BAGUETTE

HUNTING FOR EVIDENCE OF MALICIOUS BEHAVIORS IN DYNAMIC ANALYSIS REPORTS

VINCENT RAULIN, PIERRE-FRANÇOIS GIMENEZ, YUFEI HAN, VALÉRIE VIET TRIEM TONG

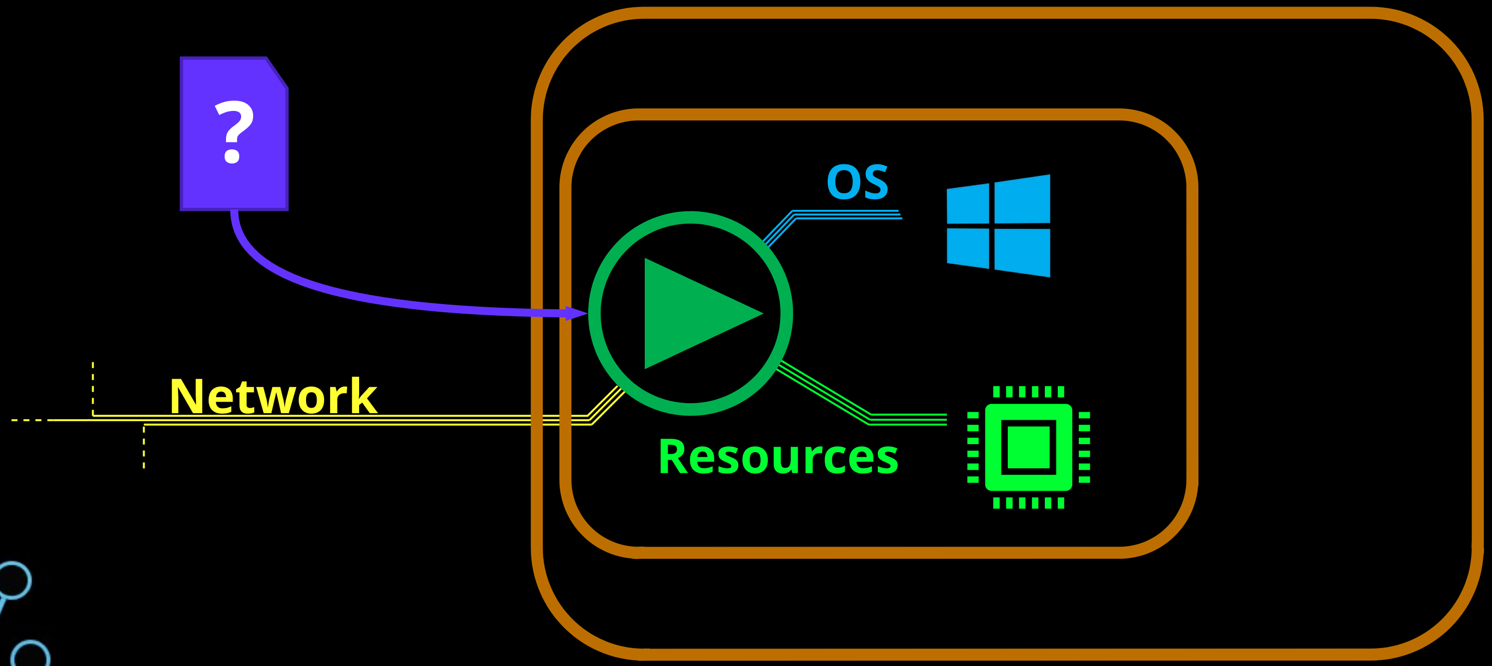
APRIL 5TH, THCON 2024

MALWARE ANALYSIS 101

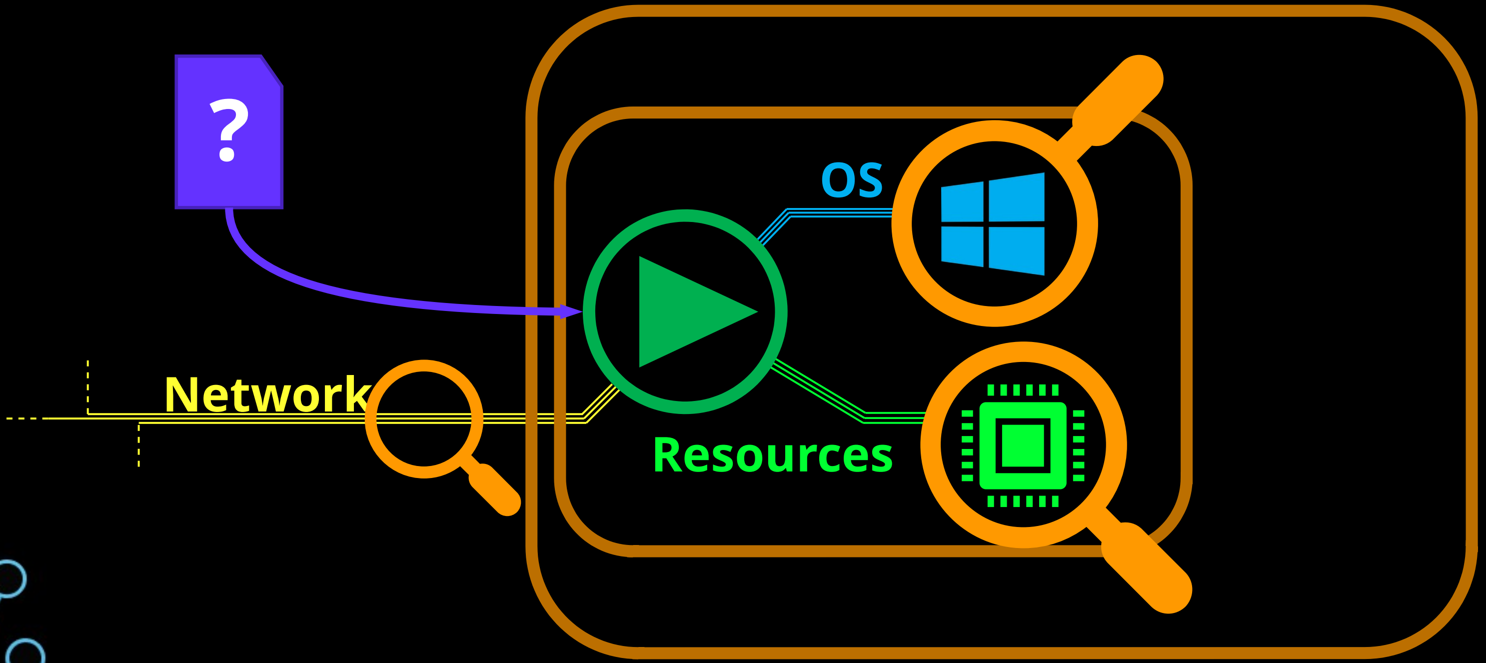
- >120 million new malware samples per year! (~4/sec) and an estimate of 265 billion USD annually by 2031!
- Exists in many flavors (MS PE, MSI, ELF, JAR archives, Android apps, scripts, PDF, MS Office macros, etc.)
- Two main approaches : static and dynamic analysis
- We focus on Windows malware dynamic analysis, using Cuckoo sandbox

CUCKOO ANALYSIS PIPELINE

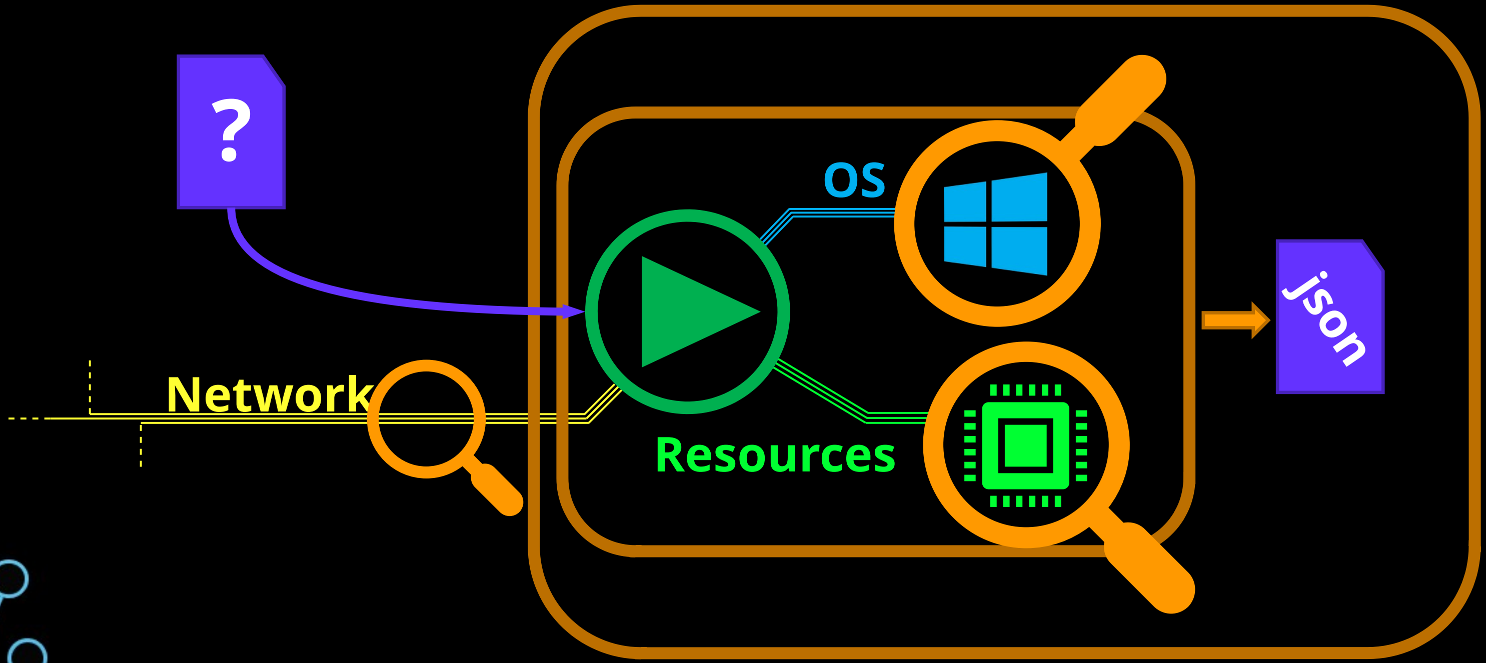
CUCKOO ANALYSIS PIPELINE



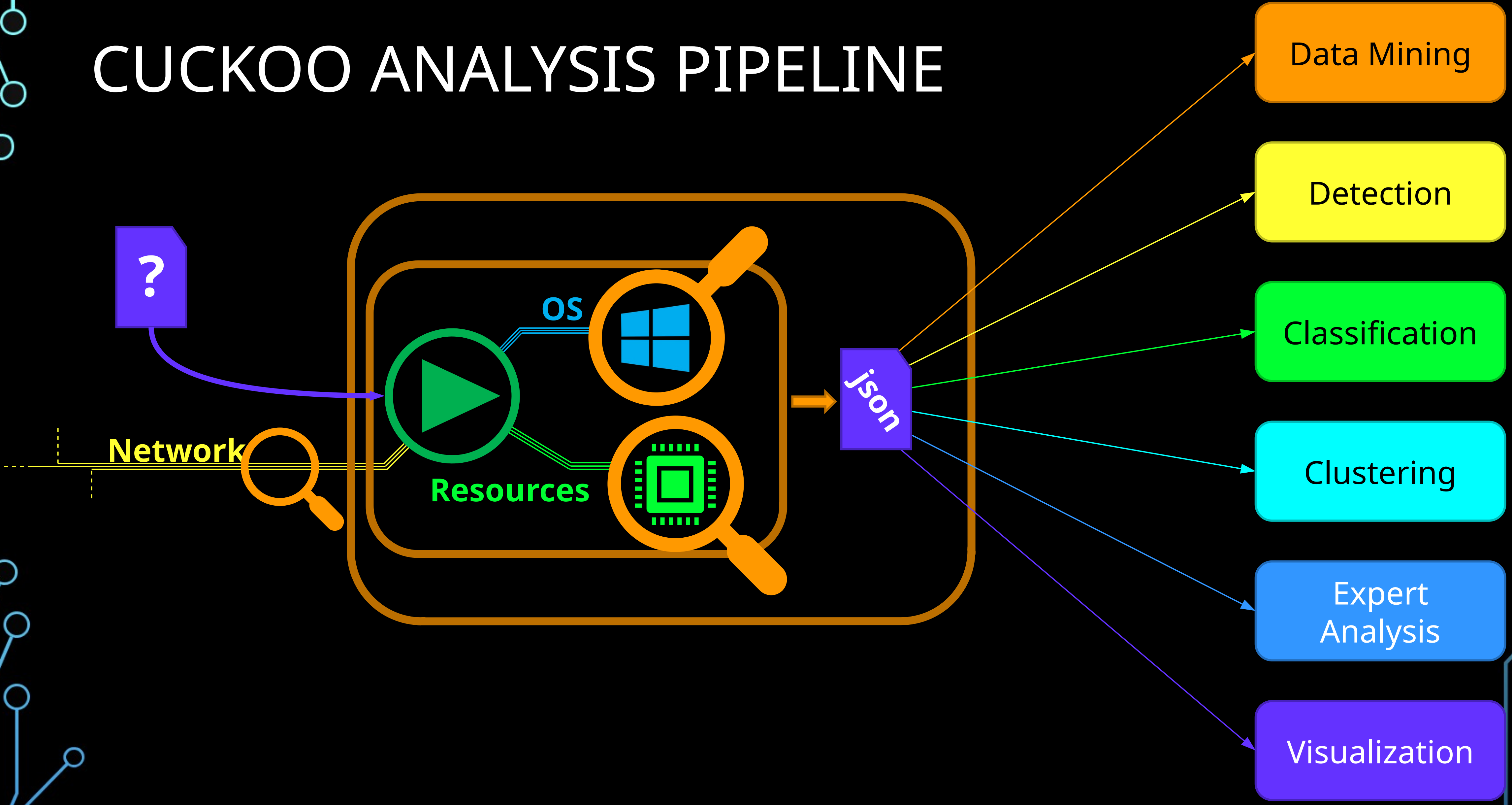
CUCKOO ANALYSIS PIPELINE



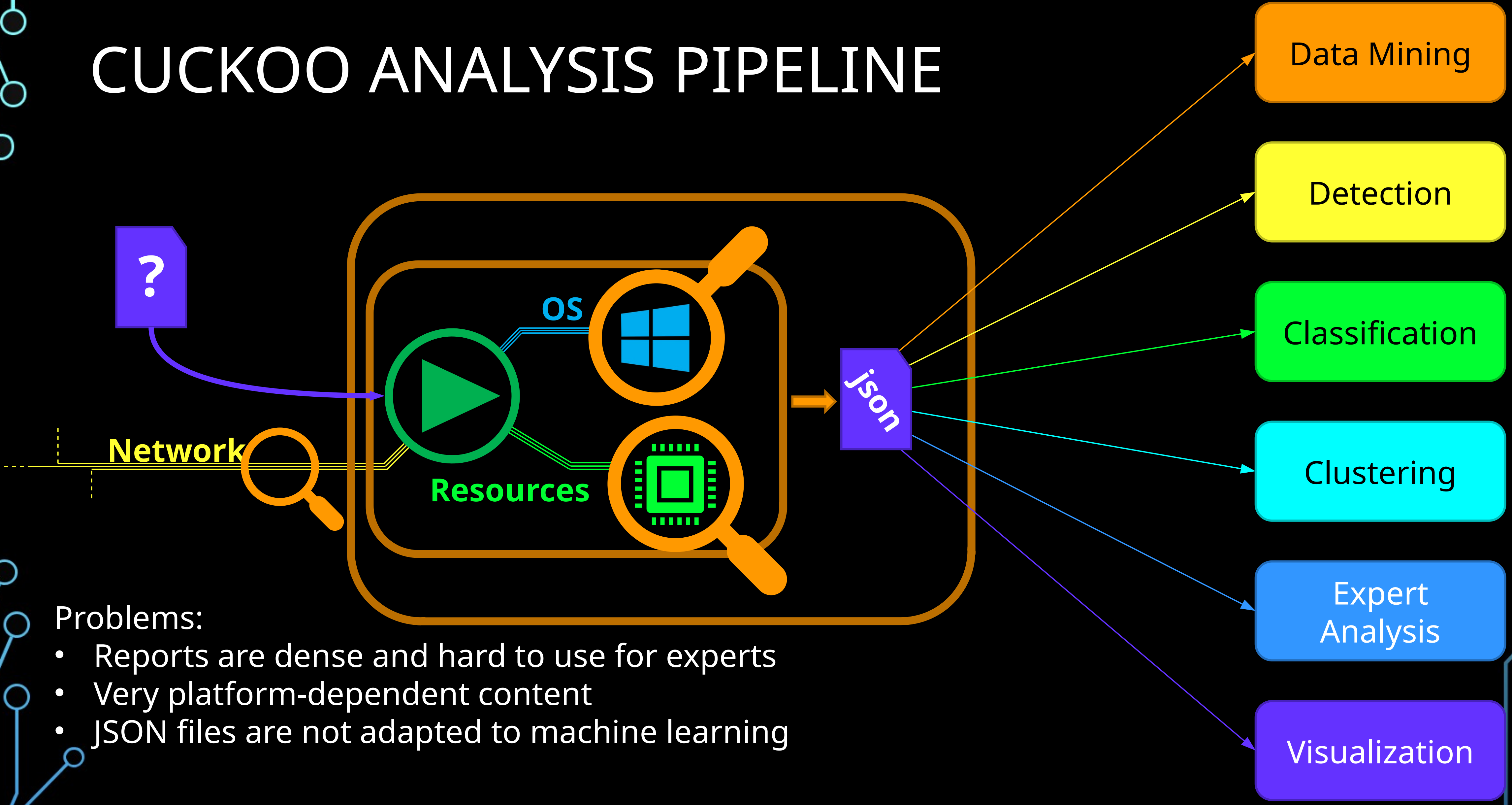
CUCKOO ANALYSIS PIPELINE



CUCKOO ANALYSIS PIPELINE



CUCKOO ANALYSIS PIPELINE



Problems:

- Reports are dense and hard to use for experts
- Very platform-dependent content
- JSON files are not adapted to machine learning

```
{
    "info": {…
    },
    "signatures": […
    ],
    "target": {…
    },
    "network": {…
    },
    "static": {
        "pdb_path": null,
        "pe_imports": [
            {
                "imports": [
                    {
                        "name": "DeleteCriticalSection",
                        "address": "0x40d0b4"
                    },
                    {
                        "name": "LeaveCriticalSection",
                        "address": "0x40d0b8"
                    },
                    {
                        "name": "EnterCriticalSection",
                        "address": "0x40d0bc"
                    },
                    {
                        "name": "InitializeCriticalSection",
                        "address": "0x40d0c0"
                    },
                    {
                        "name": "VirtualFree",
                        "address": "0x40d0c4"
                    },
                    {
                        "name": "VirtualAlloc",
                        "address": "0x40d0c8"
                    },
                    {
                        "name": "LocalFree",
                        "address": "0x40d0cc"
                    },
                    {
                        "name": "LocalAlloc",
                        "address": "0x40d0d0"
```

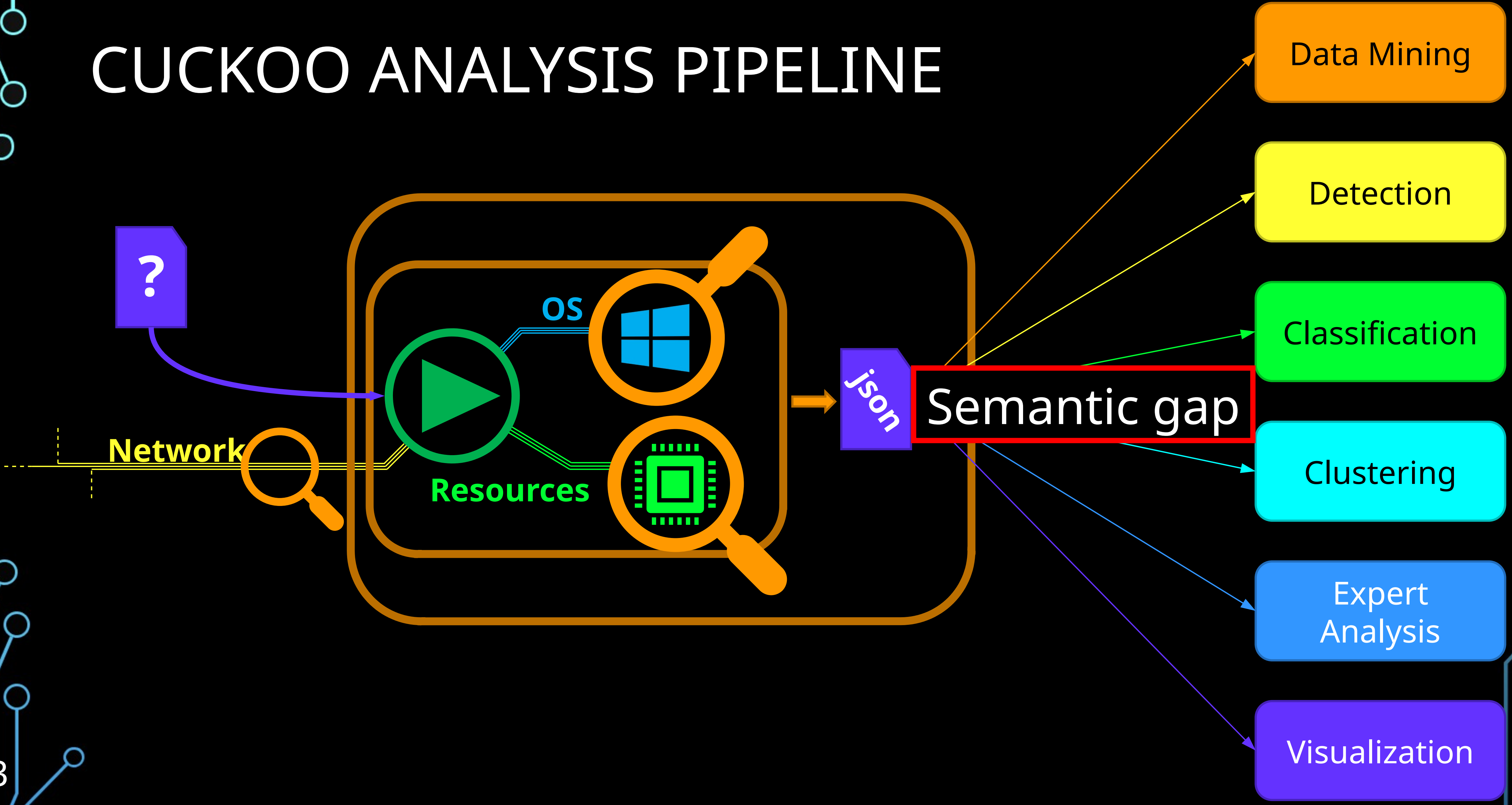
```
    },
    "network": {···
    },
    "static": {
        "pdb_path": null,
        "pe_imports": [···
        ],
        "peid_signatures": null,
        "keys": [],
        "signature": [],
        "pe_timestamp": "1992-06-20 00:22:17",
        "pe_exports": [],
        "imported_dll_count": 8,
        "pe_imphash": "884310b1928934402ea6fec1dbd3cf5e",
        "pe_resources": [···
        ],
        "pe_versioninfo": [···
        ],
        "pe_sections": [
            {
                "size_of_data": "0x00009400",
                "virtual_address": "0x00001000",
                "entropy": 6.557291120606633,
                "name": "CODE",
                "virtual_size": "0x0000933c"
            },
            {
                "size_of_data": "0x00000400",
                "virtual_address": "0x0000b000",
                "entropy": 2.7679914923058857,
                "name": "DATA",
                "virtual_size": "0x0000024c"
            },
            {
                "size_of_data": "0x00000000",
                "virtual_address": "0x0000c000",
                "entropy": 0.0,
                "name": "BSS",
                "virtual_size": "0x00000e4c"
            },
            {
                "size_of_data": "0x00000a00",
                "virtual_address": "0x0000d000",
                "entropy": 4.430733069799032,
```

```
{
    "info": { ···
    },
    "signatures": [ ···
    ],
    "target": { ···
    },
    "network": { ···
    },
    "static": { ···
    },
    "dropped": [ ···
    ],
    "behavior": {
        "generic": [
            {
                "process_path": "C:\\Users\\Marc Elbichon\\AppData\\Local\\Temp\\is-6BR7Q.tmp\\c557b1737ec0d359ebc4868caf8a31c2a31ec56954ba51bb39d90312d023dc97.tmp",
                "process_name": "c557b1737ec0d359ebc4868caf8a31c2a31ec56954ba51bb39d90312d023dc97.tmp",
                "pid": 2968,
                "summary": {
                    "file_created": [
                        "C:\\Users\\Marc Elbichon\\AppData\\Local\\Temp\\is-S7NDS.tmp\\_isetup\\_shfoldr.dll",
                        "C:\\Users\\Marc Elbichon\\AppData\\Local\\Temp\\is-S7NDS.tmp\\_isetup\\_setup64.tmp",
                        "C:\\Users\\Marc Elbichon\\AppData\\Local\\Temp\\is-S7NDS.tmp\\_isetup\\_RegDLL.tmp",
                        "C:\\Program Files (x86)\\FevsoftFR\\FinalRecovery\\data\\is-5CVH4.tmp",
                        "C:\\Program Files (x86)\\FevsoftFR\\FinalRecovery\\is-S2PNI.tmp",
                        "C:\\Program Files (x86)\\FevsoftFR\\FinalRecovery\\is-9LRAI.tmp",
                        "C:\\Users\\Marc Elbichon\\AppData\\Local\\Temp\\is-S7NDS.tmp\\_isetup\\_iscrypt.dll",
                        "C:\\Program Files (x86)\\FevsoftFR\\FinalRecovery\\unins000.dat",
                        "C:\\Program Files (x86)\\FevsoftFR\\FinalRecovery\\is-V1A4O.tmp",
                        "C:\\Program Files (x86)\\FevsoftFR\\FinalRecovery\\is-MQUFU.tmp",
                        "C:\\Program Files (x86)\\FevsoftFR\\FinalRecovery\\is-5F2AK.tmp"
                    ],
                    "file_recreated": [
                        "C:\\Program Files (x86)\\FevsoftFR\\FinalRecovery\\unins000.dat",
                        "\\Device\\KsecDD",
                        "\\Device\\DeviceApi\\CMApi"
                    ],
                    "directory_created": [
                        "C:\\Users\\Marc Elbichon\\AppData\\Local\\Temp\\is-S7NDS.tmp",
                        "C:\\Program Files (x86)\\FevsoftFR",
                        "C:\\Program Files (x86)\\FevsoftFR\\FinalRecovery\\data",
                        "C:\\Users\\Marc Elbichon\\AppData\\Local\\Temp\\is-S7NDS.tmp\\_isetup",
                        "C:\\Program Files (x86)\\FevsoftFR\\FinalRecovery"
                    ],
```

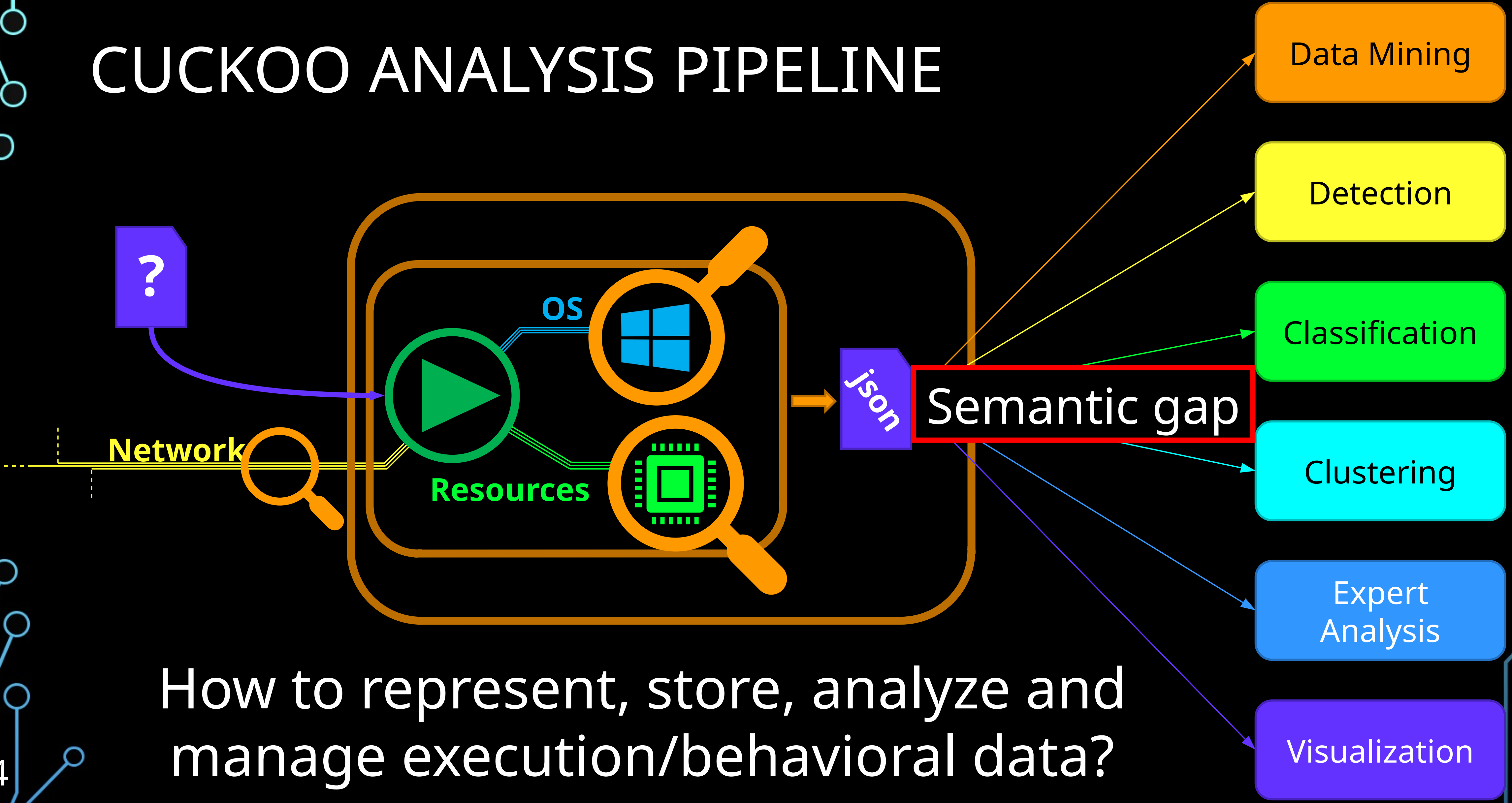
```json
{
    "process_path": "C:\\Users\\Marc Elbichon\\AppData\\Local\\Temp\\c557b1737ec0d359ebc4868caf8a31c2a31ec56954ba51bb39d90312d023dc97.exe",
    "calls": [
        {
            "category": "system",
            "status": 1,
            "stacktrace": [],
            "api": "LdrGetDllHandle",
            "return_value": 0,
            "arguments": {
                "module_name": "kernel32.dll",
                "stack_pivoted": 0,
                "module_address": "0x75d20000"
            },
            "time": 1674405919.603929,
            "tid": 6644,
            "flags": {}
        },
        {
            "category": "process",
            "status": 1,
            "stacktrace": [],
            "api": "NtAllocateVirtualMemory",
            "return_value": 0,
            "arguments": {
                "process_identifier": 5356,
                "region_size": 1048576,
                "stack_dep_bypass": 0,
                "stack_pivoted": 0,
                "heap_dep_bypass": 0,
                "protection": 1,
                "process_handle": "0xffffffff",
                "allocation_type": 8192,
                "base_address": "0x00e70000"
            },
            "time": 1674405919.603929,
            "tid": 6644,
            "flags": {
                "protection": "PAGE_NOACCESS",
                "allocation_type": "MEM_RESERVE"
            }
        },
        {
            "category": "process",
            "status": 1,
```


It is difficult to map these data to the actual effect the software has on the system

CUCKOO ANALYSIS PIPELINE

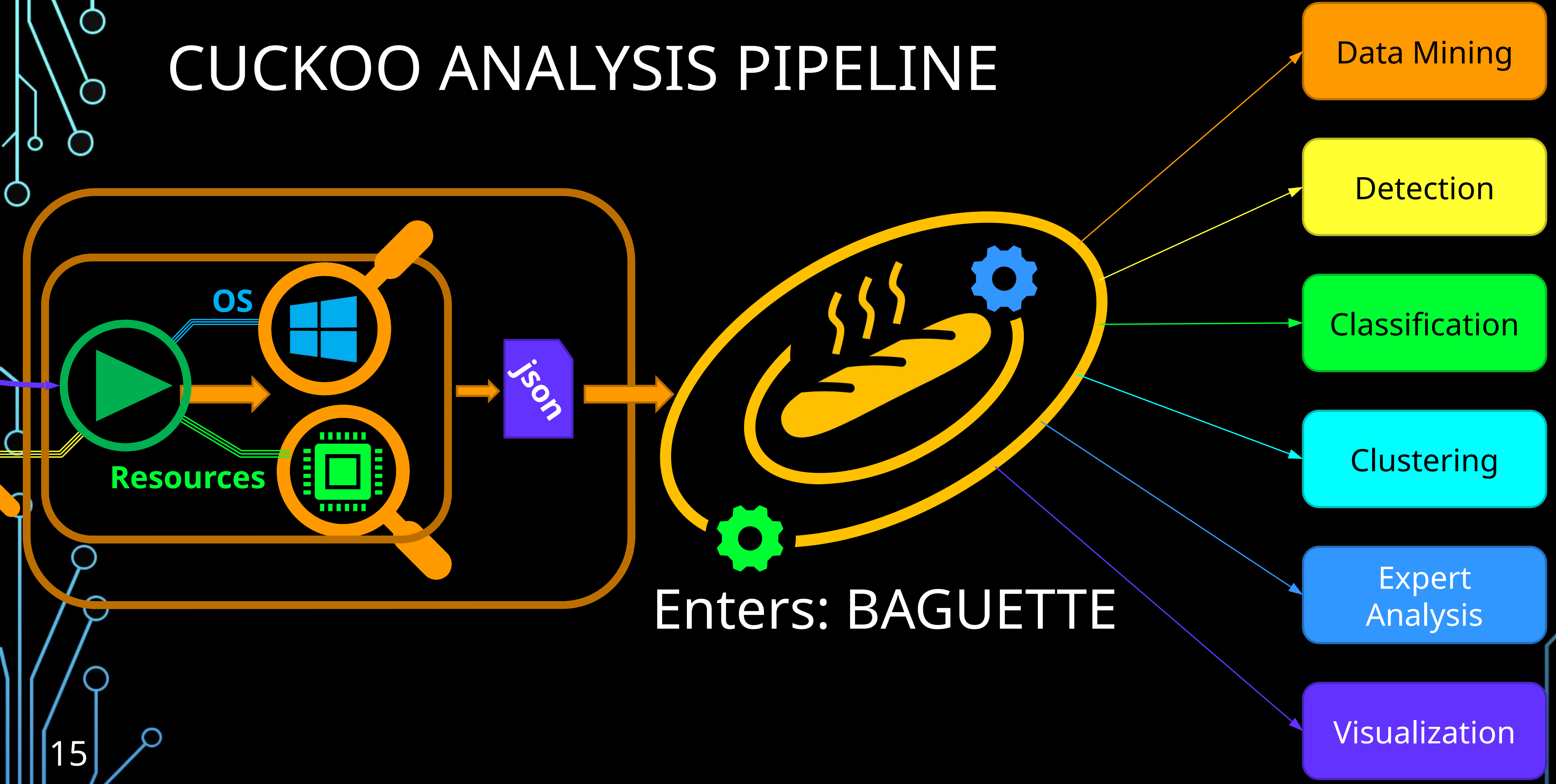


CUCKOO ANALYSIS PIPELINE



How to represent, store, analyze and manage execution/behavioral data?

CUCKOO ANALYSIS PIPELINE



Enters: BAGUETTE

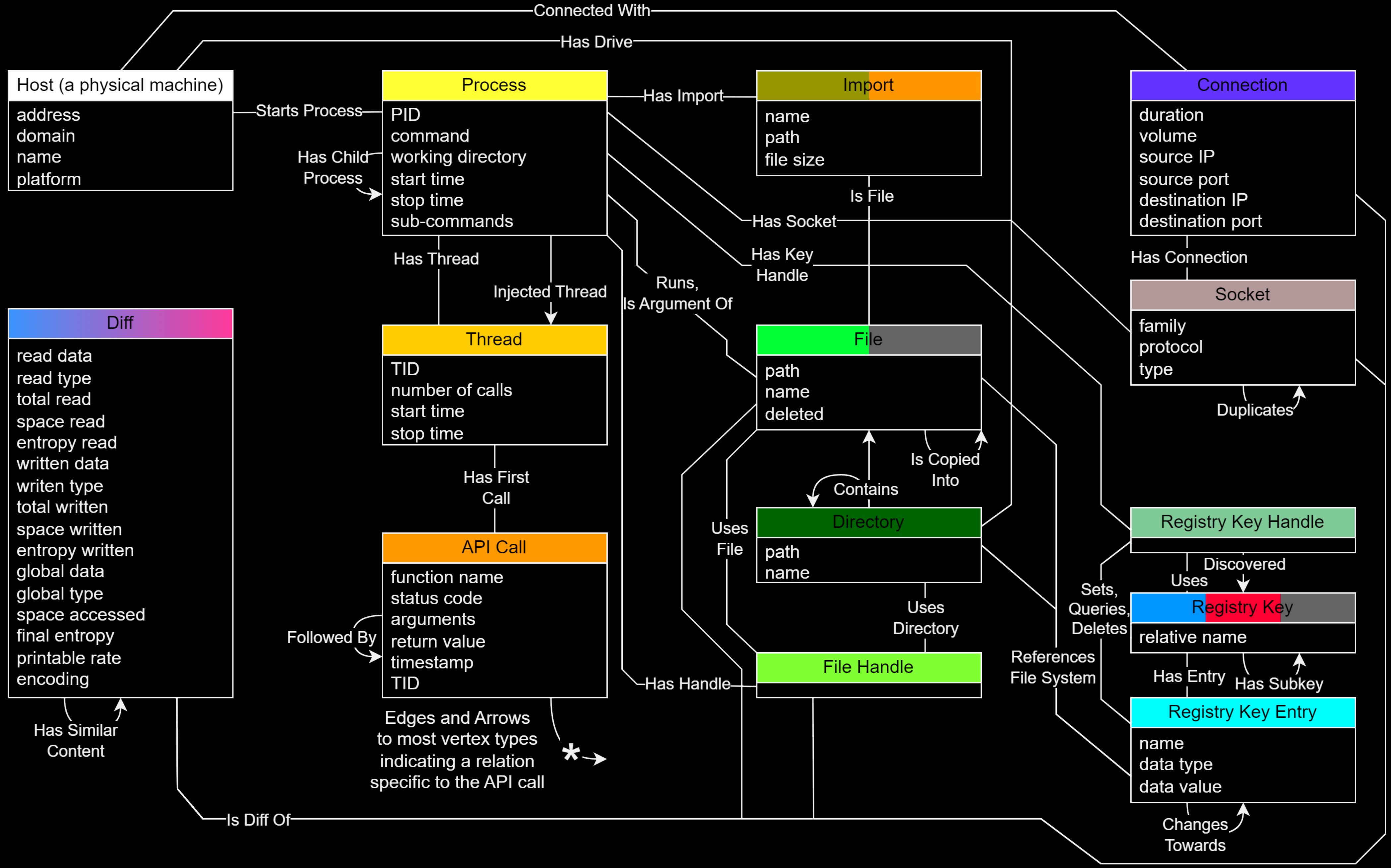
CUCKOO ANALYSIS PIPELINE

A framework to represent, and and analyze runtime behaviors

WHAT IS BAGUETTE?

- A graphical representation of dynamic analysis traces
- Heterogeneous graph
- Shows resources given by the OS: file system, registry keys, network connection, etc.
- Links related resources

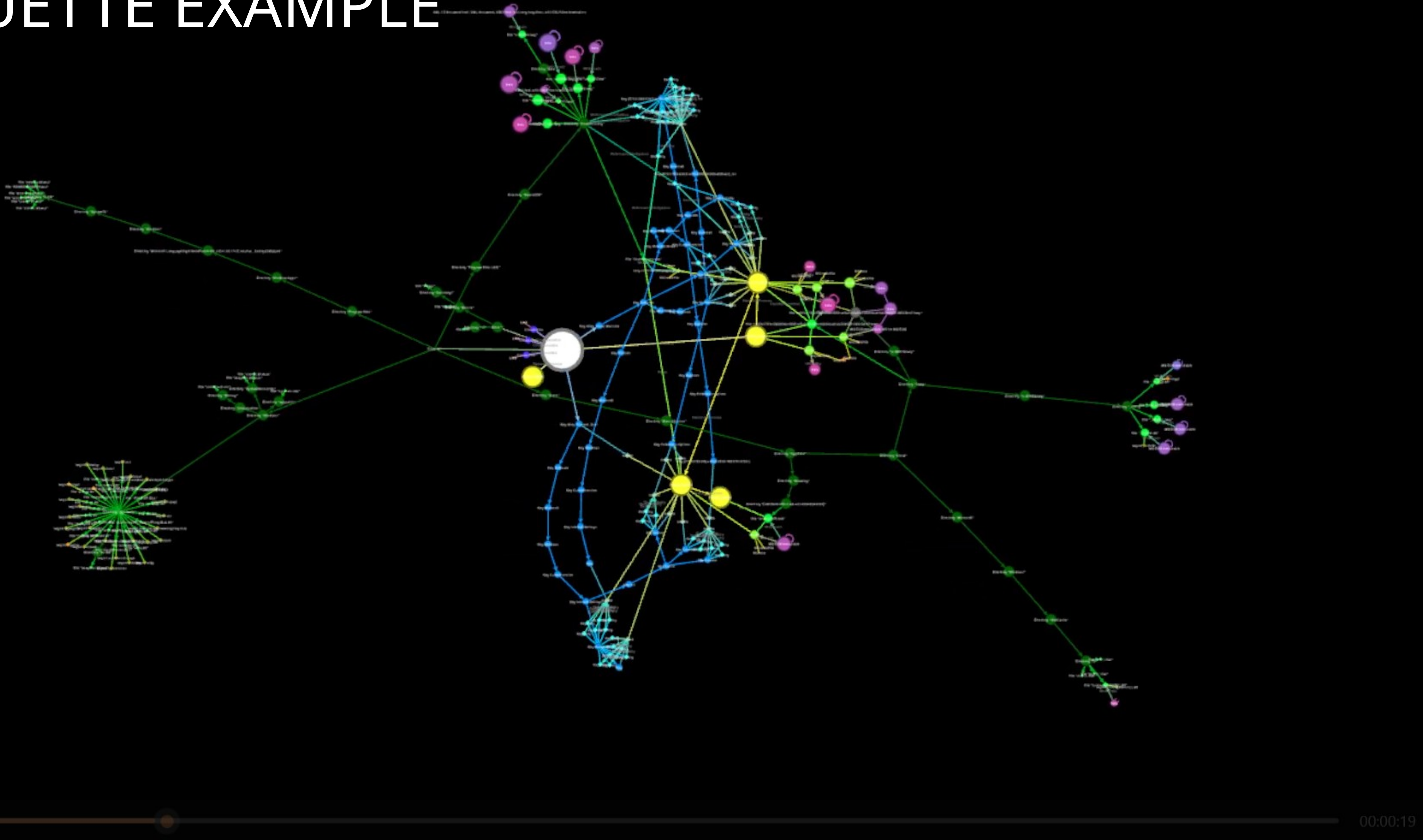
NODE TYPES AND RELATIONS



DIFF NODES

- Any high-level read or write operations involve many elementary read and write system calls
- We merge them into « diff » nodes summarizing data transfer
- Diff nodes include all the data read from and written into a socket or a file
- That way, we can easily analyzed read and written data
  - Entropy computations
  - ASCII or binary data
  - Header analysis

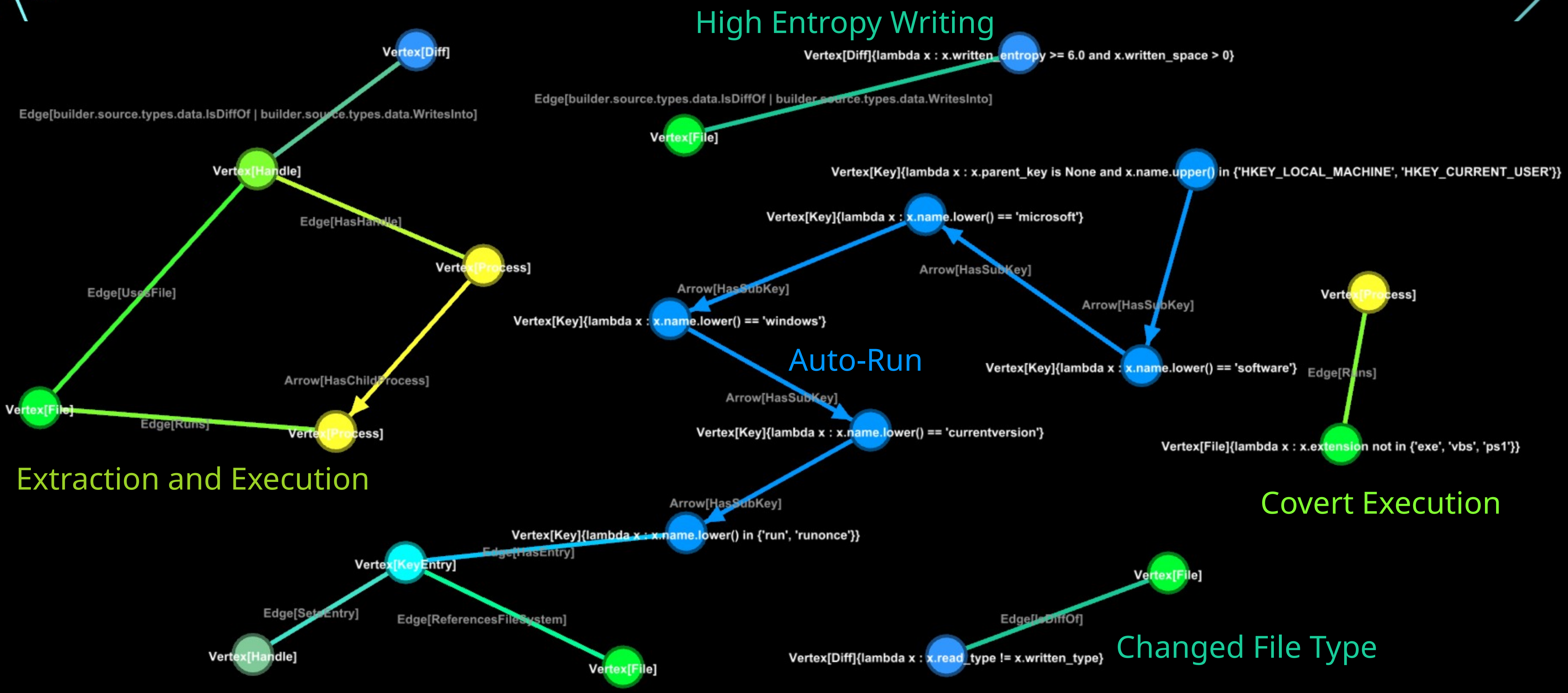
BAGUETTE EXAMPLE



METAGRAPHS TO ANALYZE A BAGUETTE

- We use graph patterns called « metagraphs »
- They are graphs where:
    - nodes can match one or several BAGUETTE nodes
    - edges can match one or several BAGUETTE edges
    - nodes can also have conditions of BAGUETTE attributes
- Since BAGUETTE are high-level, we can manually write metagraphs that match specific behaviors

METAGRAPH EXAMPLES



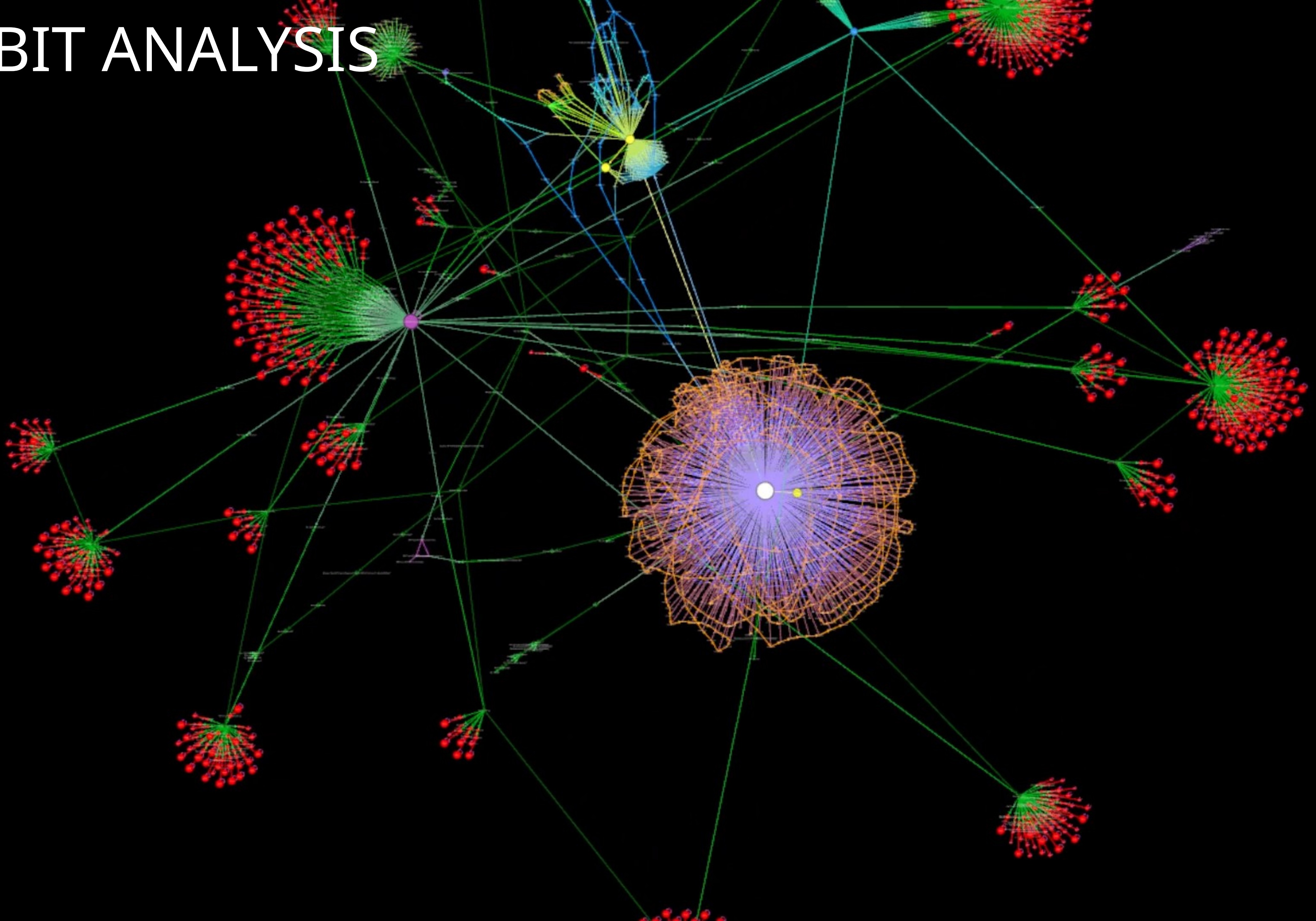
EXPERIMENTS

- We analyze three malware families:
  - GCleaner, a file dropper
  - SnakeKeyLogger, a key logger and spyware
  - LockBit, a ransomware

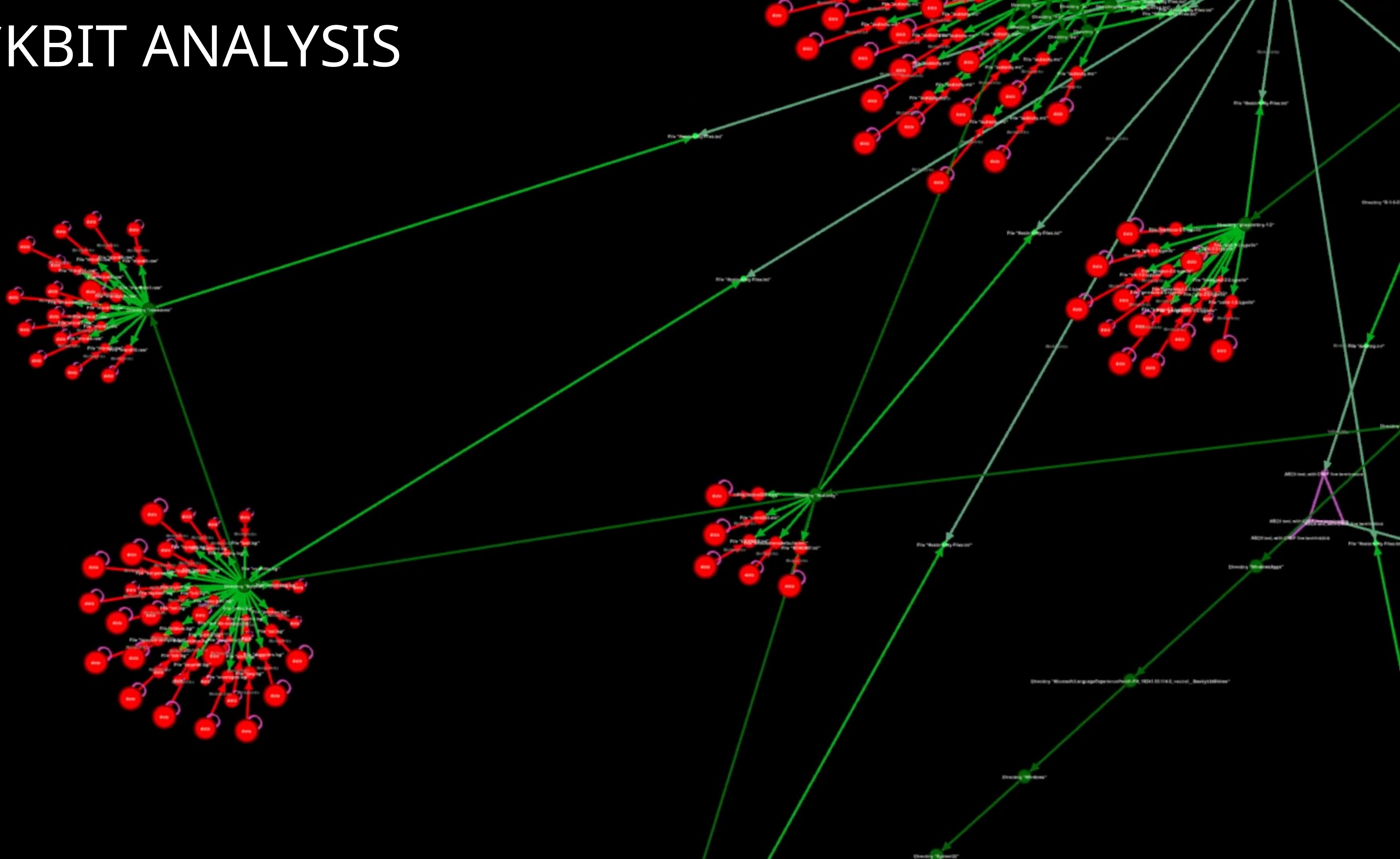
Metagraph	GCleaner (247)			SnakeKeyLogger (436)			LockBit (7)		
	p	n	σ	p	n	σ	p	n	σ
High-Entropy Writing	97.57%	1.53	0.59	13.76%	1.08	0.28	28.57%	2450.0	1878.0
Changed File Type	97.57%	1.0	0.0	4.82%	1.05	0.21	14.29%	1.0	0.0
Covert Execution	98.38%	1.0	0.0	0%	-	-	0%	-	-
Extraction and Execution	98.38%	2.97	0.17	13.53%	1.0	0.0	0%	-	-
Auto-Run	0%	-	-	0%	-	-	28.57%	1.0	0.0
p : Proportion of matches, n : average number per matching sample, σ : standard deviation per matching sample									

- Quite different proportions depending on families
- Tells us how to select samples (for example, which sample executed their payloads)

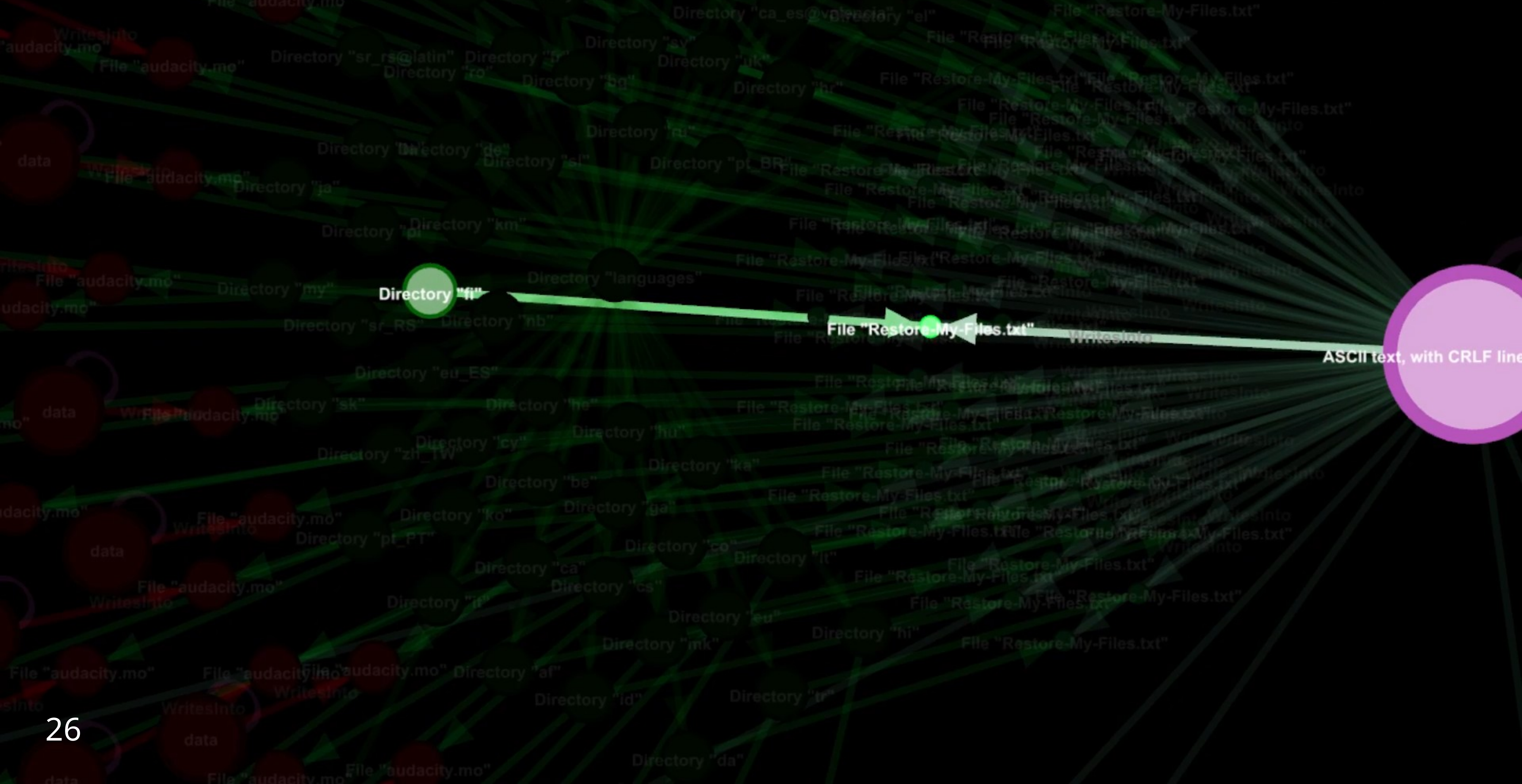
LOCKBIT ANALYSIS



LOCKBIT ANALYSIS



LOCKBIT ANALYSIS



WHAT NEXT?

This was the state of our research at the time we submitted to THCon… But since, we had some fun

Next research question: how to automatically create metagraphs from a dataset of malware ?

Still a work in progress

NEW GOAL

Given a unlabelled dataset of malware samples, how can we:

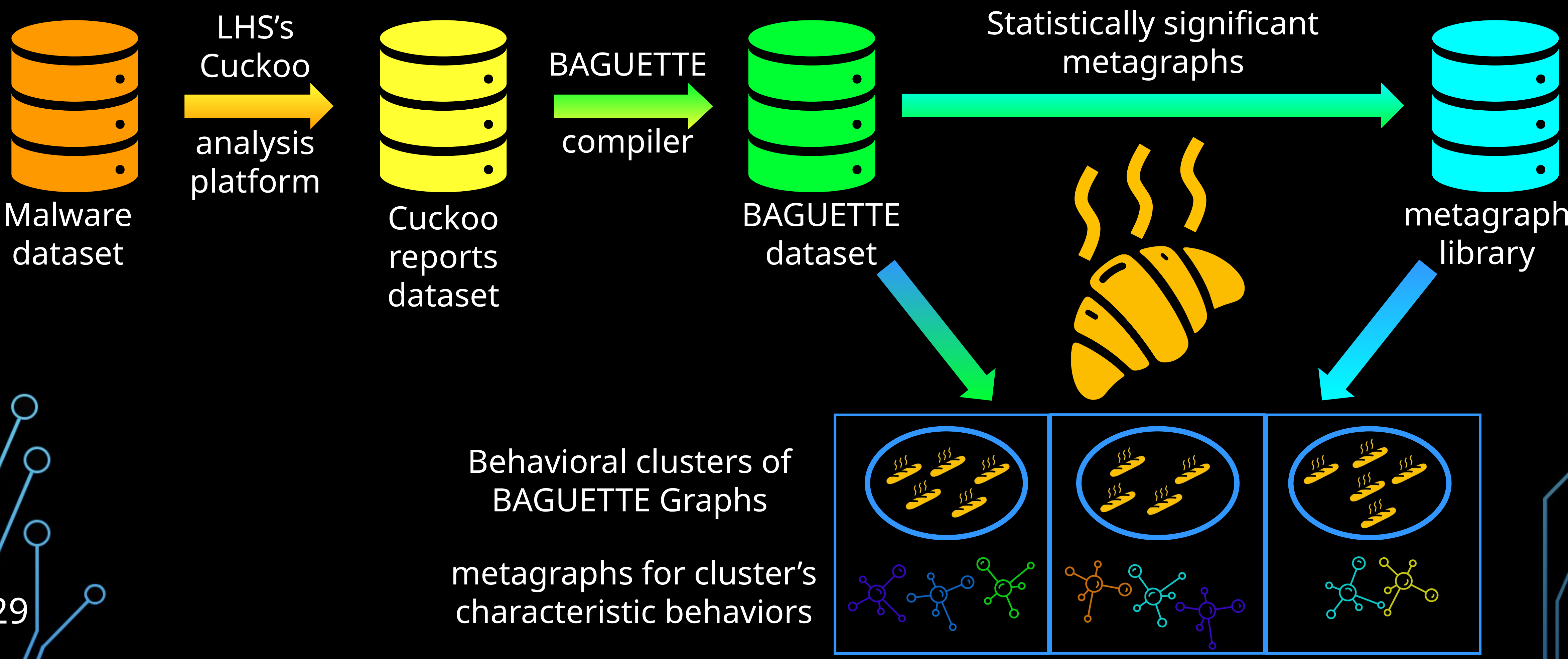
- Recover clusters of behaviors that hopefully match the families/classes
- Recover the behavioral patterns characteristic of each of these clusters

With some constrains:

- In an unsupervised way (no labels)
- Without expert knowledge on malware analysis, just on system programming

CROISSANT!

Clustering Research Ontological Interface with Systemic Signature Assimilation of Novel Tactics



EXPERIMENTAL DETAILS

- Dataset: 13 families, 100 samples for each family
- MetaGraph library generation is an iterative process:
  - Generate new valid metagraphs from the previous library
  - Search them across the BAGUETTE dataset
  - Select the best ones according to some metrics
  - Repeat
- A classic genetic algorithm, with two hurdles:
  - How to mutate metagraphs?
  - How to select the best ones?

EXPERIMENTAL DETAILS

- What is a good metagraph?
  - A metagraph that is rarely present? → probably not significative
  - A metagraph that is always present? → probably typical Windows behavior, like DLL imports, etc., not interesting for malware analysis
  - A metagraph that are very common in some software but very rare in others? → sounds like something akin to a signature!

- But what does it mean, mathematically?

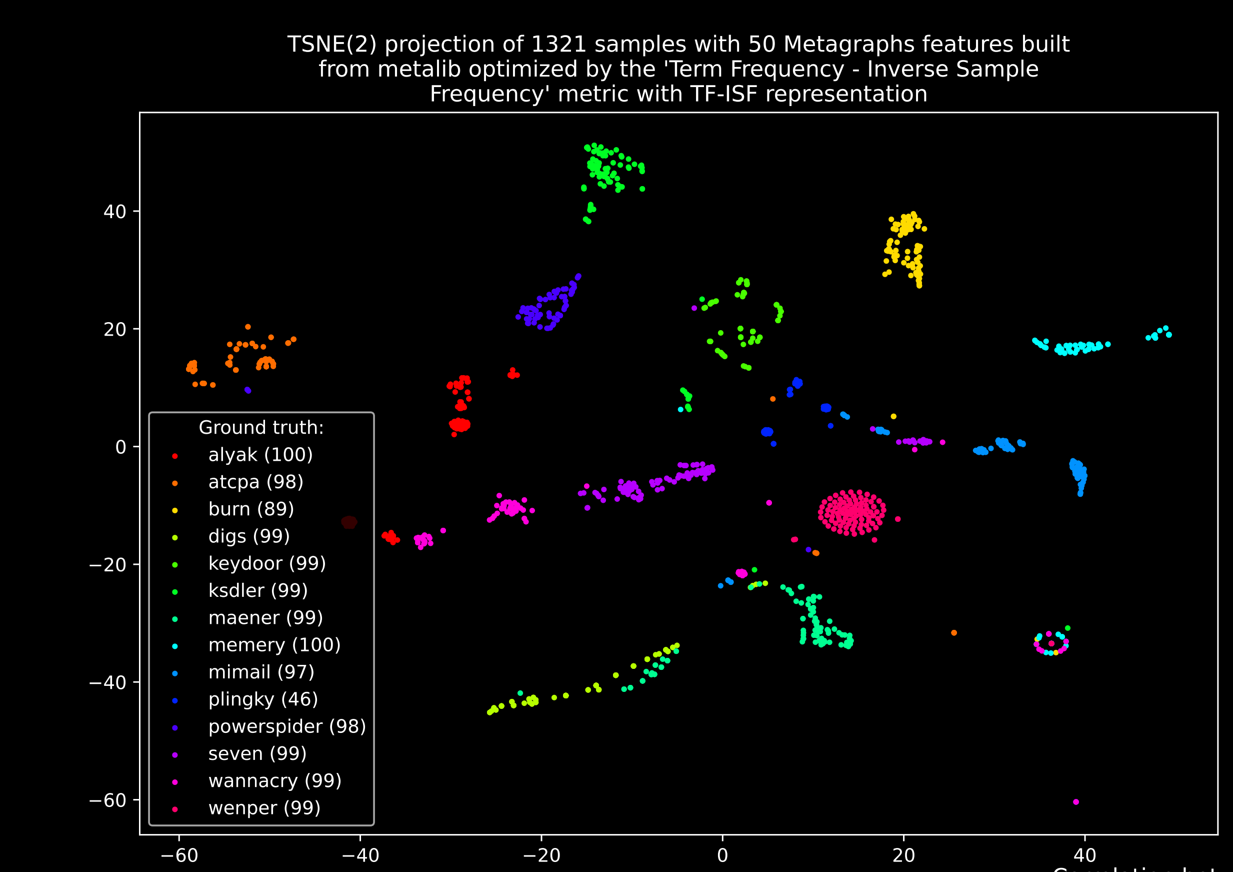
EXPERIMENTAL DETAILS

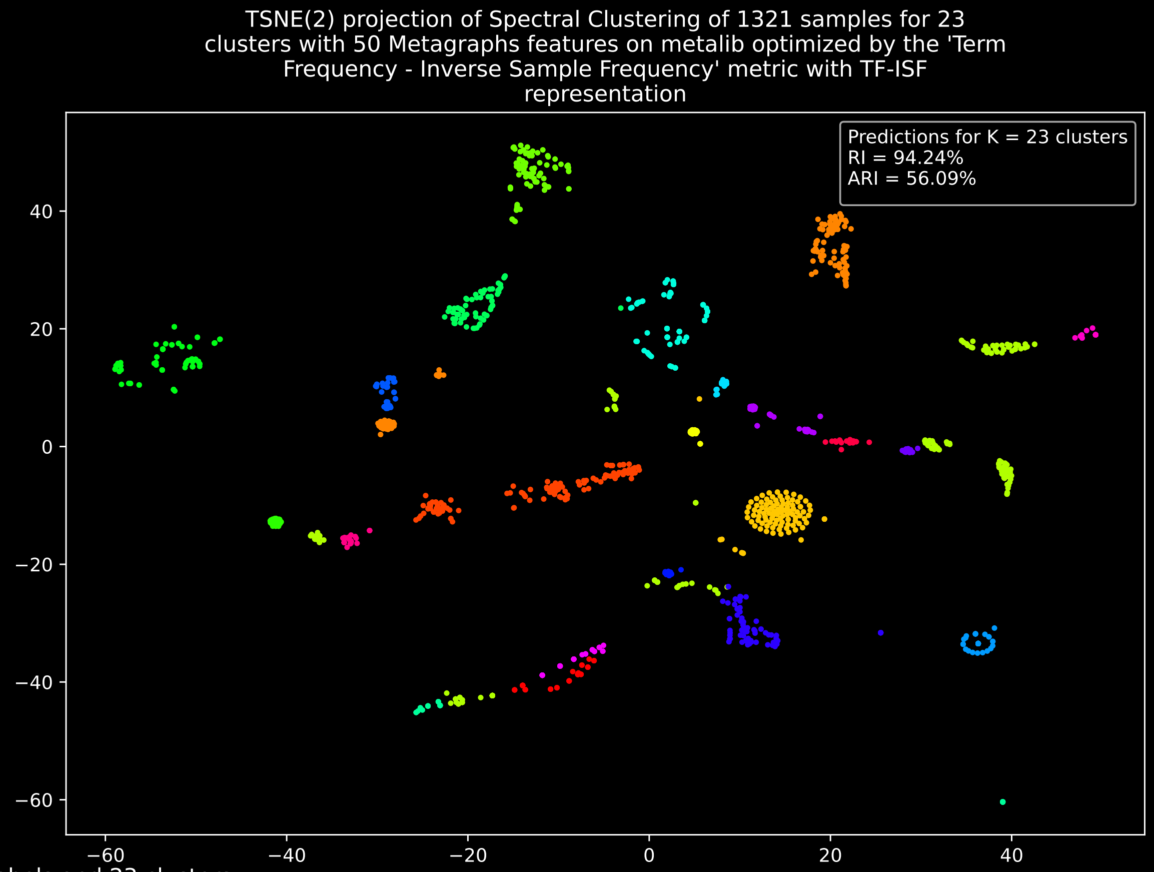
- We experimented with several metrics, I'll describe the best one
- TF-ISF (Term-Frequency / Inverse Sample Frequency) inspired from TF-IDF
- For one metagraph, it's the multiplication of two terms:
  - The number of occurrences in all BAGUETTEs
  - The logarithm of the inverse of the number of BAGUETTEs matching this metagraph
- Intuitively, we want common metagraphs that are only present in a few BAGUETTEs

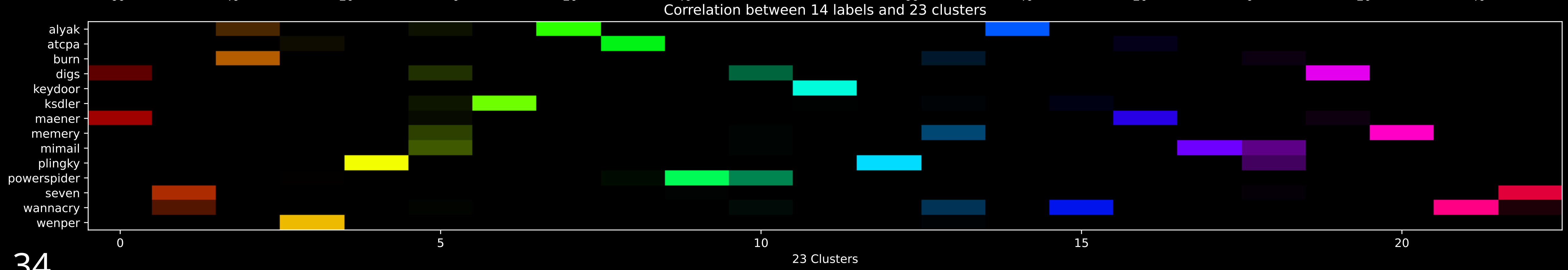
EXPERIMENTAL DETAILS

- Clustering is made in a vector space where each BAGUETTE graph is described by the number of matches for each metagraph
- So, if we have 100 metagraphs, each BAGUETTE is represented by a vector of 100 numbers
- We tried many clustering algorithms, and finally chose spectral clustering

RESULTS?


TSNE(2) projection of 1321 samples with 50 Metagraphs features built from metalib optimized by the 'Term Frequency - Inverse Sample Frequency' metric with TF-ISF representation
Ground truth:
alyak (100)
atcpa (98)
burn (89)
digs (99)
keydoor (99)
ksdler (99)
maener (99)
memery (100)
mimail (97)
plingky (46)
powerspider (98)
seven (99)
wannacry (99)
wenper (99)
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40


TSNE(2) projection of Spectral Clustering of 1321 samples for 23 clusters with 50 Metagraphs features on metalib optimized by the 'Term Frequency - Inverse Sample Frequency' metric with TF-ISF representation
Predictions for K = 23 clusters
RI = 94.24%
ARI = 56.09%
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40

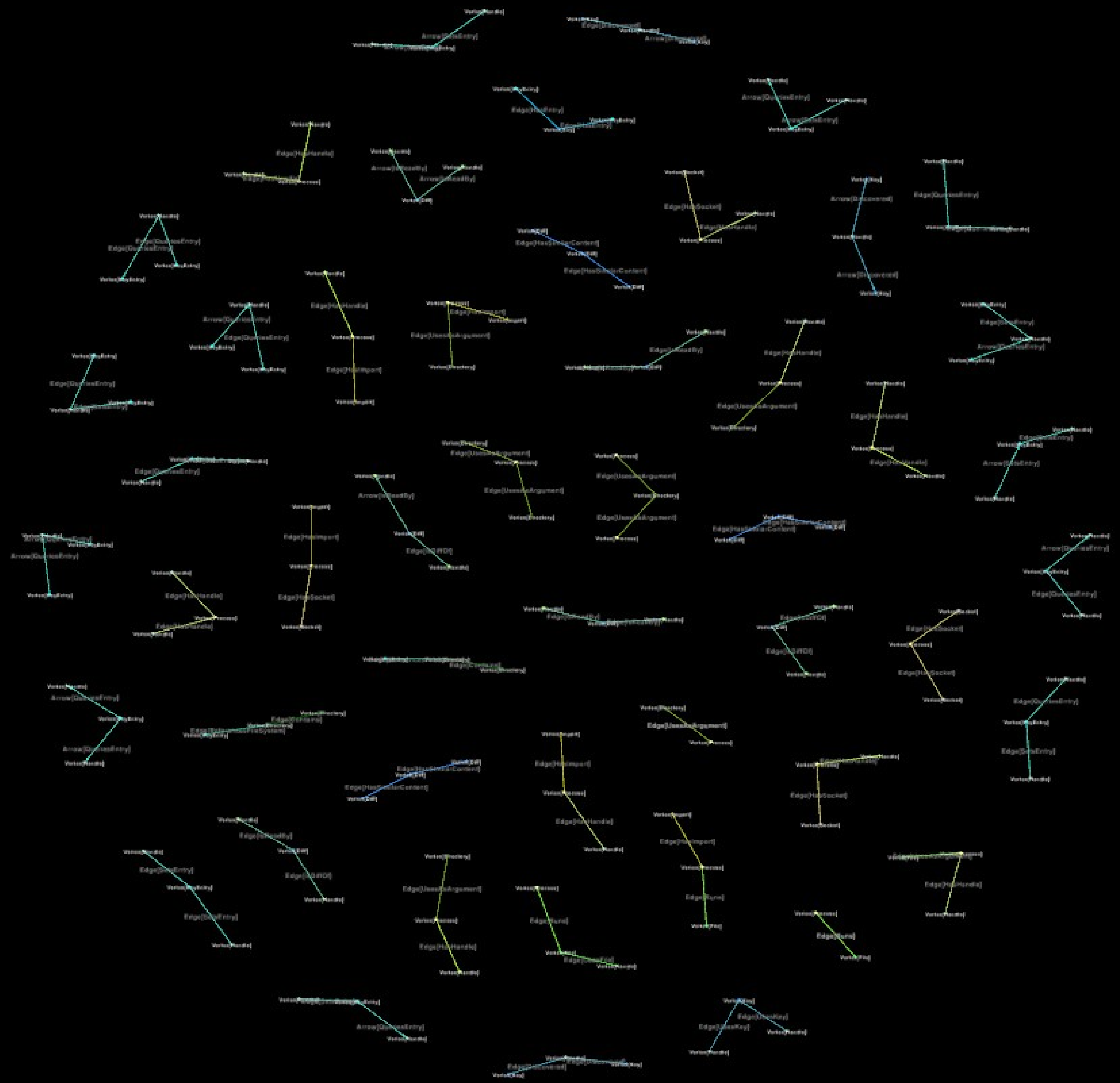

Correlation between 14 labels and 23 clusters
alyak
atcpa
burn
digs
keydoor
ksdler
maener
memery
mimail
plingky
powerspider
seven
wannacry
wenper
0
5
10
15
20
23 Clusters

GOALS

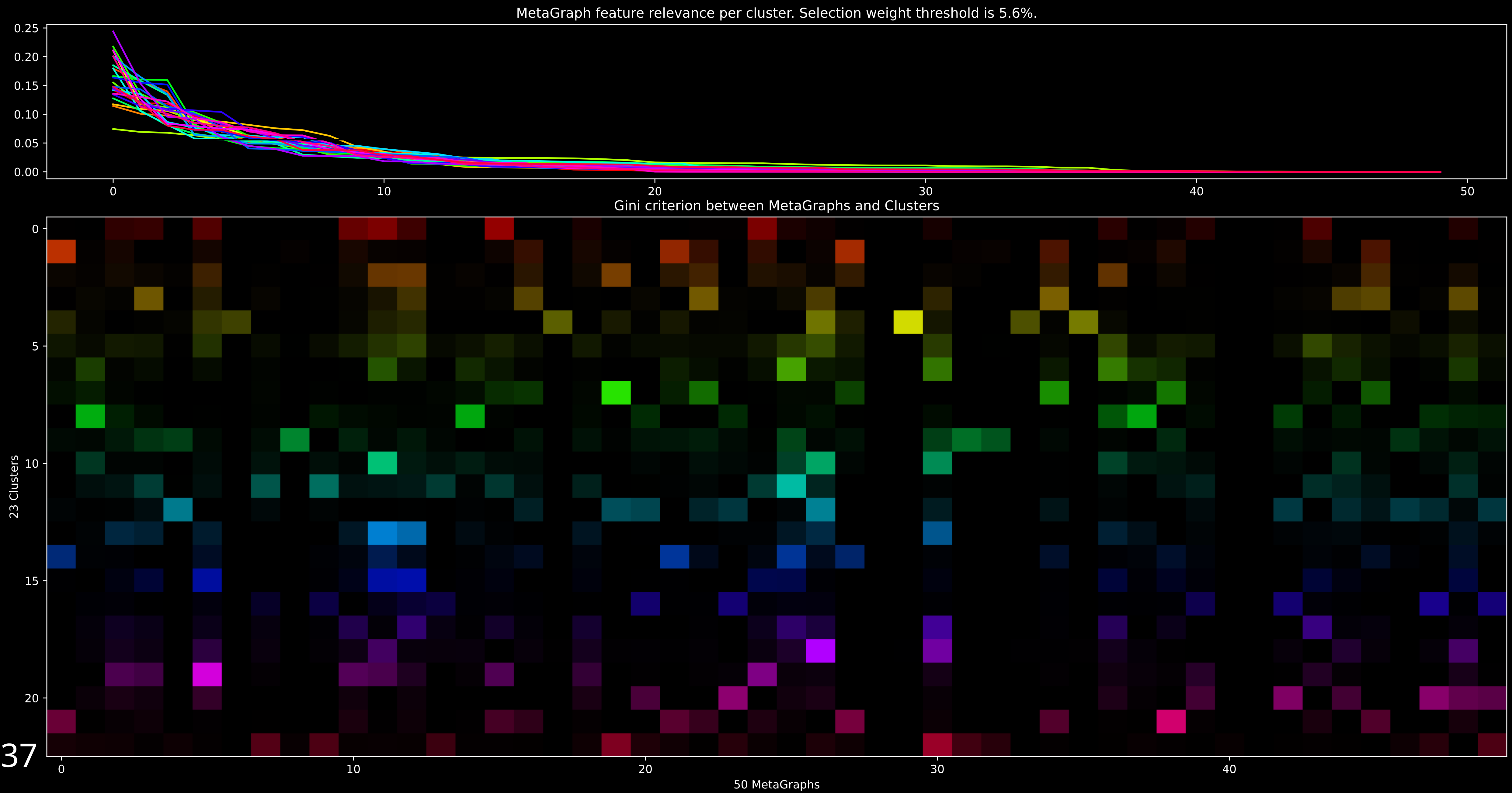
Given a unlabelled dataset of malware samples, how can we:

- Recover clusters of behaviors that hopefully match the families/classes → encouraging results
- Recover the behavioral patterns characteristic of each of these clusters

AND THE EXPLANATIONS?

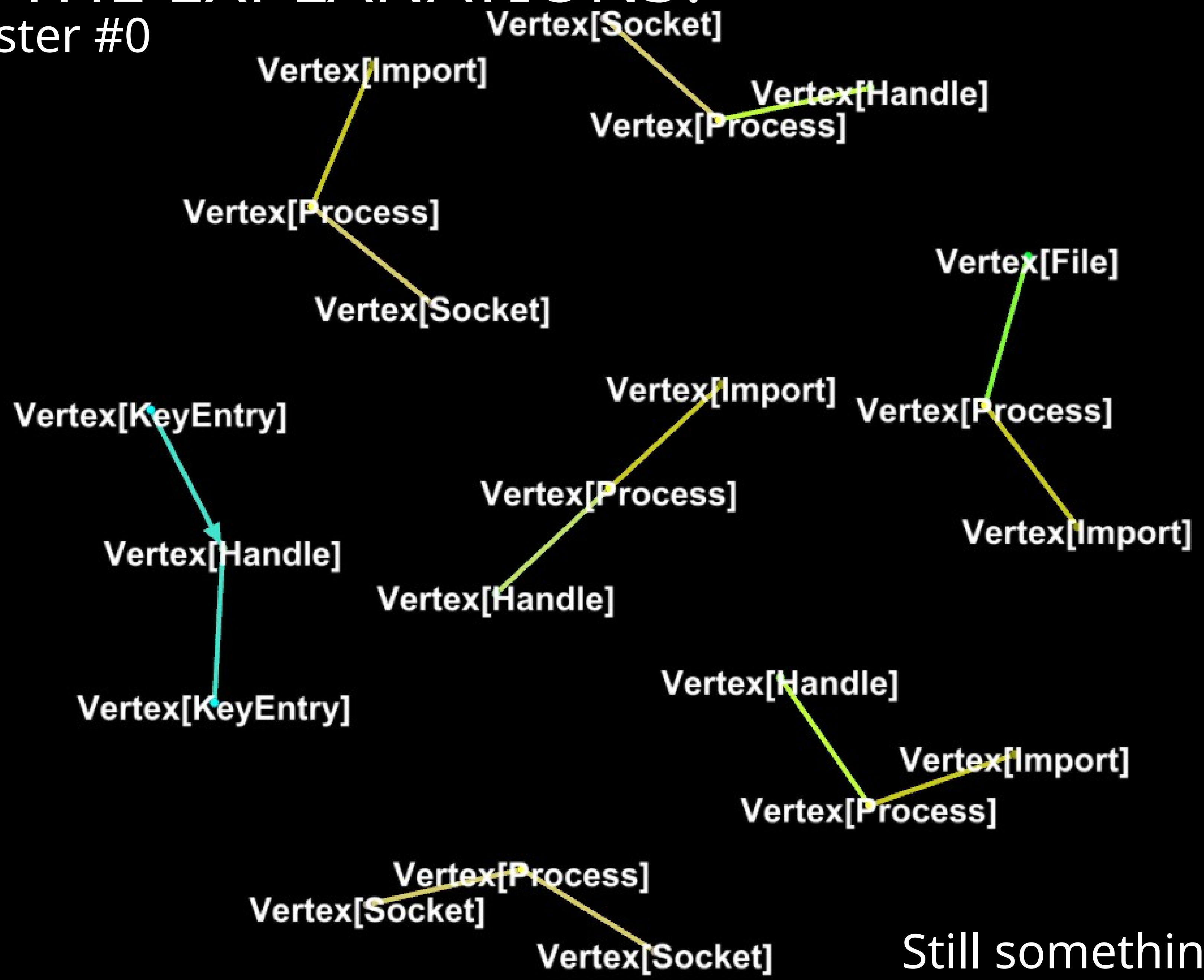


AND THE EXPLANATIONS?



AND THE EXPLANATIONS?

For cluster #0



Still something to improve…

NEXT STEPS

- This is still a work in progress
- Proper statistical analysis
    - More iterations to learn more complex metagraphs (with random selection)
    - Clustering algorithm comparison
    - Average accuracy metrics estimation
- Comparison to SOTA of heterogeneous graph pattern mining
- Better metagraph generation rules with better explainability

CONCLUSION

- BAGUETTE encapsulates the behavioral information of malware samples in a reasonable scale.
- Metagraphs allows to make advanced searches through a dataset of BAGUETTES.
- BAGUETTES give a visualization advantage when analyzing malware samples.

- From BAGUETTE, CROISSANT can:
  - Learn explainable behavioral signatures
  - Can differentiate malware families
  - No need of labelled data