Code, lint, test. In that order

or how to do a static code analysis in Crystal

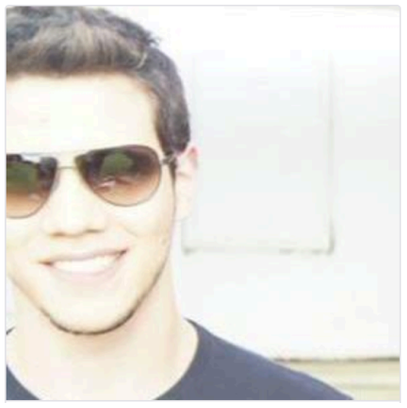
Feb, 2020 - @veelenga

About Me

awesome-crystal

https://github.com/veelenga/awesome-crystal

Blacksmoke16

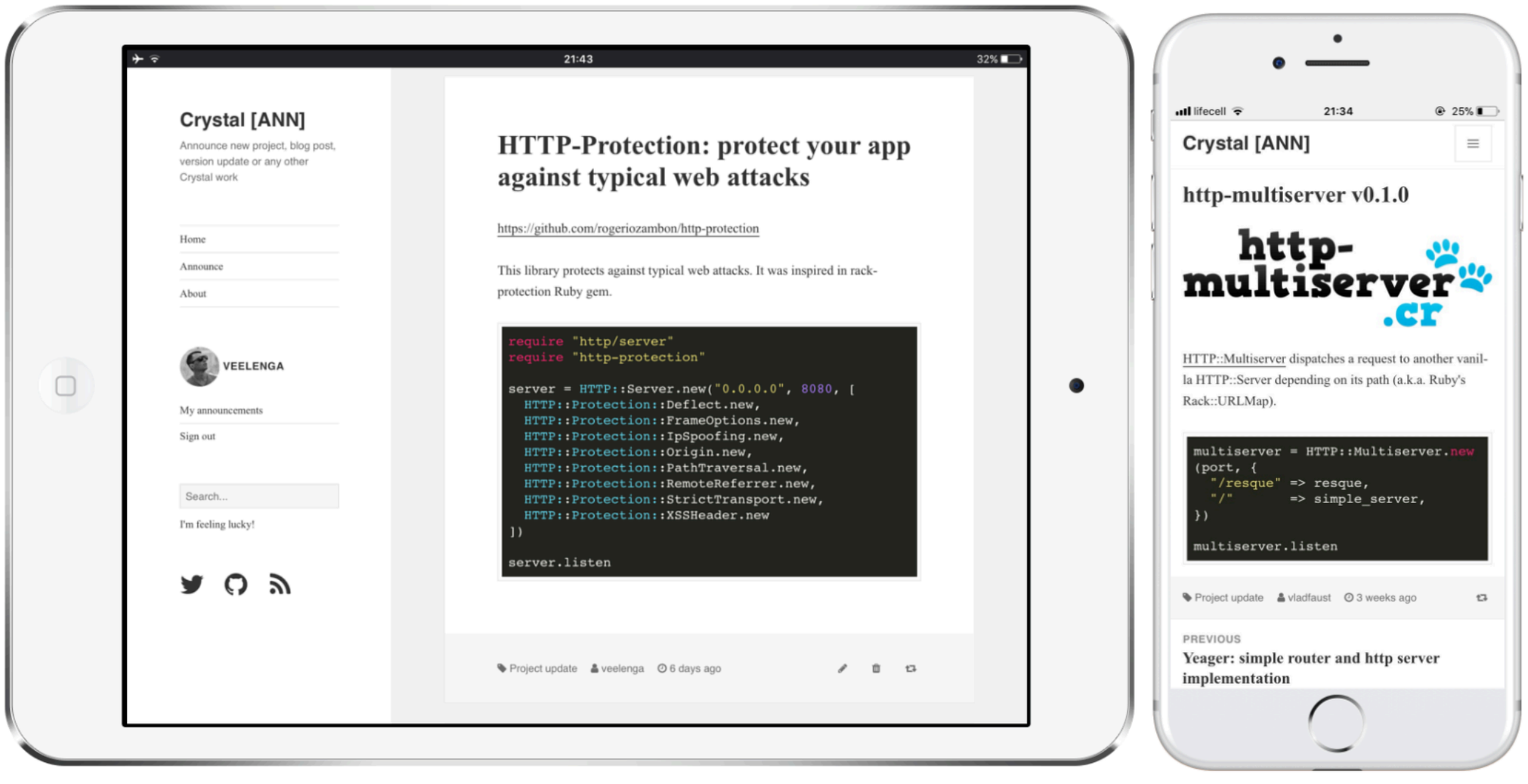


Hugo Abonizio

Stephanie Hobbs

About Me

https://crystal-ann.com/



About Me

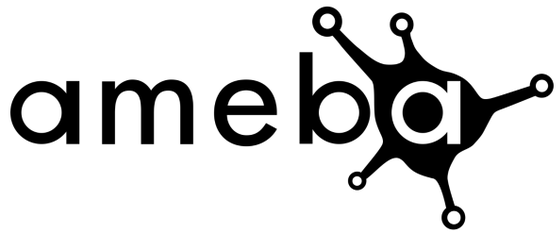
https://github.com/veelenga/lua.cr

https://github.com/crystal-community/crystal-patterns

https://github.com/crystal-community

https://github.com/amberframework

About Me



https://crystal-ameba.github.io/

https://shards.info/repos/crystal-ameba/ameba

What is Ameba intended to do?

- ✓ detect code smell
- ✓ detect potential bugs at early stage
- ✓ enforce a consistent Crystal code style
- ✓ enforce codebase to be readable and understandable

Demo time

```
amber git:master ❯ ameba
Inspecting 206 files.

.....................................................................................
.....................................................F.............F..........

src/amber/cli.cr:9:1
[W] Lint/UselessAssign: Useless assignment to variable `foo`
> foo = 1
  ^
src/amber/router/file.cr:24:7
[W] Lint/UnreachableCode: Unreachable code detected
> @size = upload.size
  ^
Finished in 693.85 milliseconds

206 inspected, 2 failures.
amber git:master ❯
0  1 〉amber                                  ~w(fast good cheap) |> Enum.take_random(2)
```

✓ running a compiled binary on a code base with intensionally added linting issues

How to use?

- ✓ a CLI binary
- ✓ an editor plugin (vim, emacs, VS code etc.)
- ✓ as a docker image
- ✓ as a development dependency
- ✓ a part of CI workflow:
  - TravisCI, CircleCI etc.
  - GitHub Actions
  - Codacy, WatchDog, Pronto etc.

Editor plugin

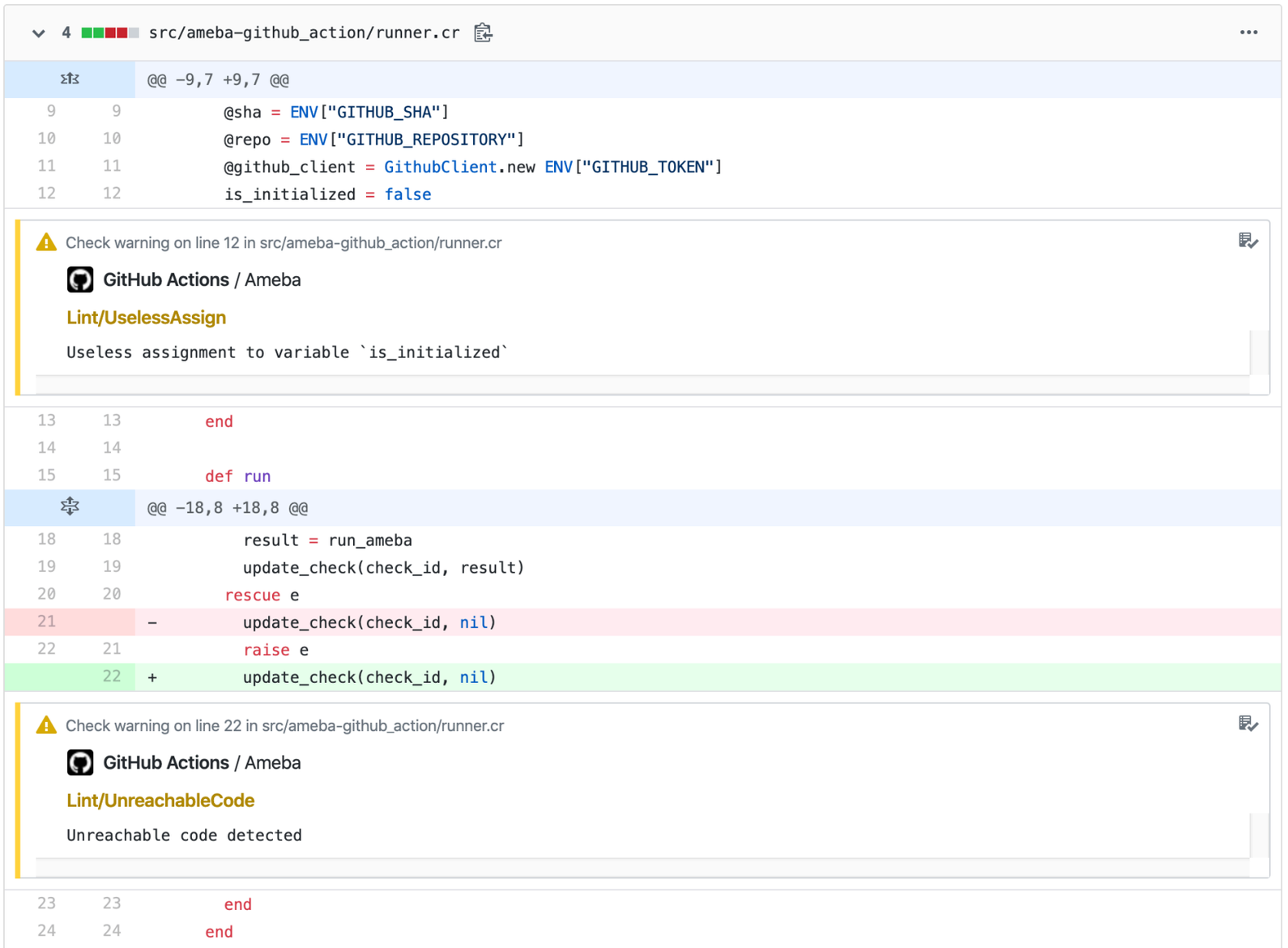
✓ running ameba through the emacs plugin on sidekiq.cr project

✓ a hint that explains what is being reported

✓ below is a scrollable mini buffer with all reported issues

```
29
28      if !svr.processors.empty?
27        logger.info "Re-enqueuing #{svr.processors.size} busy jobs"
26        svr.fetcher.bulk_requeue(svr, svr.processors.map { |p| p.job }.compact)
25      end
24
23      logger.info "Done, bye!"
22      exit(0)
21    end
20
19    def banner
18      %q{
17          m,
16          `$b
15     .ss,  $$:         .,d$
14     `$$P,d$P'    .,md$P"'
13      ,$$$$$bmmd$$$P^'
12    .d$$$$$$$$$$P'
11    $$^' `"^$$$'       ____  _     _     _     _
10    $:     ,$$:       / ___|(_) __| | ___| | _(_) __ _
9     `b     :$$        \___ \| |/ _` |/ _ \ |/ / |/ _` |
8            $$:         ___) | | (_| |  __/   <| | (_| |
7            $$         |____/|_|\__,_|\___|_|\_\_|\__, |
6         .d$$                                      |_|
5   }
4     end
3
2     def print_banner
1       if STDOut [Lint/LiteralInInterpolation] Literal value found in interpolation
123       puts "\e[#{31}m"
1         puts banner
2         puts "\e[0m"
3       end
4     end
5   end
6 end
7
8
9
3.8k  sidekiq.cr/src/sidekiq/server/cli.cr  123:13 77%                LF UTF-8  Crystal  master  3
/Users/vel/Dev/repos/sidekiq.cr/src/sidekiq/server/cli.cr:123:14: W: [Lint/LiteralInInterpola$
/Users/vel/Dev/repos/sidekiq.cr/src/sidekiq/server/cli.cr:34:55: W: [Lint/UnusedArgument] Unused argument `c`. If it's necessary, u$
/Users/vel/Dev/repos/sidekiq.cr/src/sidekiq/server/cli.cr:37:54: W: [Lint/UnusedArgument] Unused argument `c`. If it's necessary, u$
/Users/vel/Dev/repos/sidekiq.cr/src/sidekiq/web_helpers.cr:204:53: W: [Lint/ShadowingOuterLocalVar] Shadowing outer local variable $
/Users/vel/Dev/repos/sidekiq.cr/src/sidekiq/web_helpers.cr:209:40: W: [Lint/ShadowingOuterLocalVar] Shadowing outer local variable $
/Users/vel/Dev/repos/sidekiq.cr/src/sidekiq/web_helpers.cr:147:5: C: [Style/RedundantBegin] Redundant `begin` block detected
/Users/vel/Dev/repos/sidekiq.cr/src/sidekiq/api.cr:108:22: W: [Lint/ShadowingOuterLocalVar] Shadowing outer local variable `stat`
/Users/vel/Dev/repos/sidekiq.cr/src/sidekiq/api.cr:502:35: W: [Lint/ShadowingOuterLocalVar] Shadowing outer local variable `result`
/Users/vel/Dev/repos/sidekiq.cr/src/sidekiq/api.cr:646:25: W: [Lint/ShadowingOuterLocalVar] Shadowing outer local variable `x`
/Users/vel/Dev/repos/sidekiq.cr/src/sidekiq/api.cr:861:11: W: [Lint/UselessAssign] Useless assignment to variable `arr`
/Users/vel/Dev/repos/sidekiq.cr/src/sidekiq/api.cr:232:13: C: [Style/RedundantBegin] Redundant `begin` block detected
Quit
0  1 > sidekiq.cr                          ~w(fast good cheap) |> Enum.take_random(2)
```

GitHub Actions

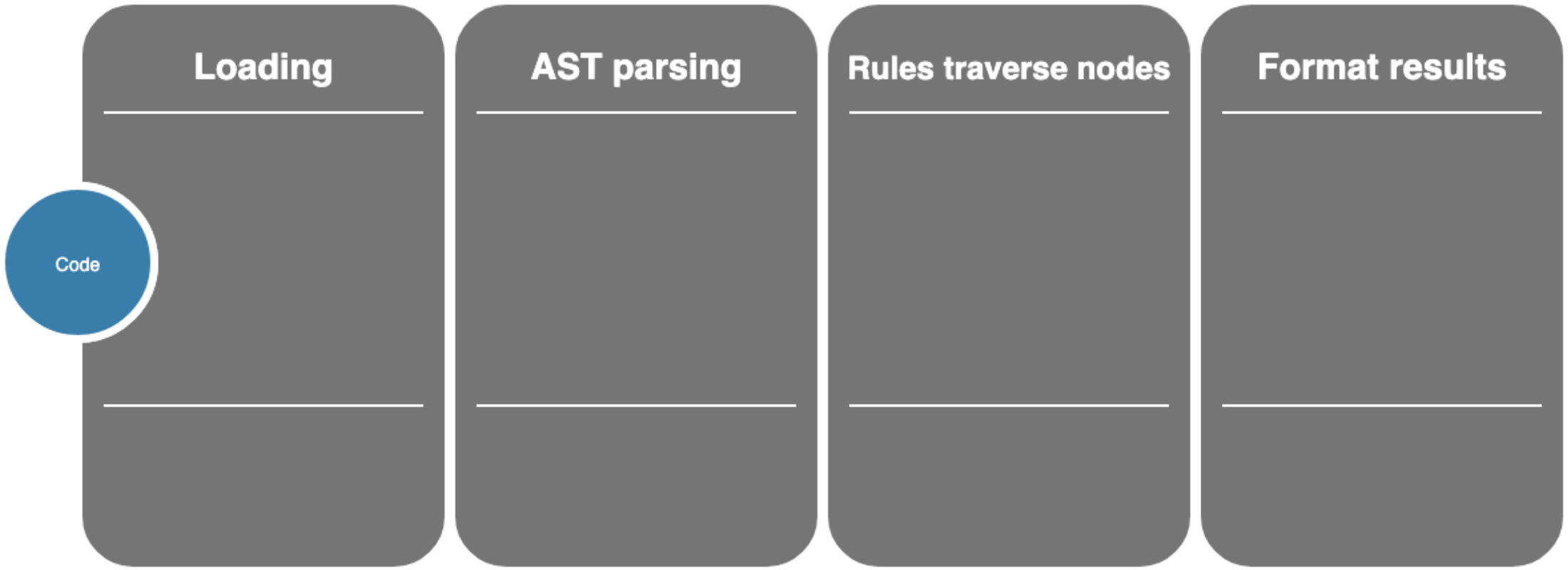


✓ it shows inlined annotations visible in the submitted pull request

How would you implement it from scratch?

- ✓ it must be able to load the source code
- ✓ it must be able to parse source code into AST nodes and traverse them
- ✓ it must be able to define specific rules which will traverse AST nodes and catch linting issues
- ✓ it must be able to format results and display linting issues and other statistics

High level picture

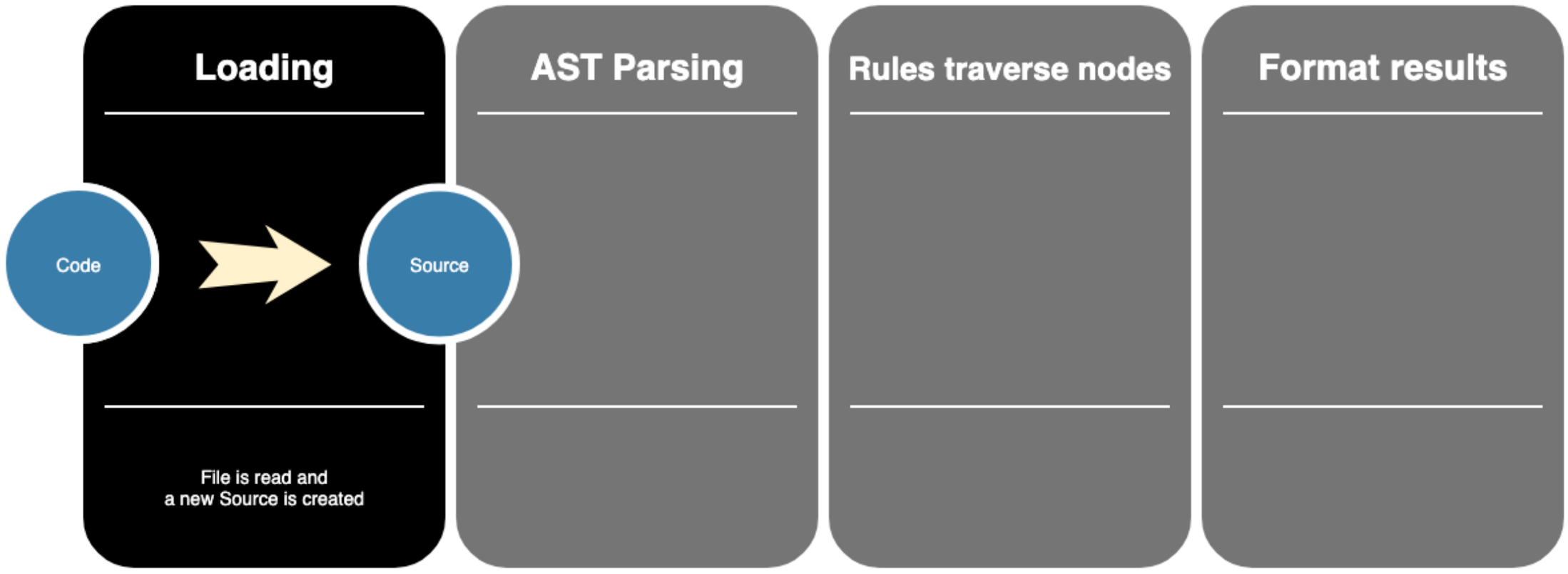


Code Loading

```
require "ameba"

path = "path_to_source_file.cr"
code = File.read(path)
source = Ameba::Source.new(code, path)

source.code #=>
  #  class Ameba::Internals
  #    getter info : Info
  #
  #    def explain
  #      puts info
  #    end
  #  end

source.issues #=> []
```

✓ code is loaded into a Ameba::Source abstraction, which utilities handy methods

Code Loading



AST Parsing

```
1  require "compiler/crystal/syntax/*"
2
3  source = "foo = 1"
4  ast_nodes = Crystal::Parser.new(source).parse
5
6  ast_nodes.class                          #=> Crystal::Assign
7  ast_nodes.as(Crystal::Assign).target  #=> Crystal::Var "foo"
8  ast_nodes.as(Crystal::Assign).value   #=> Crystal::NumberLiteral 1
```

- ✓ Crystal compiler has built in AST node parser
- ✓ Source code can be parsed into AST nodes which can be traversed
- ✓ AST nodes have properties which can hold other nodes

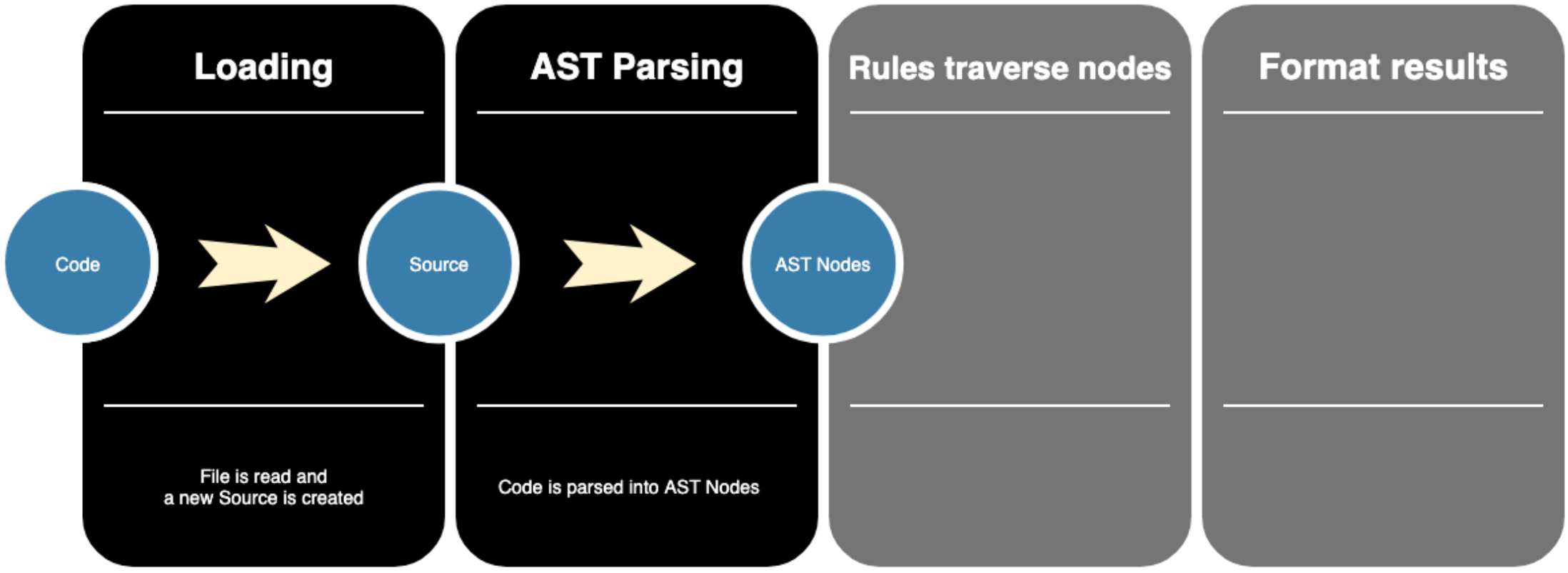
AST Parsing

```
source.ast #=>
  # class Ameba::Internals
  #   getter(info : Info)
  #   def explain
  #     puts(info)
  #   end
  # end

source.ast.class #=> Crystal::ClassDef
```

✓ a source can be converted to AST nodes which need to be traversed

✓ AST nodes have properties and children

AST Parsing


Loading
AST Parsing
Rules traverse nodes
Format results
Code
Source
AST Nodes
File is read and
a new Source is created
Code is parsed into AST Nodes

Traversing nodes

```
1   require "compiler/crystal/syntax/*"
2
3   class MyVisitor < Crystal::Visitor
4     def visit(node : Crystal::Var)
5       node.name #=> foo
6     end
7
8     def visit(node : Crystal::ASTNode)
9       true      #=> visit all other nodes
10    end
11  end
12
13  ast_nodes = Crystal::Parser.parse("foo = 1")
14  MyVisitor.new.accept(ast_nodes)
```

✓ Available projects:

https://github.com/bcardiff/crystal-ast-helper
https://github.com/arcage/ast_viewer.cr

Traversing nodes

```
1   module Ameba::Rule
2     # A rule that disallows calls to `puts` method.
3     struct NoPuts < Base
4
5       def test(source : Source)
6         # TODO: traverse source.ast, find all Crystal::Call
7         #   AST nodes with name "puts"
8         end
9       end
10    end
```

✓ "test" method is an entry point for the rule

✓ it accepts a source, inspect it and add issues

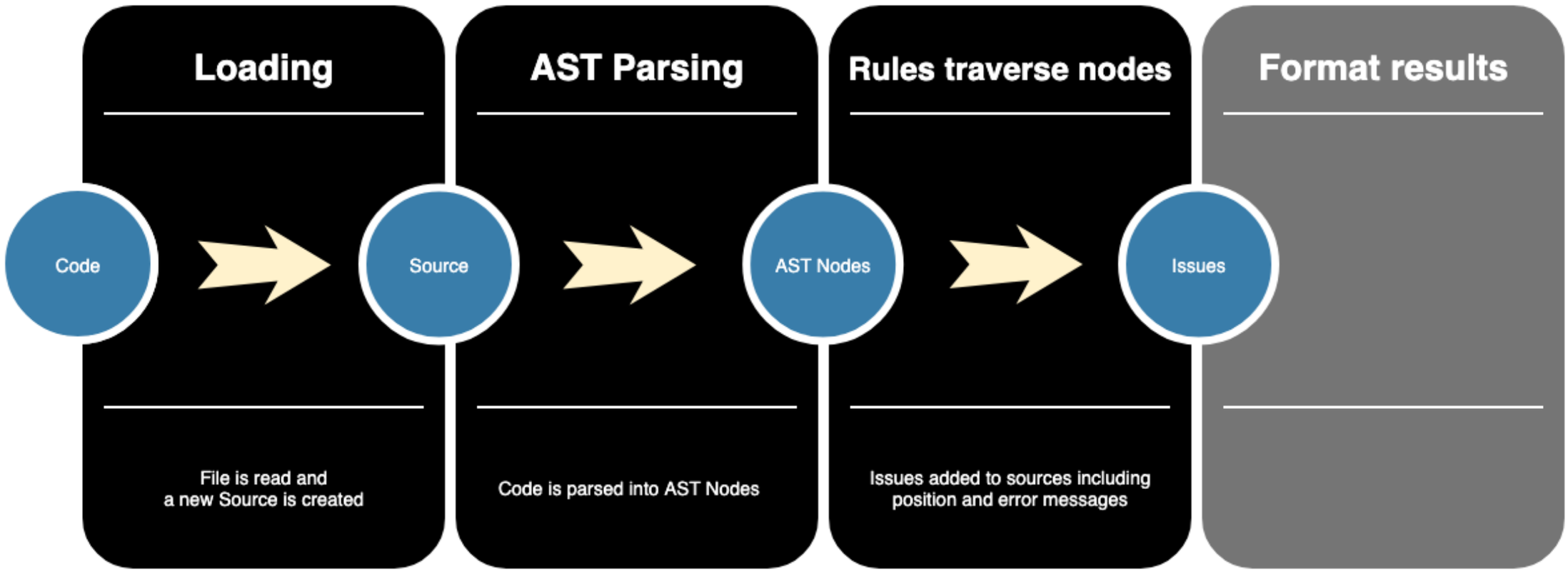
Traversing nodes

```
1   module Ameba::Rule
2     struct NoPuts < Base
3       def test(source)
4         AST::NodeVisitor.new self, source
5       end
6
7       def test(source, node : Crystal::Call)
8         return unless node.name == "puts"
9         source.add_issue(self, node, "puts method is called")
10      end
11    end
12  end
```

✓ rules use method overloading for node filtering

✓ rules add issues to sources, attaching node (position) and a message

Traversing nodes



Format results

```
formatter = Ameba::Formatter::JSONFormatter.new(STDOUT)
formatter.started([source])
formatter.source_finished(source)
formatter.finished([source])
```

✓ Different kind of formatters (dot, json, flycheck, todo etc)

✓ Ability to send results to different IOs

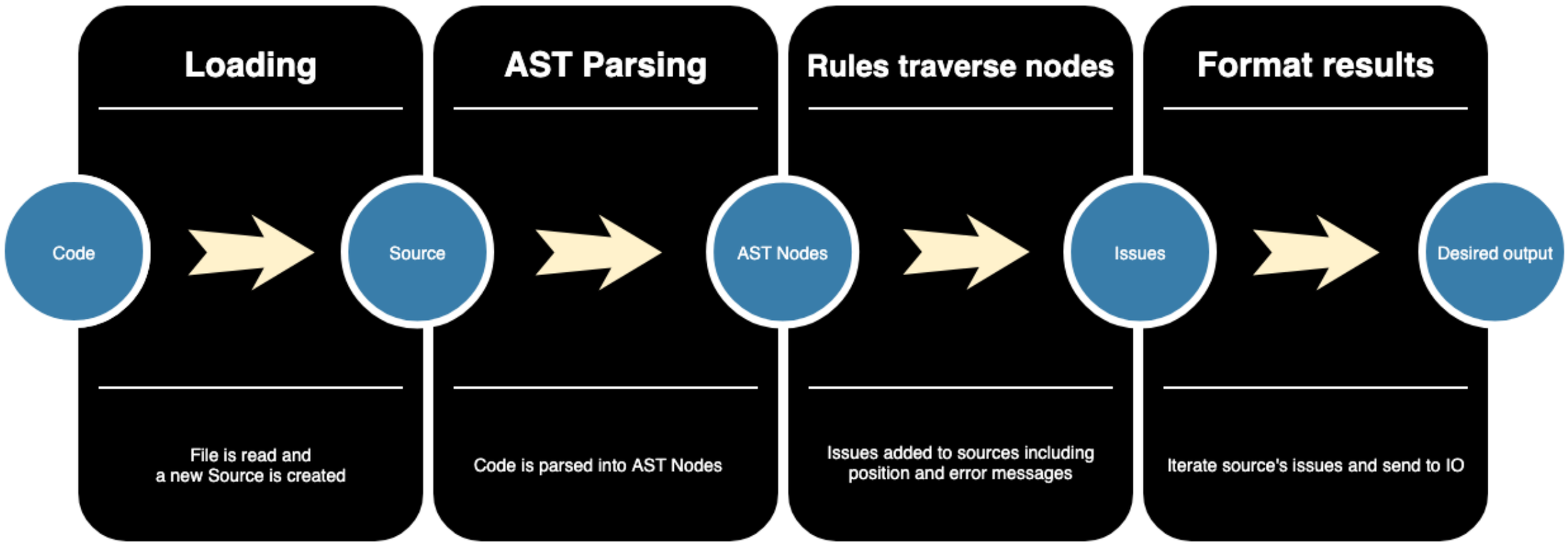
Format results

```crystal
class Ameba::Internals
  getter info : Info

  def explain
    puts info
  end
end
```



```json
{
    "metadata": {
        "ameba_version": "0.11.0",
        "crystal_version": "0.32.0"
    },
    "sources": [
        {
            "issues": [
                {
                    "end_location": {
                        "column": 13,
                        "line": 5
                    },
                    "location": {
                        "column": 5,
                        "line": 5
                    },
                    "message": "puts method is called",
                    "rule_name": "NoPuts",
                    "severity": "Convention"
                }
            ],
            "path": "path_to_source_file.cr"
        }
    ],
    "summary": {
        "issues_count": 1,
        "target_sources_count": 1
    }
}
```

Format results



Runner definition

```
1   class Runner
2     # ...
3
4     def run
5       @sources.each { |source| inspect(source) }
6     end
7
8     def inspect(source)
9       @formatter.source_started source
10      @rules.each { |rule| rule.test source }
11      @formatter.source_finished source
12    end
13  end
```

✓ iterates over sources and tests rules

Running in parallel

```
1   channels = @sources.map { Channel(Nil).new }
2
3   @sources.each_with_index do |source, idx|
4     channel = channels[idx]
5     spawn do
6       inspect(source) # run full set of rules
7       channel.send(nil)
8     end
9   end
10
11  channels.each { |c| c.receive }
```

✓ spawn each source to its own channel

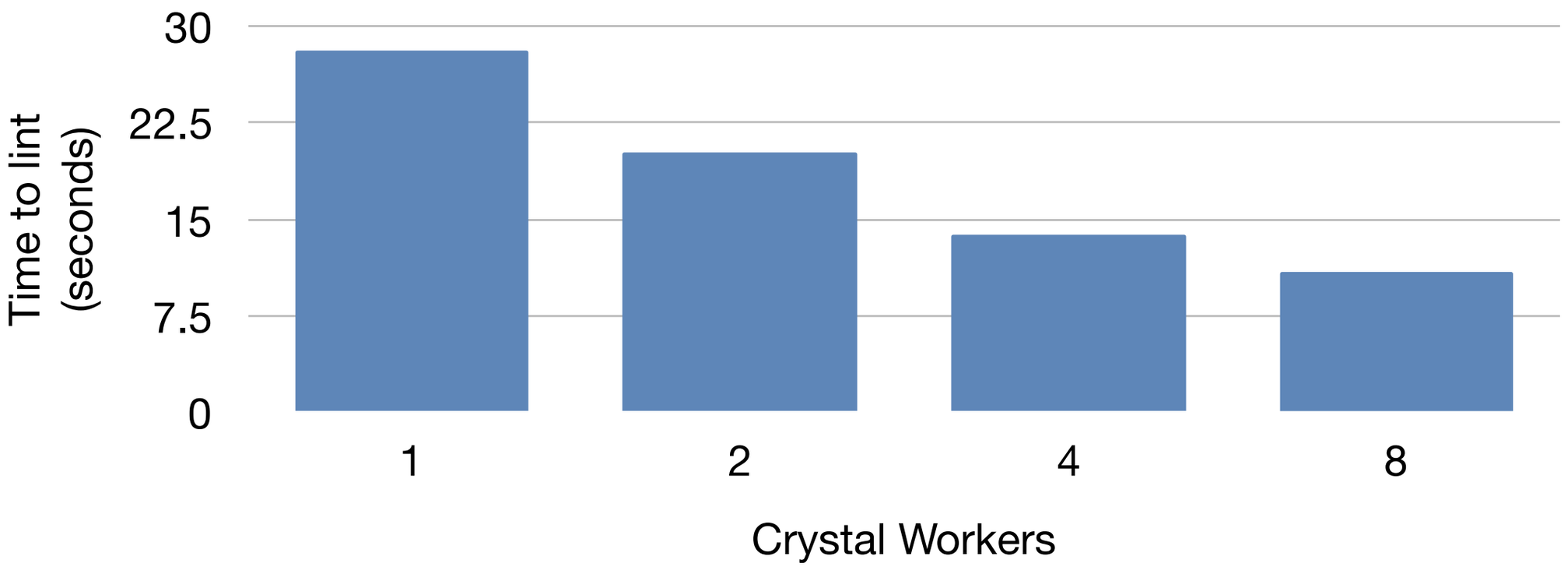
✓ waits all channels to finish

Running in parallel

```
$ crystal build src/cli.cr -Dpreview_mt -o bin/ameba
$ sudo make install

# run on Crystal repo
$ time CRYSTAL_WORKERS=1 ameba --silent
$ time CRYSTAL_WORKERS=2 ameba --silent
$ time CRYSTAL_WORKERS=4 ameba --silent
$ time CRYSTAL_WORKERS=8 ameba --silent
```

Running in parallel

```
$ crystal build src/cli.cr -Dpreview_mt -o bin/ameba
$ sudo make install

# run on Crystal repo
$ time CRYSTAL_WORKERS=1 ameba --silent # 27.78s user 0.34s system 99% cpu 28.116 total
$ time CRYSTAL_WORKERS=2 ameba --silent # 29.54s user 0.22s system 147% cpu 20.184 total
$ time CRYSTAL_WORKERS=4 ameba --silent # 30.90s user 0.28s system 226% cpu 13.742 total
$ time CRYSTAL_WORKERS=8 ameba --silent # 44.67s user 0.44s system 413% cpu 10.900 total
```



Limitations and challenges

1. Some AST nodes are transformed during parsing

```
source = %(
  class Animal
    def initialize(@name : String)
    end
  end
)

Crystal::Parser.parse(source) #=>
  # class Animal
  #   def initialize(name : String)
  #     @name = name
  #   end
  # end
```

X some cases cannot be analyzed using AST Parser

X a risk to drop some rules on future Crystal releases

Limitations and challenges

2. Code auto correction

✓ use Crystal's transformer

```
1   require "compiler/crystal/syntax"
2
3   class Charify < Crystal::Transformer
4     def transform(node : Crystal::NumberLiteral)
5       Crystal::CharLiteral.new(node.value.to_i.chr)
6     end
7   end
```

X node can already be changed by the AST parser

X conflicts with other rules

X conflicts with formatter

Limitations and challenges

3. Semantic analysis

✓ ability to have types opens much more possibilities

anykeyh commented 25 days ago

There's a trick in crystal which can confuse ruby developers: `'` stands for character definition while `"` stands for string definition.

It can lead to confusing non-working code like theses examples below:

```
x = "b"
puts "Hello world" if x == 'b' #Will be false
```

```
x = "hello"
puts "Hello world" if x[1] == "b" #Will be false too
```

X complexity

X include Ameba sources to compilation process

References

- ✓ https://github.com/crystal-ameba
- ✓ https://github.com/crystal-ameba/ameba/wiki (Roadmap)
- ✓ https://crystal-ameba.github.io/blog/

- ✓ https://crystal-lang.org/reference/conventions/coding_style.html
- ✓ https://github.com/crystal-lang/crystal/tree/master/samples/compiler
- ✓ https://github.com/crystal-lang/crystal/wiki/Compiler-internals

Credits

- ✓ https://crystal-lang.org/

- ✓ https://rubocop.readthedocs.io/en/latest/
- ✓ https://github.com/rrrene/credo/
- ✓ https://github.com/lpil/dogma

- ✓ https://kun.ai/

Thank you