Compiling XQuery to Native Machine Code

Declarative Amsterdam @ Online

2020-10-09

Adam Retter

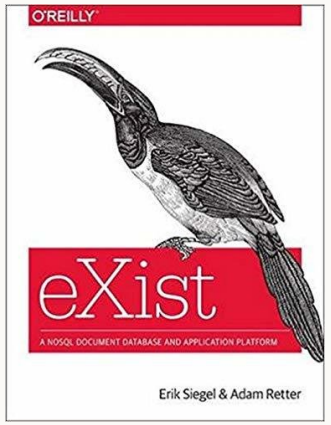
adam@evolvedbinary.com

@adamretter

EVOLVED BINARY

About Me

- **Director and CTO of Evolved Binary**
  - XQuery / XSLT / Schema / RelaxNG
  - Scala / Java / C++ / Rust*
  - Concurrency and Scalability



- **Creator of FusionDB multi-model database (5 yrs.)**
- **Contributor to Facebook's RocksDB (5 yrs.)**
- **Core contributor to eXist-db XML Database (15 yrs.)**
- **Founder of EXQuery, and creator of RESTXQ**
- **Was a W3C XQuery WG Invited expert**
- **Me: www.adamretter.org.uk**

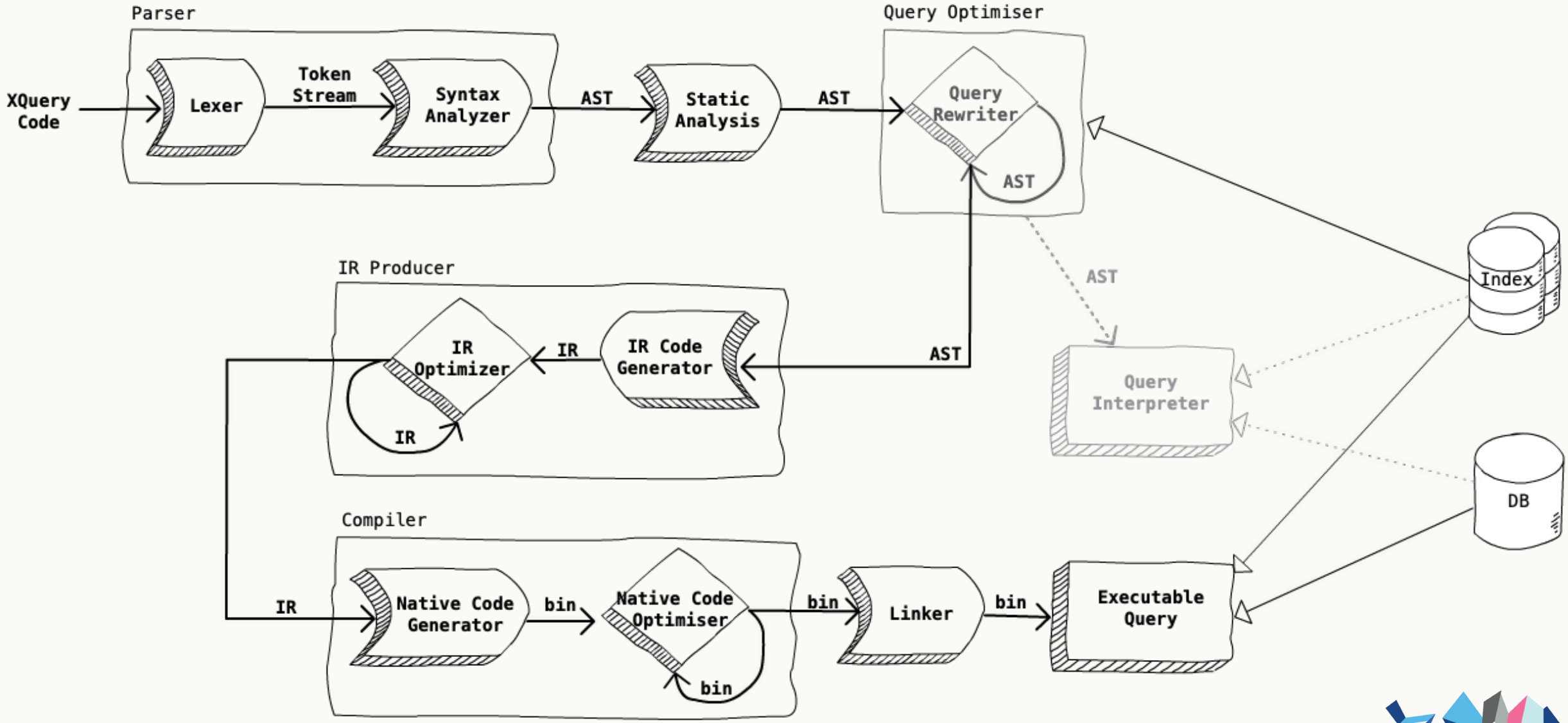
Framing the Problem

- **FusionDB inherited eXist-db's XQuery Engine**
  - Several bugs/issues
    - Especially: Context Item and Expression Precedence
    - See: https://github.com/eXist-db/exist/labels/xquery
  - EOL - Parser implemented in Antlr 2 (1997-2005)
    - Mixes concerns - Grammar with Java inlined in same file
    - Hard to optimise - lack of clean AST for transform/rewrite
    - Hard to maintain - lack of docs / support
  - We made some small incremental improvements
- **Performance Mismatch**
  - XQuery Engine is high-level (Java)
    - Operates over an inefficient DOM abstraction
  - FDB Core Storage is low-level (C++)

Resolving the Problem

- **We need/want a new XQuery Engine for FusionDB**
  - Must be able to support all XQuery Grammars
    - Query, Full-Text, Update, Scripting, etc.
  - Must offer excellent error reporting with context
    - Where did the error occur?
    - Why did it occur?
    - Should suggest possible fixes (e.g. Rust Compiler)
  - Must be able to exploit FDB storage performance
  - Should exploit modern techniques
  - Should be able to utilise modern hardware
  - Would like a clean AST without inline code
    - Enables AST -> AST transformations (Rewrite optimisation)
    - Enables Target Independence

Anatomy of an XQuery Engine



Parser Concerns

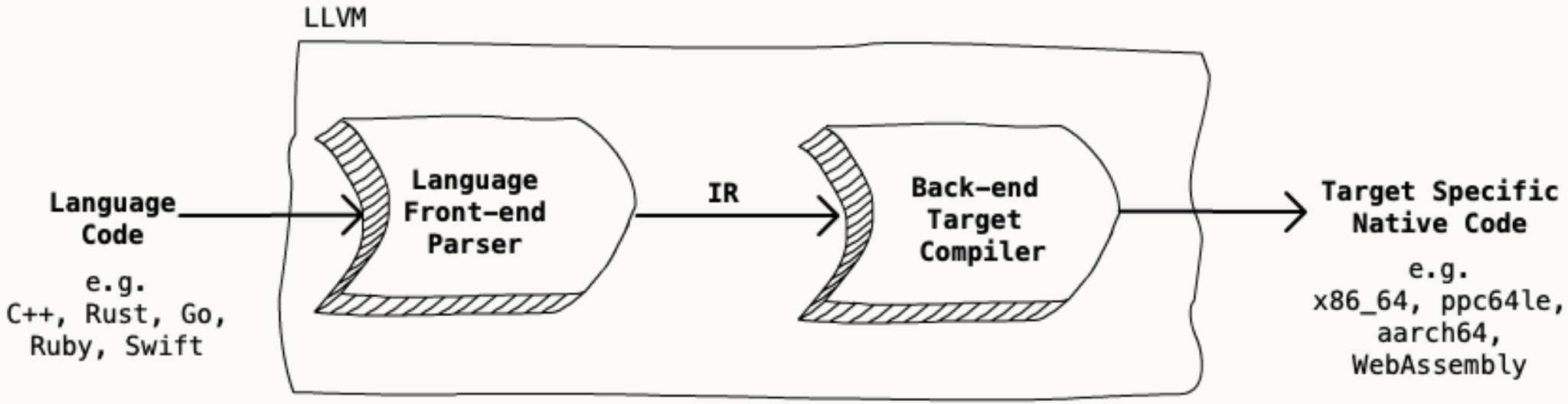
- **Many Different Algorithms**
  - LR - LALR / GLR
  - LL - (Predictive) Recursive Descent / ALL(*) / PEG
  - Others - Earley / Leo-Earley / Pratt / Combinators / Marpa
- **Implementation trade-offs**
  - Parser Generators, e.g. Flex/Bison
    - Pro - Speed of prototyping/impl.
    - Cons - Not just EBNF - Learn additional Grammar Syntax / Poor Error Messages
  - Parser Combinators, e.g. FastParse
    - Pro - Speed of prototyping/impl.
    - Con - Defined in code
  - Bespoke
    - Pro - Complete Control
    - Con - Huge amount of work

Our New XQuery Parser

- **Bespoke**
  - We really care about error messages
  - We really care about performance
  - We wanted a clean AST
  - Can also produce a CST
- **Separate Lexer and Parser**
- **Lexer is Intelligent**
  - Data type conversion, e.g. `IntegerLiteral` -> native integer type
  - Recognises Sub-tokens, e.g. `PredefinedEnityRef` within StringLiteral
- **Predictive Recursive Descent**
- **Implemented in Rust**

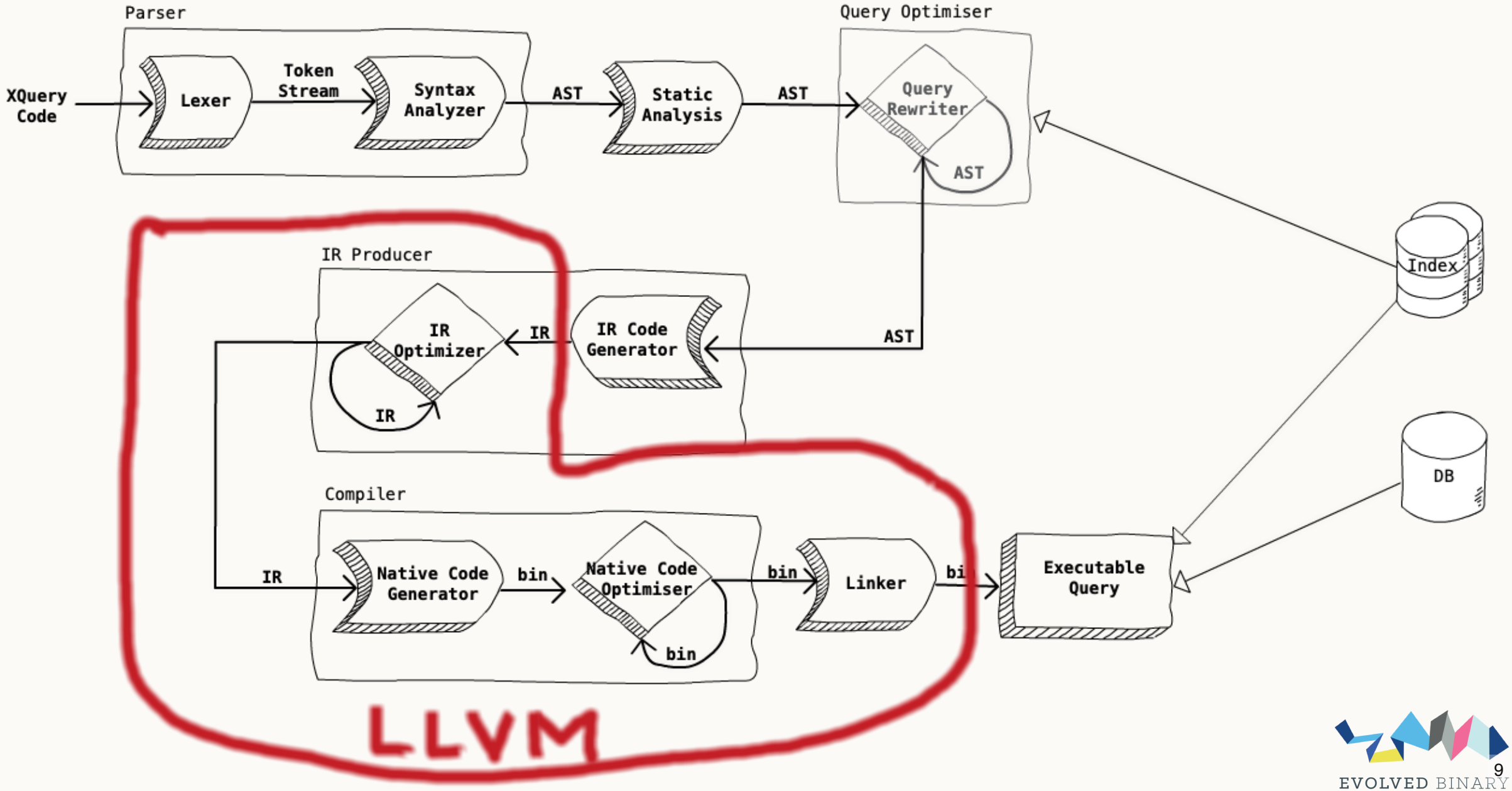
Our New XQuery Compiler

- **Mostly it's just - LLVM!**
  - Previously called: Low Level Virtual Machine
  - Open Source set of compiler and toolchain technologies
  - Front-end for language and Back-end for target
    - Separated by IR (Intermediate Representation) code



- **Our New XQuery Parser produces an AST**
  - Our Language Front-end generates IR from the AST

Anatomy of our New XQuery Engine



EVOLVED BINARY

Computing nth Fibonacci Number - XQuery Code

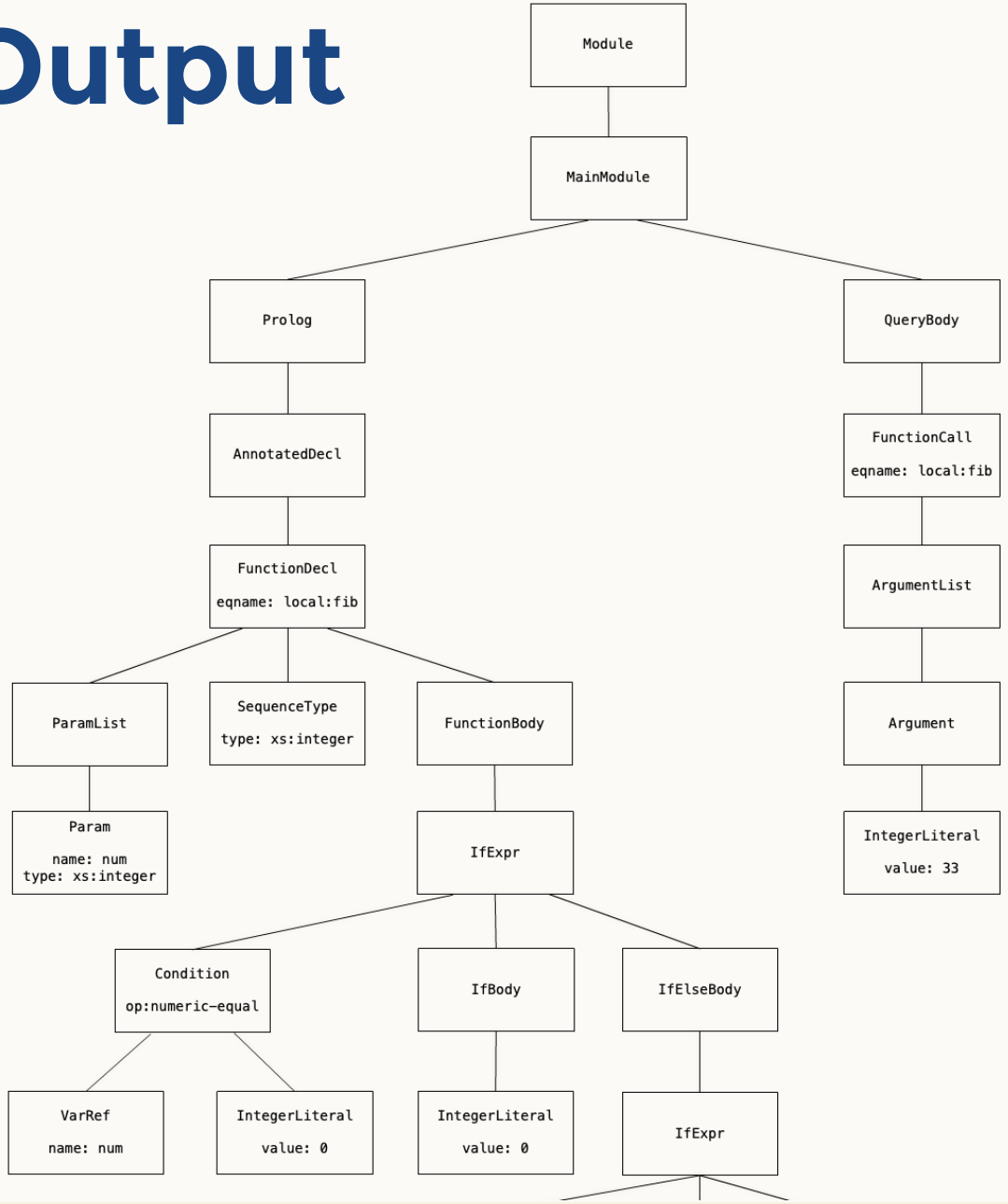
```
declare function local:fib($num as xs:integer) as xs:integer {
    if ($num = 0) then
        0
    else if ($num = 1) then
        1
    else
        local:fib($num - 1) + local:fib($num - 2)
};

local:fib(33)
```

Computing nth Fibonacci Number

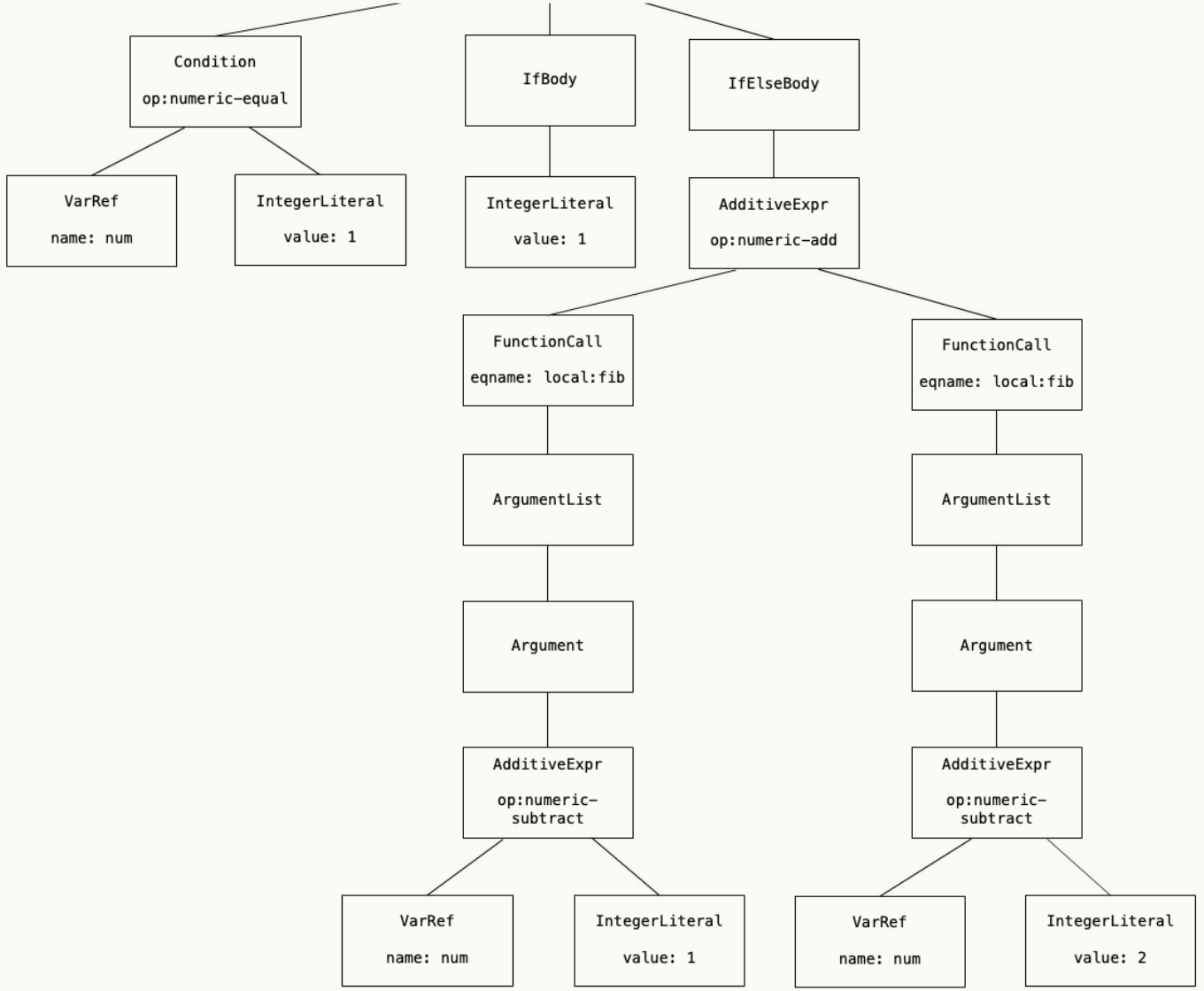
- Lexer Output

```
DeclareWord(TokenContext(Extent(0, 7), Location(1, 1)))
FucntionWord(TokenContext(Extent(8, 16), Location(1, 9)))
QName(TokenContextWithSubTokens { token_context: TokenContext(Extent(17, 26), Location(1, 18)), sub_tokens: [NCName(TokenContext(Extent(17, 22), Location(1, 18))), NCName(TokenContext(Exter
LeftParenChar(TokenContext(Extent(26, 27), Location(1, 27)))
DollarSignChar(TokenContext(Extent(27, 28), Location(1, 28)))
NCName(TokenContext(Extent(28, 31), Location(1, 29)))
NCName(TokenContext(Extent(32, 34), Location(1, 33)))
QName(TokenContextWithSubTokens { token_context: TokenContext(Extent(35, 45), Location(1, 36)), sub_tokens: [NCName(TokenContext(Extent(35, 37), Location(1, 36))), NCName(TokenContext(Exter
RightParenChar(TokenContext(Extent(45, 46), Location(1, 46)))
NCName(TokenContext(Extent(47, 49), Location(1, 48)))
QName(TokenContextWithSubTokens { token_context: TokenContext(Extent(50, 60), Location(1, 51)), sub_tokens: [NCName(TokenContext(Extent(50, 52), Location(1, 51))), NCName(TokenContext(Exter
LeftCurlyBracketChar(TokenContext(Extent(61, 62), Location(1, 62)))
IfWord(TokenContext(Extent(75, 77), Location(2, 13)))
LeftParenChar(TokenContext(Extent(78, 79), Location(2, 16)))
DollarSignChar(TokenContext(Extent(79, 80), Location(2, 17)))
NCName(TokenContext(Extent(80, 83), Location(2, 18)))
EqualsSignChar(TokenContext(Extent(84, 85), Location(2, 22)))
IntegerLiteral(TokenContextWithBigUint { token_context: TokenContext(Extent(86, 87), Location(2, 24)), value: BigUint { data: [] } })
RightParenChar(TokenContext(Extent(87, 88), Location(2, 25)))
ThenWork(TokenContext(Extent(89, 93), Location(2, 27)))
IntegerLiteral(TokenContextWithBigUint { token_context: TokenContext(Extent(110, 111), Location(3, 17)), value: BigUint { data: [] } })
ElseWord(TokenContext(Extent(124, 128), Location(4, 13)))
IfWord(TokenContext(Extent(129, 131), Location(4, 18)))
LeftParenChar(TokenContext(Extent(132, 133), Location(4, 21)))
DollarSignChar(TokenContext(Extent(133, 134), Location(4, 22)))
NCName(TokenContext(Extent(134, 137), Location(4, 23)))
EqualsSignChar(TokenContext(Extent(138, 139), Location(4, 27)))
IntegerLiteral(TokenContextWithBigUint { token_context: TokenContext(Extent(140, 141), Location(4, 29)), value: BigUint { data: [1] } })
RightParenChar(TokenContext(Extent(141, 142), Location(4, 30)))
ThenWork(TokenContext(Extent(143, 147), Location(4, 32)))
IntegerLiteral(TokenContextWithBigUint { token_context: TokenContext(Extent(164, 165), Location(5, 17)), value: BigUint { data: [1] } })
ElseWord(TokenContext(Extent(178, 182), Location(6, 13)))
QName(TokenContextWithSubTokens { token_context: TokenContext(Extent(199, 208), Location(7, 17)), sub_tokens: [NCName(TokenContext(Extent(199, 204), Location(7, 17))), NCName(TokenContext(
LeftParenChar(TokenContext(Extent(208, 209), Location(7, 26)))
DollarSignChar(TokenContext(Extent(209, 210), Location(7, 27)))
NCName(TokenContext(Extent(210, 213), Location(7, 28)))
HyphenMinusChar(TokenContext(Extent(214, 215), Location(7, 32)))
IntegerLiteral(TokenContextWithBigUint { token_context: TokenContext(Extent(216, 217), Location(7, 34)), value: BigUint { data: [1] } })
RightParenChar(TokenContext(Extent(217, 218), Location(7, 35)))
PlusSignChar(TokenContext(Extent(219, 220), Location(7, 37)))
QName(TokenContextWithSubTokens { token_context: TokenContext(Extent(221, 230), Location(7, 39)), sub_tokens: [NCName(TokenContext(Extent(221, 226), Location(7, 39))), NCName(TokenContext(
LeftParenChar(TokenContext(Extent(230, 231), Location(7, 48)))
DollarSignChar(TokenContext(Extent(231, 232), Location(7, 49)))
NCName(TokenContext(Extent(232, 235), Location(7, 50)))
HyphenMinusChar(TokenContext(Extent(236, 237), Location(7, 54)))
IntegerLiteral(TokenContextWithBigUint { token_context: TokenContext(Extent(238, 239), Location(7, 56)), value: BigUint { data: [2] } })
RightParenChar(TokenContext(Extent(239, 240), Location(7, 57)))
RightCurlyBracketChar(TokenContext(Extent(249, 250), Location(8, 9)))
SemicolonChar(TokenContext(Extent(250, 251), Location(8, 10)))
QName(TokenContextWithSubTokens { token_context: TokenContext(Extent(260, 269), Location(9, 9)), sub_tokens: [NCName(TokenContext(Extent(260, 265), Location(9, 9))), NCName(TokenContext(Ext
LeftParenChar(TokenContext(Extent(269, 270), Location(9, 18)))
IntegerLiteral(TokenContextWithBigUint { token_context: TokenContext(Extent(270, 272), Location(9, 19)), value: BigUint { data: [33] } })
RightParenChar(TokenContext(Extent(272, 273), Location(9, 21)))
```

Computing nth Fibonacci Number - Parser Output (1 of 2)



Computing nth Fibonacci Number - Parser Output (2 of 2)



Generating IR for LLVM

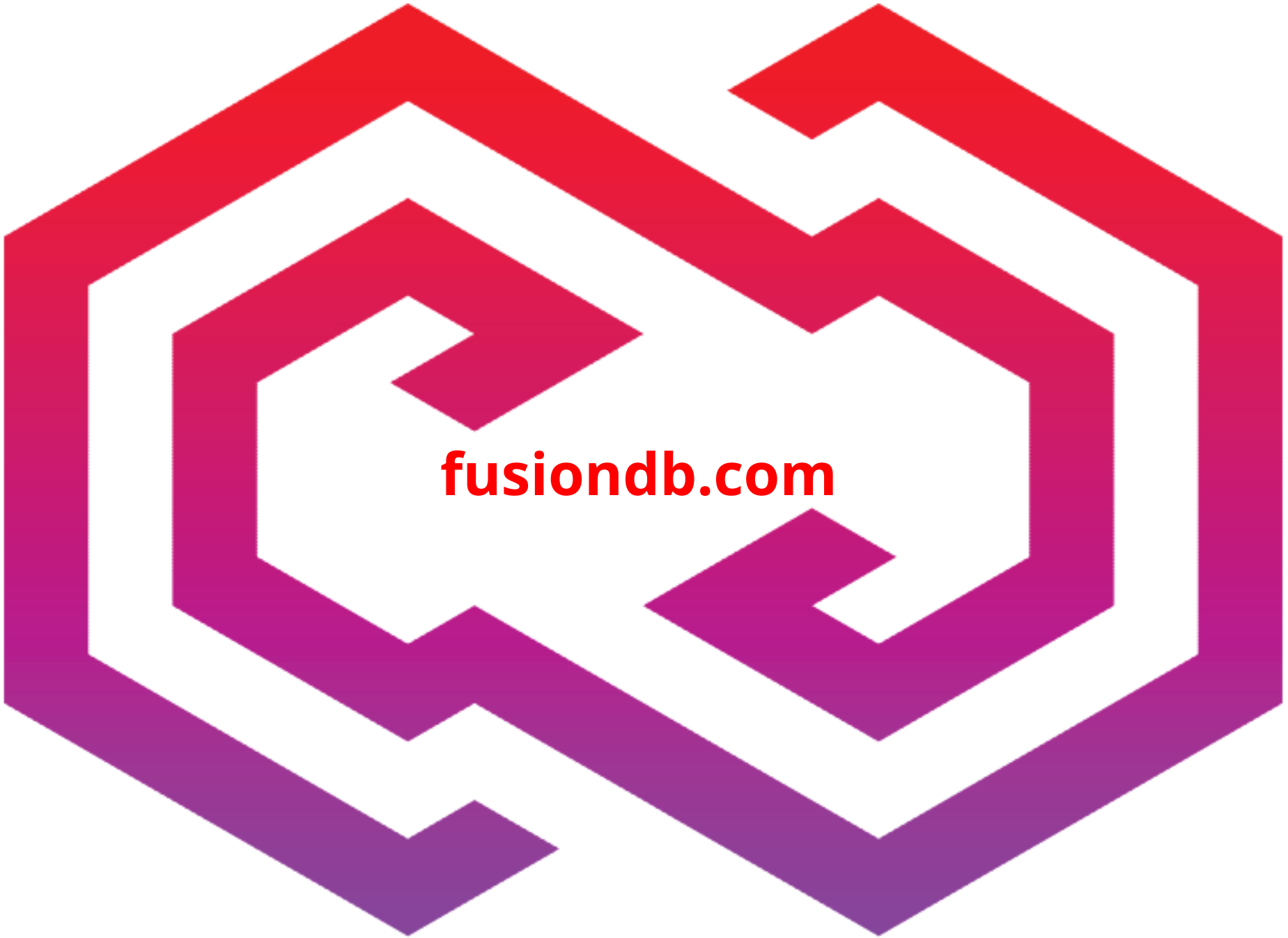
- **We use a Visitor Pattern over our AST**
- **LLVM provides an API (C++)**
  - Various bindings for other languages
    - Inkwell(s) for Rust looks promising
  - Starting point is the `IRBuilder` class
  - `IRBuilder` adds instructions to your `Module` class
  - The `NamedValues` class holds function parameters, variables, etc.
- **LLVM provides many possible Optimisations for IR**
  - See: https://llvm.org/docs/Passes.html
  - Of particular interest (for XQuery):
    - Tail Call Elimination
    - Dead Code/Argument/Type/Global/Loop Elimination

Computing nth Fibonacci Number - IR

```
;; defines our fibonacci function as local_fib
;; t1 = $num
define i64 @"local_fib"(i64 %".1")
{
local_fib_body:
  ;; t3 = ($num = 0)
  %".3" = icmp eq i64 %".1", 0
  ;; if t3 then goto local_fib_body.ifbody1 else goto local_fib_body.ifelsebody1
  br i1 %".3", label %"local_fib_body.ifbody1", label %"local_fib_body.ifelsebody1"

local_fib_body.ifbody1:
  ;; return 0
  ret i64 0

local_entry.ifelsebody1:
  ;; t4 = ($num = 1)
  %".4" = icmp eq i64 %".1", 1
  ;; if t4 then goto local_fib_body.ifbody2 else goto local_fib_body.ifelsebody2
  br i1 %".4", label %"local_fib_body.ifbody2", label %"local_fib_body.ifelsebody2"

local_fib_body.ifbody2:
  ;; return 1
  ret i64 1

local_entry.ifelsebody2:
  ;; t6 = ($num - 1)
  %".6" = sub i64 %".1", 1
  ;; t7 = ($num - 2)
  %".7" = sub i64 %".1", 2
  ;; t8 = local_fib( t6 ) ; ie local_fib( n - 1 )
  %".8" = call i64 (i64) @"local_fib"(i64 %".6")
  ;; t9 = local_fib( t7 ) ; ie local_fib( n - 2 )
  %".9" = call i64 (i64) @"local_fib"(i64 %".7")
  ;; t10 = (t8 + t9)
  %".10" = add i64 %".8", %".9"
  ;; return t10
  ret i64 %".10"
}
```

Where we are at today...

- **Still early stage!**
- **Lexer is OK**
- **Parser is in-progress**
- **We have experimental IR Generation and JIT**
- **Lots still to do...**
- **Thinking about the future...**
  - LLVM can target WebAssembly
    - Then we can run/call XQuery from Web-browser and Node.js
  - LLVM can target the GPU


fusiondb.com