SCHIZO PHRENIC FILES

Ange Albertini

2014/09/05

This talk is a collaboration with:

Gynvael Coldwind

Security researcher, Google

Captain

Likes hamburgers

http://gynvael.coldwind.pl/

ANGE ALBERTINI

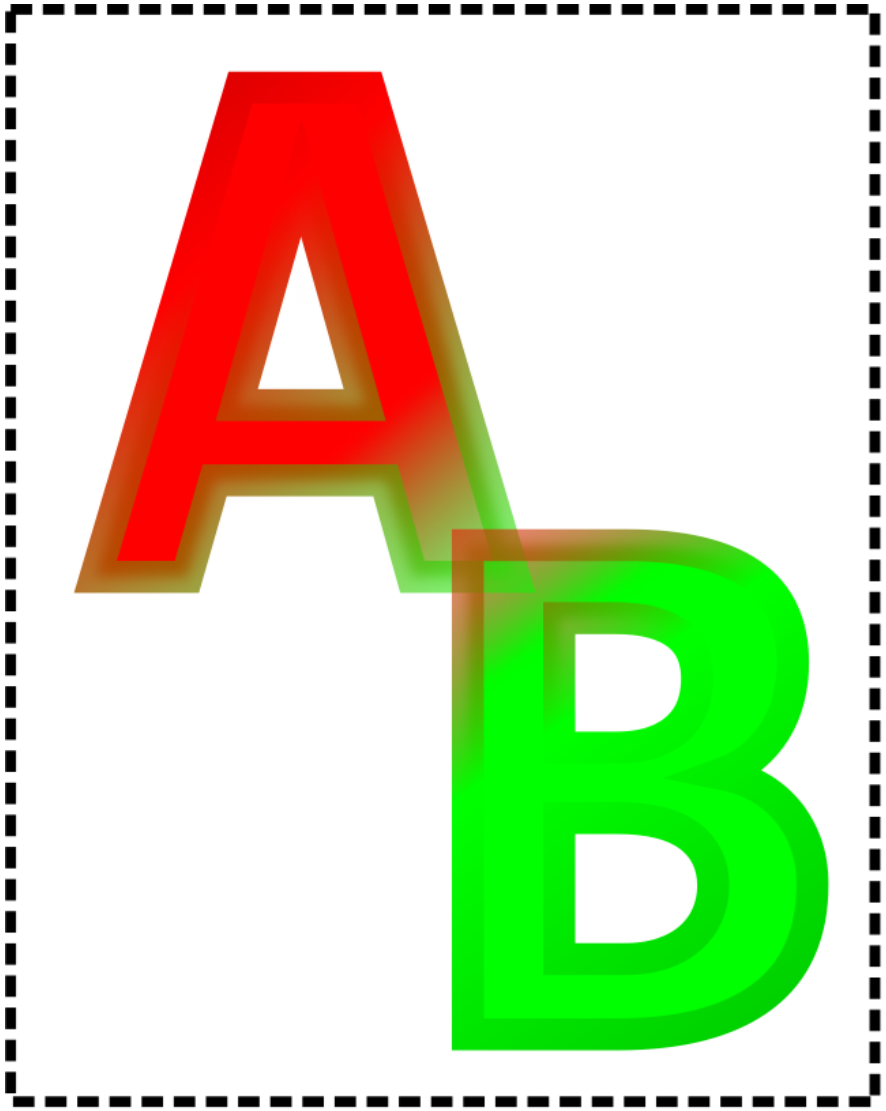
reverse engineering

VISUAL DOCUMENTATIONS

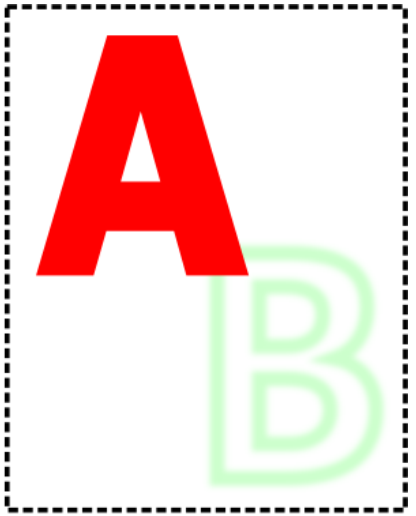
@angealbertini

ange@corkami.com

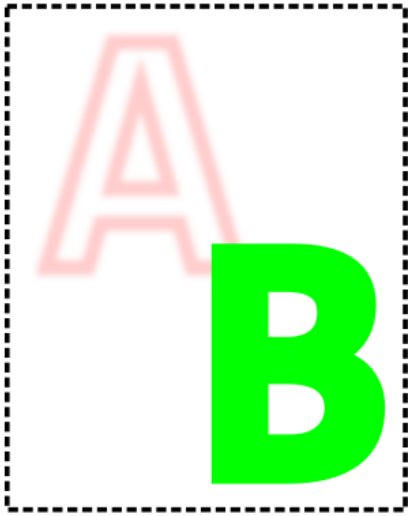
http://www.corkami.com



FILE



Prg 1



Prg 2

1 file, 2 programs
⇒ 2 different contents

No active detection of the program in the file

Fooling,
not failing

Both programs will load the file correctly:
No reported warning or error, no exploitation.

Abusing parsers for

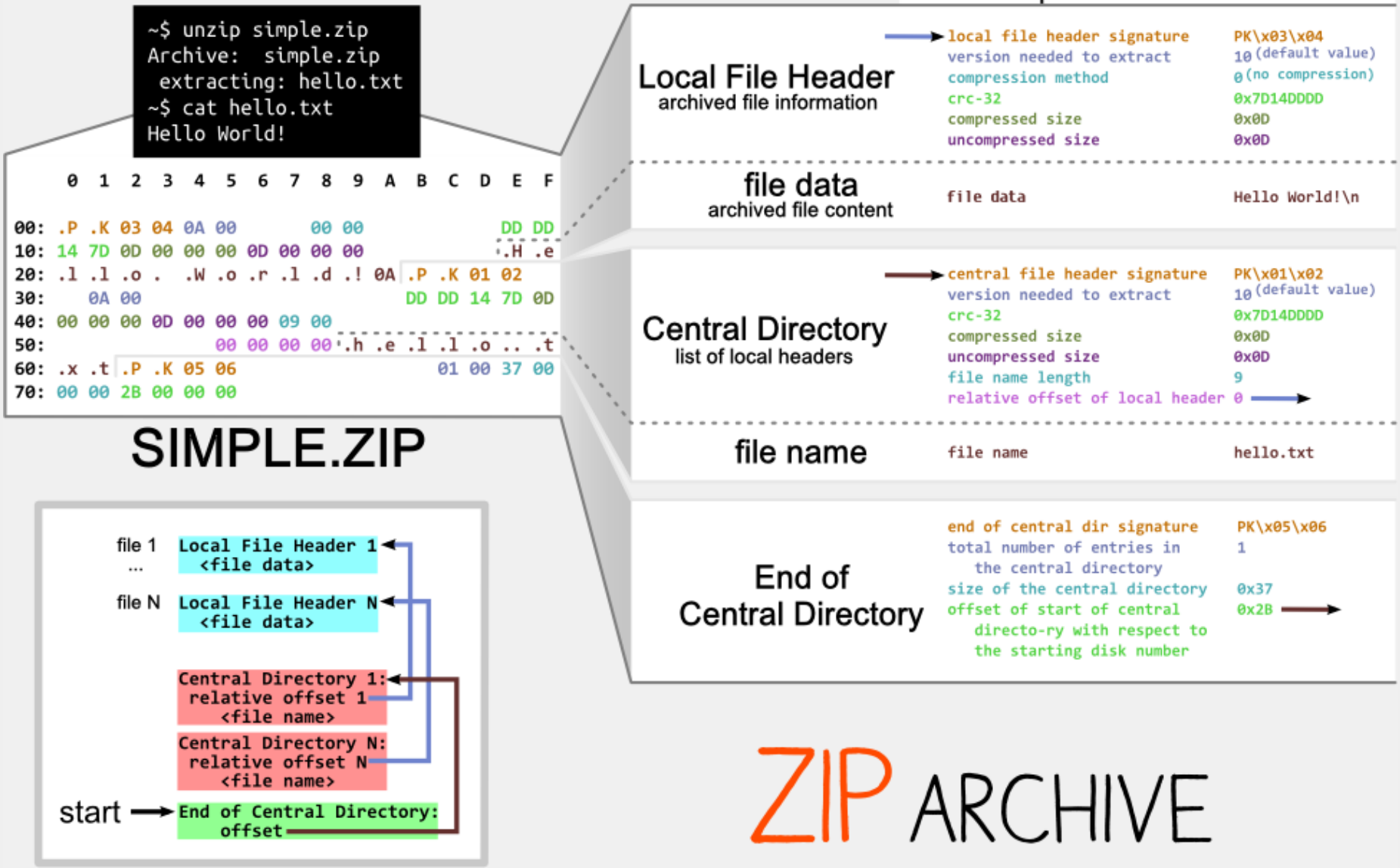
- fun
- bypassing security
  - same-origin policy
  - evade detection
  - exfiltration
  - signing
    - Android Master Key

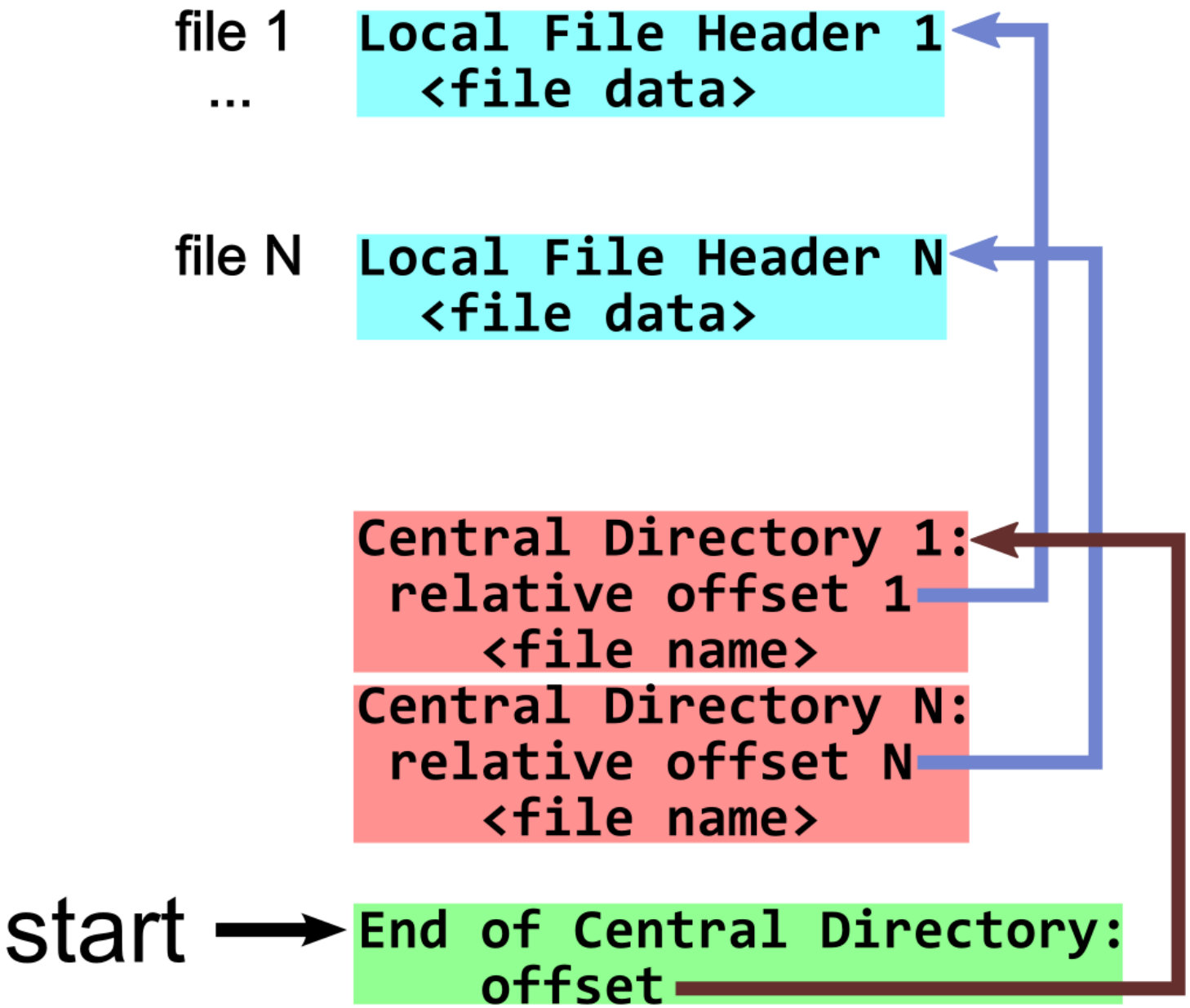
ZIP

excerpt from Gynvael's talk:

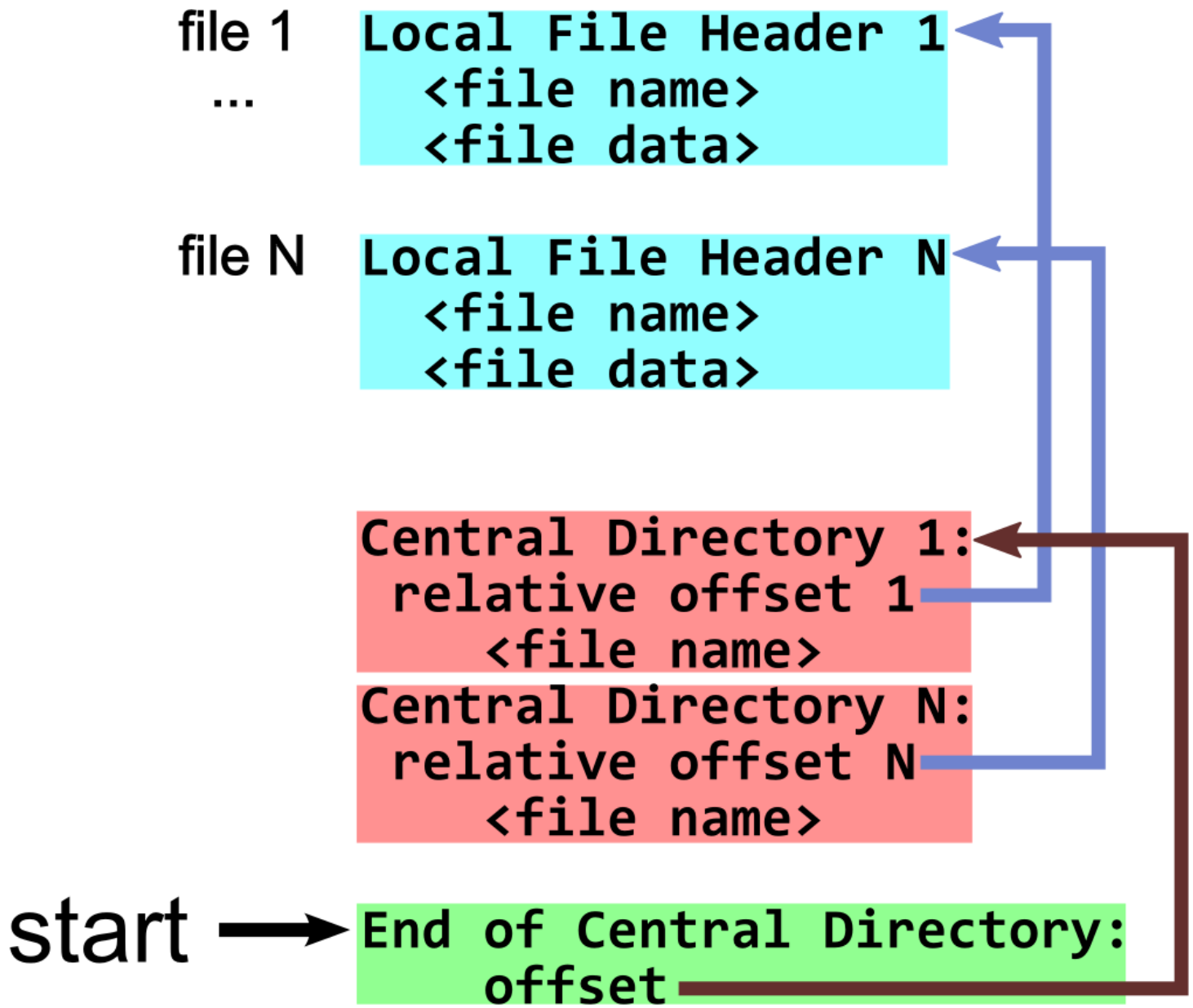
"Dziesięć tysięcy pułapek: ZIP, RAR, etc."

(http://gynvael.coldwind.pl/?id=523)


~$ unzip simple.zip
Archive: simple.zip
extracting: hello.txt
~$ cat hello.txt
Hello World!
0 1 2 3 4 5 6 7 8 9 A B C D E F
00: .P .K 03 04 0A 00 00 00 DD DD
10: 14 7D 0D 00 00 00 0D 00 00 00 .H .e
20: .l .l .o . .W .o .r .l .d .! 0A .P .K 01 02
30: 0A 00 DD DD 14 7D 0D
40: 00 00 00 0D 00 00 00 09 00
50: 00 00 00 00 .h .e .l .l .o .. .t
60: .x .t .P .K 05 06 01 00 37 00
70: 00 00 2B 00 00 00
SIMPLE.ZIP
description
value
Local File Header
archived file information
local file header signature
PK\x03\x04
version needed to extract
10 (default value)
compression method
0 (no compression)
crc-32
0x7D14DDDD
compressed size
0x0D
uncompressed size
0x0D
file data
archived file content
file data
Hello World!\n
Central Directory
list of local headers
central file header signature
PK\x01\x02
version needed to extract
10 (default value)
crc-32
0x7D14DDDD
compressed size
0x0D
uncompressed size
0x0D
file name length
9
relative offset of local header
0
file name
file name
hello.txt
End of Central Directory
end of central dir signature
PK\x05\x06
total number of entries in the central directory
1
size of the central directory
0x37
offset of start of central directo-ry with respect to the starting disk number
0x2B
file 1
...
file N
Local File Header 1
<file data>
Local File Header N
<file data>
Central Directory 1:
relative offset 1
<file name>
Central Directory N:
relative offset N
<file name>
start
End of Central Directory:
offset
ZIP ARCHIVE

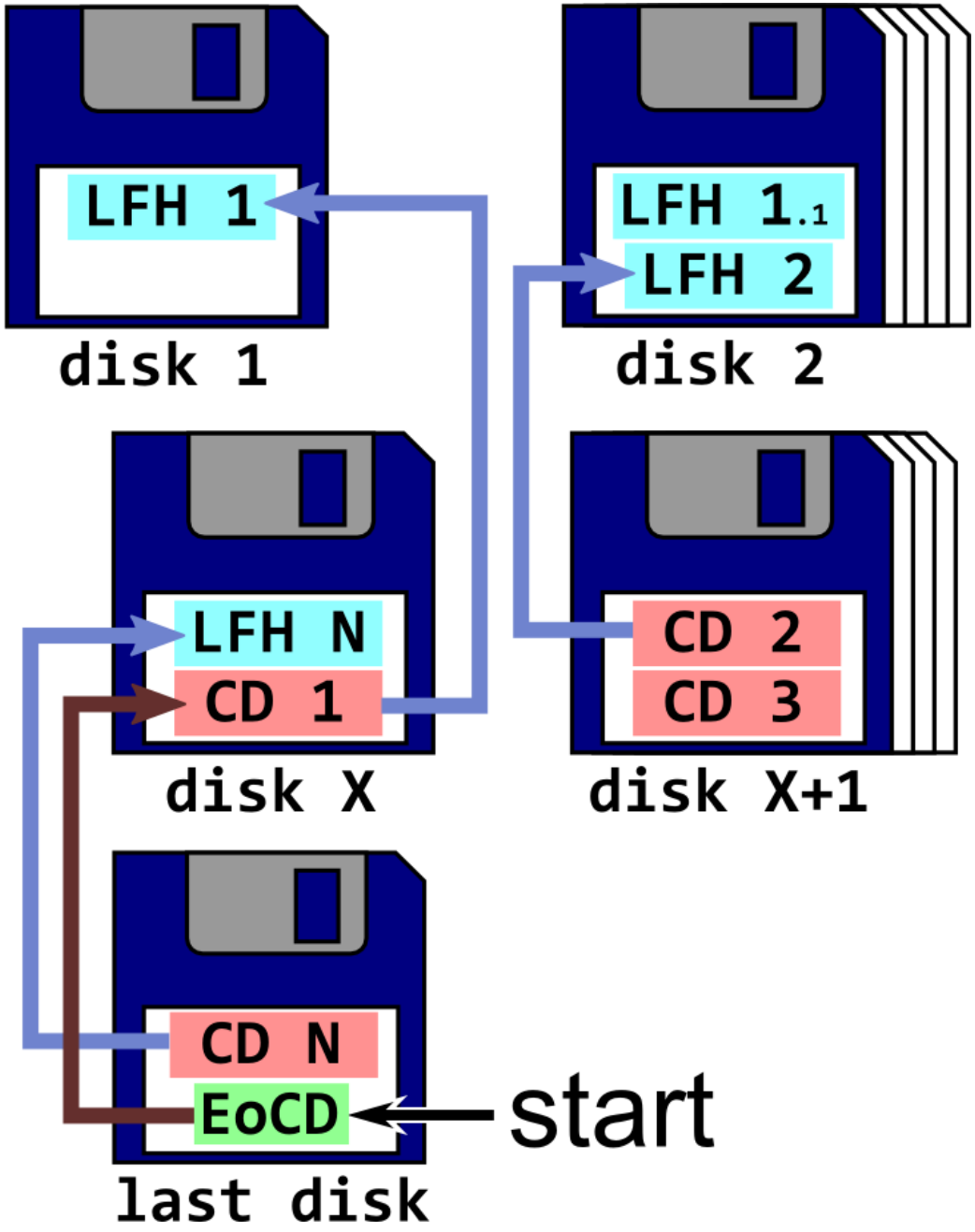


ZIP structures are parsed from the end.



File names are actually duplicated.

Why this weird structure?



ZIP archives were commonly read & written on the fly over multiple floppies.

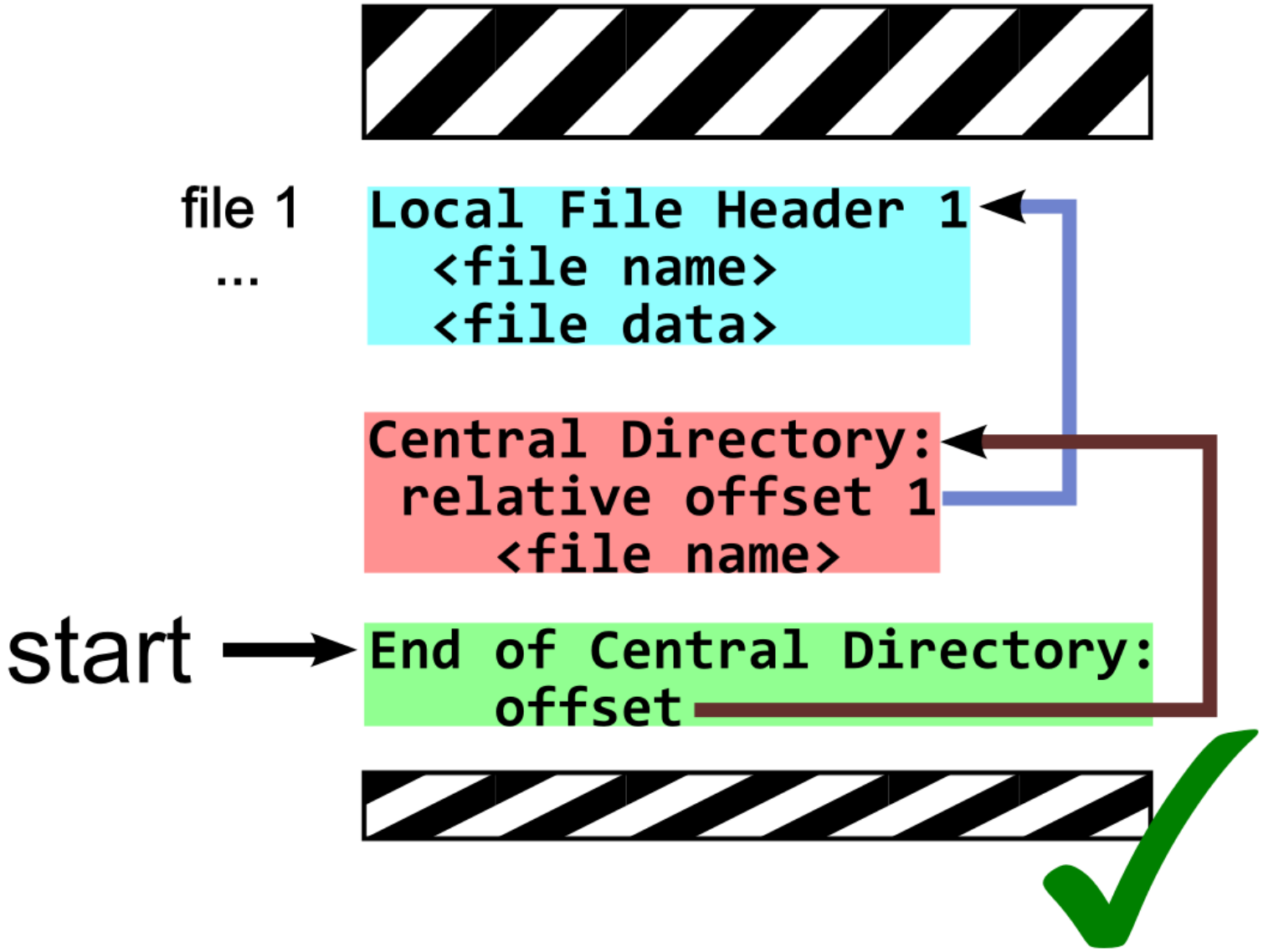
Minimize floppy swaps

- creation
  a. create one LFH per file
     Floppy full ⇒ start a new LFH on the next floppy
  b. when all files are finished, write CDs sequence (1/file)
  c. when all CDs are written, write the EoCD
- extraction
  a. insert last floppy (contains the EoCD)
  b. insert the floppy with 1st CD
     (often, the last floppy contains EoCD + all CDs)
  c. insert the corresponding LFH's first floppy
     insert next floppies if required

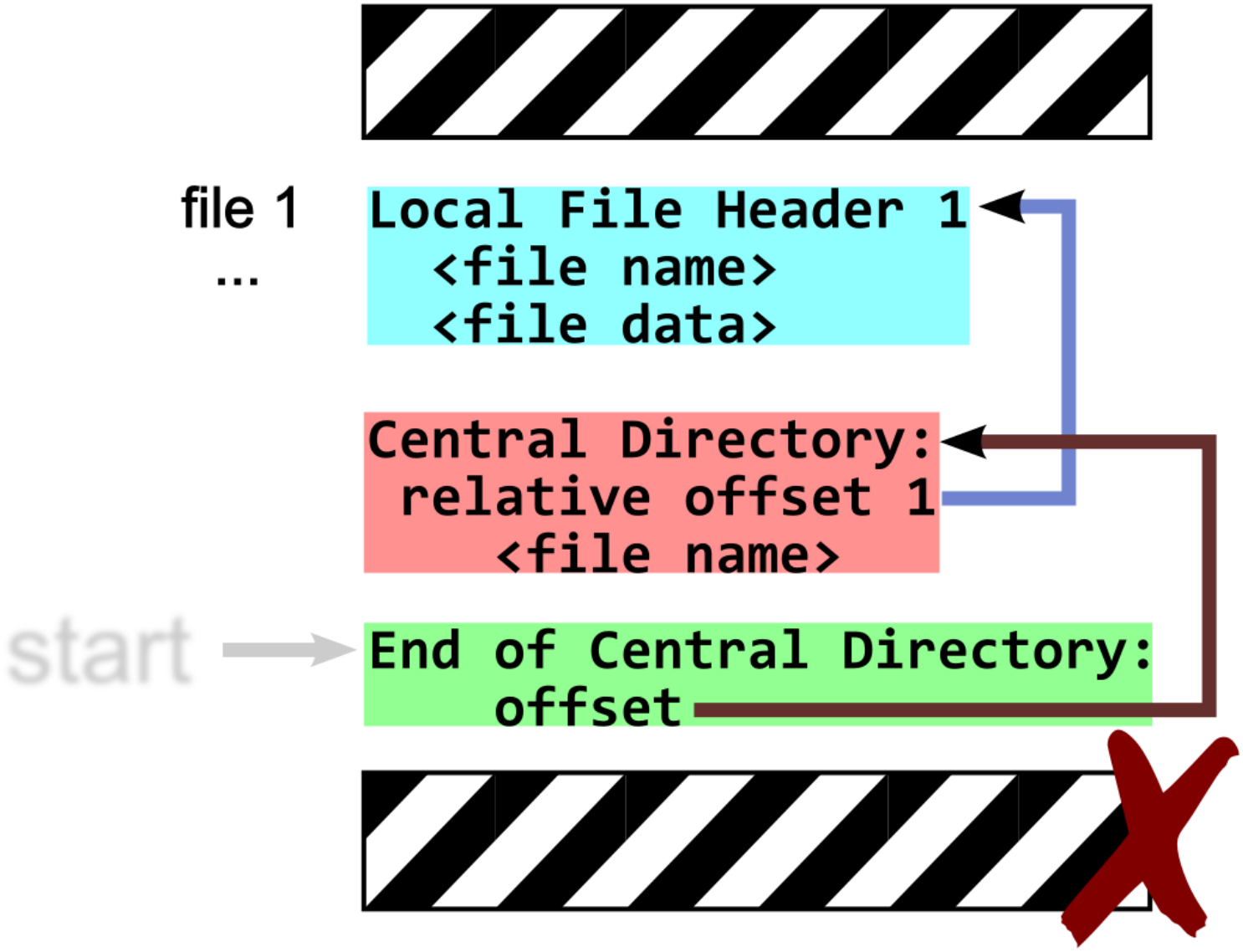
ZIP *was* very useful, but *now* it's awkward.

Newer archive formats are parsed top-down.

Position in the file

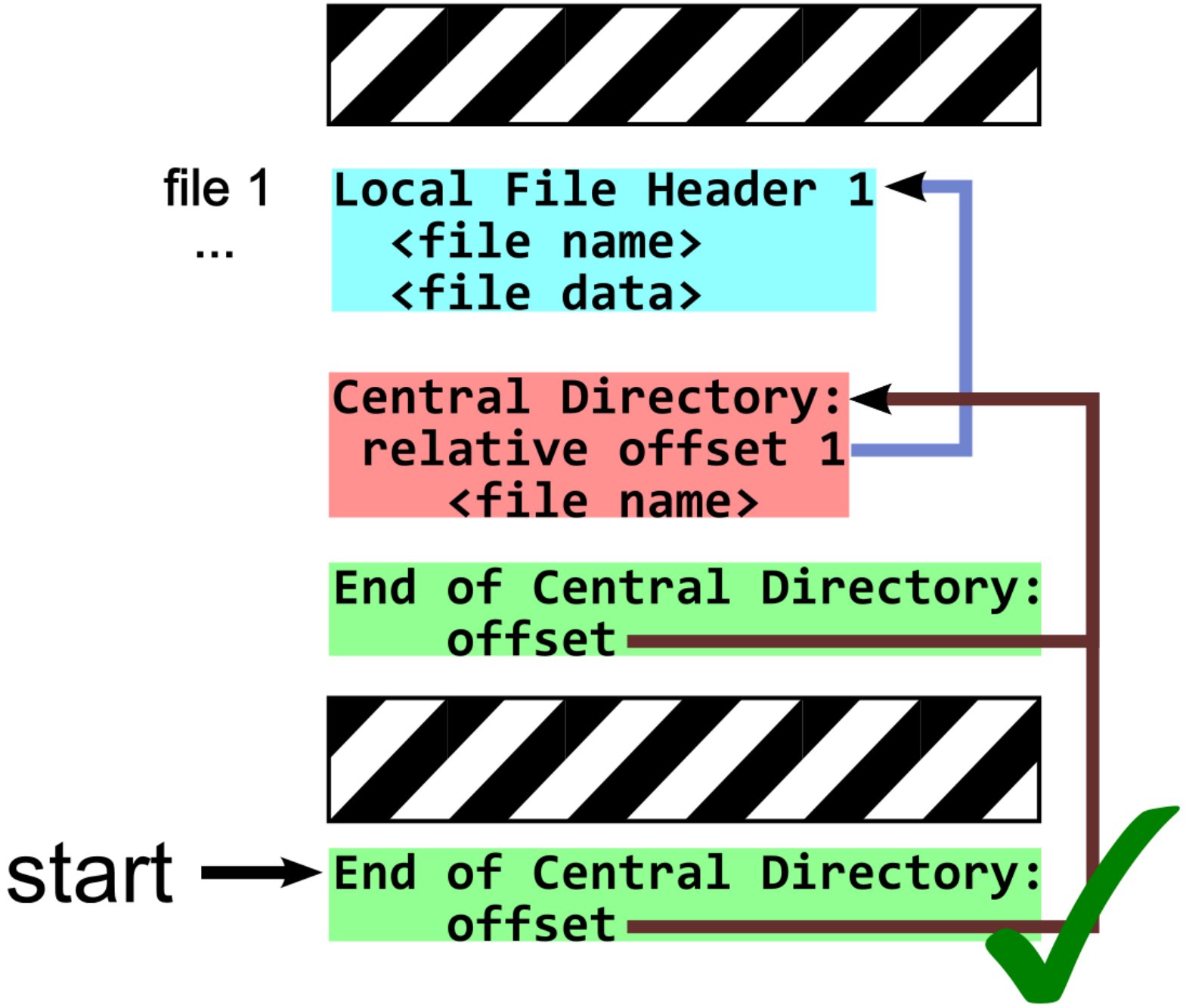


Prepended and appended data is tolerated...



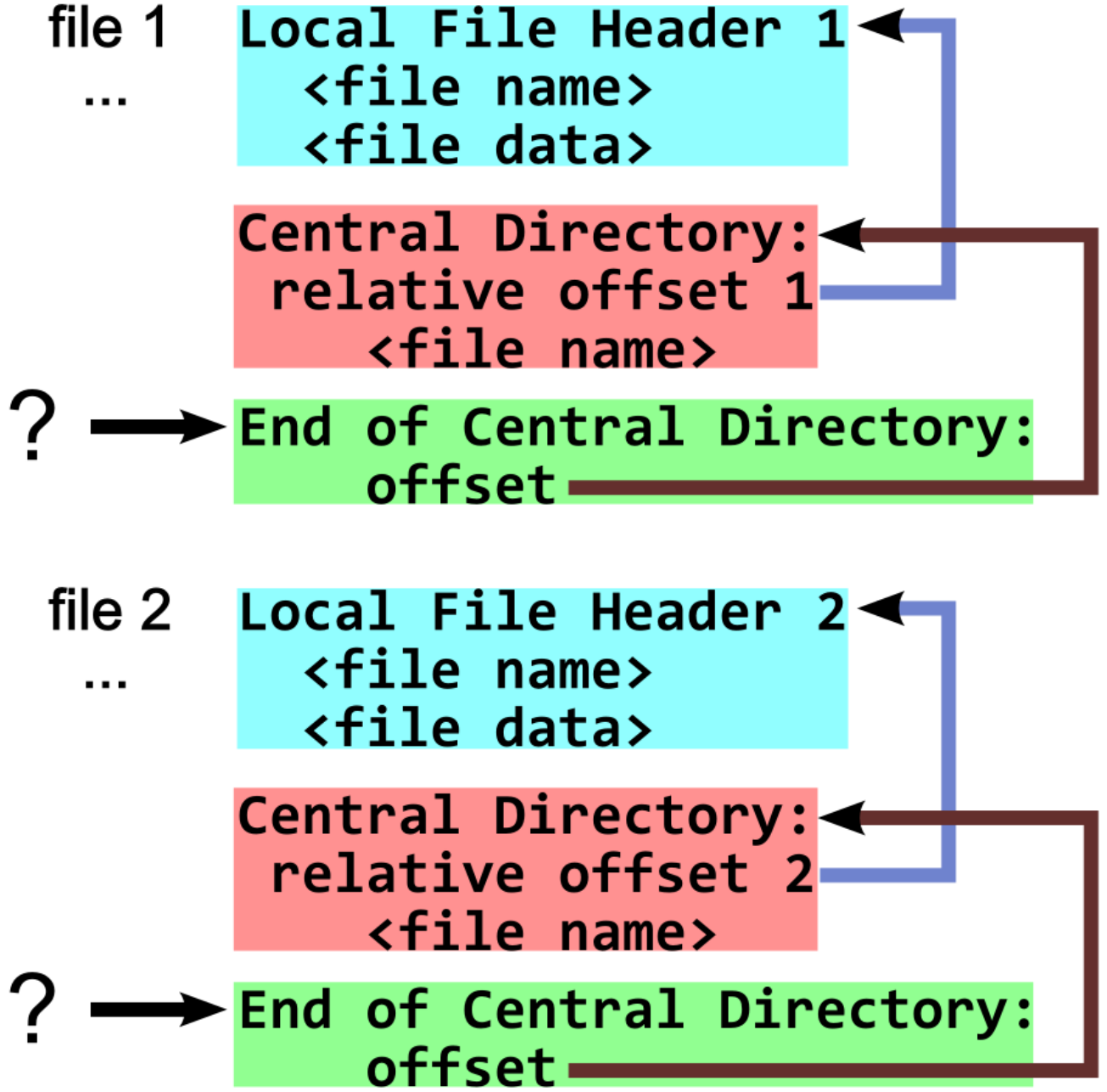
...but not too much!

(for obvious performance reason)

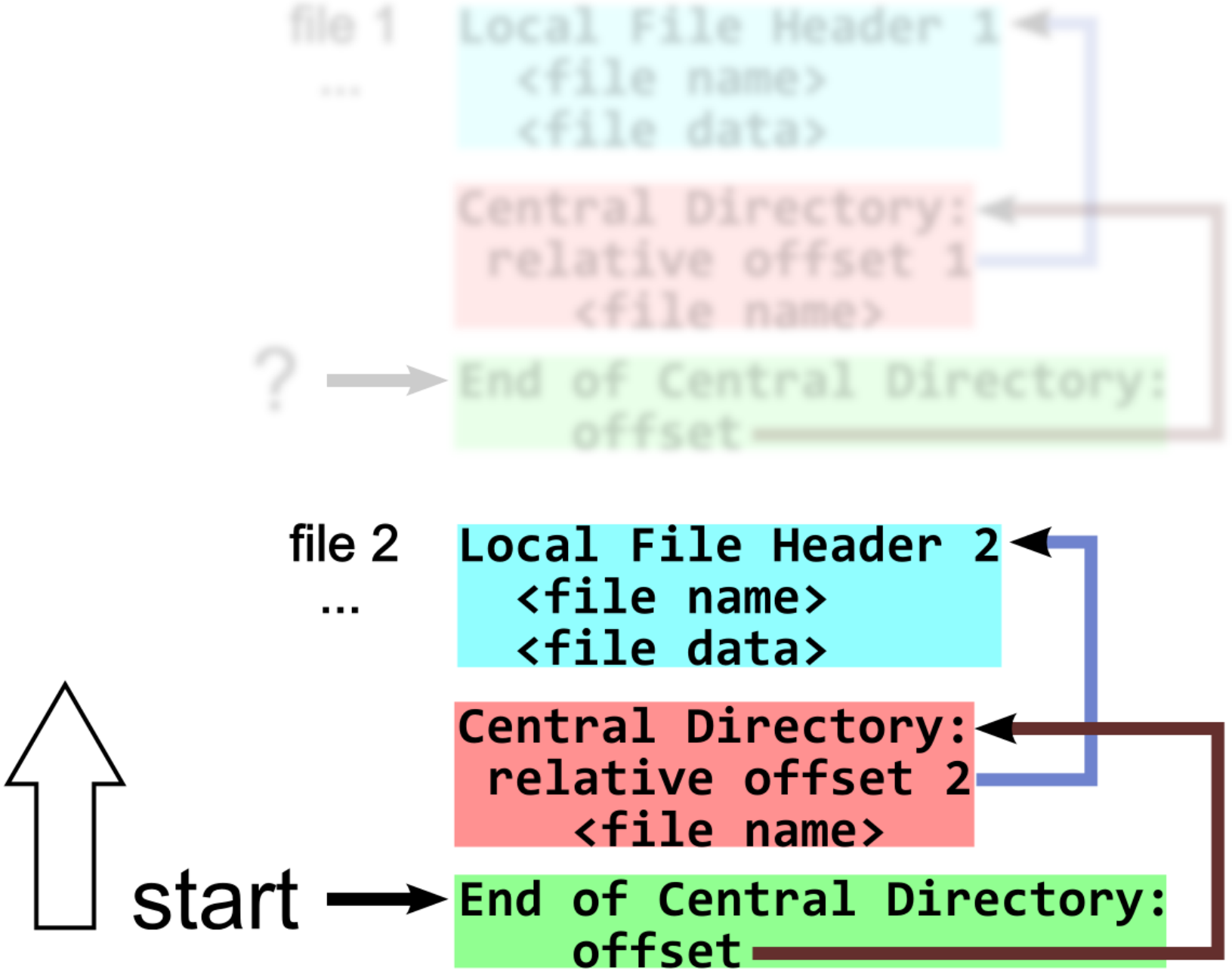


Duplicating the (relatively small) EoCD increases compatibility.

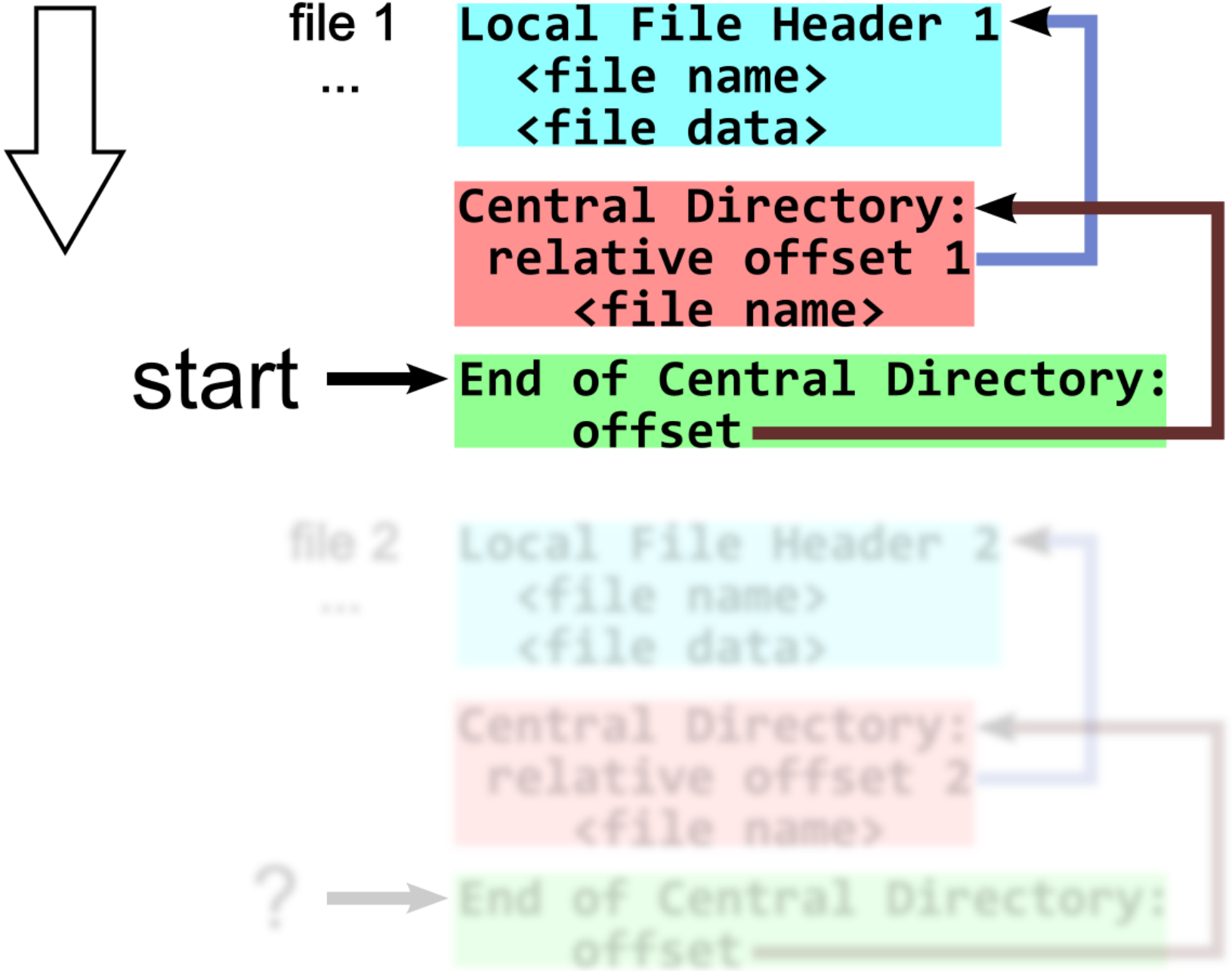
Scanning direction



If you concatenate 2 archives...

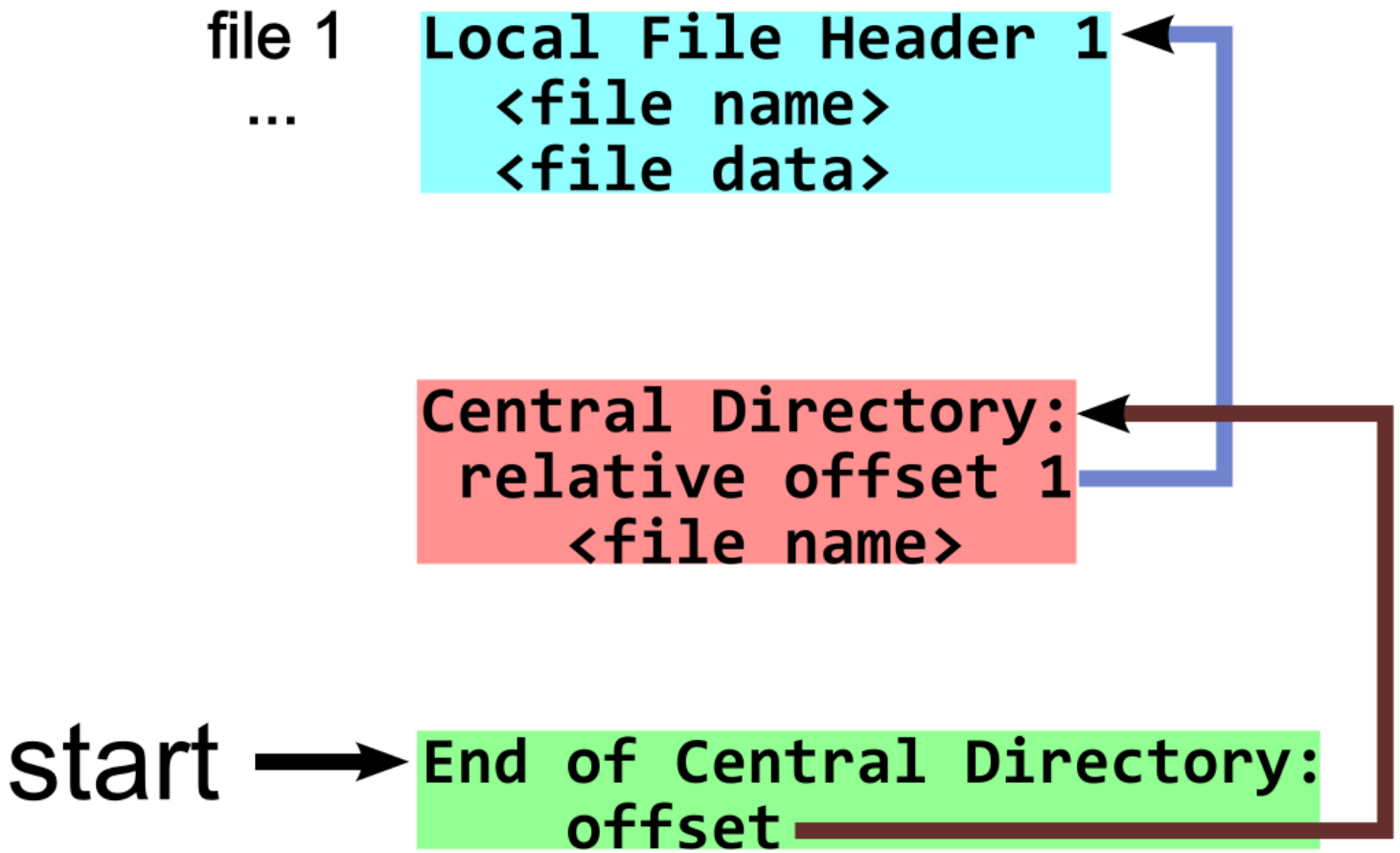


...if you parse bottom-up (standard), you find the 2nd one...

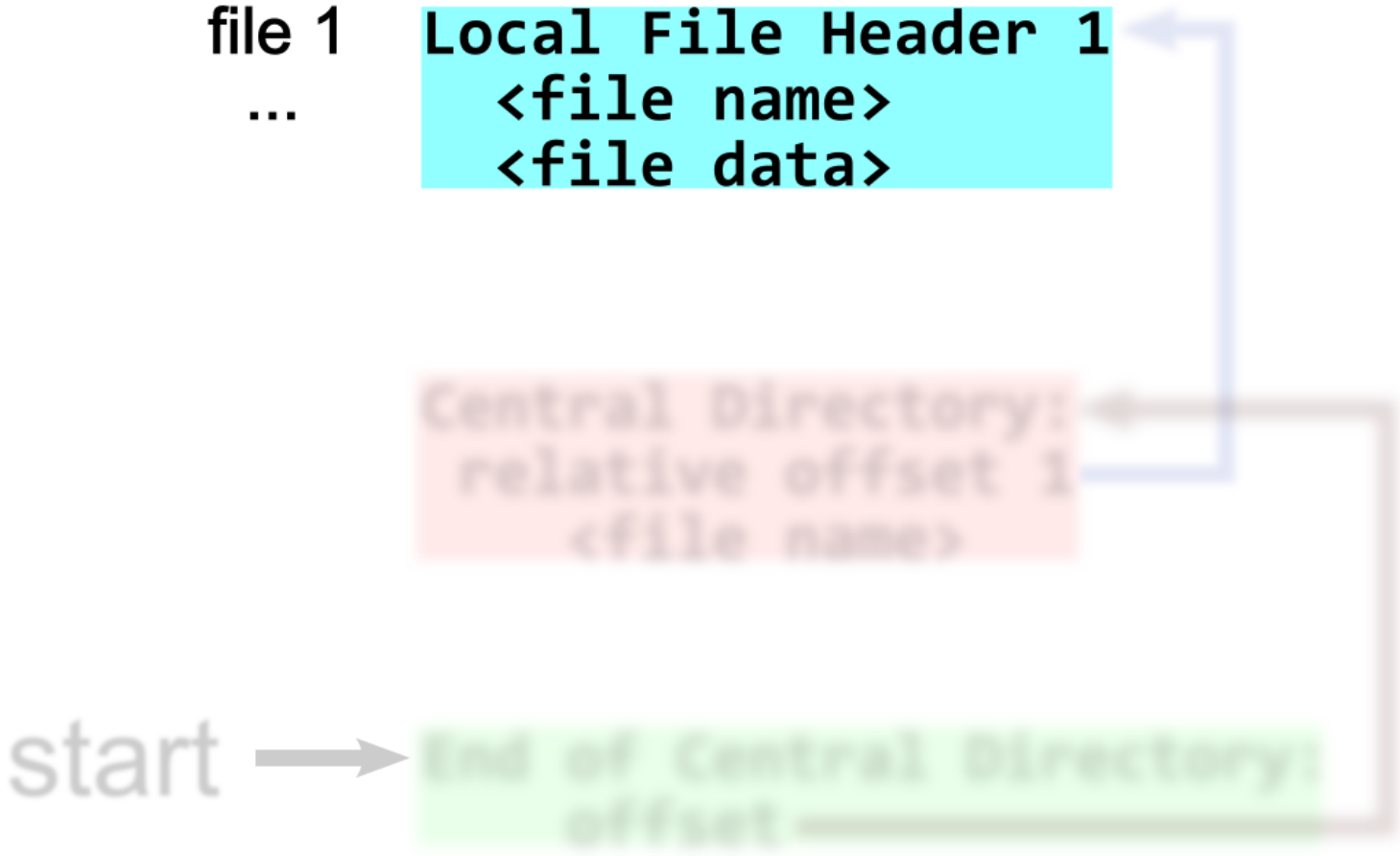


...but you will get the other archive if you parse top-down.

Superfluous headers

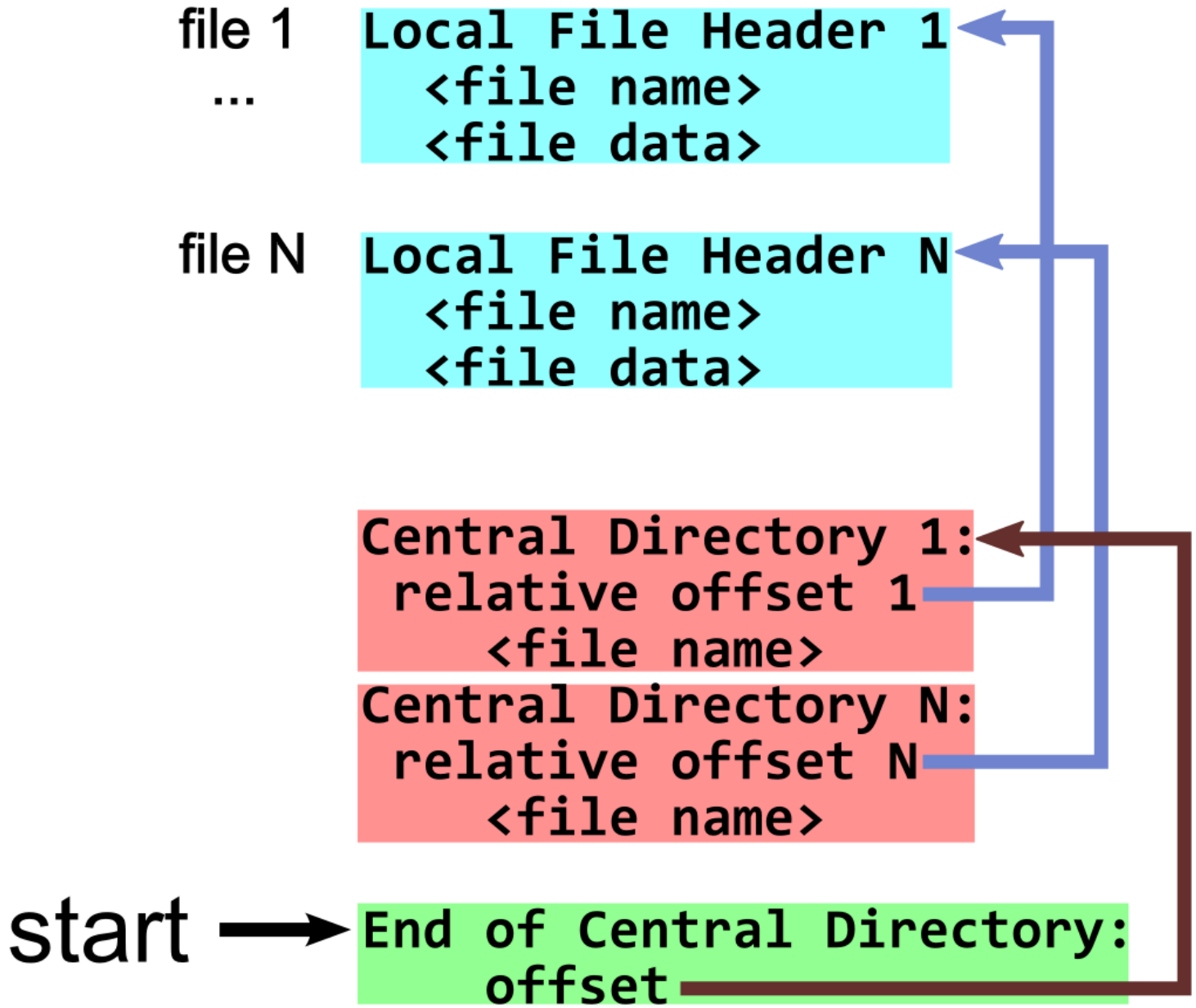


You could parse everything nicely...

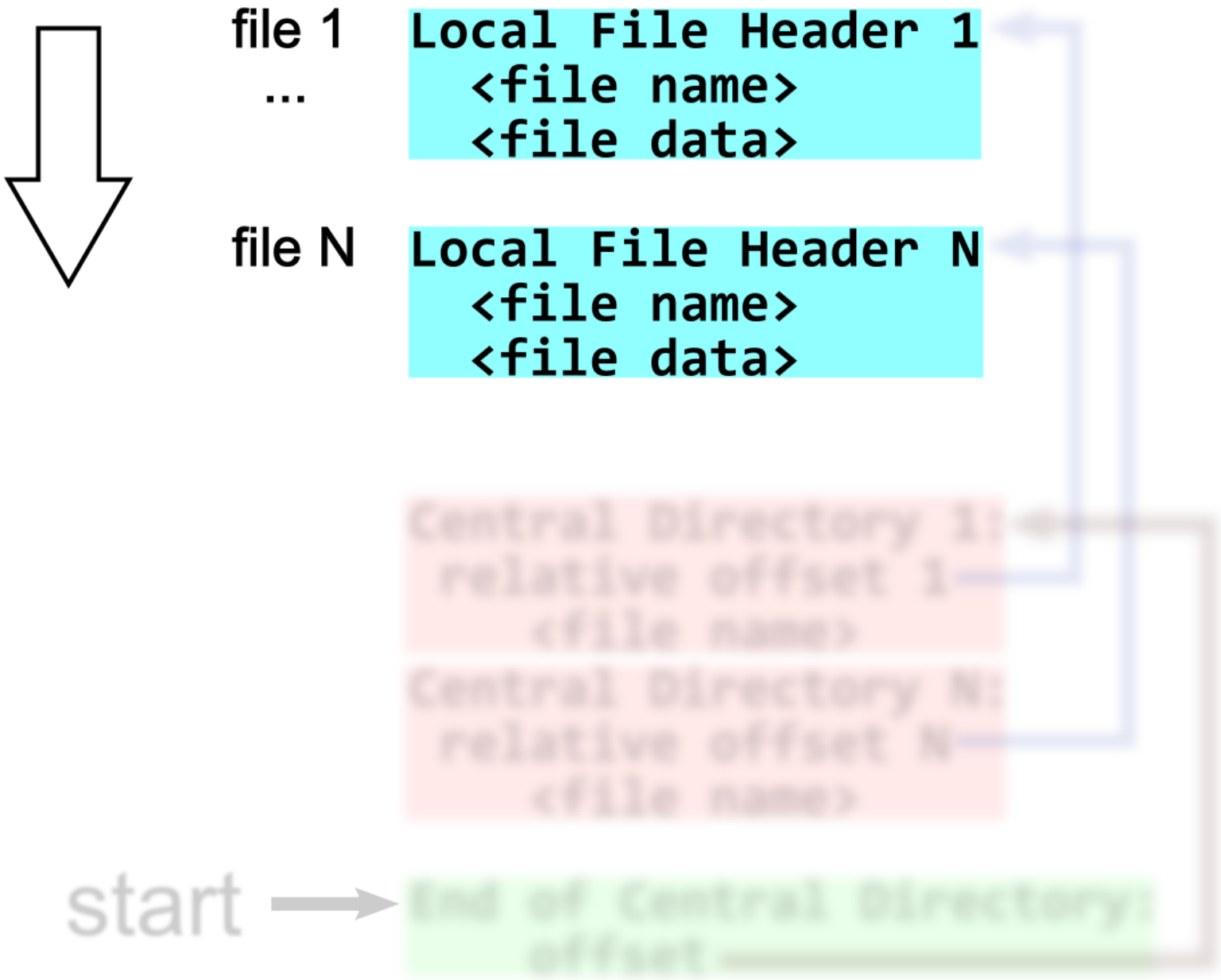


...but in the end, only the Local File Headers matter.

1 file = 1 Local File Header



Since most ZIP archives start with a sequence of Local File Headers...



You can parse them top-down (until a break) and ignore the CD and EoCD.

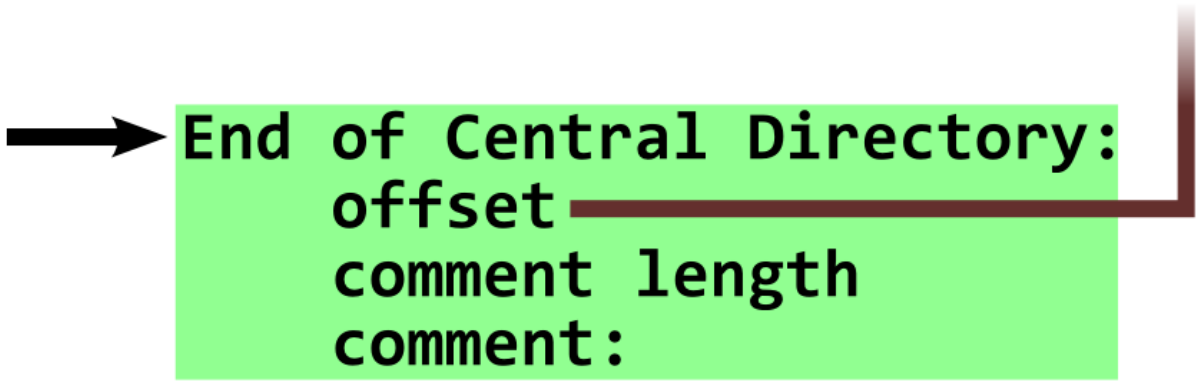
Standard parsing: bottom-up + all headers

“Efficient” parsing: top-down + LFHs only

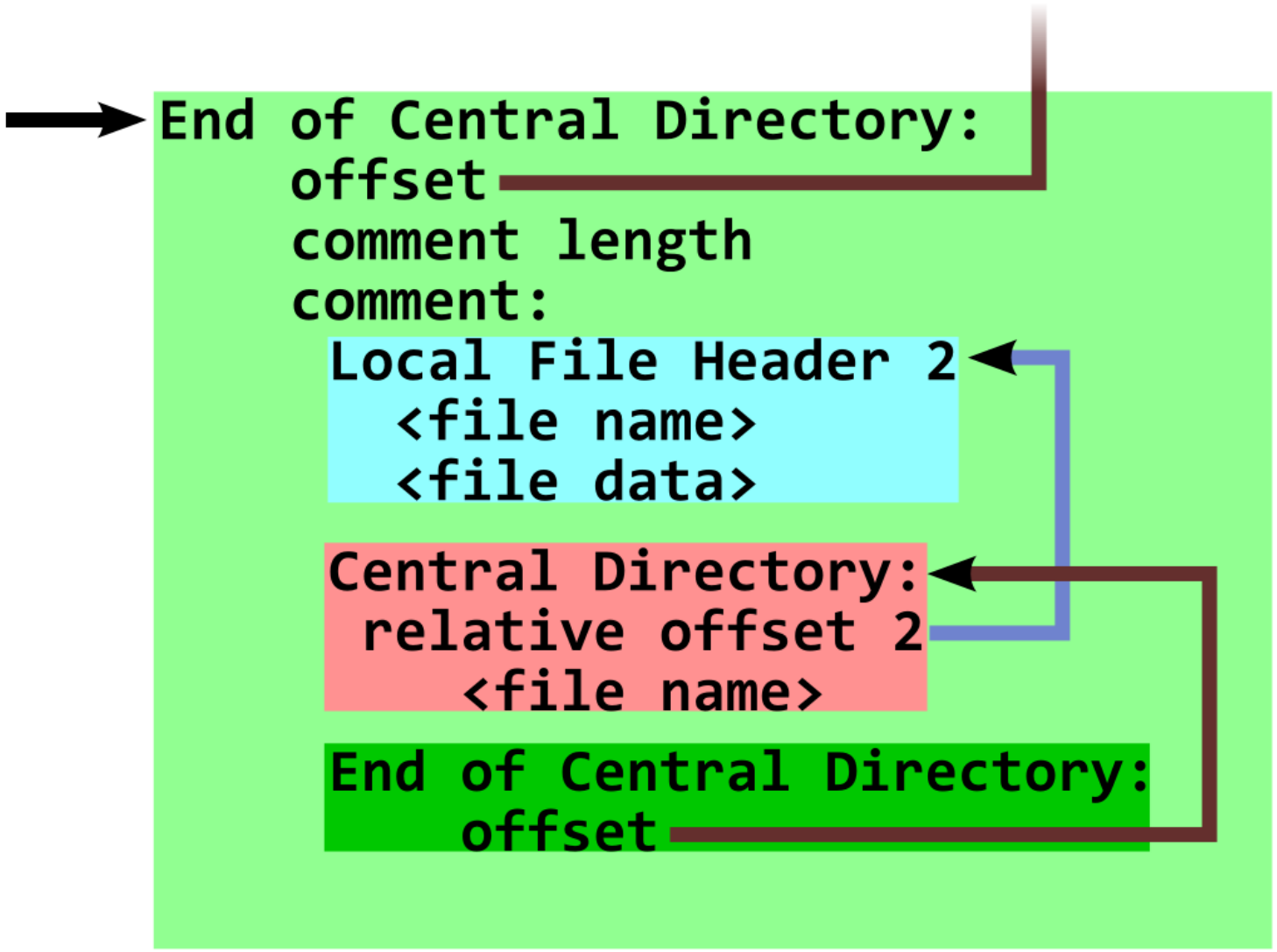
Not standard, but good enough in most cases.

Nowadays, most ZIPs are a sequence of LFHs from the start

ZIP Archive comment



The EoCD contains an optional comment field...

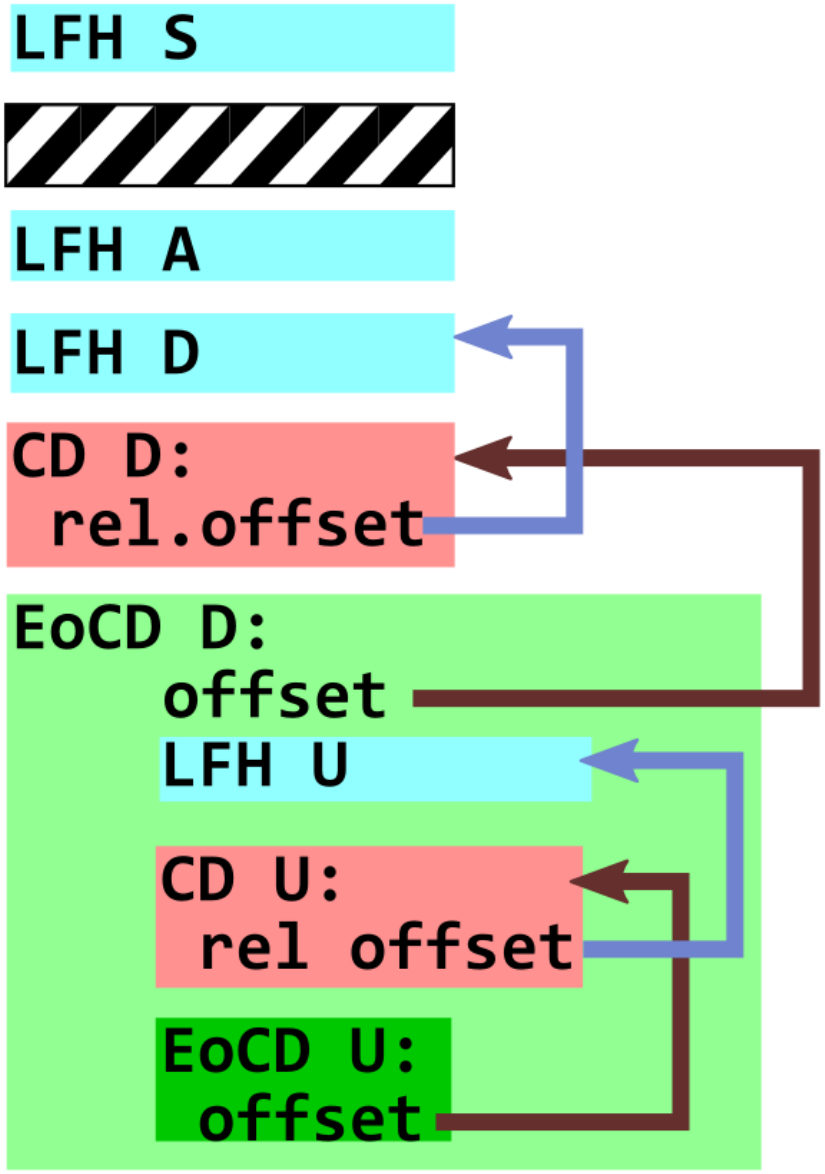


...that can contain a complete archive !

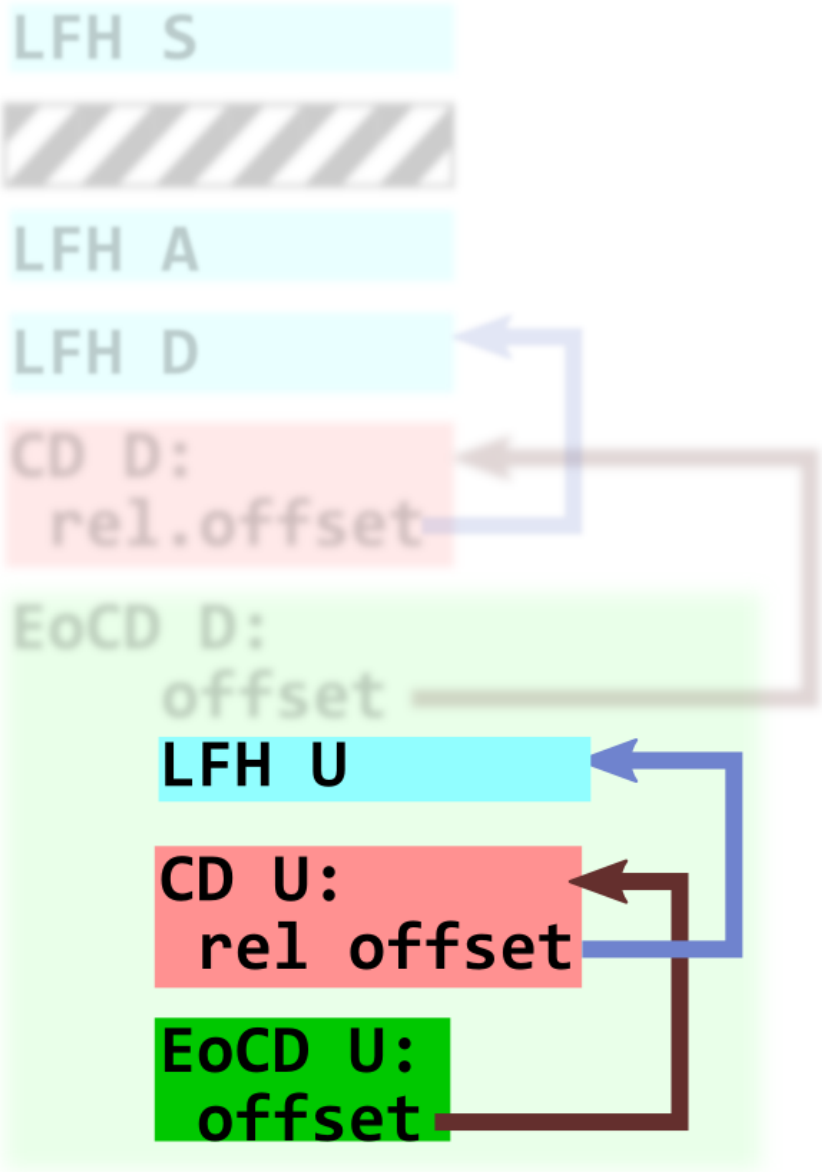
Recap

- Parsing direction:
  - standard is bottom-up
  - parsing LFHs from the start would work in most cases
- ZIP should be located near the end of the file
  - or at least, its EoCD
- An archive comment can contain another complete archive

Let's test the parsers!
abstract.zip

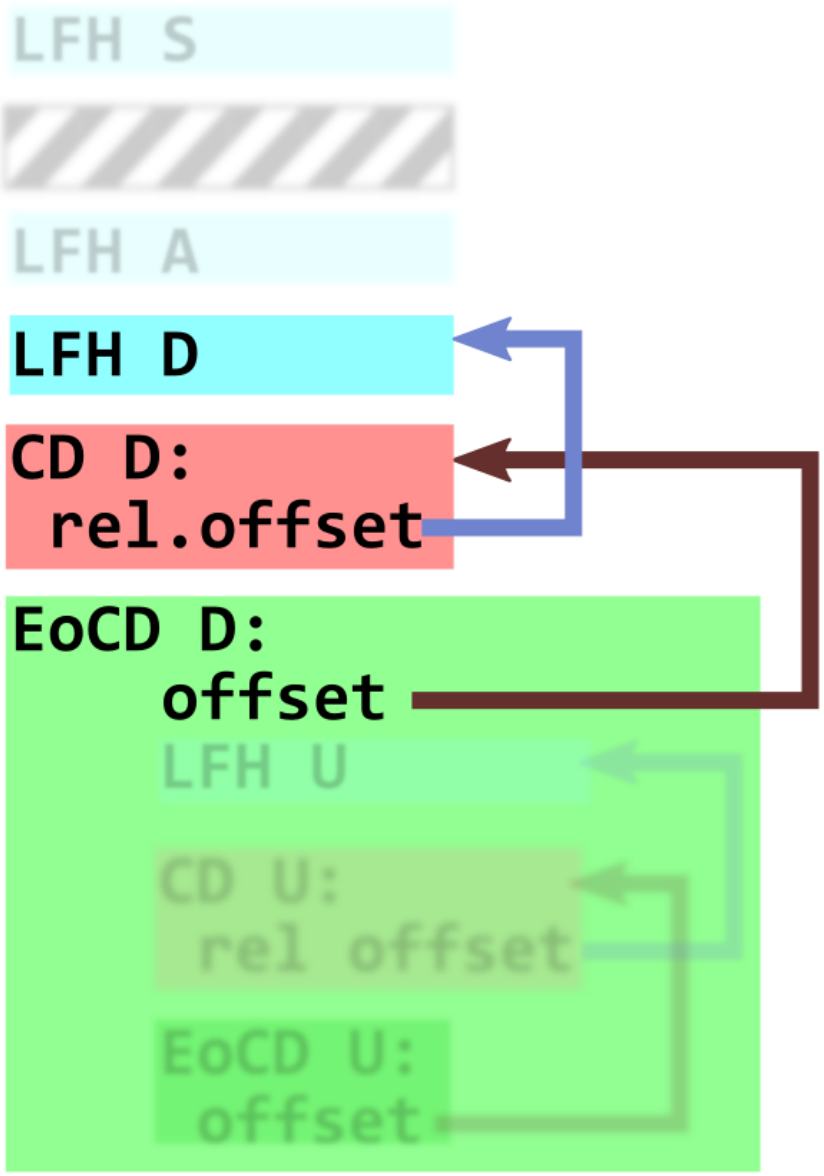


4 LFHs, 4 ways to parse this archive:



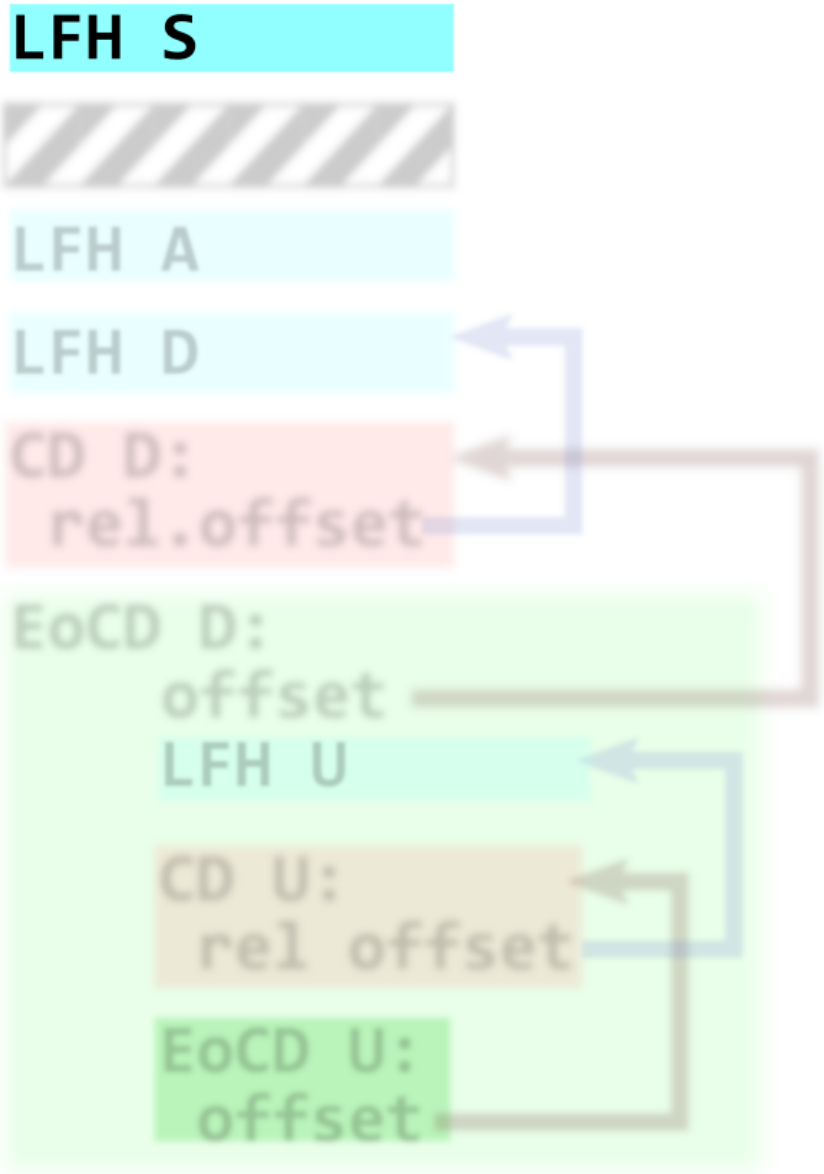
7-zip
C# DotNetZip
Java ZipFile
OSX UnZip
perl Archive::Zip
Python zipfile
Total Commander 8.01
UnZip 6.00 (Debian)
Windows 7 Explorer
WinRAR
Ada Zip-Ada v45
ALZip
b1.org
Midnight Commander
KGB Archiver
JSZip
Jeffrey's Exif Viewer
WOBZIP
GNOME File Roller
zip.vim v25
Emacs Zip-Archive mode
Go archive/zip
Pharo smalltalk 2.0 ZipArchive
Ubuntu less

1/ you parse it bottom-up



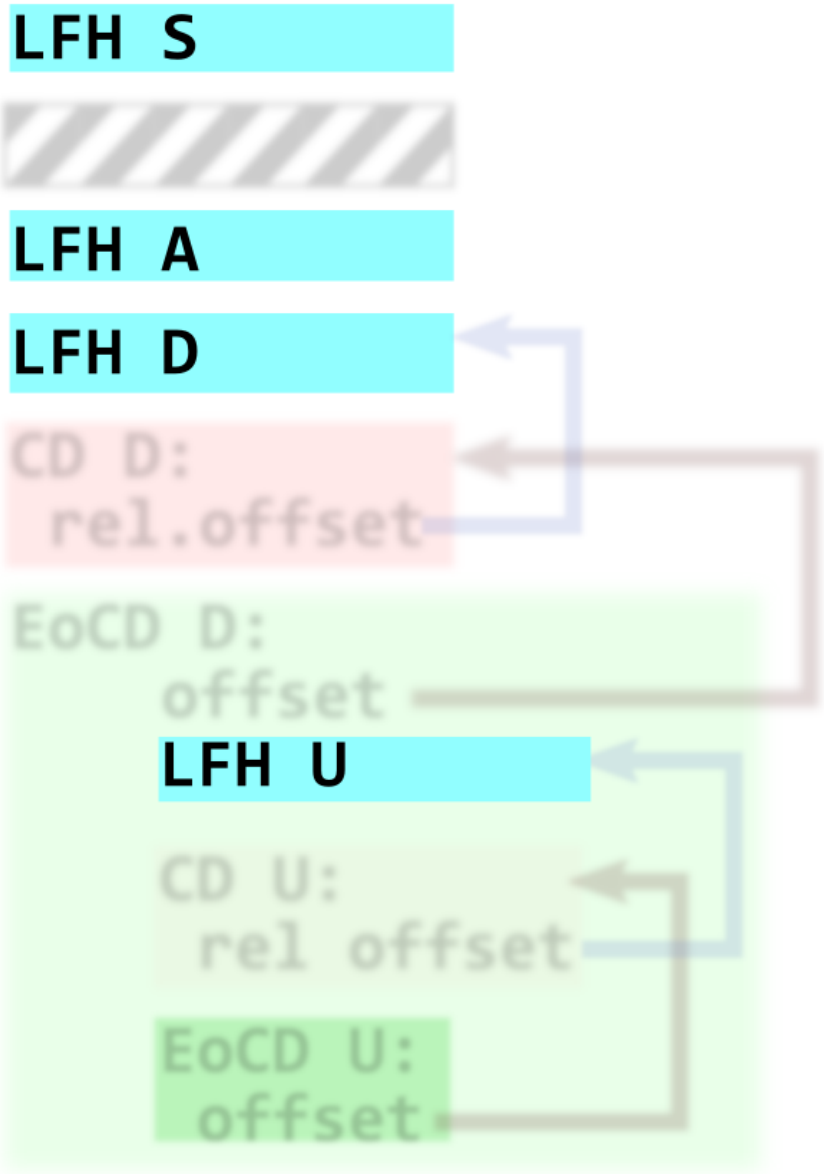
PHP ZipArchive
PHP zip_open ...
PHP zip:// wrapper
tcl + tclvfs + tclunzip

2/ you parse it top-down

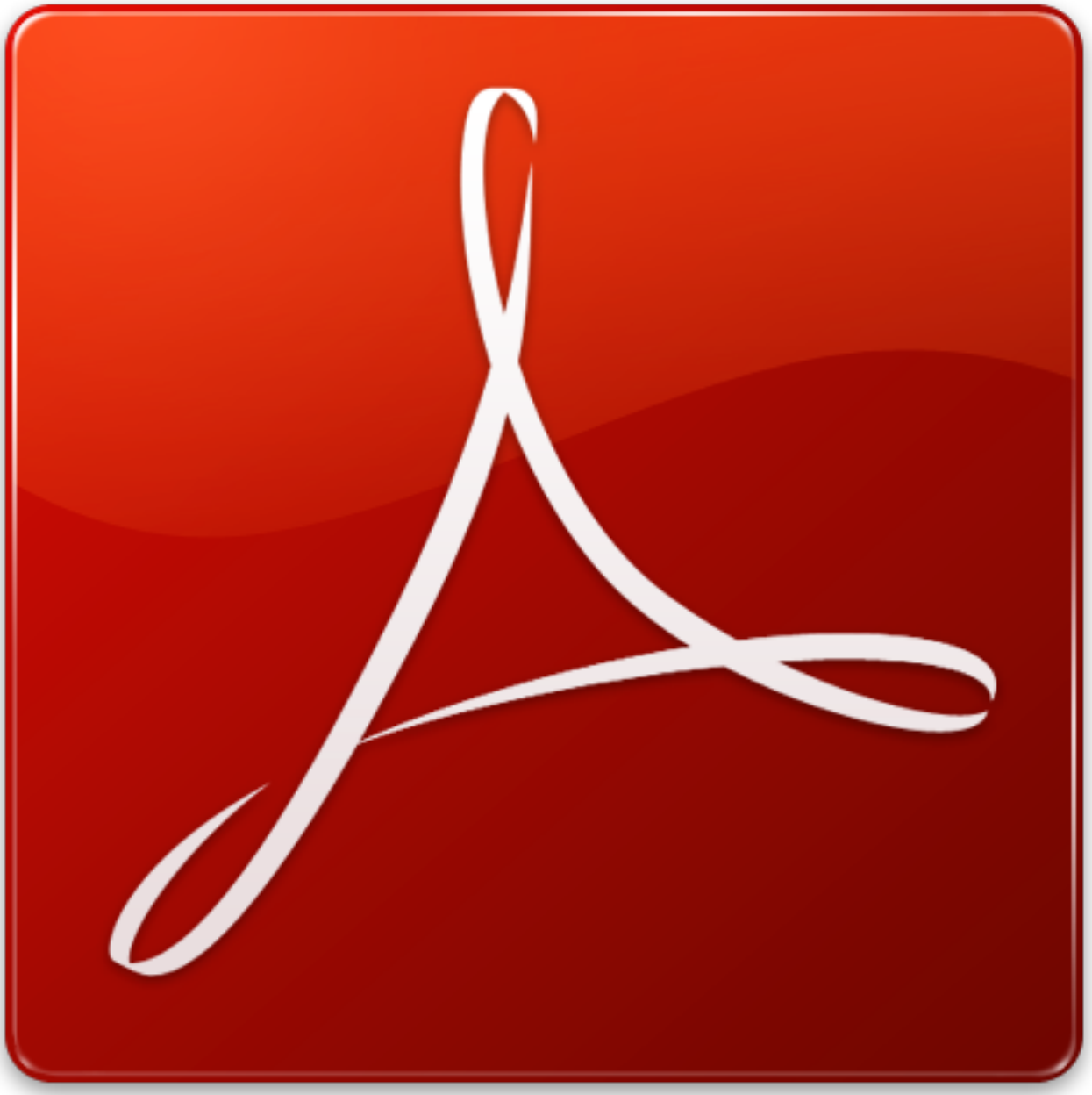


Ruby rubyzip2
Java ZipArchiveInputStream
java.util.zip.ZipInputStream

3/ look for LFHs from the start (until a break)



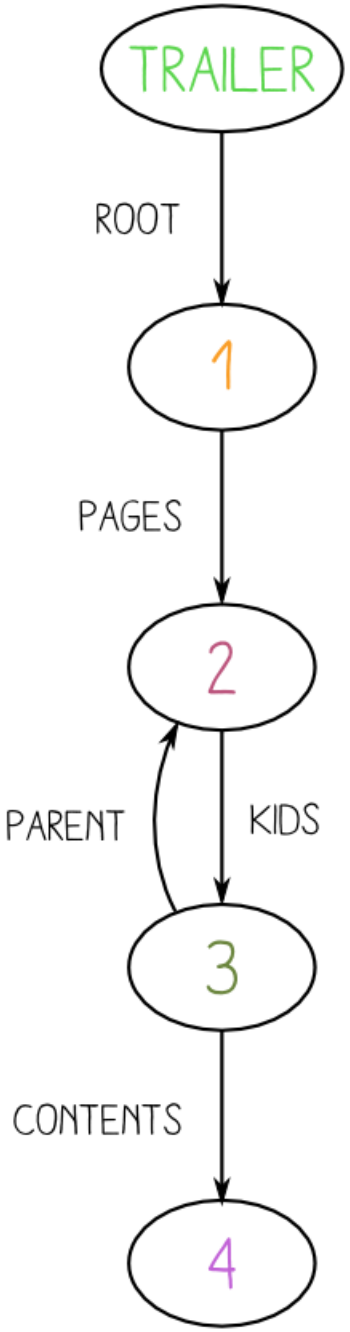
4/ scan for LFHs aggressively (you get all four)



http://youtu.be/JQrBgVRgqtc?t=11m15s
https://speakerdeck.com/ange/pdf-secrets-hiding-and-revealing-secrets-in-pdf-documents?slide=44


```
%PDF-1.1

1 0 obj
<<
  /Pages 2 0 R
>>
endobj

2 0 obj
<<
  /Type /Pages
  /Count 1
  /Kids [3 0 R]
>>
endobj

3 0 obj
<<
  /Type /Page
  /Contents 4 0 R
  /Parent 2 0 R
  /Resources <<
    /Font <<
      /F1 <<
        /Type /Font
        /Subtype /Type1
        /BaseFont /Arial
      >>
    >>
  >>
>>
endobj
```

```
4 0 obj
<< /Length 50 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!)Tj
ET
endstream
endobj

xref
0 5
0000000000 65535 f
0000000010 00000 n
0000000047 00000 n
0000000111 00000 n
0000000313 00000 n

trailer
<<
  /Root 1 0 R
>>

startxref
413
%%EOF
```



PDF Trick #1

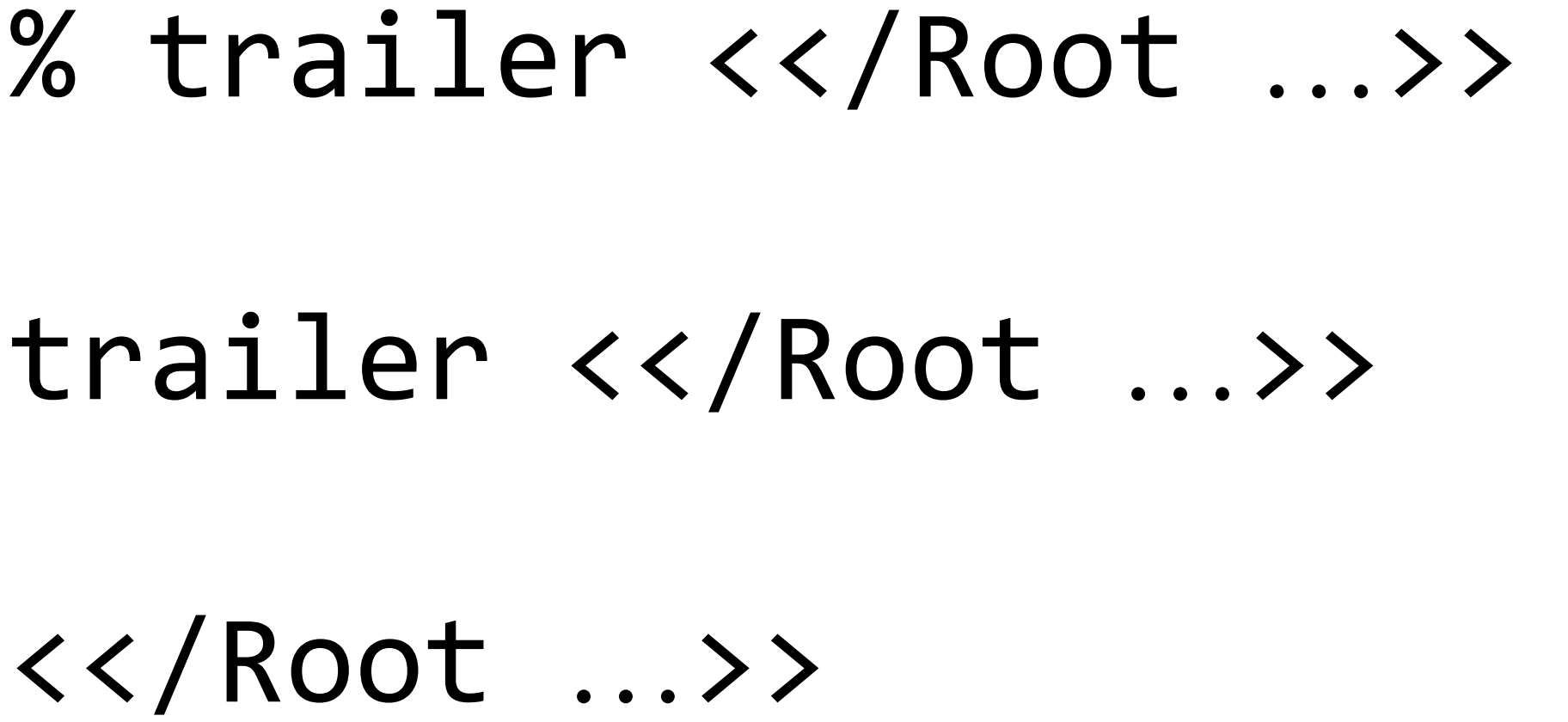
trailers

7.5.5 File Trailer

The *trailer* of a PDF file enables a conforming reader to quickly find the cross-reference table and certain special objects. Conforming readers should read a PDF file from its end. The last line of the file shall contain only the end-of-file marker, **%%EOF**. The two preceding lines shall contain, one per line and in order, the keyword **startxref** and the byte offset in the decoded stream from the beginning of the file to the beginning of the **xref** keyword in the last cross-reference section. The **startxref** line shall be preceded by the *trailer dictionary*, consisting of the keyword **trailer** followed by a series of key-value pairs enclosed in double angle brackets (<<…>>) (using LESS-THAN SIGNs (3Ch) and GREATER-THAN SIGNs (3Eh)). Thus, the trailer has the following overall structure:

trailer
<< key_1 $value_1$
key_2 $value_2$
…
key_n $value_n$
>>
startxref
Byte_offset_of_last_cross-reference_section
%%EOF

trailer ⇒ root object ⇒ complete document



a line comment - a correct trailer - a corrupted trailer

```
% trailer <</Root ...>>
```

```
trailer <</Root ...>>
```

```
<</Root ...>>
```

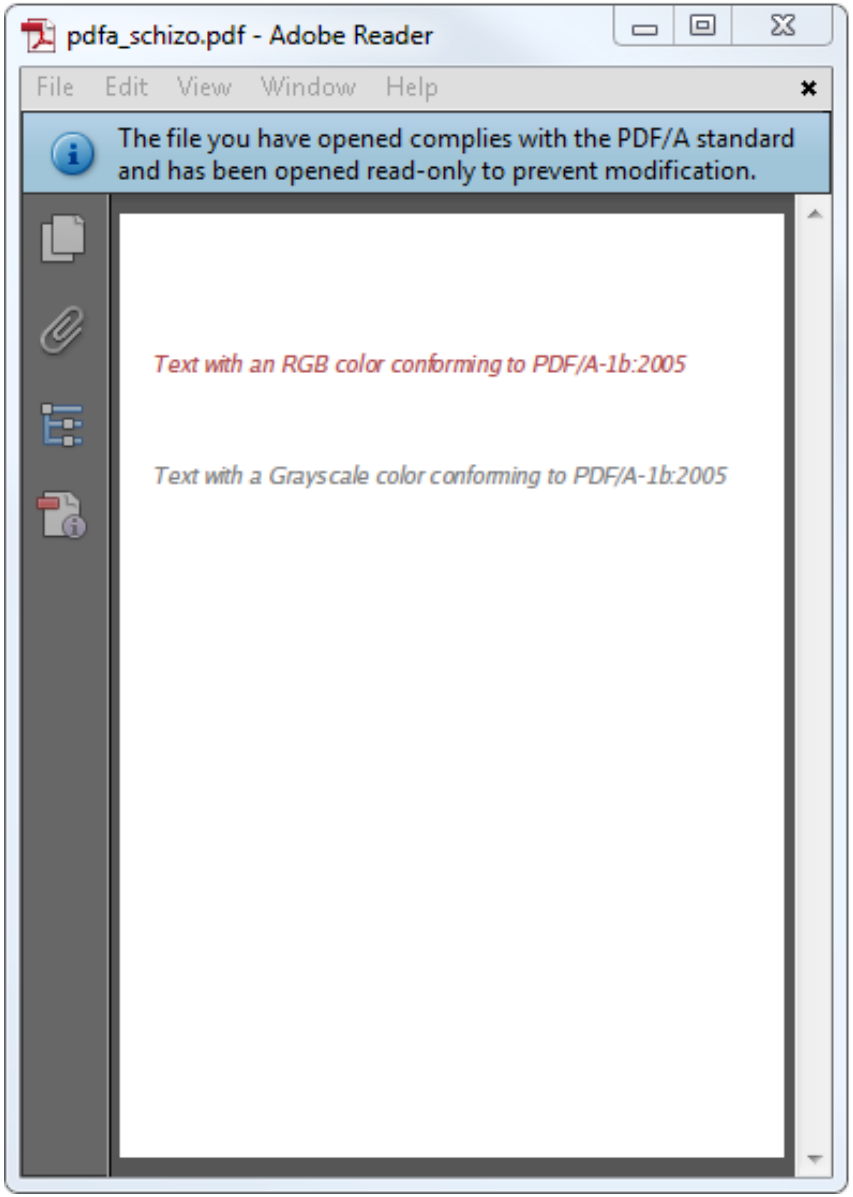
Each reader sees a different trailer.

PDF parsing

Each reader sees a completely different document

3 co-existing documents, all parsed through

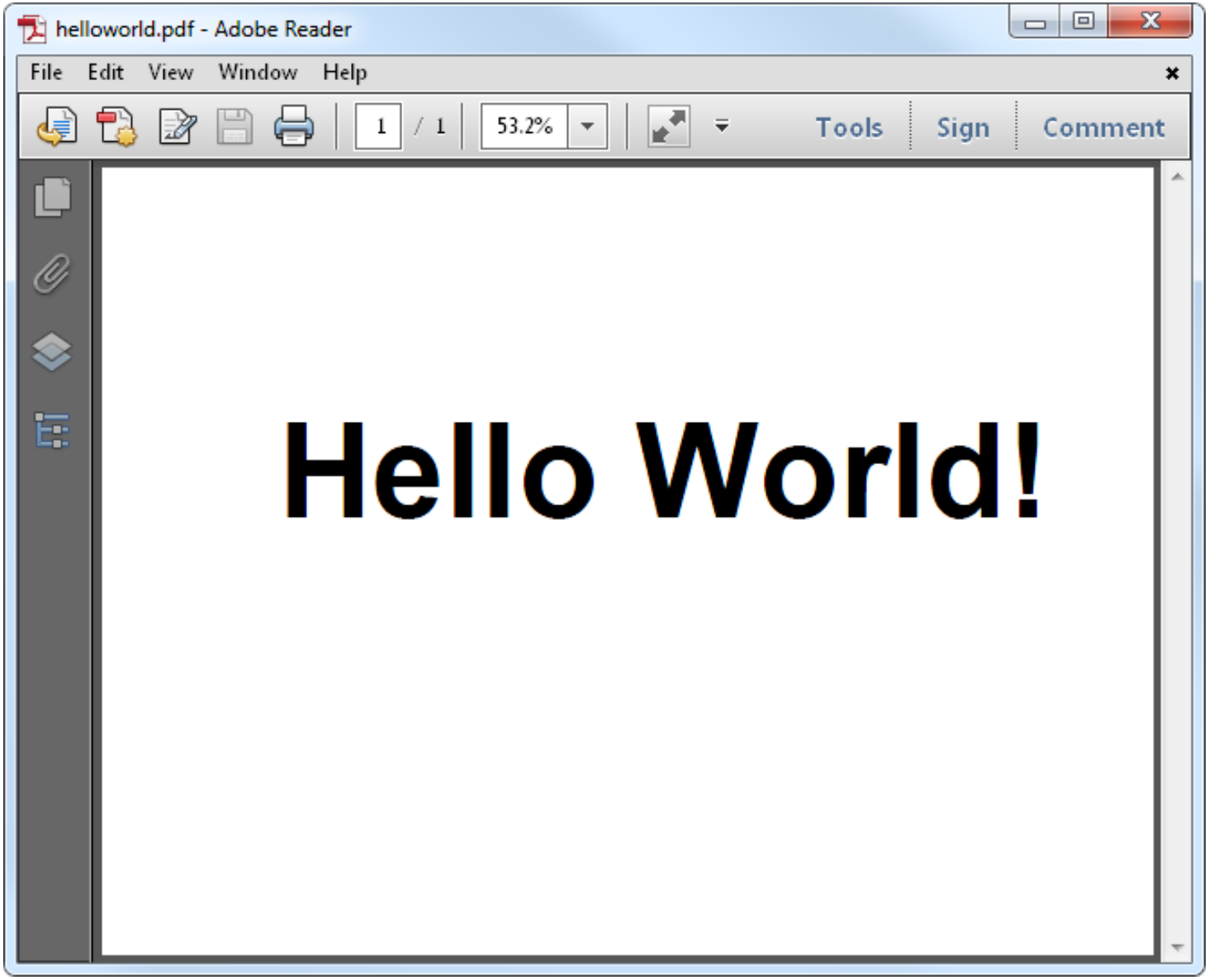
Viewers tolerance makes foreign elements ignored



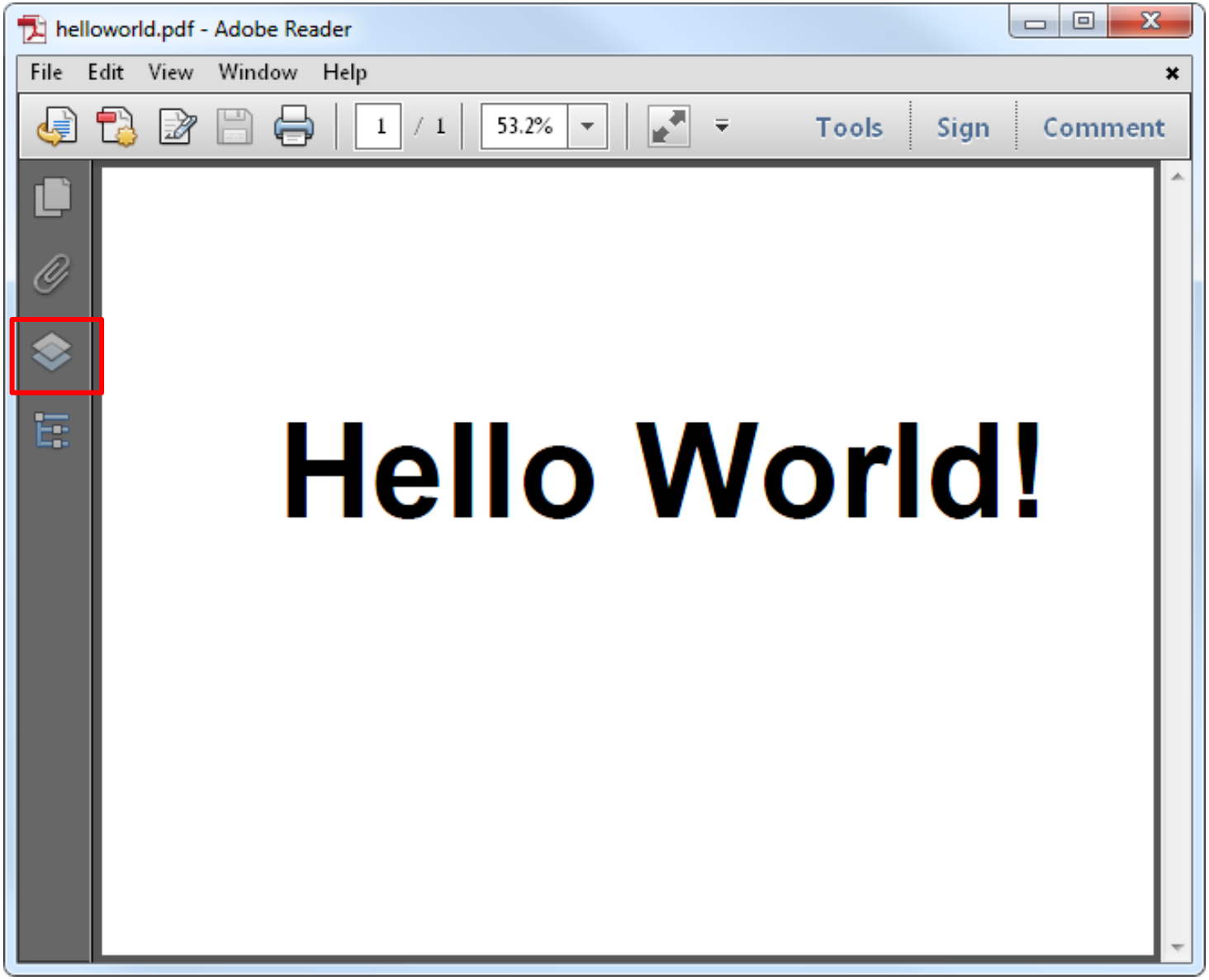
Also available in PDF/A flavor
(OK for Adobe Reader, but not for Preflight)

sometimes,
it’s in the specs...

...but who knows all of them ?
(obscurity via over-specification)

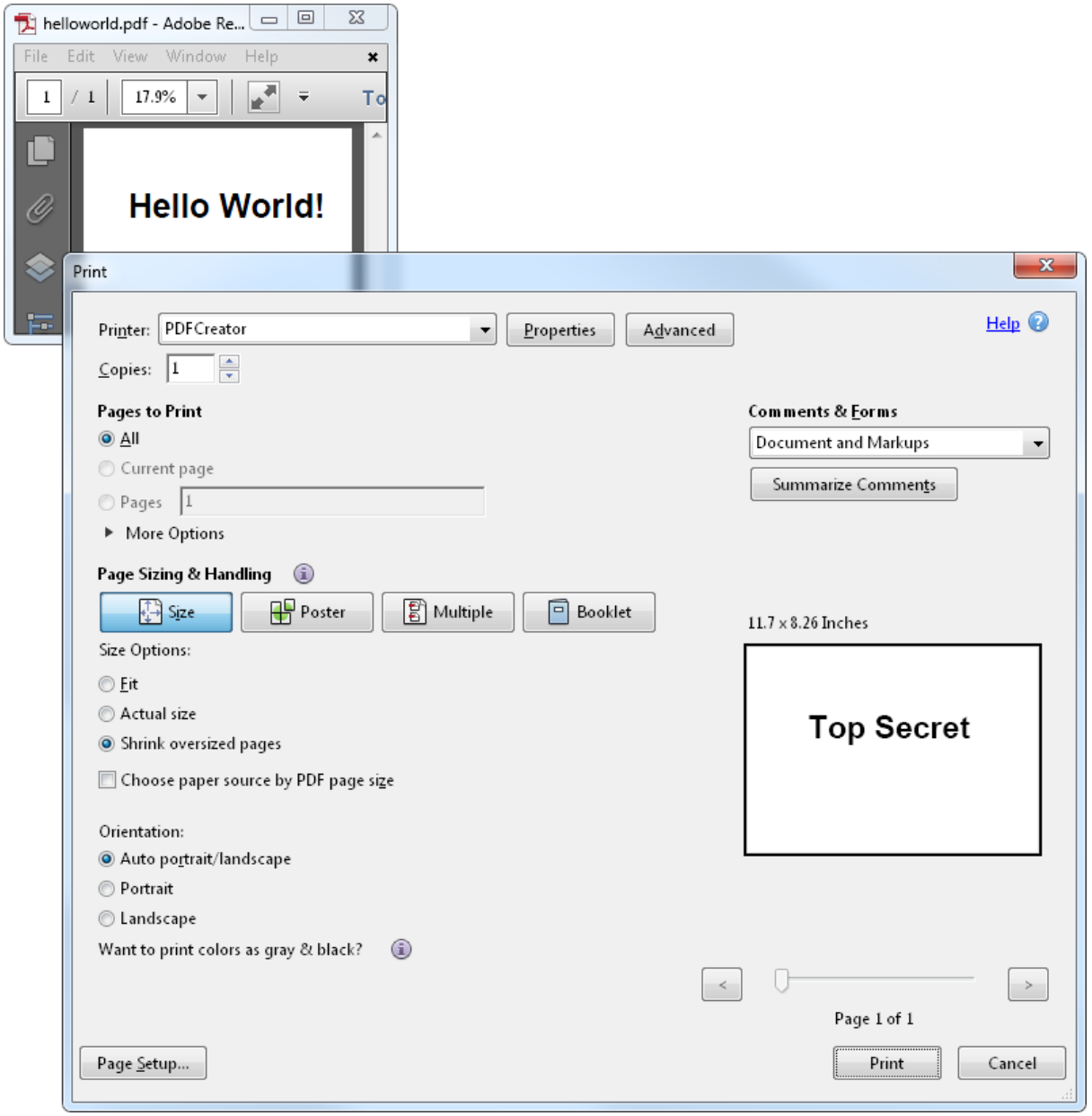


Notice anything unusual?



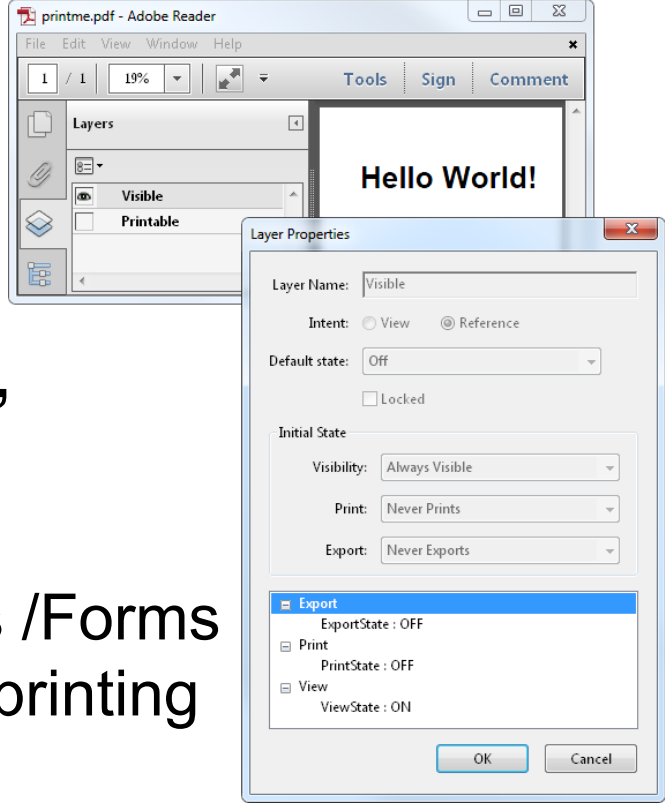
This document contains layers.

(an advanced feature)



What you see is not what you'll get...

PDF Layers 1/2



“Optional Content Configuration”

- principles
  - define layered content via various /Forms
  - enable/disable layers on viewing/printing
- no warning when printing

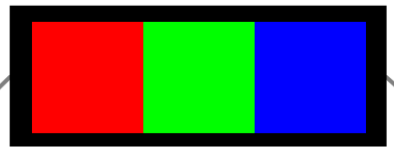
- “you can see the preview!”
  - bypass preview by keeping page 1 unchanged
  - just do a minor change in the file

PDF Layers 2/2

- it’s Adobe only
  - what’s displayed varies with readers
  - could be hidden via previous schizophrenic trick
- it was in the specs all along
  - very rarely used
  - can be abused with no warning

BITMAP / DEVICE INDEPENDENT BITMAP

SUBTYPE — TYPE



```
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: .B .M 42 00 00 00              36 00 00 00 28 00
10: 00 00 03 00 00 00 01 00 00 00 01 00 18 00 00 00
20: 00 00 0C 00 00 00
30:                  00 00 FF 00 FF 00 FF 00 00 00
40: 00 00
```

MINI.BMP

FILE HEADER

IDENTIFY AS A BMP TYPE

FIELDS	VALUES
signature	BM
file size	0x42
data start	0x36 →

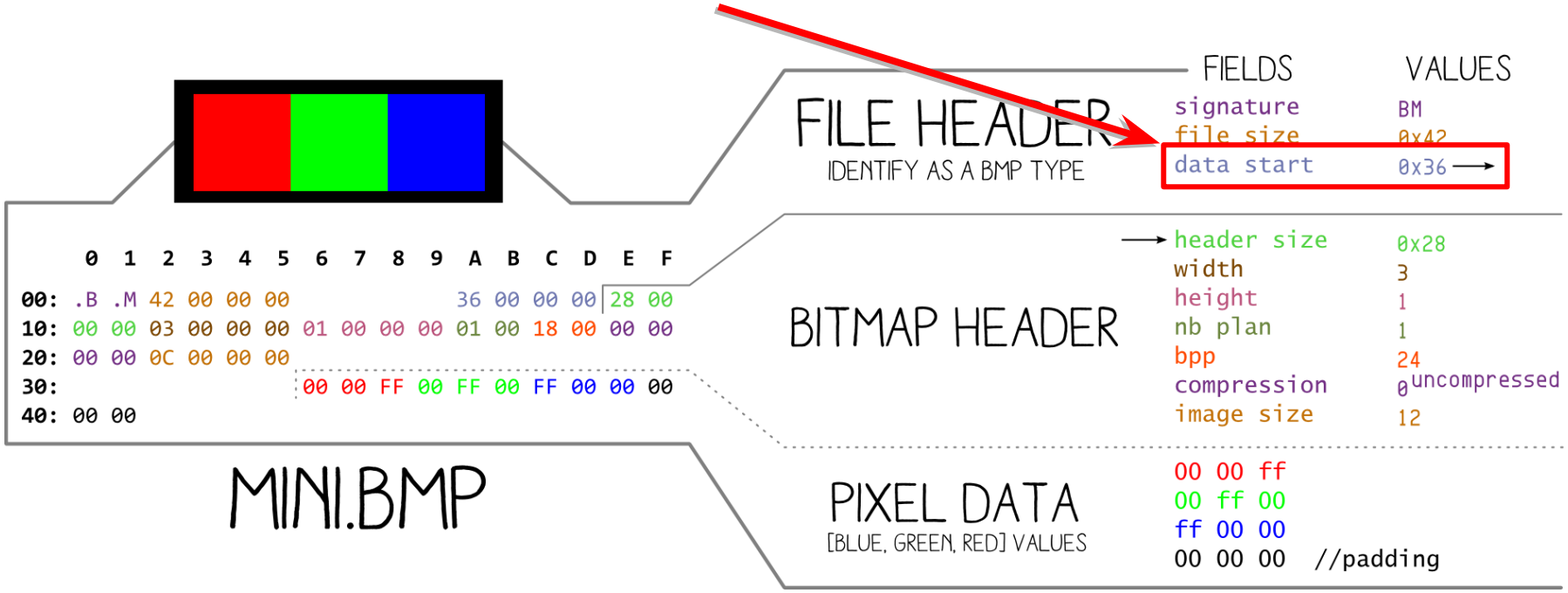
BITMAP HEADER

→ header size	0x28
width	3
height	1
nb plan	1
bpp	24
compression	0 uncompressed
image size	12

PIXEL DATA

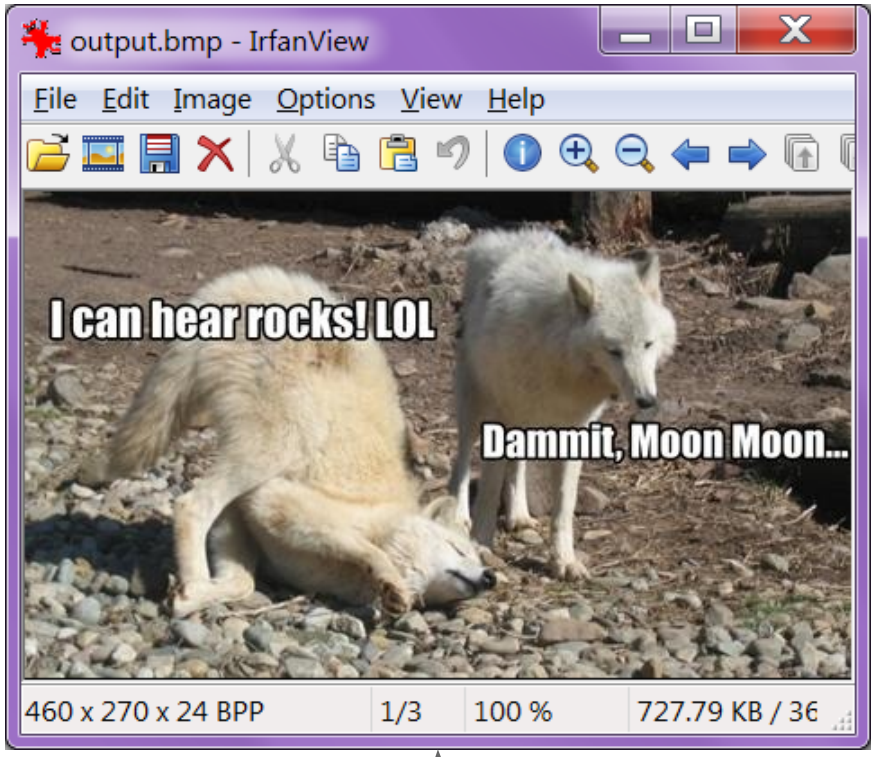
[BLUE, GREEN, RED] VALUES

```
00 00 ff
00 ff 00
ff 00 00
00 00 00  //padding
```



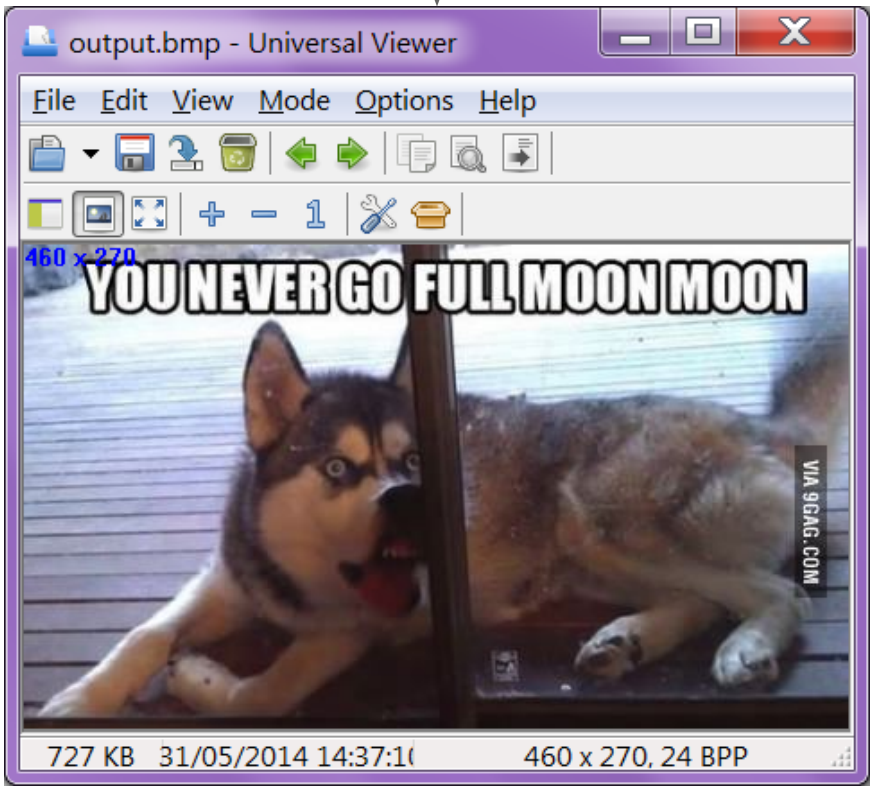
A pointer to some information that usually comes next…

What could go wrong...



getting data via the pointer (standard)

getting data right after the header

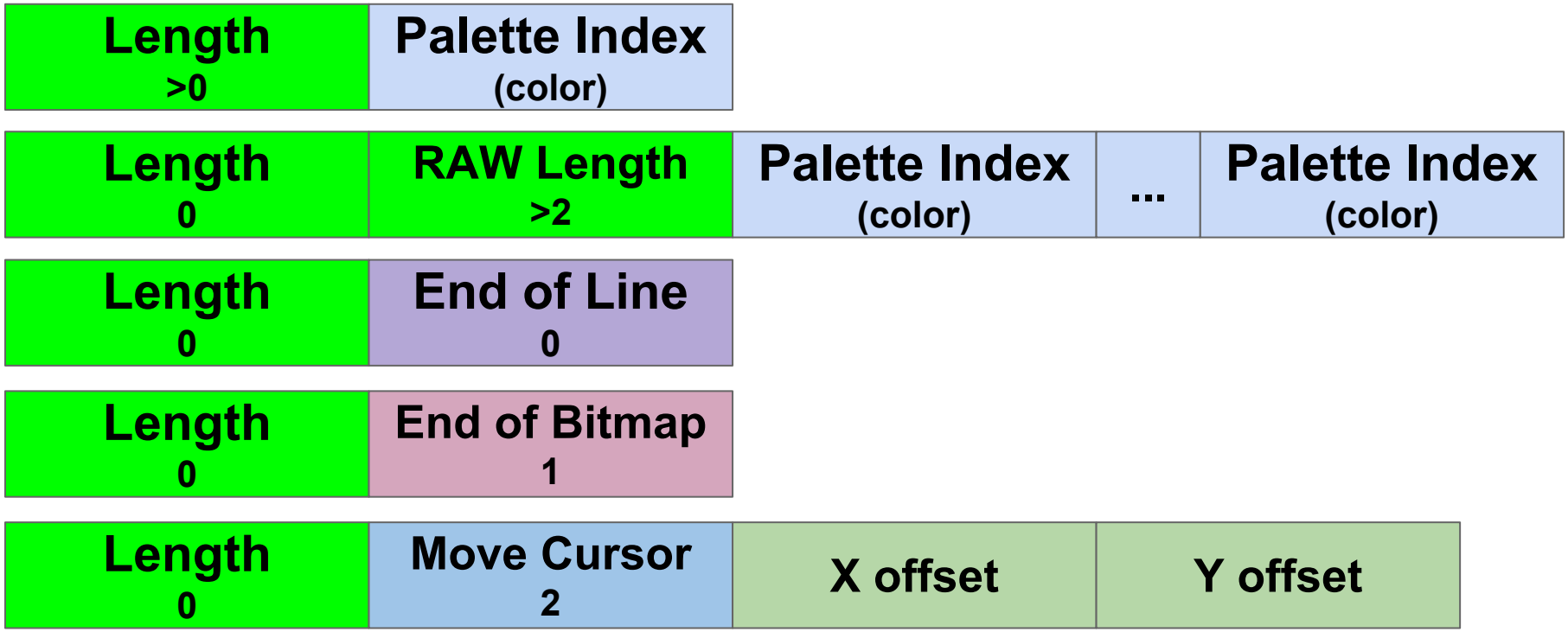


BMP Trick #1: ignoring the data pointer

Trick #2:
Run-Length Encoding

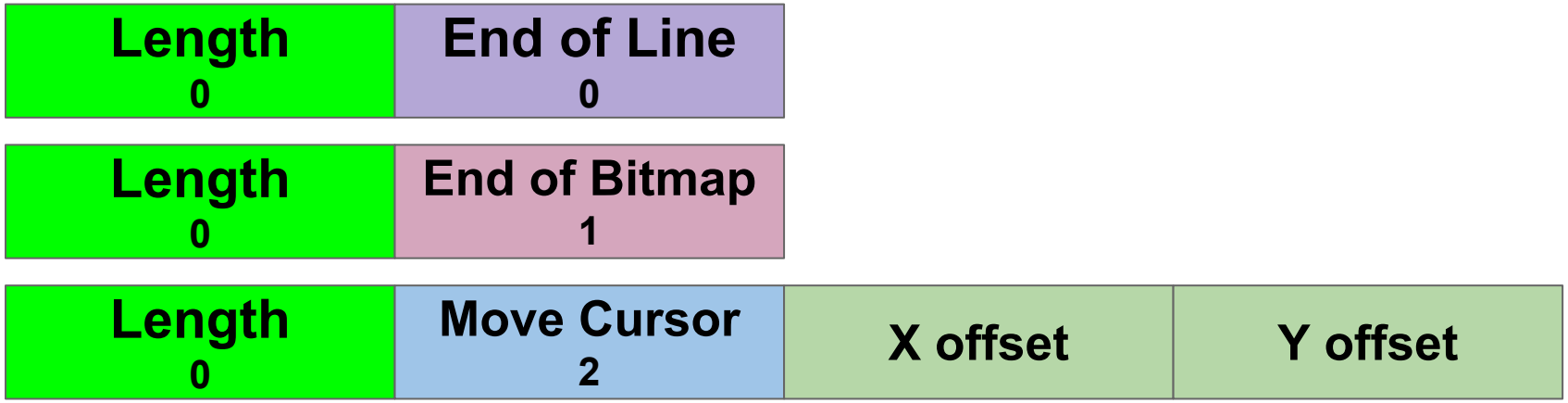
BMP RLE trick

RLE structure (each box is **1** byte)



BMP RLE trick

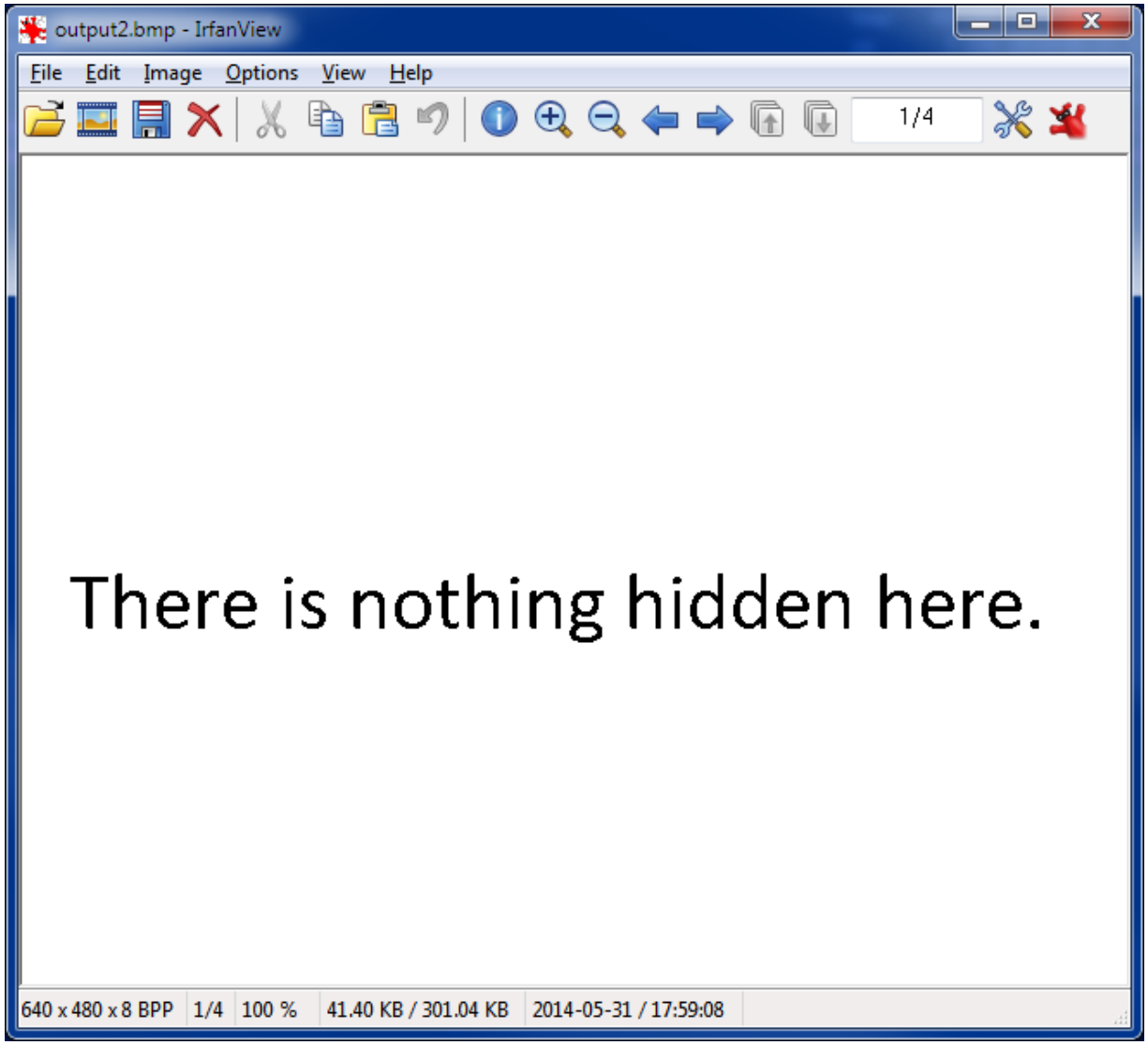
If you just skip pixels, what is their color?



Option 1

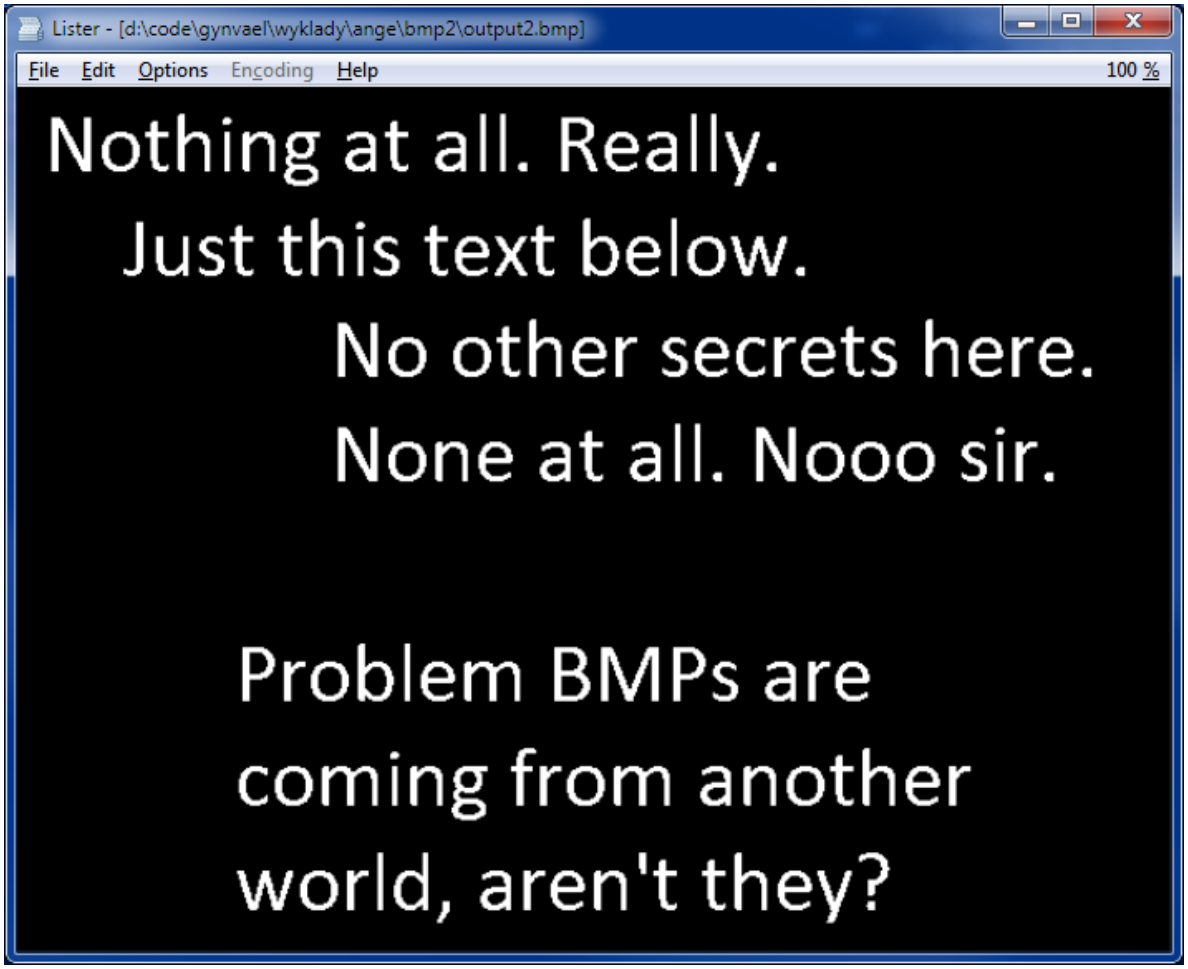
The missing data will be filled with **background color**.

(palette index 0)



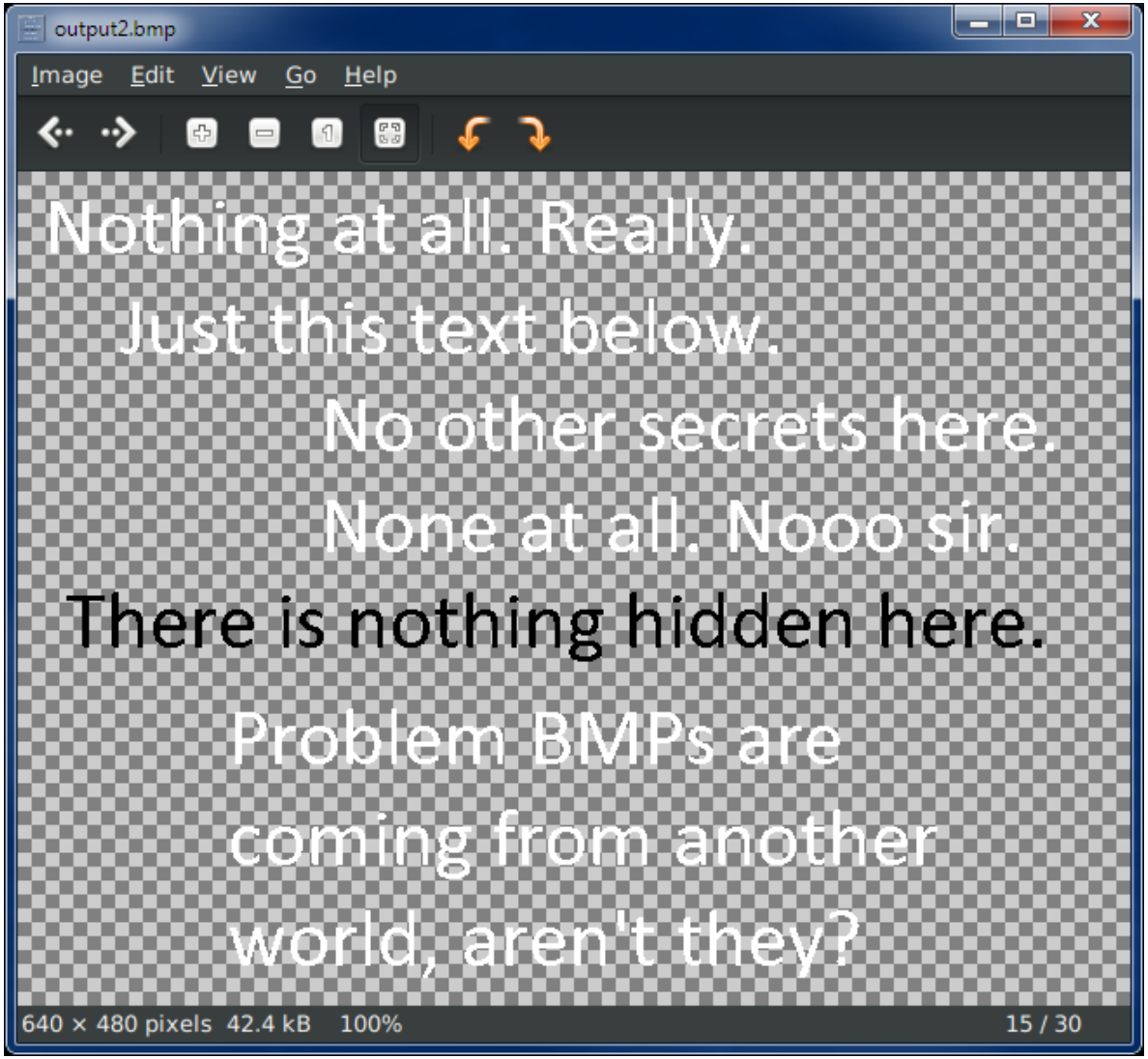
Option 2

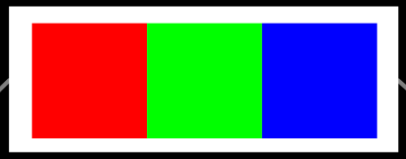
The missing data will be **black**.



Option 3

The missing data will be **transparent**.





```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 89 .P .N .G 0D 0A 1A 0A 00 00 00 0D .I .H .D .R
10: 00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83
20: E3 00 00 00 15 .I .D .A .T 08 1D 01 0A 00 F5 FF
30: 00 FF 00 00 00 FF 00 00 00 FF 0E FB 02 FE E9 32
40: 61 E5 00 00 00 00 .I .E .N .D AE 42 60 82
```

	FIELDS	VALUES
SIGNATURE	signature	\x89 PNG \r\n \x1a \n
HEADER	size	0x0000000D
	id	IHDR
	width	0x00000003
	height	0x00000001
	bpp	0x08
	color	0x02 RGB
	compression	0x00 DEFLATE
	filter	0x00
	interlace	0x00
	CRC32	0x948283E3
DATA	size	0x00000015
	id	IDAT
ZLIB	window size	0b00001000
	method	0b00001000 DEFLATE
	level / dict.	0b00011101
	checksum	0x081D % 31 = 0
DEFLATE	last block	0b00000001 FINAL
	block type	0b00000001 RAW
	data length	0x000A
	!length	0xFFF5
PIXELS	line filter	0x00 NONE
	FF 00 00 00 FF 00 00 00 FF	
	adler32	0x0EFB02FE
	CRC32	0xE93261E5
END	size	0x00000000
	id	IEND
	CRC32	0xAE426082

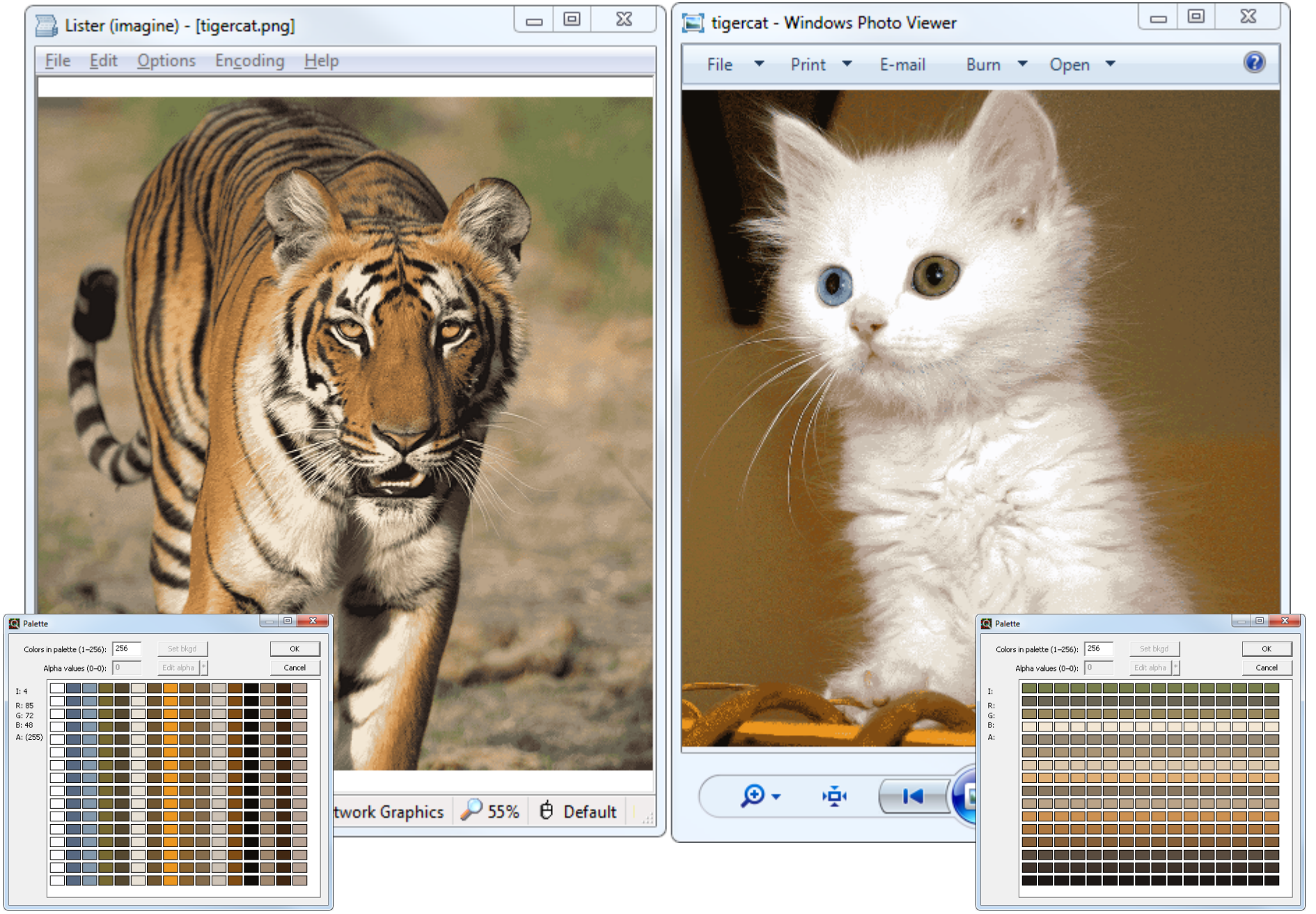
***P**ortable **N**etwork **G**raphics*

Combined data + 2 palettes

Same data chunk combining 2 images via 2 palettes

cute PoC by **@reversity**

> “There shall not be more than one PLTE chunk”



Different images depending on which PLTE chunk is used

PORTABLE EXECUTABLE

```
D:\>mini.exe

D:\>echo %errorlevel%
42
```

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000: .M .Z
030:                                    40 00 00 00
040: .P .E 00 00 4C 01
050:                  02 00 0B 01
060:                        40 01 00 00
070:            00 00 40 00 01 00 00 00 01 00 00 00
080:                        04 00
090: 60 01 00 00 40 01 00 00            03 00
140: B8 2A 00 00 00 C3
```

MINI.EXE

DOS HEADER
IT'S A BINARY

PE HEADER
IT'S A 'MODERN' BINARY

OPTIONAL HEADER
EXECUTABLE INFORMATION

FIELDS	VALUES
e_magic	MZ
e_lfanew	0x40 → PE Header
Signature	PE\0\0
Machine	0x14C [intel 386]
Characteristics	2 [executable]
Magic	0x10B [32b]
AddressOfEntryPoint	0x140
ImageBase	0x400000
SectionAlignment	1
FileAlignment	1
MajorSubsystemVersion	4 [NT 4 or later]
SizeOfImage	0x160
SizeOfHeaders	0x140
Subsystem	3 [CLI]

CODE

X86 ASSEMBLY	EQUIVALENT C CODE
mov eax, 42	return 42;
retn	

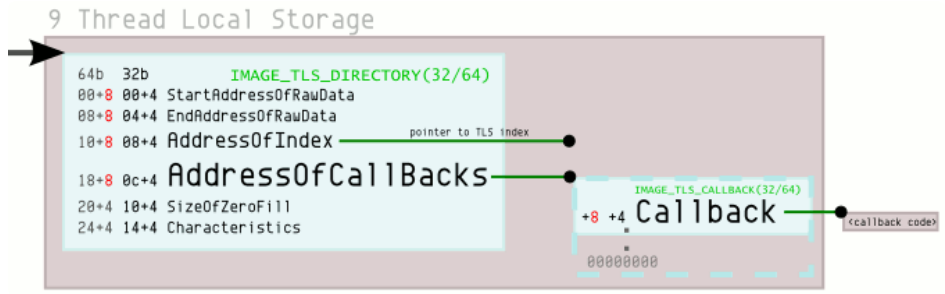
the PE Loader

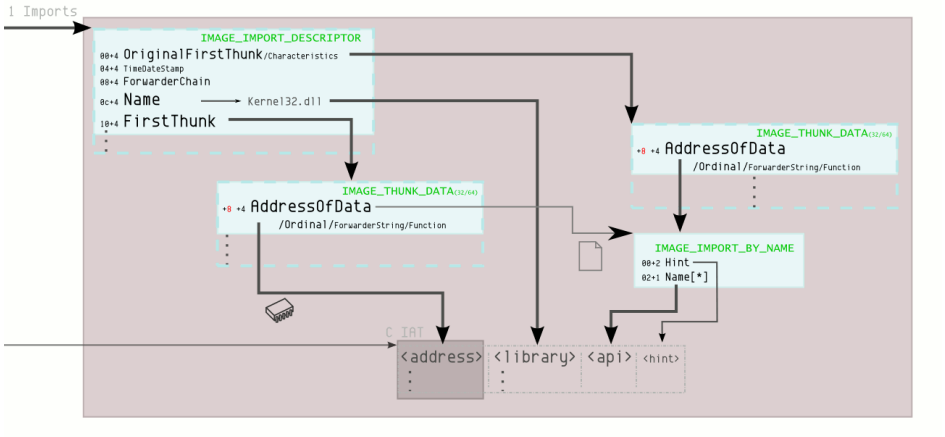
PE = complex + badly documented

- fail or fool external tools ? too easy...
- fooling Windows is much harder:
  - Windows' loader usually closes holes
    ⇒ older PEs just not working anymore

PE Trick #1

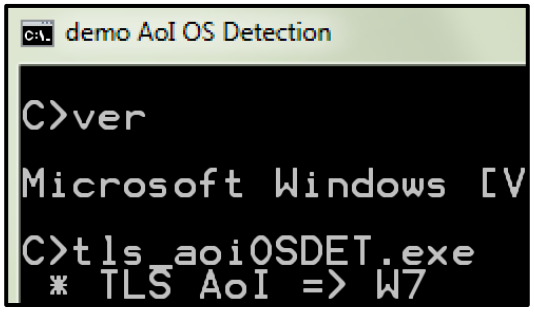
Data directory loading order





TLS
AddressOfIndex → Imports
descriptor #1
descriptor #2
descriptor #3

Pointing TLS' AddressOfIndex to an Import descriptor



TLS AddressOfIndex ⟶

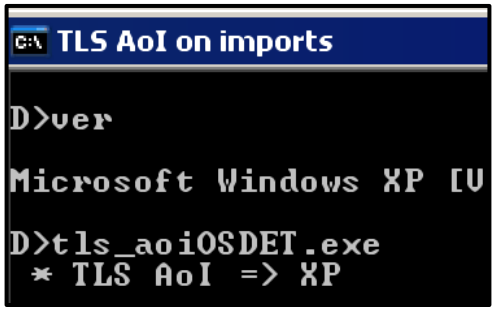
Imports descriptor #1

descriptor #3

loaded first

W7: TLS is loaded first ⇒ AoI's address set to 0

⇒ Imports descriptors's sequence is truncated before loading



TLS
AddressOfIndex

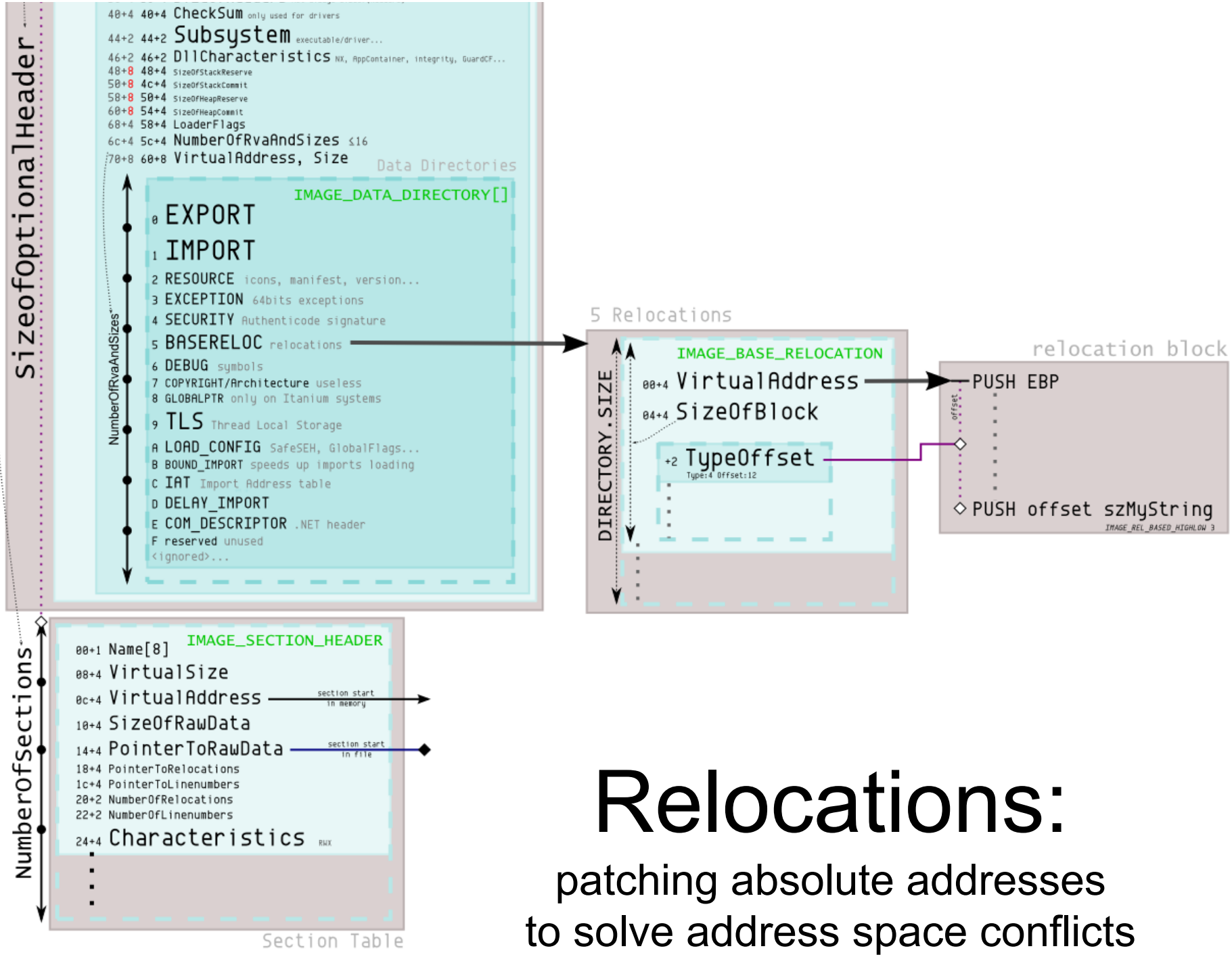
Imports
descriptor #1
descriptor #2
descriptor #3

loaded first

XP: Imports are loaded first - all descriptors are parsed
TLS is then parsed - descriptors are not relevant anymore

PE Trick #2

Relocations



Relocations:

patching absolute addresses
to solve address space conflicts

Relocations types



Type 4
HIGH_ADJ

--

--

Type 9
MIPS_JMPADDR16
IA64_IMM64
MACHINE_SPEC_9

32 bit

64 bit

as seen in PoC||GTFO #1

Relocations on relocations

Type 4
HIGH_ADJ

--

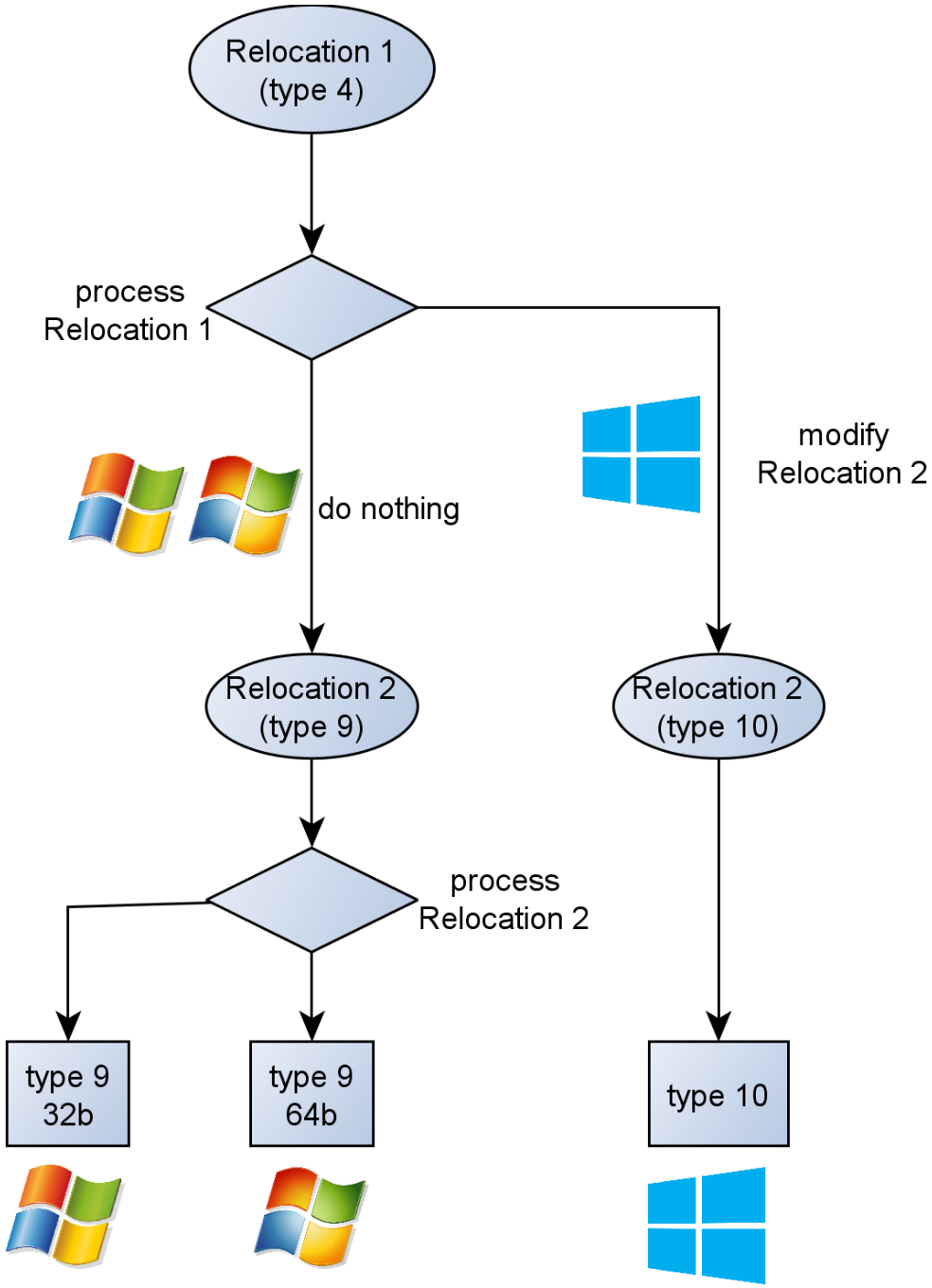
--

Type 9
MIPS_JMPADDR16
IA64_IMM64
MACHINE_SPEC_9

32 bit

64 bit

Type 10
DIR64



Relocation-based PE Schizophren

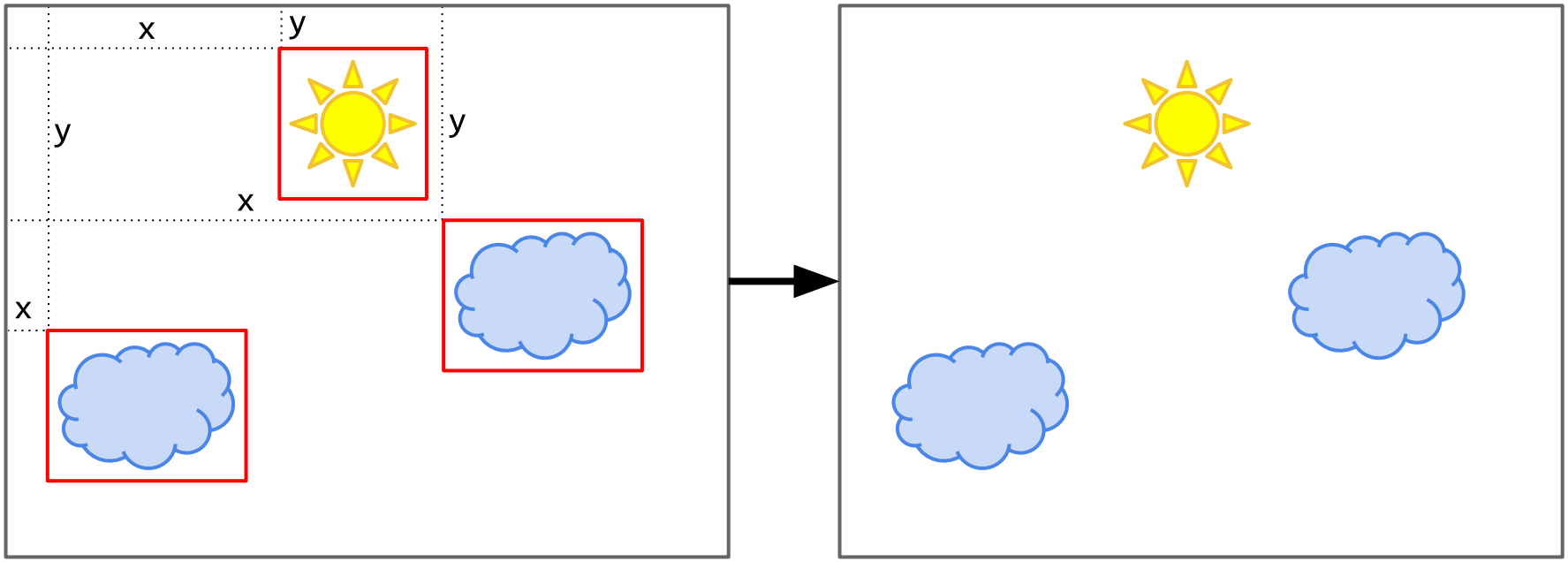
Julian Bangert, Sergey Bratus -- ELF Eccentricities
https://www.youtube.com/watch?v=4LU6N6THh2U

GIF

GIF

A GIF is made of blocks.

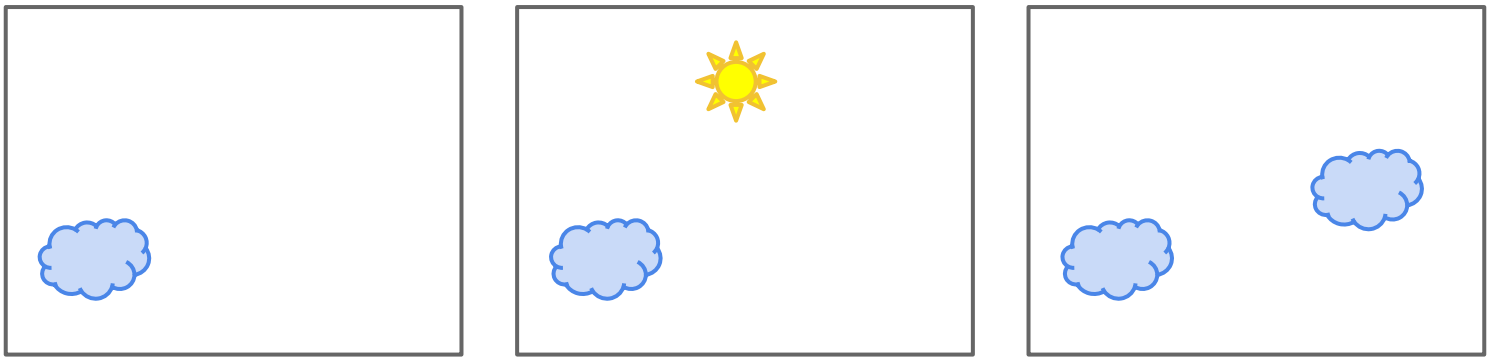
if no animation speed is defined,
they should all be displayed at once.



GIF

If a frame speed is defined, then:
first block = background

next blocks = animation frames



Background
(from block 1)

Frame 1
(with block 2)

Frame 2
(with block 3)

GIF

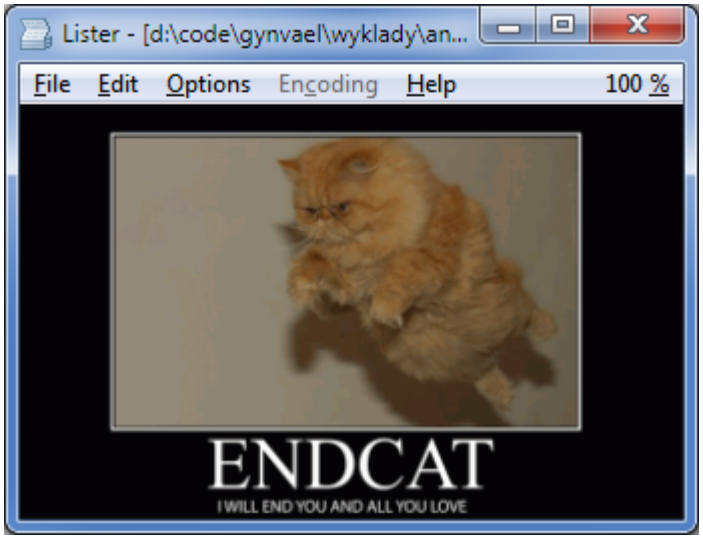
Frame 1

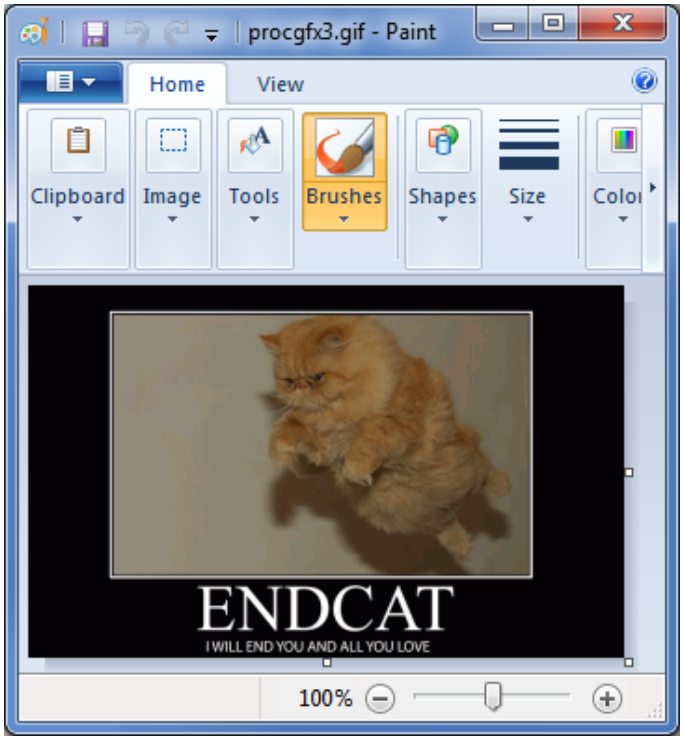
Frames 2-10001
1x1 px

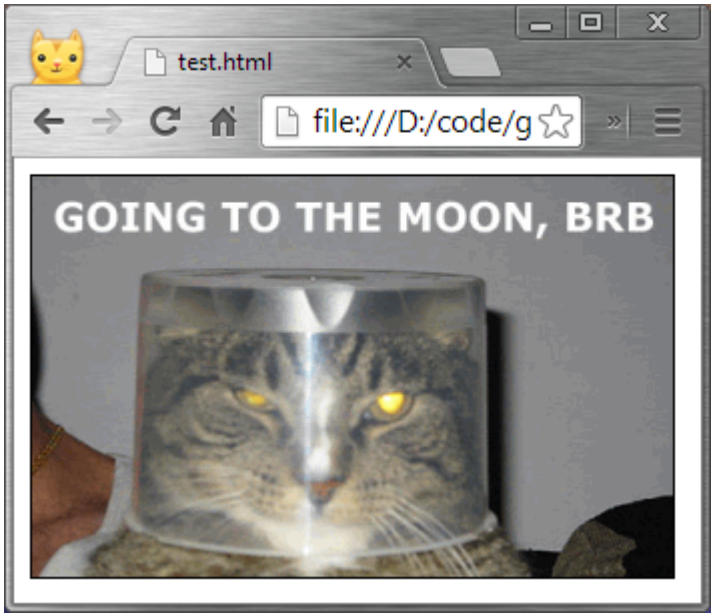
Frame 10002

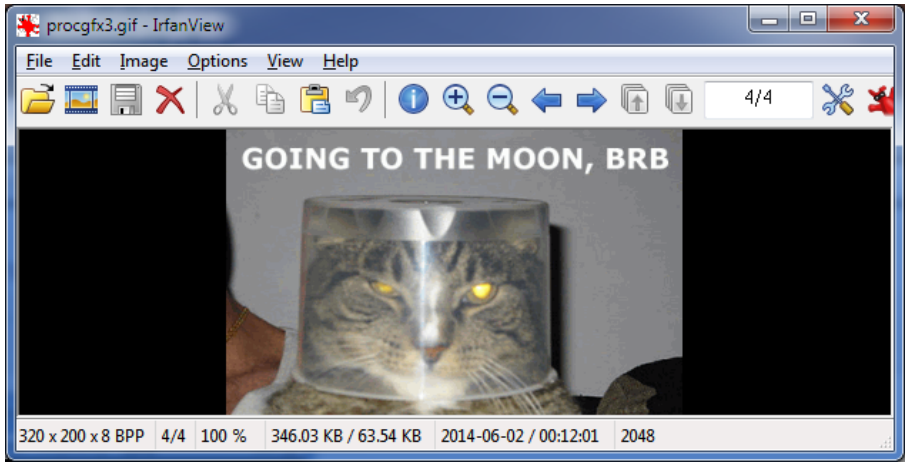
1 complete pic + 10.000 pixels + 1 complete pic

Displaying all blocks at once (standard)









Forcing animation (even if no frame speed is defined)

same-tool schizophrenia

1 file + 1 tool = 2 behaviors
(in different sub-components)

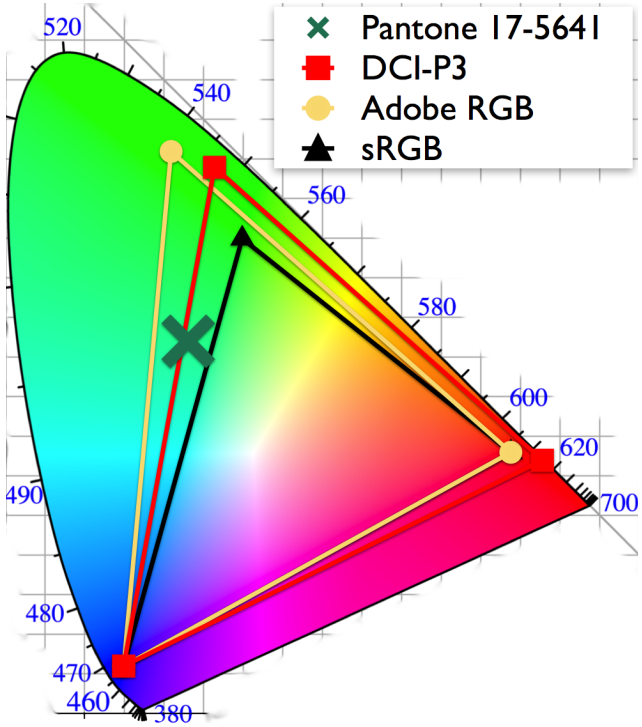
Because it was too simple...

- WinRar: viewing ⇔ extracting
  - opening/failing
  - opening/’nothing’
- Adobe: viewing ⇔ printing
  - well, it’s a *feature*

Failures & Ideas

Failures & Ideas

- screen ⇔ printer
  - embedded color profiles?
- JPG
  - IrfanView vs the world
- Video
  - FLV: early data pointer, like BMP
    PoC: video fails but plays sound



PNG

Various ancillary chunks (rendering level)

- partially supported:
  - gamma
  - transparency (for palettes)
- never supported?
  - significant bits
  - chromacities
- always supported?
  - physical size

Conclusion

We tend to take our own shortcuts.

Conclusion

- such a mess
  - specs are messy
    - unclear
    - historical reasons
  - parsers don’t even respect them
    (particularly when there is an easy shortcut)
  - official tools “forced” to be tolerant
    - They’re even trying to repair corrupted files (!)
- no CVE/blaming for parsing errors?
  - no security bug if no crash or exploit :(

Schizophrenia symptoms

- different parsing (seeing different data)
  - BMP: ignoring data pointer
  - ZIP: different parsing algorithm & directions
  - PE: different data directory loading order
  - PDF: different trailer parsing
- different interpretation (same data)
  - GIF: ignoring animation speed
  - BMP RLE: using different default color
  - PE: different relocations implementation
  - PNG: using different palette
  - PDF: conditional layers

ACK

@gynvael

@reversity @travisgoodspeed @sergeybratus
qkumba @internot @pdfkungfoo

@j00ru ise ds vx, Mulander
Felix Groebert, Salvation

@angealbertini

corkami.com

SCHIZO PHRENIC FILES