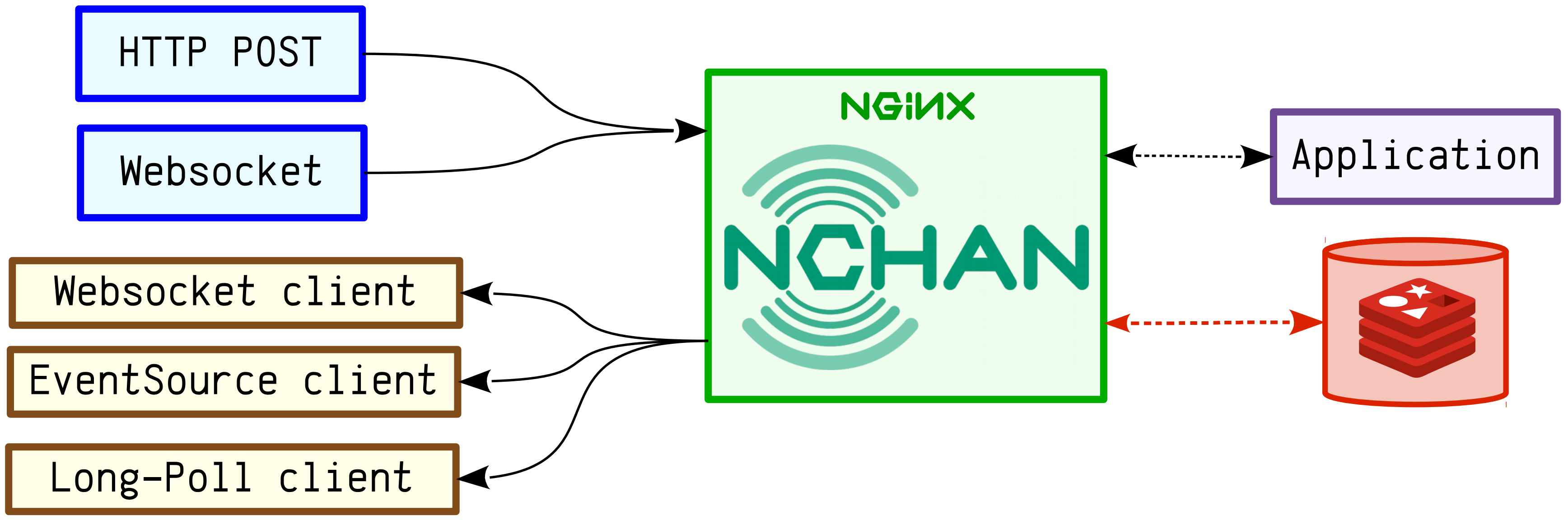
NCHAN

Pub/sub server for the modern web.
Flexible, scalable, easy to use.

NGINX redis

https://nchan.slact.net

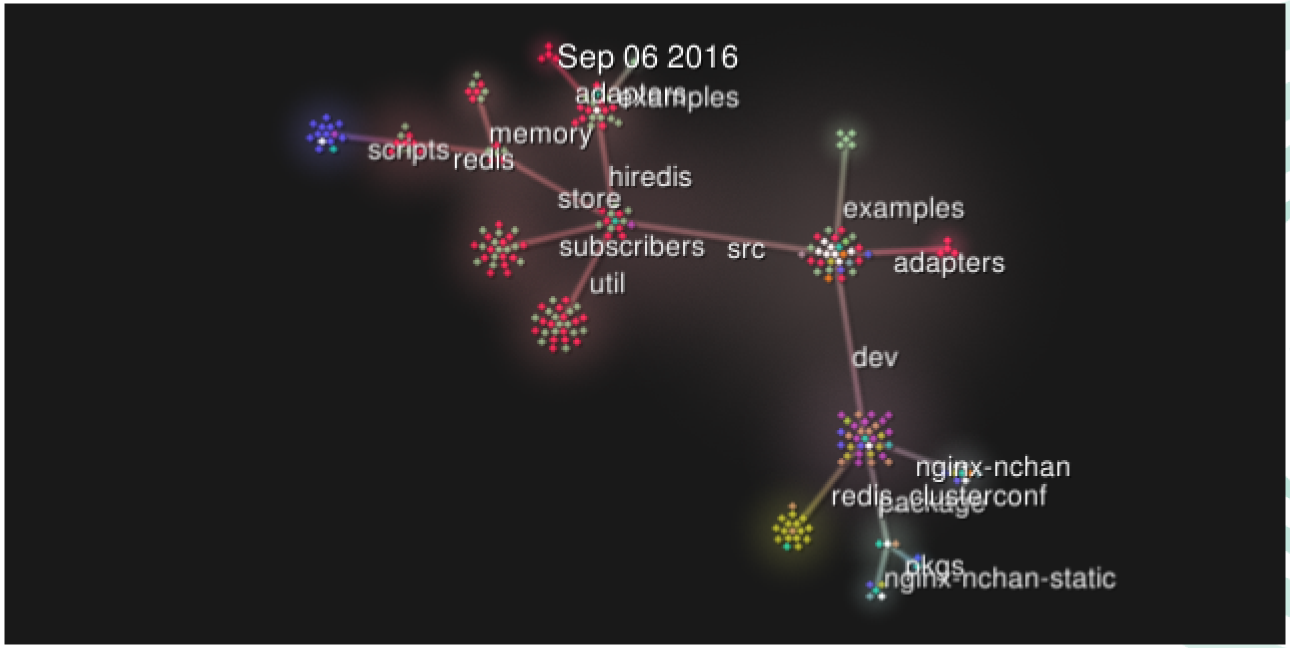
What is it?



- Buffering Pub/Sub server for web clients
- Publish via HTTP and Websocket
- Uses <u>channels</u> to coordinate publishers and subscribers.
- Flexible configuration and application hooks.
- Storage in-memory & on-disk, or in Redis.
- Scales vertically and horizontally

Some history…

- **nginx_http_push_module** (2009-2011)
  - Long-polling server
  - Used shared memory with a global mutex
- Rebuilt into Nchan in 2014-2015



The Other Guys

- *socket.io* (node.js)
  - Roll your own server
- *Lightstreamer* (java)
  - Complex session-based API.
- *Faye*
  - The oldest kid on the block. Uses a complex messaging protocol.
- Many others…

How NCHAN is different

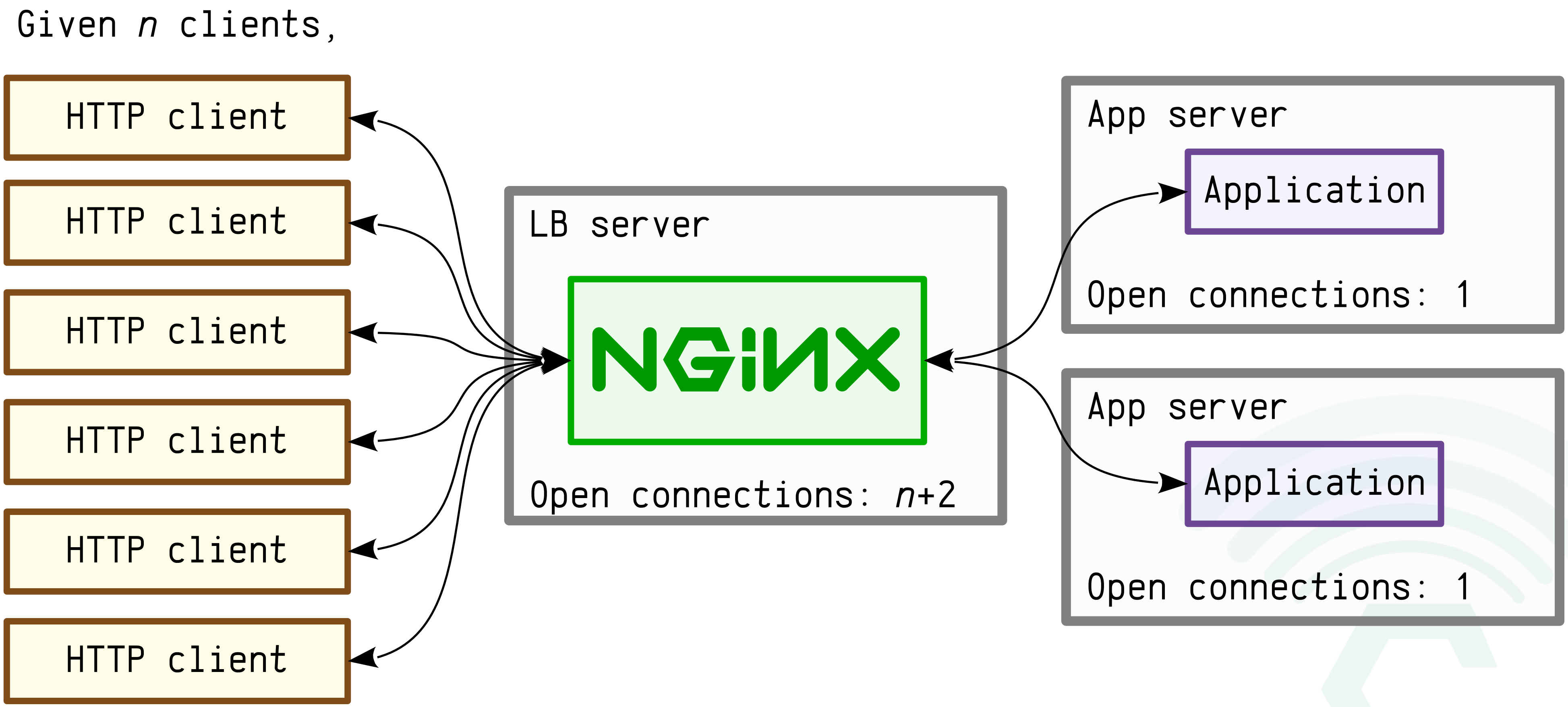
- No custom client needed
  - Just connect to a Websocket or EventSource URL.
- Configuration choices over connection complexity.
- API as RESTful as possible:
  - Publishers GET channel info, POST messages, DELETE channels.
  - Subscribers GET to subscribe.
- Everything* is configurable per-location.
- Limitless* scalability options.

* almost

Why an NGINX module?

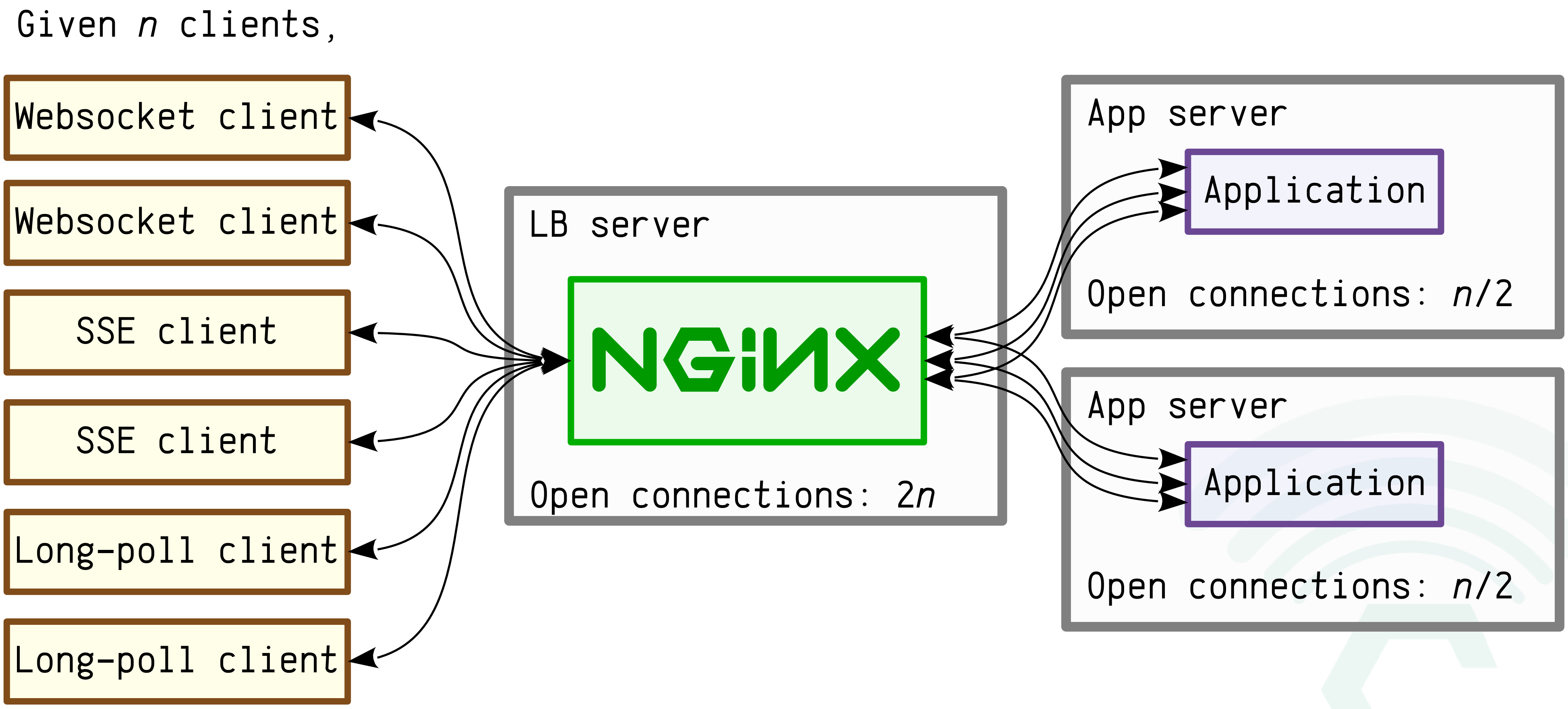
- Nginx is
  - asynchronous
  - fast
  - handles open connections well
  - probably your load balancer

Load Balancing HTTP clients



Load-balancing HTTP clients is efficient
(because HTTP is stateless)

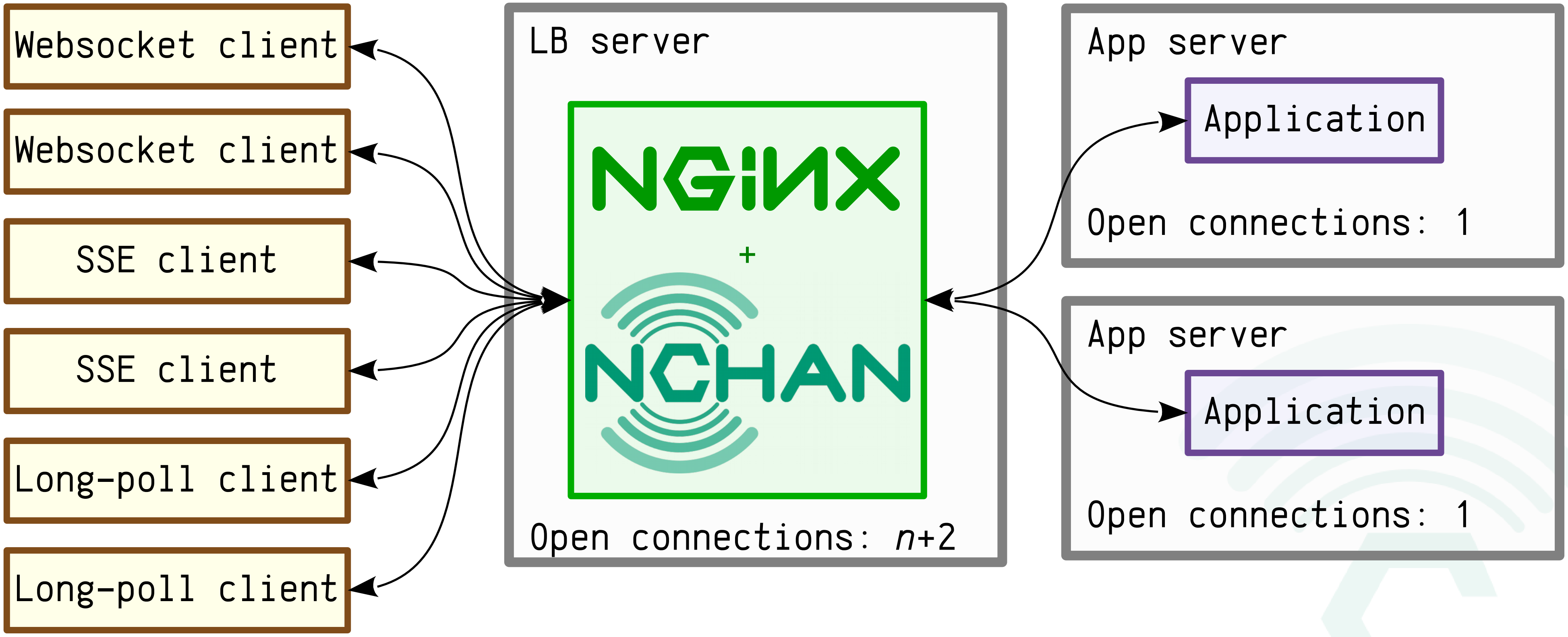
Load Balancing Websockets



Load-balancing server-push clients is not so nice (because each connection has state)

Enter Nchan

Given *n* clients,



Nchan can handle subscribers at the edge of your network

Configuration and API Simplicity

The Simplest Example

```
#very basic nchan config
worker_processes 5;
http {
  server {
    listen        80;

    location ~ /pub$ {
      nchan_publisher;
      nchan_channel_id test;
    }

    location ~ /sub$ {
      nchan_subscriber;
      nchan_channel_id test;
    }
  }
}
```

```
curl -X POST http://localhost/pub -d hi

  queued messages: 1
  last requested: 0 sec. ago
  active subscribers: 1
  last message id: 1461622867:0
```

```
var ws = new WebSocket("ws://127.0.0.1/sub");
ws.onmessage = function(e) {
  console.log(e.data);
};
```

```
hi
```

Channels & Channel IDs

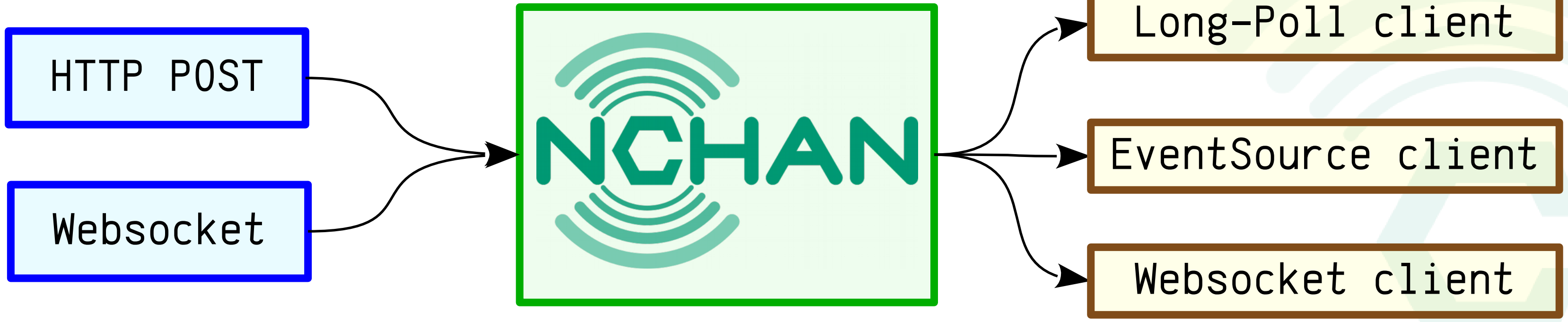
Channel ID sources

```
http {
  server {
    location /pub_by_querystring {
      #channel id from query string
      #/pub_by_querystring?id=10
      nchan_publisher;
      nchan_channel_id $arg_id;
    }
    location /pub_by_address {
      #channel id from named cookie and client ip
      nchan_publisher;
      nchan_channel_id $remote_addr;
    }
    location ~ /sub_by_url/(.*)$ {
      nchan_subscriber;
      nchan_channel_id $1;
    }
  }
}
```

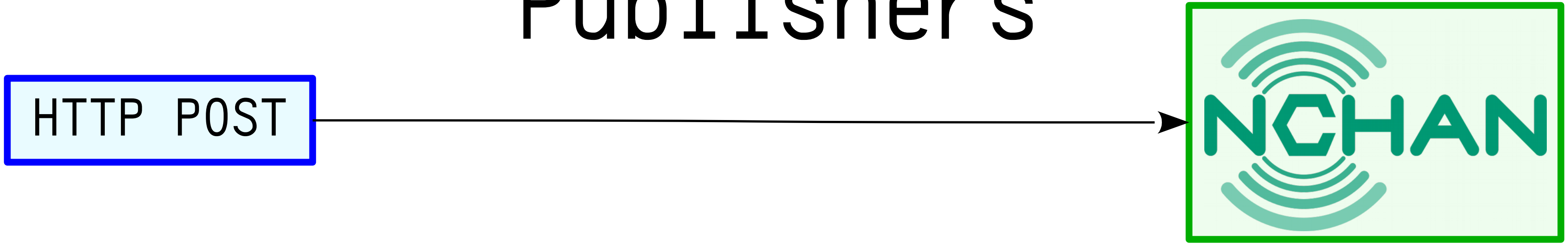
Multiplexed channels

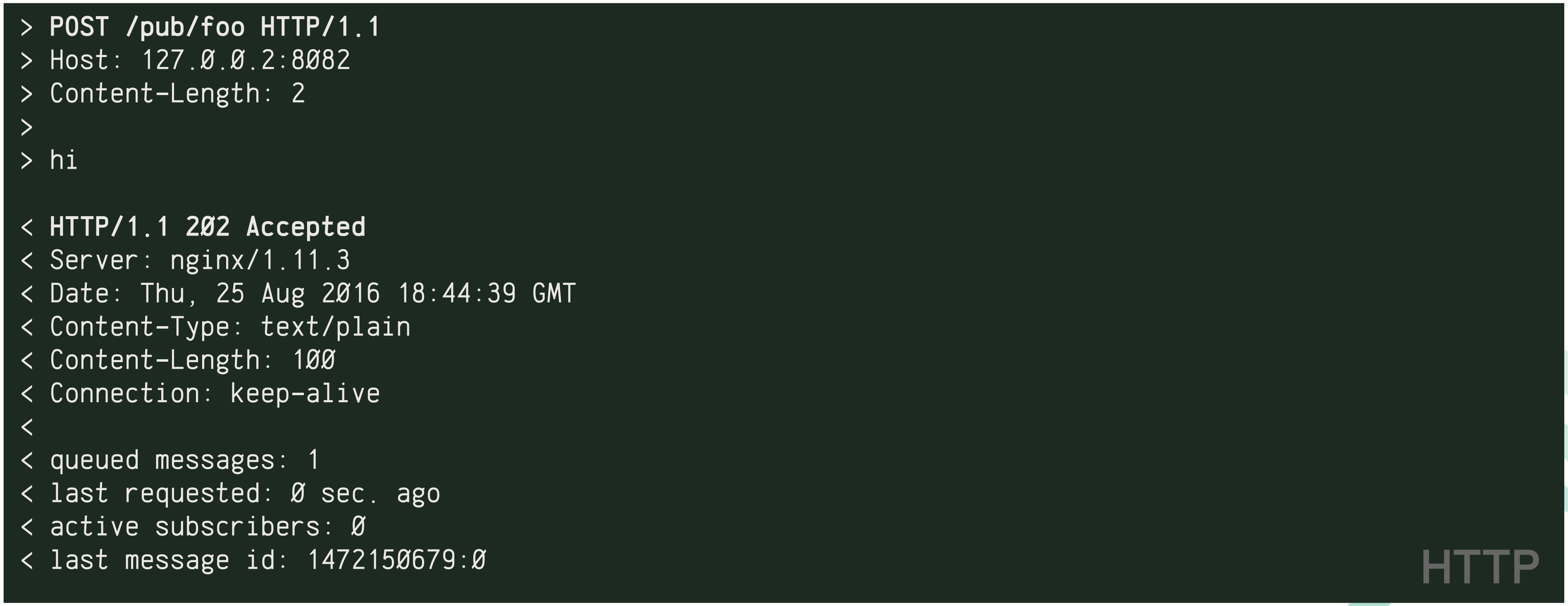
```
http {
  server {
    location ~ /sub_multi/(\w+)/(\w+)$ {
      #subscribe to 3 channels from one location
      #GET /sub_multi/foo/bar
      #subscribes to channels foo, bar, shared_channel
      nchan_subscriber;
      nchan_channel_id $1 $2 shared_channel;
    }
    location ~ /sub_multi_split/(.*)$ {
      #subscribe to up to 255 channels from one location
      #GET /sub_multi_split/1-2-3
      #subscribes to channels 1, 2, 3
      nchan_subscriber;
      nchan_channel_id $1;
      nchan_channel_id_split_delimiter "-";
    }
  }
}
```

Publishers and Subscribers



Publishers



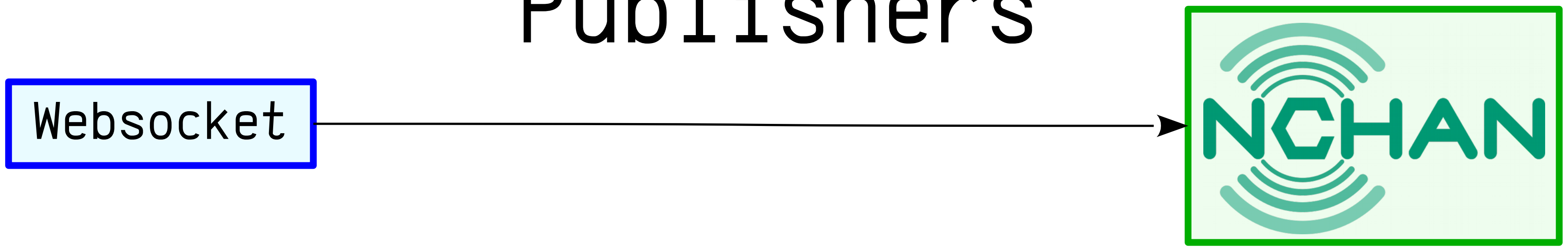


```
> POST /pub/foo HTTP/1.1
> Host: 127.0.0.2:8082
> Content-Length: 2
>
> hi

< HTTP/1.1 202 Accepted
< Server: nginx/1.11.3
< Date: Thu, 25 Aug 2016 18:44:39 GMT
< Content-Type: text/plain
< Content-Length: 100
< Connection: keep-alive
<
< queued messages: 1
< last requested: 0 sec. ago
< active subscribers: 0
< last message id: 1472150679:0
```

HTTP GET for channel information

HTTP DELETE to delete a channel

Publishers



```
var ws = new WebSocket("ws://127.0.0.1/pub/foo");
ws.onmessage = function(e) { console.log(e.data); };

ws.send("hello");
```

JS

```
queued messages: 1
last requested: 0 sec. ago
active subscribers: 0
last message id: 1472150679:0
```

console

Publisher Responses

Accept: text/plain

```
queued messages: 1
last requested: 0 sec. ago
active subscribers: 0
last message id: 1472150679:0
```

Accept: text/xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<channel>
  <messages>1</messages>
  <requested>0</requested>
  <subscribers>0</subscribers>
  <last_message_id>1472150679:0</last_message_id>
</channel>
```

Accept: text/json

```
{"messages": 1, "requested": 0, "subscribers": 0, "last_message_id": "1472150679:0" }
```

Accept: text/yaml

```
---
messages: 3
requested: 44
subscribers: 0
last_message_id: 1472330732:0
```

Subscribers

EventSource / SSE

```
var es = new EventSource("/sub/foo");
es.addEventListener("message",
  function(e){
    console.log(e.data);
  }
);
```

JS

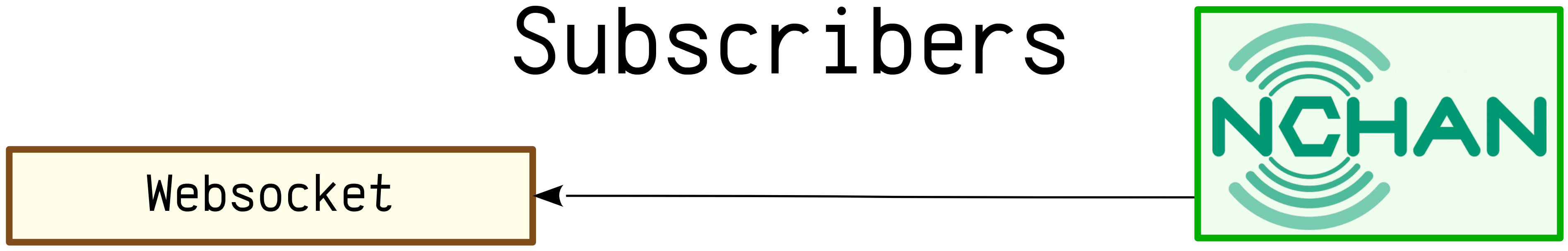
```
msg1

msg2

msg3
```

console

```
> GET /sub/foo HTTP/1.1
> Host: 127.0.0.1
> Accept: text/event-stream
>
< HTTP/1.1 200 OK
< Server: nginx/1.11.3
< Date: Thu, 25 Aug 2016 19:40:59 GMT
< Content-Type: text/event-stream; charset=utf-8
< Connection: keep-alive
<
: hi

id: 1472154531:0
data: msg1

id: 1472154533:0
data: msg2

id: 1472154537:0
data: msg3
```

HTTP

Subscribers



```
var ws = new WebSocket(”ws://127.0.0.1/sub/foo”);
ws.onmessage = function(e) { console.log(e.data); };
```

JS

```
msg1

msg2

msg3
```

console

Subscribers

HTTP Long-Polling

```
> GET /sub/foo HTTP/1.1
> Host: 127.0.0.1:8082
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: nginx/1.11.3
< Date: Thu, 25 Aug 2016 19:04:24 GMT
< Content-Length: 4
< Last-Modified: Thu, 25 Aug 2016 19:04:24 GMT
< Etag: 0
< Connection: keep-alive
< Vary: If-None-Match, If-Modified-Since
< msg1

> GET /sub/foo HTTP/1.1
> Host: 127.0.0.1:80
> Accept: */*
> If-Modified-Since: Thu, 25 Aug 2016 19:04:24 GMT
> If-None-Match: 0
>
< HTTP/1.1 200 OK
< Server: nginx/1.11.3
< Date: Thu, 25 Aug 2016 19:04:28 GMT
< Content-Length: 4
< Last-Modified: Thu, 25 Aug 2016 19:04:28 GMT
< Etag: 0
< Connection: keep-alive
< Vary: If-None-Match, If-Modified-Since
< msg2
```

HTTP

NchanSubscriber.js

Optional client wrapper library

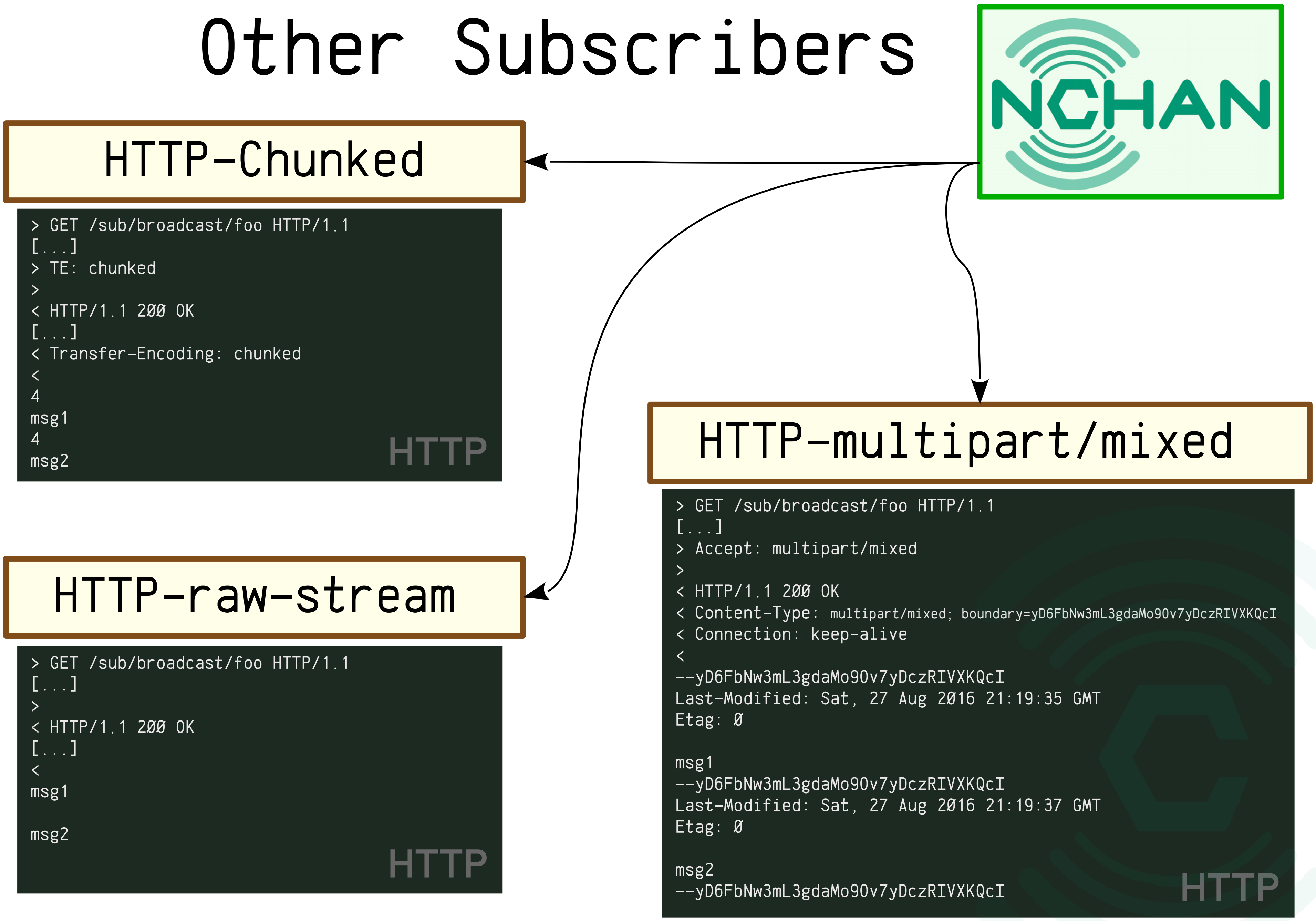
- Supports WS, EventSource, & Longpoll with fallback
- Resumable connections (even WS, using a subprotocol)
- Cross-tab connection sharing

```
var sub = new NchanSubscriber("/sub/foo", {shared: true});

sub.on("message", function(message, message_metadata) {
  console.log(message);
});

sub.start();
```

NchanSubscriber.js

```
opt = {
  subscriber: 'longpoll', 'eventsource', or 'websocket',
    //or an array of the above indicating subscriber type preference
  reconnect: undefined or 'session' or 'persist'
    //if the HTML5 sessionStore or localStore should be used to resume
    //connections interrupted by a page load
  shared: true or undefined
    //share connection to same subscriber url between browser windows and tabs
    //using localStorage.
};
var sub = new NchanSubscriber(url, opt);

sub.on("message", function(message, message_metadata) {
  // message is a string
  // message_metadata may contain 'id' and 'content-type'
});
sub.on('connect', function(evt) {
  //fired when first connected.
});
sub.on('disconnect', function(evt) {
  // when disconnected.
});
sub.on('error', function(code, message) {
  //error callback
});
sub.reconnect; // should subscriber try to reconnect? true by default.
sub.reconnectTimeout; //how long to wait to reconnect? does not apply to EventSource
sub.lastMessageId; //last message id. useful for resuming a connection without loss or repetition.

sub.start(); // begin (or resume) subscribing
sub.stop(); // stop subscriber. do not reconnect.
```

Other Subscribers



```
> GET /sub/broadcast/foo HTTP/1.1
[...]
> TE: chunked
>
< HTTP/1.1 200 OK
[...]
< Transfer-Encoding: chunked
<
4
msg1
4
msg2
```

```
> GET /sub/broadcast/foo HTTP/1.1
[...]
> Accept: multipart/mixed
>
< HTTP/1.1 200 OK
< Content-Type: multipart/mixed; boundary=yD6FbNw3mL3gdaMo90v7yDczRIVXKQcI
< Connection: keep-alive
<
--yD6FbNw3mL3gdaMo90v7yDczRIVXKQcI
Last-Modified: Sat, 27 Aug 2016 21:19:35 GMT
Etag: 0

msg1
--yD6FbNw3mL3gdaMo90v7yDczRIVXKQcI
Last-Modified: Sat, 27 Aug 2016 21:19:37 GMT
Etag: 0

msg2
--yD6FbNw3mL3gdaMo90v7yDczRIVXKQcI
```

```
> GET /sub/broadcast/foo HTTP/1.1
[...]
>
< HTTP/1.1 200 OK
[...]
<
msg1

msg2
```

Message Buffering

Message Buffer Size

```
worker_processes 5;
http {
  server {
    listen        80;
    location ~ /pub/(.+)$ {
      #POST /pub/foo
      nchan_message_buffer_length 20;
      nchan_message_timeout 5m;
      nchan_publisher;
      nchan_channel_id $1;
    }
    location ~ /sub/(.+)$ {
      nchan_subscriber;
      nchan_channel_id $1;
    }
  }
}
```

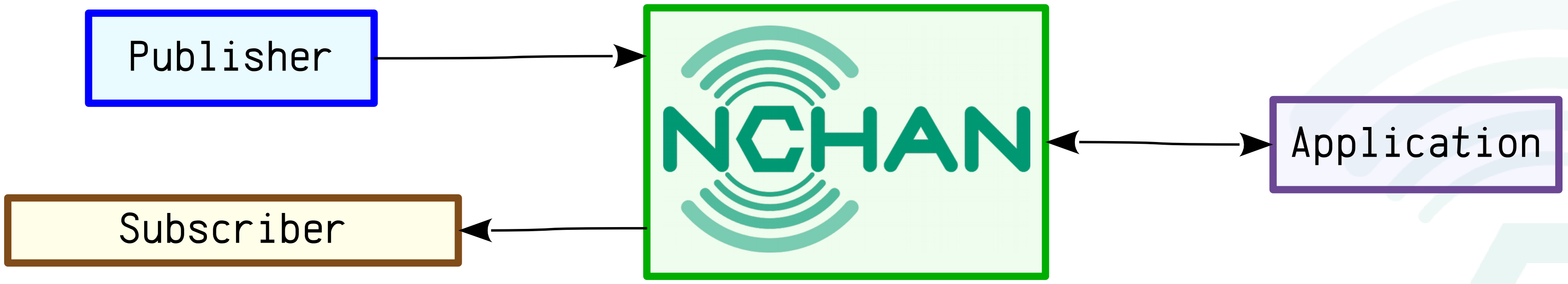
Dynamic Buffer Sizing

```
worker_processes 5;
http {
  server {
    listen       80;
    location ~ /pub/(.+)$ {
      #POST /pub/foo?buflen=10&ttl=30s
      nchan_message_buffer_length $arg_buflen;
      nchan_message_timeout $arg_ttl;
      nchan_publisher;
      nchan_channel_id $1;
    }
    location ~ /sub/(.+)$ {
      nchan_subscriber;
      nchan_channel_id $1;
    }
  }
}
```

Where to start?

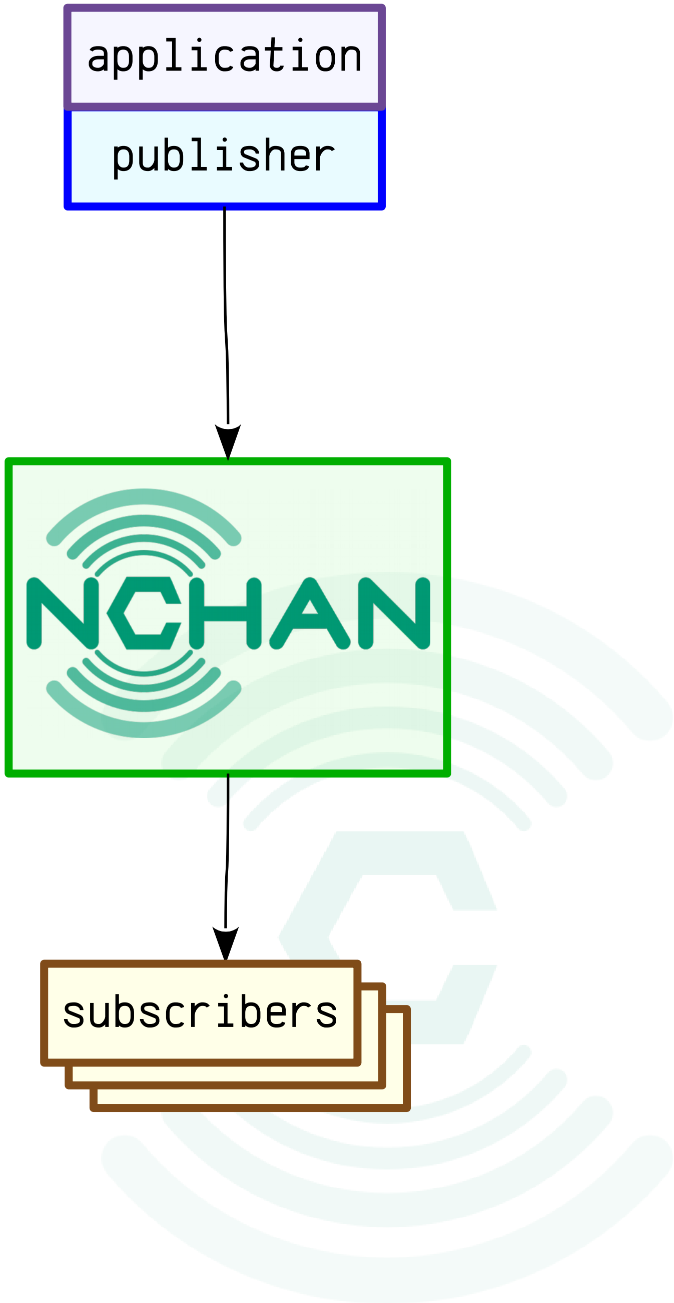
```
worker_processes 5;
http {
  server {
    listen        80;
    location ~ /pub/(.+)$ {
      nchan_message_buffer_length 20;
      nchan_message_timeout 5m;
      nchan_publisher;
      nchan_channel_id $1;
    }
    location ~ /sub/(.+)$ {
      nchan_subscriber_first_message 5;
      nchan_subscriber;
      nchan_channel_id $1;
    }

  }
}
```

Application Interface



Application Publisher

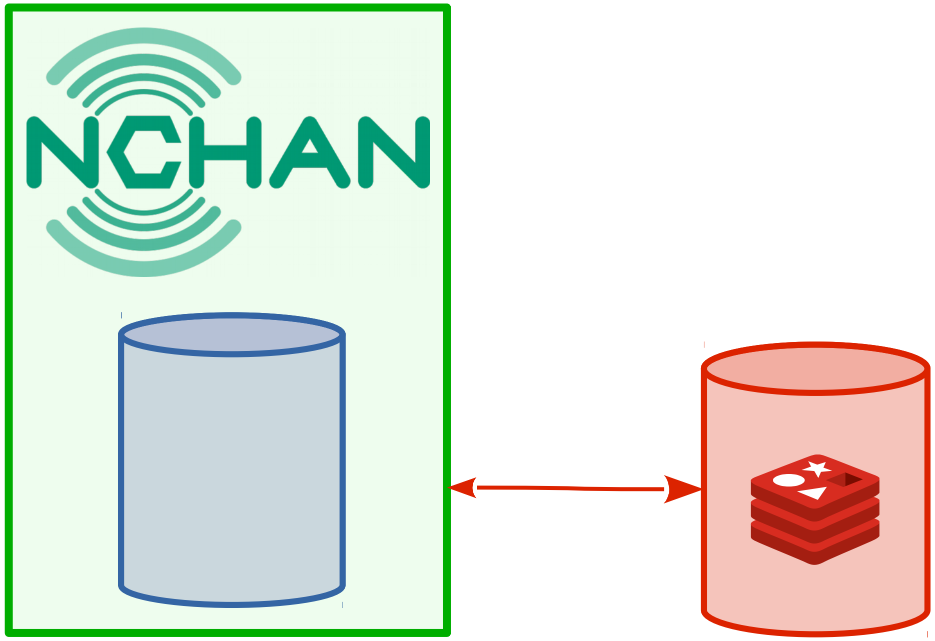
```
http {

  server {
    listen 127.0.0.1:8080;
    location ~ /pub/(.+)$ {
      nchan_publisher;
      nchan_channel_id $1;
    }
  }

  server {
    listen 80;
    location ~ /sub/(.+)$ {
      nchan_subscriber;
      nchan_channel_id $1;
    }
  }
}
```



Upstream Authentication

```
http {
  server {
    location = /upstream_auth {
      proxy_pass http://my_application.local/auth;
      proxy_set_header X-Channel-Id $nchan_channel_id;
      proxy_set_header X-Original-URI $request_uri;
    }
    location ~ /pub/(.+)$ {
      nchan_authorize_request /upstream_auth;
      nchan_publisher;
      nchan_channel_id $1;
    }
    location ~ /sub/(.+)$ {
      nchan_authorize_request /upstream_auth;
      nchan_subscriber;
      nchan_channel_id $1;
    }
  }
}
```

Storage



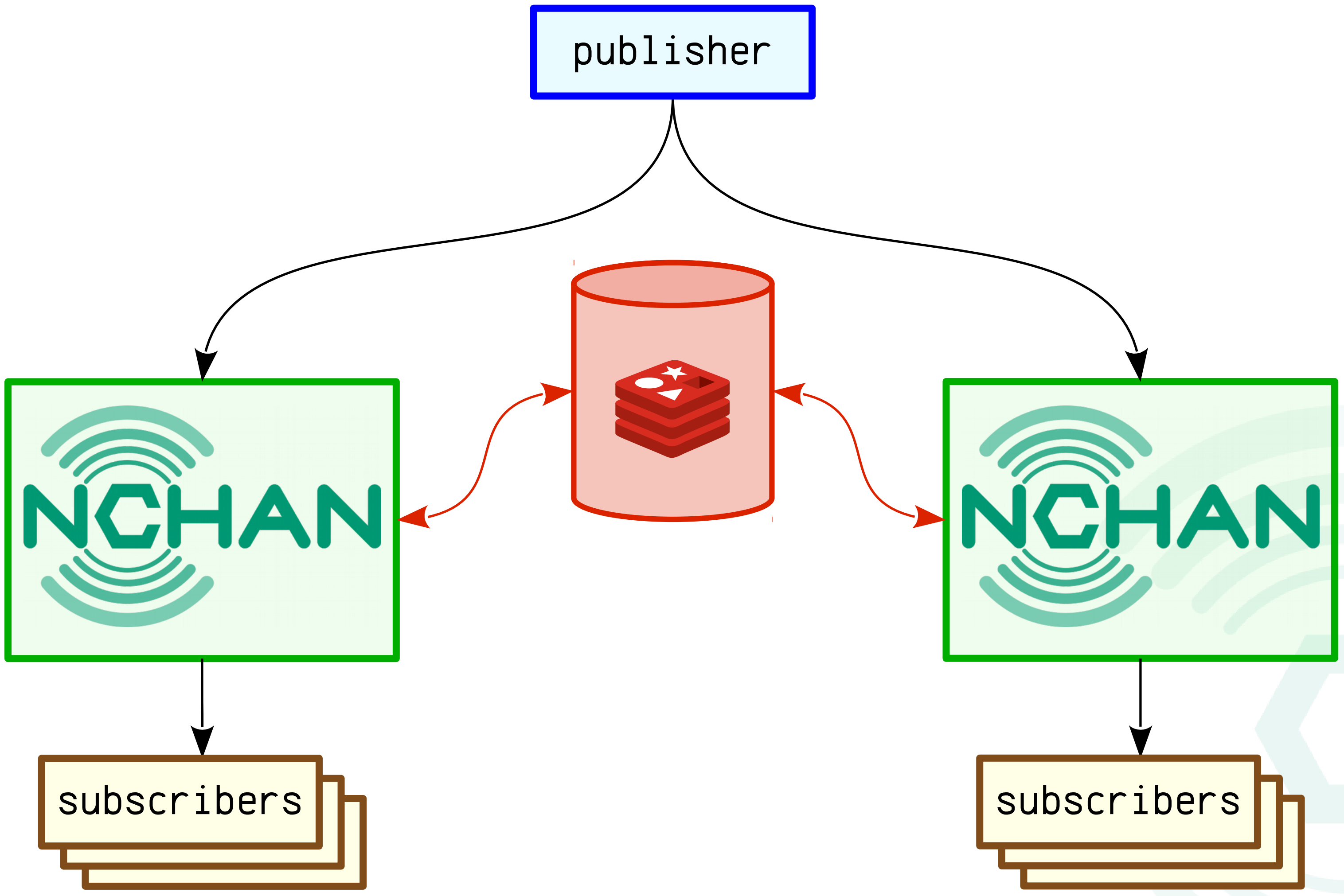
Shared Memory Storage

```
http {
  nchan_max_reserved_memory 1024M;
  server {
    location ~ /pub/(\w+)$ {
      nchan_publisher;
      nchan_channel_id $1;
    }
    location ~ /sub(\w+)$ {
      nchan_subscriber;
      nchan_channel_id $1;
    }
  }
}
```

redis Server Storage

```
http {
  nchan_redis_url "redis://redis_server.local";

  server {
    location ~ /pub/(\w+)$ {
      nchan_publisher;
      nchan_channel_id $1;
      nchan_use_redis on;
    }
    location ~ /sub(\w+)$ {
      nchan_subscriber;
      nchan_channel_id $1;
      nchan_use_redis on;
    }
  }
}
```

Scaling Broadcasts With redis

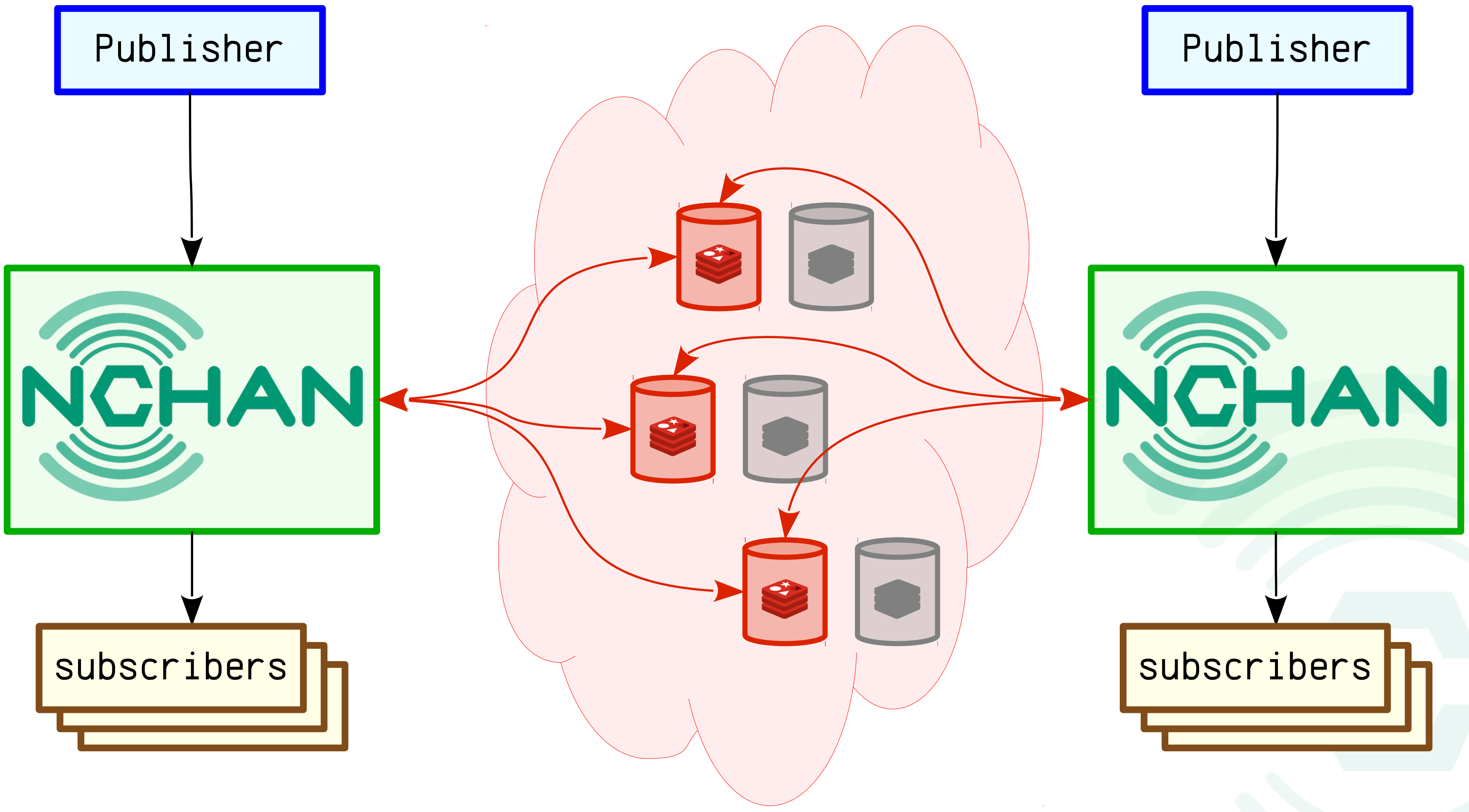


redis Cluster Storage

```
http {
  upstream redis_cluster {
    nchan_redis_server redis://redis_server1.local;
    nchan_redis_server redis://redis_server2.local;
    nchan_redis_server redis://redis_server3.local;
  }
  server {
    location ~ /pub/(\w+)$ {
      nchan_redis_pass redis_cluster;
      nchan_publisher;
      nchan_channel_id $1;
    }
    location ~ /sub(\w+)$ {
      nchan_redis_pass redis_cluster;
      nchan_subscriber;
      nchan_channel_id $1;
    }
  }
}
```

Scaling with redis Cluster:

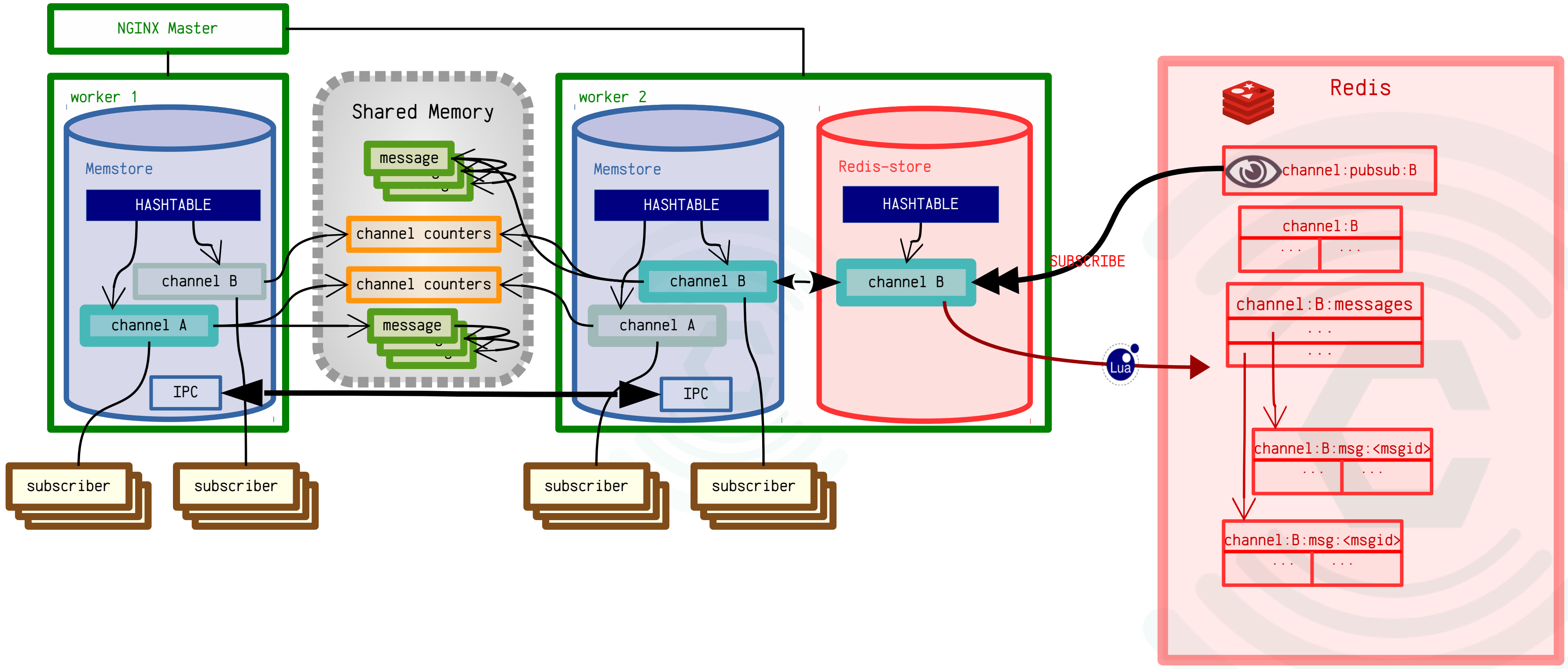
Hello High Availability



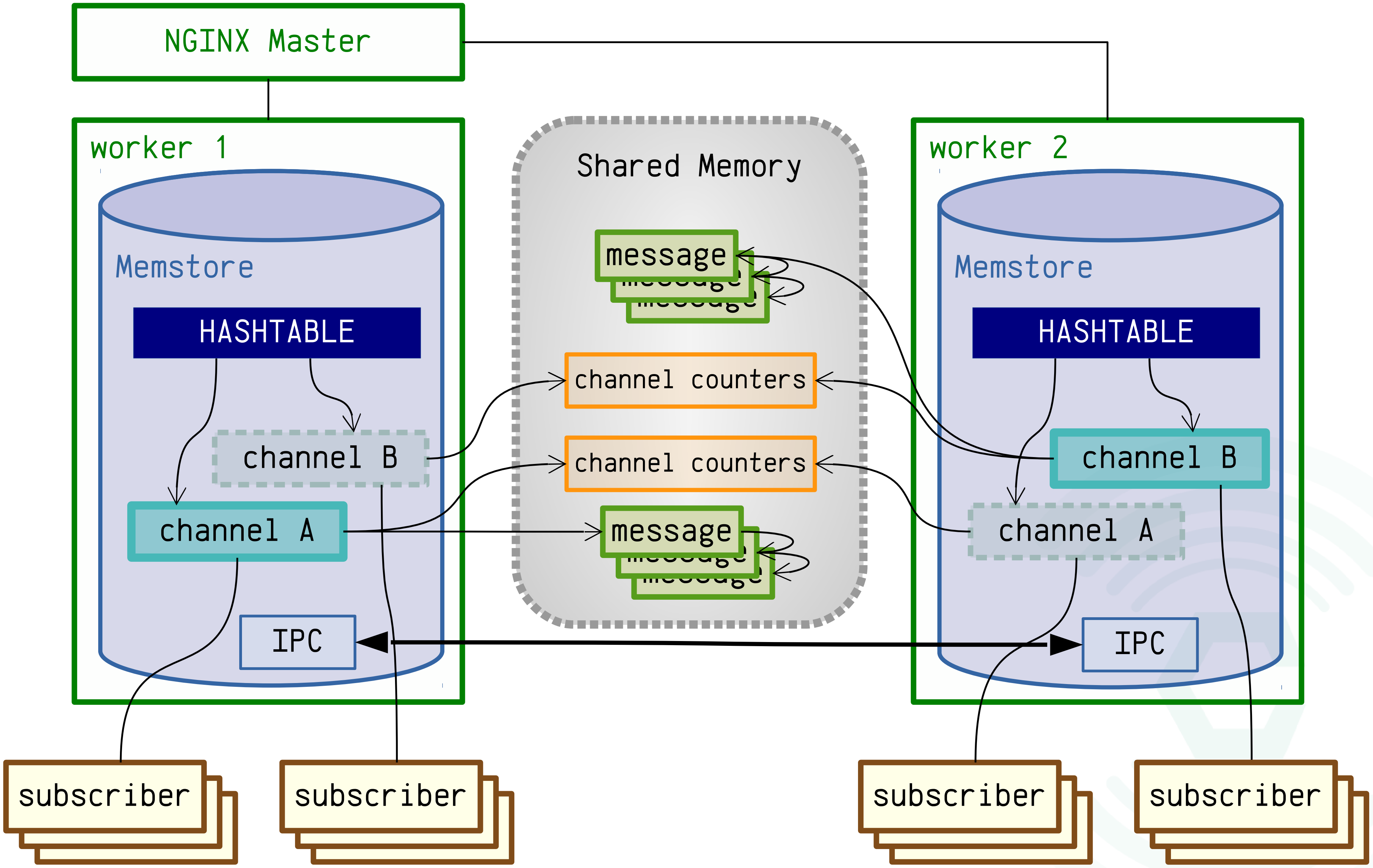
Other Features

- HTTP/2 Support
- Built-in workarounds for browser quirks
- nchan_stub_status for vitals and load monitoring
- Access-Control (CORS) support
- Upstream message passing
- Meta Channels
- Hide channel IDs with X-Accel-Redirect
- Pubsub location endpoints
- …and more

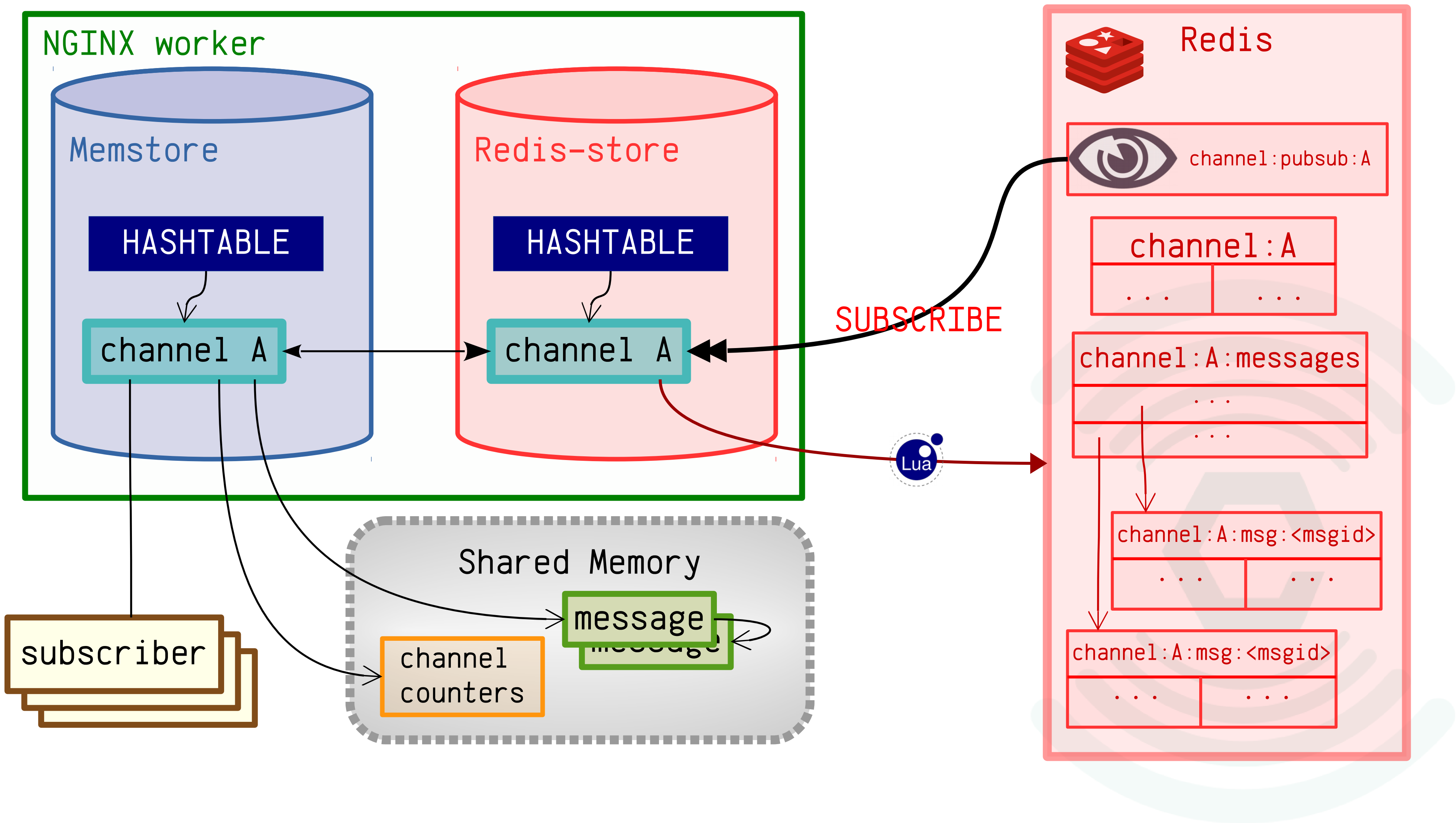
Architecture



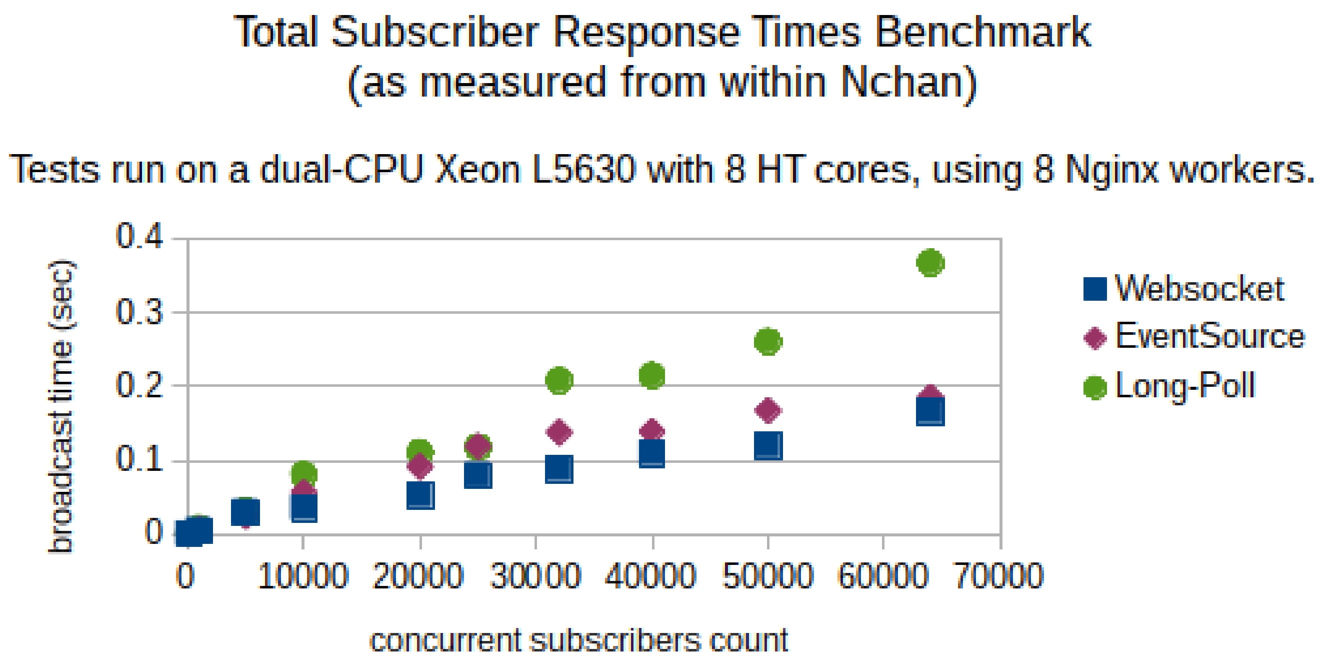
Architecture Overview: Memory Store



Architecture Overview:Memory & Redis Store



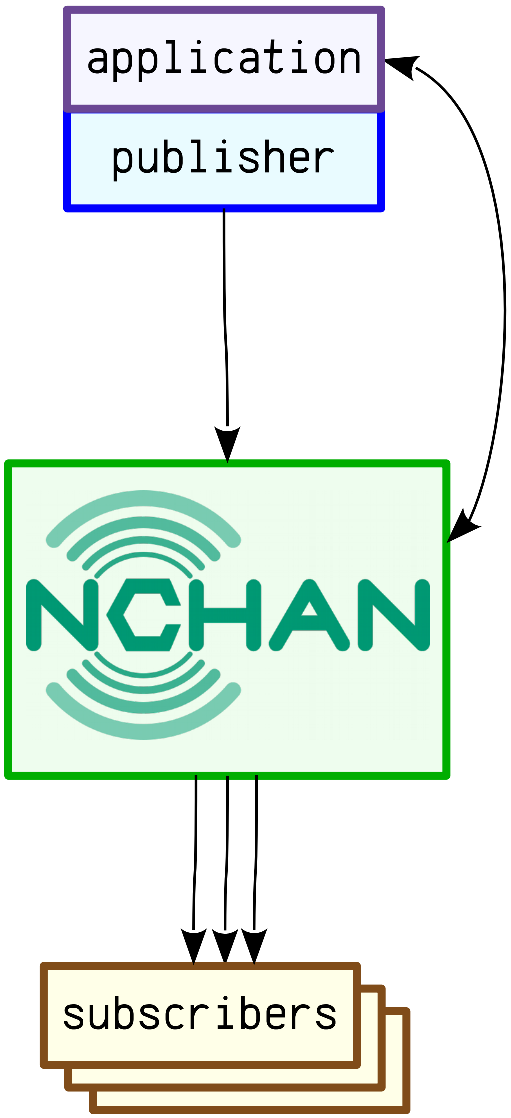
But is it fast?...



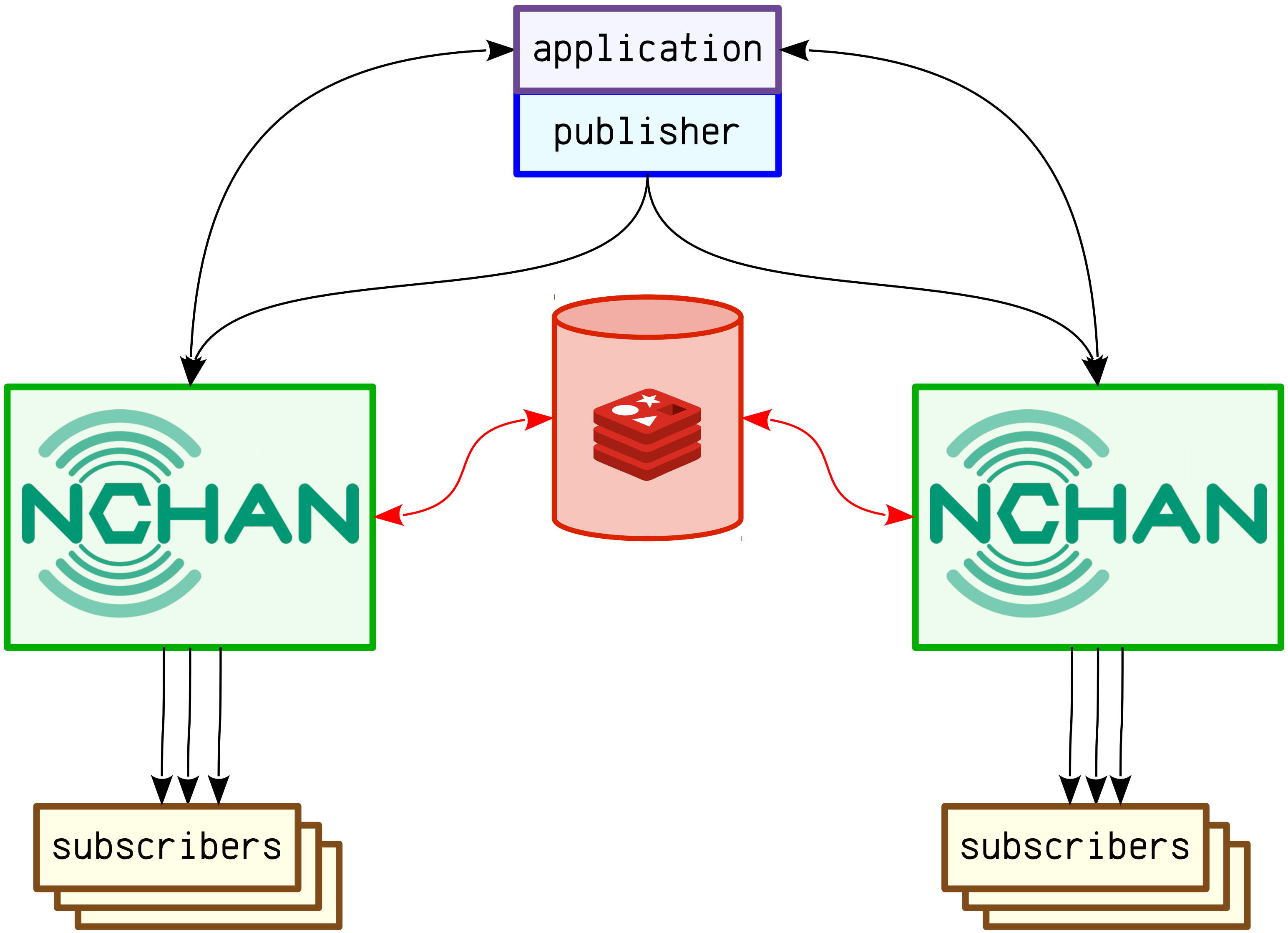
- Yeah, it’s pretty fast...
  - 300K Websocket responses per second (and that’s on 7 year old hardware)
- And it will only get faster...

Scalability

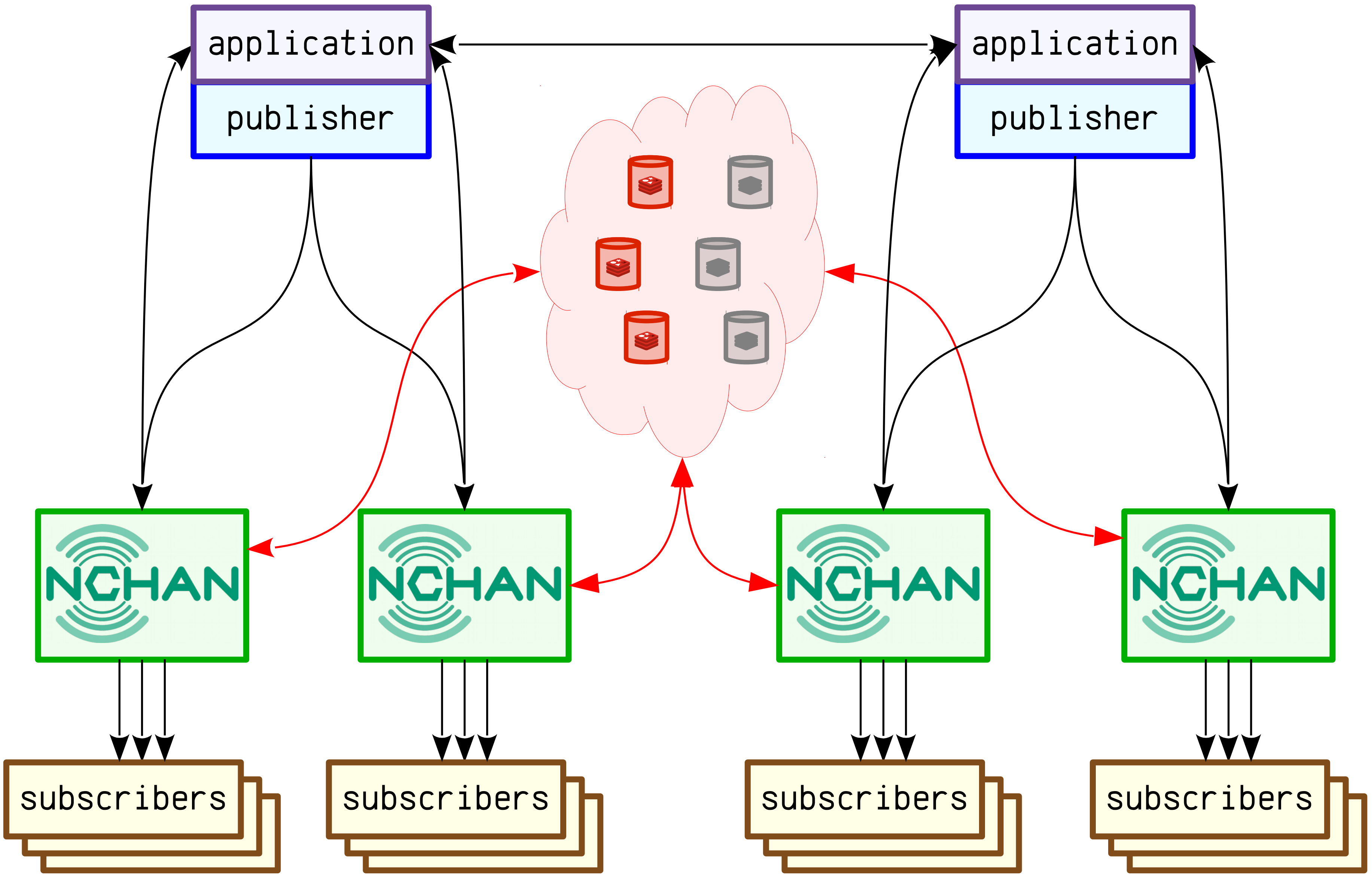
Superior Scalability: Start Small



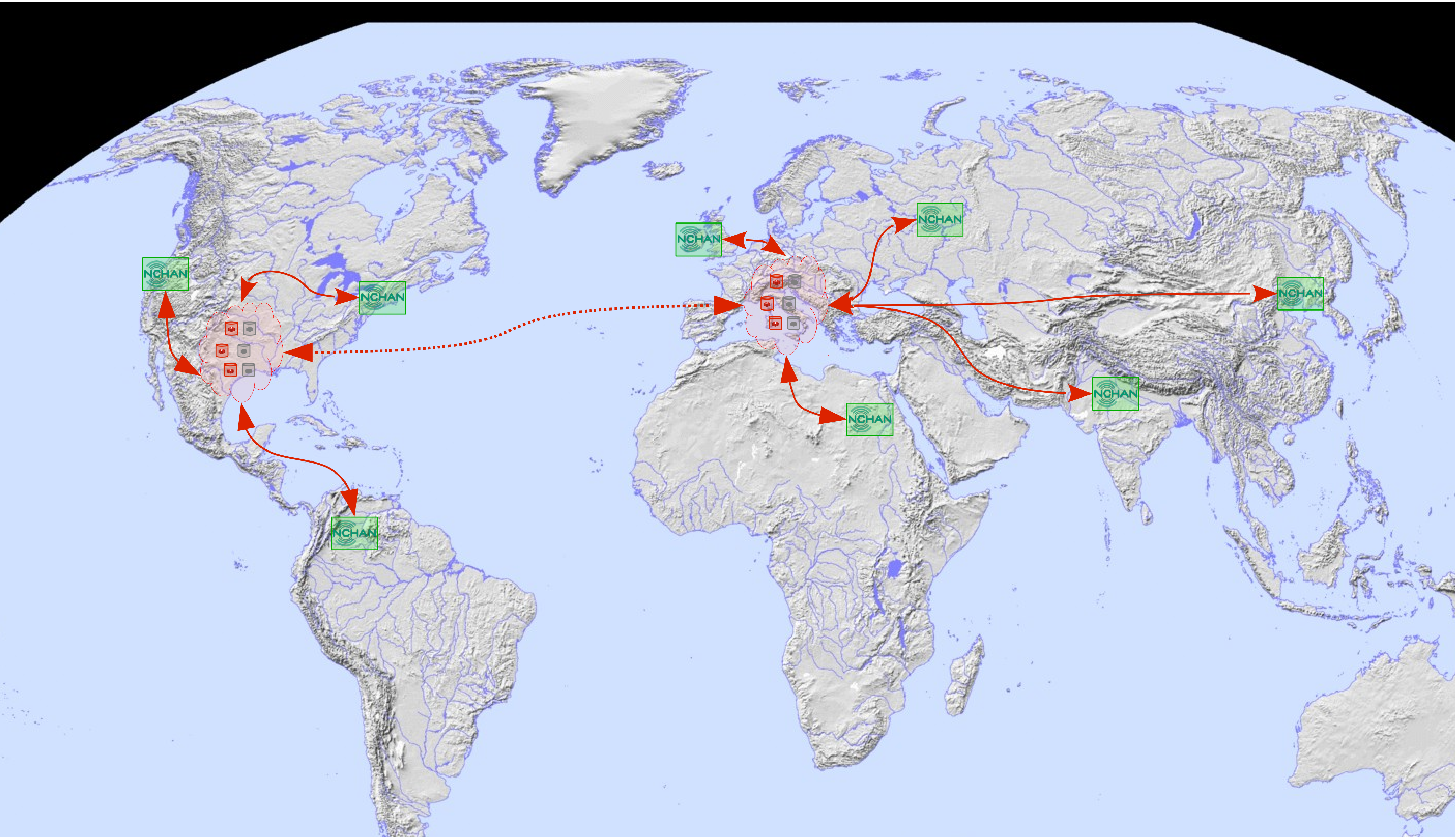
Superior Scalability: Grow Fast



Superior Scalability: Get Big



Superior Scalability: Go Global



Try NCHAN

- Thorough documentation and examples at

  https://nchan.slact.net

- Build and run:
  - From source: http://github.com/slact/nchan
    - Build as a static or dynamic module
  - Pre-packaged: https://nchan.slact.net/#download

Fin

https://nchan.slact.net

Slides and notes at https://nchan.slact.net/nginxconf

Consulting services available.

Contact me: leo@slact.net

Support Nchan development

- *Paypal*: nchan@slact.net
- *Bitcoin*: 15dLBzRS4HLRwCCVjx4emYkxXcyAPmGxM3