The Guts of Perl
(and why you should care)

A brief tour through the perl compiler backends for the impatient refactorerer.

Eric Wilhelm
Scratch Computing
http://scratchcomputing.com

What is this "Perl" thing anyway?

- Perl is an *interpreted* language
  - "scripting" doesn't really describe it
  - *runtime* features
  - cannot be compiled into machine code
    - (requires an embedded perl)
- syntax / operators
- functions
- structure
- <u>technology</u> (perl)

What is this "perl" thing anyway?

Without a compiler, C is just pseudocode.

Without an interpreter, Perl is just line noise.

- perl is the interpreter for Perl
  - parser
  - compiler
  - runtime

```
/*    perlmain.c
...
 * "The Road goes ever on and on,
 * down from the door where it began."
 */
...
#include "perl.h"
...

int
main(int argc, char **argv, char **env)
{
  ...
  exitstatus = perl_parse(my_perl, xs_init, argc, argv, (char **)NULL);
  if (!exitstatus)
    perl_run(my_perl);
...
```

perlmain.c

- main ... {
  - perl_parse(...)
  - perl_run(...)
- }

```
/*    perlmain.c
...
 * "The Road goes ever on and on,
 * down from the door where it began."
 */
...
#include "perl.h"
...

int
main(int argc, char **argv, char **env)
{
  ...
  exitstatus = perl_parse(my_perl, xs_init, argc, argv,
  if (!exitstatus)
    perl_run(my_perl);
...
```

perl.c

- perl_parse
  - parsing / compiling (optimization)
- perl_run
  - runtime: state/context, eval()

```
/*    perl.c
...
* ... mithril and of elven glass"
* --Bilbo
*/
...
int
perl_parse(pTHXx_ XSINIT_t xsinit, int argc, char **ar
{
...
int
perl_run(pTHXx)
{
  ...
  switch (ret) {
  case 1:
    cxstack_ix = -1;              /* start context stack a
    goto redo_body;
  case 0:                         /* normal completion */
```

XS/Inline Extensions

- use the perl API
  - perlguts
  - perlapi
- anything that can be embedded in C
  - perl, python, ruby, C++, C, ObjC
  - (or interfaced from C)
    - java (?), lisp, smalltalk, haskell, fortran (?), erlang
  - eventually, it's all machine code (assembly?)

embedded perl

Did your boss say you can't use Perl, perl, or PERL?

- C programs can embed perl
  - perlembed -> perlapi -> perlcall
- call_argv()
  - Performs a callback to the specified Perl sub.
- eval_sv()
  - Tells Perl to eval the string in the SV.

extensions are embedded

- callbacks
- regex engine access
  - perlapi says don't dig into proto.h

    (probably for a good reason)
  - eval_sv
    - puts you in the right context
    - garbage collection
    - other magic

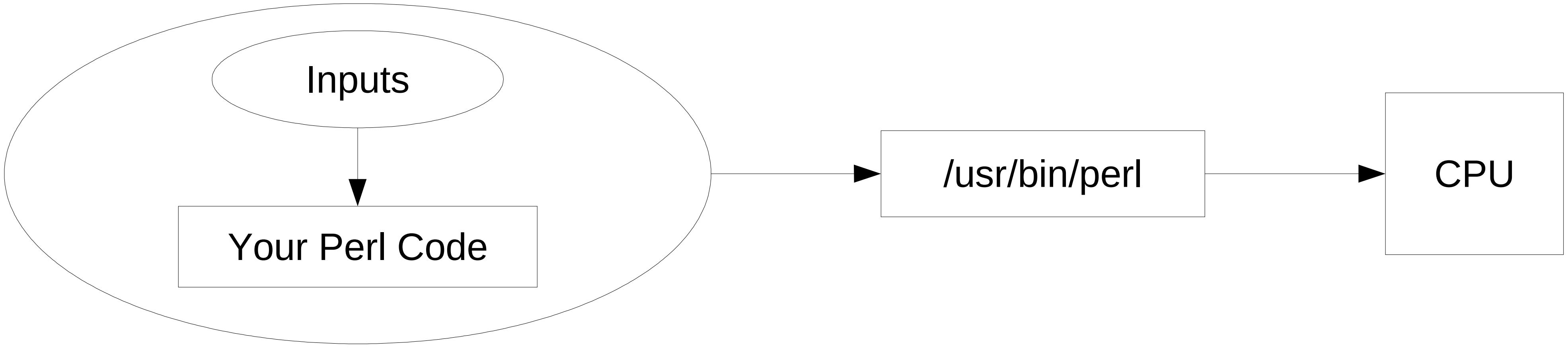
Enough C Already!

- You get the point:

  There is no man behind the curtain.

  (At least not in your box.)

  - Your Perl (and its inputs) drive the execution path of some machine code that has been written (and rewritten) in C
  - This code drives the execution path of the CPU.



Another way to see it.

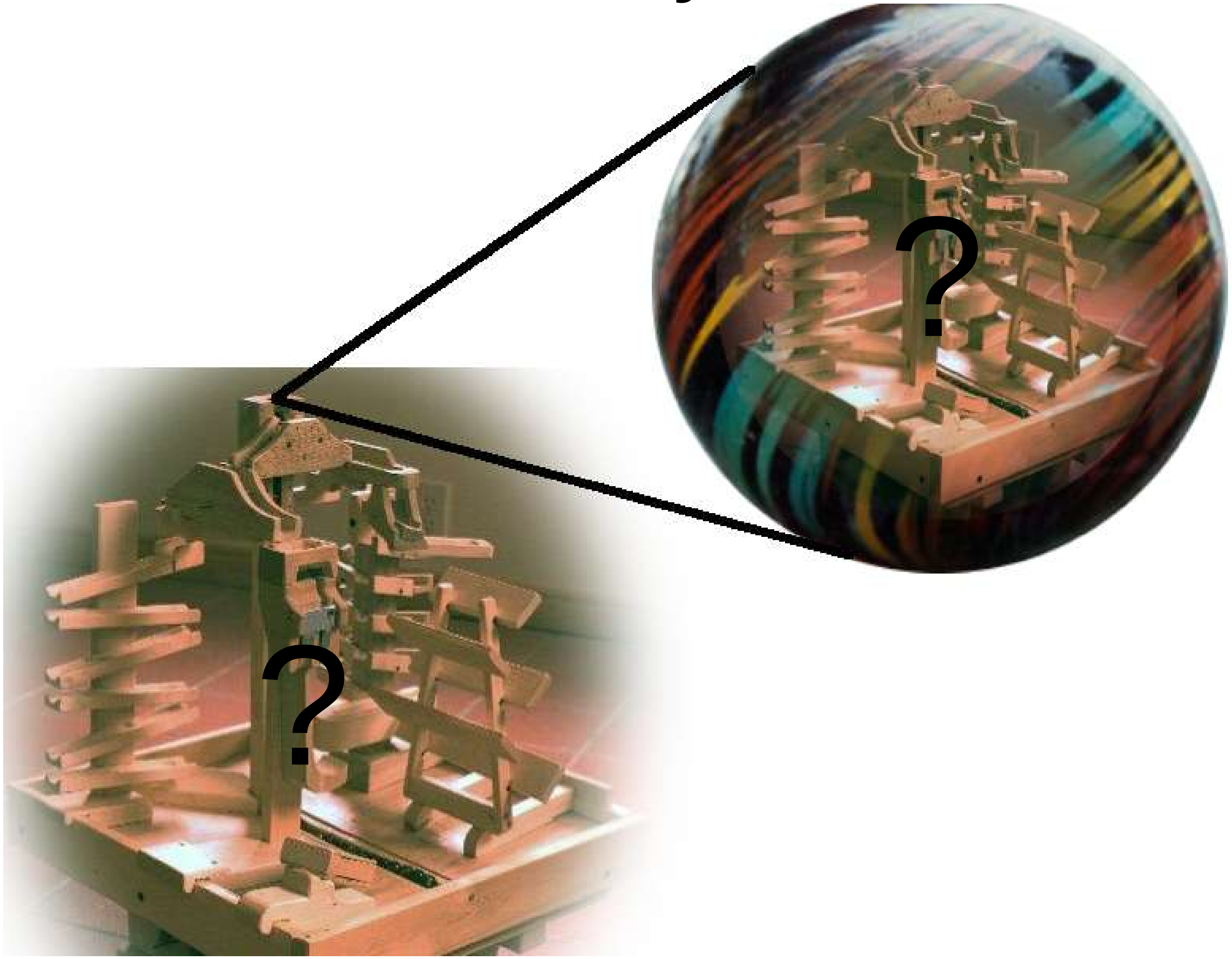


image courtesy of Matthias Wandel http://www.sentex.net/~mwandel/

perl, what is Perl?

- We need perl to parse and analyze our Perl.
  - How can we use Perl to ask perl what it thinks of our Perl without changing our Perl?
- compile
- inspect
  - opcodes / syntax tree
  - variable names, scope
  - imports, subroutine definitions

O

- Generic interface to Perl Compiler backends

```
perl -MO=[-q,]Backend[,OPTIONS] foo.pl
```

- Stops before runtime
- Calls B::*Backend*
- B - The Perl Compiler (sort of)
  - provides utility functions for B::* modules

B::*

B::Asmdata
B::Assembler
B::Bblock
B::Bytecode
B::C
B::CC
B::Concise
B::Debug
B::Deobfuscate
B::Deparse
B::Disassembler
B::Fathom
B::FindAmpersand
B::Flags
B::Graph
B::IntrospectorDeparse
B::Keywords
B::LexInfo
B::Lint
B::Lisp
B::Lisp::_impl
B::Module::Info
B::More
B::OptreeShortestPath
B::PerlReq
B::Showlex
B::Size
B::Stackobj
B::Stash
B::Terse
B::TerseSize
B::Tree
B::TypeCheck
B::Utils
B::XPath
B::Xref

B::Deparse

- Shows you what perl thinks you mean.
- What is left of your code after BEGIN.

```
$ perl -MO=Deparse -e 'use constant {FOO => 0};
FOO and print "hey\n";'

use constant ({'FOO', 0});
'???';
-e syntax OK
```

B::Deparse

- Actually executes the BEGIN blocks
  - Yes! Using B::* means I can root your editor.
- BEGIN happens at compile-time.

```
$ perl -MO=Deparse -e 'BEGIN {print "hey\n"};'
hey
sub BEGIN {
    print "hey\n";
}
-e syntax OK
```

Fun with B::Deparse

```
$ perl -MO=Deparse -e 'sub foo { 1||1;}' 2>/dev/null
sub foo {
    1;
}

$ perl -MO=Deparse -e 'sub foo { 0||1;}' 2>/dev/null
sub foo {
    1;
}

$ perl -MO=Deparse -e 'sub foo { 0&&1;}' 2>/dev/null
sub foo {
    0;
}
```

More Fun with B::Deparse

```
$ perl -MO=Deparse -e 'sub foo { 0&1; }' 2>/dev/null
sub foo {
    0;
}

$ perl -MO=Deparse -e 'sub foo { 2&1; }' 2>/dev/null
sub foo {
    0;
}

$ perl -MO=Deparse -e 'sub foo { 1&1; }' 2>/dev/null
sub foo {
    1;
}
```

Useful B::Deparse Example

```
#!/usr/bin/perl
use strict;
use warnings;

sub one {
     {
         label    => 'Foo',
         data    => [ qw/ baz /],
     };
}

sub two {
     return {
         %{ one() },
         label    => 'Bar',
     };
}

sub three {
     {
         data    => 'test',
         label   => 'Bar',
     };
}

sub four {
     {
         %{ one() },
         label    => 'Bar',
     };
}
```

```
sub three {
     {
         data    => 'test',
         label   => 'Bar',
     };
}

sub four {
     {
         %{ one() },
         label    => 'Bar',
     };
}
```

Implicit return() -> block!

```
#!/usr/bin/perl
use strict;
use warnings;

sub one {
     {
         label    => 'Foo',
         data    => [ qw/ baz /],
     };
}

sub two {
     return {
         %{ one() },
         label    => 'Bar',
     };
}

sub three {
     {
         data    => 'test',
         label   => 'Bar',
     };
}

sub four {
     {
         %{ one() },
         label    => 'Bar',
     };
}
```

```
sub four {
     {
         %{ one() },
         label    => 'Bar',
     };
}

##############################################
$ perl -MO=Deparse,-p returns_what.pl
...
sub four {
    BEGIN {${^WARNING_BITS} = "UUUUUUUUUUUU"}
    use strict 'refs';
    {
        (%{one();}, 'label', 'Bar');
    }
}
...
```

LIST!

How i18n Works

```
$ perl -MO=Deparse -e 'sub foo { ~~"hey"; }' 2>/dev/null
sub foo {
    'hey';
}


$ perl -MO=Deparse -e 'use i18n;sub foo { ~~"hey"; }' 2>/dev/null
use i18n;
sub foo {
    no warnings;
    [sub {
        package Locale::Maketext::Simple;
        use strict 'refs';
        $lh->maketext(@_);
    }
    , 'hey'];
}
```

B::Fathom

- evaluate the readability of Perl code
  - does this by inspecting the syntax tree
  - segfaults in some situations
    - perl version / code construct combinations
  - doesn't go outside 'main' package?

```
$ perl -MO=Fathom ~/.bin/crs2svg
  121 tokens
   28 expressions
   12 statements
    2 subroutines
readability is 3.51 (readable)
```

```
$ perl -MO=Fathom awstats.pl
80347 tokens
29953 expressions
 7907 statements
   71 subroutines
readability is 11.45 (obfuscated)
```

Fathom is Limited

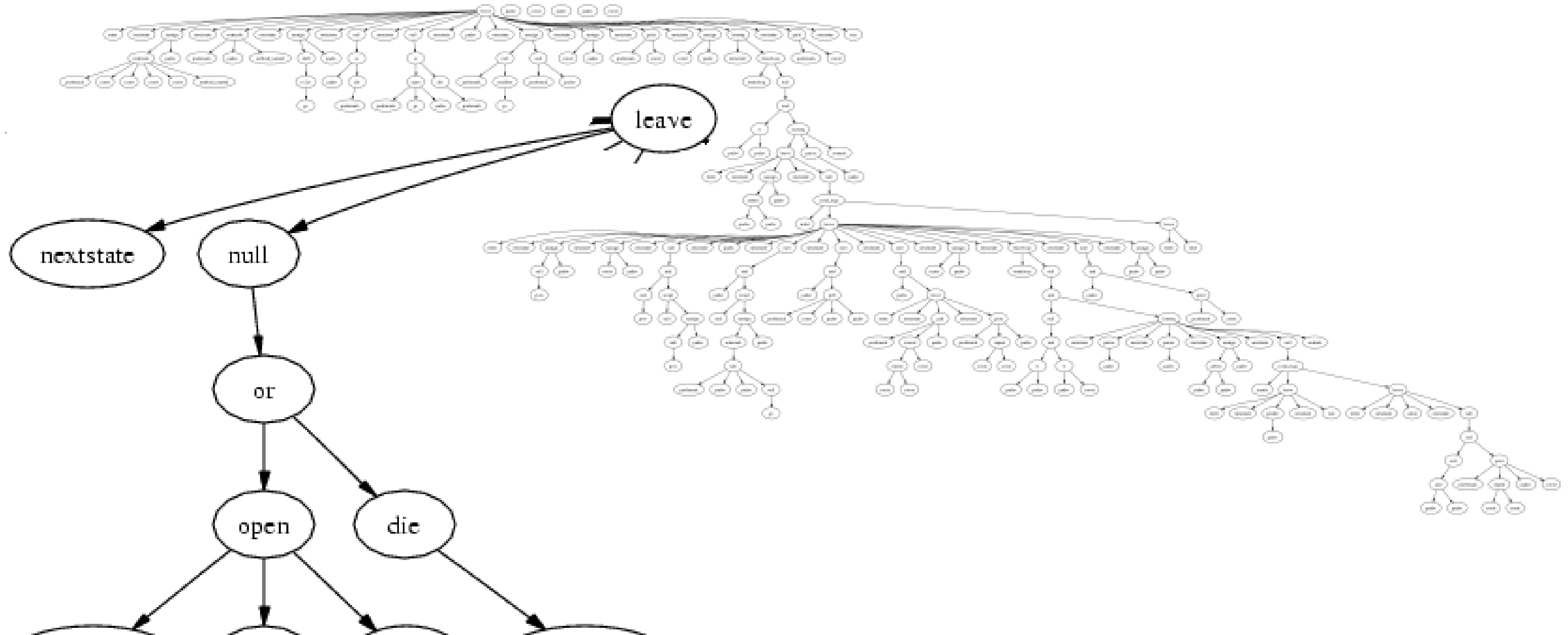
- A low number does not necessarily mean that it is fathomable.
  - It won't tell you why.

```
perl -MO=Fathom -e 'my $s = join("|", map({join("", @$_)}
  map({[$_, uc($_), $_]} split(//, join("",
    map({chr($_)} 97..122))))));
$s =~ s/(M.)\|/$1\n/;
print "\u$s\n";'

   79 tokens
   28 expressions
    3 statements
    1 subroutine
readability is 4.41 (easier than the norm)
```

B::Tree

- Simplified B::Graph
  - GraphViz
  - B::Graph is way too complex for simple demos



B::Xref

- cross reference reports for Perl programs
  - variable / subroutine names, packages
  - modules used, imports
  - global / lexical variables
    - intro
    - used
  - subroutine info
    - subdef
    - subused

B::Xref usage

- report or "raw" modes
  - both report some cruft

```
$ perl -MO=Xref -e 'my $a = 7;'
File -e
  Subroutine (main)
    Package (lexical)
      $a                i1
```

- raw mode needs work
  - joins space-laden fields with spaces

```
$ perl -MO=Xref,-r -e 'my $a = 7;'
-e   (definitions)     0 Regexp          & DESTROY            subdef
-e   (definitions)     0 UNIVERSAL       & isa                subdef
...
-e   (definitions)     0 PerlIO          & get_layers         subdef
-e   (definitions)     0 Internals       & SvREFCNT           subdef
...
-e   (main)            1 (lexical)       $ a                  intro
```

What Cruft?

```
$ perl -MO=Xref,-r -e ''
-e  (definitions) 0 Regexp     & DESTROY                subdef
-e  (definitions) 0 UNIVERSAL  & isa                    subdef
-e  (definitions) 0 UNIVERSAL  & VERSION                subdef
-e  (definitions) 0 UNIVERSAL  & can                    subdef
-e  (definitions) 0 PerlIO     & get_layers             subdef
-e  (definitions) 0 Internals  & SvREFCNT               subdef
-e  (definitions) 0 Internals  & hv_clear_placeholders subdef
-e  (definitions) 0 Internals  & hash_seed              subdef
-e  (definitions) 0 Internals  & SvREADONLY             subdef
-e  (definitions) 0 Internals  & HvREHASH               subdef
-e  (definitions) 0 Internals  & rehash_seed            subdef
```

B::Xref vs PPI

- PPI does not execute code
- PPI does not leave the document
- B::Xref does both
  - but does not see your code the way you do

```
$ perl -MO=Xref,-r -e 'foo();
sub foo { print "hello world\n"};' | grep foo

-e  (definitions)     1 main              & foo                subdef
-e  (main)            1 main              & foo                subused

$ perl -MO=Xref,-r -e 'sub foo { print "hello world\n"};
foo();' | grep foo

-e  (definitions)     1 main              & foo                subdef
-e  (main)            2 main              & foo                subused
```

Xref | grep Can get tedious.

- We want it in SQL!
  - except there is that spaces thing
  - one line hacked-in patch

```
-        printf "%-16s %-12s %5d %-12s %4s %-16s %s\n",
+        printf "%-16s|%-12s|%5d|%-12s|%4s|%-16s|%s\n",
```

```
$ perl -MO=Xreft,-r dirvish | \
    perl -pe 's/\s*\|\s*/\t/g'
```

WHEE!

```
$ sqlite dirv.db '
CREATE TABLE xref (
  filename TEXT,
  subname  TEXT,
  line     INT,
  package  TEXT,
  type     TEXT,
  name     TEXT,
  event    TEXT
  );

CREATE VIEW mainprog AS
  SELECT subname,line,package,type,name,event
    FROM xref
    WHERE filename="/path/to/dirvish";
'
```

Load the Table

```
$ perl -MO=Xreft,-r /path/to/dirvish | \
   perl -pe 's/\s*\|\s*/\t/g' | \
      sqlite dirv.db "COPY xref FROM STDIN;"
```

Now What?

- Ask it some questions.
  - "mainprog" view for filename="foo.pl"
  - "xref" table for world view
  - maybe add some PPI tables?
- Can we turn on strict and warnings?
- How many global variables are there?
- What is being Exporter'd to where?
- Use your imagination.

Events

- What (might) happen where?
- How "healthy" is the code?

```
$ sqlite dirv.db
sqlite> SELECT DISTINCT event FROM xref;
subdef
used
subused
intro
sqlite> SELECT subname,package,line,name FROM mainprog
  WHERE event='intro';
...
sqlite> SELECT DISTINCT package FROM mainprog
  WHERE event='intro';
(lexical)
```

Globs

- Should usually be filehandles

```
sqlite> SELECT DISTINCT name FROM mainprog WHERE
type='*';
key
STDERR
SUMMARY
EXCLUDE
FSBUF
LOG_FILE
HIST
INDEX
_
LOGFILE
ERR_FILE
```

Huh?

```
$ perl -MO=Xref,-r -e 'for $key (qw(a b c)) {
  print "hey $key\n";
}' | grep key

-e  (main)  1 main      * key              used
-e  (main)  2 main      $ key              used
```

```
$ perl -MO=Xref,-r -e 'for my $key (qw(a b c)) {
  print "hey $key\n";
}' | grep key

-e  (main)  2 (lexical) $ key              used
```

Package Name

- Xref(t) uses "(lexical)" or the package name

```
SELECT DISTINCT name FROM mainprog
  WHERE package='(lexical)';

SELECT DISTINCT name FROM mainprog
  WHERE package!='(lexical)';
```

```
sqlite> SELECT COUNT(name) FROM
  (SELECT DISTINCT name FROM
    mainprog WHERE package='(lexical)');
41
sqlite> SELECT COUNT(name) FROM
  (SELECT DISTINCT name FROM
    mainprog WHERE package!='(lexical)');
95
```

Variable Use Counts

- What is global?
- What is ripe for becoming a constant?

```
sqlite> SELECT package,name,COUNT(name) AS counts
  FROM mainprog WHERE package!='(lexical)'
    AND type != '&' AND TYPE != '*'
  GROUP BY package,name HAVING counts >10
  ORDER BY package,counts DESC;
package                    name           counts
-------------------------  -------------  ------
main                       Options        320
main                       status         43
...
main                       RSYNC_CODES    13
main                       log_file       12
```

Variable Types

- $,%,@,&,*, and others (%$,@$, etc)

```
sqlite> SELECT DISTINCT name,type FROM mainprog
  WHERE type!='*' AND type!='&';
...
CONFDIR                       $
Options                       %%%%%%%%%%%%
file                          $
...
```

- Show me your subs

```
sqlite> SELECT DISTINCT package,name FROM mainprog
  WHERE type='&' AND package='main';

sqlite> SELECT DISTINCT package,name FROM mainprog
  WHERE type='&' AND package!='main';
```

Exporter

- We all know something weird happens on line 65 in Exporter.pm.

```
sqlite> SELECT package,line,name,event FROM xref
  WHERE filename LIKE '%/Exporter.pm' AND event='subdef';
Time::JulianDay|65|croak|subdef
Time::JulianDay|65|confess|subdef
Time::JulianDay|65|tz_offset|subdef
...

sqlite> SELECT name FROM xref
  WHERE filename LIKE '%/Exporter.pm' AND event='subdef'
  AND package='main';
inPeriod
parsedate
strftime
```

More SQL Fun

```
SELECT DISTINCT package FROM mainprog WHERE package!='main'
  AND package!='(lexical)';

SELECT DISTINCT package,name,type,event FROM mainprog
  WHERE package!='main' AND package!='(lexical)'
  AND event LIKE 'sub%';

SELECT * FROM mainprog WHERE event='intro';

SELECT * FROM mainprog WHERE event='intro' AND package!='(lexical)';

SELECT * FROM xref WHERE name='inverse_julian_day'
  AND event='subdef';

SELECT DISTINCT package FROM mainprog WHERE name='Options';

SELECT count(name) FROM mainprog WHERE name='Options'
  AND package!='(lexical)';

SELECT * FROM mainprog WHERE name='rsyncargs';

SELECT * FROM mainprog WHERE name='seppuku' ORDER BY line;
```

Next Steps

- Add auto-update
  - SGI::Fam -> watch foo.pl, reload db on saves
- Add PPI
  - subused before subdef?
  - our() vs use vars
    - vars yields no "intro" event
  - actually automated refactoring
- Maybe a GUI
  - $filemanager =~ s/file/code/;