Analyzing and Enriching Log Data

Cristian Pascariu
Information Security Professional

www.cybersomething.com

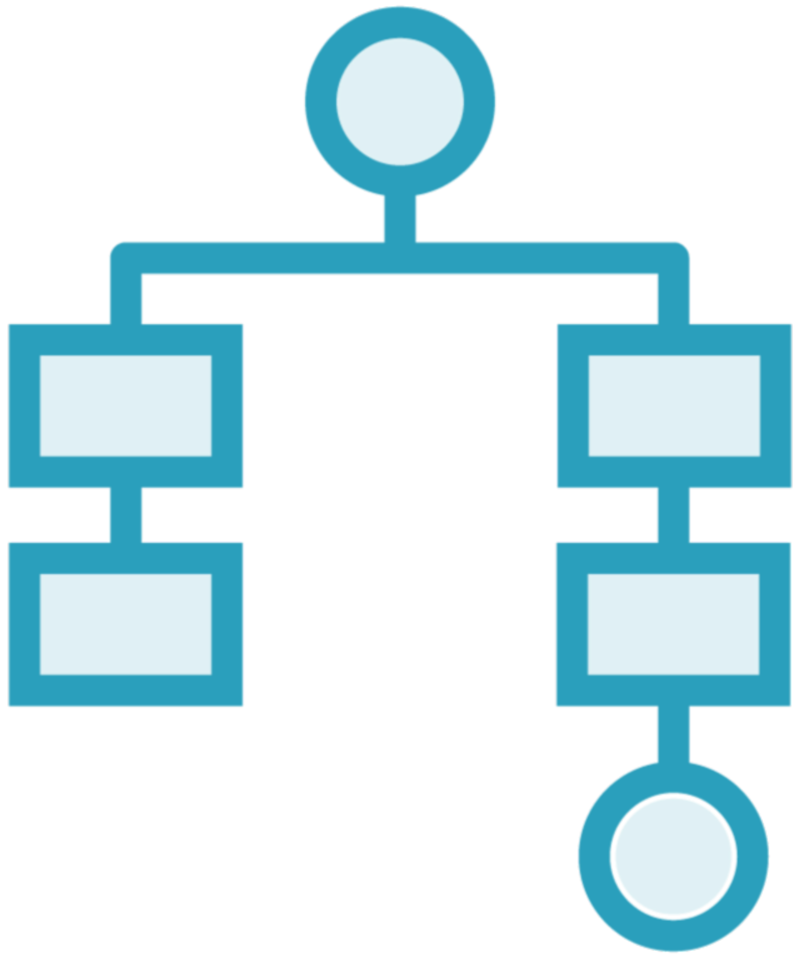
Overview

Log enrichment

Beacon analysis based on log correlation

Perform occurrence and similarity analysis on DNS traffic logs

Plotting log data to discover suspicious patterns



Log Enrichment

Add additional information or context to log data to provide more insight

- Correlate with data from other log sources or services

Log Enrichment in Practice

Suspicious file

Associated with malicious activity

Suspicious domain

Known block list

Suspicious IP

Add GeoIP data

Adding Geolocation Data

Leverage the geoip package to return get the country code based on the IP address

```
from geoip import geolite2

r['country'] = geolite2.lookup(r['8.8.8.8']).country
```

```
> print(r['country'])
"US"
```

Demo

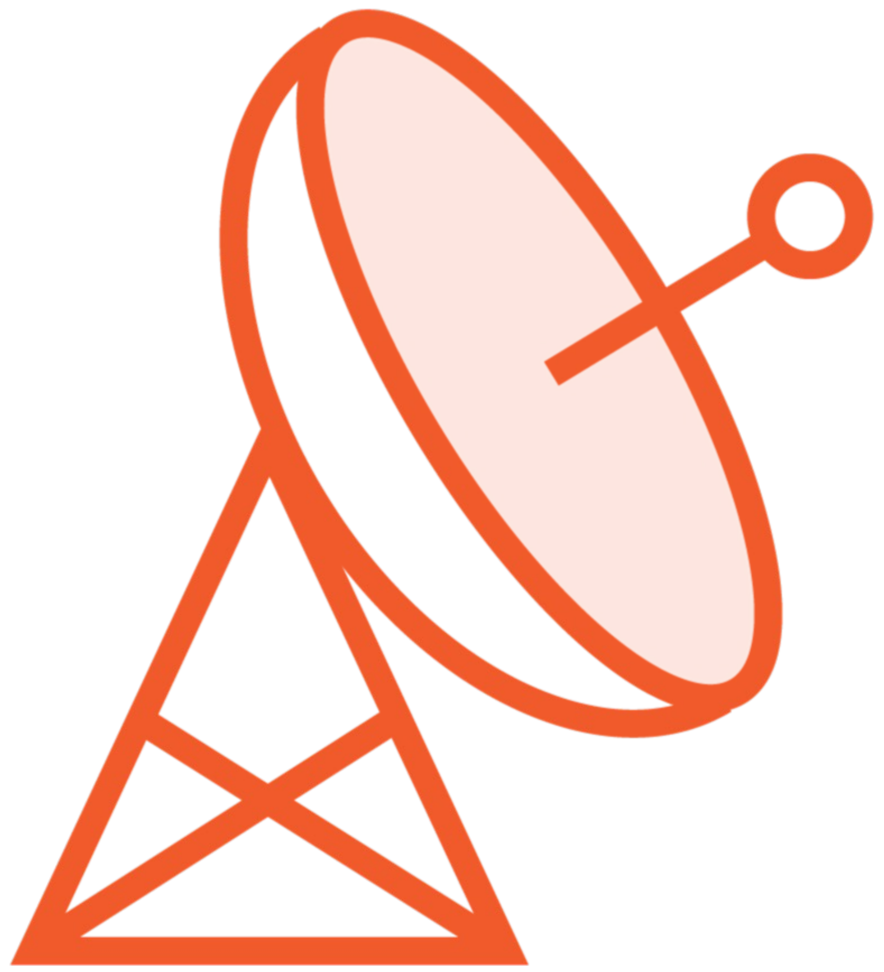
Install the geoip package

Update the parsing function

- Add country code based on IP address

DEMO 3.1 Script

Detect Beaconing



Attackers maintain control of a compromised system via a command-and-control channel

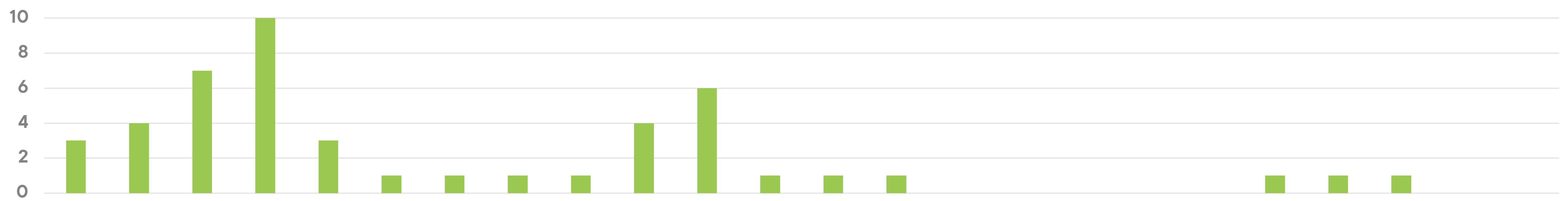
Periodically check in for additional commands

- This is also known as beaconing

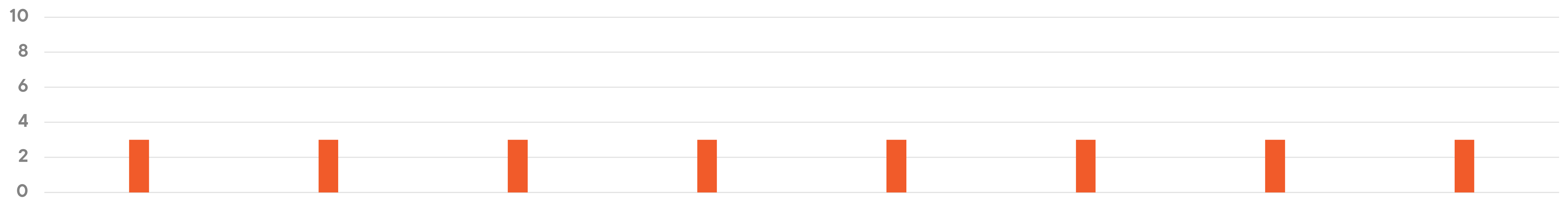
C2 can be established over legitimate protocols and services

Beaconing Characteristics

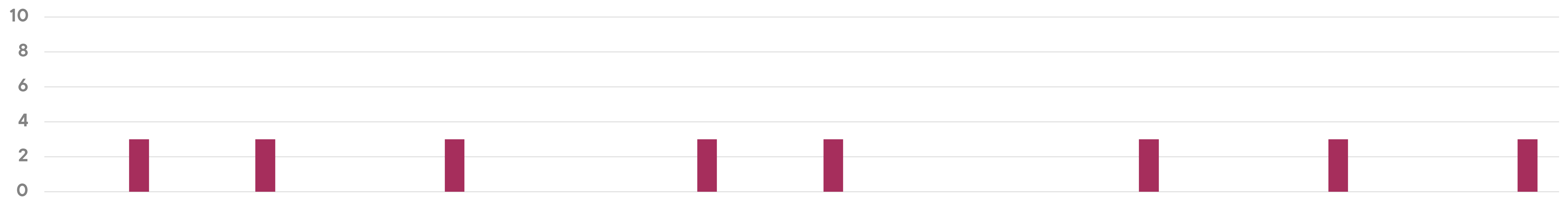
Normal traffic



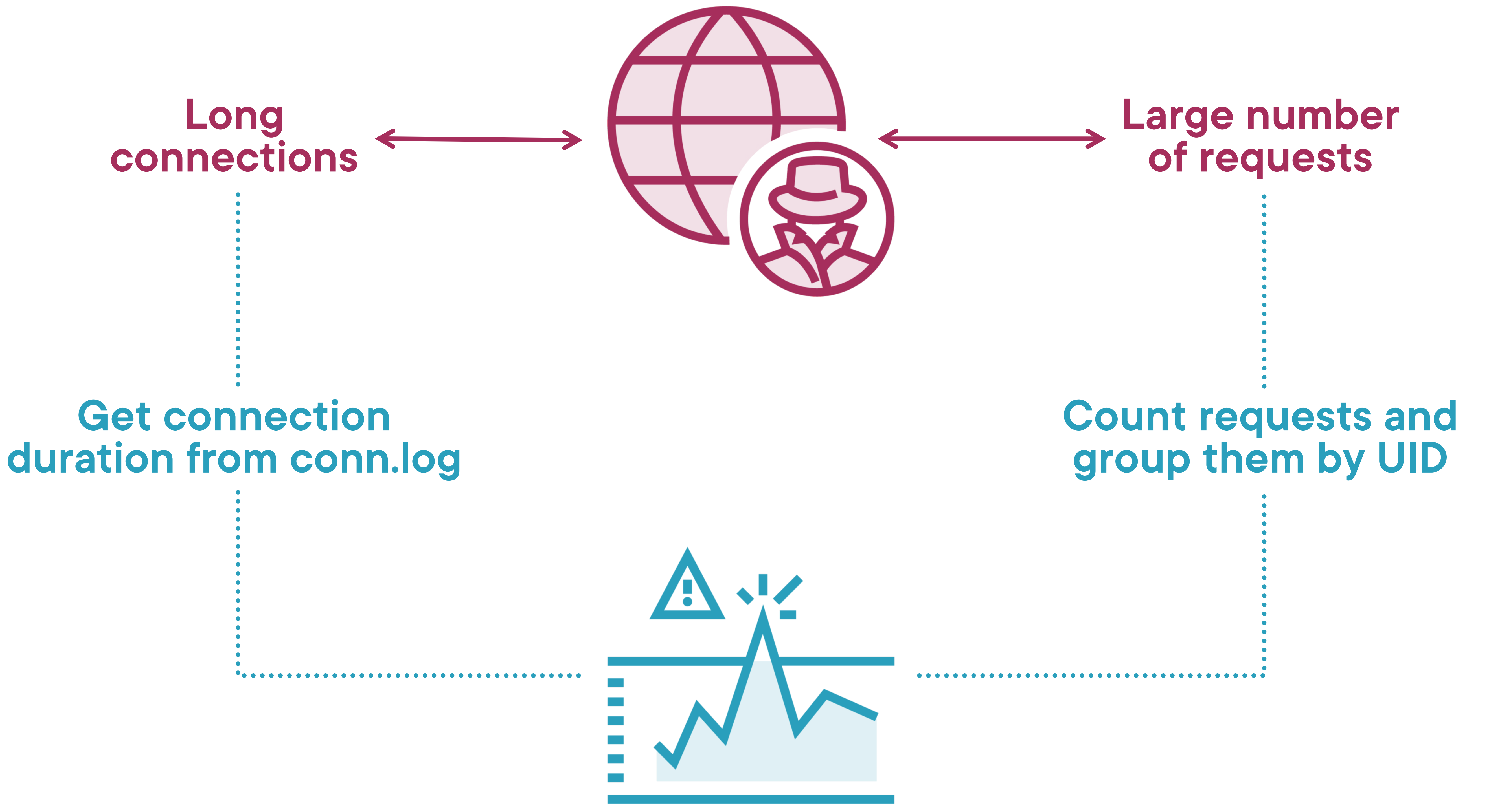
Beacon



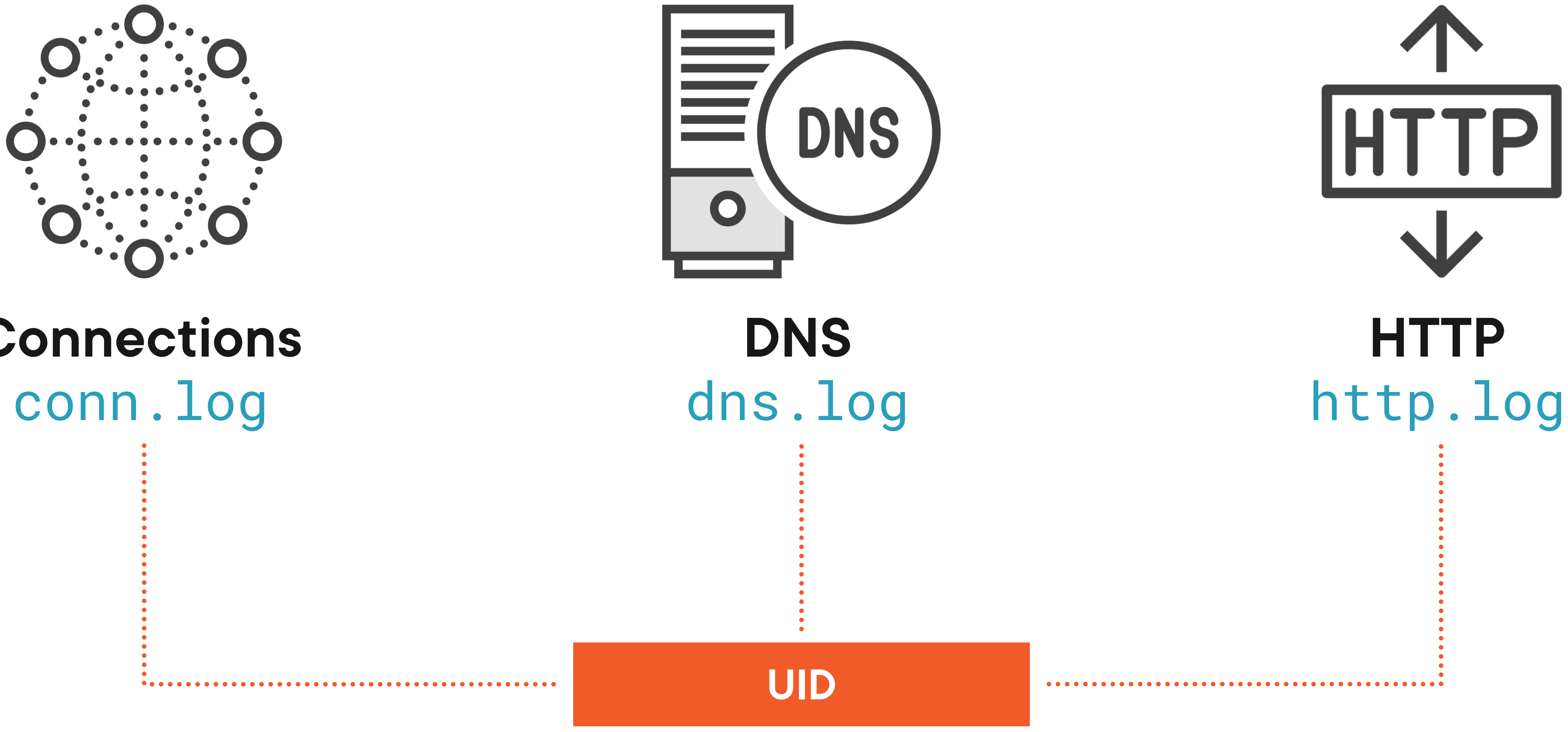
Jitter



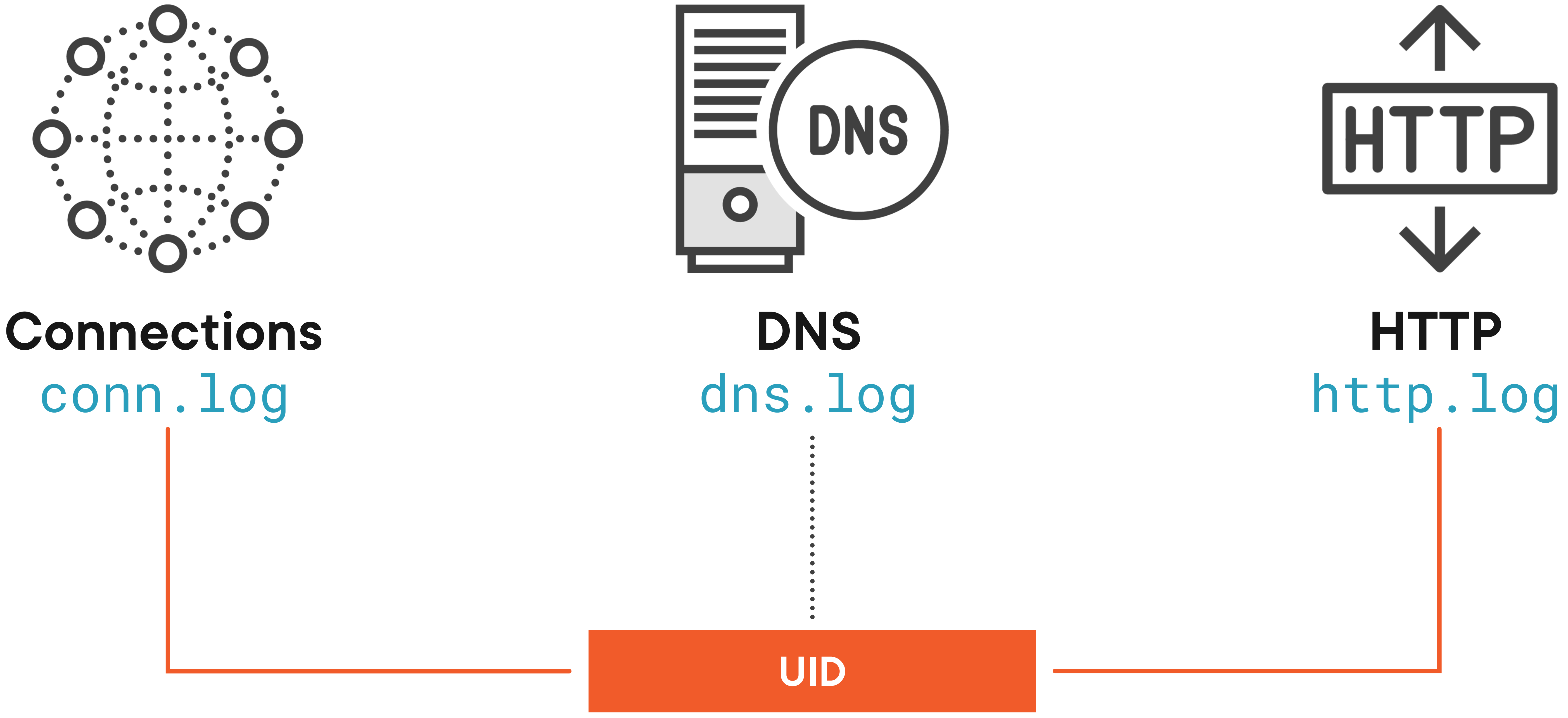
Detect Beaconing


Long connections
Large number of requests
Get connection duration from conn.log
Count requests and group them by UID

Correlating Zeek Log Files



Correlating Zeek Log Files



Python Counter Dictionaries

```
from collections import Counter

c = Counter()

c.update([‘aws.com’, ‘google.com’, ‘aws.com’])

print(c)
```

Python Counter Dictionaries

```
from collections import Counter

c = Counter()

c.update([‘aws.com’, ‘google.com’, ‘aws.com’])

print(c)
```

```
> Counter({'aws.com': 2, 'google.com': 1})
```

Python Counter Dictionaries

```
c1.update([‘aws.com’])
c2.update(‘aws.com’)

print(c1)
print(c2)
```

```
> Counter({'aws.com’: 1})
> Counter({‘a’:1, ’w’:1, ‘s’:1, ’.’:1, ‘c’:1. ‘o’:1, ‘m’:1})
```

Demo

Group HTTP logs by connection UID

Iterate through the conn log

- Filter http connections based on the service attribute
- Save results in an array

Correlate http and connection logs based on UID

- Identify beaconing based on connections with a long duration and many http requests

DEMO 3.2 Script

Frequency Analysis of DNS Traffic

Frequency represents the number of occurrences of a repeating event per unit of time

Aggregate and count occurrences based on their type

Specific to DNS traffic, group requests based on domain

- Can be applied at the host and network level

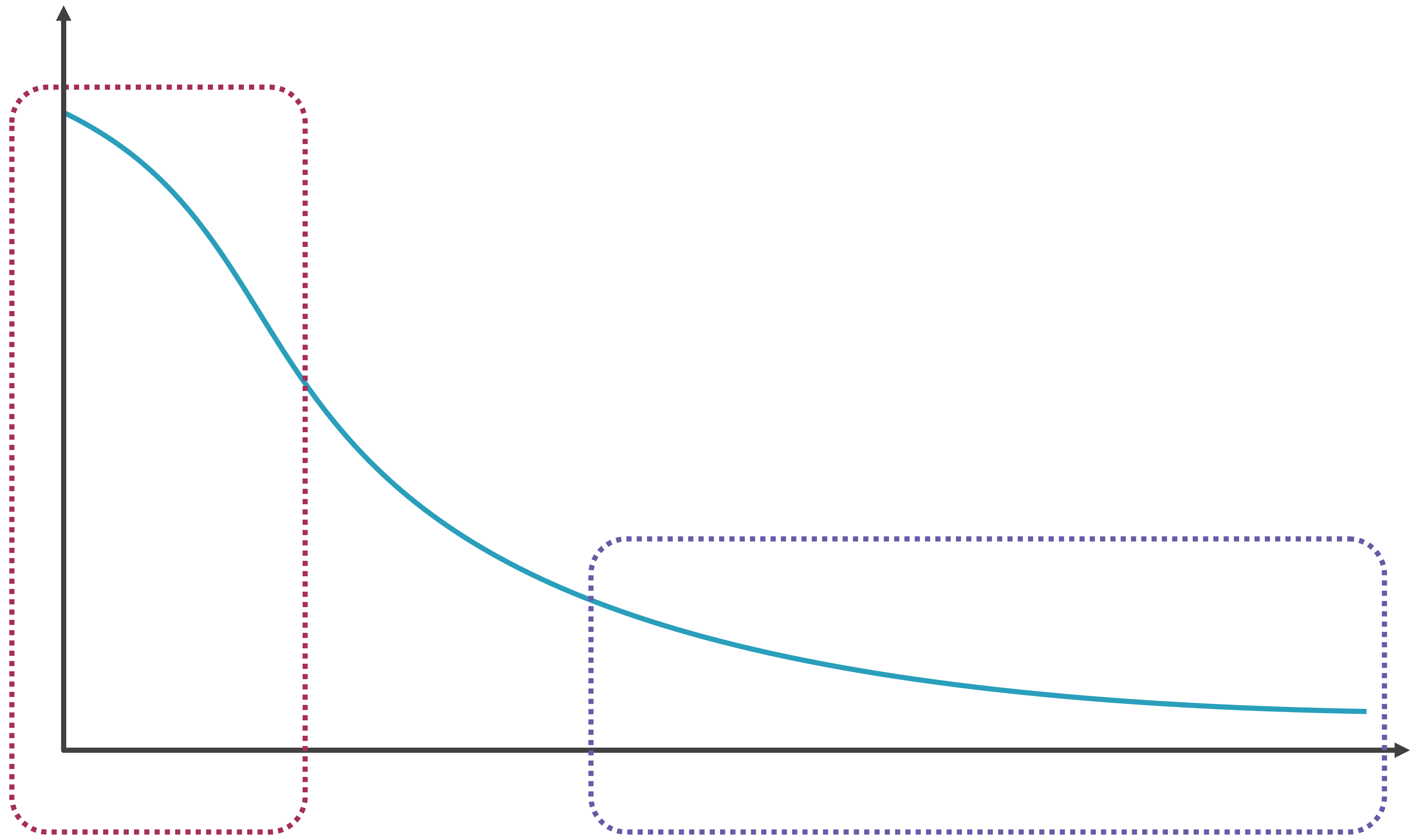
Calculating Occurrence

Log events

microsoft.com
microsoft.com
google.com
microsoft.com
microsoft.com
google.com
google.com
microsoft.com
google.com
microsoft.com
globomantics.com
microsoft.com

Domain	Occurrences
microsoft.com	6
google.com	4
globomantics.com	1

Long Tail Analysis of DNS Traffic



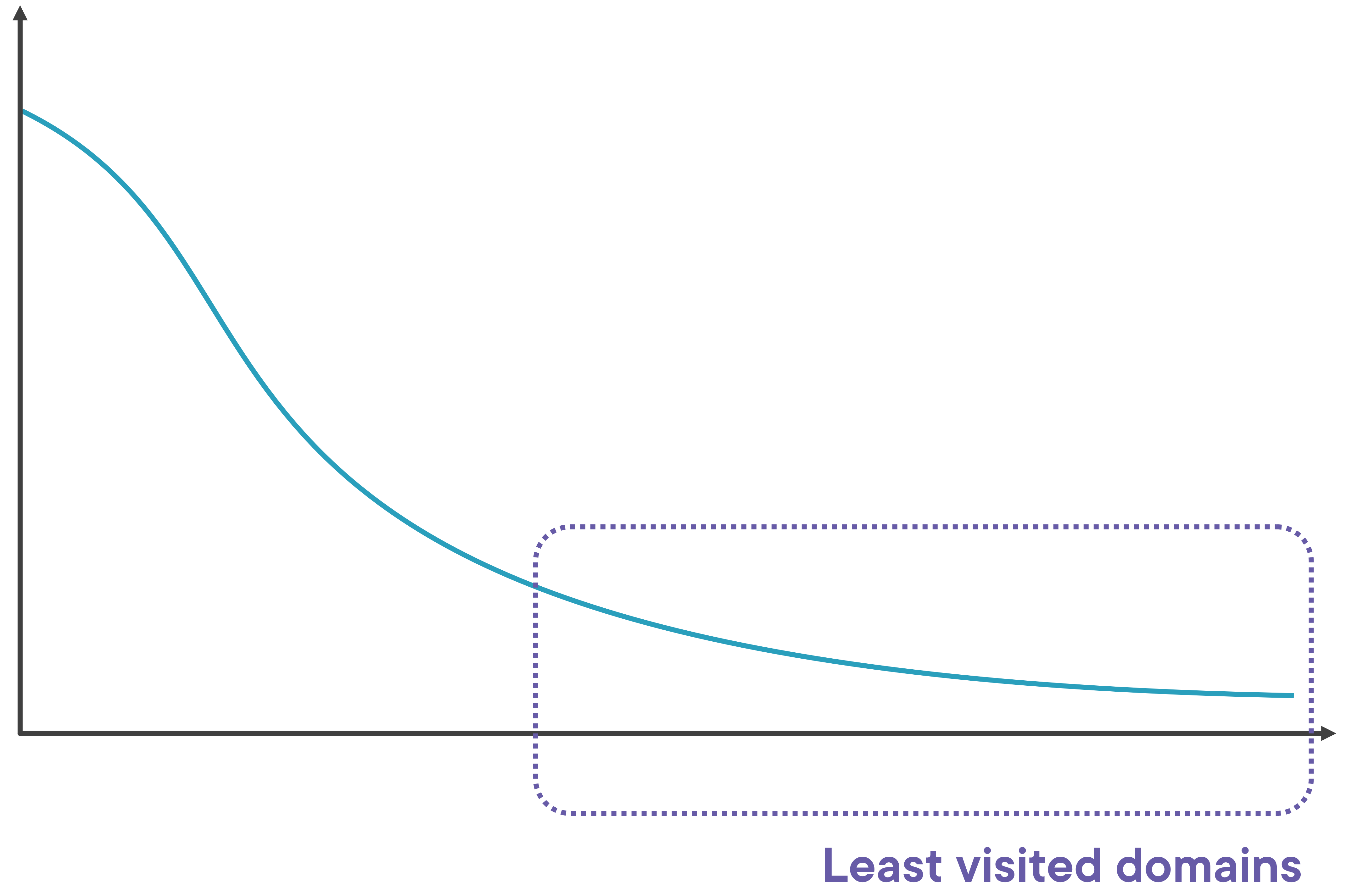
CDN Traffic

Cloud providers

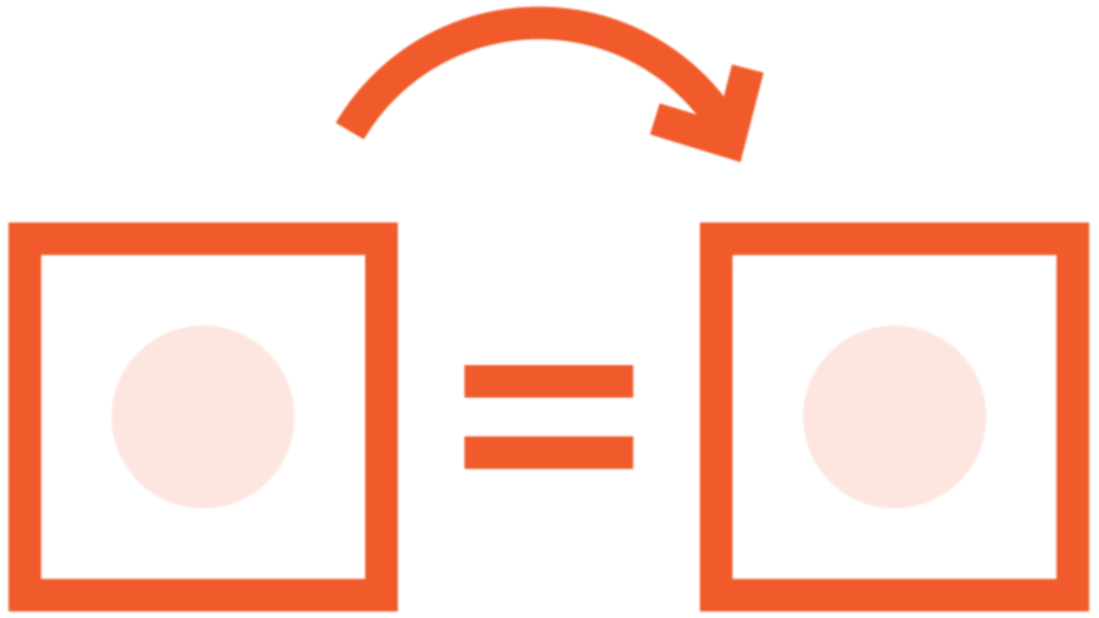
Ad networks

Least visited domains

Long Tail Analysis of DNS Traffic



Identifying Similar Domains



Typosquatting refers to mistyping a domain

- Characters can be replaced with similar-looking characters

Attackers can register these domains

- Trick people into visiting cloned websites
- Steal their credentials

String Similarity

```
from difflib import SequenceMatcher

similarity = SequenceMatcher(None, ‘yellow’, ’yell0w’).ratio()
```

```
> print(similarity)
0.83
```

Demo

Count occurrences of unique domains based on DNS traffic logs

- Extract root domain

Perform similarity analysis on least visited domains

DEMO 3.3 Script

Working with Timestamps

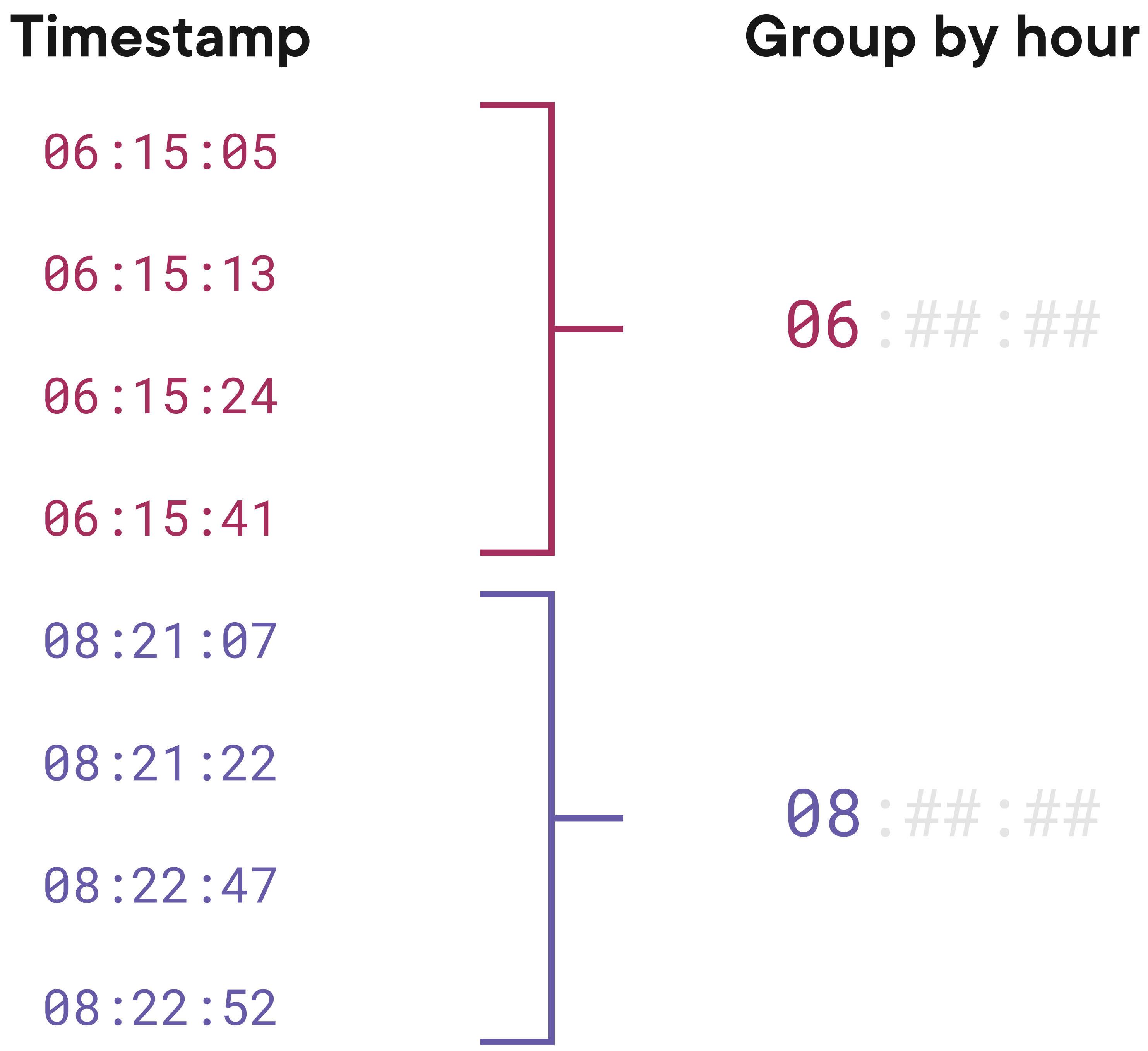
Identifying events that occurred at a specific point in time

Identifying how many events occurred during a time interval

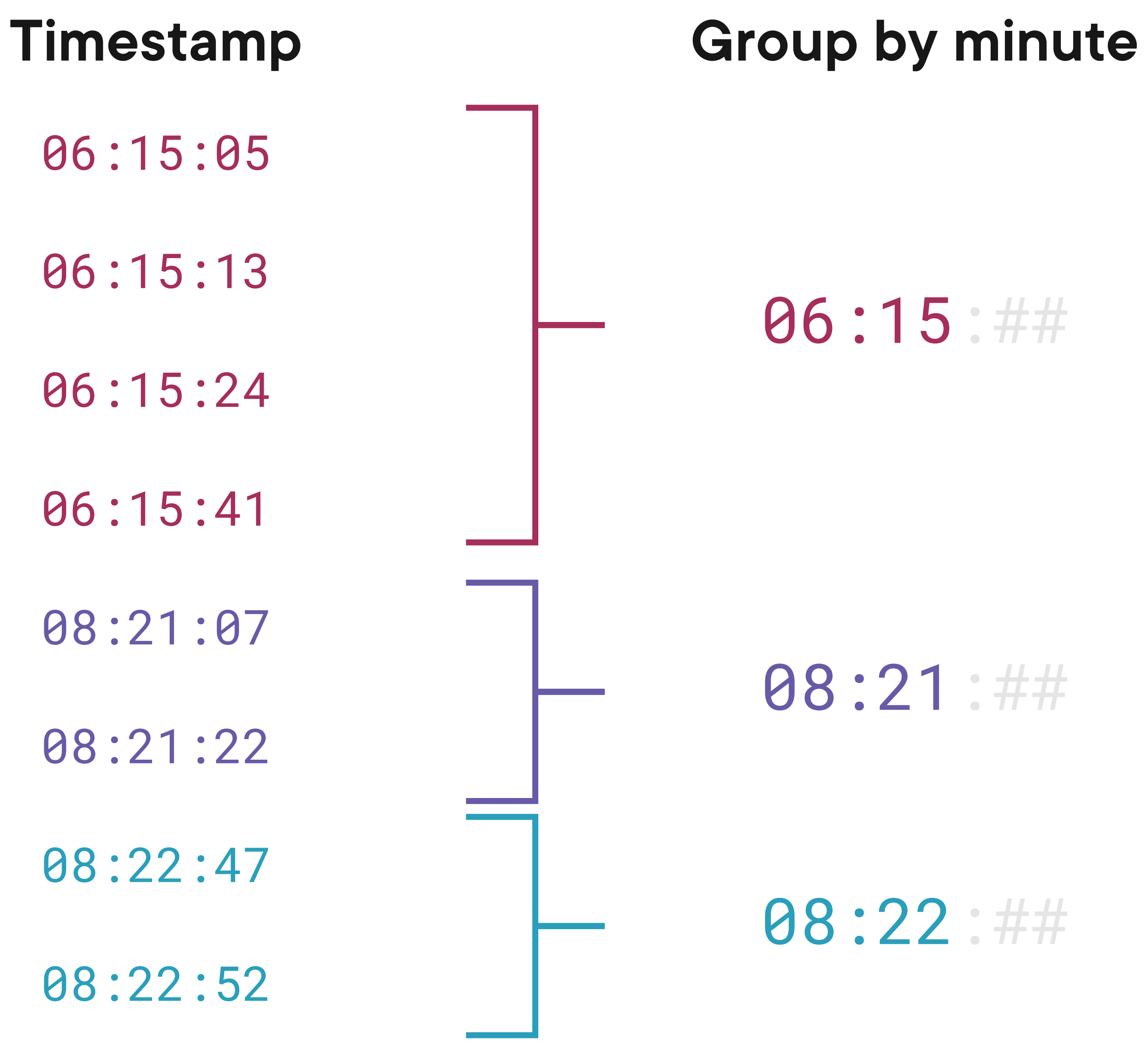
Working with Timestamps

Timestamp
06:15:05
06:15:13
06:15:24
06:15:41
08:21:07
08:21:22
08:22:47
08:22:52

Working with Timestamps



Working with Timestamps



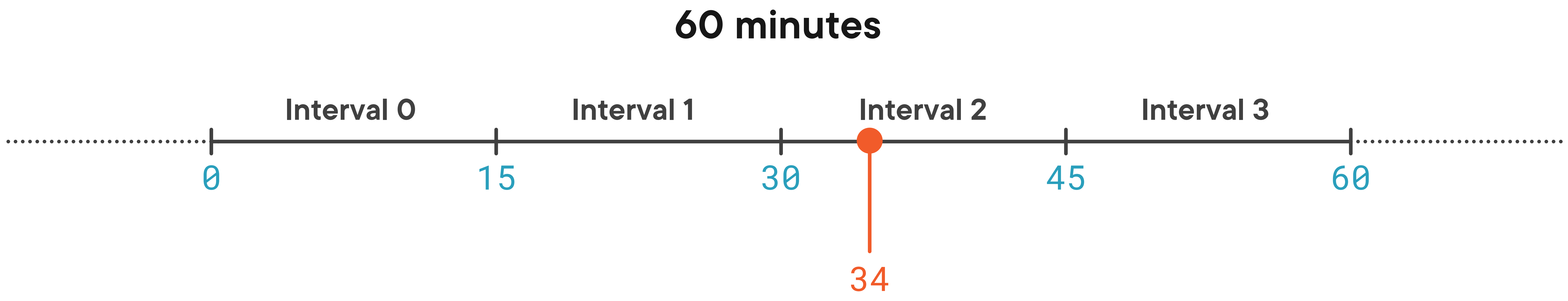
Working with Timestamps

60 minutes

Divide 60 minutes into equal intervals

`60 / 4 = 15`

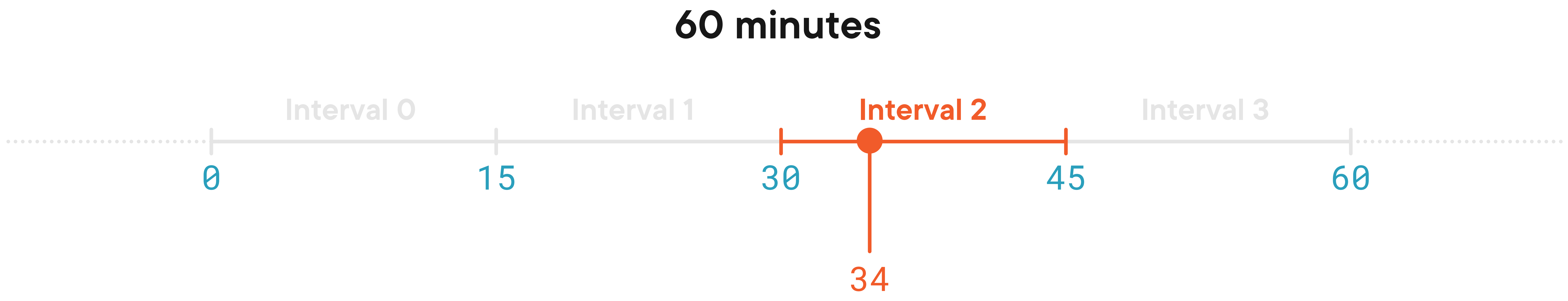
Working with Timestamps



Divide the minutes value by the interval length

`34 / 15 = 2.26`

Working with Timestamps



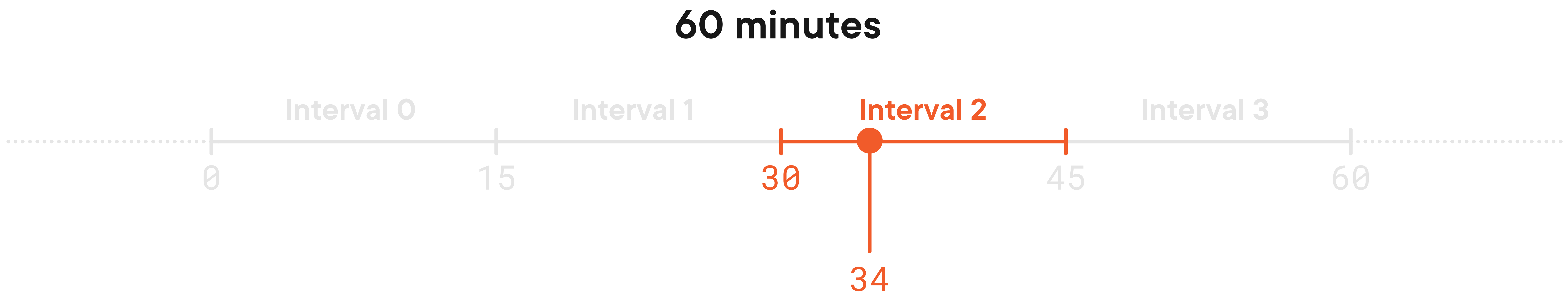
Divide the minutes value by the interval length

```
34 / 15 = 2.26
```

Get the interval number

```
int(34/15) = 2
```

Working with Timestamps



Divide the minutes value by the interval length	`34 / 15 = 2.26`
Get the interval number	`int(34/15) = 2`
Get the interval start	`int(34/15) = 2 * 15 = 30`

Visualizing Log Data

Important trends and insights can be assessed better visually

Building charts requires aggregated data

Leverage the Matplotlib library

```python
import matplotlib.pyplot as plt

def plotEvents(events):
❑   plt.bar(range(len(events)), list(events.values()), align='center’)
    plt.xticks(range(len(events)), list(events.keys()))
    plt.show()

if __name__ == "__main__":
    plotEvents({'Monday':3, 'Tuesday': 7, 'Wednesday': 2})
```

Leveraging Matplotlib

Create a bar chart based on the information contained in a dictionary

Demo

Install Matplotlib and dependencies

Create a bar chart

Aggregate and plot events

Identify suspicious file share activity related to a ransomware infection

DEMO 3.4 Script

Summary

Enriched log data with geoip information based on public IP address

Correlated connection and http request logs to detect beaconing activity

Performed long-tail and similarity analysis on DNS traffic to identify suspicious websites

Built visualizations with Matplotlib to detect anomalies attributed to ransomware infections

Next: interact with other services and technologies