WELCOME TO CPERL

perl11: 5 + 6 = 11

http://perl11.org/cperl/

QUO VADIS?

perl5+6 dev workshop 2011

Will Braswell

Austin 2012

Ingy döt net

Orlando 2013

PERL IS NOT DEAD, IT IS A DEAD END

Stevan Little
Orlando Perl Workshop 2013
stevan.little@iinteractive.com

Mouse not Moose

CPERL

- with classes (types)
- compiler support
- continuation of perl development
- company-friendly

CPERL

- Features
- Bugfixes, Security
- In Work
- Policies

FEATURES

- 44 new major features in the last 2 years
- cperl-5.22.1 (Feb. 2015) - 5.27.1
- the only full-time perl5 core developer
- ~10x more than p5p

zb . von @INC entfernt mit cperl-5.22, perl5 erst 2 Jahre später mit 5.26.

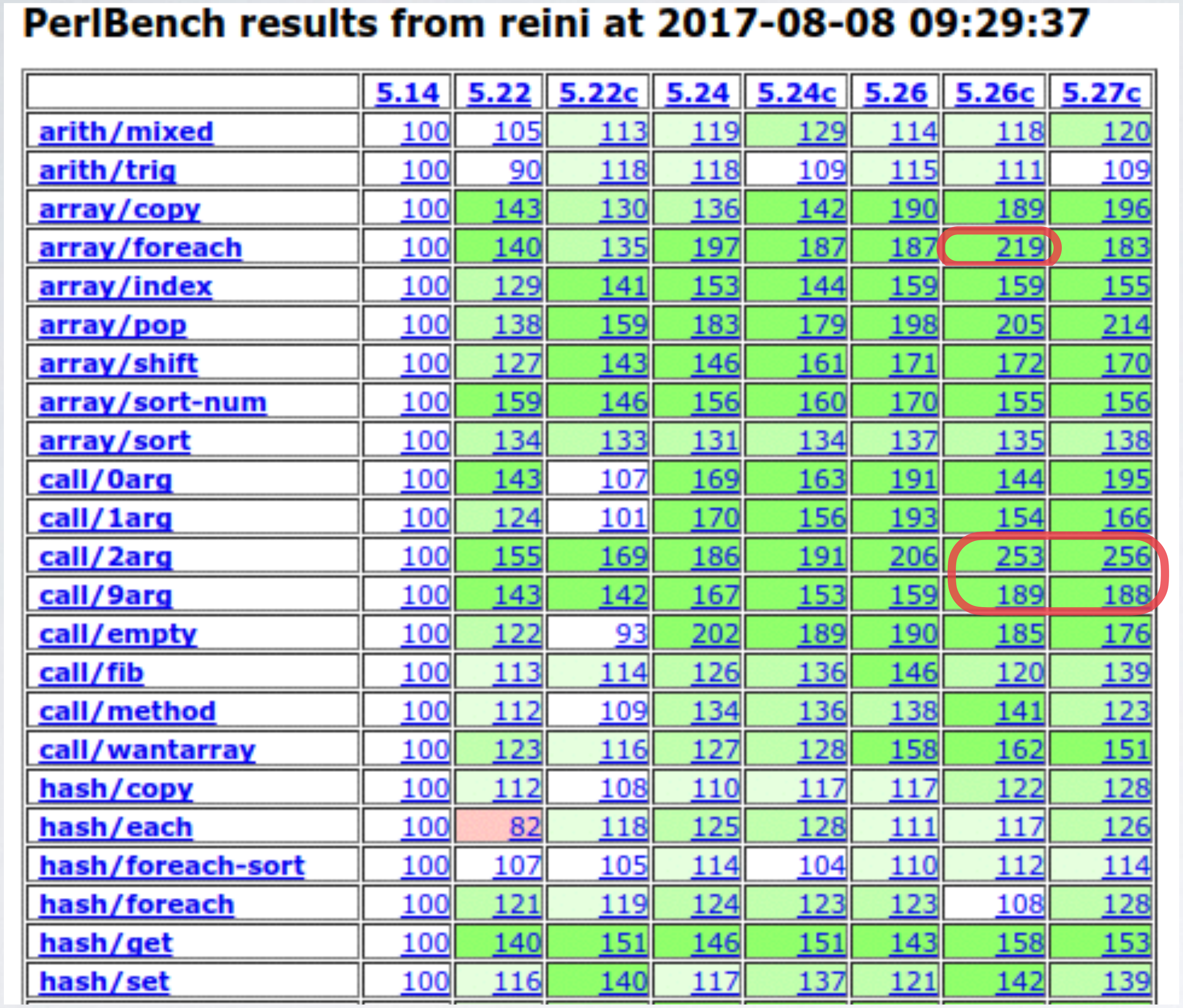
der einzige cperl fix von >100 der auch in perl5 ist.

Overview http://perl11.org/cperl/STATUS.html

FEATURES

- ~20% faster (with signatures and modern perl 2x faster)
- dramatically less memory (many helpers rewritten in XS)
- protection from p5p policies and destruction
  removal of lexical $_, use encoding, signatures, for (qw()) {}, …
- professional and public development, no mailinglist, no major features directly to master
- cperl: 90% of perl5 commits merged,
  perl5: 0% of cperl commits merged

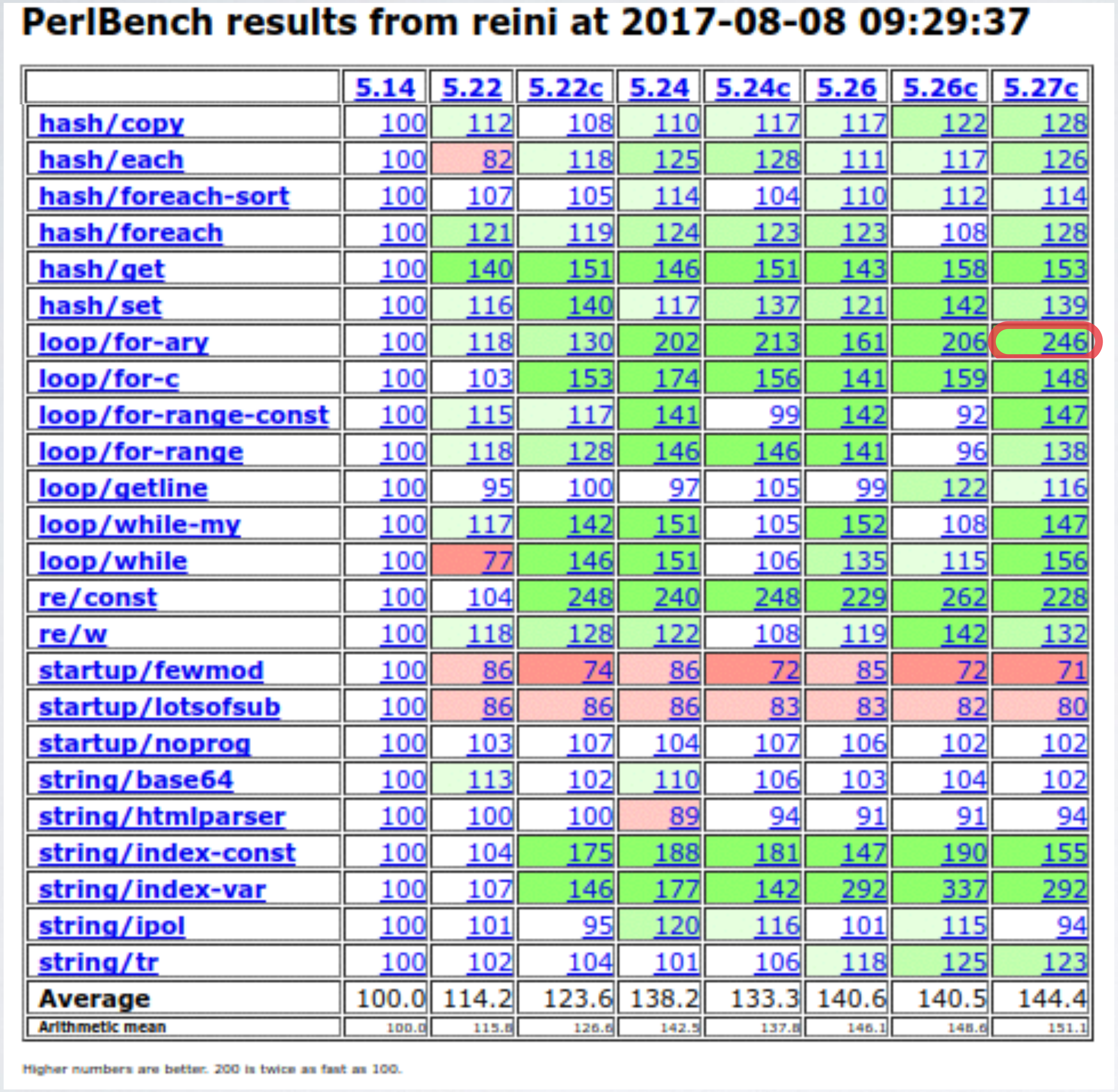
SPEED



PerlBench results from reini at 2017-08-08 09:29:37

	5.14	5.22	5.22c	5.24	5.24c	5.26	5.26c	5.27c
arith/mixed	100	105	113	119	129	114	118	120
arith/trig	100	90	118	118	109	115	111	109
array/copy	100	143	130	136	142	190	189	196
array/foreach	100	140	135	197	187	187	219	183
array/index	100	129	141	153	144	159	159	155
array/pop	100	138	159	183	179	198	205	214
array/shift	100	127	143	146	161	171	172	170
array/sort-num	100	159	146	156	160	170	155	156
array/sort	100	134	133	131	134	137	135	138
call/0arg	100	143	107	169	163	191	144	195
call/1arg	100	124	101	170	156	193	154	166
call/2arg	100	155	169	186	191	206	253	256
call/9arg	100	143	142	167	153	159	189	188
call/empty	100	122	93	202	189	190	185	176
call/fib	100	113	114	126	136	146	120	139
call/method	100	112	109	134	136	138	141	123
call/wantarray	100	123	116	127	128	158	162	151
hash/copy	100	112	108	110	117	117	122	128
hash/each	100	82	118	125	128	111	117	126
hash/foreach-sort	100	107	105	114	104	110	112	114
hash/foreach	100	121	119	124	123	123	108	128
hash/get	100	140	151	146	151	143	158	153
hash/set	100	116	140	117	137	121	142	139

SPEED

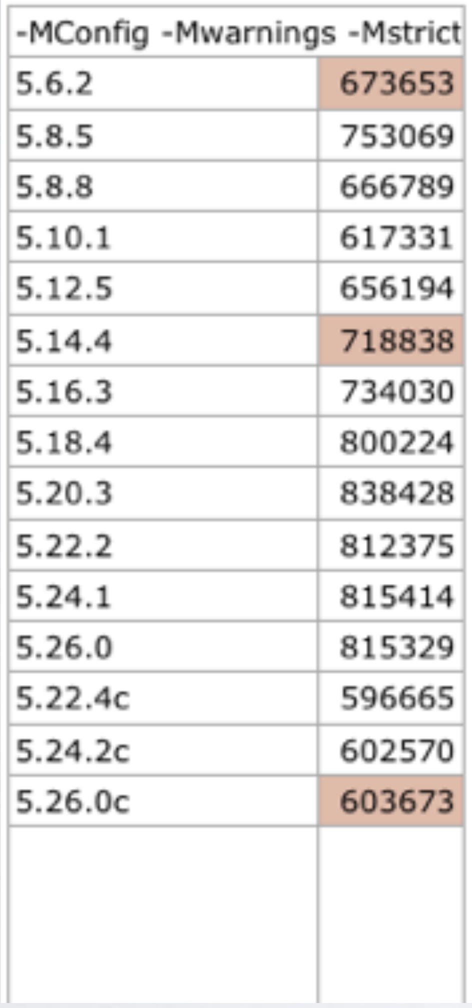


PerlBench results from reini at 2017-08-08 09:29:37

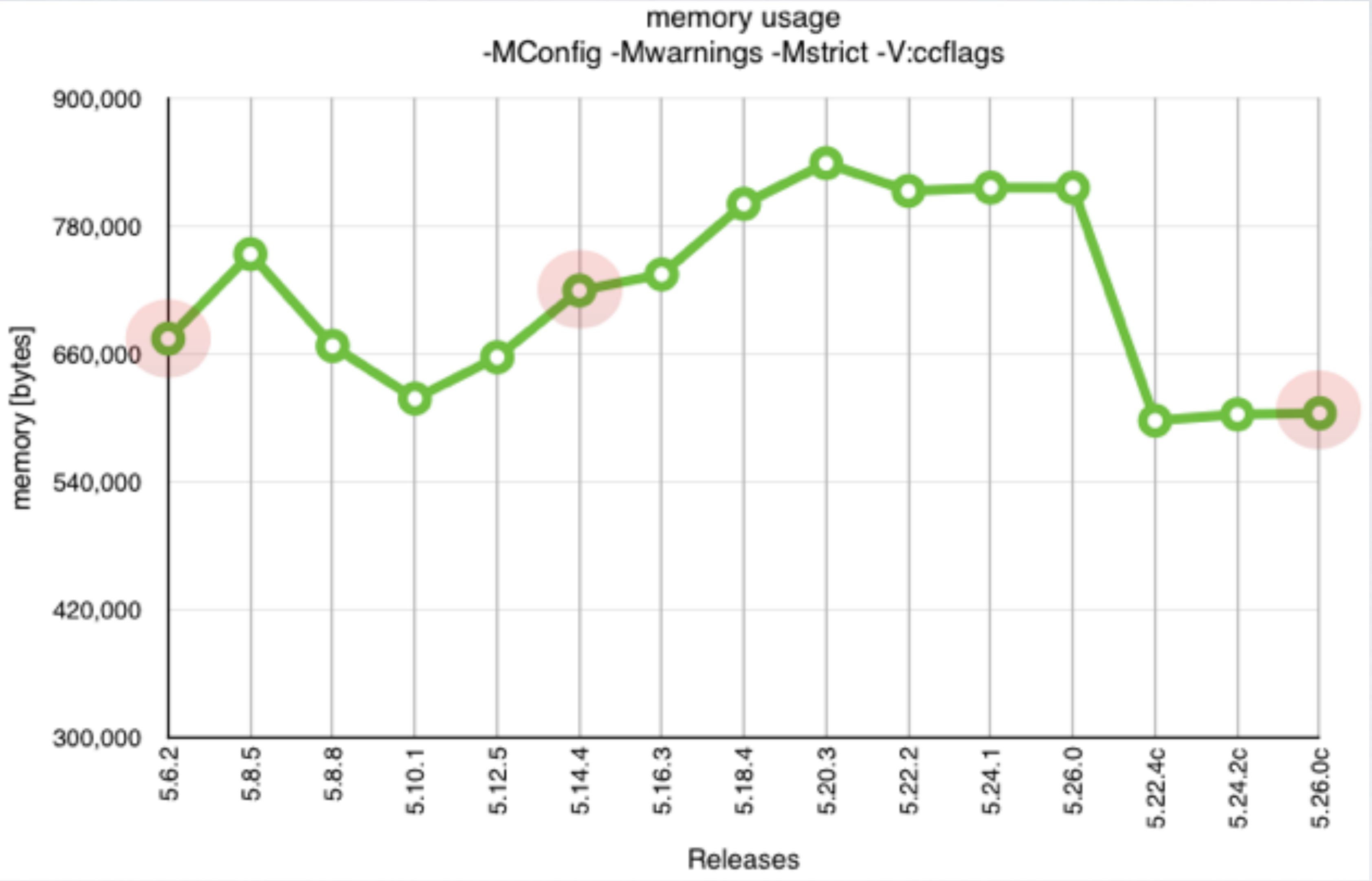
	5.14	5.22	5.22c	5.24	5.24c	5.26	5.26c	5.27c
hash/copy	100	112	108	110	117	117	122	128
hash/each	100	82	118	125	128	111	117	126
hash/foreach-sort	100	107	105	114	104	110	112	114
hash/foreach	100	121	119	124	123	123	108	128
hash/get	100	140	151	146	151	143	158	153
hash/set	100	116	140	117	137	121	142	139
loop/for-ary	100	118	130	202	213	161	206	246
loop/for-c	100	103	153	174	156	141	159	148
loop/for-range-const	100	115	117	141	99	142	92	147
loop/for-range	100	118	128	146	146	141	96	138
loop/getline	100	95	100	97	105	99	122	116
loop/while-my	100	117	142	151	105	152	108	147
loop/while	100	77	146	151	106	135	115	156
re/const	100	104	248	240	248	229	262	228
re/w	100	118	128	122	108	119	142	132
startup/fewmod	100	86	74	86	72	85	72	71
startup/lotsofsub	100	86	86	86	83	83	82	80
startup/noprog	100	103	107	104	107	106	102	102
string/base64	100	113	102	110	106	103	104	102
string/htmlparser	100	100	100	89	94	91	91	94
string/index-const	100	104	175	188	181	147	190	155
string/index-var	100	107	146	177	142	292	337	292
string/ipol	100	101	95	120	116	101	115	94
string/tr	100	102	104	101	106	118	125	123
Average	100.0	114.2	123.6	138.2	133.3	140.6	140.5	144.4
Arithmetic mean	100.0	115.8	126.6	142.5	137.8	146.1	148.6	151.1

Higher numbers are better. 200 is twice as fast as 100.

MEMORY



-MConfig -Mwarnings -Mstrict	
5.6.2	673653
5.8.5	753069
5.8.8	666789
5.10.1	617331
5.12.5	656194
5.14.4	718838
5.16.3	734030
5.18.4	800224
5.20.3	838428
5.22.2	812375
5.24.1	815414
5.26.0	815329
5.22.4c	596665
5.24.2c	602570
5.26.0c	603673



FEATURES

- support for modern compilers and features

e.g. new gcc 5, clang 3.4, icc intrinsics:

builtin_arith_overflow, builtin_clz, builtin_ctz, builtin_prefetch

FEATURES

With gcc-5 or clang-3.4 on 64bit new fast compiler intrinsics are used for the add and multiply ops, which check just for the overflow flag and then jump to the promotion to double. This results in smaller and much faster code. The old code which had to compute the results twice, and needed many more branches before to check for IV to UV or NV promotion.

builtin_arith_overflow

```
i_add:
  add %rax, %rbx
  jo +4

n_add:
  fadd %rax, %rbx
  jo +4
```

jo - jump if overflow

The recent p5p arithmetic improvements are 48% slower.

8%-25% less instructions and 20% less branches for add and multiply

FEATURES

builtin_clz ("count leading zeros")

hash splits are 10^8x faster. 100.000.000

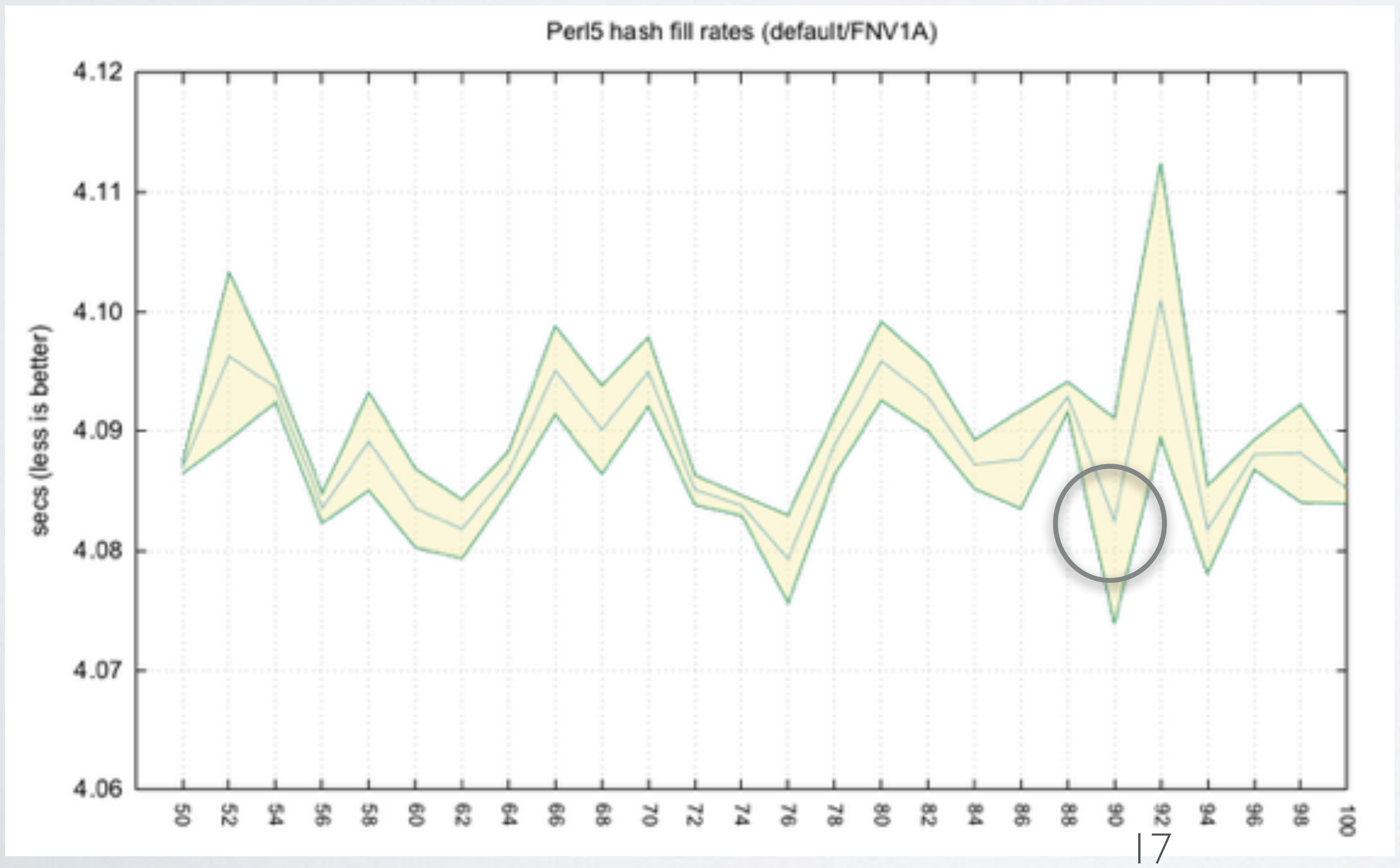
```
time clz:                        18 ns
time ceil(log2):         1060744672 ns
time perl5 core:         2904949213 ns
```

fallback if no _builtin_clz

FEATURES

builtin_ctz ("count trailing zeros")

allows -DHV_FILL_RATE=90 in DO_HSPLIT
perl5 can only do 100%



FEATURES

- support for modern compilers and features

-fsanitize=cfi, -flto, -flto=thin, -fstack-check, …

uses LD not CC for link (llvm)

re-introduced C enum’s, removed with 5.6.0

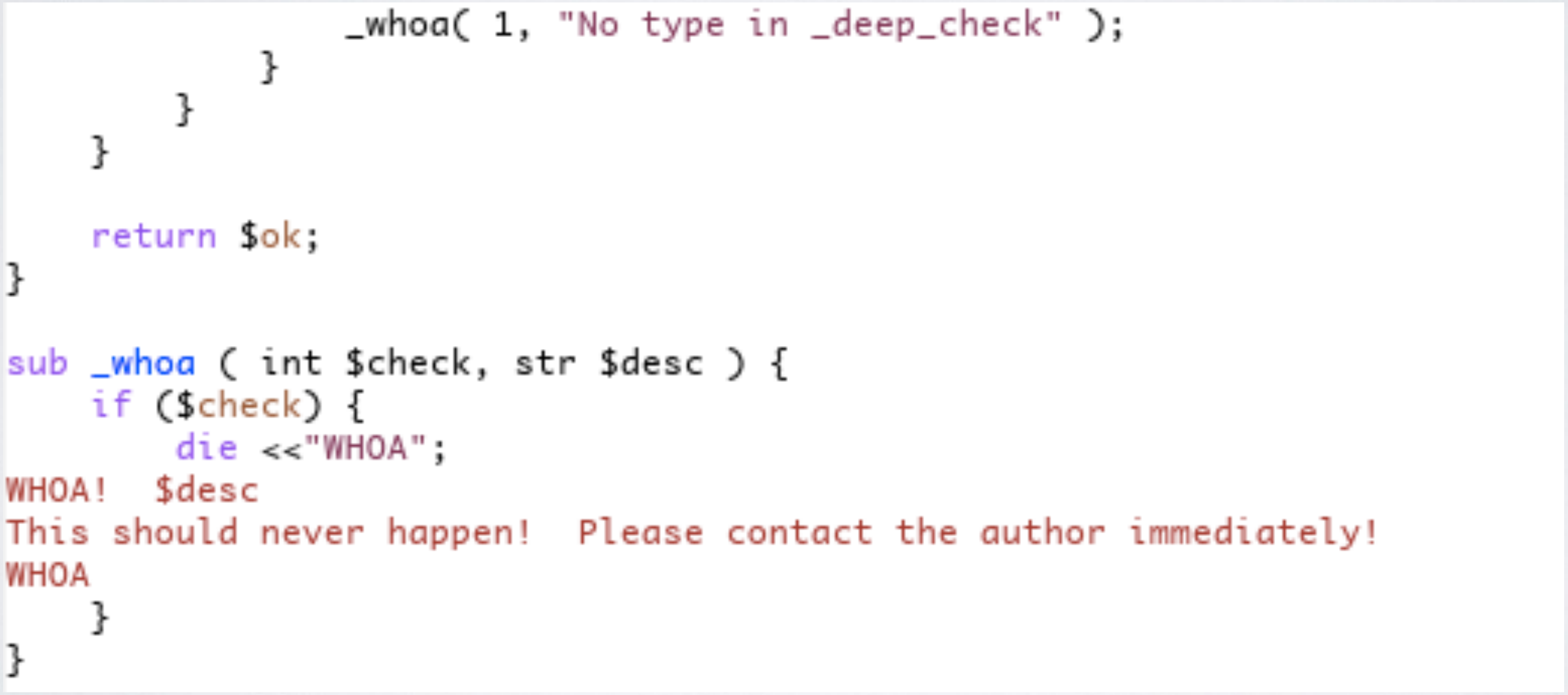
(perl6 enum later)

TYPES

- Compile-time checks and optimizations
- Faster not slower
- Safer not dynamic test-driven bug-hunting

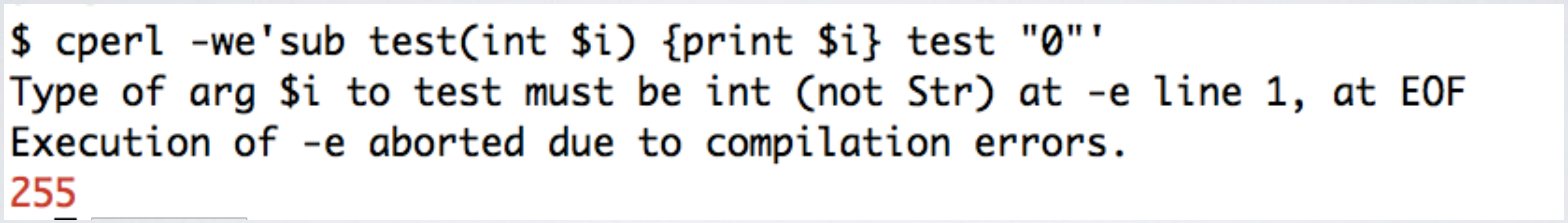
TYPES

dist/Test-Simple/lib/Test/More.pm:



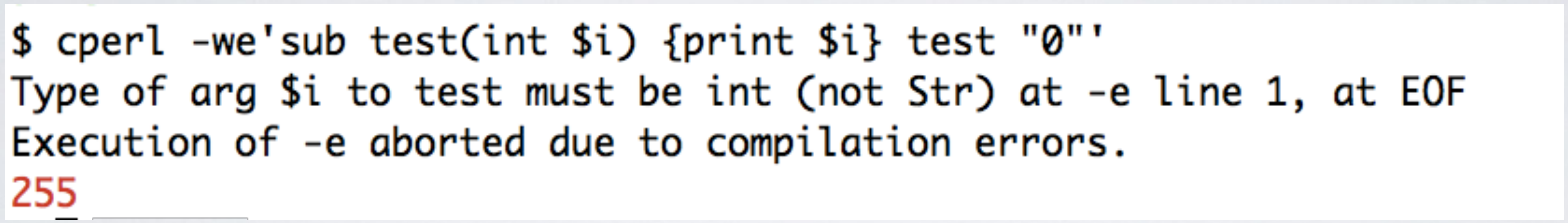
```
                _whoa( 1, "No type in _deep_check" );
            }
        }
    }

    return $ok;
}

sub _whoa ( int $check, str $desc ) {
    if ($check) {
        die <<"WHOA";
WHOA!  $desc
This should never happen!  Please contact the author immediately!
WHOA
    }
}
```

TYPES

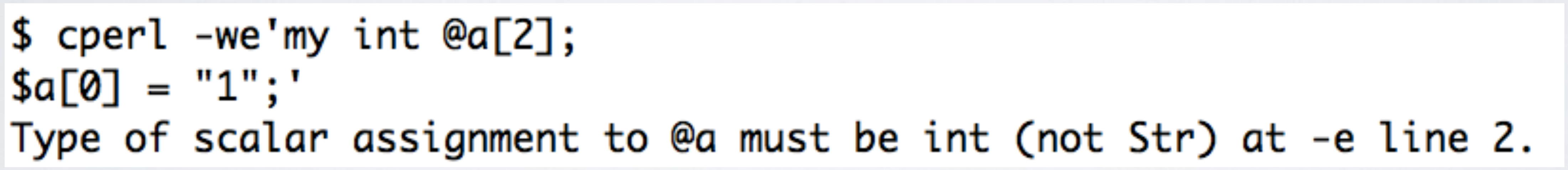


```
$ cperl -we'sub test(int $i) {print $i} test "0"'
Type of arg $i to test must be int (not Str) at -e line 1, at EOF
Execution of -e aborted due to compilation errors.
255
```

TYPES

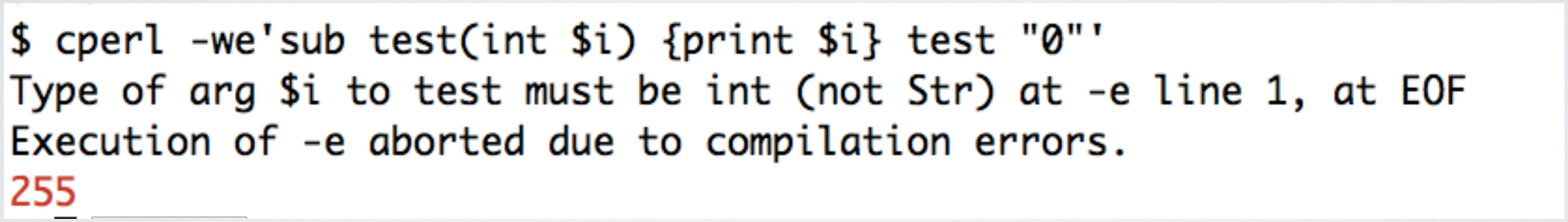


```
$ cperl -we'sub test(int $i) {print $i} test "0"'
Type of arg $i to test must be int (not Str) at -e line 1, at EOF
Execution of -e aborted due to compilation errors.
255
```

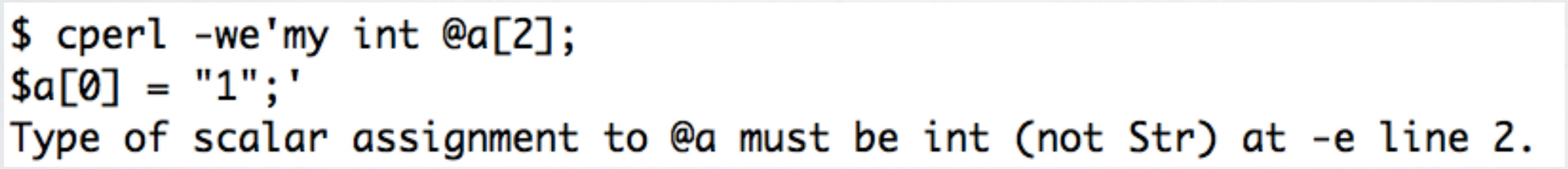


```
$ cperl -we'my int @a[2];
$a[0] = "1";'
Type of scalar assignment to @a must be int (not Str) at -e line 2.
```

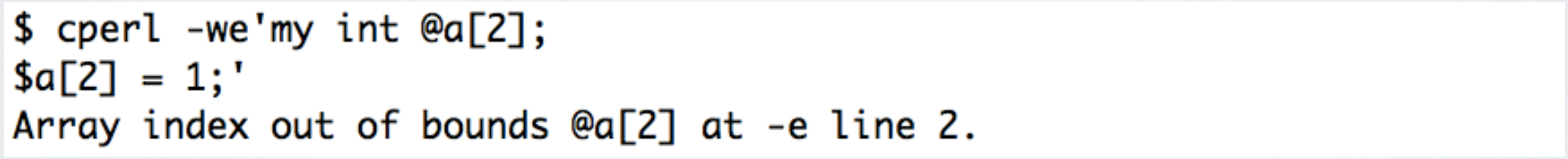
TYPES



```
$ cperl -we'sub test(int $i) {print $i} test "0"'
Type of arg $i to test must be int (not Str) at -e line 1, at EOF
Execution of -e aborted due to compilation errors.
255
```



```
$ cperl -we'my int @a[2];
$a[0] = "1";'
Type of scalar assignment to @a must be int (not Str) at -e line 2.
```



```
$ cperl -we'my int @a[2];
$a[2] = 1;'
Array index out of bounds @a[2] at -e line 2.
```

TYPES

Benefits:

* Modern Perl

* Compile-time safety

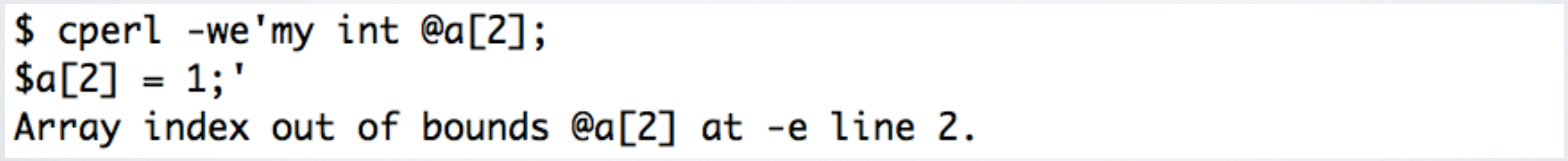
* 2x faster with signatures and types

SHAPED ARRAYS

* typed - compile-time checked

* remove run-time out-of-bounds checks

* optimized storage and usage with native types

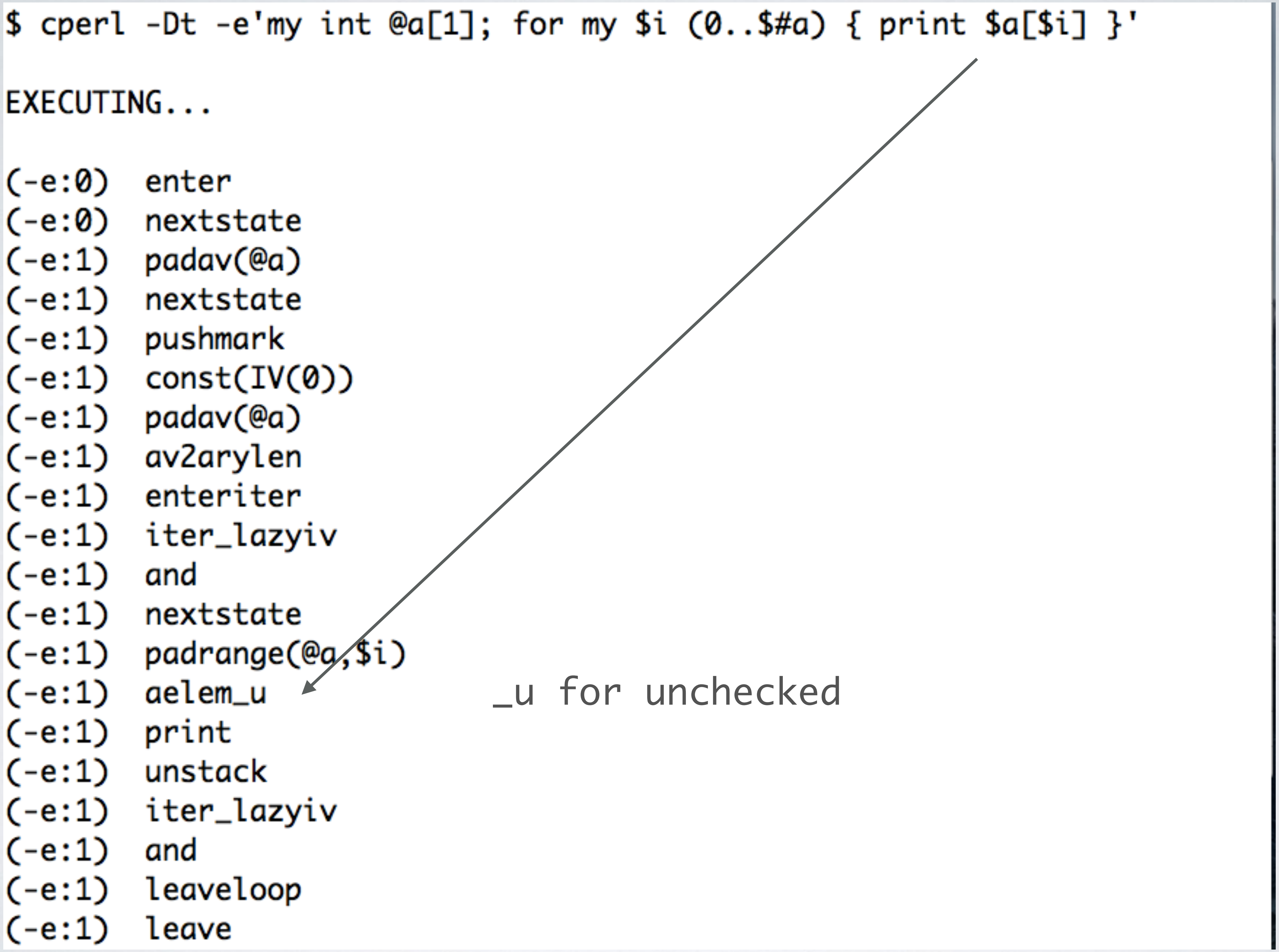


```
$ cperl -we'my int @a[2];
$a[2] = 1;'
Array index out of bounds @a[2] at -e line 2.
```

SHAPED ARRAYS

Static loop optimizations

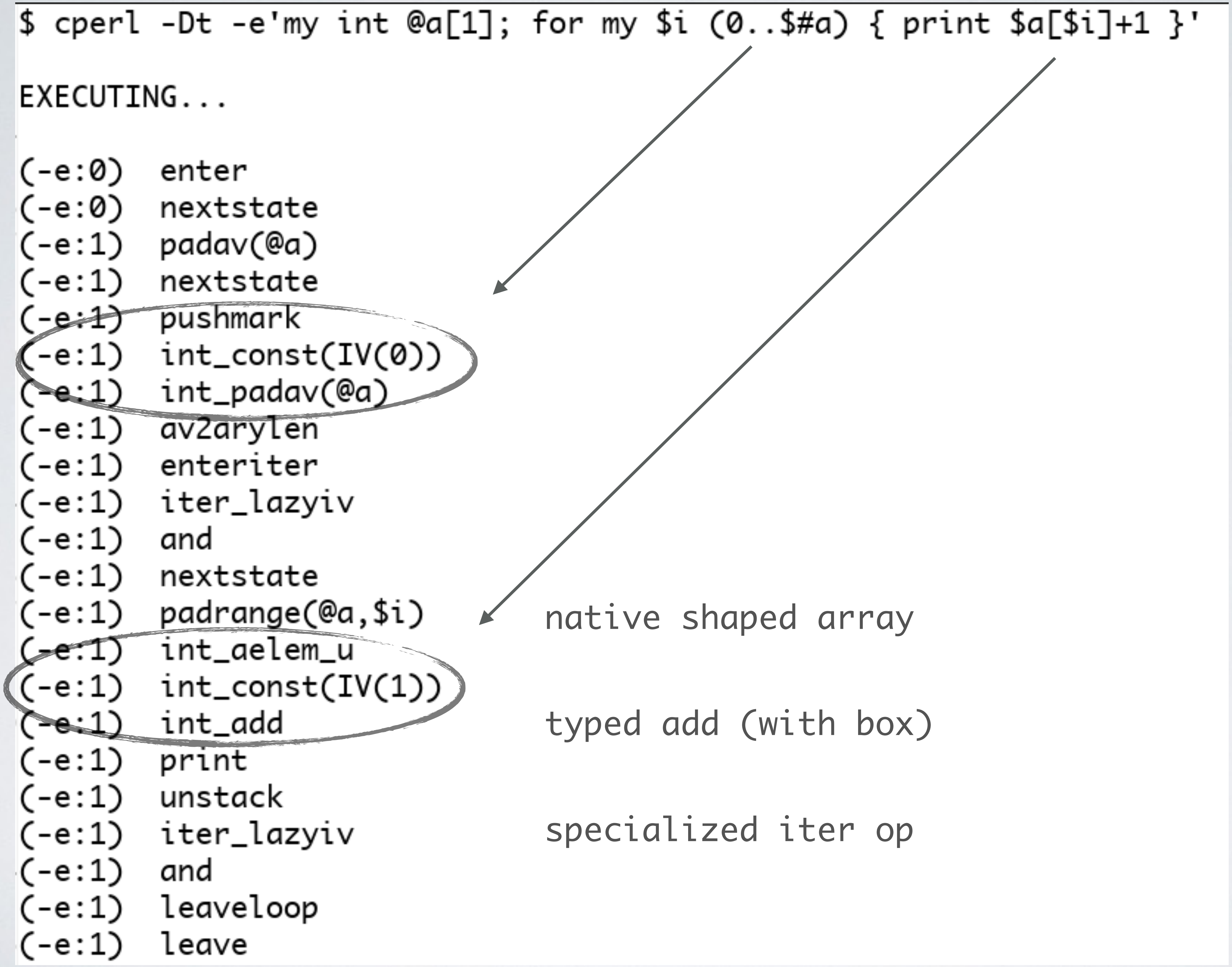
* eliminated run-time bounds checks



```
$ cperl -Dt -e'my int @a[1]; for my $i (0..$#a) { print $a[$i] }'

EXECUTING...

(-e:0)  enter
(-e:0)  nextstate
(-e:1)  padav(@a)
(-e:1)  nextstate
(-e:1)  pushmark
(-e:1)  const(IV(0))
(-e:1)  padav(@a)
(-e:1)  av2arylen
(-e:1)  enteriter
(-e:1)  iter_lazyiv
(-e:1)  and
(-e:1)  nextstate
(-e:1)  padrange(@a,$i)
(-e:1)  aelem_u
(-e:1)  print
(-e:1)  unstack
(-e:1)  iter_lazyiv
(-e:1)  and
(-e:1)  leaveloop
(-e:1)  leave
```

_u for unchecked

branch feature/gh14-native-types



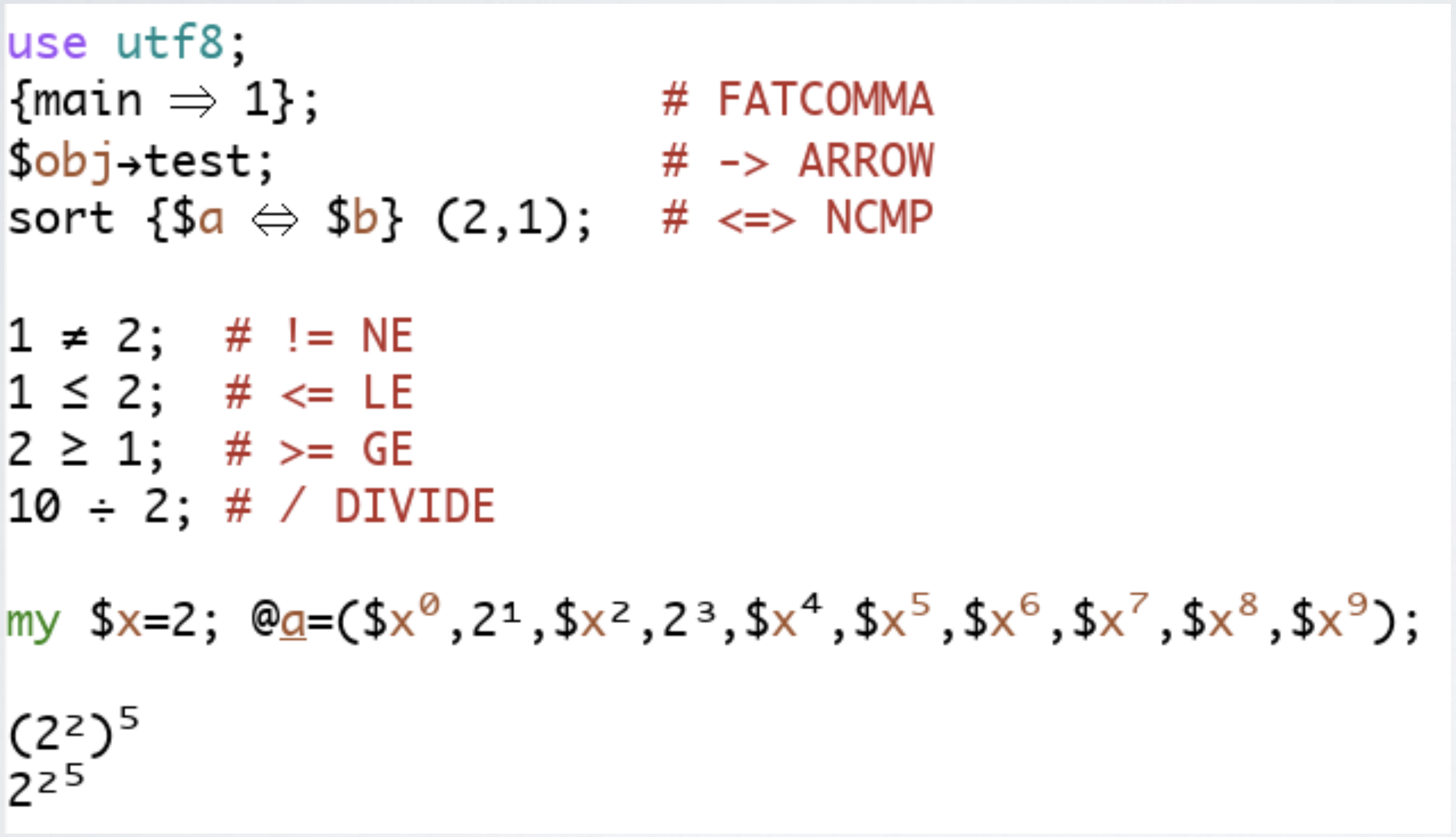
```
$ cperl -Dt -e'my int @a[1]; for my $i (0..$#a) { print $a[$i]+1 }'

EXECUTING...

(-e:0)  enter
(-e:0)  nextstate
(-e:1)  padav(@a)
(-e:1)  nextstate
(-e:1)  pushmark
(-e:1)  int_const(IV(0))
(-e:1)  int_padav(@a)
(-e:1)  av2arylen
(-e:1)  enteriter
(-e:1)  iter_lazyiv
(-e:1)  and
(-e:1)  nextstate
(-e:1)  padrange(@a,$i)
(-e:1)  int_aelem_u
(-e:1)  int_const(IV(1))
(-e:1)  int_add
(-e:1)  print
(-e:1)  unstack
(-e:1)  iter_lazyiv
(-e:1)  and
(-e:1)  leaveloop
(-e:1)  leave
```

native shaped array

typed add (with box)

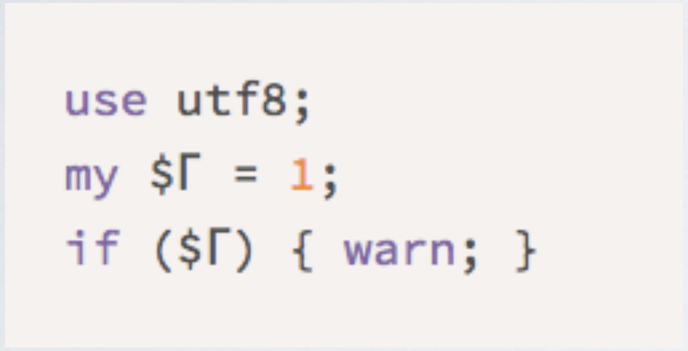
specialized iter op

UNICODE OPS



```
use utf8;
{main ⇒ 1};                # FATCOMMA
$obj→test;                 # -> ARROW
sort {$a ⇔ $b} (2,1);    # <=> NCMP

1 ≠ 2;  # != NE
1 ≤ 2;  # <= LE
2 ≥ 1;  # >= GE
10 ÷ 2; # / DIVIDE

my $x=2; @a=($x⁰,2¹,$x²,2³,$x⁴,$x⁵,$x⁶,$x⁷,$x⁸,$x⁹);

(2²)⁵
2²⁵
```

UNICODE SECURITY

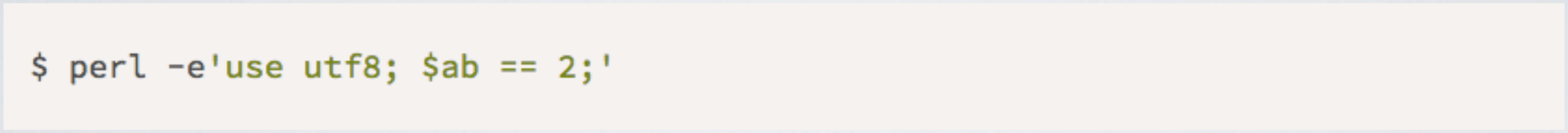


```perl
use utf8;
my $Γ = 1;
if ($Γ) { warn; }
```

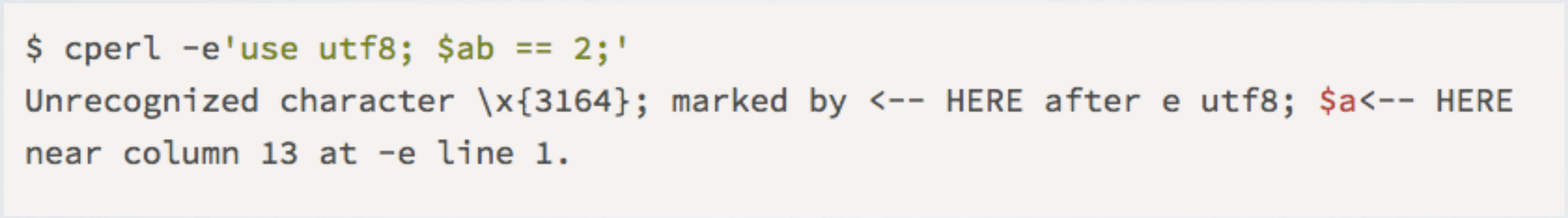
Greek
Cyrillic

```
Invalid script Cyrillic in identifier Γ) { warn }
for U+0413. Have Greek at -e line 1.
```

UNICODE SECURITY



```
$ perl -e'use utf8; $ab == 2;'
```

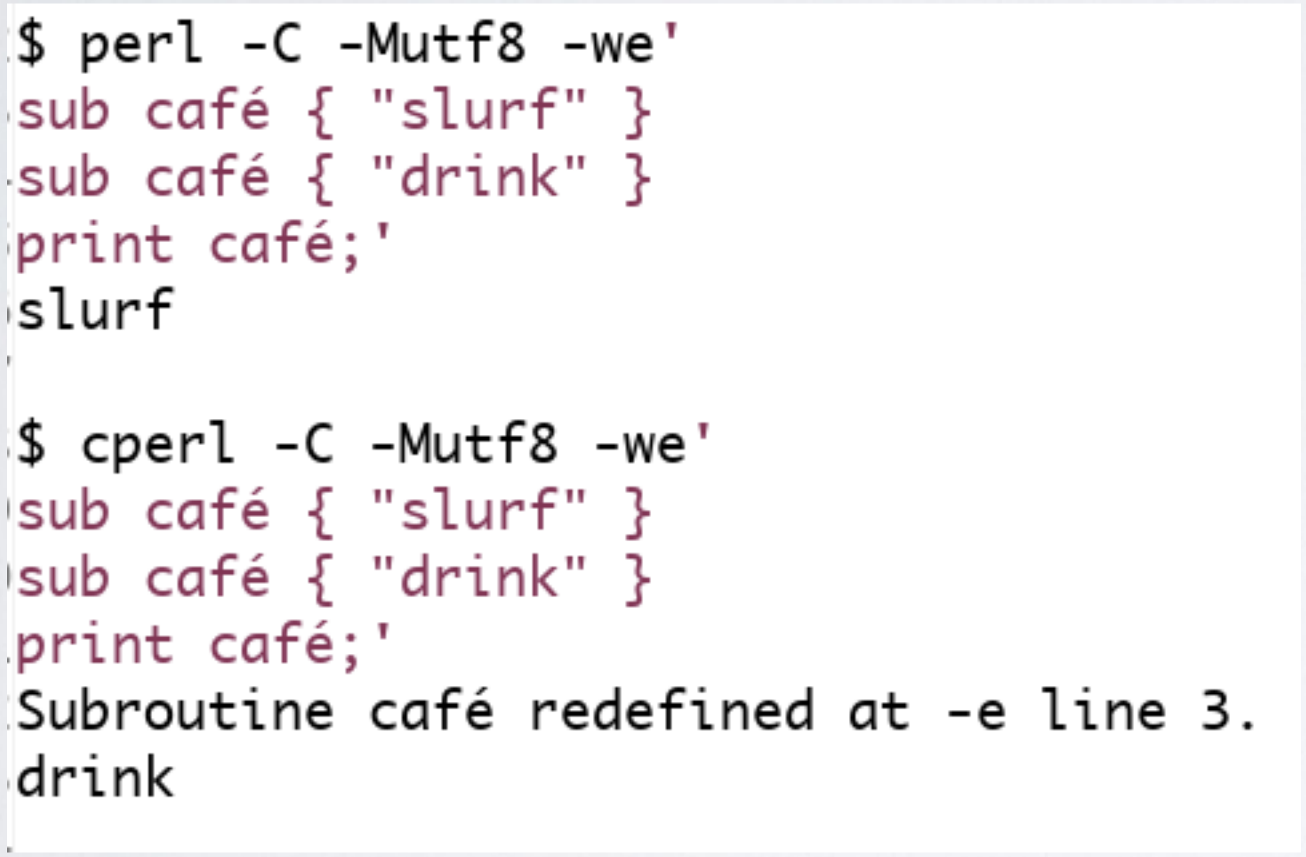


```
$ cperl -e'use utf8; $ab == 2;'
Unrecognized character \x{3164}; marked by <-- HERE after e utf8; $a<-- HERE
near column 13 at -e line 1.
```

Hangul Filler unicode bug

UNICODE SECURITY

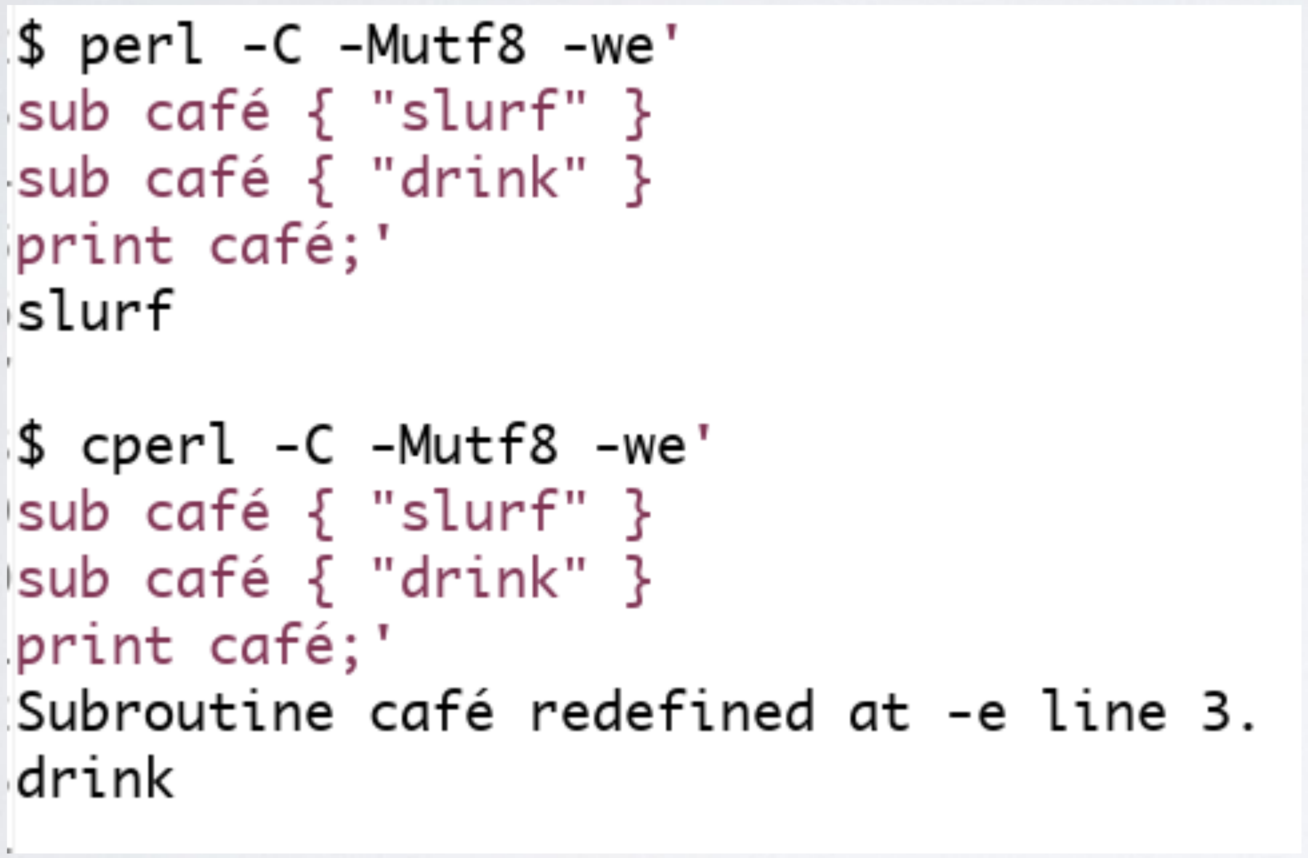
Identifier normalization



```
$ perl -C -Mutf8 -we'
sub café { "slurf" }
sub café { "drink" }
print café;'
slurf

$ cperl -C -Mutf8 -we'
sub café { "slurf" }
sub café { "drink" }
print café;'
Subroutine café redefined at -e line 3.
drink
```

UNICODE SECURITY

Identifier normalization

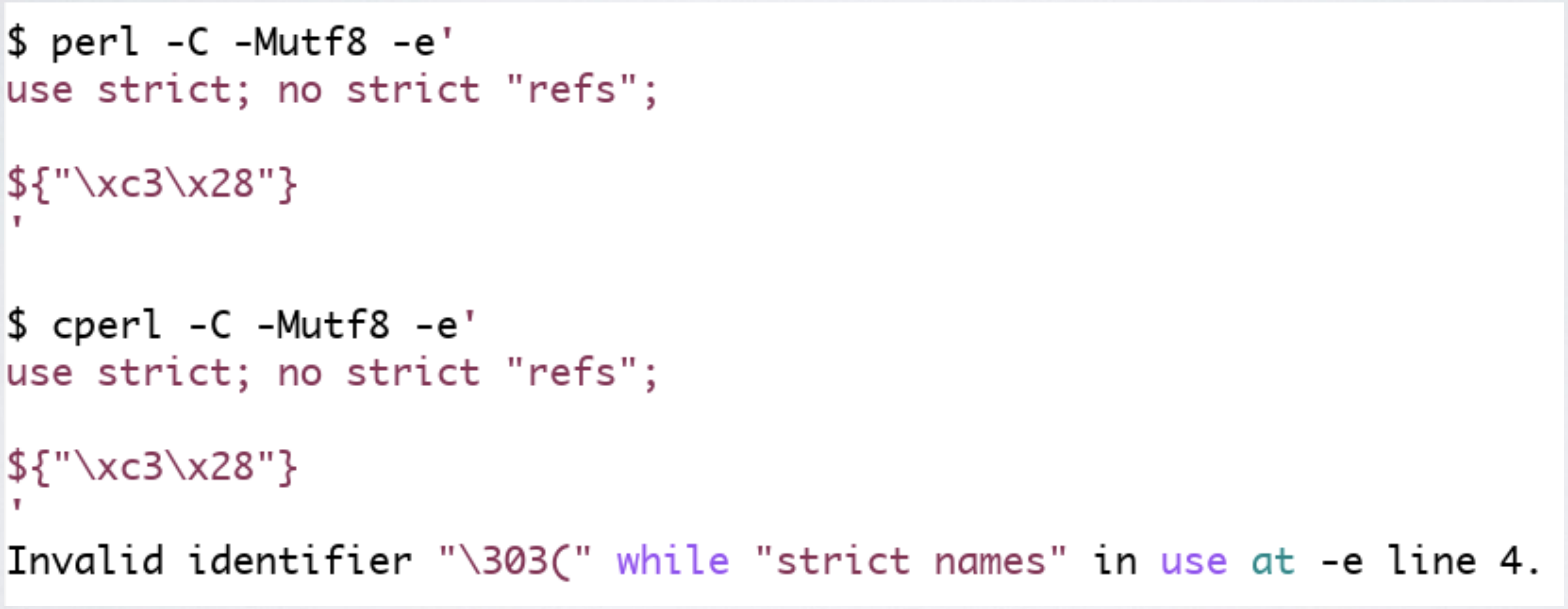


```
$ perl -C -Mutf8 -we'
sub café { "slurf" }
sub café { "drink" }
print café;'
slurf

$ cperl -C -Mutf8 -we'
sub café { "slurf" }
sub café { "drink" }
print café;'
Subroutine café redefined at -e line 3.
drink
```

```
<c, a, f, e, U+0301>
<c, a, f, U+00E9>


<c, a, f, e, U+0301>
normalized to <c, a, f, U+00E9>
```

UNICODE SECURITY

strict ‘names’



```
$ perl -C -Mutf8 -e'
use strict; no strict "refs";

${"\xc3\x28"}
'

$ cperl -C -Mutf8 -e'
use strict; no strict "refs";

${"\xc3\x28"}
'
Invalid identifier "\303(" while "strict names" in use at -e line 4.
```

Enforce parsable identifiers

MODERN TOOLCHAIN

p5p is not able to maintain the core modules

Critical fixes in:

- Storable
- ExtUtils-Constant, threads, …

Modernized:

- Test-Simple, Pod-Simple, Pod-HTML, PathTools, bigint/bignum/bigrat, c2ph

Fast XS in core:

- Cpanel::JSON::XS, YAML::XS, strict, attributes, DynaLoader, Config, XSLoader, Devel::NYTProf

MODERN TOOLCHAIN

Native support for the ‘c’ version suffix, builtin XS modules, cperl extensions, improved perf. + safety on most other toolchain modules (CPAN, YAML, …)

```
$ cperl -e'print $^V'
v5.27.1c

$^V =~ /c$/            # cperl detection
$Config{usecperl}   # cperl detection
```

MODERN TOOLCHAIN

Test2 is currently being blocked.

It is a good idea to improve streaming of subtests, but only if
* it’s not >20% slower,
* it would be optional, not mandatory.

It is in work to get into acceptable performance by modernization of the API (faster via signatures), but still too slow.

Previously you could at least trust CPAN, not p5p, but recently even CPAN eroded:
Module::Build, YAML, Moose, Scalar::Utils, EUMM, Test::Simple

OBJECTS

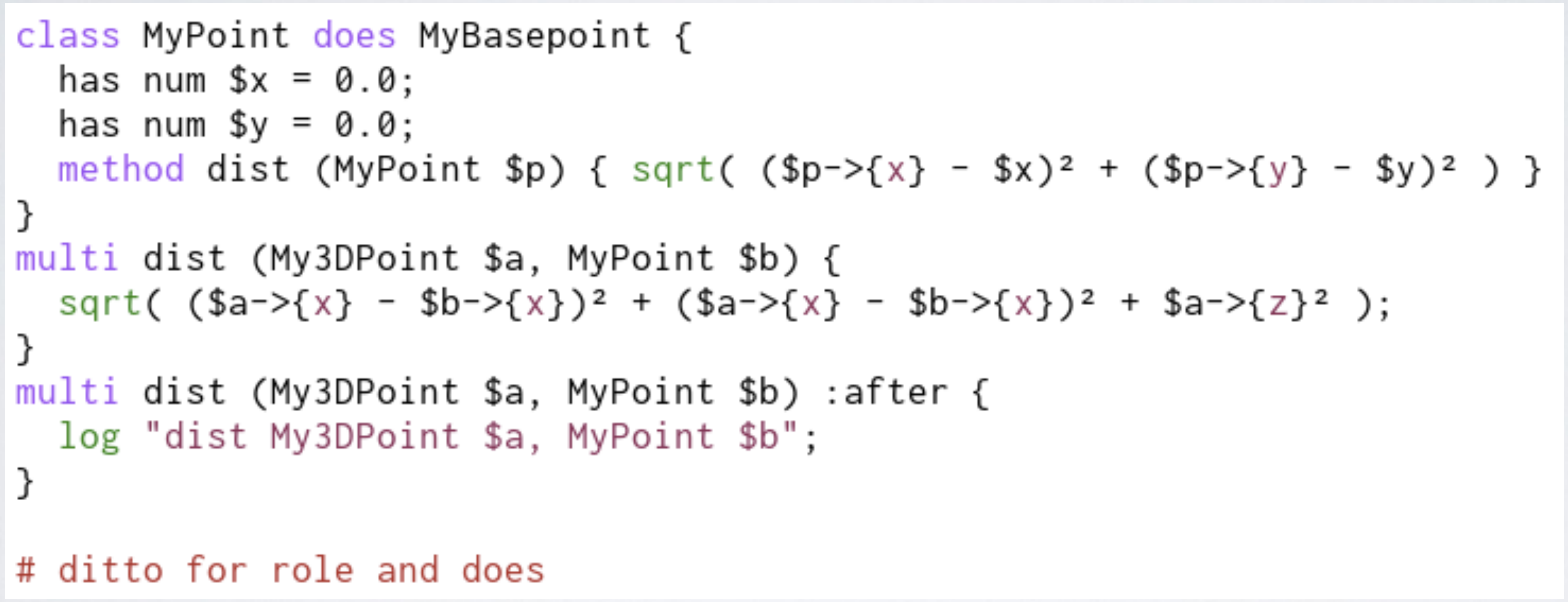
Recommended was **Mouse** or use base/fields with pseudohashes.

cperl classes, roles look like perl6, and behave like Mouse. Definitely no Moxie %HAS, still %FIELDS.

The type system is fast and integrated. Several perl6 design bugs are fixed.

But no safe structural typing, no sound types, very weak type inference.

OBJECTS



```
class MyPoint does MyBasepoint {
  has num $x = 0.0;
  has num $y = 0.0;
  method dist (MyPoint $p) { sqrt( ($p->{x} - $x)² + ($p->{y} - $y)² ) }
}
multi dist (My3DPoint $a, MyPoint $b) {
  sqrt( ($a->{x} - $b->{x})² + ($a->{x} - $b->{x})² + $a->{z}² );
}
multi dist (My3DPoint $a, MyPoint $b) :after {
  log "dist My3DPoint $a, MyPoint $b";
}

# ditto for role and does
```

OBJECTS

Builtin ffi *(NativeCall is taken with worse syntax)*

```
class MyPoint :native {
  has int $x;
  has int $y;
}
extern sub gtk2_draw (MyPoint $p);
```

Sugar for the also supported perl6 syntax:

```
class MyPoint is repr(CStruct) {
  has int $x;
  has int $y;
}
sub gtk2_draw (MyPoint $p) :native;
```

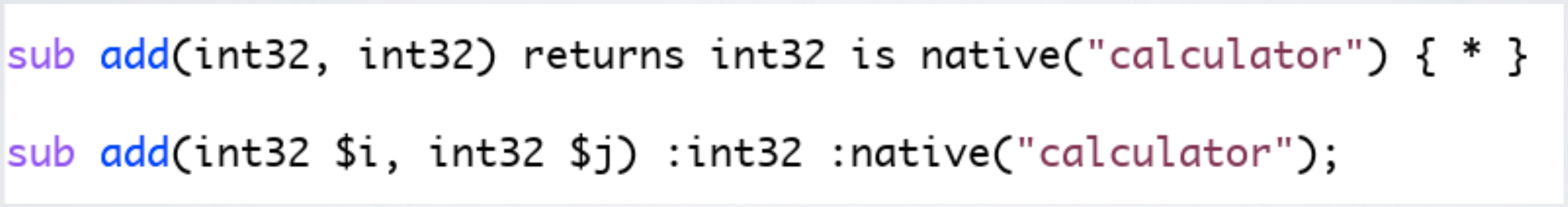
ATTRIBUTES

Lots of builtin attributes

- :const for all data types
- :pure for functions
- coretypes for signatures and as function return type
- user-types for signatures and as function return type
- safe run-time attribute values
  (no Attribute::Handler eval security risk)

ATTRIBUTES

cperl uses attributes for everything perl6 does with traits.

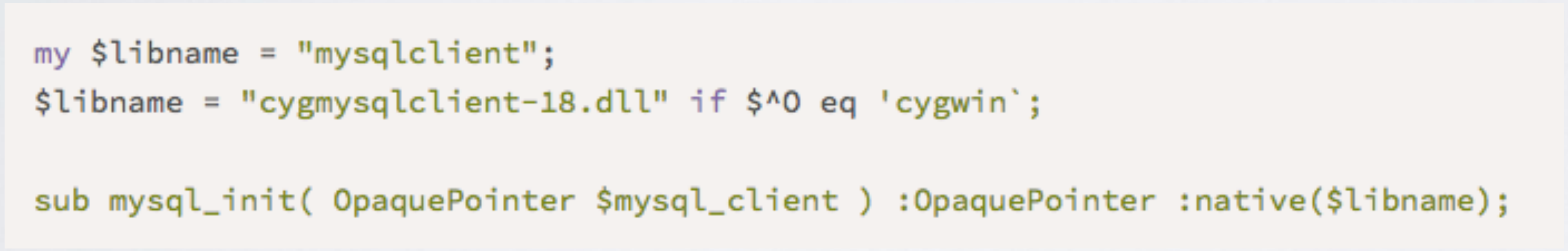


```
sub add(int32, int32) returns int32 is native("calculator") { * }

sub add(int32 $i, int32 $j) :int32 :native("calculator");
```

ATTRIBUTES

cperl fixes perl5 attributes, and improves on perl6 traits.

Even perl6 has the design bug, that trait arguments are compile-time only.



```
my $libname = "mysqlclient";
$libname = "cygmysqlclient-18.dll" if $^O eq 'cygwin`;

sub mysql_init( OpaquePointer $mysql_client ) :OpaquePointer :native($libname);
```

The FFI needs application logic at run-time to find a DLL.

Defer the `attributes->import` call to run-time, as with `my $var :attr;` declarations.

BUGFIXES

Worst perl5 bugs

BUGFIXES

Worst perl5 bugs

Denial of Service

It's trivial to DoS a perl5 system.

```
$a[9223372036854]=0;

%a=(0..4294967296);
```

Examples for a 64bit system, but also trivial on 32bit.
It creates a huge array or hash, which runs out of memory in the VMM subsystem which eventually kills the process.
cperl dies with "Too many elements", here even at compile-time.

BUGFIXES

Worst perl5 bugs

No Hash Security

Only security theatre with “secure hash functions” and slower collision resolution KEY_PERTURB_RANDOM. Secure hash functions for hash tables do not exist!
Hash function security starts with 256 bits, we have 32.
Only collision resolution can protect a hash table properly.
Even siphash is brute forcable (just 2x slower)

```
$ PERL_HASH_SEED_DEBUG=1 perl -e1
  HASH_FUNCTION = ONE_AT_A_TIME_HARD HASH_SEED = 0x5d5e02649c4f1e25
  PERTURB_KEYS = 1 (RANDOM)

$ PERL_HASH_SEED_DEBUG=1 cperl -e1
  HASH_FUNCTION = FNV1A HASH_SEED = <hidden> PERTURB_KEYS = 0 (TOP)
```

BUGFIXES

Worst perl5 bugs

Silent integer overwraps

```
  @a=(0,1); print $a[~1] => 0
```

`~1` is essentially `(UV)-2` or `0xffff_fffe`.

```
  @a=(1); print $a[18446744073709551615]' => 1
```

Silent overwrap of 18446744073709551615 to -1.

```
$ cperl -e'@a=(1); print $a[18446744073709551615]'
Too many elements at -e line 1.
```

BUGFIXES

Worst perl5 bugs

Inconsistencies with over-long data (I32 - U64)

* Silent hash or array overflows.
* chop/chomp only works on half of overlarge arrays.
* hash keys > I32 silently truncated, not converted to SV or error.
* hash insertion allows >I32, but iteration only I32.
* Storable cannot handle >I32
* ~”a"x2G complement silently truncated.
* smartmatch Array - CodeRef processes only over
half the array elements.
* regex match group of >2GB string len.
* repeat count >2GB and don't silently cap it at IV_MAX. Which was
at least better then silent wrap around.

Most of them security relevant.

BUGFIXES

Worst perl5 bugs

Names

* Silently introduces binary names, without any protection. Called "harmonization".
* Insecure unicode names
* Overlong names (silently truncated at I32_MAX)
* Same for file, keyword, mro, stash names.

More symbol table nonsense:
autovivification was for decades considered a bug, eventually fixed.
Now it’s declared a feature. Searching a name will create the name by perl5 magic side effect.
Will be fixed in cperl eventually, with the symbol table rewrite for a single flat table.

BUGFIXES

Worst perl5 bugs

Insecure taint mode

Bugzilla is plagued for years by the existing taint loopholes.
You cannot trust perl5 tainting.
They should just switch over to cperl to be safe again.

See `perldoc perlsec` vs `cperldoc perlsec`
Hash keys keep the tainted bit, regex capturing is re ‘taint’ safe.

BUGFIXES

Worst perl5 bugs

Lexical `my $_`

An important feature for nested loops, iterators, given/when rules, and much faster than global `$_` lookup.

Fixed in cperl, removed as “*unfixable*” in perl5.

BUGFIXES

Worst perl5 bugs

```
for (qw(a b c)) {}
```

p5p thinks that this new feature is better than the old

```
  for qw(a b c) {}
```

cperl allows `for qw(a b c) {}` again.

It’s only ~10 lines in the parser.

BUGFIXES

Worst perl5 bugs

The COW system

Copy-on-write vs Copy-on-grow

perl5 does nothing of those.
It would be great to have at least one these.

B::C supports both, but has to disable perl5 COW for increasing memory usage by 20% since 5.18.

It’s also unmaintainable. I’ve tried to rewrite it twice.

BUGFIXES

Worst perl5 bugs

The double readonly system, SVf_PROTECT

Was trivial to fix in cperl. No need to check for 2 bits for readonly-ness, and compile-time optimizations harmonize with the run-time.

The root cause in perl5 is still broken though.

IN WORK

IN WORK

A lot!

Unlike perl5 cperl is actively developed

IN WORK

A lot!

Unlike perl5 cperl is actively developed

And their unofficial TODO list is frightening:

* refcounted stack
* hash table via vtable (one more indirection)
* cow rewrite

IN WORK

- class/role/multi/method/has

IN WORK

- class/role/multi/method/has
- ffi

IN WORK

- class/role/multi/method/has
- ffi
- native types

IN WORK

- class/role/multi/method/has
- ffi
- native types
- inline subs

IN WORK

- class/role/multi/method/has
- ffi
- native types
- inline subs
- hash table rewrite (open addressing)

IN WORK

- class/role/multi/method/has
- ffi
- native types
- inline subs
- hash table rewrite (open addressing)
- no miniperl

…

IN WORK

- unexec

IN WORK

- unexec
- lazy parsing

IN WORK

- unexec
- lazy parsing
- llvm jit

IN WORK

- unexec
- lazy parsing
- llvm jit
- flat symbol table (gvlinear)

IN WORK

- unexec
- lazy parsing
- llvm jit
- flat symbol table (gvlinear)
- warnings and Carp as builtin XS

IN WORK

- unexec
- lazy parsing
- llvm jit
- flat symbol table (gvlinear)
- warnings and Carp as builtin XS
- unicode tables as XS

IN WORK

- unexec
- lazy parsing
- llvm jit
- flat symbol table (gvlinear)
- warnings and Carp as builtin XS
- unicode tables as XS
- fast regex compiler (hyperscan + pcre2-jit)

IN WORK

- unexec
- lazy parsing
- llvm jit
- flat symbol table (gvlinear)
- warnings and Carp as builtin XS
- unicode tables as XS
- fast regex compiler (hyperscan + pcre2-jit)

… ~90 feature branches

POLICIES

POLICIES

- No worst practices as in p5p

POLICIES

- No worst practices as in p5p
- Best practices as in parrot, perl6

POLICIES

- No worst practices as in p5p
- Best practices as in parrot, perl6
- New branch rebase policy to keep all work branches up to date

POLICIES

- No worst practices as in p5p
- Best practices as in parrot, perl6
- New branch rebase policy to keep all work branches up to date
- No tragedy of the commons by failing to pick votes for backports

POLICIES

- No worst practices as in p5p
- Best practices as in parrot, perl6
- New branch rebase policy to keep all work branches up to date
- No tragedy of the commons by failing to pick votes for backports
- No death by code of conduct

POLICIES

- No worst practices as in p5p
- Best practices as in parrot, perl6
- New branch rebase policy to keep all work branches up to date
- No tragedy of the commons by failing to pick votes for backports
- No death by code of conduct
- No promotion of bad behavior as in p5p or YAPC

QUESTIONS