Programming and Proving with Higher Inductive Types

Dan Licata

Wesleyan University
Department of Mathematics and Computer Science

Constructive Type Theory

[Martin-Löf]

Three senses of constructivity:

Constructive Type Theory

[Martin-Löf]

Three senses of constructivity:

- Non-affirmation of certain classical principles provides **axiomatic freedom**

Synthetic geometry

Euclid’s postulates

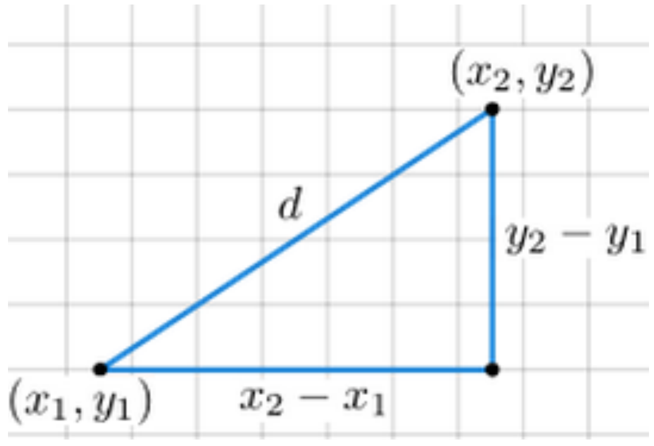
1. To draw a straight line from any point to any point.
2. To produce a finite straight line continuously in a straight line.
3. To describe a circle with any center and distance.
4. That all right angles are equal to one another.
5. That, if a straight line falling on two straight lines make the interior angles on the same side less than to right angles,the two straight lines, if produced indefinitely, meet on that side on which are the angles less than the to right angles.

Synthetic geometry

Euclid's postulates

1. To draw a straight line from any point to any point.
2. To produce a finite straight line continuously in a straight line.
3. To describe a circle with any center and distance.
4. That all right angles are equal to one another.
5. That, if a straight line falling on two straight lines make the interior angles on the same side less than to right angles,the two straight lines, if produced indefinitely, meet on that side on which are the angles less than the to right angles.

Cartesian



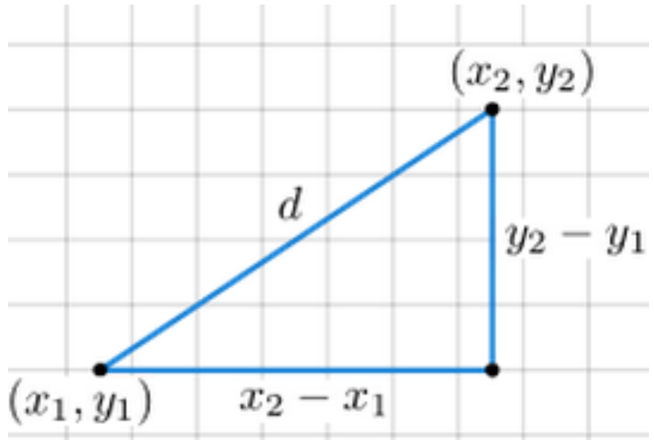
Synthetic geometry

Euclid's postulates

1. To draw a straight line from any point to any point.
2. To produce a finite straight line continuously in a straight line.
3. To describe a circle with any center and distance.
4. That all right angles are equal to one another.
5. That, if a straight line falling on two straight lines make the interior angles on the same side less than to right angles,the two straight lines, if produced indefinitely, meet on that side on which are the angles less than the to right angles.

models

Cartesian



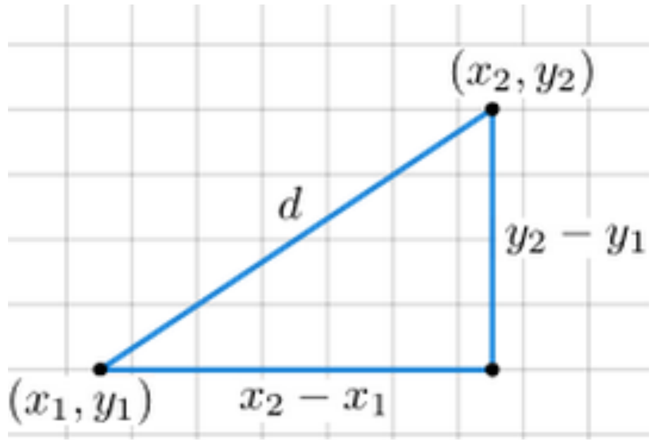
Synthetic geometry

Euclid's postulates

1. **To draw a straight line from any point to any point.**
2. **To produce a finite straight line continuously in a straight line.**
3. **To describe a circle with any center and distance.**
4. **That all right angles are equal to one another.**

models

Cartesian



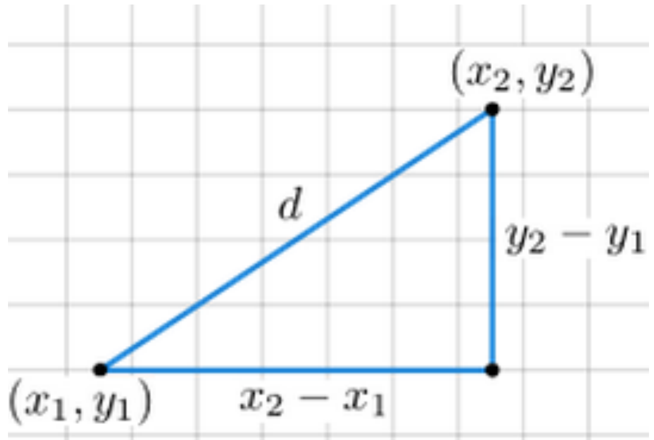
Synthetic geometry

Euclid's postulates

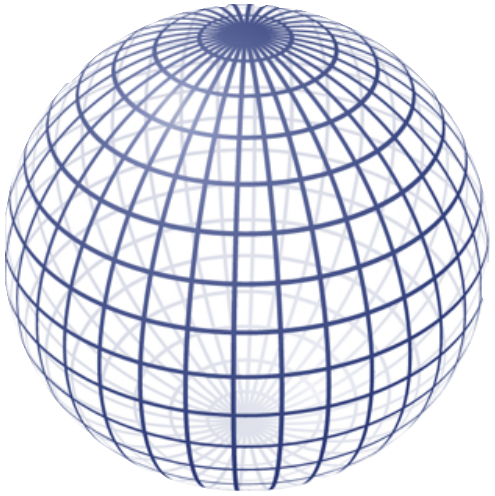
1. **To draw a straight line from any point to any point.**
2. **To produce a finite straight line continuously in a straight line.**
3. **To describe a circle with any center and distance.**
4. **That all right angles are equal to one another.**

models

Cartesian



Spherical



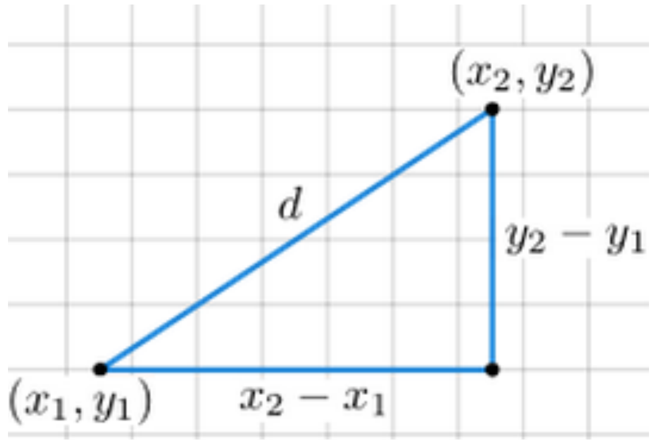
Synthetic geometry

Euclid's postulates

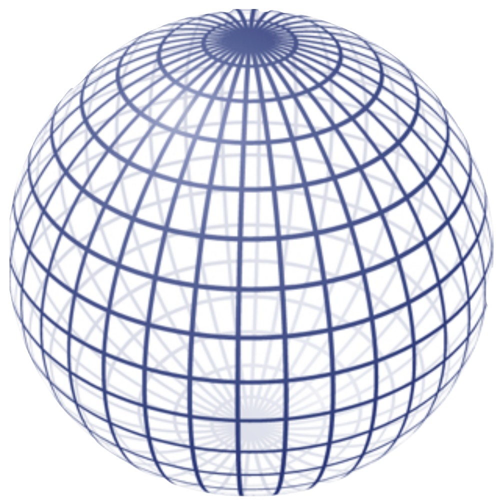
1. To draw a straight line from any point to any point.
2. To produce a finite straight line continuously in a straight line.
3. To describe a circle with any center and distance.
4. That all right angles are equal to one another.
5. Two distinct lines meet at two antipodal points.

models

Cartesian



Spherical



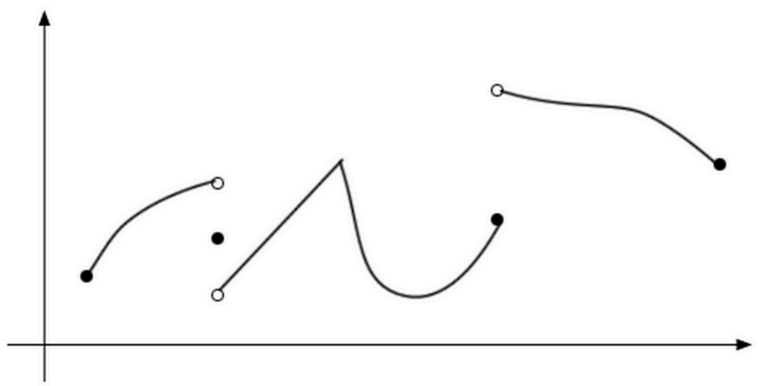
Synthetic mathematics

Type theory

1. $\tau ::= b \mid \tau_1 \to \tau_2$
2. $e ::= x \mid e_1\ e_2 \mid \lambda x.e$
3. $(\lambda x.e)e_2 = e[e_2/x]$

Synthetic mathematics

Set-theoretic functions



Type theory

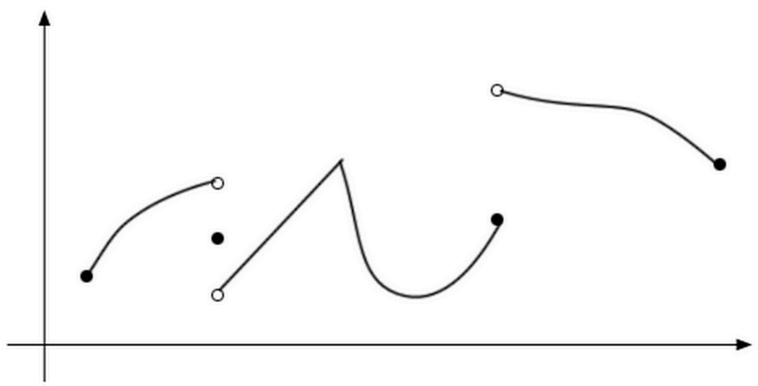
1. $\tau ::= b \mid \tau_1 \to \tau_2$
2. $e ::= x \mid e_1\ e_2 \mid \lambda x.e$
3. $(\lambda x.e)e_2 = e[e_2/x]$

Synthetic mathematics

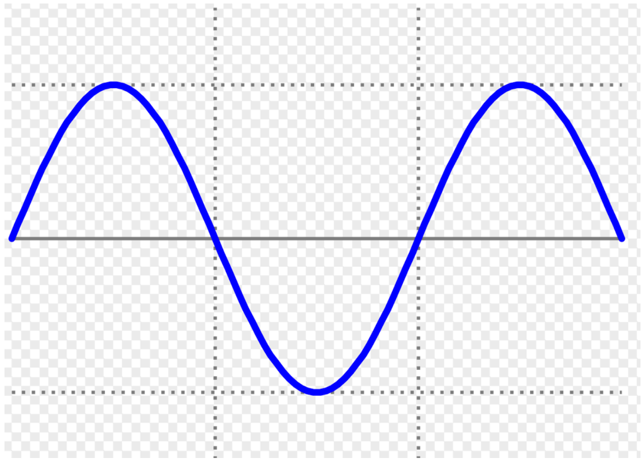
Type theory

1. $\tau ::= b \mid \tau_1 \to \tau_2$
2. $e ::= x \mid e_1\ e_2 \mid \lambda x.e$
3. $(\lambda x.e)e_2 = e[e_2/x]$

Set-theoretic functions



Domain-theoretic functions

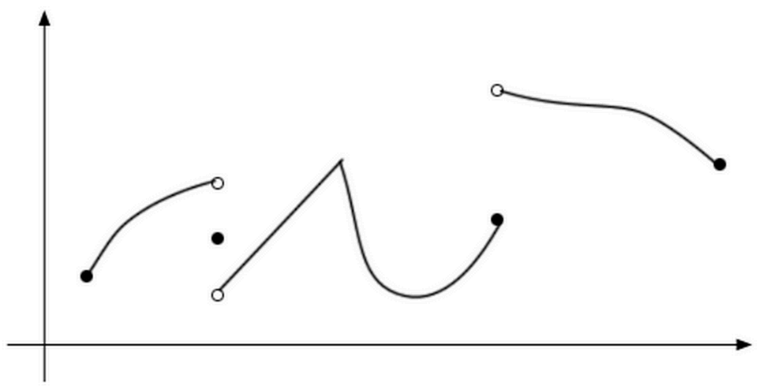


Synthetic mathematics

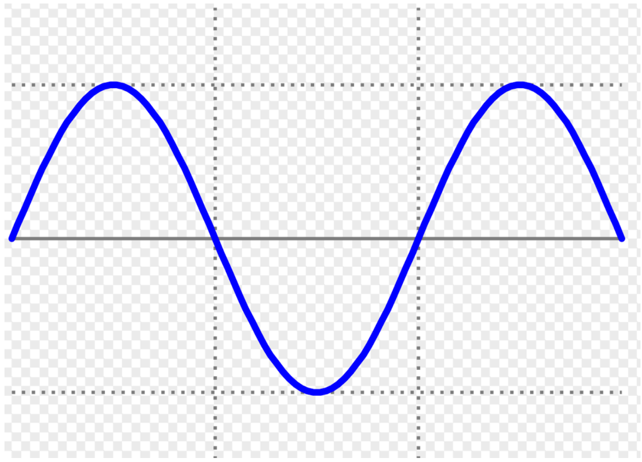
Type theory

1. $\tau ::= b \mid \tau_1 \to \tau_2$
2. $e ::= x \mid e_1\ e_2 \mid \lambda x.e$
3. $(\lambda x.e)e_2 = e[e_2/x]$
4. $Y(f) = f(Y(f))$

Set-theoretic functions



Domain-theoretic functions



Constructive Type Theory

Three senses of constructivity:

Constructive Type Theory

Three senses of constructivity:

- Non-affirmation of certain classical principles provides **axiomatic freedom**

Constructive Type Theory

Three senses of constructivity:

- Non-affirmation of certain classical principles provides **axiomatic freedom**

- **Computational interpretation** supports software verification and proof automation

Computational Interpretation

There is an algorithm that, given a closed program `e : bool`, computes either an equality `e = true`, or an equality `e = false`.

Computational Interpretation

There is an algorithm that, given a closed program `e : bool`, computes either an equality `e = true`, or an equality `e = false`.

- Requires functions with arbitrary domain/range to be computable, but stating theorem for `bool` offers some flexibility

Computational Interpretation

There is an algorithm that, given a closed program `e : bool`, computes either an equality `e = true`, or an equality `e = false`.

- Requires functions with arbitrary domain/range to be computable, but stating theorem for `bool` offers some flexibility
- Basis for software verification and proof automation

Constructive Type Theory

Three senses of constructivity:

- Non-affirmation of certain classical principles provides **axiomatic freedom**
- **Computational interpretation** supports software verification and proof automation

Constructive Type Theory

Three senses of constructivity:

- Non-affirmation of certain classical principles provides **axiomatic freedom**
- **Computational interpretation** supports software verification and proof automation
- Props-as-types allows **proof-relevant mathematics**

Proof relevance

```
x : A
```

Proof relevance

`x : A`

$x =_A y$ equality type

Proof relevance

$x : A$

$p : x =_A y$ equality type

Proof relevance

`x : A`

$p : x =_A y$ equality type

Any structure or property `C` can be transported along an equality

Proof relevance

$$x : A$$

$$p : x =_A y \qquad \text{equality type}$$

Any structure or property C can be transported along an equality

$$\mathrm{transport}_C(p) : C(x) \to C(y)$$

Proof relevance

$x : A$

$p : x =_A y$ equality type

Any structure or property C can be transported along an equality

Leibniz's indiscernability of identicals

$\mathrm{transport}_C(p) : C(x) \to C(y)$

Proof relevance

$x : A$

$p : x =_A y$ equality type

Any structure or property C can be transported along an equality

Leibniz's indiscernability of identicals

$\mathrm{transport}_C(p) : C(x) \to C(y)$

by a function: can it do real work?

Proof relevance

$x : A$

$p : x =_A y$ equality type

Proof relevance

$x : A$

$p : x =_A y$ equality type

$p_1 =_{x=y} p_2$

Proof relevance

$x : A$

$p : x =_A y$ equality type

$q : p_1 =_{x=y} p_2$

Proof relevance

$x : A$

$p : x =_A y$ equality type

$q : p_1 =_{x=y} p_2$

$q_1 =_{p1=p2} q_2$

Proof relevance

$x : A$

$p : x =_A y$ equality type

$q : p_1 =_{x=y} p_2$

$r : q_1 =_{p1=p2} q_2$

Proof relevance

$$x : A$$

$$p : x =_A y$$ equality type

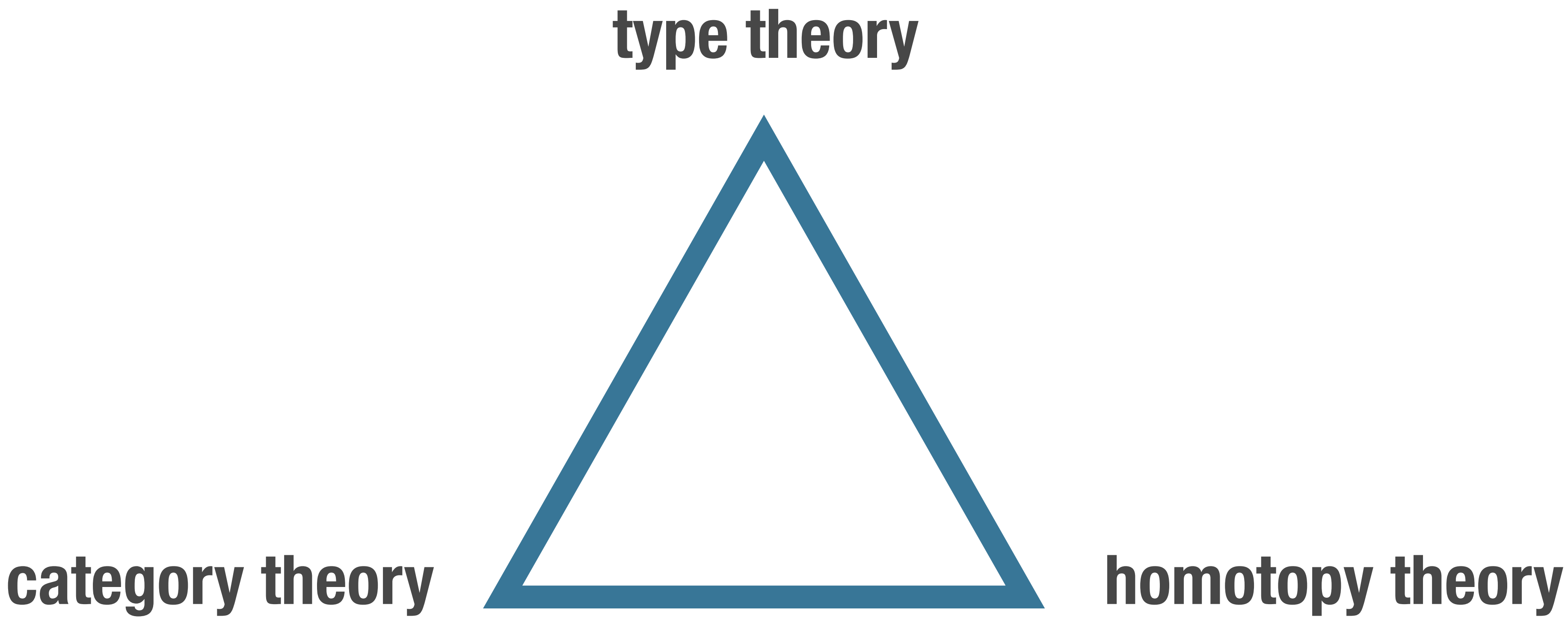
$$q : p_1 =_{x=y} p_2$$

$$r : q_1 =_{p1=p2} q_2$$

$$\vdots$$

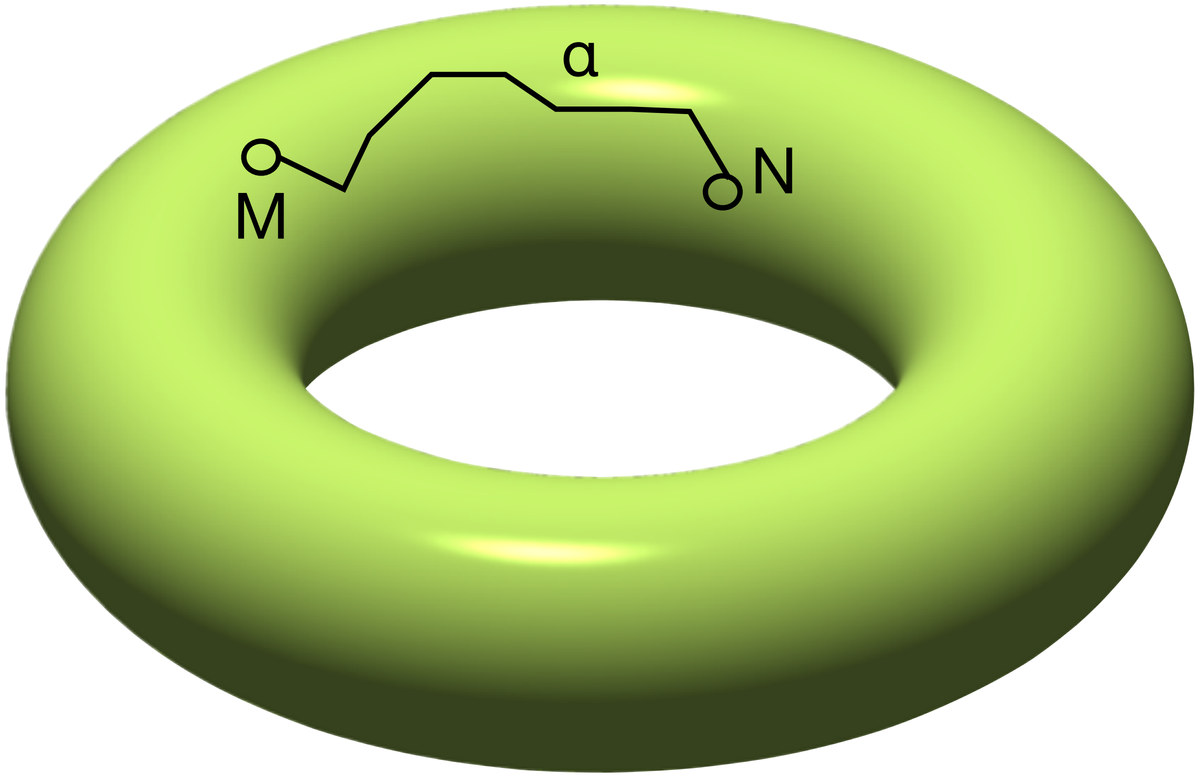
higher equalities radically expand the kind of math that can be done synthetically…

Homotopy Type Theory



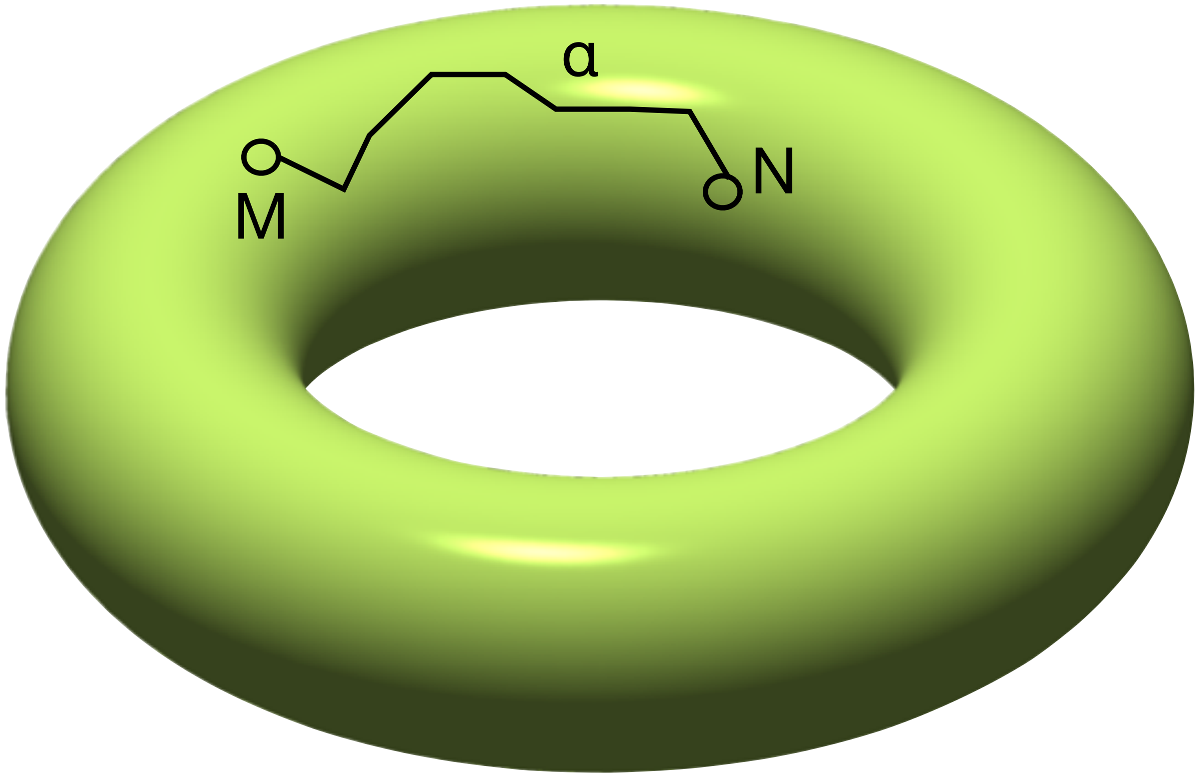
[Hofmann,Streicher,Awodey,Warren,Voevodsky
Lumsdaine,Gambino,Garner,van den Berg]

Types as spaces



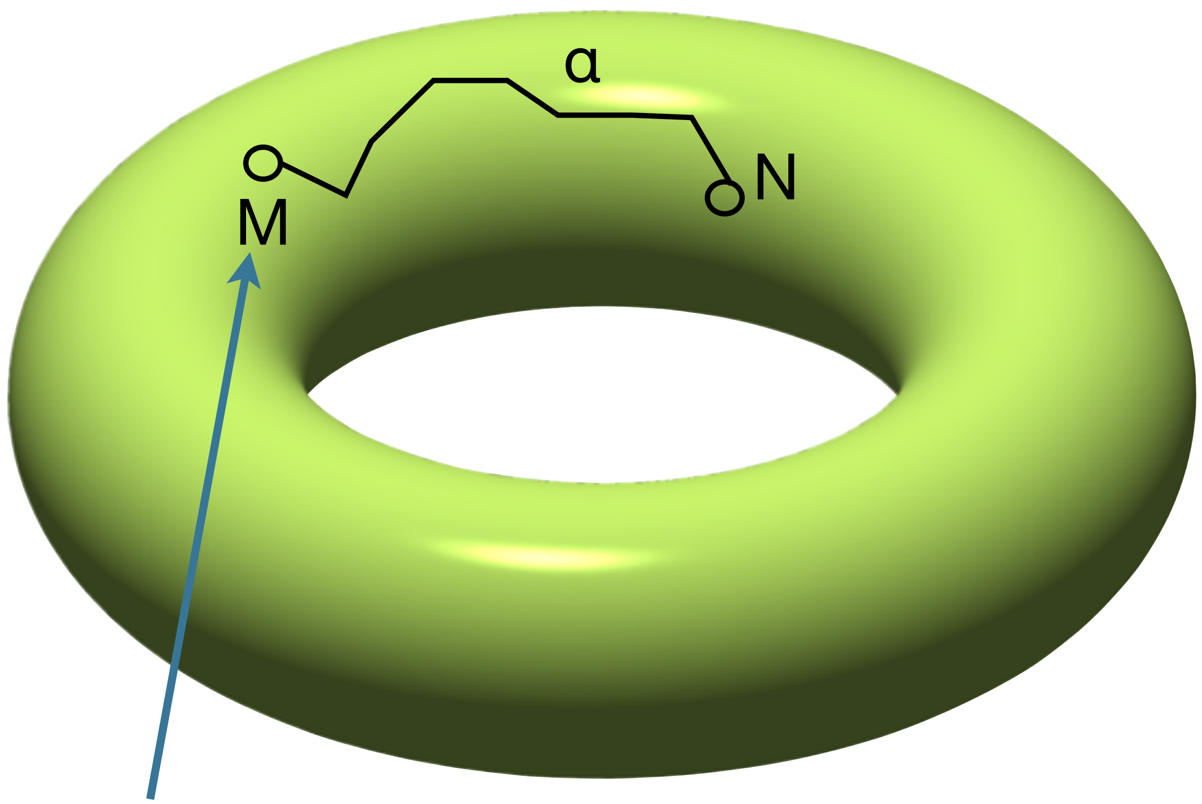
Types as spaces

type A is a space



Types as spaces

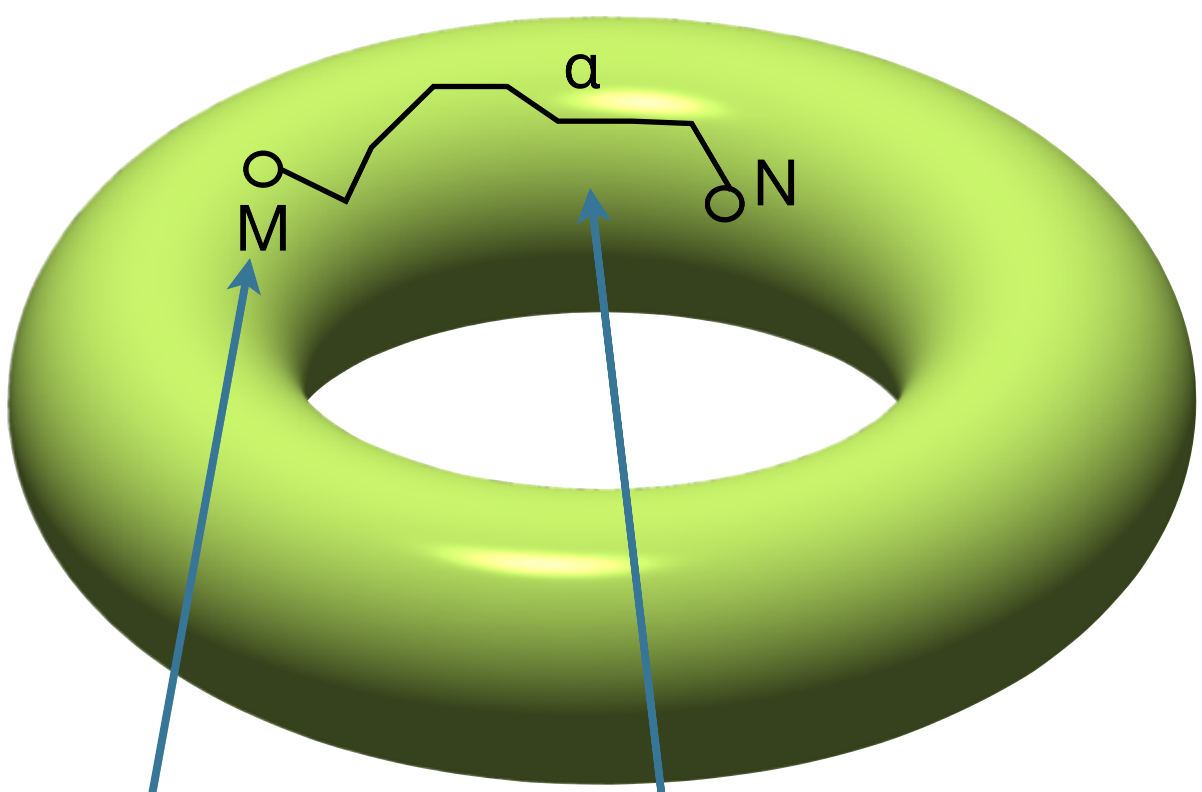
type A **is a space**



programs
M : A
are points

Types as spaces

type A is a space



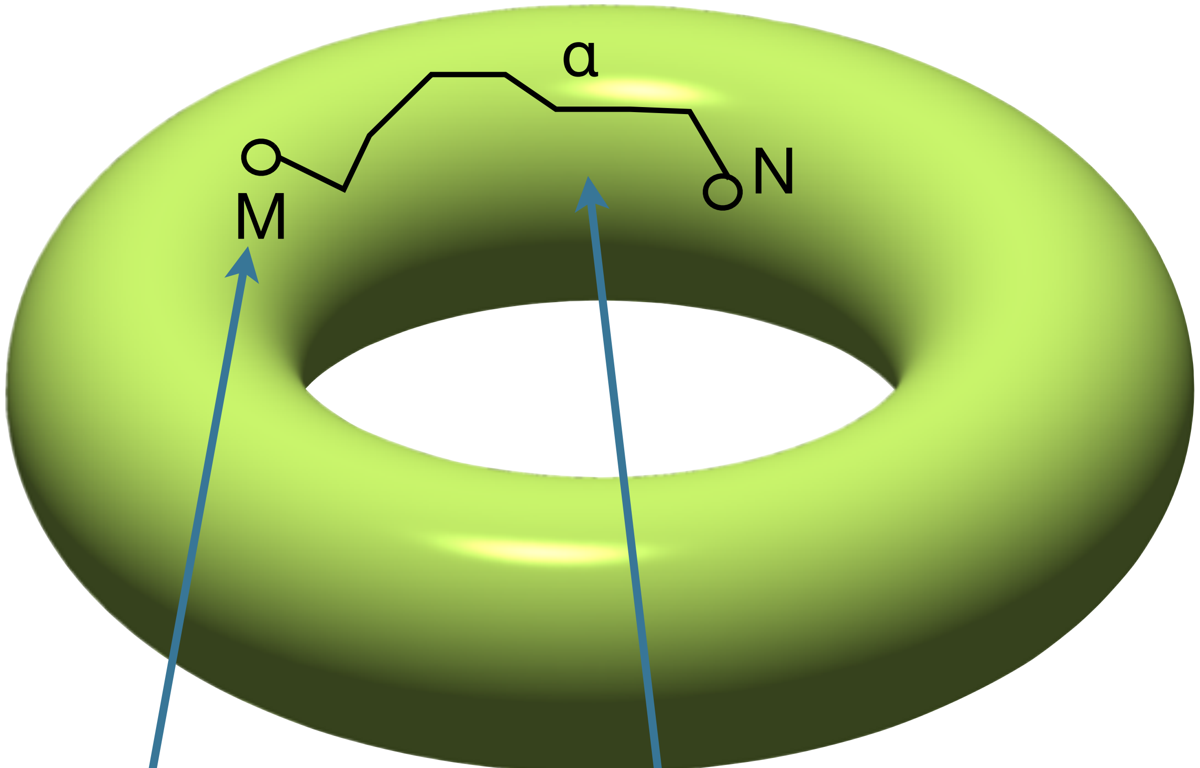
programs
$M : A$
are points

proofs of equality
$\alpha \ : \ M =_A N$
are paths

Types as spaces

type A is a space

path operations

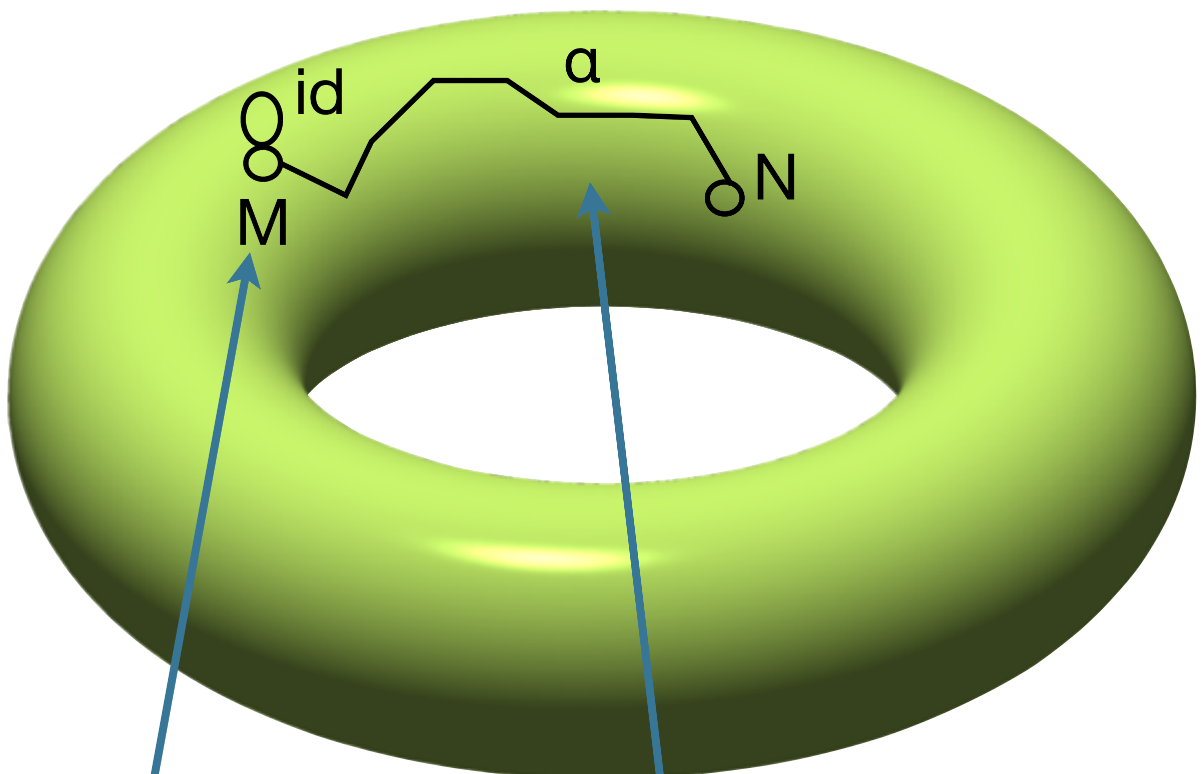


programs
$M : A$
are points

proofs of equality
$\alpha \ : \ M =_A \ N$
are paths

Types as spaces

type A is a space



programs
$M : A$
are points

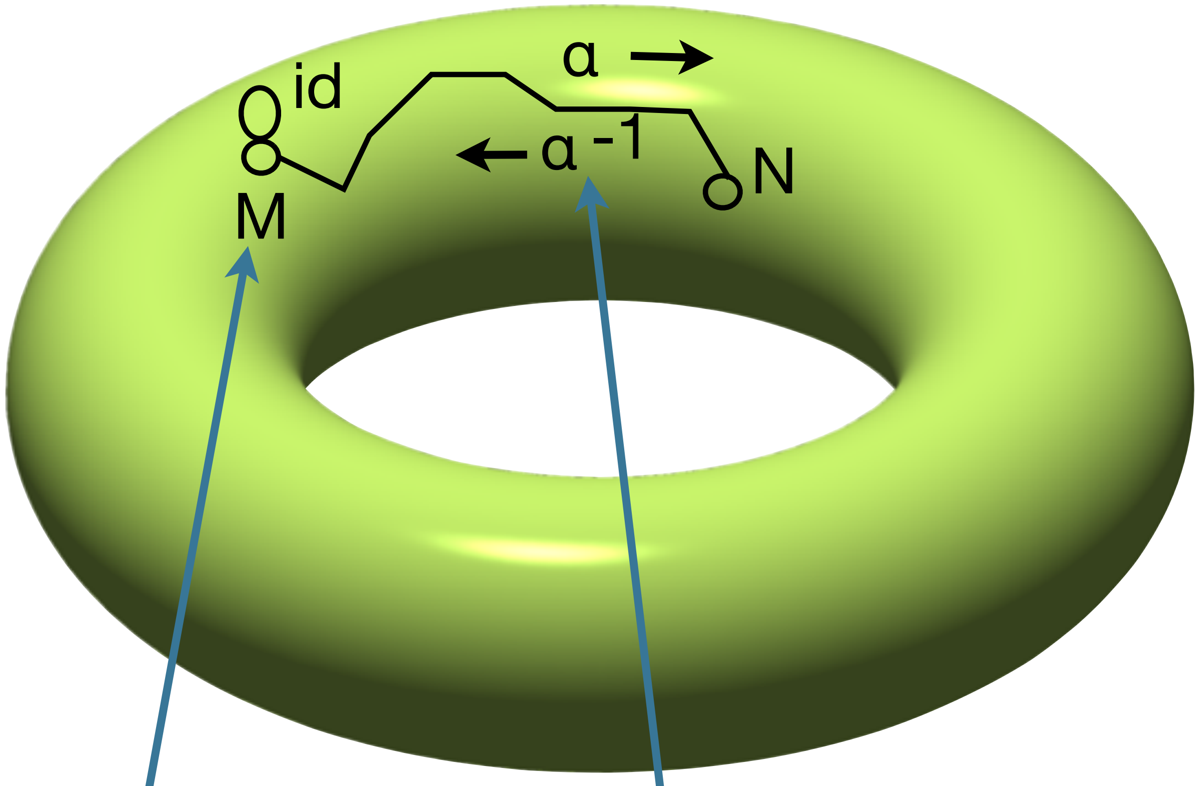
proofs of equality
$\alpha \;:\; M =_A N$
are paths

path operations

```
id      : M = M (refl)
```

Types as spaces

type A is a space



programs
$M : A$
are points

proofs of equality
$\alpha \ : \ M =_A N$
are paths

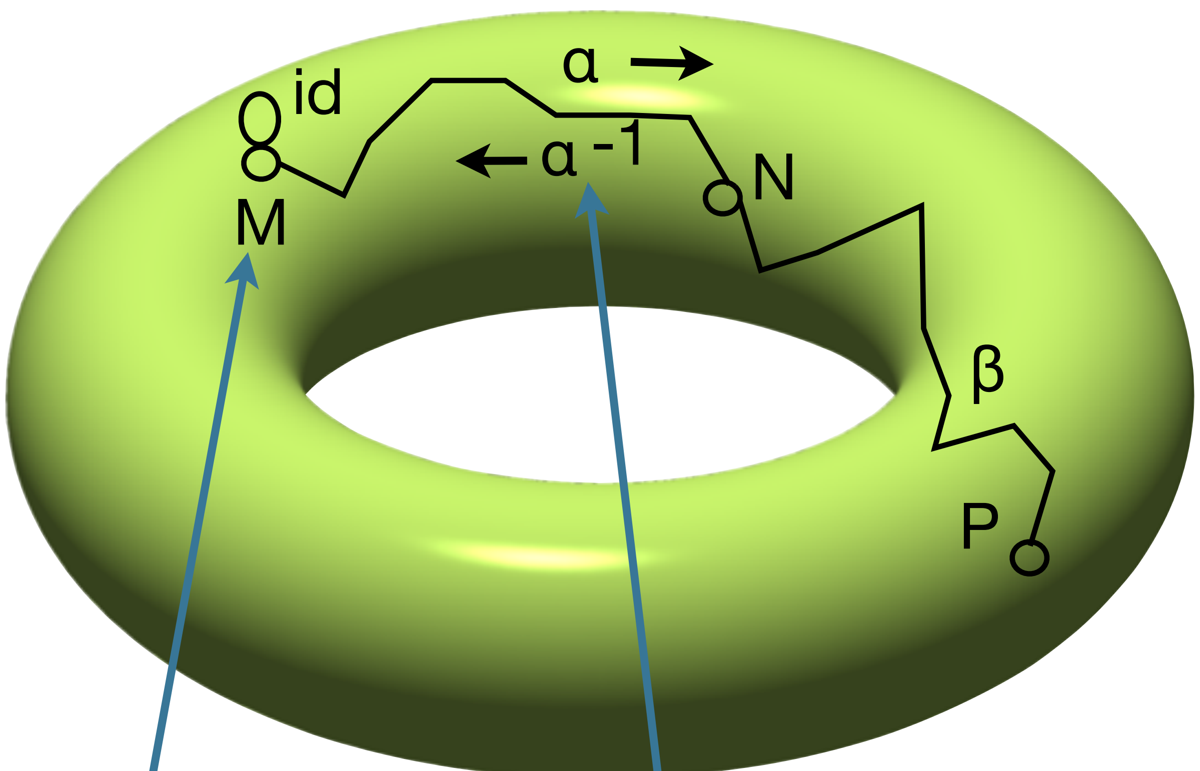
path operations

$\mathrm{id} \quad : M = M$ (refl)

$\alpha^{-1} \quad : N = M$ (sym)

Types as spaces

type A **is a space**



programs
M : A
are points

proofs of equality
$\alpha : M =_A N$
are paths

path operations

```
id      : M = M (refl)
α⁻¹     : N = M (sym)
β o α   : M = P (trans)
```

Homotopy

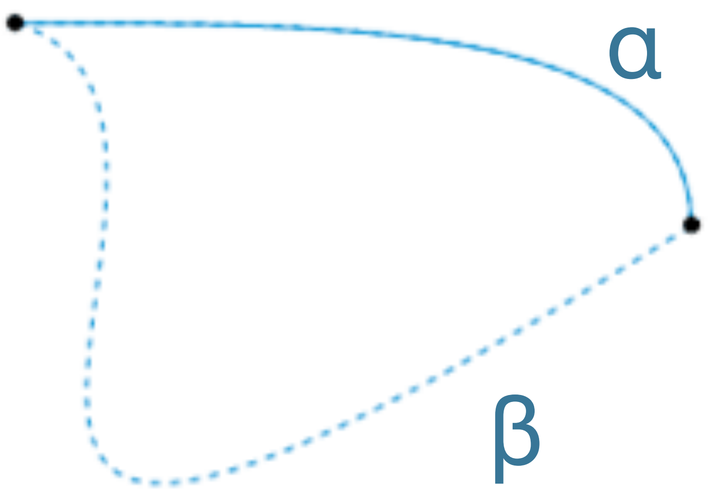
Deformation of one path into another

α

β

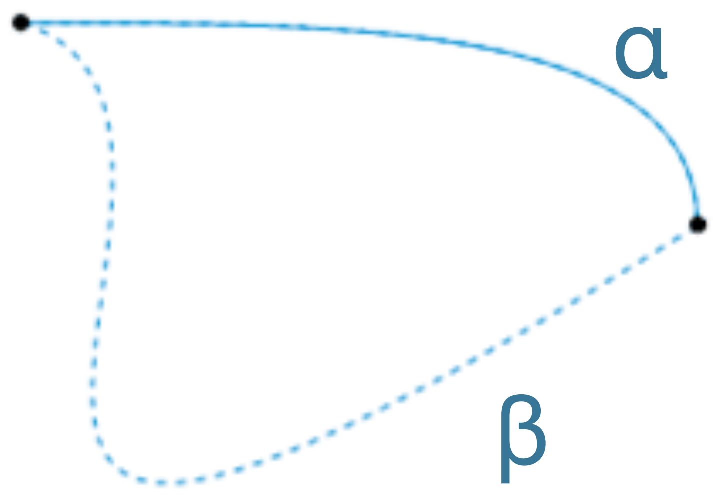
Homotopy

Deformation of one path into another



Homotopy

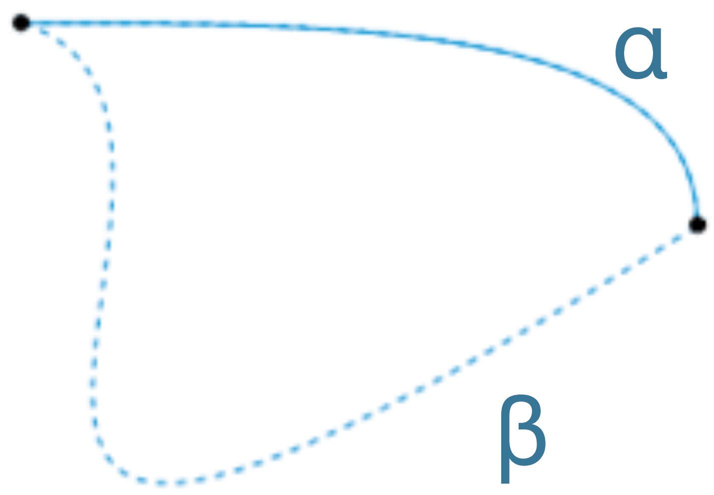
Deformation of one path into another



= 2-dimensional *path between paths*

Homotopy

Deformation of one path into another

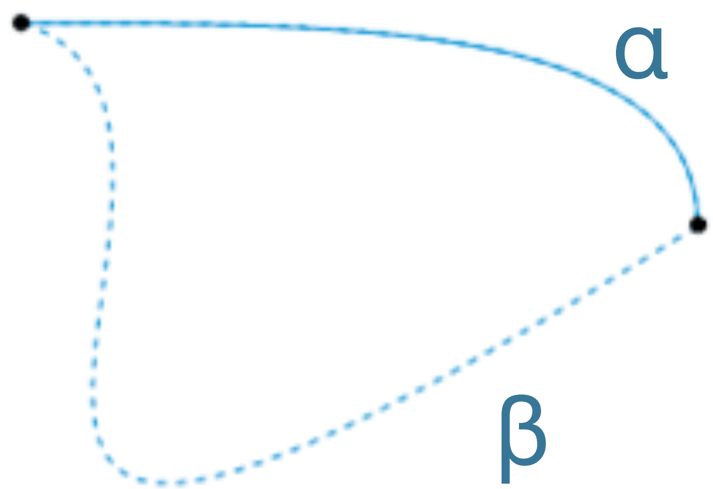


$\delta \;:\; \alpha =_{x=y} \beta$

= 2-dimensional *path between paths*

Homotopy

Deformation of one path into another



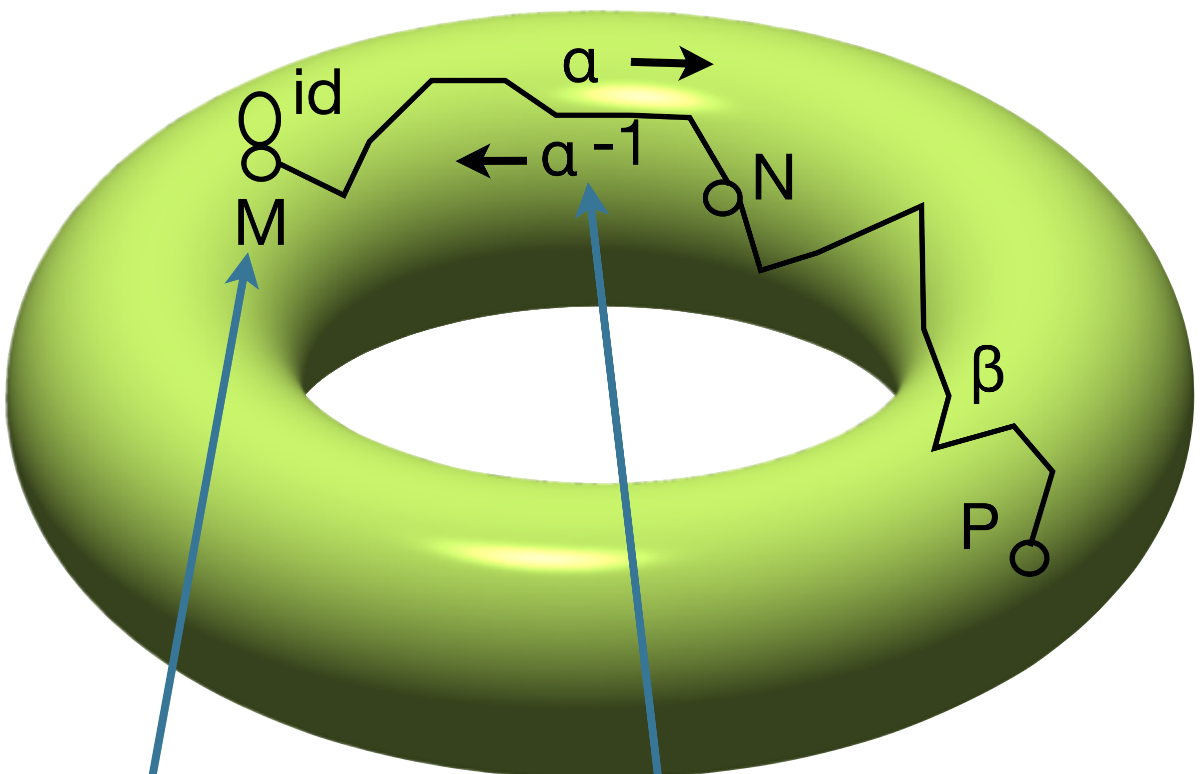
$\delta : \alpha =_{x=y} \beta$

= 2-dimensional *path between paths*

Then homotopies between homotopies ….

Types as spaces

type A **is a space**



programs
$M : A$
are points

proofs of equality
$\alpha \ : \ M =_A N$
are paths

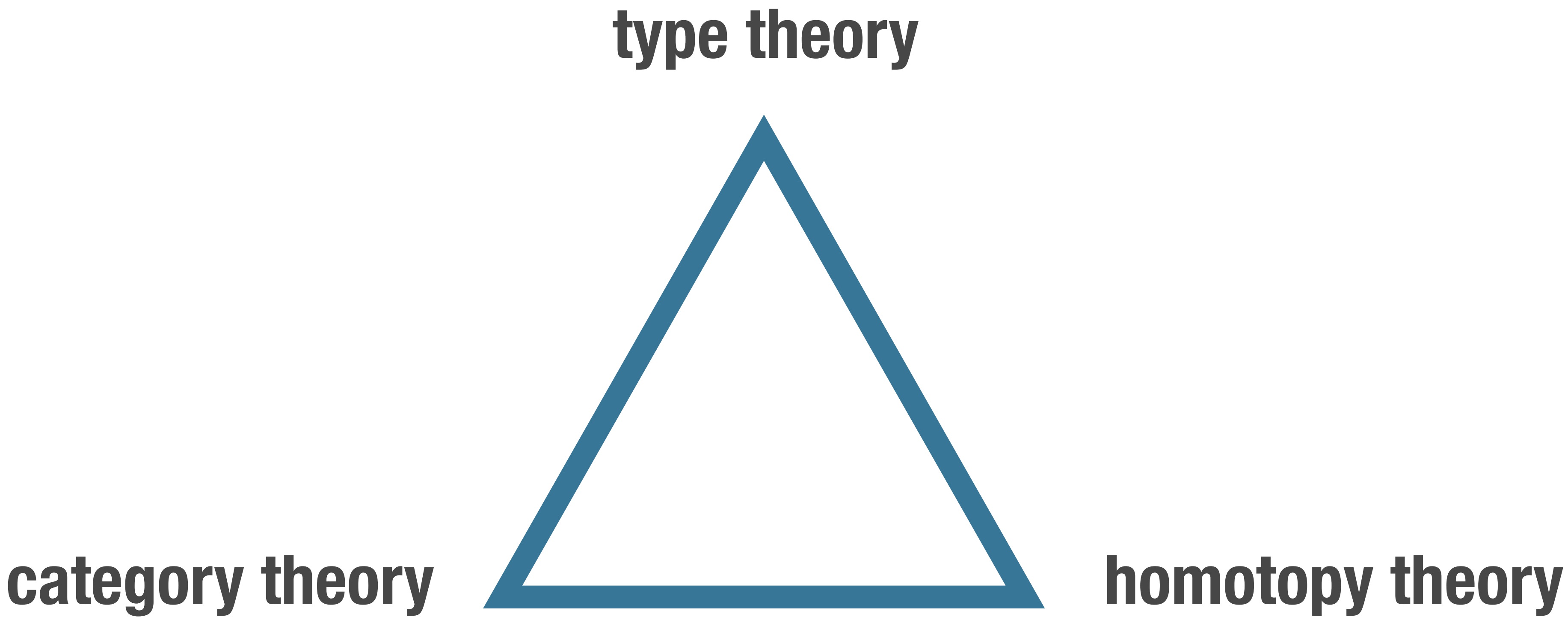
path operations

$\mathrm{id} : M = M$ (refl)
$\alpha^{-1} : N = M$ (sym)
$\beta \circ \alpha : M = P$ (trans)

homotopies

$\mathrm{ul} : \mathrm{id} \circ \alpha =_{M=N} \alpha$
$\mathrm{il} : \alpha^{-1} \circ \alpha =_{M=M} \mathrm{id}$
$\mathrm{asc} : \gamma \circ (\beta \circ \alpha)$
$=_{M=P} (\gamma \circ \beta) \circ \alpha$

Homotopy Type Theory



[Hofmann,Streicher,Awodey,Warren,Voevodsky
Lumsdaine,Gambino,Garner,van den Berg]

Types as ∞-groupoids

type A is an ∞-groupoid

- infinite-dimensional algebraic structure, with morphisms, morphisms between morphisms, ...
- each level has a groupoid structure, and they interact

morphisms

$\mathrm{id} : M = M$ (refl)

$\alpha^{-1} : N = M$ (sym)

$\beta \circ \alpha : M = P$ (trans)

morphisms between morphisms

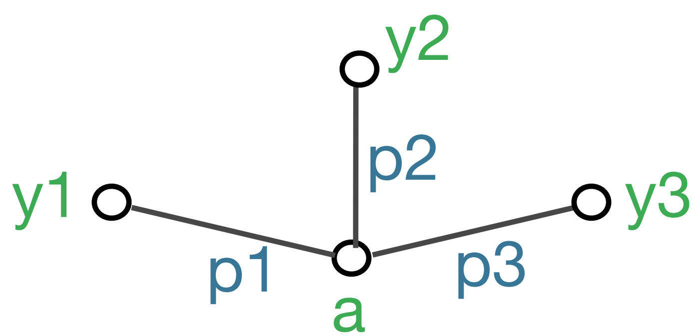
$\mathrm{ul} : \mathrm{id} \circ \alpha =_{M=N} \alpha$

$\mathrm{il} : \alpha^{-1} \circ \alpha =_{M=M} \mathrm{id}$

$\mathrm{asc} : \gamma \circ (\beta \circ \alpha)$
$\quad =_{M=P} (\gamma \circ \beta) \circ \alpha$

Path induction

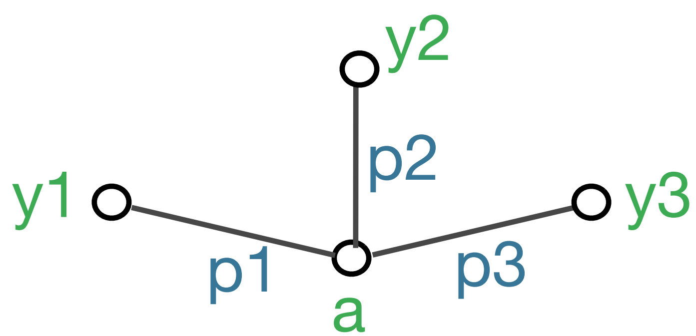
Type of paths from *a* to somewhere



is inductively generated by

Path induction

Type of paths from a to somewhere



is inductively generated by

Fix a type A with element `a:A`.

For a family of types `C(y:A, p:a=y)`,

to give an element of

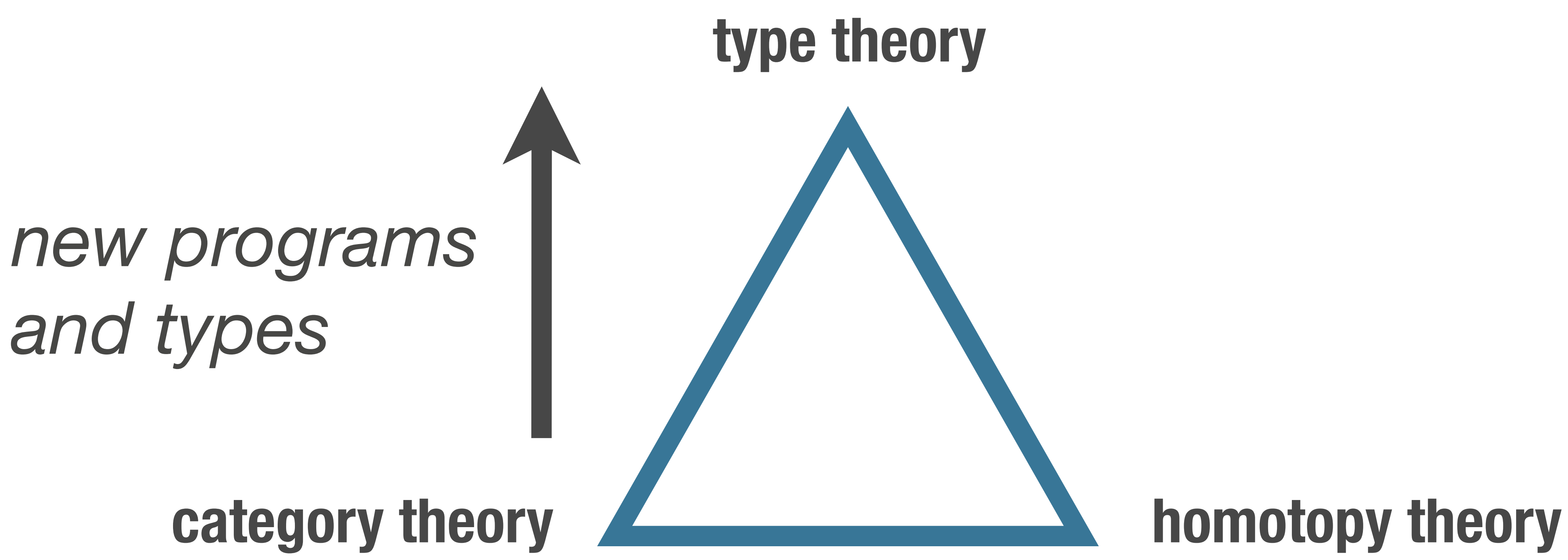
`C(y,p)` for all `y` and `p:a=y`,

suffices to give an element of

`C(a,id)`

Type theory is a synthetic theory of spaces/∞-groupoids

Homotopy Type Theory



Univalence [Voevodsky]

Univalence [Voevodsky]

- *Equivalence of types* is a generalization to spaces of bijection of sets

Univalence [Voevodsky]

- *Equivalence of types* is a generalization to spaces of bijection of sets

- Univalence axiom:
  equality of types (`A =Type B`) is (equivalent to) equivalence of types (`Equiv A B`)

Univalence [Voevodsky]

- *Equivalence of types* is a generalization to spaces of bijection of sets
- Univalence axiom:
  equality of types (`A =Type B`) is (equivalent to) equivalence of types (`Equiv A B`)
- ∴ all structures/properties respect equivalence

Univalence [Voevodsky]

- *Equivalence of types* is a generalization to spaces of bijection of sets
- Univalence axiom:
  equality of types (`A =`$_{\texttt{Type}}$ `B`) is (equivalent to) equivalence of types (`Equiv A B`)
- ∴ all structures/properties respect equivalence
- Not by collapsing equivalence,
  but by exploiting proof-relevant equality:
  `transport` *does real work*

Higher inductive types

[Bauer,Lumsdaine,Shulman,Warren]

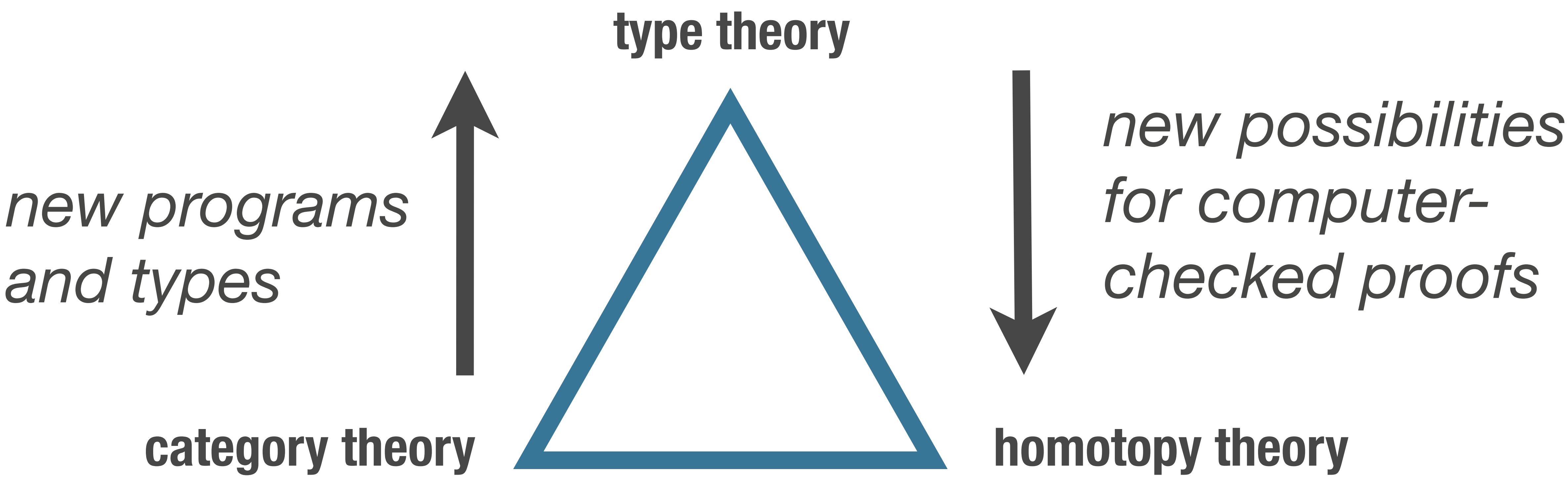
New way of forming types:

Inductive type specified by generators
not only for points (elements), but also for paths

Constructivity

- Non-affirmation of classical principles ✓
- Computational interpretation ?
- Proof-relevant mathematics ✓

Homotopy Type Theory



Outline

1. Certified homotopy theory
2. Certified software

Outline

1. **Certified homotopy theory**
2. Certified software

Homotopy Theory

A branch of topology,
the study of spaces and continuous deformations

[image from wikipedia]

Homotopy in HoTT

$\pi_1(S^1) = \mathbb{Z}$

$\pi_{k<n}(S^n) = 0$

Hopf fibration

$\pi_2(S^2) = \mathbb{Z}$

$\pi_3(S^2) = \mathbb{Z}$

James Construction

$\pi_4(S^3) = \mathbb{Z}_?$

Freudenthal

$\pi_n(S^n) = \mathbb{Z}$

K(G,n)

Cohomology axioms

Blakers-Massey

Van Kampen

Covering spaces

Whitehead for n-types

[Brunerie, Finster, Hou, Licata, Lumsdaine, Shulman]

Homotopy in HoTT

$\pi_1(S^1) = \mathbb{Z}$

$\pi_{k<n}(S^n) = 0$

Hopf fibration

$\pi_2(S^2) = \mathbb{Z}$

$\pi_3(S^2) = \mathbb{Z}$

James Construction

$\pi_4(S^3) = \mathbb{Z}_?$

Freudenthal

$\pi_n(S^n) = \mathbb{Z}$

K(G,n)

Cohomology axioms

Blakers-Massey

Van Kampen

Covering spaces

Whitehead for n-types

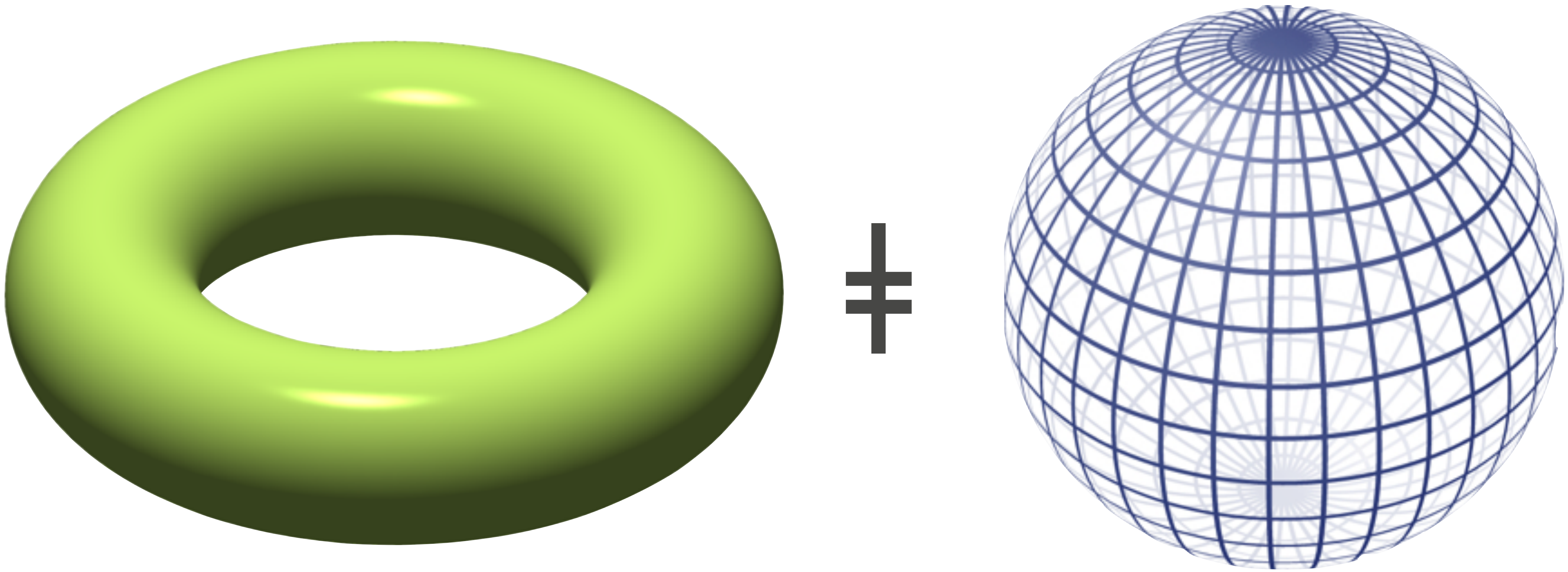
[Brunerie, Finster, Hou, Licata, Lumsdaine, Shulman]

Homotopy Groups

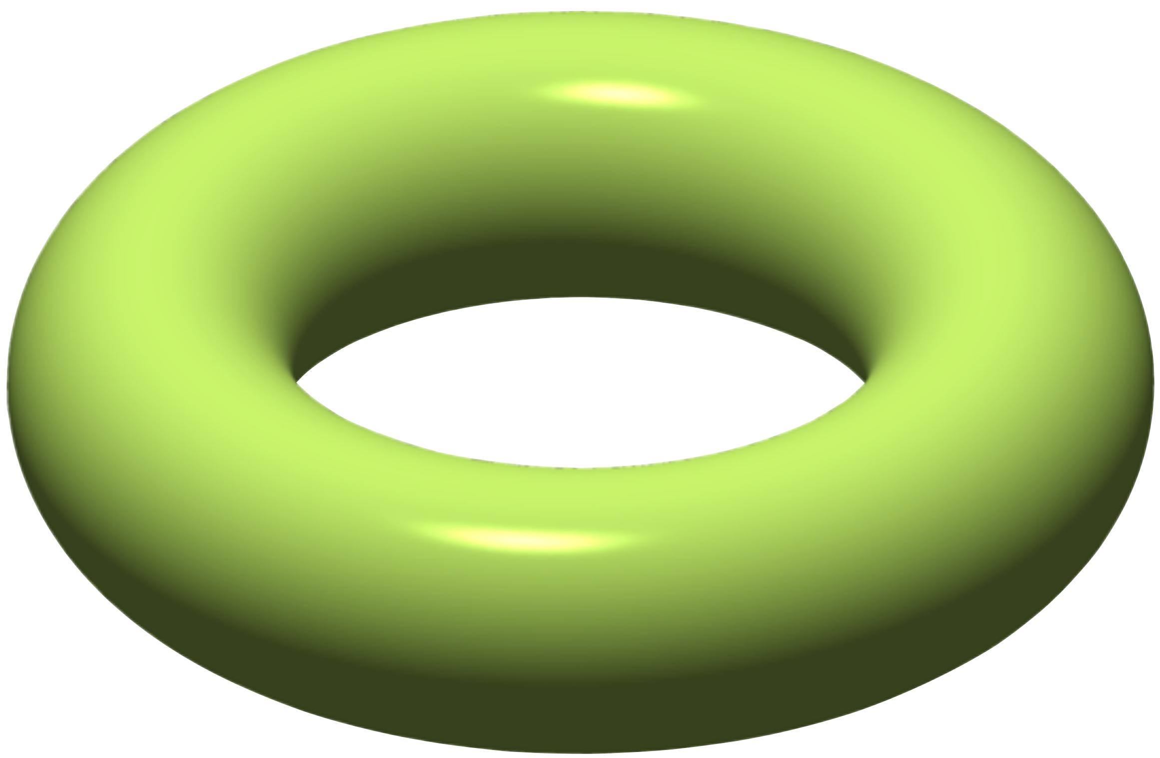
Homotopy groups of a space X:

- $\pi_1(X)$ is fundamental group (group of loops)
- $\pi_2(X)$ is group of homotopies (2-dimensional loops)
- $\pi_3(X)$ is group of 3-dimensional loops
- ...

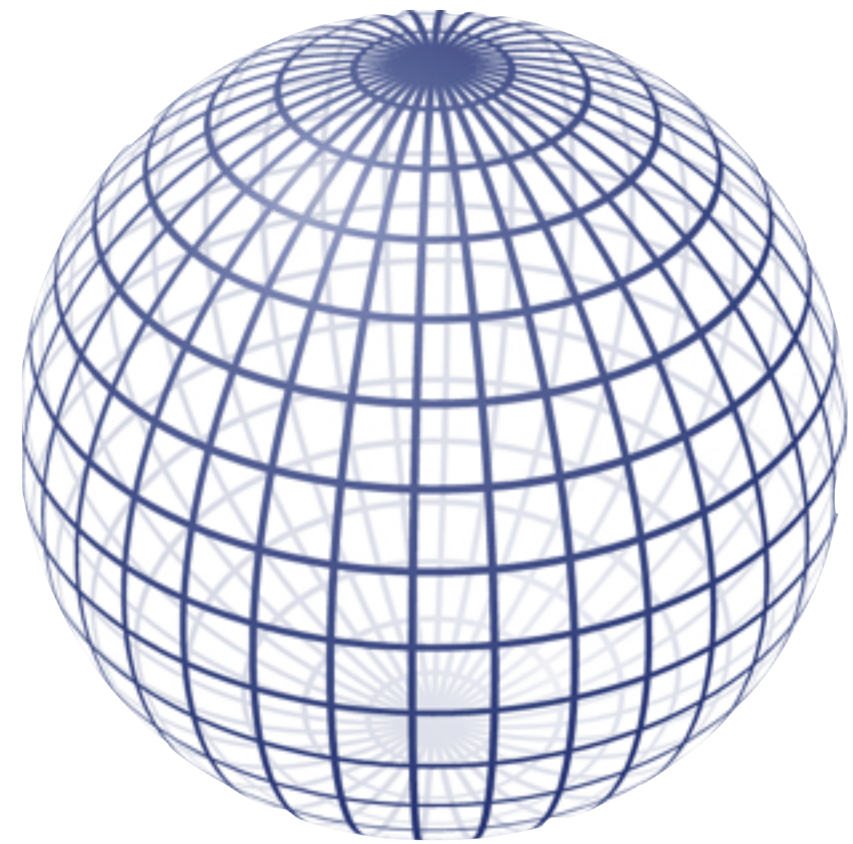
Telling spaces apart



Telling spaces apart



$\neq$

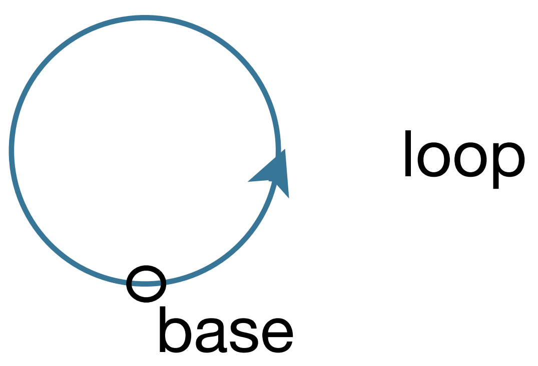


fundamental group
is non-trivial ($\mathbb{Z} \times \mathbb{Z}$)

fundamental group
is trivial

The Circle

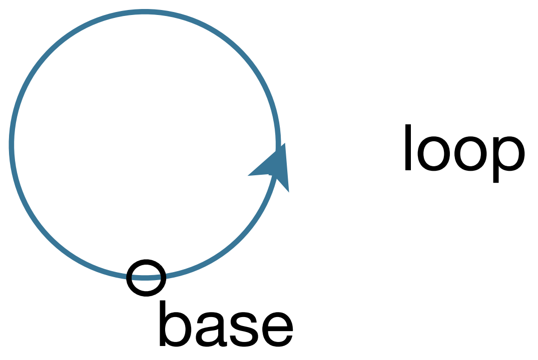
Circle S^1 is a **higher inductive type** generated by



The Circle

Circle S^1 is a **higher inductive type** generated by

```
base : S¹
loop : base = base
```

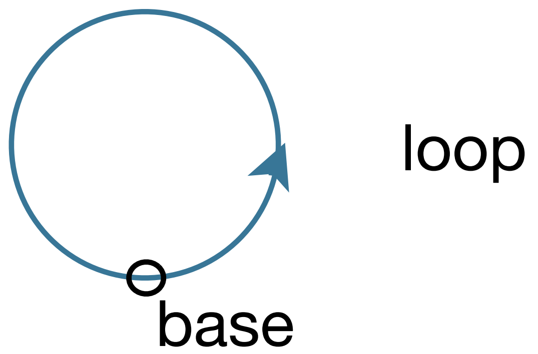


The Circle

Circle S^1 is a **higher inductive type** generated by

point `base : S¹`

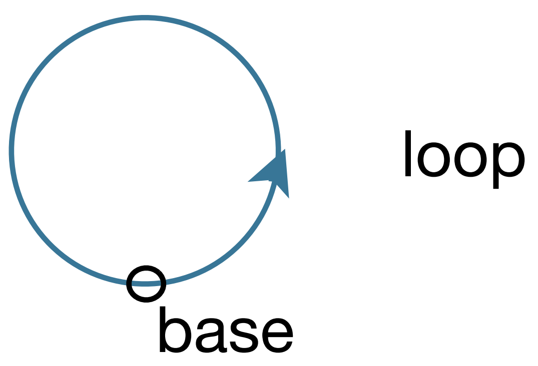
`loop : base = base`



The Circle

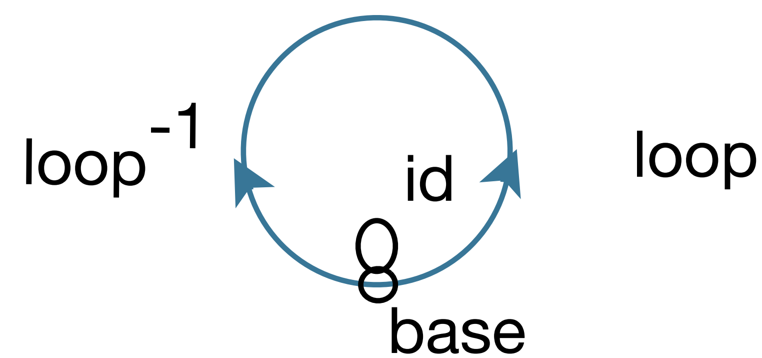
Circle S^1 is a **higher inductive type** generated by

point	`base : S`^{1}
path	`loop : base = base`



The Circle

Circle S^1 is a **higher inductive type** generated by



point `base : S¹`

path `loop : base = base`

Free type: equipped with structure

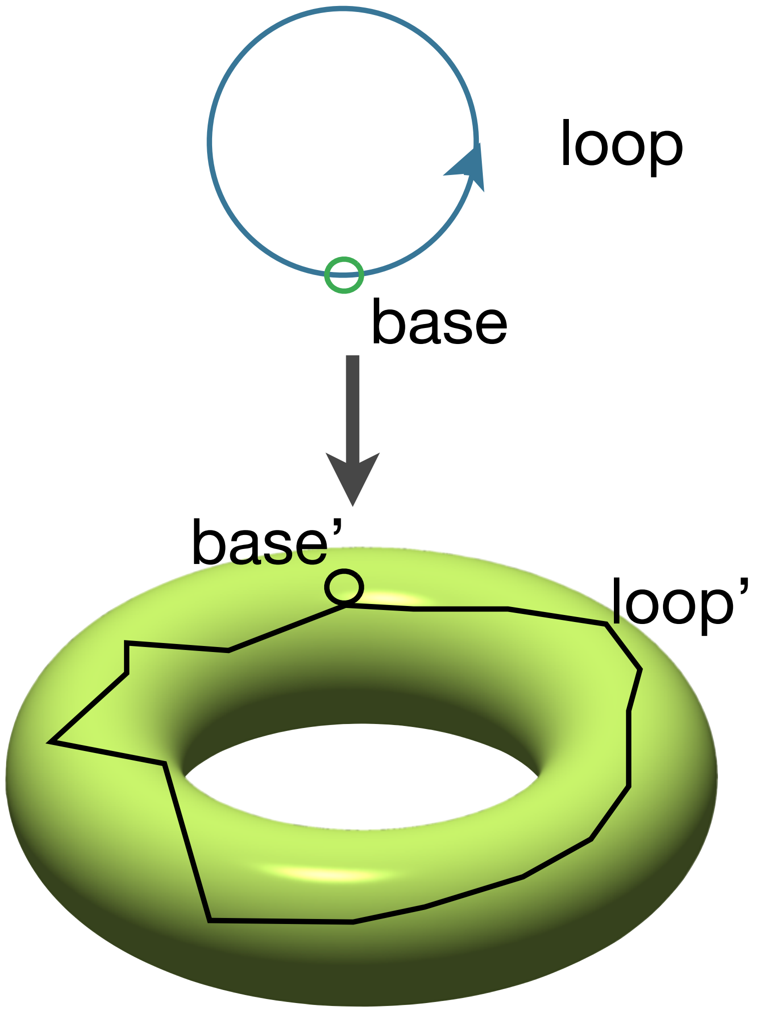
`id`	`inv : loop o loop⁻¹ = id`
`loop⁻¹`	`...`
`loop o loop`	

The Circle

Circle recursion:

function $S^1 \to X$ determined by

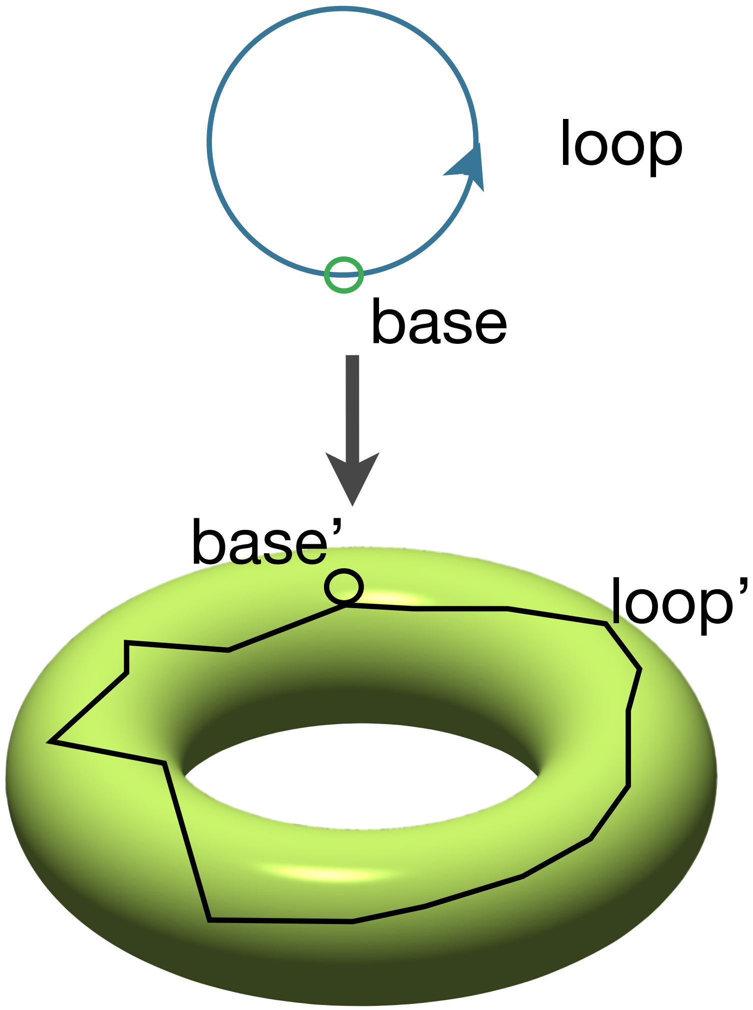
```
base' : X
loop' : base' = base'
```



The Circle

Circle recursion:
function $S^1 \to X$ determined by

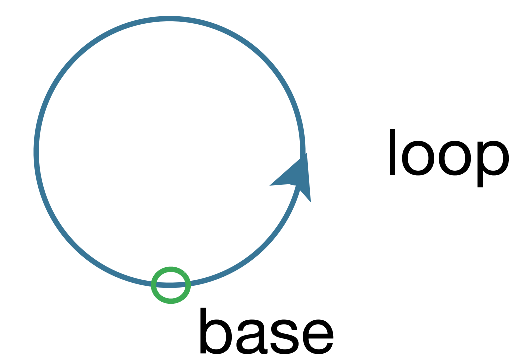
```
base' : X
loop' : base' = base'
```



Circle induction: To prove a predicate P for all points on the circle, suffices to prove P(`base`), continuously in the loop

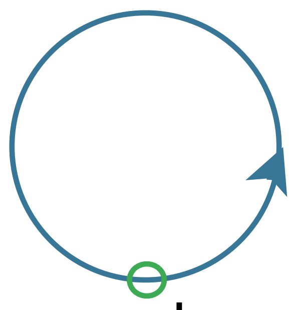
Fundamental group of circle

How many different loops are there on the circle, up to homotopy?



Fundamental group of circle

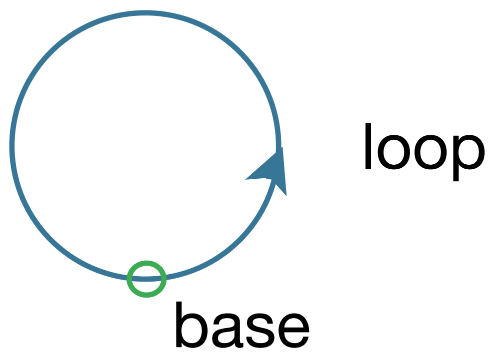
How many different loops are there on the circle, up to homotopy?



`id`

Fundamental group of circle

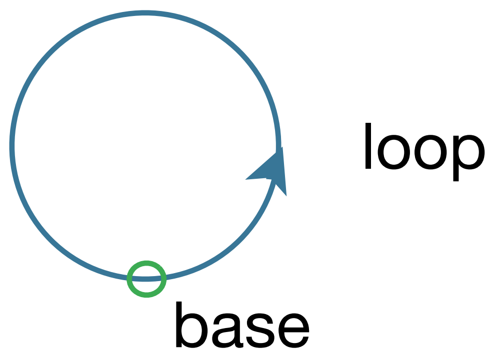
How many different loops are there on the circle, up to homotopy?




```
id
loop
```

Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

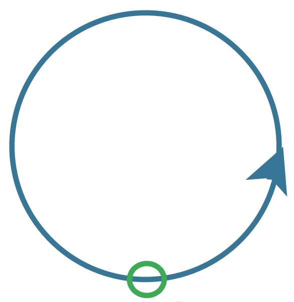



```
id
loop
```

$\mathtt{loop}^{-1}$

Fundamental group of circle

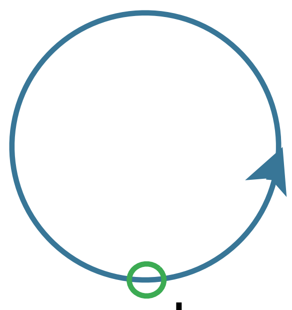
How many different loops are there on the circle, up to homotopy?




```
id
loop
loop⁻¹
loop o loop
```

Fundamental group of circle

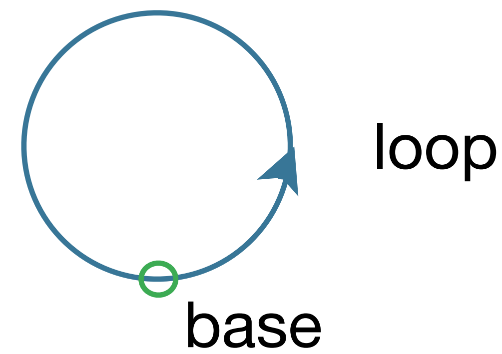
How many different loops are there on the circle, up to homotopy?




```
id
loop
loop⁻¹
loop o loop
loop⁻¹ o loop⁻¹
```

Fundamental group of circle

How many different loops are there on the circle, up to homotopy?



$\texttt{id}$

$\texttt{loop}$

$\texttt{loop}^{-1}$

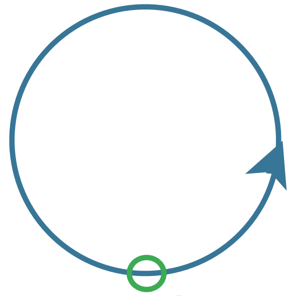
$\texttt{loop o loop}$

$\texttt{loop}^{-1}\ \texttt{o loop}^{-1}$

$\texttt{loop o loop}^{-1}$

Fundamental group of circle

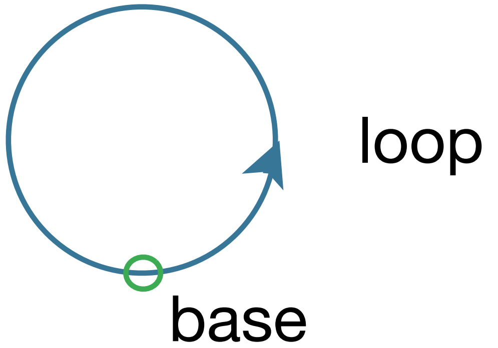
How many different loops are there on the circle, up to homotopy?




```
id
loop
loop⁻¹
loop o loop
loop⁻¹ o loop⁻¹
loop o loop⁻¹  = id
```

Fundamental group of circle

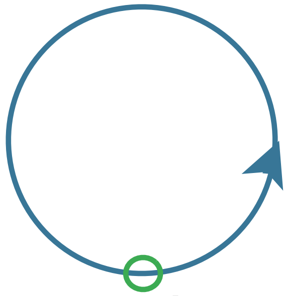
How many different loops are there on the circle, up to homotopy?




```
id                    0
loop
loop⁻¹
loop o loop
loop⁻¹ o loop⁻¹
loop o loop⁻¹  = id
```

Fundamental group of circle

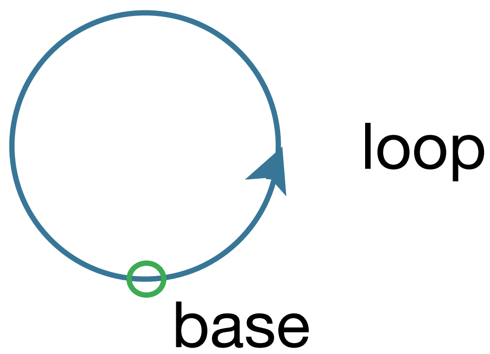
How many different loops are there on the circle, up to homotopy?



`id`	`0`
`loop`	`1`
$\texttt{loop}^{-1}$	
`loop o loop`	
$\texttt{loop}^{-1}\ \texttt{o}\ \texttt{loop}^{-1}$	
$\texttt{loop o loop}^{-1} = \texttt{id}$	

Fundamental group of circle

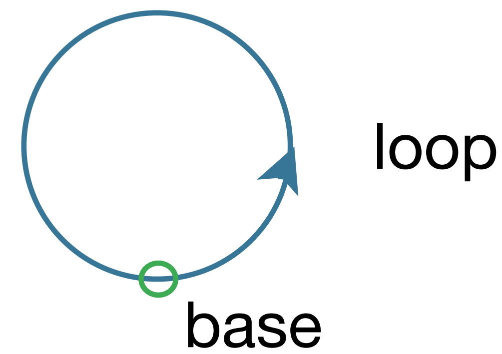
How many different loops are there on the circle, up to homotopy?



`id`	`0`
`loop`	`1`
$\texttt{loop}^{-1}$	`-1`
`loop o loop`	
$\texttt{loop}^{-1}$ `o` $\texttt{loop}^{-1}$	
`loop o` $\texttt{loop}^{-1}$ `= id`	

Fundamental group of circle

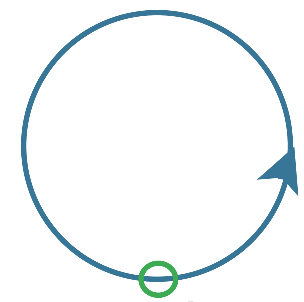
How many different loops are there on the circle, up to homotopy?



`id`	`0`
`loop`	`1`
`loop`$^{-1}$	`-1`
`loop o loop`	`2`
`loop`$^{-1}$ `o loop`$^{-1}$	
`loop o loop`$^{-1}$ `= id`	

Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

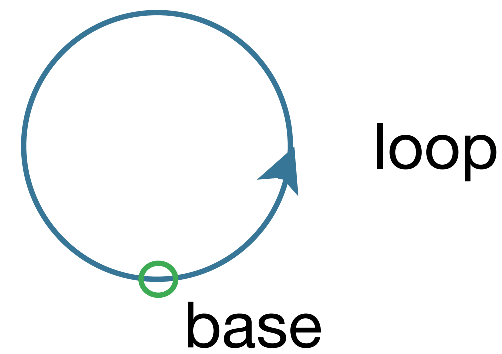


`id`	`0`
`loop`	`1`
$\texttt{loop}^{-1}$	`-1`
`loop o loop`	`2`
$\texttt{loop}^{-1} \texttt{ o } \texttt{loop}^{-1}$	`-2`

$\texttt{loop o loop}^{-1}$ `= id`

Fundamental group of circle

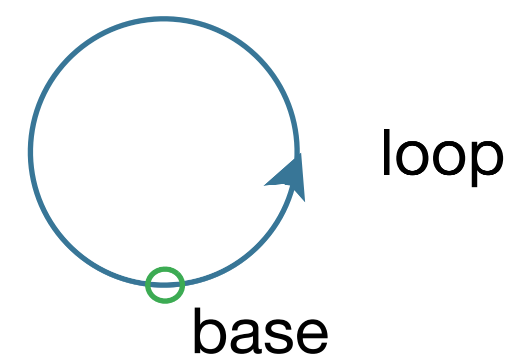
How many different loops are there on the circle, up to homotopy?



`id`	`0`
`loop`	`1`
$\texttt{loop}^{-1}$	`-1`
`loop o loop`	`2`
$\texttt{loop}^{-1}\ \texttt{o}\ \texttt{loop}^{-1}$	`-2`
$\texttt{loop o loop}^{-1}\ = \texttt{id}$	`0`

Fundamental group of circle

How many different loops are there on the circle, up to homotopy?



id	0
loop	1
loop^{-1}	-1
loop o loop	2
loop^{-1} o loop^{-1}	-2
loop o loop^{-1} = id	0

integers are "codes" for paths on the circle

Fundamental group of circle

Definition. $\Omega(S^1)$ is the **type** of loops at base
i.e. the type $(\mathtt{base} =_{S1} \mathtt{base})$

Fundamental group of circle

Definition. $\Omega(S^1)$ is the **type** of loops at base
i.e. the type `(base =`$_{S1}$ `base)`

Theorem. $\Omega(S^1)$ is equivalent to $\mathbb{Z}$,
by a map that sends o to +

Fundamental group of circle

Definition. $\Omega(S^1)$ is the **type** of loops at base
i.e. the type `(base =`$_{S1}$ `base)`

Theorem. $\Omega(S^1)$ is equivalent to $\mathbb{Z}$,
by a map that sends o to +

Corollary: Fundamental group
of the circle is isomorphic to $\mathbb{Z}$

Fundamental group of circle

Definition. $\Omega(S^1)$ is the **type** of loops at base
i.e. the type `(base =S1 base)`

Theorem. $\Omega(S^1)$ is equivalent to $\mathbb{Z}$,
by a map that sends o to +

Corollary: Fundamental group
of the circle is isomorphic to $\mathbb{Z}$

0-truncation (set of connected components) of $\Omega(S^1)$

Fundamental group of circle

Theorem. $\Omega(S^1)$ is equivalent to $\mathbb{Z}$

Proof (Shulman, L.): two mutually inverse functions

```
wind  : Ω(S¹) → ℤ

loop⁻ : ℤ → Ω(S¹)
```

Fundamental group of circle

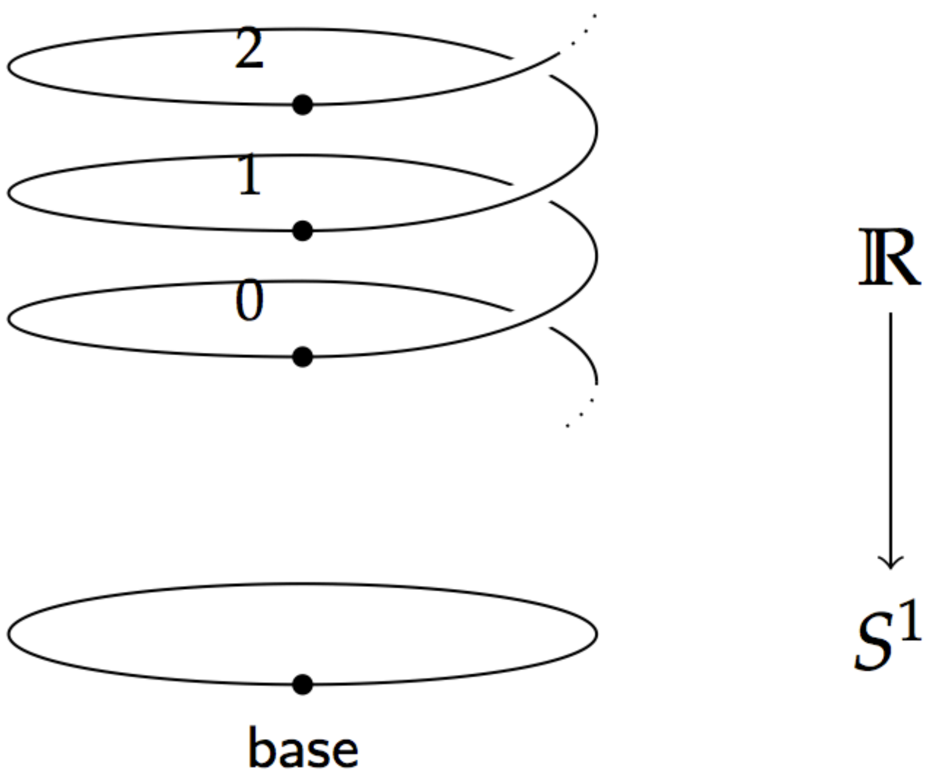
Theorem. $\Omega(S^1)$ is equivalent to $\mathbb{Z}$

Proof (Shulman, L.): two mutually inverse functions

```
wind   : Ω(S¹) → ℤ

loop⁻  : ℤ → Ω(S¹)
loop⁰   = id
loop⁺ⁿ = loop o loop o … loop         (n times)
loop⁻ⁿ = loop⁻¹ o loop⁻¹ o … loop⁻¹   (n times)
```

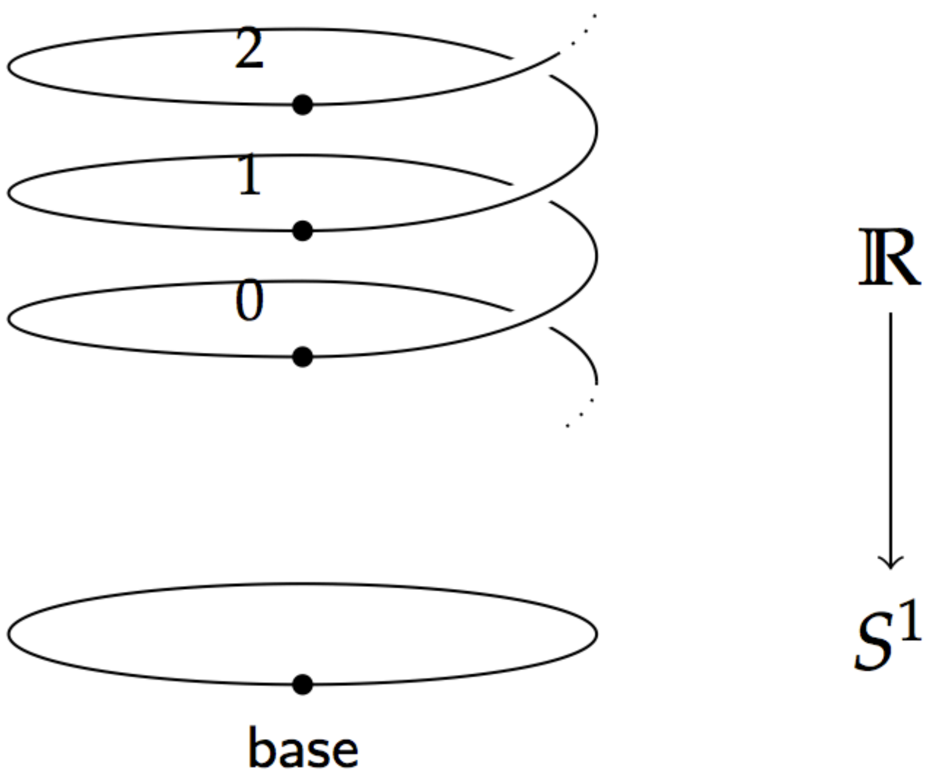
Universal Cover



`wind : Ω(S¹) → ℤ`

defined by **lifting** a loop to the cover, and giving the other endpoint of 0

Universal Cover

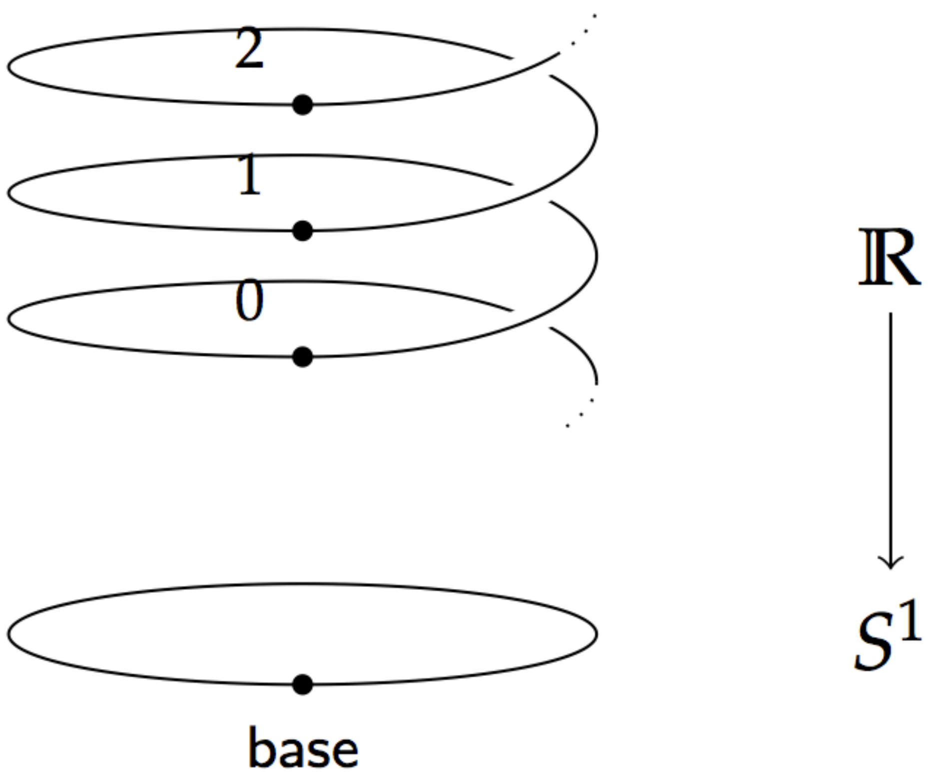


lifting is functorial

`wind` : $\Omega(S^1) \to \mathbb{Z}$

defined by **lifting** a loop to the cover, and giving the other endpoint of 0

Universal Cover



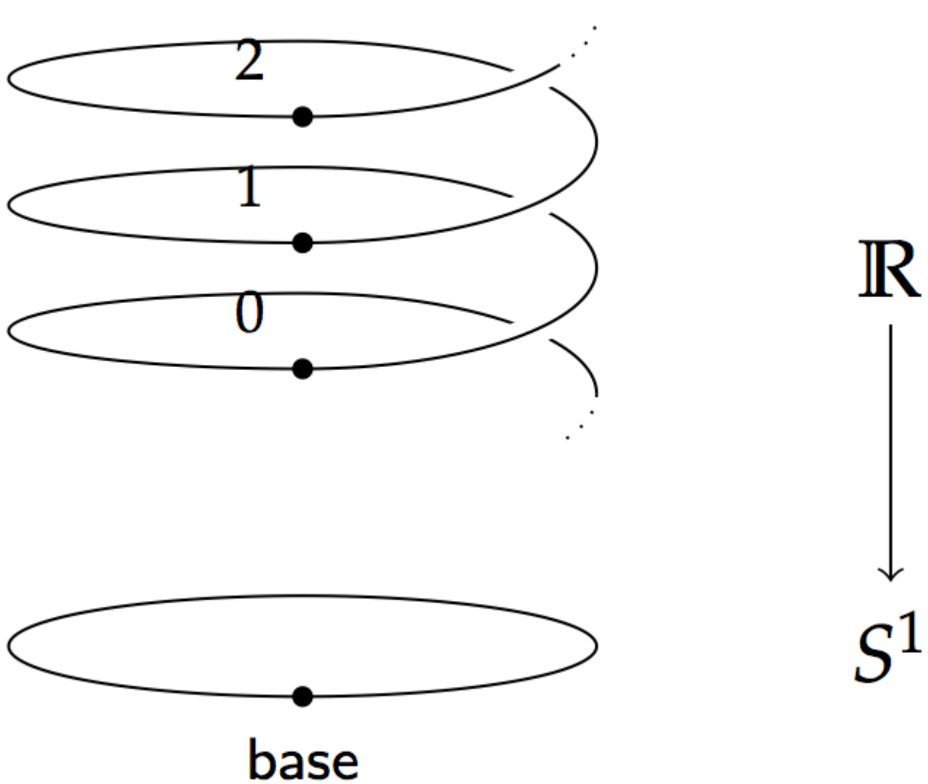
`wind` : $\Omega(S^1) \to \mathbb{Z}$

defined by **lifting** a loop to the cover, and giving the other endpoint of 0

lifting is functorial

lifting `loop` adds 1

Universal Cover



`wind : Ω(S¹) → ℤ`

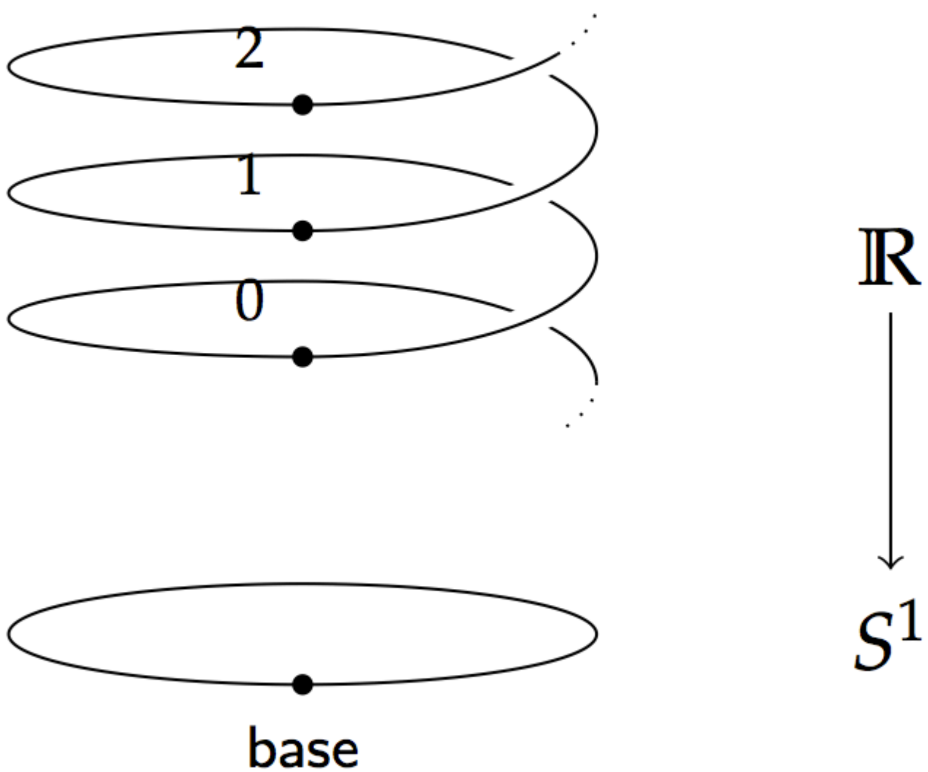
defined by **lifting** a loop to the cover, and giving the other endpoint of 0

lifting is functorial

lifting `loop` adds 1

lifting `loop`$^{-1}$ subtracts 1

Universal Cover



`wind : Ω(S¹) → ℤ`

defined by **lifting** a loop to the cover, and giving the other endpoint of 0

lifting is functorial

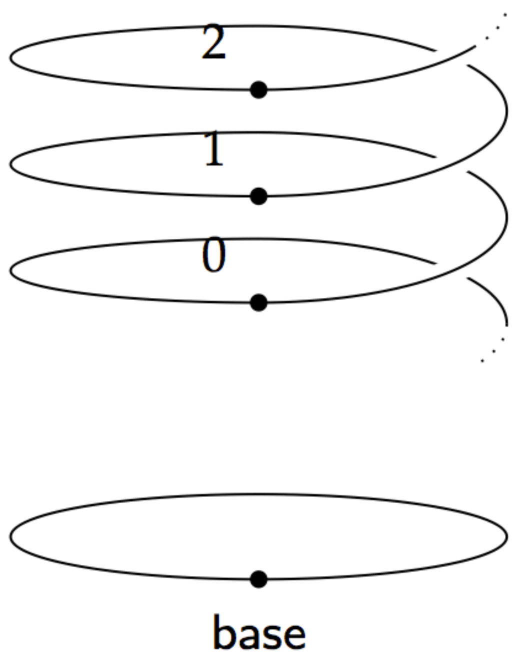
lifting `loop` adds 1

lifting `loop`$^{-1}$ subtracts 1

Example:

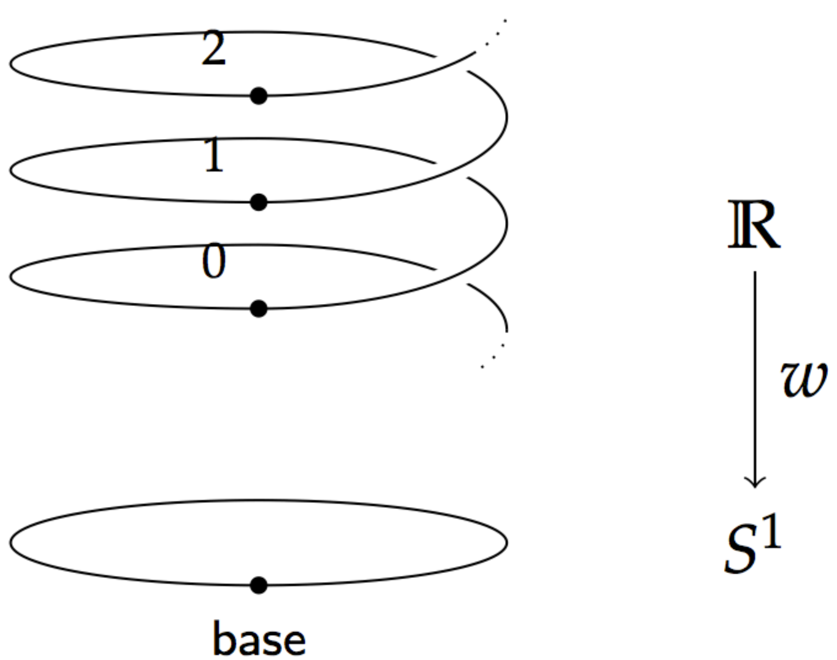
```
  wind(loop o loop⁻¹)
= 0 + 1 - 1
= 0
```

Universal Cover



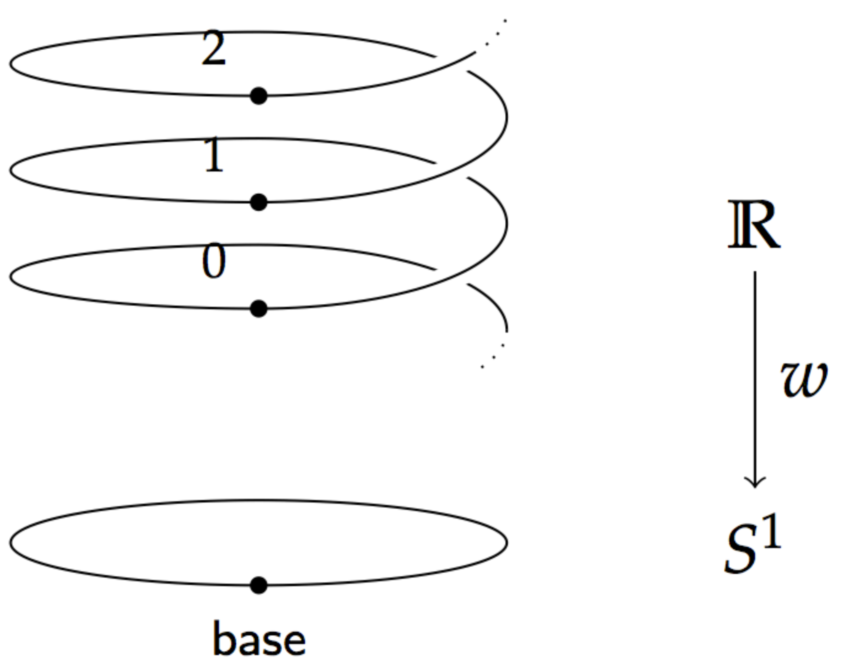
$$\begin{tikzcd} \mathbb{R} \arrow[d, "w"] \\ S^1 \end{tikzcd}$$

Universal Cover




```
Cover : S¹ → Type
Cover(base) := ℤ
Cover₁(loop) :=
  ua(successor) : ℤ = ℤ
```

Universal Cover



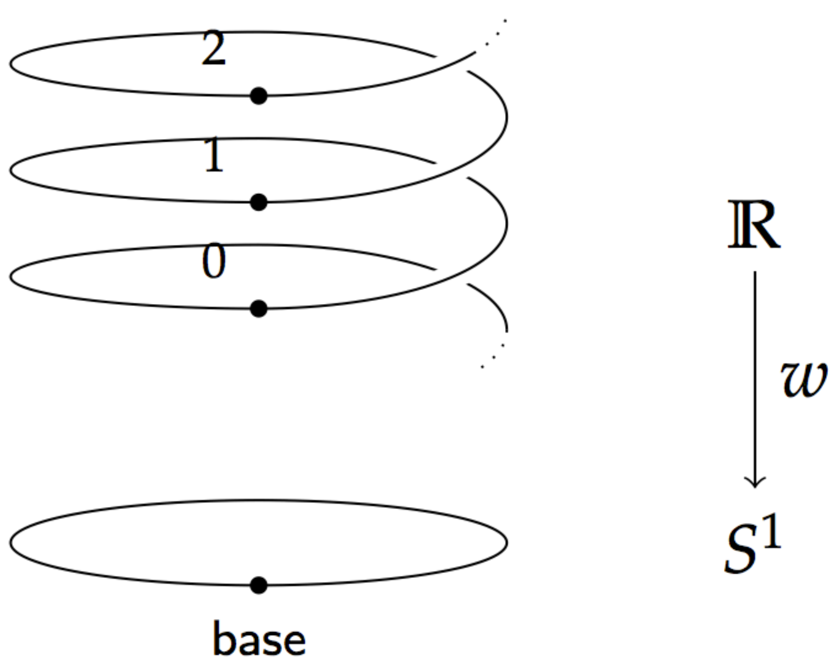
defined by circle recursion

$\mathtt{Cover} : S^1 \to \mathtt{Type}$

$\mathtt{Cover(base)} := \mathbb{Z}$

$\mathtt{Cover_1(loop)} :=$
$\quad \mathtt{ua(successor)} : \mathbb{Z} = \mathbb{Z}$

Universal Cover



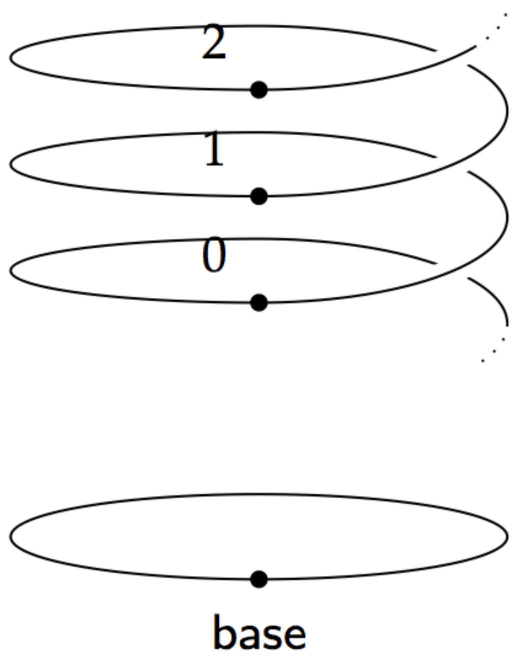
defined by circle recursion

```
Cover : S¹ → Type
Cover(base) := ℤ
Cover₁(loop) :=
  ua(successor) : ℤ = ℤ
```

interpret loop as "add 1" bijection

Universal Cover

defined by circle recursion



$$\begin{array}{c} \mathbb{R} \\ \big\downarrow w \\ S^1 \end{array}$$

$$\mathtt{Cover} : S^1 \to \mathtt{Type}$$

$$\mathtt{Cover(base)} := \mathbb{Z}$$

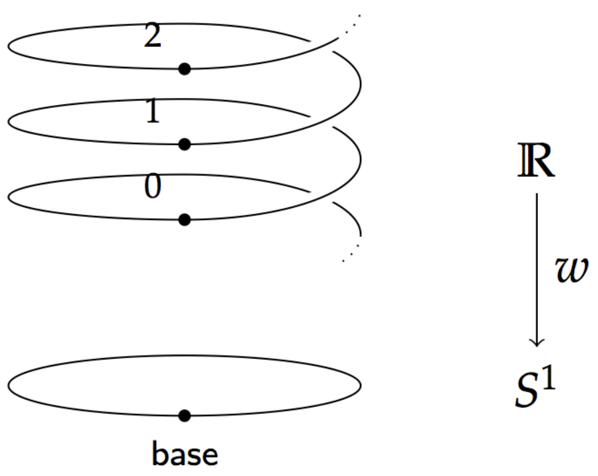
$$\mathtt{Cover}_1\mathtt{(loop)} :=$$

$$\quad \mathtt{ua(successor)} : \mathbb{Z} = \mathbb{Z}$$

univalence

interpret loop as "add 1" bijection

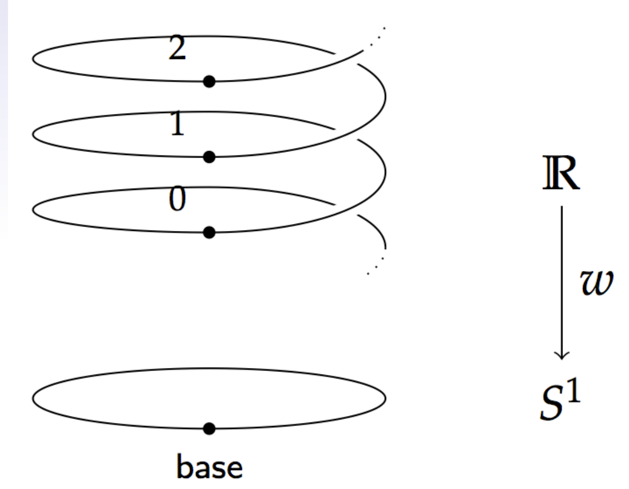
Winding number



$$\mathsf{wind} : \Omega(S^1) \to \mathbb{Z}$$
$$\mathsf{wind}(p) = \mathsf{transport}_{\mathsf{Cover}}(p, 0)$$

lift p to cover, starting at 0

Winding number



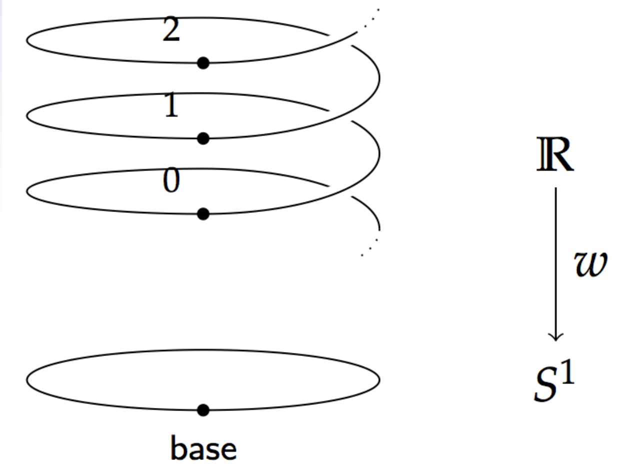
wind : Ω(S¹) → ℤ

wind(p) = transport$_{Cover}$(p,0)

lift p to cover, starting at 0

wind(loop⁻¹ o loop)

Winding number



$\texttt{wind} : \Omega(S^1) \to \mathbb{Z}$

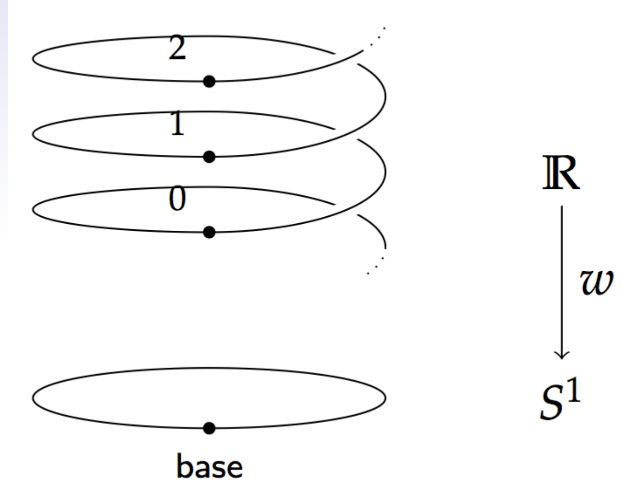
$\texttt{wind(p)} = \texttt{transport}_{\texttt{Cover}}\texttt{(p,0)}$

lift p to cover, starting at 0

$\texttt{wind(loop}^{-1}\texttt{ o loop)}$

$= \texttt{transport}_{\texttt{Cover}}\texttt{(loop}^{-1}\texttt{ o loop, 0)}$

Winding number

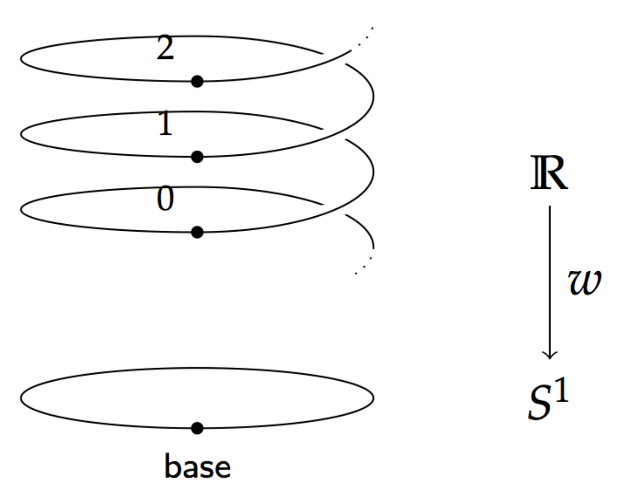



```
wind : Ω(S¹) → ℤ
wind(p) = transport_Cover(p,0)
```

lift p to cover, starting at 0

```
  wind(loop⁻¹ o loop)
= transport_Cover(loop⁻¹ o loop, 0)
= transport_Cover(loop⁻¹, transport_Cover(loop,0))
```

Winding number

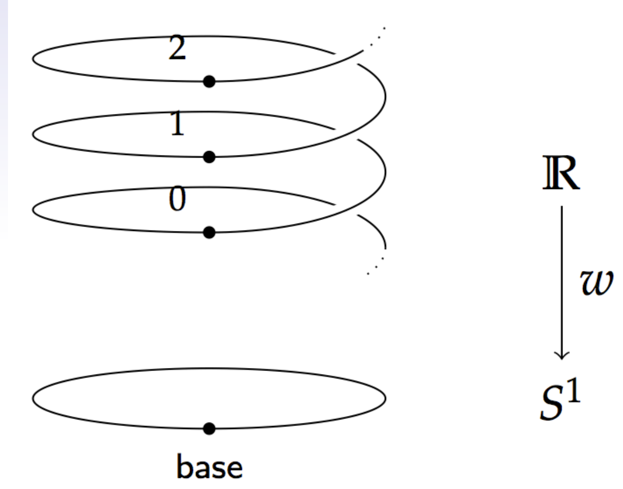


$$\text{wind} : \Omega(S^1) \to \mathbb{Z}$$
$$\text{wind}(p) = \text{transport}_{\text{Cover}}(p, 0)$$

lift p to cover, starting at 0

$$\begin{aligned}
&\text{wind}(\text{loop}^{-1} \circ \text{loop}) \\
&= \text{transport}_{\text{Cover}}(\text{loop}^{-1} \circ \text{loop}, 0) \\
&= \text{transport}_{\text{Cover}}(\text{loop}^{-1}, \text{transport}_{\text{Cover}}(\text{loop}, 0)) \\
&= \text{transport}_{\text{Cover}}(\text{loop}^{-1}, 1)
\end{aligned}$$

Winding number



$$\text{wind} : \Omega(S^1) \to \mathbb{Z}$$

$$\text{wind}(p) = \text{transport}_{\text{Cover}}(p, 0)$$

lift p to cover, starting at 0

$$\begin{aligned}
&\text{wind}(\text{loop}^{-1} \circ \text{loop}) \\
&= \text{transport}_{\text{Cover}}(\text{loop}^{-1} \circ \text{loop}, 0) \\
&= \text{transport}_{\text{Cover}}(\text{loop}^{-1}, \text{transport}_{\text{Cover}}(\text{loop}, 0)) \\
&= \text{transport}_{\text{Cover}}(\text{loop}^{-1}, 1) \\
&= 0
\end{aligned}$$

Fundamental group of the circle

The HoTT book

Computer-checked

$\pi_n(S^n)$ in HoTT

k^{th} homotopy group

n-dimensional sphere

	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	π_{12}	π_{13}	π_{14}	π_{15}
S^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^1	$\mathbf{Z}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^2	0	$\mathbf{Z}$	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{12}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_3$	$\mathbf{Z}_{15}$	$\mathbf{Z}_2$	$\mathbf{Z}_2^2$	$\mathbf{Z}_{12}\times\mathbf{Z}_2$	$\mathbf{Z}_{84}\times\mathbf{Z}_2^2$	$\mathbf{Z}_2^2$
S^3	0	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{12}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_3$	$\mathbf{Z}_{15}$	$\mathbf{Z}_2$	$\mathbf{Z}_2^2$	$\mathbf{Z}_{12}\times\mathbf{Z}_2$	$\mathbf{Z}_{84}\times\mathbf{Z}_2^2$	$\mathbf{Z}_2^2$
S^4	0	0	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}\times\mathbf{Z}_{12}$	$\mathbf{Z}_2^2$	$\mathbf{Z}_2^2$	$\mathbf{Z}_{24}\times\mathbf{Z}_3$	$\mathbf{Z}_{15}$	$\mathbf{Z}_2$	$\mathbf{Z}_2^3$	$\mathbf{Z}_{120}\times\mathbf{Z}_{12}\times\mathbf{Z}_2$	$\mathbf{Z}_{84}\times\mathbf{Z}_2^5$
S^5	0	0	0	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{24}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{30}$	$\mathbf{Z}_2$	$\mathbf{Z}_2^3$	$\mathbf{Z}_{72}\times\mathbf{Z}_2$
S^6	0	0	0	0	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{24}$	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_{60}$	$\mathbf{Z}_{24}\times\mathbf{Z}_2$	$\mathbf{Z}_2^3$
S^7	0	0	0	0	0	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{24}$	0	0	$\mathbf{Z}_2$	$\mathbf{Z}_{120}$	$\mathbf{Z}_2^3$
S^8	0	0	0	0	0	0	0	$\mathbf{Z}$	$\mathbf{Z}_2$	$\mathbf{Z}_2$	$\mathbf{Z}_{24}$	0	0	$\mathbf{Z}_2$	$\mathbf{Z}\times\mathbf{Z}_{120}$

[image from wikipedia]

$\pi_n(S^n)$ in HoTT

k^{th} homotopy group

n-dimensional sphere

	π_1	π_2	π_3	π_4	π_5	π_6	π_7	π_8	π_9	π_{10}	π_{11}	π_{12}	π_{13}	π_{14}	π_{15}
S^0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^1	$\mathbb{Z}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S^2	0	$\mathbb{Z}$	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_3$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{12}\times\mathbb{Z}_2$	$\mathbb{Z}_{84}\times\mathbb{Z}_2^2$	$\mathbb{Z}_2^2$
S^3	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_3$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{12}\times\mathbb{Z}_2$	$\mathbb{Z}_{84}\times\mathbb{Z}_2^2$	$\mathbb{Z}_2^2$
S^4	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}\times\mathbb{Z}_{12}$	$\mathbb{Z}_2^2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{24}\times\mathbb{Z}_3$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^3$	$\mathbb{Z}_{120}\times\mathbb{Z}_{12}\times\mathbb{Z}_2$	$\mathbb{Z}_{84}\times\mathbb{Z}_2^5$
S^5	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{30}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^3$	$\mathbb{Z}_{72}\times\mathbb{Z}_2$
S^6	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_{60}$	$\mathbb{Z}_{24}\times\mathbb{Z}_2$	$\mathbb{Z}_2^3$
S^7	0	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_{120}$	$\mathbb{Z}_2^3$
S^8	0	0	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	0	$\mathbb{Z}_2$	$\mathbb{Z}\times\mathbb{Z}_{120}$

[image from wikipedia]

$\pi_n(S^n) = \mathbb{Z}$ for $n \geq 1$

Proof: Induction on n

- Base case: $\pi_1(S^1) = \mathbb{Z}$
- Inductive step: $\pi_{n+1}(S^{n+1}) = \pi_n(S^n)$

$\pi_n(S^n) = \mathbb{Z}$ for $n \geq 1$

Proof: Induction on n

- Base case: $\pi_1(S^1) = \mathbb{Z}$
- Inductive step: $\pi_{n+1}(S^{n+1}) = \pi_n(S^n)$

  Key lemma: $|S^n|_n = |\Omega(S^{n+1})|_n$

$\pi_n(S^n) = \mathbb{Z}$ for $n \geq 1$

Proof: Induction on n

- Base case: $\pi_1(S^1) = \mathbb{Z}$
- Inductive step: $\pi_{n+1}(S^{n+1}) = \pi_n(S^n)$

Key lemma: $|S^n|_n = |\Omega(S^{n+1})|_n$

n-truncation:
best approximation of a type such that all (n+1)-paths are equal

$\pi_n(S^n) = \mathbb{Z}$ for $n \geq 1$

Proof: Induction on n

- Base case: $\pi_1(S^1) = \mathbb{Z}$
- Inductive step: $\pi_{n+1}(S^{n+1}) = \pi_n(S^n)$

Key lemma: $|S^n|_n = |\Omega(S^{n+1})|_n$

n-truncation:
best approximation of a type such that all (n+1)-paths are equal

higher inductive type generated by
$\texttt{base}_n : S^n$
$\texttt{loop}_n : \Omega^n(S^n)$

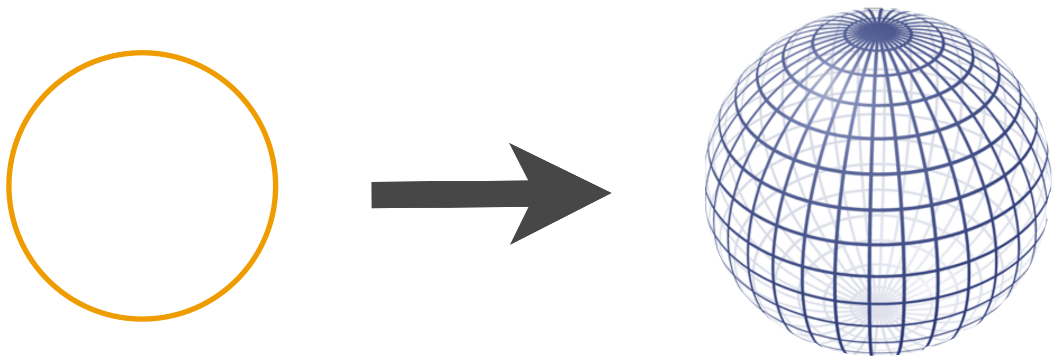
$|S^n|_n = |\Omega(S^{n+1})|_n$

n-truncation of S^n is the type of “codes” for loops on S^{n+1}

$|S^n|_n = |\Omega(S^{n+1})|_n$

n-truncation of S^n is the type of "codes" for loops on S^{n+1}

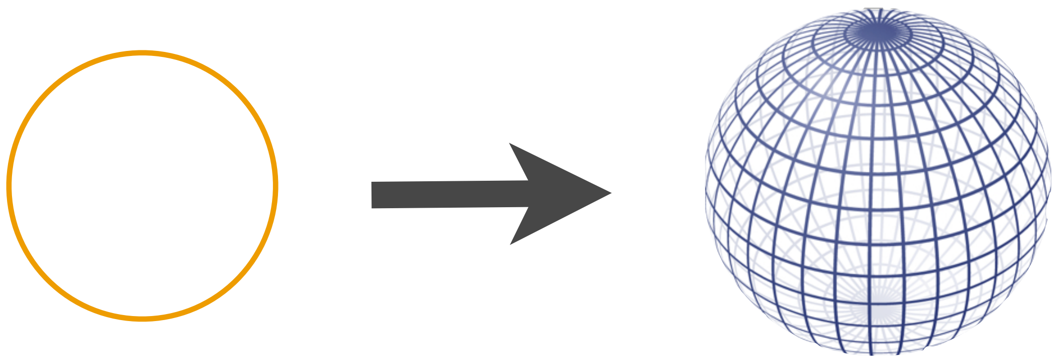
- Decode: promote n-dimensional loop on S^n to n+1-dimensional loop on S^{n+1}



$|S^n|_n = |\Omega(S^{n+1})|_n$

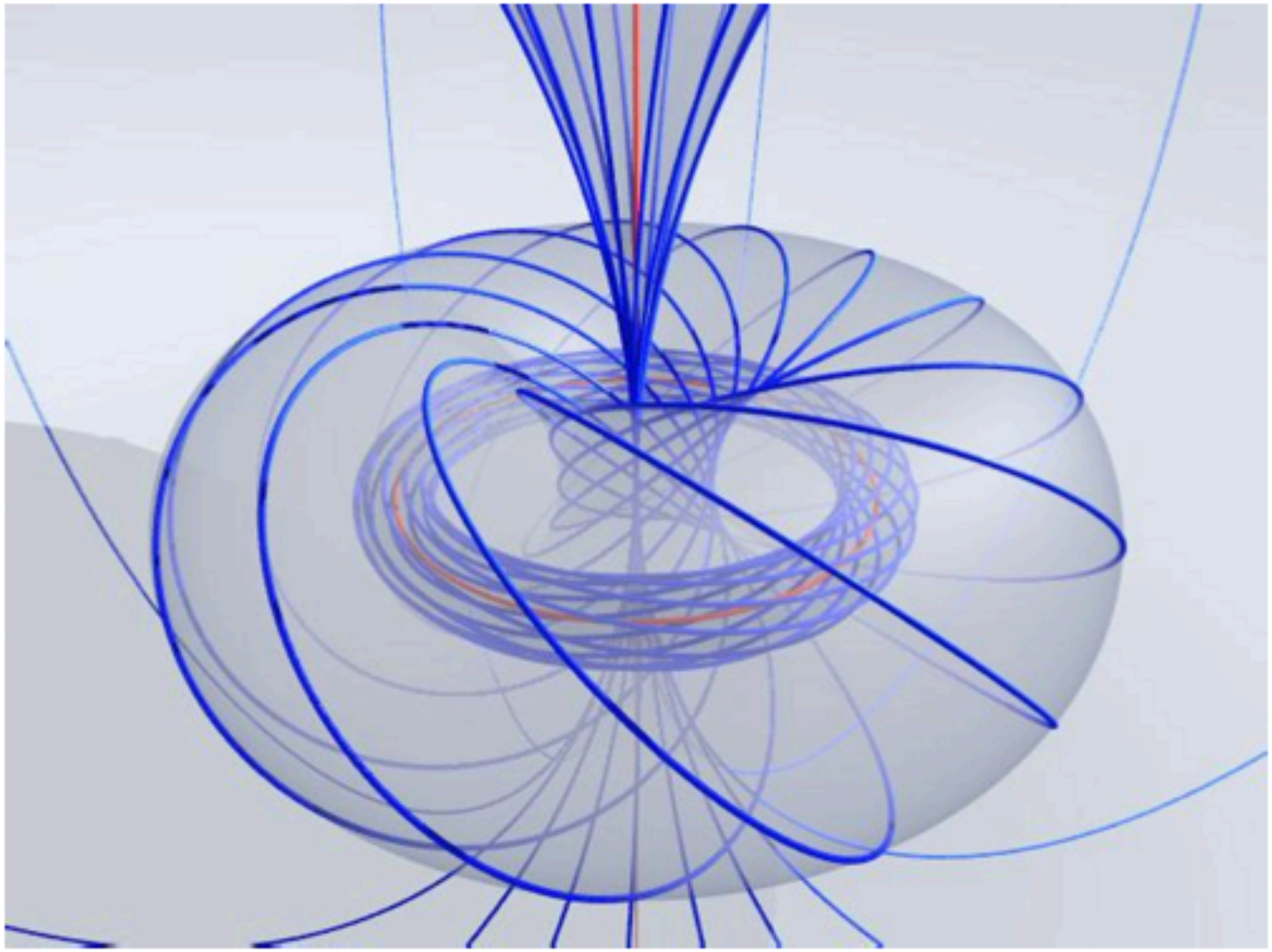
n-truncation of S^n is the type of "codes" for loops on S^{n+1}

- Decode: promote n-dimensional loop on S^n to n+1-dimensional loop on S^{n+1}



- Encode: define fibration `Code(x:`S^{n+1}`)` with
  `Code(`$\mathtt{base}_{n+1}$`)` $:= |S^n|_n$
  `Code(`$\mathtt{loop}_{n+1}$`)` := equivalence $|S^n|_n \simeq |S^n|_n$
  "rotating by $\mathtt{loop}_n$"

$\pi_2(S^2)$: Hopf fibration



Synthetic homotopy theory

- Gap between informal and formal proofs is small
- Proofs are constructive*: can run them
- Results apply in a variety of settings,
  from simplicial sets (hence topological spaces)
  to Quillen model categories and ∞-topoi*
- New type-theoretic proofs/methods

*work in progress

Outline

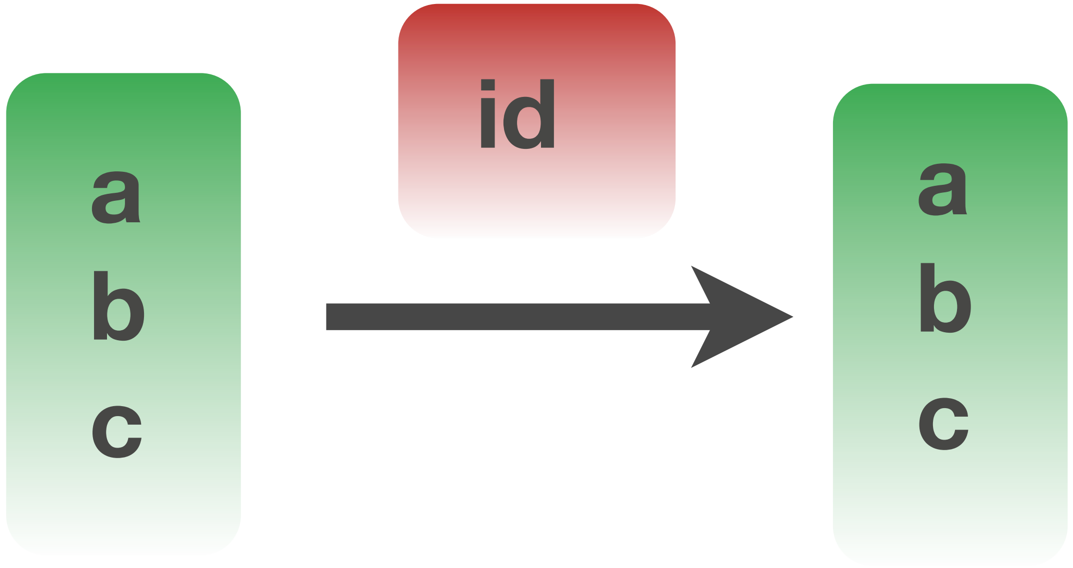
1. Certified homotopy theory
2. **Certified software**

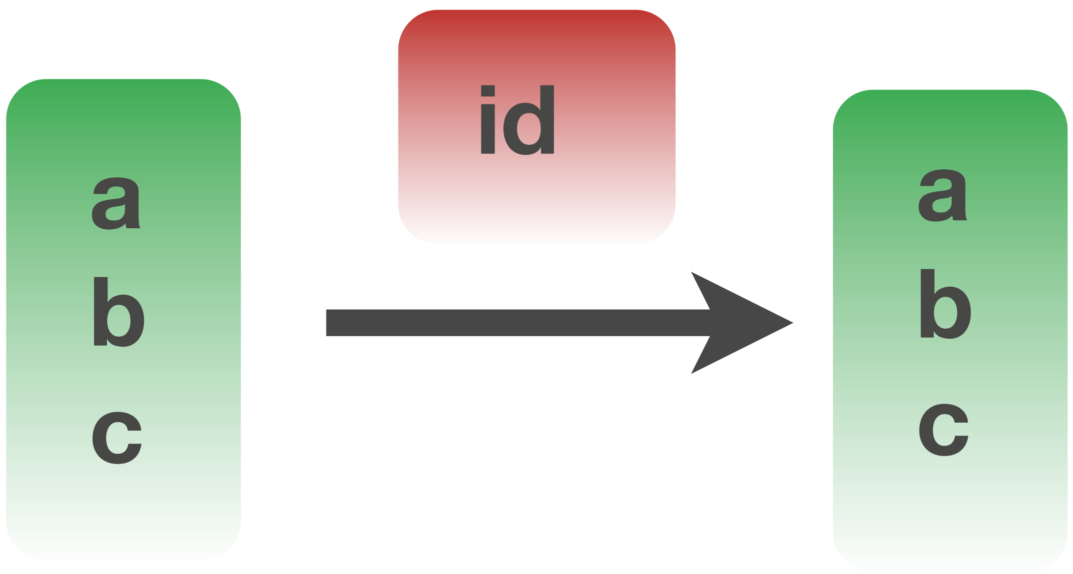
Patches

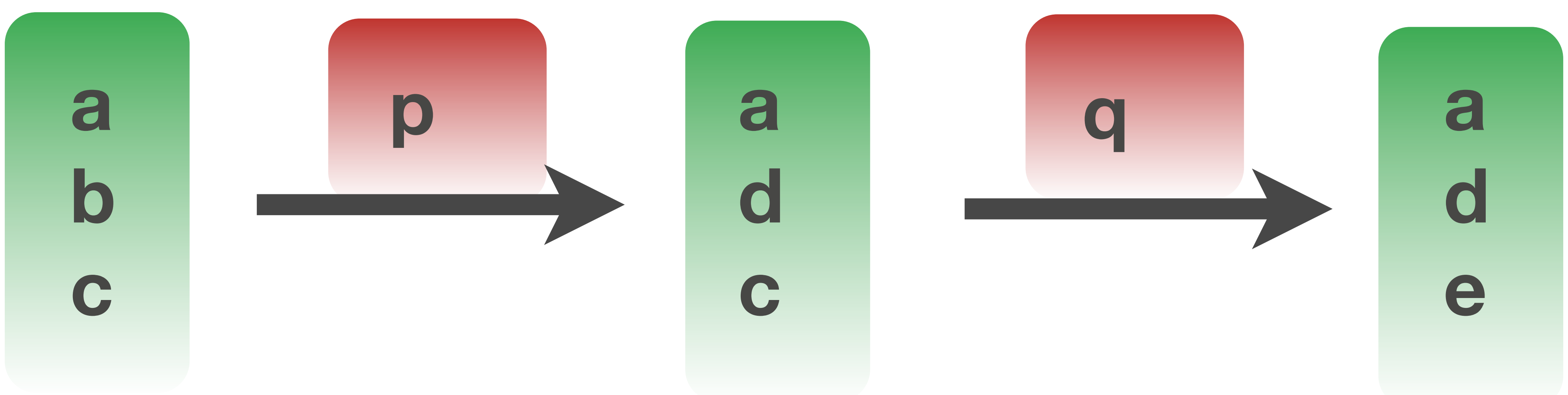
Patch

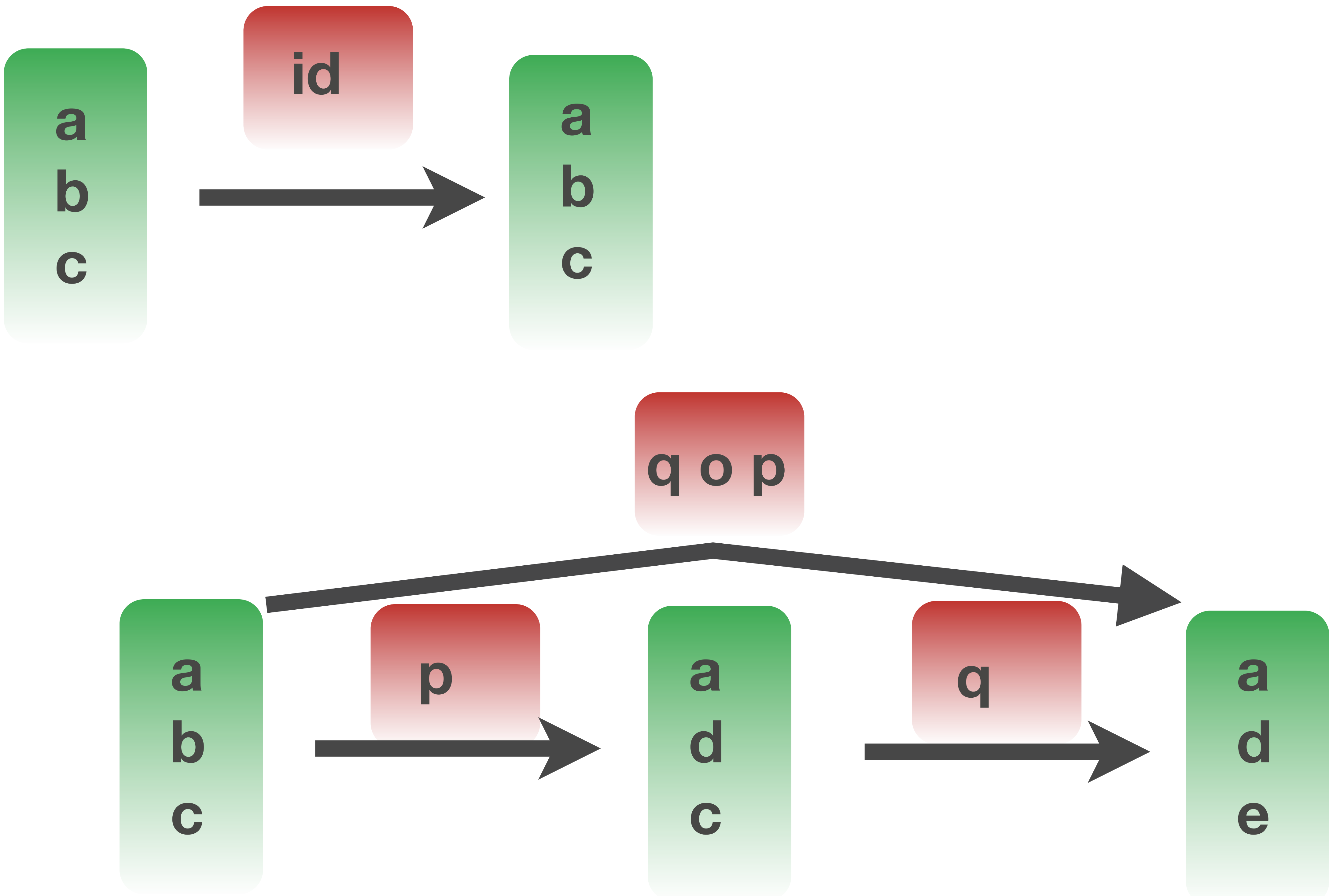
```
diff  a  a   =  2c2
      b  d      < b
      c  c      ---
                > d
```

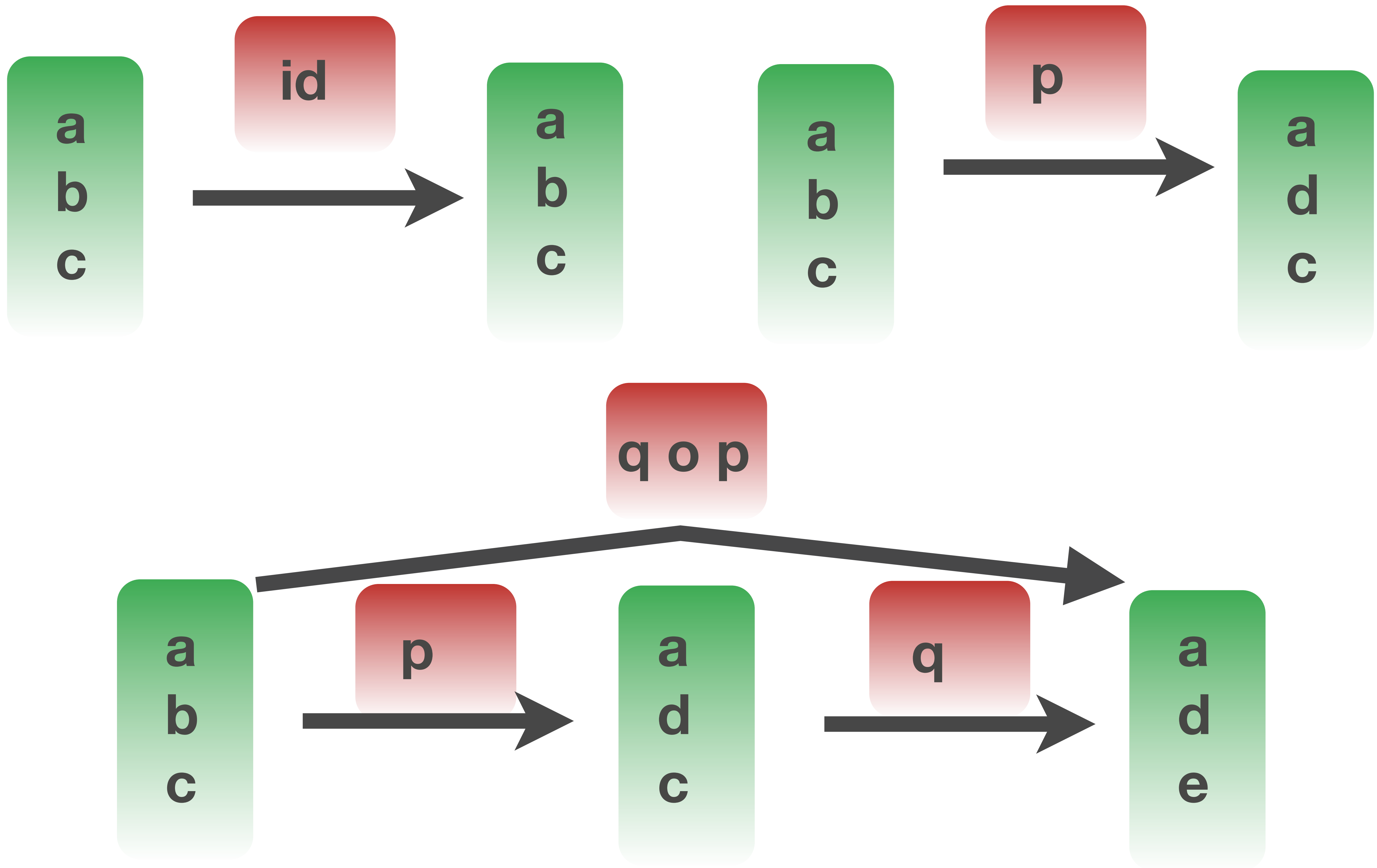
- Version control
- Collaborative editing

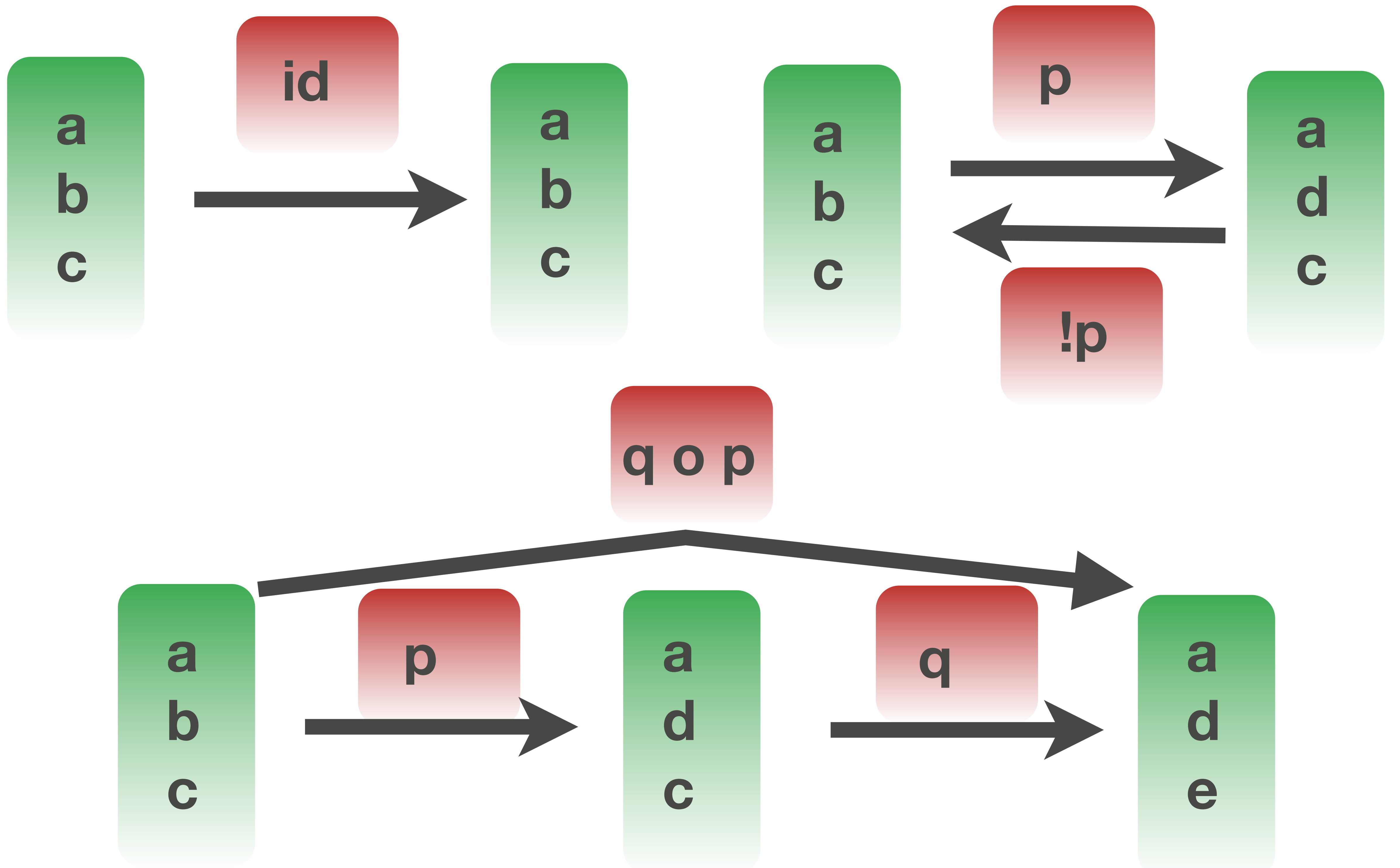

a
b
c
id
a
b
c


a
b
c
id
a
b
c

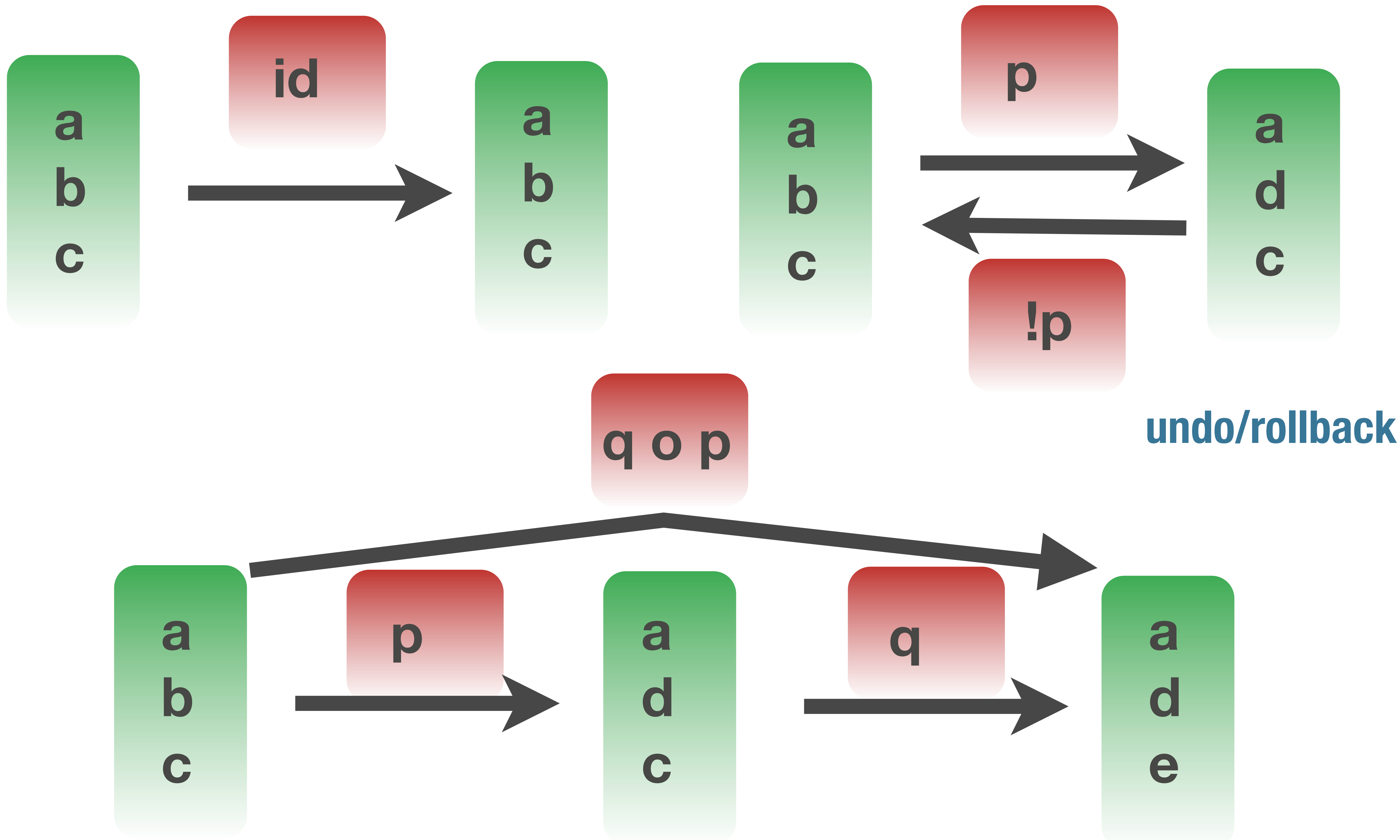

a
b
c
p
a
d
c
q
a
d
e


id
a
b
c
a
b
c
q o p
p
q
a
b
c
a
d
c
a
d
e


a
b
c
id
a
b
c
a
b
c
p
a
d
c
q o p
a
b
c
p
a
d
c
q
a
d
e

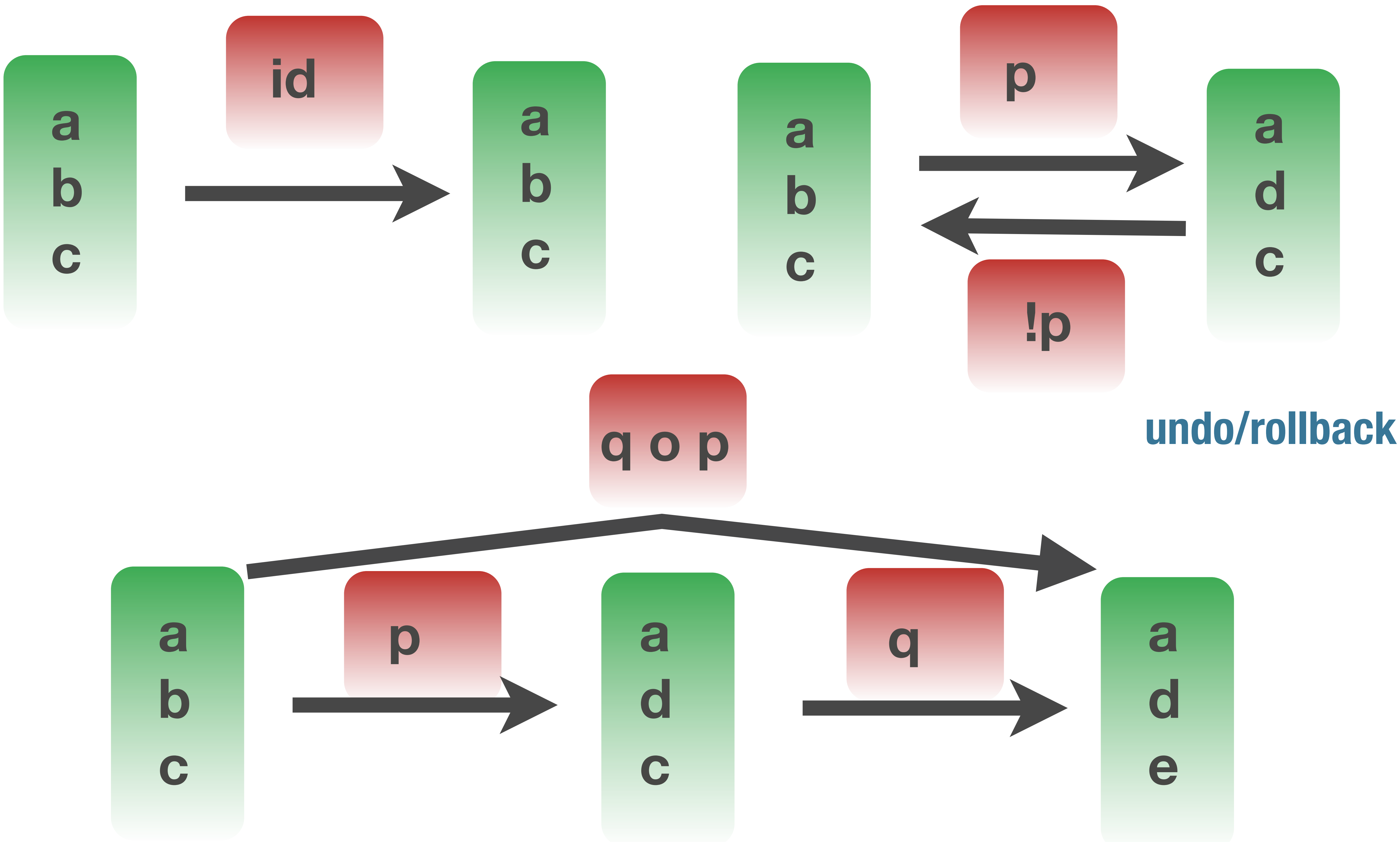


a b c
id
a b c
a b c
p
!p
a d c
q o p
a b c
p
a d c
q
a d e

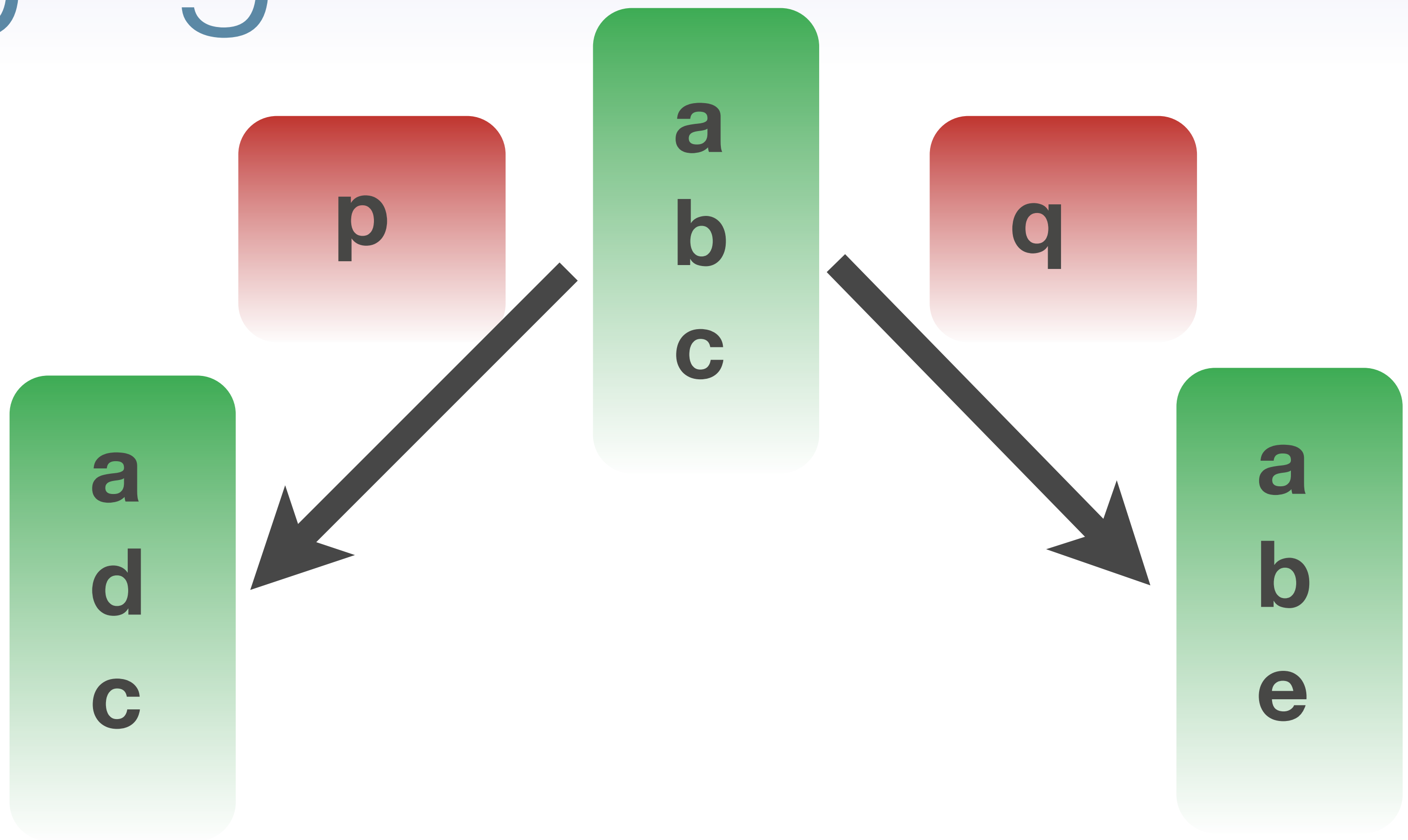


a
b
c
id
a
b
c
a
b
c
p
!p
a
d
c
undo/rollback
q o p
a
b
c
p
a
d
c
q
a
d
e

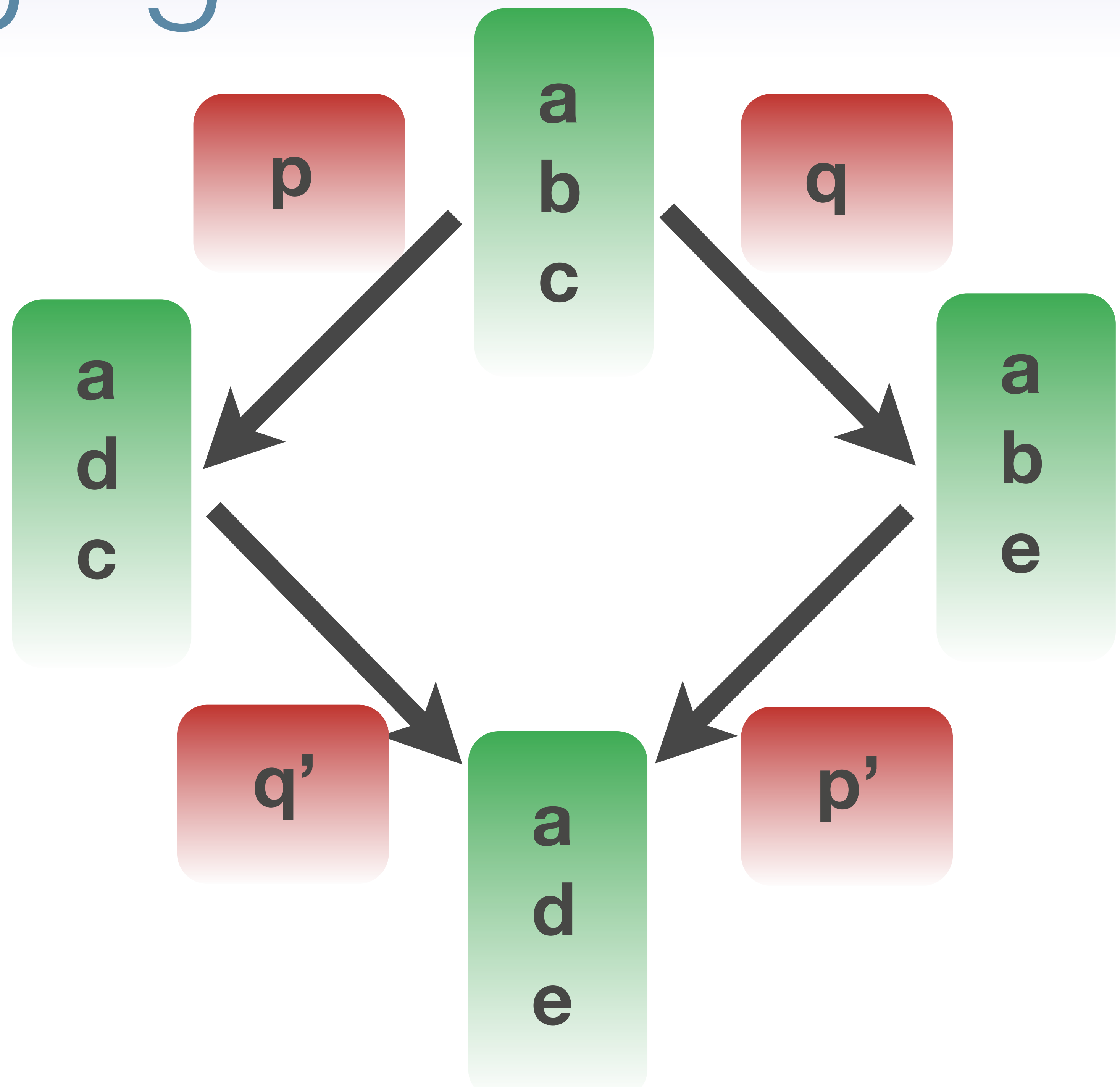
Patches are paths



Merging



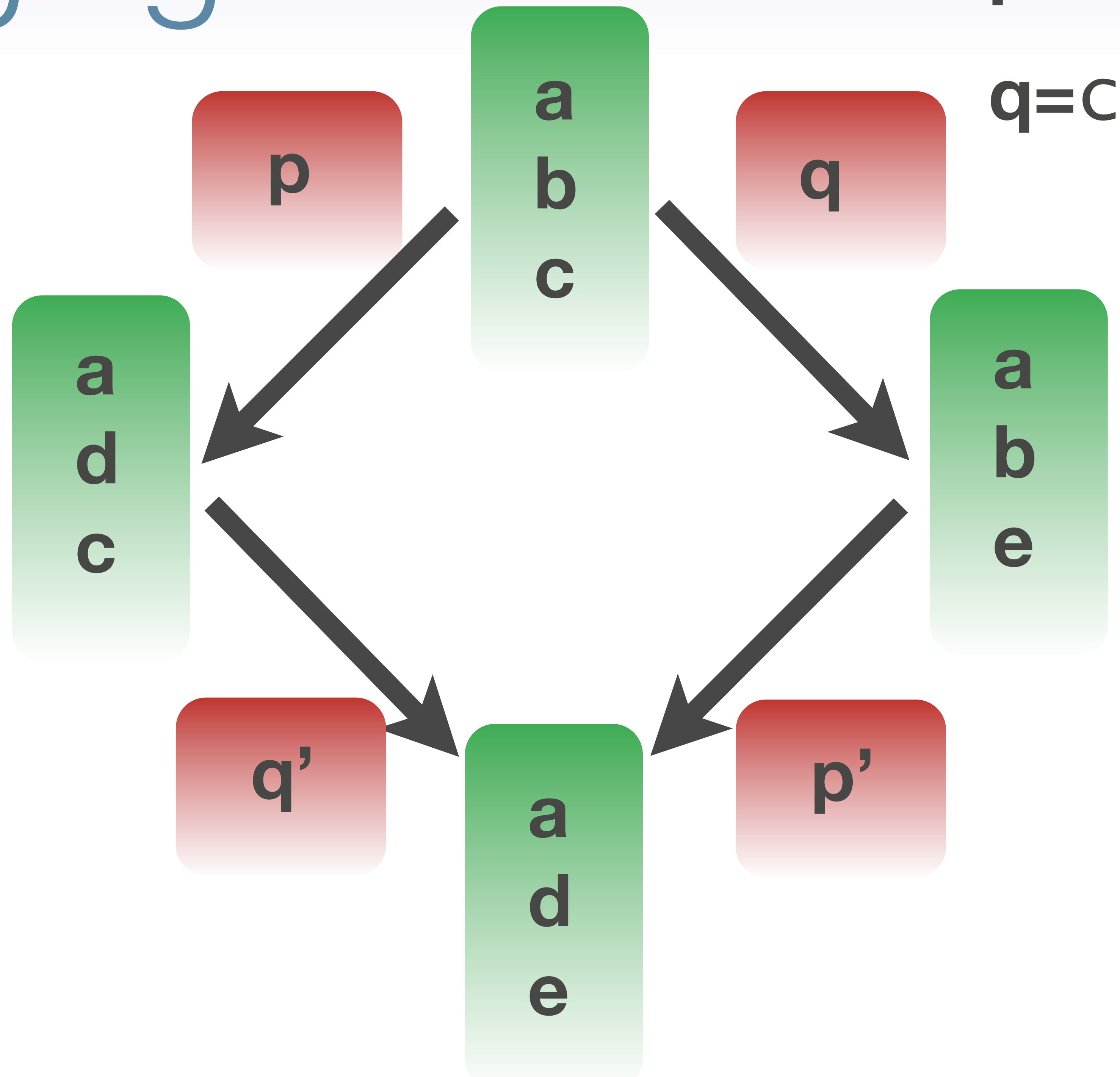
Merging



Merging

p=b↔d at 1

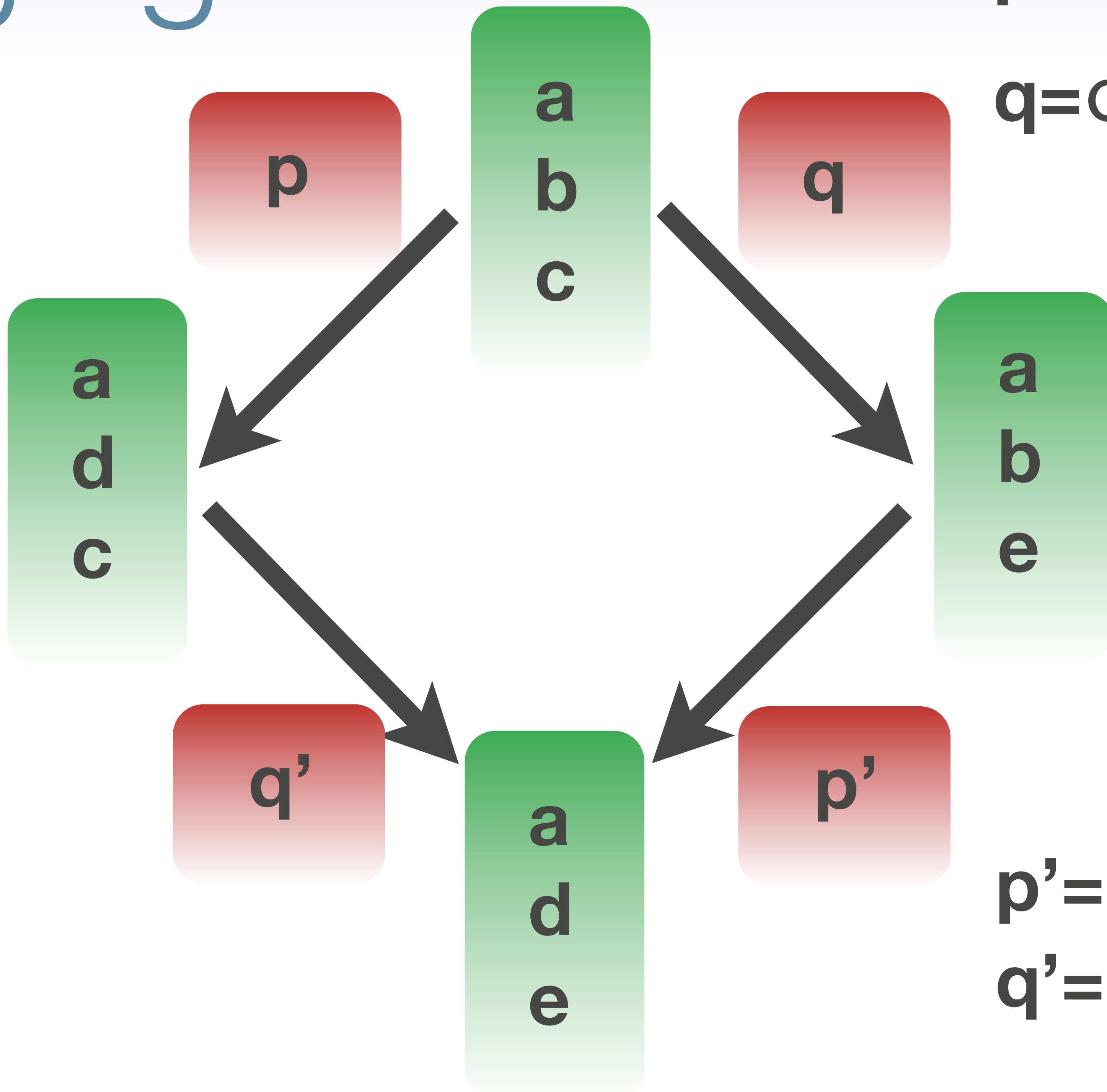
q=c↔e at 2



Merging

p=b↔d at 1

q=c↔e at 2



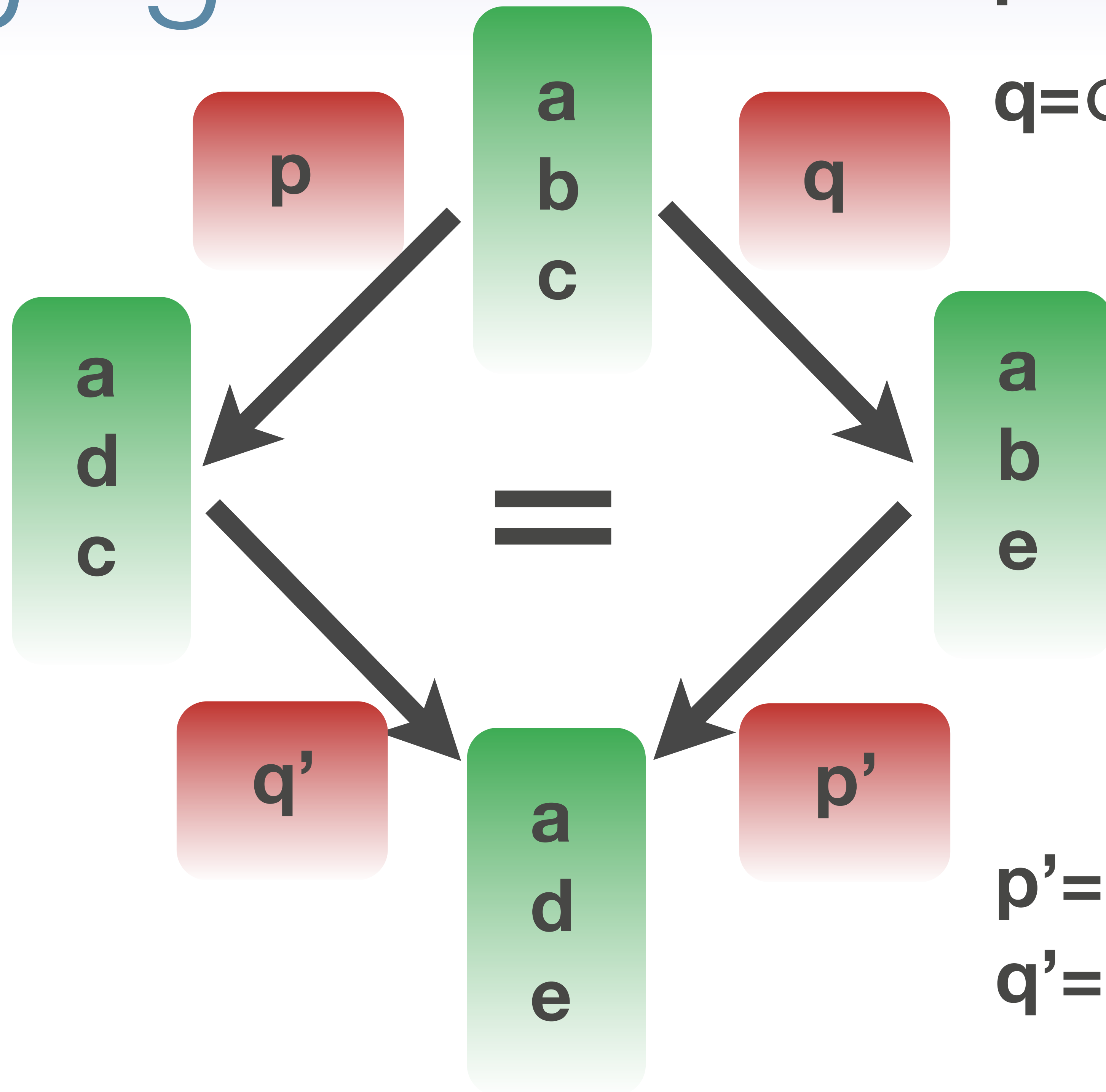
p’=p

q’=q

Merging

p=b↔d at 1

q=c↔e at 2

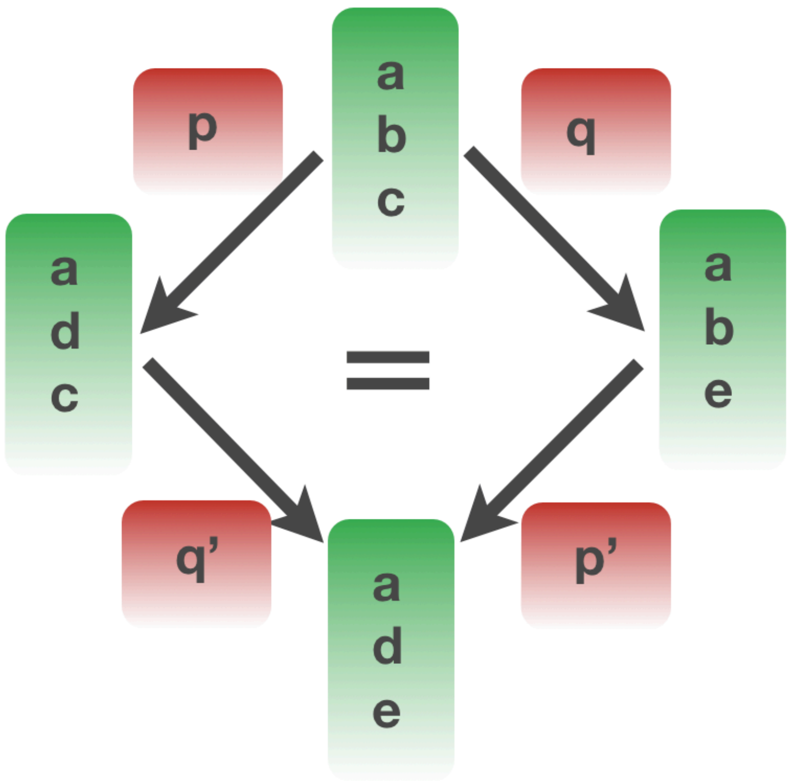


p’=p

q’=q

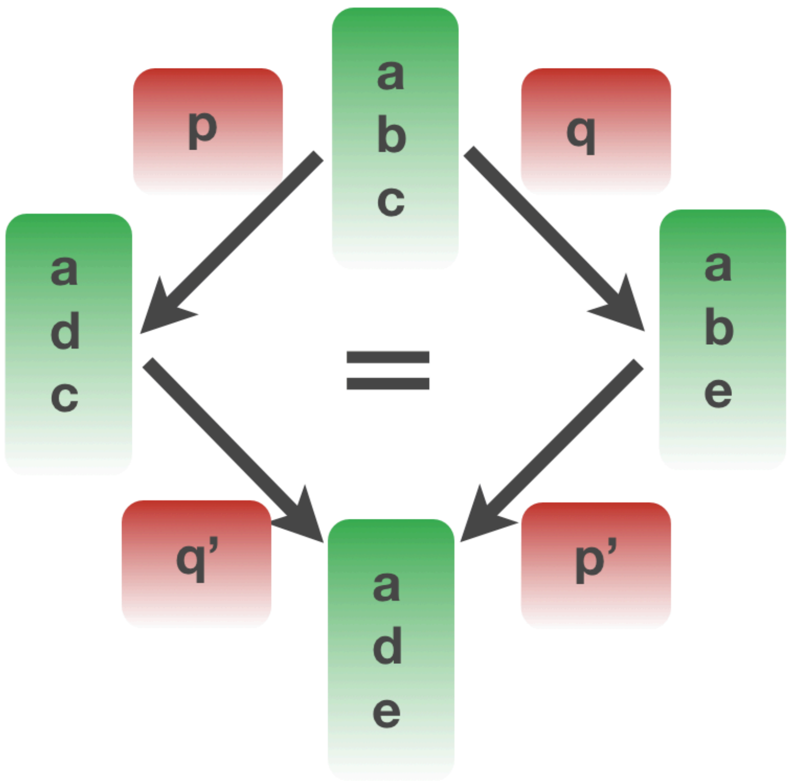
Merging

```
merge : (p q : Patch)
      → Σq',p':Patch.
        Maybe(q' o p =
              p' o q)
```



Merging

```
merge : (p q : Patch)
      → Σq’,p’:Patch.
        Maybe(q’ o p =
                  p’ o q)
```



Equational theory of patches
= paths between paths

Basic Patches

f	i	b	r	a	t	i	o	n

$a \leftrightarrow b$ @ 2

f	i	b

f	i	a

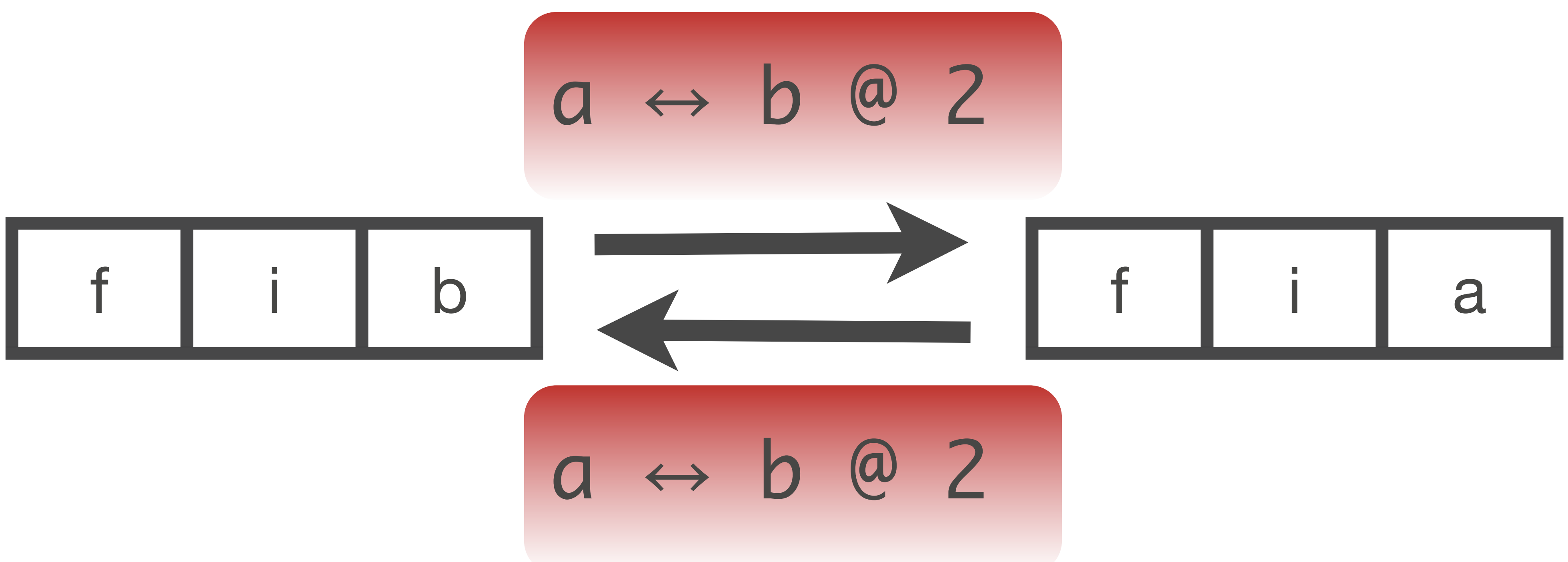
$a \leftrightarrow b$ @ 2

Basic Patches

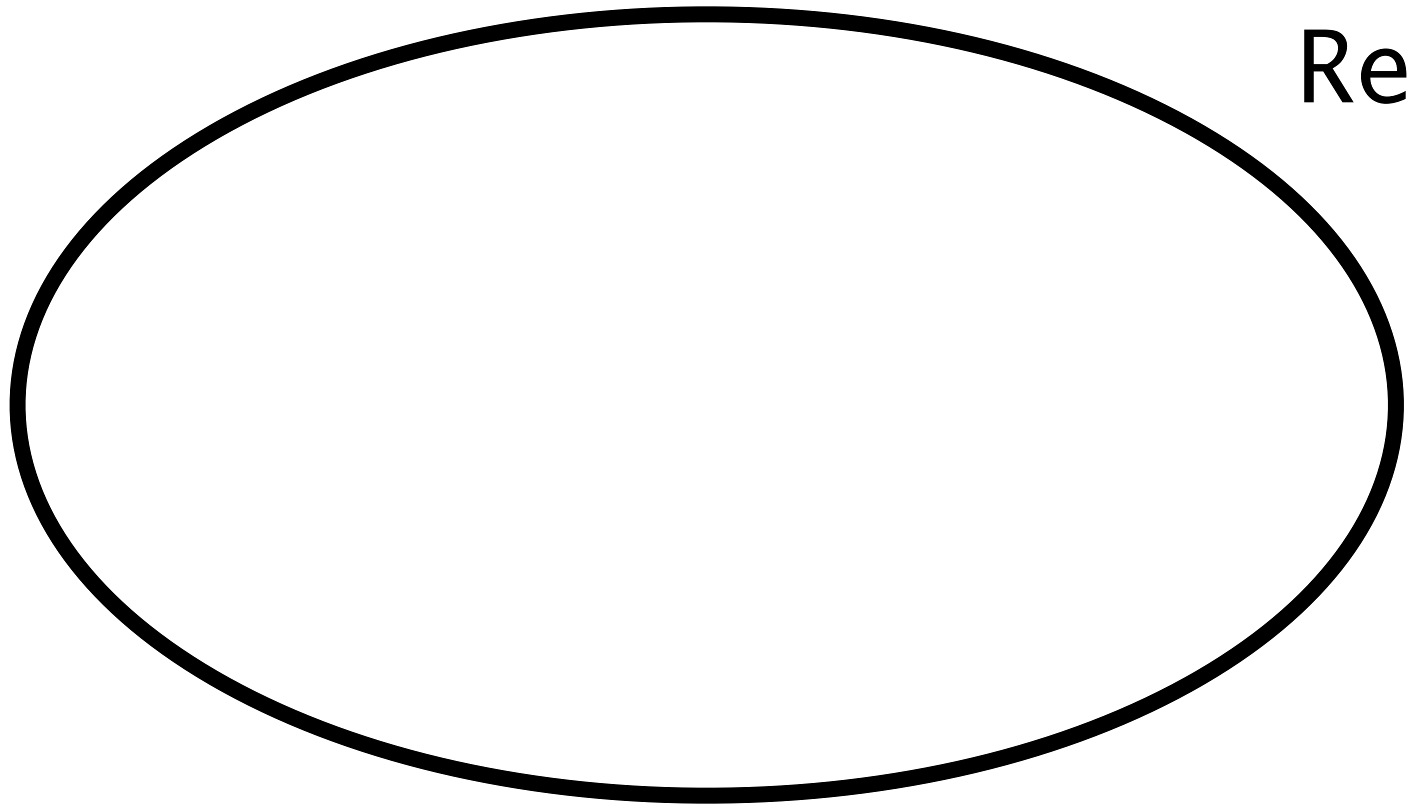
- "Repository" is a char vector of length n

f	i	b	r	a	t	i	o	n

- Basic patch is $a \leftrightarrow b$ @ i where $i<n$

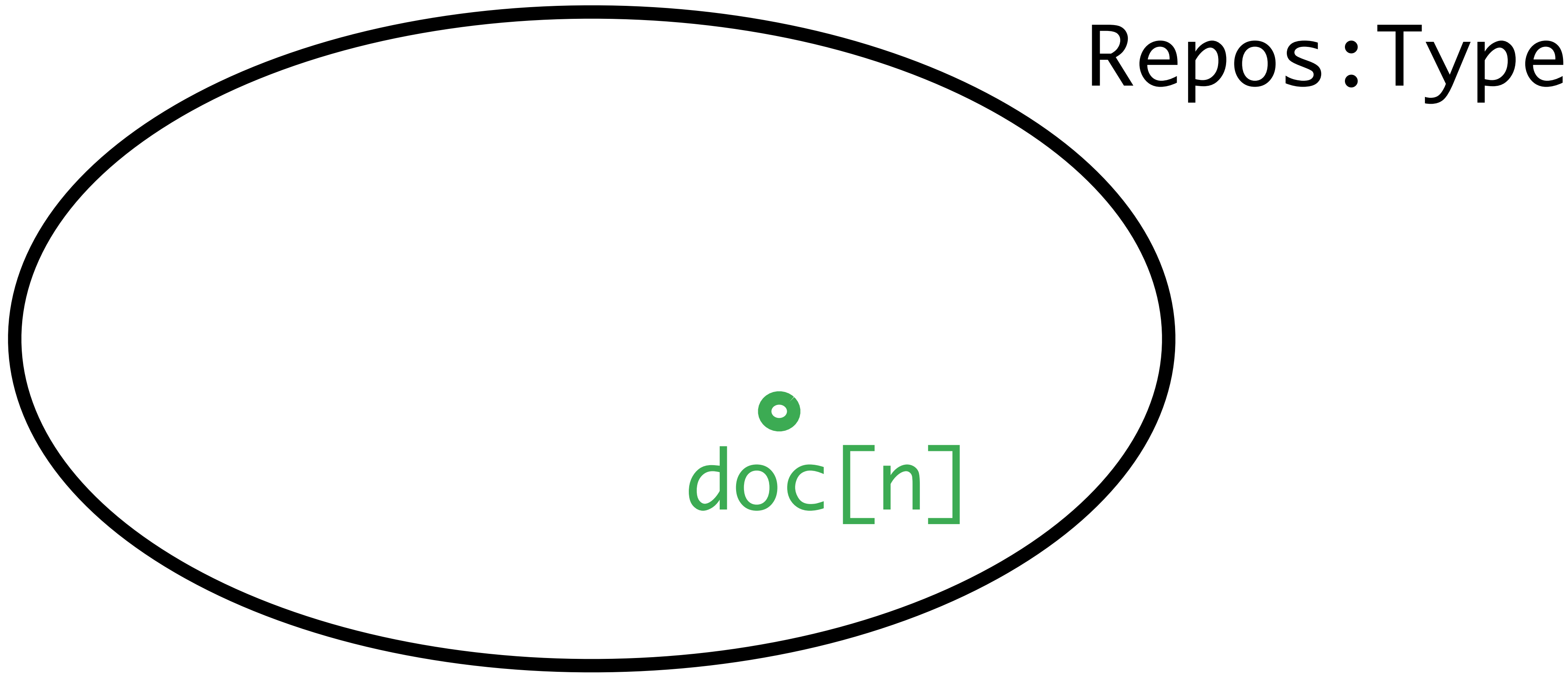


Patches as a HIT



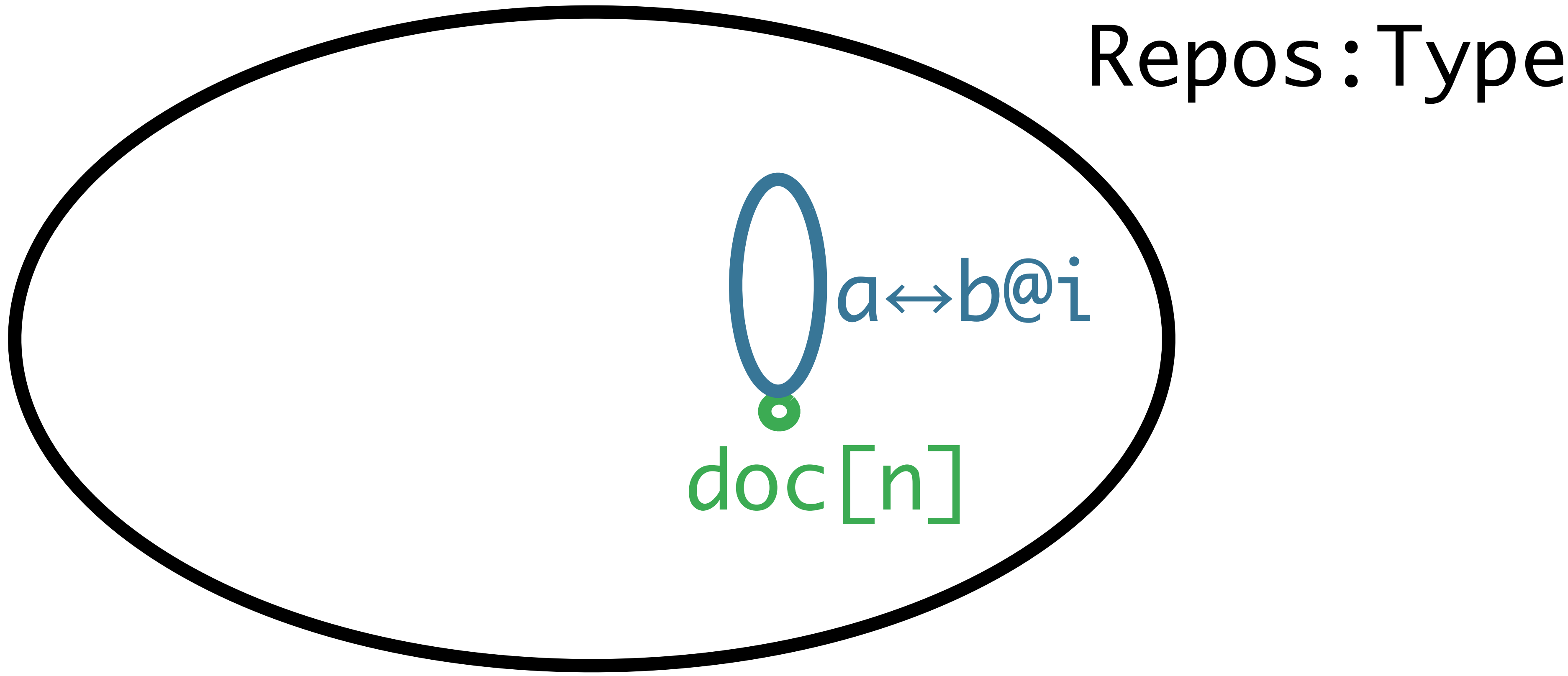
Repos : Type

Patches as a HIT



points describe repository contents

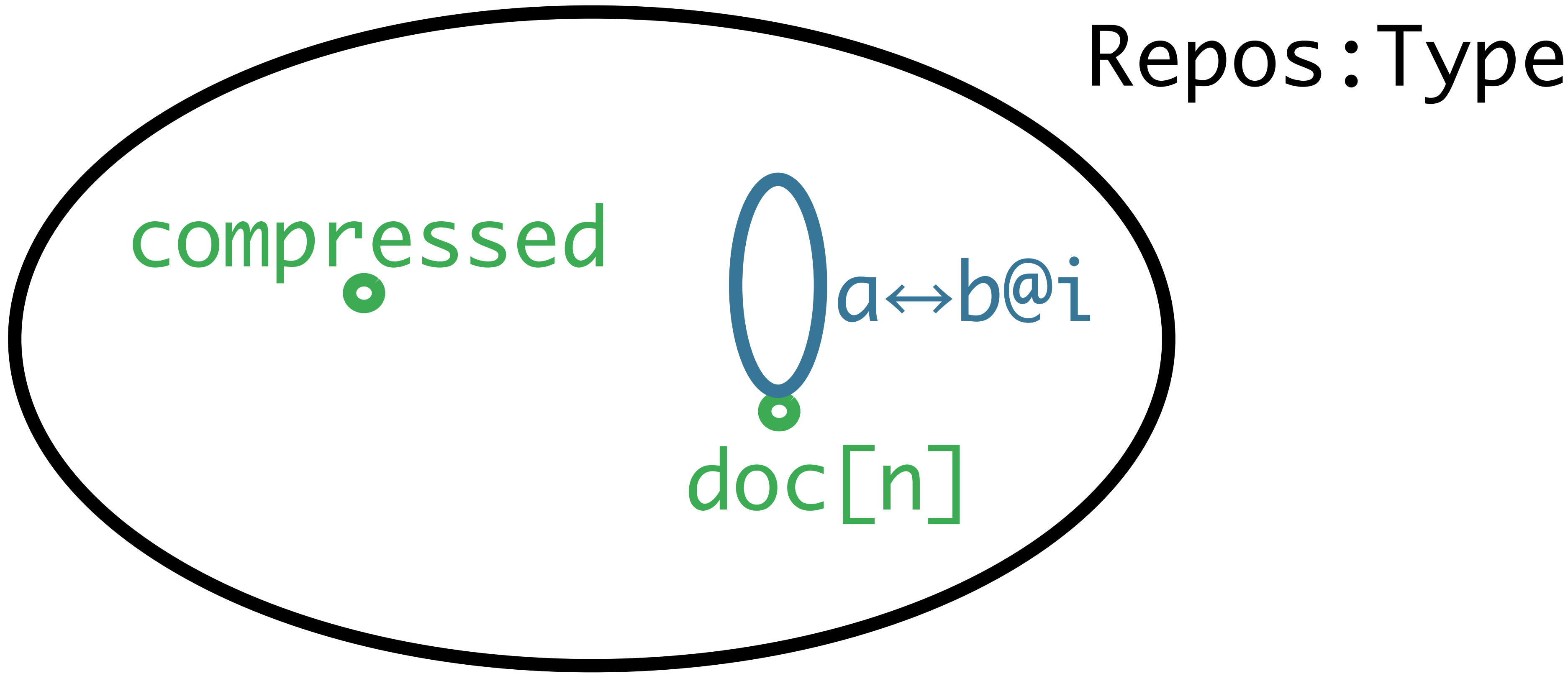
Patches as a HIT



points describe
repository contents

paths are patches

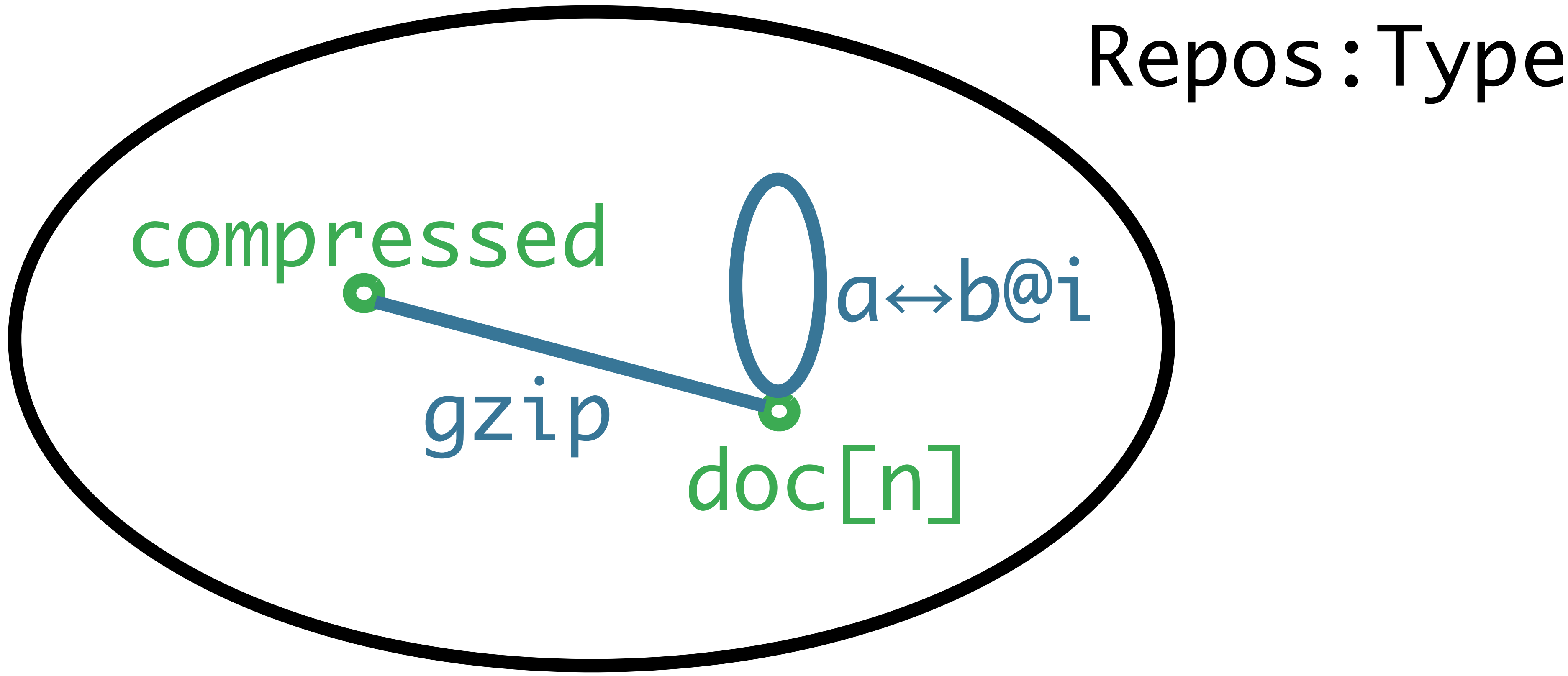
Patches as a HIT



points describe
repository contents

paths are patches

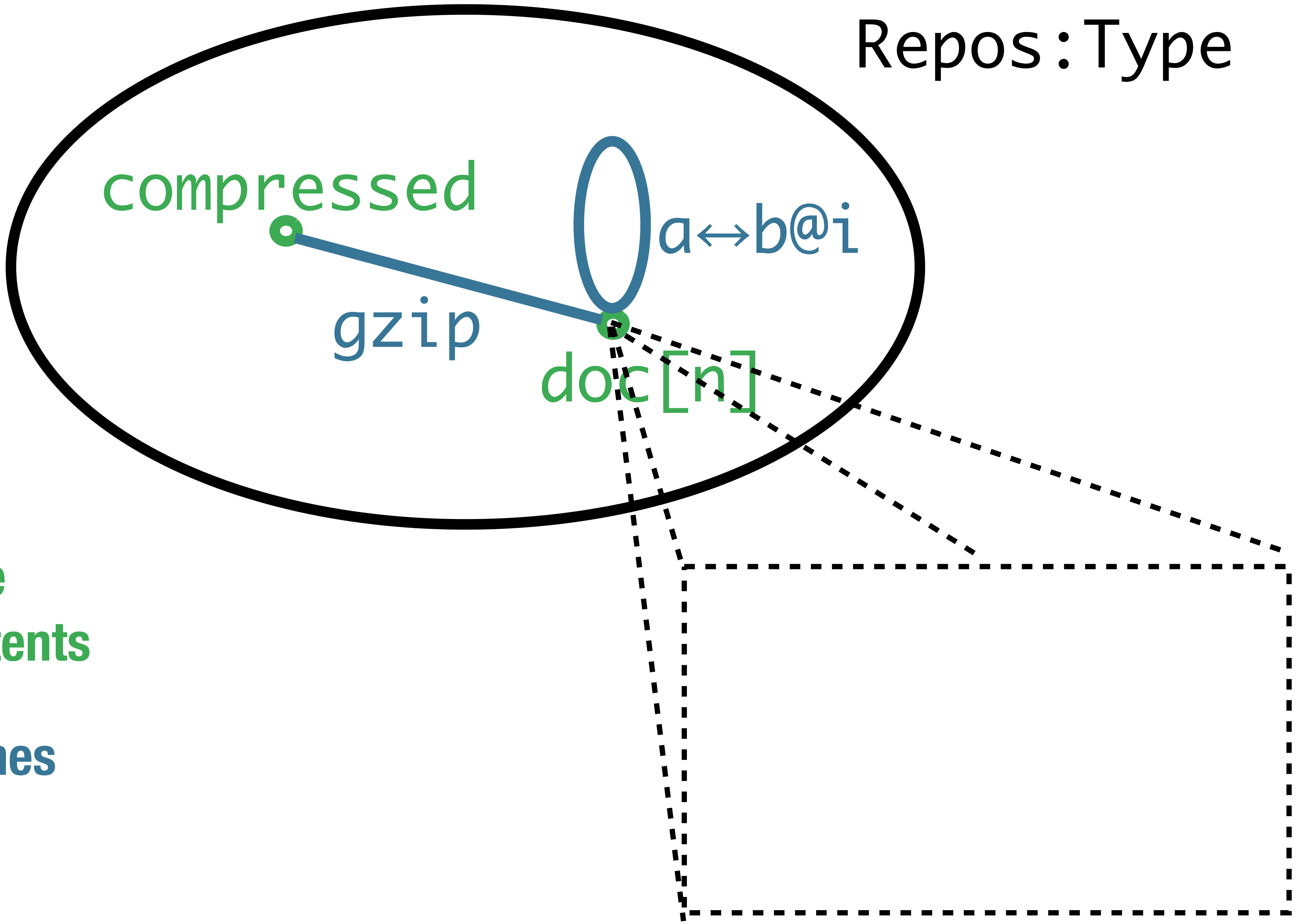
Patches as a HIT



points describe
repository contents

paths are patches

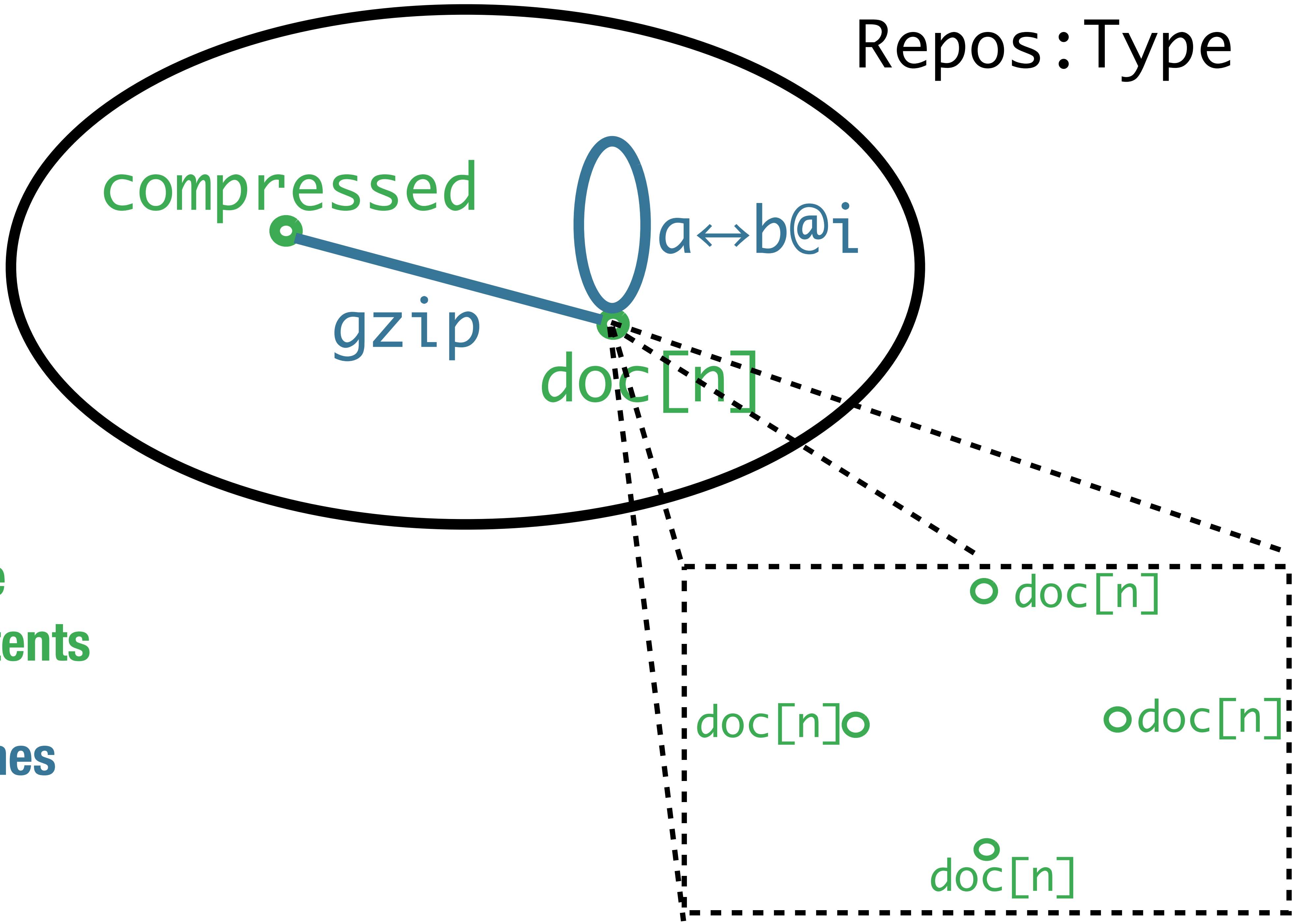
Patches as a HIT



points describe repository contents

paths are patches

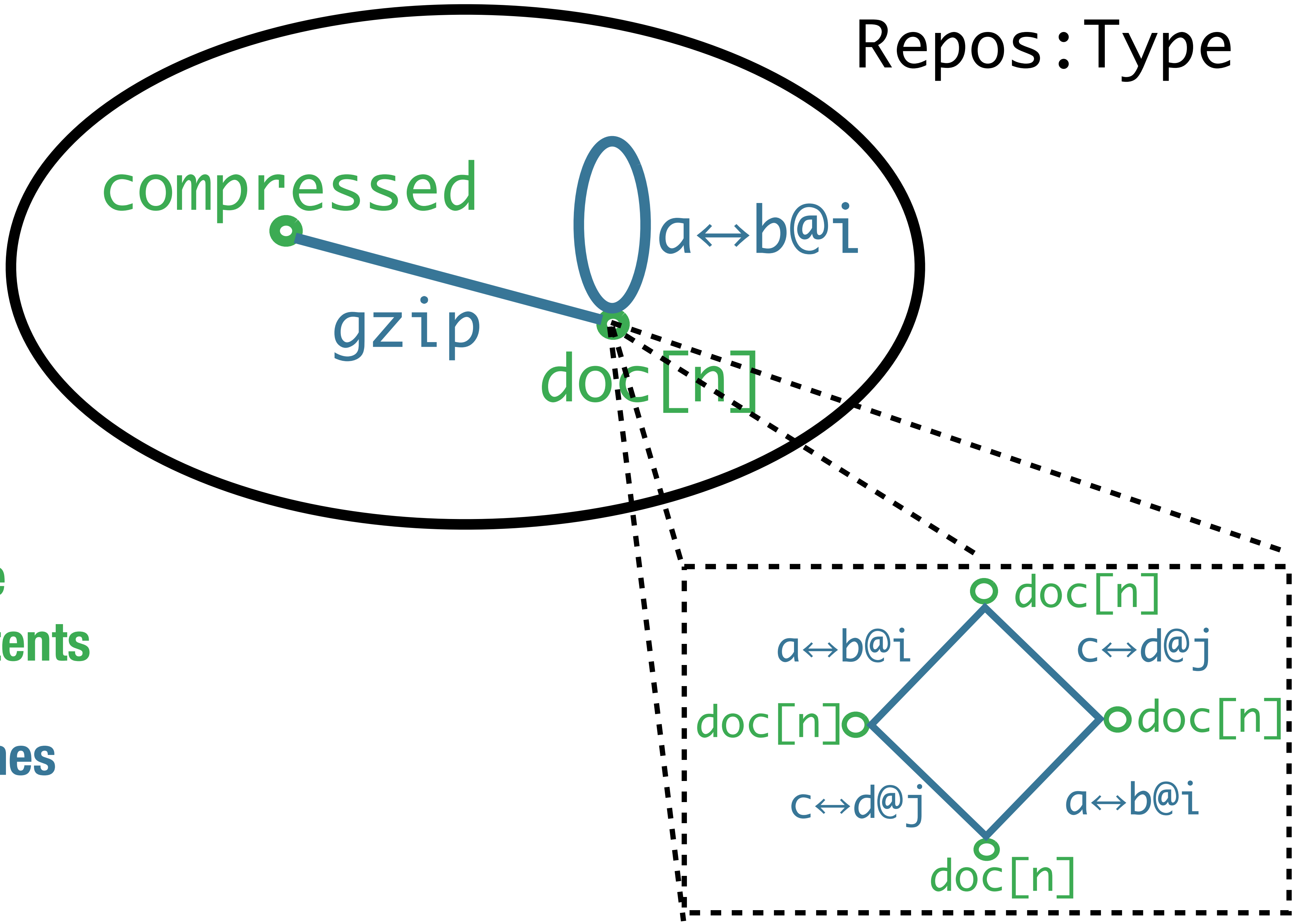
Patches as a HIT



points describe repository contents

paths are patches

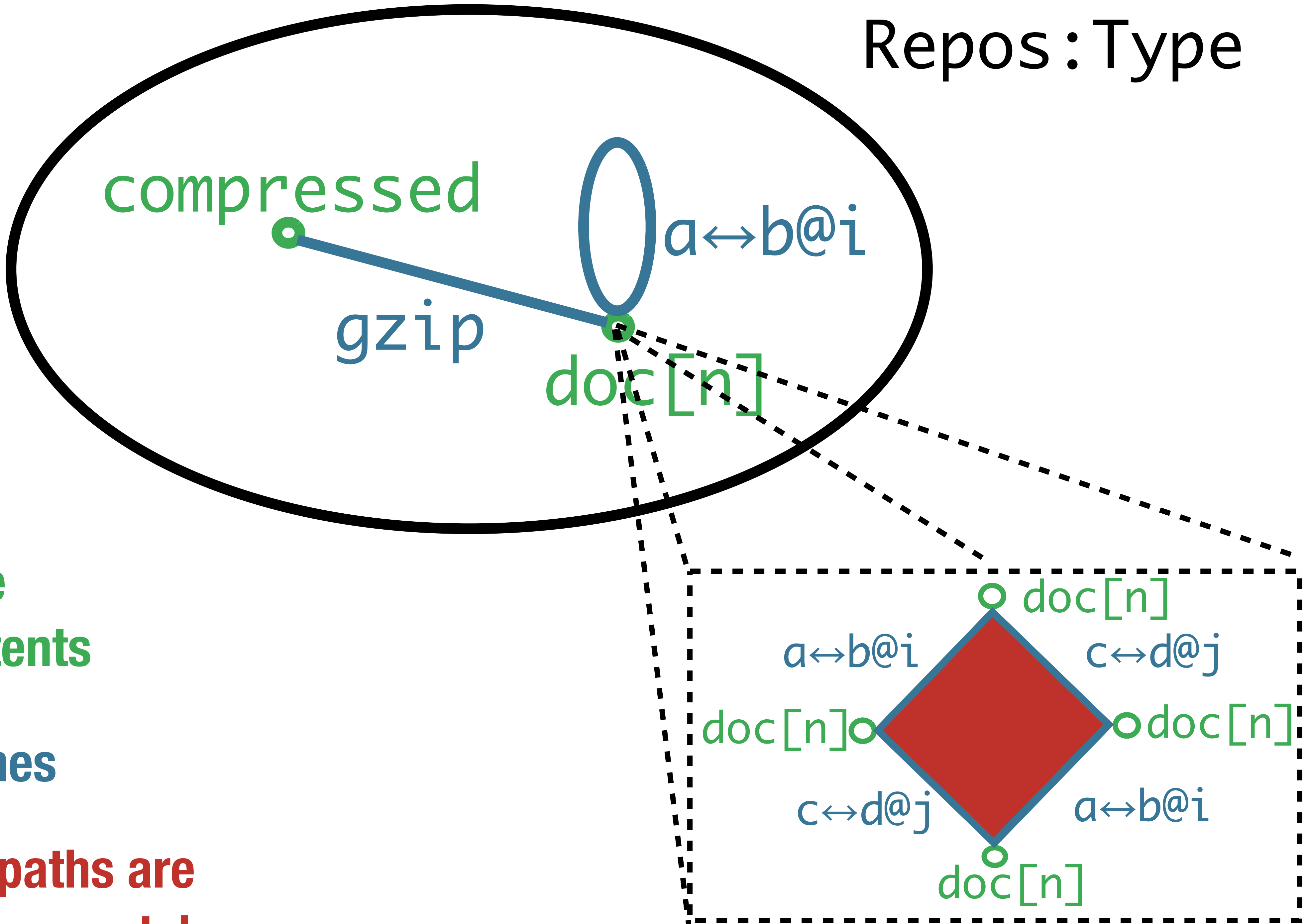
Patches as a HIT



points describe repository contents

paths are patches

Patches as a HIT

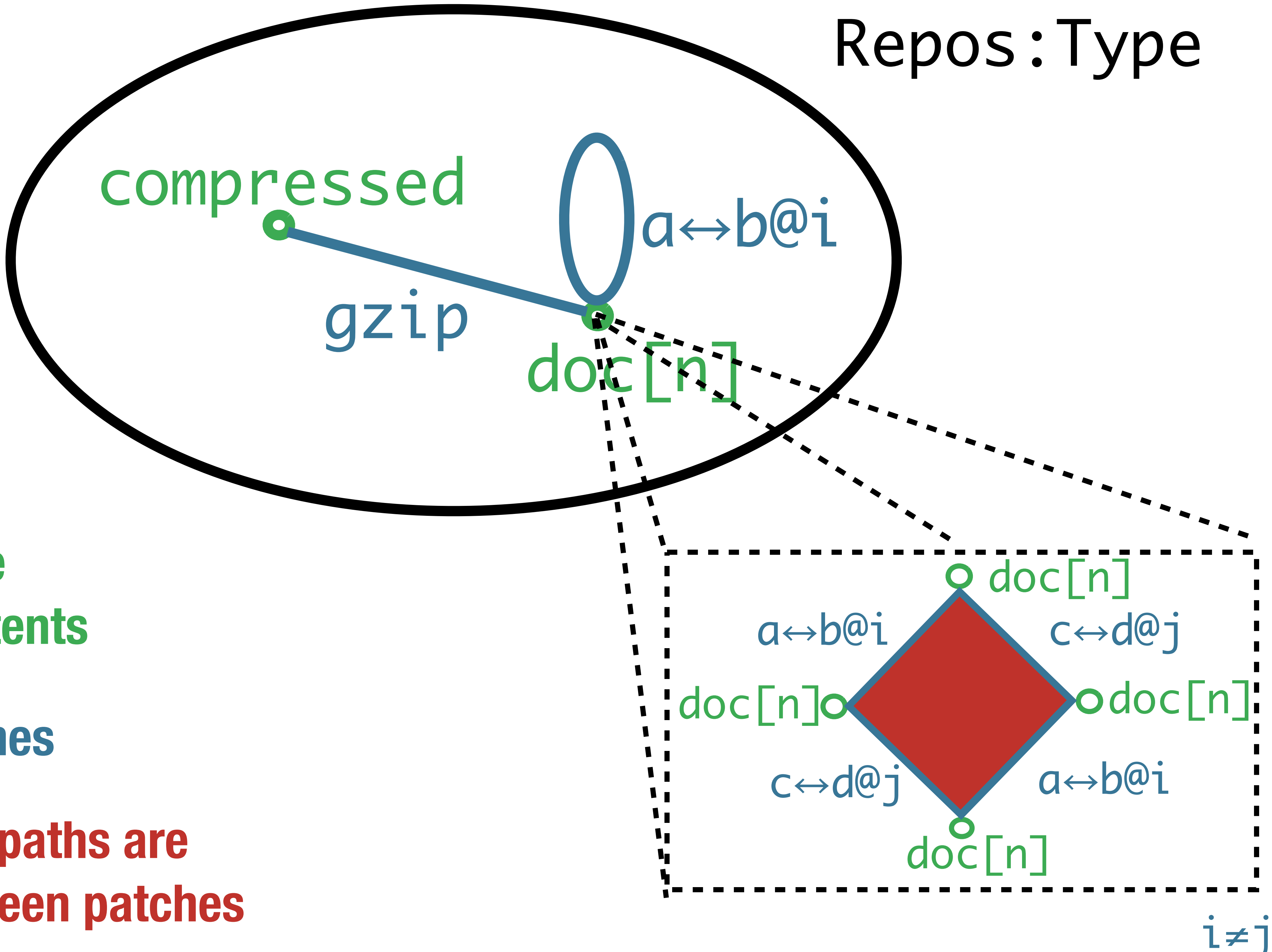


points describe repository contents

paths are patches

paths between paths are equations between patches

Patches as a HIT



points describe repository contents

paths are patches

paths between paths are equations between patches

Generators for HIT

Generators for HIT

```
Repos : Type
```

Generators for HIT

```
Repos : Type

doc[n]       : Repos
compressed : Repos
```

Generators for HIT

```
Repos : Type

doc[n]       : Repos
compressed : Repos

(a↔b@i) : doc[n] = doc[n] if a,b:Char, i<n
gzip : doc[n] = compressed
```

Generators for HIT

```
Repos : Type

doc[n]       : Repos
compressed : Repos

(a↔b@i) : doc[n] = doc[n] if a,b:Char, i<n
 gzip : doc[n] = compressed

commute:
 (a↔b at i)o(c↔d at j)     if i ≠ j
=(c↔d at j)o(a↔b at i)
```

Type: Patch

Elements:

```
id       : Patch
_∘_      : Patch → Patch → Patch
!        : Patch → Patch
_↔_at_ : Char → Char → Fin n → Patch
```

Equality:

```
(a↔b at i)o(c↔d at j)=
   (c↔d at j)o(a↔b at i)

...
id o p = p = p o id
po(qor) = (poq)or
!p o p = id = p o !p
p=p
p=q if q=p
p=r if p=q and q=r
!p = !p’ if p = p’
p o q = p’ o q’ if p = p’ and q = q’
```

Type: Repos

Points: doc[n]

Paths:

a↔b@i

Paths between paths:

```
commute :
(a↔b at i)o(c↔d at j)=
(c↔d at j)o(a↔b at i)
```

Repos recursion

To define a function `Repos → A`
it suffices to

Repos recursion

To define a function `Repos → A`
it suffices to

- map the element generators of `Repos` to elements of `A`

Repos recursion

To define a function `Repos → A`
it suffices to

- map the element generators of `Repos` to elements of `A`
- map the equality generators of `Repos` to equalities between the corresponding elements of `A`

Repos recursion

To define a function Repos → A
it suffices to

- map the element generators of Repos
  to elements of A
- map the equality generators of Repos
  to equalities between the corresponding elements of A
- map the equality-between-equality generators to
  equalities between the corresponding equalities in A

Repos recursion

To define a function `Repos → A`
it suffices to

- map the element generators of `Repos` to elements of `A`
- map the equality generators of `Repos` to equalities between the corresponding elements of `A`
- map the equality-between-equality generators to equalities between the corresponding equalities in `A`

All functions on Repos respect patches

All functions on patches respect patch equality

Interpreter

Goal is to define:

```
interp : doc[n] = doc[n]
       → Bijection (Vec Char n) (Vec Char n)
```

Interpreter

Goal is to define:

```
interp : doc[n] = doc[n]
       → Bijection (Vec Char n) (Vec Char n)
interp(id) = (λx.x, …)
interp(q o p) = (interp q) o_b (interp p)
interp(!p) = !_b (interp p)
interp(a↔b@i) = swapat a b i
```

Interpreter

Goal is to define:

```
interp : doc[n] = doc[n]
       → Bijection (Vec Char n) (Vec Char n)
interp(id) = (λx.x, …)
interp(q o p) = (interp q) o_b (interp p)
interp(!p) = !_b (interp p)
interp(a↔b@i) = swapat a b i
```

But only tool available is RepoDesc recursion: no direct recursion over paths

```
interp : doc[n]=doc[n]
       → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i
```

Need to pick A *and define*

```
  I(doc[n]) := … : A
```

```
  I₁(a↔b@i) := … : I(doc[n]) = I(doc[n])
```

```
  I₂(compose) := …
```

```
interp : doc[n]=doc[n]
       → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i
```

Key idea: pick A = Type *and define*

```
I(doc[n]) := … : Type
I₁(a↔b@i) := … : I(doc[n]) = I(doc[n])
I₂(compose) := …
```

```
interp : doc[n]=doc[n]
        → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i
```

Key idea: pick A = Type *and define*

```
I(doc[n]) := Vec Char n : Type
```

I_1`(a↔b@i) := … : I(doc[n]) = I(doc[n])`

I_2`(compose) := …`

```
interp : doc[n]=doc[n]
       → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i
```

Key idea: pick A = Type *and define*

```
I(doc[n]) := Vec Char n : Type
I₁(a↔b@i) := … : Vec Char n = Vec Char n
I₂(compose) := …
```

```
interp : doc[n]=doc[n]
       → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i
```

Key idea: pick `A = Type` *and define*

```
I(doc[n]) := Vec Char n : Type
I₁(a↔b@i) := ua(swapat a b i)
                 : Vec Char n = Vec Char n
I₂(compose) := …
```

```
interp : doc[n]=doc[n]
       → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i
```

Key idea: pick A = Type *and define*

```
I(doc[n]) := Vec Char n : Type
```

I_1`(a↔b@i) := ua(swapat a b i)`

```
             : Vec Char n = Vec Char n
```

I_2`(compose) := …`

univalence

```
interp : doc[n]=doc[n]
        → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i
```

Key idea: pick A = Type *and define*

```
I(doc[n]) := Vec Char n : Type
```

I_1`(a↔b@i) := ua(swapat a b i)`

```
                   : Vec Char n = Vec Char n
```

I_2`(compose) :=` <proof about swapat>

```
interp : doc[n]=doc[n]
       → Bijection (Vec Char n) (Vec Char n)
```

$\mathtt{interp}(p) = \mathtt{ua}^{-1}(\mathtt{I}_1(p))$

Key idea: pick A = Type *and define*

$\mathtt{I}(\mathtt{doc}[n]) := \mathtt{Vec\ Char}\ n : \mathtt{Type}$

$\mathtt{I}_1(a \leftrightarrow b@i) := \mathtt{ua}(\mathtt{swapat}\ a\ b\ i)$

$\quad : \mathtt{Vec\ Char}\ n = \mathtt{Vec\ Char}\ n$

$\mathtt{I}_2(\mathtt{compose}) :=$ <proof about swapat>

```
interp : doc[n]=doc[n]
       → Bijection (Vec Char n) (Vec Char n)
```

$\mathtt{interp(p)} = \mathtt{ua}^{-1}(\mathtt{I}_1(\mathtt{p}))$

Satisfies the desired equations (as propositional equalities):

```
interp(id) = (λx.x, …)
interp(q o p) = (interp q) o_b (interp p)
interp(!p) = !_b (interp p)
interp(a↔b@i) = swapat a b i
```

Summary

Summary

- `I : Repos → Type` interprets Repos as Types, patches as bijections, satisfying patch equalities

Summary

- `I : Repos → Type` interprets `Repos` as `Type`s, patches as bijections, satisfying patch equalities
- Higher inductive elim. defines functions that respect equality: you specify what happens on the generators; homomorphically extended to `id`,o,`!`,...

Summary

- `I : Repos → Type` interprets `Repos` as `Type`s, patches as bijections, satisfying patch equalities
- Higher inductive elim. defines functions that respect equality: you specify what happens on the generators; homomorphically extended to `id`,o,`!`,...
- Univalence lets you give a computational model of equality proofs (here, patches); guaranteed to satisfy laws

Summary

- `I : Repos → Type` interprets `Repos` as `Type`s, patches as bijections, satisfying patch equalities
- Higher inductive elim. defines functions that respect equality: you specify what happens on the generators; homomorphically extended to `id`,o,`!`,...
- Univalence lets you give a computational model of equality proofs (here, patches); guaranteed to satisfy laws
- Shorter definition and code:
  1 basic patch & 4 basic axioms of equality, instead of
  4 patches & 14 equations

Operational semantics

- Can’t run these programs yet
- Some special cases known, some recent progress:
  Licata&Harper, POPL’12
  Coquand&Barras, ’13
  Shulman, ’13
  Bezem&Coquand&Huber, ’13
- Would support proof automation and programming applications

Outline

1. Certified homotopy theory
2. Certified software

Homotopy Type Theory