# Domain-Specfic Languages by OpenResty Inc.

Implementation Strategies & Technologies

Yichun Zhang
Creator of OpenResty and Founder of OpenResty Inc.

# Who am I?

- Yichun Zhang
- Creator of the open source OpenResty project.
- CEO and cofounder of OpenResty Inc.
- Former founding team member of Cloudflare.
- Former technical expert at Taobao, Alibaba.
- Former Senior Engineer at Yahoo!.

# Why Lua and LuaJIT?

- A very efficient interpreter and a very fast tracing-based JIT compiler.
- Very small memory footprint.
- Extremely lightweight coroutines which can be used to do transparent IO multiplexing and nonblocking IO.
- Great C interoperability via FFI and the standard VM C API.
- Explicit bytecode representation which helps separating the compiling and the execution phases (like to different machines and environments).
- Easy to hack (small code base and fast VM build time).
- Great portability to many different architectures and operating systems.
- An efficient incremental mark-and-sweep garbage collector.
- Great for a common language runtime for different dynamic languages.

# Why DSLs

- Communicate intentions and knowledge more efficiently with machines.
- Domain experts can use their own languages.
- Machines have more freedom in implementations and optimizations.
- Better error handling & constraint enforcement.

# Real-World DSLs

- Regular Expressions
- BNF
- Maple & Mathematica's user languages
- SQL
- XPath
- CSS (CSS selectors)

```perl
package Point;
use Moose; # automatically turns on strict and warnings

has 'x' => (is => 'rw', isa => 'Int');
has 'y' => (is => 'rw', isa => 'Int');

sub clear {
    my $self = shift;
    $self->x(0);
    $self->y(0);
}

package Point3D;
use Moose;

extends 'Point';

has 'z' => (is => 'rw', isa => 'Int');

after 'clear' => sub {
    my $self = shift;
    $self->z(0);
};
```

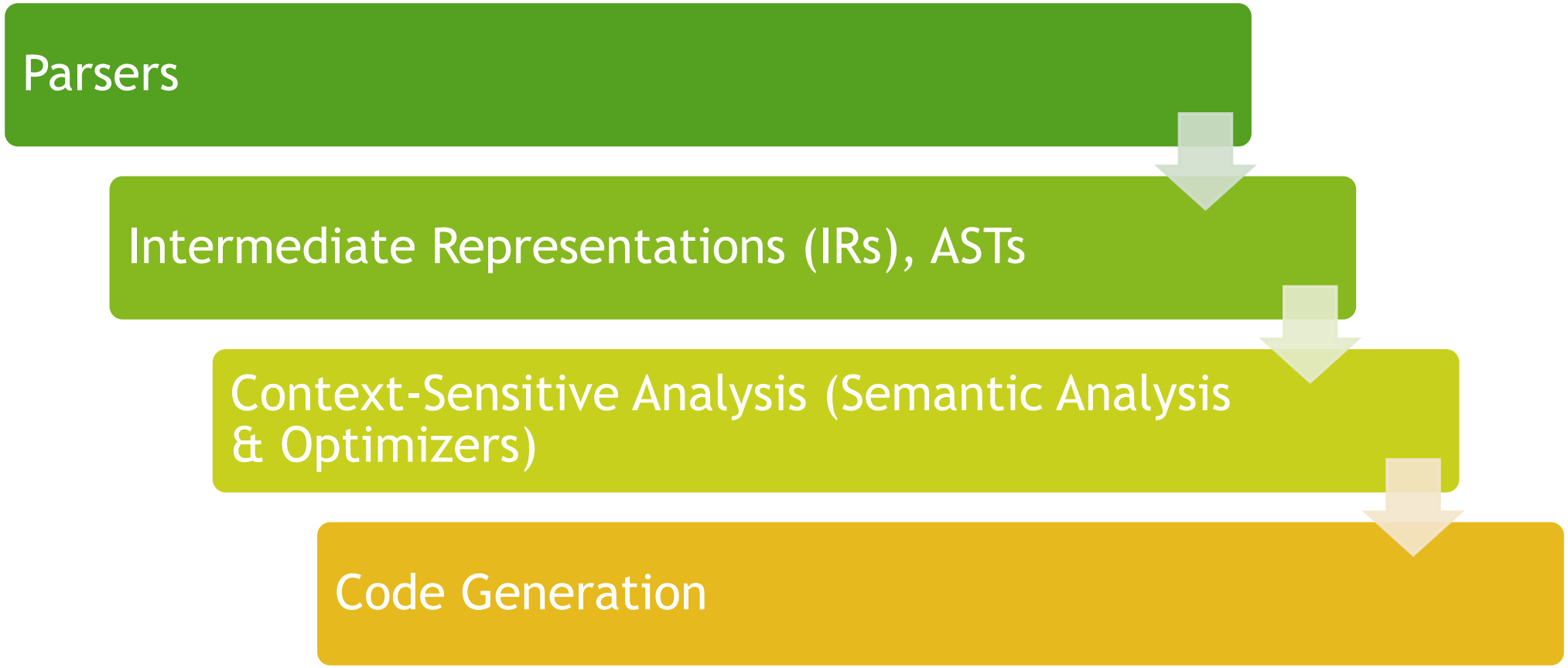
# Internal DSLs

- Parsec for Haskell
- Lpeg for Lua
- Moose for Perl

# Limitations of Internal DSLs

- Error reporting is a challenges (line numbers)
- Syntax must be compatible with the host language
- Restricted by the host language compiler & runtime
- Limited by the host language's user base

# Challenges in Implementing DSLs

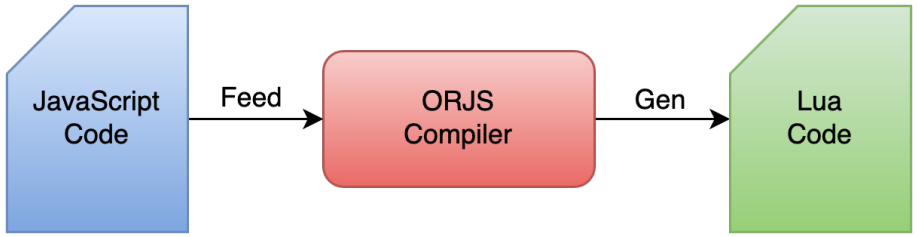


# Lua for DSLs

- Used as the source language
- Used in compilers
- Used as the target language

# ORJS: A JavaScript compiler targeting LuaJIT/OpenResty

- ORJS is still in an early phase of development and does not do any deep optimizations yet.
- For a non-recursive Fibonacci sample, ORJS is **30%** faster than V8.
- ORJS takes 70% less memory than V8 for a non-recursive Fibonacci JavaScript example (d8 is used for comparison).
- ORJS takes 80% less memory than NodeJS using the same example.
- The ORJS runtime includes the full OpenResty's nginx binary with a lot of Nginx modules compiled in.



# Enhancements to LuaJIT & OpenResty

- Perl-compatible regular expressions (PCRE)
- Expression-level `do … end` blocks
- `goto label`
- `table.new(narr, nrec)`
- `table.clone(tb)`
- `table.nkeys(tb)`
- `table.isarray(tb)`
- `table.isempty(tb)`

# Why Powerful Regular Expressions

- Look-ahead

```
/ (?! (?: continue | break | for ) \b ) ([_A-Za-z]\w*) /
/ \|\| (?! = ) /
/ \s* (?= \} | ; ) /
```

- Recursion

```
/ -- (?: \[ (=*) \[ .*? \] \2 \] \s* | [^\n]* \n? ) /
```

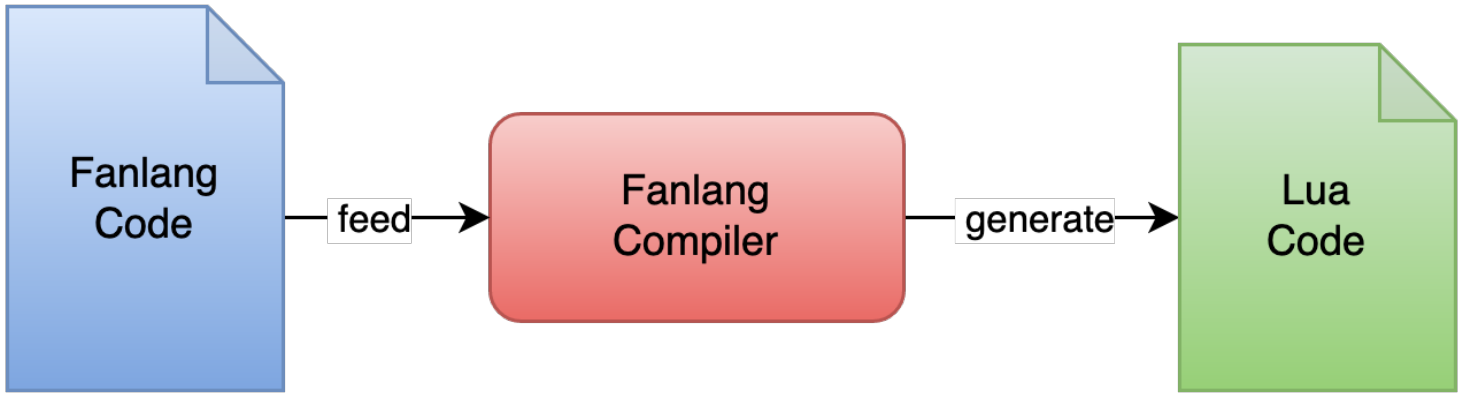
# Composable Lua Code (for codegen)

```lua
local a = (function ()
    local e = foo(
        (function ()
            local b = bar()
            if b > 3 then
                return b
            end
            return b - 1
        end)(),
        (function ()
            local res = 0
            for i = 1, 3 do
                res = res + i
            end
            return res
        end)()
    )  -- foo()
    return e * e
end)()
```

```lua
local a = do
    local e = foo(
        do
            local b = bar()
            if b > 3 then
                return b
            end
            return b - 1
        end,
        do
            local res = 0
            for i = 1, 3 do
                res = res + i
            end
            return res
        end
    )  -- foo()
    return e * e
end  -- do
```

# DSLs for Implementing DSLs

- Fanlang (by OpenResty Inc.), a Perl 6 dialect



```
grammar calc {
    expr:
      - term(s add-op) -

    add-op:
      / \s* ( [+-] ) \s* /

    term: factor(s mul-op)

    mul-op:
      / \s* ( [*\/] ) \s* /

    factor: atom(s '^')

    atom:
      | number
      | '(' expr ')'

    number:
      / ( -? \d+ (?: \. \d+ )? ) /
}

class Calc {
    has $.x;
    has $.y;
}
```

# A simple arithemetic calculator in Fanlang

- Grammar rules and Perl-compatible regular expressions are part of the language itself.

# Top-down & Bottom-up Analysis in ASTs

```
1     self.look-down: [
2         'YLang::AddrExpr' => sub ($_, $fld, $parent, $idx) {
3             my $op = .operand;
4             if $op.isa: 'YLang::DerefExpr' {
5                 my $new-node = $op.operand;
6                 replace-node $new-node, $fld, $parent, $idx;
7                 return True, $new-node;
8             }
9
10            True; # continue
11        },
12
13        'YLang::DerefExpr' => sub ($_, $fld, $parent, $idx) {
14            my $op = .operand;
15            if $op.isa: 'YLang::AddrExpr' {
16                my $new-node = $op.operand;
17                replace-node $new-node, $fld, $parent, $idx;
18                return True, $new-node;
19            }
20
21            True; # continue
22        },
23
24        ...
25    ];
```

# Edgelang

- https://doc.openresty.com/en/edge/edgelang/

```
uri("/foo"), uri-arg("n") < 1, user-agent() contains "Chrome" =>
    exit(403);
```

# Opslang

https://doc.openresty.com/en/plus/opslang/

```
$ rpmbuild --rebuild "$.file",
stream {
    out suffix /Enter pass phrase: \s*\z/ =>
        send-key("\n"),
        redo;

    found-prompt =>
        break;
},
```

# schemalang

Generate Lua API, SQL Queries,
and RESTful APIs with Data Validators

```
+builds[*]:
    -sql: created, modified
    -concrete: true
    -sort: created=desc, id=desc
    -pagesize: 20
    #commit: +int -foreign-ref=+repos/commits
    # sha:
    commit: +str
    title: +str
    pr_number?: +int
    branch?: +str
    success?: +bool
    done_time?: +int
    #target_repo: +int -foreign-ref=+repos
    target_repo: +str
    source_repo?:
        -type: +str
        -searchable: true
    committer: +str
    jobs[+]:
        job_num: +int
        desc: +str
        err_msg?: +str
        state: ["pending running", "running", "failure",
        start_running_time?: +int
        running_time?: +num
        agent_name?: +str
        round: +int
        cmds[+]:
            -type: +str
```

# Dynamic Tracing Languages

- Ylang
  https://blog.openresty.com/en/ylang-intro-part1/
  https://doc.openresty.com/en/xray/ylang/
- Ylua
  https://doc.openresty.com/en/xray/ylua/
- YSQL
  https://doc.openresty.com/en/xray/ysql/

# Questions?