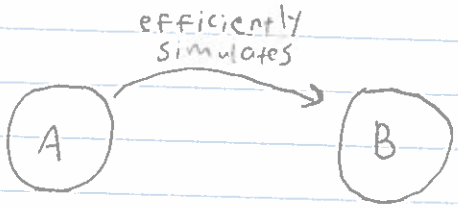
Lecture 1

Simulating Physics with Quantum Computers

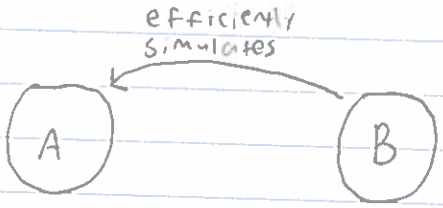
Why?

- because it will be useful (some day)
- because it allows us to compare computational power of different systems

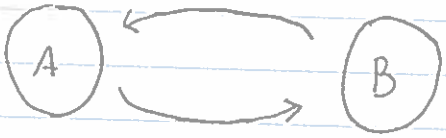


power:

$A \gtrsim B$



$B \gtrsim A$



$A \simeq B$

- what do we mean by "efficiently"?

CS: efficient = polynomial time, that is:
# of elementary logic operations scales as some polynomial in input size (# bits)

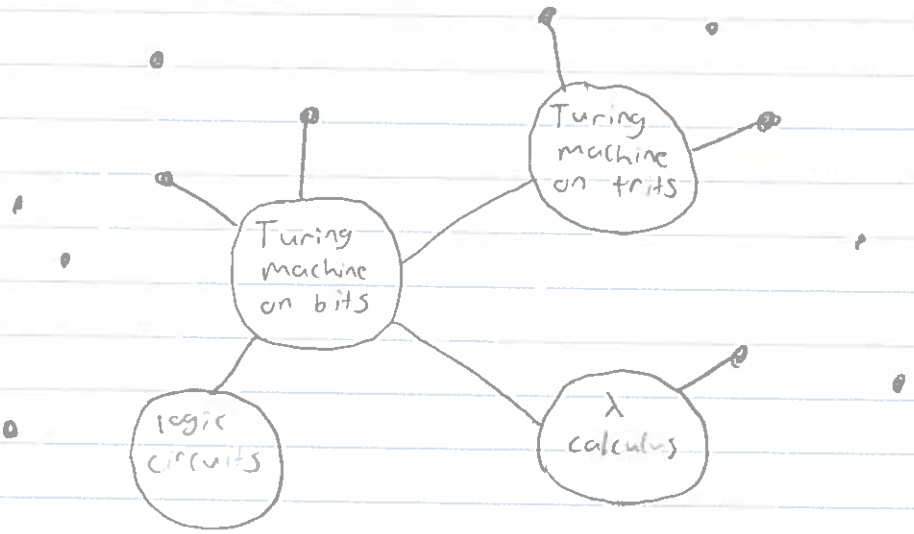
e.g. n^2 efficient
2^n not efficient

Q. Should we really call n^{65} efficient?

A1. That almost never happens in practice.

Hilroy

A2. efficient = polynomial time is a good convention because it has nice mathematical properties. In particular, it is transitive. This allows us to map out equivalence classes.



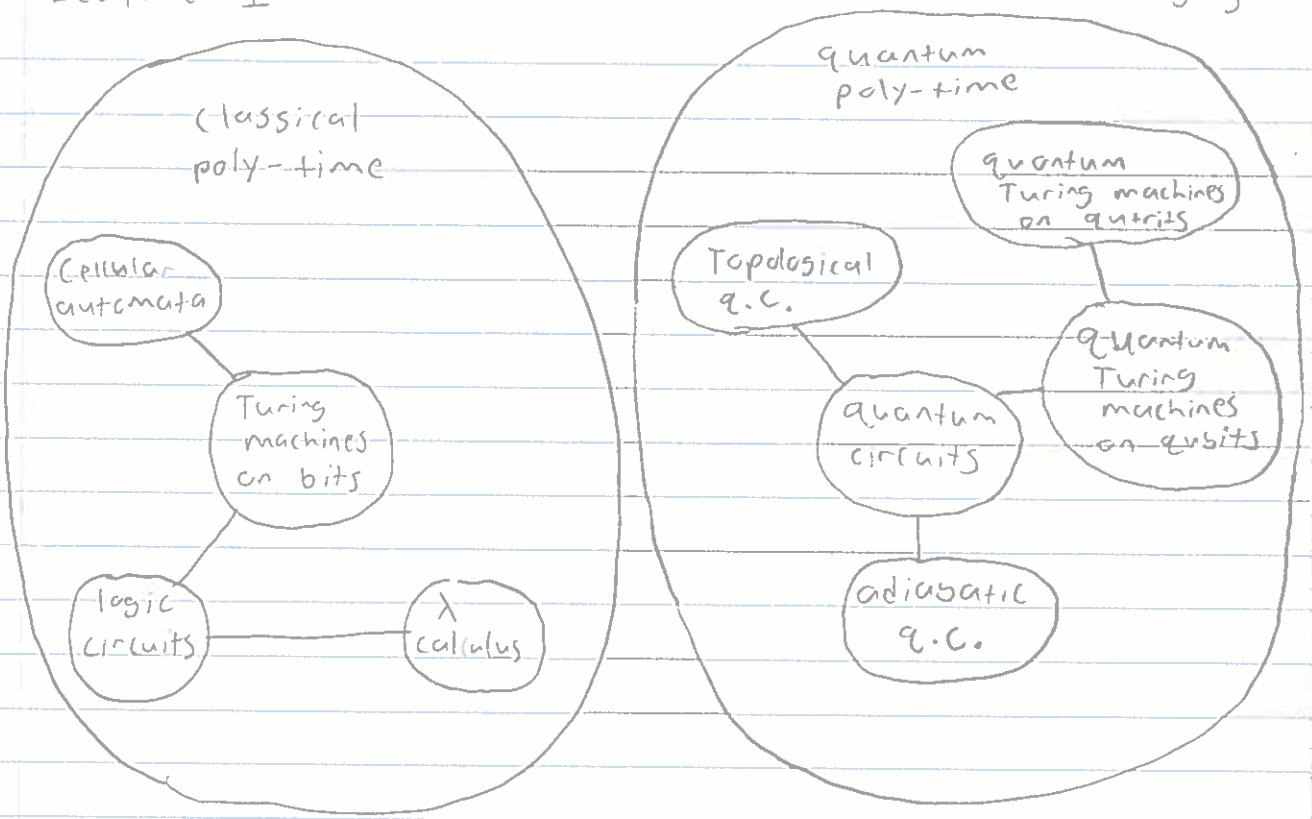
Q. Why not study finite instances? e.g. what can we compute with 1000 gates?

A1. Some people do.
A2. It's hard: very sensitive to details of model.
A3. Computers get faster all the time.

Q. How many equivalence classes (connected components) of universal computing are there?

A. Basically just two: classical and quantum. This is not a priori obvious and in fact is rather amazing (I think).

Hilroy



- Computer scientists often phrase things in terms of sets of problems rather than sets of models of computation.

P: set of problems solvable in polynomial time by classical computers

BQP: set of problems solvable in polynomial time by quantum computers

NP: set of problems whose solutions can be verified in polynomial time by classical computers.

etc. [see Scott Aaronson's complexity zoo]

Hilroy

- Some of these classes, e.g. NP are not thought to correspond to any realistic model of computation. That is not their purpose. Their purpose is to classify problems by their computational difficulty.

Q. What about DQC1, IQP, etc.?

A. Yes, there are some versions of restricted quantum computation that may be intermediate in power between classical and quantum computation. For the purpose of these lectures I'll classify these as "non-universal" and sweep them under the rug. However, they are interesting and I encourage you to read about them.

Some (cartoonish) history:

- As of 1980 or so, the program of proving everything polynomially equivalent to everything else had gotten almost boringly successful. Extrapolation:

> Extended Church-Turing thesis:
> Any reasonable model of universal computation is polynomially equivalent to a Turing machine.

With a grandiose name and a box around it this is begging to be challenged!

Hilroy

As I'm sure you have already heard, quantum computers pose a challenge to this. However, it is helpful to put this in the context of other challenges that have been or could be proposed.

Some Challenges:

1) Analog computing
   - arithmetic on $\mathbb{R}$, possibly realized by electrical or mechanical means, e.g.
   - GPAC (Shannon) $[+, \times, \int, 1,]$
   - BSS (Blum-Shub-Smale) [rational functions]
   - soap bubbles

   Many such models can solve NP-hard problems using polynomially many steps. However these speedups have invariably gone away once realistic noise and imprecision are taken into account.

2) Relativistic computing
   - just exploit time dilation in flat space
     → exponential speed-ups cost exponential energy (not realistic)
   - curved spacetime?
     → not clear
     → with closed timelike curves one might solve NP or even PSPACE [C.F. Watrous-Aaronson]
     → maybe we should consider more constrained, less pathological spacetimes
     → it has been proposed that infinite computations are accessible in AdS [Malament-Hogarth]. Probably energy cost & blueshift are prohibitive, but nobody seems to have calculated this (somebody should!)

3) Quantities from theoretical physics
- ground energies NP-hard (or worse: QMA-hard)
- partition functions (#P-hard)

→ you can't actually access these by setting up an experiment and measuring something.

4) Quantum computing
- Formulated by Feynman/Deutsch/Manin etc. in '80s
- Apparently can do some things in polynomial time that classical computers cannot, e.g. factor n-bit integers.
- initial skepticism: just like analog computing, the apparent exponential advantage will go away once realistic noise is taken into account.

> Threshold theorem: [Aharonov & Ben-Or] [others simultaneously]
> If error per operation is below some fixed threshold (say 0.01%), one can do arbitrarily long quantum computations reliably using error correction with only logarithmic overhead.

Most people, including me, feel this basically puts the issue to rest. Quantum computers can be built and scaled up, it's just (very) hard. There is current research on tolerating higher error rates with lower overhead and correcting more general noise models e.g. correlated, non-Markovian, etc.

Hilroy

So, we have:

> **Quantum Church-Turing Thesis:**
> Any computation physically realizable with polynomial resources (energy, time, etc.) can be simulated in polynomial time by a quantum circuit of polynomially-many gates and qubits.

The remainder of my lectures will be devoted to investigating this thesis. Can standard quantum computers really simulate <u>everything</u> efficiently?

- I'll describe quantum algorithms to simulate things.
  - other models of quantum computation
  - other physical systems:
    - non-relativistic QM, e.g. Chemistry
    - QFT
    - q. gravity is an open problem

- I'll consider quantum circuits to be the standard version of quantum computation.

  - qubits: $\sum_{x \in \{0,1\}^n} \psi(x) |x\rangle$

  - gates:

$$-\boxed{H}- = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad \text{CNOT} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

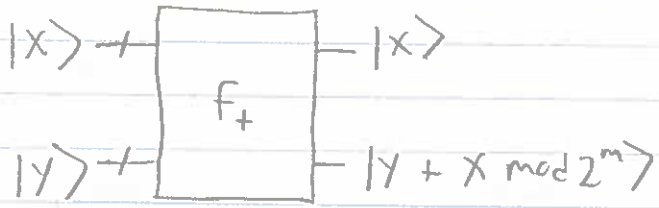
$$-\boxed{T}- = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$$

Hilroy

- quantum circuits are like assembly language
- to save time we need a higher level description like C or Python. I'll now present one (maybe more like pseudo-code). Each ingredient is in turn decomposable into polynomially-many elementary gates. For proofs see [Nielsen & Chuang Ch 4]. Most of these tools were built up in John Watrous's lectures, so this can be a recap.

Tool #1: Classical computing.

Let $f: \{0,1\}^n \to \{0,1\}^m$ be any function computable classically in polynomial time. Then we can make



Remarks:

- $|x\rangle$, $|y\rangle$ are called registers of qubits. We group qubits together (in our minds) to represent numbers in binary
- This is unitary, $|x\rangle \to |y\rangle$ is not, in general.
- If we set $Y=0$, we get X as output.

Hilroy

Tool #2: Phase kickback

Let $f: \{0,1\}^n \to \{0,1\}^m$ be efficiently computable

Then we can implement

$$U_f = \begin{bmatrix} e^{i2\pi f(0)/2^m} & & & \\ & e^{i2\pi f(1)/2^m} & & \\ & & \ddots & \\ & & & e^{i2\pi f(2^m-1)/2^m} \end{bmatrix}$$

i.e.

$$\sum_x \psi(x)|x\rangle \mapsto \sum_x e^{if(x)} \psi(x)|x\rangle.$$

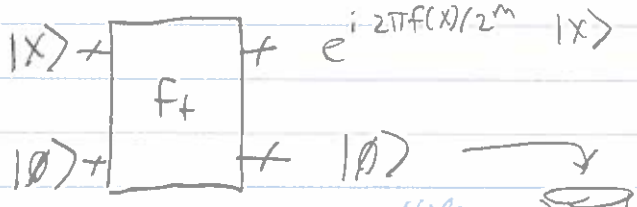
The proof is fun and quick, so I'll make an exception and show it. Let

$$|\phi\rangle = \frac{1}{\sqrt{M}} \sum_{y=0}^{M-1} e^{-i2\pi y/M} |y\rangle$$

This is an eigenstate of addition:

$$|\phi + k \bmod M\rangle = e^{i2\pi k/M} |\phi\rangle$$

So:



Hilroy

□

Tool #3 Quantum Fourier Transform

$$\sum_x \psi(x)|x\rangle \;\longrightarrow\; \boxed{\mathcal{F}} \;\longrightarrow\; \sum_p \tilde{\psi}(k)|k\rangle$$

where $\tilde{\psi}(p)$ is discrete Fourier transform of $\psi(x)$:

$$\tilde{\psi}(k) = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \psi(x)\, e^{2\pi i x k/N}$$

This uses $O(\log^2 N)$ gates. [Shor 94]

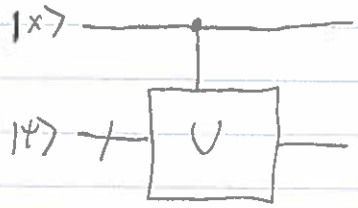
Remark: This is how we make $|\phi\rangle$

$$\mathcal{F}\underbrace{|N-1\rangle}_{\equiv -1 \bmod N} = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{-i2\pi k/N}\cdot|k\rangle$$

⚠ You must always be careful to only use states that can be efficiently made from standard reference states e.g. $|0\ldots0\rangle$. Otherwise you may fool yourself into thinking you've solved hard problems.

Tool #4: controlled unitaries

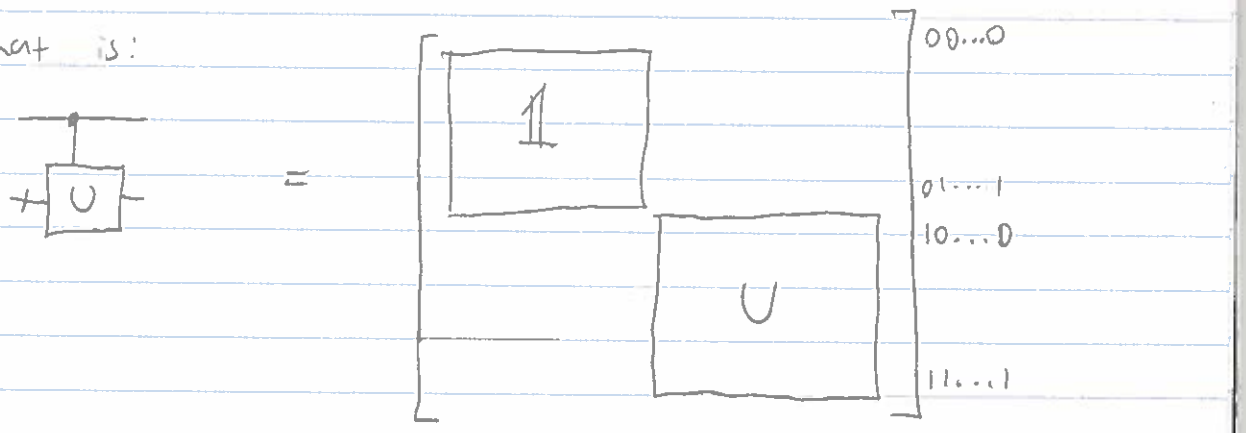
Let U be any efficiently-implementable unitary on n qubits. Then you can make



which applies U to $|\psi\rangle$ if $|x\rangle = |1\rangle$ and does nothing otherwise.

Hilroy

That is:



~ "quantum if statement"

Remark: This high-level language is complete: any quantum algorithm can be expressed. Actually, all we need is:

- classical computing
- Fourier transforms.

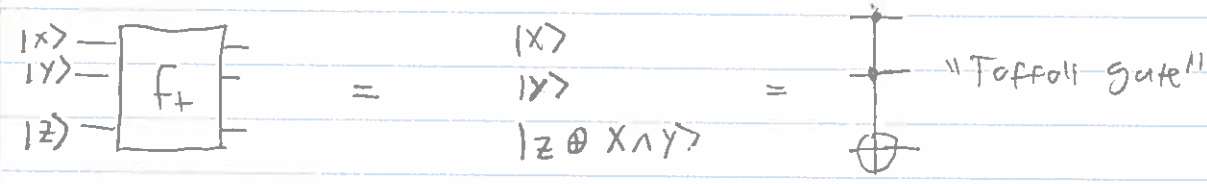
Proof: Consider the Fourier transform and set $N=2$, i.e. 1 qubit,

$$\tilde{\Psi}(k) = \frac{1}{\sqrt{2}} \sum_{x=0}^{1} \Psi(x) \underbrace{e^{2\pi i x k/2}}_{(-1)^{x}}$$

So:

$$-\boxed{\mathcal{F}_{N=2}}- = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$ "Hadamard gate"

Next consider $f: \{0,1\}^2 \to \{0,1\}$ given by the AND function. Then



Hilroy

Lastly, we invoke a big hammer:

{Hadamard, Toffoli} is a universal gate set for quantum computing.

See: [Aharonov: quant-ph/0301040]

Remark: Hadamard & Toffoli involve only real numbers. Complex amplitudes are not necessary for universal quantum computation. □

For more conventional gate sets, we have an additional tool which is not computationally necessary, but is a handy primitive for quantum algorithm design:

Tool #5: Arbitrary unitaries on log-many qubits

Let U be any $N \times N$ unitary. Then U can be implemented using $O(N^2 \log N)$ quantum gates. In other words we can make arbitrary unitaries on $\log(n)$ qubits in $\text{poly}(n)$ time.

See [Nielsen & Chuang §4.2]

Having spent a long time building motivation and tools, let's finally start simulating stuff!

Example 1:

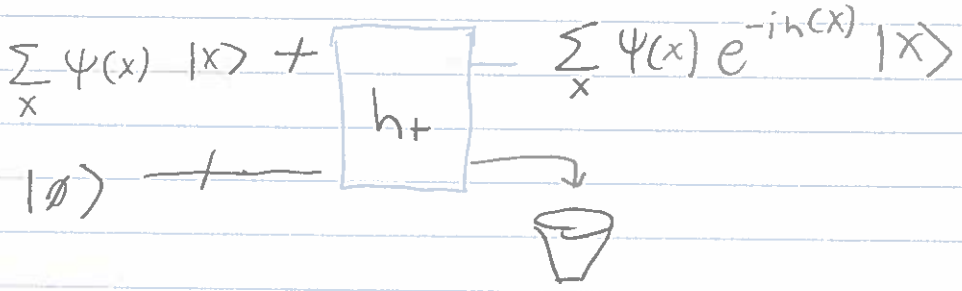
We'll start with something easy: a diagonal Hamiltonian on a 2^n-dimensional Hilbert space.

$$H = \begin{bmatrix} h(0) & & & \\ & h(1) & & \\ & & h(2) & \\ & & & \ddots \\ & & & & h(2^n-1) \end{bmatrix}$$

We want to build a quantum circuit implementing the corresponding unitary time-evolution operator.

$$e^{-iHt} = \begin{bmatrix} e^{-iH(0)t} & & & \\ & e^{-iH(1)t} & & \\ & & \ddots & \\ & & & e^{-iH(2^n-1)t} \end{bmatrix}$$

We can do this as long as $h(x)$ is efficiently computable $\forall x \in \{0, 2^n-1\}$. Just use phase kickback!



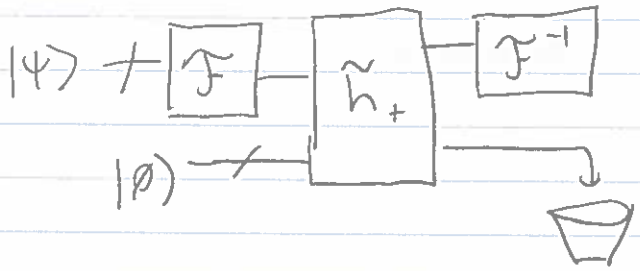
Example 2 Let H_k be a Hamiltonian that is diagonal in the Fourier basis:

$$H_k = \mathcal{F}^{-1} \underbrace{\begin{bmatrix} \tilde{h}(0) & & & \\ & \tilde{h}(1) & & \\ & & \ddots & \\ & & & \tilde{h}(2^n-1) \end{bmatrix}}_{\text{call this matrix } \tilde{H}} \mathcal{F}$$

Then:

$$e^{-iH_k t} = e^{-i\mathcal{F}^{-1}\tilde{H}\mathcal{F}t} = \mathcal{F}^{-1} e^{-i\tilde{H}t} \mathcal{F}$$

which gives us a quantum circuit!



So, as long as $\tilde{h}$ is efficiently computable we can simulate this too.

Example 3: Consider non-relativistic quantum mechanics in 1d.

$$\frac{d\psi}{dt} = -i\hbar \underbrace{\left[-\frac{1}{2m} \frac{d^2}{dx^2} + V(x) \right]}_{H_{NR}} \psi$$

Hilroy

Notice:

$$H_{NR} = \frac{-1}{2m}\frac{d^2}{dx^2} + V(x)$$

$\frac{-1}{2m}\frac{d^2}{dx^2}$: diagonal in Fourier basis

$V(x)$: diagonal in x-basis

Q. We know how to simulate each term, but how can we simulate their sum?

A. Trotter-Suzuki formulae:

(1st-order) $e^{i(A+B)\delta t} = e^{iA\delta t}\, e^{iB\delta t} + \mathcal{O}(\delta t^2)$

(2nd order) $e^{i(A+B)\delta t} = e^{iA\frac{\delta t}{2}}\, e^{iB\delta t}\, e^{iA\frac{\delta t}{2}} + \mathcal{O}(\delta t^3)$

$\vdots$

Kth order

So:

$$\left(e^{iAt/n}\, e^{iBt/n}\right)^n = e^{i(A+B)t} + \mathcal{O}\left(\frac{1}{n}\right)$$

From this formula we can read off a corresponding simulation circuit for

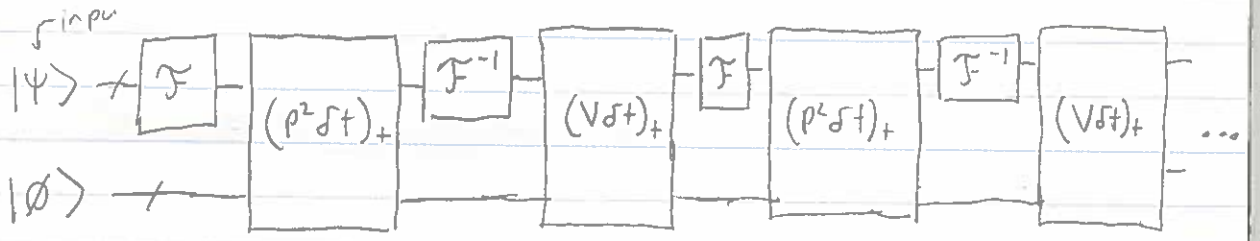
$$e^{-iH_{NR}t}$$

Hilroy

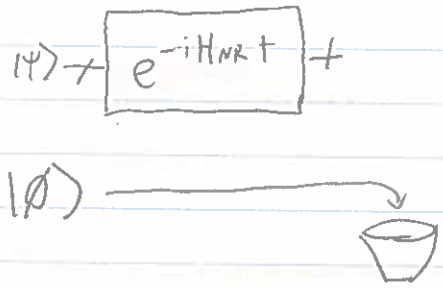
Set $\hbar = \frac{1}{2m} = 1$. Then

$$H_{NR} = p^2 + V(x)$$

and time evolution is $e^{-iH_{NR}t}$ given by:



$\simeq$



Next time:

- State preparation
- measurement
- simulating QFT
- open problems

Hilroy