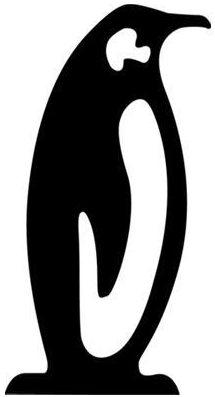


東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

bpftrace

BEGIN, builtins, and beyond...

Adin Scannell
Jordan Rome

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Who are we?

- Two maintainers of bpftrace, working at Meta
- Meta uses BPF extensively, including bpftrace
- Many different internal users & stakeholders
  - **Service owners:** on-demand and continuous system profiling (standardized scripts)
  - **Production engineers:** ad-hoc use for production debugging
  - **Kernel engineers:** iteration & debugging during development
- Meta runs bpftrace at **~HEAD**... *and so should you!*

Adin Scannell
amscanne

Jordan Rome
jordalgo

What is bpftrace, exactly?

- A tool & language to create and run Linux eBPF programs
- Allows fast, live tracing of the Linux kernel and userspace
- Originally inspired by awk, C, and DTrace

```
software:faults:1 {
  @[comm] = count();
}
```

Simple program to count page faults by thread name

General system observability

```
rawtracepoint:sys_enter
{
  @start[tid] = nsecs;
  @sc[tid] = args.id;
}

rawtracepoint:sys_exit
/@start[tid]/
{
  $lat_us = (nsecs - @start[tid]) / 1000;
  $name = syscall_name(@sc[tid]);
  @latency_us[$name] = lhist($lat_us, 0, 100, 10);
  _ = delete(@start, tid);
  _ = delete(@sc, tid);
}

end
{
  clear(@start);
  clear(@sc);
}
```

```
@latency_us[sendmsg]:
[0, 10)              729 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                   |
[10, 20)            1130 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[20, 30)             580 |@@@@@@@@@@@@@@@@@@@@@@@@@@                          |
[30, 40)             176 |@@@@@@@@                                            |
[40, 50)              44 |@@                                                  |
[50, 60)              24 |@                                                   |
[60, 70)               6 |                                                    |
[70, 80)               3 |                                                    |
[80, 90)               0 |                                                    |
[90, 100)              2 |                                                    |
[100, ...)             3 |                                                    |

@latency_us[epoll_wait]:
[0, 10)             9838 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@           |
[10, 20)           12468 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[20, 30)            3834 |@@@@@@@@@@@@@@@                                     |
[30, 40)            1378 |@@@@@                                               |
[40, 50)             767 |@@@                                                 |
[50, 60)             253 |@                                                   |
[60, 70)             227 |                                                    |
[70, 80)             208 |                                                    |
[80, 90)             364 |@                                                   |
[90, 100)            251 |@                                                   |
[100, ...)          6071 |@@@@@@@@@@@@@@@@@@@@@@@@@                           |
```

Performance deep dives

```
rawtracepoint:contention_begin
{
  @start[tid, args.lock_addr] = nsecs;
  @lock_type[args.lock_addr] = args.flags;
}

rawtracepoint:contention_end
/@start[tid, args.lock_addr]/
{
  $lat_us = (nsecs - @start[tid, args.lock_addr]) / 1000;
  @contention_us = hist($lat_us);
  @by_lock[args.lock_addr] = stats($lat_us);
  @wait_stacks[kstack(5)] = sum($lat_us);
  delete(@start, (tid, args.lock_addr));
}

end
{
  clear(@start);
  clear(@lock_type);
}
```

```
@by_lock[0xffff88888417ede0]: { .count = 1, .average = 2, .total = 2 }
@by_lock[0xffff888df1f9c428]: { .count = 1, .average = 2, .total = 2 }
@by_lock[0xffff888347965f20]: { .count = 2, .average = 2, .total = 5 }
@by_lock[0xffffffff83207080]: { .count = 1, .average = 3, .total = 3 }

@contention_us:
[0]                    1 |@@@@@@@@@@@@@@@@@
[1]                    0 |
[2, 4)                 3 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
[4, 8)                 1 |@@@@@@@@@@@@@@@@@

@wait_stacks[
        bpf_prog_1bc0c71b688765d5_rawtracepoint_vmlinux_contention_end_2+1
        bpf_prog_1bc0c71b688765d5_rawtracepoint_vmlinux_contention_end_2+1
        bpf_trace_run2+98
        queued_write_lock_slowpath+305
        __x64_sys_epoll_wait+1629
]: 5
```

System debugging

```
rawtracepoint:sched_switch
{
    $css_set = args.next.cgroups;

    if ($css_set != 0 &&
        $css_set.dfl_cgrp != 0) {
        $cgrp = $css_set.dfl_cgrp;
        $level = $cgrp.level;
        $cgrp_id = $cgrp.kn.id;
        printf("%-16s [pid=%5d] cgroup=%s\n",
               comm, pid, cgroup_path(cgroup));
    }
}
```

```
swapper/96       [pid=    0] cgroup=unified:/
kworker/u664:13  [pid=3524231] cgroup=unified:/
swapper/31       [pid=    0] cgroup=unified:/
kworker/u664:16  [pid=3703441] cgroup=unified:/
kworker/u664:4   [pid=3684615] cgroup=unified:/
swapper/93       [pid=    0] cgroup=unified:/
swapper/96       [pid=    0] cgroup=unified:/
kworker/u664:4   [pid=3684615] cgroup=unified:/
kworker/u664:13  [pid=3524231] cgroup=unified:/
kworker/u664:16  [pid=3703441] cgroup=unified:/
swapper/123      [pid=    0] cgroup=unified:/
swapper/96       [pid=    0] cgroup=unified:/
dnf              [pid=3727620] cgroup=unified:/system.slice/…
```

… and even tracing BPF with your BPF

```
config = { missing_probes = warn; }

// Attach to all running BPF programs.
fentry:bpf:* {
    @bpf_invocations[probe] = count();
}

interval:s:1 {
    print(@bpf_invocations);
    clear(@bpf_invocations);
}
```

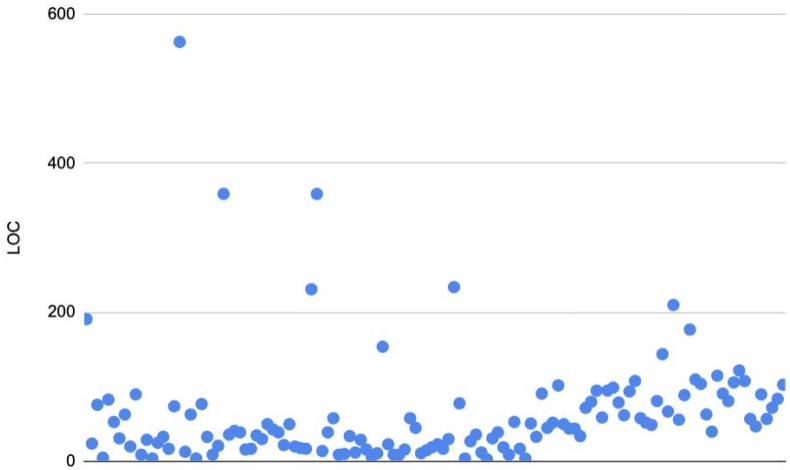
```
…
@bpf_invocations[fentry:bpf:374557:sslwall_sockops]: 238
@bpf_invocations[fentry:bpf:1061927:armr_arg_get_len]: 239
@bpf_invocations[fentry:bpf:1064300:bpfj_file_match_drop_iters]: 256
@bpf_invocations[fentry:bpf:1064301:bpfj_file_match_drop_iters]: 256
@bpf_invocations[fentry:bpf:1064304:bpfj_file_match_drop_iters]: 256
@bpf_invocations[fentry:bpf:965439:armr_match_env]: 348
@bpf_invocations[fentry:bpf:1064307:bpfj_file_match_drop_iters]: 448
@bpf_invocations[fentry:bpf:1064267:bpfj_file_match_clear_iters]: 448
@bpf_invocations[fentry:bpf:1061927:armr_match_env_key]: 470
@bpf_invocations[fentry:bpf:965439:file_iter]: 512
@bpf_invocations[fentry:bpf:965424:dnswatch_kprobe_tcp_sendmsg]: 614
@bpf_invocations[fentry:bpf:1064269:bpfj_file_match_drop_iters]: 640
@bpf_invocations[fentry:bpf:967143:armr_filemod_security_file_open]: 661
@bpf_invocations[fentry:bpf:786744:restrict_filesystems]: 709
@bpf_invocations[fentry:bpf:965427:dnswatch_kprobe_tcp_recvmsg]: 719
@bpf_invocations[fentry:bpf:1064299:bpfj_mount_load]: 931
@bpf_invocations[fentry:bpf:1064299:bpfj_get_pods_from_pid]: 931
@bpf_invocations[fentry:bpf:1064299:check_path_cached]: 931
@bpf_invocations[fentry:bpf:1064299:bpfj_fs_file_open]: 931
@bpf_invocations[fentry:bpf:1064299:check_path_restricted]: 931
@bpf_invocations[fentry:bpf:965439:do_free]: 1331
@bpf_invocations[fentry:bpf:965439:armr_proc_lookup]: 1337
@bpf_invocations[fentry:bpf:965439:armr_file_lineage_lsm_file_free]: 1349
@bpf_invocations[fentry:bpf:1064297:do_udp]: 1438
@bpf_invocations[fentry:bpf:1064297:enforce_security_socket_recvmsg]: 1438
@bpf_invocations[fentry:bpf:1064295:do_udp]: 1552
@bpf_invocations[fentry:bpf:1064295:enforce_security_socket_sendmsg]: 1552
@bpf_invocations[fentry:bpf:1064299:bpfj_mount_clear_loop]: 1916
@bpf_invocations[fentry:bpf:1064299:bpfj_mount_traverse_loop]: 2031
@bpf_invocations[fentry:bpf:1064267:bpfj_file_match_drop_iters]: 2176
@bpf_invocations[fentry:bpf:1061465:do_free]: 2494
@bpf_invocations[fentry:bpf:1061465:armr_connect_lsm_file_free]: 2503
@bpf_invocations[fentry:bpf:1064299:bpfj_file_match_check_iter]: 2738
@bpf_invocations[fentry:bpf:1064299:bpfj_file_match_process_indexes]: 2738
@bpf_invocations[fentry:bpf:1064299:bpfj_file_match_reset]: 3214
@bpf_invocations[fentry:bpf:1064299:bpfj_file_match_walk_path]: 3214
@bpf_invocations[fentry:bpf:605997:validateChainedExtReturnFn]: 4380
@bpf_invocations[fentry:bpf:1064299:bpfj_file_create_oldest_kpath]: 4826
@bpf_invocations[fentry:bpf:679541:tracepoint__sched__sched_switch]: 12871
@bpf_invocations[fentry:bpf:1061927:armr_match_env]: 18564
@bpf_invocations[fentry:bpf:1064299:bpfj_file_match_clear_iters]: 60160
@bpf_invocations[fentry:bpf:1064299:bpfj_file_match_drop_iters]: 175232
```

The point is, it can do a lot

- Many probe types: kprobe, fentry, tracepoints, rawtracepoints, hardware & software perf events, watchpoints, uprobes, USDTs, etc.
- Can traverse complex structures with types inferred automatically from BTF
- Lots of convenience features: variables, loops (bpf_loop), map aggregations, asynchronous formatting & printing, etc.
- **Even AI can generate useful scripts** (*sometimes*)

Yet, big scripts & community tools are rare

- Problems often “grow out” of bpftrace
  - We are “experts” and our largest script is ~200 LOC
  - Largest external script is an implementation of Tetris
- Only ~10k LOC in unique scripts on GitHub!
  - *What gives!?*

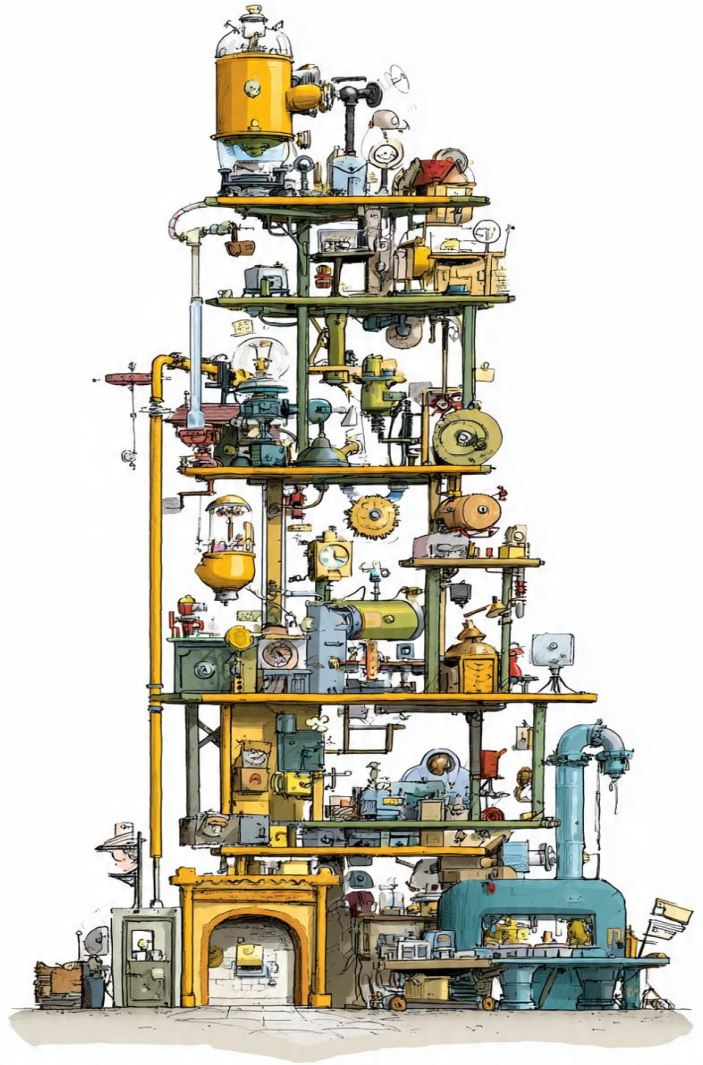


It's easy to hit a wall and be disappointed

- "Can I factor out the common code in these probes?"
- "Can I do an conditional insert into a map?"
- "Can I use an LRU map?"
- "How do I access a different field here for ARM64?"
- "Can I call this kfunc? It has exactly what I need!"
- "How can I load this per_cpu pointer in a struct?"

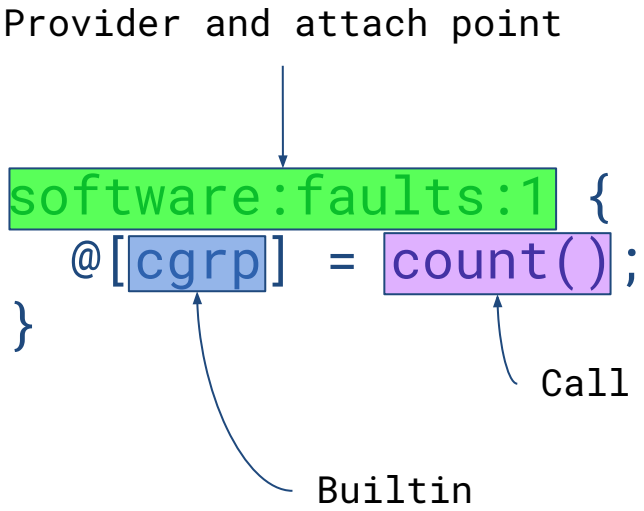
bpftrace has a long history!

- It was created before many other technologies in the space (BTF, CO-RE, etc.) were standard
- It has had different active maintainers
- Over time, many features have been added to support narrowly scoped use-cases
  - (These features aren't always finished...)
- There are no **escape hatches** (BYO C functions)



Example: adding a new “builtin”

- Idea: use the **cgroup_id** instead of **comm**
- We can use `bpf_get_current_cgroup_id`
- This can’t be hard, right?



Everything is a language feature (1)

- Every call is a special builtin, and lexed differently
- Adding a function means updating the parser, tests, etc.

```
485  primary_expr:
486                  IDENT             { $$ = driver.ctx.make_node<ast::Identifier>($1, @$); }
487          |       int               { $$ = $1; }
488          |       STRING            { $$ = driver.ctx.make_node<ast::String>($1, @$); }
489          |       STACK_MODE        { $$ = driver.ctx.make_node<ast::StackMode>($1, @$); }
490          |       BUILTIN           { $$ = driver.ctx.make_node<ast::Builtin>($1, @$); }
491          |       CALL_BUILTIN      { $$ = driver.ctx.make_node<ast::Builtin>($1, @$); }
492          |       LPAREN expr RPAREN { $$ = $2; }
493          |       param             { $$ = $1; }
494          |       map_or_var        { $$ = $1; }
```

```
33   ident     [_a-zA-Z][_a-zA-Z0-9]*
34   map       @{ident}|@
35   var       ${ident}
36   hspace    [ \t]
37   vspace    [\n\r]
38   space     {hspace}|{vspace}
39   path      :(\\.|[_\-\./a-zA-Z0-9#+\*])+
40   builtin   arg[0-9]|args|cgroup|comm|cpid|numaid|cpu|ctx|curtask|elapsed|func|gid|pid|probe|rand|retval|sarg[0-9]|tid|uid|username|jiffies
41   call      avg|buf|cat|cgroupid|clear|count|delete|exit|hist|join|kaddr|kptr|ksym|len|lhist|macaddr|max|min|ntop|override|print|printf|cgr
42
43   int_type        bool|(u)?int(8|16|32|64)
44   builtin_type    void|(u)?(min|max|sum|count|avg|stats)_t|probe_t|username|lhist_t|hist_t|usym_t|ksym_t|timestamp|macaddr_t|cgroup_path_t
45   sized_type      string|inet|buffer
46   subprog         fn
```

Types are bespoke (2)

- Core types are big list, each with custom behavior, casting rules, etc.
- Types need special handling in many places:
  - Adding types to the field analyser, for structures
  - Adding types to the semantic analyser, for checking
  - Adding types to codegen

```
65  void FieldAnalyser::visit(Builtin &builtin)
66  {
67    std::string builtin_type;
68    sized_type_ = CreateNone();
69    if (builtin.ident == "ctx") {
70      if (!probe_)
71        return;
72      switch (prog_type_) {
73        case BPF_PROG_TYPE_KPROBE:
74          builtin_type = "struct pt_regs";
75          break;
76        case BPF_PROG_TYPE_PERF_EVENT:
77          builtin_type = "struct bpf_perf_event_data";
78          break;
79        default:
80          break;
81      }
82      // For each iterator probe, the context is pointing to specific struct,
83      // make them resolved and available
84      if (probe_type_ == ProbeType::iter)
85        builtin_type = "struct bpf_iter__" + attach_func_;
86    } else if (builtin.ident == "__builtin_curtask") {
87      builtin_type = "struct task_struct";
```

```
23  enum class Type : uint8_t {
24      // clang-format off
25      none,
26      voidtype,
27      integer, // int is a protected keyword
28      boolean,
29      pointer,
30      record, // struct/union, as struct is a protected keyword
31      hist_t,
32      lhist_t,
33      tseries_t,
34      count_t,
35      sum_t,
36      min_t,
37      max_t,
38      avg_t,
39      stats_t,
40      kstack_t,
```

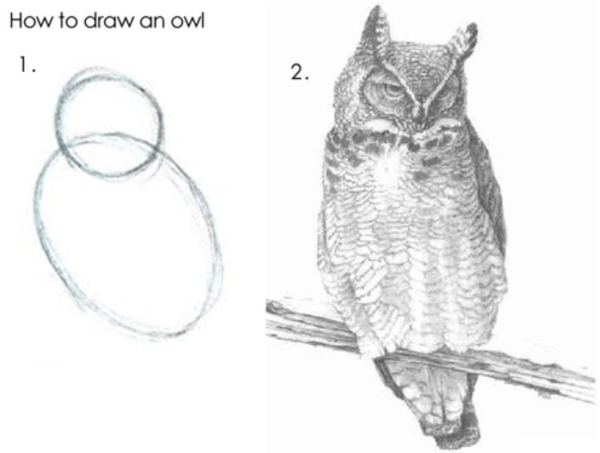
Codegen is hand-rolled (3)

- Manually inject declarations for our kfunc
- Hand-rolled IR for anything non-trivial

```
493  void CodegenLLVM::visit(Call &call)
494  {
495    if (call.func == "count") {
496      Map &map = *call.map;
497      auto [key, scoped_key_deleter] = getMapKey(map);
498      b_.CreateMapElemAdd(ctx_, map, key, b_.getInt64(1), call.loc);
499      expr_ = nullptr;
500    } else if (call.func == "sum") {
501      Map &map = *call.map;
502      auto [key, scoped_key_deleter] = getMapKey(map);
503      auto scoped_del = accept(call.vargs.front());
504      // promote int to 64-bit
505      expr_ = b_.CreateIntCast(expr_,
506                               b_.getInt64Ty(),
507                               call.vargs.front()->type.IsSigned());
508      b_.CreateMapElemAdd(ctx_, map, key, expr_, call.loc);
509      expr_ = nullptr;
510    } else if (call.func == "max" || call.func == "min") {
```

```
2302  void IRBuilderBPF::CreateMapElemAdd(Value *ctx,
2303                                      Map &map,
2304                                      Value *key,
2305                                      Value *val,
2306                                      const location &loc)
2307  {
2308    CallInst *call = CreateMapLookup(map, key);
2309    SizedType &type = map.type;
2310
2311    Function *parent = GetInsertBlock()->getParent();
2312    BasicBlock *lookup_success_block = BasicBlock::Create(module_.getContext(),
2313                                                          "lookup_success",
2314                                                          parent);
2315    BasicBlock *lookup_failure_block = BasicBlock::Create(module_.getContext(),
2316                                                          "lookup_failure",
2317                                                          parent);
2318    BasicBlock *lookup_merge_block = BasicBlock::Create(module_.getContext(),
2319                                                        "lookup_merge",
2320                                                        parent);
2321
2322    AllocaInst *value = CreateAllocaBPF(type, "lookup_elem_val");
2323    Value *condition = CreateICmpNE(CreateIntCast(call, GET_PTR_TY(), true),
2324                                    GetNull(),
2325                                    "map_lookup_cond");
2326    CreateCondBr(condition, lookup_success_block, lookup_failure_block);
2327
2328    SetInsertPoint(lookup_success_block);
2329
2330    // createMapLookup  returns an u8*
2331    auto *cast = CreatePointerCast(call, value->getType(), "cast");
2332    CreateStore(CreateAdd(CreateLoad(value->getAllocatedType(), cast), val),
2333                cast);
2334
2335    CreateBr(lookup_merge_block);
2336
2337    SetInsertPoint(lookup_failure_block);
2338
2339    CreateMapElemInit(ctx, map, key, val, loc);
2340
2341    CreateBr(lookup_merge_block);
2342    SetInsertPoint(lookup_merge_block);
2343    CreateLifetimeEnd(value);
2344    return;
2345  }
```

Draw the rest of the owl (4)



- Handle attach points and program type checks across the codebase
- Add field analyser tests, semantic analyser tests, runtime tests
- Add documentation, man page

```
src/ast/ast.cpp:    case ProbeType::software:
src/ast/ast.cpp:      case ProbeType::software:
src/ast/ast.cpp:      case ProbeType::software:
src/ast/passes/attachpoint_passes.cpp:    case ProbeType::software:
src/ast/passes/codegen_llvm.cpp:    case ProbeType::software:
src/ast/passes/pid_filter_pass.cpp:    case ProbeType::software:
src/ast/passes/semantic_analyser.cpp:    case ProbeType::software:
src/attached_probe.cpp:    case ProbeType::software:   return BPF_PROG_TYPE_PERF_EVENT; break;
src/attached_probe.cpp:    case ProbeType::software: {
src/bpftrace.cpp:    case ProbeType::software:
src/probe_matcher.cpp:    case ProbeType::software: {
src/probe_matcher.cpp:    case ProbeType::software:
src/probe_types.cpp:    case ProbeType::software:    return "software";    break;
src/probe_types.h:    .type = ProbeType::software,
```

1.0 is overdue; let's fix these problems!

- **Goals**
  - Simplified core language internals (robust & predictable)
  - Most functionality in a standard library + *user extensibility*
  - *A language contract that can be supported forever(*)!*
- Not about features; make what's there already more coherent
- Formalizing the lifts...
  - Add **language features** to remove fragile code in the language & runtime
  - Clean up **tech debt**; old workarounds are holding us back!
  - Provide **clear paths** for standard library development and contributions

The Journey is Well-Underway

So many features already added in trunk/master.
Let's look at them and how they connect to get us to 1.0

Factor out common code with Macros

```
macro add_one(x) {
  x + 1
}

BEGIN {
  print(add_one(1)); // prints 2
}
```

Factor out common code with Macros

```
macro strstr($haystack, $needle)
{
  assert_str($haystack);
  assert_str($needle);
  let $haystack_size = strcap($haystack);
  let $needle_size = strcap($needle);
  if ($needle_size == 0 || $needle[0] == 0) {
    0
  } else {
    __bpf_strnstr(&$haystack, &$needle, $haystack_size,
$needle_size)
  }
}
```

Macros: Three Flavors

```
macro add_one(x) {
  x + 1
}
```

Expressions

```
macro add_one($x) {
  $x += 1;
}
```

Variables

```
macro add_one(@x) {
  @x[1] += 1;
}
```

Maps

Imports: Other bpftrace scripts

main.bt

```
import "my_library.bt"

begin {
  print(add_one(1));
}
```

my_library.bt

```
macro add_one(x) {
  x + 1
}
```

Imports: C Files!

main.bt

```
import "my_c_lib.bpf.c"

begin {
  print(__add_one(1));
}
```

my_c_lib.bpf.c

```
int __add_one(int val) {
  return 1 + val;
}
```

C Interop

```
extern struct task_struct *bpf_task_from_pid(s32 pid) __weak __ksym;
extern void bpf_task_release(struct task_struct *p) __weak __ksym;

int __bpf_task_comm_from_pid(s32 pid, m_arg *out) {
  if (!bpf_task_from_pid || !bpf_task_release)
    return -95;

  struct task_struct *task = bpf_task_from_pid(pid);
  if (task) {
    __builtin_memcpy(&out->data, &task->comm, sizeof(*out));
    bpf_task_release(task);
    return 0;
  }
  return -3;
}
```

Why Should I Care?

- No more waiting for bpftrace to use cool new BPF features
- Easier to contribute to bpftrace, no more forced LLVM IR
- All the powers (and heartache) of C
- Faster development of core bpftrace features

Compile Time

- Constant Folding
- Branch Pruning
- Type Introspection

comptime

```
macro signal(expr) {
  if comptime (probetype != “kprobe”) {
   fail(“Signal can only be used in kprobes”);
  }
}
```

comptime

```
# bpftrace -e ‘interval:1s { signal(1); }’

if comptime (“interval” != “kprobe”) {
  fail(“Signal can only be used in kprobes”);
}

if comptime (true) {
  fail(“Signal can only be used in kprobes”);
}

  fail(“Signal can only be used in kprobes”);
```

Type Introspection: typeinfo

```
macro signal(expr) {
  let $sig = 0;
  if comptime (is_literal(expr)) {
    if comptime (is_str(expr)) {
      $sig = __builtin_signal_num(expr);
    } else if comptime (is_integer(expr)) {
       $sig = expr;
    } else {
      fail(“signal accepts a string or a positive int");
    }
  }
}
```

Type Introspection: typeinfo

```
macro is_str(expr) {
  typeinfo(expr).1 == "string"
}

macro is_integer(expr) {
  typeinfo(expr).1 == "int"
}

begin {
  print(typeinfo(1)); // (0, int, uint8)
}
```

Towards New Features

```
uprobe:uprobe_test:uprobeFunction1 {
  if comptime (arch == “ppc64”) {
    @ = usym(reg("nip");
  } else {
    @ = usym(reg("ip");
  }
}
```

Now this script works on x86 and ppc64!

The Standard Library

- Less core C++ code + Less LLVM IR
  - Easier to contribute to the project
  - Less code to manage
- More composable and powerful primitives/builtins
- Better user documentation
- Better developer documentation
- Decisions around what belongs in the stdlib...

bpftrace

Dynamic Tracing for Linux

bpftrace is a high-level tracing language for Linux and provides a quick and easy way for people to write observability-based eBPF programs, especially those unfamiliar with the complexities of eBPF.

Learn more about bpftrace, check out all the great tools built with bpftrace, and please contribute!

Version (0.24) Released!
Check out all the new features, changes, and fixes in the release notes

New Blog Post: The Path to 1.0
Read about all the new and exiciting features coming to bpftrace.

Playground

```
begin { print("Hello World!"); }
```

RUN ▶ CLEAR Execution finished successfully.

```
Attached 1 probe
Hello World!
```

Links

- https://bpftrace.org/
- https://github.com/bpftrace/bpftrace
- New Discord Chat!: https://discord.gg/CMZkV5Fg

Thank You!