Towards federated implementations of high-level languages

Stephen Kell
s.r.kell@kent.ac.uk

School of Computing

University of Kent
HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)

**SITUATION:** THERE ARE 14 COMPETING STANDARDS.

**14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.** YEAH!

**SITUATION:** THERE ARE 15 COMPETING STANDARDS.
What is federated implementation?

I say two or more language implementations are federated if

- they can coexist within one address space
- each hosting some code
- capable of mutual interaction
- with no programmer-visible boundary
$ cat >main.c <<EOF
int hello(void);
int main(void) { return hello(); } 
EOF
$ cat >hello.c <<EOF
#include <stdio.h>
int hello(void) { puts("Hello"); return 0; } 
EOF
$ gcc -c -o main.o main.c
$ clang -c -o hello.o hello.c
$ cc -o hello hello.o main.o # any cc...
$ ./hello
Federation is a strong interoperability property.

C is usually implemented in a federated fashion.

High-level languages are never (?) federated.

- ‘high-level’ \(\approx\) ‘with GC’

Of particular interest: *polyglot federation*

- many implementations... of *different* languages
Why should language implementations be federated?

Isn’t interoperation already possible, without federation?

- there are foreign function interfaces (FFIs)!

Short answer: check your Stockholm syndrome

Longer answer: it’s the economics, silly…
What’s wrong with FFIs? A very partial list

FFIs + tools/APIs

- only mylang-to-C
- non-trivial binding effort
- tolerable for the most popular libraries...
- ... doesn’t scale beyond that
- repetitive, fragile
- antipattern bingo
“Target an interface, not an implementation”

Local<Value> GetPointX(Local<String> property,
                         constAccessorInfo &info) {
    Local<Object> self = info.Holder();
    Local<External> wrap = Local<External>::Cast(self->GetInternalField
    void* ptr = wrap->Value();
    int value = static_cast<Point*>(ptr)->x_;
    return Integer::New(value);
}

void SetPointX(Local<String> property, Local<Value> value,
               constAccessorInfo& info) {
    Local<Object> self = info.Holder();
    Local<External> wrap = Local<External>::Cast(self->GetInternalField
    void* ptr = wrap->Value();
    static_cast<Point*>(ptr)->x_ = value->Int32Value();
}
FFIs are the ALGs of language interoperability

Pre-Internet, sending e-mail across networks was possible

- if the right gateways were available + running

Deploying new applications usu. economically infeasible...
The Internet Protocol’s hourglass design

… made application-layer gatewaying unnecessary

- huge economic benefit!
- hypothesis: similar potential in language adoption
An hourglass waist: IP

Embrace diversity: many \{applications, networks\}

Illusion of uniformity: \textit{addressing}, datagrams

Support evolutionary adoption

To avoid gateways, must allow \textit{transfer of meaning}…

… ‘just enough’: end-to-end suffices
An hourglass waist for language implementations?

Embrace diversity
- many languages, implementations

Illusion of uniformity

Support evolutionary adoption

To avoid gateways, must allow *transfer of meaning*…

… ‘just enough’: end-to-end suffices
An hourglass waist for language implementations?

Embrace diversity

- many languages, implementations

Illusion of uniformity

- some shared ‘object’ / ‘value’ notion?

Support evolutionary adoption

To avoid gateways, must allow *transfer of meaning*…

… ‘just enough’: end-to-end suffices
An hourglass waist for language implementations?

Embrace diversity

- many languages, implementations

Illusion of uniformity

- some shared ‘object’ / ‘value’ notion?

Support evolutionary adoption

- don’t throw away existing language impls

To avoid gateways, must allow transfer of meaning…

… ‘just enough’: end-to-end suffices
An hourglass waist for language implementations?

Embrace diversity
- many languages, implementations

Illusion of uniformity
- some shared ‘object’ / ‘value’ notion?

Support evolutionary adoption
- don’t throw away existing language impls

To avoid gateways, must allow *transfer of meaning*…
- ‘object’ or ‘value’ must carry some semantics

… ‘just enough’: end-to-end suffices
- what is better left out?
Recap

- many languages, implementations
- some shared ‘object’ / ‘value’ notion?
- don’t throw away existing language impls
- ‘object’ or ‘value’ must carry some semantics
- … but maybe not everything

Unix-style ‘opaque data’ isn’t going to work
  - it’s what got us into this mess!

But evolving Unix seems essential
  - $\approx$ all language impls target Unix-like abstractions
Evolving Unix: memory mappings

```
$ cat /proc/self/maps
55f0bf200000 -55f0bf208000  r-xp 00000000 103:02 20578329 /bin/cat
55f0bf407000 -55f0bf408000  r--p 00007000 103:02 20578329 /bin/cat
55f0bf408000 -55f0bf409000  rw-p 00008000 103:02 20578329 /bin/cat
55f0bf68b000 -55f0bf6ac000  rw-p 00000000 00:00  0 [heap]
7f5f32bde000 -7f5f335ad000  r--p 00000000 103:02 30546546 /usr/lib/locale/locale-archive
7f5f335ad000 -7f5f33794000  r-xp 00000000 103:02 23728700 /lib/x86_64-linux-gnu/libc-2.27.so
7f5f33794000 -7f5f33994000  ---p 001e7000 103:02 23728700 /lib/x86_64-linux-gnu/libc-2.27.so
7f5f33994000 -7f5f33998000  r--p 001e7000 103:02 23728700 /lib/x86_64-linux-gnu/libc-2.27.so
7f5f33998000 -7f5f3399a000  rw-p 001eb000 103:02 23728700 /lib/x86_64-linux-gnu/libc-2.27.so
7f5f3399a000 -7f5f3399e000  rw-p 00000000 00:00  0 [heap]
7f5f3399e000 -7f5f339c5000  r-xp 00000000 103:02 23728672 /lib/x86_64-linux-gnu/ld-2.27.so
7f5f33b9e000 -7f5f33bc5000  rw-p 00000000 00:00  0
7f5f33bc5000 -7f5f33bc6000  r--p 00270000 103:02 23728672 /lib/x86_64-linux-gnu/ld-2.27.so
7f5f33bc6000 -7f5f33bc7000  rw-p 00280000 103:02 23728672 /lib/x86_64-linux-gnu/ld-2.27.so
7f5f33bc7000 -7f5f33bc8000  rw-p 00000000 00:00  0 [stack]
7ffde13f1000 -7ffde1413000  rw-p 00000000 00:00  0 [vvar]
7ffde143d000 -7ffde1440000  r--p 00000000 00:00  0 [vdso]
7ffde1440000 -7ffde1442000  r-xp 00000000 00:00  0 [vsyscall]
ffffffffffffff6010000-ffffffffffffff601000 r-xp 00000000 00:00  0 [heap]
```
Evolving Unix: memory mappings are allocations

```bash
$ cat /proc/self/maps
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Size</th>
<th>Type</th>
<th>Address</th>
<th>Size</th>
<th>Address</th>
<th>Size</th>
<th>Mode</th>
<th>R/W</th>
<th>Offset</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>55f0bf200000</td>
<td>-55f0bf208000</td>
<td>r-xp</td>
<td>000000000</td>
<td>103:02</td>
<td>20578329</td>
<td>/bin/cat</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55f0bf407000</td>
<td>-55f0bf408000</td>
<td>r-p</td>
<td>00007000</td>
<td>103:02</td>
<td>20578329</td>
<td>/bin/cat</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55f0bf408000</td>
<td>-55f0bf409000</td>
<td>rw-p</td>
<td>00008000</td>
<td>103:02</td>
<td>20578329</td>
<td>/bin/cat</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55f0bf68b000</td>
<td>-55f0bf6ac000</td>
<td>rw-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>[heap]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f5f32bde00</td>
<td>-7f5f335ad000</td>
<td>r-p</td>
<td>00000000</td>
<td>103:02</td>
<td>30546546</td>
<td>/usr/lib/locale/locale-archive</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f5f335ad000</td>
<td>-7f5f33794000</td>
<td>r-xp</td>
<td>00000000</td>
<td>103:02</td>
<td>23728700</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f5f33794000</td>
<td>-7f5f33994000</td>
<td>r-p</td>
<td>001e7000</td>
<td>103:02</td>
<td>23728700</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f5f33994000</td>
<td>-7f5f33998000</td>
<td>r-p</td>
<td>001e7000</td>
<td>103:02</td>
<td>23728700</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f5f33998000</td>
<td>-7f5f3399a000</td>
<td>rw-p</td>
<td>001eb000</td>
<td>103:02</td>
<td>23728700</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f5f3399a000</td>
<td>-7f5f3399e000</td>
<td>rw-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f5f339e000</td>
<td>-7f5f339c5000</td>
<td>r-xp</td>
<td>00000000</td>
<td>103:02</td>
<td>23728672</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f5f33b9e000</td>
<td>-7f5f33bc5000</td>
<td>rw-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f5f33bc5000</td>
<td>-7f5f33bc6000</td>
<td>r-p</td>
<td>00027000</td>
<td>103:02</td>
<td>23728672</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f5f33bc6000</td>
<td>-7f5f33bc7000</td>
<td>rw-p</td>
<td>00028000</td>
<td>103:02</td>
<td>23728672</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7f5f33bc7000</td>
<td>-7f5f33bc8000</td>
<td>rw-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7ffde13f1000</td>
<td>-7ffde1413000</td>
<td>rw-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>[stack]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7ffde143d000</td>
<td>-7ffde1440000</td>
<td>r-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>[vvar]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7ffde1440000</td>
<td>-7ffde1442000</td>
<td>r-xp</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>[vdsr]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ffffffff6000000-ffffff6010000</td>
<td>r-xp</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>[vsyscall]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evolving Unix: memory mappings are top-level allocations
Hourglass proposal

My hourglass:

■ ‘typed’ allocations
■ dynamic binding

What’s left out:

■ anything dispatch semantics
  ♦ a matter for individual language impls
■ anything about static reasoning
  ♦ instead: load-time assume/guarantee
Evolving language implementations towards federatedness

Instead of throwing impls away, can we *evolve* them?

To *evolutionarily* realise

- dynamic binding
- semantic metadata

… what does the underlying (Unix-like) OS give us?
Evolving language implementations towards federatedness

Instead of throwing impls away, can we evolve them?

To evolutionarily realise

- dynamic binding
- semantic metadata

... what does the underlying (Unix-like) OS give us?

// "forward" lookup, by name

```c
double(*p_ceil)(double) = dlsym(RTLD_DEFAULT, "ceil");
```
Evolving language implementations towards federatedness

Instead of throwing impls away, can we *evolve* them?

To *evolutionarily* realise

- dynamic binding
- semantic metadata

... what does the underlying (Unix-like) OS give us?

```c
// "forward" lookup, by name
double(*p_ceil)(double)
    = dlsym(RTLD_DEFAULT, "ceil");

// "reverse" lookup, by address
Dl_info i;
dladdr(p_ceil, &i);
printf("%s\n", i.dli_sname); // "ceil"
```
Unix-style dynamism so far

Works only for ‘static’ objects (known to loader)

- not stack, heap, etc..

No *semantic* metadata

- e.g. type information

- ... including for structured data

- ... and for functions! (e.g. \texttt{ceil()})
A meta-level protocol...

```c
struct uniqtype;          /* type descriptor */
struct allocator;         /* heap, stack, static, etc */
allocator * alloc_get_allocator (void *obj); /* which one? (at leaf) */
uniqtype * alloc_get_type   (void *obj);       /* what type? */
void * alloc_get_site      (void *obj);         /* where allocated? */
void * alloc_get_base      (void *obj);         /* base address? */
void * alloc_get_limit     (void *obj);         /* end address? */
Dl_info alloc_dladdr       (void *obj);         /* dladdr-like */
```

// more calls go here...

This fills exactly the right gaps!

- works for any allocator: stack, heap, ...
- type information ≈ language-agnostic metamodel
- captures substructure and reference
How to get it

$ allocscce -o myprog ...
$ LD_PRELOAD=liballocs.so ./myprog
Type information and allocations

address space

- cat text
  - cat data
    - functions
    - variables
    - big chunk 1
      - suballocated chunks
  - malloc chunks including
- heap arena 1
- locales file
  - functions
  - file entities
- libc text
  - libc data
    - functions
    - variables
  - locales file
- ld.so text
  - ld.so data
    - functions
    - variables
- stack
  - stack frames
- kernel
  - kernel vvar
  - kernel vsyscall
  - kernel entry points
  - syscall trampolines

Type information is associated with leaf allocations
Reifying data types at run time

```c
struct ellipse {
    double maj, min;
    struct {
        double x, y;
    } ctr;
};
```

- Use the linker to keep them unique
- → “exact type” test is a pointer comparison
- _is_a() is a short search
Disjoint metadata example: malloc heap index

index by high-order bits of virtual address

entries are one byte, each covering 512B of heap

interior pointer lookups may require backward search

pointers encoded compactly as local offsets (6 bits)

instrumentation adds a trailer to each heap chunk

entries are one byte, each covering 512B of heap

interior pointer lookups may require backward search

pointers encoded compactly as local offsets (6 bits)

instrumentation adds a trailer to each heap chunk
Helping liballocs grok native code

LIBALLOCS_ALLOC_FNS="xcalloc(zZ)p xmalloc(Z)p xrealloc(pZ)p"
LIBALLOCS_SUBALLOCS_FNS="ggc Alloc(Z)p ggc Alloc_cleared(Z)p"
export LIBALLOCS_ALLOC_FNS
export LIBALLOCS_SUBALLOCS_FNS
allocscc -o myprog ... # call host compiler, postprocess metadata
One place type info can come from

$ cc -g -o hello hello.c && readelf -wi hello | column

<table>
<thead>
<tr>
<th>&lt;b&gt;:TAG_compile_unit</th>
<th>&lt;7ae&gt;:TAG_pointer_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT_language : 1 (ANSI C)</td>
<td>AT_byte_size: 8</td>
</tr>
<tr>
<td>AT_name : hello.c</td>
<td>AT_type : &lt;0x2af&gt;</td>
</tr>
<tr>
<td>AT_low_pc : 0x4004f6</td>
<td>AT_high_pc : 0x400516</td>
</tr>
<tr>
<td>AT_name : main</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;c5&gt;: TAG_base_type</th>
<th>&lt;791&gt;: TAG_formal_parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT_byte_size : 4</td>
<td>AT_type : &lt;0xc5&gt;</td>
</tr>
<tr>
<td>AT_encoding : 5 (signed)</td>
<td>AT_low_pc : 0x4004f6</td>
</tr>
<tr>
<td>AT_high_pc : 0x400516</td>
<td>AT_high_pc : 0x400516</td>
</tr>
<tr>
<td>AT_name : int</td>
<td>AT_name : argc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;2af&gt;:TAG_pointer_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT_byte_size: 8</td>
</tr>
<tr>
<td>AT_type : &lt;0x2b5&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;2b5&gt;:TAG_base_type</th>
<th>&lt;79f&gt;: TAG_formal_parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT_byte_size: 1</td>
<td></td>
</tr>
<tr>
<td>AT_encoding : 6 (char)</td>
<td></td>
</tr>
<tr>
<td>AT_name : char</td>
<td>AT_location : fbreg - 32</td>
</tr>
</tbody>
</table>
### liballocs vs C-language SPEC CPU2006 benchmarks

<table>
<thead>
<tr>
<th>bench</th>
<th>normal/s</th>
<th>liballocs/s</th>
<th>liballocs %</th>
<th>no-load</th>
</tr>
</thead>
<tbody>
<tr>
<td>bzip2</td>
<td>4.91</td>
<td>5.05</td>
<td>+2.9%</td>
<td>+1.6%</td>
</tr>
<tr>
<td>gcc</td>
<td>0.985</td>
<td>1.85</td>
<td>+88%</td>
<td>−%</td>
</tr>
<tr>
<td>gobmk</td>
<td>14.2</td>
<td>14.6</td>
<td>+2.8%</td>
<td>+0.7%</td>
</tr>
<tr>
<td>h264ref</td>
<td>10.1</td>
<td>10.6</td>
<td>+5.0%</td>
<td>+5.0%</td>
</tr>
<tr>
<td>hmmer</td>
<td>2.09</td>
<td>2.27</td>
<td>+8.6%</td>
<td>+6.7%</td>
</tr>
<tr>
<td>lbm</td>
<td>2.10</td>
<td>2.12</td>
<td>+0.9%</td>
<td>(−0.5%)</td>
</tr>
<tr>
<td>mcf</td>
<td>2.36</td>
<td>2.35</td>
<td>(−0.4%)</td>
<td>(−1.7%)</td>
</tr>
<tr>
<td>milc</td>
<td>8.54</td>
<td>8.29</td>
<td>(−3.0%)</td>
<td>+0.4%</td>
</tr>
<tr>
<td>perlbench</td>
<td>3.57</td>
<td>4.39</td>
<td>+23%</td>
<td>+1.6%</td>
</tr>
<tr>
<td>sjeng</td>
<td>3.22</td>
<td>3.24</td>
<td>+0.6%</td>
<td>(−0.7%)</td>
</tr>
<tr>
<td>sphinx3</td>
<td>1.54</td>
<td>1.66</td>
<td>+7.7%</td>
<td>(−1.3%)</td>
</tr>
</tbody>
</table>
Doing without FFIs: a proof-of-proof-of-concept-concept

$ ./node  # ← a popular JavaScript implementation
Doing without FFIs: a proof-of-proof-of-concept-concept

$ ./node  # <--- ... with liballocs extensions
> process.lm.printf("Hello, world!
")
Hello, world!
14
$ ./node # with liballocs extensions
> process.lm.printf("Hello, world!\n")
Hello, world!
14
> require(’-lxlt ’);
Doing without FFIs: a proof-of-proof-of-concept-concept

$ ./ node # with liballocs extensions
> process.lm.printf(”Hello, world!
”)
Hello, world!
14
> require(’−lXt ’)
> var toplvl = process.lm. XtInitialize ( 
    process.argv[0], ”simple”, null, 0,
    [process.argv.length], process.argv);
var cmd = process.lm.XtCreateManagedWidget( 
    ”exit ”, commandWidgetClass, toplvl, null, 0);
process.lm.XtAddCallback( 
    cmd, XtNcallback, process.lm.exit, null );
process.lm.XtRealizeWidget(toplvl);
process.lm.XtMainLoop();

Not “JS ↔ C”! One object space, many per-language views
Precise debugging

(gdb) print obj
$1 = (const void *) 0x6b4880 # unknown type!
Precise debugging

(gdb) print obj
$1 = (const void *) 0x6b4880 # unknown type!
(gdb) print __liballocs_get_alloc_type (obj)
$2 = (struct uniqtype *) 0x2b3aac997630
   <__uniqtype__InputParameters>
Precise debugging

(gdb) print obj
$1 = (void *) 0x6b4880
(gdb) print __liballocs_get_alloc_type (obj)
$2 = (struct uniqtype *) 0x2b3aac997630
   <__uniqtype__InputParameters>
(gdb) print *(struct InputParameters *) $2
$3 = {ProfileIDC = 0, LevelIDC = 0, no_frames = 0,  
     ...  }

Better debugger integration to follow...
How to retrofit an existing language VM

Basic retrofitting
- import: dispatch ‘external’ via metaprotocol
- externalise: expose complex data
- GC is via proxying

Full retrofitting
- reveal: implement the allocator protocol yourself
- … and export code/data back to the federation
- generalise: add a slow path, using *affinity*
- GC is federated
Dispatch external
Allocater affinity / retrofitting example

cmp [ebx,<class offset>],<cached class>; test class
jne <cached cache miss> ; miss? bail
mov eax,[ebx, <cached x offset>] ; hit
Allocator affinity / retrofitting example

xor ebx,<allocator mask> ; get allocator
cmp ebx,<cached allocator prefix> ; test
jne <allocator miss> ; miss? bail
cmp [ebx,<class offset>],<cached class>; test class
jne <cached cache miss> ; miss? bail
mov eax,[ebx, <cached x offset>] ; hit
Allocator affinity / retrofitting example

xor ebx,<allocator mask> ; get allocator
cmp ebx,<cached allocator prefix> ; test
jne <allocator miss> ; miss? bail

cmp [ebx,<class offset>],<cached class>; test class
jne <cached cache miss> ; miss? bail
mov eax,[ebx, <cached x offset>] ; hit

Ask me for more where this came from…
At load time, we know where the pointers are
- even in unsafe/native code!

To maintain at run time
- compute stack maps from debug info
- track ‘escape as integer’
What comes out of `dumptypes`

```c
/* uniqtype for stack frame main_0x4004f6_0x400516 */
struct uniqtype __uniqtype_main_0x4004f6_0x400516 = {
  32 /* size */,
  { composite: { COMPOSITE, /* nmemb */ 2 } },
  /* members */ {
    { &__uniqtype______PTR____PTR_signed_char$8,
      /* offset */ 0 } /* argv (size 8) */ ,
    { &__uniqtype__int$32,
      /* offset */ 12 } /* argc (size 4) (HOLE of 4B) */
  }
}

... like a struct, but describing the stack frame.

Is that hole harmless?
```
Actually not so easy...

Instead of

\[ \text{fbreg} - 20 \]

Compiler will sometimes say

\[ \text{rbx} - 12 \]

... if it knows that \( \text{rbx} == \text{frame base} + 8 \)

Solution:

- also consume DWARF frame information
- ... to recover fixed-offset relationships between regs
- solve as a graph-search problem
Some sources of holes in stack maps

- alignment (but...)
- key ABI details: saved IP, SP
- more ABI: use of callee-saved registers

We can fill these gaps in!
A better tool, filling in some gaps: dumpptrs

/* for stack frame main_vaddrs_0x4004f6_0x400516 */

struct stored_ptrs __ptrs_hello_c_main_vaddrs_0x4004f6_0x400516
0x4004f6, 0x400516, 2 /* nstored */, /* stored */ { 

    { .what = LOCAL, 
      .what_info = { local: { "argv" } }, 
      .where = STACK,  
      .where_info = { stack: { -32 } } } 

    , { 
      .what = CALLER_REG, 
      .what_info = { caller_reg: { 6 /* register rbp */ } } 
      .where = STACK, 
      .where_info = { stack: { -16 } } }

} } };

Work in progress: plugging more gaps in this tool
Some harder sources of holes in stack maps

- temporaries introduced by optimisation
- compiler whim?

i.e. things are easy at $-O0$, then get harder...
Compilers: a pragmatic attitude

Ideal-world fix: compilers describe their work perfectly!
- some interest, e.g. in LLVM, on debuginfo quality
- still not focused on high-assurance

More practical: binary analysis
- to *check* compiler description is correct
- to fill in details not captured (e.g. initializedness)
- ... Linux’s *objtool*
A two-pronged approach to compiler metadata

Yes, improve compilers; but also
- build tools to *identify* & *quantify* coverage gaps
- to work around unfixed gaps, dial down optimisation

Joint work ongoing with Francesco Zappa Nardelli
- objtool-like checking against binary instructions
- checking frame info only, initially

More to do!
- feedback loop for selective dialing down
More subtleties
1. saved registers – are they pointers or not?
2. initializedness – need a def/use analysis
3. duplicated values
4. conflated (superimposed) values
5. pointer-derived integers – are they pointers?
Subtlety 1: is it a pointer or is it an int?

/* for stack frame main_vaddrs_0x4004f6_0x400516 */

struct stored_ptrs __ptrs_hello_c_main_vaddrs_0x4004f6_0x400516

0x4004f6, 0x400516, 2 /* nstored */, /* stored */ {

    { .what = LOCAL,
        .what_info = { local: { "argv" } },
        .where = STACK,
        .where_info = { stack: { -32 } }
    },

    { .what = CALLER_REG,
        .what_info = { caller_reg: { 6 /* register rbp */ } }
        .where = STACK,
        .where_info = { stack: { -16 } }
    }
} };

‘Caller’s rbp’ could be a pointer, or not! But it’s okay…
Subtlety 2: initializedness

Just because it’s a pointer doesn’t mean it’s initialized

- def/use analysis needed in objtool-like tool
- pointers may be defined (have been written)
- pointers may be live (will be read)
- live but not defined → bad user code (check!)
- defined but not live → debuggers might care…

No tool yet, but on my list.
Subtlety 3: what about duplicates?

Suppose I copy from a stack slot to a register

…then break execution

Both copies might be live!

Debug info will only give us one

This also breaks the debugger’s `set var expr`
Subtleties 4 and 5: overlap and pointer-derived ints

/* uniqtype for stack frame 0x209a6b_0x556040_0x5562dc */

struct uniqtype __uniqtype___explow_cil_c_vaddrs_0x556040_0x5562dc

72 /* size */,

{ composite: { COMPOSITE, 4 } }, /* related */ {

  { memb: { &__uniqtype____PTR_rtx_def, 0 } } } /* (anonymous) */

  { memb: { &__uniqtype__uint$64, 8 } } } /* (anonymous) @209a

  { memb: { &__uniqtype__int$64, 16 } } } /* (anonymous) @209a

  { memb: { &__uniqtype____PTR_rtx_def, 16 } } } /* (anonymous

Hypothesis: this is less bad than it seems.

- + ask me about static analysis of pointer-derived ints

p.50
Other technology and research

Besides FFIs, there are

- multi-language VMs
- ‘marshal everything’
‘one VM to rule them all’
Current technology: ‘one VM to rule them all’

Many languages, one (core) implementation (VM)

- Microsoft .NET/CLR; Truffle/Graal; …
- C / C++ unsupported or hobbled
- doesn’t scale
'Marshal everything'

CORBA, COM, Protobufs . . .
- substantial run-time overheads
- need universal buy-in to an IDL
- (... or marshal by hand!)
- i.e. not doing language-level interop anyway
- and doesn’t scale

CoLoRS
- don’t marshal to wire; share memory
- no IDL; write runtime-specific glue
- a step in the right direction...
Why it’s not good enough

How to get an *entirely new* language up to speed:

- convince a large enough body of people
- . . . to make a large & risky investment
- . . . by spending huge amounts of money
- build all-new tools (esp. debuggers)
- FFI-wrap hundreds of must-have libraries
- (optionally: invent a new build system for no reason)

Fine if your name ∈ {Microsoft, Apple, Google, Mozilla}

To federate is to democratise . . .
Not-quite-federation

C implementations are federated
C++ implementations are federated
In practice, most other procedural languages (Fortran, Ada, Modula-2, . . . ) can join the federation via *full-multification*
C and assembly are mutually federated
C++ and C are not federated
Not to be confused with *annexation* e.g. Java/Scala, C/C++
Almost federation

- calling convention
- primitive data types’ representations
- structure packing

```c
extern "Pascal" {
  ...
}
```
Join me

http://github.com/stephenrkell/
feedback? wishlists?
“If we succeed in making an Intergalactic Network, then our main problem will be learning to communicate with Aliens.”

J.C.R. Licklider
Image credits

Squeak screenshot: Wolfgang Kreutzer

Phil Greenspun (minus Alex the Dog): Elsa Dorfman

Hot air balloons: AngMoKio

Flying cathedral: Friedrich Böhringer

Balloon over Brisbane: Cyron Ray Macey

Planets, rocket: NASA

copy of Escher’s “Hand in reflecting sphere”: Caia.or.kr

Vader reads the Bible: sirustalcelion

J.C.R. Licklider: US government / public domain
Conclusions

Unix is more dynamic than people think

Languages should be views, not worlds

Let’s build that infrastructure.

Thanks for your attention!