APPLYING MEMORY FORENSICS TO ROOTKIT DETECTION

Igor Korkin   Ivan Nesterov

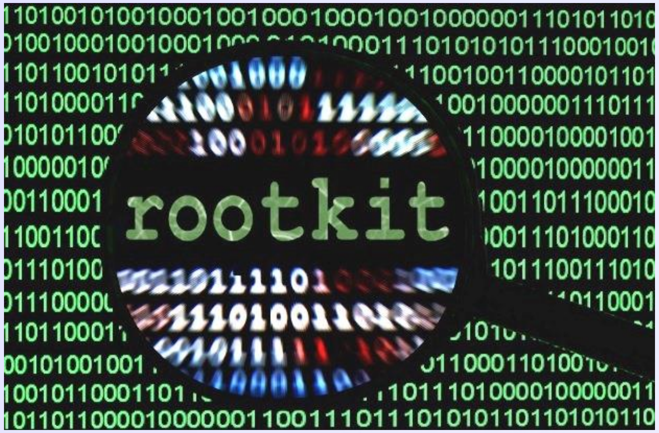
CDFSL 2014

Goals of memory forensics

Passwords, crypto keys and etc. revealing software

Software reverse engineering

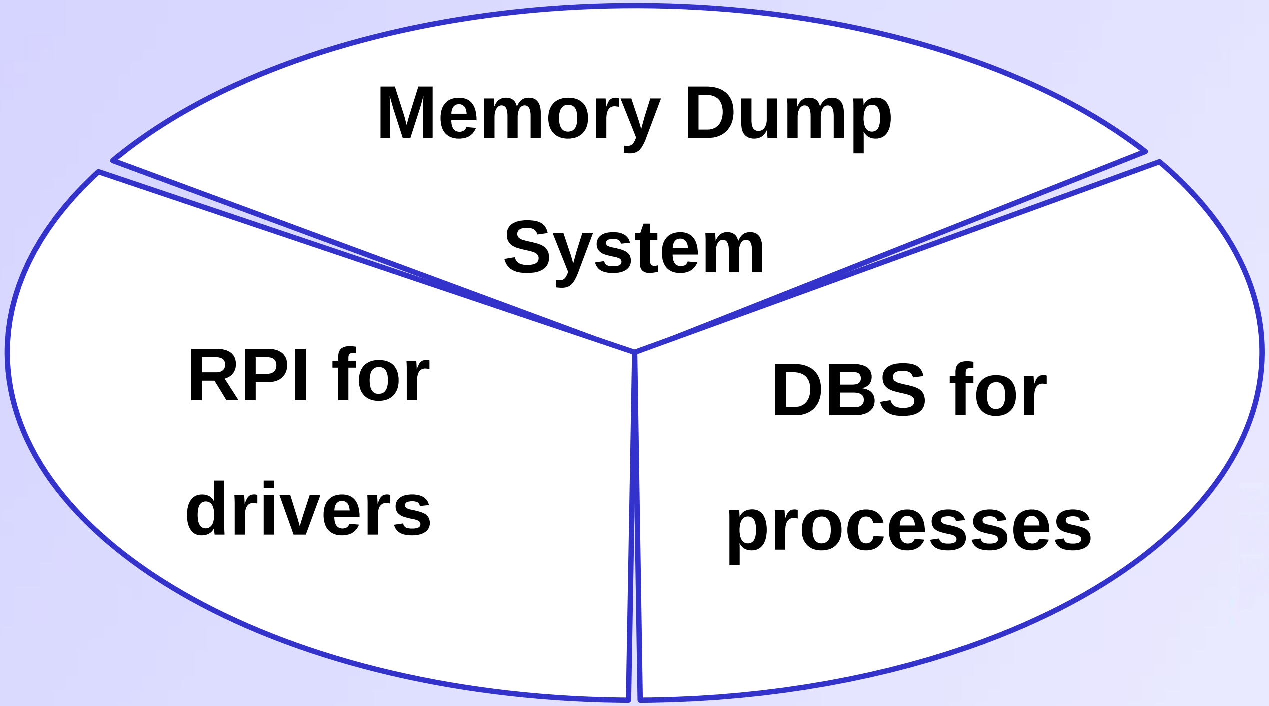
Rootkits analysis & detection



Agenda

1. Review of dump & analysis tools in rootkit conditions

2-3. MASHKA ─ Malware Analysis System for Hidden Knotty Anomalies:

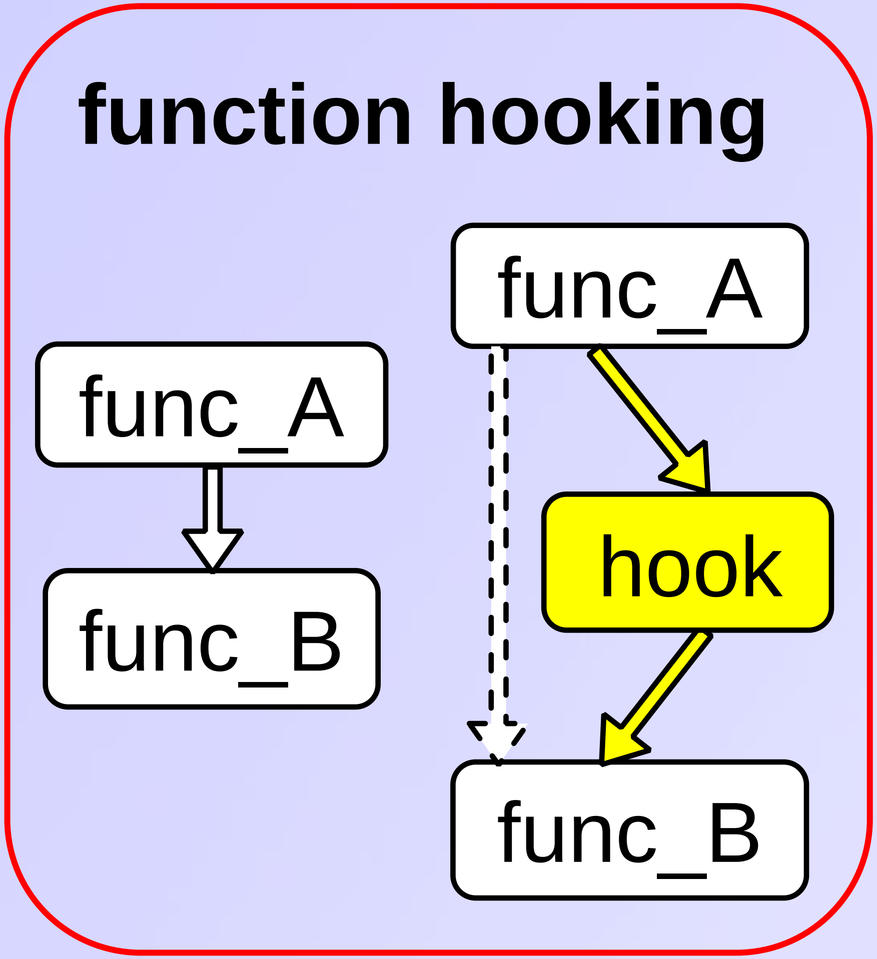

Memory Dump System

RPI for drivers

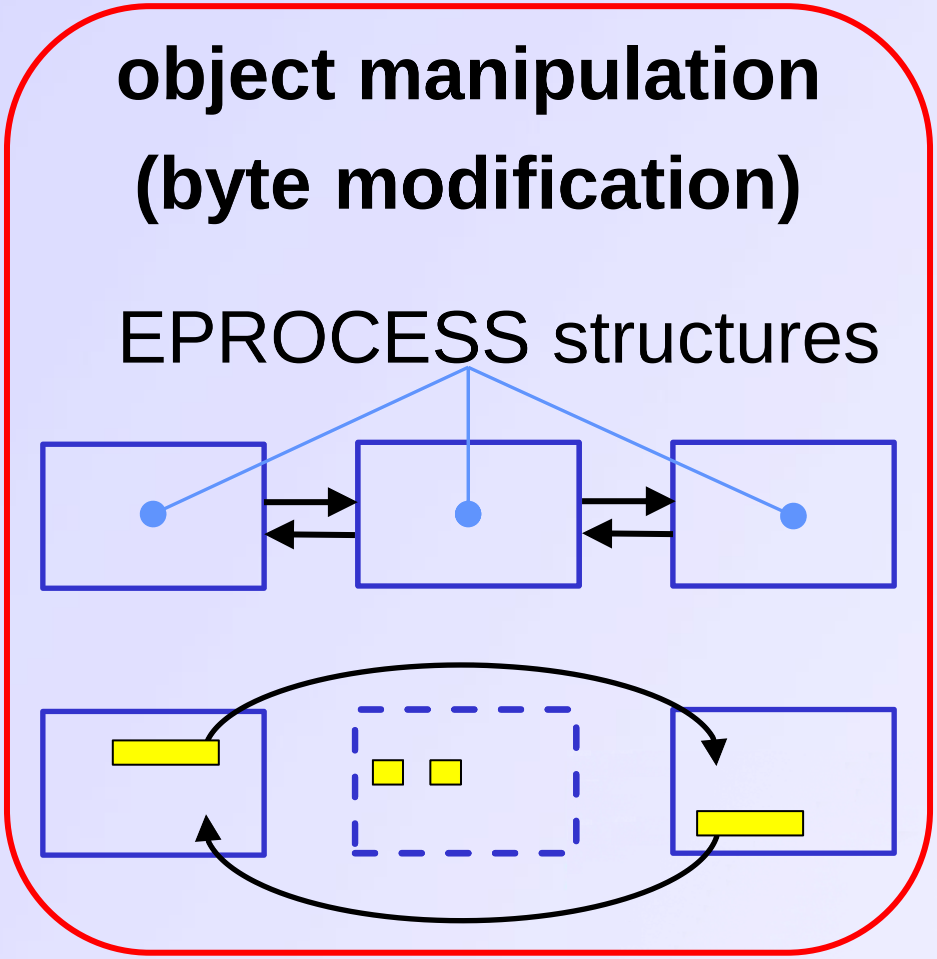
DBS for processes

Review of rootkits techniques

Rootkits techniques – malware hiding
from OS & AV

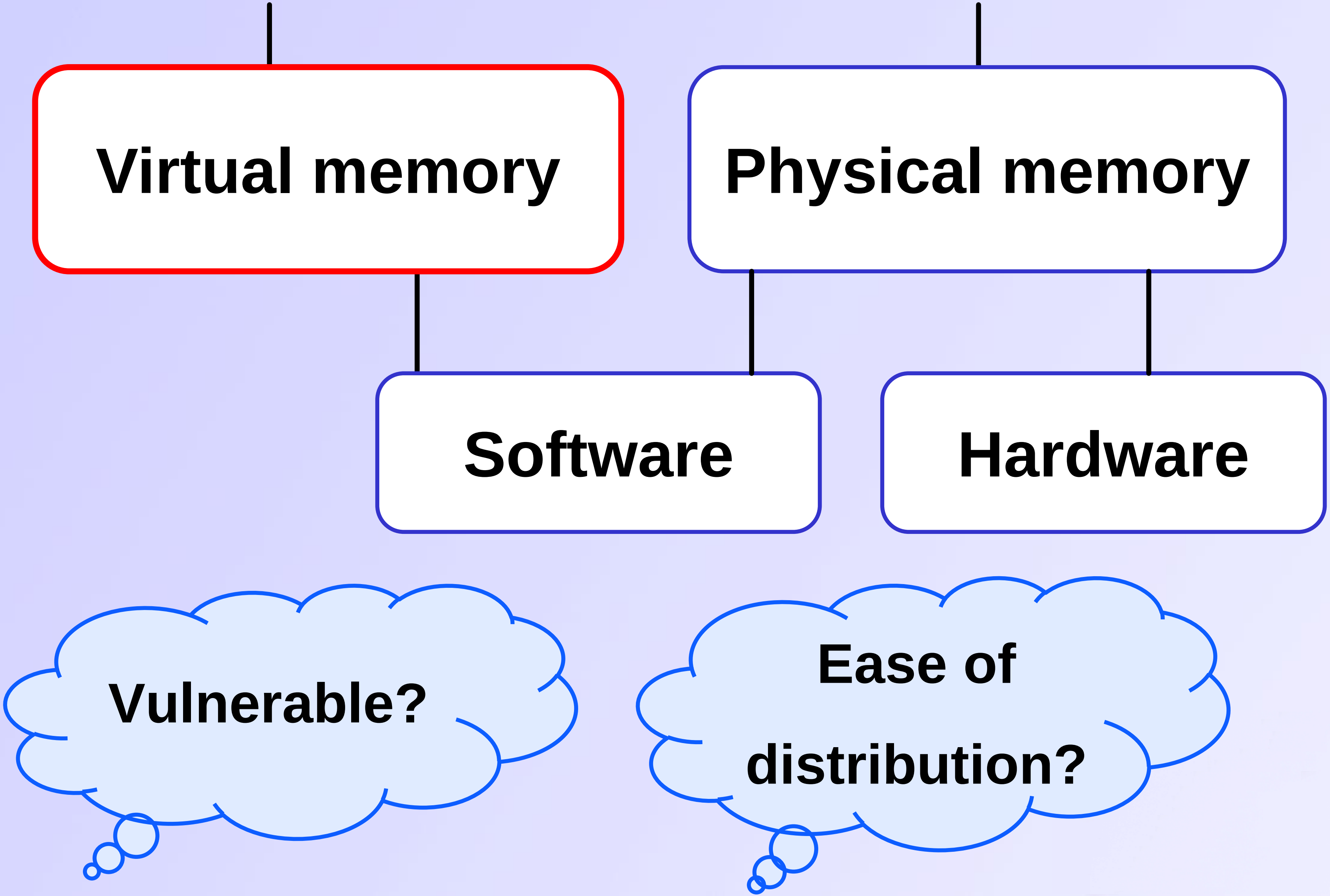


function hooking



object manipulation (byte modification)

Dump approaches classification



Virtual memory

Physical memory

Software

Hardware

Vulnerable?

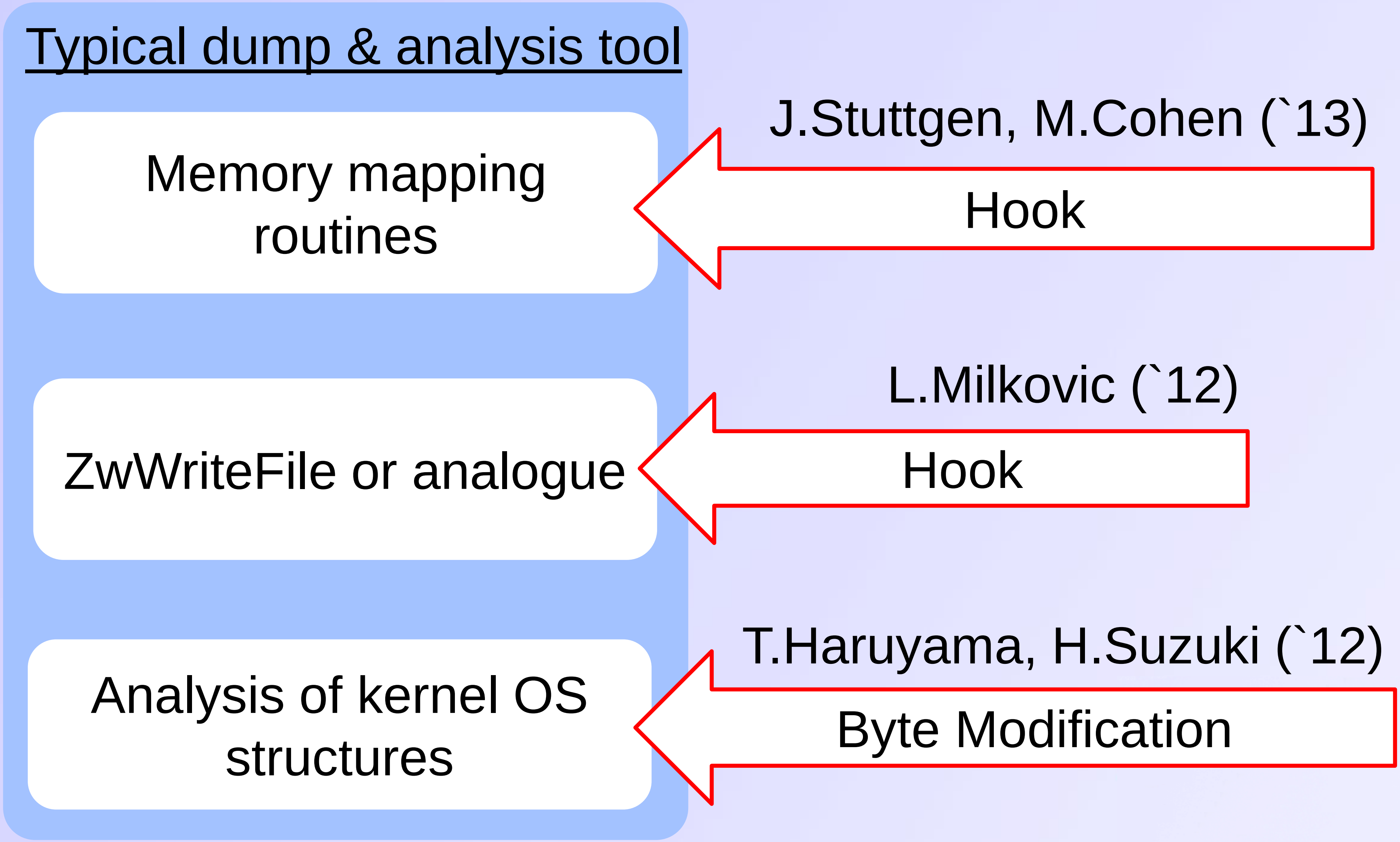
Ease of distribution?

Dump approaches are either vulnerable or non applicable in enterprises

	Hooking resilience	Ease of distribution
Software	–	+
Hardware	+	–

Why are software approaches vulnerable?

Details of dump & analysis tools



Typical dump & analysis tool

Memory mapping routines

ZwWriteFile or analogue

Analysis of kernel OS structures

J.Stuttgen, M.Cohen (`13)

Hook

L.Milkovic (`12)

Hook

T.Haruyama, H.Suzuki (`12)

Byte Modification

What can we do under these circumstances?

What can we do under these circumstances?

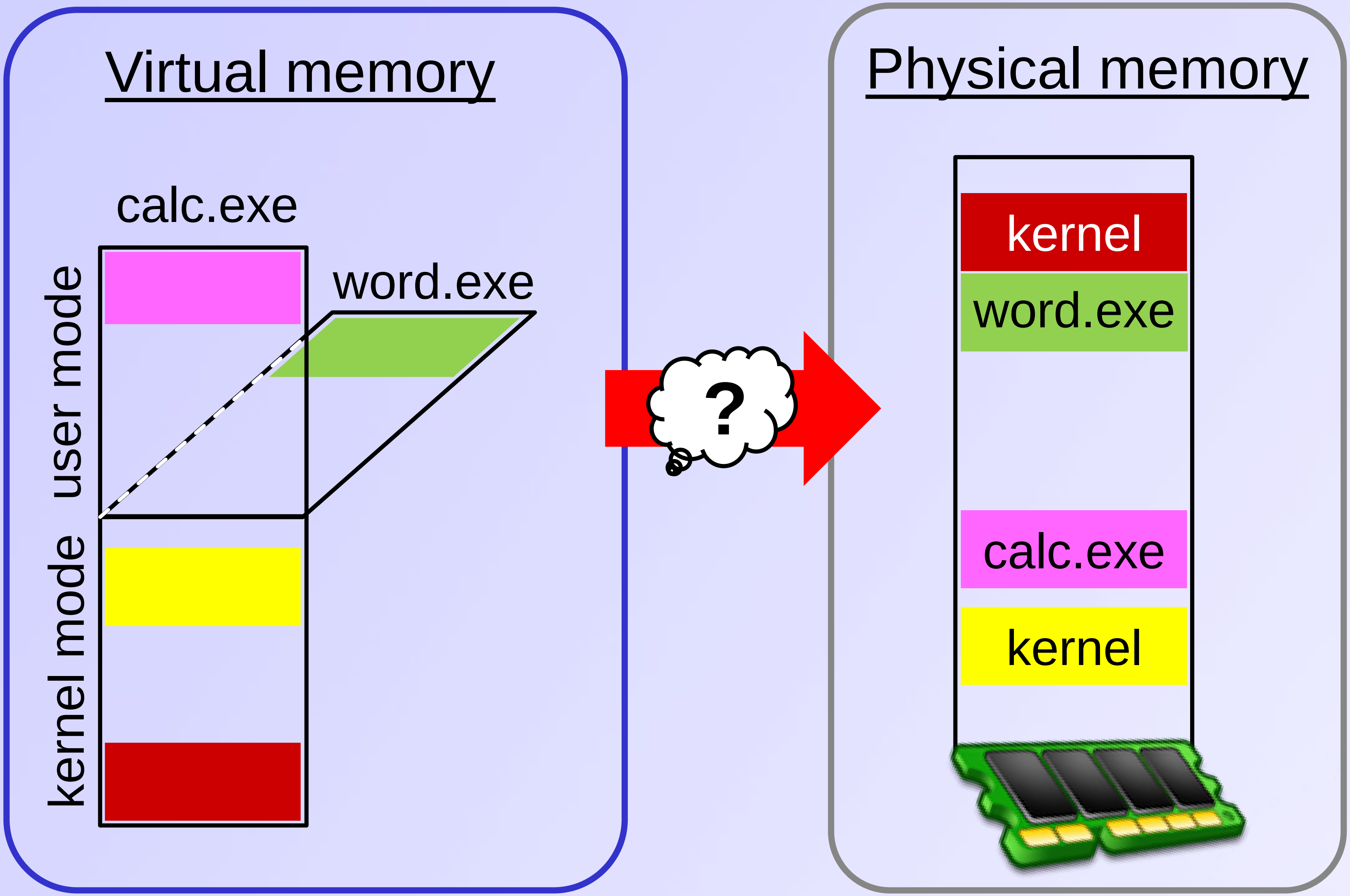
Let's omit the functions!

What can we do under these circumstances?

Let's omit the functions!

What can we use instead?

Virtual and Physical memory



Virtual memory

calc.exe

word.exe

user mode

kernel mode

?

Physical memory

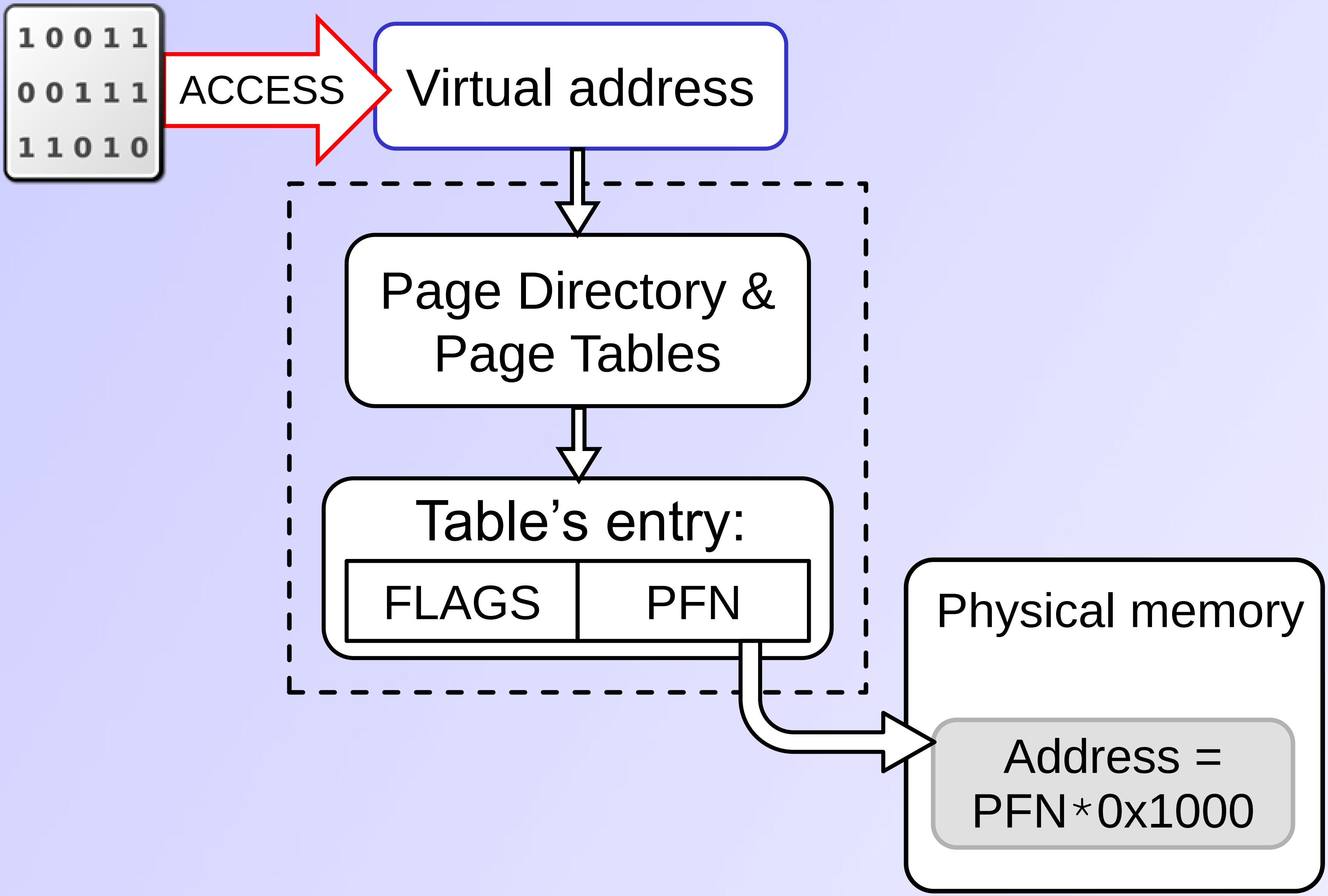
kernel

word.exe

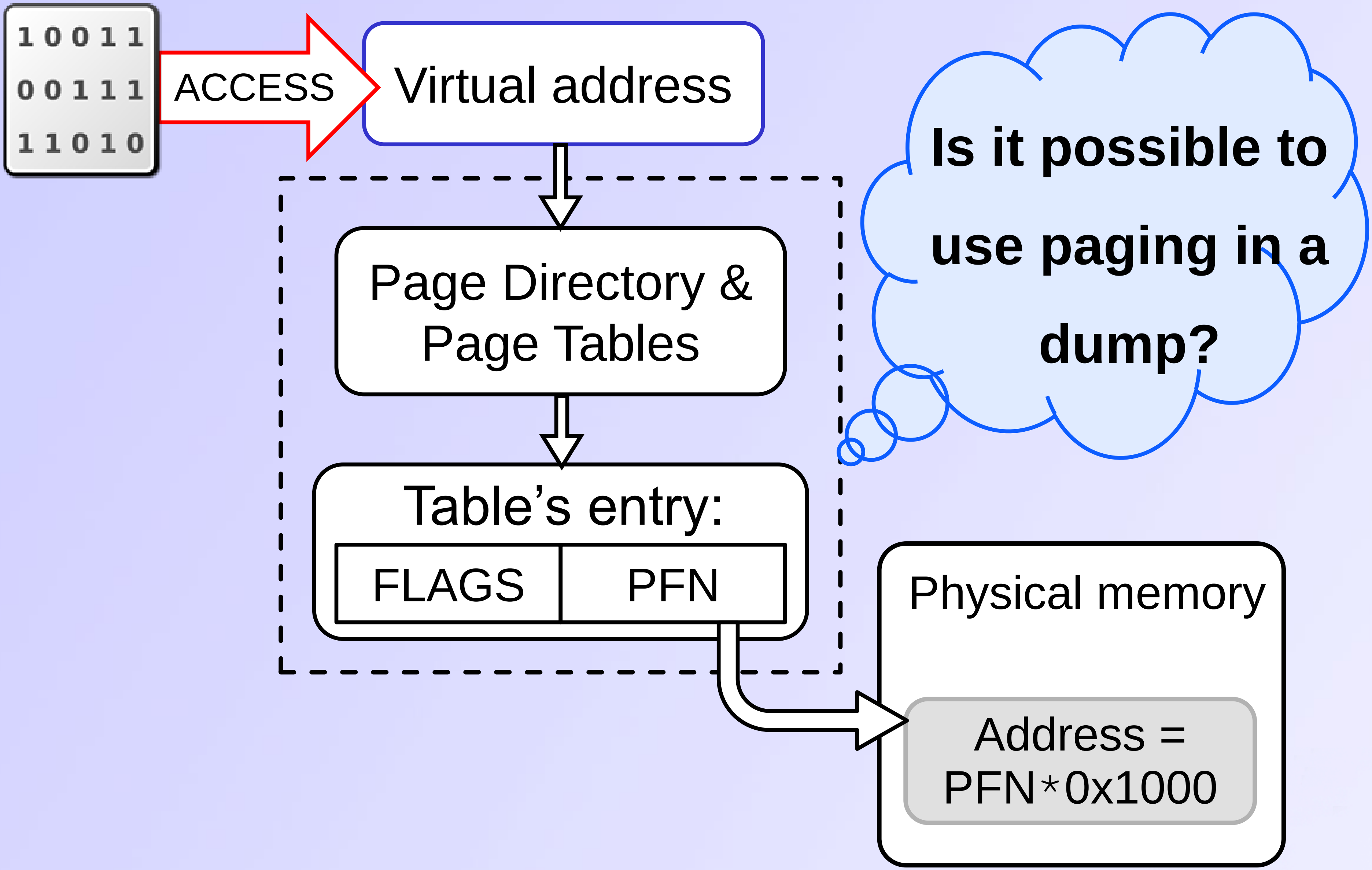
calc.exe

kernel

How does addresses translation work?

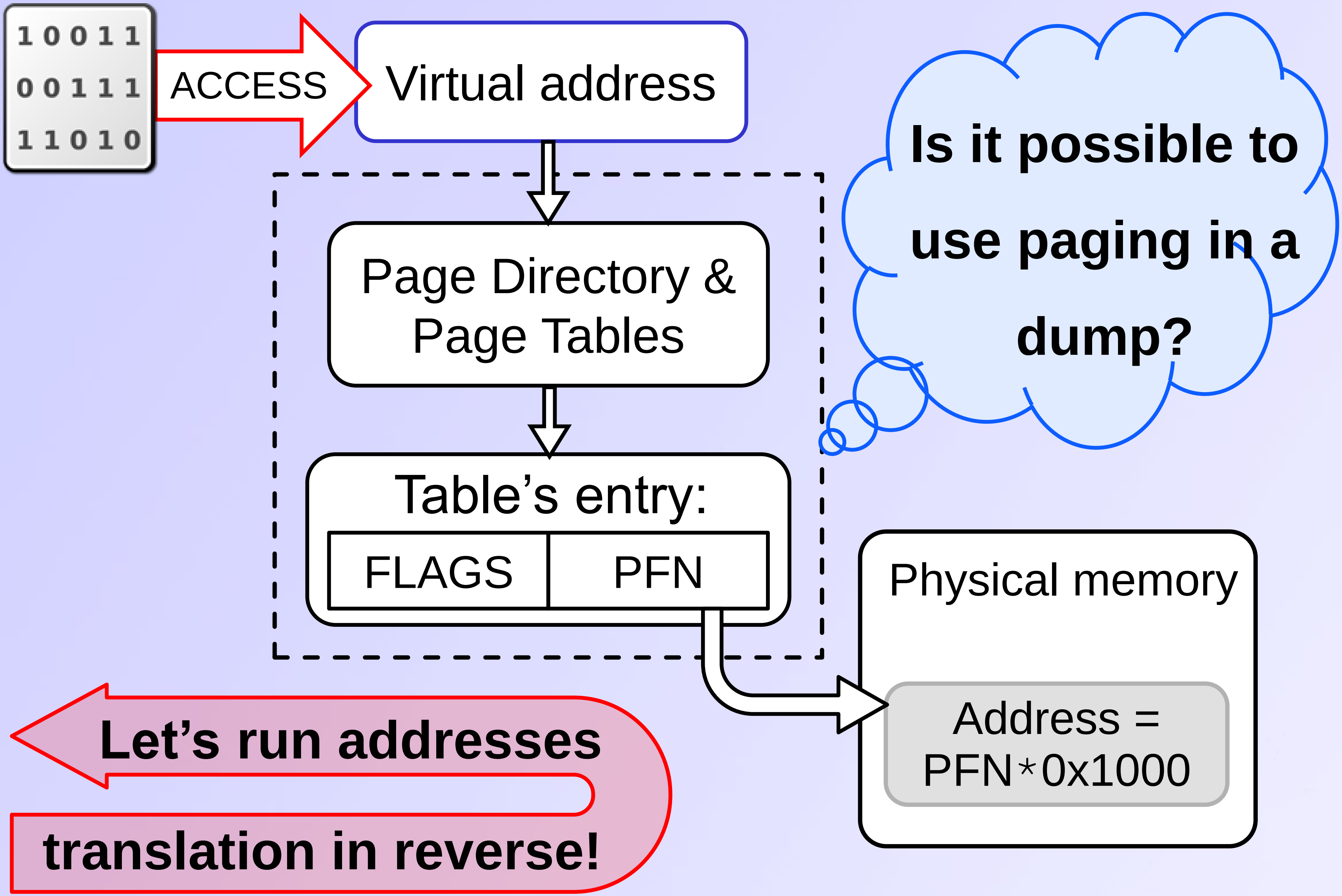


How does addresses translation work?



Is it possible to use paging in a dump?

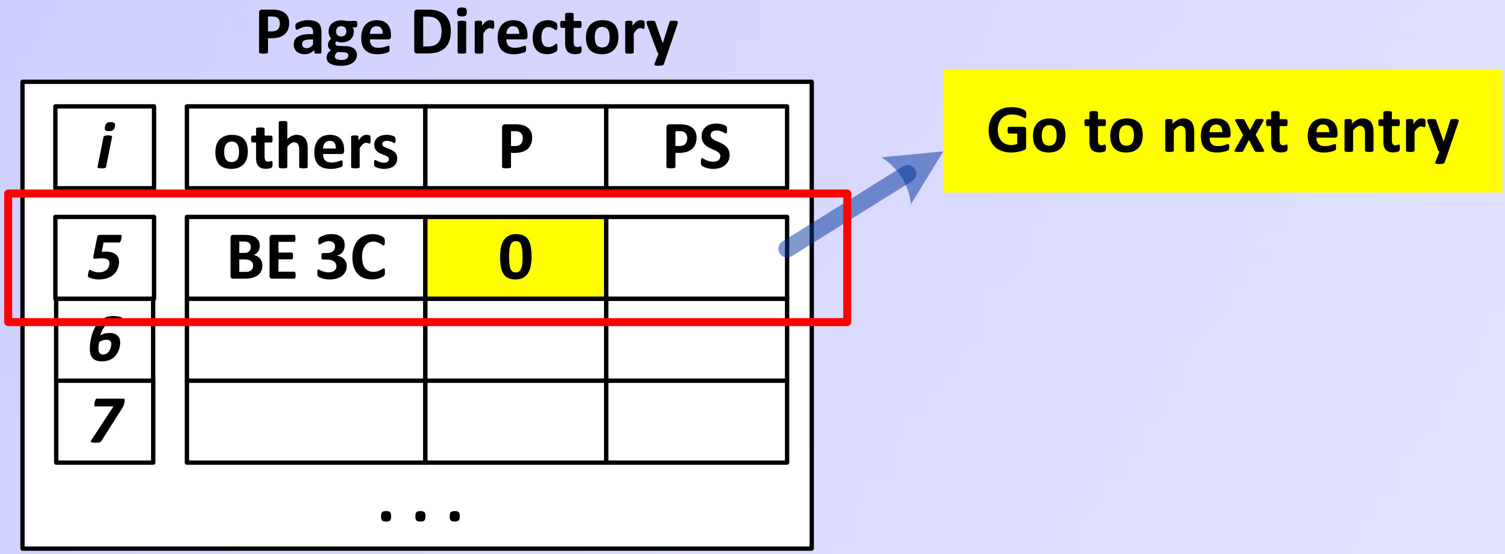
How does addresses translation work?



Is it possible to use paging in a dump?

Let's run addresses translation in reverse!

MASHKA’s memory dump algorithm



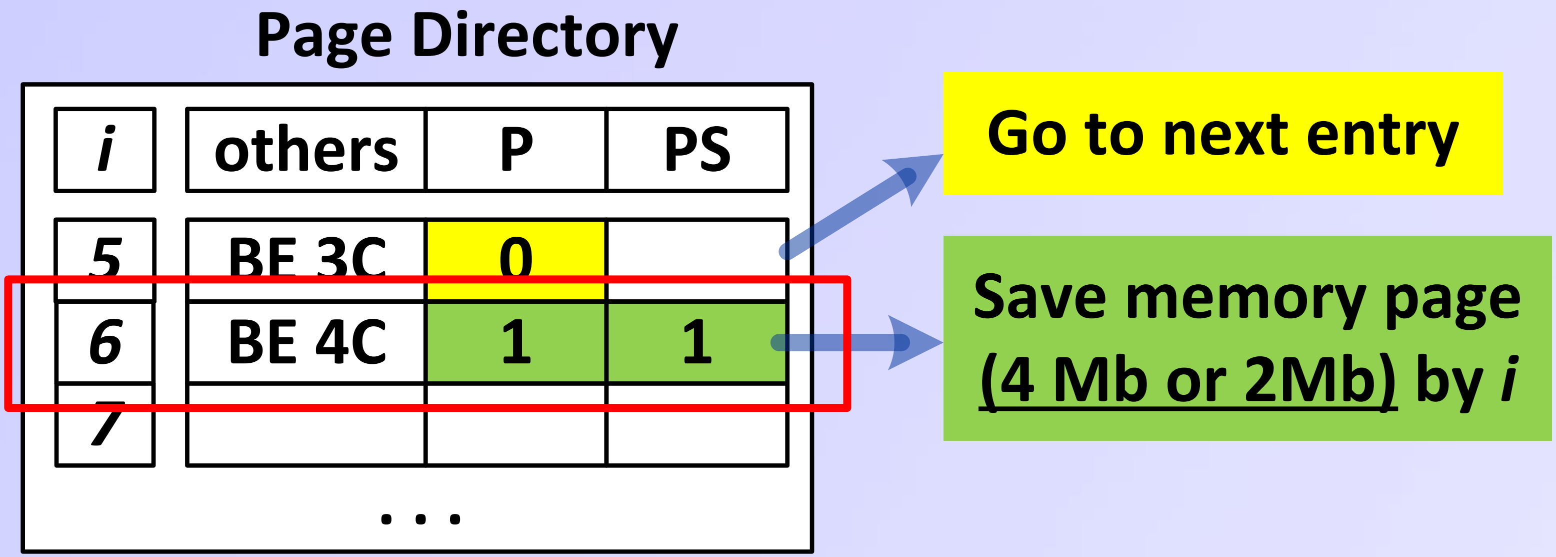
Page Directory

i	others	P	PS
5	BE 3C	0	
6			
7			

. . .

Go to next entry

MASHKA’s memory dump algorithm



Page Directory

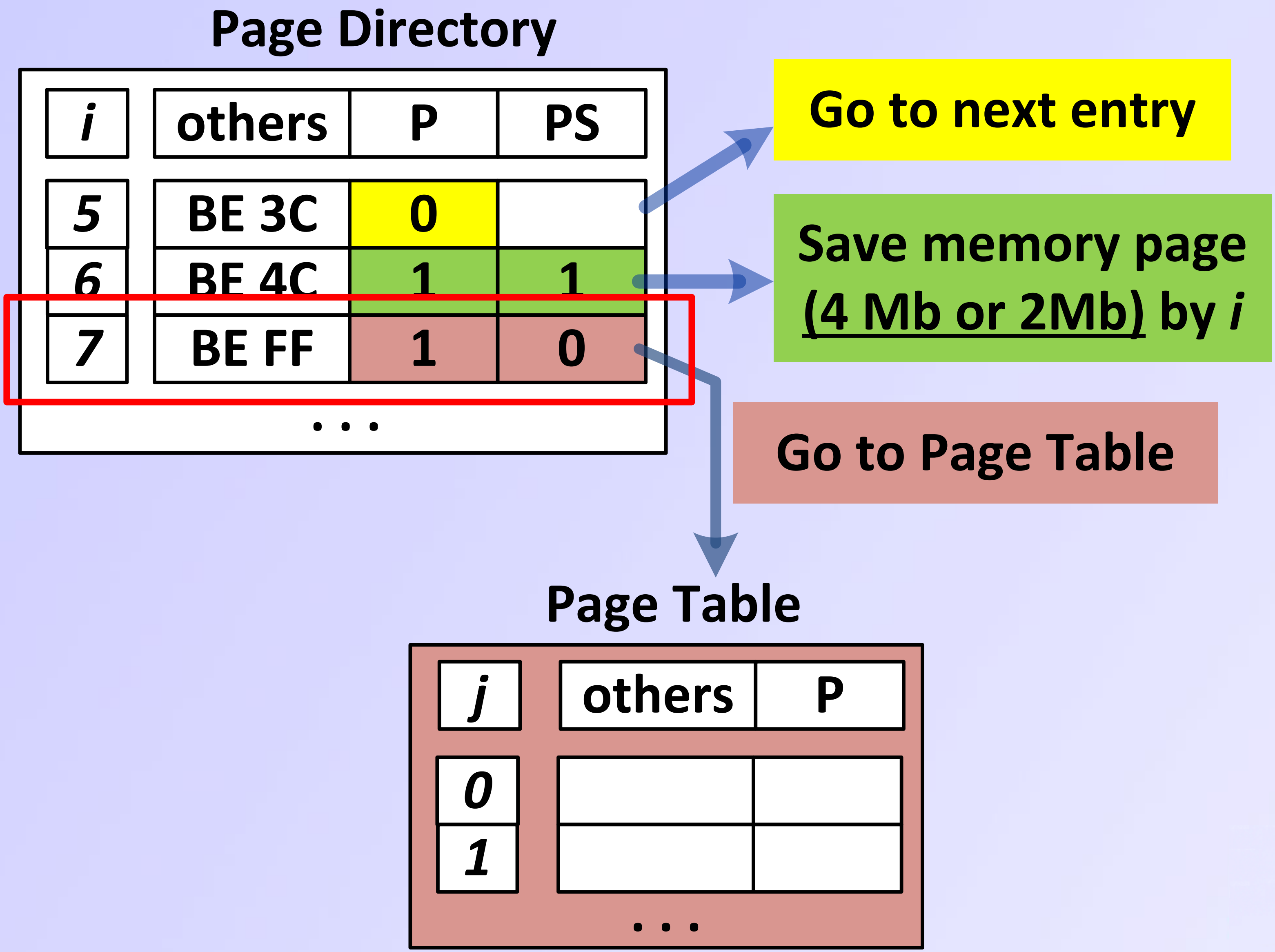
i	others	P	PS
5	BE 3C	0	
6	BE 4C	1	1
7			

. . .

Go to next entry

Save memory page <u>(4 Mb or 2Mb)</u> by *i*

MASHKA’s memory dump algorithm



Page Directory

i	others	P	PS
5	BE 3C	0	
6	BE 4C	1	1
7	BE FF	1	0

. . .

Go to next entry

Save memory page (4 Mb or 2Mb) by *i*

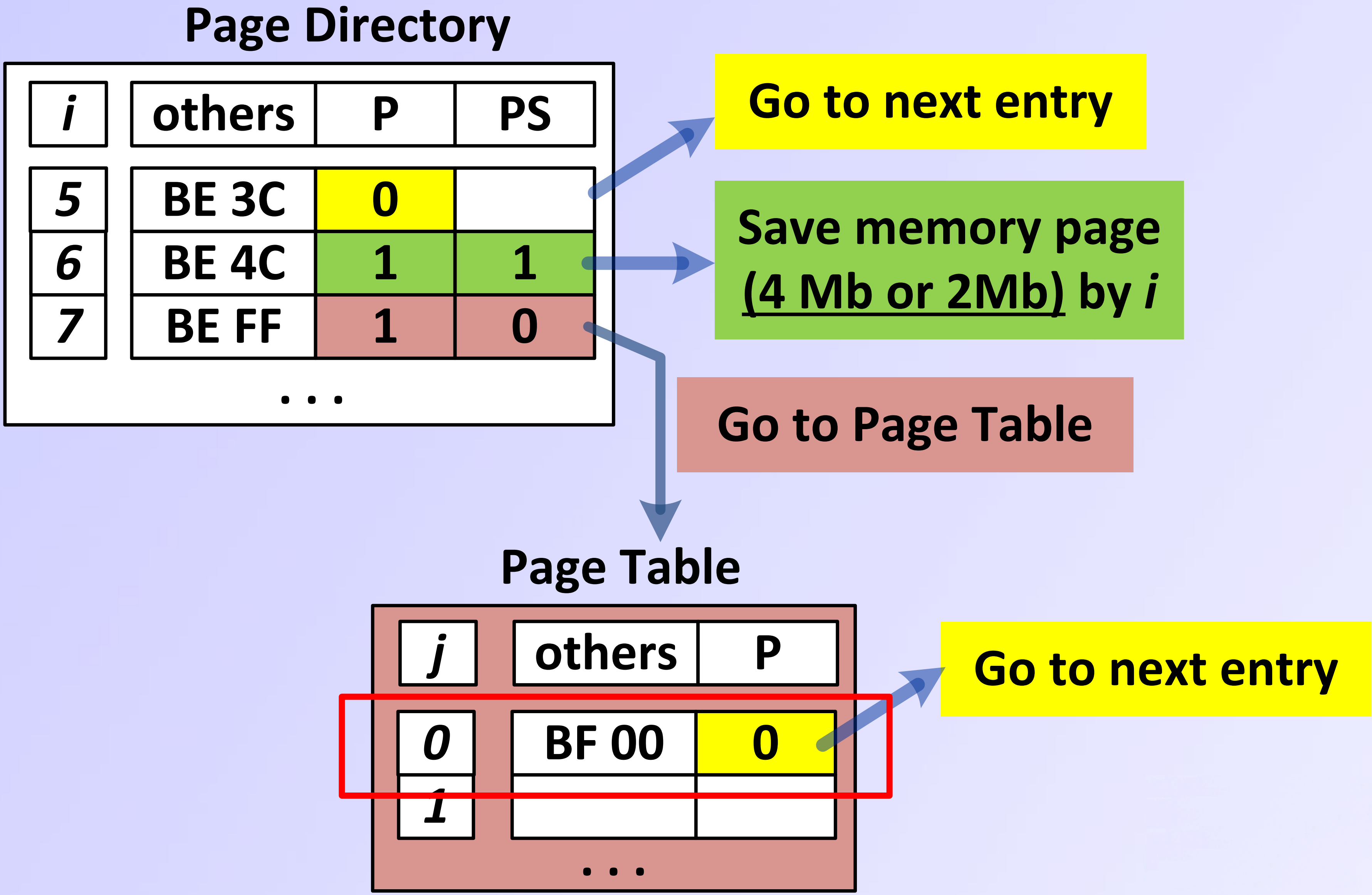
Go to Page Table

Page Table

j	others	P
0		
1		

. . .

MASHKA’s memory dump algorithm



Page Directory

i	others	P	PS
5	BE 3C	0	
6	BE 4C	1	1
7	BE FF	1	0

. . .

Go to next entry

Save memory page (4 Mb or 2Mb) by *i*

Go to Page Table

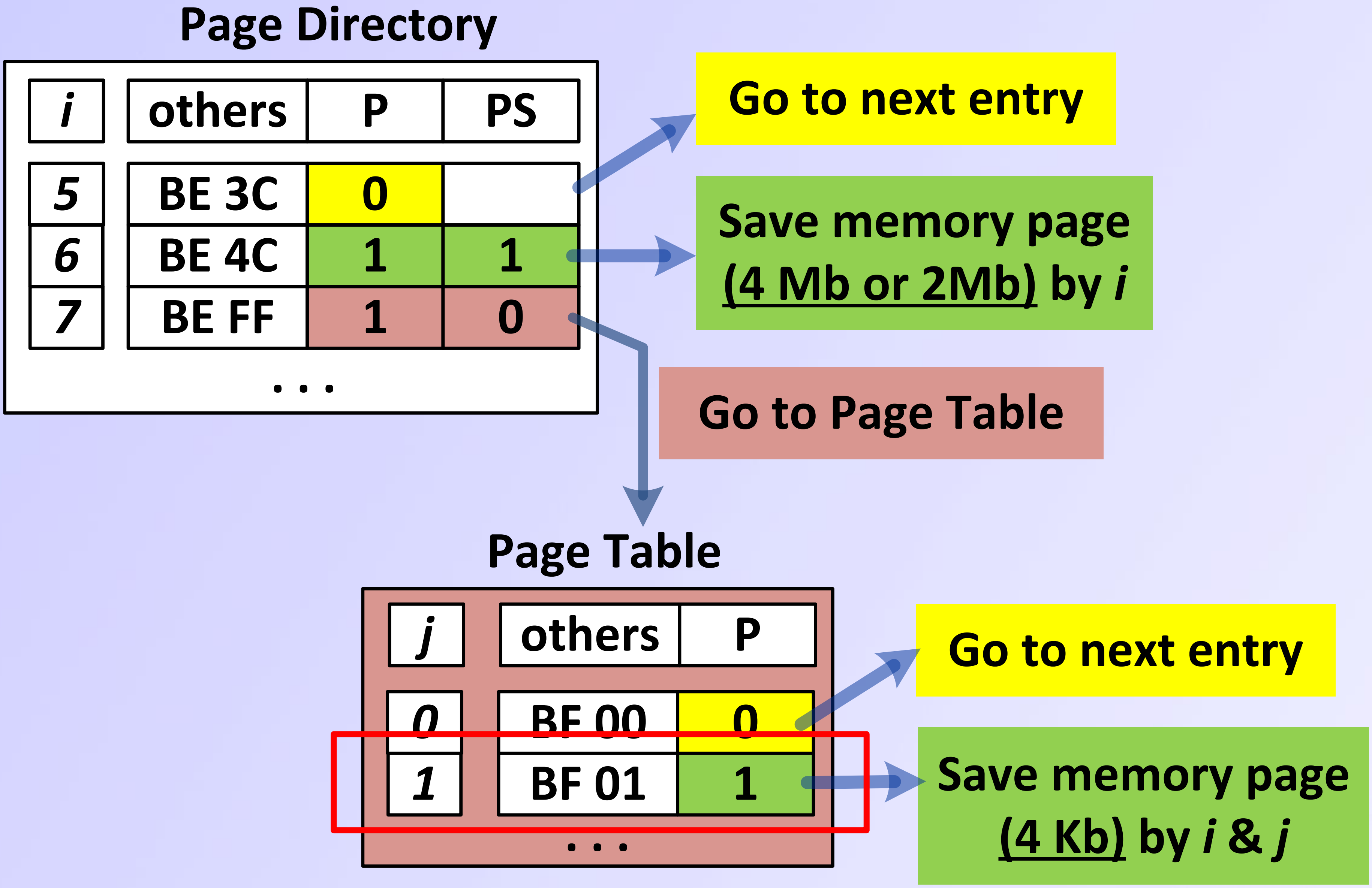
Page Table

j	others	P
0	BF 00	0
1		

. . .

Go to next entry

MASHKA’s memory dump algorithm



Page Directory

i	others	P	PS
5	BE 3C	0	
6	BE 4C	1	1
7	BE FF	1	0
	. . .		

Go to next entry

Save memory page (4 Mb or 2Mb) by *i*

Go to Page Table

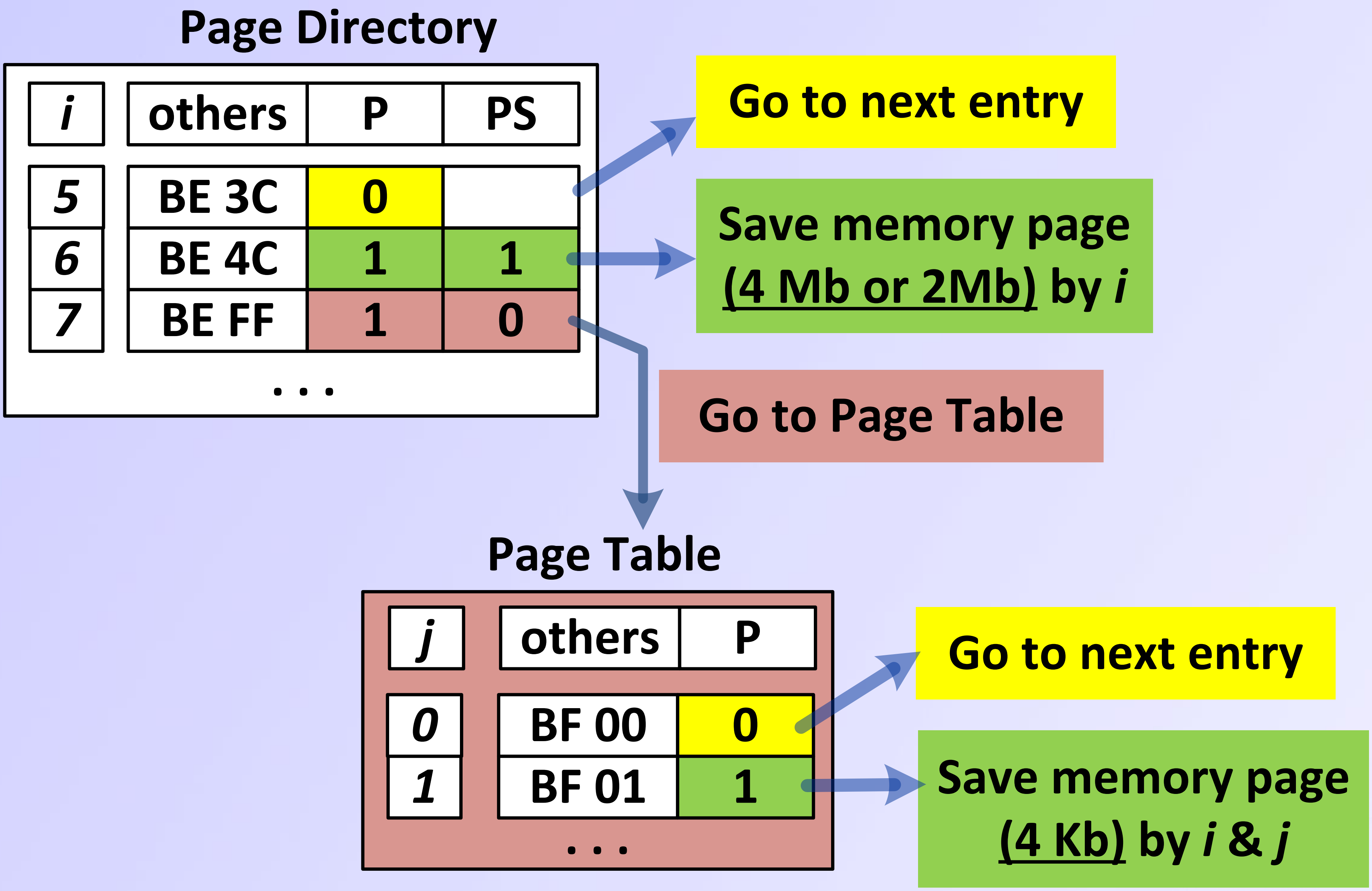
Page Table

j	others	P
0	BF 00	0
1	BF 01	1
	. . .	

Go to next entry

Save memory page (4 Kb) by *i* & *j*

MASHKA's memory dump algorithm



Page Directory

i	others	P	PS
5	BE 3C	0	
6	BE 4C	1	1
7	BE FF	1	0

. . .

Go to next entry

Save memory page (4 Mb or 2Mb) by *i*

Go to Page Table

Page Table

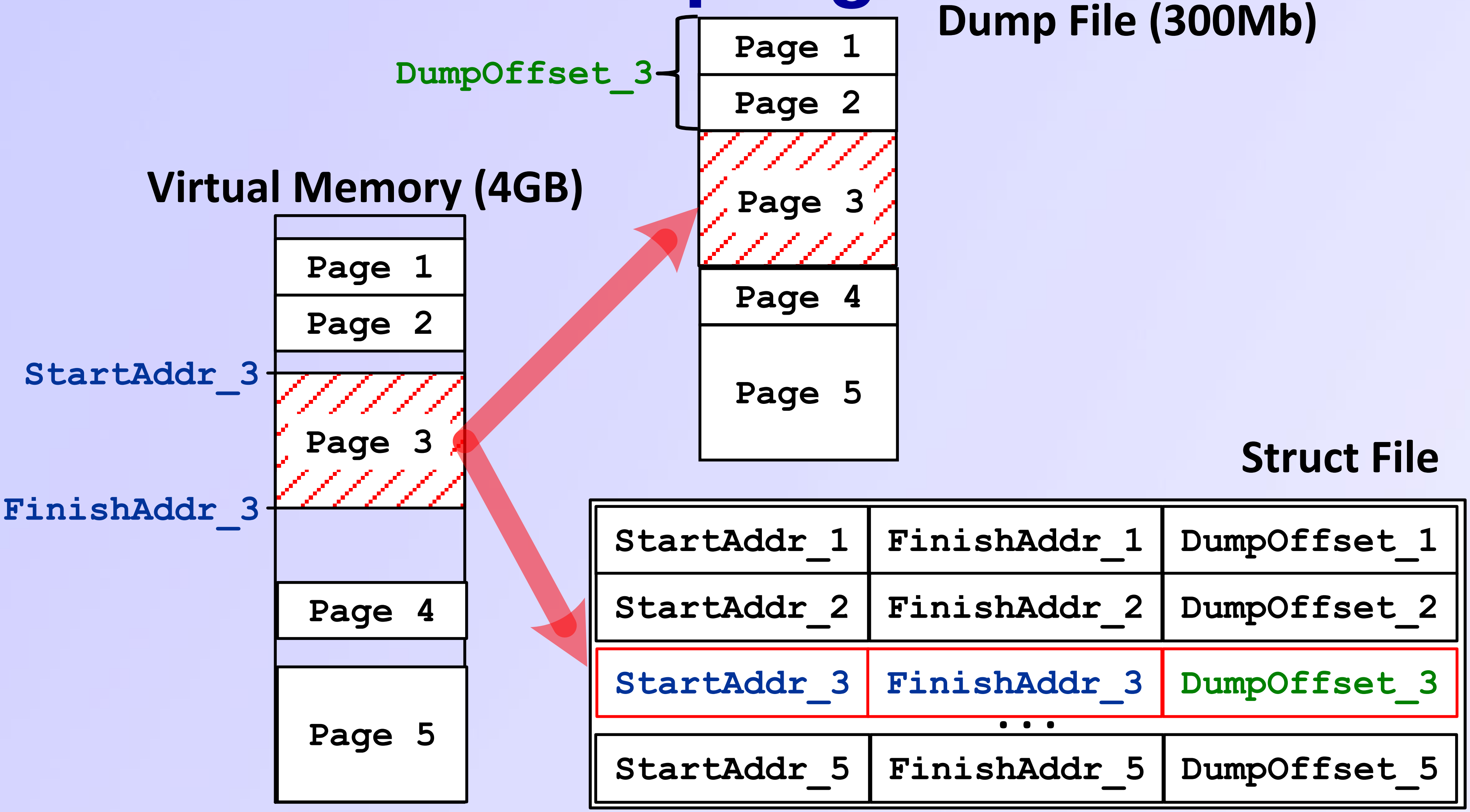
j	others	P
0	BF 00	0
1	BF 01	1

. . .

Go to next entry

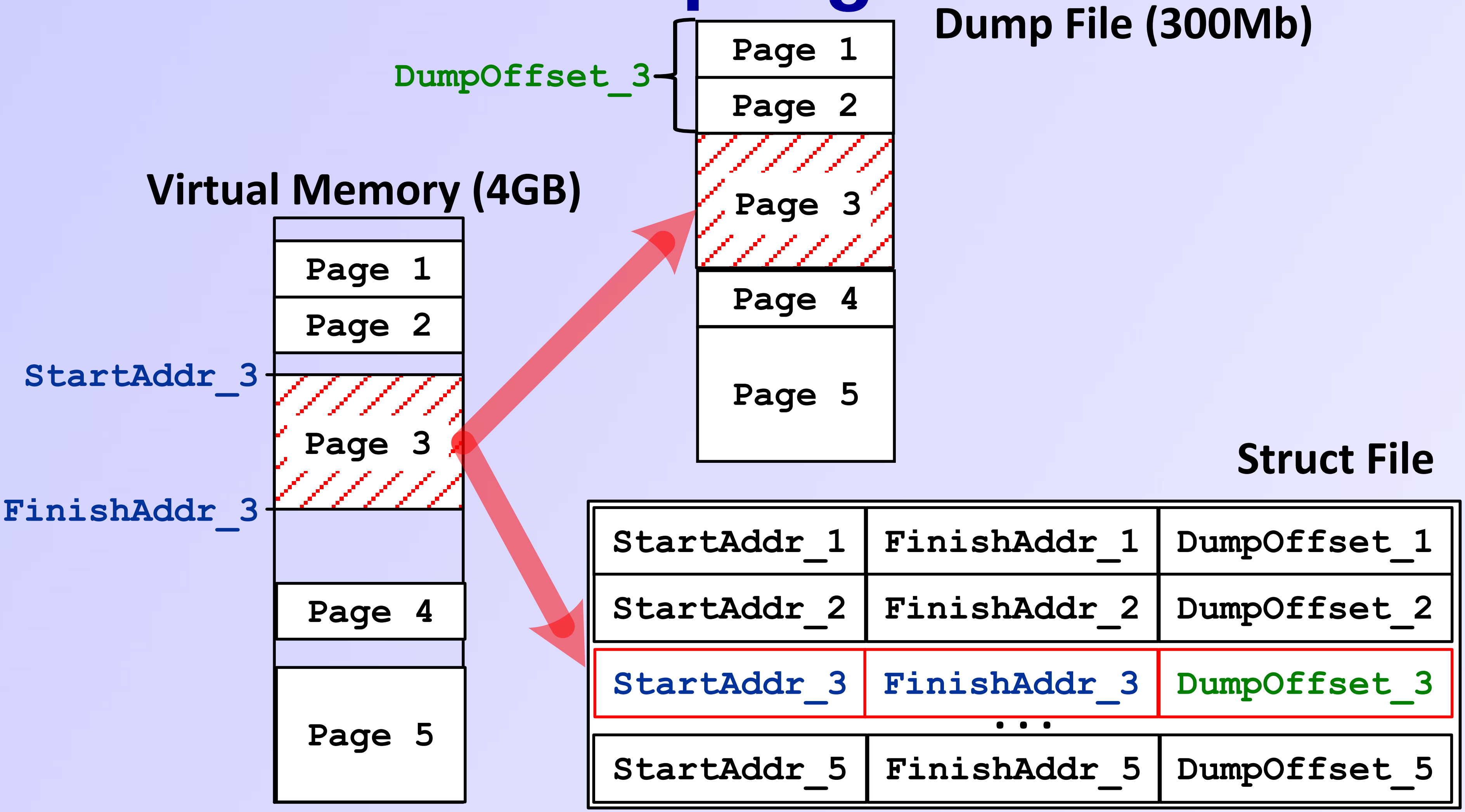
Save memory page (4 Kb) by *i* & *j*

MASHKA's dump algorithm details



StartAddr_1	FinishAddr_1	DumpOffset_1
StartAddr_2	FinishAddr_2	DumpOffset_2
StartAddr_3	FinishAddr_3	DumpOffset_3
	...	
StartAddr_5	FinishAddr_5	DumpOffset_5

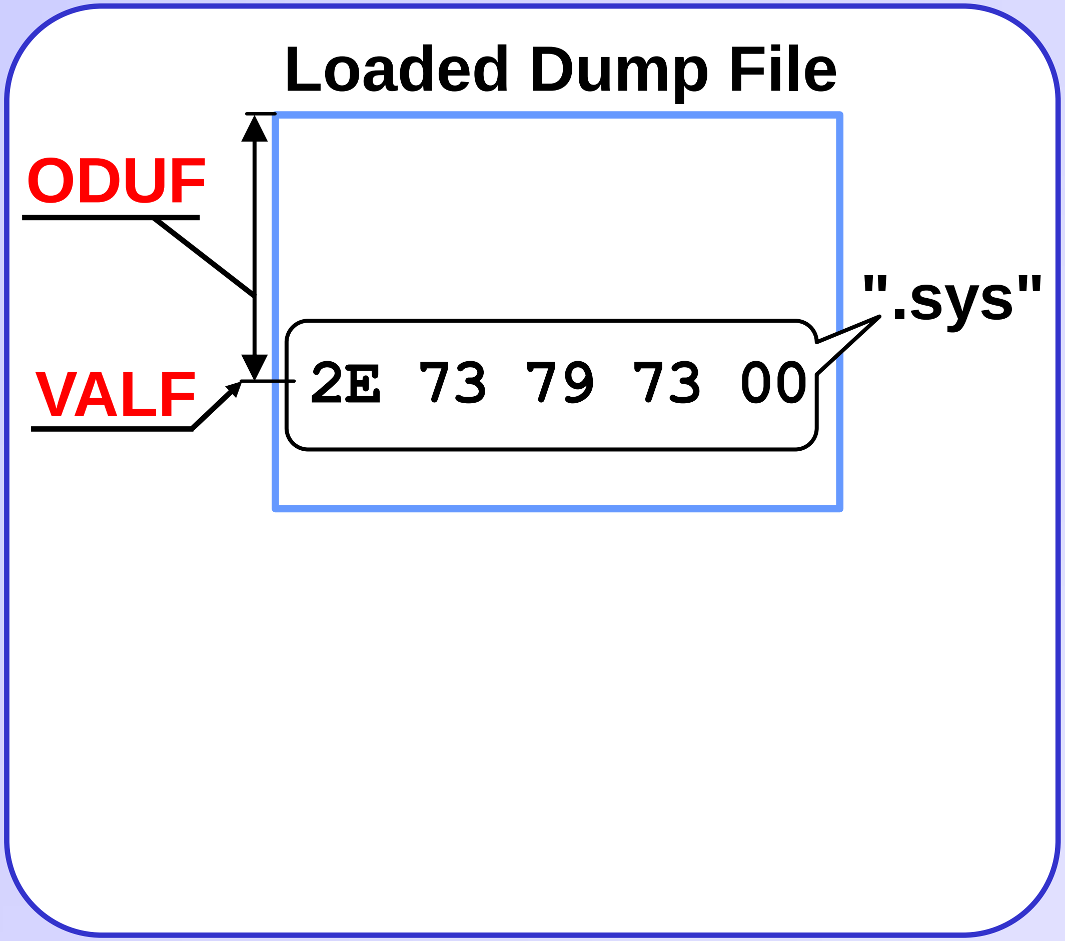
MASHKA’s dump algorithm details



StartAddr_1	FinishAddr_1	DumpOffset_1
StartAddr_2	FinishAddr_2	DumpOffset_2
StartAddr_3	FinishAddr_3	DumpOffset_3
	...	
StartAddr_5	FinishAddr_5	DumpOffset_5

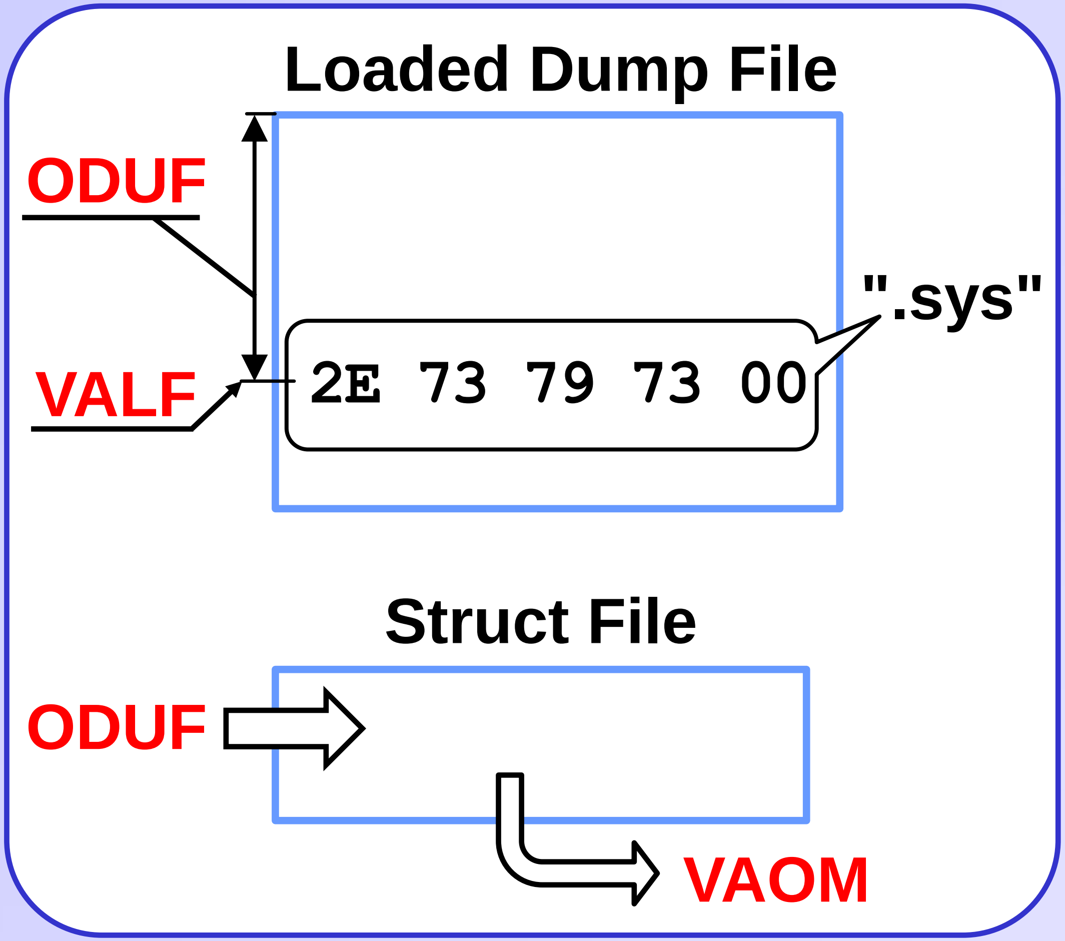
How should new files be used?

MASHKA in memory forensics tasks



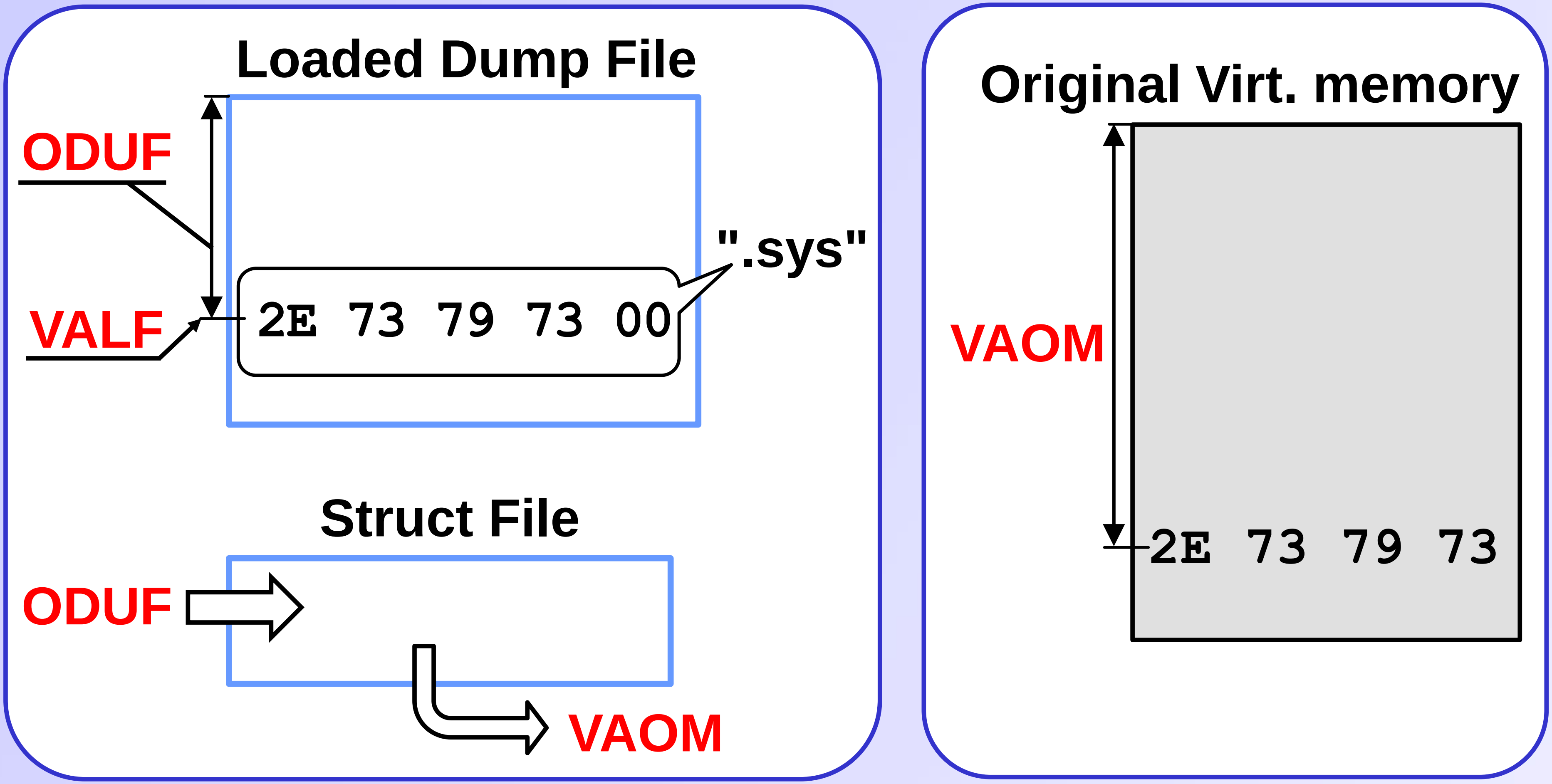
VALF	Virtual Address in the Loaded dump File
ODUF	Offset in DUmp File

MASHKA in memory forensics tasks



VAOM	Virtual Address in the Original virt. Memory
VALF	Virtual Address in the Loaded dump File
ODUF	Offset in DUmp File

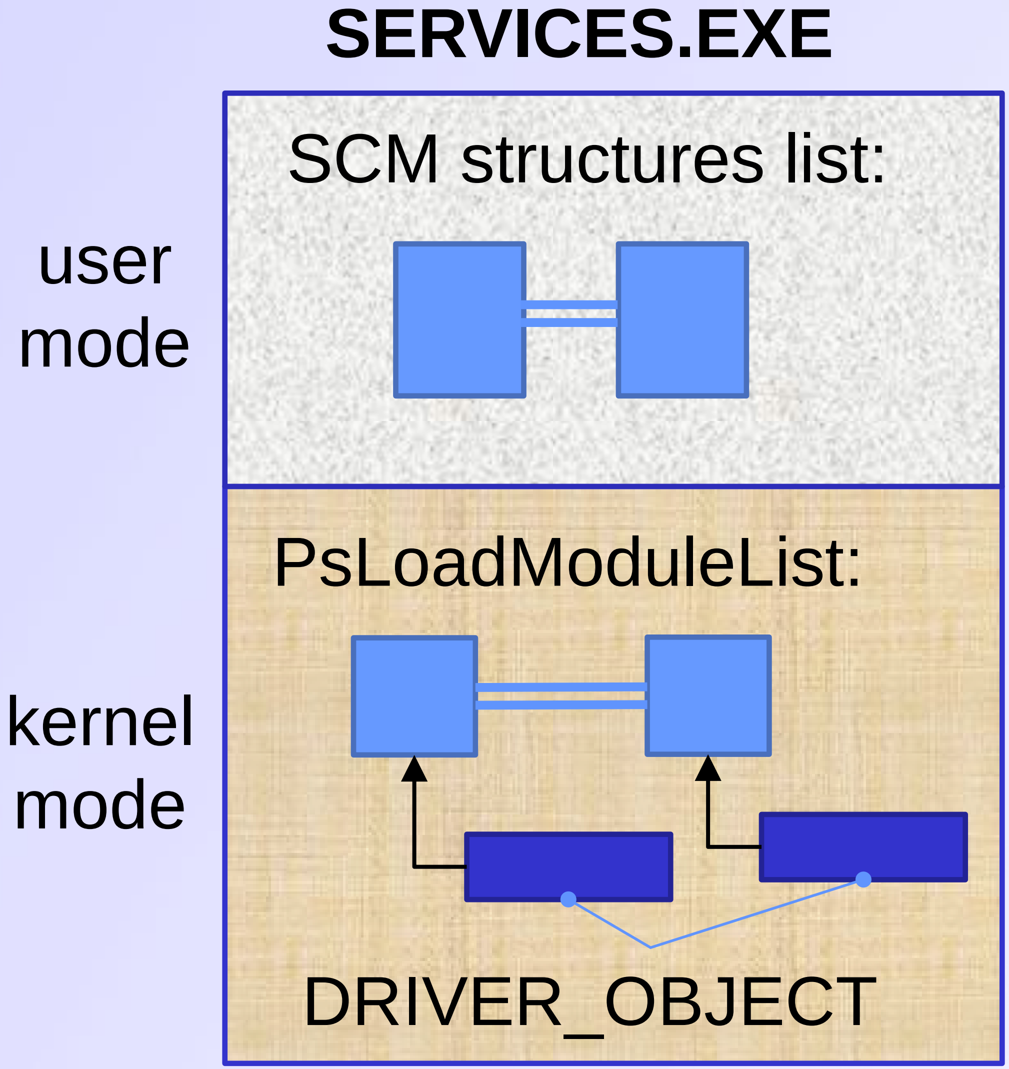
MASHKA in memory forensics tasks



VAOM	Virtual Address in the Original virt. Memory
VALF	Virtual Address in the Loaded dump File
ODUF	Offset in DUmp File

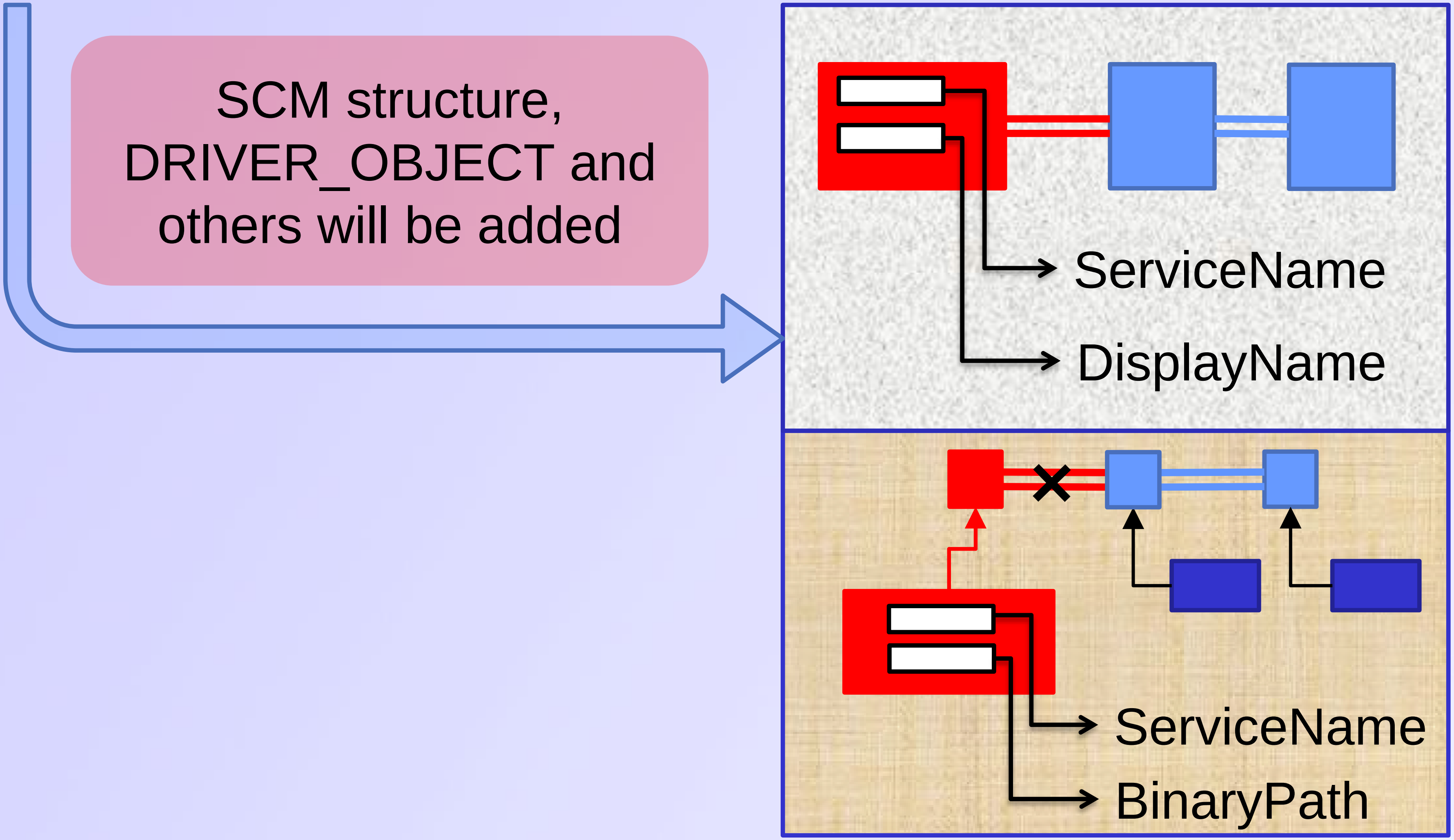
How is VAOM etc used?

Use MASHKA in drivers forensics



Use MASHKA in drivers forensics

CreateService(ServiceName, DisplayName, BinaryPath,...)

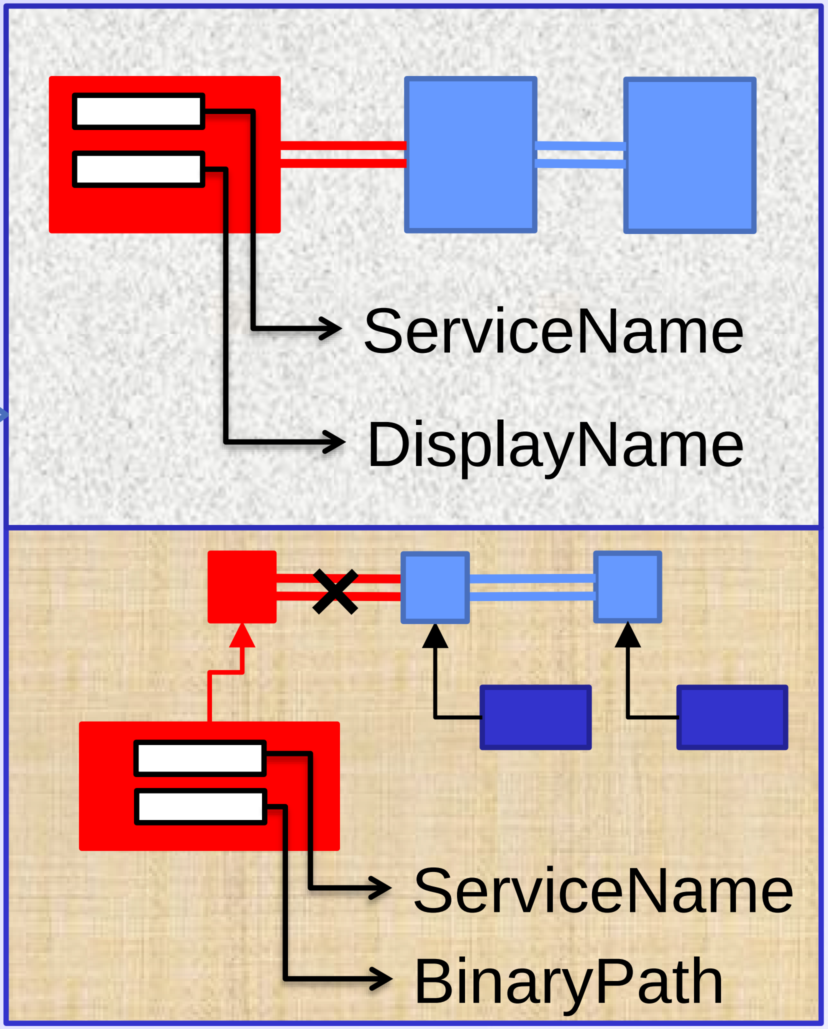


SCM structure, DRIVER_OBJECT and others will be added

Use MASHKA in drivers forensics

CreateService(ServiceName, DisplayName, BinaryPath,...)

SCM structure, DRIVER_OBJECT and others will be added

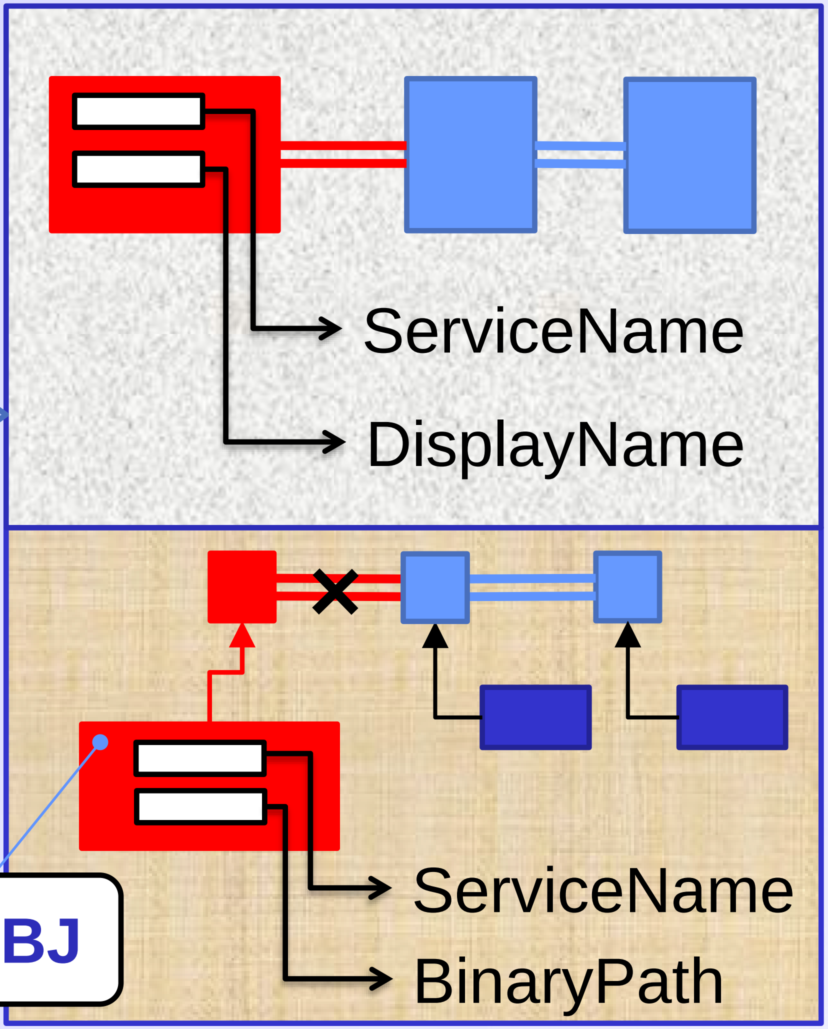


ServiceName > VAOMs of ‘SN’

Use MASHKA in drivers forensics

CreateService(ServiceName, DisplayName, BinaryPath,...)

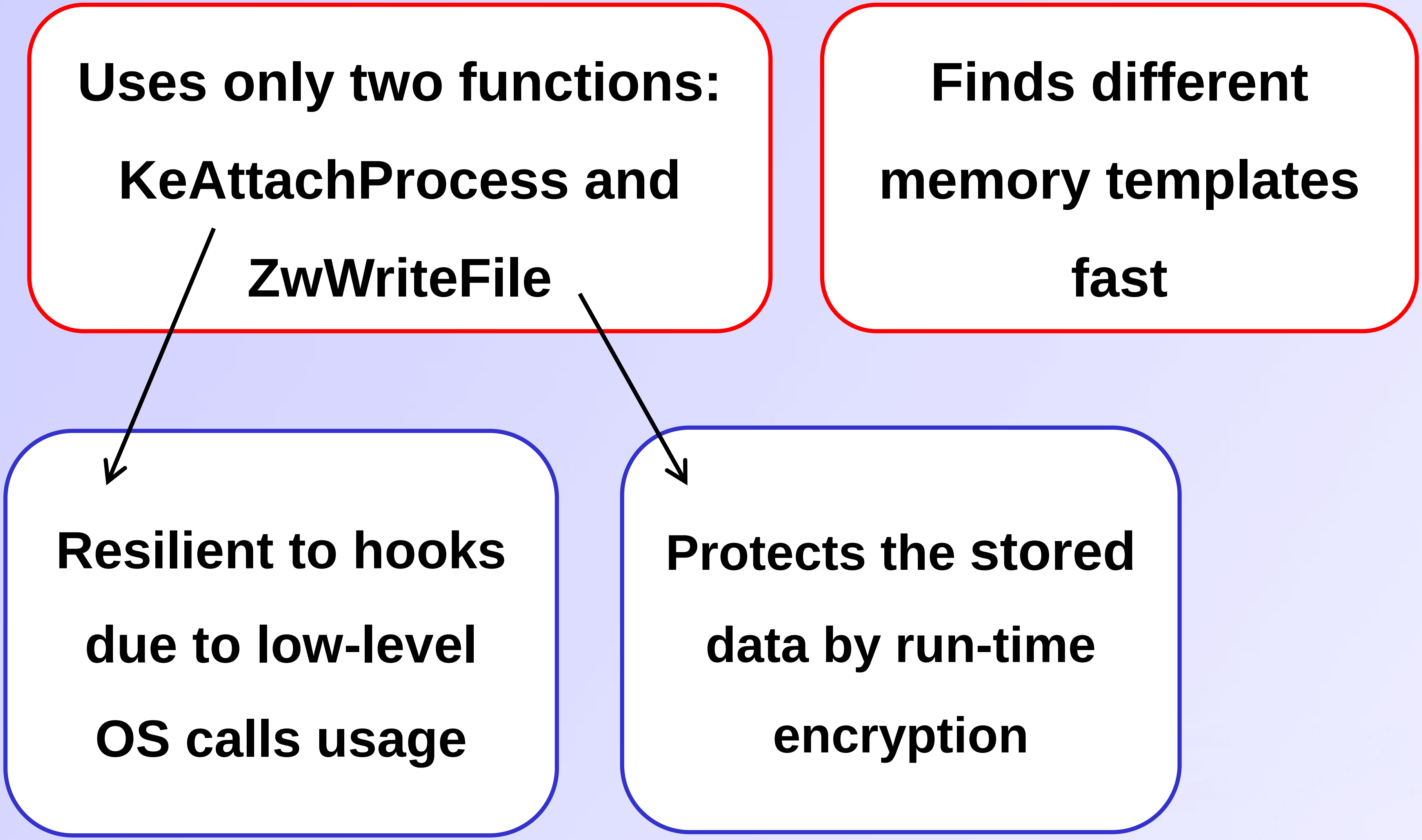
SCM structure, DRIVER_OBJECT and others will be added



ServiceName > VAOMs of ‘SN’

VAOMs of ‘SN’ > VAOM of DRV_OBJ

Advantages of MASHKA



Uses only two functions: KeAttachProcess and ZwWriteFile

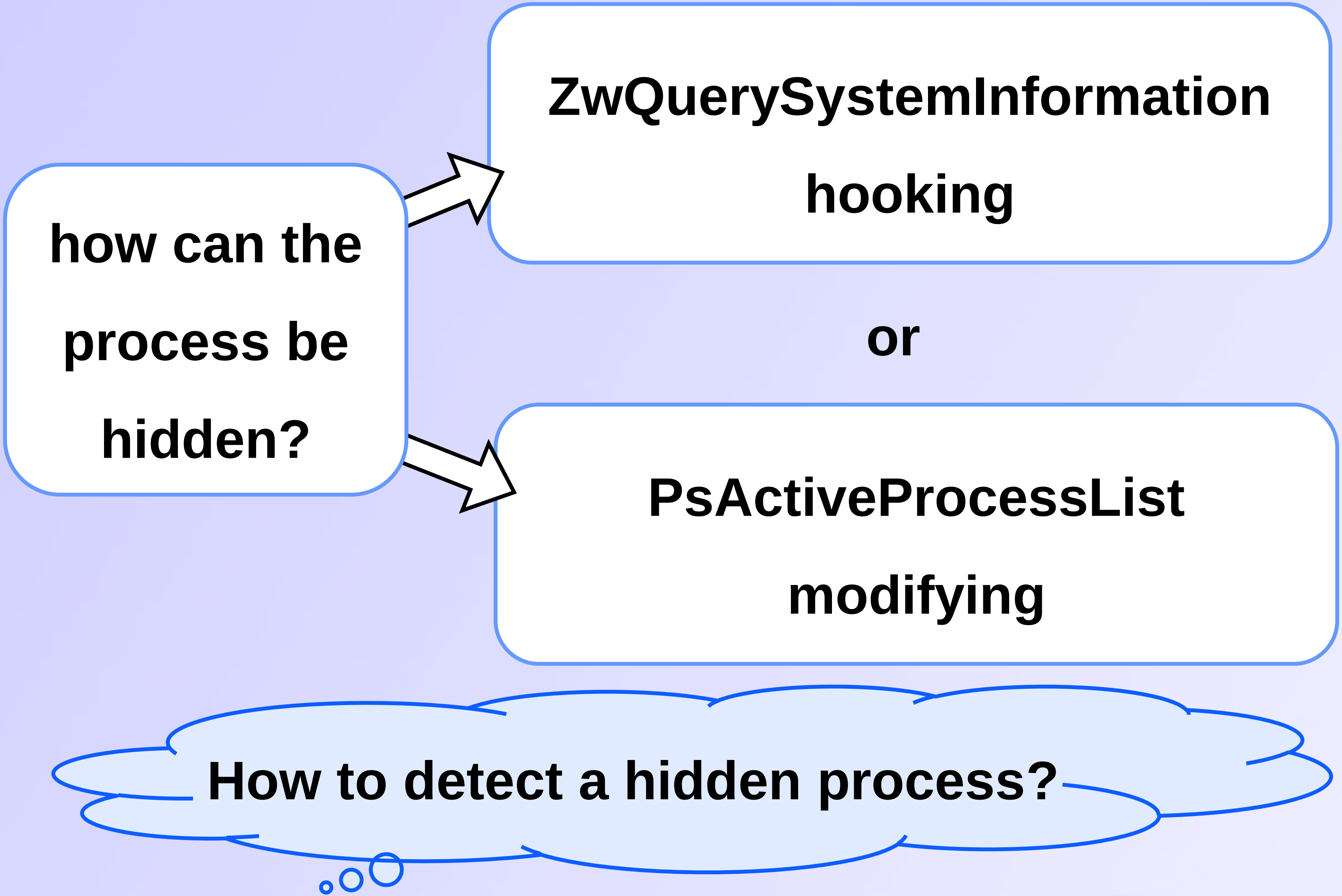
Finds different memory templates fast

Resilient to hooks due to low-level OS calls usage

Protects the stored data by run-time encryption

How to apply MASHKA to processes detection?

OS processes list handling



how can the process be hidden?

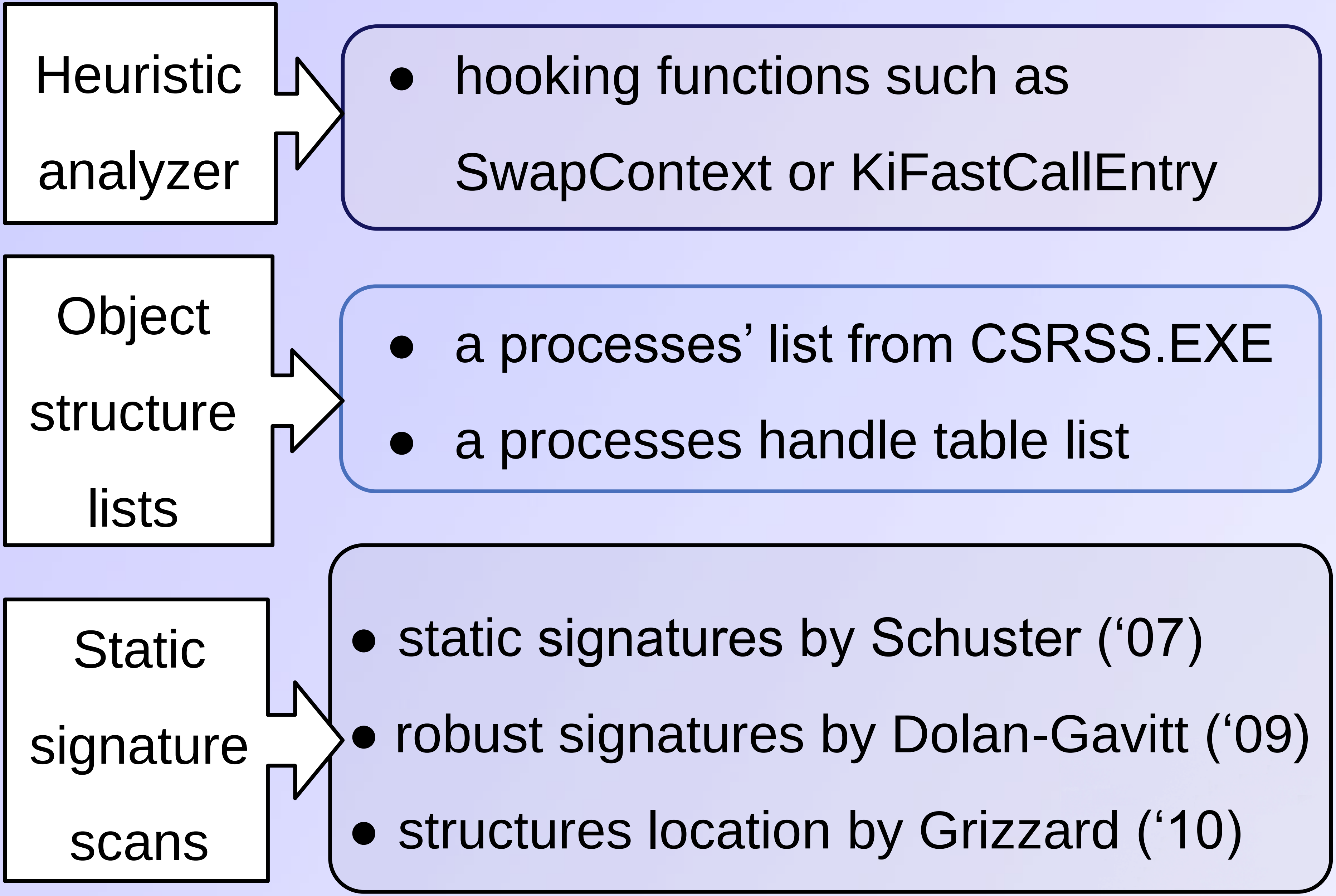
ZwQuerySystemInformation hooking

or

PsActiveProcessList modifying

How to detect a hidden process?

Process detection approaches review



Heuristic analyzer

- hooking functions such as SwapContext or KiFastCallEntry

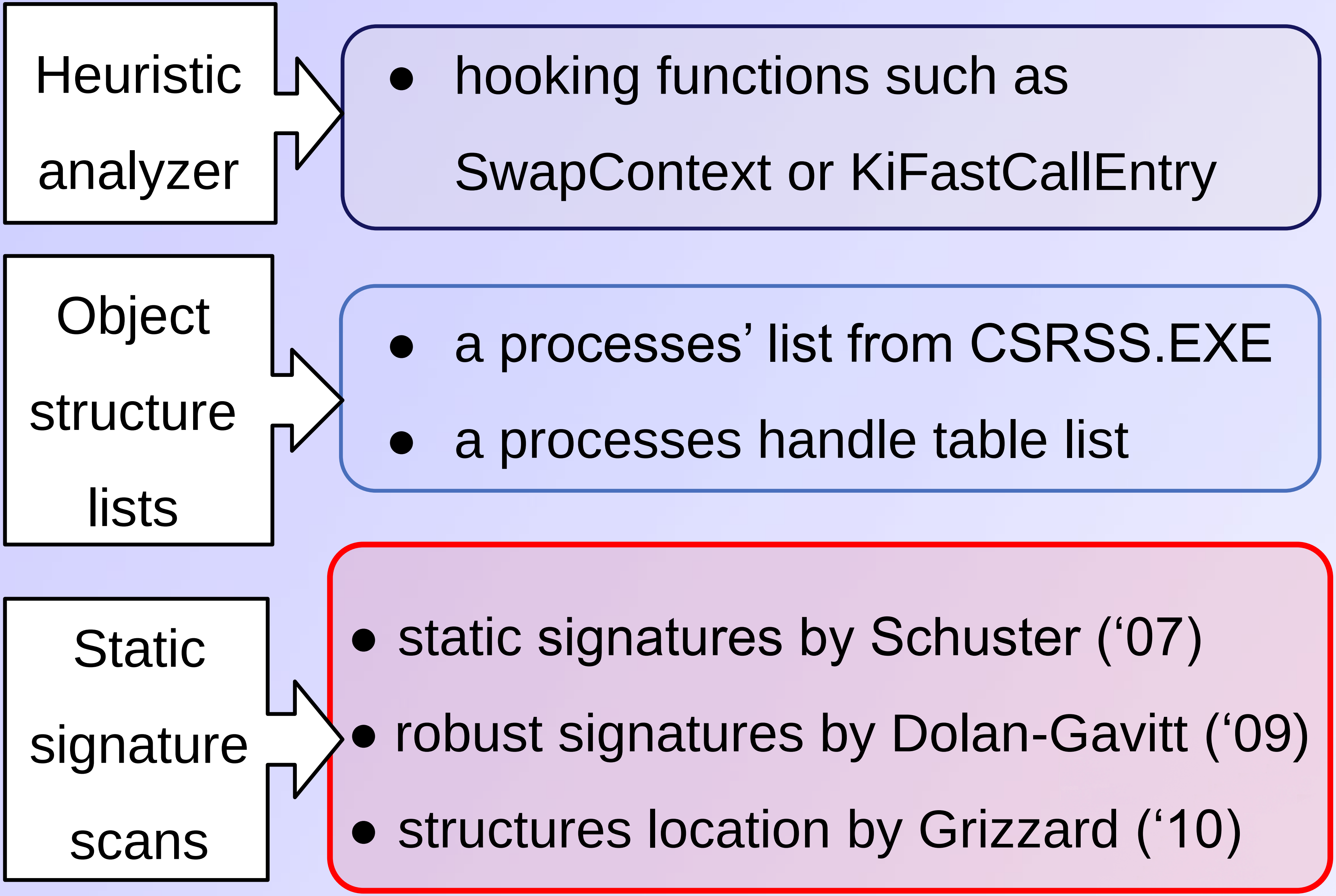
Object structure lists

- a processes’ list from CSRSS.EXE
- a processes handle table list

Static signature scans

- static signatures by Schuster (‘07)
- robust signatures by Dolan-Gavitt (‘09)
- structures location by Grizzard (‘10)

Process detection approaches review



Heuristic analyzer

- hooking functions such as SwapContext or KiFastCallEntry

Object structure lists

- a processes' list from CSRSS.EXE
- a processes handle table list

Static signature scans

- static signatures by Schuster ('07)
- robust signatures by Dolan-Gavitt ('09)
- structures location by Grizzard ('10)

Analysis of static signature scan

GMER, PowerTool and XueTr use it

Scan is based on

some EPROCESS fields values are either known or exceed the constant, e.g. 0x8000_0000

Disadvantages

vulnerable to field modifications

difficult to achieve portability

Analysis of static signature scan

GMER, PowerTool and XueTr use it

Scan is based on

some EPROCESS fields values are either known or exceed the constant, e.g. 0x8000_0000

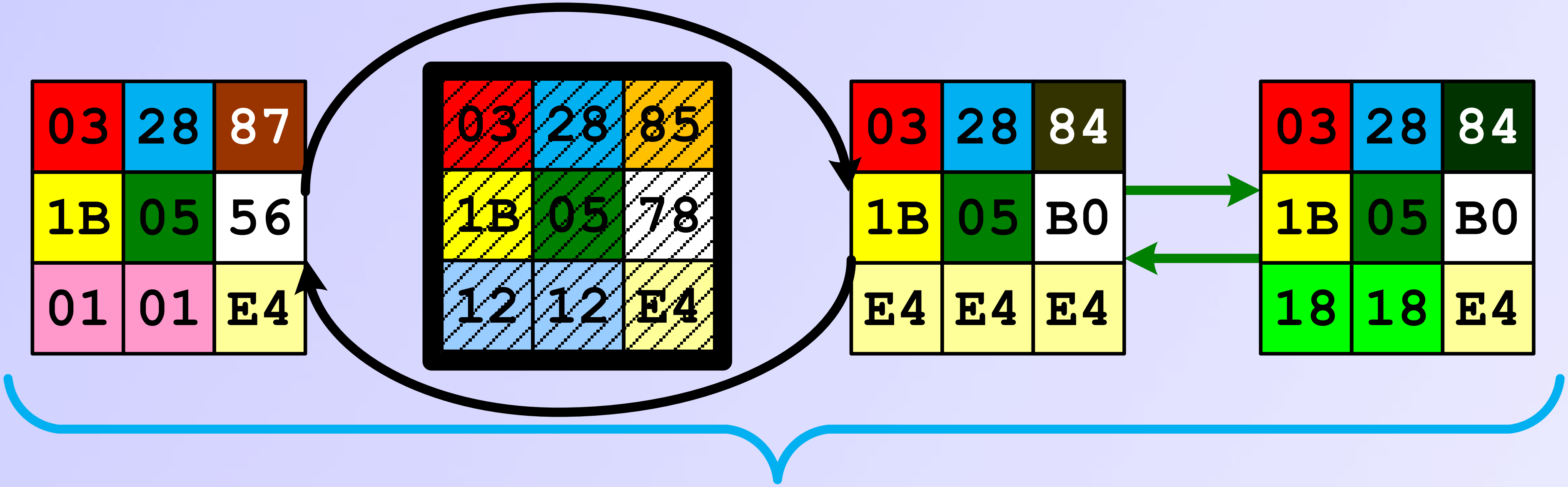
Disadvantages

vulnerable to field modifications

difficult to achieve portability

How can we improve signature scans?

Objects structures typical design



03	28	87
1B	05	56
01	01	E4

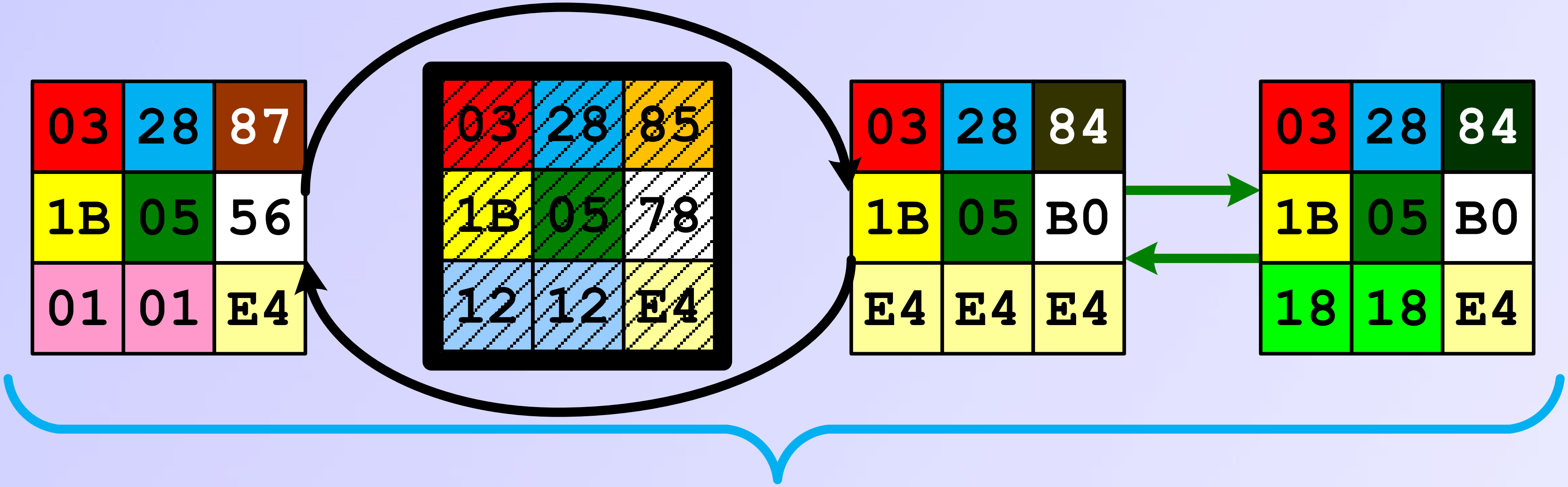
03	28	85
1B	05	78
12	12	E4

03	28	84
1B	05	B0
E4	E4	E4

03	28	84
1B	05	B0
18	18	E4

Objects structures

Objects structures typical design



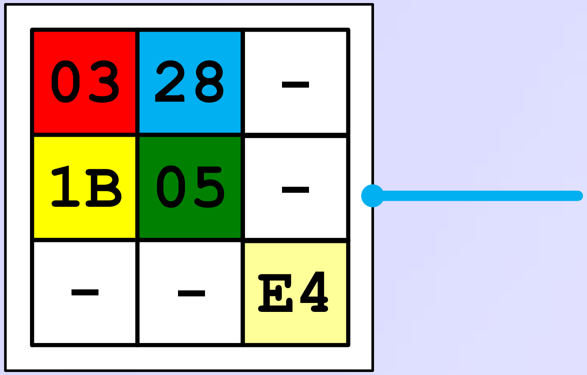
03	28	87
1B	05	56
01	01	E4

03	28	85
1B	05	78
12	12	E4

03	28	84
1B	05	B0
E4	E4	E4

03	28	84
1B	05	B0
18	18	E4

Objects structures



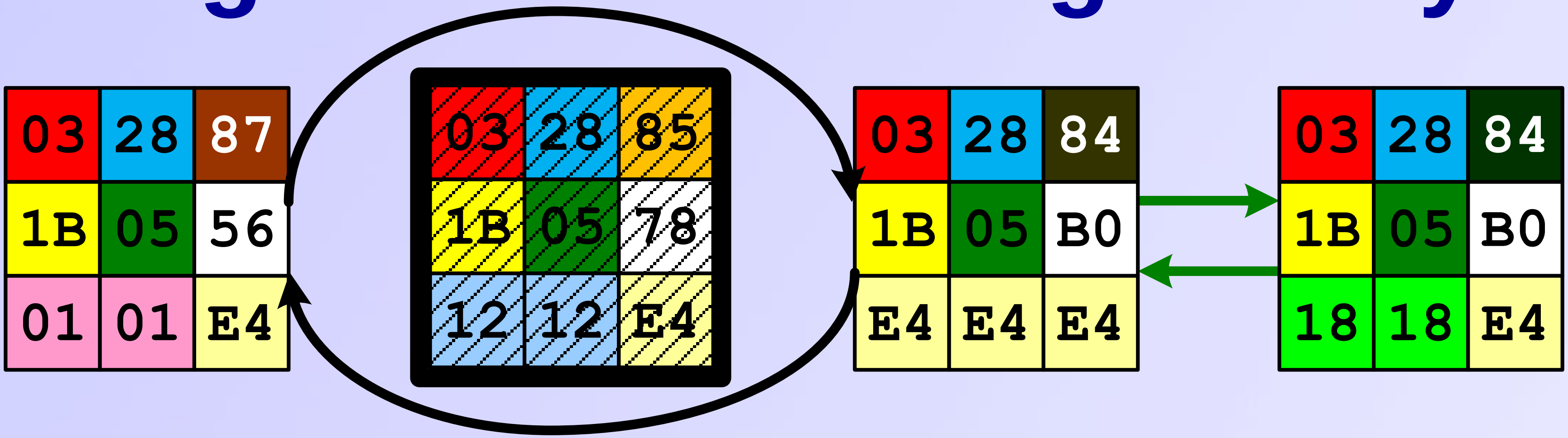
03	28	-
1B	05	-
-	-	E4

Dynamic Byte Signature memory pattern

Process detection with Dynamic Byte Signature

1. Create Dynamic Byte Signature by using EPROCESS structures in PsActiveProcessList
2. Use byte to byte DBS search to find all EPROCESS structures
3. Compare a new list with NtQuerySystemInformation list

Bit signature = thorough analysis

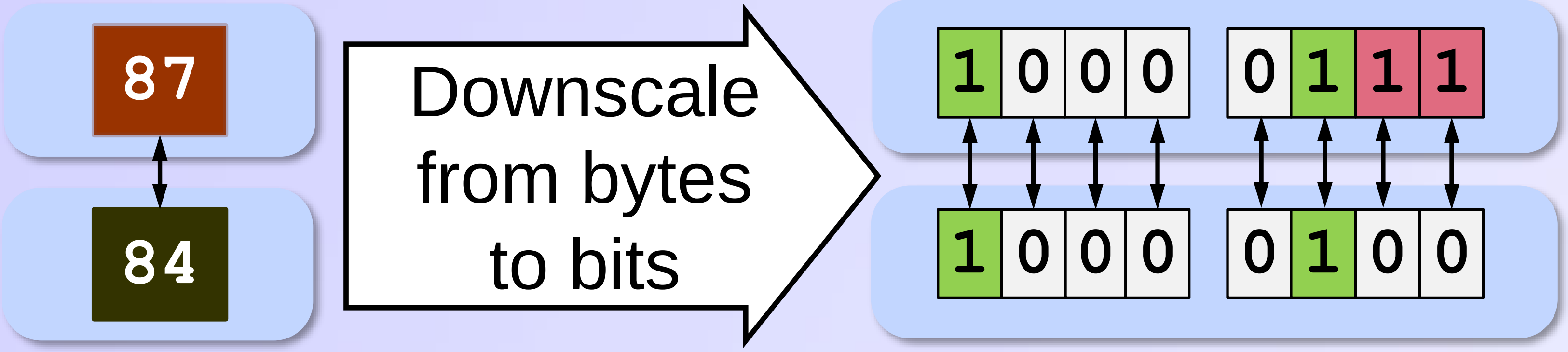


03	28	87
1B	05	56
01	01	E4

03	28	85
1B	05	78
12	12	E4

03	28	84
1B	05	B0
E4	E4	E4

03	28	84
1B	05	B0
18	18	E4



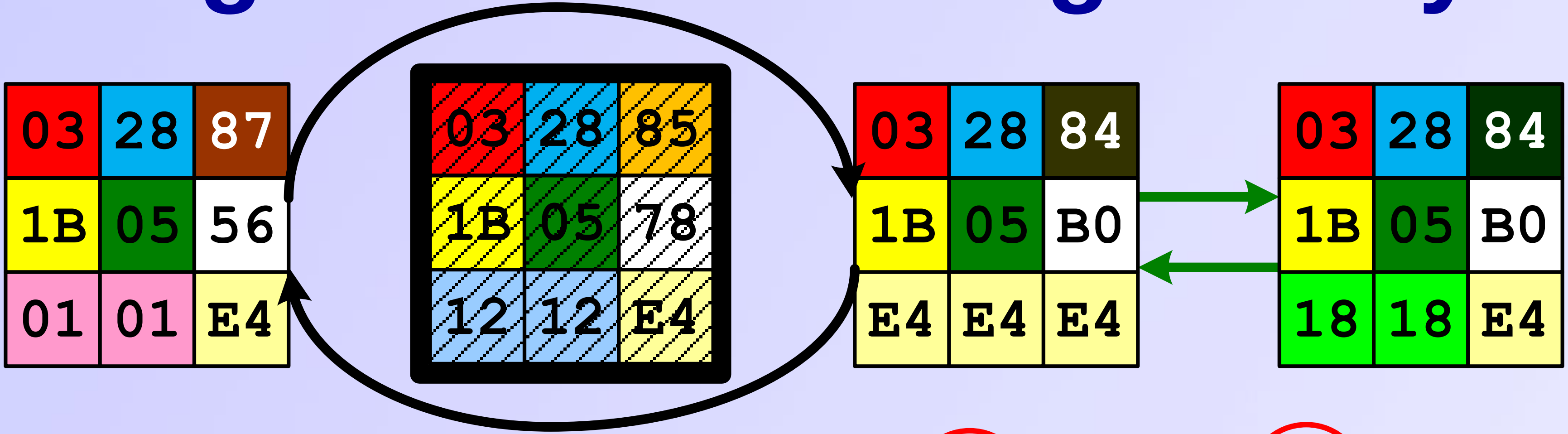
87

84

Downscale from bytes to bits

1	0	0	0	0	1	1	1
1	0	0	0	0	1	0	0

Bit signature = thorough analysis

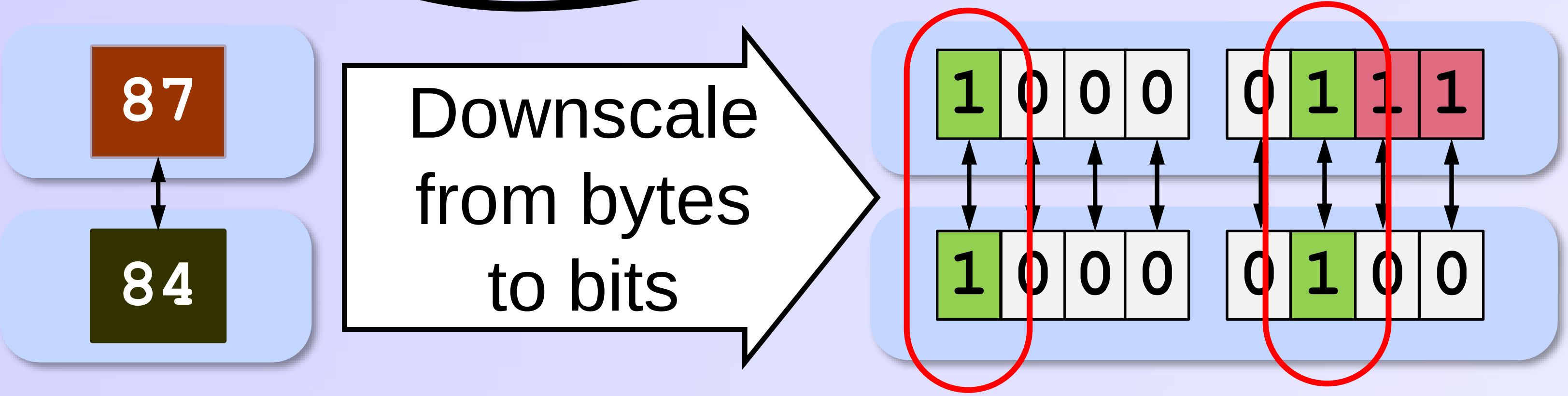


03	28	87
1B	05	56
01	01	E4

03	28	85
1B	05	78
12	12	E4

03	28	84
1B	05	B0
E4	E4	E4

03	28	84
1B	05	B0
18	18	E4



87

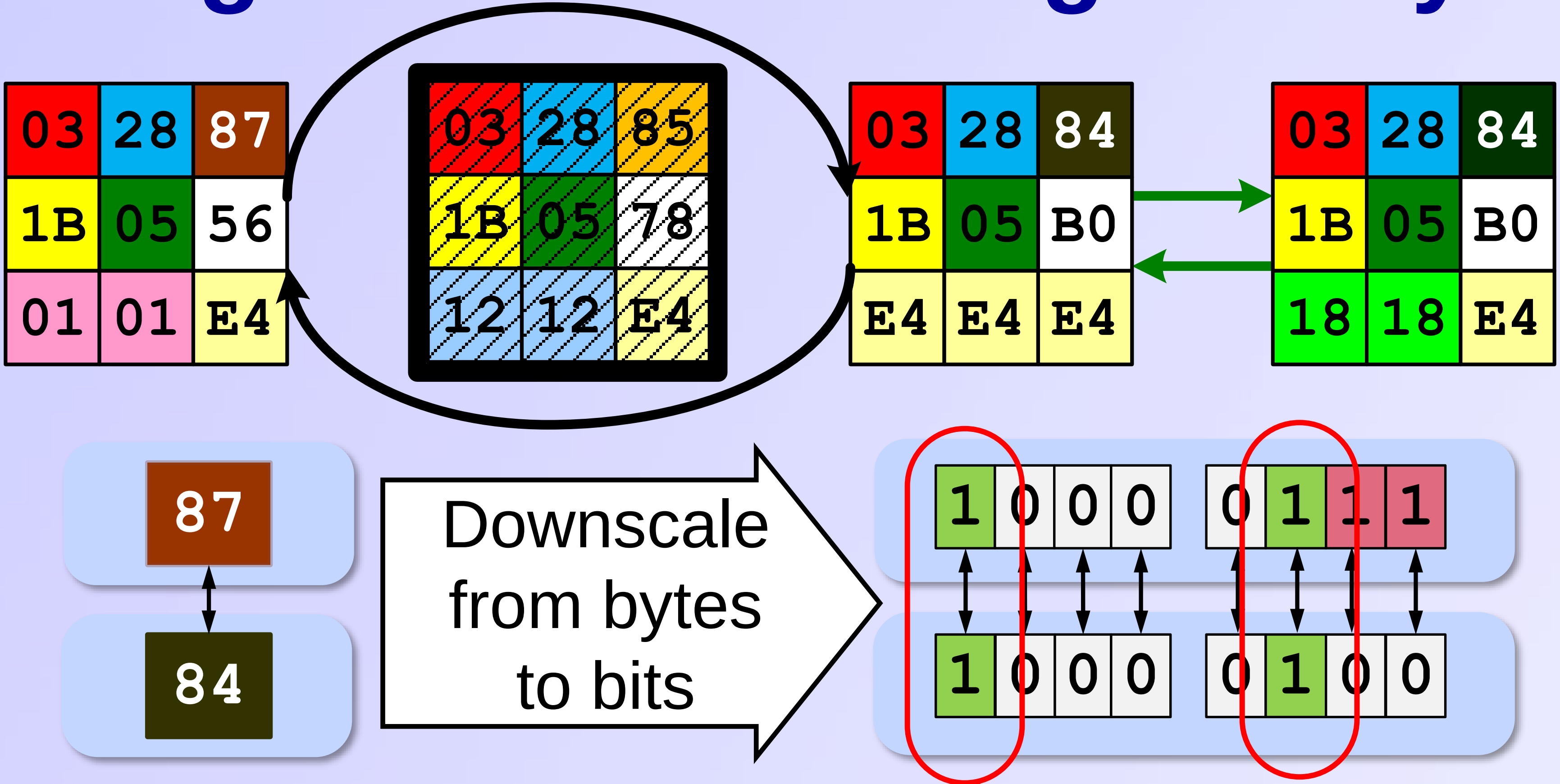
84

Downscale from bytes to bits

1 0 0 0 0 1 1 1

1 0 0 0 0 1 0 0

Bit signature = thorough analysis



03	28	87
1B	05	56
01	01	E4

03	28	85
1B	05	78
12	12	E4

03	28	84
1B	05	B0
E4	E4	E4

03	28	84
1B	05	B0
18	18	E4

Downscale from bytes to bits

Dynamic bit signature:

03	28	–
1B	05	–
–	–	E4

Dynamic Bit Signature Analysis

DBS features	**Advantages**
Automatic learning	Easily portable
Bit based analysis	More thorough analysis
Probabilistic check	Able to recognize structures even without full pattern match

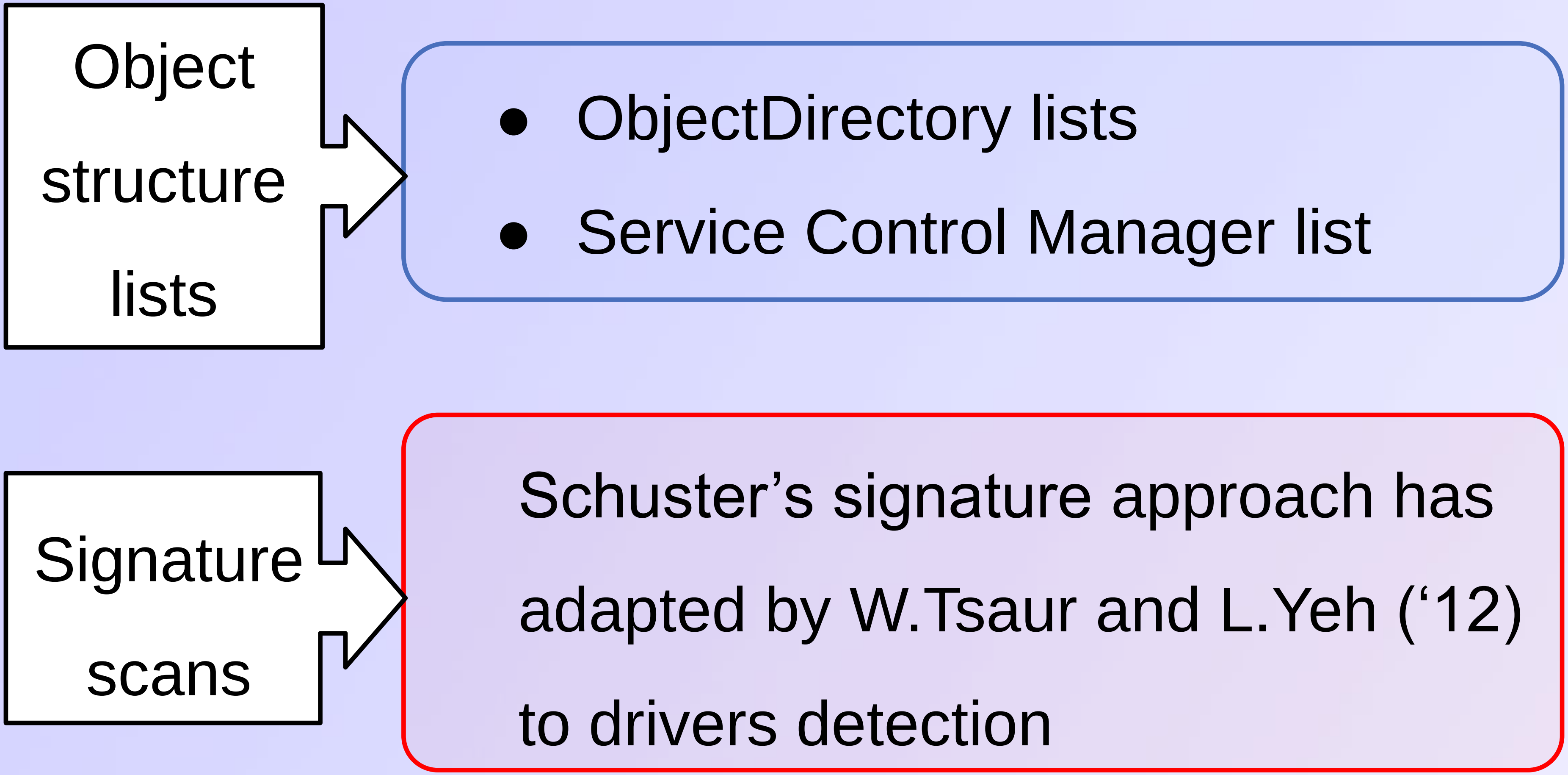
What about hidden drivers and their detection?

Hidden drivers have similar cases

	List view	Activity to hide
Processes	TaskMgr.exe	PsActiveProcessList modification
Drivers	DriverQuery.exe	PsLoadedModuleList modification

ZwQuerySystemInformation hooking leads to processes & drivers hiding

Drivers detection approaches review



Object structure lists

- ObjectDirectory lists
- Service Control Manager list

Signature scans

Schuster’s signature approach has adapted by W.Tsaur and L.Yeh (‘12) to drivers detection

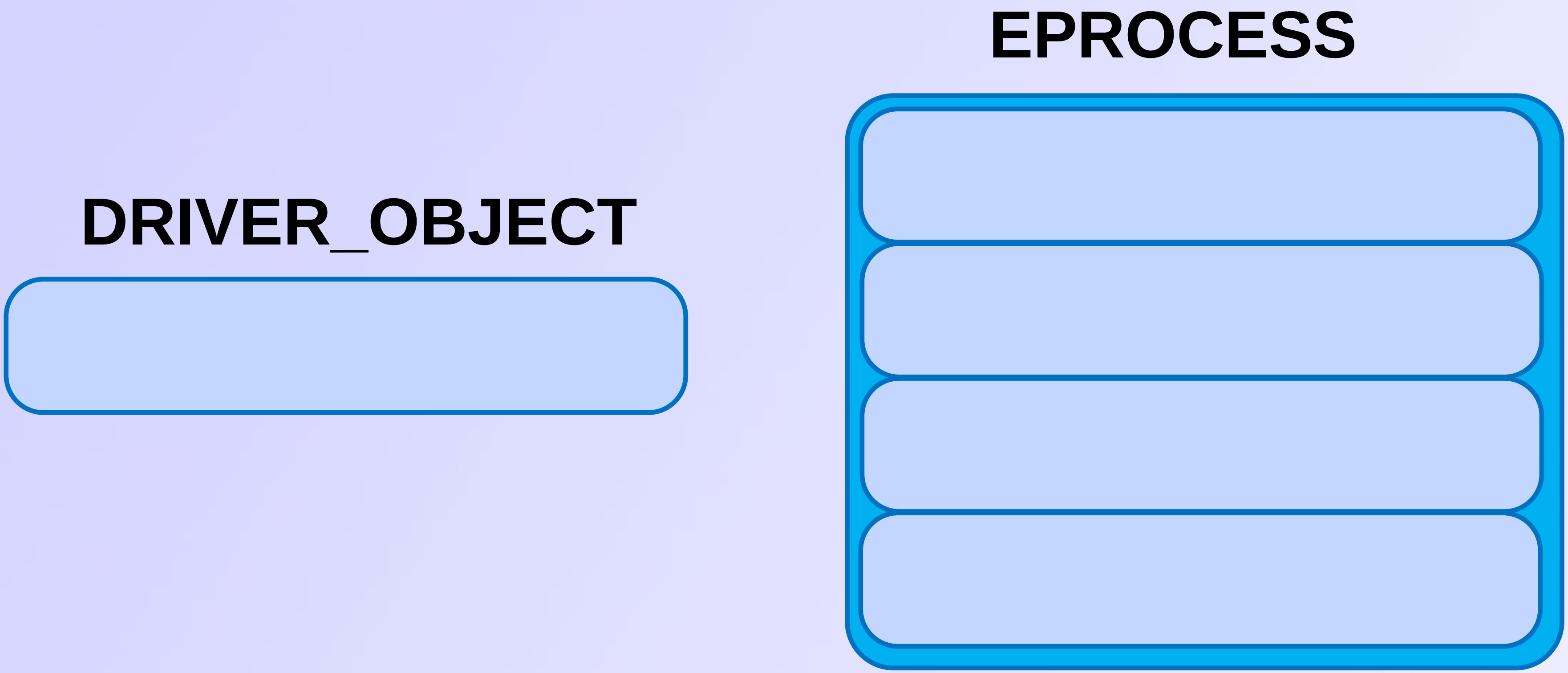
Is it possible to adapt DBS for driver detection?

Is it possible to adapt DBS for driver detection?

DBS only can detect structures with a lot of fields

Is it possible to adapt DBS for driver detection?

DBS only can detect structures with a lot of fields



Rating Point Inspection (RPI)

RPI improvements over DBS

- RPI utilizes additional weight matrix for precise pattern matching
- RPI use selective matching algorithm

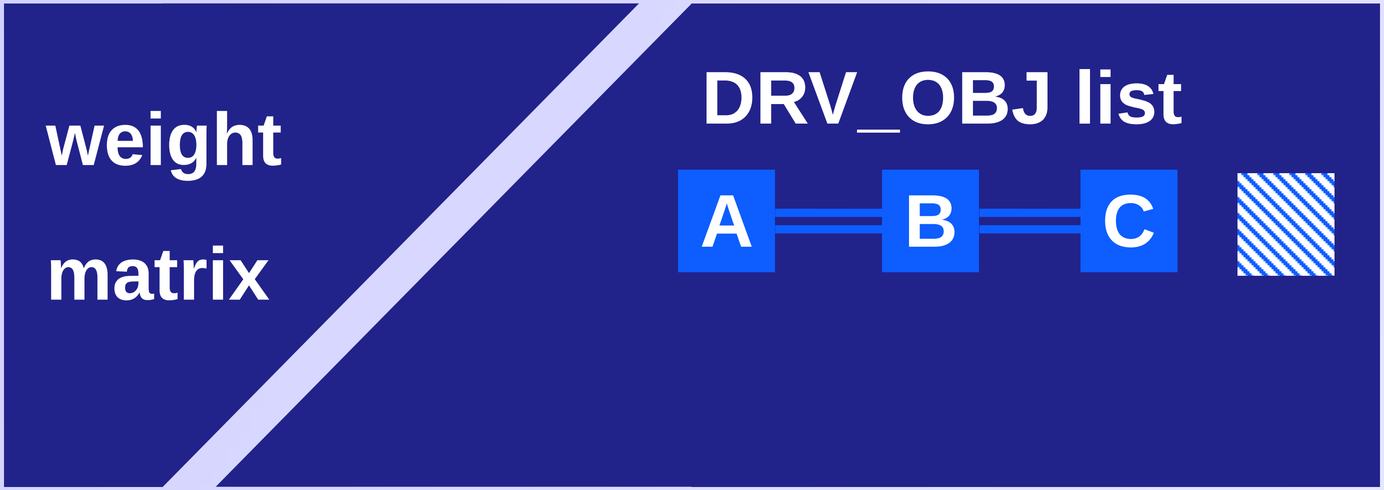
If one of the checks is true	
DBS	RPI
add 1 point	**1, 2 or etc. points** are added to the final score

Description of weight matrix for

DRIVER_OBJECT

is in the corresponding paper

How does RPI detect drivers?



weight

matrix

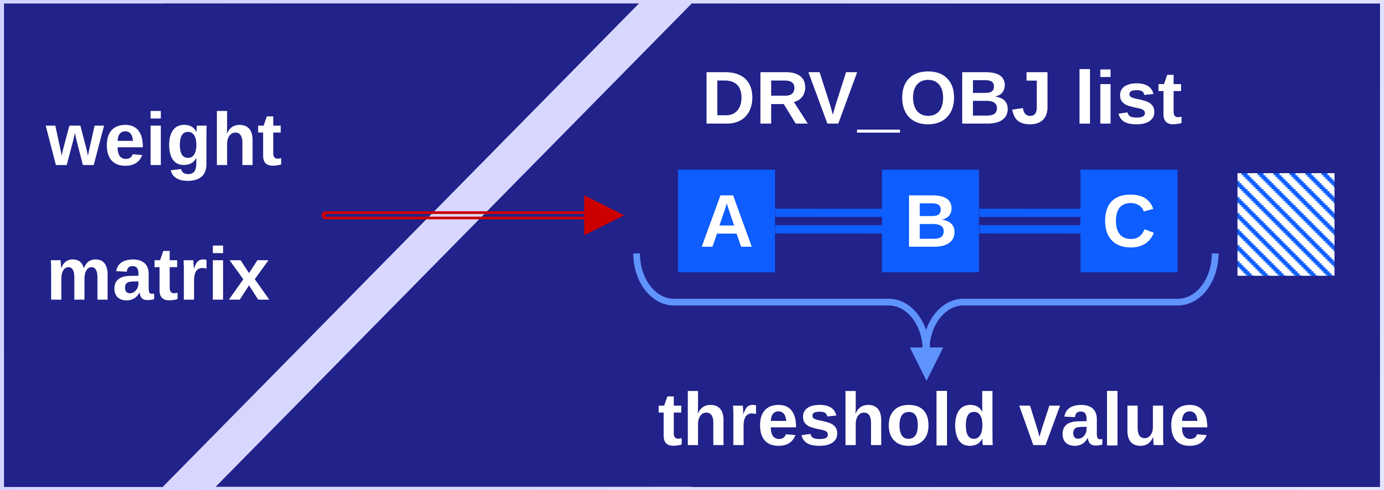
DRV_OBJ list

A

B

C

How does RPI detect drivers?



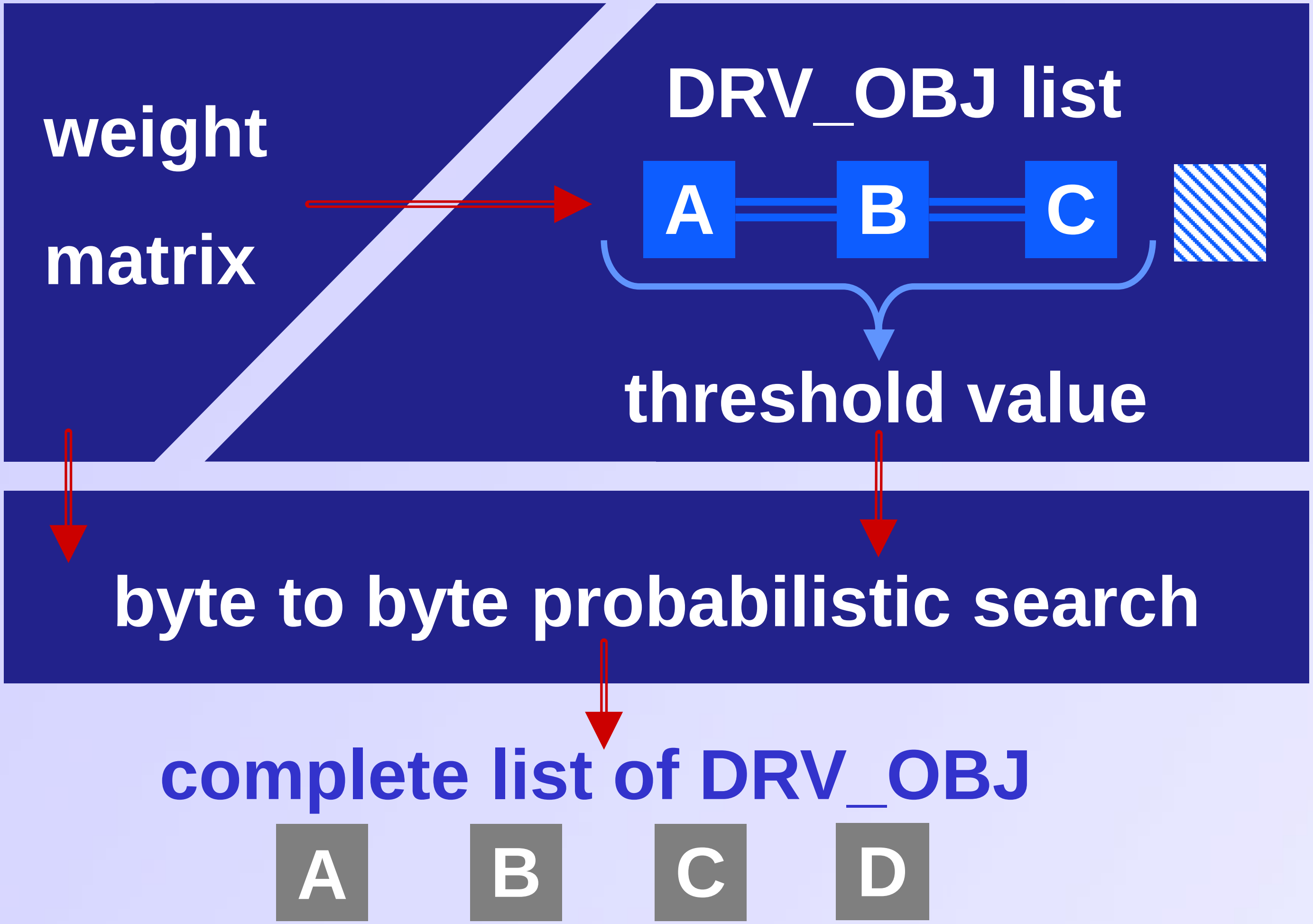
weight

matrix

DRV_OBJ list

threshold value

How does RPI detect drivers?



weight

matrix

DRV_OBJ list

A B C ▨

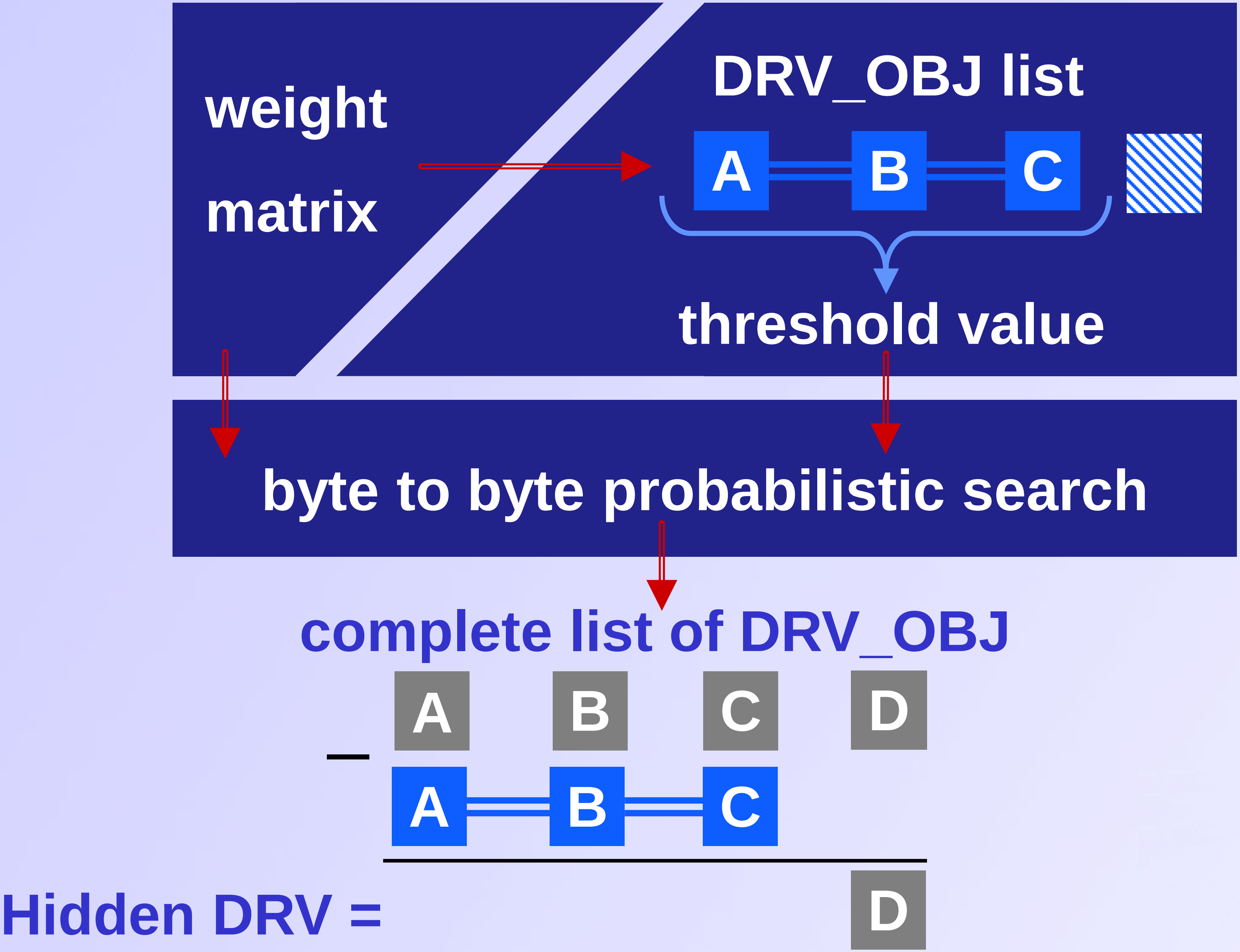
threshold value

byte to byte probabilistic search

complete list of DRV_OBJ

A B C D

How does RPI detect drivers?



weight

matrix

DRV_OBJ list

A B C

threshold value

byte to byte probabilistic search

complete list of DRV_OBJ

A B C D

–

A B C

Hidden DRV =

D

MASHKA’s achievements

Reveals rootkits:

- Deliberately hidden processes and drivers
- Virus.Win32.Sality.q
- Trojan.Win32.VB.aqt
- Hidden drivers by ATSIV

MASHKA’s achievements

Reveals rootkits:

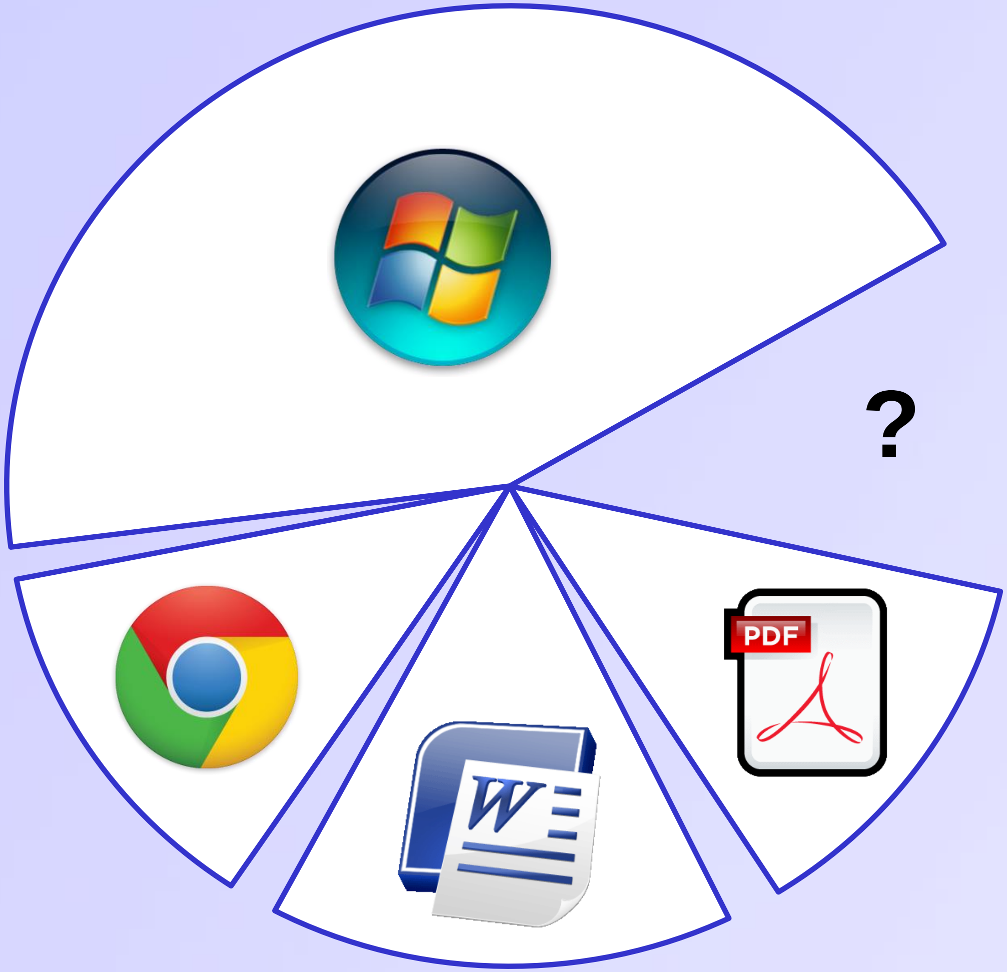
- Deliberately hidden processes and drivers
- Virus.Win32.Sality.q
- Trojan.Win32.VB.aqt
- Hidden drivers by ATSIV

Existing anti-rootkits PowerTool, TDSSKiller, Xuetr fail,

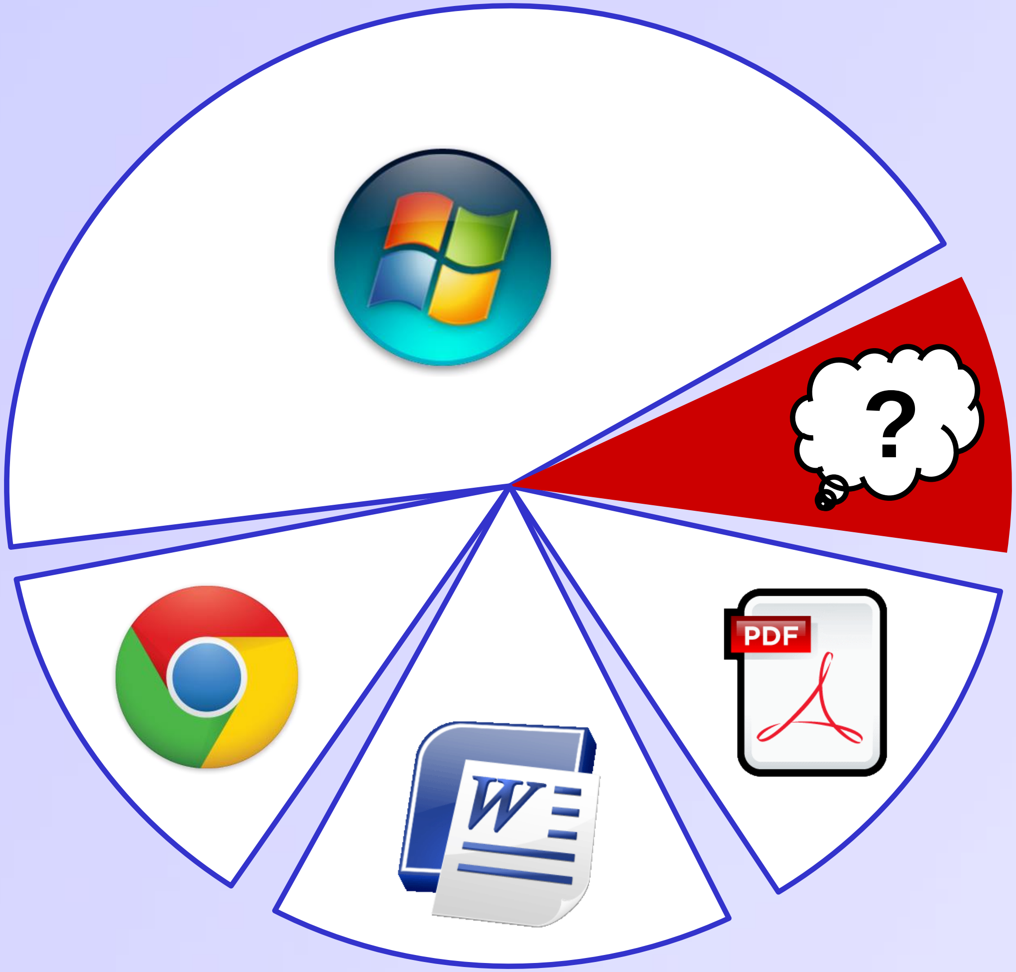
but MASHKA can detect them

`Demo: bit.ly/win8t6st`

What is the pie filling?



What is the pie filling?



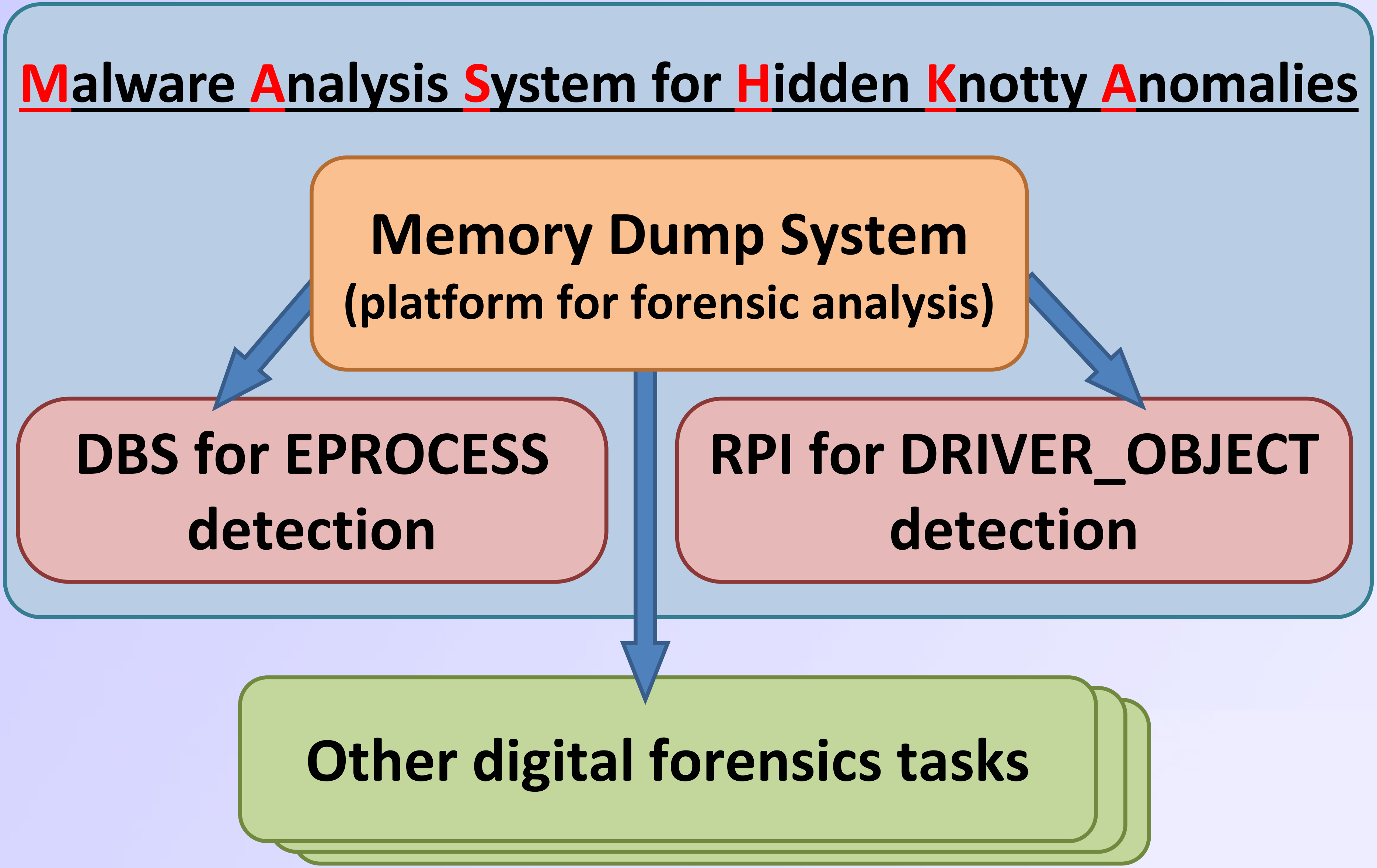
Igor Korkin, Ph.D

igor.korkin@gmail.com

sites.google.com/site/iykorkin

ADDITIONAL

WHAT IS IT MASHKA?



Malware Analysis System for Hidden Knotty Anomalies

Memory Dump System
(platform for forensic analysis)

DBS for EPROCESS detection

RPI for DRIVER_OBJECT detection

Other digital forensics tasks

MASHKA IN MEMORY FORENSICS TASKS

Various search signatures: char and wide char strings, byte fragments include addresses

As a result we receive:

What can we do with it?

Name	**Definition**	
VALF	**v**irtual **a**ddress of the **l**oaded dump **f**ile	read data
ODUF	corresponding **o**ffset in **du**mp **f**ile	calculate offsets
VAOM	**v**irtual **a**ddress of the **o**riginal **m**emory	find value in dump

USE MASHKA TO RESEARCH DRIVERS

1. **Run Windows under WinDbg control**

2. **Install a test driver with ‘ServiceName’, ‘DisplayName’ and ‘BinaryPath’**

3. **Hide this driver structure by unlinking from PsLoadedModuleList**

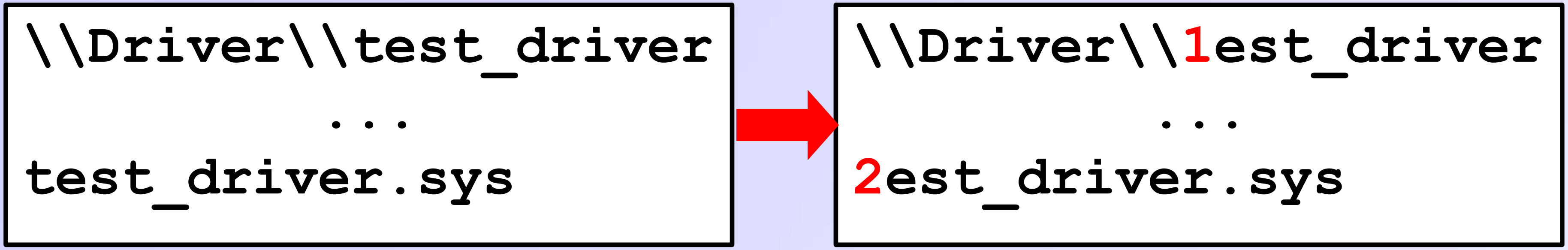
4. **Check the system with anti-rootkit tool**

5. **Dump memory with the help of MASHKA**

USE MASHKA TO RESEARCH DRIVERS

6. Search strings from step 2 and save their ‘VAOM’

7. By WinDbg and strings VAOM change their content



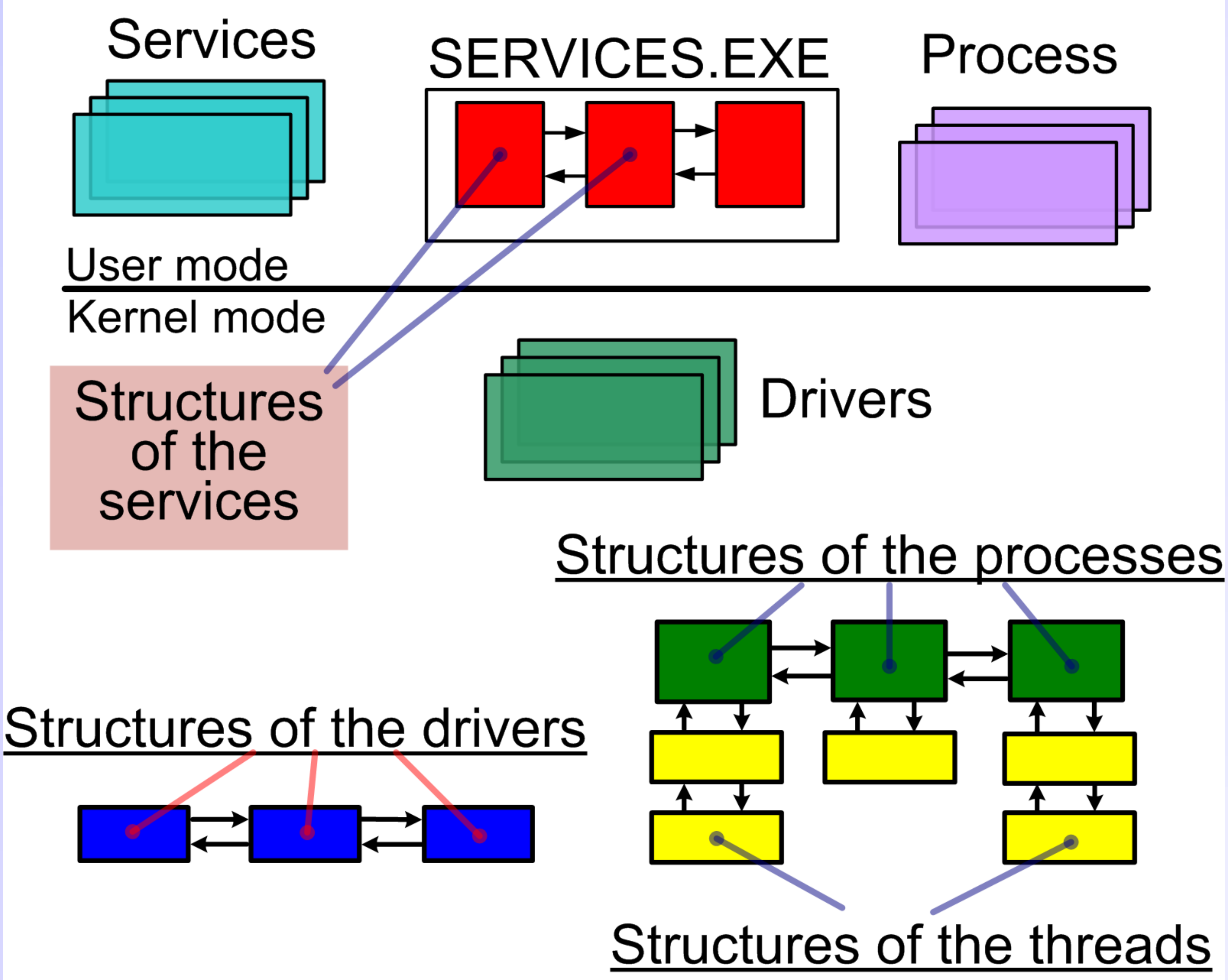
```
\\Driver\\test_driver
         ...
test_driver.sys
```

```
\\Driver\\1est_driver
         ...
2est_driver.sys
```

8. Check the system repeatedly. Detection tools will give us a changed name.

By known ‘VAOM’ run further analysis

PROBLEM STATEMENT - ROOTKITS IN WINDOWS


Services
SERVICES.EXE
Process
User mode
Kernel mode
Structures of the services
Drivers
Structures of the processes
Structures of the drivers
Structures of the threads

ANALYSIS OF CURRENT APPROACHES TO DETECTION IN FACE OF OPPOSITIONS

Cross-view detection is the main point for all tools

Low-level mechanisms:

- **<u>Heuristic analyzer</u>**
- **<u>Additional object structure lists</u>**
- **<u>Signature scans</u> are based on byte to byte search of fragments of objects structures in memory**

ANALYSIS OF SIGNATURE SCANS

- The fact that some fields' values are either known or exceed the constant, for example 0x8000_0000
- Parts of this method are implemented in the popular tools such as *GMER*, *PowerTool*, *XueTr*

Method's disadvantages:

- vulnerable to field modifications: If at least one byte does not match, the signature scan will miss the structure
- difficult to achieve portability on different versions of Windows OS, as it requires a lot of manual work

RPI FOR DRIVER_OBJECT DETECTION

Condition	Score
if (DRIVER_OBJECT_32.Type == 0x04)	1
if (DRIVER_OBJECT_32.Size == 0xa8)	1
if (chk_unicode_string(&DRIVER_OBJECT_32.DriverName))	2
if (chk_unicode_string(DRIVER_OBJECT_32.HardwareDatabase))	2
if ((DRIVER_OBJECT_32.MajorFunction[0]) >> 31)	2
if (max_same_major_functions(&DRIVER_OBJECT_32) >= min_major_function)	2
check_function_prologue(addr)	4

‘global_scope’ is a sum of points

THE ‘CHECK_FUNCTION_PROLOGUE (ADDR)’ FUNCTION

<table>
<tr><th>Condition</th><th>Result</th></tr>
<tr><td>If (((addr[i+0] == 0x55) && (addr[i+1] == 0x89) && (addr[i+2] == 0xe5)) ||
((addr[i+0] == 0x55) && (addr[i+1] == 0x8b) && (addr[i+2] == 0xec)) ||
((addr[i+0] == 0x53) && (addr[i+1] == 0x56)) ||
((addr[i+0] == 0x56) && (addr[i+1] == 0x57)) ||
((addr[i+0] == 0x56) && (addr[i+1] == 0x57)) ||
((addr[i+0] == 0x8b) && (addr[i+1] == 0xff)))</td><td>true or false</td></tr>
</table>

RPI APPLYING

1. Calculate all values, such as ‘min_major_function’ and ‘global_scope’

2. Perform a byte-to-byte search by calculating the sum of points for each memory region

3. DRIVER_OBJECT structure is found if the probabilistic comparing of matching points with the ‘global_scope’ value is true

4. Compare the RPI-matching list with the drivers list, which has been obtained by ZwOpenDirectoryObject

FUTURE PLANS OF HOW TO USE & IMPROVE MASHKA

- Detection Shadow Walker-like Rootkits
- GPU Utilization in Memory Forensics
- The Idea of Cloud Anti-Rootkit or Anti-Rootkit as a Service
- The Center of Mass of Kernel Mode Structures
- Digital Forensics in Education

TESTING RESULTS OF MASHKA

<table>
<tr><th colspan="2">DBS approach has been successfully tested</th></tr>
<tr><td rowspan="2">deliberately hidden objects</td><td>real rootkits:</td></tr>
<tr><td>• Virus.Win32.Sality.q (Kaspersky Lab)
• Trojan.Win32.VB.aqt (Kaspersky Lab)</td></tr>
</table>

<table>
<tr><th colspan="3">RPI approach has been successfully tested</th></tr>
<tr><td>deliberately hidden objects</td><td>real rootkits</td><td>for hidden drivers which were loaded by <i>ATSIV</i> (Linchpin Labs)</td></tr>
</table>

In the latter case popular tools such as PowerTool, TDSSKiller, Xuetr cannot detect a hidden driver, but the RPI can

Demo - **bit.ly/win8t6st**

CONCLUSIONS

- Level of sophisticated malware increases
- Vulnerability of Windows OS
- Popular dump systems are vulnerable to intruder attacks
- Popular anti-rootkits are stopped by malware
- To prevent a possible attack, continue to maintain systems

CONCLUSIONS

- Use the page tables to memory dump
- Dynamic bit signatures can detect structures which have a typical design with a lot of members
- Rating point inspection can detect structure by detailed analysis of its members