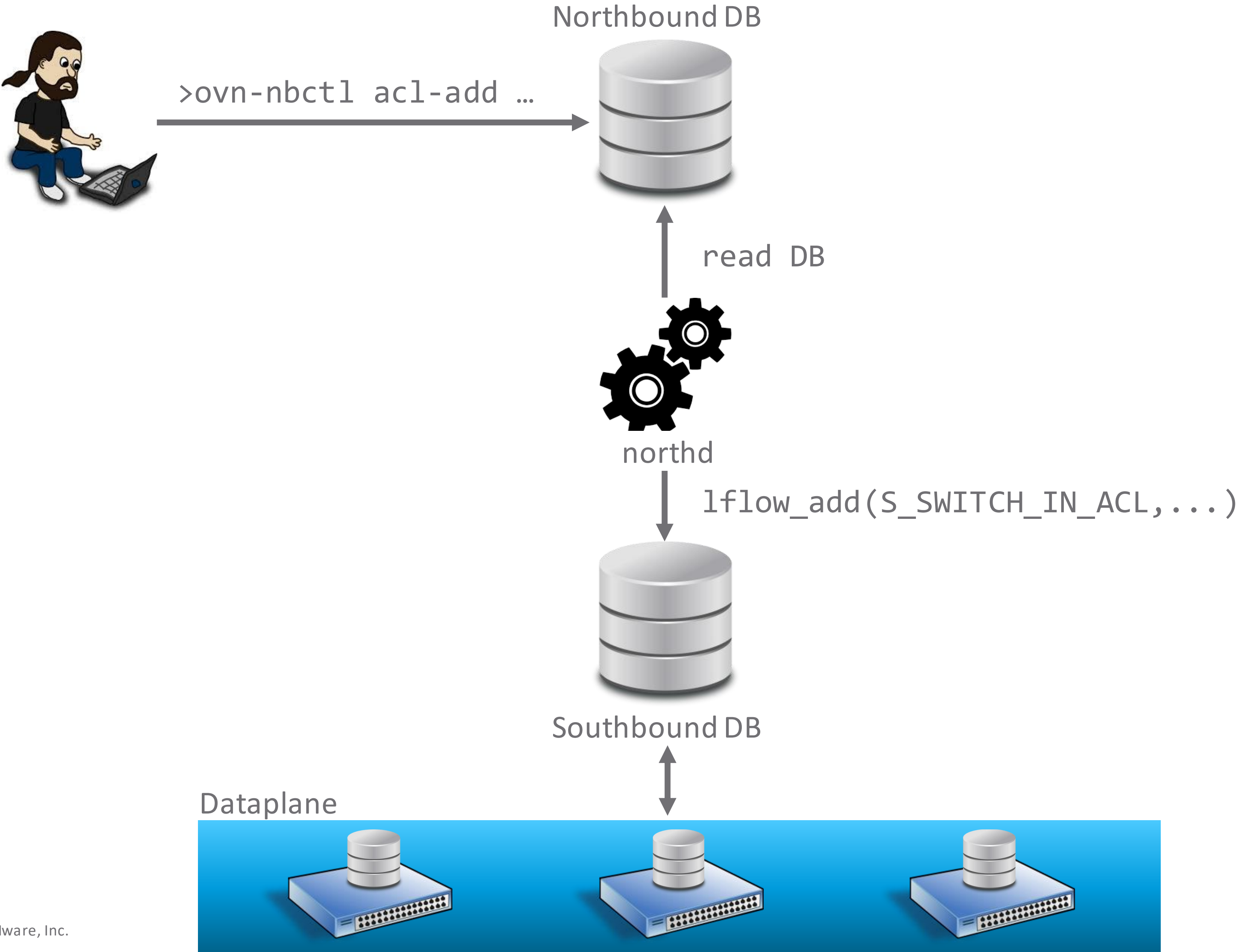
Towards Leaner, Faster ovn-northd

Mihai Budiu
Justin Pettit
Ben Pfaff
Leonid Ryzhyk

May 2018

vmware®
©2018 VMware, Inc.

Background: ovn-northd



vmware

©2018 VMware, Inc.

man ovn-northd

```
Ingress Table 10: ARP/ND responder

  This  table  implements  ARP/ND  responder in a logical switch for known IPs. The advantage of the ARP responder flow is to limit ARP
  broadcasts by locally responding to ARP requests without the need to send to other hypervisors. One common case is when the inport is
```

```
  ·      Priority-50 flows that match IPv6 ND neighbor solicitations to each known IP address A (and A's solicited node address)
         of every logical switch port, and respond with neighbor advertisements directly with corresponding Ethernet address E:

         nd_na {
             eth.src = E;
             ip6.src = A;
             nd.target = A;
             nd.tll = E;
             outport = inport;
             flags.loopback = 1;
             output;
         };

         These flows are omitted for logical ports (other than router ports or localport ports) that are down.
```

vmware®

©2018 VMware, Inc.

Example ovn-northd code

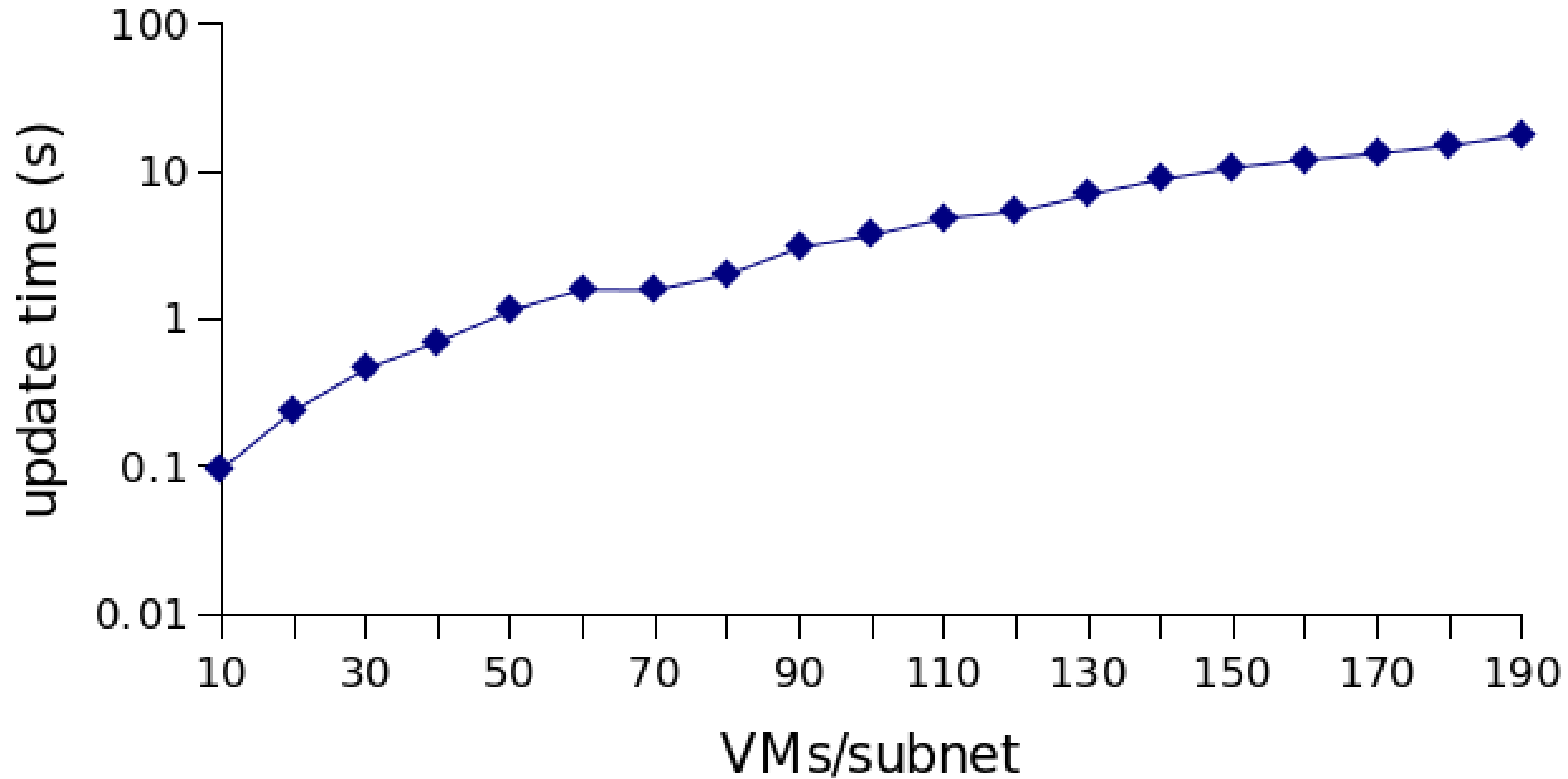
```
for (j = 0; j < op->lsp_addrs[i].n_ipv6_addrs;
     j++) {
  ds_clear(&match);
  ds_put_format(&match,
    "nd_ns && ip6.dst == {%s, %s}“
    "&& nd.target == %s",
    op->lsp_addrs[i].ipv6_addrs[j].addr_s,
    op->lsp_addrs[i].ipv6_addrs[j].sn_addr_s,
    op->lsp_addrs[i].ipv6_addrs[j].addr_s);
  ds_clear(&actions);
  ds_put_format(&actions,
    "nd_na {eth.src = %s; ip6.src = %s; "
      "nd.target = %s; nd.tll = %s; "
      "outport = inport; flags.loopback = 1;"
      "output; };",
    op->lsp_addrs[i].ea_s,
    op->lsp_addrs[i].ipv6_addrs[j].addr_s,
    op->lsp_addrs[i].ipv6_addrs[j].addr_s,
    op->lsp_addrs[i].ea_s);
  ovn_lflow_add(lflows, op->od,
    S_SWITCH_IN_ARP_ND_RSP,
    50, ds_cstr(&match), ds_cstr(&actions));
```

Problems with this code:

1. Lots of printf-like string manipulation
2. Flows recomputed from scratch on each NB DB update
   - Simplifies the implementation

©2018 VMware, Inc.

Scaling ovn-northd



Time to compute network update (adding a logical port)

©2018 VMware, Inc.

Example ovn-northd code

```
for (j = 0; j < op->lsp_addrs[i].n_ipv6_addrs;
     j++) {
  ds_clear(&match);
  ds_put_format(&match,
    "nd_ns && ip6.dst == {%s, %s}“
    "&& nd.target == %s",
    op->lsp_addrs[i].ipv6_addrs[j].addr_s,
    op->lsp_addrs[i].ipv6_addrs[j].sn_addr_s,
    op->lsp_addrs[i].ipv6_addrs[j].addr_s);
  ds_clear(&actions);
  ds_put_format(&actions,
    "nd_na {eth.src = %s; ip6.src = %s; "
      "nd.target = %s; nd.tll = %s; "
      "outport = inport; flags.loopback = 1;"
      "output; };",
    op->lsp_addrs[i].ea_s,
    op->lsp_addrs[i].ipv6_addrs[j].addr_s,
    op->lsp_addrs[i].ipv6_addrs[j].addr_s,
    op->lsp_addrs[i].ea_s);
  ovn_lflow_add(lflows, op->od,
    S_SWITCH_IN_ARP_ND_RSP,
    50, ds_cstr(&match), ds_cstr(&actions));
```

Problems with this code:

1. Lots of printf-like string manipulation
2. Flows recomputed from scratch on each NB DB update

Flow Template Language (FTL)

- Developed by Ben in 2016
- Based on FLWOR
- Addresses Problem 1.

vmware ©2018 VMware, Inc.

Same example in FTL

C

```
for (j = 0; j < op->lsp_addrs[i].n_ipv6_addrs;
     j++) {
  ds_clear(&match);
  ds_put_format(&match,
    "nd_ns && ip6.dst == {%s, %s}"
    "&& nd.target == %s",
    op->lsp_addrs[i].ipv6_addrs[j].addr_s,
    op->lsp_addrs[i].ipv6_addrs[j].sn_addr_s,
    op->lsp_addrs[i].ipv6_addrs[j].addr_s);
  ds_clear(&actions);
  ds_put_format(&actions,
    "nd_na {eth.src = %s; ip6.src = %s; "
      "nd.target = %s; nd.tll = %s; "
      "outport = inport; flags.loopback = 1;"
      "output; };",
    op->lsp_addrs[i].ea_s,
    op->lsp_addrs[i].ipv6_addrs[j].addr_s,
    op->lsp_addrs[i].ipv6_addrs[j].addr_s,
    op->lsp_addrs[i].ea_s);
  ovn_lflow_add(lflows, op->od,
    S_SWITCH_IN_ARP_ND_RSP,
    50, ds_cstr(&match), ds_cstr(&actions));
```

FTL

```
for (lspip in Logical_Switch_Port_IP
     if lspip.lsp.up) {
  let E = lspip.mac,
      A = lspip.ip,
      P = lspip.lsp.name,
      S = lspip.sn_ip in
    Flow(lspip.lsp.ls, LS_IN_ARP_RSP, 50,
         $[|nd_ns && ip6.dst == {${A}, ${S}} &&
          nd.target == ${A}|],
         $[|{
           nd_na {
             eth.src = ${E};
             ip6.src = ${A};
             nd.target = ${A};
             nd.tll = ${E};
             outport = inport;
             flags.loopback = 1;
             output;
           };|])
}
```

©2018 VMware, Inc.

Limitations of FTL

Limitations of FTL

- Still relies on complete recomputation of all logical flows (Problem 2)
- Cannot express aggregate queries

FTL

```
for (ls in Logical_Switch if ls.has_stateful_acl) {
  Flow(ls, LS_IN_ACL, 1, "ip && (!ct.est||(ct.est && ct_label.blocked))",
       "{reg0[1] = 1; next; }");
  ...
```

C

```
bool has_stateful_acl(struct ovn_datapath *od)
{
    for (size_t i = 0; i < od->nbs->n_acls; i++) {
        struct nbrec_acl *acl = od->nbs->acls[i];
        if (!strcmp(acl->action, "allow-related"))
            return true;
    }
    return false;
}
```

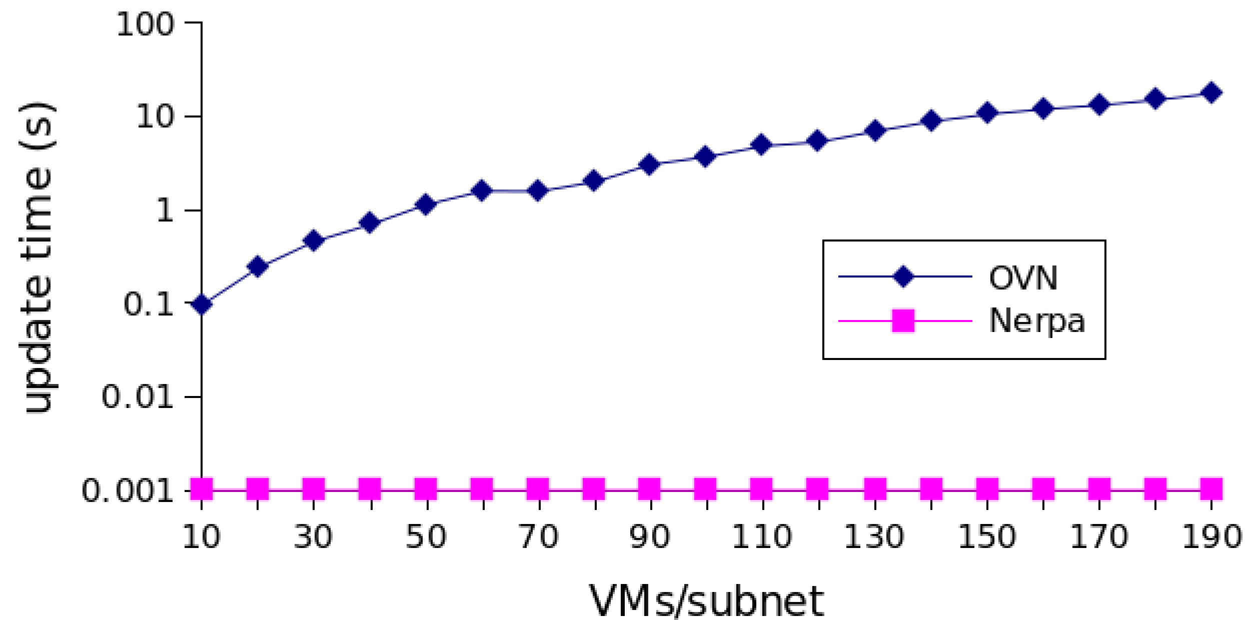
Not expressible in FTL, requires adding dynamically computed columns to tables.

vmware ©2018 VMware, Inc.

FTL++, aka Differential Datalog (DD)

FTL++ (aka Differential Datalog)

- A superset of FTL
  - Runs Ben's FTL code with trivial syntactic changes
- Supports complex queries, including aggregates and negation
- Fully incremental evaluation



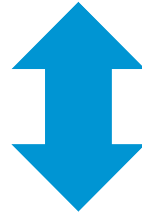
©2018 VMware, Inc.

DD

FTL is a declarative language in disguise (remember nlog?):

FTL

```
for (ls in Logical_Switch if ls.has_stateful_acl) {
  Flow(ls, LS_IN_ACL, 1, "ip && (!ct.est||(ct.est && ct_label.blocked))",
       "{reg0[1] = 1; next;}");
  ...
```



Datalog

```
Flow(.datapath = ls,
     .table    = LS_IN_ACL,
     .priority = 1,
     .match    = "ip && (!ct.est || (ct.est && ct_label.blocked))",
     .action   = "reg0[1] = 1; next;") :-
    LogicalSwitch(ls, …, has_stateful_acl), has_stateful_acl==true
```

Datalog admits incremental evaluation!

©2018 VMware, Inc.

DD

- DD is syntactic sugar on top of Datalog
- The user can also use Datalog directly to define intermediate tables (**views**)

```
relation Logical_Switch_has_stateful_acl(ls: uuid)

Logical_Switch_has_stateful_acl_action(ls) :-
    ACL(.logical_switch=ls, .action="allow-related").
```

```
for (ls in Logical_Switch)
  for (stateful in Logical_Switch_has_stateful_acl_action if stateful.ls == ls){
    Flow(ls, LS_IN_ACL, 1, "ip && (!ct.est || (ct.est && ct_label.blocked))",
         "{reg0[1] = 1; next;}");
  ...
```

vmware® ©2018 VMware, Inc.

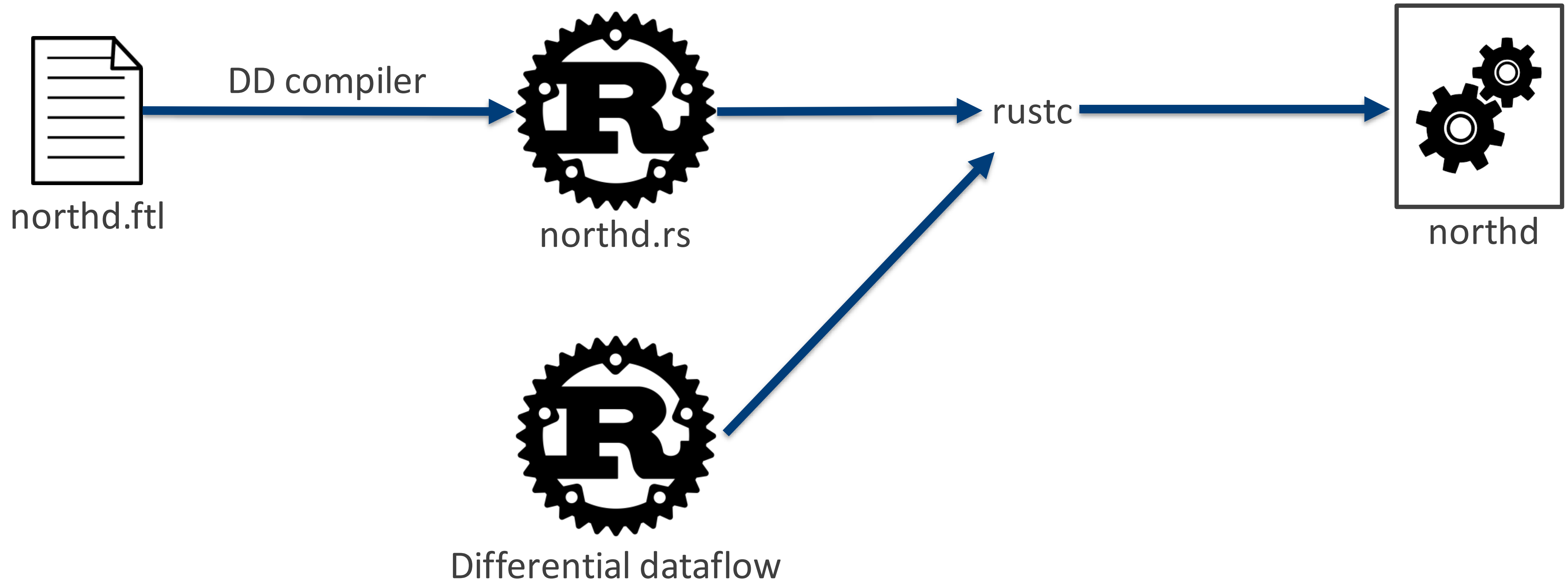
DD

- DD is a standalone tool (not part of OVN)
- We are building DD on top of Differential Dataflow—a high-performance incremental dataflow framework.

vmware®

©2018 VMware, Inc.

Compiling DD



vmware®
©2018 VMware, Inc.

Timeline

Step 1: DD compiler

- ~August 2018 (3 months from now)

Step 2: OVN

- ~December 2018
- **Need help from OVN developers**
- Possibly, adapt Ben's implementation of northd in FTL (developed in 2016, likely bit rotten)

Step 3 (future releases):

- Better abstractions for logical datapath programming
- Abstract away some of the OpenFlow complexity

©2018 VMware, Inc.

Conclusion

Why does this matter to me? (I am happy with OVN's performance)

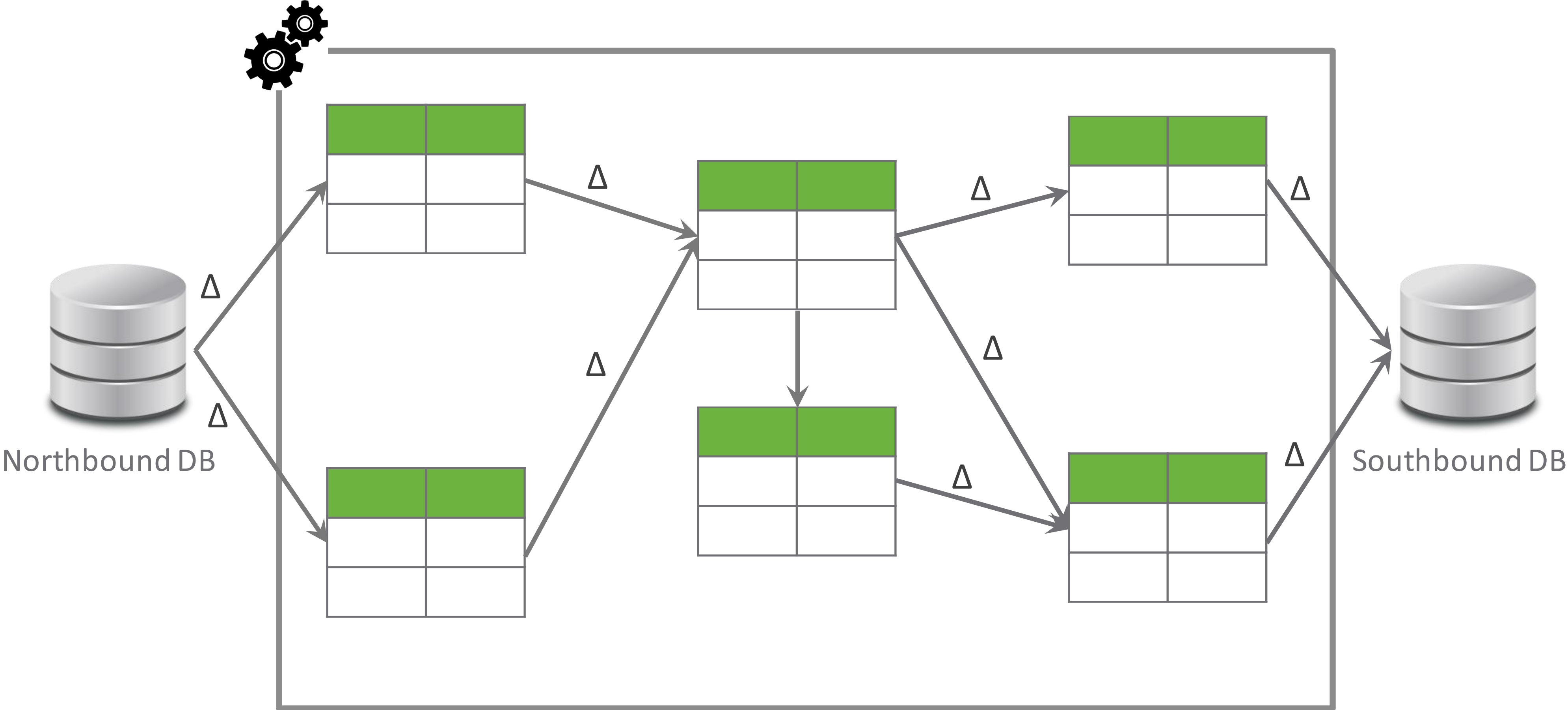
- OVN is an open SDN platform
  - OVN = OVS + OVSDB + logical flow abstraction + app logic
- DD unlocks OVN's programmability
- What do **YOU** want to build with OVN?

https://github.com/ryzhyk/differential-datalog

vmware®

©2018 VMware, Inc.

Differential Dataflow Runtime



©2018 VMware, Inc.