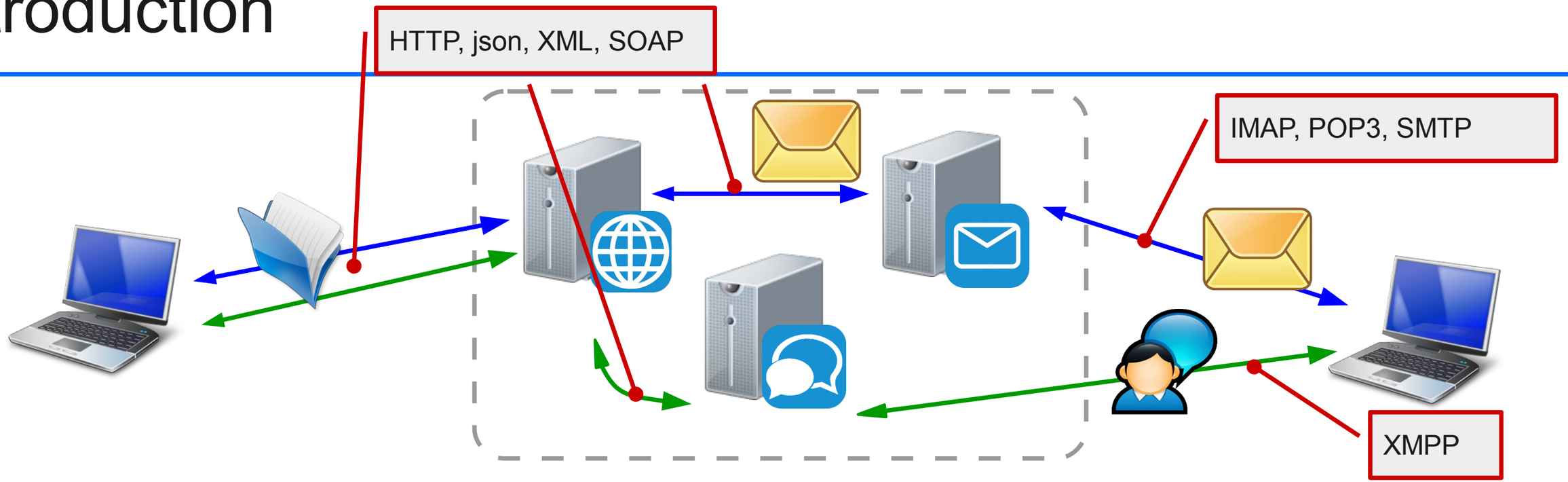
In the Compression Hornet's Nest: A Security Study of Data Compression in Network Services

Giancarlo Pellegrino[(1)], Davide Balzarotti[(2)], Stefan Winter[(3)], and Neeraj Suri[(3)]

24th USENIX Security Symposium, Washington, D.C.

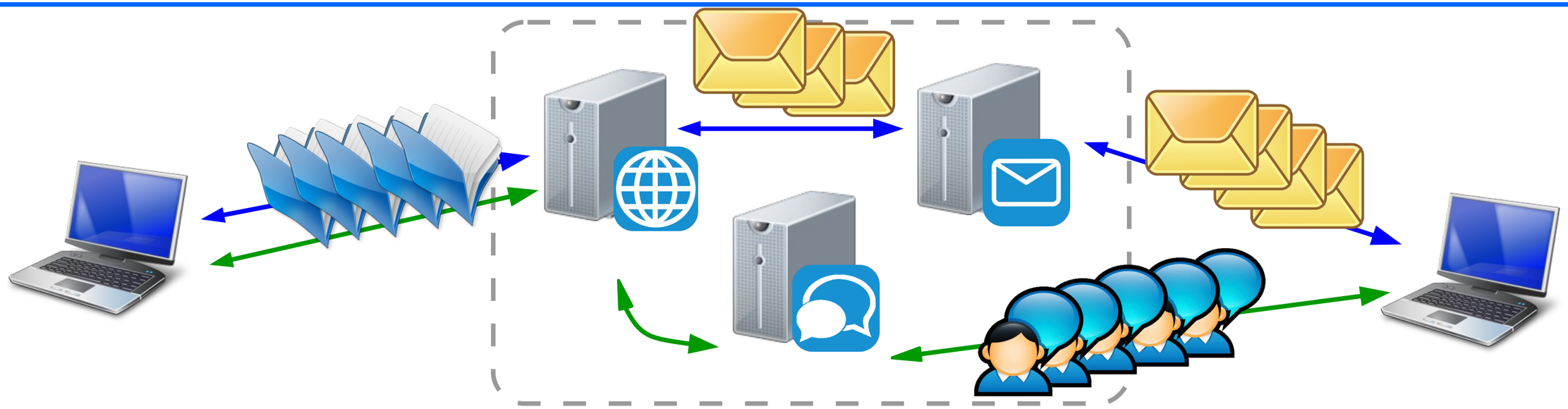
[(1)]Saarland University, Germany
[(2)]EURECOM, France
[(3)]TU Darmstadt, Germany

Introduction



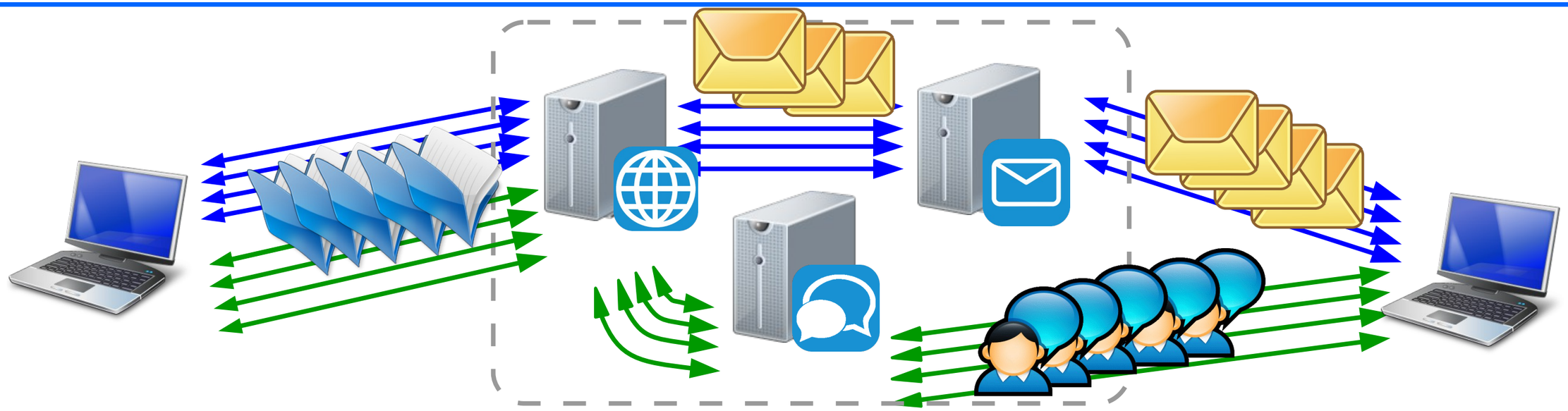
- Modern applications rely on (core) network services, e.g., Web, email, and IM services

Introduction



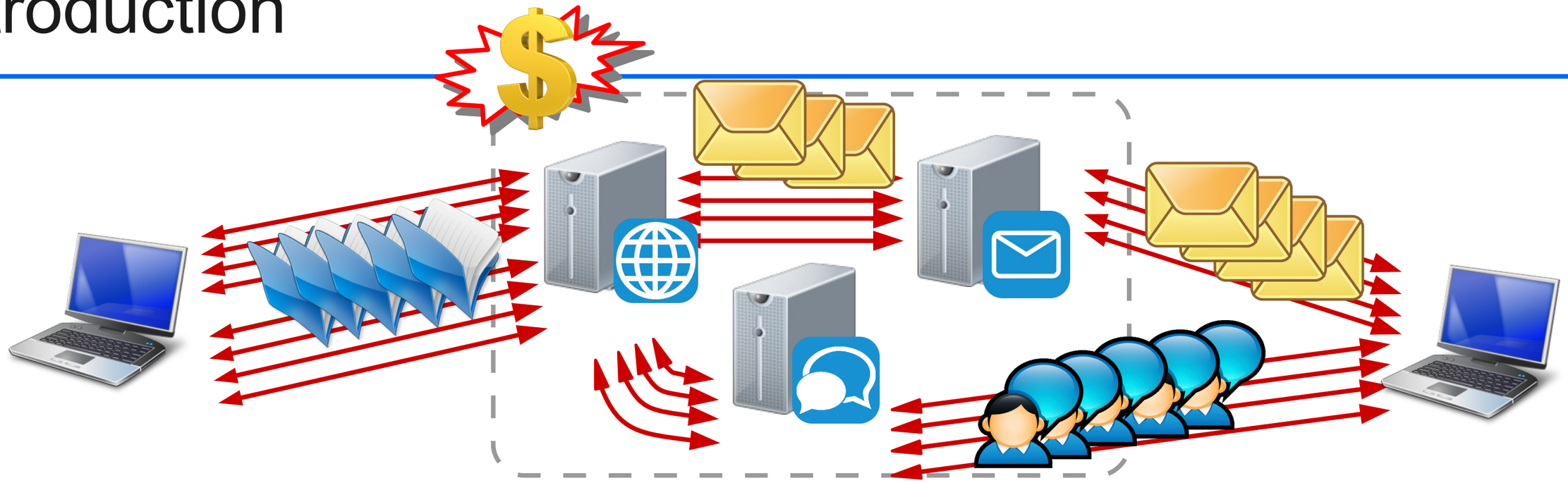
- Modern applications rely on (core) network services, e.g., web, email, and IM services
- Amount of exchanged data continues to increase steadily
  - More data → more transfer time → unresponsiveness → user unhappiness

Introduction



- Modern applications rely on (core) network services, e.g., web, email, and IM services
- Amount of exchanged data continues to increase steadily
  - More data → more transfer time → unresponsiveness → user unhappiness
- A way to solve it is to buy more bandwidth

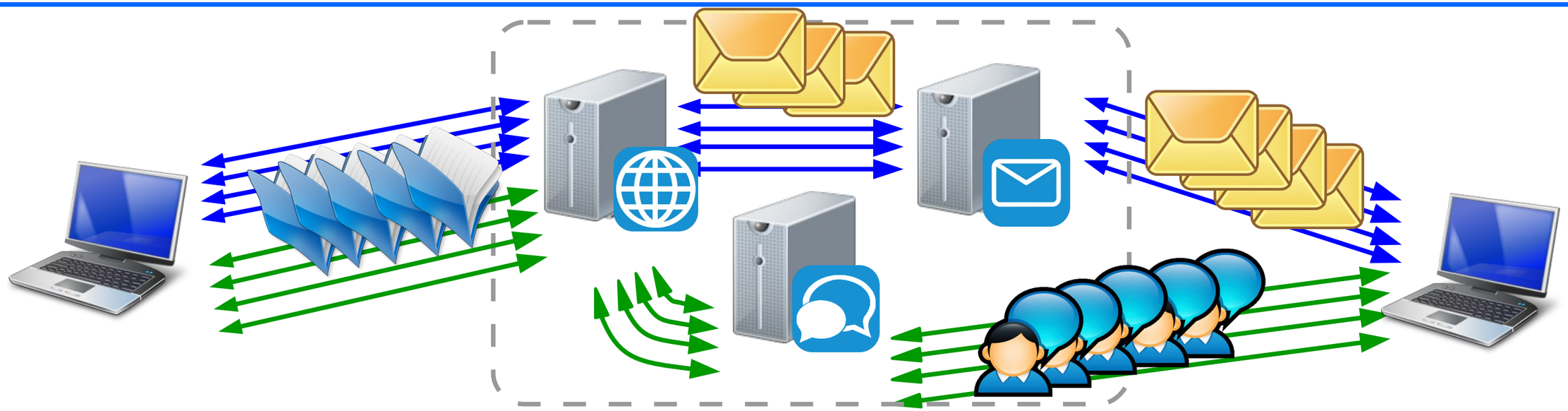
Introduction



- Modern applications rely on (core) network services, e.g., web, email, and IM services
- Amount of exchanged data continues to increase steadily
  - More data → more transfer time → unresponsiveness → user unhappiness
- A way to solve it is to buy more bandwidth

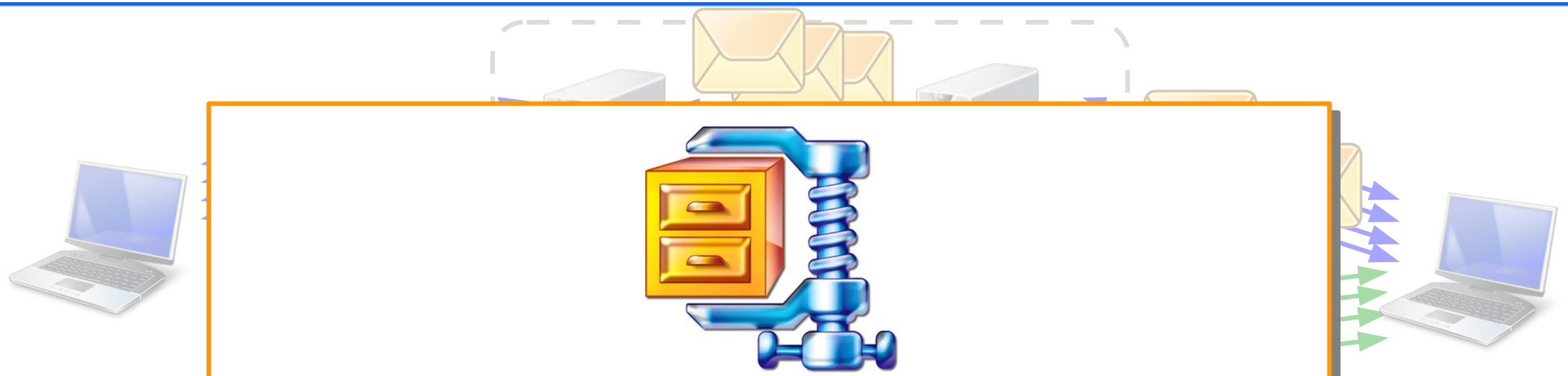
  ➔ However, bandwidth costs

Introduction



- Modern applications rely on (core) network services, e.g., web, email, and IM services
- Amount of exchanged data continues to increase steadily
  - More data → more transfer time → unresponsiveness → user unhappiness
- A way to solve it is to buy more bandwidth
  - ➔ However, bandwidth costs
- Another solution is ...

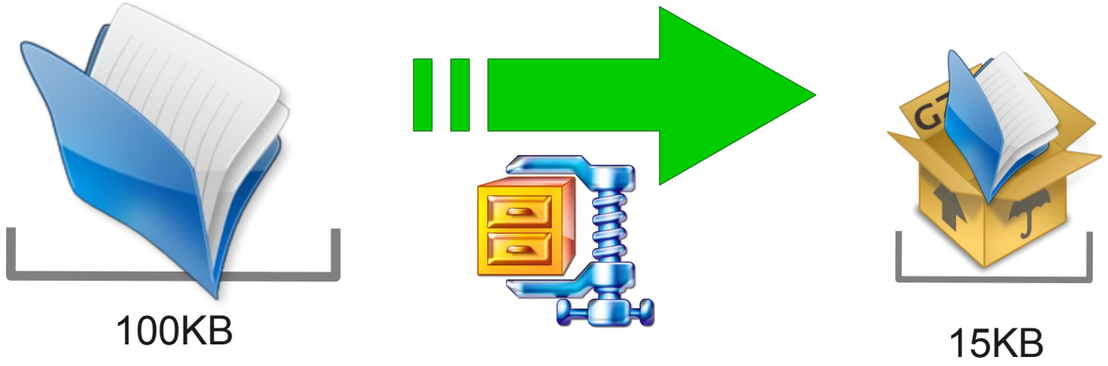
Introduction



Data compression!

- Modern appl … services
- Amount of ex…
  - More data → more transfer time → unresponsiveness → user unhappiness
- A way to solve it is to buy more bandwidth
  - ➔ However, bandwidth costs
- Another solution is ...

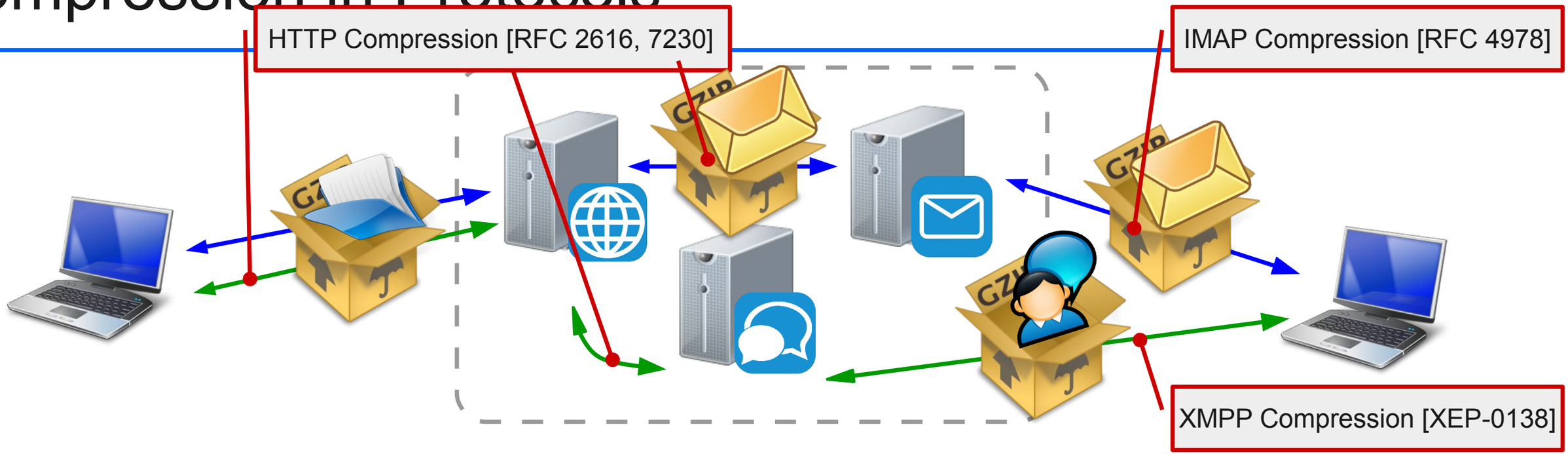
Data Compression



- Reduces # of bits of a string by removing redundancy
  - *lossless* if $decompr(compr(d)) = d$ or *lossy* if $decompr(compr(d)) \sim= d$
- Lots of algorithms (See [1])
- Among the most popular: **Deflate** [RFC 1951]
  - Implemented in libraries, e.g., `zlib`, or as a tool, e.g., `gzip`, and `zip` archive tool
  - Available in most of the programming languages

[1] SALOMON, D. Data Compression: The Complete Reference. Springer-Verlang, 2007.

Compression in Protocols



- Data compression is used by network protocols to reduce message size
- Mandated by protocol specifications
  - e.g., HTTP (response) compression, IMAP, XMPP, SSH, PPP, and others
- Or implemented as custom feature
  - e.g., HTTP request compression

The Problem of Data Compression

- If not properly implemented, it can make application vulnerable to DoS

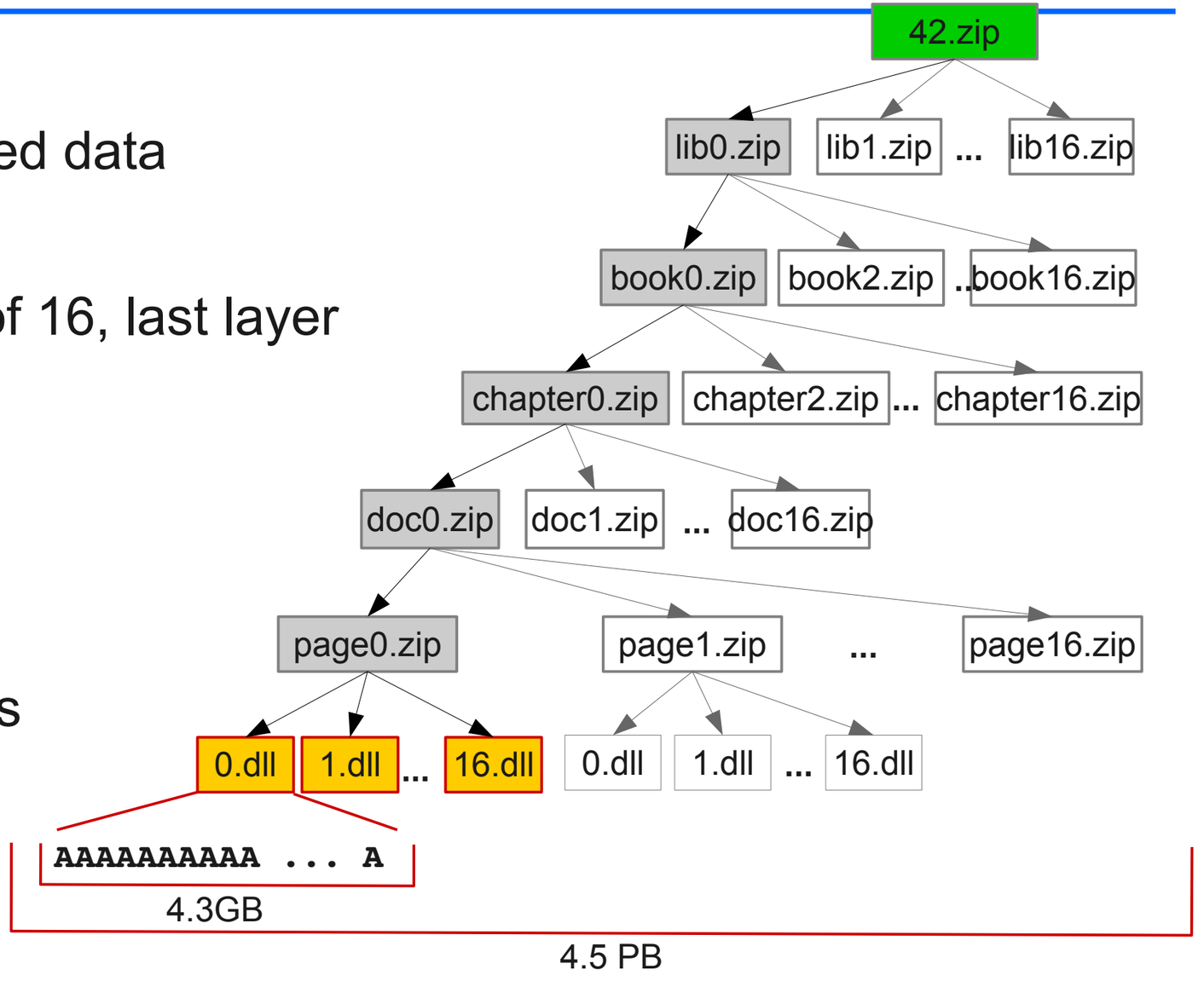
- Risks:

  1) **Intensive task**
     - Computationally intensive
     - If abused, it can stall an application
  2) **Data Amplification**
     - Decompression increases the data to be processed (compression rate of zlib ~1:1024)
     - Internal components may not be designed to handle high volume of data
  3) **Unbalanced Client-Server Scenario**
     - Clients pre-compute compressed messages
     - Server decompresses msgs each time

- Popular examples from the past...

The Past: Zip Bombs (1996)

- 42 KB zip file → **4.5 PB** uncompressed data
- 5 layers of nested zip files in blocks of 16, last layer with text files of 4.3 GB each
- Cause Disk/Memory exhaustion
- Sent as attachment to crash anti-virus software



The Past: Billion Laughs (2003)

- Resource exhaustion in `libxml2` when processing nested XML entity definitions

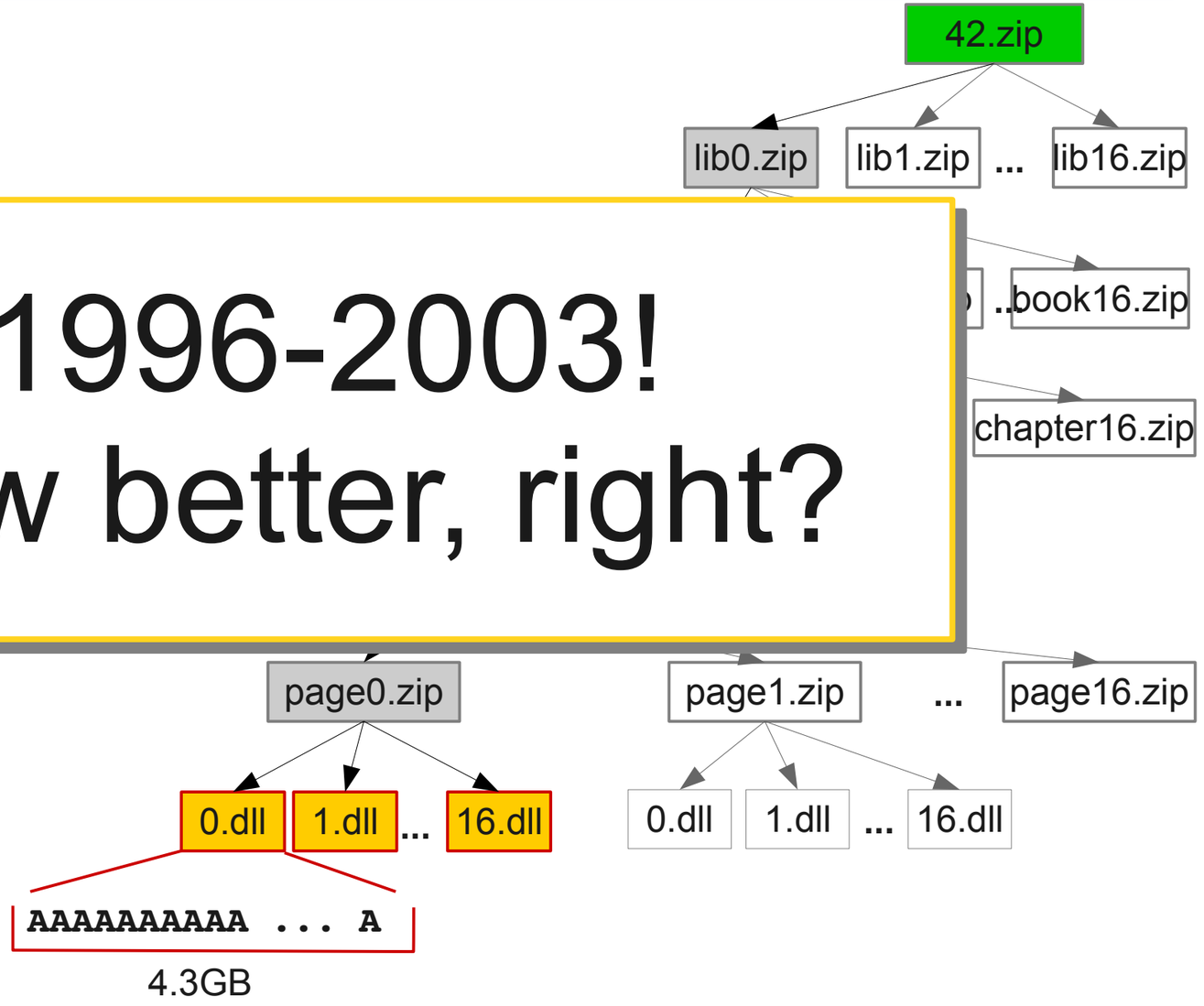
```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

- 810 bytes of XML document expanded to **3GB**

The Past: Zip Bombs and Billion Laughs

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lo
 <!ELEMENT l
 <!ENTITY lo
 <!ENTITY lo
 <!ENTITY lo
 <!ENTITY lo
 <!ENTITY lo
 <!ENTITY lo
 <!ENTITY lo
 <!ENTITY lo
 <!ENTITY lo
]>
<lolz>&lol9;</lolz>
```

This was 1996-2003!
Now we know better, right?



The Present

- Reviewed protocol specs, design patterns, and coding rules

  Unawareness of the risks, **guidelines** on handling data compression are **missing** or **misleading**

The Present

- Reviewed protocol specs, design patterns, and coding rules

  Unawareness of the risks, **guidelines** on handling data compression are **missing** or **misleading**

1. Protocol specifications:
   - ➔ No data compression handling issues, redirects to SSL/TLS (concerned with leakage and packet limits, but unexplained how they apply to other protocols)

The Present

- Reviewed protocol specs, design patterns, and coding rules

  Unawareness of the risks, **guidelines** on handling data compression are **missing** or **misleading**

1. Protocol specifications:
   - ➔ No data compression handling issues, redirects to SSL/TLS (concerned with leakage and packet limits, but unexplained how they apply to other protocols)

2. Secure Design Patterns:
   - Patterns to solve vulns. during design phase : *DoS Safety*, *Compartmentalization*, and *Small Process*
   - ➔ However, lack of the details to address implementation-level concerns

The Present

- Reviewed protocol specs, design patterns, and coding rules

  Unawareness of the risks, **guidelines** on handling data compression are **missing** or **misleading**

1. Protocol specifications:
   - ➔ No data compression handling issues, redirects to SSL/TLS (concerned with leakage and packet limits, but unexplained how they apply to other protocols)

2. Secure Design Patterns:
   - Patterns to solve vulns. during design phase : *DoS Safety*, *Compartmentalization*, and *Small Process*
   - ➔ However, lack of the details to address implementation-level concerns

3. Secure Coding Rules
   - Only one, i.e., Anti-Zip Bomb coding rule
   - ➔ Sadly, incorrect

The Present

- Reviewed protocol specs, design patterns, and coding rules

Unawareness of the risks, **guidelines** on handling data compression are **missing** or **misleading**

How does this lack of common knowledge and

understanding affect implementations?

1. Protoc
   - ➔ No c … limits, but unex
2. Secur
   - Patt … *cess*
   - ➔ However, lack of the details to address implementation-level concerns
3. Secure Coding Rules
   - Only one, i.e., Anti-Zip Bomb coding rule
   - ➔ Sadly, incorrect

Our contribution

1. Analyzed network service, extensions, protocol specifications, and documentations looking for proper or incorrect ways to handle data compression
   - ➔ Grouped findings in 12 pitfalls

2. Tested network services against compression bombs
   - ➔ Discovered 10 previously unknown vulnerabilities

Contents

- Mistakes in software
- Testing for resource exhaustion vulnerabilities

Mistakes in Software

Case Studies

Protocol	Network Service
XMPP	OpenFire, Prosody. Jabberd2, ejabberd, Tigase
HTTP	Apache HTTPD *+ mod_deflate + mod-php + CSJRPC + mod-gsoap + mod-dav*
	Apache Tomcat *+ 2Way/Webutilities + Apache CXF + (lib-)json-rpc + jsonrpc4j + Axis2*
	Axis 2 standalone
	gSOAP standalone
IMAP	Dovecot, Cyrus

- 11 popular services with 10 extensions
  - Selected via *service detection* of top 1000 of AlexaDB and of public IM services
- Analyzed specifications, documentation, and source code
- Observed 12 pitfalls...

Pitfalls

1. Implementation

2. Specification

3. Configuration

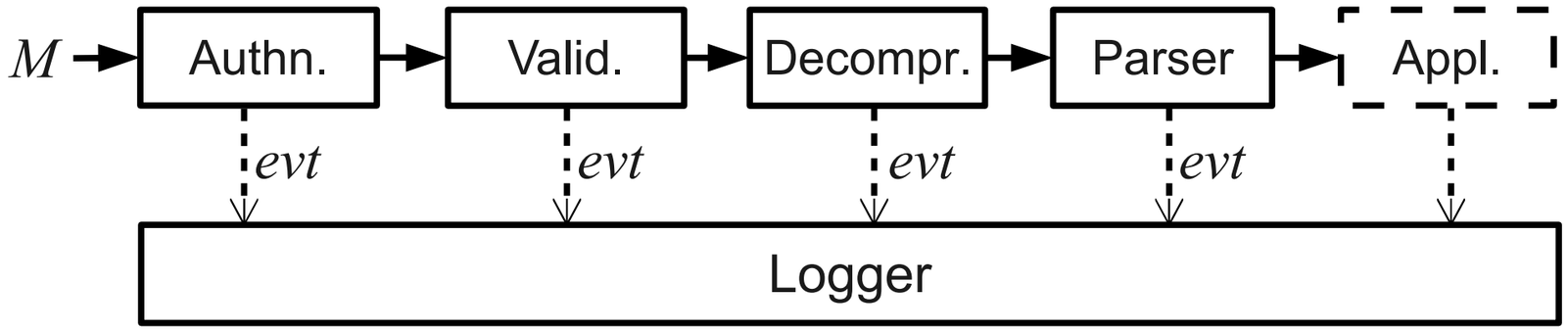
Pitfalls

1. Implementation
   - Use of Compression before Authentication
   - Improper Input Validation during Decompression
   - Logging Decompressed Messages
   - Improper Inter-Units Communication
   - Unbounded Resource Usage (CPU and Memory)
2. Specification
   - Erroneous Best Practice
   - Misleading Documentation
   - API Specs Inconsistency
3. Configuration
   - Insufficient Configuration Options
   - Insecure Default Values
   - Decentralized Configuration Parameters

Pitfalls

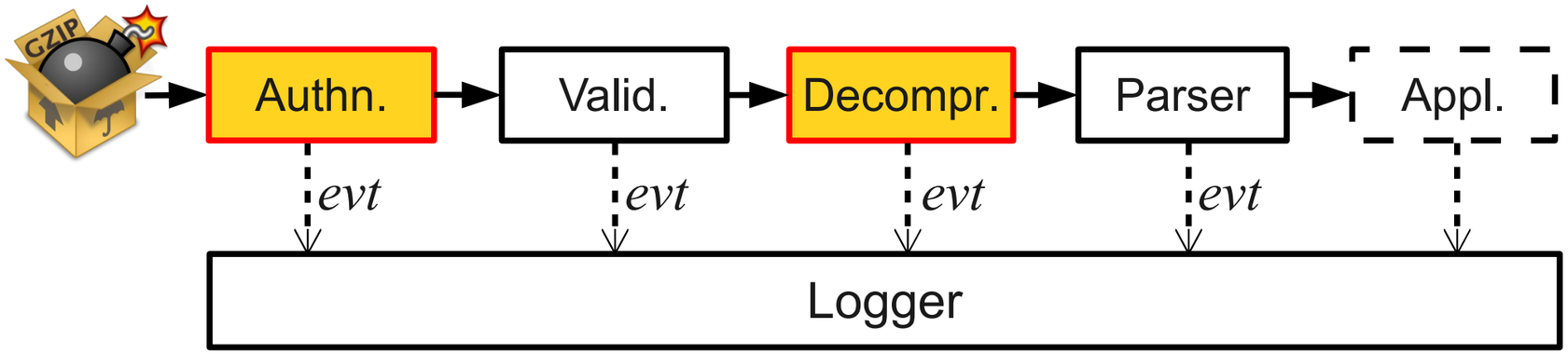
1. Implementation
   - Use of Compression before Authentication
   - Improper Input Validation during Decompression
   - Logging Decompressed Messages
   - Improper Inter-Units Communication
   - Unbounded Resource Usage (CPU and Memory)
2. Specification
   - Erroneous Best Practice
   - Misleading Documentation
   - API Specs Inconsistency
3. Configuration
   - Insufficient Configuration Options
   - Insecure Default Values
   - Decentralized Configuration Parameters

Pitfalls at Implementation level



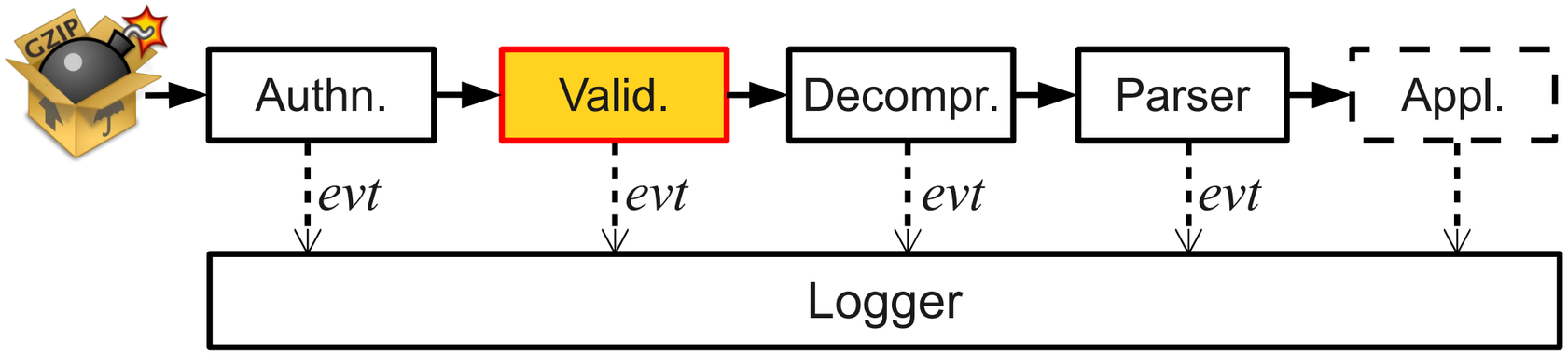
- Abstract message processing pipeline extracted from our case studies

Compression before Authentication



- Inconsistent best practice
  - Mandatory in SSL/TLS, recommended in XMPP, and undefined in IMAP and HTTP
  - Implementation may diverge from the specs, i.e., OpenSSH
- Developers may underestimate the risk or overlook recommendations
- Prosody accepted compressed messages before user authentication CVE-2014-2744
  - ➔ DoS by unauthenticated attackers

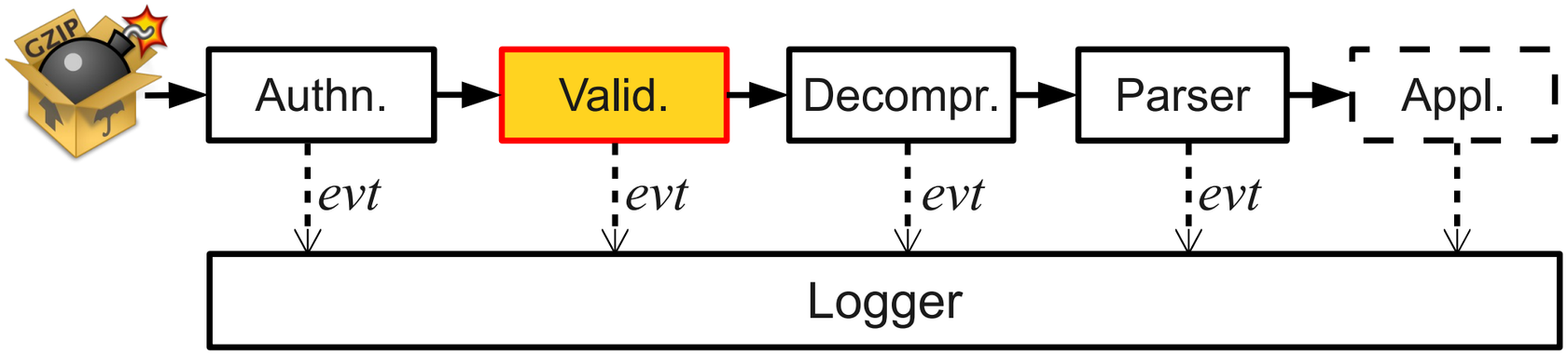
Improper Input Validation during Decompression



- 3 ways to validate a message:
  - Compressed message size
    - mod-deflate: If (compr. size > `LimitRequestBody`) → Reject CVE-2014-0118
    - ➔ However, hard to assess message size from its compressed form (1 MB compr → 1 GB decompr.)

mistake

Improper Input Validation during Decompression

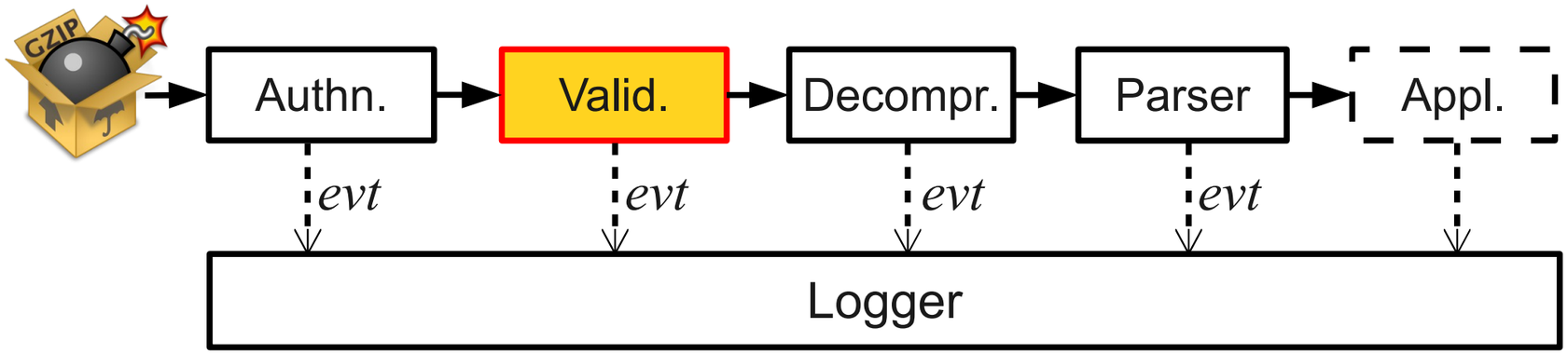


- 3 ways to validate a message:

mistake
- Compressed message size
  - mod-deflate: If (compr. size > `LimitRequestBody`) → Reject CVE-2014-0118
  - ➔ However, hard to assess message size from its compressed form (1 MB compr → 1 GB decompr.)

risky
- Decompression ratio
  - Patched mod-deflate: if (decompr ratio > threshold) → Reject
  - ➔ Problem of ratio selection

Improper Input Validation during Decompression



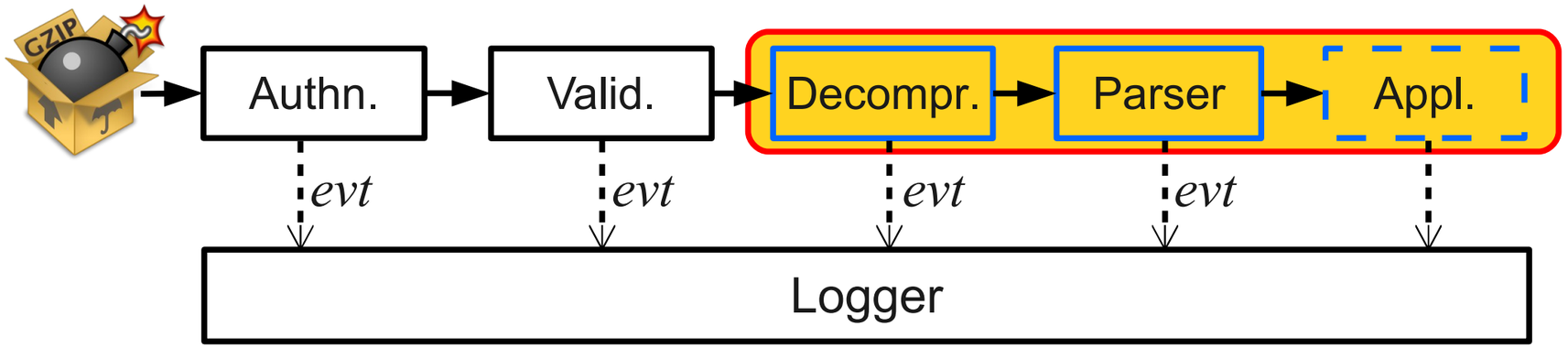
- 3 ways to validate a message:

mistake
- Compressed message size
  - mod-deflate: If (compr. size > `LimitRequestBody`) → Reject CVE-2014-0118
  - ➔ However, hard to assess message size from its compressed form (1 MB compr → 1 GB decompr.)

risky
- Decompression ratio
  - Patched mod-deflate: if (decompr ratio > threshold) → Reject
  - ➔ Problem of ratio selection

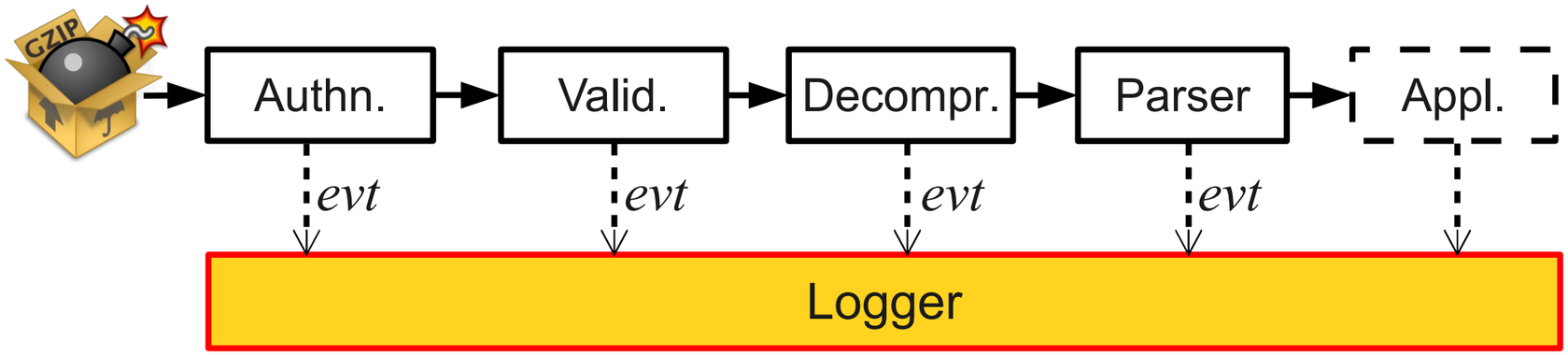
correct
- Decompressed message size
  - mod-deflate + mod-dav: If (decompr. size > `LimitXMLRequestBody`) → Reject

Improper Inter-Units Communication



- Upon exception, the pipeline halts and rejects message
- mod-php and mod-gsoap limit the size of incoming (decompressed) message
- … but had no means to halt mod-deflate
  - ➔ mod-deflate keeps on decompressing data
    - Problem addressed in CVE-2014-0118

Logging Decompressed Messages



- Frequency and verbosity of log events can cause DoS
- If exception is caused by compressed data, the needed resources may be underestimated
- Upon invalid requests, Apache CXF logs first 100KB of incoming message
  - However, first it decompresses the entire message on a file, then logs the first 100KB
  - ➔ DoS due to disk space exhaustion CVE-2014-0109/ -0110

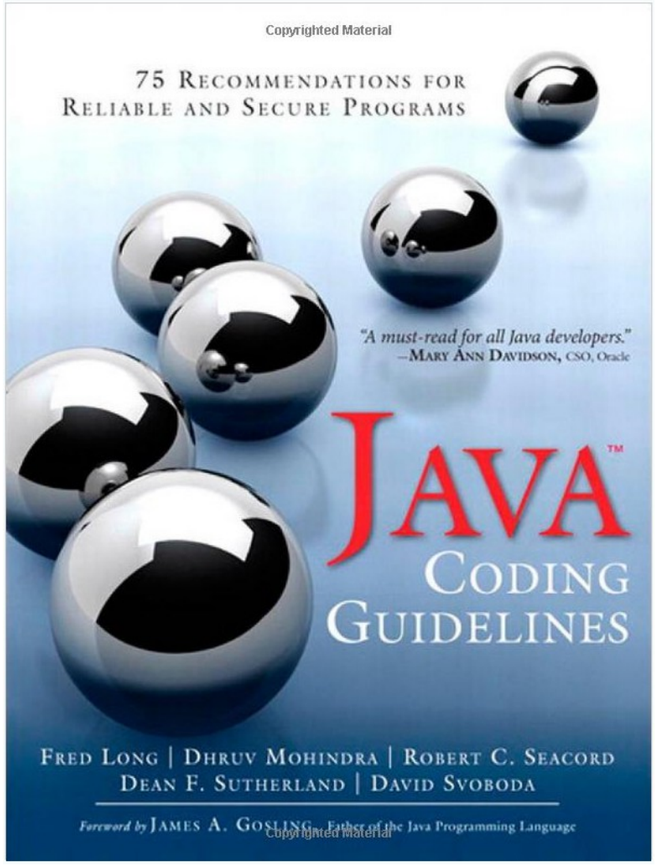
Erroneous Best Practices (Spec. level)

- Only one code pattern specific for data compression
  - Rule: “IDS04-J. Safely extract files from ZipInputStream”

```
// Write the files to the disk, but
// only if the file is not insanely big
if (zipfile.getSize() > TOOBIG ) {
    throw new IllegalStateException("File to be unzipped is huge.");
}
```

- `.getSize()` returns ZIP file header with uncompressed size

- but ZIP headers can be forged
  - ➔ DoS countermeasure bypass Notif. Authors

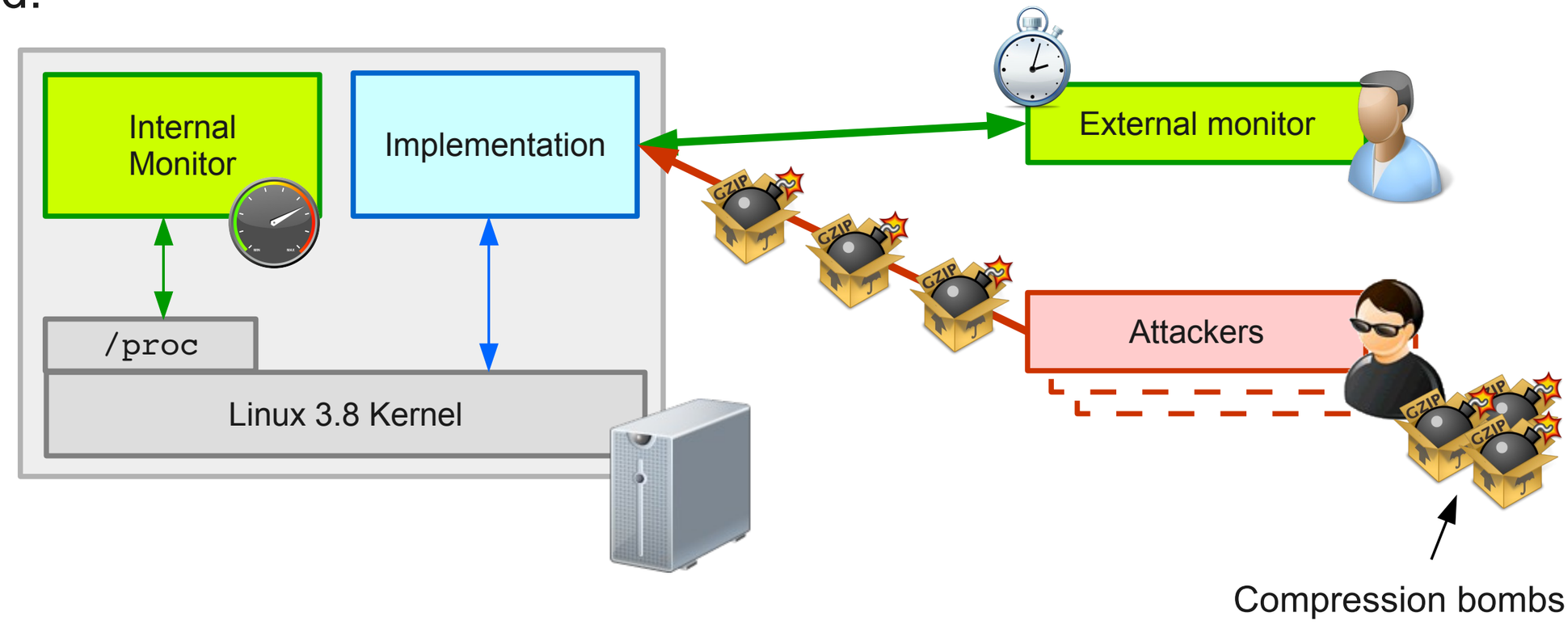


Resource exhaustion vulnerabilities

Experiments

- Case studies on local servers

- Testbed:



HTTP Compression Bomb (SOAP)

- Case studies on local servers
- Testbed:

```
POST /index.html HTTP/1.1
Content-Encoding: gzip
\r\n
<soapenv:Envelope>
```

compressed

4 GB of white spaces

```
<soapenv:Body>[...]</soapenv:Body>
</soapenv:Envelope>
\r\n
```

Compression bombs

~4 MB, ~1:1000 compr. ratio

Zip Bombs Everywhere

Protocol	Network Service	
XMPP	OpenFire	CVE-2014-2741
	Prosody	CVE-2014-2744/ -2745
	Tigase	CVE-2014-2746
	Ejabberd, jabberd2	
HTTP	Apache HTTPD + mod_deflate	CVE-2014-0118
	+ mod-php, CSJRPC, mod-gsoap, mod-dav	
	Apache Tomcat + 2Way/Webutilities filter	Notif. devel
	+ Apache CXF	CVE-2014-0109/ -0110
	+ json-rpc, lib-json-rpc	Notif. devels
	+ Axis2/ +jsonrpc4j	
	Axis 2 standalone	
	gSOAP standalone	Notif. devel
IMAP	Dovecot, Cyrus	

Conclusion

Conclusion/Takeaway

- ~20 years after the zip bombs, developers still unaware of the risks of handling data compression
- Discovered 10 previously-unknown vulns. in popular network services
- Presented 12 pitfalls which can be used by developers to build more secure services