Kolmo

Generic read/writable configuration infrastructure for automation

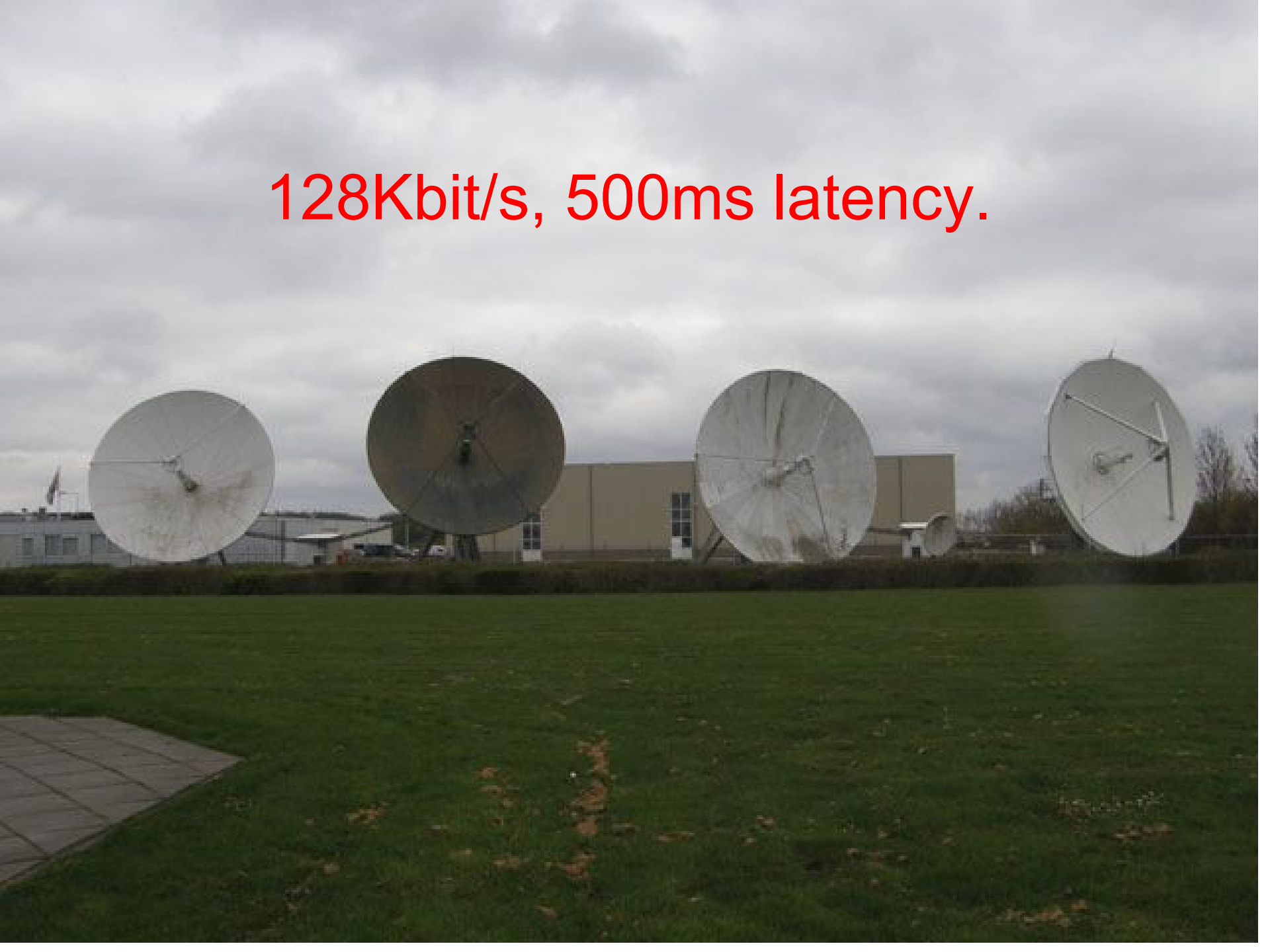
bert hubert
UKNOF38 - Sheffield

https://kolmo.org/ - http://tinyurl.com/kolmopreso

@PowerDNS_Bert

POWERDNS
AN OX COMPANY


UNHCR
The UN Refugee Agency


128Kbit/s, 500ms latency.

The Wonder Shaper

bert hubert <ahu@ds9a.nl>

Originally part of the Linux Advanced Routing & Shaping HOWTO

Before, without wondershaper, while uploading:
round-trip min/avg/max = 2041.4/2332.1/2427.6 ms
After, with wondershaper, during 220kbit/s upload:
round-trip min/avg/max = 15.7/51.8/79.9 ms

* Download version 1.1a, released 16th of April 2002.
* Download version 1.0, released 5th of March 2002.

Works on Linux 2.4 & higher.

If you get an error in the last two lines of the script, try this version of iproute instead: ftp://ftp.inr.ac.ru/ip-routing/iproute2-2.4.7-now-ss010824.tar.gz.

UN Mission: Satellite links & VoIP

5 locations, geostationary satellite links
600ms latency! Full mesh network, 4 transmitters

Linux IP^3 machines for TCP/IP trickery, 'IPMAX'
VoIP, HTTP & SMTP all over a single 128kbit conne
ction

Requirements:

- As many phonecalls as can fit
- Mail should come in speedily
- Webbrowsing should work
  - Exchange Webmail in Brussels

Even Linux is not magic, but..

Network was broken by design and the requirements where conflicting.

However, following tricks helped:

* Removal of Cisco cruft
* 5 additional Linux machines
  - Iptables MSS clamp
  - Agressive Queuecontrol
  - Shaping (CBQ+TBF)
  - Iptables driven MRTG
* Adjustment of expectations

Bufferbloat

From Wikipedia, the free encyclopedia

Bufferbloat is high latency in packet-switched networks caused by excess buffering of packets. Bufferbloat can also cause packet delay variation (also known as jitter), as well as reduce the overall network throughput. When a router or switch is configured to use excessively large buffers, even very high-speed networks can become practically unusable for many interactive applications like Voice over IP (VoIP), online gaming, and even ordinary web surfing.

Some communications equipment manufacturers placed overly large buffers in some of their network products. In such equipment, bufferbloat occurs when a network link becomes congested, causing packets to become queued in buffers for too long. In a first-in first-out queuing system, overly large buffers result in longer queues and higher latency, and do not improve network throughput.

The bufferbloat phenomenon was initially described as far back as in 1985.[1] It gained more widespread attention starting in 2009.[2]

Configuration

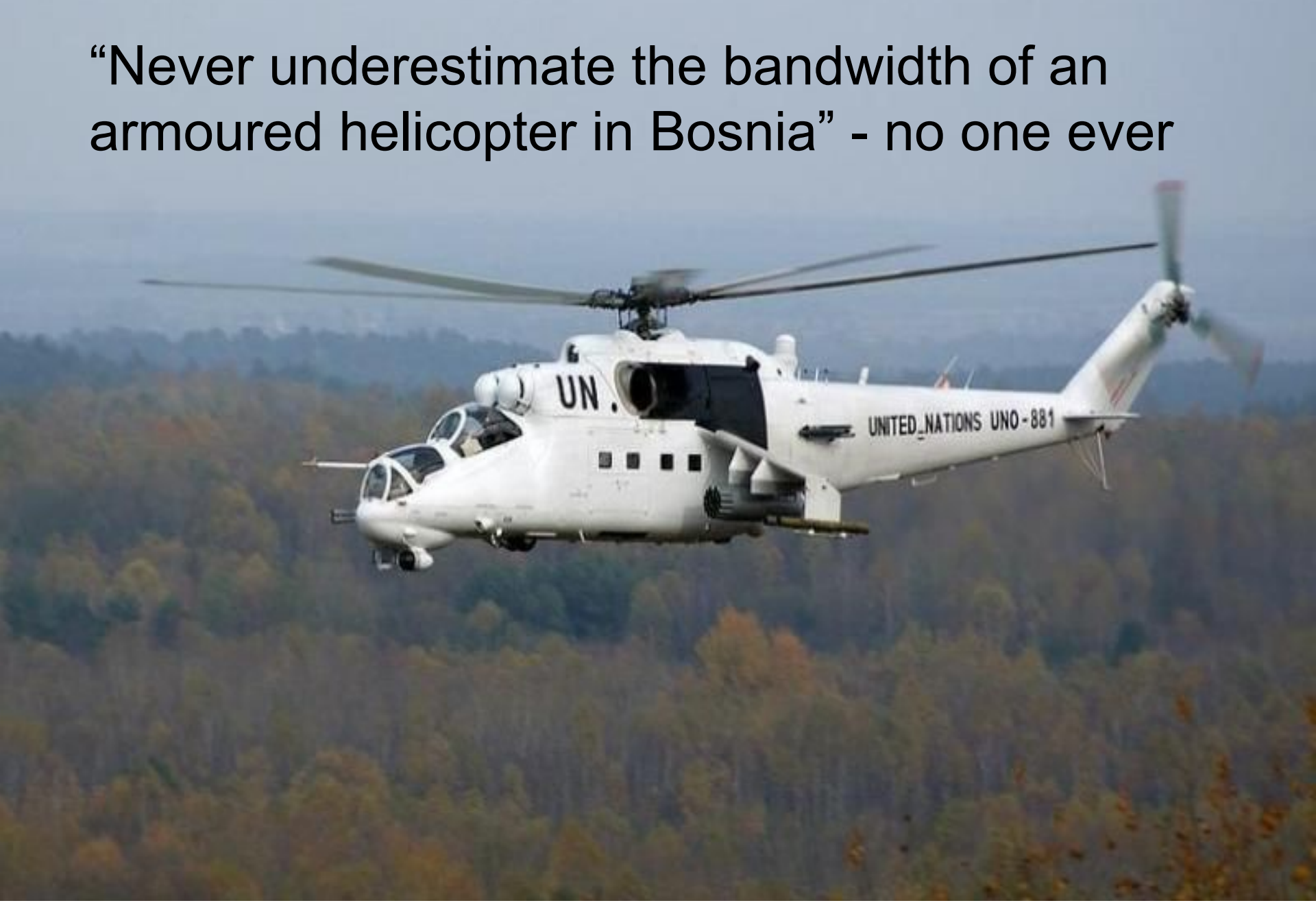
- Make runtime/boot time changes to:
  - iptables
  - ifconfig txqueuelen
  - /proc/sys/
  - bridge (brctl)
  - Actual QoS (tc, htb, cbq)

- ALL OF THESE could not be serialized
  - Pretty output is available - for humans
  - Most of them to this day can’t be saved/serialized
  - Exception: iptables-save / iptables-restore

You can’t download
the configuration?!?!

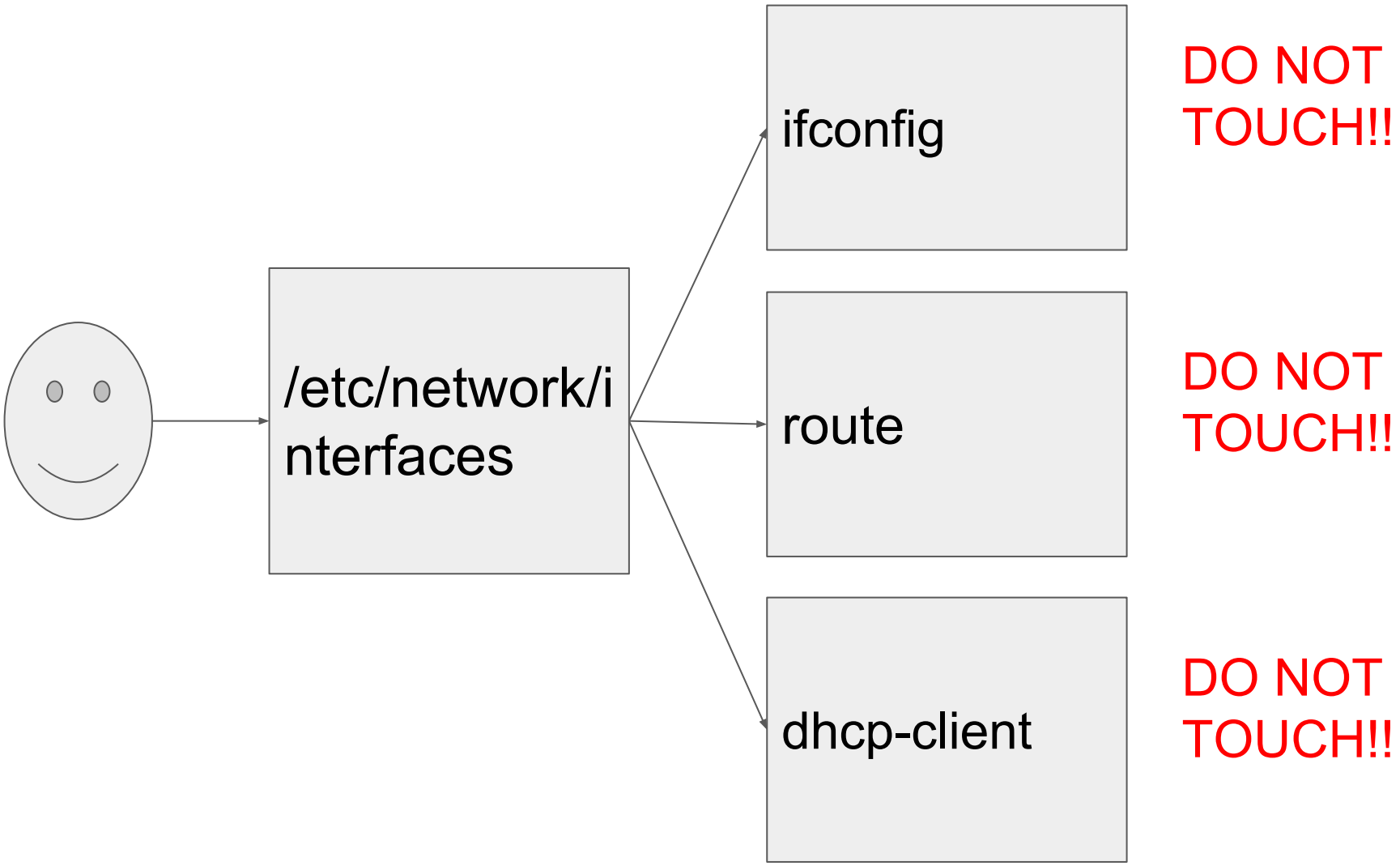
You can’t even commit
the configuration?!?!

“Never underestimate the bandwidth of an armoured helicopter in Bosnia” - no one ever

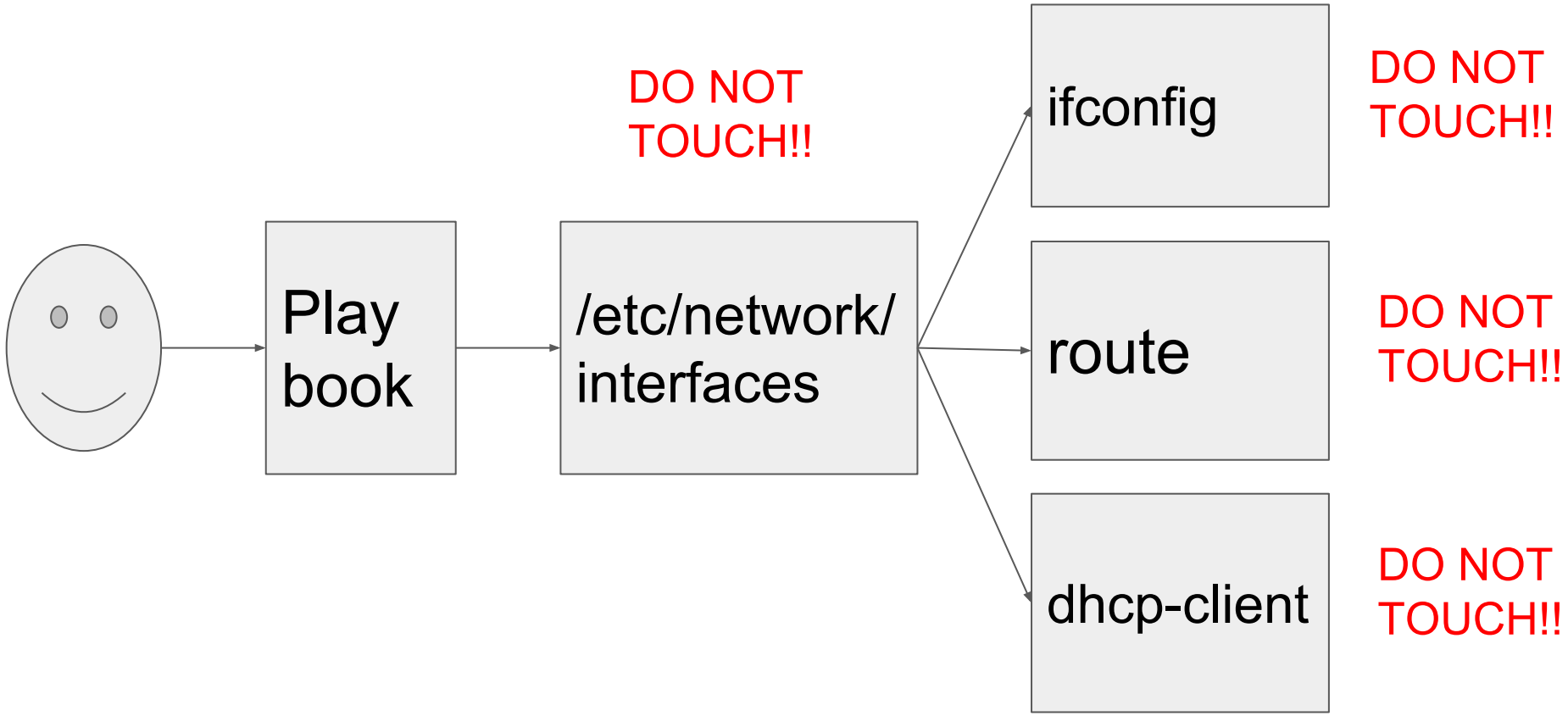


Central problem: You can set server/system state. But not retrieve it. Let alone find out what you changed!

2000 solution: generation



2014 solution: generation



Automation workflow

1. Create playbook
2. Run playbook on servers
3. Modify playbook bit in response to new needs
4. Deploy to servers
5. Happiness

Lies!

Actual automation workflow

1. Create playbook, run playbook on server
2. **Server does nothing like you want it to**
3. Inspect server
4. Find Ansible ‘lineinfile’ triggers on a comment block
5. Change playbook, rerun
6. Find further problems, change playbook, rerun
7. Webserver does not do what you want, work on server to figure out how to get it to do what you want
8. Attempt to put those changes in playbook, finally it works
9. **Rerun playbook against fresh server: nothing works**

Automation dream workflow

- Deploy fresh server
- Configure everything, on that server, until it is exactly how you like it
- **Download server state *delta***
- Insert *delta* state in playbook
- **Happiness**

Why it can’t be done “after the fact”

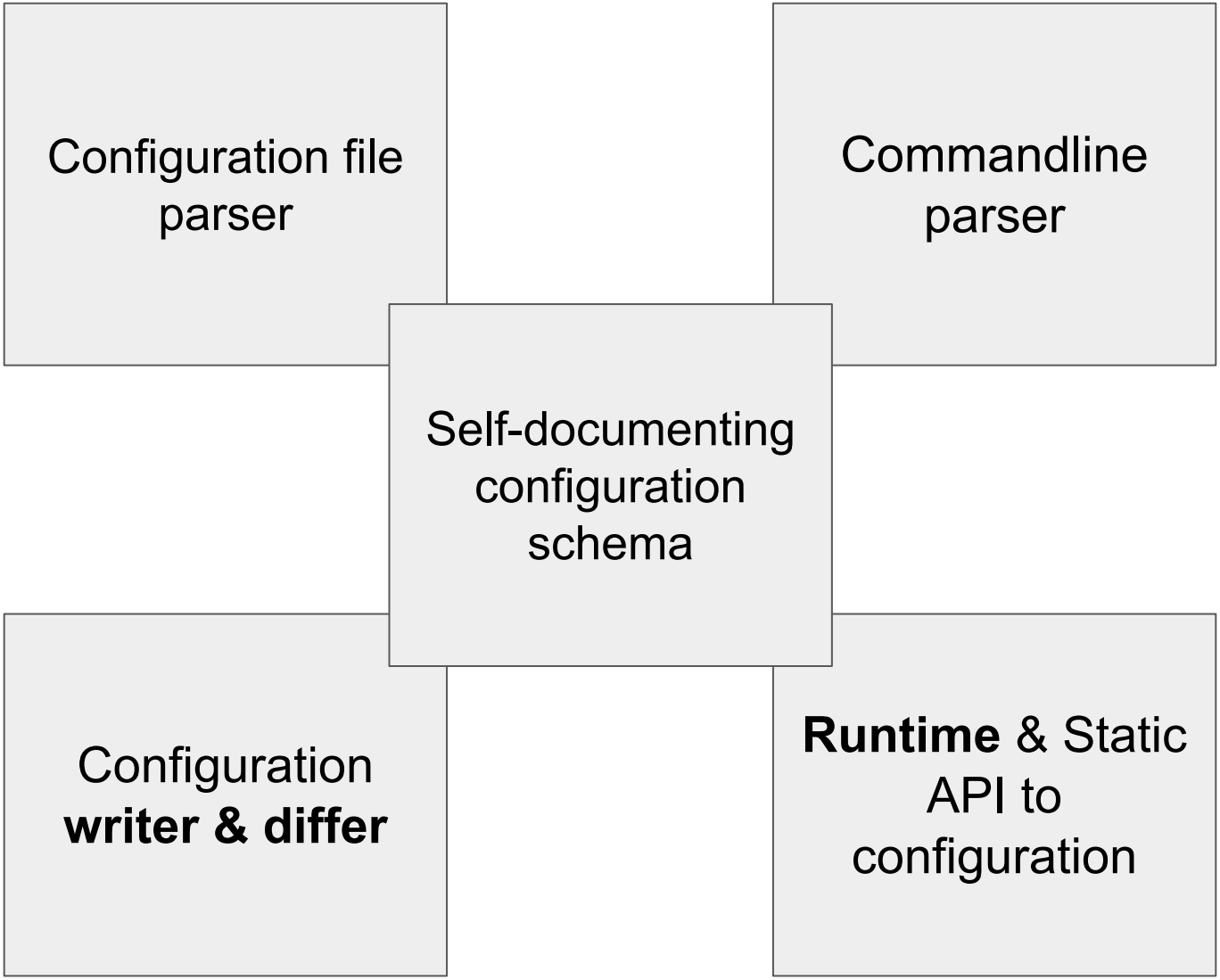
- A typical process if configured using several configuration files
- Some of them under distribution/operating system control
- Some of them explicitly meant for you to change them
  - Include.d directories, sites-enabled, sites-available
- Some of them are TEH HUGE and actually contain large parts of the software logic (Exim)
- There is no easy way to “extract your changes” from this set of configuration files
- There are only very painful ways to “insert your changes” into existing configuration files
  - (sometimes there is an ansible module that helps)

Introducing: Kolmo

Inspect, retrieve, modify & deploy configuration safely

- Programmatic access to configuration
  - Read/Write
- Typesafe configuration schema with internal constraints
- Self-documenting
- Client-side, offline, validity checking
- Meant to ease “automation”
- **Library & tools to support all kinds of systems/services**

Kolmo


Configuration file parser
Commandline parser
Self-documenting configuration schema
Configuration **writer & differ**
Runtime & Static API to configuration

The Schema: Welcome to the default free zone

- Defines all configuration settings
  - Type (YES IT IS TYPESAFE)
    - String, IP address, netmask, MAC address..
  - Defaults (ALL defaults)
  - Mandatory / optional
  - **DESCRIPTION**
  - **UNITS**
  - **CONSTRAINTS**

- If nothing else, the configuration schema is GREAT documentation of your configuration file and all defaults!
  - **“The problem with documentation is that the compiler does not read it”**

“Ws”: the Kolmo “Hello, World” application

(that powers https://kolmo.org/)

```
main:registerVariable("verbose", "bool", {
   default="true", runtime="true", cmdline="-v",
   description="Perform verbose logging"})

main:registerVariable("server-name", "string",  {
   default="", runtime="true",
   description="Name this server reports as by default"})

main:registerVariable("client-timeout", "integer", {
   default="5000",
   runtime="true",
   unit="milliseconds",
   description="Timeout before client gets disconnected",
   check=
   'if(x < 1) then error
   ("Timeout must be at least one millisecond") end'
})
```

```
site=createClass("site", "A site we serve")
site:registerVariable("name", "string", {
    runtime="false",
    description="Hostname of this website"
  })
site:registerVariable("enabled", "bool", {
  runtime="false",
  default="true",
  description="If this site is enabled"
  })
site:registerVariable("path", "string", {
  runtime="true",
  description="Path on fs where content is"
  })
```

```
$ alias wsctl='./kolctl --config=ws.json
  --schema=ws-schema.lua'

$ wsctl ls

carbon-server                    Send performance metrics to this IP
client-timeout       5000        Timeout before client gets disconnected
hide-server-type     true        If we should hide server type
hide-server-version  false       If we should hide server version number
kolmo-server         127.0.0.1:1234 If we should launch a kolmo server
listeners            {struct}    Optional configurations per IP address
loggers              {struct}    Loggers that log events and hits
max-connections      200         Maximum number of connections
server-name          kolmo.org   Name this server reports as by default
sites                {struct}    Sites we serve
verbose              false       Perform verbose logging
```

```
$ wsctl minimal-config
{}

$ wsctl full-config
{
    "carbon-server": "", "client-timeout": 5000,
    "hide-server-type": false,
    "hide-server-version": false,
    "kolmo-server": "127.0.0.1:1234",
    "listeners": {},
    "loggers": {
        "messages": {
            "log-errors": true,  "log-file": "",
            "log-hits": false, "log-warning": true,
            "syslog": true,"syslog-facility": "daemon"
        }
    },
    "max-connections": 200, "server-name": "",
    "sites": {}, "verbose": true
}
```

```
$ wsctl set verbose=true
$ wsctl minimal-config
{}

$ wsctl set verbose=false
$ wsctl minimal-config
{
    "verbose": false
}

$ wsctl set client-timeout=4000
$ wsctl minimal-config
{
    "client-timeout": 4000,
    "verbose": false
}
```

```
$ wsctl set server-name='kolmo.org'

$ ls -l ws.json ws.json.20170911-2*
l 23 sep 11 22:10 ws.json -> ws.json.20170911-221017
-rw-rw-r-- 52 sep 11 21:55   ws.json.20170911-215551
-rw-rw-r-- 84 sep 11 22:10   ws.json.20170911-221017

$ cat ws.json
{
    "client-timeout": 4000,
    "server-name": "kolmo.org",
    "verbose": false
}
$ diff -uBb ws.json.20170911-215551 ws.json
--- ws.json.20170911-215551
+++ ws.json     2017-09-11 22:10:17.568626625 +0200
@@ -1,4 +1,5 @@
 {
     "client-timeout": 4000,
+    "server-name": "kolmo.org",
     "verbose": false
 }
```

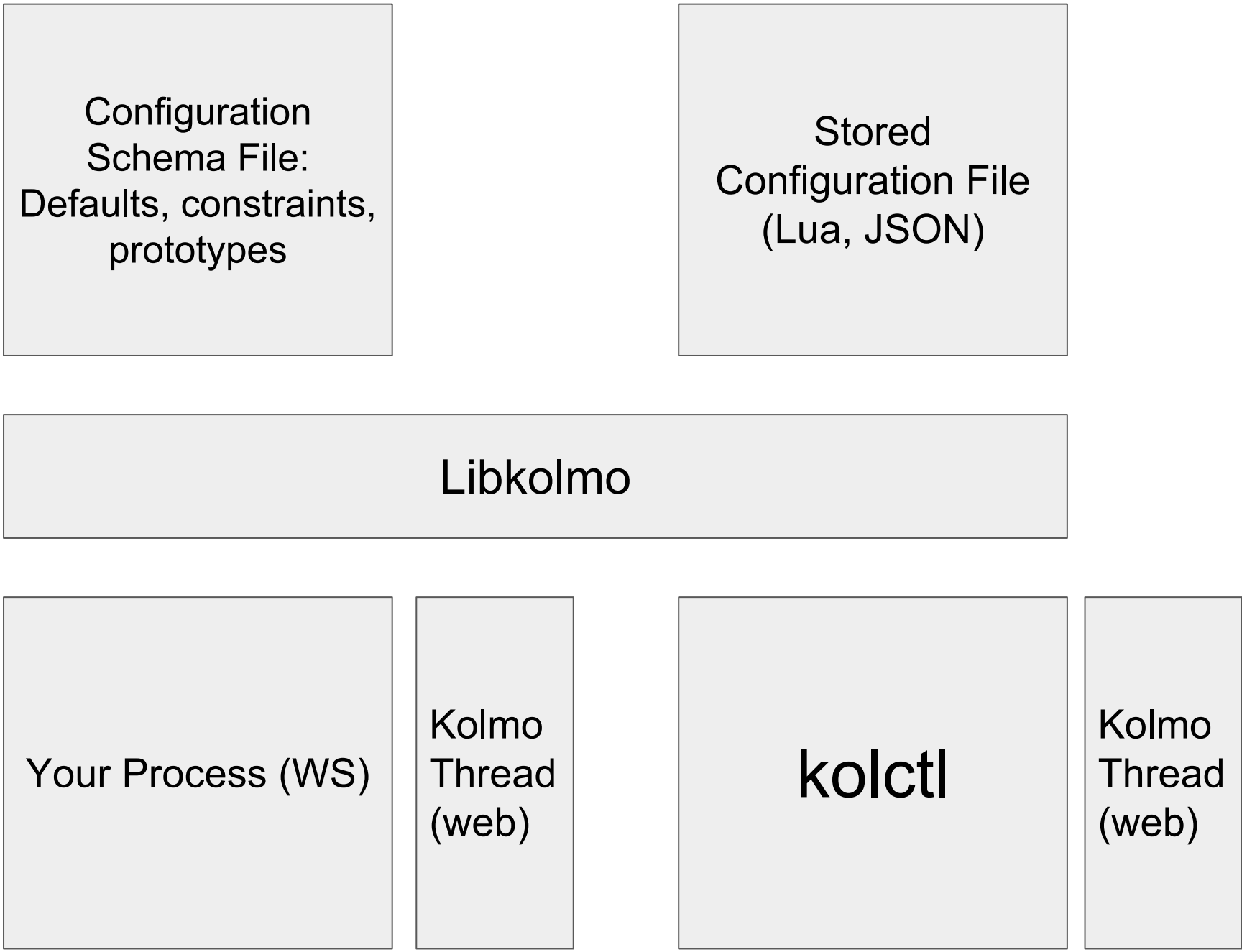
```
$ wsctl add sites kolmo '{"name": "kolmo.org",
  "path":"/var/www/kolmo.org"}'

$ wsctl add sites/kolmo/listen "[::]:8000"

$ wsctl ls sites/kolmo
enabled           true                If this site is enabled
listen            {struct}            IP endpoints we listen on
name              kolmo.org           Hostname of this website
path              /var/www/kolmo.org  Path on fs where content is
redirect-to-https false               redirected to https

$ wsctl minimal-config
{
    "client-timeout": 4000, "server-name": "kolmo.org",
    "sites": {
        "kolmo": {
            "listen": {"0": "[::]:8000"
            },
            "name": "kolmo.org", "path": "/var/www/kolmo.org"
        }
    },
    "verbose": false
}
```

Runtime & Constraints


Configuration Schema File: Defaults, constraints, prototypes
Stored Configuration File (Lua, JSON)
Libkolmo
Your Process (WS)
Kolmo Thread (web)
kolctl
Kolmo Thread (web)

```
$ ./ws &
Verbose is false
[kolmo] We run a website called kolmo.org
The site enable status: 1
We serve from path: /var/www/kolmo.org
We serve on addresses: [::]:8000

Need to listen on 1 addresses

$ alias wsctl='./kolctl -r http://127.0.0.1:1234'

$ wsctl ls
carbon-server                    Send performance metrics
client-timeout       4000        Timeout before client ge
hide-server-type     false       If we should hide server
hide-server-version  false       If we should hide server
kolmo-server         127.0.0.1:1234 If we should launch a ko
..
```

```
$ wsctl delta-config
{}
$ wsctl set verbose=true

$ wsctl delta-config
{
    "verbose": true
}

$ wsctl set max-connections=300
{"reason":"Attempting to change var at runtime
          that does not support runtime changes",
  "Result":"failure"}

$ wsctl set verbose=MORE
{"reason":"Attempt to set bool to something not
          true or false",
 "result":"failure"}
```

```
$ wsctl set client-timeout=0
{"reason":"Timeout must be at least one ms",
 "result":"failure"}

$ wsctl set carbon-server=1.2.3.4.5:1234
{"reason":"Can't convert address '1.2.3.4.5:1234' ",
 "Result":"failure"}

-----------------------------------------------

main:registerVariable("client-timeout", "integer", {
  (...)
  check=
  'if(x < 1) then error("Timeout must be
          at least one millisecond") end'
})

main:registerVariable("carbon-server", "ipendpoint",
  { (...)
  })
```

$ kolctl --schema=ws-schema.lua markdown

Automatically generated Kolmo configuration for ws

Contents

1 Markdown documentation for ws

1.1 Classes used in configuration

1.1.1 listener

Some usecases

- Webserver is misbehaving “all of a sudden”
  - Run kolctl delta-config on running process: gives all runtime changes versus startup configuration
  - Discover someone helpfully changed the certificate file to the wrong path, at runtime

- Webserver is misbehaving “all of a sudden”
  - “delta-config” shows nothing interesting
  - “ls -l ws.json*” however shows a new configuration was created 5 minutes ago
  - “diff -uBb ws.json.20170908-1245 ws.json” shows client-timeout was set to 1 millisecond
    - By operator who had ignored the helpful “units” output in “kolctl ls”

```
int main(int argc, char** argv)
{
  KolmoConf kc;
  kc.initSchemaFromFile("ws-schema.lua");
  kc.initConfigFromJSON("ws.json");
  kc.declareRuntime();
  kc.initConfigFromCmdline(argc, argv);

  kc.d_main.tieBool("verbose", &g_verbose);

  if(g_verbose) {
    cerr<<"Must be verbose"<<endl;
    cerr<<"Server name is "<<kc.d_main.getString("server-name")<<endl;
  }
  else {
    cerr<<"Verbose is false"<<endl;
  }
```

Relation to Automation

For “Kolmo” enabled services

1. Instead of hacks to manipulate configuration file, execute ‘kolctl’ commands on server

2. Alternatively, deploy the ‘minimal-config’ you got from your Kolmo-enabled service when it was “Just Right” (but mind the ‘migration’ slide)

Summarising

- Kolmo is an API for changing/consulting a system's state/configuration
  - Typesafe with constraints

- HTTP-based access to stored (committed) configuration
  - And to runtime changeable variables

- Keeps track of 'delta' between runtime & committed
- Keeps log of all changes
- Built-in documentation generation from schema
- Free! (MIT licensed)

Status

- Working prototype, running code!
  - https://kolmo.org/ & https://github.com/ahupowerdns/kolmo
- Tremendous fun to work with, concept works for me as a programmer
  - (I'm not ever writing or even USING a configuration file parser ever again)
- Quality of code: this segmentation fault is your hint
- Only available for C++ 2014
- GOAL: Get everyone hot for this concept

Now what?

- Please join in!
- If you are a developer: share your thoughts on the API
- As an automation person: how does this fit in your life? Has it been done?
- As software users: get annoyed that all your favourite servers do not come with a configuration schema file!
  - You'll miss 'wctl ls' from today onwards!

Kolmo

Generic read/writable configuration infrastructure for automation

bert hubert
UKNOF38 - Sheffield

https://kolmo.org/ - http://tinyurl.com/kolmopreso

@PowerDNS_Bert

POWERDNS
AN OX COMPANY