Seifert-van Kampen Theorem in Homotopy Type Theory

[Toronto version]

Michael Shulman @ U of San Diego
* **Favonia @ CMU**

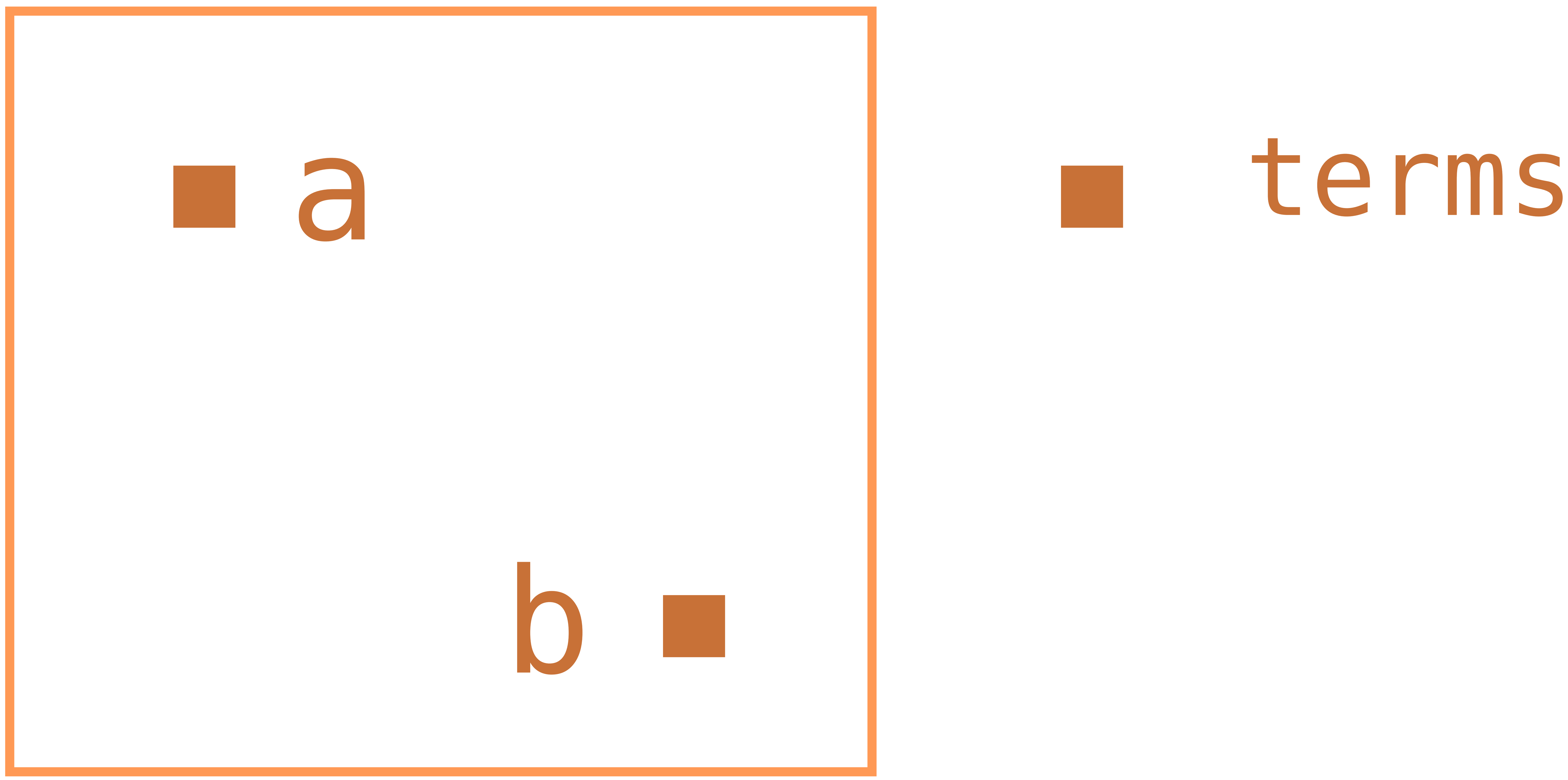
Homotopy Type Theory

* Type theory <-> topology

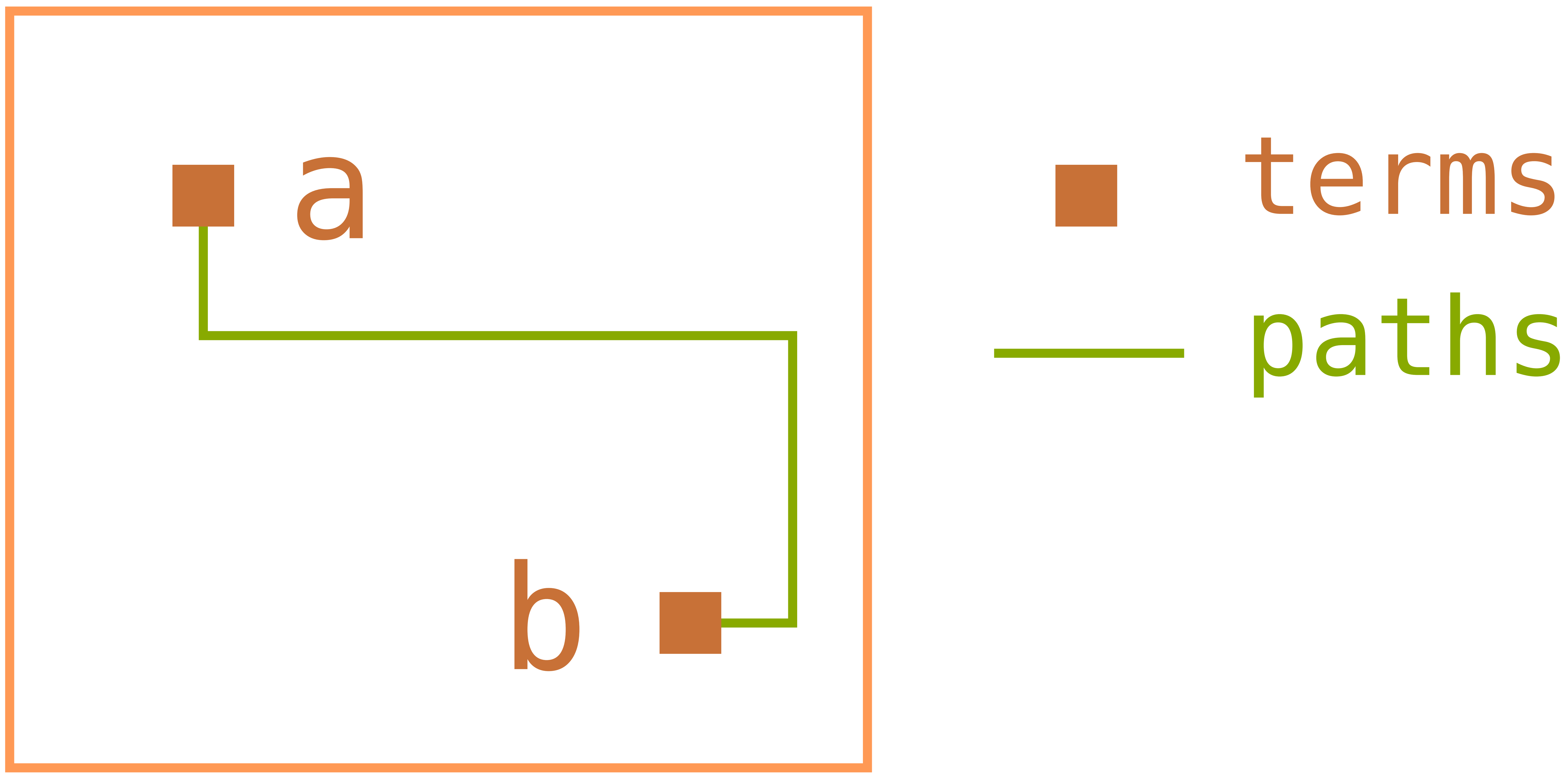
  - types ~= spaces
  - terms ~= points
  - functions ~= continuous maps
  - identifications ~= paths

* Non-trivial identifications

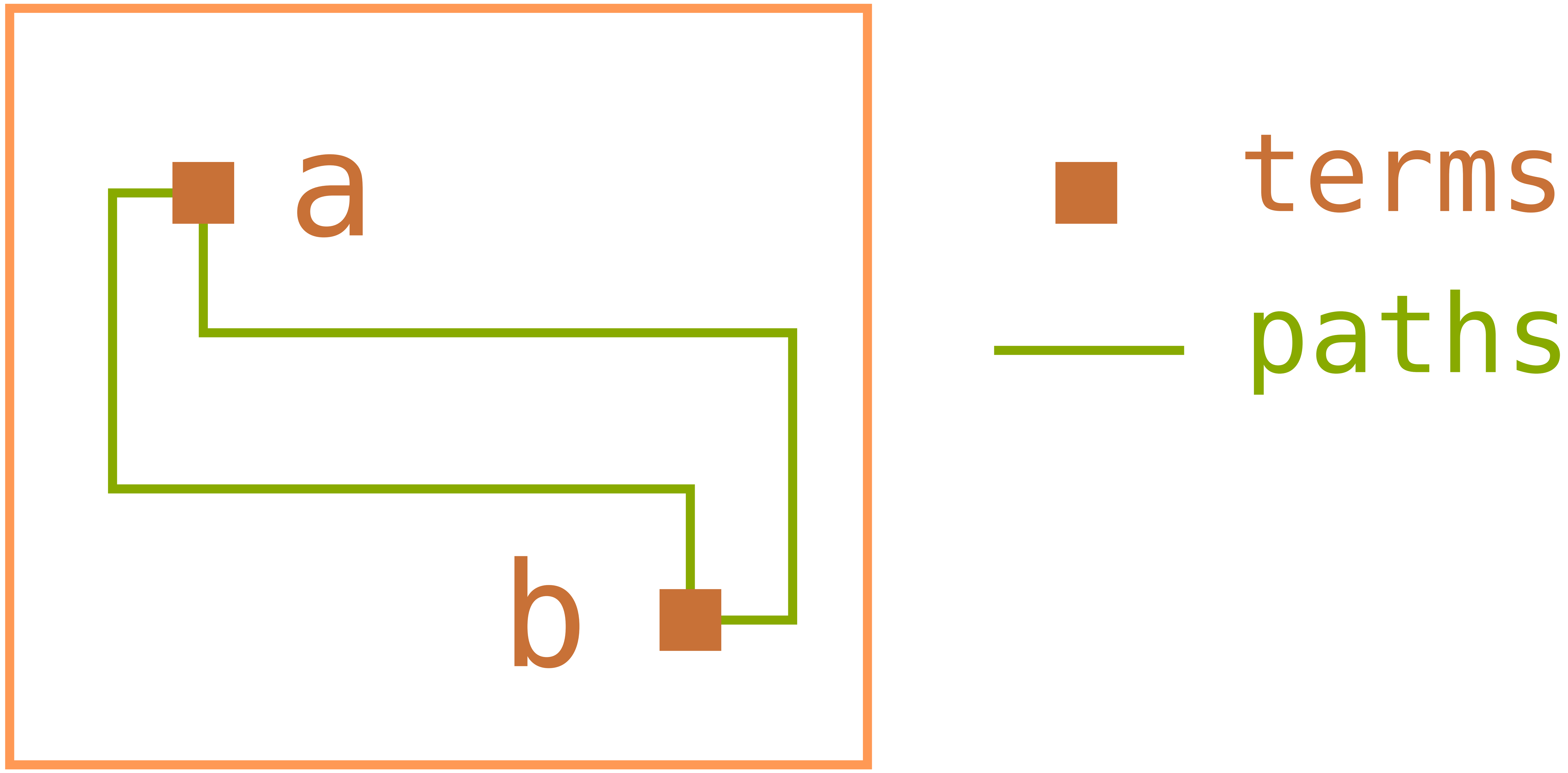
Iterated Paths



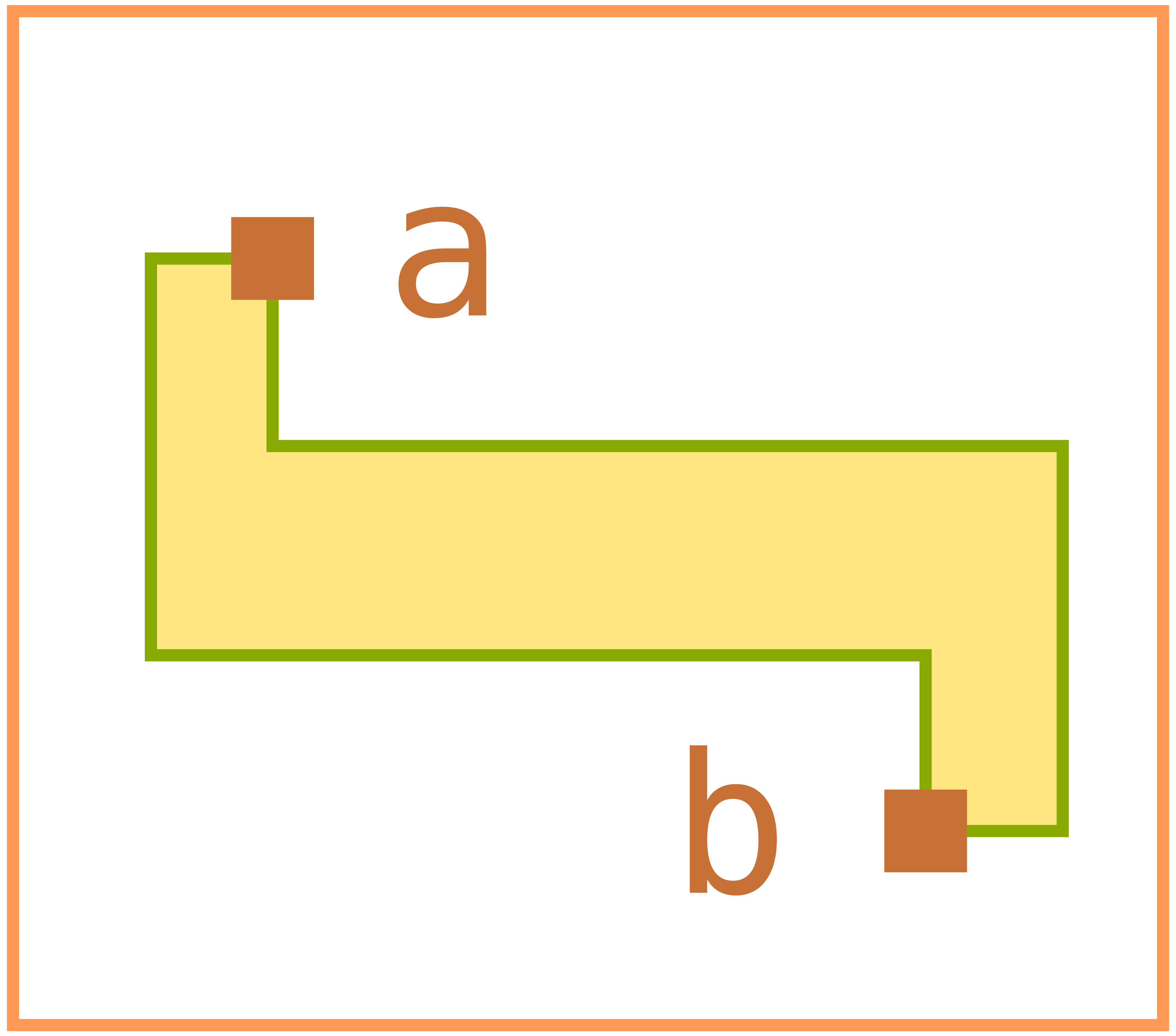
Iterated Paths



Iterated Paths



Iterated Paths

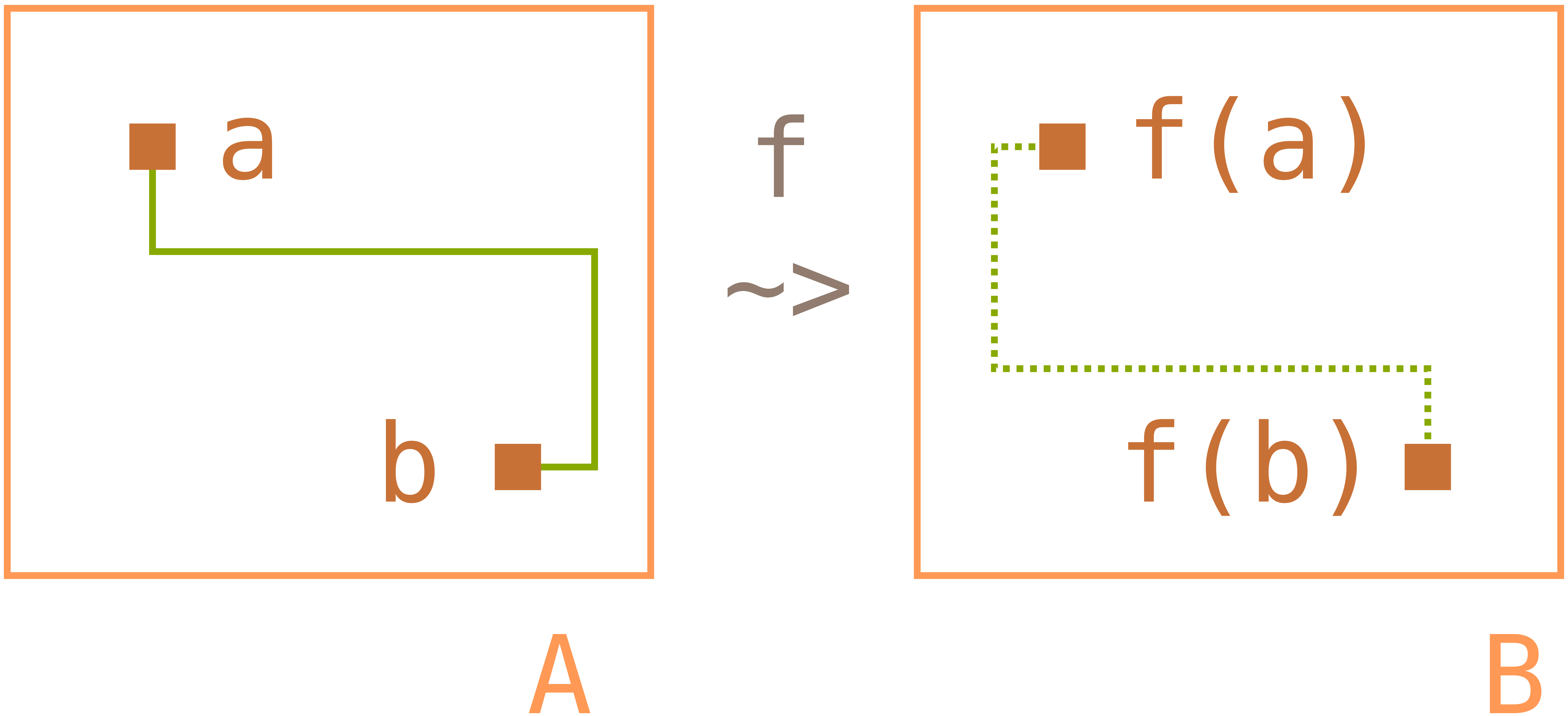


terms

paths

paths of paths

Functorial



Subject of Study

fundamental groups of pushouts

Subject of Study

fundamental groups of pushouts

"structure of loops"

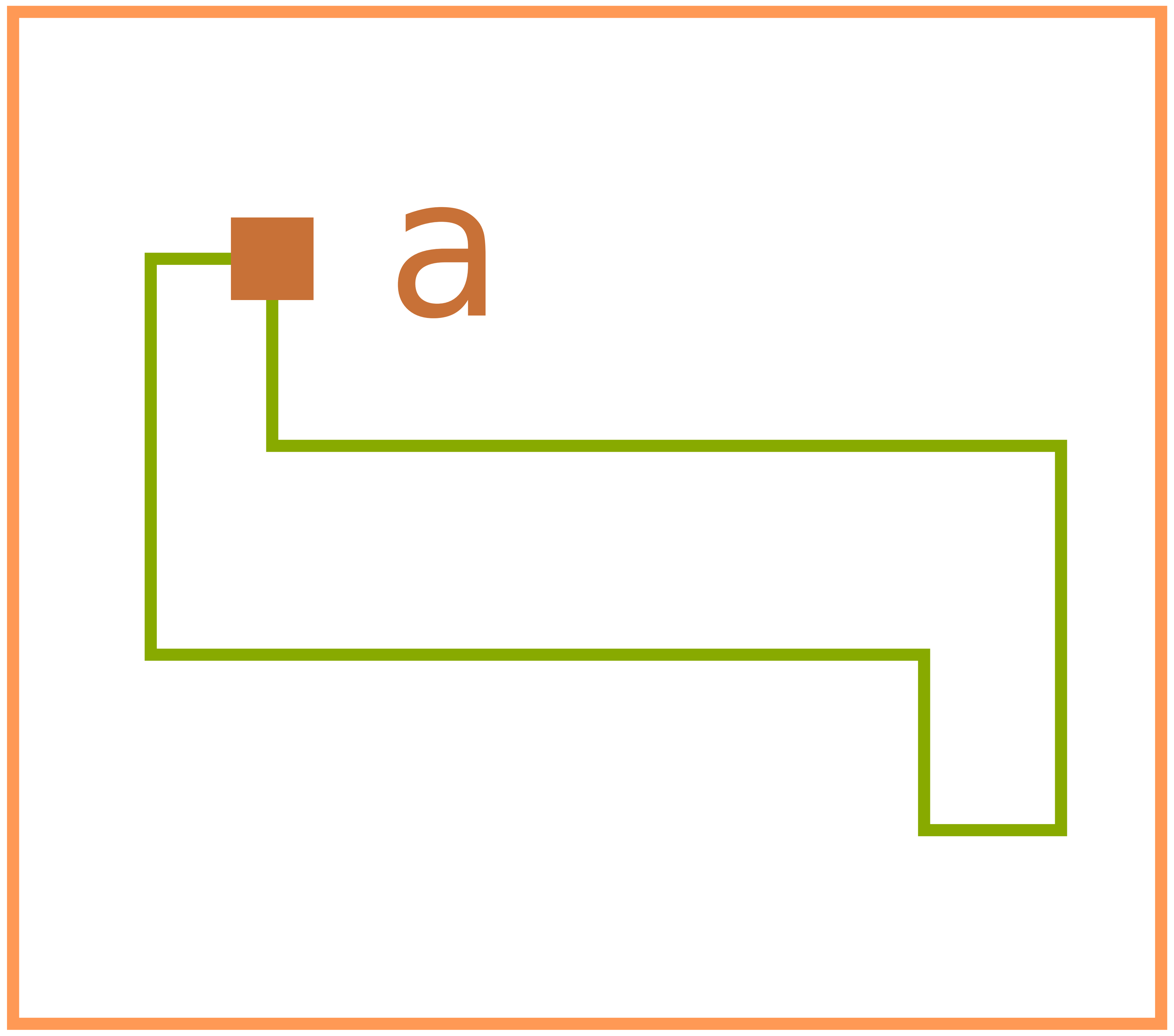
Subject of Study

fundamental groups of pushouts

"structure of loops"

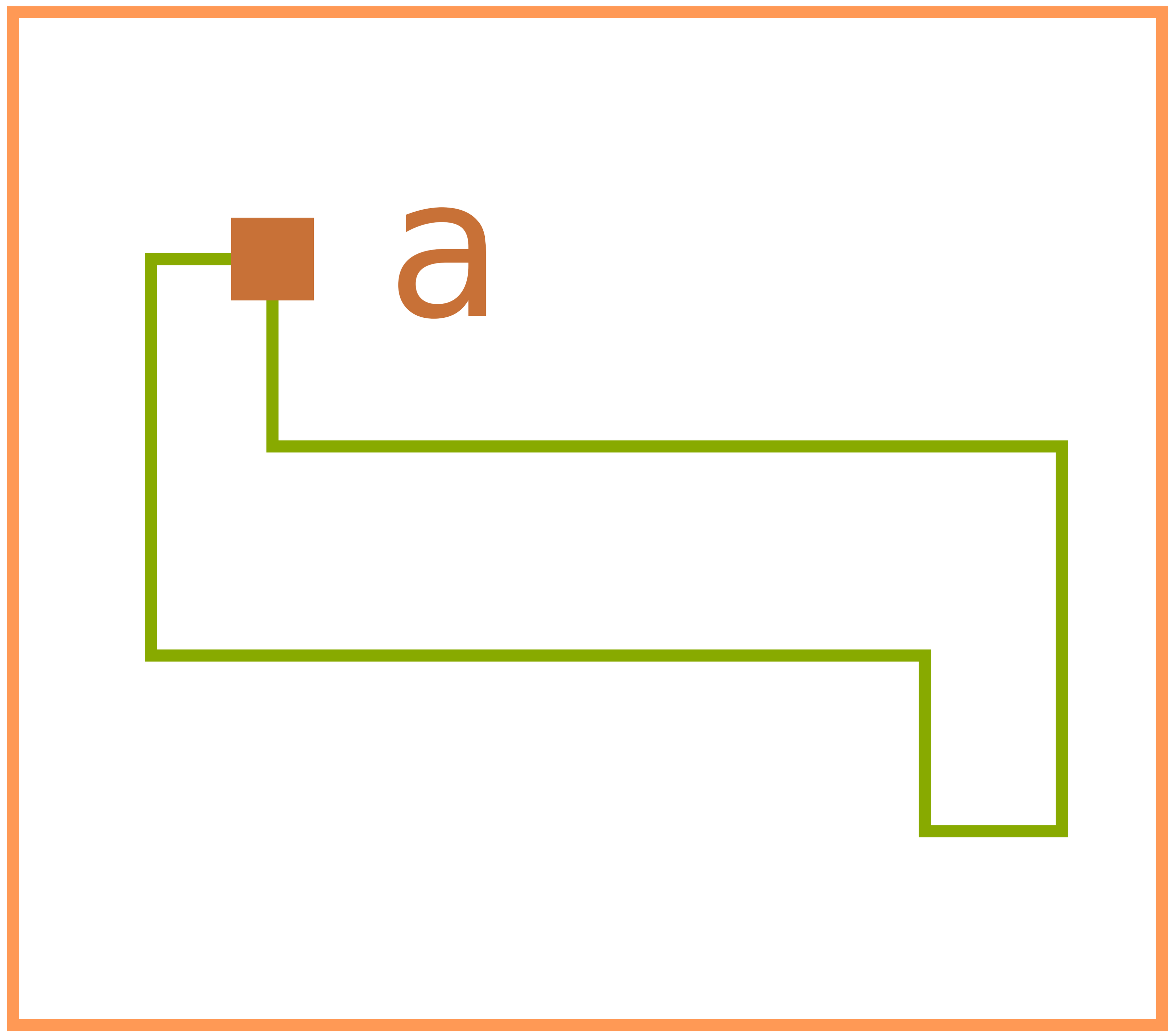
"disjoint union added with bridges"

Fundamental Groups



(unique) ways to travel from a to a

Fundamental Groups

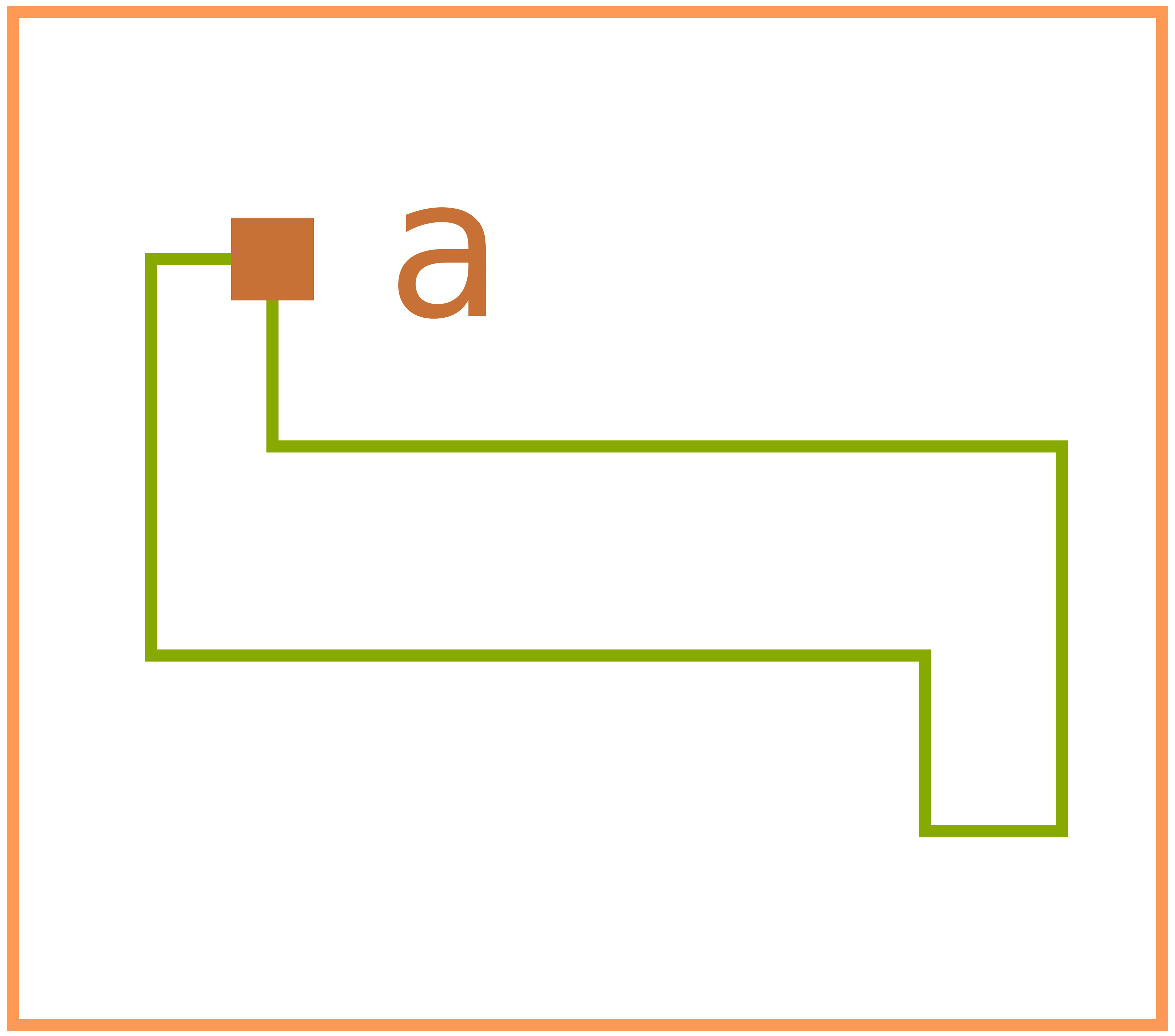


(unique) ways to travel from a to a

here they correspond to integers

positive <--> clockwise
negative <--> counter
zero <--> staying

Fundamental Groups



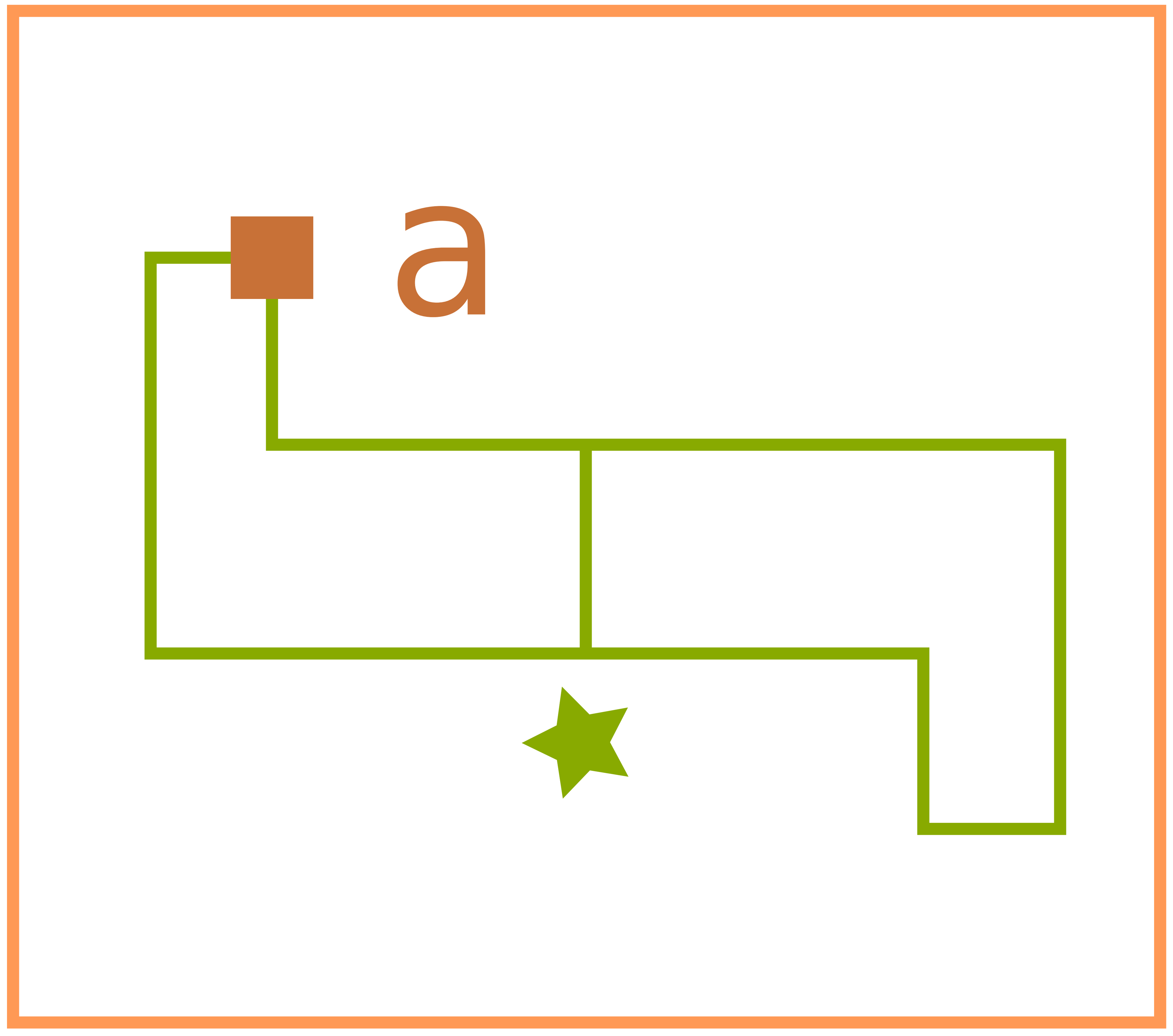
(unique) ways to travel from a to a

here they correspond to integers

positive <--> clockwise
negative <--> counter
zero <--> staying

Trunc 0 (a == a) ~= Z

Fundamental Groups



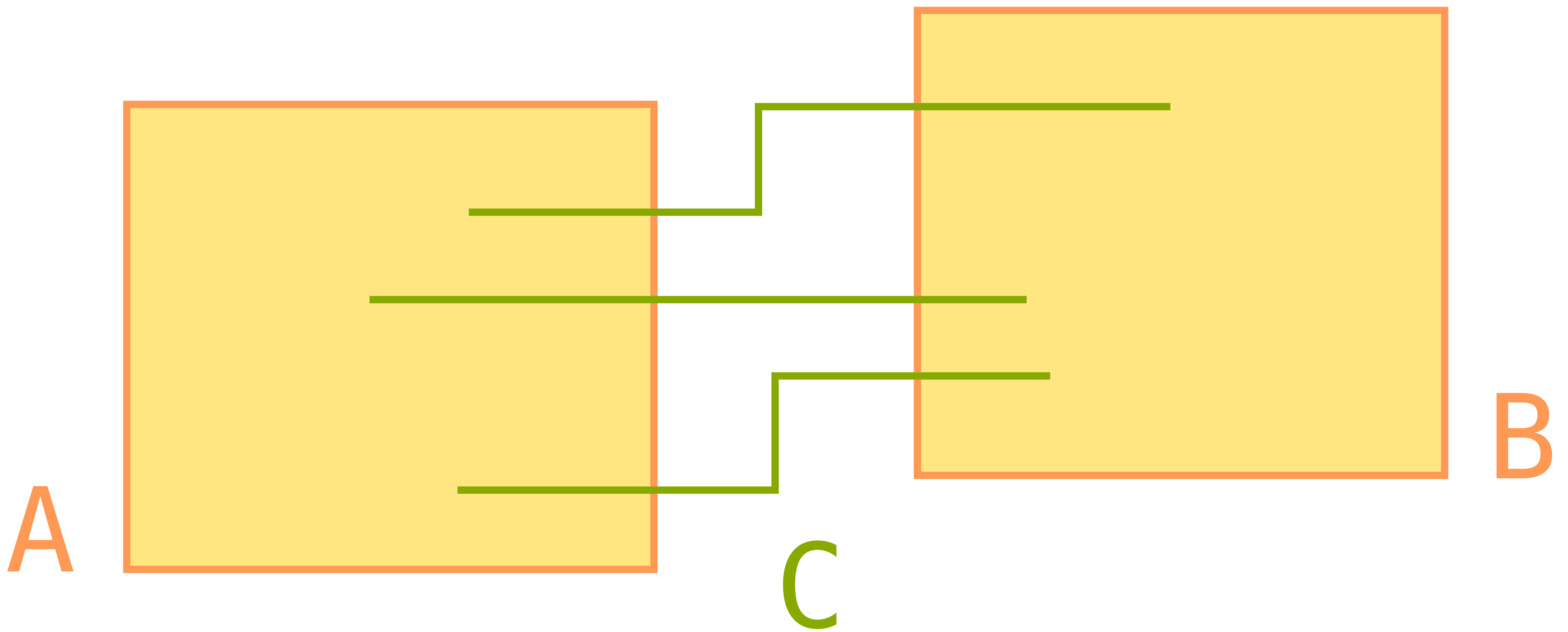
(unique) ways to travel from a to a

much more if a new path ★ is added

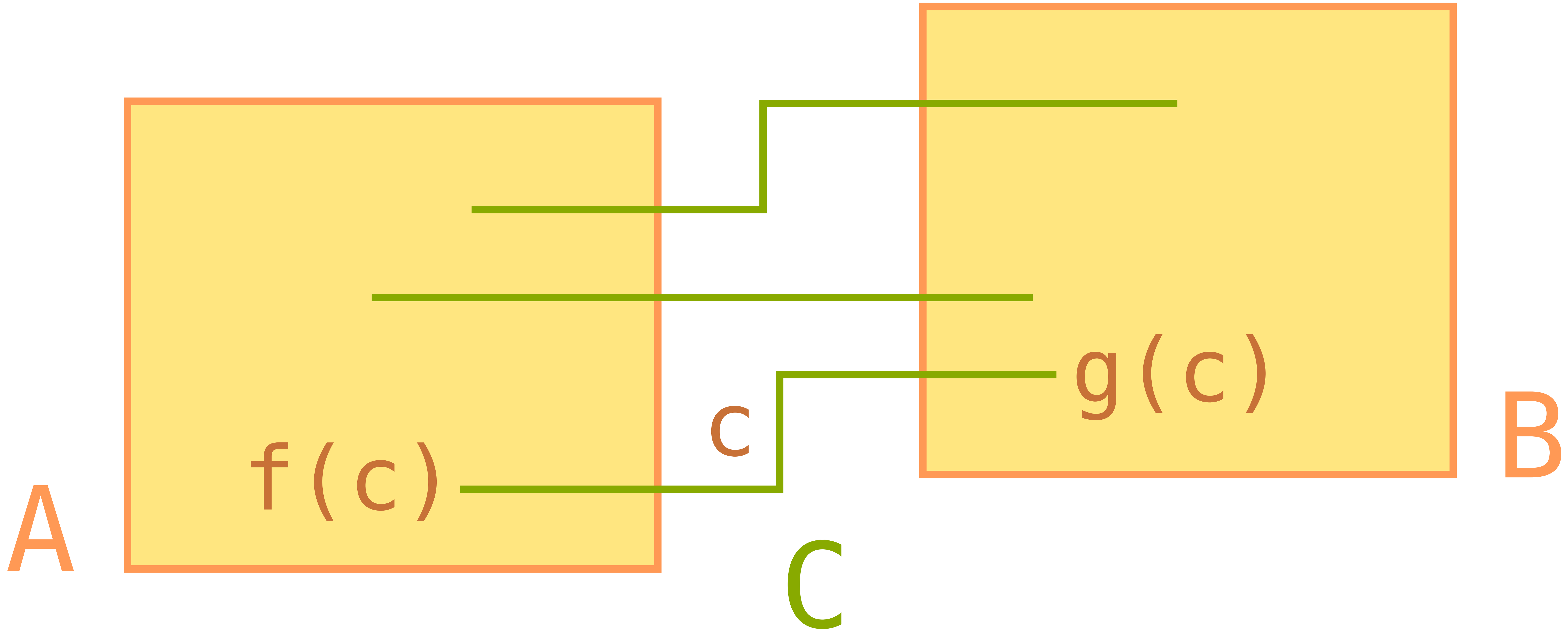
Trunc 0 (a == a) ~= Z * Z

(free product)

(Homotopy) Pushouts

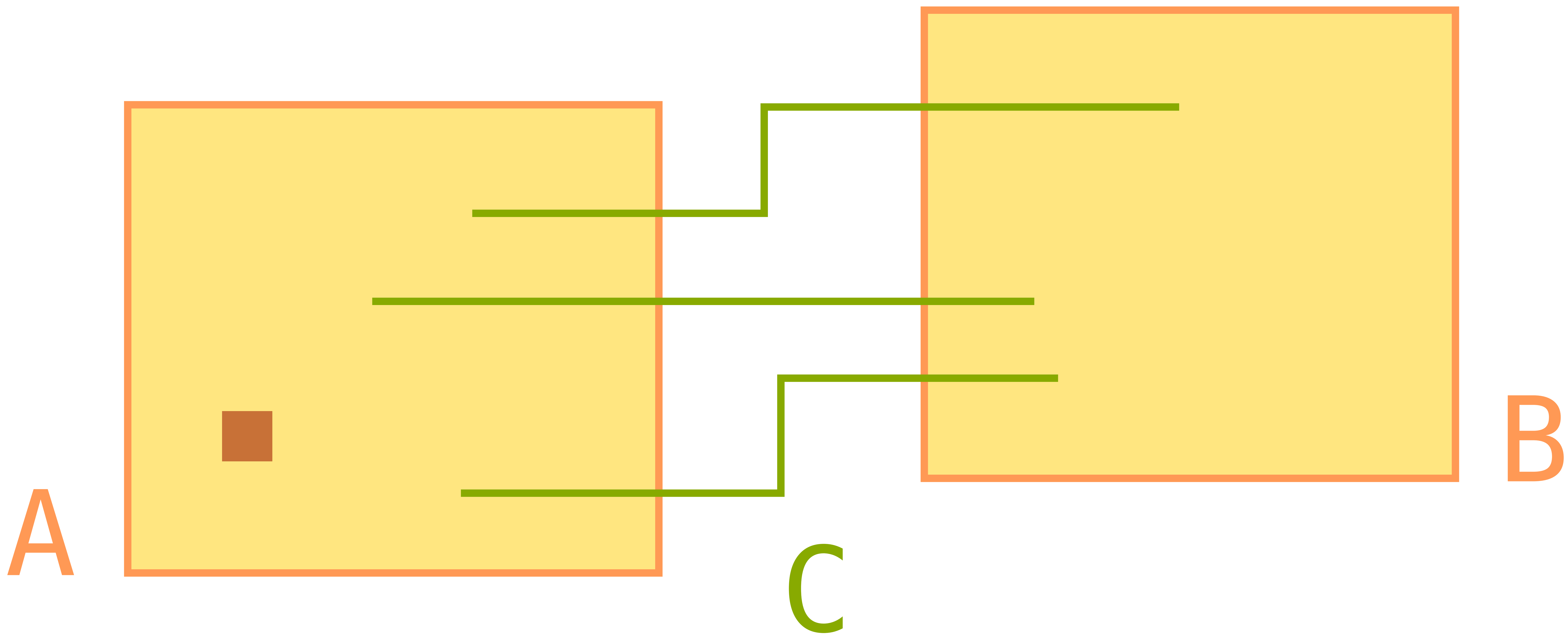


(Homotopy) Pushouts



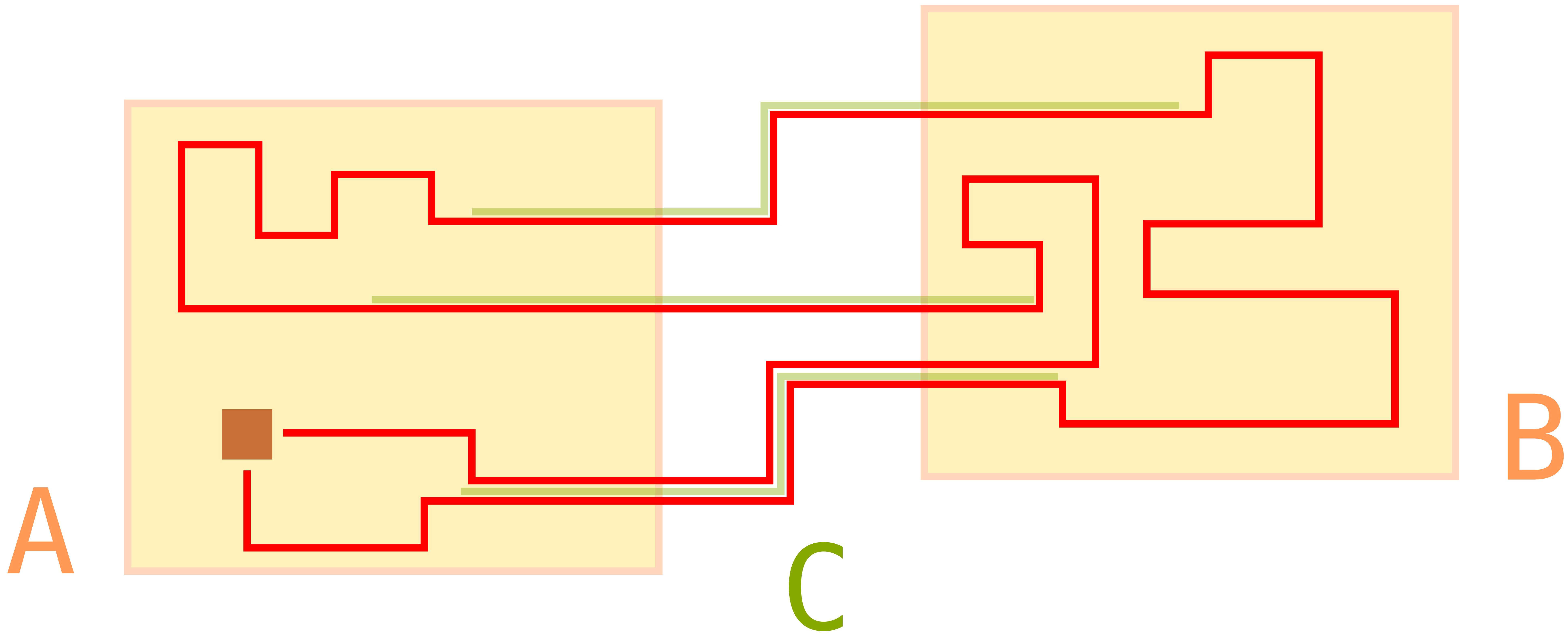

```
data Pushout (A B C : Type)
             (f : C -> A) (g : C -> B) : Type where
  left  : A -> Pushout A B C f g
  right : B -> Pushout A B C f g
  glue  : (c : C) -> left (f c) == right (g c)
```

(Homotopy) Pushouts

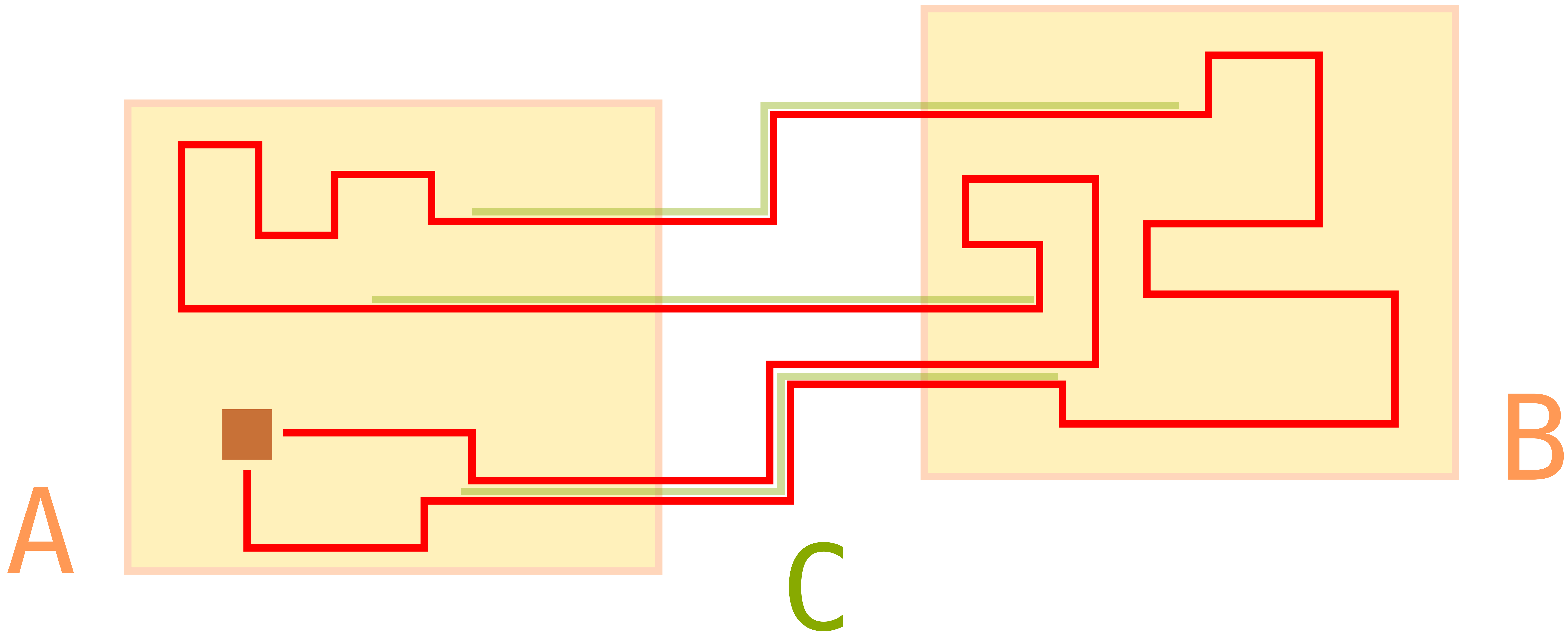


ways to travel from ■ to ■ ?

(Homotopy) Pushouts



(Homotopy) Pushouts



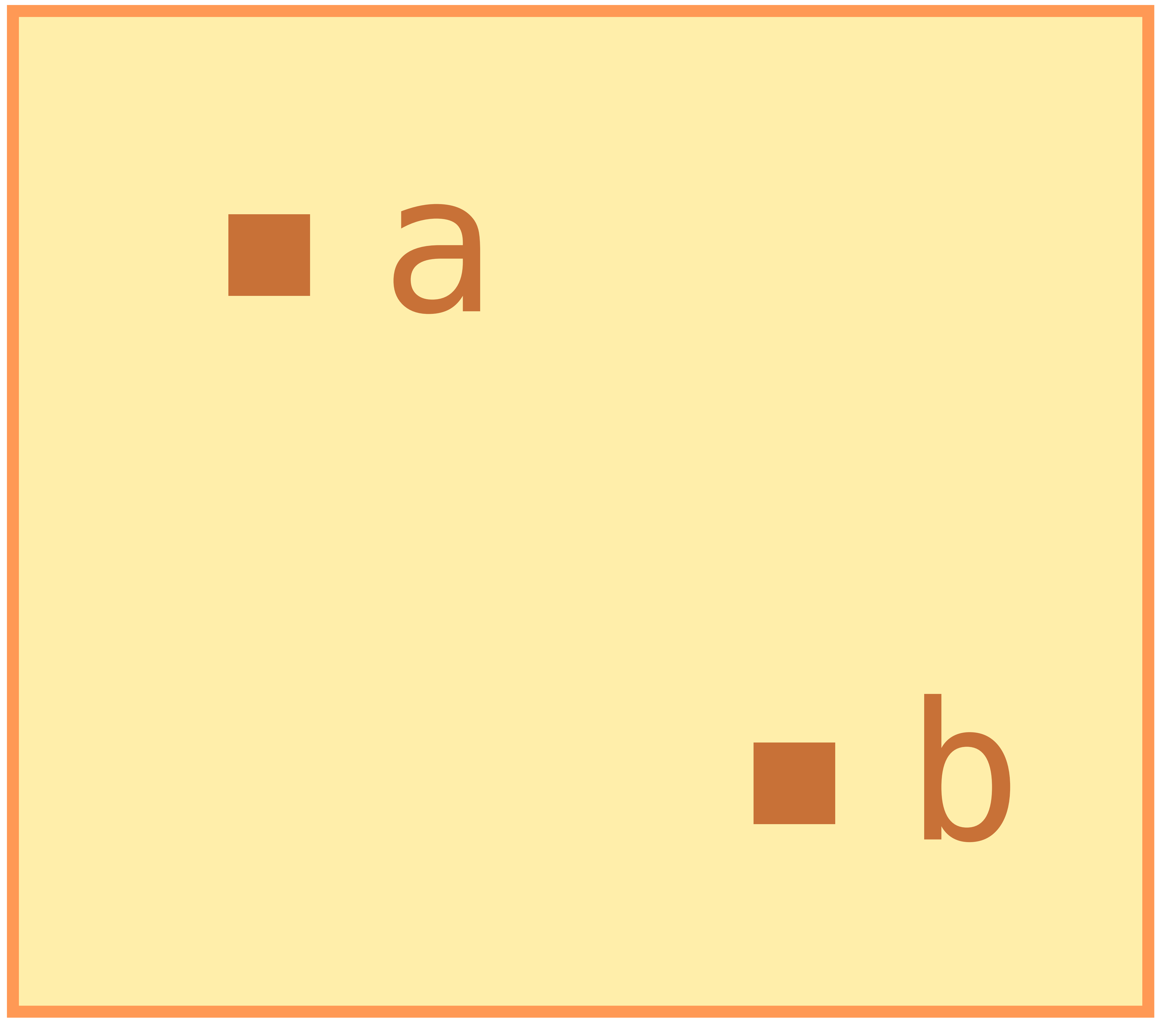
alternative paths in A and B

Theorem Statement

```
for any A, B, C, f and g,
fund.grp(pushout)
        ~= ?(??(A), ??(B), C)
```

??: paths between any two points

Fundamental Groupoids



(unique) ways to travel from a to b

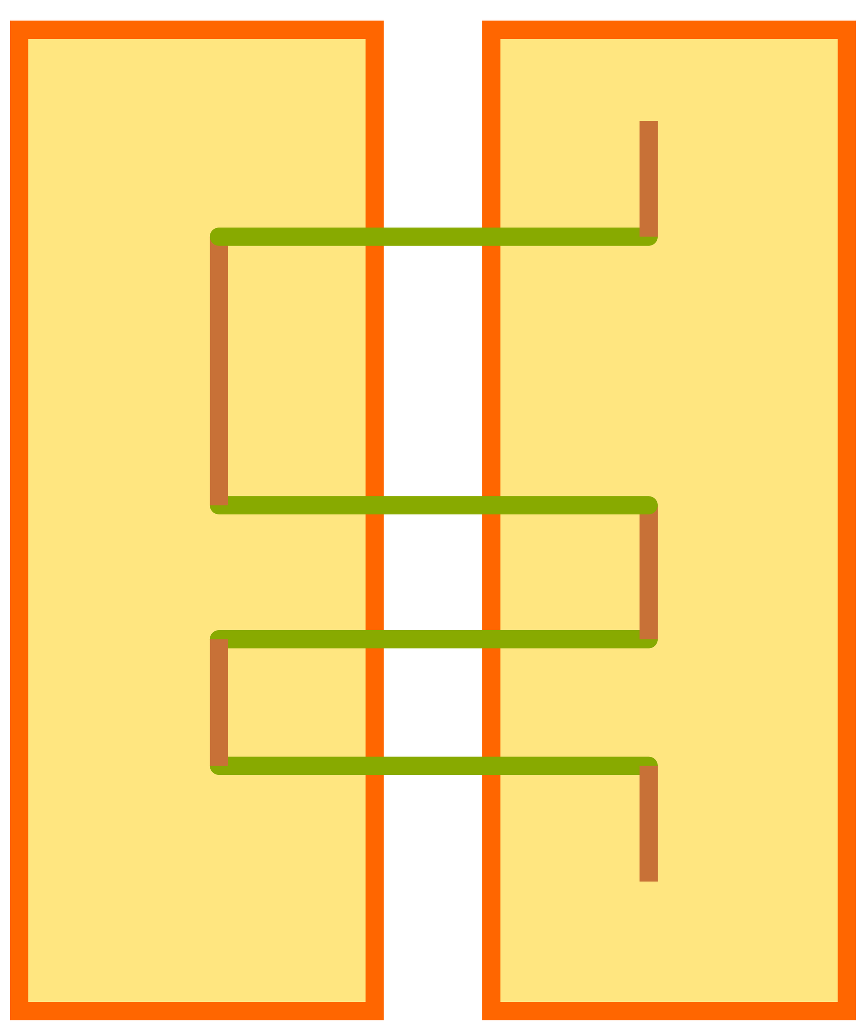
```
Trunc 0 (a == b)
```

Theorem Statement

```
for any A, B, C, f and g,
fund.groupoid(pushout)
      ~= ?(fund.groupoid(A),
           fund.groupoid(B), C)
```

?: "seqs of alternative elems"

Alternative Sequences



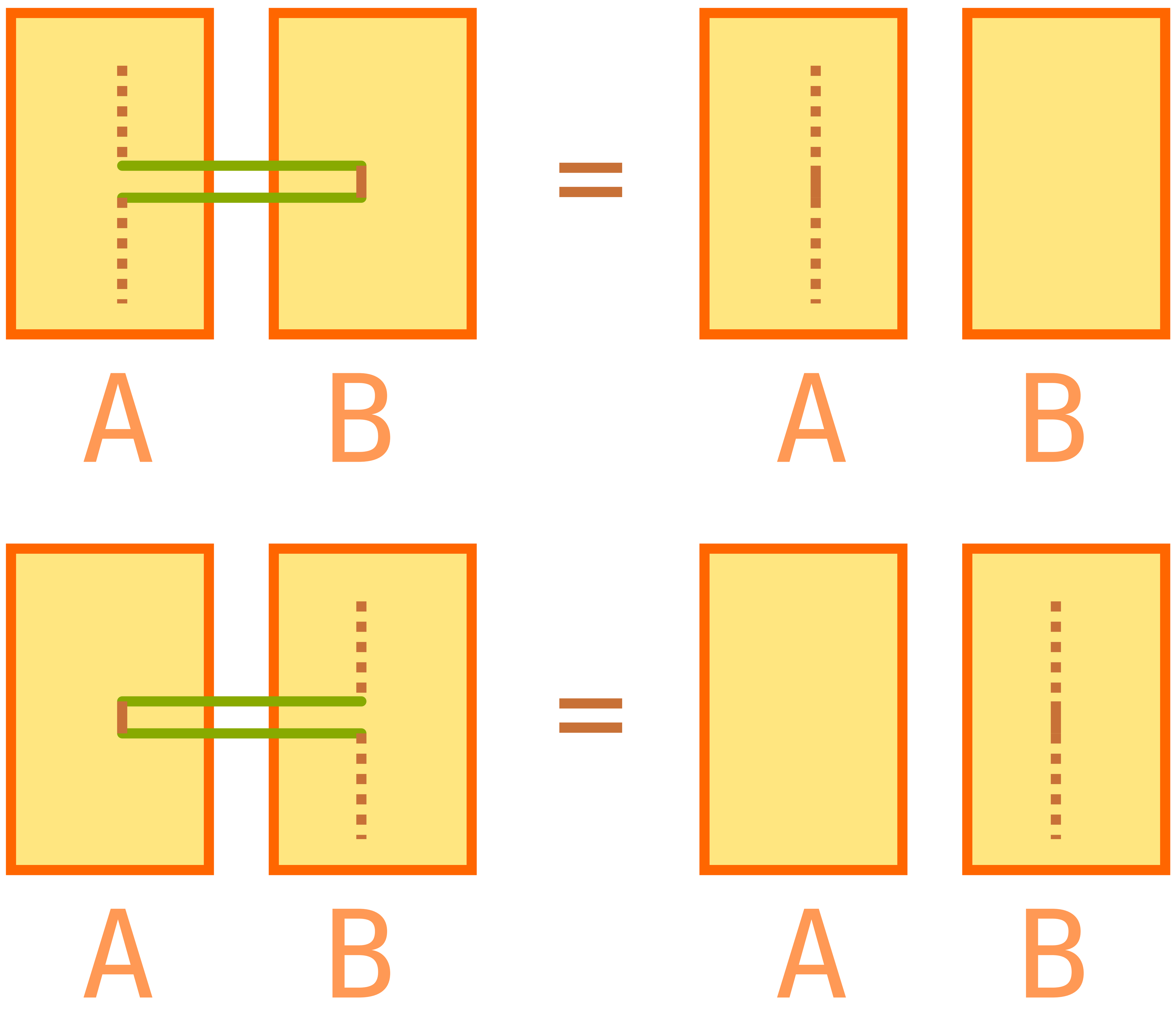
[p1, p2, ..., pn]

induction on both ends:

A to A, A to B,

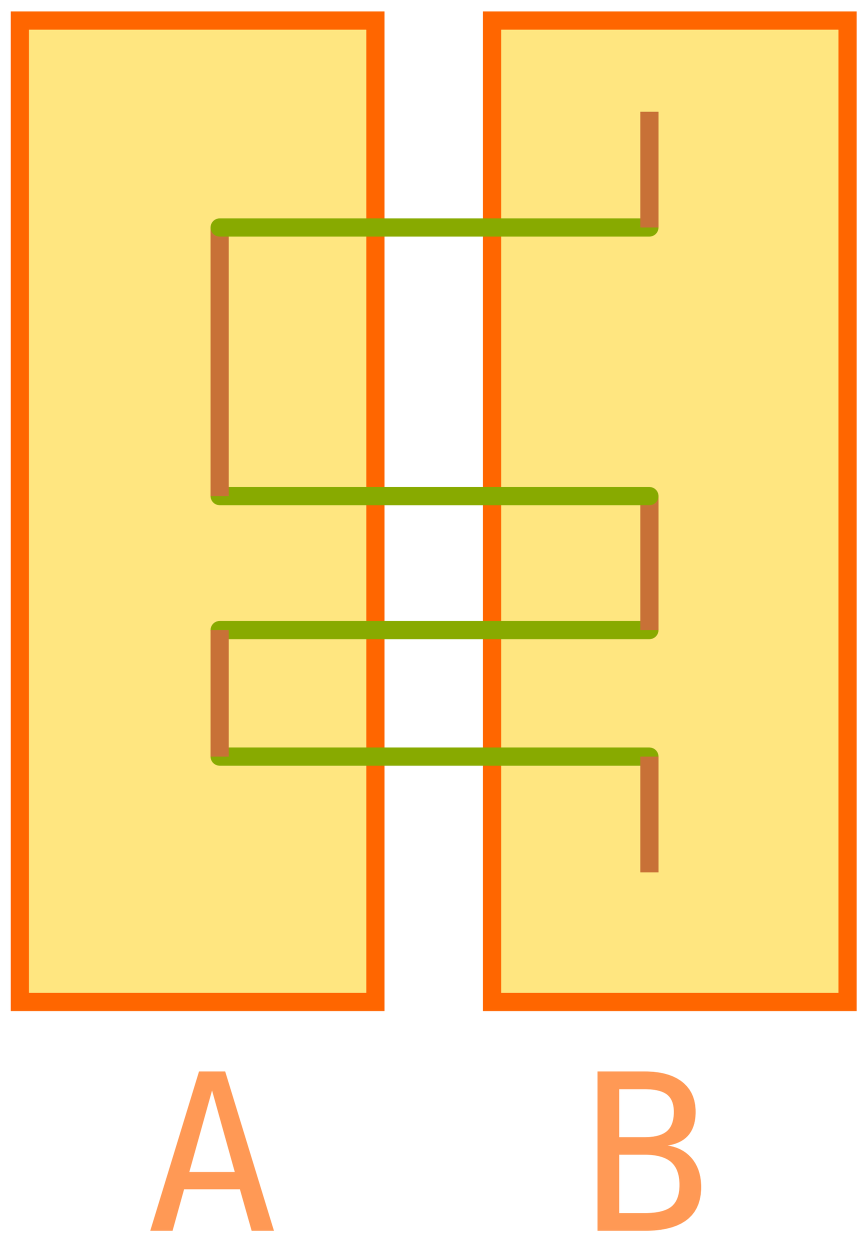
B to A, B to B

Alternative Sequences



quotients of alternative sequences by killing trivial identifications

Alternative Sequences



[p1, p2, ..., pn]

induction on both ends:

A to A, A to B,

B to A, B to B

each case is a quotient
of alternative sequences

Alternative Sequences

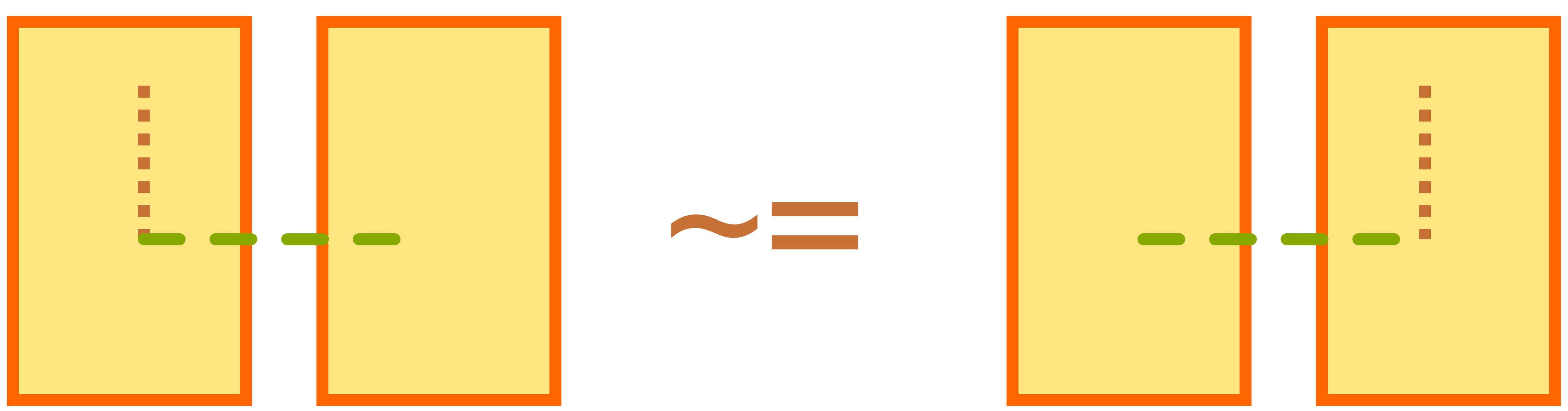
```
next: unify four cases into
      one type family alt.seq
```

Alternative Sequences

next: unify four cases into
one type family alt.seq

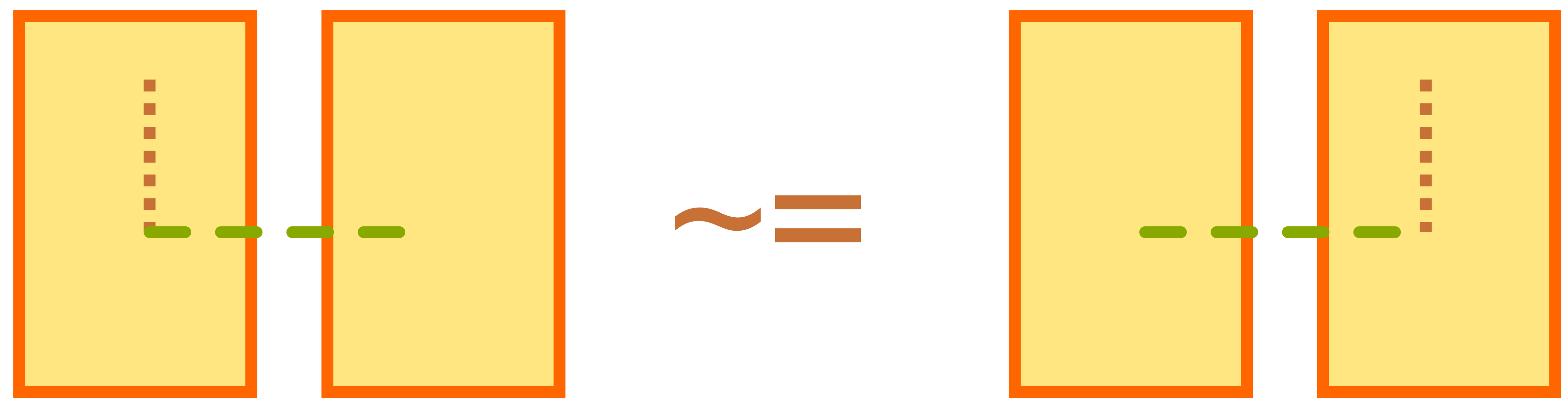
show respects for bridges by C. ex:

alt.seq a (f c) ~= alt.seq a (g c)

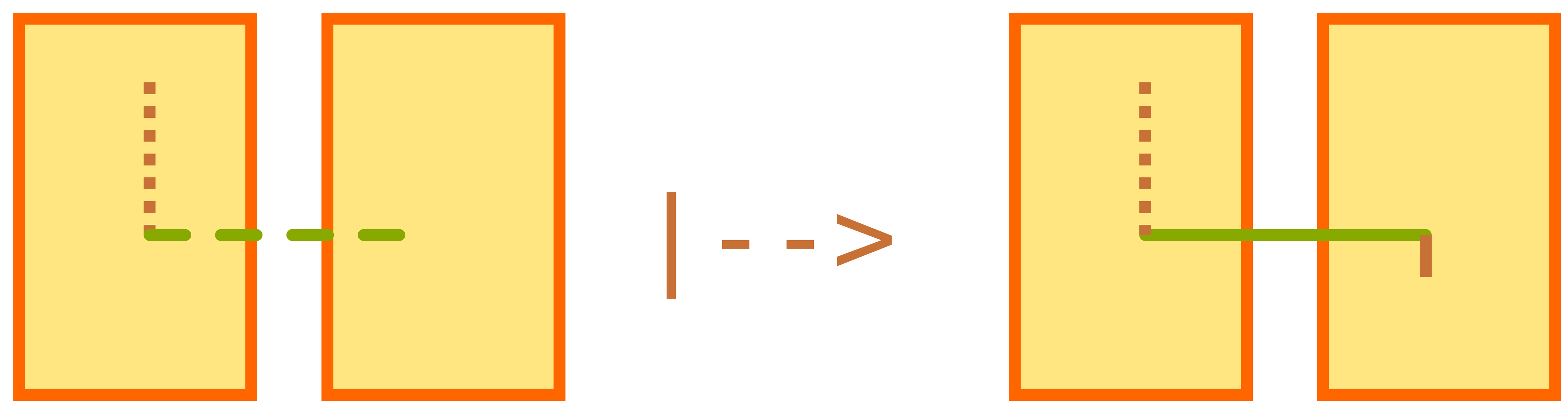


```
alt.seq a (f c) ~= alt.seq a (g c)
```

alt.seq a (f c) ~= alt.seq a (g c)

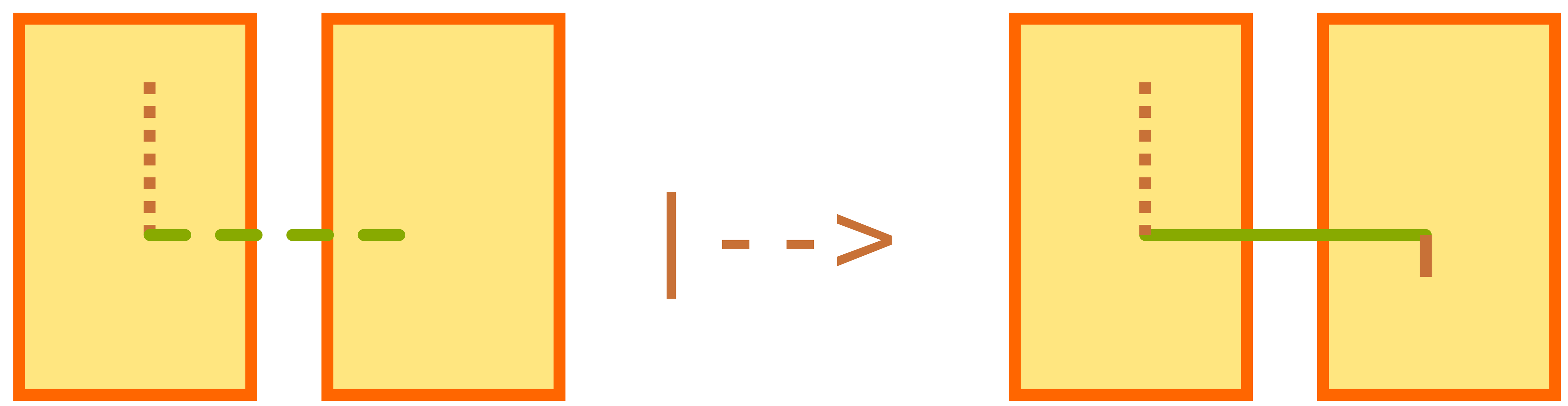


[..., p] |--> [..., p, trivial]



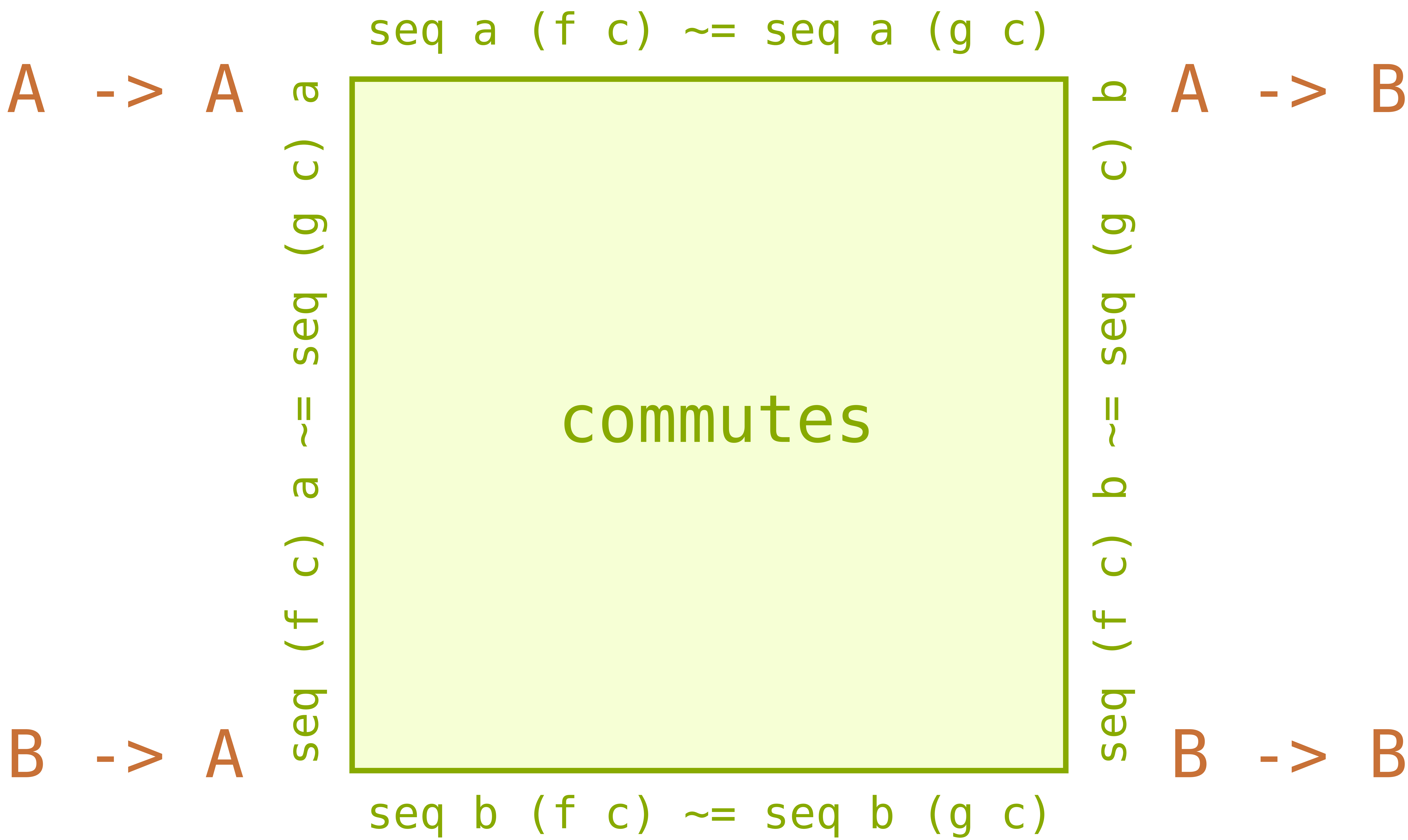
alt.seq a (f c) ~= alt.seq a (g c)

[..., p] |--> [..., p, trivial]



[..., p, trivial] <--| [..., p]

Alternative Sequences



Theorem

```
for any A, B, C, f and g,
fund.groupoid(pushout)
 ~= alt.seqs(fund.groupoid(A),
             fund.groupoid(B), C)
```

(zero pages left before the proofs)

Recipe of Equivalences

* two functions back and forth
  ("decode" and "encode")

* round-trips are identity

```
fund.groupoid -> alt.seqs
                encode
    (all paths)

 Trunc 0 (p == q) -> alt.seqs p q
```

```
fund.groupoid ->encode alt.seqs
    (all paths)

 Trunc 0 (p == q) -> alt.seqs p q

   path induction:
     consider only trivial paths

 (p : Pushout) -> alt.seqs p p
```

fund.groupoid ->encode alt.seqs

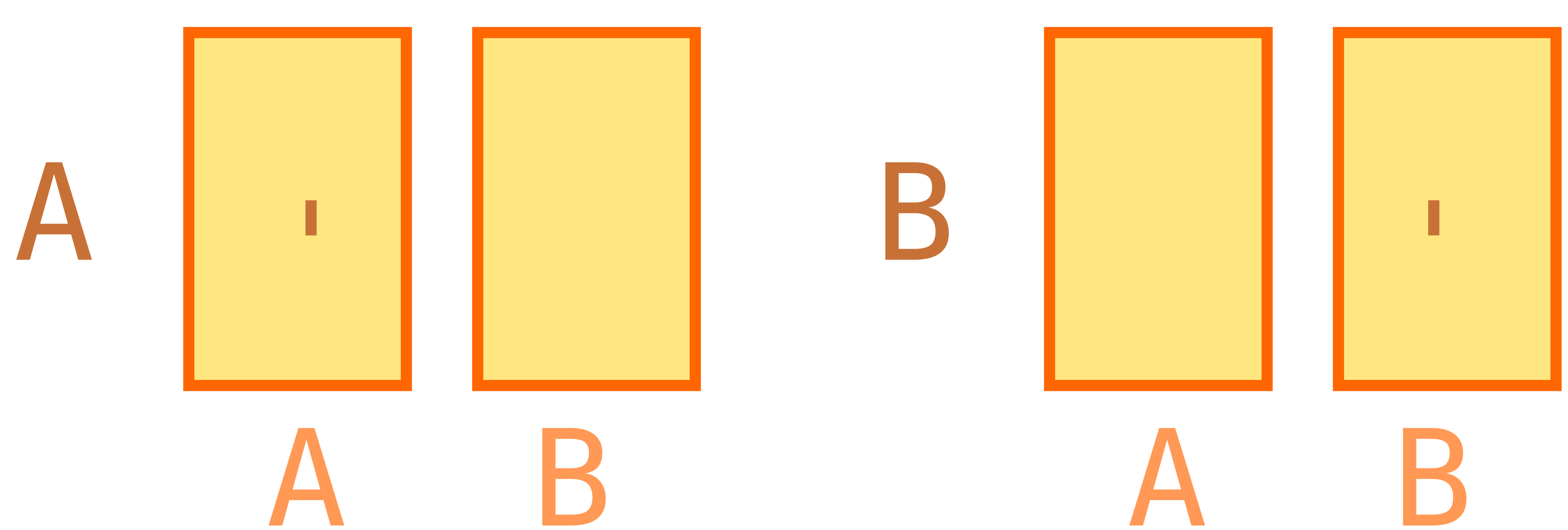
(all paths)

Trunc 0 (p == q) -> alt.seqs p q

path induction:
consider only trivial paths

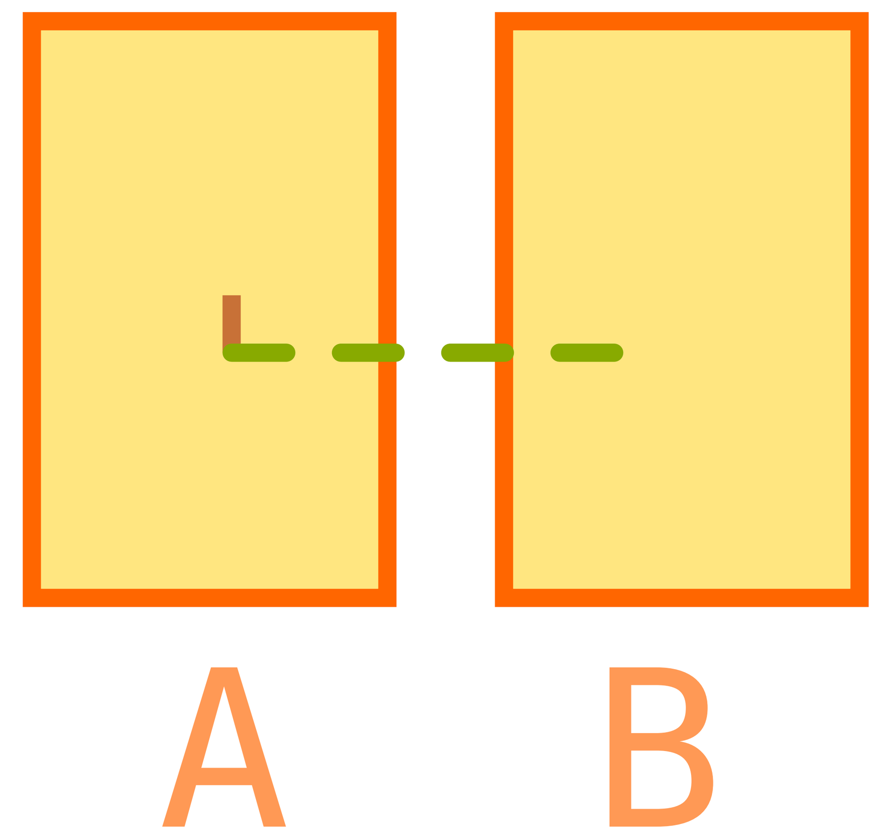
(p : Pushout) -> alt.seqs p p

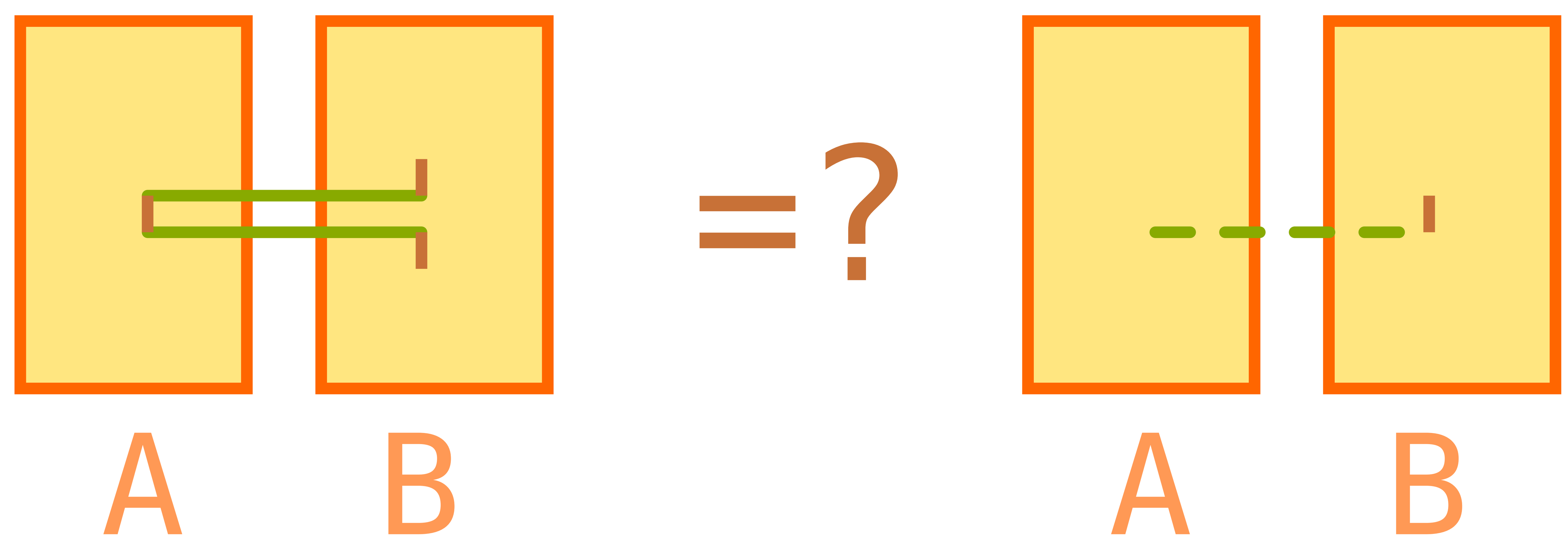
pushout induction



bridges by C
(next page)

case A

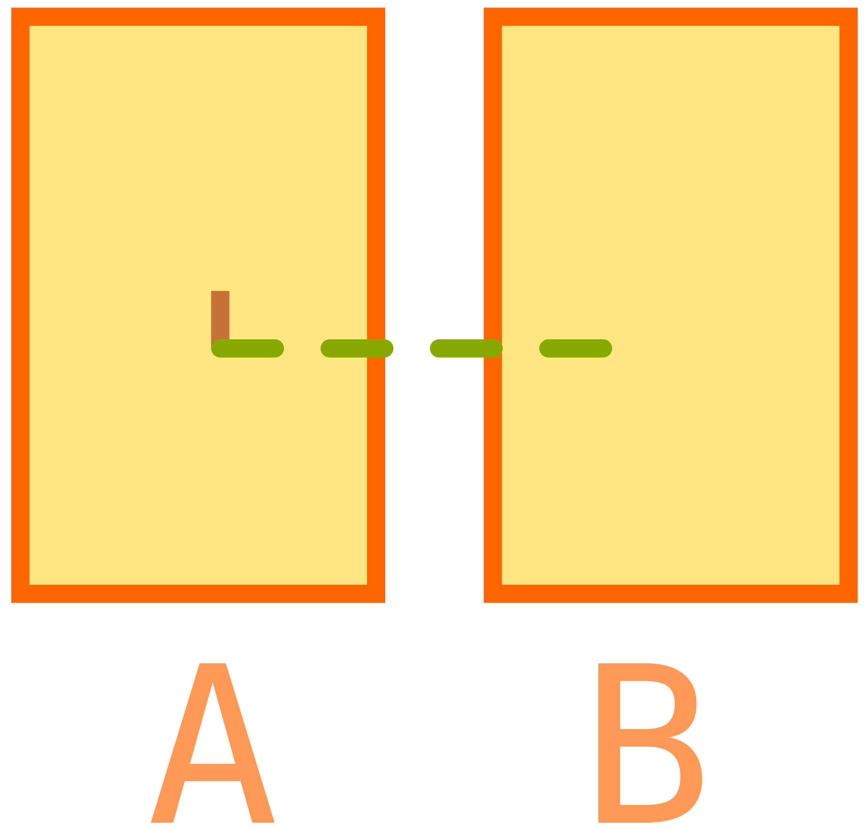


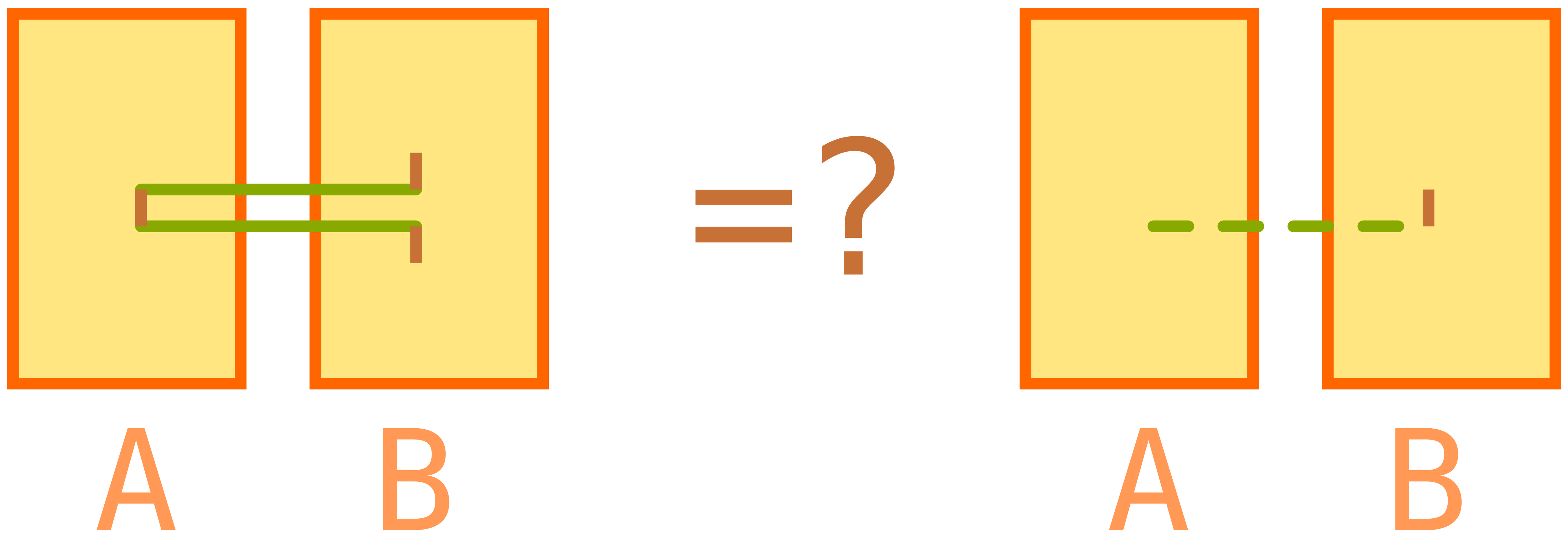


case B

applying the diagonal
in coherence square

case A





case B

applying the diagonal
in coherence square

witnessed by the quotient

alt.seq -> fund.groupoid

decode

just compositions!

$\text{alt.seq} \xrightarrow[\text{decode}]{} \text{fund.groupoid}$

just compositions!

$\text{grpd} \xrightarrow[\text{encode}]{} \text{seqs} \xrightarrow[\text{decode}]{} \text{grpd}$

again by path induction
(similar to "encode")

alt.seq $\to_{\text{decode}}$ fund.groupoid

just compositions!

grpd $\to_{\text{encode}}$ seqs $\to_{\text{decode}}$ grpd

again by path induction
(similar to "encode")

seqs $\to_{\text{decode}}$ grpd $\to_{\text{encode}}$ seqs

induction on sequences

```
lemma: encode(decode[p1,p2,...])
     = p1 :: encode(decode[p2,...])
```

Theorem

```
for any A, B, C, f and g,
fund.groupoid(pushout)
  = alt.seqs(fund.groupoid(A),
             fund.groupoid(B), C)
```

Final Notes

* Refined version: Can focus on just the set of base points of C covering its components.

* All mechanized in Agda
  github.com/HoTT/HoTT-Agda/blob/1.0/Homotopy/VanKampen.agda

* Submitted to CSL 2016
  www.cs.cmu.edu/~kuenbanh/files/vankampen.pdf