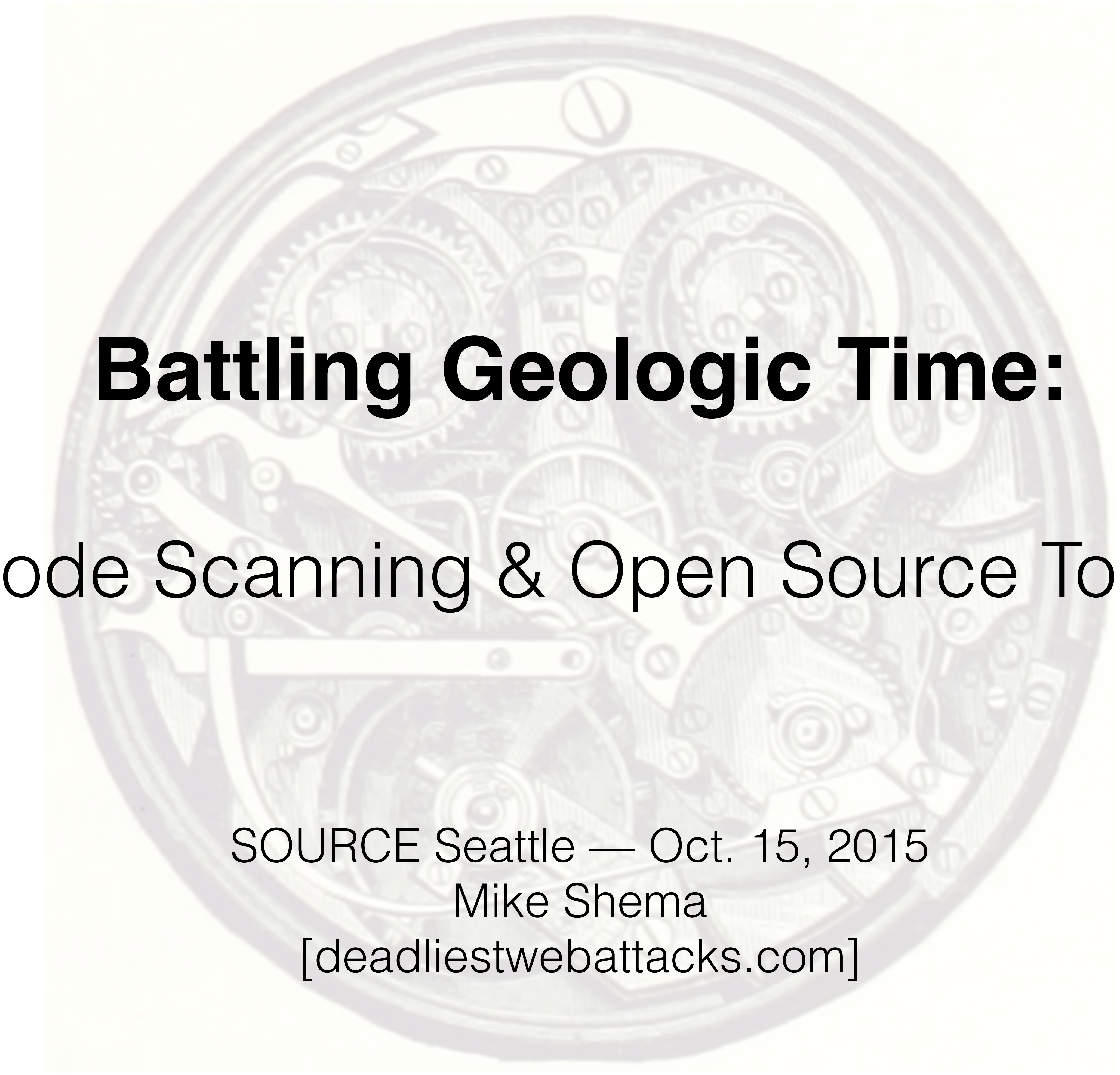


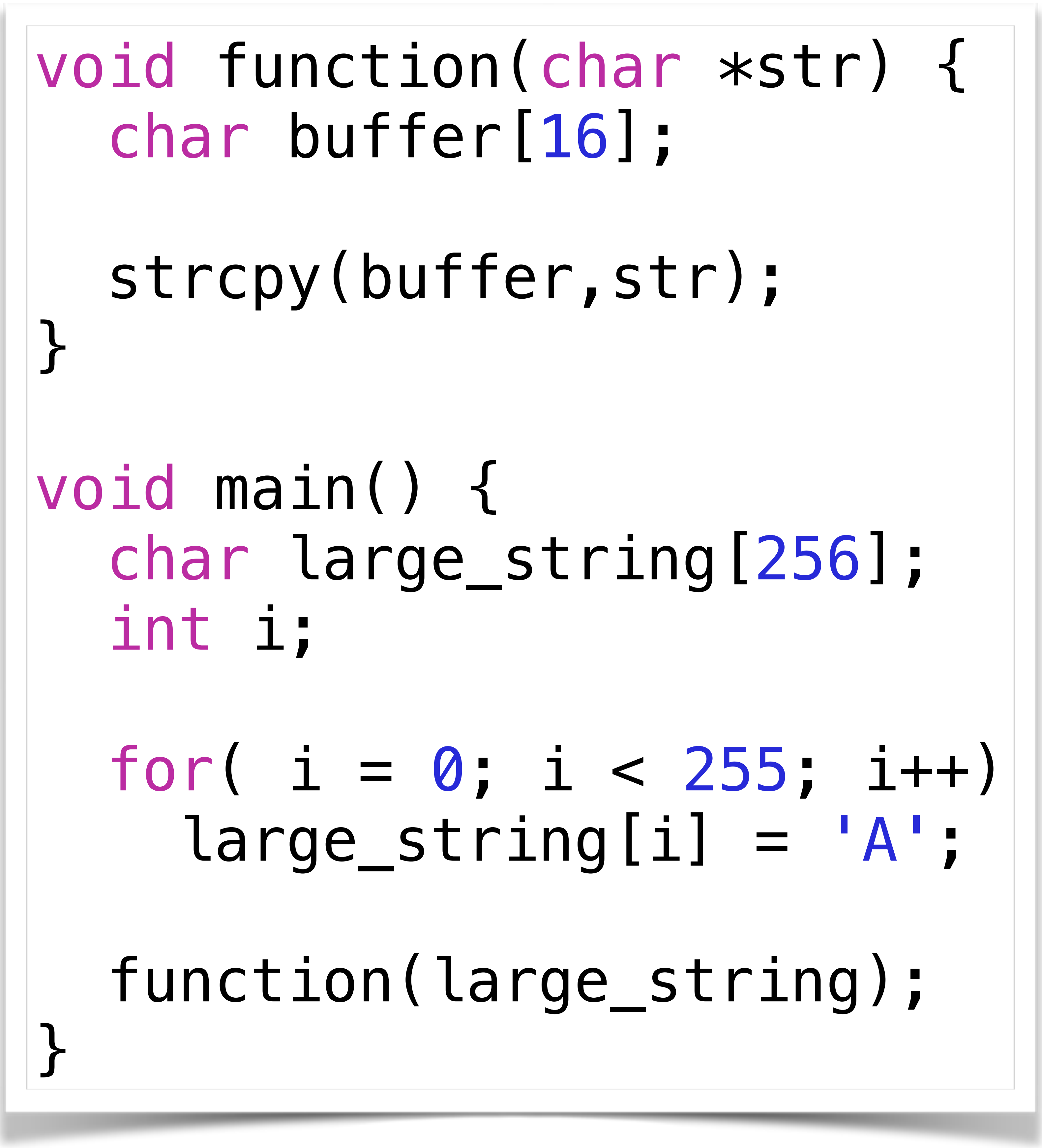
Battling Geologic Time:

Code Scanning & Open Source Tools

SOURCE Seattle — Oct. 15, 2015
Mike Shema
[deadliestwebattacks.com]

Phrack 49

“Smashing…”



```c
void function(char *str) {
  char buffer[16];

  strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
```

> 10 Lines of Code

Windows XP ~**45M**

Bash 4.3 ~**99K** C

OpenSSL trunk ~**240K** C & ~**74K** ASM (**!?**)

Firefox trunk ~**4.5M** C++ & ~**2.8M** JavaScript

Linux 2.6 ~**6.3M** C … v3.18 ~**10M** C … v4.2 ~**11M** C

http://windows.microsoft.com/en-US/windows/history#T1=era6
https://github.com/AlDanial/cloc

From Trivia to Toil

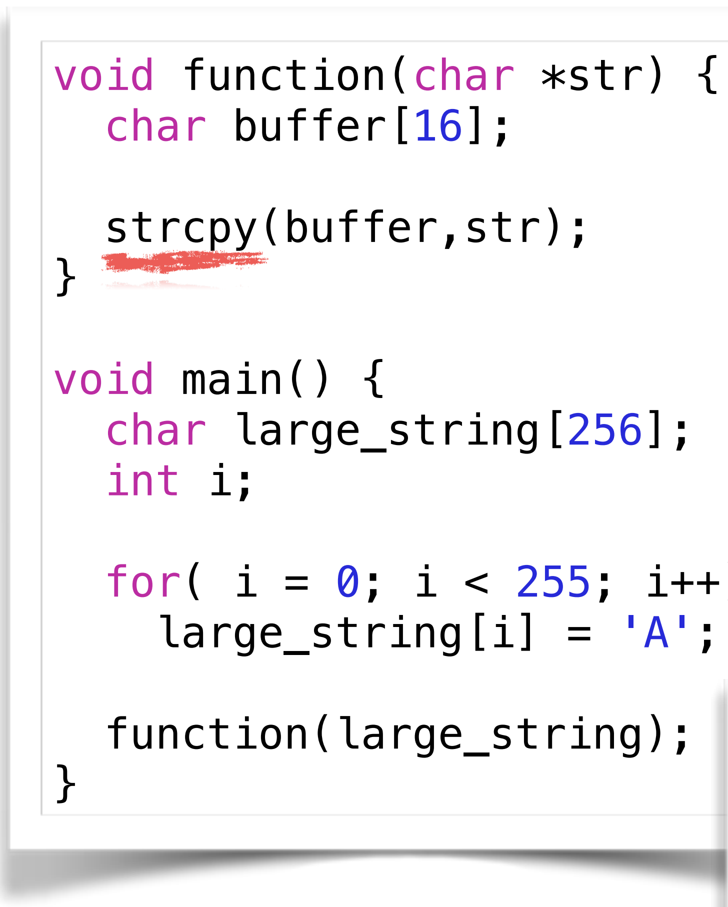
Focus on quality of code, not quantity.

Rate of change is as crucial as number of lines.

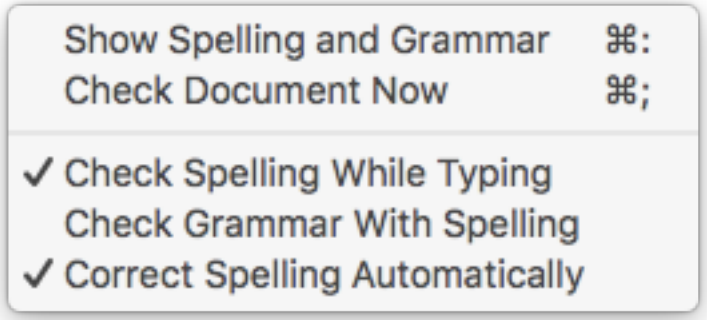
Build code with SDLC, build habits with security feedback.

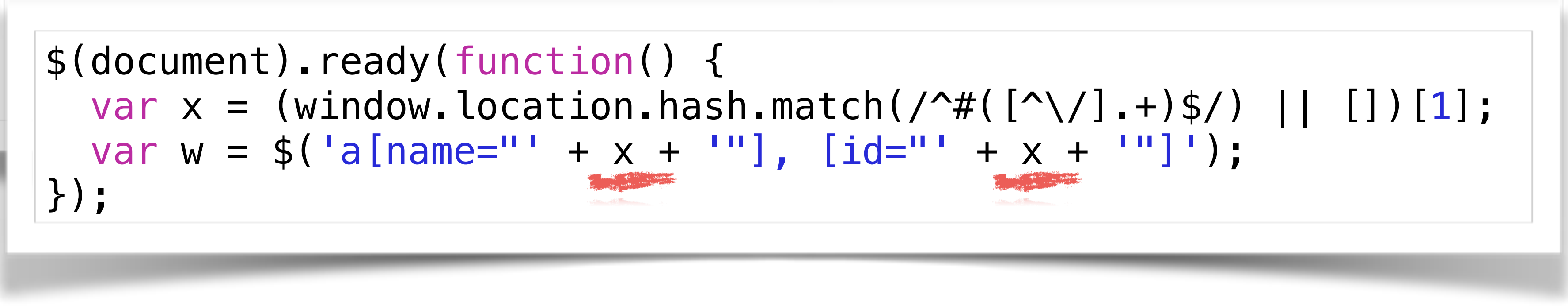
Apply feedback to rescan old code with new patterns.

The Red Squiggle is sort of Nigh

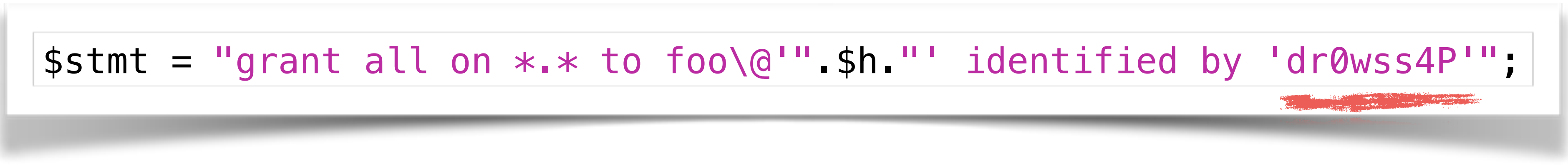


```
void function(char *str) {
  char buffer[16];

  strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
```


Show Spelling and Grammar ⌘:
Check Document Now ⌘;
✓ Check Spelling While Typing
Check Grammar With Spelling
✓ Correct Spelling Automatically

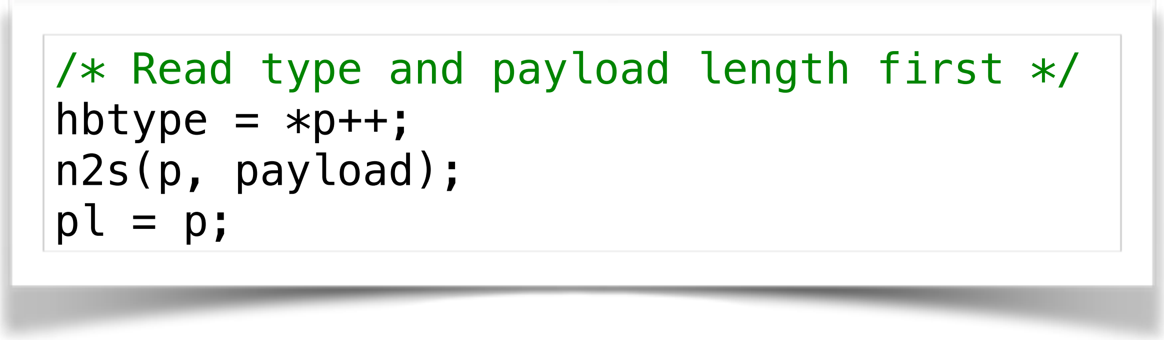


```
$(document).ready(function() {
  var x = (window.location.hash.match(/^#([^\/].+)$/) || [])[1];
  var w = $('a[name="' + x + '"], [id="' + x + '"]');
});
```

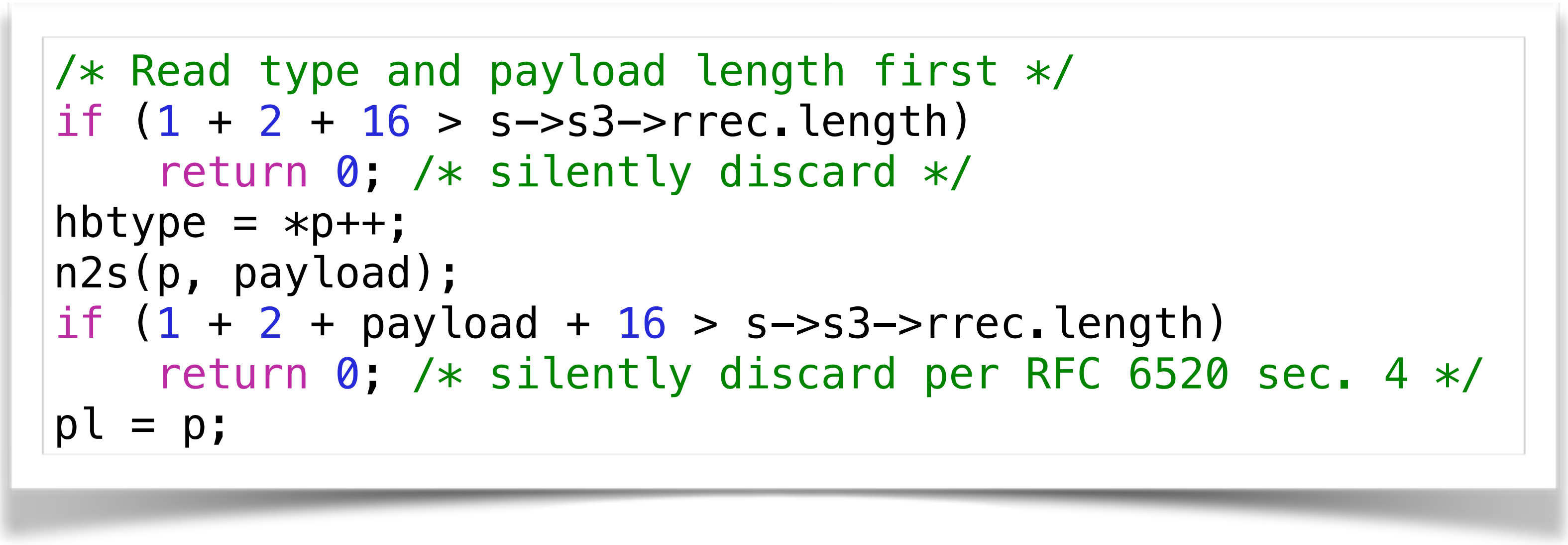


```
$stmt = "grant all on *.* to foo\@'".$h."' identified by 'dr0wss4P'";
```

The End is still Nigh



```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
```



```
/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
    return 0; /* silently discard */
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;
```

Scan the Scannable

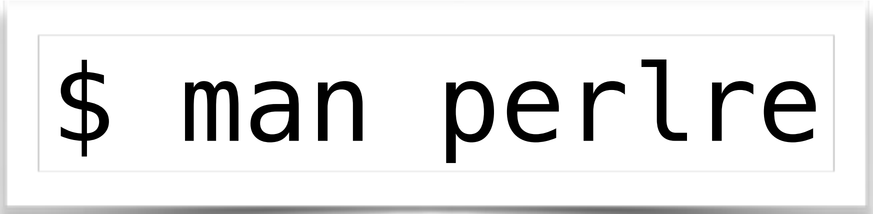
API use & misuse (cppcheck, valgrind)

Style violations (warnings as errors)

Syntax, semantics (one translation unit)

Concrete vs. Context

Regexes



```
$ man perlre
```

Fig. 22.

Practical Extraction

PCRE C API (http://pcre.org)

pcre_study

PCRE2!

JIT

python-pcre (https://github.com/awahlig/python-pcre)

c++11 <regex> (http://en.cppreference.com/w/cpp/regex)

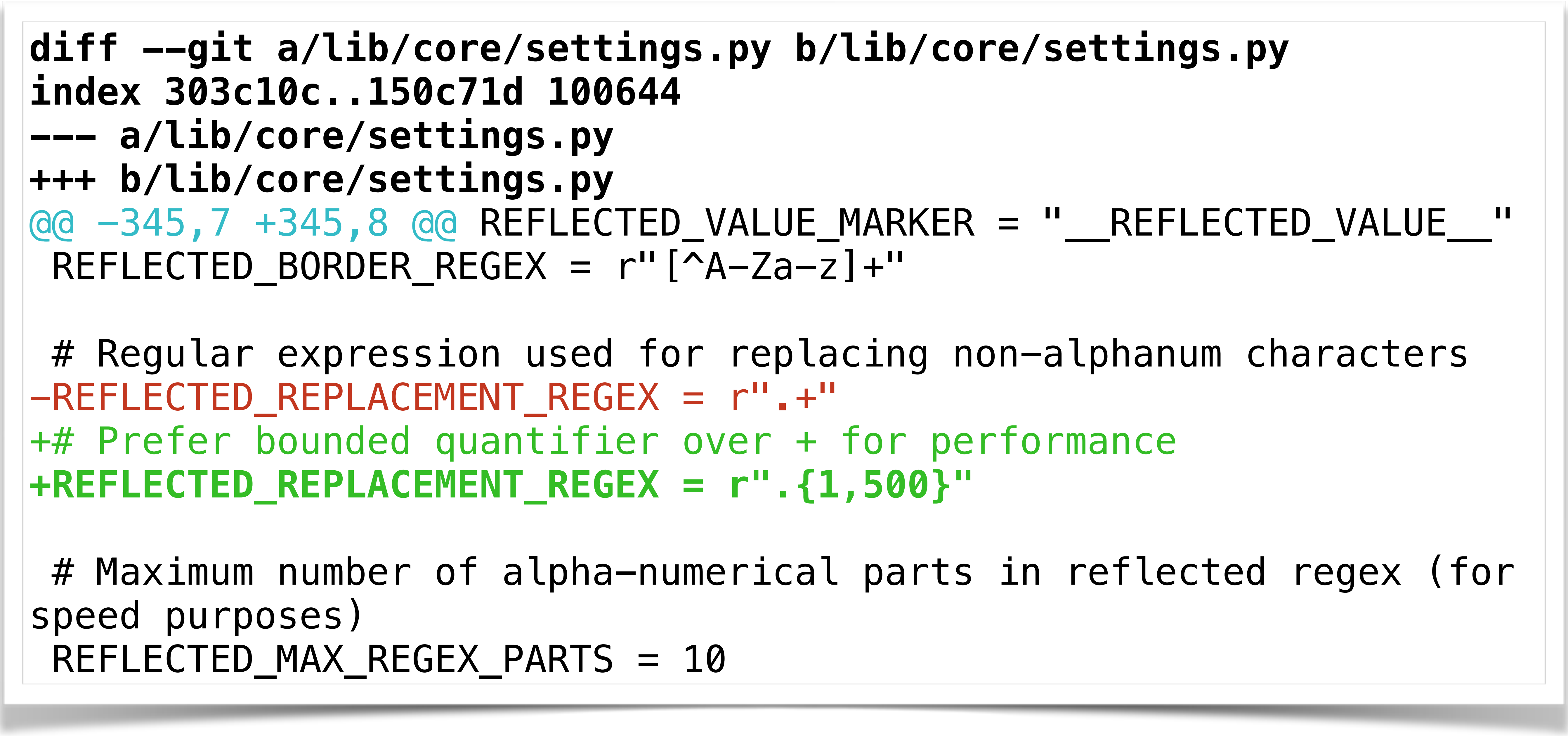
Tweaks

Work in a single character encoding (UTF-8).

Explicit quantifiers avoid pathological cases.

(?|(Branch reset)|with|(sub)-(groups)) optimizes a single pattern vs. loops over many small ones.

Quantified Performance



```
diff --git a/lib/core/settings.py b/lib/core/settings.py
index 303c10c..150c71d 100644
--- a/lib/core/settings.py
+++ b/lib/core/settings.py
@@ -345,7 +345,8 @@ REFLECTED_VALUE_MARKER = "__REFLECTED_VALUE__"
 REFLECTED_BORDER_REGEX = r"[^A-Za-z]+"

 # Regular expression used for replacing non-alphanum characters
-REFLECTED_REPLACEMENT_REGEX = r".+"
+# Prefer bounded quantifier over + for performance
+REFLECTED_REPLACEMENT_REGEX = r".{1,500}"

 # Maximum number of alpha-numerical parts in reflected regex (for speed purposes)
 REFLECTED_MAX_REGEX_PARTS = 10
```

https://sqlmap.org

Patterns vs. Parsing

Accommodating valid syntax, whitespace

```
foo\s*\([\s\r\n]*[^)]*\)
```

Handling quoted strings, balancing quotes, and “escape \”sequences\\\””

```
['"].*?['"]
```

```
'[^']*'|"[^"]*"
```

Two Wrongs, Not Right

How to annoy a security person:

Expect to build SQL with string concatenation

How to annoy a developer:

Expect to find SQL injection with regexes without false positives

All Benefit (Some Bother)

Comprehensive (yet complex)

Fast (when bounded)

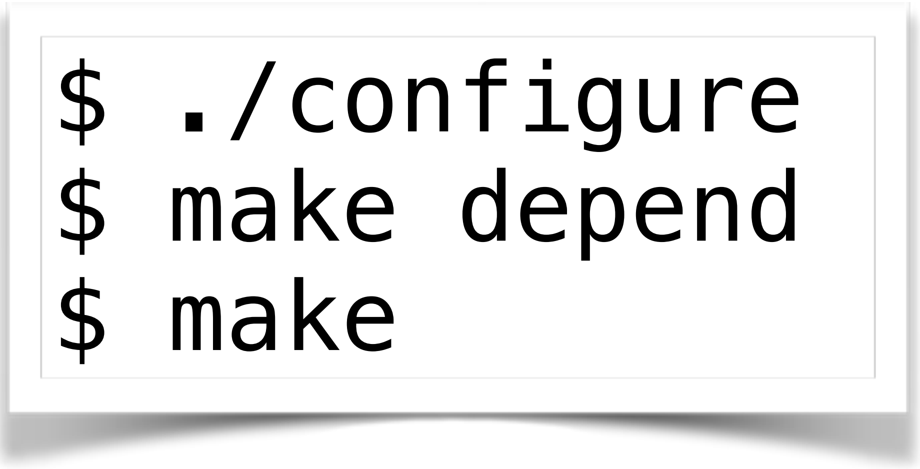
Good enough (except when they aren’t)

Patterns aren’t always parsers

https://github.com/facebook/**pfff**

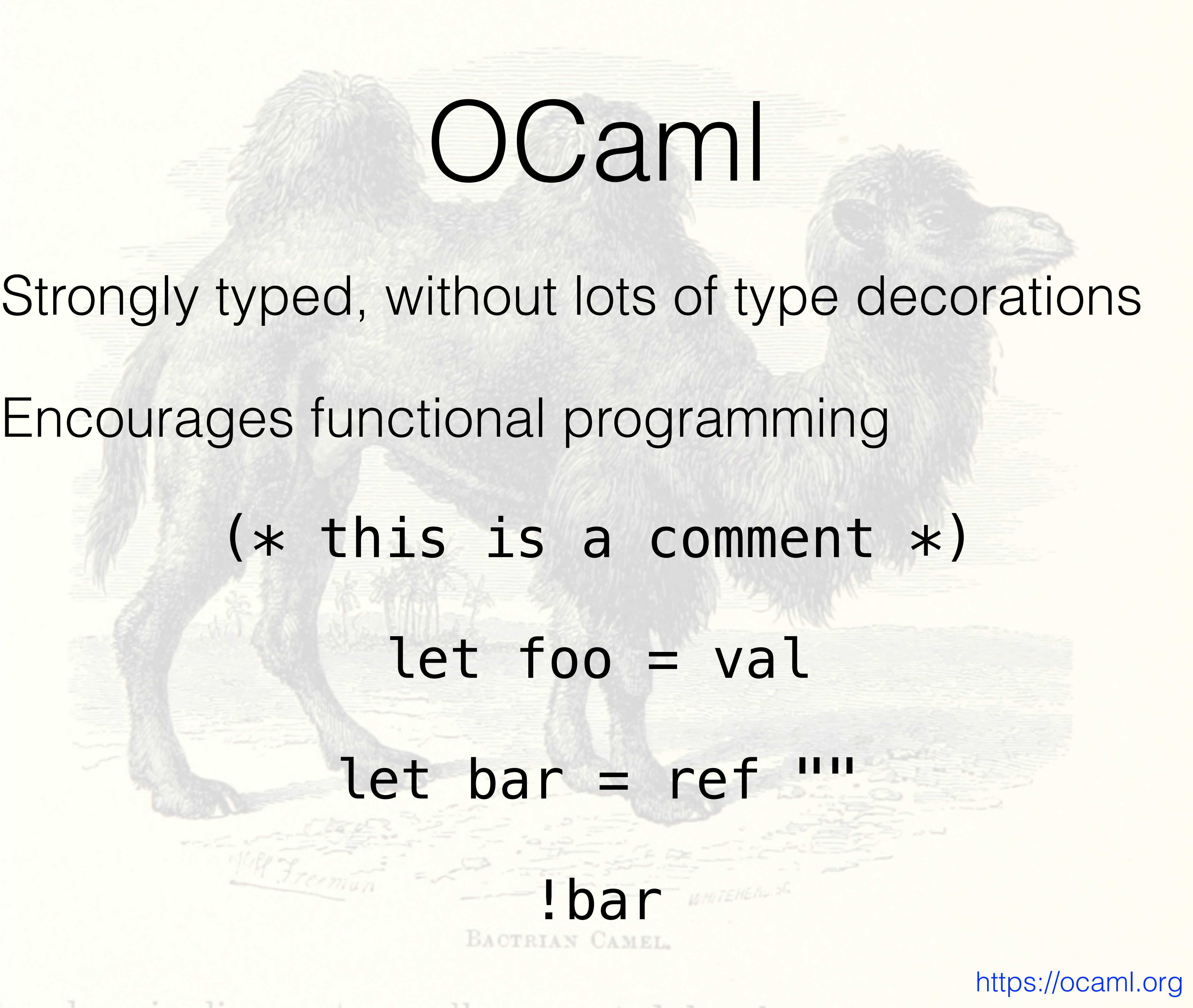
Programmable Foundation from Facebook

Group of tools built on a shared code base for conducting various types of code analysis against various languages.



```
$ ./configure
$ make depend
$ make
```

OCaml



Strongly typed, without lots of type decorations

Encourages functional programming

```
(* this is a comment *)

let foo = val

let bar = ref ""

!bar
```

https://ocaml.org

OCaml Match

```
let lang = ref "php"

let create_ast file =
  match !lang with
  | ("c" | "c++") ->
    ...
  | "java" ->
    ...
  | "js" ->
    ...
  | _ ->
    failwith ("unsupported language: " ^ !lang)
```

main_sgrep.ml

OCaml Match Many

```
let patterns, query_string =
  match !pattern_file, !pattern_string, !json_file, !use_multiple_patterns with
  | "", "", "", _ ->
    failwith "I need a pattern; use -f, -r, or -args"
  | file, _, _, true when file <> "" ->
    read_patterns file, “multi”
  | _, s, _, true when s <> "" ->
    failwith "cannot combine -multi with -e"
  | _, _, file, _ when file <> "" ->
    read_json_patterns file, "json args"
  | _ -> raise Impossible
```

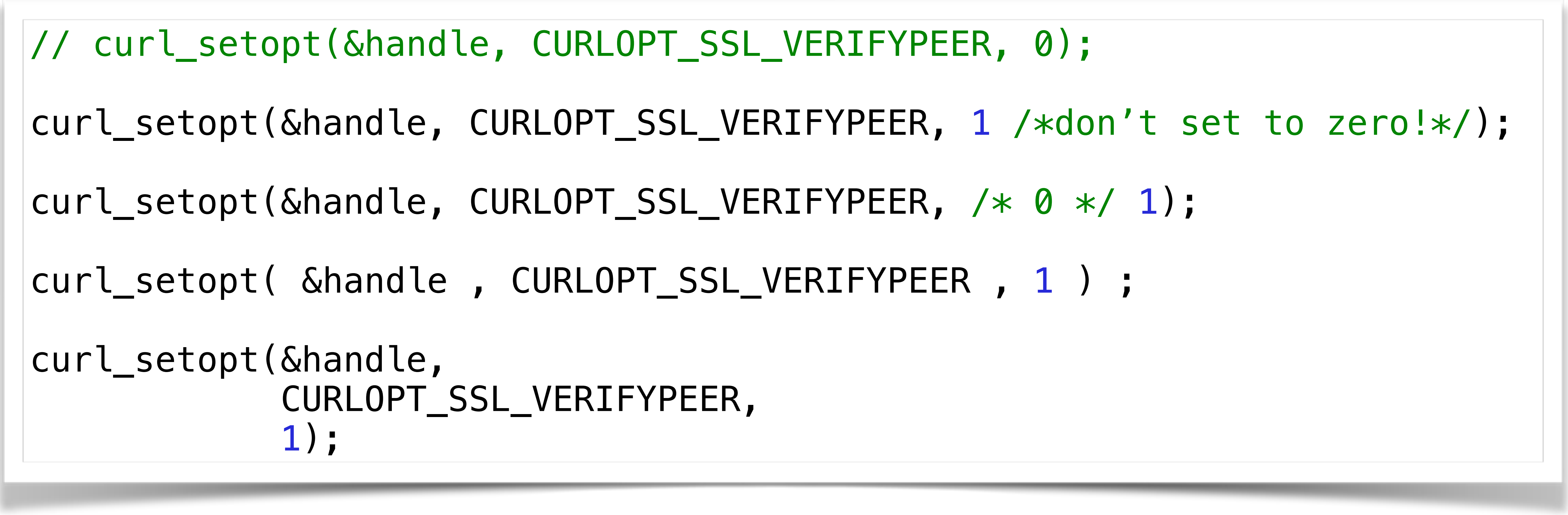
main_sgrep.ml

Matching Option Types

```
let string_or_nothing s =
  match s with
  | None -> ""
  | Some s -> string s
```

lang_php/matcher/**php_vs_php.ml**

Scan ~~Text~~ Code



```
// curl_setopt(&handle, CURLOPT_SSL_VERIFYPEER, 0);

curl_setopt(&handle, CURLOPT_SSL_VERIFYPEER, 1 /*don’t set to zero!*/);

curl_setopt(&handle, CURLOPT_SSL_VERIFYPEER, /* 0 */ 1);

curl_setopt( &handle , CURLOPT_SSL_VERIFYPEER , 1 ) ;

curl_setopt(&handle,
            CURLOPT_SSL_VERIFYPEER,
            1);
```

Syntactical Grep

```
sgrep –e 'password' .

sgrep –e 'bar(...)' .

sgrep –e 'curl_setopt(...,CURLOPT_SSL_VERIFYHOST,1)' .

sgrep –e "ini_set('open_basedir', ...)" .

sgrep –e "X = 'password'" –pvar X .

sgrep –e ‘bar(X, ...)' –pvar X .
```

Abstract Syntax Tree

Functions

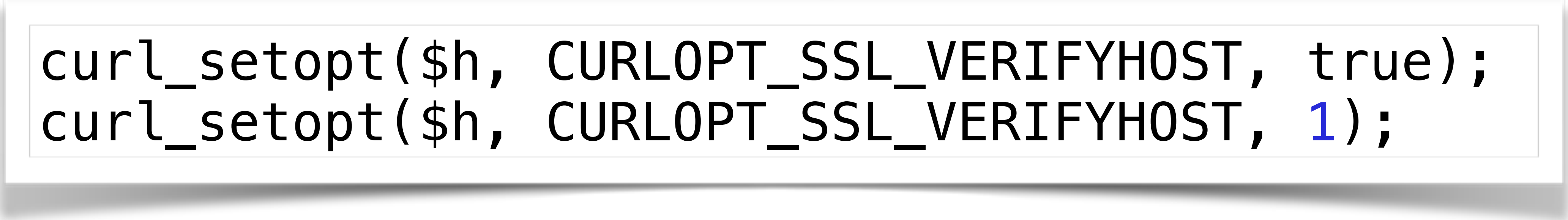
Arguments

Literals

Class names, methods

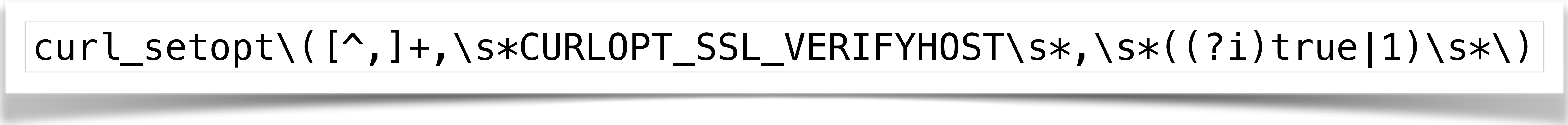
Types (strings, integers, Booleans)

Type Coercion (PHP)



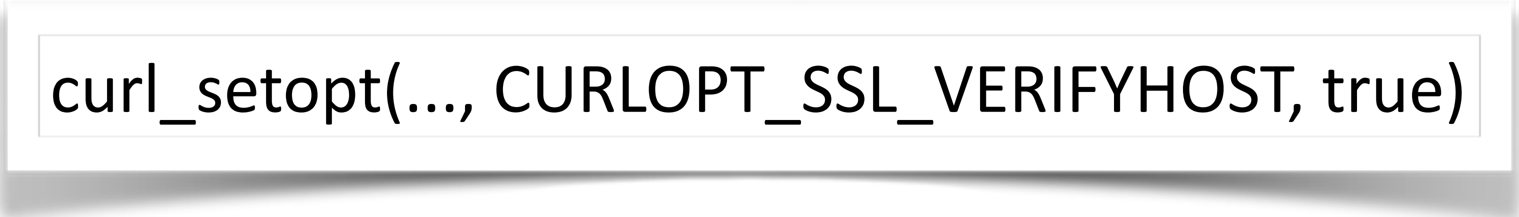
```
curl_setopt($h, CURLOPT_SSL_VERIFYHOST, true);
curl_setopt($h, CURLOPT_SSL_VERIFYHOST, 1);
```

regex

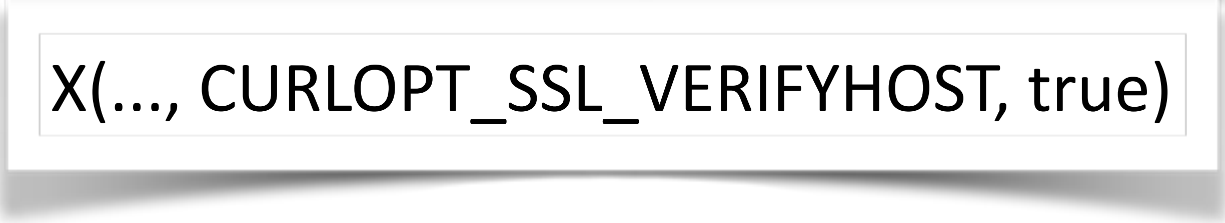


```
curl_setopt\([^,]+,\s*CURLOPT_SSL_VERIFYHOST\s*,\s*((?i)true|1)\s*\)
```

sgrep



curl_setopt(..., CURLOPT_SSL_VERIFYHOST, true)



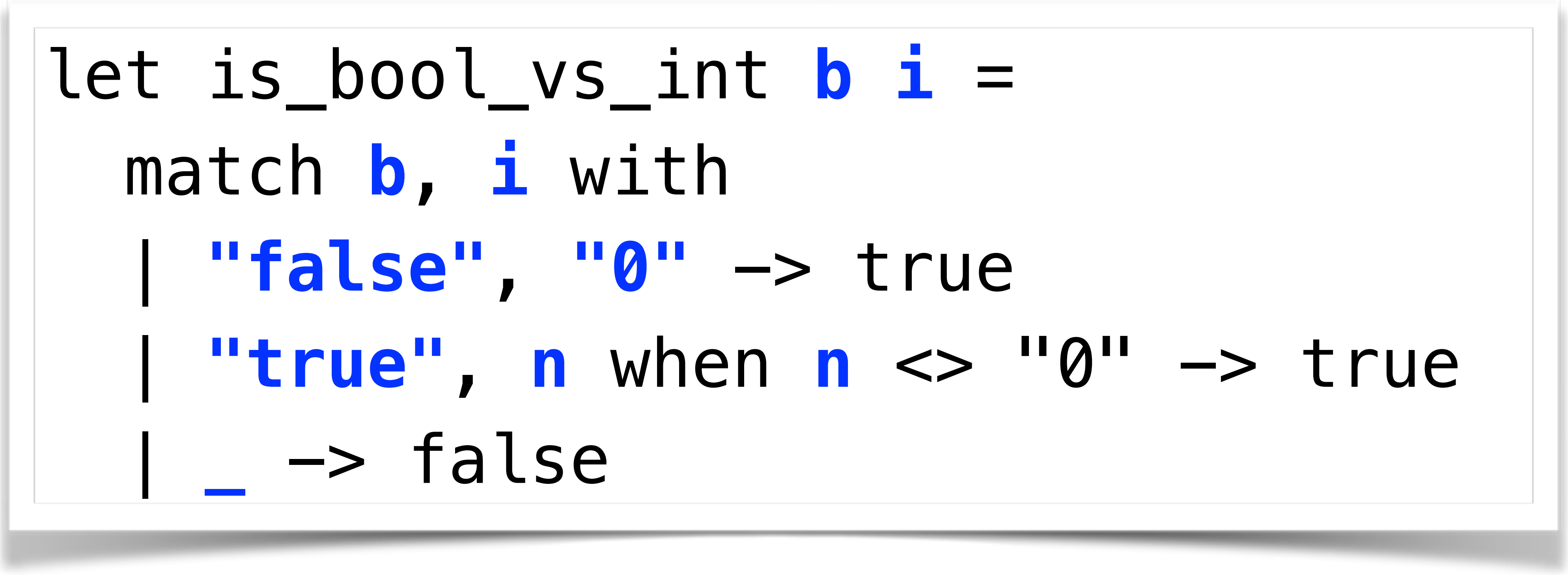
X(..., CURLOPT_SSL_VERIFYHOST, true)

Creating an Isomorphism

false == 0   true == 1

true == 2

lang_php/matcher/php_vs_php.ml



```
let is_bool_vs_int b i =
  match b, i with
  | "false", "0" -> true
  | "true", n when n <> "0" -> true
  | _ -> false
```

https://github.com/facebook/pfff/pull/117

Generating ML

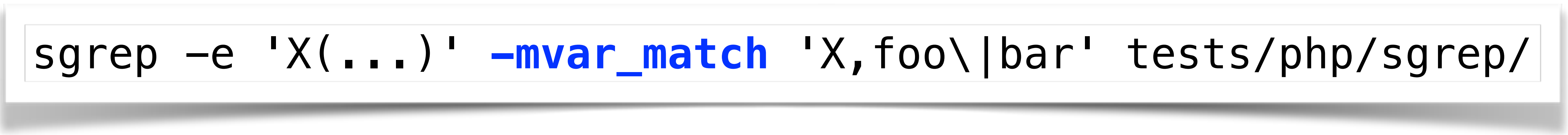
```
$ ./pfff -dump_php_ml tests/php/sgrep/boolean_vs_int.php
[StmtList(
   [ExprStmt(
      Call(Id(XName([QI(Name(("curl_setopt", i_1)))])),
        (i_2,
         [Left(Arg(IdVar(DName(("ch", i_3)), Ref(NoScope)))); Right(i_4);
          Left(Arg(Id(XName([QI(Name(("CURLOPT_SSL_VERIFYHOST", i_5)))]))));
          Right(i_6); Left(Arg(Id(XName([QI(Name(("true", i_7)))]))))],
         i_8)), i_9);
    ExprStmt(
      Call(Id(XName([QI(Name(("curl_setopt", i_10)))])),
        (i_11,
         [Left(Arg(IdVar(DName(("ch", i_12)), Ref(NoScope)))); Right(i_13);
          Left(Arg(Id(XName([QI(Name(("CURLOPT_SSL_VERIFYHOST", i_14)))]))));
          Right(i_15); Left(Arg(Sc(C(Int(("1", i_16))))))],
         i_17)), i_18)]); FinalDef(i_19)]
```

Hints for php_vs_php.ml

Harnessing meta_var

Match many functions and one argument

Bring back regex capabilities



```
sgrep -e 'X(...)' -mvar_match 'X,foo\|bar' tests/php/sgrep/
```

A String

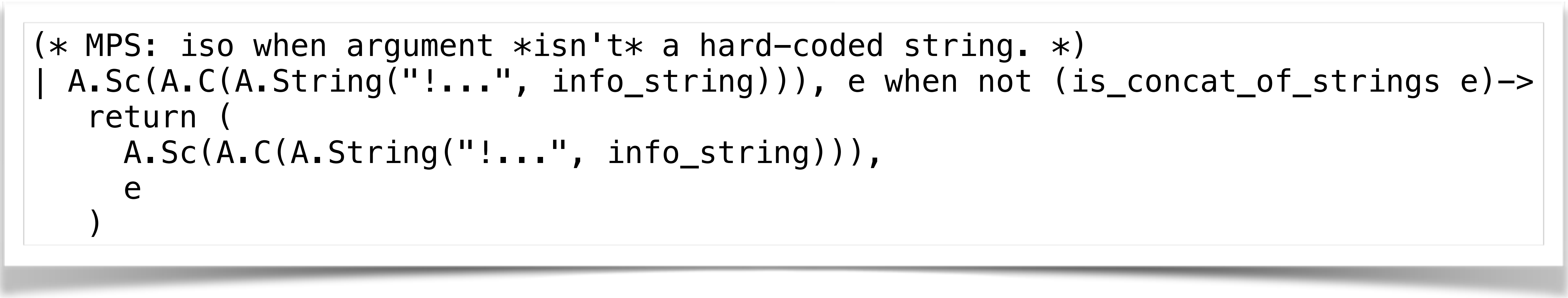
```
ini_set('X', true)

X
  allow_url_fopen
  expose_php
```

Not a String

```
X('!...')
```

```
X
    eval|passthru|popen|system
```

lang_php/matcher/php_vs_php.ml



```
(* MPS: iso when argument *isn't* a hard-coded string. *)
| A.Sc(A.C(A.String("!...", info_string))), e when not (is_concat_of_strings e)->
   return (
     A.Sc(A.C(A.String("!...", info_string))),
     e
   )
```

https://github.com/facebook/pfff/pull/119

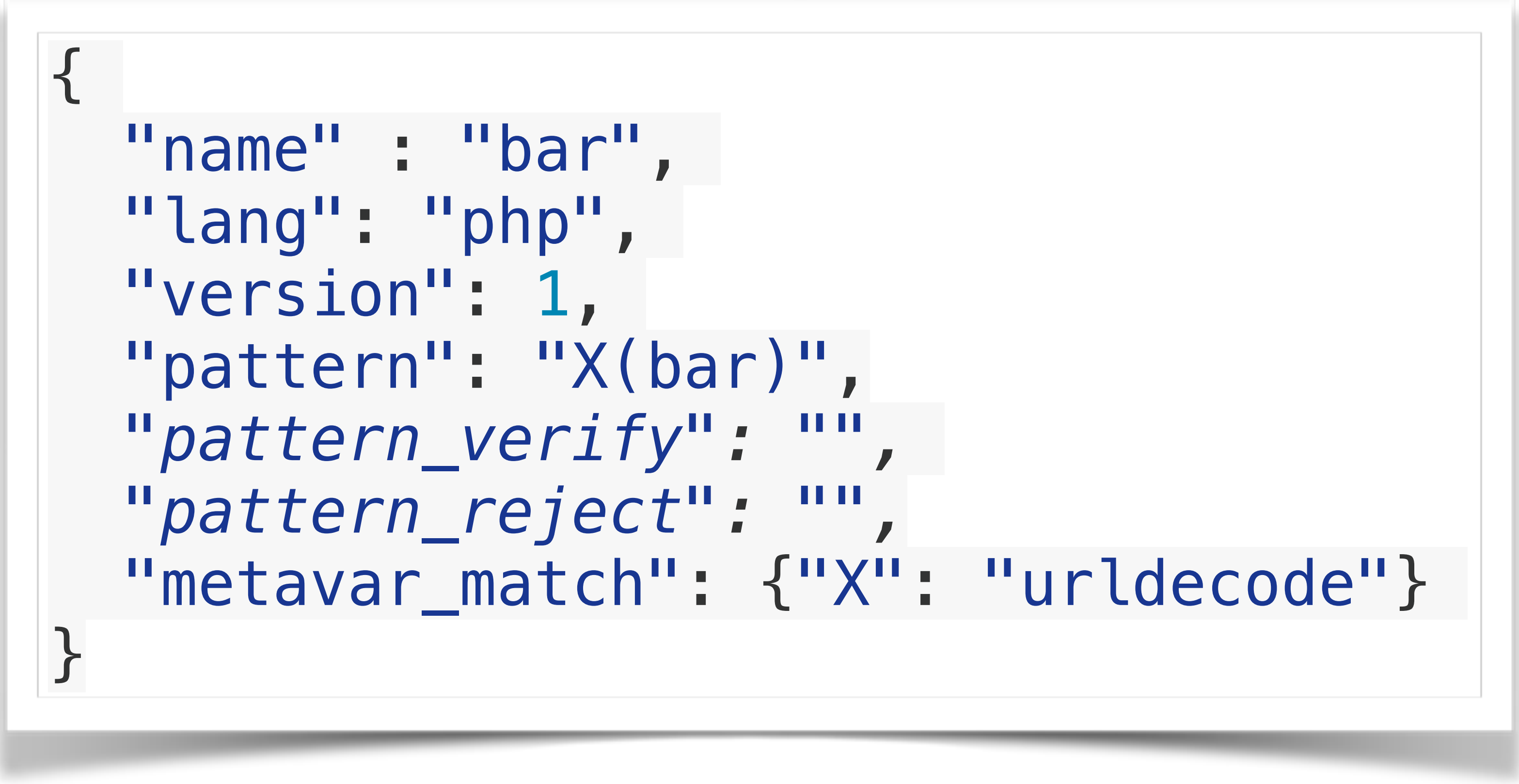
Patterns for Finding Flaws

This and that

This, but not that

A string

Not a string

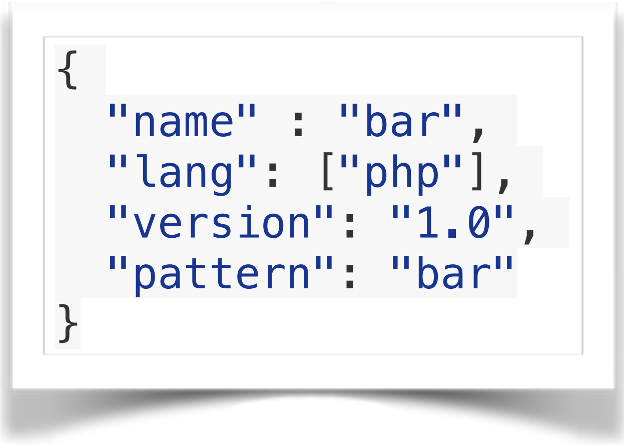


```
{
  "name" : "bar",
  "lang": "php",
  "version": 1,
  "pattern": "X(bar)",
  "pattern_verify": "",
  "pattern_reject": "",
  "metavar_match": {"X": "urldecode"}
}
```

Abstracting Behavior

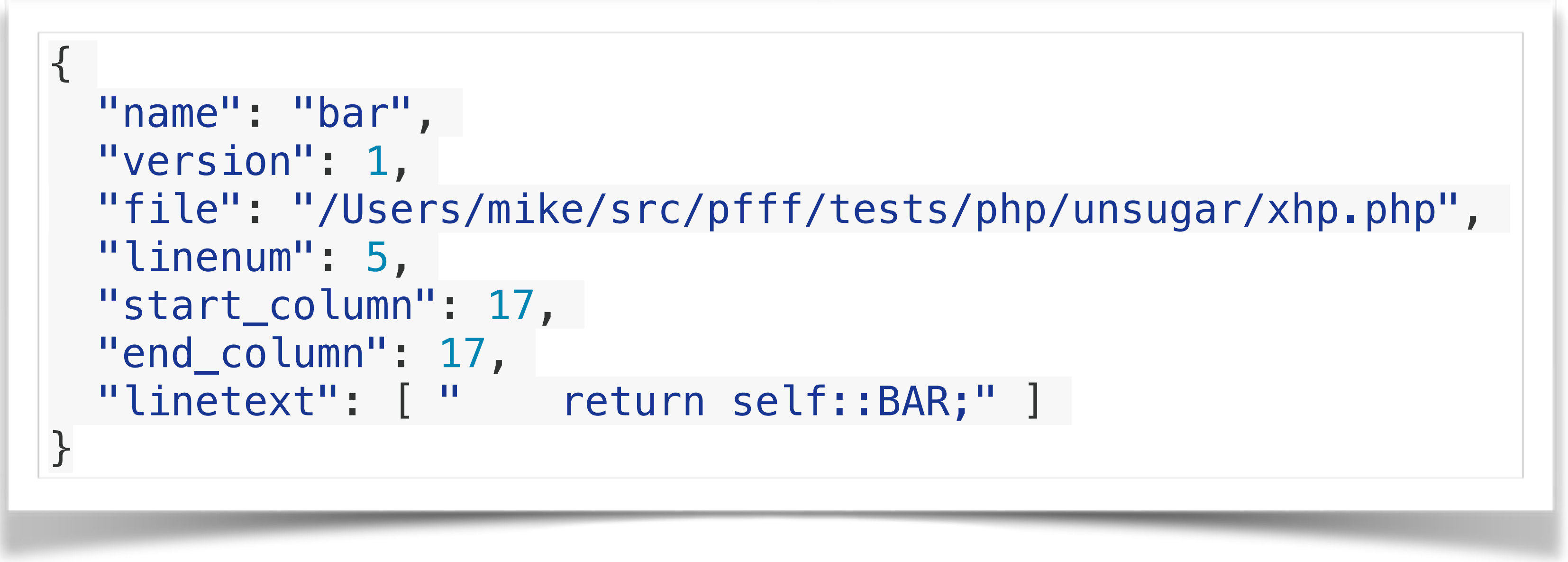
Write security checks instead of code

`sgrep -args <json file>` ←



```
{
  "name" : "bar",
  "lang": ["php"],
  "version": "1.0",
  "pattern": "bar"
}
```

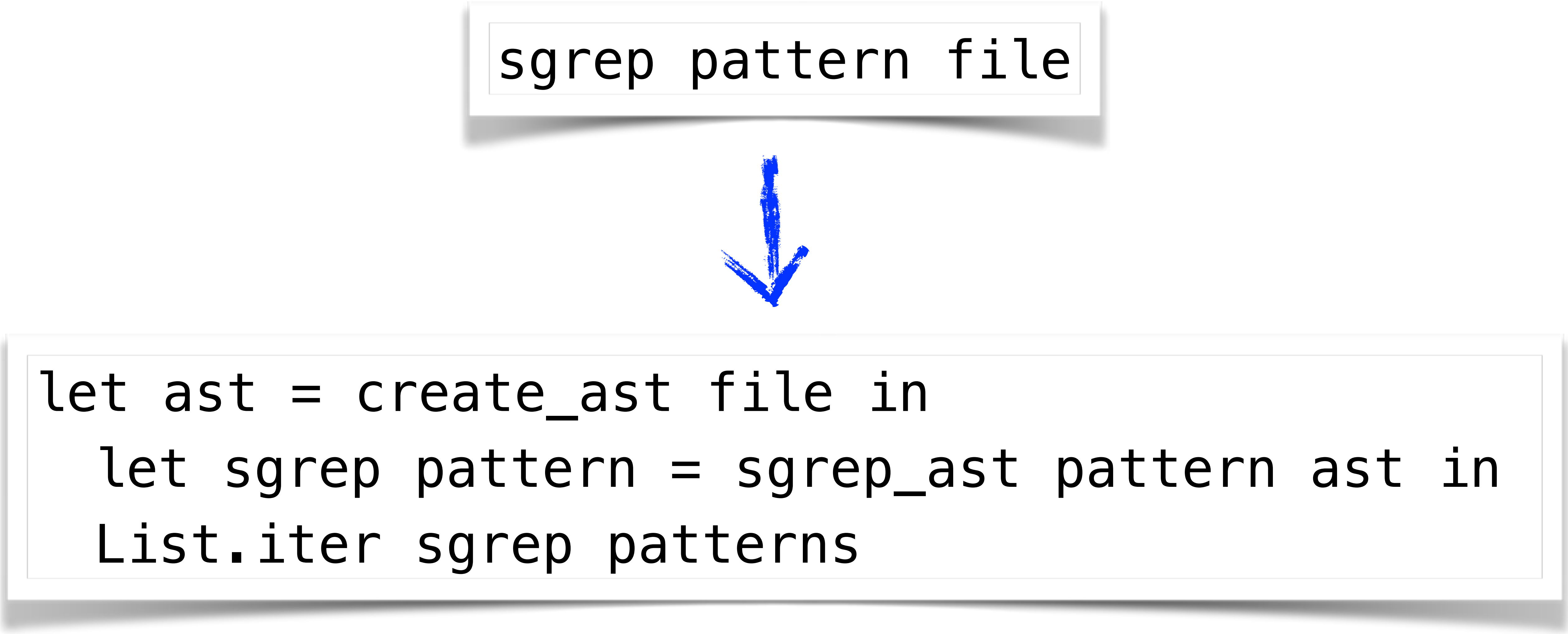
`sgrep -json` →



```
{
  "name": "bar",
  "version": 1,
  "file": "/Users/mike/src/pfff/tests/php/unsugar/xhp.php",
  "linenum": 5,
  "start_column": 17,
  "end_column": 17,
  "linetext": [ "    return self::BAR;" ]
}
```

https://github.com/mutantzombie/pfff/tree/json_io

Many Patterns, One AST



```
sgrep pattern file
```

```
let ast = create_ast file in
  let sgrep pattern = sgrep_ast pattern ast in
  List.iter sgrep patterns
```

https://github.com/facebook/pfff/pull/118

Navigating Pfff

lang_*

analyze

matcher

parsing

pretty

./tests/{lang}/sgrep

Problems for Future Fixes

Data flow analysis

Tainted variables, sanitization functions

Extend scope beyond translation unit

The World of Tomorrow

Coding in Future Tense

llvm, clang

Compile to intermediate representation

Address, Memory,Thread Sanitizers

-fsanitize=integer

-fsanitize=undefined

Code

Clean	Passes linter
	No dead paths
Readable	Written for humans
	Reasonable
Testable	Confidence in changes

Emergent Security

Handle large code bases across numerous repos under rapid change

Relatively easy to deploy and integrate

Produces accurate, actionable results

SDLC Touch Points

Configuration files

Stored secrets

Dynamically generated code

Compilation

Dependencies & versioning

Package signing

TODO: *FIXME*

sgrep for code, regex for text, parsers for XML (HTML, etc.)

Integrate with build process

Curate checks

Dive into pfff

OCaml is initially daunting, but ultimately rewarding

Thank You!

References

http://phrack.org/issues/49/14.html#article

http://windows.microsoft.com/en-US/windows/history#T1=era6

http://cppcheck.sourceforge.net

http://valgrind.org

https://github.com/facebook/pfff

https://github.com/mutantzombie/pfff (experiments)

1,023,705

A Few More Words

British Library has added public domain images on Flickr.

https://flic.kr/p/hUBkRN

https://flic.kr/p/hPYqyy

https://flic.kr/p/i8DSiY

https://www.flickr.com/people/britishlibrary/