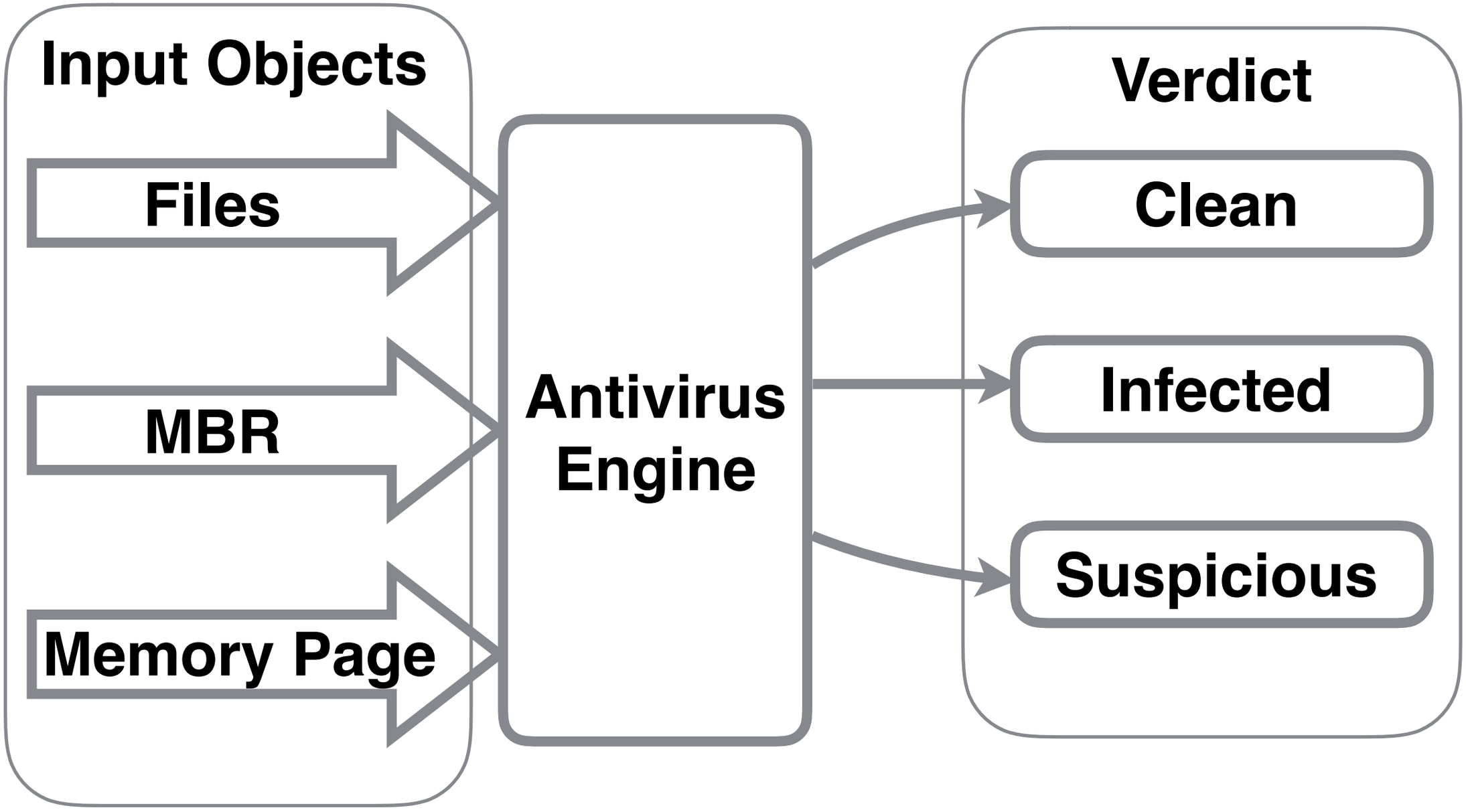
Dr.WEB®
www.drweb.com

Defend what you create

ANTIVIRUS ENGINE

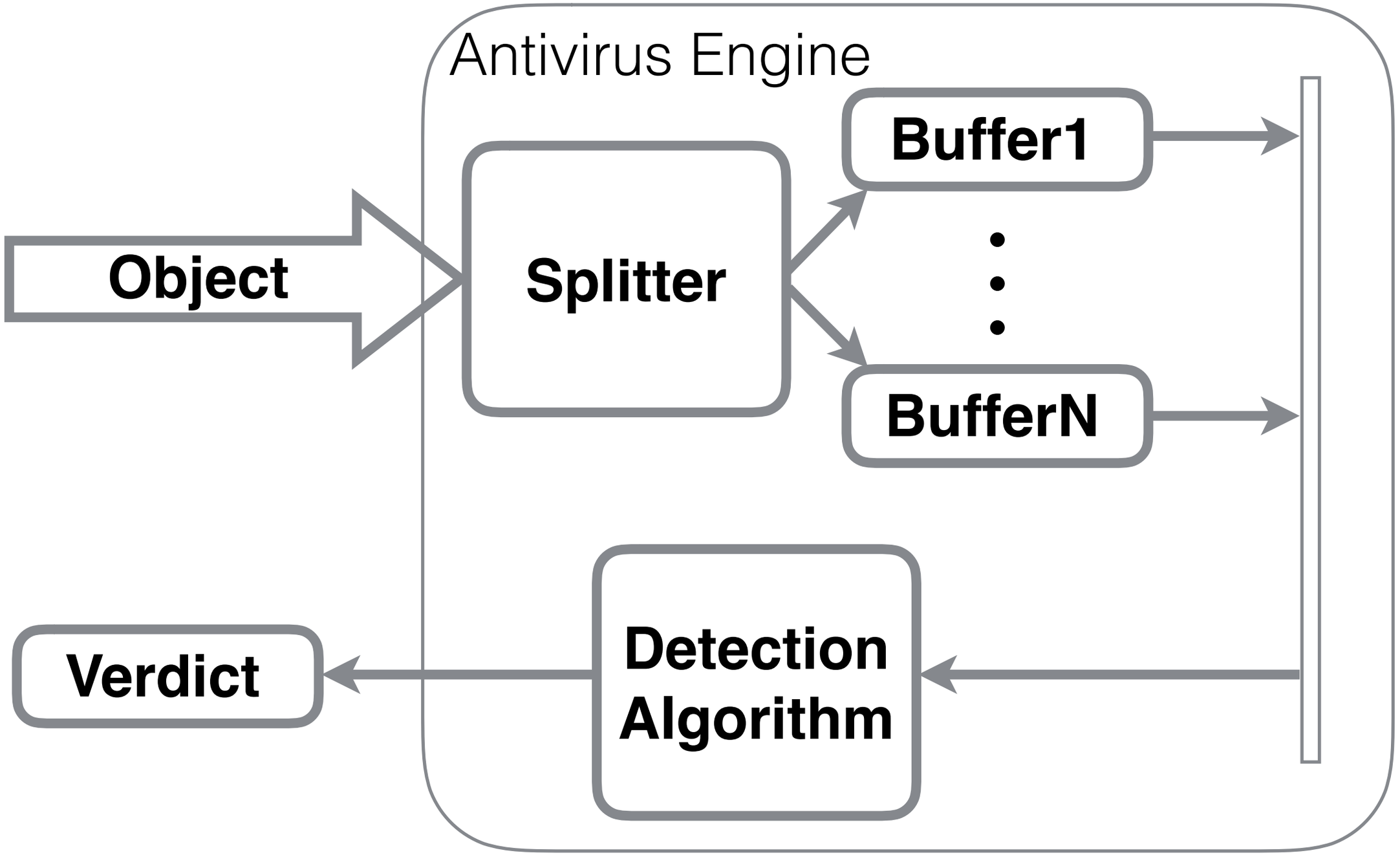
Denis Akimov <d.akimov@drweb.com>

Overview of the Engine

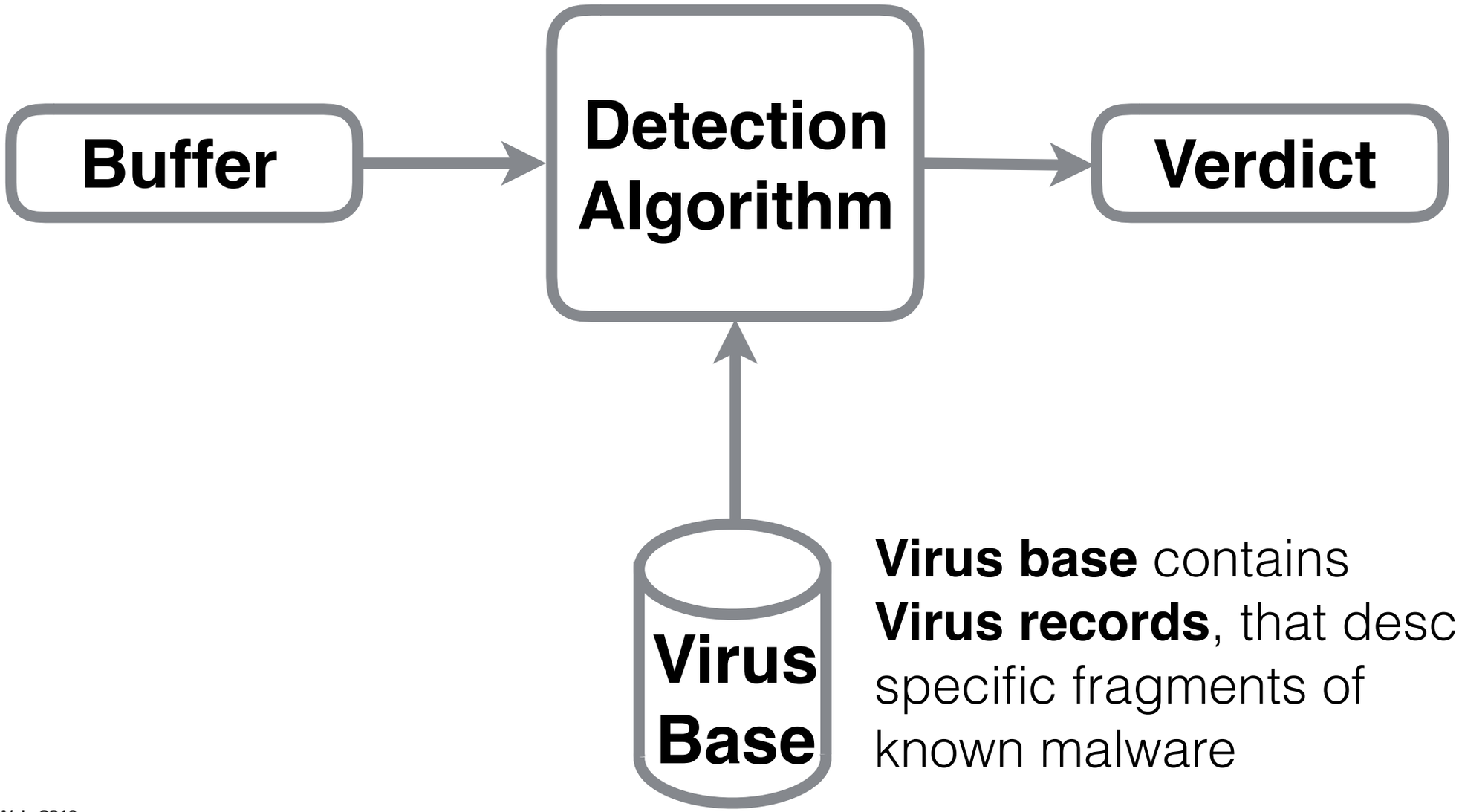


- MBR — Master Boot Record

Overview of the Engine



Detection Algorithm



Virus base contains **Virus records**, that describe specific fragments of known malware

Detection Algorithm
(signature based)

- For each *Virus Record* we see if the buffer contains data described in the *Virus Record*

- If the current buffer does contain that data, we say that the file is infected

- If all the file buffers are checked against all the *Virus records* and no match found, we say that the file is clean

Virus Record

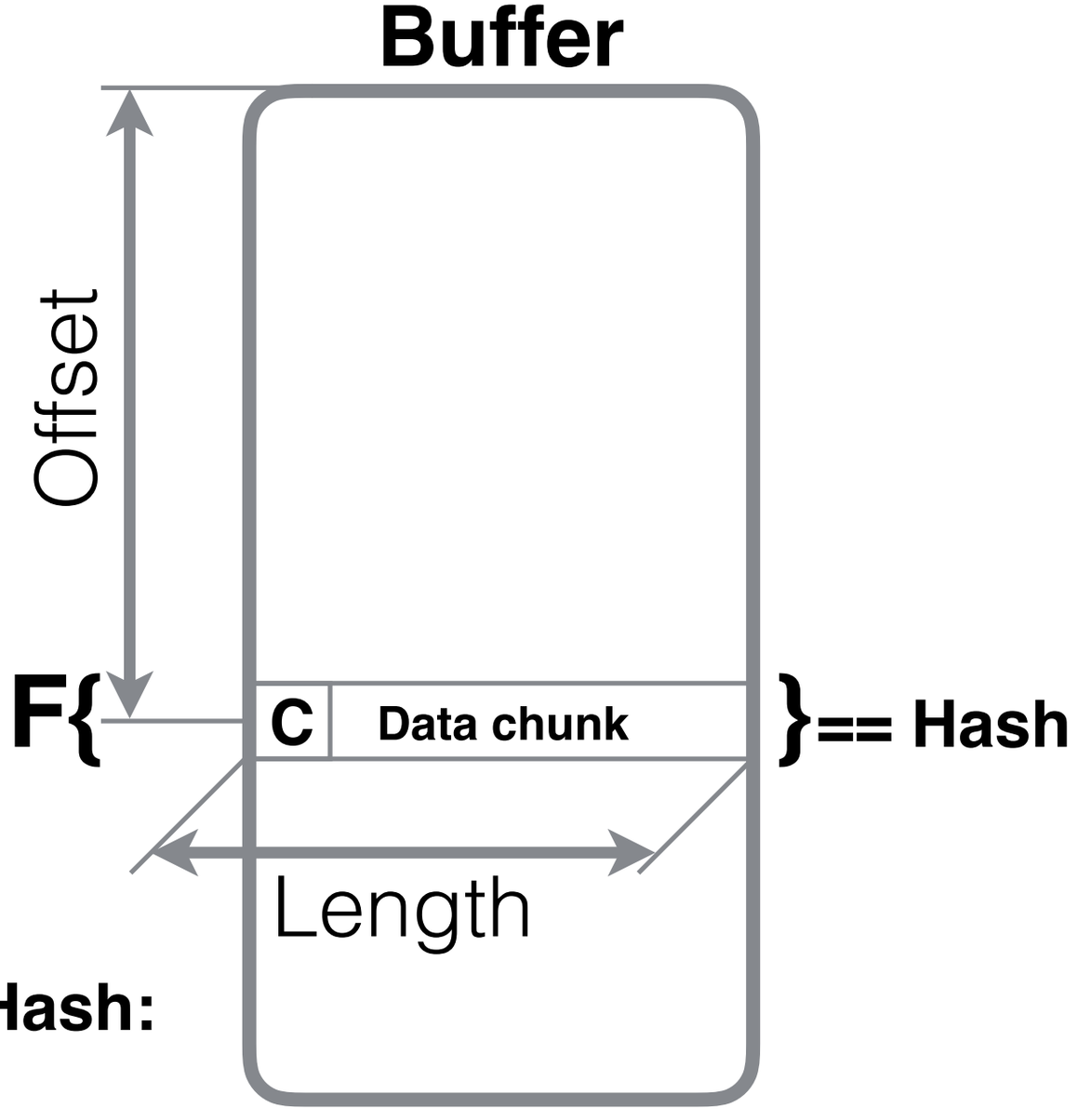
- **Virus Record is a description of how to find a certain malicious chunk of data. It describes a specific part of known malicious sample and uniquely identifies it**

- **Virus record should specify the location of the chunk and its length. Also it has to specify the chunk contents itself as a reference. But it is not optimal to put all the chunk inside the virus record. Instead, it’s better to calculate a hash of the chunk contents**

Virus Record

Virus Record

Type, ChkByte, Offset, Length,
Hash, VirusName

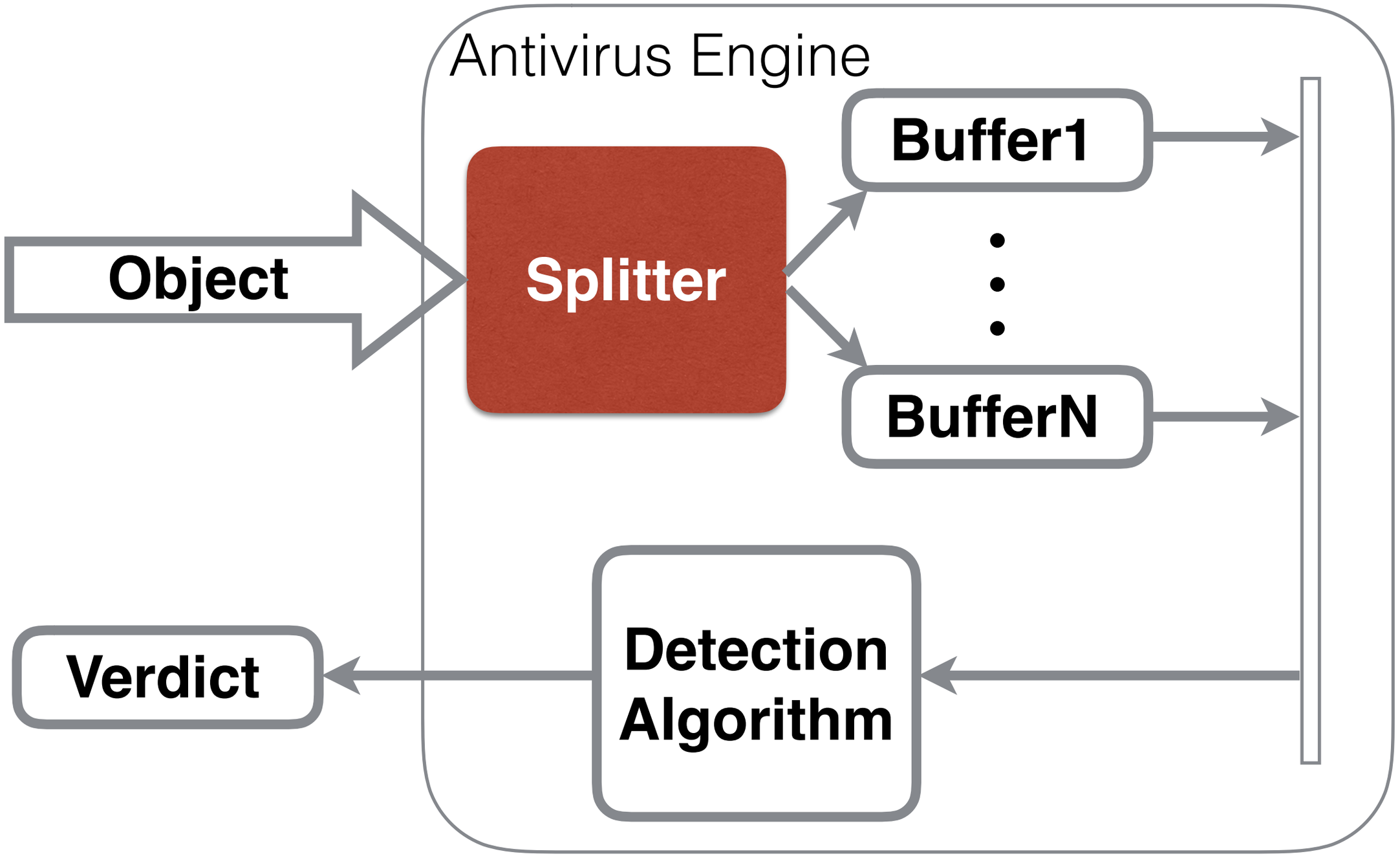


In python

- **Virus record:**
  [Type, ChkByte, Offset, Length,
  Hash, VirusName]

- **Testing Virus record:**
  if Buffer[Offset] == ChkByte and
  F(Buffer[Offset:Offset+Length] == Hash:
  return VirusName

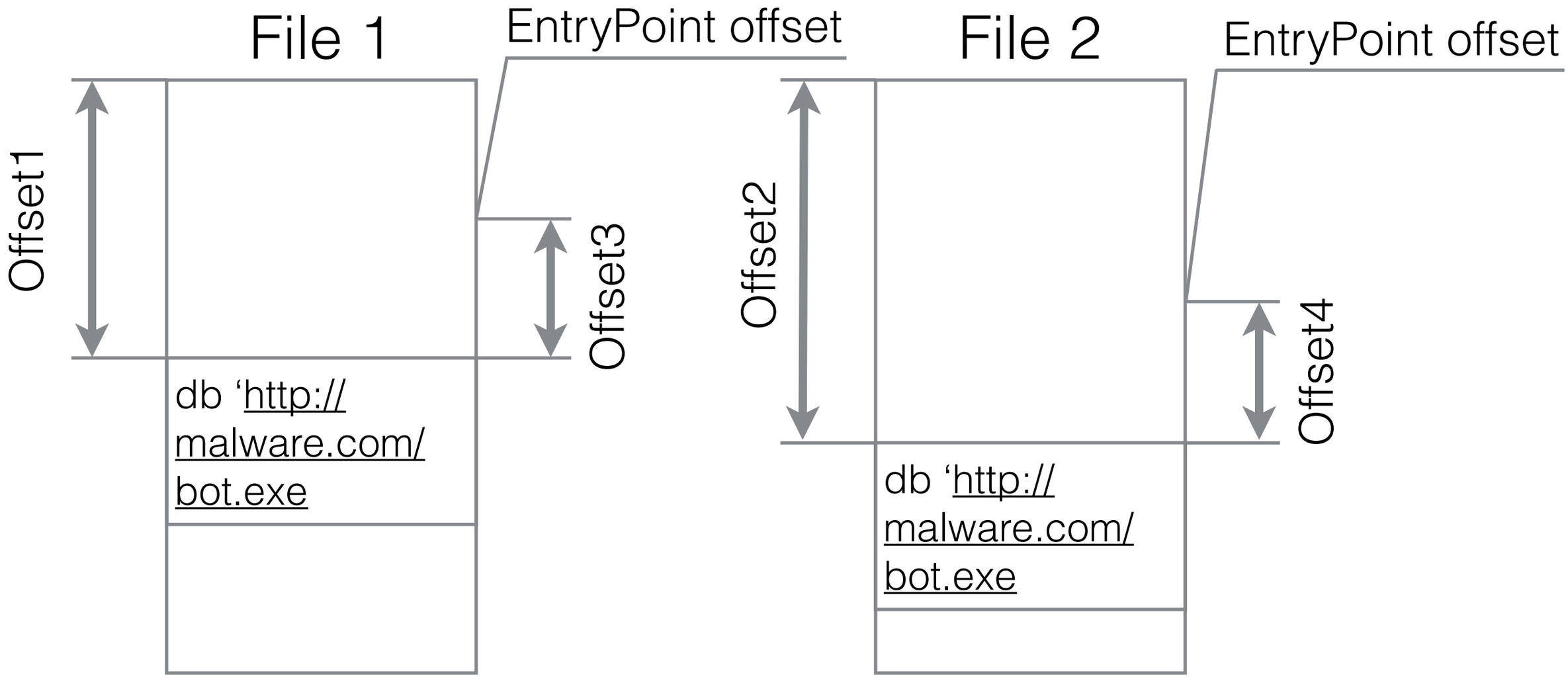
Overview of the Engine



Splitting a file into sections

- Executable Files are usually split into a number of buffers (sections)

Splitting a file into sections



Offset1 != Offset2
EP1 offset != EP2 offset
But Offset3 == Offset4

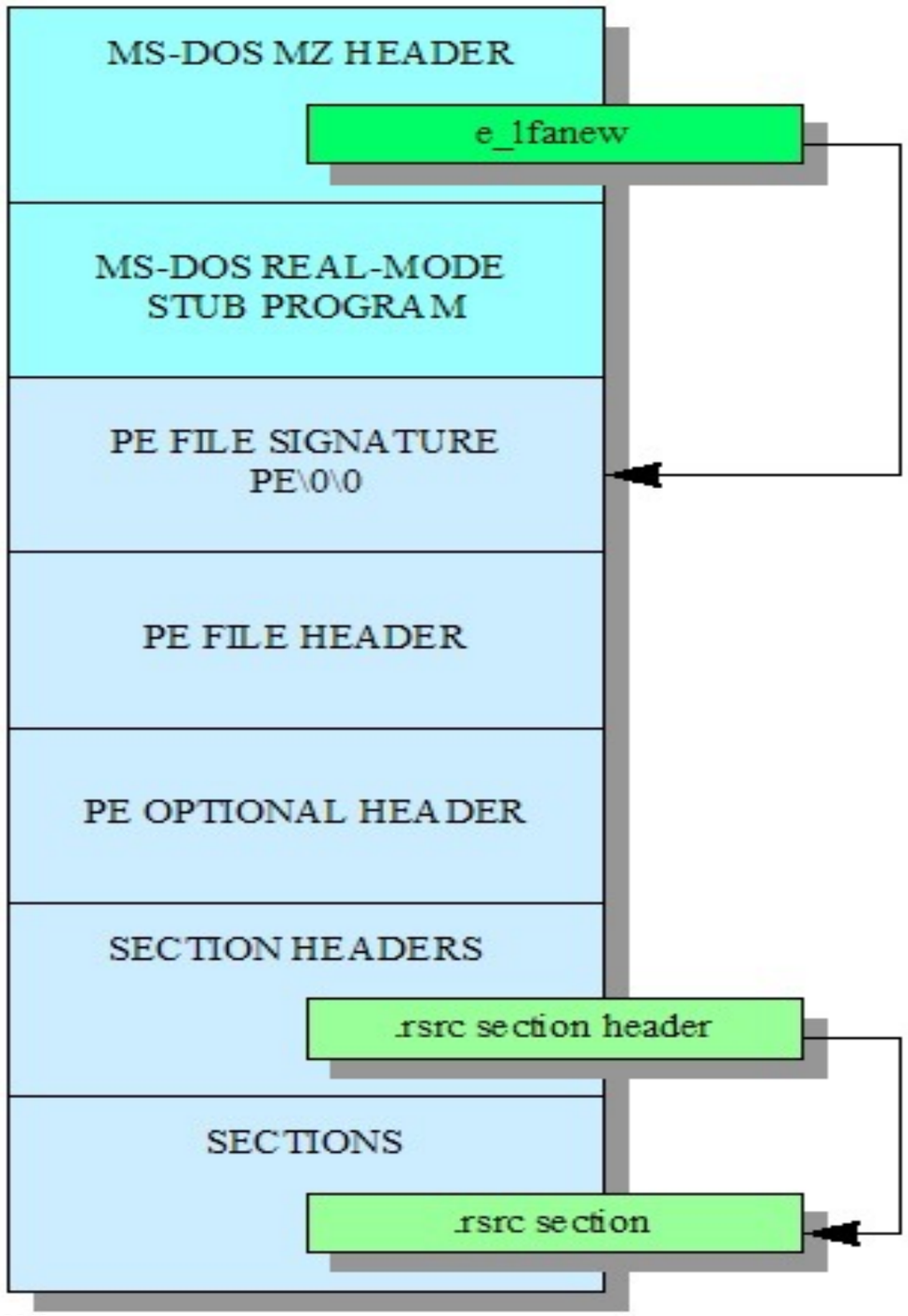
Splitting a file into sections

- **In our simple engine we will use two types of sections (or buffers):**

— «DATA» is the data buffer of the first nonempty PE section

— «EASY» is the data buffer at offset of executable file Entry Point. Also we can emulate code here to get another buffers to check

PE File structure



File Offset - the location of an item within the file itself

VA - Virtual Address. The Address of an item after it is loaded into memory by the loader

PE File structure

- **PE file is loaded into memory at certain ImageBase**
- **Sections must be placed in memory in accordance with section table**
- **Section is described by parameters *PointerToRawData*, *SizeOfRawData*, *VirtualAddress*, *VirtualSize***
- ***RVA* = (*FileOffset* - Section.PointerToRawData) + Section.VirtualAddress,**
  where *FileOffset* is the file offset of an item in a certain section.
  To find the section to which *FileOffset* belongs we iterate over sections and find one which satisfies relation:
  Section.PointerToRawData <= *FileOffset* < Section.PointerToRawData + Section.SizeOfRawData
- ***VA* = *RVA* + *ImageBase***
- ***FileOffset* = (*RVA* - Section.VirtualAddress) + Section.PointerToRawData,**
  Section must be chosen to satisfy relation:
  Section.VirtualAddress <= *RVA* < Section.VirtualAddress + min(Section.VirtualSize, Section.SizeOfRawData)

AvEngine

- **avengine.py — main driver that parses command line and scans files**

- **pefile.py — contains helper class *PeFileParser* to work with PE files**

- **utils.py — functions to print colored text and a convenient function to print hexdump of a buffer**

AvEngine template

```
bash-3.2$ python avengine.py -h
usage: avengine.py [-h] [-m] [-v VIRBASEDIR] [dirs [dirs ...]]

Simple Antivirus Engine

positional arguments:
  dirs                  Directories to scan

optional arguments:
  -h, --help            show this help message and exit
  -m, --memoryscan      Force memory scanning
  -v VIRBASEDIR, --virbasedir VIRBASEDIR
                        Directory where virus bases located
```

AvEngine template

```
bash-3.2$ python avengine.py ../
loaded 0 records
Scanning files
../p29.tiff - Not a PE file
../test/.DS_Store - Not a PE file
../test/1.exe._ - Valid PE file
  * Header (fo): 000000F0
  *  Data (rva): 00001000 Data bytes:
    00000000: 9A 18 DD 77 EA 22 DD 77 D7 23 DD 77 3D 7E DD 77 ...w.".w.#.w=~.w
    00000010: 6B 1A DD 77 00 00 00 00 78 F0 95 71 00 00 00 00 k..w....x..q....
    00000020: 1D 51 C7 77 31 75 C7 77 46 F9 C7 77 BD 81 C8 77 .Q.w1u.wF..w...w
    00000030: C0 6B C7 77 1C 3A C7 77 53 DF C7 77 81 2D C7 77 .k.w.:.wS..w.-.w
  *  Easy (rva): 0005EF30
../test/777.exe._ - Valid PE file
  * Header (fo): 000000E0
  *  Data (rva): 00001000 Data bytes:
    00000000: FF 1C 25 9C 40 78 19 8B C0 10 98 1C 94 8E 47 90 ..%.@x........G.
    00000010: 23 8C 91 88 C8 84 E4 80 72 7C 39 78 1C 74 8E 47 #.......r|9x.t.G
    00000020: 70 23 6C 91 68 C8 64 60 50 A1 2C 20 83 11 0C 34 p#l.h.d`P., ...4
    00000030: 30 0C E8 7E FF 02 C3 0E 90 53 8B D8 75 17 0A 83 0..~.....S..u...
  *  Easy (rva): 00001C98
```

avengine.py

- **Main class AvEngine:**
  - load virus bases
  - iterate over files
  - check each file

- **You have to add your code to a function *scanfile* that should check DATA and EASY sections using loaded Virus records**

pefile.py

- **Class *PeFileParser* - simple PE parser:**
  - check if a file is PE file
  - *get_pe_header_offset* – get PE header offset
  - *get_datasect_rva* – get rva of the first PE section with data
  - *get_ep_rva* – get entry point rva
  - *rva_to_offset* – convert rva to file offset

utils.py

- ***cprinter*** **— function for making colored output**
- ***cprinter(‘[red]Error: {}[/red]\n’, error_message)***
- ***hexdump*** **— prints hexdump of a buffer**

Task

- **Use avengine.py as a template**
- **Choose hash sum (CRC32, MD5, whatever)**
- **Choose virus record format**
- **Write scan logic to check if a file contains viruses. You can add certain code to a function *scanfile***
- **Look on samples (samples_to_detect folder) and decide how to add them to virusbase, i.e. which sections (DATA, EASY) to use**

Task (cont.)

- **BackDoor.Tinyshell contains 2 files: packed and unpacked version. You'll create two records. Expiro and Keisan should be detected by one virus record**

- **Implement the detection of Expiro infected samples, where you need to «emulate» a call instruction to get a new offset of a buffer to check**