

NetOps Coding 101

...building your first robot!

NANOG 66

02.09.2016

david swafford

facebook

**what we'll
accomplish**

automating

the remediation of network faults

**using the examples of
link flaps &
line card failures**

based on parsing of
syslog messages

focusing today

coding in Python

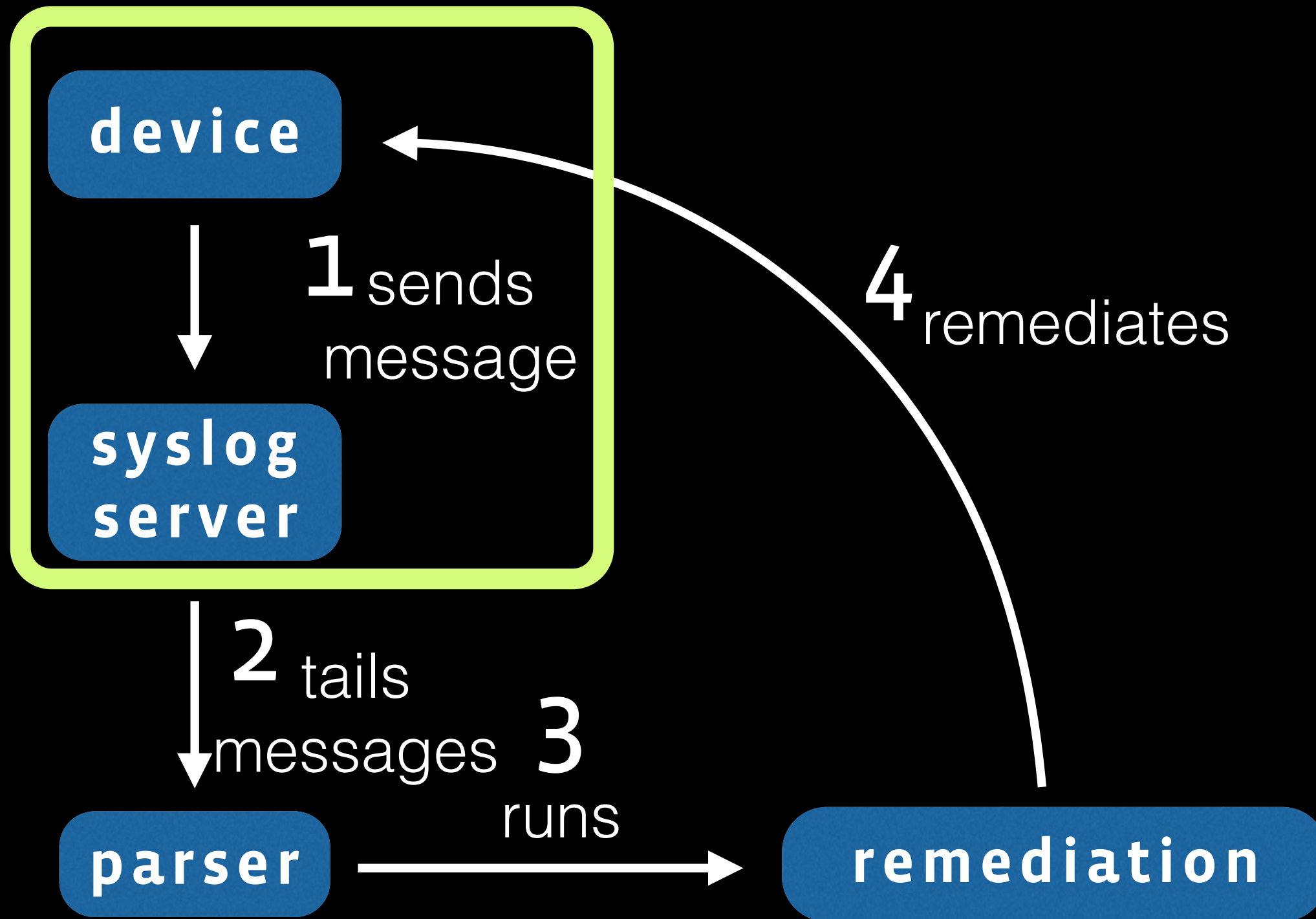
**interacting with
network devices
through software**

regular expressions

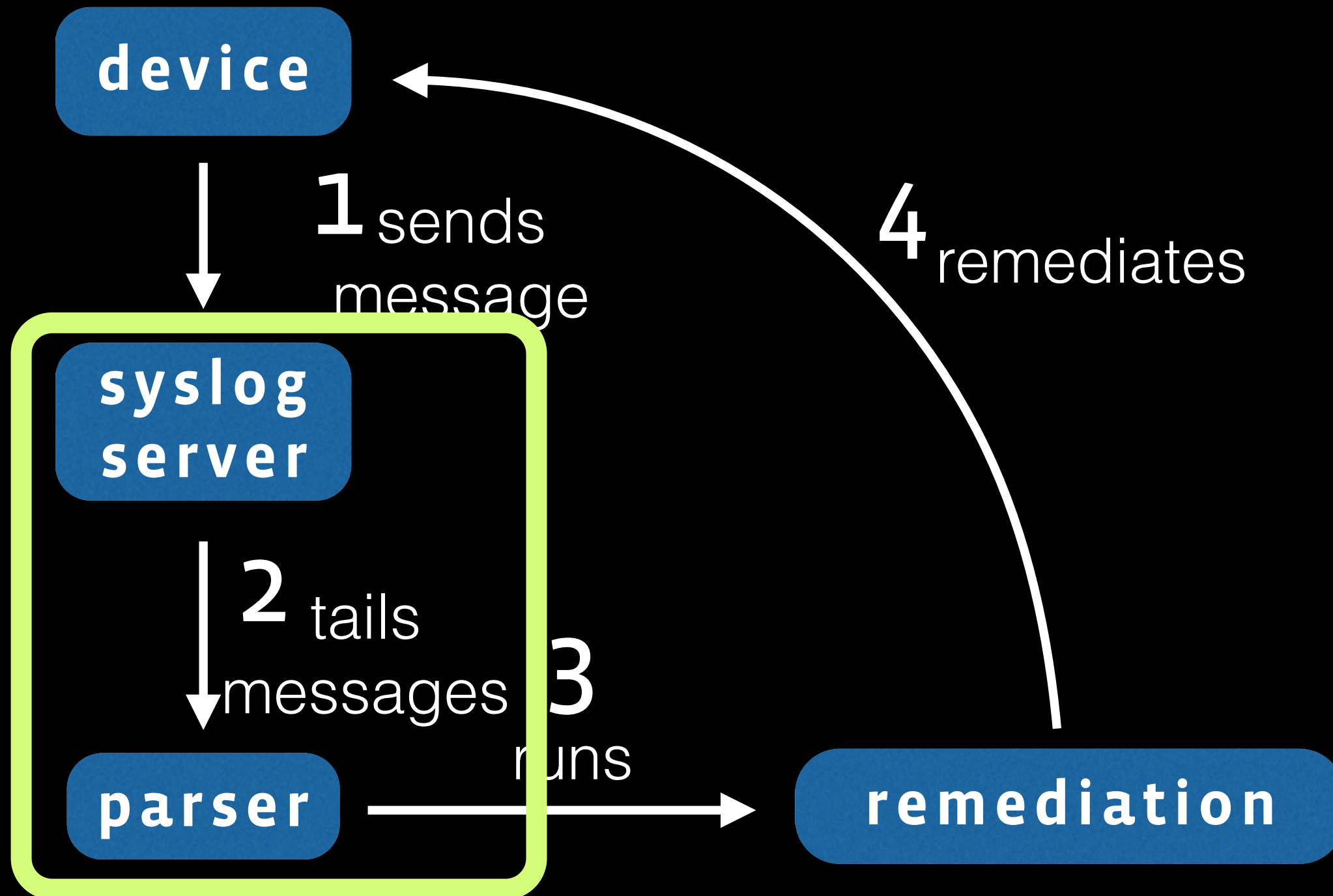
**keeping track of
what happened**

how it works

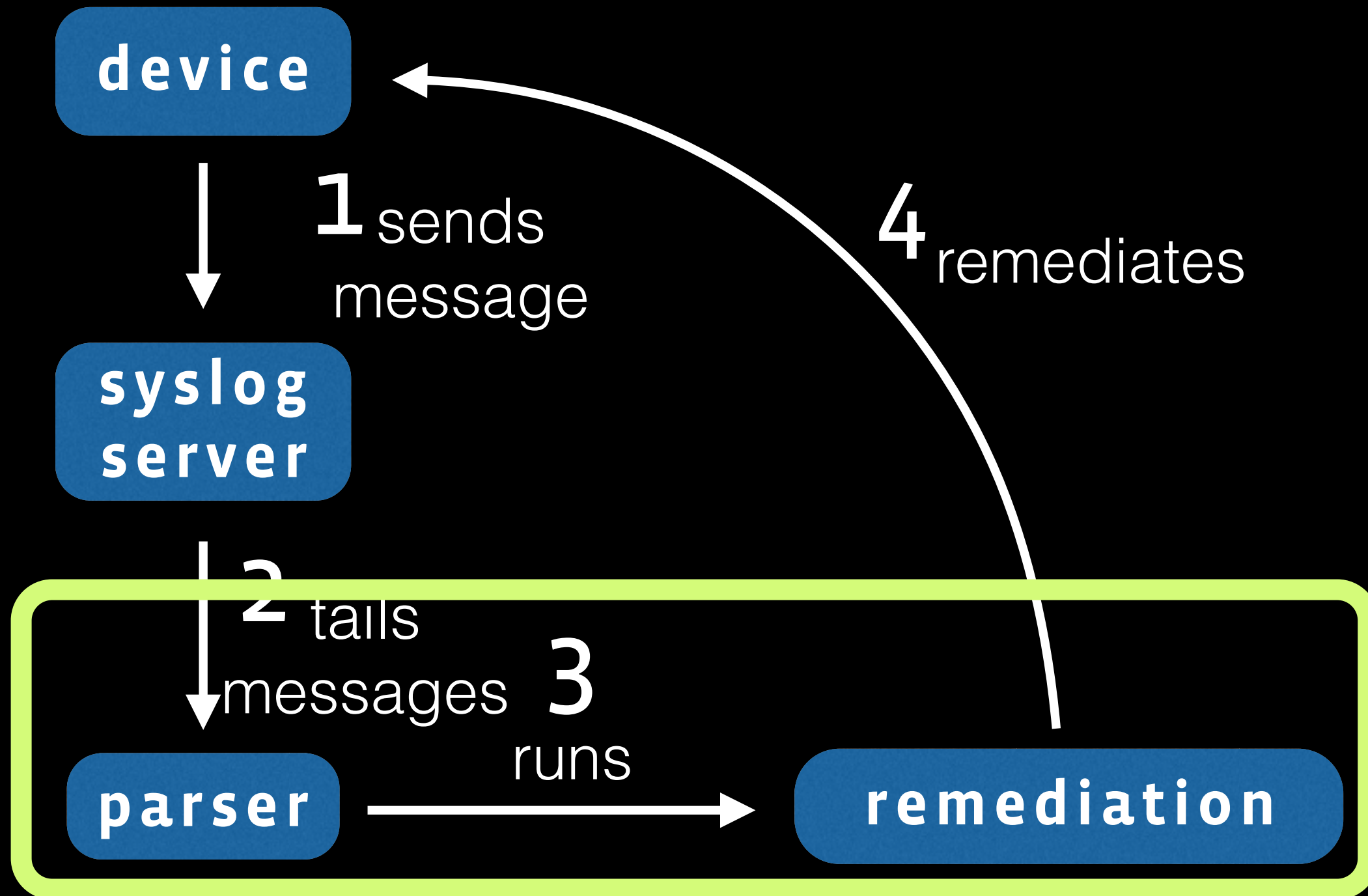
system design



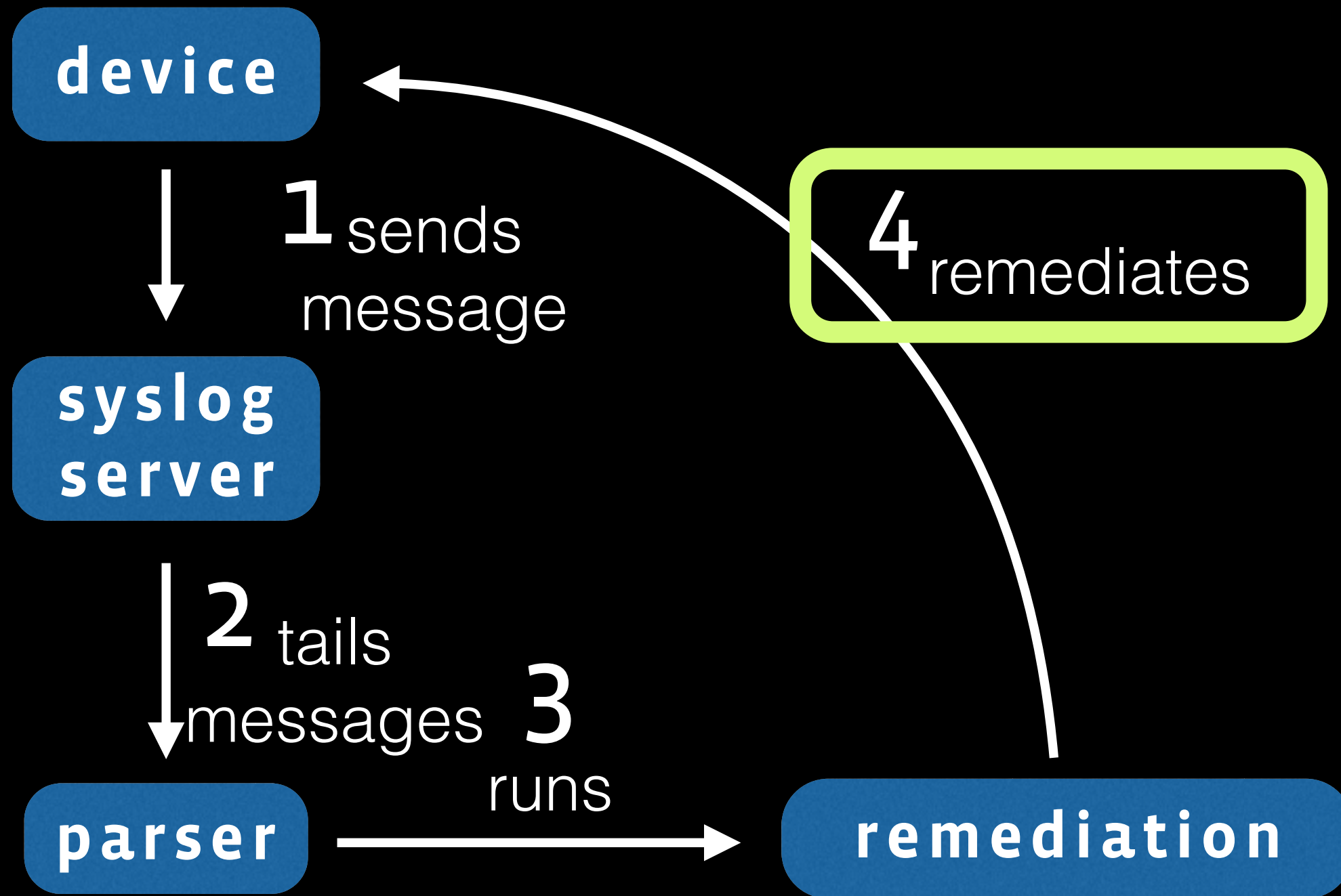
system design



system design



system design



live demo!

```
[2]: run nanog_demo.py
```

```
In [2]: run nanog_demo.py
```

```
.....  
NEW LOG EVENT:
```

```
2015 Apr 2:14:25:06 switch1 ETHPORT-5-IF_DOWN_INTERFACE_REMOVED  
Interface Ethernet5/1 is down (Interface removed)
```

```
In [2]: run nanog_demo.py
```

```
.....  
NEW LOG EVENT:  
    2015 Apr  2:14:25:06  switch1  ETHPORT-5-IF_DOWN_INTERFACE_REMOVED  
        Interface Ethernet5/1 is down (Interface removed)  
  
[switch1]  Log event matched remediation for:  ETHPORT-5-IF DOWN INTERFA
```

```
In [2]: run nanog_demo.py
```

```
.....  
NEW LOG EVENT:  
    2015 Apr  2:14:25:06  switch1  ETHPORT-5-IF_DOWN_INTERFACE_REMOVED  
    Interface Ethernet5/1 is down (Interface removed)  
  
[switch1]  Log event matched remediation for:  ETHPORT-5-IF_DOWN_INTERFA  
[switch1]  Checking status of module 5.
```

```
In [2]: run nanog_demo.py
```

```
.....  
NEW LOG EVENT:  
    2015 Apr  2:14:25:06  switch1  ETHPORT-5-IF_DOWN_INTERFACE_REMOVED  
    Interface Ethernet5/1 is down (Interface removed)  
  
[switch1]  Log event matched remediation for:  ETHPORT-5-IF_DOWN_INTERFA  
[switch1]  Checking status of module 5.  
[switch1]  Module is currently online.  Checking uptime.
```

```
In [2]: run nanog_demo.py
```

```
.....  
NEW LOG EVENT:  
    2015 Apr  2:14:25:06  switch1  ETHPORT-5-IF_DOWN_INTERFACE_REMOVED  
    Interface Ethernet5/1 is down (Interface removed)  
  
[switch1]  Log event matched remediation for:  ETHPORT-5-IF_DOWN_INTERFA  
[switch1]  Checking status of module 5.  
[switch1]  Module is currently online.  Checking uptime.  
[switch1]  Module uptime appears sane - taking no further action.
```

```
In [2]: run nanog_demo.py
```

```
.....  
NEW LOG EVENT:
```

```
    2015 May 18:12:43:15  switch2  ETHPORT-5-IF_DOWN_LINK_FAILURE  
        Interface Ethernet1/4 is down (Link failure)
```

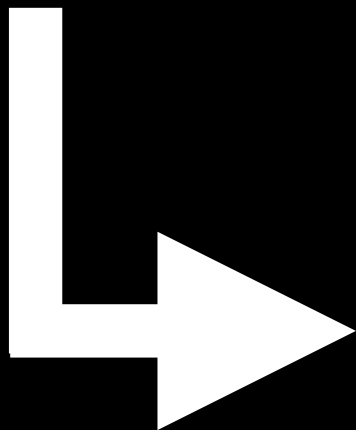
```
In [2]: run nanog_demo.py
```

```
.....  
NEW LOG EVENT:  
    2015 May 18:12:43:15  switch2  ETHPORT-5-IF_DOWN_LINK_FAILURE  
    Interface Ethernet1/4 is down (Link failure)  
  
[switch2]  Log event matched remediation for:  ETHPORT-5-IF_DOWN_LINK_  
[switch2]  Flapping link detected for interface Ethernet1/4!  Interfac  
of 229 exceeds threshold!  
[switch2]  Checking light levels for interface Ethernet1/4  
[switch2]  Rx Power for interface Ethernet1/4 is too low [-8.84 dBm]!  
[switch2]  This link should be drained it's fiber and patch-panel port  
aned or replaced.
```

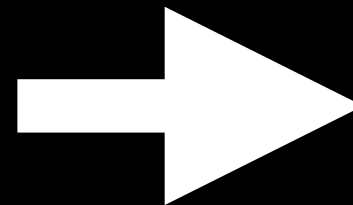
```
[switch2] WARNING: Rx Power for 1/4 is too low [-8.84 dB
```

```
The fiber and patch-panel ports should be checked.
```

**syslog
message**



**problem
identified**



**solution
suggested**

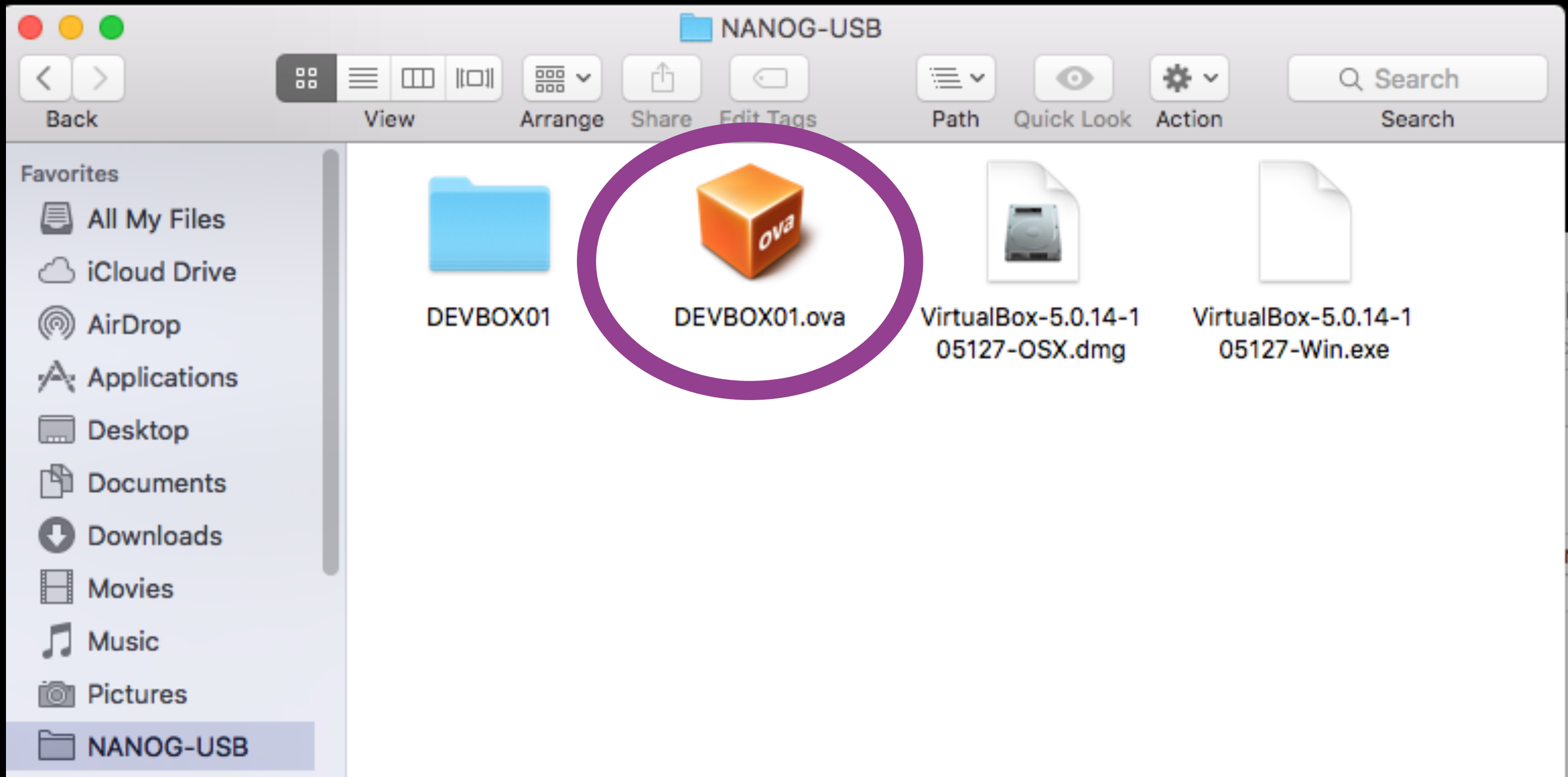
the lab environment

a virtual machine of
Ubuntu desktop

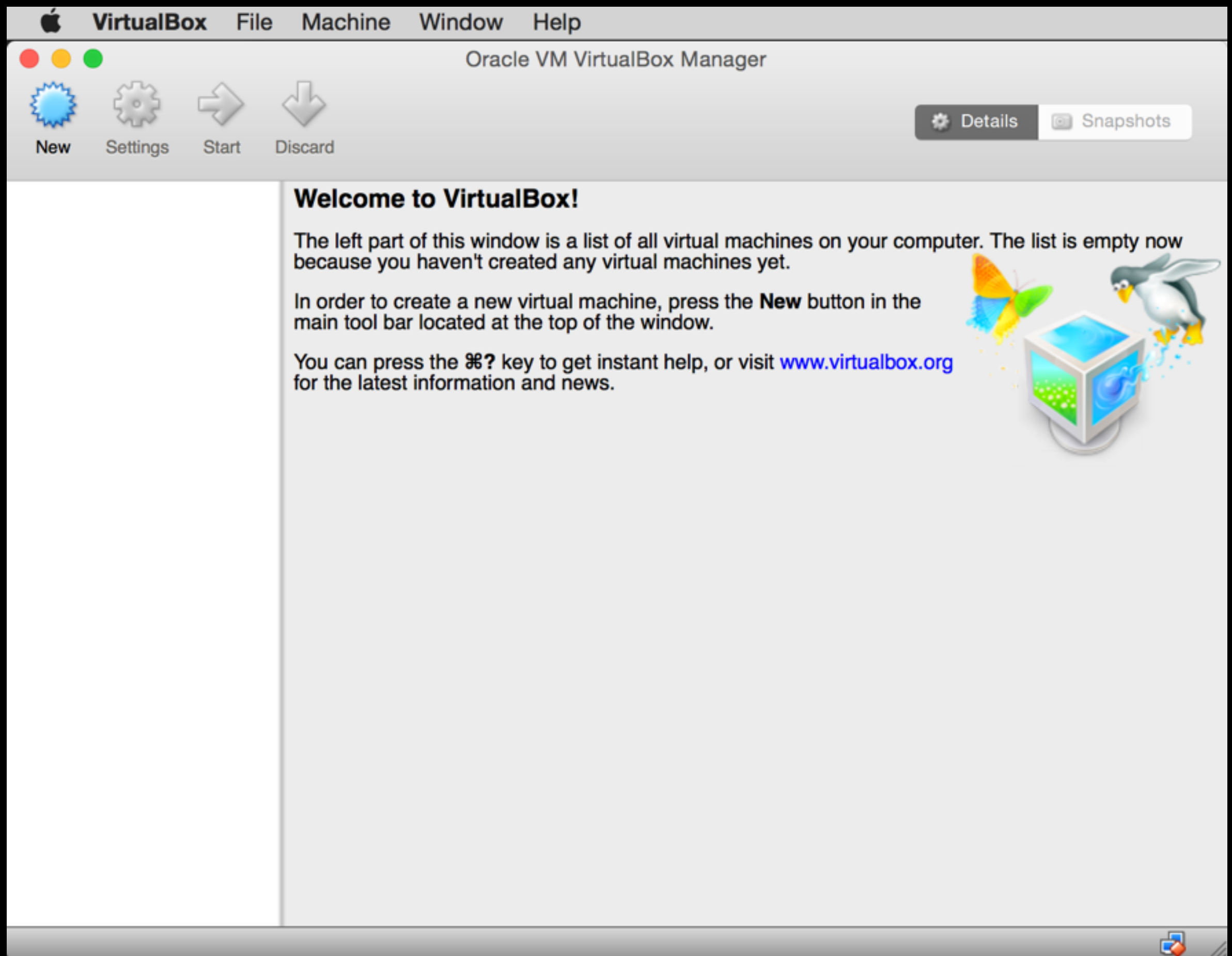
customized with iPython! and extras

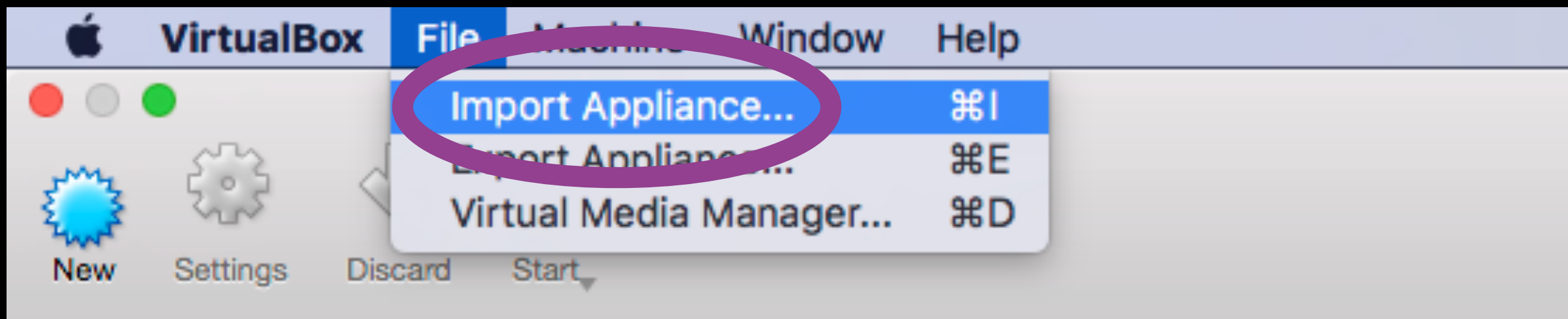
Python 2.7.11 & 3.5.1
MySQL server & Python libs

importing the
virtual appliance



download it from netengcode.com!







Appliance settings

Importing virtual disk image 'DEVBOX01-disk1.vmdk' ... (2/3)

48 seconds remaining

These are the virtual machines contained in the appliance and the top of the window shows the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

Description	Configuration
Virtual System 1	
 Name	DEVBOX01
 Product	netfbar
 Guest OS Type	 Ubuntu (64-bit)
 CPU	1
 RAM	1024 MB

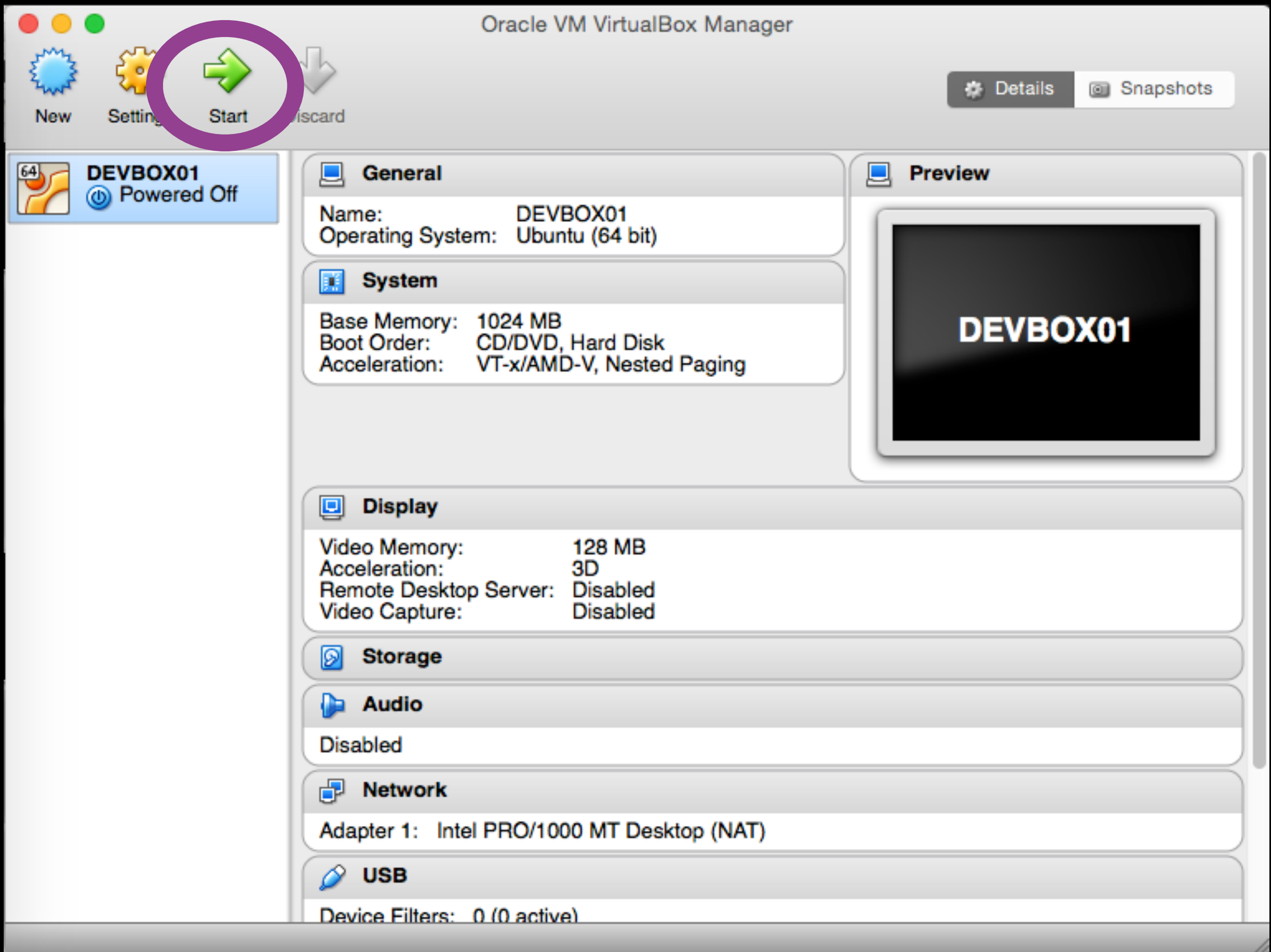
☐ Reinitialize the MAC address of all network cards

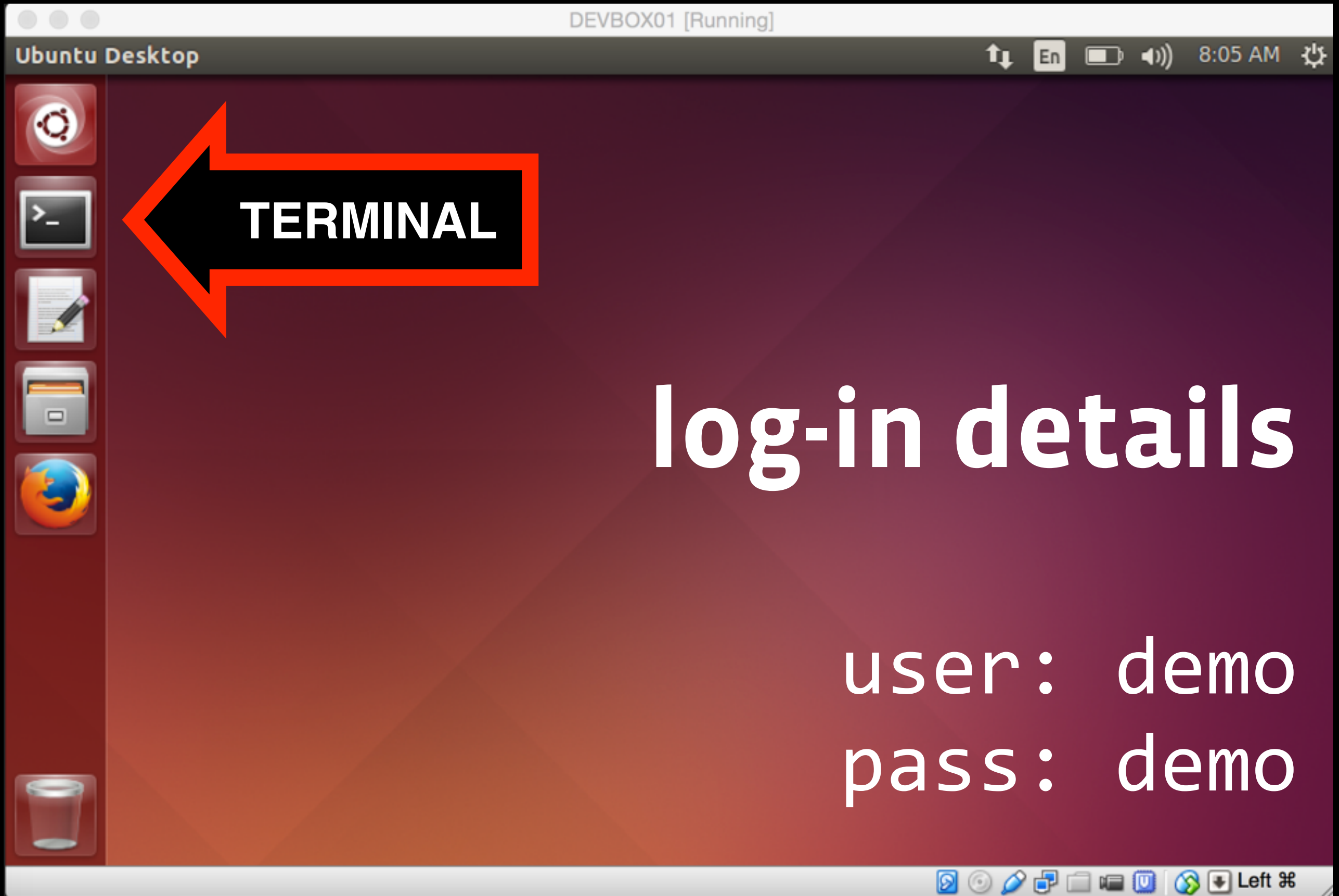
Restore Defaults

Go Back

Import

Cancel





keyboard shortcuts to break out of the VM



f

enter or leave
full-screen

introducing IPython!

<http://ipython.org/ipython-doc/3/install/install.html>

File Edit View Search Terminal Help

```
demo@DEVBOX01:~$ ipython
```

```
demo@DEVBOX01:~$ ipython
```

```
Python 2.7.9 (default, May 21 2015, 15:34:19)
```

```
Type "copyright", "credits" or "license" for more in  
formation.
```

```
IPython 3.1.0 -- An enhanced Interactive Python.
```

```
?          -> Introduction and overview of IPython's  
features.
```

```
%quickref  -> Quick reference.
```

```
help       -> Python's own help system.
```

```
object?    -> Details about 'object', use 'object??'  
for extra details.
```

```
In [1]: █
```

simple prototyping

```
In [1]: print('Welcome to NANOG!')
```

simple prototyping

```
In [1]: print('Welcome to NANOG!')  
Welcome to NANOG!
```

scrolling back

```
In [2]: history
```

scrolling back

```
In [2]: history  
print('Welcome to NANOG!')  
history
```

leaving

```
In [3]: exit
```

```
demo@DEVBOX01:~$
```

a few Python basics

the basic data structures

the basic data structures

- **strings**
- **lists**
- **tuples**
- **dictionaries**

strings in Python

```
session1 = 'is ipv6 really faster?'
```

any sequence of characters
enclosed by single or double-quotes

<https://docs.python.org/3/tutorial/introduction.html#strings>

lists in Python

```
session1 = 'is ipv6 really faster?'  
tutorial1 = 'netops coding 101'
```

```
my_schedule = [session1, tutorial1]
```

“comma-separated values (items)
between **square brackets**”

<https://docs.python.org/3/tutorial/introduction.html#lists>

tuples in Python

```
session1 = 'is ipv6 really faster?'  
tutorial1 = 'netops coding 101'
```

```
my_schedule = session1, tutorial1
```

comma-separated values

<https://docs.python.org/3/library/stdtypes.html#tuples>

lists vs tuples?

lists vs tuples

the contents of a list

may be changed

the contents of a tuple

may not be changed

quick caveat!

"...it is actually the comma
which makes a tuple,
not the parentheses."

<https://docs.python.org/3/library/stdtypes.html#tuple>

dictionaries
(“hash tables”)

dictionaries in Python

```
nanogs = {  
    66: 'San Diego',  
    65: 'Montreal',  
    64: 'San Francisco'  
}
```

“an unordered set of **key: value** pairs” — keys must be unique

<https://docs.python.org/3/tutorial/datastructures.html#dictionaries>

dictionaries in Python

```
nanogs = {  
    66: 'San Diego',  
    65: 'Montreal',  
    64: 'San Francisco'  
}
```

66 is the key and
'San Diego' the value

<https://docs.python.org/3/tutorial/datastructures.html#dictionaries>

dictionaries in Python

```
In [8]: print(nanogs[64])  
San Francisco, CA
```

values are accessed using the
notation of **[key]**

<https://docs.python.org/3/tutorial/datastructures.html#dictionaries>

try it out!

try it out!

- **create a variable** with your neighbor's name
- **create a list** of your adjacent neighbor's names
- **create a dictionary** mapping those last names to first names

try it out!

- create a variable with your neighbor's name
`name = ''`
- create a list of your adjacent neighbor's names
`names = []`
- create a dictionary mapping those last names to first names
`names = {'last': 'first'}`

questions?

working with modules

what is a module?

```
In [2]: import re
```

**imports,
or makes available,
the “re” module**

part of the standard library

https://docs.python.org/3/reference/simple_stmts.html#import

```
In [2]: import re
```

“re” is the
module’s
“namespace”

```
In [1]: import re as re_module
```

“**as**” imports the module
under a different
“**namespace**”

```
In [2]: import re
```

**methods from a
module are accessed by
namespace.method()**

```
In [2]: import re
```

**we will be using
re.match()**

```
In [2]: import re
```

using `re.match()`

two inputs —

“a regex” and

“thing to match against”

```
In [2]: import re
```

using `re.match()`

it returns —

None or a match object

try it out!

try it out!

- **import** the "re" module
- **type** `help(re)`
- **type** `re.match('eth', 'ethernet')`

questions?

reading syslog messages

reading in a file

```
SYSLOG_FILE = 'syslog.txt'
```

assigning the
file path to a **variable**

reading in a file

```
SYSLOG_FILE = 'syslog.txt'
```

```
with open(SYSLOG_FILE, mode='r') \  
    as syslog:
```

opening a **file handle**

reading in a file

```
SYSLOG_FILE = 'syslog.txt'
```

```
with open(SYSLOG_FILE, mode='r') \  
    as syslog:
```

```
    log_lines = syslog.readlines()
```

using **.readlines()**

to get a list of lines

reading in a file

```
SYSLOG_FILE = 'syslog.txt'
```

```
with open(SYSLOG_FILE, mode='r') \  
    as syslog:
```

```
    log_lines = syslog.readlines()
```

```
for line in log_lines:  
    print(line)
```

try it out!

try it out!

- **open the file "syslog.txt"**
- **try out ".readlines()"**

questions?

parsing syslog messages

a syslog message

```
2015 Apr  2 14:25:06 switch1 %ETHERPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down  
(Interface removed)
```

looking with an abstract lens

```
2015 Apr 2 14:25:06 switch1 %ETHERPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down  
(Interface removed)
```

looking with an abstract lens

```
DATESTAMP      14:25:06 switch1 %ETHPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down  
(Interface removed)
```

looking with an abstract lens

```
DATESTAMP    14:25:06 switch1 %ETHERPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down  
(Interface removed)
```

looking with an abstract lens

```
DATESTAMP TIMESTAMP switch1 %ETHERPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down  
(Interface removed)
```

looking with an abstract lens

```
DATESTAMP  TIMESTAMP  DEVICE          %ETHPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down  
(Interface removed)
```

looking with an abstract lens

```
DATESTAMP  TIMESTAMP  DEVICE          %ETHERPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down  
(Interface removed)
```

looking with an abstract lens

DATESTAMP	TIMESTAMP	DEVICE	ERROR_CODE:
Interface	Ethernet5/1	is down	(Interface removed)

looking with an abstract lens

DATESTAMP	TIMESTAMP	DEVICE	ERROR_CODE :
Interface	Ethernet5/1	is down	(Interface removed)

looking with an abstract lens

DATESTAMP	TIMESTAMP	DEVICE	ERROR_CODE :
ERROR_MESSAGE			

**from log message to
structured data**

structured data

LOG MESSAGE

DATESTAMP

TIMESTAMP

DEVICE

ERROR
CODE

ERROR
MESSAGE

**regular
expressions!**

a way to ask —

**does this pattern exist in a
given block of text?**

<https://docs.python.org/3/howto/regex.html>

where
“this pattern” is
created by you

**specifying the pattern
is often called
“building a regex”**

an example with BGP

```
ip as-path  
access-list 1  
permit "^$"
```

"^\$"

**beginning of line
followed by
end of line
(an empty line!)**

getting started with regular expressions in Python

try it out!

```
In [1]: import re
```

```
In [2]: date = '2015 Jun 2'
```

try it out!

```
In [1]: import re
```

```
In [2]: date = '2015 Jun 2'
```

```
In [1]: DATESTAMP_RE = r' (
```

a raw string,
with an opening “grouping” symbol

try it out!

```
In [1]: import re
```

```
In [2]: date = '2015 Jun 2'
```

```
In [1]: DATESTAMP_RE = r'(\d
```

\ to indicate that a

pattern-matching character follows

try it out!

```
In [1]: import re
```

```
In [2]: date = '2015 Jun 2'
```

```
In [1]: DATESTAMP_RE = r'(\d
```

d to match one **digit** (integer)

<http://www.merriam-webster.com/dictionary/digit>

try it out!

```
In [1]: import re
```

```
In [2]: date = '2015 Jun 2'
```

```
In [1]: DATESTAMP_RE = r'(\d
```

instead of pattern-matching,
we could put exact text too --
"2015" for example

try it out!

```
In [1]: import re
```

```
In [2]: date = '2015 Jun 2'
```

```
In [1]: DATESTAMP_RE = r'(\d+
```

+ to indicate **one or more**
(of the previous thing)

try it out!

```
In [1]: import re
```

```
In [2]: date = '2015 Jun 2'
```

```
In [1]: DATESTAMP_RE = r'(\d+
```

\d+ will match 2015

try it out!

```
In [1]: import re
```

```
In [2]: date = '2015 Jun 2'
```

```
In [1]: DATESTAMP_RE = r'(\d+\s
```

\s will match **one space**

try it out!

```
In [1]: import re
```

```
In [2]: date = '2015 Jun 2'
```

```
DATESTAMP_RE = r'(\d+\s\w+
```

\W+ will match one or more
characters (Jun)

try it out!

```
matched = re.match(DATESTAMP_RE, date)
```

assigning the

return of **re.match()**

to a **variable** (matched)

try it out!

```
In [3]: DATESTAMP_RE = r'(\d+\s\w+\s\d+)'

In [4]: matched = re.match(DATESTAMP_RE, c

In [5]: print(matched)
<_sre.SRE_Match object at 0x7fb822500468>
```

a **match** object

was **returned** (it matched)

try it out!

```
DATESTAMP_RE = r'(\d+\s\w+\s\d+)'
```

```
In [6]: print(matched.group(1))  
2015 Jun 2
```

accessing the matched
data using **.group()**

questions?

parsing syslog

```
2015 Apr  2 14:25:06 switch1 %ETHPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down (Interface removed)
```

```
DATESTAMP_RE = r'(\d+\s+\w+\s+\d+)'
```

\d+

one or
more
digits

```
2015 Apr 2 14:25:06 switch1 %ETHPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down (Interface removed)
```

DATESTAMP_RE = `r'(\d+\s+\w+\s+\d+)'`

`\s+`

one or
more
spaces

```
2015 Apr 2 14:25:06 switch1 %ETHPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down (Interface removed)
```

```
DATESTAMP_RE = r'(\d+\s+\w+\s+\d+)'
```

\w+

**one or
more
characters**

```
2015 Apr  2 14:25:06 switch1 %ETHPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down (Interface removed)
```

```
DEVICE_NAME_RE = r'(\S+)'
```

\S+

**one or more
non-space
characters**

```
2015 Apr 2 14:25:06 switch1 %ETHPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down (Interface removed)
```

```
ERROR_CODE_RE = r'%(\S+):'
```

\S+

**one or more
non-space
characters**

```
2015 Apr  2 14:25:06 switch1 %ETHPORT-5-  
IF_DOWN_INTERFACE_REMOVED: Interface  
Ethernet5/1 is down (Interface  
removed)
```

```
ERROR_MESSAGE_RE = r'(\.*)'
```

`\.*`

one or more
non-newline
characters

2015 Apr 2 14:25:06 switch1 %ETHPORT-5-
IF_DOWN_INTERFACE_REMOVED: Interface
Ethernet5/1 is down (Interface removed)

DATESTAMP_RE = r'(\d+\s+\w+\s+\d+)'

TIMESTAMP_RE = r'(\d+:\d+:\d+)'

DEVICE_NAME_RE = r'(\S+)'

ERROR_CODE_RE = r'%(\S+):'

ERROR_MESSAGE_RE = r'(\.*)'

COLUMN_DELIMITER_RE = r'(\s+)'

```
# Combine all of the regexes together
SYSLOG_RE = (
    DATESTAMP_RE + COLUMN_DELIMITER_RE +
    TIMESTAMP_RE + COLUMN_DELIMITER_RE +
    DEVICE_NAME_RE + COLUMN_DELIMITER_RE +
    ERROR_CODE_RE + COLUMN_DELIMITER_RE +
    ERROR_MSG_RE)
```

using the “re” module to match

```
# Parsing time!  
for line in log_lines:
```

using the “re” module to match

```
# Parsing time!  
for line in log_lines:  
    matched = re.match(SYSLOG_RE, line)
```

using the “re” module to match

```
# Parsing time!  
for line in log_lines:  
    matched = re.match(SYSLOG_RE, line)  
  
    if not matched:  
        continue
```

using the “re” module to match

```
# Parsing time!  
for line in log_lines:
```

```
    matched = re.match(SYSLOG_RE, line)
```

```
        if not matched:  
            continue
```

**continue — skip this
log line and go to the next**

https://docs.python.org/3/reference/simple_stmts.html#continue

using the “re” module to match

```
# Parsing time!
for line in log_lines:

    matched = re.match(SYSLOG_RE, line)

    if not matched:
        continue

    datestamp, timestamp, device_name, \
        error_code, error_message = matched.groups()
```

```
result of "matched.groups()"
al variables
mestamp, device_name, \
e, error_message = matched.groups()
```

start of group 1

end of group 1

DATESTAMP_RE = r'(\d+\s+\w+\s+\d+).'

TIMESTAMP_RE = r'(\d+:\d+:\d+).'

DEVICE_NAME_RE = r'(\S+).'

ERROR_CODE_RE = r'%(\S+):.'

ERROR_MESSAGE_RE = r'(\.*).'

try it out!

try it out!

```
In [1]: print(SYSLOG_RE)
```

try it out!

```
In [1]: print(SYSLOG_RE)
```

```
(\d+\s+\w+\s+\d+)\s+(\d+:\d+:\d+)\s+(\S+)\s+%(\S+):\s+(.*)
```

note -

SYSLOG_RE is already staged

try it out!

```
In [10]: for line in log_lines:
        ....:     matched = re.match(SYSLOG_RE, line)
```

try it out!

```
In [10]: for line in log_lines:
.....:     matched = re.match(SYSLOG_RE, line)
.....:     if not matched:
.....:         continue
```

try it out!

```
In [10]: for line in log_lines:
.....:     matched = re.match(SYSLOG_RE, line)
.....:     if not matched:
.....:         continue
.....:     print(matched.groups())
.....:     print(' ')
.....:
```

try it out!

```
In [10]: for line in log_lines:
.....:     matched = re.match(SYSLOG_RE, line)
.....:     if not matched:
.....:         continue
.....:     print(matched.groups())
.....:     print('')
.....:
('2015 Apr  2', '14:25:06', 'switch1', 'ETHPORT-5-IF-
REMOVED', 'Interface Ethernet5/1 is down (Interface

('2015 May 18', '12:43:15', 'switch2', 'ETHPORT-5-IF-
RE', 'Interface Ethernet1/4 is down (Link failure)')
```

questions?

building our first remediation

**first remediation —
a linecard failure**

2015 Apr 2 14:25:06 switch1 %ETHERPORT-5-
IF_DOWN_INTERFACE_REMOVED: Interface
Ethernet5/1 is down (Interface removed)

2015 Apr 2 14:25:06 switch1 %ETHPORT-5-
IF_DOWN_INTERFACE_REMOVED: Interface
Ethernet5/1 is down (Interface removed)

```
datestamp, timestamp, device_name, \  
error_code, error_message = matched.groups()
```

2015 Apr 2 14:25:06 switch1 %ETHPORT-5-
IF_DOWN_INTERFACE_REMOVED: Interface
Ethernet5/1 is down (Interface removed)

```
datestamp, timestamp, device_name, \  
error_code, error_message = matched.groups()
```

```
if 'IF_DOWN_INTERFACE_REMOVED' in error_code:
```

2015 Apr 2 14:25:06 switch1 %ETHPORT-5-
IF_DOWN_INTERFACE_REMOVED: Interface
Ethernet5/1 is down (Interface removed)

```
datestamp, timestamp, device_name, \  
    error_code, error_message = matched.groups()
```

```
if 'IF_DOWN_INTERFACE_REMOVED' in error_code:
```

```
    print('Found a known error on {0}'  
          '-- attempting remediation.'  
          .format(device_name))
```

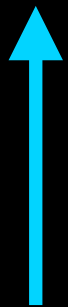
2015 Apr 2 14:25:06 switch1 %ETHPORT-5-
IF_DOWN_INTERFACE_REMOVED: Interface
Ethernet5/1 is down (Interface removed)

```
interface = re.match(  
    r' .+(\d+)/\d+', error_message)
```

2015 Apr 2 14:25:06 switch1 %ETHPORT-5-
IF_DOWN_INTERFACE_REMOVED: Interface
Ethernet5/1 is down (Interface removed)

```
interface = re.match(  
    r'^(.+)(\d+)/\d+', error_message)
```

```
In [20]: print(bool(interface))  
True
```



Did the interface regex find a match?

2015 Apr 2 14:25:06 switch1 %ETHERPORT-5-
IF_DOWN_INTERFACE_REMOVED: Interface
Ethernet5/1 is down (Interface removed)

```
interface = re.match(  
    r'.+(\d+)/\d+', error_message)
```

```
In [20]: print(bool(interface))  
True
```

```
module_number = interface.group(1)
```

Group "1" is the data matched by the first set of ()

2015 Apr 2 14:25:06 switch1 %ETHPORT-5-
IF_DOWN_INTERFACE_REMOVED: Interface
Ethernet5/1 is down (Interface removed)

```
interface = re.match(  
    r' .+(\d+)/\d+', error_message)
```

```
In [20]: print(bool(interface))  
True
```

```
module_number = interface.group(1)
```

```
In [23]: print(module_number)
```

5

2015 Apr 2 14:25:06 switch1 %ETHPORT-5-
IF_DOWN_INTERFACE_REMOVED: Interface
Ethernet5/1 is down (Interface removed)

```
module_number = interface.group(1)
```

```
command = ('show module {0}'.format(  
            module_number))
```

try it out!

try it out!

- parse out **only** the module number from:

Ethernet5/1

- populate that value into the string:

show module {module}

sending commands over SSH

sending commands over SSH

```
In [1]: import ssh_helper
```

sending commands over SSH

```
In [1]: import ssh_helper
```

a wrapper around Paramiko
(Python SSH client)
included with our demo code

<http://docs.paramiko.org/>

no paramiko?

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ImportError: No module named paramiko
```

<http://docs.paramiko.org/>

installing paramiko

```
$ sudo pip install paramiko
```

installing paramiko

```
$ sudo pip install paramiko
```

```
Downloading/unpacking paramiko
  Downloading paramiko-1.15.2-py2.py3-none-any.whl (165kB): 165kB downloaded
Requirement already satisfied (use --upgrade to upgrade): pycrypto in
/usr/lib/python2.7/dist-packages (from paramiko)
Downloading/unpacking ecdsa>=0.11 (from paramiko)
  Downloading ecdsa-0.13-py2.py3-none-any.whl (86kB): 86kB downloaded
Installing collected packages: paramiko, ecdsa
Successfully installed paramiko ecdsa
Cleaning up...
demo@DEVBOX01:~$
```

sending commands over ssh

```
import ssh_helper
```

sending commands over ssh

```
import ssh_helper  
  
ssh = ssh_helper.SSHSession(  
    device_name,  
    username='demo',  
    passwd='demo' )
```

sending commands over ssh

```
import ssh_helper
```

```
ssh = ssh_helper.SSHSession(  
    device_name,  
    username='demo',  
    passwd='demo' )
```

Connection opened!

sending commands over ssh

```
command = ('show module {0}'.format(  
            module_number))
```

```
show_module = ssh.write([command])
```

sending commands over ssh

```
In [41]: command
```

```
Out[41]: 'show module 5'
```

```
In [42]: commands = [command]
```

sending commands over ssh

```
In [41]: command
```

```
Out[41]: 'show module 5'
```

```
In [42]: commands = [command]
```

```
In [43]: output = ssh.write(commands)
```

try it out!

try it out!

- open a dummy ssh connection using:

```
import ssh_helper
```

```
ssh = ssh_helper.SSHSession('switch1')
```

- inspect the output from:

```
show module 5
```

inspecting the output

```
import pprint  
pprint.pprint(output[0:5])
```

output is a **list** —

[0:5] is a **slice** of the first 5 lines

<https://docs.python.org/3/tutorial/introduction.html>

inspecting the output

```
online = False
```

```
for line in output:
```

inspecting the output

```
online = False  
  
for line in output:  
    if 'ok' in line.lower():
```

checking if one **string**
is present in another string

inspecting the output

```
online = False
```

```
for line in output:  
    if 'ok' in line.lower()
```

ignoring case using **.lower()**

inspecting the output

```
online = False
```

```
for line in output:  
    if 'ok' in line.lower():  
        online = True
```

inspecting the output

```
online = False
```

```
for line in output:  
    if 'ok' in line.lower():  
        online = True
```

```
In [68]: print(online)  
True
```

reacting to the output

```
if online:
```

```
    command = (  
        'show module uptime | '  
        'egrep -A 3 "Module 3"')
```

using on-device
filtering with **egrep**

reacting to the output

```
if online:
```

```
    command = (  
        'show module uptime | '  
        'egrep -A 3 "Module 3"')  
  
    output = ssh.write([command])
```

reacting to the output

```
if online:
```

```
    command = (  
        'show module uptime | '  
        'egrep -A 3 "Module 3"')  
  
    output = ssh.write([command])
```

idea! parse the uptime and
react based on the value

reacting to the output

```
if online:
```

```
...
```

```
else:
```

```
    print('Module {0} on {1} is '  
          'offline!'.format(  
            module_number, device_name))
```

questions?

**next remediation —
link flaps &
light level checking**

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
datestamp, timestamp, device_name, \  
error_code, error_message = matched.groups()
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
datestamp, timestamp, device_name, \  
    error_code, error_message = matched.groups()
```

```
if 'IF_DOWN_LINK_FAILURE' in error_code:
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
datestamp, timestamp, device_name, \  
    error_code, error_message = matched.groups()
```

```
if 'IF_DOWN_LINK_FAILURE' in error_code:
```

```
    interface = re.match(  
        r'.+(\d+/\d+)', error_message)
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
interface = re.match(  
    r'.+(\d+/\d+)', error_message)
```

```
if interface:
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
interface = re.match(  
    r'.+(\d+/\d+)', error_message)
```

```
if interface:  
    interface = interface.group(1)
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
interface = re.match(  
    r'.+(\d+/\d+)', error_message)
```

```
if interface:  
    interface = interface.group(1)
```

```
In [22]: print(interface)  
1/4
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
command = (  
    'show interface eth {0}'.format(  
        interface))
```

```
2015 May 18 12:43:15 switch2 %ETHPORT-5-  
IF_DOWN_LINK_FAILURE: Interface  
Ethernet1/4 is down (Link failure)
```

```
command = (  
    'show interface eth {0}'.format(  
        interface))
```

```
In [24]: command
```

```
Out[24]: 'show interface eth 1/4'
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
In [24]: command  
Out[24]: 'show interface eth 1/4'
```

```
commands = [command]
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
In [24]: command  
Out[24]: 'show interface eth 1/4'
```

```
commands = [command]
```

```
output = ssh.write(commands)
```

```
2015 May 18 12:43:15 switch2 %ETHPORT-5-  
IF_DOWN_LINK_FAILURE: Interface  
Ethernet1/4 is down (Link failure)
```

```
for line in output:
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
for line in output:
```

```
    interface_resets = re.match(  
        r'^\s+(\d+) interface resets',
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
for line in output:
```

```
    interface_resets = re.match(  
        r'^\s+(\d+) interface resets',  
        line.lower())
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
for line in output:
```

```
    interface_resets = re.match(  
        r'^\s+(\d+) interface resets',  
        line.lower())
```

```
    if not interface_resets:  
        continue
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
for line in output:
```

```
    interface_resets = re.match(  
        r'^\s+(\d+) interface resets',  
        line.lower())
```

```
    if not interface_resets:  
        continue
```

```
    interface_resets = int(  
        interface_resets.group(1)
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
for line in output:
```

```
    interface_resets = re.match(  
        r'^\s+(\d+) interface resets',  
        line.lower())
```

```
    if not interface_resets:  
        continue
```

```
    interface_resets = int(  
        interface_resets.group(1))
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
if interface_resets > 10:
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
if interface_resets > 10:
```

```
    print('Warning!  Interface resets are '  
          'too high for {1} on {0}!'.format(  
              device_name, interface_resets))
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
if interface_resets > 10:
```

```
    print('Warning!  Interface resets are '  
          'too high for {1} on {0}!'.format(  
                device_name, interface_resets))
```

```
command = (  
    'show interface eth {0} transceiver '
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
if interface_resets > 10:
```

```
    print('Warning!  Interface resets are '  
          'too high for {1} on {0}!'.format(  
                device_name, interface_resets))
```

```
    command = (  
        'show interface eth {0} transceiver '  
        'details | egrep "(Rx|rx)".format(  
            interface_name)
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
if interface_resets > 10:
```

```
    print('Warning!  Interface resets are '  
          'too high for {1} on {0}!'.format(  
          device_name, interface_resets))
```

```
    command = (  
        'show interface eth {0} transceiver '  
        'details | egrep "(Rx|rx)".format(  
        interface))
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
command = (  
    'show interface eth {0} transceiver '  
    'details | egrep "(Rx|rx)".format(  
    interface))
```

```
output = ssh.write([command])
```

"show transceiver details"

```
2015 May 18 12:43:15 switch2 %ETHPORT-5-  
IF_DOWN_LINK_FAILURE: Interface  
Ethernet1/4 is down (Link failure)
```

```
for line in output:
```

example output:

```
'    Rx Power      -8.84 dBm      ....'
```

```
2015 May 18 12:43:15 switch2 %ETHPORT-5-  
IF_DOWN_LINK_FAILURE: Interface  
Ethernet1/4 is down (Link failure)
```

```
for line in output:
```

```
    rx_power = re.match(
```

example output:

```
'    Rx Power      -8.84 dBm      ....'
```

```
2015 May 18 12:43:15 switch2 %ETHPORT-5-  
IF_DOWN_LINK_FAILURE: Interface  
Ethernet1/4 is down (Link failure)
```

```
for line in output:
```

```
    rx_power = re.match(  
        ' .+power\s+(\s+(-\d+\.\d+) dBm)',
```

example output:

```
' Rx Power      -8.84 dBm      ....'
```

```
2015 May 18 12:43:15 switch2 %ETHPORT-5-  
IF_DOWN_LINK_FAILURE: Interface  
Ethernet1/4 is down (Link failure)
```

```
for line in output:
```

```
    rx_power = re.match(  
        '.*+power\s+(-\d+\.\d+) dbm',  
        output.lower())
```

example output:

```
'    Rx Power      -8.84 dBm      ....'
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
for line in output:
```

```
    rx_power = re.match(  
        '.*+power\s+(-\d+\.\d+) dbm',  
        output.lower())
```

```
    if not rx_power:  
        continue
```

example output:

```
'    Rx Power      -8.84 dBm      ....'
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
for line in output:
```

```
    rx_power = re.match(  
        '.*+power\s+(-\d+\.\d+) dbm',  
        output.lower())
```

```
    rx_power = float(  
        rx_power.group(1))
```

example output:

```
'    Rx Power      -8.84 dBm      ....'
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
rx_power = float(  
    rx_power.group(1))
```

```
In [13]: rx_power  
Out[13]: -8.84
```

**converting a string to a float
for mathematical comparison**

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
if rx_power < -7.00:
```

2015 May 18 12:43:15 switch2 %ETHPORT-5-
IF_DOWN_LINK_FAILURE: Interface
Ethernet1/4 is down (Link failure)

```
if rx_power < -7.00:
```

```
    print('Warning!  Rx power for {int} '  
          'is too low [{power}dBm]!'.format(  
            int=interface, power=rx_power))
```

send_email()

questions?

**keeping track of
what happened**

options

- **in memory (lists, dictionaries, etc)**
- **on disk (as a file)**
- **in a database**

comparing a few database options

- **sqlite**

- **mysql**

an event as a database table

EVENT

ID

DATESTAMP

DEVICE

ERROR
CODE

ERROR
MESSAGE

RESULT

getting started with sqlite

```
import sqlite3
```

building the database

```
import sqlite3
```

```
SCHEMA = (''
```

```
    CREATE TABLE events (
```

```
        id            INTEGER PRIMARY KEY AUTOINCREMENT N
```

```
        timestamp    DATE,
```

```
        device       TEXT,
```

```
        error_code   TEXT,
```

```
        error_message TEXT,
```

```
        result       INTEGER)
```

```
''')
```

building the database

```
import sqlite3
```

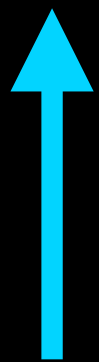
```
SCHEMA = (''  
    CREATE TABLE events (  
        id                INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,  
        datestamp         DATE,  
        device            TEXT,  
        error_code        TEXT,  
        error_message     TEXT,  
        result            INTEGER)  
)
```

```
with sqlite3.connect(db_file) as session:  
    session.execute(schema)
```

creating events

```
sql = (''  
    INSERT INTO events (datestamp, device, error_code, error_message)  
    VALUES (?, ?, ?, ?)  
    ''')
```

```
cursor = session.cursor()  
cursor.execute(  
    sql, (datestamp, device, error_code, error_message))
```

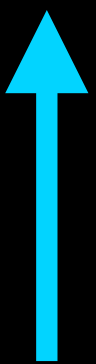


the safe way to handle string substitution

creating events

```
sql = (''  
    INSERT INTO events (datestamp, device, error_code, error_message)  
    VALUES (?, ?, ?, ?)  
    ''')
```

```
cursor = session.cursor()  
cursor.execute(  
    sql, (datestamp, device, error_code, error_message))  
event_id = cursor.lastrowid
```



auto-generated by the database

updating events

```
sql = (''  
      UPDATE events  
      SET result=?  
      WHERE id=?  
      ''')
```

updating events

```
sql = (''  
    UPDATE events  
    SET result=?  
    WHERE id=?  
    ''  
)  
cursor = session.cursor()  
cursor.execute(sql, (event_id, result))
```

questions?

opportunities for improvement

scoping

- **globally scoped code is a bad design**
- **use functions, classes, and modules to keep things organized**

scoping

- **globally scoped code and code inside large functions are both hard to test**
- **concise functions are easier to test**
- **testability saves much pain and suffering later**

organization

- **remediation and system code together**
 - **it can be quite a lot to digest.**
- **"sparse is better than dense" - use smaller concise files and split logically.**

scaling

- **it's not a real-time system, but would need to be.**
- **for log parsing, continuous parsing with some intelligence re: new vs old events.**

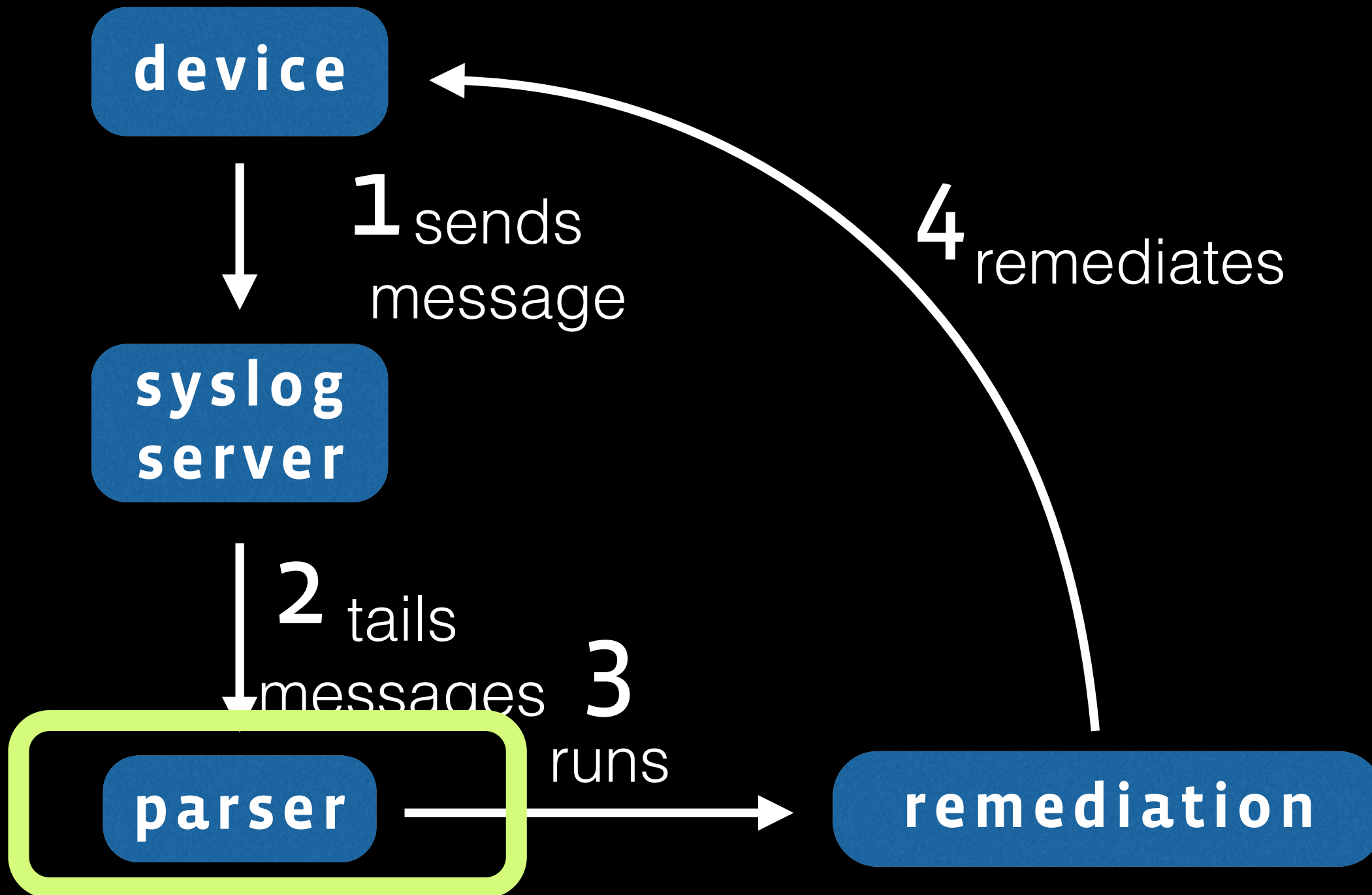
stability

- **persistence is key!**
- **a restart to the parsing script should not re-run previously handled events / remediations.**

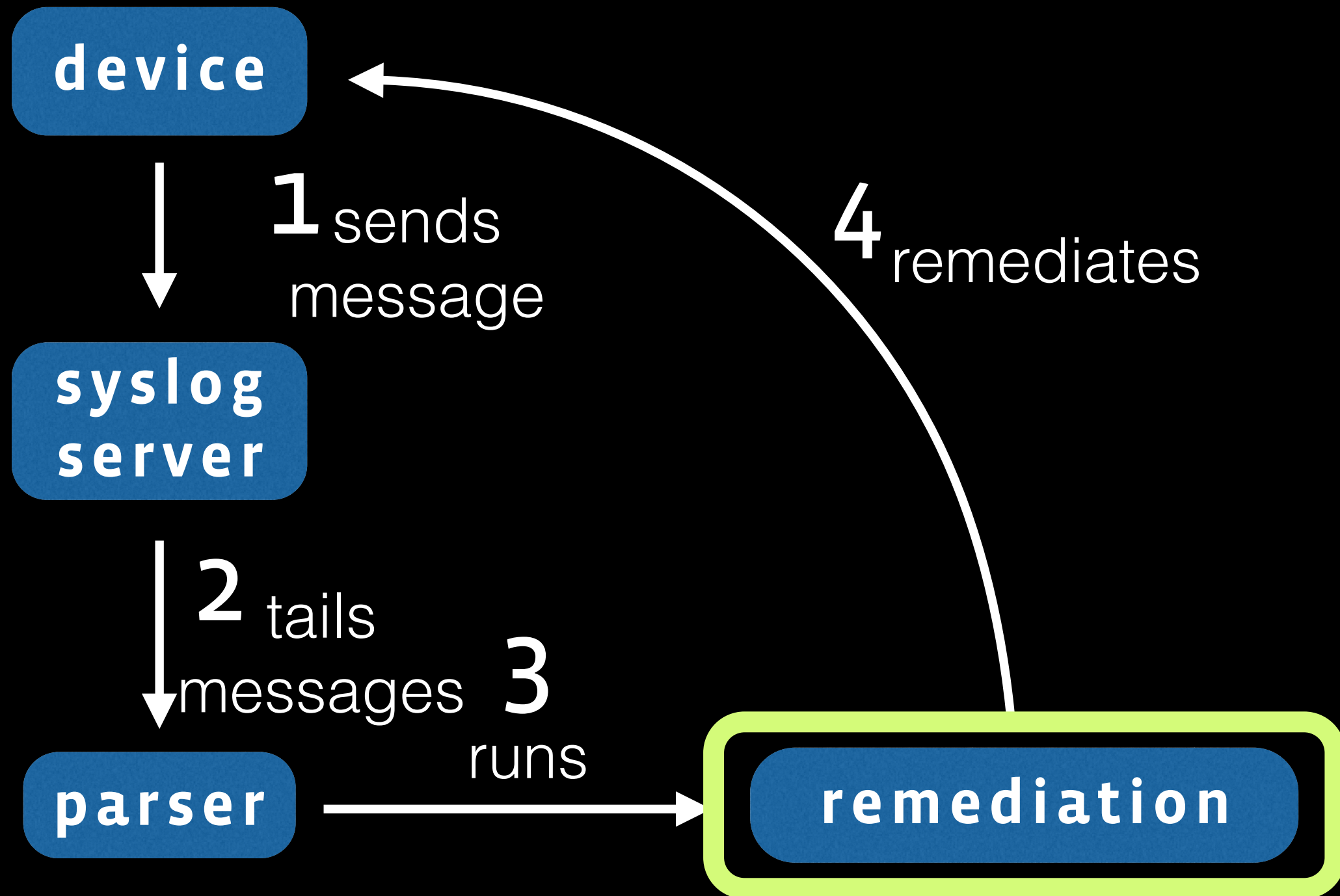
perf

- **one by one processing is okay for a demo...**
- **in real life though, this should break up the load within the script, or as separate scripts/instances.**

system design



system design



appendix

**separating backend
code from
remediation logic**

separating the backend from remediations

```
import remediations

ERROR_CODES_TO_REMEDIATIONS = {
    'IF_DOWN_INTERFACE_REMOVED':
        remediations.linecard_failure,
    'IF_DOWN_LINK_FAILURE':
        remediations.link_failure,
}
```

a **dictionary** where the
values are **functions**!

separating the backend from remediations

```
for known_error_code in ERROR_CODES_TO_REMEDIATIONS:  
    if known_error_code in error_code:
```

separating the backend from remediations

```
for known_error_code in ERROR_CODES_TO_REMEDIATIONS:
    if known_error_code in error_code:
        print('{0} NOTICE: Found a known error [{1}] - attempt  
remediation.'.format(device_name, error_code))
        # Fetch the function assigned to this error code
        remediation = \
            ERROR_CODES_TO_REMEDIATIONS[known_error_code]
```

a **function**

```
# Fetch the function assigned to this error code
remediation = \
    ERROR_CODES_TO_REMEDIATIONS[known_error_code]

# Run that function, passing in the event_id.
remediation(event_id, device_name, error_message)
```

running a function
who's name is **not defined**
in this block of code

```
import remediations

ERROR_CODES_TO_REMEDIATIONS = {
    'IF_DOWN_INTERFACE_REMOVED':
        remediations.linecard_failure,
    'IF_DOWN_LINK_FAILURE':
        remediations.link_failure,
}
```

```
import remediations

ERROR_CODES_TO_REMEDIATIONS = {
    'IF_DOWN_INTERFACE_REMOVED':
        remediations.linecard_failure,
    'IF_DOWN_LINK_FAILURE':
        remediations.link_failure,
}
```

```
demo@DEVBOX01:~/nanog_demo$ vim remediations.py
```

```
import remediations

ERROR_CODES_TO_REMEDIATIONS = {
    'IF_DOWN_INTERFACE_REMOVED':
        remediations.linecard_failure,
    'IF_DOWN_LINK_FAILURE':
        remediations.link_failure,
}
```

```
demo@DEVBOX01:~/nanog_demo$ vim remediations.py
```

```
def linecard_failure(event_id, device_name, error_message)
    """ Linecard Failure Remediation """

    # Matches "5/1" from:
    #     Interface Ethernet5/1 is down (Interface removed
    interface = re.match(r'.+(\d+)/\d+', error_message)
    if not interface:
        return
```



**WHAT WOULD
YOU DO IF
YOU WEREN'T
AFRAID?**

#netengcode

NANOG 66

02.09.2016

david swafford

facebook