Dedup Est Machina

+

Flip Feng Shui

Erik Bosman

VU VRIJE UNIVERSITEIT AMSTERDAM

WARNING

THIS PRESENTATION
MAY CONTAIN POINTERS

Part I: Dedup est Machina

Part I: Dedup est Machina

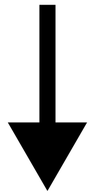
Deduplication
(software side-channel)

Part I: Dedup est Machina

Deduplication
(software side-channel)
+
Rowhammer
(hardware bug)

Part I: Dedup est Machina

Deduplication
(software side-channel)
+
Rowhammer
(hardware bug)



Exploit MS Edge without software bugs
(from JavaScript)

Part I: Dedup est Machina

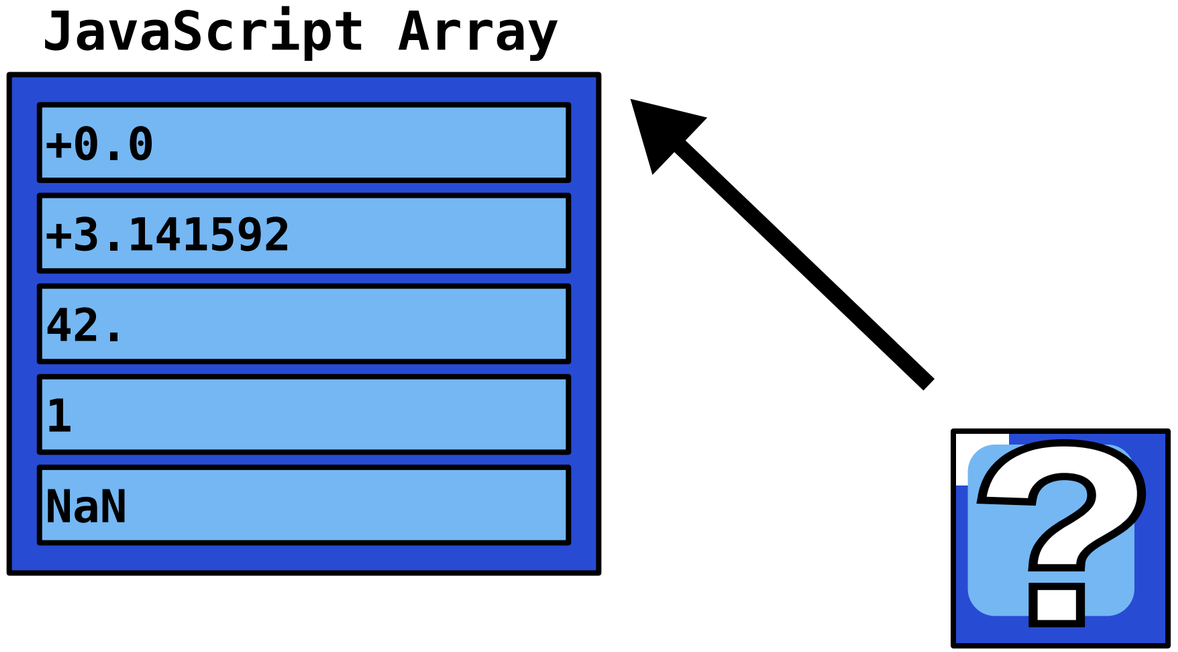
Deduplication
(software side-char
+
Rowhammer
(hardware bug)

Exploit MS Edge without software bugs
(from JavaScript)

Outline:

Deduplication

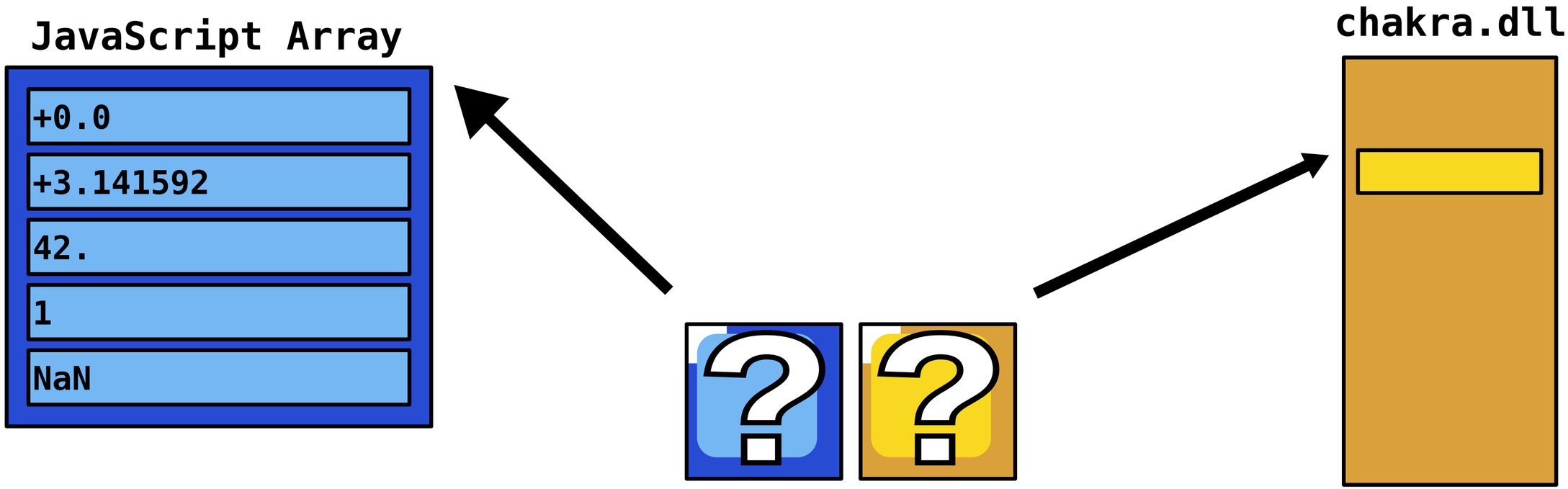
- leak heap & code addresses



Outline:

Deduplication

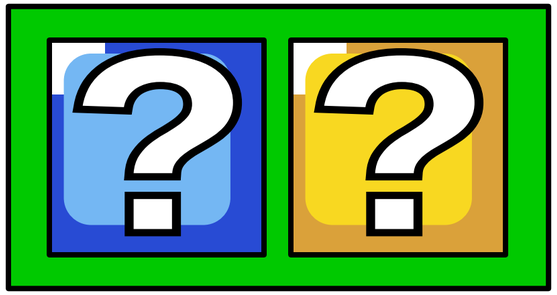
- leak heap & code addresses



Outline:

Deduplication

- leak heap & code addresses
- create a fake object



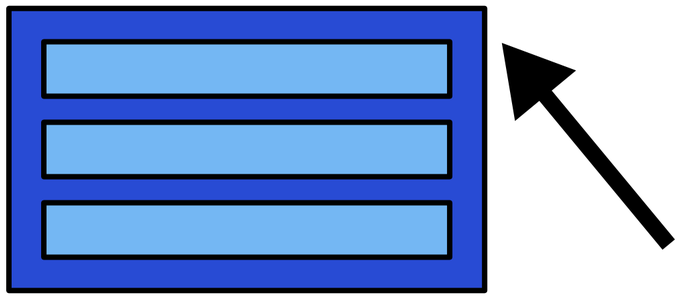
Outline:

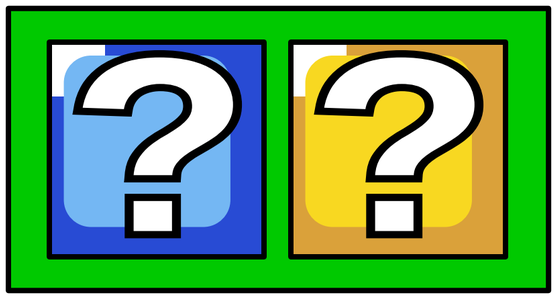
Deduplication

- leak heap & code addresses
- create a fake object

Rowhammer

- create reference to our fake object





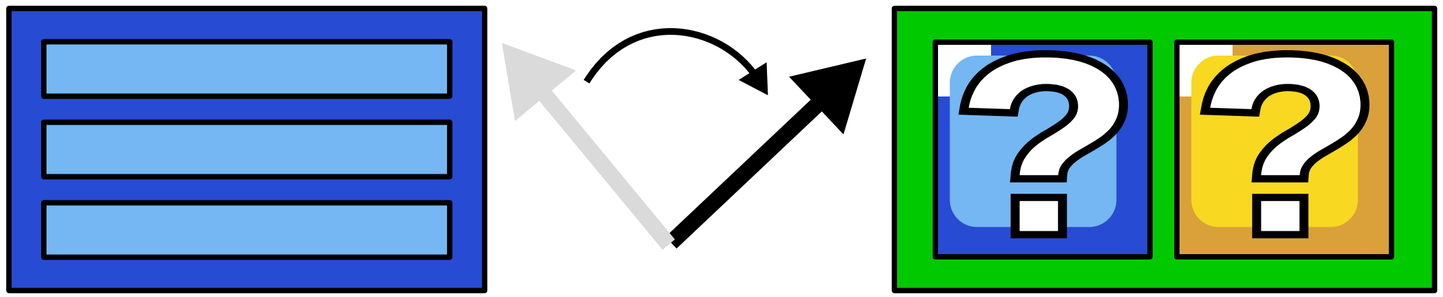
Outline:

Deduplication

- leak heap & code addresses
- create a fake object

Rowhammer

- create reference to our fake object



memory deduplication

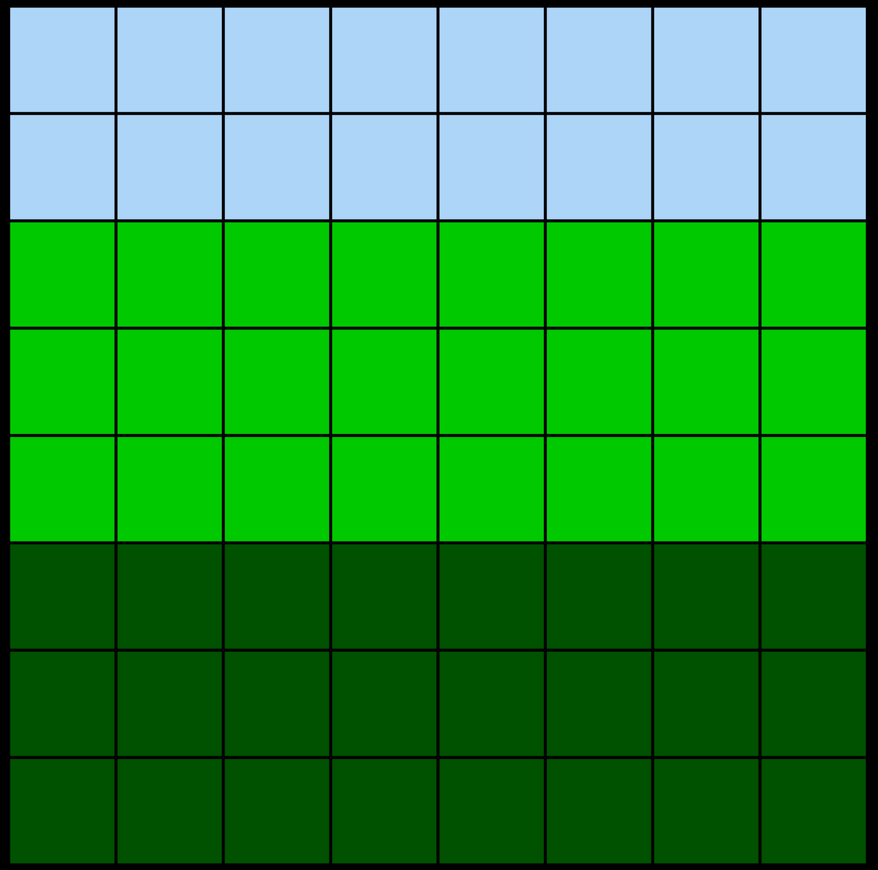
A method of reducing memory usage.

Used in virtualisation environments,

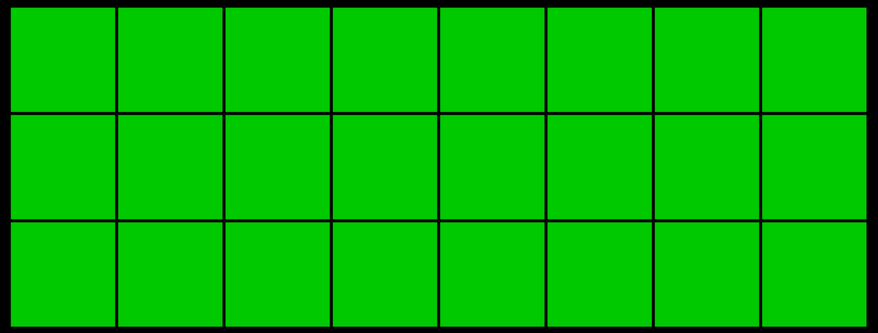
(was) also enabled by default on Windows 8.1 and 10.

memory deduplication

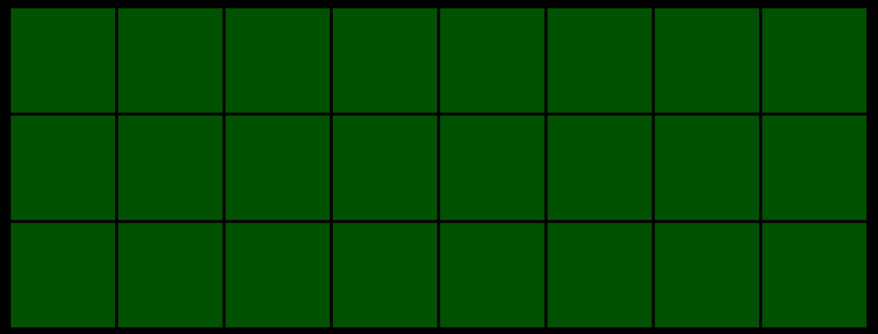
physical memory



process A

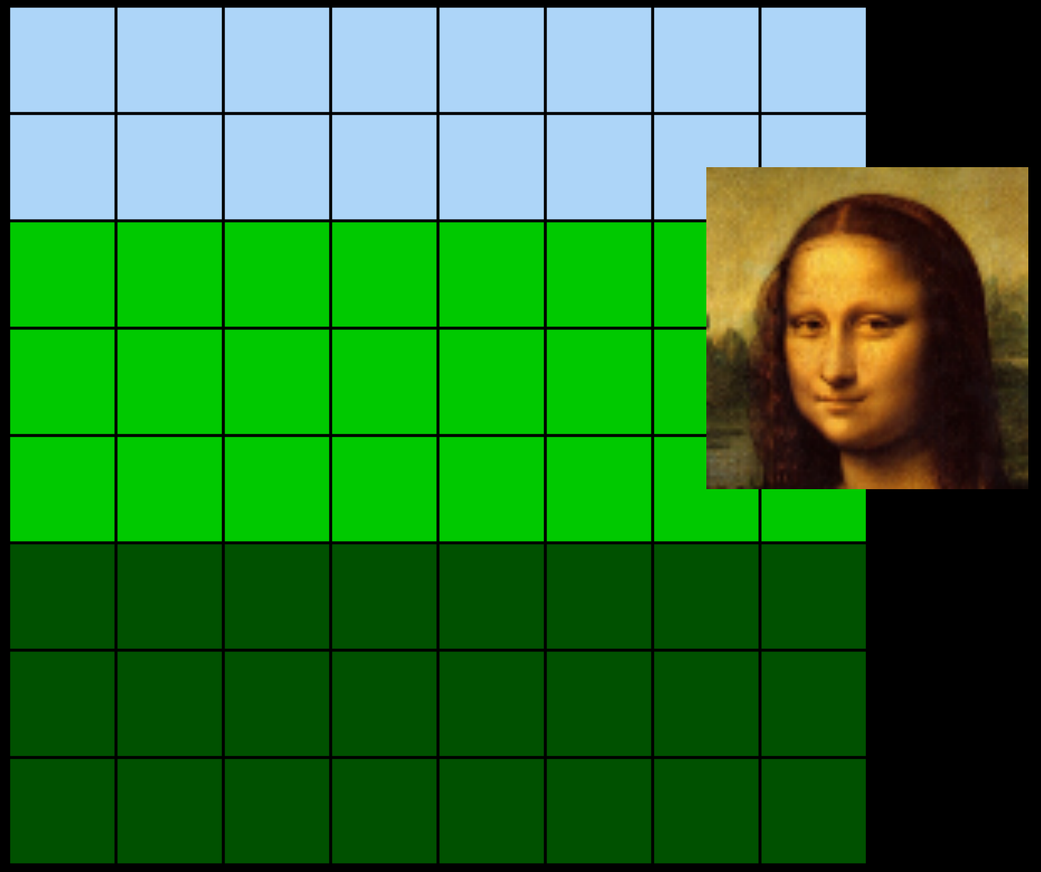


process B

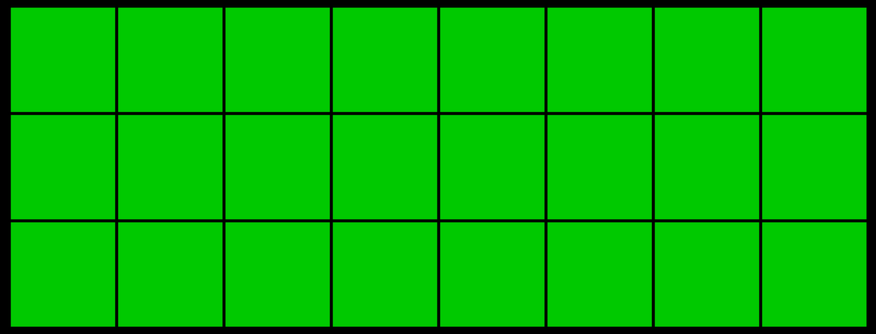


memory deduplication

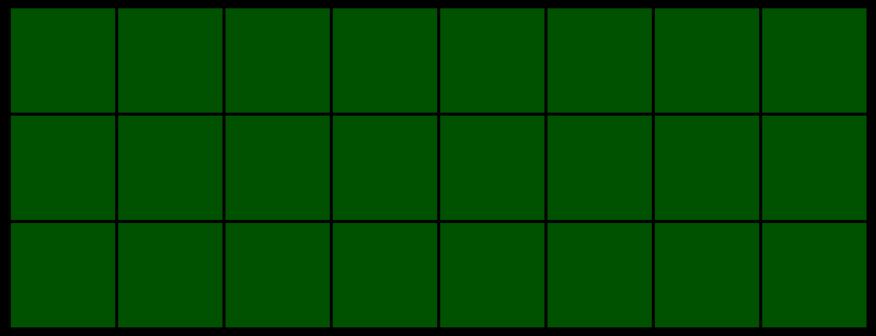
physical memory



process A

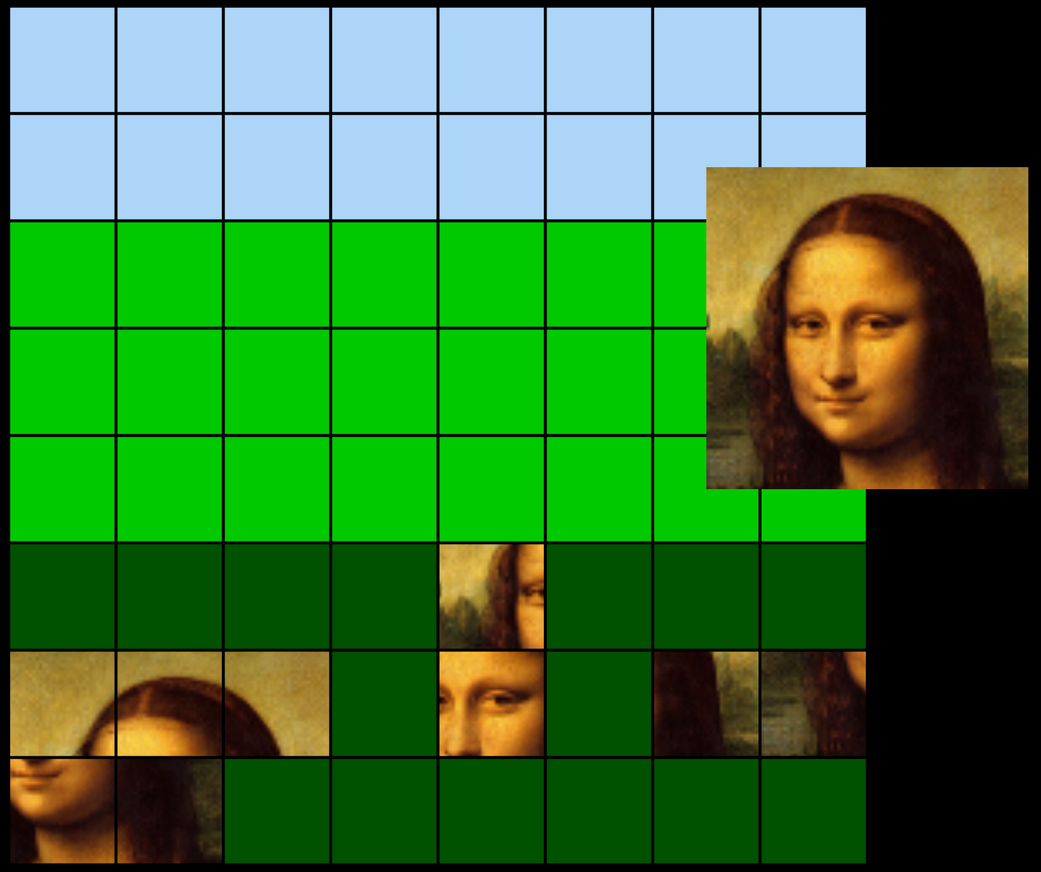


process B

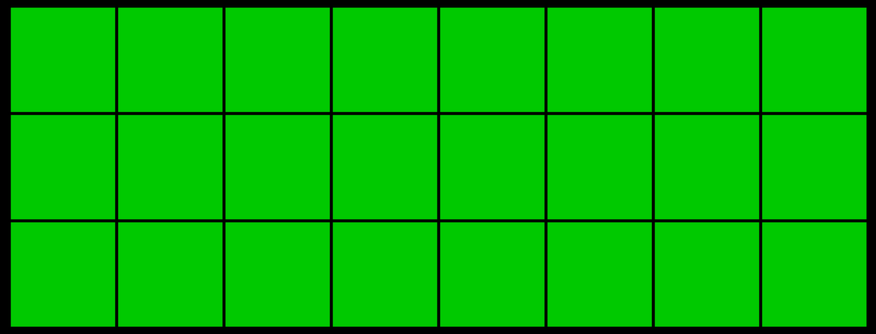


memory deduplication

physical memory



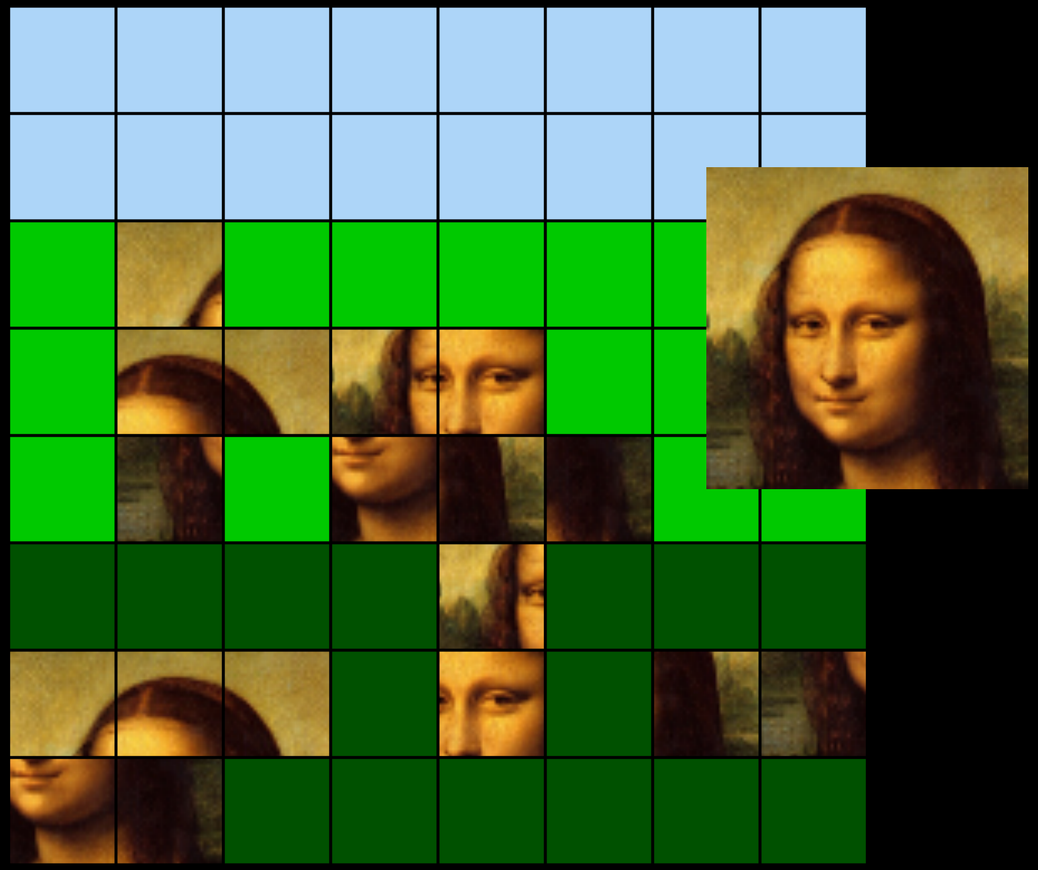
process A



process B

memory deduplication

physical memory

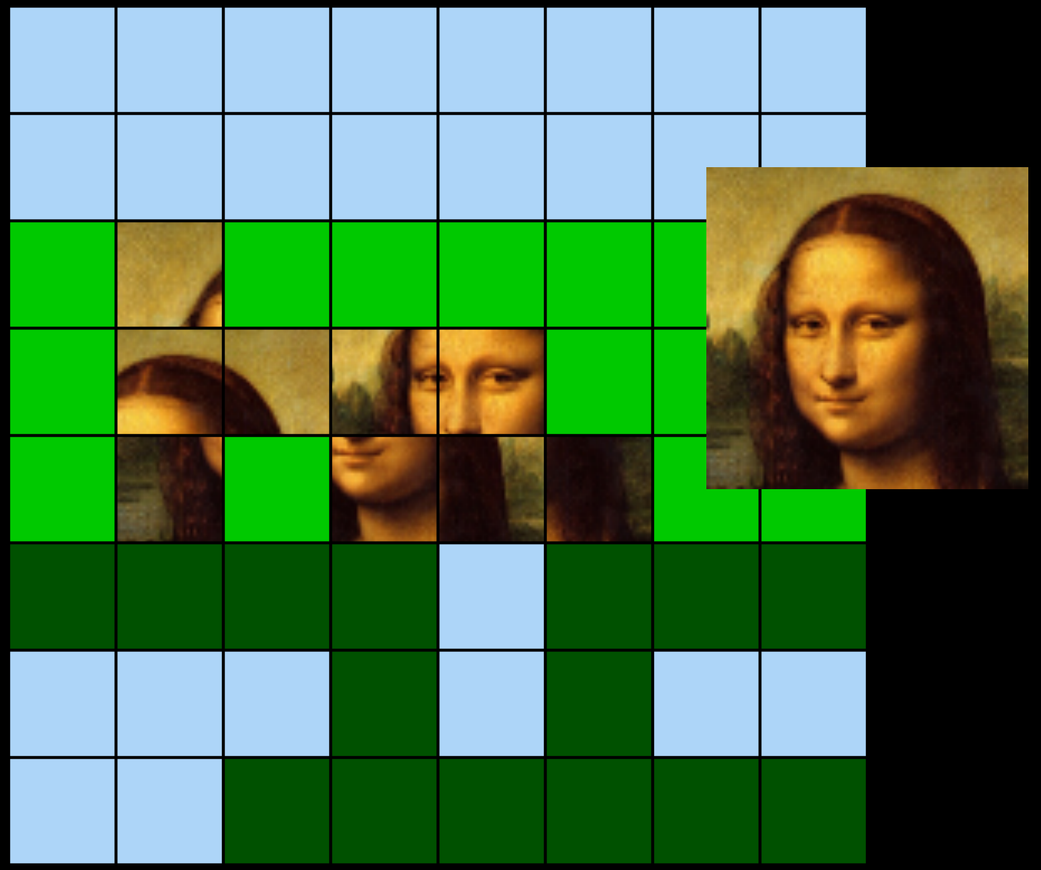


process A

process B

memory deduplication

physical memory

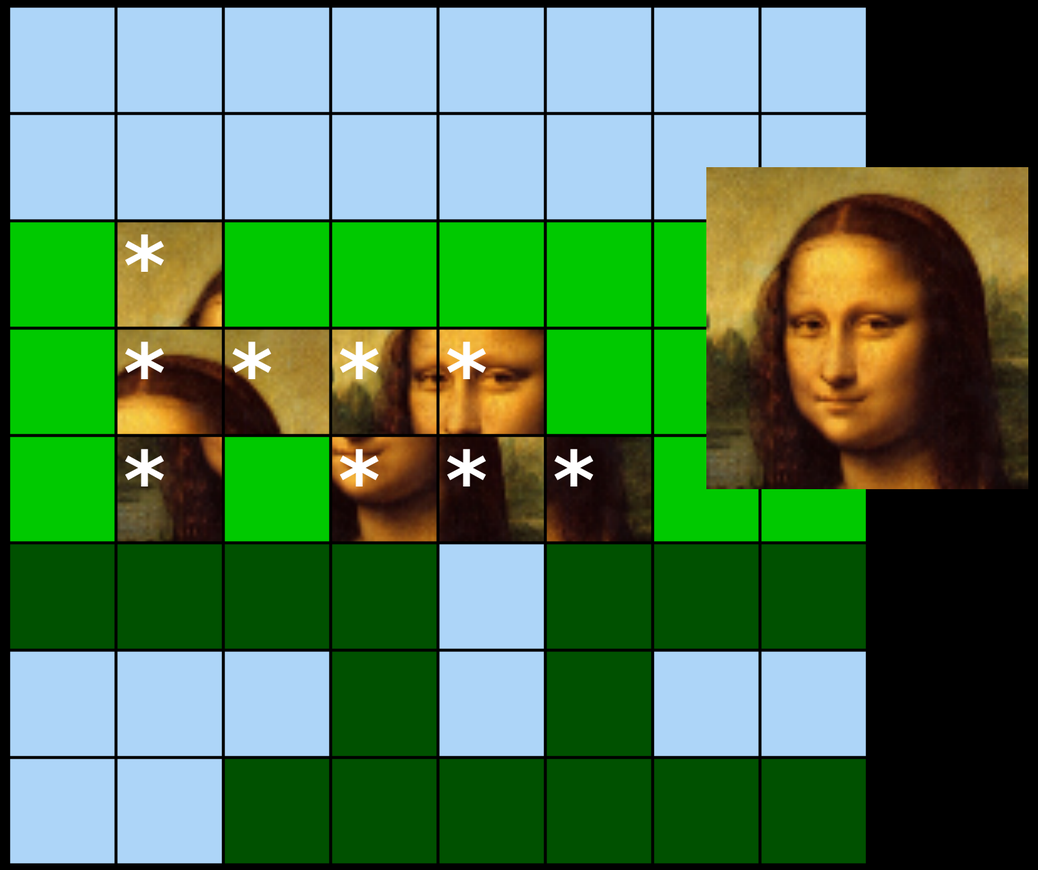


process A

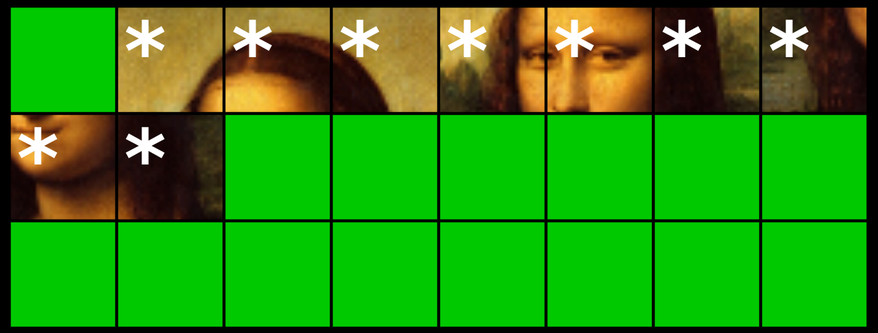
process B

memory deduplication

physical memory



process A



process B

memory deduplication: The Problem

Deduplicated memory does not need to have the same origin.

(unlike fork(), file-backed memory)

An attacker can use deduplication as a side-channel

deduplication side-channel attack

normal write

deduplication side-channel attack

normal write

deduplication side-channel attack

normal write

copy on write (due to deduplication)

deduplication side-channel attack

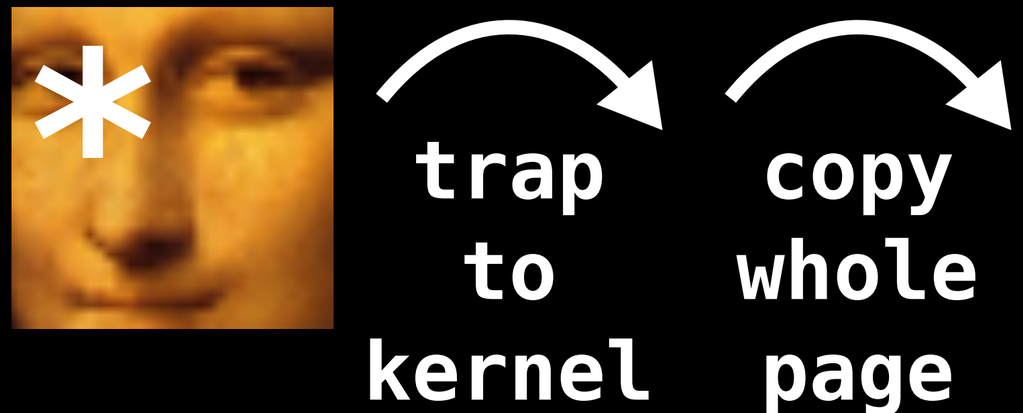
normal write

copy on write (due to deduplication)

deduplication side-channel attack

normal write

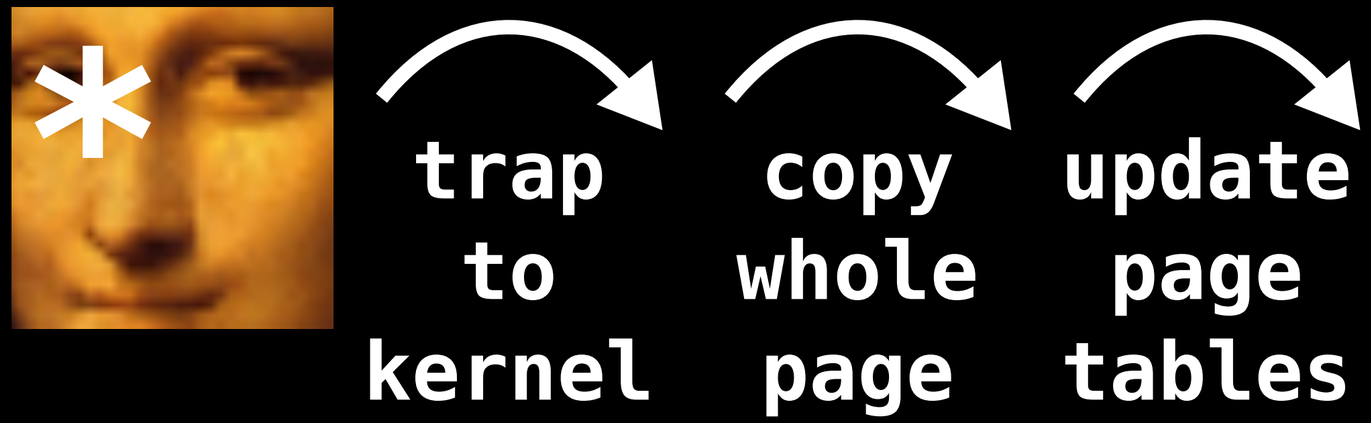
copy on write (due to deduplication)



deduplication side-channel attack

normal write

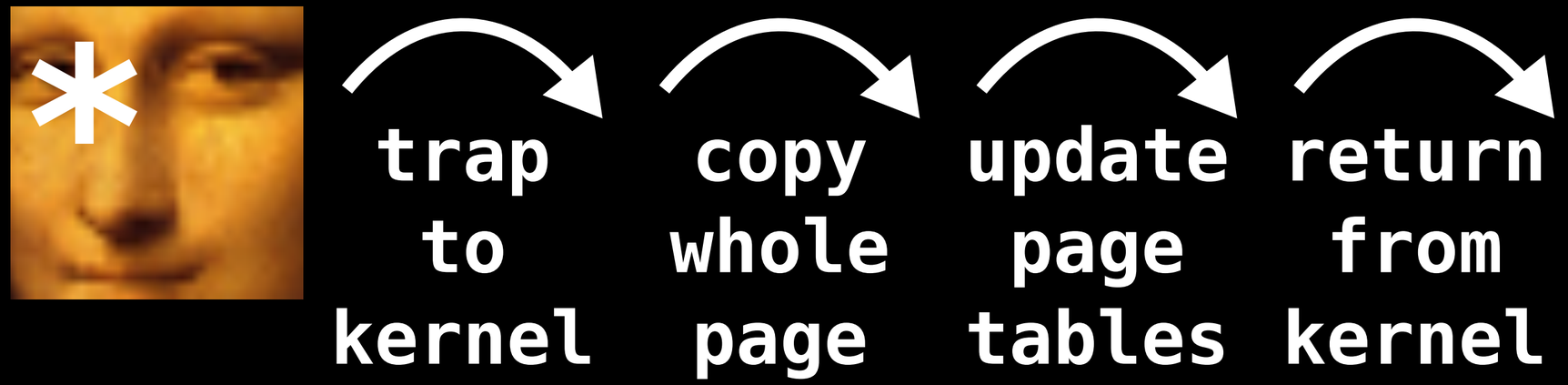
copy on write (due to deduplication)



deduplication side-channel attack

normal write

copy on write (due to deduplication)



deduplication side-channel attack

normal write

copy on write (due to deduplication)

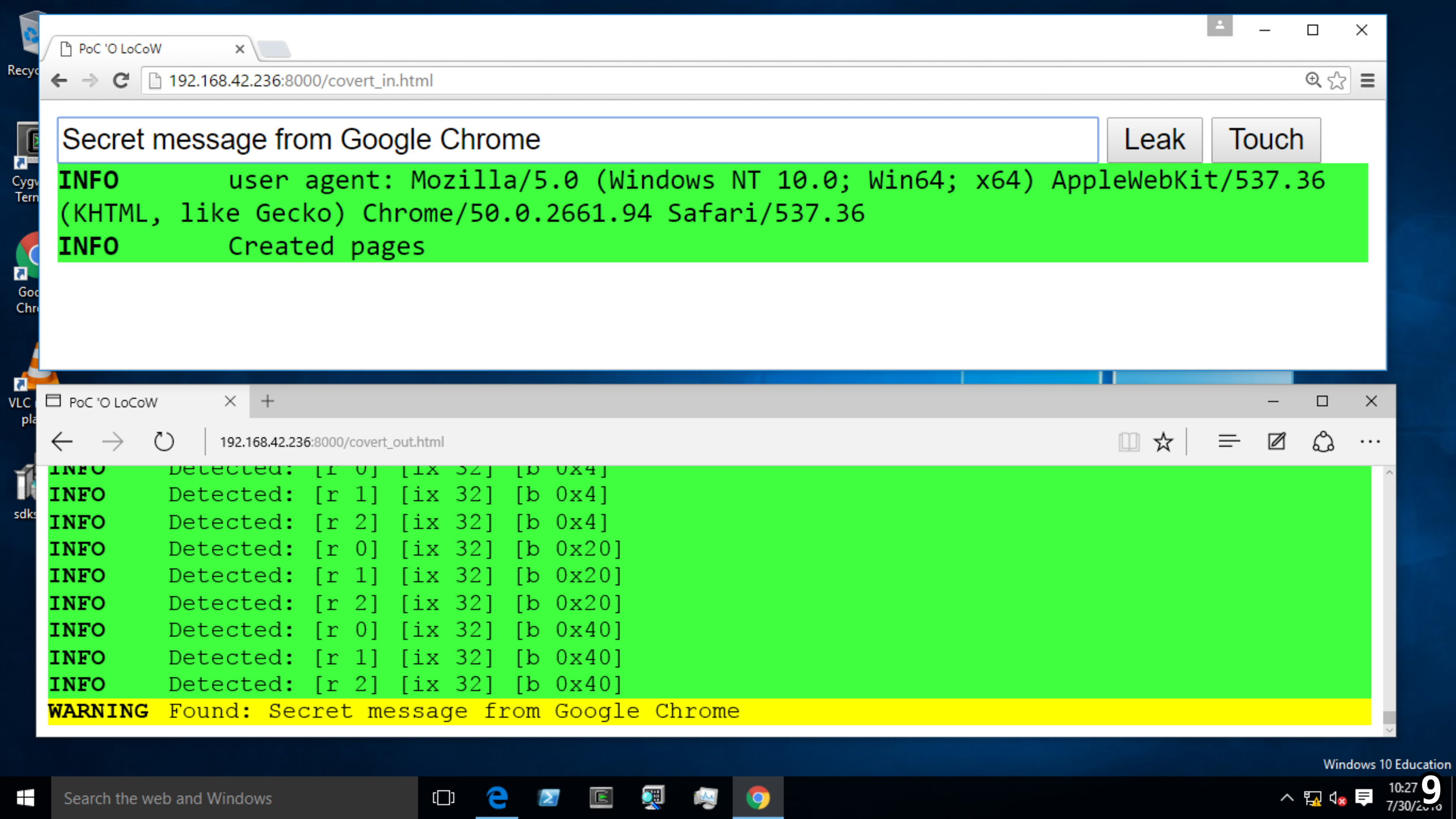
Deduplication side-channel attack

A 1-bit side channel which is able
to leak data across security boundaries

> cross VM
> cross-process
> leak process data from javascript code

Having fun with deduplication

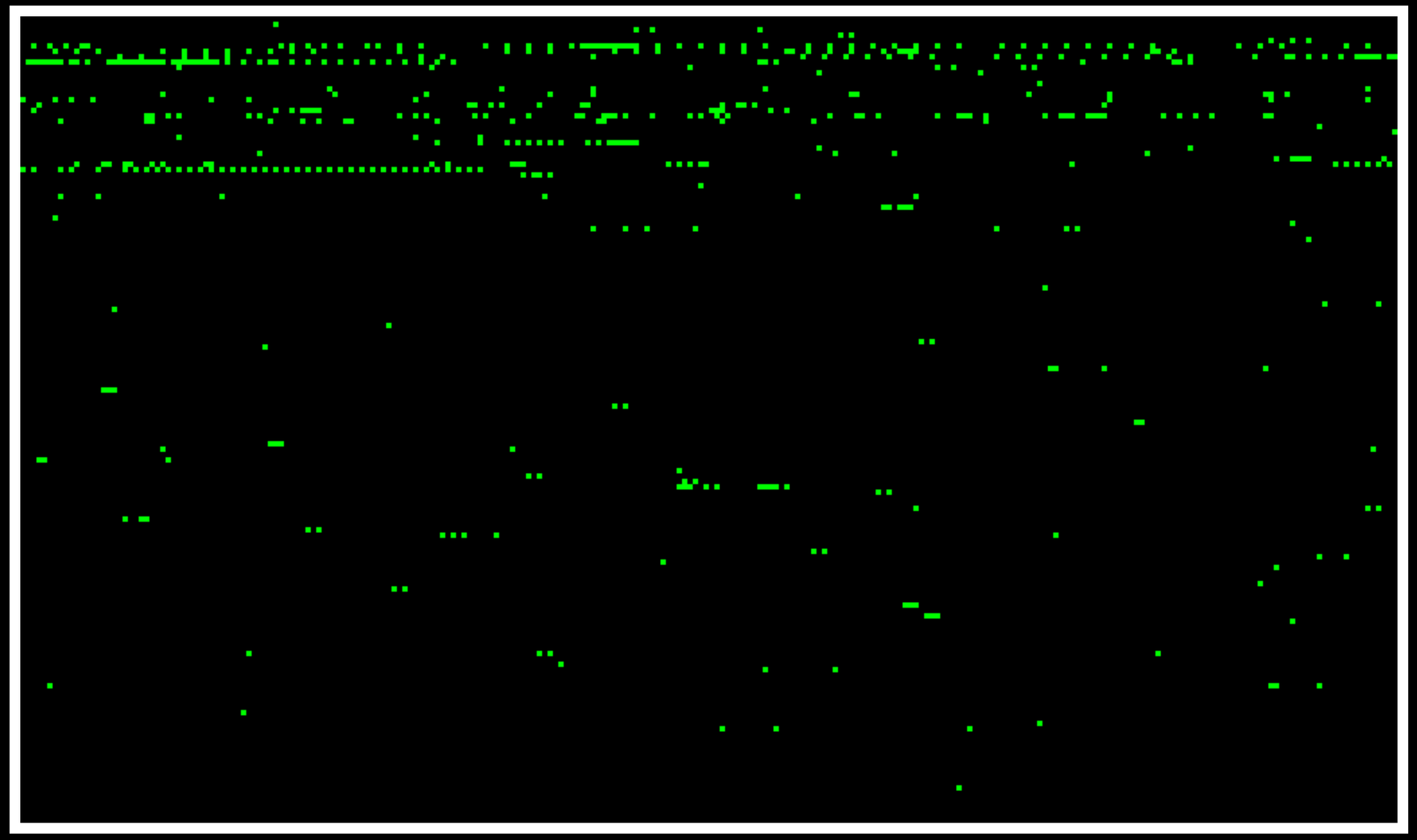
> covert channel


PoC 'O LoCoW
192.168.42.236:8000/covert_in.html
Secret message from Google Chrome
Leak
Touch
INFO user agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.94 Safari/537.36
INFO Created pages
PoC 'O LoCoW
192.168.42.236:8000/covert_out.html
INFO Detected: [r 1] [ix 32] [b 0x4]
INFO Detected: [r 2] [ix 32] [b 0x4]
INFO Detected: [r 0] [ix 32] [b 0x20]
INFO Detected: [r 1] [ix 32] [b 0x20]
INFO Detected: [r 2] [ix 32] [b 0x20]
INFO Detected: [r 0] [ix 32] [b 0x40]
INFO Detected: [r 1] [ix 32] [b 0x40]
INFO Detected: [r 2] [ix 32] [b 0x40]
WARNING Found: Secret message from Google Chrome
Search the web and Windows
Windows 10 Education

Having fun with deduplication

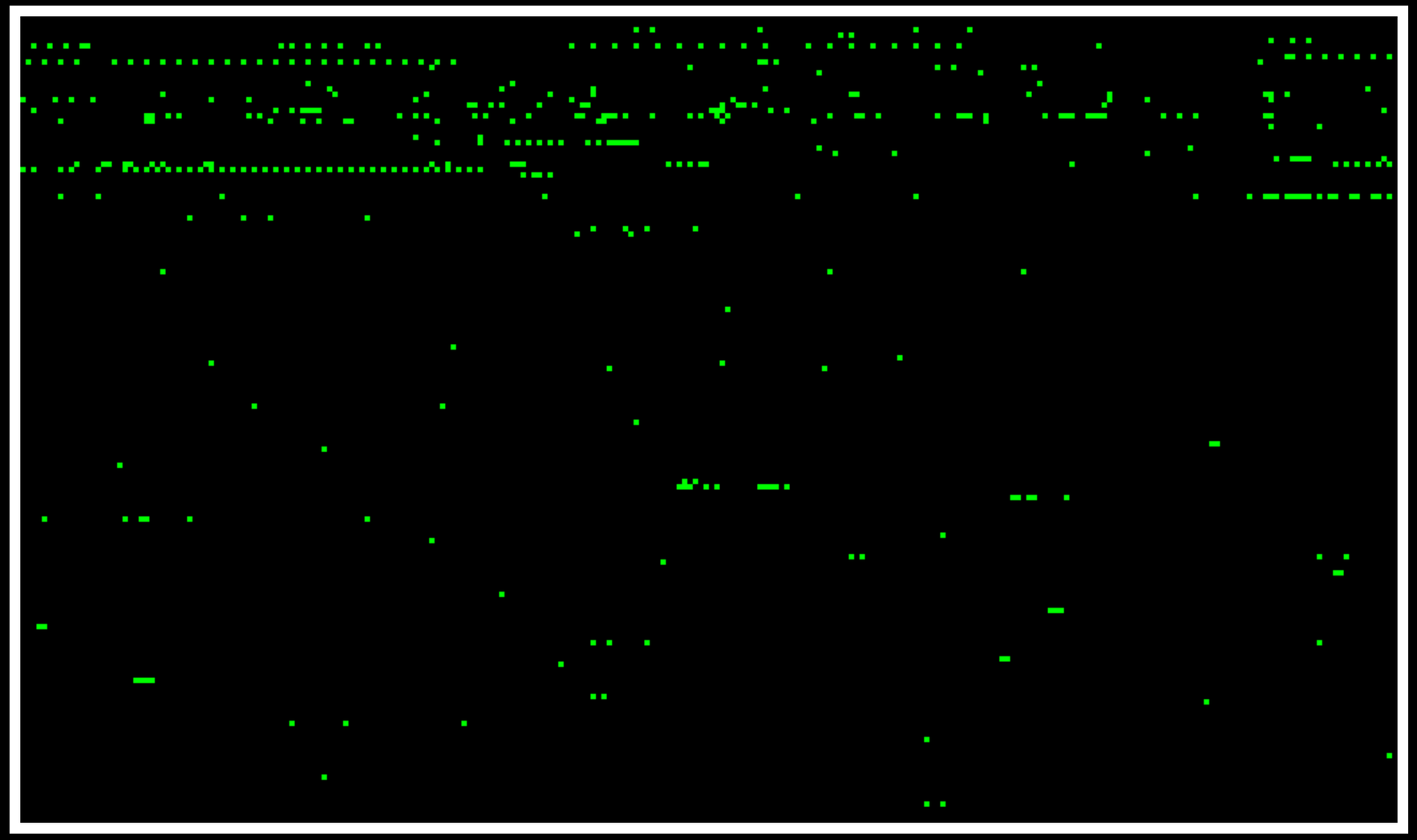
> covert channel
> detect running software

Wordpad memory dump



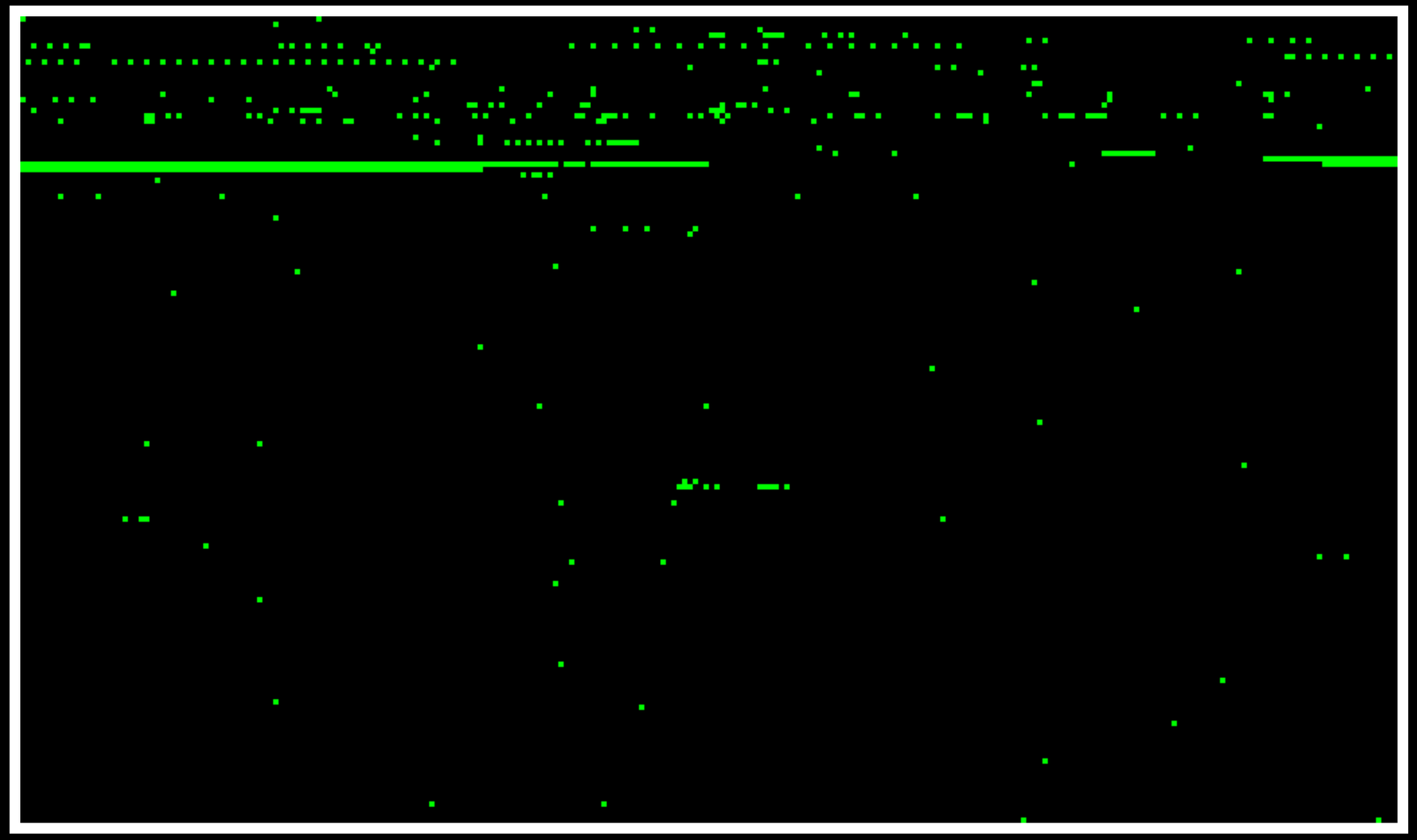
wordpad not running

Wordpad memory dump



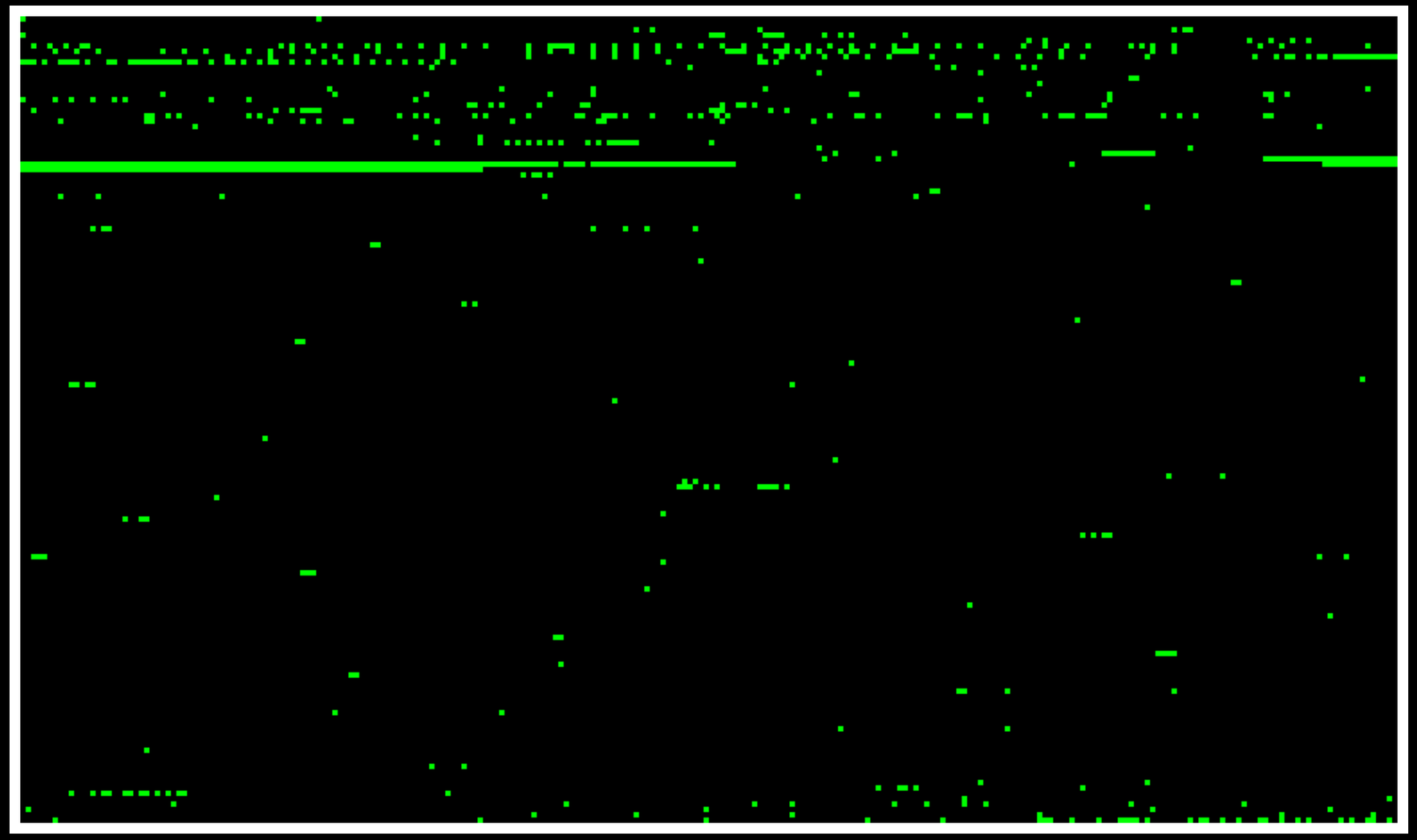
wordpad not running

Wordpad memory dump



wordpad running

Wordpad memory dump



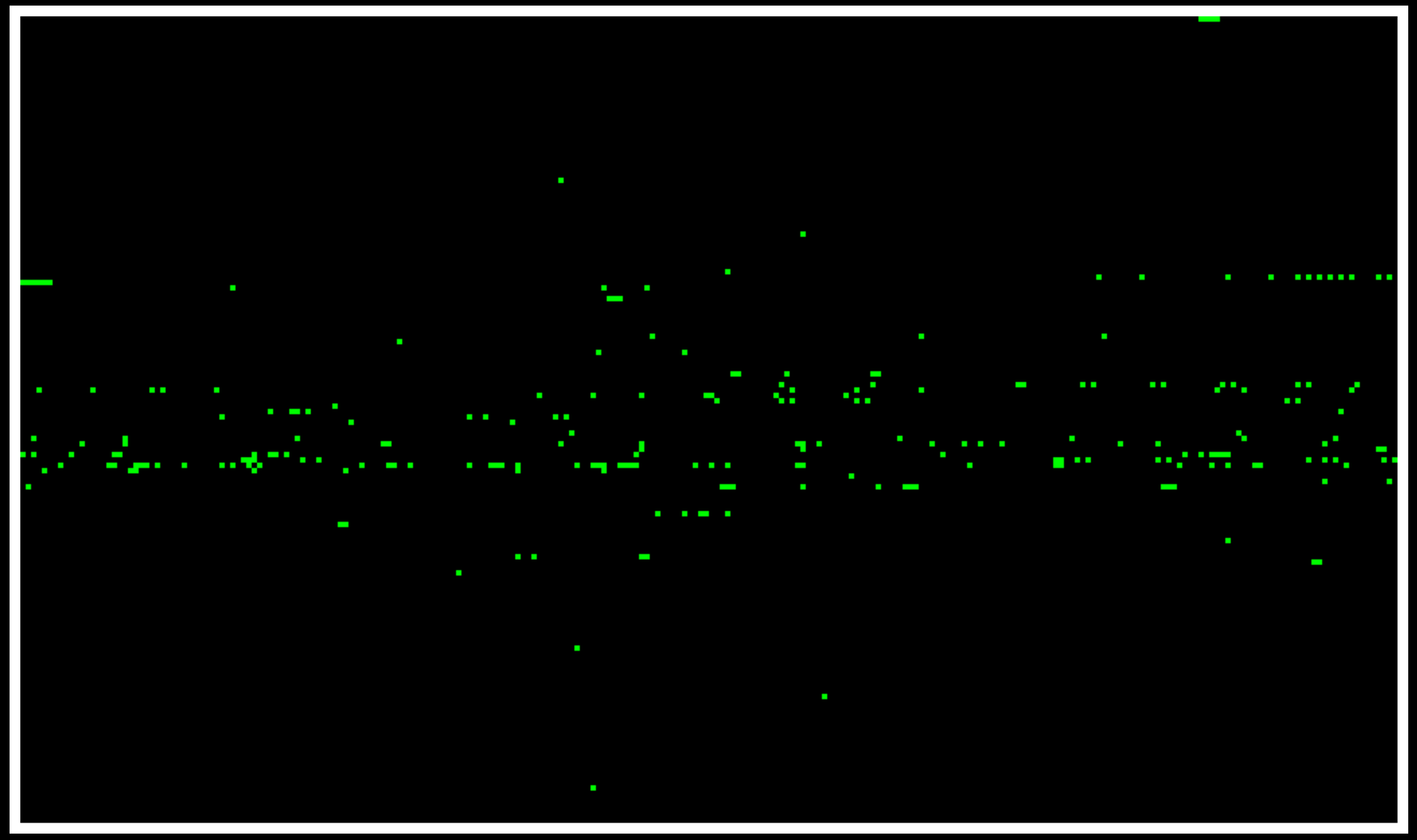
wordpad running

Signal not as clear as expected,

Reason:

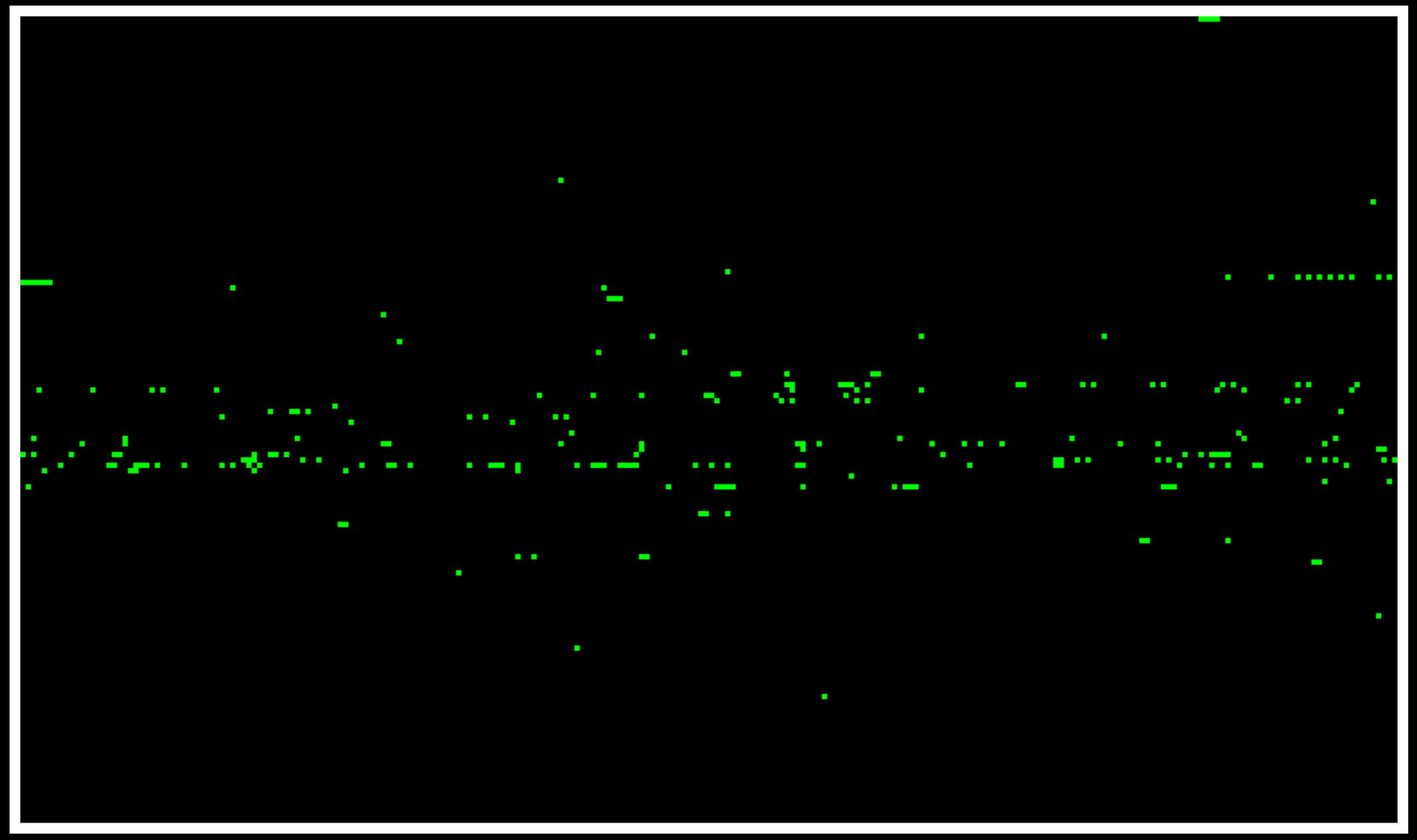
- file backed memory
- not deduplicated the same way
- on Windows.

Skype memory dump



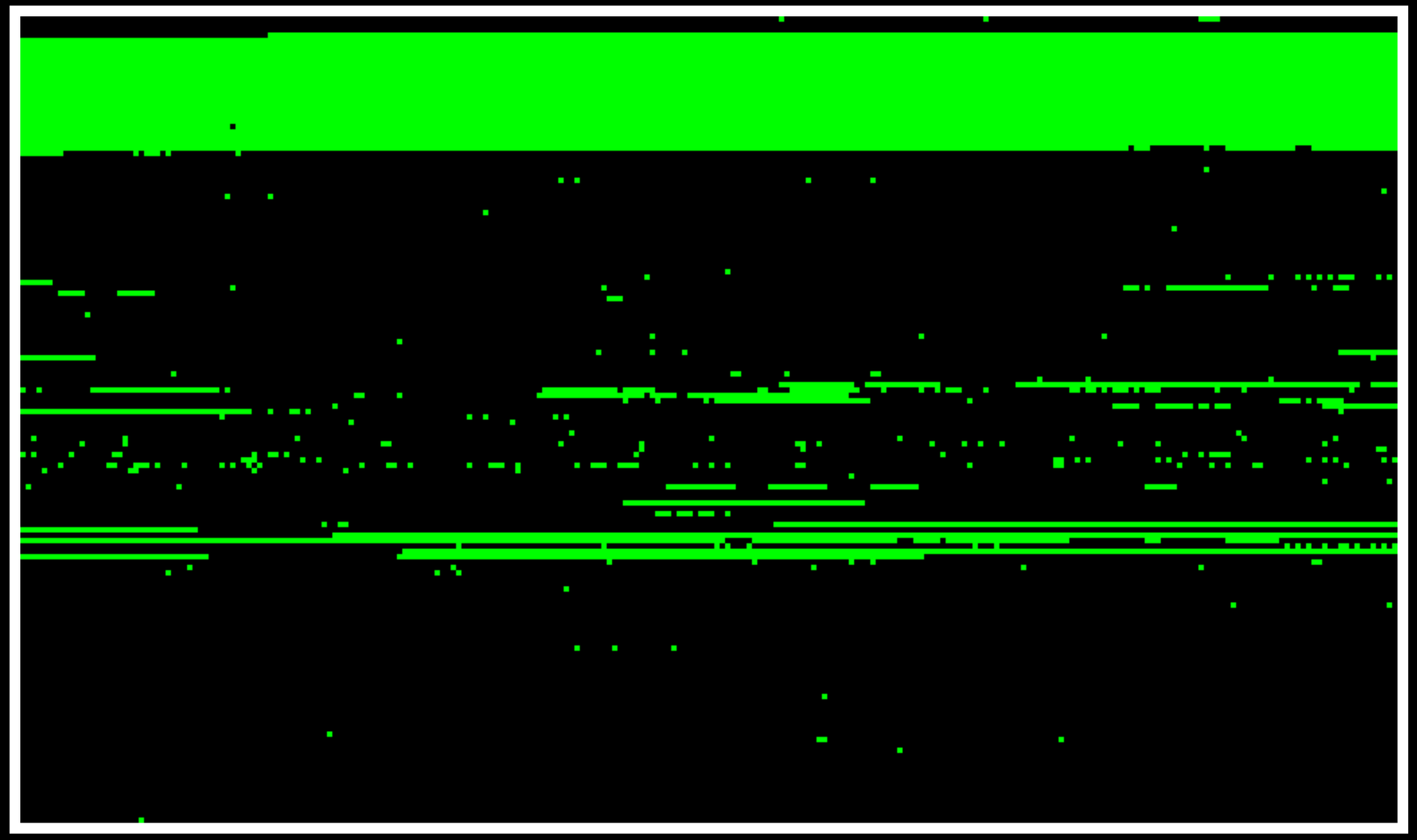
skype not running

Skype memory dump



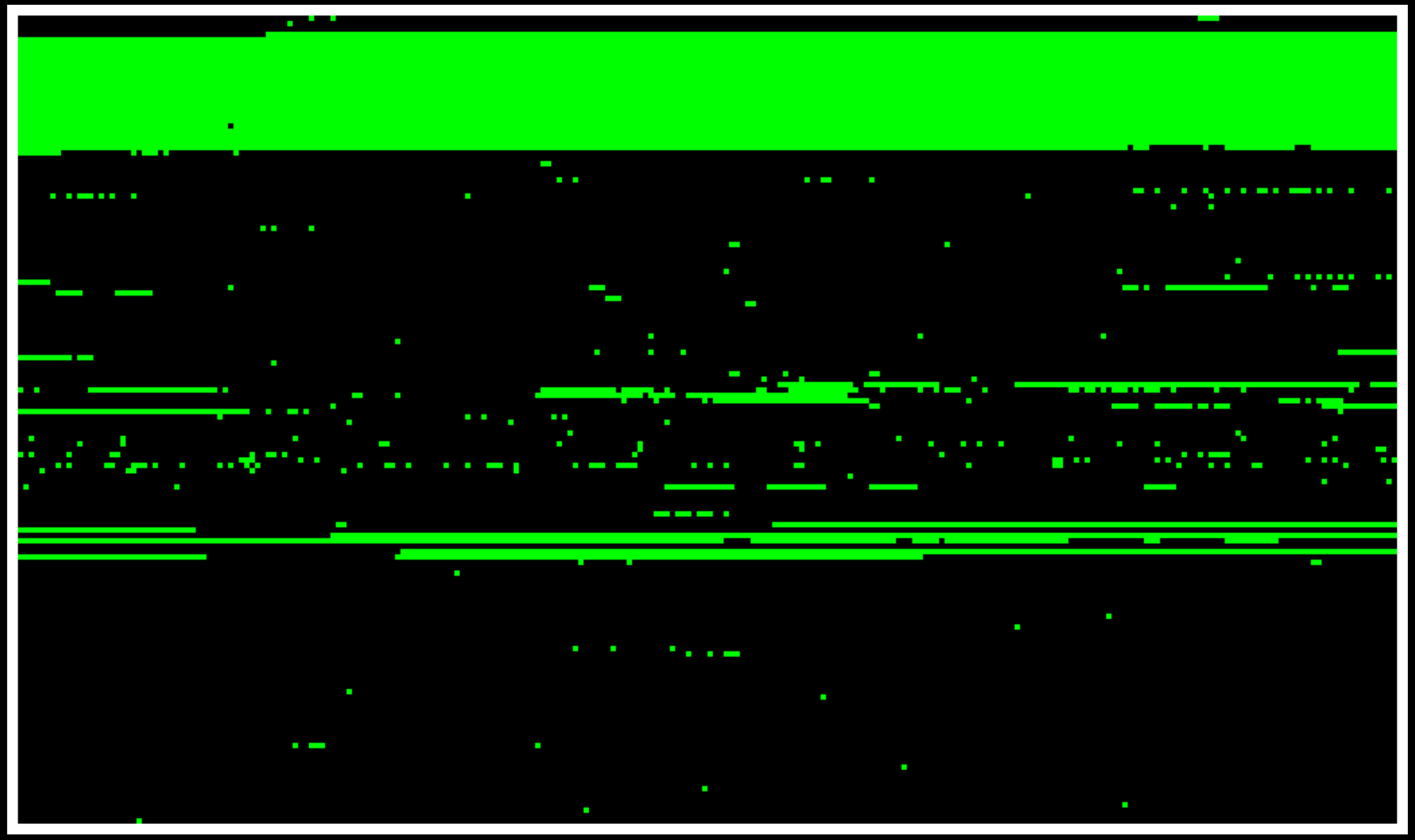
skype not running

Skype memory dump



skype running

Skype memory dump



skype running

For our Edge exploit,
a single-bit, page-granularity
info leak isn't enough

Can we generalize this to leaking arbitrary data, like an ASLR pointer or a password?

Leaking existing pages is slow and the gained information is limited.

What if we can manipulate the contents of the victim's memory to leak secrets hand-picked by the attacker.

Challenge 1:

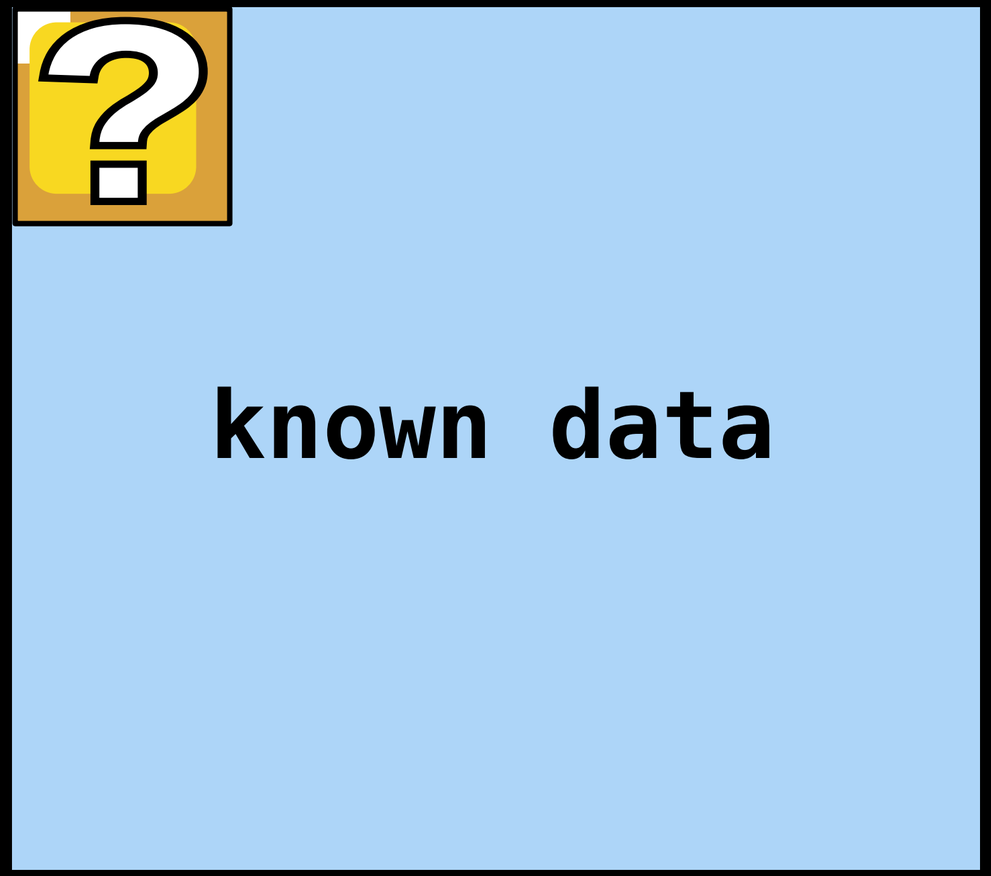
The secret we want to leak does not span an entire page.

turning a secret into a page

secret

turning a secret into a page

secret

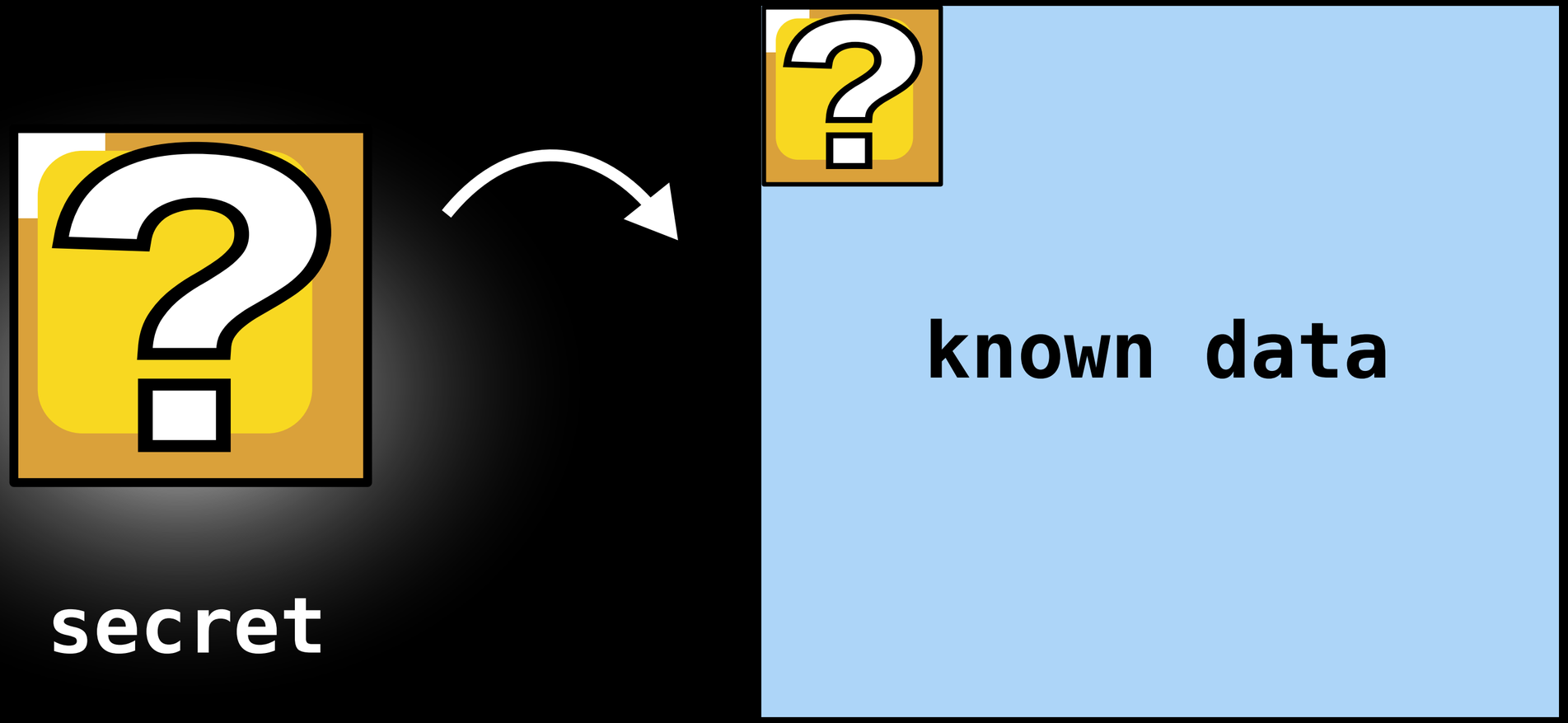


secret page

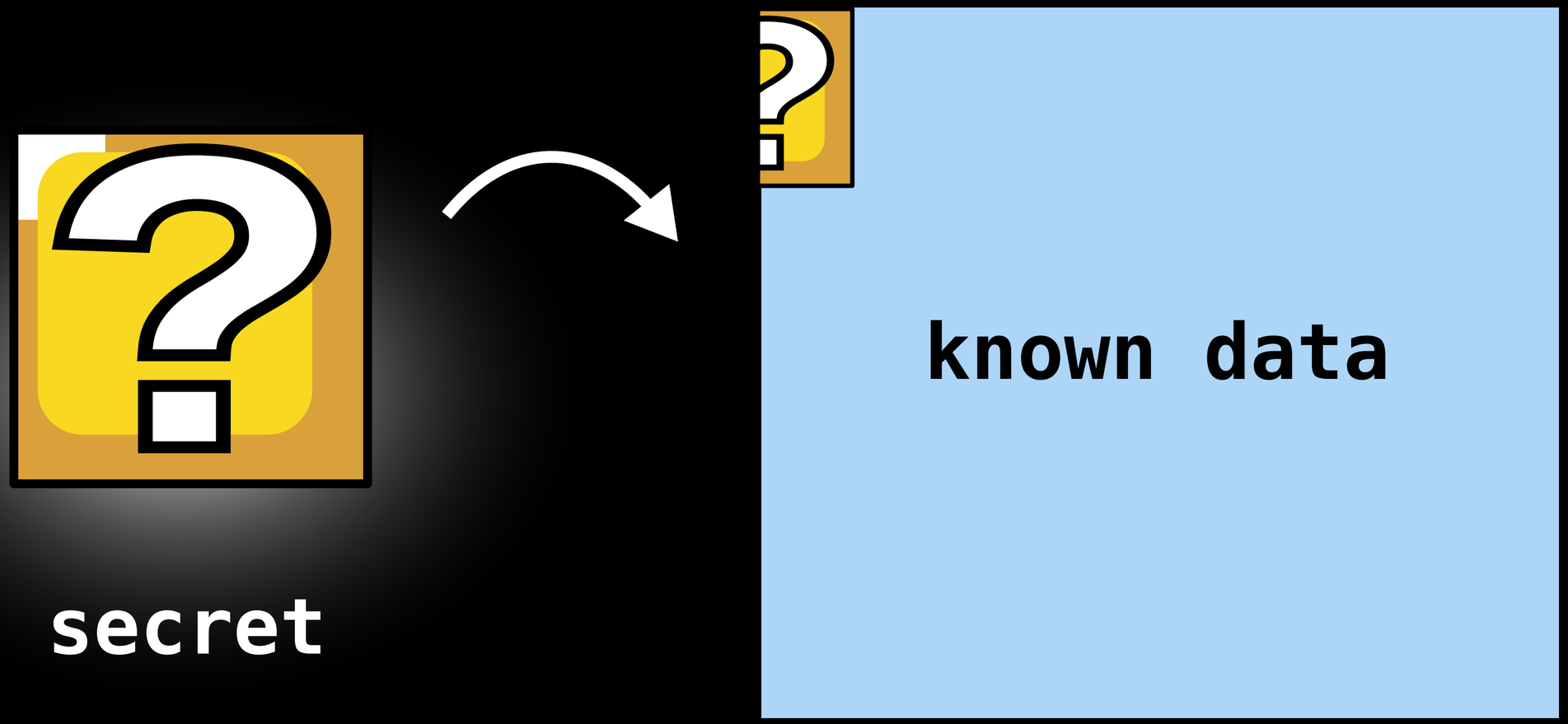
Challenge 2:

The secret we want to leak has too much entropy to leak all at once.

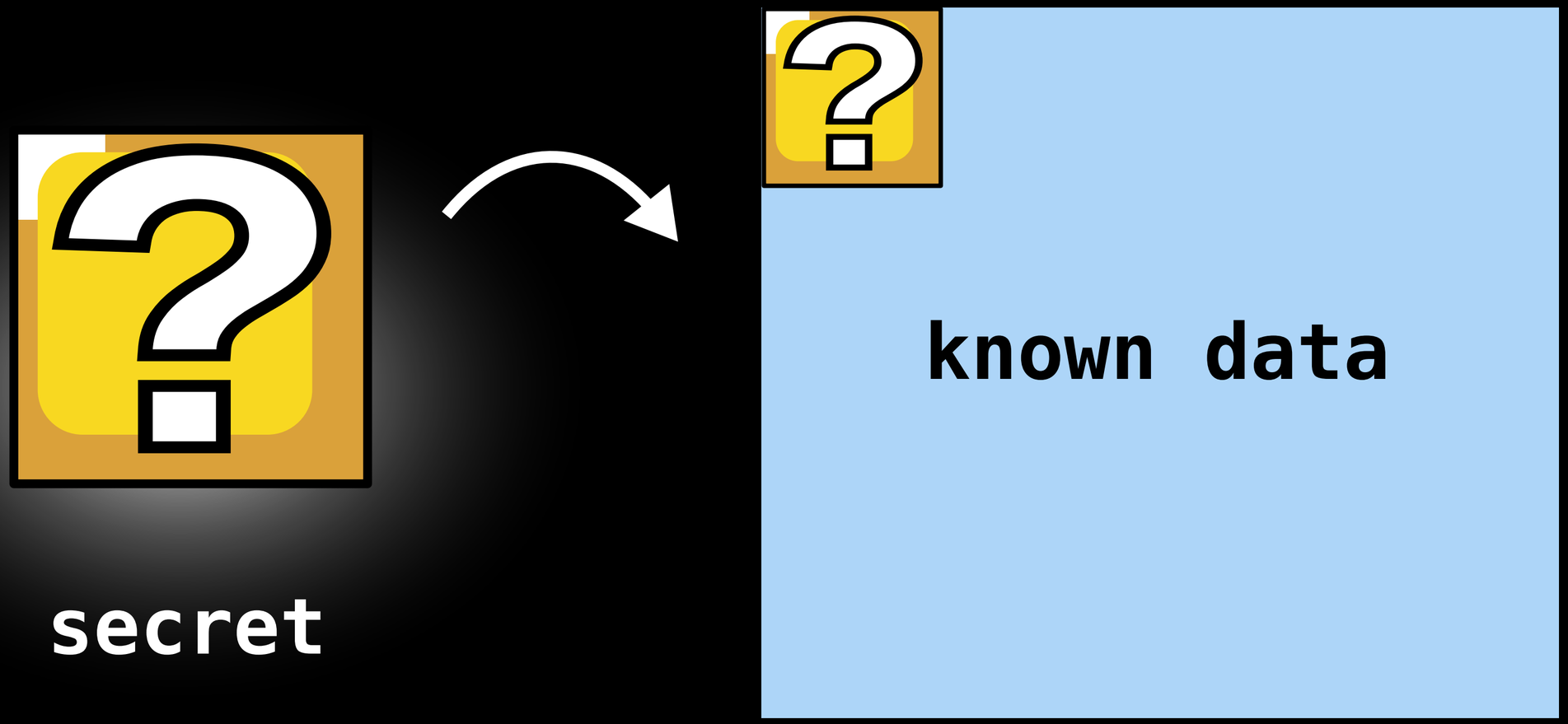
primitive #1: alignment probing



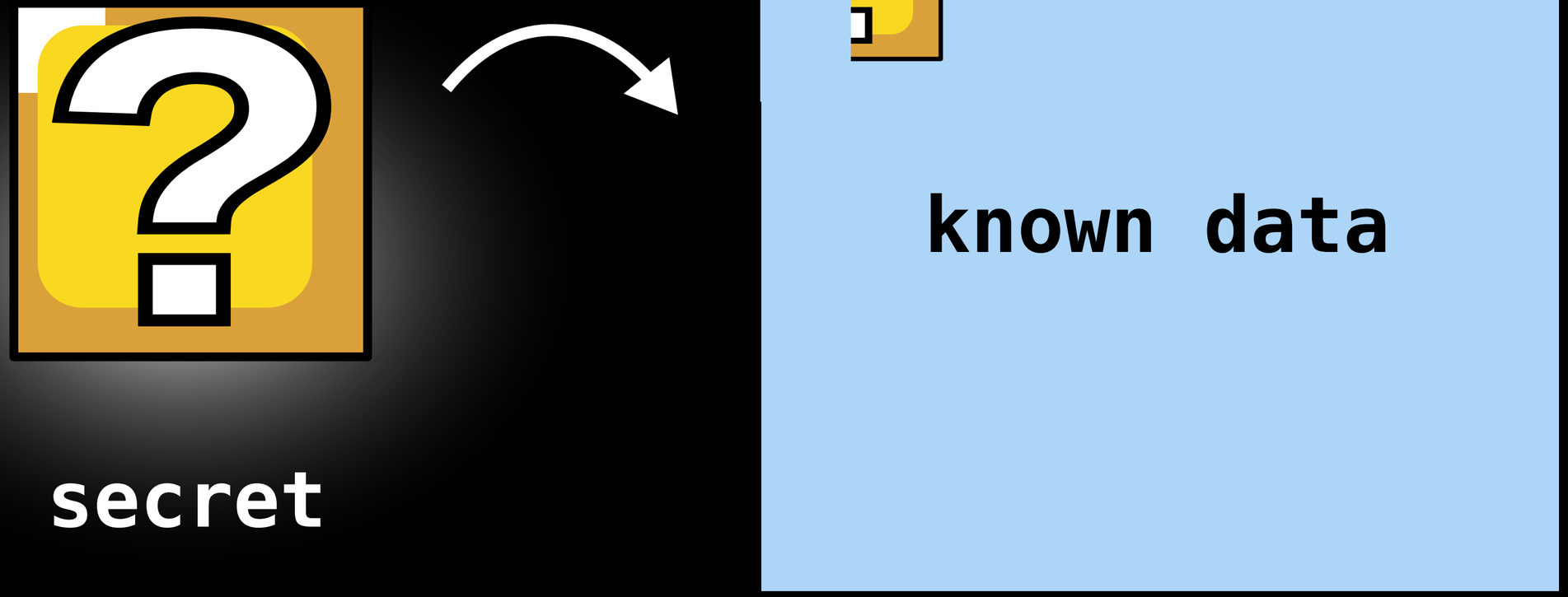
primitive #1: alignment probing



primitive #2: partial reuse



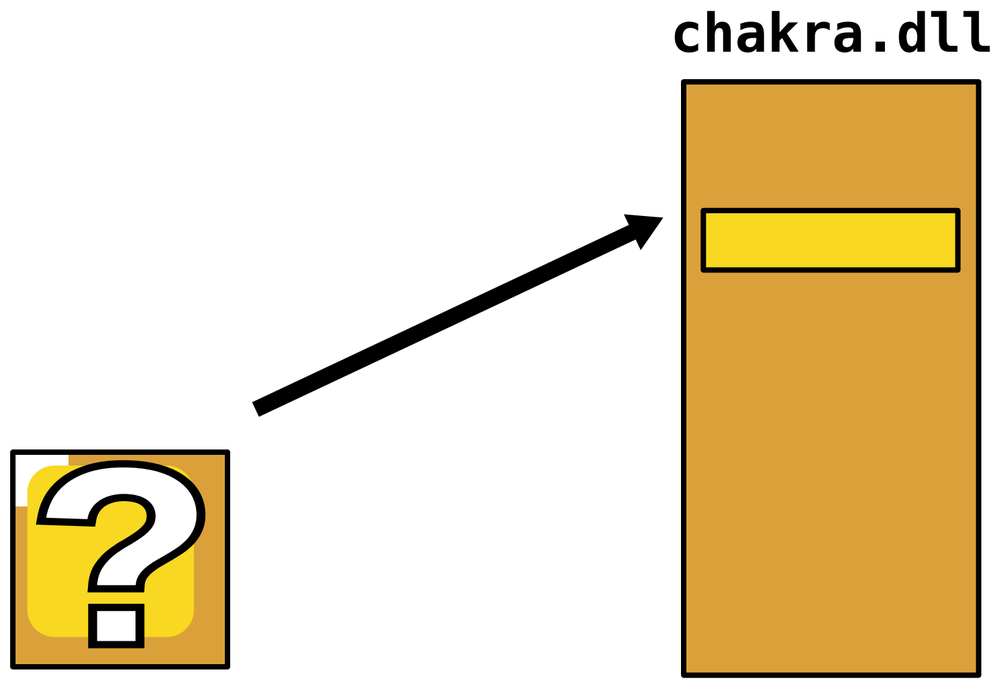
primitive #2: partial reuse



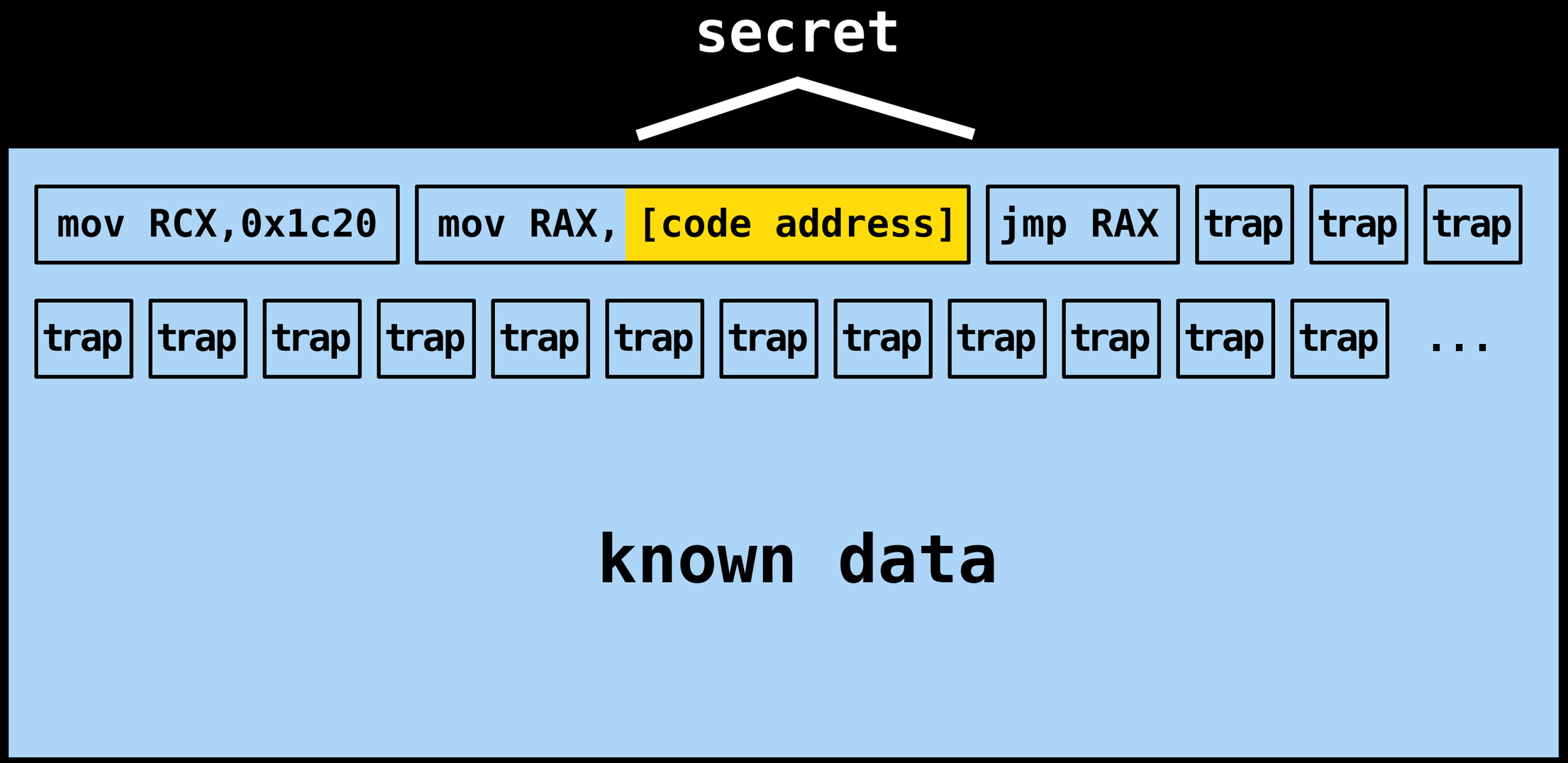
Outline:

Deduplication

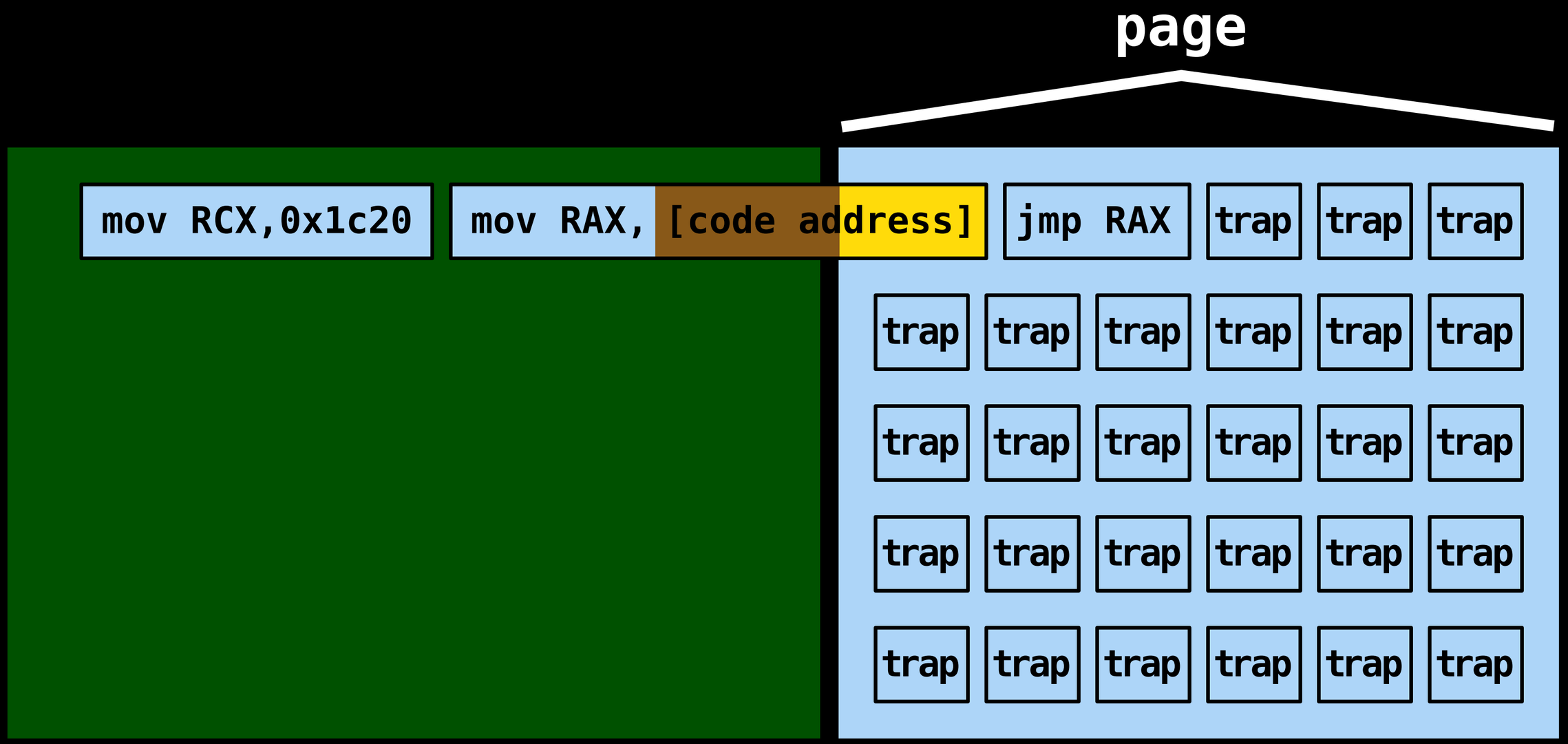
- leak heap & code addresses



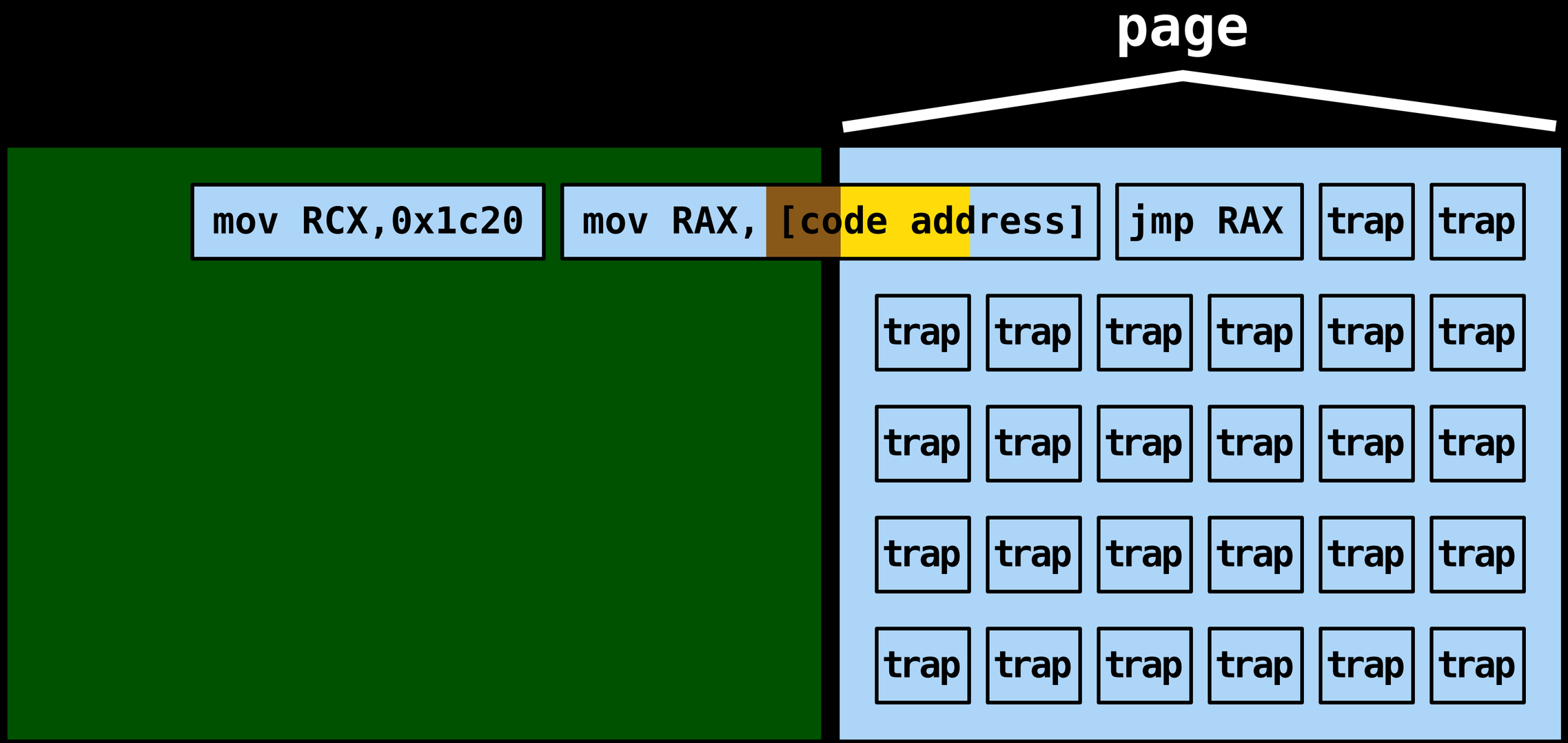
JIT function epilogue (MS Edge)



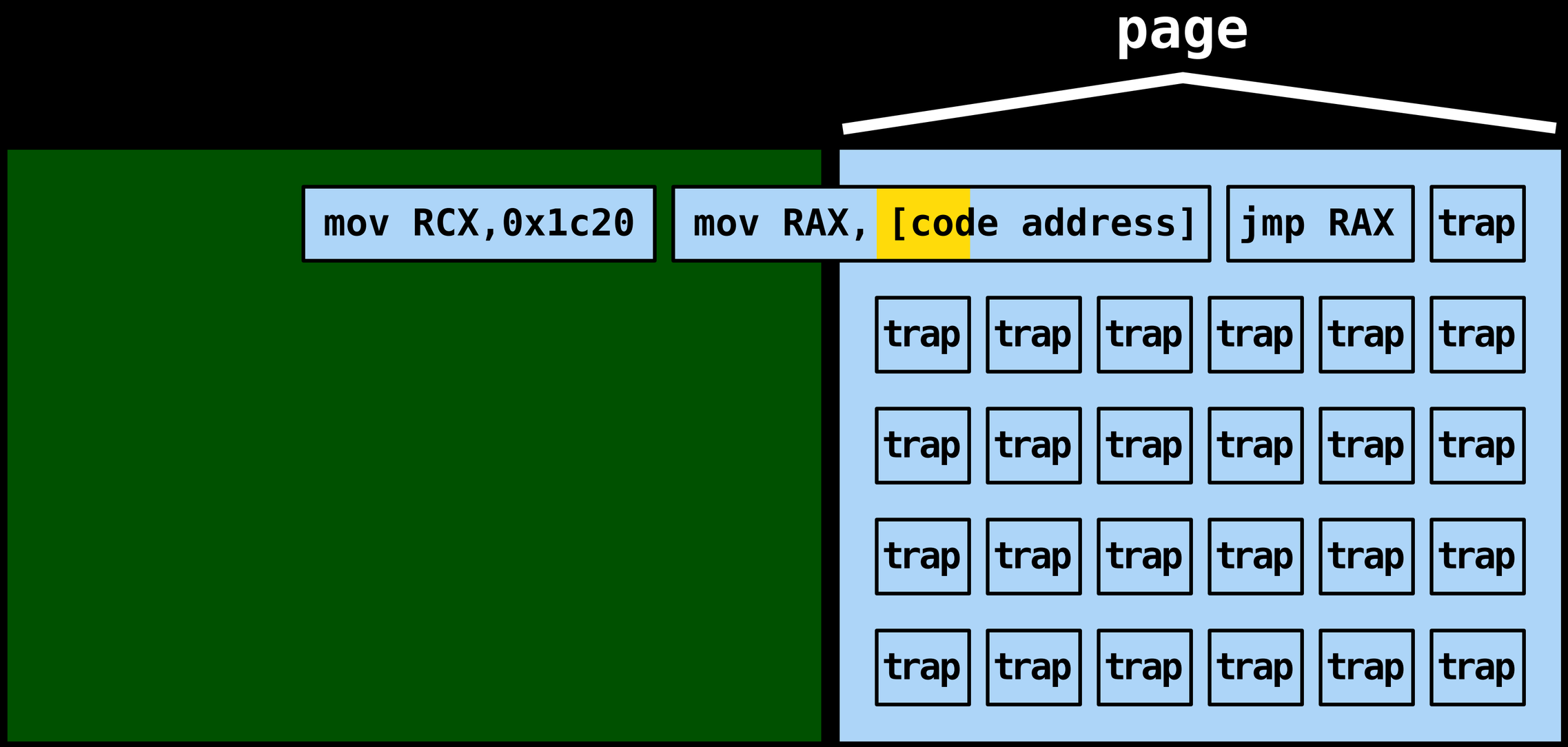
JIT function epilogue (MS Edge)



JIT function epilogue (MS Edge)



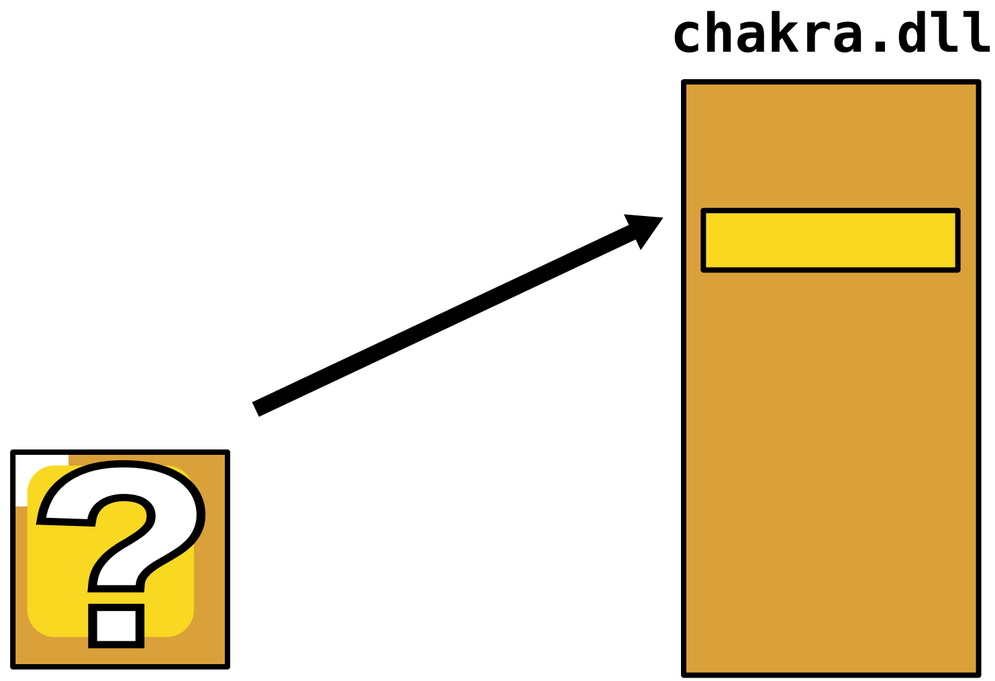
JIT function epilogue (MS Edge)



Outline:

Deduplication

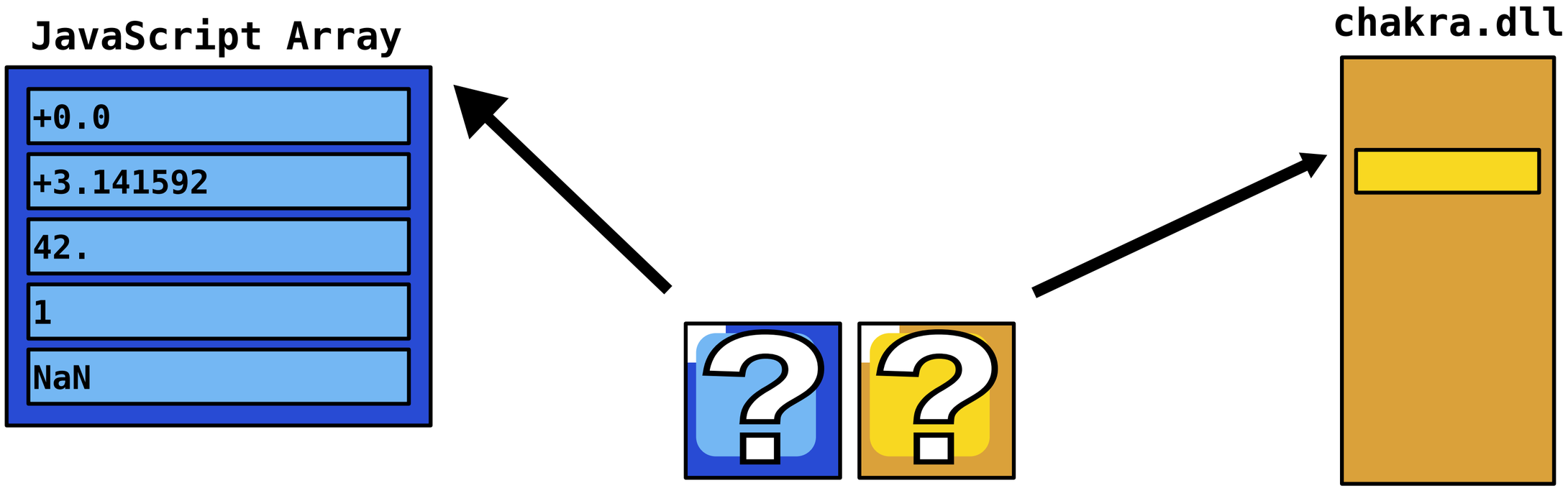
- leak heap & code addresses



Outline:

Deduplication

- leak heap & code addresses



What if leaking a heap pointer in stages is not possible...

We need to guess a page containing the complete pointer.

Heap pointer entropy in Edge

0x5F48143540

Heap pointer entropy in Edge

advertised ASLR (24 bit)

0x5F48143540

Heap pointer entropy in Edge

advertised ASLR (24 bit)

0x5F48143540

non-deterministic bits
(+/- 36 bit)

Heap pointer entropy in Edge

64G

advertised ASLR (24 bit)

0x5F48143540

non-deterministic bits
(+/- 36 bit)

Heap pointer entropy in Edge

64G

advertised ASLR (24 bit)

0x5F48143540

256T

non-deterministic bits
(+/- 36 bit)

Heap pointer entropy in Edge

advertised ASLR (24 bit)

64G * redundancy

0x5F48143540

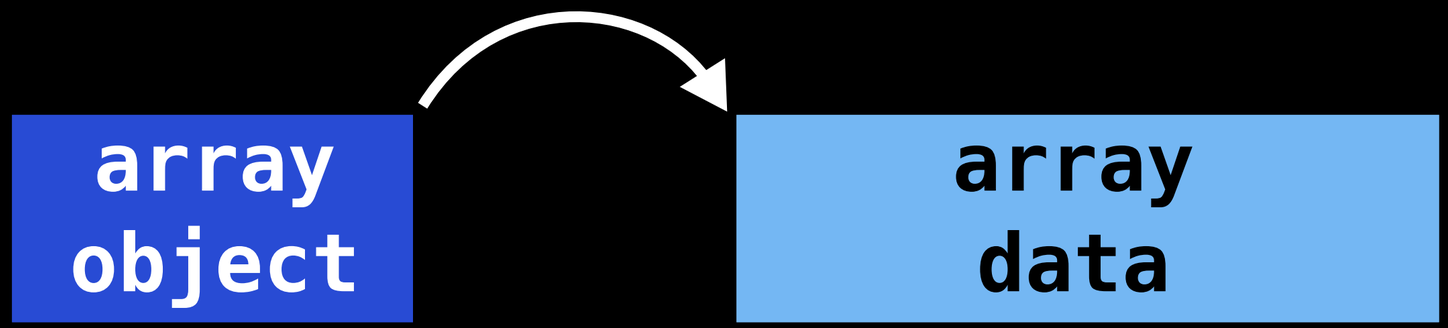
non-deterministic bits (+/- 36 bit)

256T * redundancy

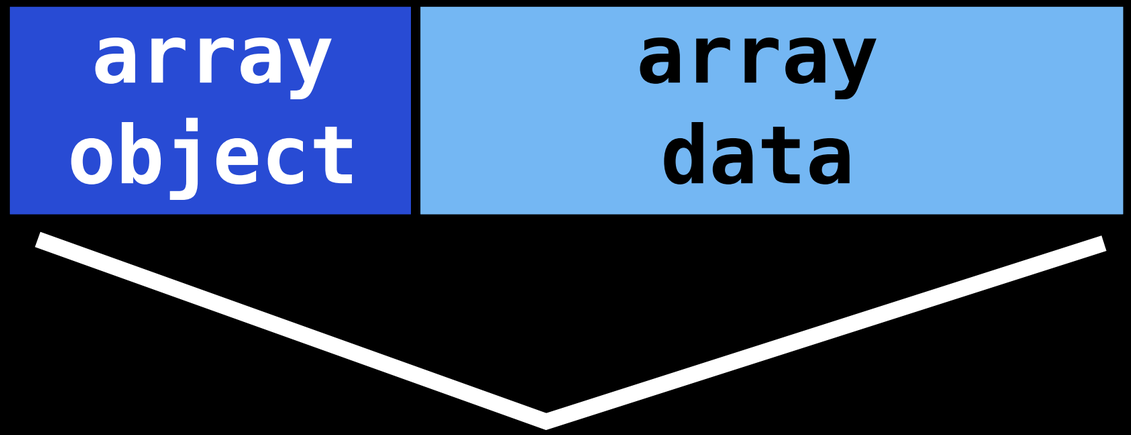
Slab allocator for JavaScript objects

array
object

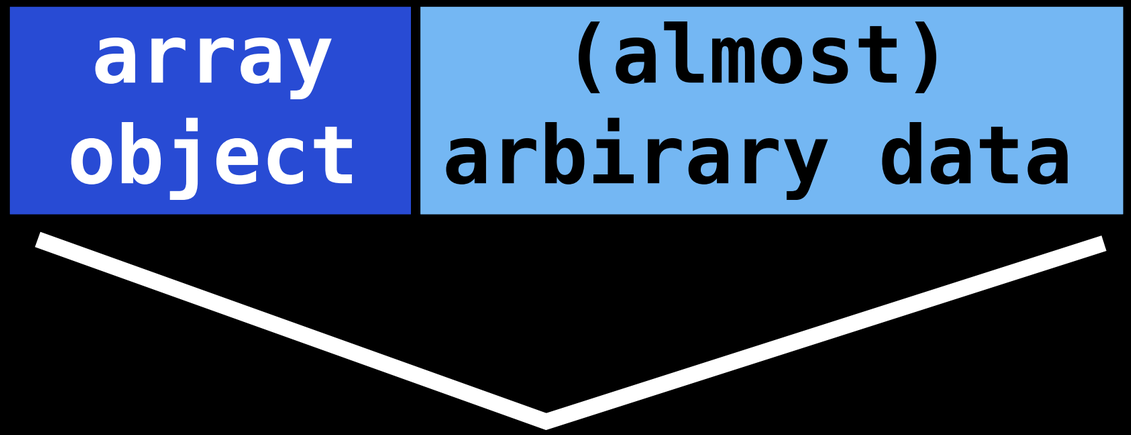
Slab allocator for JavaScript objects



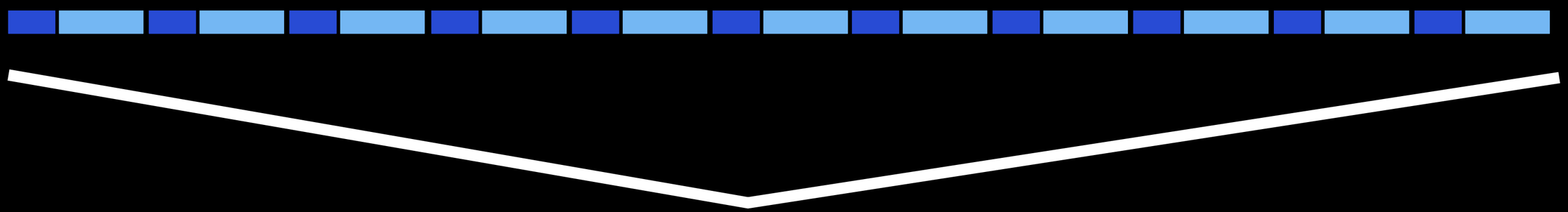
Slab allocator for JavaScript objects



Slab allocator for JavaScript objects

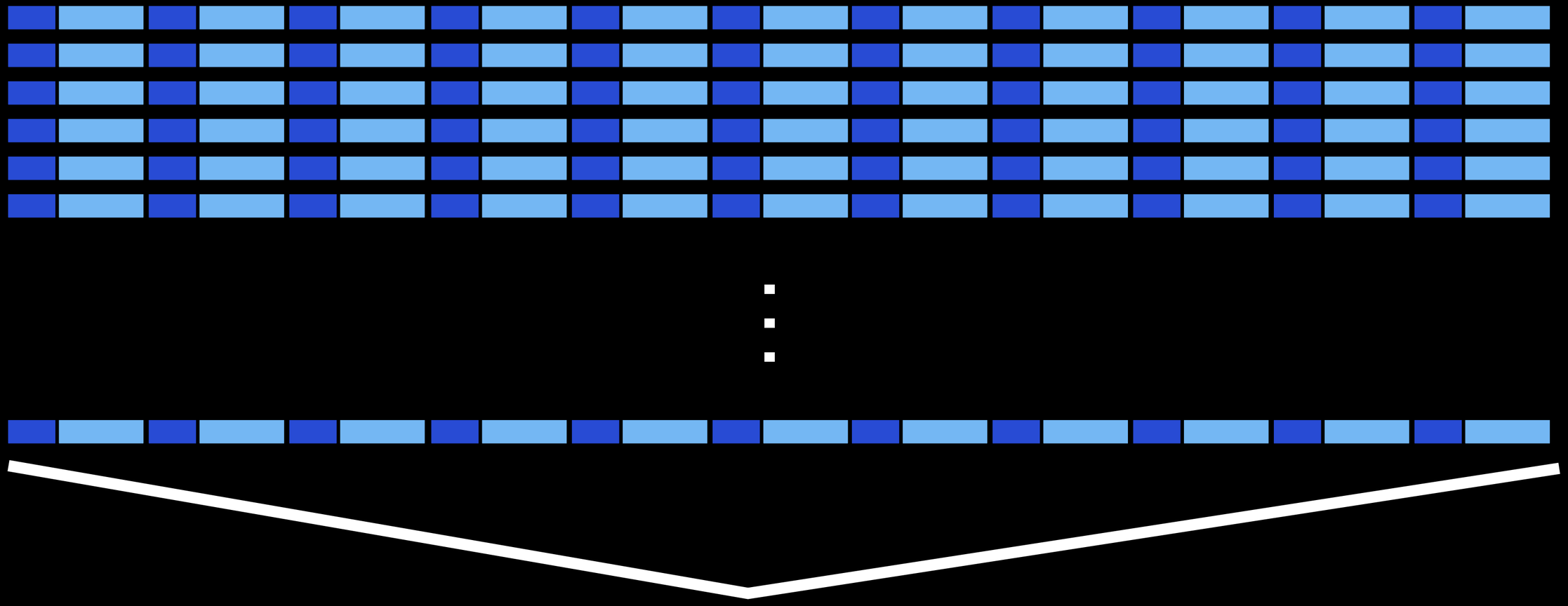


Slab allocator for JavaScript objects



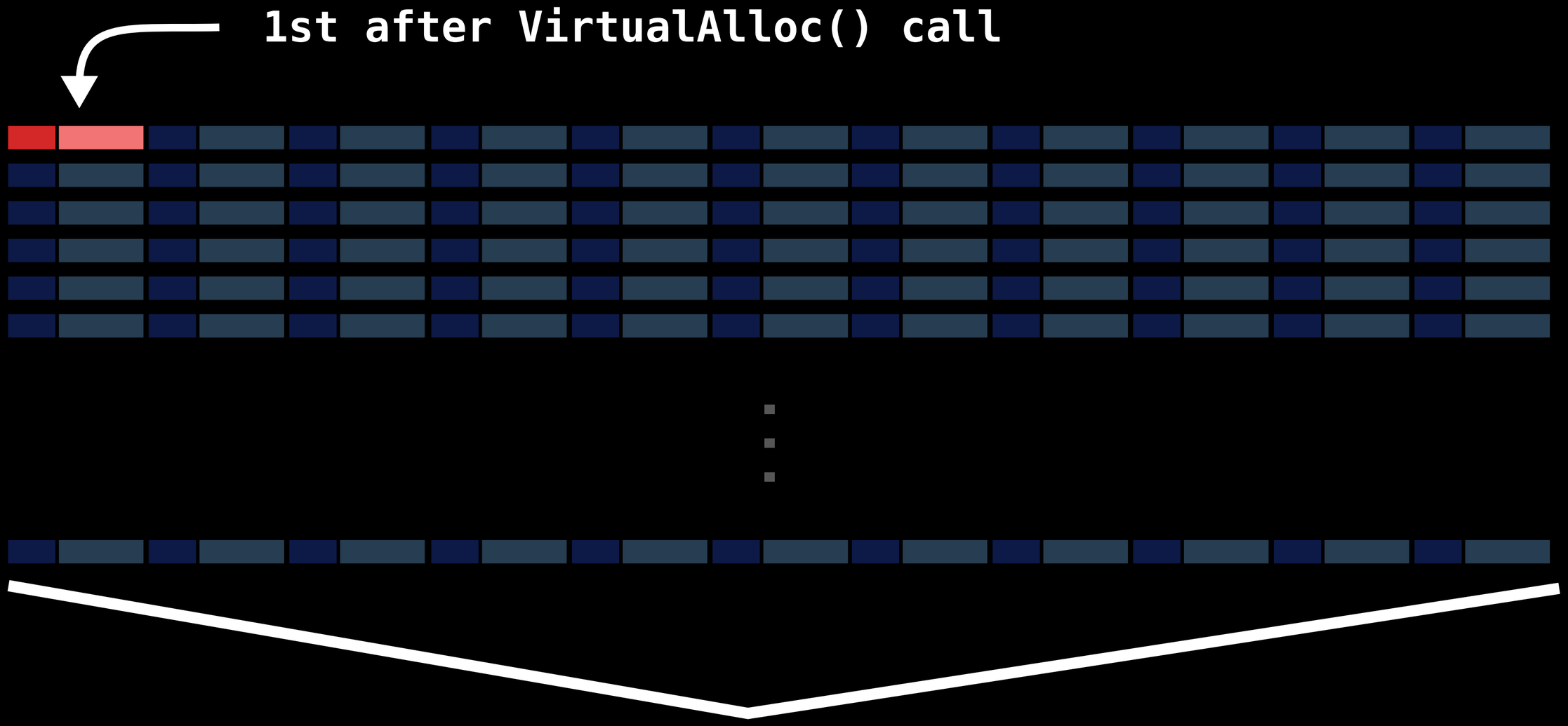
16K slab

Slab allocator for JavaScript objects

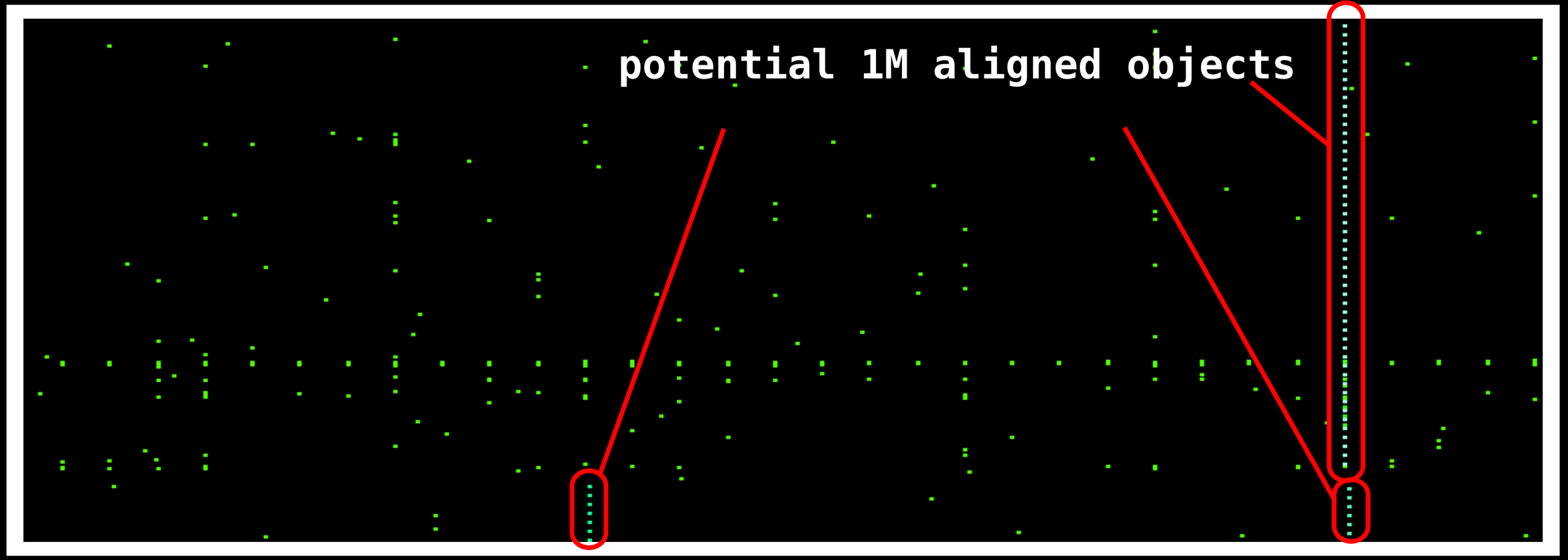


1M VirtualAlloc()

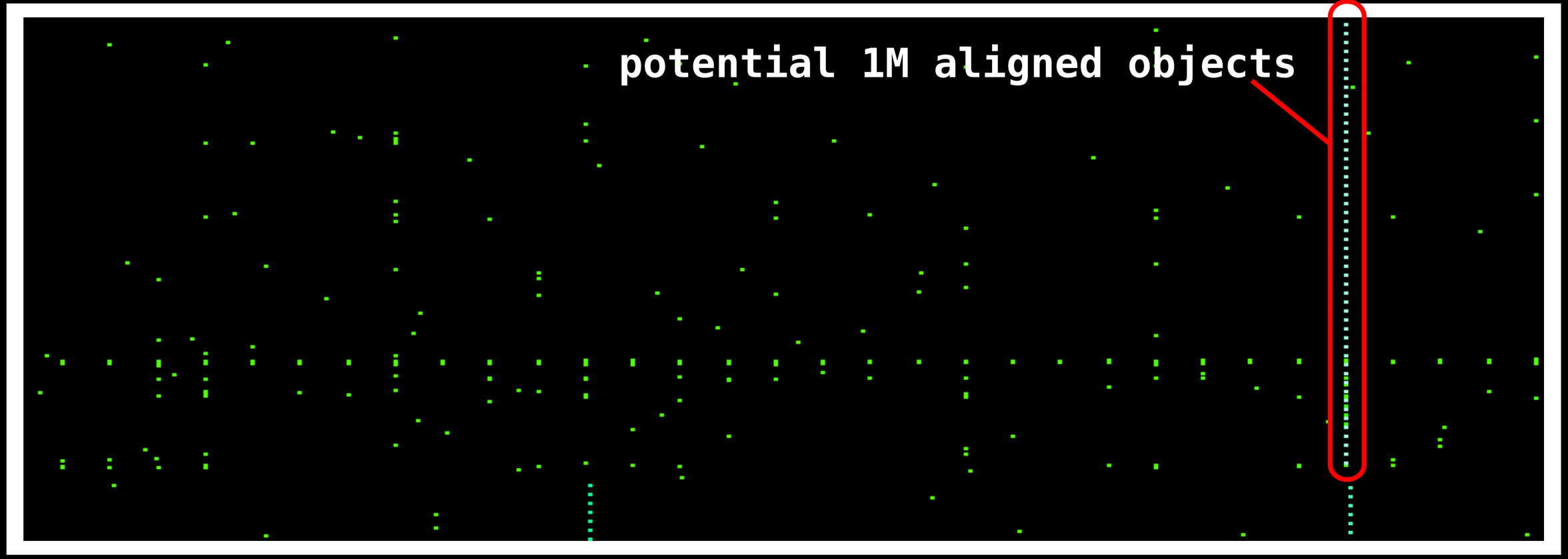
Slab allocator for JavaScript objects



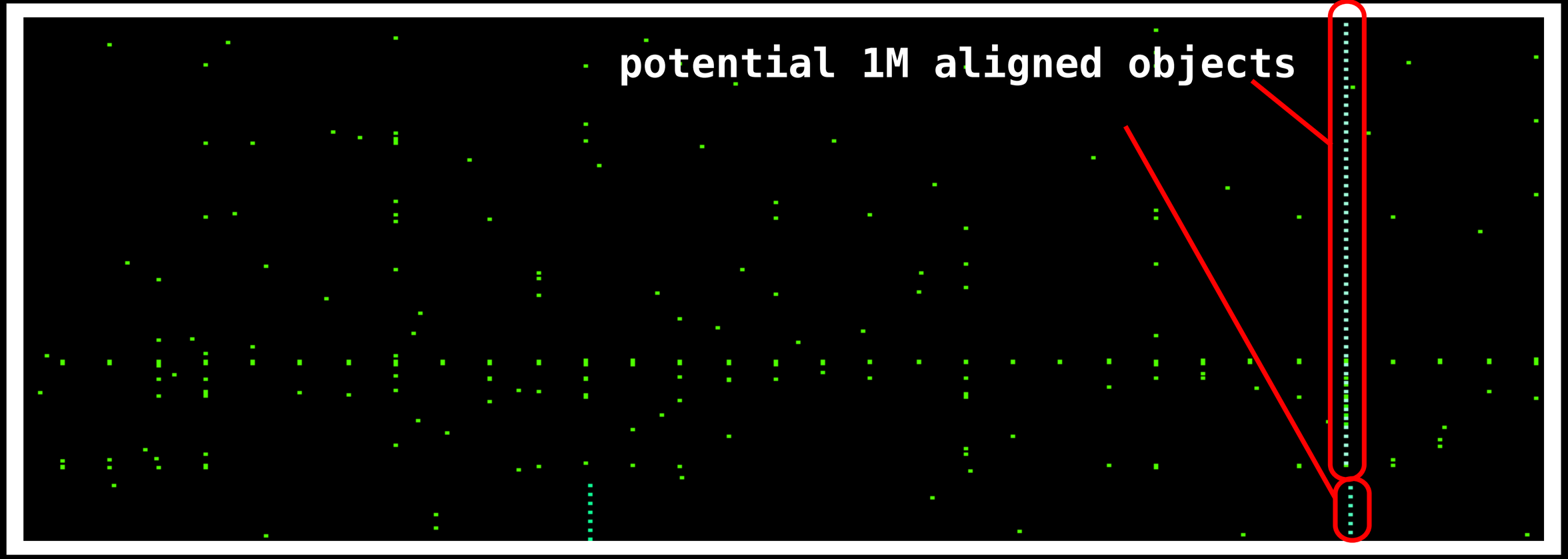
Slab allocator for JavaScript objects



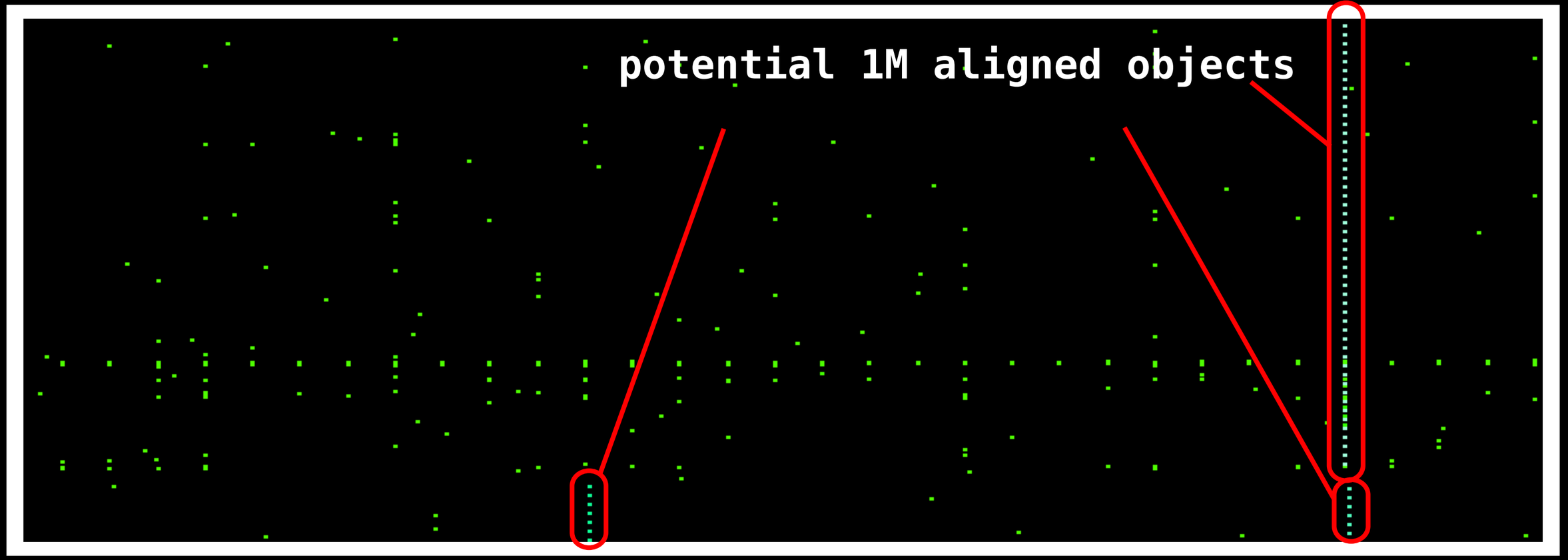
Slab allocator for JavaScript objects



Slab allocator for JavaScript objects



Slab allocator for JavaScript objects



Heap pointer entropy in Edge

advertised ASLR (24 bit)

64G * redundancy

0x5F48143540

256T * redundancy

non-deterministic bits
(+/- 36 bit)

Heap pointer entropy in Edge

64G

advertised ASLR (24 bit) * redundancy

0x5F48100000

entropy after alignment
(20 bit)

Heap pointer entropy in Edge

64G * redundancy

advertised ASLR (24 bit)

0x5F48100000

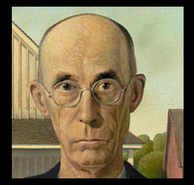
4G * redundancy

entropy after alignment (20 bit)

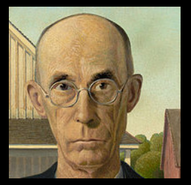
birthday problem

birthday problem

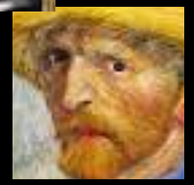
birthday problem

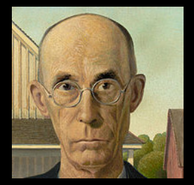


birthday problem



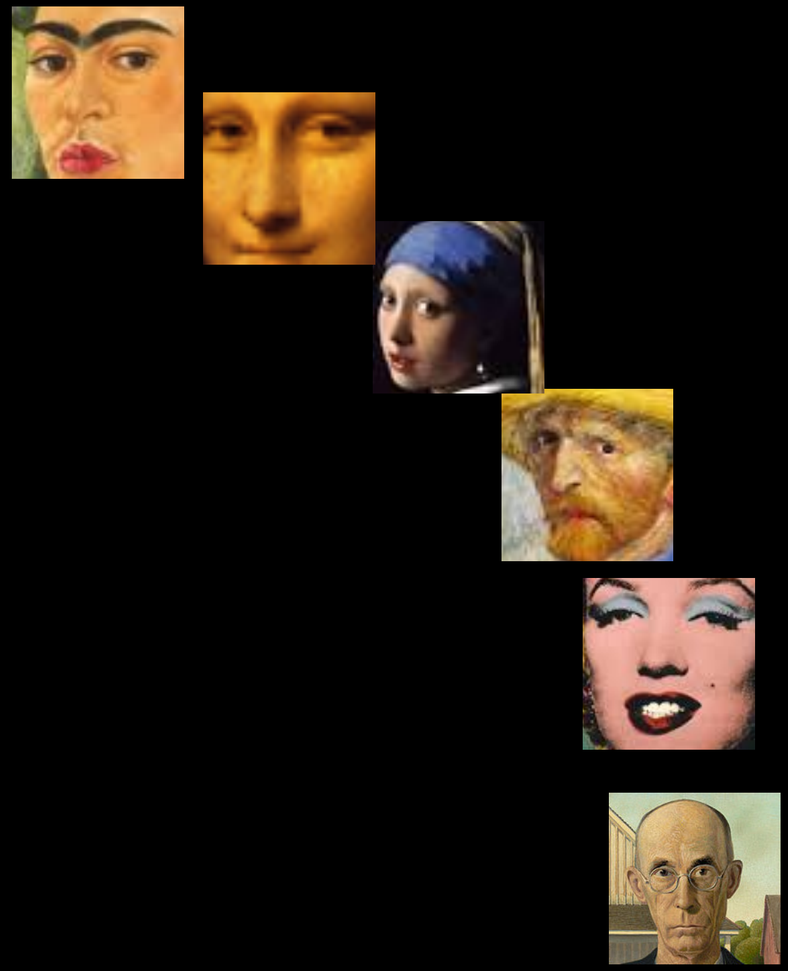
birthday problem



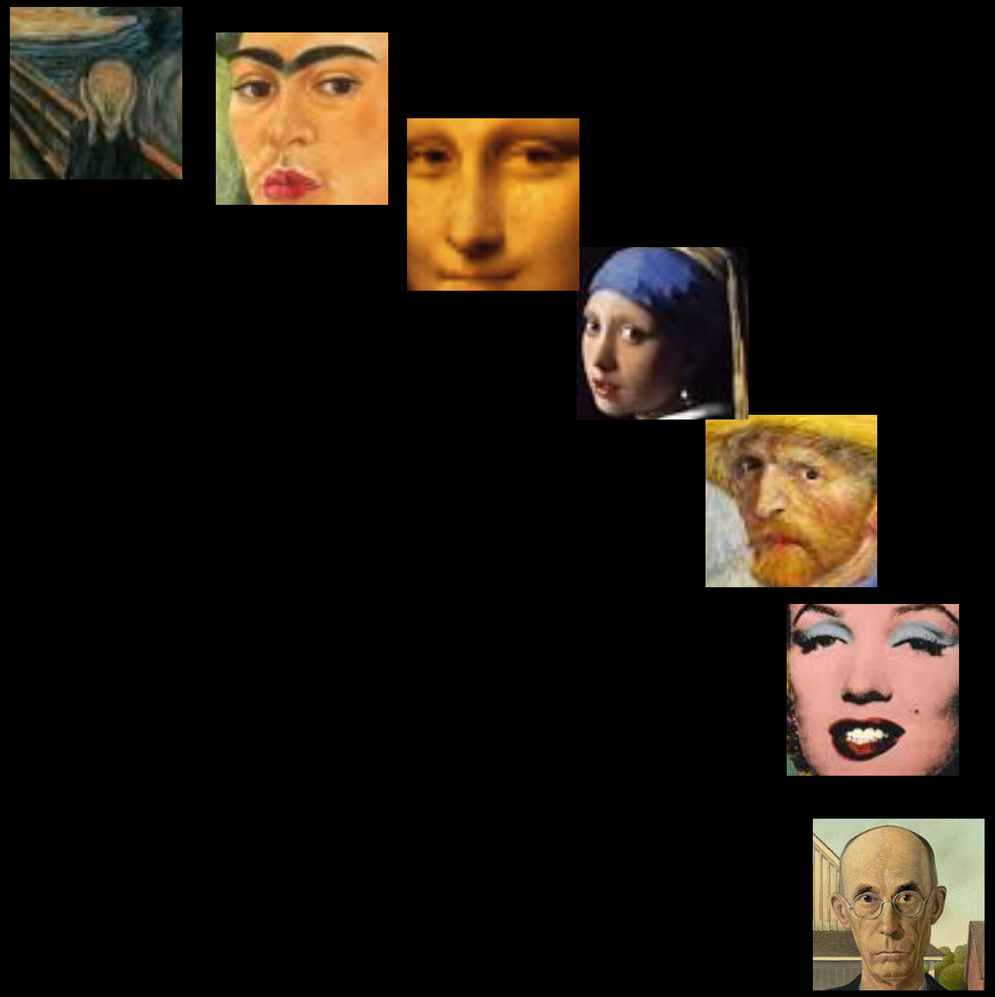


birthday problem

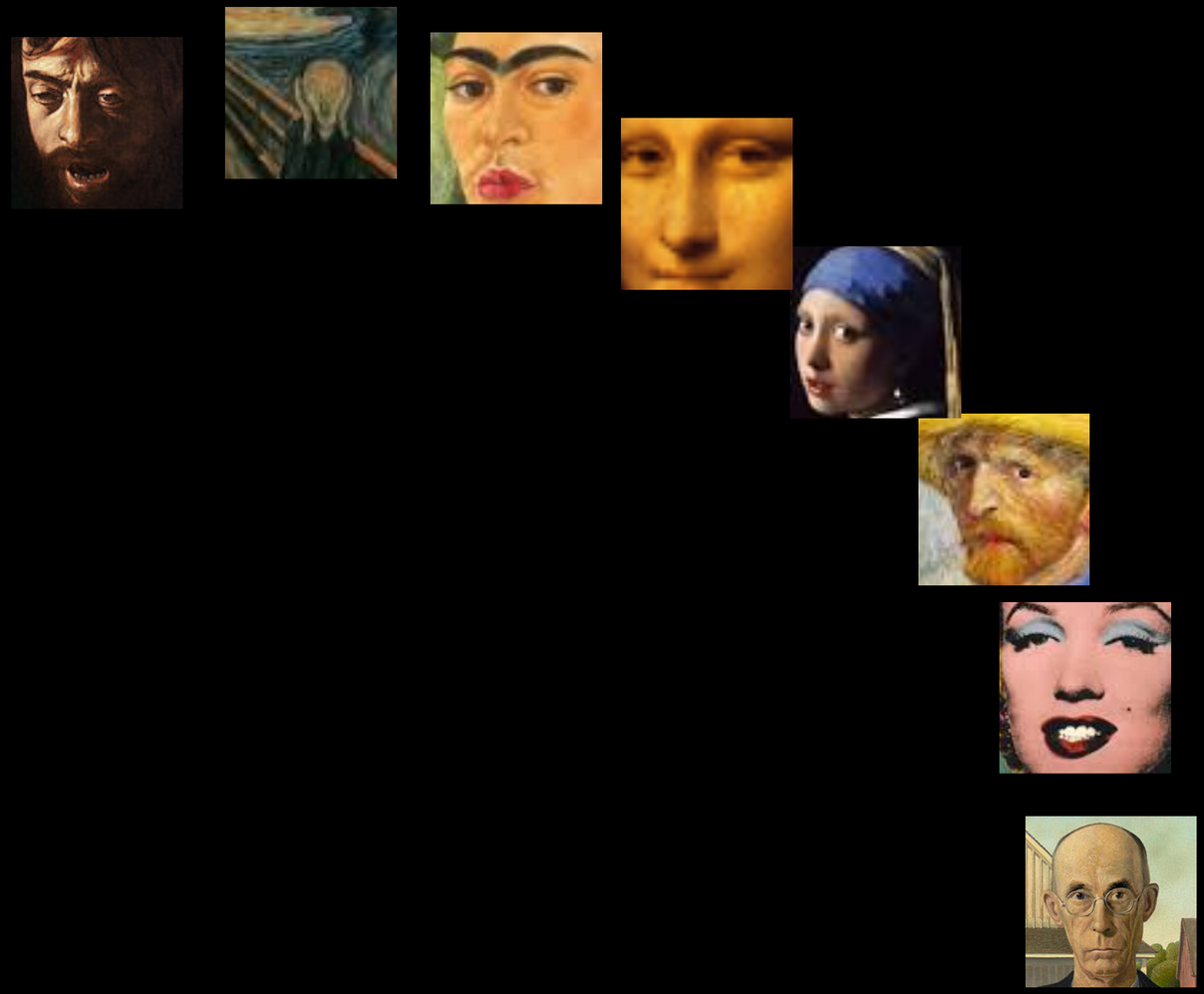
birthday problem



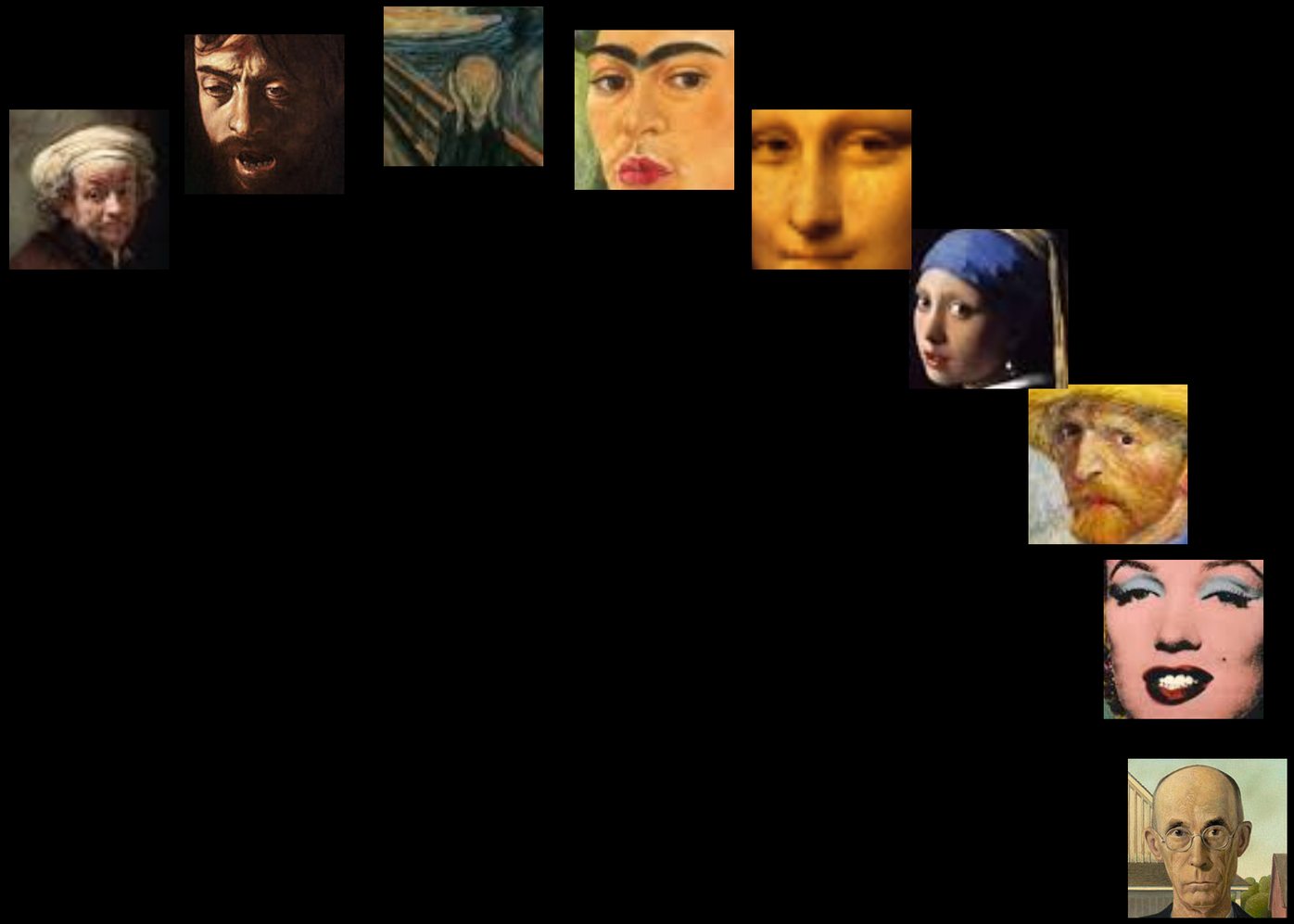
birthday problem



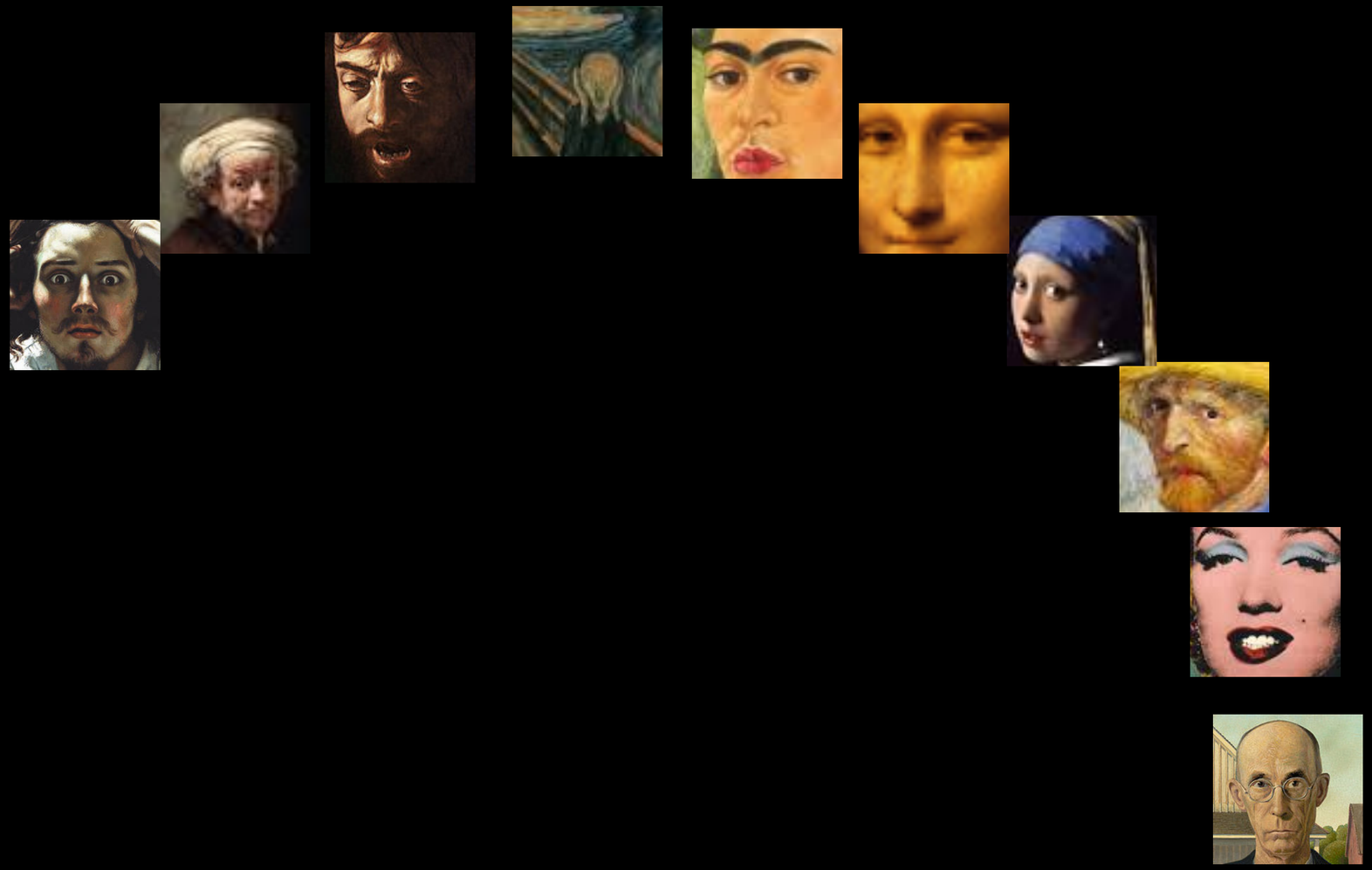
birthday problem



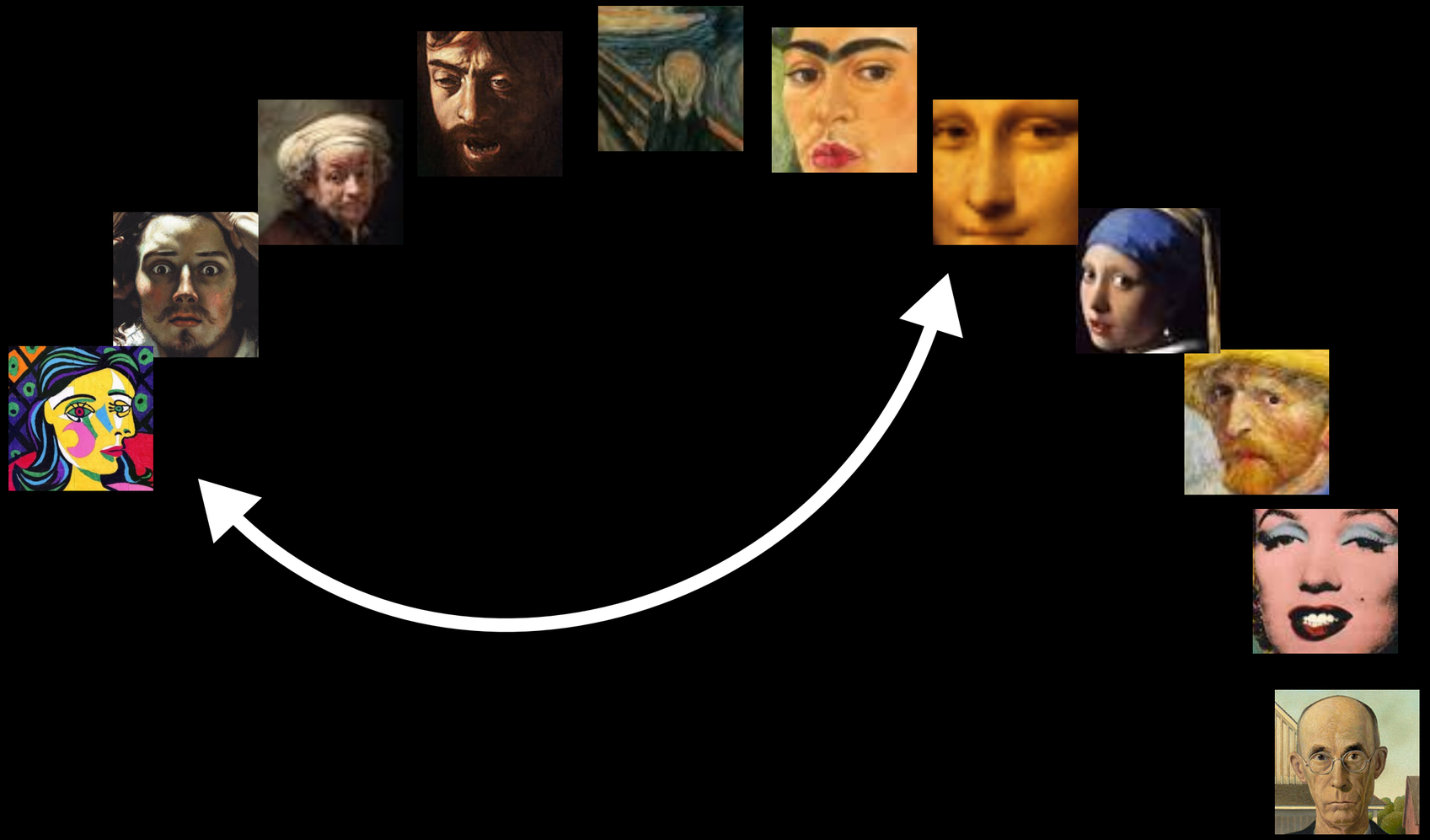
birthday problem



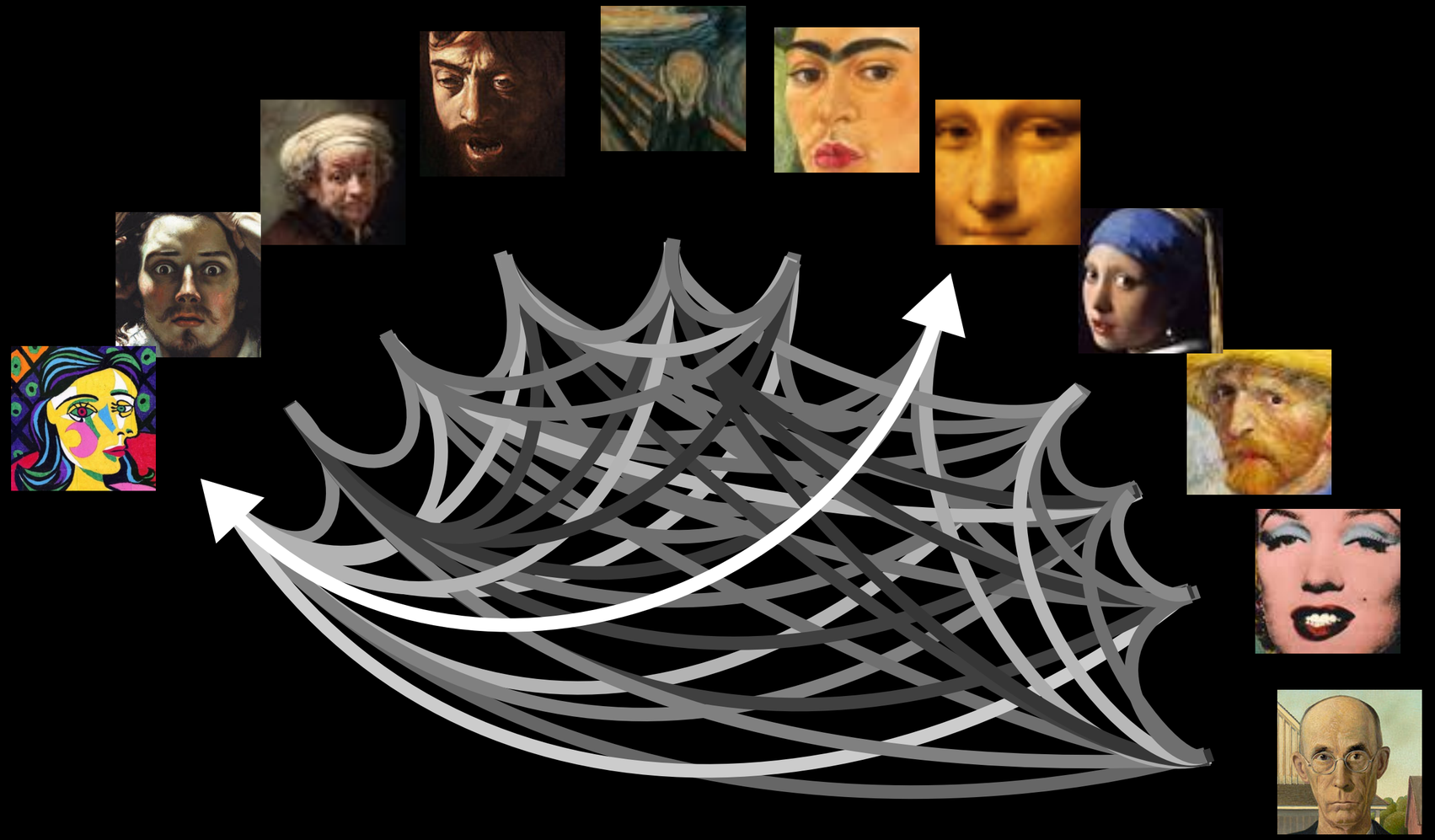
birthday problem



birthday problem

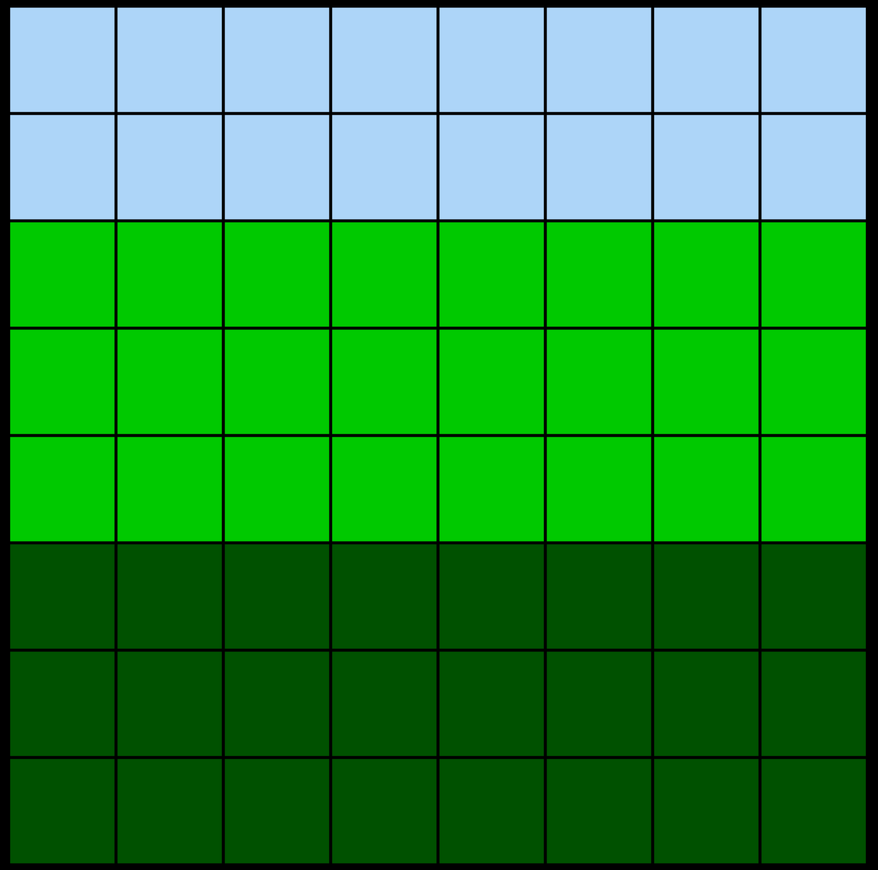


birthday problem

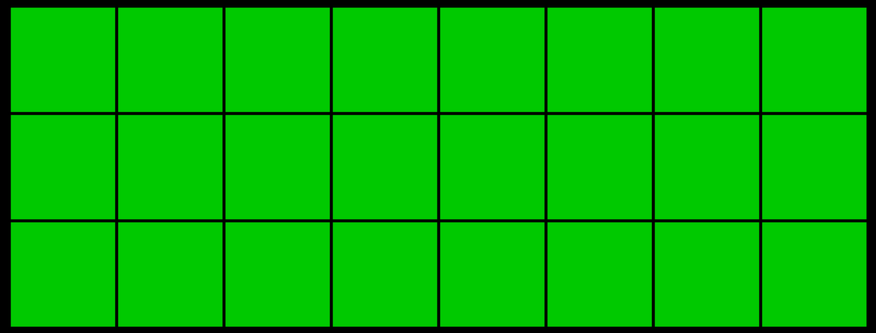


primitive #3: birthday heapspray

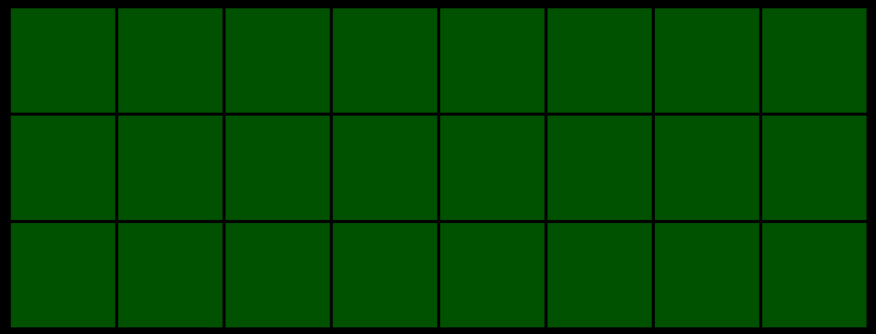
physical memory



attacker memory

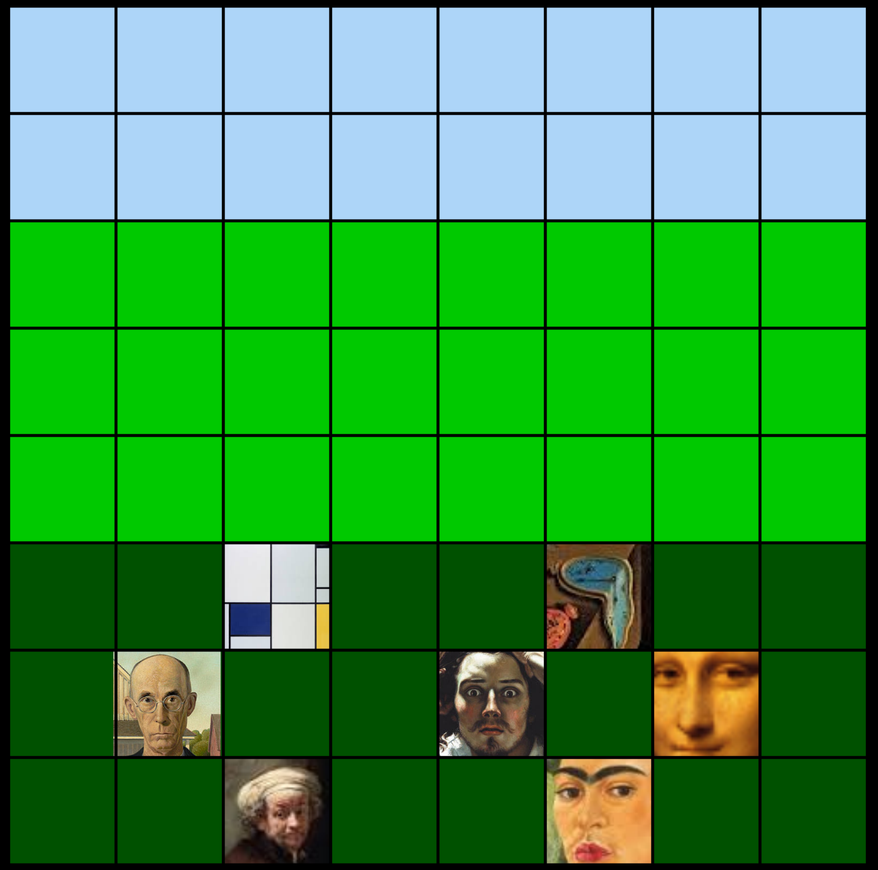


victim memory

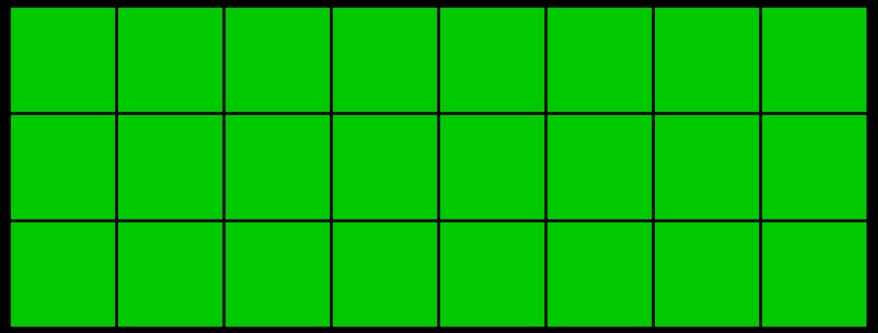


primitive #3: birthday heapspray

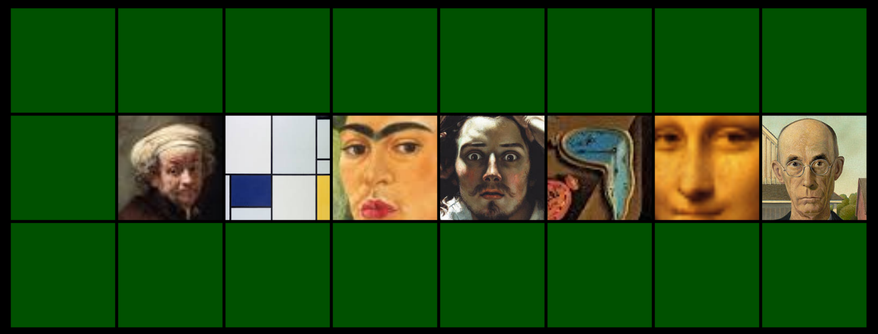
physical memory



attacker memory

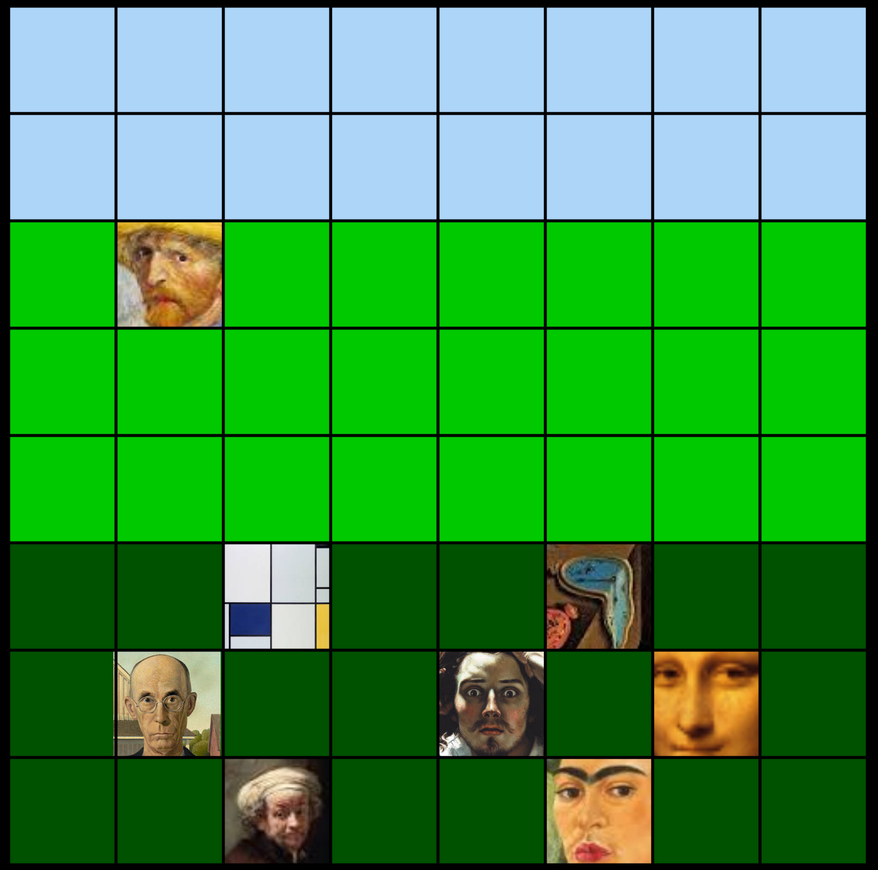


victim memory

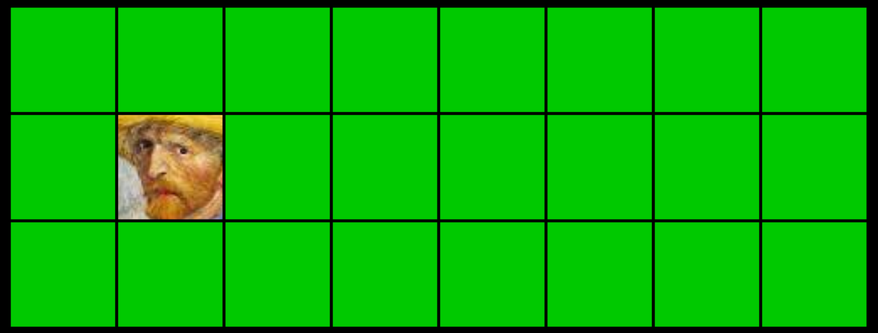


primitive #3: birthday heapspray

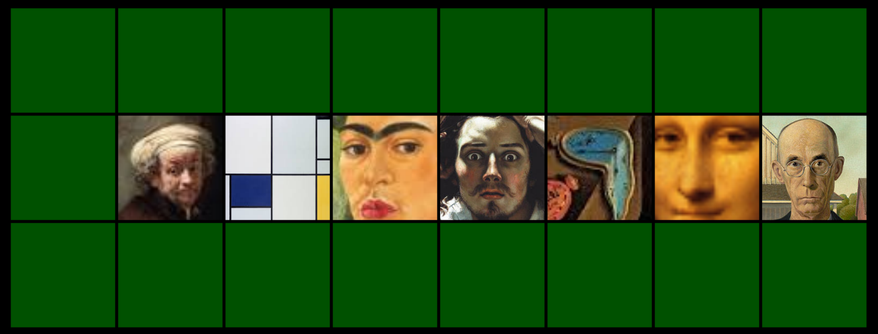
physical memory



attacker memory

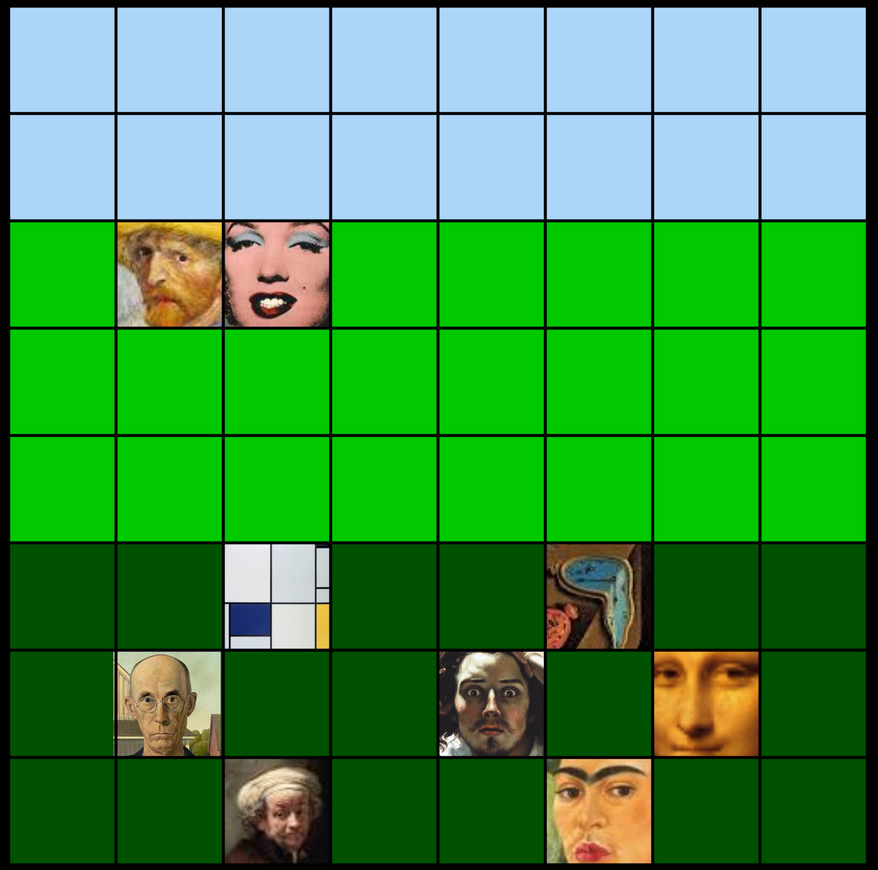


victim memory

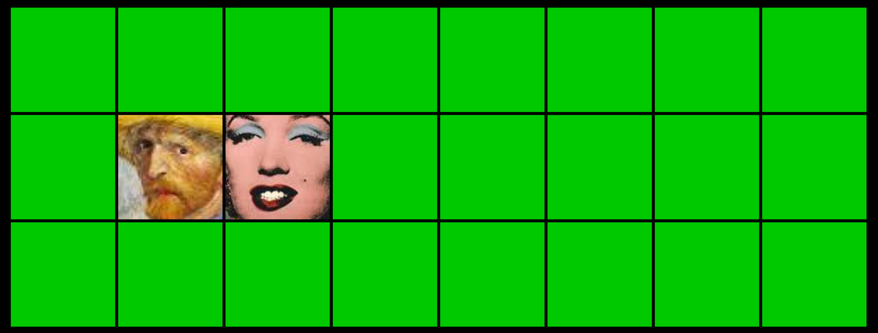


primitive #3: birthday heapspray

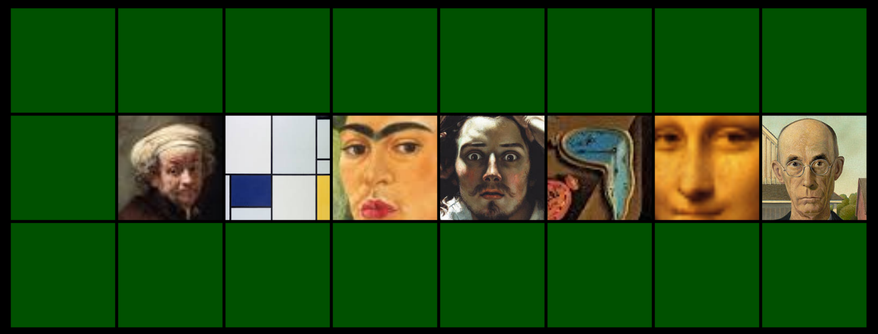
physical memory



attacker memory

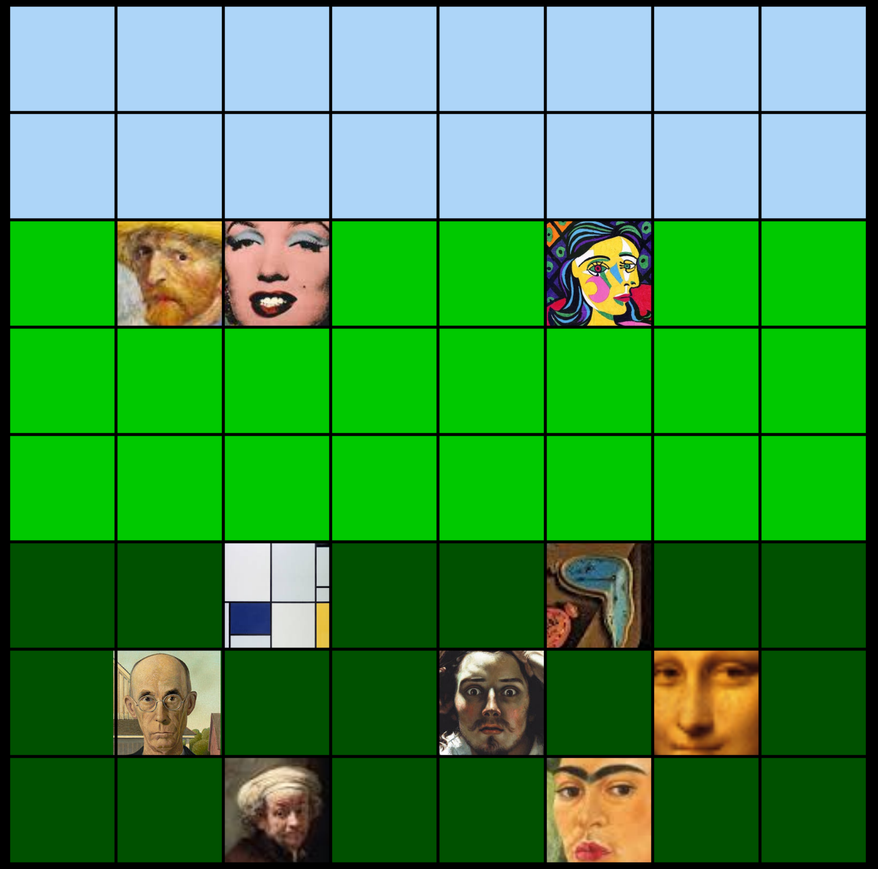


victim memory

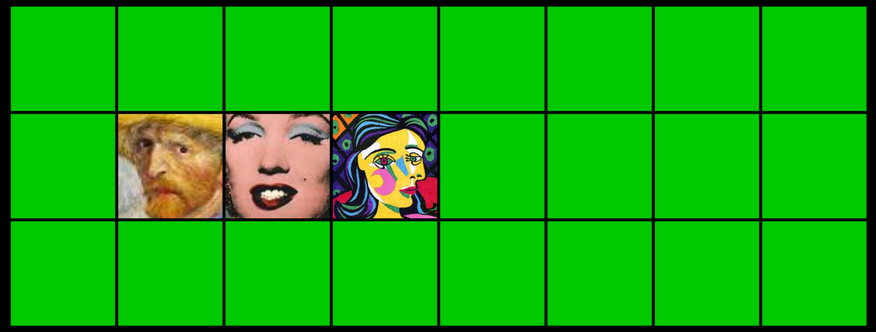


primitive #3: birthday heapspray

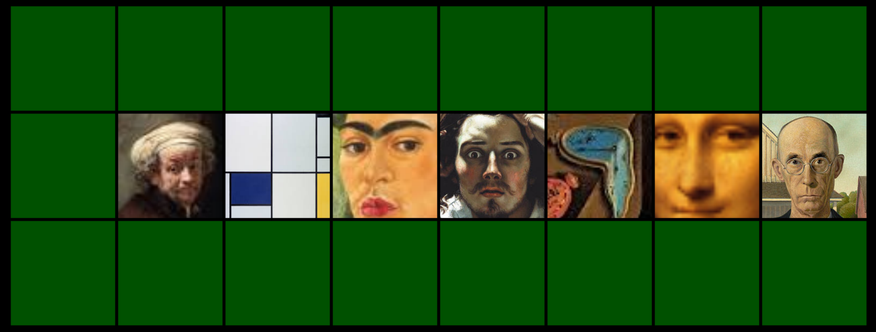
physical memory



attacker memory

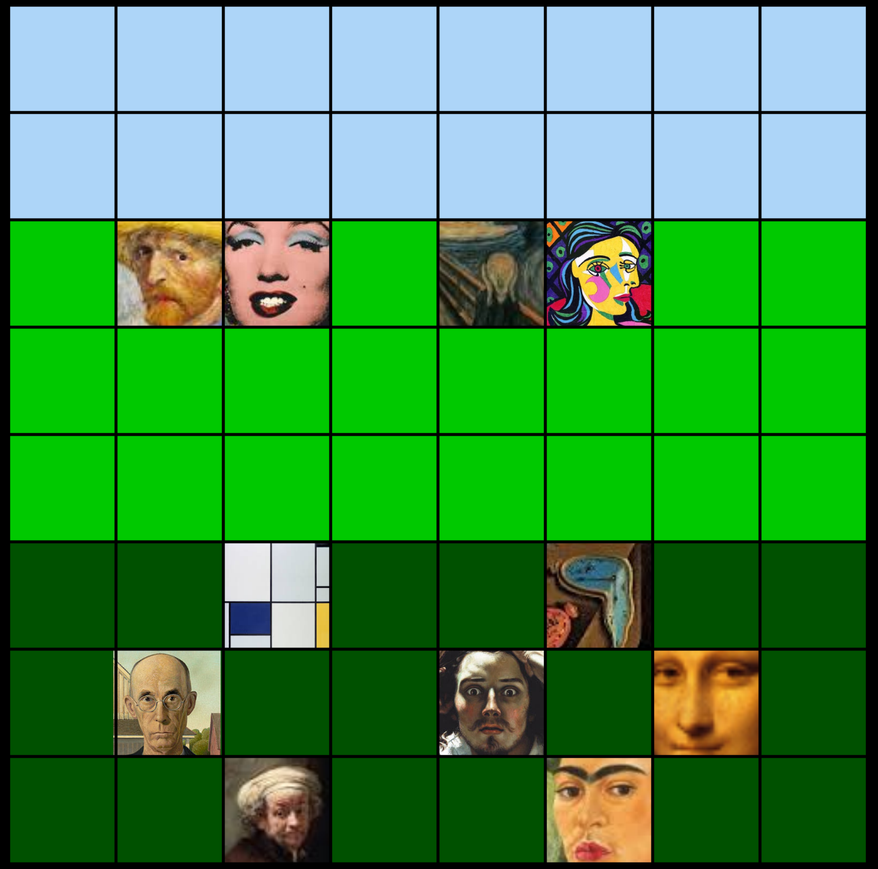


victim memory

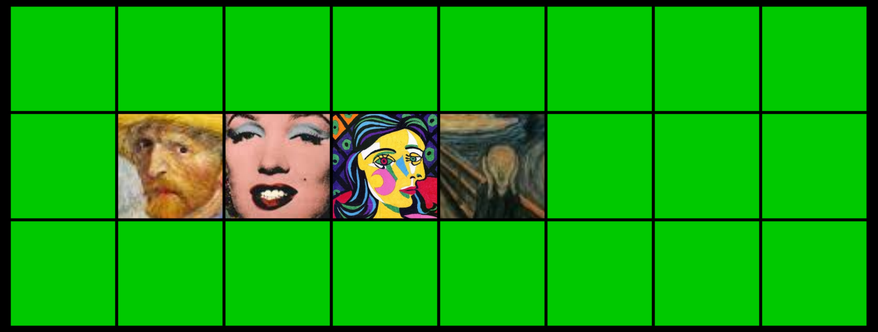


primitive #3: birthday heapspray

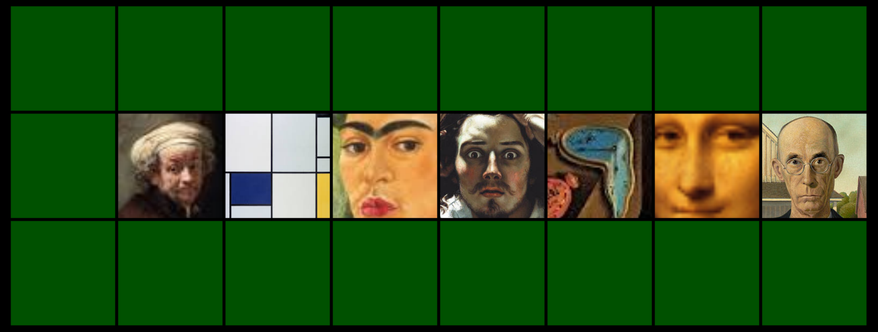
physical memory



attacker memory

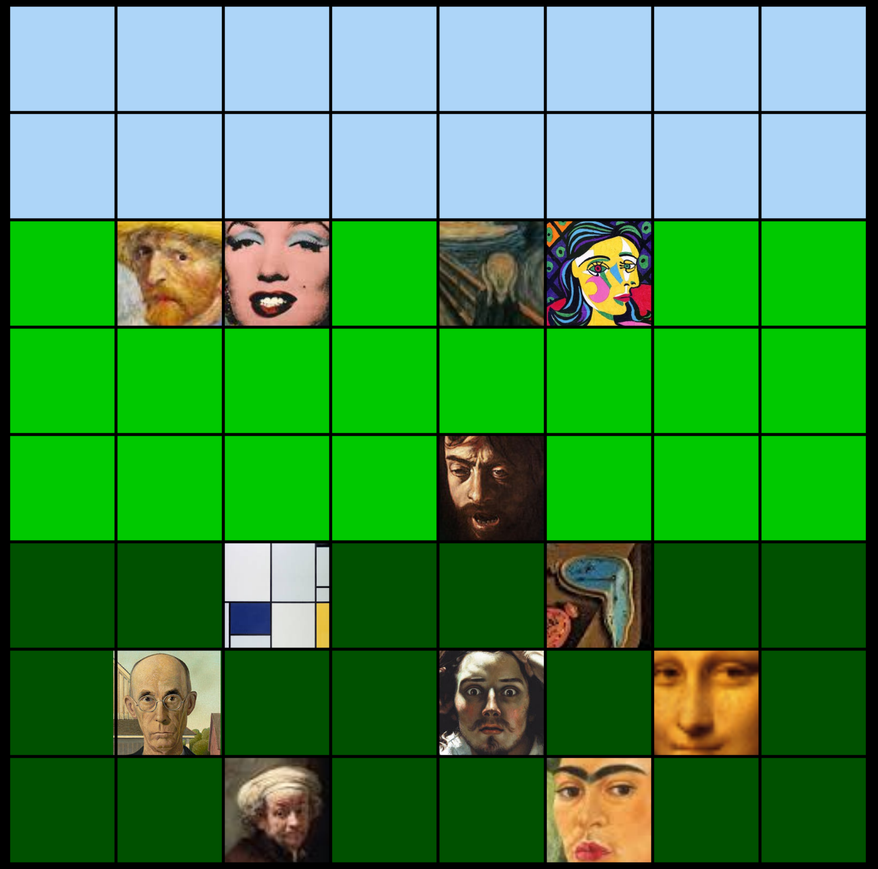


victim memory

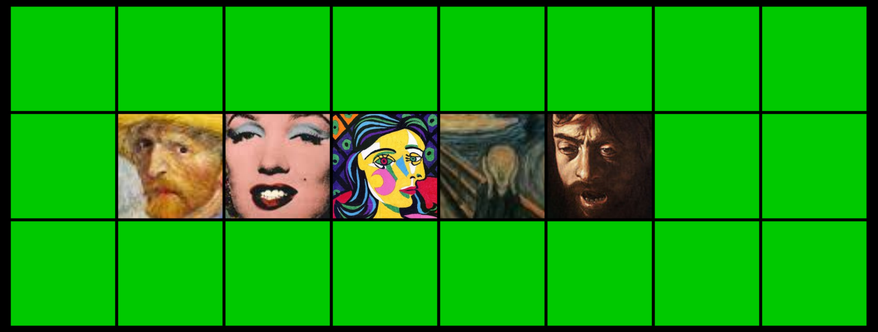


primitive #3: birthday heapspray

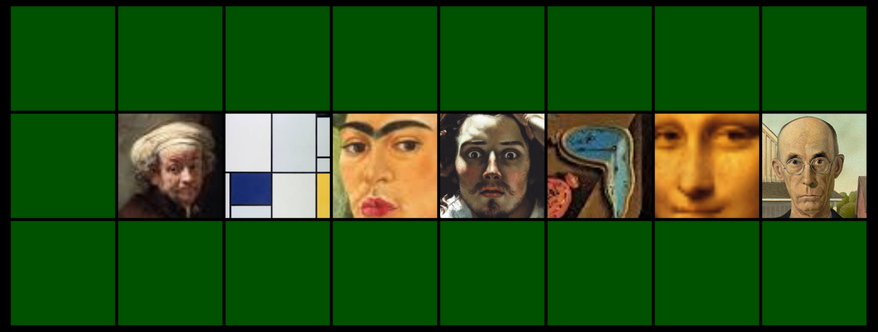
physical memory



attacker memory

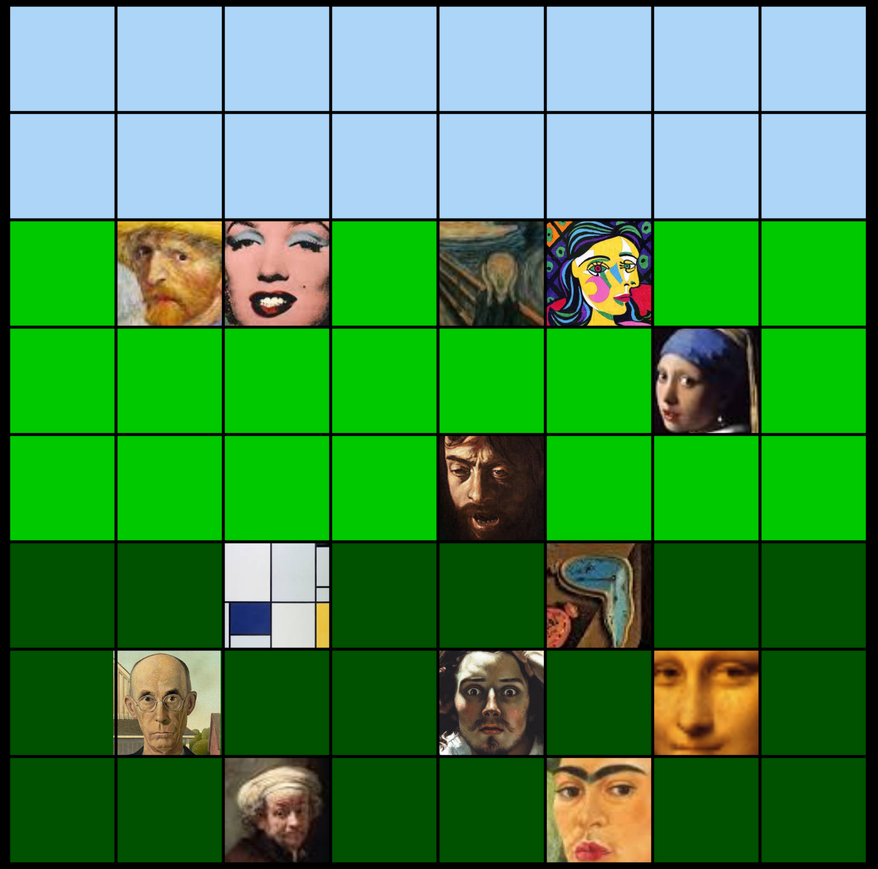


victim memory

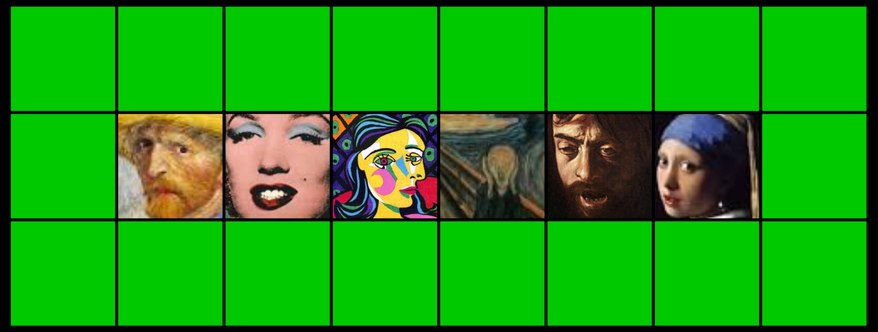


primitive #3: birthday heapspray

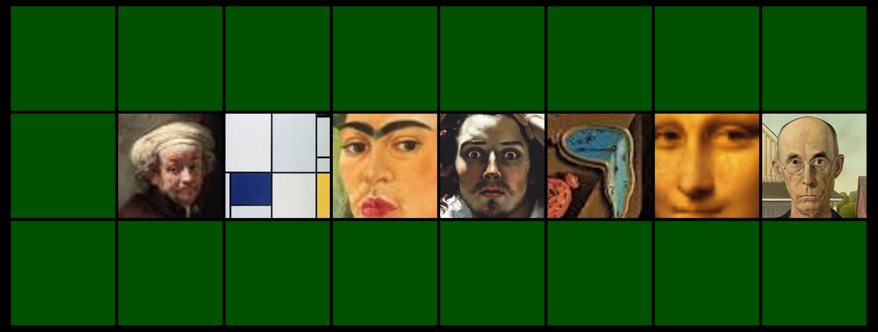
physical memory



attacker memory

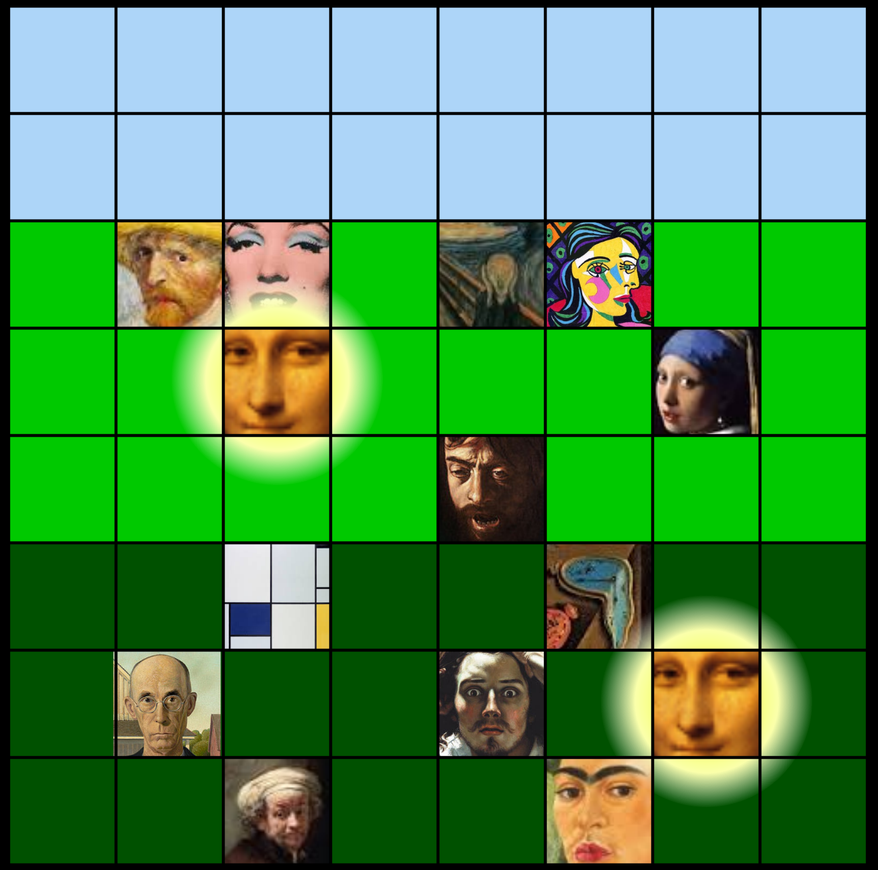


victim memory

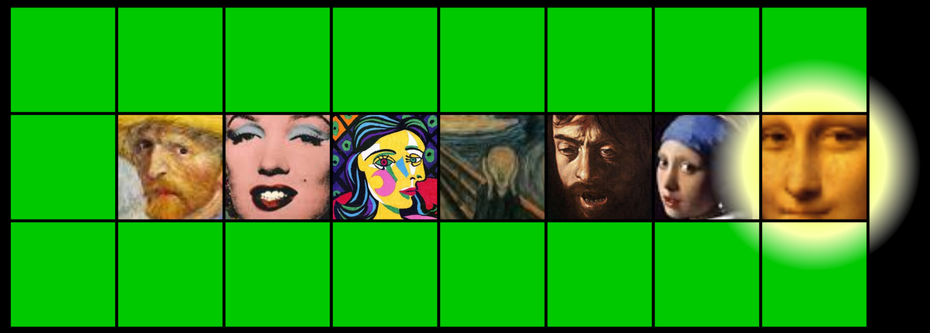


primitive #3: birthday heapspray

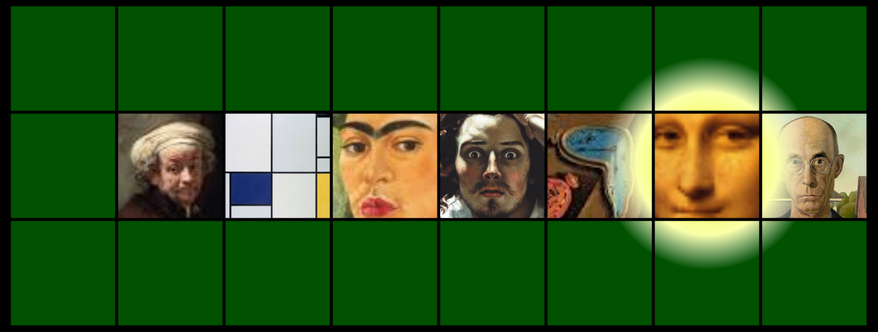
physical memory



attacker memory



victim memory

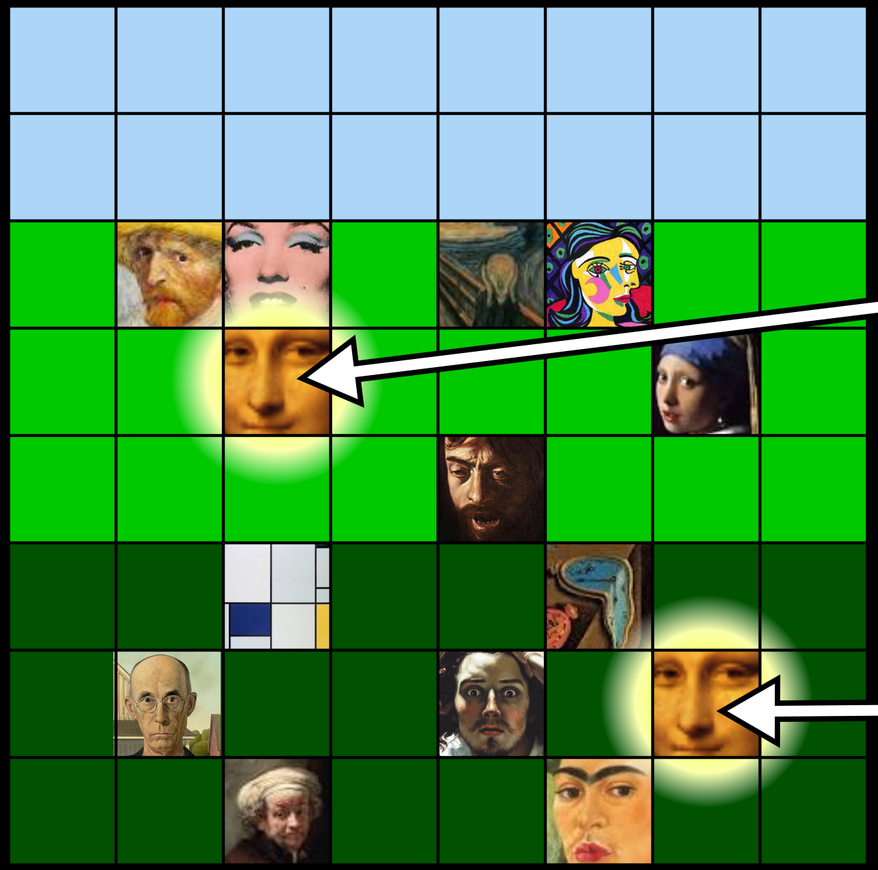


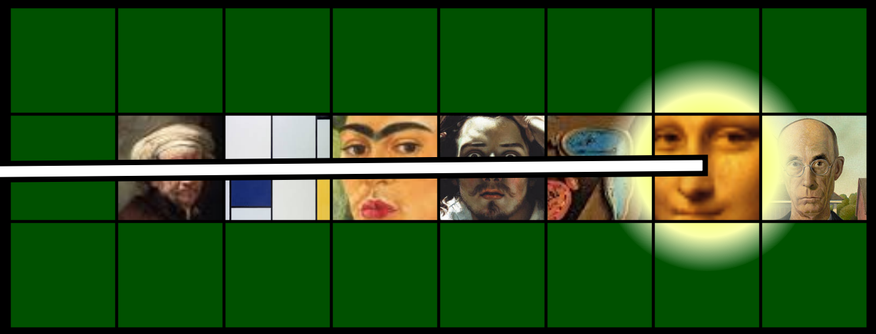
primitive #3: birthday heapspray

physical memory

attacker memory

victim memory



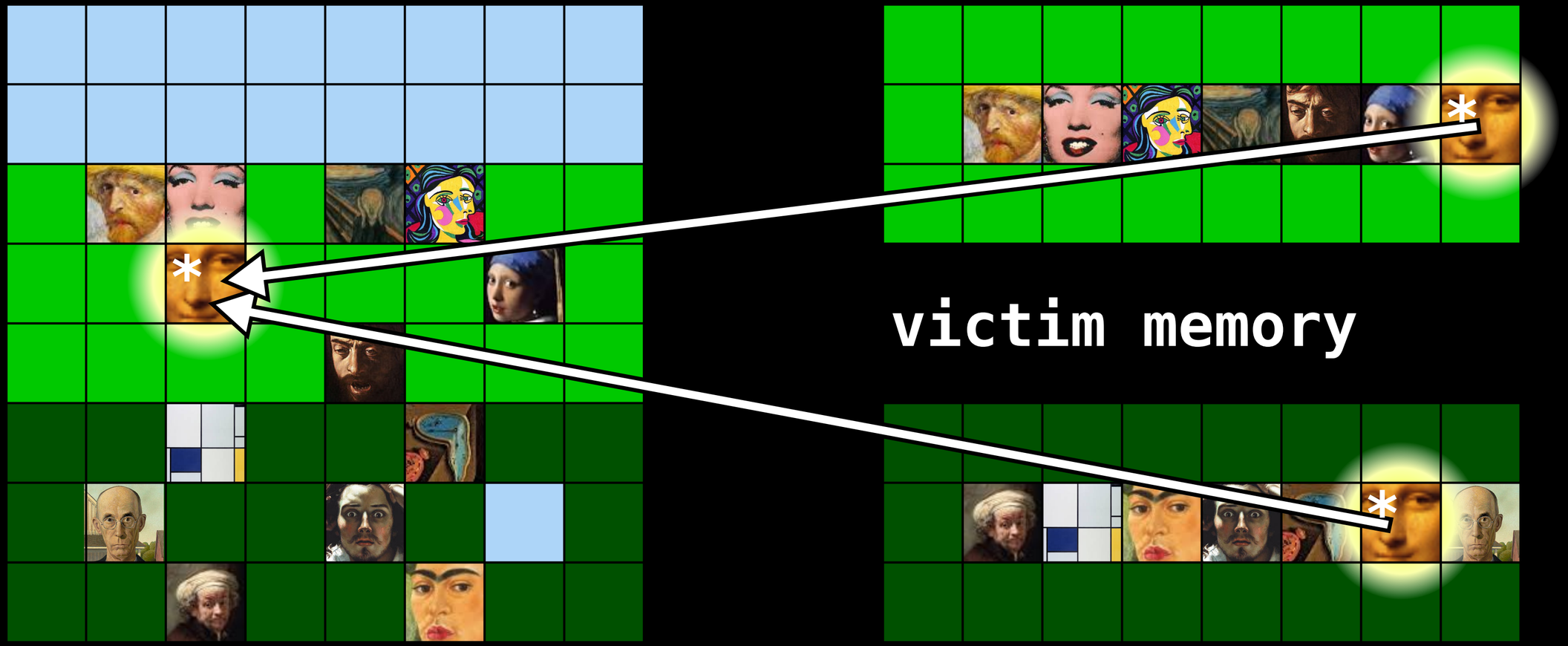


primitive #3: birthday heapspray

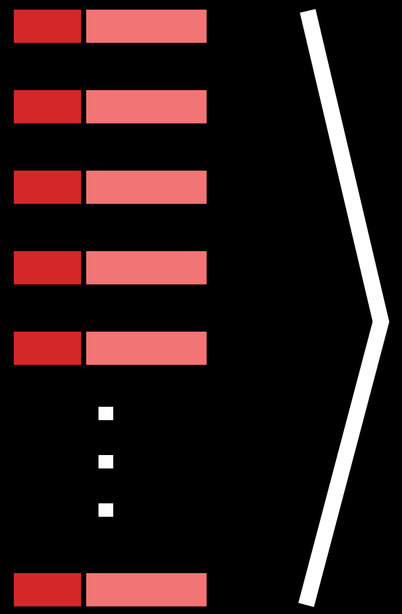
physical memory

attacker memory

victim memory

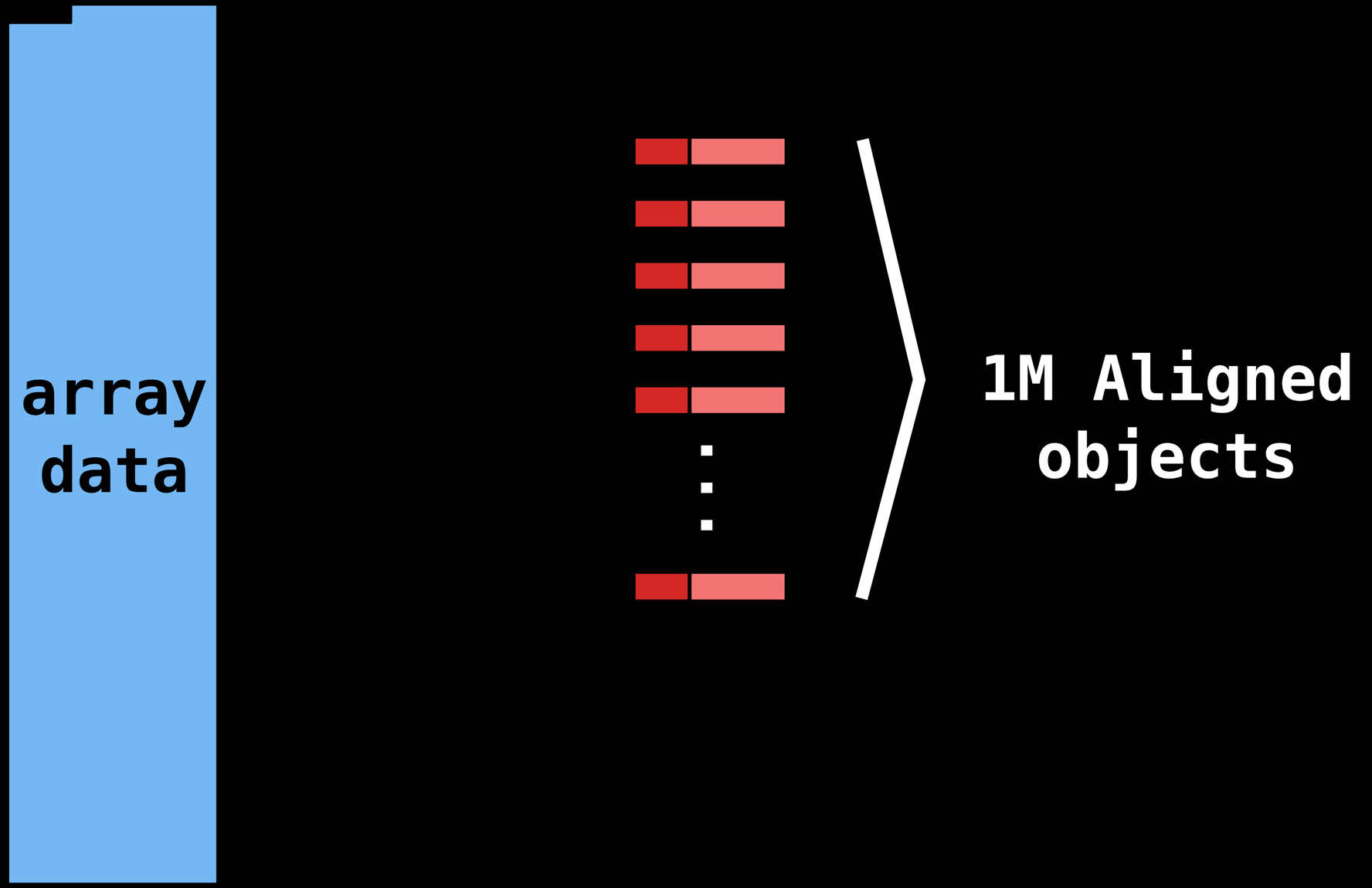


Creating Secret Pages

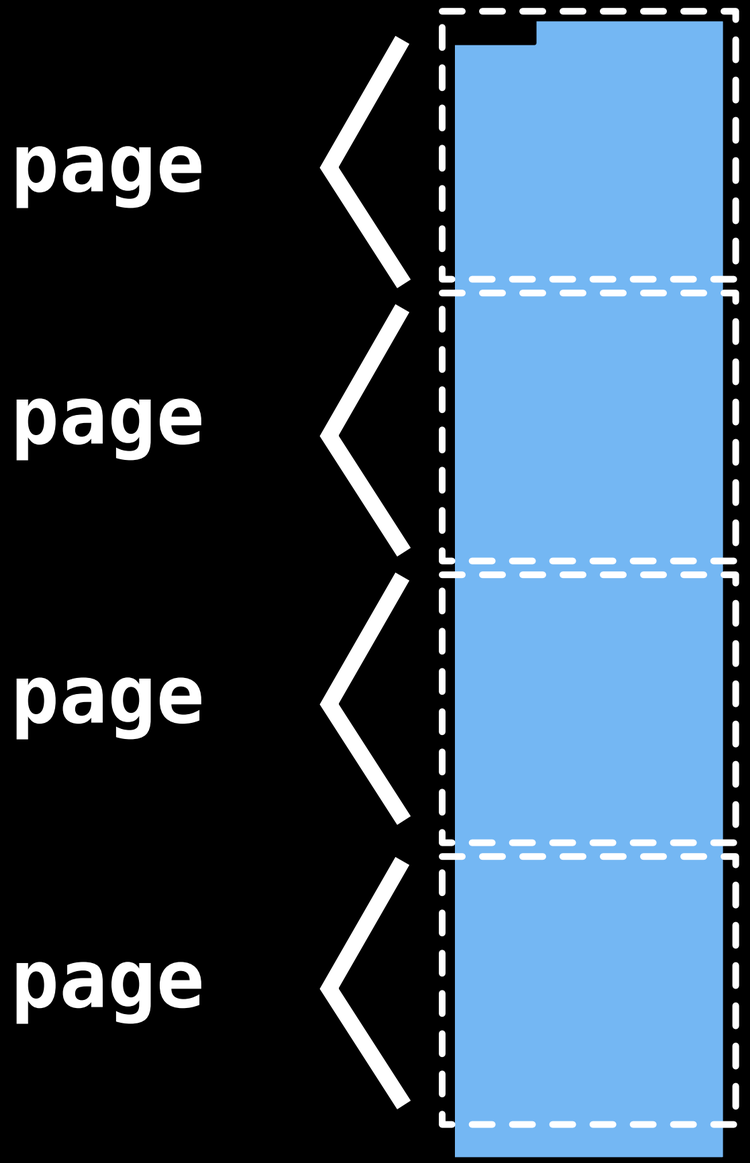


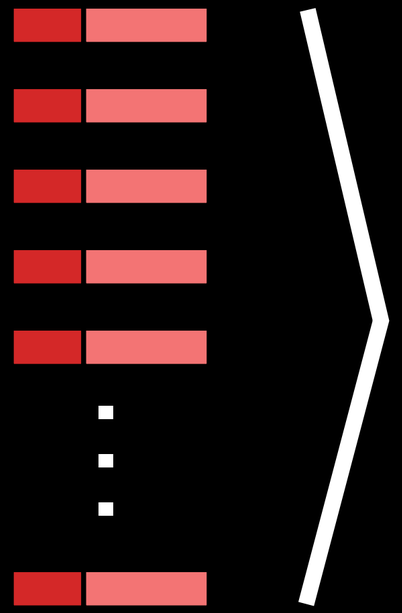
1M Aligned objects

Creating Secret Pages



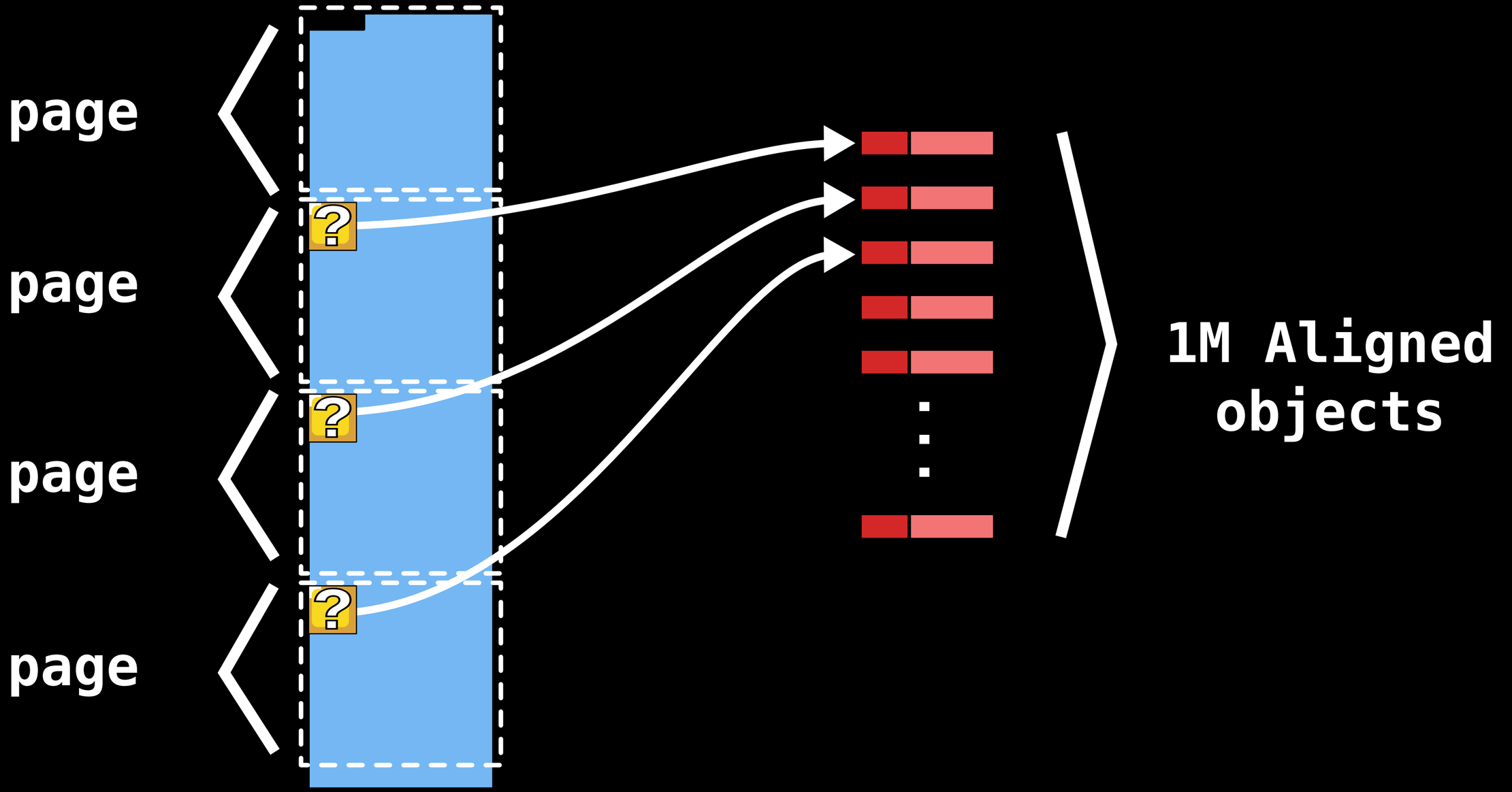
Creating Secret Pages



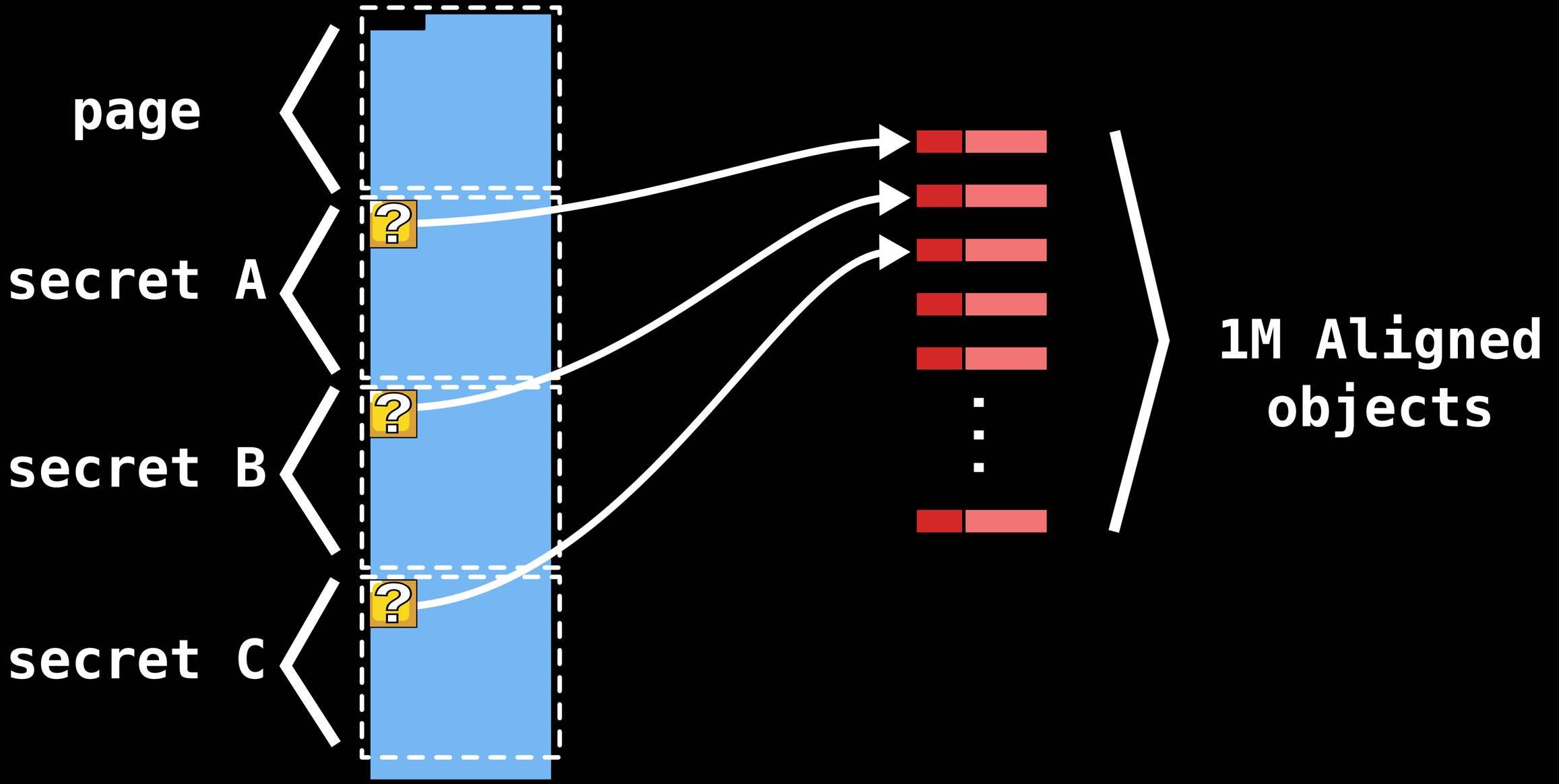


1M Aligned objects

Creating Secret Pages



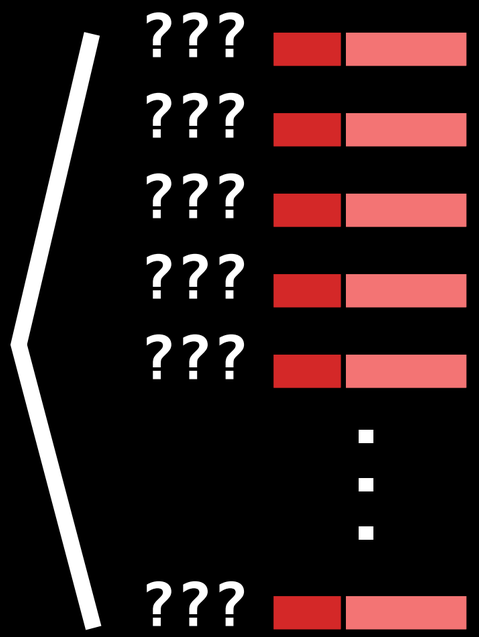
Creating Secret Pages



Creating Probe Pages

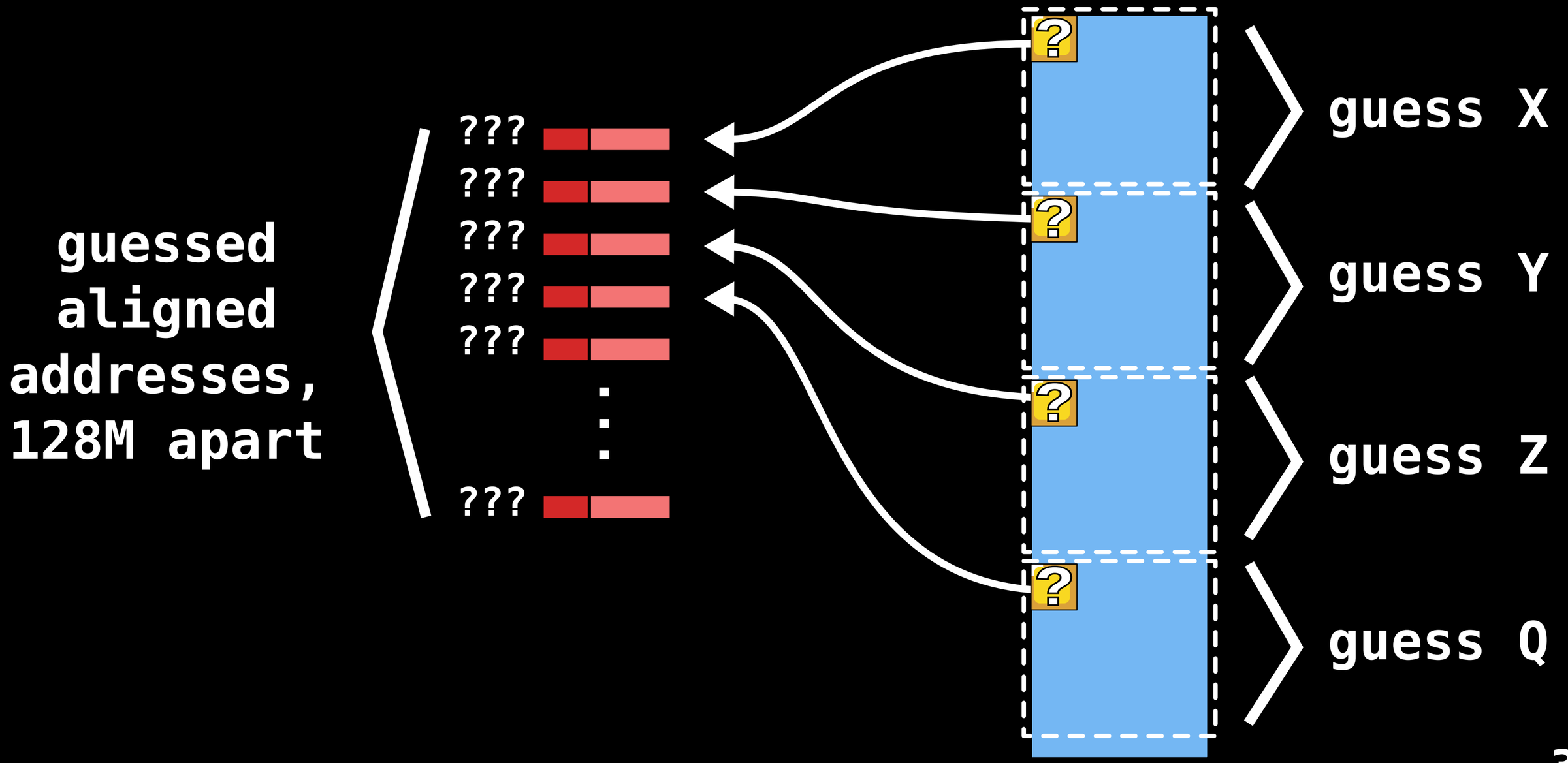
Creating Probe Pages

guessed aligned addresses, 128M apart

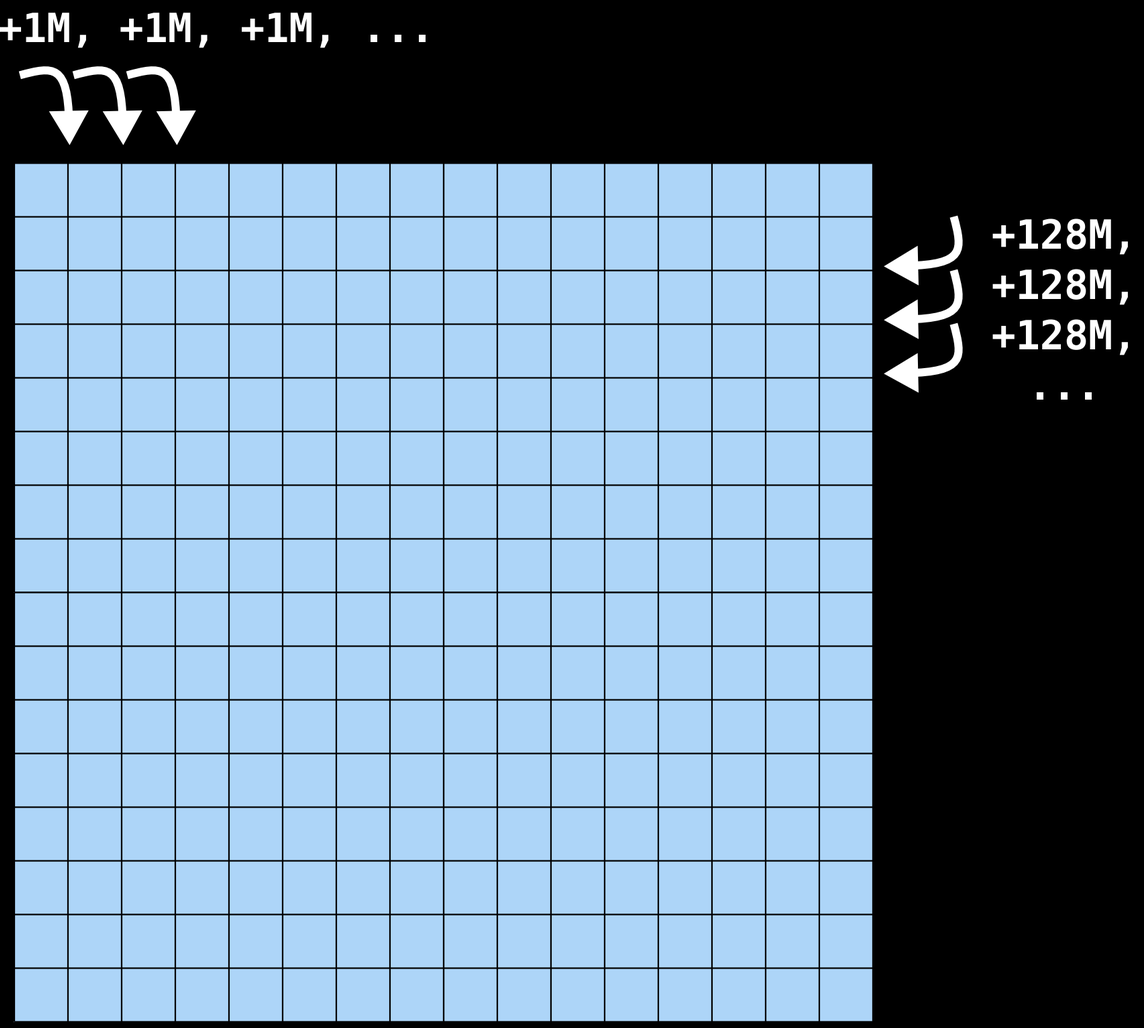


typed array data

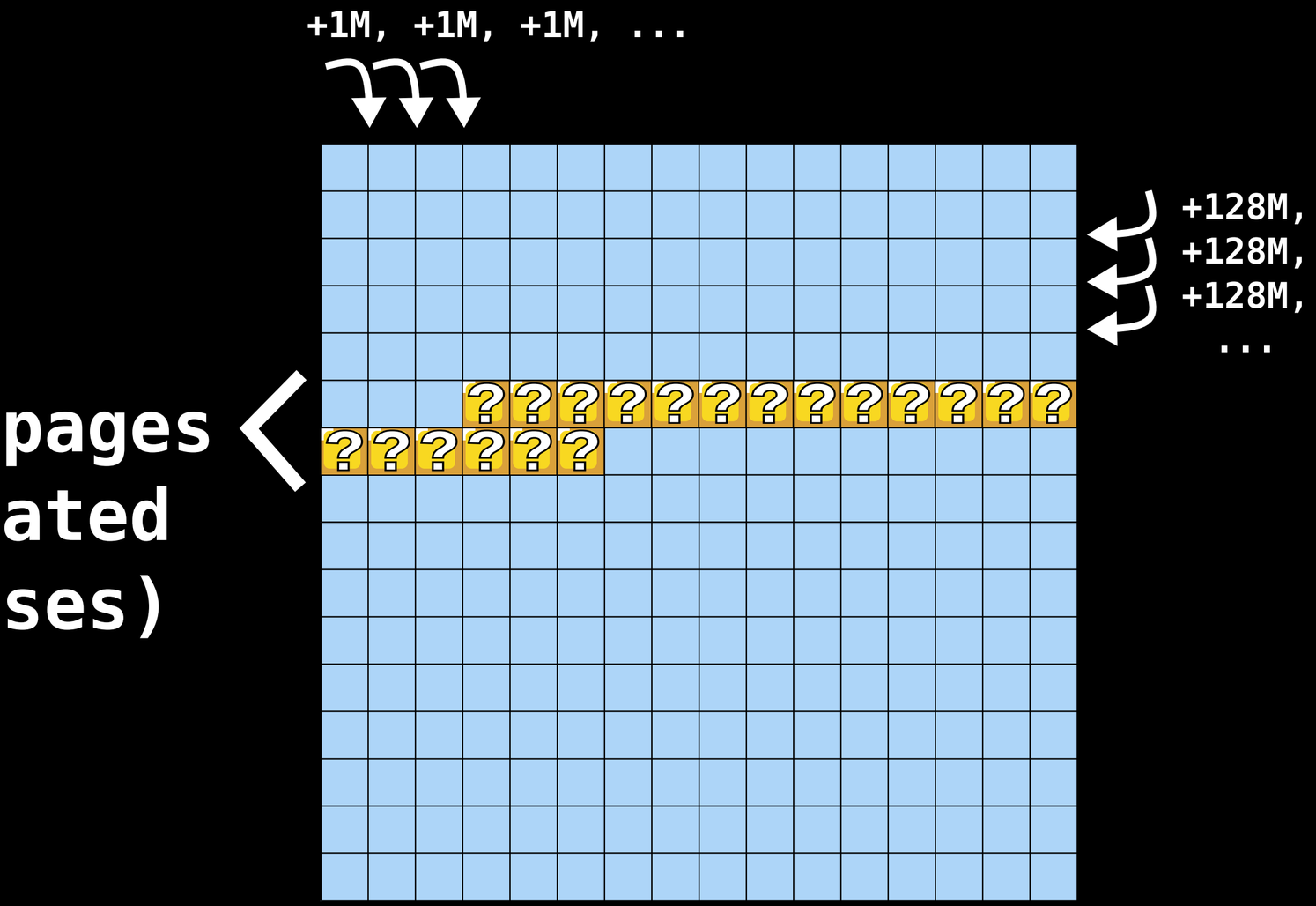
Creating Probe Pages



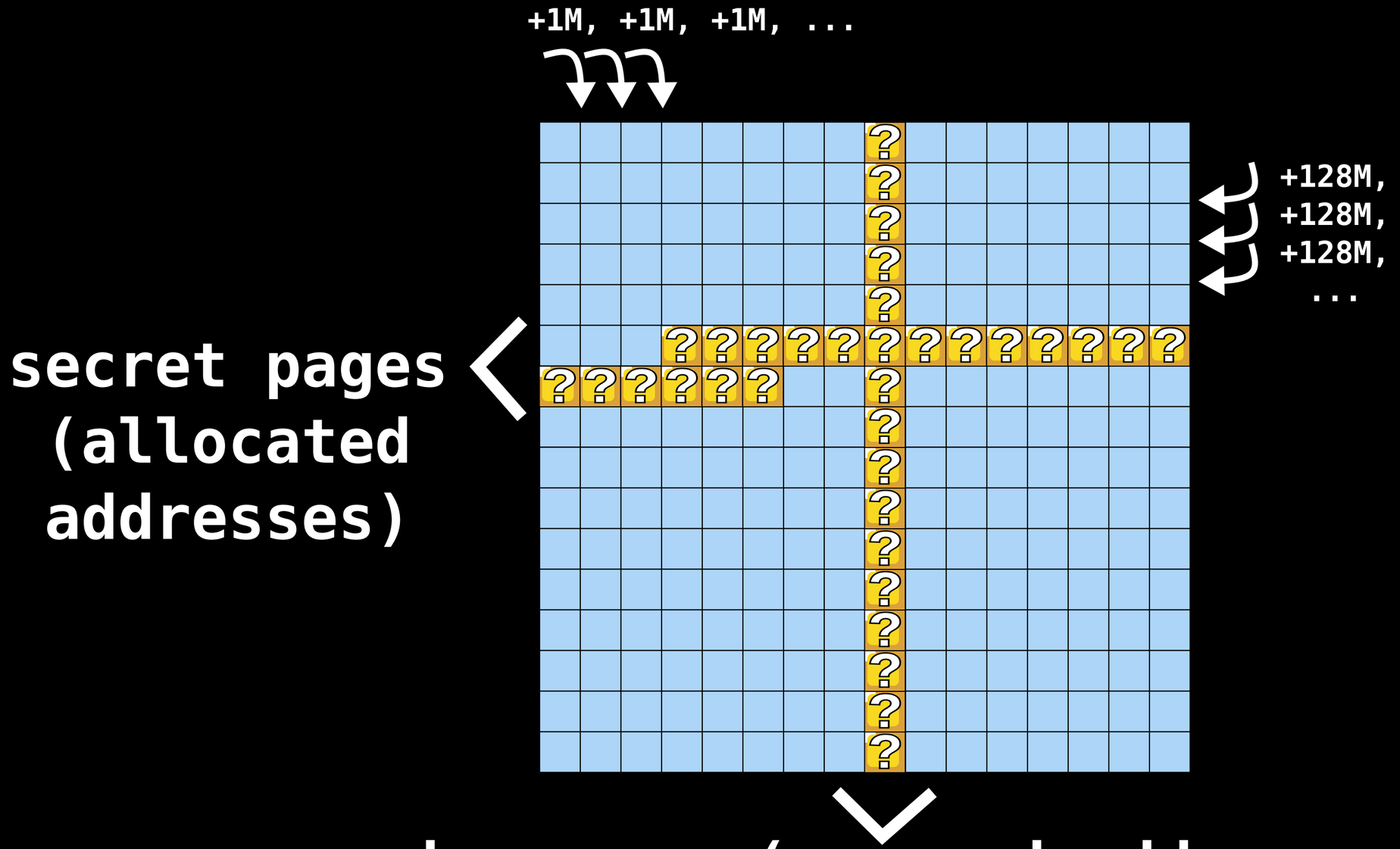
Birthday heap spray



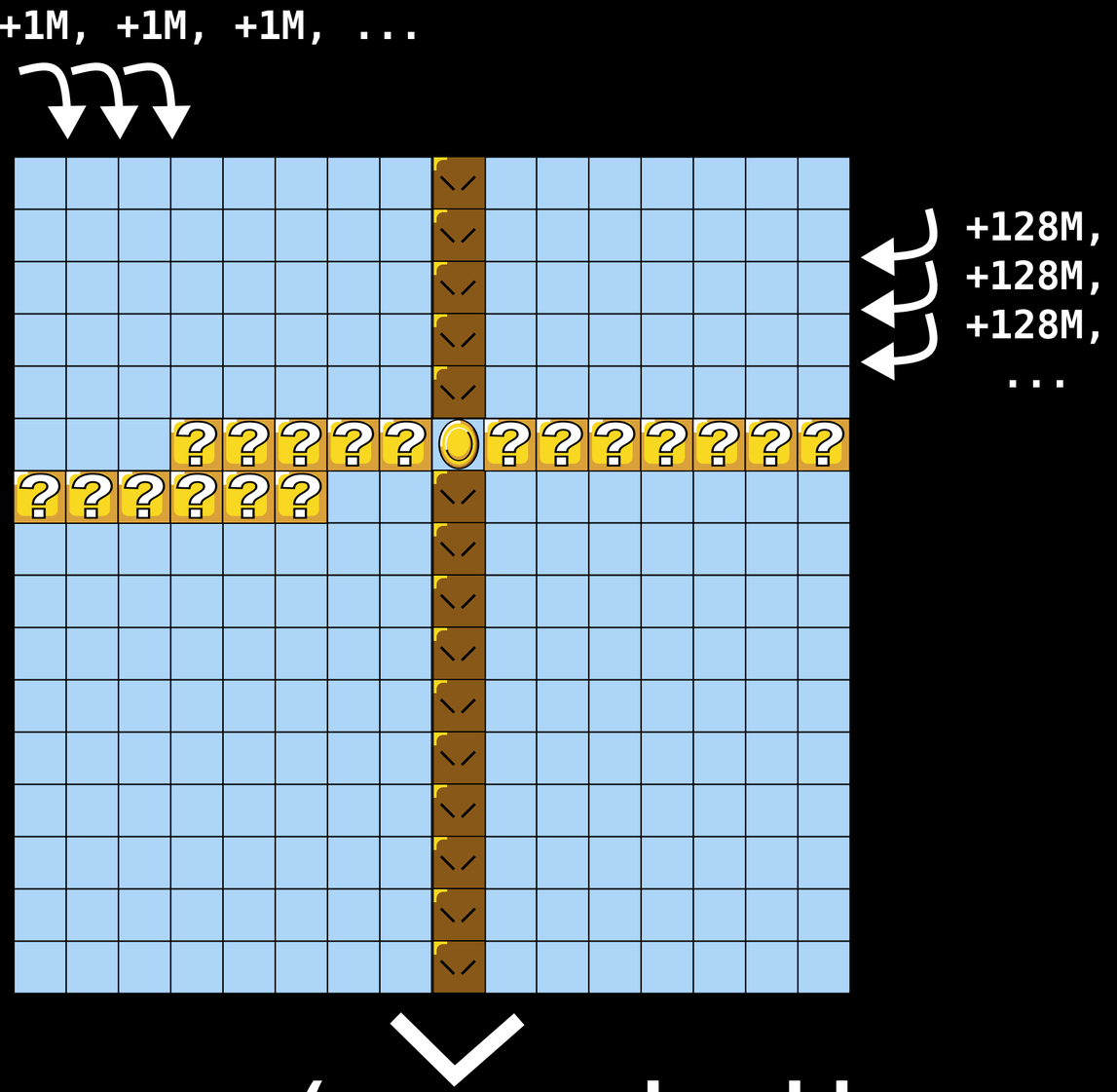
Birthday heap spray



Birthday heap spray



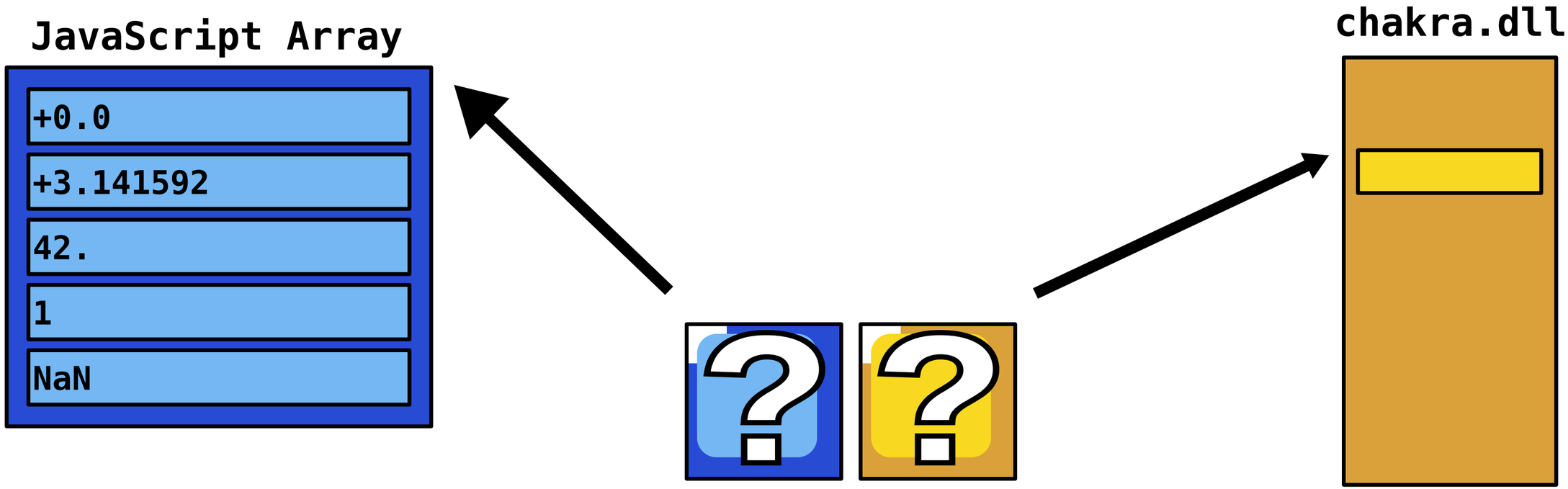
Birthday heap spray



Outline:

Deduplication

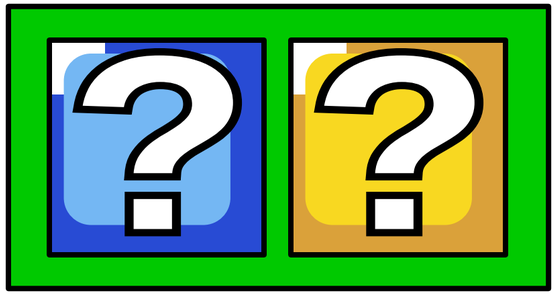
- leak heap & code addresses



Outline:

Deduplication

- leak heap & code addresses
- create a fake object



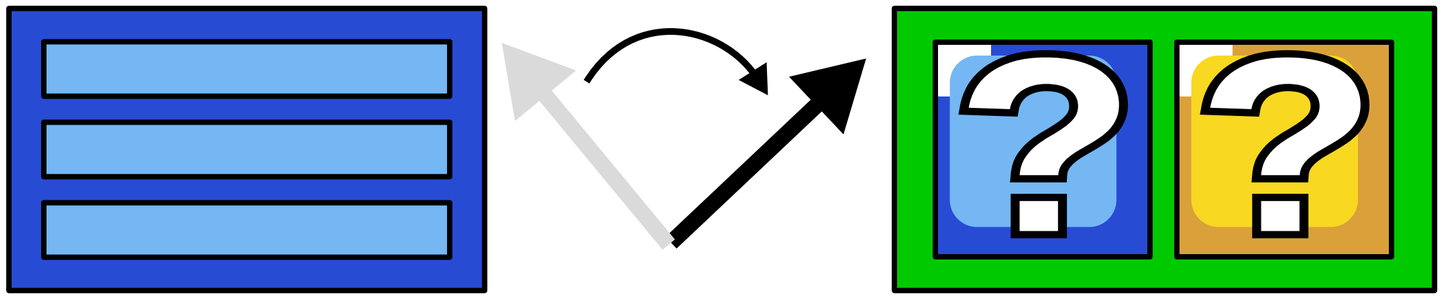
Outline:

Deduplication

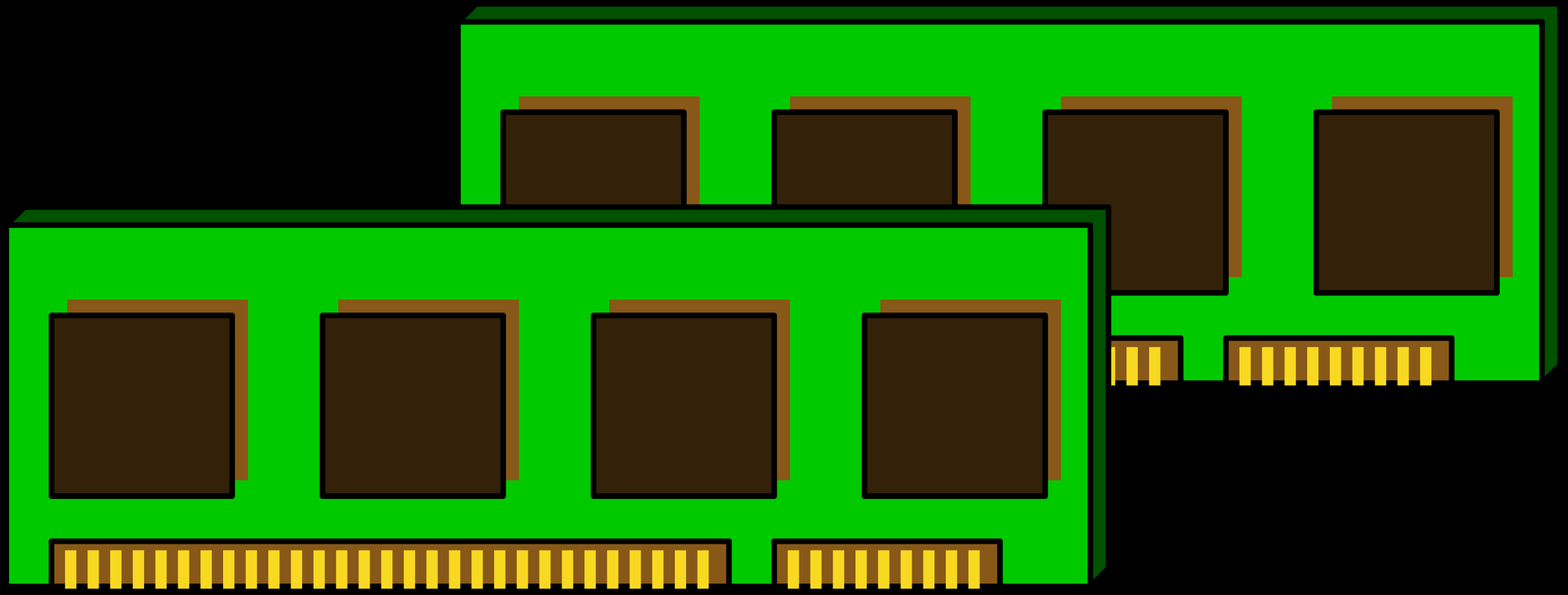
- leak heap & code addresses
- create a fake object

Rowhammer

- create reference to our fake object

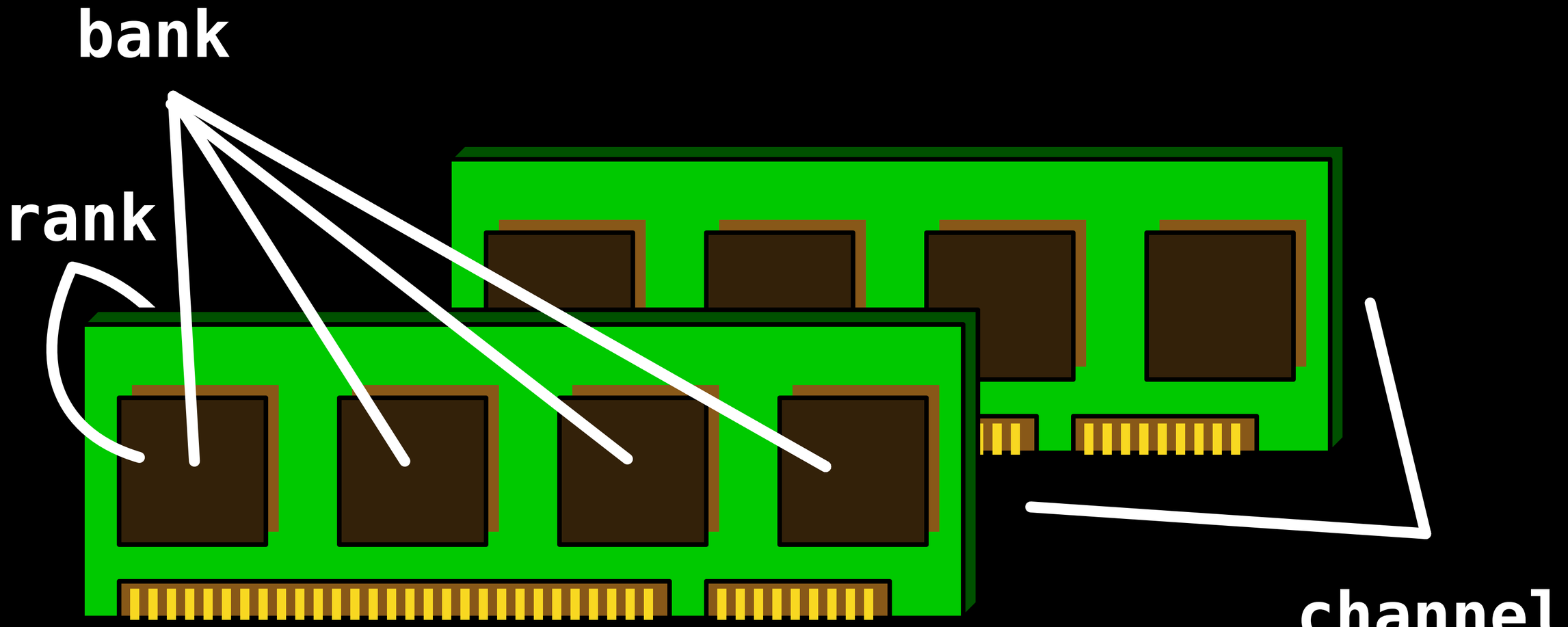


rowhammer attack



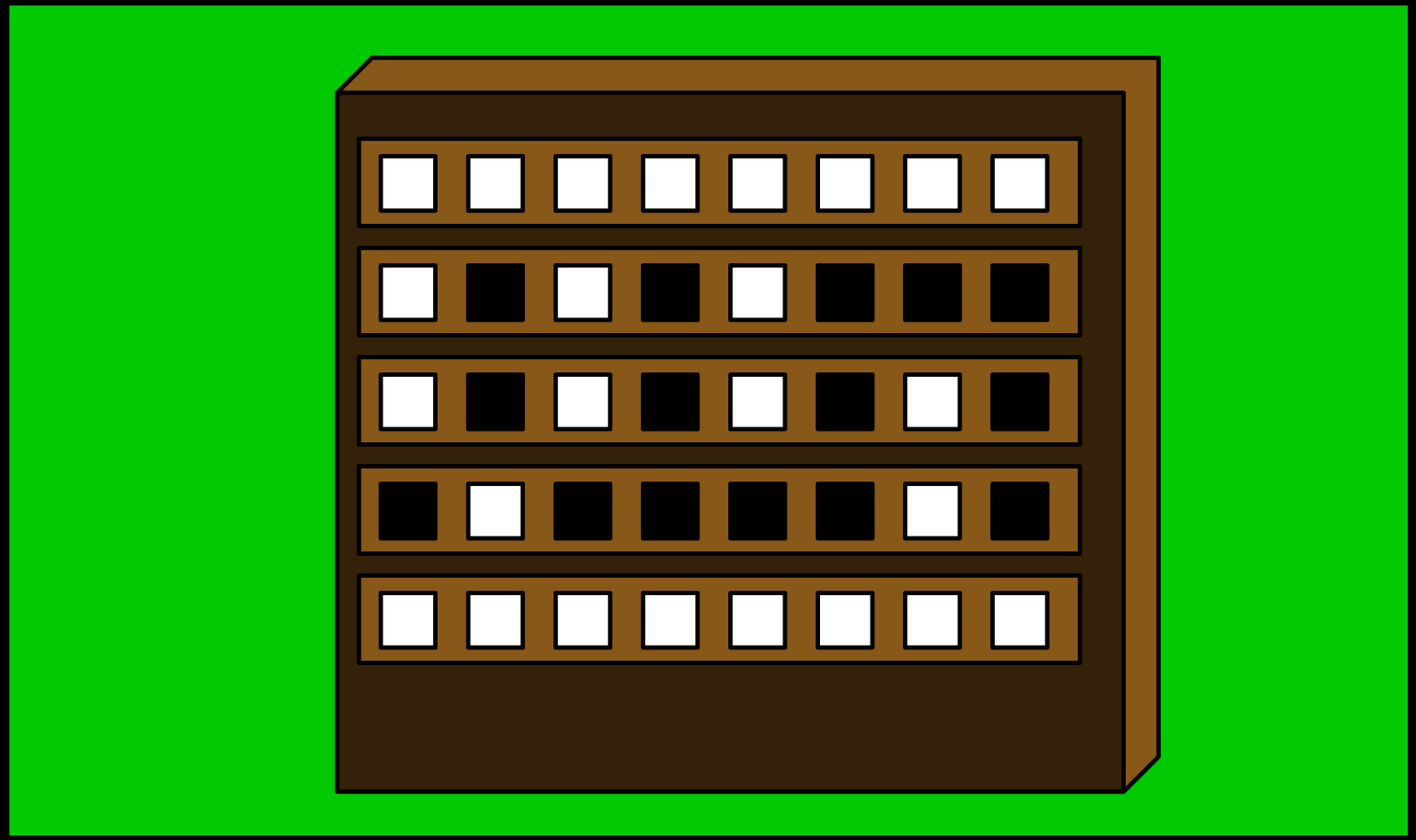
DDR memory

rowhammer attack

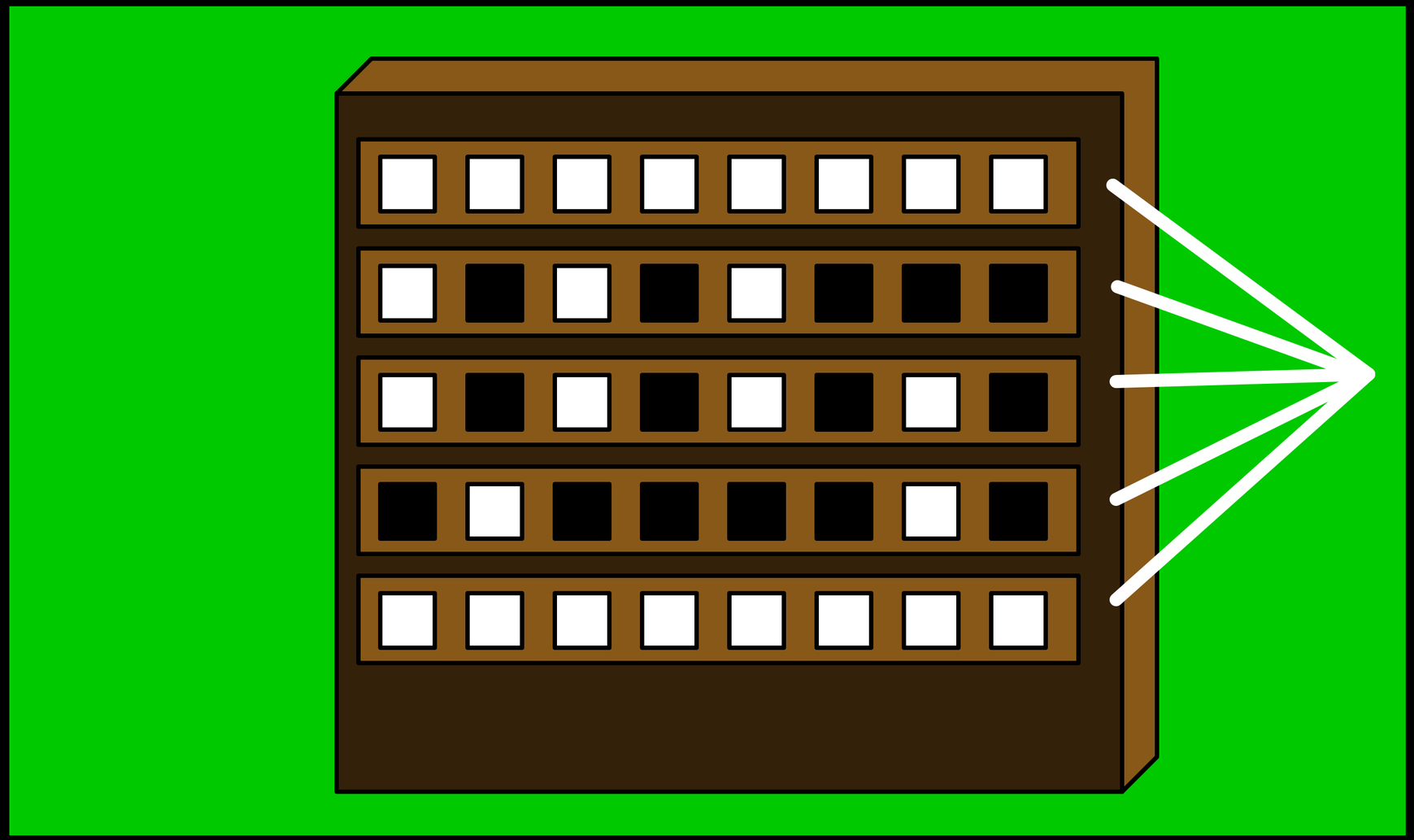


DDR memory

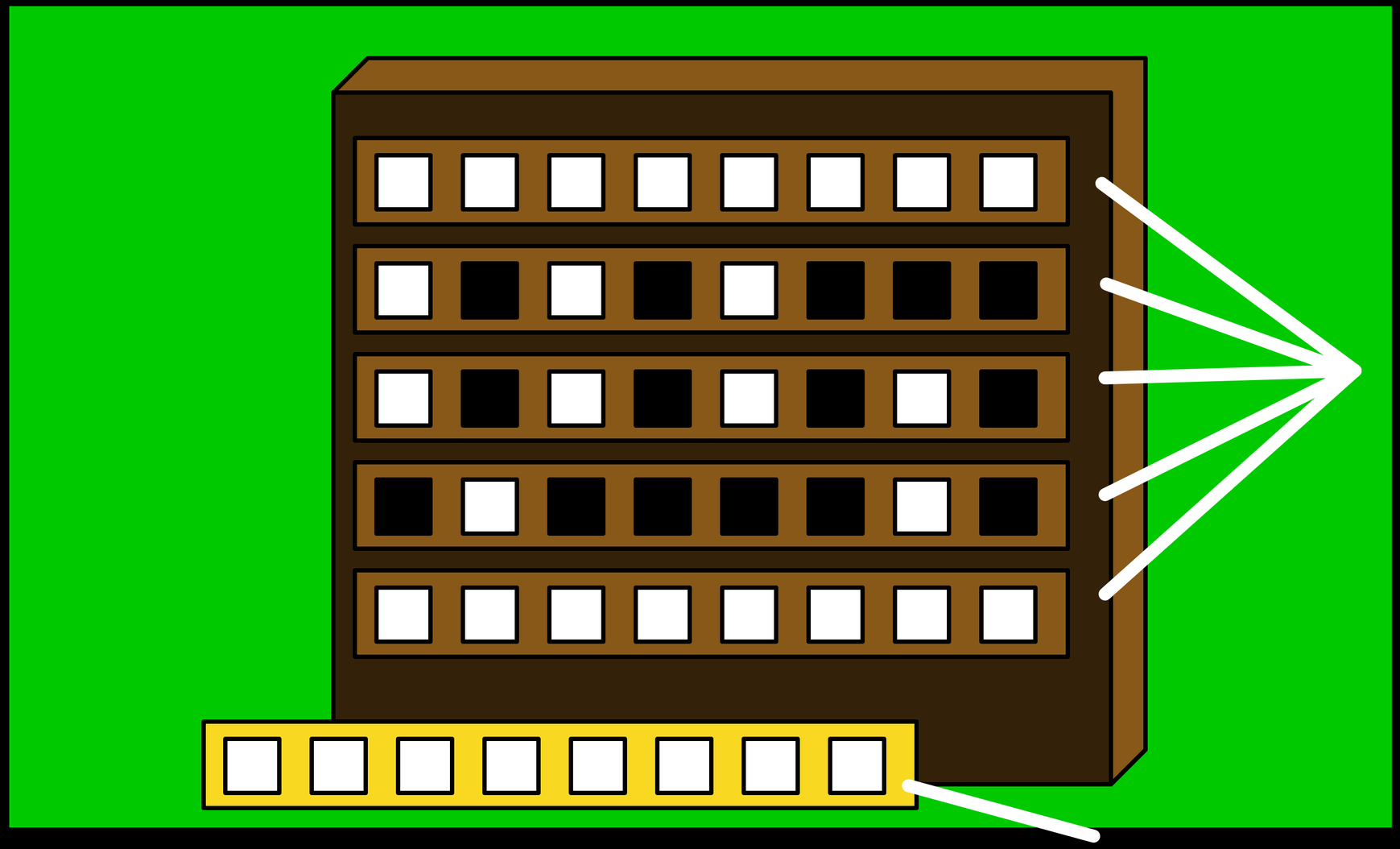
rowhammer attack



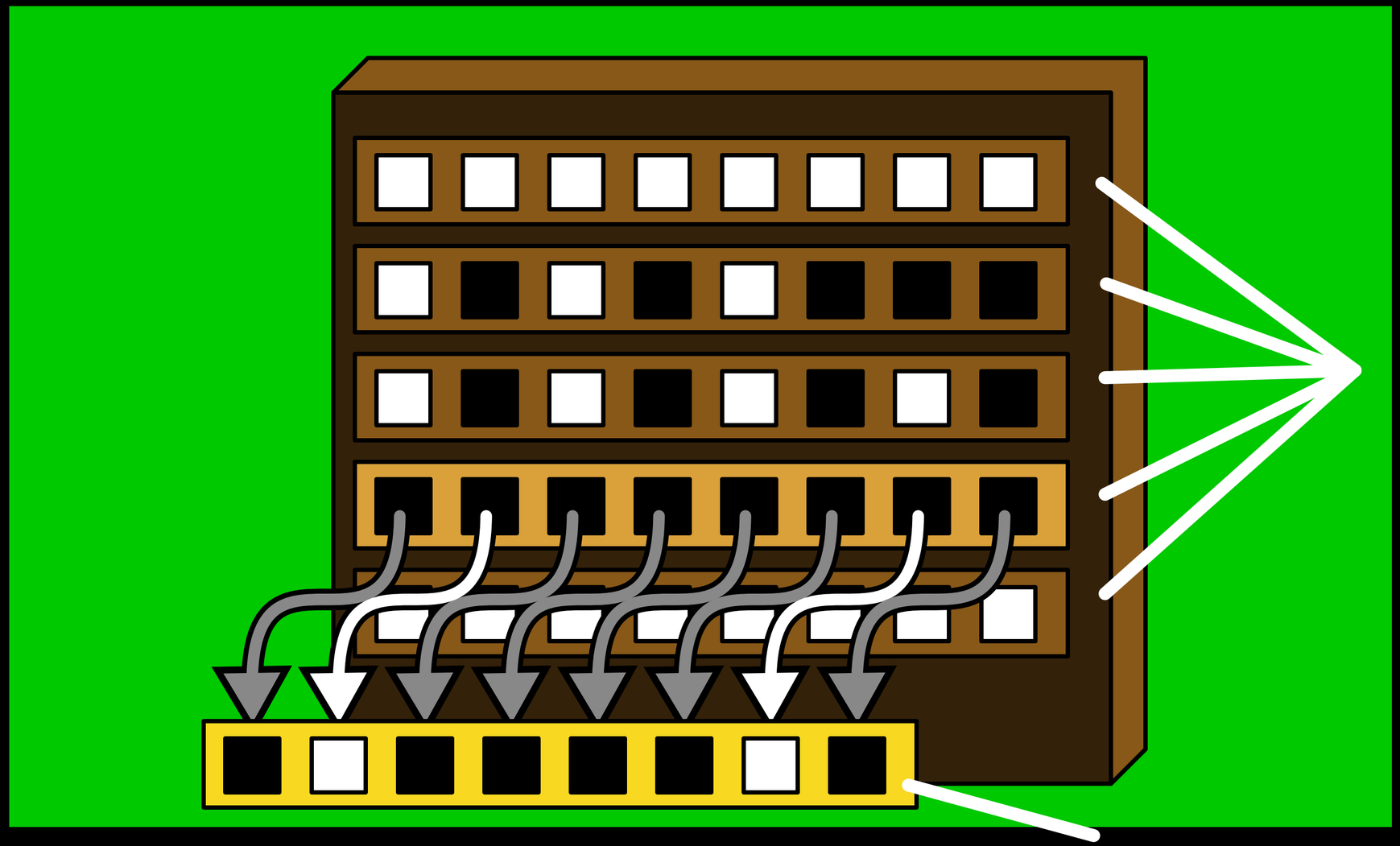
rowhammer attack



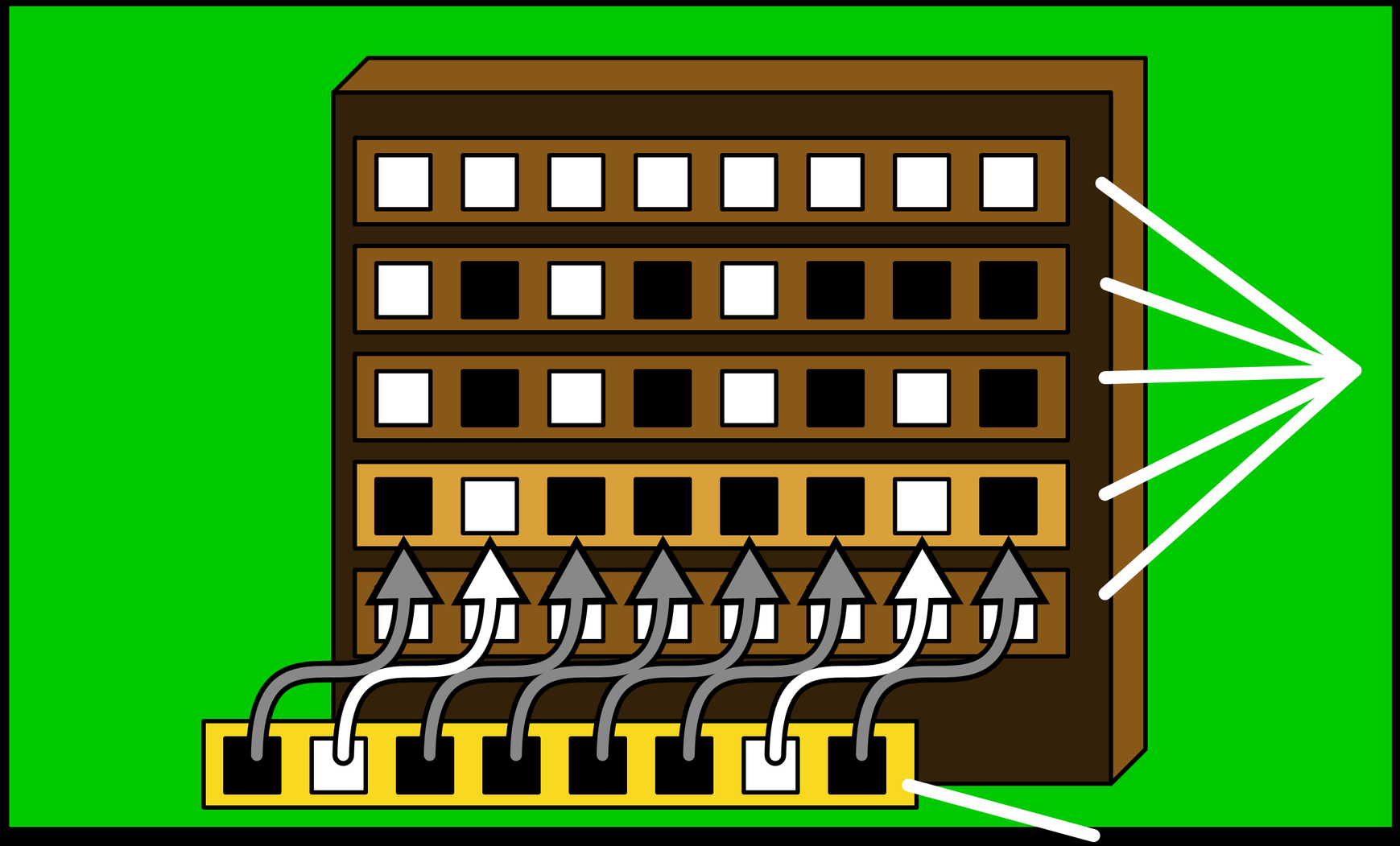
rowhammer attack



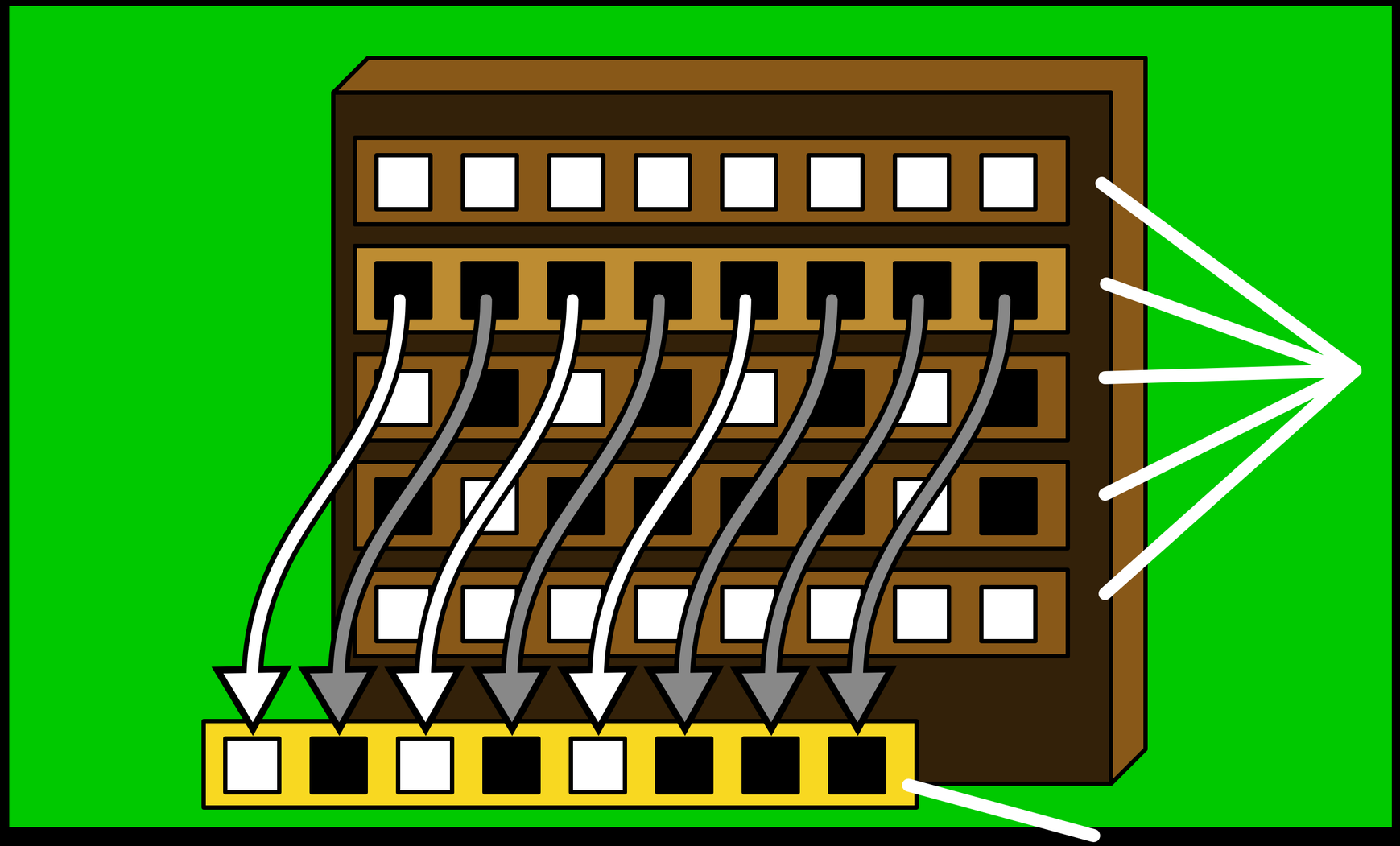
rowhammer attack



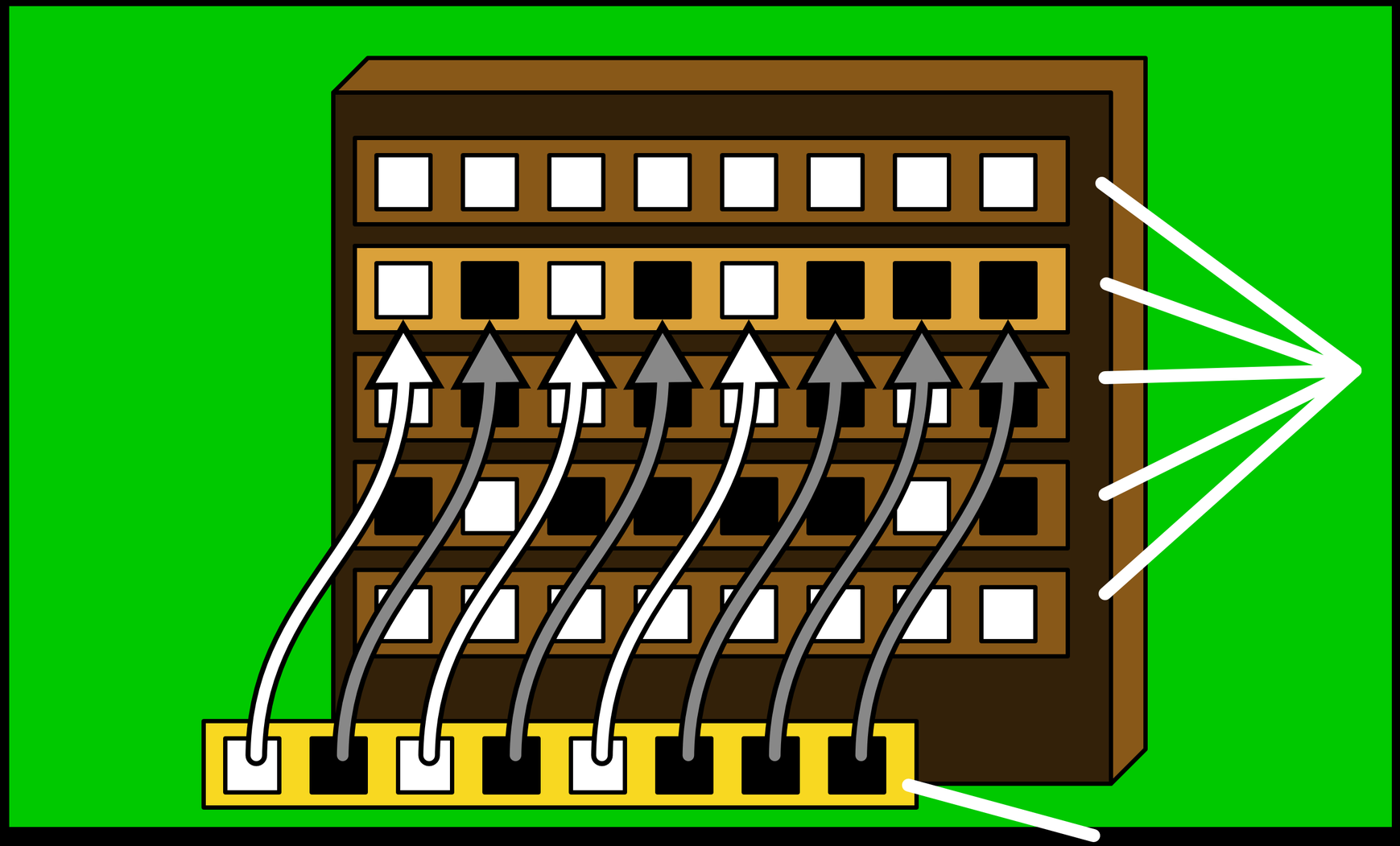
rowhammer attack



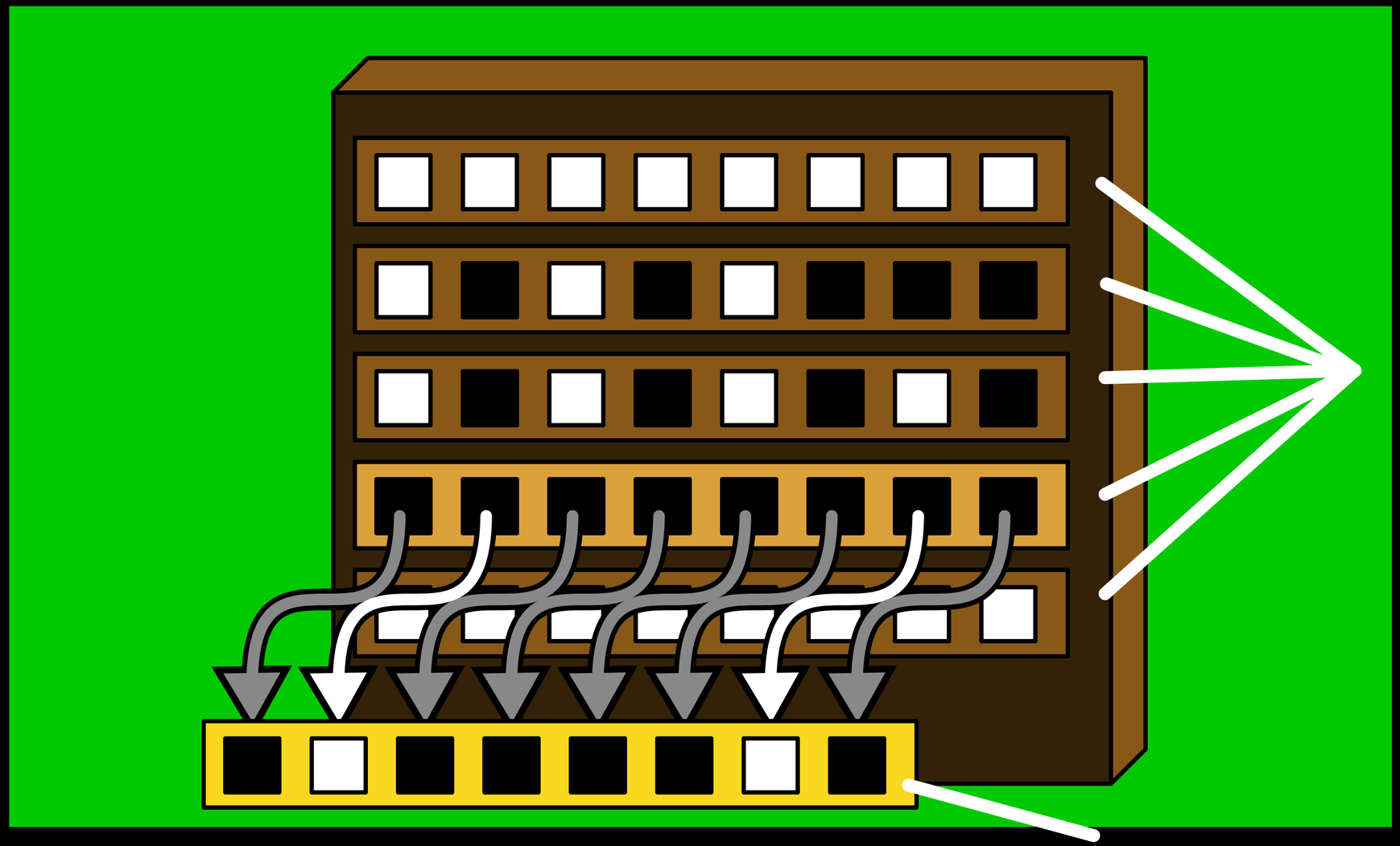
rowhammer attack



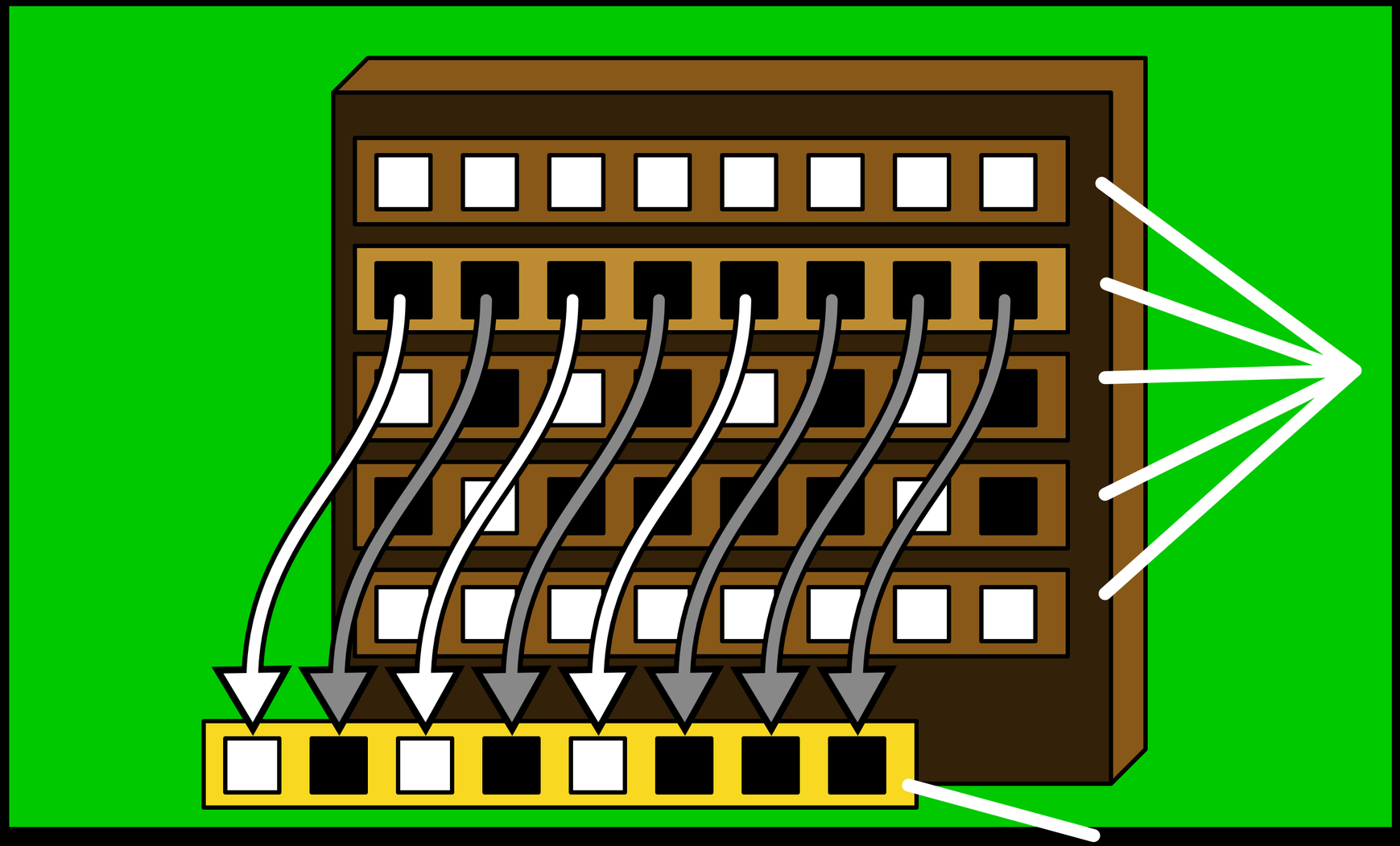
rowhammer attack



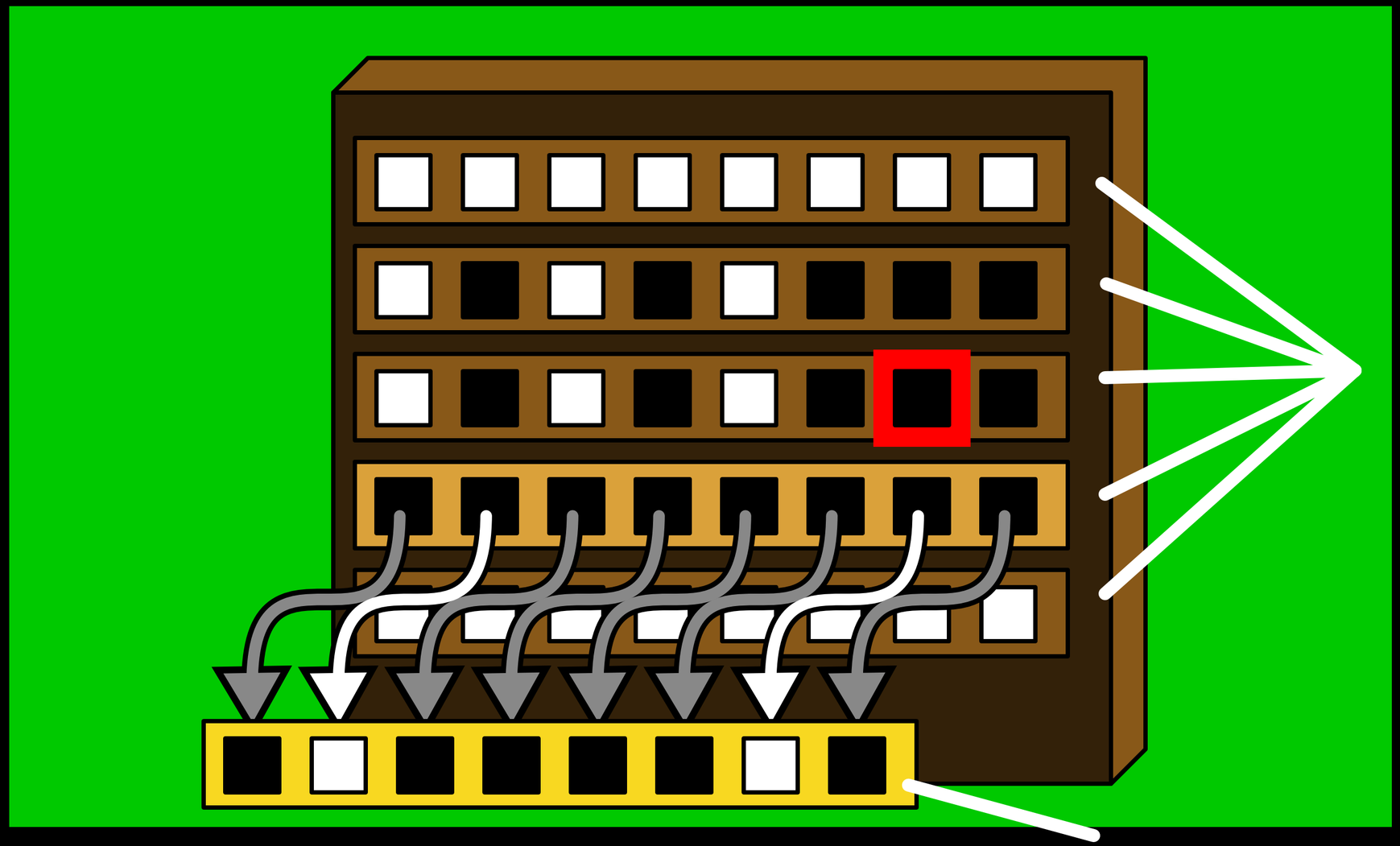
rowhammer attack



rowhammer attack

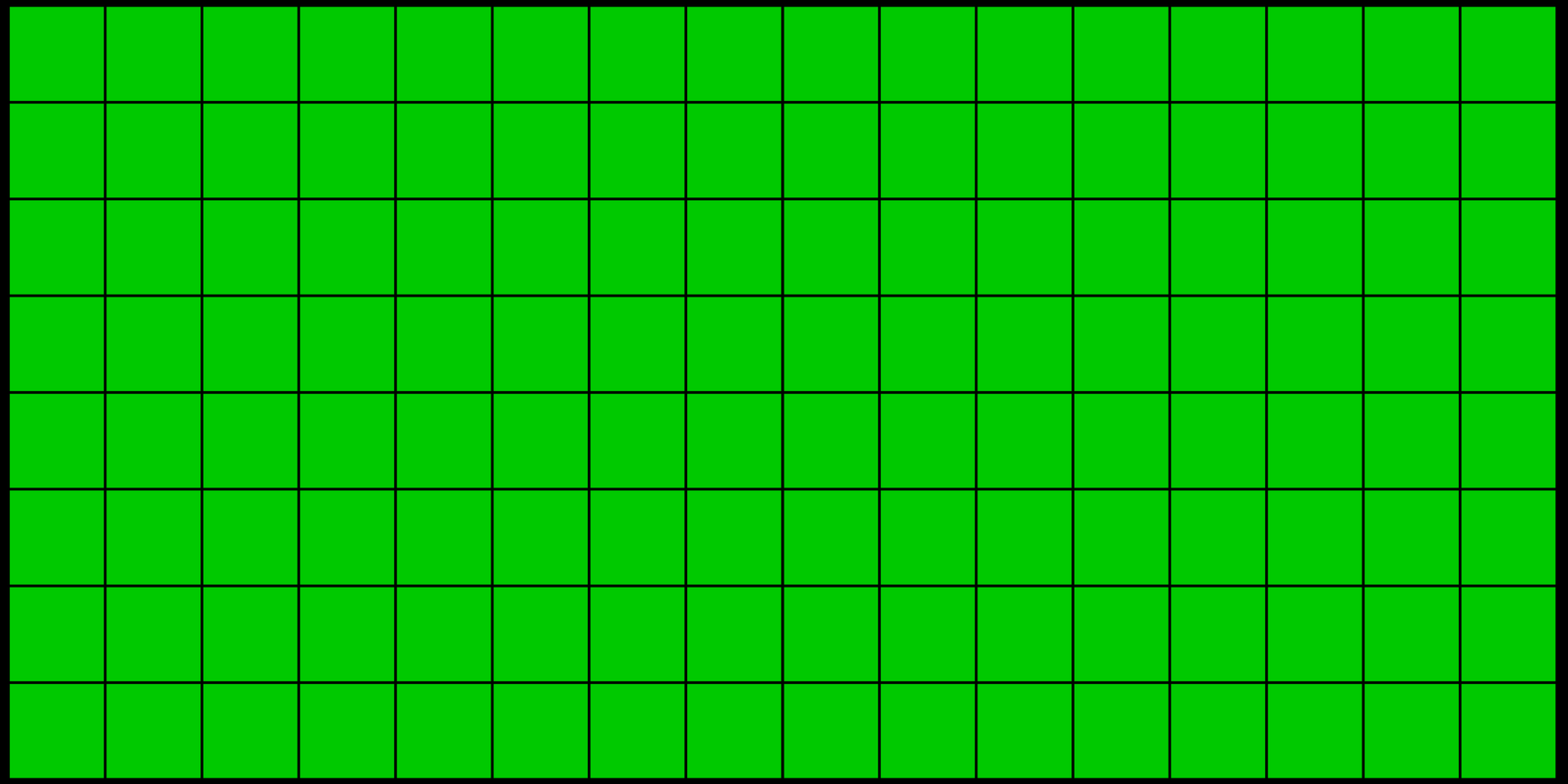


rowhammer attack



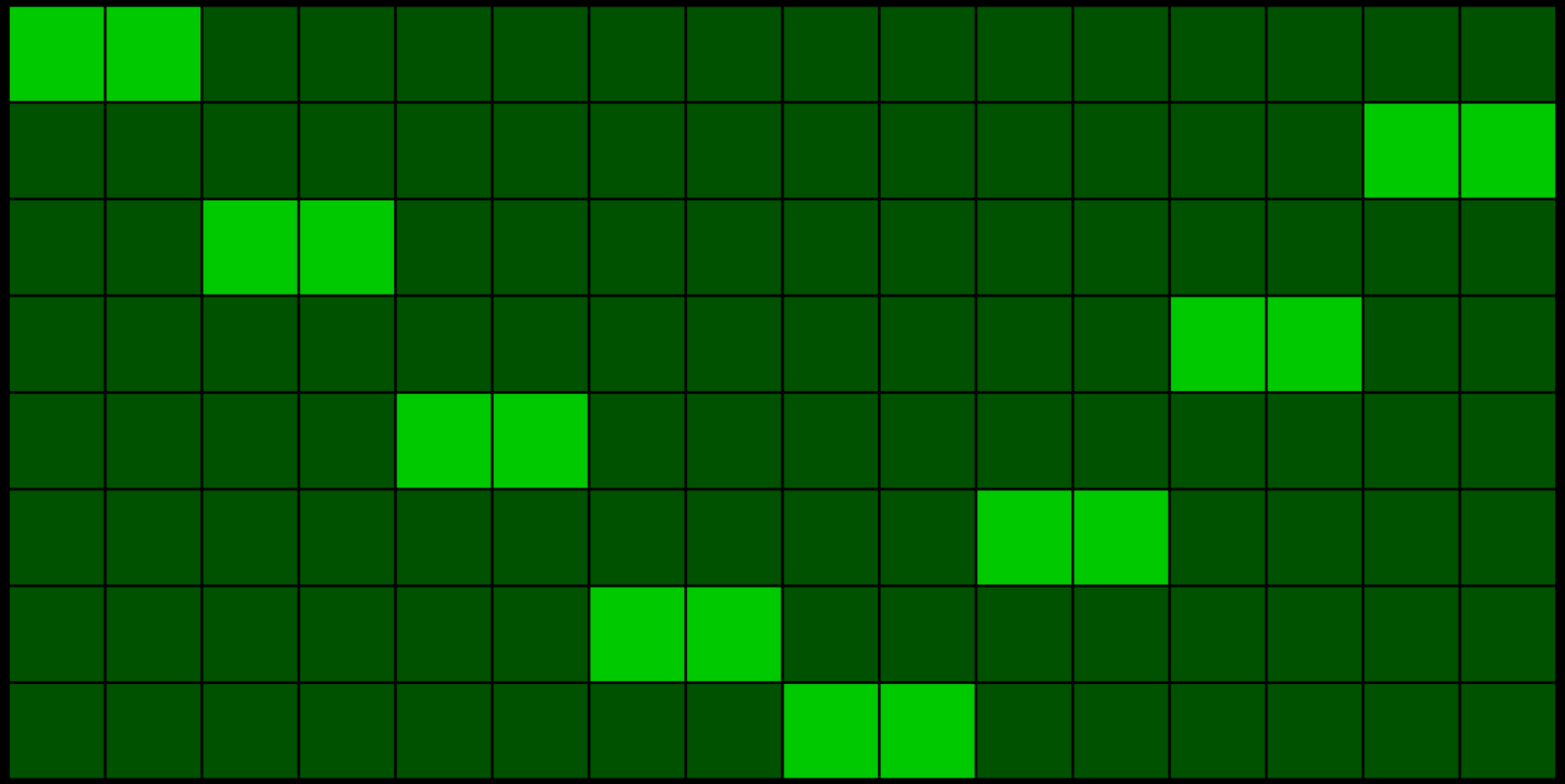
rowhammer

physical memory



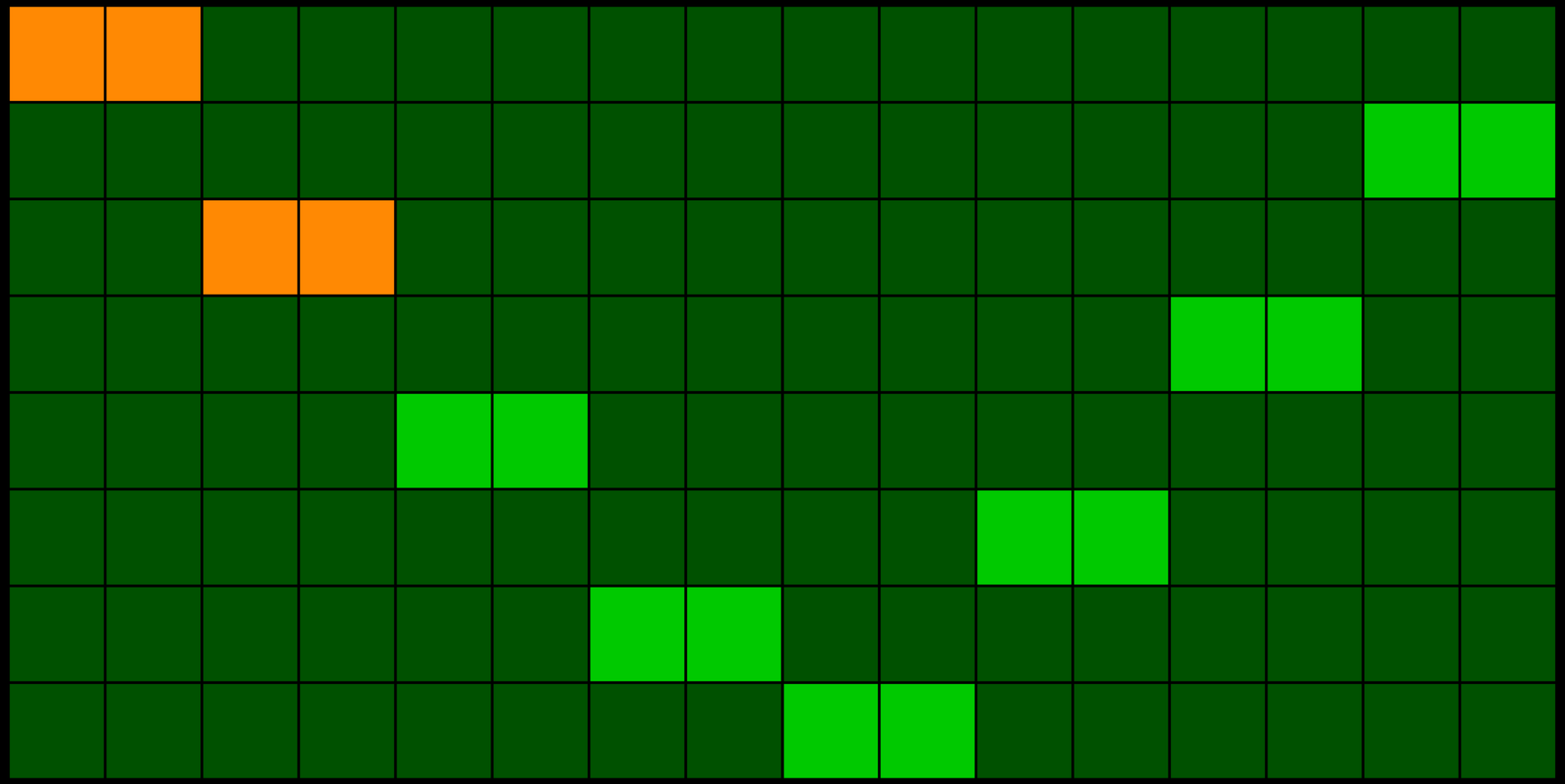
rowhammer

physical memory



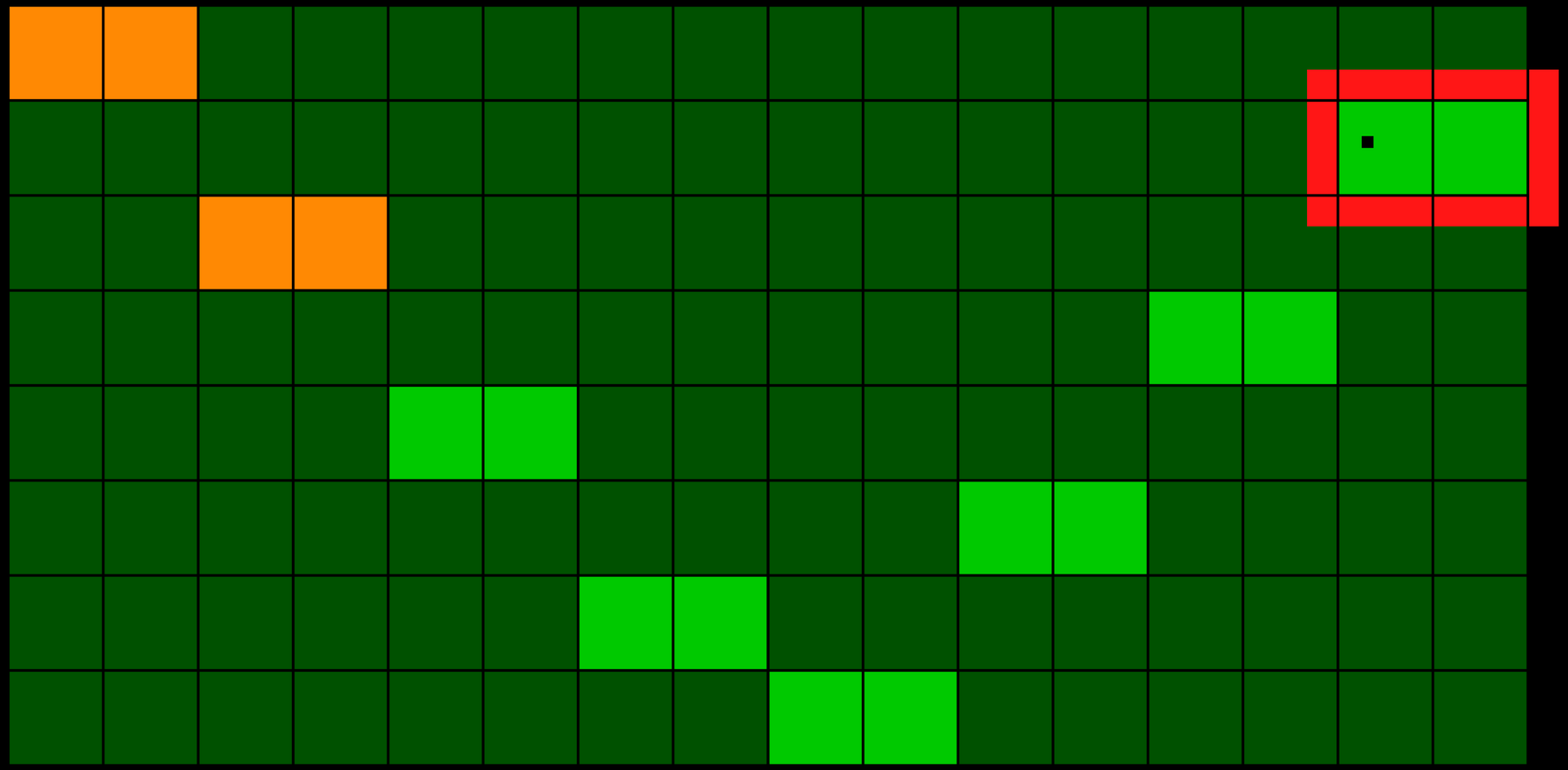
rowhammer

physical memory



rowhammer

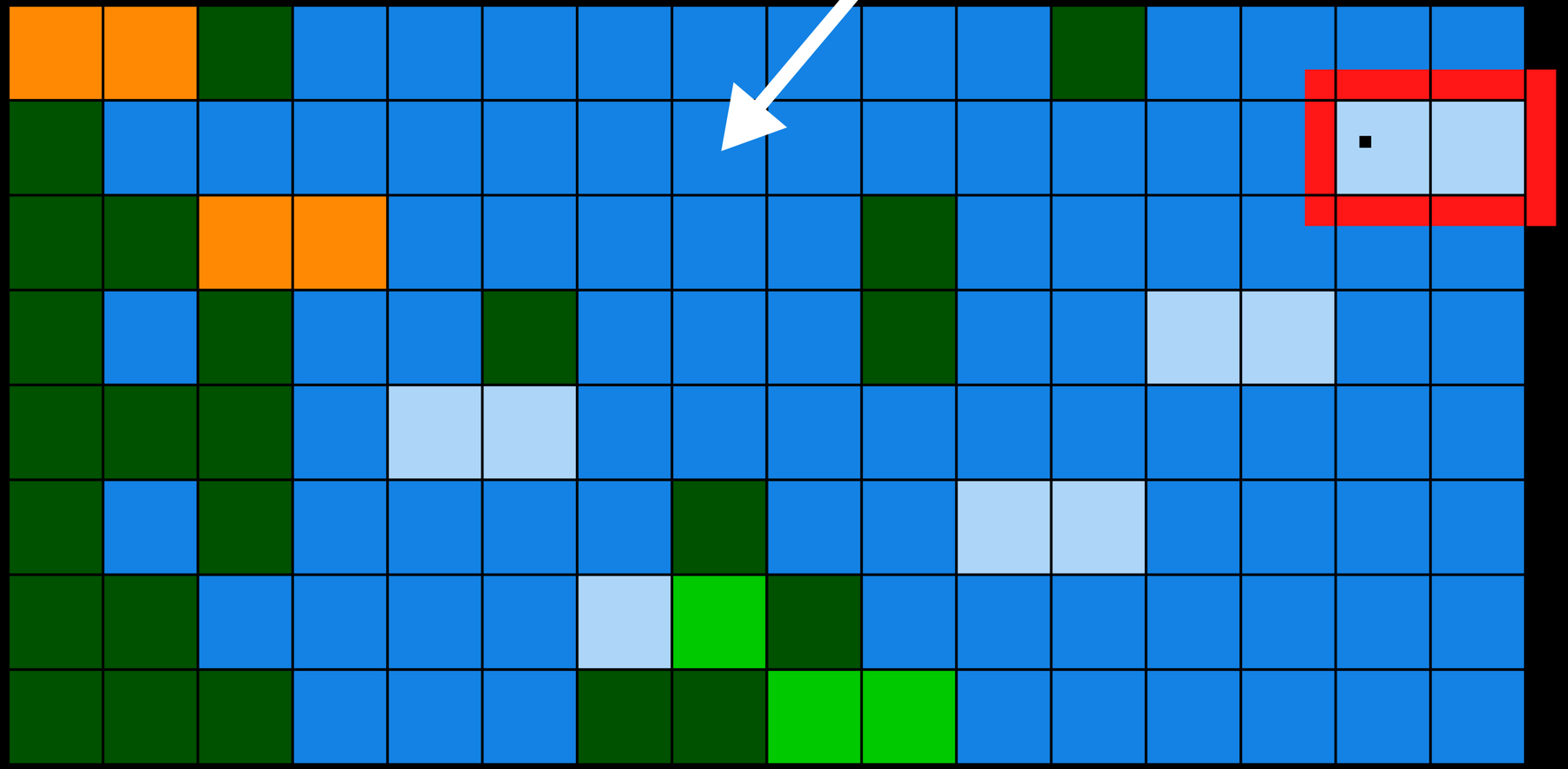
physical memory



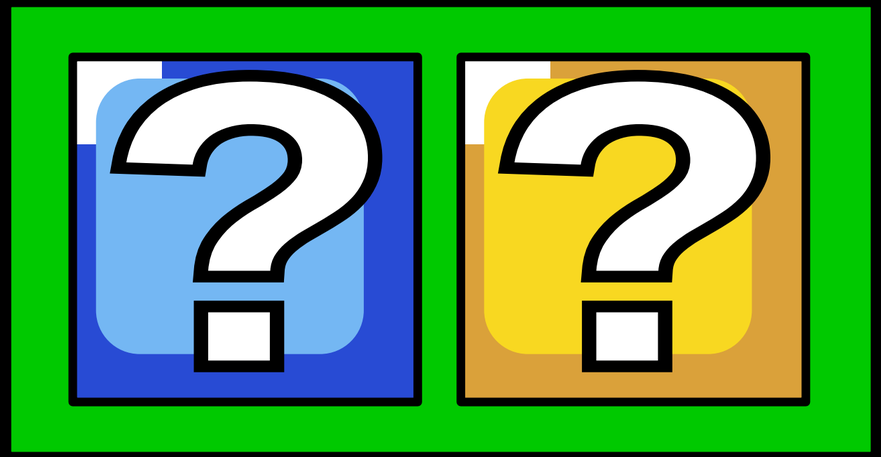
rowhammer (seaborn attack)

physical memory

sprayed page tables



fake Uint8Array object



Data in Chakra's mixed-type arrays

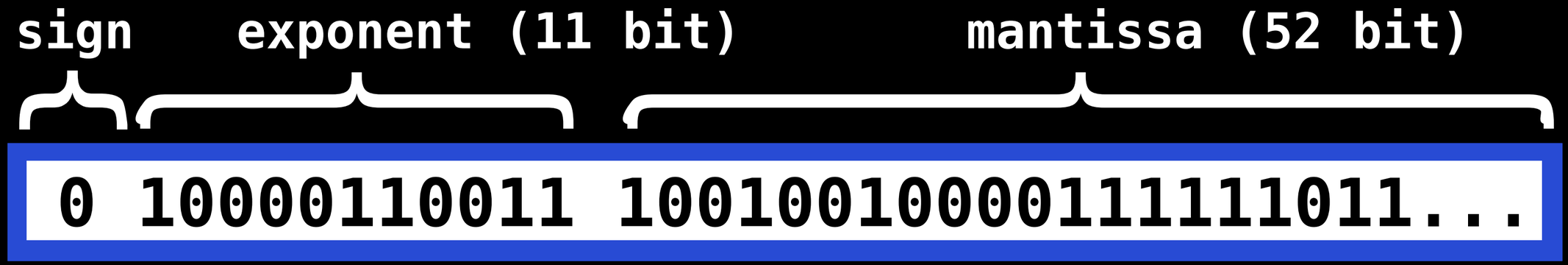
Data in Chakra's mixed-type arrays

Numbers (doubles)

```
0 10000110011 1001001000011111011...
```

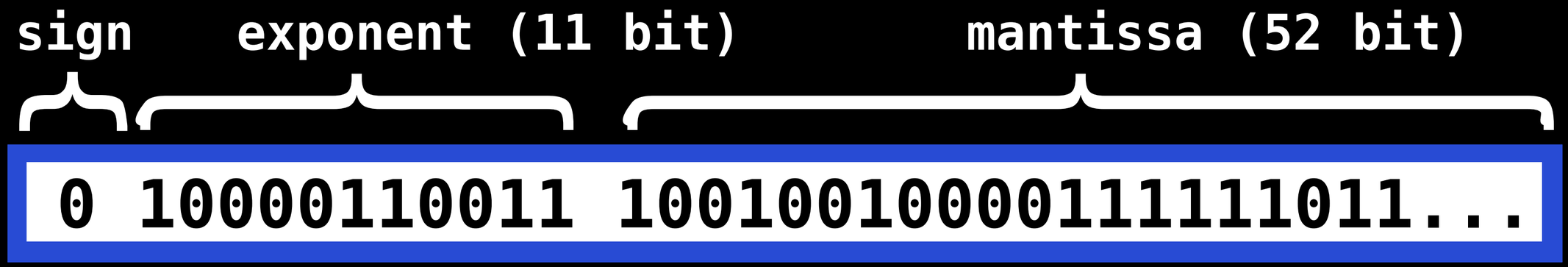
Data in Chakra's mixed-type arrays

Numbers (doubles)

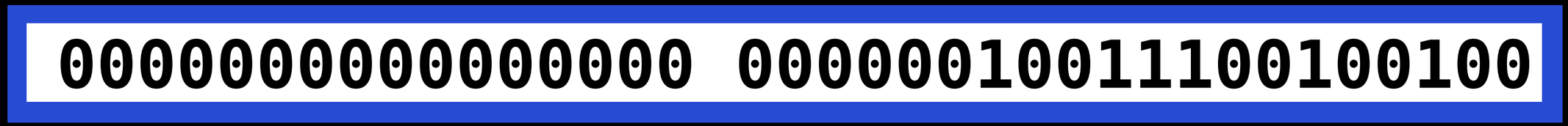


Data in Chakra's mixed-type arrays

Numbers (doubles)

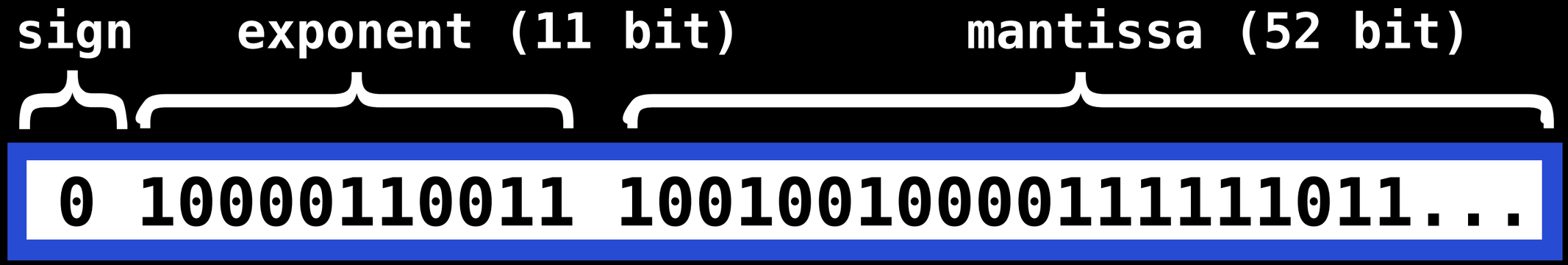


Objects (pointers)

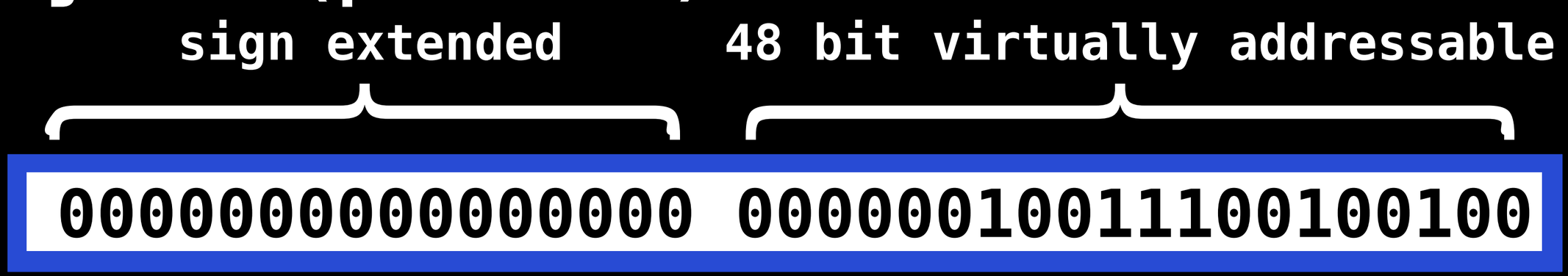


Data in Chakra's mixed-type arrays

Numbers (doubles)

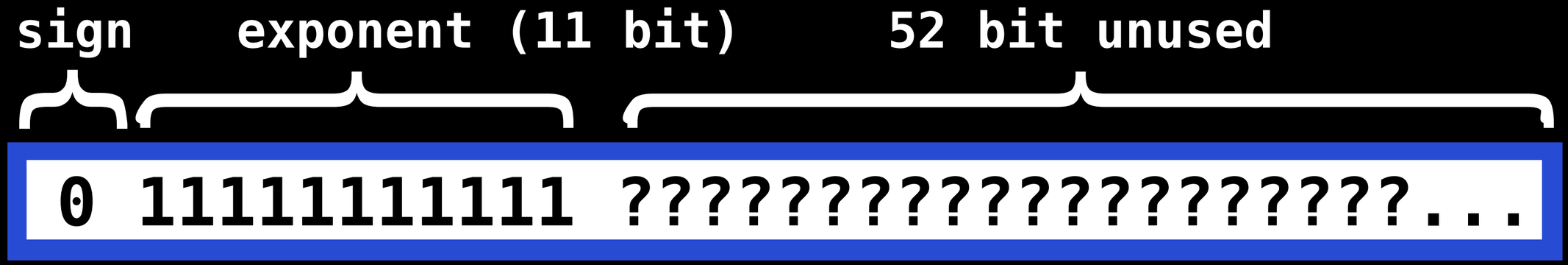


Objects (pointers)

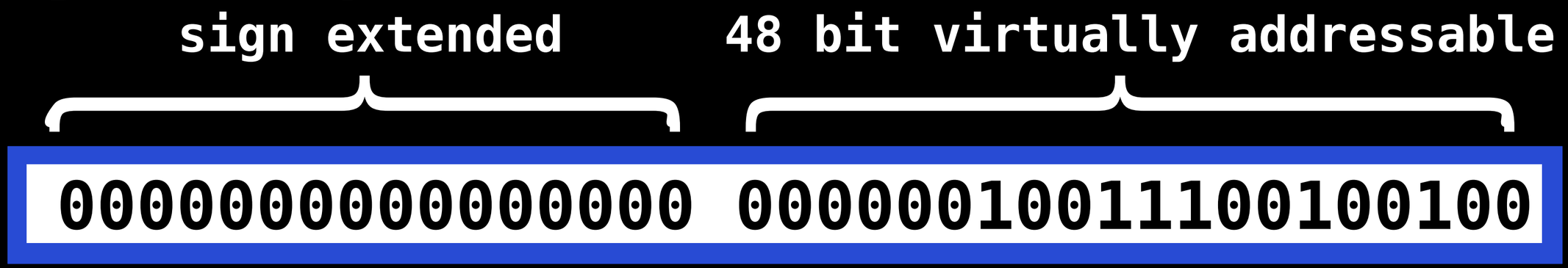


Data in Chakra's mixed-type arrays

Numbers, NaN

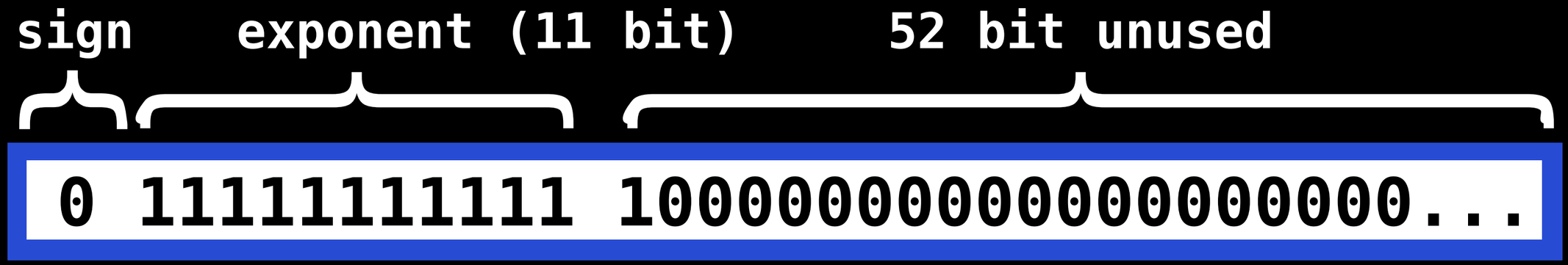


Objects (pointers)

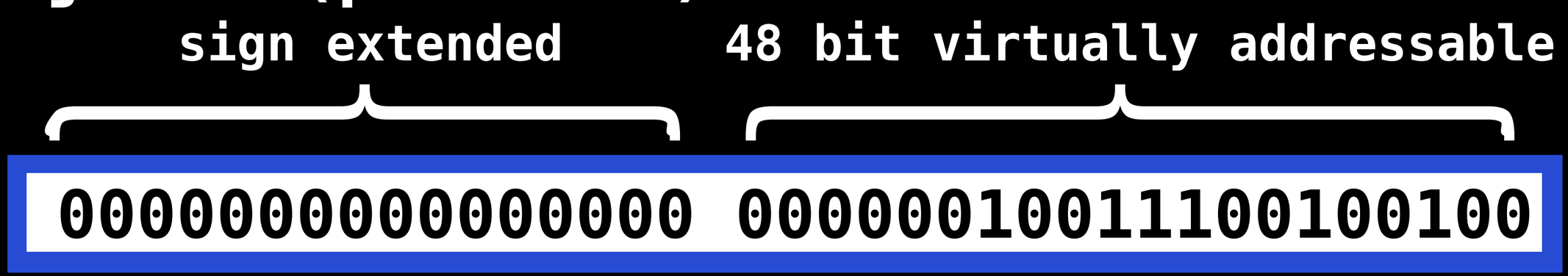


Data in Chakra's mixed-type arrays

Numbers, NaN

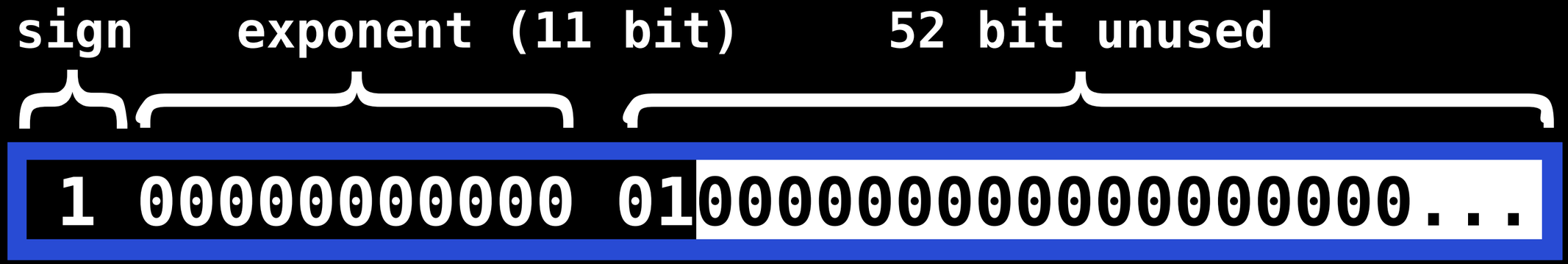


Objects (pointers)

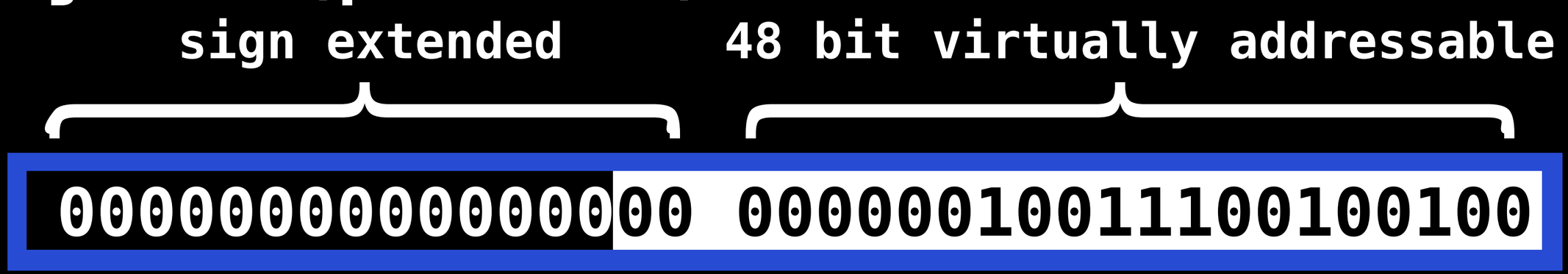


Data in Chakra's mixed-type arrays

Numbers, NaN, 14 high bits flipped

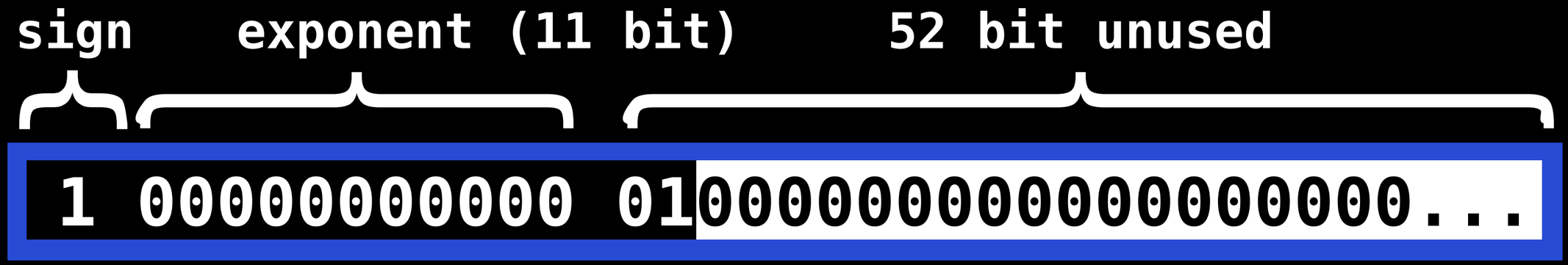


Objects (pointers)

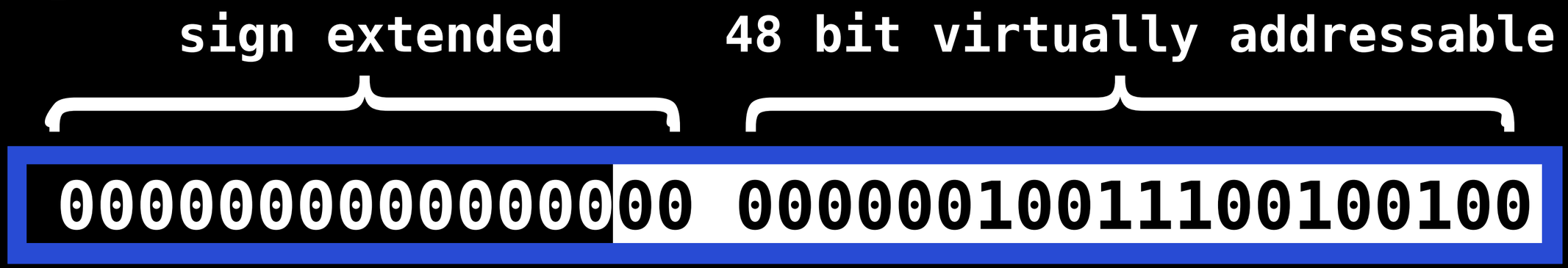


Data in Chakra's mixed-type arrays

Numbers, NaN, 14 high bits flipped

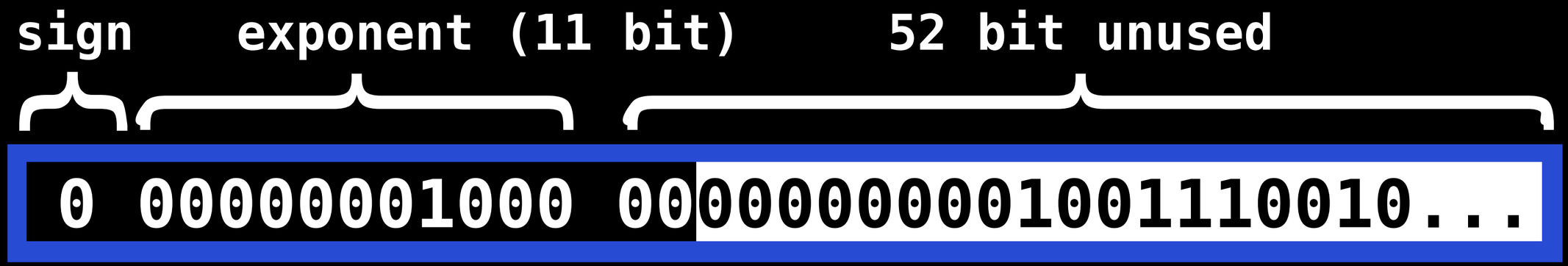


Objects, Non-canonical -NaN



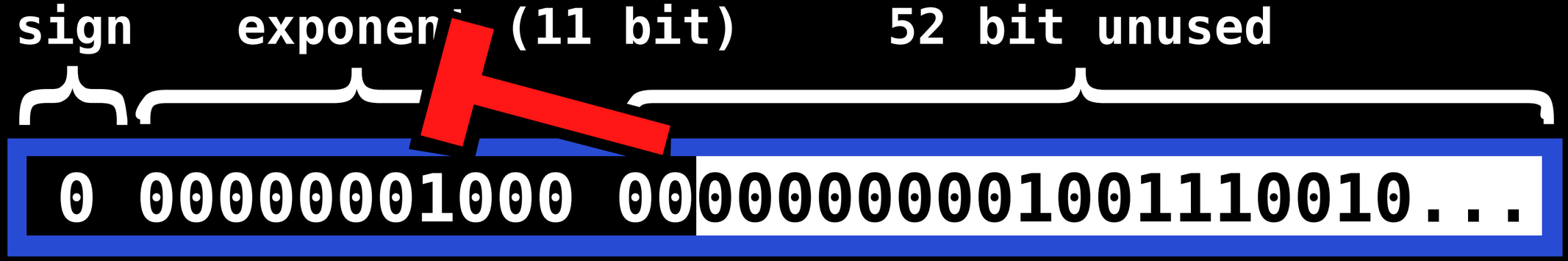
Type Flipping

Number, 1 bit flip away from a pointer

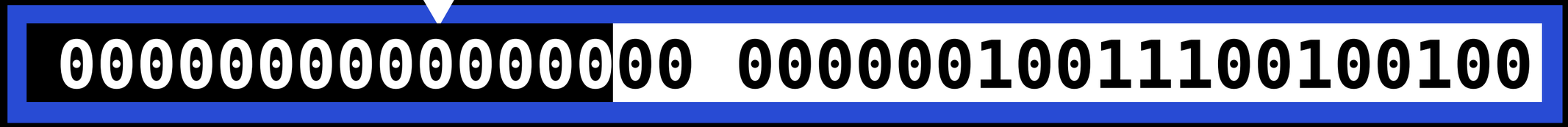


Type Flipping

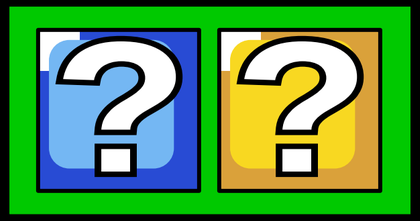
Number, 1 bit flip away from a pointer



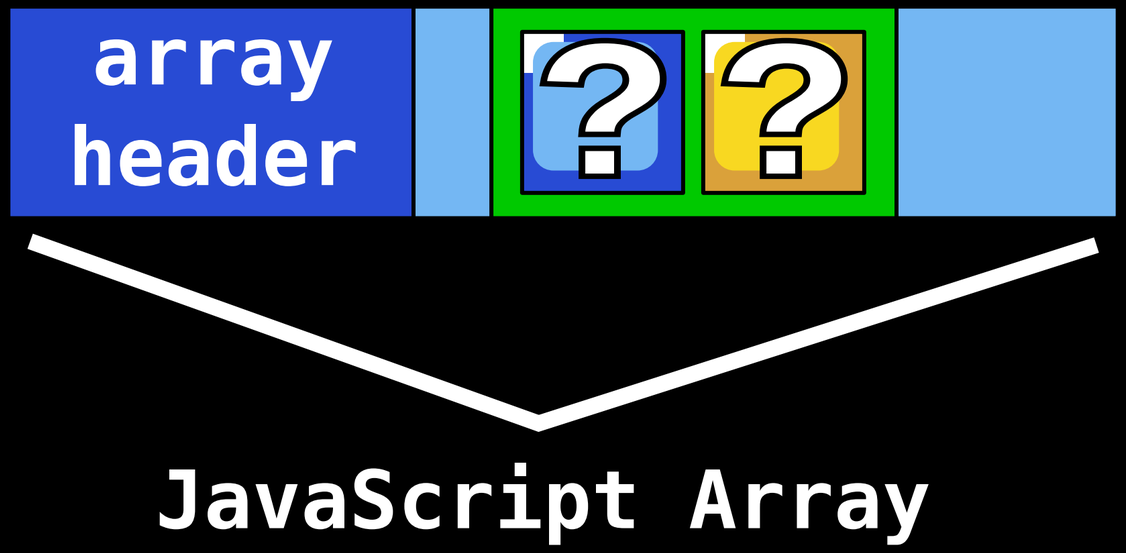
Fake Object reference



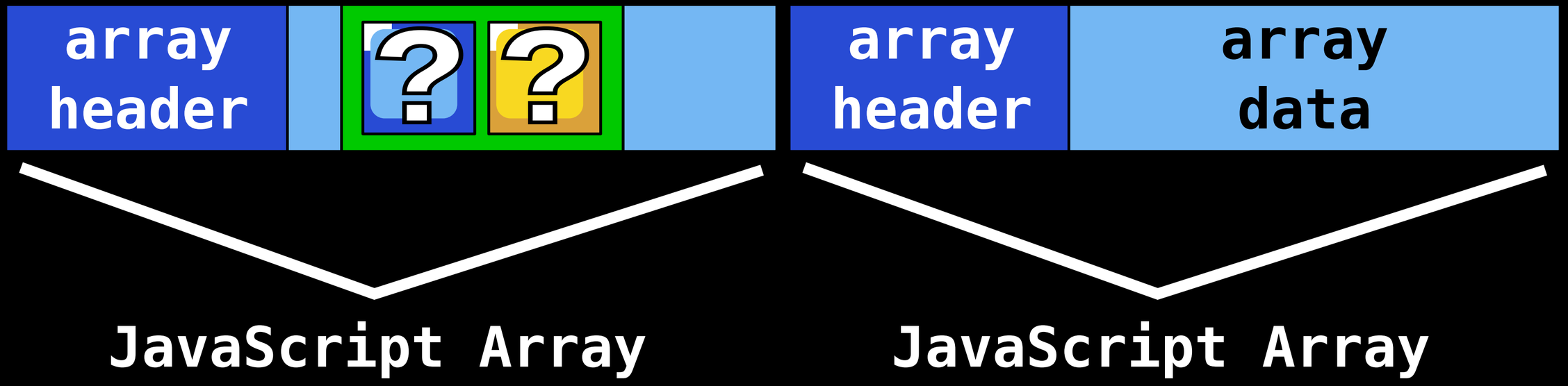
pointer pivotting



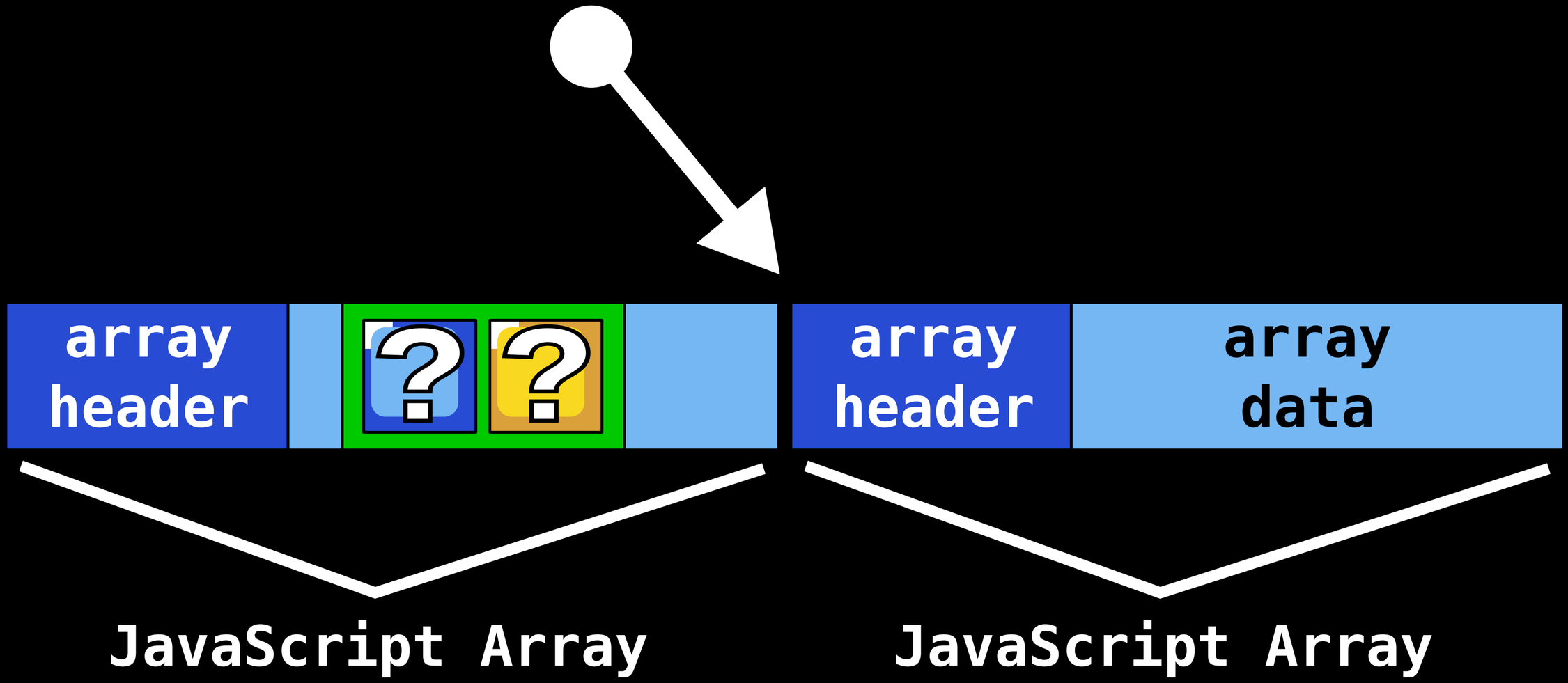
pointer pivotting



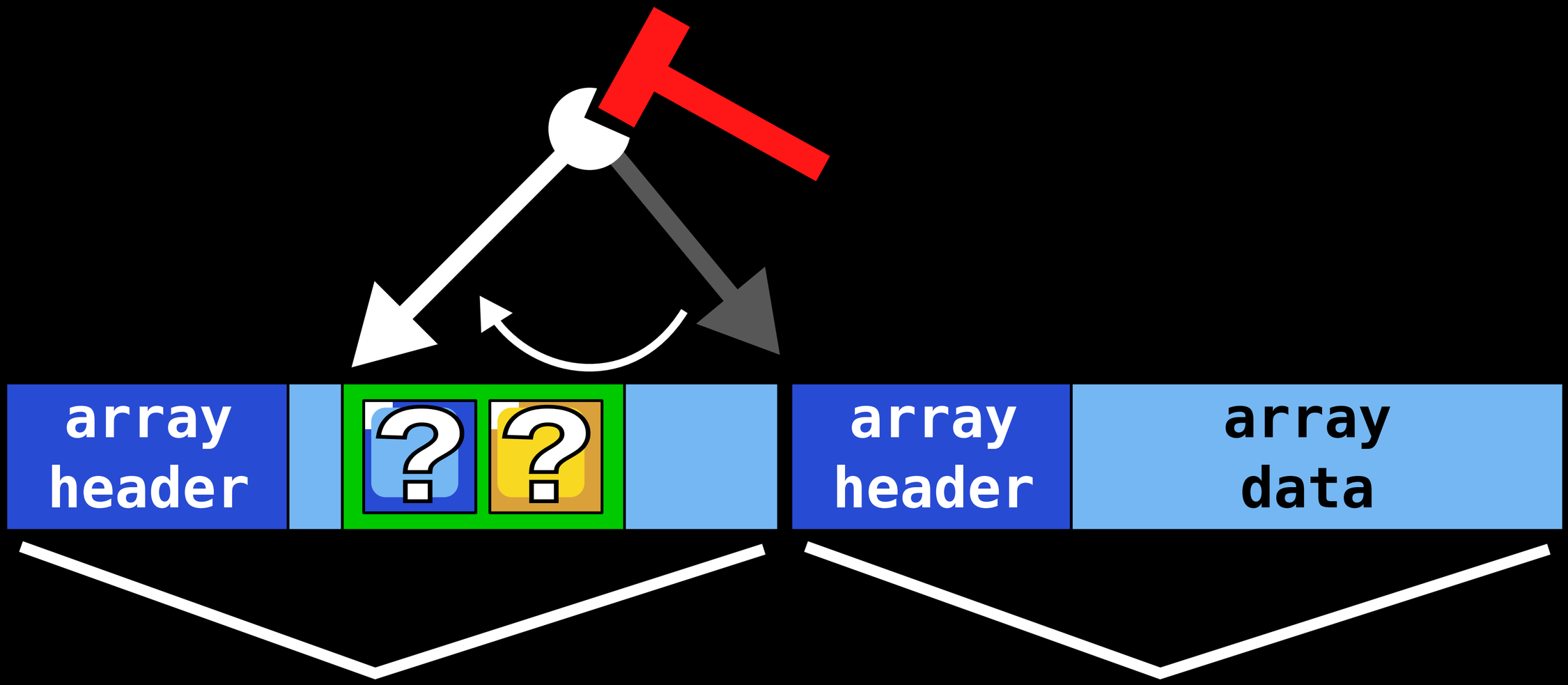
pointer pivotting



pointer pivotting



pointer pivotting



Probability of a useful 1 to 0 flips

Type flipping:	11 / 64 bits
Pointer pivotting:	12 / 64 bits
Total:	23 / 64 bits

Dedup mitigation

- Disable memory deduplication

```
> Disable-MMAgent -PageCombining
```

- We've reported this issue to Microsoft and they have addressed this issue in ms-16-093, July 18th (CVE-2016-3272) by disabling dedup.

Part II: Flip Feng Shui

Part II: Flip Feng Shui

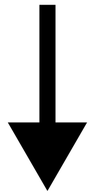
Rowhammer
(hardware bug)

Part II: Flip Feng Shui

Rowhammer
(hardware bug)
+
Deduplication
(more than a software side-channel)

Part II: Flip Feng Shui

Rowhammer
(hardware bug)
+
Deduplication
(more than a software side-channel)



Cross-VM compromise

Rowhammer bit flips:

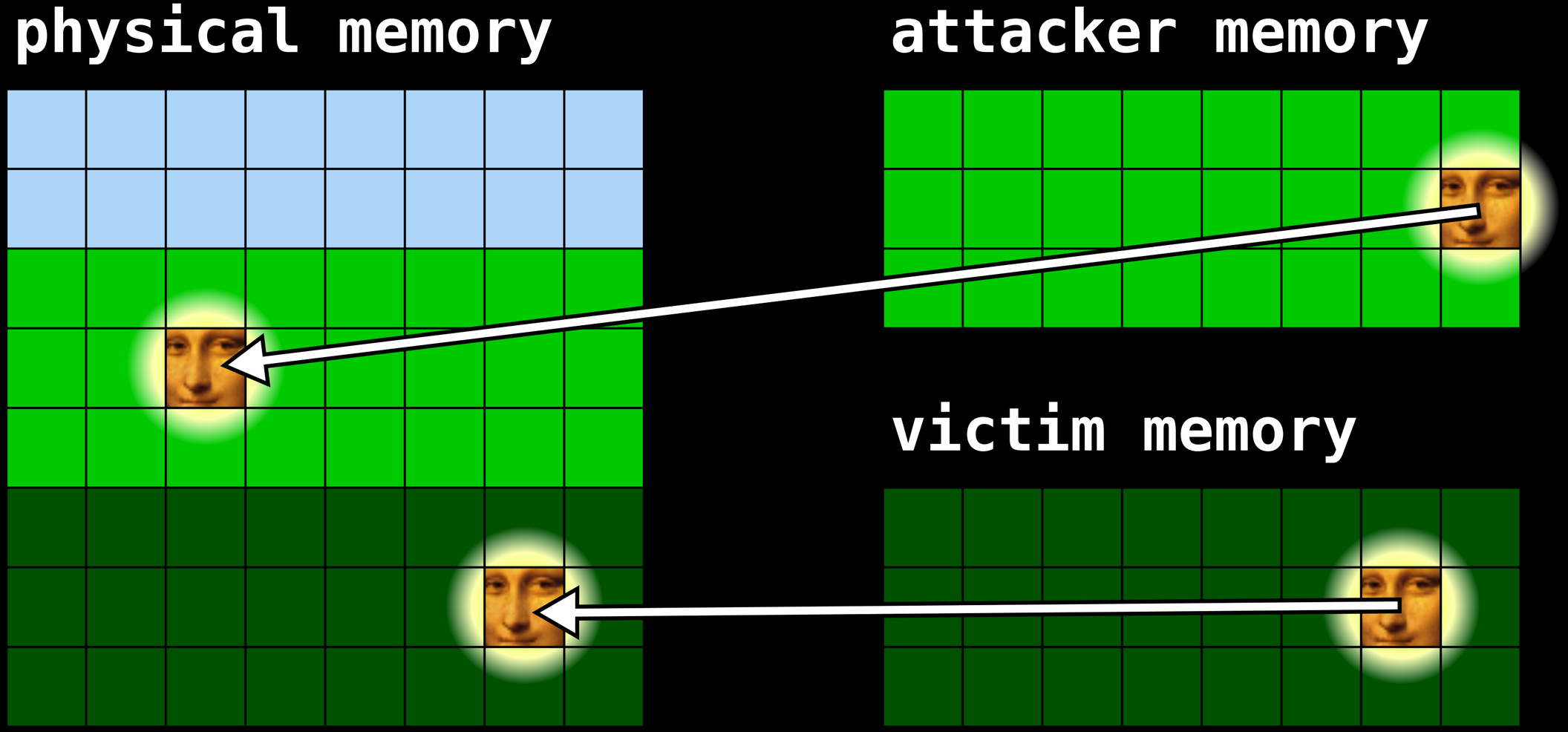
1) Unpredictable on which (virtual) page
2) Unpredictable where in the page
3) Repeatable once you've found a flip

Flip Feng Shui goal:

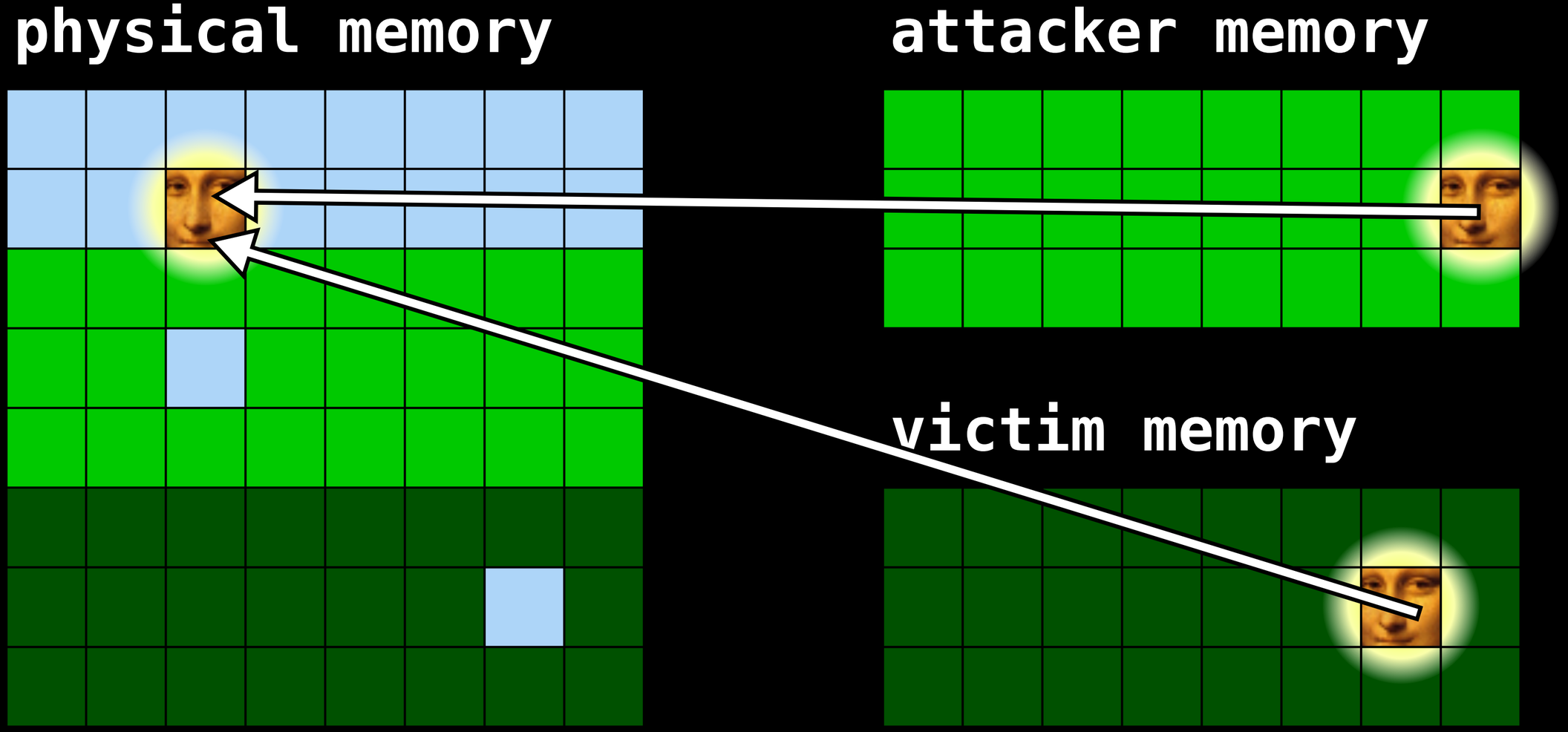
> Find victim pages with known content which allow for exploitation when certain bits are flipped

> Land this victim page in a physical memory location where this bit is flippable

Deduplication implementation: Windows 10

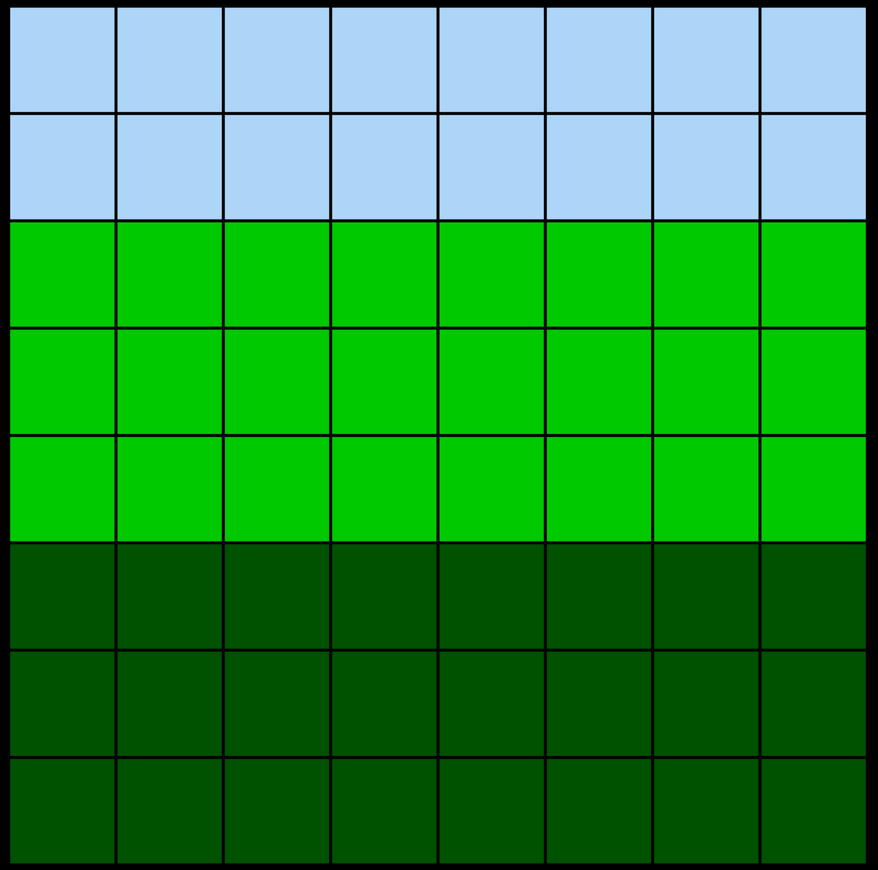


Deduplication implementation: Windows 10

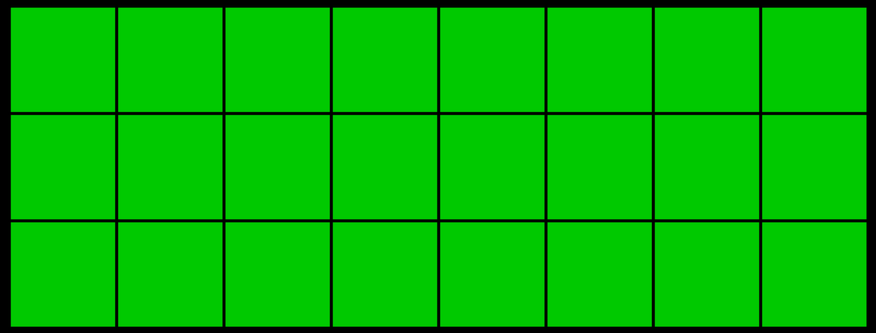


Deduplication implementation: KVM on Linux (KSM)

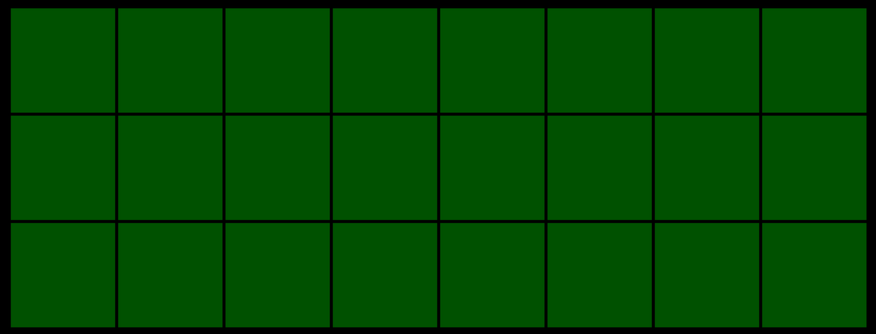
physical memory



attacker memory

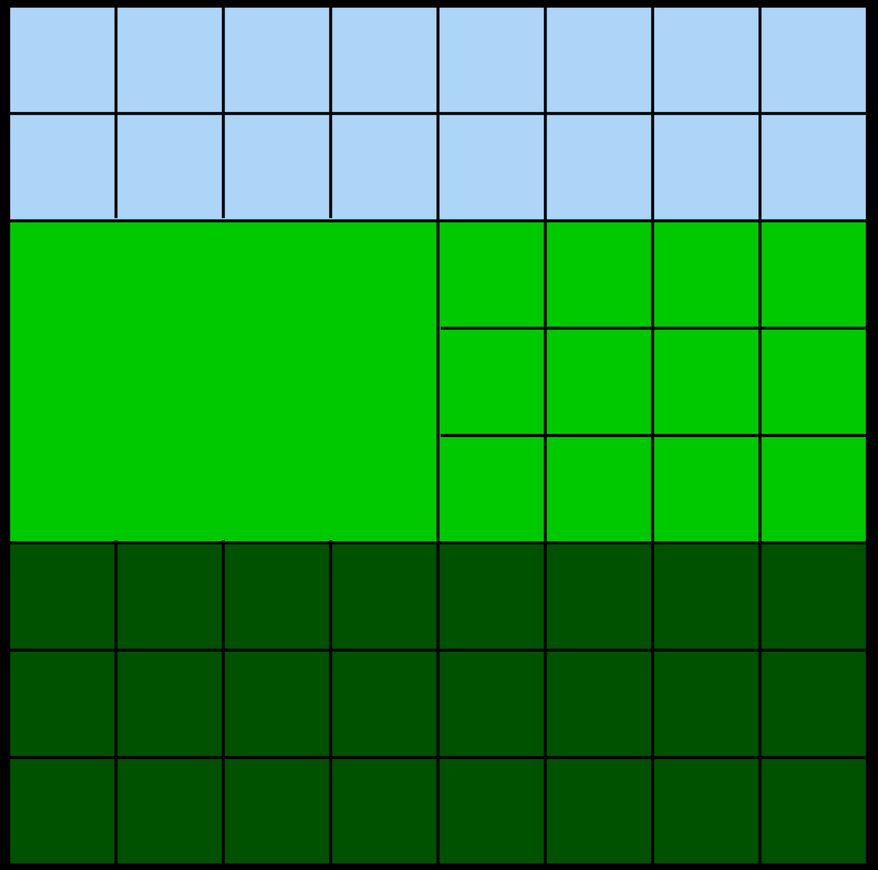


victim memory

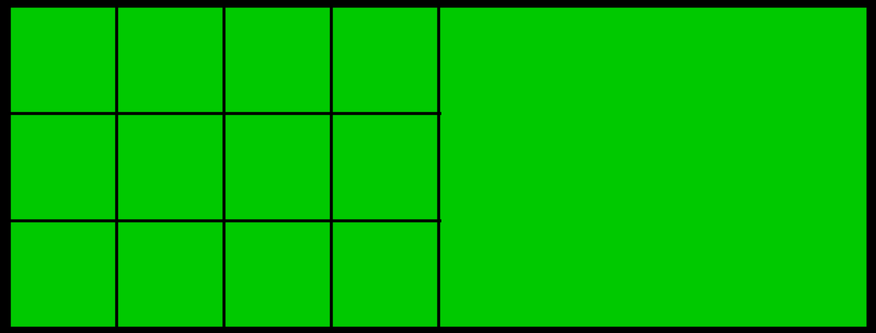


Deduplication implementation: KVM on Linux (KSM)

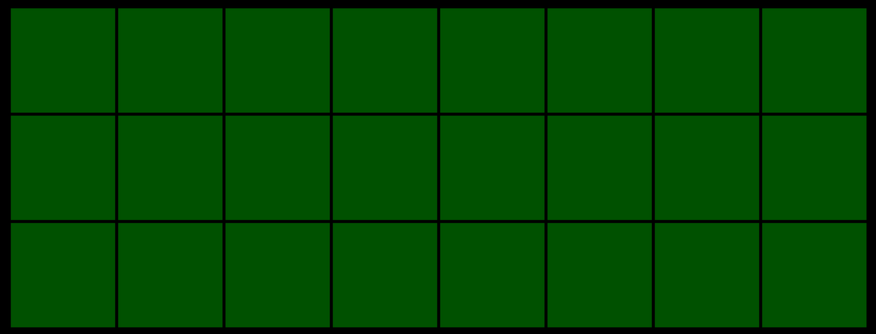
physical memory



attacker memory

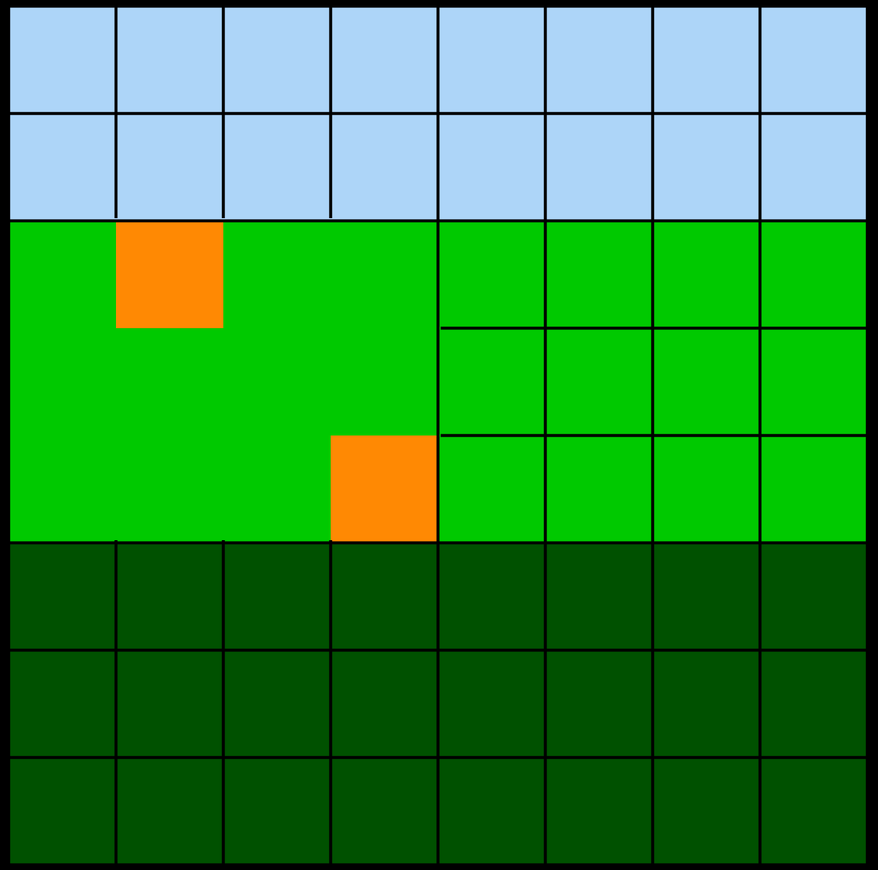


victim memory

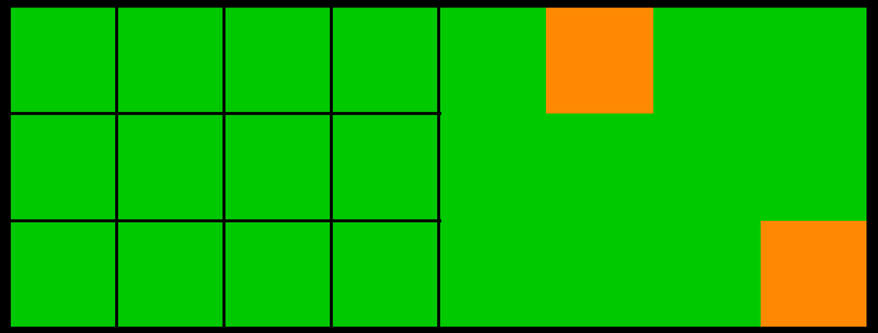


Deduplication implementation: KVM on Linux (KSM)

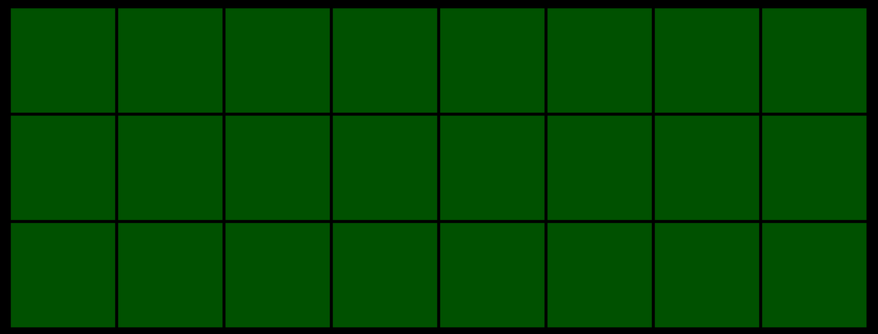
physical memory



attacker memory

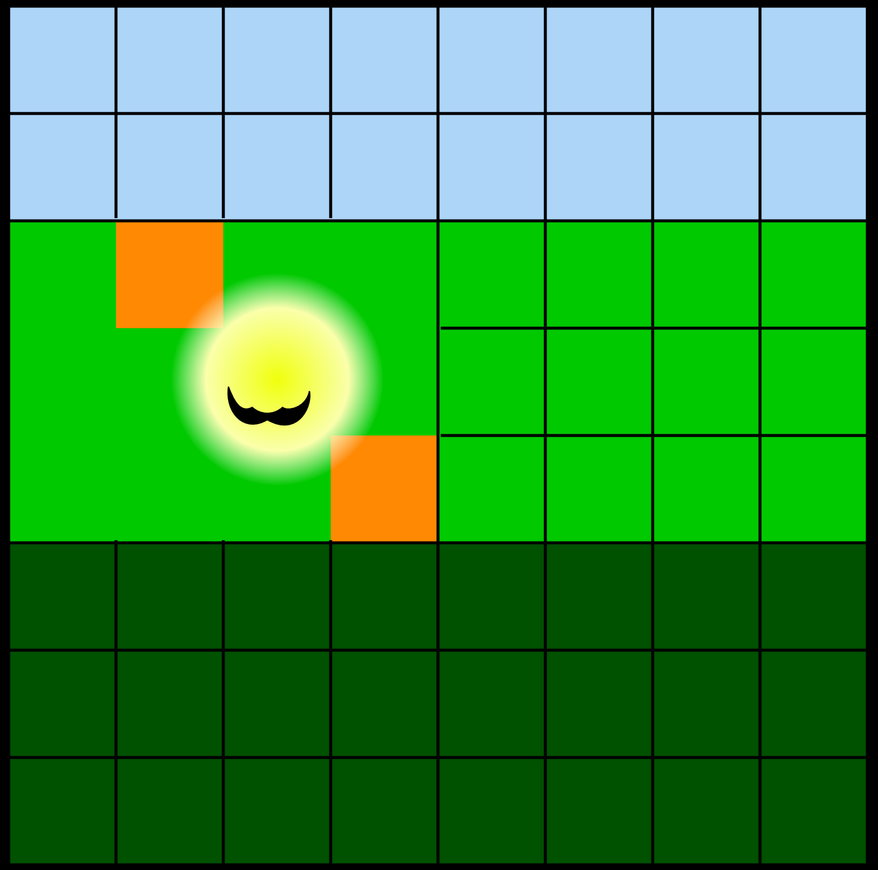


victim memory

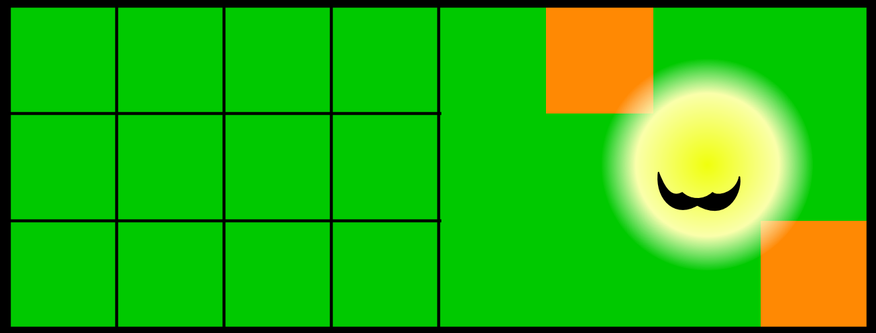


Deduplication implementation: KVM on Linux (KSM)

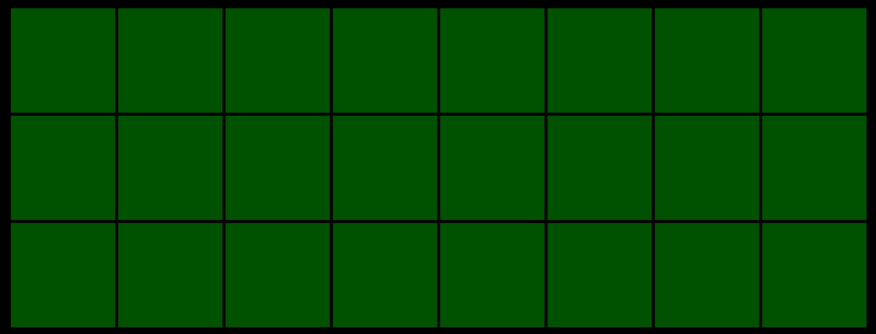
physical memory



attacker memory

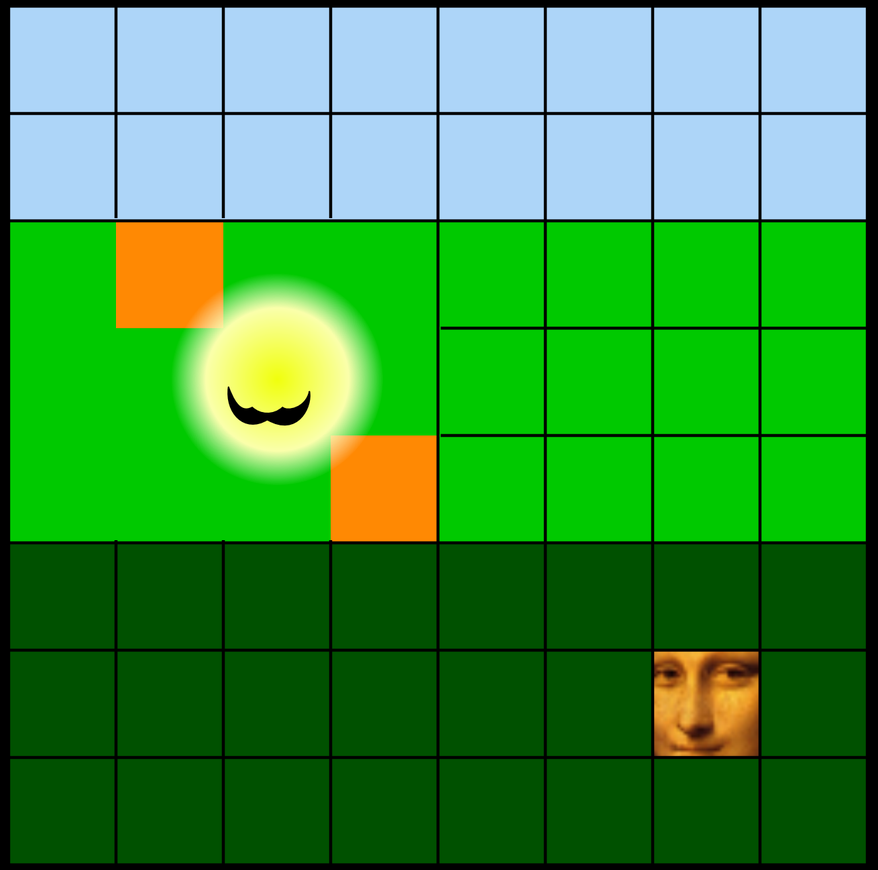


victim memory

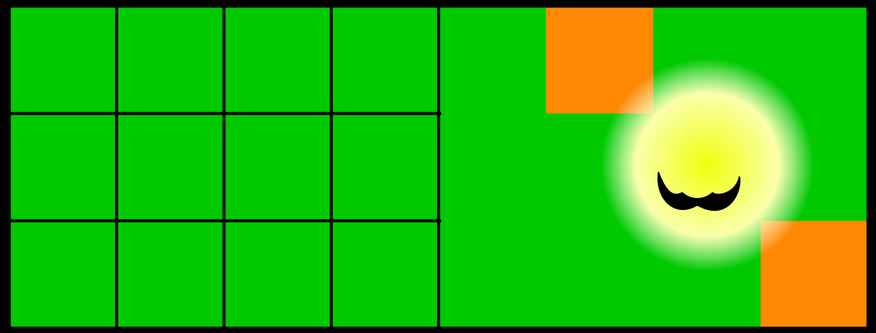


Deduplication implementation: KVM on Linux (KSM)

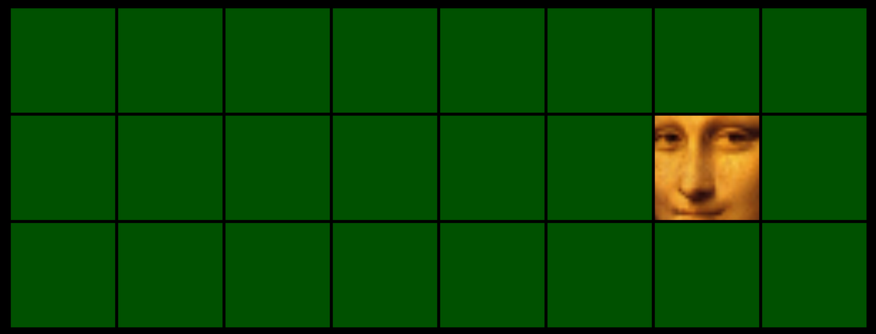
physical memory



attacker memory

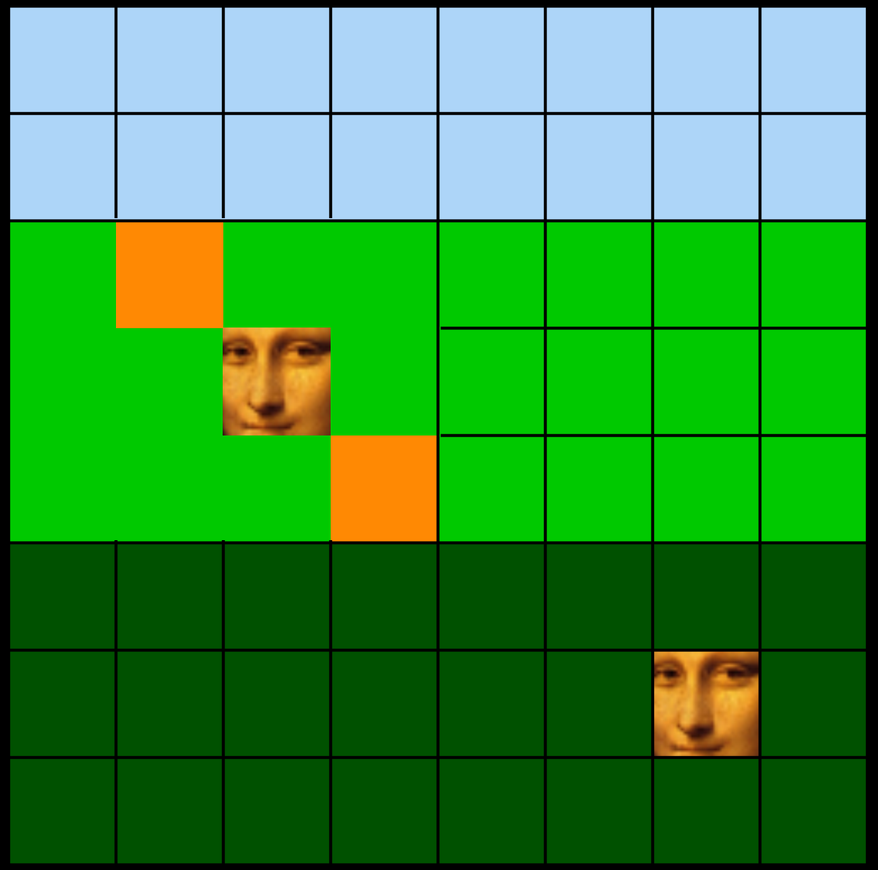


victim memory

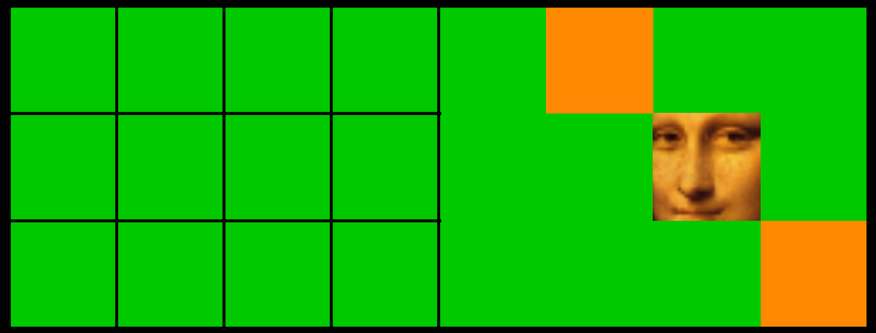


Deduplication implementation: KVM on Linux (KSM)

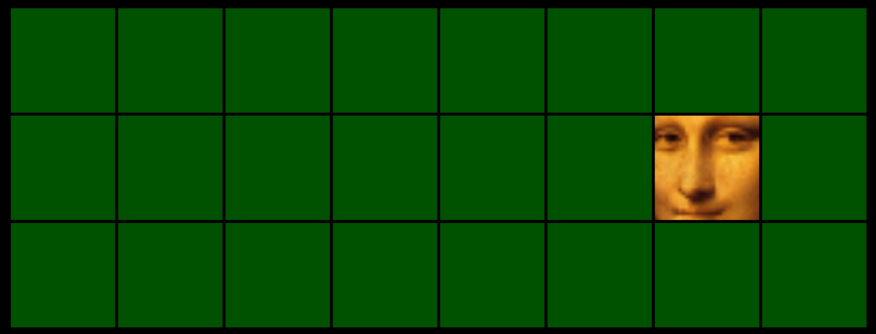
physical memory



attacker memory

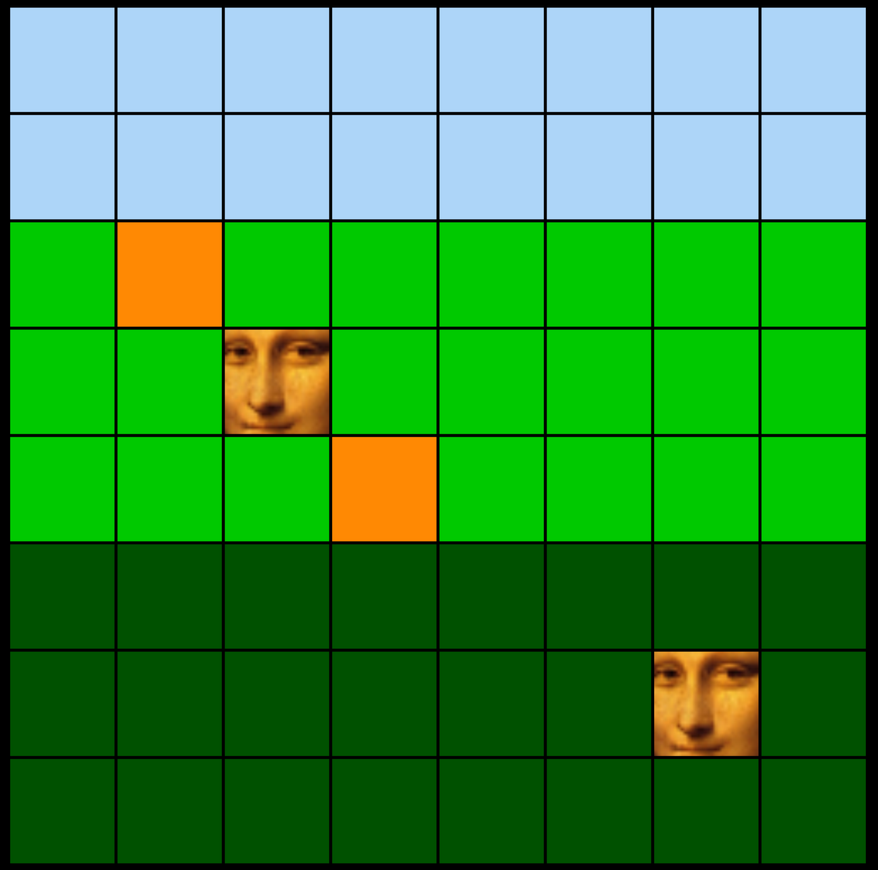


victim memory

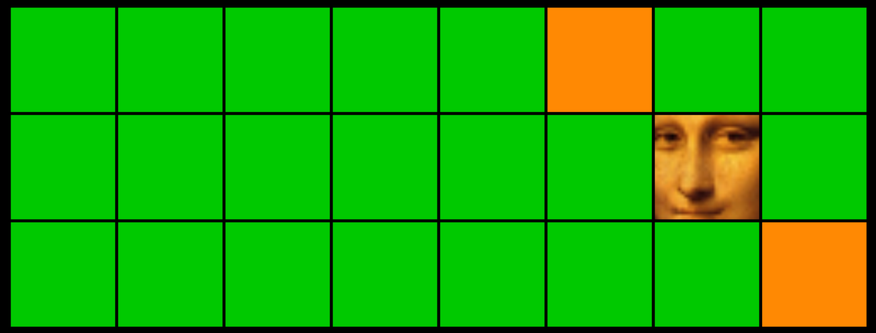


Deduplication implementation: KVM on Linux (KSM)

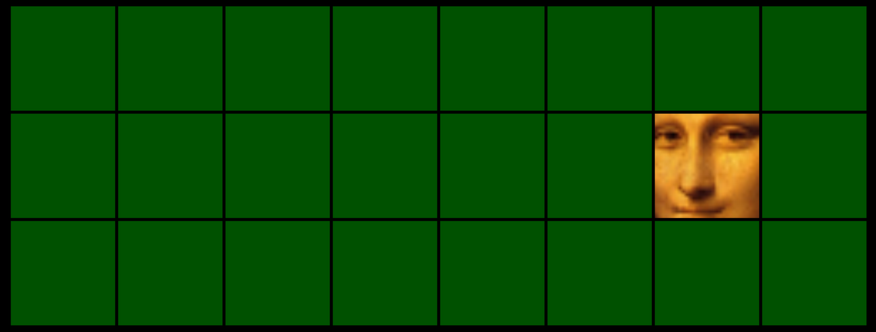
physical memory



attacker memory



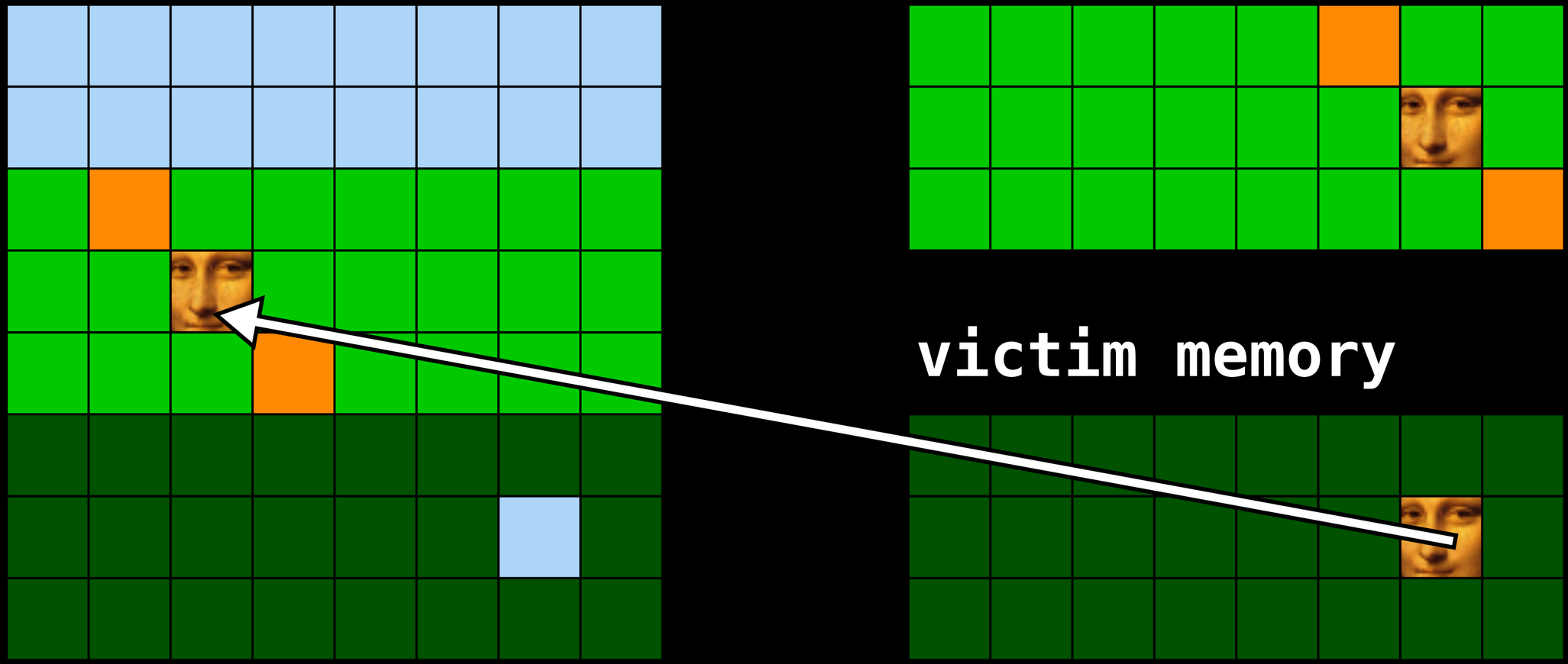
victim memory



Deduplication implementation: KVM on Linux (KSM)

physical memory

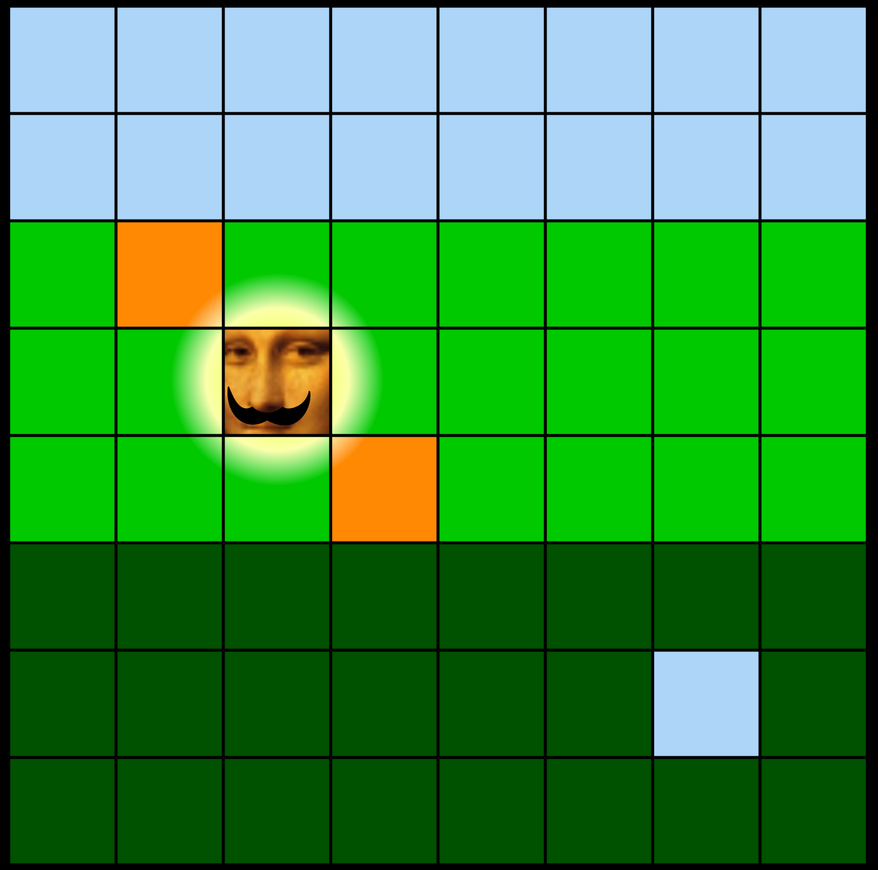
attacker memory



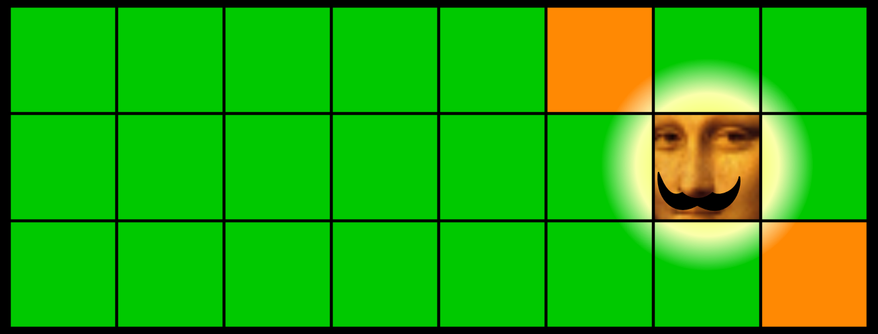
victim memory

Deduplication implementation: KVM on Linux (KSM)

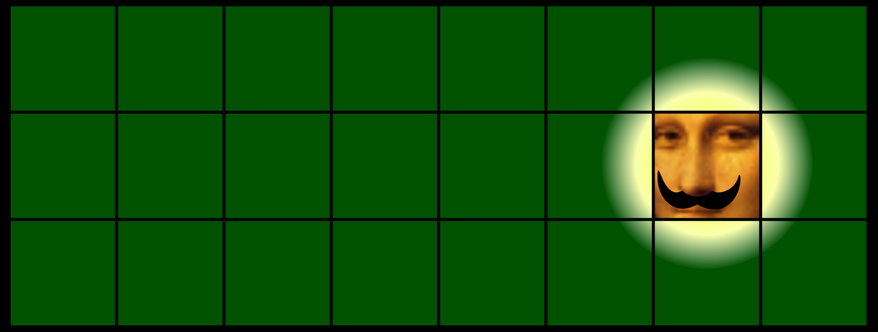
physical memory



attacker memory



victim memory



Example 1: OpenSSH

Target: ~/.ssh/authorized_keys

OpenSSH ~/.ssh/authorized_keys

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC52/Uk84iUmmic
el7ESr+/D/PWZ6Ljkhlu8yv35bEEoTwXm9eGxJyzV+1s68tRyzpD
3VQvwSHiKqDnCg+OtaAo0KvCqZcoBQFB9XawIfJI5dSeGtcUBuok
Uv+TlmAZ+D9MNNAxjuSBBH0ShbaiH65imlauISfR3VZWFE7uy6sB
26j52LhWG5BRwSkMnMRN2E2fqHaP96J9ROFlHuykw8jwUXJwl4kJ
8vRo1uhX0SVu8Z9wGrKR5b+GQWJ3Ph7vjoMVU/KoAbWnNnYKR8IT
BnkPD0LrEyAKRygEfi7gwcixOvQR79by8lL6ypJ4kM5eyobSBsNC
jmghxQj8RRzGUtd1 victim@laptop
```

Exponent

Modulus (p * q)

OpenSSH ~/.ssh/authorized_keys

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC52/Uk84iUmmic
el7ESr+/D/PWZ6Ljkhlu8yv35bEEoTwXm9eGxJyzV+1s68tRyzpD
3VQvwSHiKqDnCg+OtaAo0KvCqZcoBQFB9XawIfJI5dSeGtcUBuok
Uv+TlmAZ+D9MNNAxjuSBBH0ShbaiH65imlauISfR3VZWFE7uy6sB
26j52LhWG5BRwSkMnMRN2E2fqHaP96J9ROFlHuykw8jwUXJwl4kJ
8vRo1uhX0SVu8Z9wGrKR5b+GQWJ3Ph7vjoMVU/KoAbWnNnYKR8IT
BnkPD0LrEyAKRygEfi7gwcixOvQR79by8lL6ypJ4kM5eyobSBsNC
jmghxQj8RRzGUtd1 victim@laptop
```

■ Exponent ■ Modulus (p' * q' * r' ...)

Example 1: OpenSSH

Target: ~/.ssh/authorized_keys

> Flip a bit in the RSA modulus
> Factorize it
> Reconstruct the new private key

Example 2: GPG & apt-get

Targets: sources.list

flip package repository domain name

eg. ubuntu.com -> ubunvu.com

Example 2: GPG & apt-get

Targets: sources.list

+

GPG keyring

corrupt signing key