



**Hewlett Packard  
Enterprise**

# **PLANNED IMPROVEMENTS TO THE CHAPEL COMPILER**



Michael Ferguson, HPE

June 4, 2021

# COMPILER IMPROVEMENTS: OUTLINE

- Where We Are
- Missing Features
- Proposed Direction
- Progress Report





**WHERE WE ARE**

# PROBLEMS WITH THE CURRENT COMPILER

---

- **Speed**

- The current compiler is generally slow, and extremely so for large programs (~7s to 15 minutes)

- **Structure and Program Representation**

- The compiler is structured only for whole-program analysis, preventing separate/incremental compilation
- Unclear how to integrate an interpreter, provide IDE support, or 'eval' Chapel snippets

- **Development**

- The modularity of the compiler implementation needs improvement
- There is a steep learning curve to become familiar with the compiler implementation



# WHAT IS WORKING WELL

---

- The current compiler code base has enabled the design and evolution of Chapel to date
- The compiler includes key optimizations that support program performance
- The compiler is relatively fast to build and very portable
- Chapel's internal and library modules are extensive and largely independent of the compiler
- The runtime libraries are well-architected and not in need of major changes



# SUMMARY OF CURRENT COMPILER PASSES

<b>pass</b>	<b>time for Hello World</b>	<b>time for Arkouda</b>	<b>approx lines of .cpp</b>
parse+	0.5s	0.8s	10,000 lines
scopeResolve	0.4s	0.7s	4,500 lines
normalize+	0.9s	2.2s	9,000 lines
resolve	2.0s	165s	35,000 lines
post-resolve	0.3s	16s	16,000 lines
lowerIterators	0.1s	7.1s	6,000 lines
parallel	0.1s	17s	2,000 lines
optimization	0.5s	46s	7,000 lines
insertWideRefs+	0.1s	15s	6,000 lines
codegen	0.5s	48s	20,000 lines
makeBinary	1.1s	422s	-
TOTAL OF ABOVE	6.0s	724s	114,000 lines
TOTAL	6.4s	743s	170,000 lines in 'compiler/*'

**MISSING FEATURES**

A vibrant nebula with a bright cyan core and reddish-pink outer layers, set against a starry black background. The text "MISSING FEATURES" is overlaid on the left side of the image.

# MISSING FEATURES

---

- Some commonly requested features:
  - IDE support (beyond syntax highlighting)
  - Separate and/or incremental compilation
- Good IDE integration requires the compiler to
  - behave more like a server
  - respond quickly to limited queries (e.g. code completion or mouse-over)
- Incremental compilation and separate compilation require
  - a more flexible compiler that can instantiate some generics but re-use other instantiations
- Compiler architecture improvements can make these problems easier to solve

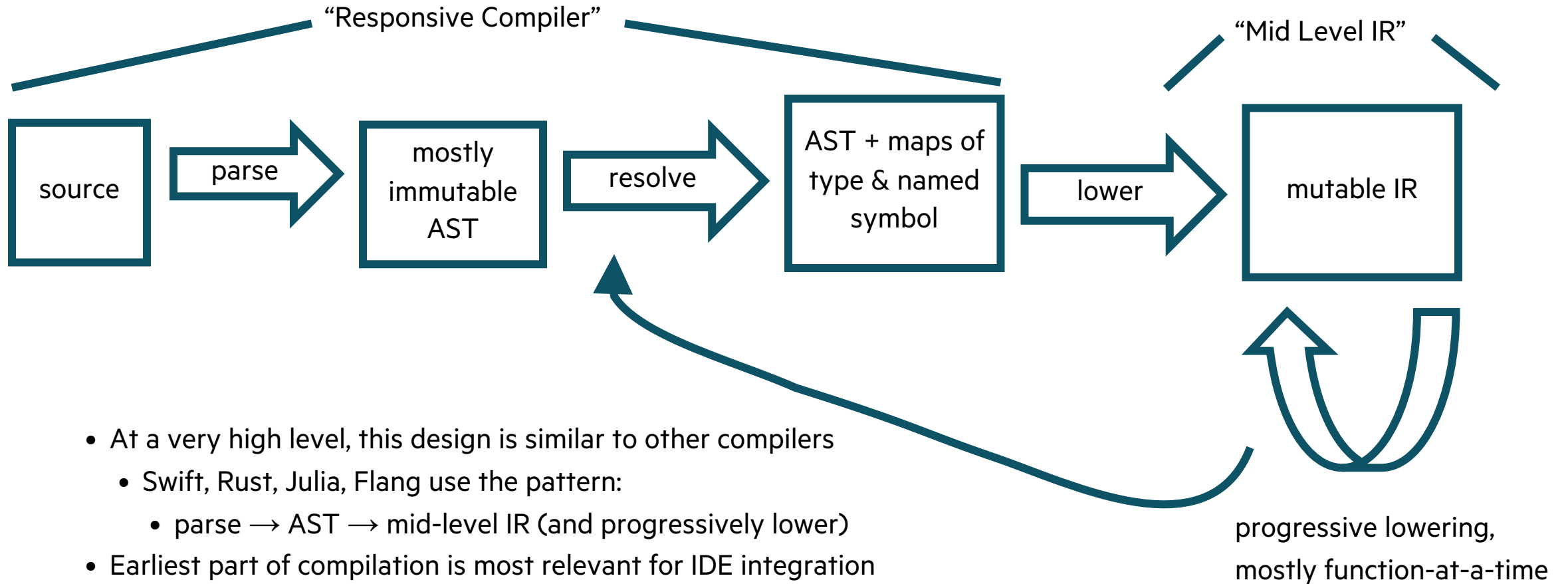






**PROPOSED DIRECTION**

# IMPROVED COMPILER ARCHITECTURE



- At a very high level, this design is similar to other compilers
  - Swift, Rust, Julia, Flang use the pattern:
    - parse → AST → mid-level IR (and progressively lower)
- Earliest part of compilation is most relevant for IDE integration
  - so is designed to be very incremental



# CHANGES TO GET TO IMPROVED ARCHITECTURE

---

- Create a new AST more faithful to source code for early passes
- Develop a new pass architecture with less rigid ordering
  - make passes typically run per-function rather than whole-program and otherwise be idempotent
- Convert the new AST into the old AST to enable incremental development
- Gradually port later passes over to a new IR more suited for optimization



# RESPONSIVE COMPILERS

---

- Matsakis' talk, *Responsive Compilers*<sup>1</sup>, presents a vision for good IDE support:
  - highly incremental and demand-driven—just process enough to answer a query
    - e.g., how to complete `newBlock<tab>`
    - fast response times are key for a satisfying experience
  
- The strategy relies on:
  - structuring compilation in terms of many fine-grained queries
    - e.g., what is the type of this variable?
  - framework uses these queries to manage dependencies among results
  - each query saves its result and is re-run when necessary
  - query results are represented separately from the input—which tends to mean a lot of maps
  - AST elements are given IDs to support these maps

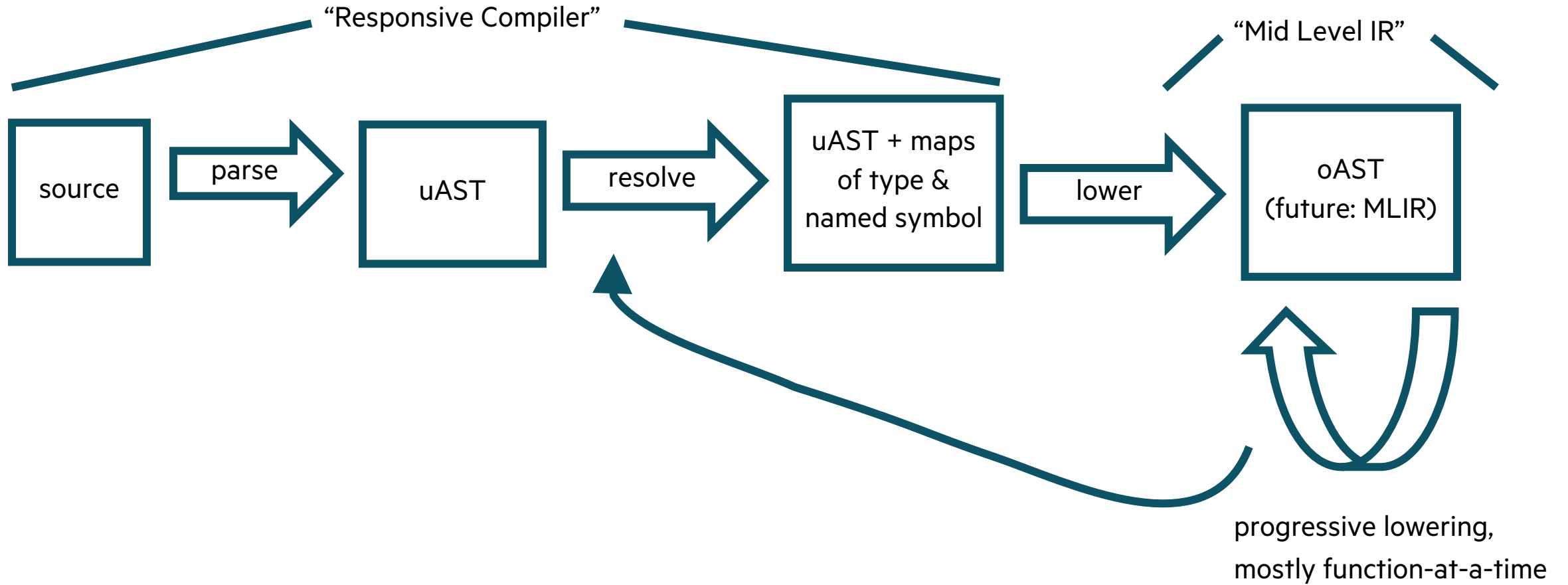
# RESPONSIVE COMPILER QUERIES

---

- `parse(filePath)` → AST for file (which also establishes IDs)
- `locate(AST)` → (line number, column number)
  
- `getDefinedIn(Expr, name)` → Symbols defined in 'Expr' named 'name'
- `getVisible(Expr, name)` → Symbols visible from 'Expr' named 'name'
- `types(Expr)` → map from Symbols to Types for Symbols defined in 'Expr'
- `resolve(Expr)` → map from Identifiers to Symbols they refer to



# FUTURE COMPILER ARCHITECTURE





# **PROGRESS REPORT**

# COMPILER LIBRARY

- Developing new code as a library
- New uAST nodes have documentation!
- These features will enable community members to contribute Chapel tools
- Library Use Cases:
  - Linter
  - Documentation tools
  - IDE integrations

The screenshot shows a web browser window displaying the Chapel Compiler Internals documentation. The page title is "Compiler Internals — Chapel Doc X" and the URL is "file:///Users/mferguson/w/master/build/doc/html/compiler-internals/index.html". The page version is 1.25. The left sidebar contains a search bar and a navigation menu with sections: COMPILING AND RUNNING CHAPEL, WRITING CHAPEL PROGRAMS, LANGUAGE HISTORY, and COMPILER INTERNALS. The main content area shows the documentation for the `While` class, which inherits from `chpl::uast::Loop`. It includes an `#include <While.h>` directive and a description: "This class represents a while loop. For example:". An example code block shows a `while` loop that prints numbers 1 through 5. Below the example, there are sections for "Public Functions" and "Public Static Functions". The "Public Functions" section shows the `~While() override = default` method. The "Public Static Functions" section shows the `static owned<While> build(Builder *builder, Location loc, owned<Expression> condition, ASTList stmts, bool usesImplicitBlock)` method, which is described as "Create and return a while loop." Below this, the documentation for the `WithClause` class is shown, which inherits from `chpl::uast::Expression`. It includes an `#include <WithClause.h>` directive and a description: "This class represents a with clause. For example:". An example code block shows a `forall` loop with a `with` clause: `forall myRange with (var x = 0) { writeln(x); }`. At the bottom of the browser window, a search bar shows the search term "inherited" and indicates "17 of 29 matches".



# IMPLEMENTATION PROGRESS

---

- Part-way through implementing the new uAST, parsing it, and translating it into the old AST
- Have demonstrated incremental re-compilation with simple examples

```
// mymodule.chpl
```

```
module M {  
  proc f() {  
    writeln();  
  }  
}
```

```
prompt % testInteractive mymodule.chpl
```

```
mymodule.chpl:3: error: 'writeln' undeclared (first use t
```

```
Module M:
```

```
Module 0x7fc962406250 M
```

```
Function M 0x7fc962406140 f
```

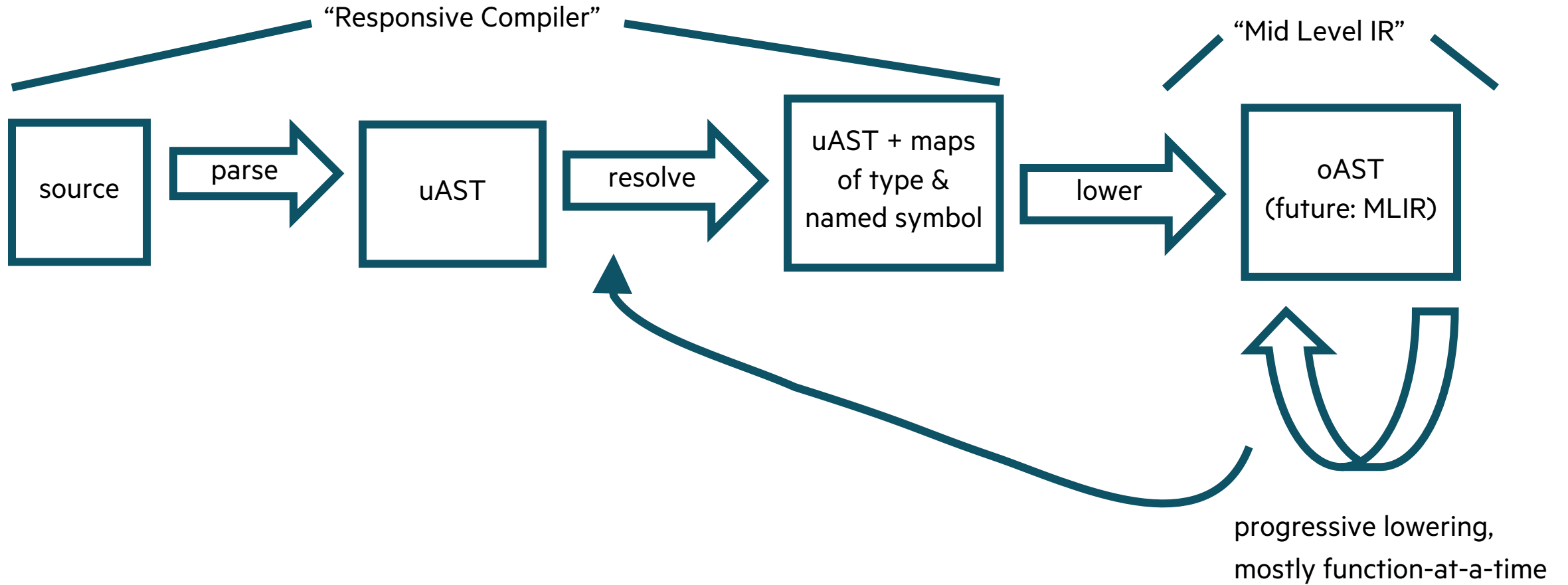
```
FnCall M.f@1 0x7fc9624060a0
```

```
Identifier M.f@0 0x7fc962406020 writeln
```

```
Would you like to incrementally parse again? [Y]:
```



# QUESTIONS?





# THANK YOU

---

<https://chapel-lang.org>  
@ChapelLanguage

