Freeze / Seal for GC values and globals

Léo Andres, Pierre Chambart

October 12th, 2023

LMF, OCamlPro

Building recursive values

```
(module
  (rec
    (type $t (struct (field $f (mut (ref null $t))))))

  (func $loop (result (ref $t))
    (local $l (ref $t))
    (local.set $l (struct.new $t (ref.null $t)))
    (struct.set $t $f (local.get $l) (local.get $l))
    (local.get $l)
  )
)
```

Building recursive values, but immutable ?

```
(module
  (rec
    (type $t  (struct (field $f (mut (ref null $t))))))

  (rec
    (type $t' (struct (field $f      (ref      $t')))))

  (func $loop (result (ref $t)) ... )
)
```

There is currently no way to build a value of type `$t'`

Freezing/Sealing

```
(module
  (rec
    (type $t          freezable  (struct (field $f (mut (ref null $t))))))

  (rec
    (type $t_freeze (freeze $t) (struct (field $f      (ref      $t_freeze)))))

  (func $loop_tmp (result (ref $t)) ... )

  (func $loop (result (ref $t_freeze))
    (ref.freeze $t_freeze $t (call $loop_tmp))
  )
)
```

The point ?

The same as immutable values in general:

- cleaner API / preserve code invariants
- avoid read barriers
- just more explicit about the use

Also can avoid null checks

Could also allow creating immutable arrays

The idea

- The *freezability* check can be similar to the subtyping rules:
  - there should be the same (or less ?) fields
  - can remove `mut` annotations
  - can remove `null` annotations
  - fields should be subtypes
    or frozen versions of subtypes
    (maybe also upcasts ?)
- After the freeze the freezable values should not be accessed
  => dynamic checks on freezable types accesses
  - The freeze operation is expected to walk the value and flip a *frozen* bit
  - Trap if fields are still null at freeze time

The idea

- heuristic: unfrozen values are seldom accessed, frozen ones can be accessed a lot
- Freezing is not ‘fixed number of hardware instruction’
  But the combined time of building then freezing is kind of an amortized version of it

Globals

```
(module
  (rec (type $t (freeze $t) (struct (field $f (ref $t)))))
  (global $g (mut (ref null $t)) (ref.null $t))

  (func $f (result (ref $t))
    (ref.as_non_null (global.get $g))
  )
  (func $loop (result (ref $t)) ...)

  (func $st
    (global.set $g (call $loop))
    (drop (call $f))
  )
  (start $st))
```

Freezing globals

```
(module
  (global $g (mut (ref null $t)) (ref.null $t))
  (global $g_frozen (ref $t) (freeze $g))
  (func $f (result (ref $t))
    (global.get $g_frozen) <-- no test
  )

  (func $st
    (global.set $g (call $loop))
    (drop (call $f))
  )
  (start $st))
```

But when are the globals actually frozen ?

Phases

```
(module
  (global $g (mut (ref null $t)) freezable (ref.null $t))
  (global $g_frozen (ref $t) (freeze $g) (phase 1))

  (func $f (phase 1) (result (ref $t))
    (global.get $g_frozen)
  )
  (func $loop (phase 0) (result (ref $t)) ...)

  (func $st_0 (phase 0) (global.set $g (call $loop)))
  (func $st_1 (phase 1) (drop (call $f)))
  (start $st_0 (phase 0))
  (start $st_1 (phase 1))
)
```

Phases

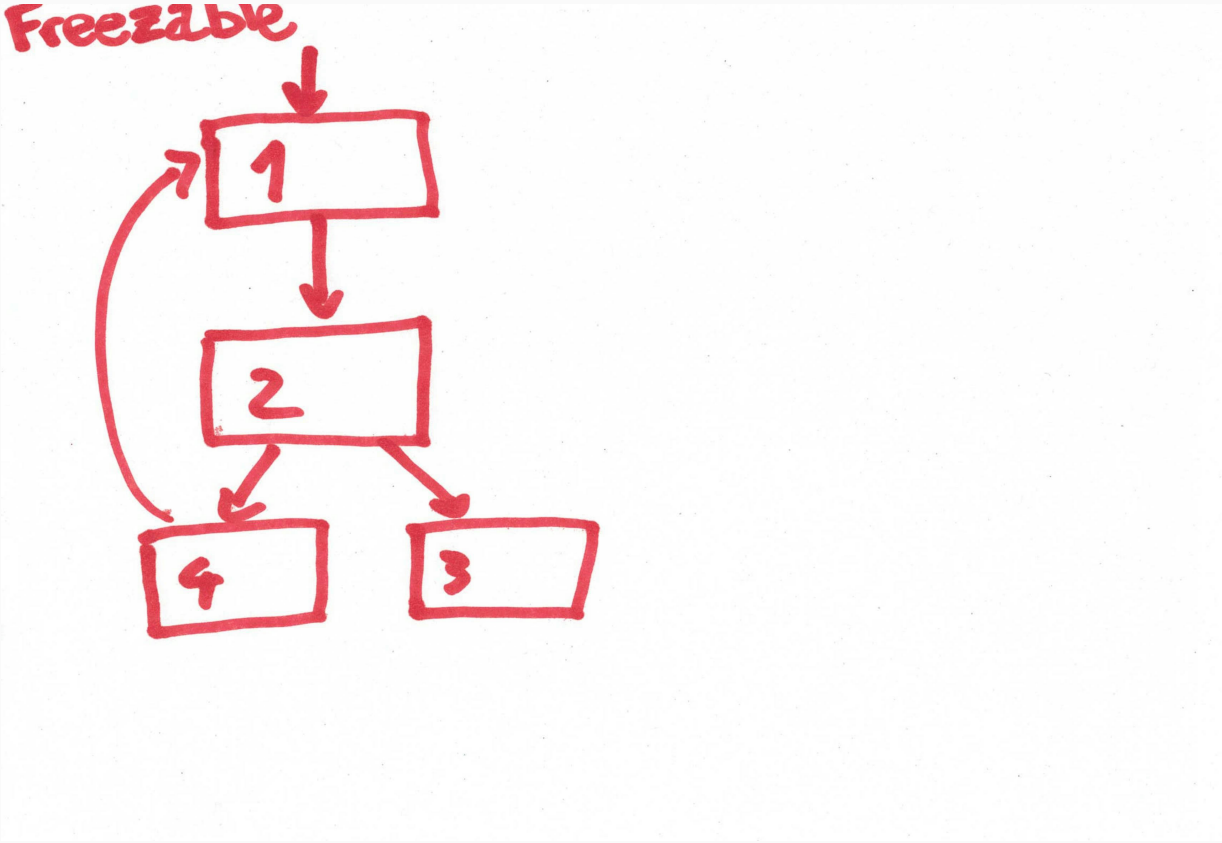
- invariant: cannot call a function of phase n before the end of the start of phase $n-1$.
- A function can only refer to values of phase less or equal than its own phase.
  (ref and calls)
- Each phase can have one start
- Starts are run in phase order
- Globals are frozen at the end of the previous phase
  Failure to freeze is a panic
- Accessing an unfrozen global from a previous phase is a panic
- cannot export freezable global

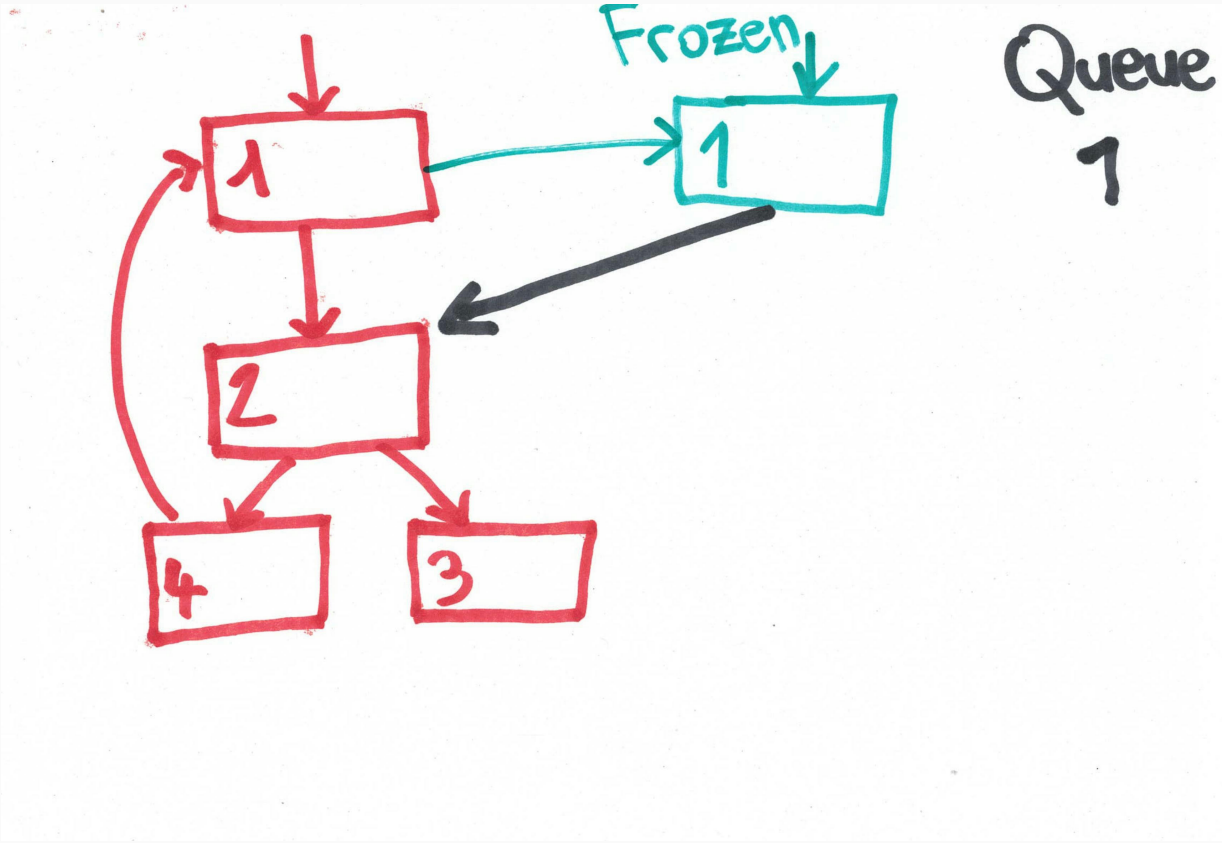
Phases

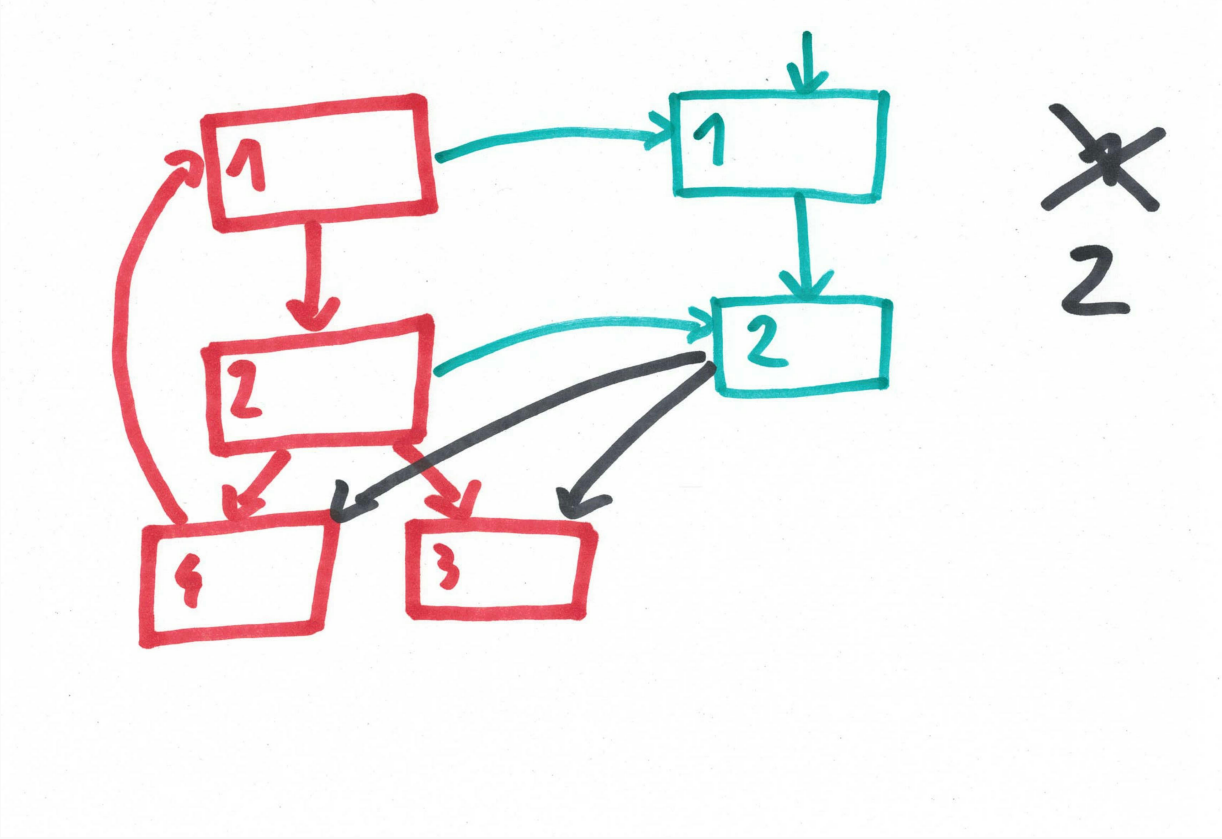
- if multiple start functions seems distastefull, we could have a `call_and_freeze` instruction moving to the next phase
- default phase is 0
- global freeze can the change type of its contents
- Encoding: not really thought, could be compact

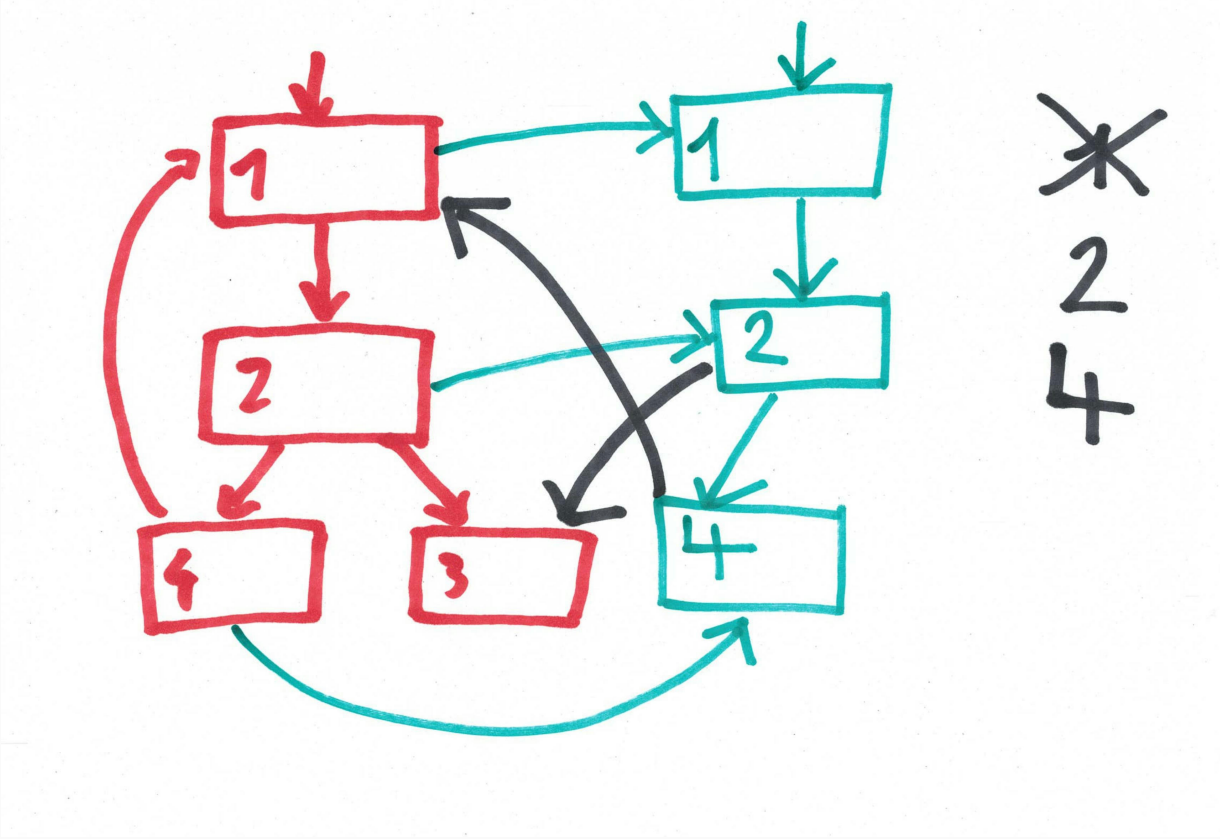
Proposal

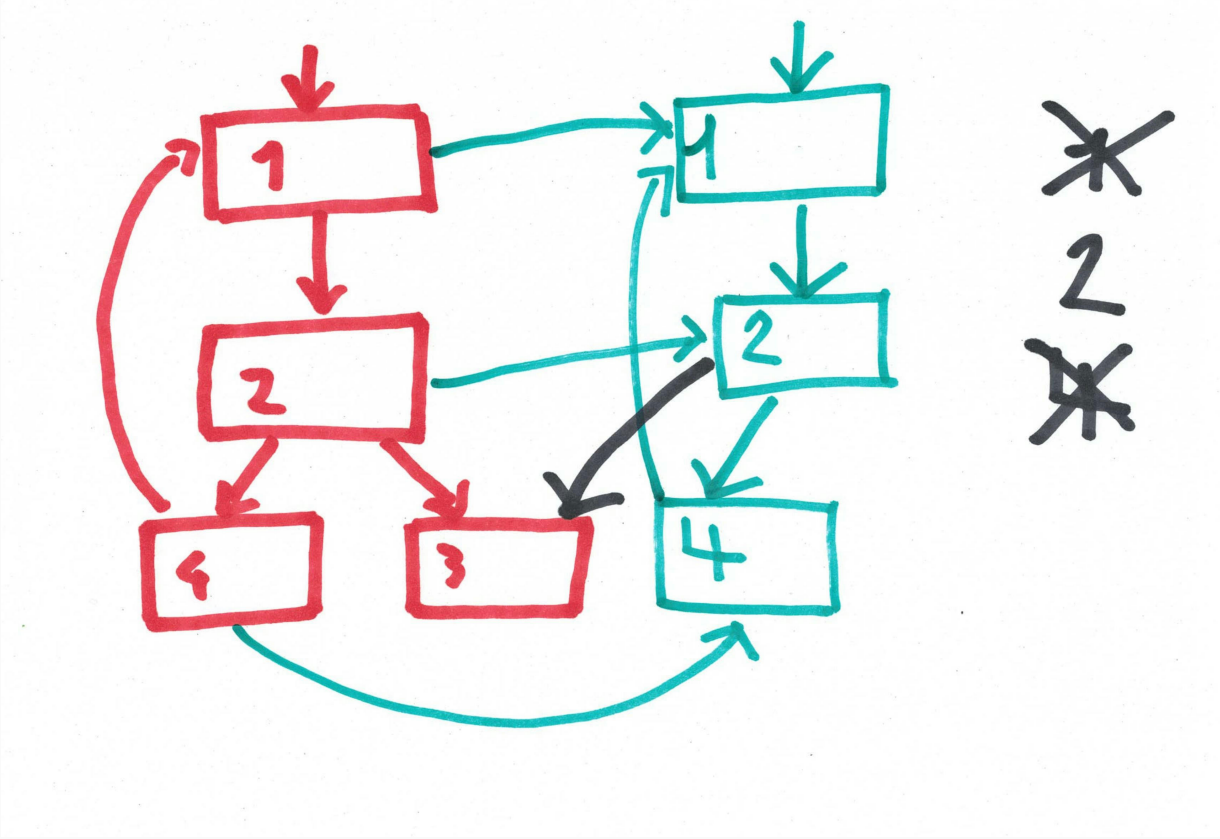
Should there be a phase 1 proposal to explore that kind of needs ?

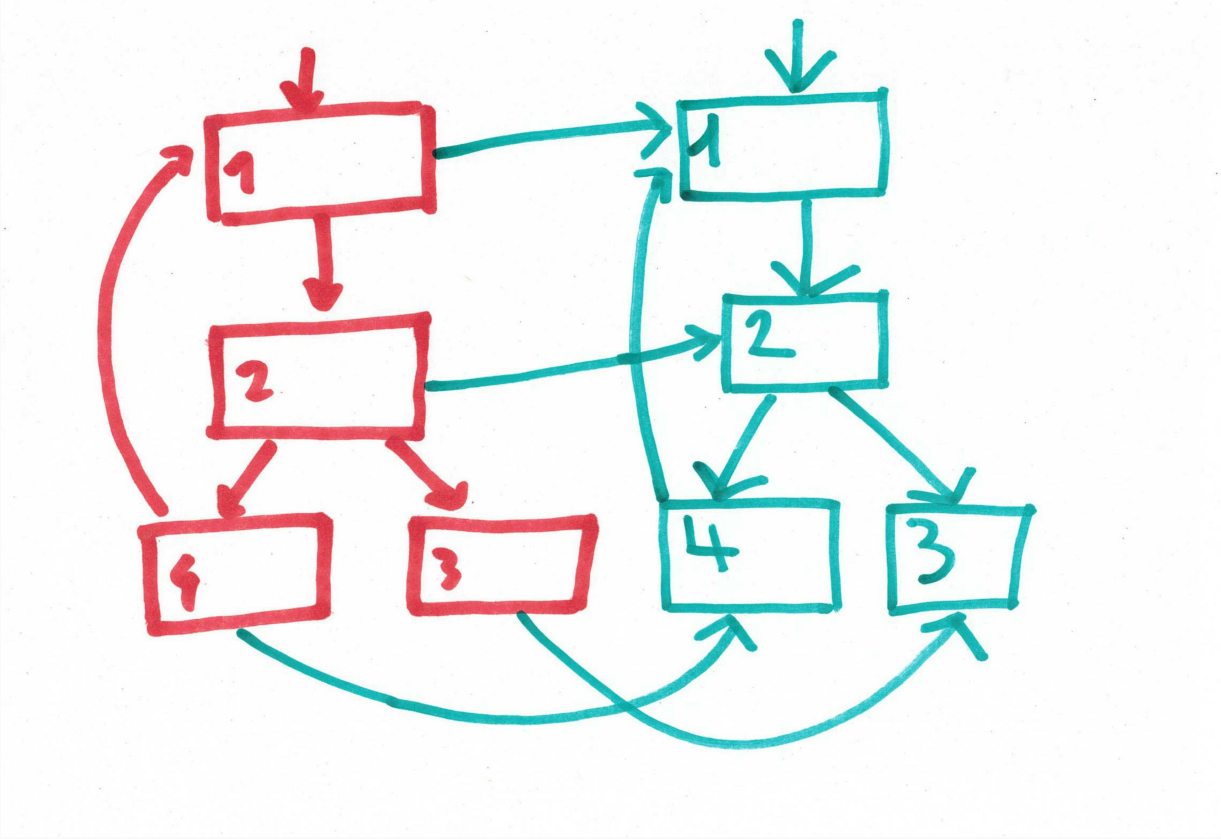

Freezable
1
2
4
3

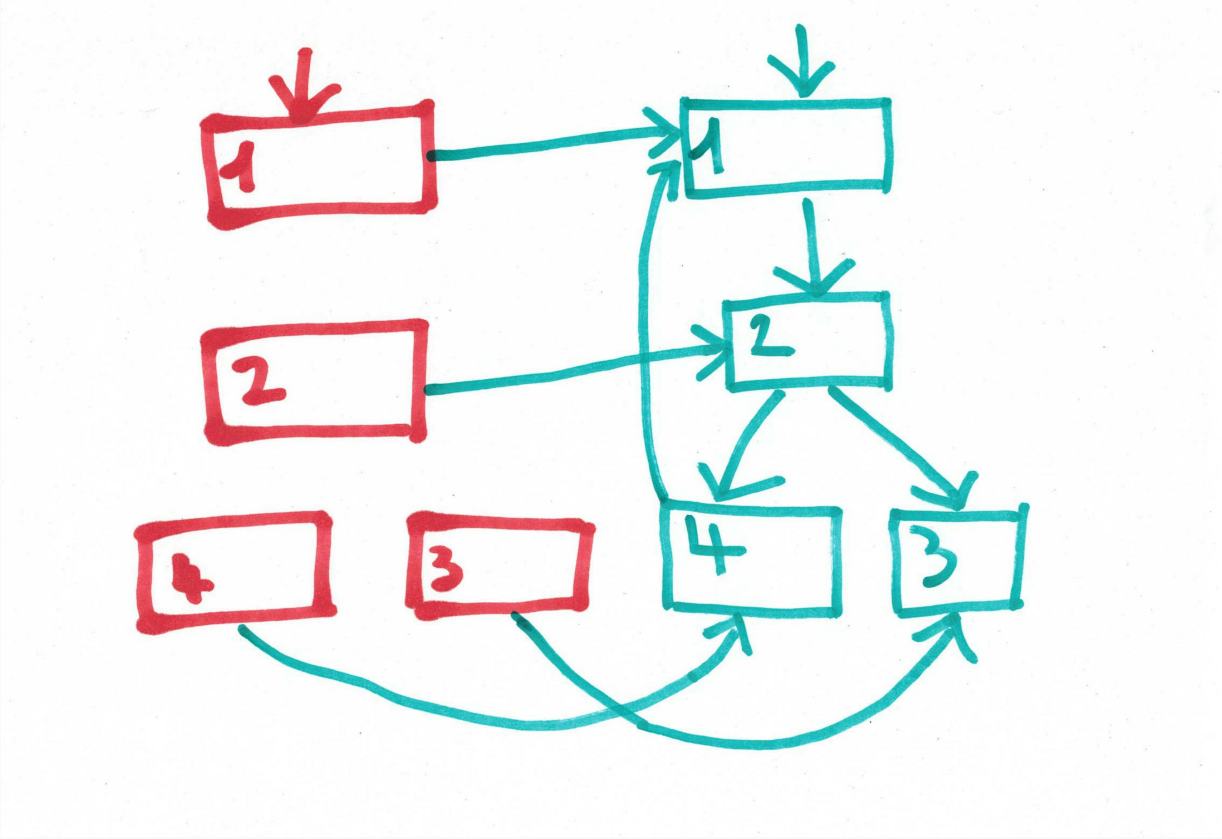

Frozen
Queue
1
1
1
2
4
3


1
2
4
3
1
2
2


1
2
4
3
1
2
4
2
4


1
2
4
3
1
2
4
2


1
2
4
3
1
2
4
3


1
2
4
3
1
2
4
3