# Lecture 2 - Rotate, Compute, Rotate

(Goal: Explain why Q.C. not too complicated, like classical computing with 1 extra power. Next lecture we'll get to brass tacks, start from the beginning.)

**Quantum Mechanics = probability with minus signs**

(Indeed, if you had to invent the most conservative extension of probability theory in which there were somehow "negative probabilities", you'd invent Q.M.)

(Q.C. is not much more of a riff on classical computing than probabilistic computing is. So let's review that for a while.)

# Probabilistic Computing

(In '70s, some C.S.ists — Gill, J. Simon, ... — had a fantastic crazy idea: what if classical computing was augmented with... randomness!)

Deterministic code/circuits + Coin flip instruction → { 0 w.prob. 1/2 ; 1 w.prob. 1/2

(Now what do you get? BTW, this is exact parallel to Q.C....)

↳ (Philosophical interlude: is this physically possible?
Can we get "true random bits" in Nature?
Ironically: yes, by the prob. nature of Quantum Mech!
Practically: pseudorandomness seems fine, in theory & in practice.)

Q: Is prob. computing ≫ "classical" deterministic comp.?

A1: Yes, by definition: (more powerful) → can simulate a coin flip; ← cannot

(If the task is literally "do something random", e.g.: "Monte Carlo - simulate a physical process" or "pick a random 1024-bit prime number.")
↑ for a secret key

A2: Maybe not, when just computing functions.

(For this, det. and prob. computing on potentially equal footing. E.g., tasks like...)

- Multiply 2 numbers
- Test if # is prime
- Min. Spanning Tree

Q: Why would you want prob. computing for such tasks?

(Nature of prob. computing is chance of failure. If you insist circuit computes fcn w/ 100% accuracy, may as well be determ'ic. On HW 1 you'll explore answer...) A: trading error prob. for efficiency

(There are some tasks that we can solve noticeably more efficiently if we allow $2^{-500}$ prob. of error.) ← unphysically small, so don't worry.

e.g.: **Primality Testing** – ~~1024~~ n-bit integers

(a super-important task in crypto: testing if a given 1024-bit integer is prime.)

(Far from obvious it's solvable in "P".
Naive grade school alg. takes $\approx \sqrt{2^{1024}}$ steps (unphysical!)

G. Miller '76: Assuming ERH (← ext. Riemann Hypo., well-believed # theory conjecture)
(at CMU) an alg. using $\approx n^4$ steps

(e.g. $n = 1000 \Rightarrow 10^{12} = 1$ tril steps
(1 sec. on a PlayStation)

Solovay-Strassen '77:
Probabilistic alg. using $\approx n^3$ steps. (Now 1 millisecond!)
(WOW!!! Like the "Shor's Alg." of probabilistic computing!)
(Although... "only" replacing one "P" alg. with another "P" alg.)

Rabin '80: probabilistic riff on Miller: $\approx n^2$ steps
("Miller-Rabin alg." – used billions of times per day? (https) (Now a microsec.)

AKS '02: Determic alg, <u>provably</u> $\approx n^{12}$ steps.
(Lenstra-Pomerance) $\approx n^6$
(1 week on a PS4 to test n-bit #. still, in "P".)

g a a
r y x
a p
w a n
a l a
l — undergrads

(There are many examples of important tasks that we can do in "P" deterministically, but more efficiently (seemingly) in "P" probabilistically.)

(There are, like, 1 or 2 problems we can do in "P" probabilistically, but we don't know how to provably (but we have unproven det. algs) do in "P" det'istically.)

(However, for ALL "compute-a-function" tasks doable in "P" probabilistically, we know a deterministic "P" alg. (albeit, slower) that we STRONGLY BELIEVE works. Basically, "use a good pseudorandom # generator.")

STRONGLY BELIEVED CONJ: (related to P vs NP)

Every (compute-a-function) problem in "P" probabilistically is also in "P" deterministically.

# Probabilistic Computing Summary

- It's cool!
- Classical computing + one simple power
- Analyzing it requires some new math (probability)
- Quintessential use: simulate something random
- (seems to) give speedups over det. comp. from one level of P efficiency to another for many problems
- (strongly believed:) doesn't give speedups from "unphysical" (exponential time) to "physical" (polynomial time) for any problem (of the compute-a-function type)
- "Initialize array A[] of length 1000
  - for $0 \le i < 1000$
    - $A[i] := \text{CoinFlip}(0/1)$"
  - // classical determ. comp.

Describing A[]'s state now reqs. $2^{1000}$ numbers ($\Pr[A = x]$ $\forall x \in \{0,1\}^{1000}$)

# Quantum Computing Summary

(tell Feynman story)

- It's cool!
- Classical computing + one simple power "rotate"
- Analyzing it requires some new math (linear algebra)
- Quintessential use: simulate something quantum
- (seems to) give speedups over prob. comp. from one level of P efficiency to another for many problems ✓ (e.g. "Grover's Alg.")
- (seems to) give EXPONENTIAL speedup for one famous problem (Factoring) (and a couple more non-famous problems)
- (strongly believed) NOT to give speedups from unphysical (expon.) to physical (poly.) for "most" probs. (e.g. NP-complete probs.)
- Quantum computer code:
  - Initialize 1000 photons ("qubits")
  - Run them thru an obstacle course of mirrors/prisms/etc.

// Now describing "state" of photons requires $2^{1000}$ possibly negative numbers ("amplitudes")

(Feynman story, ca. early '80s:

As a physicist, wanted to be able to use a computer to simulate/predict Q.M.'ical behavior of 1000 particles

seems to require storing $2^{1000}$ numbers

→ impossible!

Yet the particles themselves do it!

Why not regard the particles as being part of the computer (like they're simulating themselves)!

→ Quantum Compating!

Feynman went on to have good preliminary thoughts on the potential for Q.C.s to simulate classical computation)

(Re: those $2^{1000}$ "amplitudes".

Important contrast w/ probabilistic case, like if you decided to add flipping coins to your classical computer.

Cf. Hmwk prob. on simulating such a machine

vs.

subsequent problem on outputting such a machine's probabilities.

You <u>cannot</u> do the former for a quantum machine w/ photons. Unlike with coins, it's not that the photons are "secretly" in some state defined by 1000 bits, and the $2^{1000}$ #'s (probabilities) just reflect our mathematical analysis, or lack of knowledge about them. The $2^{1000}$ amplitudes the only true way to describe the photons' state.)

# The one extra power of quantum computers

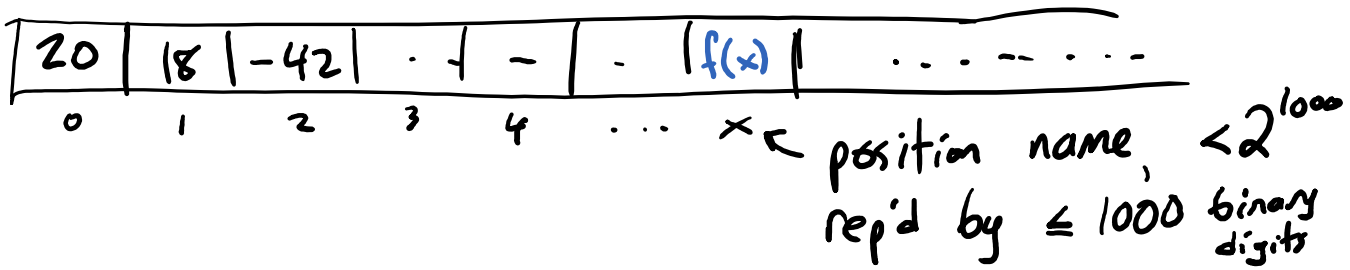
(I'll explain it in the style of those YouTube videos "one person explains concept at 5 different levels of difficulty"...)

4-year-old: Finding patterns in data lists of numbers

8-year-old: Getting clues about 10000....0ng (e.g., length $10^{500}$)

high school student: IMPLICITLY-represented lists

(You see, they have to be "implicitly represented" - it's physically impossible to explicitly rep. $10^{500}$ numbers.)




f is given by a python program, or a circuit, say

```
def f(x):
    // compute entry := x^th number in list
    return x
```

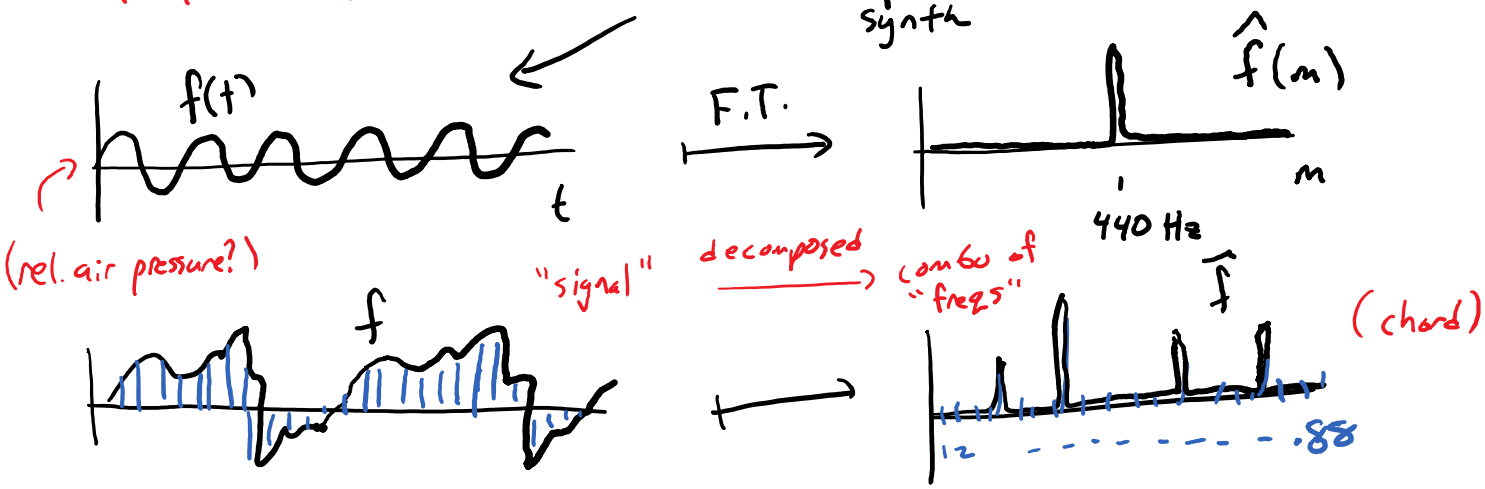
$f: \{0,1,2,\ldots, 2^{1000}-1\} \to \mathbb{Z} \to \mathbb{R} \to \mathbb{C}$

(↳ these $2^{1000}$ #'s encoded by the state of 1000 photons)

Undergrad: ~~Patterns~~ → Discrete Fourier Transform

(Probably you've heard of this at least, but maybe only vague notion of it. Don't worry, we'll go over it a lot later....)

Smash same ~~piano~~ synth notes




Discrete F.T. (Different (closely related) "Fourier Transforms" have different domains)

$f: \{domain\} \to \mathbb{R} \to \mathbb{C} \xleftrightarrow{FT} \hat{f}: \{domain\} \to \mathbb{C}$

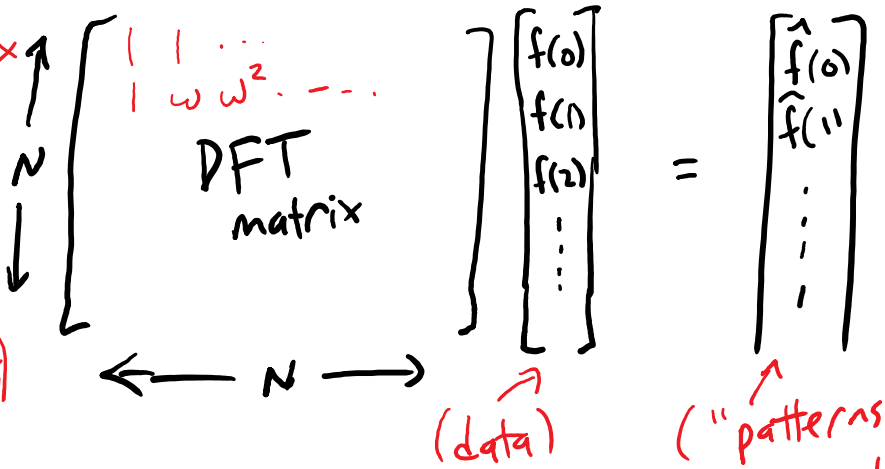
e.g.: • Domain = $\mathbb{R}$ or $[0,1]$ (Physicists love this one; in all intro QM texts; so hard because not discrete!)

• = $\{0,1,2,\ldots,2^n-1\}$
↑
ints mod $N=2^n$
⟼ freqs are discrete sines/cosines
(good for # theo & Shor factoring)

• = $\{0,1\}^n$
binary strings
$f: \{0,1\}^n \to \mathbb{C}$
boolean fcn.
⟼ freqs are XOR functions
(My favorite! I have a book... good for Simon's Alg. So trivial to implem. on a Q.C. ...)

(DFT is a matrix multip.)

(A ROTATION matrix; preserves lengths)




(data)

("patterns"/"freqs" in data)

(DFT used ALL THE TIME in real life computing/engineering.
BUT: "N" is always a "physical" #, like $2^{20}$.)

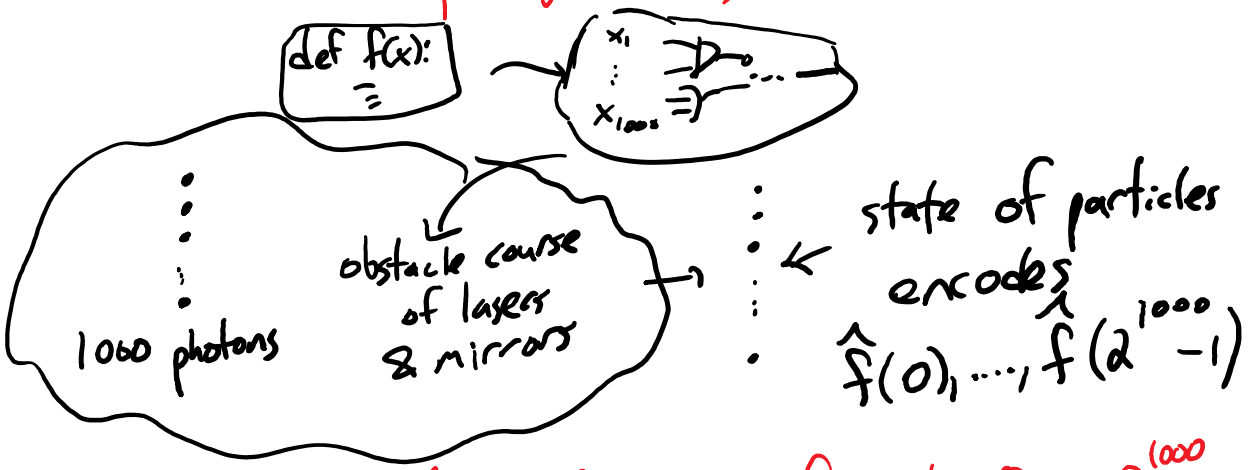
Naive DFT: $\approx N^2$ steps (matrix-vector mult.)

Fast: "FFT" $\approx N \log N$ steps (this is what makes it so gloriously awesome: "top 10 most important algs. ever")

# Quantum computer can implicitly do DFT

$N = 2^{1000}$, e.g. !! ($\approx \log N$ steps)

(How? And what does "implicitly" mean?) $\cdot \log\log N$




(Cannot now just ask for $\hat{f}(y)$ for your favorite $0 \le y < 2^{1000}$.
If you could, Q.C.'s could solve NP-complete probs & more!)

(So what can you do??)

"Measure" particles: (their polarization or spin or whatever)

Q.M. says: You "detect" $y^* \in \{0,1\}^{1000}$ with probability $|\hat{f}(y^*)|^2$.


(Turns out things always normalized so these add up to 1.)

. State then "snaps to" this $\hat{f}$:

$$\hat{f}(y) = \begin{cases} 1 & \text{if } y = y^* \\ 0 & \text{else} \end{cases}$$

(So you kind of get 1 sample of a freq./pattern, w/ probability related to how big/important it is for f.)

5th level, grad student: (Well, that's you. I'll give you the further explanation over the course of the next 3 months!)

(Next class: We'll start at very begining, basic rules of QM, "qubits", vectors, amplitudes, operators, etc.