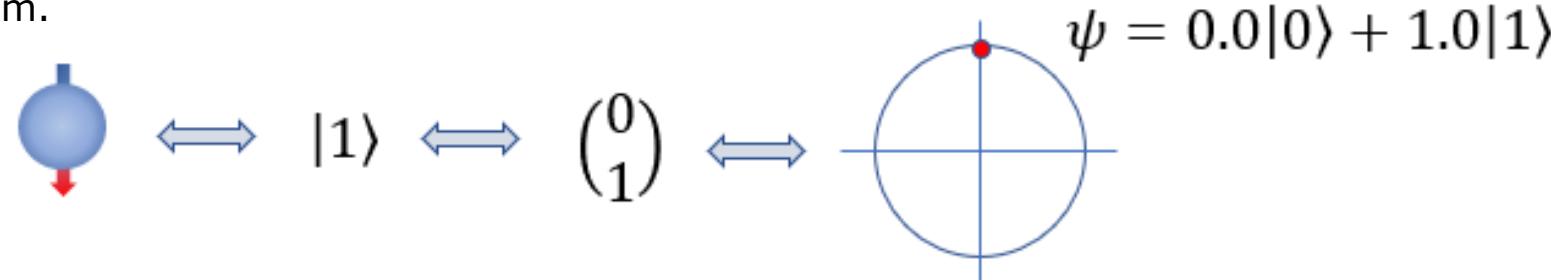
Copyright: [https://en.wikipedia.org/wiki/Quantum\\_logic\\_gate](https://en.wikipedia.org/wiki/Quantum_logic_gate)

For a single qubit, we have a unit sphere in  $\mathbb{C}^2$  with the quantum state  $a|0\rangle + b|1\rangle$  such that  $|a|^2 + |b|^2 = 1$ . The state can be re-written as  $|\cos \theta|^2 + |\sin \theta|^2 = 1$ , or  $|a|^2 = \cos^2 \theta$  and  $|b|^2 = \sin^2 \theta$ . Note:  $|a|^2$  is the probability of measuring  $|0\rangle$  and  $|b|^2$  is the probability of measuring  $|1\rangle$ .

The left most one is how upspin is represented graphically. The next strange symbol  $|>$  This symbol is called Bra-Ket (Dirac Notation).  $|0\rangle$  represents the quatom bit 0. And then you see a vector. This vector also represent the quantum bit 0. The last one also represents the quantum bit 0 in a probability function and in a coordinate system.



The left most one is how downspin is represented graphically.  $|1\rangle$  represents the quatom bit 1. And then you see a vector. This vector also represent the quantum bit 1. The last one also represents the quantum bit 1 in a probability function and in a coordinate system.



The horizontal axis of the coordinate system indicates the probability of the bit existing as a quantum bit 0 and the vertical axis indicate the probability of the bit existing as a quantum bit 1 as indicated below. Very important thing to note here is '**alpha and beta are complex numbers**'.

**Qbit state**

$$\psi = \alpha|0\rangle + \beta|1\rangle$$

The square of this number indicates the probability of the spin existing as  $|1\rangle$

The square of this number indicates the probability of the spin existing as  $|0\rangle$

This probability function (Quamtum State function) is represented in a vector notation as follows.

$$\psi = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

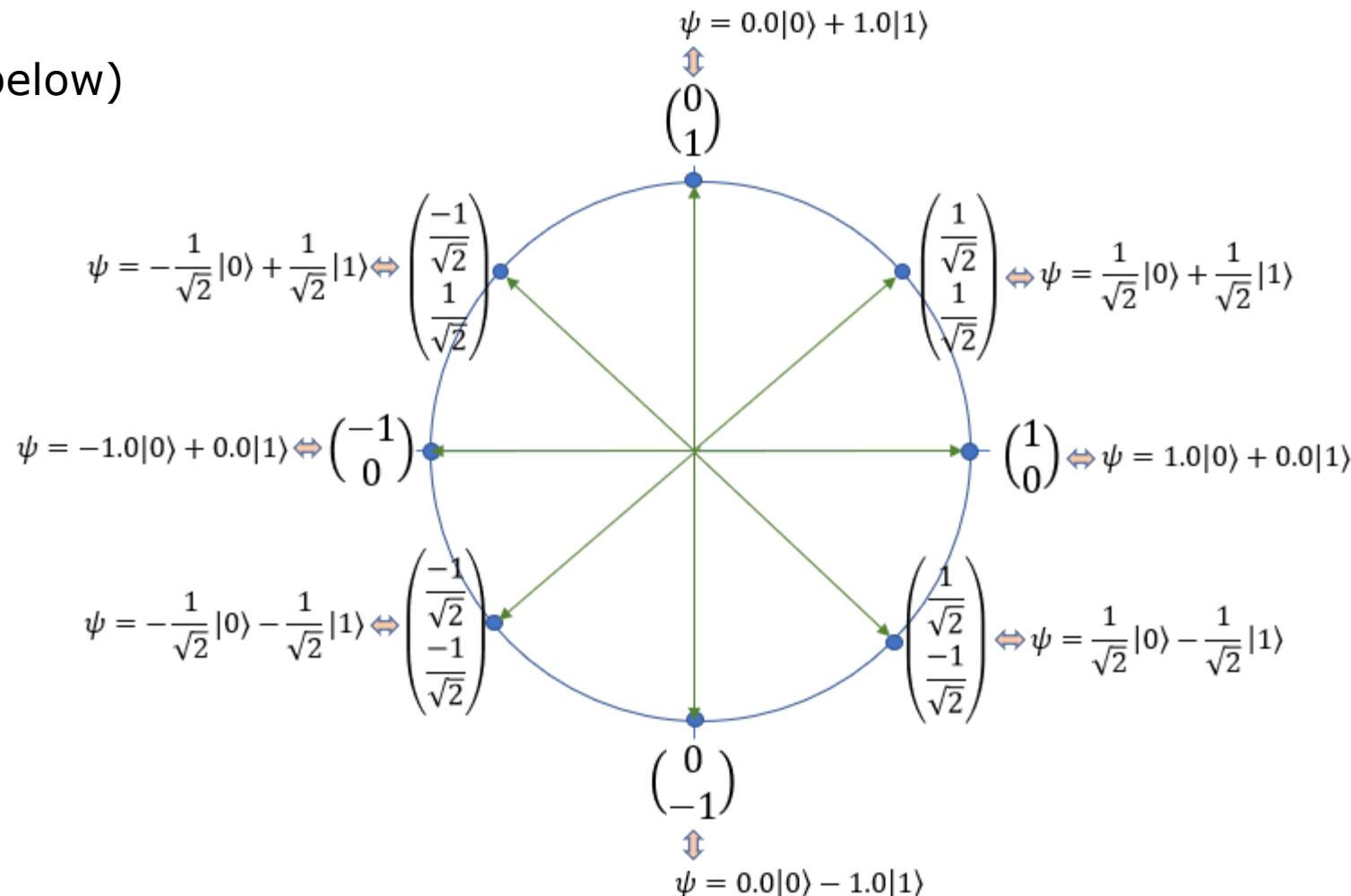
The number of this vector (defined as below) should be 1 as below.

$$\alpha^2 + \beta^2 = 1$$

If alpha and beta is complex numbers, the total probability is calculated as follows (i.e, multiplying each value with its complex conjugate).

$$\alpha\bar{\alpha} + \beta\bar{\beta} = 1$$

Since the norm of the state vector is 1, all the possible states represented by the state vector should all be on the peripheral of a unit circle. Some of the most commonly used state vector is mapped onto a unit circle as below. Try to translate each location marked here in plain language. For example,  $[1/\text{Sqr}(2) 1/\text{Sqr}(2)]$  indicates that the bit would be existing as the quantum bit 0 with the probability of 50% and be existing as the quantum bit 1 with the probability of 50%.



## Mathematics – Recap Log/Ln, Complex Numbers, Euler's number e, Polar vs Cartesian, Linear Algebra, Trigonometry, ....

In [linear algebra](#), a **rotation matrix** is a transformation matrix that is used to perform a [rotation](#) in [Euclidean space](#). For example, using the convention below, the matrix

$$R = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

rotates points in the  $xy$  plane counterclockwise through an angle  $\theta$  about the origin of a two-dimensional [Cartesian coordinate system](#). To perform the rotation on a plane point with standard coordinates  $\mathbf{v} = (x, y)$ , it should be written as a [column vector](#), and multiplied by the matrix  $R$ :

$$R\mathbf{v} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{bmatrix}.$$

If  $x$  and  $y$  are the endpoint coordinates of a vector, where  $x$  is cosine and  $y$  is sine, then the above equations become the [trigonometric summation angle formulae](#). Indeed, a rotation matrix can be seen as the trigonometric summation angle formulae in matrix form. One way to understand this is say we have a vector at an angle  $30^\circ$  from the  $x$  axis, and we wish to rotate that angle by a further  $45^\circ$ . We simply need to compute the vector endpoint coordinates at  $75^\circ$ .

The examples in this article apply to [active rotations](#) of vectors *counterclockwise* in a *right-handed coordinate system* ( $y$  counterclockwise from  $x$ ) by [pre-multiplication](#) ( $R$  on the left). If any one of these is changed (such as rotating axes instead of vectors, a [passive transformation](#)), then the [inverse](#) of the example matrix should be used, which coincides with its [transpose](#).

## Relationship with complex plane [ edit ]

Since

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}^2 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} = -I,$$

the matrices of the shape

$$\begin{bmatrix} x & y \\ -y & x \end{bmatrix}$$

form a [ring isomorphic](#) to the [field](#) of the [complex numbers](#)  $\mathbb{C}$ . Under this isomorphism, the rotation matrices correspond to [circle](#) of the [unit complex numbers](#), the complex numbers of modulus 1.

If one identifies  $\mathbb{R}^2$  with  $\mathbb{C}$  through the [linear isomorphism](#)  $(a, b) \mapsto a + ib$ , the action of a matrix of the above form on vectors of  $\mathbb{R}^2$  corresponds to the multiplication by the complex number  $x + iy$ , and rotations correspond to multiplication by complex numbers of modulus 1.

As every rotation matrix can be written

$$\begin{pmatrix} \cos t & \sin t \\ -\sin t & \cos t \end{pmatrix},$$

the above correspondence associates such a matrix with the complex number

$$\cos t + i \sin t = e^{it}$$

(this last equality is [Euler's formula](#)).

[https://en.wikipedia.org/wiki/Rotation\\_matrix](https://en.wikipedia.org/wiki/Rotation_matrix)

A basic rotation (also called elemental rotation) is a rotation about one of the axes of a coordinate system. The following three basic rotation matrices rotate vectors by an angle  $\theta$  about the  $x$ -,  $y$ -, or  $z$ -axis, in three dimensions, using the [right-hand rule](#)—which codifies their alternating signs. Notice that the right-hand rule only works when multiplying  $R \cdot \vec{x}$ . (The same matrices can also represent a clockwise rotation of the axes.[\[nb 1\]](#))

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For [column vectors](#), each of these basic vector rotations appears counterclockwise when the axis about which they occur points toward the observer, the coordinate system is right-handed, and the angle  $\theta$  is positive.  $R_z$ , for instance, would rotate toward the  $y$ -axis a vector aligned with the  $x$ -axis, as can easily be checked by operating with  $R_z$  on the vector  $(1,0,0)$ :

$$R_z(90^\circ) \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos 90^\circ & -\sin 90^\circ & 0 \\ \sin 90^\circ & \cos 90^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

This is similar to the rotation produced by the above-mentioned two-dimensional rotation matrix. See [below](#) for alternative conventions which may apparently or actually invert the sense of the rotation produced by these matrices.

## General rotations [ edit ]

Other rotation matrices can be obtained from these three using [matrix multiplication](#). For example, the product

$$R = R_z(\alpha) R_y(\beta) R_x(\gamma) = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \gamma & -\sin \gamma \\ 0 & \sin \gamma & \cos \gamma \end{bmatrix}$$

$$= \begin{bmatrix} \cos \alpha \cos \beta & \cos \alpha \sin \beta \sin \gamma - \sin \alpha \cos \gamma & \cos \alpha \sin \beta \cos \gamma + \sin \alpha \sin \gamma \\ \sin \alpha \cos \beta & \sin \alpha \sin \beta \sin \gamma + \cos \alpha \cos \gamma & \sin \alpha \sin \beta \cos \gamma - \cos \alpha \sin \gamma \\ -\sin \beta & \cos \beta \sin \gamma & \cos \beta \cos \gamma \end{bmatrix}$$

represents a rotation whose [yaw, pitch, and roll](#) angles are  $\alpha, \beta$  and  $\gamma$ , respectively. More formally, it is an [intrinsic rotation](#) whose [Tait–Bryan angles](#) are  $\alpha, \beta, \gamma$ , about axes  $z, y, x$ , respectively. Similarly, the product

$$R = R_z(\gamma) R_y(\beta) R_x(\alpha) = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}$$

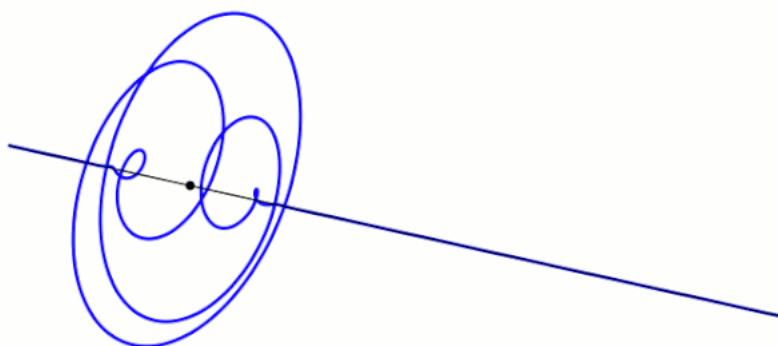
$$= \begin{bmatrix} \cos \beta \cos \gamma & \sin \alpha \sin \beta \cos \gamma - \cos \alpha \sin \gamma & \cos \alpha \sin \beta \cos \gamma + \sin \alpha \sin \gamma \\ \cos \beta \sin \gamma & \sin \alpha \sin \beta \sin \gamma + \cos \alpha \cos \gamma & \cos \alpha \sin \beta \sin \gamma - \sin \alpha \cos \gamma \\ -\sin \beta & \sin \alpha \cos \beta & \cos \alpha \cos \beta \end{bmatrix}$$

represents an extrinsic rotation whose (improper) [Euler angles](#) are  $\alpha, \beta, \gamma$ , about axes  $x, y, z$ .

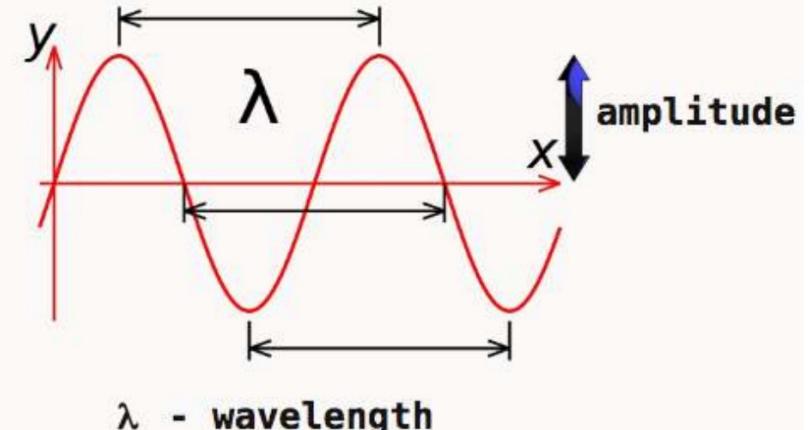
## Math Recap FINISHED

### Quantum statements:

- Once observation done in a qubit, there is no way to return to the previous state
- The obsevation affects the behaviour of the subatomic particles
- Each Gate is Unitary – applied twice is restoring the previous state of the q-bit
- There is no “clone”/“memcpy”/“strcpy” mechanism – the matter/subatomic particles cannot be copied
- There are a lot of specifics in quantum mechaniscs for qubits:
  - Superposition
  - Bell state / Entenglments between q-bits - “spookie action of the distance” (Einstein) with “Schröndniger Cat”
  - ...



### Frequency and Wavelength



$\lambda$  - wavelength

### Wave Speed

The speed of a wave can be calculated from its wavelength (measured in meters) and its frequency (measured in  $s^{-1}$ ):

$$\text{speed} = \text{wavelength (m)} \times \text{frequency (s}^{-1}\text{)}$$

For electromagnetic radiation this is:

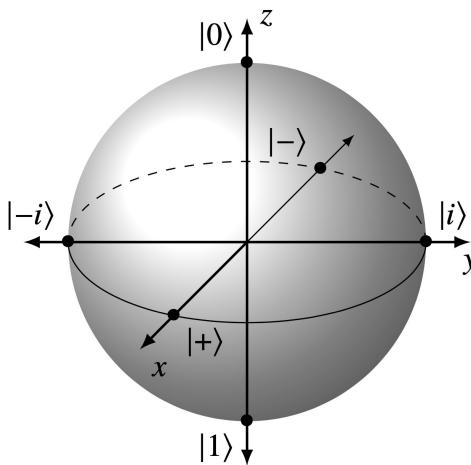
$$c = \lambda \times \nu$$

c - speed of light  
 $\lambda$  - wavelength  
 $\nu$  - frequency

## Quantum Gates

When we considered rotations we saw that the matrix for  $\mathbb{R}^3$  that does a rotation around the  $x$  axis by  $\theta$  radians works like

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \cos(\theta) - z \sin(\theta) \\ y \sin(\theta) + z \cos(\theta) \end{bmatrix}$$



Plugging in  $\theta = \pi$ , the rotation matrix is

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

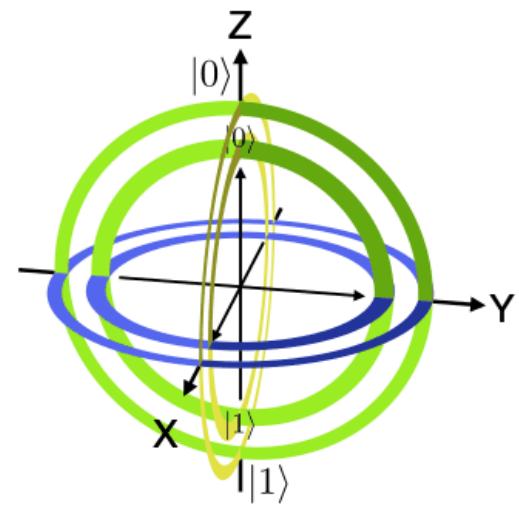
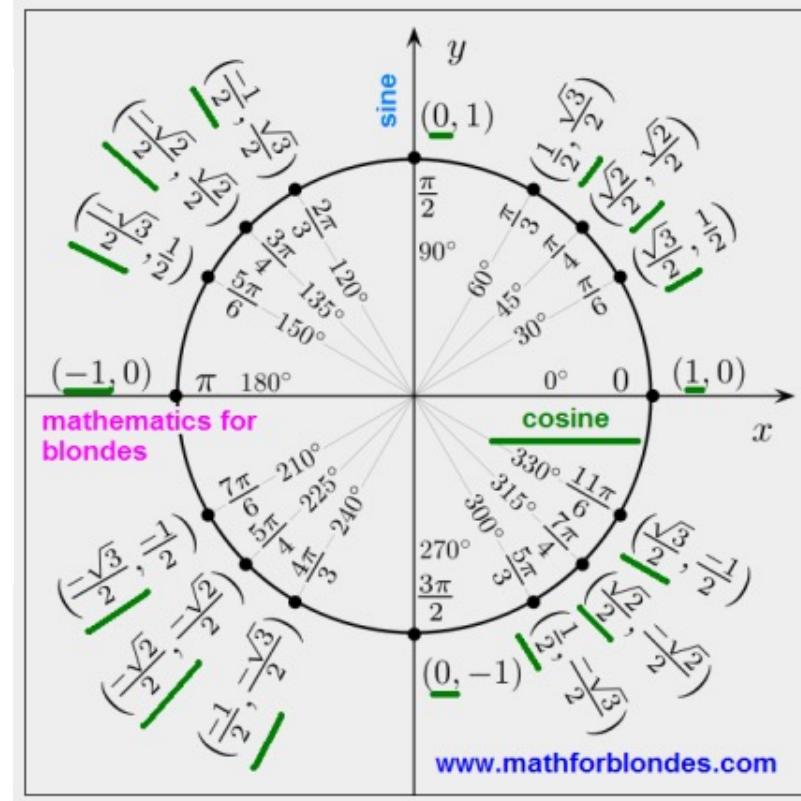
In  $\mathbb{R}^3$  standard coordinates,  $|0\rangle = (0, 0, 1)$  and  $|1\rangle = (0, 0, -1)$ . Applying the above matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}.$$

You can see this flipped  $|0\rangle$  and  $|1\rangle$ . What about  $|+\rangle$  and  $|-\rangle$ ?

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \end{bmatrix}.$$

As you would expect from looking at the geometry of the Bloch sphere, it leaves these alone.



## Quantum Gates

The **X** gate has the matrix

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

and this is the Pauli **X** matrix, named after Wolfgang Pauli. By “abuse of notation” I often use the same name (in this case **X**) for both the gate and its unitary matrix in the standard basis kets.

It has the property that

$$\sigma_x |0\rangle = |1\rangle \quad \text{and} \quad \sigma_x |1\rangle = |0\rangle .$$

It “flips” between  $|0\rangle$  and  $|1\rangle$ . The classic **not** gate is



The **not** gate is a “bit flip” and by analogy we also say **X** is a bit flip.

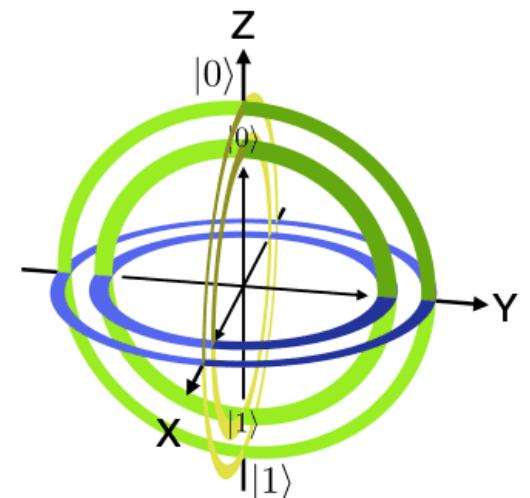
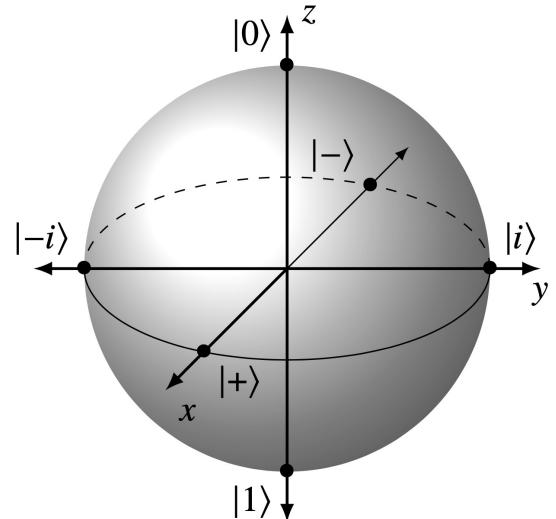
For  $|\psi\rangle = a|0\rangle + b|1\rangle$  in  $\mathbb{C}^2$ ,

$$\mathbf{X}|\psi\rangle = b|0\rangle + a|1\rangle .$$

This reverses the probabilities of measuring  $|0\rangle$  and  $|1\rangle$ . In  $\mathbb{C}^2$ ,  $\sigma_x$  has eigenvalues  $+1$  and  $-1$  for eigenvectors  $|+\rangle$  and  $|-\rangle$ , respectively.

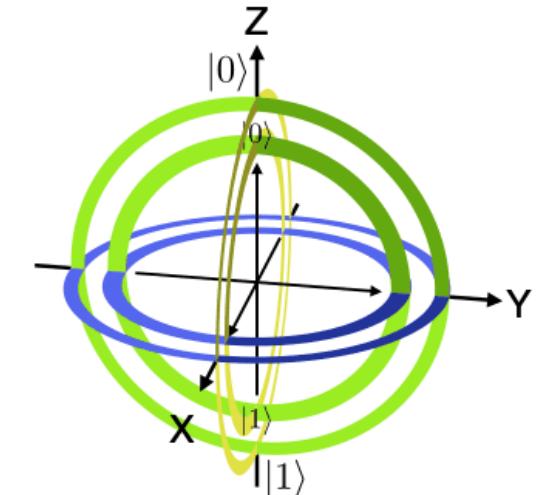
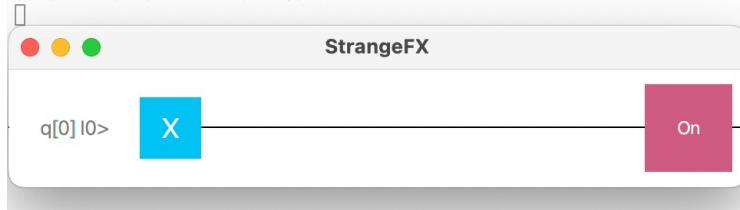
In terms of the Bloch sphere, the **X** gate rotates by  $\pi$  around the  $x$  axis. So not only are the poles flipped but points in the lower hemisphere move to the upper and vice versa.

Since  $\mathbf{X} \mathbf{X} = I_2$ , the **X** gate is its own inverse. This is reasonable because in the classical case it is also true that **not**  $\circ$  **not** is the identity operation.



<https://github.com/critoma/quantum/tree/main/quantumjava/ch03/paulix>

<https://github.com/critoma/quantum/tree/main/quantumjava/ch03/paulixui>



cristianvtoma @ g – IBM Quantum Composer:

<https://quantum-computing.ibm.com/composer/files/5e7652feec4ec71371f84637dd9e6557bbc758eac5d71fce4ca32a01d832a214>

<https://livebook.manning.com/book/quantum-computing-in-action/chapter-1/>

<https://livebook.manning.com/#!/book/quantum-computing-in-action/discussion>

F01\_PauliX | IBM Quantum Learning

quantum-computing.ibm.com/composer/files/5e7652feec4ec71371f84637dd9e6557bbc758eac5d71fce4ca32a01d832a214

Gmail YouTube Maps News Translate

IBM Quantum Learning Home Catalog Composer Lab

Jobs / cju8cr2k1qvd8te9ldv0 See more details

Completed Sep 09, 2023 5:51 PM (in 968ms)

Compute resource simulator\_stabilizer

Status timeline Completed

Details

Results

```
{ "quasi_dists": [ { "1": 1 } ], "metadata": [ { "shots": 1024, "readout_mitigation_overhead": 1, "readout_mitigation_time": 0.040719 } ] }
```

Operations F01\_PauliX Saved File Edit View Visualizations seed 4618 Setup and run

OpenQASM 2.0

Open in Quantum Lab

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[1];
creg c[1];
x q[0];
measure q[0] -> c[0];
```

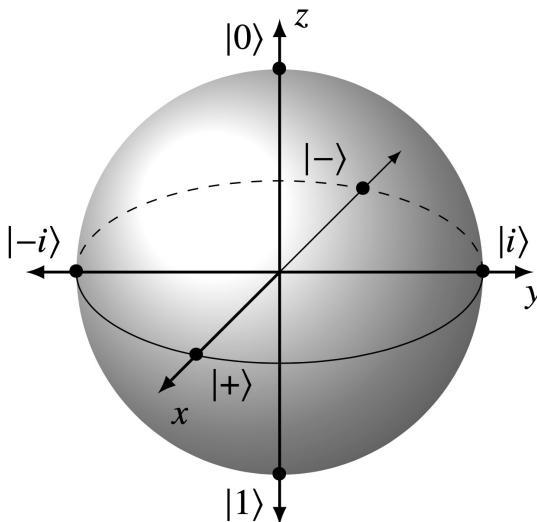
Probabilities Q-sphere

Computational basis states

Phase angle State

## Quantum Gates

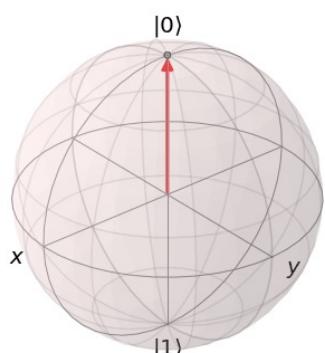
$$\begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \cos \theta + z \sin \theta \\ y \\ -x \sin \theta + z \cos \theta \end{bmatrix}$$



$$R_y(\pi) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$R_y(\pi)|0\rangle = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

$$R_y(\pi)|1\rangle = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

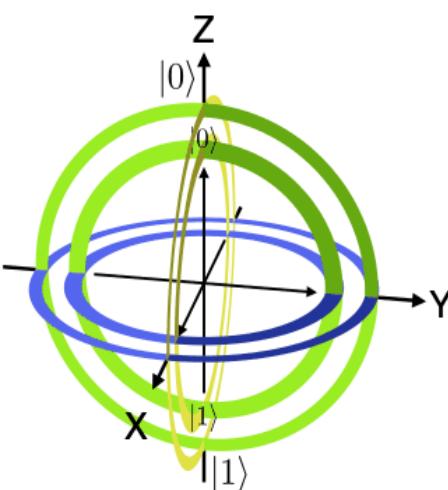
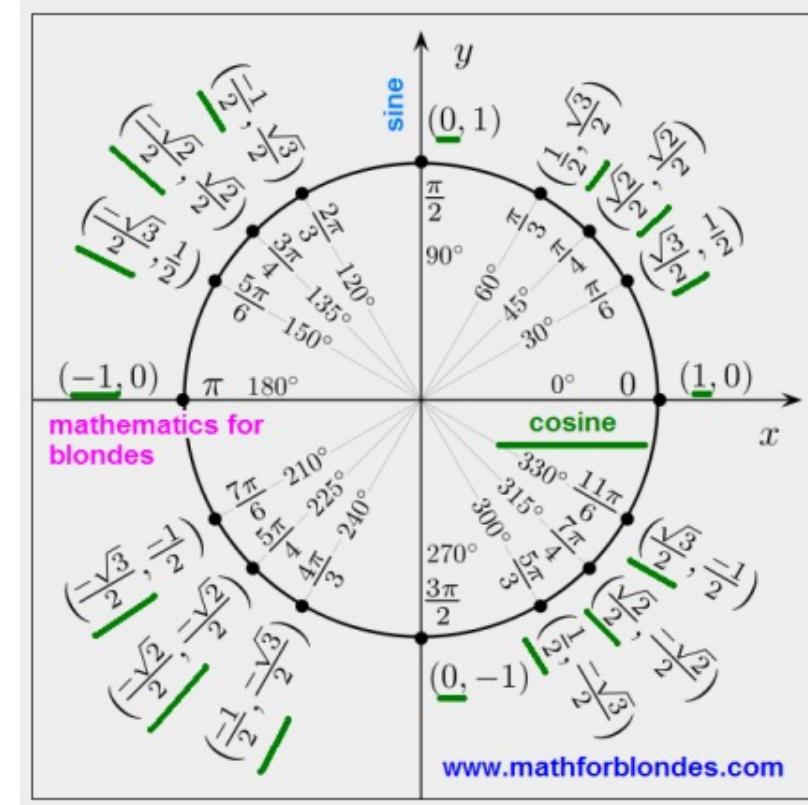


$$R_y(\pi)|i\rangle = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$R_y(\pi)|-i\rangle = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

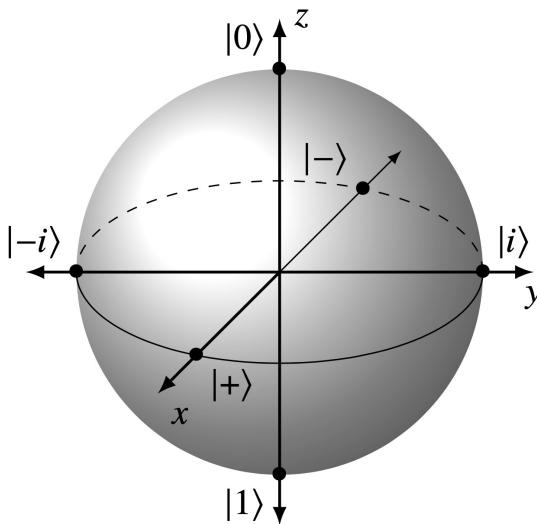
$$R_y(\pi)|+\rangle = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$$

$$R_y(\pi)|-\rangle = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$



## Quantum Gates

$$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \cos \theta - y \sin \theta \\ y \sin \theta + y \cos \theta \\ z \end{bmatrix}$$



$$R_z(\pi) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\pi)|0\rangle = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

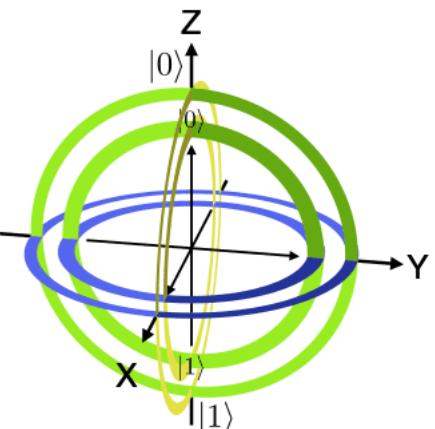
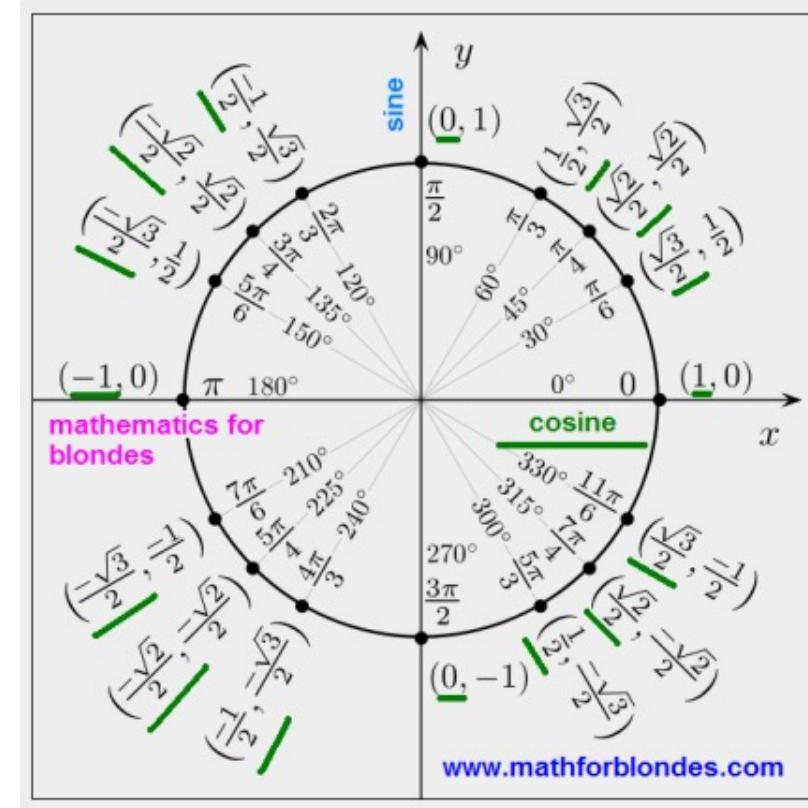
$$R_z(\pi)|1\rangle = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

$$R_z(\pi)|i\rangle = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

$$R_z(\pi)|-i\rangle = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$R_z(\pi)|+\rangle = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$$

$$R_z(\pi)|-\rangle = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$



Superposition

Superposition allows for much more processing power

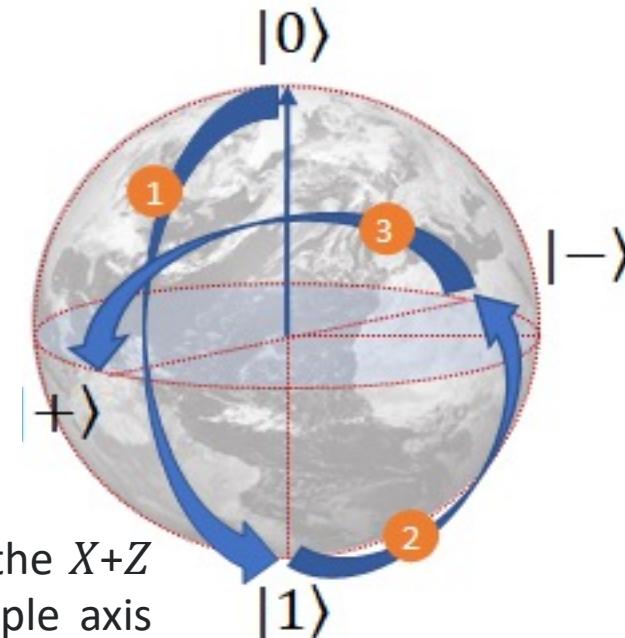
Processing power

Processing is achieved by applying gates

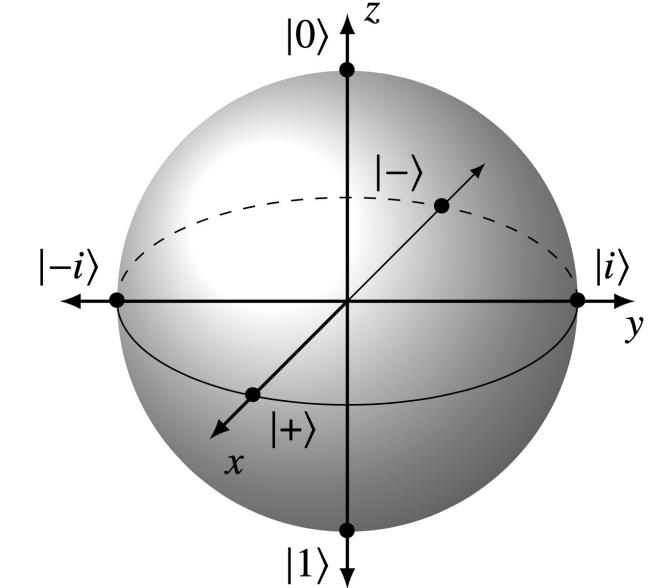
Gates

The Hadamard Gate allows for superposition

Hadamard Gate



## Quantum Gates



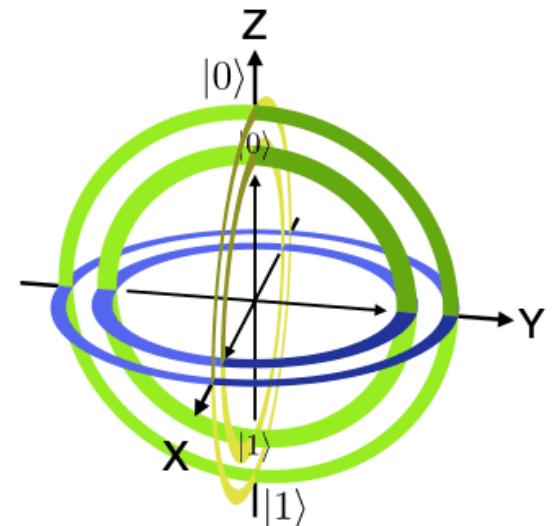
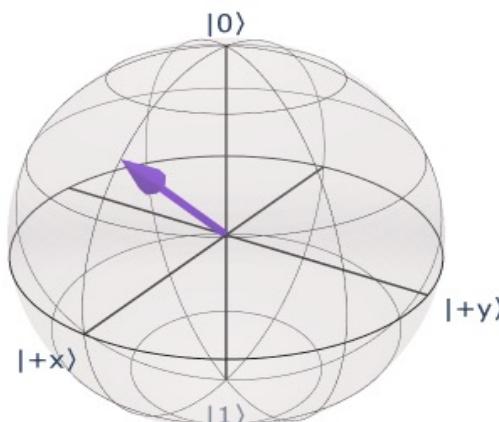
The Hadamard gate is a  $\pi$  rotation (180 degree) around the  $X+Z$  axis. That is, it is a 180 degree rotation around the purple axis indicated in the below figure, and the  $Rx(\pi/2)$  follows by a  $Rz(\pi/2)$ , then follow by  $Rx(\pi/2)$  does this rotation as well (slightly different).

The other way is analytically explained in this [pdf](#). This yields  $X-Z-X$

$$= e^{i\pi/2} R_x(\pi/2) R_z(\pi/2) R_x(\pi/2)$$

$$= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$= H$$



<https://quantumcomputing.stackexchange.com/questions/18031/how-to-visualize-hadamard-gate-as-x-z-x-decomposition>

[https://medium.com/@quantum\\_wa/quantum-computation-a-journey-on-the-bloch-sphere-50cc9d73530](https://medium.com/@quantum_wa/quantum-computation-a-journey-on-the-bloch-sphere-50cc9d73530)

## H Gate – Superposition:

<https://oreilly-qc.github.io/?p=2-1>

Run Program Ex 2-1: Random bit ▾ QCEngine ▾

```

1 // Programming Quantum Computers
2 // by Eric Johnston, Nic Harrigan and Mercedes Gimeno-Segovia
3 // O'Reilly Media
4
5 // To run this online, go to http://oreilly-qc.github.io?p=2-1
6
7 // This sample generates a single random bit.
8
9 qc.reset(1);           // allocate one qubit
10 qc.write(0);          // write the value zero
11 qc.had();             // place it into superposition of 0 and 1
12 var result = qc.read(); // read the result as a digital bit
13

```

Source code on Github [QCEngine](#) [Qiskit](#) [OpenQASM](#) [Q#](#) [Cirq](#)

cristianvtoma @ g – IBM Quantum Composer:

<https://quantum-computing.ibm.com/composer/files/c60278b623001dc0a6bdd7ca6154e27d3f945cbe76a0e7480795e38950d639f2>

## Quantum Gates

Program circuit

Circle notation

quantum-computing.ibm.com/composer/files/c60278b623001dc0a6bdd7ca6154e27d3f945cbe76a0e7480795e38950d639f2

IBM Quantum Learning Home Catalog Composer Lab

F02\_HGate Saved File Edit View Visualizations seed 4618 Setup and run

Operations: H, T, RZ, RXX, S, Z, T†, S†, P, RX, RY, RZX, U, RCCX, RC3X

q[0] H c1

Status timeline: Completed Sep 09, 2023 6:03 PM (in 6.2s), Compute resource simulator\_stabilizer, Running.

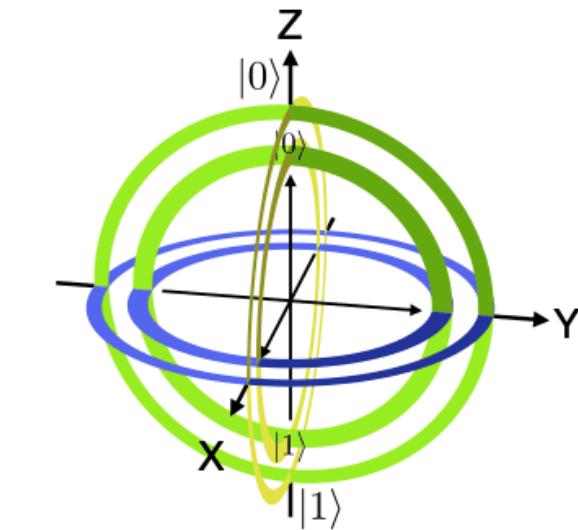
Probabilities: 0: ~50%, 1: ~50%

Q-sphere: State |0⟩, Phase angle 0, Phase 0, 3π/2, π/2.

```

1 OPENQASM 2.0;
2 include "qelib1.inc";
3
4 // Run this sample in the IBM
5 // at https://quantum-computing
6 // This sample generates a si
7
8 qreg q[1];
9 creg c[1];
10
11 h q[0];
12 measure q[0] -> c[0];
13
14

```



# Quantum Gates

<https://github.com/critoma/quantum/blob/main/quantumjava/ch04/hadamard/src/main/java/org/redfx/javaqc/ch04/hadamard/Main.java>

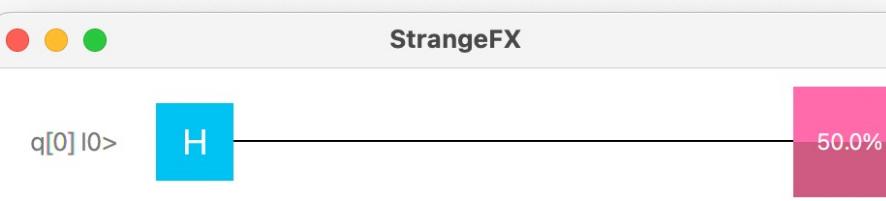
<https://github.com/critoma/quantum/blob/main/quantumjava/ch04/hadamard2/src/main/java/org/redfx/javaqc/ch04/hadamard2/Main.java>

```
critoma@critoma-mac java % # Hadamard - Superposition
# https://github.com/critoma/quantum/blob/main/quantumjava/ch04/hadamard/src/main/java/org/redfx/javaqc/ch04/hadamard/Main.java
cd $QHOME/ch04/hadamard/src/main/java
javac -cp .:$QJSTRANGELIB/target/strange-0.1.4-SNAPSHOT.jar:$QJSTRANGEFLIB/target/strangefx-0.1.5-SNAPSHOT.jar org/redfx/javaqc/ch04/hadamard/Main.java
java -cp ".:$QJSTRANGELIB/target/strange-0.1.4-SNAPSHOT.jar:$QJSTRANGEFLIB/target/strangefx-0.1.5-SNAPSHOT.jar:$JAVAFX_HOME/lib/*" --module-path $JAVAFX_HOME/lib --add-modules=javafx.controls,javafx.fxml org.redfx.javaqc.ch04.hadamard.Main
rm org.redfx.javaqc.ch04.hadamard/Main.class
```

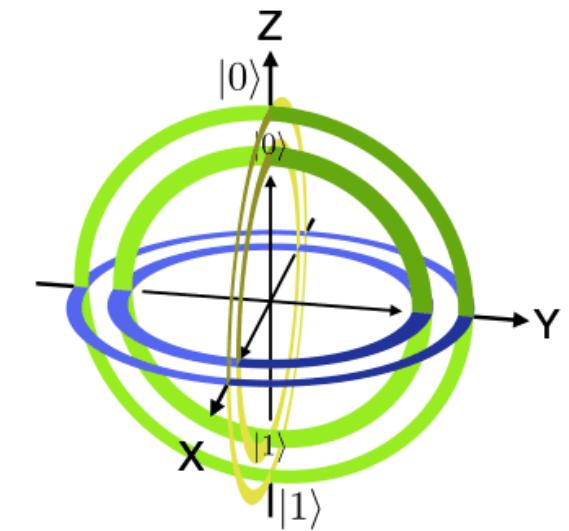
```
# Hadamard 2 - Superposition but ANY Quantum GATE is UNITARY => applied twice one after eaachother is coming to the previous state - coin
# https://github.com/critoma/quantum/blob/main/quantumjava/ch04/hadamard2/src/main/java/org/redfx/javaqc/ch04/hadamard2/Main.java
cd $QHOME/ch04/hadamard2/src/main/java
javac -cp .:$QJSTRANGELIB/target/strange-0.1.4-SNAPSHOT.jar:$QJSTRANGEFLIB/target/strangefx-0.1.5-SNAPSHOT.jar org/redfx/javaqc/ch04/hadamard2/Main.java
java -cp ".:$QJSTRANGELIB/target/strange-0.1.4-SNAPSHOT.jar:$QJSTRANGEFLIB/target/strangefx-0.1.5-SNAPSHOT.jar:$JAVAFX_HOME/lib/*" --module-path $JAVAFX_HOME/lib --add-modules=javafx.controls,javafx.fxml org.redfx.javaqc.ch04.hadamard2.
```

```
Main
zsh: command not found: #
zsh: command not found: #
=====
```

```
Single run of a Quantum Circuit with Hadamard Gate
Value = 0
JavaFX Platform initialized
=====
```



```
=====
1000 runs of a Quantum Circuit with Hadamard Gate
Applied Hadamard circuit 1000 times, got 486 times 0 and 514 times 1.
=====
```



## A.I. Bot – ChatGPT:

**“Qubits, the basic units of quantum information, can be stored in a variety of ways.**

One common method is to use a qubit that is encoded in the state of an atom or nucleus. This can be done by trapping the atom or nucleus in a magnetic or electric field, and then controlling the state of the atom or nucleus using laser or microwave pulses. Another method is to use a qubit that is encoded in the state of a superconducting circuit. This can be done by controlling the state of the superconducting circuit using microwave pulses. It's important to note that qubits are fragile, and they can be difficult to store and preserve due to the effects of noise and decoherence. Additionally, the technology for creating and manipulating qubits is still in its early stages, and there is ongoing research to find new ways to store and preserve qubits.”

**“Quantum gates are physical operations performed on qubits,** the fundamental building blocks of quantum computers. There are several ways to construct quantum gates in practice, depending on the specific physical platform used for the qubits. One common approach is to use superconducting circuits, which are made of thin films of superconducting materials that can conduct electrical current with zero resistance. These circuits can be used to create artificial atoms, known as qubits, that can be manipulated using microwave pulses to perform quantum gates. Another approach is to use trapped ions, which are charged atoms that are held in place using electromagnetic fields. Laser pulses can be used to manipulate the internal states of the ions to perform quantum gates. Yet another approach is to use quantum dots, which are small regions of semiconductor material that can trap electrons. By controlling the voltage applied to the quantum dots, the electrons can be manipulated to perform quantum gates. There are also other approaches to constructing quantum gates, such as using neutral atoms, photonic qubits, and more. However, the specific details of how quantum gates are constructed can vary greatly depending on the physical platform used for the qubits.”

A Quantum Engineer’s Guide to Superconducting Qubits (Y 2021)

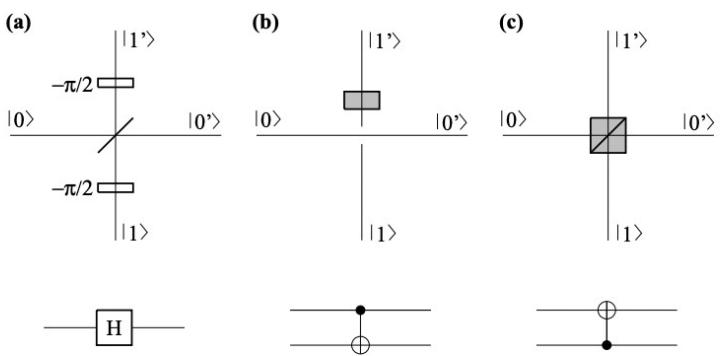
<https://arxiv.org/abs/1904.06560> | <https://arxiv.org/pdf/1904.06560.pdf>

Optical Simulation of Quantum Logic (Y 1997)

<https://arxiv.org/pdf/quant-ph/9706022.pdf>

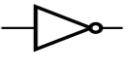
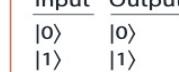
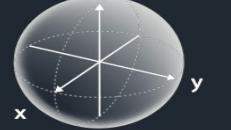
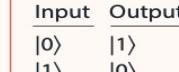
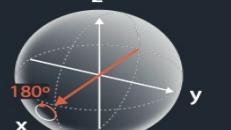
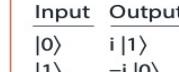
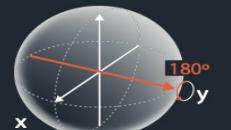
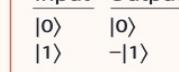
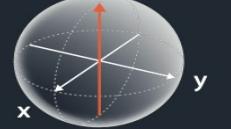
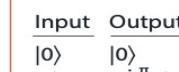
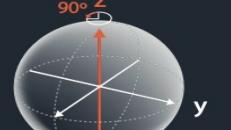
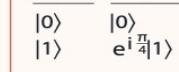
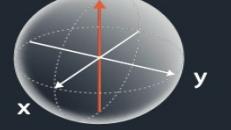
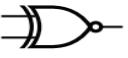
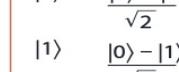
we use the fact that several (say  $n$ ) quantum bits can be represented by a *single* photon in an interferometric setup involving essentially  $2^n$  paths, so that conditional dynamics can easily be implemented by using different optical elements in distinct paths. The appropriate cascading of beam splitters and other linear optical devices entails the possibility of simulating networks of 1- and 2-bit quantum gates (such as the Hadamard or the controlled-NOT gate, see Fig. 1), and thereby in principle achieving universal  $n$ -bit quantum computations.

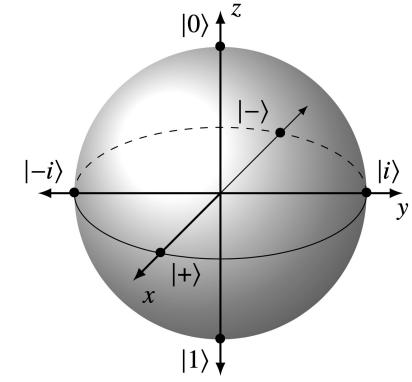
FIG. 1. Example of optical simulation of basic quantum logic gates. (a) Hadamard gate on a “location” qubit, using a lossless symmetric beam splitter. (b) Controlled-NOT gate using a polarization rotator. The location and polarization are the control and target qubit, respectively. (c) Same as (b) but the control and target qubits are interchanged by the use of a polarizing beam splitter.



This is in contrast with traditional optical models of quantum logic, where in general  $n$  photons interacting through nonlinear devices (acting as 2-bit quantum gates) are required to represent  $n$  qubits (see, e.g., [6]).

# Quantum Gates

GATE	CIRCUIT SYMBOL	TRUTH TABLE	GATE	CIRCUIT REPRESENTATION	MATRIX REPRESENTATION	TRUTH TABLE	BLOCH SPHERE
NOT	The output is 1 when the input is 0 and 0 when the input is 1.			$I$	$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$		
AND	The output is 1 only when both inputs are 1, otherwise the output is 0.			$X$	$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$		
OR	The output is 0 only when both inputs are 0, otherwise the output is 1.			$Y$	$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$		
NAND	The output is 0 only when both inputs are 1, otherwise the output is 1.			$Z$	$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$		
NOR	The output is 1 only when both inputs are 0, otherwise the output is 0.			$S$	$S = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix}$		
XOR	The output is 1 only when the two inputs have different value, otherwise the output is 0.			$T$	$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$		
XNOR	The output is 1 only when the two inputs have the same value, otherwise the output is 0.			$H$	$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$		

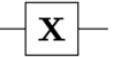
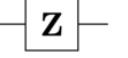
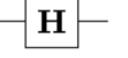
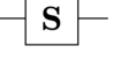
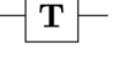
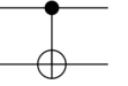
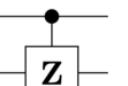
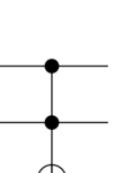


A Quantum Engineer's Guide to Superconducting Qubits (Y 2021)

<https://arxiv.org/abs/1904.06560> | <https://arxiv.org/pdf/1904.06560.pdf>

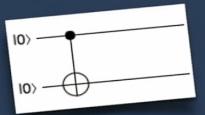
Optical Simulation of Quantum Logic (Y 1997)

<https://arxiv.org/pdf/quant-ph/9706022.pdf>

Operator	Gate(s)	Matrix	
<b>Pauli-X (X)</b>		$\oplus$	$ 0\rangle =  00\rangle =  0\rangle \otimes  0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$
<b>Pauli-Y (Y)</b>			$ 1\rangle =  01\rangle =  0\rangle \otimes  1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$
<b>Pauli-Z (Z)</b>			$ 2\rangle =  10\rangle =  1\rangle \otimes  0\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$
<b>Hadamard (H)</b>		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	$ 3\rangle =  11\rangle =  1\rangle \otimes  1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$
<b>Phase (S, P)</b>			
$\pi/8$ (T)			
<b>Controlled Not (CNOT, CX)</b>		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	
<b>Controlled Z (CZ)</b>		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$	
<b>SWAP</b>		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	
<b>Toffoli (CCNOT, CCX, TOFF)</b>		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$	

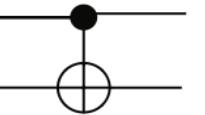
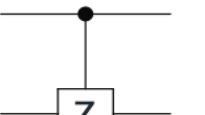
$$M \otimes N = \begin{bmatrix} M_{11} & \cdots & M_{1n} \\ \vdots & \ddots & \vdots \\ M_{m1} & \cdots & M_{mn} \end{bmatrix} \otimes \begin{bmatrix} N_{11} & \cdots & N_{1q} \\ \vdots & \ddots & \vdots \\ N_{p1} & \cdots & N_{pq} \end{bmatrix} = \begin{bmatrix} M_{11} \begin{bmatrix} N_{11} & \cdots & N_{1q} \\ \vdots & \ddots & \vdots \\ N_{p1} & \cdots & N_{pq} \end{bmatrix} & \cdots & M_{1n} \begin{bmatrix} N_{11} & \cdots & N_{1q} \\ \vdots & \ddots & \vdots \\ N_{p1} & \cdots & N_{pq} \end{bmatrix} \\ \vdots & \ddots & \vdots \\ M_{m1} \begin{bmatrix} N_{11} & \cdots & N_{1q} \\ \vdots & \ddots & \vdots \\ N_{p1} & \cdots & N_{pq} \end{bmatrix} & \cdots & M_{mn} \begin{bmatrix} N_{11} & \cdots & N_{1q} \\ \vdots & \ddots & \vdots \\ N_{p1} & \cdots & N_{pq} \end{bmatrix} \end{bmatrix}$$

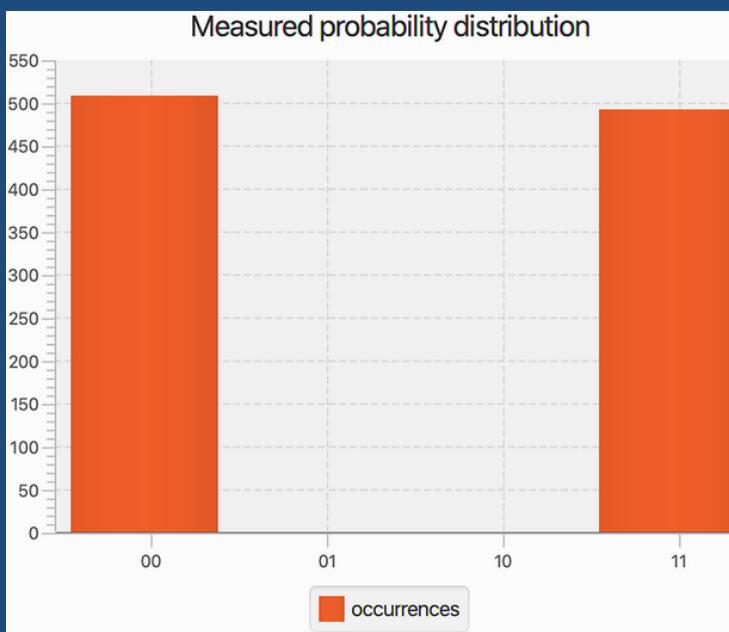
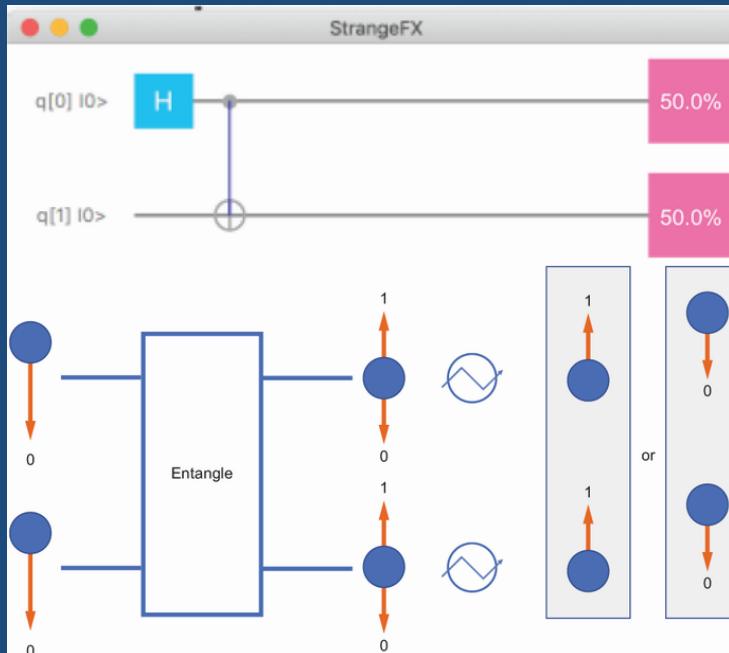
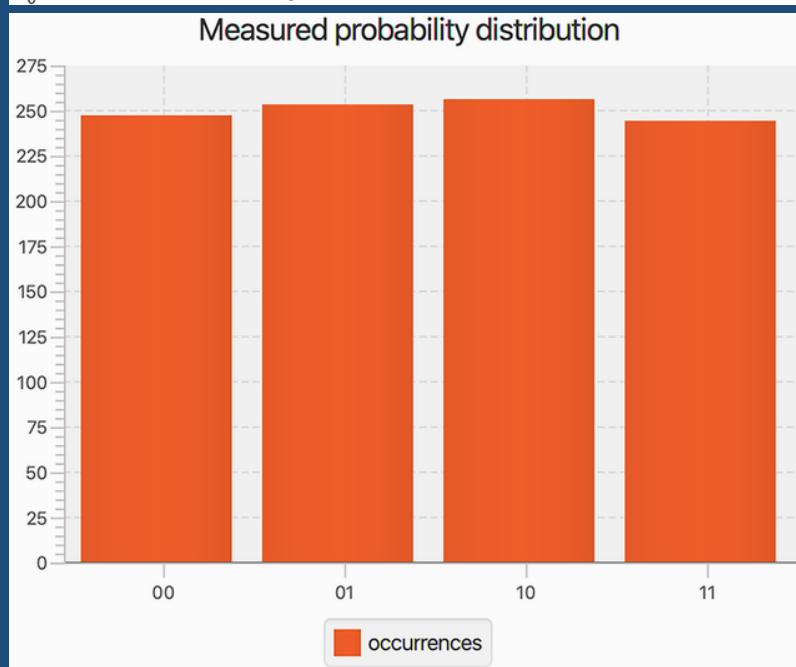
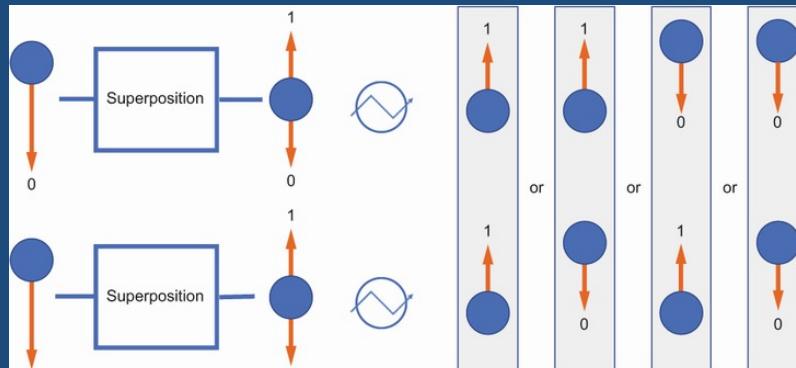
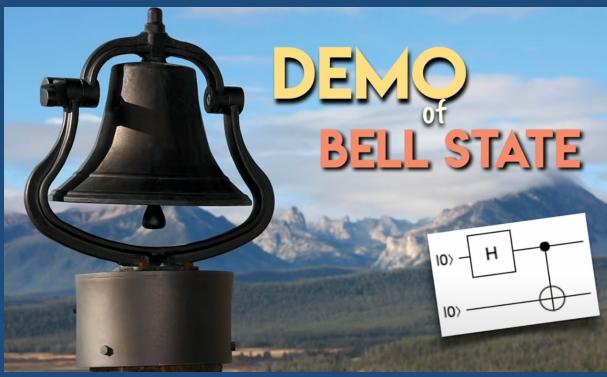
## CONTROLLED NOT GATE



$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

GATE	CIRCUIT REPRESENTATION	MATRIX REPRESENTATION	TRUTH TABLE										
Controlled-NOT gate: apply an X-gate to the target qubit if the control qubit is in state $ 1\rangle$		$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	<table border="1"> <thead> <tr> <th>Input</th><th>Output</th></tr> </thead> <tbody> <tr> <td><math> 00\rangle</math></td><td><math> 00\rangle</math></td></tr> <tr> <td><math> 01\rangle</math></td><td><math> 01\rangle</math></td></tr> <tr> <td><math> 10\rangle</math></td><td><math> 11\rangle</math></td></tr> <tr> <td><math> 11\rangle</math></td><td><math> 10\rangle</math></td></tr> </tbody> </table>	Input	Output	$ 00\rangle$	$ 00\rangle$	$ 01\rangle$	$ 01\rangle$	$ 10\rangle$	$ 11\rangle$	$ 11\rangle$	$ 10\rangle$
Input	Output												
$ 00\rangle$	$ 00\rangle$												
$ 01\rangle$	$ 01\rangle$												
$ 10\rangle$	$ 11\rangle$												
$ 11\rangle$	$ 10\rangle$												
Controlled-phase gate: apply a Z-gate to the target qubit if the control qubit is in state $ 1\rangle$		$\text{CPHASE} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$	<table border="1"> <thead> <tr> <th>Input</th><th>Output</th></tr> </thead> <tbody> <tr> <td><math> 00\rangle</math></td><td><math> 00\rangle</math></td></tr> <tr> <td><math> 01\rangle</math></td><td><math> 01\rangle</math></td></tr> <tr> <td><math> 10\rangle</math></td><td><math> 10\rangle</math></td></tr> <tr> <td><math> 11\rangle</math></td><td><math>- 11\rangle</math></td></tr> </tbody> </table>	Input	Output	$ 00\rangle$	$ 00\rangle$	$ 01\rangle$	$ 01\rangle$	$ 10\rangle$	$ 10\rangle$	$ 11\rangle$	$- 11\rangle$
Input	Output												
$ 00\rangle$	$ 00\rangle$												
$ 01\rangle$	$ 01\rangle$												
$ 10\rangle$	$ 10\rangle$												
$ 11\rangle$	$- 11\rangle$												



# Qubits circuits – Entanglement – Bell State

GitHub:

<https://github.com/critoma/quantum/tree/main/quantumjava/ch05>

IBM Composer:

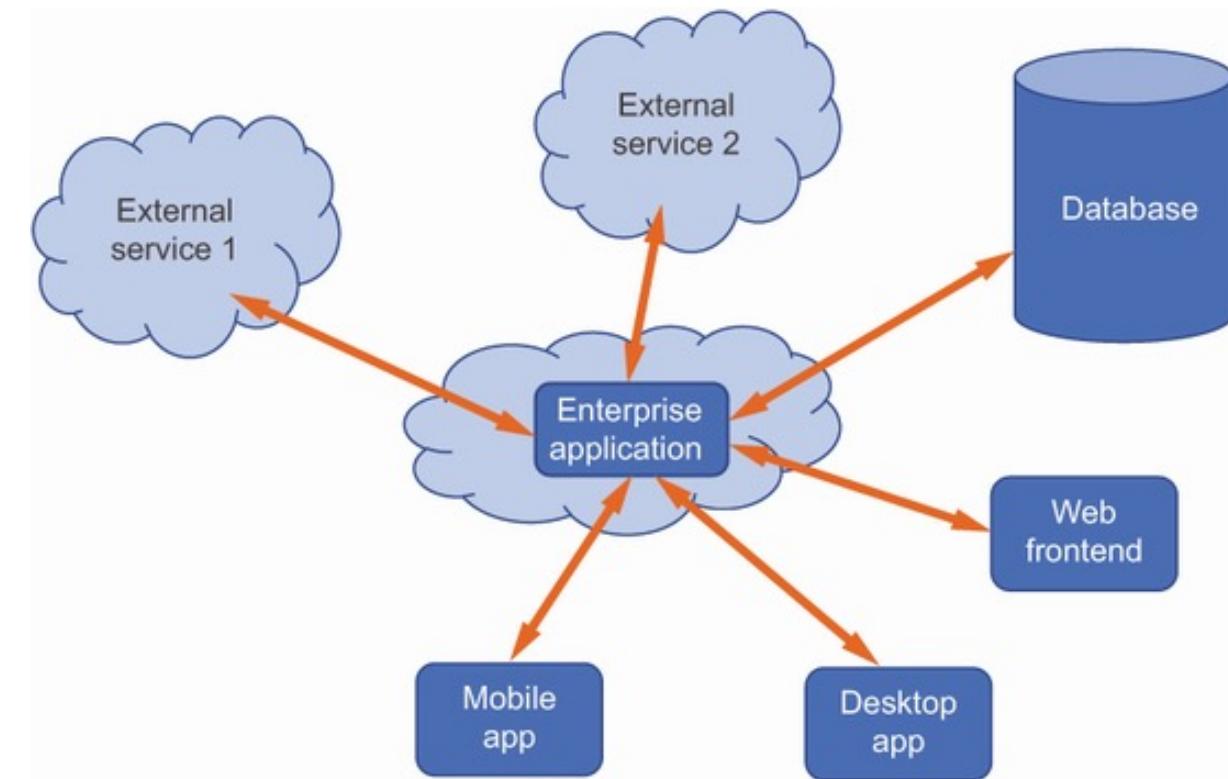
<https://quantum-computing.ibm.com/composer/files/48ebcac6f90ea9c882d0e9971336e002735198a9f741a5b50899fa89df6b2807>



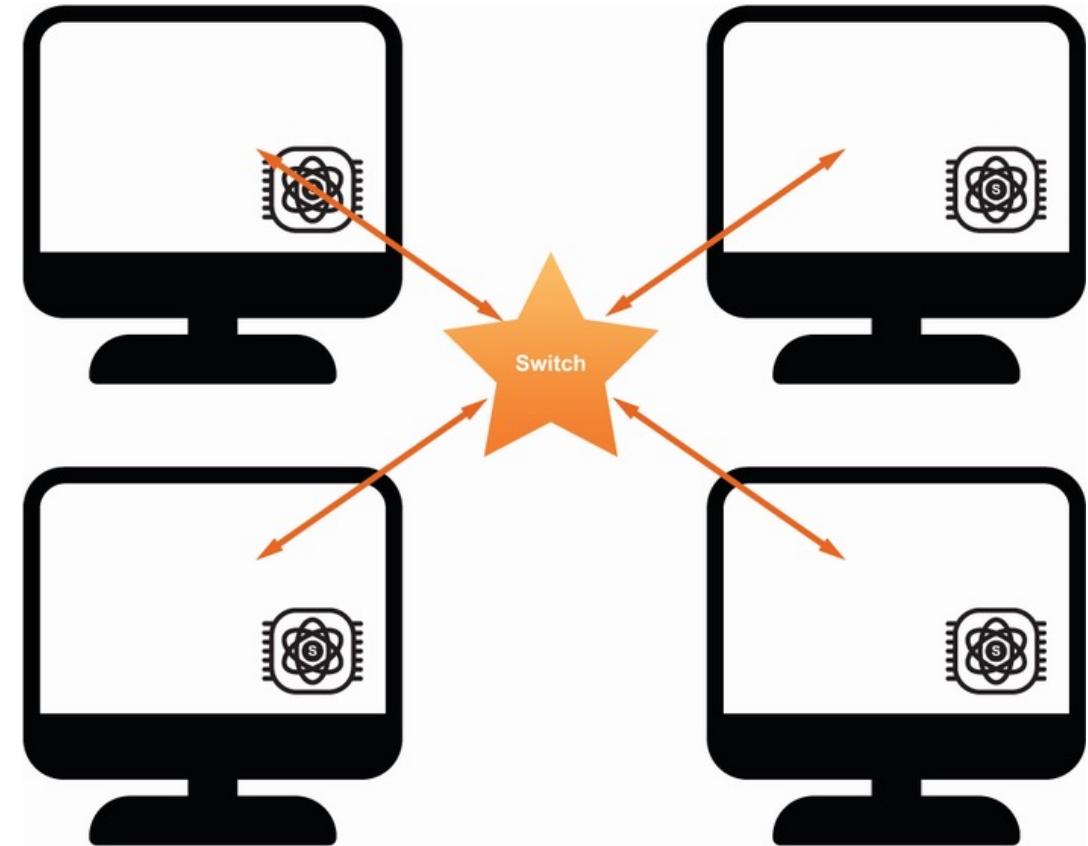
# How to construct a microwave trapped-ion quantum computer

# Quantum Networking

<https://github.com/critoma/quantum/blob/main/quantumjava/ch06/>

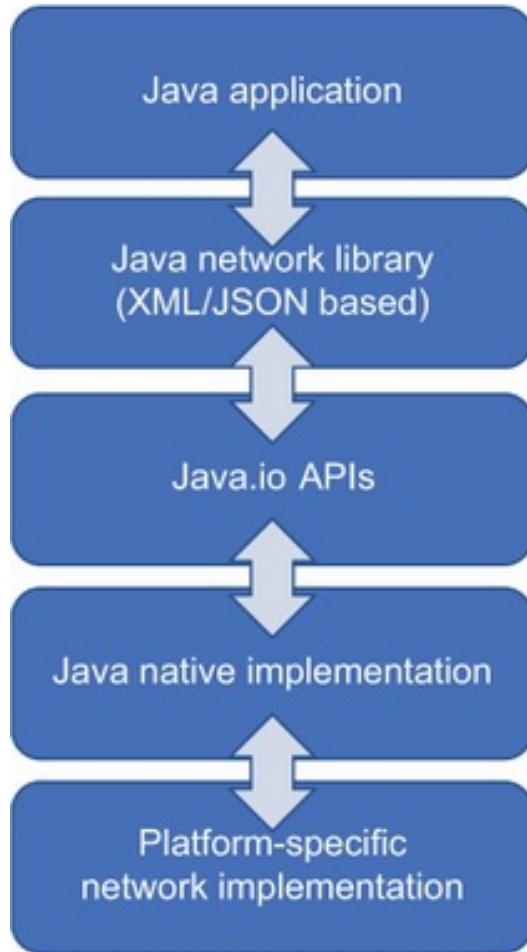


Classic application using modules in a network



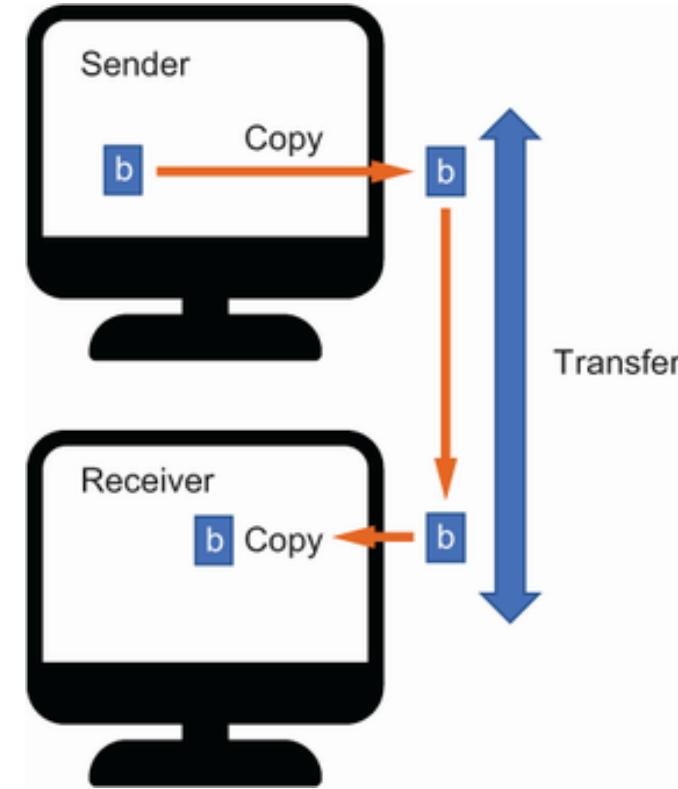
A quantum network combining a large number of small quantum computers

# Quantum Networking



Application	
Transport	Qubit transmission
Network	Long-distance entanglement
Link	Robust entanglement generation
Physical	Attempt entanglement generation

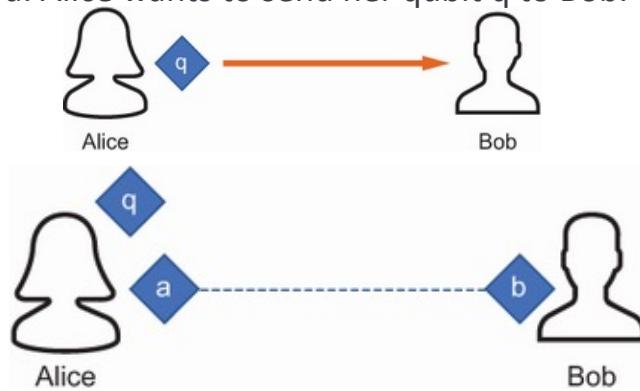
A QuTech proposal for a quantum network stack



# Quantum Networking - Quantum Teleportation

Copyright: <https://www.manning.com/books/quantum-computing-in-action>

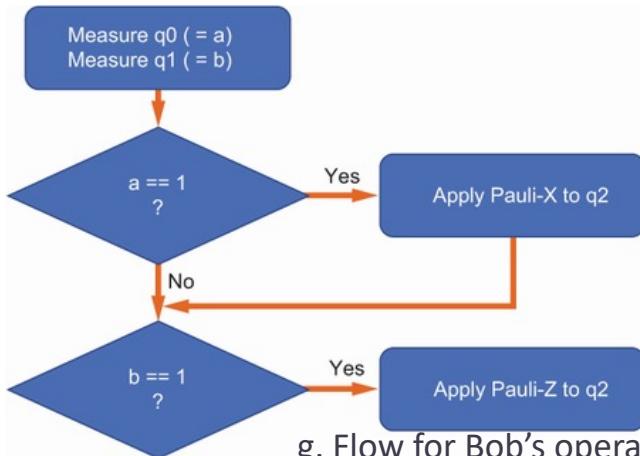
a. Alice wants to send her qubit  $q$  to Bob.



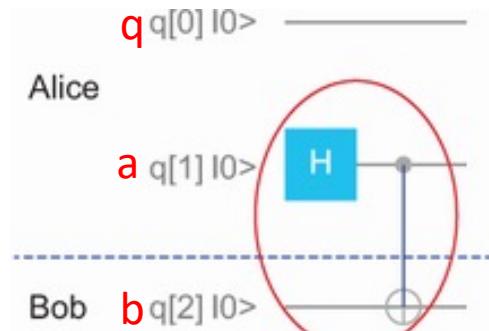
b. Alice and Bob share an entangled pair of qubits.



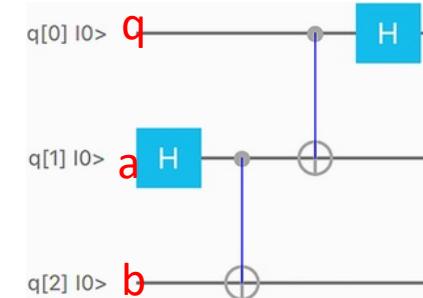
c. Alice lets her qubit interact with her part of the entangled pair.



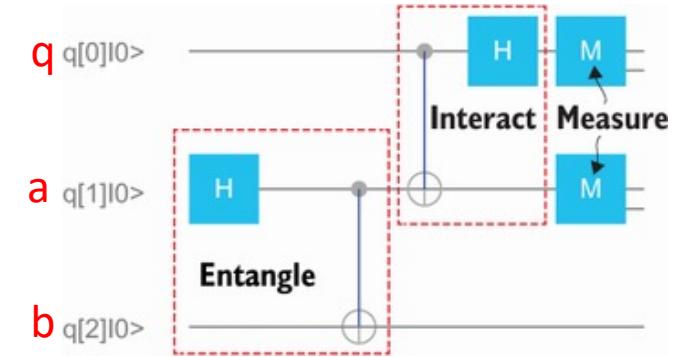
g. Flow for Bob's operations, depending on the values of Alice's qubits



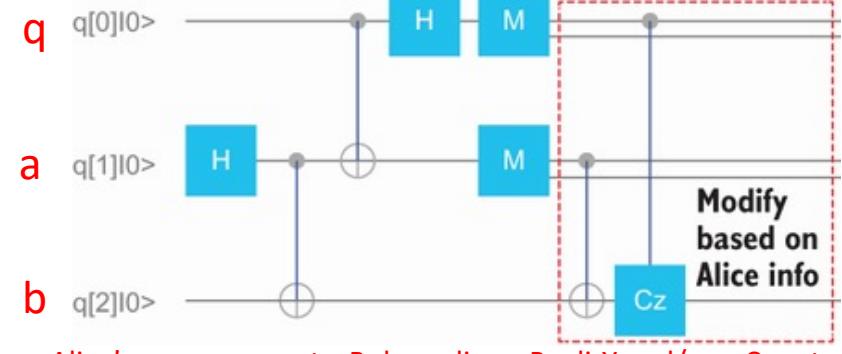
c. Circuit showing Alice with qubit  $q$  and Alice and Bob sharing an entangled pair (circled) –  $a$  and  $b$



e. Adding the interaction between Alice's qubit and her half of the entangled pair



f. Alice measures her qubit  $q$  and qubit  $a$ .



h. Depending on Alice's measurements, Bob applies a Pauli-X and/or a Cz gate.

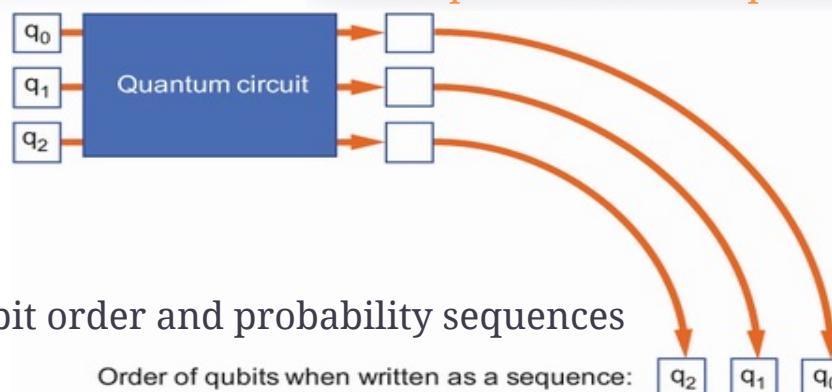
# Quantum Networking - Quantum Teleportation



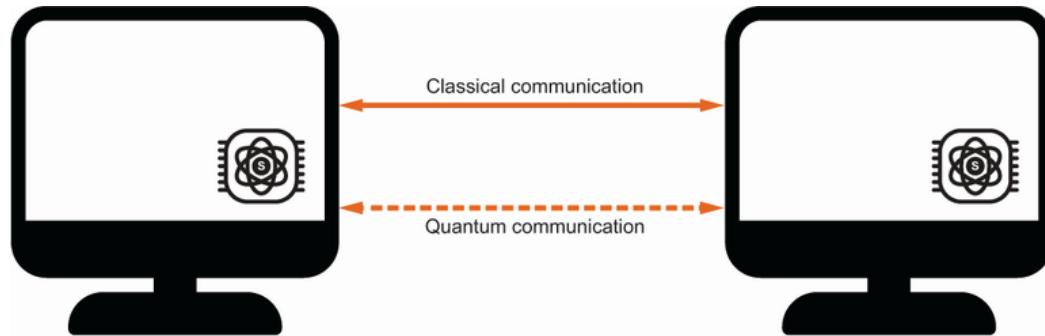
Output from the teleport program



Output from the teleport program with an initialized qubit



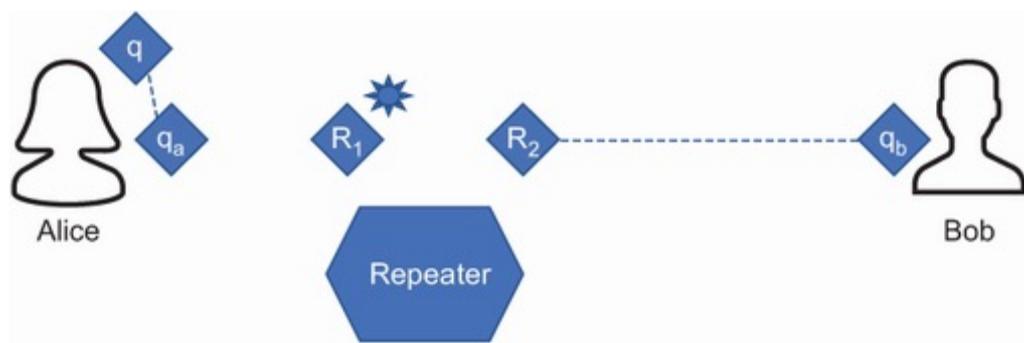
# Quantum Networking - Quantum & Classical Communication / Quantum Repeater



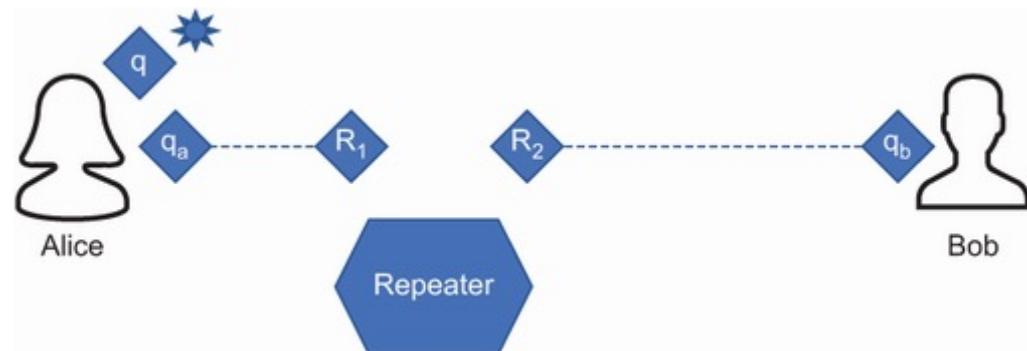
a. Communication between Alice and Bob is split between a classical channel and a quantum channel.



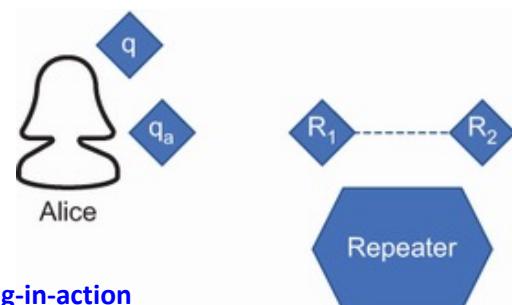
b. A quantum repeater separating the distance between Alice and Bob into segments, transferring the qubit from one segment to the other.



d. Alice interacts with the first entangled pair, teleporting her qubit to  $R_1$ .



c. A quantum repeater between Alice and Bob



e. The repeater let the  $R_1$  qubit interact with the second entangled part, thereby teleporting the information in the qubit to  $q_B$ .

# Quantum Networking - Quantum & Classical Communication / Quantum Repeater

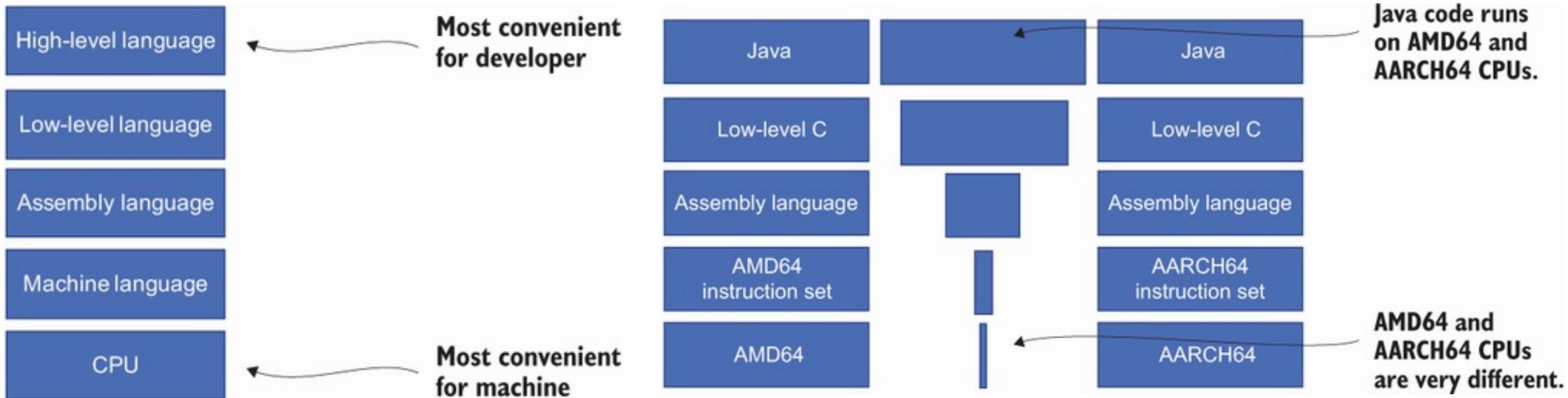


# Quantum Algorithms and Coding

An exhaustive list of quantum simulators, sorted by programming language, can be found @ (the links could be modified)

<https://www.quantiki.org/wiki/list-qc-simulators>

- The IBM Qiskit project is available at <https://qiskit.org>
- Microsoft has information about its Q# programming language at <https://docs.microsoft.com/en-us/quantum>
- Cirq, a quantum simulator in Python created by Google, can be found at <https://quantumai.google>



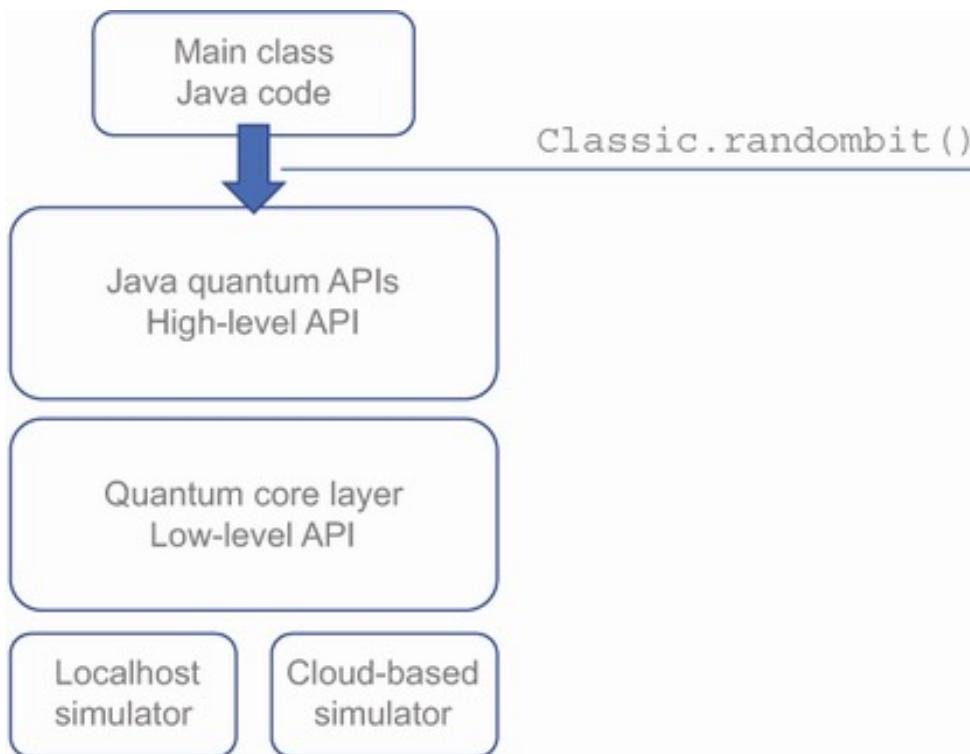
Typical software stack on top of a classical CPU. The hardware is at the bottom. The top of the stack is a high-level language used by software developers. The layers in between allow the next layer to be built on top of the previous layer.

Comparison of code reuse when targeting two different CPUs. The wider the bar in the middle column, the more code reuse. Code for high-level languages typically runs on different kinds of hardware. Low-level code requires parts that are relevant only to a specific architecture (such as AMD64 or AARCH64).

# Quantum Algorithms and Coding

It is recommended to use the **High-Level APIs** when:

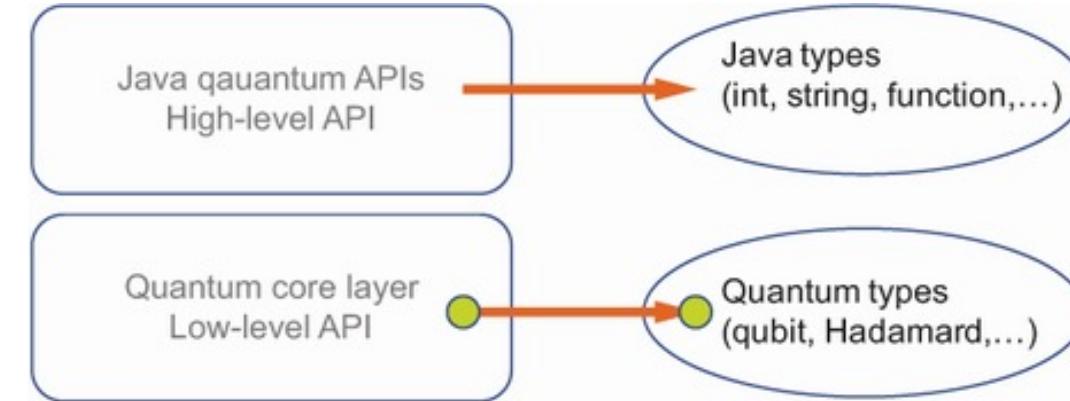
- You need to work on a project in which an existing, well-known, already-implemented quantum algorithm can provide an advantage—typically a performance advantage.
- You want to experiment with classical code that could benefit from quantum algorithms.



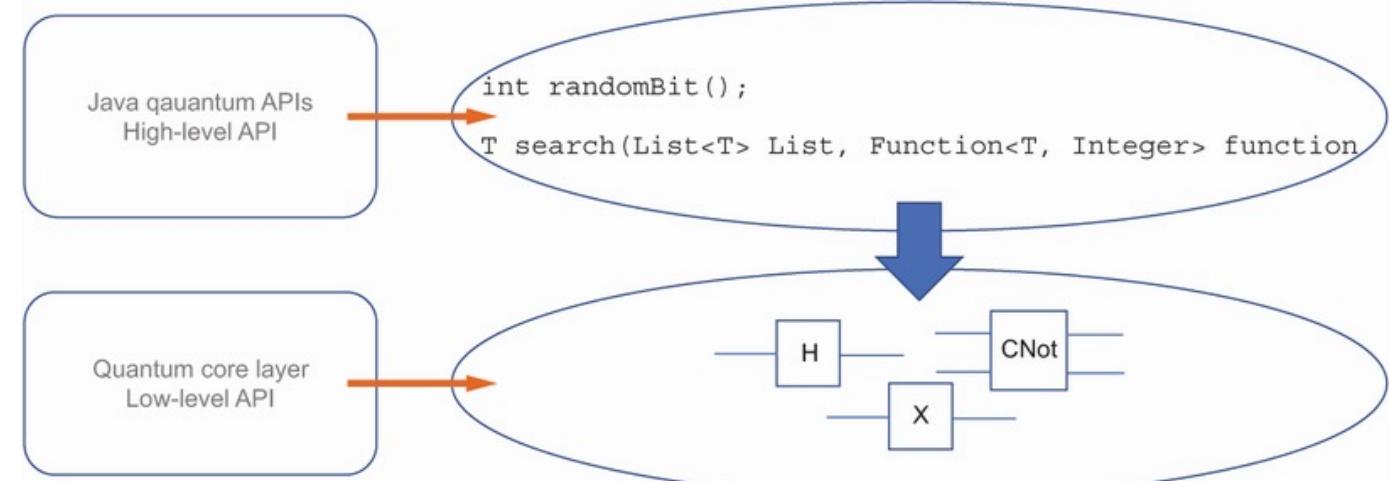
HelloWorld example and Strange high- and low-level APIs

It is recommended to use the **Low-Level APIs** when:

- You want to learn about quantum computing.
- You want to experiment with existing quantum algorithms.
- You want to develop new quantum algorithms.



High-level and low-level APIs and the types they use



High-level and low-level APIs and their implementations

Github: <https://github.com/critoma/quantum/tree/main/quantumjava/ch07>

# Quantum Algorithms and Coding

## Visualization of an Quantum Circuit in Java Strange Simulator

```
Program program = new Program(dim);
Step step0 = new Step(new Hadamard(0), new X(3));
Step step1 = new Step(new Cnot(0,1));

program.addSteps(step0, step1);
QuantumExecutionEnvironment qee = new SimpleQuantumExecutionEnvironment();
Result result = qee.runProgram(program);

Qubit[] qubits = result.getQubits();

for (int i = 0; i < dim; i++) {
    System.err.println("Qubit["+i+"]: "+qubits[i].measure());
}

Renderer.renderProgram(program);
```

- The static method *renderProgram* from Renderer class generates a window with a graphical overview of the circuit.
- The visualization shows that the quantum program starts with four qubits. In a first step, a Hadamard and a NOT gate are added to the circuit. On the right side of the figure, the resulting qubits are shown, with the probability that they will be measured as 1.

The output of the measurements can be:

Qubit[0]: 0

Qubit[1]: 0

Qubit[2]: 0

Qubit[3]: 1

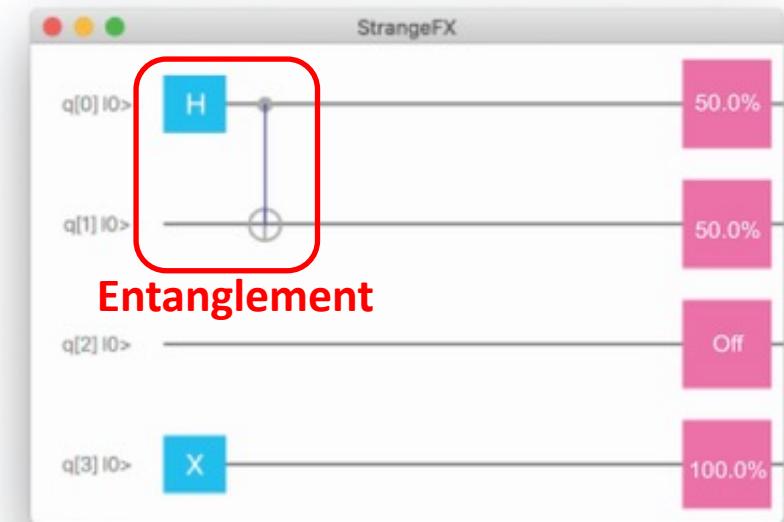
OR:

Qubit[0]: 1

Qubit[1]: 1

Qubit[2]: 0

Qubit[3]: 1



Run Program Ex 3-2: Entangled q... OpenQASM

```
1 // Programming Quantum Computers
2 // by Eric Johnston, Nic Harrigan and Mercedes Gimeno-Segovia, O'Reilly Media
3 // More samples like this can be found at http://oreilly-qc.github.io
4
5 // OpenQASM code was generated with the help of Qiskit (http://qiskit.org)
6 // A complete set of all Qiskit samples (including this one) can be found at
7 // https://github.com/oreilly-qc/oreilly-qc.github.io/tree/master/samples/Qiskit
8
9 // Run this sample in the IBM Q Experience Circuit Composer
10 // at https://quantum-computing.ibm.com
11 // Example 3-2: Entangled Qubits
12
13 OPENQASM 2.0;
14 include "qelib1.inc";
15
16 qreg a[1];
17 qreg b[1];
18 creg ac[1];
19 creg bc[1];
20
21 h a[0];
22 cx a[0],b[0];
23 measure a[0] -> ac[0];
24 measure b[0] -> bc[0];
25
```

<https://oreilly-qc.github.io/?p=3-2#>

Run Program Ex 3-2: Entangled q... Q#

Run the Q# Chapter 3 notebook online [here](#), or download [here](#) to run locally.

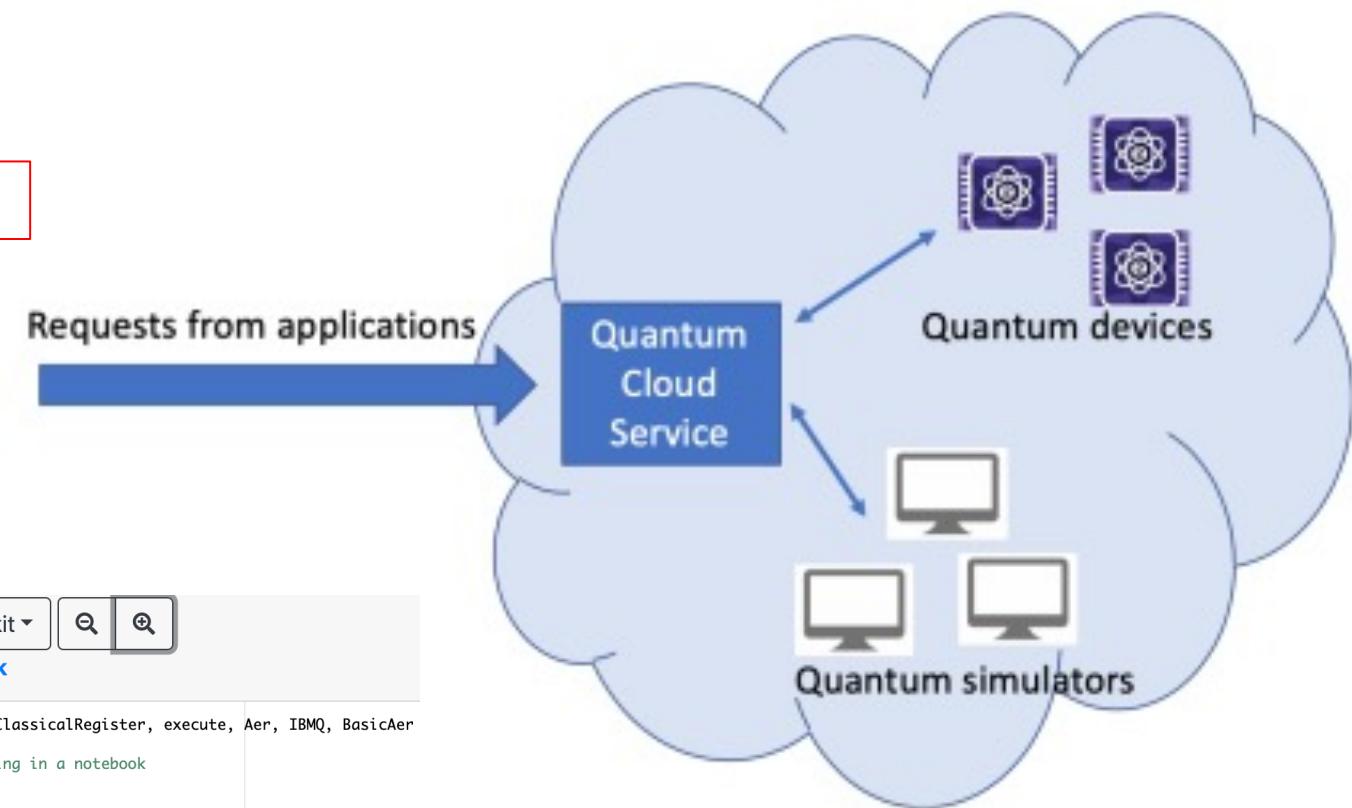
```
1 namespace QSharp.Chapter3
2 {
3     open Microsoft.Quantum.Canon;
4     open Microsoft.Quantum.Intrinsic;
5     // Example 3-2: Make a Bell pair
6     operation PrepareAndMeasureBellPair () : Unit {
7         // allocate the qubits
8         use (a, b) = (Qubit(), Qubit());
9         // put qubit a in superposition
10        H(a);
11
12        // entangle qubits a and b
13        CNOT(a, b);
14
15        // measure both qubits and output the results
16        Message($"Measurement results: {M(a)}, {M(b)}");
17        // make sure the qubits are back to the 0 state
18        ResetAll([a, b]);
19    }
20
21    operation PrepareMultipleBellPairs () : Unit {
22        // repeat the experiment multiple times to observe that
23        // the two bits will be random but always the same
24        for i in 1..10 {
25            PrepareAndMeasureBellPair();
26        }
27    }
28}
```

Run Program Ex 3-2: Entangled q... Qiskit

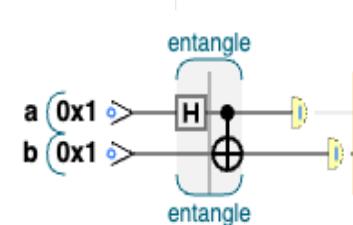
Download the complete Chapter3 [Qiskit Notebook](#)

```
5
6 from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, execute, Aer, IBMQ, BasicAer
7 import math
8 ## Uncomment the next line to see diagrams when running in a notebook
9 #%matplotlib inline
10
11 ## Example 3-2: Entangled Qubits
12 # Set up the program
13 a = QuantumRegister(1, name='a')
14 b = QuantumRegister(1, name='b')
15 a_c = ClassicalRegister(1, name='ac')
16 b_c = ClassicalRegister(1, name='bc')
17 qc = QuantumCircuit(a, b, a_c, b_c)
18
19 qc.h(a)           # put a into a superposition of 0 and 1
20 qc.cx(a, b)       # entangle a and b
21 qc.measure(a, a_c)
22 qc.measure(b, b_c)
23
24 backend = BasicAer.get_backend('statevector_simulator')
25 job = execute(qc, backend)
26 result = job.result()
27
28 counts = result.get_counts(qc)
29 print('counts:', counts)
30
31 outputstate = result.get_statevector(qc, decimals=3)
32 print(outputstate)
33 qc.draw()          # draw the circuit
```

## Cloud services with real quantum hardware or simulators



## Java Development stack versus runtime stack



### Development



### Runtime



### Strange API



# Quantum Algorithms and Coding – Java Strange Debugging

One of the most important restrictions of quantum circuits is that measuring a qubit influences its state. If a qubit is in a superposition state and it is measured, it will fall back to either 0 or 1. It can't go back to the state it was in before it was measured.

Although this provides great opportunities for security (as we explain in the next chapter), it makes debugging quantum circuits difficult. In a typical classical application, you often want to follow the value of a specific variable during the program flow. Debuggers are popular with developers, and examining the change in a variable often provides valuable insight into why a specific application is not behaving the way we expect it to behave.

However, if measuring a variable changes the behavior of the application—as is the case in quantum computing—this technique cannot be used. To make it more complex, even if we were able to restore the original state of a qubit after measuring it, the measurement itself, being 0 or 1, doesn't give all the information. As we explained several times before, the real value in quantum programs is not the measured value of a qubit but mainly the probability distributions.

Fortunately, **Strange** and **StrangeFX** allow for a way to render the probability distributions. Strange allows us to use a fictive gate, **ProbabilitiesGate**, which can visualize the probability vector at a given moment in the program flow.

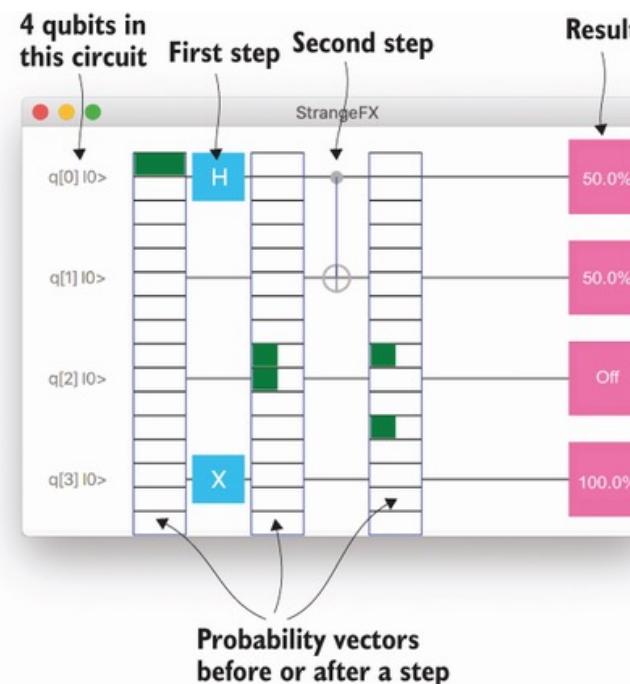
We'll reuse the program from the previous section, but this time we use **ProbabilitiesGate** to render the probabilities after a given step. The first part of the code is changed as follows:

```
Program program = new Program(dim);
Step step0 = new Step(new Hadamard(0), new X(3));
Step step1 = new Step(new Cnot(0,1));

program.addSteps(step0, step1);
```

```
Program program = new Program(dim);
Step p0 = new Step (new ProbabilitiesGate(0));
Step step0 = new Step(new Hadamard(0), new X(3));
Step p1 = new Step (new ProbabilitiesGate(0));
Step step1 = new Step(new Cnot(0,1));
Step p2 = new Step (new ProbabilitiesGate(0));

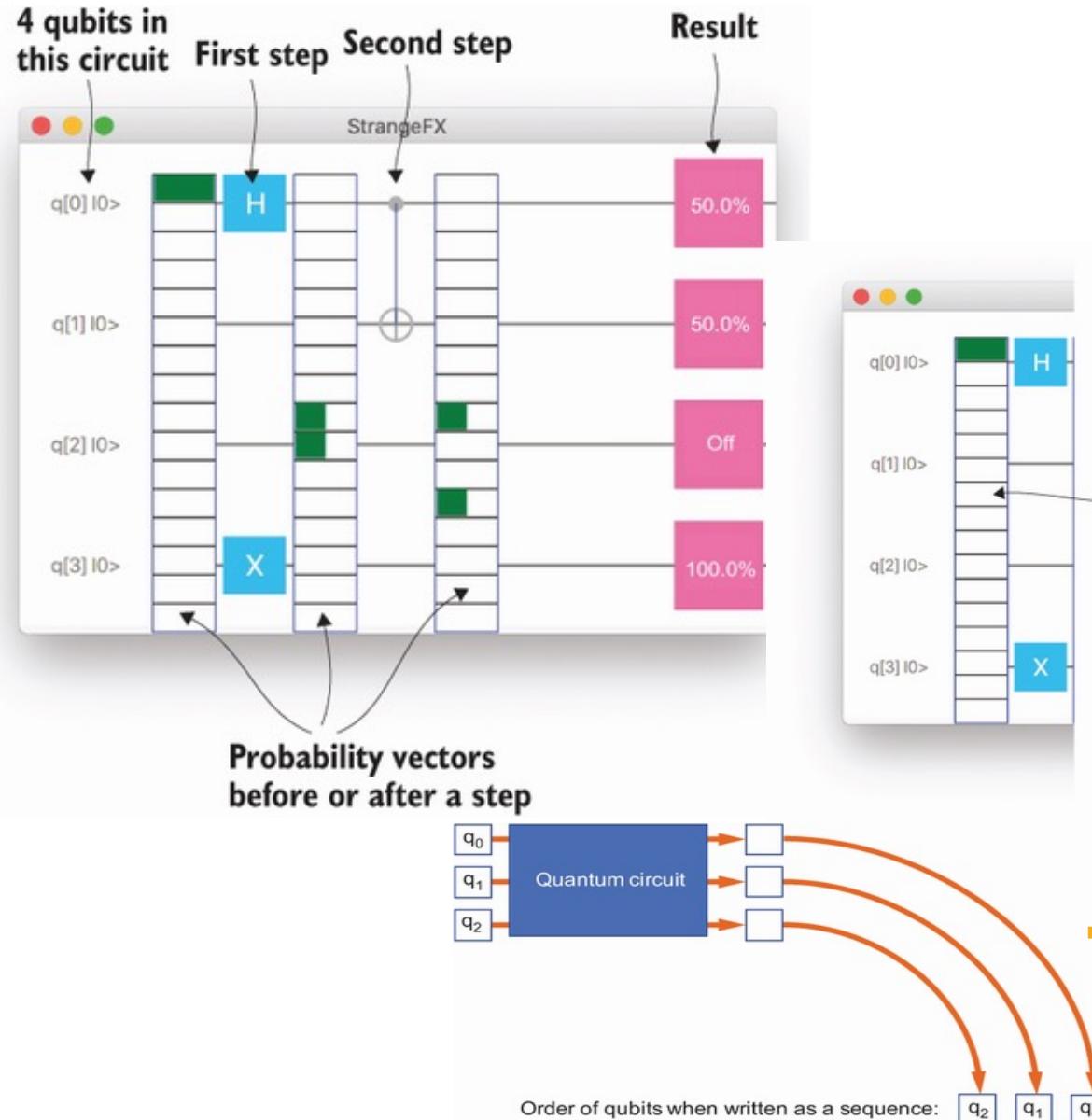
program.addSteps(p0, step0, p1, step1, p2);
```



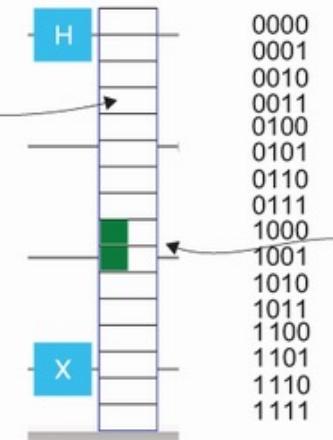
Visualization of the circuit with probabilities. Before and after each step, the probability vector is shown. This gives an indication of possible outcomes after each step without measuring the qubits.

# Quantum Algorithms and Coding – Java Strange Debugging

<https://github.com/critoma/quantum/blob/main/quantumjava/ch07/randombitdebug/src/main/java/org/redfx/javaqc/ch07/randombitdebug/Main.java>



Probability vector after applying first step

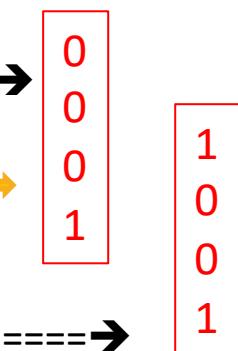


The two bars that are partially colored correspond to nonzero probabilities.

Probability vector before processing

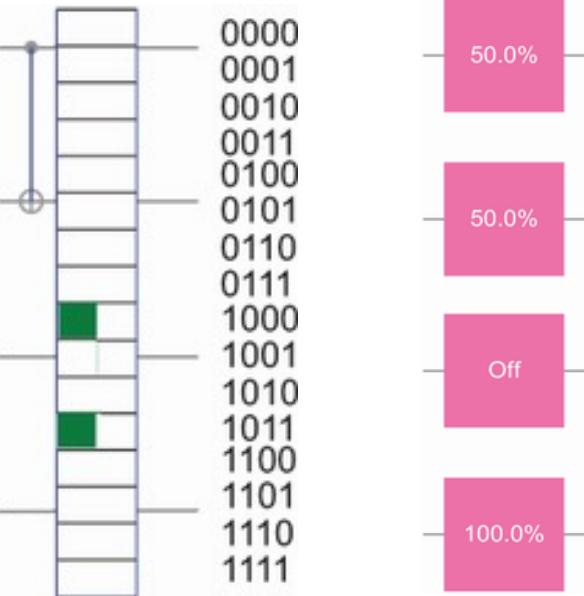
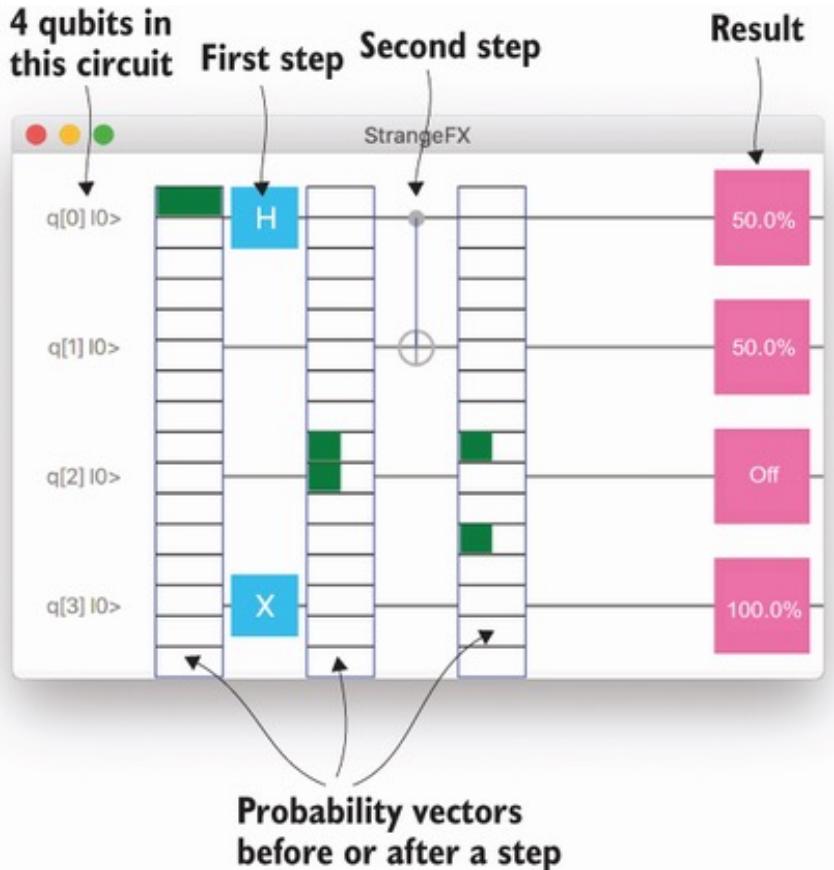
Probability vector after applying a Hadamard and a NOT gate. The **1000** (for top-down representation, read from right to left) and **1001** states are the only possible ones after this step, with an equal probability. Rendering this vector does not measure the qubits, so the processing can continue.

- A 50% chance that we would measure **1000**
- A 50% chance that we would measure **1001**



# Quantum Algorithms and Coding – Java Strange Debugging

<https://github.com/critoma/quantum/blob/main/quantumjava/ch07/randombitdebug/src/main/java/org/redfx/javaqc/ch07/randombitdebug/Main.java>



## Possible measurements

This corresponds with what you learned when creating a Bell state. Applying a Hadamard gate followed by a CNOT gate brings the two involved qubits ( $q_0$  and  $q_1$ ) into an entangled state. Both qubits can be 0, and both qubits can be 1. If both qubits are 0, the total state of the quantum circuit is measured as 1000. If both qubits are 1, the total state is measured as 1011.

## Probability vector after applying a CNOT gate

This figure indicates two possible outcomes to measure at this point:

- A 50% chance to measure 1000  $\longleftrightarrow$

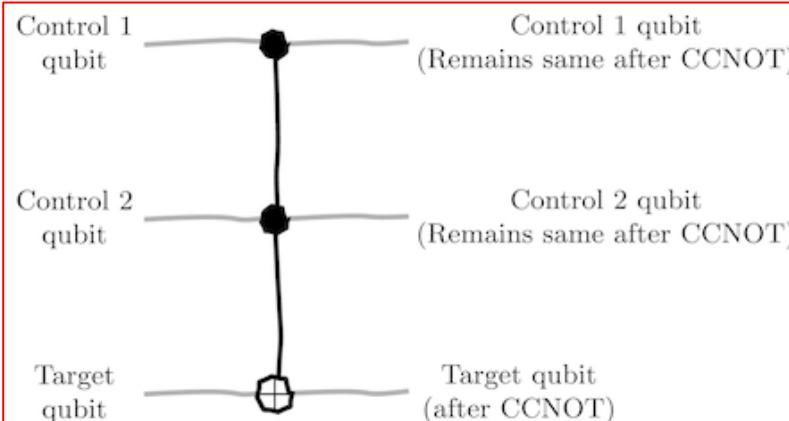
0  
0  
0  
1

- A 50% chance to measure 1011  $\longleftrightarrow$

1  
1  
0  
1

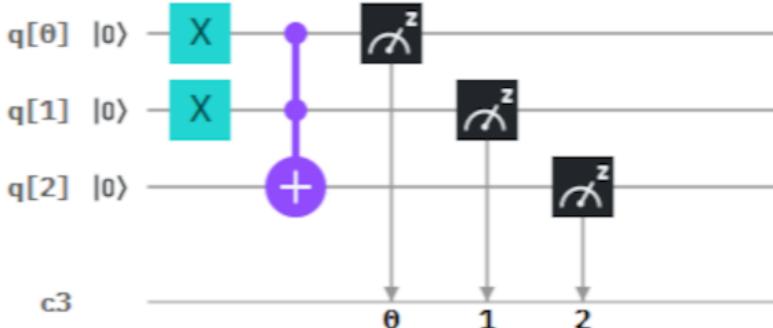
We know that  $q_2$  will always be measured as 0, and  $q_3$  will always be measured as 1. The other two qubits,  $q_0$  and  $q_1$ , can be either 0 or 1. At first glance, this might correspond to what we learned by looking at the probability vector, but some important information is missing when we look at the possible measurements for the qubits only. Indeed, the probability vector states that the first two qubits can be either 00 or 11 but never 01 or 10 because they are entangled. There are only two possible combinations, not four. This is something that cannot be seen simply by looking at the potential outcomes for qubit measurements.

# Quantum Algorithms and Coding – Multiple qubits operations (>= 3) – CCNOT Gate



$$\begin{pmatrix} |a\rangle \\ |b\rangle \\ |c\rangle \end{pmatrix} \mapsto \begin{pmatrix} |a\rangle \\ |b\rangle \\ |c\rangle \oplus (|a\rangle \wedge |b\rangle) \end{pmatrix}$$

$ 110\rangle$	$\rightarrow$	$ 111\rangle$
$ 111\rangle$	$\rightarrow$	$ 110\rangle$



In [logic circuits](#), the **Toffoli gate** (also **CCNOT gate**), invented by [Tommaso Toffoli](#), is a universal [reversible logic gate](#), which means that any classical reversible circuit can be constructed from Toffoli gates. It is also known as the "controlled-controlled-not" gate, which describes its action. It has 3-bit inputs and outputs; if the first two bits are both set to 1, it inverts the third bit, otherwise all bits stay the same.

[https://en.wikipedia.org/wiki/Toffoli\\_gate](https://en.wikipedia.org/wiki/Toffoli_gate)

Truth table

INPUT			OUTPUT		
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Permutation matrix form

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

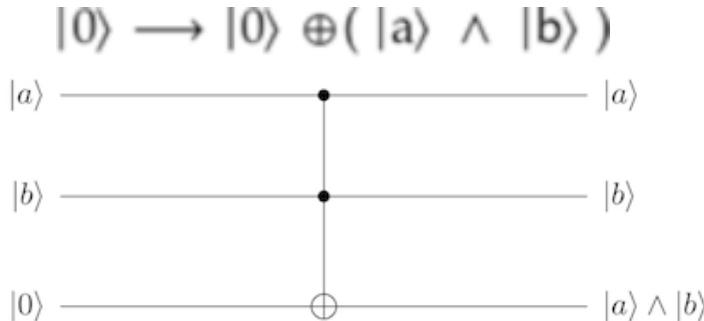
$$|000\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |001\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |010\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |011\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$|100\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |101\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \quad |110\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |111\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

[https://learning.oreilly.com/library/view/quantum-computing/9781680508086/f\\_0024.xhtml#sec.CCNOT.Gate](https://learning.oreilly.com/library/view/quantum-computing/9781680508086/f_0024.xhtml#sec.CCNOT.Gate)

# Quantum Algorithms and Coding – Multiple qubits operations (>= 3) – AND/OR Gate

To build the quantum equivalent of an **AND** gate, we can use the **CCNOT** gate



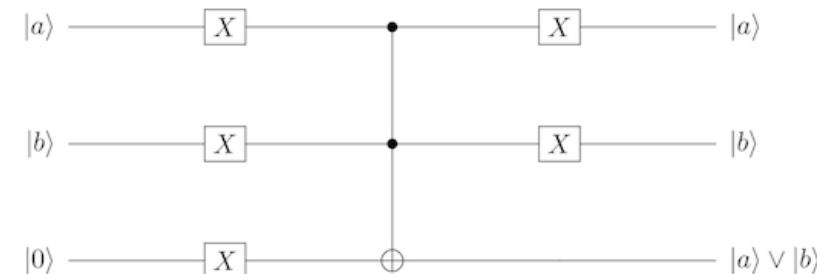
```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[3];
creg c[3];

// |110> --CCNOT aka AND--> |111>
x q[0];
x q[1];
barrier q;
ccx q[0],q[1],q[2];
barrier q;
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

<https://quantum-computing.ibm.com/composer/files/fcb852eb582d9b6ba9c723f10fb26e90d67ce05f35637f1ebbe2f6639729f0c1>

To build the quantum equivalent of an **OR** gate, we can use the **CCNOT** gate



```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[3];
creg c[3];

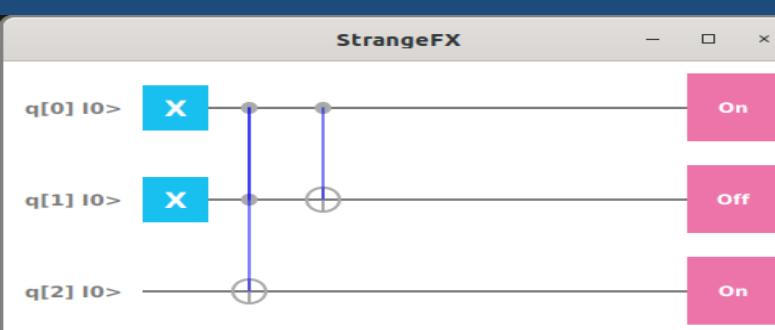
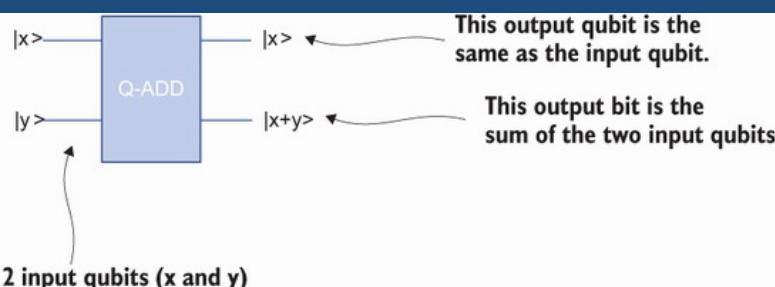
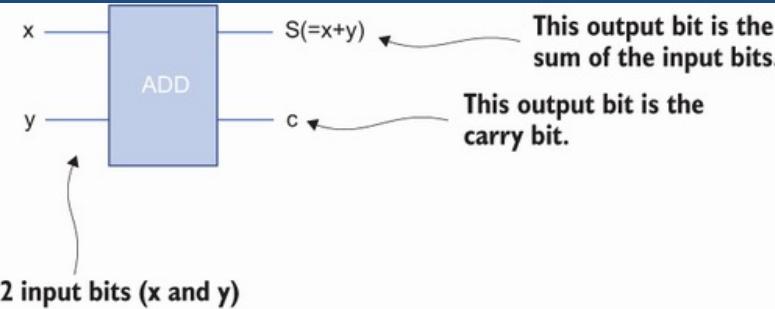
// |010> --OR Gate w CCNOT-> |011>
x q[1];
barrier q;
x q[0];
x q[1];
x q[2];
ccx q[0],q[1],q[2];
x q[0];
x q[1];
barrier q;
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

<https://quantum-computing.ibm.com/composer/files/543bd25f6eb537018283cae157845c7b7ce6cb746339e6b94ee3f40cf4c7c12>

# Qubits circuits –2 Qubits Java Strange Adder Method

[https://learning.oreilly.com/library/view/quantum-computing-in/9781617296321/OEBPS/Text/07.htm#heading\\_id\\_20](https://learning.oreilly.com/library/view/quantum-computing-in/9781617296321/OEBPS/Text/07.htm#heading_id_20)

<https://github.com/critoma/quantum/blob/main/quantumjava/ch07/add2/src/main/java/org/redfx/javaqc/ch07/add2/Main.java>



```
static int add(int a, int b) {  
    Program program = new Program(3);  
    Step prep = new Step();  
    if (a > 0) {  
        prep.addGate(new X(0));  
    }  
    if (b > 0) {  
        prep.addGate(new X(1));  
    }  
  
    Step step0 = new Step(new Toffoli(0,1,2));  
    Step step1 = new Step(new Cnot(0,1));  
  
    program.addSteps(prep, step0, step1);  
    QuantumExecutionEnvironment qee =  
        new SimpleQuantumExecutionEnvironment();  
    Result result = qee.runProgram(program);  
    Qubit[] qubits = result.getQubits();  
    return qubits[1].measure()  
        + (qubits[2].measure() <<1);  
}
```

- ❶ We now need three qubits, as explained in the text.
- ❷ Prepares the two qubits that we want to add. The third qubit is initially in the 0 state.
- ❸ Applies a Toffoli gate with the qubits at index 0 and index 1 as control qubits. The qubit at index 2 is the target qubit.
- ❹ Similar to the code in the first step, applies a CNOT gate sum to the first two qubits and stores the value in the second qubit, leaving the first one unchanged.
- ❺ Executes the program and obtains the results
- ❻ The value in the qubit with index 1 is the sum modulus 2, and the value in the qubit with index 2 has the carry bit, which is 1 if there was an overflow. In that case, the total sum is  $2 + \text{the sum in qubit with index 1}$ .

# Qubits circuits – 2 Bits vs 2 Qubits Full Adder

<https://quantum-computing.ibm.com/composer/files/1268acbc213d41da0f2a4bb5cf2b9cdda88bf08ee209993e4c1e39ba39d8347b>

IBM Quantum Learning      Home Catalog Composer Lab

F06\_Add2Qubits      Saved      File Edit View

Operations

Left alignment      Inspect

q[0]      q[1]      q[2]      q[3]      q[4]

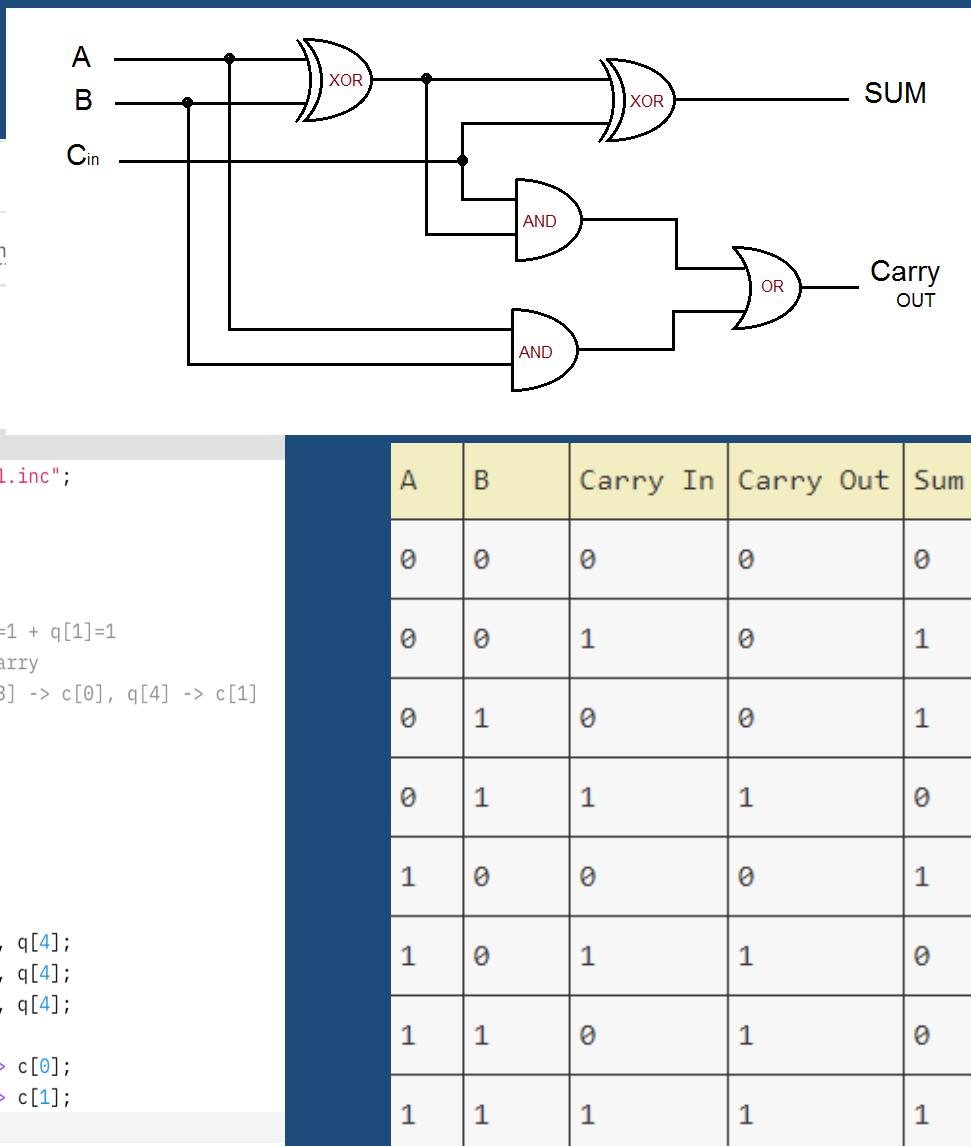
c2

Probabilities

Probability (% of 1024 shots)

Computational basis states

Q-sphere



<https://quantumcomputinguk.org/tutorials/performing-addition-on-ibms-quantum-computers>

# Qubits circuits – Adding 2 integers (multiple qubits involved)

[https://github.com/oreilly-qc/oreilly-qc.github.io/blob/master/samples/OpenQASM/ch05\\_02\\_quantum\\_add.qasm](https://github.com/oreilly-qc/oreilly-qc.github.io/blob/master/samples/OpenQASM/ch05_02_quantum_add.qasm) | <https://oreilly-qc.github.io/?p=5-2#>

← → C oreilly-qc.github.io/?p=5-2#

Gmail YouTube Maps News Translate

Docs O'Reilly Buy Book Engines ▾ Errata Contact

O'REILLY Programming Quantum Computers Essential Algorithms and Code Samples Eric R. Johnston, Nic Harrigan & Mercedes Gimeno-Segovia

## Programming Quantum Computers

### Code Samples

Run Program Ex 5-2: Adding two ... QC Engine ▾

```
1 // Programming Quantum Computers
2 // by Eric Johnston, Nic Harrigan and Mercedes Gimeno-Segovia O'Reilly Media
3 // To run this online, go to http://oreilly-qc.github.io?p=5-2
4
5 // Initialize
6 var num_qubits = 6;
7 qc.reset(num_qubits);
8 var a = qint.new(4, 'a');
9 var b = qint.new(2, 'b');
10
11 // prepare
12 qc.label('prepare');
13 a.write(1);
14 a.hadamard(0x4);
15 a.phase(45, 0x4);
16 b.write(1);
17 b.hadamard(0x2);
18 b.phase(90, 0x2);
19 qc.nop();
20 qc.label('');
21 qc.nop();
22
23 // a += b
24 qc.label('a += b');
25 a.add(b);
```

Source code on Github

QC Engine Qiskit OpenQASM Q# Circ

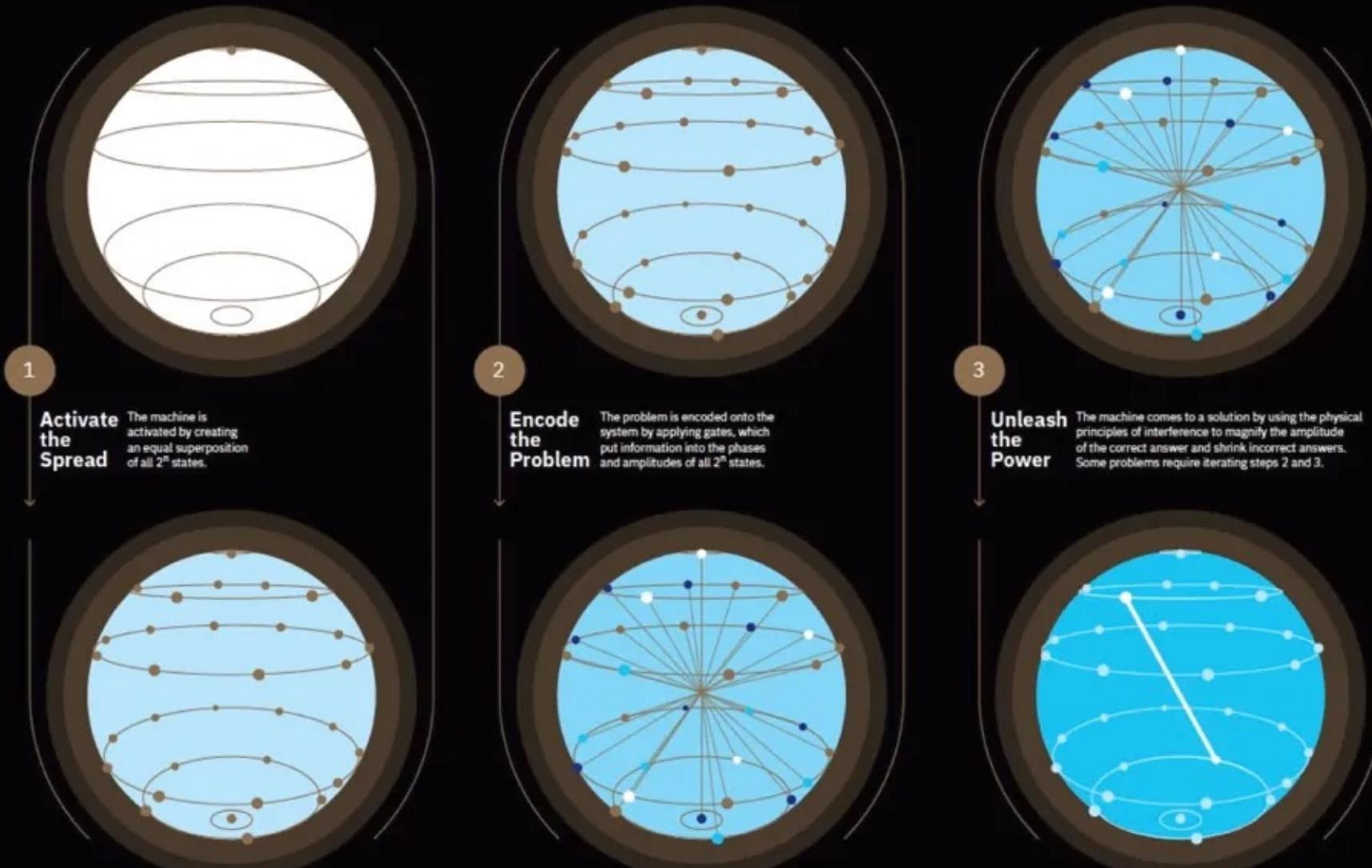
#### Program circuit

The circuit consists of two main sections: 'prepare' and 'a += b'. In the 'prepare' section, two 3-qubit registers, 'a' and 'b', are initialized. Register 'a' starts at state 1 and undergoes a Hadamard gate at address 0x4 and a phase gate at address 0x4 with a 45-degree angle. Register 'b' starts at state 1 and undergoes a Hadamard gate at address 0x2 and a phase gate at address 0x2 with a 90-degree angle. In the 'a += b' section, both registers are combined. The first three qubits of register 'a' are added to the first three qubits of register 'b' using CNOT gates. The last three qubits of register 'a' are also added to the last three qubits of register 'b' using CNOT gates.

#### Circle notation

A series of circles representing quantum states labeled |0>, |1>, |2>, |3>, |4>, |5>, |6>, and |7>. Each circle is a small circle with a dot inside, representing a single qubit state.

# Quantum algorithms

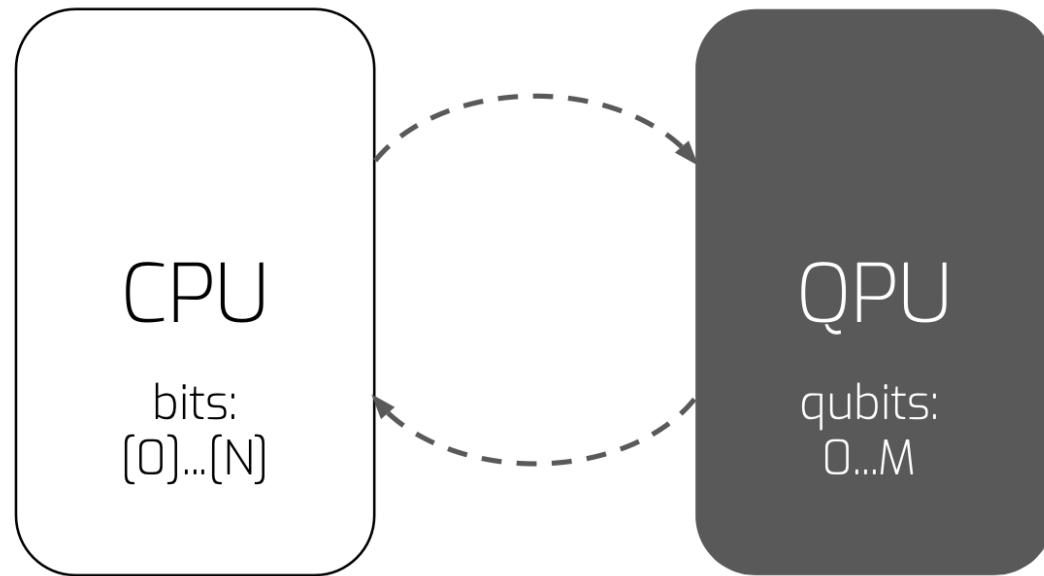


<https://towardsdatascience.com/the-need-promise-and-reality-of-quantum-computing-4264ce15c6c0>

# How do I program a quantum computer?

Hybrid Quantum Computers | Quantum Programming | Hybrid Programming | Hybrid Algorithms

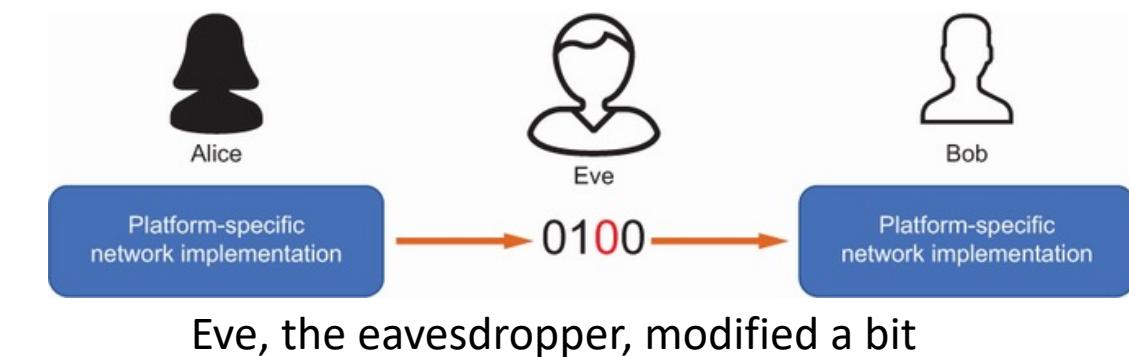
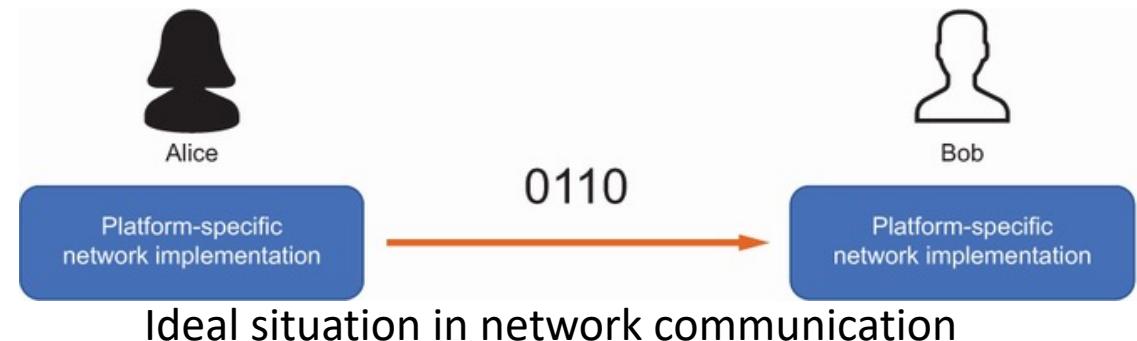
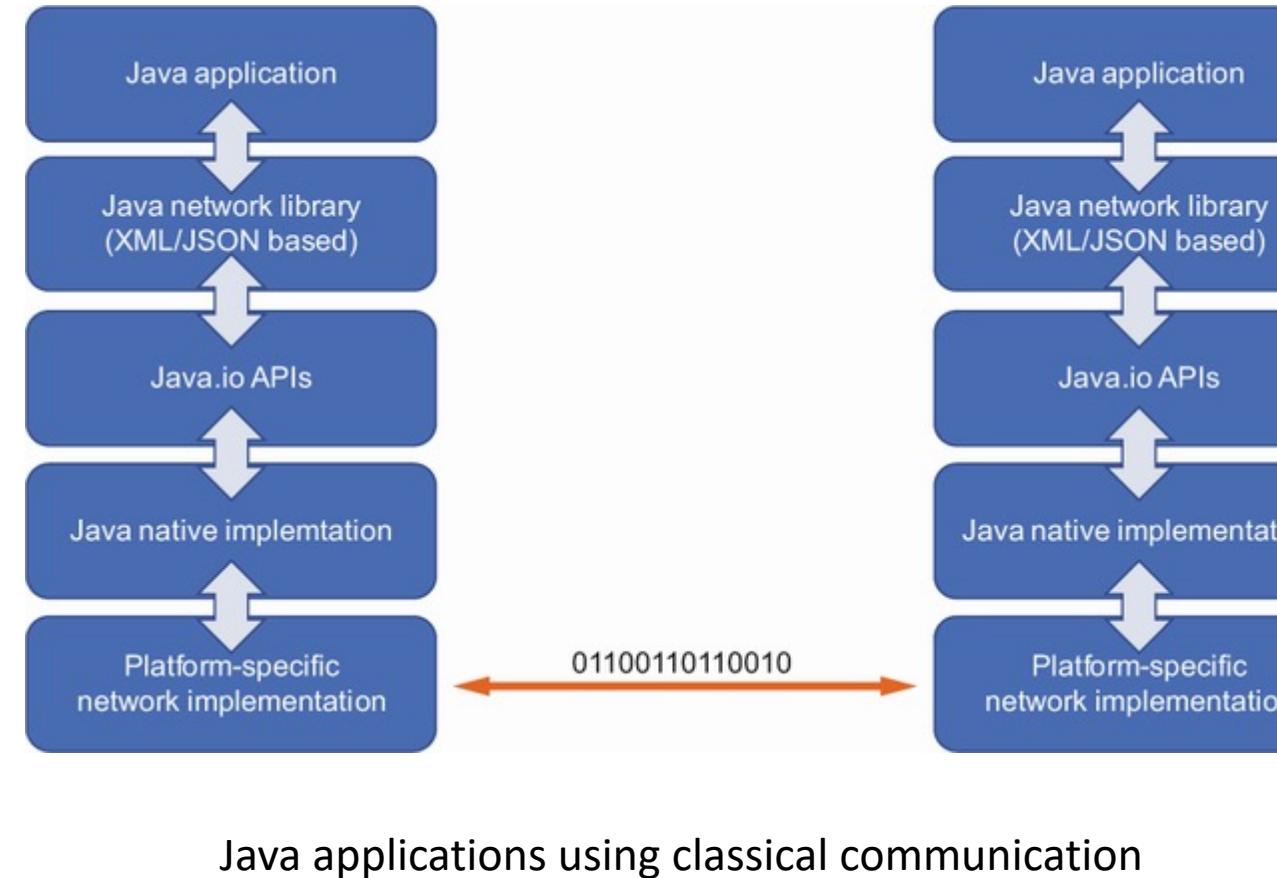
Practical, valuable quantum computing is **Hybrid** Quantum/Classical Computing



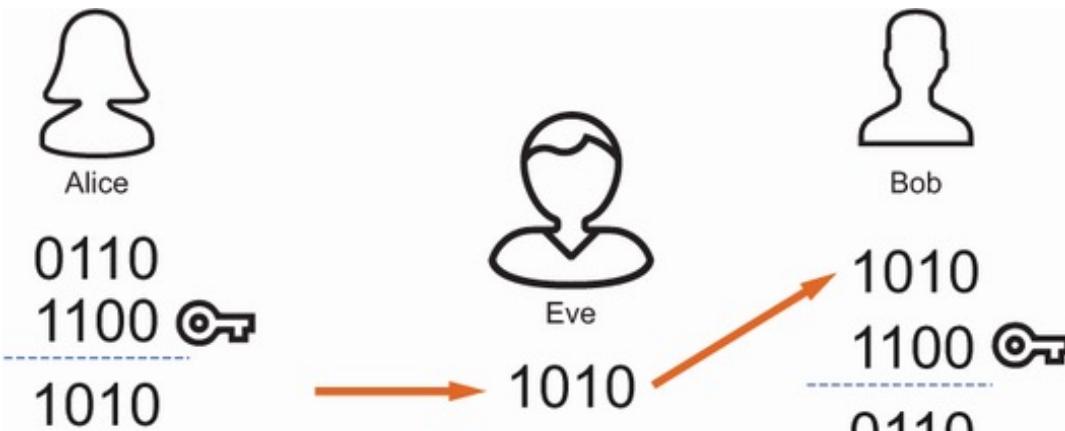
The Quil **[01]** instruction set is optimized for this.

Smith, Curtis, Zeng. "A Practical Quantum Instruction Set Architecture" arXiv:1608.03355

# Secure Communication using Quantum Computing



# Secure Communication using Quantum Computing – Bootstrap Problem



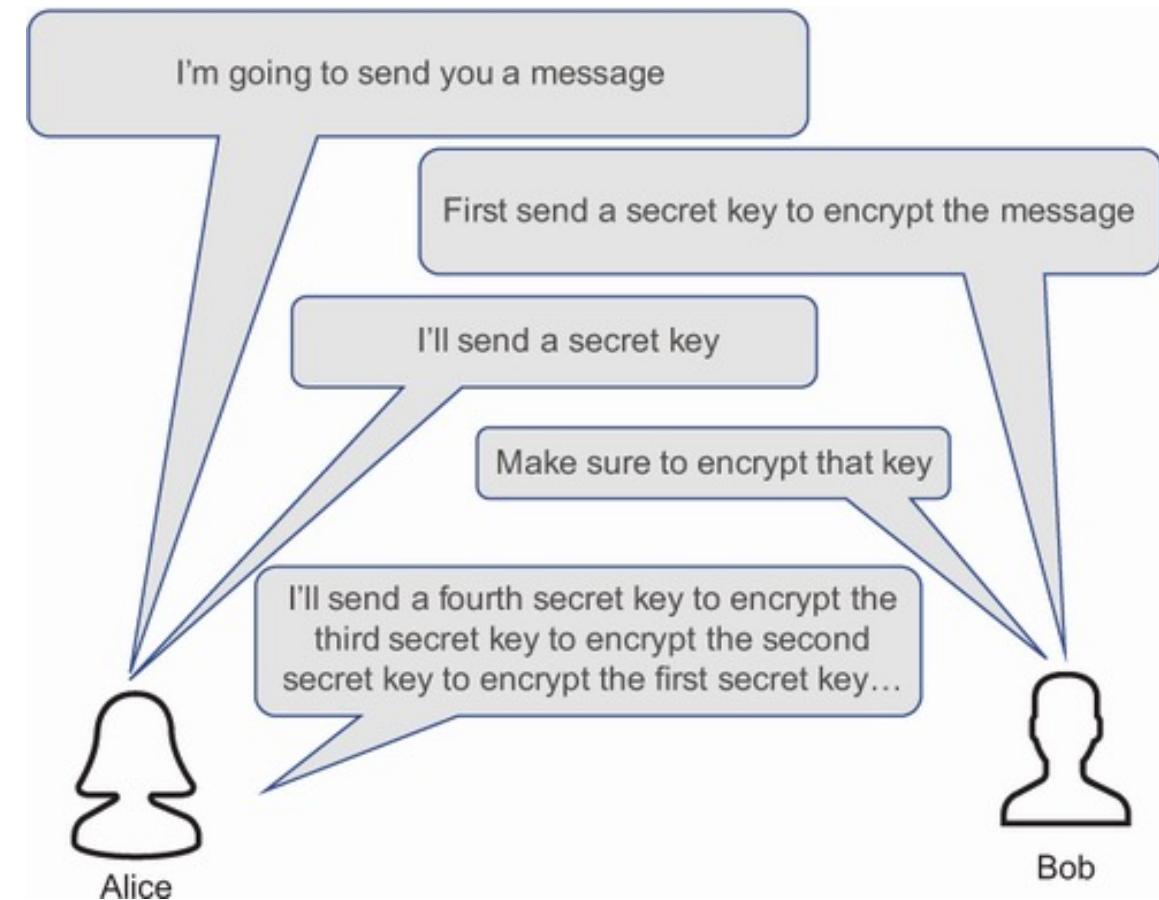
Alice and Bob communicating via a one-time pad

The result of this combination is an encrypted bit sequence:

```
0110 (original message)  
1100 (one time pad)  
---- (XOR operation)  
1010 (result)
```

Alice now sends the encrypted bit sequence 1010 to Bob, who needs to decode the sequence. To do this, Bob also applies an XOR operation on every bit he receives with the corresponding bit in the key:

```
1010 (encrypted message, received from Alice)  
1100 (one time pad)  
---- (XOR)  
0110 (original message)
```



Alice and Bob discover the bootstrap problem

# Secure Communication using Quantum Computing – Quantum Key Distribution (QKD) – Naive Approach

```
final int SIZE = 4;
Random random = new Random();

boolean[] aliceBits = new boolean[SIZE];
for (int i = 0 ; i < SIZE; i++) {
    aliceBits[i] = random.nextBoolean();
}

QuantumExecutionEnvironment simulator =
    new SimpleQuantumExecutionEnvironment();
Program program = new Program(SIZE); ❸
Step step1 = new Step();
Step step2 = new Step();
for (int i = 0; i < SIZE; i++) {
    if (bits[i]) step1.addGate(new X(i)); ❹
    step2.addGate(new Measurement(i)); ❺
}

program.addStep(step1);
program.addStep(step2);

Result result = simulator.runProgram(program);
Qubit[] qubit = result.getQubits(); ❻

int[] measurement = new int[SIZE];
boolean[] bobBits = new boolean[SIZE];

for (int i = 0; i < SIZE; i++) {
    measurement[i] = qubit[i].measure(); ❼
    bobBits[i] = measurement[i] == 1;

    System.err.println("Alice sent "+(bits[i] ? "1" : "0") +
        " and Bob received "+ bobBits[i] ? "1" : "0");
}
```

- ❶ This example creates a key with a fixed size: 4 bits.
- ❷ Alice generates the key by assigning random values to each bit.
- ❸ Creates a program that involves 1 qubit for every bit in the key
- ❹ When a bit is TRUE, a Pauli-X gate is applied to the corresponding qubit.
- ❺ All qubits will be measured in step 2.
- ❻ Executes the program; the results are in an array of qubits.
- ❼ Measures the qubits and prints their value next to the original value of the corresponding bit used by Alice
- ❽ Renders the quantum circuit of this application

Alice sent 0 and Bob received 0  
Alice sent 1 and Bob received 1  
Alice sent 0 and Bob received 0  
Alice sent 0 and Bob received 0



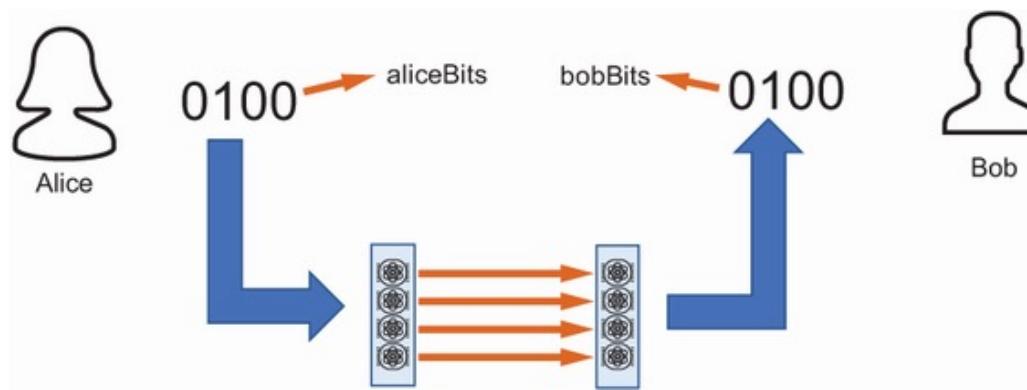
Quantum circuit showing the algorithm used by our application

Github:

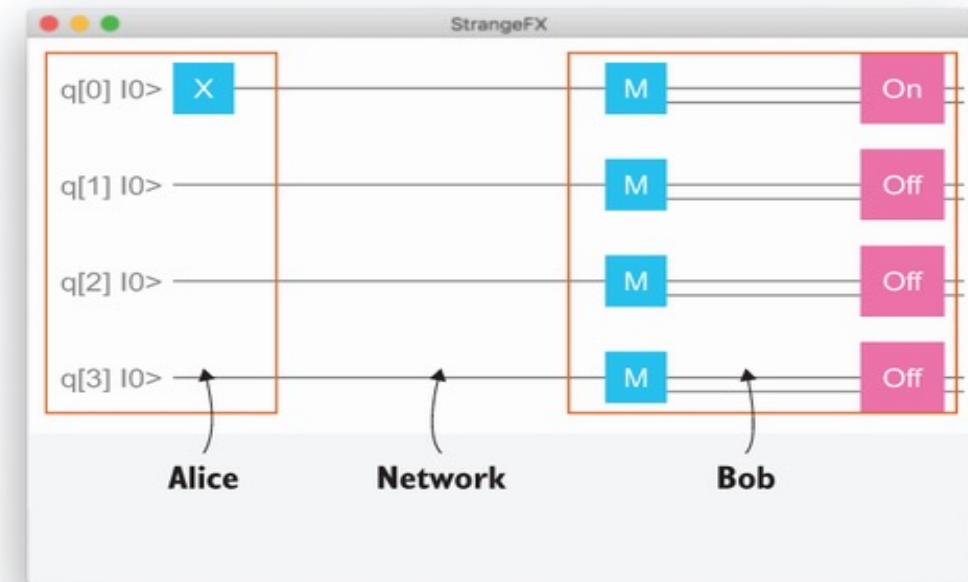
<https://github.com/critoma/quantum/blob/main/quantumjava/ch08/naive/src/main/java/org/redfx/javaqc/ch08/naive/Main.java>

# Secure Communication using Quantum Computing – Quantum Key Distribution (QKD) – Naive Approach

## Naive approach for generating and sending a quantum key



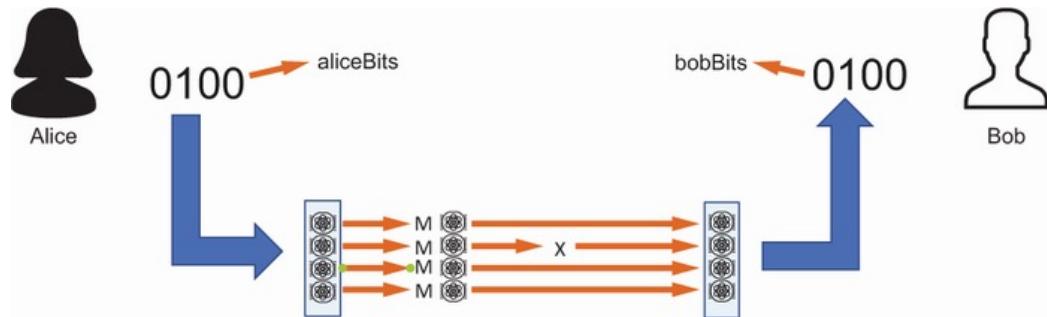
In the previous program/section, we explained that for now, we are running the examples on a single node. That means both the part of the algorithm executed by Alice and the part executed by Bob are executed on the same node. Remember that there is an implicit point in the algorithm where we assume that the qubits are sent from Alice to Bob.



# Secure Communication using Quantum Computing – Quantum Key Distribution (QKD) – Naive Approach

## Naive approach for generating and sending a quantum key

Eve reads the qubits and creates new qubits based on what she measures:



The output from the application shows that Alice can create a sequence of random bits and that Bob can receive the same sequence of random bits. You used qubits to transport the bits over a network cable.

If the quantum network is reliable and secure, this approach should work. You learned earlier that ***qubits can't be cloned and that once a qubit is measured, it falls back to one of its basic states. This behavior can be very helpful when dealing with quantum networks that should prevent eavesdropping.***

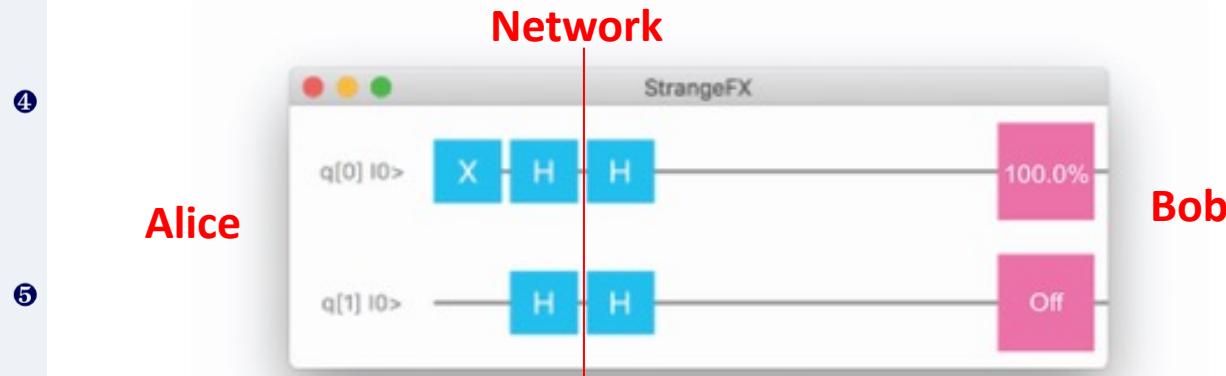
However, the current application is far from secure. Suppose Eve is still in the middle, and she is measuring all qubit communication between Alice and Bob. We know that when Eve measures a qubit, the qubit will hold the value 0 or the value 1. If it was in a superposition state, the information about that superposition is lost. But in the current algorithm, there are no qubits in a superposition state. Hence, Eve knows that when she is measuring 0, the original qubit was in the state  $|0\rangle$ . She can then create a new qubit in the initial  $|0\rangle$  state and put that back on the wire toward Bob. Similarly, when Eve measures a qubit and obtains the value 1, she knows that the qubit was in the  $|1\rangle$  state. She can create a new qubit in the  $|0\rangle$  state, apply a Pauli-X gate to bring it in the  $|1\rangle$  state, and send it to Bob.

You can see that the eavesdropping part happens in the network layer. When Eve has access to the network, she can obtain the (not so) secret key that Alice and Bob share. She measures the same values that Alice used to generate the qubits, and those values are the secret key. What is especially dangerous is that Alice and Bob are not aware of what is happening. Bob receives qubits, measures them, and constructs the secret key. Bob and Alice successfully exchange a message encrypted with their secret key—but if Eve intercepts this message, she can decrypt it.

# Secure Communication using Quantum Computing – Quantum Key Distribution (QKD) – Superposition with Haha Approach

```
QuantumExecutionEnvironment simulator =  
    new SimpleQuantumExecutionEnvironment();  
  
Program program = new Program(2);  
Step step0 = new Step();  
step0.addGate(new X(0));  
  
Step step1 = new Step();  
step1.addGate(new Hadamard(0));  
step1.addGate(new Hadamard(1));  
  
Step step2 = new Step();  
step2.addGate(new Hadamard(0));  
step2.addGate(new Hadamard(1));  
  
program.addStep(step0);  
program.addStep(step1);  
program.addStep(step2);  
  
Result result = simulator.runProgram(program);  
Qubit[] qubit = result.getQubits();  
  
Renderer.renderProgram(program);
```

- 1 Creates a program with two qubits
- 2 Flips the first qubit to be  $|1\rangle$  while we keep the second qubit at  $|0\rangle$
- ③ Applies a Hadamard gate to both qubits
- 4 Applies another Hadamard gate to both qubits
- ② 5 Adds all the steps to the program
- 6 Executes the program
- ③ 7 Measures the qubits
- 8 Renders the results graphically



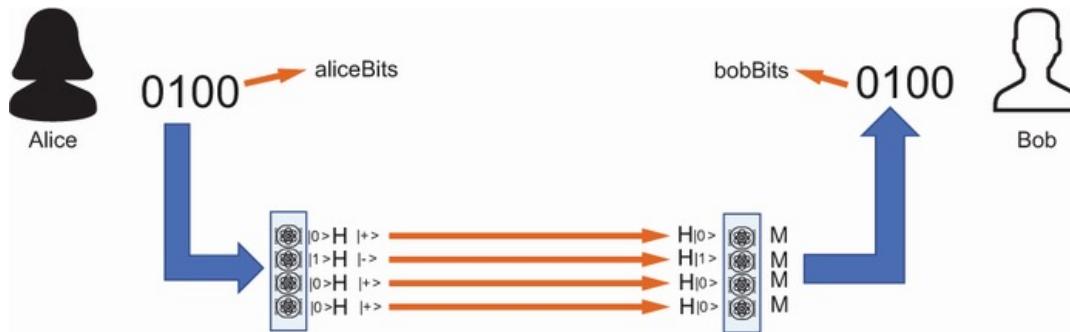
Result of applying two Hadamard gates in a row

- ④
- ⑤
- ⑥ We learn the following from this code: if Alice applies a Hadamard gate before she sends her qubit to Bob, and Bob applies another Hadamard gate before he measures the qubit, the qubit is back in the state that Alice prepared (either  $|0\rangle$  or  $|1\rangle$  ).
- ⑦
- ⑧

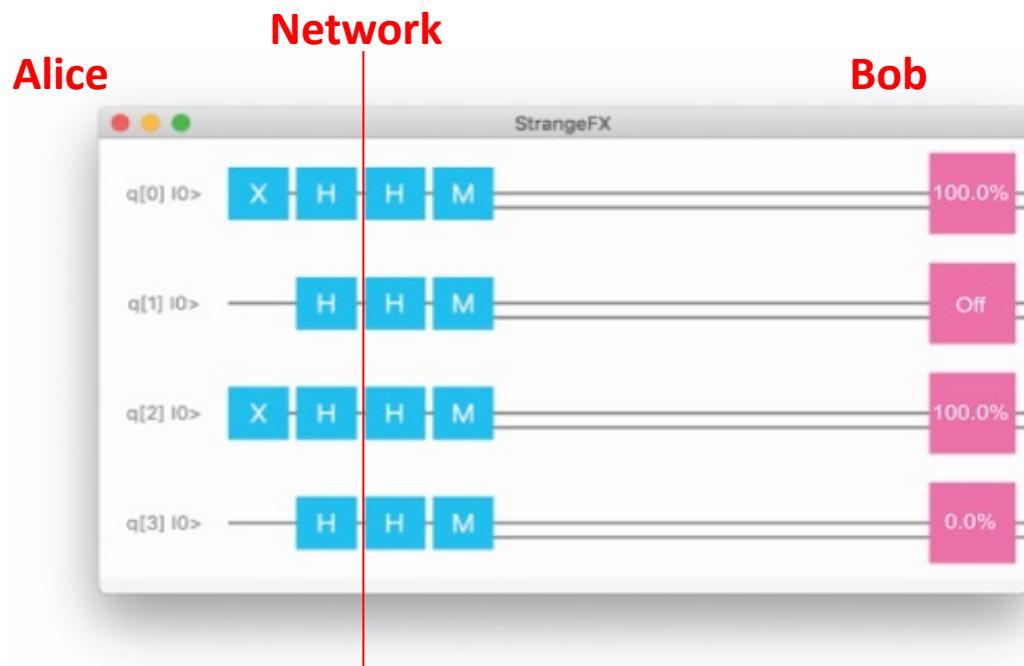
Github:

<https://github.com/critoma/quantum/blob/main/quantumjava/ch08/haha/src/main/java/org/redfx/javaqc/ch08/haha/Main.java>

# Secure Communication using Quantum Computing – Quantum Key Distribution (QKD) – Superposition Approach



Alice applies a Hadamard gate before sending a qubit, and Bob applies a Hadamard gate before measuring the qubit.



```
Random random = new Random();

boolean[] aliceBits = new boolean[SIZE];
for (int i = 0 ; i < SIZE; i++) {
    aliceBits[i] = random.nextBoolean();
}

QuantumExecutionEnvironment simulator =
    new SimpleQuantumExecutionEnvironment();
Program program = new Program(SIZE);
Step prepareStep = new Step();
Step superPositionStep = new Step();
Step superPositionStep2 = new Step();
Step measureStep = new Step();
for (int i = 0; i < SIZE; i++) {
    if (aliceBits[i]) prepareStep.addGate(new X(i));
    superPositionStep.addGate(new Hadamard(i));
    superPositionStep2.addGate(new Hadamard(i));
    measureStep.addGate(new Measurement(i));
}

program.addStep(prepareStep);
program.addStep(superPositionStep);
program.addStep(superPositionStep2);
program.addStep(measureStep);

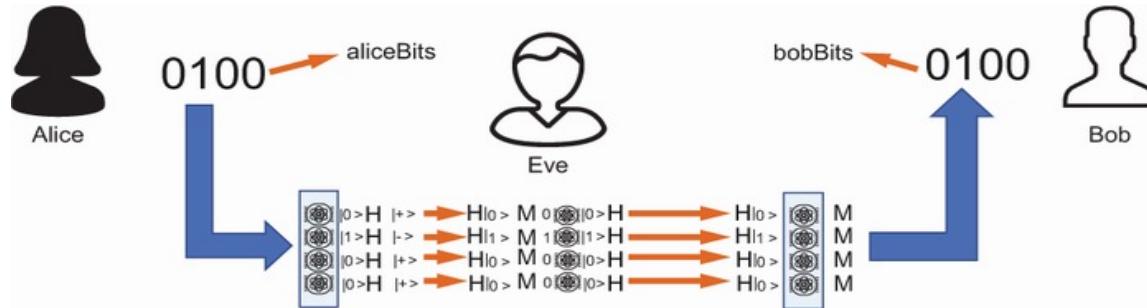
Result result = simulator.runProgram(program);
Qubit[] qubit = result.getQubits();

int[] measurement = new int[SIZE];
boolean[] bobBits = new boolean[SIZE];
for (int i = 0; i < SIZE; i++) {
    measurement[i] = qubit[i].measure();
    bobBits[i] = measurement[i] == 1;
    System.out.println("Alice sent " +
        (aliceBits[i] ? "1" : "0") +
        " and Bob received " +
        (bobBits[i] ? "1" : "0"));
}
```

- ➊ Alice creates a key containing random bits (SIZE=4).
- ➋ She initializes her qubits according to these random bits. A random bit of 0 will lead to a  $|0\rangle$  qubit; a random bit of 1 will lead to a  $|1\rangle$  qubit.
- ➌ Alice performs a Hadamard transformation to bring the qubit into a superposition and sends it over the network.
- ➍ Bob receives the qubit and performs a second Hadamard transformation.
- ➎ Bob measures the qubit.
- ➏ Adds the steps to the quantum program
- ➐ Executes the program
- ➑ Bob's bit is measured, and both the bits from Alice and Bob are printed. They should be bitwise equal.

Alice sent 0 and Bob received 0  
Alice sent 1 and Bob received 1  
Alice sent 0 and Bob received 0  
Alice sent 0 and Bob received 0

# Secure Communication using Quantum Computing – Quantum Key Distribution (QKD) – Superposition Approach

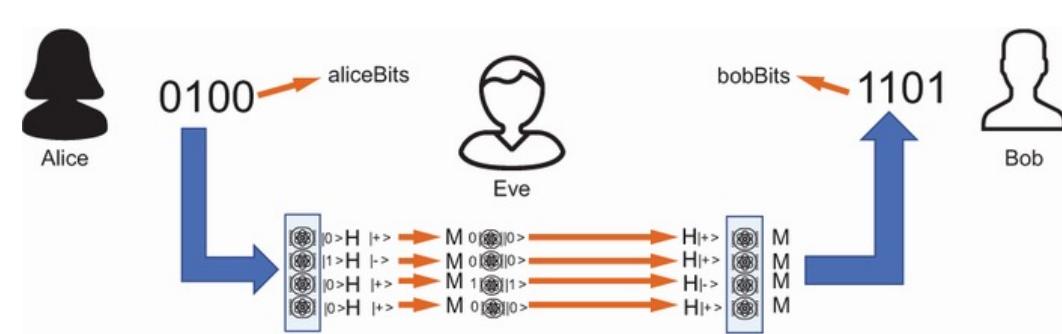


Eve measures the qubits sent by Alice and sends new qubits to Bob.

As you can see, things will go terribly wrong for Eve. When she measures the qubits from Alice, she randomly obtains a value of 0 or a value of 1. The qubits sent by Alice are all either in the  $|+\rangle$  state or the  $|-\rangle$  state. Both these states, when measured, will have a 50% chance of having the value 0 and a 50% chance of having the value 1. The real information is somehow hidden in the superposition composition. Eve is not aware of this, and the values she reads may be correct or may be wrong. For example, the first qubit in figure above, which originally was  $|0\rangle$ , is measured as  $|0\rangle$  by Eve, so she is correct. However, the second value, which originally was  $|1\rangle$ , is measured as  $|0\rangle$  by Eve. Hence, Eve will not obtain the correct shared key using this approach.

To make things worse, when Eve tries to hide her traces, she creates a new qubit based on her measurement and sends that to Bob. In the case of the first qubit, where she was lucky enough to measure 0, she constructs a new qubit  $|0\rangle$  and sends that to Bob. Bob, not realizing what has happened, assumes Alice sent him a qubit in a superposition, and he applies a Hadamard gate. This brings the qubit sent by Eve into a superposition. When Bob measures this qubit, he can measure either  $|0\rangle$  or  $|1\rangle$ . In the figure, Bob measures a 1, which is not what Alice sent.

In typical encryption algorithms, Alice and Bob use part of the transmitted bits to check if everything went correctly. They share the values of those bits (which makes those particular bits useless as they are no longer secure). If the values of the bits are different, Alice and Bob know something went wrong, and the whole key is not considered secure. Consequently, it is clear that using this approach, Eve can't obtain the secret without getting errors or being detected.



Eve applies a Hadamard gate before she measures the qubits sent by Alice and sends a new qubit to Bob after she applies another Hadamard gate

But Eve can learn. If Eve knows that Alice applied a Hadamard gate before sending the qubit over the wire, Eve might apply a Hadamard gate as well before measuring—doing exactly what Bob is doing. This will give her the information that would otherwise be obtained by Bob: the same bits that Alice used to prepare the qubits.

This won't help Eve, since Bob won't receive qubits. By measuring them, Eve destroys the superposition. However, now that Eve knows what Alice is doing, she can create new qubits by doing the same thing as Alice. Hence, Bob will receive a qubit in the same state as it would have come from Alice. He applies a Hadamard gate and then measures the qubit, and he gets the same bits used by Alice. This is schematically shown in figure above.

As a consequence, using this approach, not only Alice and Bob share the bits of their secret key—Eve does, too. Hence, this approach is not secure either.

Github:

# Secure Communication using Quantum Computing – Quantum Key Distribution (QKD) – BB84 (Guess)

The previous approach failed because Eve knew up front what Alice did and what Bob was going to do. In this section/BB84, we make it harder for Eve—or rather, impossible. Using superposition to prevent easy reading of a secret key.

The reason Eve can go undetected is that she manages to send a qubit in the same state to Bob as the one she intercepted from Alice. If Alice only uses a Pauli-X gate or nothing at all before transferring her qubit to Bob, Eve can measure the qubit, and she will obtain the original information. If Alice also applies a Hadamard gate, Eve needs to apply a Hadamard gate as well before measuring the qubit.

But what if Eve doesn't know whether Alice used a Hadamard gate? Should she apply a Hadamard gate herself, or not? Let's analyze the situation.

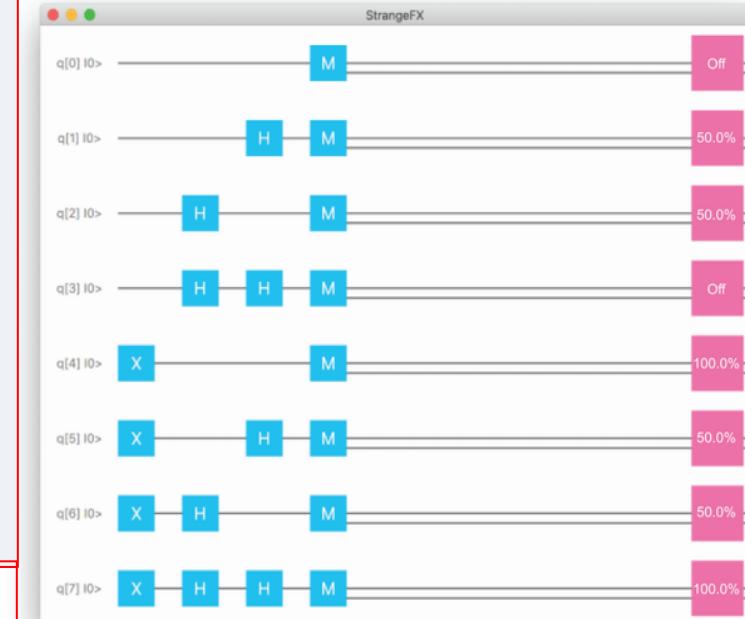
We have three variables that can each take two options, leading to eight scenarios:

- Alice sends a 0 or a 1.
- Alice applies a Hadamard gate or does not.
- Eve applies a Hadamard or does not.

The example (ch08/guess) simulates the possible outcomes for the eight different scenarios. The relevant part of this algorithm is shown in the following listing.

```
final int SIZE = 8;  
// The code in the for loop creates the eight scenarios.  
...  
// The visual output of the application  
for (int i = 0; i < SIZE; i++) {  
    if (i > (SIZE/2-1)) {  
        prepareStep.addGate(new X(i));  
    }  
    if ((i/2) % 2 == 1) {  
        superPositionStep.addGate(new Hadamard(i));  
    }  
    if (i%2 ==1) {  
        superPositionStep2.addGate(new Hadamard(i));  
    }  
    measureStep.addGate(new Measurement(i));  
}
```

- ❶ Considers the eight possible cases (numbered from 0 to 7)
- ❷ In the first four cases, Alice applies a Pauli-X gate.
- ❸ In cases 2, 3, 6, and 7, Alice applies a Hadamard gate.
- ❹ In cases 1, 3, 5, and 7, Eve applies a Hadamard gate.
- ❺ Performs a measurement



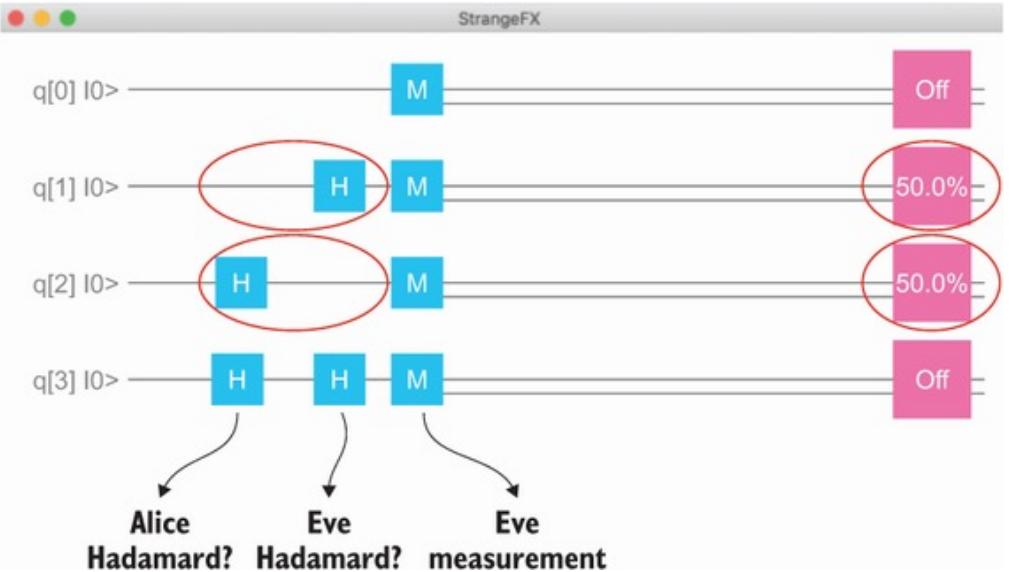
Different scenarios and their outcome

Github:

<https://github.com/critoma/quantum/blob/main/quantumjava/ch08/guess/src/main/java/org/redfx/javaqc/ch08/guess/Main.java>

## Secure Communication using Quantum Computing – Quantum Key Distribution (QKD) – BB84 (Guess)

No Pauli-X gate was applied to the first four qubits, so they represent a bit value of 0. Let's look at those four scenarios in a bit more detail (see figure). The analysis we do here also applies to the last four qubits, with the difference that the initial value in that case is 1.



Alice is sending a 0, and the measurement depends on the presence of Hadamard gates.

If both Alice and Eve apply a Hadamard operation or do not apply a Hadamard operation, Eve will measure a 0 value. But if either of them applies a Hadamard operation while the other doesn't, there is a 50% chance that Eve will measure a 0 and a 50% chance that Eve will measure a 1. These cases are circled in figure. The problem for Eve is that she can't tell whether her measurement is correct. She doesn't know if Alice applied a Hadamard gate, so she can't tell with certainty which scenario applies. To make things worse for Eve, she is also unable to create a qubit in the same state as the original one.

**Bob is confused, too ...**

If Eve can't reconstruct the original scenario, the same must be true for Bob. We assume that Alice and Bob have no upfront knowledge—otherwise, we would have fixed the bootstrap problem already.

In our algorithm, we will instruct Bob to randomly either apply a Hadamard gate or not before measuring the incoming qubit. The situation for Bob is then very similar to the situation for Eve. If Alice and Bob both apply a Hadamard gate, or if neither of them apply a Hadamard gate, Bob's measurement is guaranteed to correspond to Alice's initial value.

### Alice and Bob are talking

From the previous discussion, we can see that if Alice and Bob both use a Hadamard gate, or neither of them use a Hadamard gate, the original bit used by Alice and the measured bit used by Bob are guaranteed to be the same and can be used in a secret key. But how do they know that? The answer is simple: they tell each other whether they applied a Hadamard gate.

**NOTE:** This might sound surprising. If Alice and Bob tell each other over a public channel whether they applied a Hadamard gate, Eve may be listening! The trick is this: Alice and Bob only share that information with each other after Bob has received and measured his qubit. At that moment, Eve can no longer do anything. If Eve knew the information up front, she could manipulate the system, since she could easily reproduce the qubit from Alice if she knew whether Alice applied a Hadamard gate. But she has to make a decision before sending a qubit to Bob—and that decision is made based on a measurement of the qubit she intercepted from Alice. Hence, all information in that qubit is destroyed. It's a pity for Eve, but the public information is useless.

When Alice and Bob have each other's information about the Hadamard gates, they simply remove the values measured on qubits that had non-matching Hadamard gates. The remaining values are guaranteed to be correct.

**NOTE:** Alice and Bob only share the information about the Hadamard gates. They do not share the initial value (in Alice's case) or the measured value (in Bob's case). They know, though, that those values are equal, and they can use them as part of a shared secret key.

Typically, Alice and Bob use part of their secret key to check whether the connection was eavesdropped on, for example by reserving some space to add a checksum. If Eve isn't discouraged by the fact that she can't get the original key without getting noticed, she might still try to get the key—but she will have to make wild guesses about whether to apply a Hadamard gate. If she makes the wrong guess, she will send a qubit to Bob that is in a different state from the one Alice sent to Bob. So, there is a chance that Bob will measure a different value than Alice used. If both or neither Alice and Bob apply a Hadamard gate and the initial value from Alice is different from the value measured by Bob, Alice and Bob know that the connection was tampered with.

# Secure Communication using Quantum Computing – Quantum Key Distribution (QKD) – BB84 (QKD in Java)

## BB84 - INITIALIZING VARIABLES

First we initialize some arrays:

```
final int SIZE = 8;  
Random random = new Random();  
  
boolean[] aliceBits = new boolean[SIZE];  
  
boolean[] bobBits = new boolean[SIZE];  
  
boolean[] aliceBase = new boolean[SIZE];  
  
boolean[] bobBase = new boolean[SIZE];
```

- ①
- ②
- ③
- ④
- ⑤
- ⑥

- ❶ Creates a key with a maximum of 8 bits. Keep in mind that we have to remove the bits for which Alice and Bob used a different strategy (Hadamard or not), so on average, the real length of the key is half of the size specified here.
- ❷ Creates an instance of `java.util.Random`, which will be used later to generate random boolean values
- ❸ In this array, Alice keeps the random bits she generates, and those are for the base she uses: if a specific bit of these random bits is 1, she will apply a Hadamard gate to the qubit with the same index.
- ❹ In this array, Bob stores the bits he measured.
- ❺ When Alice decides to apply a Hadamard gate for a specific qubit, the corresponding value in this array is set to true.
- ❻ When Bob decides to apply a Hadamard gate for a specific qubit, the corresponding value in this array is set to true.

## BB84 - PREPARING THE STEPS

The quantum application we create contains different steps. The first two steps are performed by Alice, and third and fourth steps are performed by Bob:

```
Step prepareStep = new Step();  
Step superPositionStep = new Step();  
Step superPositionStep2 = new Step();  
Step measureStep = new Step();
```

- ①
- ②
- ③
- ④

- ❶ Alice applies a Pauli-X gate if the random bit under consideration is 1.
- ❷ Alice applies a Hadamard gate (or not).
- ❸ Bob applies a Hadamard gate (or not).
- ❹ Bob measures the result.

# Secure Communication using Quantum Computing – Quantum Key Distribution (QKD) – BB84 (QKD in Java)

```
for (int i = 0; i < SIZE; i++) {  
    aliceBits[i] = random.nextBoolean();  
    if (aliceBits[i]) {  
        prepareStep.addGate(new X(i));  
    }  
  
    aliceBase[i] = random.nextBoolean();  
    if (aliceBase[i]) {  
        superPositionStep.addGate(new Hadamard(i));  
    }  
  
    bobBase[i] = random.nextBoolean();  
    if (bobBase[i]) {  
        superPositionStep2.addGate(new Hadamard(i));  
    }  
  
    // Finally, Bob measures the result  
    measureStep.addGate(new Measurement(i));  
}
```

①

②

③

④

⑤

⑥

## BB84 - FILLING THE STEPS

All the steps are created for each bit that can be part of the key. Three of those steps depend on random values.

Based on a first random value, a *Pauli-X* gate is applied to *prepareStep*. There is a 50% chance that the Pauli-X gate is applied, causing the qubit to be in the  $|1\rangle$  state, and a 50% chance that no gate is applied and the qubit stays in the  $|0\rangle$  state.

The second random value defines whether a Hadamard gate is applied to *superPositionStep*, which is executed by *Alice*. The next step, *superPositionStep2*, uses a random value to decide whether *Bob* applies a Hadamard gate:

- 1 The following steps are applied for each bit that is a candidate for the secret key.
- 2 A random value determines whether Alice's bit is 0 or 1.
- 3 If Alice's bit is 1, applies an X gate to the  $|0\rangle$  state
- 4 A random value (stored in the *aliceBase* array) decides whether Alice applies a Hadamard gate.
- 5 A random value (stored in the *bobBase* array) decides whether Bob applies a Hadamard gate.
- 6 Bob measures the qubit.

```
QuantumExecutionEnvironment simulator =  
    new SimpleQuantumExecutionEnvironment();  
program.addStep(prepareStep);  
①  
program.addStep(superPositionStep);  
②  
program.addStep(superPositionStep2);  
program.addStep(measureStep);  
  
Result result = simulator.runProgram(program);  
③  
Qubit[] qubit = result.getQubits();  
④
```

①

②

③

④

## BB84 - EXECUTING THE APPLICATION

We now have to execute the application in our quantum simulator.

- 1 Creates a *QuantumExecutionEnvironment*
- 2 Adds the steps created in the previous phases
- 3 Runs the quantum program on the simulator
- 4 Assigns the results to an array of qubits

# Secure Communication using Quantum Computing – Quantum Key Distribution (QKD) – BB84 (QKD in Java)

## BB84 - PROCESSING THE RESULTS

```
int[] measurement = new int[SIZE];
for (int i = 0; i < SIZE; i++) {
    measurement[i] = qubit[i].measure();
    bobBits[i] = measurement[i] == 1;
    if (aliceBase[i] != bobBase[i]) {
        System.err.println("Different bases used,
                           ignore values "+aliceBits[i]+
                           " and "+bobBits[i]);
    } else {
        System.err.println("Same bases used.
                           Alice sent " + (aliceBits[i] ? "1" : "0")
                           + " and Bob received "
                           + (bobBits[i] ? "1" : "0"));
        key.append(aliceBits[i] ? "1" : "0");
    }
}
```

Same bases used. Alice sent 1 and Bob received 1  
Same bases used. Alice sent 0 and Bob received 0  
Same bases used. Alice sent 1 and Bob received 1  
Different bases used, ignore values false and true  
Same bases used. Alice sent 1 and Bob received 1  
Different bases used, ignore values false and true  
Same bases used. Alice sent 1 and Bob received 1  
Different bases used, ignore values true and true  
Secret key = 10111

- ➊ For each candidate bit, we run the following steps that evaluate whether the bit should be part of the key.
- ➋ Sets the bit in the bobBits array to the measurement value of the qubit
- ➌ If the random bases chosen by Alice and Bob for this bit are different, ignores values and prints a message
- ➍ Otherwise, Alice and Bob used the same Hadamard strategy. The initial value from Alice matches the measurement from Bob.
- ➎ This bit now becomes part of the secret key.



Output obtained by running the BB84 application

From both the text output and the graphical output, it is clear that Alice and Bob used the same Hadamard strategy for bits 0, 1, 2, 4, and 6. Those five bits are thus part of the secret key. The other bits are useless since Alice and Bob used a different Hadamard strategy (either Alice applied one and Bob didn't, or the other way round).

Github:

<https://github.com/critoma/quantum/blob/main/quantumjava/ch08/bb84/src/main/java/org/redfx/javaqc/ch08/bb84/Main.java>

# Secure Communication using Quantum Computing – Quantum Key Distribution (QKD) – BB84 (QKD in Java)

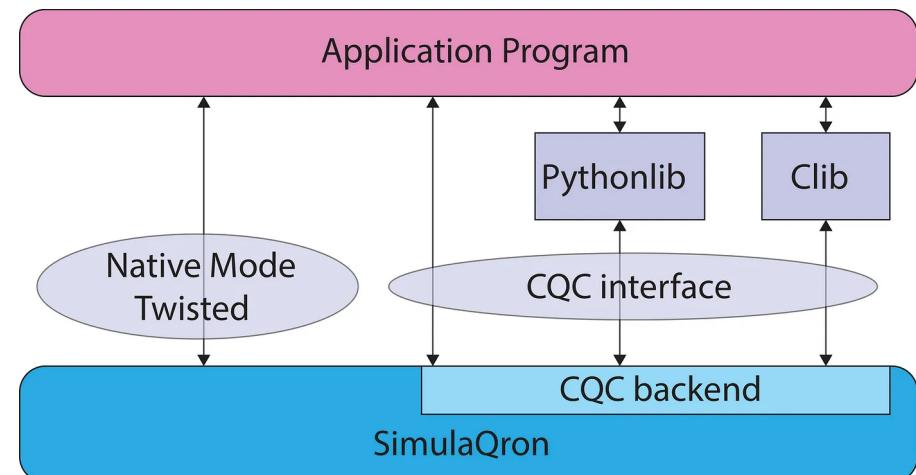
An interesting project that provides a way to transfer qubits from one node to another is the **SimulaQron** project from **QuTech** (<http://qutech.nl>). One of the goals of QuTech is to build a **network of quantum computers using fiber optic cables**. From the code we have shown so far, it should be clear that even with a very limited number of qubits available, quantum networking has huge benefits. A single qubit can be used to generate a shared secret bit between two parties. By repeating this process as often as necessary, a shared secret with as many bits as required can be obtained.

While work on the physical quantum network is being carried out, **QuTech is also building a protocol stack similar to a protocol stack for classical networking**. Such a stack makes abstraction of the hardware implementation and shields developers from the low-level implementations. Developers using the top layer of such a protocol stack can create applications that can then run on a different implementation of the protocols. This is useful because the same code can work with different kinds of hardware; plus it allows code to use simulators while the hardware is not yet available.

**SimulaQron** provides a **protocol called CQC** that allows high-level programming languages (**Java, Python, C, and so on**) to interact with the implementation and use quantum networking functionality. **Support for the CQC protocol is being added to the Strange simulator**. As a consequence, quantum applications you write using Java Strange will work on a distributed system. In the first phase, this will be a network containing quantum simulators, but in a later phase—once there are real quantum nodes in a network—this should also work on real hardware.

## Summary

- Secure communication is an important aspect in today's IT landscape, and hackers are eager to intercept messages and security keys.
- Using a one-time pad, a message can be encrypted with a key that is not reused later. Every message uses a new key, making it harder for hackers to intercept.
- The **BB84** algorithm is a famous quantum algorithm that generates a one-time pad.
- Using **Strange**, you can implement the **BB84** algorithm in Java.



**SimulaQron:** <http://www.simulaqron.org>  
<https://softwarequtech.github.io/SimulaQron/html/index.html>  
**CQC**, that stands for **Classical-Quantum Combiner**:  
<https://softwarequtech.github.io/CQC-Python/index.html>

# Quantum Algorithms & Coding – Deutsch-Jozsa algorithm

## Goals of the section:

- Obtaining information from classical functions
- Function evaluations vs. function properties
- Quantum gates that correspond to classical black box functions
- Understanding the Deutsch algorithm and the Deutsch-Jozsa algorithm

*The first approach (using the calculator) gave us a result of a division, and it provided us with the real answer. The second approach (sum of the individual digits) only provided the real answer, not the outcome of the division.*

*The relevance of this is that in many cases, we are interested in a specific property of something (e.g., a number or a function). We are not interested in a function evaluation, but we somehow want to obtain information about the function. Evaluating the function is often the easiest way to do so, but it can be more efficient to indirectly look at the properties of the function and draw conclusions from there.*

*Do you know if the number 168,153 can be divided by 3?*

For example, you can simply take a calculator and obtain the result:

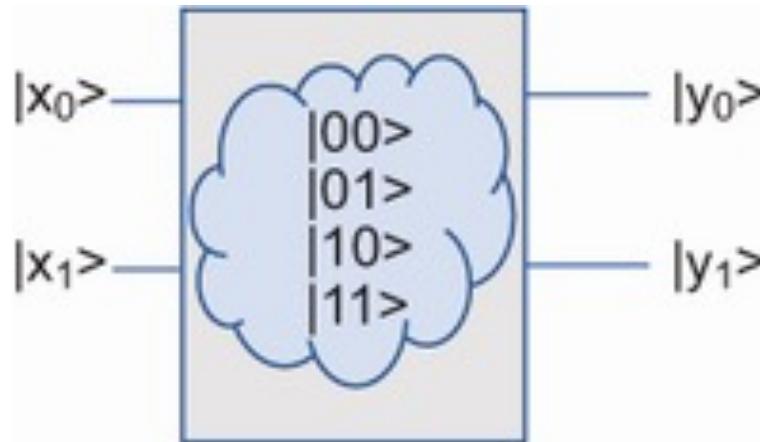
$$168153 / 3 = 56051$$

There is another simple approach to find the answer to this question: take the sum of the individual digits that compose the number, and see if that sum can be divided by 3. If so, the original number can be divided by 3 as well.

$$1 + 6 + 8 + 1 + 5 + 3 = 24$$

## Quantum Algorithms & Coding – Deutsch-Jozsa algorithm

This is of particular interest in quantum computing. A quantum computer with  $n$  qubits that needs to examine a specific function can only do one function evaluation at a time. Applying the quantum circuit to a given specific set of input qubits will result in a modified state of these qubits. Measuring them gives a particular result, and if you want a new result, you need to run the circuit again. Even though we can apply Hadamard gates to the input qubits to bring them into superposition—which allows us to evaluate the qubits in the 0 and 1 states simultaneously—we can't magically create new qubits that will hold the information of the different cases. This is shown in figure 9.1 for a system with two qubits.



A quantum system with two qubits can do many evaluations, but only two qubits can be measured

The internal computations can contain the equivalent of many evaluations, but we can't obtain those simultaneously. We are limited to a result of  $n$  qubits. But that is often enough to solve problems. In the case of our number being a multiplicator of 3 or not, a single qubit is enough to store the answer. We don't need to evaluate the division function.

*In this section, we demonstrate this approach using quantum computing.*

We investigate a property of a function  $f$  acting on  $n$  bits, without being interested in the individual function evaluations. We show that retrieving the property in the classical way requires  $(2^{n-1} + 1)$  function evaluations.

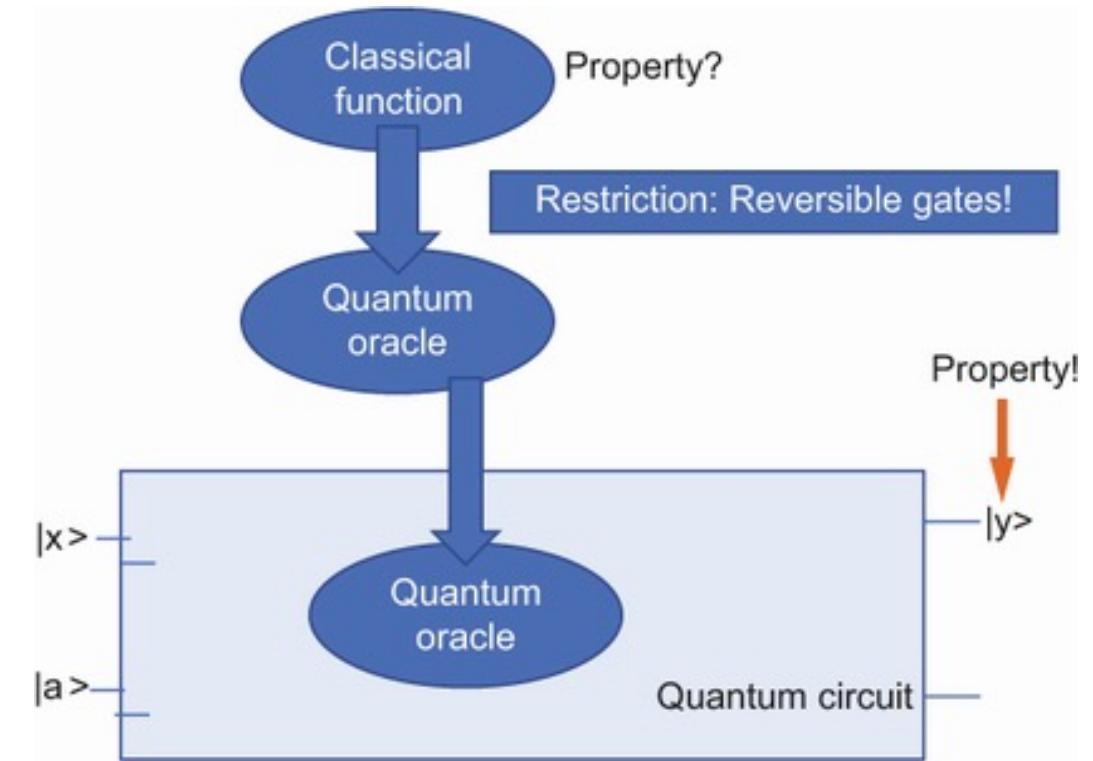
*With the quantum algorithm, the property can be obtained with a single evaluation.*

## Quantum Algorithms & Coding – Deutsch-Jozsa algorithm

The functions we use are very simple, and there is no direct use case for this problem. But it demonstrates an essential aspect of quantum computing, and it explains why quantum computing is often associated with “exponential” complexity. You can easily see that the more input bits the function has, the harder it becomes to solve the problem in a classical way. The number  $n$  is indeed in the exponent, and as we showed in chapter 1, exponential functions quickly result in huge values. If a quantum algorithm can fix the same problem in just a single evaluation (or, in general, in less than an exponential number of equations), this is a huge advantage for a quantum computer.

We will gradually come to the algorithm that achieves this.

First we talk about properties of functions and how to obtain them in a classical way. Next, we convert the functions into quantum blocks called *oracles*. We show that there are some requirements for doing this. ***Once we can create a Quantum ORACLE that represents a classical function, this oracle can be used in a quantum circuit.*** Evaluating this quantum circuit once results in the property of the function we are looking for.



**Approach for finding the property of a function**

## Quantum Algorithms & Coding – Deutsch-Jozsa algorithm

### *Properties of functions*

In most typical cases where functions are involved, you are interested in finding the result of a function. For example, consider the **function  $y = y(x) = x^2$** .

If we want to know the value of this function for, for example  $x = 4$  and  $x = 7$ , we need to evaluate this function:

$$y(4) = 4^2 = 16$$

$$y(7) = 7^2 = 49$$

In some cases, though, the function evaluations are not important, but the characteristics of the functions are. In this area, quantum algorithms can be helpful. An example that we will discuss in chapter 11 is the periodicity of a function. We are not interested in the individual evaluations of a function, but we are interested in the periodicity. A periodic function is a function where the same pattern of values comes back with a fixed periodicity. Here's an example:

$x \rightarrow 0\ 1\ 2\ 3\ 4\ 5\ 6$

$y \rightarrow 7\ 9\ 5\ 7\ 9\ 5\ 7$

From this table, you can tell that the function has a periodicity of 3: for every value of  $x$ , the result of the function is the same as the function applied to  $x + 3$ . This is an example where the property of a function can be more interesting than the function evaluations themselves.

## Quantum Algorithms & Coding – Deutsch-Jozsa algorithm

### Constant and balanced functions

In general, the functions we consider are noted as  $f(x)$ , where  $f$  is the function that operates on an input variable called  $x$ . The evaluation of **a function** for a specific input is also called the result and sometimes noted as  $y$ , where  $y = f(x)$ .

In this section, we start with a very simple family of functions that have simple properties. We start with a function with only a single input bit, and we extend it later to a function with  $n$  input bits. In all cases, the result of the function is either 0 or 1.

The functions we discuss here have a special property: they are either **balanced functions** or **constant functions**.

A function is called:

- **constant** when the result is not dependent on the input. In our case, that means the result is either 0 for all input cases or 1 for all input cases;
- **balanced** when the result is 0 in 50% of the cases and 1 in the other cases.

The **Deutsch algorithm**, which we discuss shortly, deals with a **function** called  **$f$**  that takes a single bit (a Boolean value) as its input and produces a single bit as well – **the function only operates on 0 and 1, and its result is either 0 or 1**.

The combination of **two input options** and **two output options** leads to four possible cases for this function, which we name  $f_1$ ,  $f_2$ ,  $f_3$ , and  $f_4$ :

- $f_1: f(0) = 0 \text{ and } f(1) = 0 \Rightarrow f_1 \text{ constant}$
- $f_2: f(0) = 0 \text{ and } f(1) = 1 \Rightarrow f_2 \text{ balanced}$
- $f_3: f(0) = 1 \text{ and } f(1) = 0 \Rightarrow f_3 \text{ balanced}$
- $f_4: f(0) = 1 \text{ and } f(1) = 1 \Rightarrow f_4 \text{ constant}$

From those definitions, it appears that  **$f_1$  and  $f_4$  are constant functions**, and  **$f_2$  and  $f_3$  are balanced functions**.

In many classical algorithms, it is important to know the output of a function for specific values. In many quantum algorithms, on the other hand, it is useful to know the properties of the function under consideration.

## Quantum Algorithms & Coding – Deutsch-Jozsa algorithm

This is part of the “different thinking” that is required when considering quantum algorithms.

Thanks to superposition, a quantum computer can evaluate many possibilities simultaneously; but since obtaining a result requires a measurement, the superposition is gone, and we are back to a single value.

Hence, the added value is in the function evaluation and not in the result of the function evaluation.

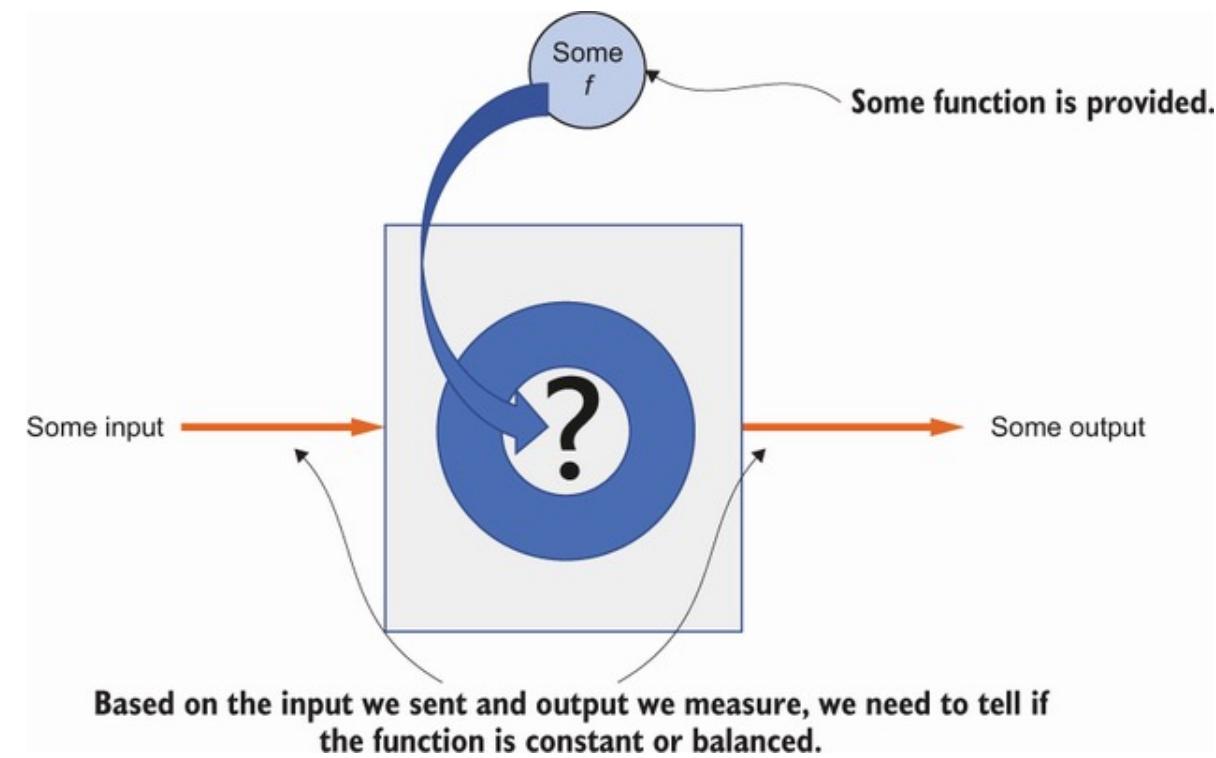
In the **Deutsch algorithm**, a **function is provided, but we don’t know what function it is**. We know it is  $f_1$ ,  $f_2$ ,  $f_3$ , or  $f_4$ , but that’s all we know.

We are now asked to find out if this function is constant or balanced.

Our task is not to determine whether the provided function is  $f_1$ ,  $f_2$ ,  $f_3$ , or  $f_4$ . We are asked about a property of the function, not the function itself.

How many function evaluations do we need before we can answer this question (function property of being constant or balanced) with 100% certainty? If we do only a single evaluation (we only calculate either  $f(0)$  or  $f(1)$ ), we don’t have enough information.

Suppose we measure  $f(0)$  and the result is 1. From the earlier table, it seems that, in this case, our function is either  $f_3$  (which is balanced) or  $f_4$  (which is constant). So, we don’t have enough information. If the result of measuring  $f(0)$  is 0, the table shows that the function is either  $f_1$  (which is constant) or  $f_4$  (which is balanced). Again, this proves that measuring  $f(0)$  is not enough to conclude whether the function is constant or balanced. It can be either.



Finding the properties of an unknown function by doing evaluations

# Quantum Algorithms & Coding – Deutsch algorithm

*It turns out we need two classical function evaluations before we can determine whether the provided function is constant or balanced.  
Let's write a Java application that demonstrates this. You can find this application in the ch09/function directory of the examples.*

```
static final List<Function<Integer, Integer>> functions = new ArrayList<>();  
  
static {  
    Function<Integer, Integer> f1 = (Integer t) -> 0; 1  
    Function<Integer, Integer> f2 = (Integer t) -> (t == 0) ? 0 : 1;  
    Function<Integer, Integer> f3 = (Integer t) -> (t == 0) ? 1 : 0;  
    Function<Integer, Integer> f4 = (Integer t) -> 1;  
  
    functions.addAll(Arrays.asList(f1, f2, f3, f4));  
}  
  
public static void main(String[] args) {  
    Random random = new Random();  
    for (int i = 0; i < 10; i++) { 2  
        int rnd = random.nextInt(4);  
        Function<Integer, Integer> f = functions.get(rnd); 3  
        int y0 = f.apply(0); 4  
        int y1 = f.apply(1);  
        System.err.println("f" + (rnd + 1 + " is a " 5  
            + ((y0 == y1) ? "constant" : "balanced") + " function"));  
    } // end for  
} // end main
```

<https://github.com/critoma/quantum/tree/main/quantumjava/ch09/function>

- 1** Prepares the four possible functions. This step needs to be done only once.
- 2** We will do 10 experiments.
- 3** Picks a random function, not knowing anything about its implementation
- 4** Performs two function evaluations: one for input 0 and one for input 1
- 5** If the results of those two evaluations are similar, the function is constant; otherwise, the function is balanced.

In this code, we create the four possible functions in a static block. We do this is because we want to stress that the creation of the function and the determination of whether they are constant or balanced should be considered two independent processes. After the functions are created, the application really starts. Inside the for loop, a random function is picked. **Based on the two function evaluations, we can determine whether the function is constant or balanced.**

A possible output of this application is the following:

f4 is a constant function  
f4 is a constant function  
f3 is a balanced function  
f1 is a constant function  
f2 is a balanced function  
...  
f2 is a balanced function

*As expected, the application has the correct answers for every loop. But in every loop, we performed two evaluations of the function. As we showed earlier, a single evaluation would not be sufficient to conclude whether the function is balanced.*

## Defining an ORACLE

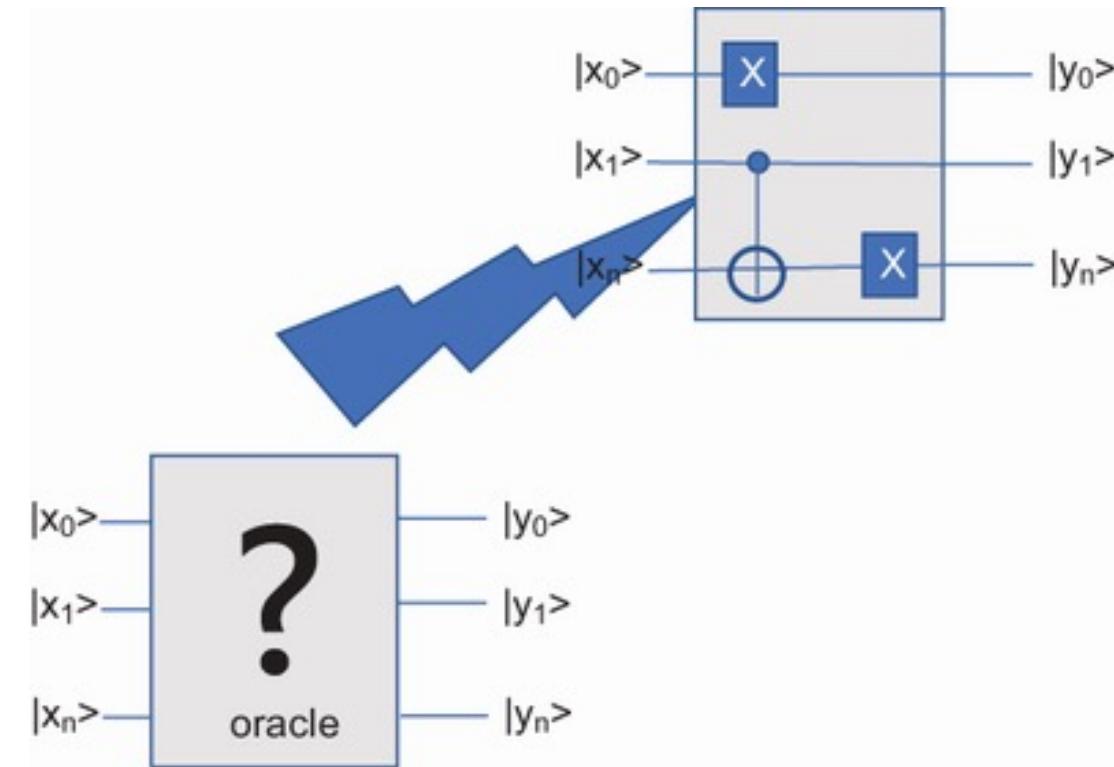
## Quantum Algorithms & Coding – Deutsch algorithm

In many **quantum algorithms**, the term **ORACLE** is used. We will use an **ORACLE** when we create the **Deutsch algorithm**, so some explanation is needed now.

An **oracle** is used to describe a **quantum black box** — similar to how each of the earlier functions ( $f_1, f_2, f_3$ , and  $f_4$ ) can be described as a **classical black box**. *Internally, the functions had some computational flow; but in the main loop of the program, we pretended we didn't know about the internals — we simply evaluated the functions for different input values.*

The same concepts can be applied to an **oracle**. *Internally, an ORACLE is composed of one or more quantum gates, but we typically don't know which gates.* By querying the oracle (e.g., by sending input and measuring the output), we can learn more about the properties of the oracle. *Because an oracle is composed of quantum gates, the oracle itself also needs to be reversible.*

An **oracle** can thus be considered the quantum equivalent of the black-box functions. Both an oracle and a function perform some calculations, but we don't know the internal details about these calculations.



A Quantum ORACLE, acting as a black box in a quantum circuit, is internally composed of a number of gates.

Keep in mind that Quantum Gates are REVERSIBLE => applied twice in row, the qubit comes back to the initial state  
<https://github.com/critoma/quantum/blob/main/quantumjava/ch09/reversibleX>

Let's look at an example of an oracle used in a simple quantum application. Using the Strange simulator, you define an **ORACLE** by providing ***the matrix that is the mathematical representation of the oracle***. That sounds like we are cheating: ***are we really defining something that we later claim we don't know anything about?*** The answer is yes. We indeed create this **Oracle**, since someone has to do it.

In real quantum applications, the **ORACLE** is an external component given to us.

**NOTE:** The *creation of the function* and the *creation of the oracle* should be considered totally separated processes. In the upcoming algorithms, ***we assume that someone created an oracle for us***. The algorithm itself has no clue how the **oracle** was created, how complex or simple it is, etc. ***This is often confusing, since to demonstrate the algorithm, we obviously need an oracle***. However, the complexity of creating the oracle should not be considered part of the complexity of the algorithm.

***Just assume that someone (yourself, another developer, a real piece of hardware, or nature itself) created the oracle and provided it to you.***

We learned before that all gates can be represented by a matrix. An **oracle** contains ***zero or more gates***, and therefore the ***oracle itself can also be represented by a matrix***. An **oracle** can be considered part of the quantum circuit that is given to you. You can't see the gates that together constitute the **oracle**, but you can use them in your quantum programs.

***In Strange, an oracle is defined by providing the matrix that represents it. In a real hardware situation, the oracle is available as is. In a software simulation, we somehow need to define the oracle's behavior, so a matrix needs to be provided.***

An oracle, pictured as a vertical box, is used in this quantum circuit. First a Hadamard gate is applied, followed by the **ORACLE**.



```

QuantumExecutionEnvironment simulator =
    new SimpleQuantumExecutionEnvironment();
Program program = new Program(2);
Step step1 = new Step();
step1.addGate(new Hadamard(1));
①

Complex[][][] matrix = new Complex[][][]{
    {Complex.ONE, Complex.ZERO,
        Complex.ZERO, Complex.ZERO},
    {Complex.ZERO, Complex.ONE,
        Complex.ZERO, Complex.ZERO},
    {Complex.ZERO, Complex.ZERO,
        Complex.ZERO, Complex.ONE},
    {Complex.ZERO, Complex.ZERO,
        Complex.ZERO, Complex.ZERO}
    {Complex.ZERO, Complex.ZERO,
        Complex.ONE, Complex.ZERO}
};

Oracle oracle = new Oracle(matrix);

Step step2 = new Step();
step2.addGate(oracle);

program.addStep(step1);
program.addStep(step2);

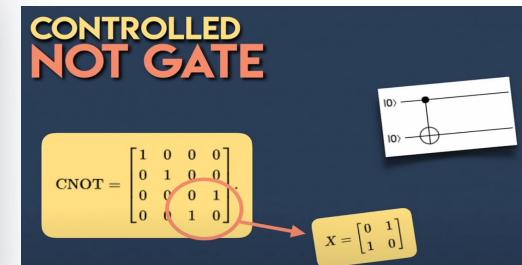
Result result = simulator.runProgram(program);
Renderer.showProbabilities(program, 1000);
Renderer.renderProgram(program);
②
③
④
⑤
⑥
⑦

```

We will create some code in which we apply an oracle in a quantum program. We create the oracle by providing a matrix that may look familiar to you. It is recommended that you not look at the contents of the matrix, though, as doing so takes away the mystery of the oracle. Instead, you can look at the result of the entire program and try to find out what the oracle is doing.

<https://github.com/critoma/quantum/tree/main/quantumjava/ch09/oracle>

- 1 Creates a quantum program that requires two qubits
- 2 The first step applies a Hadamard gate to the second qubit.
- 3 Creates a **matrix containing complex numbers**. For now, we don't interpret these numbers.
- 4 Creates an **oracle** based on this matrix
- 5 Creates a second step in which the oracle is applied
- 6 Adds both steps to the quantum program
- 7 Executes the program and displays its circuit and the results of 1,000 runs

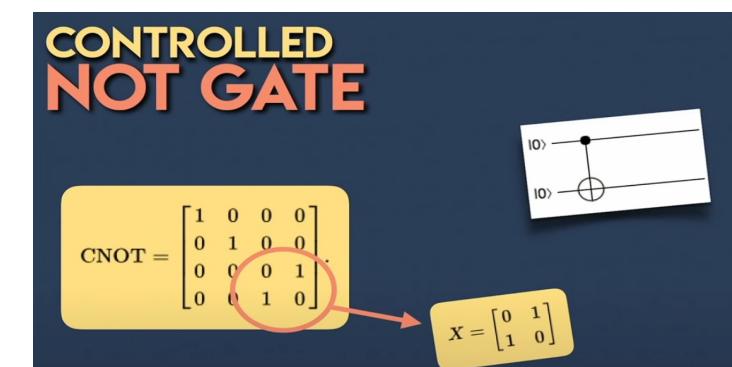




Statistical results of running 1,000 times this circuit (a quantum program containing an oracle)

If you look at those statistical results, it seems that there are only two possible outcomes:  $|00\rangle$  or  $|11\rangle$ .

Remember from **Bell State / Entanglement** topic that a circuit with two entangled qubits has the same probabilities: there is a 50% of measuring  $|00\rangle$  and a 50% chance of measuring  $|11\rangle$ . This is an indication that the **ORACLE** we created, combined with the initial Hadamard gate, results in two entangled qubits. In Bell State sample, you created two entangled qubits by applying a **CNOT gate** after a **Hadamard gate**. So, we learn that the **oracle** we created behaves like a **CNOT gate**. If we cheat and look at the contents of the **oracle**, we see that the matrix that represents **the oracle matches the matrix of the CNot gate**. **This exercise shows that we can apply oracles to a quantum circuit. We can apply an oracle without knowing its internal details.**



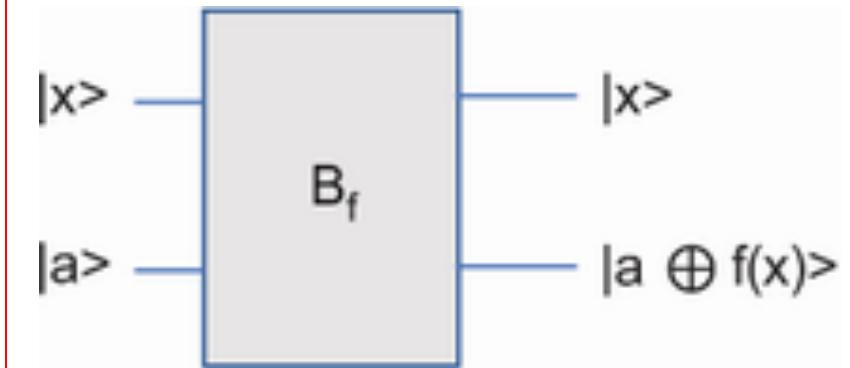
In the Deutsch algorithm, we show that a single evaluation is enough to find out if a provided function is constant or balanced. Before we can do that, we need to convert the classical function into a quantum operation.

We can't simply apply a function to a qubit. Remember, we explained that all quantum gates need to be reversible. A function that, after being applied, makes it impossible to retrieve the original input can't be used in a quantum circuit. Therefore, the function first needs to be transformed into a reversible oracle.

In this section, we demonstrate how oracles can be created that can then be used in the Deutsch algorithm we explain in the next section. Similar to how a classical function is handed to the classical algorithm, an oracle is handed to the quantum algorithm.

Every classical function that we described earlier in this chapter can be represented by a specific oracle. Since we had four possible functions, we also have four possible oracles.

The general way to construct an oracle based on a function is shown in figure A. In this approach, we have an input qubit called  $|x\rangle$  and an additional qubit named  $|a\rangle$ .



A. Oracle used in the Deutsch algorithm

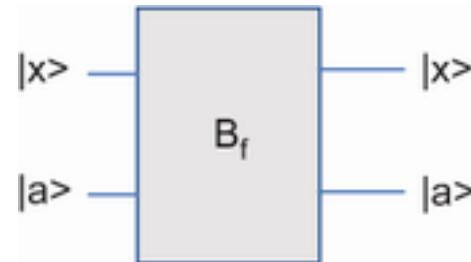
## From Functions to ORACLE

### \* Constant Function

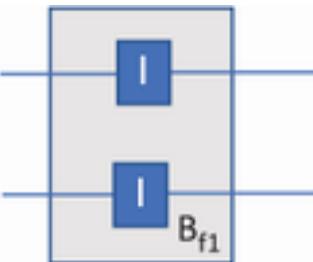
The first function,  $f_1$ , is a constant function that returns 0 regardless of the input. Hence, since  $f(x) = 0$  for any value of  $x$ , the output status of the second qubit can be simplified as follows:

$$a \oplus f(x) = a \oplus 0 = a$$

The resulting **oracle** can be pictured as in figure B. In this case, the **oracle** is simply the **Identity matrix**. Both  $|x\rangle$  and  $|a\rangle$  are unaltered between the input and the output. The hidden logic inside the **oracle** can thus be represented by the scheme in figure C.



B. Oracle used in the Deutsch algorithm for  $f_1$



C. Oracle circuit for  $f_1$

As a result, the matrix representing the oracle is an Identity matrix: it does not alter the probabilities when measuring the qubits. Multiplying this matrix with any probability vector yields the original probability vector.

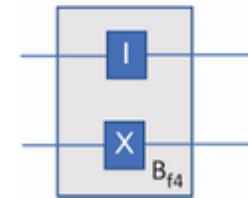
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**NOTE:** The circuit shown in figure isn't the only possible circuit that results in the Identity matrix. Many other circuits operate on two qubits and return the two qubits in the same state. For example, applying two Pauli-X gates on each qubit results in the exact same state. This is part of the black-box aspect of an oracle: we don't know the internal details, and we are typically not interested in those. We want to investigate a specific property of the oracle, not its internal implementation.

The fourth function,  $f_4$ , is a constant function that always returns the value 1, regardless of the input. As a consequence, the output of the second qubit after applying the oracle can be written as follows:

$$a \oplus f(x) = a \oplus 1 = \bar{a}$$

The horizontal bar above a variable indicates that this is the inverted variable, which in this case corresponds to a Pauli-X gate being applied to  $|a\rangle$ . So, this oracle can be schematically presented as shown in figure D.



D. Oracle circuit for  $f_4$

Prove that the matrix corresponding to this oracle can be written as follows:

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

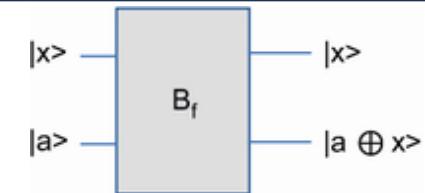
Let's have a look at the second classical function,  $f_2$ . This function is defined as follows:

$$f(0) = 0$$

$$f(1) = 1$$

This can also simply be written as  $f(x) = x$ .

Using this in the general description of the oracle, as shown in figure A, the scheme simplifies to figure E.

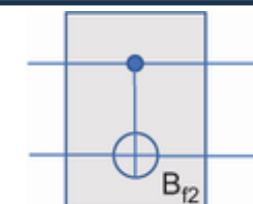


E. Oracle used in the Deutsch algorithm for  $f_2$

This is exactly the state that would be obtained if the oracle had a CNot gate. Hence, a possible circuit for the oracle corresponding to the  $f_2$  function is shown in figure F.

The matrix representation of this oracle is, thus, also the matrix representation of the CNot gate:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$



F. Oracle circuit for  $f_2$

***Calculate the matrix representation of the oracle corresponding to the  $f_3$  function. The result you should obtain is this:***

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

The **Deutsch algorithm** requires only **a single evaluation** of the **ORACLE** to know whether the function under consideration is:  
- **constant or balanced.**

Let's start with a naive approach and assume that all we need to do is apply the oracle and measure the result.

<https://github.com/critoma/quantum/tree/main/quantumjava/ch09/applyoracle>

Before we explain the algorithm, we first point to the part of the code where the different **oracles** are created. As we stated before, **the creation of an oracle IS NOT part of the algorithm** that tries to find out whether a function is balanced. *Although for practical reasons, we create the oracle in the same Java class file as the algorithm, it should be stressed that the creator of the oracle (who might know the outcome of the problem) and the creator of the algorithm are not the same.*

From the previous section, it should be clear that there are **four different types of oracles**. *An infinite number of oracles can be used to represent the simple functions we discussed at the beginning of this section, but they all corresponds to one of the four gate matrices that we showed in the previous section. The algorithm will ask to pick a random oracle, and the code that constructs this random oracle is as follows.*

```

static Oracle createOracle(int f) {
    Complex[][] matrix = new Complex[4][4];

    switch (f) {
        case 0:
            matrix[0][0] = Complex.ONE;
            matrix[1][1] = Complex.ONE;
            matrix[2][2] = Complex.ONE;
            matrix[3][3] = Complex.ONE;
            return new Oracle(matrix);
        case 1:
            matrix[0][0] = Complex.ONE;
            matrix[1][3] = Complex.ONE;
            matrix[2][2] = Complex.ONE;
            matrix[3][1] = Complex.ONE;
            return new Oracle(matrix);
        case 2:
            matrix[0][2] = Complex.ONE;
            matrix[1][1] = Complex.ONE;
            matrix[2][0] = Complex.ONE;
            matrix[3][3] = Complex.ONE;
            return new Oracle(matrix);
        case 3:
            matrix[0][2] = Complex.ONE;
            matrix[1][3] = Complex.ONE;
            matrix[2][0] = Complex.ONE;
            matrix[3][1] = Complex.ONE;
            return new Oracle(matrix);
        default:
            throw new IllegalArgumentException("Wrong
                index in oracle");
    }
}

```

①

②

③

④

⑤

⑥

⑦

- ➊ When this function is called, an integer needs to be provided indicating what oracle type to return.
- ➋ In all cases, the result is a  $4 \times 4$  matrix of complex numbers.
- ➌ If the caller provides 0, an oracle with the matrix corresponding to  $f_1$  will be returned.
- ➍ If the caller provides 1, an oracle with the matrix corresponding to  $f_2$  will be returned.
- ➎ If the caller provides 2, an oracle with the matrix corresponding to  $f_3$  will be returned.
- ➏ If the caller provides 3, an oracle with the matrix corresponding to  $f_4$  will be returned.
- ➐ If we reach here, the caller provided a wrong value, and we throw an exception.

***Now that we have the code that returns an oracle corresponding to a value we provide, we can focus on the algorithm that should detect whether the oracle is linked with a constant function or a balanced function.***

We start with the naive approach where we just apply the oracle to two qubits that are initially 0. We hope the result will tell us with 100% confidence whether the underlying function is balanced.

## Applying the oracle

```

static void try00() {
    QuantumExecutionEnvironment simulator =
        new SimpleQuantumExecutionEnvironment();
    Program program = null;
    for (int choice = 0; choice < 4; choice++) {
        program = new Program(2);

        Step oracleStep = new Step();
        Oracle oracle = createOracle(choice);
        oracleStep.addGate(oracle);
        program.addStep(oracleStep);

        Result result = simulator.runProgram(program);
        Qubit[] qubits = result.getQubits();

        boolean constant =
            (choice == 0) || (choice == 3);

        System.err.println((constant ? "C" : "B") +
            ", measured = |" + qubits[1].measure() +
            " , " + qubits[0].measure()+">");
    }
}

```

- ➊ This function is called `try00` as it applies the oracles to two qubits that are in their initial state of  $|0\rangle$ .
  - ➋ Iterates over the four possible oracle types
  - ➌ Creates a quantum program that contains two qubits
  - ➍ Creates the oracle corresponding to the loop index “choice” and adds it to the program
  - ➎ Executes the program and obtains the results
  - ➏ Based on the loop index “choice,” we know if the oracle corresponds to a balanced or constant function. We print that information together with the measurements of the two qubits.
- The results of this application are as follows:
- C, measured =  $|00\rangle$
  - B, measured =  $|00\rangle$
  - B, measured =  $|10\rangle$
  - C, measured =  $|10\rangle$

Note that since we didn’t use superposition, these results are always the same.

Let’s investigate these results. There are two possible outcomes we can measure: the result is either  $|00\rangle$  or  $|10\rangle$ . Unfortunately, a single result doesn’t tell us whether the function was constant (as indicated by the C) or balanced (as indicated by the B). For example, if we measure  $|00\rangle$ , the function is either  $f_1$  or  $f_2$ ; but since the first is constant and the second is balanced, we don’t have an answer to our question.

We can be clever and try to run the application again, but this time we first flip one of the qubits, or both, to the  $|1\rangle$  state using a Pauli-X gate. This is shown in the code in the same file.

Doing so, we run four versions of our simple program, each corresponding to a different initial state of the two qubits (methods: try10, try11).

The results are shown here:

### Use $|00\rangle$ as input

```
C, measured = |00>
B, measured = |00>
B, measured = |10>
C, measured = |10>
```

### Use $|01\rangle$ as input

```
C, measured = |01>
B, measured = |11>
B, measured = |01>
C, measured = |11>
```

### Use $|10\rangle$ as input

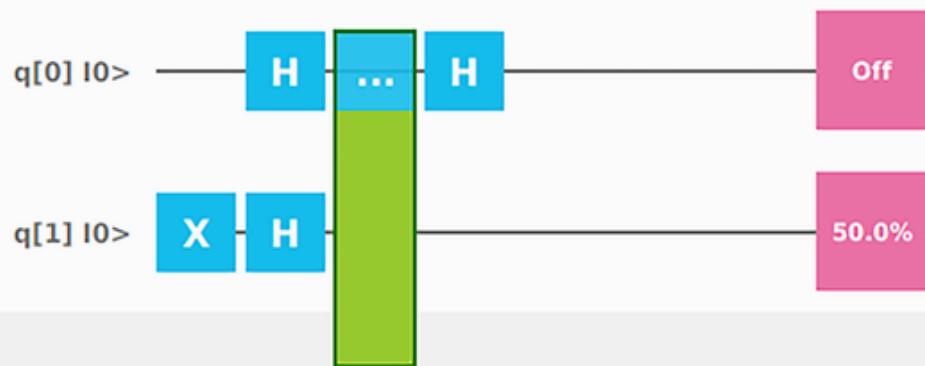
```
C, measured = |10>
B, measured = |10>
B, measured = |00>
C, measured = |00>
```

### Use $|11\rangle$ as input

```
C, measured = |11>
B, measured = |01>
B, measured = |11>
C, measured = |01>
```

If you analyze this result, you will conclude that **none of those versions is sufficient** to detect whether the **oracle** corresponds to a constant or a balanced function by doing **a single evaluation**.

**But so far, we didn't use the powerful superposition. We will do that now.**



We start with two qubits. The first qubit,  $q[0]$ , will be evaluated. But instead of evaluating twice, the first time in the state  $|0\rangle$  and the second time in the state  $|1\rangle$ , we apply a Hadamard transform to it to bring it into a superposition. You can think of this as allowing us to evaluate both possible values in a single quantum step.  
 The second qubit,  $q[1]$ , which is initially  $|0\rangle$  as well, is first flipped into  $|1\rangle$  by applying a Pauli-X gate. Next, a Hadamard gate is applied to this qubit. Both qubits are then used as input to the oracle we discussed in the previous section. After the oracle has been applied, we discard the second qubit. On the first qubit, a Hadamard gate is applied, and the qubit is measured.

*Now we arrive at the great thing about this algorithm. The following statement can be mathematically proven:  
 If the measurement is 0, we are guaranteed that the function represented by the oracle is balanced.  
 If the measurement is 1, we know the considered function represented by the oracle is constant.*

If you are interested in the mathematics behind this, it can be proven mathematically that the probability of measuring 0 for the first qubit after the circuit is applied is given by

$$\left( \frac{1}{2} \left( (-1)^{f(0)} + (-1)^{f(1)} \right) \right)^2$$

If  $f$  is a constant function, this will always result in 1. If  $f$  is a balanced function, this will always result in 0.

Instead of proving this, we will create code that runs the circuit for the four different functions and measure the output to see if our statement is correct. The interesting part of this algorithm is that we managed to make the first qubits dependent on the evaluation of all values. We don't have the measurements for all these evaluations, but that was not the original question.

The original goal was to determine whether a given function is constant or balanced.

```

QuantumExecutionEnvironment simulator =
    new SimpleQuantumExecutionEnvironment();
Random random = new Random();
Program program = null;
for (int i = 0; i < 10; i++) {
    program = new Program(2);
    Step step0 = new Step();
    step0.addGate(new X(1));

    Step step1 = new Step();
    step1.addGate(new Hadamard(0));
    step1.addGate(new Hadamard(1));

    Step step2 = new Step();
    int choice = random.nextInt(4);
    Oracle oracle = createOracle(choice);
    step2.addGate(oracle);

    Step step3 = new Step();
    step3.addGate(new Hadamard(0));

    program.addStep(step0);
    program.addStep(step1);
    program.addStep(step2);
    program.addStep(step3);
    Result result = simulator.runProgram(program);
    Qubit[] qubits = result.getQubits();
    System.out.println("f = " + (choice+1) +
        ", val = " + qubits[0].measure());
}

```

- ➊ The loop will be executed 10 times, each time with a random oracle.
  - ➋ Creates a program with two qubits
  - ➌ The first step applies a Pauli-X gate to the second qubit.
  - ➍ The second step applies Hadamard gates to both qubits.
  - ➎ Chooses a random oracle (from a predefined list)
  - ➏ Adds the oracle to the quantum circuit
  - ➐ Applies another Hadamard gate to the first qubit
  - ➑ Adds the steps to the quantum program
  - ➒ Executes the quantum program
  - ➓ The first qubit is measured, and, based on its value, we know whether the oracle corresponded with a constant or balanced function.
- If you run this application, you will see the circuit, and the console will show output similar to the following:
- |  |  |
|--|--|
| <b>➎</b><br><b>f = 3, val = 1</b><br><b>➏</b><br><b>f = 3, val = 1</b><br><b>f = 3, val = 1</b><br><b>➐</b><br><b>f = 1, val = 0</b><br><b>f = 4, val = 0</b><br><b>➑</b><br><b>f = 4, val = 0</b><br><b>f = 2, val = 1</b><br><b>f = 1, val = 0</b><br><b>f = 2, val = 1</b><br><b>➒</b><br><b>f = 4, val = 0</b> |  |
|--|--|

For every line in this output, the type of function is printed (f1, f2, f3, or f4) followed by the measured value of the first qubit. As you can see, this value is always 1 for f2 and f3 (the balanced functions), and it is 0 for f1 and f4 (the constant functions).

## Quantum Algorithms & Coding – Deutsch-Jozsa algorithm

*The Deutsch algorithm shows that a specific problem that requires two evaluations in a classical approach can be solved by a single evaluation using a quantum algorithm. While this may sound a bit disappointing, the principle is very promising.*

The Deutsch algorithm can easily be extended to the Deutsch-Jozsa algorithm, in which the input function is operating not on a single Boolean value but on  $n$  Boolean values. In this case, the function can be represented as:

$$f(x_0, x_1, \dots, x_{n-1})$$

which indicates that the function uses as input  $n$  bits that are either 0 or 1.

We are given such a function, and we are told that again the function is either constant (which means it always returns 0 or it always returns 1) or balanced (in half the cases it returns 0, and in the other half it returns 1).

The Deutsch algorithm is a special case of this situation, where  $n = 1$ . In that case, there are only two possible input scenarios.

If  $n = 2$ , there are four possible input scenarios. In general, there are  $2^n$  scenarios when there are  $n$  input bits.

How many classical evaluations do we need to do before we are 100% certain that the function is either constant or balanced? Suppose that we evaluate half of the possible scenarios ( $2^n/2$ , which is  $2^{n-1}$ ). If at least one of the results is 0 and at least one of the results is 1, we know that the function is not constant, so it must be balanced. But what can we conclude if all evaluations result in the value 1? In that case, it looks like the function is constant. But we still need one additional evaluation, as there is a probability that all the other evaluations will result in 0. To be 100% certain, a function with  $n$  bits as input requires  $2^{n-1} + 1$  evaluations before we can conclude that the function is either balanced or constant.

*However, using a quantum circuit similar to the one in the Deutsch algorithm, only a single evaluation is required. The importance of this is that it shows that quantum algorithms are great for problems that require exponential complexity using a classical approach.*

*The Deutsch-Jozsa algorithm is very similar to the Deutsch algorithm.*

```

static final int N = 3;
...
QuantumExecutionEnvironment simulator =
    new SimpleQuantumExecutionEnvironment();
Random random = new Random();
Program program = null;
for (int i = 0; i < 10; i++) {
    program = new Program(N+1);
    Step step0 = new Step();
    step0.addGate(new X(N));

    Step step1 = new Step();
    for (int j = 0; j < N+1; j++) {
        step1.addGate(new Hadamard(j));
    }

    Step step2 = new Step();
    int choice = random.nextInt(2);
    Oracle oracle = createOracle(choice);
    step2.addGate(oracle);

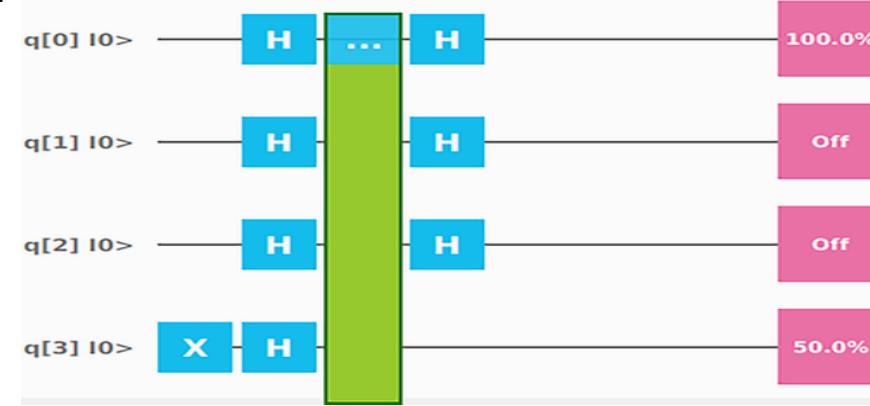
    Step step3 = new Step();
    for (int j = 0; j < N; j++) {
        step3.addGate(new Hadamard(j));
    }

    program.addStep(step0);
    program.addStep(step1);
    program.addStep(step2);
    program.addStep(step3);
    Result result = simulator.runProgram(program);
    Qubit[] qubits = result.getQubits();
    System.out.println("f = " + choice + ", val = "
        + qubits[0].measure());
}

```

## Quantum Algorithms & Coding – Deutsch-Jozsa algorithm

- ➊ ① Defines how many input bits we use (in this case, three)
- ➋ ② Creates a program with  $N + 1$  qubits. We need  $N$  qubits for the input bits and an additional ancilla qubit.
- ➌ ③ Applies a Pauli-X gate to the ancilla qubit
- ➍ ④ Applies a Hadamard gate to all qubits, bringing them into superposition
- ➎ ⑤ Adds a random oracle to the circuit
- ➏ ⑥ Applies a Hadamard gate to all input qubits (not to the ancilla qubit)
- ➐ ⑦ Executes the program and measures the result of the first qubit



Prove that the Deutsch-Jozsa algorithm is exactly the same as the Deutsch algorithm if  $N = 1$ .

- ➑ ⑥ Similar to the Deutsch algorithm, when  $f(x)$  is a constant function, this equation shows that the probability of measuring 0 on the first qubit is 100%. When  $f(x)$  is a balanced function, the first qubit will always be measured as 1 (as the probability of measuring 0 is null). The code in the example randomly picks one of two predefined oracles. The first oracle corresponds to the Identity gate, and that corresponds to a constant function that always returns 0. The second oracle corresponds to a CNot gate, where the ancilla qubit is swapped when the last input qubit is 1. Again, our goal is not to create those oracles. You can assume that the oracles are somehow provided to you, and you have to find out if they correspond to either constant or balanced functions.

## Quantum Algorithms & Coding – Deutsch-Jozsa algorithm

### Conclusion

In this section, you created the **Deutsch-Jozsa algorithm**. *While there is no direct practical usage of this algorithm, you achieved a major milestone. For the first time, you created a quantum algorithm that can execute a task much faster than a corresponding classical algorithm. The real speedup can only be seen if you use a real quantum computer; but the algorithms you created clearly show that a single evaluation is required to fix a specific problem, whereas in classical computing an exponential number of evaluations is required.*

Two very important but challenging parts of quantum computing are:

- Coming up with quantum algorithms like this that are proven to be faster than corresponding classical algorithms
  - Finding practical use cases for such algorithms
- \* e.g. Two algorithms that satisfy these requirements are search with Grover and prime factorization with Shor

### Summary

- Some problems can be solved without calculating the end results.
- There is a difference between function evaluation and function properties.
- Classical algorithms can distinguish between balanced and constant functions, but they require many function evaluations to do so.
- A quantum oracle is a black box related to a classical black box function.
- Using the **Deutsch-Jozsa algorithm**, you can detect whether a supplied oracle

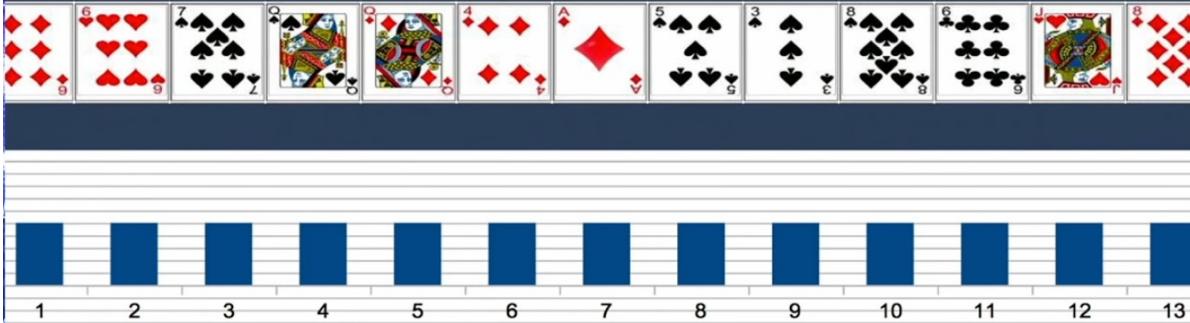
# Quantum Algorithms & Coding – Grover Searching Algorithm

Where is “ACE of Diamonds” in the deck?



<https://sgp.fas.org/othergov/doe/lanl/pubs/00783367.pdf> | <https://www.youtube.com/watch?v=8p6P-QTQM2M> | <https://www.youtube.com/watch?v=fFW36-CGkPY>

## STEP 1: ENCODE SUPERPOSITION



1. In Quantum, entangle 4 particles for 13 cards (e.g. ions of Magnesium/Calcium at -273 Celsius – 3 microkelvins and within electric field of 8/27MHz frequency) for 13 cards and the combined states are random equal.

## STEP 2: ORACLE FUNCTION

$f(w) =$  OTHER CARDS: 0  
ACE OF DIAMONDS: 1

2. Define an ORACLE function, and apply this to all the qubits at once.

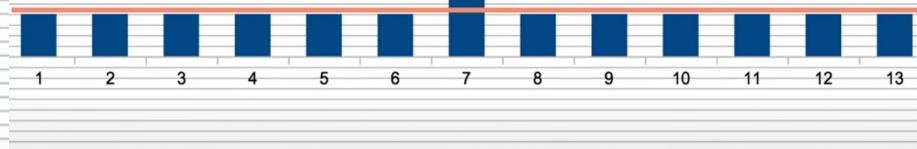
# Quantum Algorithms & Coding – Grover Searching Algorithm

## STEP 2: ORACLE FUNCTION



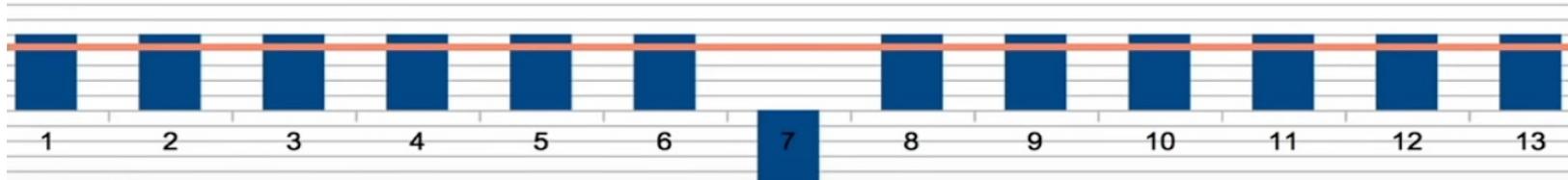
2. In quantum probability of measuring Ace of Diamonds is the same but is flipped

## STEP 3: AMPLIFY THE RESULT



3.2 In quantum Grover flip around the average axis

## STEP 3: AMPLIFY THE RESULT



3.1 In quantum Grover take the average probability of everything

Copyright: <https://www.manning.com/books/quantum-computing-in-action>

## STEP 4: REPEAT 2 AND 3



4. In quantum Grover repeat step 2 and 3 couple of times  
=>  $O(\sqrt{N})$

Github: <https://github.com/critoma/quantum/tree/main/quantumjava/ch10>

## What is Grover's search algorithm?

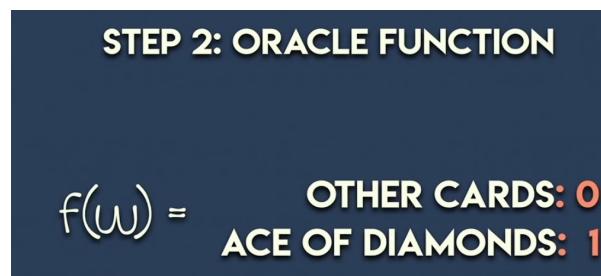
## Quantum Algorithms & Coding – Grover Searching Algorithm

Suppose we are given a black box that requires an integer number as input and returns an output. The output is always 0, except for one specific input value (often noted as  $w$ ), in which case the output is 1. Grover's search algorithm lets us retrieve this specific input value in a performant way.

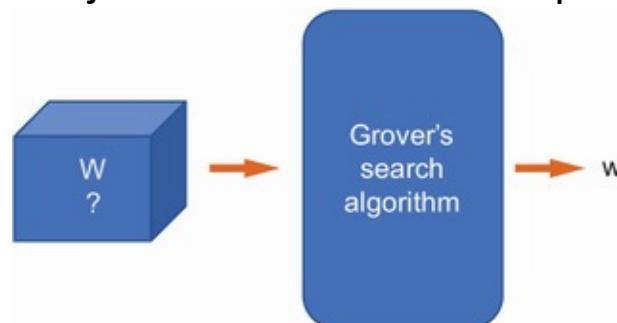
The concept of a black box is shown in figure A. Somehow, the black box checks whether the provided input equals  $w$ . If so, the output is 1. Otherwise, the output is 0. It is important to realize that we don't know how the black box works internally. It might contain a very simple or a very complex algorithm. We have to assume someone created the black box and handed it over to us—we were not involved in how it was created. Indeed, if we were the creator of the black box, there would be no point in writing algorithms to retrieve the value of  $w$ , as we would have used that value to create the black box.



A. Black box returning either 0 or 1 based on an integer number



Somehow, the black box contains information about the value  $w$ , and by querying it in a smart way, Grover's algorithm can retrieve that value. The input for Grover's search algorithm is not a number, a search query, or an SQL string—but it is the black box that we just discussed. This is explained in figure B



B. Grover's search algorithm takes a black box as input and returns the value  $w$  that causes the black box to evaluate to 1.

## Quantum Algorithms & Coding – Grover Searching Algorithm

Most enterprise IT cases use multiple databases with tables and rows. For demo purposes, we will create a simple data storage containing people with their age and country

Search applications can use this data to provide answers to questions like these:

Who is 36 years old and lives in Australia?

How many of the people live in Ukraine?

Is there someone named “Joe” who lives in Greece?

Give me the names of all people older than 34 years.

We will create an application that answers one specific question:

***“Find the person who is 29 years old and lives in Mexico.” => “Albert”***

	age	country
Alice	42	USA
Bob	36	UK
Eve	85	Australia
Nikos	18	Greece
Albert	29	Mexico
Alexandra	29	Romania
Marie	15	Ukraine
Janice	52	Israel

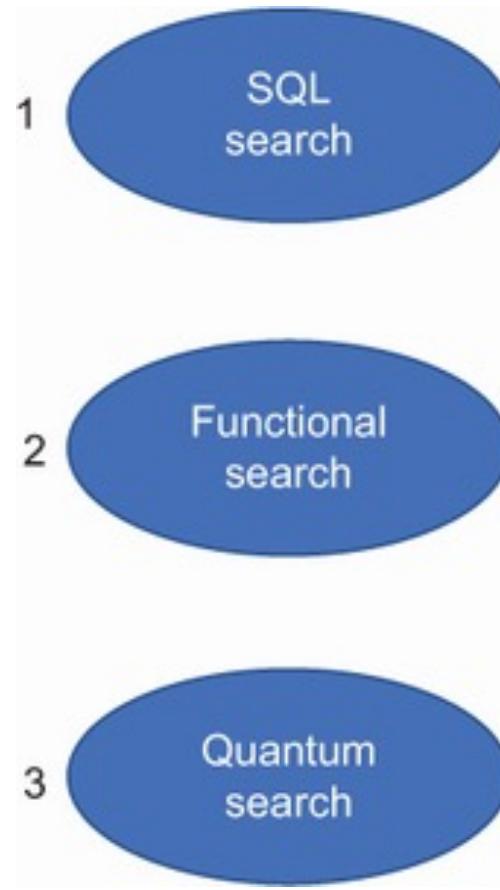
**NOTE:** The SQL query to match this question would be something like

**SELECT \* FROM PERSON WHERE PERSON.AGE=29 AND PERSON.COUNTRY='MEXICO'**

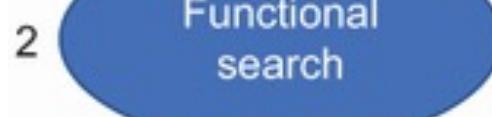
We will work with Java code instead of SQL queries, as this approach allows us to gradually introduce the ideas behind Grover’s search algorithm. This is more closed to reality than searching the Ace of diamonds in a shuffled deck of cards.

## Quantum Algorithms & Coding – Grover Searching Algorithm

We first address our question using the classical approach. Next, we reformulate the question so we can deal with it using a more functional approach that comes closer to Grover's algorithm. Finally, we use Grover's algorithm to implement the functionality for this search. This process corresponds to the mental model:



<https://github.com/critoma/quantum/blob/main/quantumjava/ch10/classicsearch/>



<https://github.com/critoma/quantum/blob/main/quantumjava/ch10/classicsearch/>



<https://github.com/critoma/quantum/tree/main/quantumjava/ch10/quantumsearch>

<https://github.com/critoma/quantum/blob/main/quantumjava/ch10/grover>

<https://github.com/critoma/quantum/blob/main/quantumjava/ch10/stepbystepgrover>

<https://github.com/critoma/quantum/blob/main/quantumjava/ch10/groveroracle>

From SQL search to quantum search

## “SQL Search” – standard data structures search

```
Person findPersonByAgeAndCountry(List<Person> persons,
                                 int age, String country) {

    boolean found = false;

    int idx = 0;

    while (!found && (idx<persons.size())) {

        Person target = persons.get(idx++);

        if ((target.getAge() == age) &&
            (target.getCountry().equals(country))) {
            found = true;
        }
    }

    System.out.println("Got result in "+idx+" tries");

    return persons.get(idx-1);
}
```

```
void complexSearch() {
    for (int i = 0; i < 10; i++) {
        List<Person> persons = prepareDatabase();
        Collections.shuffle(persons);
        Person target = findPersonByAgeAndCountry(persons, 29, "Mexico");
        System.out.println("Result of complex search= " + target.getName());
    }
}
```

## Quantum Algorithms & Coding – Grover Searching Algorithm

- 1 Boolean variable that indicates if we already got the result
  - 2 Index that tells us the position of the element we are investigating
  - 3 As long as we don't have a result and the index is still lower than the total number of elements, we execute the following loop.
  - 4 Obtains the element under consideration from the list
  - 5 Checks the properties (age and country) of that element
  - 6 If the properties match, we flip the boolean variable to true, so the loop is not executed needlessly.
  - 7 Prints the number of evaluations
  - 8 Returns the result to the caller
- 6 The main method of the `classicsearch` application runs the search function 10 times, and each time the number of required evaluations is printed. The function that invokes the `findPersonByAgeAndCountry` method.

7 Note that before we do a find operation, we shuffle the list of people, so the result is truly random. **Running the application therefore results in output similar to this:**

```
Got result after 8 tries
Result of complex search = Albert
Got result after 1 tries
Result of complex search = Albert
Got result after 2 tries
Result of complex search = Albert
Got result after 3 tries
Result of complex search = Albert
Got result after 5 tries
Result of complex search = Albert
Got result after 7 tries
Result of complex search = Albert
Got result after 2 tries
Result of complex search = Albert
Got result after 1 tries
Result of complex search = Albert
Got result after 2 tries
Result of complex search = Albert
Got result after 5 tries
Result of complex search = Albert
```

The example code from the previous section is very flexible: we can easily modify the search criteria by providing a different age or a different country to the `findPersonByAgeAndCountry` method. *Unfortunately, this is not how Grover's search algorithm works. With Grover's algorithm, we don't provide search parameters, but we have to provide a single function that evaluates to 1 for exactly one input case and evaluates to 0 in all other cases.*

In this section, we often use the variable `w` to indicate the input case that results in a function evaluation of 1—in other words, `w` is the value we are looking for. We can define this as follows:

$$\begin{aligned} f(w) &= 1 \\ f(x) &= 0, x \neq w \end{aligned}$$

In this section, *we modify the previous classical example so that it is using a functional approach, which can then be mapped conceptually to the quantum algorithm in the next section.*

We perform the same search query, but instead of checking the entries by examining their properties one by one, we apply a function to each entry. When the function evaluates to 1, we know we have the correct entry.

We use the **Java Function API** to achieve this. Since the functions will always evaluate to an integer (0 or 1), we use the `ToIntFunction`. First we need to create the function.

Now that we have this function, we can write some Java code that iterates over the list of people and applies the function over each entry until the function returns 1—which means the correct answer has been found. The code is shown next in the right side ...

**1** *Instead of checking the properties of the target, as we did in the previous example, we apply our function to it. When the function returns 1, we know the target is the correct result.*

*While the approach is different (via functional programming), the amount of time required is similar to the previous algorithm. We still iterate over every person in the list and check whether the age and country of the considered person satisfy our criteria.*

### Creating the function

```
ToIntFunction<Person> f29Mexico ①
    = (Person p) -> ②
        ((p.getAge() == 29)
        && (p.getCountry().equals("Mexico"))) ? 1 : 0; ③
```

- ① Creates a `ToIntFunction` that takes a `Person` as input and returns an integer as output
- ② When the function is applied, the parameter `p` contains the supplied person.
- ③ If the age of the supplied person is 29 and the country is Mexico, the function returns 1. In all other cases, the function returns 0.

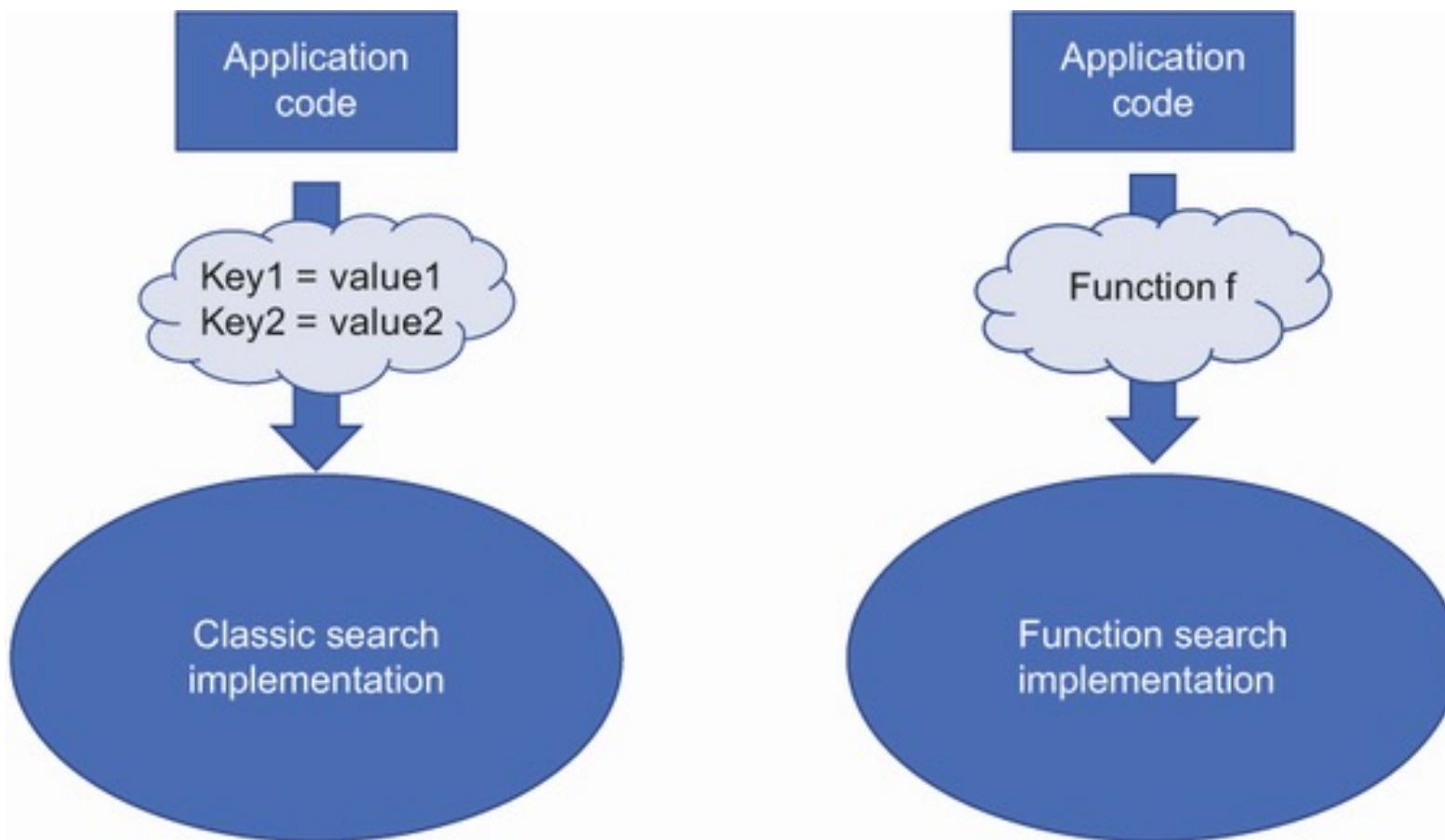
**NOTE** This is a fixed function for a particular problem. If we want to retrieve a person whose age is 36, we need to create a new function.

```
Person findPersonByFunction(List<Person> persons,
                           ToIntFunction<Person> function) {
    boolean found = false;
    int idx = 0;
    while (!found && (idx < persons.size())) {
        Person target = persons.get(idx++);
        if (function.applyAsInt(target) == 1) {
            found = true;
        }
    }
    System.out.println("Got result in " + idx + " tries");
    return persons.get(idx - 1);
}
```

## Functional programming search – Java Lambda

## Quantum Algorithms & Coding – Grover Searching Algorithm

The second approach (with **Functional Programming**) is closer to the quantum approach we discuss in the following section. Instead of providing several parameters, we provide a function. **The search algorithm does not have to *create* that function, but it has to *evaluate* it.**



Searching using a classic approach versus using functions

Note that the function search implementation, as we call it in the figure, is still using classical code. However, it brings us closer to how Grover's search algorithm works on a quantum computer.

## Quantum Algorithms & Coding – Grover Searching Algorithm

Quantum Search: Using Grover's search algorithm: <https://github.com/critoma/quantum/blob/main/quantumjava/ch10/quantumsearch>

**Grover's algorithm** has some similarities with the function-based search discussed in the previous section. When a black box (or a **quantum oracle**) is provided that is linked to a function like the one described earlier, Grover's algorithm can return the unique input **w** that would result in a function evaluation of 1 in about  $\sqrt{N}$  steps, **where each step requires a single oracle evaluation.**

**NOTE:** In other words, while the classical search algorithm requires on average  $n/2$  function evaluations, Grover's algorithm achieves the same goal in  $\sqrt{N}$  function evaluations.

*For small lists, this is not impressive. A list with eight elements requires on average four function evaluations for the classical case and about three evaluations for the quantum case. However, for large lists, the advantage becomes clear. A list with 1 million elements requires on average 500,000 evaluations, and in the worst case, it might require 1 million classical evaluations. The same result can be obtained with only 1,000 quantum evaluations.*

**NOTE:** Grover's search algorithm requires a predefined number of function evaluations, and we show later that this is number is  $\sqrt{N}$ . Contrary to the classical approach, there is no best case of one evaluation or worst case of, for example,  $n$  evaluations. We can only do a single measurement for a quantum application. If we tried to peek before the optimal number of steps was performed, we would have no way to resume the quantum calculation when it turned out that the number we measured was wrong. Therefore, the required number of evaluations is fixed.

## Quantum Algorithms & Coding – Grover Searching Algorithm

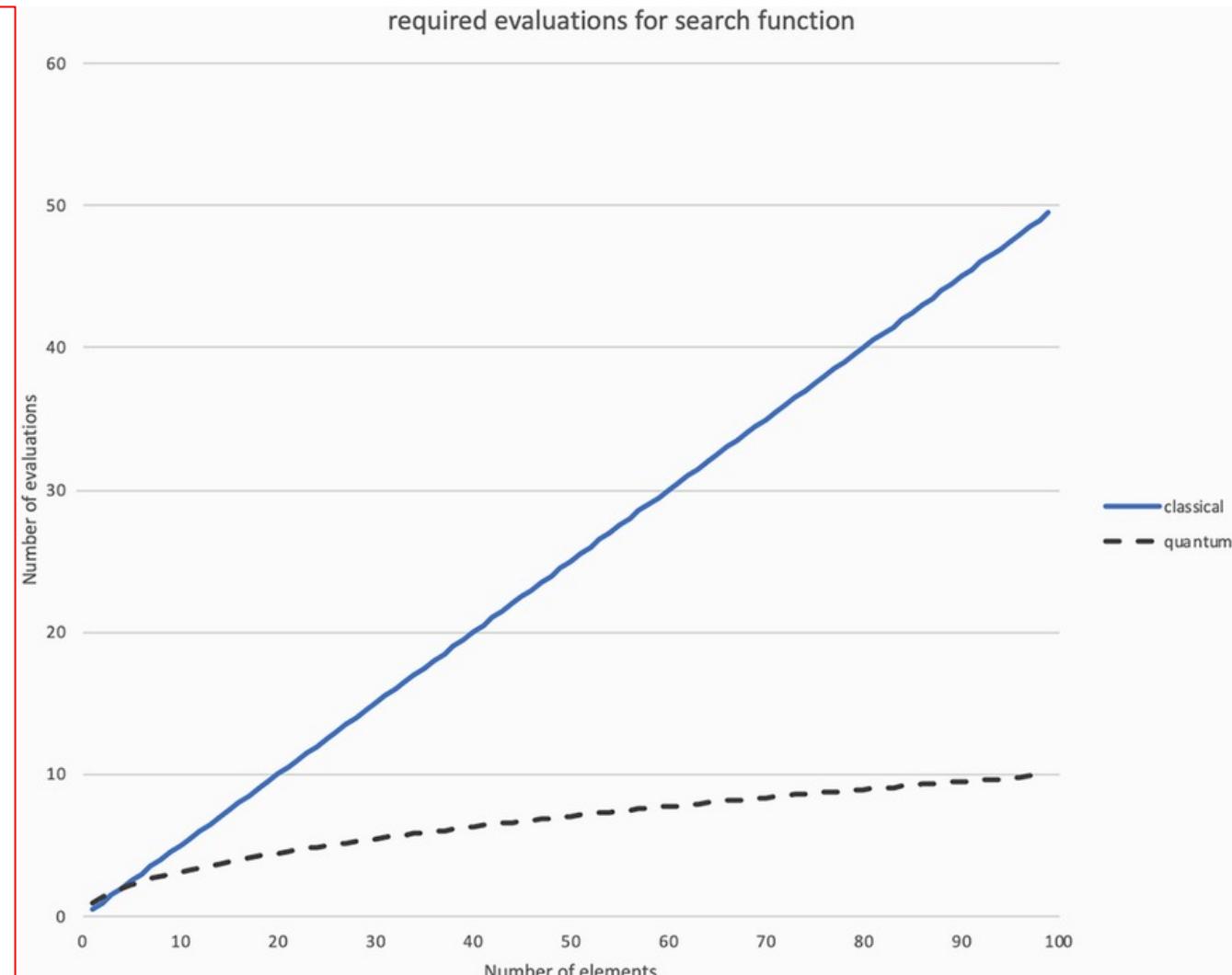
Quantum Search: Using Grover's search algorithm: <https://github.com/critoma/quantum/blob/main/quantumjava/ch10/quantumsearch>

The difference between the required evaluations for a classical search versus quantum search using **Grover's algorithm** is visualized in the next figure. We show only the difference between the classical search and Grover's search algorithm for lists of up to 100 elements. As you can see, the larger the size of the list, the more remarkable the differences become. So, it is clear that Grover's algorithm is particularly useful for lists with a huge number of elements.

It is often said that Grover's algorithm provides a **quadratic speedup** compared to classical search algorithms. This is true, since for a given number of evaluations (e.g.,  $N$ ), Grover's algorithm can deal with a list of  $N^2$  elements, while a classical algorithm can only deal with lists of  $N$  elements.

***For developers, it is often more important to realize when an algorithm is applicable rather than how it works.***

Therefore, we first explain how to use the built-in Grover functionality in Strange, which hides the underlying implementation.



**Quadratic speedup:** Required number of function evaluations as a function of the number of items

# Quantum Algorithms & Coding – Grover Searching Algorithm

Quantum Search: Using Grover's search algorithm: <https://github.com/critoma/quantum/blob/main/quantumjava/ch10/quantumsearch>

The **Java Strange quantum library** contains a classical method that under the hood uses Grover's search algorithm to implement a search operation. The signature of the classical method resembles the signature of the example we discussed earlier in this section:

**public static<T> T search(List<T> list, Function<T, Integer> function);**

This method takes two parameters as input:

- A list with elements of type T, where T can be a Person or any other Java class
- A function that takes as input an element of type T and returns either 1 (if the provided input is the one we are looking for) or 0 (in all other cases)

The following code snippet, from the example in *ch10/quantumsearch*

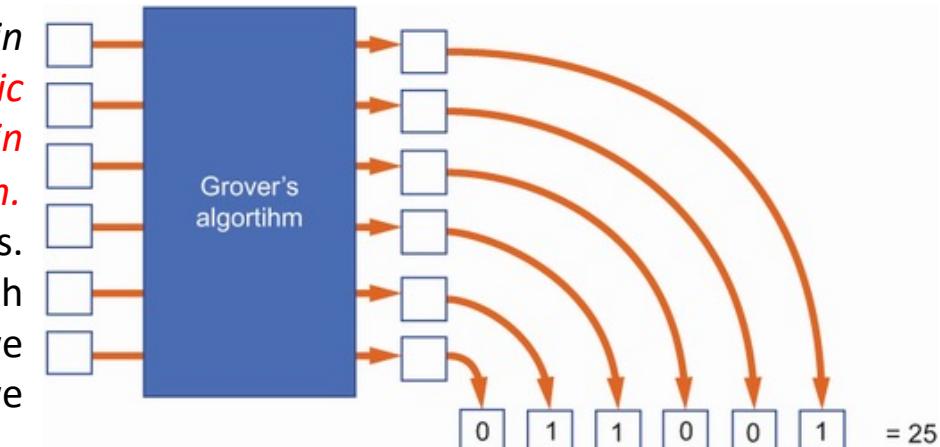
(<https://github.com/critoma/quantum/blob/main/quantumjava/ch10/quantumsearch>), shows how we can use this method:

```
void quantumSearch() {  
    Function<Person, Integer> f29Mexico  
        = (Person p) -> ((p.getAge() == 29) &&  
            (p.getCountry().equals("Mexico"))) ? 1 : 0;  
    List<Person> persons = prepareDatabase();  
    Collections.shuffle(persons);  
    Person target = Classic.search(persons, f29Mexico);  
    System.out.println("Result of function Search = "  
        + target.getName());  
}
```

- ① 1 Creates a function similar to the function in the previous example
  - ② 2 Creates the initial database again
  - ③ 3 Randomly shuffles the elements in the database
  - ④ 4 Calls the search method that is under the hood invoking Grover's search
  - ⑤ 5 Prints the result
- It is important to emphasize that the provided function is created outside the algorithm. In this case, the function is called **f29Mexico**, and the **Classic.search** method ("Classic" ... misleading, the search is QUANTUM - <https://github.com/critoma/quantum/blob/main/strange/src/main/java/org/redfx/strange/algorith/Classic.java> | <https://github.com/redfx/quantum/strange/blob/master/src/main/java/org/redfx/strange/algorith/Classic.java>) does not need to know anything about the internals of that function. The function is evaluated, but to the algorithm, this evaluation is a black box.

After applying a quantum circuit, we are left with a number of qubits that can be in different states. The probability vector describes how likely we are to measure a specific value. The goal of many quantum algorithms is to manipulate the probability vector in such a way that the measured outcome is likely to be very relevant to the original question.

In Grover's search algorithm, we need  $n$  qubits if we want to search in a list of  $2^n$  elements. For example, if our list has 128 elements, we need 7 qubits ( $2^7$ ). If we have to deal with 130 elements, we need 8 qubits, and so on. After applying Grover's algorithm, we hopefully obtain a set of qubits that, when measured, return the index of the element we are searching for in the list. This is explained in figure for a list with up to 64 elements.



High-level overview of what Grover's search algorithm achieves

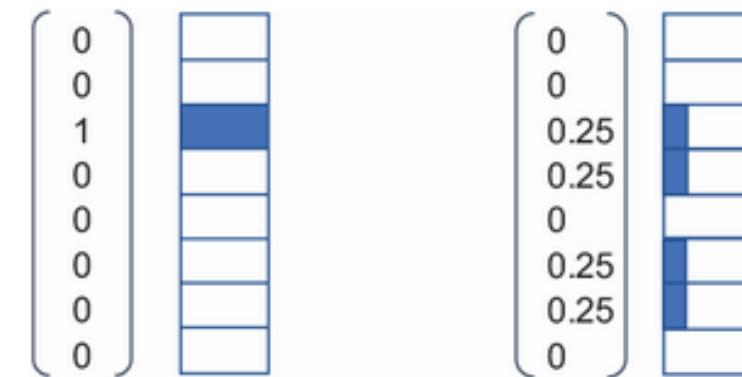
Suppose the element we are searching for has index 25. Initially, all qubits have the value 0. After applying Grover's algorithm, we measure the qubits, and we hope there is a very high probability of measuring the values 0, 1, 1, 0, 0, and 1, which is the binary representation of 25.

*With 6 qubits, there are  $2^6 = 64$  possible outcomes, so we have a probability vector with 64 elements. The goal of Grover's algorithm is to maximize the value of element 25 and minimize the value of all other elements. We will show that in most cases, the resulting probability for the correct value is very high, but not 100%.*

Grover's search algorithm requires several steps. We will show that with each step, the probability of finding the correct answer increases.

Probabilities can be indicated by numbers or by corresponding horizontal bars. A probability of 1 indicates that this specific outcome is guaranteed to be measured, and it corresponds with a filled horizontal bar. A probability of 0 means there is no chance that this specific outcome can be measured, and it corresponds with an empty horizontal bar. Numbers between 0 and 1 correspond with partially filled horizontal bars.

If we have a system with three qubits, we have eight possible outcomes. Figure with 8 possible outcomes associated to 3 qubits shows two different probability sets for this system, both with a vector with numbers and a notation with bars. Note that the sum of all probabilities should equal 1.



Probabilities in a vector and bar notation

The probabilities in the **probability vector** are real, non-negative numbers. So far, we have managed to hide the internal details of the relationship between a qubit and its probability vector. To understand what Grover's search algorithm is exploiting, it helps to shine a bit of light on what is below those probabilities. Again, we will not go deep into the mathematics or physics, but we will explain why it is important to understand that just the probability is not the entire story.

*The values in the state vector are actually complex numbers. A complex number  $c$  contains a real part and an imaginary part, such as  $c = a + ib$ , with  $i$  being the imaginary unit defined by  $i = \sqrt{-1}$ .*

In this equation, both  $a$  and  $b$  are real numbers. (For more information about complex numbers, see [https://en.wikipedia.org/wiki/Complex\\_number](https://en.wikipedia.org/wiki/Complex_number))

*The probability vector contains real, non-negative numbers only.*

**Q: How do we convert the values from the state vector to the probability vector?**

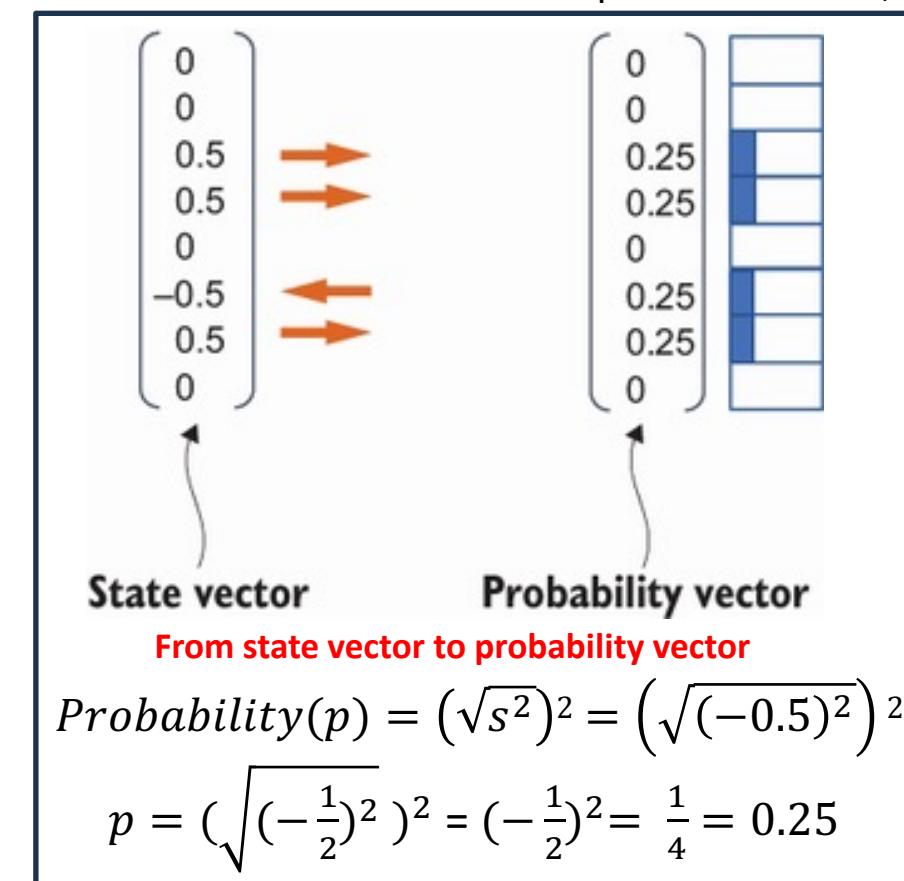
**A: This is done by taking the square of the modulus of the value in the state vector.**

The modulus of a complex number is defined as follows:

$$\text{mod}(s) = \sqrt{a^2 + b^2}$$

*The difference between a state vector and a probability vector is shown in figure for a simple case where the state vector contains real values only. There are four entries in the state vector with a value of 0, three entries with a value of 0.5, and one entry with the value -0.5. By taking the square of these values, we arrive at the probability vector in the same figure. By simply looking at the probability vector, we can no longer distinguish between the states with value 0.5 and the state with value -0.5.*

**NOTE:** The **probability vector** contains information about the likelihood of a particular outcome being measured. It is based on the **state vector**, but the latter contains even more information about **the internal state of the quantum system**. That internal state is important for a number of quantum algorithms.



### The **Classic.search** method

(<https://github.com/critoma/quantum/blob/main/strange/src/main/java/org/redfx/strange/algorith/Classic.java> | <https://github.com/redfx-quantum/strange/blob/master/src/main/java/org/redfx/strange/algorith/Classic.java>)

available in **Java Strange (FX)** lets us use Grover's search algorithm using classical computing only. One of its main benefits is that it allows developers to understand what type of problems could benefit from Grover's search.

*The internal workings of the quantum algorithm are less relevant to most developers, but here are some reasons a basic understanding is beneficial:*

- *Grover's search algorithm doesn't require a classical function as input; rather, it uses a quantum oracle that corresponds to a classical function. We discuss this oracle in the upcoming sections.*
- *Understanding how Grover's search algorithm uses quantum computing characteristics can help us create or understand other quantum algorithms.*

In the following sections, we explain the parts of Grover's search algorithm. The mathematical evidence is omitted, though.

### Running the example code

The code <https://github.com/critoma/quantum/tree/main/quantumjava/ch10/grover> allows you to run Grover's search algorithm step by step. We use this code to explain the algorithm. The implementation of the algorithm is done in a method called doGrover that has the following signature:

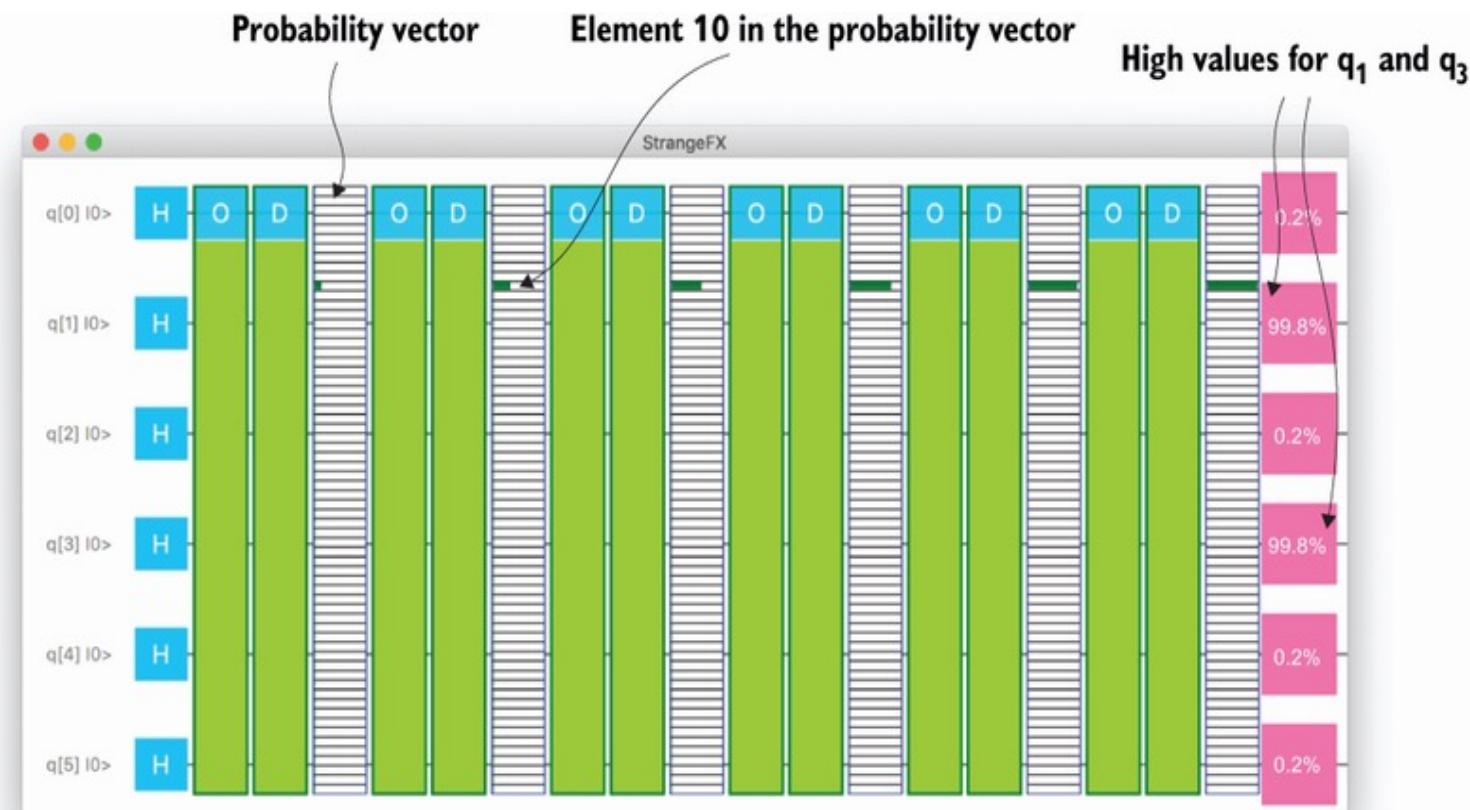
```
private static void doGrover(int dim, int solution)
```

*The first argument to this method, dim, specifies how many qubits should be involved. The second argument, solution, specifies the index of the element we are searching for. Again, note that in a real scenario, it doesn't make sense to provide the answer we are looking for (the solution). Someone should provide us with a black-box function. In this case, though, the code uses the solution to construct the black box.*

The main method of this example is very short:

```
public static void main (String[] args) {  
    doGrover(6,10);  
}
```

We simply call the doGrover method and specify that we have a system with 6 qubits (so we can accommodate a list with  $2^6 = 64$  elements) and the target element is at index 10. When we run the example, the quantum circuit is shown, demonstrating the steps of the algorithm, along with the probability vectors after each step.



Shortly, we explain the circuit and its visualization, but first we take a quick look at the end results. The right side of the figure shows the qubits after the algorithm has been applied. Qubits 1 and 3 (denoted as  $q[1]$  and  $q[3]$ ) have a very high probability of being measured as 1 (99.8%), while all other qubits have a very low probability of being measured as 1 (0.2%). As a result, there is a very high probability that the following sequence will be obtained when the qubits are measured:

**0 0 1 0 1 0**

Note that this sequence shows the highest-order qubit ( $q[5]$ ) on the left side and the lowest-order qubit ( $q[0]$ ) on the right side. Converting this binary sequence to a decimal number is done by adding powers of two when the corresponding qubit is 1:

**001010**

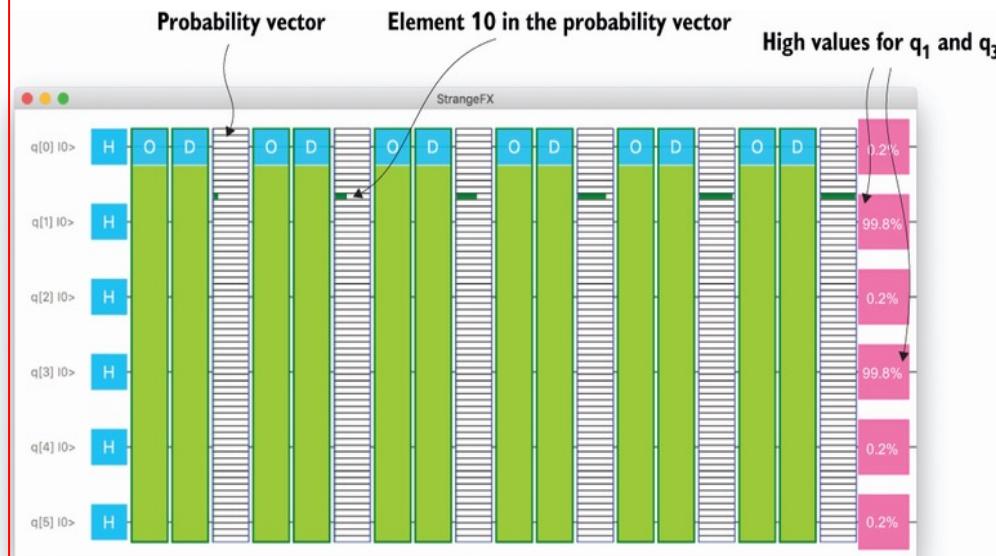
$$0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 8 + 2 = 10$$

This is the binary representation of the number 10. Hence, Grover's search algorithm resulted in returning the index of the element we are searching for.

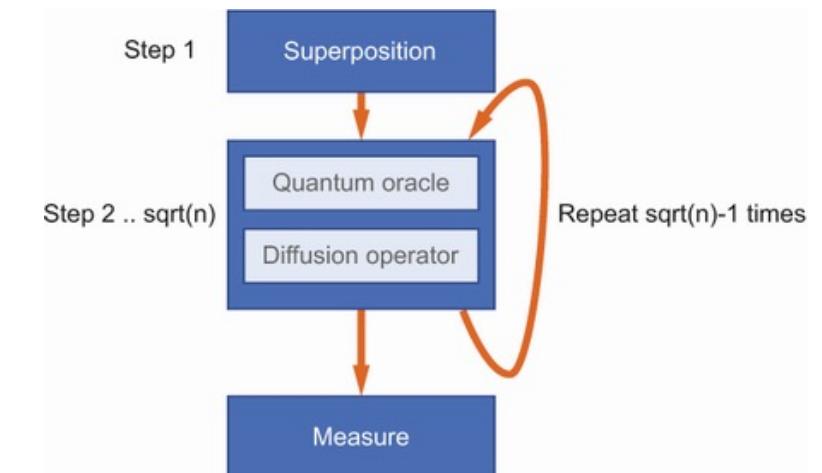
Each invocation of the step applies a quantum oracle denoted by an O and a diffusion operator denoted by a D. The quantum oracle is related to the function we have been given, and it will be explained later in this section, as will the diffusion operator.

After every invocation of this step, the probability vector is rendered. After the first step, which applies a Hadamard gate to each qubit, all options have the same probability. After the second step, all options have a rather low probability. The element at index 10 has a higher probability than the others, but it is still low. So, if we measure the system after this first step, there would be a fair chance of measuring the correct answer, 10, but an even bigger chance of measuring something else.

With each step, however, the probability of measuring 10 increases. At the last step, the probability of measuring 10 is 99.8%.



Running the example code



Flow of Grover's search algorithm

## The algorithm behind Grover's search

### Superposition

The example in [stepbystepgrover](#) is very similar to the example discussed earlier, but it deals with a list of only four items and thus can be handled by two qubits. This makes it easier to explain, and we use that example in the following subsections. Note that we also added more probability visualizations in this example. After every step, the probability vector is rendered; the result is shown in figure.

The first step in Grover's algorithm brings all qubits into a superposition state. This approach is often used in quantum algorithms, as it allows the processing to be applied to different cases simultaneously.

In our two-qubit example, the two qubits initially have the state  $|0\rangle$ . Hence, the initial state vector is described by

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The probability vector, which is obtained by each element taking the square of the corresponding element in the state vector, looks exactly the same. The square of 1 is 1, and the square of 0 is 0. The first element in this vector corresponds with the probability of measuring the two qubits in the  $|00\rangle$  state, which is exactly the initial state.

After applying a Hadamard gate to both qubits, the state vector becomes

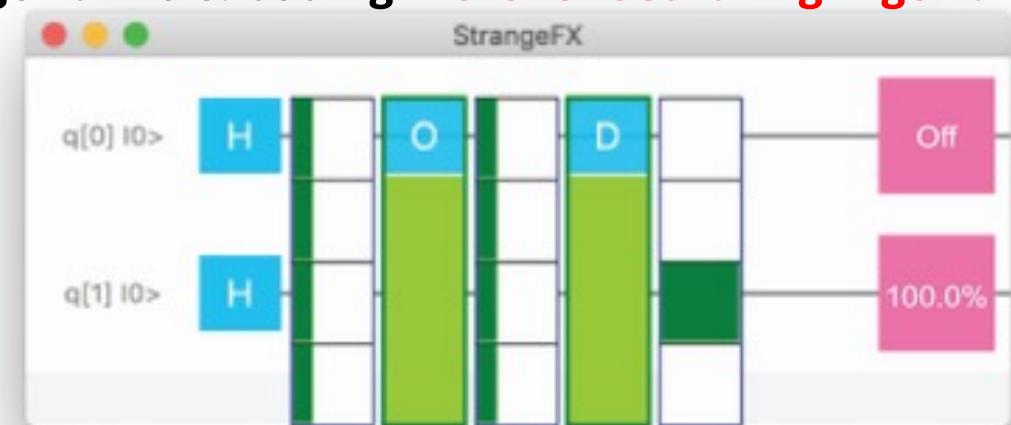
$$\begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$$

Each element in this vector has an amplitude of  $1/2$  or 0.5. In the corresponding probability vector, each element equals  $1/4$  or 0.25, which is the square of  $1/2$ . Therefore, the probability vector after applying the Hadamard gates is written as  $(0.25, 0.25, 0.25, 0.25)$ .

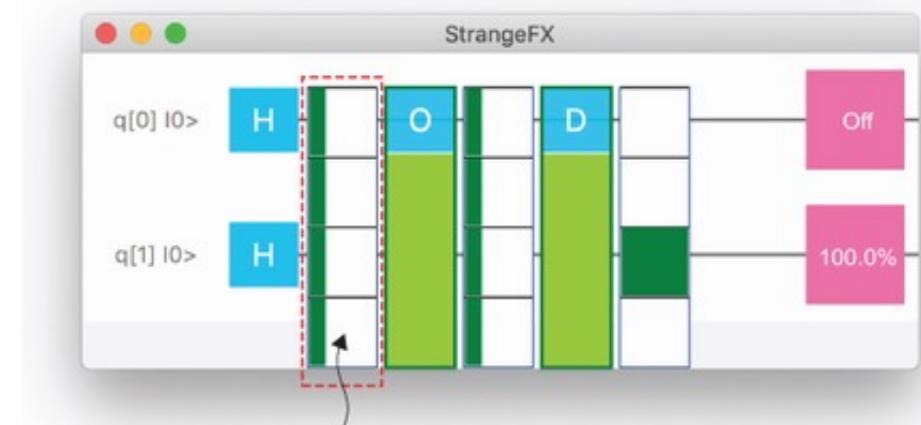
TIP: The state vector shows amplitudes, while the probability vector shows probabilities. For real numbers, the probability is the square of the amplitude. This is also shown in the probability infogate in the second figure.

$$\begin{bmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{bmatrix}$$

## Quantum Algorithms & Coding – Grover Searching Algorithm



Running the Grover step-by-step example with only two qubits



Probability vector after step 1

After step 1, all probabilities are equal.

## The algorithm behind Grover's search

### Quantum ORACLE

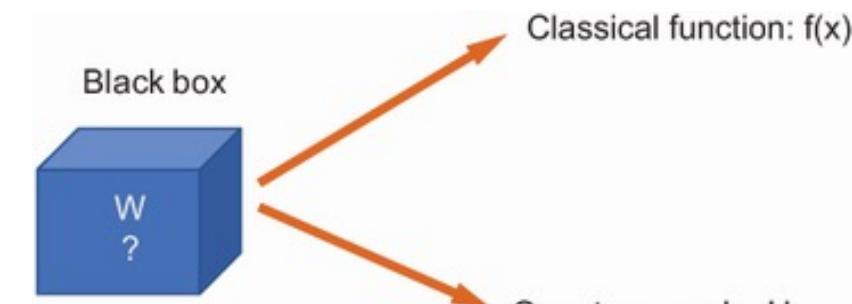
The main requirement for the classical variant of [Grover's search algorithm](#) is that we are given a function that for a single specific value returns 1 and for all other values returns 0. You learned that Grover's search algorithm considers this function a black box and does not have any knowledge about the internals of this function. However, that is a classical function. If we want to really use the quantum algorithm, we need a [quantum oracle](#) that is linked to this classical function.

Remember the [quantum oracle](#) from the [Deutsch algorithm](#)

This is very similar to what you learned in the previous section about the Deutsch algorithm. In the case of the [Deutsch algorithm](#), we were dealing with a function that was either constant or balanced. You created an oracle that operated on two qubits, where the first qubit was left intact and the second qubit was transferred via an operation that depended on the function evaluation.

Figure schematically shows the difference between the classical version of Grover's search algorithm and the quantum version: in the classical version, the black box is realized by a classical functional programming, while in the quantum version, the black box is realized by a quantum oracle. Obviously, there is a relationship between the classical function representing the black box and the quantum oracle representing the same black box.

## Quantum Algorithms & Coding – Grover Searching Algorithm



Black box in a classical versus quantum context

The quantum oracle related to a classical function  $f(x)$  does the following: for any value of  $|x\rangle$  that is not the specific value  $w$ ,  $f(x)$  is 0, so the original value  $|x\rangle$  is returned. If the value  $w$  is passed through the oracle, the result is  $-|x\rangle$ .

Let's give a concrete example. Suppose we have a list with four elements. In that case, we require two qubits (as  $2^2 = 4$ ). The element that we hope to find has index 2. Hence, the function we would pass to a classical algorithm is the following:

$$\begin{aligned}f(0) &= 0 & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\f(1) &= 0 \\f(2) &= 1 \\f(3) &= 0\end{aligned}$$

This corresponds to the oracle defined by the following matrix:

## Quantum ORACLE

The code to create this quantum oracle in the example is as follows:

```
static Oracle createOracle(int dim, int solution) {
    int N = 1<<dim;
    System.out.println("dim = "+dim+" hence N = "+N);
    Complex[][] matrix = new Complex[N][N];
    for (int i = 0; i < N;i++) {
        for (int j = 0 ; j < N; j++) {
            if (i != j) {
                matrix[i][j] = Complex.ZERO;
            } else {
                if (i == solution) {
                    matrix[i][j] = Complex.ONE.mul(-1);
                } else {
                    matrix[i][j] = Complex.ONE;
                }
            }
        }
    }
    Oracle answer = new Oracle(matrix);
    return answer;
}
```

- ➊ This method creates **the oracle (which is a special gate)** and it is invoked with a dimension parameter (the number of qubits) and the correct solution (the value that, when the classical function is applied, would return a result of 1).
- ➋ The oracle is represented by a matrix with dimension N: e.g., with 3 qubits, we need an  $2^3 \times 2^3$  aka.  $8 \times 8$  matrix.
- ➌ Loops over all rows (via the index i) and all columns (via the index j) of the matrix
- ➍ If the considered element is not a diagonal element, its value is 0.
- ➎ If the considered element is a diagonal element and its row (and thus also column) matches the correct solution, the matrix element should be -1.
- ➏ If the considered element is a diagonal element but its row (and thus also column) does not match the correct solution, the matrix element should be 1.
- ➐ Creates the oracle by passing the matrix to its constructor
- ➑ Returns the oracle

## Quantum ORACLE

When this function is applied to the example where we have two qubits ( $\text{dim} = 2$ ) and where the correct solution is 2 (solution = 2), you can verify that the  $4 \times 4$  matrix created by this function matches the matrix shown previously.

Before we apply this gate to the probability vector we obtained before, we apply it to a state vector containing a wrong value and to a state vector containing the correct value. Since the correct value is 2, the state vector representing 1 is a wrong state vector. As stated previously, applying the classical function  $f(1)$  results in 0; and due to the definition of the oracle, we expect that applying the oracle to  $|01\rangle$  will not change the input value. Let's double-check by doing the following matrix multiplication:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

As you can observe, the state did not change. Next, we apply this oracle to the state vector representing 2 since the classical function  $f(2)$  returns 1. The state vector 2 corresponds to the qubit sequence  $|10\rangle$ , so let's apply the quantum oracle to this vector:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}$$

Doing the matrix multiplication shows that the state vector is inverted.

**TIP:** From this simple example, you can see that while the state vector has changed, the probability vector is still the same as the original input. The square of 1 is equal to the square of  $-1$ , so we won't notice a difference when just looking at the probability vector.

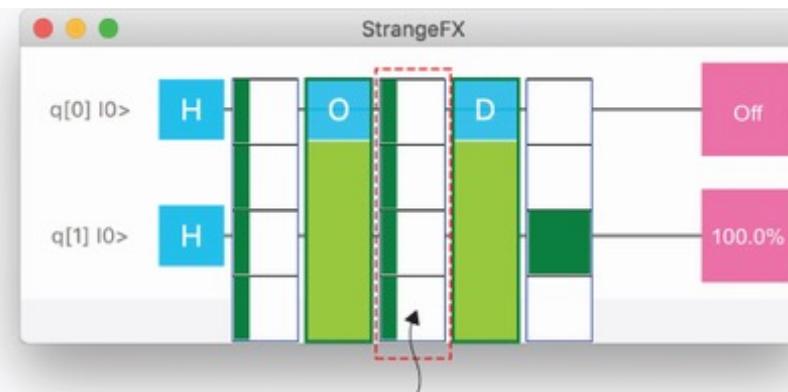
## Quantum ORACLE

In theory, we could apply this oracle to every single state vector representing a possible index. In all cases except one, the result would be identical to the input. When we applied the oracle to the correct value, the result would be inverted. However, this would mean we would again need on average  $N/2$  evaluations before finding the correct value. Using the superposition state we created earlier, we can apply the oracle to a combination of all possible input states.

Multiplying this matrix and the state vector obtained after applying the Hadamard gates results in the following:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ -\frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$$

Note that the third element in this vector, corresponding to the state  $|10\rangle$  (which is the value 2), is now negative. If we look at the probability vector, though, all elements in this vector are still equal to  $1/4$ , as shown in figure:



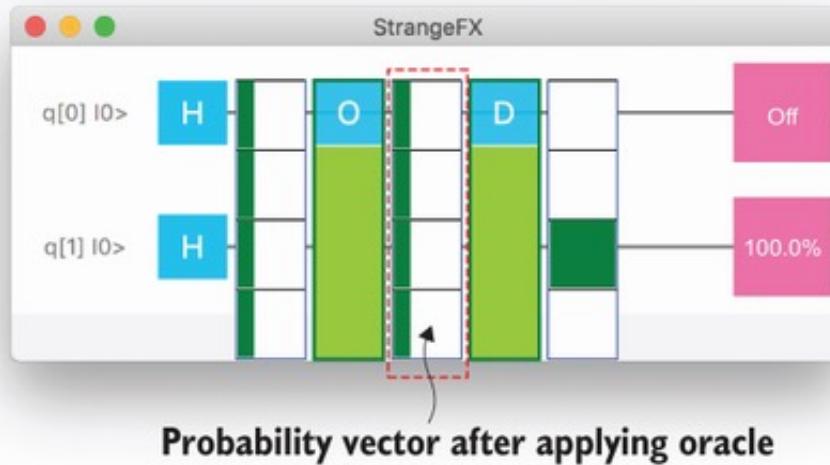
Probability vector after applying oracle

After applying the quantum oracle, all steps still have an equal probability.

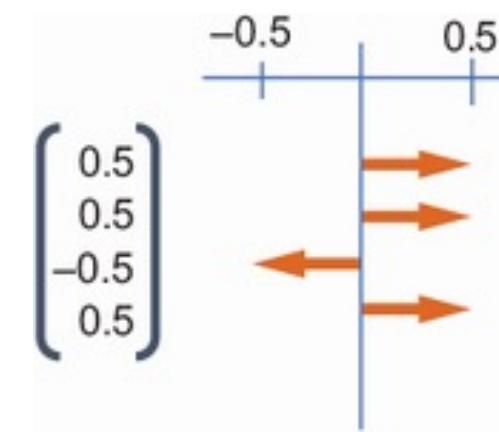
### Quantum ORACLE

The quantum oracle does not change the probabilities. If we measured the system now, we would have an equal chance of measuring any value. However, the quantum circuit itself works with the amplitudes, which are modified. In the next step, we take advantage of this. This situation shows a very important difference between the state vector, which contains amplitudes, and the probability vector, which contains probabilities. We typically talk about probabilities, but in this case, let's take a deeper look at the amplitudes.

Second figure shows the state vector after applying the quantum oracle, with the four different amplitudes as horizontal lines. A line to the right indicates a positive amplitude, and a line to the left indicates a negative amplitude.



After applying the quantum oracle, all steps still have an equal probability.



Visualization of the state vector after applying the quantum oracle

The probability is the squared modulus of the amplitude. So, if the amplitude is 0.5, the probability is 0.25. If the amplitude is -0.5, the probability is 0.25 as well, as shown in first figure.

**NOTE** By applying the quantum oracle to a superposition of all possible states, we put the quantum system into a state where the desired result is different from all other possible outcomes. However, we are not able to measure this yet.

## The algorithm behind Grover's search: Grover diffusion operator: Increasing the probability

The next step of Grover's search algorithm applies a diffusion operator to the state of the system. This operator somehow makes the implicit information of the quantum system more explicit. Remember from the previous step that the system already contains the information we need (the entry at the desired outcome has a negative amplitude), but we can't measure it yet (all probabilities are equal). This diffusion operator can be constructed by applying quantum gates or creating its matrix. (You can see the implementation in the code either in Strange or in the ch10/grover example.)

This code can be found in the `createDiffMatrix(int dim)` method in the Main class in the `ch10/grover` example. The resulting diffusion operator is a mathematical construct that is needed to ultimately increase the probability of receiving the correct result while decreasing all other probabilities. The mathematical concepts behind this operator are less relevant for developers, so we don't go into detail about them. Instead, we explain at a high level what this operator is achieving. You can find the proof online; for example, in this article about Grover's algorithm on the website of Carnegie Mellon University:

<https://www.cs.cmu.edu/~odonnell/quantum15/lecture04.pdf>

**NOTE** The diffusion operator is the part of Grover's search algorithm that allows us to retrieve higher probabilities for negative amplitudes.

The diffusion operator does an *inversion about the mean*, which is as follows:

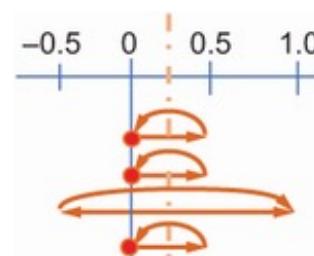
1. All values in the state vector are summed.
2. The average is calculated.
3. All values are replaced with the value that would be obtained by mirroring the value about the mean.

Let's calculate what that does with our current state vector. The four elements in the state vector are  $1/2, 1/2, -1/2, 1/2$ . The sum of those elements is thus 1:

$$\frac{1}{2} + \frac{1}{2} - \frac{1}{2} + \frac{1}{2} = 1$$

So, the average is:  $\frac{1}{4}$

We now need to "mirror" the elements (which are either  $1/2$  or  $-1/2$ ) around this value of  $1/4$ . As shown in figure 10.18, mirroring  $1/2$  results in 0. Interestingly, mirroring  $-1/2$  results in 1!



Visualization of the state vector after applying the diffusion operator

## The algorithm behind Grover's search: Grover diffusion operator: Increasing the probability

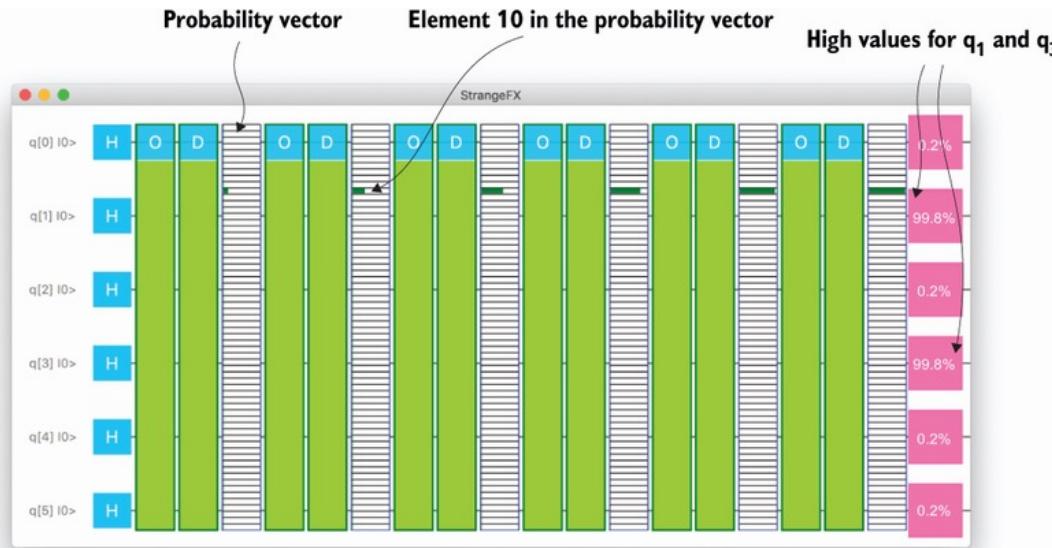
The real power of **Grover's search algorithm** comes from the combination of the quantum oracle, which flips the sign of the amplitude of the target value, and the diffusion operator, which inverts all amplitudes over their mean, thereby amplifying the negative amplitude into the largest element. In this particular case, with only two qubits, a single step is sufficient to find the correct answer to the original problem. We were provided with an oracle, and a single evaluation of that oracle was enough to determine that the element at index 2 was given the correct answer to the original function.

If there are more than two qubits, the probability of measuring the correct answer is larger than the probability of measuring any of the other options, but it is not 100%. In that case, the quantum oracle and the diffusion operator have to be applied multiple times.

It can be proven mathematically that the number of steps that provides the optimal result is the value closest to  $\sqrt{N} * \pi/4$ . In the `doGrover` method in the Main class of **ch10/grover**, this is done by the following construct:

```
private static void doGrover(int dim, int solution) {
    int N = 1 << dim;
    double cnt = Math.PI*Math.sqrt(N)/4;
    ...
    for (int i = 1; i < cnt; i++) {
        // apply a step
    }
    ...
}
```

**Looking back at figure, where  $N = 64$ , we saw a good result after six steps. Indeed, according to the algorithm just shown, the optimal number of steps is 6.28.**



**NOTE:** When we apply more steps than are optimal, the quality of the result will decrease. Therefore, it is highly recommended to follow the algorithm we have shown.

# Quantum Algorithms & Coding – Grover Searching Algorithm

## Conclusion

**Grover's search algorithm is one of the most popular quantum algorithms.** In this section, you learned that while the algorithm itself is not related to searching a database, it can be used in applications that require searching through unstructured lists.

As is often the case with quantum algorithms, Grover's search algorithm increases the probability of measuring the correct response and reduces the probability of measuring the wrong response.

Without any upfront knowledge, all possible answers have the same probability. After applying one step of the algorithm, the correct answer already has a higher probability than the other possible outcomes. After applying the optimal number of steps (the number closest to  $\sqrt{N} \cdot \pi/4$ ), the correct answer will have the highest probability.

## Summary

- Classical algorithms to search for a specific element in an unstructured list can be written in Java using a Java standard function or functional programming - Lambda.
- Using the quantum equivalent of a classical Java functional programming, you can use a quantum algorithm to do the same search.
- The time it takes a classical algorithm to find the desired element is linearly proportional to the number of elements in the list. In the quantum approach, using Grover's search algorithm, the time to find this element is proportional to the square root of the number of elements.
- **Grover's search algorithm** can be implemented using **Java Quantum Strange or QASM**.

## SHOR'S ALGORITHM



### BREAKING ALMOST ALL OF CRYPTOGRAPHY

<https://www.schneier.com/blog/archives/2023/01/breaking-rsa-with-a-quantum-computer.html>

"A group of Chinese researchers have [just published https://arxiv.org/pdf/2212.12372.pdf](https://arxiv.org/pdf/2212.12372.pdf) (January 2023) a paper claiming that they can—although they have not yet done so—break 2048-bit RSA. This is something to take seriously. It might not be correct, but it's not obviously wrong."

GitHub:

<https://github.com/johanvos/quantumjava/tree/master/ch11>

<https://github.com/critoma/quantum/tree/main/quantumjava/ch11>

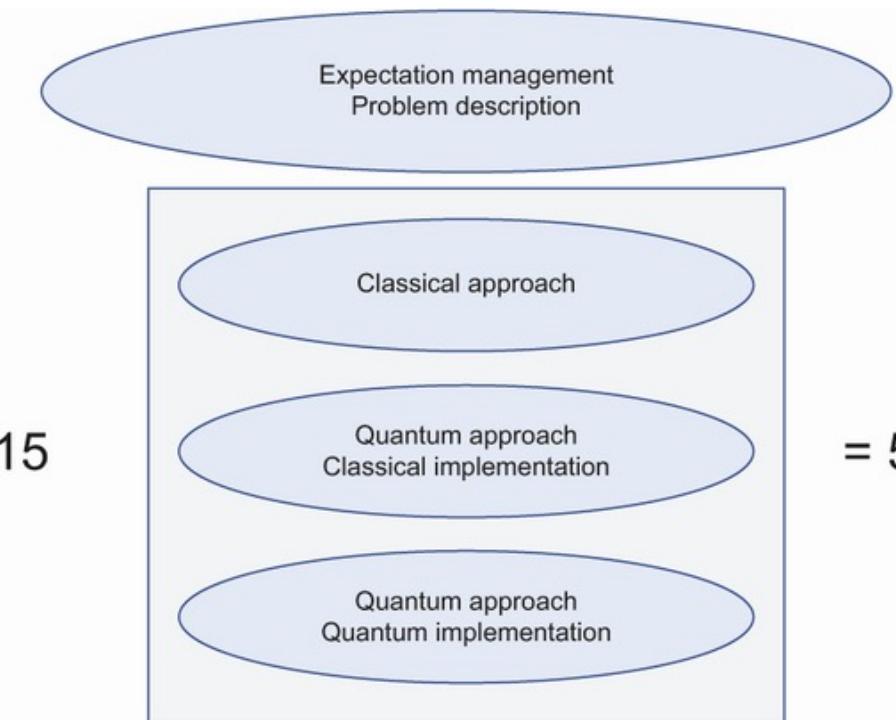
Copyright: <https://www.youtube.com/watch?v=fFW36-CGkPY>

Copyright: <https://www.manning.com/books/quantum-computing-in-action>

This section covers:

- Understanding Shor's algorithm and why it is relevant
- Solving integer factorization with classical and quantum computing techniques

In this section, we discuss one of the most famous quantum algorithms currently known. More important than the results of this algorithm is the approach taken to reach the algorithm.



**IBM Page about Shor algo doesn't exist anymore:**

<https://quantum-computing.ibm.com/composer/docs/iqx/guide/shors-algorithm>

In the Internet web archive some snapshots are inconsistent – e.g. July 2023:

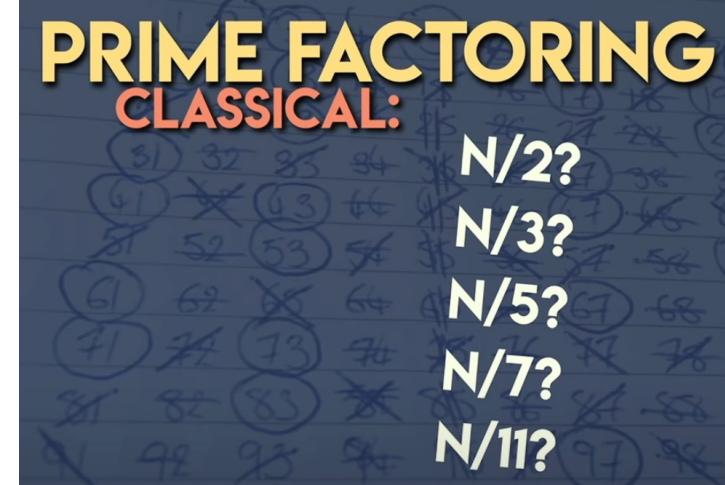
<https://web.archive.org/web/20230731022058/http://quantum-computing.ibm.com/composer/docs/iqx/guide/shors-algorithm>

**The last meaningful snapshot from IBM about Shor algo is in June 2023:**

<https://web.archive.org/web/20230605160920/https://quantum-computing.ibm.com/composer/docs/iqx/guide/shors-algorithm>

Although any integer number has a unique decomposition into a product of primes, finding the prime factors is believed to be a hard problem. In fact, the security of our online transactions rests on the assumption that factoring integers with a thousand or more digits is practically impossible. This assumption was challenged in 1995 when Peter Shor proposed a polynomial-time quantum algorithm for the factoring problem. Shor's algorithm is arguably the most dramatic example of how the paradigm of quantum computing changed our perception of which problems should be considered tractable. In this section we briefly summarize some basic facts about factoring, highlight the main ingredients of Shor's algorithm, and illustrate how it works by using a toy factoring problem.

Suppose our task is to factor an integer  $N$  with  $d$  decimal digits. The brute force algorithm goes through all primes  $p$  up to  $\sqrt{N}$  and checks whether  $p$  divides  $N$ . In the worst case, this would take time roughly  $\sqrt{N}$ , which is exponential in the number of digits  $d$ . A more efficient algorithm, known as the quadratic sieve, attempts to construct integers  $a, b$  such that  $a^2 - b^2$  is a multiple of  $N$ . Once such  $a, b$  are found, one checks whether  $a \pm b$  have common factors with  $N$ . The quadratic sieve method has asymptotic runtime exponential in  $\sqrt{d}$ . The most efficient classical factoring algorithm, known as general number field sieve, achieves an asymptotic runtime exponential in  $d^{1/3}$ .



## Quantum Algorithms & Coding – Shor Algorithm

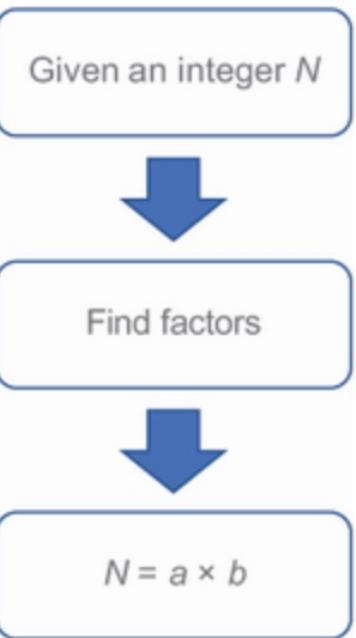
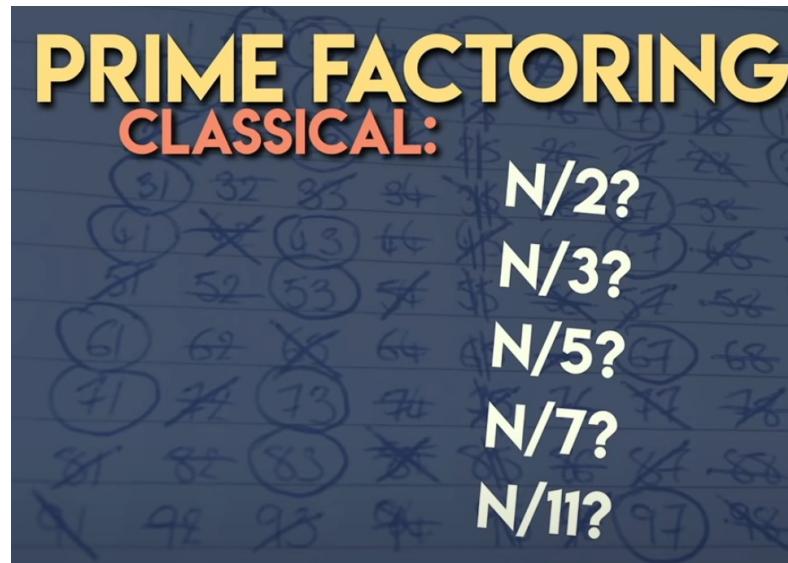
```
package com.javaqc.ch11.classicfactor;

import java.math.BigInteger;

public class Main {

    public static void main (String[] args) {
        int target = (int)(10000 * Math.random());
        int f = factor (target);
        System.out.println("Factored "+target+" in "+f+" and "+target/f);
    }

    public static int factor (int N) {
        int i = 1;
        int max = (int) Math.sqrt(N);
        while (i++ < max ) {
            if (N%i == 0) return i;
        }
        return N;
    }
}
```



Classic flow for factoring integers. A classic algorithm focuses on finding the factors for a given number and returning those factors.

```
critoma@critoma-mac: ~ % cd $QHOME/ch11/classicfactor/src/main/java
javac -cp ".:$QJSTRANGELIB/target/strange-0.1.4-
SNAPSHOT.jar:$QJSTRANGEFLIB/target/strangefx-0.1.5-SNAPSHOT.jar"
com/javaqc/ch11/classicfactor/Main.java

java -cp ".:$QJSTRANGELIB/target/strange-0.1.4-
SNAPSHOT.jar:$QJSTRANGEFLIB/target/strangefx-0.1.5-SNAPSHOT.jar"
com.javaqc.ch11.classicfactor.Main
```

Factored 9357 in 3 and 3119

# Quantum Algorithms & Coding – Shor Algorithm

## STEP 1:

$N = 15$

$a = 7$

pick a random number  $a < N$   
(check if  $a$  is a factor of  $N$ )

**SHOR** find factors for  $N$

## STEP 2:

$N = 15$

$a = 7$

find period  $r$  for  $a \bmod N$

**SHOR** find factors for  $N$

## EXAMPLE:

$N = 15$   
 $a = 7$

$$\left. \begin{array}{l} 7^1 \bmod 15 = 7 \\ 7^2 \bmod 15 = 4 \\ 7^3 \bmod 15 = 13 \\ 7^4 \bmod 15 = 1 \\ 7^5 \bmod 15 = 7 \\ 7^6 \bmod 15 = 4 \\ 7^7 \bmod 15 = 13 \end{array} \right\} r = 4$$

**SHOR** find factors for  $N$

# Quantum Algorithms & Coding – Shor Algorithm

<https://github.com/critoma/quantum/blob/main/quantumjava/ch11/semiclassicfactor/src/main/java/com/javaqc/ch11/semiclassicfactor/Main.java>

## Periodic functions

A function is called *periodic* if its evaluations are repeated at regular intervals.

The length of this interval is called the *periodicity* of the function.

An interesting periodic function is modular exponentiation, which is defined as

$$f(x) = a^x \bmod N$$

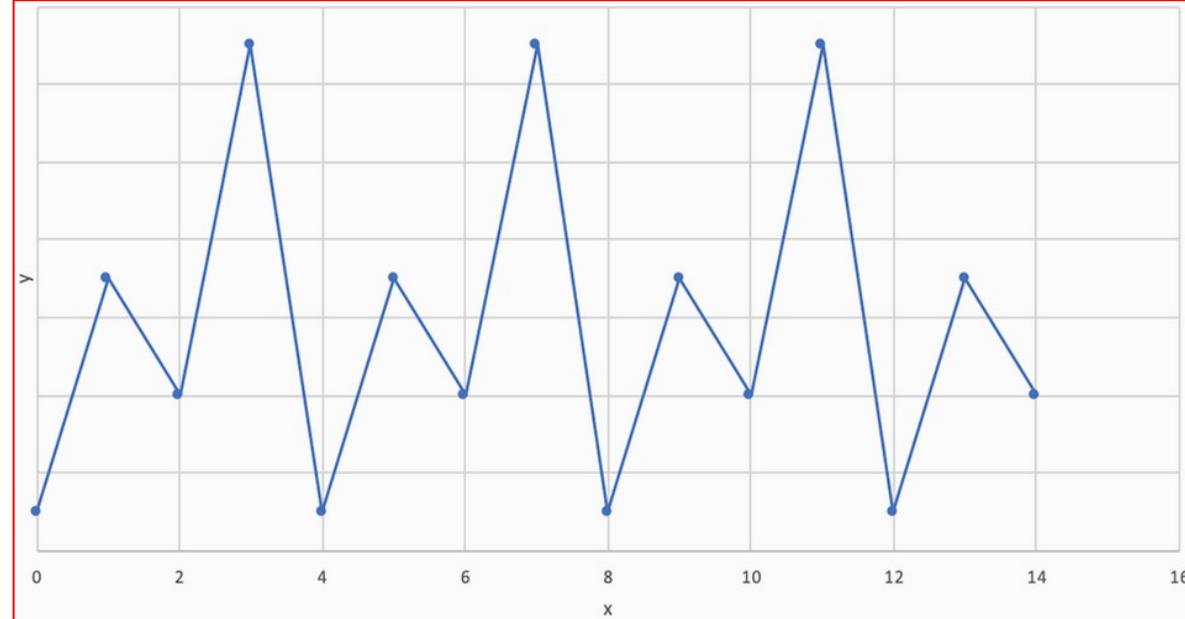
where  $a$  and  $N$  are parameters with  $a < N$ . Because of the modular operator, the outcome of this function is always smaller than  $N$ .

To give you a more tangible idea of what this function looks like, consider the case where  $a = 7$  and  $N = 15$ . In that case, figure 11.6 shows the value for

$$y = 7^x \bmod 15$$

There is a pattern in this function that repeats itself. Whenever we increase  $x$  by 4, the value of the function is the same as the original value. This is illustrated in the following table, where we show the values of  $y$  for  $x$  ranging from 0 to 8.

From these observations, it turns out that the periodicity of this function is 4.



Example of a periodic function. The periodicity of this function is 4: the same pattern in the y values comes back after every four evaluations. For example, the peaks in the function occur at x values of 3, 7, and 11.

$x$	0	1	2	3	4	5	6	7	8
$y$	1	7	4	13	1	7	4	13	1

## Quantum Algorithms & Coding – Shor Algorithm

### Period finding

It has been known to mathematicians since the 1970s that factoring becomes easy if one can solve another hard problem: find a period of the modular exponential function. The period-finding problem is defined as follows. Given integers  $N$  and  $a$ , find the smallest positive integer  $r$  such that  $a^r - 1$  is a multiple of  $N$ . The number  $r$  is called the period of  $a$  modulo  $N$ . Recall that in modular arithmetic, the remainder of a division  $a/N$  is called the value of  $a$  modulo  $N$  and denoted  $a \pmod{N}$ . For example,  $1 = 16 = 91 \pmod{15}$ . Thus, the period of  $a$  modulo  $N$  is the smallest positive integer  $r$  such that  $a^r = 1 \pmod{N}$ . For example, suppose  $N = 15$  and  $a = 7$ . Then

$$\begin{aligned} 7^2 &= 4 \pmod{15} \\ 7^3 &= 4 \cdot 7 = 13 \pmod{15} \\ 7^4 &= 13 \cdot 7 = 1 \pmod{15} \end{aligned}$$

That is, 7 has period 4 modulo 15. Note that computing the higher powers of 7 would give rise to a periodic sequence:  $7^{x+4} = 7^x \pmod{15}$  for any integer  $x$ . Thus,  $r = 4$  is the period of the modular exponential function  $7^x$ . In general, the period-finding problem is well-defined if  $N$  and  $a$  are co-prime (have no common factors).

## Quantum Algorithms & Coding – Shor Algorithm

### From factoring to period finding

Assume for a moment that we are given a period-finding machine that takes as input co-prime integers  $N, a$  and outputs the period of  $a$  modulo  $N$ . We will now show how to use the machine to find all prime factors of  $N$ . For simplicity, assume that  $N$  has only two distinct prime factors:

$$N = p_1 p_2$$

First, pick a random integer  $a$  between 2 and  $N - 1$  and compute the greatest common divisor ( $\text{gcd}$ ) ( $N, a$ ). This can be done very efficiently using [Euclid's algorithm](#) ↗. If we are lucky,  $N$  and  $a$  have some common prime factors, in which case  $\text{gcd}(N, a)$  equals  $p_1$  or  $p_2$ , so we are done. From now on, let us assume that  $\text{gcd}(N, a) = 1$ , that is,  $N$  and  $a$  are co-prime. Let  $r$  be the period of  $a$  modulo  $N$  computed by the machine. Repeat the above steps with different random choices of  $a$  until  $r$  is even. It can be shown that a significant fraction of all integers  $a$  have an even period (see Table 1 for examples), so on average one needs only a few repetitions.

At this point we have found some pair  $r, a$  such that  $r$  is even, and  $r$  is the smallest integer such that  $a^r - 1$  is a multiple of  $N$ . Let us use the identity

$$a^r - 1 = (a^{r/2} - 1)(a^{r/2} + 1)$$

<https://web.archive.org/web/20230605160920/https://quantum-computing.ibm.com/composer/docs/iqx/guide/shors-algorithm>

The above shows that  $a^{r/2} - 1$  is not a multiple of  $N$  (otherwise the period of  $a$  would be  $r/2$ ). Assume for a moment that  $a^{r/2} + 1$  is not a multiple of  $N$ . Then neither of the integers  $a^{r/2} \pm 1$  is a multiple of  $N$ , but their product is. This is possible only if  $p_1$  is a prime factor of  $a^{r/2} - 1$  and  $p_2$  is a prime factor of  $a^{r/2} + 1$  (or vice versa). We can thus find  $p_1$  and  $p_2$  by computing  $\gcd(N, a^{r/2} \pm 1)$ ; see Table 1 for examples. In the remaining “unlucky” case, when  $a^{r/2} + 1$  is a multiple of  $N$ , we give up and try a different integer  $a$ . For example,  $a = 14$  is the only unlucky integer in Table 1. In general, it can be shown that the unlucky integers  $a$  are not too frequent, so on average, only two calls to the period-finding machine are sufficient to factor  $N$ .

$2^1 = 2$	$3^1 = 3$	$4^1 = 4$
$2^2 = 4$	$3^2 = 9$	$4^2 = 16$
$2^3 = 8$	$3^3 = 27$	$4^3 = 64$
$2^4 = 16$	$3^4 = 81$	$4^4 = 256$
$2^5 = 32$	$3^5 = 243$	...
$2^6 = 64$	$3^6 = 729$	

## Quantum Algorithms & Coding – Shor Algorithm

### EXAMPLE:

$N = 15$		$a = 7$
$7^1 \bmod 15 = 7$		
$7^2 \bmod 15 = 4$		
$7^3 \bmod 15 = 13$		$r = 4$
$7^4 \bmod 15 = 1$		
$7^5 \bmod 15 = 7$		
$7^6 \bmod 15 = 4$		
$7^7 \bmod 15 = 13$		

**SHOR** find factors for  $N$

$a$	Period $r$	$\gcd(15, a^{r/2} - 1)$	$\gcd(15, a^{r/2} + 1)$
1	1		
2	4	3	5
4	2	3	5
7	4	3	5
8	4	3	5
11	2	5	3
13	4	3	5
14	2	1	15

## Quantum Algorithms & Coding – Shor Algorithm

### STEP 3:

$N = 15$

$a = 7$

$r = 4$

if  $r$  is odd

or

$$(a^{r/2} + 1) \bmod N == 0$$

go back to step 1

**SHOR** find factors for  $N$

pick number  $a$   
find period  $r$  !  
check period  
calculate factors  $p$  and  $q$

### STEP 4:

calculate prime factors

$$p = \gcd(a^{r/2} - 1, N)$$

$$q = \gcd(a^{r/2} + 1, N)$$

**SHOR** find factors for  $N$

$N = 15$

$a = 7$

$r = 4$

$N = 15$

$a = 7$

$r = 4$

### STEP 4:

$$\gcd(7^{4/2} - 1, 15) = 3$$

$$\gcd(7^{4/2} + 1, 15) = 5$$

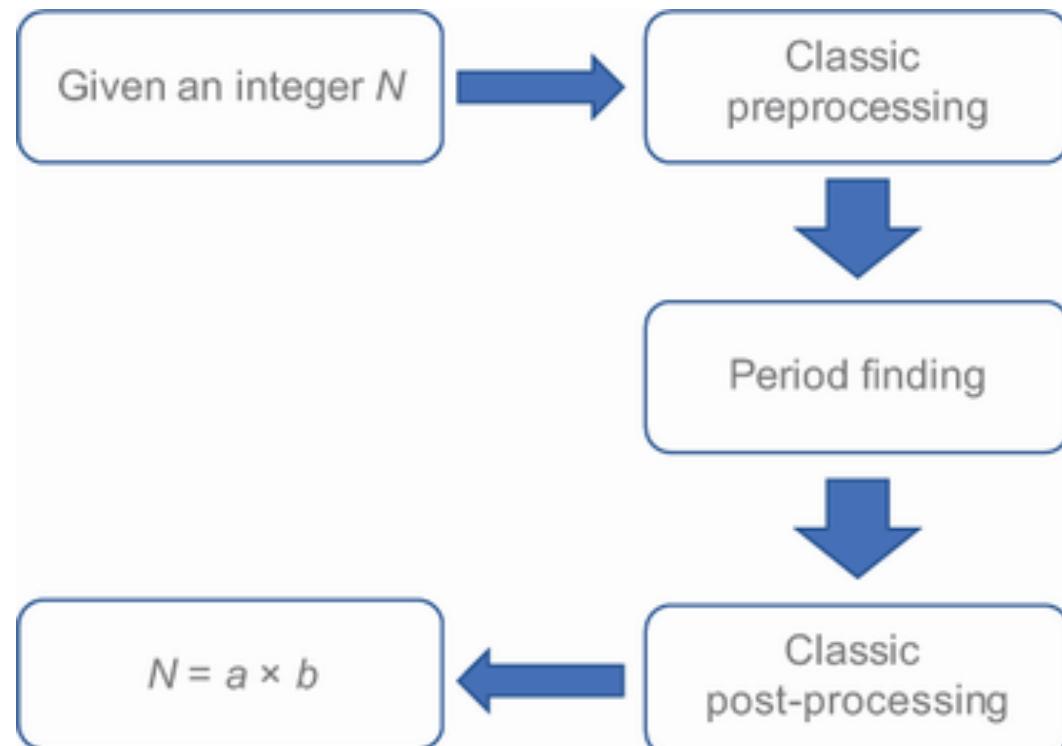
**SHOR** find factors for  $N$

## A. Solving a different problem

## Quantum Algorithms & Coding – Shor Algorithm

<https://github.com/critoma/quantum/blob/main/quantumjava/ch11/semitropicfactor/src/main/java/com/javaqc/ch11/semitropicfactor/Main.java>

Finding the periodicity of a function is a problem that is even harder than the prime factors finding with a standard computer, but it can be solved by a quantum computer in polynomial time. After the quantum computing part, the result needs to be translated to the original problem again.



Solving a different problem. Instead of directly finding the factors for the number  $N$ , we translate the original problem to a different problem, solve that, and translate the result back to the original question, which can then be answered.

**NOTE** Quantum computers can provide a huge speedup for some algorithms, but not all. Therefore, the key to creating quantum applications is often in finding a way to translate the original problem into a problem that can be solved easily by a quantum computer (e.g., in polynomial time instead of exponential time) and then transforming it back to the original domain.

The original problem is to find two integers that, when multiplied together, yield the integer  $N$  that we want to factor. The problem that we actually solve using a quantum computer looks very different and is formulated as follows: given an integer  $a$  and an integer  $N$ , find the periodicity of the function

$$a^x \bmod N$$

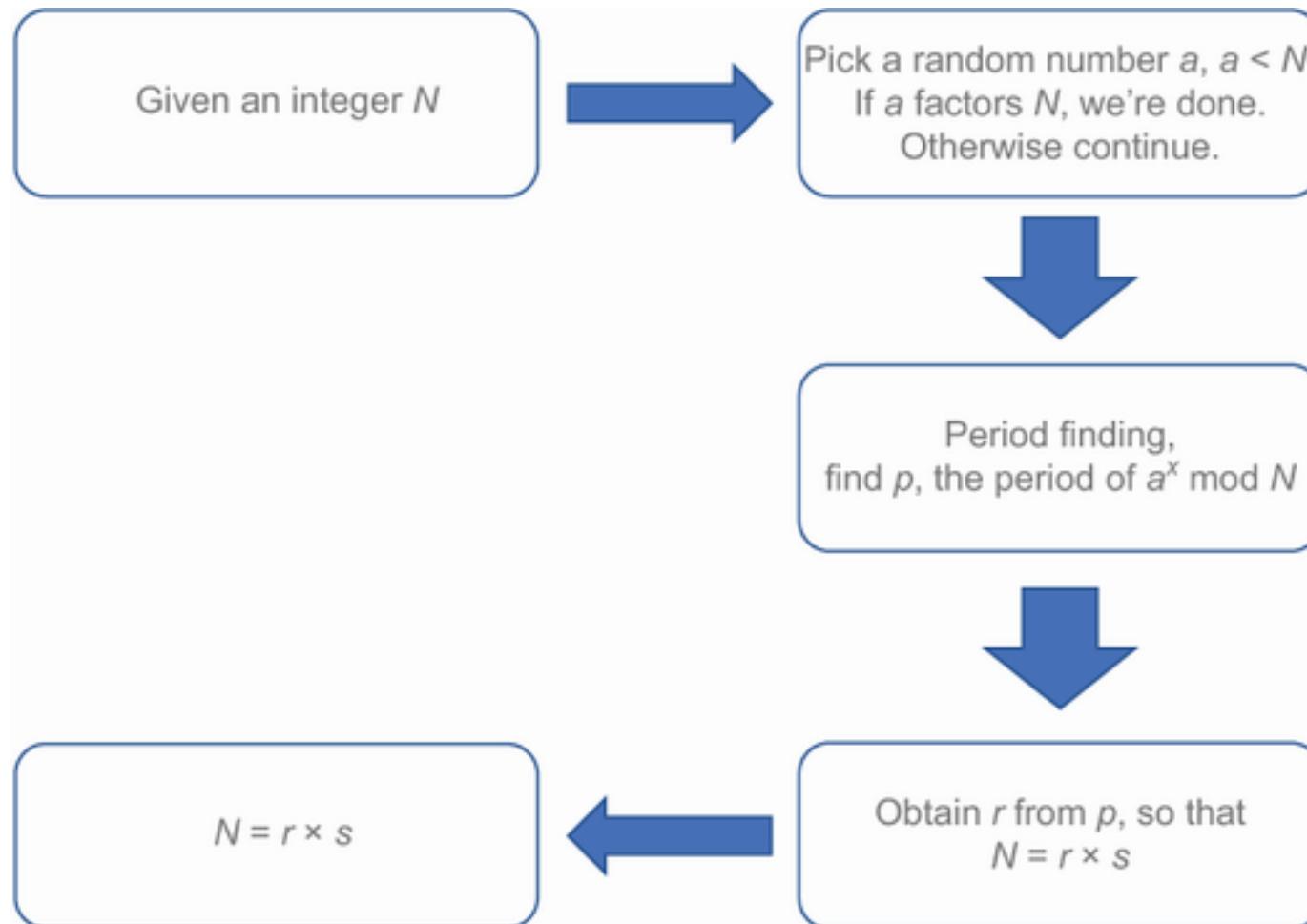
## A. Solving a different problem

## Quantum Algorithms & Coding – Shor Algorithm

<https://github.com/critoma/quantum/blob/main/quantumjava/ch11/semiclassicfactor/src/main/java/com/javaqc/ch11/semiclassicfactor/Main.java>

While that problem looks very different from the original problem, it can be proven mathematically that they are related. Once we find the periodicity of this function, we can find the factors for  $N$  easily.

We are not going to provide the mathematical proof, but we will show the relationship between periodicity and factors by looking at some Java code. The flow that we follow in this code is illustrated in figure:



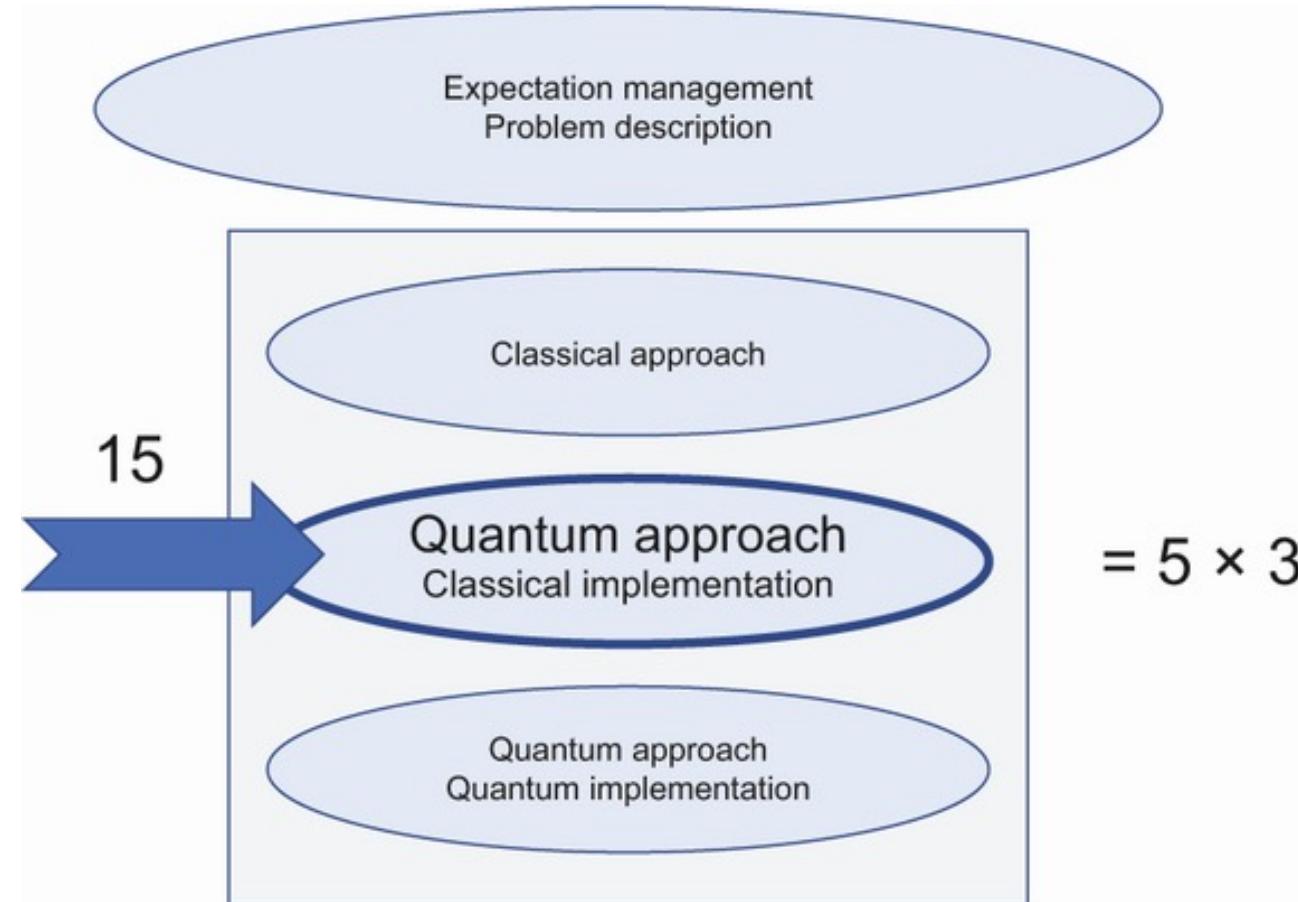
Detailed flow for both classic and quantum implementations. The preprocessing and post-processing, which transform the original problem into the problem of period finding, are similar for the classic and quantum implementations. The period finding can be implemented in a classic way or a quantum way.

## A. Solving a different problem

## Quantum Algorithms & Coding – Shor Algorithm

<https://github.com/critoma/quantum/blob/main/quantumjava/ch11/semiclassicfactor/src/main/java/com/javaqc/ch11/semiclassicfactor/Main.java>

In the next section, we explain how the critical part—finding the periodicity of the modular exponentiation—is achieved using a quantum algorithm. Before we do that, we will write out the complete algorithm using classical computing. This is the second approach in our mental model:



Mental model: classic implementation of the quantum approach. We don't use direct factorization techniques as we did in the classic approach. We use the technique of period finding, which is part of the quantum approach, but we first develop that in a classic way.

## A. Solving a different problem

## Quantum Algorithms & Coding – Shor Algorithm

<https://github.com/critoma/quantum/blob/main/quantumjava/ch11/semiclassicfactor/src/main/java/com/javaqc/ch11/semiclassicfactor/Main.java>

```
public static int factor (int N) {  
    // PREPROCESSING  
    System.out.println("We need to factor "+N);  
    int a = 1+ (int)((N-1) * Math.random());  
    System.out.println("Pick a random  
                    number a, a < N: "+a);  
  
    int gcdan = gcd(N,a);  
    System.out.println("calculate gcd(a, N):"+ gcdan);  
    if (gcdan != 1) return gcdan;  
  
    // PERIOD FINDING  
    int p = findPeriod (a, N);  
  
    // POSTPROCESSING  
    System.out.println("period of f = "+p);  
    if (p%2 == 1) {  
        System.out.println("odd period, restart.");  
        return -1;  
    }  
    int md = (int)(Math.pow(a, p/2) +1);  
    int m2 = md%N;  
    if (m2 == 0) {  
        System.out.println("m^p/2 + 1 = 0 mod N,  
                           restart");  
        return -1;  
    }  
    int f2 = (int)Math.pow(a, p/2) -1;  
    return gcd(N, f2);  
}
```

Factor method: classical implementation of the quantum approach

```
public static void main (String[] args) {  
    int target = (int)(10000 * Math.random());  
    int f = factor (target);  
    System.out.println("Factored "+target+" in "+f+" and "+target/f);  
}
```

- ➊ Preprocessing part begins
- ➋ Picks a random number a between 1 and N
- ➌ Calculates the greatest common denominator (GCD) between a and N
- ➍ If the GCD is not 1, we are done, since that means the GCD is a factor of N.
- ➎ Finds the periodicity of the modular exponentiation function. This is the bulk of the work, and it is detailed in the next listing.
- ➏ If the period turns out to be an odd number, we can't use it and have to repeat the process. We return -1 to inform the caller that the operation failed.
- ➐ Performs minor mathematical operations on the period to obtain a factor of N. This may still fail, in which case we return -1 to the caller.

The factor method calls the **findPeriod** method to obtain the periodicity of the function  **$a^x \bmod N$** . This function can be executed fast on a quantum computer.

**NOTE:** When we say the function can be executed fast, we mean it is done in polynomial time.

## B. Classic period finding

## Quantum Algorithms & Coding – Shor Algorithm

<https://github.com/critoma/quantum/blob/main/quantumjava/ch11/semiclassicfactor/src/main/java/com/javaqc/ch11/semiclassicfactor/Main.java>

Classic implementation for finding the period of a function

```
public static int findPeriod(int a, int N) {  
    int r = 1;  
    long mp = (long) (Math.pow(a,r)) % N;  
    BigInteger bn = BigInteger.valueOf(N);  
    BigInteger bi = BigInteger.valueOf(a);  
    while (mp != 1) {  
        r++;  
        BigInteger mpd = bi.pow(r);  
        BigInteger mpb = mpd.mod(bn);  
        mp = mpb.longValue();  
    }  
    return r;  
}
```

- ① 1 The periodicity of a function is at least 1. Note that for  $r = 0$ , the value of any  $a^r \bmod N$  is always 1.
- ② 2 Calculates the first result of  $a^1 \bmod N$
- ③ 3 As long as the result is not 1 (which is the result of  $a^0 \bmod N$ ), we need to keep increasing  $r$  and continue.
- ④ 4 Calculates the next value of  $a^r$
- ⑤ 5 Calculates the modulus of this value with  $N$
- ⑥ 6 Once the modulus is 1, we have the same result as with  $r = 0$ , so  $r$  now holds the periodicity.

This function (finding the periodicity of a modular exponentiation) can also be implemented on a quantum computer, using a quantum algorithm with quantum properties. It is the core of Shor's algorithm, and we discuss it in the next section.

## C. Post-processing step

## Quantum Algorithms & Coding – Shor Algorithm

<https://github.com/critoma/quantum/blob/main/quantumjava/ch11/semiclassicfactor/src/main/java/com/javaqc/ch11/semiclassicfactor/Main.java>

While the most critical part of the algorithm is to find the period of a function, it is still important that we can convert this periodicity into a factor. This is done in the post-processing steps. We won't give a rigorous mathematical proof for the algorithm but rather will show the different steps together with some values so that you can follow what is happening.

Let's first recap the parameters we are dealing with:

- $N$  is the number we want to factor: e.g.,  $N = 493$ .
- $a$  is the random number, smaller than  $N$ , that we use to initiate the period finding part. The random number used in our example was  $a = 6$ .
- $p$  is the period of the function  $a^x \bmod N$ , which we obtained from the period-finding part of the algorithm. It turned out that in our case,  $p = 112$ .

Because of the definition of a period function, we know that

$$f = a^x \bmod a^{x+p} \bmod N$$

For  $x = 0$ , this means

$$a^0 \bmod N = a^p \bmod N$$

or

$$1 = a^p \bmod N$$

With the numbers we have, this comes down to

$$6^{112} \bmod 493 = 1$$

Github: <https://github.com/critoma/quantum/tree/main/quantumjava/ch11/semiclassicfactor>

## C. Post-processing step Quantum Algorithms & Coding – Shor Algorithm <https://github.com/critoma/quantum/tree/main/quantumjava/ch11/jshell>

The example repository contains a script named `checkperiod` in the `ch11/jshell` directory that can be loaded into `jshell`. The contents of this script are as follows:

```
int N = 493;                                ①
int a = 6;
int p = 112;
int u = 1;
for (int i = 0; i < p; i++) {
    u = (u * p) % N;                         ②
}
System.out.println("This should be 1: " + u);  ④
```

- ① Initializes the values to the ones we use in our example
- ② initializes variable `u` with the value 1. It will hold the result of the calculations.
- ③ Inside this loop, the modular exponentiation is executed as a number of modular multiplications. This allows the result to stay below `N`, as otherwise, the result would soon be too large to fit in an Integer value.
- ④ Prints the result, which we hope is equal to 1.

## C. Post-processing step Quantum Algorithms & Coding – Shor Algorithm <https://github.com/critoma/quantum/tree/main/quantumjava/ch11/jshell>

You can either type the entries in the script or use the `/open` command in `jshell` to load the script. In both cases, the result will be printed. Following is the result of this operation in shell. Note that we added a `/list` command at the end to show the list of commands that were executed. The `/exit` command closes jshell.

$$a^p \bmod N = 1$$

```
| Welcome to JShell -- Version 17  
| For an introduction type: /help intro
```

```
jshell> /open jshell/checkperiod  
This should be 1: 1
```

```
jshell> /list
```

```
1 : int N = 493;  
2 : int a = 6;  
3 : int p = 112;  
4 : int u = 1;  
5 : for (int i = 0; i < p; i++) {  
     u = (u * p) % N;  
}  
6 : System.out.println("This should be 1: " + u);
```

```
jshell>
```

it follows that

$$a^p - 1 = kN$$

with `k` an integer as well. We now use the equation

$$x^2 - 1 = (x + 1)(x - 1)$$

and we can write the previous equation as

$$(a^{p/2} + 1)(a^{p/2} - 1) = kN$$

Let's simplify this as follows:

$$(u + 1)(u - 1) kN$$

with

$$u = a^{p/2}$$

By expanding both the left and right terms to their factors, it follows that the factors of  $N$  (if there are factors, which is not the case if  $N$  is prime) should also be on the left-hand side of the equation. Therefore, the greatest common denominator ( $\gcd$ ) of  $N$  and  $u + 1$  should be a factor of  $N$ .

## C. Post-processing step Quantum Algorithms & Coding – Shor Algorithm <https://github.com/critoma/quantum/tree/main/quantumjava/ch11/jshell>

Let's verify this with the values from our example. The same directory containing the checkperiodjshell script also has the script calculatef, which does exactly what we described:

```
int gcd(int a, int b) {  
    int x = a > b ? a : b;  
    int y = x == a ? b : a;  
    int z = 0;  
    while (y != 0) {  
        z = x % y;  
        x = y;  
        y = z;  
    }  
    return x;  
  
}  
  
int N = 493;  
int a = 6;  
int p = 112;  
int u = 1;  
for (int i = 0; i < p/2; i++) {  
    u = (u * p) % N;  
}  
  
System.out.println("This is u mod N: " + u);  
System.out.println("This is gcd: " + gcd(u + 1, N));
```

In this section, you succeeded in calculating a factor for  $N$ , assuming that you could calculate the period of a function  $a^x \bmod N$ . For now, that period calculation was still done in a classic way, but in the next section, we present the quantum implementation. The post-processing, explained in this step, stays the same.

①

① Defines a function that calculates the greatest common denominator of two integers. We don't go into the details of this function, but you can check that it works by invoking it with numbers you know.

②

② Again, defined the values we use in this example

③

③ Calculates the  $u$  value as described previously

④

④ Prints the value of  $u$

⑤

⑤ Calculates the gcd of  $u + 1$  and  $N$  to find a factor of  $N$ , and prints it

Running this in `jshell` by invoking `/load jshell/calculatef` shows the following result:

②

```
| Welcome to JShell -- Version 16-ea  
| For an introduction type: /help intro
```

③

```
jshell> /open jshell/calculatef  
This is u mod N: 407  
This is gcd: 17
```

④

And indeed, 17 is a factor of the original value of  $N = 493$ :

⑤

$493 = 17 * 29$

## Quantum Algorithms & Coding – Shor Algorithm

### Quantum based Implementation

<https://github.com/critoma/quantum/tree/main/quantumjava/ch11/quantumfactor>

<https://github.com/critoma/quantum/blob/main/quantumjava/ch11/quantumfactor/src/main/java/com/javaqc/ch11/quantumfactor/Main.java>

<https://github.com/critoma/quantum/blob/main/strange/src/main/java/org/redfx/strange/algorithm/Classic.java>

$$7^1 \bmod 15 = 7$$

$$7^2 \bmod 15 = 4$$

$$7^3 \bmod 15 = 13$$

$$7^4 \bmod 15 = 1$$

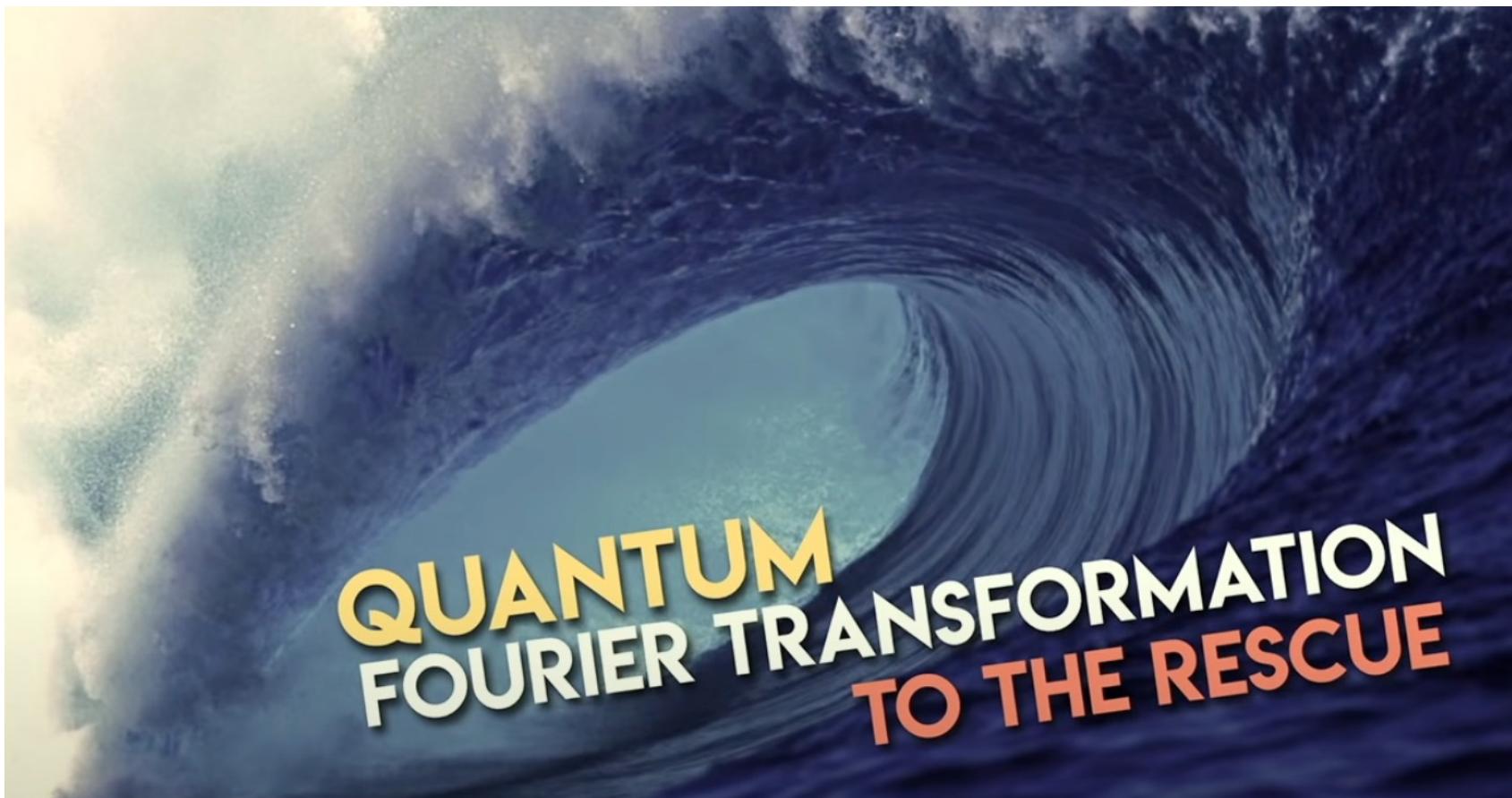
$$7^5 \bmod 15 = 7$$

$$7^6 \bmod 15 = 4$$

$$7^7 \bmod 15 = 13$$

...

SAME PROBLEM...



# Quantum Algorithms & Coding – Shor Algorithm

## Quantum based Implementation

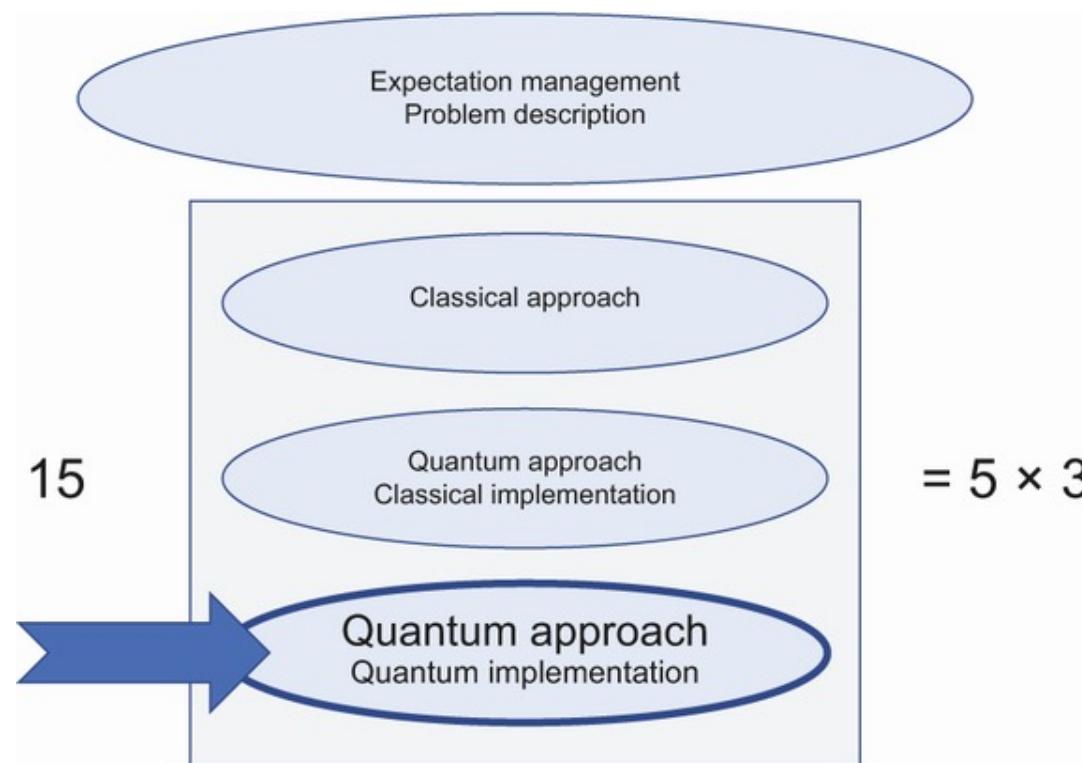
<https://github.com/critoma/quantum/tree/main/quantumjava/ch11/quantumfactor>

<https://github.com/critoma/quantum/blob/main/quantumjava/ch11/quantumfactor/src/main/java/com/javaqc/ch11/quantumfactor/Main.java>

<https://github.com/critoma/quantum/blob/main/strange/src/main/java/org/redfx/strange/algorithm/Classic.java>

<https://github.com/redfx-quantum/strange/blob/main/src/main/java/org/redfx/strange/algorithm/Classic.java>

<https://quantum-computing.ibm.com/composer/>



Mental model: quantum implementation of the quantum approach. In this approach, the preprocessing and post-processing are done in a classic way, but the period finding is done using a quantum algorithm.

## Quantum Algorithms & Coding – Shor Algorithm

### ENTANGLE SOME QUBITS

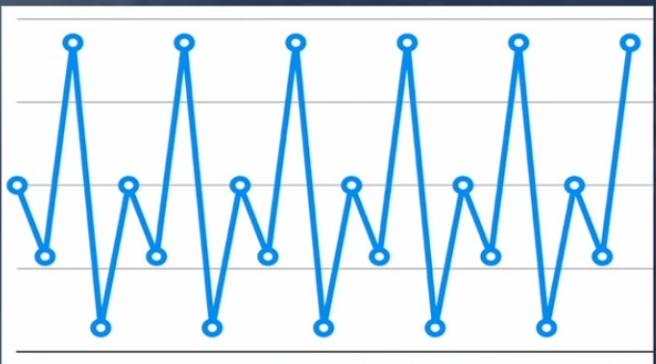
$|000\rangle$   
 $|001\rangle$   
 $|010\rangle$   
 $|011\rangle$       ← entangle  
 $|100\rangle$       HADAMARD!  
 $|101\rangle$   
 $|110\rangle$   
and CNOT gates

$|000\rangle : 7^1 \text{ mod } 15 = 7$   
 $|001\rangle : 7^2 \text{ mod } 15 = 4$   
 $|010\rangle : 7^3 \text{ mod } 15 = 13$   
 $|011\rangle : 7^4 \text{ mod } 15 = 1$   
 $|100\rangle : 7^5 \text{ mod } 15 = 7$   
 $|101\rangle : 7^6 \text{ mod } 15 = 4$   
 $|110\rangle : 7^7 \text{ mod } 15 = 13$

TRANSFORM A SIGNAL FROM  
TIME DOMAIN  
INTO THE  
FREQUENCY DOMAIN

with Fourier Transform

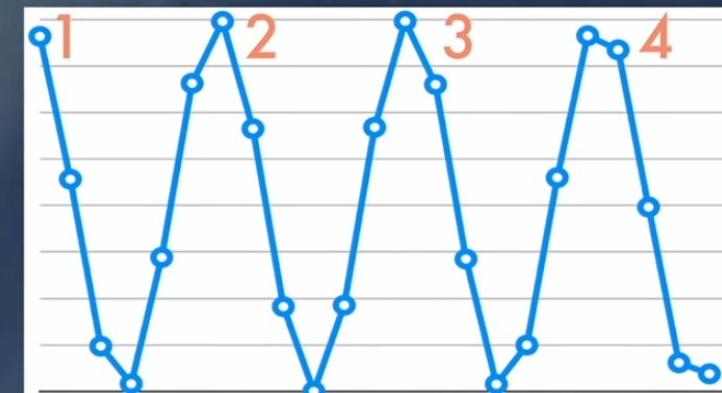
### TIME DOMAIN



7, 4, 13, 1, 7, 4, 13, 1, 7, 4, 13, 1, ...

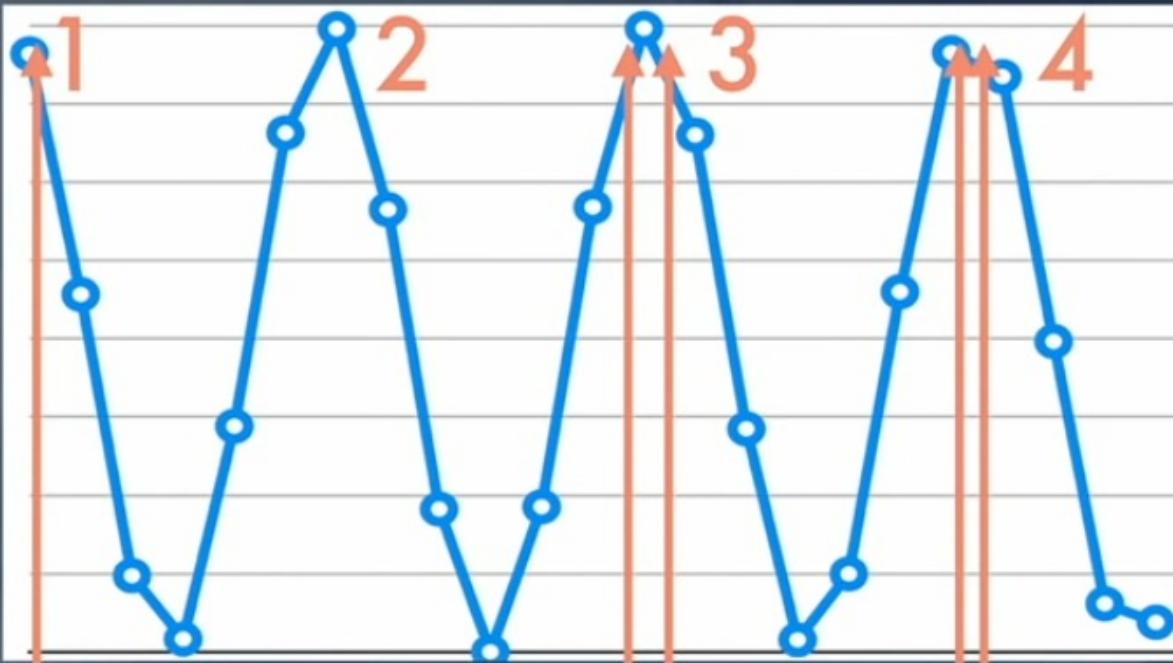
Fourier Transform

### FREQUENCY DOMAIN



0.....128

## USING LIMITED CONTINUED FRACTIONS



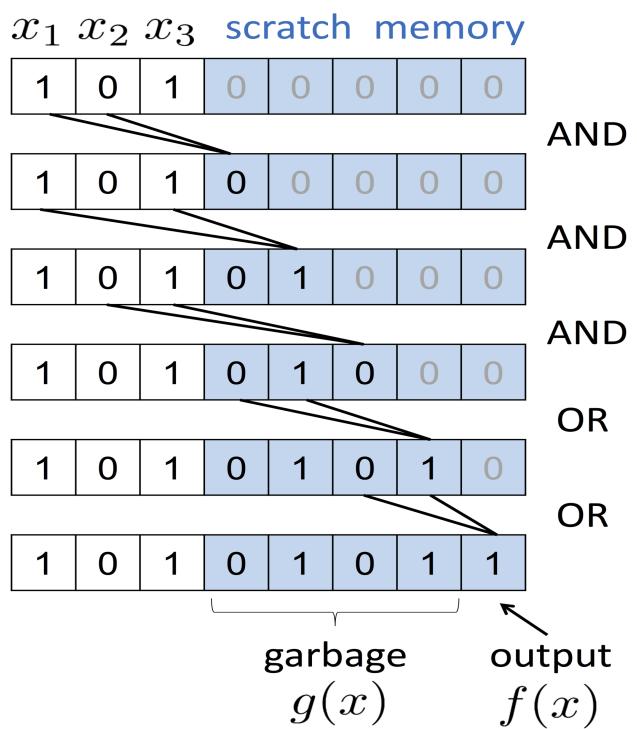
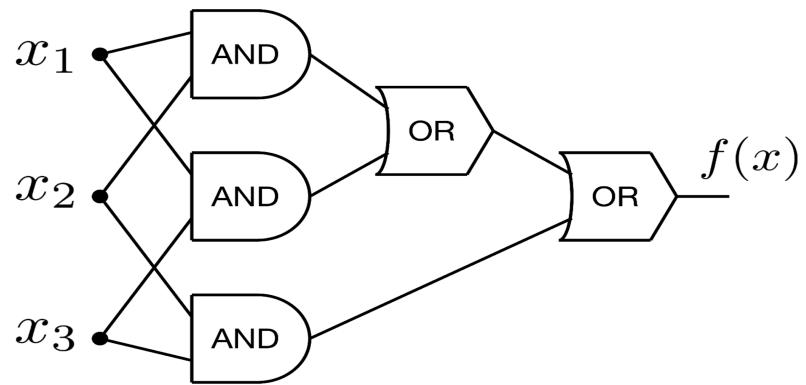
65	$1/2$	$2/4$
98	$3/4$	$3/4$
96	$3/4$	$3/4$
63	$1/2$	$2/4$
0	0	0

MOST LIKELY  $r = 4$

# Quantum Algorithms & Coding – Shor Algorithm

$a$	Period $r$	$\gcd(15, a^{r/2} - 1)$	$\gcd(15, a^{r/2} + 1)$
1	1		
2	4	3	5
4	2	3	5
7	4	3	5
8	4	3	5
11	2	5	3
13	4	3	5

$2^1 = 2$	$3^1 = 3$	$4^1 = 4$
$2^2 = 4$	$3^2 = 9$	$4^2 = 16$
$2^3 = 8$	$3^3 = 27$	$4^3 = 64$
$2^4 = 16$	$3^4 = 81$	$4^4 = 256$
$2^5 = 32$	$3^5 = 243$	$4^5 = \dots$
$2^6 = 64$	$3^6 = 729$	



## Summary

- Finding the factors of an integer is a very popular challenge in a number of IT domains.
- Using a classical approach requires an exponential amount of time when the target (the integer that needs to be factored) becomes larger.
- The goal of Shor's algorithm is to find the factors of an integer.
- Shor's algorithm shows that it can be beneficial to transform a specific problem into another problem: one that can be solved more easily (faster) by a quantum computer. In particular, Shor's algorithm transforms the problem of factorization into the problem of finding the periodicity of a periodic function.



Shor Algorithm, ECDSA/RSA Replacement – SSL/PKI/Blockchain implications, NIST Rounds, ...

**PQC** – Post Quantum Crypto



It's not just about combining Mathematics, Quantum Mechanics/Physics and Quantum Computing, but providing hybrid pragmatic secure solutions

---

What is quantum safe cryptography? Quantum safe

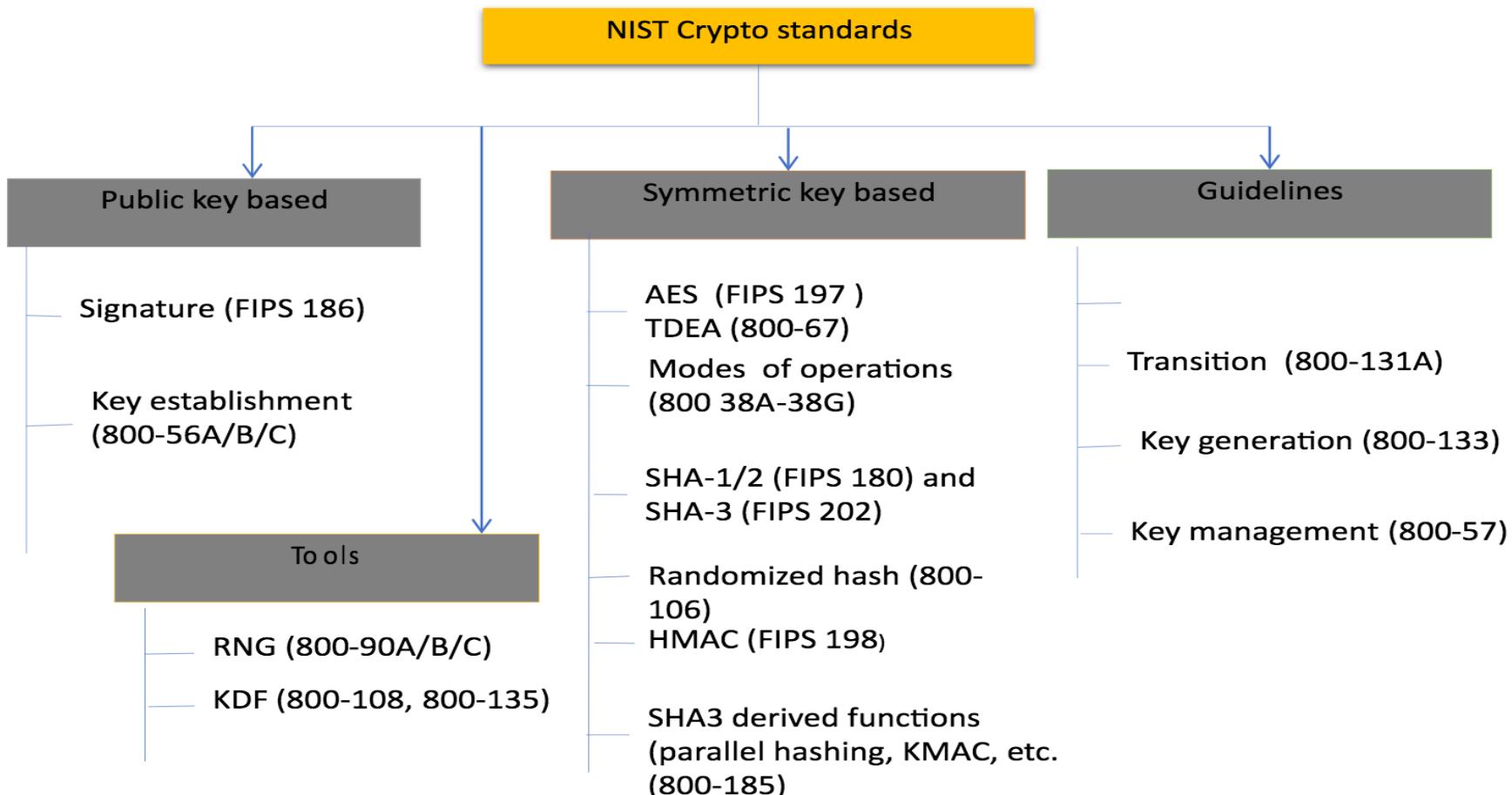
**Quantum-safe cryptography** refers to efforts to identify algorithms that are resistant to attacks by both classical and quantum computers, to keep assets **secure** even after a large-scale quantum computer has been built.

**"Post-quantum cryptography"** (sometimes referred to as **quantum-proof**, or **quantum-resistant**) refers to **cryptographic** algorithms (usually **public-key** algorithms) that are thought to be secure against an attack by a **quantum computer**. As of 2019/2023, **this is not true** for the most popular public-key algorithms, which can be efficiently broken by a sufficiently strong quantum computer. The problem with currently popular algorithms is that their security relies on one of three hard mathematical problems: the **integer factorization problem**, the **discrete logarithm problem** or the **elliptic-curve discrete logarithm problem**. All of these problems can be easily solved on a sufficiently powerful quantum computer running **Shor's algorithm**.

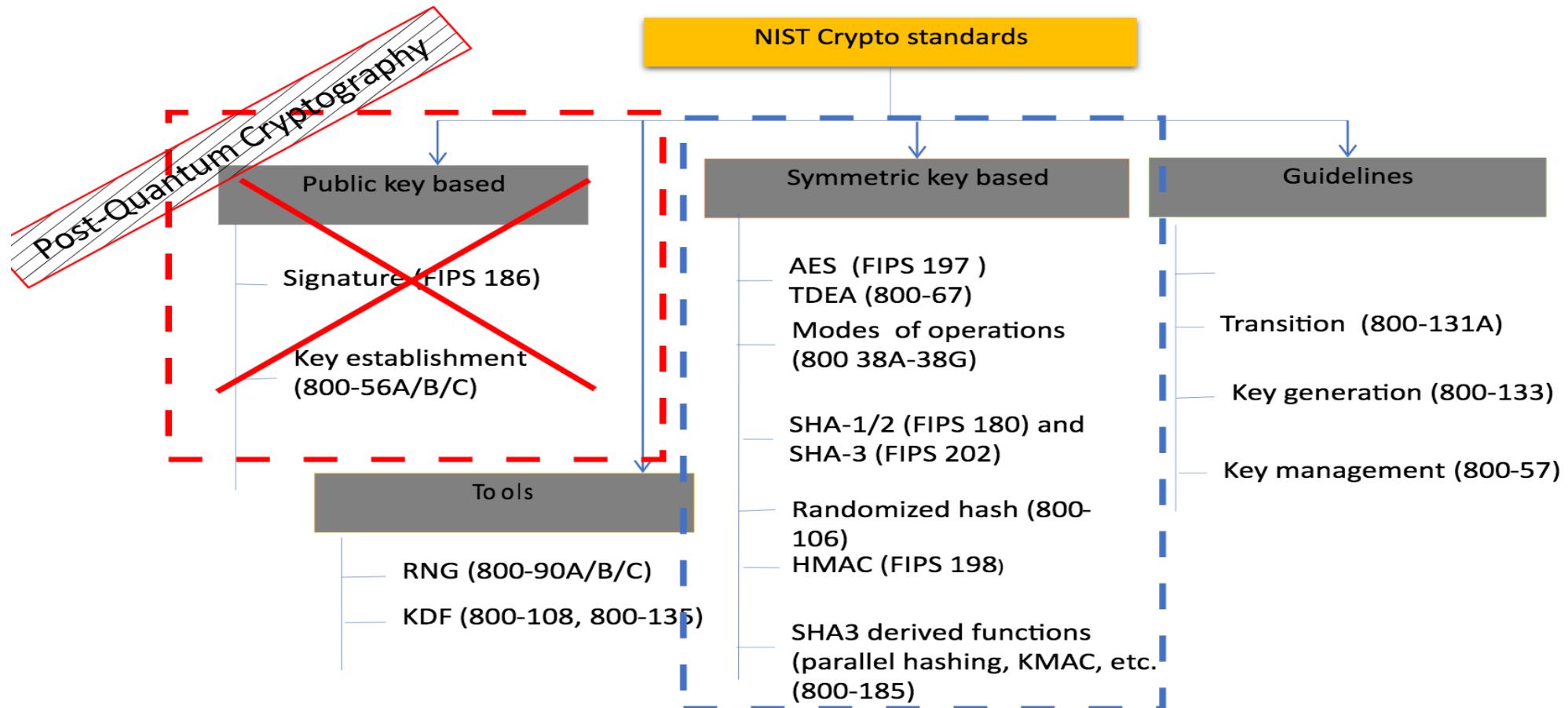
## PQC – POST QUANTUM CRYPTOGRAPHY, QUANTUM SAFE CRYPTO, CRYPTO AGILITY

WHY IS POST-QUANTUM-SAFE CRYPTOGRAPHY IMPORTANT FOR NETWORKING – SSL/TLS, DATABASES / PAYMENT SYSTEMS/ IOT / BLOCKCHAIN / SECURE ELEMENTS / WALLETS / JAVA CARD ...?

# The Sky is Falling?



# The Sky is Falling?



## When will a Quantum Computer be Build?

– IT IS ALREADY (D-Wave, IBM, Google, ...)

- Quantum computers are 25 years ahead in the future and always will be
- Dr. Michele Mosca, University of Waterloo (2016):

*“There is a 1 in 7 chance that some fundamental public-key crypto will be broken by quantum by 2026, and a 1 in 2 chance of the same by 2031”*

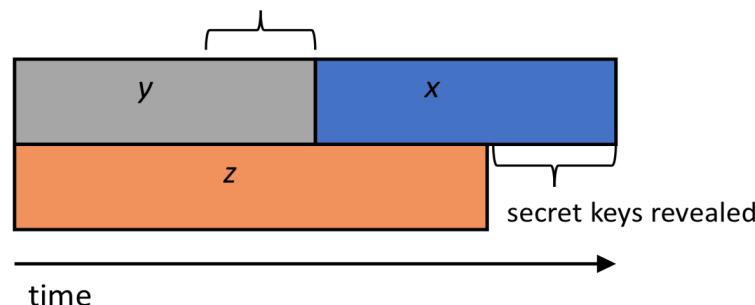
\* **Is the entire network traffic between US and Europe stored?**

## How soon do we need to worry?

- How long does your information need to be secure (*x years*)
- How long to re-tool existing infrastructure with quantum safe solution (*y years*)
- How long until large-scale quantum computer is built (*z years*)

Theorem (Mosca): If  $x + y > z$ , then worry

What do we do here??



BUT WHEN WILL  
THE SKY FALL ?

Copyright: Dustin Moody,  
“NIST Update on Post -  
Quantum Cryptography and  
Lightweight Cryptography”,  
NIST Presentation

# WHY TO SEE IMPACT OF POST-QUANTUM CRYPTOGRAPHY INTO IOT / BLOCKCHAIN / SECURE ELEMENTS WALLETS / JAVA CARD?

csr.csrc.nist.gov/publications/detail/nistir/8240/final

Bookmarks Mozilla Firefox Oracle / Sun Java / Android / O... Java / Kotlin / And... Python JavaScript - Node... Objective-C / Swift » | Other

PUBLICATIONS

**NISTIR 8240**

## Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process

f G+ t

Date Published: January 2019

**Author(s)**

Gorjan Alagic (NIST), Jacob Alperin-Sheriff (NIST), Daniel Apon (NIST), David Cooper (NIST), Quynh Dang (NIST), Carl Miller (NIST), Dustin Moody (NIST), Rene Peralta (NIST), Ray Perlner (NIST), Angela Robinson (NIST), Daniel Smith-Tone (NIST), Yi-Kai Liu (NIST)

**Abstract**

The National Institute of Standards and Technology is in the process of selecting one or more public-key cryptographic algorithms through a public competition-like process. The new public-key cryptography standards will specify one or more additional digital signature, public-key encryption, and key-establishment algorithms to augment FIPS 186-4, Digital Signature Standard (DSS), as well as special publications SP 800-56A Revision 2, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, and SP 800-56B, Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization. It is intended that these algorithms will be capable of protecting sensitive information well into the foreseeable future, including after the advent of quantum computers.

**Second Round Candidates**

BIKE	LEDAcrypt	Rainbow
Classic McEliece	LUOV	ROLLO
CRYSTALS-DILITHIUM	MQDSS	Round5
CRYSTALS-KYBER	NewHope	RQC
FALCON	NTRU	SABER
FrodoKEM	NTRU Prime	SIKE
GeMSS	NTS-KEM	SPHINCS+
HQC	Picnic	Three Bears
LAC	qTESLA	

**TOPICS**

Security and Privacy  
[digital signatures](#); [encryption](#); [key](#)

<https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8240.pdf>

# WHY TO SEE IMPACT OF POST-QUANTUM CRYPTOGRAPHY INTO IOT / BLOCKCHAIN / SECURE ELEMENTS WALLETS / JAVA CARD?



Information Technology Laboratory

COMPUTER SECURITY RESOURCE CENTER

Search CSRC

≡ CSRC MENU



UPDATES

2022

## PQC Standardization Process: Announcing Four Candidates to be Standardized, Plus Fourth Round Candidates

July 05, 2022



### Summary

NIST has completed the third round of the Post-Quantum Cryptography (PQC) standardization process, which selects public-key cryptographic algorithms to protect information through the advent of quantum computers. A total of four candidate algorithms have been [selected for standardization](#), and four additional algorithms will continue into the [fourth round](#).

A detailed description of the decision process and selection rationale is included in NIST Internal Report (NIST IR) 8413, [Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process](#), which is also available on the [NIST PQC webpage](#). Questions may be directed to [pqc-comments@nist.gov](mailto:pqc-comments@nist.gov).

This announcement also discusses plans for a [Fourth PQC Conference](#) and an [upcoming call for additional quantum-resistant digital signature algorithms](#).

### PQC Standardization

After careful consideration during the third round of the [NIST PQC Standardization Process](#), **NIST has identified four candidate algorithms for standardization**. NIST will recommend **two primary algorithms** to be implemented for most use cases: **CRYSTALS-KYBER (key-establishment)** and **CRYSTALS-Dilithium (digital signature)**.

In addition, the signature schemes **FALCON** and **SPHINCS<sup>+</sup>** will also be standardized.

Algorithms to be Standardized

#### Public-Key Encryption/KEMs

CRYSTALS-KYBER

#### Digital Signatures

CRYSTALS-Dilithium

FALCON

SPHINCS<sup>+</sup>

### PQC Fourth Round Candidate Key-Establishment Mechanisms (KEMs)

The following candidate KEM algorithms will advance to the fourth round:

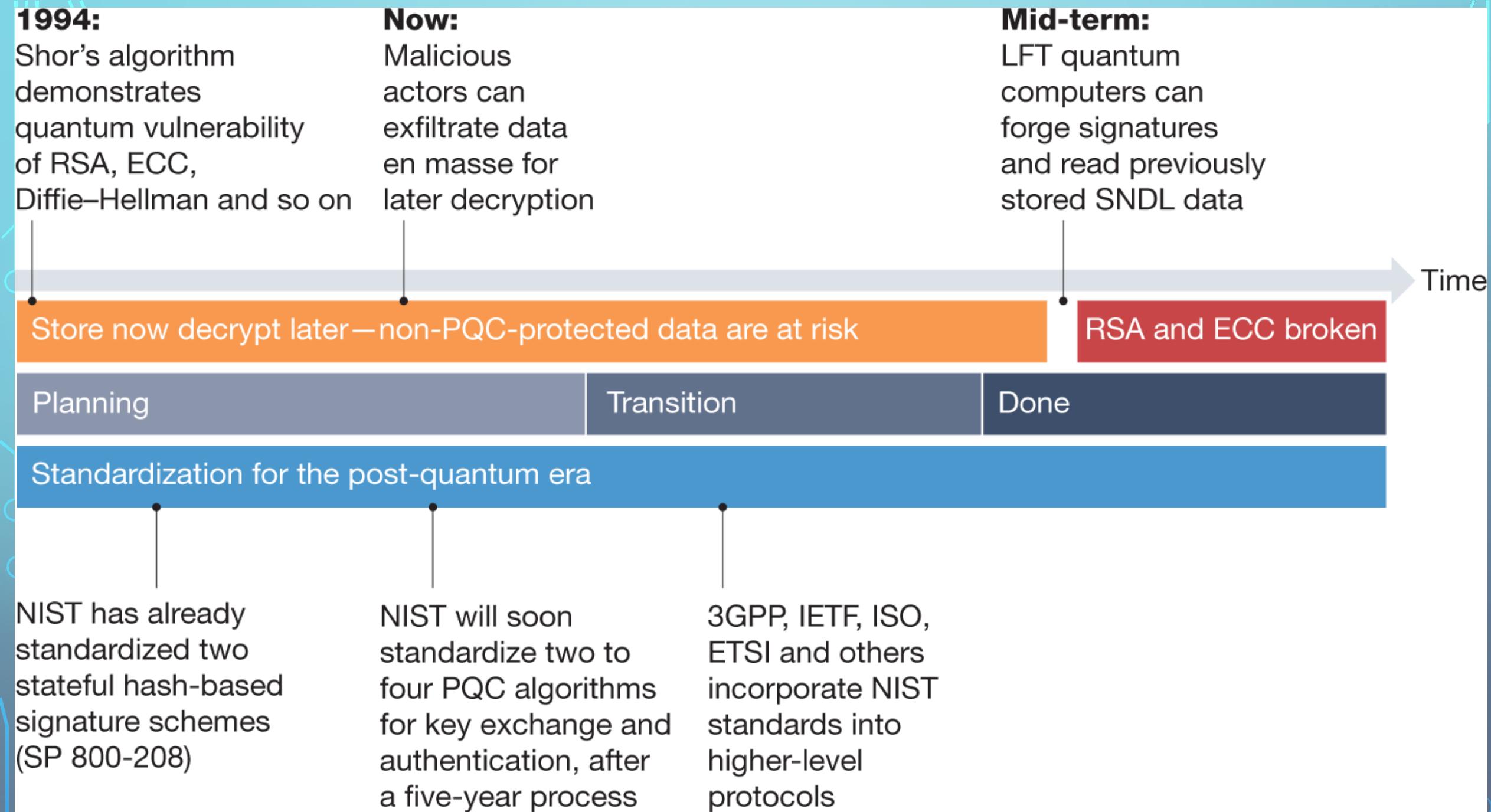
#### Public-Key Encryption/KEMs

BIKE

Classic McEliece

HQC

SIKE



# NIST PQC Standards – Milestones and Timeline

NIST

2016 Criteria and requirements and call for proposals

2017 Received 82 submissions and announced 69 1<sup>st</sup> round candidates

2018 The 1<sup>st</sup> NIST PQC standardization Conference

2019 Announced 26 2<sup>nd</sup> round candidates

The 2<sup>nd</sup> NIST PQC Standardization Conference

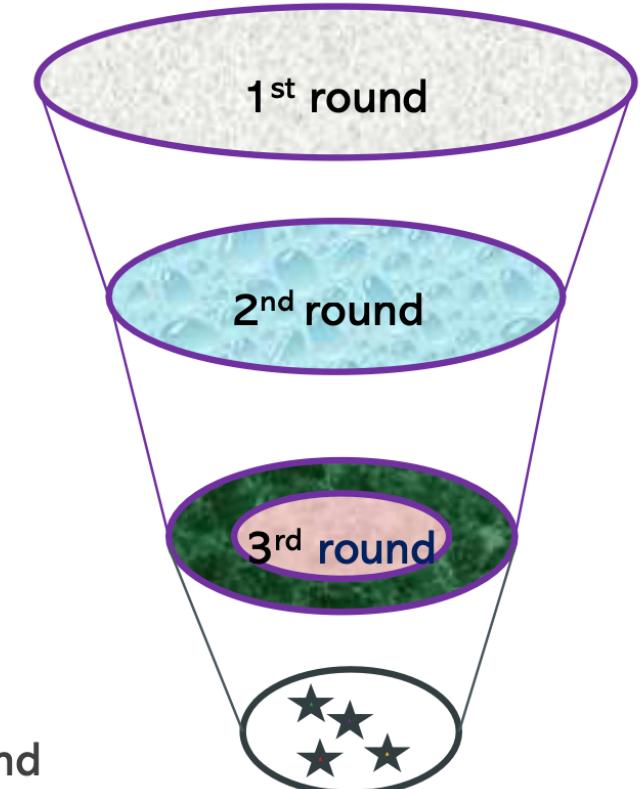
2020 Announced 3rd round 7 finalists and 8 alternate candidate

2021 The 3<sup>rd</sup> NIST PQC Standardization Conference

2022 Announced the 3<sup>rd</sup> round selection and the 4<sup>th</sup> round candidates

→ 2023 Release draft standards for public comments

2024 Publish the 1<sup>st</sup> set of PQC Standards



# PQC Y2023 Status

## Current Status



	Digital Signature	Key Encapsulation (KEM)
Approved	SP 800-208 Stateful Hash-Based Signature (LMS, XMSS)	
Selected Algorithms	<ul style="list-style-type: none"><li>CRYSTALS-Dilithium: Draft FIPS 204 (released)</li><li>SPHINCS+: Draft FIPS 205 (released)</li><li>Falcon: Draft FIPS 206 (under development)</li></ul>	<ul style="list-style-type: none"><li>CRYSTALS-Kyber: Draft FIPS 203 (released)</li></ul>
The 4 <sup>th</sup> Round Candidates		<ul style="list-style-type: none"><li>Classic McEliece</li><li>BIKE</li><li>HQC</li><li><del>SIKE</del></li></ul>
Onramp signatures	<ul style="list-style-type: none"><li>40 candidates are under analysis and evaluation</li></ul>	

# Post Quantum Cryptography

## Agenda

X Post Quantum Cryptography Overview

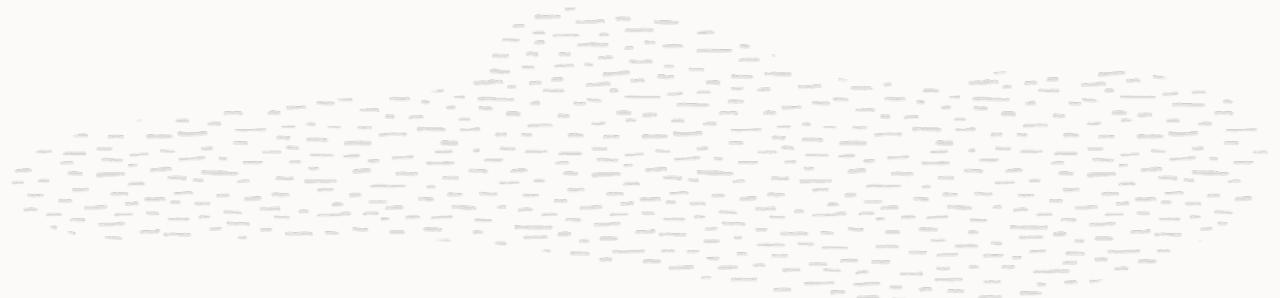
Oracle Post Quantum Cryptography Summit

Post quantum Cryptography and Java Card

Oracle PQC Summit Java Card

Nicolas Ponsini & Sebastian Hans

July, 2023

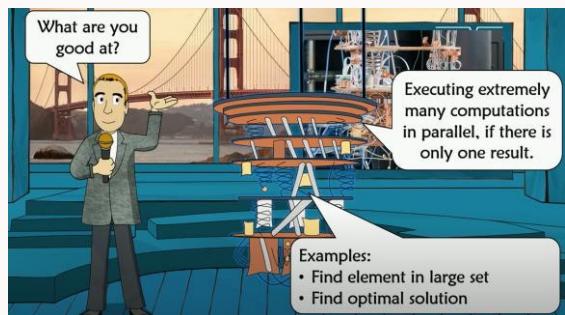
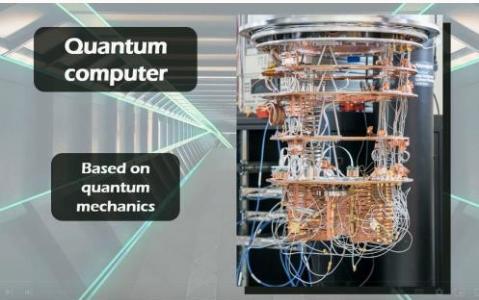


# Post Quantum Cryptography Overview

## Quantum Computing

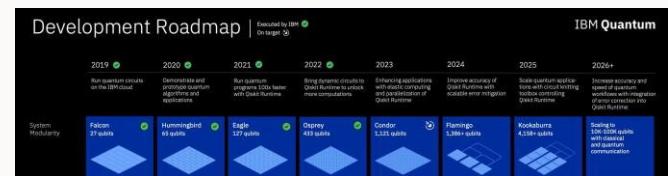
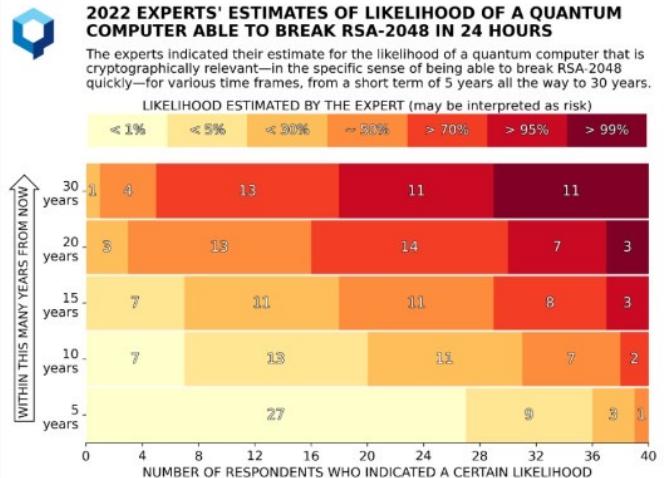
1980s

The basic unit of memory is a quantum bit or qubit, a superposition of 0 and 1. 8 qubits is 0 to 255 at the same time !



## Technology

Copyright © 2023, Oracle and/or its affiliates



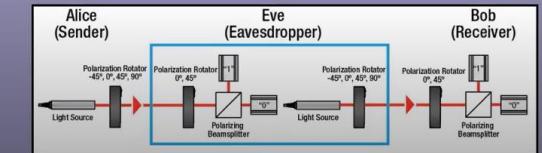
"The question of "if" or "when" there will be quantum computers is no longer in the foreground. **Post-quantum cryptography will become the standard in the long term.**"

## Likelihood

## Quantum Cryptography

Cryptography based on quantum physics

- Quantum cryptography is cryptography based on quantum physics e.g. QKD (Quantum Key Distribution)
- Security like no other form of cryptography by laws of quantum mechanics against quantum and classical attacks



**Problem:** Quantum cryptography is expensive, large, impractical for many uses, requires extreme conditions



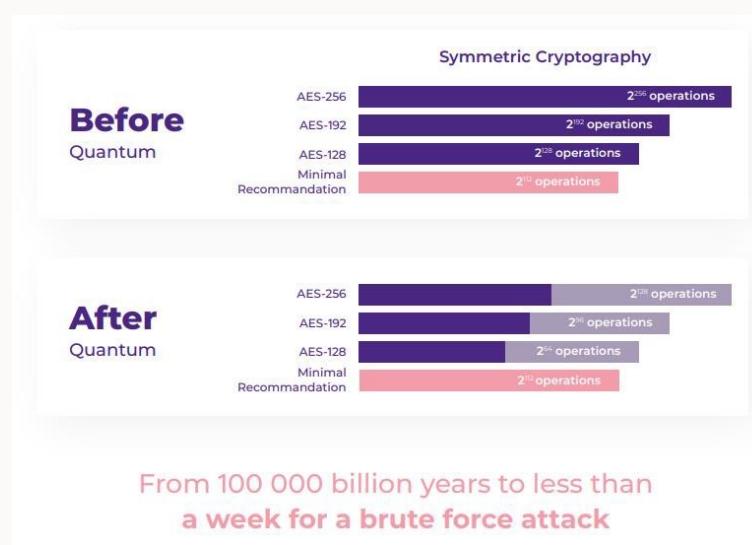
Illustration of the three cooperating ground stations (Graz, Hanover, and Xinglong). Listed are all paths used for key generation and the corresponding key length: Graz-Hanover: University of Science and Technology of China

## Quantum Cryptography

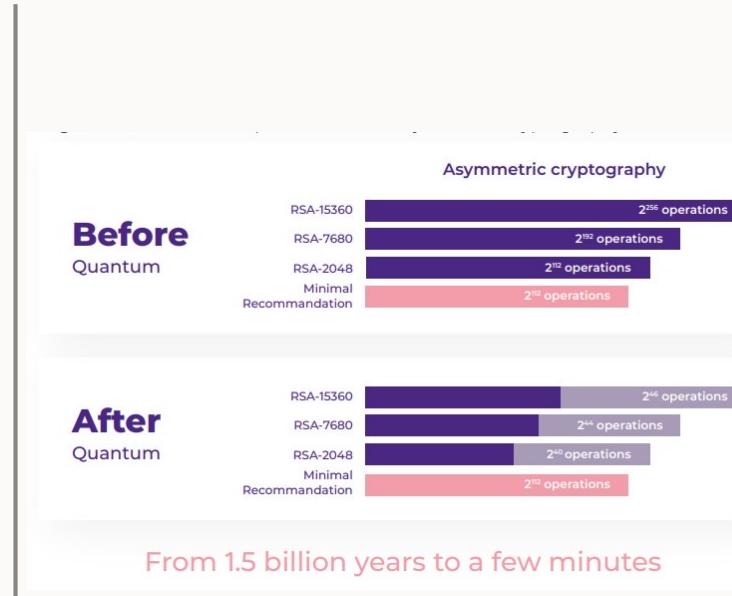


# Post Quantum Cryptography Overview

## Quantum threat and Post-Quantum Cryptography (PQC)



Symmetric  
Cryptography



Asymmetric  
Cryptography

Cryptographic Algorithm	Type	Purpose	Impact from large-scale quantum computer
AES	Symmetric key	Encryption	Larger key sizes needed
SHA-2, SHA-3	-----	Hash functions	Larger output needed
RSA	Public key	Signatures, key establishment	No longer secure
ECDSA, ECDH (Elliptic Curve Cryptography)	Public key	Signatures, key exchange	No longer secure
DSA (Finite Field Cryptography)	Public key	Signatures, key exchange	No longer secure

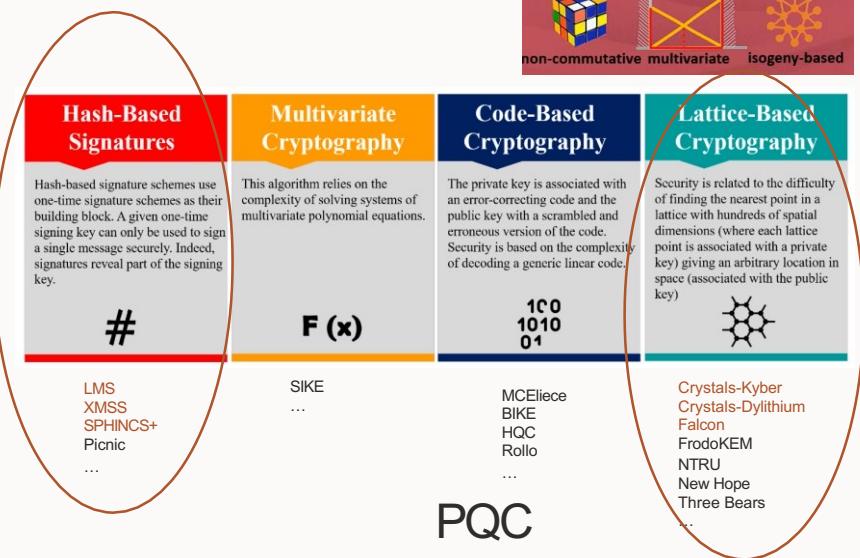
Impact of Shor (Asym) & Grover (Sym) Quantum Computer based algorithms

# Post Quantum Cryptography Overview

## Quantum threat and Post-Quantum Cryptography (PQC)

Post Quantum Cryptography is “Cryptography that runs on current computers and is secure against classical and quantum attacks.”

### Mathematics of PQC:



### Why Switch to PQC Now?

Why we should take the quantum threat seriously and start preparing now

Works on our current computers and hybrid algorithms are available	More secure against quantum and classical attacks	It's difficult to estimate when the quantum threat will occur due to unknowns
Attackers are already storing our encrypted information to decrypt later	Transitioning cryptography takes many years	Some implementations may not be able to switch in time e.g. satellites

- Cryptographic algorithms is only the submerged part of the iceberg
- But what about
  - Communication protocols (TLS, IPsec, SSH, ...)
  - Certificates (X.509)
  - Key management protocols
  - ...
- All use asymmetric blocks



Why Now ?

Not Solely Cryptography



# Post Quantum Cryptography Overview

## PQC gets attention and investment



[NSA Releases Future Quantum-Resistant \(QR\) Algorithm Requirements for National Security Systems](#)



[ANSSI VIEWS ON THE POST-QUANTUM CRYPTOGRAPHY TRANSITION](#)



[Quantum Technologies and Quantum-Safe Cryptography](#)



[Post-Quantum Cryptography: Current state and quantum mitigation](#)

+ others

Security Agencies



- NIST
  - [NIST SP 800-208 published in October 2020 «Recommendation for Stateful Hash-Based Signature Schemes» \(LMS XMSS\)](#)
  - [Selected Algorithms 2022 \(Crystals-Kyber, Crystals-DILITHIUM, FALCON, SPHINCS+\)](#)
- ISO SC27
  - WG2: Cryptography & Security Mechanisms
  - [SC27WG2 SD8 Post-Quantum Cryptography](#)
  - 1488-4 «Digital signatures with appendix —Part 4: Statefulhash-based mechanisms(workingdraft)
  - FrodoKEM ?
- ITU
  - X.509 Hybrid certificate
- IETF
  - X.509, Algorithm Identifiers for Dilithium
  - X.509, Algorithm Identifiers for Kyber
  - hybrid key exchange for TLS 1.3
  - Composite Public and Private Keys For Use In Internet PKI
  - ...
- ETSI
  - WG: CYBER QSC (Quantum Safe)
  - ETSI GR QSC 001 Quantum-safe algorithmic framework Described classes of quantum-safe algorithm
  - ETSI GR QSC 003 Case Studies and Deployment Scenarios Some applications where cryptographic primitives are deployed, e.g. TLS, IoT, Satellite comms
  - ETSI GR QSC 004 Quantum-Safe threat assessment Threat assessment against algorithm vulnerabilities, protocols and industries
  - ETSI TR 103 823 Quantum-Safe Public-Key Encryption and Key Encapsulation
  - ETSI TR 103 616 Quantum-Safe Signatures
  - ETSI TR 103 744 Quantum-Safe Hybrid Key Establishment
- GSMA
  - [Post Quantum Telco Network Impact Assessment](#)
- ...

## [Bouncycastle](#)

## [Openssl](#)

- [For those interested in using the proposed algorithms now, the Open Quantum Safe project has written a provider for OpenSSL 3.x which includes the candidates.](#)

## [Openssh](#)

- The [latest OpenSSH 9.0 release](#) defaults to the NTRU Prime algorithm – a scheme designed to resist brute-force attacks that might be enabled by future quantum computers – while supporting the previous default (X25519 ECDH key exchange) as a backstop. In either case, the algorithms are used to negotiate session keys that protect data in transit.

## [Crystals-Kyber:](#)

- The following lists currently non Oracle approved implementations. Work is in progress to determine best implementation strategy with OCI and Java crypto teams.

- Reference implementation <https://github.com/pq-crystals/kyber>
- (commercial) <https://www.isara.com/products/isara-radate.html>
- liboqs <https://github.com/open-quantum-safe/liboqs/blob/main/docs/algorithms/kem/kyber.md>
- cloudflare <https://github.com/cloudflare/circd>
- https://github.com/PQClean/PQClean
- Java <https://github.com/fisherstevenk/kyberJCE>
- Go <https://git.schwanenlied.me/yawning/kyber> and <https://github.com/SymbolicSoft/kyber-k2so>
- js <https://github.com/antoniotutoveanu/crystals-kyber-javascript>
- python / sage <https://github.com/fvirdia/we-on-rsa-copro>

- ...

## [About JC](#)

- [Practicability of XMSS on Javacards](#)
- [Java Card XMSS](#)



# Post Quantum Cryptography Overview

## PQC Gets all the attention and investment

This is a question of maturity!

- The maturity level of the post-quantum algorithms should not be overestimated.
  - This level is comparable to the maturity level of RSA in the mid 90's
- Maturity at different levels
  - Difficulty of the underlying problem in the classical and quantum computation models
  - Dimensioning
  - Integration of schemes in protocols
  - Secure implementations

**PQC will not become mature with the publication of NIST standards**

### Schneir on Security

#### SIKE Broken

SIKE is one of the new algorithms that NIST [recently added](#) to the post-quantum cryptography competition. It was just [broken](#), really badly.

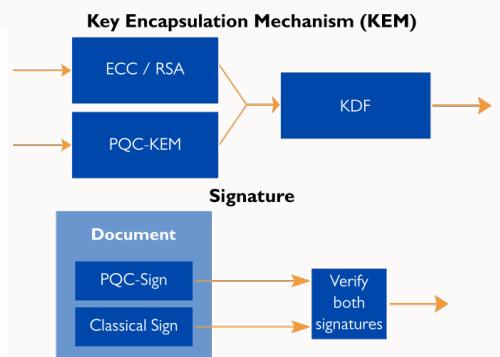
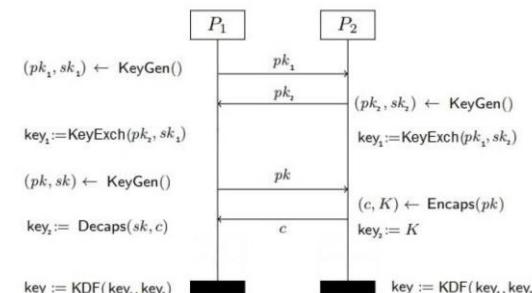
Only Two Standardized PQC algos & For Firmware update only are recognized by NIST, BSI, ANSSI to be used as is:

Hash Based Signatures: LMS & XMSS

Maturity

- Hybrid algorithm Principle = perform both classical and Post Quantum Cryptography for same operation
  - Both ECDH KeyAgreement & PQC Encapsulation
  - Classical & PQC Signatures:

#### Example of hybrid Key Exchange:



- E.g. IETF is working heavily on TLS with Hybrid algorithms

- Hybrid solutions are requested by ANSSI (France) and BSI (Germany)
- Hybrid is encouraged by ENISA (EU) and ETSI (EU)
- Hybrid is discouraged by NSA (US), NCSC (UK) and CSE (Canada)

### Hybrids

- Hybrids are not required for security purposes
- Hybrids add complexity to protocols
  - The incorporation of additional negotiation and error handling
  - The additional code may introduce implementation flaws that could lead to unintentional cryptanalytic breakthroughs
- Methods of hybridization must be common to all parties to a communication
- NSA has confidence that CNSA 2.0 algorithms will sufficiently protect NSS without requiring hybrid solutions

Hybrid algorithms

# Post Quantum Cryptography Overview

## NIST Selected Algorithms

### Crystals-Kyber (KEM)

Table 1: Parameter sets for KYBER

	$n$	$k$	$q$	$\eta_1$	$\eta_2$	$(d_u, d_v)$	$\delta$
KYBER512	256	2	3329	3	2	(10,4)	$2^{-130}$
KYBER768	256	3	3329	2	2	(10,4)	$2^{-164}$
KYBER1024	256	4	3329	2	2	(11,5)	$2^{-174}$

	KYBER512	KYBER768	KYBER1024
NIST Security level	1	3	5

KYBER512	
Sizes (in Bytes)	
sk:	1632 (or 32)
pk:	800
ct:	768

KYBER768		KYBER1024	
Sizes (in Bytes)		Sizes (in Bytes)	
sk:	2400 (or 32)	sk:	3168 (or 32)
pk:	1184	pk:	1568
ct:	1088	ct:	1568

### Crystals-Dilithium (Signature)

Table 2: Parameters of Dilithium

NIST Security Level	2	3	5
Parameters			
$q$ [modulus]	8380417	8380417	8380417
$d$ [dropped bits from $t$ ]	13	13	13
$\tau$ [<# of ±1's in $c$ ]	39	49	60
challenge entropy [ $\log \binom{2^m}{\tau} + \tau$ ]	192	225	257
$\gamma_1$ [ $y$ coefficient range]	$2^{17}$	$2^{19}$	$2^{19}$
$\gamma_2$ [low-order rounding range]	$(q - 1)/88$	$(q - 1)/32$	$(q - 1)/32$
$(k, \ell)$ [dimensions of $A$ ]	(4,4)	(6,5)	(8,7)
$\eta$ [secret key range]	2	4	2
$\beta$ [ $\tau \cdot \eta$ ]	78	196	120
$\omega$ [max. # of 1's in the hint $h$ ]	80	55	75
Repetitions (from Eq. (5))	4.25	5.1	3.85

NIST Security Level	2	3	5
Output Size			
public key size (bytes)	1312	1952	2592
signature size (bytes)	2420	3293	4595

### FALCON (Signature)

#### 3.13 Recommended Parameters

We specify two sets of parameters that address security levels I and V as defined by NIST [NIS16, Section 4.A.5]. These can be found in Table 3.3. Core-SVP hardness is given for the best known classical (C) and quantum (Q) algorithms.

	FALCON-512	FALCON-1024
Target NIST Level	I	V
Ring degree $n$	512	1024
Modulus $q$	12289	
Standard deviation $\sigma$	165.736.617.183	168.388.571.447
$\sigma_{\min}$	1.277.833.697	1.298.280.334
$\sigma_{\max}$		1.8205
Max. signature square norm $ \beta^2 $	34.034.726	70.265.242
Public key bytelength	897	1793
Signature bytelength shytelen	666	1280
Key-recovery:	$\begin{cases} BKZ blocksize B (2.3) \\ \text{Core-SVP hardness (C)} \\ \text{Core-SVP hardness (Q)} \end{cases}$	$\begin{cases} 458 \\ 133 \\ 121 \\ 248 \end{cases}$
Forgery:	$\begin{cases} BKZ blocksize B (2.4) \\ \text{Core-SVP hardness (C)} \\ \text{Core-SVP hardness (Q)} \end{cases}$	$\begin{cases} 411 \\ 120 \\ 108 \\ 252 \end{cases}$

Table 3.3: FALCON parameter sets.

### SPHINCS+ (Signature)

$n$	$h$	$d$	$\log(t)$	$k$	$w$	bitsec	sec level	sig bytes
SPHINCS <sup>+</sup> -128s	16	63	7	12	14	16	133	1 7856
SPHINCS <sup>+</sup> -128f	16	66	22	6	33	16	128	1 17088
SPHINCS <sup>+</sup> -192s	24	63	7	14	17	16	193	3 16224
SPHINCS <sup>+</sup> -192f	24	66	22	8	33	16	194	3 35664
SPHINCS <sup>+</sup> -256s	32	64	8	14	22	16	255	5 29792
SPHINCS <sup>+</sup> -256f	32	68	17	9	35	16	255	5 49856

Table 2: Key and signature sizes

SK	PK	Sig
Size	$4n$	$2n$

# Post Quantum Cryptography Overview

## XMSS NIST SP800 208

Name	Functions	n	w	len	h
REQUIRED:					
XMSS-SHA2_10_256	SHA2-256	32	16	67	10
XMSS-SHA2_16_256	SHA2-256	32	16	67	16
XMSS-SHA2_20_256	SHA2-256	32	16	67	20
OPTIONAL:					
XMSS-SHA2_10_512	SHA2-512	64	16	131	10
XMSS-SHA2_16_512	SHA2-512	64	16	131	16
XMSS-SHA2_20_512	SHA2-512	64	16	131	20
XMSS-SHAKE_10_256	SHAKE128	32	16	67	10
XMSS-SHAKE_16_256	SHAKE128	32	16	67	16
XMSS-SHAKE_20_256	SHAKE128	32	16	67	20
XMSS-SHAKE_10_512	SHAKE256	64	16	131	10
XMSS-SHAKE_16_512	SHAKE256	64	16	131	16
XMSS-SHAKE_20_512	SHAKE256	64	16	131	20

Table 2

The XDR formats for XMSS are listed in [Appendix B](#).

Scheme	h	k	w	Timings (ms)			Sizes (byte)			b
				KeyGen	Sign	Verify	Secret key	Public key	Signature	
XMSS <sup>+</sup>	16	2	4	5,600	106	25	3,760	544	3,476	96 (85)
XMSS <sup>+</sup>	16	2	8	5,800	105	21	3,376	512	2,436	95 (81)
XMSS <sup>+</sup>	16	2	16	6,700	118	22	3,200	512	1,892	93 (71)
XMSS <sup>+</sup>	16	2	32	10,500	173	28	3,056	544 (480)	1,588	92 (54)
XMSS <sup>+</sup>	20	4	4	22,200	106	25	4,303	608	3,540	92 (81)
XMSS <sup>+</sup>	20	4	8	22,800	105	21	3,920	576	2,500	91 (77)
XMSS <sup>+</sup>	20	4	16	28,300	124	22	3,744	576	1,956	89 (67)
XMSS <sup>+</sup>	20	4	32	41,500	176	28	3,600	544	1,652	88 (50)
XMSS	10	4	4	14,600	86	22	1,680	608	2,292	103 (92)
XMSS	10	4	16	18,800	100	17	1,648	576	1,236	100 (78)
XMSS	16	4	4	925,400	134	23	2,448	800	2,388	97 (86)
XMSS	16	4	16	1,199,100	159	18	2,416	768	1,332	94 (72)

### Signature

$$\text{XMSS-SHA2\_10\_256} \rightarrow (n = 32; \text{len} = 67, h = 10) = 4 + n + (\text{len} + h) * n \\ = 2500 \text{ bytes for signature length}$$

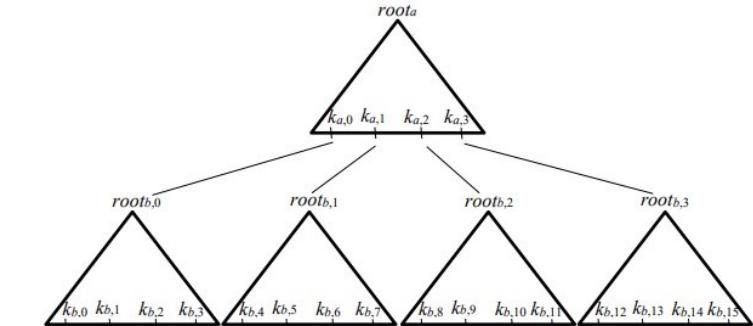
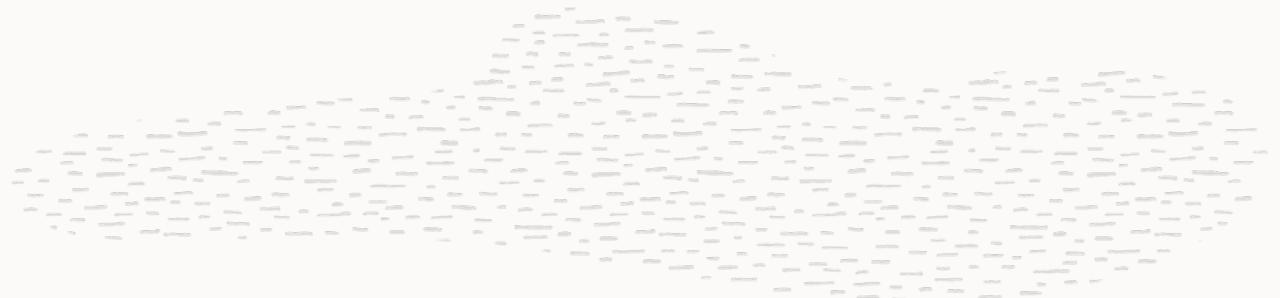


Figure 5: A two-Level Merkle tree

# Agenda

---

- Post Quantum Cryptography Overview
- X Oracle Post Quantum Cryptography Summit
- Post quantum Cryptography and Java Card



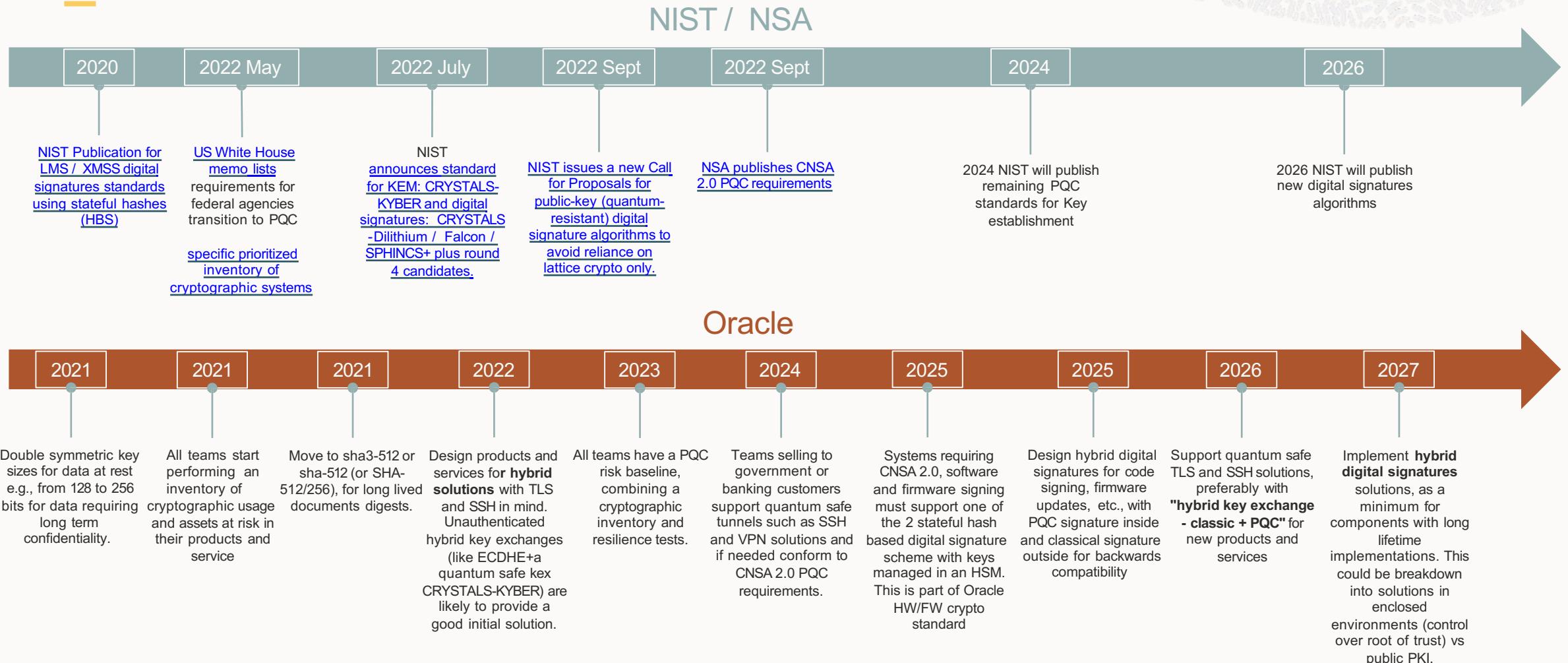
# Oracle PQC Summit

---

- Organized by GPS June 29<sup>th</sup>
- Complete overview on Oracle PQC activities
  - Technical Briefing
  - US Government Impact
  - Trade Compliance Issues
  - Legal Issues
  - Hardware and Code Signing
  - Java Briefing
  - OCI Briefing
  - Database Briefing
  - Oracle Health (Cerner) Briefing

# Oracle PQC Summit

## Take Away - Timeline



# Oracle PQC Summit

Take Away – GPS Team

Industry consensus around  
PQC ephemeral keys agreement  
with classic  
RSA/ECDSA authenticity  
as first step

Many other use cases not requiring PKI changes  
for now...



TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256

Key exchange    Authentication /signature

Encryption with key size and mode    Hash/MAC

"TLS\_KYBER\_ECDSA\_WITH\_AES\_256\_GCM\_SHA512"

New PQC algo; No PKI change

Resistant to passive attacks

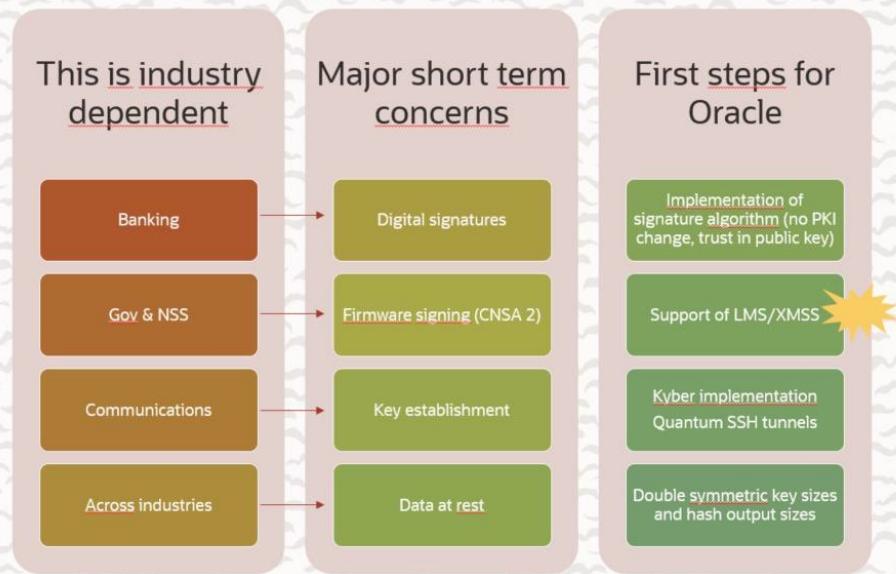
Valid for “current” “Store Now Decrypt Later Attack.”

Will not be valid once the authentication of the KYBER key based on ECDSA will not be reliable anymore because of QC defeating ECDSA

# Oracle PQC Summit

## Take Away – GPS Team

### Prep work Identify first PQC transition steps



#### Collaborative work

- Documenting quantum attacks and algorithms analysis (Crypto Review Board )
- LMS/XMSS algorithms analysis (CRB )
- Patents discussions (Legal, CRB)
- Export restrictions checks (Global Trade)
- Review of regulation for NSS Systems/DoD IL5-6/CNSA (public sector groups / OCI Hw security / CRB )
- NIAP working group participation for CNSA (GPS / seceval)
- publication of a new HW/FW crypto standard (OCI Hardware security / OHD/ GPS/CRB )
- PQC Java implementations (Java crypto team)
- Copyright/licenses + PLS/3PLTA (Legal / Java crypto team / CorpArch)
- Enhancements to Corp Code signing service (GPS / Key Mgt governance board / Corp Code Signing service)
- OCI KMS roadmap changes (KMS / Corp Code Signing service)
- Reqs & request to external HSM supplier to support LMS (OCI Crypto / OCI KMS / GPS / CRB)

Transition is a collective work

# Oracle PQC Summit

## Take Away – Hardware Team

### CNSA 2.0 Timelines

- Software and Firmware Signing:
  - Begin transition immediately. Support and prefer CNSA 2.0 by 2025. Use CNSA 2.0 exclusively by 2030.
- Web browsers/servers and cloud services:
  - Support and prefer CNSA 2.0 by 2025. Use CNSA 2.0 exclusively by 2033.
- [https://media.defense.gov/2022/sep/07/2003071834/-1/-1/0/csa\\_cnsa\\_2.0\\_algorithms.pdf](https://media.defense.gov/2022/sep/07/2003071834/-1/-1/0/csa_cnsa_2.0_algorithms.pdf)

### CNSA 2.0 Timeline

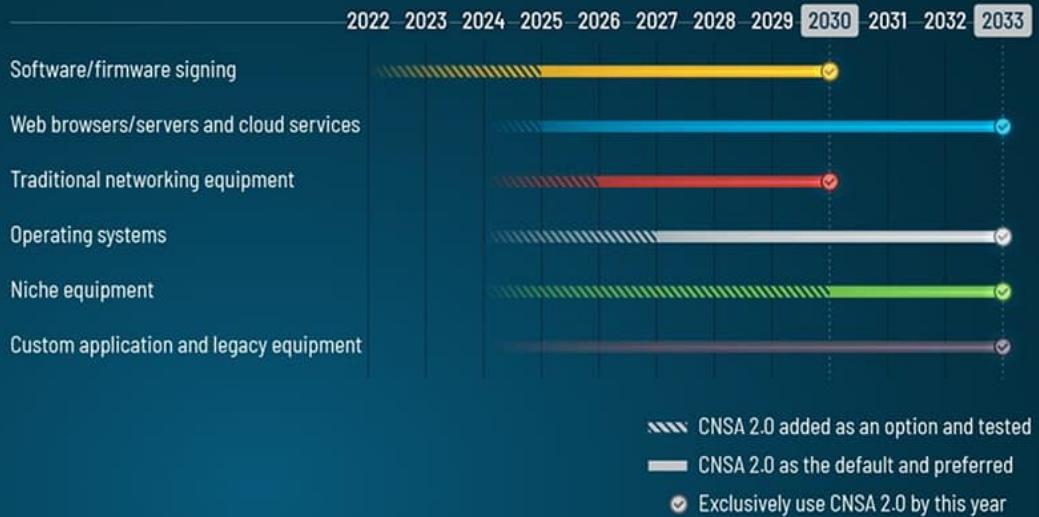


Table IV: CNSA 2.0 algorithms

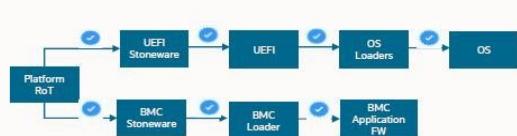
Algorithm	Function	Specification	Parameters
Advanced Encryption Standard (AES)	Symmetric block cipher for information protection	<a href="#">FIPS PUB 197</a>	Use 256-bit keys for all classification levels.
CRYSTALS-Kyber	Asymmetric algorithm for key establishment	TBD	Use Level V parameters for all classification levels.
CRYSTALS-Dilithium	Asymmetric algorithm for digital signatures	TBD	Use Level V parameters for all classification levels.
Secure Hash Algorithm (SHA)	Algorithm for computing a condensed representation of information	<a href="#">FIPS PUB 180-4</a>	Use SHA-384 or SHA-512 for all classification levels.
Leighton-Micali Signature (LMS)	Asymmetric algorithm for digitally signing firmware and software	<a href="#">NIST SP 800-208</a>	All parameters approved for all classification levels. SHA256/192 recommended.
Xtended Merkle Signature Scheme (XMSS)	Asymmetric algorithm for digitally signing firmware and software	<a href="#">NIST SP 800-208</a>	All parameters approved for all classification levels.

# Oracle PQC Summit Take Away – Hardware Team

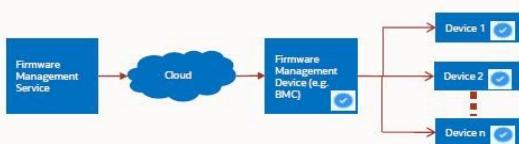
## Secure Boot and Secure FW Update Overview

Digital Signed Firmware

Secure Boot Flow (verify and execute)



Secure Firmware Update Flow (verify and update)



7 Copyright © 2023, Oracle and/or its affiliates | Confidential: internal | Ver 1.1

## Secure Boot and Secure Firmware Update PQC Migration Proposal

Digital Signed Firmware Signing Scheme

Industry prefers hybrid signatures:

- ClassicSign(Firmware + PQCSignature(Firmware))

National Security Systems(NSS) disallows hybrid signatures

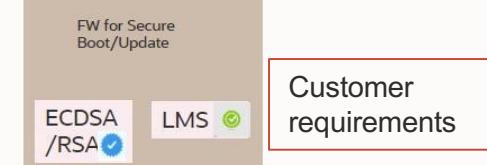
Proposal:

- Sign with Classical and PQC side by side and ensure both signatures are verified.

Classical Algorithms



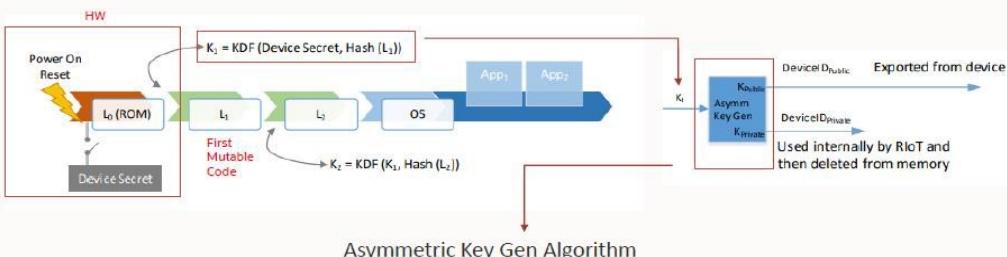
Classical and PQC side by side



## Device Identity PQC Migration Proposal

DICE/RIoT Style Key Generation and Signing Scheme

The Device Identity (DeviceID) is an asymmetric key pair that serves as a long-term identifier for a device, device authentication via a certificate chain rooted in CA or a device self signed certificate or a device bare key.



Classical Algorithms



PQC Algorithms



## Call To Action

- Oracle to work with HSM vendor to firm up LMS code signing requirements
- Oracle to work with HSM vendor to support Crystal-Dilithium code signing for device identity endorsement
- Oracle to update code signing infrastructure to support PQC
- Oracle to develop LMS and Crystal-Dilithium code signing capabilities
- OCI/OHD to collaborate on a plan for hardware used in OCI to support PQC(in progress)
- OCI/OHD to communicate with hardware partners on Oracle's PQC requirements(in planning)

# Oracle PQC Summit

## Take Away – Java Team

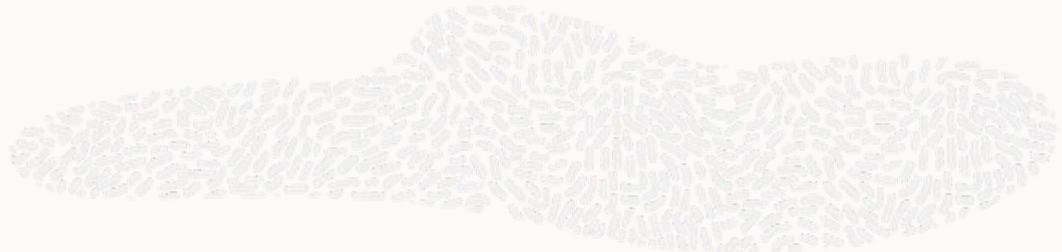


### What JDK Aims to Deliver for PQC

- JDK (Java Development Kit) will add PQC support in our OpenJDK & OracleJDK releases
- JDK only includes compliant implementations of specifications that have been standardized
  - Must wait till NIST completes its standardization of candidates before we can add to our GA releases
  - EA / Preview offerings will be problematic
    - Public Java API is standardized in feature releases and cannot be changed easily from one release to another (see later about MR releases)
    - If we add our implementation before standardization, we run the risk that there may be further changes later on
    - Even if we are willing to update our implementation iteratively until the final version, we will run into regression and compatibility problems for our customers
  - We will add crypto (JCE provider) support first, TLS support to follow

# Oracle PQC Summit

## Take Away – Java Team



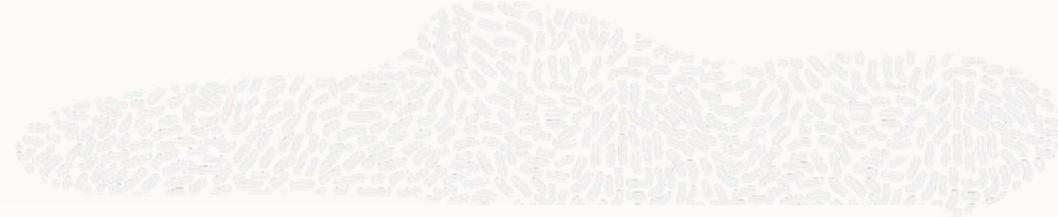
### What JDK Aims to Deliver for PQC

- Will add mainline support first, backport as necessary or appropriate
- Will need to work out feasibility of backport to earlier releases (e.g. 8u)
  - Older code base may not have the required libraries or APIs to support the new PQC implementations, or the backport effort has significant regression risk
  - It took 2 years before TLS 1.3 support was backported to OracleJDK 8u261
- We follow the [Java Community Process](#) to standardize our technical specifications
  - Each feature release has a JSR (Java Specification Request) that covers all of the Java specifications in that feature release
  - To amend the specification, we need to file a Maintenance Review (MR) and follow the JCP processes, which includes community review and updated specifications, and may include updated TCK tests, and updated Reference Implementations.
- PQC support may need new APIs, requiring a Maintenance Release for older trains (8u, 11u, 17u)
  - May need to backport sizable additional code
- All add up to huge effort and significant risks



# Oracle PQC Summit

## Take Away – Java Team

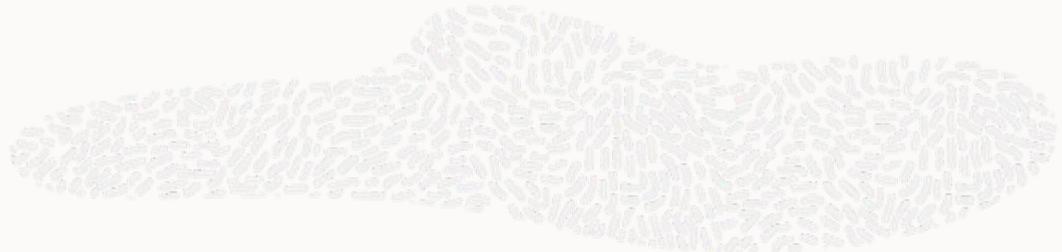


### What JDK Aims to Deliver for CNSA 2.0

- NSA [advisory](#) on future requirements for QR (Quantum-Resistant) algorithms for NSS
- Algorithm requirements include:
  - Algorithms for software/firmware signing
    - LMS (Leighton-Macали Hash-based Signature), [RFC 8554](#)
      - We providing signature verification only using the JCE APIs since HSMs are required for signing [ [a JDK 21 feature](#) ]
      - We can proceed with this since RFC 8554 has been out since 2019
    - XMSS (eXTended Merkle Signature Scheme), [RFC 8391](#)
      - Has performance challenges, requiring noncompliant changes - [not planned at this time](#)
  - Symmetric Key Algorithms
    - AES: use 256-bit keys for all classification levels [ [Supported in all releases, default since JDK19](#) ]
    - SHA: use SHA-384 or SHA512 for all classification levels [ [Supported in all release, default since JDK19](#) ]
      - These default sizes may be backported to earlier releases via the [Java Crypto Roadmap program](#)
  - General-use quantum-resistant public-key algorithms
    - CRYSTALS-Kyber, asymmetric algorithm for key establishment [ [in progress](#) ]
    - CRYSTALS-Dilithium, asymmetric algorithm for digital signatures [ [in progress](#) ]

# Oracle PQC Summit

## Take Away – Java Team



### Status on CNSA 2.0 support in JDK

- Code to support LMS was integrated on June 1<sup>st</sup>, 2023 for JDK 21
  - JDK 21 will GA on Sept 19<sup>th</sup>, 2023
- Performance benchmark (signature verification only) in progress
  - Our implementation is significantly better (at least 30%) than Bouncy Castle
- Working with OCI Crypto Team on interoperability testing
  - Marvell HSM
- Jarsigner support for LMS is being planned for JDK 22
- CRYSTALS-Kyber & CRYSTALS-Dilithium
  - Part of the PQC support, see next slide
- Full compliance to CNSA 2.0 is dependent on NIST standardizes Kyber & Dilithium

# Oracle PQC Summit

## Take Away – Java Team



### Status on PQC support in JDK

- CRYSTALS-Kyber

- Kyber RFC draft: <https://datatracker.ietf.org/doc/draft-cfrg-schwabe-kyber/>
- Newest version: 3/31/2023
- Development in progress
  - Challenges
    - Understanding the NTT (Number Theoretic Transform) and its inverse was the biggest challenge
    - Initial implementation is complete
- What's next
  - Working on intrinsic functions for the target architectures (Aarch64, x86-64 with AVX2 and AVX512)
    - Performance will be key to adoption
  - Additional work to integrate into the JCE framework, and in the TLS layer
    - Will need to modify JSSE (Java Secure Socket Extension), JDK's TLS implementation, be "KEM aware"
  - Waiting for standardization by NIST

# Oracle PQC Summit

## Take Away – OCI Team (Cloud)

3

### Where do we need quantum resistance?

Area	Details
TLS	<ul style="list-style-type: none"><li>• Control planes</li><li>• Data planes</li><li>• Intra and inter-service communication</li><li>• Load balancer, API gateway, WAF, and other TLS terminating/initiating services</li></ul>
SSH	<ul style="list-style-type: none"><li>• Operator access</li><li>• Bastions for customers</li></ul>
Signatures	<ul style="list-style-type: none"><li>• Software and firmware signing for our own <b>deployments</b></li></ul>
X.509 certificates	<ul style="list-style-type: none"><li>• Internal PKI</li><li>• Public certificate management for customers</li></ul>
Key management systems support	<ul style="list-style-type: none"><li>• Software (wrapped key hierarchies require a mix of software and hardware support)</li><li>• Hardware Security Modules (HSMs)</li></ul>
Identity systems	
Operating system images	<ul style="list-style-type: none"><li>• In the supported software of the operating system images we offer to customers</li></ul>

# Oracle PQC Summit

## Take Away – OCI Team (Cloud)

5

### What do we intend to use?

Area	Details
TLS	<ul style="list-style-type: none"><li>• We prefer hybrid solutions to minimize risk to customers by default</li><li>• IETF is currently defining new cipher suites in the TLS WG, see <a href="#">draft-ietf-tls-hybrid-design-06</a> (latest version as of May 2023)</li><li>• The current draft defined code points are for hybridized Kyber (based on round 3 NIST parameters), but the CFRG will ultimately provide recommendations based on NIST and other standards.</li><li>• If FrodoKEM, Classic McEliece, or others get code points defined by IETF/IRTF as well as implemented in core cryptography libraries (e.g. in the TLS stack of OpenSSL, Windows, macOS, iOS, Android, etc) we will investigate supporting them as well.</li></ul>
SSH	<ul style="list-style-type: none"><li>• OpenSSH implemented a hybrid X25519-Streamlined NTRU Prime as of version 9.0.</li><li>• As standards are finalized, we hope to see additional cipher suites added to OpenSSH that use more broadly accepted constructions.</li></ul>
Signatures	<ul style="list-style-type: none"><li>• Dilithium (and possibly SPHINCS+) for signatures, but customer and regulatory demand will drive support timelines.</li><li>• For firmware likely LMS/XMSS to meet CNSA 2.0 timelines.</li><li>• LMS/XMSS are stateful, which makes life more difficult for a distributed system.</li></ul>

# Oracle PQC Summit

## Take Away – Database Team

### Overall Strategy

- All features update to default minimum AES256 and SHA256
- Migrated to OpenSSL for DB crypto to achieve crypto agility
- Remain agile and provide capability to move to newer algorithms and larger key size easily
  - Encryption on storage: easy switch between algorithms/key sizes
  - **Encryption in transition**
    - Update to NIST approved PQC asymmetric ciphers along with OpenSSL
    - Inject pre-shared secret to achieve quantum safe

Rely on Openssl

Waiting for it, mitigation with Pre shared secrets for TLS protected by passwords

# Oracle PQC Summit

## Take Away – Database Team

### DB Features using Crypto

- Dictionary credentials
- Oracle wallet
- Column/tablespace encryption
- Secure backup
- Data pump encryption
- Secure files
- Golden gate extract file
- ACFS file system encryption
- Redo log encryption
- DBMS\_CRYPTO (symmetric)
- Kerberos

With  
AES256  
support

- Asymmetric algorithms (RSA, ECDSA)
  - TLS, Block Chain
- Key exchange DH
  - Native network encryption
- DBMS\_CRYPTO (asymmetric for signing)
- Other legacy use cases
  - HTTP digest authentication
  - DB verifiers SHA1 for compatibility

# Oracle PQC Summit

## Take Away – Database Team

### Challenges

- Changing default configurations across all features
- Performance impact when using larger key size
- Storage changes
- Backward compatibility
- Encrypted data migration

Removal of Native Network Encryption initially planned (considered less secure than TLS) but still requested by customers because flexible i.e no certificates involved.  
The Key exchange weakness is mitigated with Key Vault protected by password + mix with Kerberos authentication  
New Version is currently designed.

### Encrypted Data Migration

- Wallet conversion to encrypt with AES256 as default
- Re-encrypt existing data to switch to larger key size
  - Column encryption
  - Tablespace encryption
  - Secure files
  - Redo log

### 23c Updates

- 23c Changes
- GoldenGate encryption
  - Column encryption
  - LOB/securefile encryption
  - PDB export keys
  - Tablespace encryption
  - Dictionary credentials
  - Redo log encryption
  - RMAN encryption
  - Data pump encryption

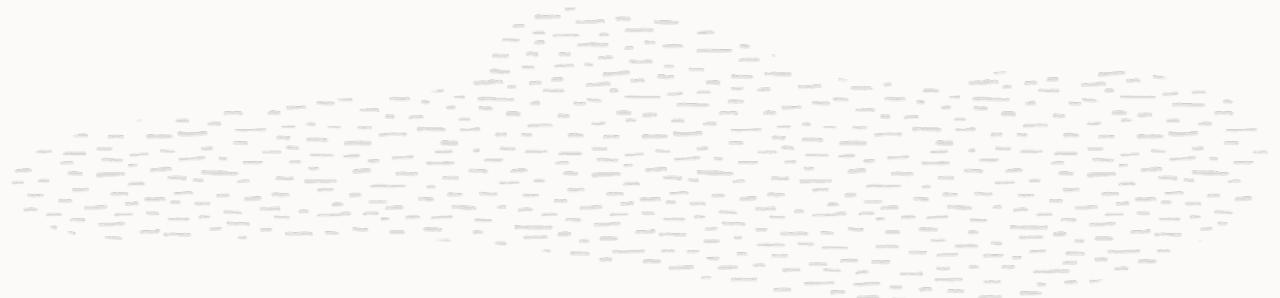
Continuous effort:

- Pluggability/switch of crypto algorithms across various features for post quantum crypto

# Agenda

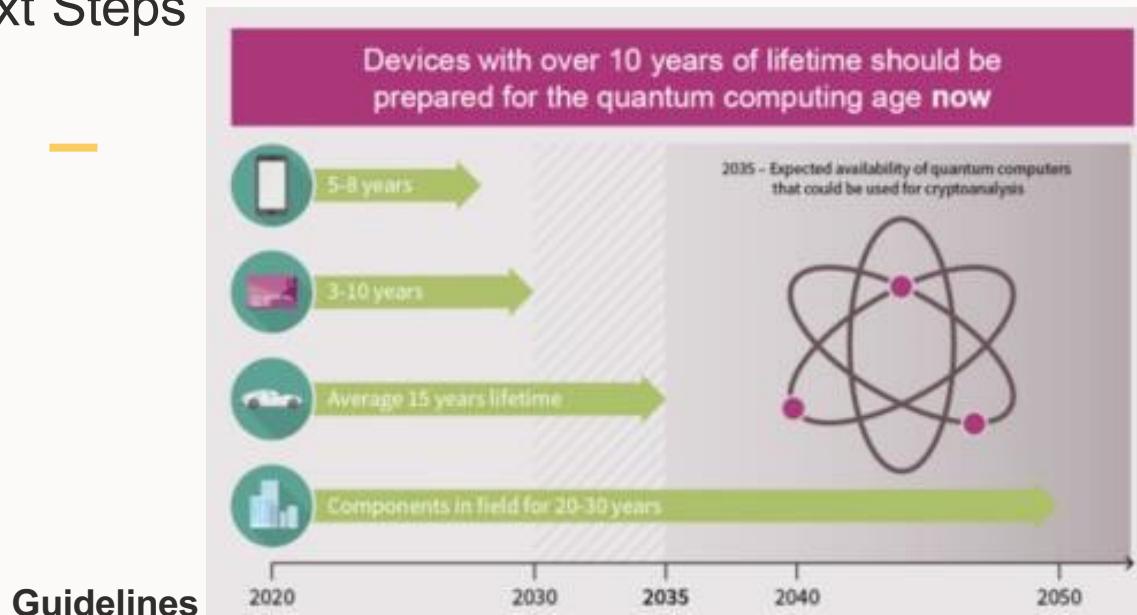
---

- Post Quantum Cryptography Overview
- Oracle Post Quantum Cryptography Summit
- X Post quantum Cryptography and Java Card



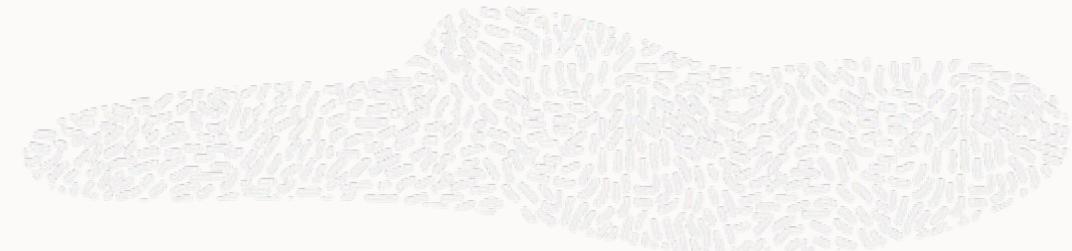
# Post Quantum Cryptography and Java Card

## Next Steps



### Guidelines

- (As usual) take into account resource constraints of an SE (NVM, RAM, processing)
- (As others) Focus on what “really” exists and/or foreseen as the “most likely future” !
  - Crystals-Kyber for Key Encapsulation Mechanism (KEM)
  - Crystals-Dilithium for PQC Signature
  - Hashed Based Signature LMS & XMSS
- “Crypto Agility” for what it could mean: have an API extendable, versatile, adaptable, ...
- PoC with RI/JoT with available/authorized implementation(s)

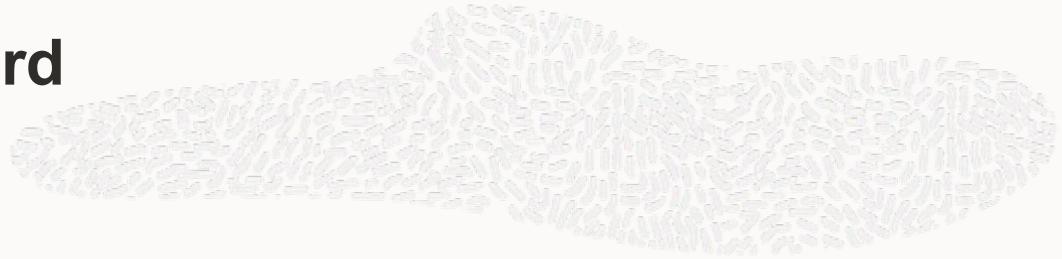


### Java Card API:

- ✓ Sym crypto: AES 256
- ✓ Hash crypto: SHA(3)-384 & SHA(3)-512
- ✗ PQC Key Management
  - Import Export Generation
- ✗ Add Support to KEM or Encapsulation/Decapsulation
  - Both Classic and PQC
- ✗ PQC Signature
  - Hash Based specificities
- ✗ Hybrid algorithms
  - Shared secrets, KDF, TLS ...
- ✗ Overall alignment with Standards
  - For all of the above
  - And more E.g. GP and new SCP to come

# Post Quantum Cryptography and Java Card

## Next Steps Challenges



### PQC Key Management

- Many Parameters
  - Likely Named Curves Like management where all parameters are known by the platform at key creation/generation
- Long Lengths
  - Adapt Import / Export mechanisms

### PQC Signatures

- Stateful Management
- Additional Parameters
  - Merkle tree
- Long Lengths
  - Adapt sign/verify mechanisms

### Hybrid algorithms

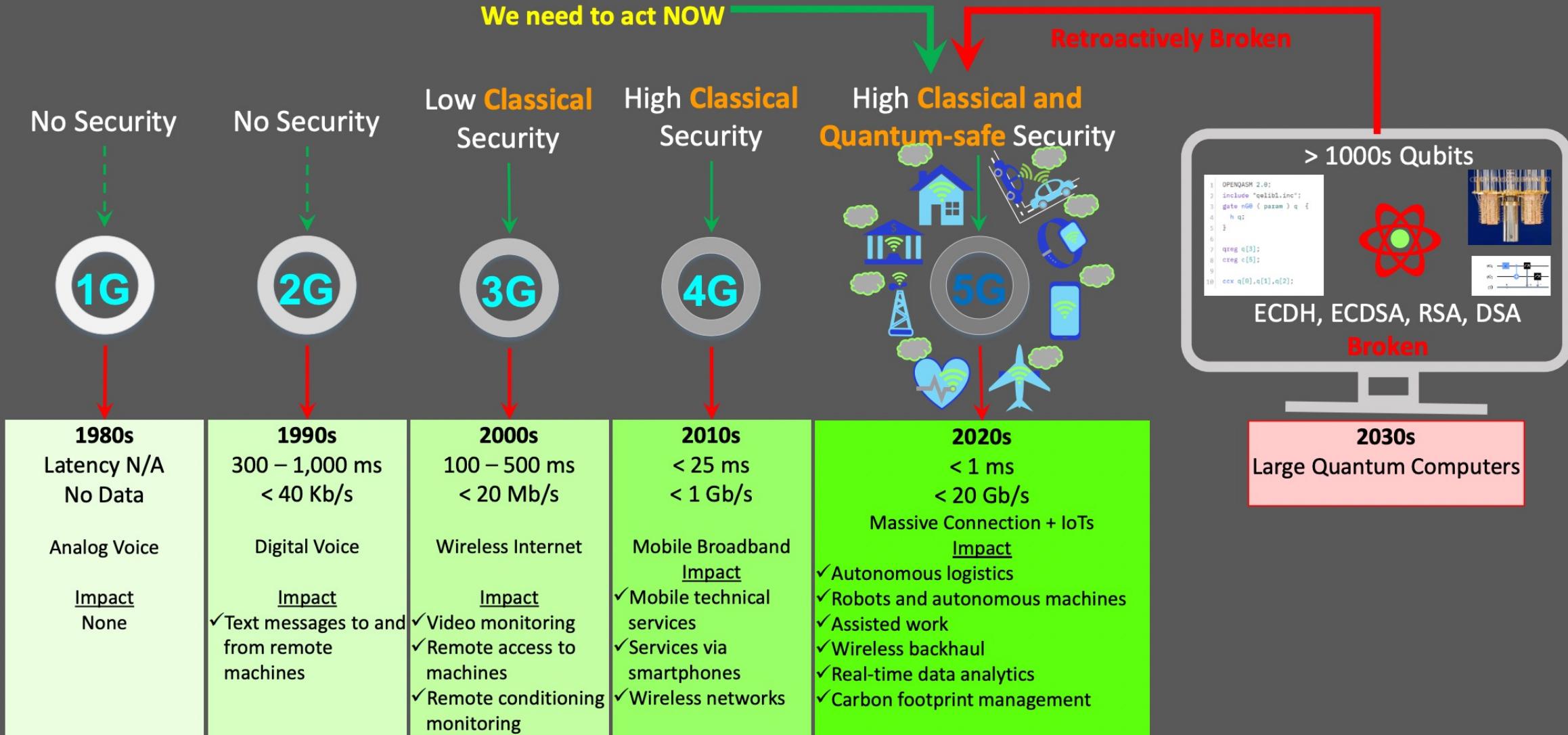
- KEM
  - KEM with and without pre-shared keys
  - Key Combiner of the pre and post Quantum Algorithms
    - CAT, KDF, ...
- Signature
  - On principle just the Signature concatenations
  - Sometimes more: Signature Nesting
- Protocols
  - TLS Copyright © 2023, Oracle and/or its affiliates

- KEM is not something new E.g RSA-KEM  
(<https://www.rfc-editor.org/rfc/rfc5990.html>)
  - Input = (n,e) and K (key data to share)
  - z = RandomInteger (0, n-1)
  - c =  $z^e \text{ mod } n$
  - KEK = KDF (Z, kekLen) where KDF = Key Deriv. Fcn.
  - WK = Wrap (KEK, K)
  - EK = C || WK → passed to the owner of the private key that will decrypt C and generate KEK and unwrap WK to retrieve K
- Principle: Use the Encryption/Decryption capability of an asymmetric Algorithms to share a secret
  - Feature Needed by numerous protocols
  - Mostly “unused” because Key Agreement mechanisms like DH & ECDH were preferred
  - But PQC is coming with KEM only, not Key Agreement

### KEM (Key Exchange Methods)



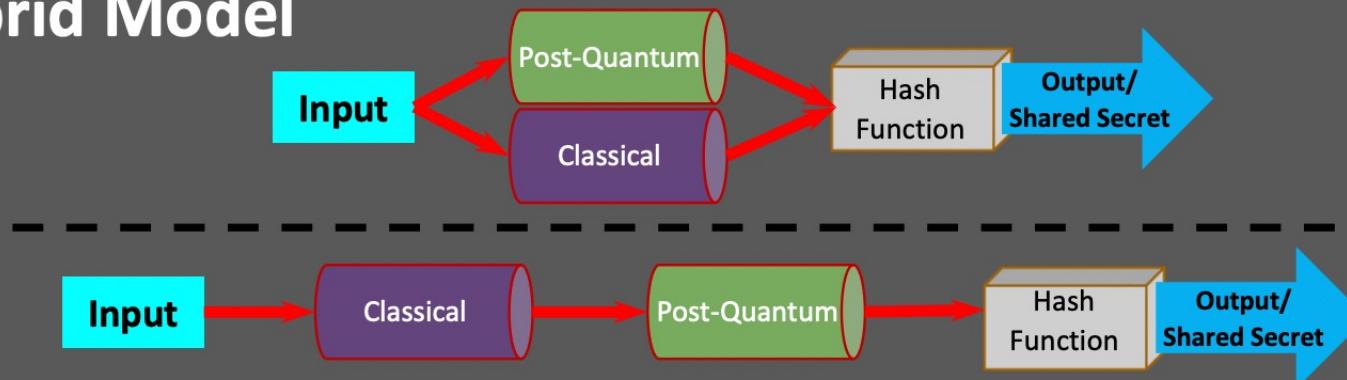
# PQC for IoT in 5G: New Attack Surface



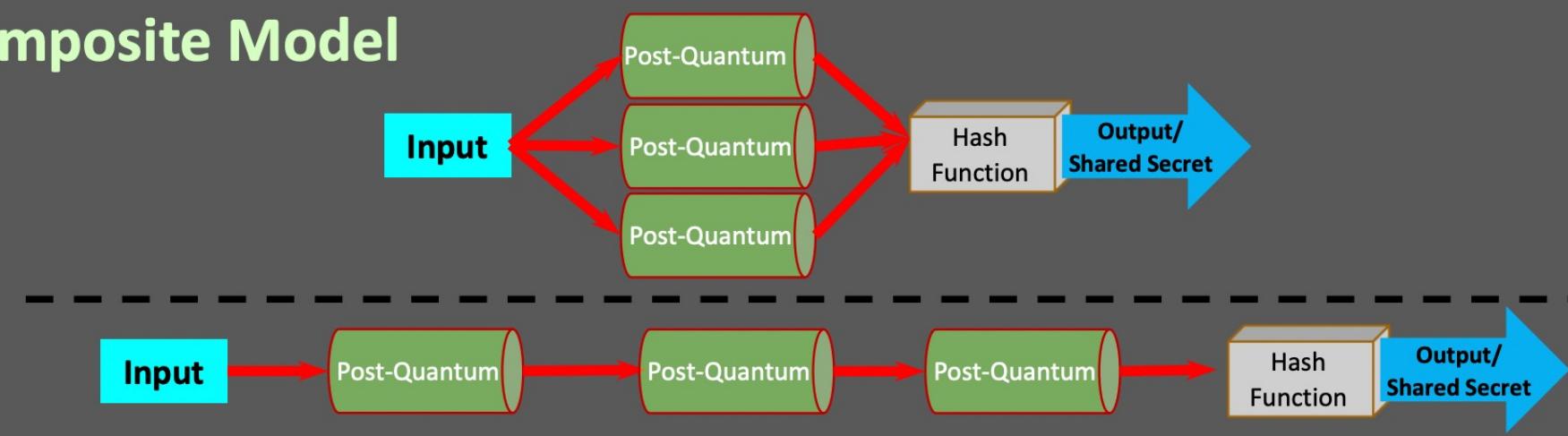
"Distribution Statement A – Approved for Public Release, Distribution Unlimited"

# Implementing hybrid/composite Sys.

## Hybrid Model



## Composite Model



"Distribution Statement A – Approved for Public Release, Distribution Unlimited"

<b>Algorithm</b>	<b>Secret Key Bytes</b>	<b>Public Key Bytes</b>	<b>Ciphertext/Signature Key Bytes</b>
<b>CRYSTALS-Kyber</b>	1,632	800	768
<b>CRYSTALS-Dilithium</b>	2,528	1,312	2,420
<b>FALCON</b>	1,281	897	666
<b>SPHINCS+</b>	32	64	7,856
<b>BIKE</b>	281	1,541	1,573
<b>Classic McEliece</b>	6,492	261,120	128
<b>HQC</b>	2,289	2,249	4,481
<b>XMSS</b>	2,144	64	2,500
<b>LMS</b>	1,820	56	8,684



### nRF5280

Arm® Cortex™-M4  
@ 64 MHz  
Arm CryptoCell CC310  
HW Crypto Engine:  
AES, Hash, RSA/ ECC, etc.



### STM32L5xx

Arm® Cortex®-M33  
TrustZone®  
@ 110 MHz  
HW Crypto Engine:  
AES/ DES3, MD5/  
SHA/SHA2. RNG etc.



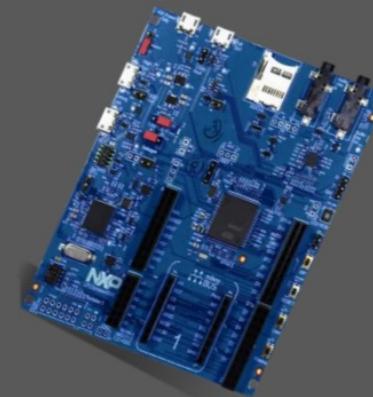
### CC2642R

ARM® Cortex® -M4F  
@ 48 MHz  
HW Crypto Engine:  
AES/ DES/ DES3, SHA2,  
RNG, RSA/ ECC, etc.



### EFM32

ARM Cortex-M4  
@ 40MHz  
HW Crypto Engine:  
AES, Hash, ECC, etc.



### LPC55S6x

Arm® Cortex®-M33  
TrustZone®  
@ 150 MHz  
HW Crypto Engine:  
AES, SHA2, RNG, etc.

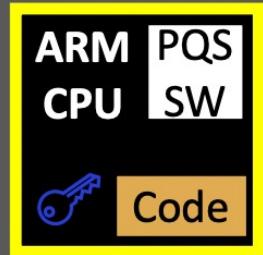


### ATECC608A

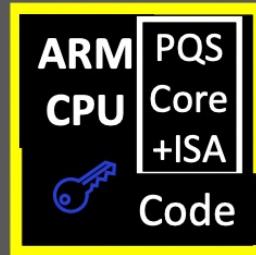
Crypto Co-Processor  
HWCrypto Engine:  
AES, SHA2, ECDH,  
ECDSA etc.  
SCA protection

"Distribution Statement A – Approved for Public Release, Distribution Unlimited"

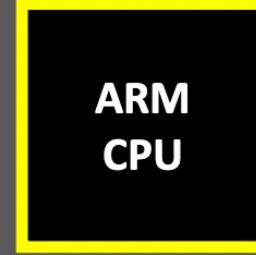
# PQSecure's Design Choices



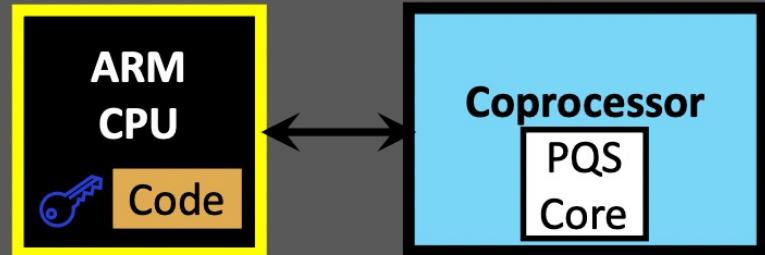
(a) SW only design  
(e.g., ARM Cortex-M4)



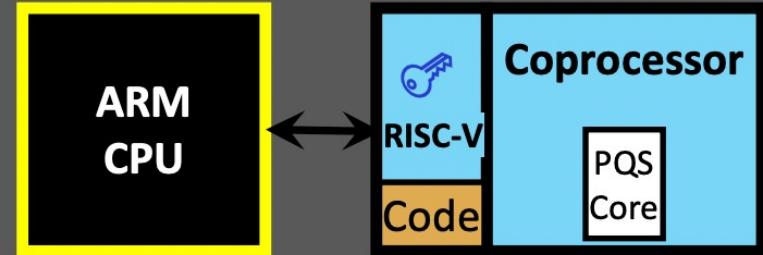
(b) Custom Instructions design  
(e.g., ARM Cortex-M33)



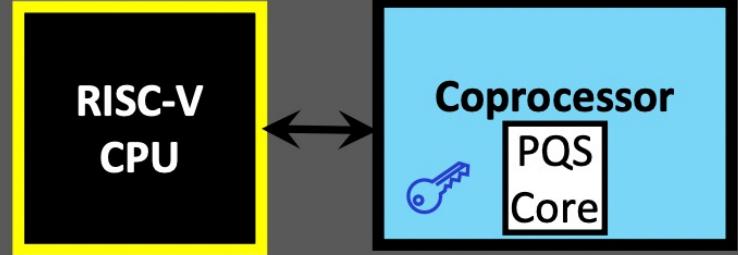
(c) HW only design with  
ASIC or FPGA prototype



(d) HW/SW co-design with HW in ASIC/  
FPGA and SW in ARM Cortex-M4



(e) HW/SW RISC-V (or custom) with  
ASIC or FPGA



(f) HW only with RISC-V with ASIC or  
FPGA

---

---

3

Security?

## **Q&A – Exchange Ideas**



A large, colorful word cloud centered around the words "thank" and "you" in various languages. The words are rendered in different colors and sizes, creating a dense and visually appealing composition. The languages represented include English, German, French, Spanish, Italian, Portuguese, Dutch, Swedish, Polish, Russian, Chinese, Japanese, Korean, Vietnamese, Thai, Indonesian, Malay, and many others from around the world.

**www: ism.ase.ro | acs.ase.ro | dice.ase.ro**

**Scan the Tag  
to get @ the web  
Mobile Address**

