# String Diagramを学ぶ

量子過程を図解する
String Diagram 入門

# String Diagramを学ぶ

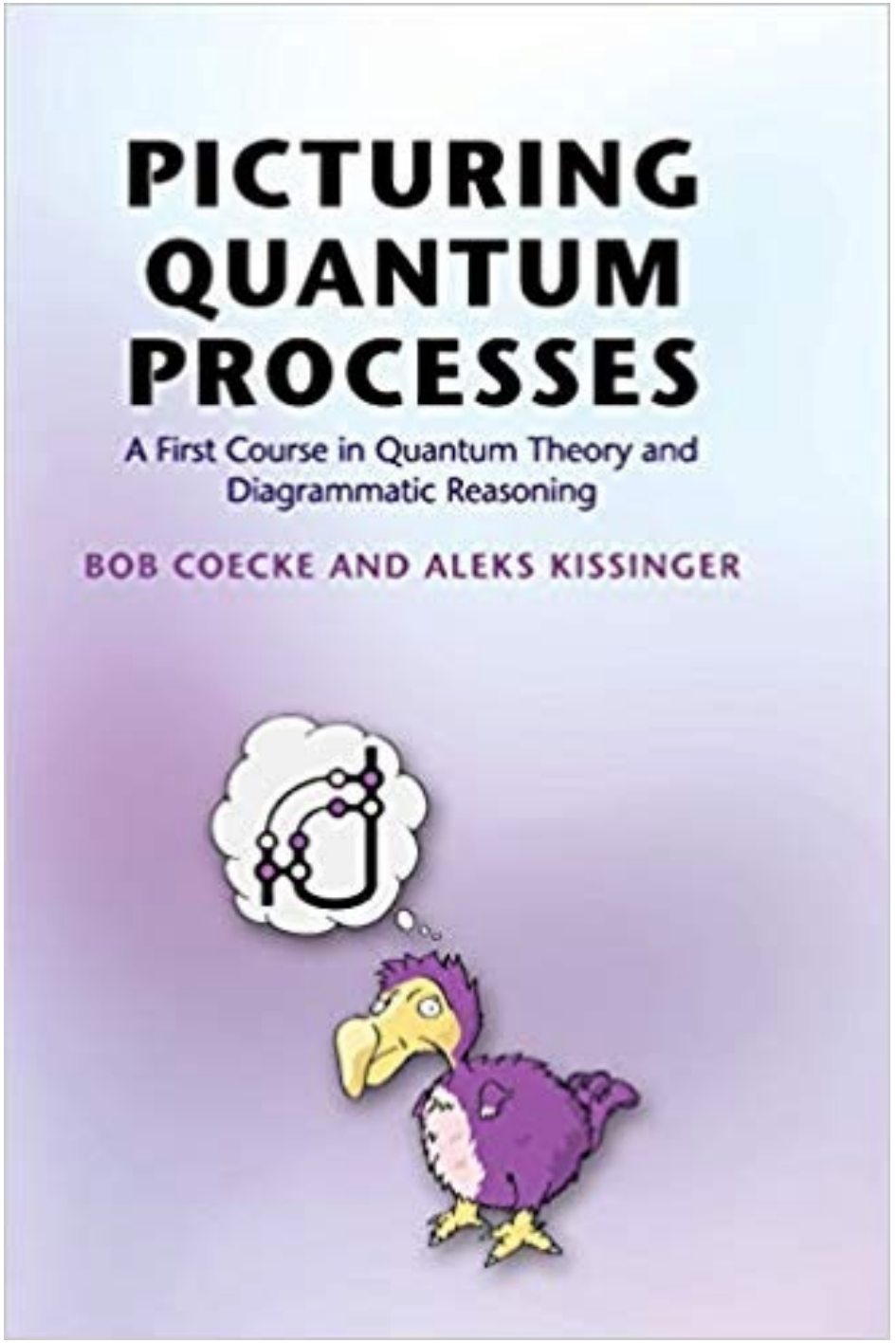
はじめに

# String Diagram とは何か？

String Diagramは、直観では理解しにくい（むしろ直観に反した）量子過程の諸特徴を、数式を使わずに「図式Diagram」を使って、直観的に理解することを可能にする新しい方法である。

String Diagram については、2019/03/25に開催したマルレク「量子コンピュータをやさしく理解する三つの方法」でも簡単に紹介したのだが量子論に対するこのアプローチは、重要だと僕は考えている。改めてすこし詳しく紹介しようと思う。

このシリーズは、Bob Coeckeの“Picturing Quantum Processes” に基本的に依拠している。 https://amzn.to/2MbE5Xh



# String Diagramと従来のアプローチ

- String Diagramと従来のアプローチとの比較を次の四点で見てみよう。

1. 「状態」と「過程」
2. 「重ね合わせ」と「エンタングルメント」
3. 「量子過程」と「情報過程」
4. 「数式」と「図式」

# 「状態」と「過程」

- フォン・ノイマンによって定式化された従来のアプローチでは、量子の「状態」をヒルベルト空間上のベクトルとして捉える。
- 「観測可能量」は、ヒルベルト空間上のエルミート演算子に対応し、実際に観測される「状態」の確率は、ボルンのルールで与えられる。
- String Diagramでは、全てのものが「過程 process」として捉えられる。「状態」も「観測結果」と、特殊な「過程 process」の一つである。
- こうした理論を Process Theory と呼ぶ。String Diagramは、Process Theory である。

*I would like to make a confession which may seem immoral: I do not believe absolutely in Hilbert space anymore.*

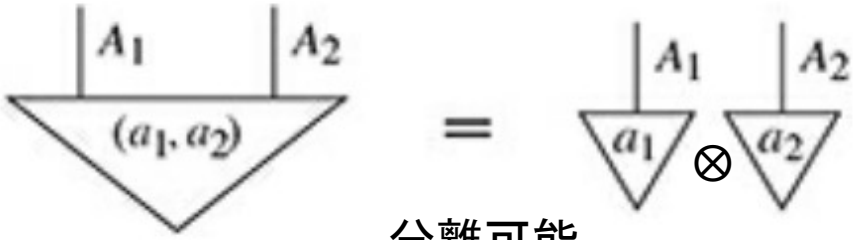
*— John von Neumann,*
*letter to Garrett Birkhoff, 1935.*

# 「重ね合わせ」と「エンタングルメント」

- 従来の定式化は、量子論の主要な特徴を「状態」の「重ね合わせ」と「観測」の「確率解釈」として捉える。ただ、それは、「エンタングルメント」のような量子過程に、直接フォーカスしているわけではない。

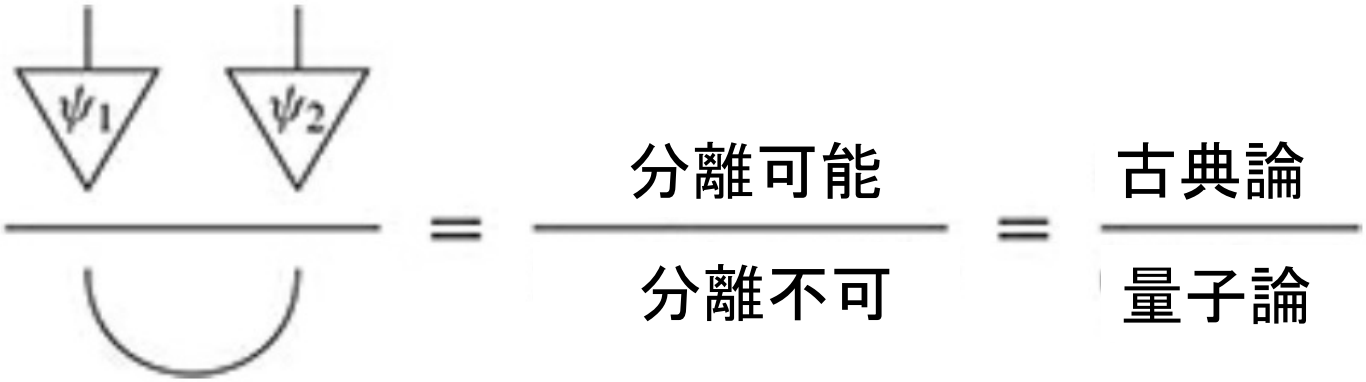
- String Diagramは、エンタングルメントのような「不可分離性 non-separability」を量子論の特徴として捉えようとする。古典論は、それとの対比で同じ枠組みの中で、「分離可能 sepalable」な理論として特徴づけられる。

ある状態が、二つの状態のテンソル積で
表されるとき、その状態を、「分離可能」という




分離可能

古典論では、全ての状態が「分離可能」であり、
量子論は、「分離不可」の状態を含む




String Diagramでは、Entanglement は、
こうした記号 “Cup” であらわされる。

# 「量子過程」と「情報過程」

- 従来のアプローチでは、量子過程の情報的側面には、ほとんど関心が向けられていなかった。
- String Diagramでは、量子コンピュータ技術・量子通信技術の立ち上がりを背景に、量子テレポーテーション、エンタングルメント・スワップメント等の量子過程の情報的側面に大きな関心が寄せられている。
- こうした動きは、「量子過程」＝「情報過程」という現代物理学の新しい流れに連動している。

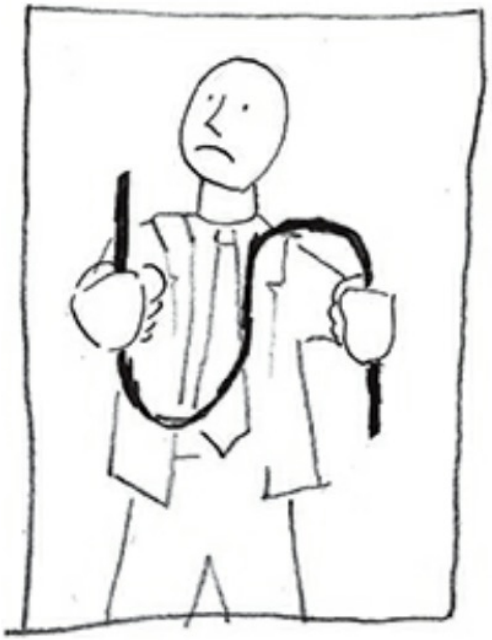
*Coecke: "Why it took 60 years for quantum teleportation to be discovered?"*

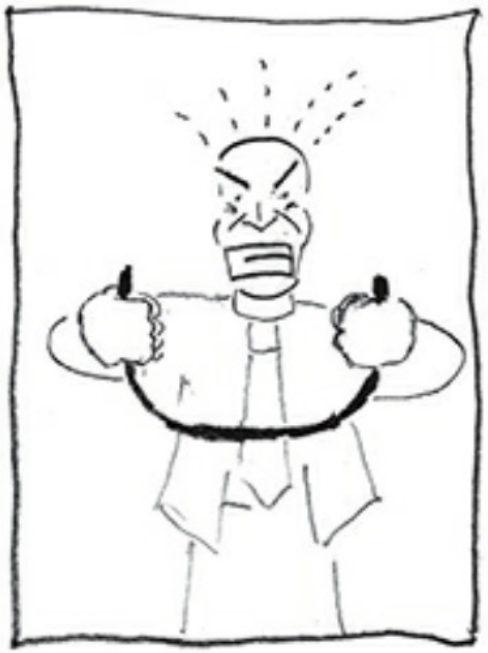
*Brassard: "No one before had considered the information-processing features of quantum theory and had therefore simply not thought to ask the question."*

# 「数式」と「図式」

- 「数式」の方が「図式」より、正確で情報量が多いというのは、ある種の思い込みである。
- String Diagramの「図式」は、「数式」で表現される抽象的テンソル・システムと同値であり、「数式」＋「図式」で表現されるカテゴリー論のTrace付きのSymmetric Monoidal Categoryとも同じ表現力を持っている。
- String Diagramの「図式」は、直観的で分かりやすいだけでなく、それを理解するのに、カテゴリー論を学ぶ必要もない。

# String Diagramの基本的定理
# Yanking 等式





# カテゴリー論入門

第一部
量子過程を図解する String Diagram 入門

第二部
String DiagramとSymmetric Monoidal Category

第三部
カテゴリー論の応用の射程

第四部
Homotopy Type Theory

# String Diagramを学ぶ

量子過程を図解する
String Diagram 入門

# String Diagramを学ぶ
# **Agenda**

はじめに

第一部　String Diagramの基本

第二部　量子回路をDiagramで表す

第三部　プロセスの理論

第四部　String Diaggramの世界

# String Diagramを学ぶ

## 第一部　String Diagramの基本

# String Diagramを学ぶ
# 第一部　String Diagramの基本

Diagram = Box + Wire

Diagramの「等式」

プロセスの「等式」

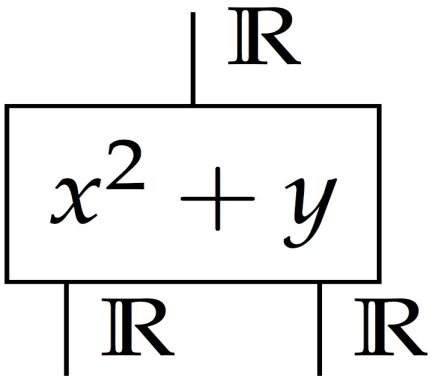
Diagramの並列合成

Diagramの直列合成

Diagram = Box + Wire

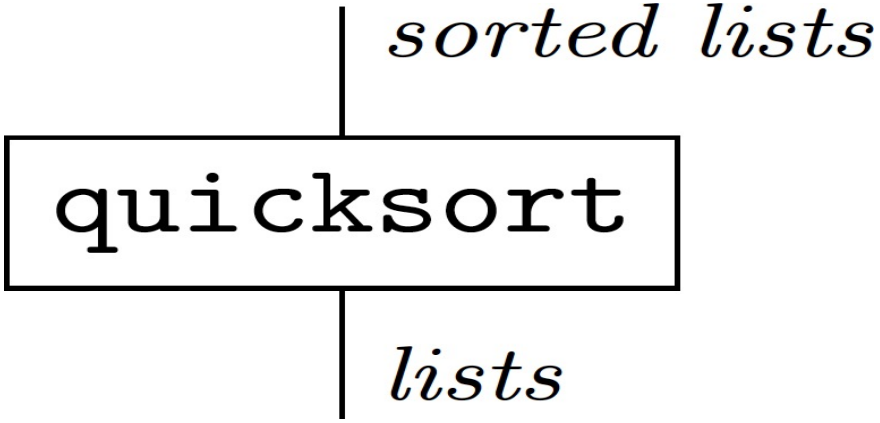
# Diagram = Box + Wire

- Diagramは、BoxをWireでつなげたものである。
- 次のものは、Diagramである。




- このDiagramは、二つの実数 x と y を入力として、実数 $x^2+y$を出力として返すプロセスを表している。

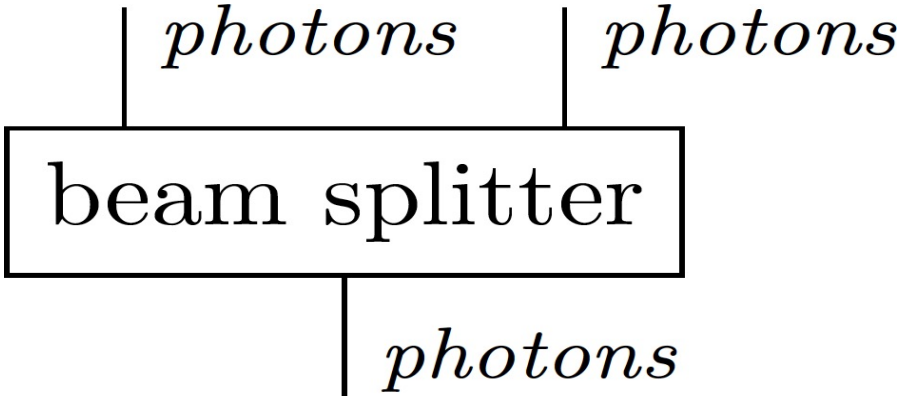
# Diagramの例




このDiagramは、プログラム quicksort で、リストが整列される過程を表現している。

# Diagramの例

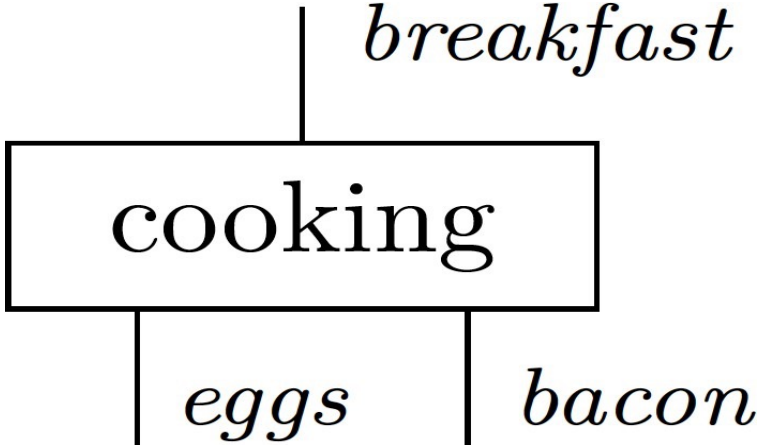
Diagramが表現するのは
計算やプログラムだけとは限らない




このDiagramは、ビーム・スプリッターで、光(photons)が、
二つの光(photons)に分岐する過程を表現している。

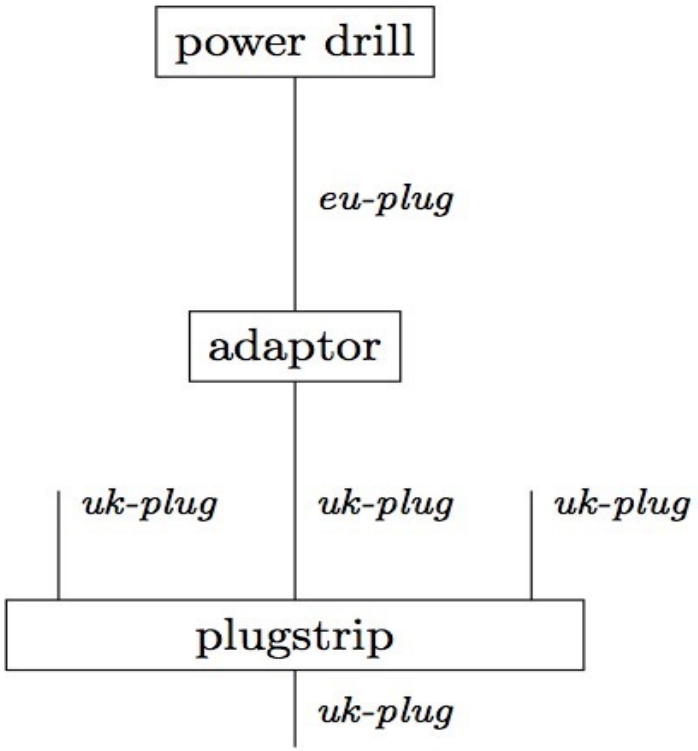
# Diagramの例

Diagramが表現するのは
計算やプログラムだけとは限らない

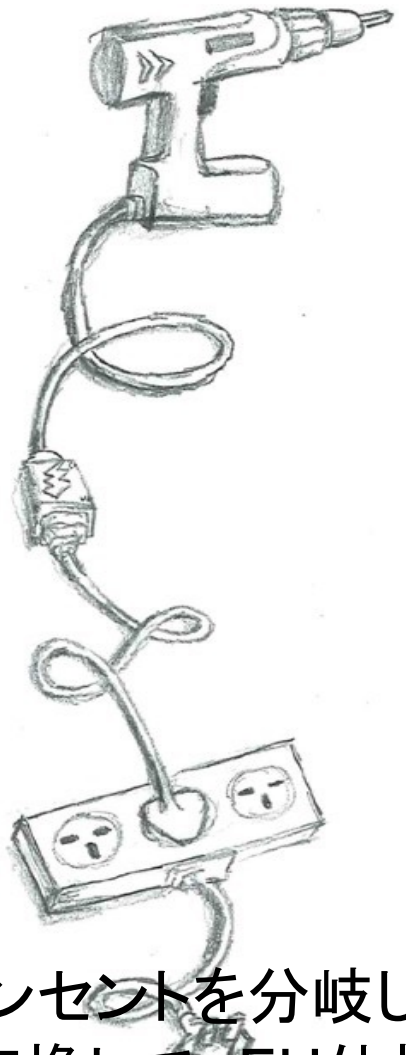



このDiagramは、卵とベーコンを調理して、朝食を作る過程を表現している。

# Diagramの例



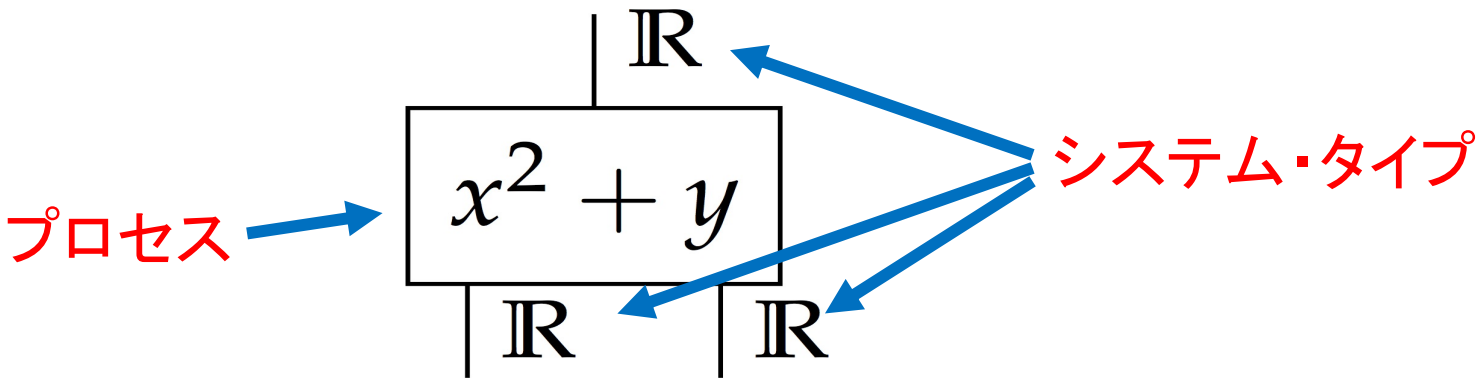

:=



このDiagramは、イギリス(uk)のコンセントを分岐してアダプターでEU仕様のコンセントに変換して、EU仕様の電気ドリルを使う過程を表現している。

# Diagram = Box + Wire

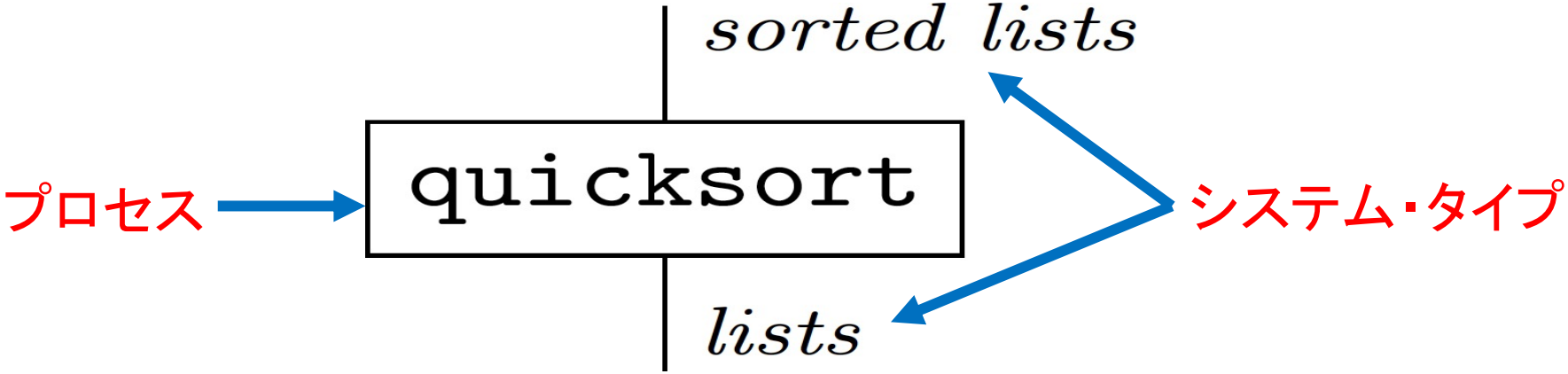
- ダイアグラムは「箱」と「ワイアー」からできている。
- 「箱」は「プロセス」を、「ワイアー」は「システム・タイプ」を表している。




- このDiagramは、二つの実数 x と y を入力として、実数 $x^2+y$を出力として返すプロセスを表している。

# Diagram = Box + Wire

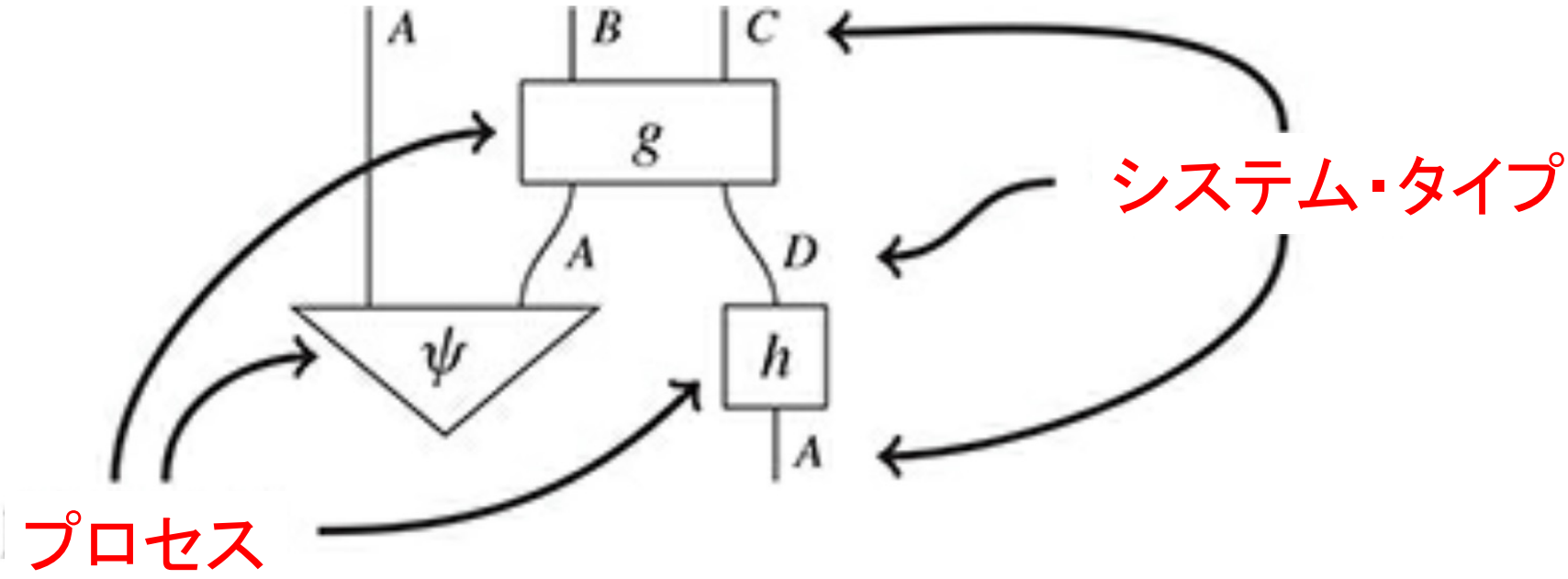
ダイアグラムは「箱」と「ワイアー」からできている。
「箱」は「プロセス」を、「ワイアー」は「システム・タイプ」を表している。



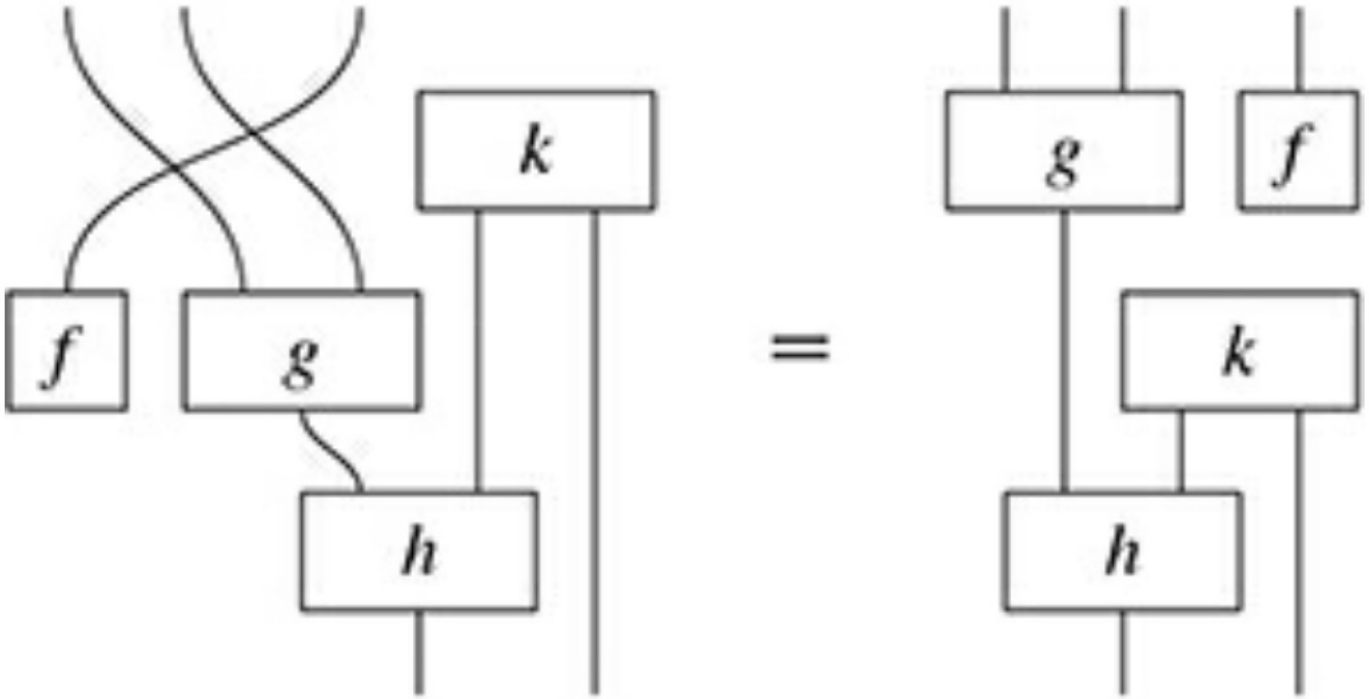

このDiagramは、プログラム quicksort で、リストが整列される過程を表現している。

# Diagram（ダイアグラム）

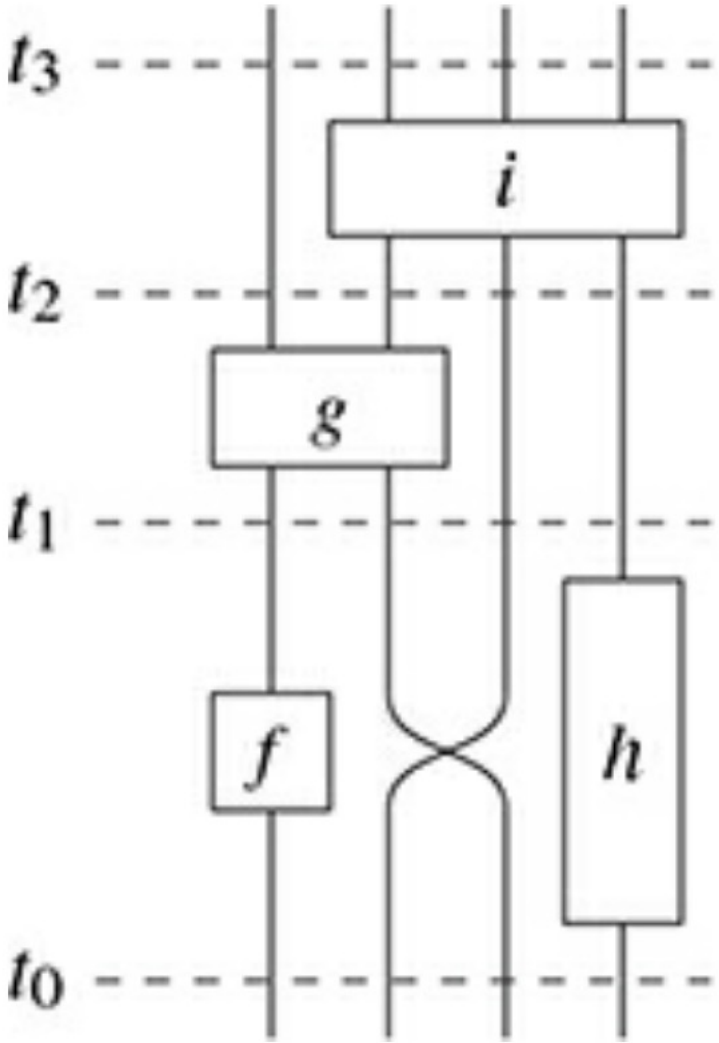
ダイアグラムは「箱」と「ワイアー」からできている。
「箱」は「プロセス」を、「ワイアー」は「システム・タイプ」を
表している。



# Diagramの黄金律
# どうつながっているかだけが重要

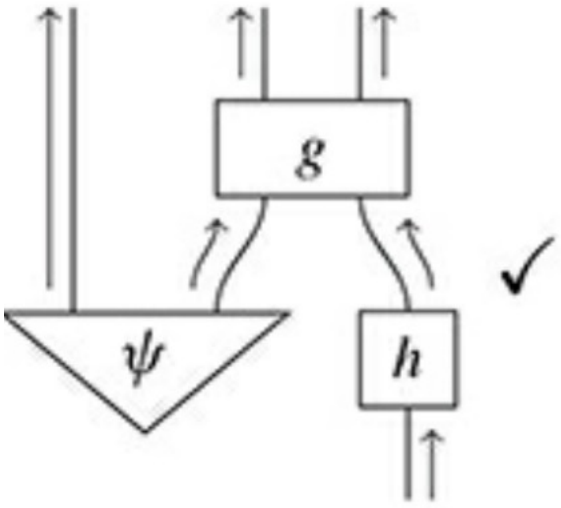


# 未来と過去
## Diagramは、時間のステップにそって組織される

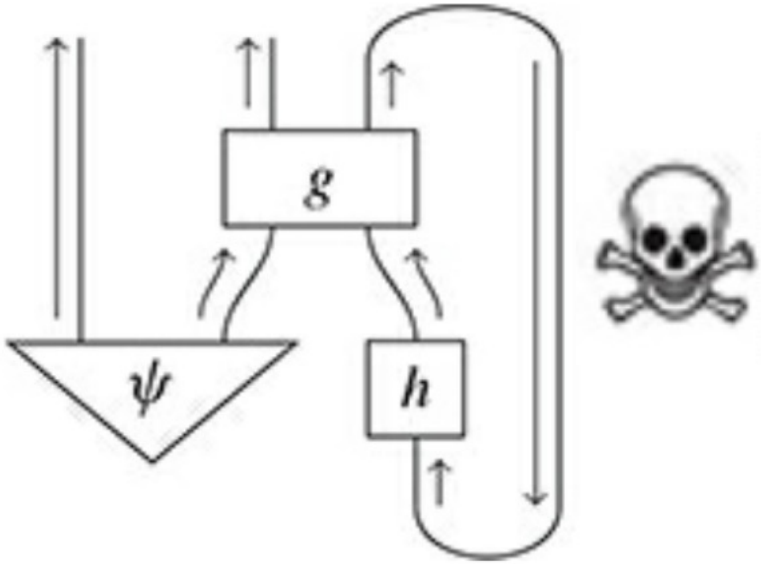



時間の流れ

# 回路のDiagram
# ループを含んでいてはいけない

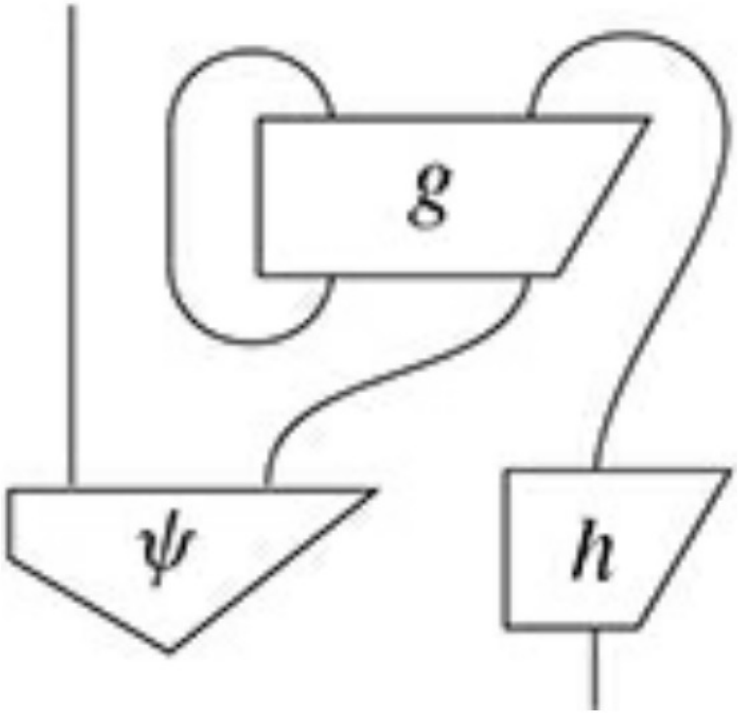



✓



# String Diagrams

String diagramsは、ループの禁止を取り除いたDiagramである。さらに、入力を入力に、出力を出力につなげるのもゆるされる。



# Diagram = Process Theory

- ここでは、特定のシステムのプロセスではなく、一般的なプロセスの理論の特徴づけをしてみよう。

1. 「Diagram=プロセスの理論」は、Wireで表現される「システム・タイプ」T を持つ。
2. 「Diagram=プロセスの理論」は、Boxで表現される「プロセス」P を持つ。
3. Box(プロセス)をWire(システム型)で結ぶことで、その理論のプロセスは表現される。

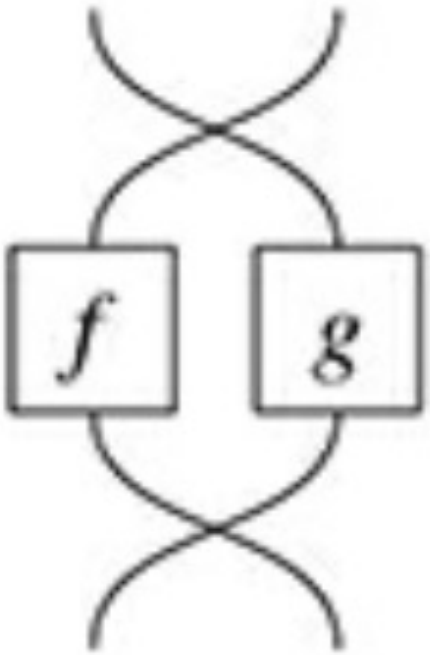
# Diagramの「等式」

# Diagramの「等式」

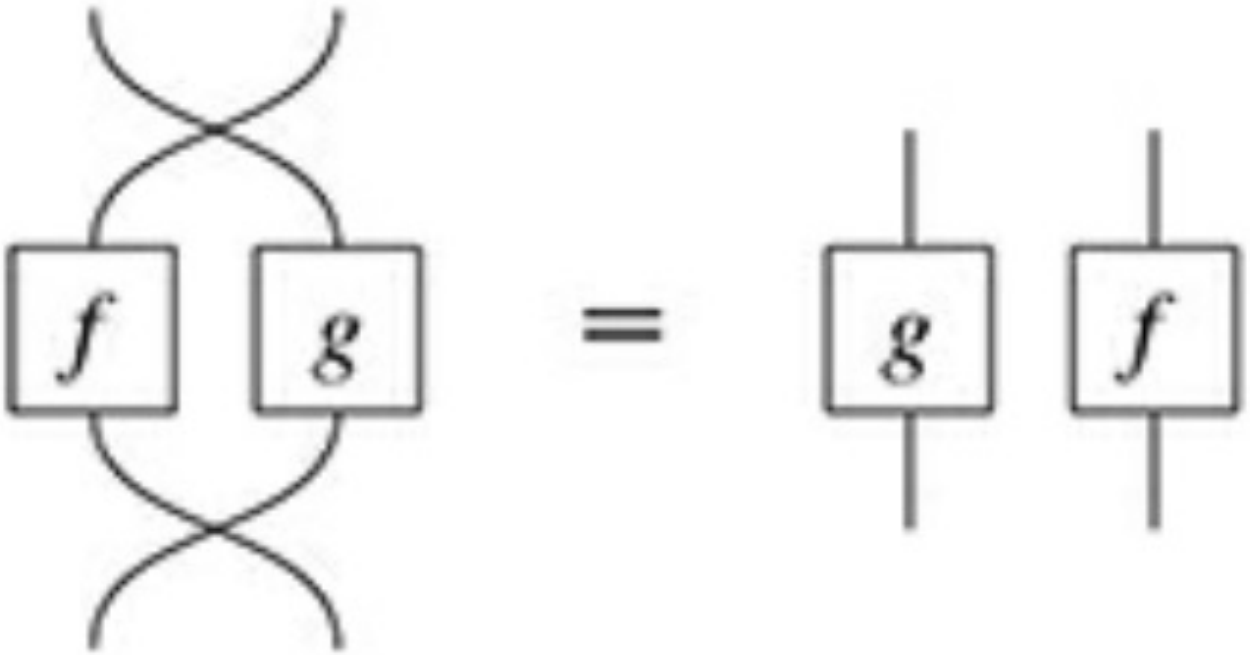
- 二つのDiagramが、Wireの接続を変えずに、互いに変形できるとき、二つのDiagramは「等しい」という。
- Diagramにとっては、どうつながっているかだけが重要である。

- Diagramの「等式」の例をいくつか見ておこう。

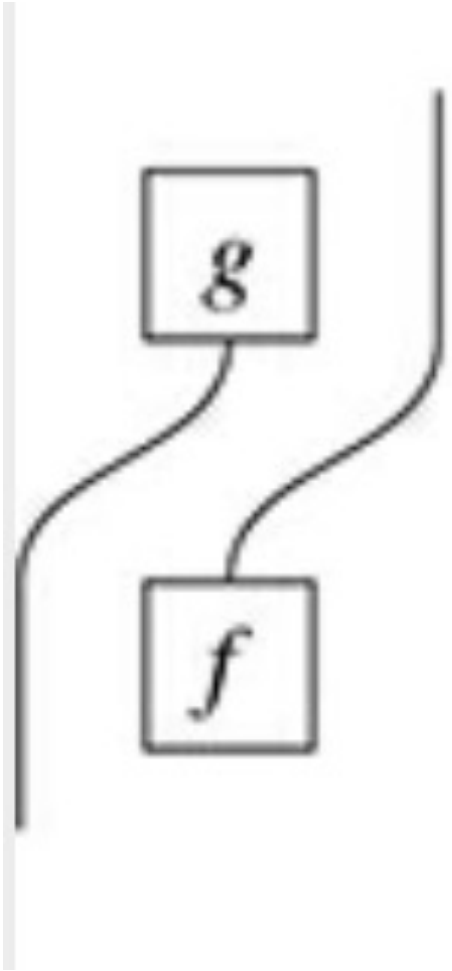
# Diagramの「等式」



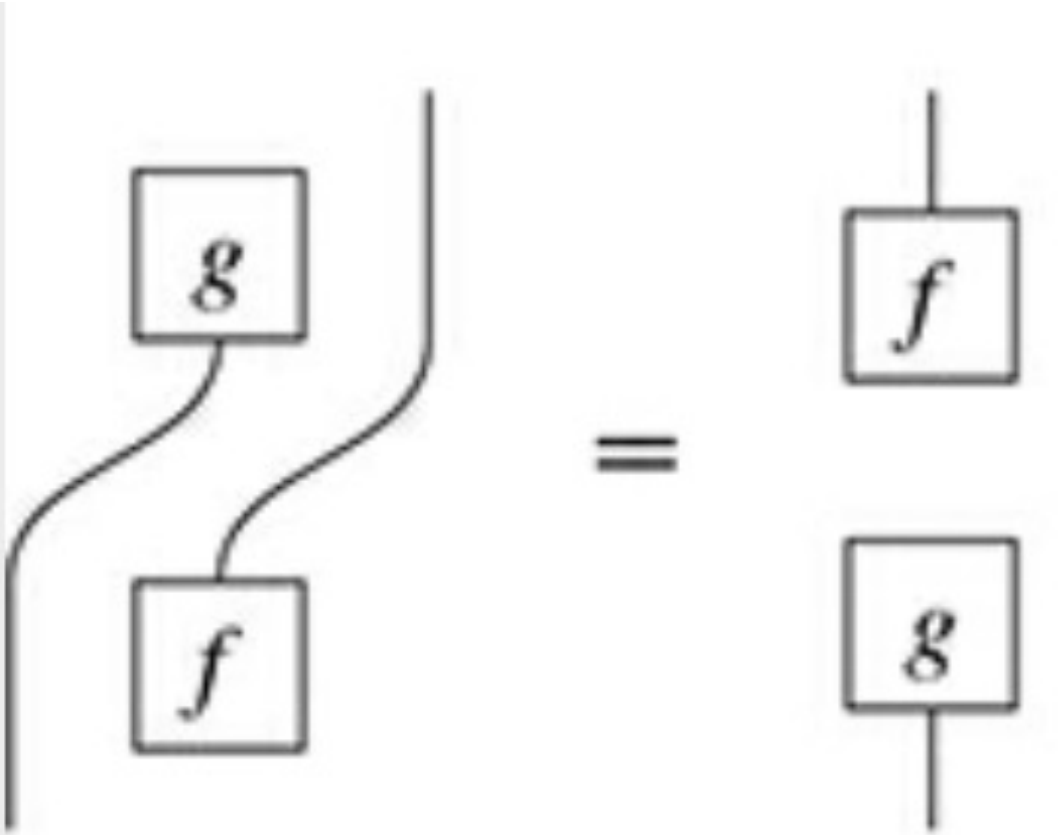
# Diagramの「等式」



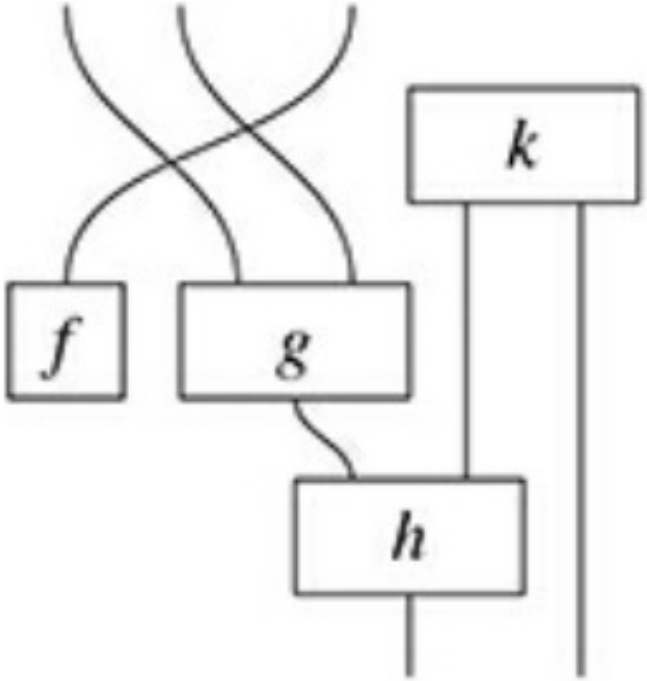
# Diagramの「等式」



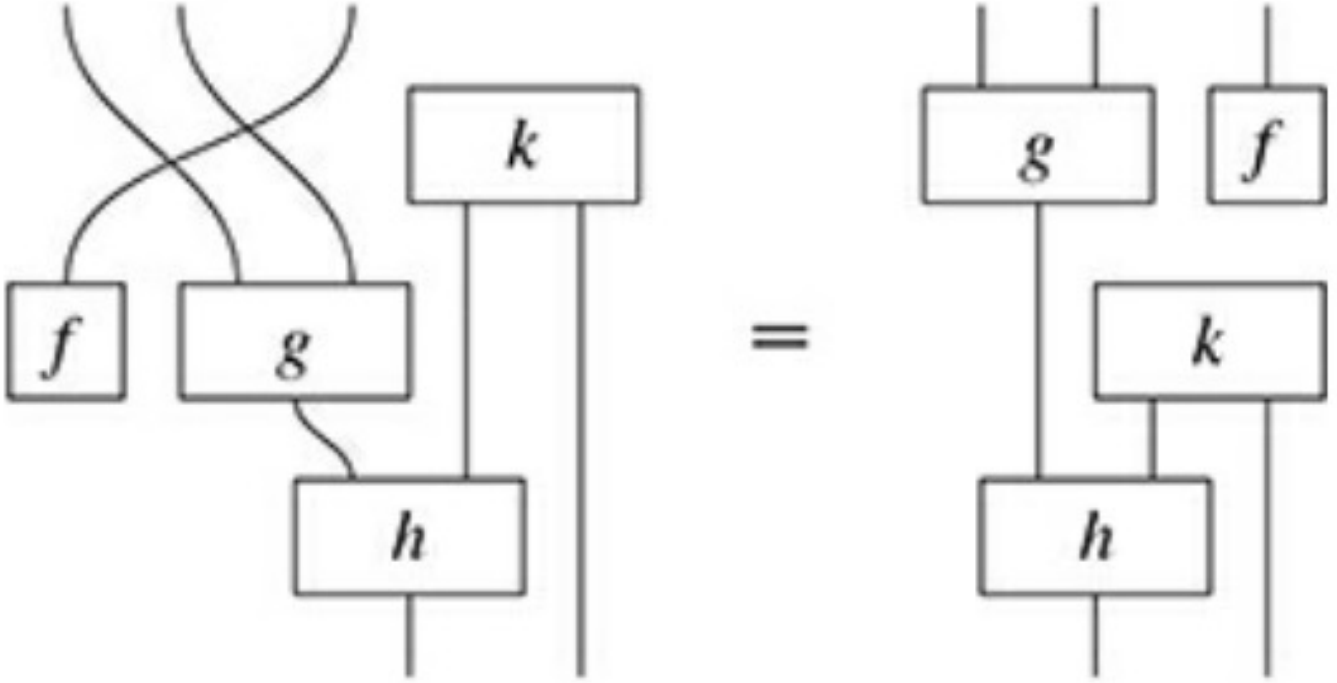
# Diagramの「等式」



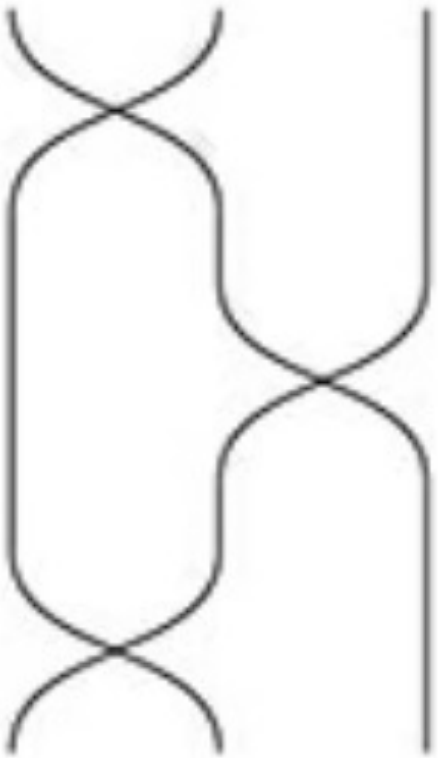
# Diagramの「等式」



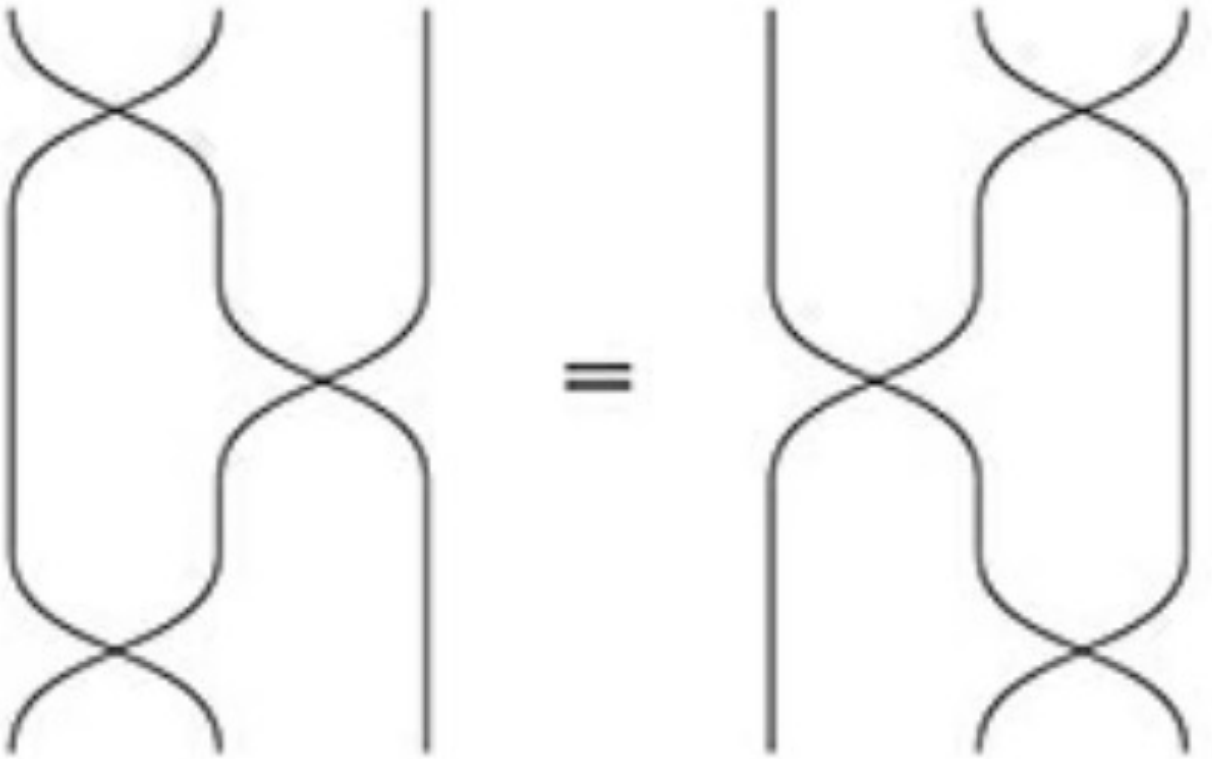
# Diagramの「等式」



# Diagramの「等式」



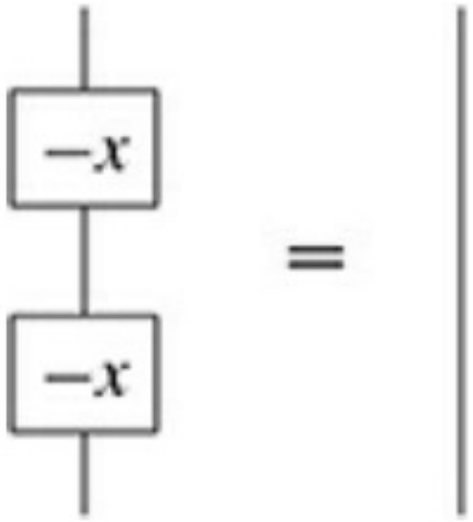
# Diagramの「等式」



# プロセス の「等式」

# プロセスの「等式」

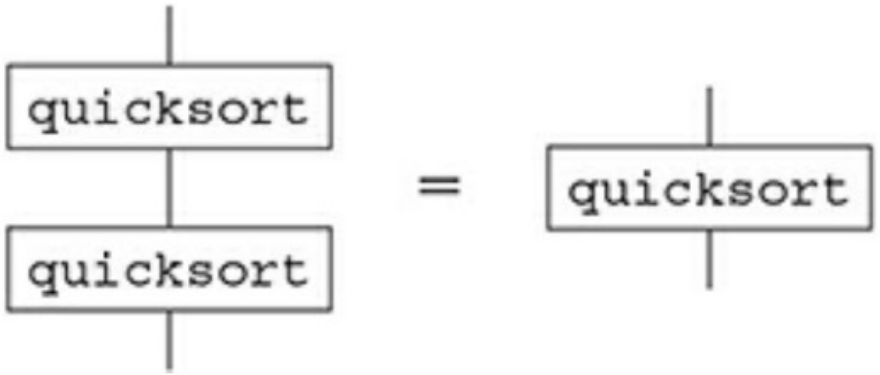
- 一般的には、Diagramとしては等しくないのだが、ある特定の「プロセスの理論」においては成り立つDiagram間の「等式」がある。
- 例えば、次のDiagramのBoxが、数字の符号を反転させるプロセスだとすると、次の「等式」が成り立つ。




符号が二回変われば、
符号は元に戻って、何も
しなかったのと同じになる

# プロセスの「等式」

- コンピュータ・プラグラムの「プロセスの理論」を考えれば、例えば、次のような「等式」が成り立つ。



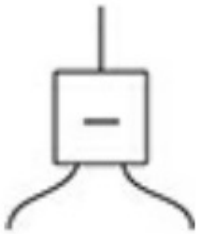

（リストを二回ソートしても、一回ソートしても結果は変わらない。）

# プロセスの「等式」

- ある「プロセスの理論」で成り立つ「等式」を「プロセスの等式」という。
- 「プロセスの等式」は、それぞれの「プロセスの理論」を特徴付ける意味を持つ。

# プロセス の「等式」の例

- ここでは、「関数」の「プロセスの理論」で、「プロセスの等式」を考えてみよう。
- 次の二つの「関数」を考える。

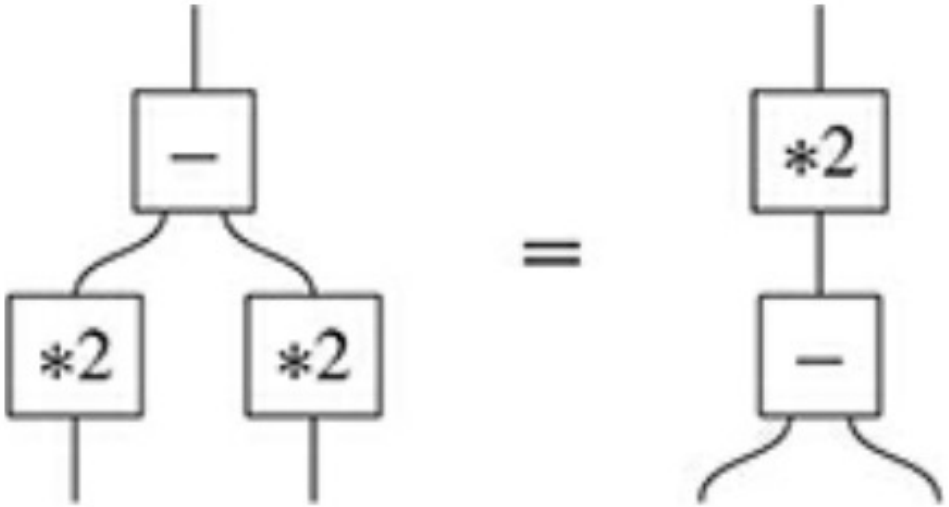




それぞれ入力の差をとる、入力に2を掛けるという関数である。

- この時、次のような「プロセスの等式」が成り立つ。

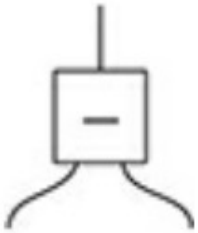
# プロセス の「等式」の例

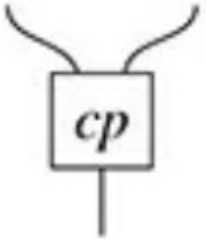



なぜなら、入力をm,nとすると、
左のDiagramは、2*m −2*n という関数を表し、
右のDiagramは、2*(m−n)という関数を表し、
両者は等しい。

# プロセス の「等式」の例

- 次の二つの「関数」を考える。

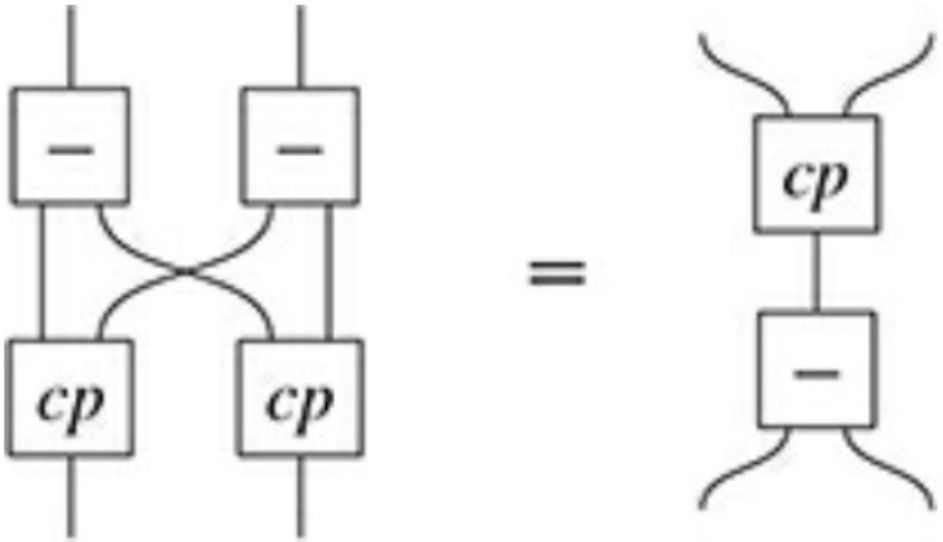


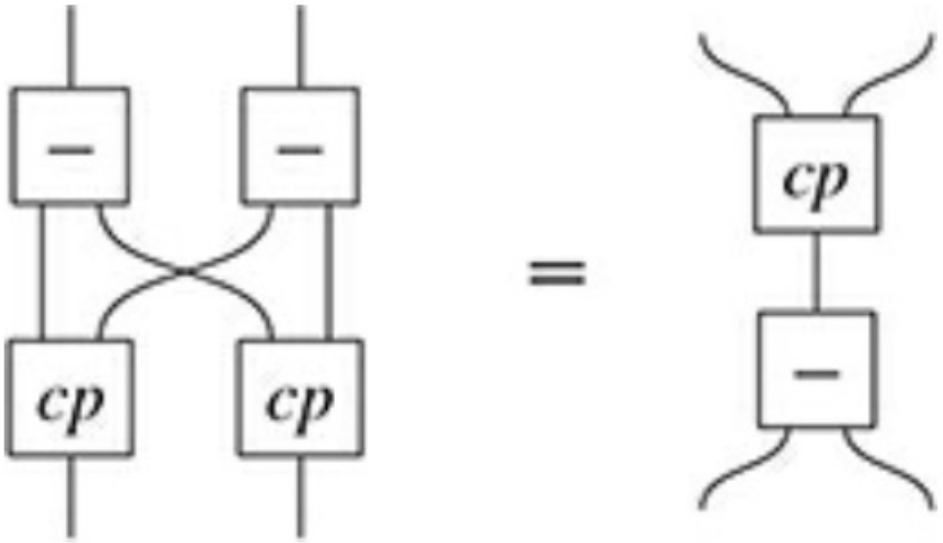

それぞれ入力の差をとる、入力の二つのコピーを出力するという関数である。

- この時、次のような「プロセスの等式」が成り立つ。

# プロセス の「等式」の例



# プロセス の「等式」の例

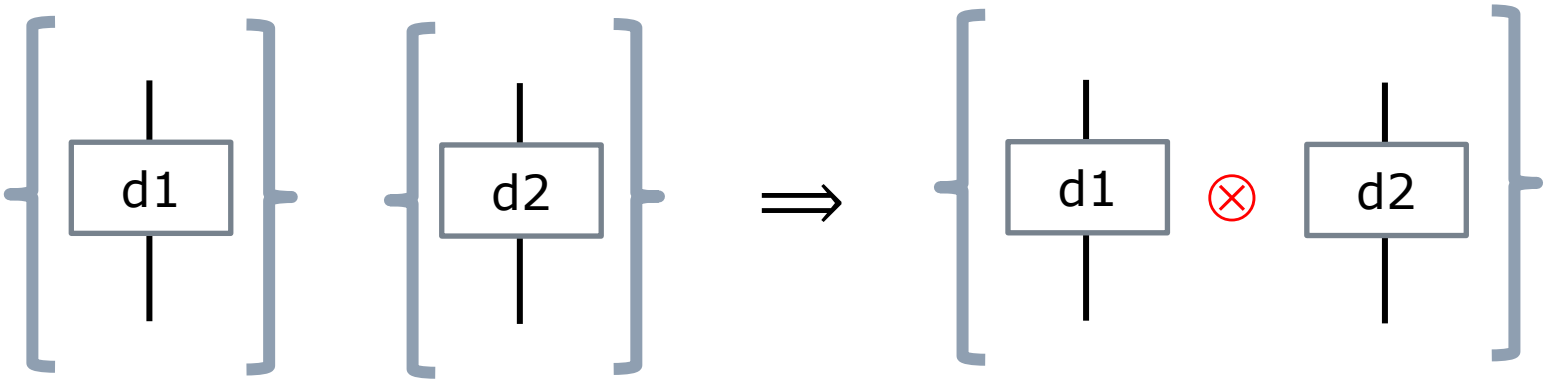



- 各自で確かめよ。

# Diagram を並列に合成する

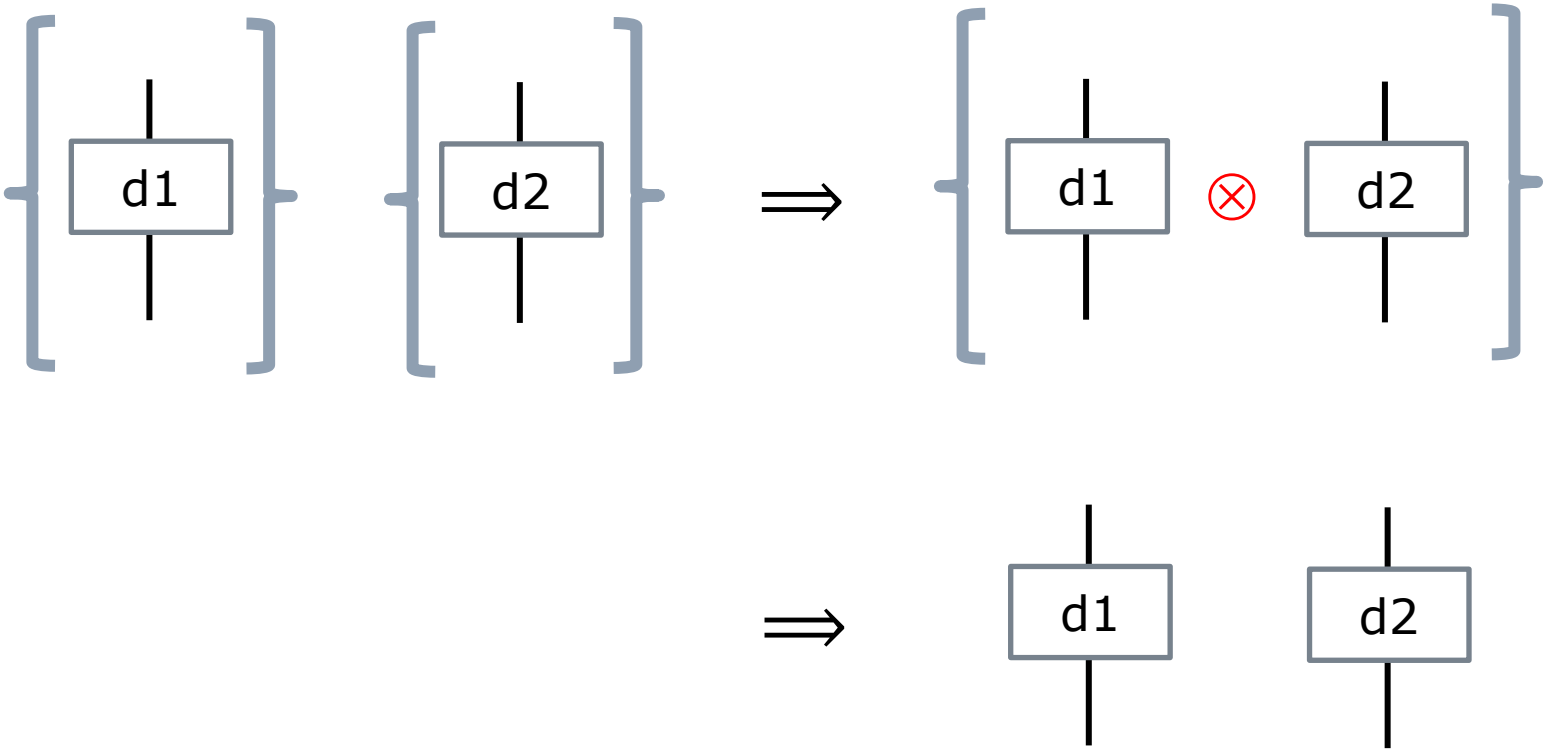
# 並列合成　⊗ 記号

- 「並列合成」 ⊗は、二つの並列なプロセスが同時に走ることを、一つのプロセスとして捉える。



# 並列合成　⊗ 記号の省略

- 「並列合成」 ⊗は、二つの並列なプロセスが同時に走ることを、一つのプロセスとして捉える。「並列合成」の⊗ 記号は省略できる。

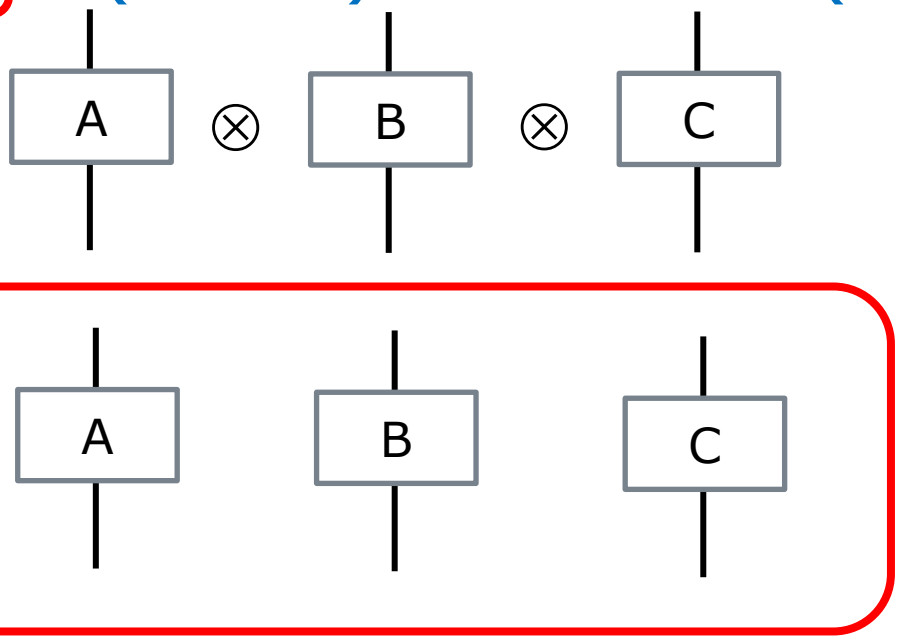


# 並列合成の結合律

$$A \otimes B \otimes C = (A \otimes B) \otimes C = A \otimes (B \otimes C)$$

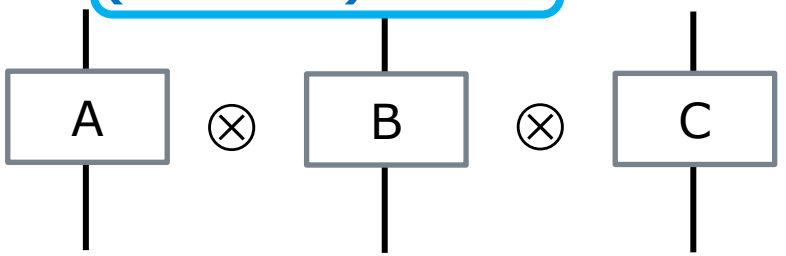
# 並列合成の結合律

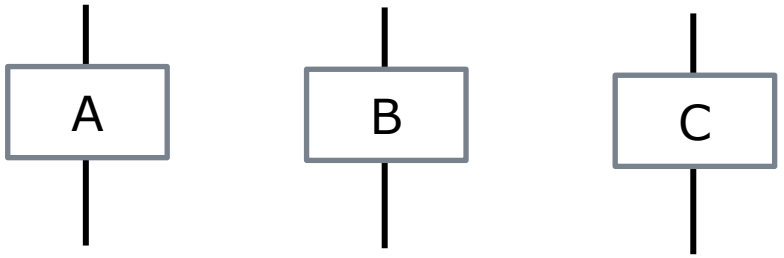
$A \otimes B \otimes C = (A \otimes B) \otimes C = A \otimes (B \otimes C)$

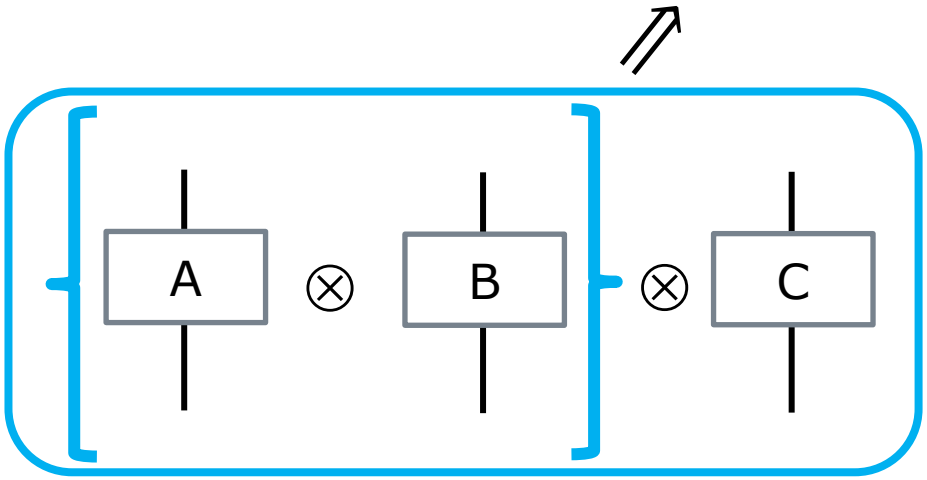


# 並列合成の結合律

$$A \otimes B \otimes C = (A \otimes B) \otimes C = A \otimes (B \otimes C)$$

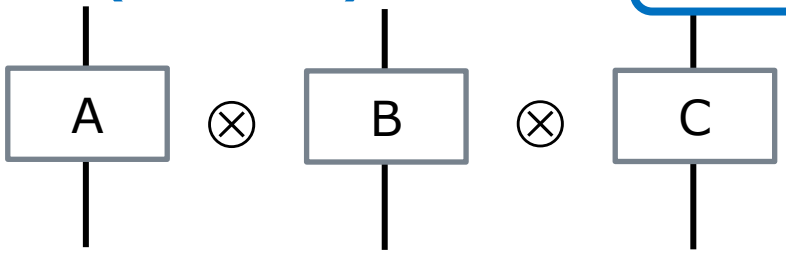


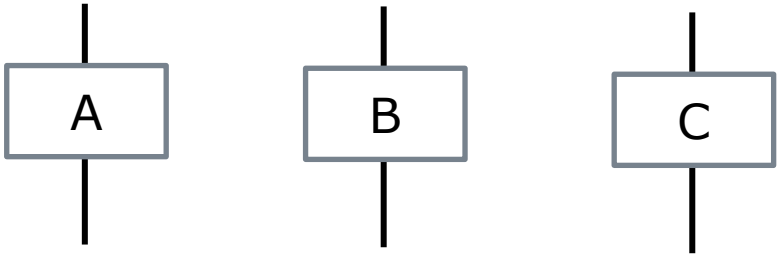



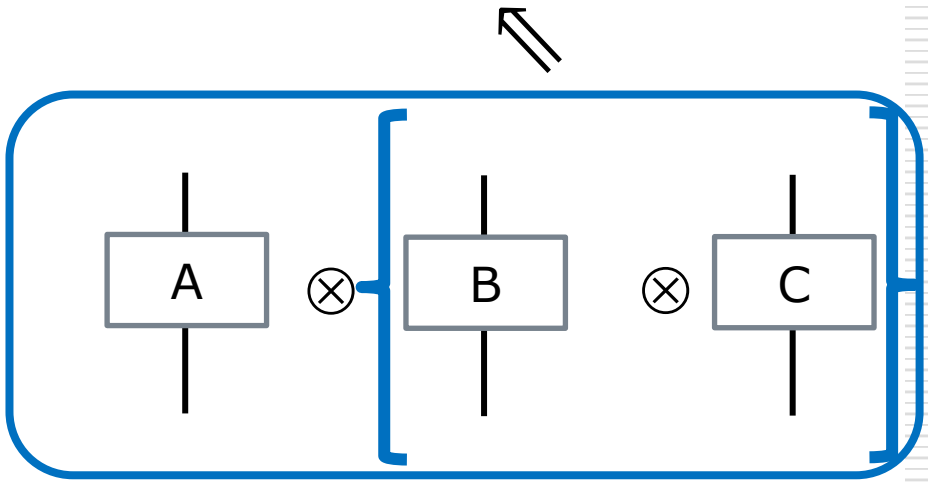



# 並列合成の結合律

$A \otimes B \otimes C = (A \otimes B) \otimes C = A \otimes (B \otimes C)$

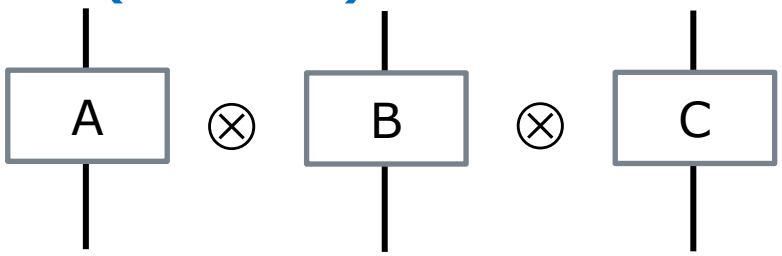


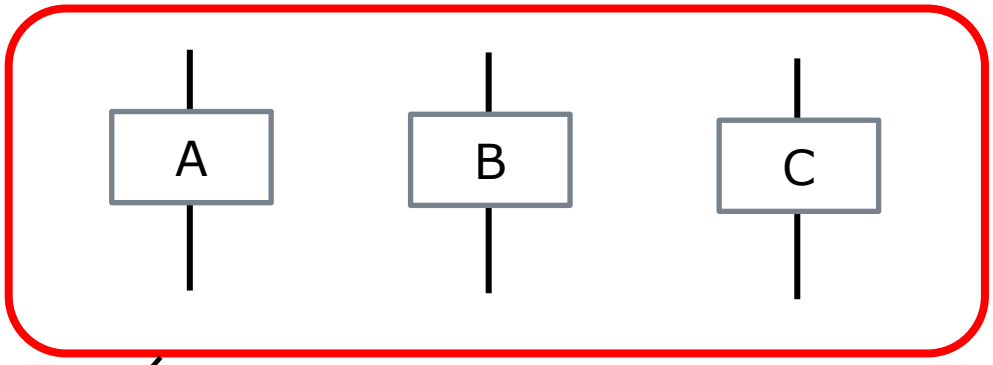



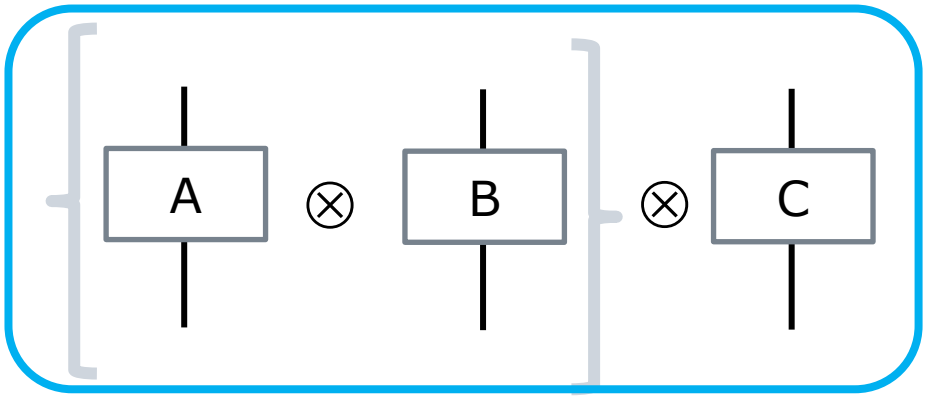
# 並列合成の結合律

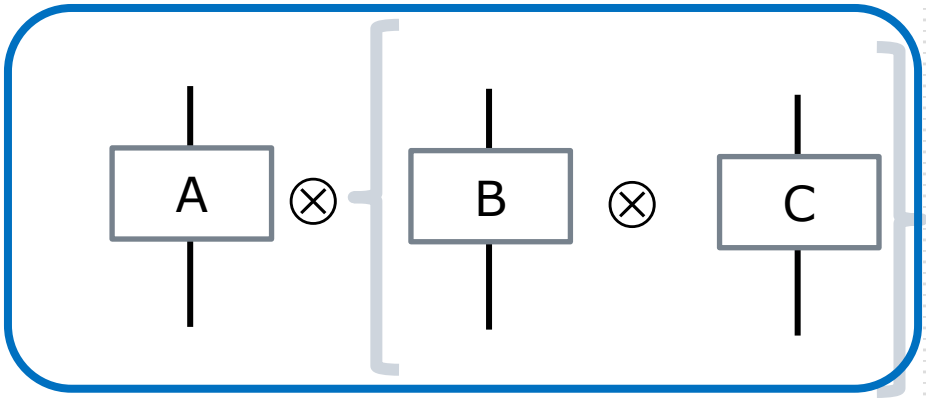
$$A \otimes B \otimes C = (A \otimes B) \otimes C = A \otimes (B \otimes C)$$




Diagramから、
並列合成の
結合律は、
自然に導かれる。



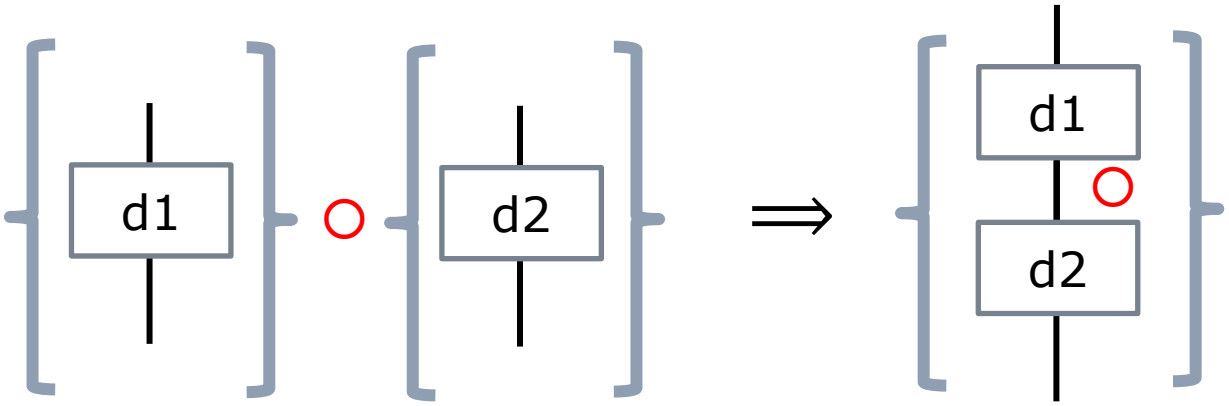



# Diagram を直列に合成する

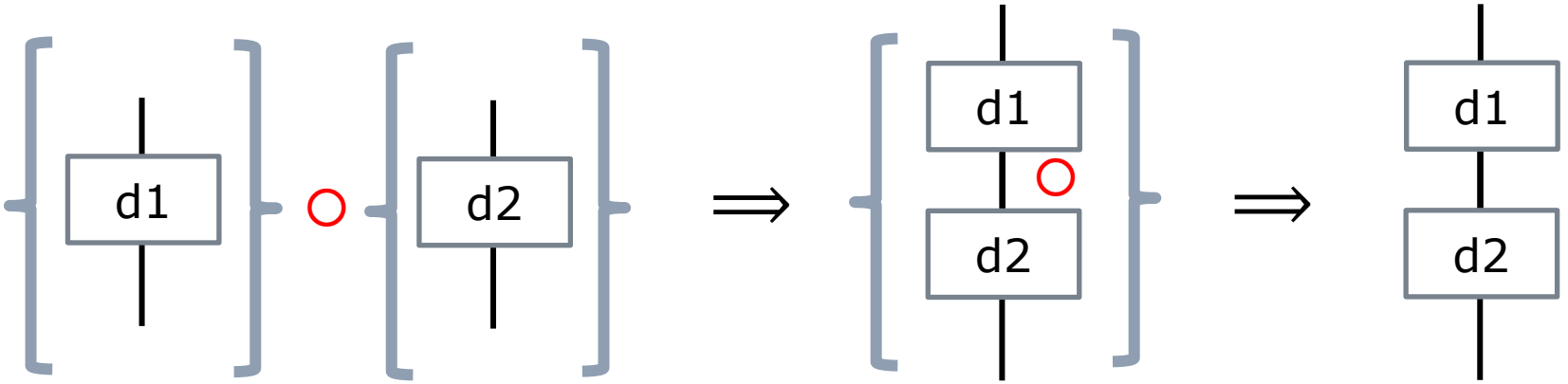
# 直列合成　○ 記号

- 「直列合成」の○ 記号は、二つのプロセス（Box）を直列につなぐ



# 直列合成　○ 記号の省略

- 「直列合成」の○ 記号は、二つのプロセス（Box）をWireで結ぶことで表現されるので省略できる。

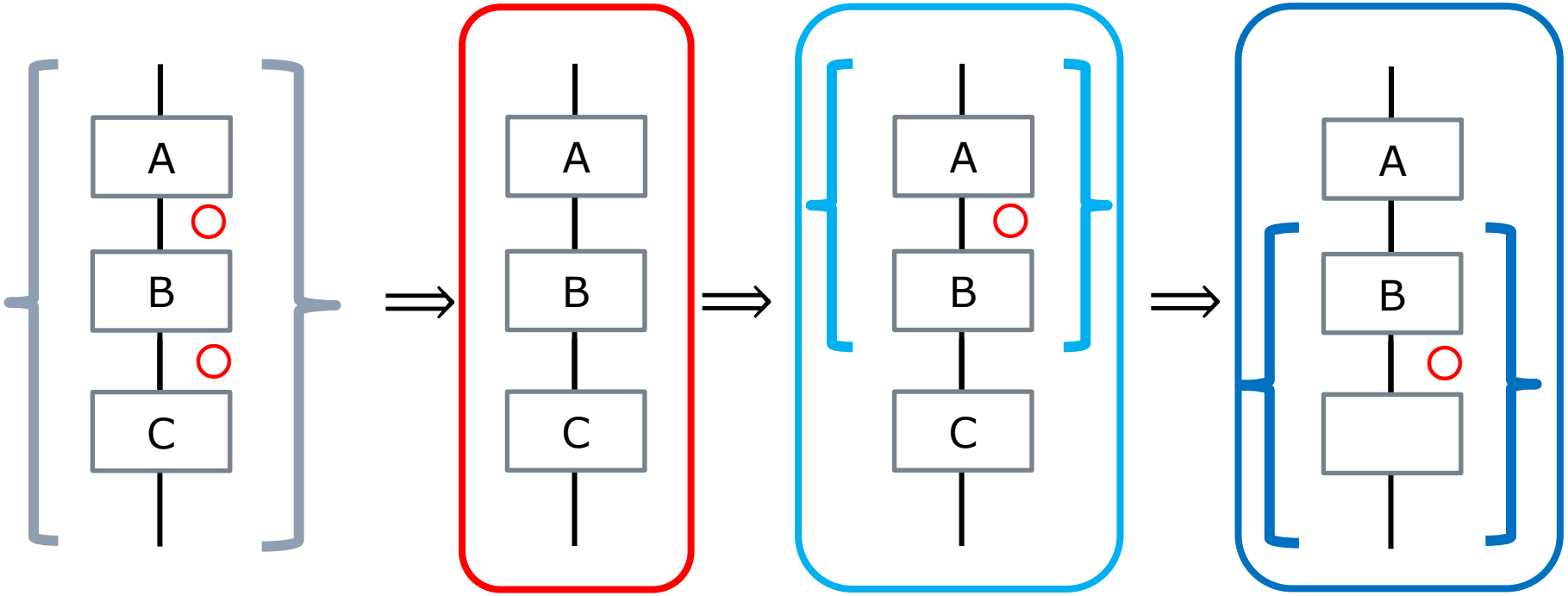


# 直列合成の結合律

$$A \circ B \circ C = (A \circ B) \circ C = A \circ (B \circ C)$$

# 直列合成の結合律

$A \circ B \circ C = (A \circ B) \circ C = A \circ (B \circ C)$



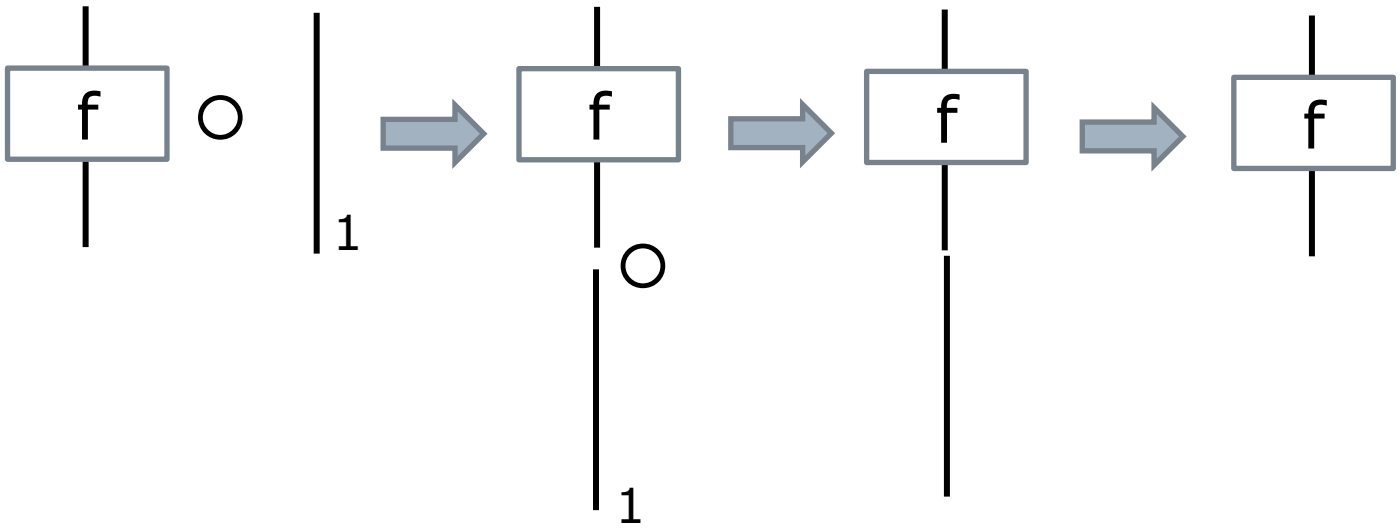
# 並列合成の単位元

- 何もしない空のスペース（下の破線のBox）を$1_I$とする。
  この時、次の式が成り立つ。


- $1_I$は、並列合成の単位元である。

# 直列合成の単位元

- Wire上にBoxを持たないWireを1とする。
  この時、次の式が成り立つことが、Diagramからわかる。




- この1 は、直列合成の単位元である。


50

# String Diagramを学ぶ

## 第二部　量子回路をDiagramで表す

# String Diagramを学ぶ
# 第二部　量子回路をDiagramで表す

Diagramと回路

量子ゲート

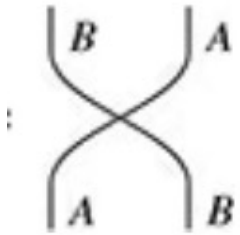
量子ゲートの行列表現

量子回路

# Diagramと回路

# Diagramと「回路」

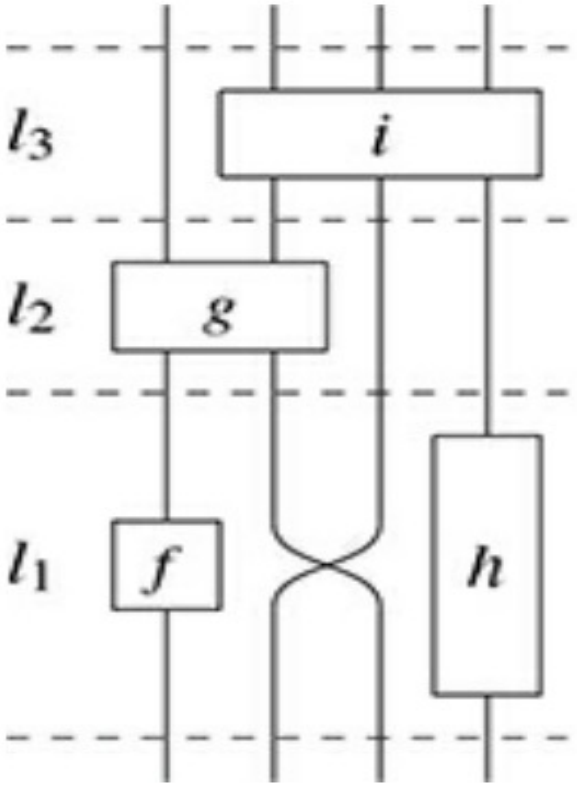
- 「並列合成」⊗ と「直列合成」 ○と、次の「スワップ」で




構成されるDiagramを「回路 circuit 」という。

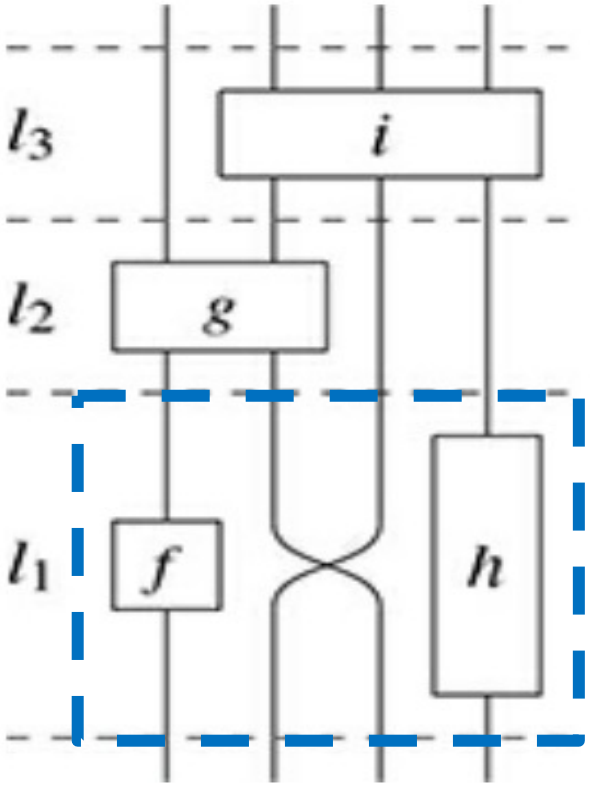
# 「回路 circuit 」の例

- 次のDiagram Dは「回路 circuit」である。（「並列合成」と「直列合成」と「スワップ」で構成される）



# 「回路 circuit 」の例

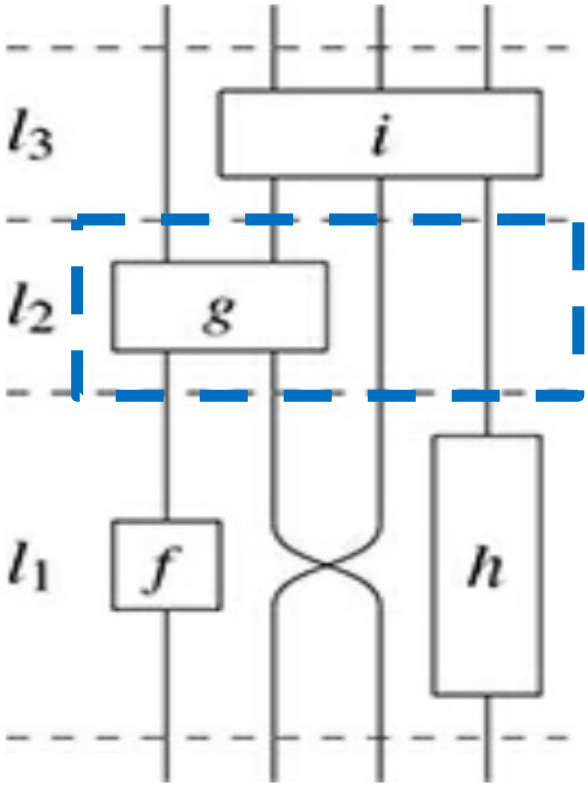
- 次のDiagram Dは「回路 circuit」である。（「並列合成」と「直列合成」と「スワップ」で構成される）




$l_1 = f \otimes swap \otimes h$

# 「回路 circuit 」の例

- 次のDiagram Dは「回路 circuit」である。（「並列合成」と「直列合成」と「スワップ」で構成される）

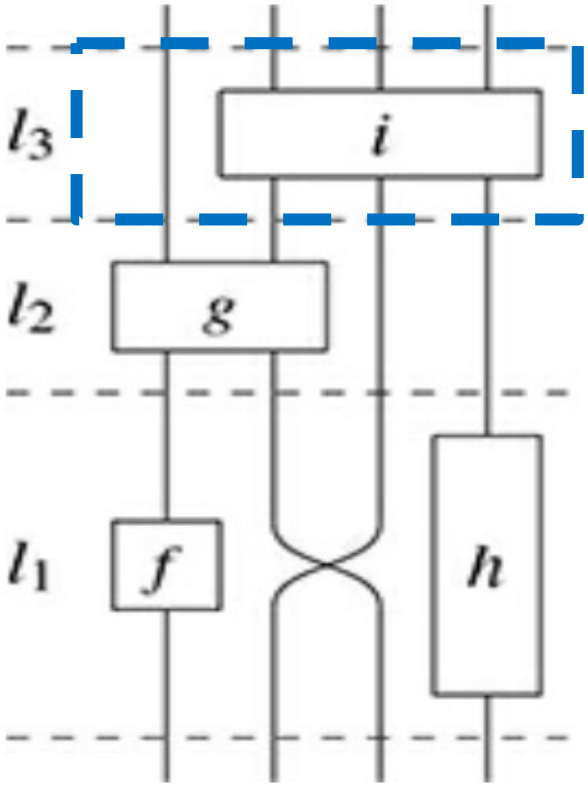



$l_2 = g \otimes 1 \otimes 1$

$l_1 = f \otimes swap \otimes h$

# 「回路 circuit 」の例

- 次のDiagram Dは「回路 circuit」である。（「並列合成」と「直列合成」と「スワップ」で構成される）




$l_3 = 1 \otimes i$

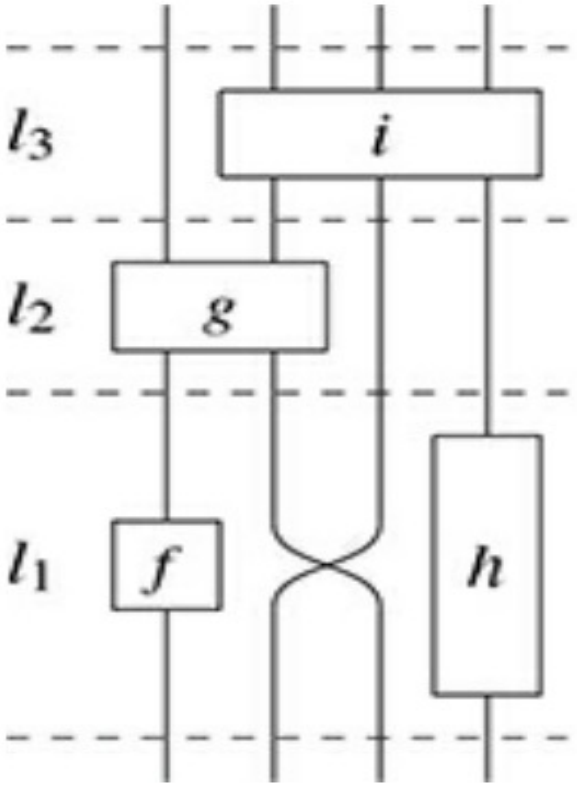
$l_2 = g \otimes 1 \otimes 1$

$l_1 = f \otimes swap \otimes h$

# 「回路 circuit 」の例

- 次のDiagram Dは「回路 circuit」である。（「並列合成」と「直列合成」と「スワップ」で構成される）

$D = l_3 \circ l_2 \circ l_1$



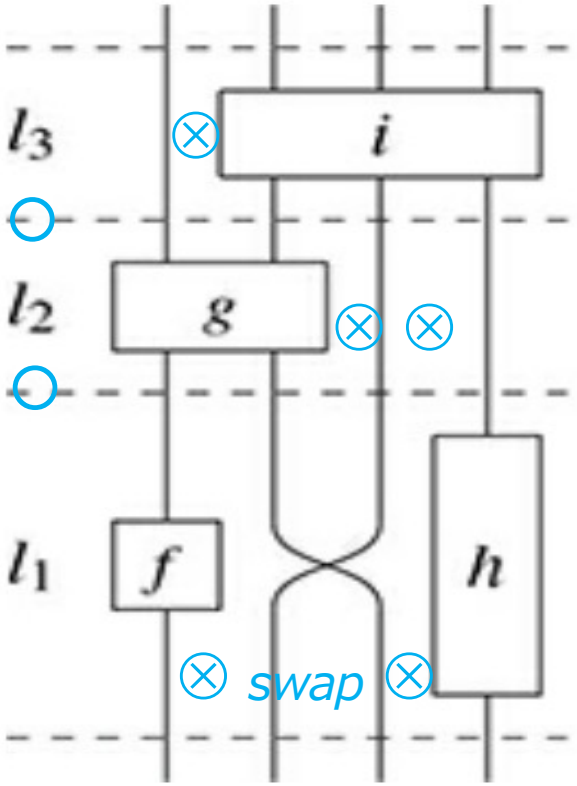

$l_3 = 1 \otimes i$

$l_2 = g \otimes 1 \otimes 1$

$l_1 = f \otimes swap \otimes h$

# 「回路 circuit 」の例

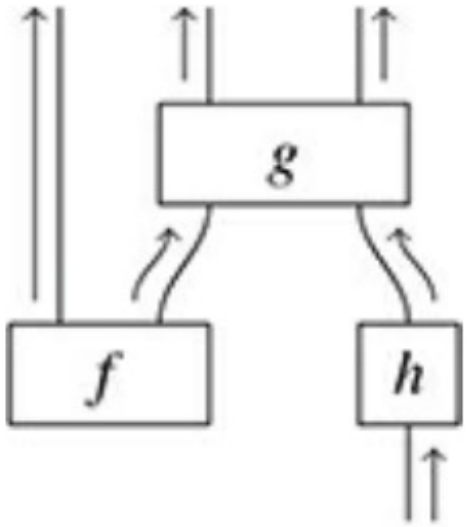
- 次のDiagram Dは「回路 circuit」である。（「並列合成」と「直列合成」と「スワップ」で構成される）



# 「回路」の特徴

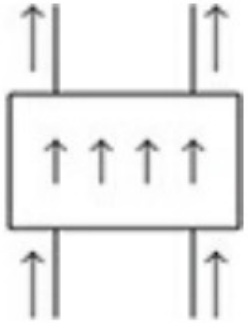
# 向きを持つ経路

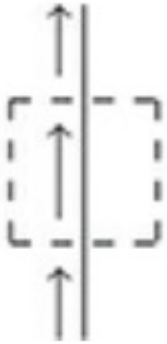
- Diagram上の経路は、向きを持つ。

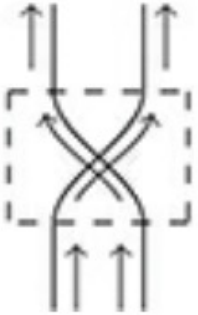


# 向きを持つ経路

- Diagram上の経路は、向きを持つ。
  このことは、DiagramのBoxの内部でも、入力から出力に向かう流れは、同じ向きを持つということを想定している。

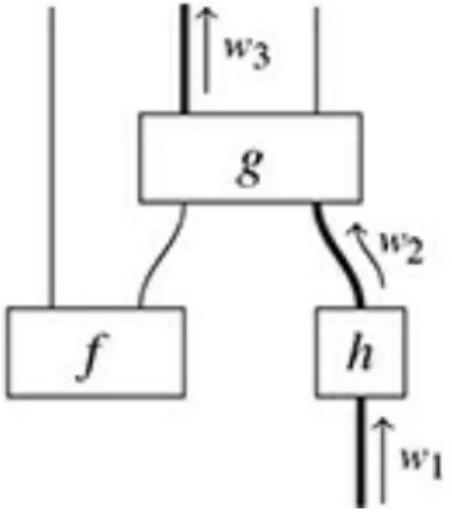






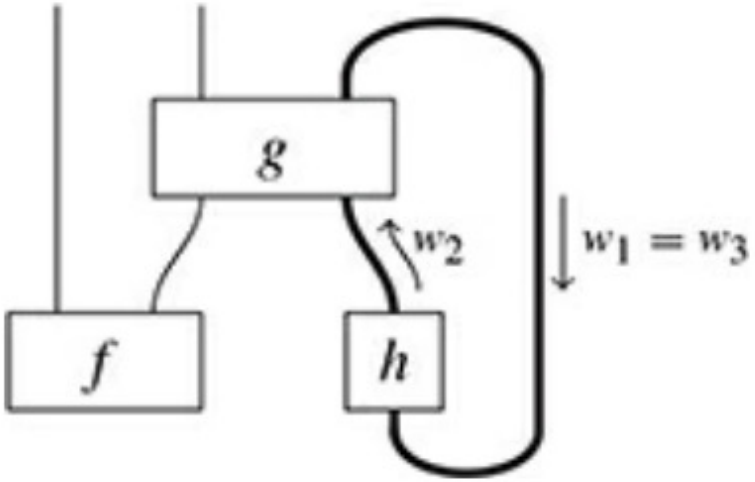
# 向きを持つ経路

- Wireの向きを持つ経路というのは、Diagram上のwireのリスト(w1, w2, ..., wn) で、i < n なる全ての i について $w_i$ はあるBoxの入力で、$w_{i+1}$ はそのBoxの出力になっているものである。



# 向きを持つ経路のサイクル

- 向きを持つ「サイクル」というのは、同じBoxから出て同じBoxに帰ってくる向きを持つ経路である。



# 回路の特徴

- 「並列合成」⊗ と「直列合成」○と「スワップ」で構成される「回路 circuit 」のDiagramは、「サイクル」を含まない。

# 量子ゲート回路

- 「並列合成」⊗ と「直列合成」○と「スワップ」で構成される「回路 circuit 」のDiagramは、「サイクル」を含まない。

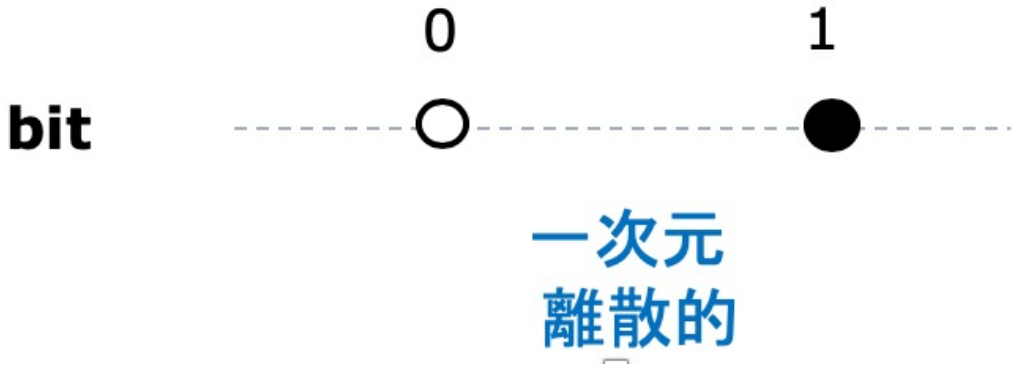
- 次に見るように、量子ゲート回路も基本的には「並列合成」⊗ と「直列合成」○で構成される。
  量子ゲート回路は、「サイクル」を含まない。

# 量子ゲート

# 古典bitと量子bit

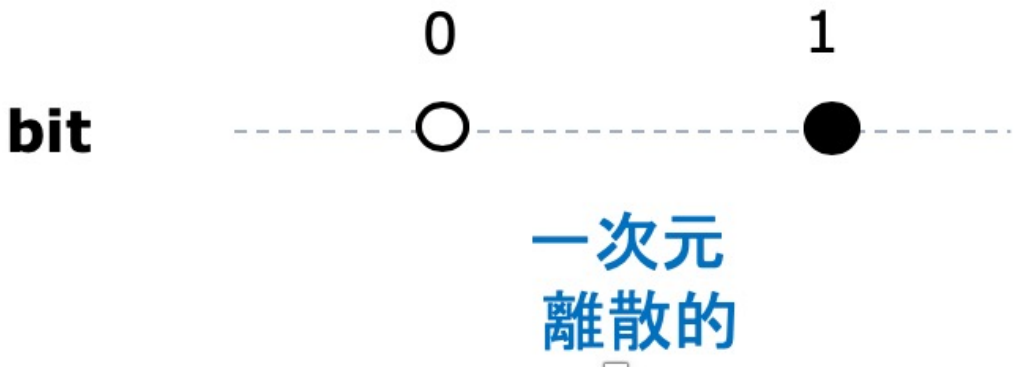
# 一次元の点としての古典bit

- 古典bitは、0 または 1 の値しかとらない二つの離散的な点として表現される。



# 一次元の点としての古典bit
# 二次元のベクトルとしての量子bit

- 古典bitは、0 または 1 の値しかとらない二つの離散的な点として表現される。




- 量子bit qubitは、成分を二つ持つ、二次元の複素ベクトルとして表現される。

# 一次元の点としての古典bit
# 二次元のベクトルとしての量子bit

- 量子bit qubitは、$\begin{pmatrix}1\\0\end{pmatrix}$と$\begin{pmatrix}0\\1\end{pmatrix}$という、直交する二つのベクトルによって張られ、aとbという二つの成分によって決定される、二次元のベクトル$\begin{pmatrix}a\\b\end{pmatrix}$として表現される。
  a, b は、複素数の値を取る。

# 一次元の点としての古典bit
# 二次元のベクトルとしての量子bit

- 量子bit qubitは、$\begin{pmatrix}1\\0\end{pmatrix}$と$\begin{pmatrix}0\\1\end{pmatrix}$という、直交する二つのベクトルによって張られ、aとbという二つの成分によって決定される、二次元のベクトル$\begin{pmatrix}a\\b\end{pmatrix}$として表現される。
  a, b は、複素数の値を取る。

$$\textbf{Qubit} = \begin{pmatrix}a\\b\end{pmatrix} = a\begin{pmatrix}1\\0\end{pmatrix} + b\begin{pmatrix}0\\1\end{pmatrix}$$

# 一次元の点としての古典bit
# 二次元のベクトルとしての量子bit

- 量子bit qubitは、$\begin{pmatrix}1\\0\end{pmatrix}$と$\begin{pmatrix}0\\1\end{pmatrix}$という、直交する二つのベクトルによって張られ、aとbという二つの成分によって決定される、二次元のベクトル$\begin{pmatrix}a\\b\end{pmatrix}$として表現される。
  a, b は、複素数の値を取る。

$$\textbf{Qubit} = \begin{pmatrix}a\\b\end{pmatrix} = a\begin{pmatrix}1\\0\end{pmatrix} + b\begin{pmatrix}0\\1\end{pmatrix}$$

この時、$|a|^2+|b|^2=1$ という条件がつく。

# 一次元の点としての古典bit
# 二次元のベクトルとしての量子bit

- 量子bit qubitは、$\begin{pmatrix}1\\0\end{pmatrix}$と$\begin{pmatrix}0\\1\end{pmatrix}$という、直交する二つのベクトルによって張られ、aとbという二つの成分によって決定される、二次元のベクトル$\begin{pmatrix}a\\b\end{pmatrix}$として表現される。
  a, b は、複素数の値を取る。

$$\textbf{Qubit} = \begin{pmatrix}a\\b\end{pmatrix} = a\begin{pmatrix}1\\0\end{pmatrix} + b\begin{pmatrix}0\\1\end{pmatrix}$$

この時、$|a|^2+|b|^2=1$ という条件がつく。

- ベクトル$\begin{pmatrix}1\\0\end{pmatrix}$を$|0\rangle$, $\begin{pmatrix}0\\1\end{pmatrix}$を$|1\rangle$と表す。

# 一次元の点としての古典bit
# 二次元のベクトルとしての量子bit

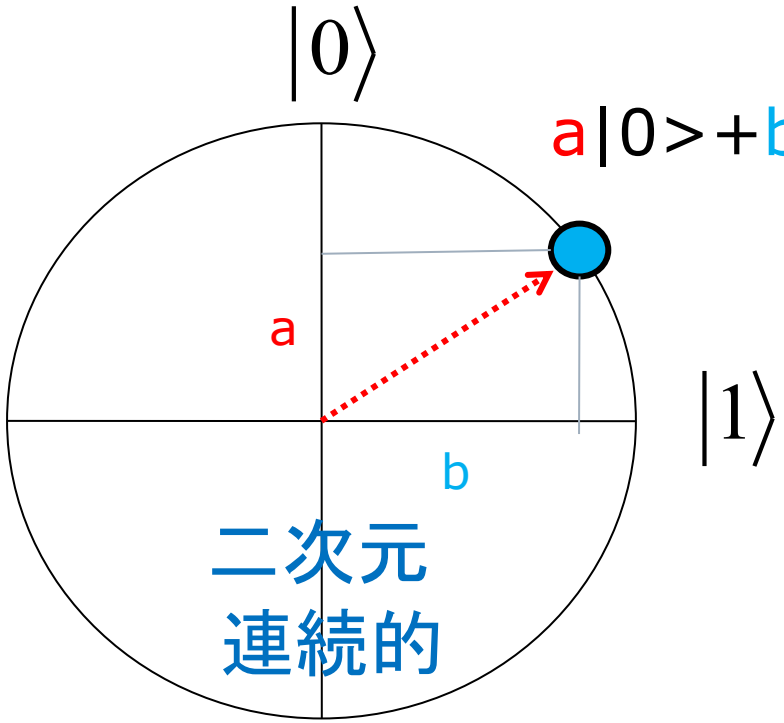
- 量子bit qubitは、$\begin{pmatrix}1\\0\end{pmatrix}$=|0>と$\begin{pmatrix}0\\1\end{pmatrix}$=|1> という、直交する二つのベクトルによって張られ、aとbという二つの成分によって決定される、二次元のベクトル$\begin{pmatrix}a\\b\end{pmatrix}$として表現される。a, b は、複素数の値を取る。

$$\textbf{Qubit} = \begin{pmatrix}a\\b\end{pmatrix} = a\begin{pmatrix}1\\0\end{pmatrix} + b\begin{pmatrix}0\\1\end{pmatrix} = a|0> + b|1>$$

この時、$|a|^2+|b|^2=1$ という条件がつく。

二つのビットは、異なる性質を持つ

**qubit**




qubitは、|0>と|1>の「重ね合わせ」で、aとbの二つの数で決まる

$$\begin{pmatrix} a \\ b \end{pmatrix}$$

|0>の成分

|1>の成分

という二次元ベクトルである。

$|a|^2+|b|^2=1$

である。

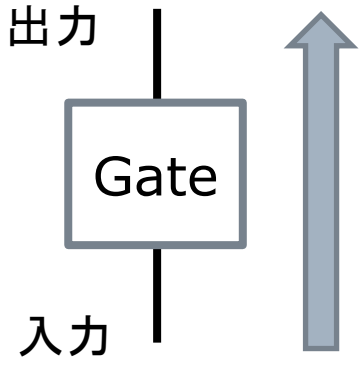
**bit**


一次元
離散的

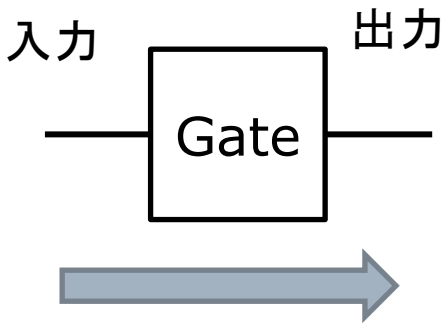
# 古典ゲートと量子ゲート

この節の回路の記法は、これまで見てきたDiagramの記法と流儀が異なっていることに注意。
Diagramでは、入力が下で出力が上で、時間は、下から上に流れるのだが、
量子回路の通常の記法は、入力が左で出力が右で、時間は左から右に流れる。

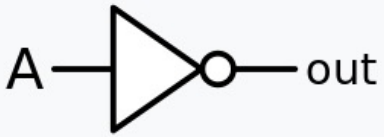
Diagramの記法




通常の量子回路の記法



# 古典論理ゲートの例

| 論理 | 論理式 | 回路記号（MIL記号） |
| --- | --- | --- |
| NOT | $\overline{A}$ |   |
| OR | $A + B$ |   |
| AND | $A \cdot B$ |   |
| XOR | $A \oplus B$ |   |

# 代表的な1-qubitの量子ゲート X, Z, H

入力 → 出力


## Bit Flipper

$a|0\rangle+b|1\rangle \rightarrow b|0\rangle+a|1\rangle$


## Phase Flipper

$a|0\rangle+b|1\rangle \rightarrow a|0\rangle-b|1\rangle$

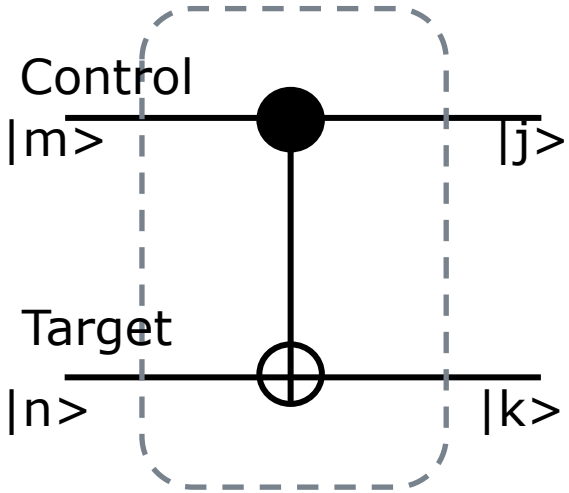

## Hadamard

$a|0\rangle+b|1\rangle \rightarrow$

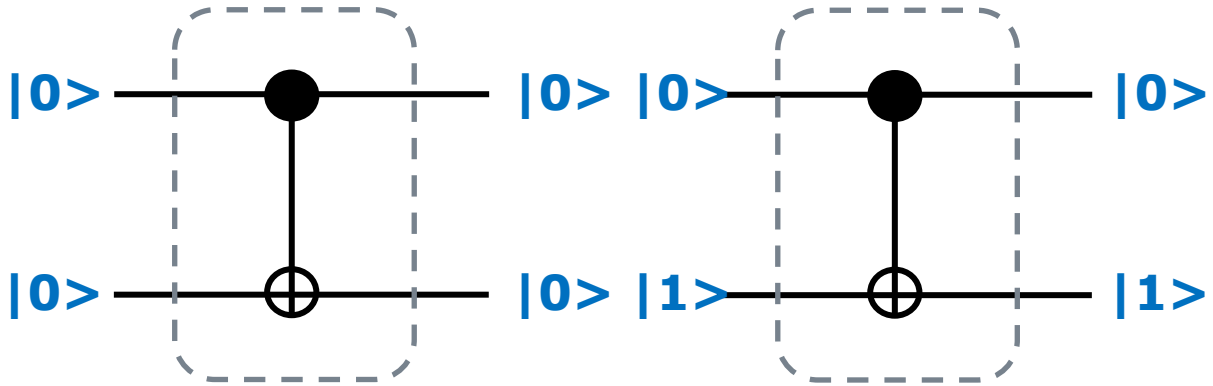
$\frac{1}{\sqrt{2}}(a+b)|0\rangle+\frac{1}{\sqrt{2}}(a-b)|1\rangle$

# 2-qubitのゲート
# CNOT (Control-NOT)

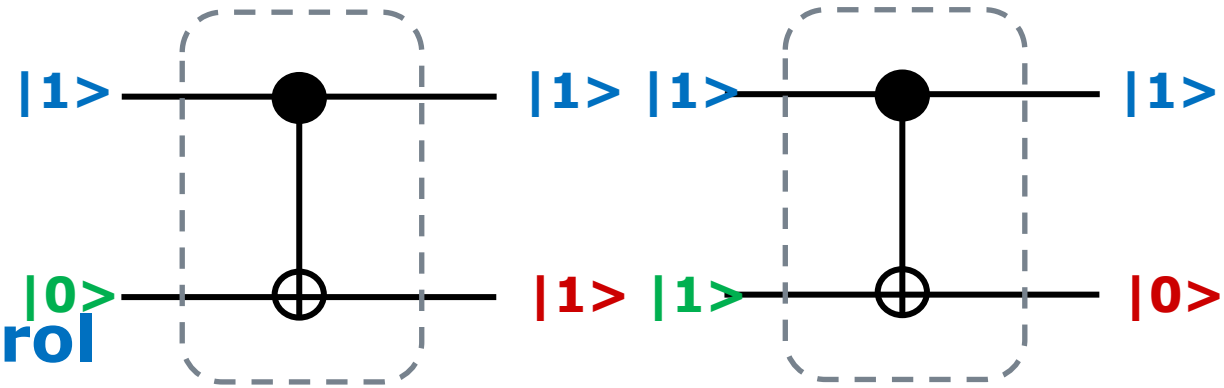
Control が |1> の時
Target のNOTをとる




Controlが |0>なら何もしない




Controlが |1>ならNOT操作




**|mn>の**
**第一bit |m>がControl**
**第二bit |n>がTarget**

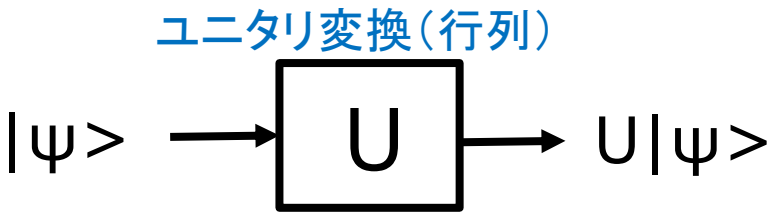
# 古典ゲートと量子ゲート

- 古典的なゲートは、古典ビットの入力を古典ビットの出力に変換する。量子ゲートは、量子ビット qubitの入力を、qubitの出力に変換する。
- 古典的なゲートは、基本的には、AND, OR, NOTといった「論理的」な演算で定義されるが、量子的なゲートは数学的には、「ユニタリ変換」として定義される。「ユニタリ変換」は、「ユニタリ行列」で定義される。
- 「ユニタリ行列」は、入力のqubitの「入力ベクトル」を出力のqubitの「出力ベクトル」に変換する。
- 一つの量子ゲートには、一つの「ユニタリ行列」が対応する。
- 古典的なゲートとは異なって、量子ゲートでは入力のqubitの数と出力のqubitの数は等しい。

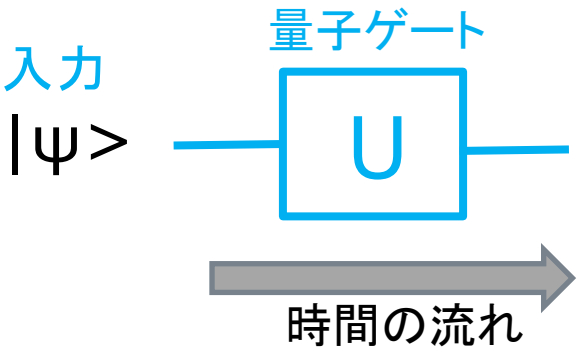
# 量子ゲートの行列表現

# 量子ゲートとユニタリ行列

- 量子の状態 $|\psi\rangle$は、ユニタリ変換U（ユニタリ行列）の作用を受けて、状態 $U|\psi\rangle$ に変化する。
- この変化 $|\psi\rangle \rightarrow U|\psi\rangle$を、次のように表そう。

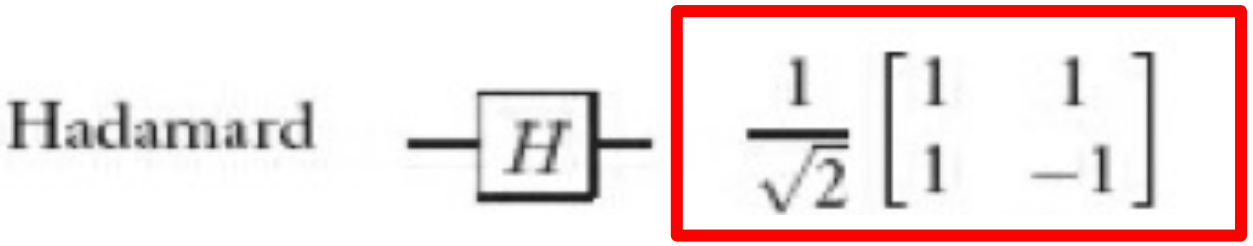



- この時、Uを、$|\psi\rangle$ を入力、$U|\psi\rangle$ を出力とする回路と考えることができる。これを、「量子ゲート」と呼ぶ。

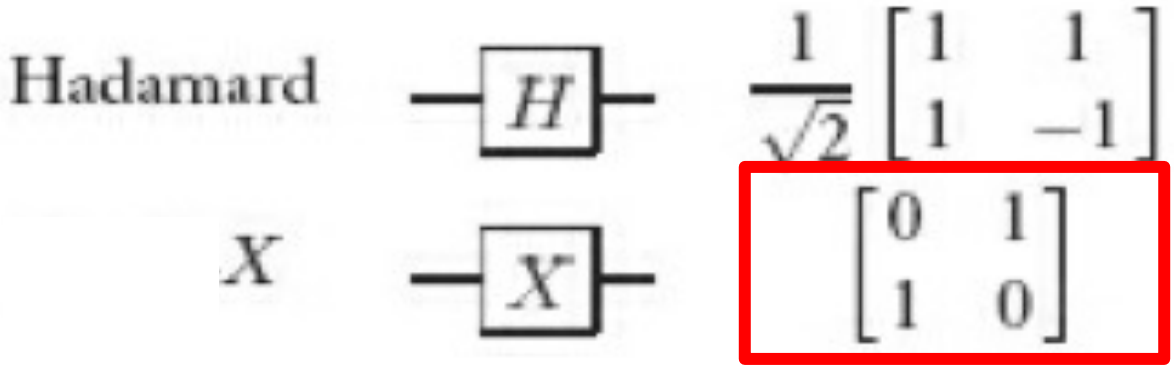



量子ゲートは
ユニタリ行列と
一対一に対応する

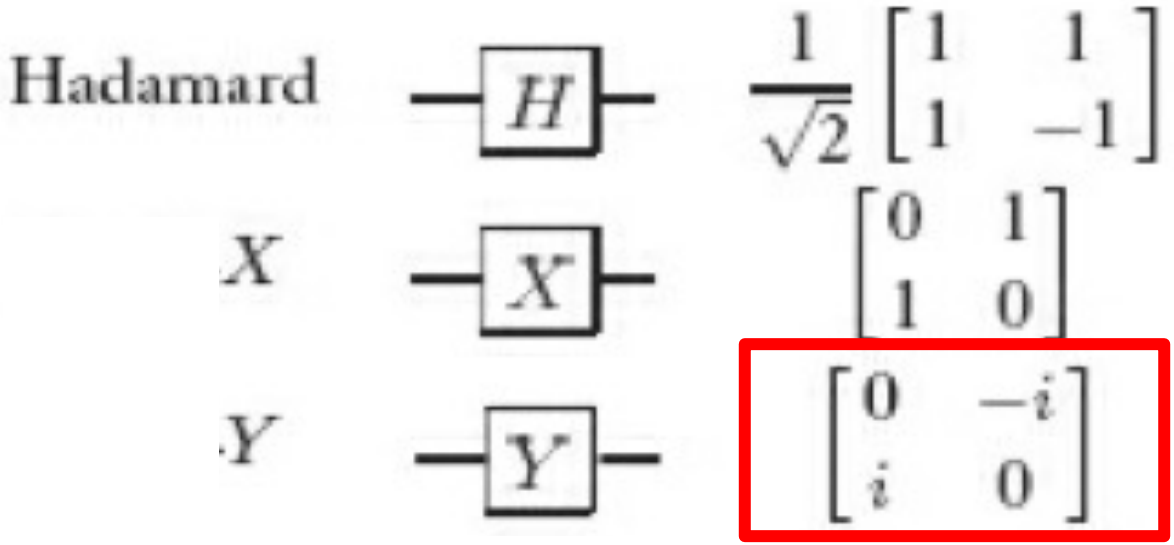
# 1-qubitの量子ゲートの例と
# その行列表示



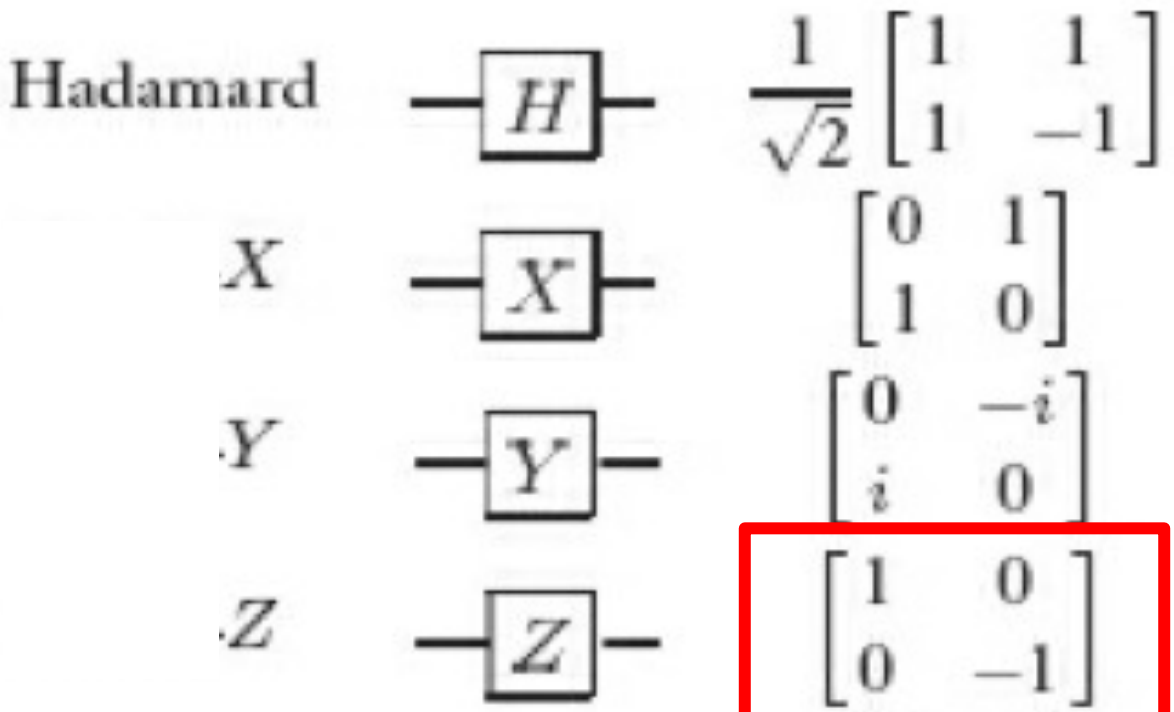

1-qubitの量子ゲートの例とその行列表示



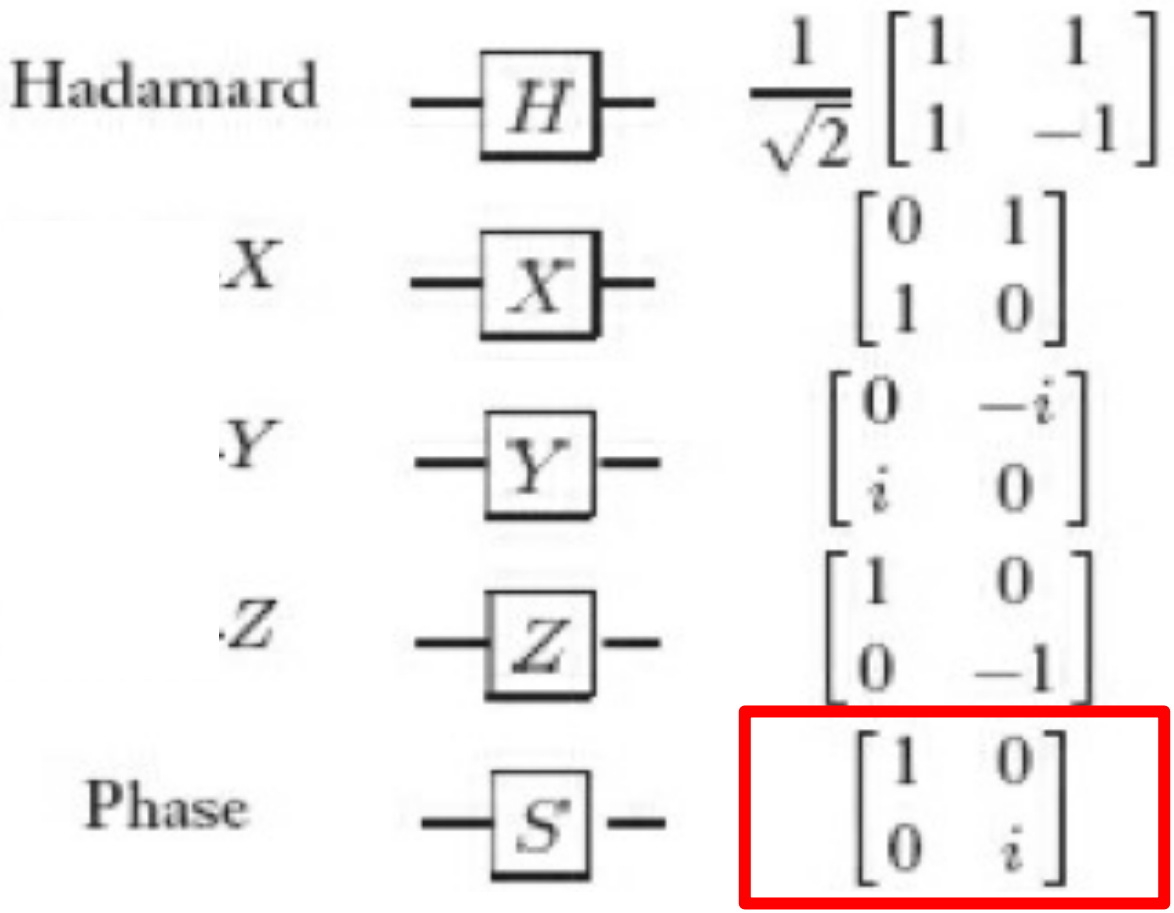

1-qubitの量子ゲートの例とその行列表示



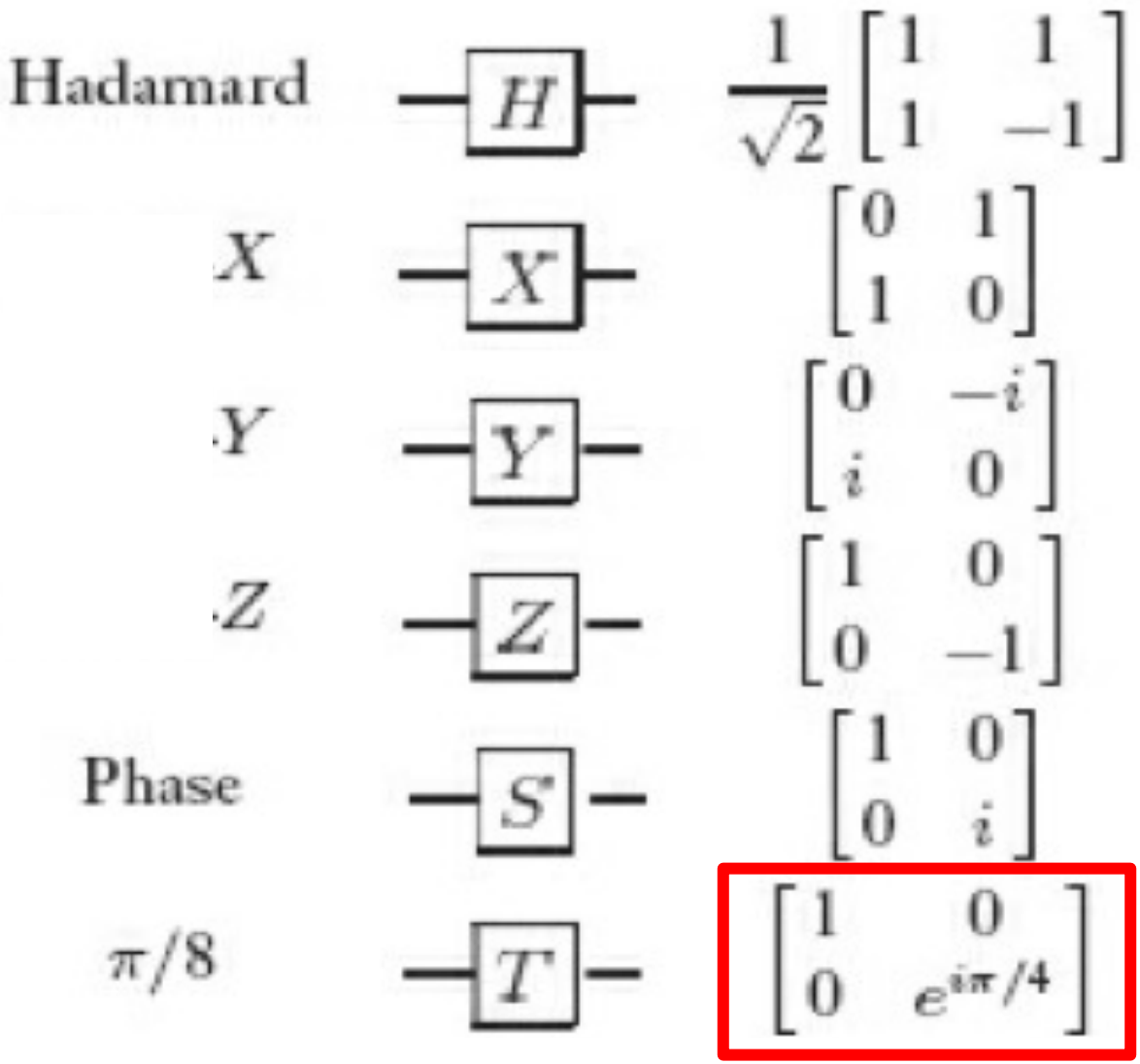

1-qubitの量子ゲートの例とその行列表示




1-qubitの量子ゲートの例とその行列表示




1-qubitの量子ゲートの例とその行列表示




1-qubitの量子ゲートの例とその行列表示

# 量子回路の行列表現

# 量子ゲートと量子回路の行列表現

- 量子ゲートから、「直列合成」と「並列合成」で構成されたDiagramを「量子回路」と呼ぶ。
- 量子ゲートは、ユニタリ行列で表現されるのだが、量子回路も、次のルールで対応する行列表現を持つ。
- 二つの量子ゲートから「直列合成」された量子回路に対応する行列は、二つの量子ゲートに対応するユニタリ行列の「行列の積」である。
- 二つの量子ゲートから「並列合成」された量子回路に対応する行列は、二つの量子ゲートに対応するユニタリ行列の「テンソル積」である。

# 量子回路の行列表現

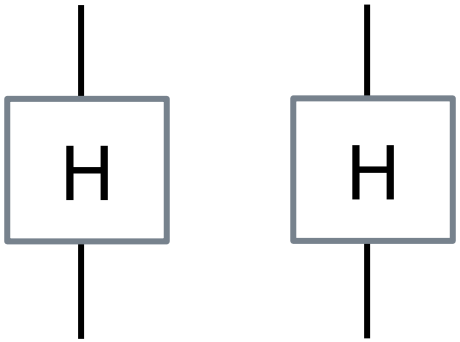
- 「直列合成」された量子回路　：「行列の積」

- 「並列合成」された量子回路　：「行列のテンソル積」

# 量子回路の行列表現の例

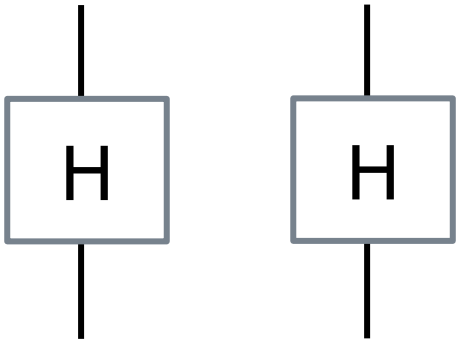
# 量子回路の行列表示（例1）

- Hが量子ゲートの時、次のDiagram Dは量子回路である。



# 量子回路の行列表示

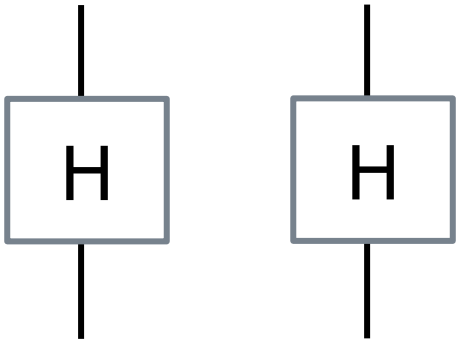
- Hが量子ゲートの時、次のDiagram Dは量子回路である。




$$D = H \otimes H$$

# 量子回路の行列表示

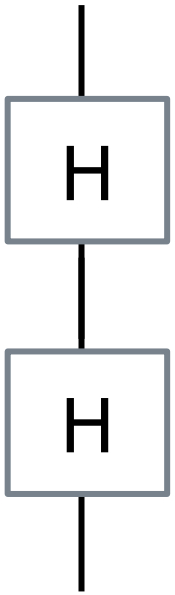
- 量子回路Dは、次の行列表示を持つ。




$$D = H \otimes H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

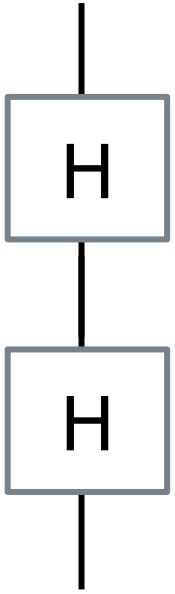
# 量子回路の行列表示（例2）

- Hが量子ゲートの時、次のDiagram Dは量子回路である。



# 量子回路の行列表示

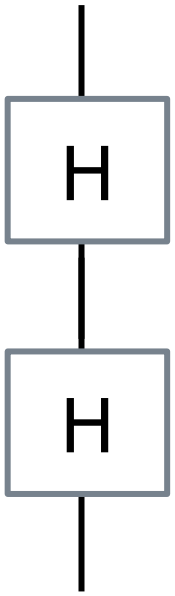
- Hが量子ゲートの時、次のDiagram Dは量子回路である。




$$D = H \circ H$$

# 量子回路の行列表示

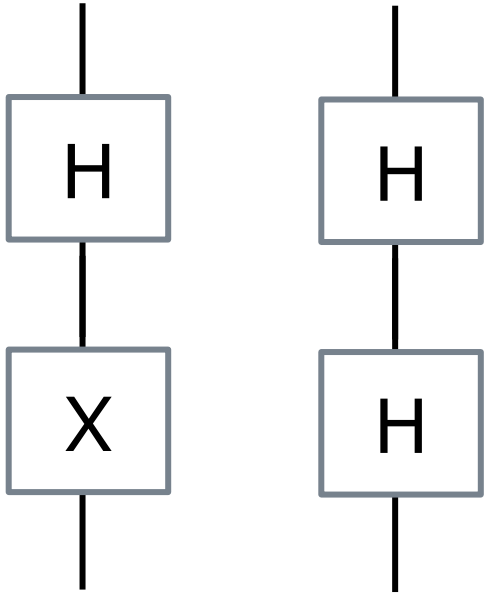
- 量子回路Dは、次の行列表示を持つ。




$$D = H \circ H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

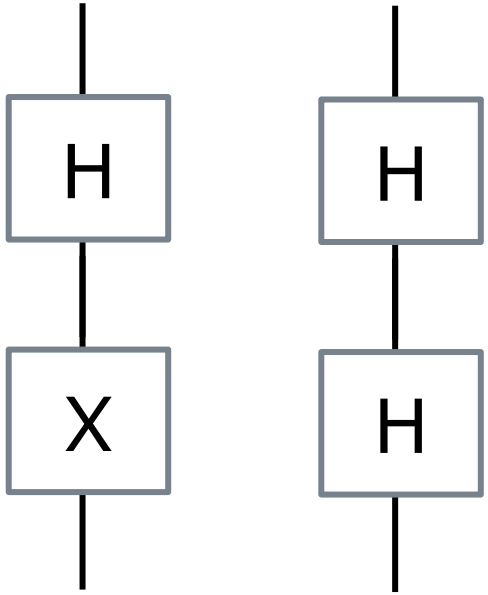
# 量子回路の行列表示（例3）

- H, Xが量子ゲートの時、次のDiagram Dは量子回路である。



# 量子回路の行列表示

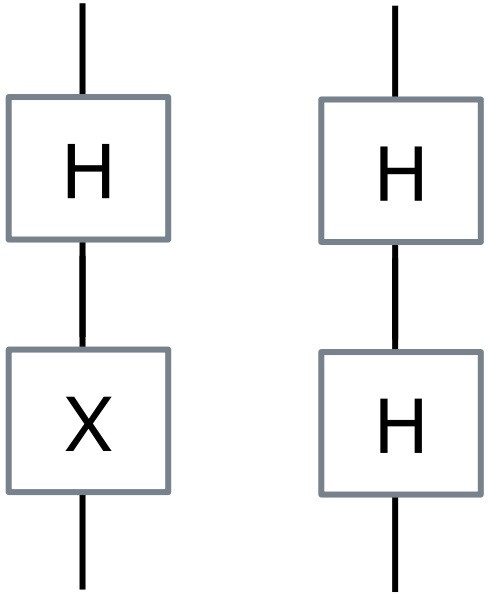
- H, Xが量子ゲートの時、次のDiagram Dは量子回路である。




$$D = (H \circ X) \otimes (H \circ H)$$

# 量子回路の行列表示

- 量子回路Dは、次の行列表示を持つ。




$$D = (H \circ X) \otimes (H \circ H)$$

$$= \left( \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right) \otimes \left( \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right)$$

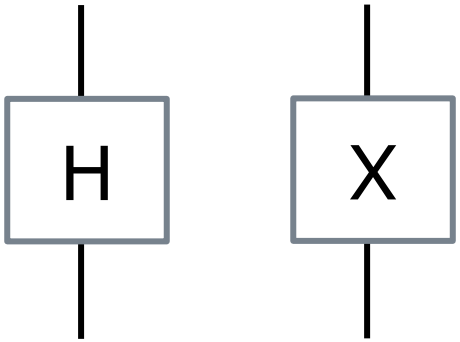
# 量子回路

# 量子回路

- Boxが「量子ゲート」を表し、Wireが「q-bitの状態」を表すDiagramを考える。
- このDiagramが、「並列合成」と「直列合成」で構成されている時、このDiagramを「量子回路」と呼ぶ。
- 量子回路は、ループを含まない。

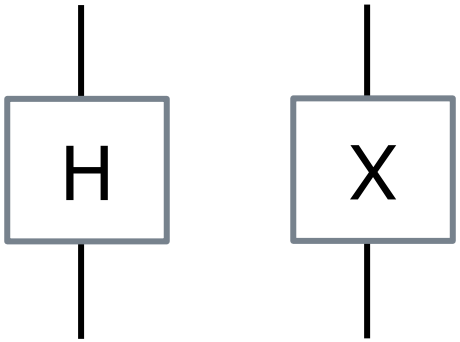
# 量子回路の例

- H, Xが量子ゲートの時、次のDiagram Dは量子回路である。



# 量子回路の例

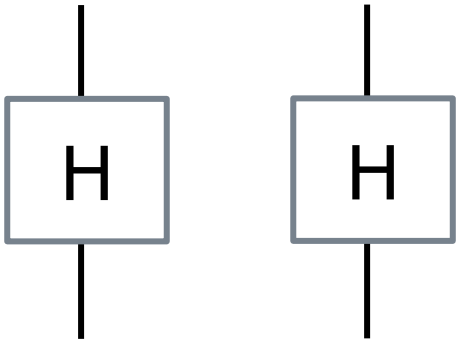
- H, Xが量子ゲートの時、次のDiagram Dは量子回路である。




$$D = H \otimes X$$

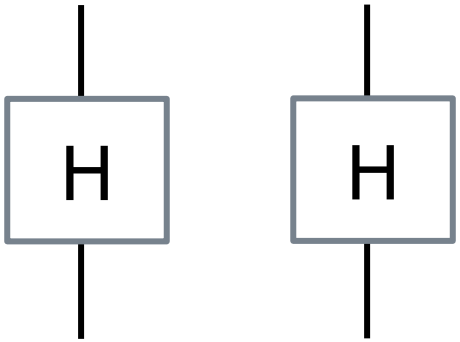
# 量子回路の例

- Hが量子ゲートの時、次のDiagram Dは量子回路である。



# 量子回路の例

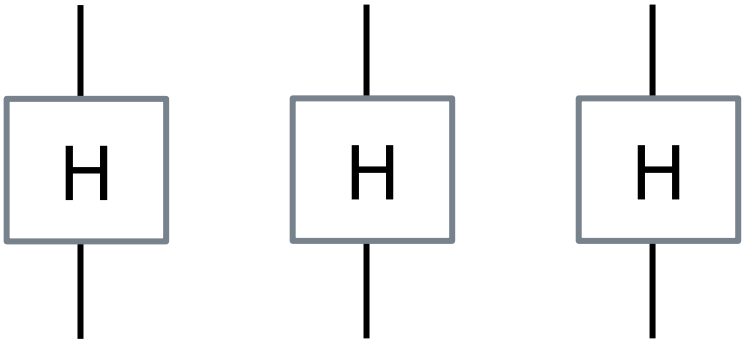
- Hが量子ゲートの時、次のDiagram Dは量子回路である。




$$D = H \otimes H$$

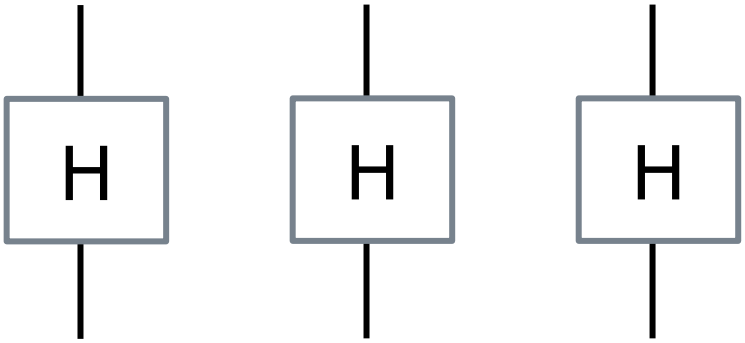
# 量子回路の例

- Hが量子ゲートの時、次のDiagram Dは量子回路である。



# 量子回路の例

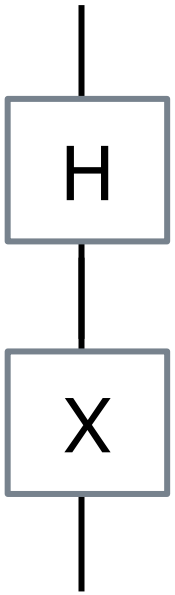
- Hが量子ゲートの時、次のDiagram Dは量子回路である。




$$D = H \otimes H \otimes H$$

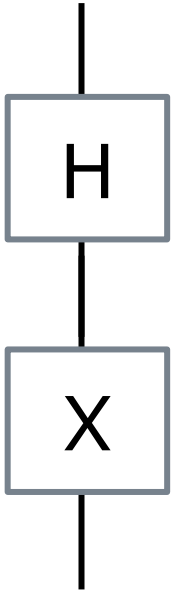
# 量子回路の例

- H, Xが量子ゲートの時、次のDiagram Dは量子回路である。



# 量子回路の例

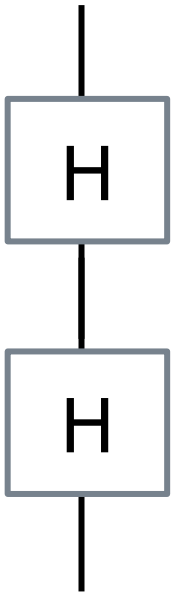
- H, Xが量子ゲートの時、次のDiagram Dは量子回路である。




$$D = H \circ X$$

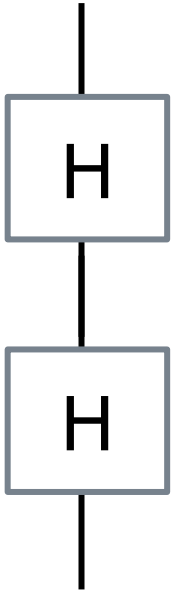
# 量子回路の例

- Hが量子ゲートの時、次のDiagram Dは量子回路である。



# 量子回路の例

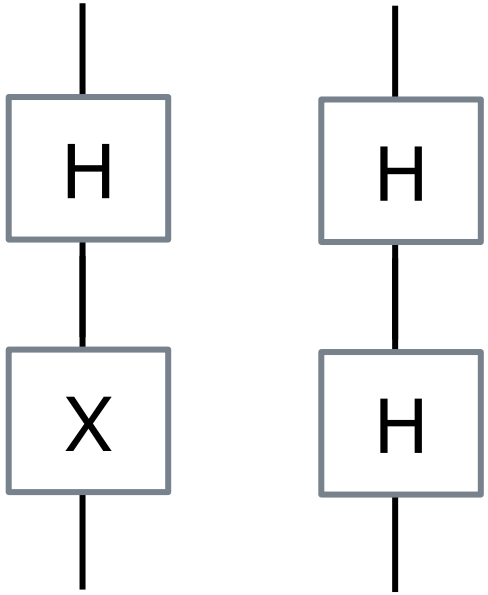
- Hが量子ゲートの時、次のDiagram Dは量子回路である。




$$D = H \circ H$$

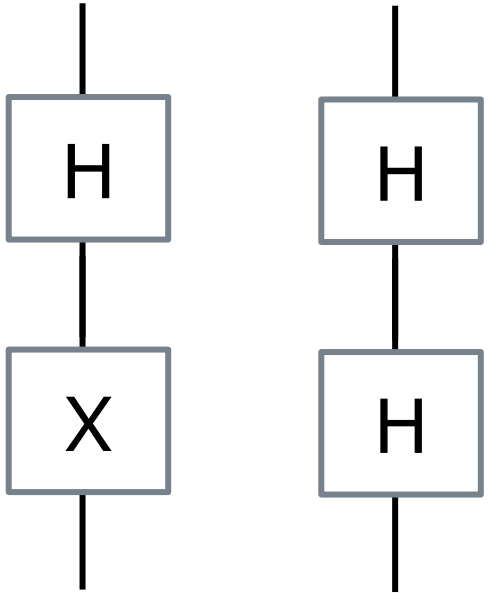
# 量子回路の例

- H, Xが量子ゲートの時、次のDiagram Dは量子回路である。



# 量子回路の例

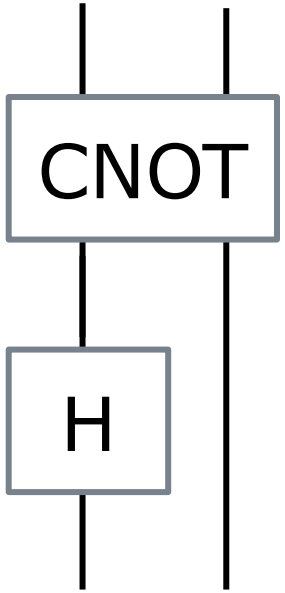
- H, Xが量子ゲートの時、次のDiagram Dは量子回路である。




$$D = (H \circ X) \otimes (H \circ H)$$

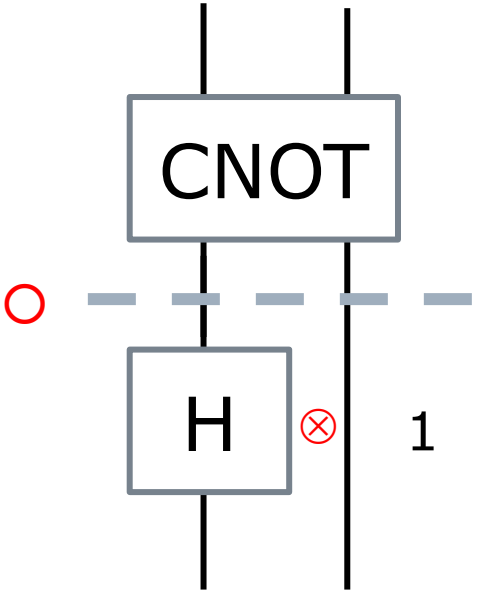
# 量子回路の例

- H, CNOTが量子ゲートの時、次のDiagram Dは量子回路である。



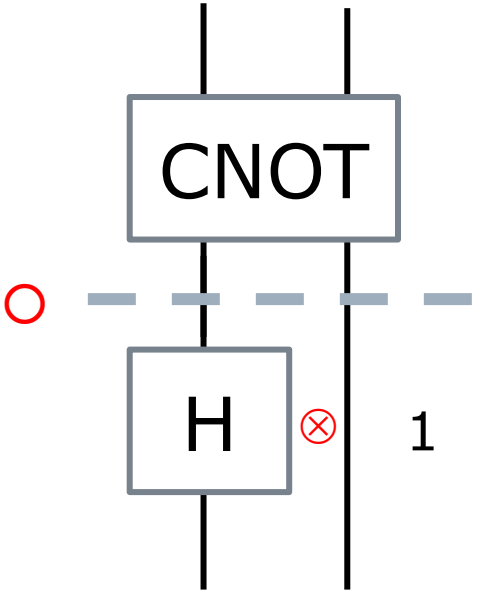
# 量子回路の例

- H, CNOTが量子ゲートの時、次のDiagram Dは量子回路である。



# 量子回路の例

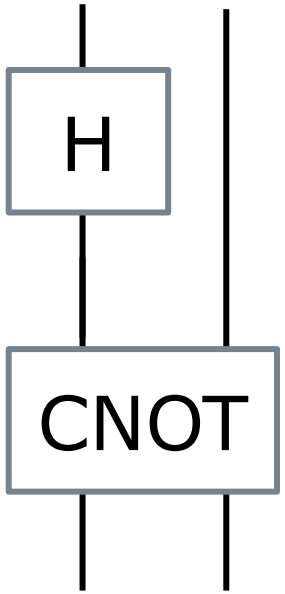
- H, CNOTが量子ゲートの時、次のDiagram Dは量子回路である。




$$D = \mathrm{CNOT} \circ (H \otimes 1)$$

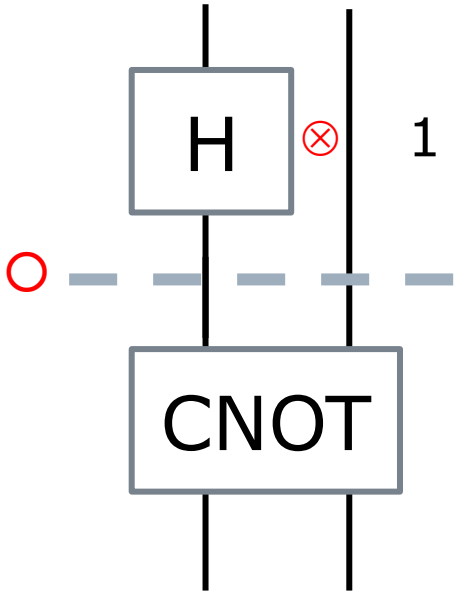
# 量子回路の例

- H, CNOTが量子ゲートの時、次のDiagram Dは量子回路である。



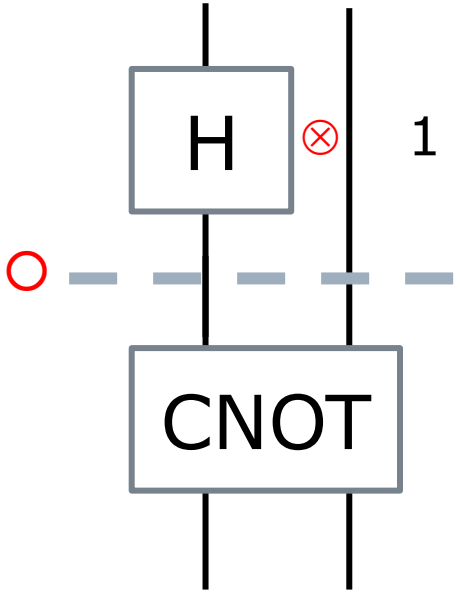
# 量子回路の例

- H, CNOTが量子ゲートの時、次のDiagram Dは量子回路である。



# 量子回路の例

- H, CNOTが量子ゲートの時、次のDiagram Dは量子回路である。




$$D = (H \otimes 1) \circ CNOT$$


50

# String Diagramを学ぶ

## 第三部　プロセスの理論

# String Diagramを学ぶ
# 第三部　プロセスの理論

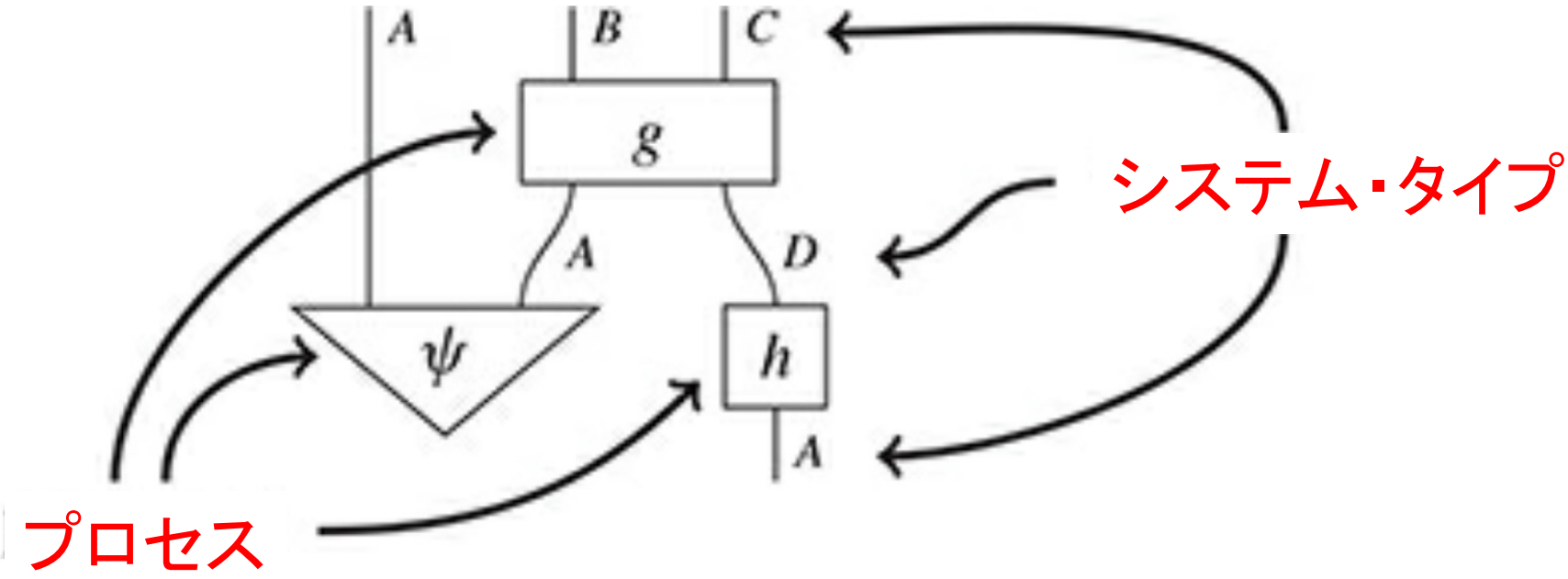
基本的な型としての「集合」
プロセスとしての「関数」
プロセスとしての「関係」

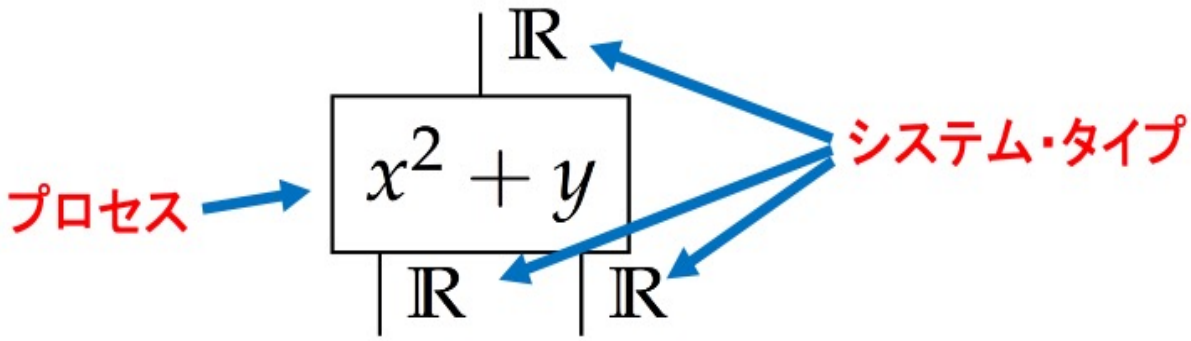
プロセス中心のアプローチ
特別な「プロセス」
「状態」、「効果」、「数」
ket記法との対応

# 基本的な型としての「集合」

# Diagram（ダイアグラム）

ダイアグラムは「箱」と「ワイアー」からできている。
「箱」は「プロセス」を、「ワイアー」は「システム・タイプ」を
表している。






# 代表的なシステム・タイプ

- ℕ: 自然数
- ℝ: 実数
- ℂ: 複素数
- 𝔹 : ブール値の集合

これらは全て集合である

# 二つのシステムが結合したシステムの型

集合の直積

$$A \otimes B := A \times B = \{(a, b) \mid a \in A, b \in B\}$$

tuple

$$A_1 \times \cdots \times A_n := \{(a_1, \ldots, a_n) \mid a_i \in A_i\}$$

# 二つのシステムが結合したシステムの型

## 集合の直積

$$A \otimes B := A \times B = \{(a, b) \mid a \in A, b \in B\}$$

tuple

$$A_1 \times \cdots \times A_n := \{(a_1, \ldots, a_n) \mid a_i \in A_i\}$$

## 集合の直積の結合律

$$((a, b), c) = (a, b, c) = (a, (b, c))$$

$$A \times (B \times C) = (A \times B) \times C$$

# 長さ n のビット列の集合

長さ n のビット列の集合は、それらを二進数の表現とみなせば、
0から $2^n-1$までの自然数の集合とみなすことができる。
例えば、n = 4 の場合なら、

(0, 0, 0, 0) <-> 0
(0, 0, 0, 1) <-> 1
(0, 0, 1, 0) <-> 2
.....
(1, 1, 1, 1) <-> 15

# 自明な型 I

自明な型 I は、ただ一つの要素だけを含む集合 {*}として定義される。

$$(a, *) = a = (*, a)$$

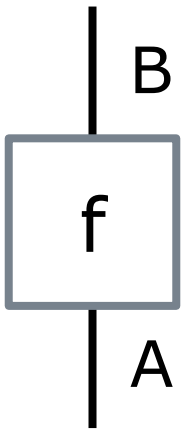
$$A \times \{*\} = A = \{*\} \times A$$

自明な型 I は、直積の単位元である。

# プロセスとしての「関数」

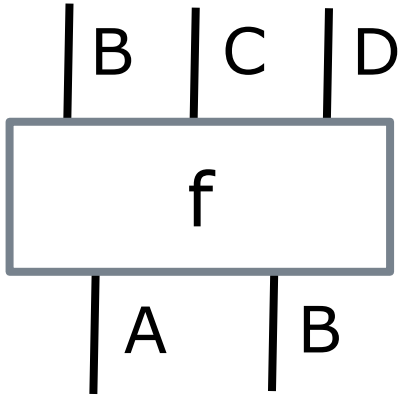
# Diagram での一変数の関数の表現

Diagramでは、集合 A から集合Bへの 一変数の関数 fは、次のようなプロセスとして表現される。



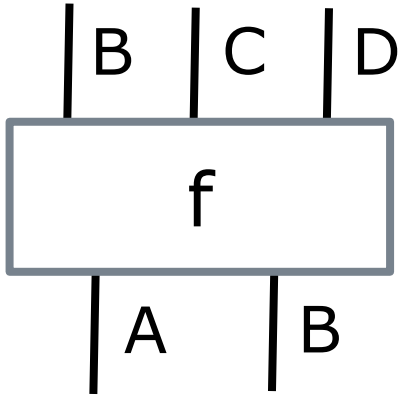
# Diagram での多変数の関数の表現

Diagramで、次のように表現される関数 fを考える。



# Diagram での多変数の関数の表現

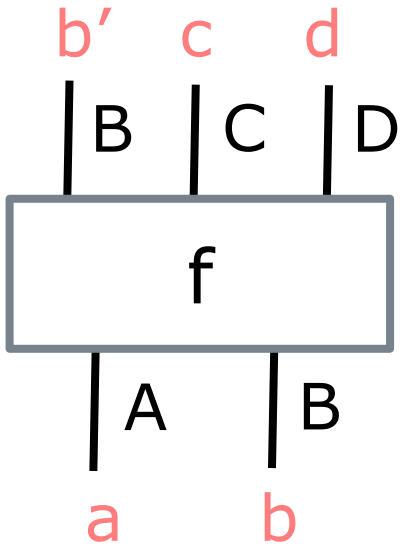
Diagramで、次のように表現される関数 fを考える。




集合の直積を使って表現すれば、
f は、集合 A × B から集合 B × C × D への関数である。

# Diagram での多変数の関数の表現

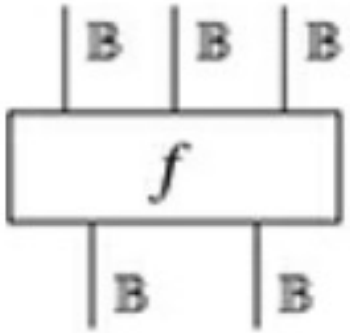
関数 f の入力、出力に具体的な値を入れてみる。




f は、集合 A × B から集合 B × C × D への関数である。
だから、(a,b) ∈ A × B で、(b′,c, d) ∈ B × C × D
これは、f(a, b) = (b′, c, d) と書ける。

# 別の例

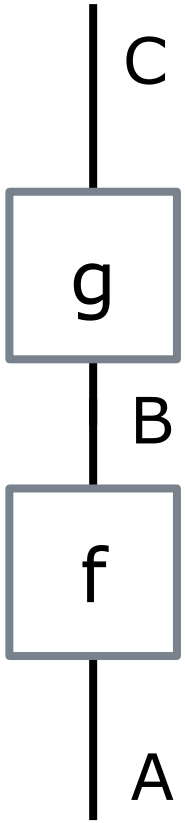
これは、長さ2のビット列から、長さ3のビット列への関数の例である。




$$::\begin{cases}(0,0)\mapsto(0,0,0)\\(0,1)\mapsto(0,1,1)\\(1,0)\mapsto(1,0,1)\\(1,1)\mapsto(0,1,1)\end{cases}$$

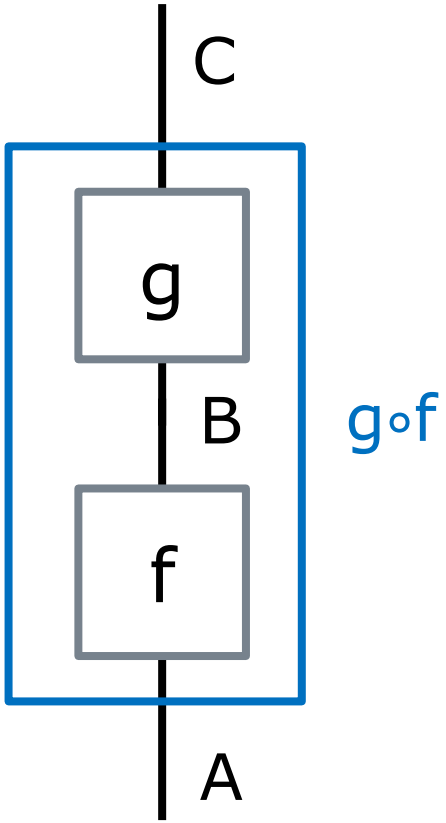
# 関数の合成

# 関数の直列合成




関数 fとgの直列合成は、次の図式で表される。

# 関数の直列合成

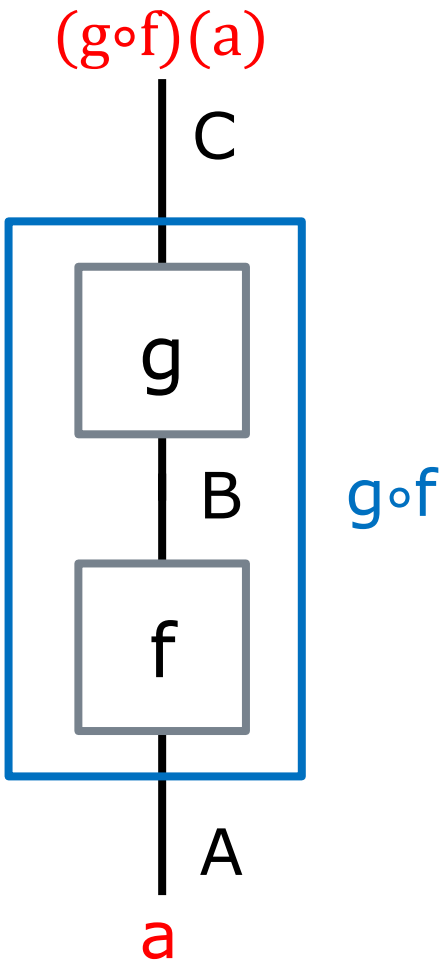



関数 fとgの直列合成は、次の図式で表される。

$f : A \to B , g : B \to C$ の時、

$g \circ f : A \to C$

# 関数の直列合成



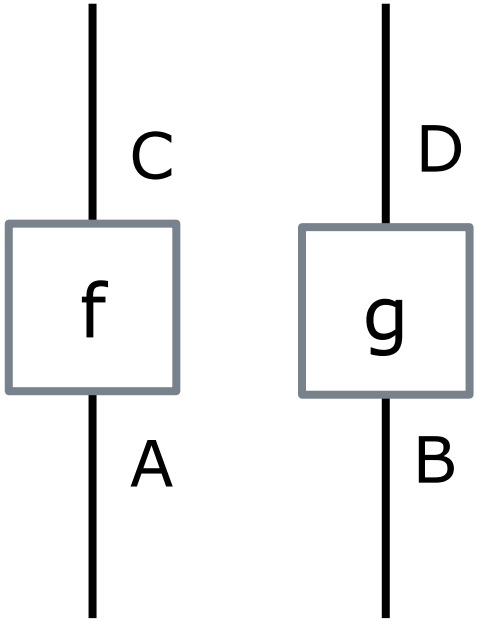

関数 fとgの直列合成は、次の図式で表される。

$f : A \to B , g : B \to C$ の時、

$$g \circ f : A \to C$$

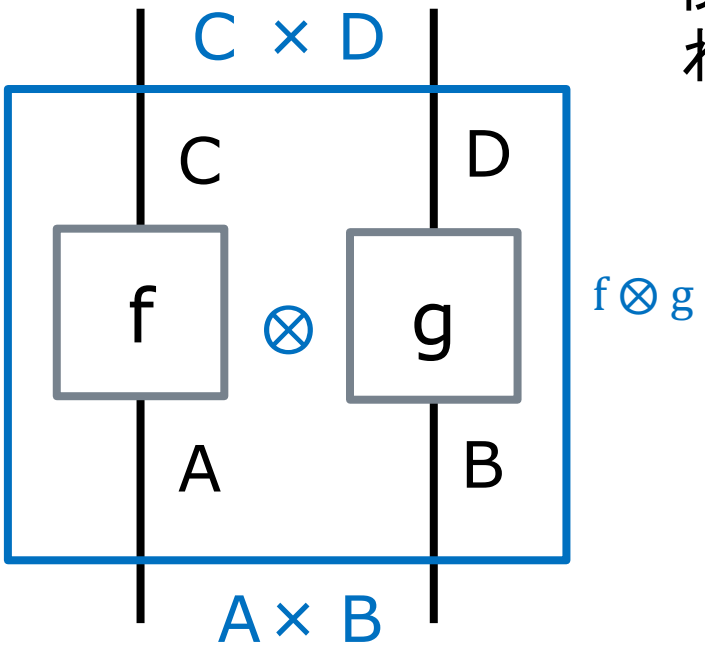
$$(g \circ f)(a) = g(f(a))$$

# 関数の並列合成




関数 fとgの並列合成は、次の図式で表される。

# 関数の並列合成



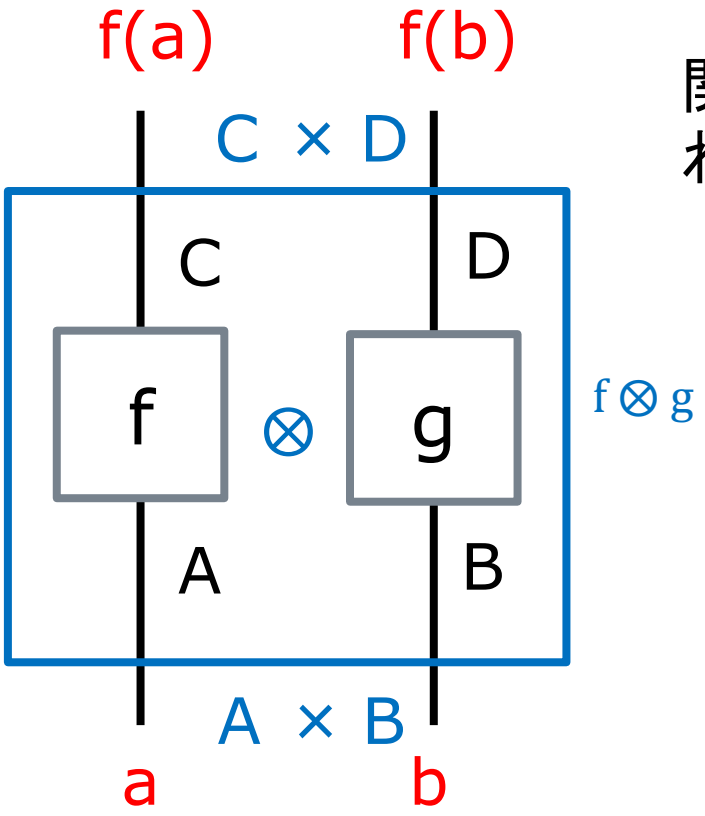

$f \otimes g$

関数 fとgの並列合成は、次の図式で表される。

$f : A \to C , g : B \to D$ の時、

$f \otimes g : A \times B \to C \times D$

# 関数の並列合成




関数 fとgの並列合成は、次の図式で表される。

$f : A \to C , g : B \to D$ の時、

$f \otimes g : A \times B \to C \times D$

$$(f \otimes g)(a, b) = (f(a), g(b))$$

# プロセスとしての「関係」

# 「関係」の表現 1

## aRb

次のような数の大小関係を考える。

$1<2$

$10<100$

$10<1000$

`<`記号が、「前者は後者より小さい」という関係Rを表していると考えれば、先の例は、次のように書ける。

$1R2$

$10R100$

$10R1000$

# 「関係」の表現 2

$(a, b) \in R$

aRbの時、すなわち、a と bとの間に関係 Rが成り立つ時、それを次のように表す。

$(a, b) \in R$

たとえば、Rを「前者は後者より小さい」関係を表すとすれば、

$(1, 2) \in R$

$(10, 100) \in R$

$(10, 1000) \in R$

Rが、AとBの関係なら

$R \subseteq A \times B$

# 関係の表現 3
## R(a)

aRb、すなわち (a, b) ∈ R の時、R(a) で、(a, b) ∈ R を満たすすべてのbの集合を表す。

$$R(a) \coloneqq \{\, b \mid (a, b) \in R \,\}$$

例えば、 Rを 今までと同じように、「前者は後者より小さい」という関係を表すとすれば、R(1)は、1より大きいすべての数である。

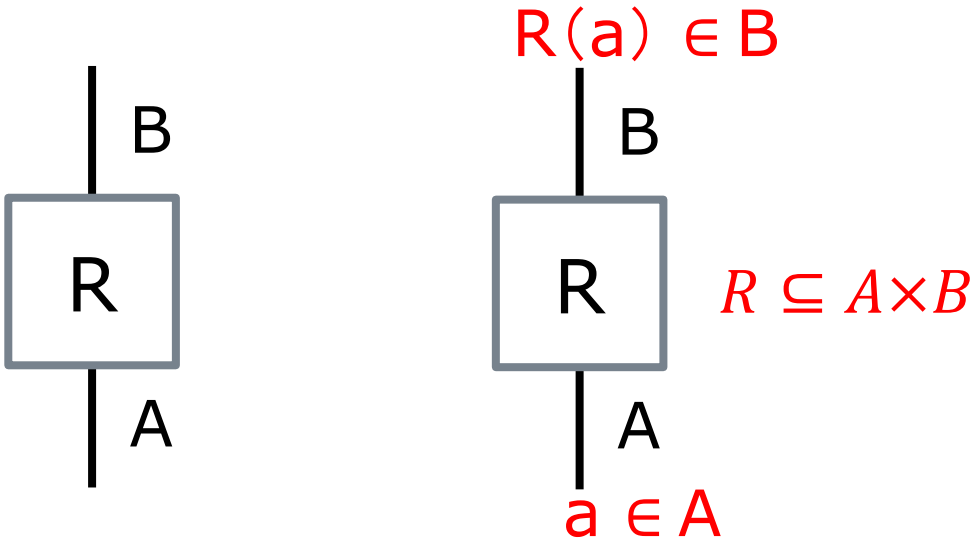
$2 \in R(1)$

$100 \in R(10)$

$1000 \in R(10)$

# Diagram での関係の表現

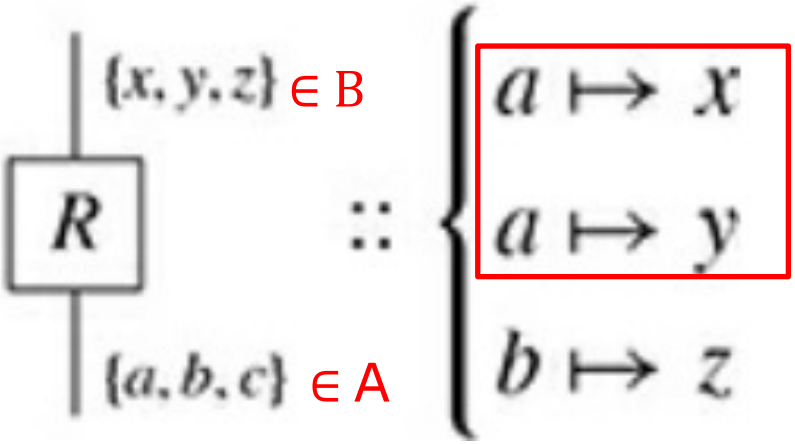
AとBとの関係R( $R \subseteq A \times B$ )は、次のDiagramで表現される。




関数の場合と同じように、aという入力がR(a)という出力に変換されたと考えればいい。

# 関係のDiagramの例

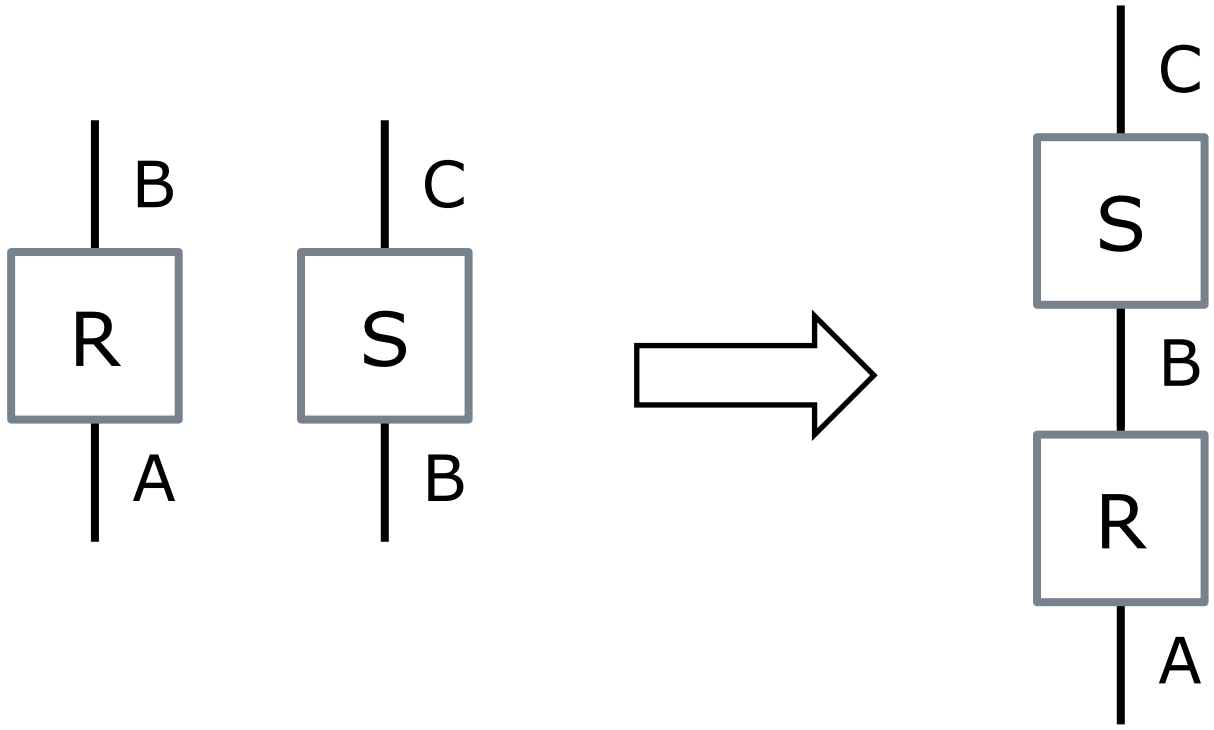
次のものは、AとBの関係のDiagramである。




ただし、このRは関数ではない。
入力aが、二つの値 x, yを取りうるから。

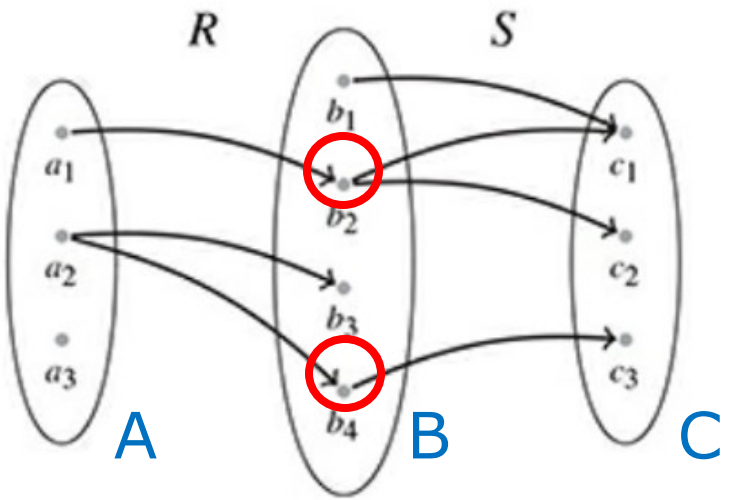
# 関係のDiagramの直列合成

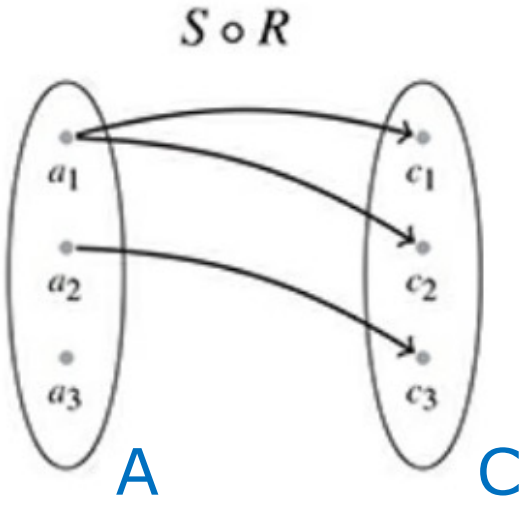
RがAとBとの関係で、SがBとCとの関係である時、二つのDiagramの直列合成を考える。



# 関係のDiagramの直列合成

次のような関係を考えてみる。

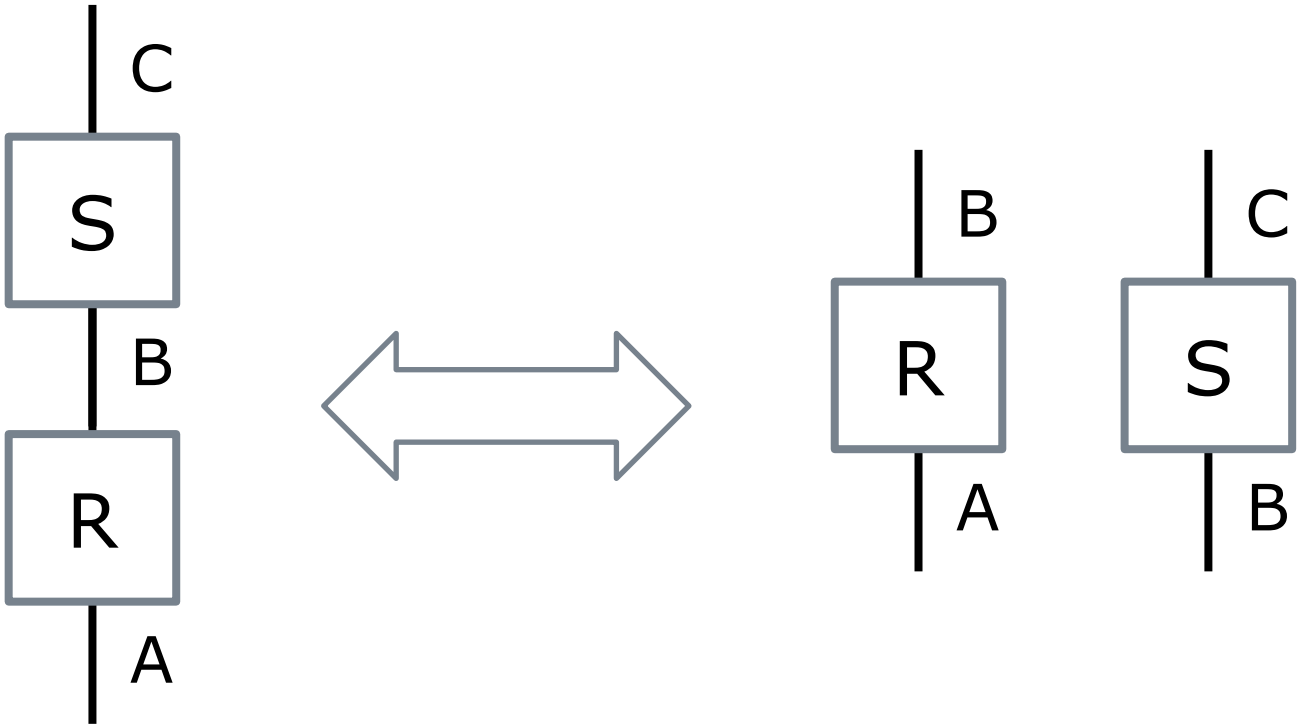



関係 S∘R では、この図のすべての関係が生き残るわけではない。

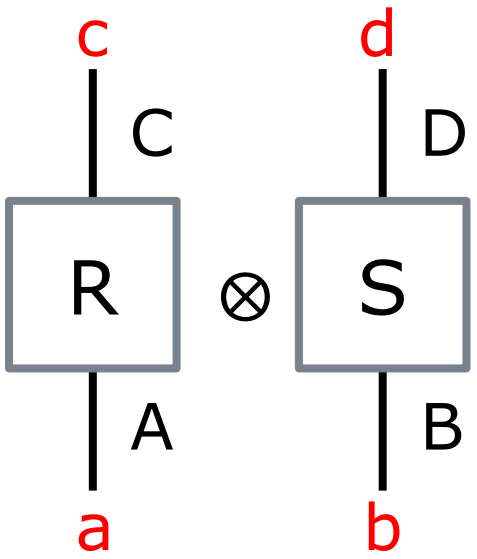
(a, c) ∈ S∘ Rとなるのは、$(a, b) \in R$で$(b,\ c) \in S$ が成り立つ bが存在する場合だけである。

# 関係のDiagramの直列合成




$(a, c) \in S \circ R \quad if\ and\ only\ if \quad \exists b((a, b) \in R\ and\ (b,\ c) \in S)$

# 関係のDiagramの並列合成




$(a, c) \in R \; and \; (b, d) \in S$

$((a, b), (c, d)) \in R \otimes S$

# 「関数」と「関係」

1. 関数も関係も、システム・タイプは集合である。関係のシステム・タイプは、関数のシステム・タイプを含んでいる。
2. 関数は、関係の特別のケースである。関係のプロセスは、関数のプロセスを含んでいる。
3. 関数の直列合成は、関係の直列合成の特殊ケースである。関数の並列合成は、関係の並列合成の特殊ケースである。
4. 関数のプロセス理論は、関係のプロセス理論に含まれる。

$$\mathbf{functions} \subseteq \mathbf{relations}$$

# プロセス中心のアプローチ

# String Diagramのアプローチ

- 前回、量子回路のDiagramと行列表現との対応について述べたが、行列を計算して具体的な展開をしなかった。
- n個の量子からなる系の状態は、ヒルベルト空間上の$2^n$次元のベクトルで表現される。この系の量子状態の変化は、$2^n \times 2^n$ 次元のユニタリ行列の作用として記述される。
- ただ、こうして表現されたヒルベルト空間上の「状態ベクトル」も「行列」も、nが大きくなるにつれて、実際に計算することは急速に難しくなる。
- String Diagramのアプローチは、こうしたアプローチとは異なっている。

*I would like to make a confession which may seem immoral: I do not believe absolutely in Hilbert space anymore.*

*— John von Neumann,*
*letter to Garrett Birkhoff, 1935.*

# 「状態」中心から「プロセス」中心へ

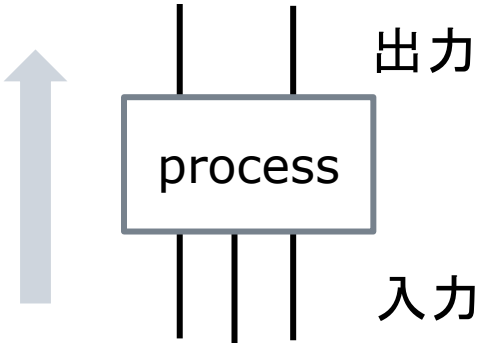
# 「状態」中心から「プロセス」中心へ

- String Diagramは、伝統的なアプローチのようにヒルベルト空間の「状態」からは、出発しない。
- String Diagramは、「状態」の変化を表す「プロセス」から出発して、かつ、それを図式として表示しようとする。
- それでは。従来のアプローチでは大きな役割を担っていた「状態」や「行列」は、String Diagram では、どのように表現されるのであろうか？

# 特別な「プロセス」

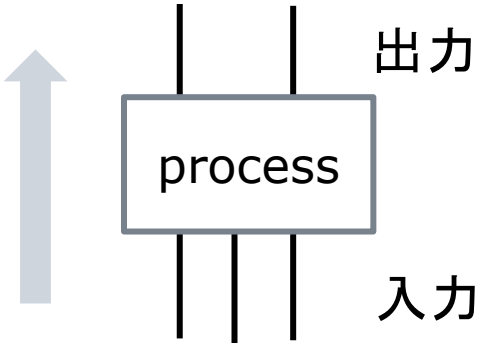
# 特別な「プロセス」

- DiagramでBoxで示される「プロセス」は、一般には、下からBoxに入るWireで表される「入力」と、Boxから出て上に向かうWireで表される「出力」を持つ。



# 特別な「プロセス」

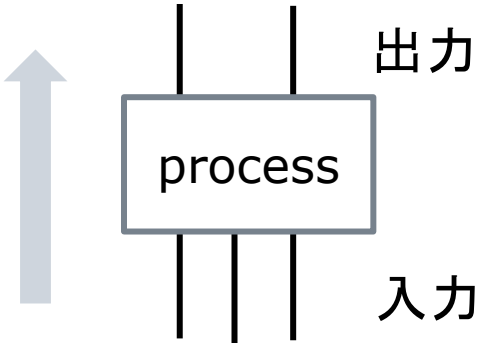
- DiagramでBoxで示される「プロセス」は、一般には、下からBoxに入るWireで表される「入力」と、Boxから出て上に向かうWireで表される「出力」を持つ。




- ここでは、次のような特別の「プロセス」を考える。

# 特別な「プロセス」

- DiagramでBoxで示される「プロセス」は、一般には、下からBoxに入るWireで表される「入力」と、Boxから出て上に向かうWireで表される「出力」を持つ。



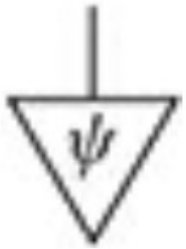

- ここでは、次のような特別の「プロセス」を考える。

1. 入力を持たず出力のみを持つプロセス
2. 出力を持たず入力のみを持つプロセス
3. 入力も出力も持たないプロセス

# 「状態」、「効果」、「数」

# 「状態」
# 入力を持たず出力のみを持つプロセス

- 「状態」は、全く入力を持たず、出力のみを持つ特別なプロセスである。次のような図で表す。



- ある系Aが、可能的には複数の状態を取りうるとしても、この図で表される「状態」は、ある操作からなる「準備過程」によって作られた、系Aの「特定」の「状態」である。

# 「状態」
# 入力を持たず出力のみを持つプロセス

- 入力を持たないということは、このシステムの来歴を我々は知らないことを意味している。
- 例えば、「ここに 0-stateに初期化された qubitがある」という時、我々は、その状態がどのように準備されたかについて知る必要はない。

# 「効果」
# 出力を持たず入力のみを持つプロセス

- 「効果」は、全く出力を持たず、入力のみを持つ特別なプロセスである。次のような図で表す。

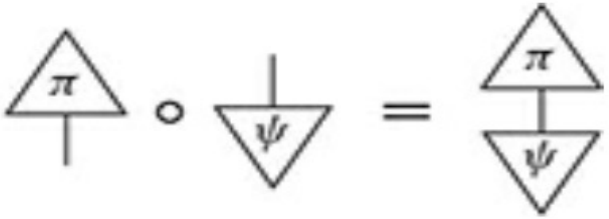

- 「効果」は「状態」の双対な概念である。「効果」はあるシステムによるものなのだが、出力を持たないというのは、その後、その結果は二度と使われることがないことを意味する。
- 「効果」の最も簡単な例は、システムが破壊され、以前のシステムを無視することである。

# 「効果/テスト」
# 出力を持たず入力のみを持つプロセス

- 「効果」のもう一つの解釈は、それを「テスト」と考えることである。
- 我々が、ある「効果」が起きたということは、我々がシステムを「テスト」して、ある特徴が確認できたということである。

# 「数」
# 入力も出力も持たないプロセス

- String Diagramの、極めて興味ふかいプロセスの解釈の一つは、入力も出力持たないプロセスを「数」に対応づけることである。
- 「状態」の出力を「効果」の入力と直列に合成しよう。




- この入力も出力も持たないプロセスを「数」と言って、次のような記号で表す。

# 「数」
# 入力も出力も持たないプロセス

- もし、λだとμが「数」ならば、次のものも「数」である。

$$\lambda \otimes \mu :=$$ 

- 「数」は、「結合律」を満たし、単位元を持つ。

$$(\lambda \otimes \mu) \otimes \nu =$$  $$= \lambda \otimes (\mu \otimes \nu)$$

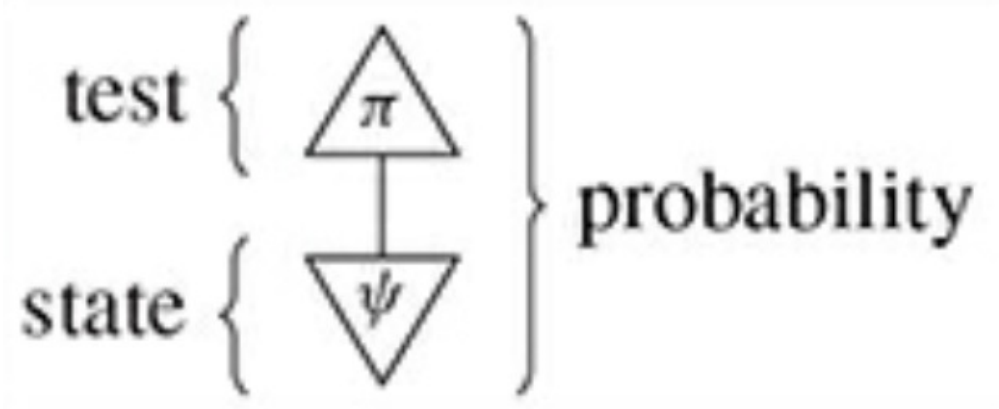
$$\lambda \otimes 1_I =$$  $$= \lambda$$

# 「数」
# 入力も出力も持たないプロセス

- 「数」は入力も出力もないので、Diagramの中を自由に移動できる。

# 一般化されたBornルール

- 「状態」と「効果」が出会うと「数」がうまれる？
  このことの一つの解釈は、この「数」を「確率」として解釈することである。これを「一般化されたBornルール」と呼ぶ。

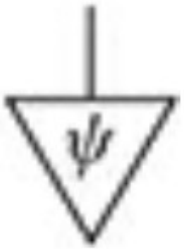



- この解釈については、「観測」との関係で、のちに詳しく触れる。

# ket記法との対応

# 三つの特別な「プロセス」

状態

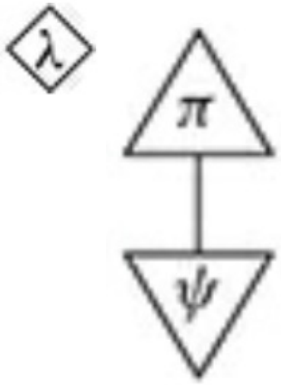



「状態」は、全く入力を持たず、出力のみを持つ特別なプロセスである。

効果

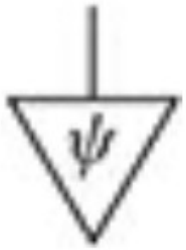

「効果」は、全く出力を持たず、入力のみを持つ特別なプロセスである。

数




「数」は、入力も出力持たないプロセスである。

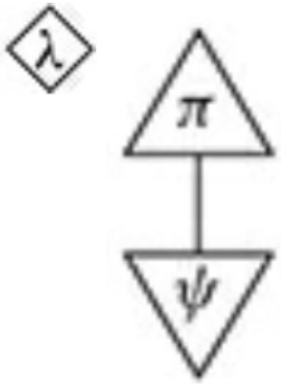
# 三つの特別な「プロセス」と ket記法の対応

状態 


$|\psi\rangle$ Diracのket

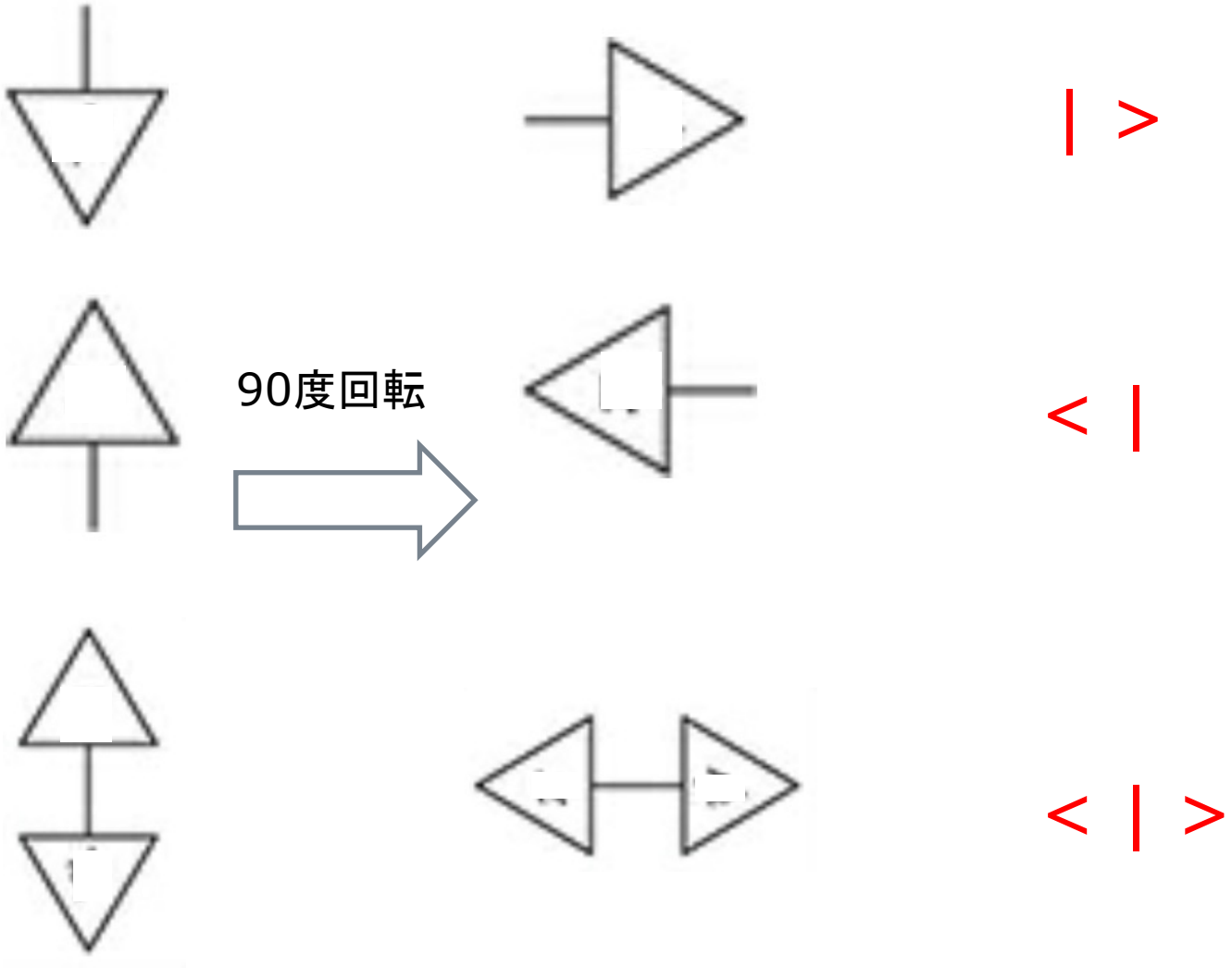
効果 


$\langle\pi|$ Diracのbra

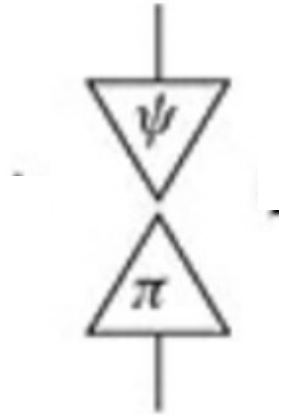
数 


$\langle\pi|\psi\rangle$ Diracのbraket

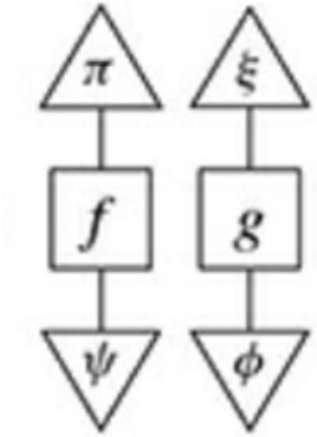
# 対応の覚え方



# 対応の例

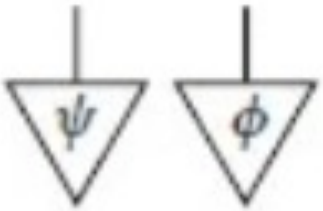



$$|\psi\rangle\langle\pi|$$



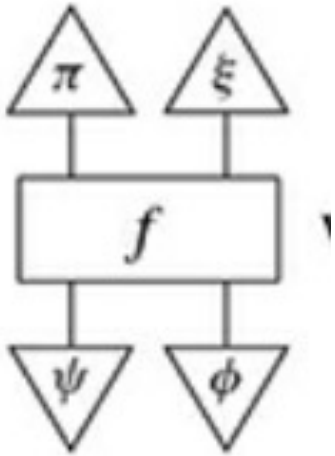

$$\langle\pi|f|\psi\rangle\langle\xi|g|\phi\rangle$$

# 対応の例




$|\psi\phi\rangle$


$\langle\pi\xi|$




$\langle\pi\xi|f|\psi\phi\rangle$


50

# String Diagramを学ぶ

## 第四部　String Diagramの世界

# String Diagramを学ぶ
# 第四部　String Diagramの世界

分離可能性

プロセスと状態の双対性

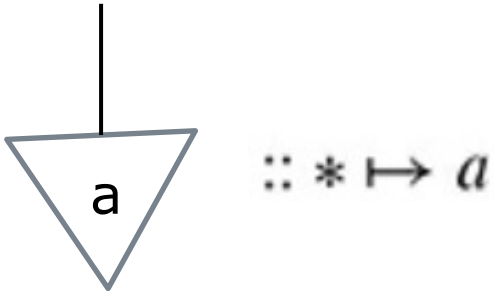
yanking 等式

Transpose

# 分離可能性

# 関数で表現される状態

状態が、ある集合Aの要素aで表されるとしよう。これを次のように表す。



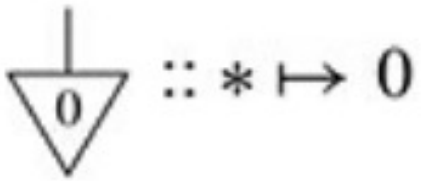

一点からなる集合を {*}で表し、{*}から集合Aへの関数を考える。この関数は、Aの一点 $a \in A$ をポイントする。

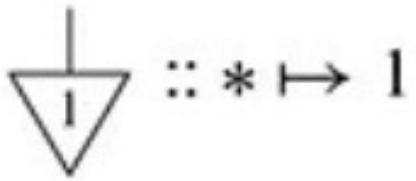
逆に、すべての $a \in A$について、* をaに移す関数が一つ存在する。Aの要素と、{*}から集合Aへの関数は、一対一に対応し、同じものと考えていい。この関数を、a と呼ぼう。

集合の要素で表される状態を、関数で表現される状態と呼ぶ。

# 関数で表現される状態
# A := {0, 1}の場合の例

A := {0, 1}とすれば、関数で表現される状態は、次の二つがあることになる。

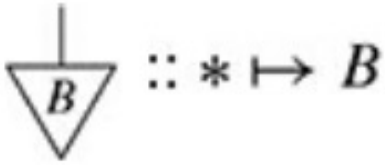


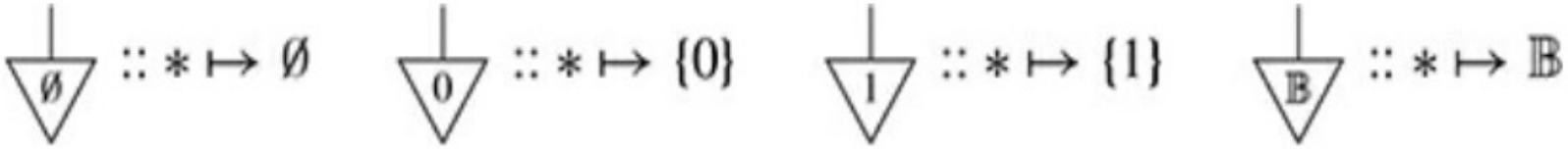
# 関係で表現される状態

今度は、関係で表現される状態を考えよう。

関数で表現される状態は、一点に対応する集合Aの要素によって表されたのだが、今度は、集合Aの部分集合Bとの対応を考える。




B:={0, 1} とすると、その部分集合は、{∅, {0},{1},{0,1}}の4つある。この時、関係で表現される状態は、次のようになる。



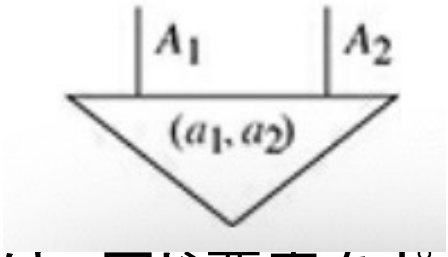
# ⊗ -分離可能性

二部状態（二つの出力を持つ状態）ψが、次のように二つの状態$\psi_1$ と$\psi_2$の並列合成として表される時、ψは「⊗-分離可能」であるという。

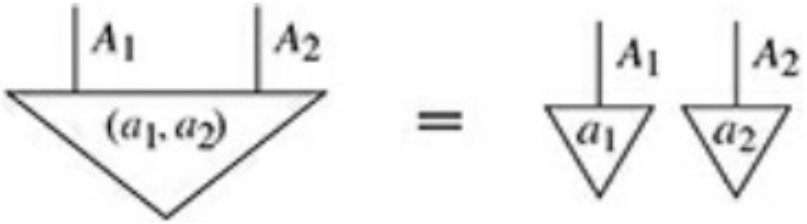
# 関数で表現される二部状態

関数で表現される全ての二部状態は、⊗ -分離可能である。

先に見たように、関数で表現される状態は、関数がポイントする一点で表現される。 関数が表現する二部状態がポイントするこの点を $(a_1, a_2) \in A_1 \times A_2$ とすると、次のように表すことができる。



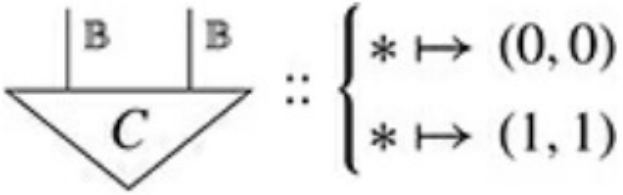

次の図の両方の状態は、同じ要素をポイントしている。



# 関係で表現される二部状態

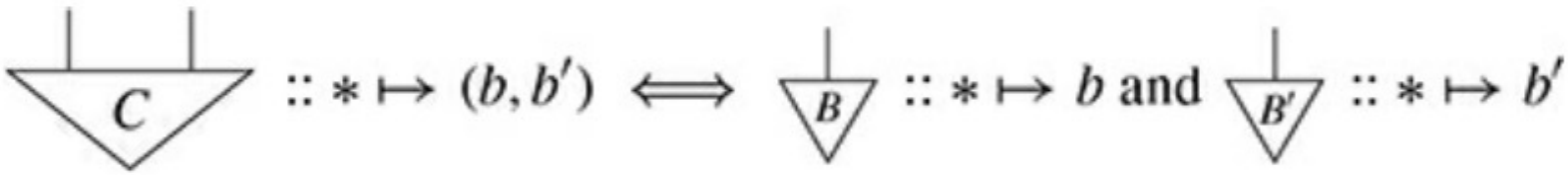
関係で表現される二部状態には、⊗ -分離可能でないものが存在する。

次のような、関係で表現される二部状態を考えよう。




$\mathrm{C} = \{(0,0),(1,1)\} \subseteq \mathbb{B} \times \mathbb{B}$ である。
Cが⊗ -分離可能で、$B, B' \subseteq \mathbb{B}$である $\mathbb{B}$の部分集合$B, B'$ について、次が成り立っているとしよう。

# 関係で表現される二部状態



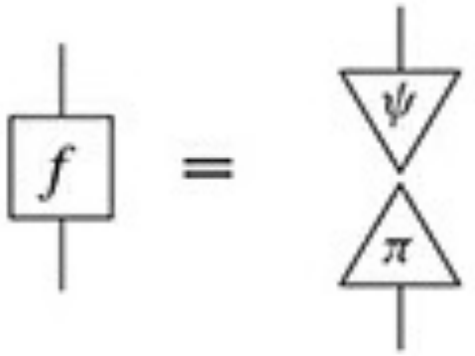

すなわち、(b、b’) ∈ $C \Leftrightarrow b \in B$ $and$ $b' \in B'$ だとしよう。

この時、(0、0) ∈ $C$ から、0 ∈ B がいえ、
(1、1) ∈ $C$ から、1 ∈ B′ がいえる。
これから、(0、1) ∈ $C$ ということになるが、
Cは、(0, 1) を含んでいない。これは矛盾。

Cは、⊗ -分離可能でない

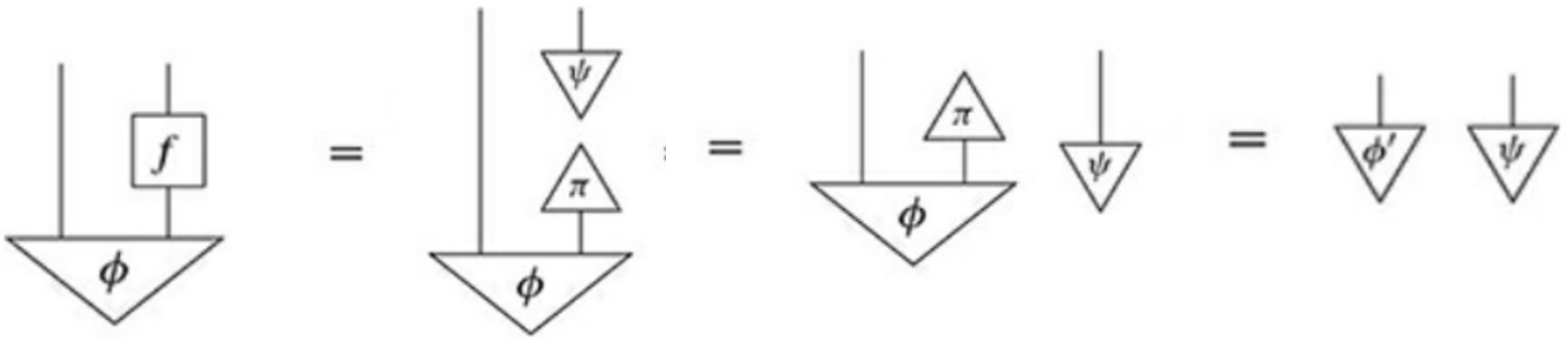
# ∘-分離可能性

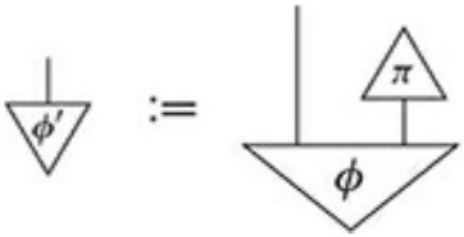
プロセス f が、ある効果 π と状態 ψ で次の形にかける時、プロセス f は、「∘-分離可能」という。



# ∘-分離可能性 と ⊗-分離可能性

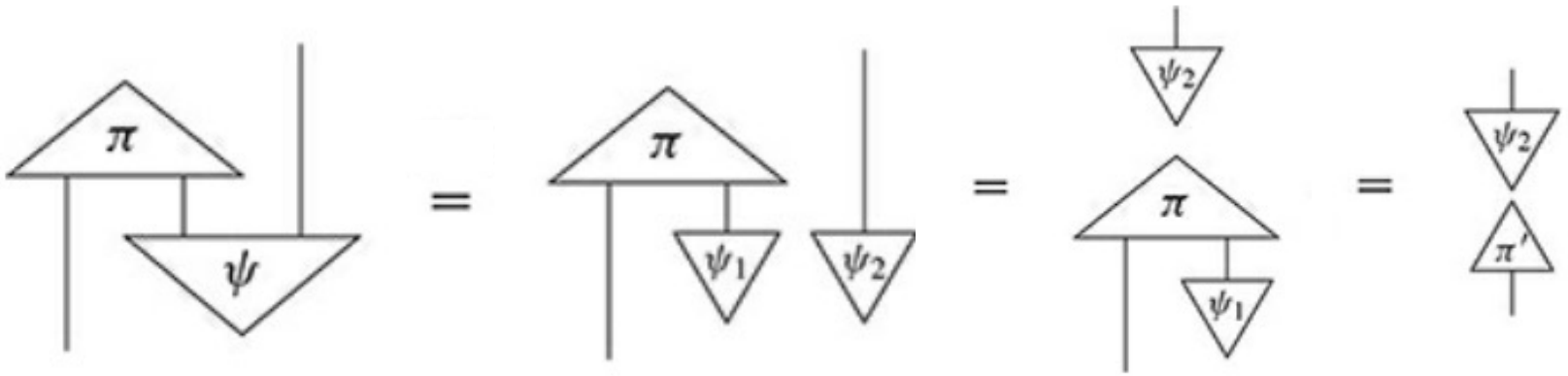
プロセス f が、∘-分離可能だとしよう。この時、任意の二部状態 φ について、次の状態は、⊗-分離可能である。

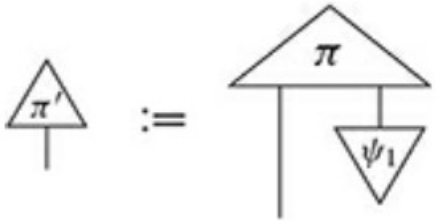


# ∘-分離可能性 と ⊗-分離可能性

もし、状態 ψ が⊗-分離可能だとすれば、任意の二部効果 π について、次のプロセスは、∘-分離可能である。



# プロセスと状態の双対性

# プロセスと状態の双対性

ここでは、まず、プロセスを二部状態に変換する操作と、二部状態をプロセスに変換する操作を定義する。次に、この二つの操作が、ある条件のもとで、お互いに逆変換になっていることを示す。

このことは、この条件のもとでは、プロセスと状態のあいだに、一対一の対応が存在することを意味する。あるプロセスにはある状態に一意に対応し、同様に、ある状態にはあるプロセスが一意に対応する。

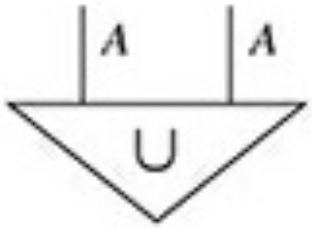
同型なものは同じであるという見方をすれば、プロセスと状態は、同じものである。これを「プロセスと状態の双対性（duality）」という。

ここでは、そのことを見ていく。

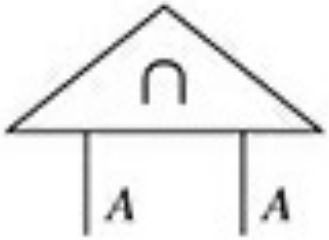
ここで中心的な役割を果たすのは、次の cup とcap である。

# cup と cap

次のような二部状態と二部効果を考える。




cup



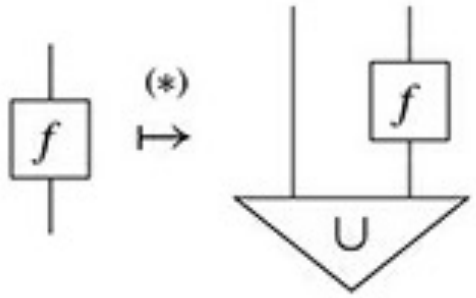

cap

前者を cup 、後者を cap と呼ぶ。

# cup と cap を使って
# プロセスを状態に、状態をプロセスに変換する

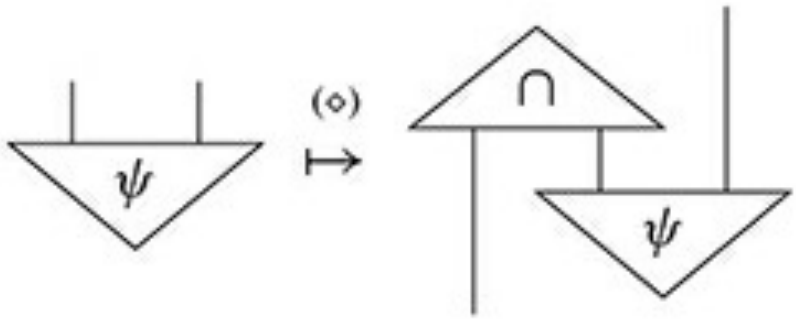
cup を使って、プロセスを状態に変える次のような操作を考えることができる。

変換 PtoS




cap を使って、状態をプロセスに変える次のような操作を考えることができる。

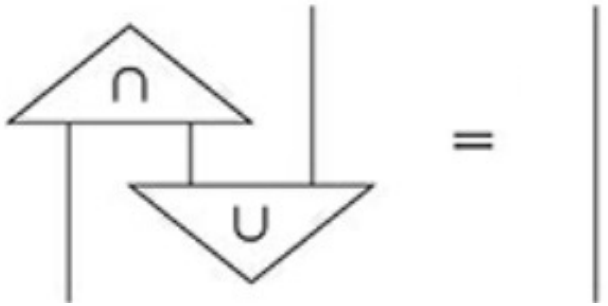
変換 StoP



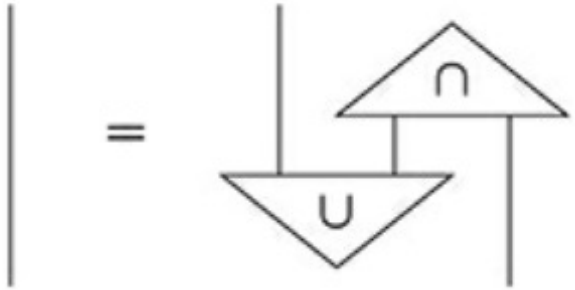
# 変換PtoSと変換StoPの関係

ここでは、cupとcapが次の条件 a, b を満たす時、

条件 a



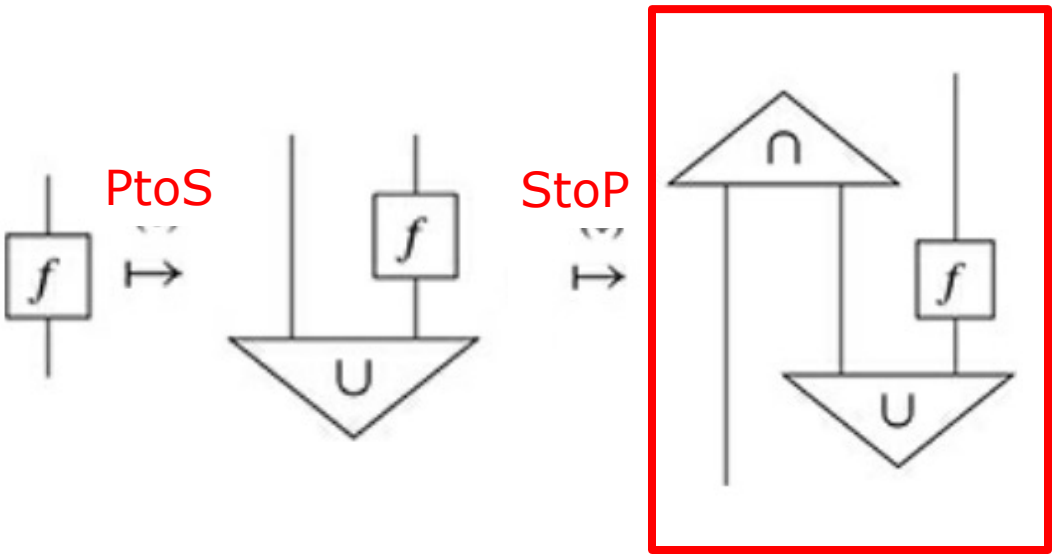
条件 b



変換PtoS と変換StoPは、逆の操作であることを示す。

# 変換PtoSと変換StoPの関係
# プロセスの場合

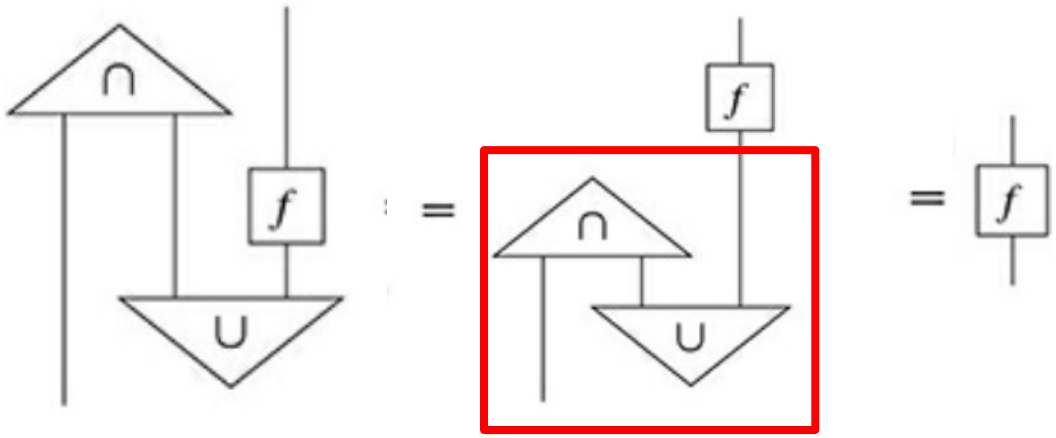
プロセスを状態に変換する（変換PtoS）。




その後に、状態をプロセスに変換する（変換StoP）。
二つの変換の結果に注目する。

# 変換PtoSと変換StoPの関係
# プロセスの場合

二つの変換の結果は、次のようになる。




この部分に、条件 a を使うと、

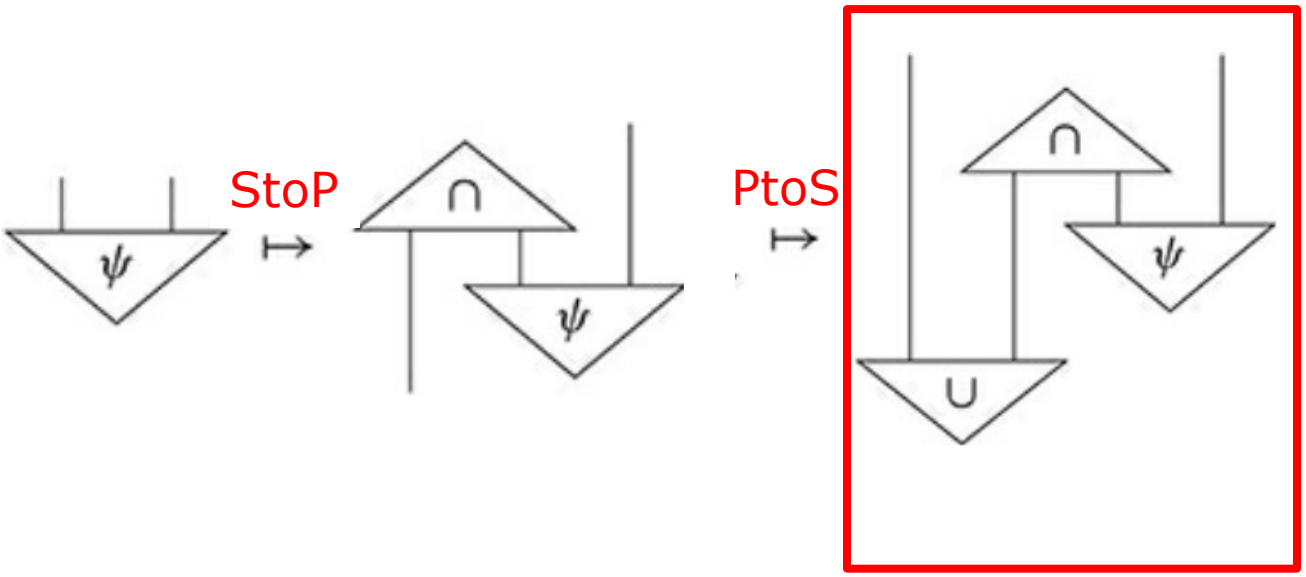
次が導かれる。

このことから、StoP ∘ PtoS(f) = f が分かる。

プロセス f について、StoPとPtoSは、逆の変換である。

# 変換PtoSと変換StoPの関係
# 状態の場合

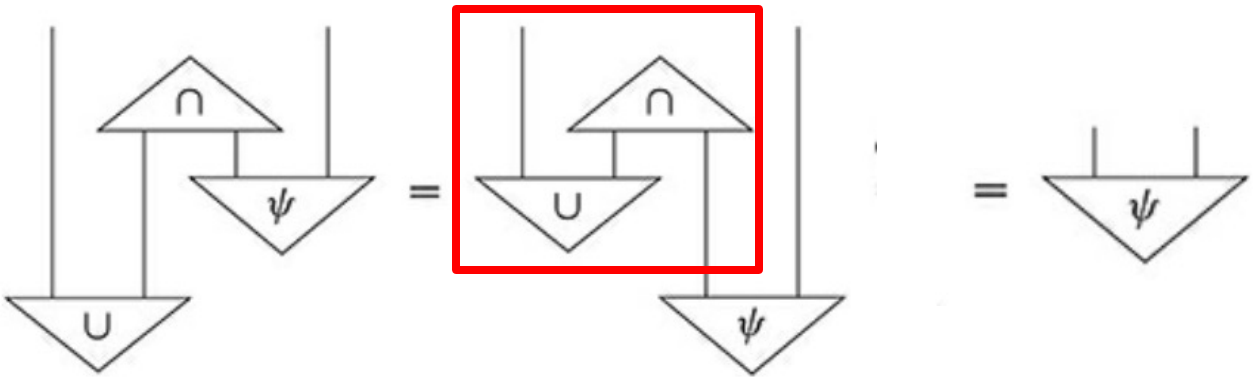
状態をプロセスに変換する（変換StoP）。




その後に、プロセスを状態に変換する（変換PtoS）。
二つの変換の結果に注目する。

# 変換PtoSと変換StoPの関係
# 状態の場合

二つの変換の結果は、次のようになる。




この部分に、条件 b を使うと、

次が導かれる。

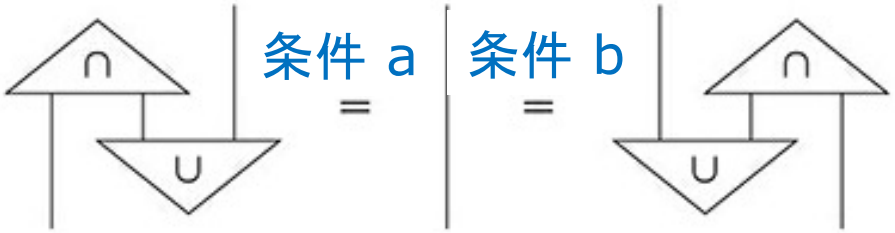
このことから、PtoS ∘ StoP(ψ) = ψ　が分かる。

プロセス ψについて、PtoSとStoPは、逆の変換である。

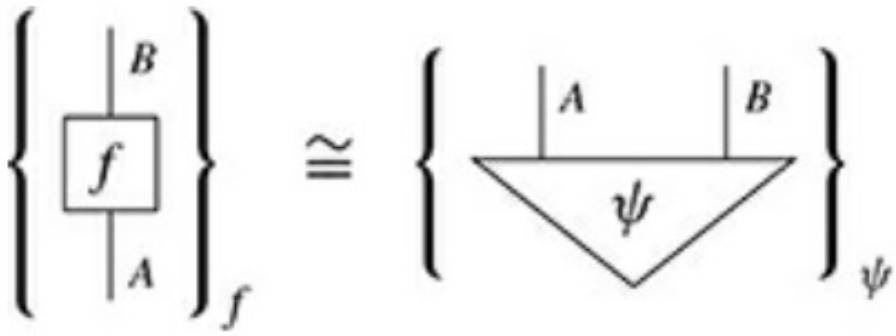
# 条件a,b とプロセスと状態の対応

条件a, b　が成り立つと、
プロセスを状態に変換する PtoS と
状態をプロセスに変換する StoP とは、
お互いの逆変換になっている。

条件a, b　が成り立つと、
プロセスと状態は、一対一に対応する




条件 a, b が成り立つ時、
入力タイプがAで出力タイプがBのプロセスと、
A × Bのタイプの状態は、
一対一に対応する。

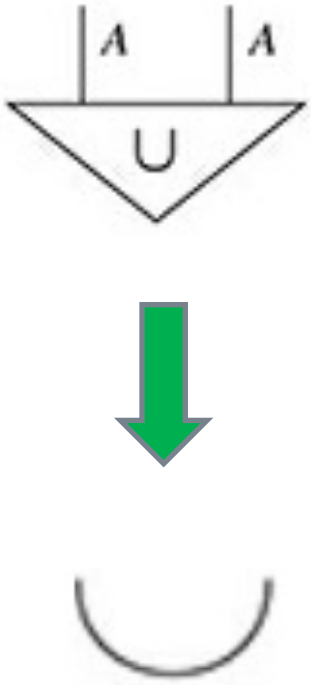


# yanking 等式

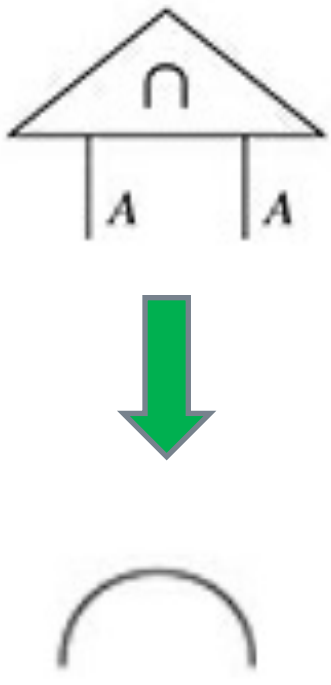
# cup, cap の新しいノテーション

cupとcapに、次のような新しいノテーションを与える。

cup



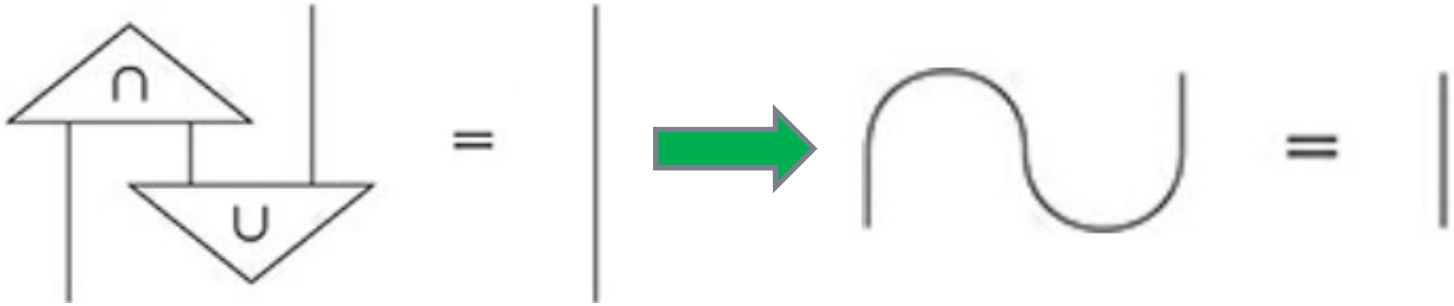

cap



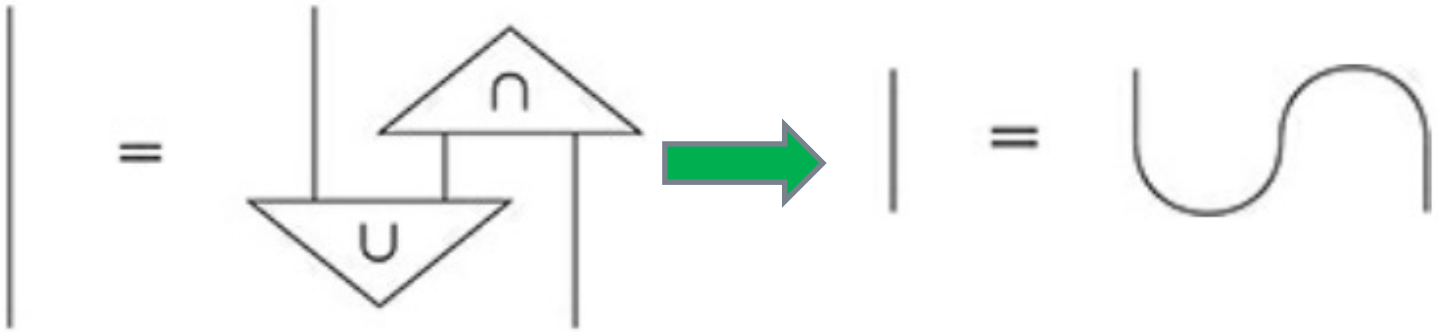
# cup, capについての条件a, b を
# 新しいノテーションで表す

このノテーションで、条件 a, bは、次のように表現される

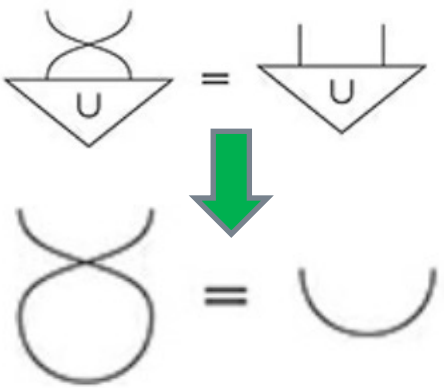
条件 a

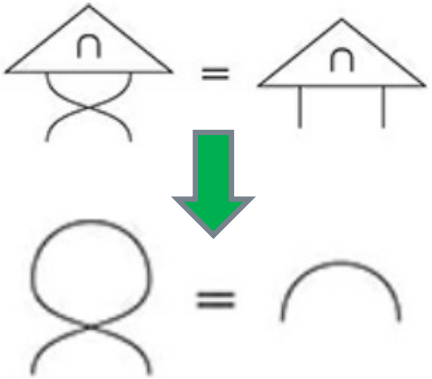


条件 b

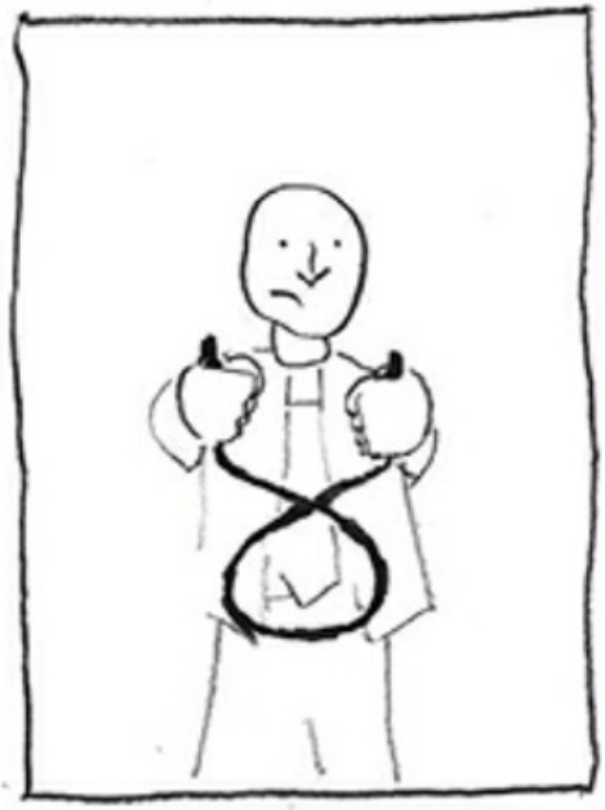


yanking 等式





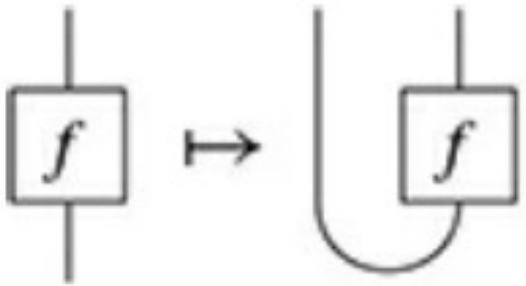
yanking 等式



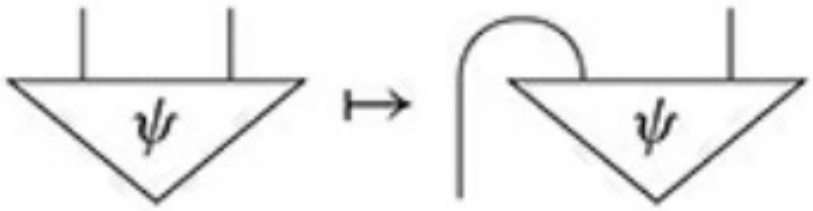
# プロセスと状態の対応

プロセスと状態の一対一対応を与えるのは、新しいノテーションによれば、次のような変換である。

プロセスから状態



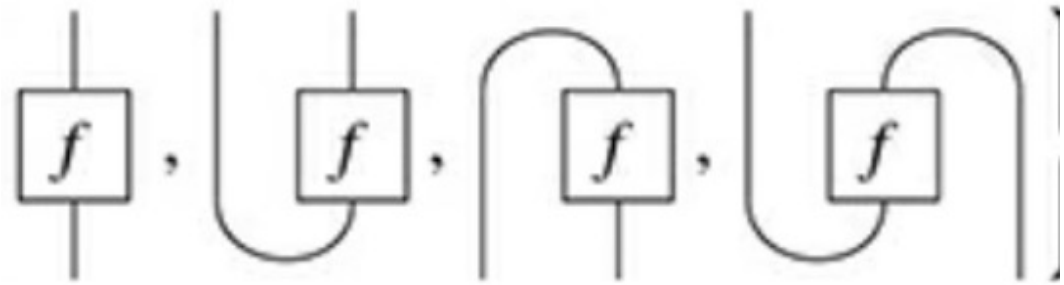

状態からプロセス




これらの変換は、入力を出力に、出力を入力に変える変換である。

# TransposeとTrace

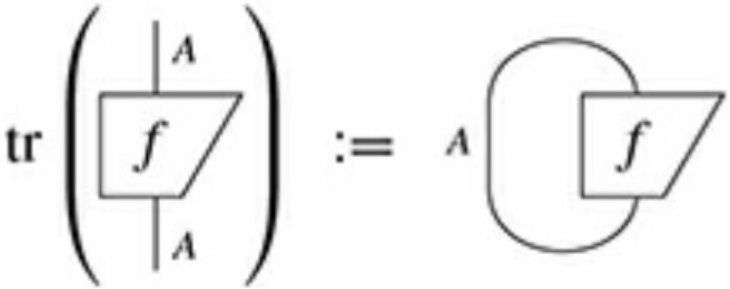
入力が一つ、出力が一つのプロセス f は、入力と出力の変換で、次の四つの形を持つ。最後の形をTranspose という。




Transpose

String Diagram では、プロセスの出力をプロセスの入力に結合することも可能である。これをプロセスのTraceといい、tr(f)で表す。

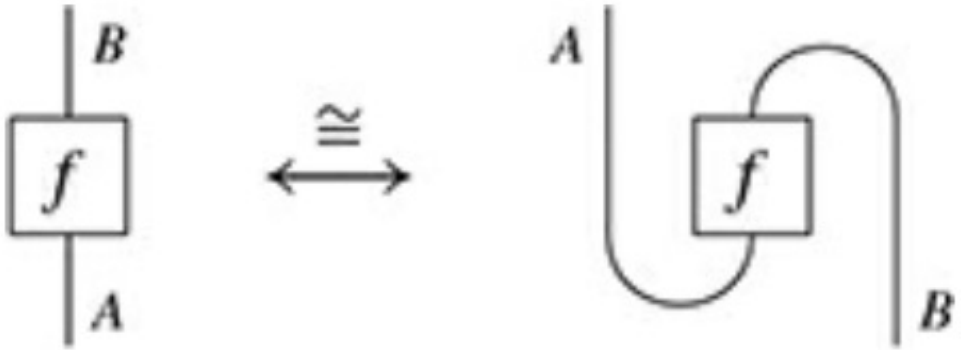
Trace



# Transpose

# Transpose $f^T$

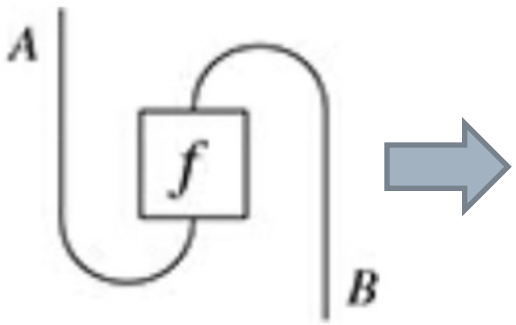
入力Aで出力Bを持つプロセス f を、次の操作で入力Bで出力Aをもつプロセスに変換することができる。



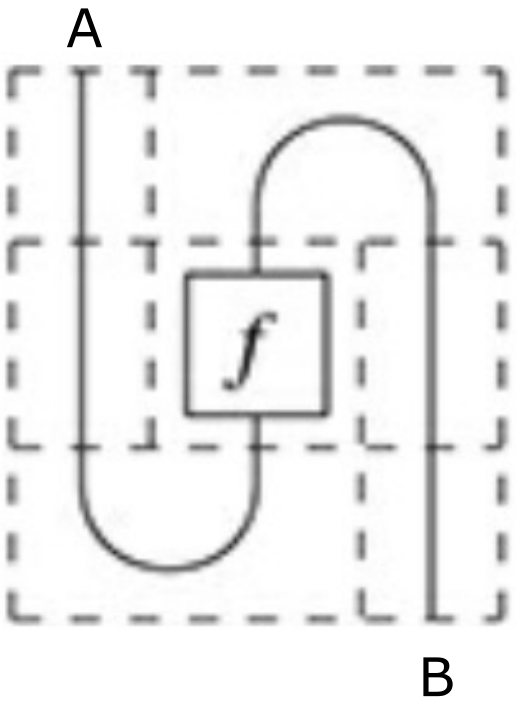

Transpose $f^T$

このプロセスを プロセス f のTranspose といい、$f^T$で表す。

# Transpose $f^T$の分解




Transpose $f^T$




$(1_A \otimes \cap_B)$

$(1_A \otimes f \otimes 1_B)$

$(\cup_A \otimes 1_B)$

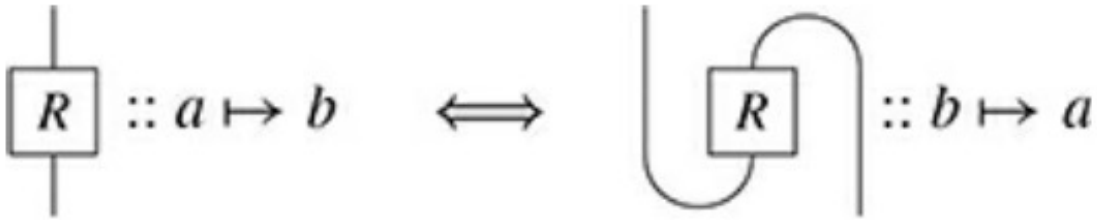
$$f^T = (1_A \otimes \cap_B) \circ (1_A \otimes f \otimes 1_B) \circ (\cup_A \otimes 1_B)$$

# 逆プロセス $f^{-1}$

入力Aで出力Bを持つプロセス f に対して、入力Bで出力Aを持つあるプロセスが存在して、次の式を満たす時、プロセスf は「逆を持つ」といい、$f^{-1}$で表す。

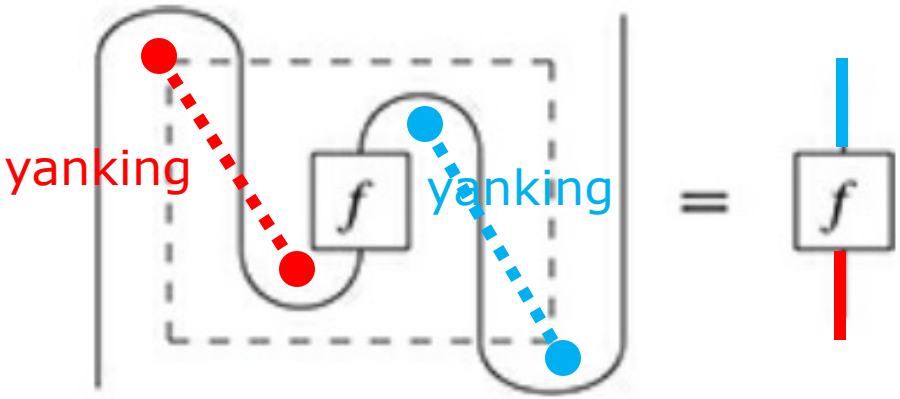
# 関係RのTranspose

関係RのTransposeは、Rの逆関係になる。



# TransposeのTranspose

あるプロセスのTransposeのTranspose は、元のプロセスに等しい。



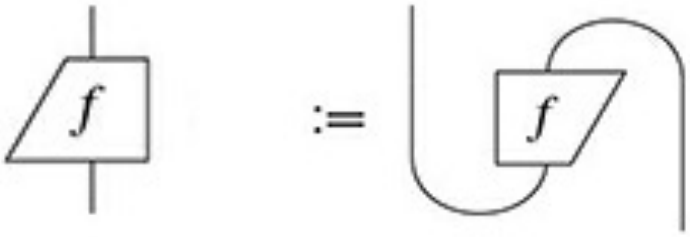

$$(f^T)^T = f$$

# プロセスの新しいノテーション

プロセスを表現するのに、長方形のボックスでなく、その隅を切り落とした台形の図形を使う。


このノテーションで、fのtransposeを次の図形で表す。

Transposeの定義から、次の関係が成り立つ。

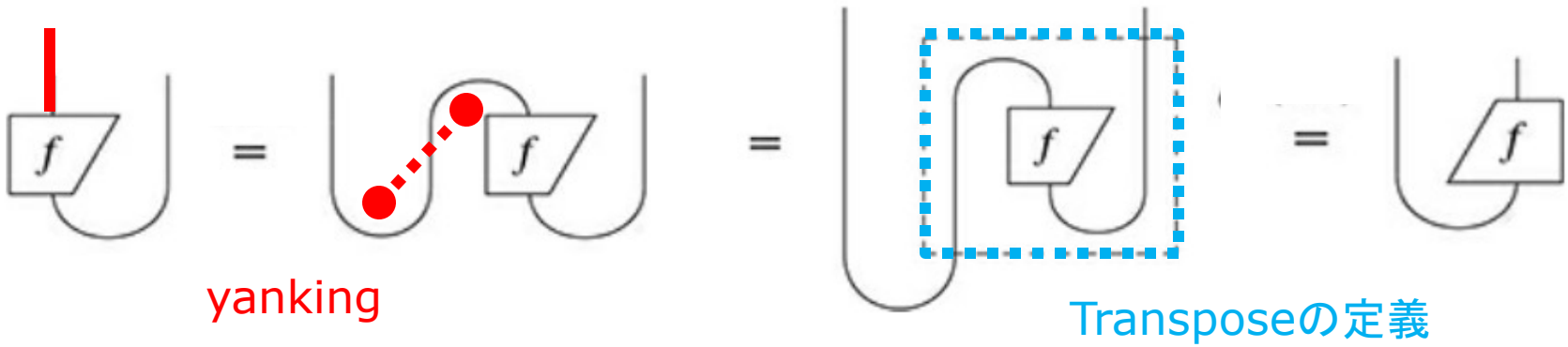



fのtranspose

# cup, capと Transpose

次の関係が成り立つ。


証明

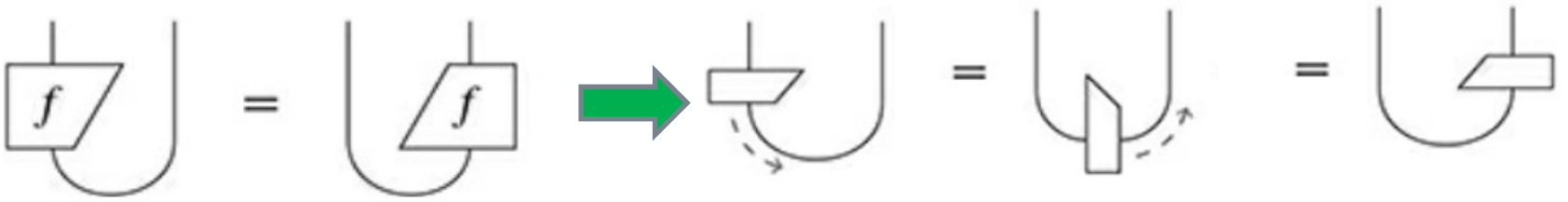


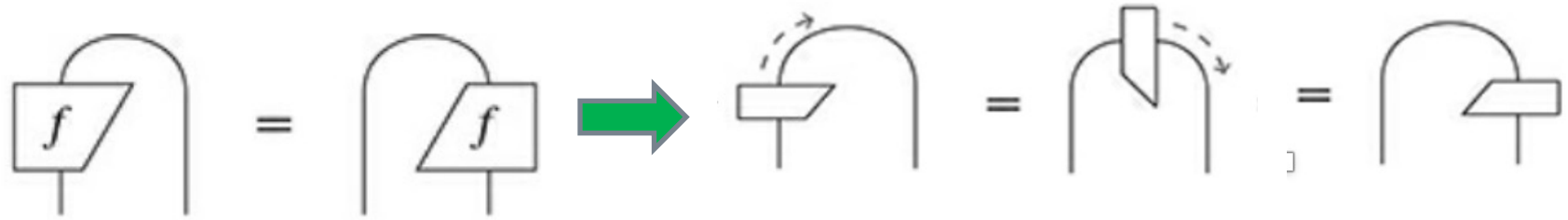

yanking

Transposeの定義

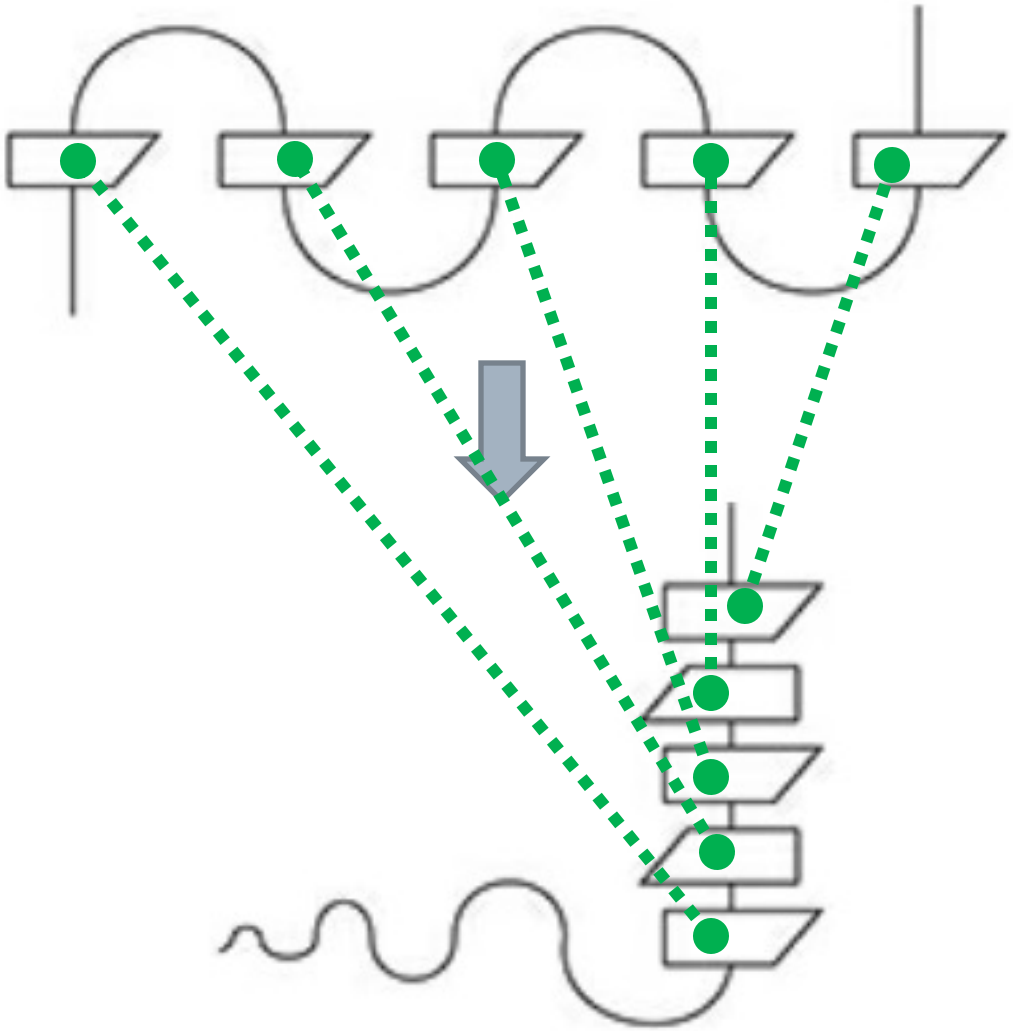
同様に、次の関係も成り立つ。

# 直観的な導出 …

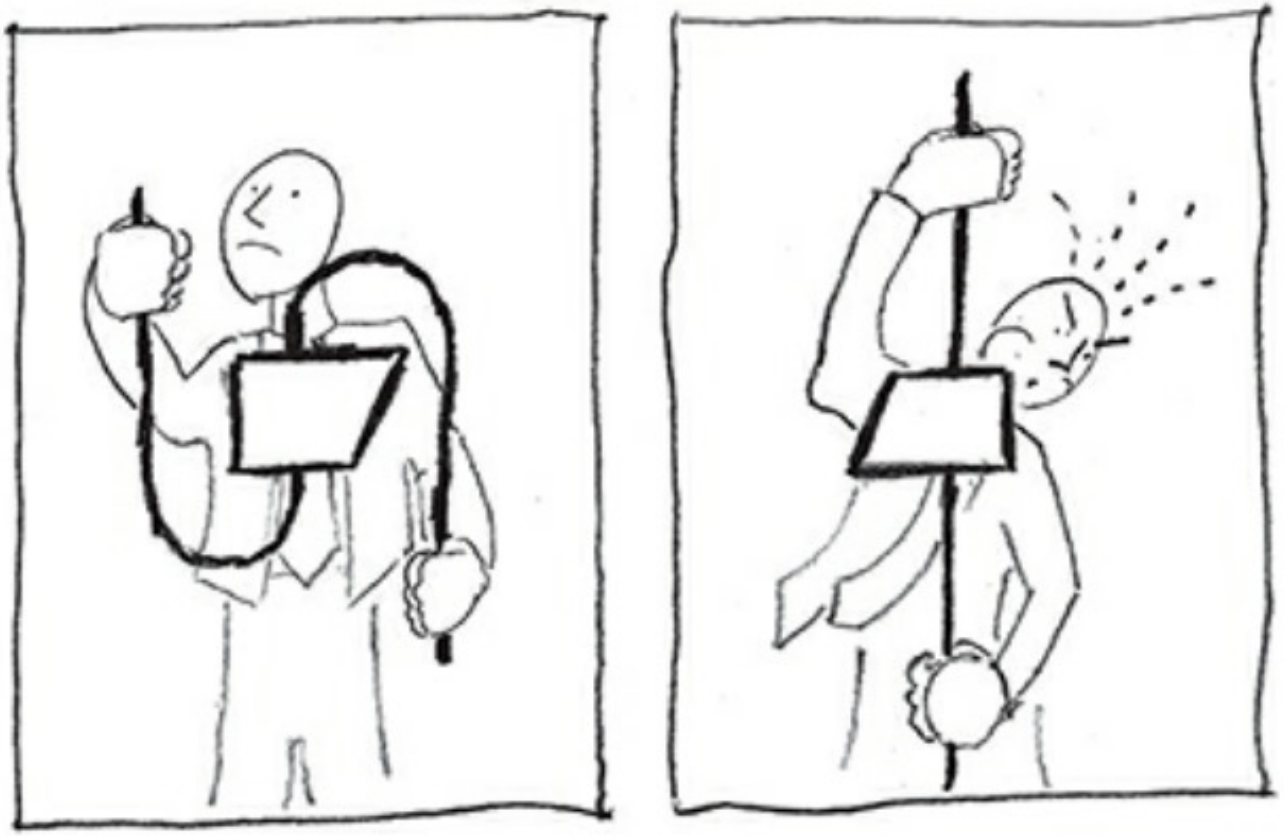


# 一般的に …



# こんな感じ




50