Quantum teleportation, diagrams, and the one-time pad

A. Kissinger

Digital Security Group
Institute for Computing and Information Sciences
Radboud University Nijmegen

23rd November 2016

Outline

Outline

Processes

- A process is anything with zero or more *inputs* and zero or more *outputs*

Processes

- A process is anything with zero or more *inputs* and zero or more *outputs*
- For example, this function:

$$f(x, y) = x^2 + y$$

Processes

- A process is anything with zero or more *inputs* and zero or more *outputs*
- For example, this function:

$$f(x, y) = x^2 + y$$

...is a process when takes two real numbers as input, and produces a real number as output.

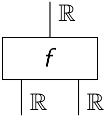
Processes

- A process is anything with zero or more *inputs* and zero or more *outputs*
- For example, this function:

$$f(x, y) = x^2 + y$$

...is a process when takes two real numbers as input, and produces a real number as output.

- We could also write it like this:



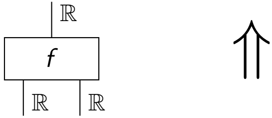
Processes

- A process is anything with zero or more *inputs* and zero or more *outputs*
- For example, this function:

$$f(x, y) = x^2 + y$$

...is a process when takes two real numbers as input, and produces a real number as output.

- We could also write it like this:



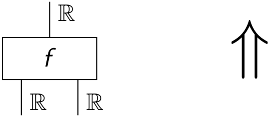
Processes

- A process is anything with zero or more *inputs* and zero or more *outputs*
- For example, this function:

$$f(x, y) = x^2 + y$$

...is a process when takes two real numbers as input, and produces a real number as output.

- We could also write it like this:



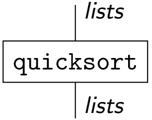
- The labels on wires are called system-types or just types

More processes

- Similarly, a computer programs are processes

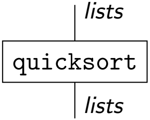
More processes

- Similarly, a computer programs are processes
- For example, a program that sorts lists might look like this:

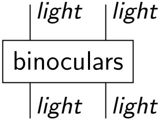


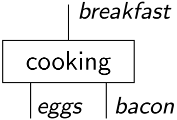
More processes

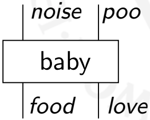
- Similarly, a computer programs are processes
- For example, a program that sorts lists might look like this:



- These are also perfectly good processes:

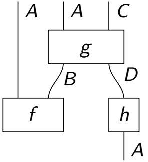






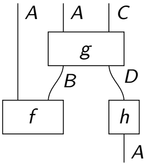
Diagrams

- We can combine simple processes to make more complicted ones, described by diagrams:

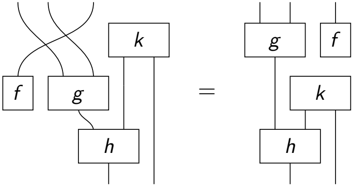


Diagrams

- We can combine simple processes to make more complicted ones, described by diagrams:

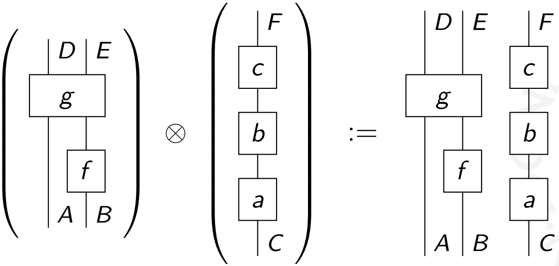


- The golden rule: only connectivity matters!



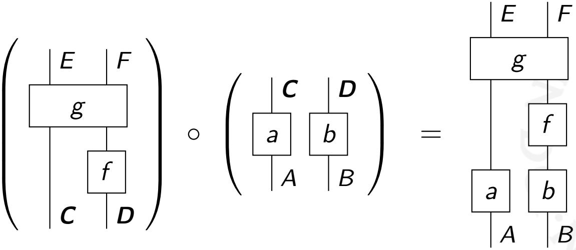
Diagrams

- Special cases are parallel composition:



Diagrams

- ...and sequential composition:

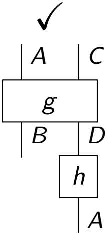


Types

- Connections are only allowed where the types match

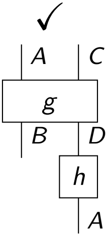
Types

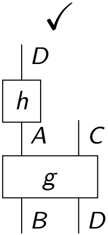
- Connections are only allowed where the types match, e.g.:



Types

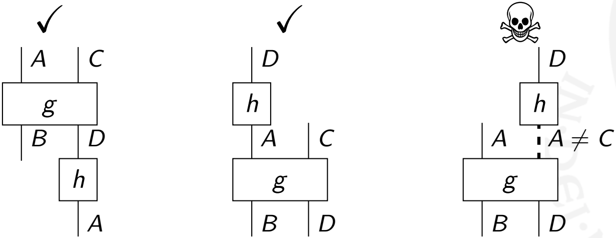
- Connections are only allowed where the types match, e.g.:





Types

- Connections are only allowed where the types match, e.g.:

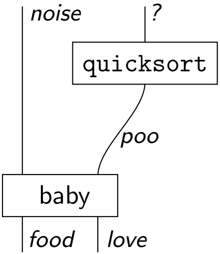


Types and Process Theories

- Types tell us when it makes sense to plug processes together

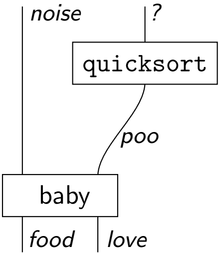
Types and Process Theories

- Types tell us when it makes sense to plug processes together
- Ill-typed diagrams are undefined:



Types and Process Theories

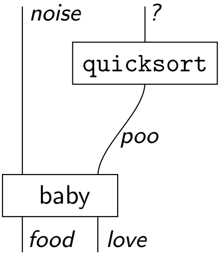
- Types tell us when it makes sense to plug processes together
- Ill-typed diagrams are undefined:



- In fact, these processes don't ever sense to plug together

Types and Process Theories

- Types tell us when it makes sense to plug processes together
- Ill-typed diagrams are undefined:



- In fact, these processes don't ever sense to plug together
- A family of processes which *do* make sense together is called a process theory

Example: **relations**

In the process theory of **relations**:

Example: **relations**

In the process theory of **relations**:

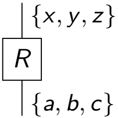
- system-types are sets

Radboud University Nijmegen

Example: **relations**

In the process theory of **relations**:

- system-types are sets
- processes are relations

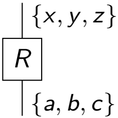


$$= \begin{cases} a \mapsto x \\ a \mapsto y \\ b \mapsto z \end{cases}$$

Example: **relations**

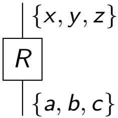
In the process theory of **relations**:

- system-types are sets
- processes are relations



$$= \begin{cases} a \mapsto x \\ a \mapsto y \\ b \mapsto z \end{cases}$$

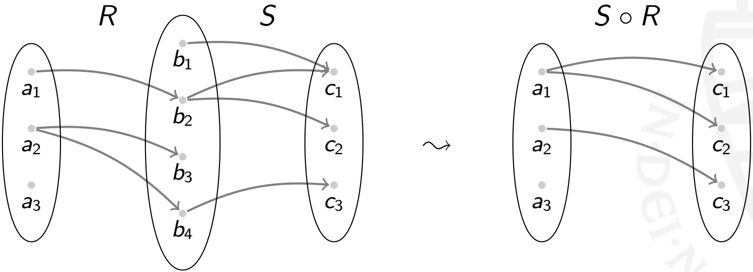
- ...which we can think of as non-deterministic computations:



$$= \begin{cases} a \mapsto \{x, y\} \\ b \mapsto z \\ c \mapsto \emptyset \end{cases}$$

Example: **relations**

Relations compose in sequentially just like you learned in school:



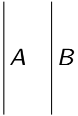
Example: **relations**

...and they compose in parallel via the cartesian product.

Example: **relations**

...and they compose in parallel via the cartesian product.

- that is, systems compose like this:



Example: **relations**

...and they compose in parallel via the cartesian product.

- that is, systems compose like this:

$$|A \; |B \quad := \quad \{(a, b) \mid a \in A, b \in B\}$$

Example: **relations**

...and they compose in parallel via the cartesian product.

- that is, systems compose like this:

$$\Big| A \;\; \Big| B \quad := \quad \{(a,b) \mid a \in A, b \in B\}$$

- so relations compose like this:

$$\boxed{R}\;\boxed{S} \;::\; (a,b) \mapsto (c,d) \iff \left(\boxed{R} \;::\; a \mapsto c \text{ and } \boxed{S} \;::\; b \mapsto d \right)$$

Some processes in **relations**

- ‘no wire’ is a one-element set:

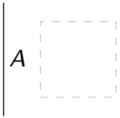
 $:= \{\bullet\}$

Some processes in **relations**

- ‘no wire’ is a one-element set:

$$:= \{\bullet\}$$

- ...because:



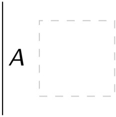
$$= \{(a, \bullet) \mid a \in A\} \cong A =$$

Some processes in **relations**

- ‘no wire’ is a one-element set:

$$:= \{\bullet\}$$

- ...because:



$$= \{(a, \bullet) \mid a \in A\} \cong A =$$

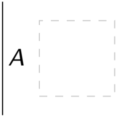
- processes from ‘no wire’ represent (non-deterministic) states

Some processes in **relations**

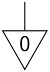
- ‘no wire’ is a one-element set:

 $:= \{\bullet\}$

- ...because:

 $= \{(a, \bullet) \mid a \in A\} \cong A =$ 

- processes from ‘no wire’ represent (non-deterministic) states, e.g. for a bit:



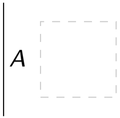
$= \Big\{ \bullet \mapsto 0$

Some processes in **relations**

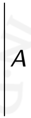
- ‘no wire’ is a one-element set:

$$:= \{\bullet\}$$

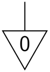
- ...because:



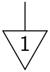
$$= \{(a, \bullet) \mid a \in A\} \cong A =$$



- processes from ‘no wire’ represent (non-deterministic) states, e.g. for a bit:



$$= \{\bullet \mapsto 0$$



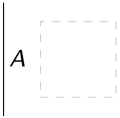
$$= \{\bullet \mapsto 1$$

Some processes in **relations**

- 'no wire' is a one-element set:

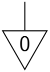
$:= \{\bullet\}$

- ...because:

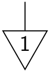


$= \{(a, \bullet) \mid a \in A\} \cong A =$

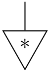
- processes from 'no wire' represent (non-deterministic) states, e.g. for a bit:



$= \{\bullet \mapsto 0$



$= \{\bullet \mapsto 1$



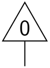
$= \{\bullet \mapsto \{0, 1\}$

Some processes in **relations**

- ...whereas processes to ‘no wire’ are called effects.

Some processes in **relations**

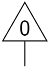
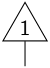
- ...whereas processes to ‘no wire’ are called effects. These test for the given state(s):



$$= \Big\{ 0 \mapsto \bullet$$

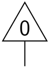
Some processes in **relations**

- ...whereas processes to 'no wire' are called effects. These test for the given state(s):

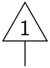
 $= \{0 \mapsto \bullet$  $= \{1 \mapsto \bullet$

Some processes in **relations**

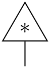
- ...whereas processes to ‘no wire’ are called effects. These test for the given state(s):



$= \{0 \mapsto \bullet$

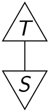


$= \{1 \mapsto \bullet$

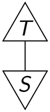


$= \{\{0,1\} \mapsto \bullet$

- when state meets effect, there are two possibilities:



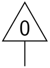
$= \{\bullet \mapsto \bullet$



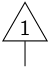
$= \emptyset$

Some processes in **relations**

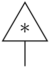
- ...whereas processes to 'no wire' are called effects. These test for the given state(s):



$= \{0 \mapsto \bullet$

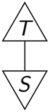


$= \{1 \mapsto \bullet$

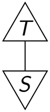


$= \{\{0, 1\} \mapsto \bullet$

- when state meets effect, there are two possibilities:



$= \{\bullet \mapsto \bullet$



$= \emptyset$

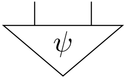
These stand for **true** and **false**.

States on two systems

- States on two systems are more interesting

States on two systems

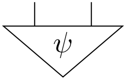
- States on two systems are more interesting, e.g.:



$$:= \left\{ * \mapsto \{(0,0),(1,1)\} \right.$$

States on two systems

- States on two systems are more interesting, e.g.:

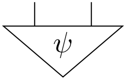


$$:= \left\{ * \mapsto \{(0,0),(1,1)\} \right.$$

Interpretation: "I don't know what bit I have, but I know its the same as yours"

States on two systems

- States on two systems are more interesting, e.g.:



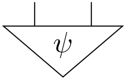
$$:= \left\{ * \mapsto \{(0,0),(1,1)\} \right.$$

Interpretation: "I don't know what bit I have, but I know its the same as yours"

- States of the two systems no longer have their own, separate identities

States on two systems

- States on two systems are more interesting, e.g.:



$$:= \left\{ * \mapsto \{(0,0),(1,1)\} \right.$$

Interpretation: "I don't know what bit I have, but I know its the same as yours"

- States of the two systems no longer have their own, separate identities
- Hence we get...

Outline

Process theories

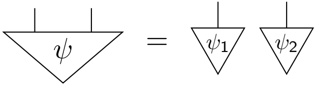
Non-separability

One-time pad

Quantum teleportation

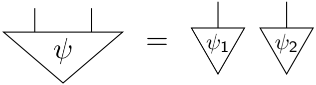
Separable states

- A state ψ on two systems is *separable* if there exist ψ_1, ψ_2 such that:



Separable states

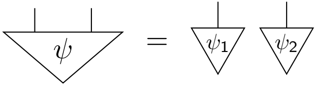
- A state ψ on two systems is *separable* if there exist ψ_1, ψ_2 such that:



- **Intuitively:** the properties of the system on the left are *independent* from those on the right

Separable states

- A state ψ on two systems is *separable* if there exist ψ_1, ψ_2 such that:



- **Intuitively:** the properties of the system on the left are *independent* from those on the right
- In the deterministic-land, *all states* to separate...

Characterising non-separability

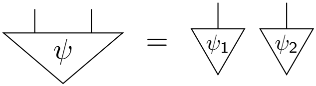
- ...which is why non-separable states are way more interesting!

Characterising non-separability

- ...which is why non-separable states are way more interesting!
- But, how do we know we've found one?

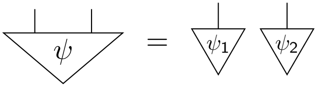
Characterising non-separability

- ...which is why non-separable states are way more interesting!
- But, how do we know we've found one?
- i.e. that there do not exist states ψ_1, ψ_2 such that:



Characterising non-separability

- ...which is why non-separable states are way more interesting!
- But, how do we know we've found one?
- i.e. that there do not exist states ψ_1, ψ_2 such that:



- **Problem:** Showing that something doesn't exist is hard.

Characterising non-separability

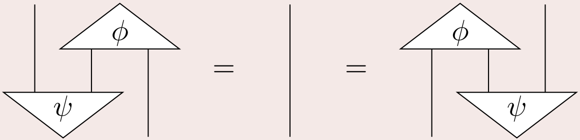
Solution: Replace a negative property with a postive one:

Characterising non-separability

Solution: Replace a negative property with a postive one:

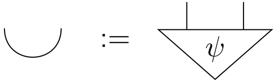
Definition

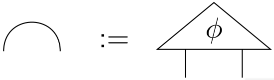
A state ψ is called *cup-state* if there exists an effect ϕ, called a *cap-effect*, such that:



Cup-states

- By introducing some clever notation:

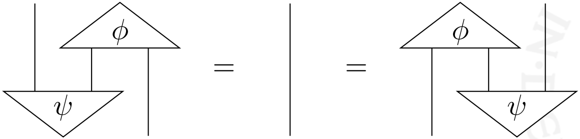




Cup-states

- By introducing some clever notation:

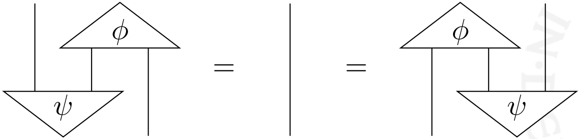
- Then these equations:



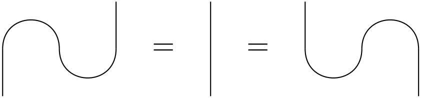
Cup-states

- By introducing some clever notation:

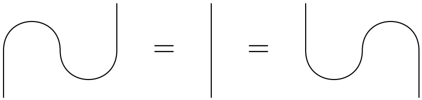
- Then these equations:



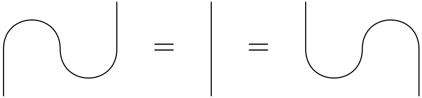
- ...look like this:



Yank the wire!



Yank the wire!



Example

- In **relations**, there is an obvious choice of cup-state:

$$:= \left\{ * \mapsto \{(0,0),(1,1)\} \right\}$$

Example

- In **relations**, there is an obvious choice of cup-state:

$$:= \left\{ * \mapsto \{(0,0),(1,1)\} \right.$$

- The associated cap-effect corresponds to "checking if two bits are the same":

$$:= \left\{ \{(0,0),(1,1)\} \mapsto * \right.$$

Example

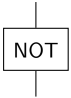
- In **relations**, there is an obvious choice of cup-state:

$$:= \left\{ * \mapsto \{(0,0),(1,1)\} \right.$$

- The associated cap-effect corresponds to "checking if two bits are the same":

$$:= \left\{ \{(0,0),(1,1)\} \mapsto * \right.$$

- This, plus NOT...



$$:= \begin{cases} 0 \mapsto 1 \\ 1 \mapsto 0 \end{cases}$$

...gives us enough to start building interesting stuff.

Outline

Process theories

Non-separability

One-time pad

Quantum teleportation

An incredibly sophisticated security protocol

- Suppose Aleks and Bob each have an envelope with the same (random) bit sealed inside

An incredibly sophisticated security protocol

- Suppose Aleks and Bob each have an envelope with the same (random) bit sealed inside
- Aleks wants to send a bit to Bob, but is paranoid (as usual)

An incredibly sophisticated security protocol

- Suppose Aleks and Bob each have an envelope with the same (random) bit sealed inside
- Aleks wants to send a bit to Bob, but is paranoid (as usual)
- He opens his envelope, and tells Bob if the bit inside is the same as the one he wants to send

An incredibly sophisticated security protocol

- Suppose Aleks and Bob each have an envelope with the same (random) bit sealed inside
- Aleks wants to send a bit to Bob, but is paranoid (as usual)
- He opens his envelope, and tells Bob if the bit inside is the same as the one he wants to send
- Bob opens his envelope, and:

An incredibly sophisticated security protocol

- Suppose Aleks and Bob each have an envelope with the same (random) bit sealed inside
- Aleks wants to send a bit to Bob, but is paranoid (as usual)
- He opens his envelope, and tells Bob if the bit inside is the same as the one he wants to send
- Bob opens his envelope, and:
  - if the bits matched before, Bob now has Aleks' bit,

An incredibly sophisticated security protocol

- Suppose Aleks and Bob each have an envelope with the same (random) bit sealed inside
- Aleks wants to send a bit to Bob, but is paranoid (as usual)
- He opens his envelope, and tells Bob if the bit inside is the same as the one he wants to send
- Bob opens his envelope, and:
    - if the bits matched before, Bob now has Aleks' bit,
    - otherwise he flips the bit.

One-time pad with relations

- we can represent the envelopes with the shared random bit as a cup-state:

$$:= \left\{ * \mapsto \{(0,0),(1,1)\} \right.$$

One-time pad with relations

- we can represent the envelopes with the shared random bit as a cup-state:

$$:= \Big\{ * \mapsto \{(0,0),(1,1)\}$$

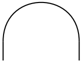
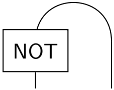
- then checking whether two bits are the same is a '*measurement*' that Aleks can perform on his systems

One-time pad with relations

- we can represent the envelopes with the shared random bit as a cup-state:

$$:= \left\{ * \mapsto \{(0,0),(1,1)\} \right.$$

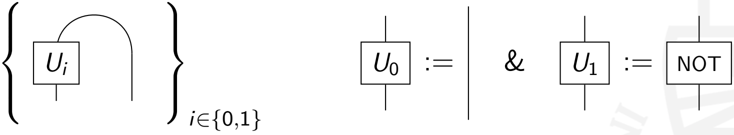
- then checking whether two bits are the same is a '*measurement*' that Aleks can perform on his systems
- There are two possible outcomes:

$\Bigg\{$  := "the same" , 

:= "NOT the same" $\Bigg\}$

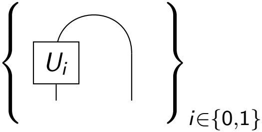
One-time pad with relations

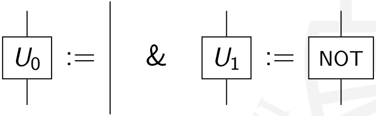
- ...which we can write as:



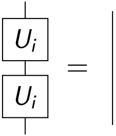
One-time pad with relations

- ...which we can write as:



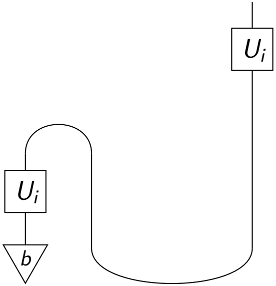


- Then, the U_i satisfy:



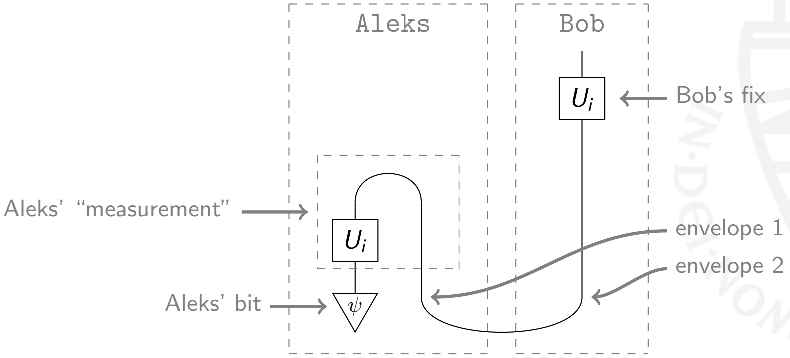
One-time pad diagram

So, the OTP protocol looks like this:

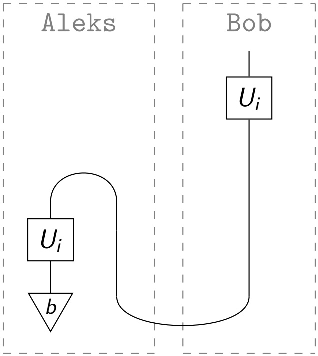


One-time pad diagram

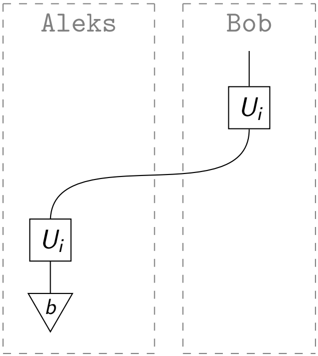
So, the OTP protocol looks like this:



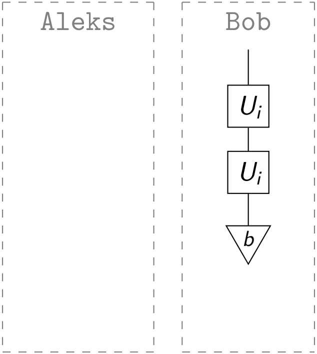
...and it works



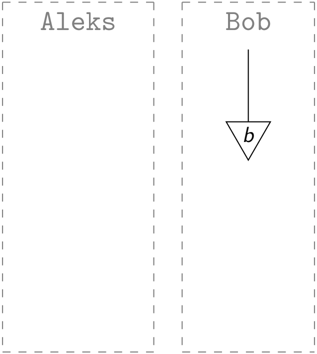
...and it works



...and it works



...and it works



Outline

Quantum bits

- We go from classical to quantum by changing the process theory:

relations $\Rightarrow$ **quantum maps**

Quantum bits

- We go from classical to quantum by changing the process theory:

  relations $\Rightarrow$ **quantum maps**

- The quantum analogue to a bit is a qubit, which represents the state of the simplest non-trivial quantum system

Quantum bits

- We go from classical to quantum by changing the process theory:

relations ⇒ quantum maps

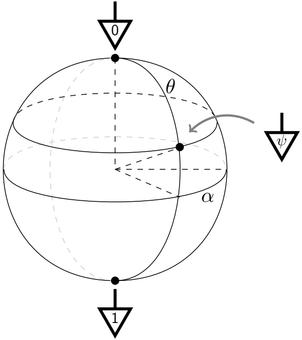
- The quantum analogue to a bit is a qubit, which represents the state of the simplest non-trivial quantum system
- Example: polarization of a photon

Quantum bits

- The state space of a bit consists of two points: 0 and 1

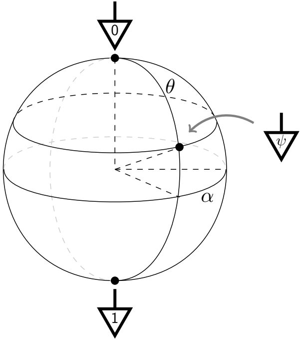
Quantum bits

- The state space of a bit consists of two points: 0 and 1
- ...whereas qubits, it forms a sphere:



Quantum bits

- The state space of a bit consists of two points: 0 and 1
- ...whereas qubits, it forms a sphere:



- "Plain old" bits live at the North Pole and the South Pole.

Quantum entanglement

- In quantum-land, we can realise a ‘cup’ using *quantum entanglement*

 $\Longleftarrow$ “Bell state”

Quantum entanglement

- In quantum-land, we can realise a 'cup' using *quantum entanglement*


$\Longleftarrow$ "Bell state"

- Even though this thing is (slightly) more complicated to describe, it acts just like before

Quantum measurement

- We also have a quantum analogue for Aleks' measurement:

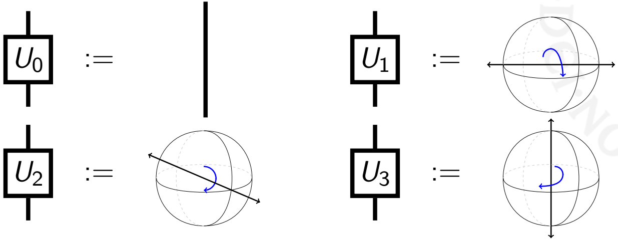
$\Longleftarrow$ "Bell measurement"

Quantum measurement

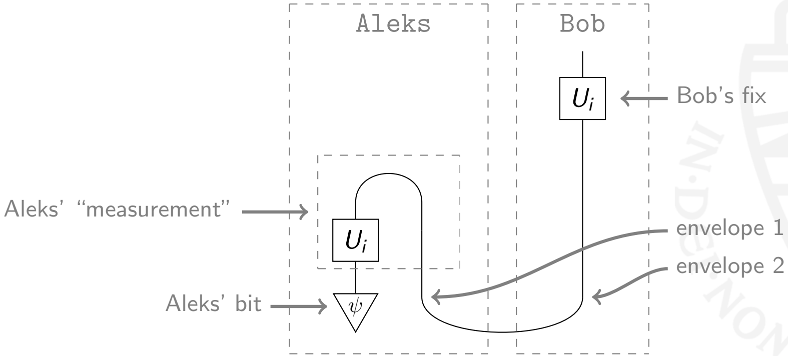
- We also have a quantum analogue for Aleks' measurement:

$\Longleftarrow$ "Bell measurement"

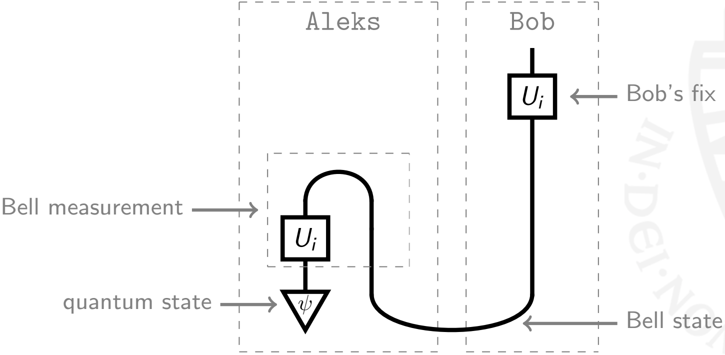
where there are now three different ways to "NOT":



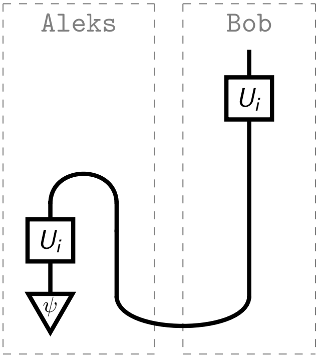
OTP ⇒ quantum teleportation



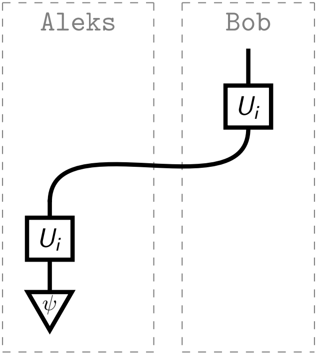
OTP ⇒ quantum teleportation



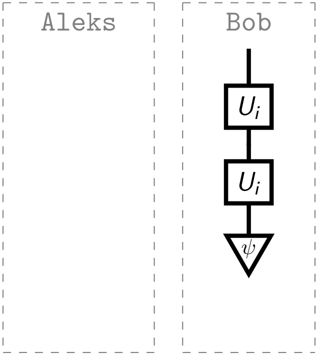
...and it works



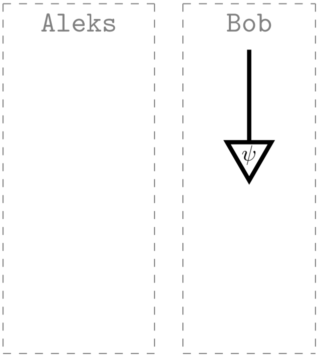
...and it works



...and it works



...and it works



Two for the price of one

- **The moral:** In both OTP and teleporation, Aleks must send Bob i, otherwise the whole thing fails

Two for the price of one

- **The moral:** In both OTP and teleporation, Aleks must send Bob *i*, otherwise the whole thing fails
- By using a **shared resource**:

 := shared random bit

 := Bell state

Two for the price of one

- **The moral:** In both OTP and teleporation, Aleks must send Bob i, otherwise the whole thing fails
- By using a **shared resource**:

 := shared random bit

 := Bell state

- Aleks can send **one kind of thing**:

$i \in \{0, 1\}$:= public data

$i \in \{0, 1, 2, 3\}$:= classical data

Two for the price of one

- **The moral:** In both OTP and teleporation, Aleks must send Bob i, otherwise the whole thing fails
- By using a **shared resource**:

 := shared random bit

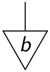
 := Bell state

- Aleks can send **one kind of thing**:

$i \in \{0, 1\}$:= public data

$i \in \{0, 1, 2, 3\}$:= classical data

- ...and Bob gets **another kind of thing**:



:= private data

:= quantum state

Thanks!