# GopCaml:
## A Structural Editor for OCaml

Kiran Gopinathan

National University of Singapore

https://gitlab.com/gopiandcode/gopcaml-mode

Let’s say you’re writing a function...

# Let’s say you’re writing a function...

```
let map f = function
| [] -> []
| h :: t -> f h :: t
```

Let’s say you’re writing a function...

```
let map f = function
| [] -> []
| h :: t -> f h :: t
```

and need to change the definition...

Let’s say you’re writing a function...

```
let map f = function
| [] -> []
| h :: t -> f h :: t
```

and need to change the definition...

Let’s say you’re writing a function...

```
let rec map f = function
| [] -> []
| h :: t -> f h :: t
```

and need to change the definition...

Let’s say you’re writing a function...

How can we provide

**editor-support** for this operation?

and need to change the definition...

# OCaml Editor Support

# OCaml Editor Support

Emacs' `beginning-of-defun` (`C-M-a`)

# OCaml Editor Support

Emacs' `beginning-of-defun` (`C-M-a`)

*...but how should it be implemented?*

# OCaml Editor Support

*...but how should it be implemented?*

```
let f x = ...
```

# OCaml Editor Support

*...but how should it be implemented?*

```
let f x =

   let ... = ... in

   ...
```

# OCaml Editor Support

*...but how should it be implemented?*

```
let f x =
   let module ... = struct

      ...

   end in
   ...
```

# OCaml Editor Support

*...but how should it be implemented?*

```
let f x =
   let module ... = struct

      let ... = ...

   end in
   ...
```

# OCaml Editor Support

Not as simple as it seems...

# OCaml Editor Support

Not as simple as it seems...

...what denotes an expression?

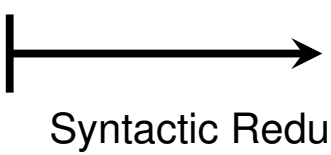
# *A **fundamental** dichotomy*

*A **fundamental** dichotomy*

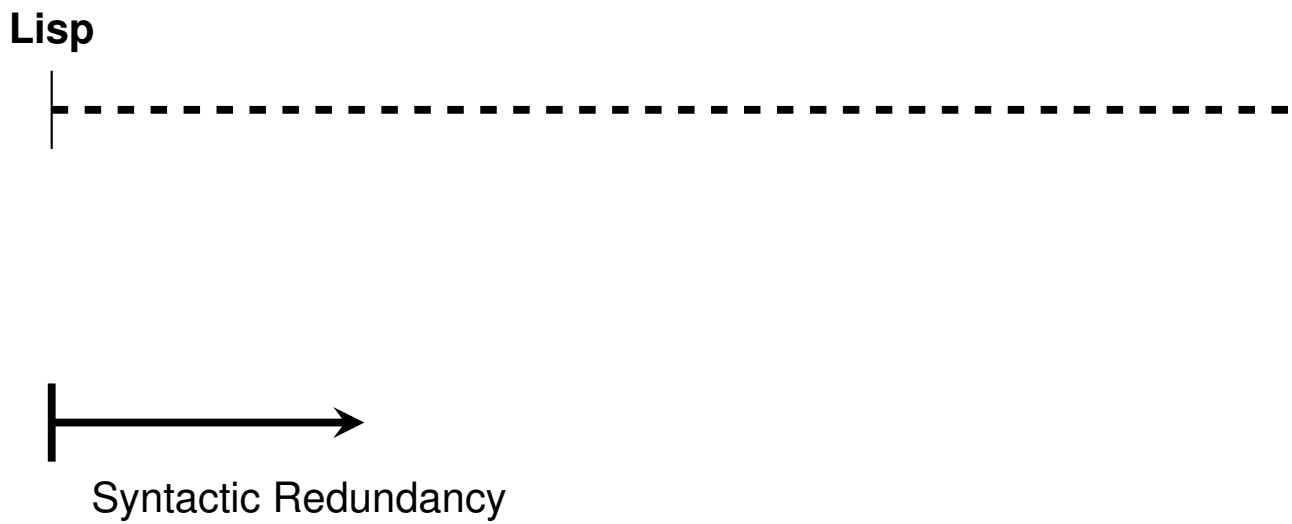
---

# *A **fundamental** dichotomy*

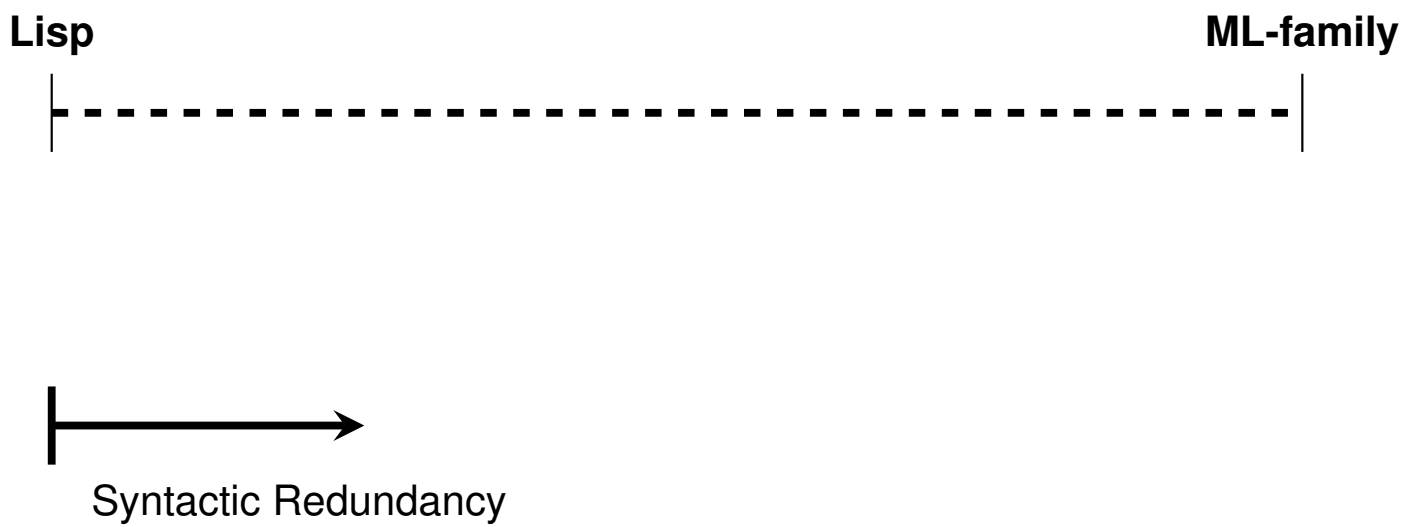
# *A **fundamental** dichotomy*



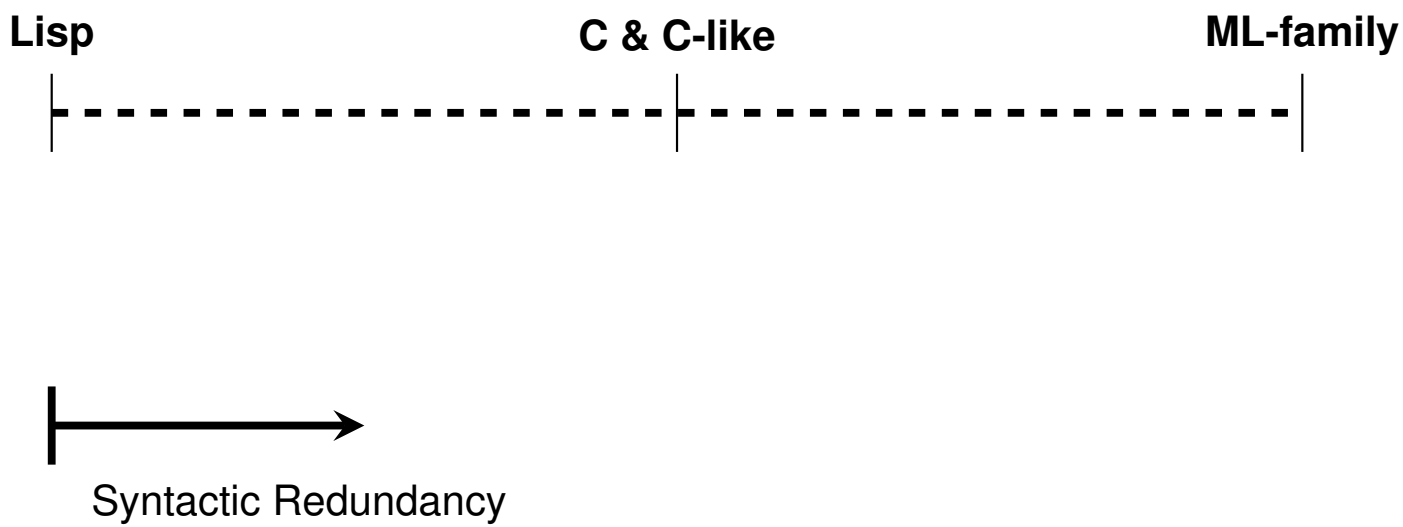
# *A **fundamental** dichotomy*



# *A **fundamental** dichotomy*

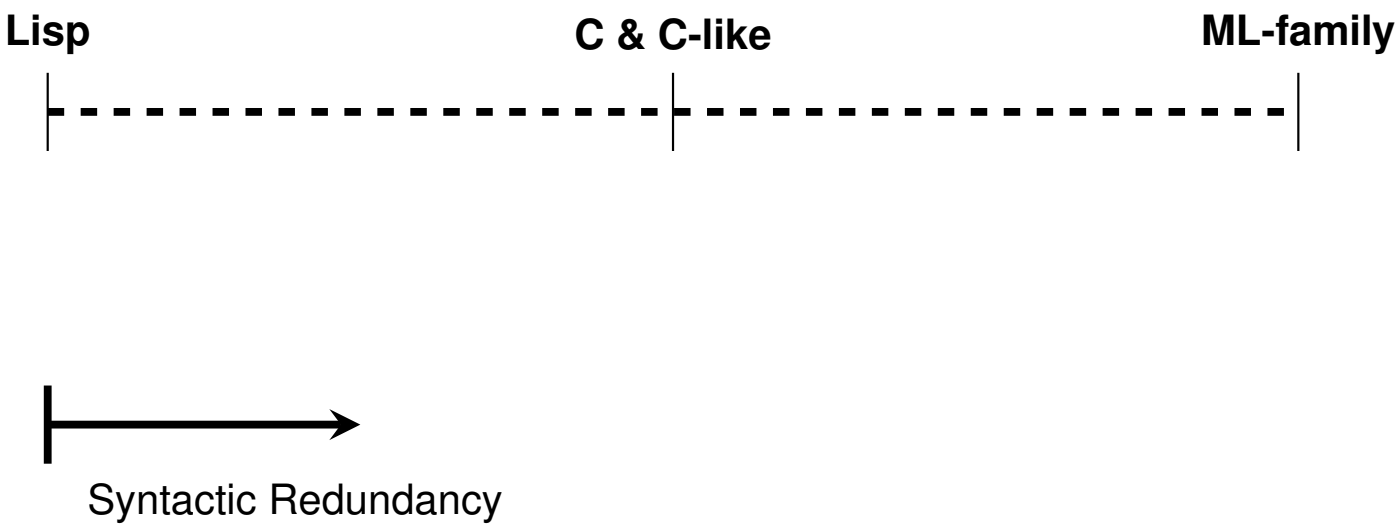


# *A **fundamental** dichotomy*



# *A **fundamental** dichotomy*

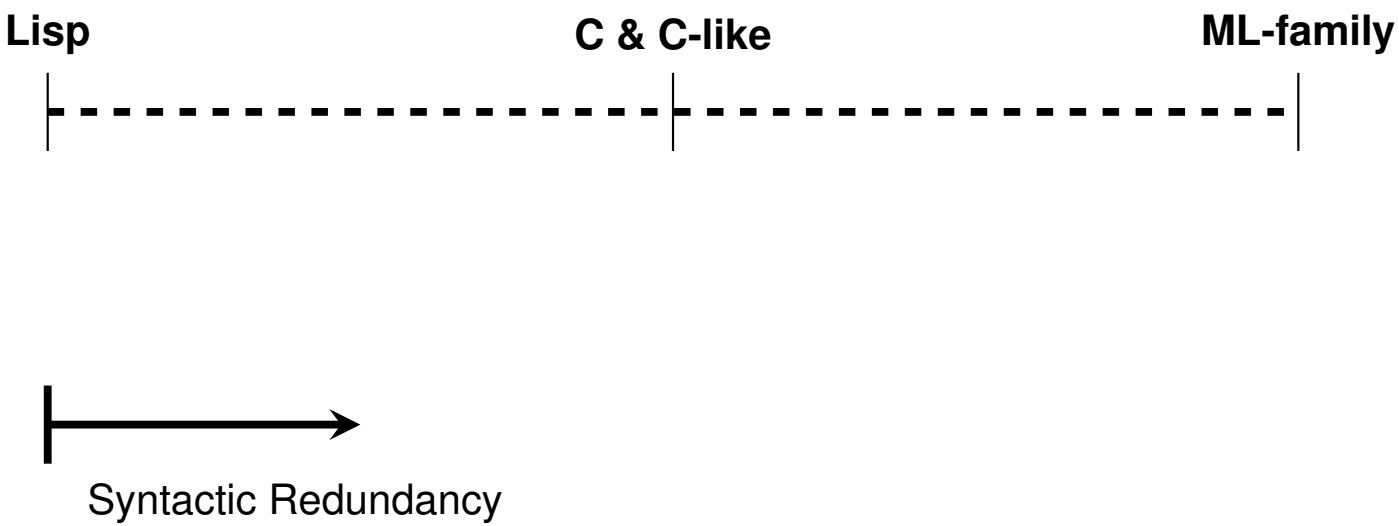
+ **Robust** editor support



# *A **fundamental** dichotomy*

+ **Robust** editor support

- Syntactically **Noisy**

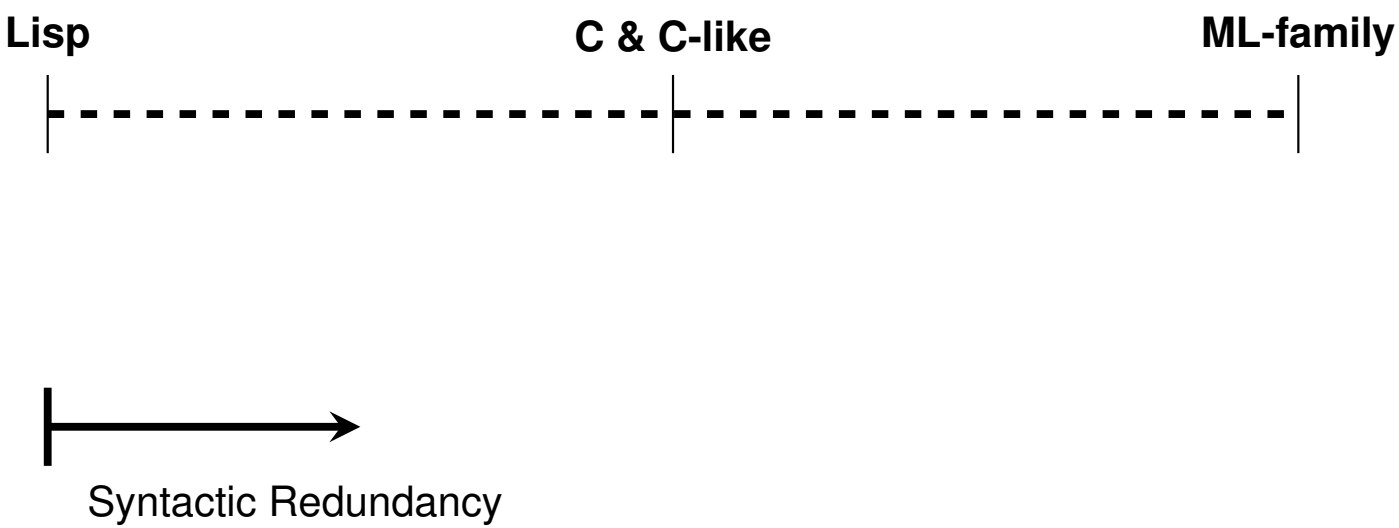


# *A **fundamental** dichotomy*

+ **Robust** editor support

+ **Clean** and **Concise** syntax

- Syntactically **Noisy**



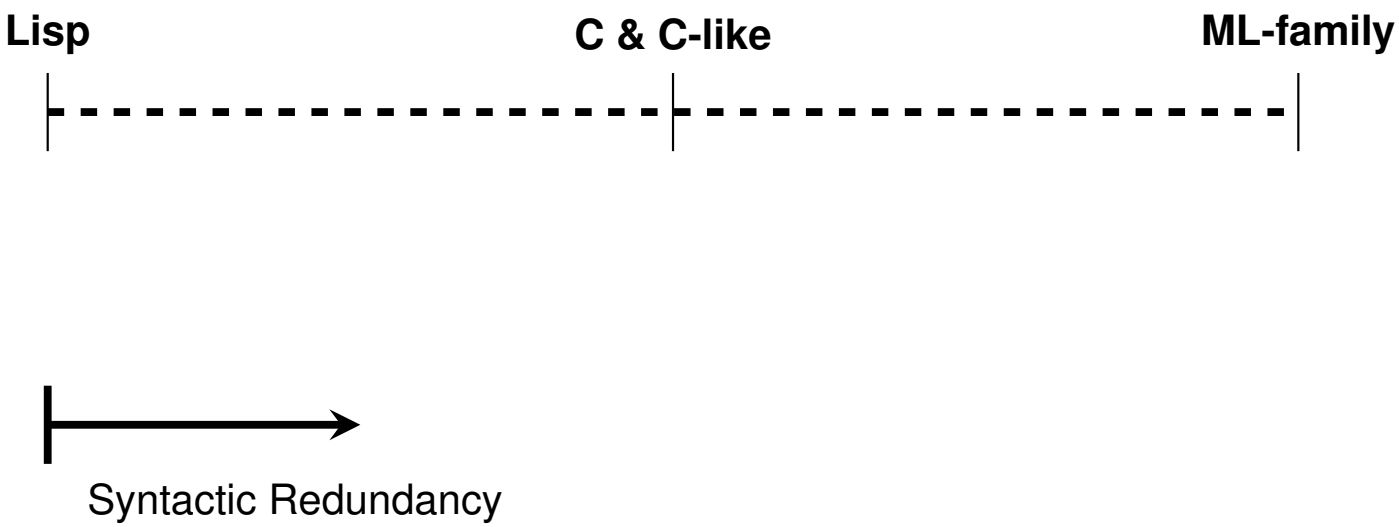
# *A **fundamental** dichotomy*

+ **Robust** editor support

- Syntactically **Noisy**

+ **Clean** and **Concise** syntax

- **Ad-hoc** editor support



# A ***fundamental*** *dichotomy*

How can we get the

**best of both worlds**?

Syntactic Redundancy

# *A **fundamental** dichotomy*

Track the **syntax tree**

from the editor!

Syntactic Redundancy

# Contributions

- **GopCaml**: Generic Framework for Structural Editing
  - Leverages OCaml compiler pipeline for **faithful** parsing
  - Tracks Concrete-Syntax-Tree (CST) of edited file
  - Defines *common* editing operations as CST transformations
- **GopCaml-mode**: Emacs plugin using Gopcaml
  - **Robust** and **consistent** OCaml support
  - **Seamless** integration with Emacs workflows

# Overview

1. A tour of GopCaml
2. **Live** demo
3. Under the hood
4. Future work

# A tour of GopCaml

- Move to definition
- Structural navigation
- Structural transposition
- Structural deletion
- Extract expression

# A tour of GopCaml

Move to definition ( `C-M-a` )

# A tour of GopCaml

Structural navigation (`C-M-{f,b}`, `C-M-{u,d}`)

# A tour of GopCaml

Structural transposition ( `C-S-M-{f,b,u,d}` , `C-M-t` )

# A tour of GopCaml

Structural Deletion ( `C-M-d` , `C-M-w` )

# A tour of GopCaml

Extract expression ( `C-c C-e` )

# “Live” Demo!

*Talk is cheap*... **Show us some code**!

# Under the hood

How does it work?

# Under the hood

How does it work?

Tracking the CST

# Under the hood

A small problem...

# Under the hood

OCaml AST

# Under the hood

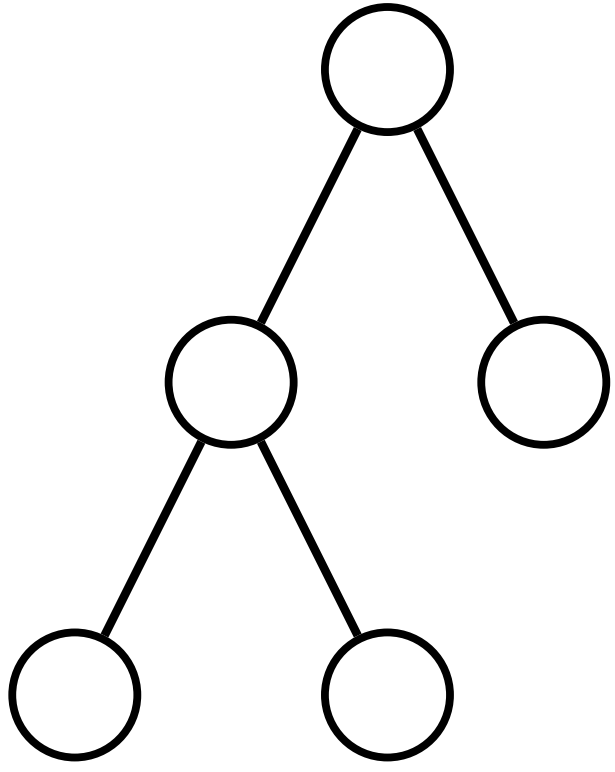
## OCaml AST

```
type expression = {
        pexp_desc: ...;
        ...
  }
and expression_desc =
  | Pexp_ident of ...
  | Pexp_let of ...
  | Pexp_function of ...
  | Pexp_fun of ...
  | Pexp_apply of ...
    ...
```

# Under the hood

## OCaml AST

```
type expression = {
        pexp_desc: ...;
        ...
  }
and expression_desc =
  | Pexp_ident of ...
  | Pexp_let of ...
  | Pexp_function of ...
  | Pexp_fun of ...
  | Pexp_apply of ...
    ...
```

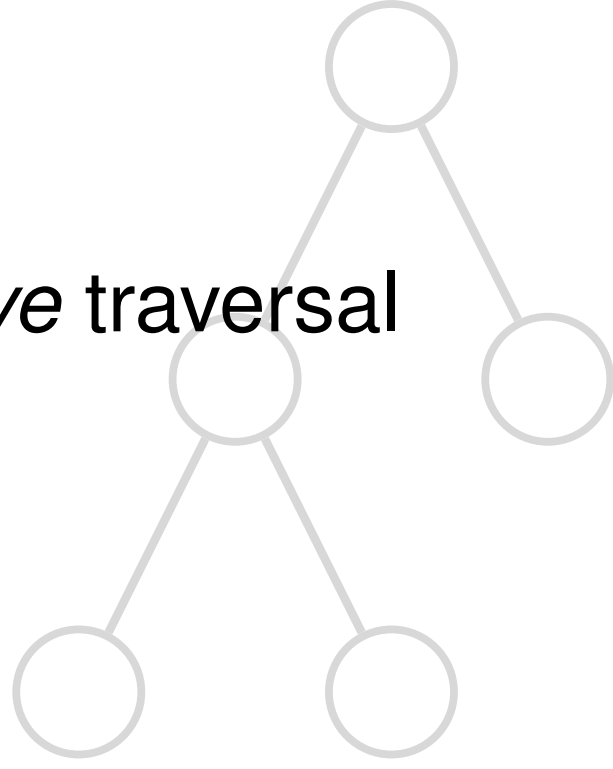


# Under the hood

OCaml AST

```
type expression = {
        pexp_desc: ...;
        ...
  }
and expression_desc =
  | Pexp_ident of ...
  | Pexp_let of ...
  | Pexp_function of ...
  | Pexp_fun of ...
  | Pexp_apply of ...
    ...
```

**Not suited** for *interactive* traversal



# Under the hood

**Not suited** for *interactive* traversal

# Under the hood

**Not suited** for *interactive* traversal

# Under the hood

**Not suited** for *interactive* traversal

**Solution**: Huet's Zipper

# Under the hood

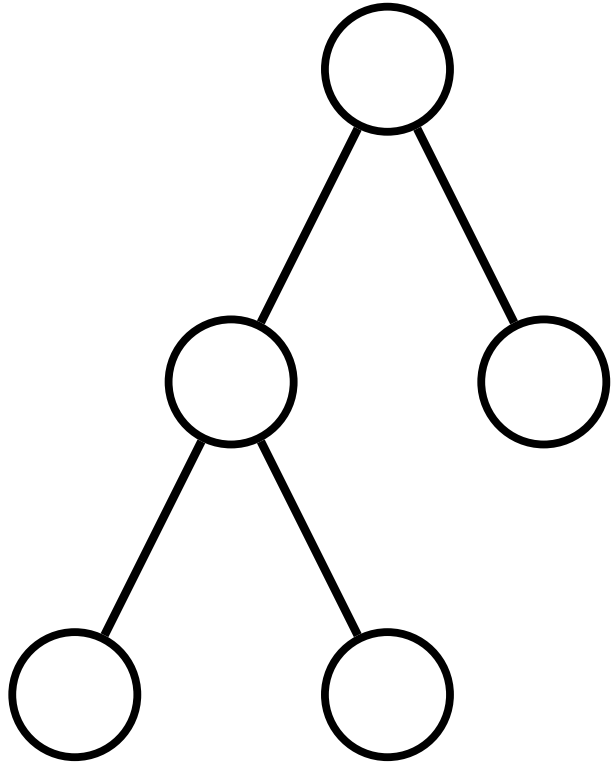
A zipper for **editing**

```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```

# Under the hood

A zipper for **editing**

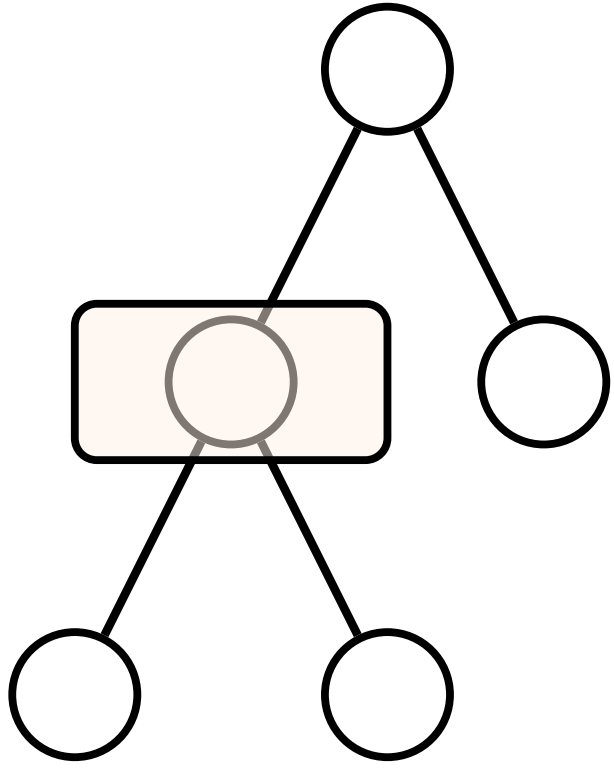
```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

A zipper for **editing**

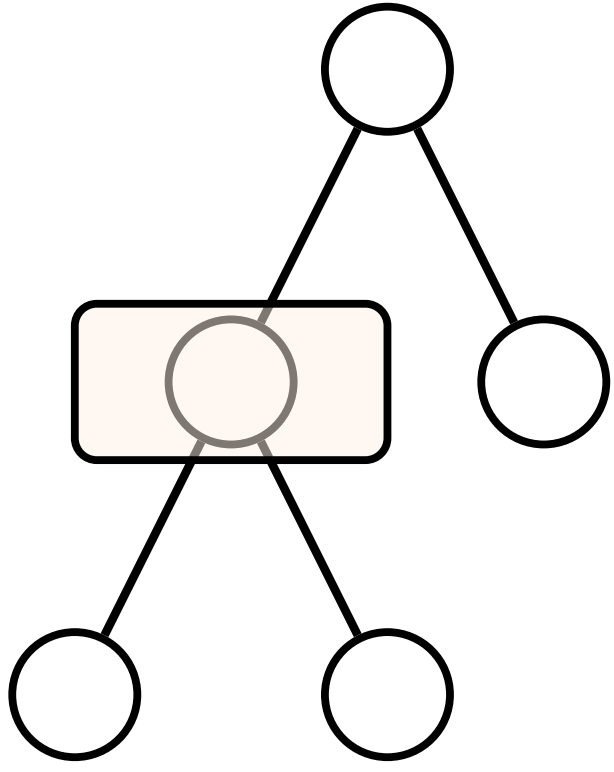
```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

A zipper for **editing**

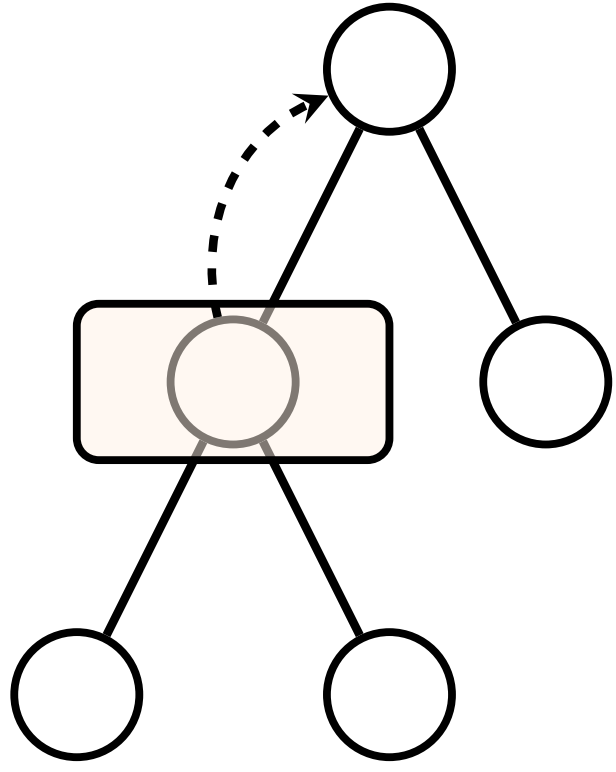
```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

## A zipper for **editing**

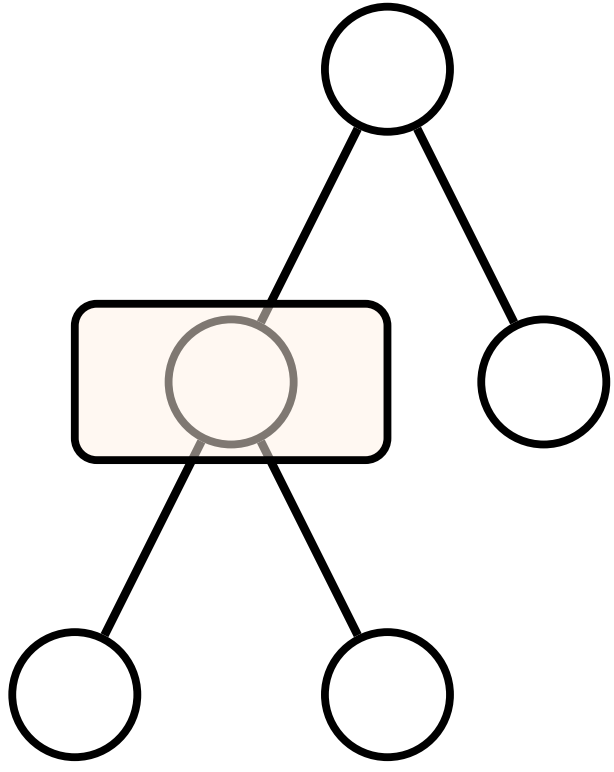
```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

A zipper for **editing**

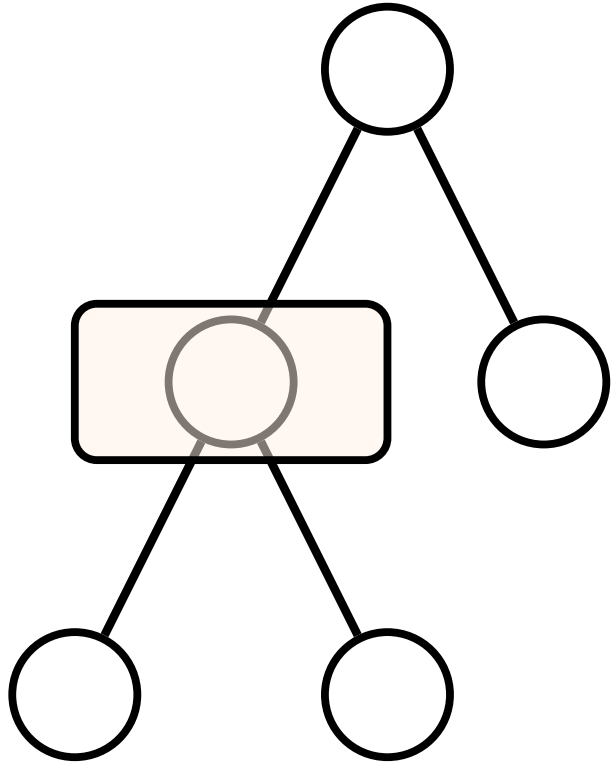
```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

A zipper for **editing**

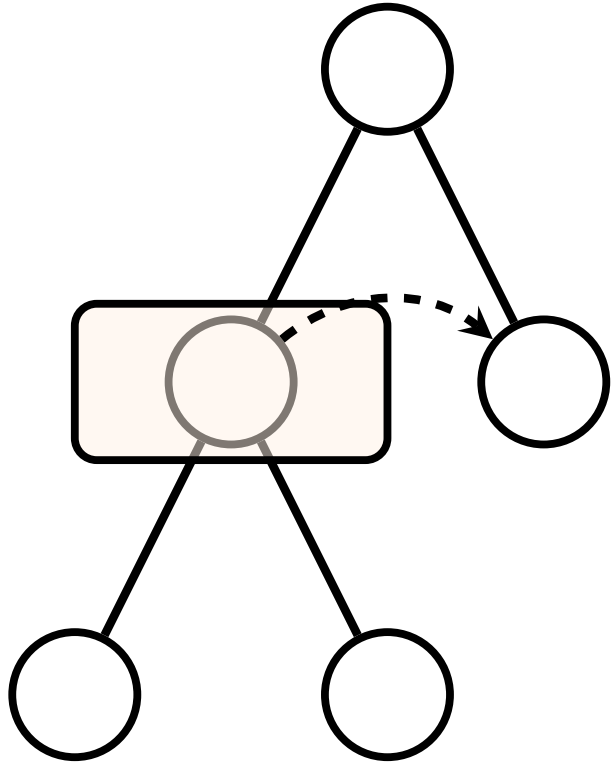
```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

A zipper for **editing**

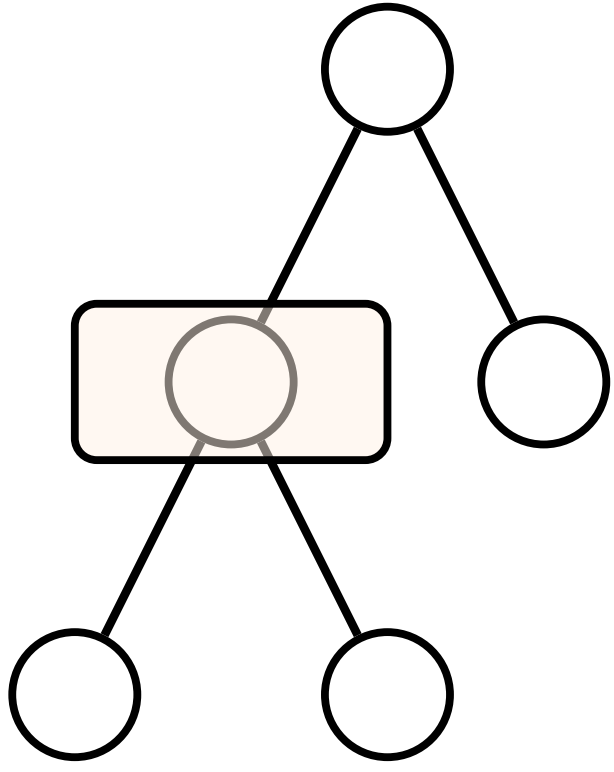
```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

A zipper for **editing**

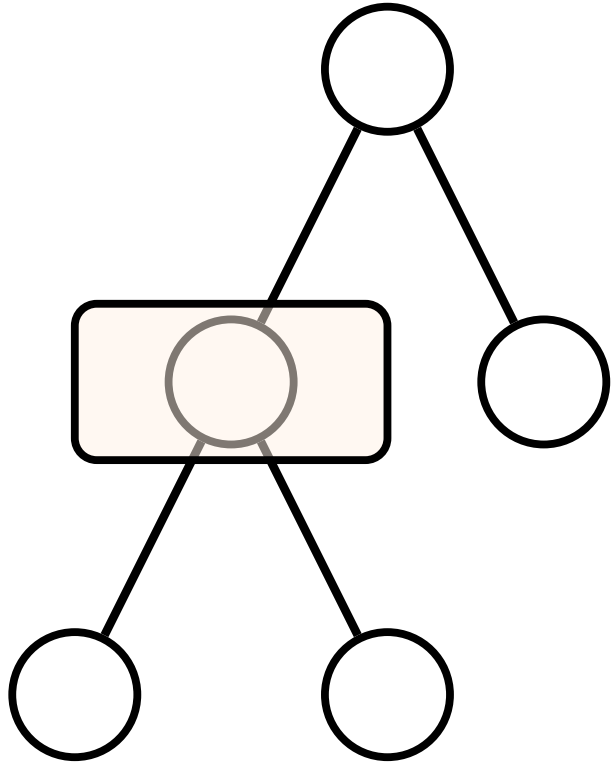
```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

A zipper for **editing**

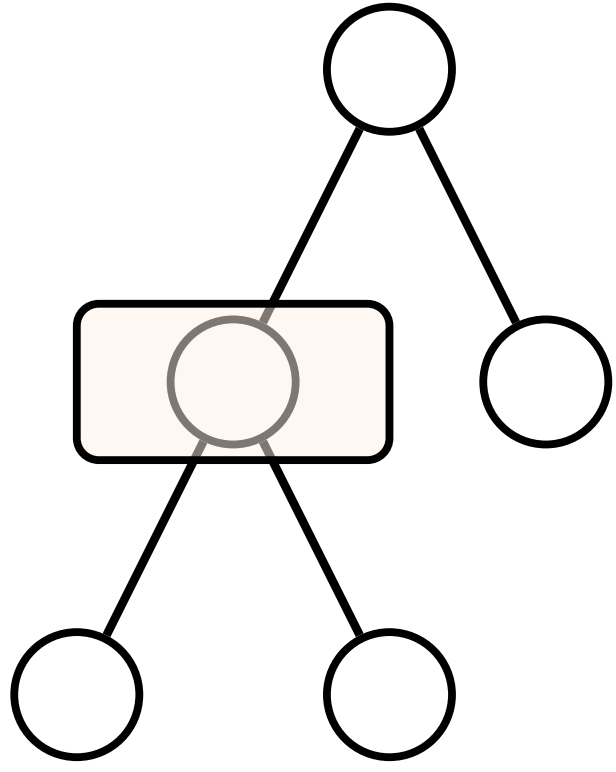
```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

A zipper for **editing**

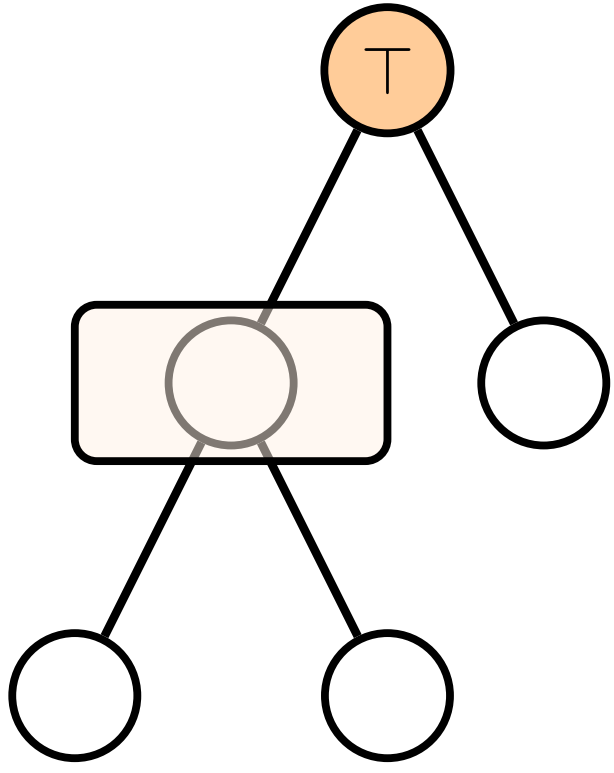
```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

A zipper for **editing**

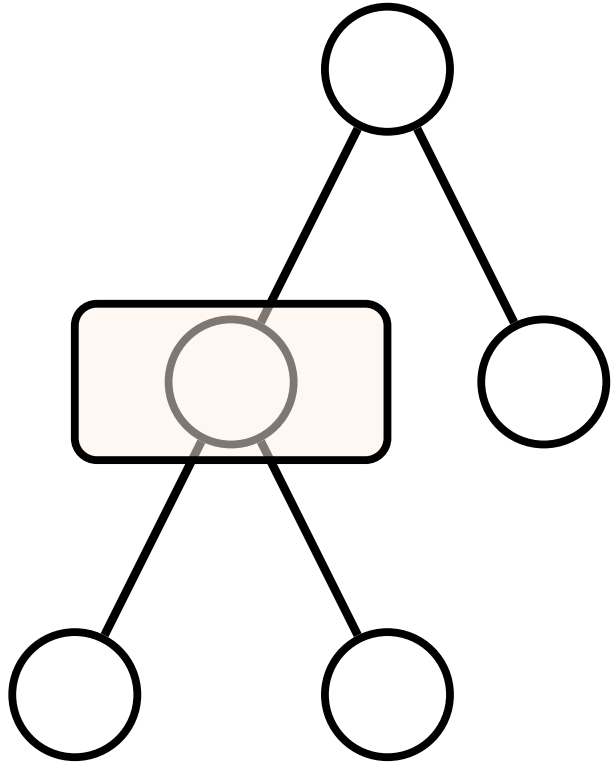
```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

A zipper for **editing**

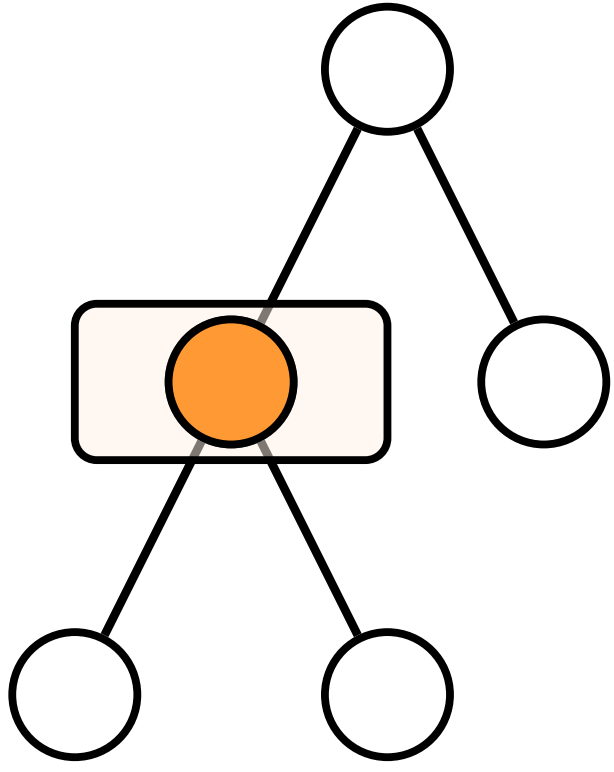
```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

A zipper for **editing**

```
type zipper =
  | Top
  | Node of {
      item: t;
      below: t list;
      above: t list;
      parent: zipper;
      bounds: text_region;
    }
```



# Under the hood

A *lazy* zipper for **editing**

```
type t =
  | Sequence of
      text_region option * t list * t * t list
  | Signature_item of Parsetree.signature_item
  | Structure_item of Parsetree.structure_item
  | Value_binding of Parsetree.value_binding
  (* ... *)
```

# Under the hood

A *lazy* zipper for **editing**

```
type t =
  | Sequence of
      text_region option * t list * t * t list
  | Signature_item of Parsetree.signature_item
  | Structure_item of Parsetree.structure_item
  | Value_binding of Parsetree.value_binding
  (* ... *)
```

# Under the hood

A *lazy* zipper for **editing**

```
type t =
  | Sequence of
      text_region option * t list * t * t list
  | Signature_item of Parsetree.signature_item
  | Structure_item of Parsetree.structure_item
  | Value_binding of Parsetree.value_binding
  (* ... *)
```

# Under the hood

A *lazy* zipper for **editing**

```
type t =
  | Sequence of
      text_region option * t list * t * t list
  | Signature_item of Parsetree.signature_item
  | Structure_item of Parsetree.structure_item
  | Value_binding of Parsetree.value_binding
  (* ... *)
```

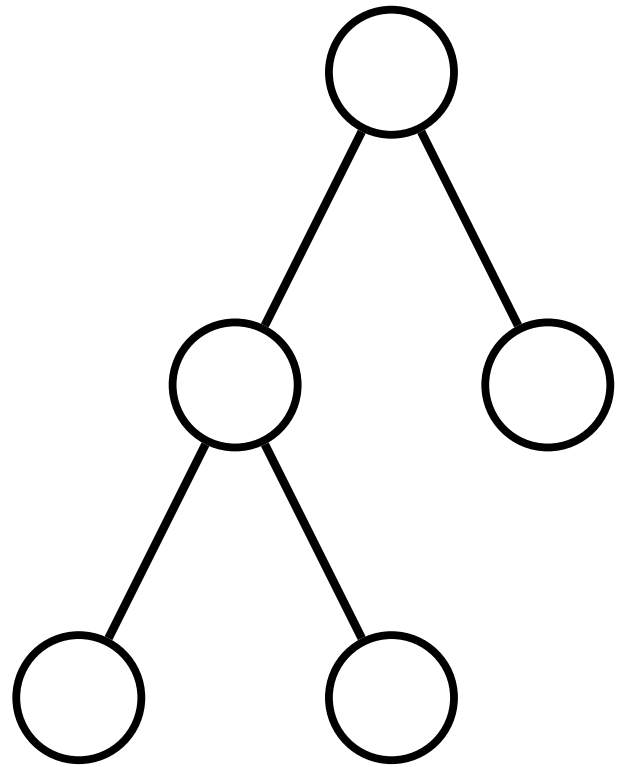
# How does it work?

# Under the hood

- Structural navigation
- Structural transposition
- Structural deletion

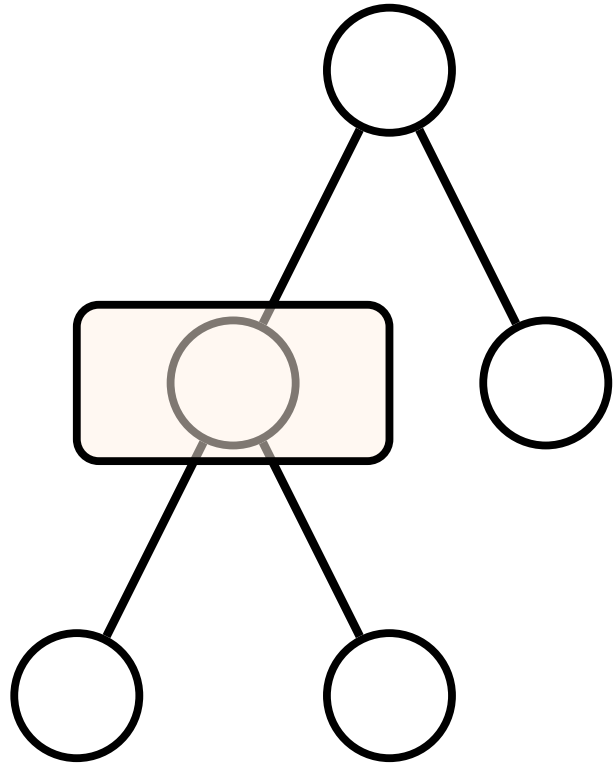
# Under the hood

Structural navigation



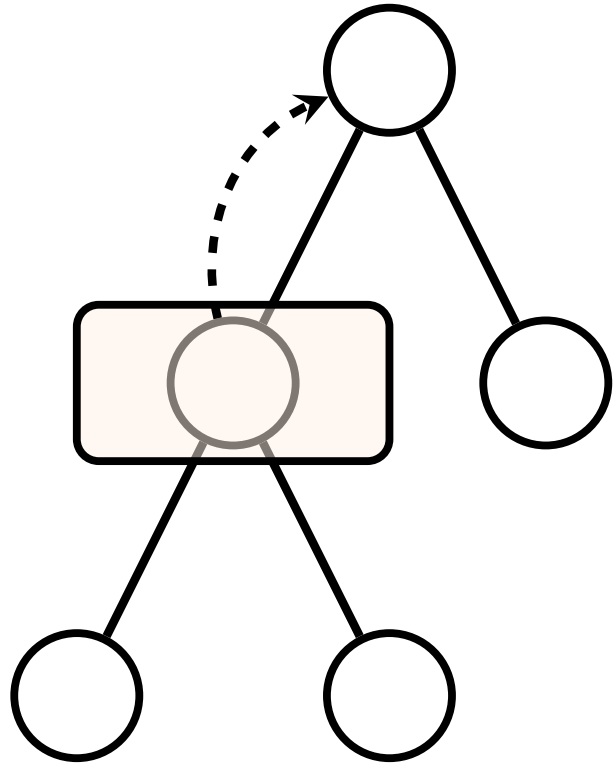
# Under the hood

Structural navigation



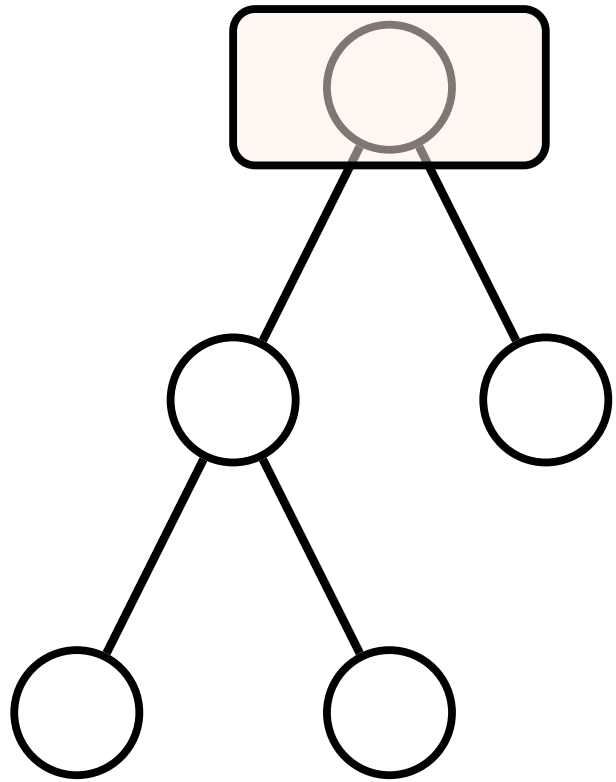
# Under the hood

Structural navigation



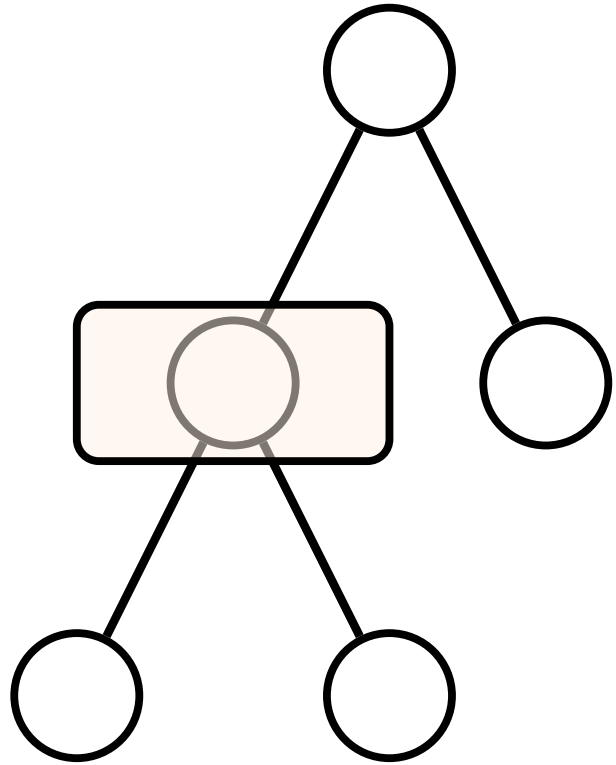
# Under the hood

Structural navigation



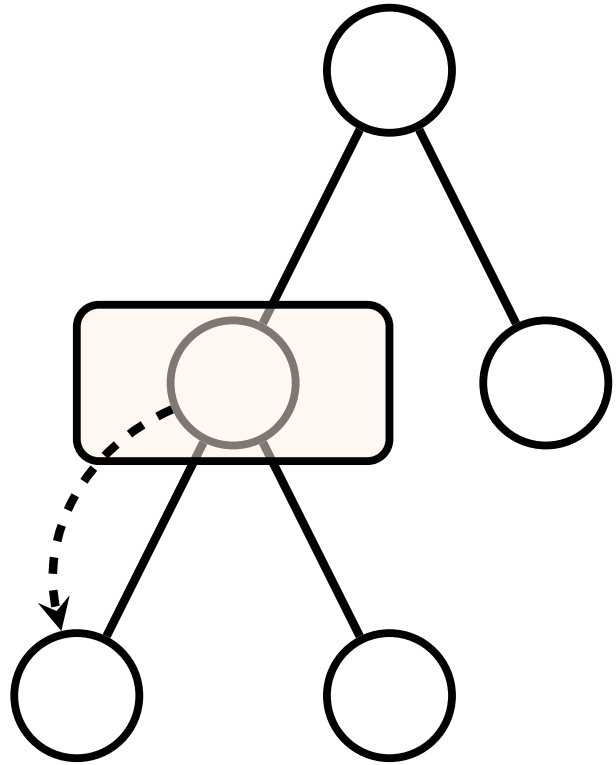
# Under the hood

Structural navigation



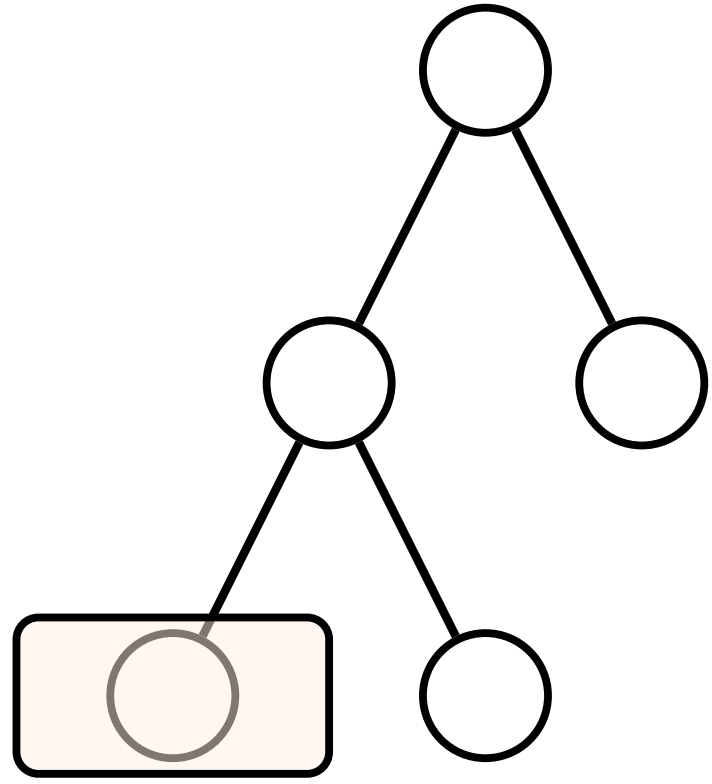
# Under the hood

Structural navigation



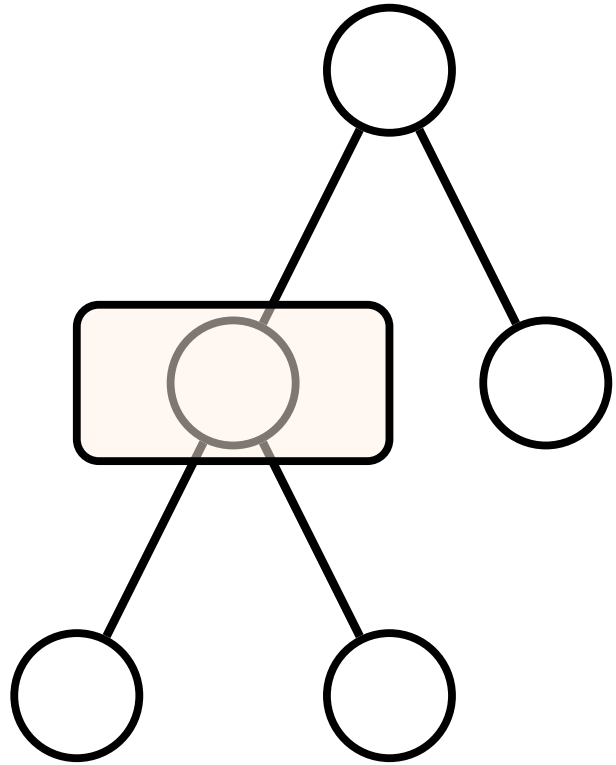
# Under the hood

Structural navigation



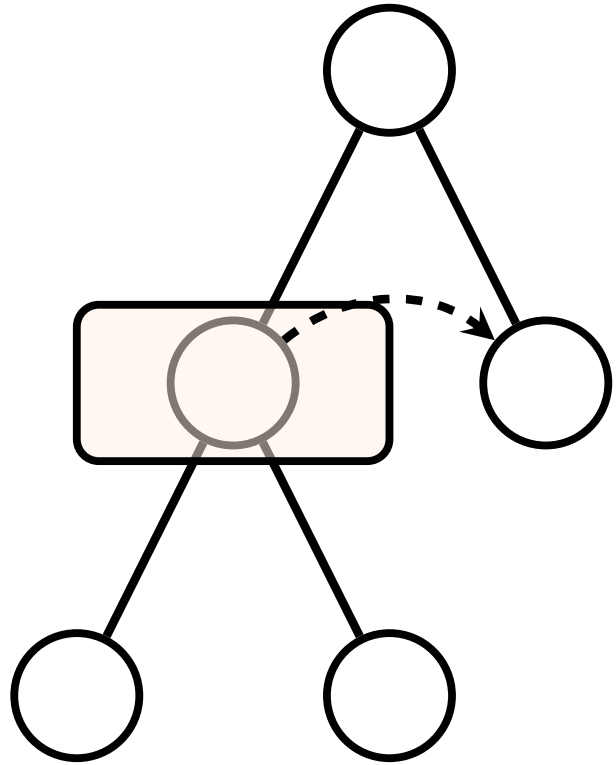
# Under the hood

Structural navigation



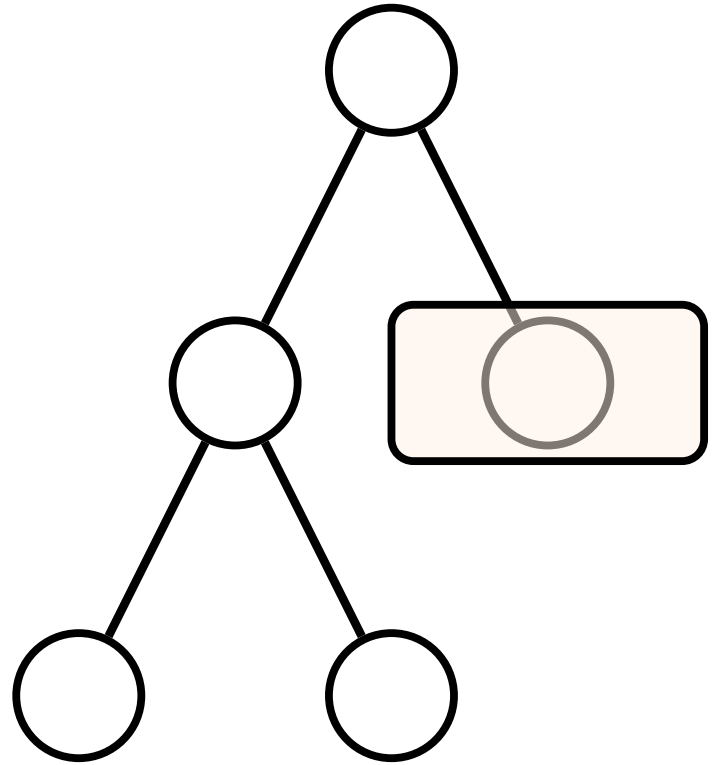
# Under the hood

Structural navigation



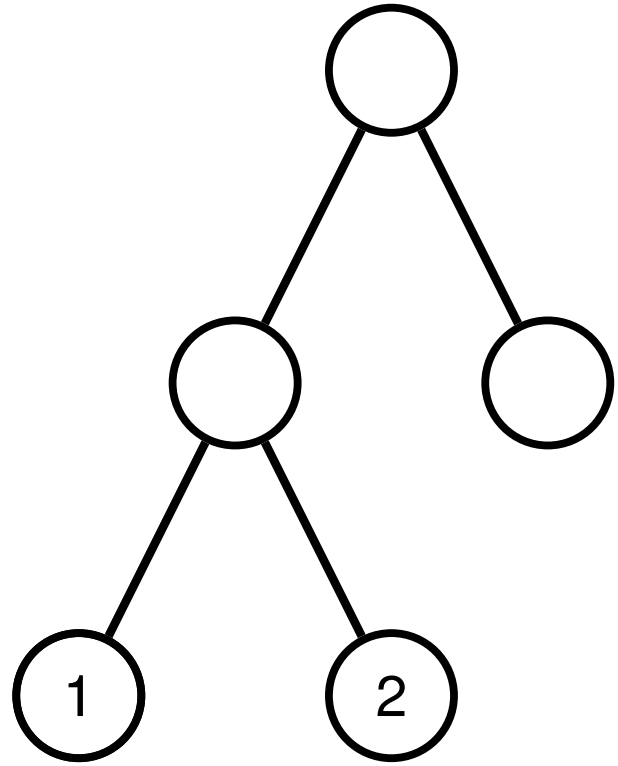
# Under the hood

Structural navigation



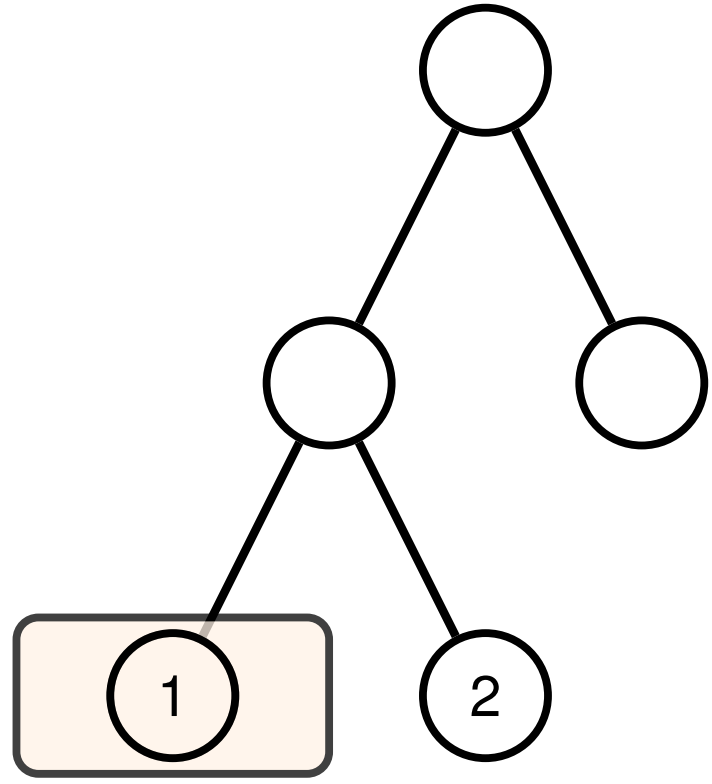
# Under the hood

Structural transposition



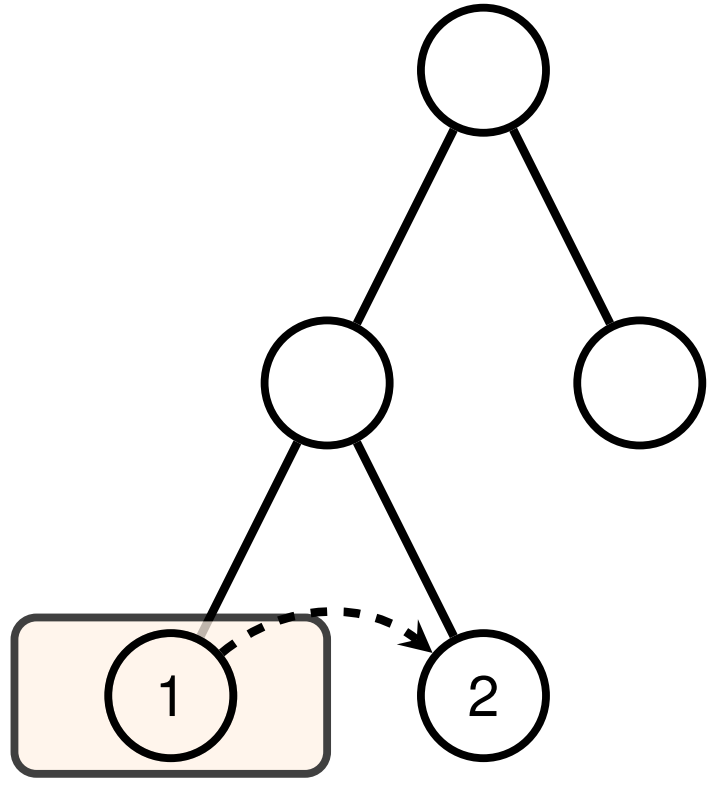
# Under the hood

Structural transposition



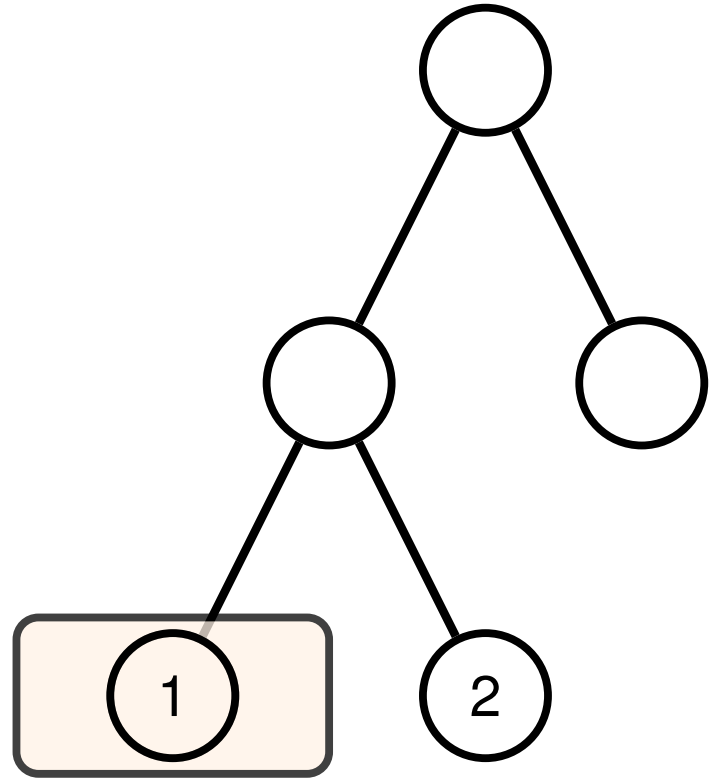
# Under the hood

Structural transposition



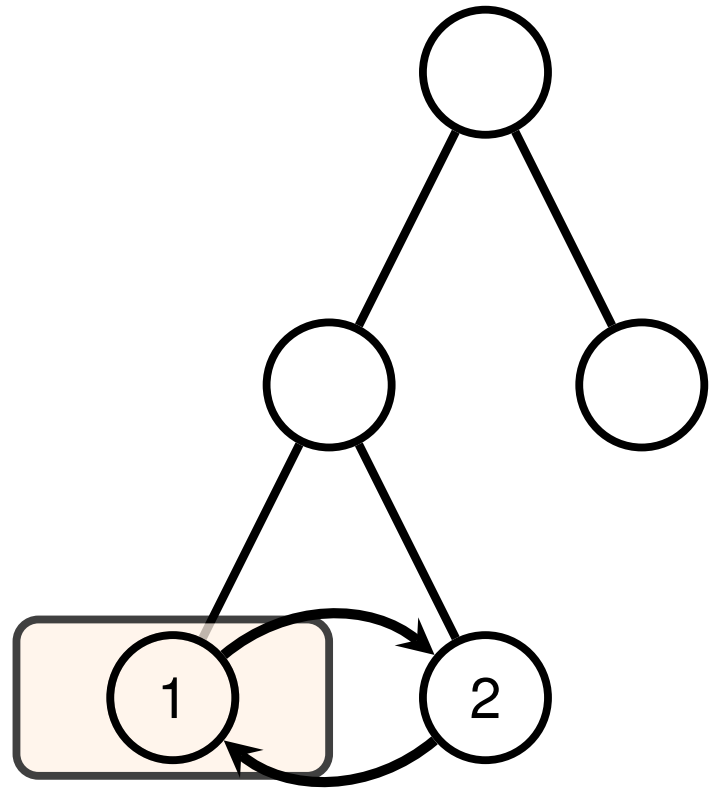
# Under the hood

Structural transposition



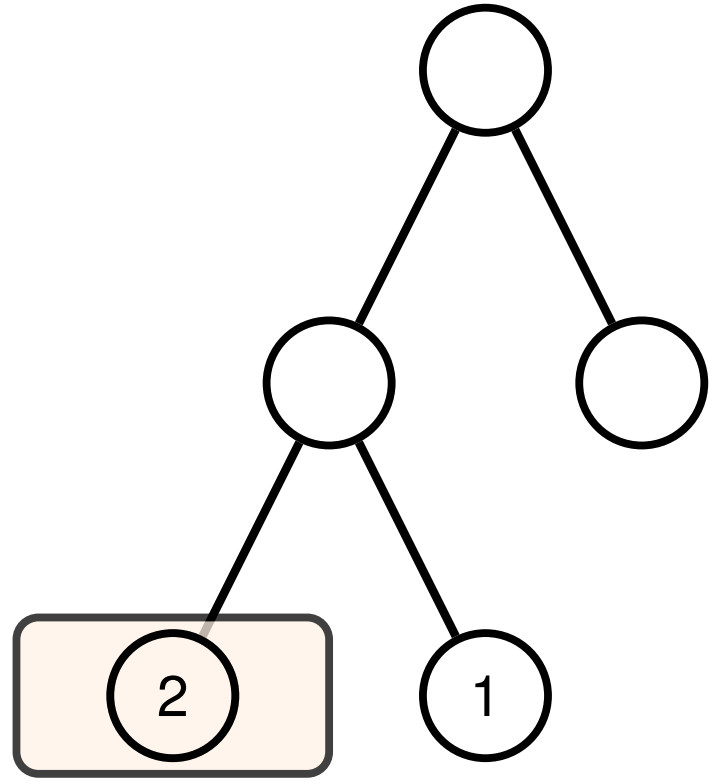
# Under the hood

Structural transposition



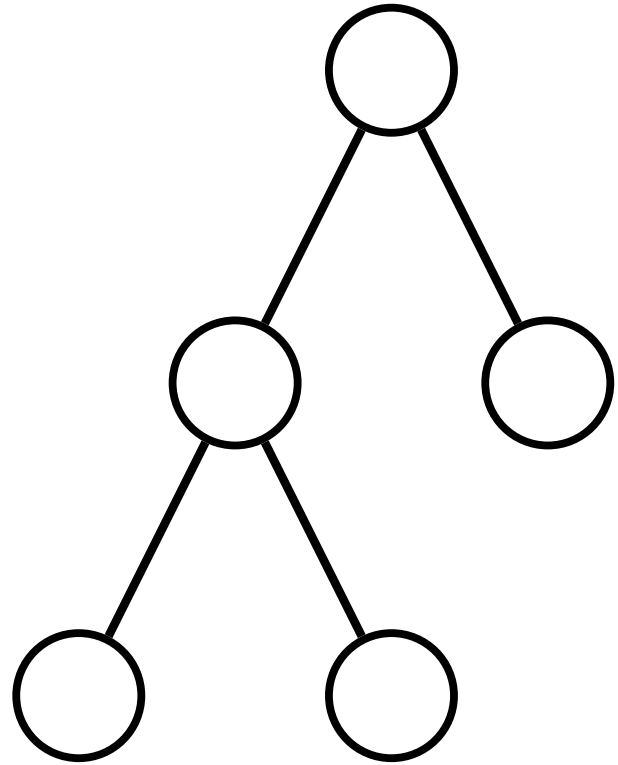
# Under the hood

Structural transposition



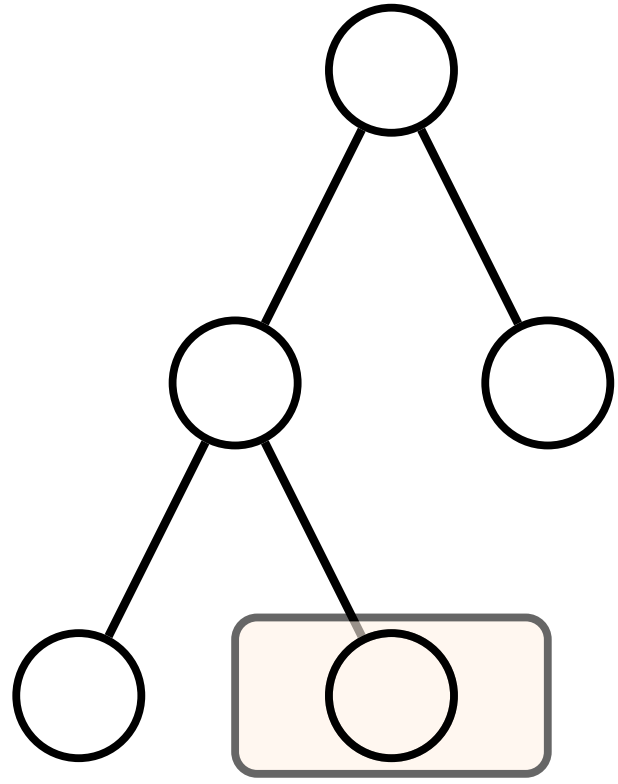
# Under the hood

Structural deletion



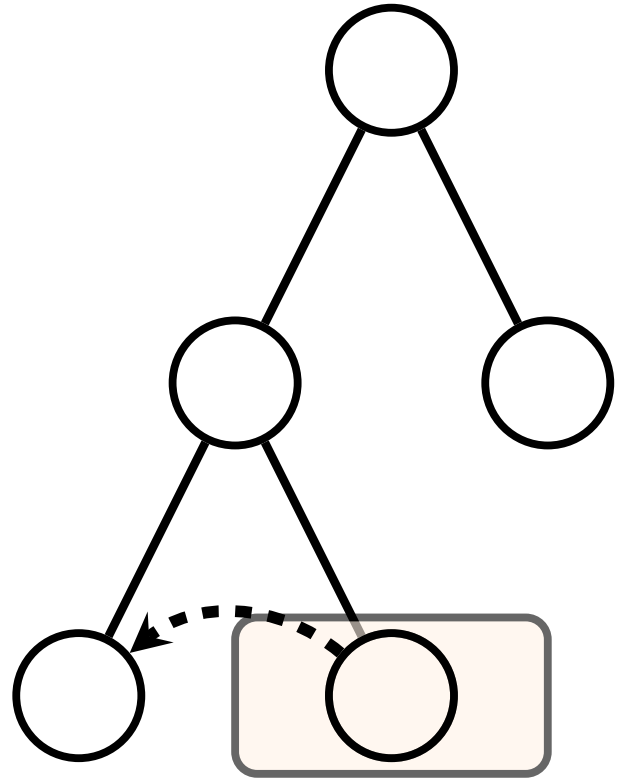
# Under the hood

Structural deletion



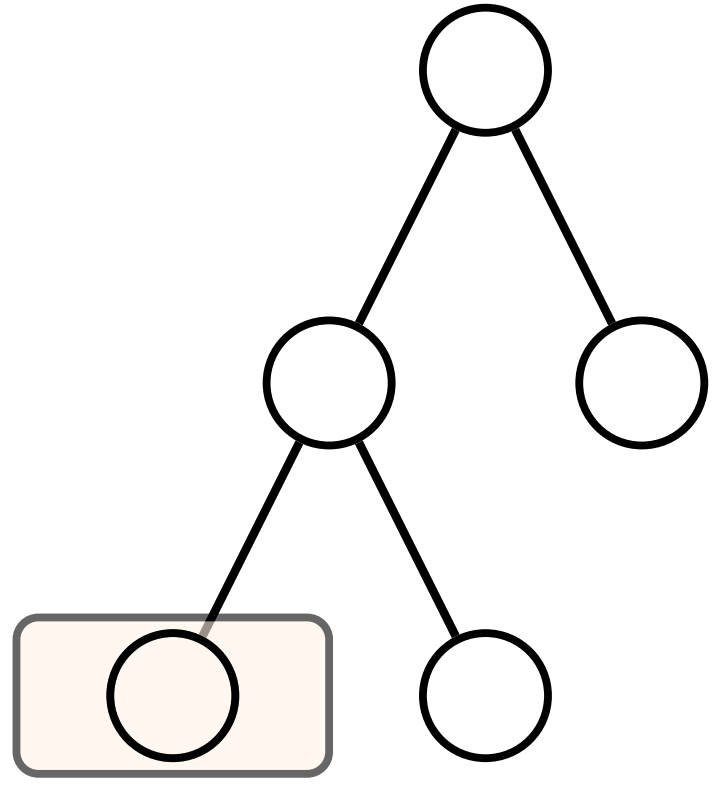
# Under the hood

Structural deletion



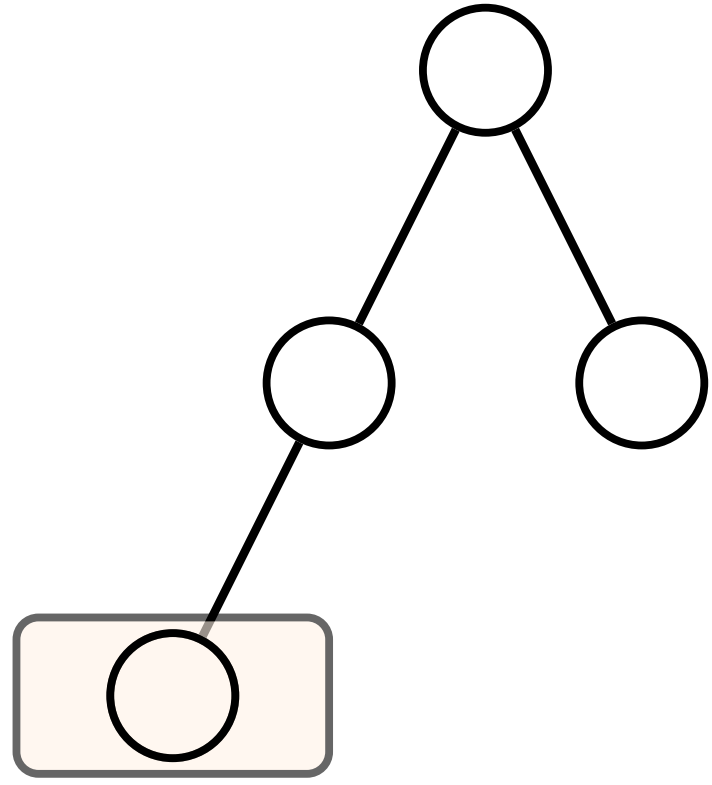
# Under the hood

Structural deletion



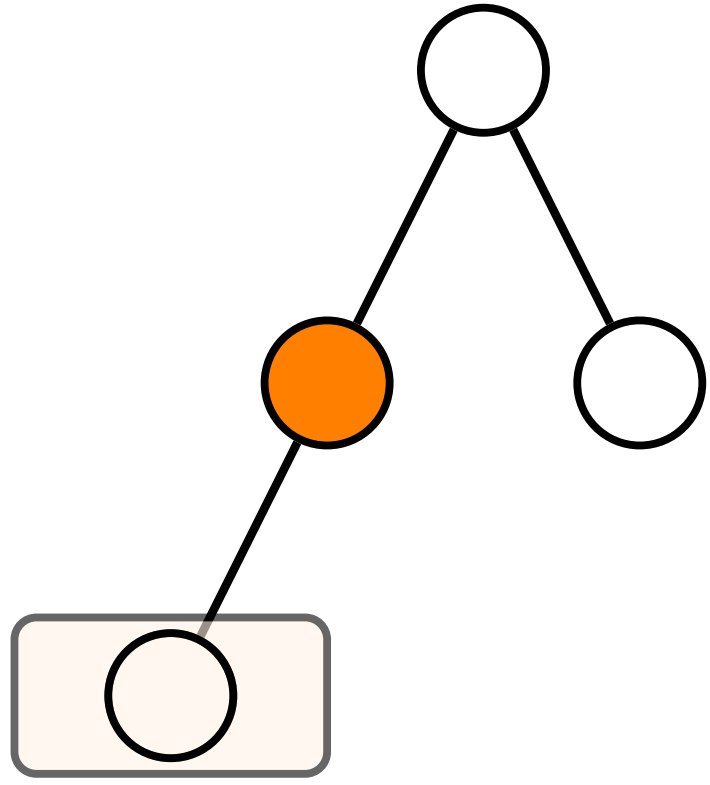
# Under the hood

Structural deletion



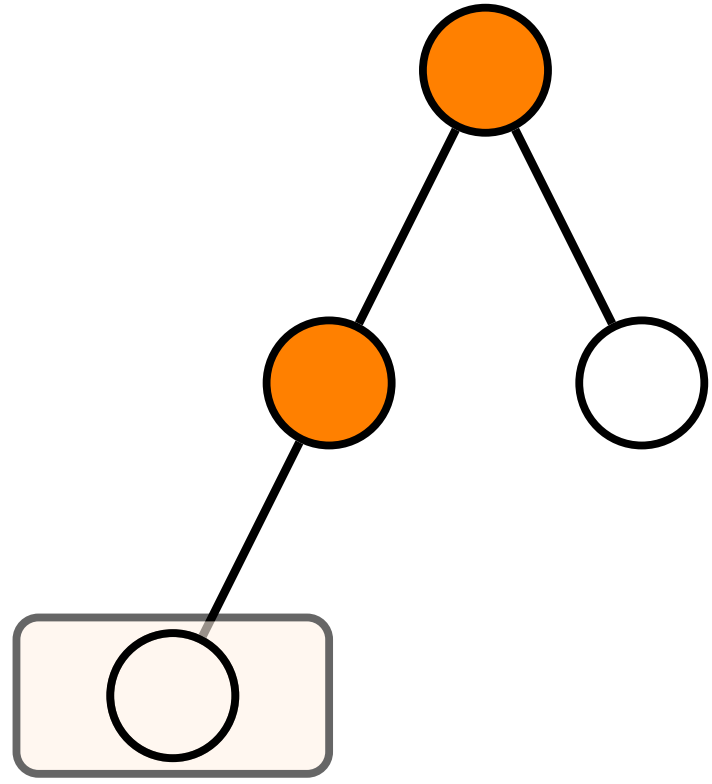
# Under the hood

Structural deletion



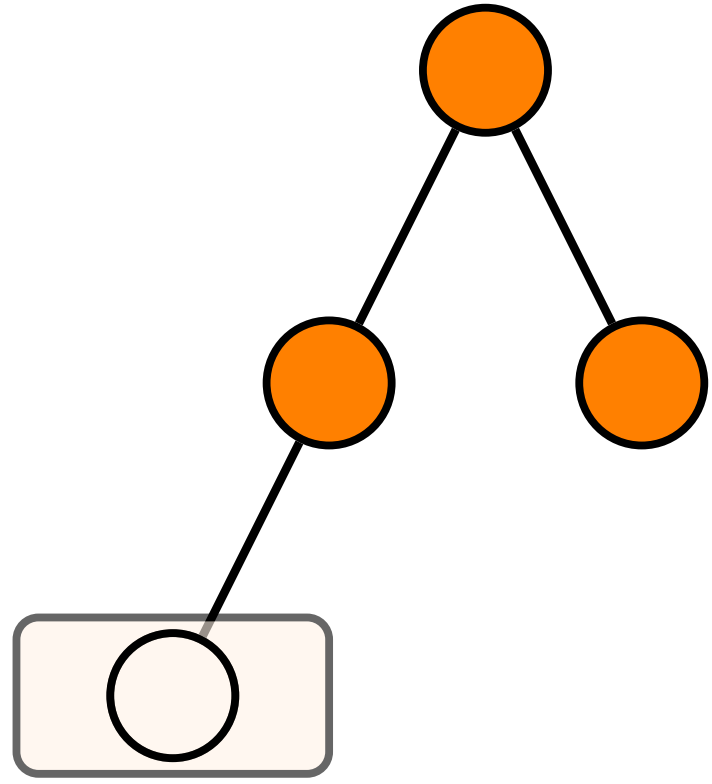
# Under the hood

Structural deletion



# Under the hood

Structural deletion



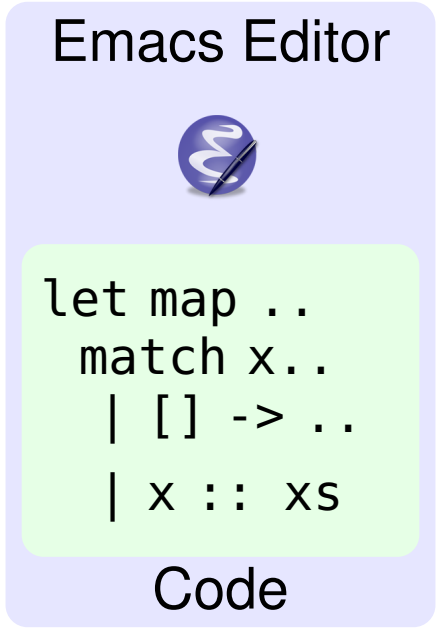
# Under the hood

Structural deletion

How can we *integrate* this with an **editor**?

# Under the hood

## System Architecture



# Under the hood

## System Architecture

An **Emacs** package...

Emacs Editor

```
let map ..
 match x..
  | [] -> ..
  | x :: xs
```

Code

Gopcaml mode

*...written in **OCaml***
*using **Ecaml***

# Under the hood

## System Architecture

Emacs Editor

```
let map ..
 match x..
  | [] -> ..
  | x :: xs
```

Code

Gopcaml mode

# Under the hood

## System Architecture

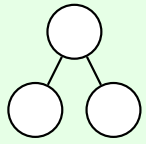
Emacs Editor

```
let map ..
 match x..
  | [] -> ..
  | x :: xs
```

Code

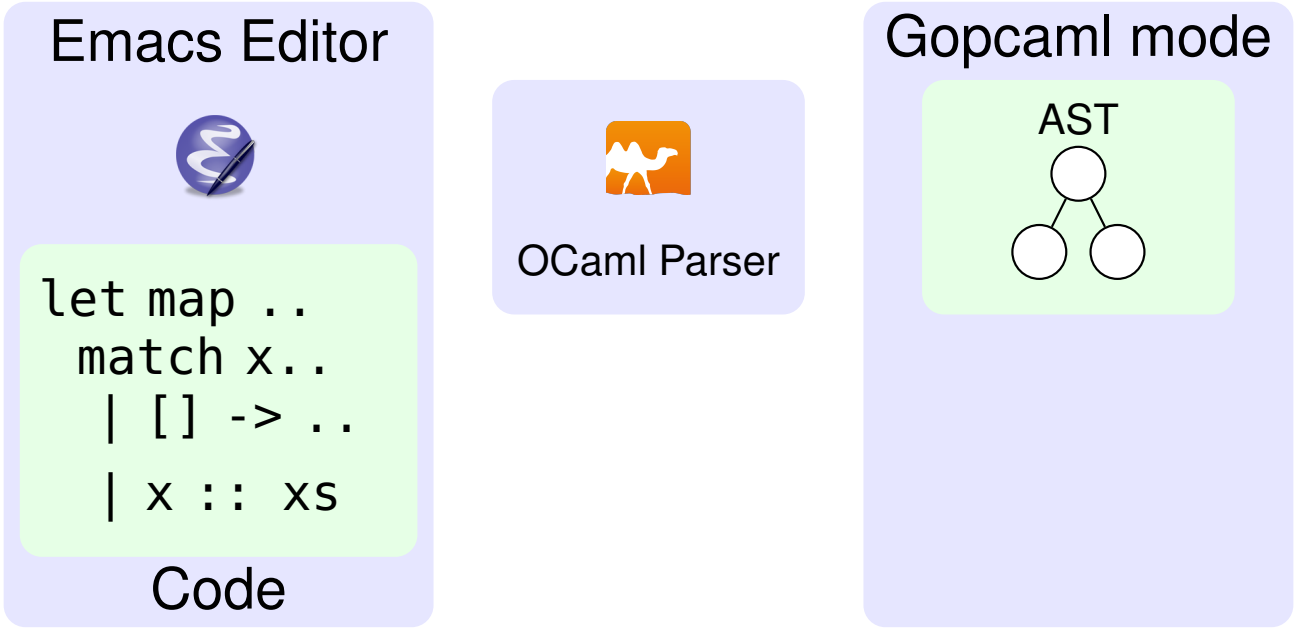
Gopcaml mode

AST



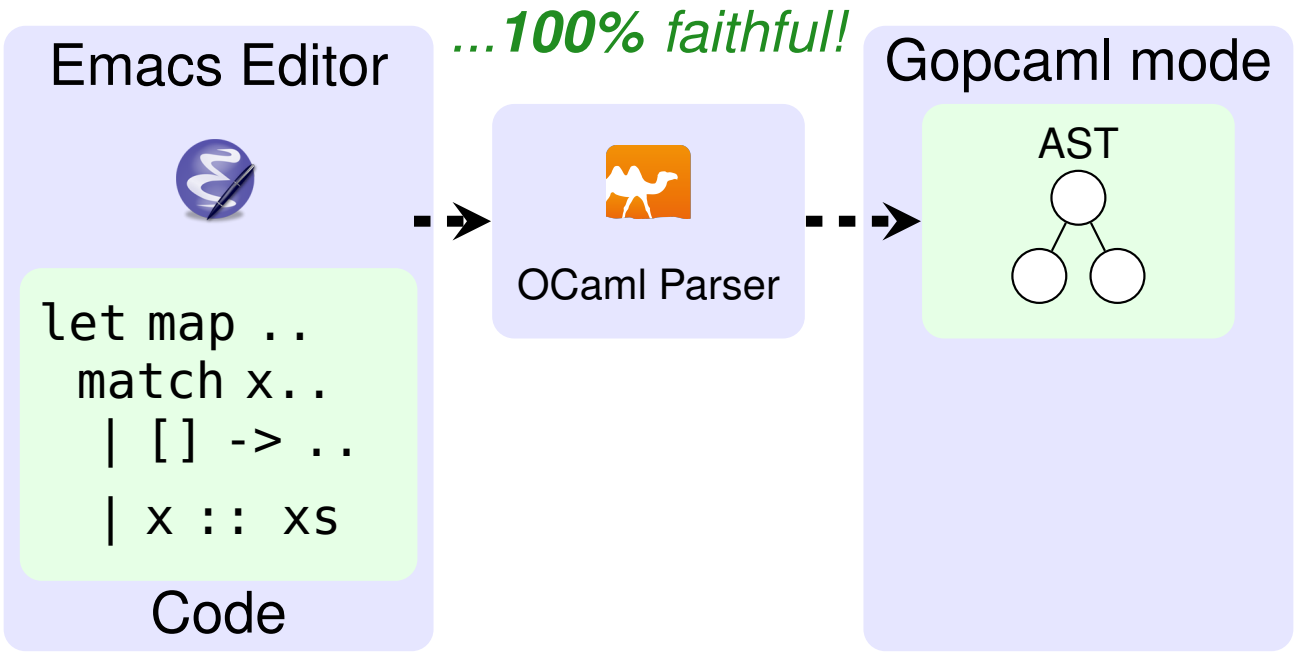
# Under the hood

## System Architecture



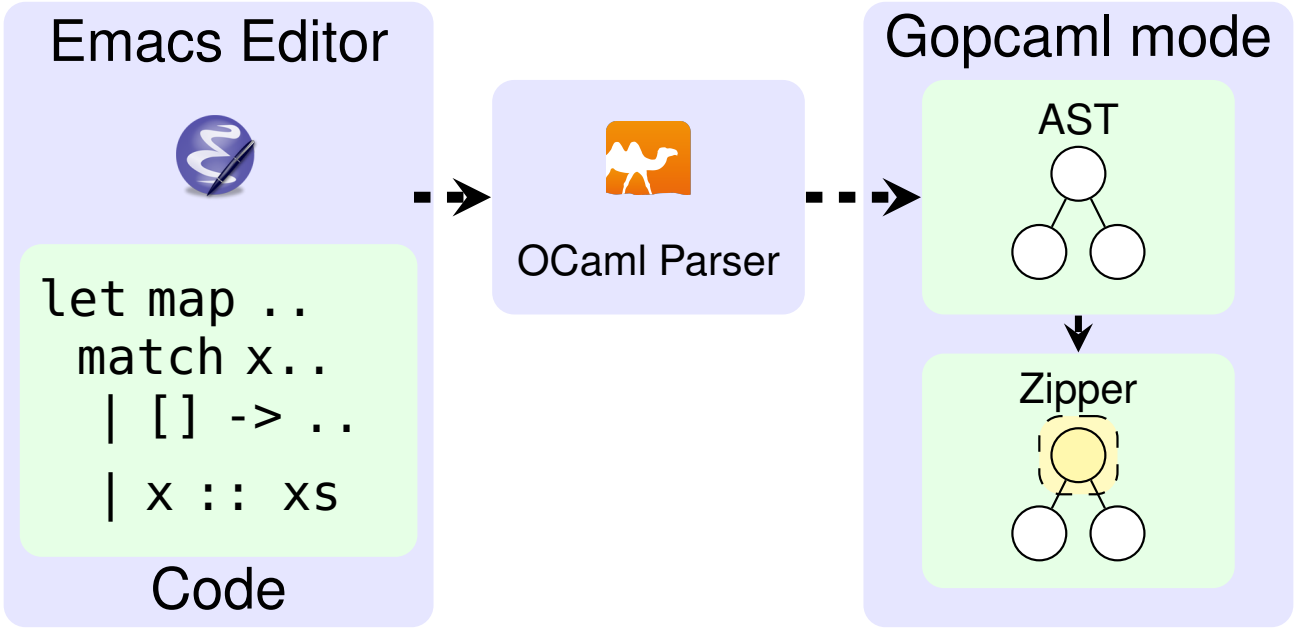
# Under the hood

## System Architecture



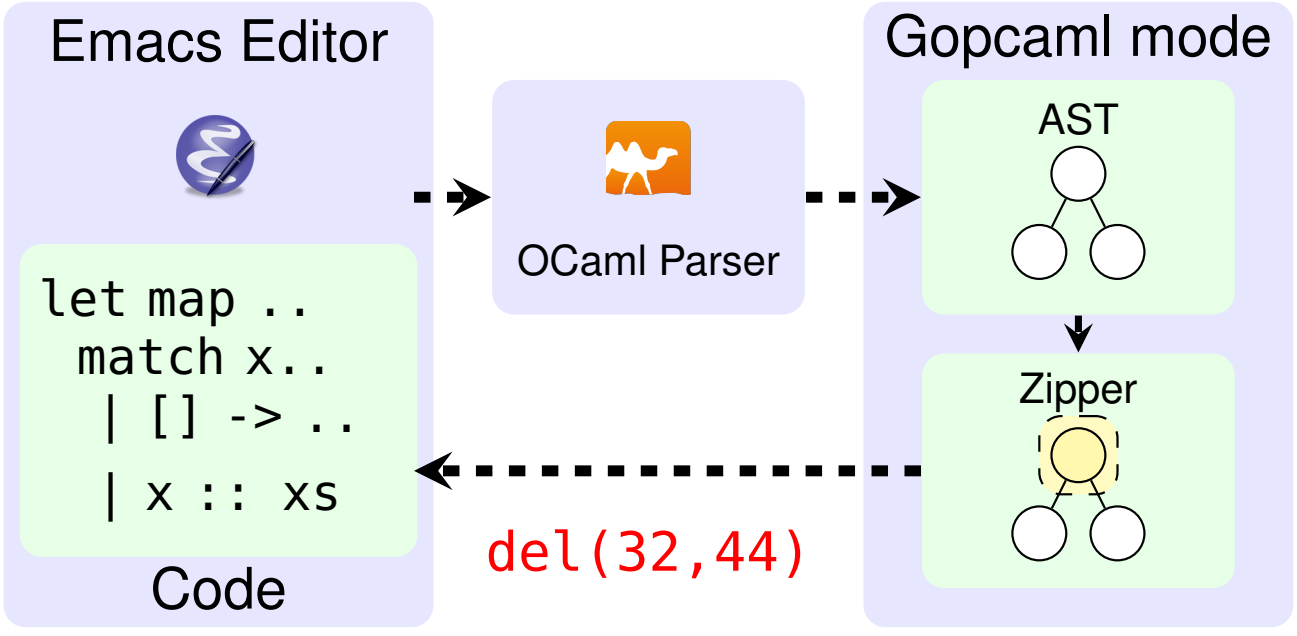
# Under the hood

## System Architecture



# Under the hood

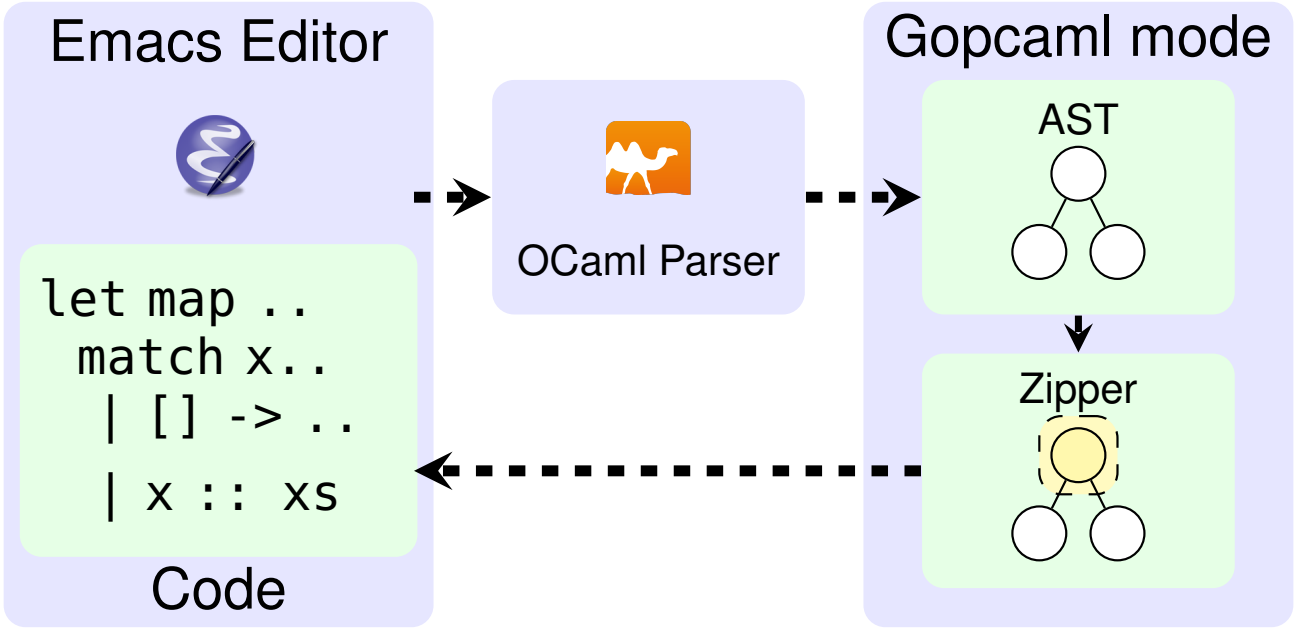
## System Architecture




*...**simple** text operations*

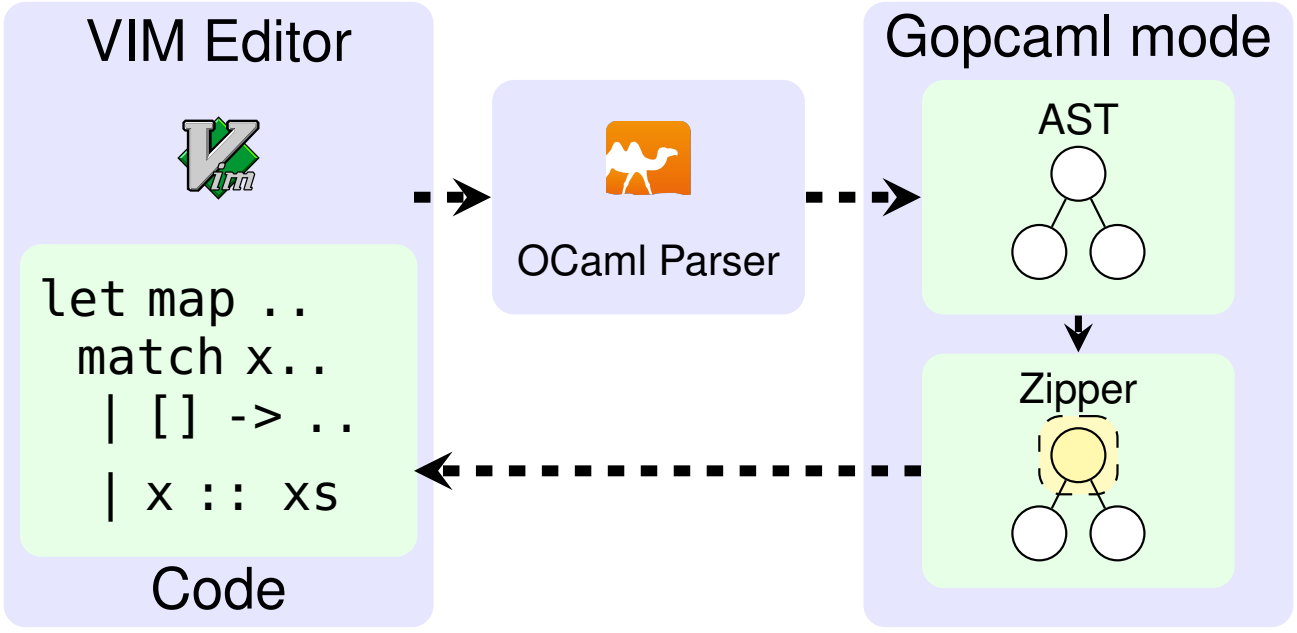
# Under the hood

## System Architecture



# Under the hood

## System Architecture




Core framework is **generic** over editor

# Overview

- **GopCaml**: Generic Framework for Structural Editing
  - Leverages OCaml compiler pipeline for **faithful** parsing
  - Tracks Concrete-Syntax-Tree (CST) of edited file
  - Defines *common* editing operations as CST transformations
- **GopCaml-mode**: Emacs plugin using Gopcaml
  - **Robust** and **consistent** OCaml support
  - **Seamless** integration with Emacs workflows

# Future work

- Support for other editors (VIM, Neovim, VScode)
- Robustness to invalid syntax (a la `Merlin`)
- Semantic aware transformations
- MetaOCaml Support

https://gitlab.com/gopiandcode/gopcaml-mode

# *Interested*?.... **Try it out**!

Install from OPAM:

```
opam install gopcaml-mode
```

Load in `.emacs.d`:

```
(add-to-list 'load-path
    "~/.opam/default/share/emacs/site-lisp")
(autoload 'gopcaml-mode "gopcaml-mode" nil t nil)
(add-to-list 'auto-mode-alist
    '("\\.ml[ily]?$" . gopcaml-mode))
```

https://gitlab.com/gopiandcode/gopcaml-mode

# *Interested*?.... **Try it out**!

Install from OPAM:

```
opam install gopcaml-mode
```

Load in `.emacs.d`:

```
(add-to-list 'load-path
    "~/.opam/default/share/emacs/site-lisp")
(autoload 'gopcaml-mode "gopcaml-mode" nil t nil)
(add-to-list 'auto-mode-alist
    '("\\.ml[ily]?$" . gopcaml-mode))
```

# ...Profit!