NextHop

Layer 7 Routing

COMCAST John Rushford

Vijay Mamidi

Oath: Aaron Canary

Agenda

HostStatus	in master
XDebug probe	testing
NextHop Design	wiki soon
Shared Data Storage	experimental soon
NH HealthCheck	experimental soon

Host Status

- Hosts managed by ATS can be origin servers and parents
- Hosts are configured in remap.config and parent.config
- remap.config hosts status is maintained in HostDB
- parent.config hosts status is maintained in Parent Structures (in memory only)
- Status of the Hosts is currently managed Passively.
  - Passive: Did the host respond to queries. (variables configurable)
- Config files are edited to manage hosts.

Active Liveness Check

- Passive Host Status : Requests are involved to determine the status.
    - Latency problems.
    - Timeouts.
    - Maintenance of a Hosts need config file edits.
    - Harder to figure out Parent Host is down or the OS that the Parent Host talking to is down
        - Per Parent Host Status
            - Works Well when OS is down -- isolates specific origin problem.
            - Can’t reuse the parent liveness information.
            - The more configs, the more sacrificial requests to keep parent down.
- Manual Host Status : Take this host out of rotation.
    - Solves editing config files.
    - Does not solve other problems.

Active Liveness Check

- Active Host Status: Is this host responding?
  - Solves the Latency problems and Timeouts.
  - Can distinguish between Parent/OS problems.
  - But.. Increase in Network Traffic and requests/second
  - What if the Host goes down in between the checks?
- Active Distributed Host Status: External process performs health check and notifies ATS
  - Solves the Latency problems and Timeouts
  - Can distinguish between Parent/OS problems.
  - Reduces Network Traffic.
  - Does not solve the problem of host being down in between the health checks
  - An External process communication to a Host doesn’t necessarily mean that ATS can talk to that particular host.

Solution

- No Host single liveness checking strategy solve all the use cases.
- Why don’t we use all with some hierarchy ?
- Highest Priority: Manual.
  - No more config file edits.
- Followed by Distributed Host Status.
  - Reduces network traffic.
  - Solves the Latency problems and Timeouts.
  - Can distinguish between Parent/OS problems.
- Followed by Local Host Status.
  - Solves the problem when there are network connectivity issues between ATS and Host
- Followed by passive
  - Solves the problem when the host is done in between the distributed/local health checks.

Completed…

- Parent hosts defined in parent.config
- Parents hosts can be manually marked up/down using traffic_ctl(new)
- Parents hosts can be manually marked up/down using API (new)
- Manual/Plugin based checks are considered before a parent is chosen(new)
- The http state machine marks down a parent due to connection errors or timeouts (passive down). A parent will be marked for retry once the retry window has elapsed. The parent is marked up if a retry is successful.(existing)

traffic_ctl enhancement (PR #3302)

- Using traffic_ctl mark down parents(s) globally.
- Only used with parents listed in parent.config.
- Future use with next hop to mark any origin or parent down.

Example:

```
# traffic_ctl host status  parent-cache-01.kabletown.net
host_status.parent-cache-01.kabletown.net 0
# traffic_ctl host up parent-cache-01.kabletown.net
# traffic_ctl host status  parent-cache-01.kabletown.net
host_status.parent-cache-01.kabletown.net 1
# traffic_ctl host down parent-cache-01.kabletown.net
```

Global status available as stats

- Host status is available in metrics and from the stats_over_http endpoint.

```
# curl http://192.168.1.66:8080/_stats
…
"proxy.process.traffic_server.memory.rss": "365764608",
"host_status.parent-cache-01.kabletown.net": "1",
"host_status.parent-cache-02.kabletown.net": "1",
"host_status.parent-cache-03.kabletown.net": "1",
"host_status.parent-cache-04.kabletown.net": "1",
```

Example traffic_ctl metric subcommand

```
# /opt/trafficserver/bin/traffic_ctl metric match host_status
host_status.parent-cache-01.kabletown.net 1
host_status.parent-cache-02.kabletown.net 1
host_status.parent-cache-03.kabletown.net 1
host_status.parent-cache-04.kabletown.net 1
host_status.parent-cache-05.kabletown.net 1
```

Use with management tools

- Use traffic_ctl to manage the state of parents on a trafficserver proxy host.
- Use ‘stats_over_http’ to monitor the current state of parents on a trafficsever proxy host.
- Incorporate ‘traffic_ctl host’ into management and monitoring tools:
  - OpenNMS
  - Nagios
  - Puppet
  - Pdsh scripts.

-H “X-Debug: probe”

NOT SECURE!

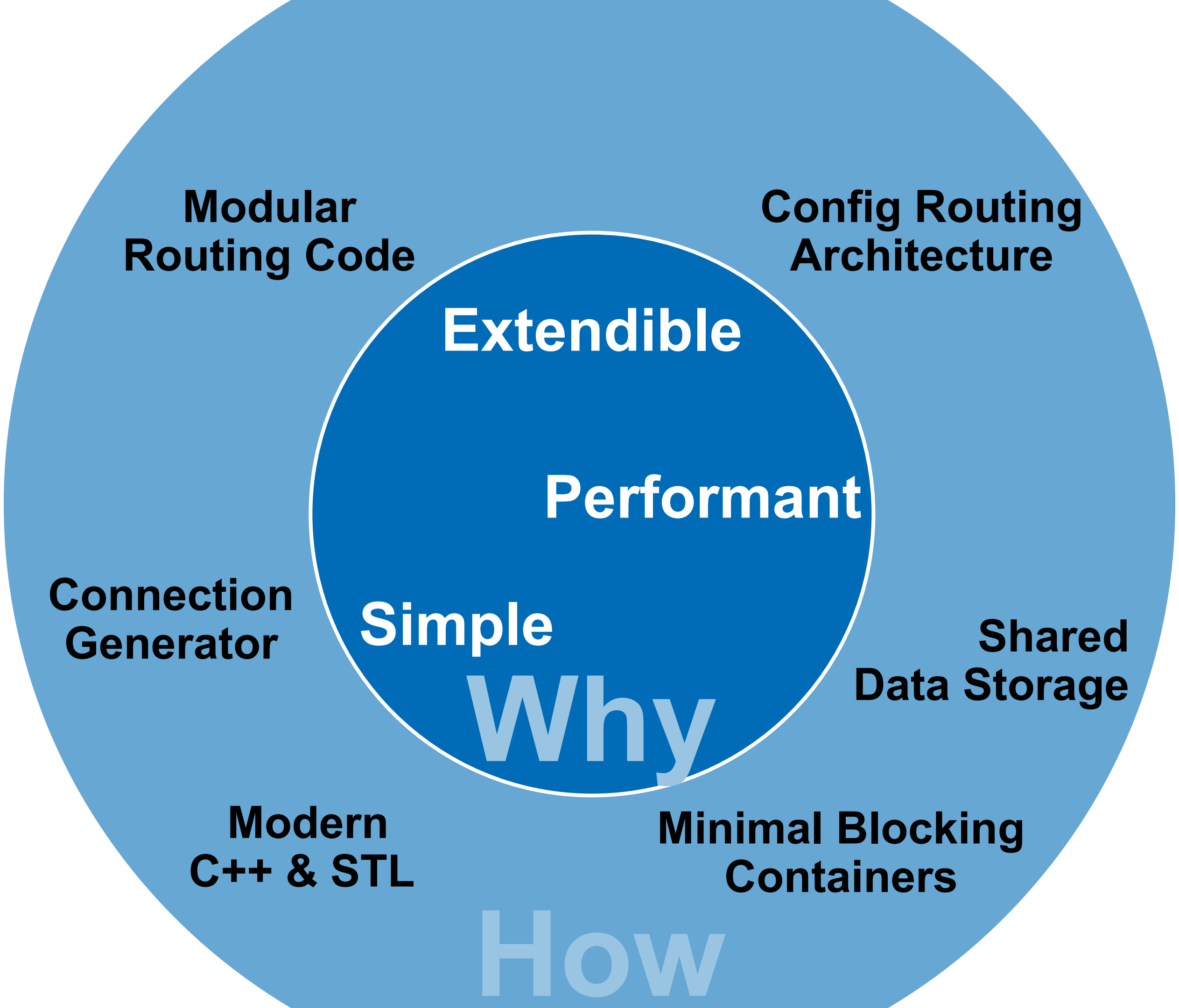
NextHop Design

Extendible

Performant

Simple

Why



Modular Routing Code

Config Routing Architecture

Extendible

Performant

Simple

Connection Generator

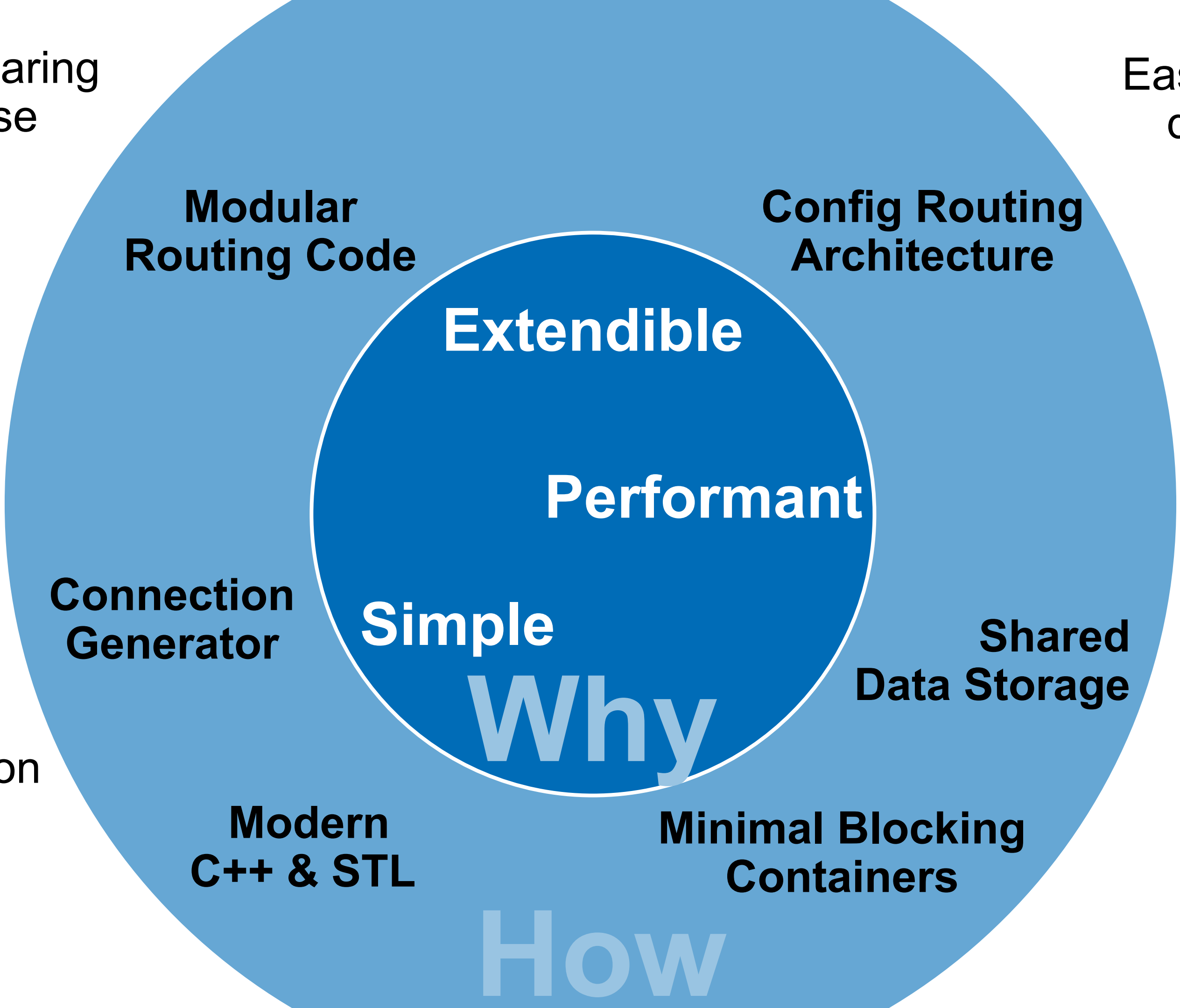
Shared Data Storage

Why

Modern C++ & STL

Minimal Blocking Containers

How



More sharing and reuse

Easy test and deploy new routing

Modular Routing Code

Config Routing Architecture

Extendible

Reduce HttpSM

Performant

Meets variety of needs

Connection Generator

Simple

Shared Data Storage

Why

Comprehension

Modern C++ & STL

Minimal Blocking Containers

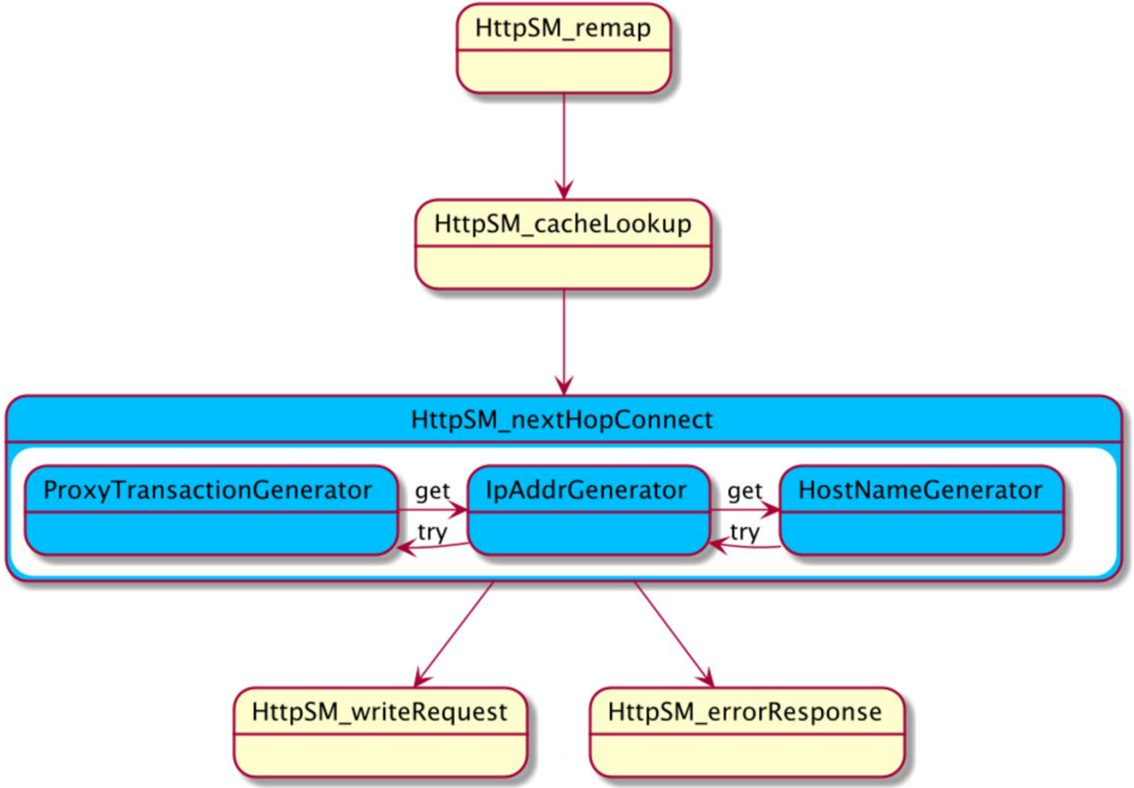
Easy to write fast code

How

What

Define the selection behavior of *transaction proxy connection* with modular plugins.

AKA: Robustly find a valid upstream.



Design Update:

Internalizing Session Manager to allow more flexibility to the resolver script.
NextHop IP Generator -> Proxy Transaction Generator

CDN Config Example

```
cdn_map = {
"pod_a": {parents: [],               "seed": 13, "vip": vip_a, "hosts": [a1,a2,a3,a4,...]},
"pod_b": {parents: ["pod_a"], "seed": 17, "vip": vip_b, "hosts": [b1,b2,b3,b4,...]},
"pod_c": {parents: ["pod_a"], "seed": 23, "vip": vip_c, "hosts": [c1,c2,c3,c4,...]},
… }
selfPod = findSelfPod(cdn_map)
peers = cdn_map[selfPod]["hosts"]
parents = listParentHosts(cdn_map, selfPod)
```

not origin

Resolver Config Example

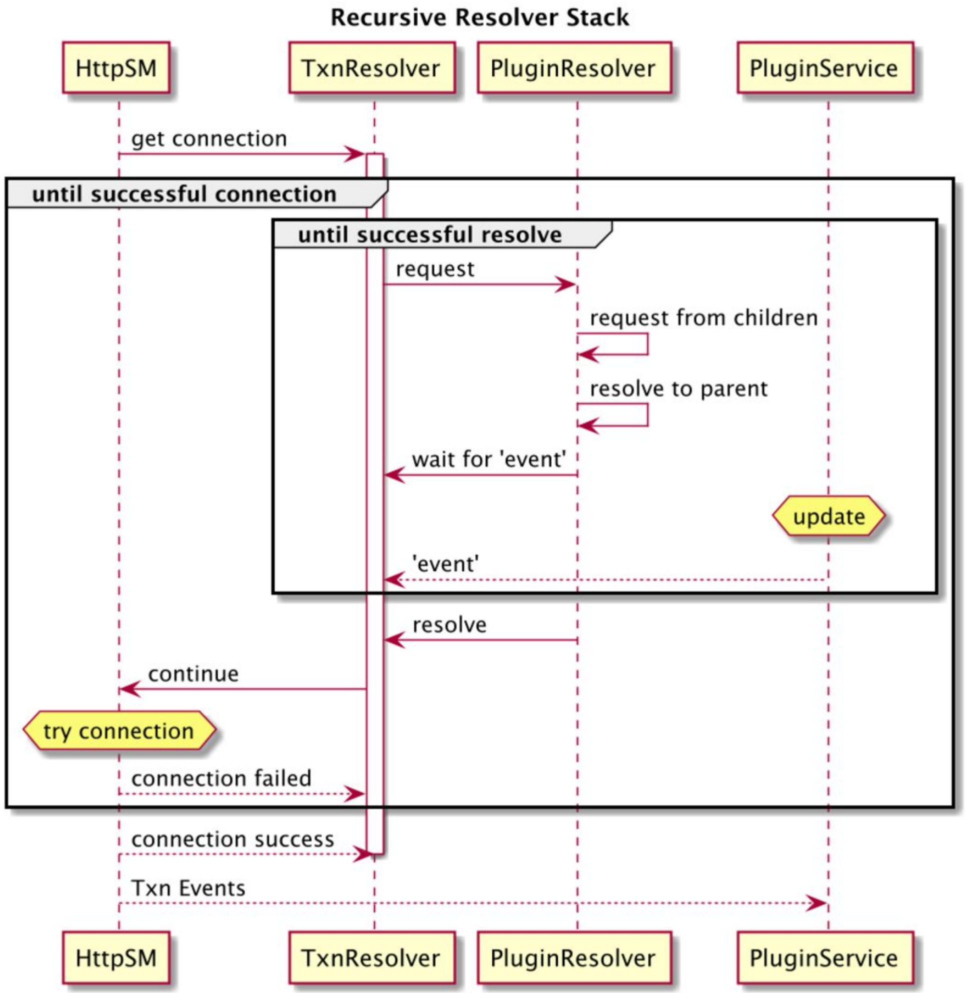
```
request = Request()

hosts = First(2, CHash(peers, request)) + CHash(parents, request) + request

hosts = First(2, CHash(peers, request)) + CHash(parents, request) + request

ips = EtcHost(ok_hosts) + CurrentIp(ok_hosts) + DNSCache(ok_hosts) + DNS(ok_hosts)

ok_ips = IpStatus(HealthCheck(ips))

pxtxn_stream = SessionMgr(ok_ips)

Resolve(pxtxn_stream, “tumblr”)
```

Remap Config Example

```
map https://static.tumblr.com        https://sc.yimg.com     @resolver=“tumblr”
```

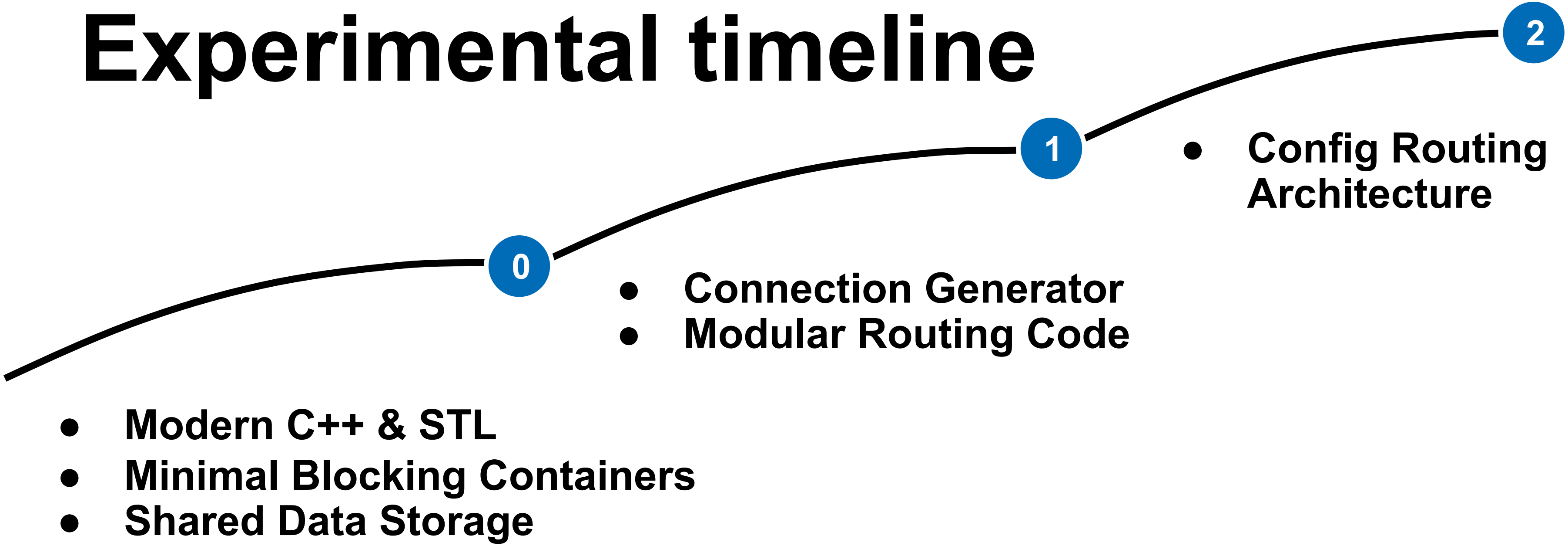
- Only used to map origins. No longer have to remap to next layer of cache.
- Remap configs at CDN layers will likely converge.

```
map https://static.tumblr.com        https://static.tumblr.com     @resolver=“tumblr”
```


Recursive Resolver Stack
HttpSM
TxnResolver
PluginResolver
PluginService
get connection
until successful connection
until successful resolve
request
request from children
resolve to parent
wait for 'event'
update
'event'
resolve
continue
try connection
connection failed
connection success
Txn Events
HttpSM
TxnResolver
PluginResolver
PluginService

Discuss:
Async Event System

Experimental timeline



- Modern C++ & STL
- Minimal Blocking Containers
- Shared Data Storage

- Connection Generator
- Modular Routing Code

- Config Routing Architecture

Phase 0:

Shared Data Storage

of NextHop

Currently host state is stored by system.

HostDB
HttpConnectionCount
HostStatus
CARP/Host
ParentHost
HealthCheckPlugin

Each new system requires

- new storage container
- reimplement thread safety
- indexing and hashing
- performance optimization

Scale with lower overhead.

:Extendible & Modular Routing Code

HostDB
HttpConnectionCount
HostStatus
CARP/Host
ParentHost
HealthCheckPlugin

store

HostSharedData
AddrSharedData

Each new system requires

- Alloc fields in sharedData container

Data storage API

:Simple & Easy to write fast code

HostDB

```
auto fld_dns_record = HostSharedData.schema.addField<COPYSWAP, vector<IpAddr>>(“dns_record”);
auto fld_dns_ttl = AddrSharedData.schema.addField<ATOMIC, uint32_t>(“dns_ttl”);
...
auto host_data = HostSharedData.find(hostname);
if (host_data) {
        auto addrs = host_data>get(fld_dns_record);
        auto ip_ttl = AddrSharedData.find(addrs[0])->get(fld_dns_ttl);
}
```

Type Safety

Data storage API

:Simple & Minimal Blocking Containers

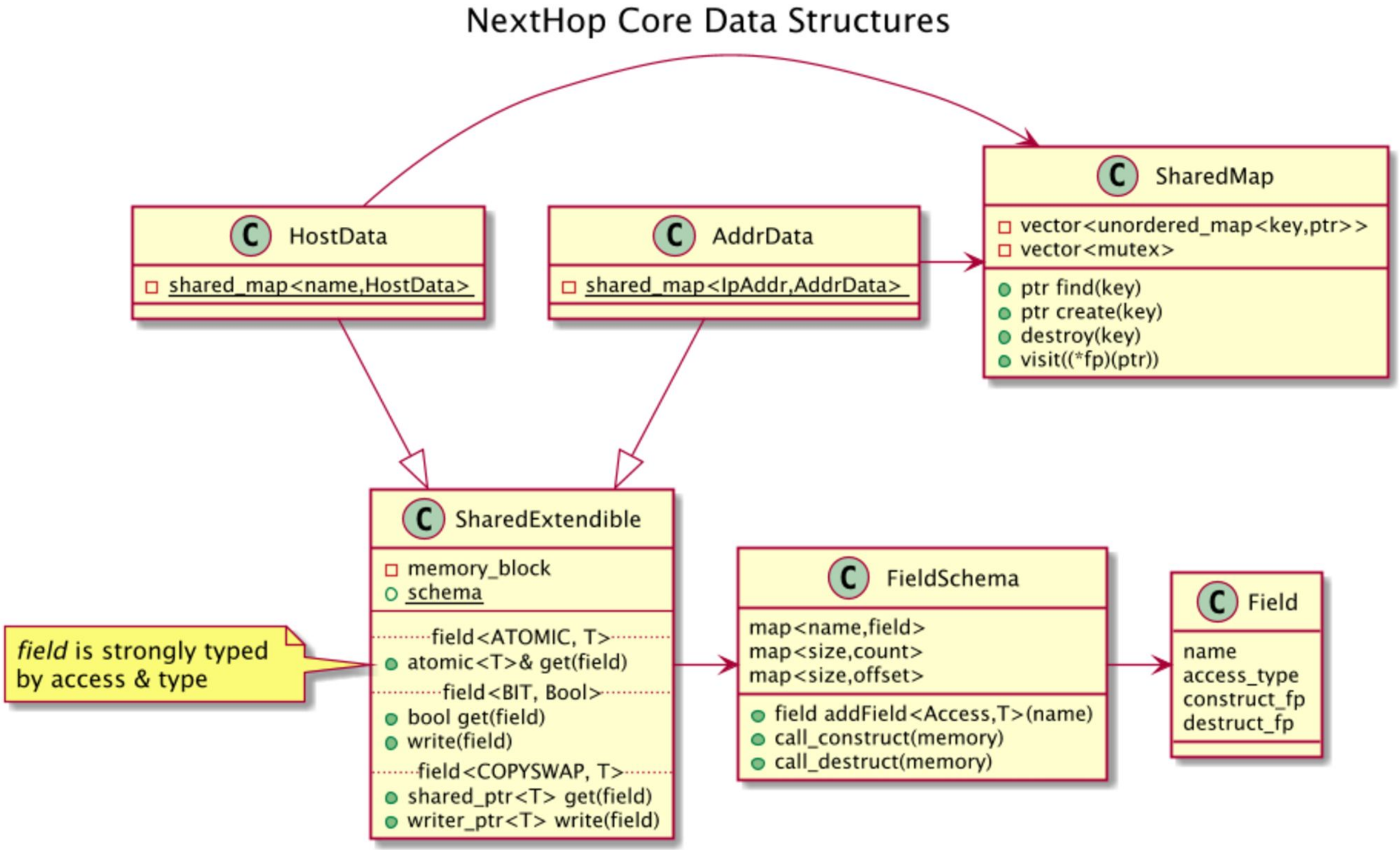
HostStatus

```
auto fld_host_oor = HostSharedData.schema.addField<BIT, bool>(“host_OOR”);
auto fld_addr_oor = AddrSharedData.schema.addField<BIT, bool>(“addr_OOR”);
...
auto host_data = HostSharedData.find(hostname);
if (host_data) {
        if (host_data->get(fld_host_oor) {
…
auto addr_data = AddrSharedData.find(addr);
if (addr_data) {
        if (addr_data->get(fld_addr_oor) {
```

lock, cp shared_ptr, unlock

lock free atomic

Data Design: Extendible Simple Performant



Code Review Invite:

(Can we make a Nexthop Branch?)

- Aligning atomics in heap
- writer_ptr w/ COPYSWAP

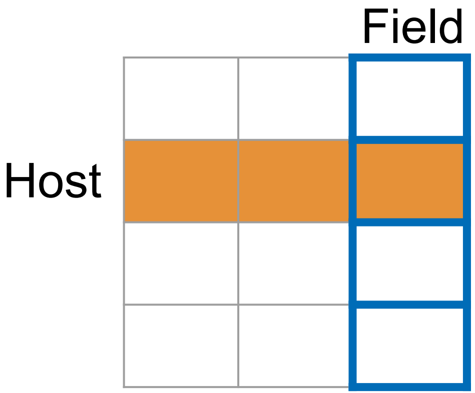
Data Design: Extendible Simple Performant

Less continuations.

More concise code.

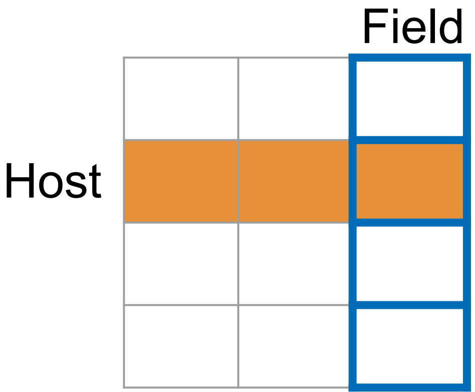
Columns vs Rows

Optimizing CPU cache usage



Data Design: Extendible Simple Performant

CPUs will more reliably precache data when it is stored for contiguous reads.



OperateAllHosts(Field)	stored by **Field**	✔ **Optimal CPU cache usage**
OperateAllFields(Host)	stored by **Field**	× CPU Cannot preload cache
OperateAllHosts(Field)	stored by **Host**	× CPU Cannot preload cache
OperateAllFields(Host)	stored by **Host**	✔ **Optimal CPU cache usage**

Improve CPU cache usage. Expect performance wins.

HostDB
HttpConnectionCount
HostStatus
CARP/Host
ParentHost
HealthCheckPlugin

→

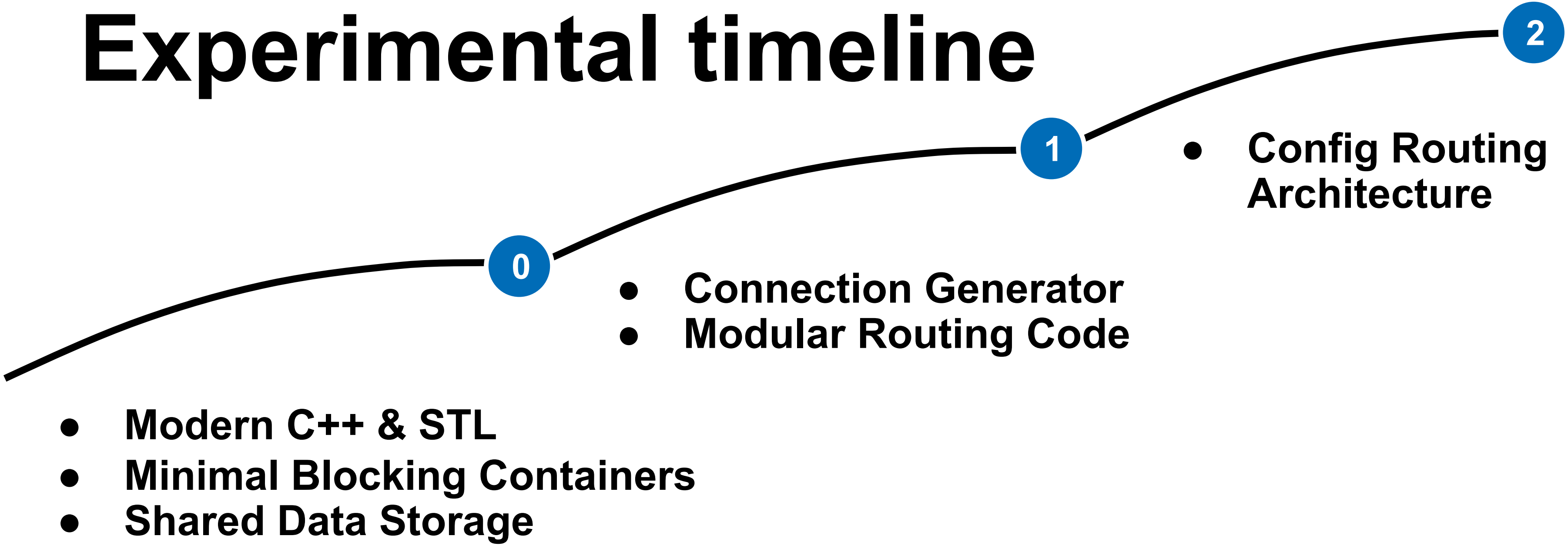
HostSharedData
AddrSharedData

Shared Data storage:

- reduce overhead work
- more concise code
- improve CPU cache efficiency

Discuss:
Shared Storage
Opportunities & Risks

Experimental timeline



- Modern C++ & STL
- Minimal Blocking Containers
- Shared Data Storage

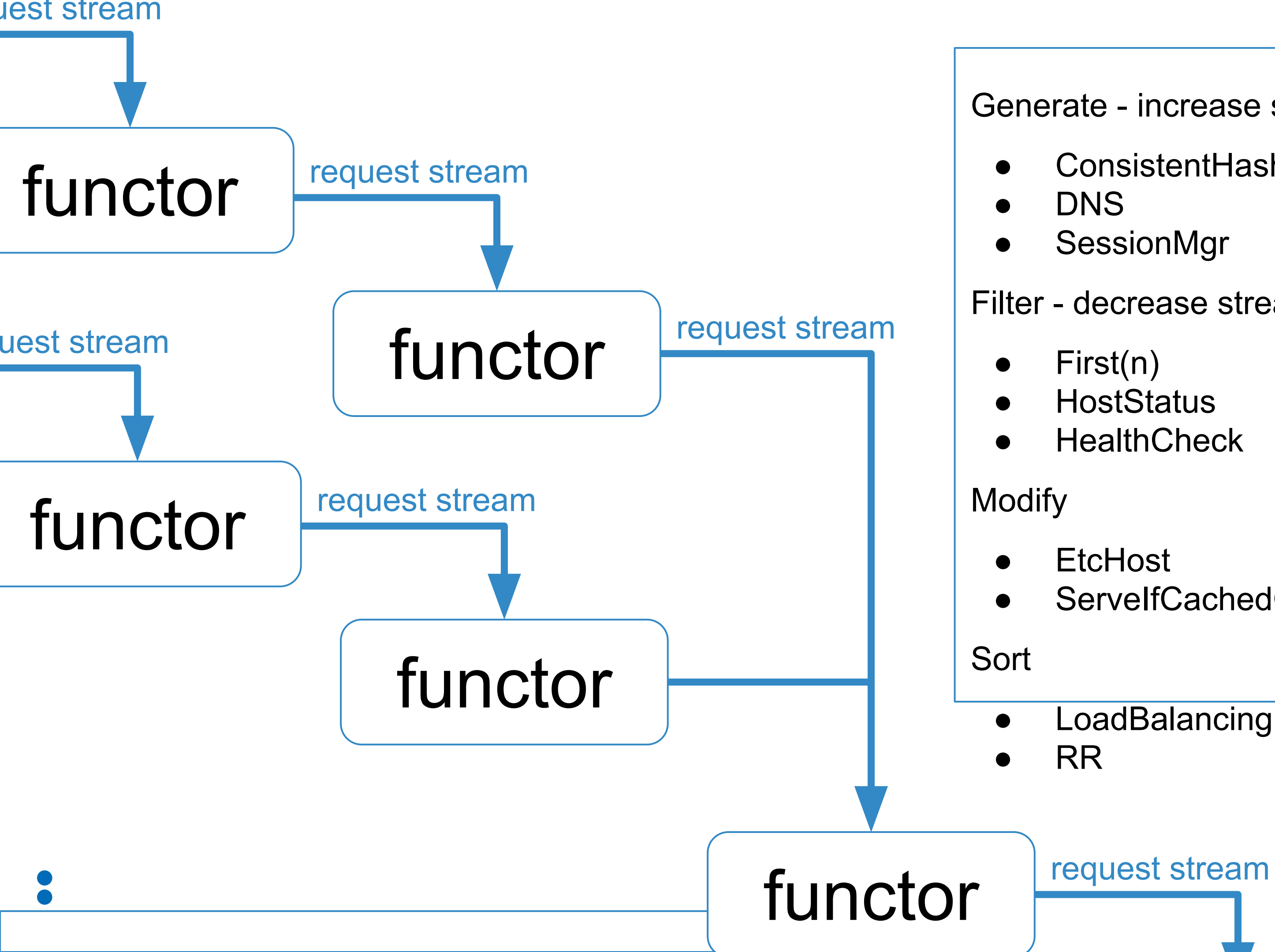
- Connection Generator
- Modular Routing Code

- Config Routing Architecture

Phase 1:

Modular Routing Code

```
request = Request()

hosts = First(2, CHash(peers, request)) + CHash(parents, requ

ok_hosts = HostStatus(hosts)

ips = EtcHost(ok_hosts) + CurrentIp(ok_hosts) + DNSCache(o

DNS(ok_hosts)

ok_ips = IpStatus(HealthCheck(ips))

pxtxn_stream = SessionMgr(ok_ips)

Resolve(pxtxn_stream, “tumblr”)
```



uest stream

functor

request stream

functor

request stream

uest stream

functor

request stream

functor

functor

request stream

Generate - increase stream length

- ConsistentHash
- DNS
- SessionMgr

Filter - decrease stream length

- First(n)
- HostStatus
- HealthCheck

Modify

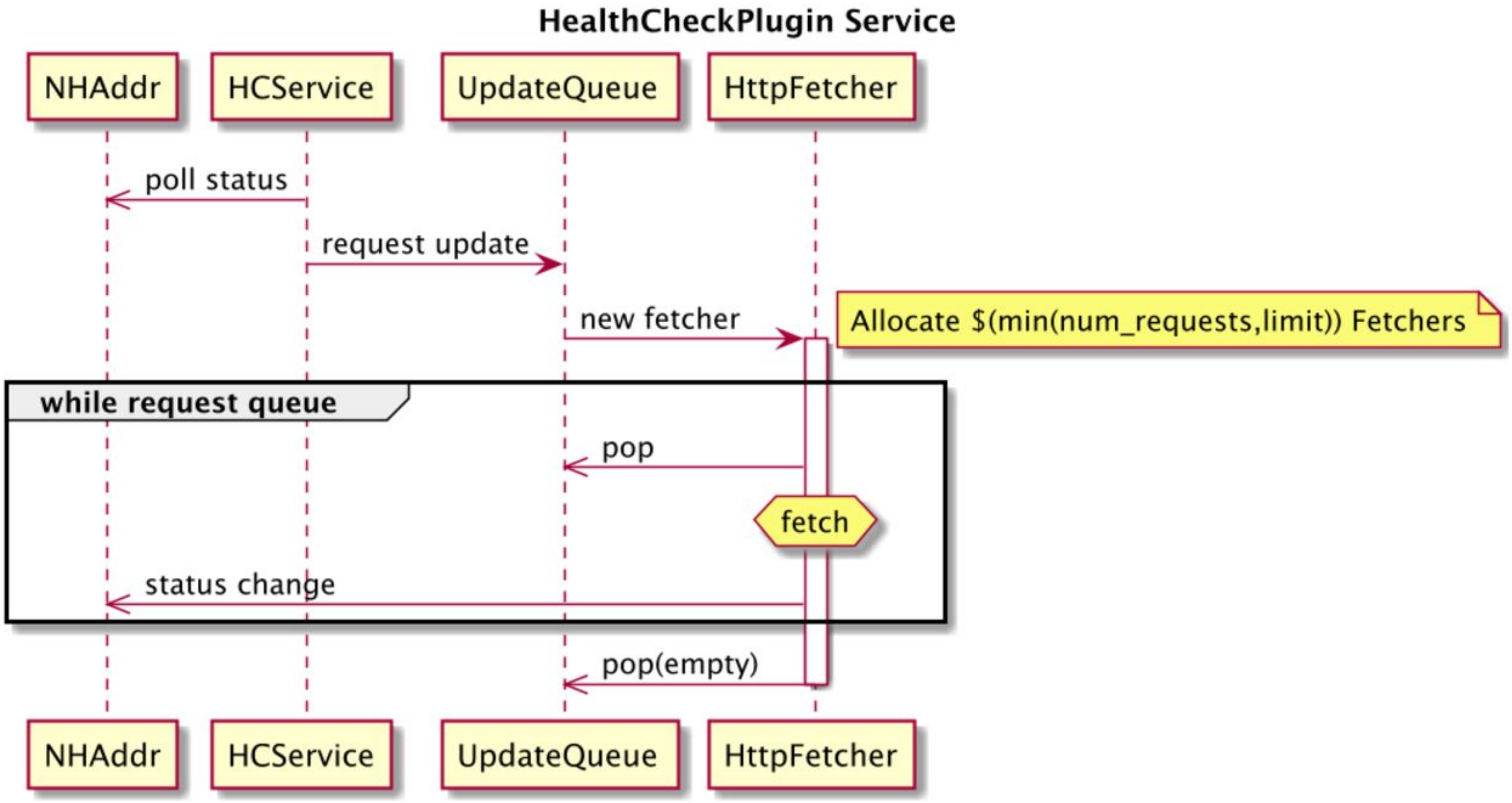
- EtcHost
- ServeIfCachedOnly

Sort

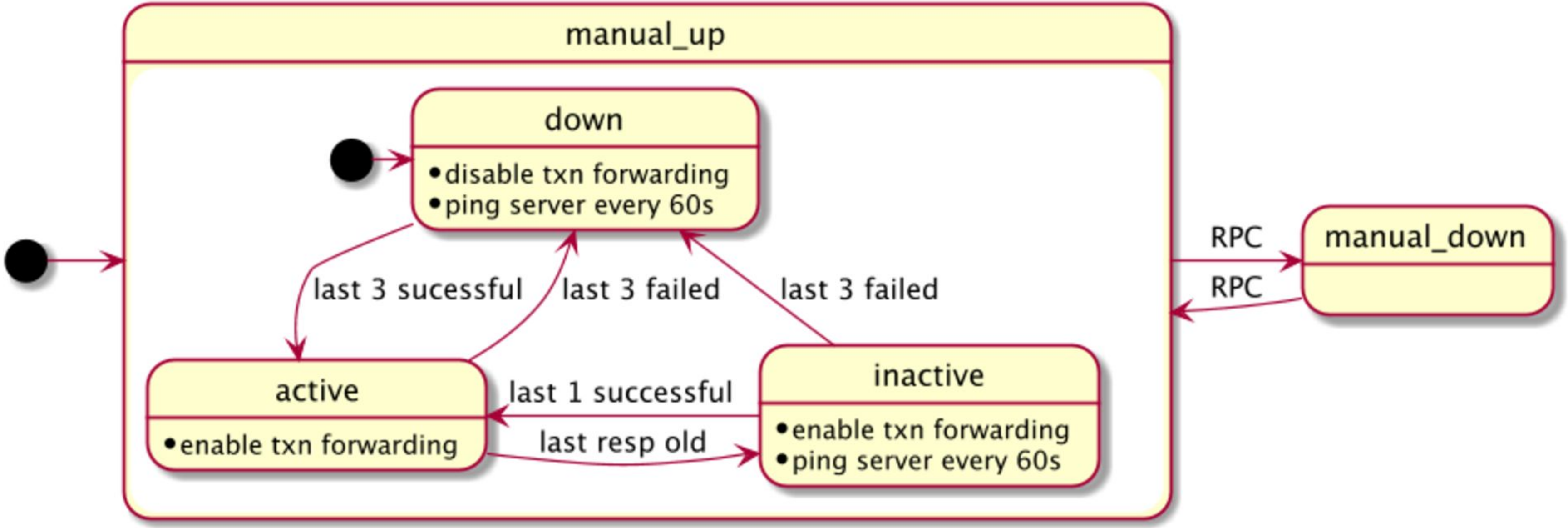
- LoadBalancing
- RR

Discuss: RoundRobin vs Uniform Random

NH Health Check


HealthCheckPlugin Service
NHAddr
HCService
UpdateQueue
HttpFetcher
poll status
request update
new fetcher
Allocate $(min(num_requests,limit)) Fetchers
while request queue
pop
fetch
status change
pop(empty)
NHAddr
HCService
UpdateQueue
HttpFetcher

Upstream Health Check States



***all numbers configurable separately.

Discuss:
Host Up/Down Metrics

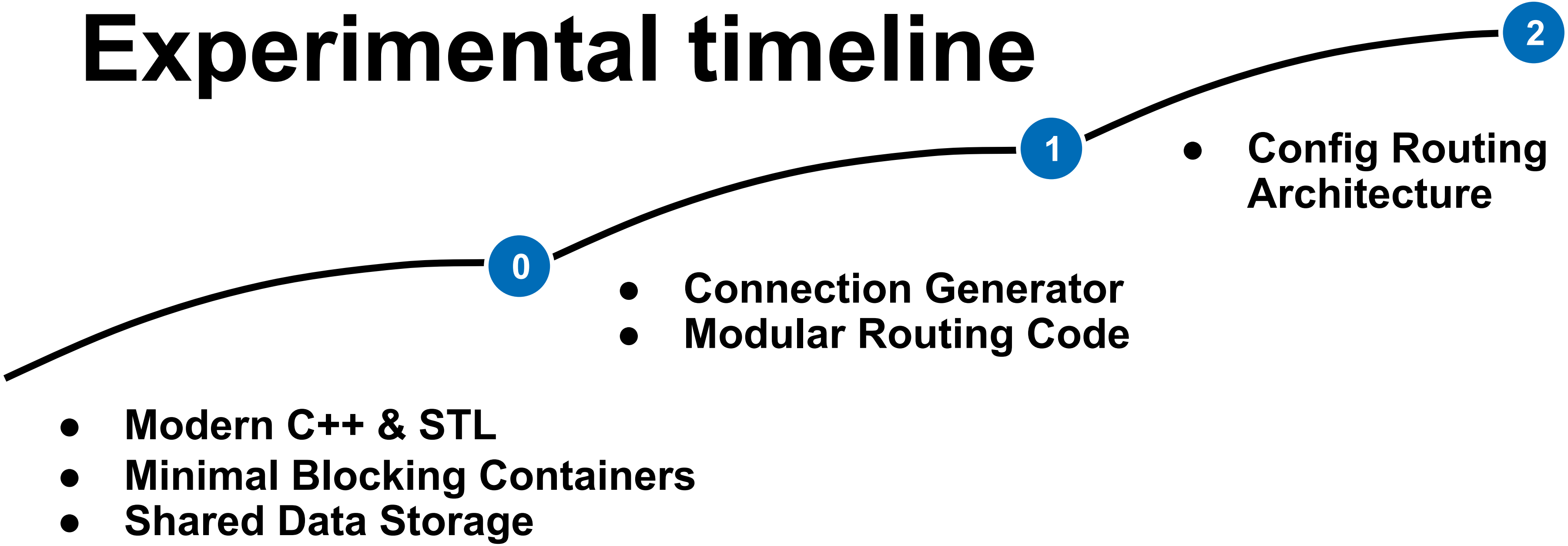
Discuss:
Pre-warming cache

Discuss:
Hot Object Caching

Discuss:
Load Balancing

Metrics & Methods

Experimental timeline



- Modern C++ & STL
- Minimal Blocking Containers
- Shared Data Storage

- Connection Generator
- Modular Routing Code

- Config Routing Architecture

The End

BONUS Discussion: Edge Compute