facebook

FBTFTP

Facebook's Python3 open-source framework to build dynamic tftp servers

Angelo Failla

Production Engineer
Cluster infrastructure team
Facebook Ireland

EUROPYTHON 2016 Bilbao, 17-24 July

Who am I?

- **A Production Engineer**
  - Similar to SRE / DevOps
- **Based in Facebook Ireland, Dublin**
  - Since 2011
- **Cluster Infrastructure team member**
  - Owns data center core services
  - Owns E2E automation for bare metal provisioning and cluster management.

“There is no cloud, just other people’s computers...”

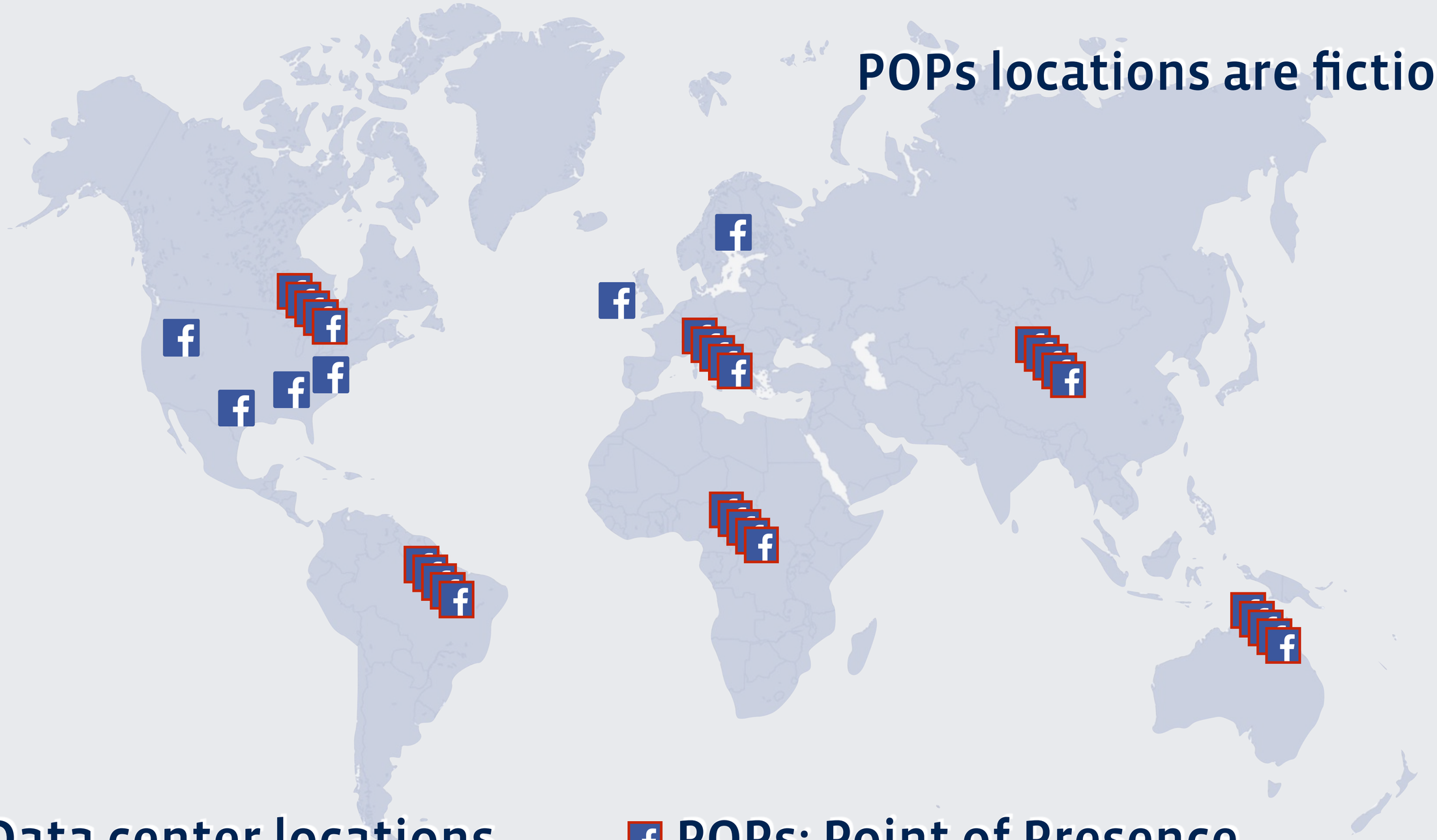
- a (very wise) person on the interwebz

“... and someone’s got to provision them.”

- Angelo


facebook

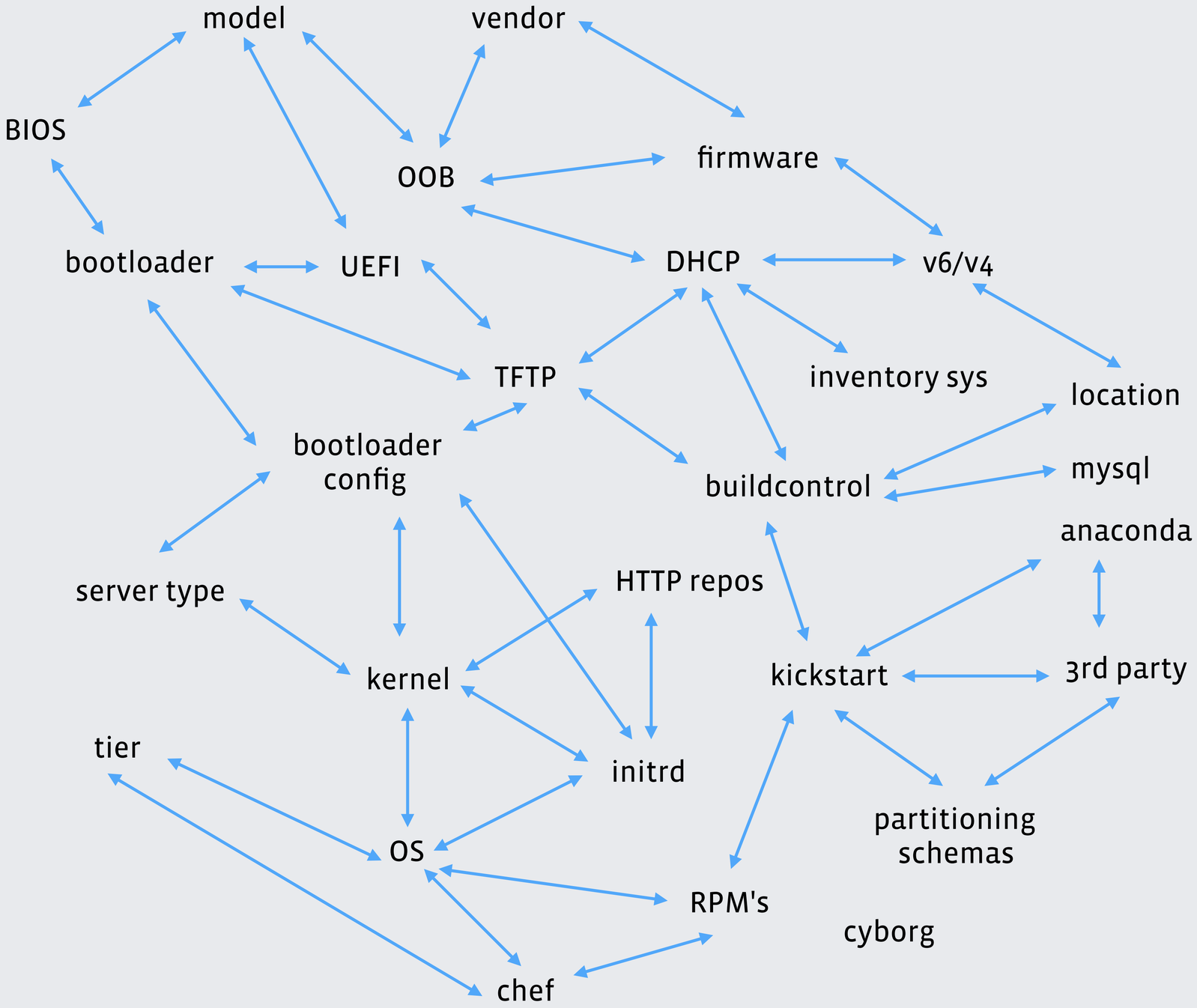
POPs locations are fictional

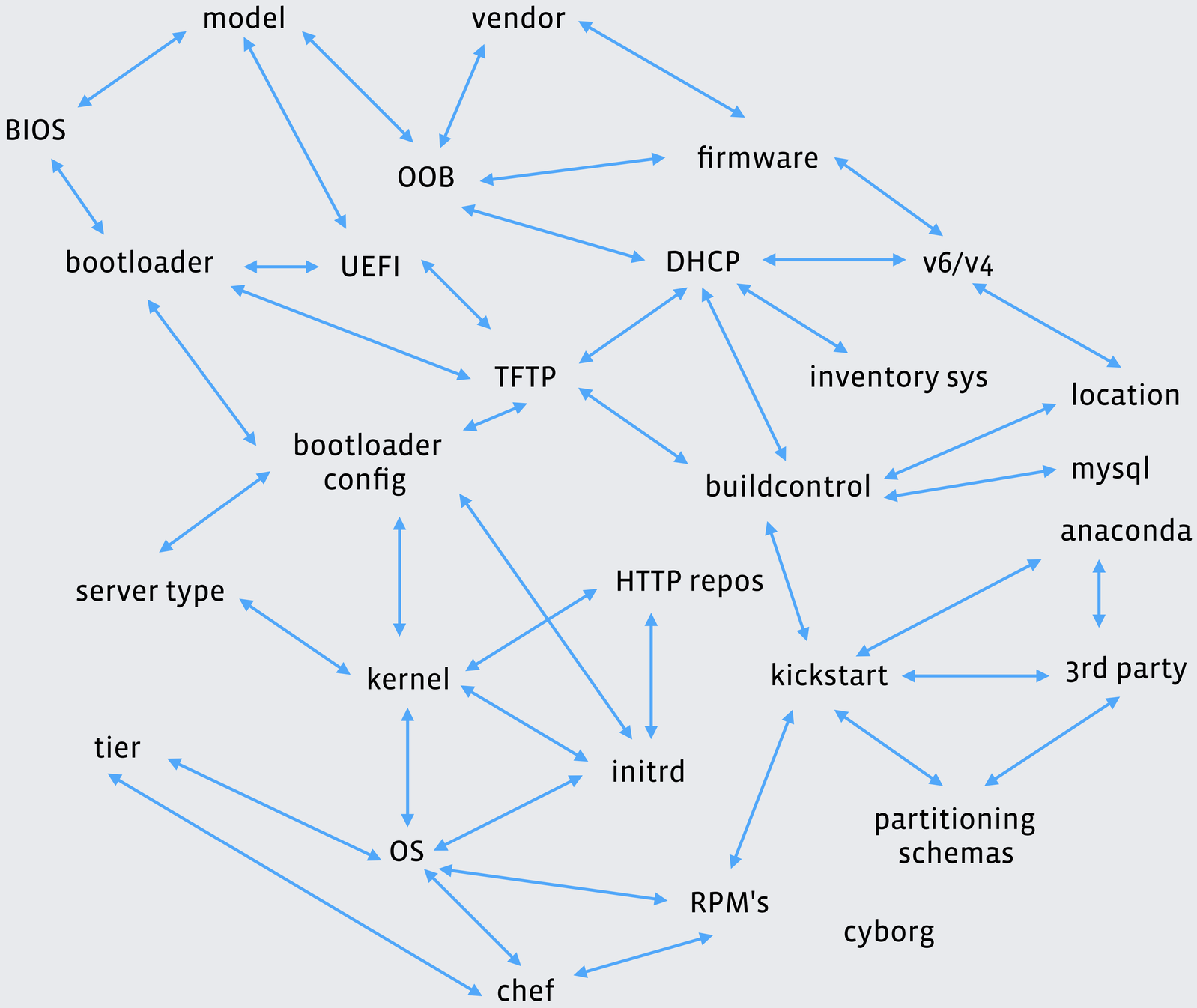


Data center locations

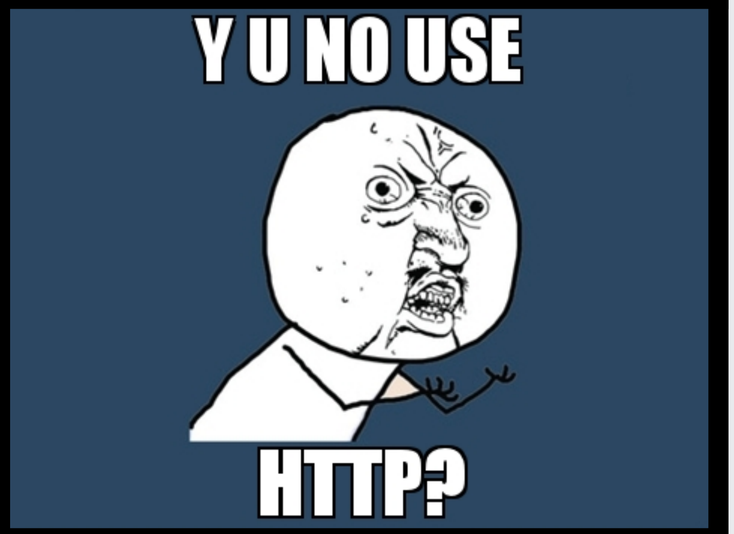
POPs: Point of Presence

HANDS FREE PROVISIONING:


model
vendor
BIOS
OOB
firmware
bootloader
UEFI
DHCP
v6/v4
TFTP
inventory sys
location
bootloader
config
buildcontrol
mysql
anaconda
server type
HTTP repos
kernel
kickstart
3rd party
tier
initrd
partitioning
schemas
OS
RPM's
cyborg
chef


model
vendor
BIOS
OOB
firmware
bootloader
UEFI
DHCP
v6/v4
TFTP
inventory sys
location
bootloader
config
buildcontrol
mysql
anaconda
server type
HTTP repos
kernel
kickstart
3rd party
tier
initrd
partitioning
schemas
OS
RPM's
cyborg
chef

TFTP



It’s common in Data Center/ISP environments

Simple protocol specifications

- Easy to implement
- UDP based -> produces small code footprint
  - Fits in small boot ROMs
- Embedded devices and network equipment

Traditionally used for netboot (with DHCPv[46])

Provisioning phases

- provides network config
- provides path for NBPs binaries

- provides NBPs
- provides config files for NBPs
- provides kernel/initrd

- fetches config via tftp
- fetches kernel/initrd (via http or tftp)

30+ years old protocol

[Docs] [txt|pdf] [Errata]

Obsoleted by: 1350

Errata Exist

Network Working Group
Request for Comments: 783

K. R. Sollins
MIT
June, 1981

Updates: IEN 133

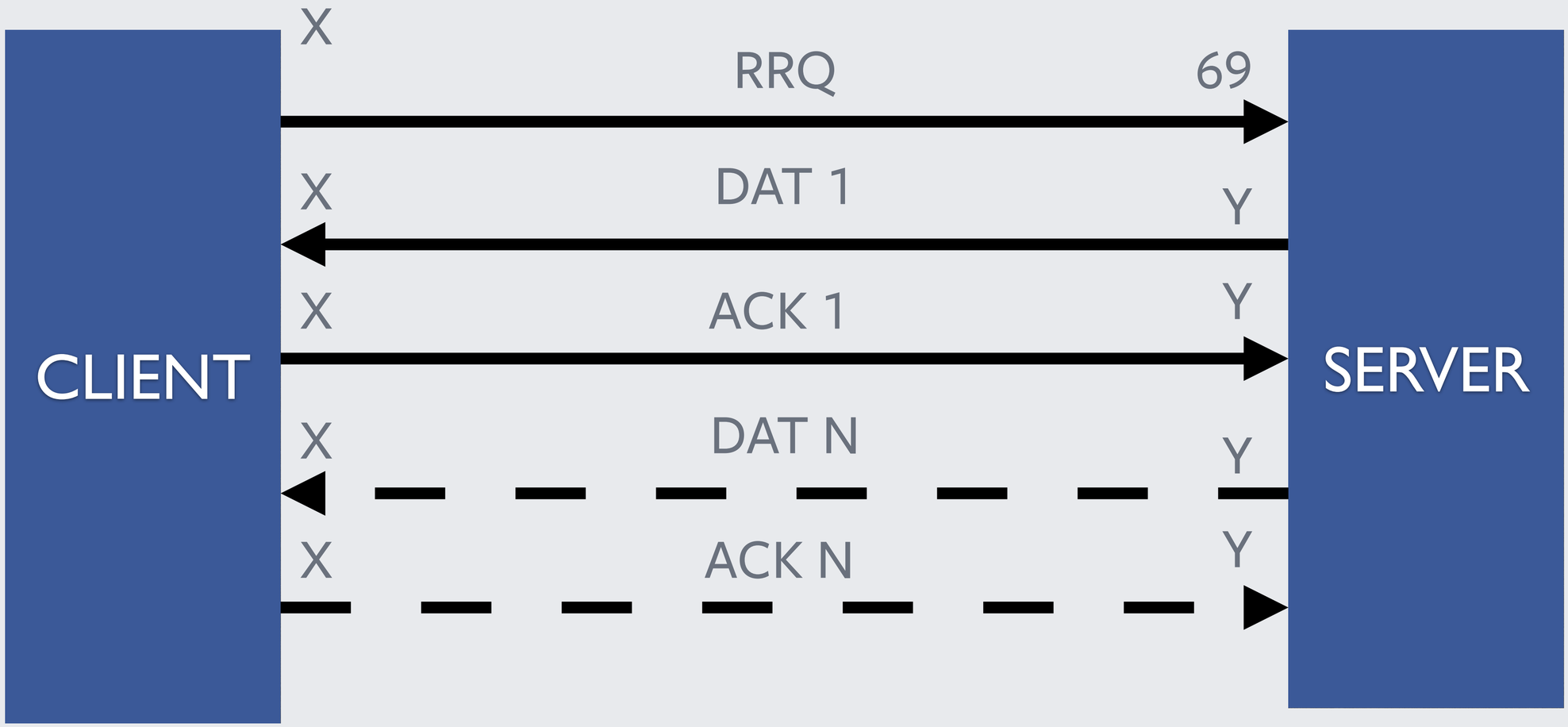
THE TFTP PROTOCOL (REVISION 2)

Summary

TFTP is a very simple protocol used to transfer files. It is from this that its name comes, Trivial File Transfer Protocol or TFTP. Each nonterminal packet is acknowledged separately. This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions.

me, ~1982 circa

Protocol in a nutshell (RRQ)



File size	Block Size	Latency	Time to download
80 MB	512 B	150ms	12.5 hours
80 MB	1400 B	150ms	4.5 hours
80 MB	512 B/ 1400 B	1ms	<1 minute

POP

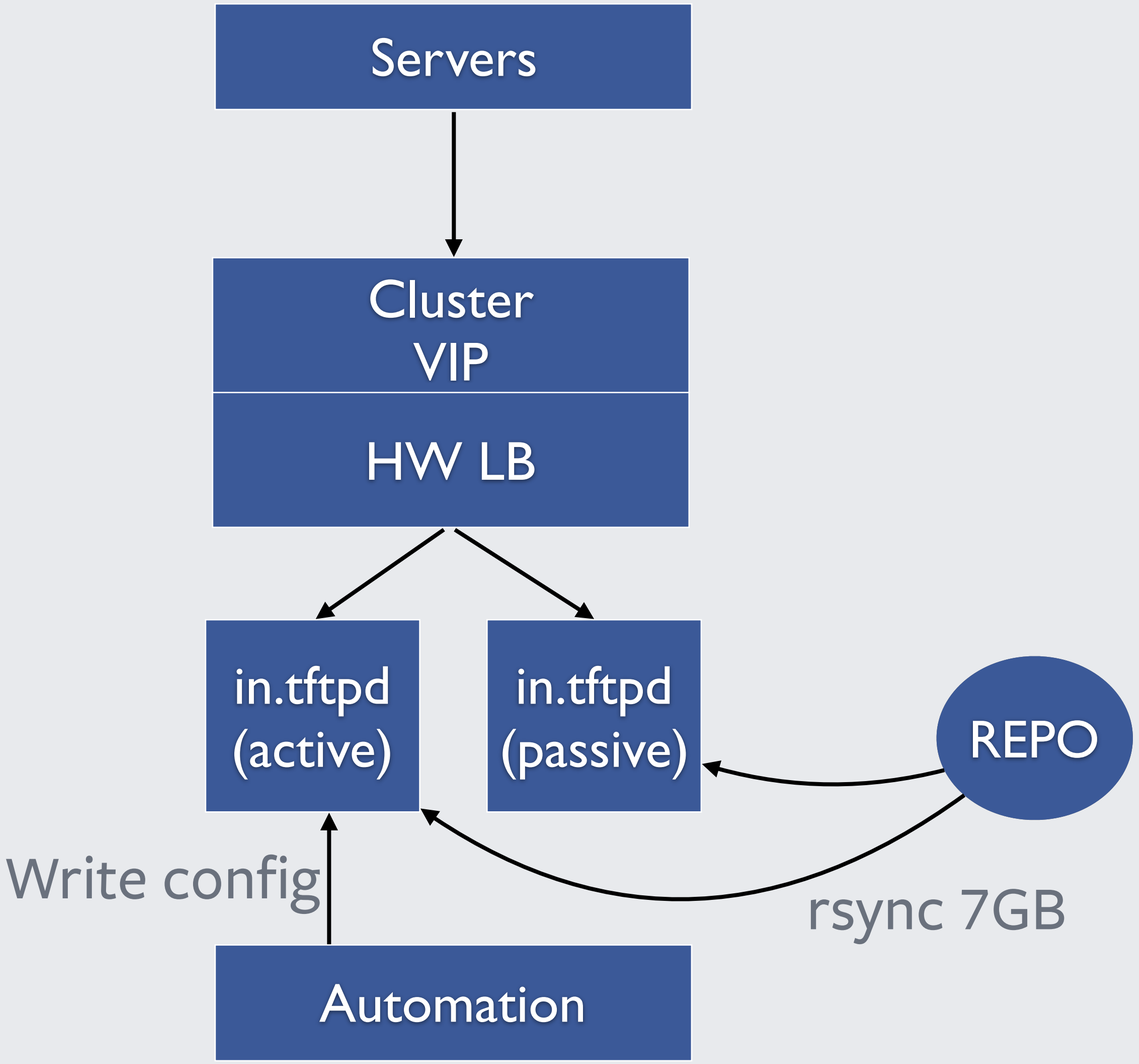
DC

Latency: ~150ms

POPs locations are fictional

A look in the past ~2014 (and its problems)

- Physical load balancers
- Waste of resources
- Automation needs to know which server is active
- No stats
- TFTP is a bad protocol in high latency environments
- Too many moving parts



How did we solve those problems?

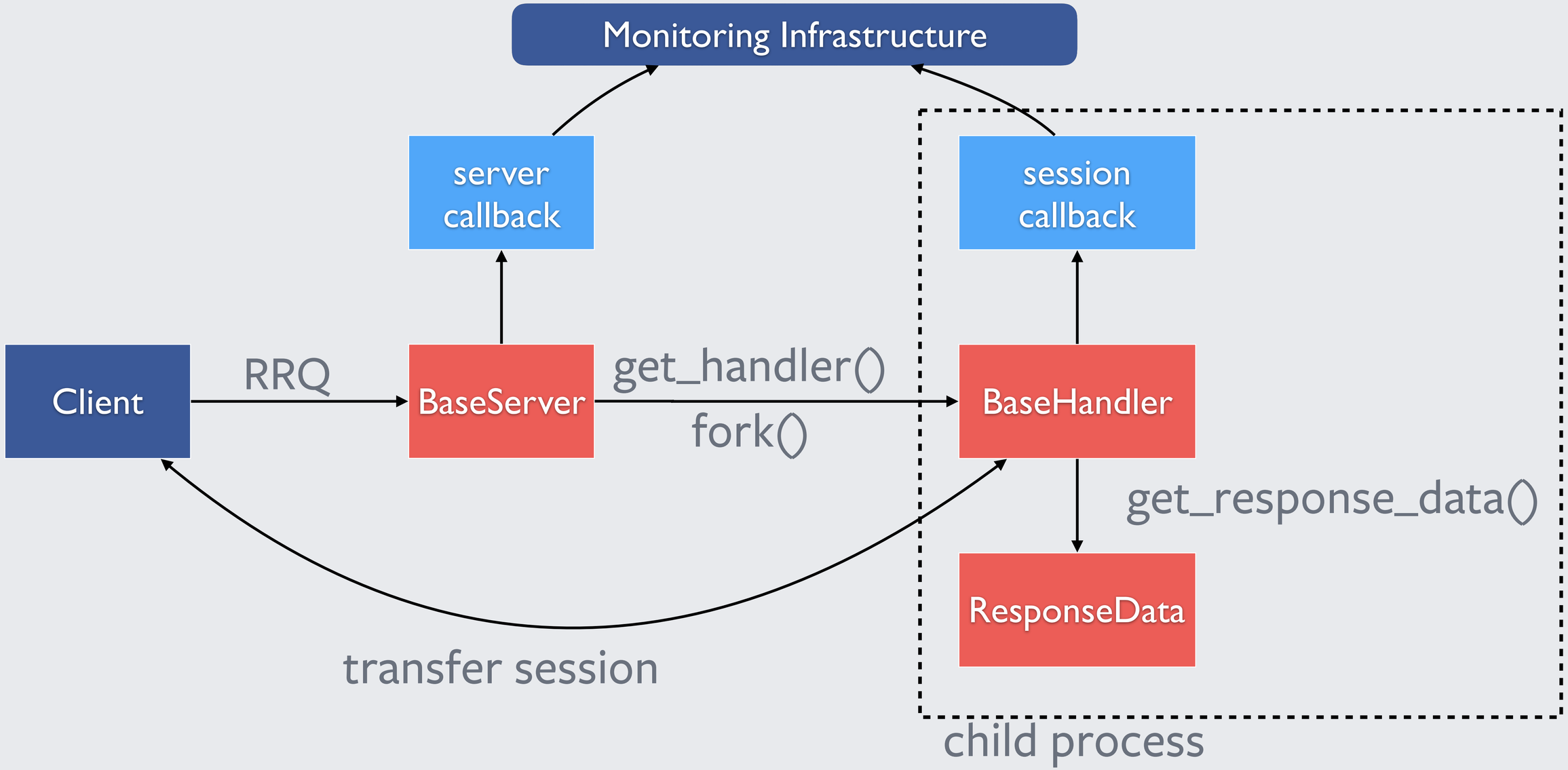
We built FBTFTP...

...A python3 framework to build dynamic TFTP servers

- Supports only RRQ (fetch operation)
  - Main TFTP spec[1], Option Extension[2], Block size option[3], Timeout Interval and Transfer Size Options[4].
- Extensible:
  - Define your own logic
  - Push your own statistics (per session or global)

[1] RFC1350, [2] RFC2347, [3] RFC2348, [4] RFC2349

Framework overview

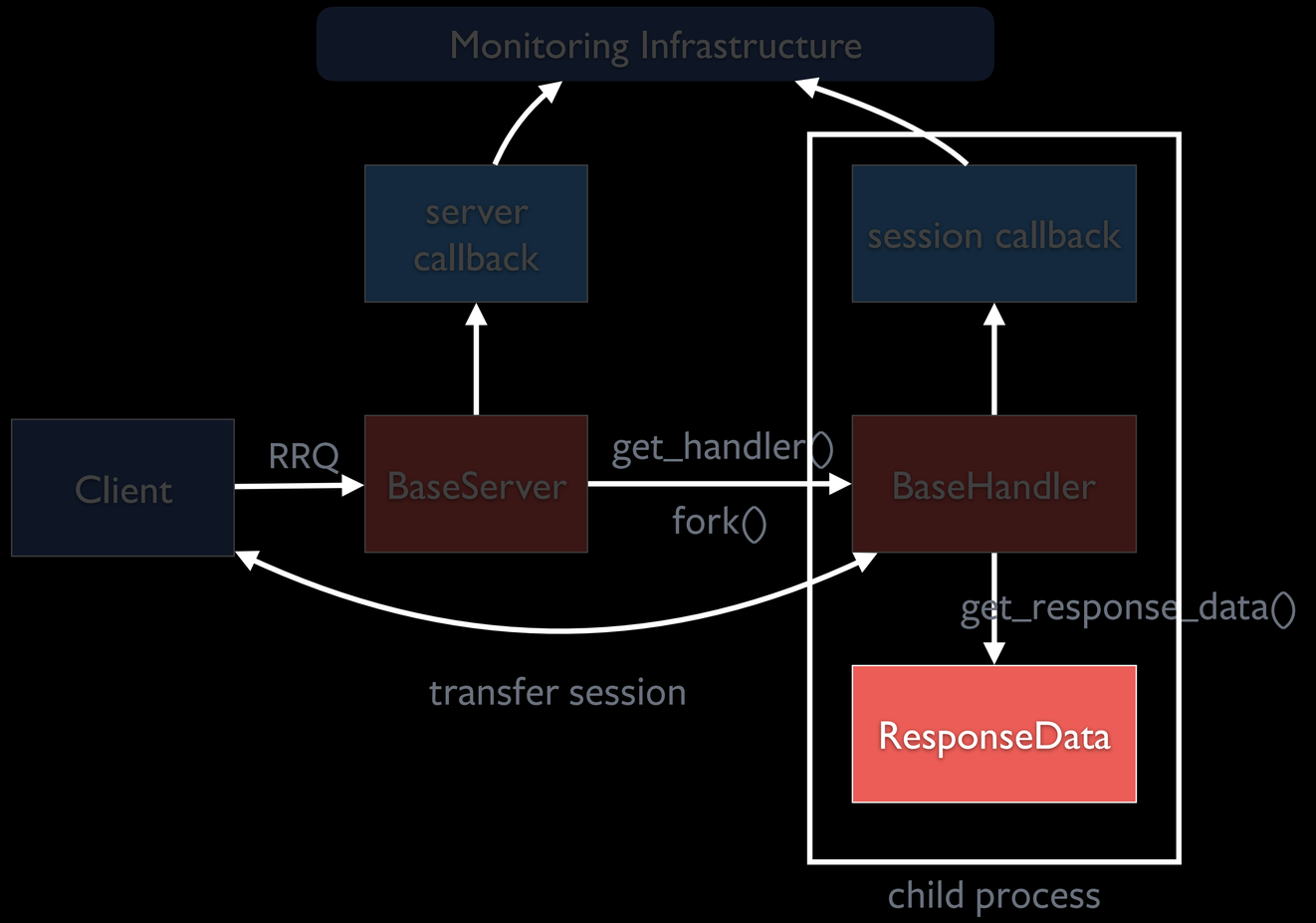


Example:

a simple server serving files from disk

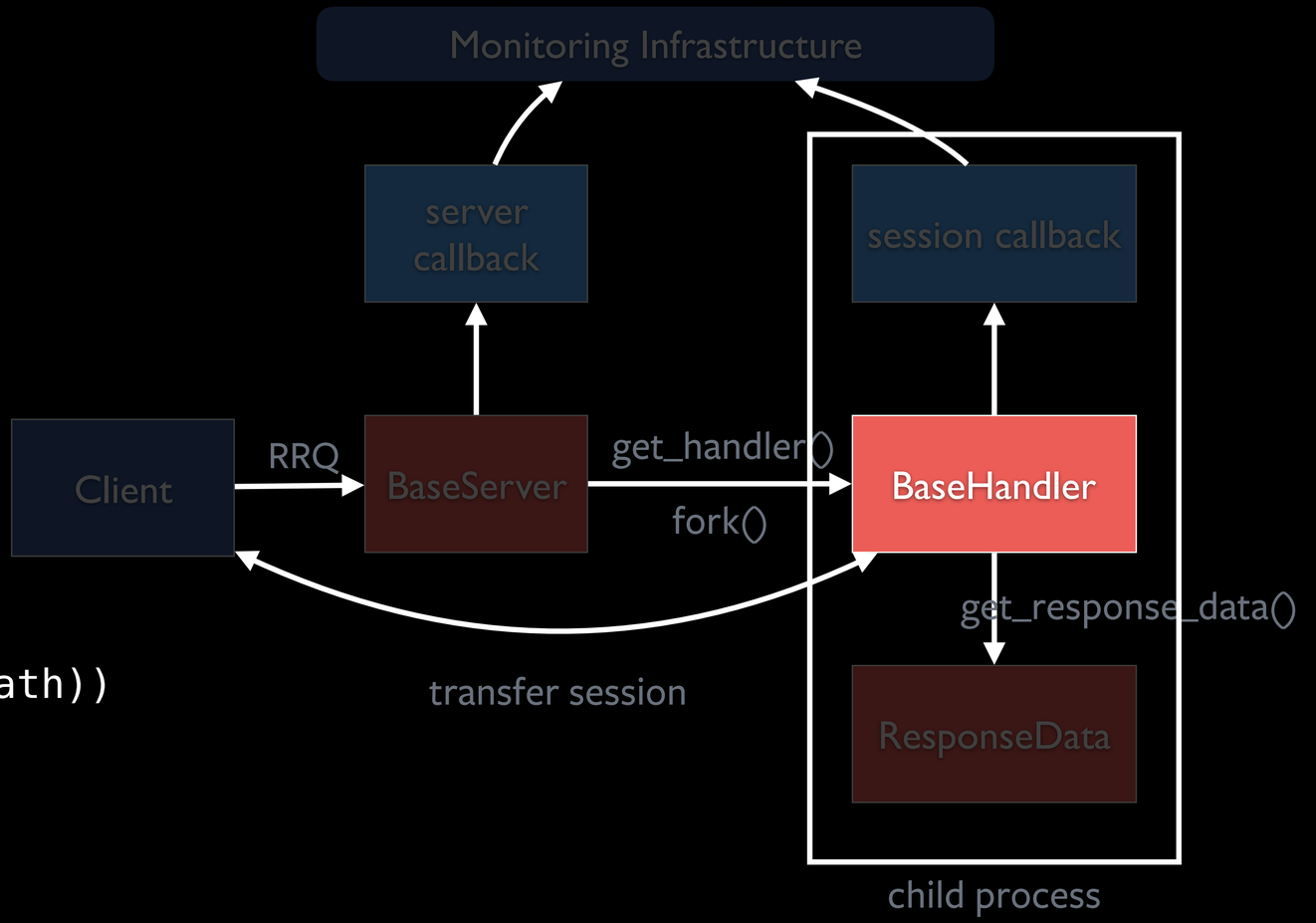
A file-like class that represents a file served:

```
class FileResponseData(ResponseData):
    def __init__(self, path):
        self._size = os.stat(path).st_size
        self._reader = open(path, 'rb')

    def read(self, n):
        return self._reader.read(n)

    def size(self):
        return self._size

    def close(self):
        self._reader.close()
```



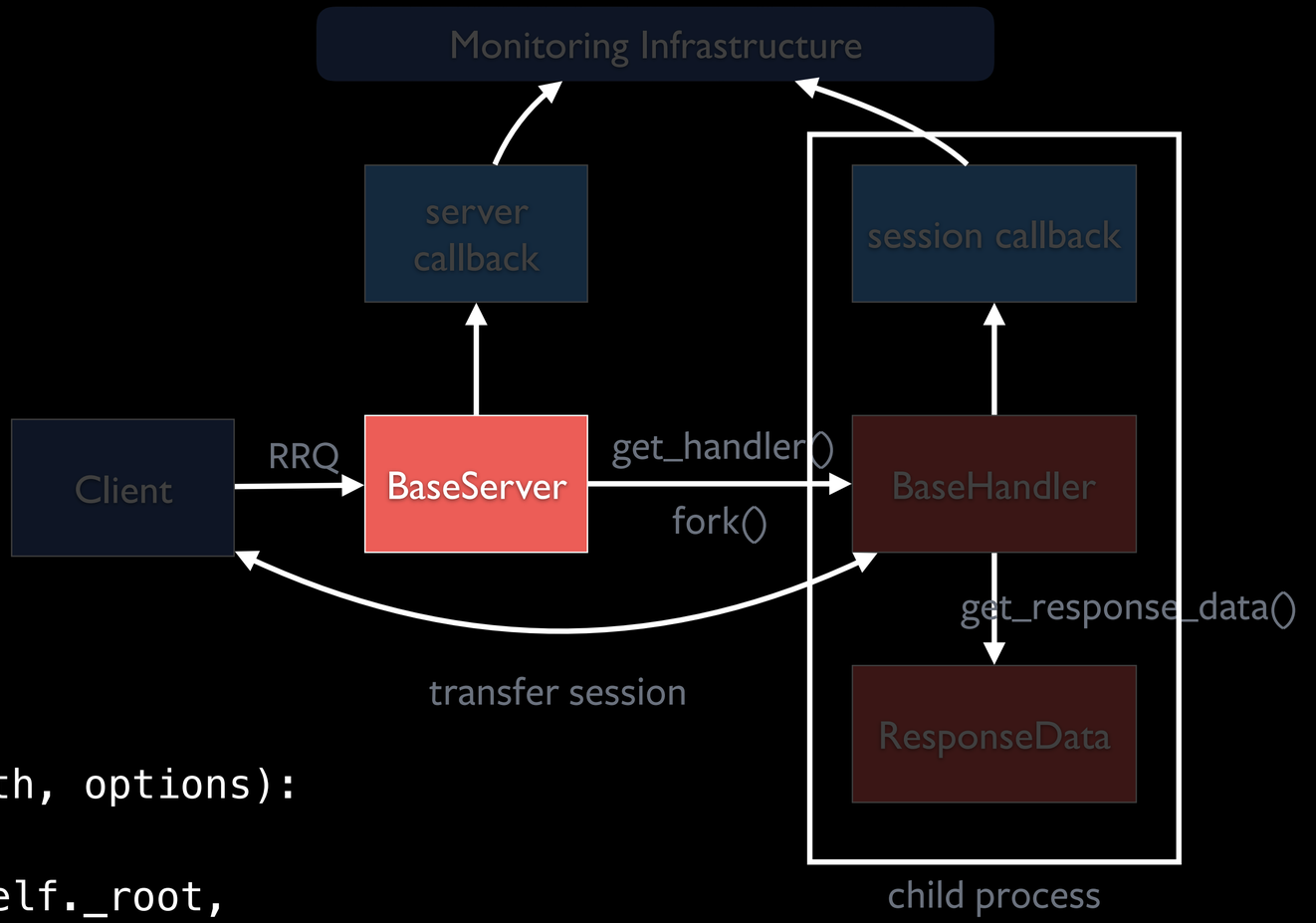
A class that deals with a transfer session:

```
class StaticHandler(BaseHandler):
    def __init__(self, server_addr, peer, path,
                 options, root, stats_callback):
        super().__init__(
            server_addr, peer, path,
            options, stats_callback)
        self._root = root
        self._path = path

    def get_response_data(self):
        return FileResponseData(
                os.path.join(self._root, self._path))
```



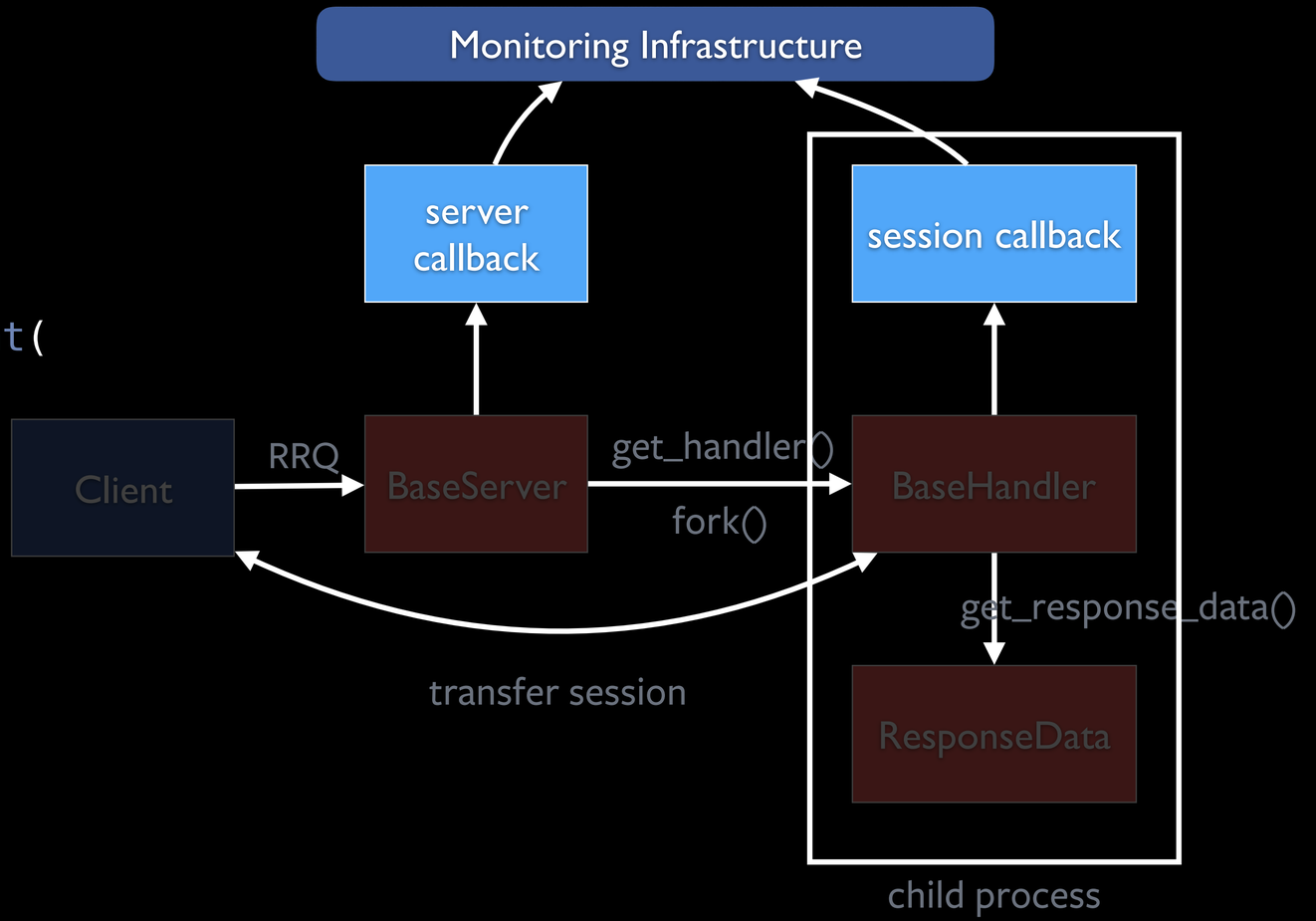
BaseServer class ties everything together:

```
class StaticServer(BaseServer):
    def __init__(
        self, address, port, retries, timeout,
        root, handler_stats_callback,
        server_stats_callback
    ):
        self._root = root
        self._handler_stats_callback = \
            handler_stats_callback
        super().__init__(
            address, port, retries, timeout,
            server_stats_callback)

    def get_handler(self, server_addr, peer, path, options):
        return StaticHandler(
            server_addr, peer, path, options, self._root,
            self._handler_stats_callback)
```

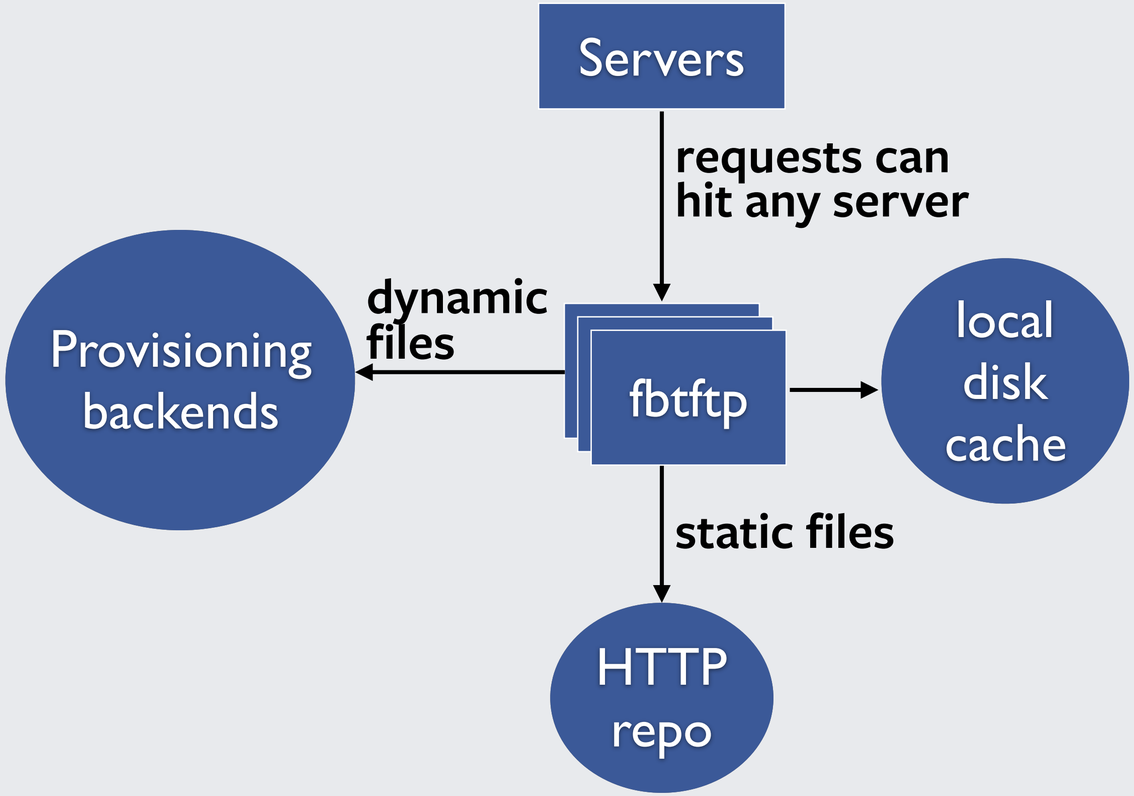


The “main”

```
def print_session_stats(stats):
    print(stats)

def print_server_stats(stats):
    counters = stats.get_and_reset_all_counters()
    print('Server stats - every {} seconds'.format(
          stats.interval))
    print(counters)

server = StaticServer(
    ip='', port='69', retries=3, timeout=5,
    root='/var/tftproot/', print_session_stats,
    print_server_stats)

try:
    server.run()
except KeyboardInterrupt:
    server.close()
```



How do we use it?

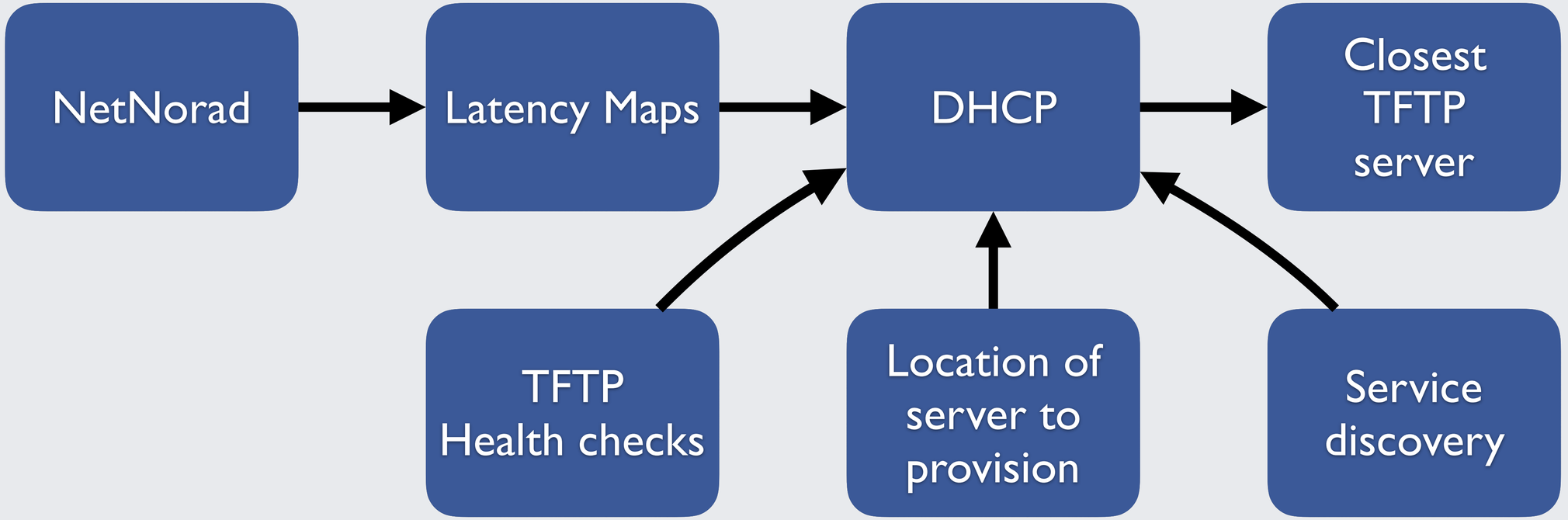


Improvements

- No more physical LBs
- No waste of resources
- Stats!
- TFTP servers are dynamic
  - Config files (e.g. grub/ipxe configs) are generated
  - Static files are streamed
  - You can hit any server
- No need to rsync data
- Container-friendly

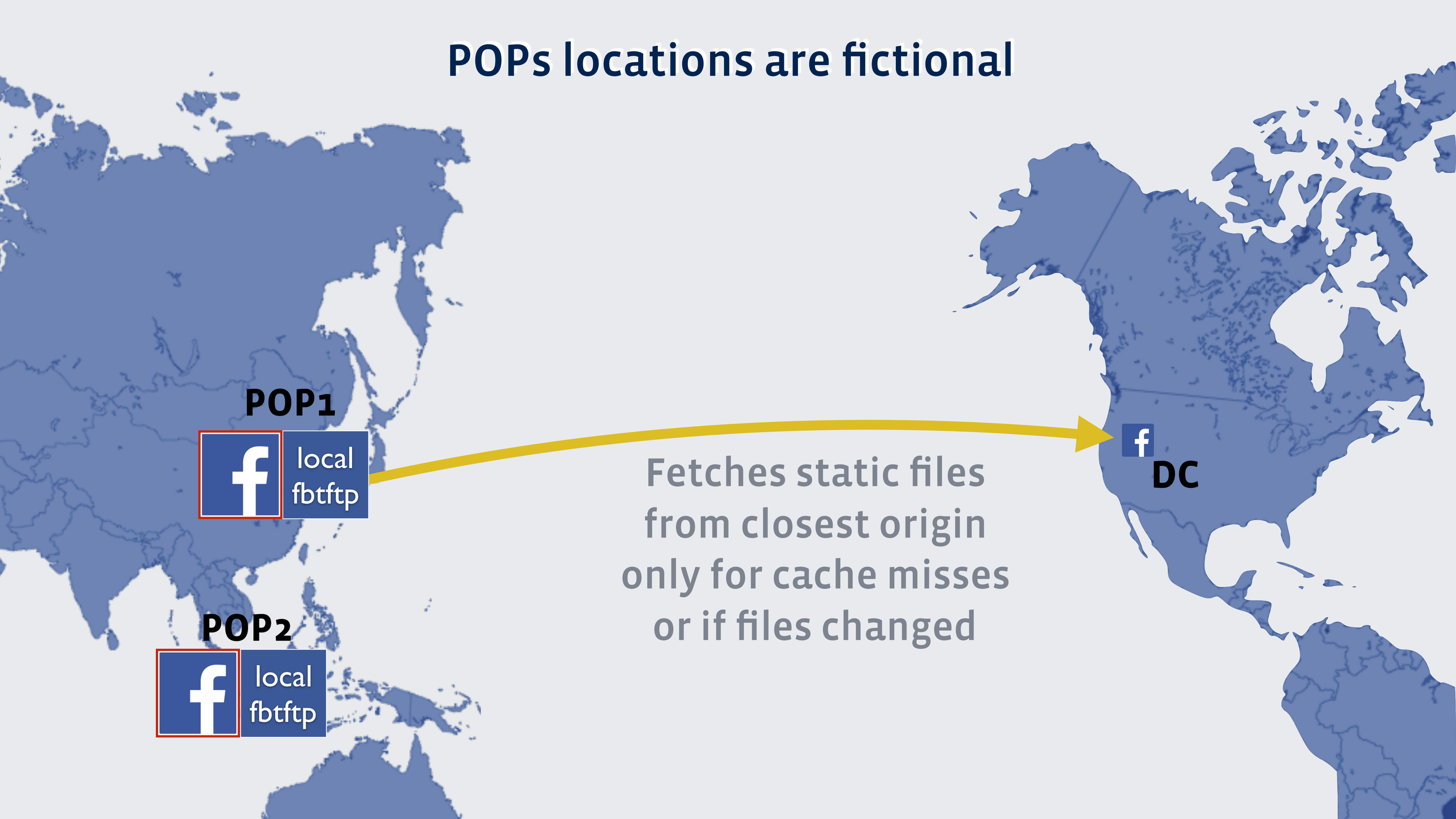
Routing TFTP traffic

LBs are gone: which TFTP server will serve a given client?
NetNorad publishes latency maps periodically, DHCP consumes it.



Read about NetNorad on our blog: http://tinyurl.com/hacrw7c

POPs locations are fictional



Thanks for listening!

Project home:
https://github.com/facebook/fbtftp/

Install and play with it:

```
$ pip3 install fbtftp
```

Poster session Tuesday at 14.45:
Python in Production Engineering

Feel free to email me at pallotron@fb.com