BinaryInferno:
A Semantic-Driven Approach to Field Inference for Binary Message Formats

Jared Chandler
Tufts University

Adam Wick
Fastly

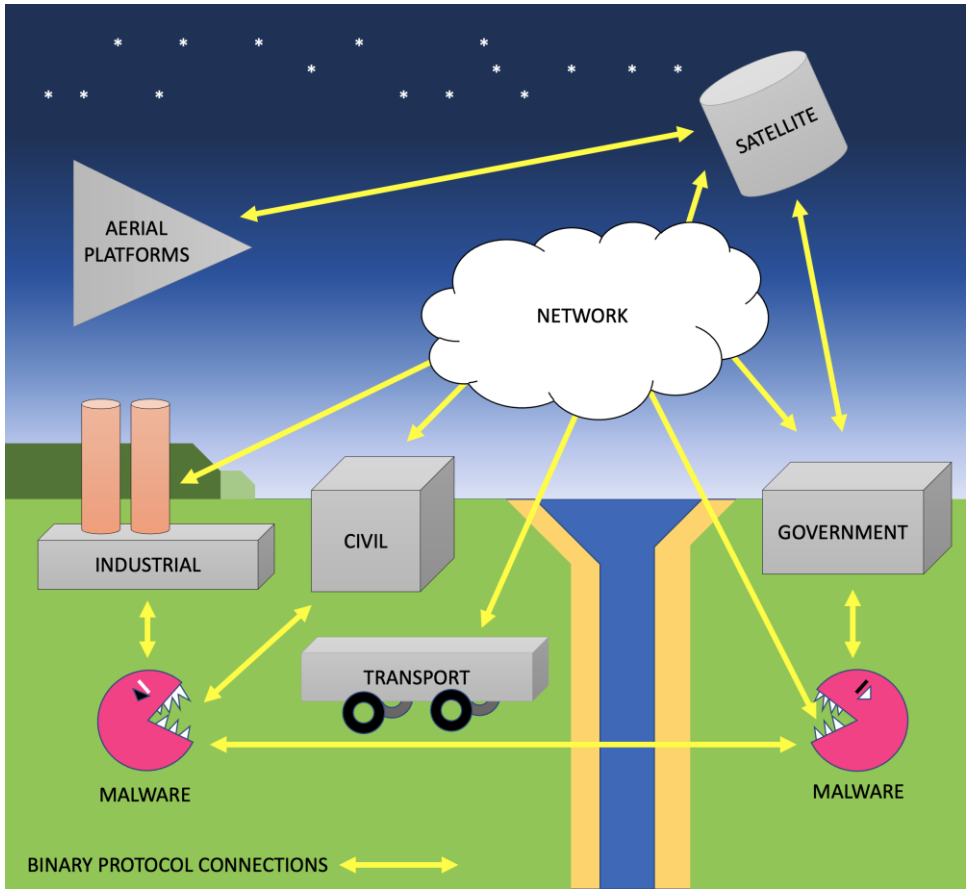
Kathleen Fisher
DARPA

This material is based upon work partly supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-19-C-0073. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Approved for Public Release, Distribution Unlimited.

Talk Overview

- Introduction to Protocol Reverse Engineering
- BinaryInferno
  - How we identify fields.
  - How we infer the best possible format.
- Evaluation
  - BinaryInferno significantly improves on current methods.
- Limitations & Future Work

Protocol Reverse Engineering



Why Reverse Engineer Protocols?

- Find Security Vulnerabilities
- Identify Malicious Traffic
- Validate Specifications
- Rehost Legacy Systems
- Data Exfiltration

```
0103FF00
010B010200070104D3ADC0
DE01070A04BADC0DE5
02C0A80001C0A80002C0A8
000302AC14020F
040001
04000A
```

Binary

```
.38400,38400....#.Sandbox:
0.0....'..DISPLAY.Sandbox:
0.0......xterm.............
..login:tteesstt.Password:
capture.$ uunnaammee  --
aa.Linux cm4116 2.6.30.2-
uc0 #3 Tue Feb 22 00:57:18
EST 2011 armv4tl unknown$
...$ eexxiitt.logout
```

Text Based

How are Protocols Currently Reverse Engineered?

A. Static Analysis / Disassembly

*Examine **source code** manually / automatically*

B. Dynamic Analysis

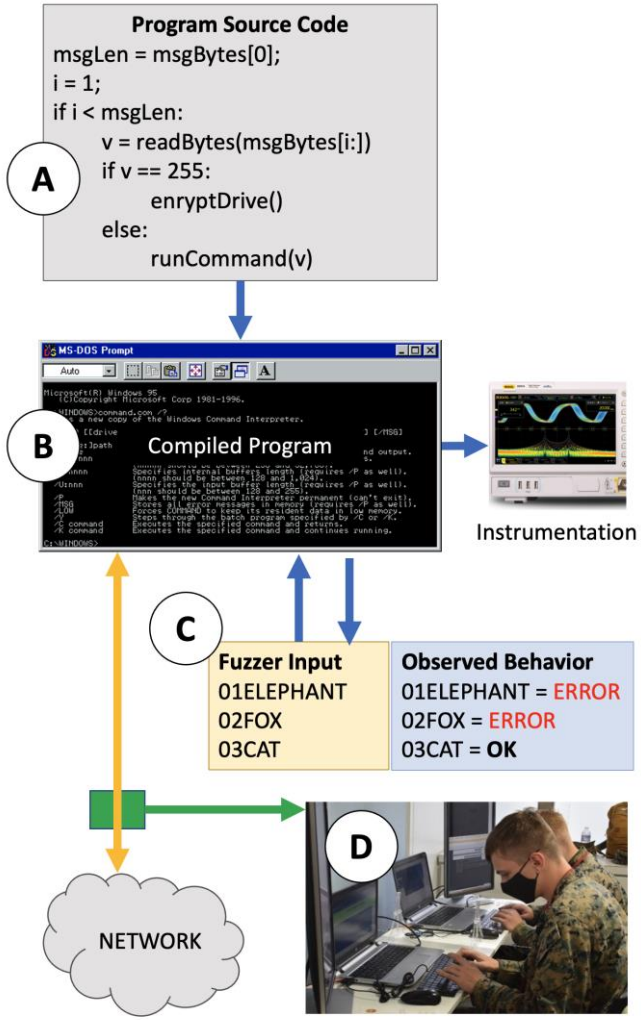
*Examine a **running instance** of the program*

C. Fuzzing

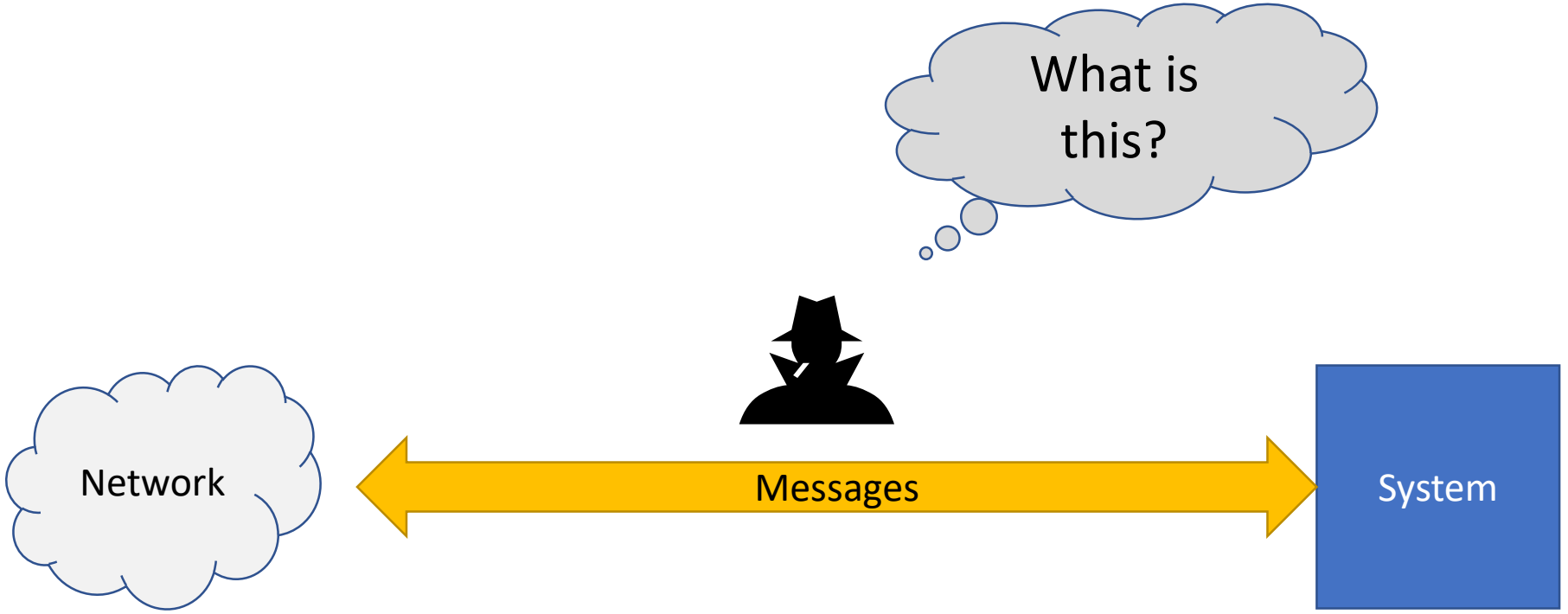
*Send various inputs to **running system** and observe.*

D. Manual Reverse Engineering

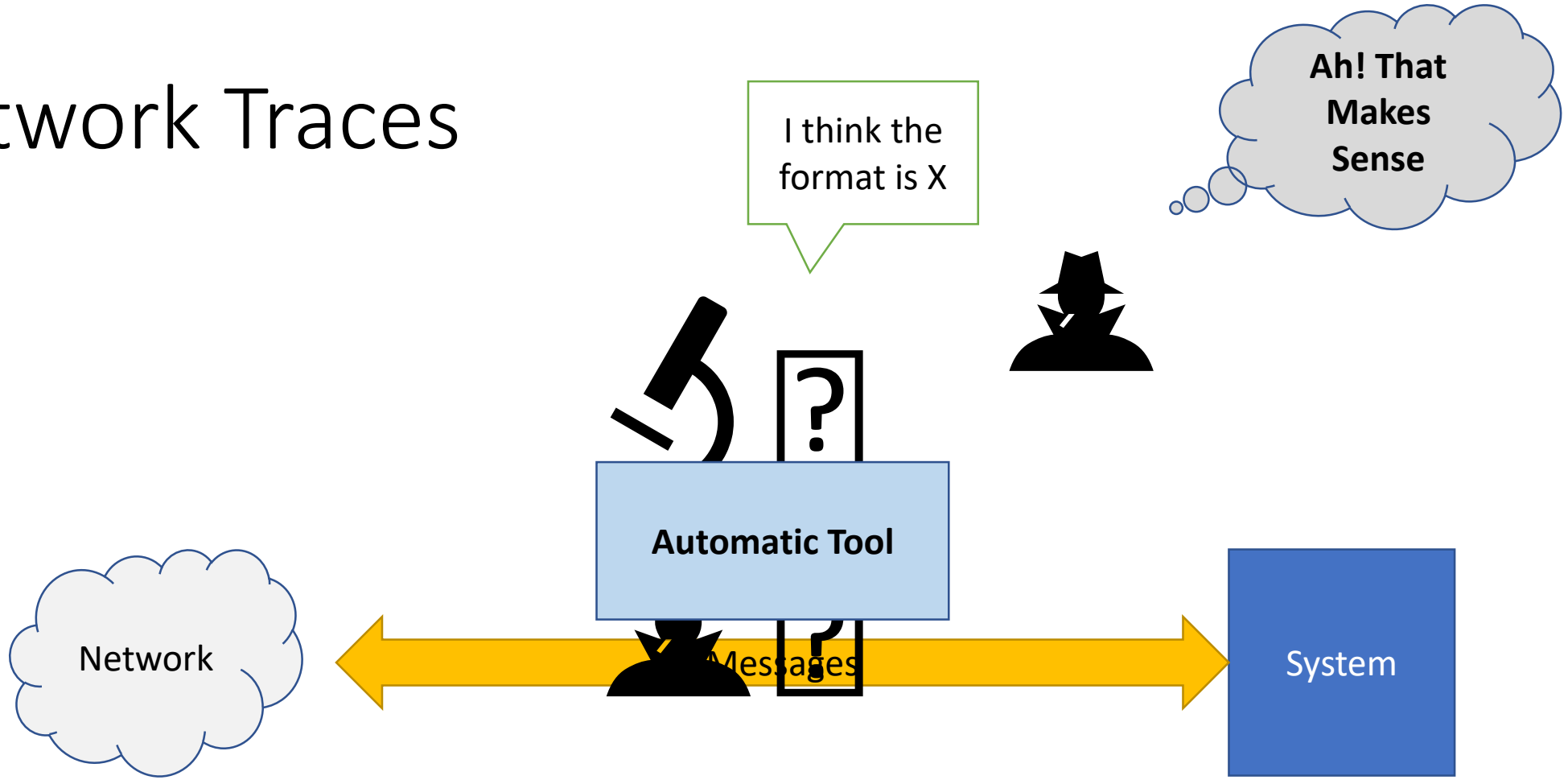
Examine the network data by hand



Network Traces



Network Traces



What's New About Our Approach?

The observation of binary protocol messages implies the existence of a concrete program to consume these messages.

Messages

```
0103FF00
010B010200070104D3ADC0DE
01070A04BADC0DE5
02C0A80001C0A80002C0A80003
02AC14020F
040001
04000A
```

Deserializing Program

```
tag = read2Bytes()
if tag == 1:
   measureLength = read1Byte()
   i = 0
   while i < measureLength:
      tag = read1Byte()
      len = read1Byte()
      value = readNBytes(len)
      logMeasurement(tag,value)i = i + len + 2
elif tag == 2:
   while targetIP = read4Bytes():
      launchAttack(targetIP)
elif tag ==4:
   driveId = read2Bytes()
   encryptDrive(driveId)
```

Insight: The format of a binary message reflects the structure of the program deserializing it.

We exploit this property by searching for representations based on common program conventions, including atomic datatypes, if/else statements, loops, and more.

BinaryInferno

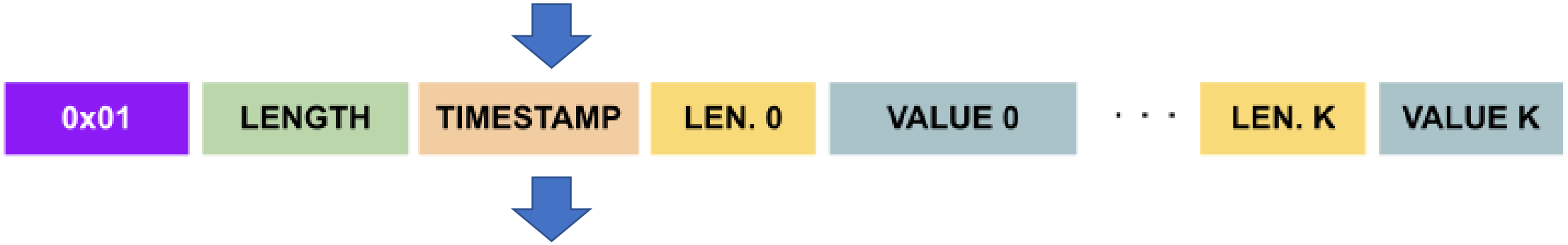
```
Msg1 01000D60A67AED054150504C45
Msg2 01000E60A67AF9064F52414E4745
Msg3 01001160A67B0504504C554D0450454152
```

BinaryInferno

```
01000D60A67AED054150504C45
01000E60A67AF9064F52414E4745
01001160A67B0504504C554D0450454152
```

BinaryInferno

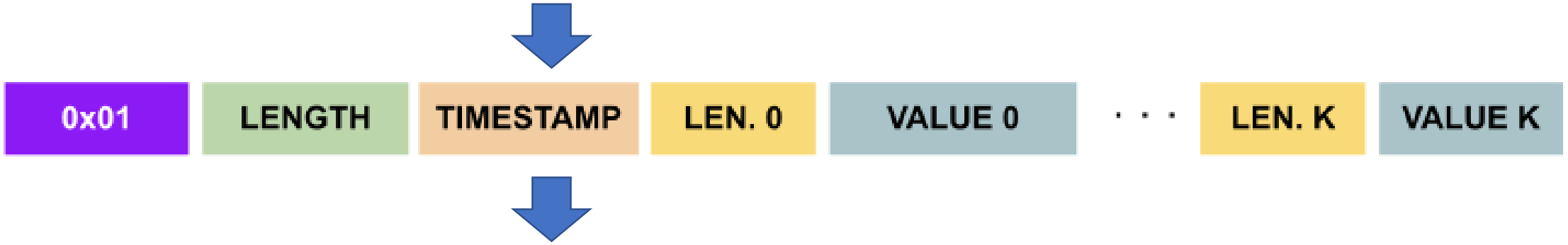
```
01000D60A67AED054150504C45
01000E60A67AF9064F52414E4745
01001160A67B0504504C554D0450454152
```




```
01 000D 60A67AED 05 4150504C45
01 000E 60A67AF9 06 4F52414E4745
01 0011 60A67B05 04 504C554D 04 50454152
```

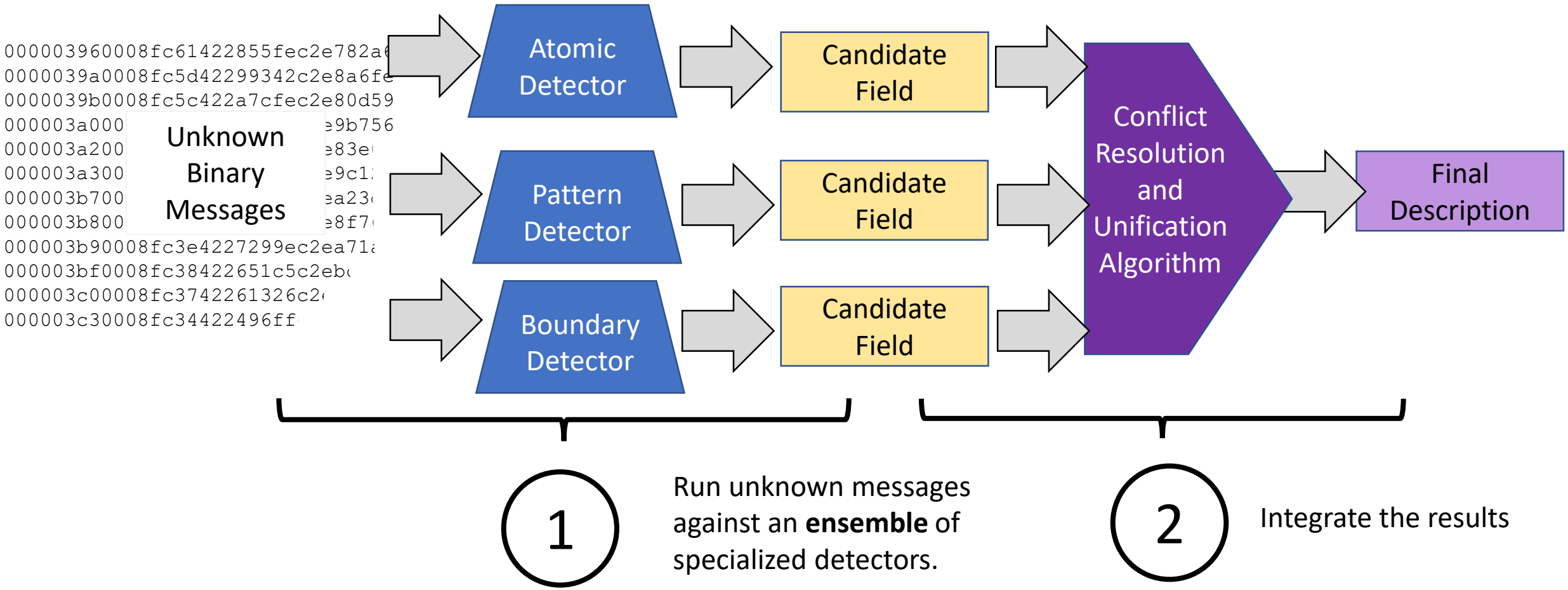
BinaryInferno

```
01000D60A67AED054150504C45
01000E60A67AF9064F52414E4745
01001160A67B0504504C554D0450454152
```

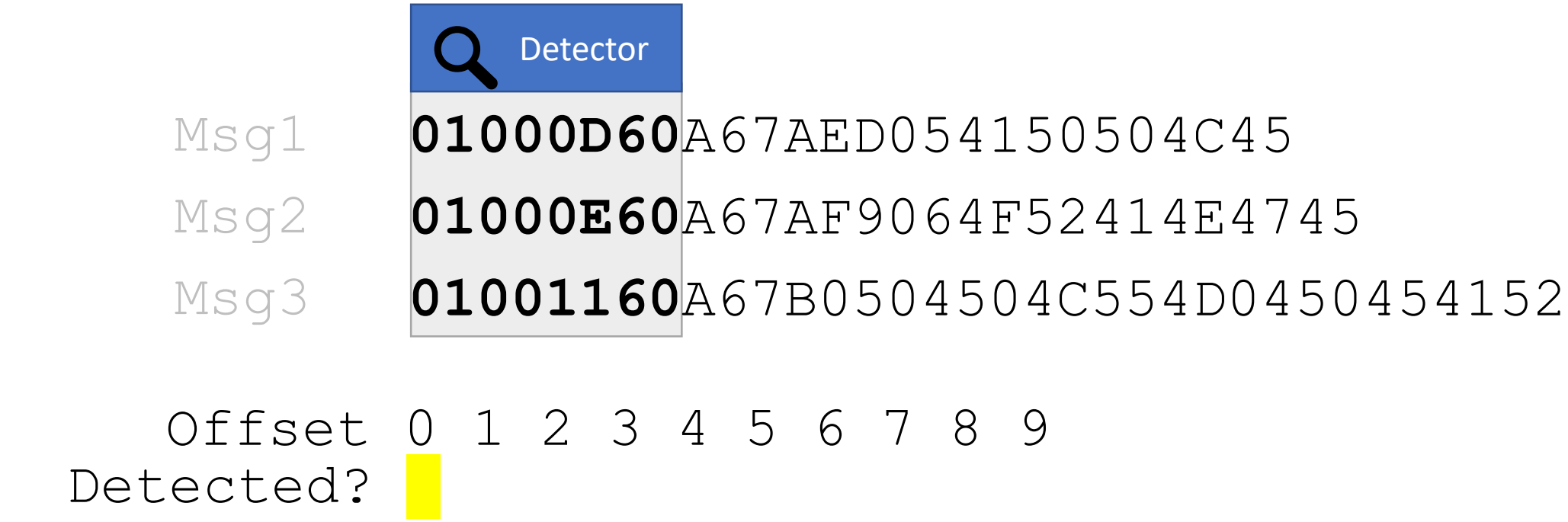



```
01 13 5/20/21 15:06:21 05 4150504C45
01 14 5/20/21 15:06:33 06 4F52414E4745
01 17 5/20/21 15:06:45 04 504C554D 04 50454152
```

Approach

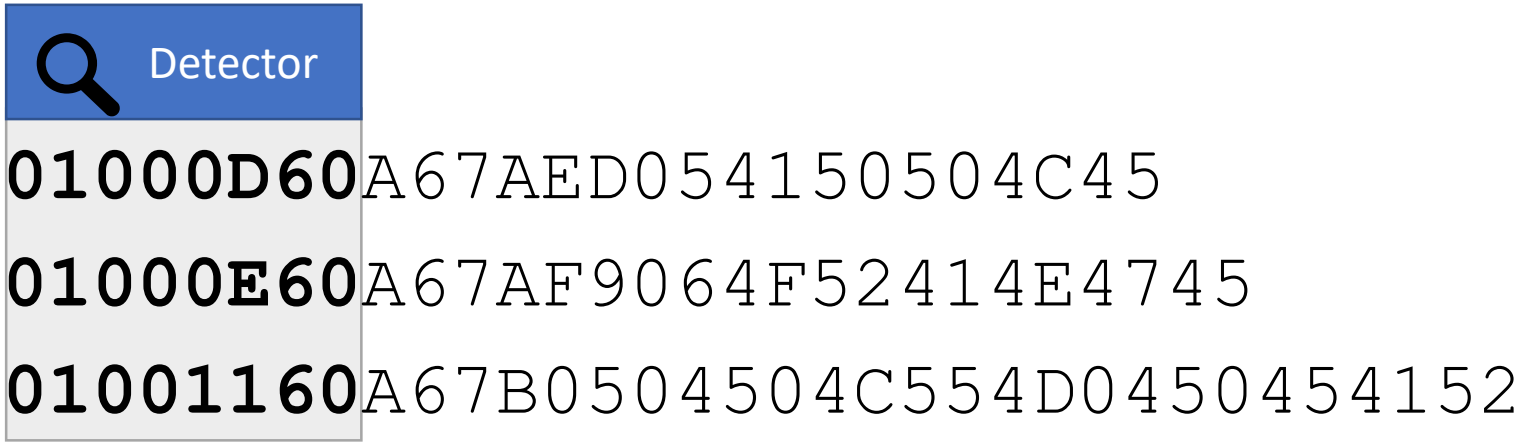

000003960008fc61422855fec2e782a
0000039a0008fc5d42299342c2e8a6fe
0000039b0008fc5c422a7cfec2e80d59
000003a000 e9b756
000003a200 e83e
000003a300 e9c1
000003b700 ea23
000003b800 e8f7
000003b90008fc3e4227299ec2ea71
000003bf0008fc38422651c5c2eb
000003c00008fc3742261326c2
000003c30008fc34422496ff
Unknown Binary Messages
Atomic Detector
Pattern Detector
Boundary Detector
Candidate Field
Candidate Field
Candidate Field
Conflict Resolution and Unification Algorithm
Final Description
1
Run unknown messages against an **ensemble** of specialized detectors.
2
Integrate the results

Detectors Search over Messages



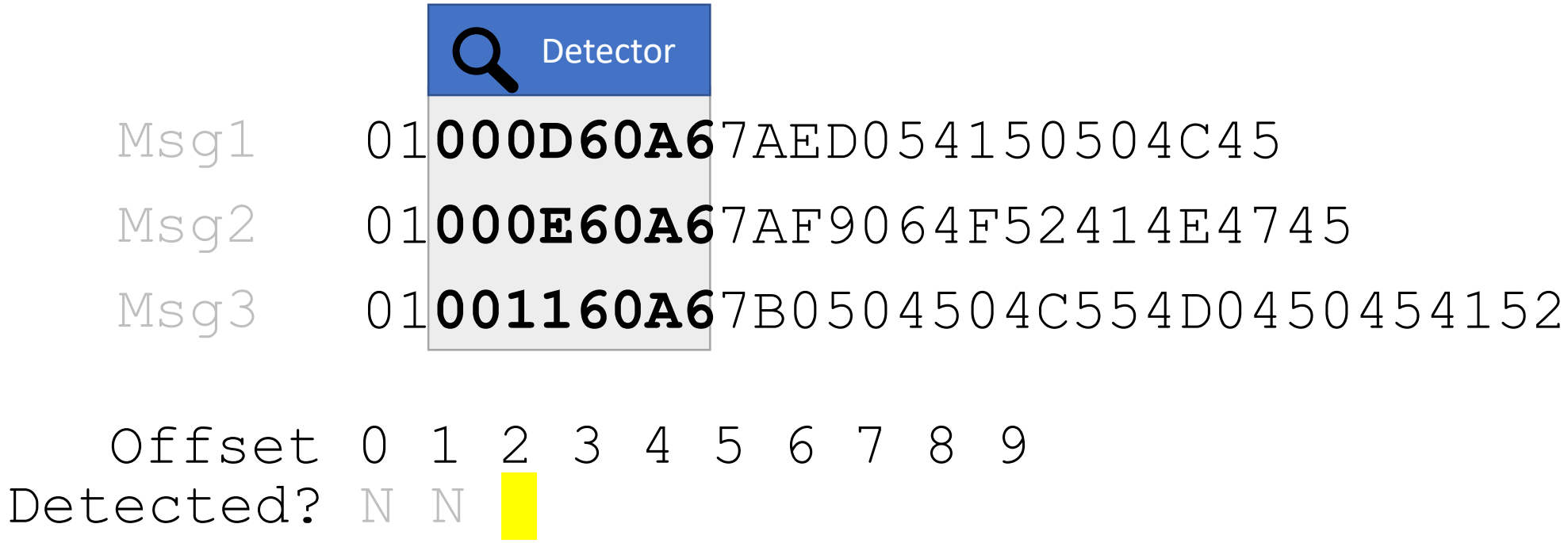
Detector only looks at a vertical **slice** of bytes across the messages.
Slices are either fixed width (such as 4 bytes), or a starting offset in the case of serialization patterns.

Detectors Search over Messages



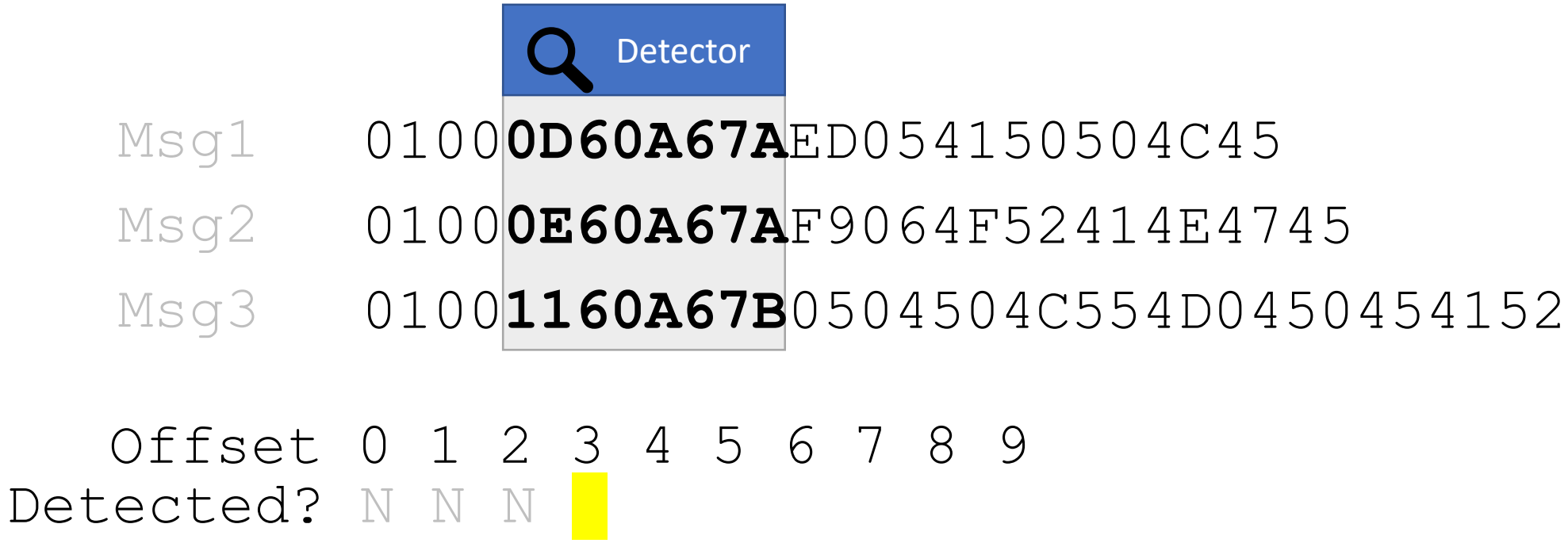
Detector only looks at a vertical **slice** of bytes across the messages.
Slices are either fixed width (such as 4 bytes), or a starting offset in the case of serialization patterns.

Detectors Search over Messages



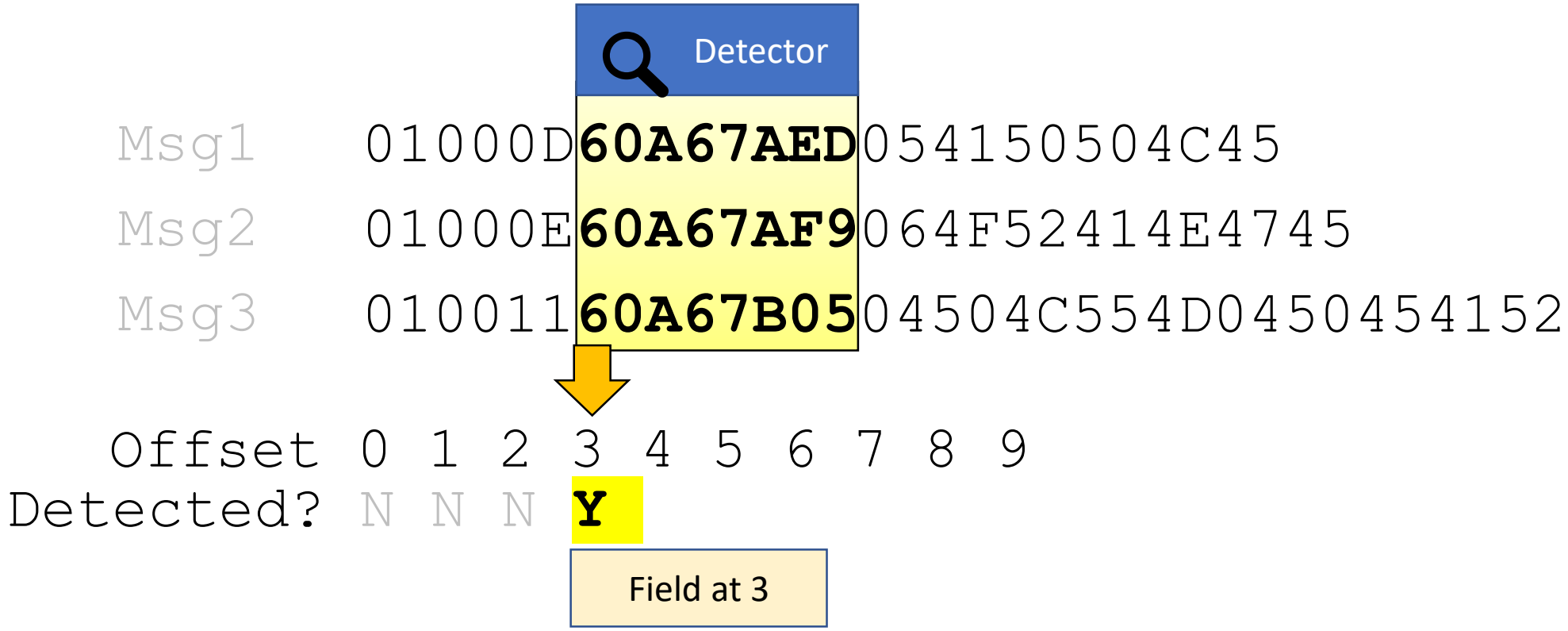
Detector only looks at a vertical **slice** of bytes across the messages.
Slices are either fixed width (such as 4 bytes), or a starting offset in the case of serialization patterns.

Detectors Search over Messages



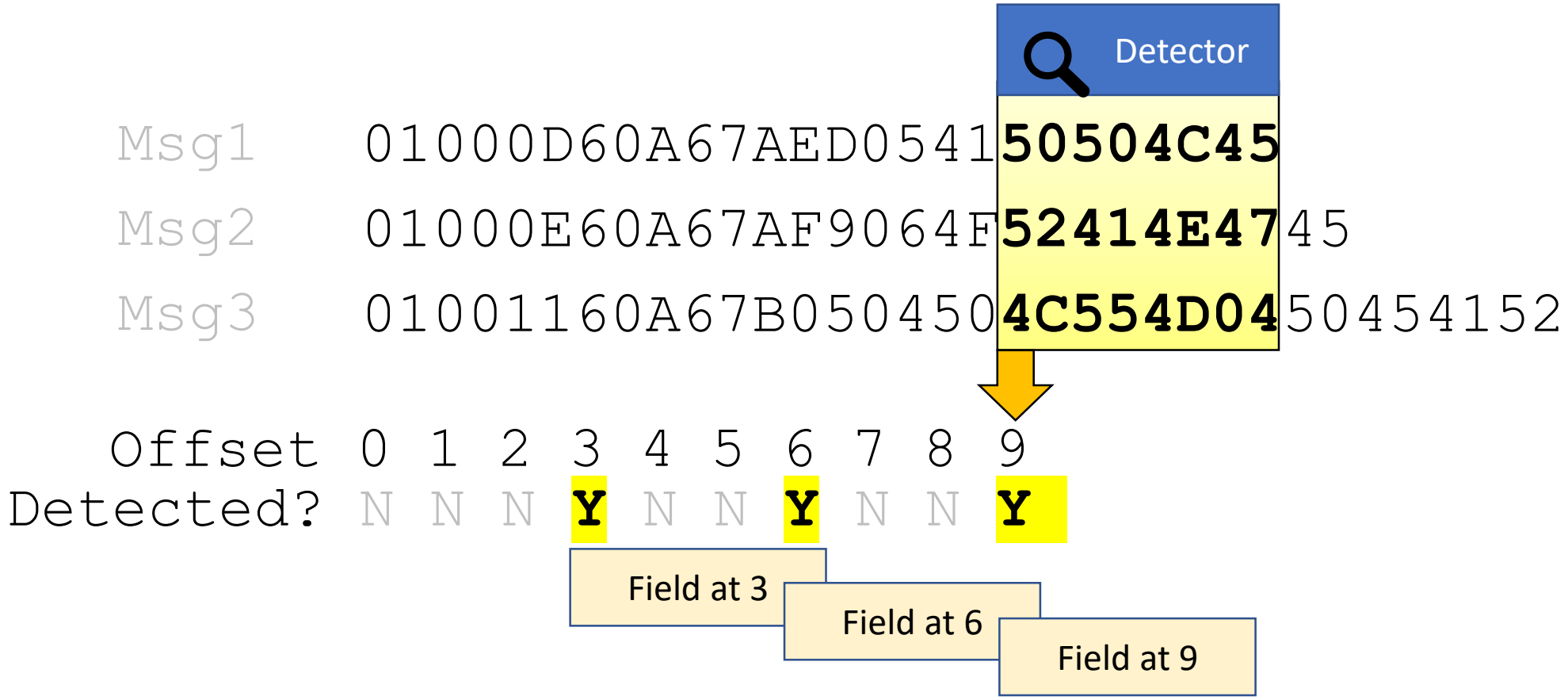
Detector only looks at a vertical **slice** of bytes across the messages.
Slices are either fixed width (such as 4 bytes), or a starting offset in the case of serialization patterns.

Detectors Search over Messages



Detector only looks at a vertical **slice** of bytes across the messages.
Slices are either fixed width (such as 4 bytes), or a starting offset in the case of serialization patterns.

Detectors Search over Messages



Detector only looks at a vertical **slice** of bytes across the messages.
Slices are either fixed width (such as 4 bytes), or a starting offset in the case of serialization patterns.

What's in our Ensemble of Detectors?

Atomic Data Types

- Floats
- Length Fields
- Timestamps

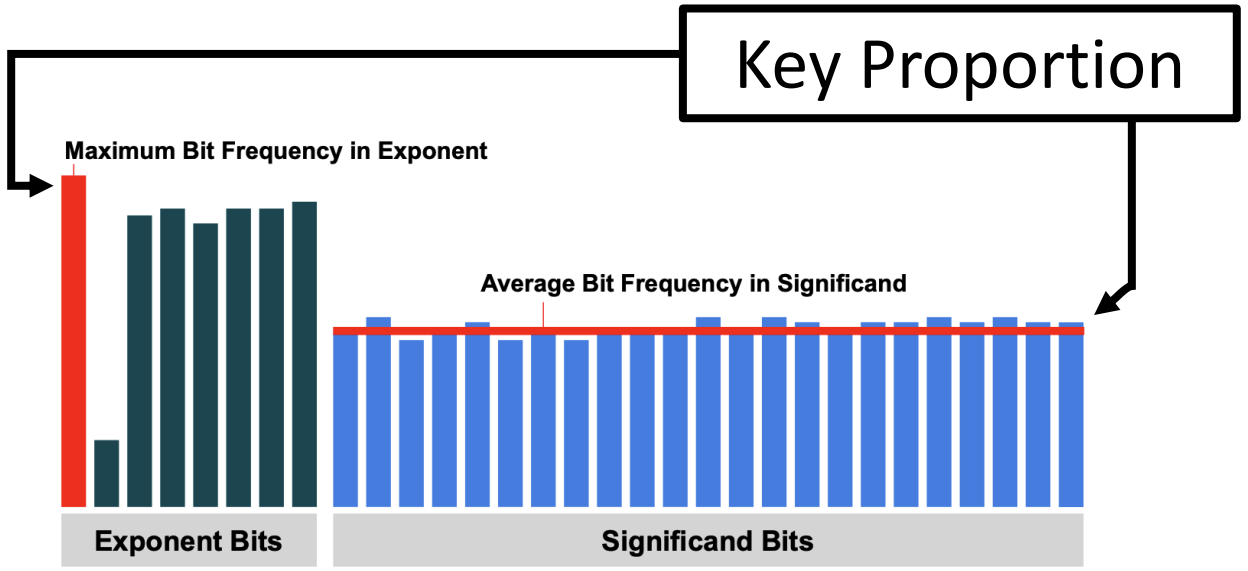
Serialization Patterns

- Length Value / Type Length Value
- Depth First Search using Iterative Deepening

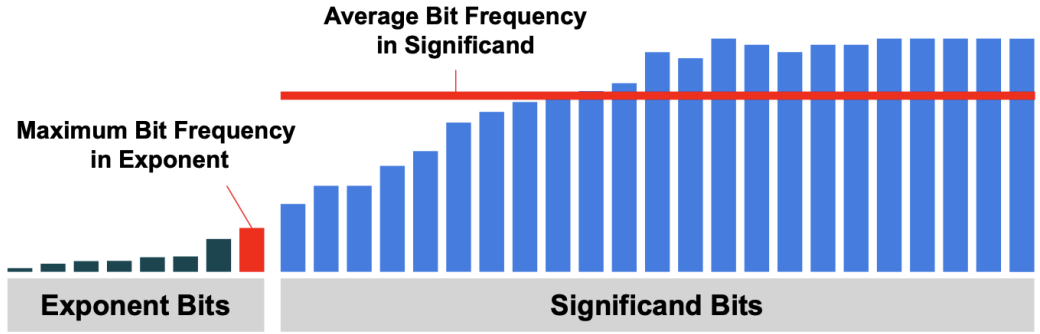
Entropy Boundaries

- For previously unseen field types using Shannon Entropy

IEEE 754 Floats



Floats

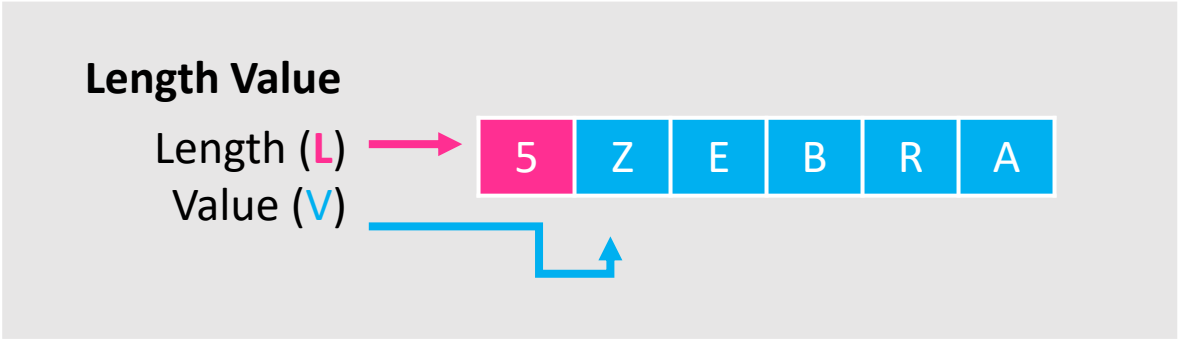


Integers

Serialization Pattern Grammar

$$
\begin{aligned}
\langle PATTERN \rangle &::= L(V)^{[L]} \\
&\mid TL(V)^{[L]} \\
\langle T,L,Q \rangle &::= \texttt{BYTE}^{N}\ \texttt{(1} \leq \texttt{N} \leq \texttt{4)} \\
\langle V \rangle &::= \texttt{BYTE} \\
\langle FIELD \rangle &::= \langle VLFIELD \rangle \\
&\mid \texttt{BYTE}^{P}\ \texttt{(P} \geq \texttt{1)} \\
\langle VLFIELD \rangle &::= \langle PATTERN \rangle \\
&\mid Q(\langle PATTERN \rangle)^{[Q]} \\
&\mid Q(VV)^{[Q]} \mid \ldots \mid Q(VVVVVVVVV)^{[Q]} \\
\langle FORMAT \rangle &::= \langle FIELD \rangle \\
&\mid \langle FIELD \rangle\ \langle FORMAT \rangle
\end{aligned}
$$

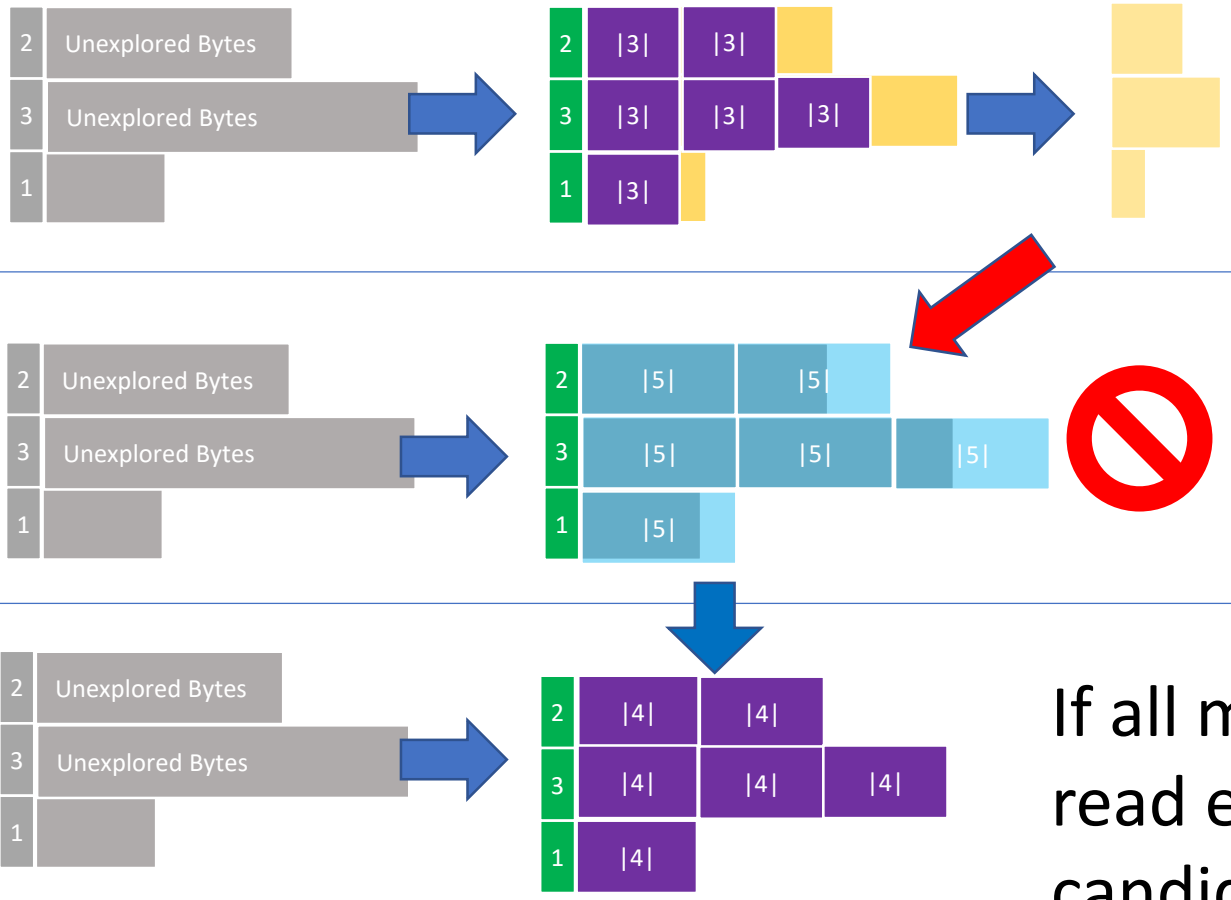
Grammar 1: Serialization Pattern Description Grammar



$$
\begin{aligned}
\langle STAR\text{-}PAT \rangle &::= \langle PATTERN \rangle^{*}\ \texttt{BYTE}^{S}\ \texttt{(S} \geq \texttt{0)} \\
\langle STAR\text{-}FMT \rangle &::= \langle FORMAT \rangle \\
&\mid \langle FORMAT \rangle\ \langle STAR\text{-}PAT \rangle \\
&\mid \langle STAR\text{-}PAT \rangle
\end{aligned}
$$

Grammar 2: Star Pattern Description Grammar

Fitting Serialization Patterns to Data



Apply a single serialization pattern to all messages. Recurse on any unread bytes.

If a pattern tries to read more bytes than exist for *any* message, try a different pattern.

If all message bytes are read exactly, report as candidate description

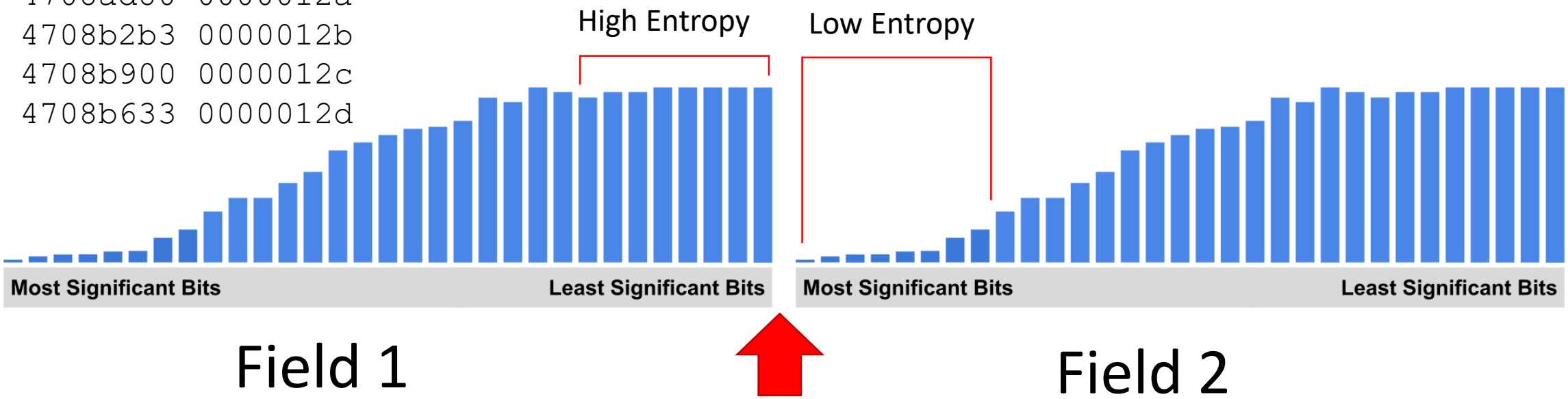
Iterative Deepening with Parallelization

Entropy Difference Detector

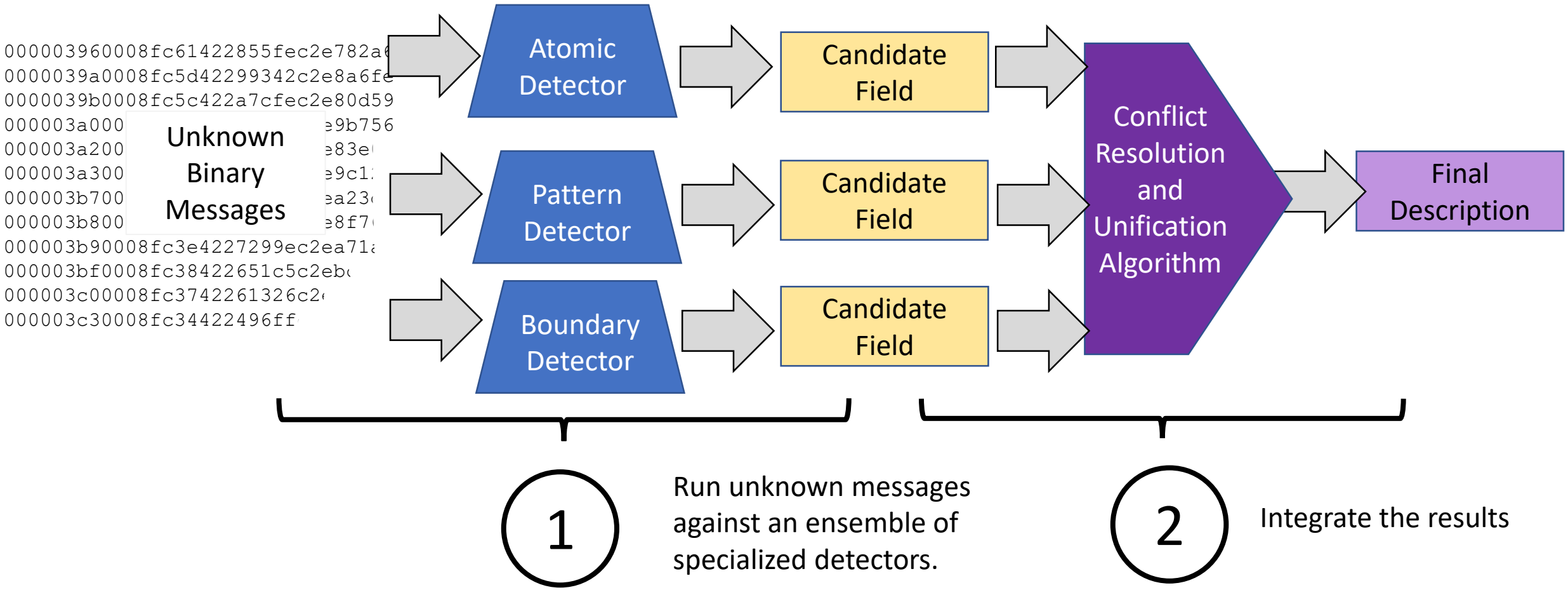
Shannon Entropy $H(\mathbf{M}_k) = -\Sigma_{v \in \mathbf{M}_k} P(v)\, log\, P(v)$

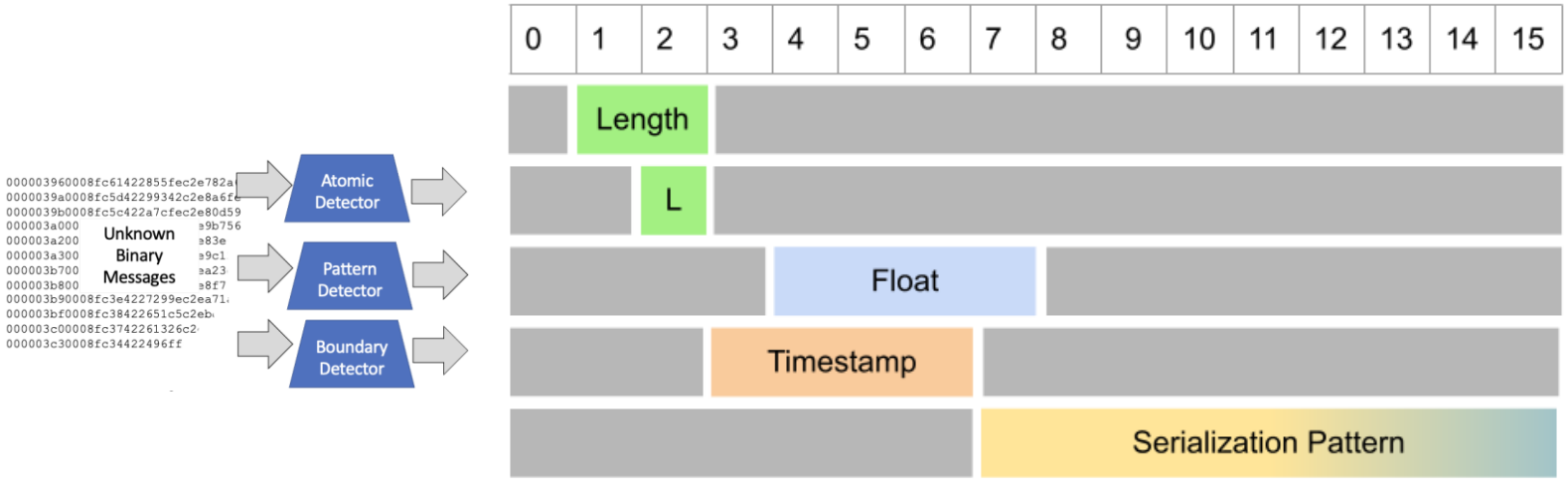
```
4708be66 0000012d
4708bdb3 0000012c
4708ad80 0000012a
4708b2b3 0000012b
4708b900 0000012c
4708b633 0000012d
```

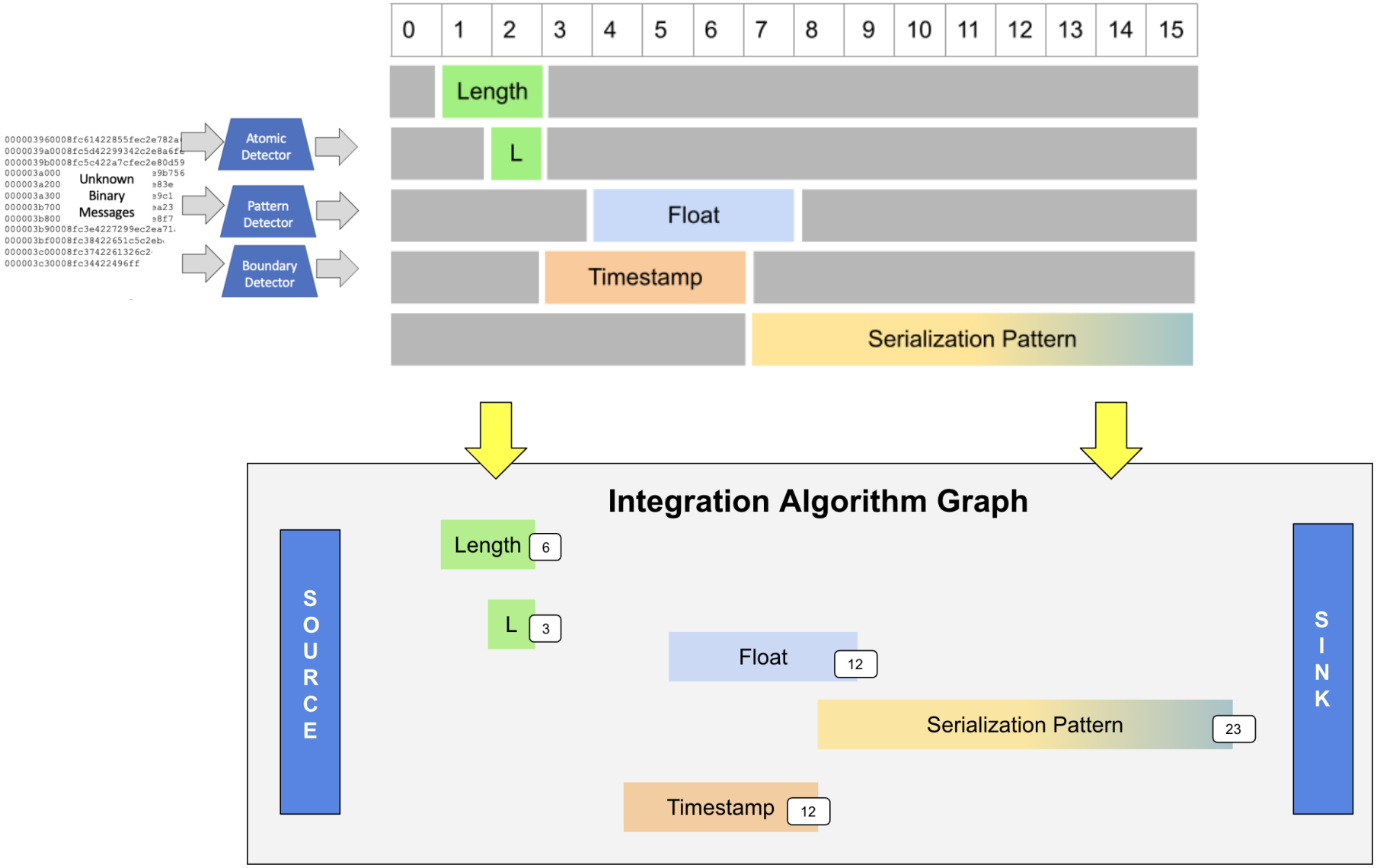
H(Byte K) - H(Byte K+1) > 1

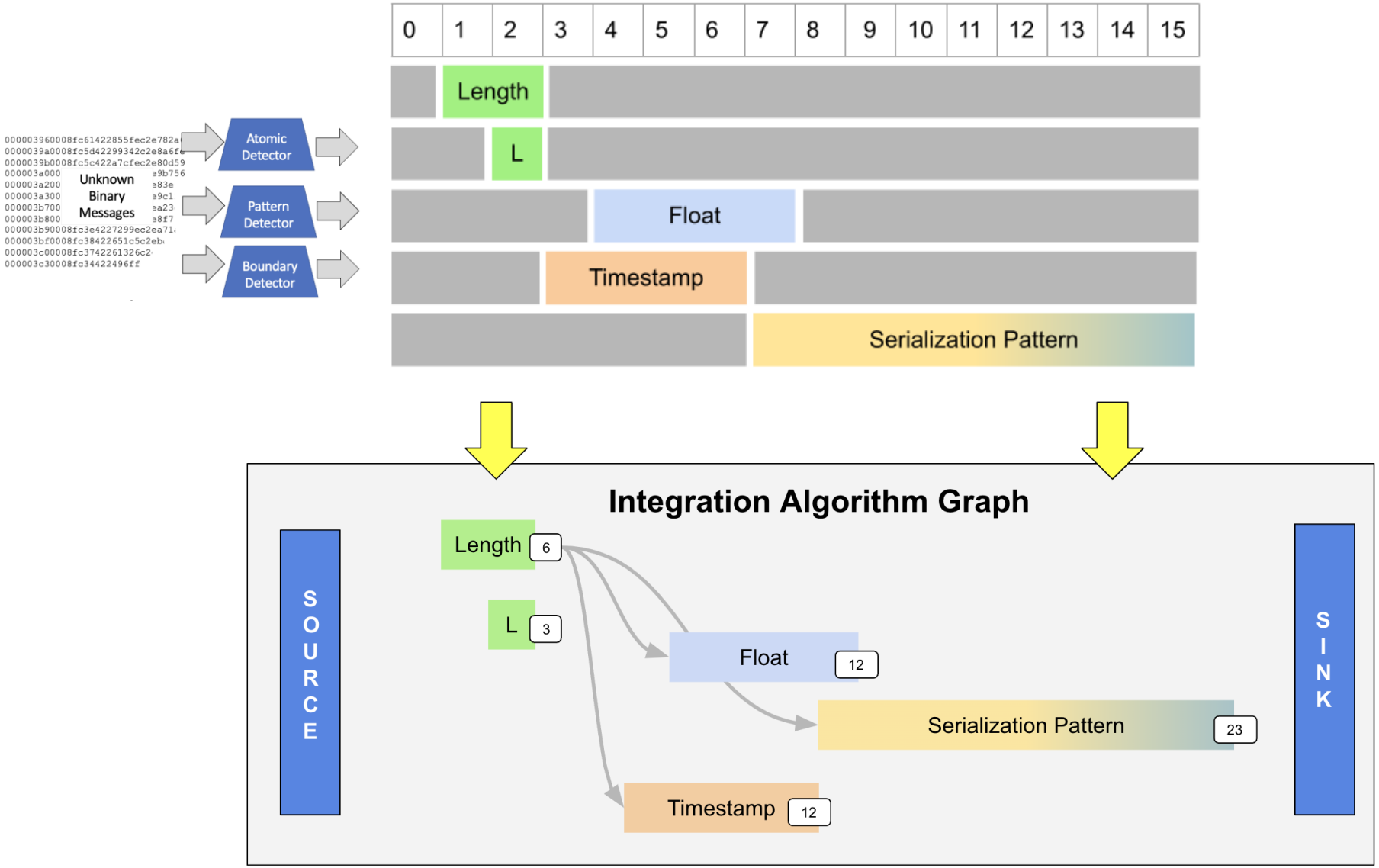


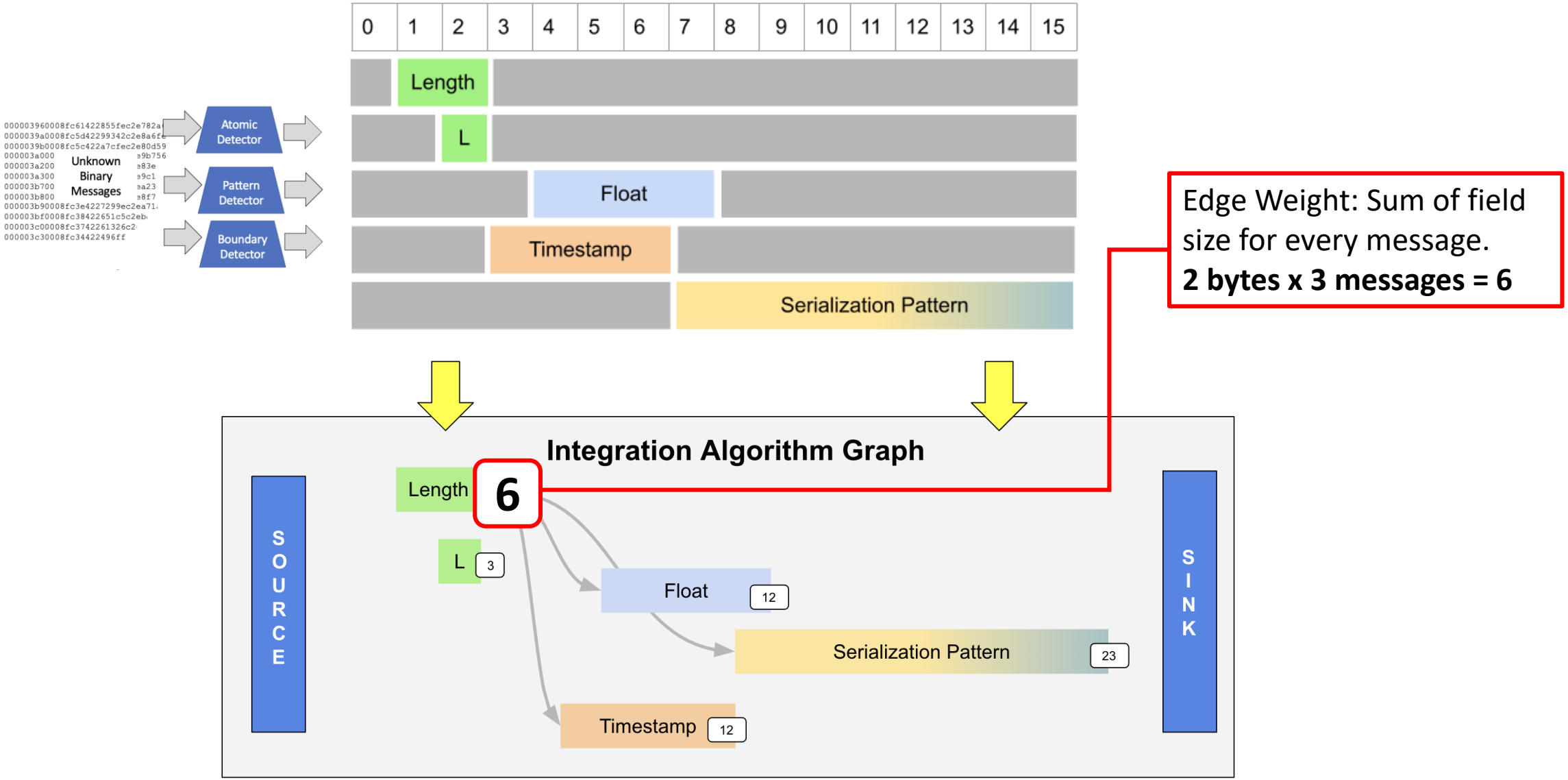
Approach

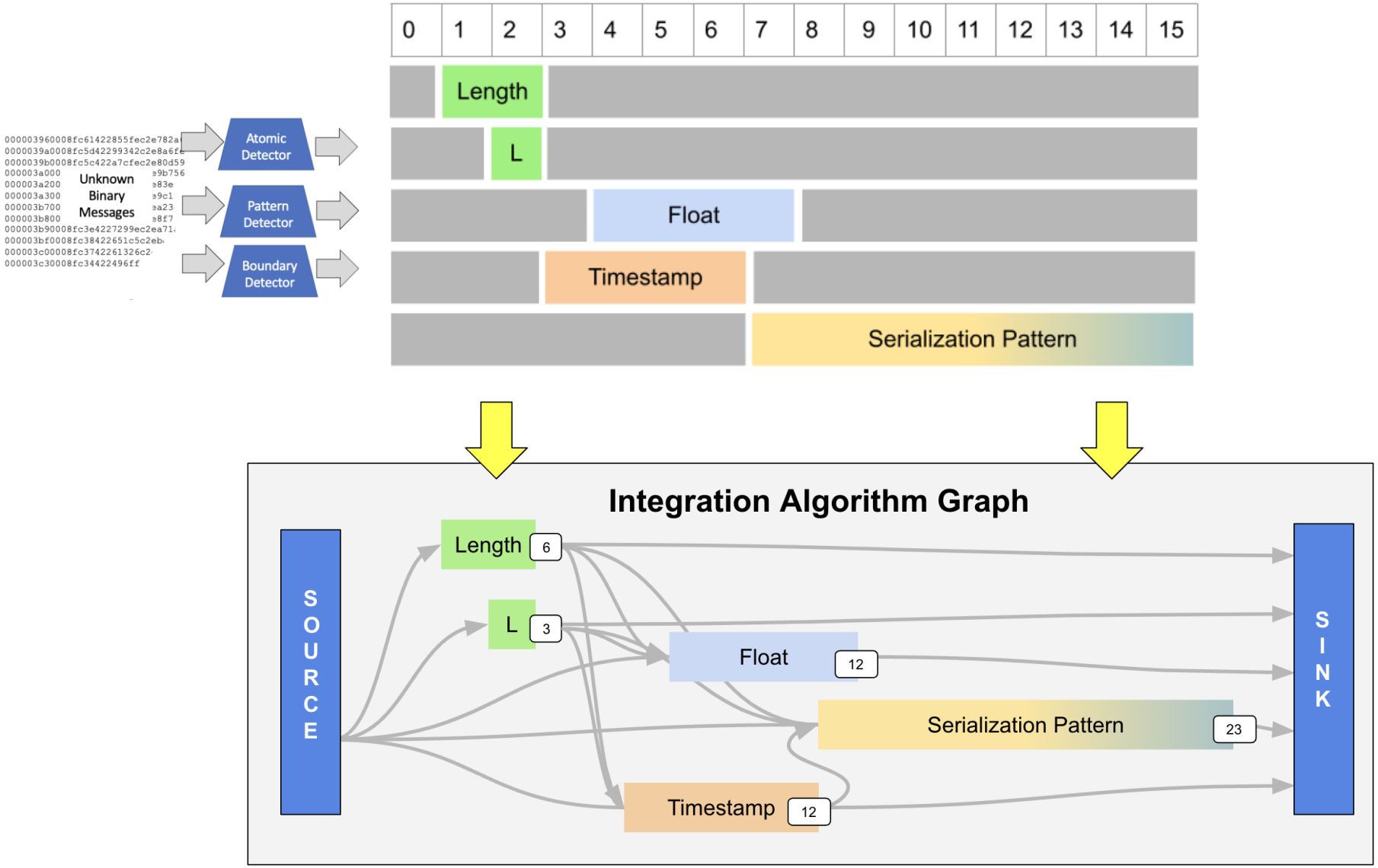

000003960008fc61422855fec2e782a
0000039a0008fc5d42299342c2e8a6fe
0000039b0008fc5c422a7cfec2e80d59
000003a000 e9b756
000003a200 e83e
000003a300 e9c1
000003b700 ea23
000003b800 e8f7
000003b90008fc3e4227299ec2ea71
000003bf0008fc38422651c5c2eb
000003c00008fc3742261326c2
000003c30008fc34422496ff
Unknown Binary Messages
Atomic Detector
Pattern Detector
Boundary Detector
Candidate Field
Candidate Field
Candidate Field
Conflict Resolution and Unification Algorithm
Final Description
1
Run unknown messages against an ensemble of specialized detectors.
2
Integrate the results


Unknown Binary Messages
Atomic Detector
Pattern Detector
Boundary Detector
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Length
L
Float
Timestamp
Serialization Pattern

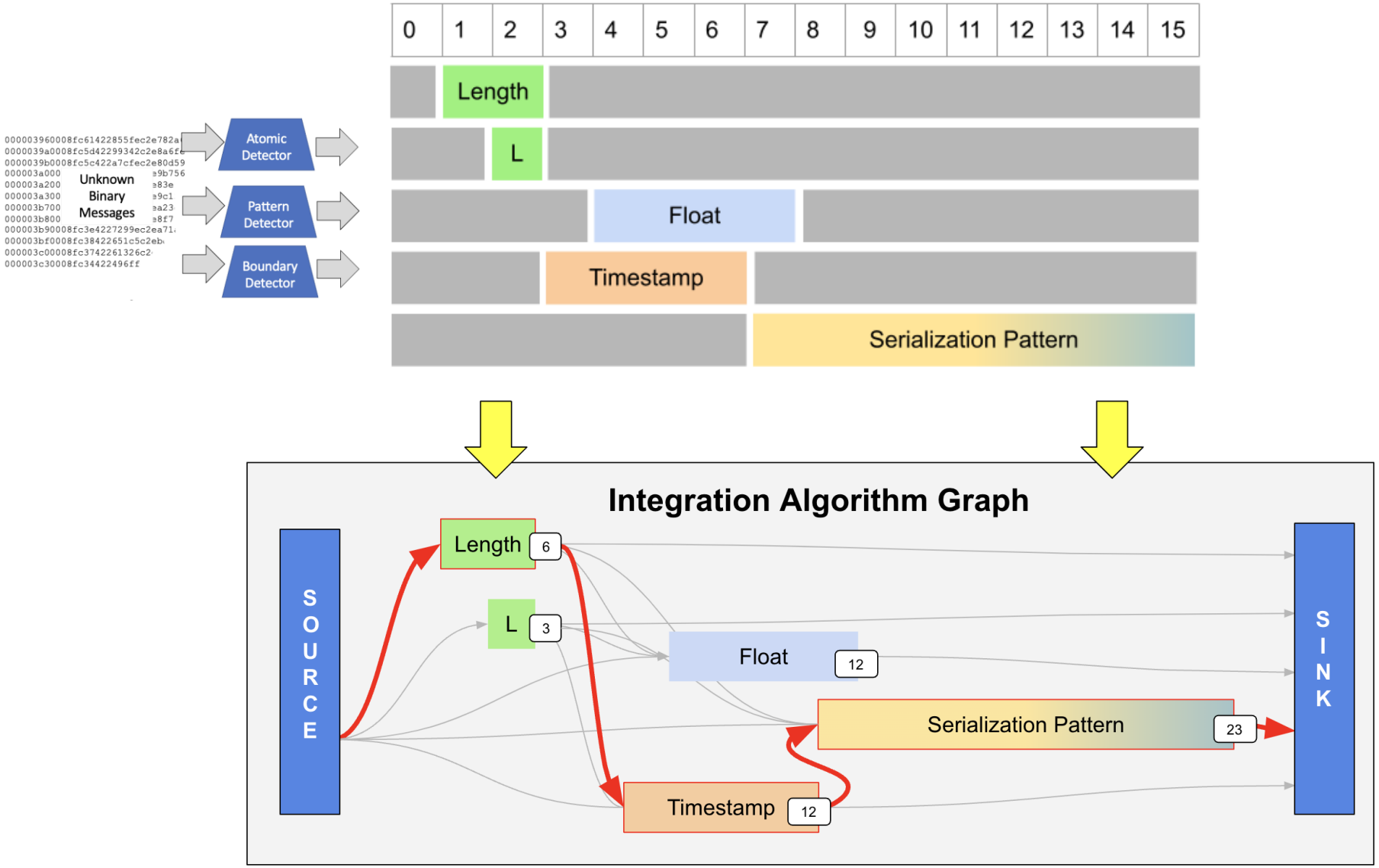

Unknown Binary Messages
Atomic Detector
Pattern Detector
Boundary Detector
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Length
L
Float
Timestamp
Serialization Pattern
Integration Algorithm Graph
SOURCE
Length 6
L 3
Float 12
Serialization Pattern 23
Timestamp 12
SINK


Unknown Binary Messages
Atomic Detector
Pattern Detector
Boundary Detector
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Length
L
Float
Timestamp
Serialization Pattern
Integration Algorithm Graph
SOURCE
Length 6
L 3
Float 12
Serialization Pattern 23
Timestamp 12
SINK

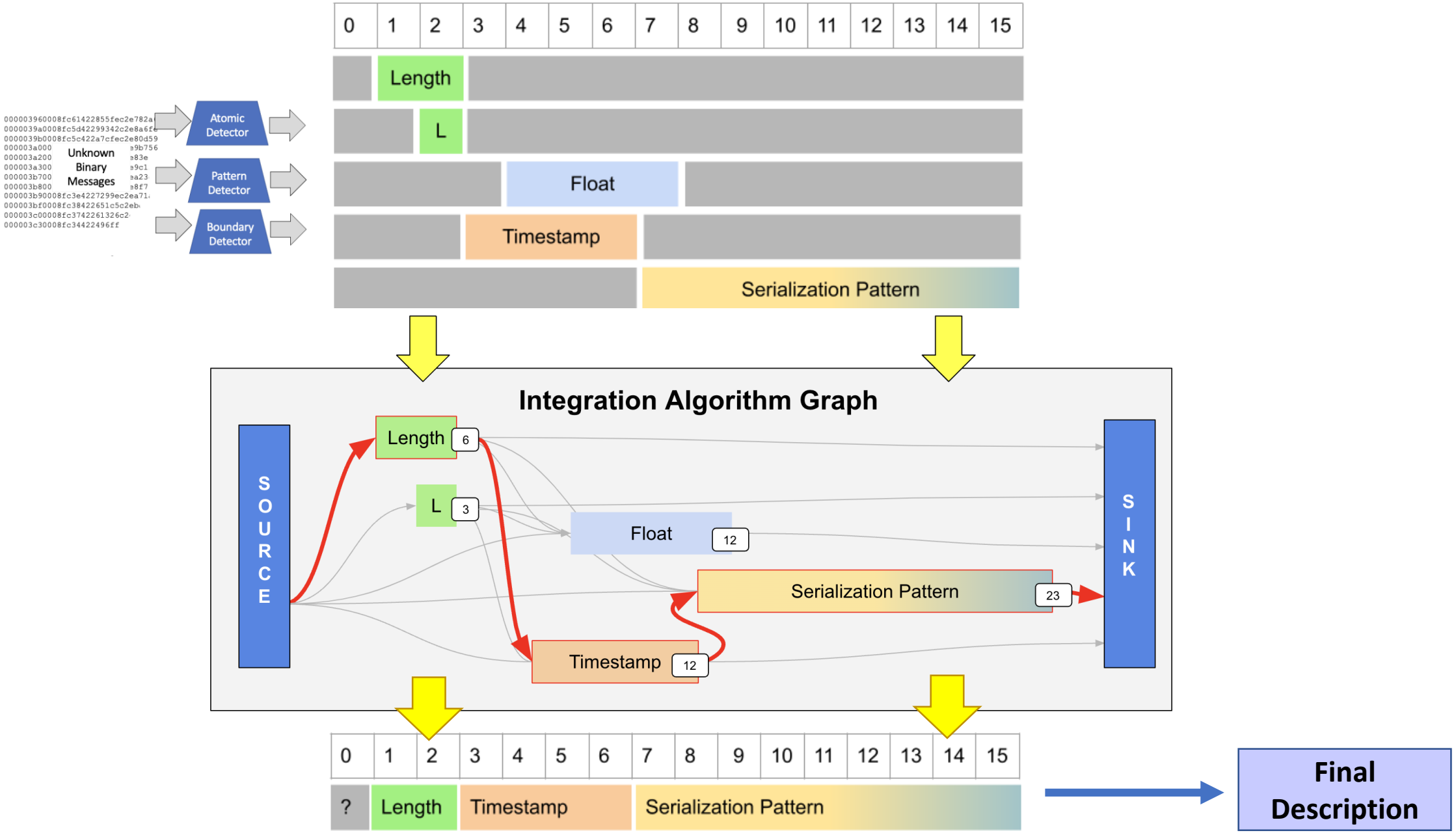

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Length
L
Float
Timestamp
Serialization Pattern
Unknown Binary Messages
Atomic Detector
Pattern Detector
Boundary Detector
Edge Weight: Sum of field size for every message.
2 bytes x 3 messages = 6
Integration Algorithm Graph
SOURCE
SINK
Length
6
L
3
Float
12
Serialization Pattern
23
Timestamp
12



Best Description = Describes the most data = Path with Maximum Edge Values



Best Description = Describes the most data = **Path with Maximum Edge Values = 6 + 12 + 23 = 41 bytes**

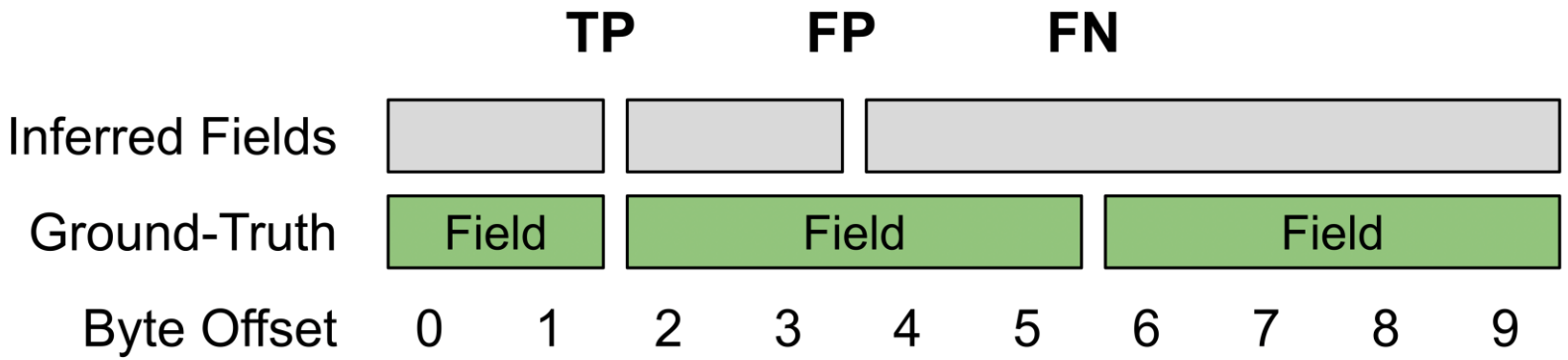

Unknown Binary Messages
Atomic Detector
Pattern Detector
Boundary Detector
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Length
L
Float
Timestamp
Serialization Pattern
Integration Algorithm Graph
SOURCE
Length 6
L 3
Float 12
Serialization Pattern 23
Timestamp 12
SINK
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
? Length Timestamp Serialization Pattern
Final Description

Evaluation: Measuring Performance

$$Precision = \frac{\text{Inferred True Field Boundaries (TP)}}{\text{Inferred Field Boundaries (TP + FP)}}$$

$$Recall = \frac{\text{Inferred True Field Boundaries (TP)}}{\text{True Field Boundaries (positives)}}$$

$$FPR = \frac{\text{Inferred False Field Boundaries (FP)}}{\text{Adjacent Field Bytes (negatives)}}$$



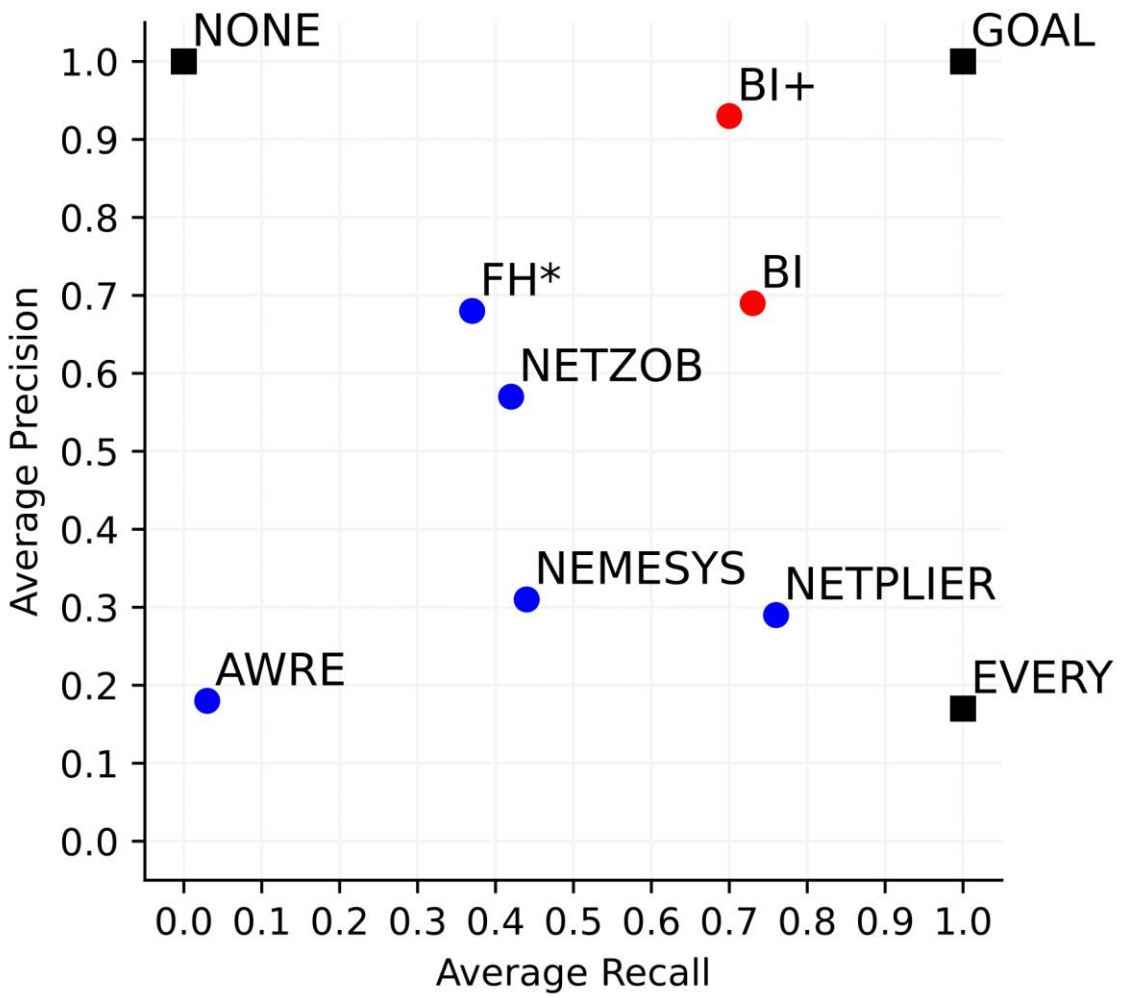
Evaluation

Protocol	Category
• bgp	Network
• dhcp	Network
• dnp3	Industrial Control
• mavlink	UAV/Drone
• mirai c2	Malware
• modbus	Industrial Control
• ntp	Network
• smb	Network
• smb2	Network
• tutorial	Teaching

Protocol Reverse Engineering Tools

- AWRE [WOOT 19]
- FIELDHUNTER [IFIP 15]
- NEMESYS [WOOT 18]
- NETPLIER [NDSS 21]
- NETZOB [SSTIC 12]

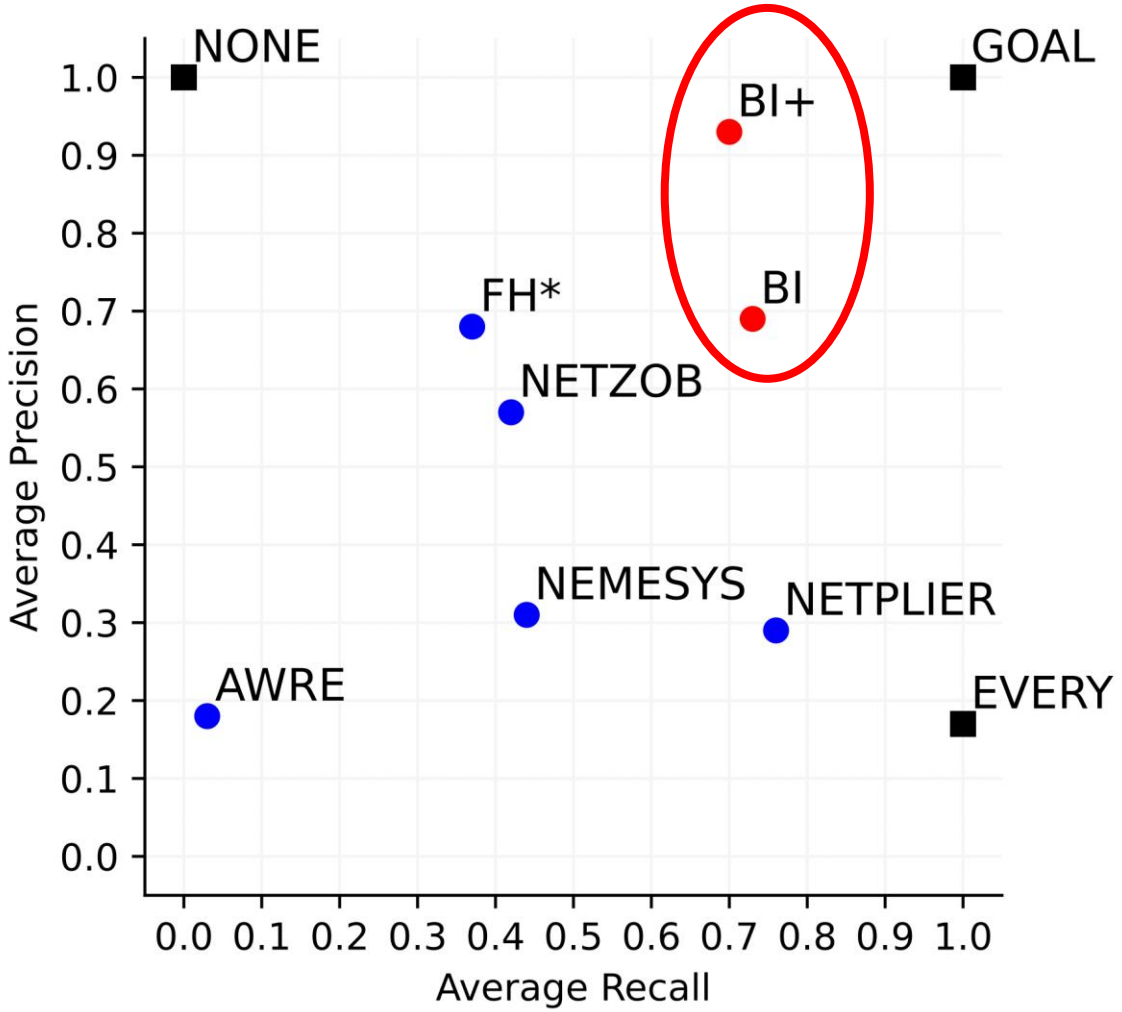
Average Performance on Top-Level Samples



Tool	# Msgs.	Prec.	Rec.	FPR	F1
BI+	1000	**0.93**	0.70	**0.005**	**0.78**
	500	**0.91**	0.69	**0.005**	**0.77**
	100	**0.82**	0.63	0.02	**0.70**
BI	1000	0.69	0.73	0.04	0.70
	500	0.69	0.72	0.04	0.69
	100	0.60	0.62	0.05	0.59
AWRE	1000	0.18	0.03	0.04	0.05
	500	0.18	0.03	0.04	0.05
	100	0.08	0.01	0.05	0.02
FIELDHUNTER*	1000	0.68	0.37	0.01	0.45
	500	0.68	0.37	0.01	0.45
	100	0.48	0.29	**0.01**	0.34
NEMESYS	1000	0.31	0.44	0.11	0.34
	500	0.31	0.45	0.11	0.34
	100	0.30	0.44	0.10	0.33
NETPLIER	1000	0.29	**0.75**	0.22	0.38
	500	0.27	**0.73**	0.22	0.37
	100	0.25	**0.83**	0.26	0.37
NETZOB	1000	0.57	0.42	0.03	0.47
	500	0.57	0.42	0.03	0.48
	100	0.53	0.45	0.04	0.45

* Results where FIELDHUNTER found no fields are excluded from averages.

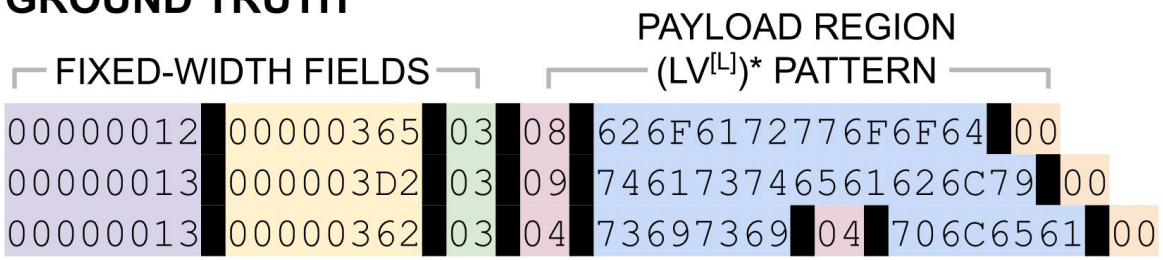
Average Performance on Top-Level Samples



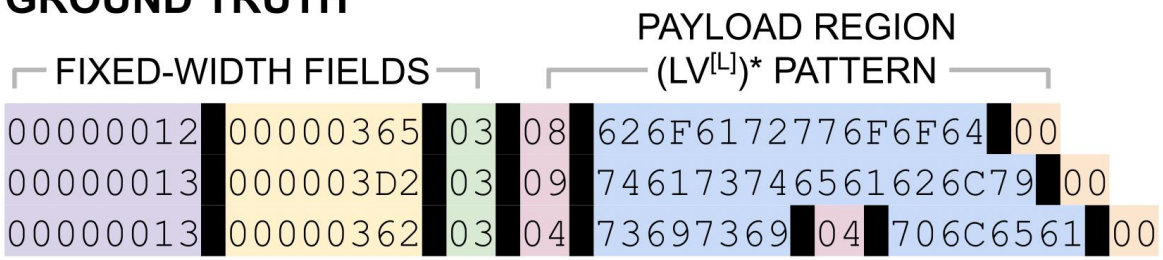
Tool	# Msgs.	Prec.	Rec.	FPR	F1
BI+	1000	**0.93**	0.70	**0.005**	**0.78**
	500	**0.91**	0.69	**0.005**	**0.77**
With Priors	100	**0.82**	0.63	0.02	**0.70**
BI	1000	0.69	0.73	0.04	0.70
	500	0.69	0.72	0.04	0.69
Naive	100	0.60	0.62	0.05	0.59
AWRE	1000	0.18	0.03	0.04	0.05
	500	0.18	0.03	0.04	0.05
	100	0.08	0.01	0.05	0.02
FIELDHUNTER*	1000	0.68	0.37	0.01	0.45
	500	0.68	0.37	0.01	0.45
	100	0.48	0.29	**0.01**	0.34
NEMESYS	1000	0.31	0.44	0.11	0.34
	500	0.31	0.45	0.11	0.34
	100	0.30	0.44	0.10	0.33
NETPLIER	1000	0.29	**0.75**	0.22	0.38
	500	0.27	**0.73**	0.22	0.37
	100	0.25	**0.83**	0.26	0.37
NETZOB	1000	0.57	0.42	0.03	0.47
	500	0.57	0.42	0.03	0.48
	100	0.53	0.45	0.04	0.45

* Results where FIELDHUNTER found no fields are excluded from averages.

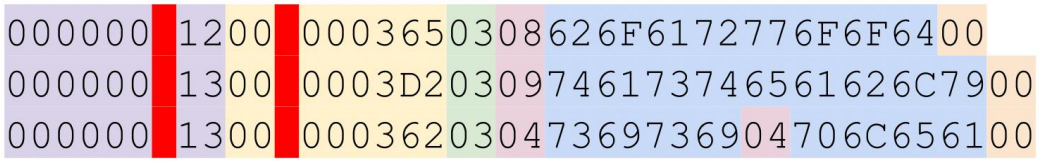
GROUND TRUTH



GROUND TRUTH

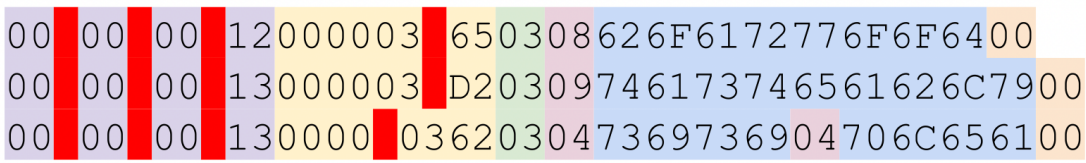


AWRE

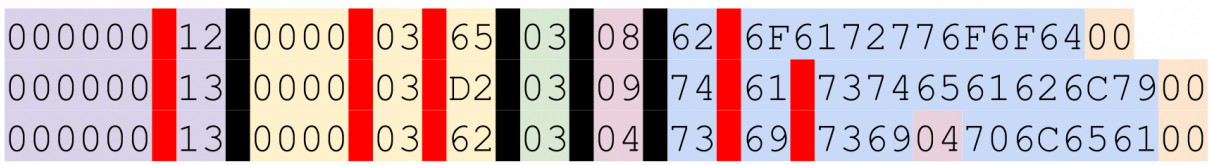


FIELDHUNTER

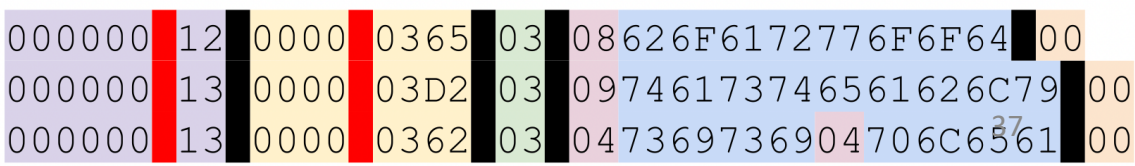
NEMESYS



NETPLIER



NETZOB



GROUND TRUTH

PAYLOAD REGION

FIXED-WIDTH FIELDS — (LV[L])* PATTERN

```
00000012 00000365 03 08 626F6172776F6F64 00
00000013 000003D2 03 09 7461737465626C79 00
00000013 00000362 03 04 73697369 04 706C6561 00
```

BI+

```
00000012 00000365 03 08 626F6172776F6F64 00
00000013 000003D2 03 09 7461737465626C79 00
00000013 00000362 03 04 73697369 04 706C6561 00
```

BI

```
00000012 0000 03 65 03 08 626F6172776F6F64 00
00000013 0000 03 D2 03 09 7461737465626C79 00
00000013 0000 03 62 03 04 73697369 04 706C6561 00
```

AWRE

```
000000 1200 000365030862 6F6172776F6F6400
000000 1300 0003D203097461737465626C7900
000000 1300 00036203047369736904706C656100
```

FIELDHUNTER

```
00000012 00000365 0308626F6172776F6F6400
00000013 000003D2 0309746173746561626C7900
00000013 00000362 0304736973690470 6C656100
```

NEMESYS

```
00 00 00 12000003 650308626F6172776F6F6400
00 00 00 13000003 D20309746173746561626C7900
00 00 00 130000 03620304736973690470 6C656100
```

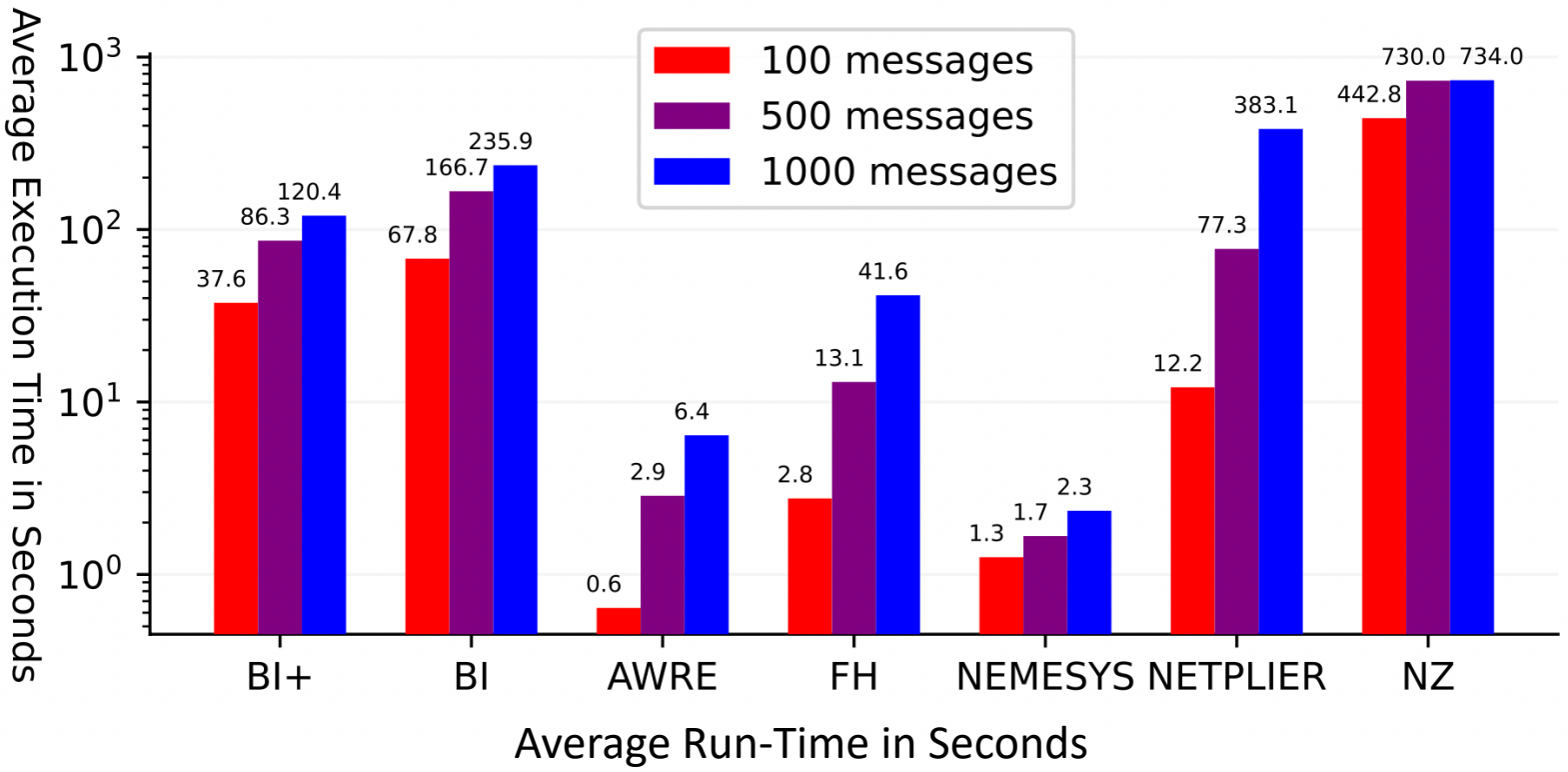
NETPLIER

```
000000 12 0000 03 65 03 08 62 6F6172776F6F6400
000000 13 0000 03 D2 03 09 74 61 73746561626C7900
000000 13 0000 03 62 03 04 73 69 736904706C656100
```

NETZOB

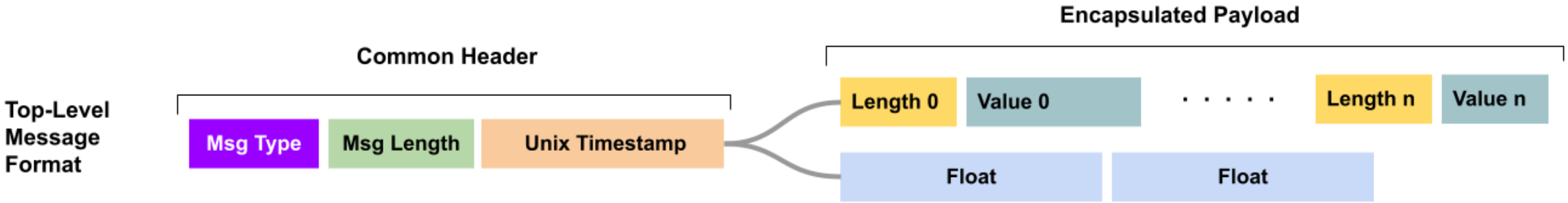
```
000000 12 0000 0365 03 08626F6172776F6F64 00
000000 13 0000 03D2 03 0974617374656 1626C79 00
000000 13 0000 0362 03 04736973690470 6C6561 00
```

Average Execution Times



Limitations & Future Work

- At Present BinaryInferno only infers fields or serialization patterns common to every message.
- We have ongoing work to handle unions over payload sub-fomats



- Add New Detectors to our Ensemble
- Expanding our Serialization Pattern Search Grammar

Conclusion

BinaryInferno

- Automatically reverse engineer binary message formats from network traces.
- Uses an ensemble of specialized detectors.
- Graph-based approach to integrate results.
- Significantly improves on existing approaches.

Open-Source: April 2023 **BinaryInferno.net**

Jared Chandler

PhD Candidate
Tufts University
Department of Computer Science

jared.chandler@tufts.edu

Acknowledgements: This material is based upon work partly supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-19-C-0073. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Distribution Statement A: Approved for public release. Distribution is unlimited.