# From Reversible Programming Languages to Univalent Universes and Back

MFPS XXXIII
Ljubljana
June 2017

*Jacques Carette (C*),*
*Chao-Hong Chen (C**),*
*Vikraman Choudhury (C**),*
*Robert Rose (M**), and*
***Amr Sabry** (C**)*
*(*) McMaster University*
*(**) Indiana University)*
*(C) Computer Science*
*(M) Mathematics*

# Models of Computation

*Mathematical models of computation are abstract constructions, by their nature unfettered by physical laws. However, if these models are to give indications that are relevant to concrete computing, they must somehow capture, albeit in a selective and stylized way, certain general physical restrictions to which all concrete computing processes are subjected.*

*(Toffoli 1980)*

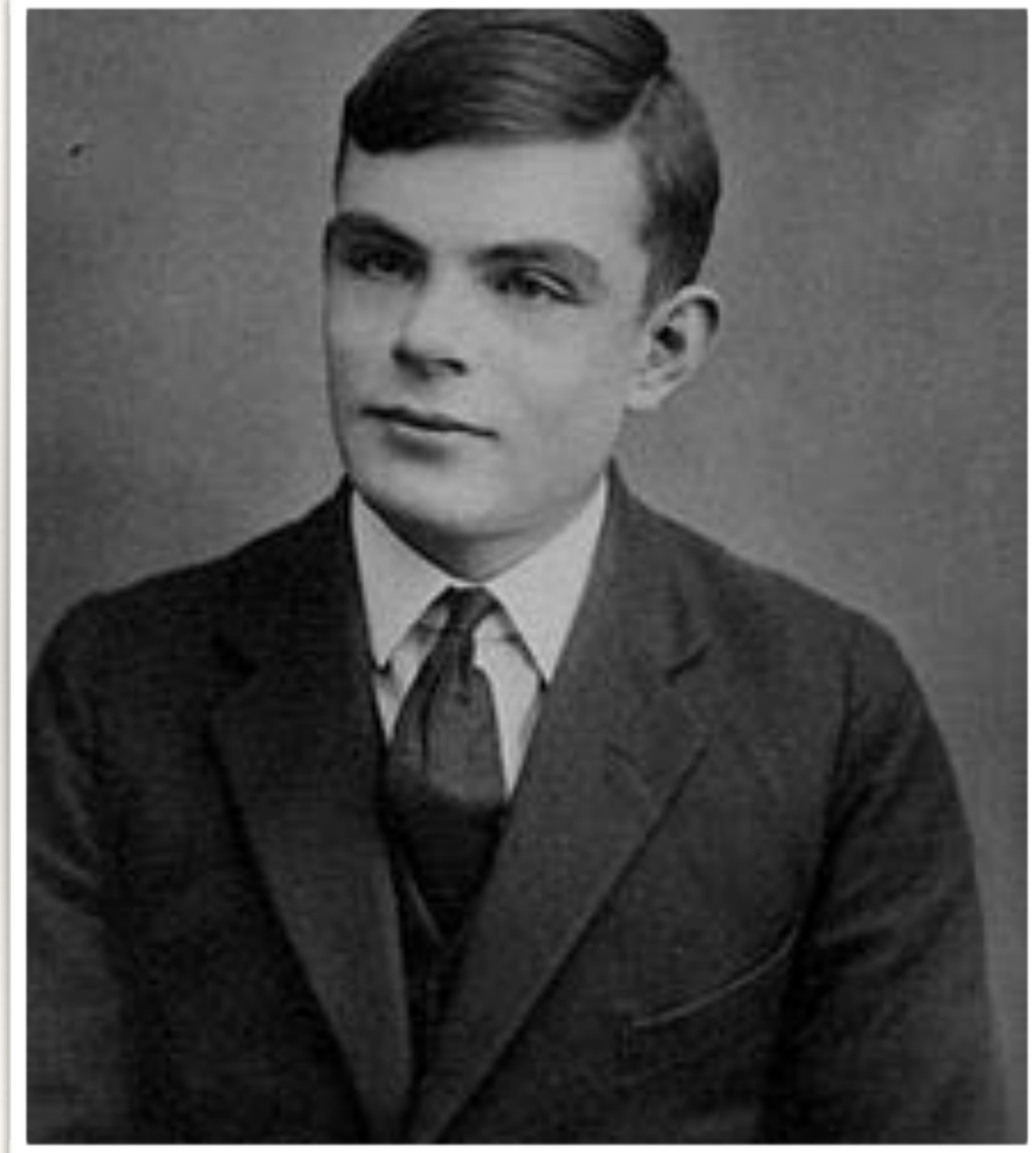
# Turing's 1936 paper

Opening sentence of Sec. 1

*We have said that the computable numbers are those whose decimals are calculable by finite means … the justification lies in the fact that the human memory is necessarily limited*

Also assumes that a finite volume can only contain finite information…

Also assumes locality: one cannot access two squares on the tape simultaneously…



# Turing did not understand “paper” ?

*Turing hoped that his abstracted-paper-tape model was so simple, so transparent and well defined, that it would not depend on any assumptions about physics that could conceivably be falsified, and therefore that it could become the basis of an abstract theory of computation that was independent of the underlying physics. ‘He thought,’ as Feynman once put it, ‘that he understood paper.’ But he was mistaken. Real, quantum-mechanical paper is wildly different from the abstract stuff that the Turing machine uses. The Turing machine is entirely classical, and does not allow for the possibility the paper might have different symbols written on it in different universes, and that those might interfere with one another.*

*(Deutsch 1985)*

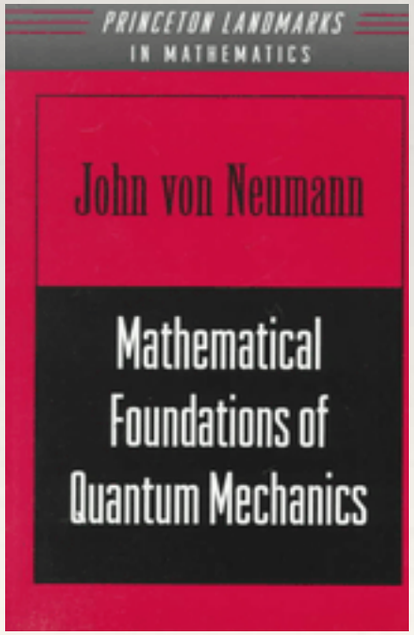
# Physics and Computation

# Physics

- Most accurate theory known to us is *Quantum Mechanics*

# Physics

- Most accurate theory known to us is *Quantum Mechanics*

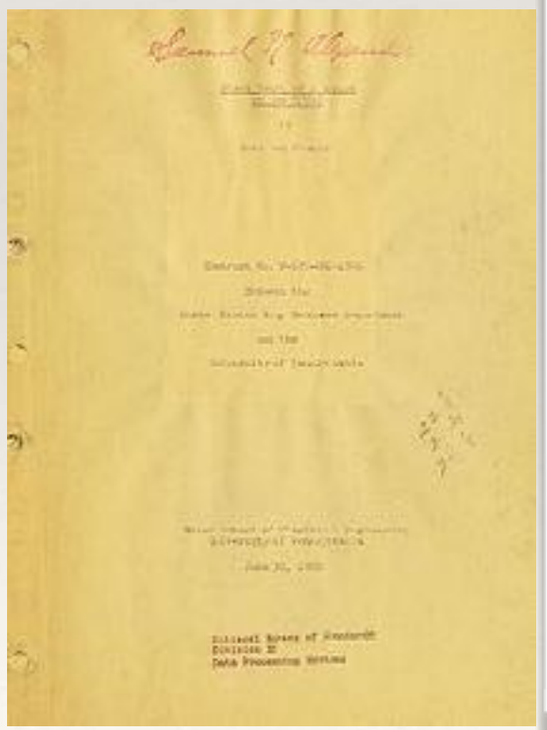



# Computation

- Universal model of computation is based on the Von Neumann architecture
- Equivalent to Turing machines, the $\lambda$-calculus, and many other models

# Computation

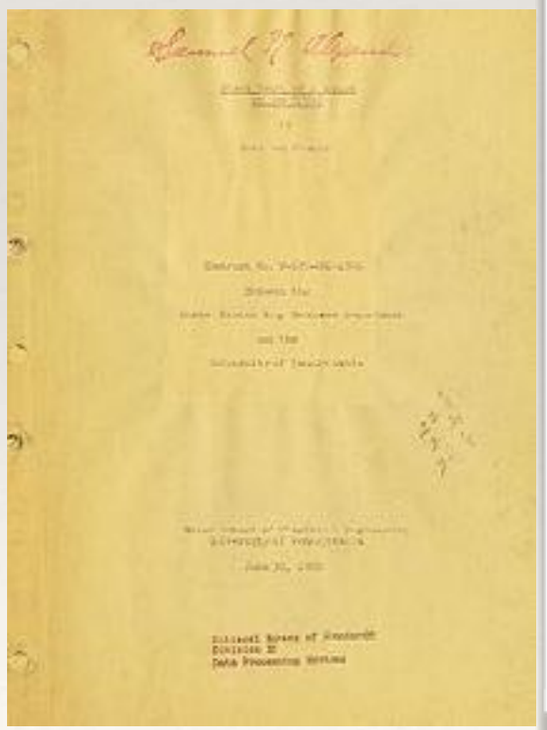
- Universal model of computation is based on the Von Neumann architecture
- Equivalent to Turing machines, the $\lambda$-calculus, and many other models



# Computation

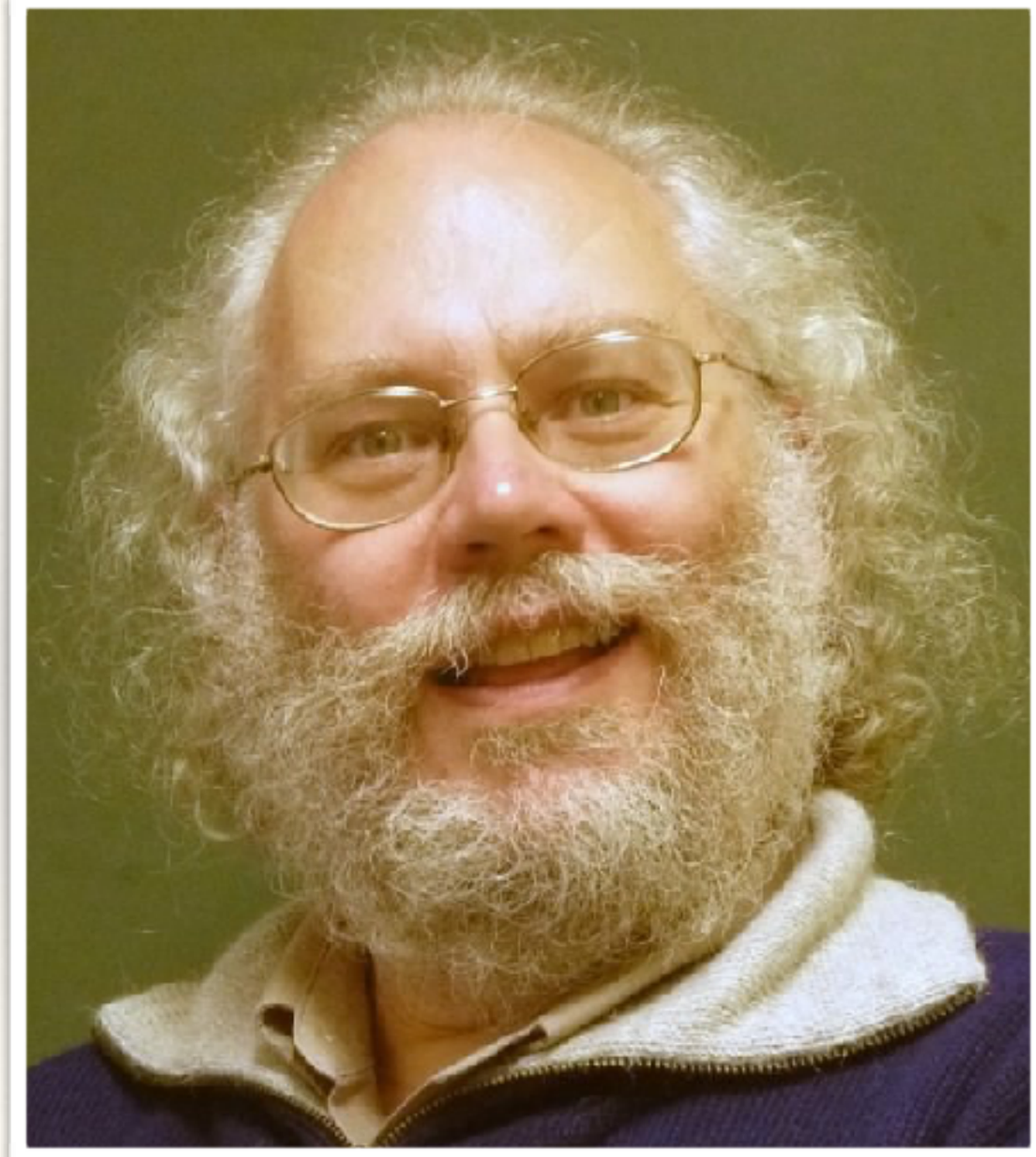
- Universal model of computation is based on the Von Neumann architecture
- Equivalent to Turing machines, the $\lambda$-calculus, and many other models

First Draft of a Report on the EDVAC
by John von Neumann,
Contract No. W-670-ORD-4926,
Between the United States Army Ordinance Department
and the University of Pennsylvania Moore School of Electrical Engineering
University of Pennsylvania
June 30, 1945



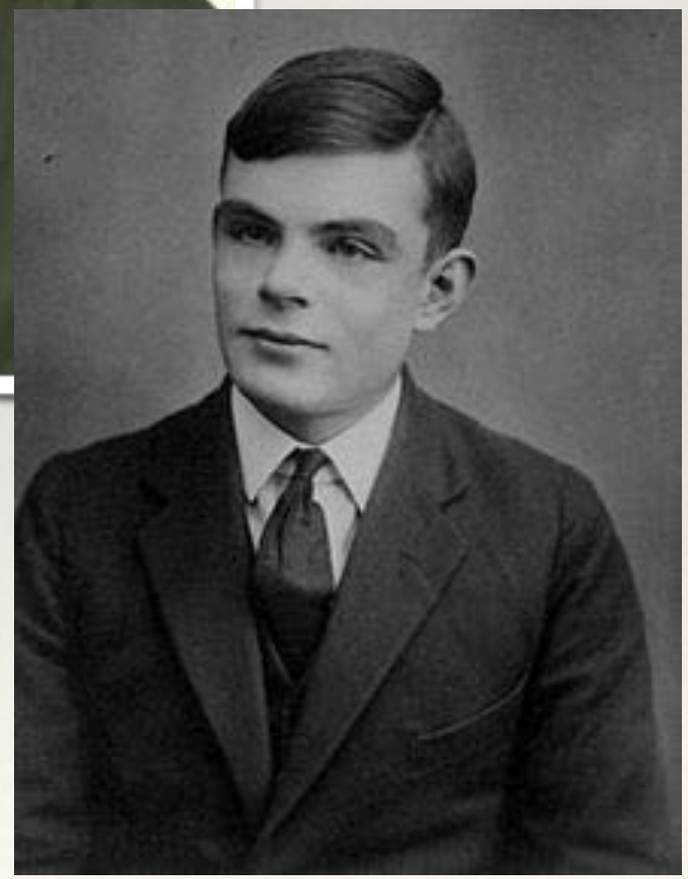
# Physics Worldview

- Quantum mechanics implies the existence of an efficient (polynomial) algorithm for factoring integers (Shor)
- RSA is not secure.



# Computer Science Worldview

- Extended Church-Turing Thesis: Everything that is computable in Nature is computable by a von Neumann machine —- *and is not exponentially faster.*
- Despite intense interest and effort, no one knows how to factor integers efficiently in that model.
- RSA is secure.



# Something is Wrong

- Either Shor's algorithm is not "natural". (Textbook quantum mechanics is wrong);
- or, Shor's algorithm is "natural" and there is no classical counterpart. (There are "natural" computing models that are exponentially faster than the Turing Machine);
- or, there is an efficient classical factoring algorithm. (Possible but surprising);
- At least one of these wild claims is true!!!

# Possibility I: Revise Physics

*The mathematician's vision of an unlimited sequence of totally reliable operations is unlikely to be implementable in this real universe.*

*But the real world is unlikely to supply us with unlimited memory or unlimited Turing machine tapes. Therefore, continuum mathematics is not executable, and physical laws which invoke that can not really be satisfactory. They are references to illusionary procedures.*

*(Landauer 1996 and 1999)*

# Possibility I: Revise Physics

*I want to talk about the possibility that there is to be an exact simulation, that the computer will do exactly the same as nature. If this is to be proved and the type of computer is as I've already explained, then it's going to be necessary that everything that happens in a finite volume of space and time would have to be exactly analyzable with a finite number of logical operations. The present theory of physics is not that way, apparently. It allows space to go down into infinitesimal distances, wavelengths to get infinitely great, terms to be summed in infinite order, and so forth; and therefore, if this proposition is right, physical law is wrong.*

*(Feynman 1981)*

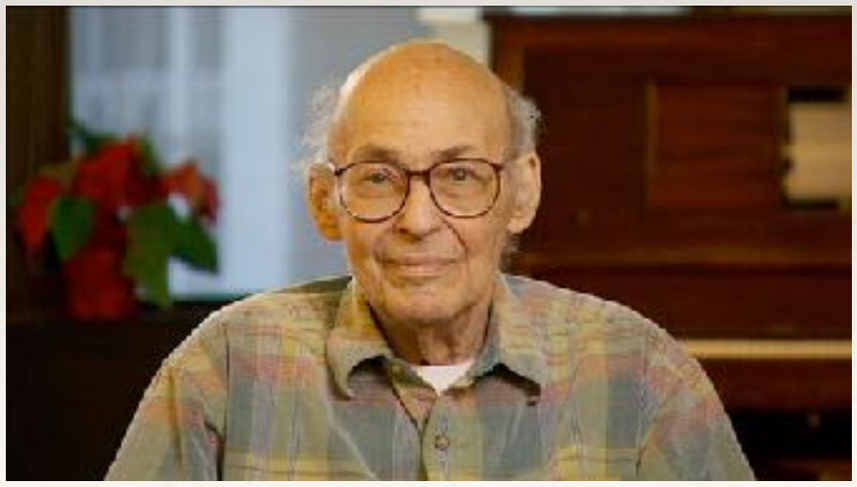
# Possibility II: Revise Computer Science

*Another thing that had been suggested early was that natural laws are reversible, but that computer rules are not. But this turned out to be false; the computer rules can be reversible, and it has been a very, very useful thing to notice and to discover that. This is a place where the relationship of physics and computation has turned itself the other way and told us something about the possibilities of computation. So this is an interesting subject because it tells us something about computer rules…*

*(Feynman 1981)*

# Possibility II: Revise Computer Science

*Ed Fredkin pursued the idea that information must be finite in density. One day, he announced that things must be even more simple than that. He said that he was going to assume that information itself is conserved. "You're out of you mind, Ed." I pronounced. "That's completely ridiculous. Nothing could happen in such a world. There couldn't even be logical gates. No decisions could ever be made." But when Fredkin gets one of his ideas, he's quite immune to objections like that; indeed, they fuel him with energy. Soon he went on to assume that information processing must also be reversible — and invented what's now called the Fredkin gate.*



*(Minsky 1999)*

# Possibility II': Revise Logic

*In other terms, what is so good in logic that quantum physics should obey? Can't we imagine that our conceptions about logic are wrong, so wrong that they are unable to cope with the quantum miracle? […] Instead of teaching logic to nature, it is more reasonable to learn from her. Instead of interpreting quantum into logic, we shall interpret logic into quantum.*

*(Girard 2007)*

# Logic, Physics, Computation

No matter where you start, you are led to these principles:

**Conservation of (physical) information**

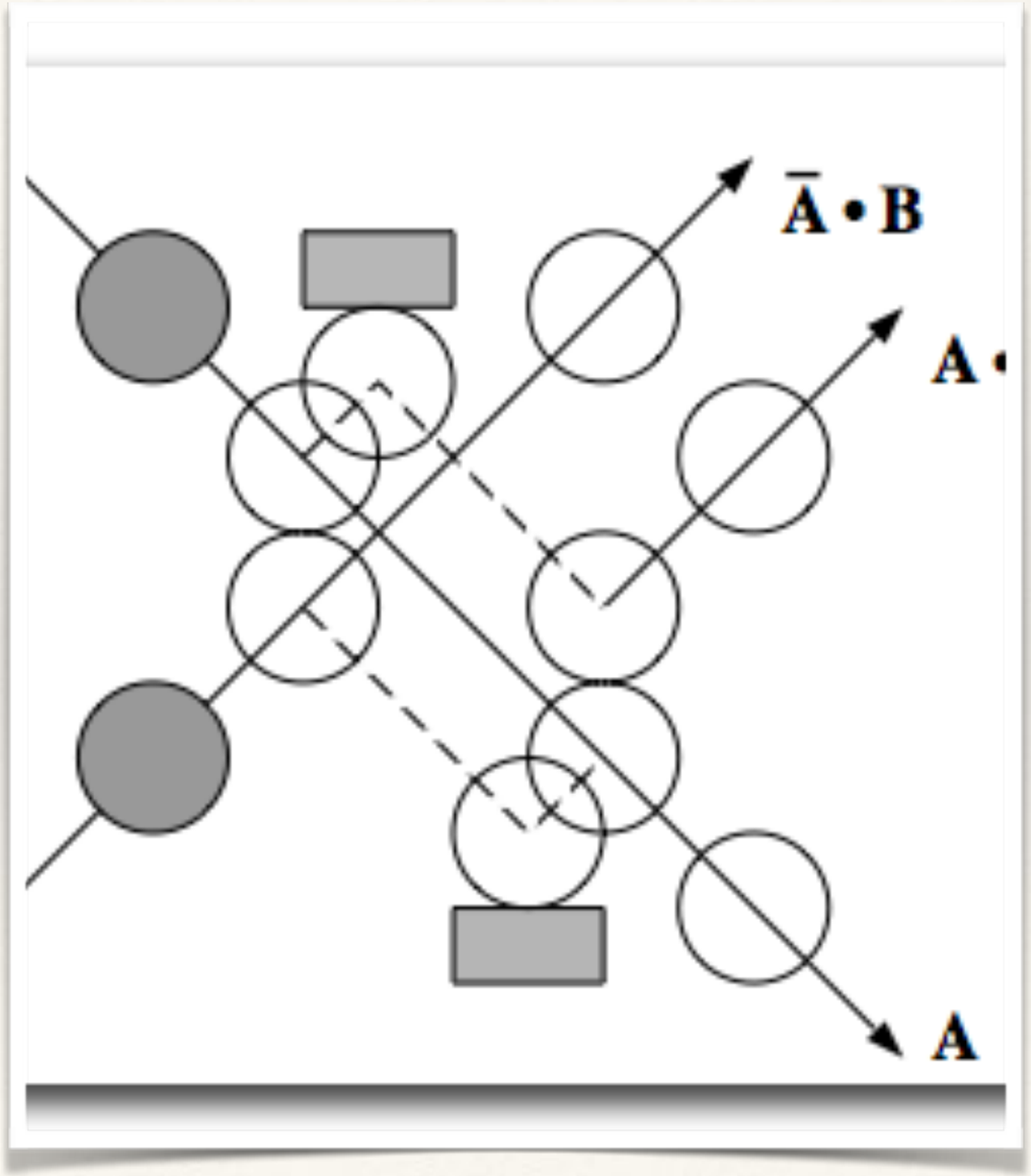
**No creation of (physical) information**

**No erasure of (physical) information**

**No duplication of (physical) information**

# Conservation of Information

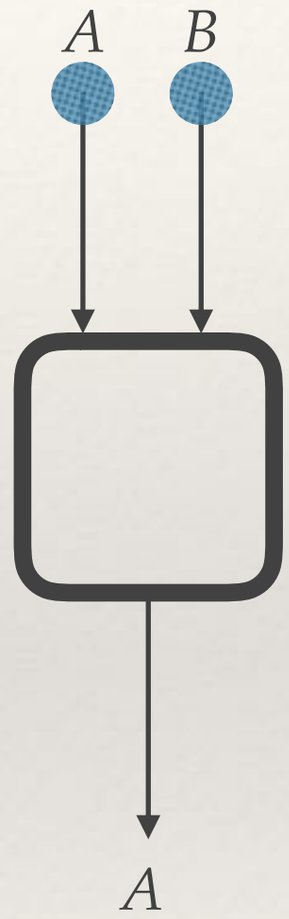
- Is it just a low-level (physical) concern, a curiosity, an esoteric model, or is it really foundational?
- We argue that, if taken seriously, it revolutionizes the theory and practice of computation, and even its logical foundations.
- This perspective has far reaching implications in many high-level applications



# Conservation of Information: Logical vs. Physical

None of the common foundational models of computation is based on conservation of information

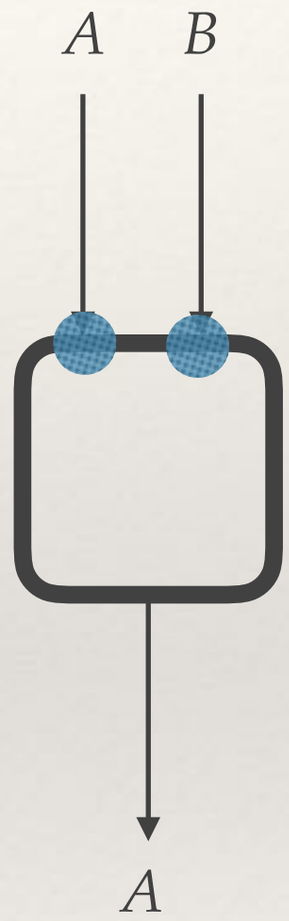
- Circuit model has AND, OR, etc gates that lose information;
- Turing Machine allows one to overwrite a cell losing information;
- $\lambda$-calculus allows functions that throw away their arguments losing information;
- etc etc etc



# Conservation of Information: Logical vs. Physical

None of the common foundational models of computation is based on conservation of information

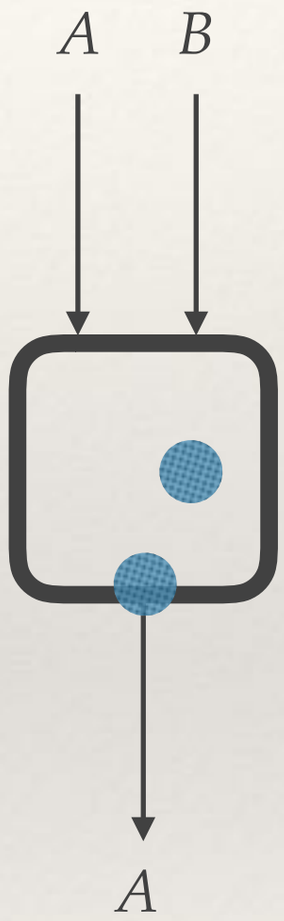
- Circuit model has AND, OR, etc gates that lose information;
- Turing Machine allows one to overwrite a cell losing information;
- $\lambda$-calculus allows functions that throw away their arguments losing information;
- etc etc etc



# Conservation of Information: Logical vs. Physical

None of the common foundational models of computation is based on conservation of information

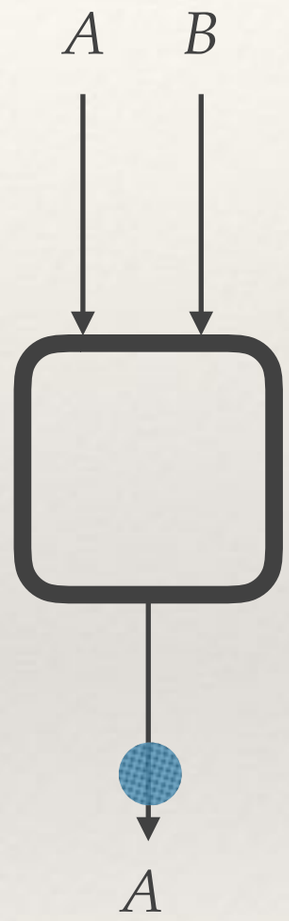
- Circuit model has AND, OR, etc gates that lose information;
- Turing Machine allows one to overwrite a cell losing information;
- $\lambda$-calculus allows functions that throw away their arguments losing information;
- etc etc etc



# Conservation of Information: Logical vs. Physical

None of the common foundational models of computation is based on conservation of information

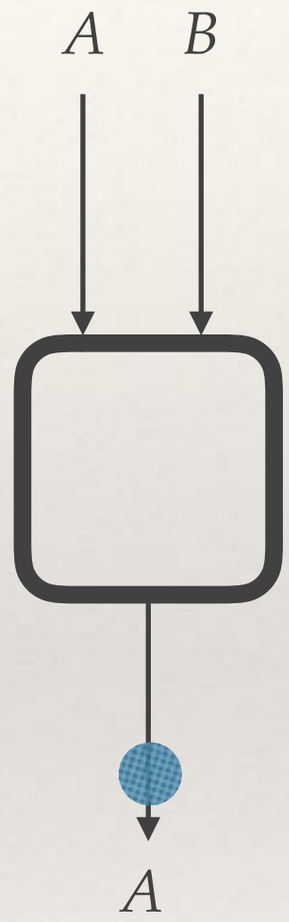
- Circuit model has AND, OR, etc gates that lose information;
- Turing Machine allows one to overwrite a cell losing information;
- $\lambda$-calculus allows functions that throw away their arguments losing information;
- etc etc etc



# Conservation of Information: Logical vs. Physical

None of the common foundational models of computation is based on conservation of information

- Circuit model has AND, OR, etc gates that lose information;
- Turing Machine allows one to overwrite a cell losing information;
- $\lambda$-calculus allows functions that throw away their arguments losing information;
- etc etc etc




***Laundauer's principle:***
***Erasure of information generates heat!!!***

# From Irreversible to Reversible

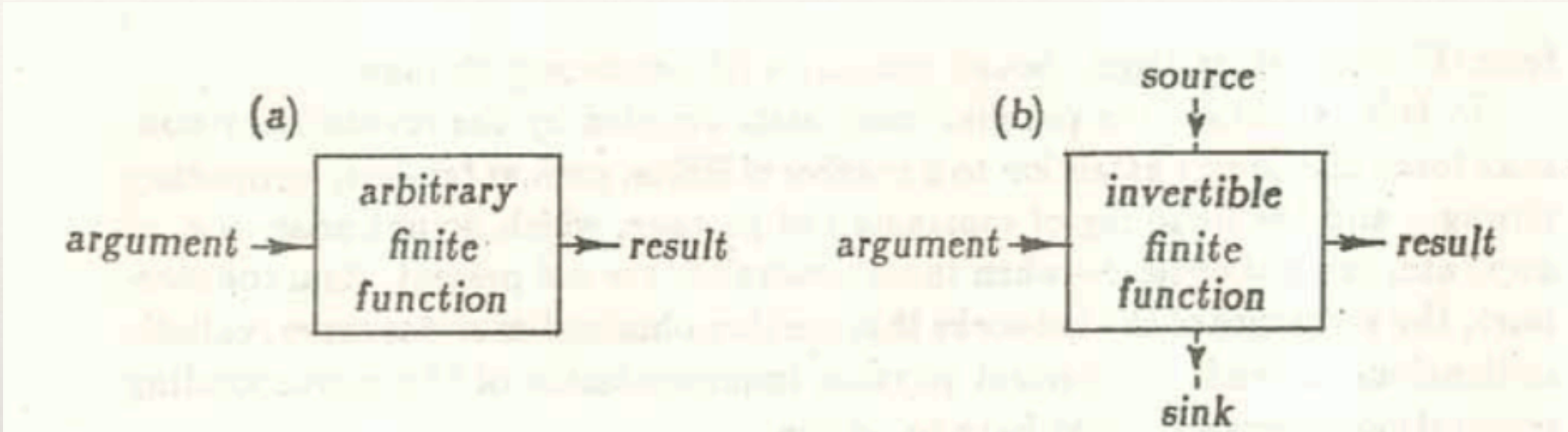
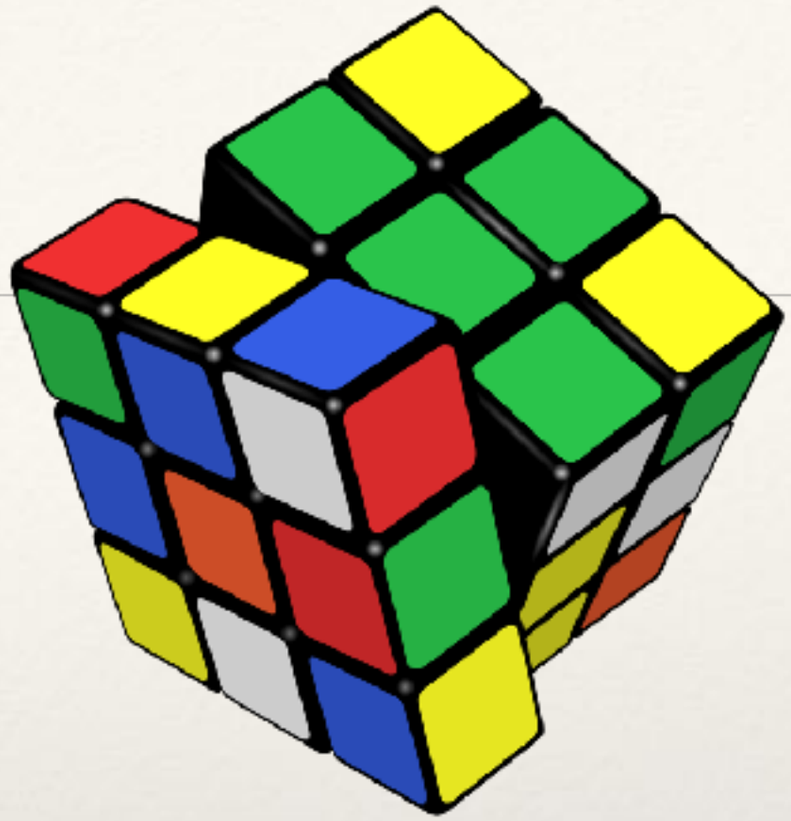


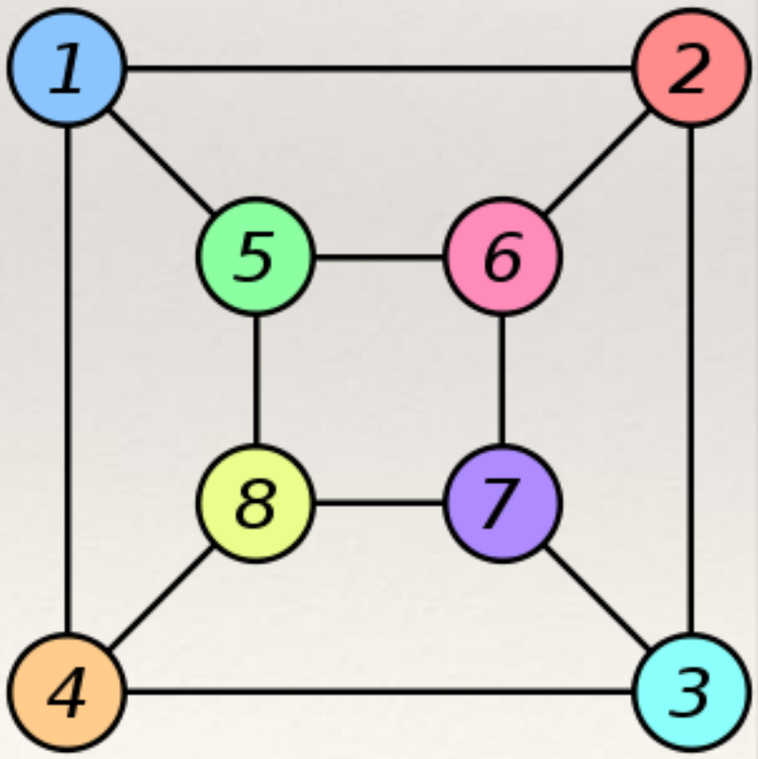

FIG. 4.2 *Any finite function (a) can be realized as an invertible finite function (b) having a number of auxiliary input lines which are fed with constants and a number of auxiliary output lines whose values are disregarded.*

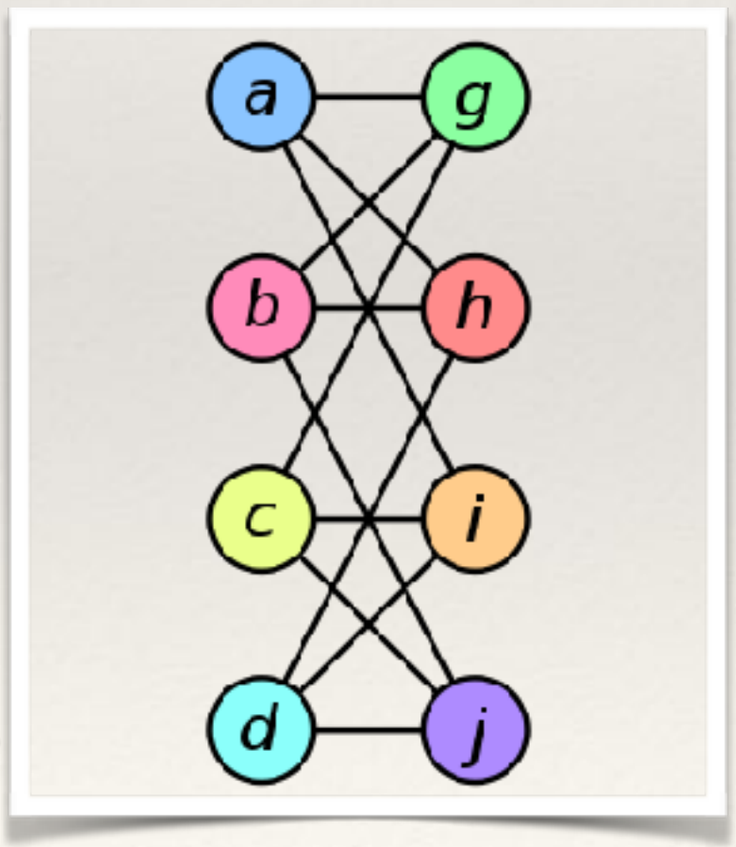
*(Toffoli 1980)*

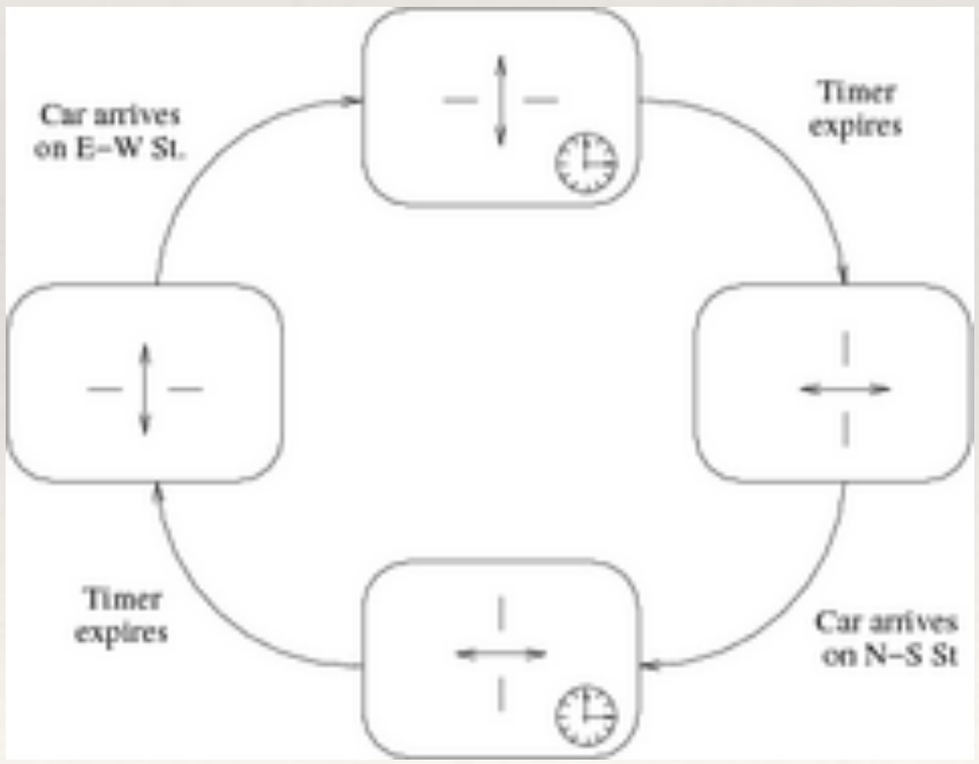
# Key Idea



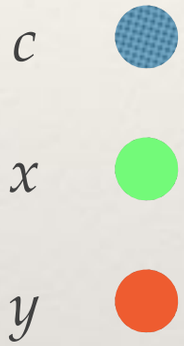
- All programs / proofs / deductions are isomorphisms / equivalences

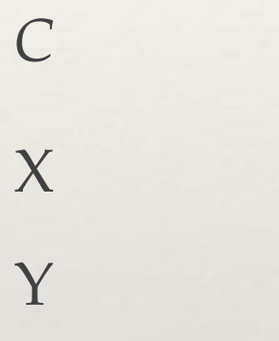


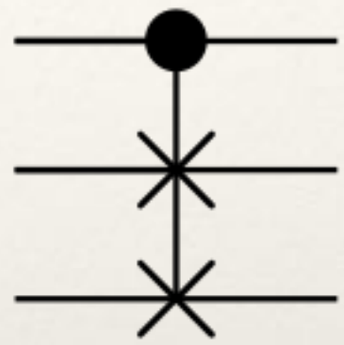



# Fredkin Gate

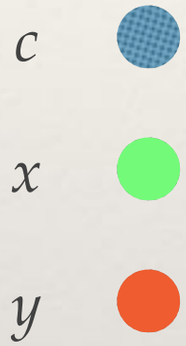


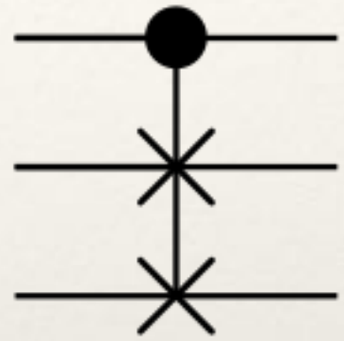
# Fredkin Gate

$c = false$

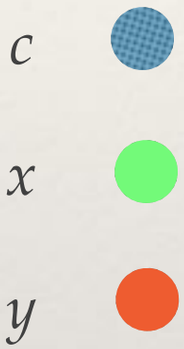



# Fredkin Gate

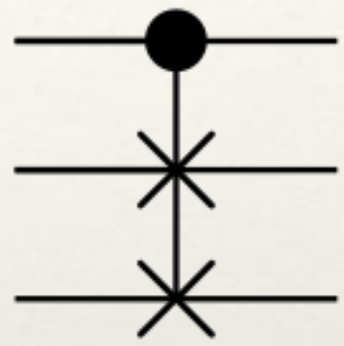
$c = false$



# Fredkin Gate

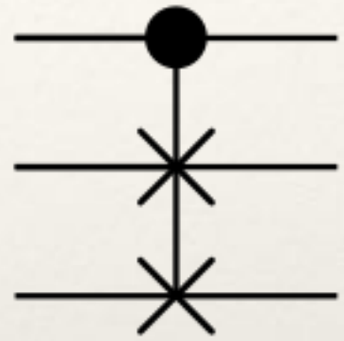




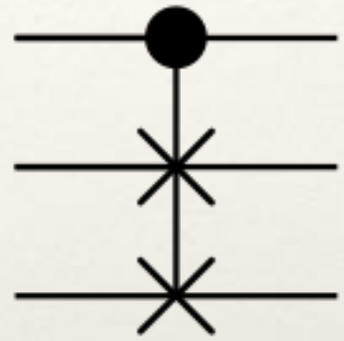


# Fredkin Gate

$c = true$

# Fredkin Gate



$c = true$


If $y = false$, then $Y = c$ AND $x$
If $x = false$, $y = true$, then $y =$ NOT $c$
If $y = true$, then $X = c$ OR $x$

# Conservation of Information in A Programming Language

# Reversibility

- Many ad hoc constructions: database transactions, data provenance, checkpoints, logs, rollback recovery, version control systems, backtracking, continuations, etc.
- Many reversible languages designed from different perspectives
- Special recognition to Henry Baker

NREVERSAL of Fortune[1] —
The Thermodynamics of Garbage Collection

1992

Henry G. Baker

Nimble Computer Corporation
16231 Meadow Ridge Way
Encino, California 91436
U.S.A.
(818) 501-4956 (818) 986-1360 (FAX)

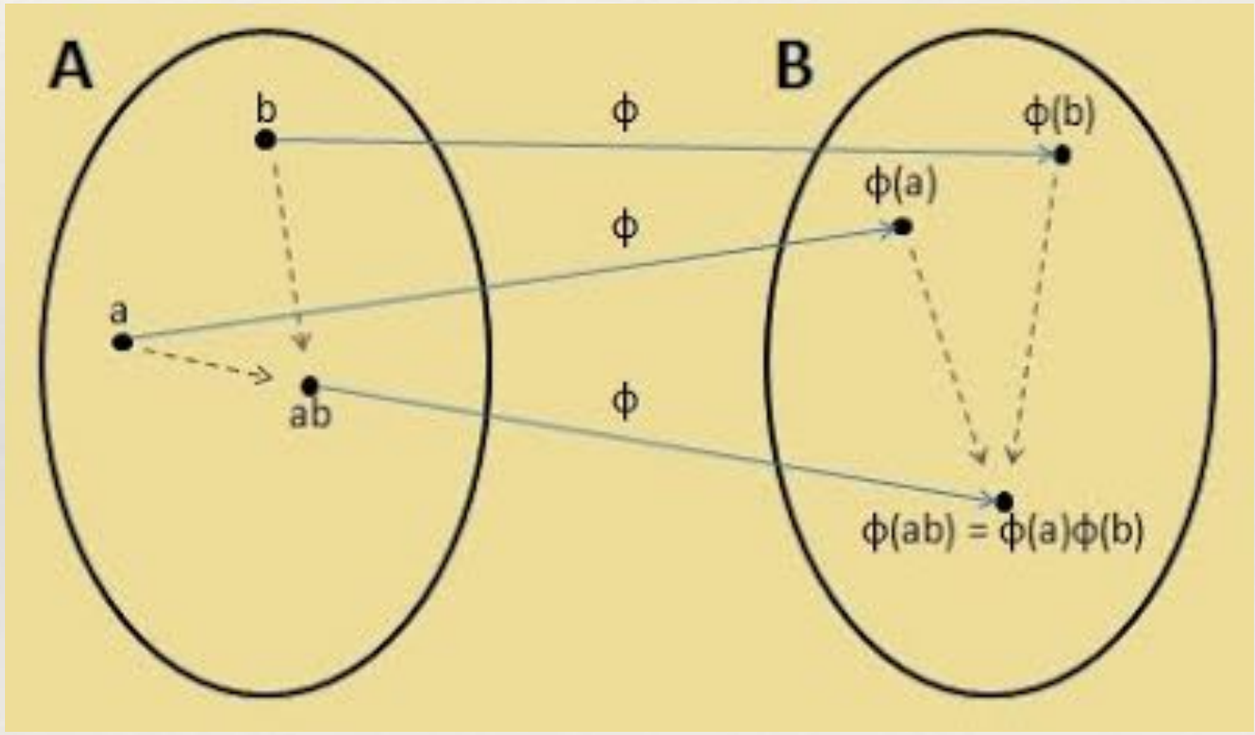
**Abstract**

The need to *reverse* a computation arises in many contexts—debugging, editor undoing, optimistic concurrency undoing, speculative computation undoing, trace scheduling, exception handling undoing, database recovery, optimistic discrete event simulations, subjunctive computing, etc. The need to *analyze* a reversed computation arises in the context of static analysis—liveness analysis, strictness analysis, type inference, etc. Traditional means for restoring a computation to a previous state involve checkpoints; checkpoints require time to copy, as well as space to store, the copied material. Traditional reverse abstract interpretation produces relatively poor information due to its inability to guess the previous values of assigned-to variables.

We propose an abstract computer model and a programming language—Ψ-Lisp—whose primitive operations are injective and hence reversible, thus allowing arbitrary undoing without the overheads of checkpointing. Such a computer can be built from reversible conservative

# Type Theoretic Approach: Type Isomorphisms

- Some problems are naturally reversible.
- Those that are not can be embedded in reversible problems.
- When it comes to writing programs to solve these problems, all is needed is a language for expressing equivalences.
- Technically a language that is sound and complete with respect to isomorphisms between types.



# Sound and Complete Type Isomorphisms (for finite types)

$$
\begin{array}{rcll}
0 + b & \rightleftharpoons & b & \textit{identity for } + \\
b_1 + b_2 & \rightleftharpoons & b_2 + b_1 & \textit{commutativity for } + \\
b_1 + (b_2 + b_3) & \rightleftharpoons & (b_1 + b_2) + b_3 & \textit{associativity for } +
\end{array}
$$

$$
\begin{array}{rcll}
1 \times b & \rightleftharpoons & b & \textit{identity for } \times \\
b_1 \times b_2 & \rightleftharpoons & b_2 \times b_1 & \textit{commutativity for } \times \\
b_1 \times (b_2 \times b_3) & \rightleftharpoons & (b_1 \times b_2) \times b_3 & \textit{associativity for } \times
\end{array}
$$

$$
\begin{array}{rcll}
0 \times b & \rightleftharpoons & 0 & \textit{distribute over } 0 \\
(b_1 + b_2) \times b_3 & \rightleftharpoons & (b_1 \times b_3) + (b_2 \times b_3) & \textit{distribute over } +
\end{array}
$$

$$
\frac{}{b_1 \rightleftharpoons b_1} \quad \frac{b_1 \rightleftharpoons b_2}{b_2 \rightleftharpoons b_1} \quad \frac{b_1 \rightleftharpoons b_2 \quad b_2 \rightleftharpoons b_3}{b_1 \rightleftharpoons b_3}
$$

$$
\frac{b_1 \rightleftharpoons b_3 \quad b_2 \rightleftharpoons b_4}{(b_1 + b_2) \rightleftharpoons (b_3 + b_4)} \quad \frac{b_1 \rightleftharpoons b_3 \quad b_2 \rightleftharpoons b_4}{(b_1 \times b_2) \rightleftharpoons (b_3 \times b_4)}
$$

# Name the Isomorphisms

$$
\begin{array}{rrcll}
\mathit{zeroe}: & 0 + b & \rightleftharpoons & b & : \mathit{zeroi} \\
\mathit{swap}^{+}: & b_1 + b_2 & \rightleftharpoons & b_2 + b_1 & : \mathit{swap}^{+} \\
\mathit{assocl}^{+}: & b_1 + (b_2 + b_3) & \rightleftharpoons & (b_1 + b_2) + b_3 & : \mathit{assocr}^{+} \\
 & & & & \\
\mathit{unite}: & 1 \times b & \rightleftharpoons & b & : \mathit{uniti} \\
\mathit{swap}^{\times}: & b_1 \times b_2 & \rightleftharpoons & b_2 \times b_1 & : \mathit{swap}^{\times} \\
\mathit{assocl}^{\times}: & b_1 \times (b_2 \times b_3) & \rightleftharpoons & (b_1 \times b_2) \times b_3 & : \mathit{assocr}^{\times} \\
 & & & & \\
\mathit{distrib}_0: & 0 \times b & \rightleftharpoons & 0 & : \mathit{factor}_0 \\
\mathit{distrib}: & (b_1 + b_2) \times b_3 & \rightleftharpoons & (b_1 \times b_3) + (b_2 \times b_3) & : \mathit{factor}
\end{array}
$$

$$
\frac{}{\mathit{id} : b \rightleftharpoons b} \qquad \frac{c : b_1 \rightleftharpoons b_2}{\mathit{sym}\ c : b_2 \rightleftharpoons b_1} \qquad \frac{c_1 : b_1 \rightleftharpoons b_2 \quad c_2 : b_2 \rightleftharpoons b_3}{(c_1 \fatsemi c_2) : b_1 \rightleftharpoons b_3}
$$

$$
\frac{c_1 : b_1 \rightleftharpoons b_3 \quad c_2 : b_2 \rightleftharpoons b_4}{(c_1 + c_2) : (b_1 + b_2) \rightleftharpoons (b_3 + b_4)} \qquad \frac{c_1 : b_1 \rightleftharpoons b_3 \quad c_2 : b_2 \rightleftharpoons b_4}{(c_1 \times c_2) : (b_1 \times b_2) \rightleftharpoons (b_3 \times b_4)}
$$

# Example

The type isomorphism:

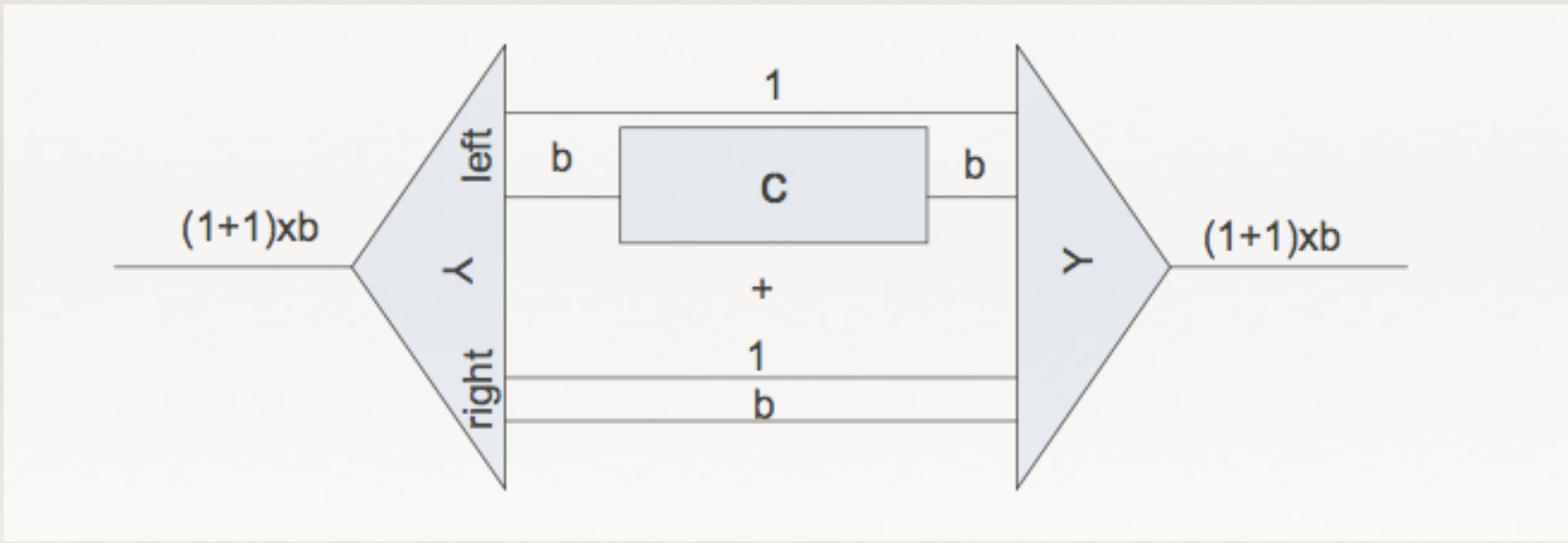
$$
\begin{aligned}
 & (1+1)\times((1+1)\times b) \\
= \quad & (1+1)\times((1\times b)+(1\times b)) \\
= \quad & (1+1)\times(b+b) \\
= \quad & (1\times(b+b))+(1\times(b+b)) \\
= \quad & (b+b)+(b+b)
\end{aligned}
$$

corresponds to the program:

$$(id \times (distrib \fatsemi (unite \times unite))) \fatsemi (distrib \fatsemi (unite \times unite))$$

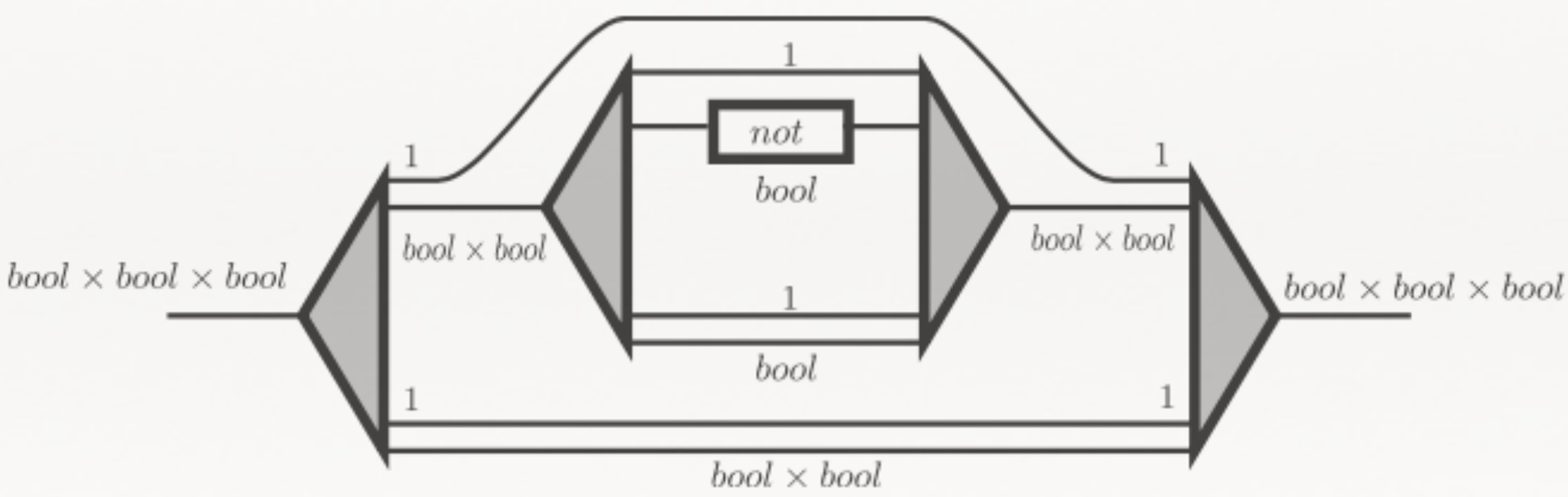
# Conditionals

$$\mathbf{if}_c : bool \times b \rightleftharpoons bool \times b$$
$$\mathbf{if}_c = distrib \fatsemi ((id \times c) + id) \fatsemi factor$$



# Toffoli and Fredkin Gates

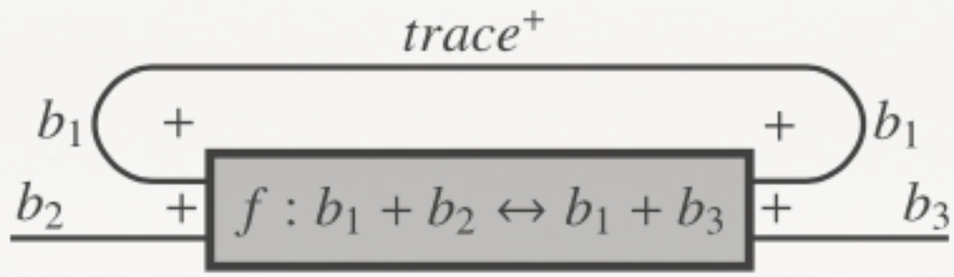
- Fredkin gate: $\mathbf{if}_{swap^{\times}}$
- Toffoli gate: $\mathbf{if}_{\mathbf{if}_{swap^{+}}}$



# Recursion

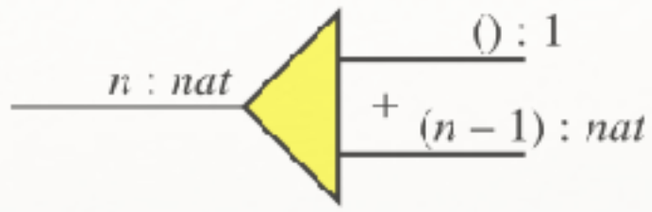
- Add recursive types (so that we can now define natural numbers, lists, trees, etc.)

- Add categorical trace (the categorical abstraction of feedback, looping, and recursion)

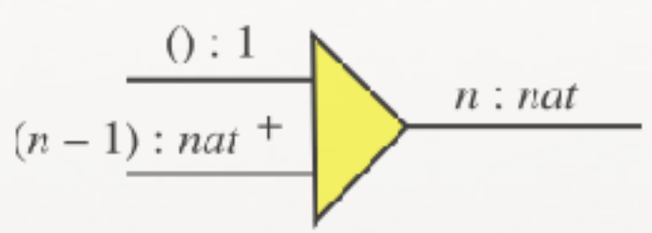


# Numbers

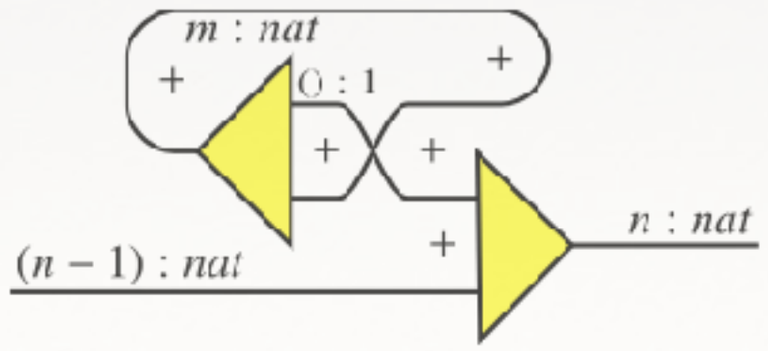
- Unfolding a natural number:



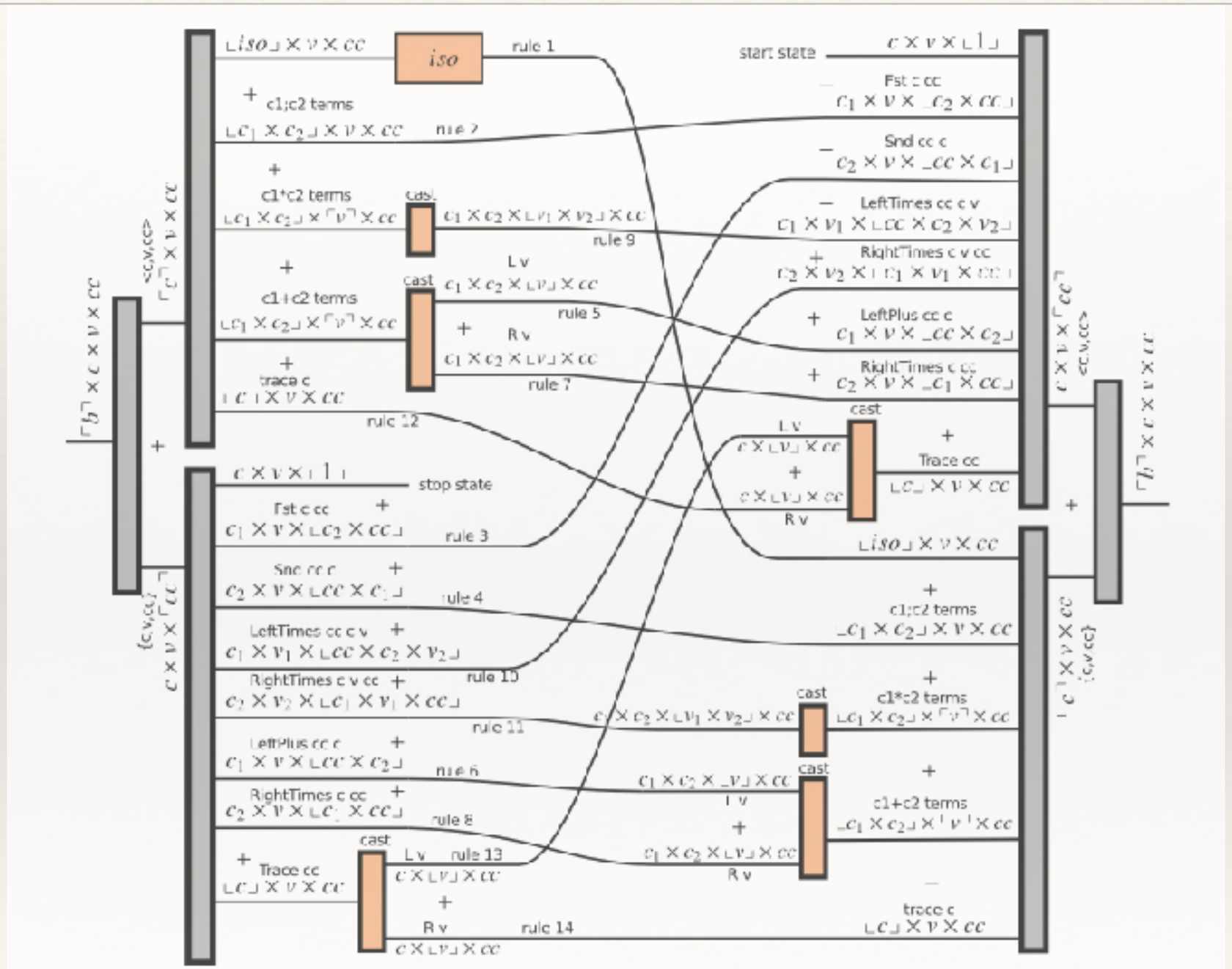

- Folding a natural number:




- add1/sub1 (partial isomorphisms)



# A meta-circular interpreter



# Curry-Howard

A tautology of propositional logic $\tau \times \tau \Leftrightarrow \tau$

When translated to type theory, only valid if we only care about whether both types are inhabited

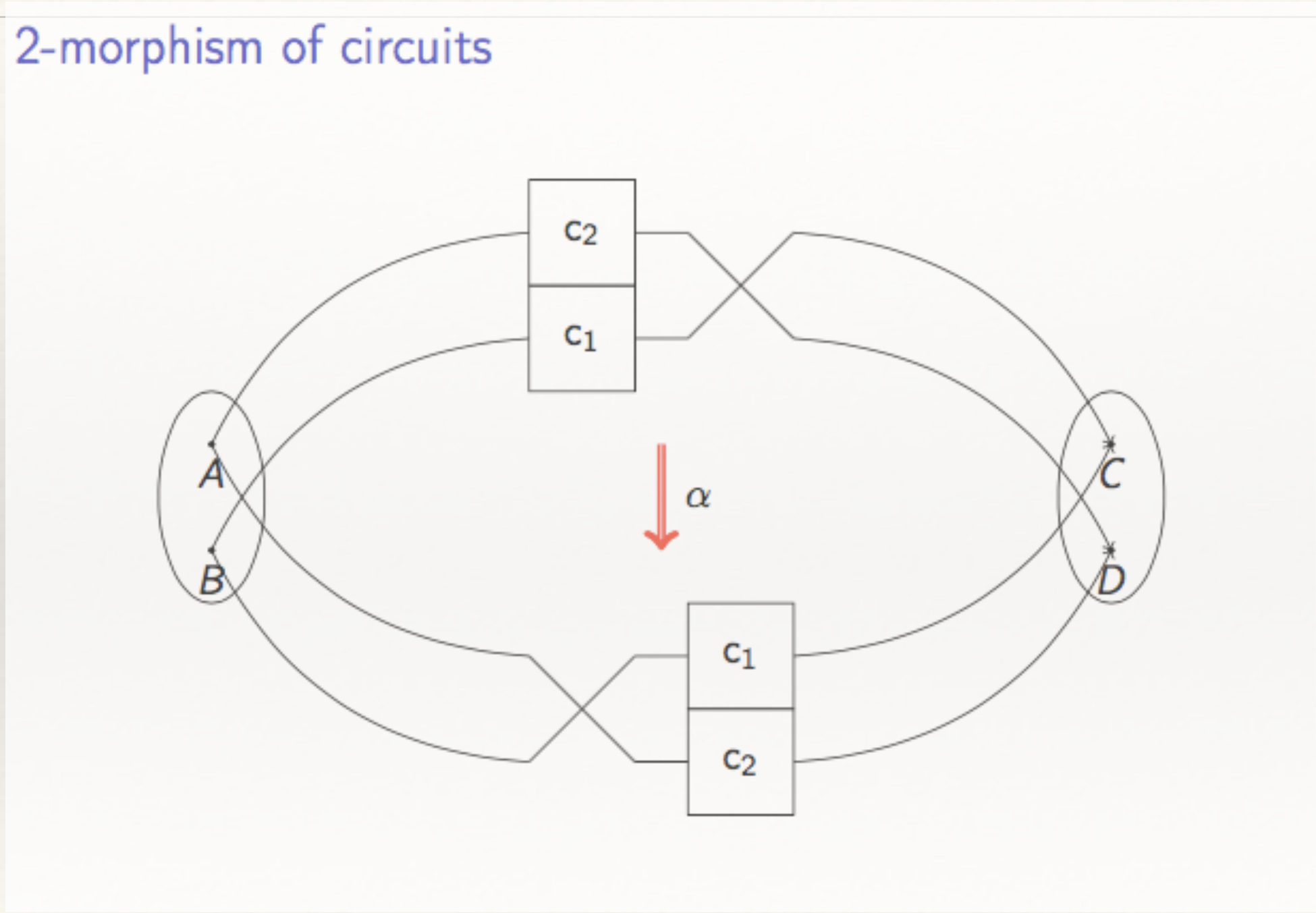
Type isomorphism is a more refined relation on types that clearly invalidates the “tautology”; gives rise to a new linear-like logic

# Reversible Logic

$$\dfrac{\dfrac{}{A \otimes B \sim B \otimes A} \quad \dfrac{\dfrac{}{B \vdash B}\, id \quad \dfrac{}{A \vdash A}\, id}{B \otimes A \vdash B \times A}\, \times R}{\dfrac{A \otimes B \vdash B \times A}{A \times B \vdash B \times A}\, \times L}\, struct$$

**Theorem 2.4** (Logical reversibility). *If* $\Gamma \vdash A$, *then* $A \vdash \ulcorner \Gamma \urcorner$.

# Relating Multiple Isomorphisms



# Reversible Program Transformations

## Π level 2

Let $c_1 : t_1 \leftrightarrow t_2$, $c_2 : t_2 \leftrightarrow t_3$, and $c_3 : t_3 \leftrightarrow t_4$:

$$c_1 \odot (c_2 \odot c_3) \Leftrightarrow (c_1 \odot c_2) \odot c_3$$

$$(c_1 \oplus (c_2 \oplus c_3)) \odot assocl_+ \Leftrightarrow assocl_+ \odot ((c_1 \oplus c_2) \oplus c_3)$$

$$(c_1 \otimes (c_2 \otimes c_3)) \odot assocl_* \Leftrightarrow assocl_* \odot ((c_1 \otimes c_2) \otimes c_3)$$

$$((c_1 \oplus c_2) \oplus c_3) \odot assocr_+ \Leftrightarrow assocr_+ \odot (c_1 \oplus (c_2 \oplus c_3))$$

$$((c_1 \otimes c_2) \otimes c_3) \odot assocr_* \Leftrightarrow assocr_* \odot (c_1 \otimes (c_2 \otimes c_3))$$

$$assocr_+ \odot assocr_+ \Leftrightarrow ((assocr_+ \oplus id) \odot assocr_+) \odot (id \oplus assocr_+)$$

$$assocr_* \odot assocr_* \Leftrightarrow ((assocr_* \otimes id) \odot assocr_*) \odot (id \otimes assocr_*)$$

$$((a \oplus b) \otimes c) \odot dist \Leftrightarrow dist \odot ((a \otimes c) \oplus (b \otimes c))$$

$$(a \otimes (b \oplus c)) \odot distl \Leftrightarrow distl \odot ((a \otimes b) \oplus (a \otimes c))$$

$$((a \otimes c) \oplus (b \otimes c)) \odot factor \Leftrightarrow factor \odot ((a \oplus b) \otimes c)$$

$$((a \otimes b) \oplus (a \otimes c)) \odot factorl \Leftrightarrow factorl \odot (a \otimes (b \oplus c))$$

Let $c, c_1, c_2, c_3 : t_1 \leftrightarrow t_2$ and $c', c'' : t_3 \leftrightarrow t_4$:

$$id \odot c \Leftrightarrow c \quad c \odot id \Leftrightarrow c \quad c \odot !c \Leftrightarrow id \quad !c \odot c \Leftrightarrow id$$

$$c \Leftrightarrow c \qquad \frac{c_1 \Leftrightarrow c_2 \quad c_2 \Leftrightarrow c_3}{c_1 \Leftrightarrow c_3} \qquad \frac{c_1 \Leftrightarrow c' \quad c_2 \Leftrightarrow c''}{c_1 \odot c_2 \Leftrightarrow c' \odot c''}$$

Let $c_0 : 0 \leftrightarrow 0$, $c_1 : 1 \leftrightarrow 1$, and $c : t_1 \leftrightarrow t_2$:

$$identl_+ \odot c \Leftrightarrow (c_0 \oplus c) \odot identl_+ \qquad identr_+ \odot (c_0 \oplus c) \Leftrightarrow c \odot identr_+$$

$$unite_+r \odot c \Leftrightarrow (c \oplus c_0) \odot unite_+r \qquad uniti_+r \odot (c \oplus c_0) \Leftrightarrow c \odot uniti_+r$$

$$identl_* \odot c \Leftrightarrow (c_1 \otimes c) \odot identl_* \qquad identr_* \odot (c_1 \otimes c) \Leftrightarrow c \odot identr_+$$

$$unite_*r \odot c \Leftrightarrow (c \otimes c_1) \odot unite_*r \qquad uniti_*r \odot (c \otimes c_1) \Leftrightarrow c \odot uniti_*r$$

$$identl_* \Leftrightarrow distl \odot (identl_* \oplus identl_*)$$

$$identl_+ \Leftrightarrow swap_+ \odot uniti_+r \qquad identl_* \Leftrightarrow swap_* \odot uniti_*r$$

$$(id \otimes swap_+) \odot distl \Leftrightarrow distl \odot swap_+$$

$$dist \odot (swap_* \oplus swap_*) \Leftrightarrow swap_* \odot distl$$

Let $c_1 : t_1 \leftrightarrow t_2$ and $c_2 : t_3 \leftrightarrow t_4$:

$$swap_+ \odot (c_1 \oplus c_2) \Leftrightarrow (c_2 \oplus c_1) \odot swap_+ \quad swap_* \odot (c_1 \otimes c_2) \Leftrightarrow (c_2 \otimes c_1) \odot swap_*$$

$$(assocr_+ \odot swap_+) \odot assocr_+ \Leftrightarrow ((swap_+ \oplus id) \odot assocr_+) \odot (id \oplus swap_+)$$

$$(assocl_+ \odot swap_+) \odot assocl_+ \Leftrightarrow ((id \oplus swap_+) \odot assocl_+) \odot (swap_+ \oplus id)$$

$$(assocr_* \odot swap_*) \odot assocr_* \Leftrightarrow ((swap_* \otimes id) \odot assocr_*) \odot (id \otimes swap_*)$$

$$(assocl_* \odot swap_*) \odot assocl_* \Leftrightarrow ((id \otimes swap_*) \odot assocl_*) \odot (swap_* \otimes id)$$

Let $c_1 : t_1 \leftrightarrow t_2$, $c_2 : t_3 \leftrightarrow t_4$, $c_3 : t_1 \leftrightarrow t_2$, and $c_4 : t_3 \leftrightarrow t_4$.
Let $a_1 : t_5 \leftrightarrow t_1$, $a_2 : t_6 \leftrightarrow t_2$, $a_3 : t_1 \leftrightarrow t_3$, and $a_4 : t_2 \leftrightarrow t_4$.

$$\frac{c_1 \Leftrightarrow c_3 \quad c_2 \Leftrightarrow c_4}{c_1 \oplus c_2 \Leftrightarrow c_3 \oplus c_4} \qquad \frac{c_1 \Leftrightarrow c_3 \quad c_2 \Leftrightarrow c_4}{c_1 \otimes c_2 \Leftrightarrow c_3 \otimes c_4}$$

$$id \oplus id \Leftrightarrow id \qquad id \otimes id \Leftrightarrow id$$

$$(a_1 \odot a_3) \oplus (a_2 \odot a_4) \Leftrightarrow (a_1 \oplus a_2) \odot (a_3 \oplus a_4)$$

$$(a_1 \odot a_3) \otimes (a_2 \odot a_4) \Leftrightarrow (a_1 \otimes a_2) \odot (a_3 \otimes a_4)$$

$$unite_+r \oplus id \Leftrightarrow assocr_+ \odot (id \oplus identl_+)$$

$$unite_*r \otimes id \Leftrightarrow assocr_* \odot (id \otimes identl_*)$$

Let $c : t_1 \leftrightarrow t_2$:

$$(c \otimes id) \odot absorbl \Leftrightarrow absorbl \odot id \quad (id \otimes c) \odot absorbr \Leftrightarrow absorbr \odot id$$

$$id \odot factorl_0 \Leftrightarrow factorl_0 \odot (id \otimes c) \quad id \odot factorzr \Leftrightarrow factorzr \odot (c \otimes id)$$

$$absorbr \Leftrightarrow absorbl$$

$$absorbr \Leftrightarrow (distl \odot (absorbr \oplus absorbr)) \odot identl_+$$

$$unite_*r \Leftrightarrow absorbr \qquad absorbl \Leftrightarrow swap_* \odot absorbr$$

$$absorbr \Leftrightarrow (assocl_* \odot (absorbr \otimes id)) \odot absorbr$$

$$(id \otimes absorbr) \odot absorbl \Leftrightarrow (assocl_* \odot (absorbl \otimes id)) \odot absorbr$$

$$id \otimes identl_+ \Leftrightarrow (distl \odot (absorbl \oplus id)) \odot identl_+$$

$$((assocl_+ \otimes id) \odot dist) \odot (dist \oplus id) \Leftrightarrow (dist \odot (id \oplus dist)) \odot assocl_+$$

$$assocl_* \odot distl \Leftrightarrow ((id \otimes distl) \odot distl) \odot (assocl_* \oplus assocl_*)$$

$$(distl \odot (dist \oplus dist)) \odot assocl_+ \Leftrightarrow dist \odot (distl \oplus distl) \odot assocl_+ \odot (assocr_+ \oplus id) \odot ((id \oplus swap_+) \oplus id) \odot (assocl_+ \oplus id)$$

# A Featherweight Version

- For concreteness, let's look at a "featherweight" version in full detail
- One type *Bool*
- Isomorphisms on that type (level-1 programs)
- Coherence between isomorphisms (level-2 programs)

# A Featherweight Reversible Language: Π2

$$\begin{array}{rcl} \tau & ::= & 2 \\ \\ v & ::= & false : 2 \\ & | & true : 2 \end{array}$$

# Programs / Combinators / Isomorphisms

$$
\begin{array}{lll}
c ::= & \texttt{id} & : \tau \leftrightarrow \tau \\
\mid & \texttt{swap} & : 2 \leftrightarrow 2 \\
\mid & ! & : (\tau_1 \leftrightarrow \tau_2) \to (\tau_2 \leftrightarrow \tau_1) \\
\mid & \circ & : (\tau_1 \leftrightarrow \tau_2) \to (\tau_2 \leftrightarrow \tau_3) \to (\tau_1 \leftrightarrow \tau_3)
\end{array}
$$

# Level 2 Programs / Level 2 Isomorphisms

$$
\begin{aligned}
\alpha ::= \quad & \mathtt{id} : c \Leftrightarrow c \\
\mid \quad & idl : \mathtt{id} \circ c \Leftrightarrow c \\
\mid \quad & idr : c \circ \mathtt{id} \Leftrightarrow c \\
\mid \quad & invl : c \circ\, !\, c \Leftrightarrow \mathtt{id} \\
\mid \quad & invr : !\, c \circ c \Leftrightarrow \mathtt{id} \\
\mid \quad & \rho : \mathtt{swap} \circ \mathtt{swap} \Leftrightarrow \mathtt{id} \\
\mid \quad & assoc : (c_1 \circ c_2) \circ c_3 \Leftrightarrow c_1 \circ (c_2 \circ c_3) \\
\mid \quad & \odot : (c_1 \Leftrightarrow c_1') \to (c_2 \Leftrightarrow c_2') \to (c_1 \circ c_2 \Leftrightarrow c_1' \circ c_2') \\
\mid \quad & !! : (c_1 \Leftrightarrow c_2) \to (c_2 \Leftrightarrow c_1) \\
\mid \quad & \bullet : (c_1 \Leftrightarrow c_2) \to (c_2 \Leftrightarrow c_3) \to (c_1 \Leftrightarrow c_3)
\end{aligned}
$$

# Π2 Theorems

Several simple theorems regarding reversibility (omitted)

Most important theorem:

**Canonical Forms: Given a level-1 program *p*, the level-2 transformations are complete: they can transform *p* to either `*id* or `*not***

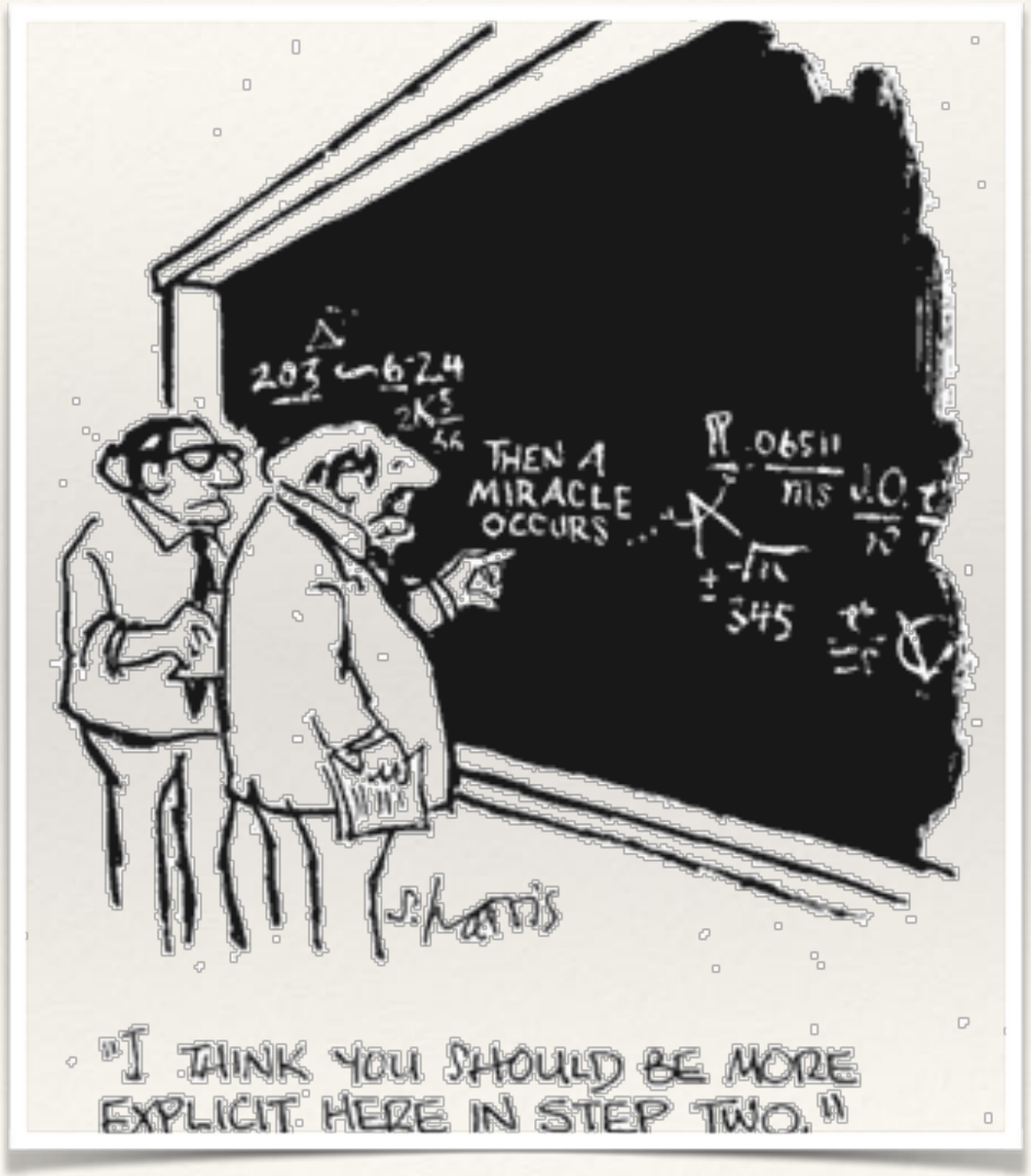
```
cmpl2-lem : (p : `2 ↔₁ `2) → (p ↔₂ `id) + (p ↔₂ `not)
```

# Homotopy Type Theory

# Is Mathematics Immune to this Physics Perspective?

# Equality

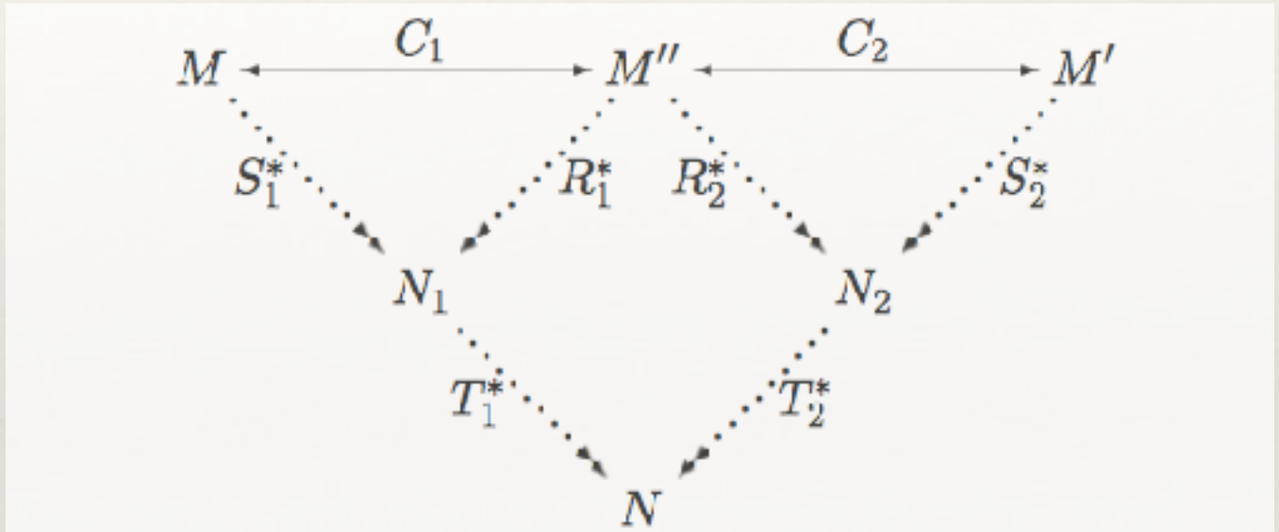
- In mathematics, we often use the equality sign without *explicit* justification.
- Sometimes this is innocent: 2+2 = 2+2 has an *"obvious"* justification.
- Or does it? 2+2 = 2+2 has (at least) two unrelated justifications:
- the relation = is reflexive
- the operation + is commutative



# Equality of Programs in a Computational Setting

In the $\lambda$-calculus $M = M'$ implies
$M \to^* N_1 \to^* N \to^{-1*} N_2 \to^{-1*} M'$

- An *operational* understanding of equality based on a notion of reduction
- Take compatible, reflexive, symmetric, and transitive closure of reduction to define equality
- Need Church-Rosser theorem to show that 1 is ***not equal*** to 2



# Equality of Types in a Computational Setting

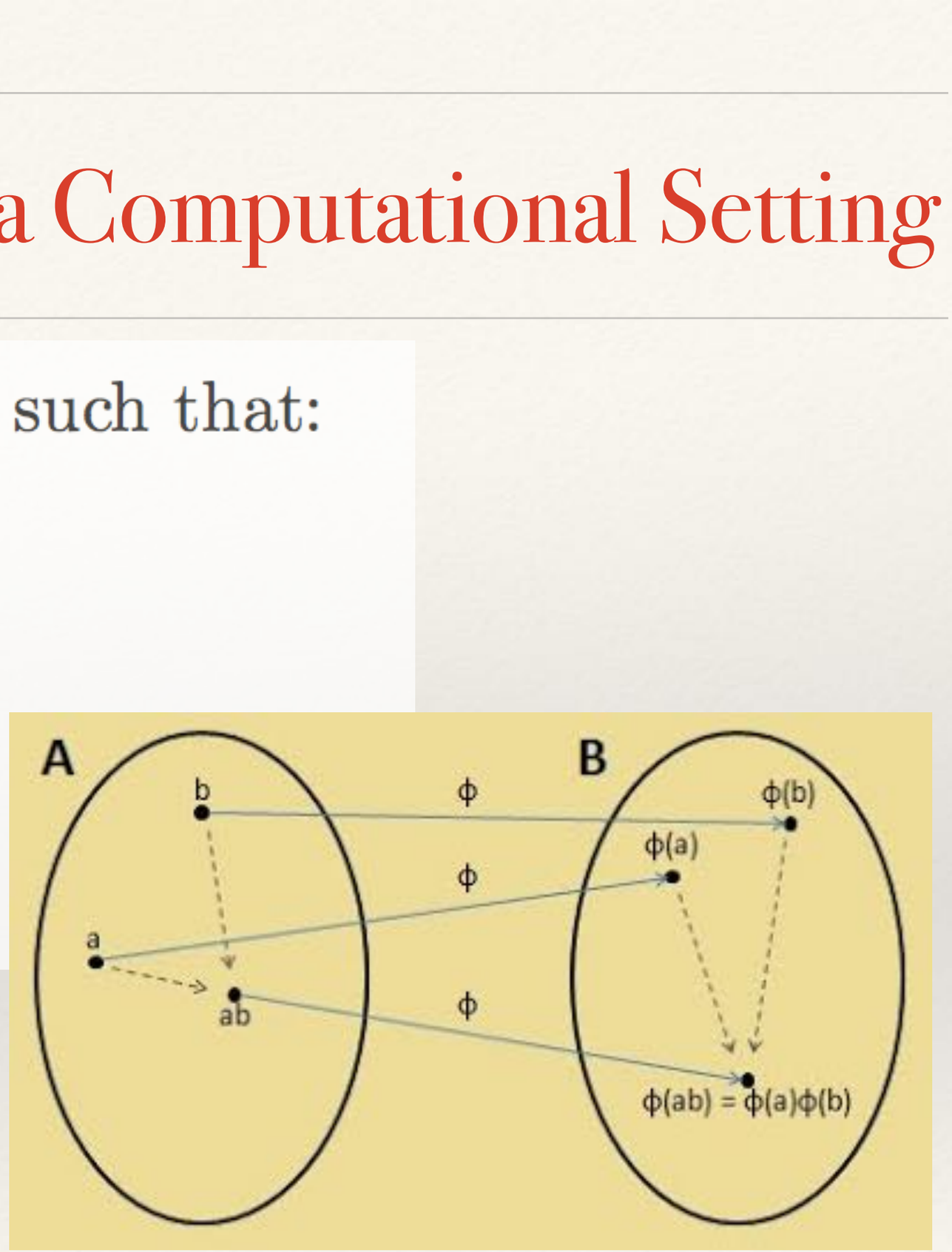
- Given an "interesting" language of types (polymorphism, dependent types, etc.)
- We have an *operational* understanding of type equality

$$\textsc{Type-reduction}\ \frac{}{(\Lambda X.\, e)\,[\tau] \to e\{\tau/X\}}$$

# Equality of Types in a Computational Setting

$A \simeq B$ if exists $f$ and $g$ such that:

- $f : A \to B$
- $g : B \to A$
- $g \circ f = id_A$
- $f \circ g = id_B$



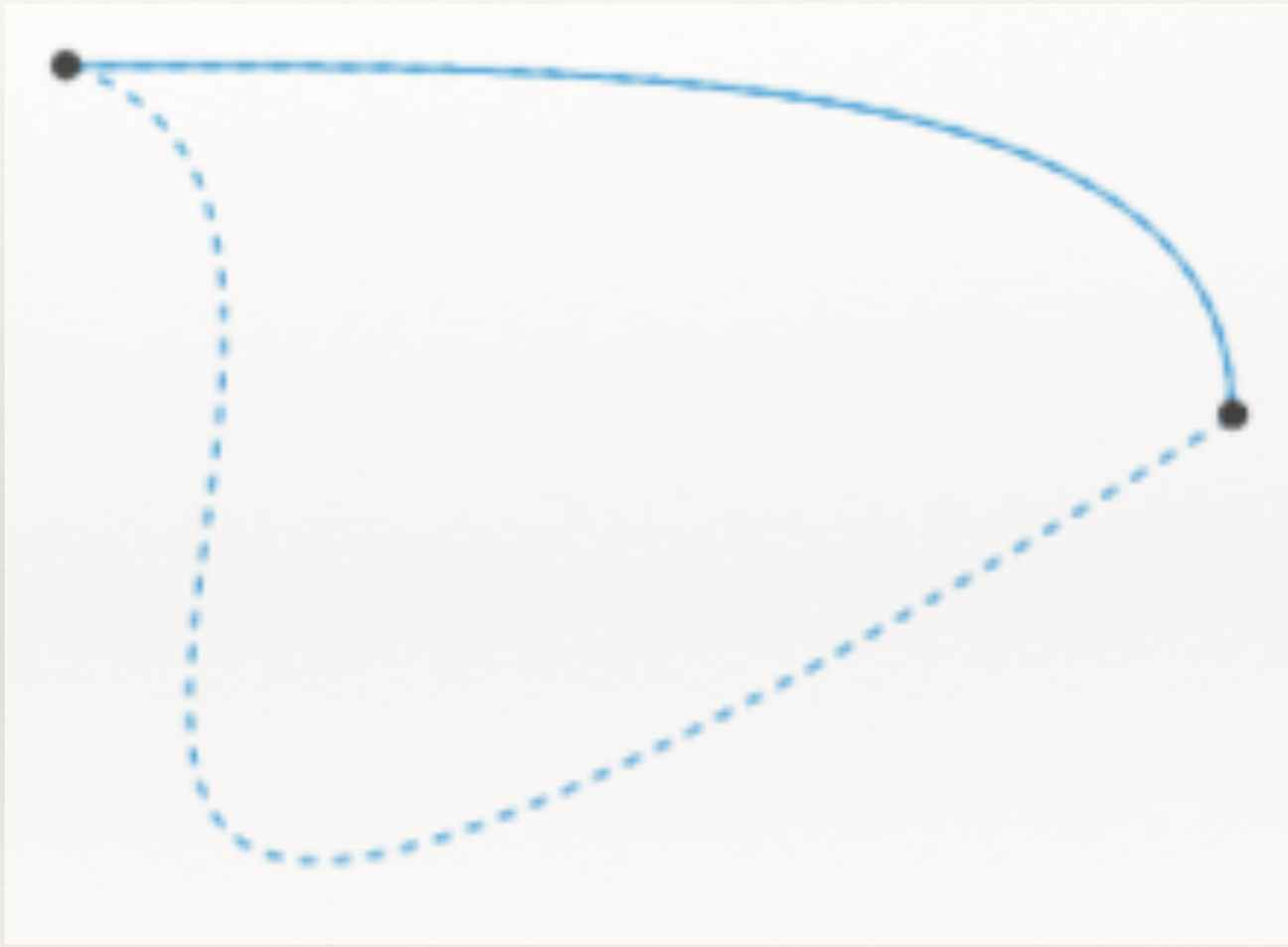

We also have an *extensional* understanding of type equality

# Type Equality

- Given an operational model of how a type $A$ **reduces** to a type $B$
- Given a denotational model of which types $A$ and $B$ are equivalent
- Want soundness and completeness results
- Guarantee that the two notions of equality are themselves "equivalent"
- Generalized "identity of indiscernibles"

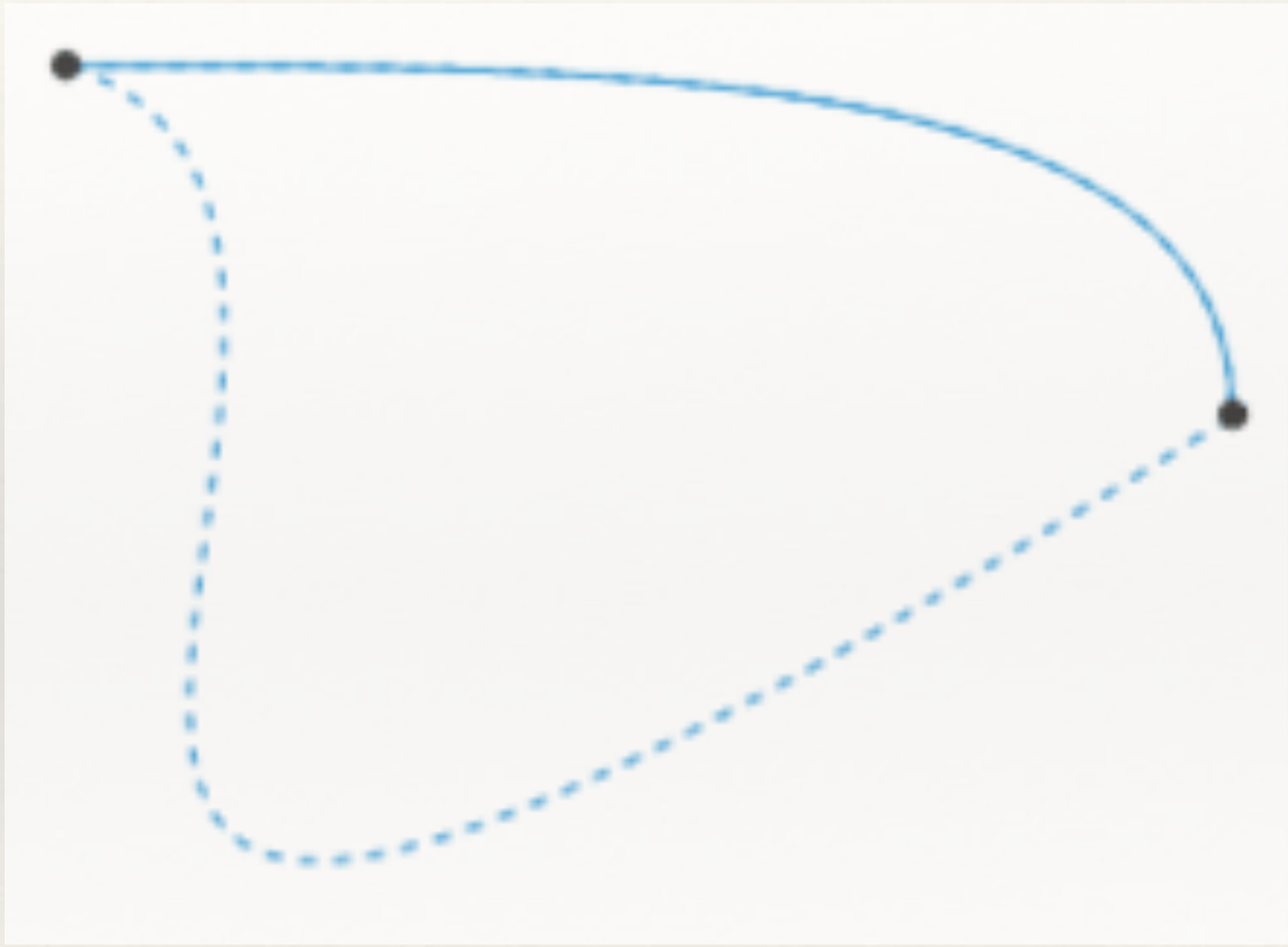
# Path Types and Equivalences

- In HoTT
- *Path types* capture the operational perspective on equality
- *Equivalences* capture the extensional perspective on equality



# Path Types and Equivalences

- In HoTT
- *Path types* capture the operational perspective on equality
- *Equivalences* capture the extensional perspective on equality



# Univalence

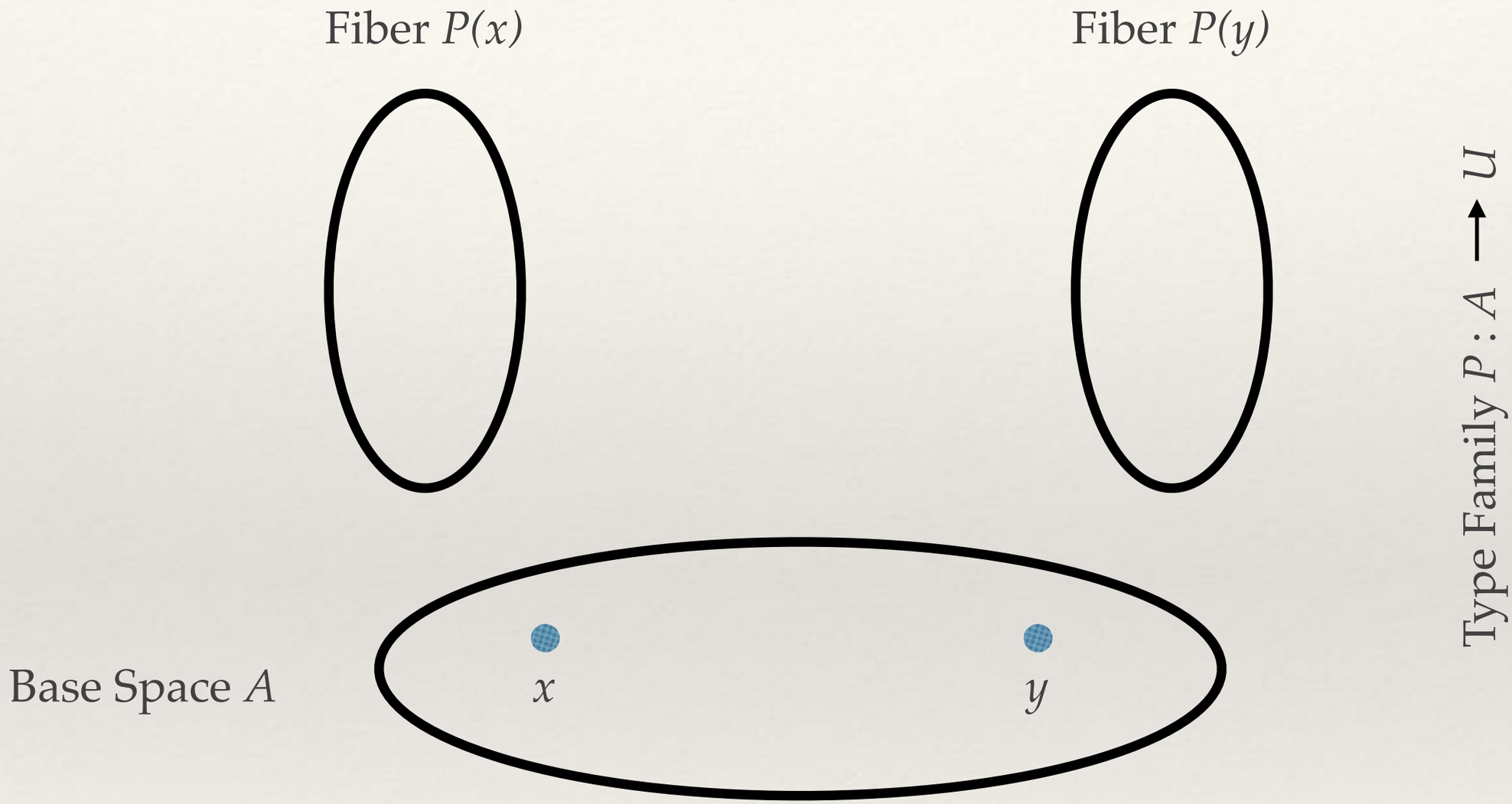
$(A \equiv B) \simeq (A \simeq B)$
where $A \equiv B$ is the type of paths
and $A \simeq B$ is the type of equivalences

A postulate / axiom in "book HoTT"

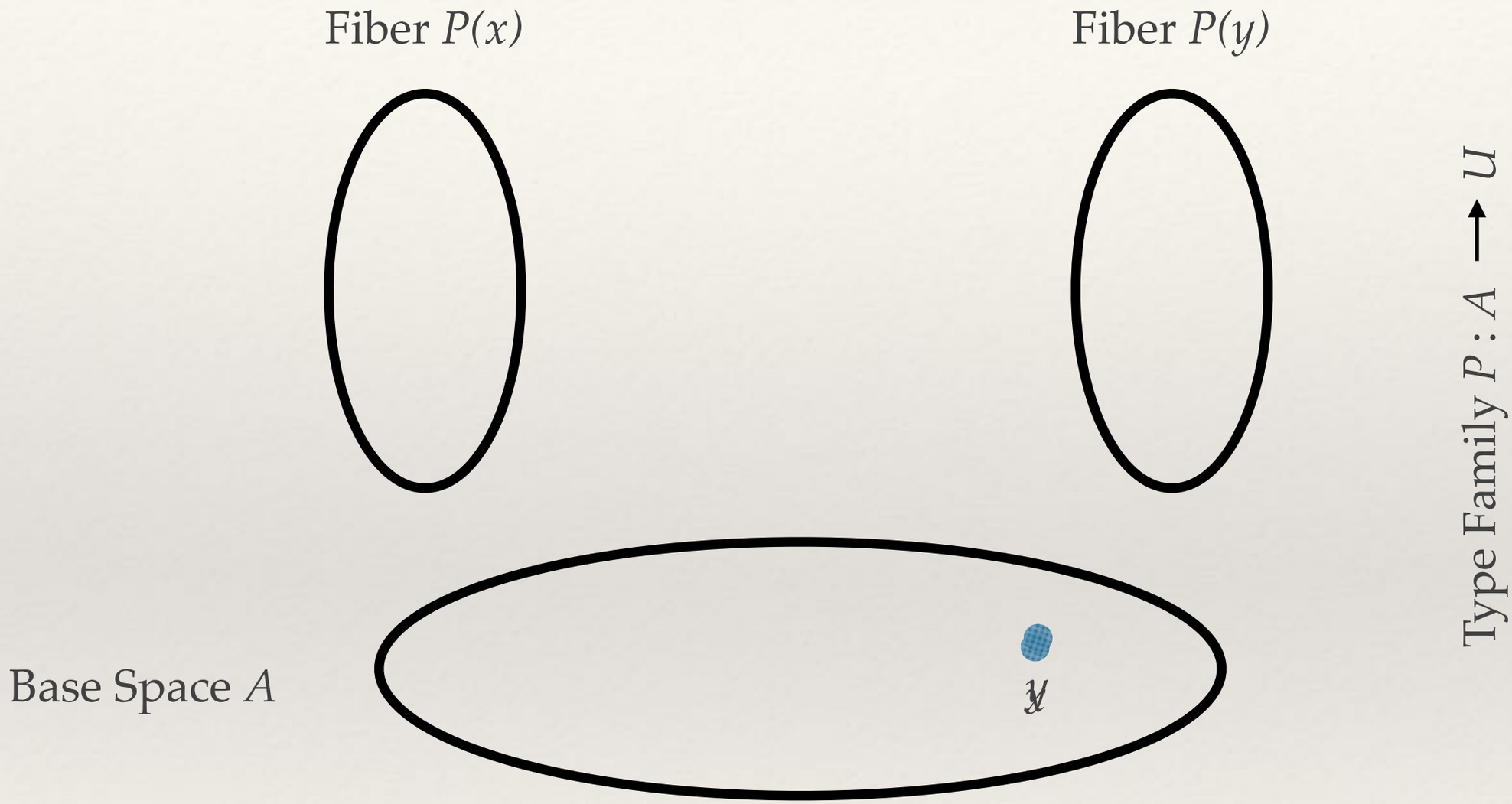
If $A$ and $B$ are types in universe $U$, we say $U$ is a **univalent universe**

# Generalize from Univalent Universes to Univalent Fibrations

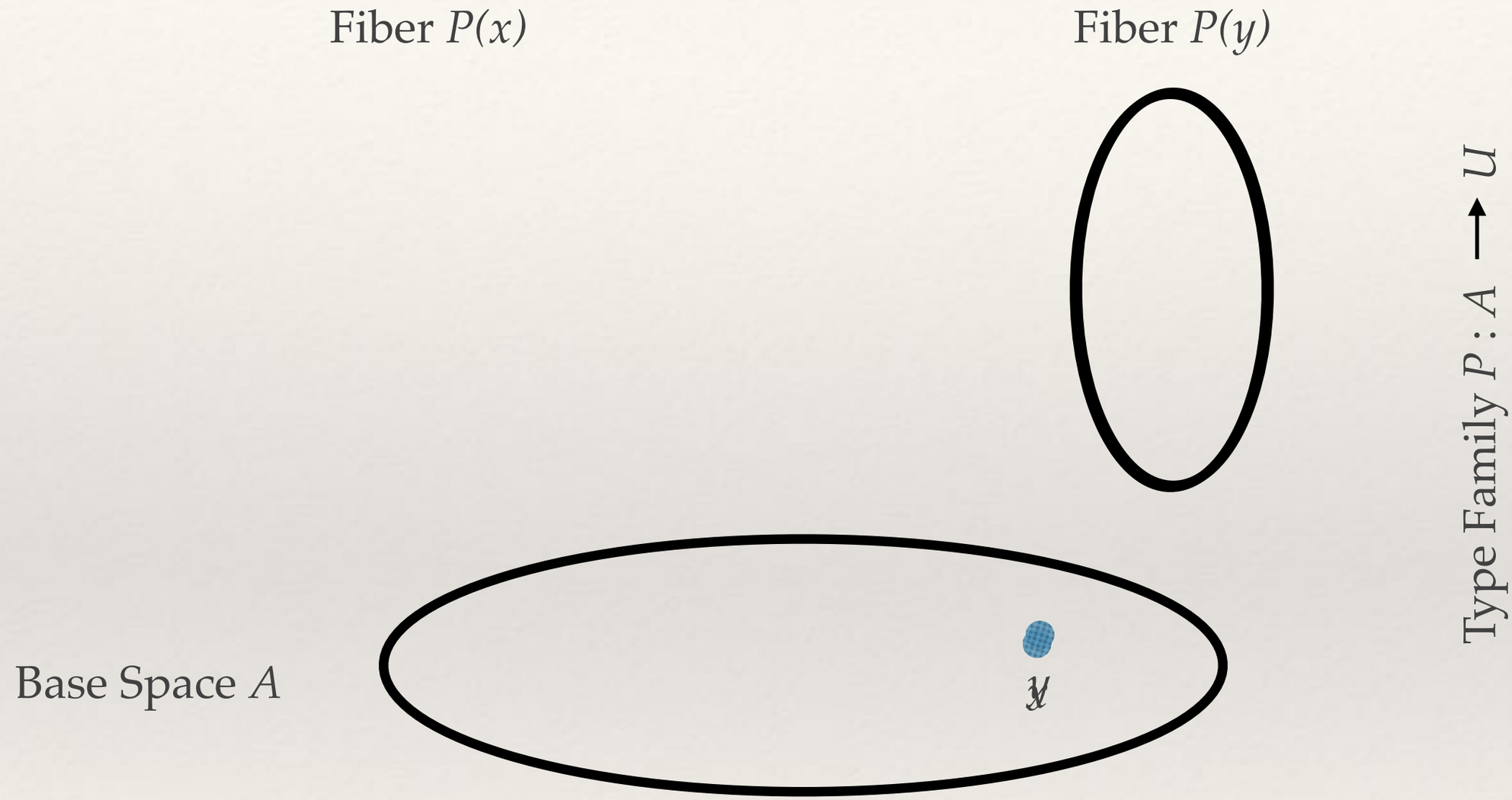
# Fibrations



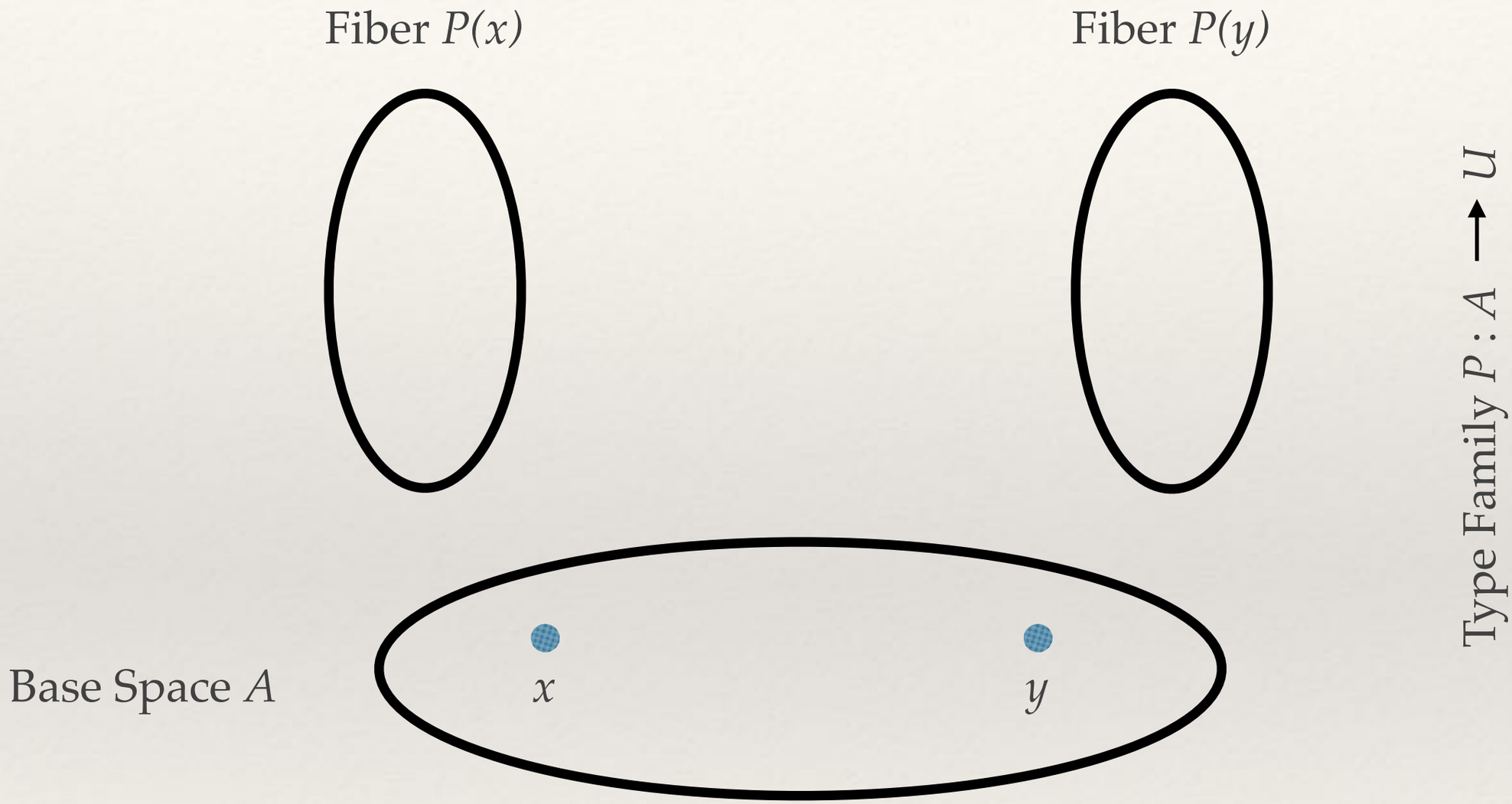
# Fibrations



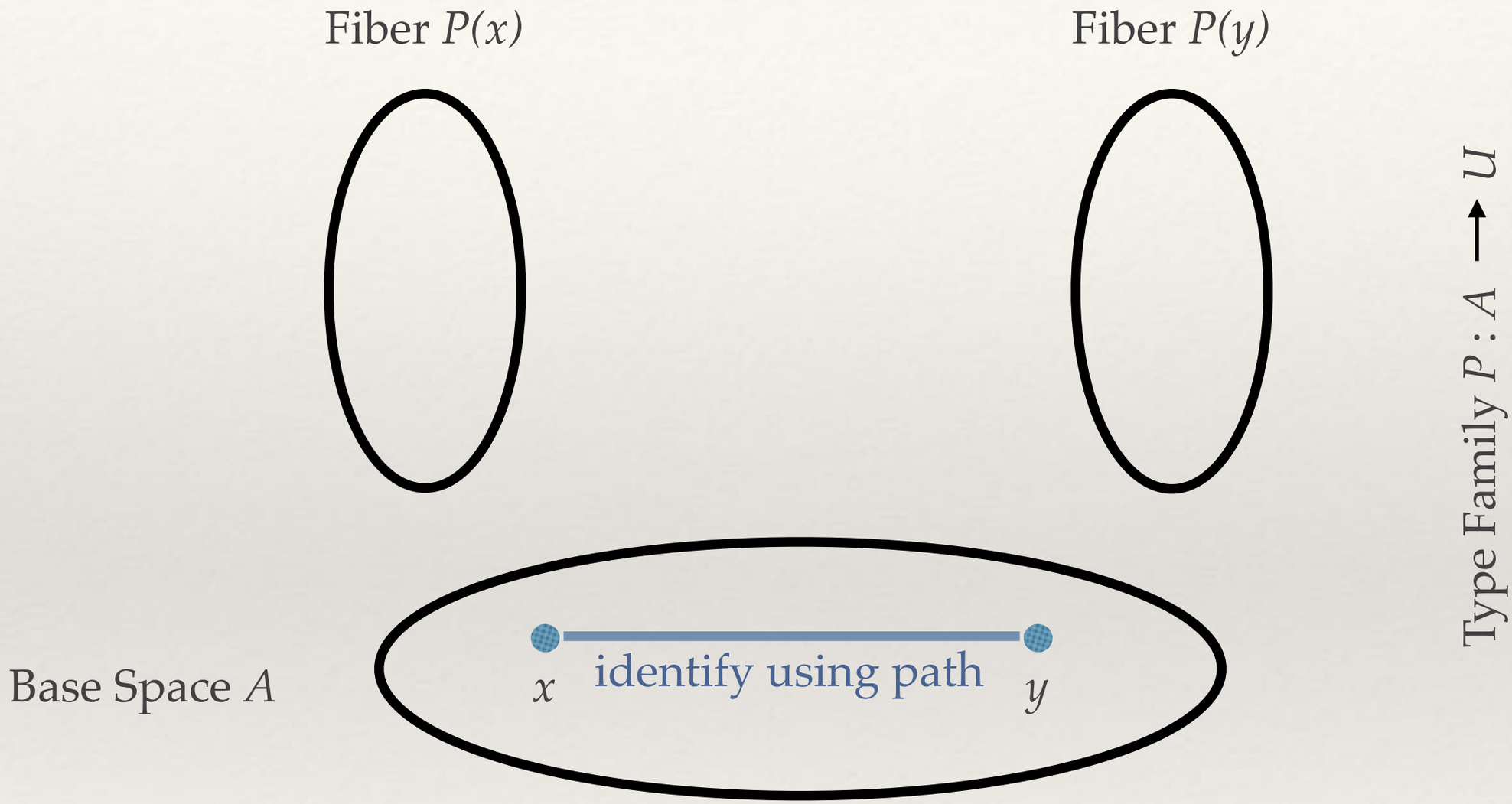
# Fibrations



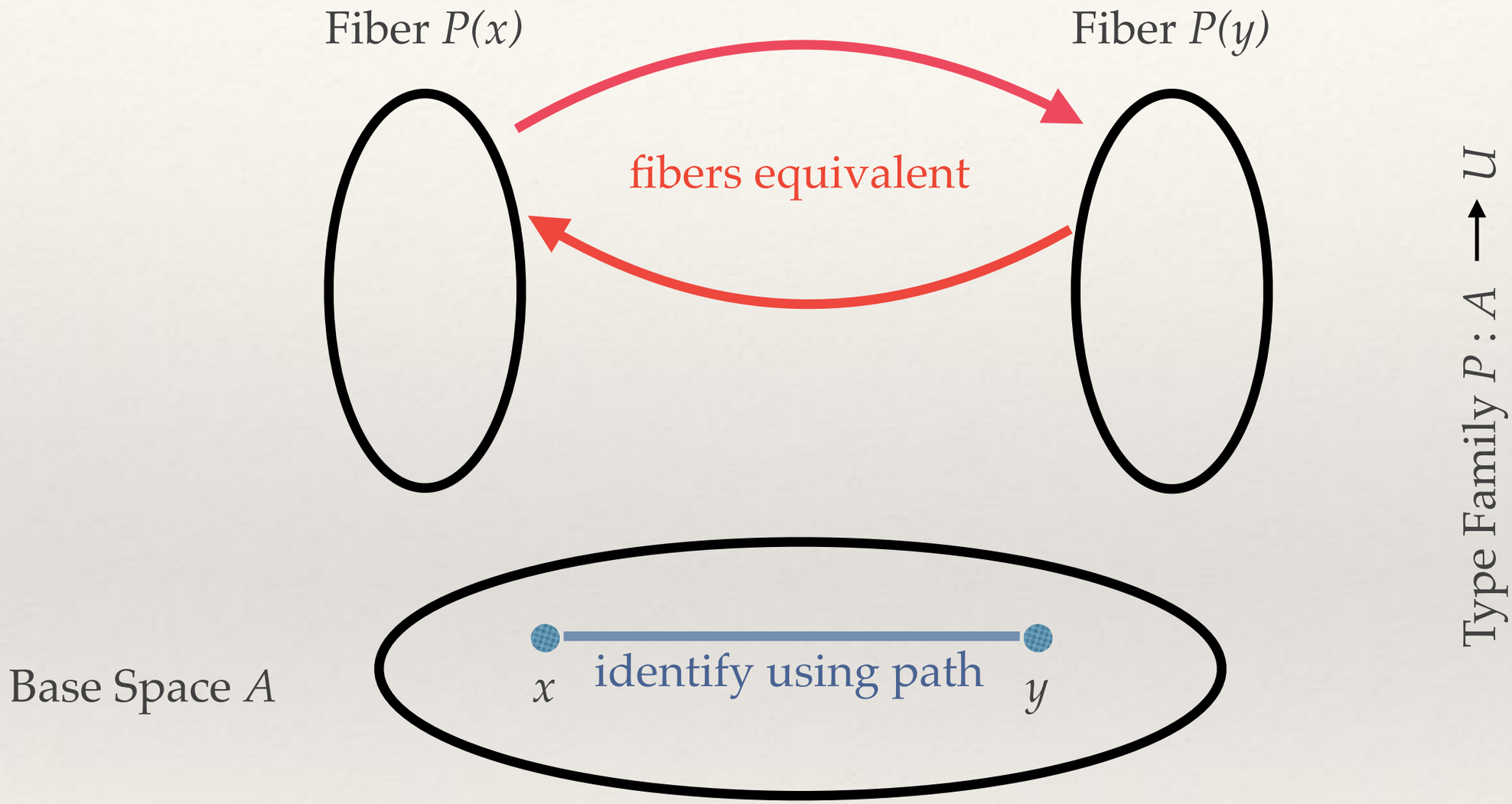
# Fibrations



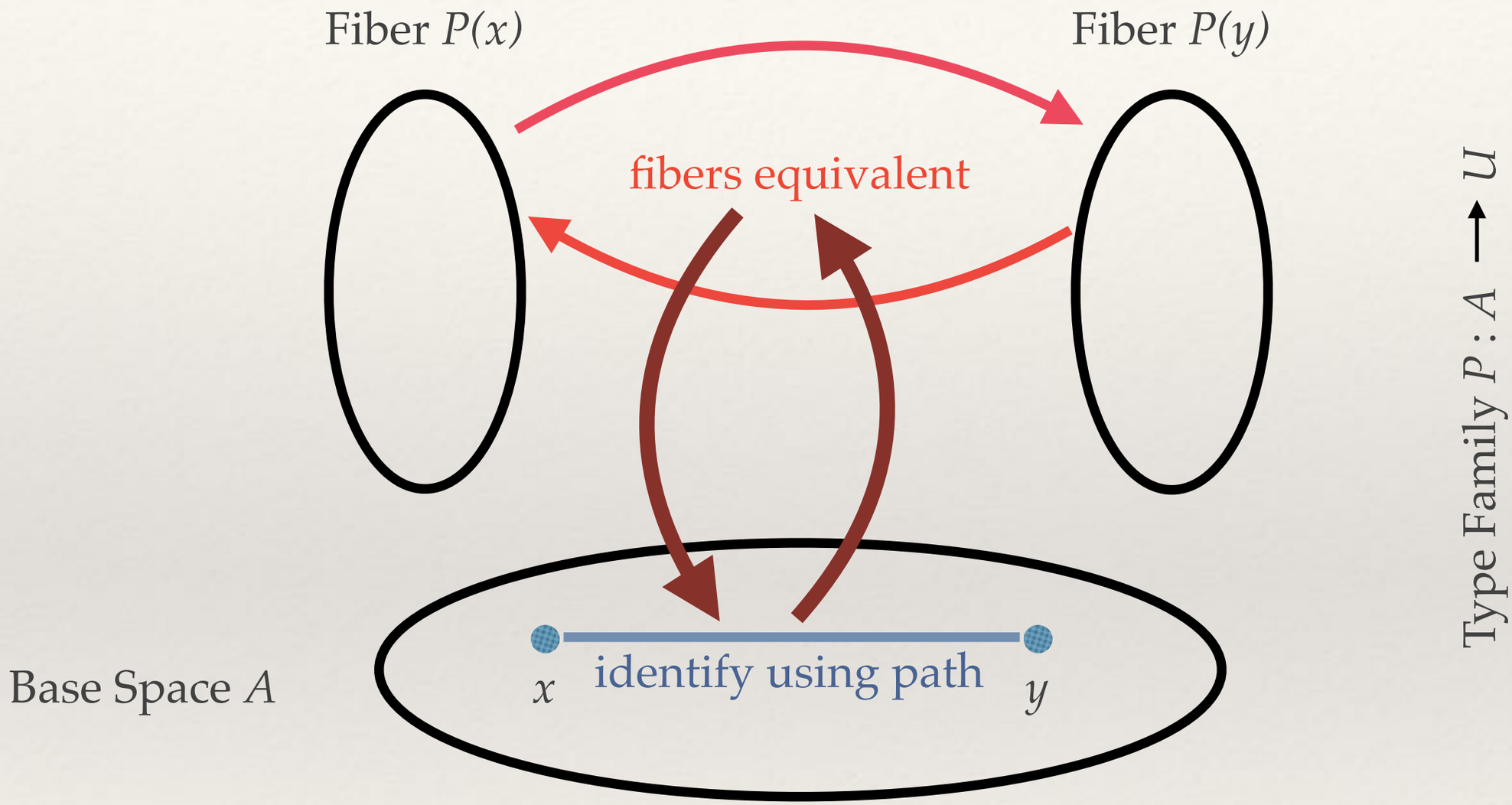
# Fibrations


Fiber $P(x)$
Fiber $P(y)$
Type Family $P : A \to U$
Base Space $A$
$x$
identify using path
$y$

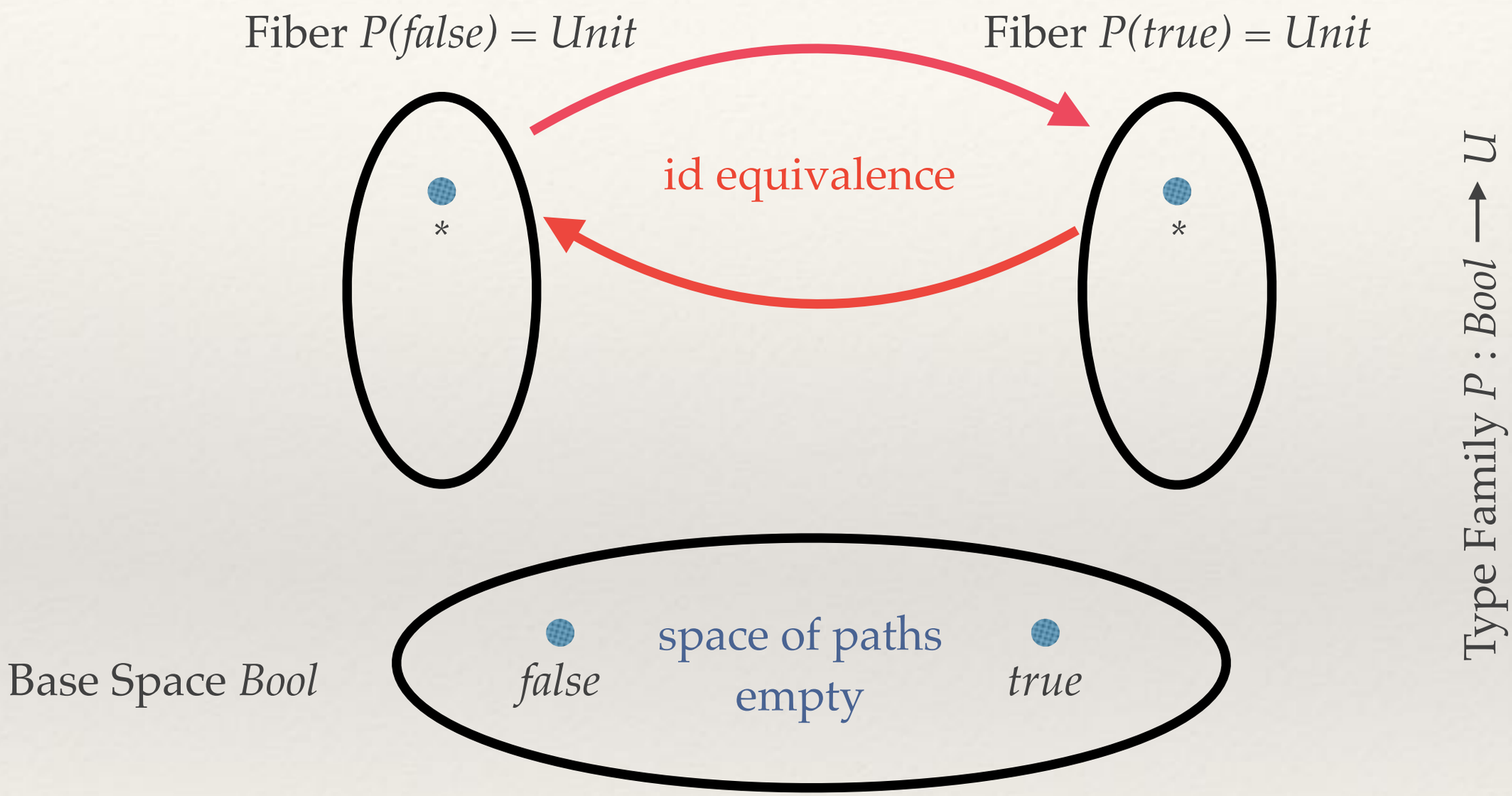
# Fibrations


Fiber $P(x)$
Fiber $P(y)$
fibers equivalent
Type Family $P : A \to U$
Base Space $A$
$x$
identify using path
$y$

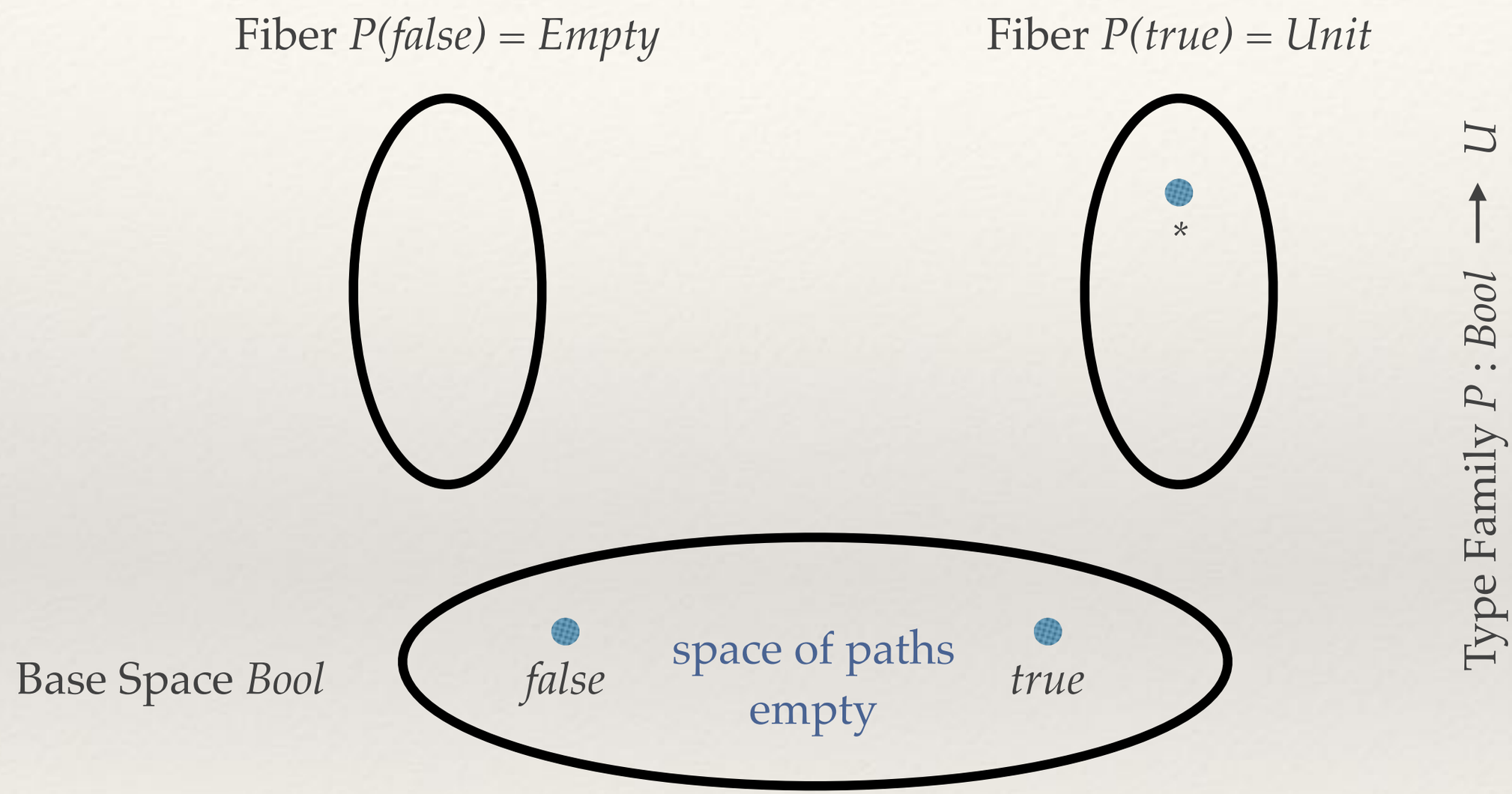
# *Univalent* Fibrations



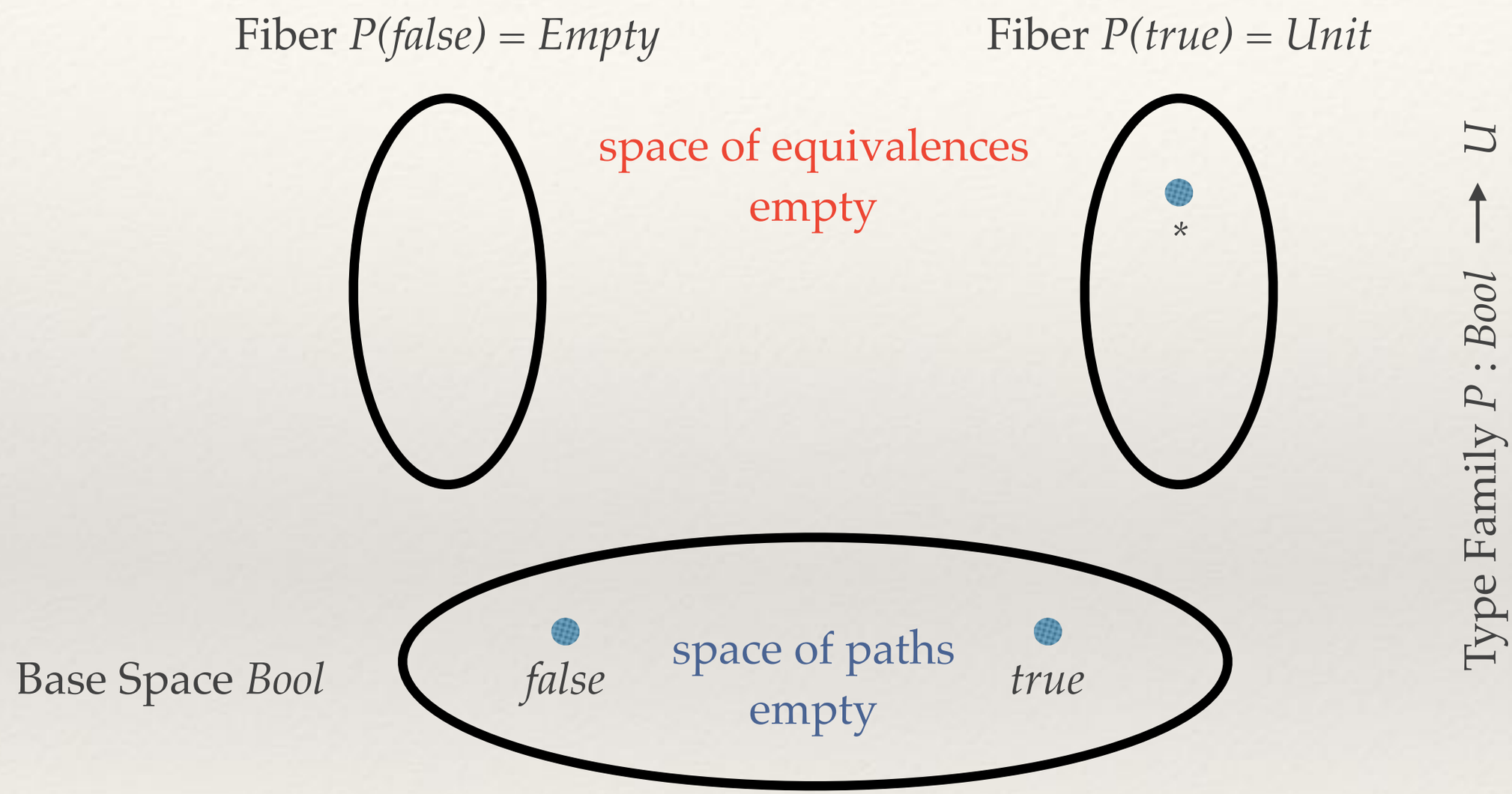
# A Non-Univalent Fibration



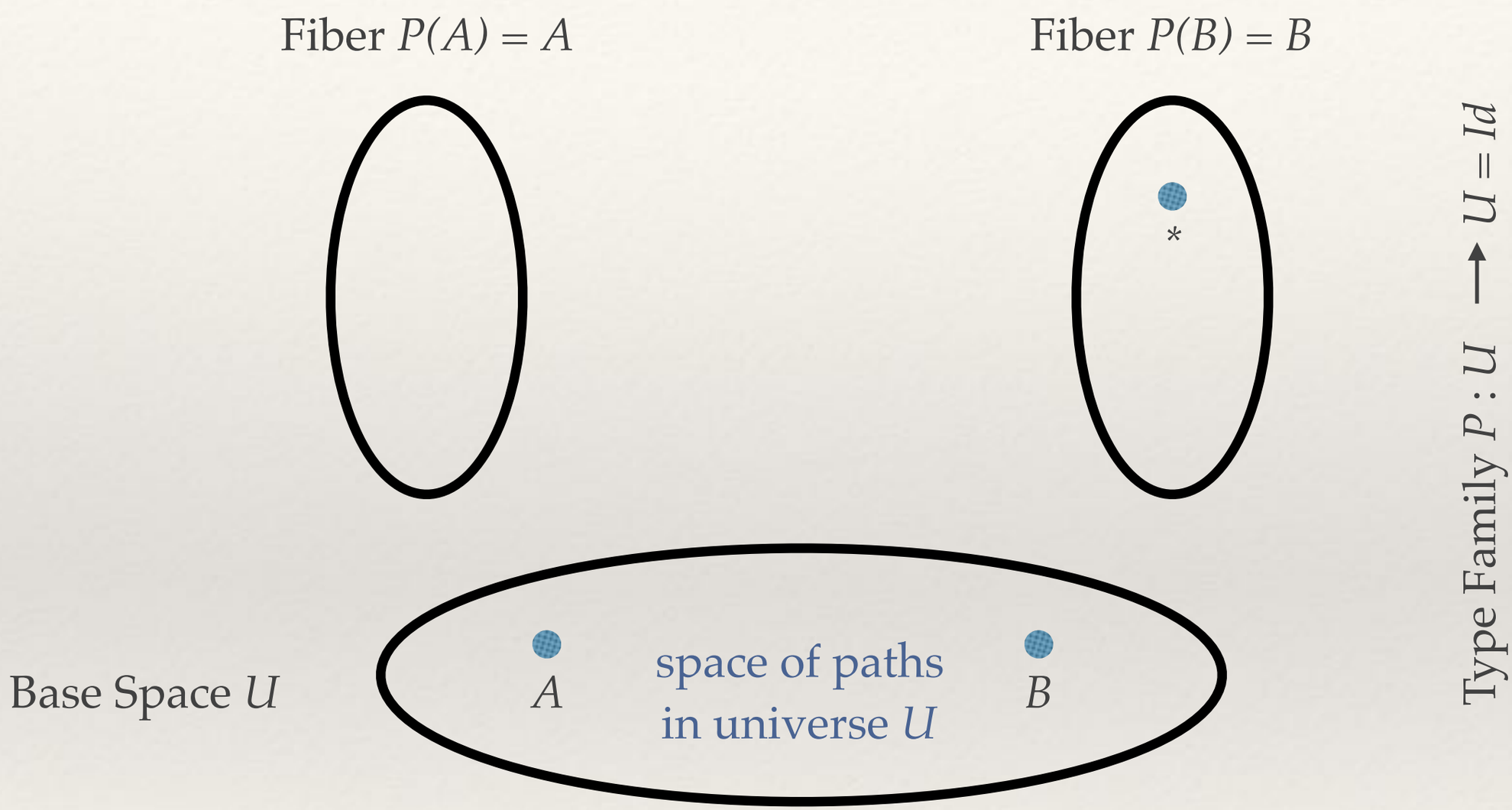
# A Univalent Fibration



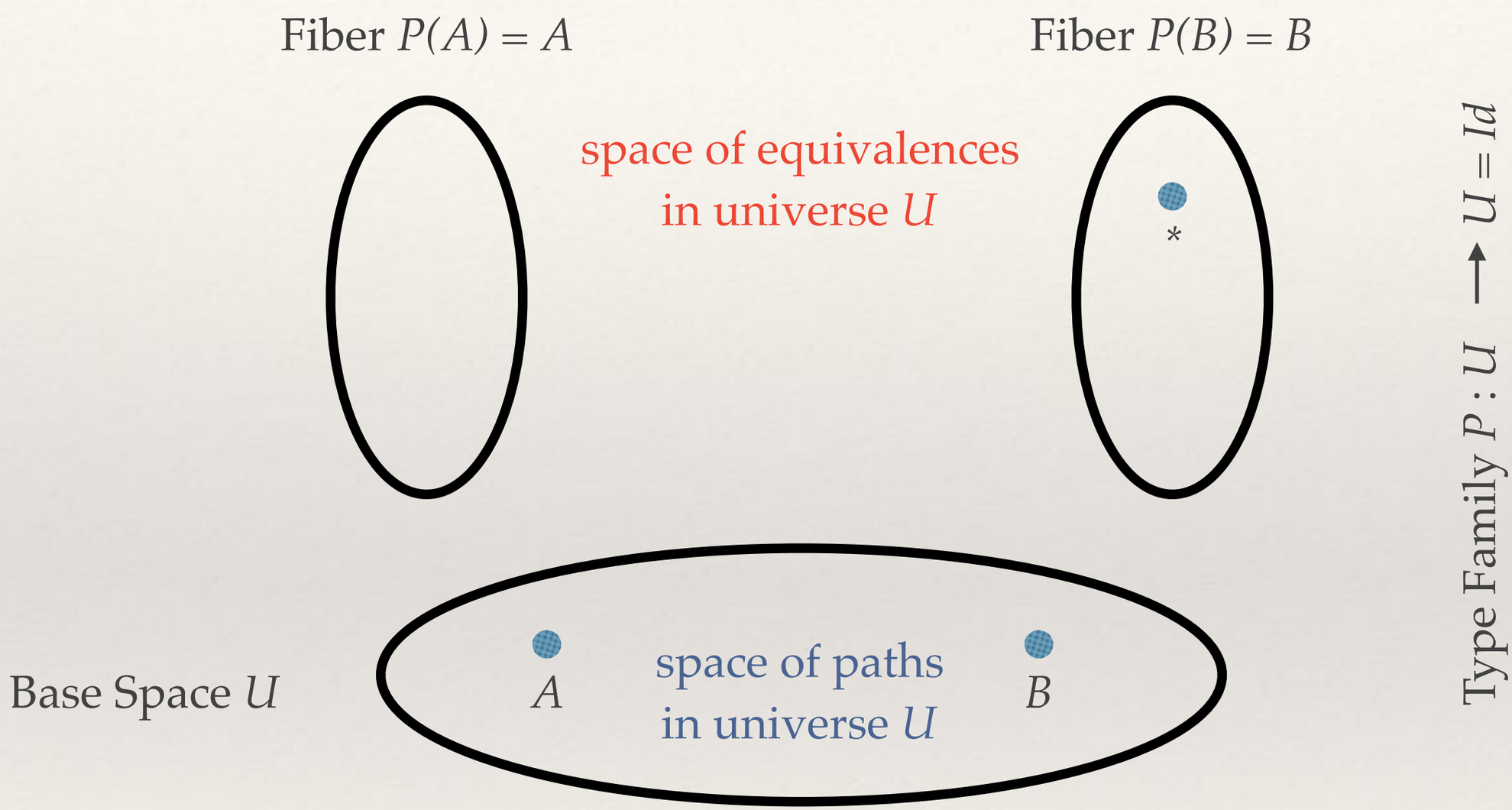
# A Univalent Fibration



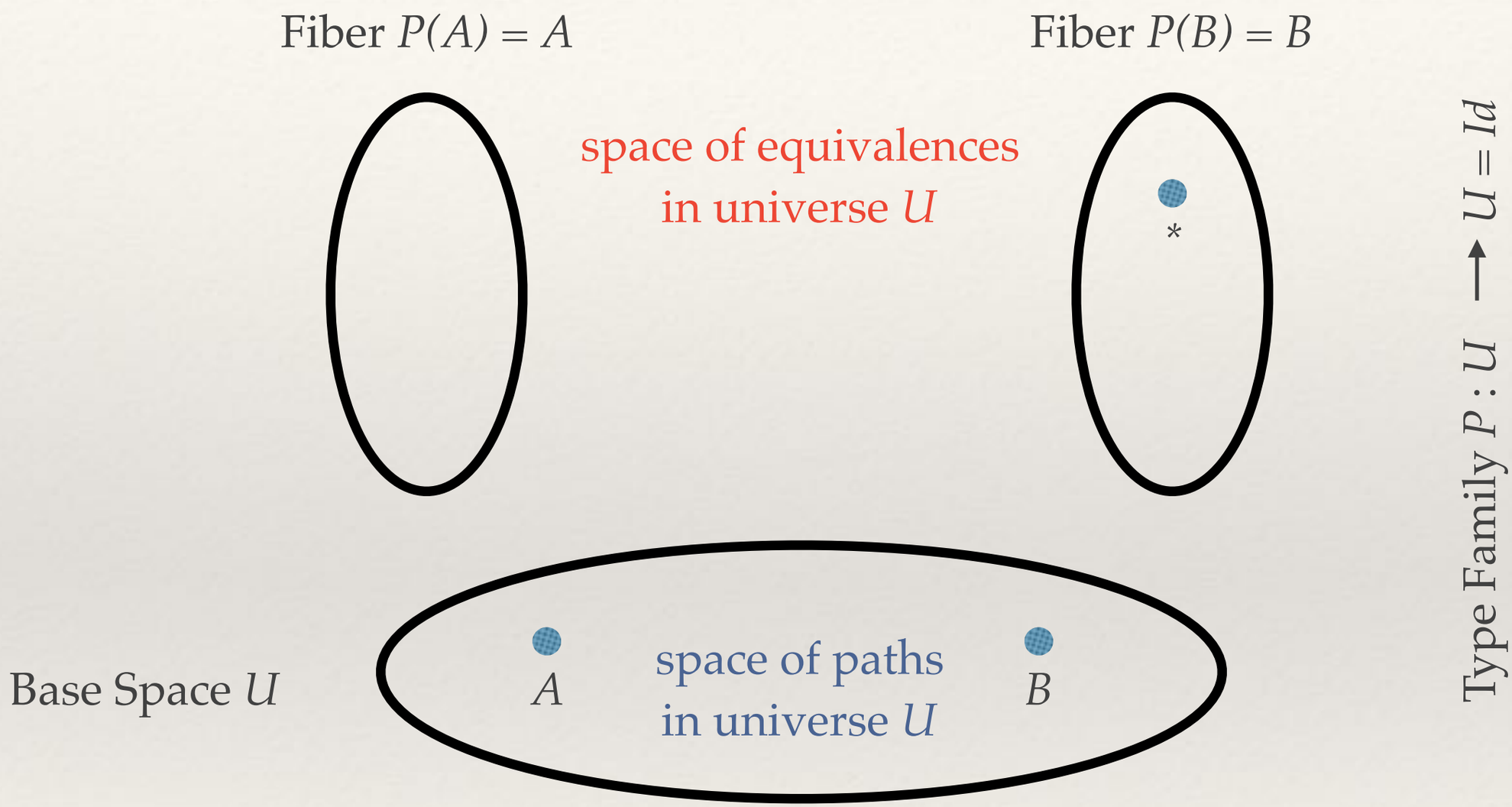
# Connection between univalent universes and univalent fibrations



# Connection between univalent universes and univalent fibrations



# Connection between univalent universes and univalent fibrations




**Fibration is univalent if the universe $U$ is univalent**

# Characterizing Univalent Fibrations

A given type $A$ is rarely the base of a univalent fibration; but we have the following theorem.

**Theorem [Christensen].**
*For any type F, the type family:*

$$\Sigma A{:}U \, \| F \equiv A \|$$

*is always the base of a univalent fibration.*

# Parsing $\Sigma A{:}U \,\|F \equiv A\|$

$\Sigma$

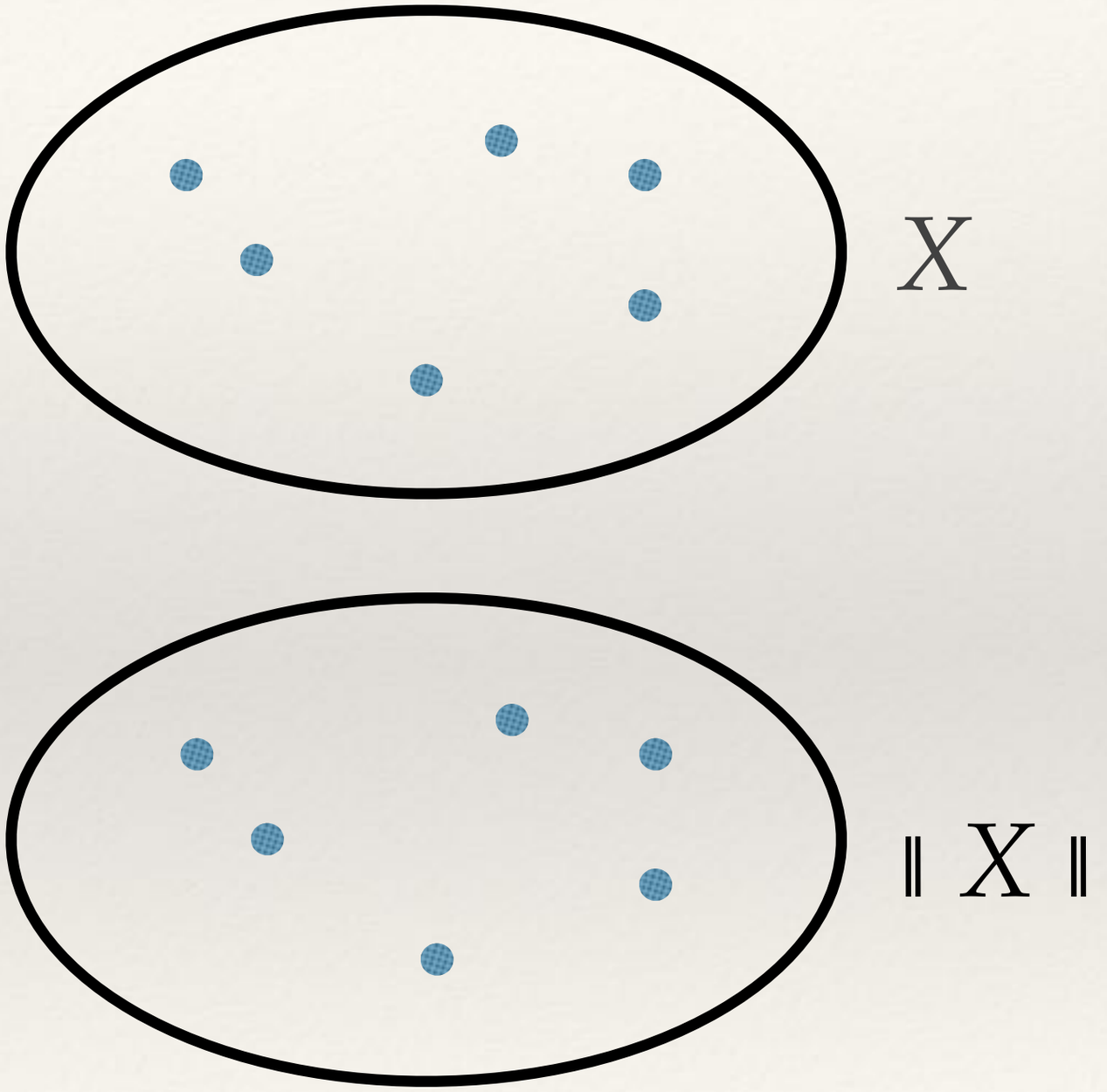
(dependent sum)

$F \equiv A$

(path type)

$\| X \|$

(propositional truncation)



# Parsing $\Sigma A{:}U \,\|F \equiv A\|$

$\Sigma$

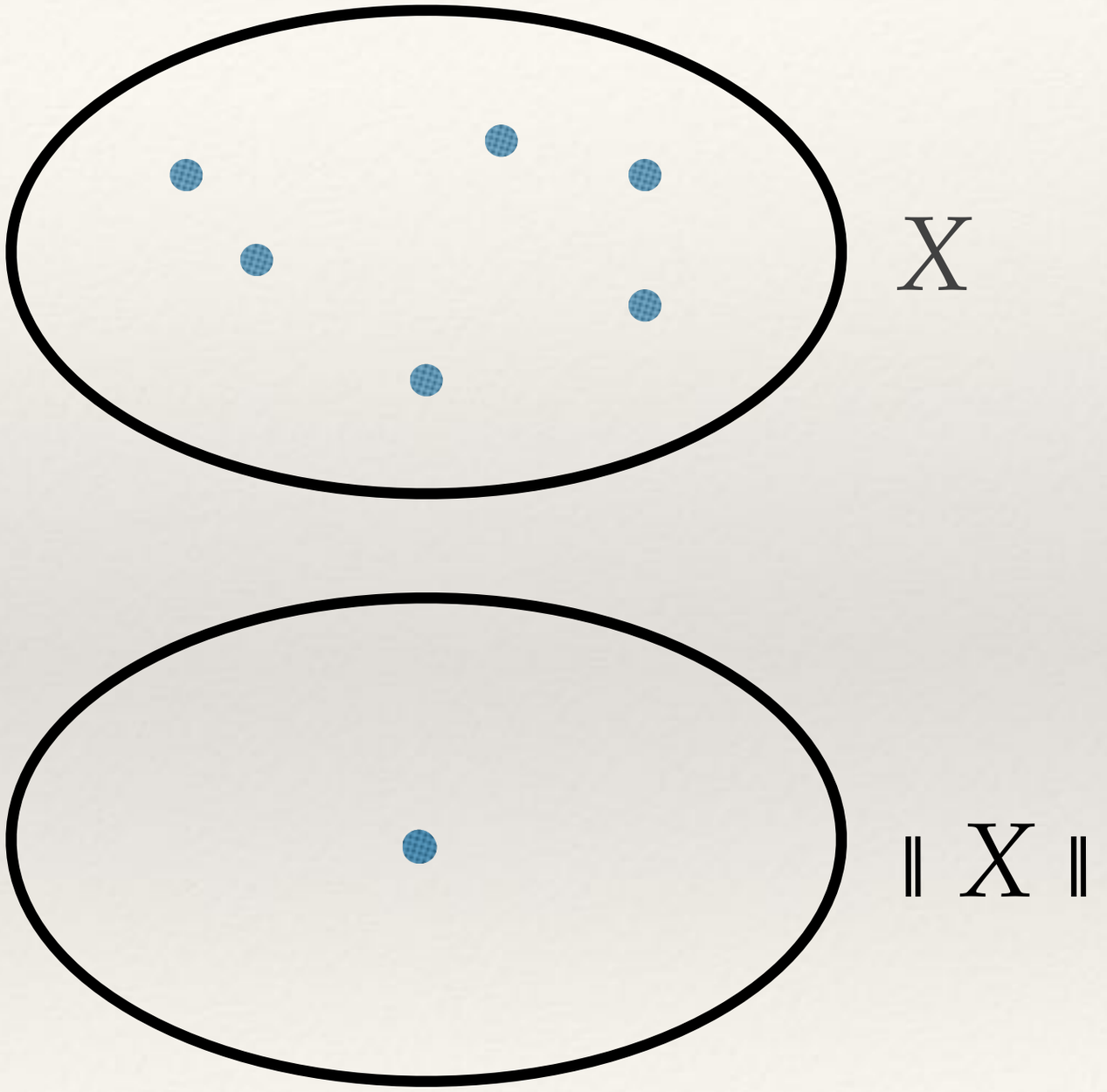
(dependent sum)

$F \equiv A$

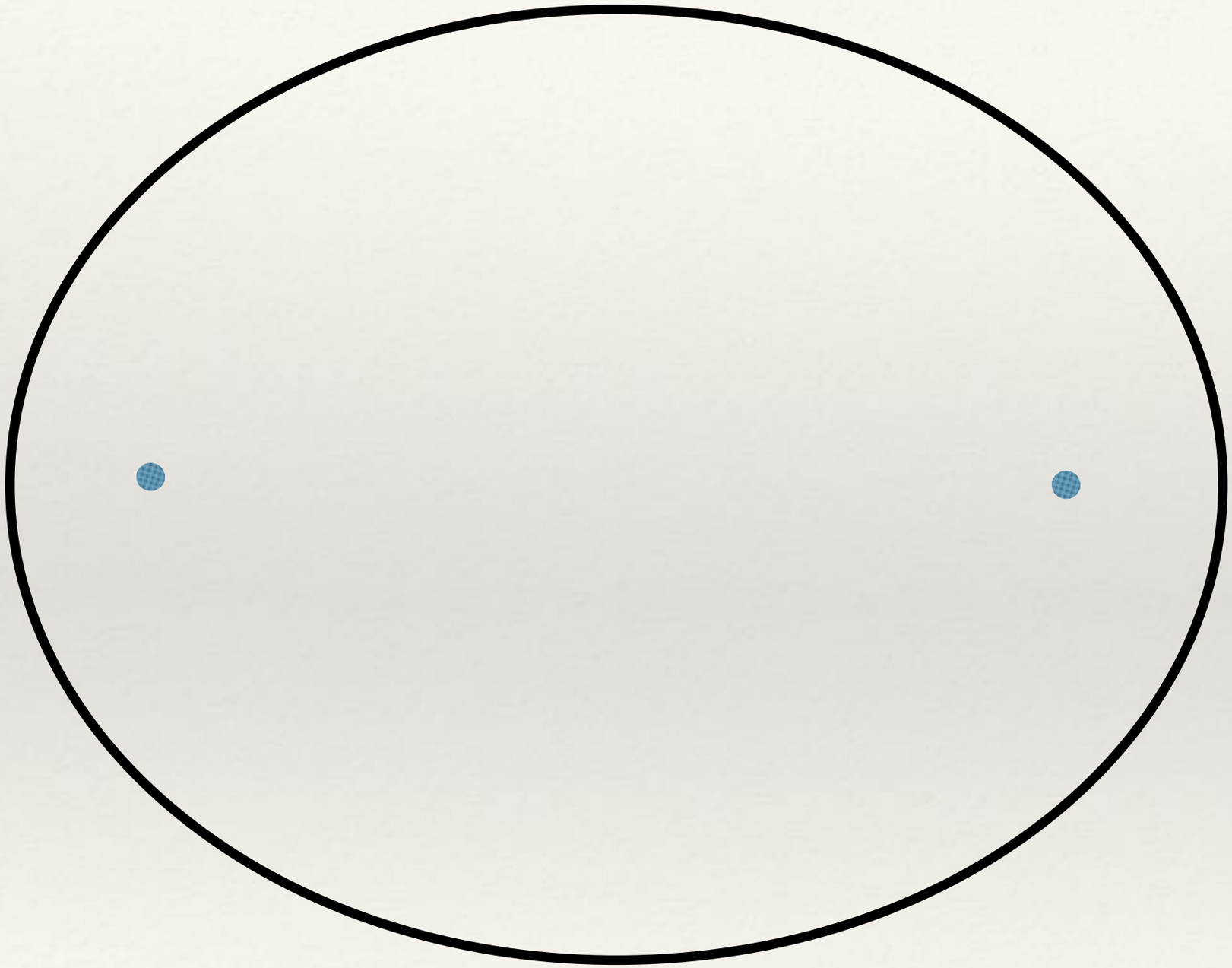
(path type)

$\| X \|$

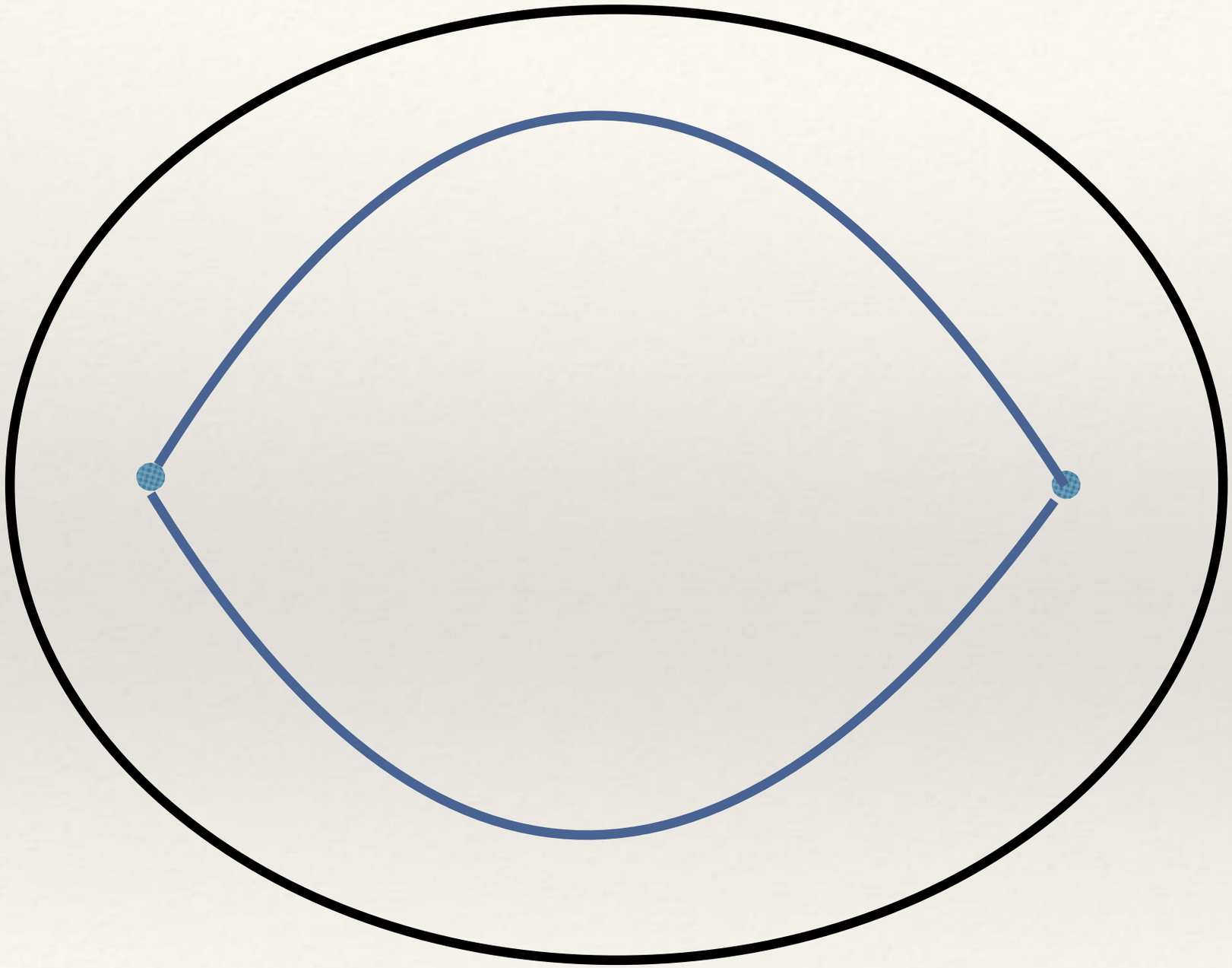
(propositional truncation)



# Propositional Truncation as Colimit (van Doorn)

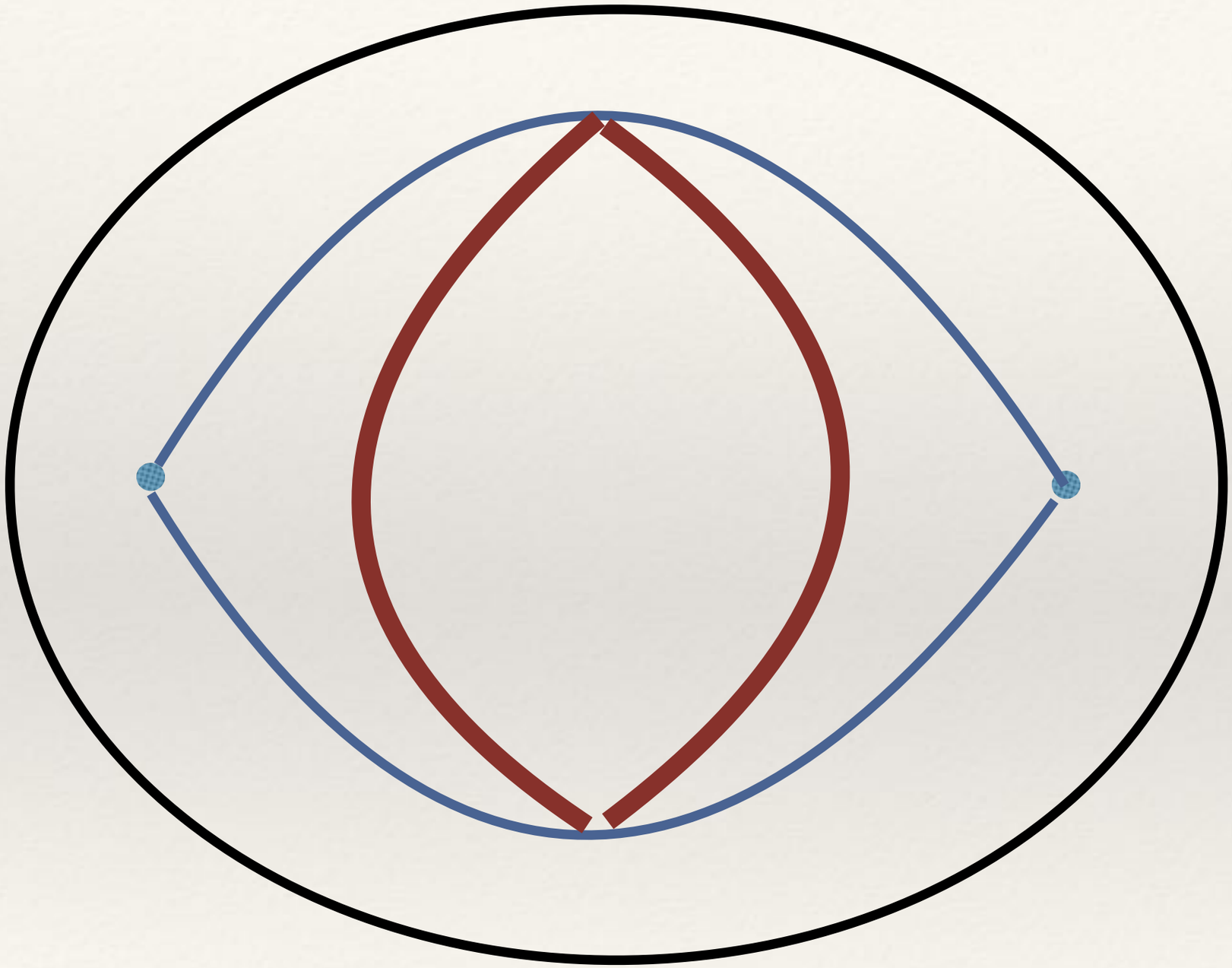


# Propositional Truncation as Colimit (van Doorn)



For any two points, add a path between them

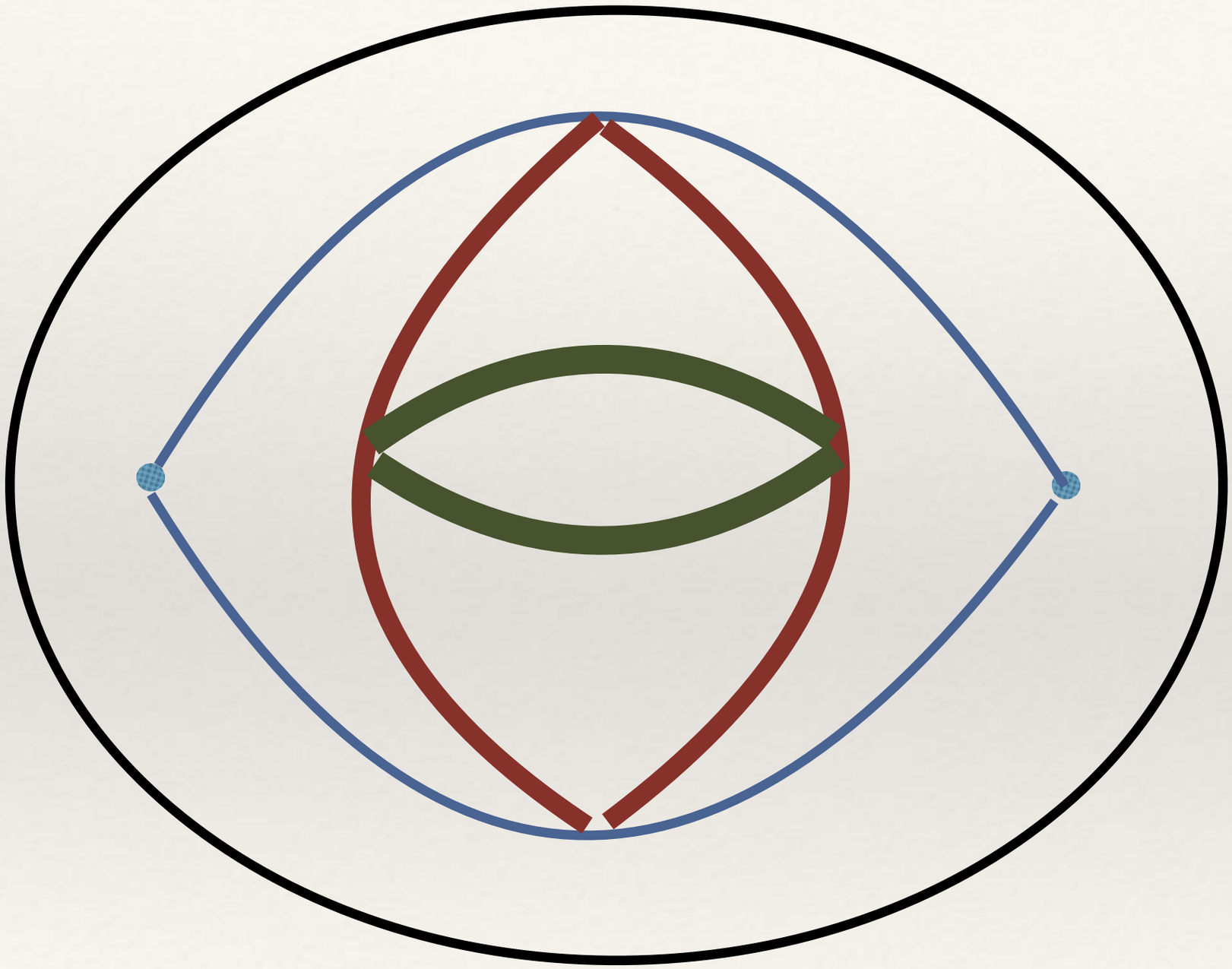
# Propositional Truncation as Colimit (van Doorn)



For any two points, add a path between them

For any new paths, add a higher-level path between them

# Propositional Truncation as Colimit (van Doorn)

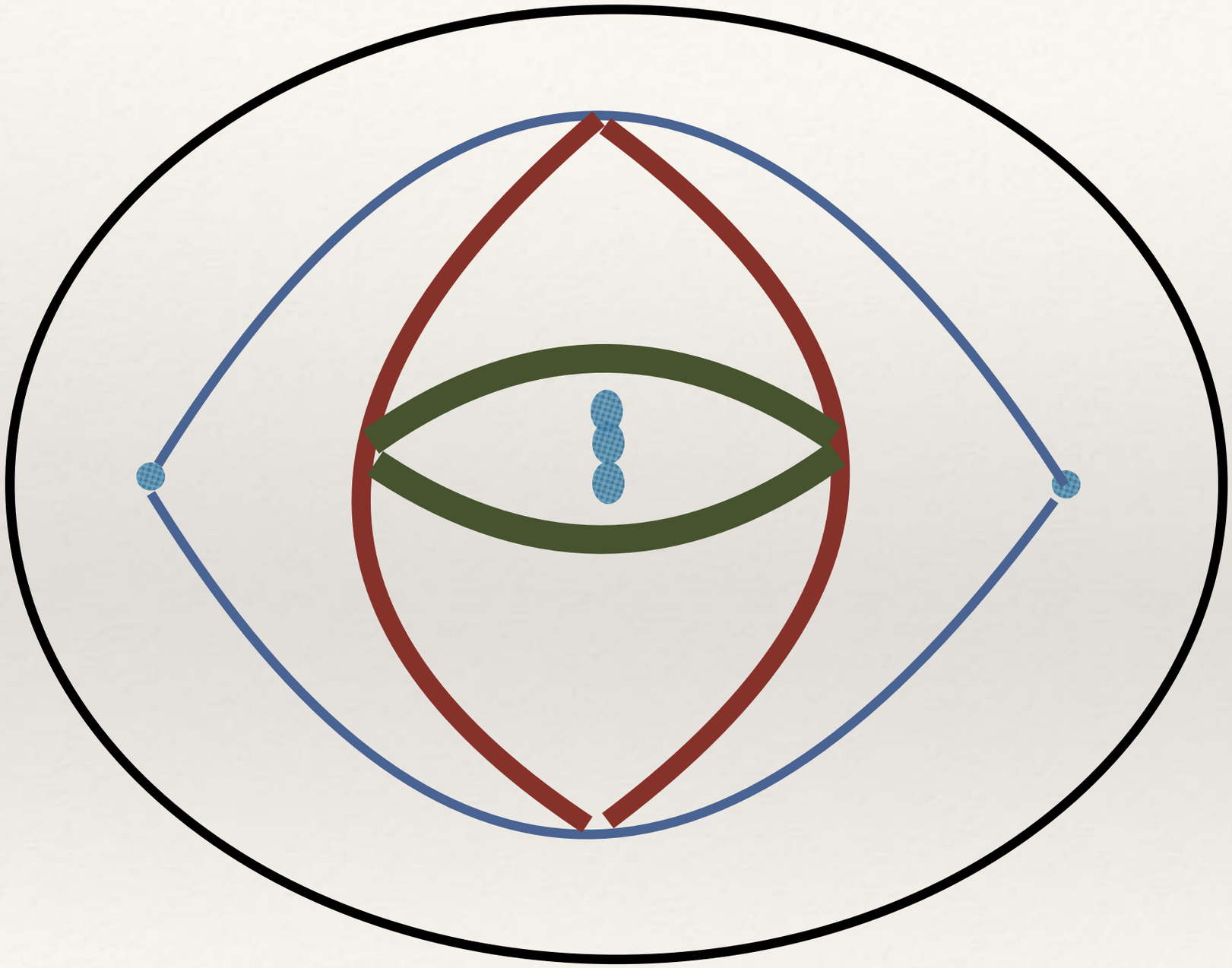


For any two points, add a path between them

For any new paths, add a higher-level path between them

For any new paths, add a higher-level path between them

# Propositional Truncation as Colimit (van Doorn)



For any two points, add a path between them

For any new paths, add a higher-level path between them

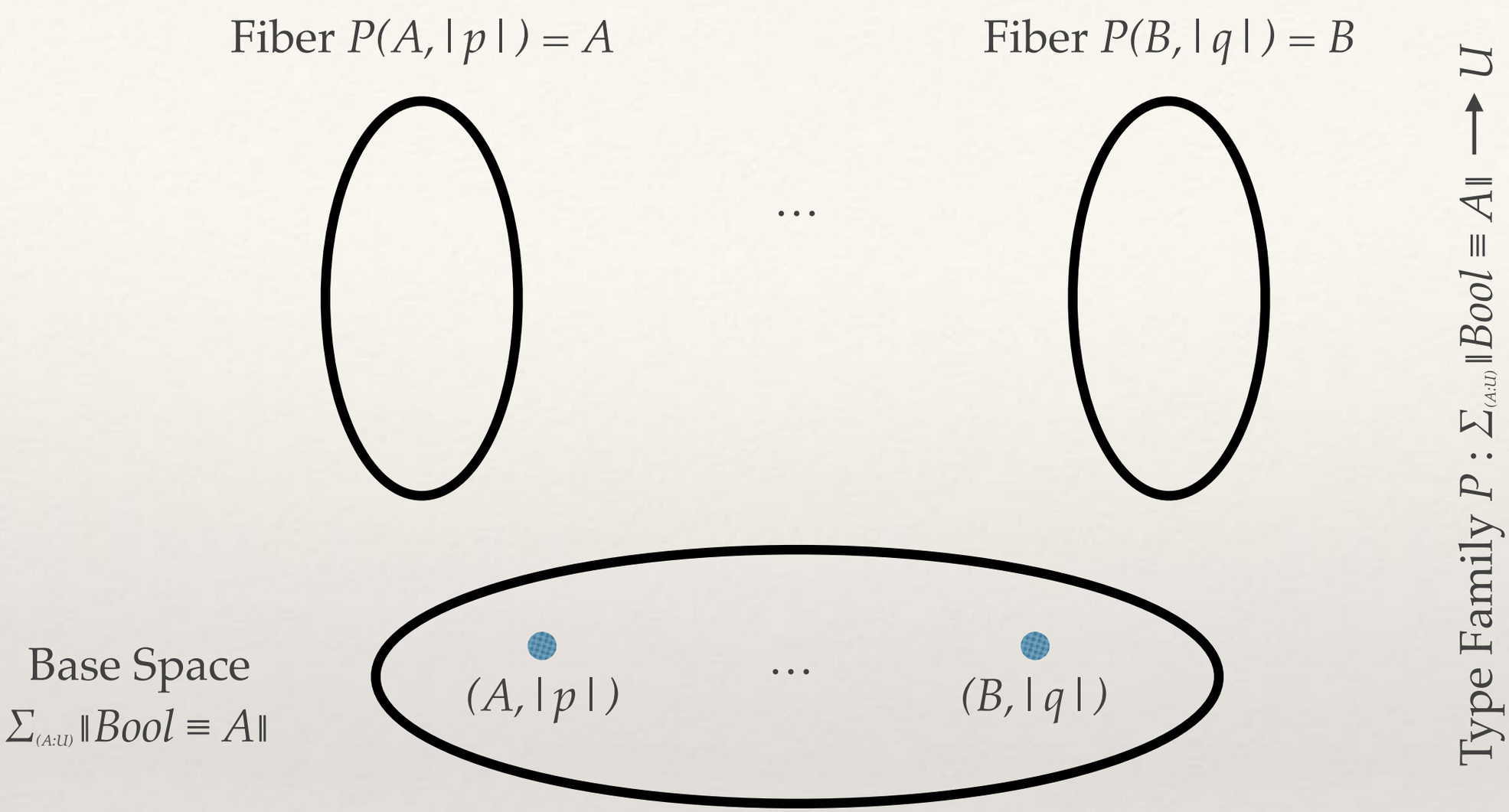
For any new paths, add a higher-level path between them

For any new paths, add a higher-level path between them…

# Questions

- What does $\Sigma_{(A:U)} \| F \equiv A \|$ look like for arbitrary $F$ ?

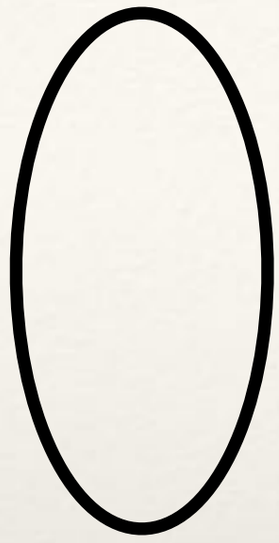
- Easier question: what does $\Sigma_{(A:U)} \| Bool \equiv A \|$ look like?



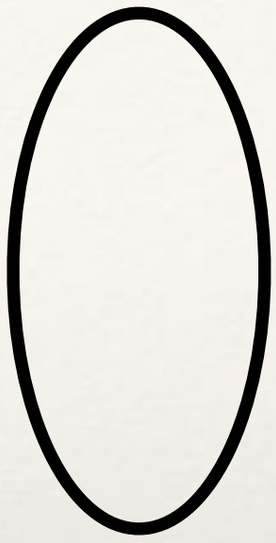

All the types $A$, $B$, …
are essentially
equivalent to *Bool*

Fiber $P(A, |\, p\, |) = A$

Fiber $P(B, |\, q\, |) = B$



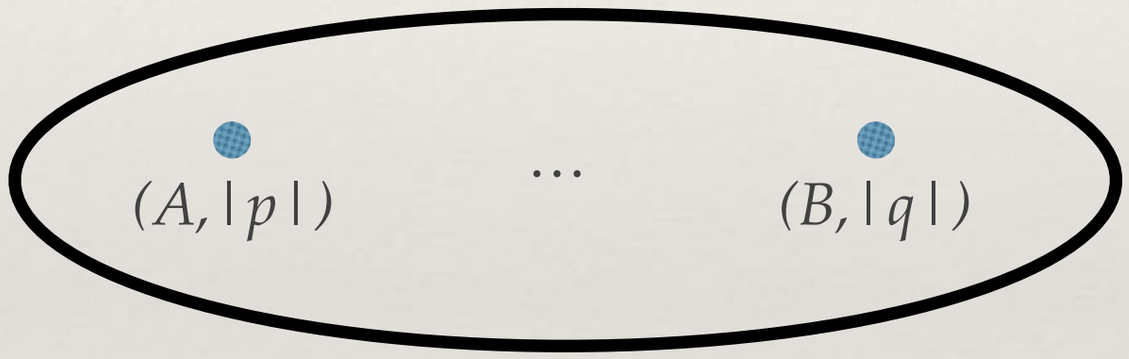
…



Type Family $P : \Sigma_{(A:\mathcal{U})} \| Bool \equiv A \| \longrightarrow \mathcal{U}$

Base Space

$\Sigma_{(A:\mathcal{U})} \| Bool \equiv A \|$

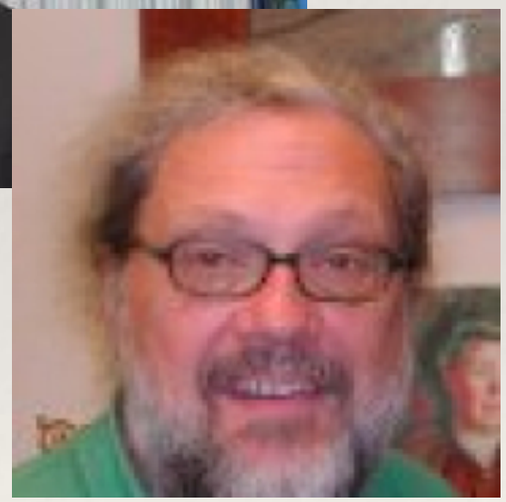



All the types $A$, $B$, … are essentially equivalent to $Bool$

$$
\begin{array}{rcl}
(A, |p|) \equiv (B, |q|) & \simeq & (A \simeq B) \\
(Bool, |p|) \equiv (Bool, |q|) & \simeq & (Bool \simeq Bool)
\end{array}
$$

# $\Sigma A{:}U \, \| Bool \equiv A \|$

- It is a space (a weak infinity groupoid)
- Let's try to find out the points
- Then the 1-paths
- Then the 2-paths
- Then the 3-paths
- And so on…
- Or perhaps the paths become all trivial after some level?



# Agda to the Rescue

Base of univalent fibration

```
is-type : ∀ {ℓ} (T : Type ℓ) → _
is-type T = λ X → ∥ X == T ∥

is-2 = is-type 2

U[2] : Type₁
U[2] = Σ Type₀ is-2
```

# Points and Paths

```
-- Labels for some of the pertinent terms
`2 : U[2]
`2 = (2 , | refl 2 |)


`id : {A : U[2]} → A == A
`id {A} = refl A


`not : `2 == `2
`not = dpair= (ua not-eqv , identify _ _)
```

# Points and Paths

```
-- Labels for some of the pertinent terms
`2 : U[2]
`2 = (2 , | refl 2 |)


`id : {A : U[2]} → A == A
`id {A} = refl A


`not : `2 == `2
`not = dpair= (ua not-eqv , identify _ _)
```

```
all-1-paths : (p : `2 == `2) → (p == `id) + (p == `not)
```

# Recall: Π2 Theorem

**Canonical Forms: Given a level-1 program *p*, the level-2 transformations are complete: they can transform *p* to either `*id* or `*not***

```
cmpl2-lem : (p : `𝟚 ↔₁ `𝟚) → (p ↔₂ `id) + (p ↔₂ `not)
```

# Could it be possible that…

- the base $\Sigma_{(A:U)} \| \mathrm{Bool} \equiv A \|$ of the univalent fibration is *exactly* $\Pi 2$ ?

- What would it mean?

# Yes! The Correspondence

- $\Sigma(A:U)\ \|Bool \equiv A\|$ is a topological space, a higher groupoid that corresponds exactly to a 2-layer reversible programming language
- Points
- Paths
- 2-Paths
- Trivial 3-Paths

$$
\begin{array}{rcll}
\tau & ::= & 2 & \\
\\
v & ::= & \mathit{false} : 2 & \\
& | & \mathit{true} : 2 & \\
\\
c & ::= & \mathtt{id} : \tau \leftrightarrow \tau & \\
& | & \mathtt{swap} : 2 \leftrightarrow 2 & \\
& | & ! : (\tau_1 \leftrightarrow \tau_2) \to (\tau_2 \leftrightarrow \tau_1) & \\
& | & \circ : (\tau_1 \leftrightarrow \tau_2) \to (\tau_2 \leftrightarrow \tau_3) \to (\tau_1 \leftrightarrow \tau_3) & \\
\\
\alpha & ::= & \mathtt{id} : c \Leftrightarrow c & \\
& | & \mathit{idl} : \mathtt{id} \circ c \Leftrightarrow c & \\
& | & \mathit{idr} : c \circ \mathtt{id} \Leftrightarrow c & \\
& | & \mathit{invl} : c \circ ! c \Leftrightarrow \mathtt{id} & \\
& | & \mathit{invr} : ! c \circ c \Leftrightarrow \mathtt{id} & \\
& | & \rho : \mathtt{swap} \circ \mathtt{swap} \Leftrightarrow \mathtt{id} & \\
& | & \mathit{assoc} : (c_1 \circ c_2) \circ c_3 \Leftrightarrow c_1 \circ (c_2 \circ c_3) & \\
& | & \odot : (c_1 \Leftrightarrow c_1') \to (c_2 \Leftrightarrow c_2') \to (c_1 \circ c_2 \Leftrightarrow c_1' \circ c_2') & \\
& | & !! : (c_1 \Leftrightarrow c_2) \to (c_2 \Leftrightarrow c_1) & \\
& | & \bullet : (c_1 \Leftrightarrow c_2) \to (c_2 \Leftrightarrow c_3) \to (c_1 \Leftrightarrow c_3) &
\end{array}
$$

# The Role of Univalence

- On the semantic side we pick a base type (e.g., *Bool*), all equivalences on it, all equivalences on the equivalences, and so on… until the equivalences become trivial
- Univalence ensures every equivalence at every level is expressible as a reversible program
- Gives us syntax, i.e., an induction principle, for equivalences

# A General Correspondence

- Correspondence generalizes beyond Π2
- Plugging F=Bool in $\Sigma_{(A:U)} \| F \equiv A \|$ gives us Π2 (confirmed)
- Plugging F={all finite types} gives us Π (almost confirmed)
- Plugging F={torus} gives us ???

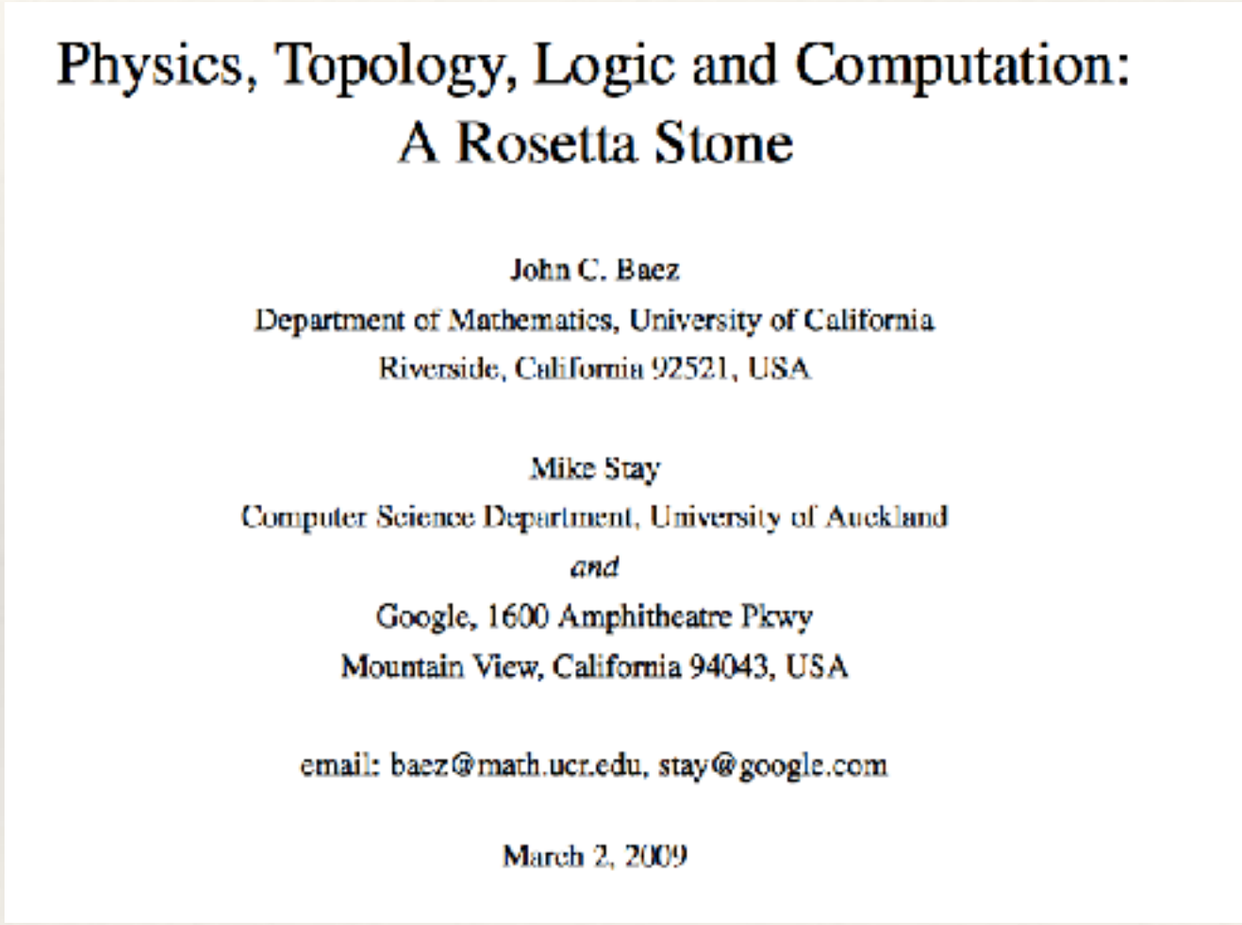
# General Correspondence

- A simple computational interpretation of paths as reversible programs at the appropriate level
- Univalence guarantees completeness: every equivalence is expressible as a reversible program
- A recipe for giving syntactic descriptions to various "beasts" such as the sub-universe of finitely 1-truncated higher inductive types ??

# Conclusions

# What have we learned

- One way to understand univalence is to study reversible languages; no topology required
- One way to systematically specify / generate reversible languages with guaranteed properties is via univalent fibrations
- Opens the door for a wealth of opportunities: taking concepts from one side to the other
- Intriguing and fascinating convergence of ideas from physics, computation, logic, and topology



Physics, Topology, Logic and Computation:
A Rosetta Stone

John C. Baez
Department of Mathematics, University of California
Riverside, California 92521, USA

Mike Stay
Computer Science Department, University of Auckland
*and*
Google, 1600 Amphitheatre Pkwy
Mountain View, California 94043, USA

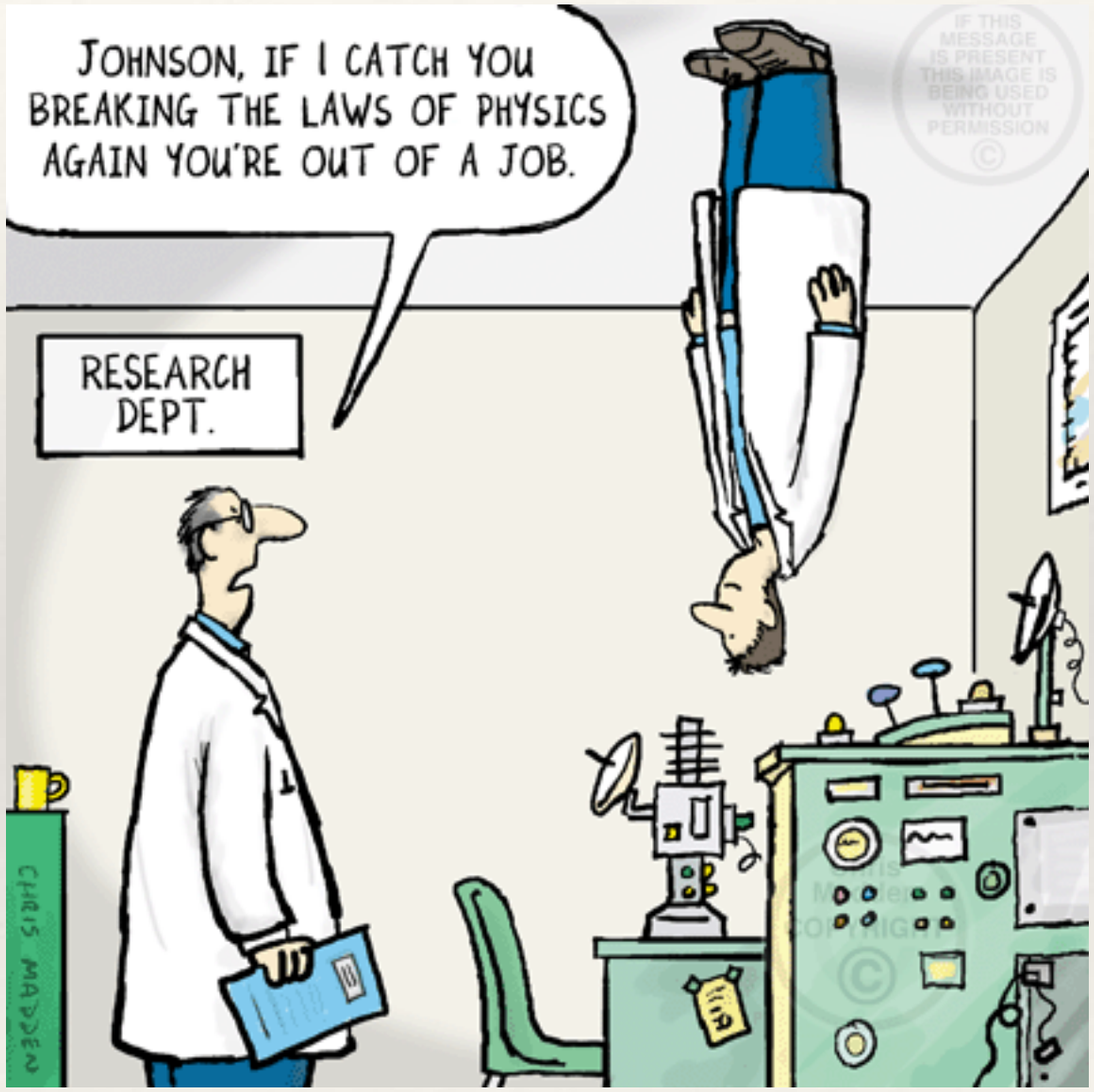
email: baez@math.ucr.edu, stay@google.com

March 2, 2009

# Back to Physics

Fascinating research program that I hope you will consider

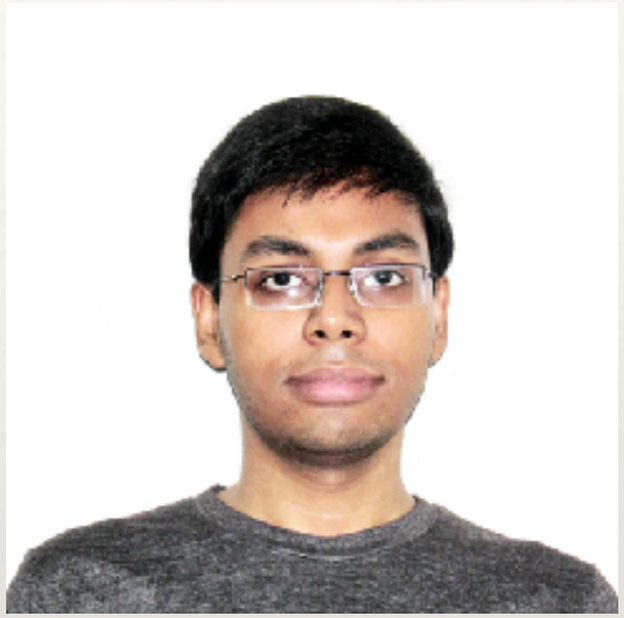
Making the foundations of computing / mathematics / logic more and more in harmony with the laws of physics



# Status of Π Family of Languages

- Fragment with finite types universal for reversible combinational circuits
- Curry-Howard correspondence gives a variant of linear logic: a superstructural reversible logic
- Extension with recursion Turing-complete reversible language
- Extension with “exclusive disjunctions” corresponds to quantum computing over a finite field
- Extension with isomorphisms between isomorphisms gives a reversible language for program transformations (optimizations) that is sound and complete with respect to appropriate univalent universes
- Extensions with exotic types coming from HoTT (work in progress)

# Team



# Fun: Negative, Fractional, Irrational, and Imaginary Types

```
data Tree = Leaf | Node Tree Tree
```

$$t = 1 + t^2$$

$$t^2 = t - 1$$

$$t^3 = t^2 - t$$

$$t^3 = -1$$

$$t^6 = 1$$

$$t^7 = t$$

$$t^2 - t + 1 = 0$$

$$t = \frac{1 \pm i\sqrt{3}}{2}$$

SEVEN TREES IN ONE

ANDREAS BLASS

Objects of Categories as Complex Numbers

Marcelo Fiore*
Computer Laboratory, University of Cambridge
United Kingdom
Marcelo.Fiore@cl.cam.ac.uk

Tom Leinster†
Institut des Hautes Études Scientifiques
France
leinster@ihes.fr