Type System for Metaprogramming

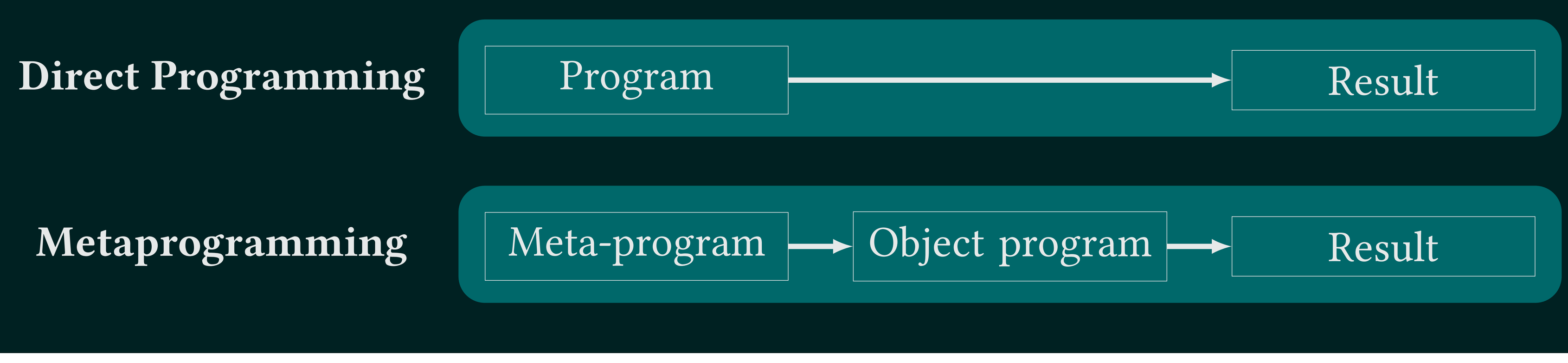
McGill

Junyoung "Clare" Jang
https://Ailrun.github.io
Complogic Group

Metaprogramming

The art of generating, manipulating, and analyzing code

(Used for Compiler, Program Analyzer, Program generator, etc.)



Metaprogramming with Box/Let-box

`box(2 + 3)` A code fragment for `2 + 3`

`let box(C) = box(2 + 3) in C` Execution of code `2 + 3` to get `5`

```
untypedNth n =
  if n <= 0 then
    box(head xs)
  else
    let box(C) = untypedNth (n - 1) in
    box(let xs = tail xs in C)
```

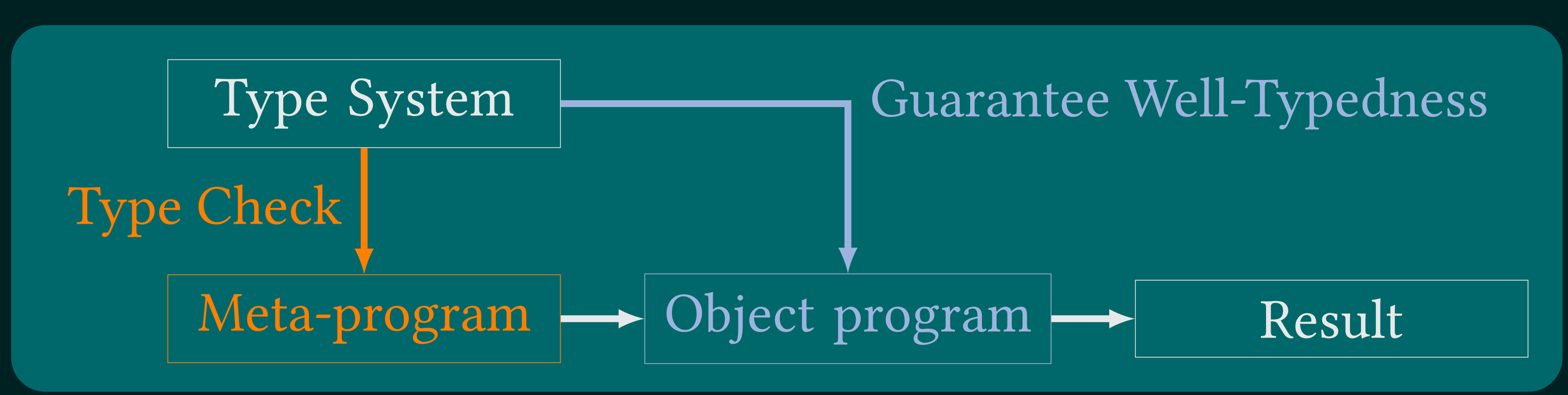
Untyped Metaprogram to get n-th item in a list

If we evaluate `untypedNth 2`, we get

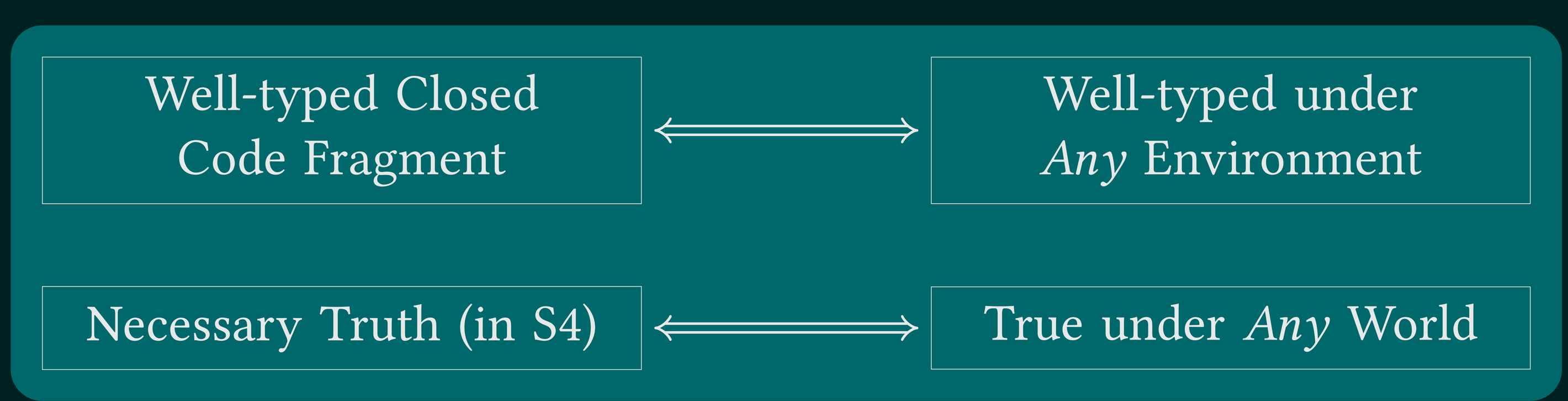
```
box(let xs = tail xs in
    let xs = tail xs in
    head xs)
```

Typed Metaprogramming

We want to ensure that generated code will be well-typed *before* executing a meta-program.



Modal Logic



`□ A` A type for code generating `A`-typed program

`box(M)` has type `□ A` — `M` has type `A` but does not use any environment-specific information

```
intNth : Int -> □ (Int list -> Int)
intNth n =
  if n <= 0 then
    box(fun xs => head Int xs)
  else
    let box(C) = intNth (n - 1) in
    box(fun xs => C (tail Int xs))
```

Typed Metaprogram to get nth item in an int list

If we execute `intNth 2`, we get

```
box(fun xs =>
      (fun xs =>
        (fun xs => head xs)
        (tail xs))
      (tail xs))
```

Problem 1 — Well-typed Open Code Fragments

`untypedNth 2` has a free variable `xs`, but we cannot type check this as-is with Simple S4. Thus, `intNth 2` has some redundant anonymous functions and function calls.

Problem 2 — Polymorphism

`intNth 2` works only for `Int` lists. The following approach to polymorphism does not work:

```
badPolyNth : ('a : Type) -> Int -> □ ('a list -> 'a)
badPolyNth 'a n =
  if n <= 0 then
    box(fun xs => head 'a xs)
  else
    let box(C) = badPolyNth 'a (n - 1) in
    box(fun xs => C (tail 'a xs))
```

Ill-typed Metaprogram to get nth item in a list

because code `fun xs => head 'a xs` depends on `'a` and thus is not closed.

Problem 3 — Pattern Matching on Code Fragments

Some text

Contextual Modality

Some text

Levels (Opt)

Some text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text

Levels in Polymorphism

Some text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text

Levels in Pattern Matching

Some text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text

References

Some text