“So, you think your Nameservers are Correct?”

Finding Errors Automatically in Nameserver Implementations

Siva Kesava Reddy Kakarla[1]

Ryan Beckett[2] Todd Millstein[1,3] George Varghese[1]

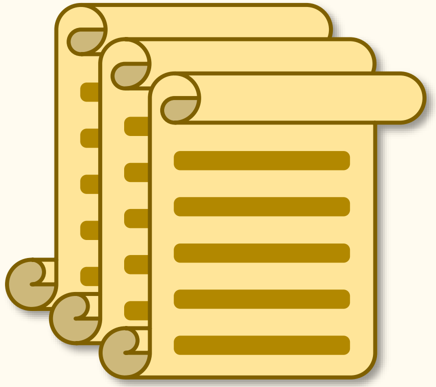
[1] University of California, Los Angeles

[2] Microsoft [3] Intentionet

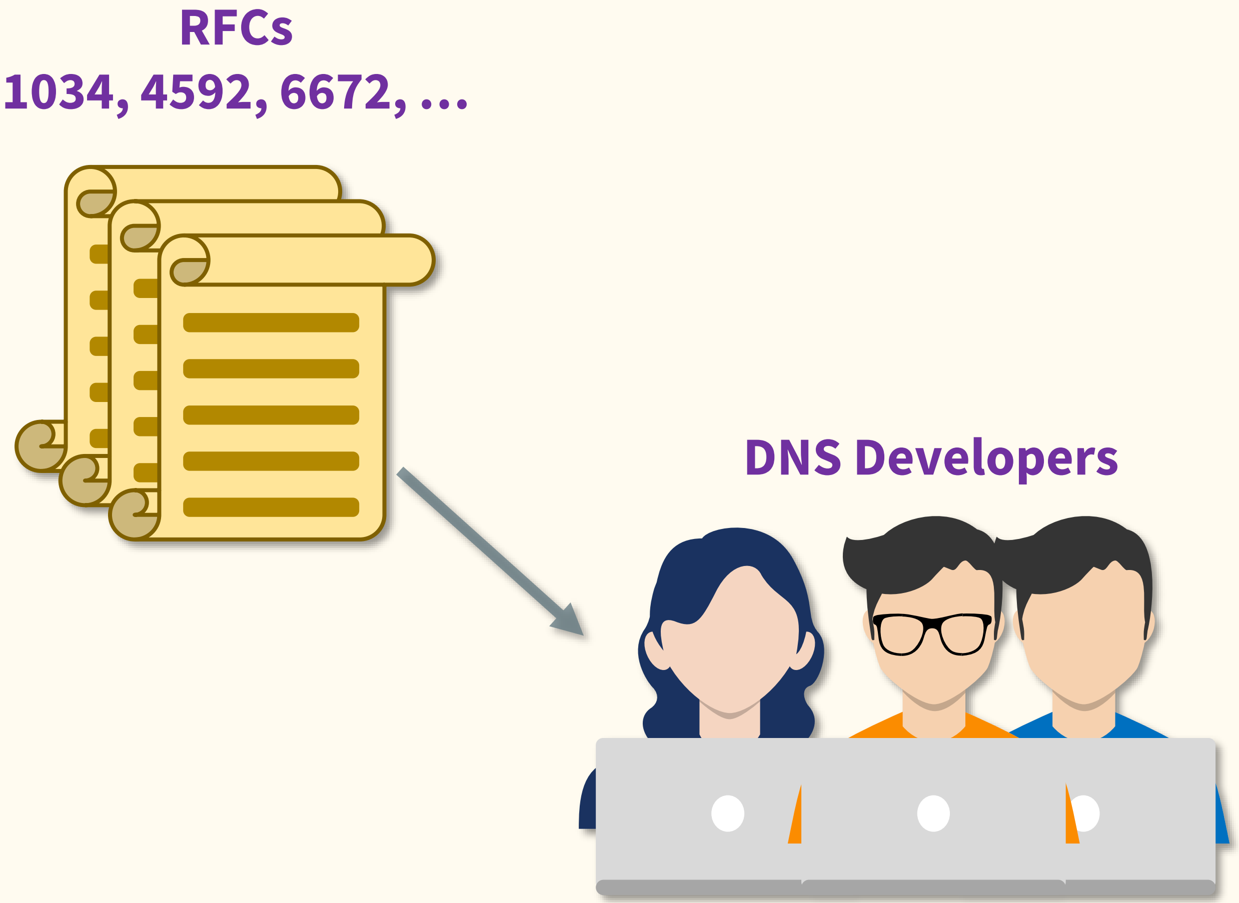
Implementing Nameservers Correctly is Hard!

Implementing Nameservers Correctly is Hard!

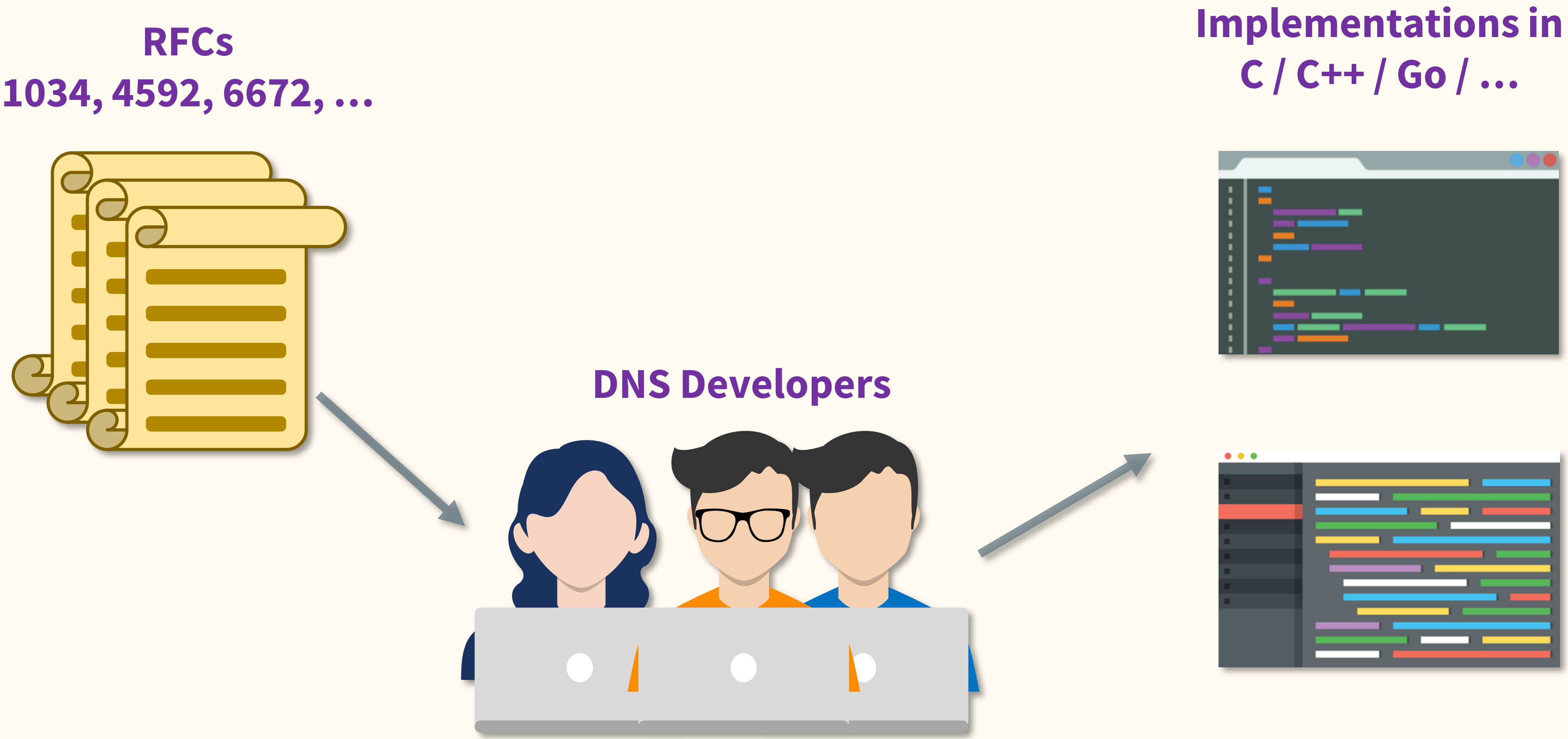
RFCs
1034, 4592, 6672, …



Implementing Nameservers Correctly is Hard!



Implementing Nameservers Correctly is Hard!

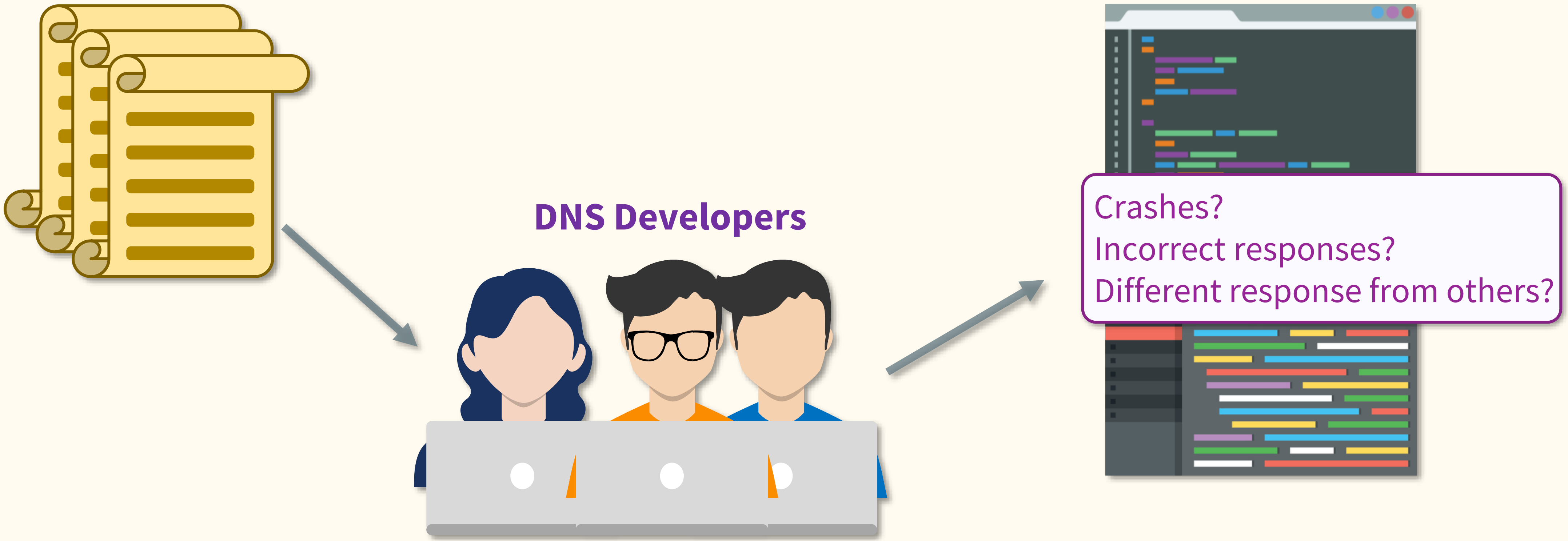


RFCs
1034, 4592, 6672, ...

DNS Developers

Implementations in
C / C++ / Go / ...

Implementing Nameservers Correctly is Hard!



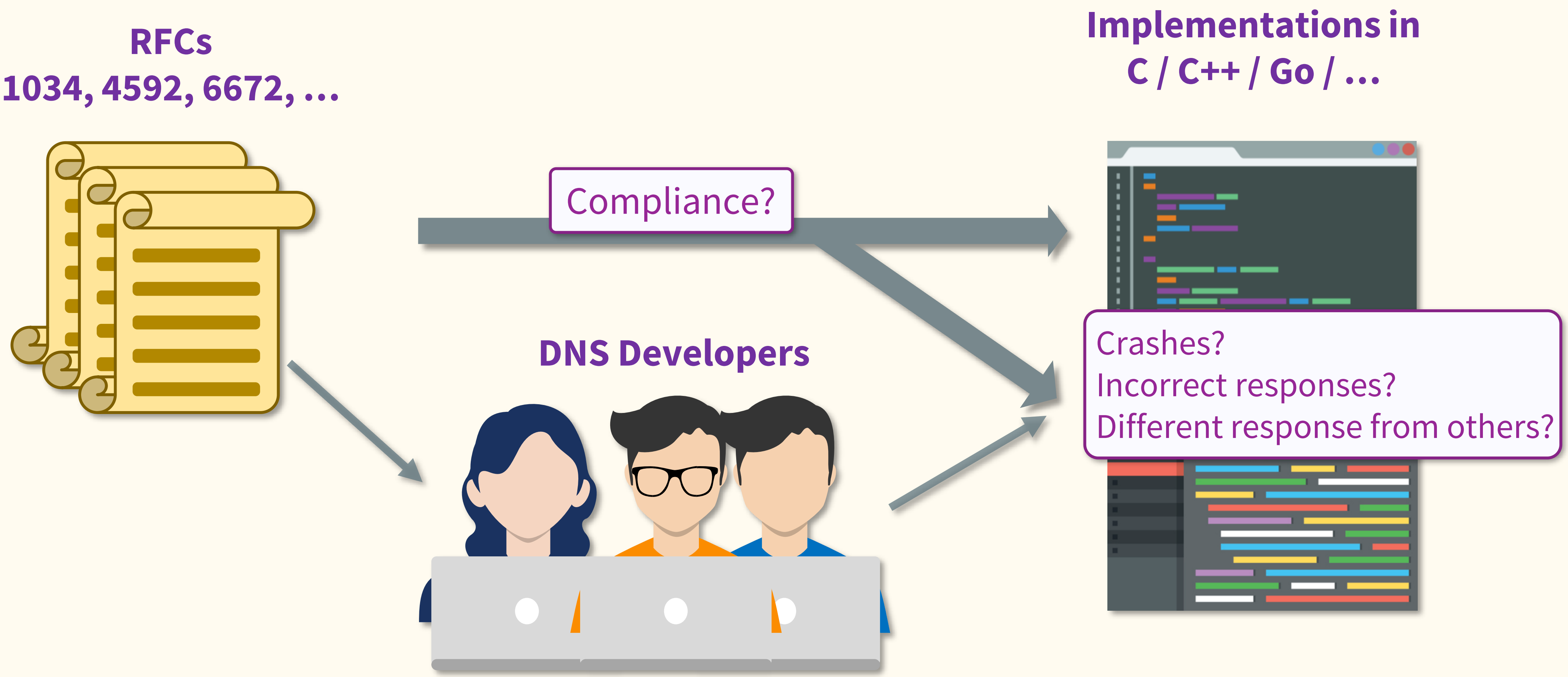
RFCs
1034, 4592, 6672, …

DNS Developers

Implementations in
C / C++ / Go / …

Crashes?
Incorrect responses?
Different response from others?

Implementing Nameservers Correctly is Hard!



RFCs
1034, 4592, 6672, …

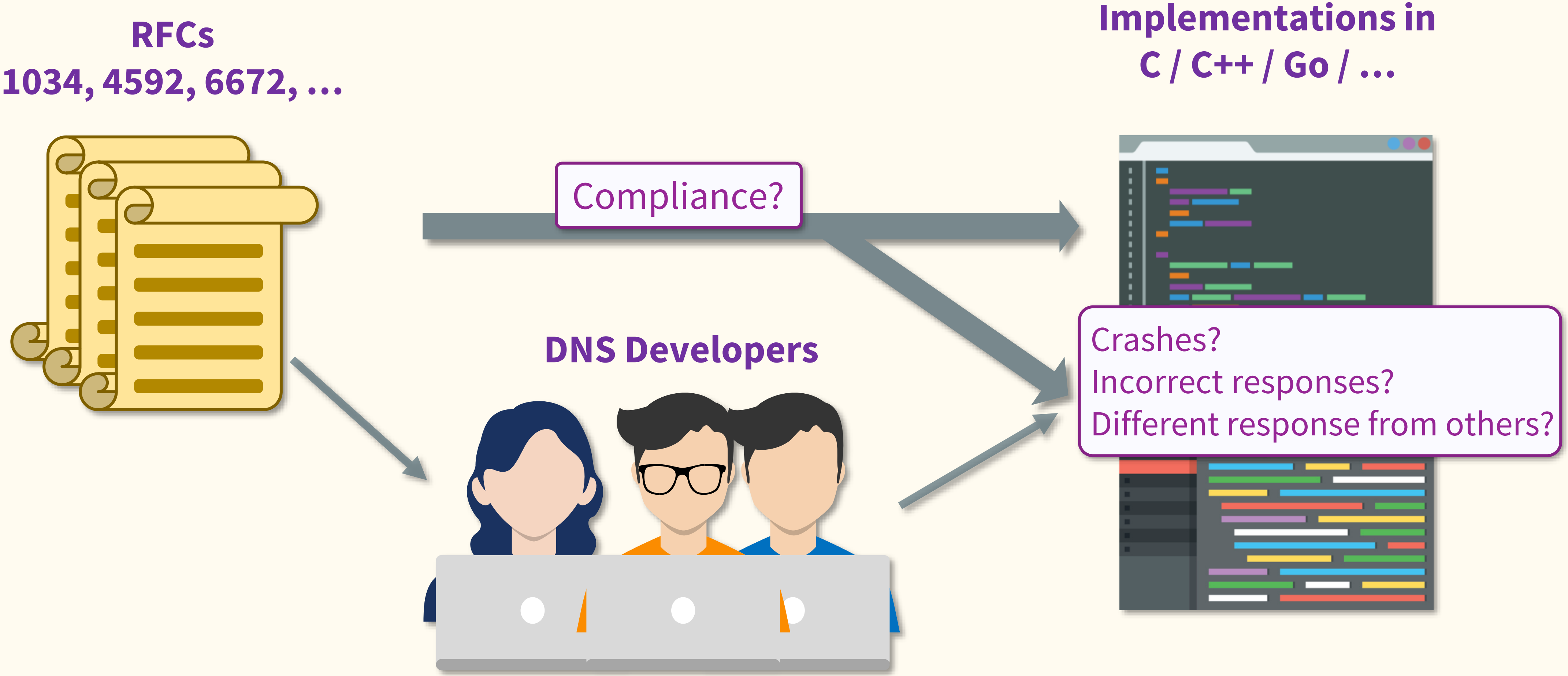
Implementations in
C / C++ / Go / …

Compliance?

DNS Developers

Crashes?
Incorrect responses?
Different response from others?

Current Practice: Ad Hoc Manual Testing



RFCs
1034, 4592, 6672, …

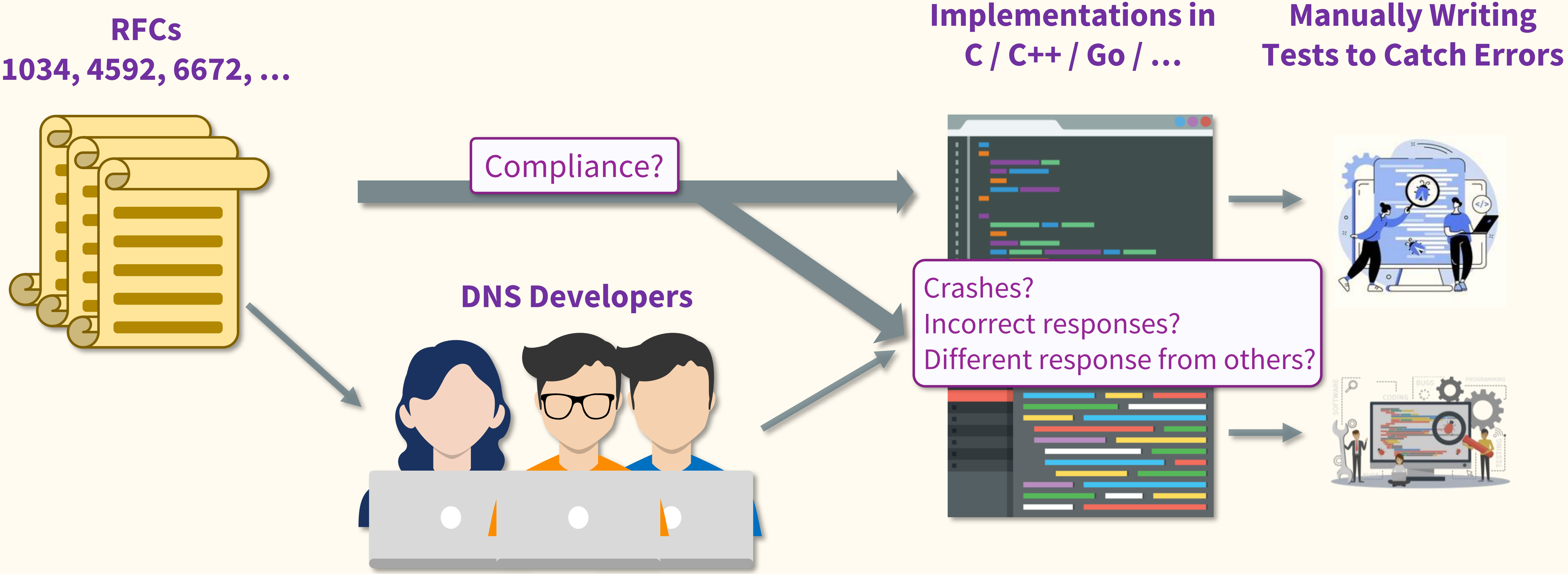
Compliance?

Implementations in
C / C++ / Go / …

DNS Developers

Crashes?
Incorrect responses?
Different response from others?

Current Practice: Ad Hoc Manual Testing



RFCs
1034, 4592, 6672, …

Compliance?

DNS Developers

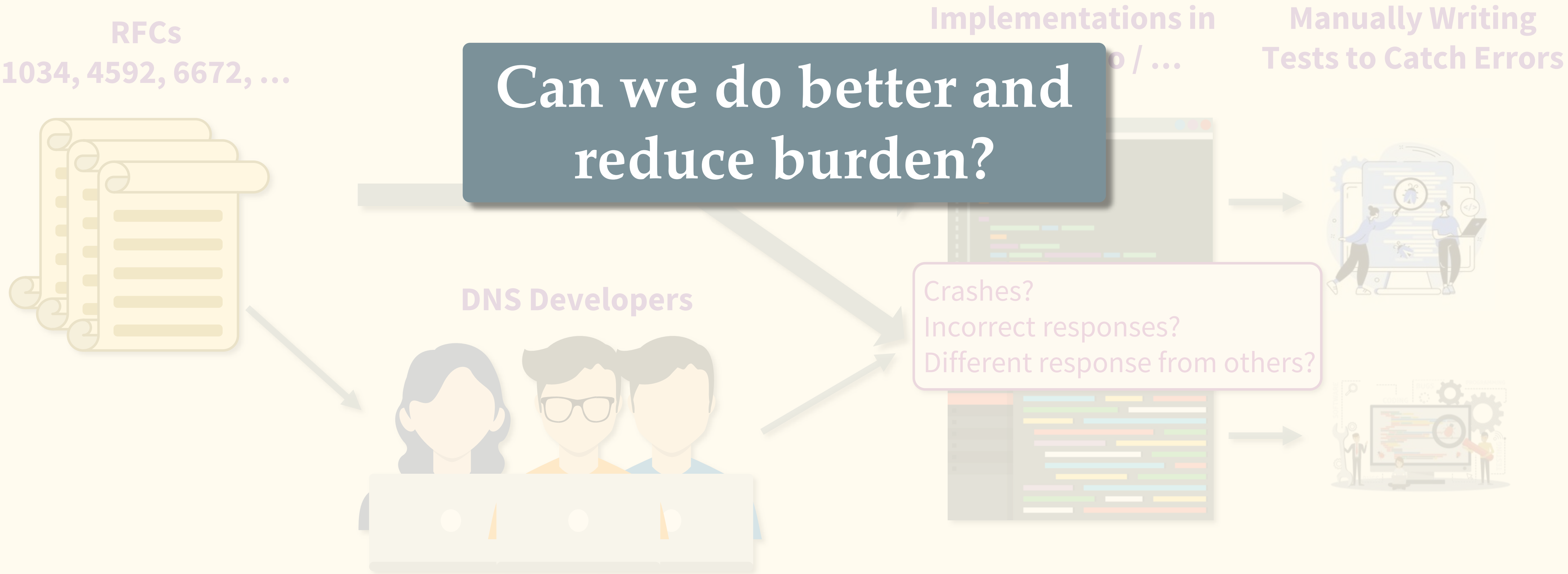
Implementations in
C / C++ / Go / …

Crashes?
Incorrect responses?
Different response from others?

Manually Writing
Tests to Catch Errors

Current Practice: Ad Hoc Manual Testing

Can we do better and reduce burden?



Crashes?
Incorrect responses?
Different response from others?

Current Practice: Ad Hoc Manual Testing

RFCs
1034, 4592, 6672, …
DNS Develo
Implementations in
o / …
Manually Writing
Tests to Catch Errors
rashes?
ncorrect responses?
Different response from others?

Can we do better and reduce burden?

Yes!!

Current Practice: Ad Hoc Manual Testing

Can we do better and reduce burden?

Yes!!

But how?

Our Idea:

FERRET - Generate tests *automatically* and compare across implementations

Our Idea:

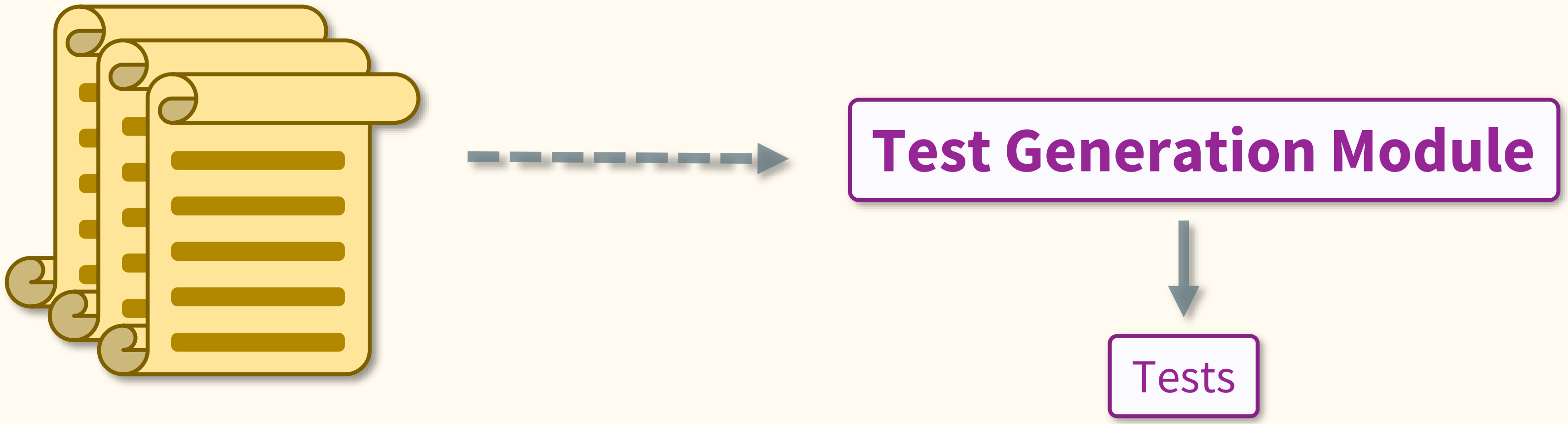
FERRET - Generate tests *automatically* and compare across implementations

How to generate high-coverage tests that identify functional correctness errors?

Ferret: End-to-End Design

RFCs
1034, 4592, 6672, ...

FERRET: End-to-End Design



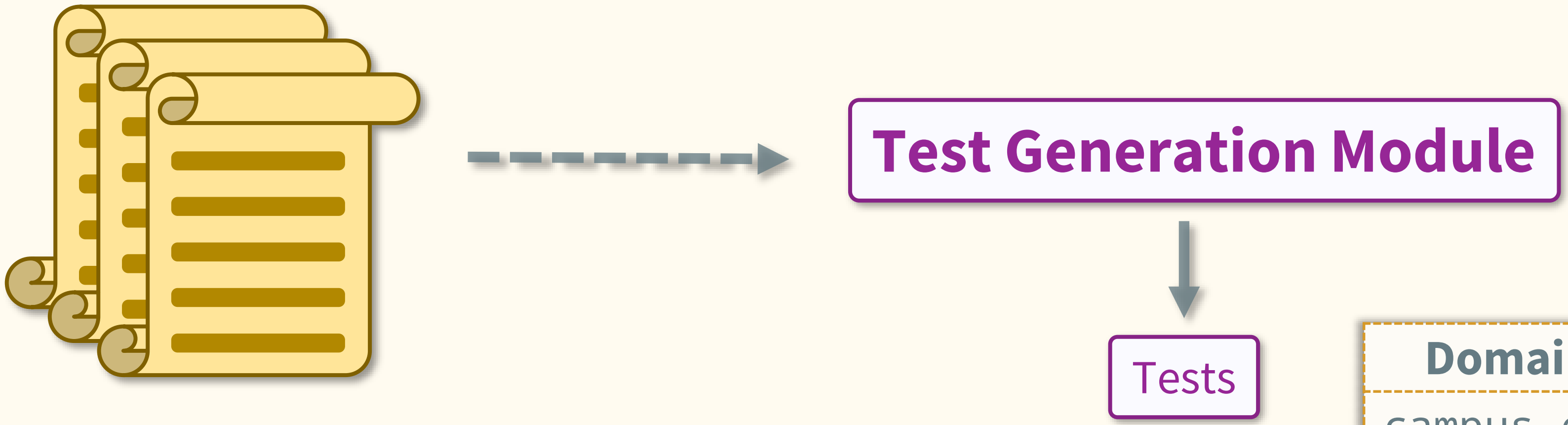
RFCs
1034, 4592, 6672, …

Test Generation Module

Tests

$\langle \text{zone file}_1, \text{query}_1 \rangle$ $\quad$ $\langle \text{zone file}_2, \text{query}_2 \rangle$ $\quad$ … $\quad$ $\langle \text{zone file}_n, \text{query}_n \rangle$

Ferret: End-to-End Design



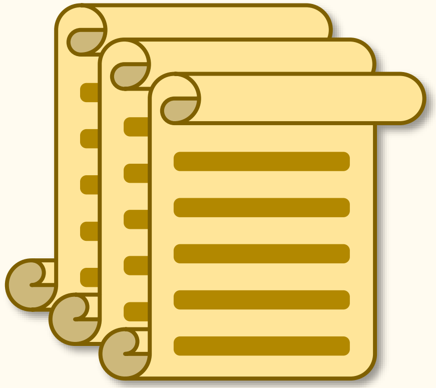
RFCs
1034, 4592, 6672, …

$\langle \text{zone file}_1, \text{query}_1 \rangle$ $\langle \text{zone file}_2,$

Domain Name	Type	Data
campus.edu.	SOA	ns1.exm. …
foo.campus.edu.	NS	ns1.campus.edu
ns1.campus.edu.	A	1.1.1.1

Query: ⟨anything.foo.campus.edu., A⟩

FERRET: End-to-End Design



RFCs
1034, 4592, 6672, …

Test Generation Module

Tests

$\langle \text{zone file}_1, \text{query}_1 \rangle$

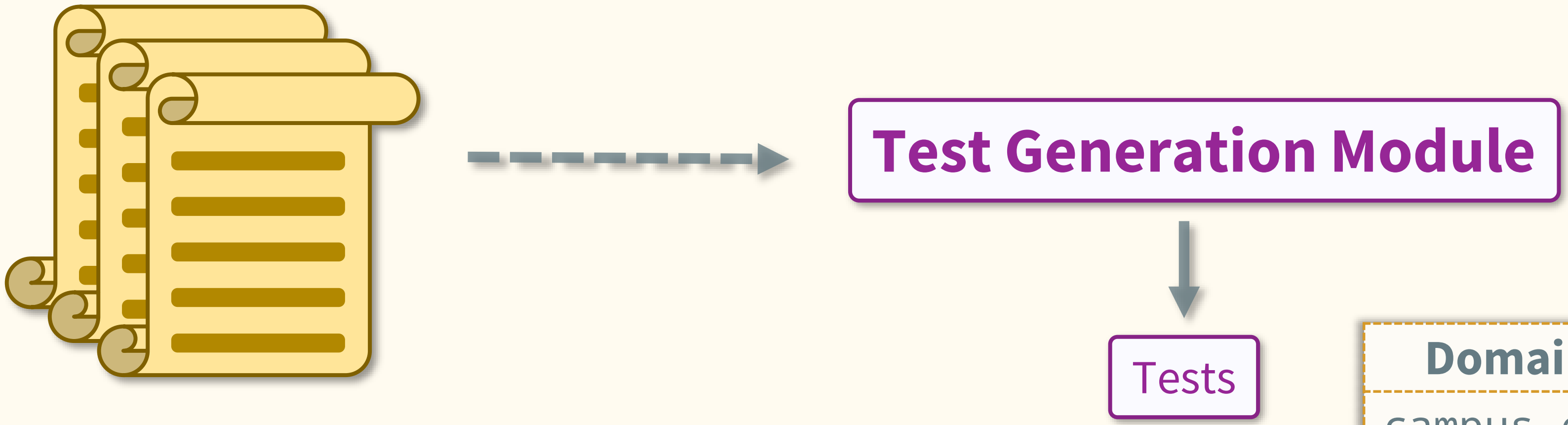
$\langle \text{zone file}_2,$

Domain Name	Type	Data
`campus.edu.`	SOA	`ns1.exm. …`
`foo.campus.edu.`	NS	`ns1.campus.edu`
`ns1.campus.edu.`	A	`1.1.1.1`

Query: ⟨`anything.foo.campus.edu., A`⟩

Modular Approach: Nameservers keep *no internal state* → A zone file is enough to test the *logic* at an isolated nameserver

FERRET: End-to-End Design



RFCs
1034, 4592, 6672, …

$\langle \text{zone file}_1, \text{query}_1 \rangle$ $\langle \text{zone file}_2,$

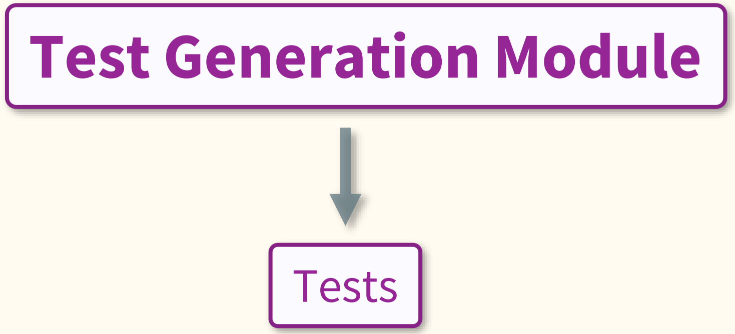
Domain Name	Type	Data
campus.edu.	SOA	ns1.exm. …
foo.campus.edu.	NS	ns1.campus.edu
ns1.campus.edu.	A	1.1.1.1

Query: ⟨anything.foo.campus.edu., A⟩

Modular Approach: Nameservers keep *no internal state* → A zone file is enough to test the *logic* at an isolated nameserver

FERRET generates tests that are *independent* of the *source code* → Can test any nameserver implementation

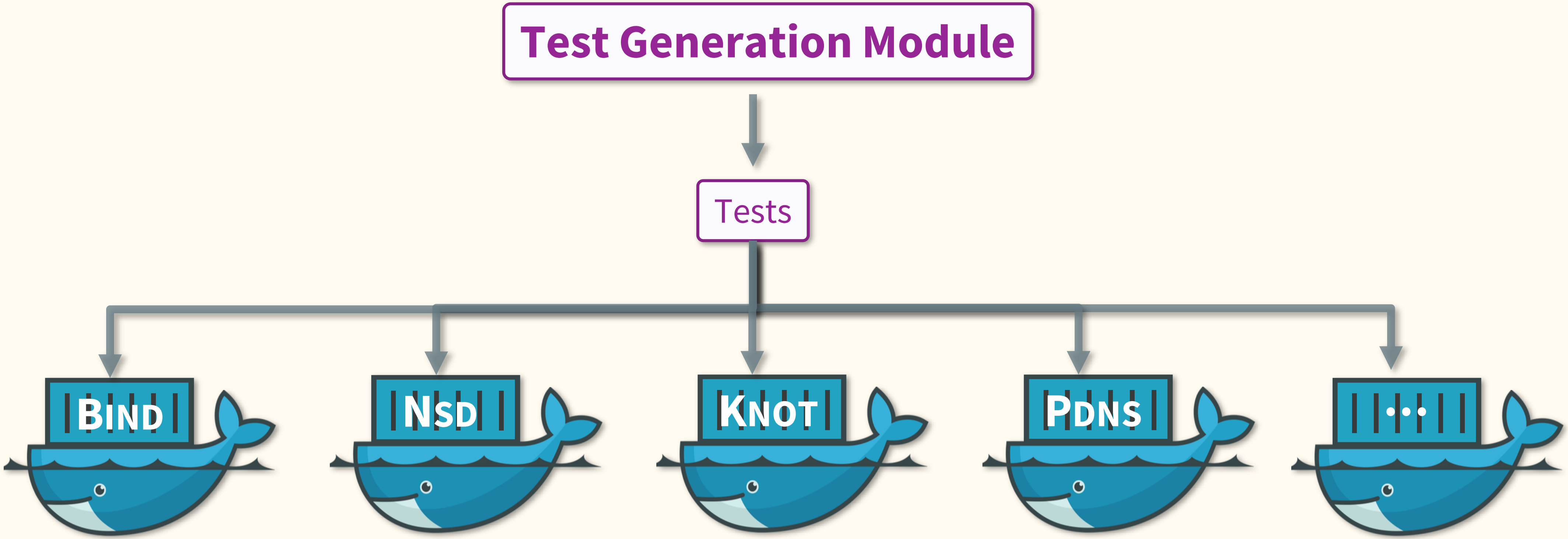
FERRET: End-to-End Design



Test Generation Module

Tests

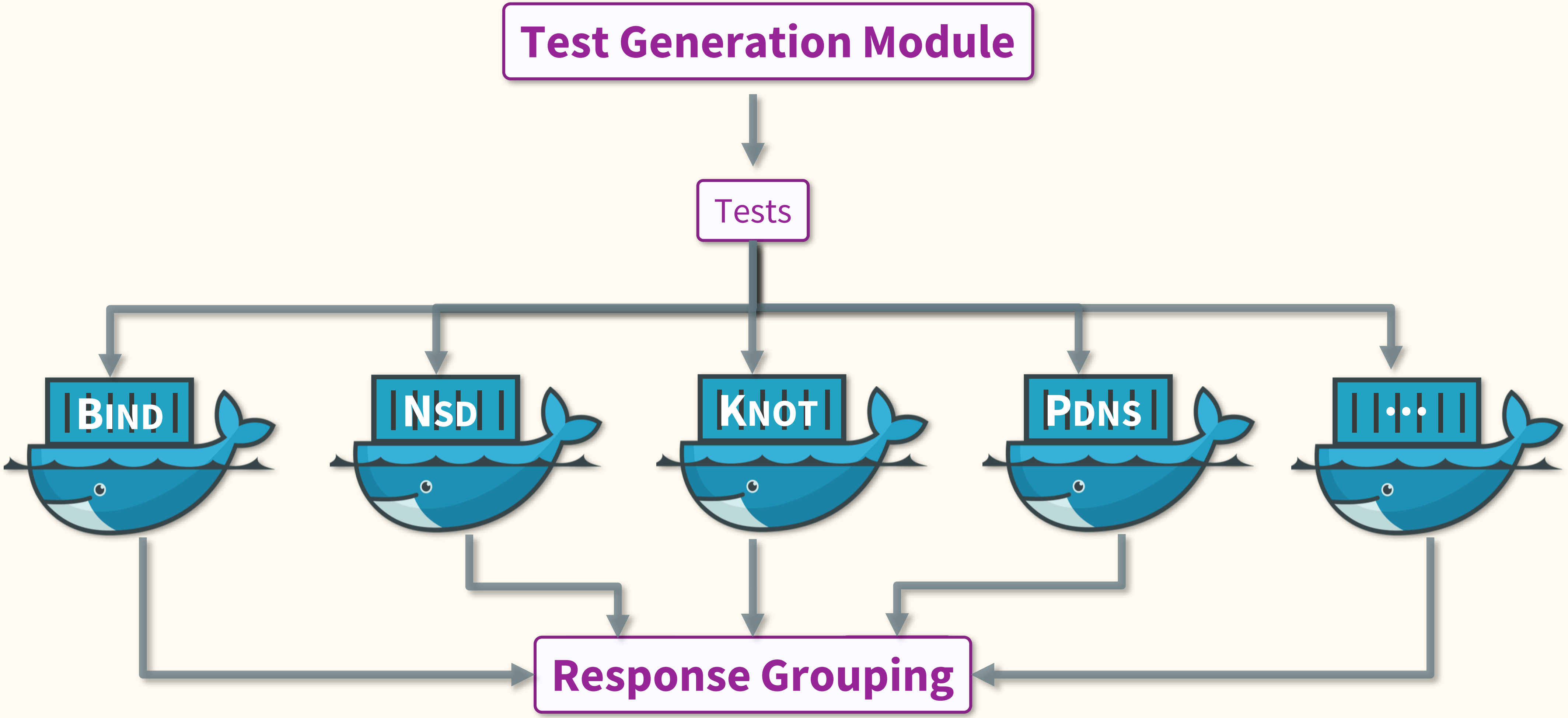
FERRET: End-to-End Design



Test Generation Module

Tests

FERRET: End-to-End Design

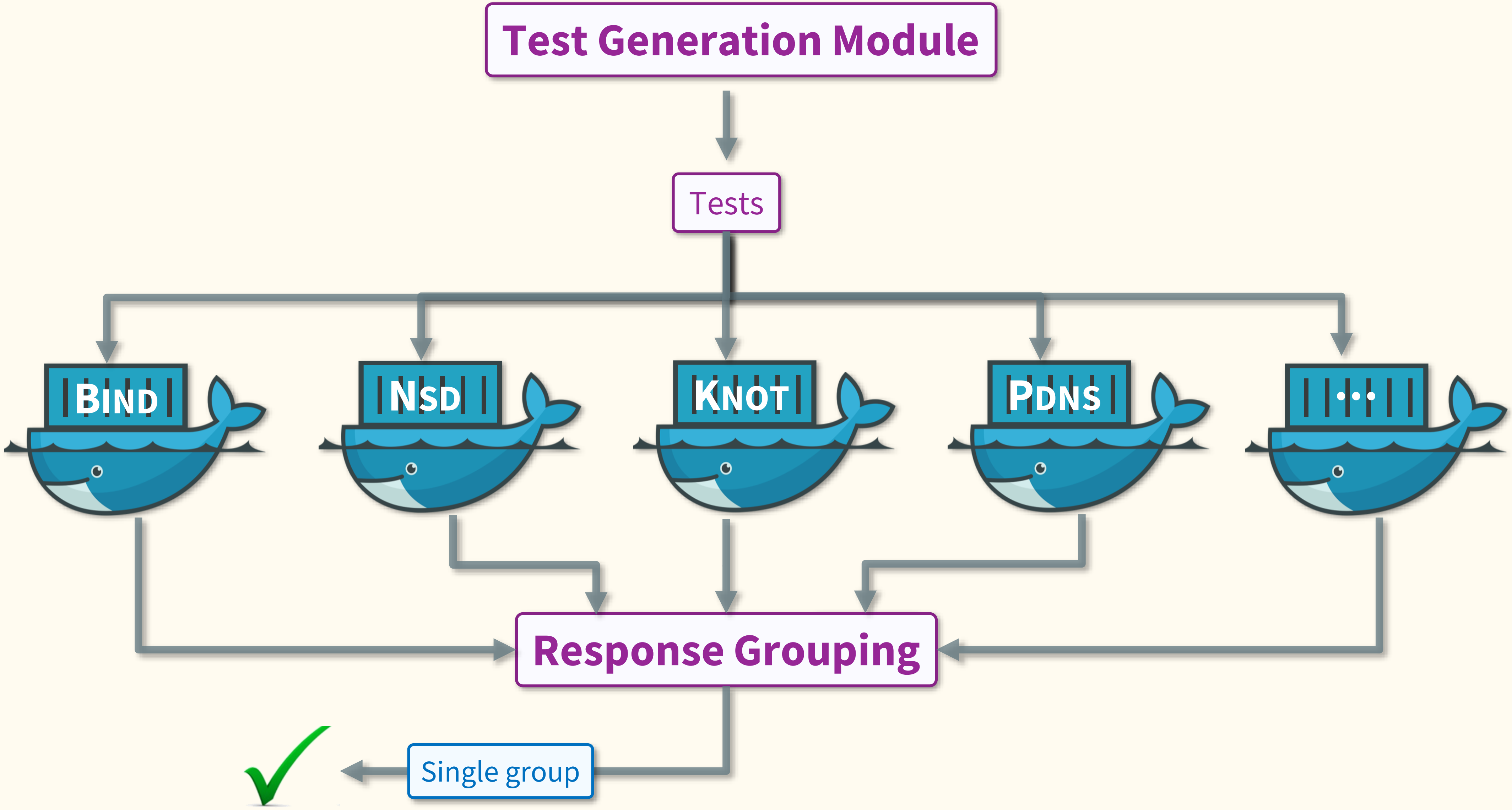


Test Generation Module

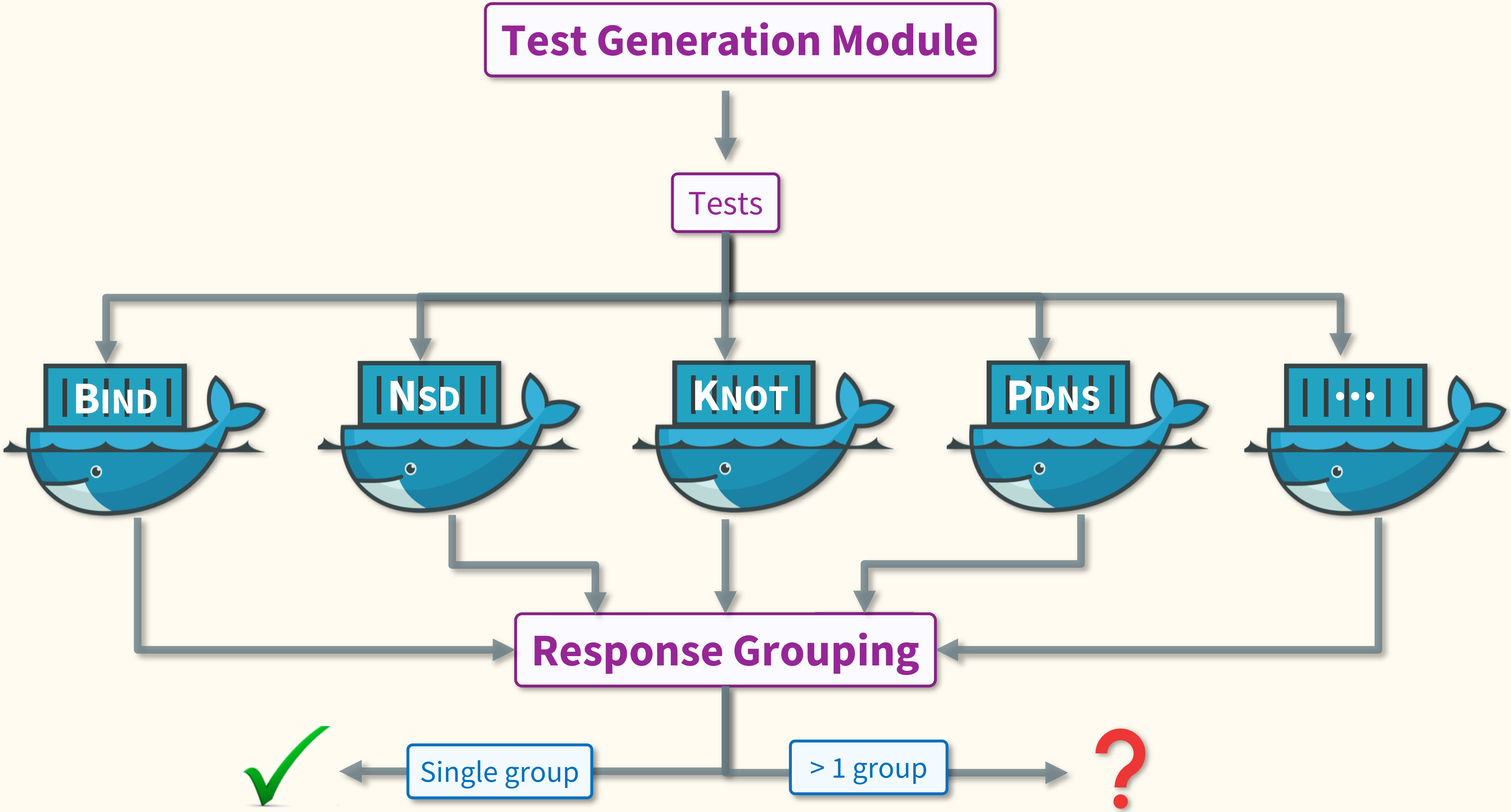
Tests

Response Grouping

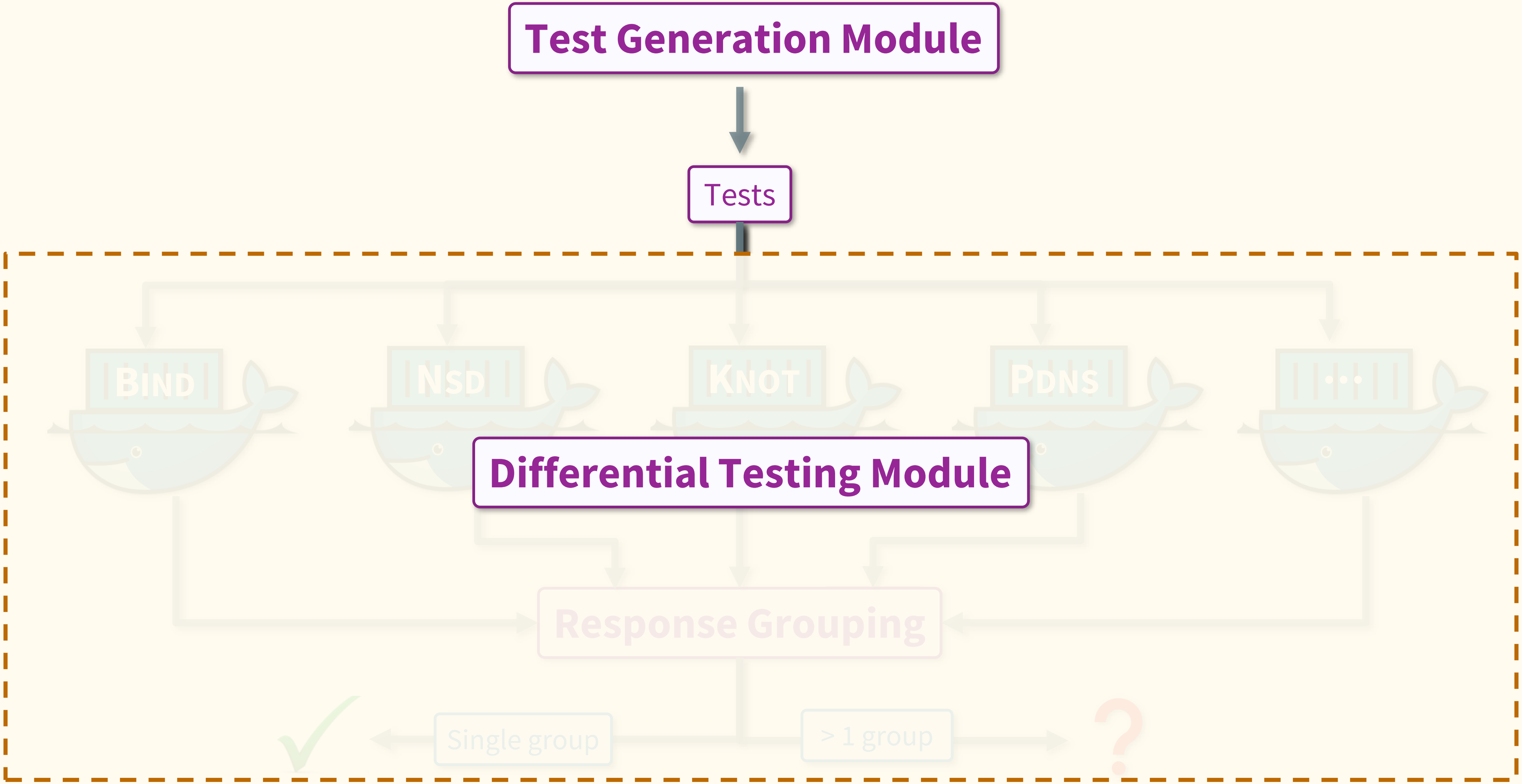
FERRET: End-to-End Design



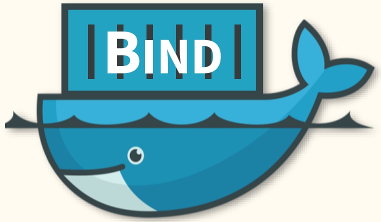
Ferret: End-to-End Design



FERRET: End-to-End Design



Differential Testing



Open-source Nameserver Implementations Tested

Implementation	Language	Description
Bind	**C**	*de facto* standard
PowerDns	**C++**	popular in North Europe
Nsd	**C**	hosts several TLDs
Knot	**C**	hosts several TLDs
CoreDns	**Go**	used in Kubernetes
Yadifa	**C**	created by EURid (.eu)
TRustDns	**Rust**	security, safety focused
MaraDns	**C**	lightweight server

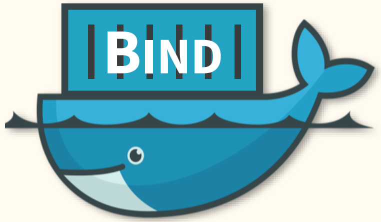
Differential Testing

Open-source Nameserver Implementations Tested

Implementation	Language	Description
Bind	**C**	*de facto* standard
PowerDns	**C++**	popular in North Europe
Nsd	**C**	hosts several TLDs
Knot	**C**	hosts several TLDs
CoreDns	**Go**	used in Kubernetes
Yadifa	**C**	created by EURid (.eu)
TRustDns	**Rust**	security, safety focused
MaraDns	**C**	lightweight server

- Docker image for each implementation

Differential Testing



Open-source Nameserver Implementations Tested

Implementation	Language	Description
Bind	**C**	*de facto* standard
PowerDns	**C++**	popular in North Europe
Nsd	**C**	hosts several TLDs
Knot	**C**	hosts several TLDs
CoreDns	**Go**	used in Kubernetes
Yadifa	**C**	created by EURid (.eu)
TRustDns	**Rust**	security, safety focused
MaraDns	**C**	lightweight server

- Docker image for each implementation
- Ferret starts a container for each image

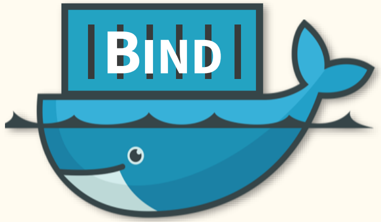
Differential Testing

Open-source Nameserver Implementations Tested

Implementation	Language	Description
Bind	C	*de facto* standard
PowerDns	C++	popular in North Europe
Nsd	C	hosts several TLDs
Knot	C	hosts several TLDs
CoreDns	Go	used in Kubernetes
Yadifa	C	created by EURid (.eu)
TRustDns	Rust	security, safety focused
MaraDns	C	lightweight server

- Docker image for each implementation
- Ferret starts a container for each image
- Unique host port is mapped to port 53 of the container

Differential Testing

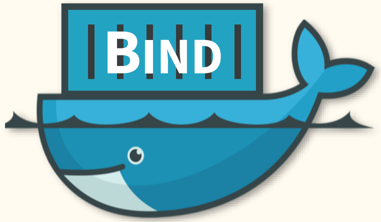


Open-source Nameserver Implementations Tested

Implementation	Language	Description
Bind	**C**	*de facto* standard
PowerDns	**C++**	popular in North Europe
Nsd	**C**	hosts several TLDs
Knot	**C**	hosts several TLDs
CoreDns	**Go**	used in Kubernetes
Yadifa	**C**	created by EURid (.eu)
TRustDns	**Rust**	security, safety focused
MaraDns	**C**	lightweight server

- Docker image for each implementation
- Ferret starts a container for each image
- Unique host port is mapped to port 53 of the container
- Each container servers one zone file at a time as an authoritative zone

Differential Testing

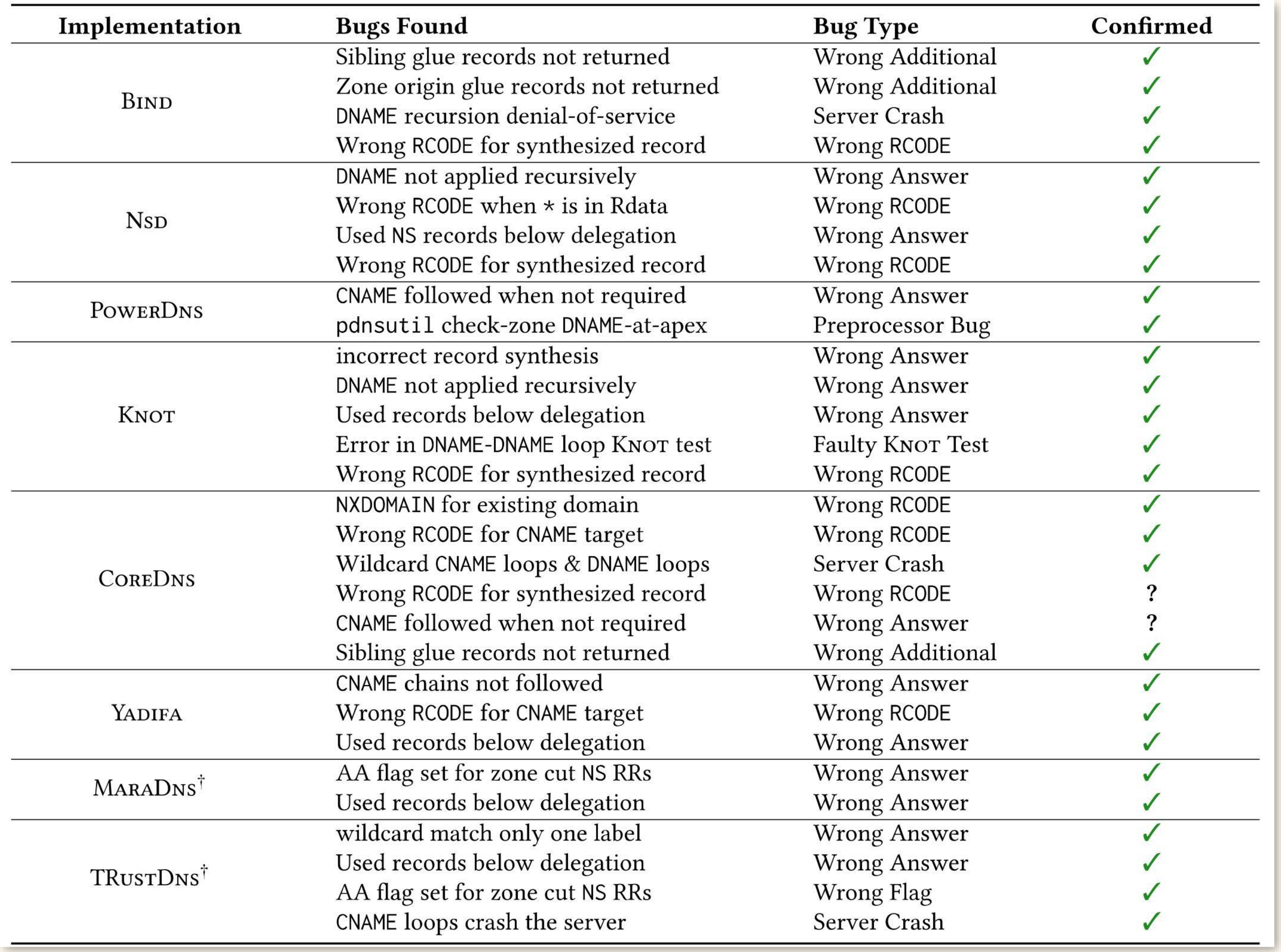


Open-source Nameserver Implementations Tested

Implementation	Language	Description
Bind	**C**	*de facto* standard
PowerDns	**C++**	popular in North Europe
Nsd	**C**	hosts several TLDs
Knot	**C**	hosts several TLDs
CoreDns	**Go**	used in Kubernetes
Yadifa	**C**	created by EURid (.eu)
TRustDns	**Rust**	security, safety focused
MaraDns	**C**	lightweight server

- Docker image for each implementation
- Ferret starts a container for each image
- Unique host port is mapped to port 53 of the container
- Each container servers one zone file at a time as an authoritative zone
- Ferret uses python library dnspython to send queries and collect responses

Bugs Found



Implementation	Bugs Found	Bug Type	Confirmed
Bind	Sibling glue records not returned	Wrong Additional	✓
	Zone origin glue records not returned	Wrong Additional	✓
	DNAME recursion denial-of-service	Server Crash	✓
	Wrong RCODE for synthesized record	Wrong RCODE	✓
Nsd	DNAME not applied recursively	Wrong Answer	✓
	Wrong RCODE when * is in Rdata	Wrong RCODE	✓
	Used NS records below delegation	Wrong Answer	✓
	Wrong RCODE for synthesized record	Wrong RCODE	✓
PowerDns	CNAME followed when not required	Wrong Answer	✓
	pdnsutil check-zone DNAME-at-apex	Preprocessor Bug	✓
Knot	incorrect record synthesis	Wrong Answer	✓
	DNAME not applied recursively	Wrong Answer	✓
	Used records below delegation	Wrong Answer	✓
	Error in DNAME-DNAME loop Knot test	Faulty Knot Test	✓
	Wrong RCODE for synthesized record	Wrong RCODE	✓
CoreDns	NXDOMAIN for existing domain	Wrong RCODE	✓
	Wrong RCODE for CNAME target	Wrong RCODE	✓
	Wildcard CNAME loops & DNAME loops	Server Crash	✓
	Wrong RCODE for synthesized record	Wrong RCODE	?
	CNAME followed when not required	Wrong Answer	?
	Sibling glue records not returned	Wrong Additional	✓
Yadifa	CNAME chains not followed	Wrong Answer	✓
	Wrong RCODE for CNAME target	Wrong RCODE	✓
	Used records below delegation	Wrong Answer	✓
MaraDns[†]	AA flag set for zone cut NS RRs	Wrong Answer	✓
	Used records below delegation	Wrong Answer	✓
TRustDns[†]	wildcard match only one label	Wrong Answer	✓
	Used records below delegation	Wrong Answer	✓
	AA flag set for zone cut NS RRs	Wrong Flag	✓
	CNAME loops crash the server	Server Crash	✓

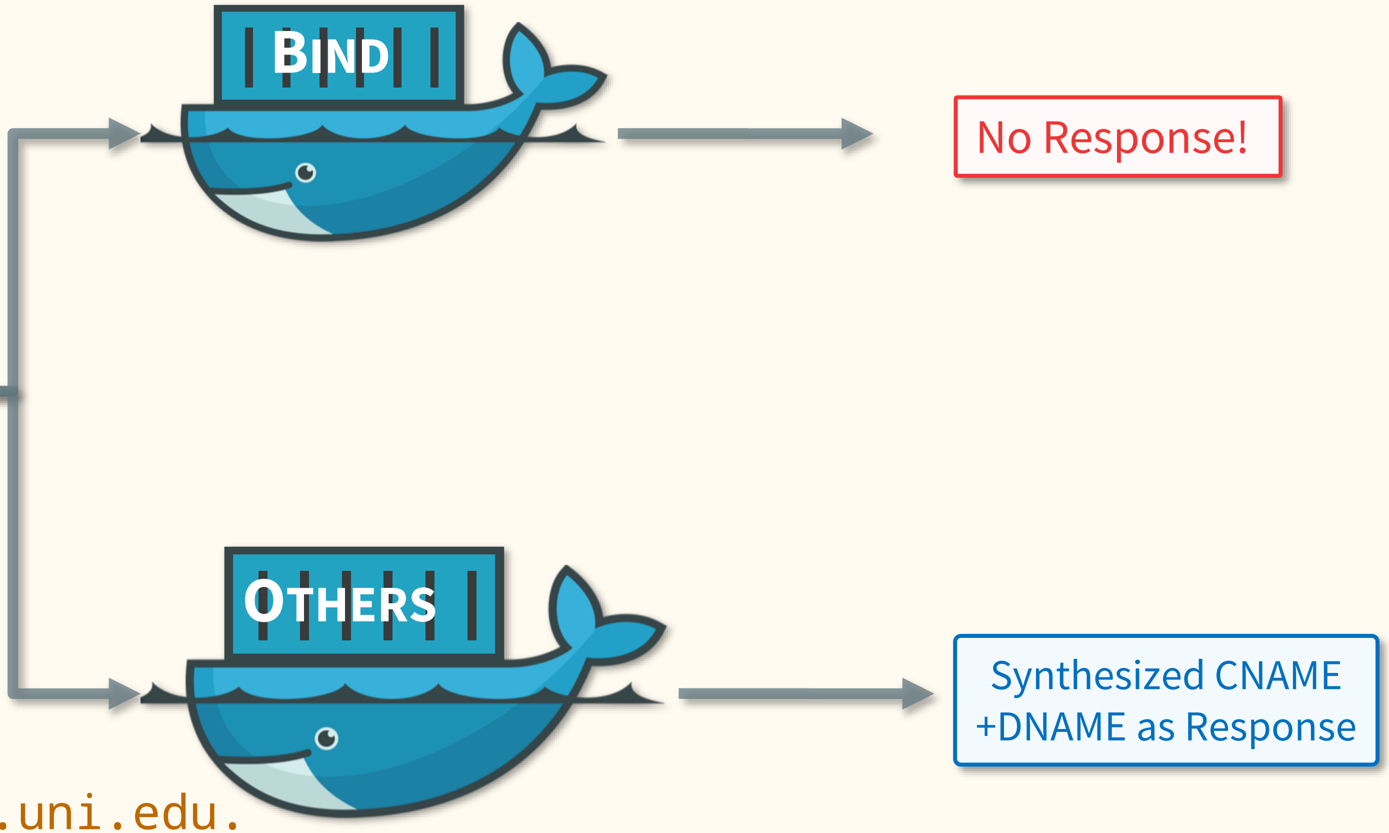
[†] Implementations with unreported issues due to missing or unimplemented features

Example Bugs

FERRET Generated Test Case

Domain Name	Type	Data
uni.edu	SOA	ns1.exm. …
test.uni.edu.	DNAME	edu.

Query: ⟨`test.uni.test.uni.edu.`,DNAME⟩



- Query is rewritten using DNAME to:
  `test.uni.test.uni.edu. CNAME test.uni.edu.`

- The rewritten query will match exactly with the DNAME record.

Example Bugs

Crash in BIND

BIND

FERRET Generated

Domain Name	Ty	
uni.edu	SO	
test.uni.edu.	DNAME	edu.

```
/lib/x86_64-linux-gnu/libpthread.so.0(+0x76db) [0x7f2094e876db]
/lib/x86_64-linux-gnu/libc.so.6(clone+0x3f) [0x7f209498b71f]
exiting (due to assertion failure)
Aborted (core dumped)
fatal error: stack overflow
```

NAME⟩

- Server Crashes !!
- Easily-weaponizable denial-of-service vector
- Remotely Exploitable
- Affected all currently maintained BIND 9 branches

Initiated a responsible disclosure

CVE-2021-25215 (High Severity): An assertion check can fail while answering queries for DNAME records that require the DNAME to be processed to resolve itself

Example Bugs

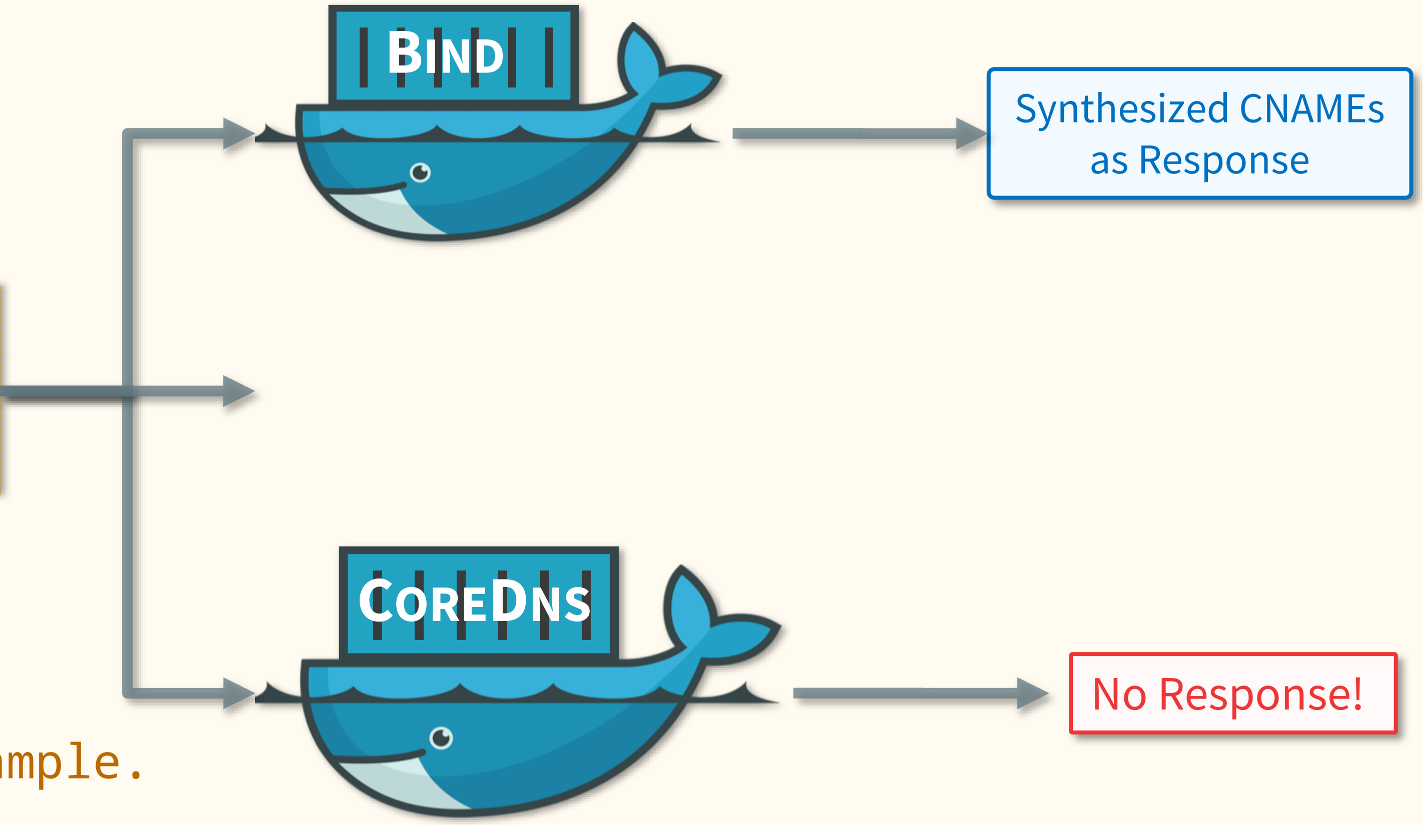
FERRET Generated Test Case

Domain Name	Type	Data
`example.`	`SOA`	`ns1.exm. …`
`*.example.`	`CNAME`	`foo.example.`

Query: ⟨`baz.bar.example., CNAME`⟩

- Query is rewritten using CNAME to:
  `baz.bar.example. CNAME foo.example.`

- The rewritten query will match the wildcard again !



BIND

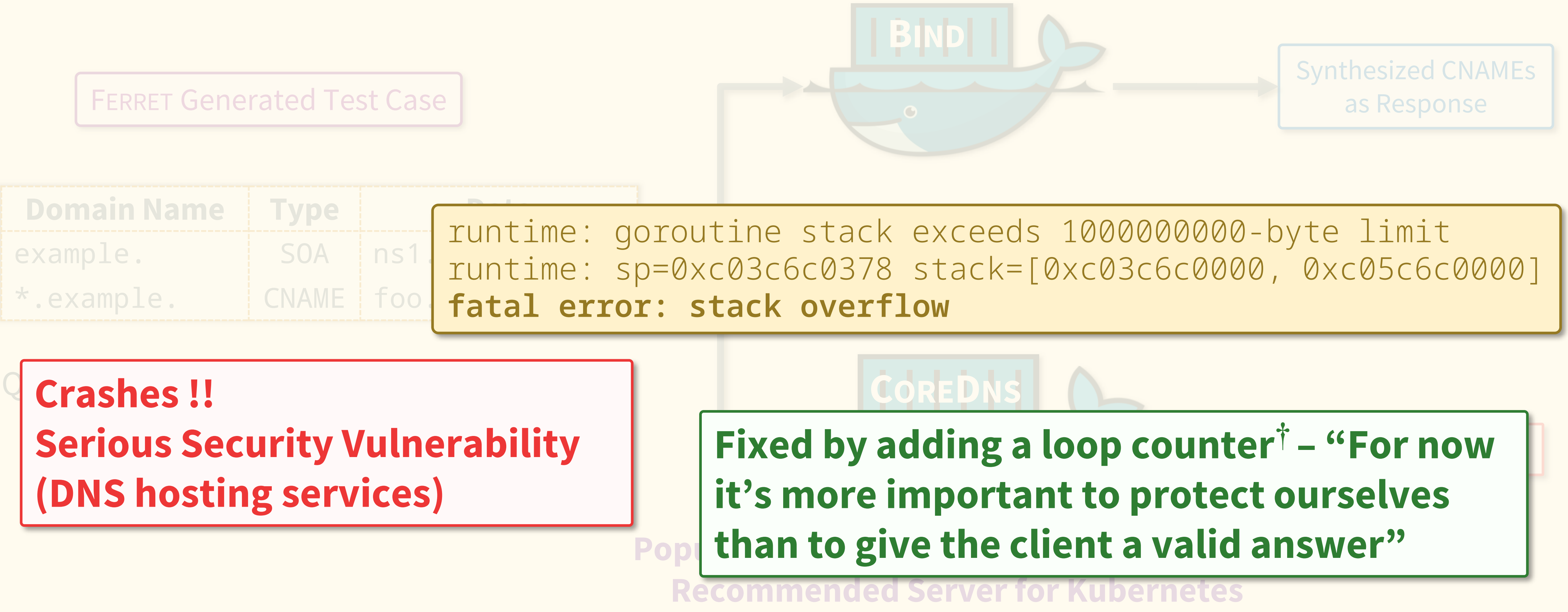
Synthesized CNAMEs as Response

COREDNS

No Response!

Popular open-sourced server written in Go
Recommended Server for Kubernetes

Example Bugs


FERRET Generated Test Case
Domain Name
Type
example.
SOA
ns1.
*.example.
CNAME
foo.
BIND
Synthesized CNAMEs as Response
COREDNS
Popu
Recommended Server for Kubernetes

```
runtime: goroutine stack exceeds 1000000000-byte limit
runtime: sp=0xc03c6c0378 stack=[0xc03c6c0000, 0xc05c6c0000]
fatal error: stack overflow
```

Crashes !!
Serious Security Vulnerability (DNS hosting services)

Fixed by adding a loop counter[†] – “For now it’s more important to protect ourselves than to give the client a valid answer”

[†] https://github.com/coredns/coredns/issues/4378

Example Bugs

Performance Bug in BIND

Open issue - May 2021 milestone

Domain Name	Type	Data
`campus.edu.`	SOA	`ns1.exm. …`
`foo.campus.edu.`	NS	`ns1.campus.edu`
`ns1.campus.edu.`	A	`1.1.1.1`

Query: ⟨`anything.foo.campus.edu., A`⟩

Response from POWERDNS, KNOT, NSD:

```
Authority Section:
    foo.campus.edu.    NS   ns1.campus.edu
Additional Section:
    ns1.campus.edu.    A    1.1.1.1
```

- ✦ BIND does not return the glue record
- ✦ Response from BIND "This report turns out to be very interesting. Here is what I managed to find out"
- ✦ BIND uses a "glue cache" to speed up the identification of glue records, but it had two unrelated errors.
  - ↳ If the cache lookup fails, then glue records are supposed to be searched for in the zone file, but the latter was never happening.
  - ↳ glue records for siblings domain nameservers were accidentally never searched for at all.

https://gitlab.isc.org/isc-projects/bind9/-/issues/2384

Example Bugs

Data Structure Bug in NSD

Fixed the issue

Domain Name	Type	Data
`booksonline.`	`SOA`	`ns1.exm. …`
`buy.booksonline.`	`CNAME`	`www.*.booksonline.`

Query: ⟨`buy.booksonline., A`⟩

Response from NSD:

RCODE: **NOERROR**

Answer Section:

```
   buy.booksonline. CNAME www.*.booksonline.
```

- BIND, KNOT, POWERDNS return with **NXDOMAIN** as CNAME target does not exist
- RCODE is important as resolvers use it to determine whether domains exist or not
- NSD responded - “It has to do with the internal data structure for storing domains in the memory of NSD, there a domain struct is created for the right hand of the CNAME, and it is set to be non-existing. The is_existing was not checked for the wildcard expansion, and this is fixed by the commit. …**Thanks for the report**!”

https://github.com/NLnetLabs/nsd/issues/152

Example Bugs

CNAME Bug in Yadifa

Fixed the issue

Domain Name	Type	Data
`dept.com.`	SOA	`ns1.exm. …`
`www.cs.dept.com.`	CNAME	`cs.dept.com.`
`cs.dept.com.`	CNAME	`dept.com`
`dept.com.`	A	`2.2.2.2`

Query: ⟨`www.cs.dept.com., A`⟩

- ✦ Expected response is to rewrite the query twice and return the IP record
- ✦ Yadifa rewrote it only once and was not following the CNAME chains.
- ✦ CNAME chains are used extensively by CDNs so its important to follow
- ✦ Yadifa acknowledged and said – “The rerun of the query was incorrectly disabled, the issue is fixed and will be updated on github on our next update of the code.”

https://github.com/yadifa/yadifa/issues/10

Example Bugs

DNAME-DNAME Loop Bug in KNOT

Domain Name	Type	Data
corp.	SOA	ns1.exm. …
corp.	NS	ns1.com.
corp.	DNAME	us.corp.

Query: ⟨`www.corp., NS`⟩

- Query is rewritten using DNAME to:
  `www.corp. CNAME www.us.corp.`

- The rewritten query will again be rewritten using DNAME to:
  `www.us.corp. CNAME www.us.us.corp.`

- Leads to an infinite recursion !!

- ✦ BIND applies DNAME multiple times and stops when limit reaches 17

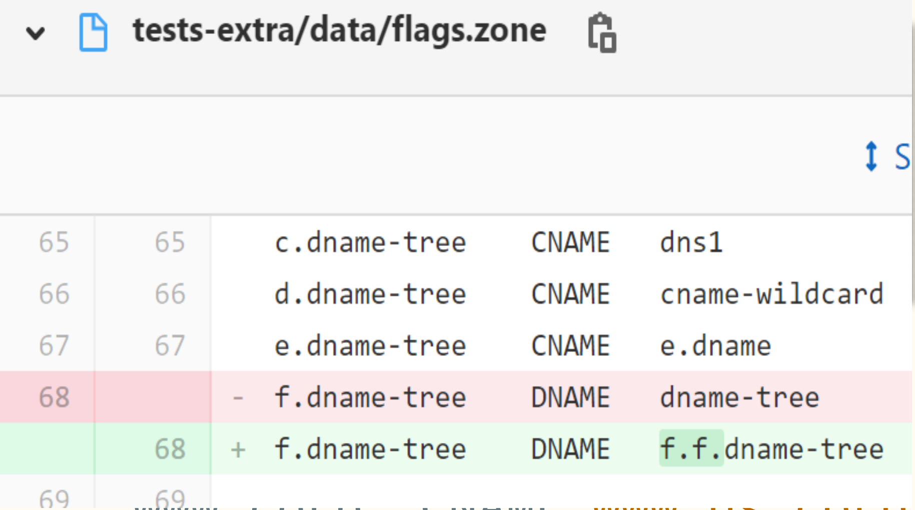
- ✦ POWERDNS returns **SERVFAIL**

- ✦ KNOT and NSD applied DNAME only once
  - Works here but had to be applied multiple times when there is no loop
  - Both fixed the issue [†]

[†] https://github.com/NLnetLabs/nsd/issues/151
[†] https://gitlab.nic.cz/knot/knot-dns/-/issues/714

Example Bugs

DNAME-DNAME Loop Bug in KNOT

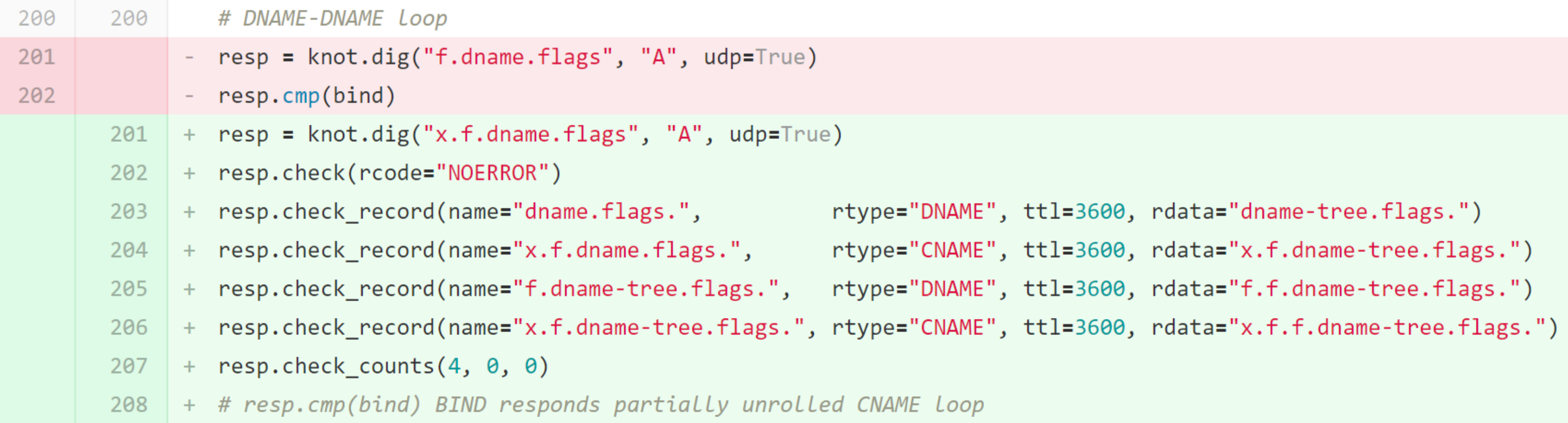


tests-extra/data/flags.zone

```
   c.dname-tree    CNAME   dns1
   d.dname-tree    CNAME   cname-wildcard
   e.dname-tree    CNAME   e.dname
-  f.dname-tree    DNAME   dname-tree
+  f.dname-tree    DNAME   f.f.dname-tree
```

www.corp. CNAME www.us.corp.

- ✦ BIND applies DNAME multiple times and stops when limit reaches 17



```
  # DNAME-DNAME Loop
- resp = knot.dig("f.dname.flags", "A", udp=True)
- resp.cmp(bind)
+ resp = knot.dig("x.f.dname.flags", "A", udp=True)
+ resp.check(rcode="NOERROR")
+ resp.check_record(name="dname.flags.",          rtype="DNAME", ttl=3600, rdata="dname-tree.flags.")
+ resp.check_record(name="x.f.dname.flags.",      rtype="CNAME", ttl=3600, rdata="x.f.dname-tree.flags.")
+ resp.check_record(name="f.dname-tree.flags.",   rtype="DNAME", ttl=3600, rdata="f.f.dname-tree.flags.")
+ resp.check_record(name="x.f.dname-tree.flags.", rtype="CNAME", ttl=3600, rdata="x.f.f.dname-tree.flags.")
+ resp.check_counts(4, 0, 0)
+ # resp.cmp(bind) BIND responds partially unrolled CNAME loop
```

- The rewritten query will again be rewritten using DNAME to:
  `www.us.corp. CNAME www.us.us.corp.`
- Leads to an infinite recursion !!

† https://github.com/NLnetLabs/nsd/issues/151
† https://gitlab.nic.cz/knot/knot-dns/-/issues/714
★ https://gitlab.nic.cz/knot/knot-dns/-/issues/703

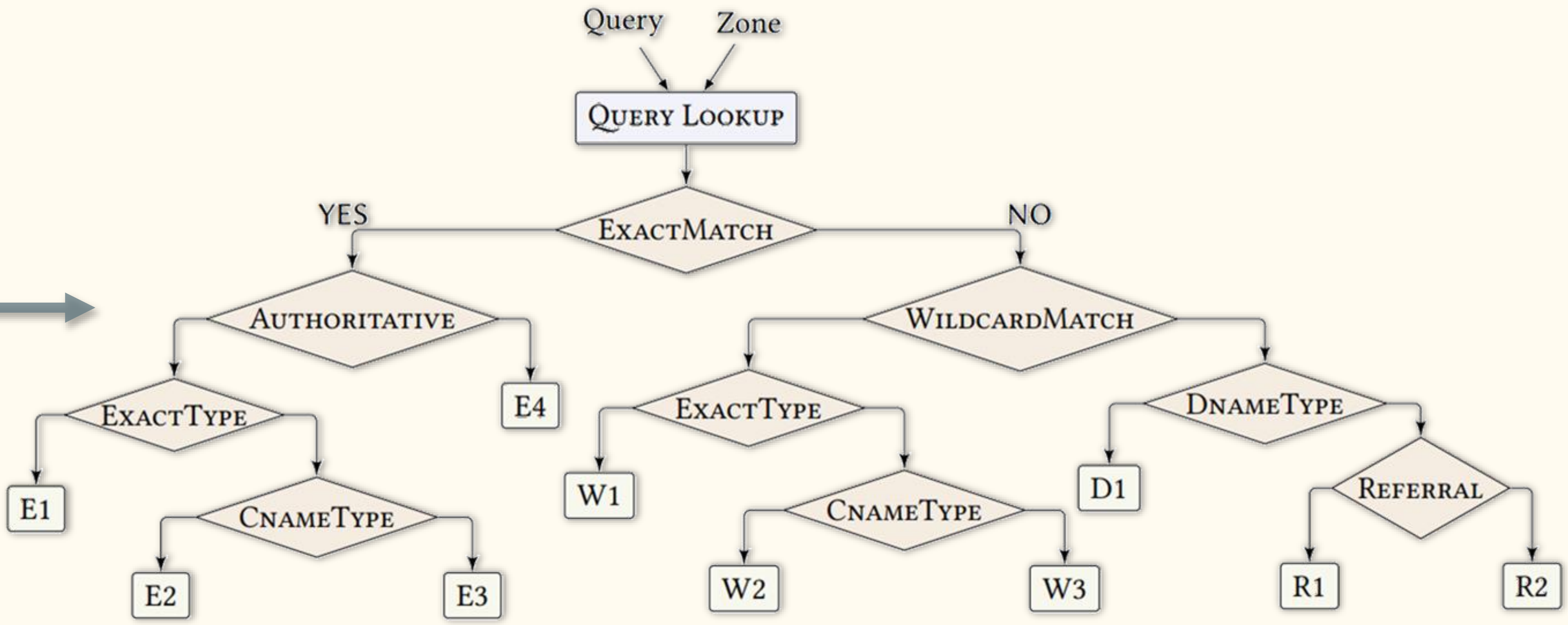
- ✦ KNOT had a test suite comparing responses with BIND and a test is mentioned as testing the infinite loop as this
  - ↘ Test zone file was not properly constructed, and that error led to having no loop
  - ↘ Fixed it★ and went with single response unlike 17 for a loop

Test Generation Module

Test Generation Module

RFCs
1034, 4592, 6672, …

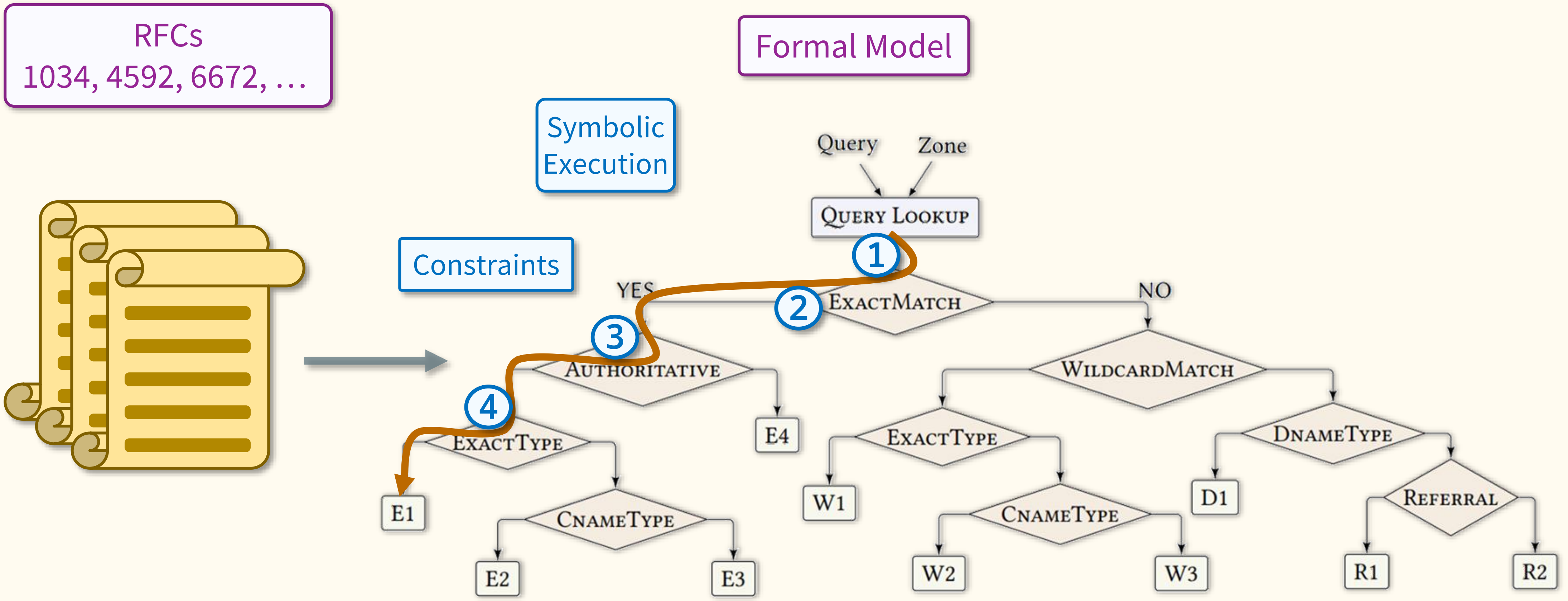
Formal Model †



English → Declarative (Mathematical) specification of the nameserver logic

†GROOT: Proactive Verification of DNS Configurations – Siva Kakarla *et al.*, SIGCOMM 2020

Test Generation Module



RFCs
1034, 4592, 6672, …

Formal Model

Symbolic Execution

Constraints

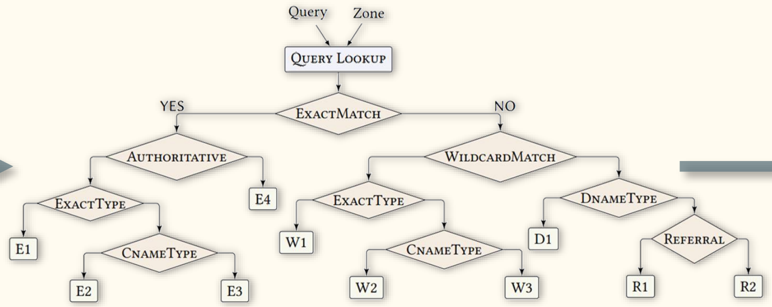
Solve(①,②,③,④) for inputs→ $\langle z_1, q_1\rangle, \langle z_2, q_2\rangle, \ldots \langle z_n, q_n\rangle$

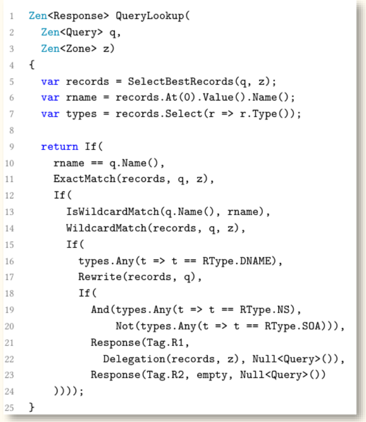
Test Generation Module

RFCs
1034, 4592, 6672, …

Formal Model

Executable Version in Zen





```
Zen<Response> QueryLookup(
  Zen<Query> q,
  Zen<Zone> z)
{
  var records = SelectBestRecords(q, z);
  var rname = records.At(0).Value().Name();
  var types = records.Select(r => r.Type());

  return If(
    rname == q.Name(),
    ExactMatch(records, q, z),
    If(
      IsWildcardMatch(q.Name(), rname),
      WildcardMatch(records, q, z),
      If(
        types.Any(t => t == RType.DNAME),
        Rewrite(records, q),
        If(
          And(types.Any(t => t == RType.NS),
              Not(types.Any(t => t == RType.SOA))),
          Response(Tag.R1,
            Delegation(records, z), Null<Query>()),
          Response(Tag.R2, empty, Null<Query>())
    ))));
}
```

An **executable version** of formal model is implemented in **Zen**, a domain-specific modeling language embedded in **C#** with built-in support for **symbolic execution**

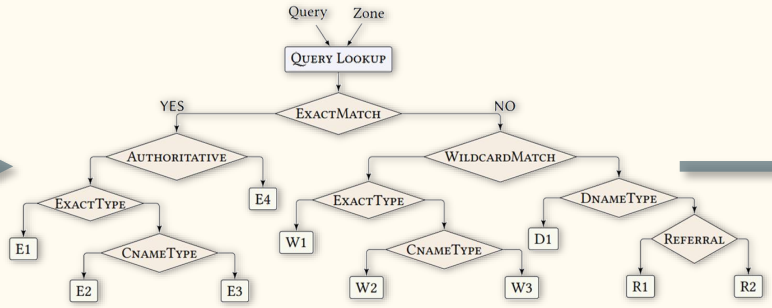
Test Generation Module

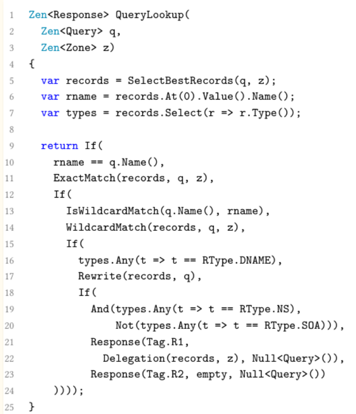
RFCs
1034, 4592, 6672, …

Formal Model

Executable Version in Zen

Tests





```
Zen<Response> QueryLookup(
  Zen<Query> q,
  Zen<Zone> z)
{
  var records = SelectBestRecords(q, z);
  var rname = records.At(0).Value().Name();
  var types = records.Select(r => r.Type());

  return If(
    rname == q.Name(),
    ExactMatch(records, q, z),
    If(
      IsWildcardMatch(q.Name(), rname),
      WildcardMatch(records, q, z),
      If(
        types.Any(t => t == RType.DNAME),
        Rewrite(records, q),
        If(
          And(types.Any(t => t == RType.NS),
              Not(types.Any(t => t == RType.SOA))),
          Response(Tag.R1,
            Delegation(records, z), Null<Query>()),
          Response(Tag.R2, empty, Null<Query>())
    ))));
}
```

Symbolic Execution

$\langle \text{zone file}_1, \text{query}_1 \rangle$

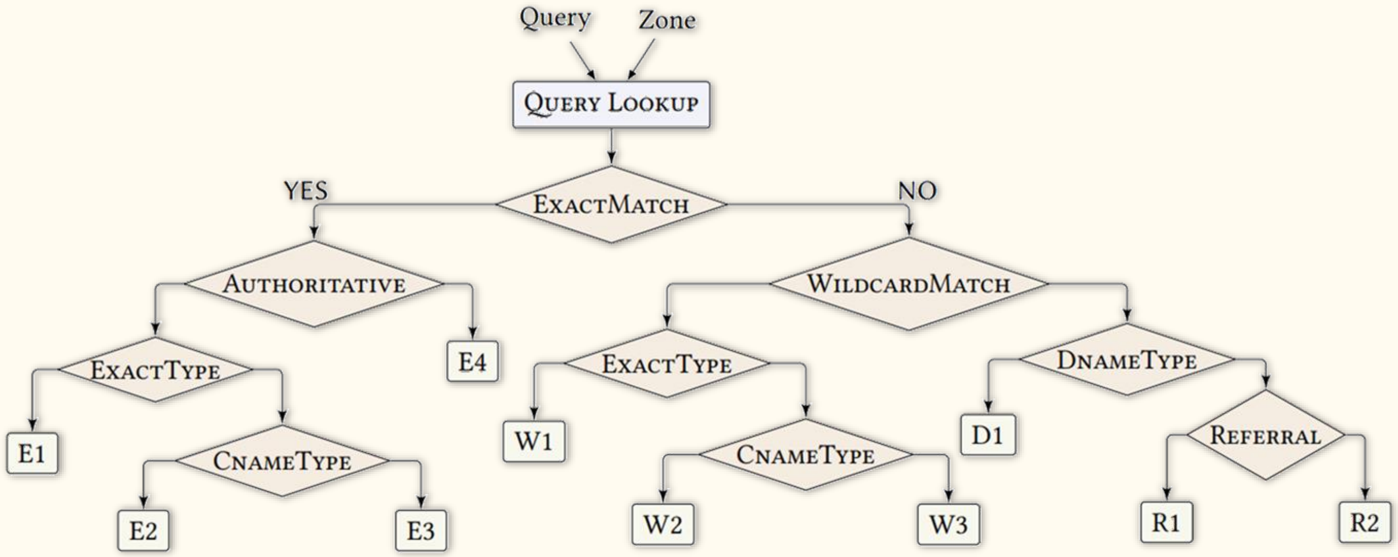
$\langle \text{zone file}_2, \text{query}_2 \rangle$

…

$\langle \text{zone file}_n, \text{query}_n \rangle$

An **executable version** of formal model is implemented in **Zen**, a domain-specific modeling language embedded in **C#** with built-in support for **symbolic execution**

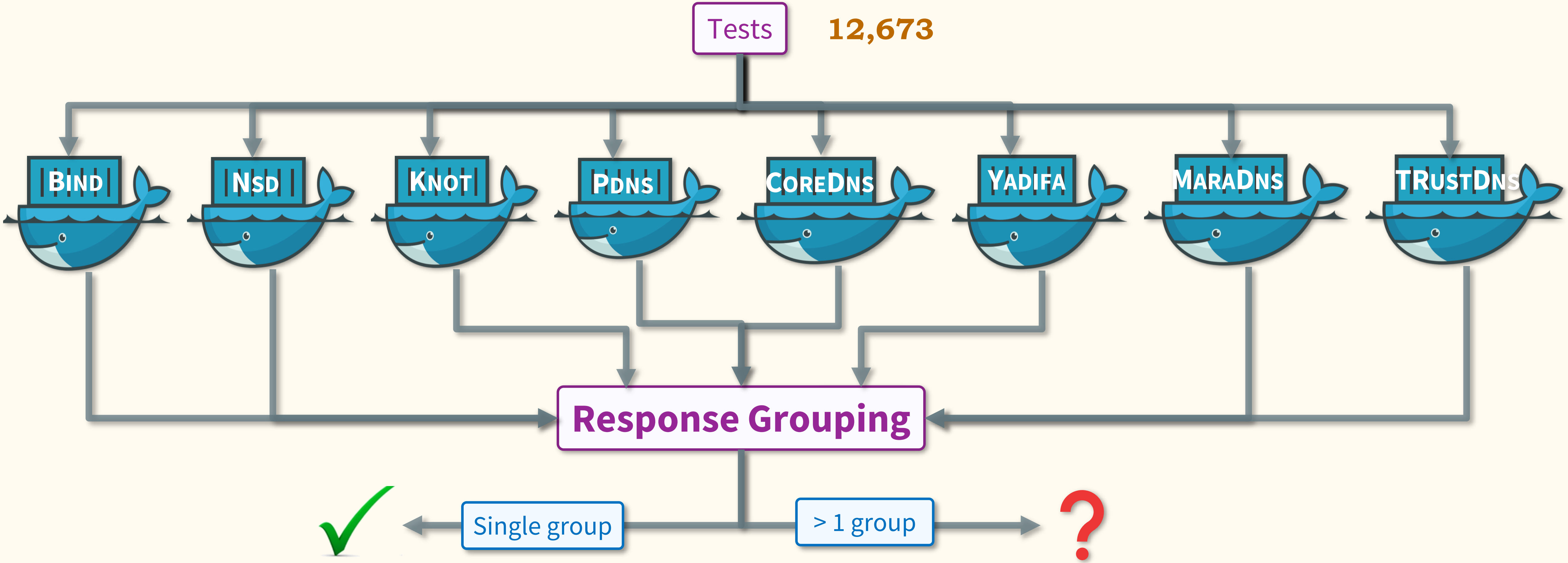
Test Generation Statistics



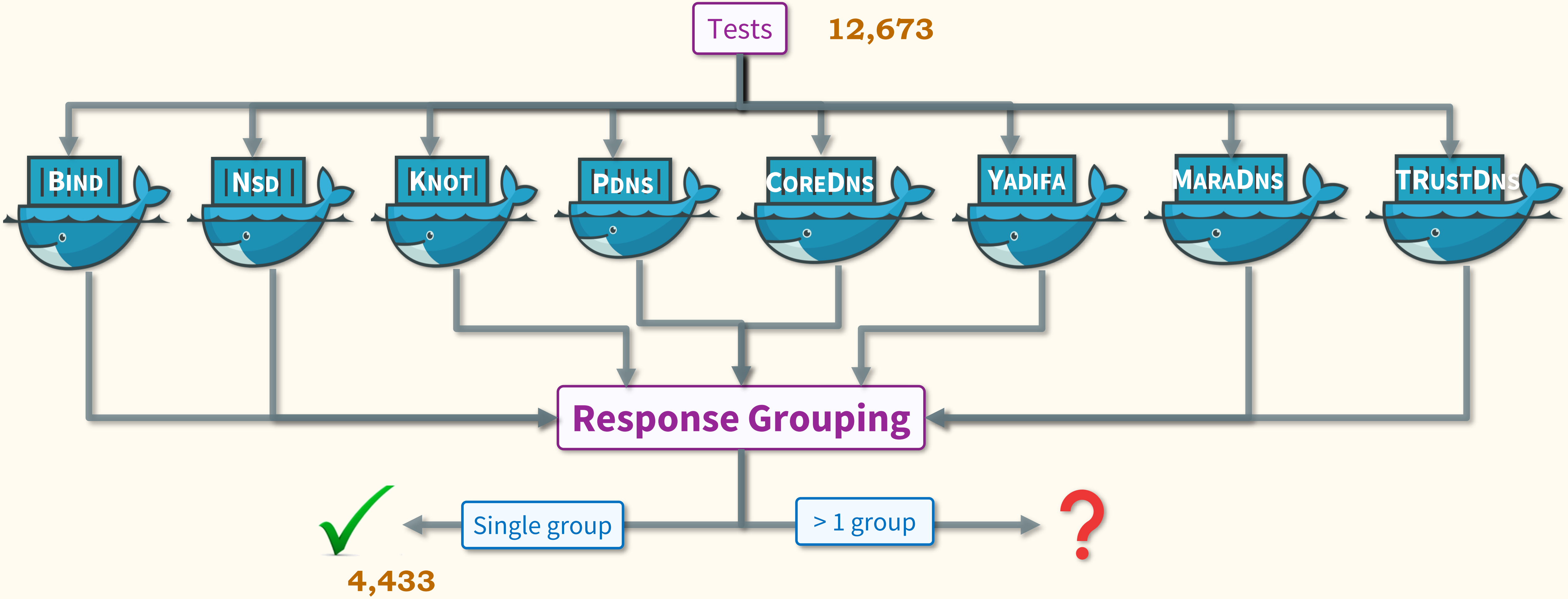
Length of each domain name and the number of records in the zone ≤ 4

Model Case	Number of Tests
E1	3180
E2	12
E3	96
E4	6036
W1	60
W2	24
W3	18
D1	230
R1	2980
R2	37
Total	12,673

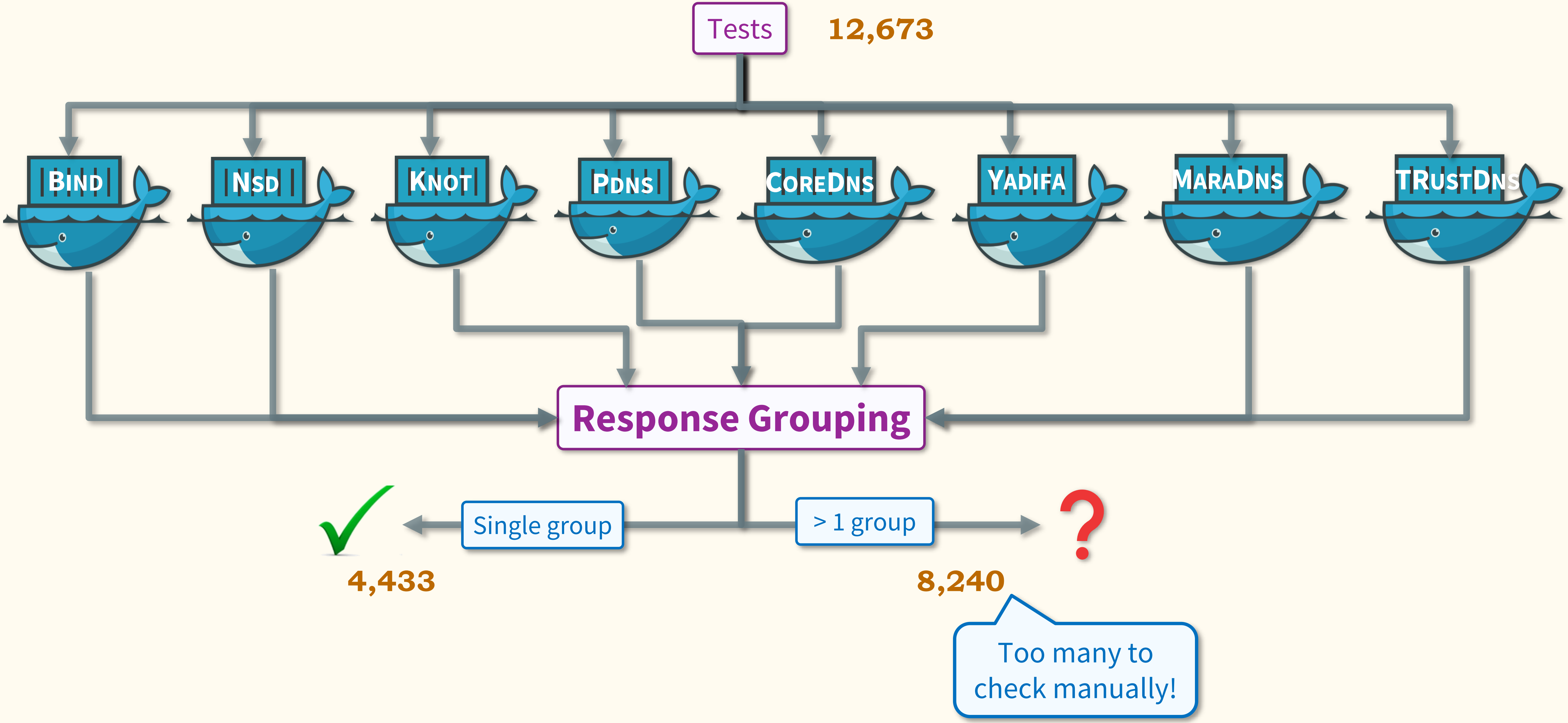
Differential Testing



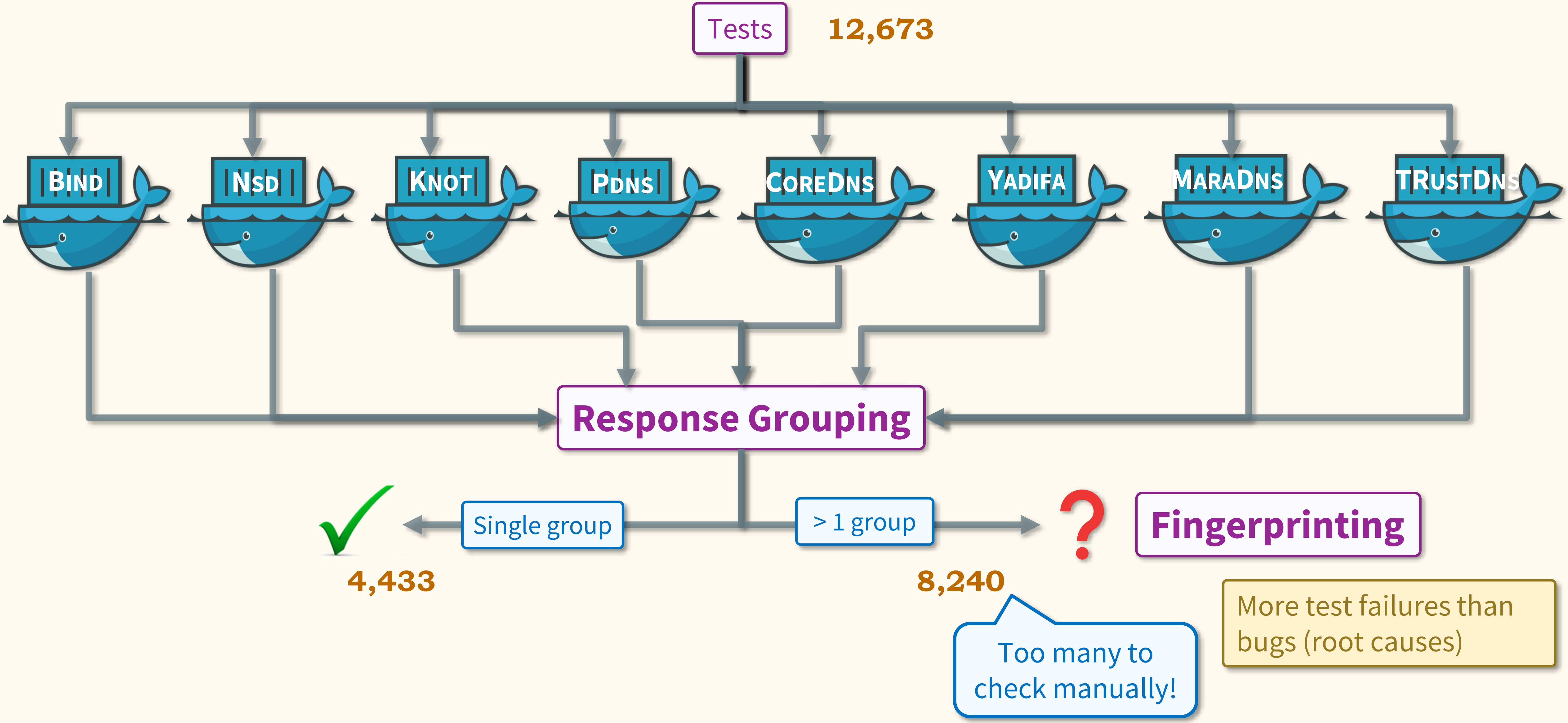
Differential Testing



Differential Testing



Differential Testing



Too many to check manually!

More test failures than bugs (root causes)

Fingerprinting

Model Case	Number of Tests	Number of Tests Failing
E1	3180	239
E2	12	10
E3	96	12
E4	6036	5312
W1	60	33
W2	24	21
W3	18	16
D1	230	65
R1	2980	2529
R2	37	3

- **Fingerprint** failed tests

Fingerprinting

Model Case	Number of Tests	Number of Tests Failing
E1	3180	239
E2	12	10
E3	96	12
E4	6036	5312
W1	60	33
W2	24	21
W3	18	16
D1	230	65
R1	2980	2529
R2	37	3

- **Fingerprint** failed tests
- Based on model case and the unique implementations in each group from the responses

Fingerprinting

Model Case	Number of Tests	Number of Tests Failing
E1	3180	239
E2	12	10
E3	96	12
E4	6036	5312
W1	60	33
W2	24	21
W3	18	16
D1	230	65
R1	2980	2529
R2	37	3

- **Fingerprint** failed tests
- Based on model case and the unique implementations in each group from the responses
- Example fingerprint – ⟨R1, {NSD, KNOT, POWERDNS, YADIFA}, {BIND, COREDNS}, {TRUSTDNS, MARADNS}⟩

Fingerprinting

Model Case	Number of Tests	Number of Tests Failing
E1	3180	239
E2	12	10
E3	96	12
E4	6036	5312
W1	60	33
W2	24	21
W3	18	16
D1	230	65
R1	2980	2529
R2	37	3

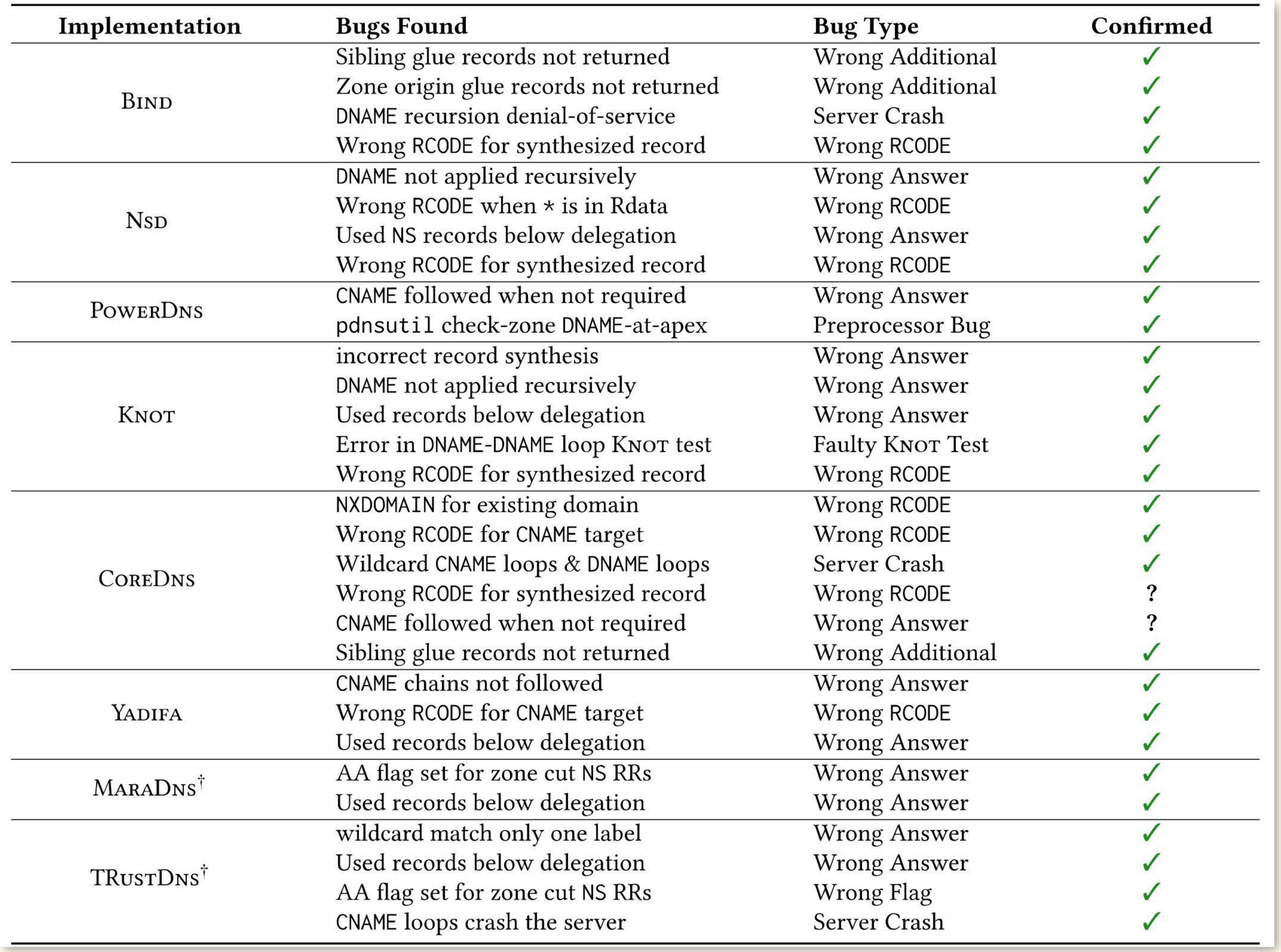
- **Fingerprint** failed tests
- Based on model case and the unique implementations in each group from the responses
- Example fingerprint – ⟨R1, {Nsd, Knot, PowerDns, Yadifa}, {Bind, CoreDns}, {TrustDns, MaraDns}⟩
- Unlikely for different unique bugs to have the same fingerprint

Fingerprinting

Model Case	Number of Tests	Number of Tests Failing	Number of Fingerprints
E1	3180	239	**7**
E2	12	10	**5**
E3	96	12	**3**
E4	6036	5312	**11**
W1	60	33	**8**
W2	24	21	**9**
W3	18	16	**1**
D1	230	65	**4**
R1	2980	2529	**27**
R2	37	3	**1**

- **Fingerprint** failed tests
- Based on model case and the unique implementations in each group from the responses
- Example fingerprint – ⟨R1, {NSD, KNOT, POWERDNS, YADIFA}, {BIND, COREDNS}, {TRUSTDNS, MARADNS}⟩
- Unlikely for different unique bugs to have the same fingerprint

Bugs Found



Implementation	Bugs Found	Bug Type	Confirmed
Bind	Sibling glue records not returned	Wrong Additional	✓
	Zone origin glue records not returned	Wrong Additional	✓
	DNAME recursion denial-of-service	Server Crash	✓
	Wrong RCODE for synthesized record	Wrong RCODE	✓
Nsd	DNAME not applied recursively	Wrong Answer	✓
	Wrong RCODE when * is in Rdata	Wrong RCODE	✓
	Used NS records below delegation	Wrong Answer	✓
	Wrong RCODE for synthesized record	Wrong RCODE	✓
PowerDns	CNAME followed when not required	Wrong Answer	✓
	pdnsutil check-zone DNAME-at-apex	Preprocessor Bug	✓
Knot	incorrect record synthesis	Wrong Answer	✓
	DNAME not applied recursively	Wrong Answer	✓
	Used records below delegation	Wrong Answer	✓
	Error in DNAME-DNAME loop Knot test	Faulty Knot Test	✓
	Wrong RCODE for synthesized record	Wrong RCODE	✓
CoreDns	NXDOMAIN for existing domain	Wrong RCODE	✓
	Wrong RCODE for CNAME target	Wrong RCODE	✓
	Wildcard CNAME loops & DNAME loops	Server Crash	✓
	Wrong RCODE for synthesized record	Wrong RCODE	?
	CNAME followed when not required	Wrong Answer	?
	Sibling glue records not returned	Wrong Additional	✓
Yadifa	CNAME chains not followed	Wrong Answer	✓
	Wrong RCODE for CNAME target	Wrong RCODE	✓
	Used records below delegation	Wrong Answer	✓
MaraDns†	AA flag set for zone cut NS RRs	Wrong Answer	✓
	Used records below delegation	Wrong Answer	✓
TRustDns†	wildcard match only one label	Wrong Answer	✓
	Used records below delegation	Wrong Answer	✓
	AA flag set for zone cut NS RRs	Wrong Flag	✓
	CNAME loops crash the server	Server Crash	✓

†Implementations with unreported issues due to missing or unimplemented features

Testing New Implementations

1. Generate a Docker image

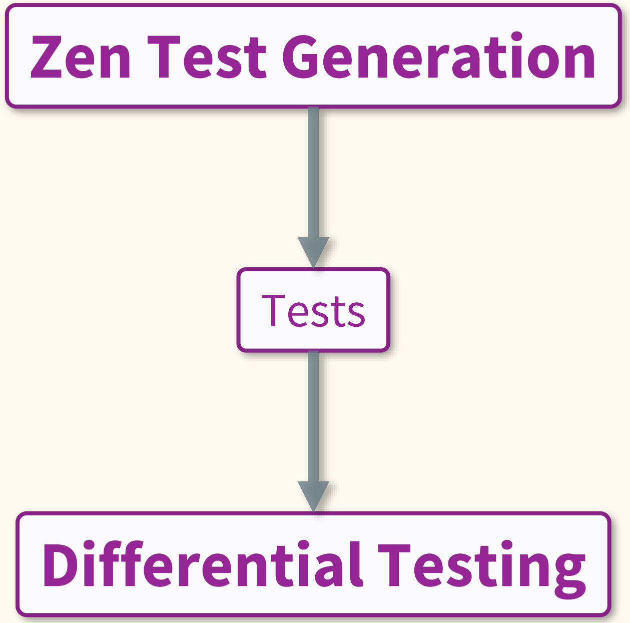
2. Start a container with a host port mapped to port 53 of the container

3. A small Script to:
   ⇢ Stop the running server in the container
   ⇢ Copy the test zone file
   ⇢ Modify the configuration (metadata)
   ⇢ Start the server

4. Pick other implementations to compare with

Custom Tests

FERRET



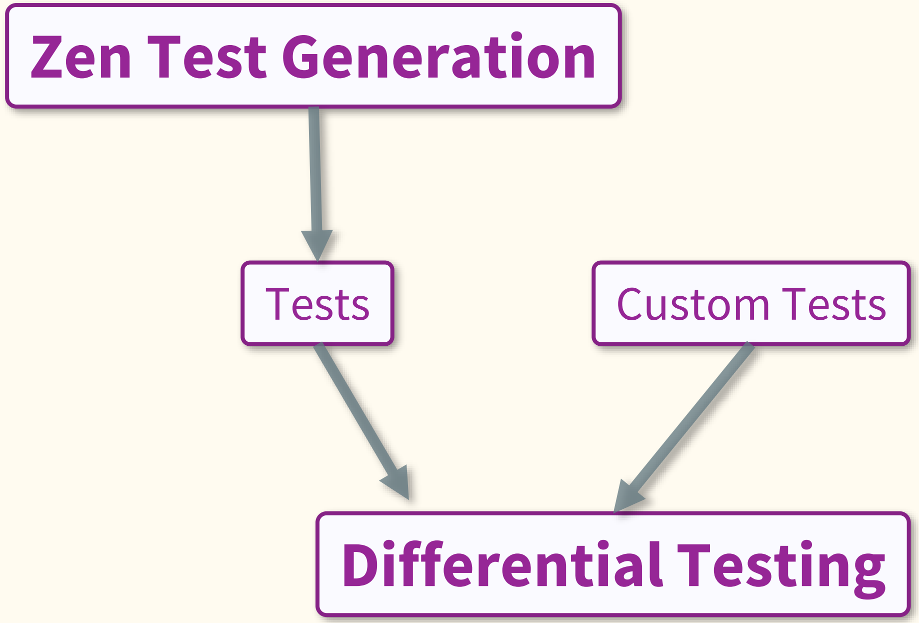
Zen Test Generation

Tests

Differential Testing

Custom Tests

FERRET



Zen Test Generation

Tests

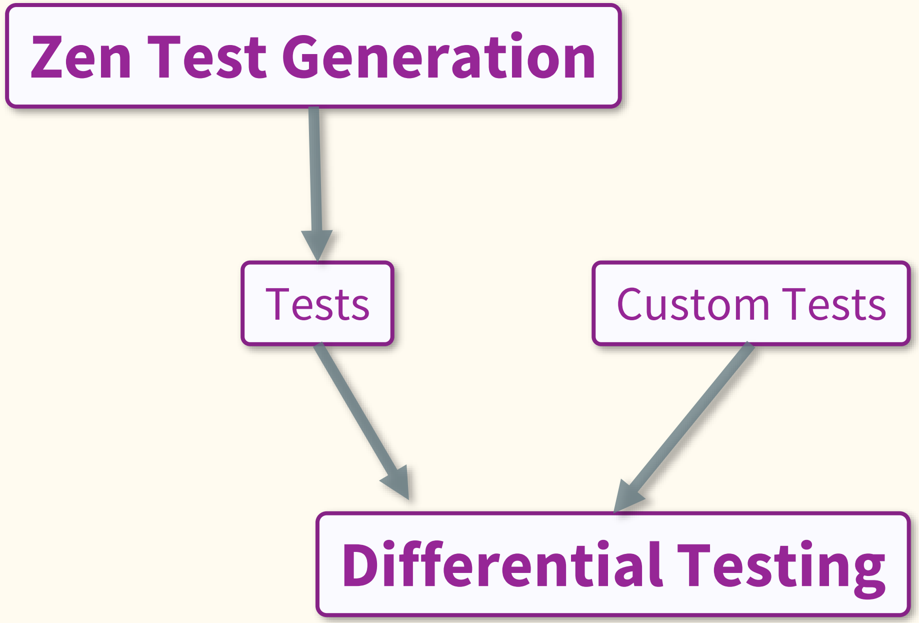
Custom Tests

Differential Testing

Organization Zone Files

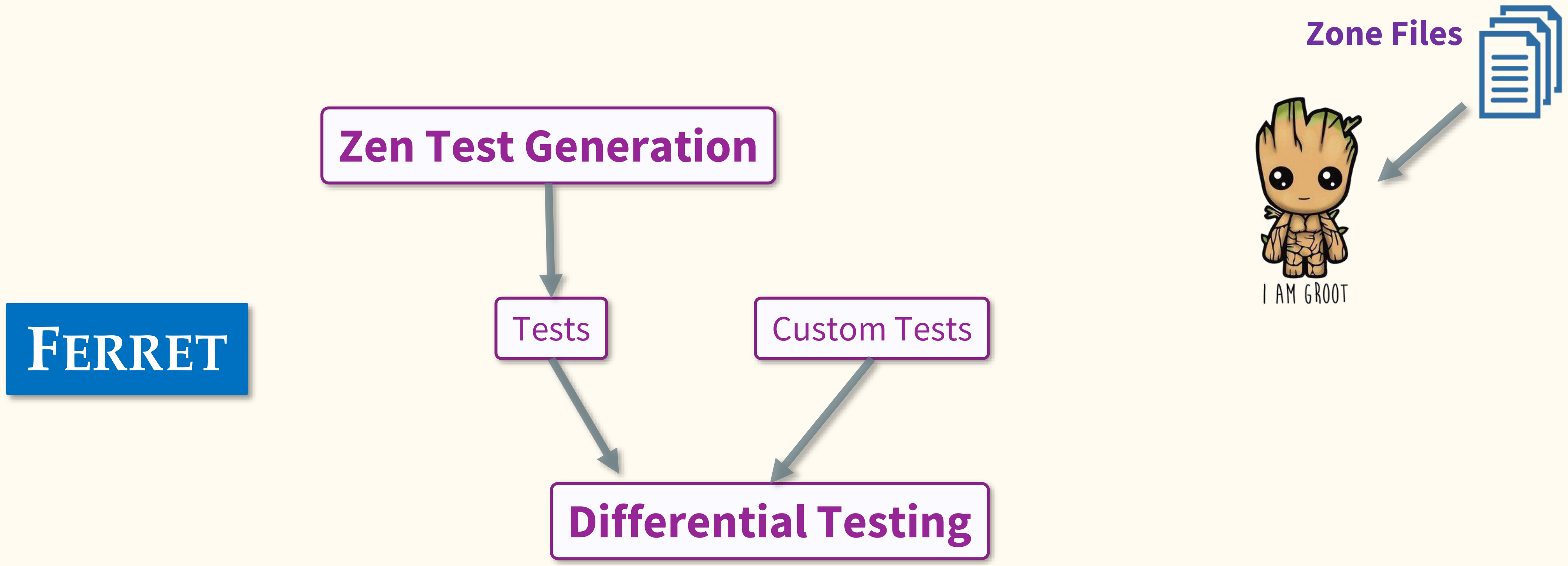
Zone Files

FERRET



How do we test for any implementation-specific behaviors on *our zone files?*

Organization Zone Files



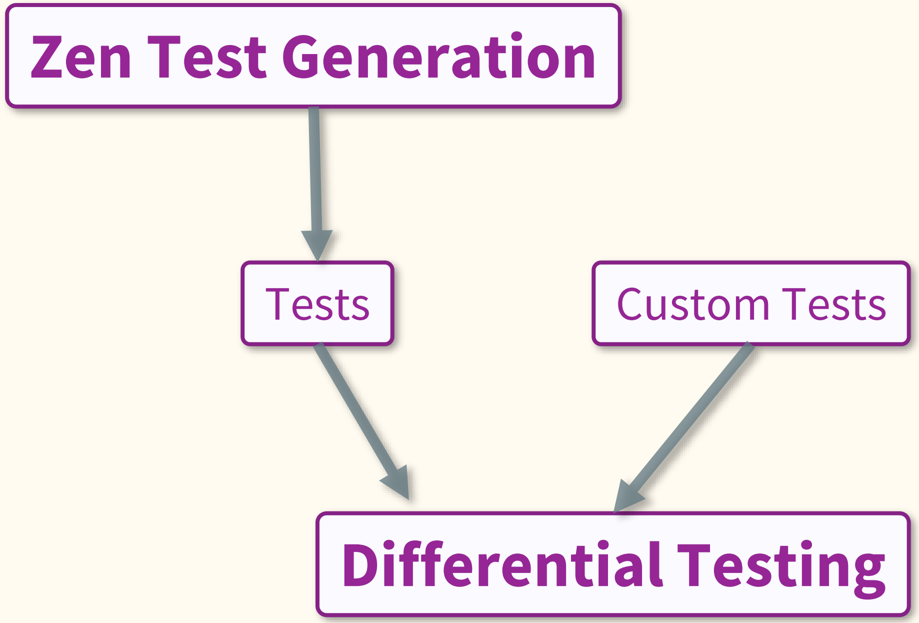
How do we test for any implementation-specific behaviors on *our zone files?*

Use GROOT to generate query *equivalence classes*

(see our paper/tool for details)

Organization Zone Files

FERRET

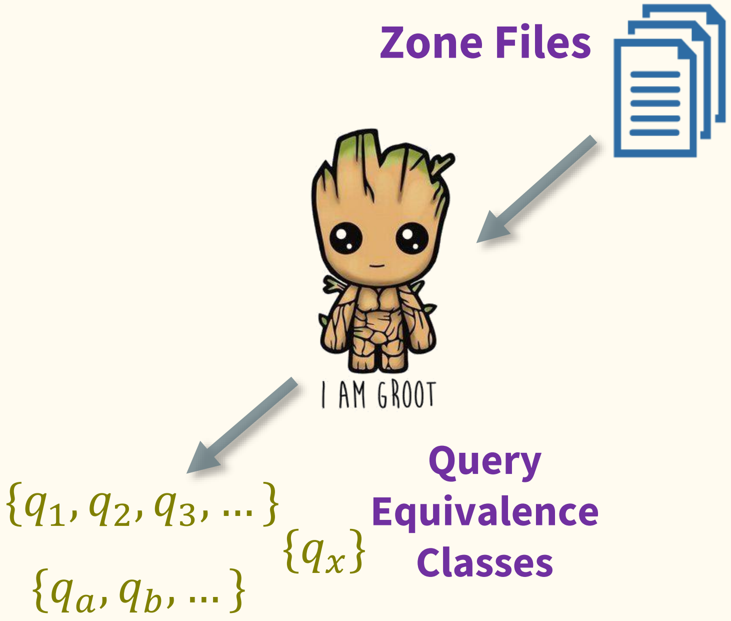


Zen Test Generation

Tests

Custom Tests

Differential Testing



Zone Files

$\{q_1, q_2, q_3, \ldots\}$

$\{q_x\}$

$\{q_a, q_b, \ldots\}$

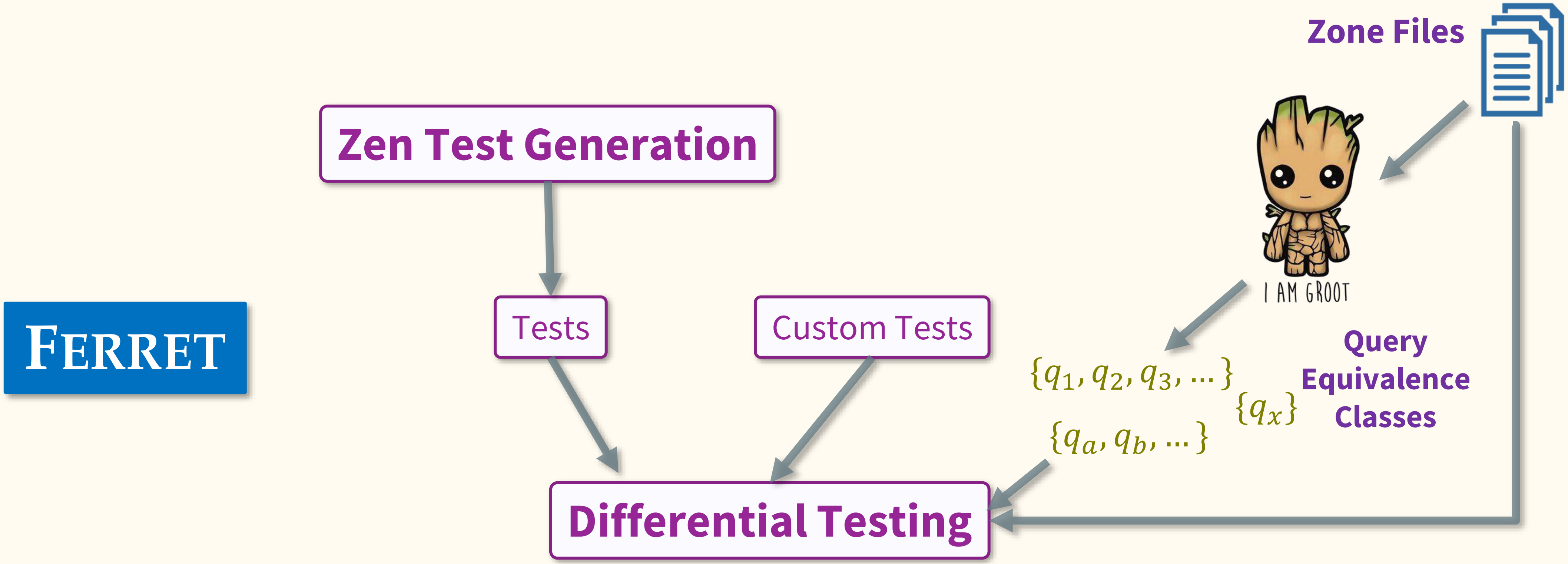
Query Equivalence Classes

How do we test for any implementation-specific behaviors on *our zone files?*

Use GROOT to generate query *equivalence classes*

(see our paper/tool for details)

Organization Zone Files



How do we test for any implementation-specific behaviors on *our zone files?*

Use GROOT to generate query *equivalence classes*

(see our paper/tool for details)

No Two Nameservers Agree!

- Nobody agrees with RFCs too!

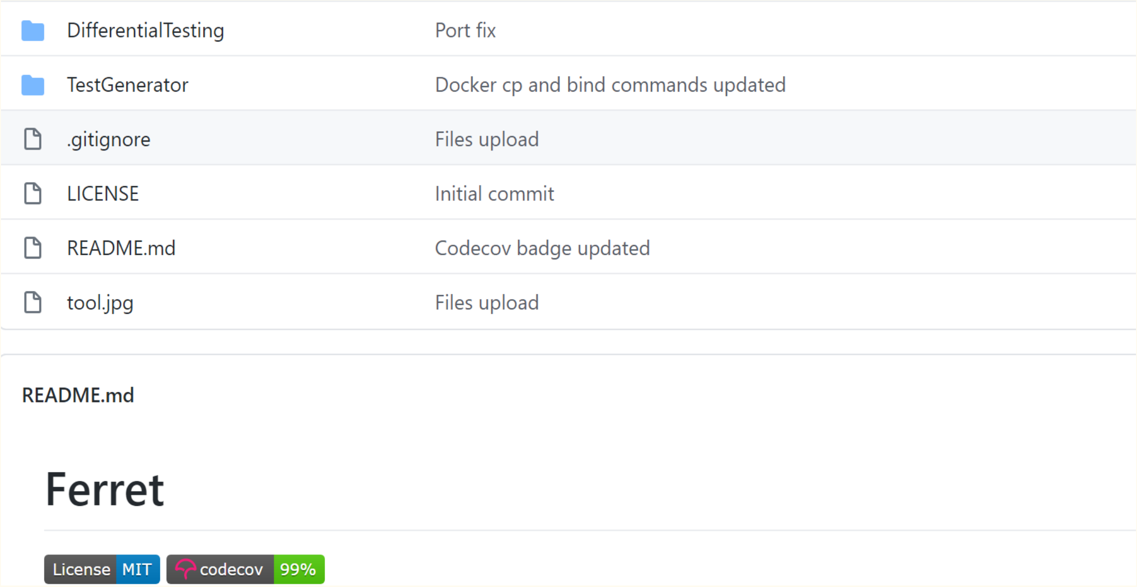
- RFCs do the job well but there are gaps and ambiguities
  - ↳ CNAME loops should be signaled as errors (RFC 1034)
    - At what point?
    - Should it be unrolled at all?
    - Should the loop RRs be returned?
  - ↳ Is a synthesized CNAME from DNAME perfect response to a CNAME query?

- When RFCs are open to interpretation, implementations make choices based on – performance, resource constraints, safety, …

- Should resolvers account for different choices? (complex resolvers, interoperability issues)
  Or
  Should the RFCs be more verbose and stringent?

Conclusion

- FERRET – Our tool for automatic test generator for nameserver implementations
- Generates high-coverage test suites stress testing many corner cases of RFCs
- Differential testing to compare multiple implementations
- Tested 8 implementations
- Found 30 new bugs
- https://github.com/dns-groot/Ferret, https://github.com/dns-groot/groot
- Reach me at: sivakesava@cs.ucla.edu



DifferentialTesting	Port fix
TestGenerator	Docker cp and bind commands updated
.gitignore	Files upload
LICENSE	Initial commit
README.md	Codecov badge updated
tool.jpg	Files upload

README.md

Ferret