Quantum…

Computing

From Classical to Quantum

Logistics

- Instructors:
  - Bhiksha Raj
  - Daniel Justice
  - Rita Singh
- TA:
  - Thomas Cantalapiedra
- Timings/Location: 5-6.20pm Mon/Wed, GHC 4215
  - Lectures will be on zoom until Feb 8, in person thereafter
  - Labs will be in person, 5-6.20, GHC
- Grading: 10% attendance, 30% quiz, 30% homeworks, 30% group project

Logistics

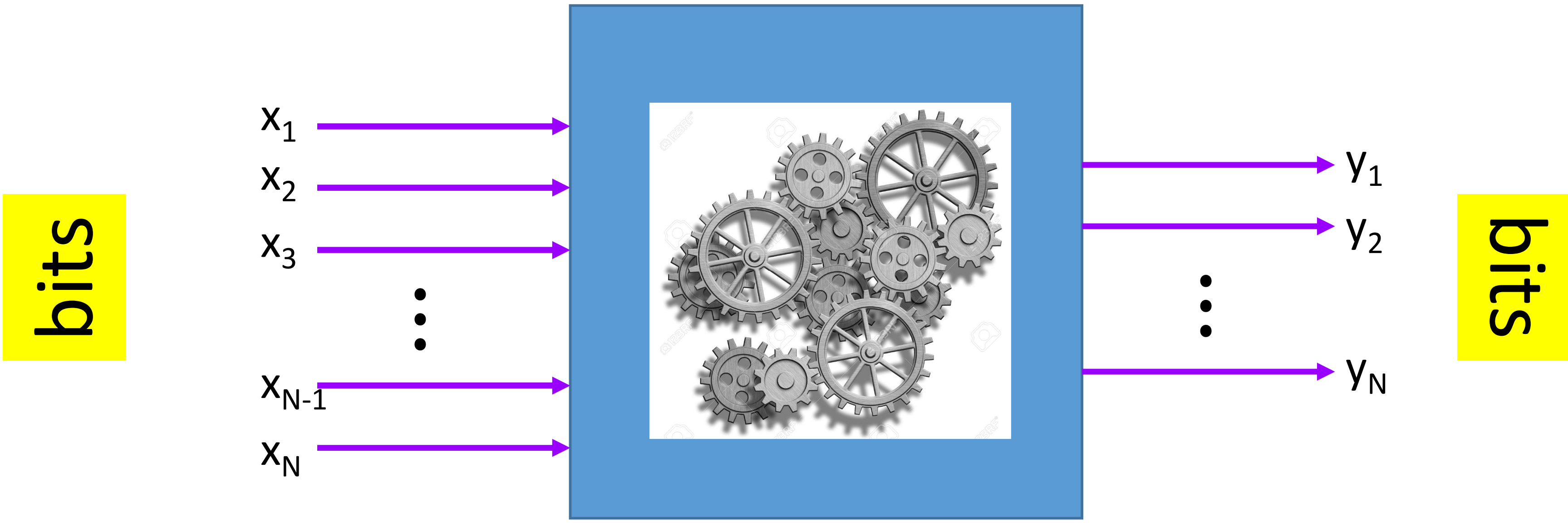
- Syllabus posted on course website, piazza and canvas

- Course website: https://thequantumturtle.github.io/courses/2025-Spring-11860/homepage/
    - Soon to be https://quantum.lti.cs.cmu.edu

- Course piazza: https://piazza.com/class/m5szmfcg9a12k9

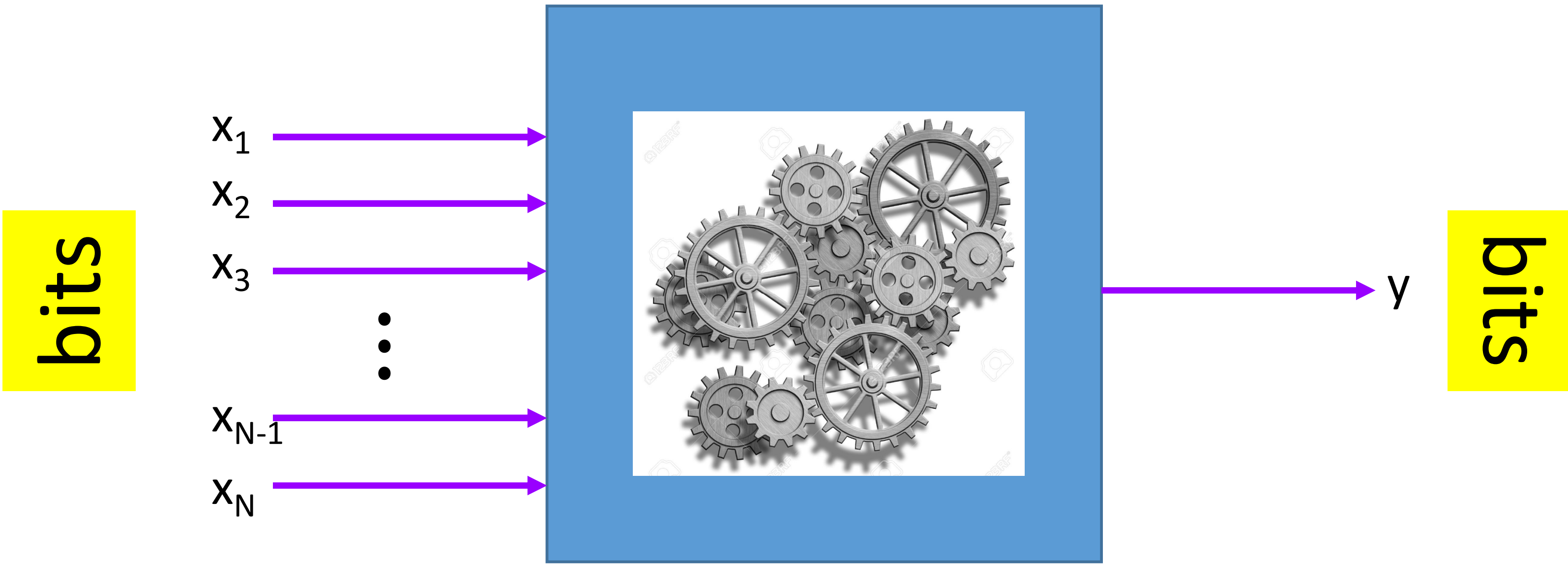
- Course canvas: https://canvas.cmu.edu/courses/44546

A standard model for Boolean functions (aka algorithms)

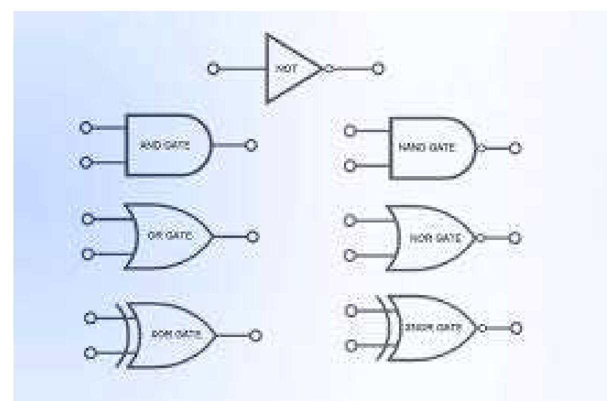


- A bunch of bits go in, and a bunch of bits come out

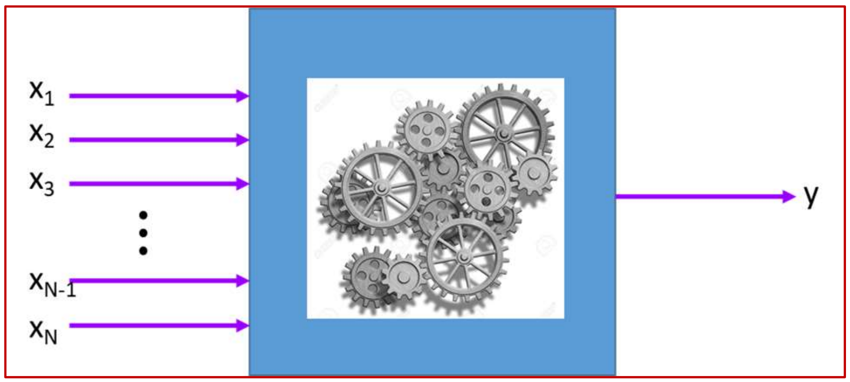
A simpler model for computation

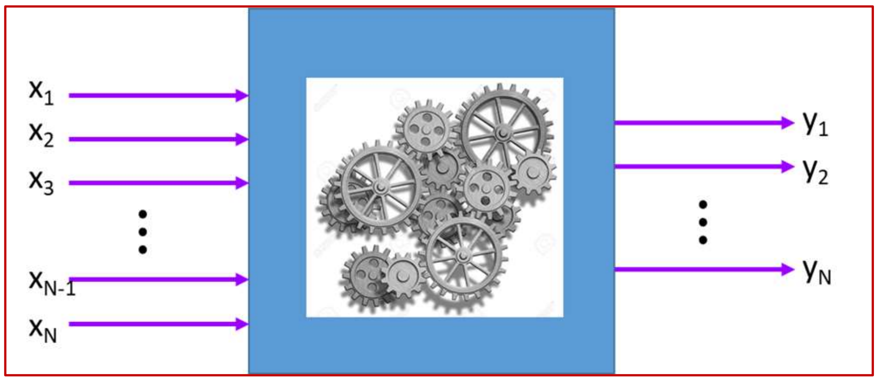


- A bunch of bits go in, and one bit comes out.
- Examples to the right



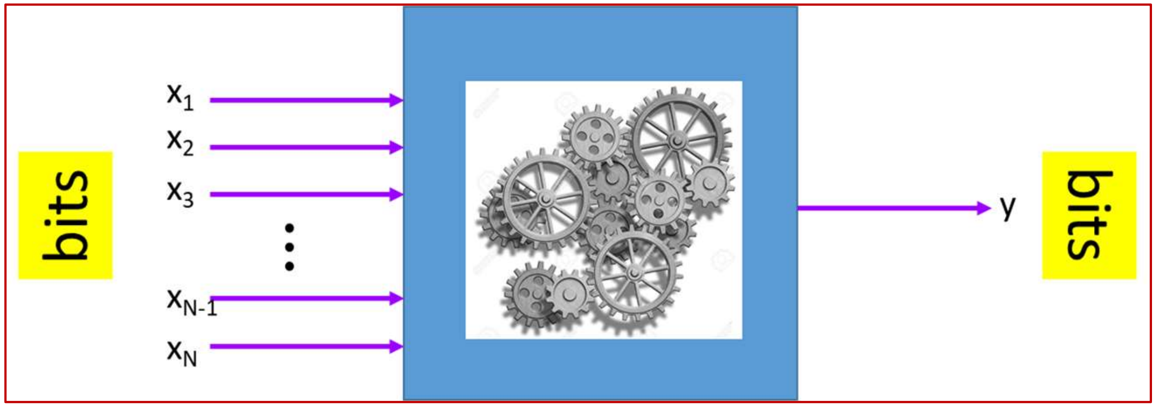
A simpler model for computation

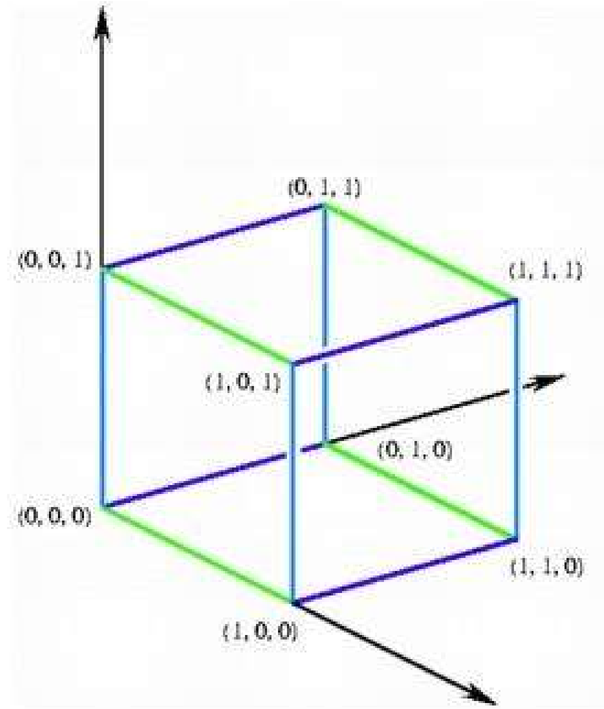




- The model to the left is in fact a generalization of the model to the right
  - How?

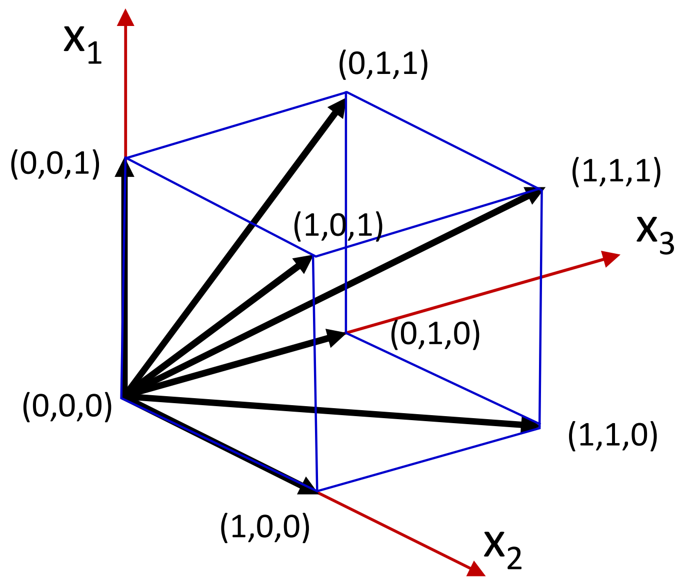
The actual model of computation





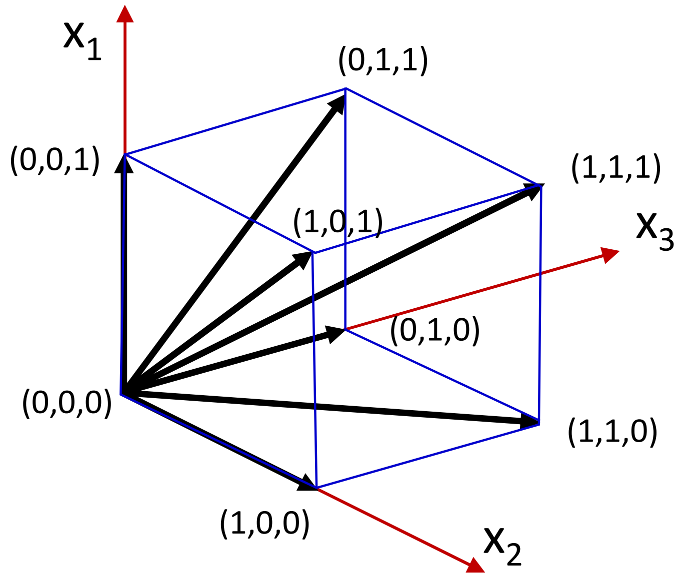
- It is a function of N bits
  - An input goes in, a unique value comes out
- The inputs lie in an N-dimensional space
  - Each input dimension can take only two values [0,1].
  - Other values are not defined
- The entire set of all possible inputs lies on the corners of an N-dimensional hypercube
  - The interior of the cube is infeasible
- The function is defined on the corners of this hypercube

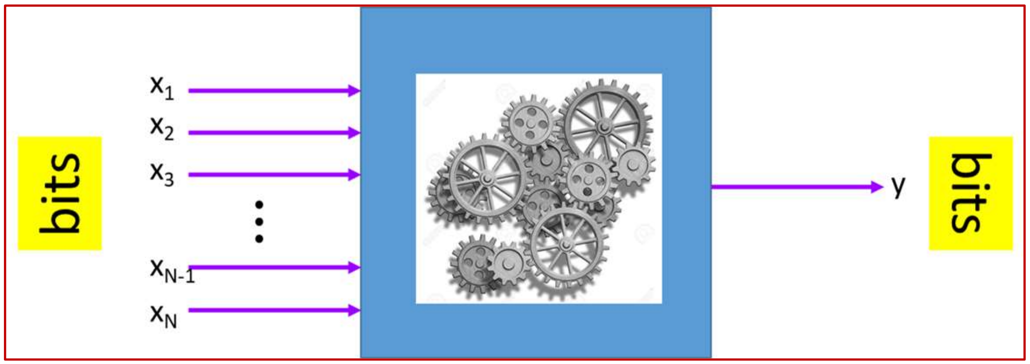
The input space representation



- For a function over N bits, the function is defined over an N-dimensional input space
  - **Each input bit represents an orthogonal direction in the space!**
- Each feasible input combination is an N-dimensional vector in this space
  - The feasible set of inputs is a countable, finite set of 2^N points.
- In order to fully represent an input, N values must be provided
  - One number for each input coordinate

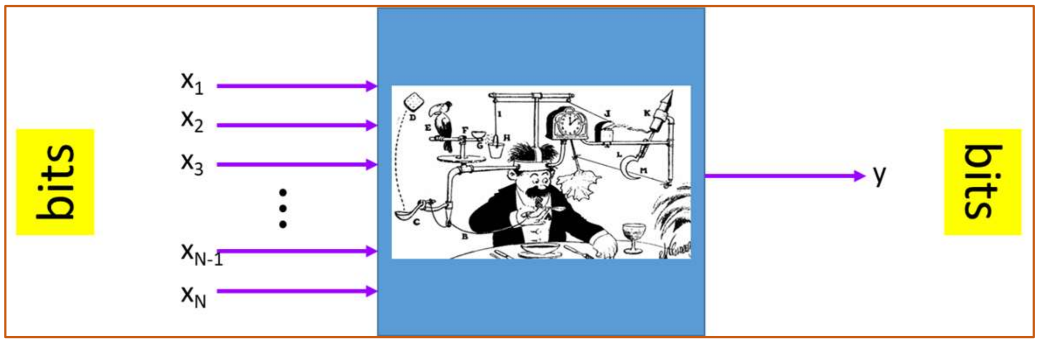
The function

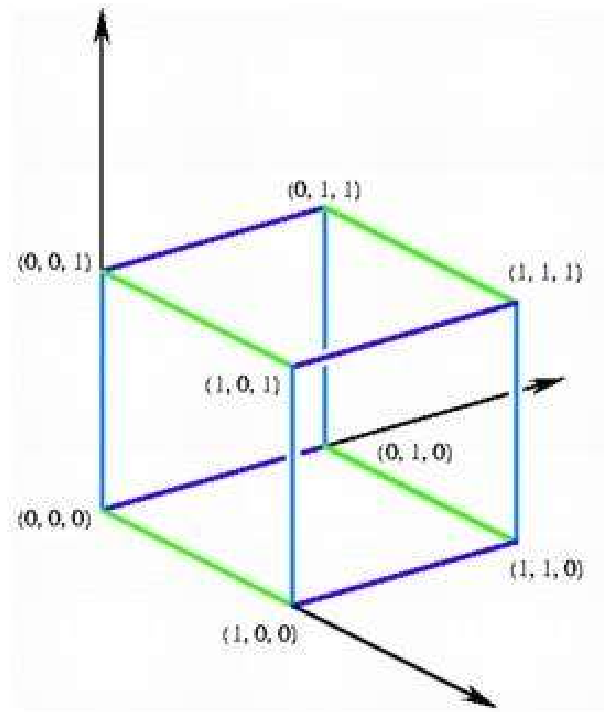




- The algorithm is simply a function that operates on this input space
  - It transforms each input vector to a Boolean value
- Note again that each input vector represents a *single* possible combination of bits
  - When operated on a vector, the function computes the output for that combination of input bits

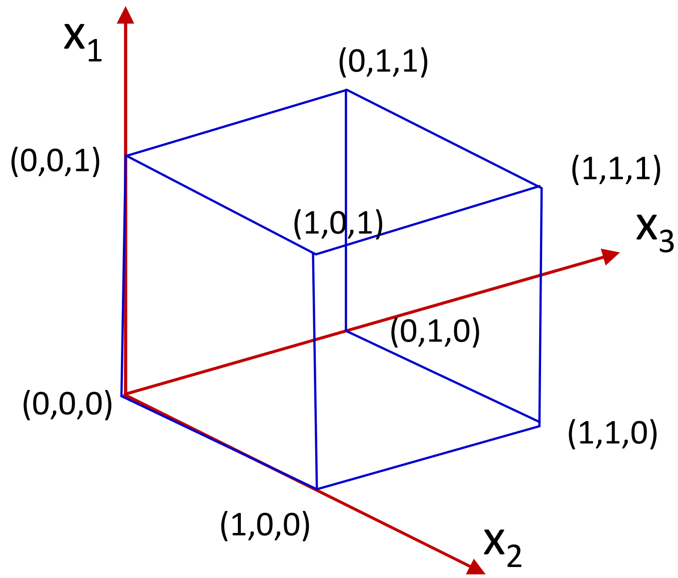
The unknown function problem

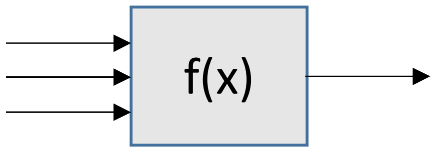




- You are given an uncharacterized function
  - N inputs
  - You know the formulae in it, but don't know what outputs it will produce for any input
- You must characterize the function fully
- How many measurements must you make?
  - "Measurement" : provide an input and note the output

"Discovering" a 3-bit function

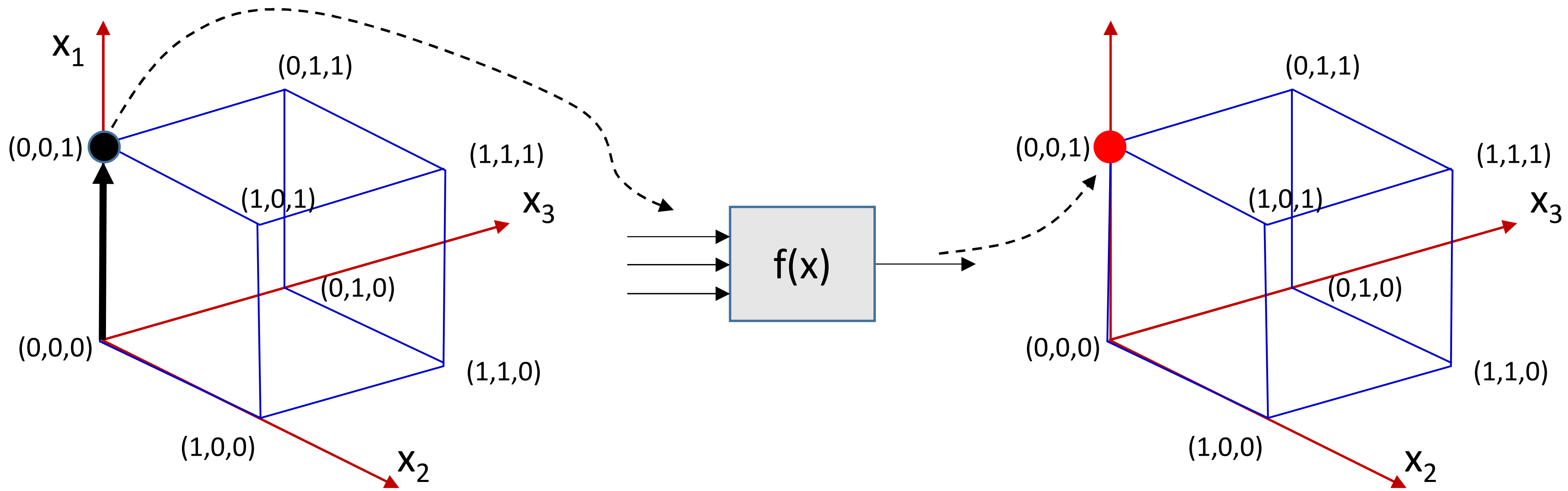




Determine this function fully!

- Find out how it responds to any input

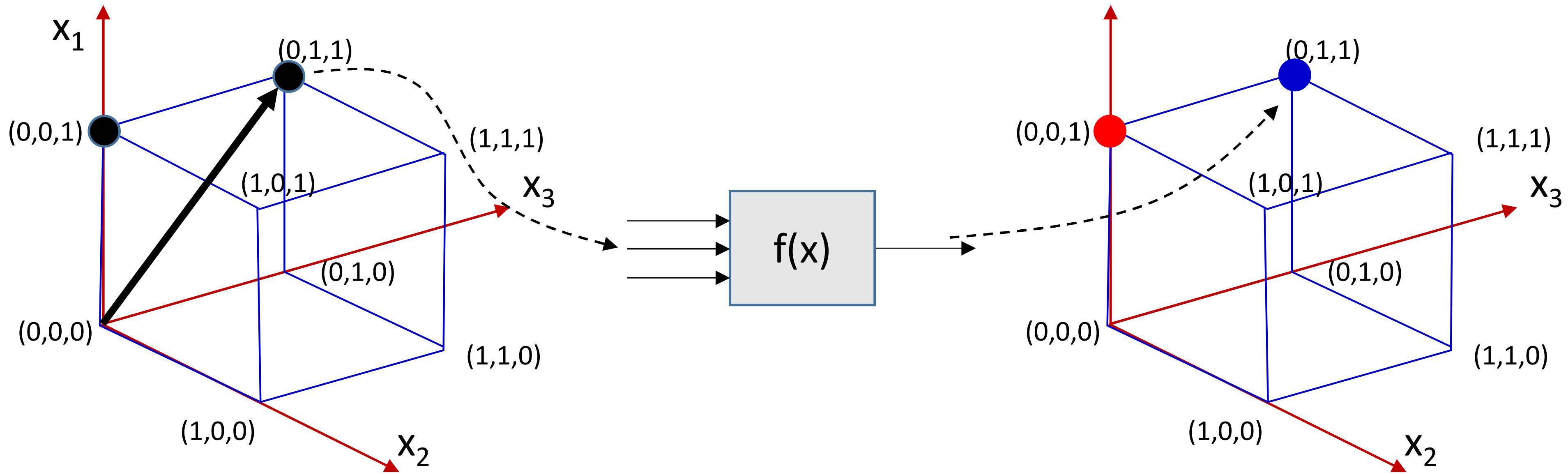
"Discovering" a 3-bit function



Determine this function fully!

- Pass each possible input vector through the function and compute its response

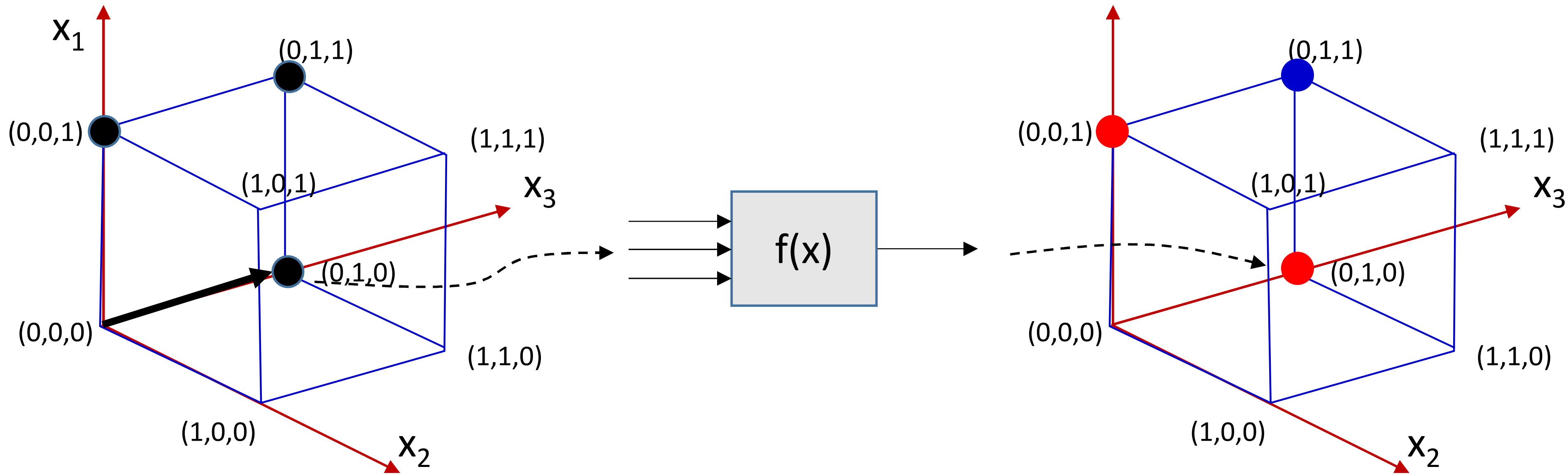
“Discovering” a 3-bit function



Determine this function fully!

- Pass each possible input vector through the function and compute its response

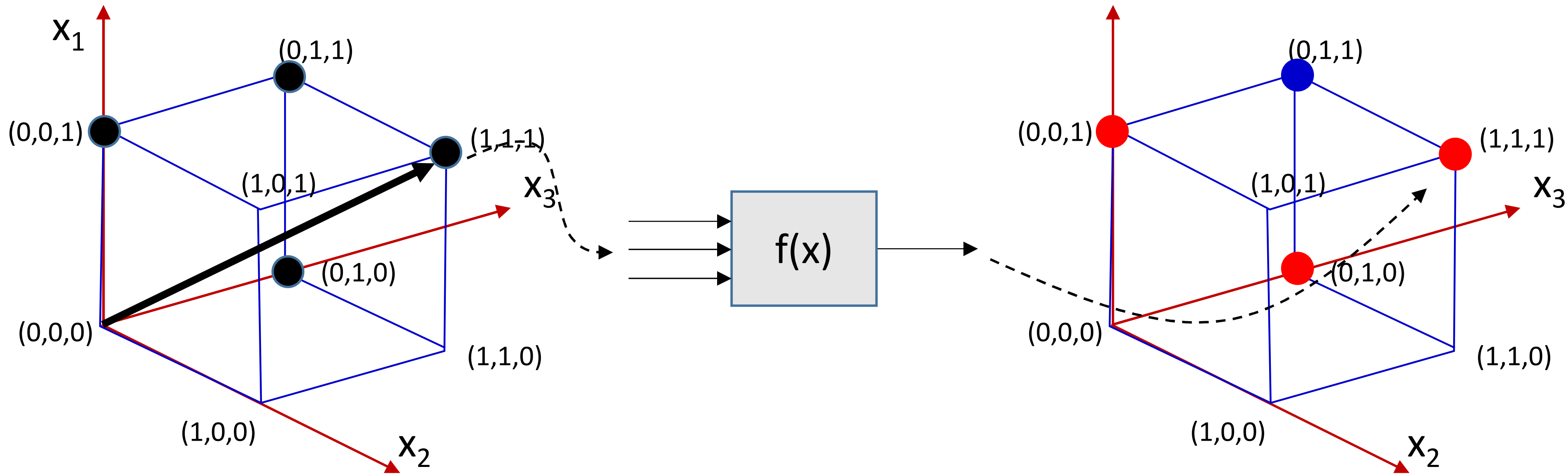
“Discovering” a 3-bit function



Determine this function fully!

- Pass each possible input vector through the function and compute its response

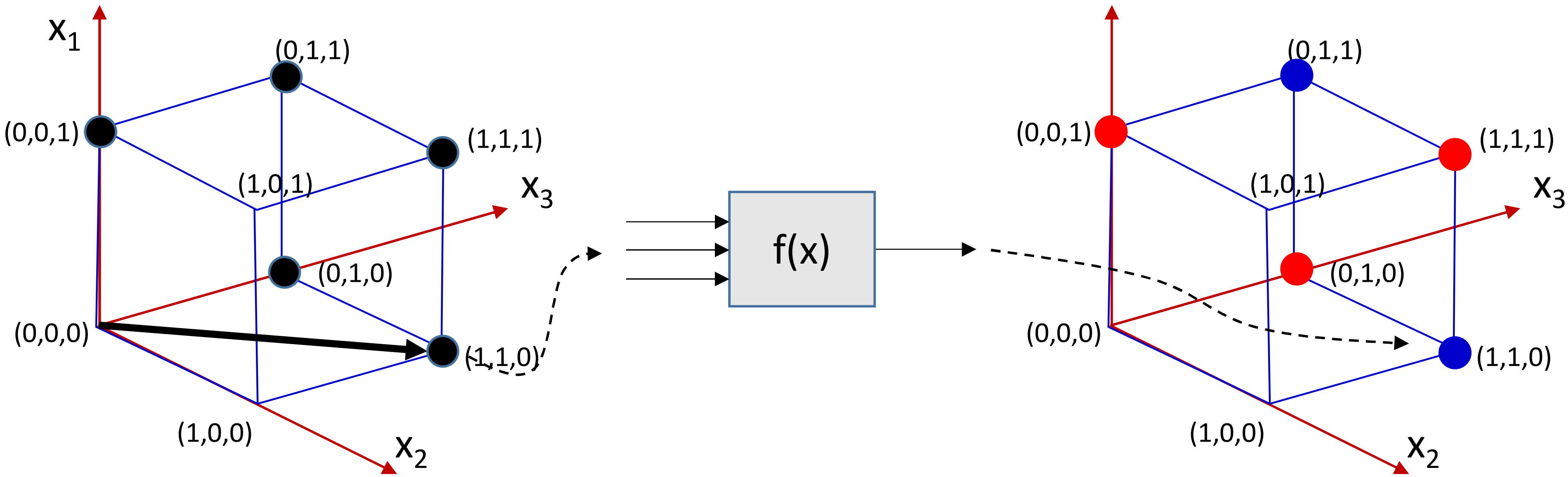
“Discovering” a 3-bit function



Determine this function fully!

- Pass each possible input vector through the function and compute its response

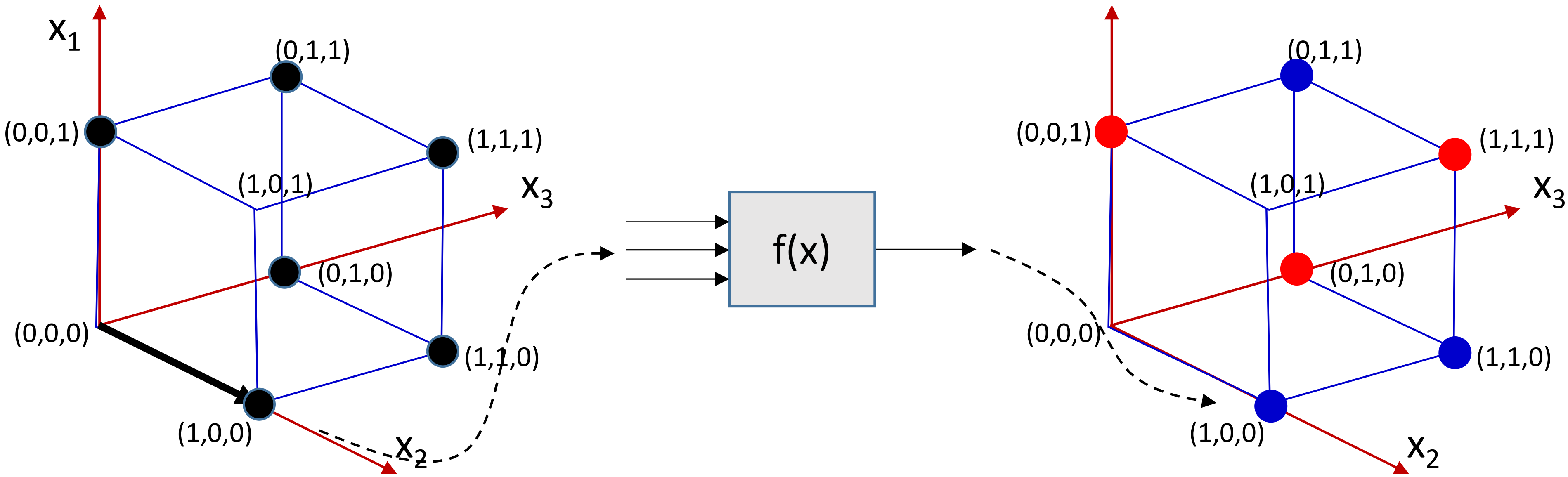
“Discovering” a 3-bit function



Determine this function fully!

- Pass each possible input vector through the function and compute its response

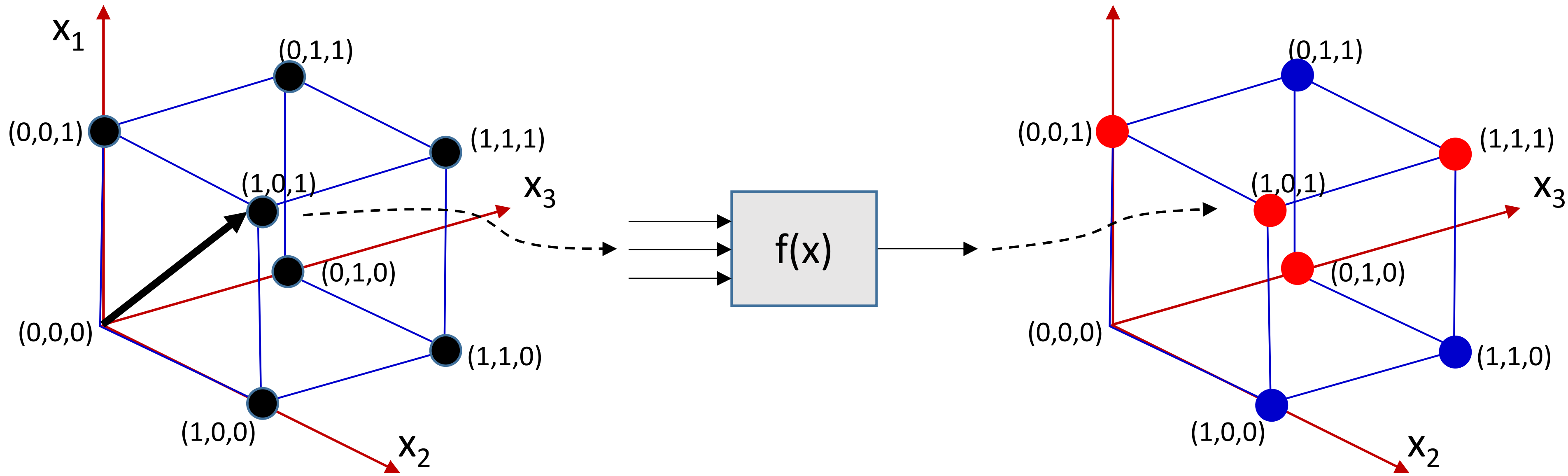
"Discovering" a 3-bit function



Determine this function fully!

- Pass each possible input vector through the function and compute its response

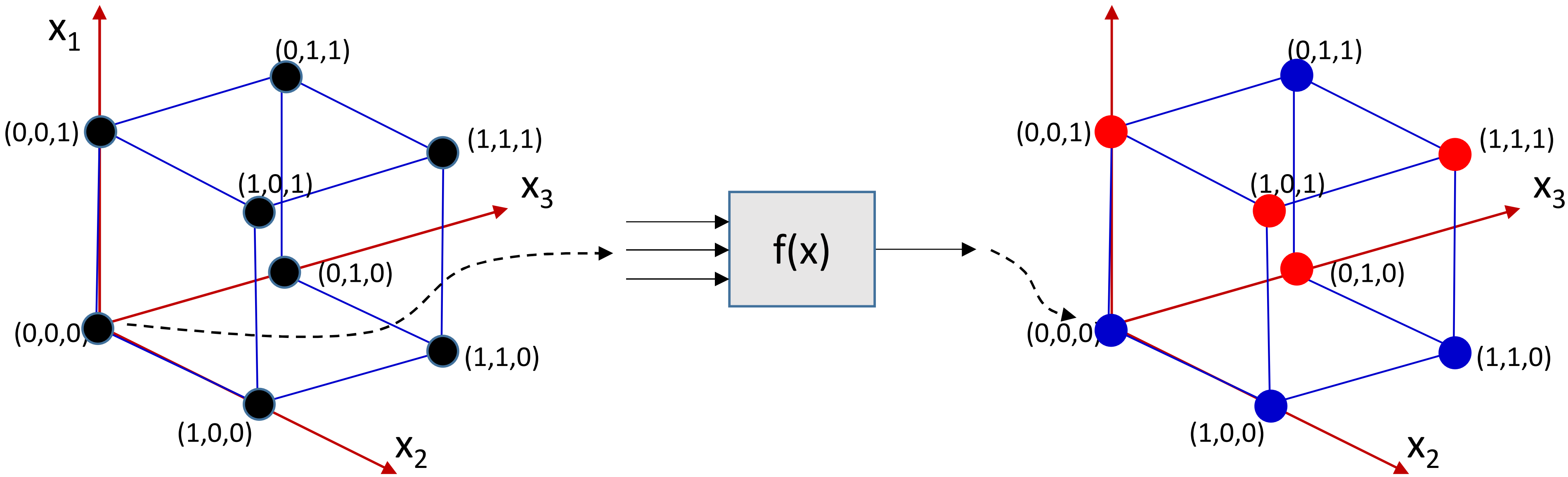
“Discovering” a 3-bit function



Determine this function fully!

- Pass each possible input vector through the function and compute its response

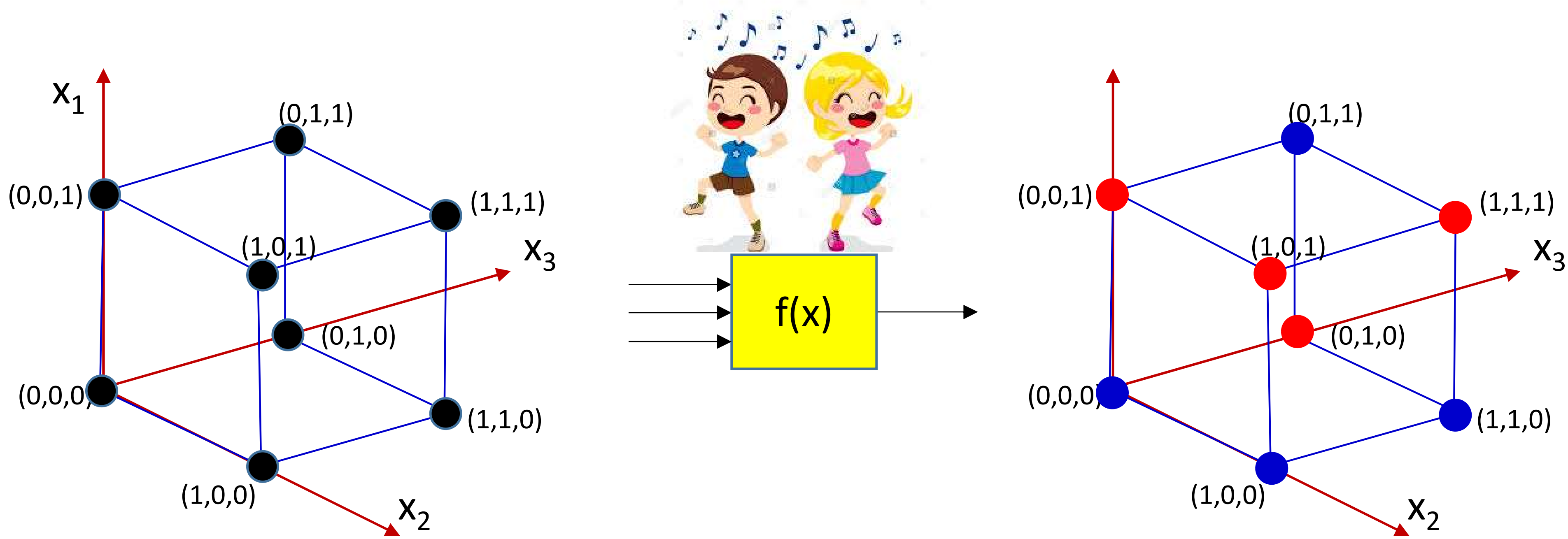
“Discovering” a 3-bit function



Determine this function fully!

- Pass each possible input vector through the function and compute its response

"Discovering" a 3-bit function



- Pass each possible input vector through the function and compute its response
- *And now its known*

Poll 1 A

- You are given an N-input Boolean function and a claim that the function outputs a 1 for some inputs, without specifying which. You would like to verify that this is true.

  What is the only way to be absolutely certain of your conclusion?

Poll 1 A

- You are given an N-input Boolean function and a claim that the function outputs a 1 for some inputs, without specifying which. You would like to verify that this is true.

  What is the only way to be absolutely certain of your conclusion?

- *You must test the function for each possible input, until you find one that results in an output of 1*
  If none of the inputs output a 1, the original claim is false

Poll 1 B

You are given an N-input Boolean function and a claim that the function outputs a 1 for some inputs. You would like to verify that this is true.

1. How many different inputs must you test to verify the claim (in the worst case):

Poll 1 B

You are given an N-input Boolean function and a claim that the function outputs a 1 for some inputs. You would like to verify that this is true.

1. How many different inputs must you test to verify the claim (in the worst case):
   - **2^N**

Poll 1 B

You are given an N-input Boolean function and a claim that the function outputs a 1 for some inputs. You would like to verify that this is true.

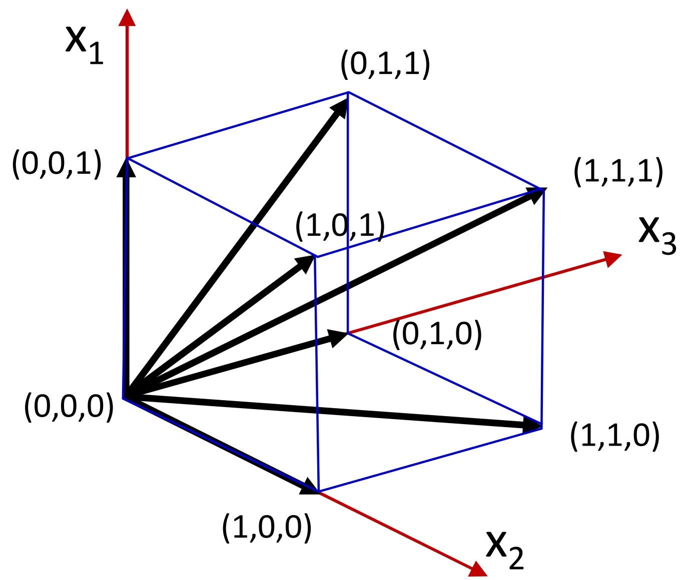
1. How many different inputs must you test to verify the claim (in the worst case):
   - **2^N**
2. Will you always need to test those many inputs?
   - Yes
   - No

Poll 1 B

You are given an N-input Boolean function and a claim that the function outputs a 1 for some inputs. You would like to verify that this is true.

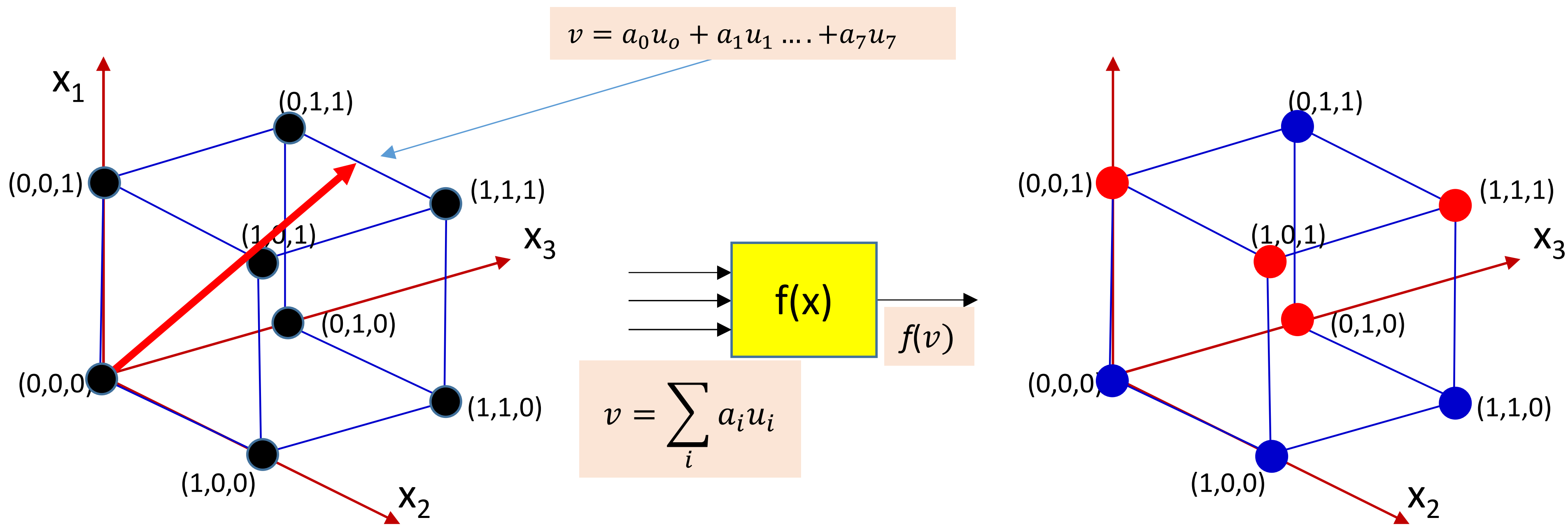
1. How many different inputs must you test to verify the claim (in the worst case):
   - **2^N**
2. Will you always need to test those many inputs?
   - Yes
   - **No**. There are many problems where you can verify the claim in sub-exponential, polynomial, or even linear time

"Discovering" a 3-bit function



- Problem with this approach: Each possible input vector must be *individually* evaluated
- *Is there a shortcut?*

Let’s try to improve on this…

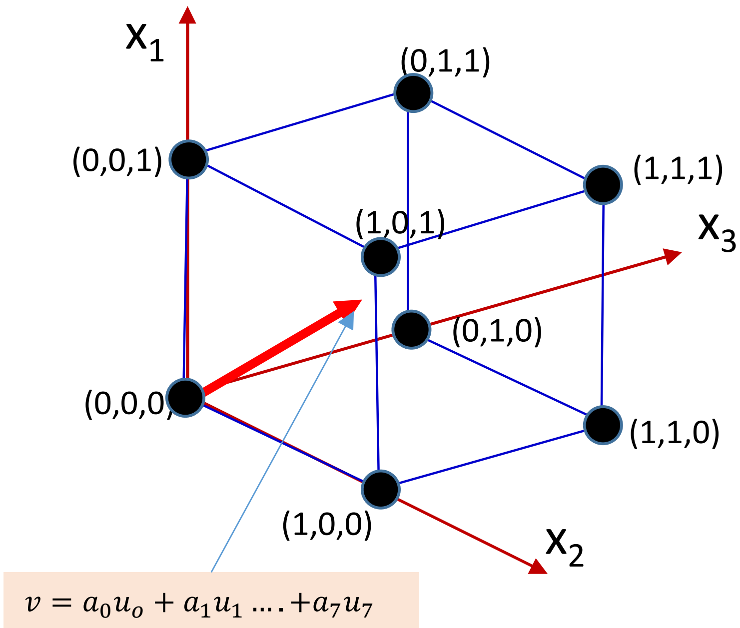


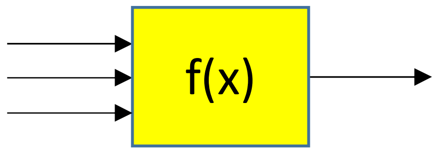
Here’s an idea…

- How about we send in a *linear combination of* valid inputs?
  - Would the resultant output allow us to determine the outputs for each of the constituents of the combination in one shot?

Poll 2

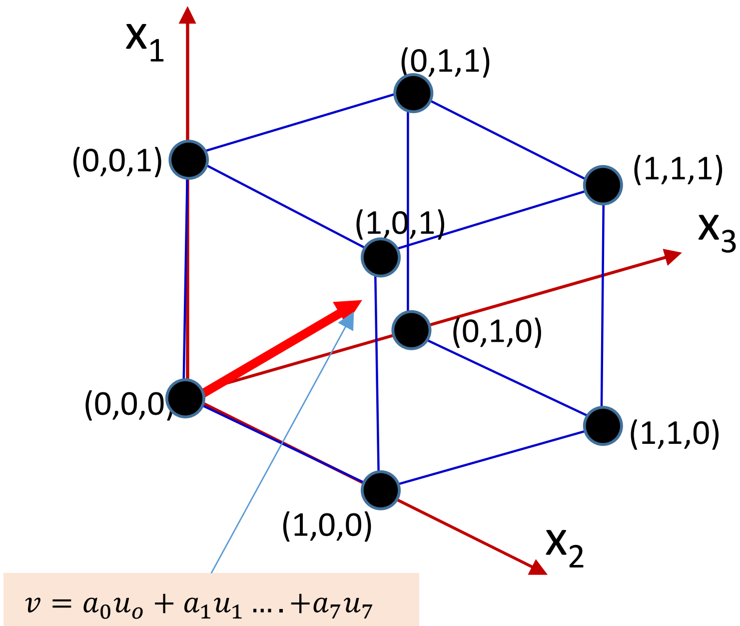
- Consider a 3-bit function
- Let $u_0 \ldots u_7$ be the 8 possible binary inputs
- We input a linear combination $v = a_0 u_o + a_1 u_1 \ldots . + a_7 u_7$ to the function, and obtain an output $f(v)$

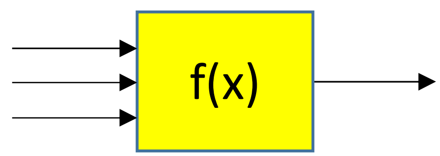




- From the one measurement $f(v)$, we can recover the responses to the individual patterns: $f(u_i), i = 0 \ldots 7$
  - True: we can unambiguously identify the outputs in response to each input from the output in response to the linear combination of inputs
  - False. This is not possible.

Poll 2



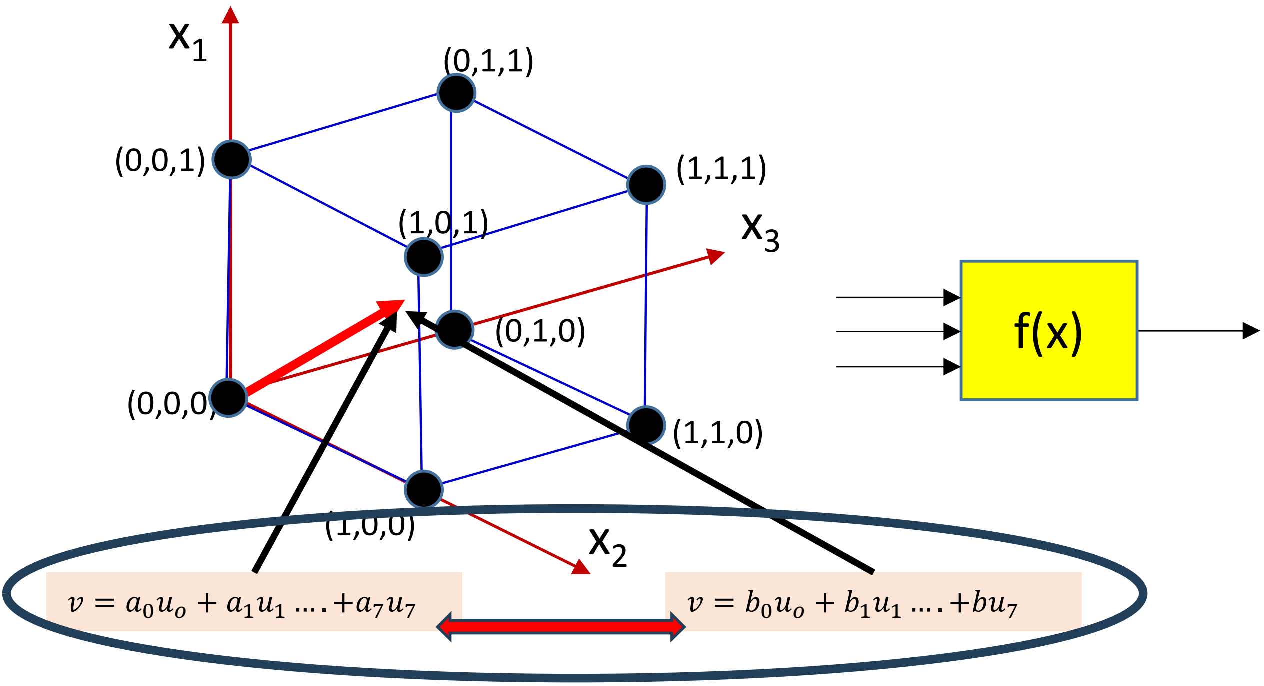


- Consider a 3-bit function
- Let $u_0 \ldots u_7$ be the 8 possible binary inputs
- We input a linear combination $v = a_0u_o + a_1u_1 \ldots. + a_7u_7$ to the function, and obtain an output $f(v)$

- From the one measurement $f(v)$, we can recover the responses to the individual patterns: $f(u_i), i = 0 \ldots 7$
  - True: we can unambiguously identify the outputs in response to each input from the output in response to the linear combination of inputs
  - **False. This is not possible.**

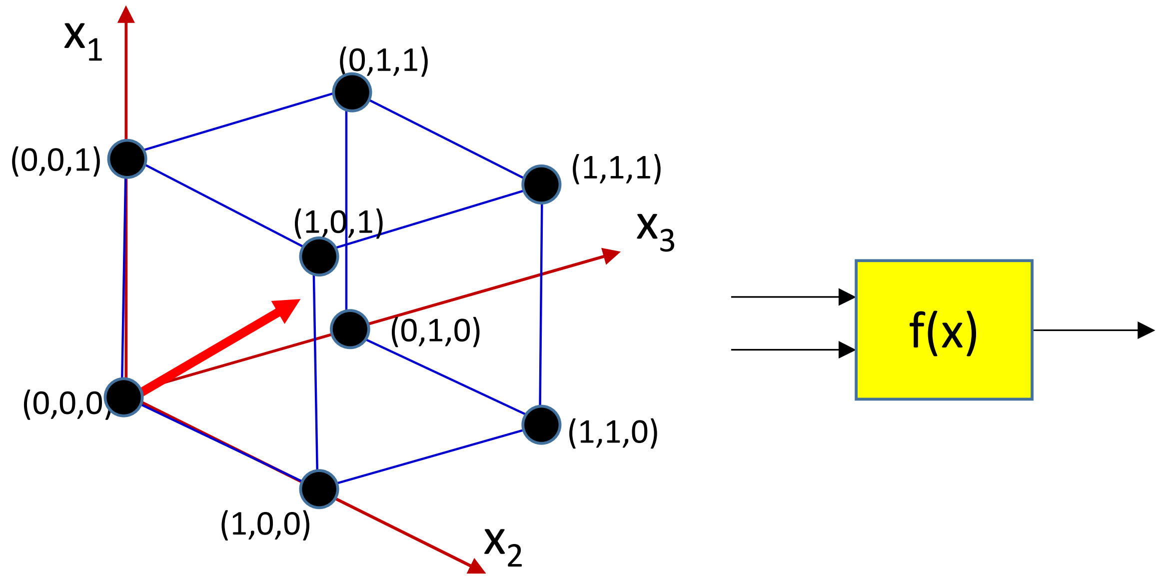
Poll 2

- Consider a 3-bit function
- Let $u_0 \ldots u_7$ be the 8 possible binary inputs
- We input a linear combination $v = a_0 u_o + a_1 u_1 \ldots . + a_7 u_7$ to the function, and obtain an output $f(v)$



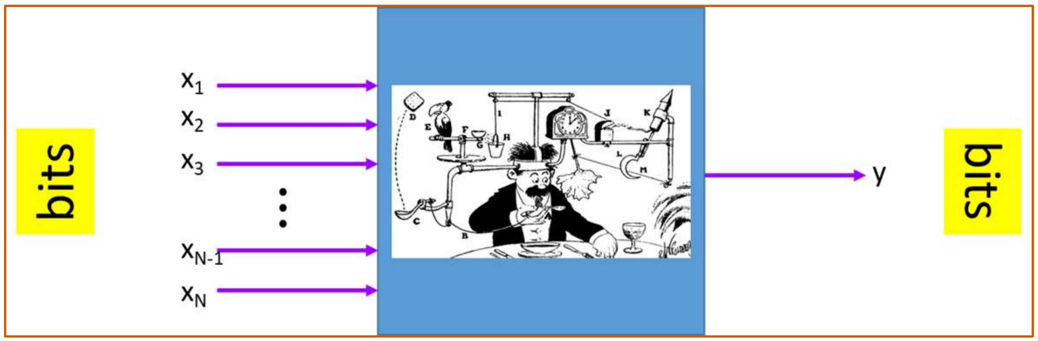
- FALSE for many reasons
- This is a 3 dimensional space. v can be composed in infinite different ways from the 8 u_is (underdetermined)
  - So, we would not know which $u_i s$ to attribute the result to, and how much
- $f(v)$ may itself be non-invertible
  - Many different vs may result in the same $f(v)$, further increasing the ambiguity

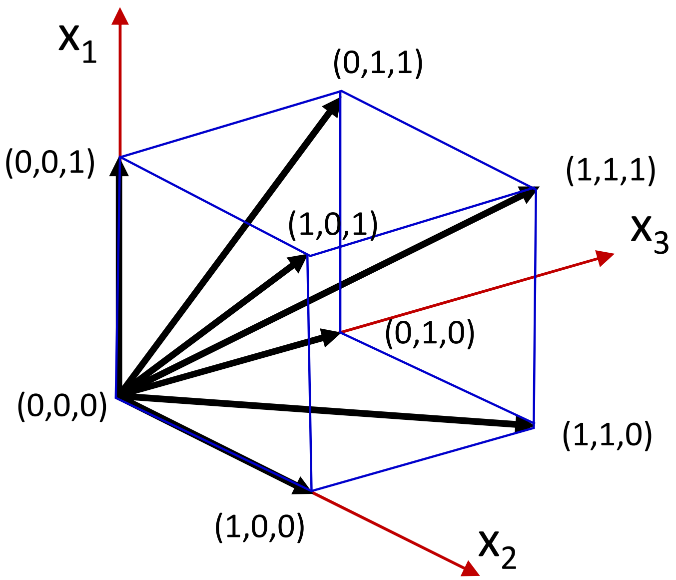
The problem with classical computation



- The function must explicitly be computed on each bit pattern u_i
- Cannot take shortcuts by using linear combinations $v = \sum_i a_i u_i$ of patterns as inputs to the function
  - Any vector v is a non-unique combination of bit patterns, and assignment of the output to the individual bit patterns is ambiguous
  - The functions $f(v)$ themselves are generally uninvertible, increasing the ambiguity
- Explicit computation of the function for each of the 2^N inputs becomes mandatory

Recap: The N-bit unknown function problem

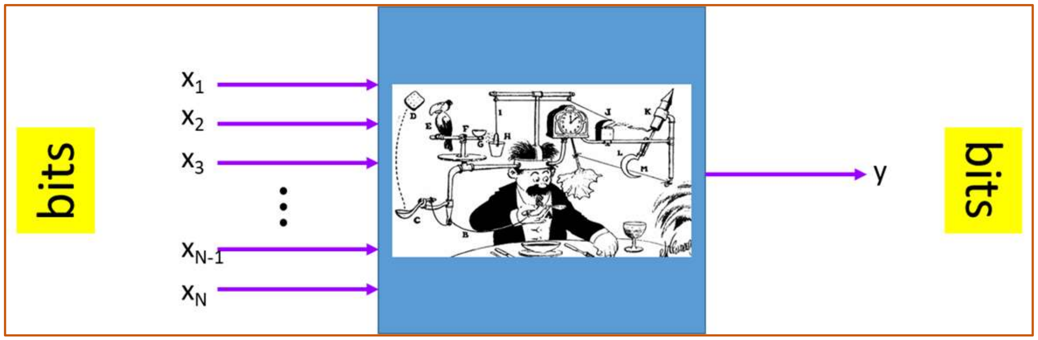


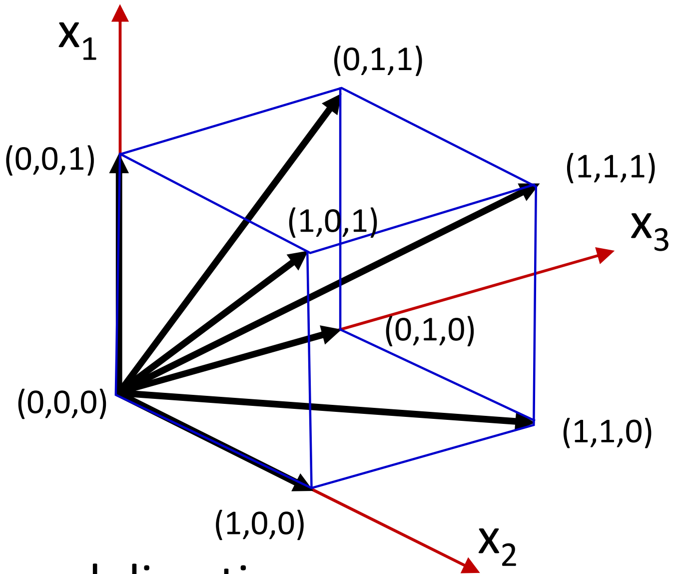


- Recall: For a function over N bits, the function is defined over an N-dimensional input space

- The feasible set of inputs is a countable, finite set of 2^N vectors.
- The function must be individually evaluated at each of these 2^N vectors to define it fully
  - Requiring 2^N evaluations in the worst case
  - This cannot be reduced

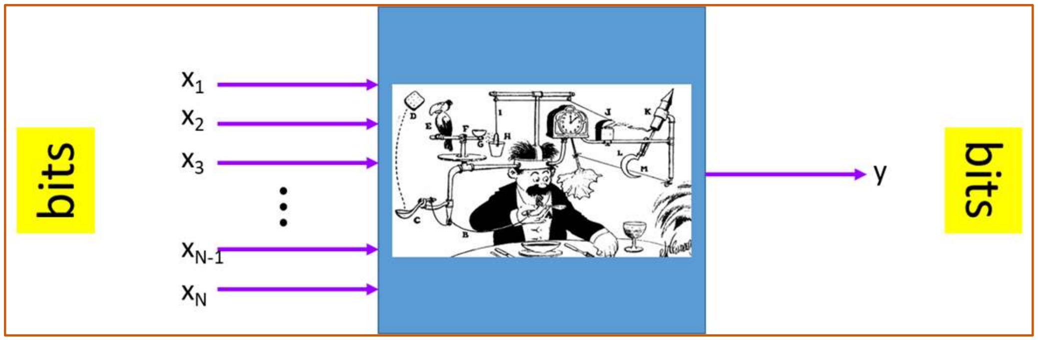
Why this limitation?

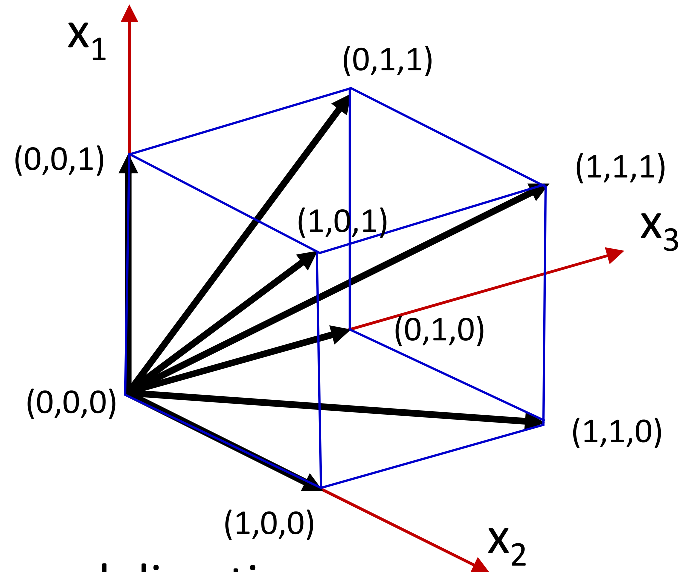




- In the classical paradigm each of the bits is an orthogonal direction
- Feasible bit patterns are vectors in this N-D space. To characterize the function all 2^N feasible input vectors must be explicitly evaluated
- Cannot 'mix' feasible input vectors to recover the outputs at multiple feasible inputs in a single measurement
  - An infinity of linear combinations can result in the same 'mixed' vector making unambiguous recovery of response to individual vectors impossible
  - The function itself may be non-invertible, making unambiguous recovery of response to individual vectors impossible

Why this limitation?



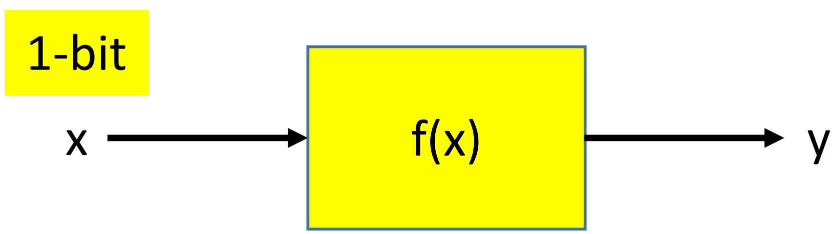


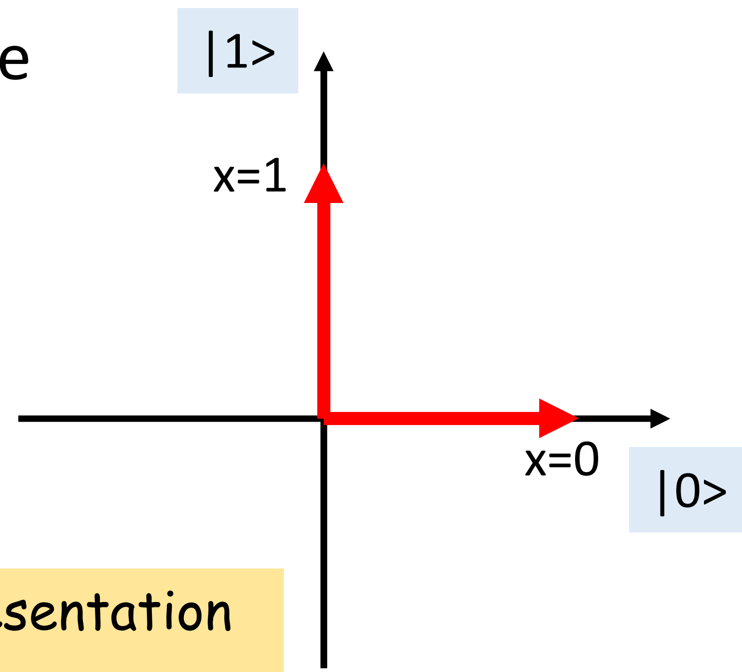
- In the classical paradigm each of the bits is an orthogonal direction
- Feasible bit patterns are vectors in this N-D space. To characterize the function all 2^N feasible input vectors must be explicitly evaluated
- Cannot 'mix' feasible input vectors to recover the outputs at multiple feasible inputs in a single measurement
  - An infinity of linear combinations can result in the same 'mixed' vector making unambiguous recovery of response to individual vectors impossible
  - The function itself may be non-invertible, making unambiguous recovery of response to individual vectors impossible

- Can we change the mathematical paradigm itself to resolve this problem?

A different approach

- **Modify the representation**
- Instead of each *bit* representing a (orthogonal) coordinate direction we will make each *combination of bits* represent a orthogonal coordinate direction!
- Even 1 bit is now in a 2-D input space

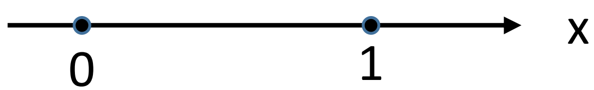




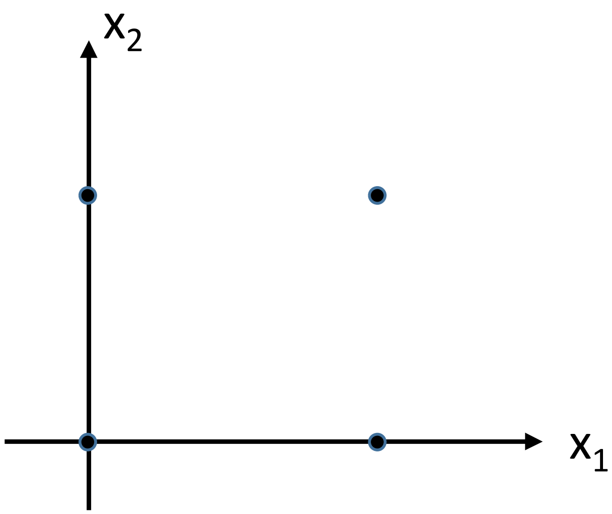
Note that this a fundamentally different representation from standard representations!

A different approach

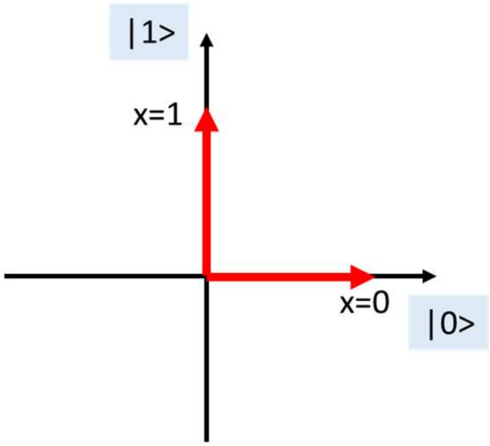
- **1 bit: Old representation**



- **2 bits: Old representation**



1 bit: New representation



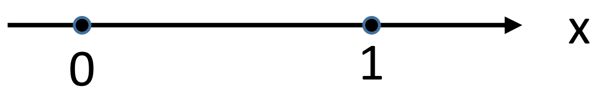
2 bits: New representation

Can't really visualize

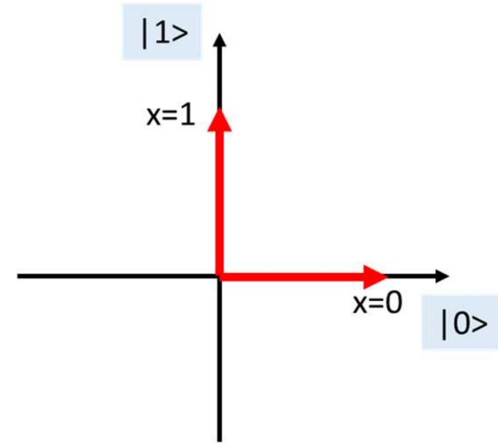
(why)?

A different approach

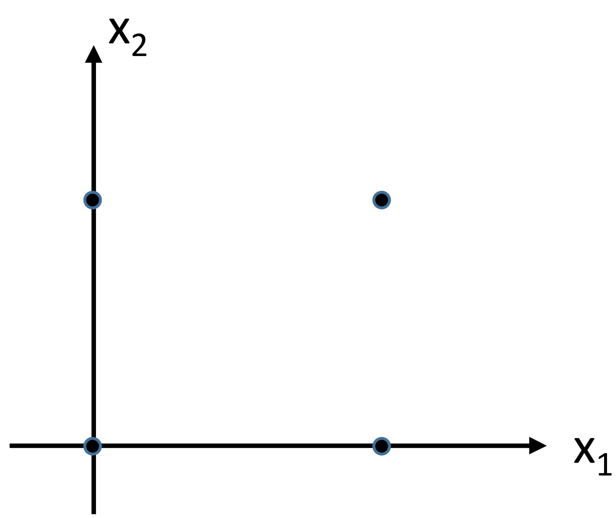
- **1 bit: Old representation**



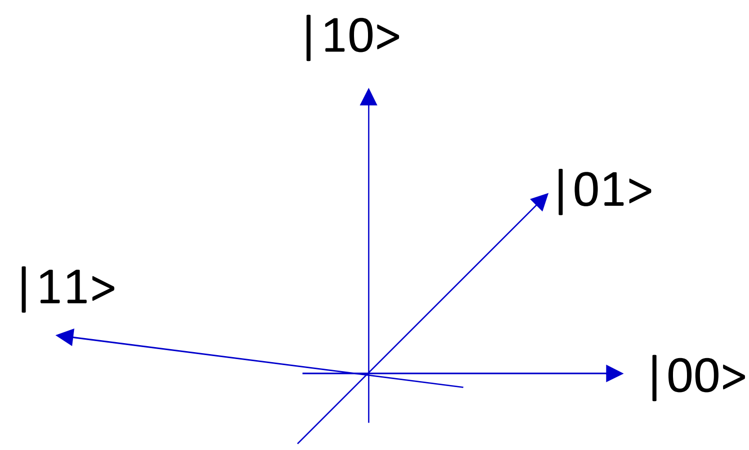
1 bit: New representation



- **2 bits: Old representation**



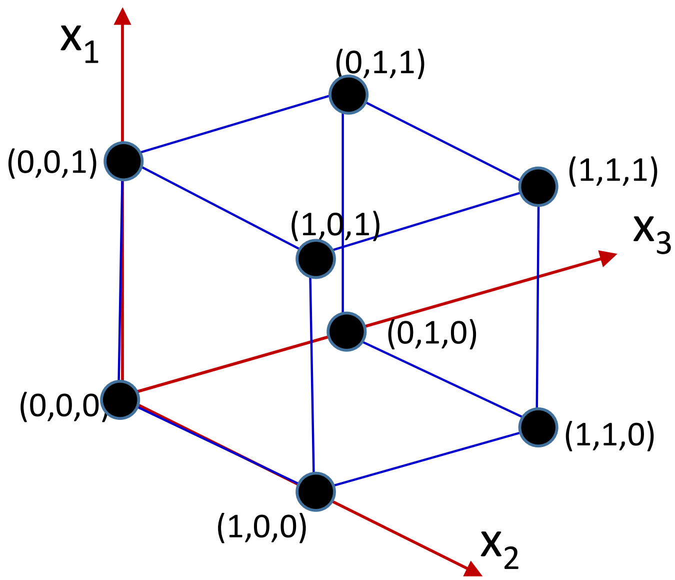
2 bits: New representation



Lame attempt at visualizing 4D

A different approach

- **3 bits: Old representation**



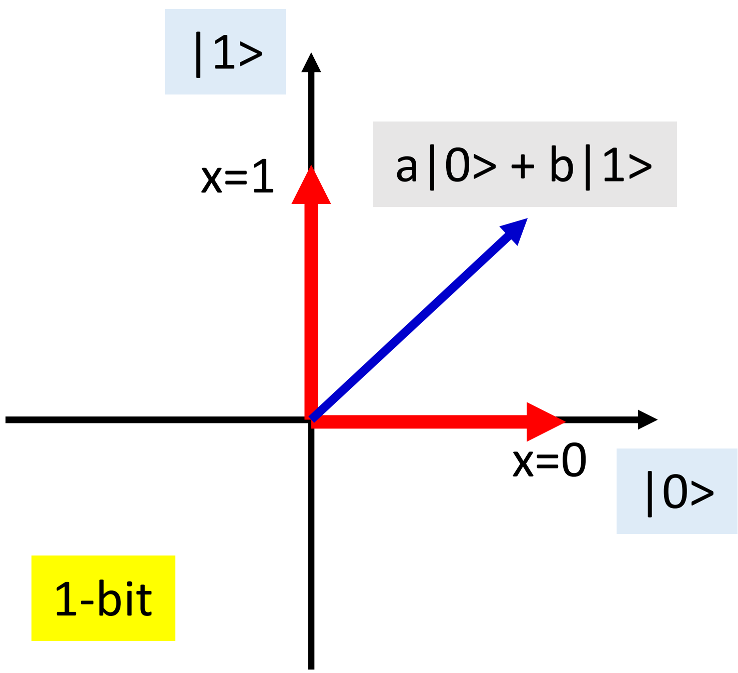
3 bits: New representation

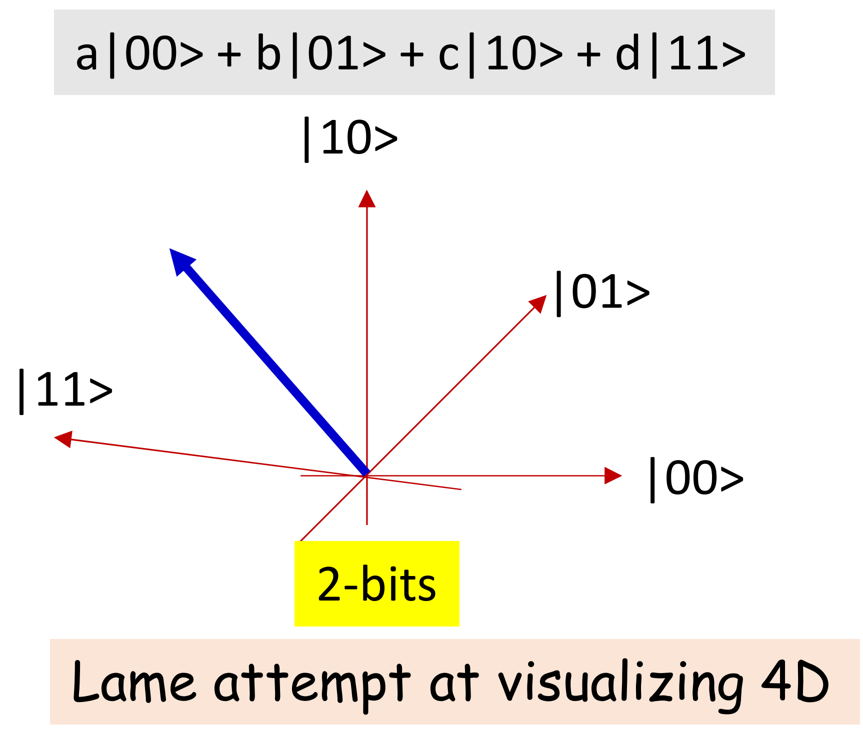
Can't really visualize

(why)?

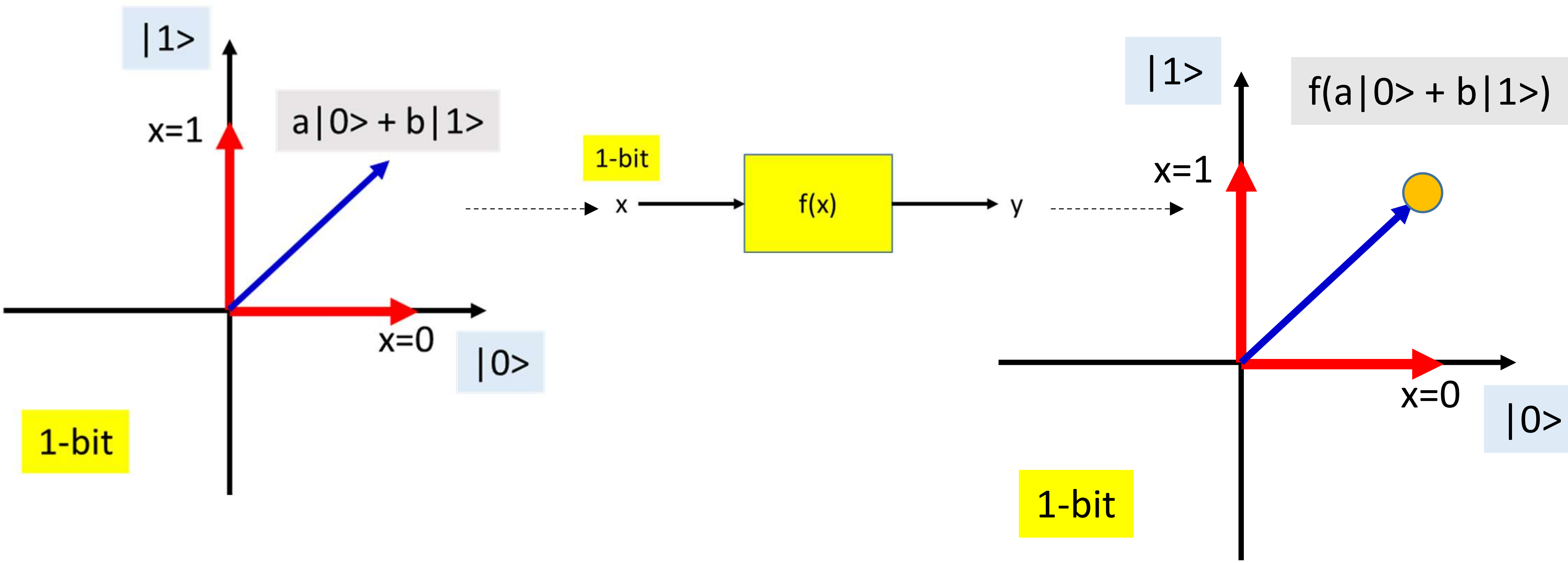
The modified representation

- A vector in this representation is a linear, *unambiguous* combination of *all* input bit patterns
- Unambiguous because we set each bit pattern to be an *orthogonal* direction to every other pattern



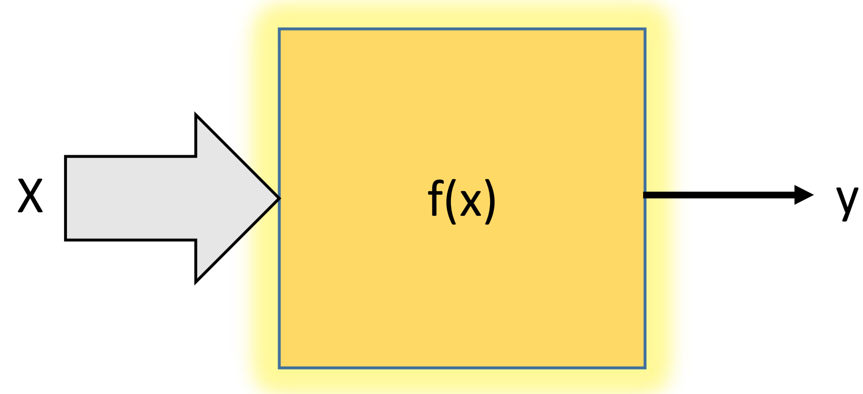


Working with the modified representation



- When the function operates on a vector, it operates on a linear combination of all possible input values
  - How does this help?
  - Not directly, we need to make some assumptions

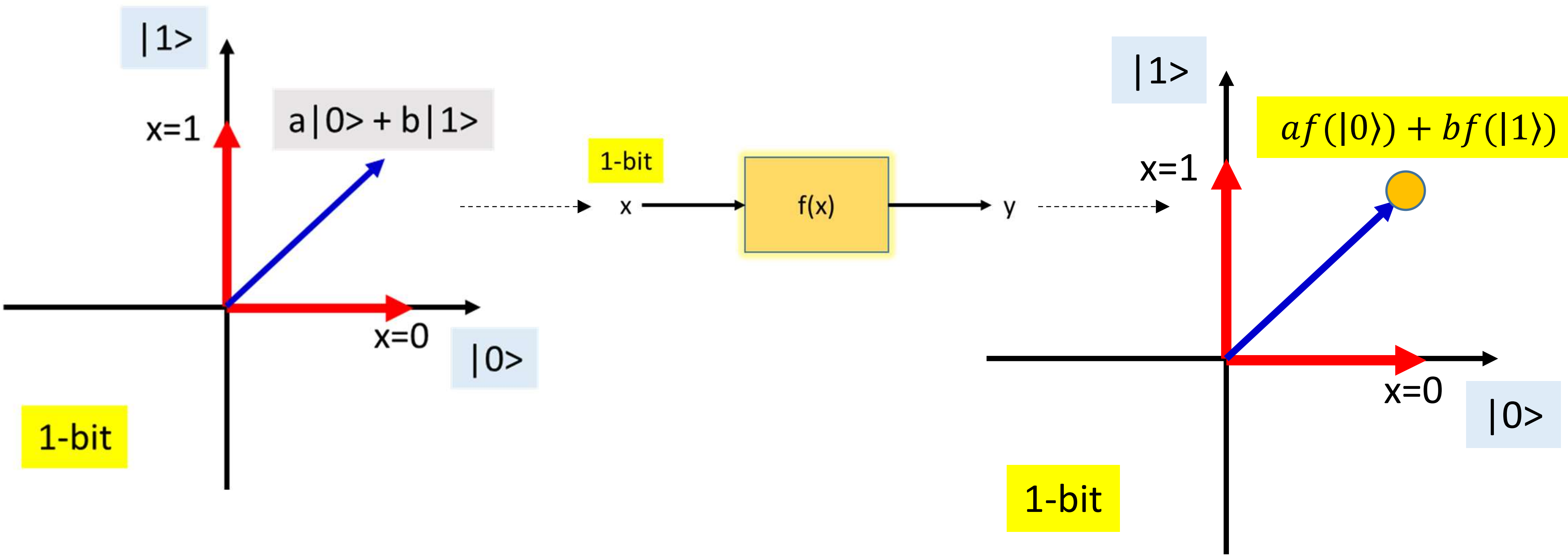
Adjustment 1: Linear functions



- A function $f(x)$ is *linear* if
$$f(ax + by) = af(x) + bf(y)$$
for any two scalars a and b

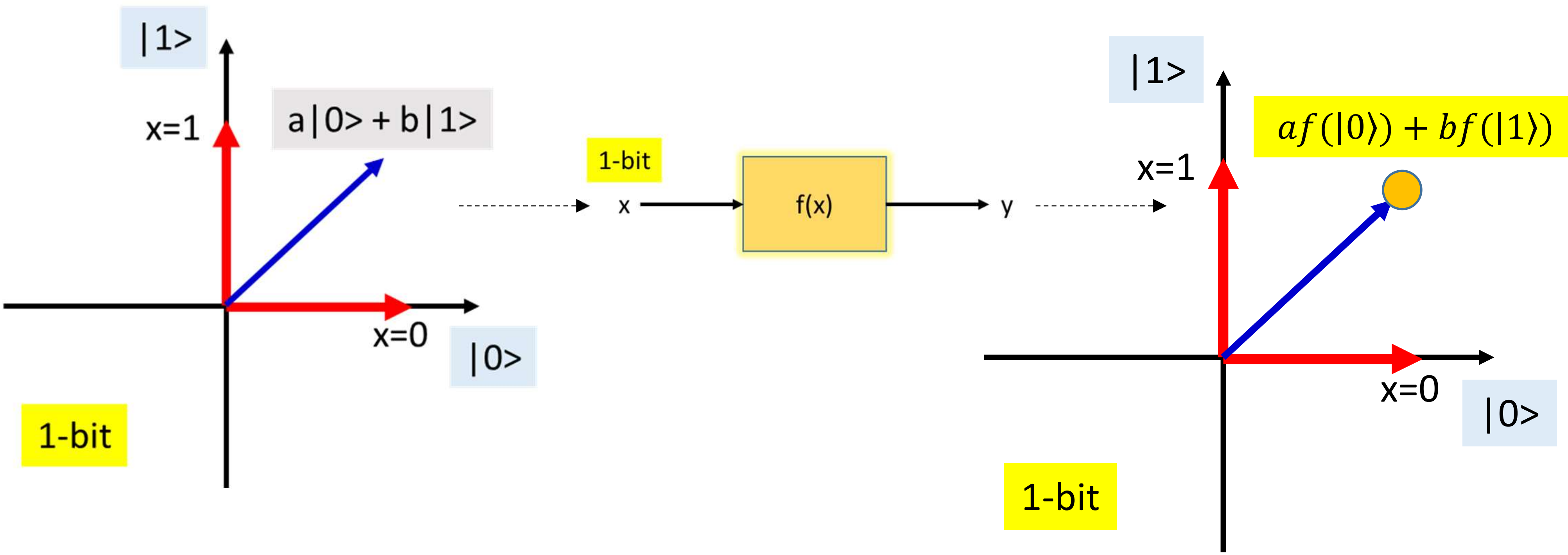
- We assume our function $f(x)$ to be linear
  - As it happens, Boolean functions can always be cast as linear operators

For linear f(x)



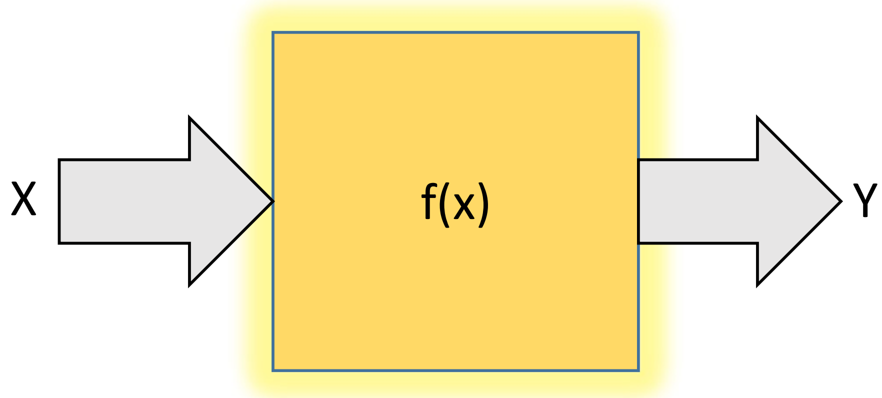
- For linear f(x), the output is a linear combination of the outputs for the individual bit patterns!
- By simply measuring the output at a *single* input vector, we obtain the combined outputs for *all* input bit patterns!
  - *One evaluation!!* (As opposed to 2^N)
- But are we done yet?

For linear f(x)



- The combined output for the individual bit patterns doesn't tell us what the output is for any *single* bit pattern
  - I.e you can't divine $f(|0\rangle)$ and $f(|1\rangle)$ from $af(|0\rangle) + bf(|1\rangle)$
- We need the output to maintain the distinction
  - The function must separately compute the output for each input combination and keep the answers distinct

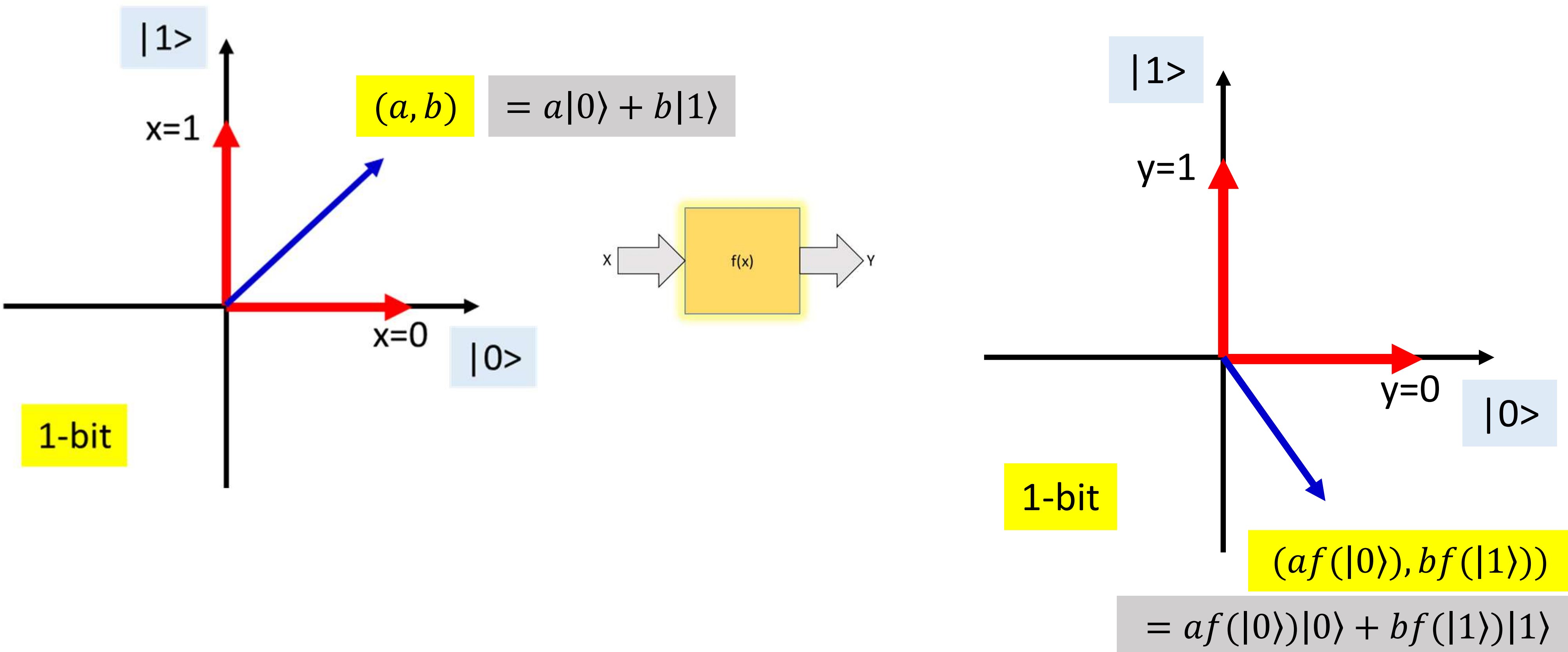
Adjustment 2: Vector functions



- The output too is a vector

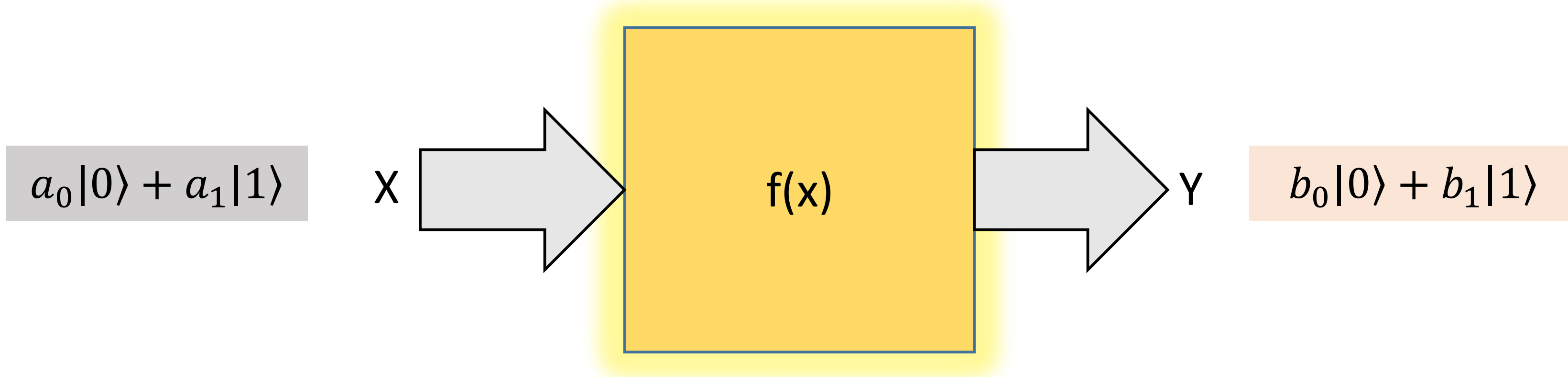
$$f(a|0\rangle + b|1\rangle) = af(|0\rangle)|0\rangle + bf(|1\rangle)|1\rangle$$

For linear f(x)



- Instead of merely computing a value at the vector, the function moves the vector to a new position, where the individual components represent the responses to individual bit patterns
  - Amazingly enough, every Boolean function can be recast in this manner

Adjustment 2: Vector functions



- $f(.)$is a linear transform

$$f \circ \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

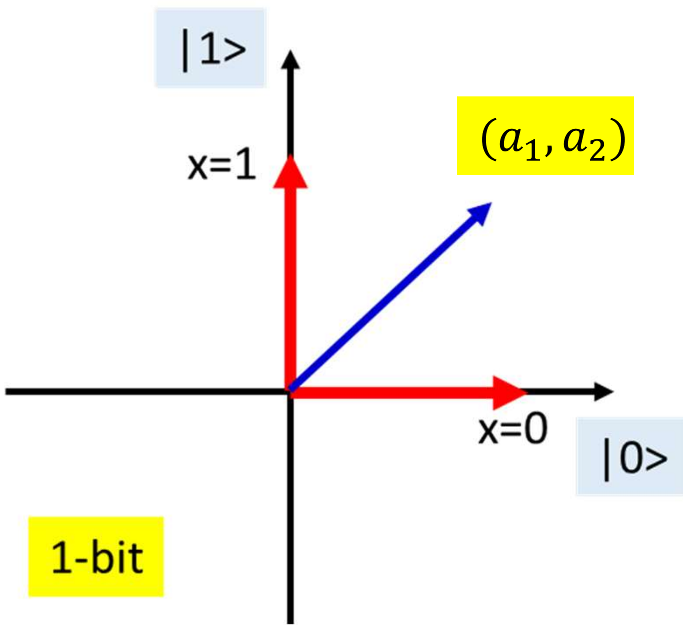
- More generally:

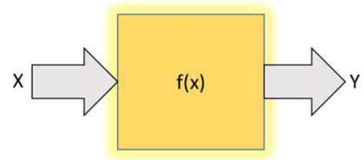
$$f \circ \begin{bmatrix} a_{00\ldots0} \\ a_{00\ldots1} \\ \vdots \\ a_{11\ldots1} \end{bmatrix} = \begin{bmatrix} b_{00\ldots0} \\ b_{00\ldots1} \\ \vdots \\ b_{11\ldots1} \end{bmatrix}$$

In other words $f()$ is a matrix operator.

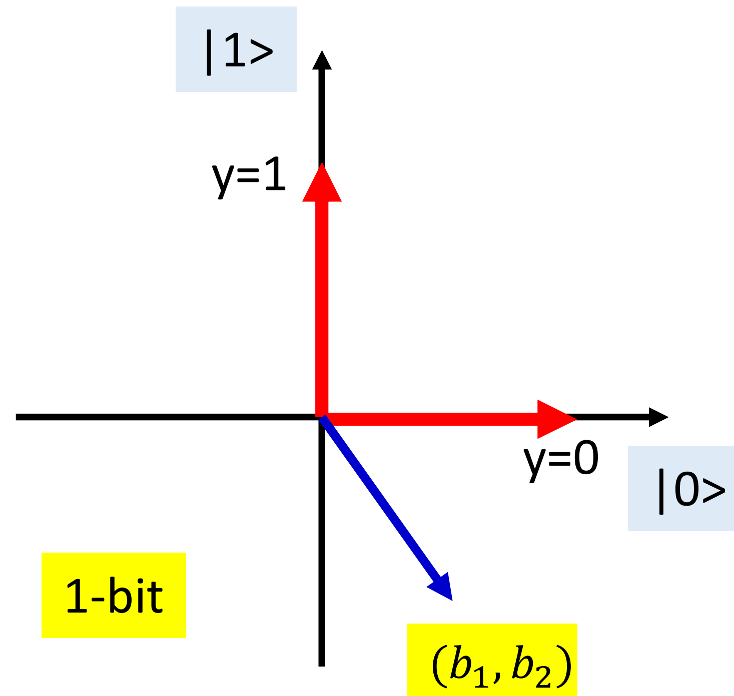
Note that this is a linear operation

For f(x) to retain information



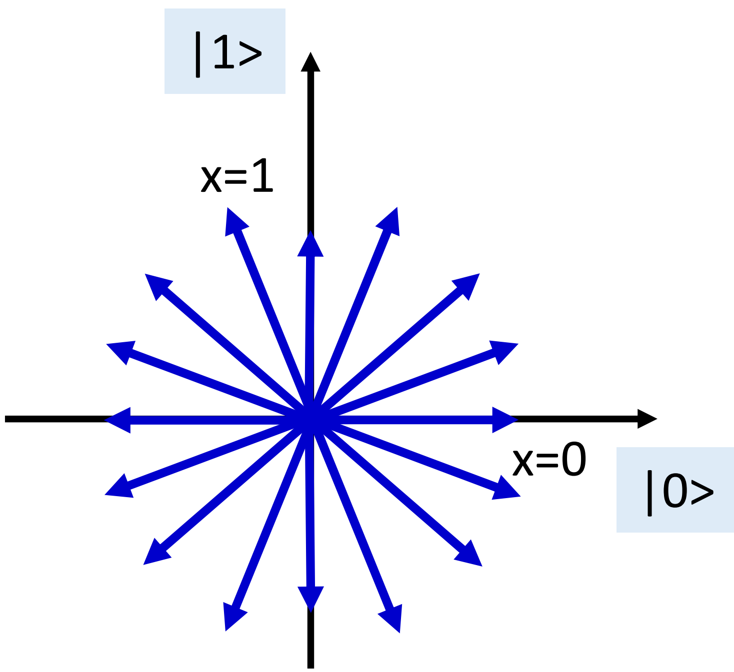


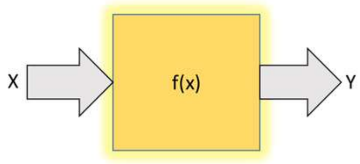
$$Fa = b$$



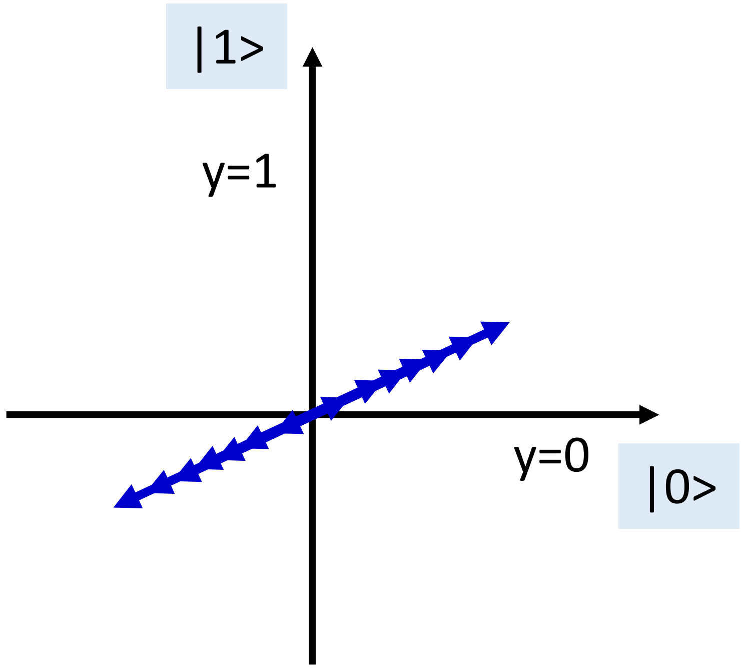
- The function is a linear transform that transforms an input vector to an output vector
  - In the process, it determines the output for *every input bit pattern* in *one step!!*
  - A single computation uncovers the entire function!
  - But there is one more requirement… (what)?

For f(x) to retain information



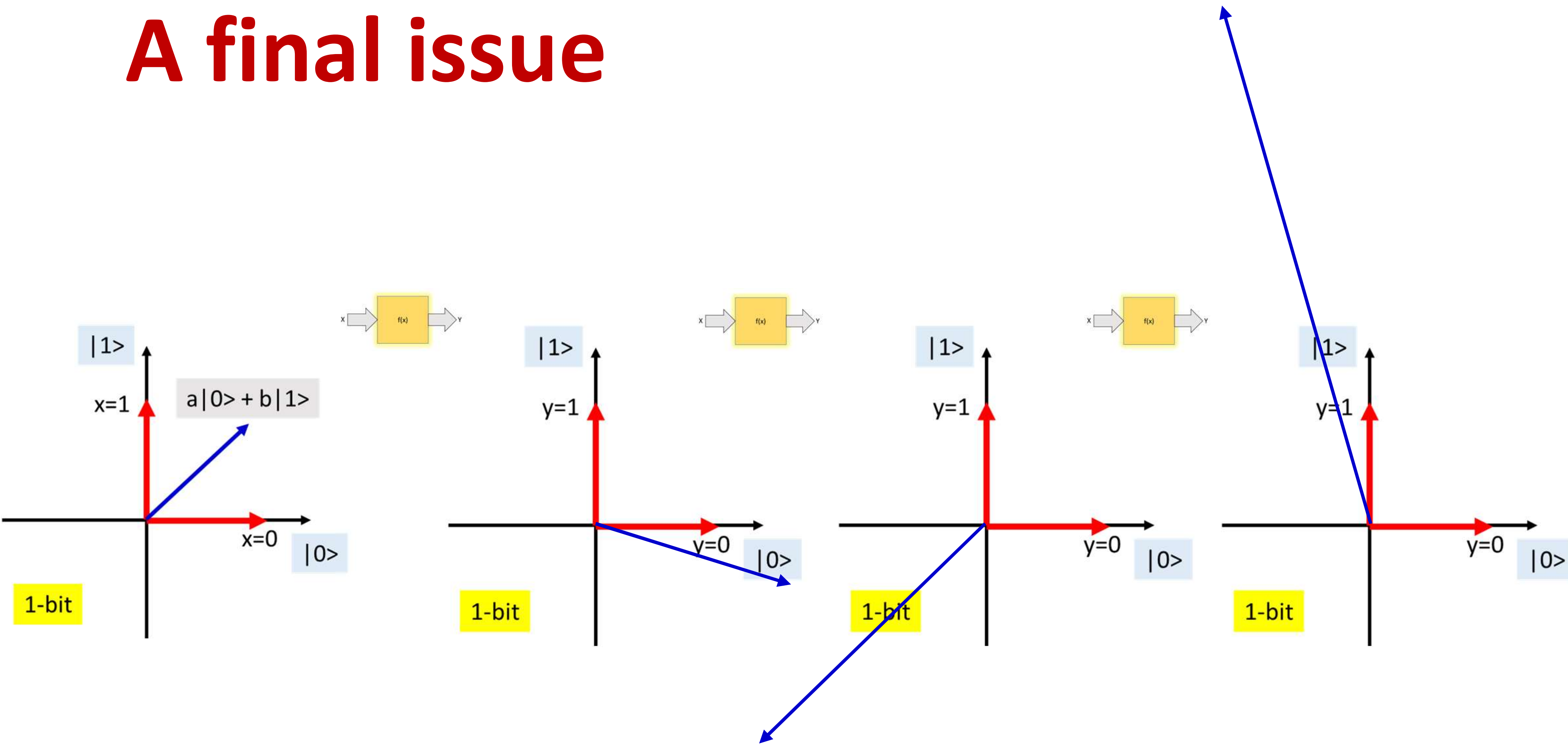


$$Fa = b$$



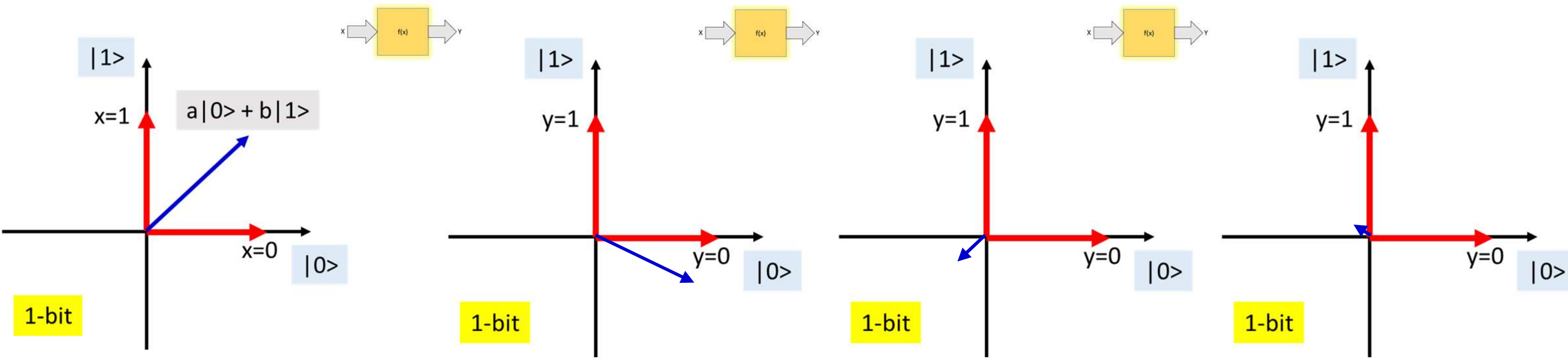
- F must be full rank
  - Otherwise, we cannot recover the contribution of all components of the input vector
    - I.e. we cannot resolve the contributions of all bit patterns to the result
- In other words, F must be *invertible!!*

A final issue



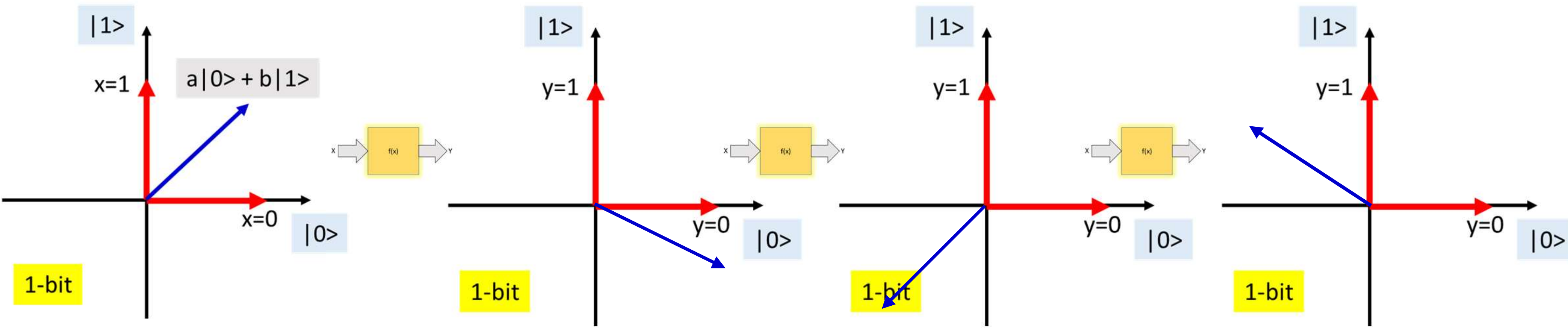
- Repeated applications of linear transforms can make a vector longer and longer and blow up…

A final issue



- Repeated applications of linear transforms can make a vector longer and longer and blow up…
- … or shrink and vanish

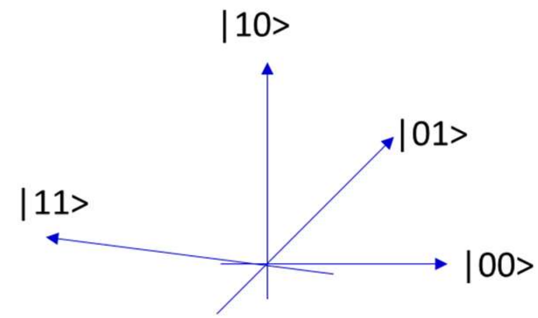
The solution



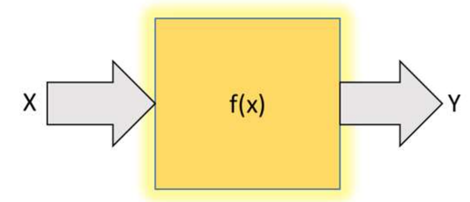
- F must not change the length of the vector
- i.e. it must be a *unitary* transform!
    - Actually, a *rotation* transform as we shall see.

The new paradigm

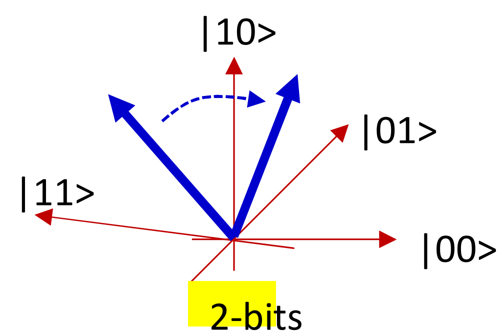
- Modified representation: Every bit *pattern* is an orthogonal direction of the input space
  - A vector in this space is a linear combination of all bit patterns



- The function being computed is representable as an invertible linear transform
  - More specifically a rotation



- Evaluation of the function on a single vector can compute the output for every possible bit pattern, in an identifiable way



Poll 3a

- What are the key differences between the classical and new paradigms
    a) In the classical paradigm, each bit is a unique dimension of the representation, in the new paradigm bit *patterns* are the unique dimensions
    b) In the classical paradigm any vector in the space is a valid input to the function, in the new paradigm only the axes, representing unique bit patterns, are valid inputs
    c) In the classical paradigm only the vectors representing the corners of the hypercube are valid inputs to the function, whereas in the new paradigm any vector is valid
    d) In the classical paradigm the function f(x) must be invertible, whereas in the new paradigm f(x) need not be invertible
    e) In the classical paradigm f(x) is unrestricted, whereas in the new paradigm f(x) must be linear and invertible

Poll 3a

- What are the key differences between the classical and new paradigms
  a) **In the classical paradigm, each bit is a unique dimension of the representation, in the new paradigm bit *patterns* are the unique dimensions**
  b) In the classical paradigm any vector in the space is a valid input to the function, in the new paradigm only the axes, representing unique bit patterns, are valid inputs
  c) **In the classical paradigm only the vectors representing the corners of the hypercube are valid inputs to the function, whereas in the new paradigm any vector is valid**
  d) In the classical paradigm the function f(x) must be invertible, whereas in the new paradigm f(x) need not be invertible
  e) **In the classical paradigm f(x) is unrestricted, whereas in the new paradigm f(x) must be linear and invertible**

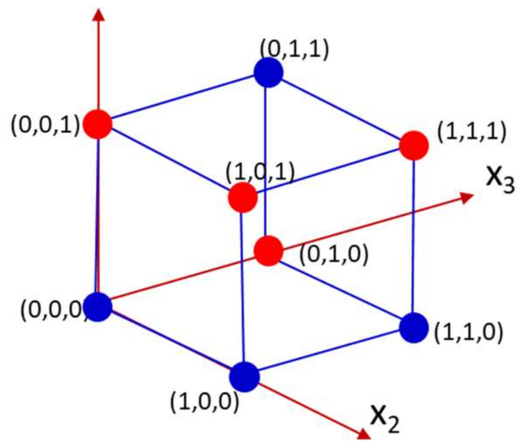
Poll 3b

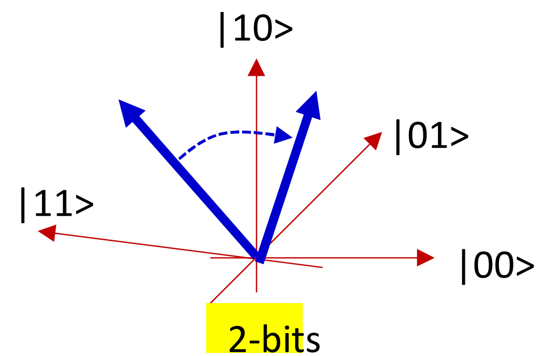
- Which of the following are representations from the new math. Here “|0>” represents a single bit taking value 0, “|01>” represents a bit pair taking the values 0 and 1 respectively
  - 0
  - a |0> + b |1>
  - 11
  - a |00> + b |01> + c |10> + d |11>

Poll 3b

- Which of the following are representations from the new math. Here “|0>” represents a single bit taking value 0, “|01>” represents a bit pair taking the values 0 and 1 respectively
  - 0
  - **a |0> + b |1>**
  - 11
  - **a |00> + b |01> + c |10> + d |11>**

Old vs. the new paradigm

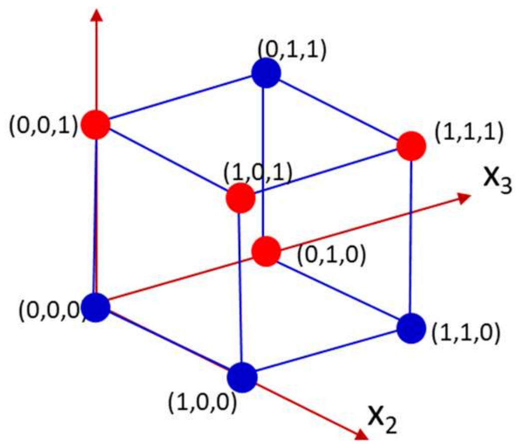


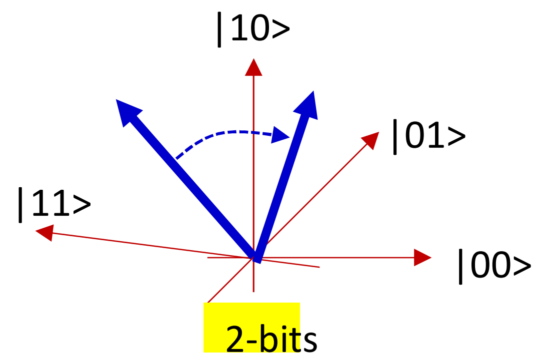


- Each bit is a coordinate dimension
- Must explicitly evaluate function for every input to fully determine it
- In reverse: Given only the output, must evaluate every input to determine which one generated it
  - 2^Ncomputations

- Each bit pattern is a coordinate dimension
- A single evaluation fully determines the function
- In reverse: Given an output, determining which input produced it is a single-step computation
  - Because computation is reversible

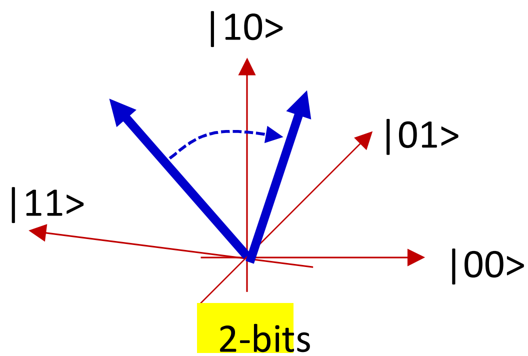
Where is this useful





- Satisfiability problems
  - Does any bit pattern produce the output 1
- Search problems
  - Is any bit pattern in my library exactly equal to 11001010
    - Equivalent to SAT problems
- Combinatorial optimization problems
- ….
- Any problem that can be set as a SAT problem

What are the practical issues?



- Is this practically realizable?

Poll 4

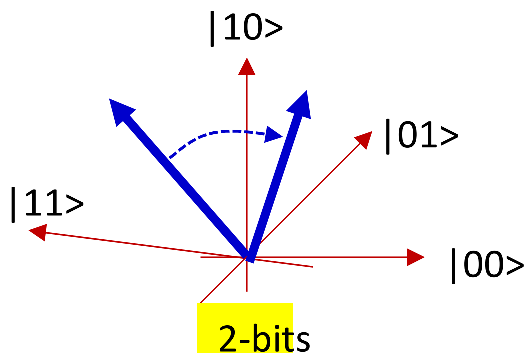
- In the new match, how many numbers are required to represent an input, for a function of 100 bits
  - 100
  - 100^2
  - 2^{100}
  - Insufficient information to decide

Poll 4

- In the new match, how many numbers are required to represent an input, for a function of 100 bits
  - 100
  - 100^2
  - **2^{100}**
  - Insufficient information to decide

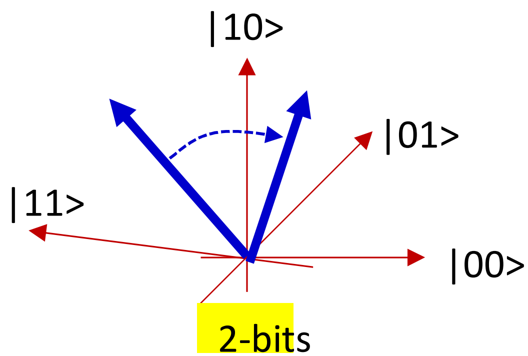
- **Since each of the 2^{100} bit patterns is an independent axis, and a vector has as many components as axes, you need 2^{100} numbers to represent an input**

What are the practical issues?



- Is this practically realizable?
- A conventional (classical) computer requires a *single* N-bit register to represent a value
  - A function takes in a single N-bit value and produces a single bit
- The new representation requires 2^N numbers to represent a single value!
  - For even N=100 bits, this requires ~1000000000000000000000000000000 numbers
  - A function take in 2^{100} values and produces 2^{100} values
    - I.e it's a 2^{200}-valued transform!
  - Which is why its not a very *useful* way of thinking about things
- Is there a *parsimonious* way of representing a 2^{100} component vector without taking up the entire universe?

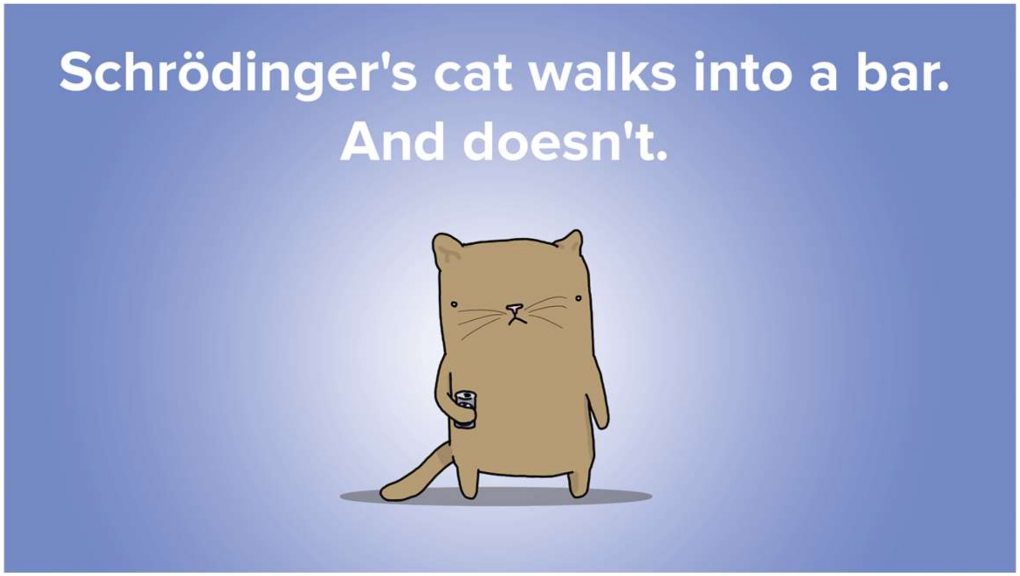
What are the practical issues?



Fact that may only interest me: "Graham's number" is a number that's so large there isn't enough space in the universe to write it..

- Is this practically realizable?
- A conventional (classical) computer requires a *single* N-bit register to represent a value
  - A function takes in a single N-bit value and produces a single bit
- The new representation requires 2^N numbers to represent a single value!
  - For even N=100 bits, this requires ~1000000000000000000000000000000 numbers
  - A function take in 2^{100} values and produces 2^{100} values
    - I.e it's a 2^{200}-valued transform!
  - Which is why its not a very *useful* way of thinking about things
- Is there a *parsimonious* way of representing a 2100 component vector without taking up the entire universe?

Enter.. The cat!!



- Introducing Quantum, the cat

The world according to Schroedinger

$$ih\frac{d}{dt}|\psi(t)\rangle = 2\pi H|\psi(t)\rangle$$

- The magical formula that represents the quantum number 42

The world according to Schroedinger

$$ih\frac{d}{dt}|\psi(t)\rangle = 2\pi H|\psi(t)\rangle$$

- The magical formula that represents the quantum number 42
- The wave function!
  - For any physical system you have a wave function and a Hamiltonian
    - Which you could design

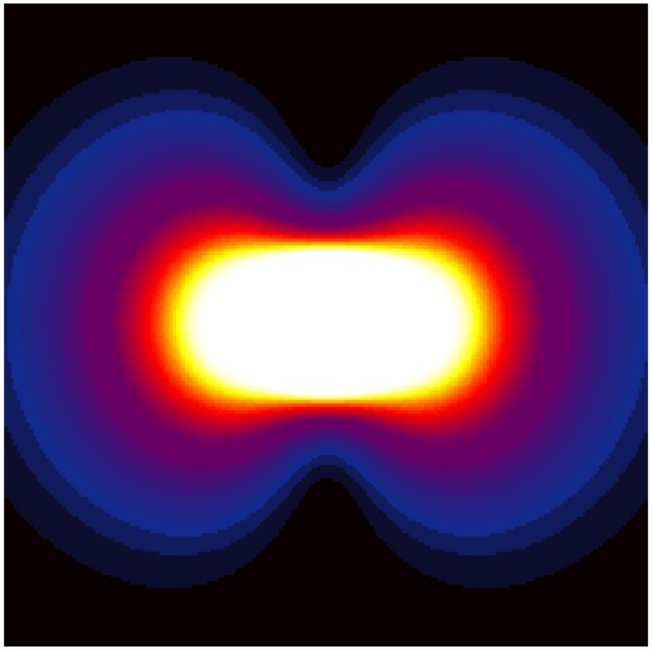
The wave function for any particle predicts its probability

$$\psi(t, x)$$

$|\psi(t, x)|^2$ is the probability that the system will be in configuration x at time t

- What does this mean

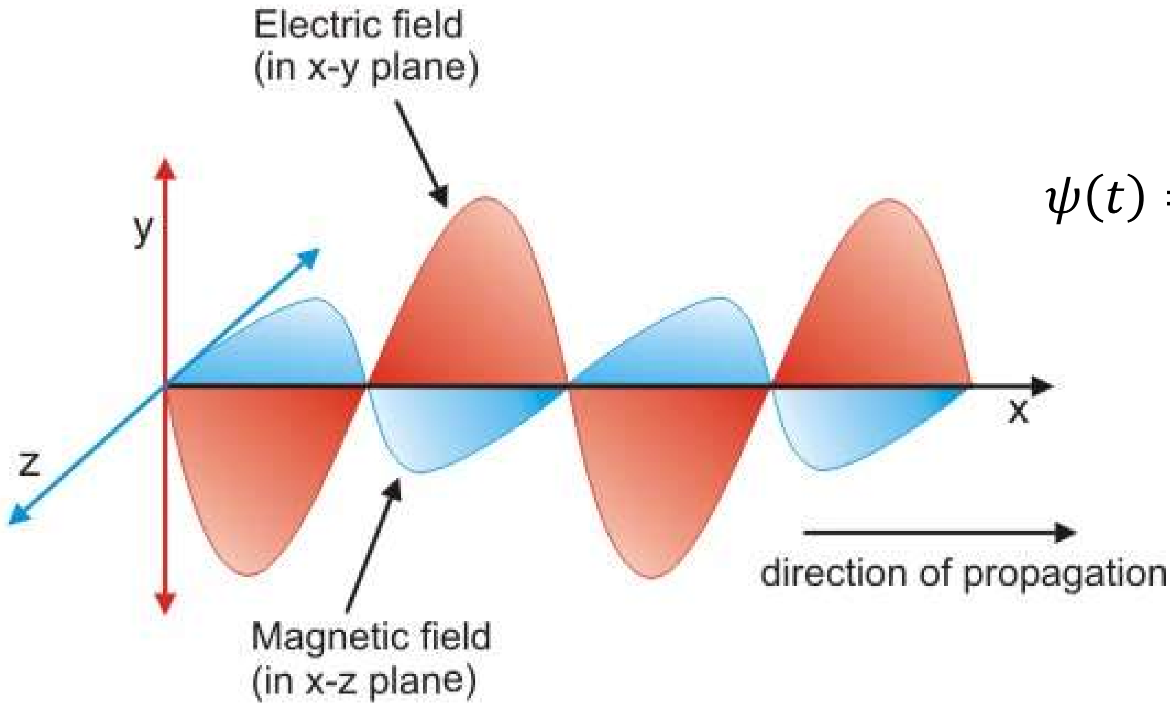
The hydrogen atom



$$\psi(t) = \alpha(t)|0\rangle + \beta(t)|1\rangle$$

- The probability of the electron in the hydrogen molecule

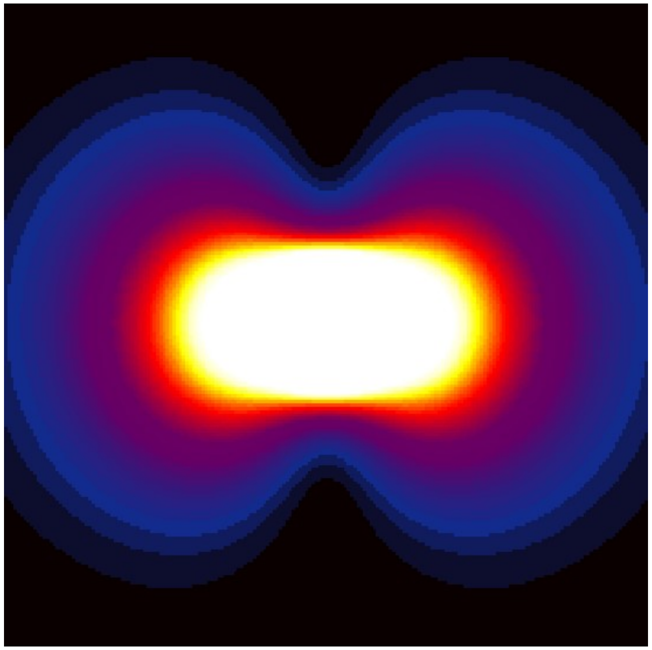
The polarization of light

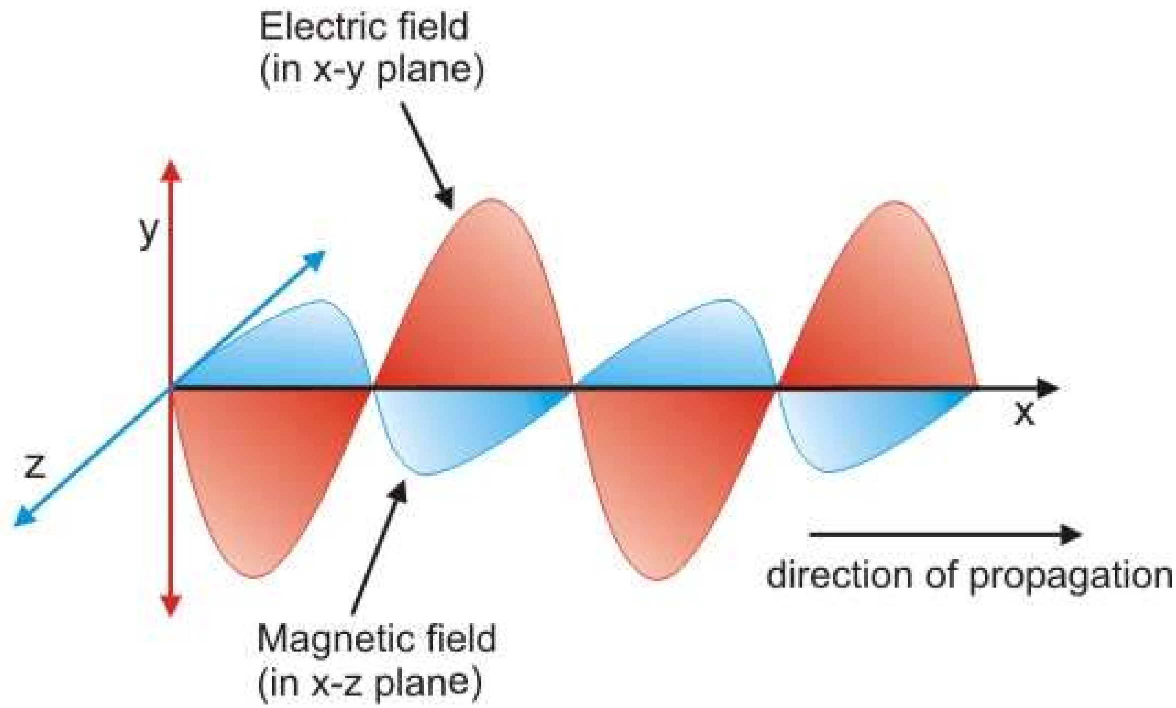


$$\psi(t) = \alpha(t)|0\rangle + \beta(t)|1\rangle$$

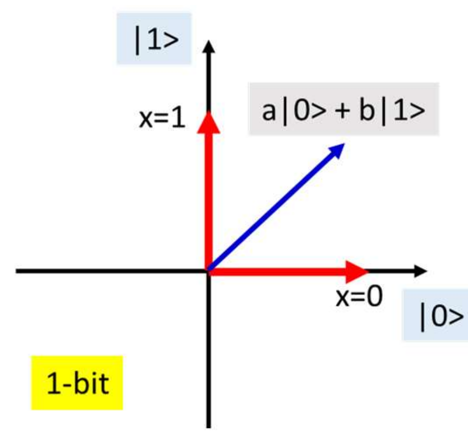
- How do polarizing glasses work?
  - Send through the component of the E/M fields that are aligned with the polarizer
- The light actually exists in both polarizations at the same time

Quantum systems



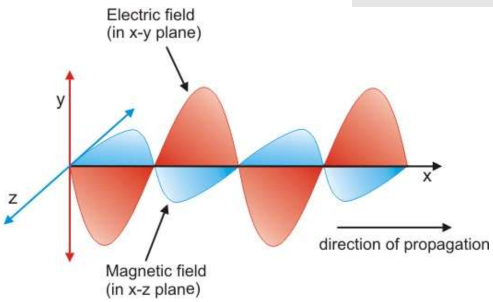


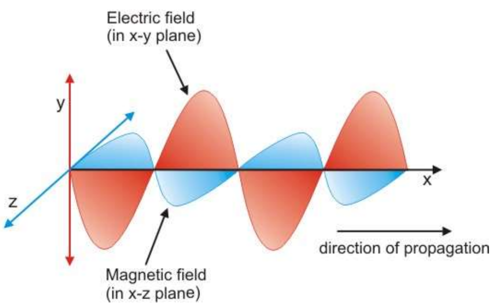
- Quantum systems naturally exist in a superposition of multiple values
  - If we assign each value to a bit value (or bit pattern), we get a quantum computing platform

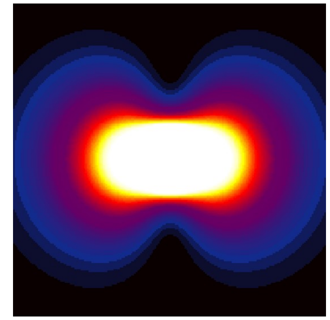


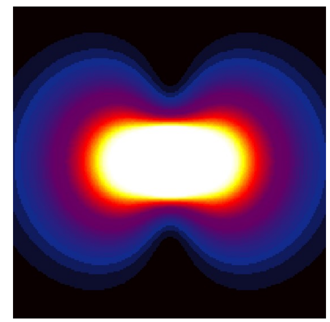
Multiple bits

$a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle$

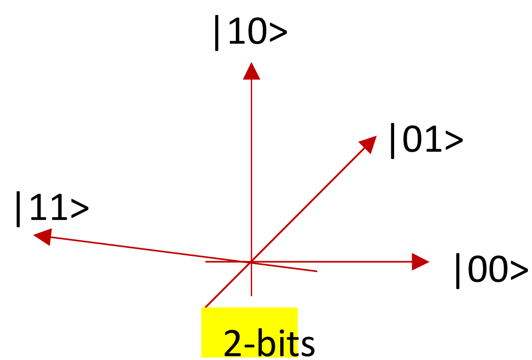




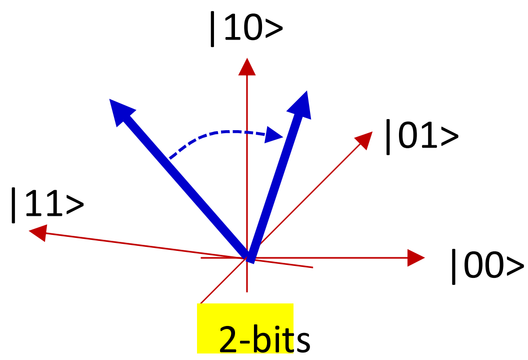




- Increasing the number of bits only takes increasing the number of basic quantum units



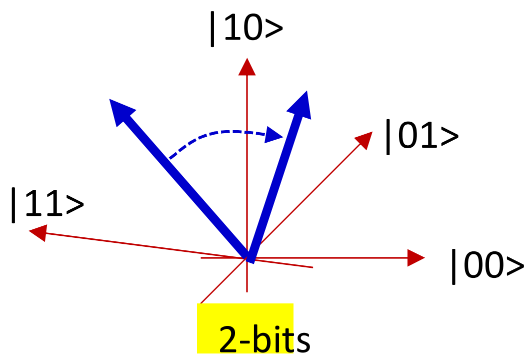
Practical implementation



- Simply use a collection of quantum bits
  - Will simultaneously represent all states

- What is missing?

Practical implementation



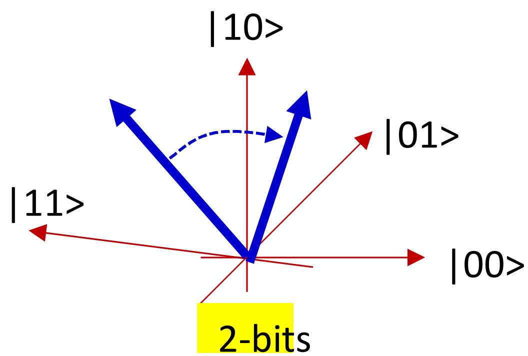
- Simply use a collection of quantum bits
  - Will simultaneously represent all states

- What is missing?
  - How do you implement the functions?
    - Invertible rotations

$$ih\frac{d}{dt}|\psi(t)\rangle = 2\pi H|\psi(t)\rangle$$

But first you must design the functions (we will see how)

Practical implementation

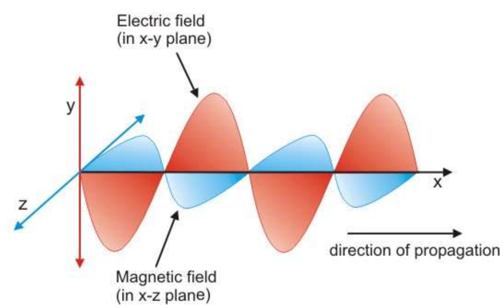


- Simply use a collection of quantum bits
  - Will simultaneously represent all states

- What is missing?
  - How do you implement the functions?
    - Invertible rotations $ih\frac{d}{dt}|\psi(t)\rangle = 2\pi H|\psi(t)\rangle$
  - How do you measure the output vectors?

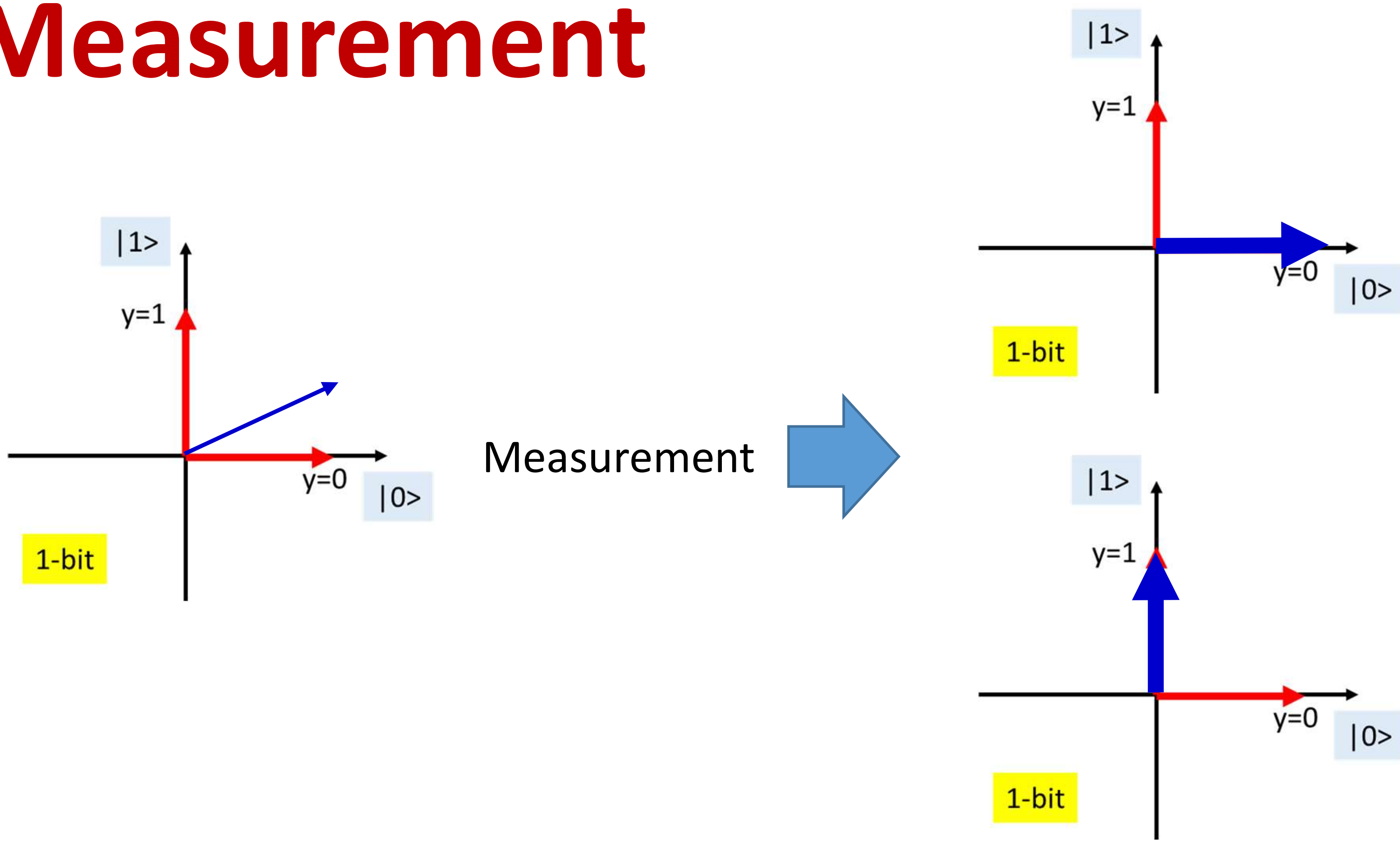
The problem with measurement

- **Reality Doesn't Exist Until We Measure It, Quantum Experiment Confirms**
- https://www.sciencealert.com/reality-doesn-t-exist-until-we-measure-it-quantum-experiment-confirms



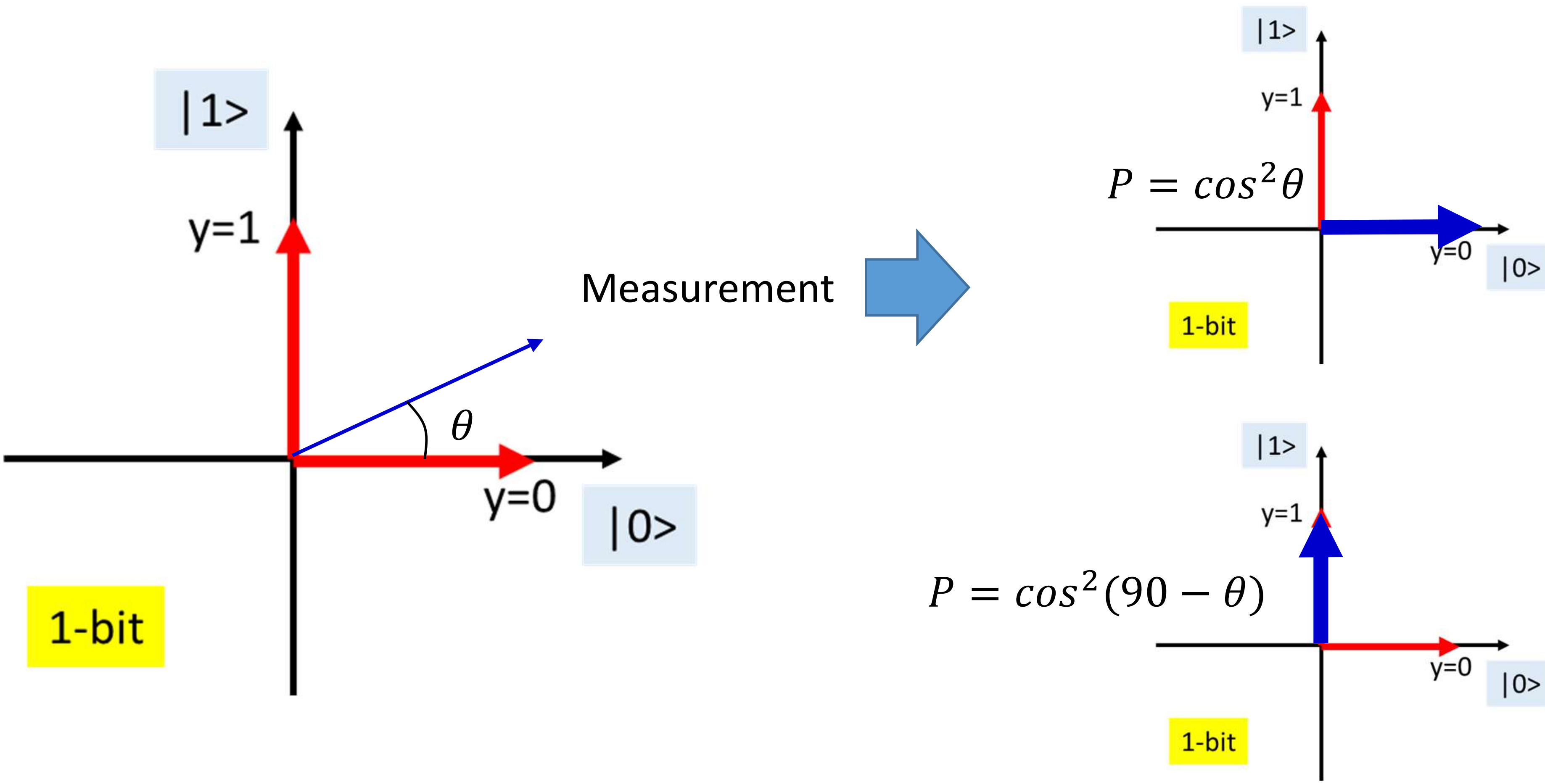
- Measuring a quantum variable "collapses" it

Measurement

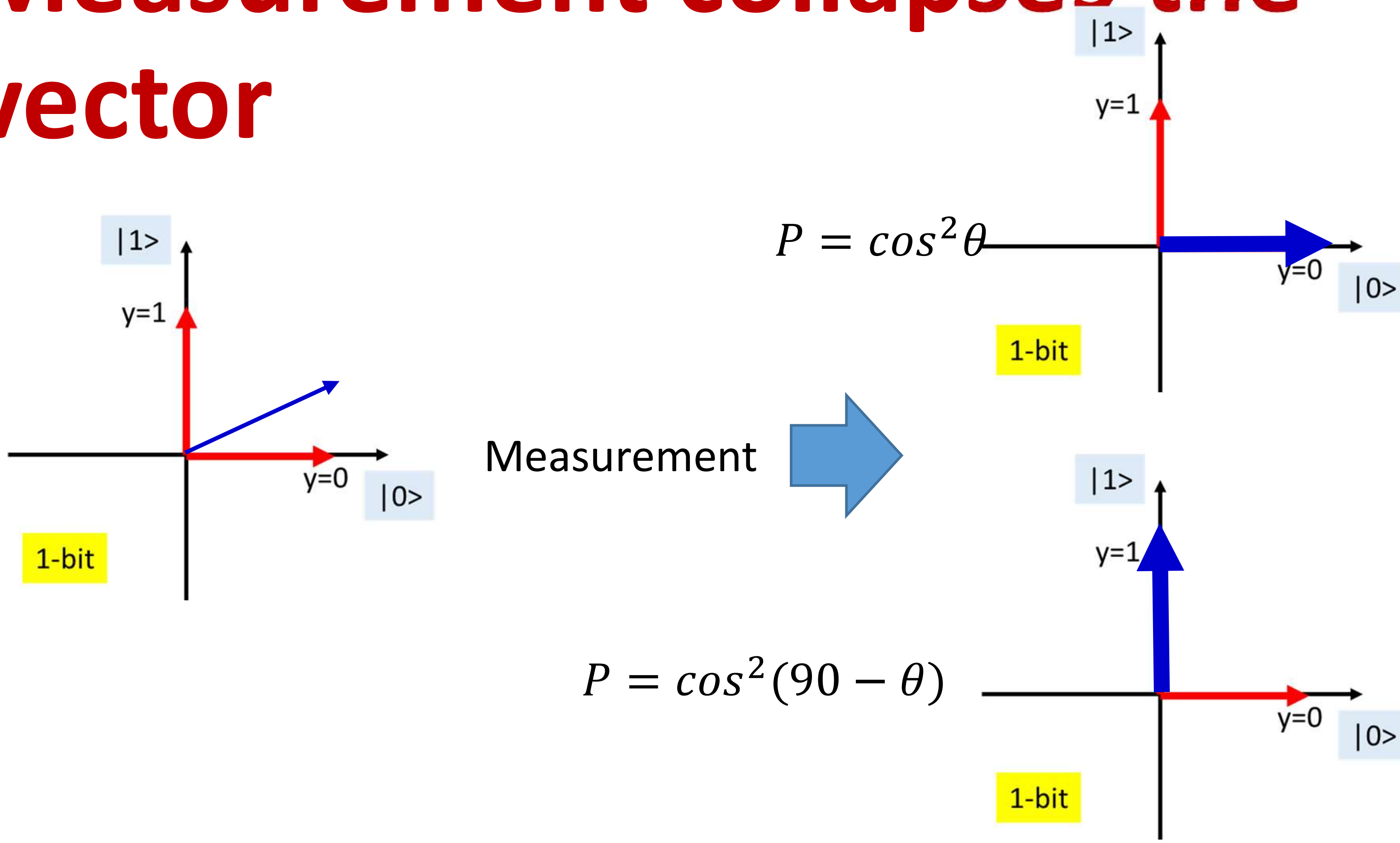


- Measuring the output collapses the vector to one of the states
  - Bit pattern
- Which one

Measurement

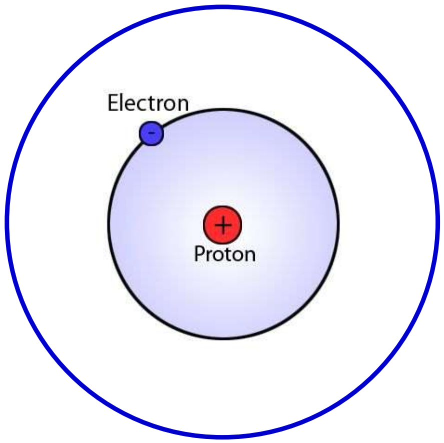


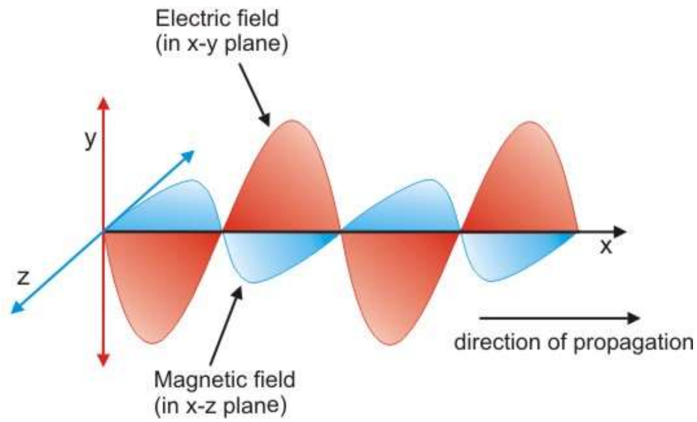
Measurement collapses the vector

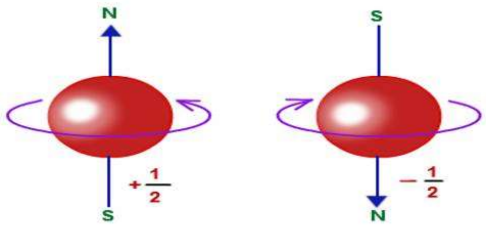


- If you want to recover the vector how many measurements must you take
  - Keeping in mind that each measurement means creating and manipulating “quantum bits” or “qubits” from scratch

The physical Qubit

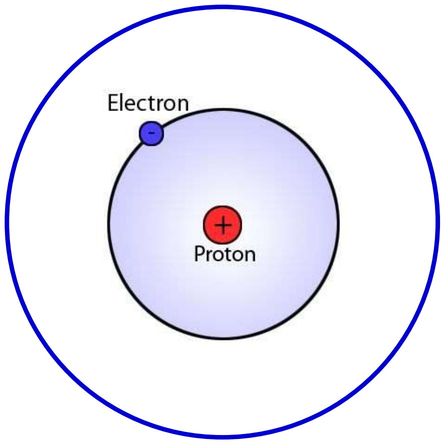


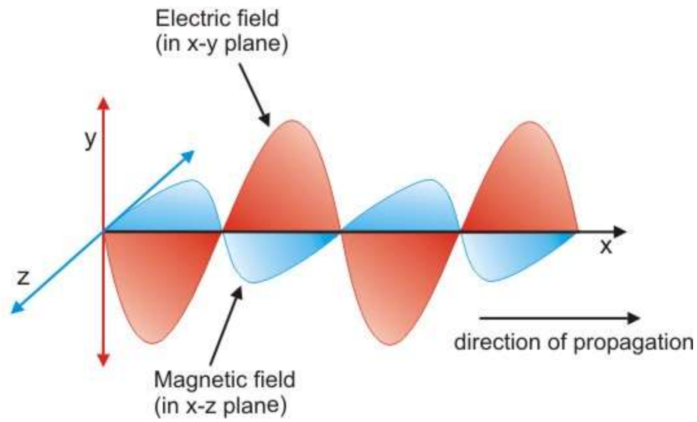


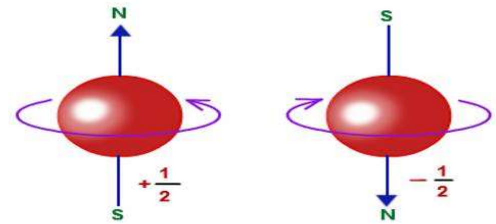


- Hydrogen atom electron:
  - Ground state = |0>, excited state = |1>
- Photon polarization
  - Horizontal = |0>, vertical = |1>
- Electron spin
  - NS = |0>, SN = |1>

The physical Qubit





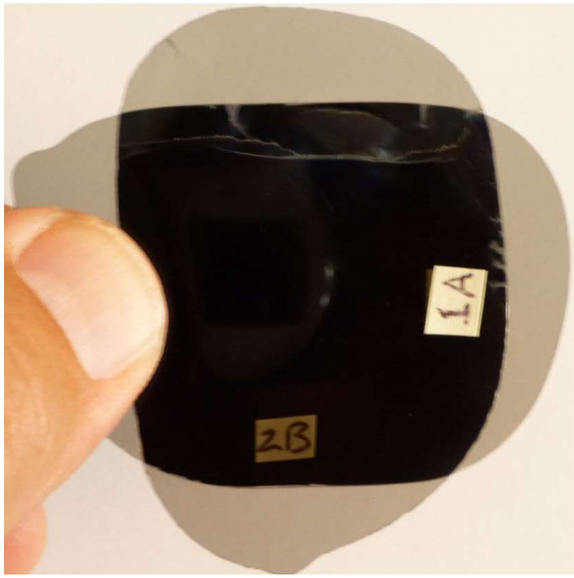


- Hydrogen atom electron:
  - Ground state = |0>, excited state = |1>
- Photon polarization
  - Horizontal = |0>, vertical = |1>
- Electron spin
  - NS = |0>, SN = |1>

Note: The definition of your "bases" is a matter of convention

The physical Qubit

|1>

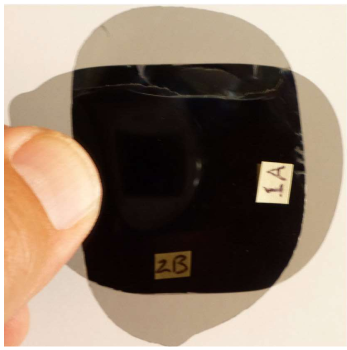


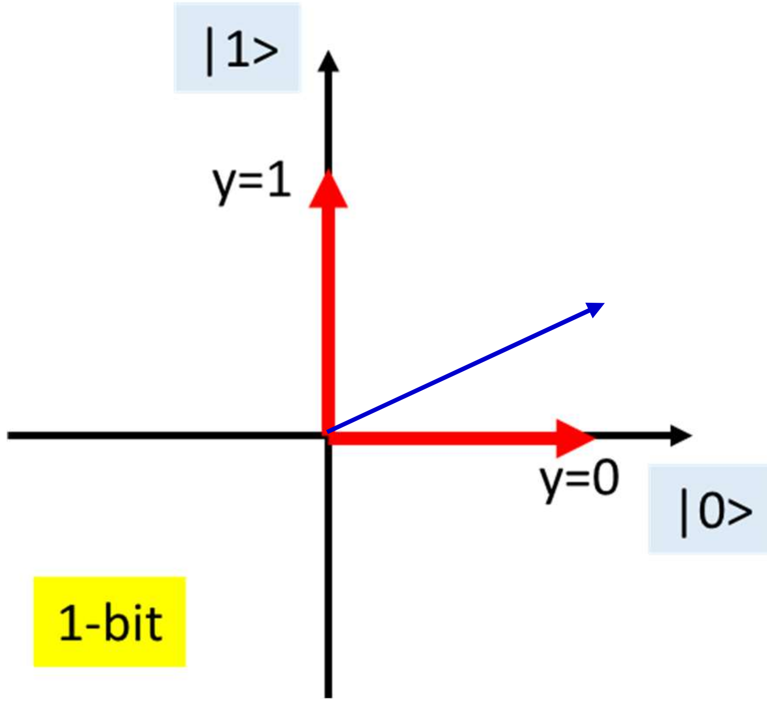
|0>

- Hydrogen atom electron:
  - Ground state = |0>, excited state = |1>
- Photon polarization
  - Horizontal = |0>, vertical = |1>
- Electron spin
  - NS = |0>, SN = |1>

Note: The definition of your "bases" is a matter of convention
The only requirement is that they are at right angles to one another.

The physical Qubit

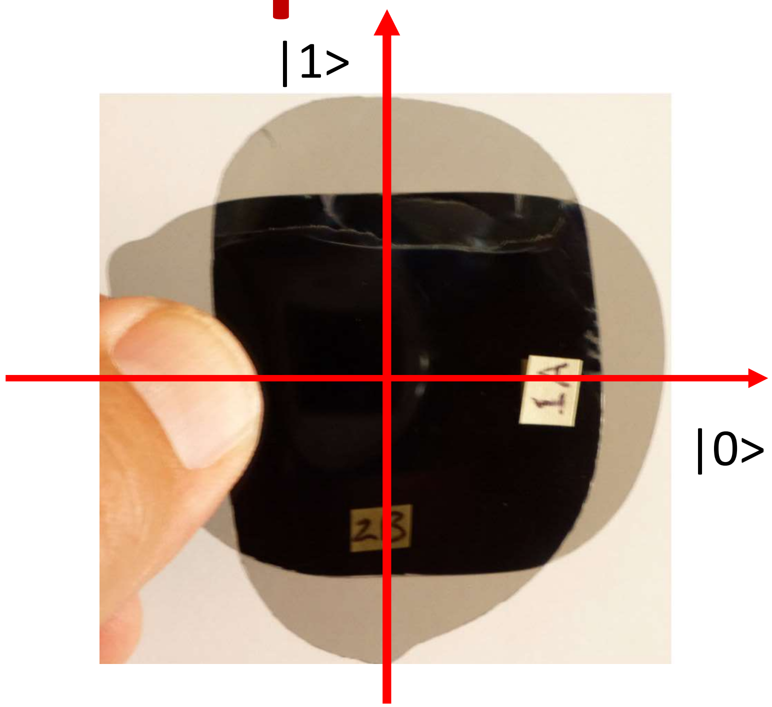


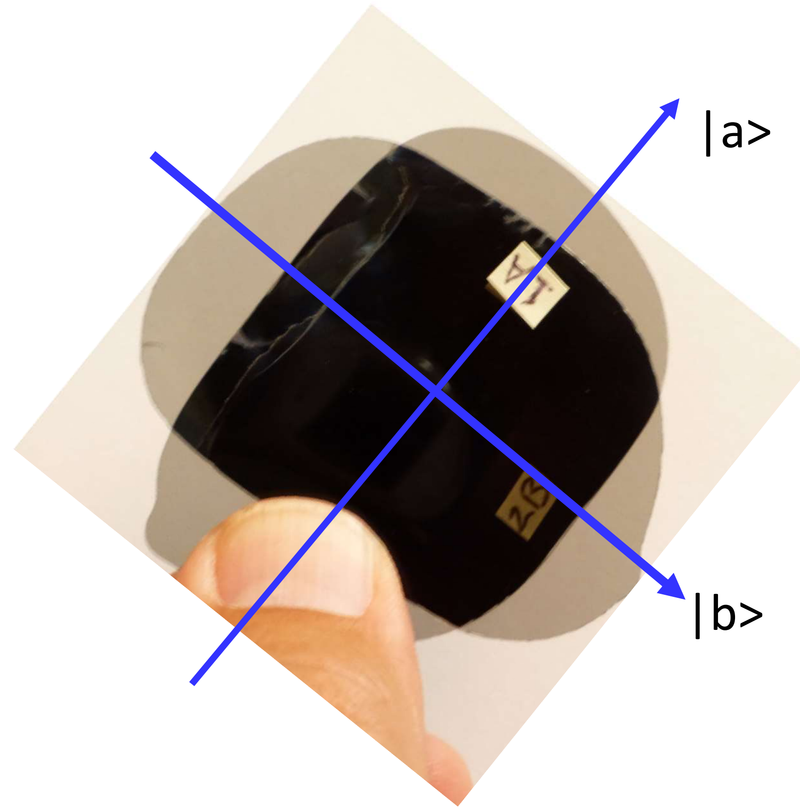


- Hydrogen atom electron:
  - Ground state = |0>, excited state = |1>
- Photon polarization
  - Horizontal = |0>, vertical = |1>
- Electron spin
  - NS = |0>, SN = |1>

Note: The definition of your "bases" is
a matter of convention
The only requirement is that they are
at right angles to one another.

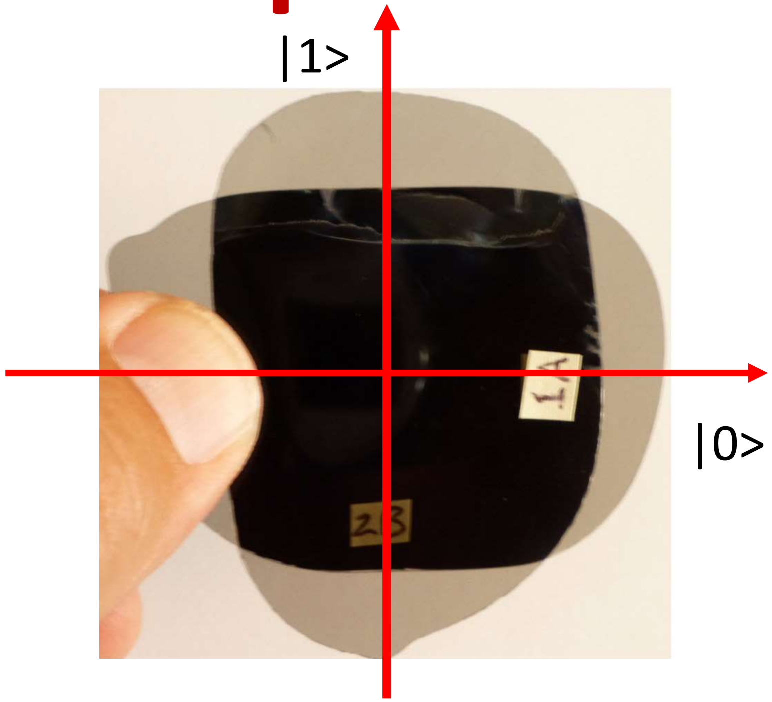
Multiple bases

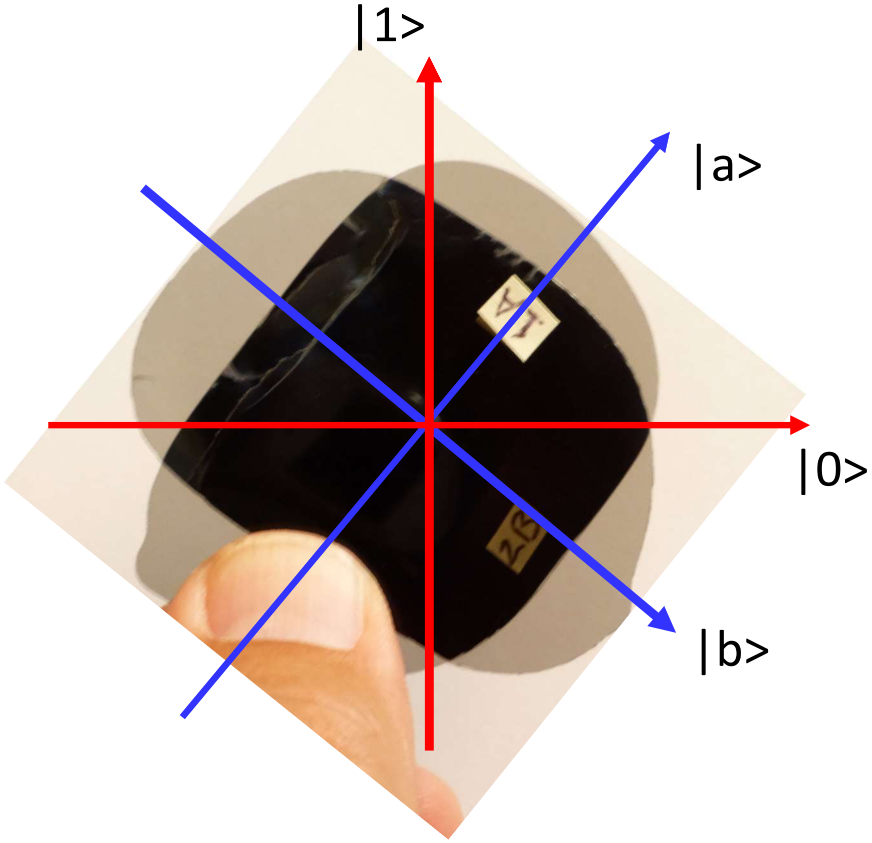




- The two sets of bases shown here are both valid bases, but they are not the same
- If we (quite arbitrarily) designate the first set of bases (on the left) as |0> and |1>, then the second set of bases on the right will require a different designation

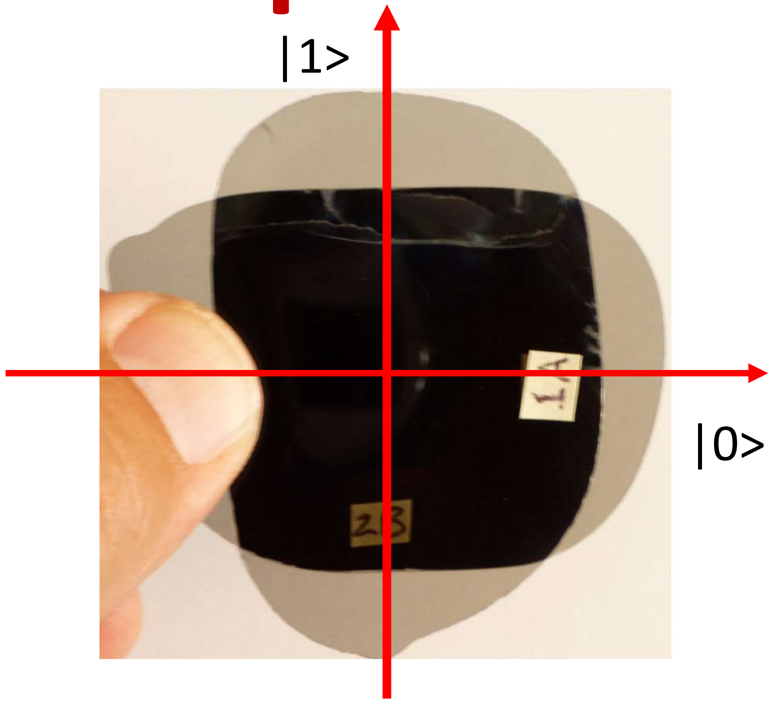
Multiple bases

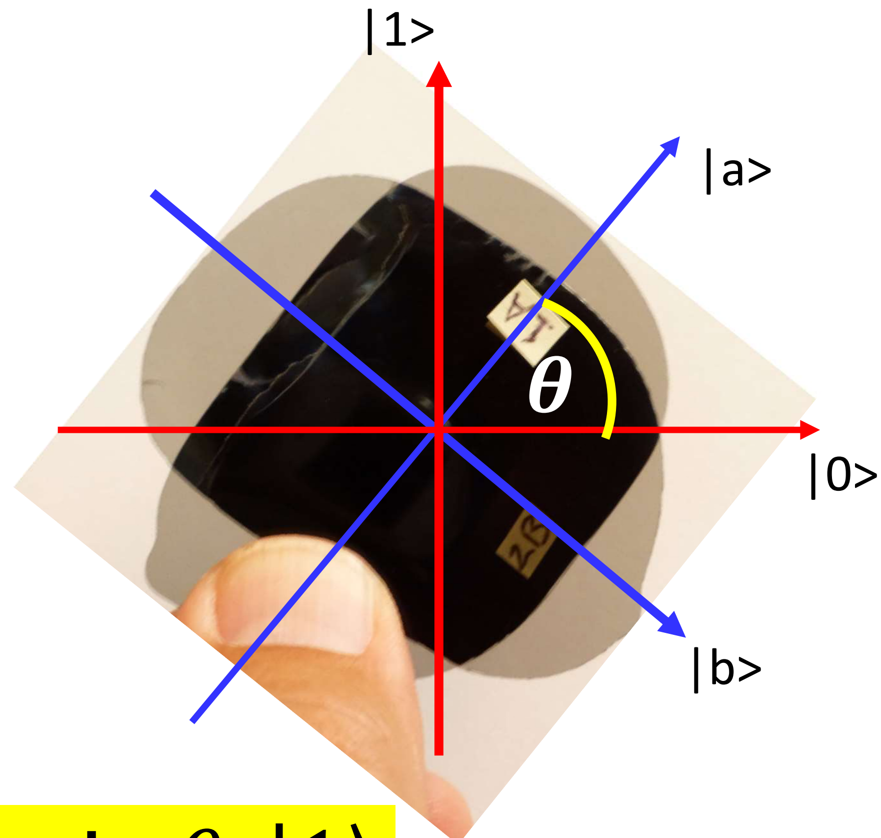




- The two sets of bases shown here are both valid bases, but they are not the same
- If we (quite arbitrarily) designate the first set of bases (on the left) as |0> and |1>, then the second set of bases on the right will require a different designation
- One set of bases can, in fact, be specified in terms of the other!

Multiple bases





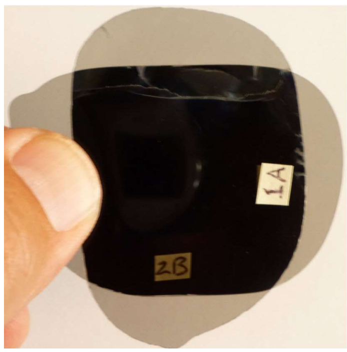
- The two set ut they are not the same

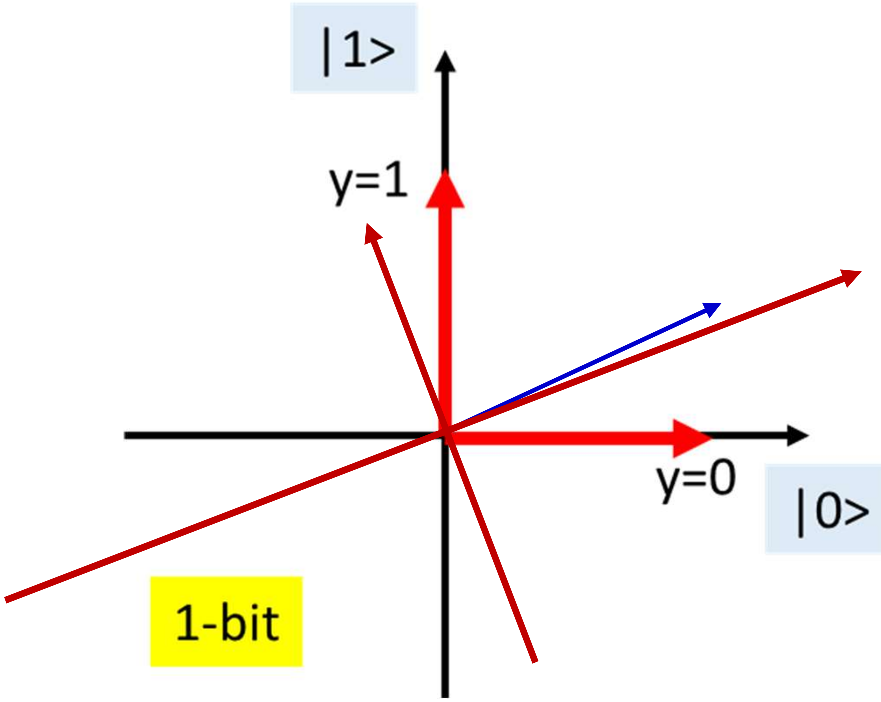
$$|a\rangle = \cos\theta\ |0\rangle + \sin\theta\ |1\rangle$$

$$|b\rangle = \sin\theta\ |0\rangle - \cos\theta\ |1\rangle$$

- If we (quite the left) as |0> and |1>, then the second set of bases on the right will require a different designation
- One set of bases can, in fact, be specified in terms of the other!

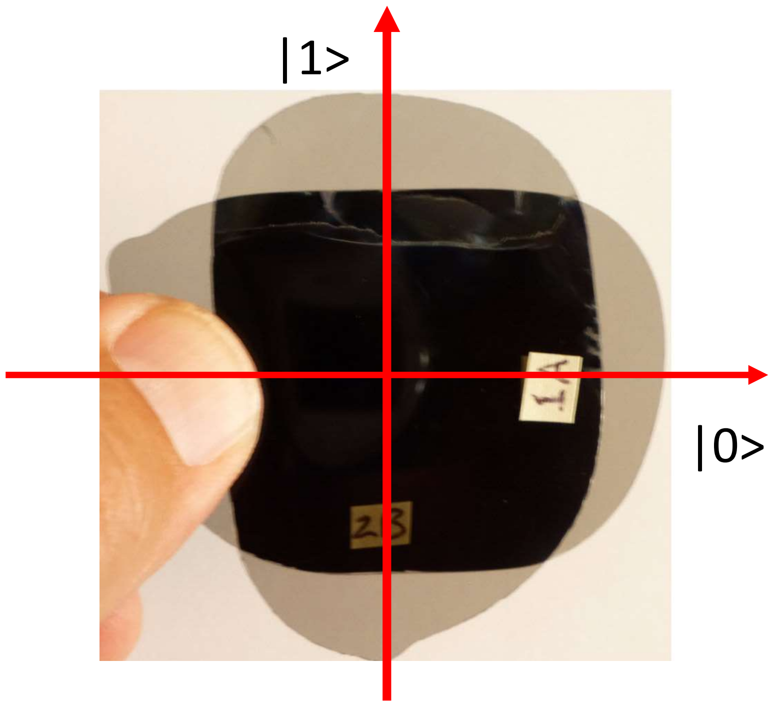
The physical Qubit

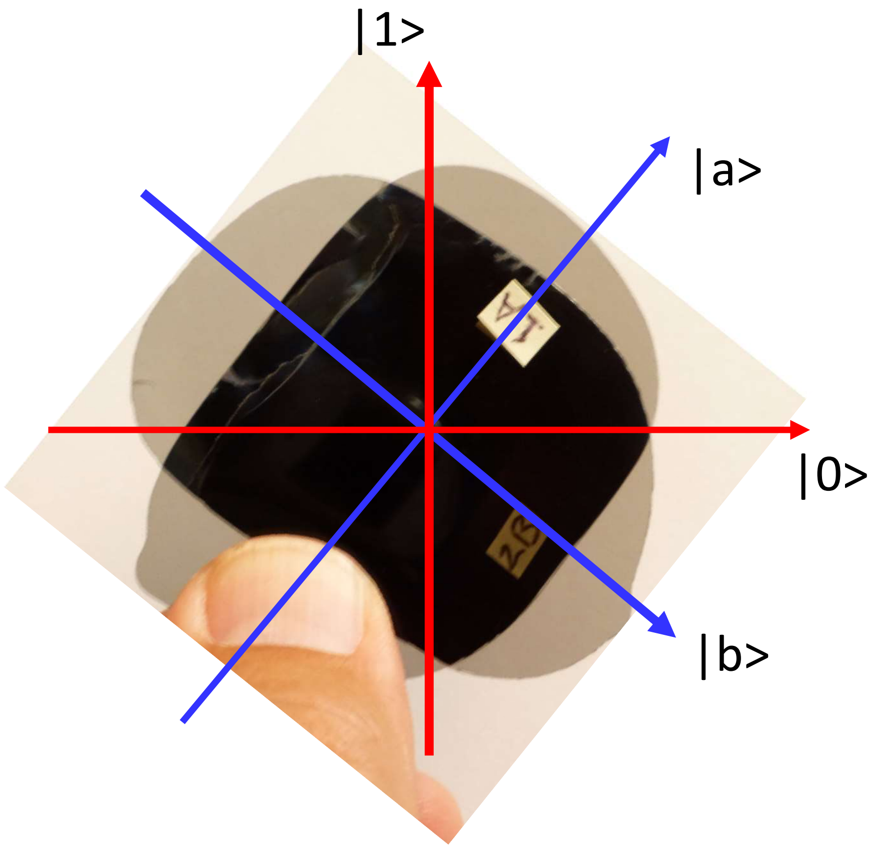




- The definition of your “bases” is a matter of convention
- The only requirement is that they are at right angles to one another.
- The representation of the vector will, obviously, depend on the bases

BASES

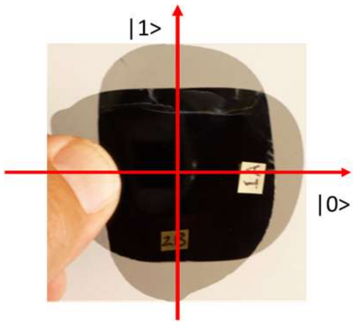


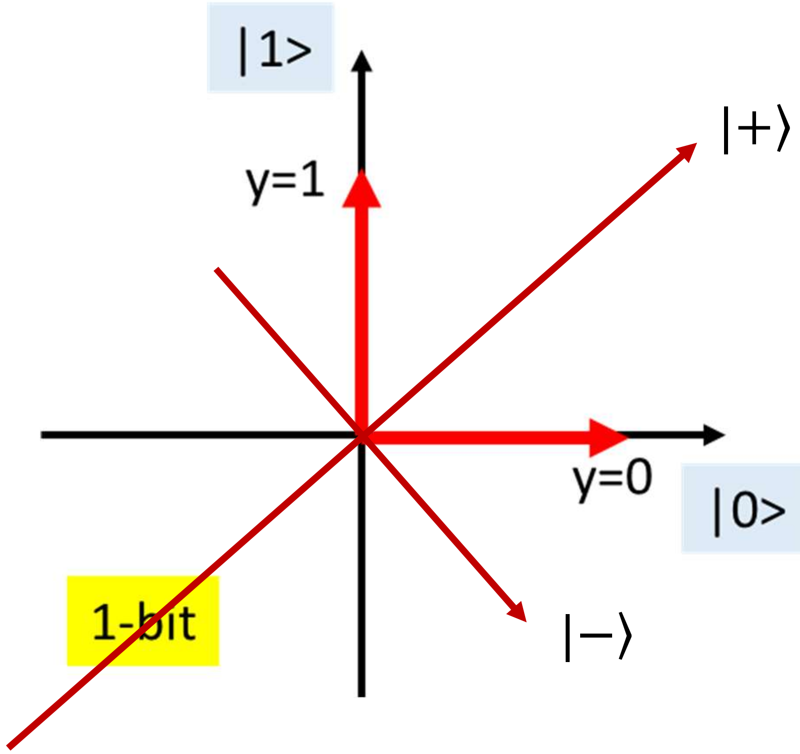


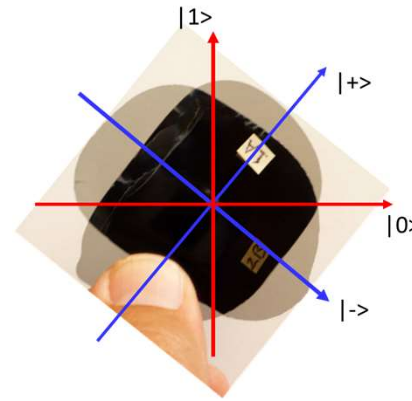
- We *always* specify some (possibly arbitrarily chosen) directions as our ``canonical'' bases
  - These are typically designated as the ``bit'' bases, representing the bit values |0> and |1>

- But we can *also* have *other* bases
  - Which can be defined in terms of our bit bases (or, alternately, our bit bases can be defined in terms of these other bases)

Alternate bases: The "sign" bases

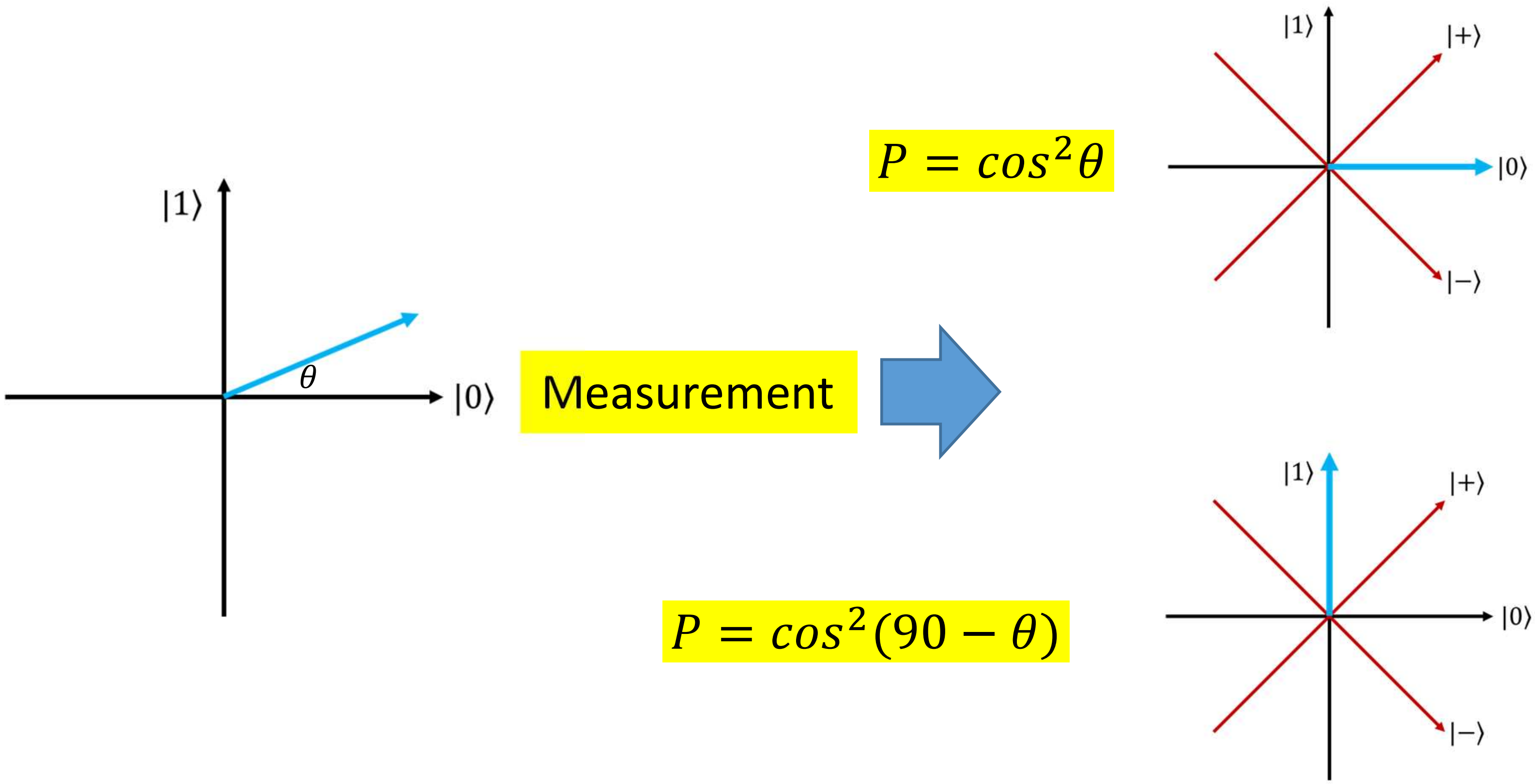






- A very popular set of alternate bases are the "sign" bases
  - Designated as $|+\rangle$ and $|-\rangle$ respectively
  - These are at +45 and -45 degrees to the bit bases, respectively
- Flipping between bit-based representations and sign-based representations is an often-encountered operation

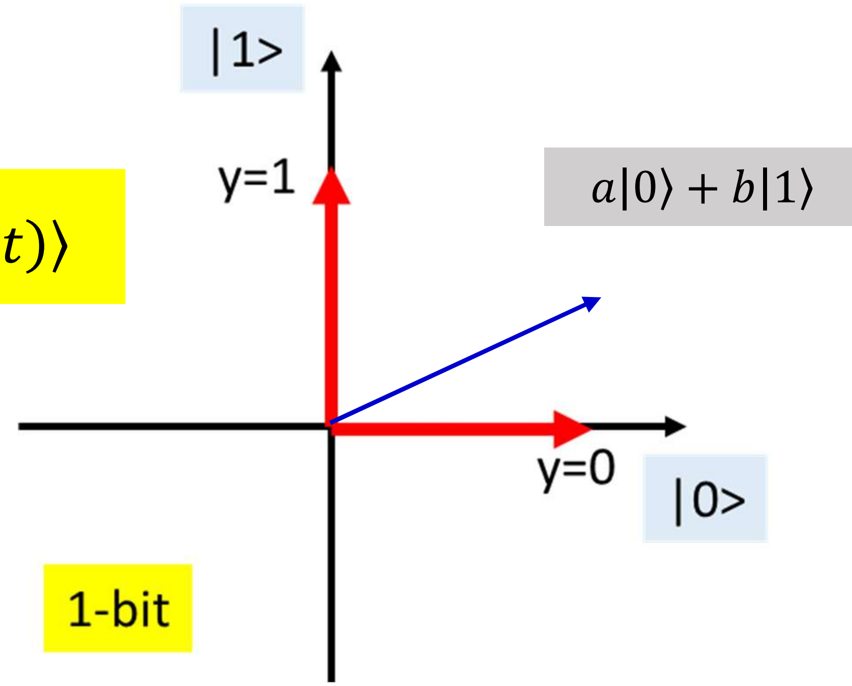
Measurement is not absolute



- Collapsing the vector according to one basis can still keep it indeterminate for other bases!
  - We will use this feature

Its all complex, but not at all complicated

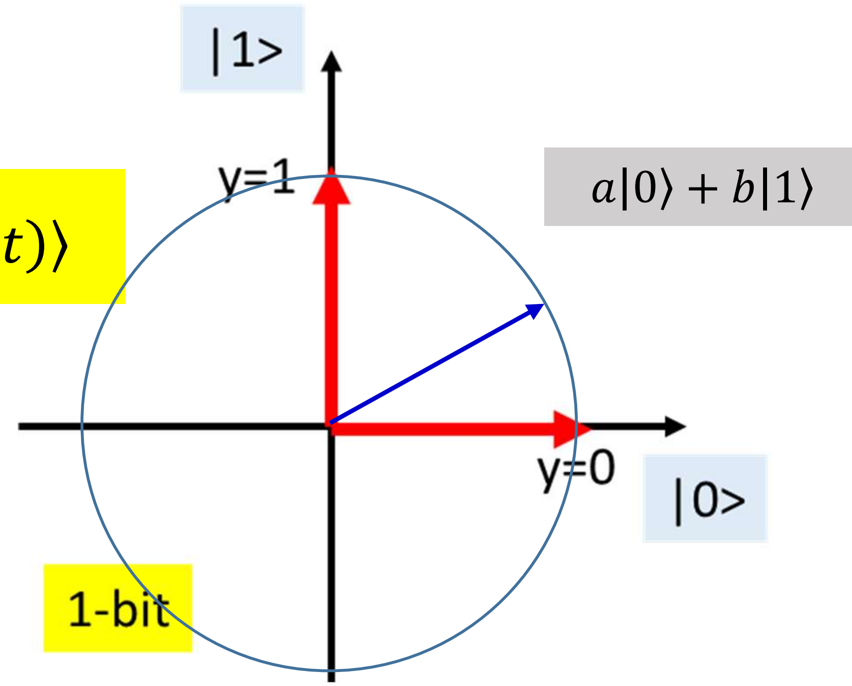
$$i h \frac{d}{dt}|\psi(t)\rangle = 2\pi H|\psi(t)\rangle$$



- The "weights" a and b are actually complex variables
  - Because Schroedinger's equation describes them as complex
- This simple visualization is *wrong*
  - Its missing two dimensions
    - The imaginary components of a and b

Restrictions on the weights

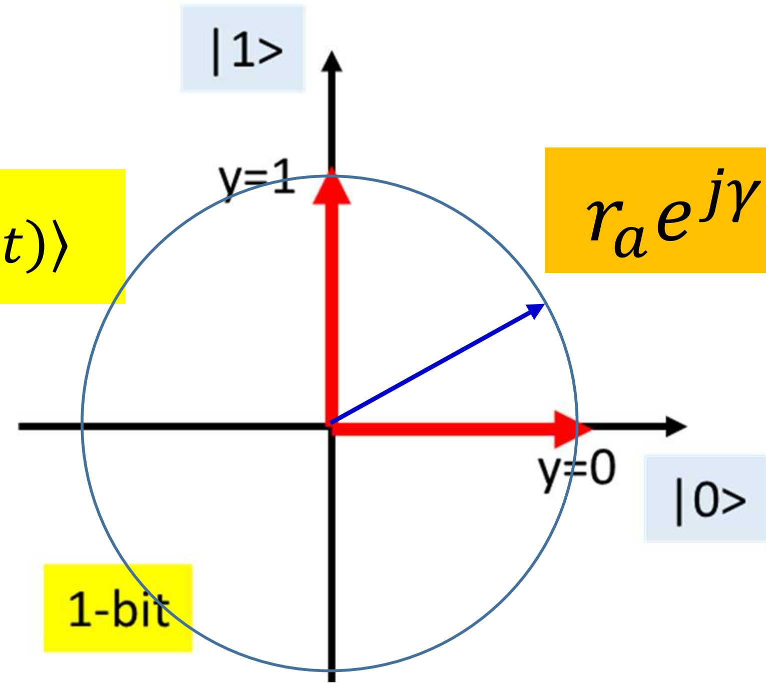
$$ih\frac{d}{dt}|\psi(t)\rangle = 2\pi H|\psi(t)\rangle$$



- $|a|^2 + |b|^2 = 1$
  - The qubits live on the surface of a hypersphere
- $P(|0\rangle) = |a|^2, \;\; P(|1\rangle) = |b|^2$
- $a = r_a e^{j\gamma}, b = r_b e^{j\delta}$
  - What is the relation between r_a and r_b

Restrictions on the weights

$$ih\frac{d}{dt}|\psi(t)\rangle = 2\pi H|\psi(t)\rangle$$

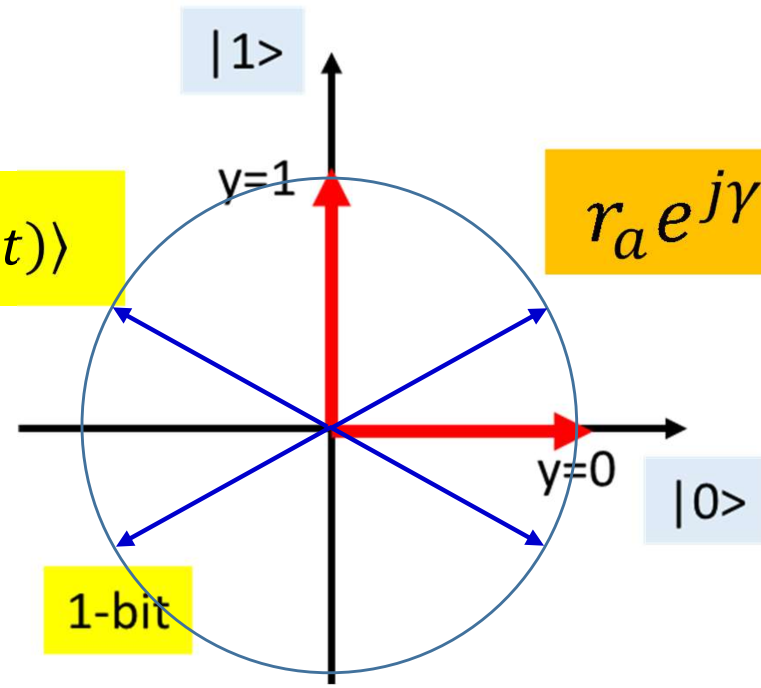


$$r_a e^{j\gamma}|0\rangle + r_b e^{j\delta}|1\rangle$$

- $|a|^2 + |b|^2 = 1$
  - The qubits live on the surface of a hypersphere
- $P(|0\rangle) = |a|^2, \;\; P(|1\rangle) = |b|^2$
- $a = r_a e^{j\gamma}, b = r_b e^{j\delta}$
  - What is the relation between r_a and r_b

Something odd

$$i h \frac{d}{dt} |\psi(t)\rangle = 2\pi H |\psi(t)\rangle$$



$$r_a e^{j\gamma} |0\rangle + r_b e^{j\delta} |1\rangle$$

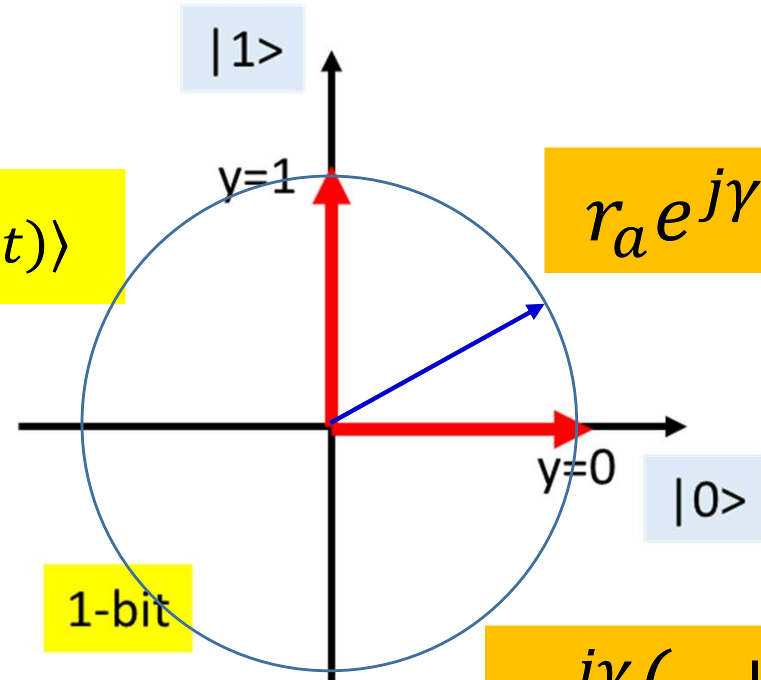
$$P(|0\rangle) = |r_a|^2$$

$$P(|1\rangle) = |r_b|^2$$

- All of these vectors represent the *same* $P(|0\rangle)$ and $P(|1\rangle)$
- But they're actually different "phasors"
    - Something we will use all the time

Restrictions on the weights

$$ih\frac{d}{dt}|\psi(t)\rangle = 2\pi H|\psi(t)\rangle$$

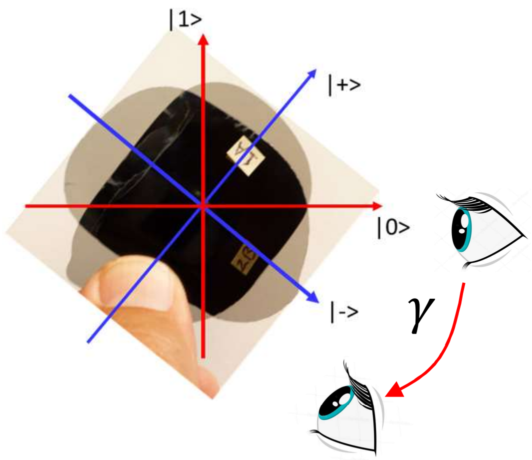


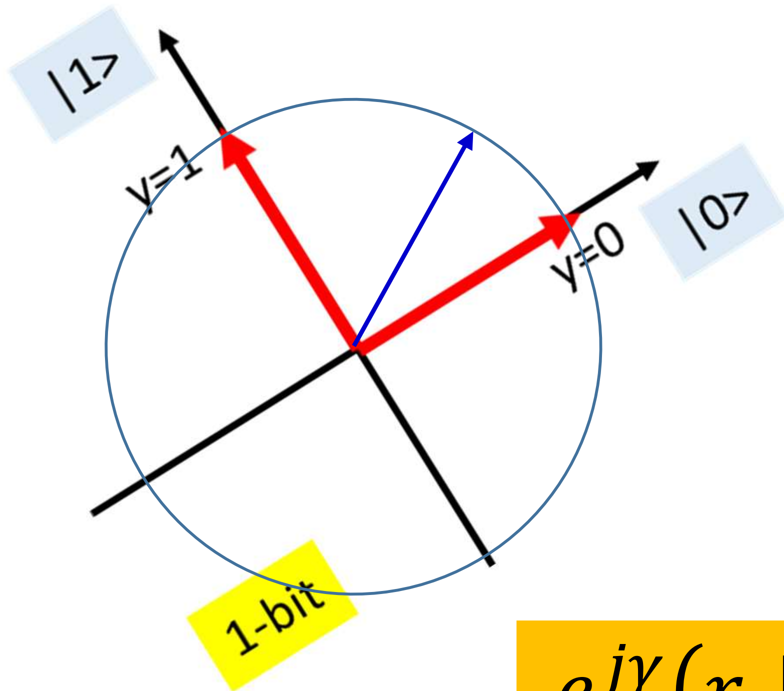
$$r_a e^{j\gamma}|0\rangle + r_b e^{j\delta}|1\rangle$$

$$e^{j\gamma}\left(r_a|0\rangle + r_b e^{j\delta-\gamma}|1\rangle\right)$$

- $|a|^2 + |b|^2 = 1$
  - The qubits live on the surface of a hypersphere
- $P(|0\rangle) = |a|^2, \;\; P(|1\rangle) = |b|^2$
- $a = r_a e^{j\gamma}, b = r_b e^{j\delta}$
  - What is the relation between r_a and r_b

The qubit is rotation invariant

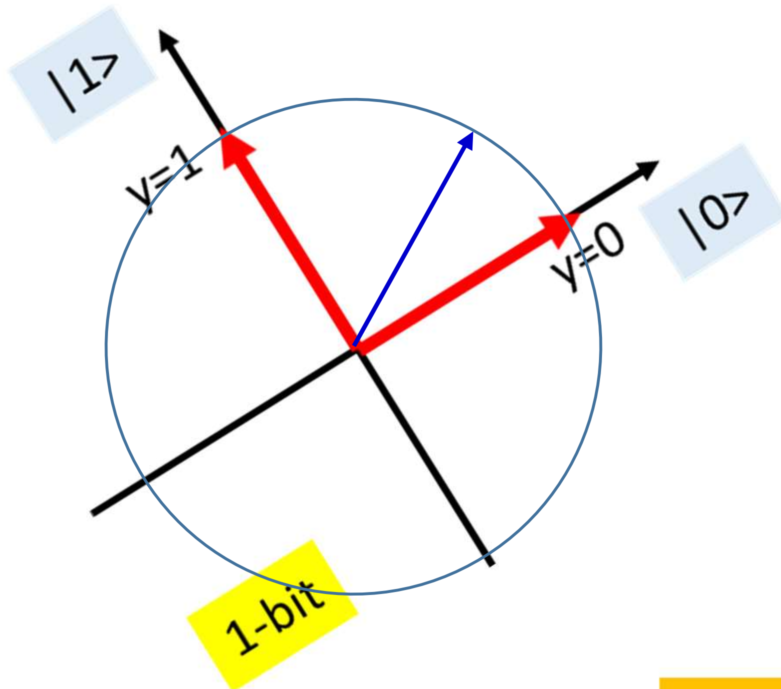




$$e^{j\gamma}\left(r_a|0\rangle + r_b e^{j(\delta-\gamma)}|1\rangle\right)$$

- Rotating your viewpoint doesn't matter
  - Alternately, rotating the entire space doesn't matter
  - Does not change its overall relative arrangement
- γ represents a global change of viewpoint and can be ignored

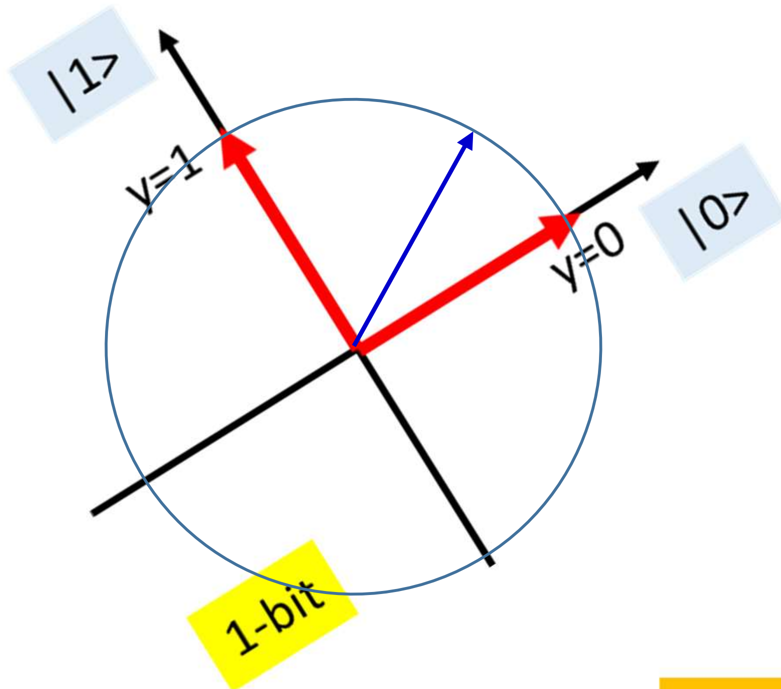
The qubit is rotation invariant



$$r_a|0\rangle + r_b e^{j(\delta-\gamma)}|1\rangle$$

- Rotating the space doesn't matter
  - What is the relation between r_a and r_b

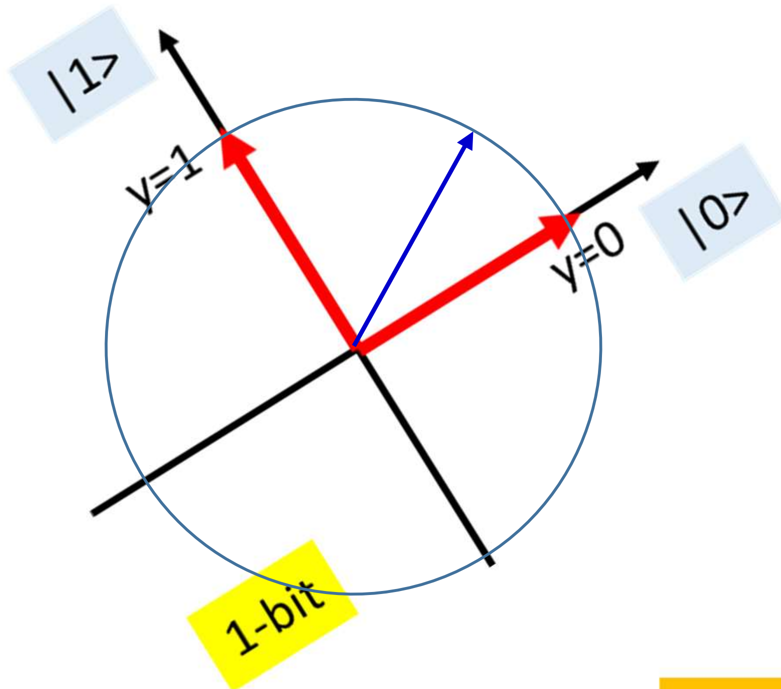
The qubit is rotation invariant



$$r_a|0\rangle + r_b e^{j\phi}|1\rangle$$

- Rotating the space doesn't matter
  - What is the relation between r_a and r_b

The qubit is rotation invariant

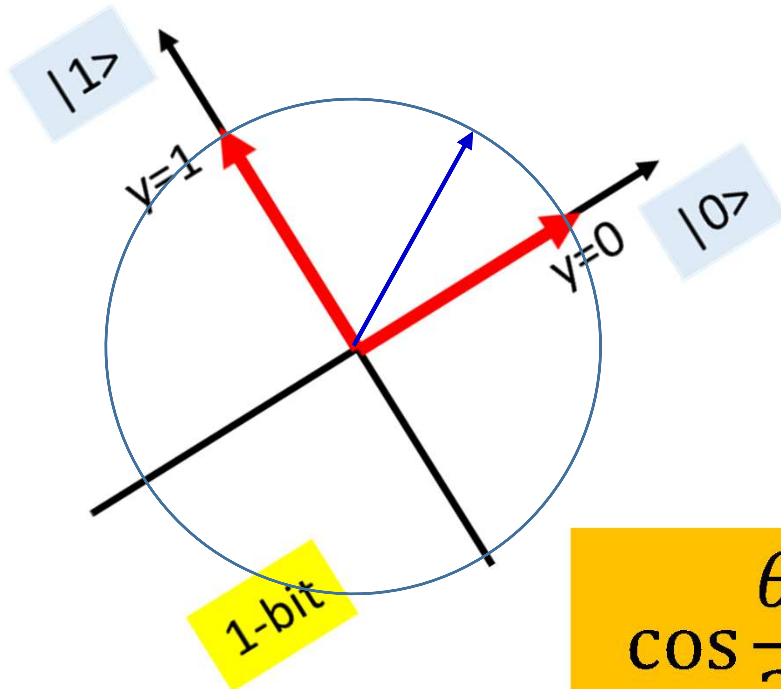


$$r_a|0\rangle + r_b e^{j\phi}|1\rangle$$

- Rotating the space doesn't matter
  - What is the relation between r_a and r_b

$r_a^2 + r_b^2 = 1 \implies r_a$ and r_b are analogous to the sine and cosine

The qubit is rotation invariant

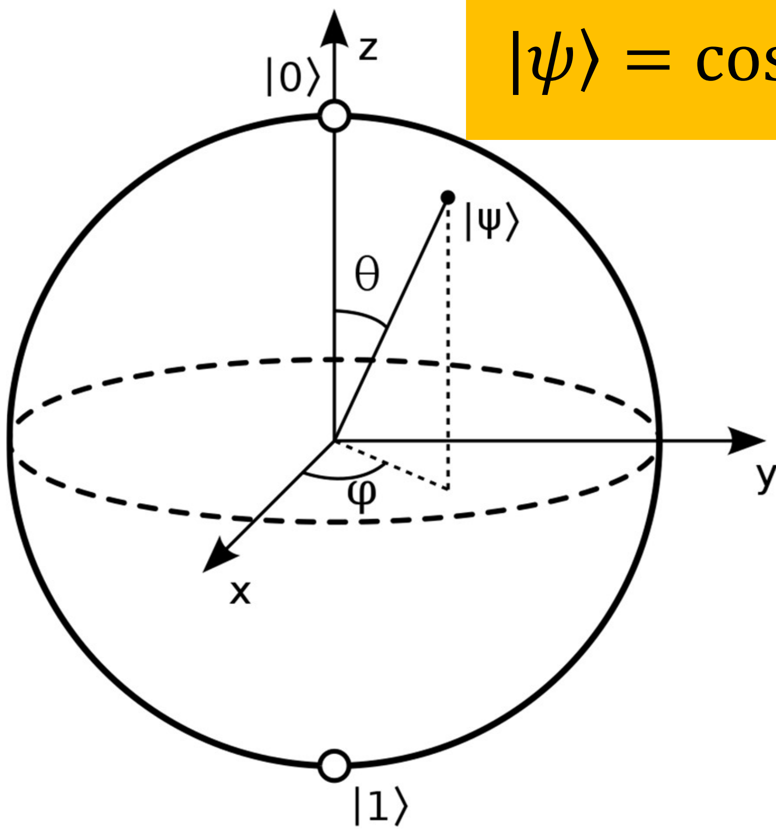


$$\cos\frac{\theta}{2}|0\rangle + \sin\frac{\theta}{2}e^{j\phi}|1\rangle$$

- Set $r_a = \cos\frac{\theta}{2}$ and $r_b = \sin\frac{\theta}{2}$
  - The ½ in the angle is for convenience of visualization…
- This is now a two-variable representation of two variables!
  - Can be visualized

The Bloch Sphere

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + \sin\frac{\theta}{2}e^{j\phi}|1\rangle$$



- Visualizing the qubit
  - 2 variable visualization in a 3D space
  - More on this later…