NetOps Coding 201

auto remediation for your network!

NANOG 66

02.10.2016

david swafford

facebook

continuing our theme

automating

the remediation of network faults

based on **parsing** of

syslog messages

focusing today

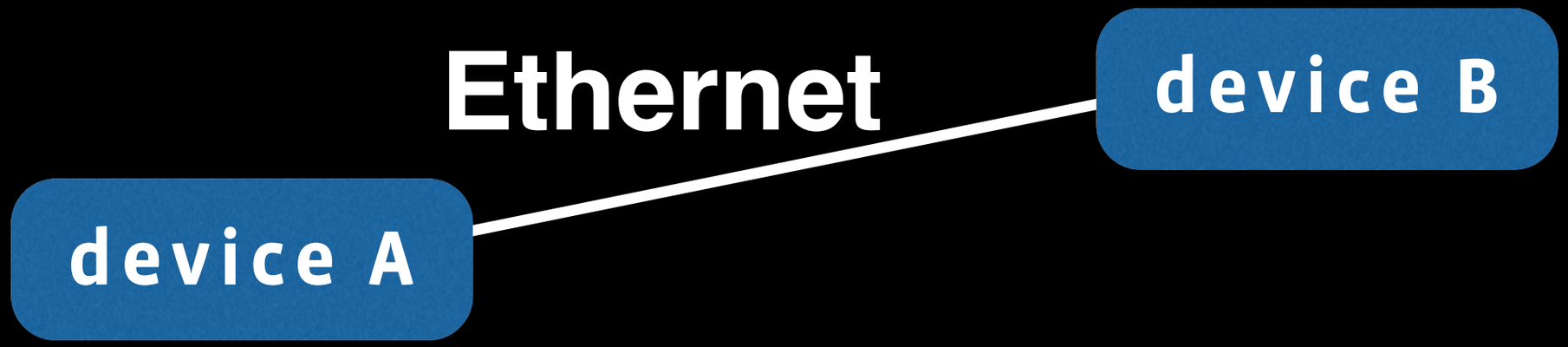
on

building the system

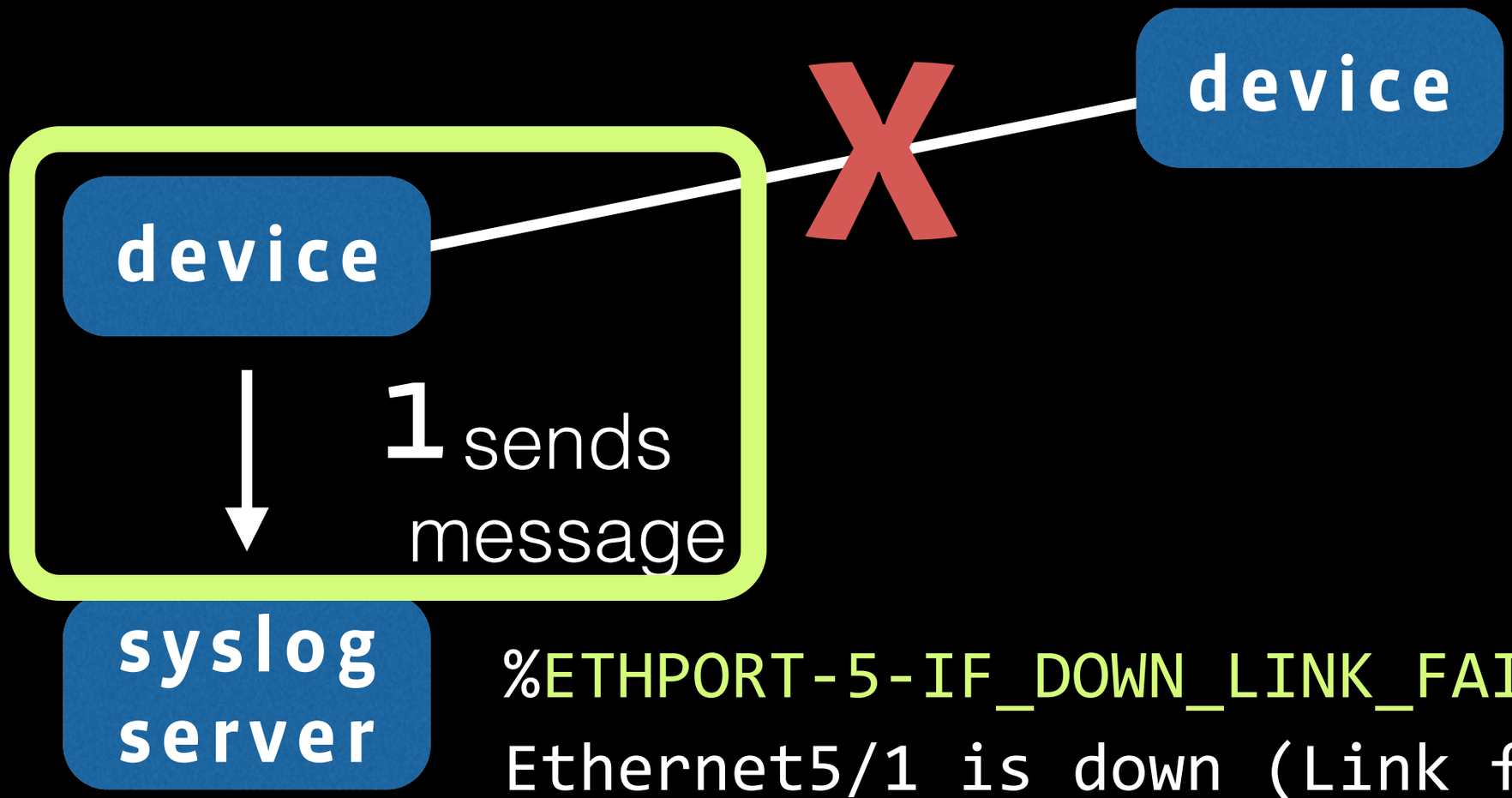
goals

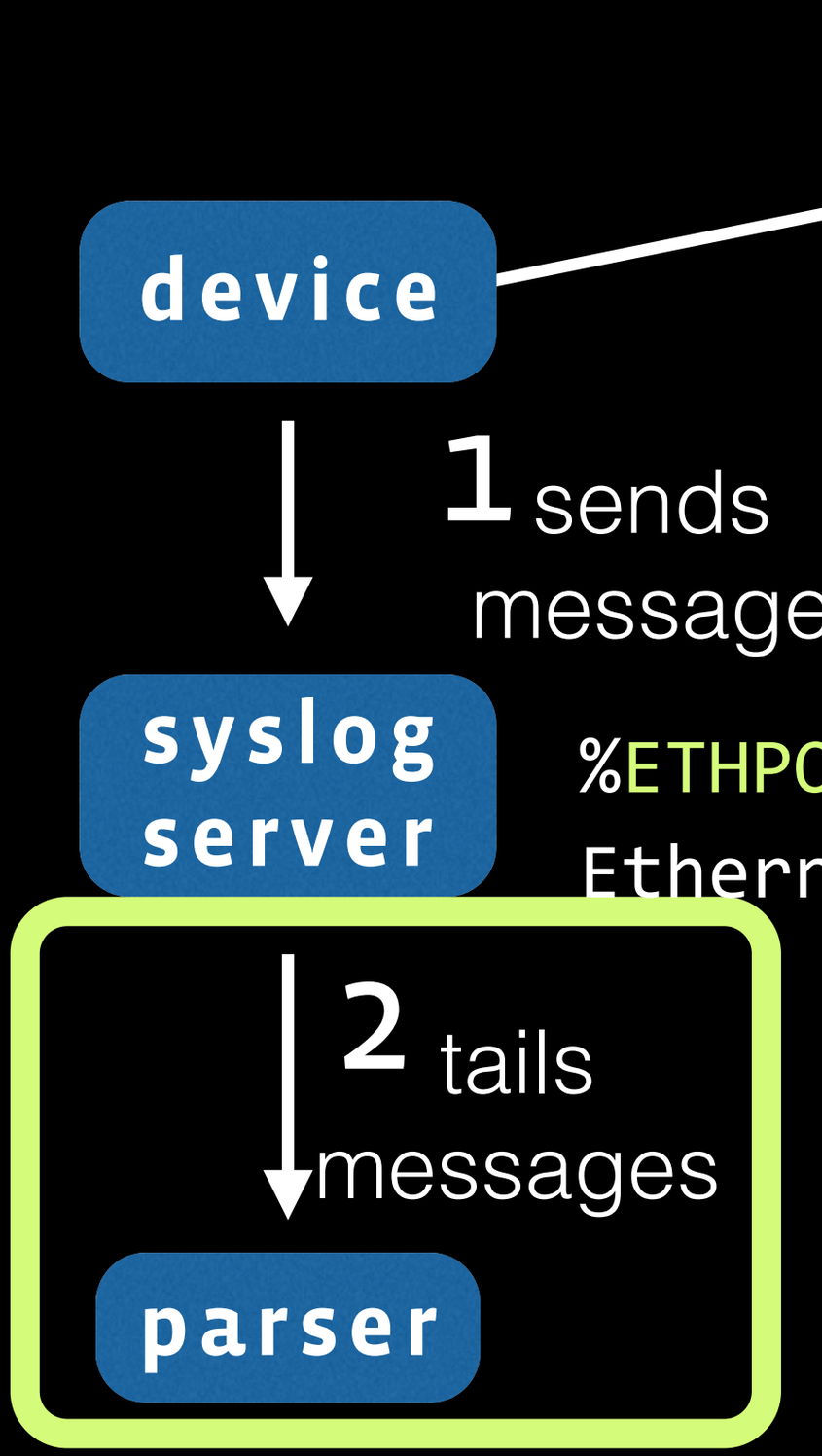
- **organized** to support 100s of remediations
- **fast** enough to react to 100s of events at at time
- **simple** enough to manage without a CS background

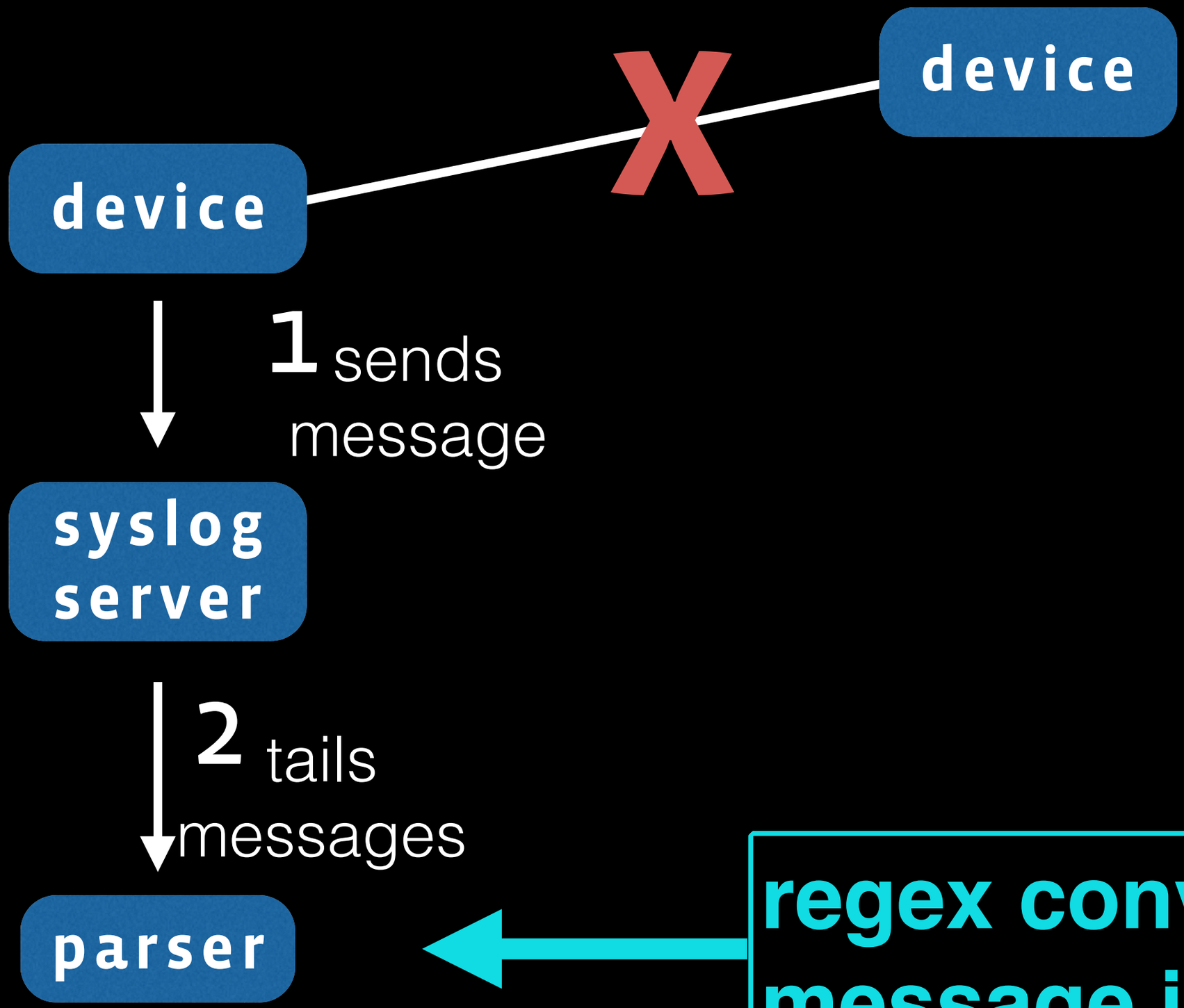
how it works

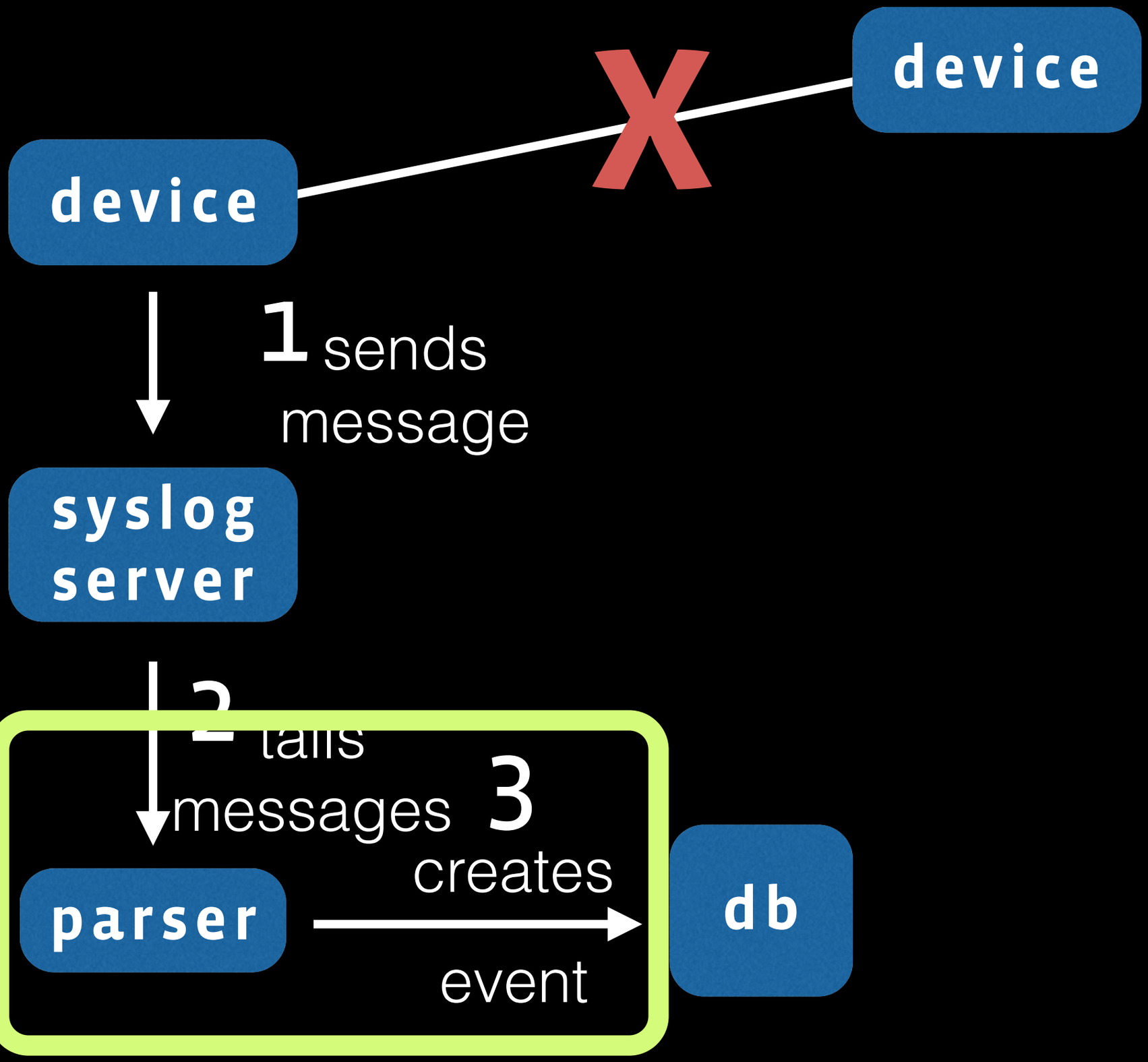

Ethernet
device B
device A

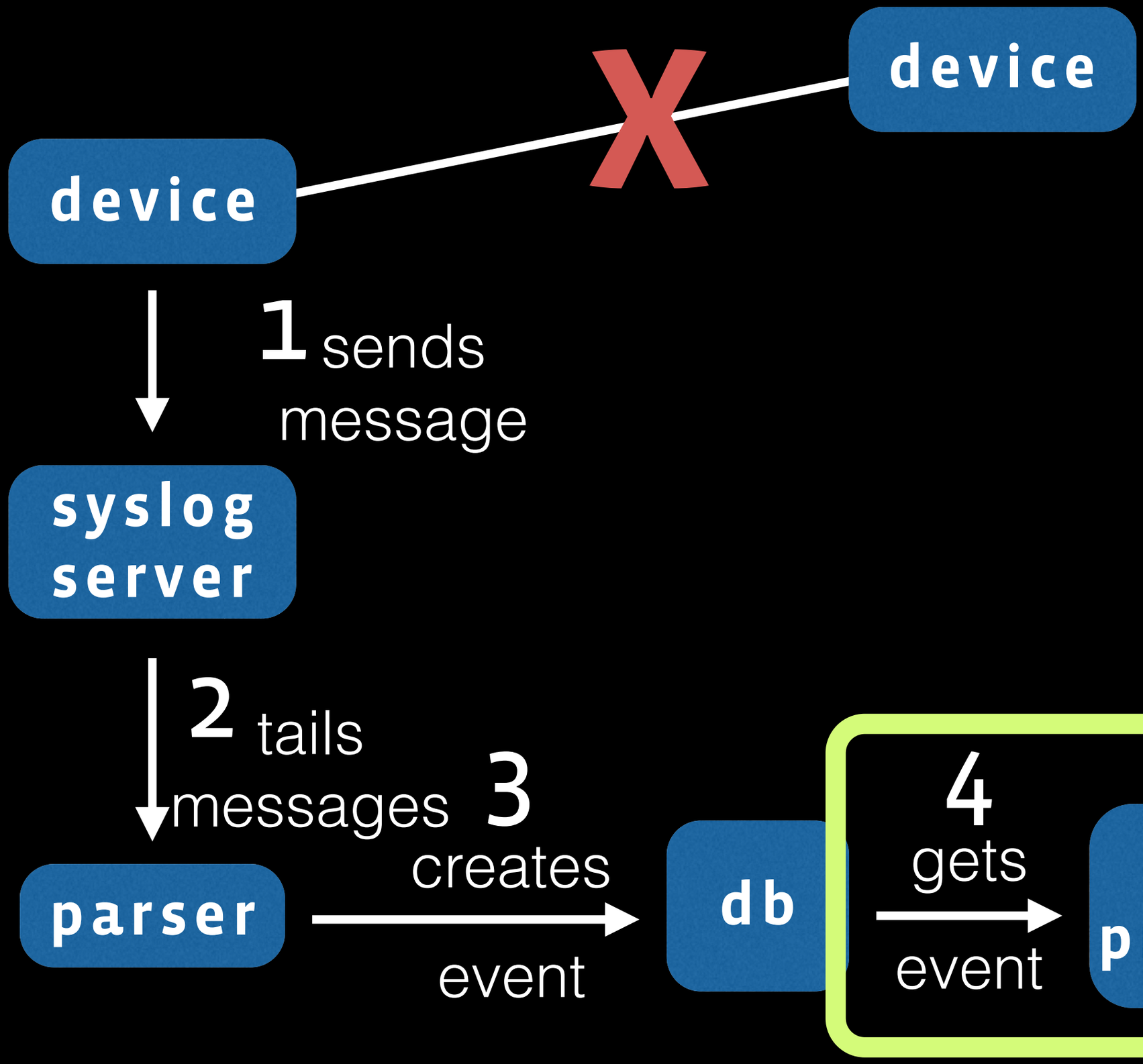

Ethernet
device A
device B
link down!

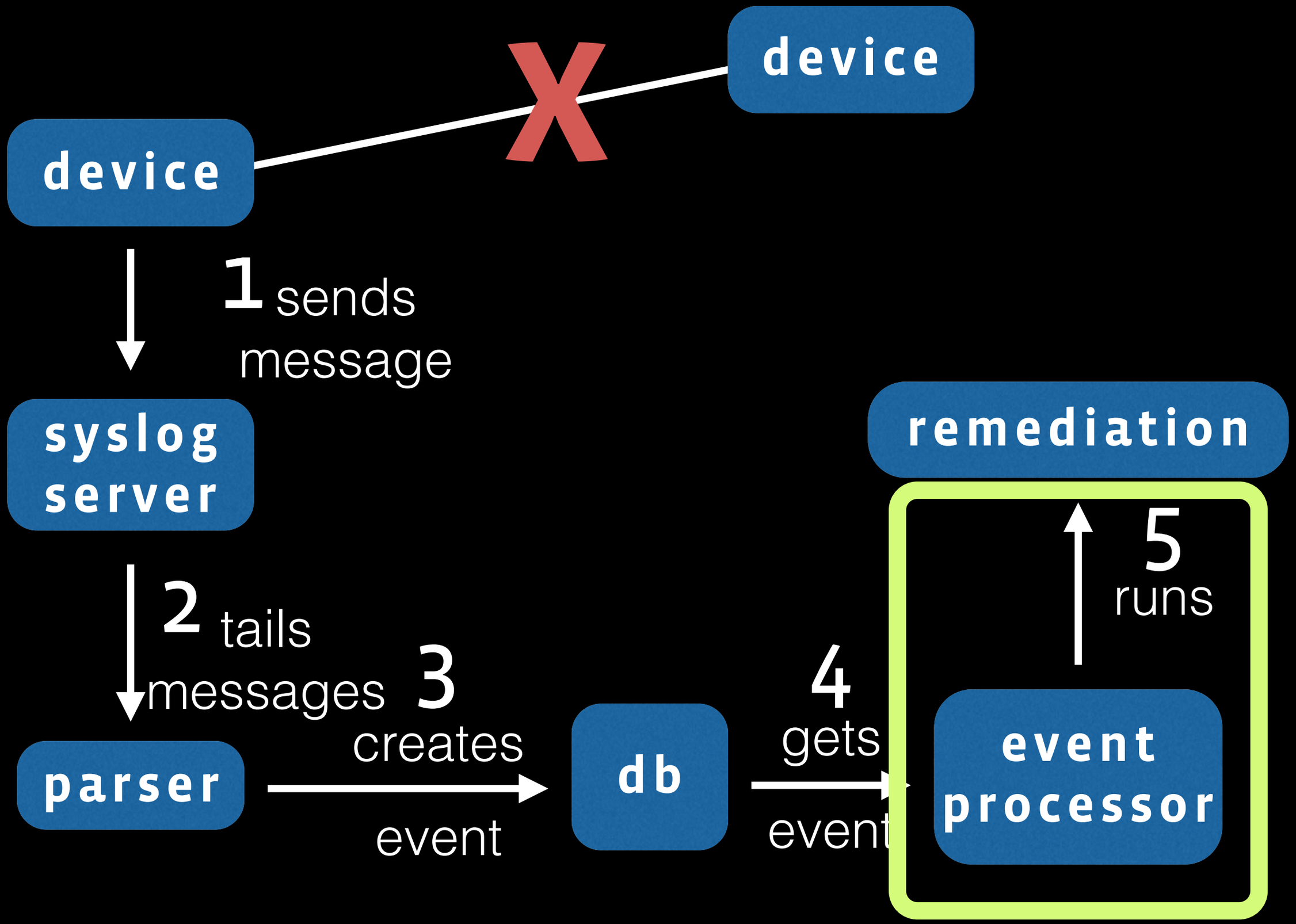

device
X
device
device
1 sends message
syslog server
%ETHPORT-5-IF_DOWN_LINK_FAILURE: Interface Ethernet5/1 is down (Link failure)

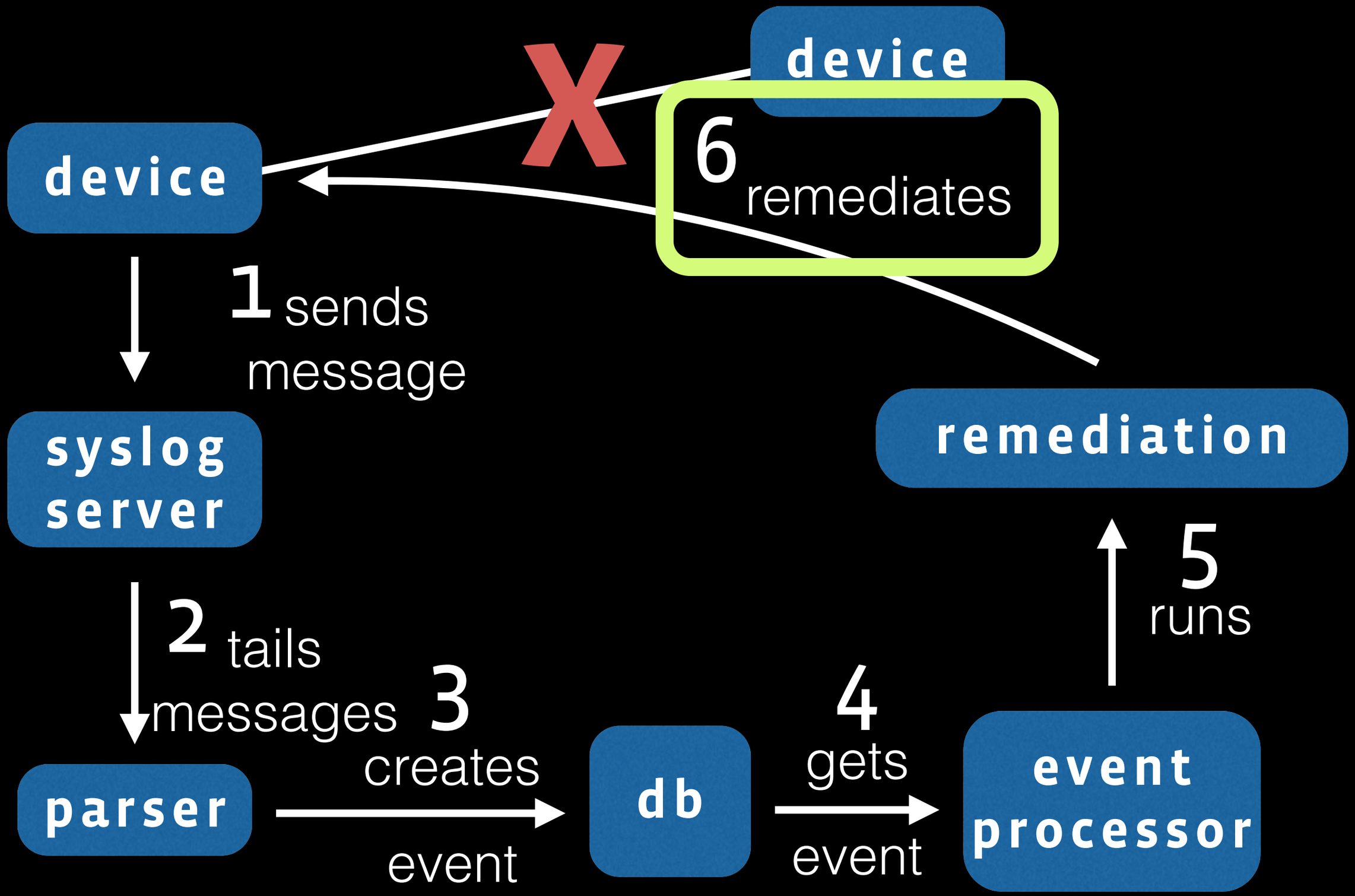

device
device
1 sends message
syslog server
%ETHPORT-5-IF_DOWN_LINK_FAILURE: Interface Ethernet5/1 is down (Link failure)
2 tails messages
parser

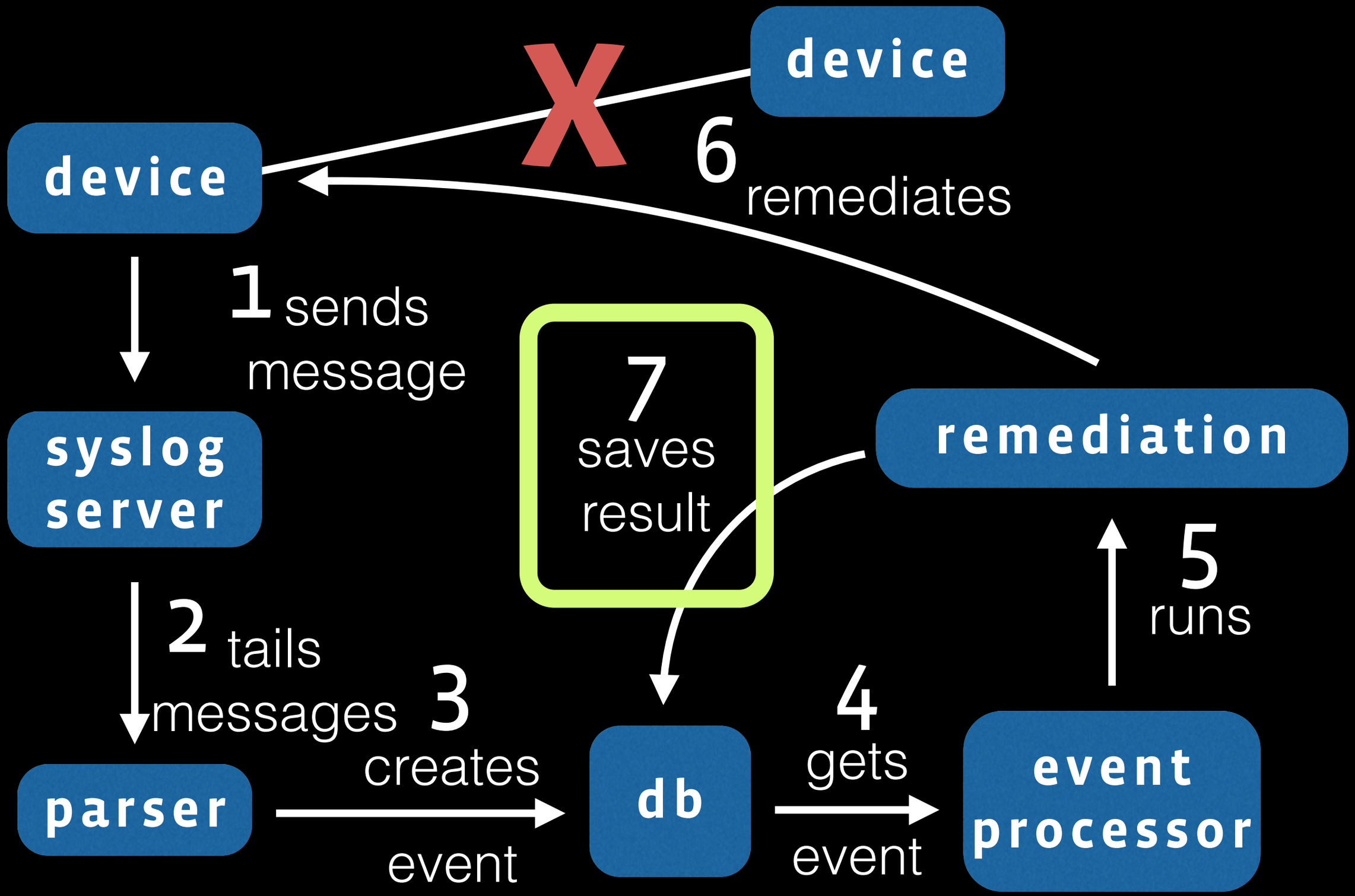

device
device
1 sends message
syslog server
2 tails messages
parser
regex converts message into data


device
device
1 sends message
syslog server
2 tails messages
3 creates event
parser
db

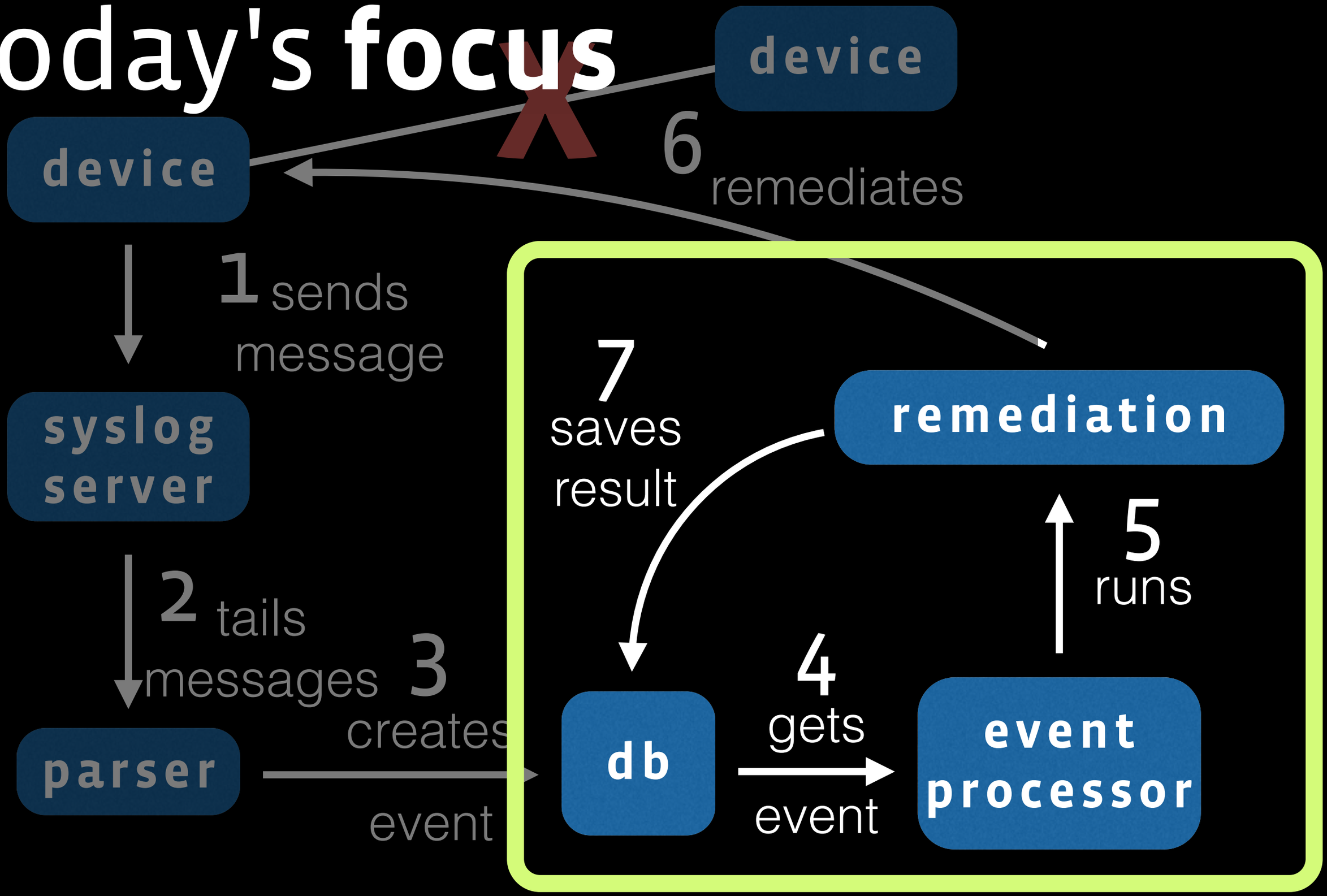

device
device
1 sends message
syslog server
2 tails messages
3 creates event
parser
db
4 gets event
event processor


device
device
1 sends message
syslog server
remediation
5 runs
2 tails messages
3 creates event
parser
db
4 gets event
event processor


device
X
device
6
remediates
1
sends
message
syslog
server
remediation
5
runs
2
tails
messages
3
creates
event
parser
db
4
gets
event
event
processor


device
X
device
6 remediates
device
1 sends message
syslog server
7 saves result
remediation
5 runs
2 tails messages
3 creates event
parser
db
4 gets event
event processor

today's **focus**



demo!

syslog generator

```
In [1]: run syslog_generator.py --continuous
Output filename:  /tmp/messages
```

syslog generator

```
In [1]: run syslog_generator.py --continuous
Output filename:  /tmp/messages
```

```
In [2]: cat /tmp/messages
```

syslog generator

```
In [1]: run syslog_generator.py --continuous
Output filename:  /tmp/messages
```

```
In [2]: cat /tmp/messages
```

```
2016-01-25 06:45:34.699452  cpe0089.lga.example.com(ve
or2)    LINEPROTO-5-UPDOWN:Line protocol on Interface
rt-Channel101 (port-channel to xyz), changed state to
wn
```

syslog generator

```
In [1]: run syslog_generator.py --continuous
Output filename:  /tmp/messages
```

```
In [2]: cat /tmp/messages
```

```
2016-01-25 06:45:34.699452  cpe0089.lga.example.com(ve
or2)    LINEPROTO-5-UPDOWN:Line protocol on Interface
rt-Channel101 (port-channel to xyz), changed state to
wn
```

```
2016-01-25 06:45:34.699486  cpe0075.sjc.example.com(ve
or2)    PWRMGMT-4-INPUT_POWER_OK:PowerSupply2 has rega
ed input power.
```

syslog parser

```
In [16]: run parser.py
```

syslog parser

```
In [16]: run parser.py
```

```
Created event 1695:  status=QUEUED, timestamp=145370433
, device=cpe0075.sjc.example.com(vendor2), error_code=P
RMGMT-4-INPUT_POWER_OK, error_message=PowerSupply2 has
egained input power., result=UNKNOWN
```

syslog parser

```
In [16]: run parser.py
```

```
Created event 1695:  status=QUEUED, timestamp=145370433
, device=cpe0075.sjc.example.com(vendor2), error_code=P
RMGMT-4-INPUT_POWER_OK, error_message=PowerSupply2 has
egained input power., result=UNKNOWN
```

```
Created event 1696:  status=QUEUED, timestamp=145370433
, device=cpe0089.lga.example.com(vendor2), error_code=L
NEPROTO-5-UPDOWN, error_message=Line protocol on Interf
ce Port-Channel101 (port-channel to xyz), changed state
to Down, result=UNKNOWN
```

event processor

```
In [8]: run processor.py
```

```
[Thread-141][EventID:157]  rtr0041.sjc.exampl     SUCCESS
    remediation:bgp_adjacency    time:2s
```

event processor

```
In [8]: run processor.py
```

```
[Thread-141][EventID:157]  rtr0041.sjc.exampl     SUCCESS
    remediation:bgp_adjacency    time:2s
[Thread-172][EventID:188]  sw0044.iad.example     FAILURE
    remediation:lsp_change     time:2s
```

event processor

```
In [8]: run processor.py
```

```
[Thread-141][EventID:157]  rtr0041.sjc.exampl     SUCCESS
    remediation:bgp_adjacency    time:2s
[Thread-172][EventID:188]  sw0044.iad.example     FAILURE
    remediation:lsp_change     time:2s
[Thread-181][EventID:197]  sw0076.den.example     FAILURE
    remediation:lsp_change     time:2s
```

event processor

```
In [8]: run processor.py
```

```
[Thread-141][EventID:157]  rtr0041.sjc.exampl     SUCCESS
    remediation:bgp_adjacency    time:2s
[Thread-172][EventID:188]  sw0044.iad.example     FAILURE
    remediation:lsp_change     time:2s
[Thread-181][EventID:197]  sw0076.den.example     FAILURE
    remediation:lsp_change     time:2s
[Thread-185][EventID:201]  sw0062.lga.example     FAILURE
    remediation:bgp_adjacency    time:2s
```

event processor

```
In [8]: run processor.py
```

```
[Thread-141][EventID:157]  rtr0041.sjc.exampl     SUCCESS
    remediation:bgp_adjacency    time:2s
[Thread-172][EventID:188]  sw0044.iad.example     FAILURE
    remediation:lsp_change    time:2s
[Thread-181][EventID:197]  sw0076.den.example     FAILURE
    remediation:lsp_change    time:2s
[Thread-185][EventID:201]  sw0062.lga.example     FAILURE
    remediation:bgp_adjacency    time:2s
[Thread-113][EventID:143]  sw0035.lga.example     FAILURE
    remediation:default    time:1s
```

the lab environment

a virtual machine of

Ubuntu desktop

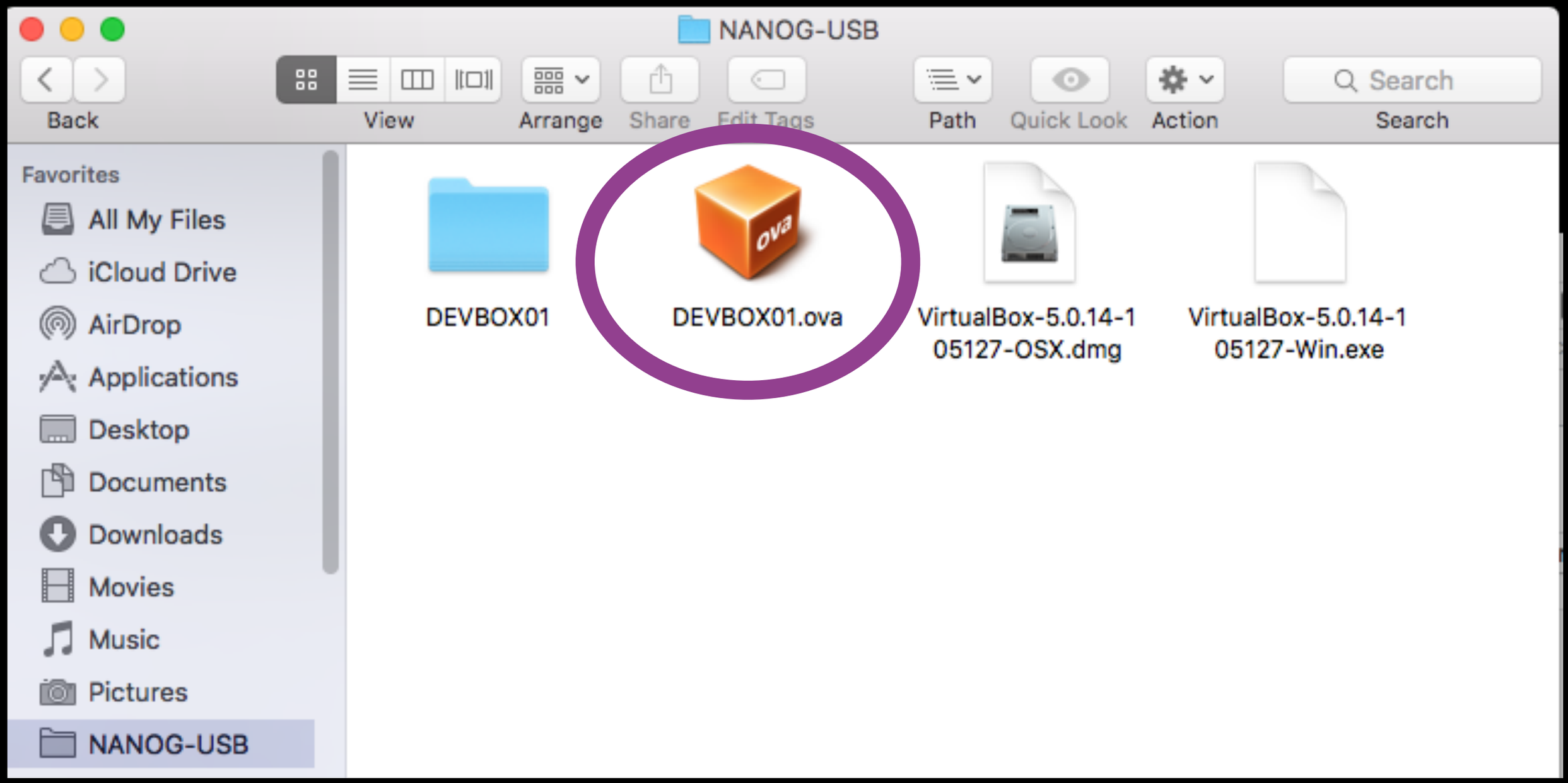
customized with

iPython!

and extras

Python 2.7.11 & 3.5.1
MySQL server & Python libs

importing the virtual appliance



download it from netengcode.com!

VirtualBox File Machine Window Help

Oracle VM VirtualBox Manager

New Settings Start Discard

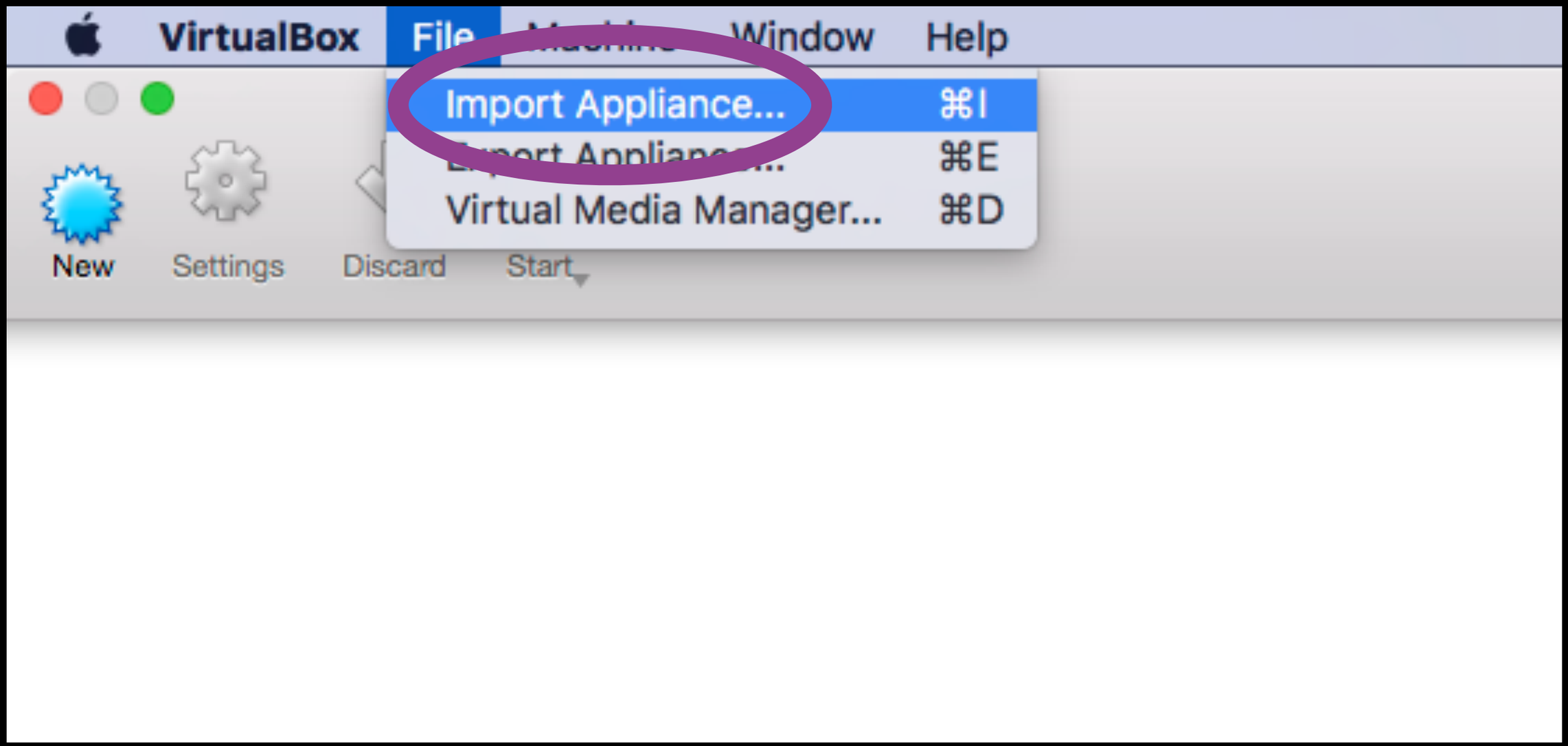
Details Snapshots

Welcome to VirtualBox!

The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the ⌘**?** key to get instant help, or visit www.virtualbox.org for the latest information and news.


VirtualBox
File
Window
Help
Import Appliance... ⌘I
⌘E
Virtual Media Manager... ⌘D
New
Settings
Discard
Start

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

48 seconds remaining

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

Oracle VM VirtualBox Manager

New Setting Start Discard

Details Snapshots

DEVBOX01
Powered Off

General

Name: DEVBOX01
Operating System: Ubuntu (64 bit)

System

Base Memory: 1024 MB
Boot Order: CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging

Preview

DEVBOX01

Display

Video Memory: 128 MB
Acceleration: 3D
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Audio

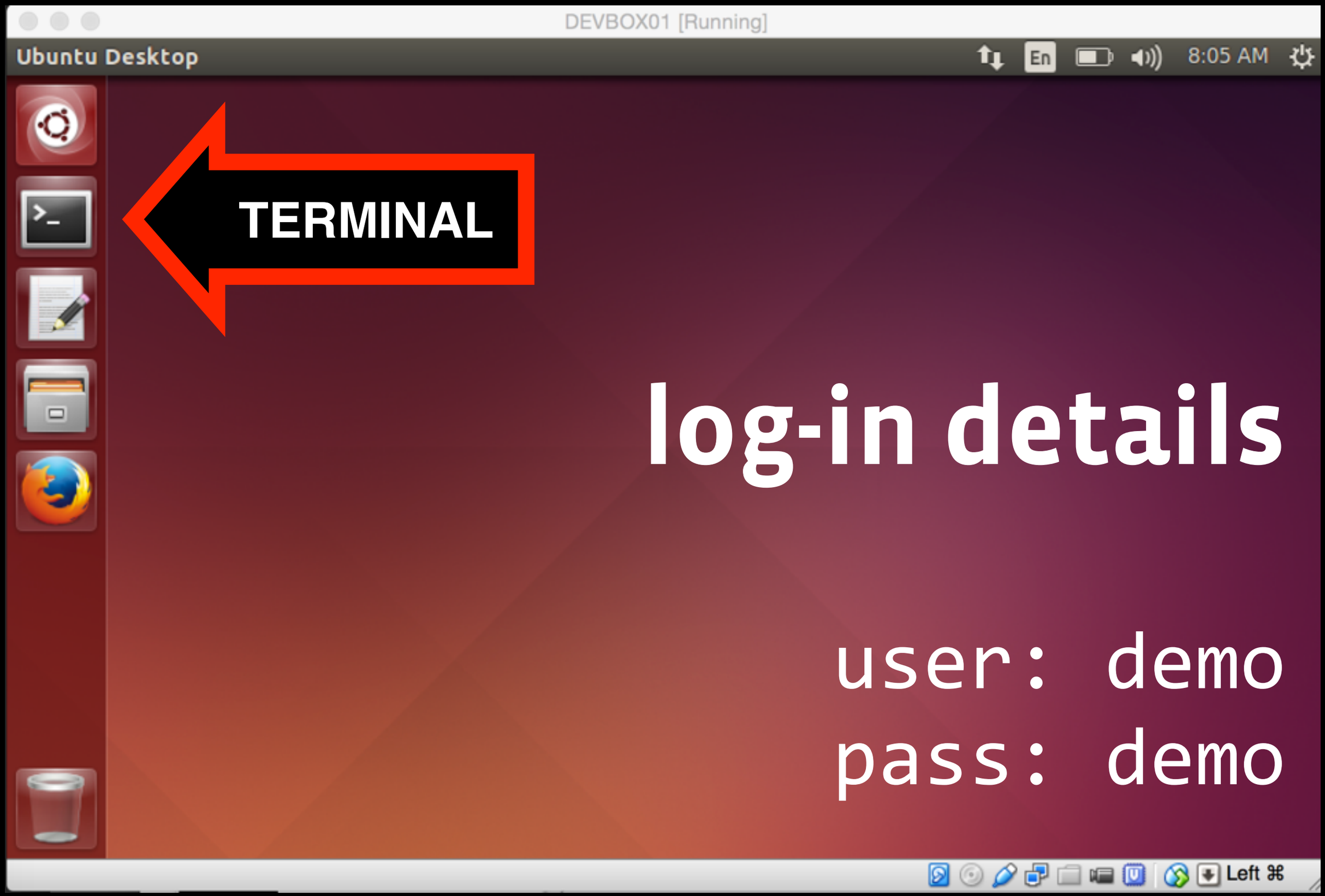
Disabled

Network

Adapter 1: Intel PRO/1000 MT Desktop (NAT)

USB

Device Filters: 0 (0 active)


DEVBOX01 [Running]
Ubuntu Desktop
En
8:05 AM
TERMINAL
log-in details
user: demo
pass: demo
Left ⌘

keyboard shortcuts to break out of the VM

enter or leave full-screen

getting started

organizing

our data

the event as a tuple

```
In [23]: print(m.groups())
('2015 Apr  2', '14:25:06', 'swi
-IF_DOWN_INTERFACE_REMOVED', 'In
/1 is down (Interface removed)')
```

the event as a tuple

```
In [23]: print(m.groups())
('2015 Apr  2', '14:25:06', 'swi
-IF_DOWN_INTERFACE_REMOVED', 'In
/1 is down (Interface removed)')
```

```
date, t_stamp, device, \
    err_code, err_msg = m.groups()

date
'2015 Apr  2'
```

what's the problem?

```
date, t_stamp, device, \
    err_code, err_msg = m.groups()
```

what's the problem?

new field!

```
date, t_stamp, status, device, \
    err_code, err_message = m.groups()
```

```
ValueError: not enough values to unpack
```

data formats **will** change

the event as a dictionary

```
In [40]: pprint.pprint(e)
{'device': 'switch1',
 'error_code': 'IF_DOWN_LINK_FAILUR
 'error_message': 'Interface Ethern
(link failure)',
 'timestamp': 1453687499.7067747}
```

the event as an object

```
e = Event(
    timestamp = time.time(),
    device = 'switch1',
    error_code = 'IF_DOWN_LINK_FAILURE',
    error_message = (
        'Interface Ethernet5/1 is down')
)
```

link

accessing attributes of an object

```
e = Event(
    timestamp = time.time(),
    device = 'switch1',
    error_code = 'IF_DOWN_LINK_FAILURE'
```

```
In [35]: e.device
Out[35]: 'switch1'

In [36]: e.error_code
Out[36]: 'IF_DOWN_LINK_FAILURE'
```

link

which to choose?

the advantage of an object

```
class Event:
    def __init__(self, timestamp,
                 device, error_code,
                 error_message):
        self.timestamp = timestamp
        self.device = device
        self.error_code = error_code
        self.error_message = error_message
```

enforcement of structure

the advantage of an object

```
class Event:
    def __init__(self, timestamp,
                 device, error_code,
                 error_message):
        self.timestamp = timestamp
        self.device = device
        self.error_code = error_code
        self.error_message = error_message
```

input validation (not shown)

where to find it?

```
netfbar
├── lib
│   ├── db.py:Event()
```

what's already staged?

- a database (MySQL)

- an events table

- a library for managing our data

- installation of "netfbar"
  (a Python package)

database setup

```
├── scripts
│   └── dbsetup.sh
```

```
Resetting the netfbar database...
Created a database named netfbar successf
Created a user named netfbar_rw with pass
successfully!
Created the events table on netfbar succe
Done!
```

netfbar package

```
nanog_demo_201
└── setup.py
```

```
demo@DEVBOX01:~/nanog_demo_201$ \
> sudo python3 setup.py install
```

the parser

where to find it?

```
netfbar
├── parsing_engine
│   ├── parser.py
│   └── regexes.py
```

remediations,
refactored.

a few problems

- duplication
- no structure
- limited debug-ability

organizing our remediations

"sparse is better than dense"

```
netfbar
├── lib
│   ├── remediations
│   │   ├── base.py
│   │   ├── bgp_adjacency.py
│   │   ├── linecard_failure.py
│   │   ├── link_failure.py
│   │   └── lsp_change.py
```

1 remediation, 1 file.

remediations, v2.

code that repeats in each remediation

```
...
module_number = interface.group(1)
command = ('show module {module_number}'.format(
            module_number=module_number))

# Open an SSH connection to the device
ssh = ssh_helper.SSHSession(
    device=device_name,
    username='',
    passwd='')

# Fetch output of "show module"
show_module = ssh.write([command])
```

remediations, v2.

```
class BaseRemediation:

    def __init__(self, event):
        self.event = event
        self.ssh_session = None
```

moved to a base remediation

```
def run_cmd(self, cmd):
    if not self.ssh_session:
        self.ssh_session = ssh_helper.SSHSession(
            device=self.event.device,
            username='',
            passwd='')
    output = self.ssh_session.write([cmd])
    return output
```

remediations, v2.

```
class Remediation(base.BaseRemediation):

    def run(self):
        """  Interface Link Down Remediation """
        interface = interface.group(1)
        command = 'show interface eth {interface}'.format

        # Fetch output of "show interface"
        show_interface = self.run_cmd(command)
```

from base.BaseRemediation

connecting remediations to error codes

```
from netfbar.lib.remediations import \
    default, \
    bgp_adjacency, linecard_failure, link_failure, lsp_cl
```

```
ERROR_CODES_TO_REMEDIATIONS = {
    'BGP-5-ADJCHANGE': bgp_adjacency,
    'BGP-3-NOTIFICATION': bgp_adjacency,
    'MODULE-5-MOD_OK': linecard_failure,
    'ETHPORT-5-IF_DOWN_INTERFACE_REMOVED': linecard_failu
    'ETHPORT-5-IF_DOWN_LINK_FAILURE': link_failure,
```

connecting remediations to error codes

```
from netfbar.lib.remediations import \
    default, \
    bgp_adjacency, linecard_failure, link_failure, lsp_cl
```

```
ERROR_CODES_TO_REMEDIATIONS = {
    'BGP-5-ADJCHANGE': bgp_adjacency,
    'BGP-3-NOTIFICATION': bgp_adjacency,
    'MODULE-5-MOD_OK': linecard_failure,
    'ETHPORT-5-IF_DOWN_INTERFACE_REMOVED': linecard_failu
    'ETHPORT-5-IF_DOWN_LINK_FAILURE': link_failure,
```

a dictionary of "error_code" to "module"

connecting remediations to error codes

```
ERROR_CODES_TO_REMEDIATIONS = {
    'BGP-5-ADJCHANGE': bgp_adjacency,
    'BGP-3-NOTIFICATION': bgp_adjacency,

def get_remediation(error_code):
    """ Returns the remediation module associated with the error
    provided.  If there is not one associated, a default reme
    is returned instead.
"""

    # Locate the module
    if error_code in ERROR_CODES_TO_REMEDIATIONS:
        module = ERROR_CODES_TO_REMEDIATIONS[error_code]
    else:
        module = DEFAULT_REMEDIATION
```

where to find it?

```
netfbar
├── lib
│   ├── remediations
│   │   ├── linecard_failure.py
│   │   ├── link_failure.py
│   │   └── mapping.py
```

questions?

building the event processor

starting simple

building it live

```
In [2]: from netfbar.lib import db
```

building it live

```
In [6]: status = db.Event.STATUS_CODES[
   ...:     'QUEUED']
```

building it live

```
In [6]: status = db.Event.STATUS_CODES[
   ...:          'QUEUED']

In [7]: with db.Db() as db_conn:
```

building it live

```
In [6]: status = db.Event.STATUS_CODES[
   ...:     'QUEUED']

In [7]: with db.Db() as db_conn:
   ...:     events = (
   ...:         db_conn.get_events_by_status(
   ...:             status))
   ...:
```

building it live

```
In [6]: status = db.Event.STATUS_CODES[
   ...:     'QUEUED']

In [7]: with db.Db() as db_conn:
   ...:     events = (
   ...:         db_conn.get_events_by_status(
   ...:             status))
   ...:
```

```
In [9]: print(len(events))
340
```

no events?

```
In [9]: print(len(events))
340
```

building it live

```
In [10]: event = events[0]

In [11]: print(event)
event_id=1, status=QUEUED, timestamp=145370433
 device=rtr0098.atl.example.com(vendor3), erro
code=ETHPORT-5-IF_BANDWIDTH_CHANGE, error_mess
e=Interface port-channel101,bandwidth changed
 100000 Kbit, result=UNKNOWN
```

building it live

```
In [3]: from netfbar.lib.remediations \
   ...:     import mapping
```

building it live

```
In [3]: from netfbar.lib.remediations \
   ...:     import mapping
```

```
In [17]: help(mapping)
```

building it live

```
NAME
    netfbar.lib.remediations.mapping - Remediation mapping

FUNCTIONS
    get_remediation(error_code)
        Returns the remediation module associated with the error_c
        provided.  If there is not one associated, a default remed
        is returned instead.
```

building it live

```
In [32]: event.error_code
Out[32]: 'ETHPORT-5-IF_BANDWIDTH_CHANGE'
```

building it live

```
In [32]: event.error_code
Out[32]: 'ETHPORT-5-IF_BANDWIDTH_CHANGE'
```

```
In [33]: cls = mapping.get_remediation(
   ....:         event.error_code)
```

building it live

```
In [32]: event.error_code
Out[32]: 'ETHPORT-5-IF_BANDWIDTH_CHANGE'

In [33]: cls = mapping.get_remediation(
    ....:         event.error_code)

In [34]: remediation = cls(event)
```

building it live

```
In [32]: event.error_code
Out[32]: 'ETHPORT-5-IF_BANDWIDTH_CHANGE'
```

```
In [33]: cls = mapping.get_remediation(
    ....:         event.error_code)

In [34]: remediation = cls(event)
```

```
In [36]: print(remediation)
default
```

building it live

```
In [40]: result = remediation.run()
```

building it live

```
In [40]: result = remediation.run()

In [41]: print(result)
True
```

(no logging in the default remediation)

building it live

```
In [43]: def process_event(event):
    ....:     cls = mapping.get_remediation(
    ....:         event.error_code)
    ....:     remediation = cls(event)
    ....:     result = remediation.run()
    ....:     return result
    ....:
```

create a function for handling one event

building it live

```
In [44]: for event in events:
    ...:         result = process_event(event)
    ...:         print(event.device,
    ...:               event.error_code,
    ...:               result)
```

iterating through events

building it live

```
cpe0075.iad.example.com(vendor5) RPD_BGP
R_STATE_CHANGED True
cpe0035.atl.example.com(vendor5) UI_COMM
sw0001.iad.example.com(vendor5) BGP_CONN
ED:bgp_connect_start False
cpe0010.iad.example.com(vendor2) PWRMGMT
 POWER_OK True
```

so slow!

questions?

making this a little faster

so many choices!

- threading
- multiprocessing
- asyncio
- gevent
- go...

starting simple!

threading & multiprocessing

threading's pros

- simple API
- lightweight
- shared memory with the parent

https://docs.python.org/3/library/threading.html

threading's cons

- not true parallelism
- all threads are limited to a single CPU - the parent's

https://docs.python.org/3/library/threading.html

threads are evil! ....?!

"In CPython, due to the [GIL], only **one thread** can execute Python code at once"

...

"However, threading is still an appropriate model if you want to run multiple **I/O-bound** tasks simultaneously."

https://docs.python.org/3/library/threading.html

multiprocessing's pros

- simple API
- true parallelism using all CPUs

https://docs.python.org/3/library/threading.html

multiprocessing's cons

- based on forking (cloning/ copying the current process)
- heavier memory footprint
- stale memory / state

https://docs.python.org/3/library/threading.html

multiprocessing's cons

- no shared memory with parent

https://docs.python.org/3/library/threading.html

working with threads

working with threads

```
import threading
```

```
In [48]: for event in events:
    ....:     thread = threading.Thread(
    ....:         target=process_event,
    ....:         args=[event])
    ....:     thread.start()
    ....:
```

https://docs.python.org/3/library/threading.html

working with threads

```
import threading
```

```
In [48]: for event in events:
    ...:     thread = threading.Thread(
    ...:         target=process_event,
    ...:         args=[event])
    ...:     thread.start()
    ...:
```

target - the method to run inside the thread

https://docs.python.org/3/library/threading.html

working with threads

```
import threading
```

```
In [48]: for event in events:
    ....:     thread = threading.Thread(
    ....:         target=process_event,
    ....:         args=[event])
    ....:     thread.start()
    ....:
```

args - the values passed as input to process_event()

https://docs.python.org/3/library/threading.html

working with threads

```
import threading
```

```
In [48]: for event in events:
    ....:     thread = threading.Thread(
    ....:         target=process_event,
    ....:         args=[event])
    ....:     thread.start()
    ....:
```

.start() - spins up the thread in the background (the for loop continues)

https://docs.python.org/3/library/threading.html

working with threads

```
import threading
```

```
In [48]: for event in events:
    ....:     thread = threading.Thread(
    ....:         target=process_event,
    ....:         args=[event])
    ....:     thread.start()
    ....:
```

caution! this is dangerous!
(starting an unknown number of threads)

https://docs.python.org/3/library/threading.html

adding visibility to process_event()

```
def process_event(event):
    cls = mapping.get_remediation(
        event.error_code)
    remediation = cls(event)
    t_name = threading.current_thread().name
    print('{}  running remediation:{} on {}'
          .format(t_name, remediation.name,
                  event.device))
    result = remediation.run()
    return result
```

starting our threads again (with visibility)

```
In [48]: for event in events:
    ....:     thread = threading.Thread(
    ....:         target=process_event,
    ....:         args=[event])
    ....:     thread.start()
    ....:
```

https://docs.python.org/3/library/threading.html

starting our threads again (with visibility)

```
Thread-1018  running remediation:default on rtr0023.sjc
.example.com(vendor4)
Thread-1019  running remediation:default on cpe0020.atl
.example.com(vendor2)
Thread-1020  running remediation:default on cpe0098.lga
.example.com(vendor3)
```

oh my!
...that just started 1020 threads!

https://docs.python.org/3/library/threading.html

creating a thread pool

creating a thread pool

```
def event_handler(event_queue):
    while True:
        event = event_queue.get()
        result = process_event(event)
        with db.Db() as db_conn:
            db_conn.update_result(
                event.event_id,
                2)
```

the work to perform

inside a thread

creating a thread pool

```
In [27]: num_threads = 10

In [28]: for num in range(num_threads):
    ....:     thread = threading.Thread(
    ....:         target=event_handler,
    ....:         args=[event_queue])
    ....:     thread.start()
    ....:
```

starting the background threads

creating a thread pool

```
In [23]: threading.active_count()
Out[23]: 12

In [24]: threading.enumerate()
```

looking at active threads

creating a thread pool

```
In [26]: for t in threading.enumerate():
    ...:     print(t)
    ...:
<_MainThread(MainThread, started 139711738705728)>
<HistorySavingThread(IPythonHistorySavingThread, started 139711643440896)>
<Thread(Thread-2, started 139711528040192)>
<Thread(Thread-5, started 139711502862080)>
<Thread(Thread-10, started 139711049885440)>
<Thread(Thread-1, started 139711610705664)>
<Thread(Thread-3, started 139711519647488)>
<Thread(Thread-4, started 139711511254784)>
<Thread(Thread-9, started 139711058278144)>
<Thread(Thread-7, started 139711486076672)>
<Thread(Thread-8, started 139711477683968)>
<Thread(Thread-6, started 139711494469376)>
```

looking at active threads

creating a thread pool

```
with db.Db() as db_conn:
    events = (
        db_conn.get_events_by_status(
            status=1))
```

fetching events that we'll send to the threads

creating a thread pool

```
with db.Db() as db_conn:
    events = (
        db_conn.get_events_by_status(
            status=1))
```

```
In [22]: print(len(events))
1000
```

creating a thread pool

```
In [2]: import queue
```

```
In [3]: event_queue = queue.Queue()
```

creating a queue to pass events to threads

https://docs.python.org/3/library/threading.html

creating a thread pool

```
In [29]: for event in events:
    ....:     event_queue.put(event)
    ....:
```

putting the pieces together

creating a thread pool

```
In [29]: for event in events:
    ....:     event_queue.put(event)
    ....:
```

```
Thread-3 running remediation:default on sw0027.sjc.e
e.com(vendor4)
```

• • •

putting the pieces together

creating a thread pool

```
def get_events():
    with db.Db() as db_conn:
        events = db_conn.get_events_by_status(1
        return events
```

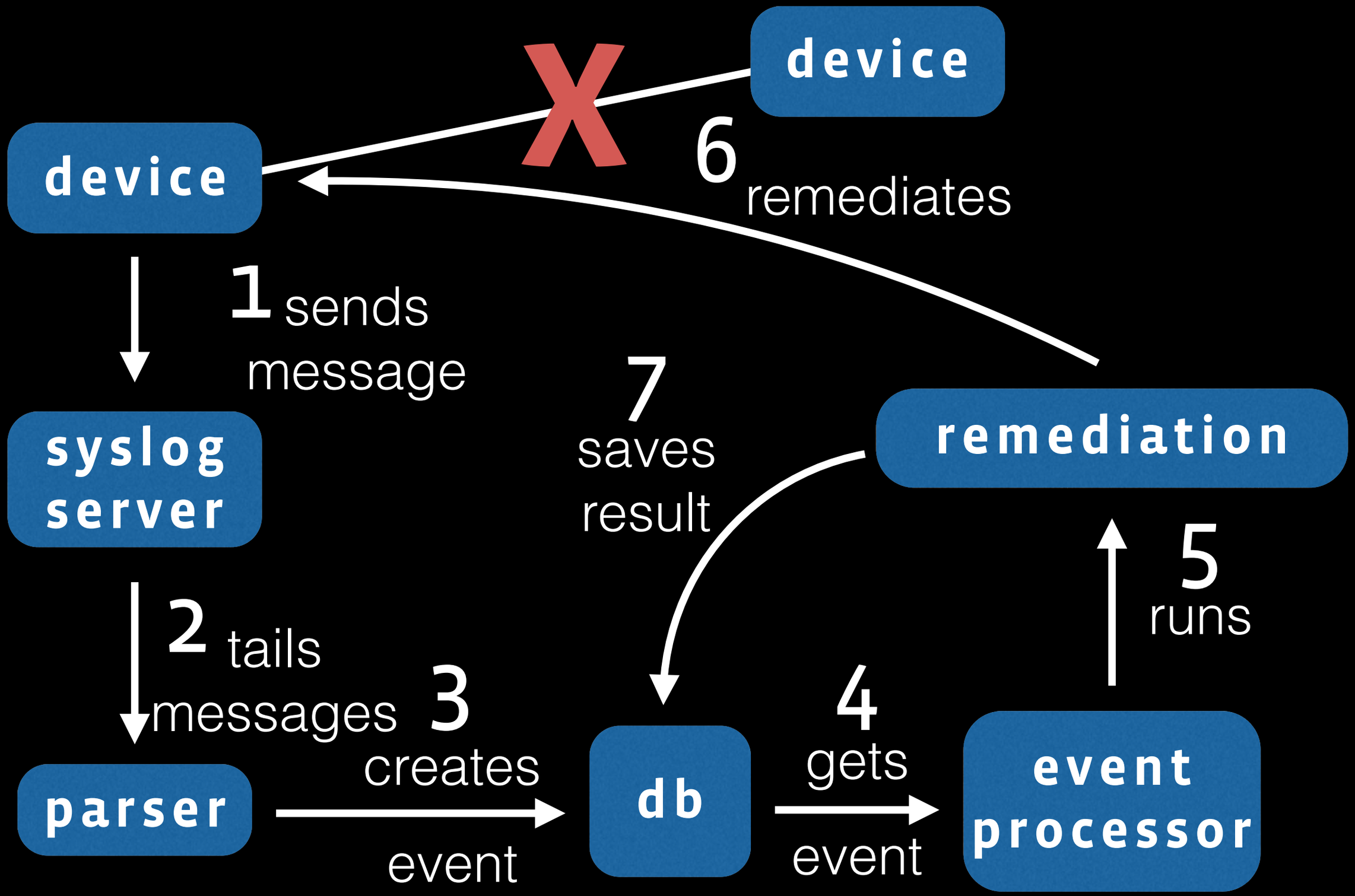
now, continuously!

creating a thread pool

```
while True:
    events = get_events()
    for event in events:
        event_queue.put(event)
```

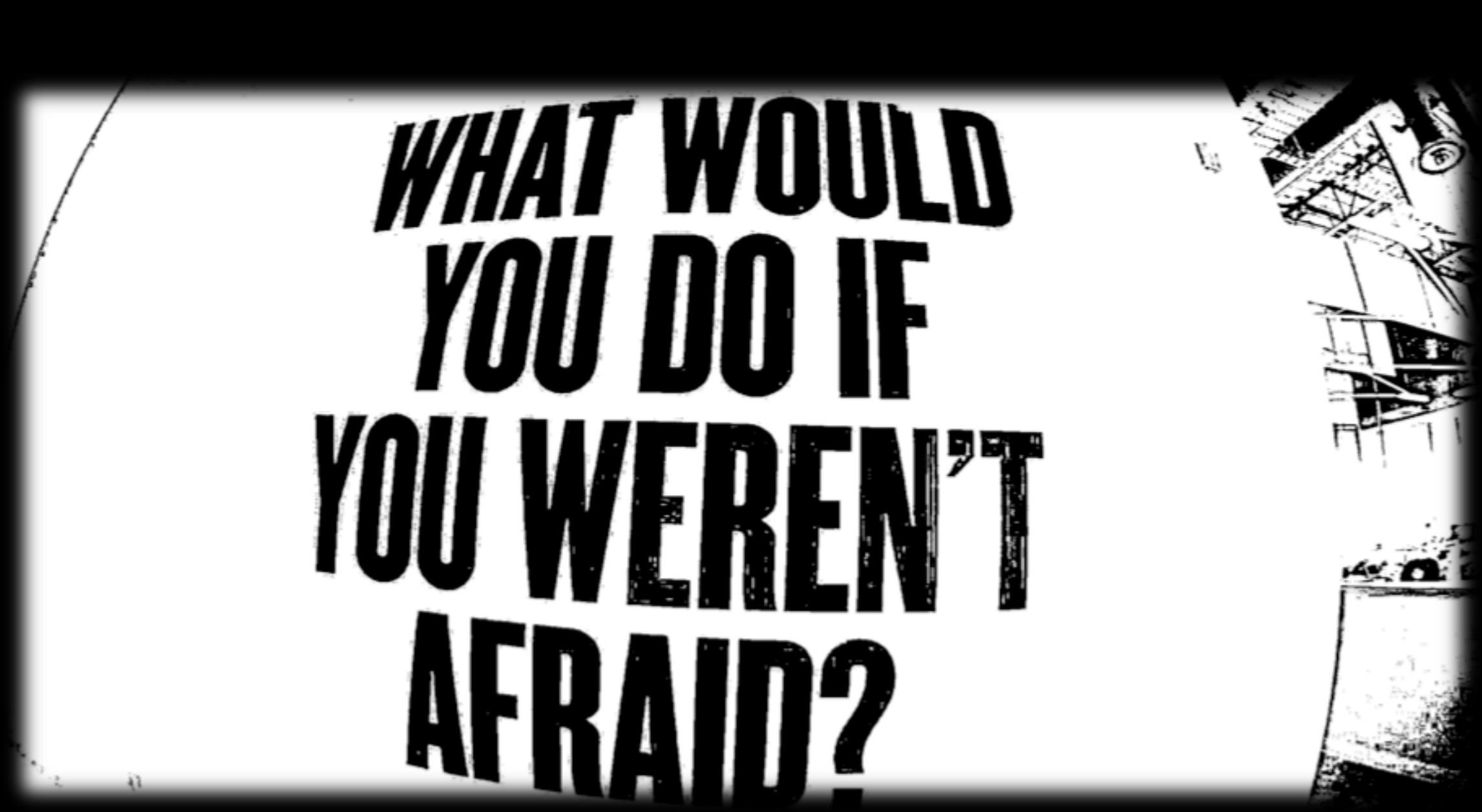
questions?

closing out


device
X
device
6
remediates
1
sends
message
syslog
server
7
saves
result
remediation
5
runs
2
tails
messages
3
creates
event
db
4
gets
event
parser
event
processor

goals

- **organized** to support 100s of remediations
- **fast** enough to react to 100s of events at at time
- **simple** enough to manage without a CS background



#netengcode

david swafford

facebook

NANOG 66

02.10.2016