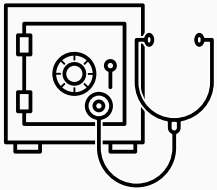
Microarchitectural Attacks and Beyond

Daniel Gruss

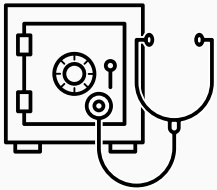
February 21, 2019

Graz University of Technology

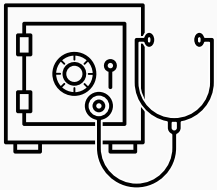
- Bug-free software does not mean safe execution



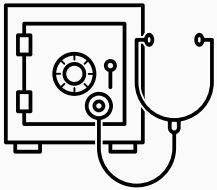
- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware



Side-Channel Attacks



- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware
- Exploit leakage through side-effects

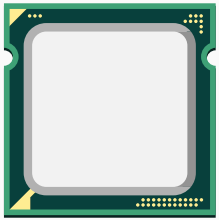


- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware
- Exploit leakage through side-effects

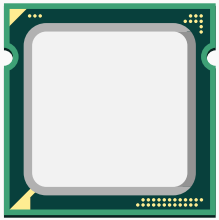
Power consumption

Execution time

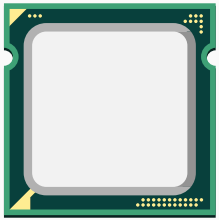
CPU caches



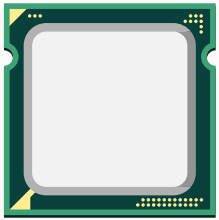
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)



- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Interface between hardware and software

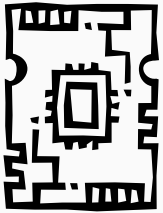


- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Interface between hardware and software
- Microarchitecture is an ISA implementation

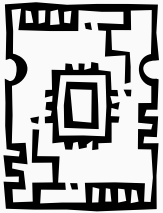


- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Interface between hardware and software
- Microarchitecture is an ISA implementation

●●●



- Modern CPUs contain multiple microarchitectural elements

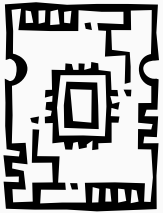


- Modern CPUs contain multiple microarchitectural elements

Caches and buffer

Predictor

Microarchitectural Elements



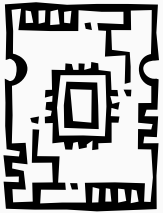
- Modern CPUs contain multiple microarchitectural elements

Caches and buffer

Predictor

- Transparent for the programmer

Microarchitectural Elements



- Modern CPUs contain multiple microarchitectural elements

Caches and buffer

Predictor

- Transparent for the programmer
- Timing optimizations → side-channel leakage

1996

1996

2004

1996

2004

2006

1996

2004

2006

2009

1996

2004

2006

2009

2011

1996

2004

2006

2009

2011

1996

2004

2006

2009

2011

2013

1996

2004

2006

2009

2011

2013

1996

2004

2006

2009

2011

2013

1996

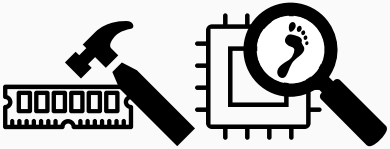
2004

2006

2009

2011

2013



2014

1996

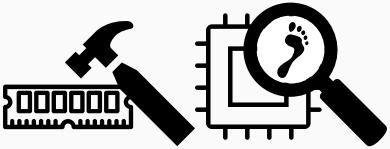
2004

2006

2009

2011

2013



2014

1996

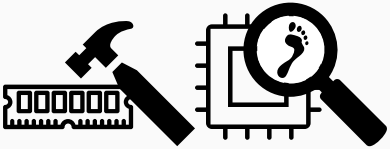
2004

2006

2009

2011

2013



2014

1996

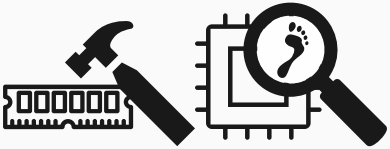
2004

2006

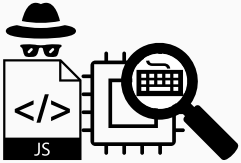
2009

2011

2013



2014



1996

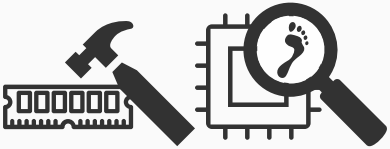
2004

2006

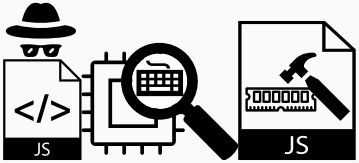
2009

2011

2013



2014



2015

1996

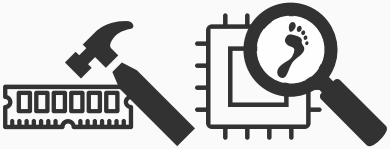
2004

2006

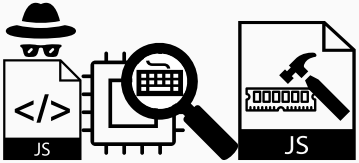
2009

2011

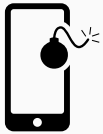
2013



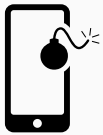
2014



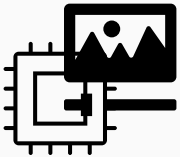
2015



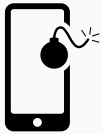
2016



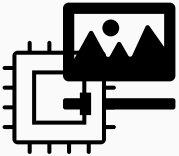
2016



2017



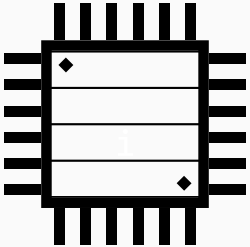
2016

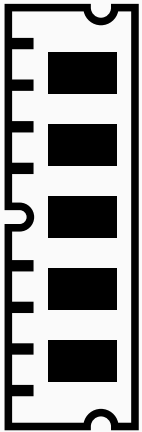


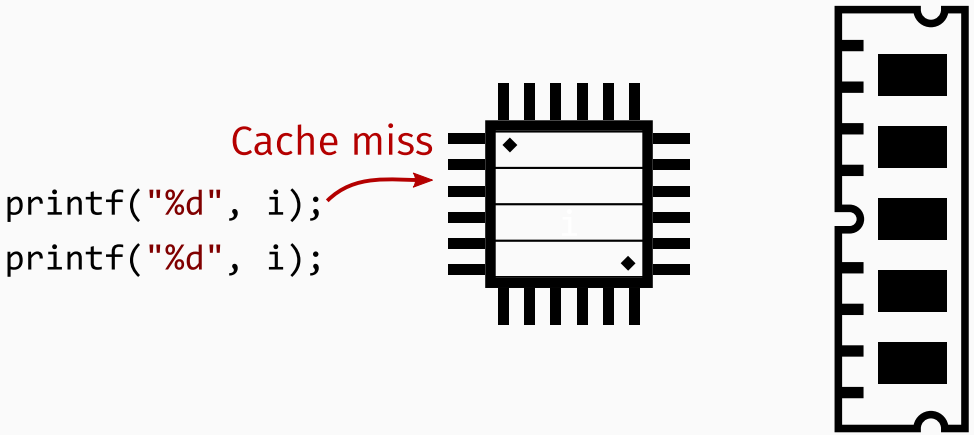
2017

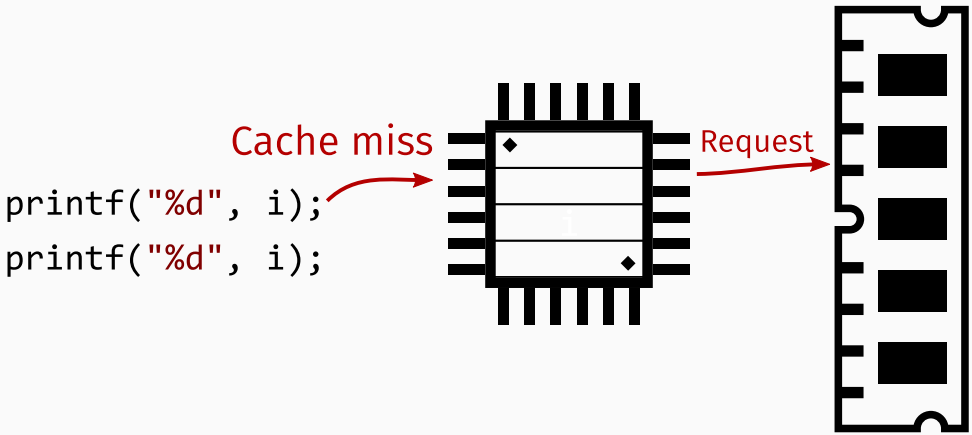
2018

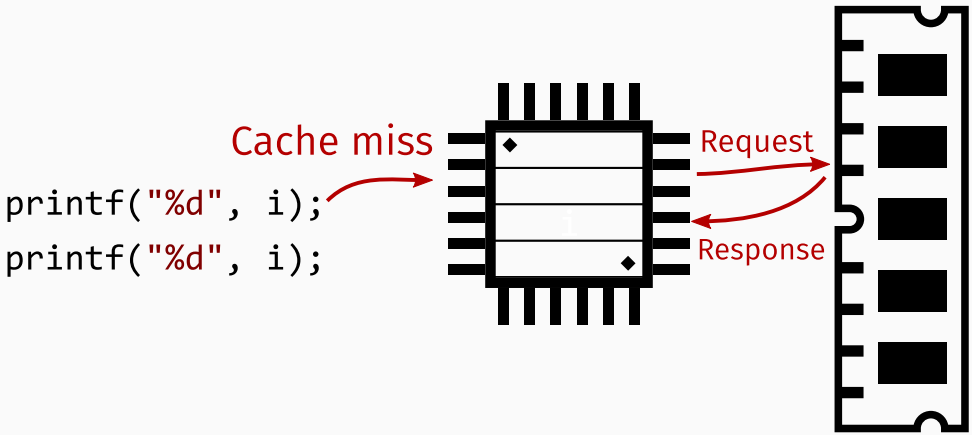
```
printf("%d", i);
printf("%d", i);
```

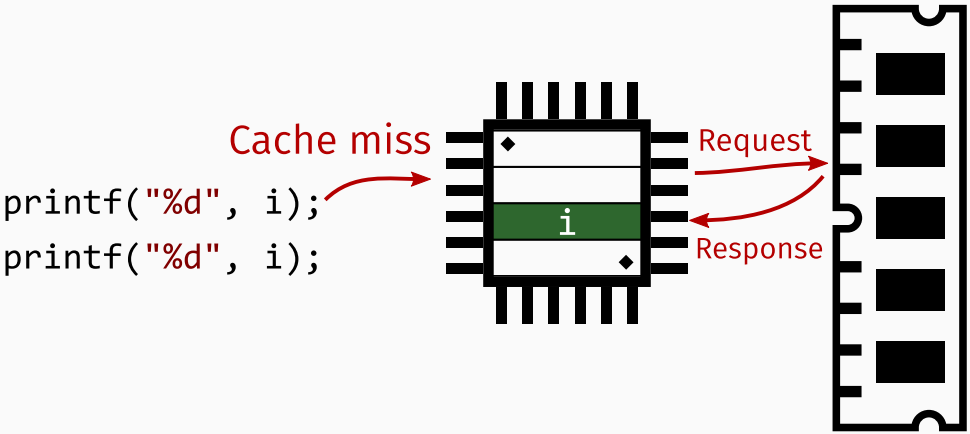


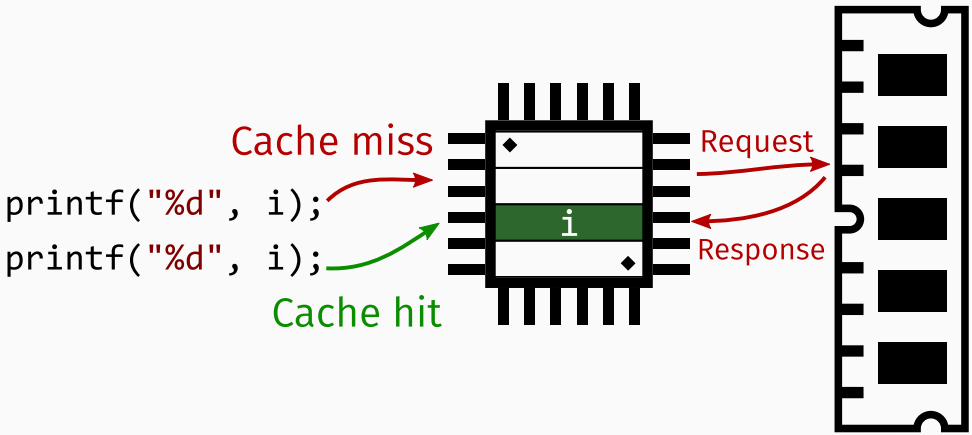


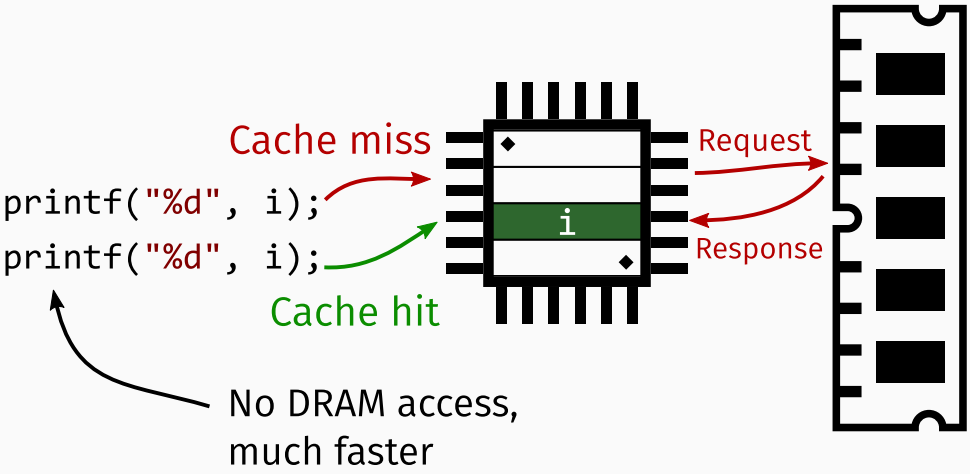

Cache miss
printf("%d", i);
printf("%d", i);

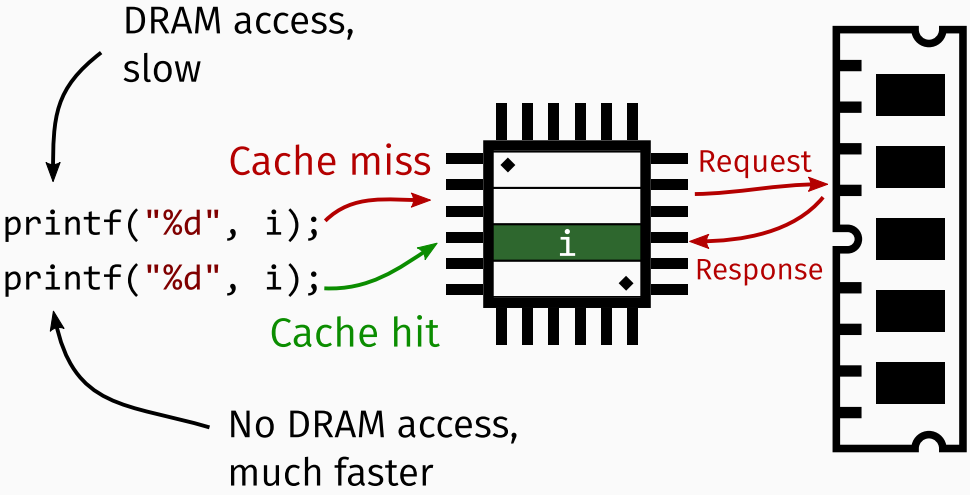

Cache miss
Request
printf("%d", i);
printf("%d", i);


Cache miss
printf("%d", i);
printf("%d", i);
Request
Response


Cache miss
printf("%d", i);
printf("%d", i);
i
Request
Response


Cache miss
printf("%d", i);
printf("%d", i);
Cache hit
i
Request
Response


Cache miss
printf("%d", i);
printf("%d", i);
Cache hit
i
Request
Response
No DRAM access,
much faster

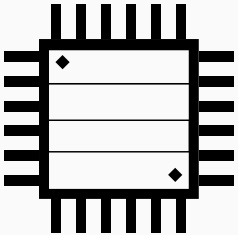

DRAM access,
slow
Cache miss
printf("%d", i);
printf("%d", i);
Cache hit
No DRAM access,
much faster
i
Request
Response

Shared Memory

ATTACKER

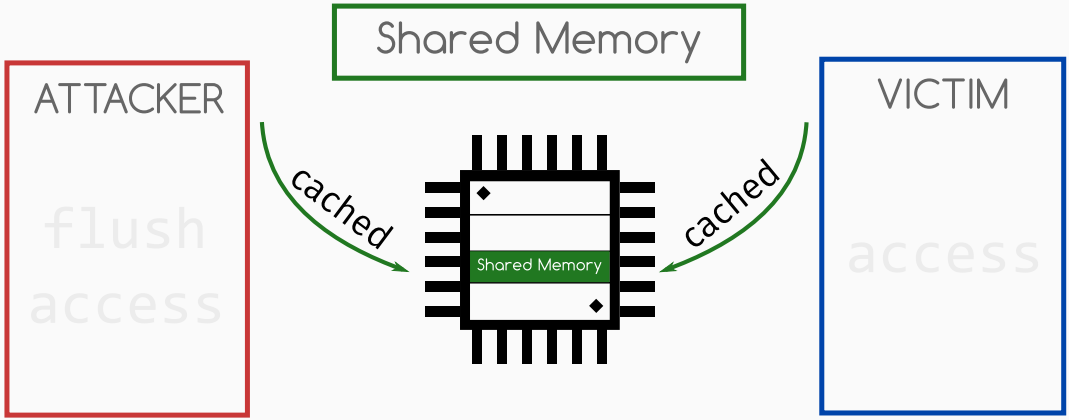
flush

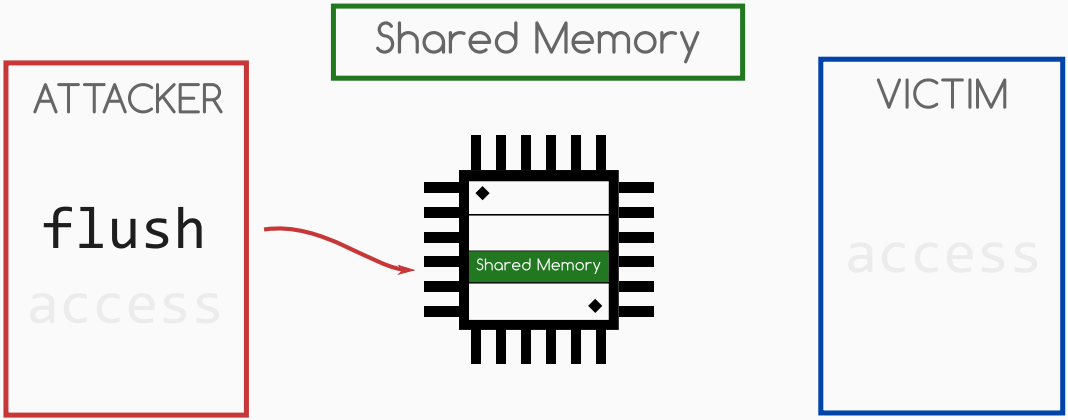
access



VICTIM

access


Shared Memory
ATTACKER
flush
access
cached
Shared Memory
cached
VICTIM
access

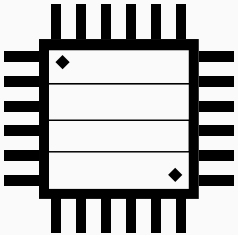

Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access

Shared Memory

ATTACKER

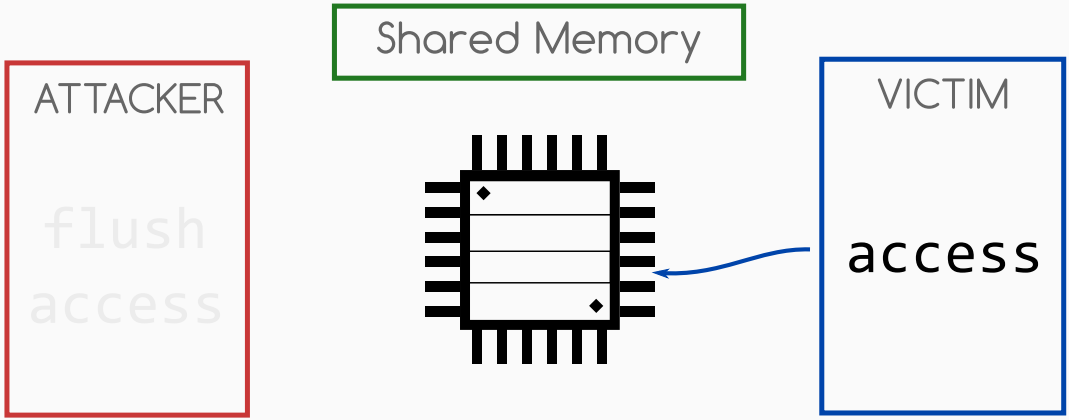
flush

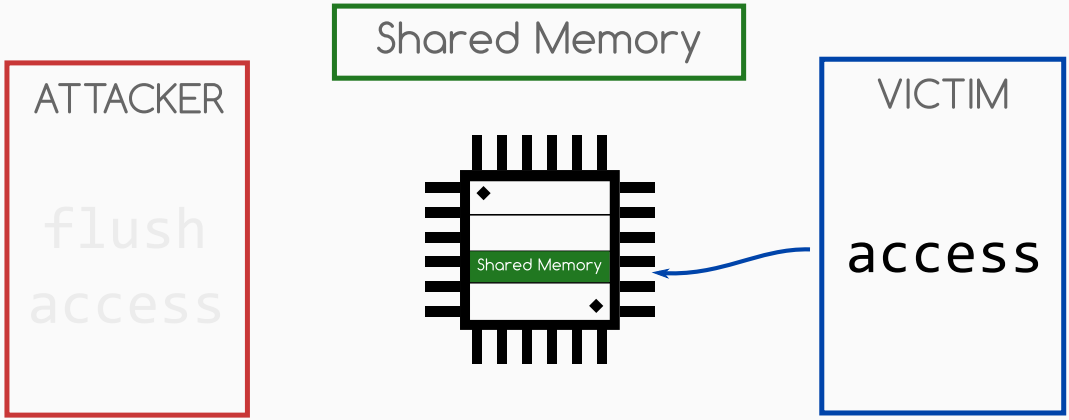
access

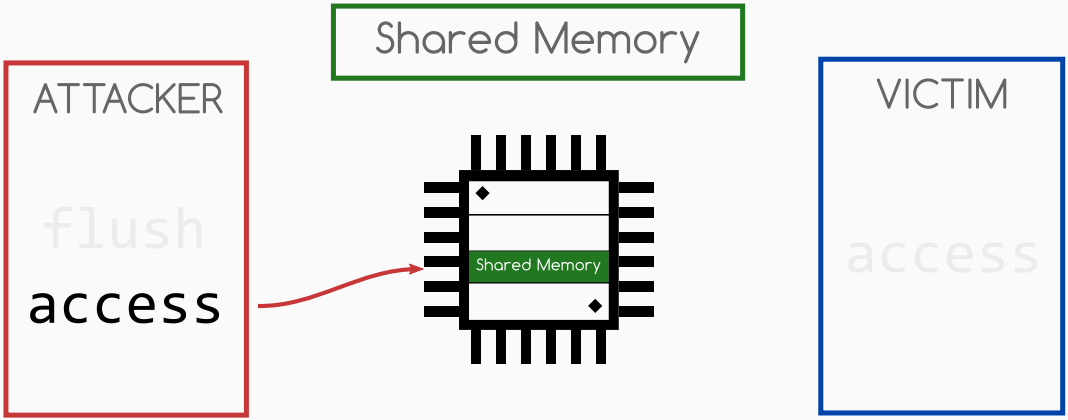


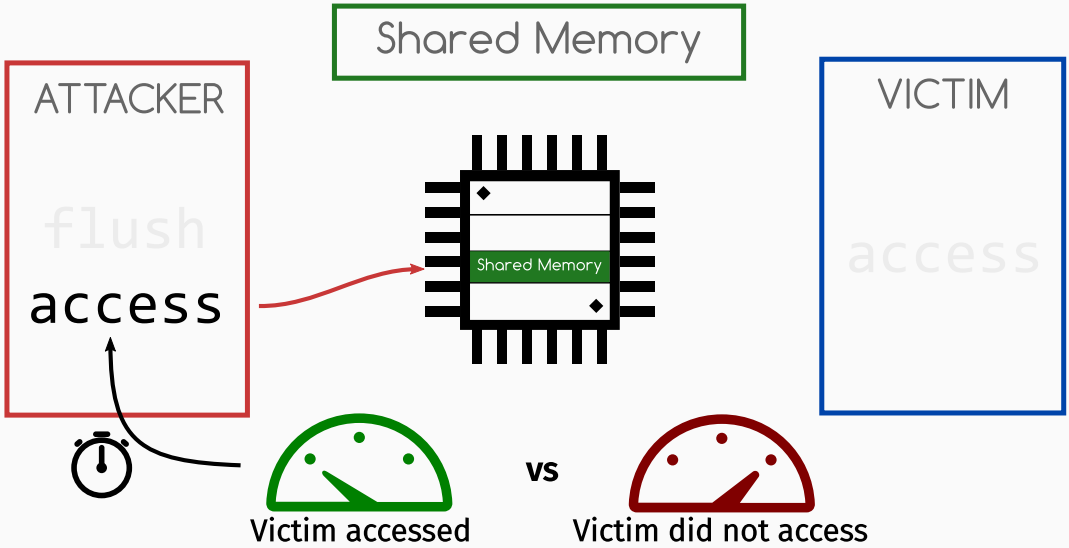
VICTIM

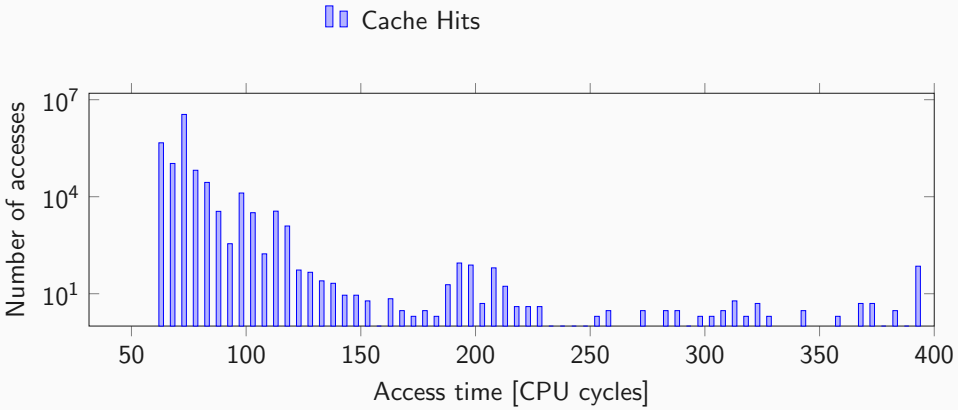
access

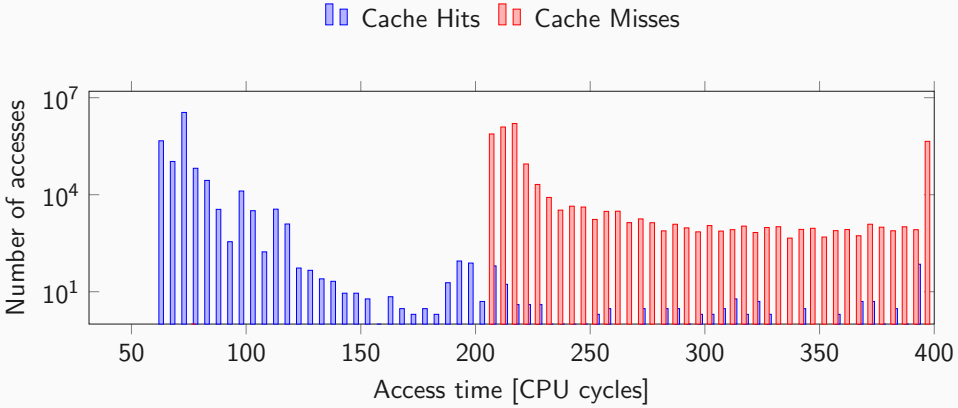

Shared Memory
ATTACKER
flush
access
VICTIM
access

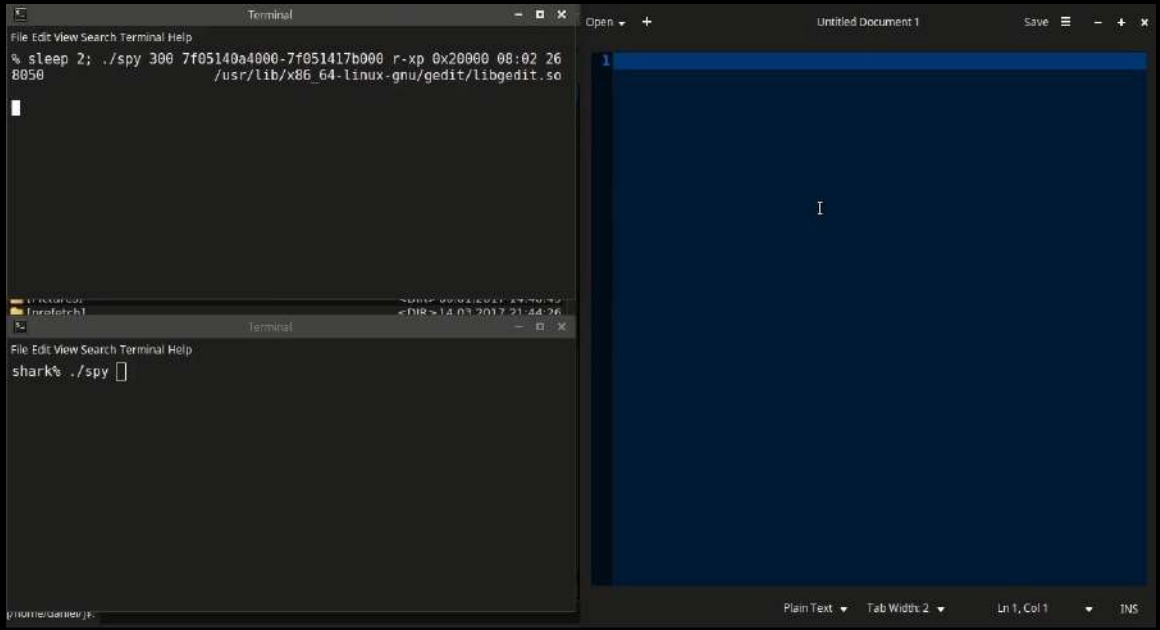

Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access


Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access

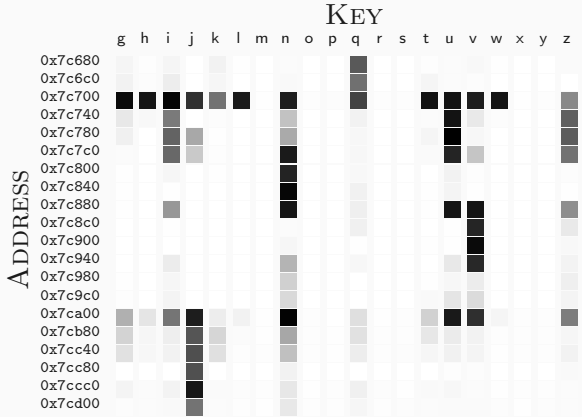

Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access
vs
Victim accessed
Victim did not access


Cache Hits
Number of accesses
10^7
10^4
10^1
50
100
150
200
250
300
350
400
Access time [CPU cycles]


Cache Hits
Cache Misses
Number of accesses
10^7
10^4
10^1
50
100
150
200
250
300
350
400
Access time [CPU cycles]



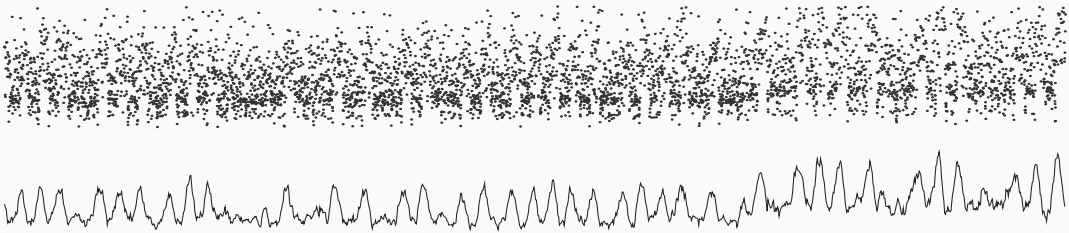
Cache Template Attack Demo


Key
g h i j k l m n o p q r s t u v w x y z
Address
0x7c680
0x7c6c0
0x7c700
0x7c740
0x7c780
0x7c7c0
0x7c800
0x7c840
0x7c880
0x7c8c0
0x7c900
0x7c940
0x7c980
0x7c9c0
0x7ca00
0x7cb80
0x7cc40
0x7cc80
0x7ccc0
0x7cd00

Raw Prime+Probe trace...[1]

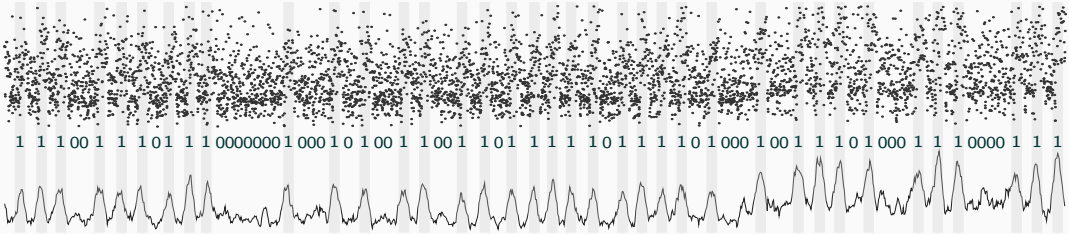
[1] Michael Schwarz et al. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.

...processed with a simple moving average...[1]

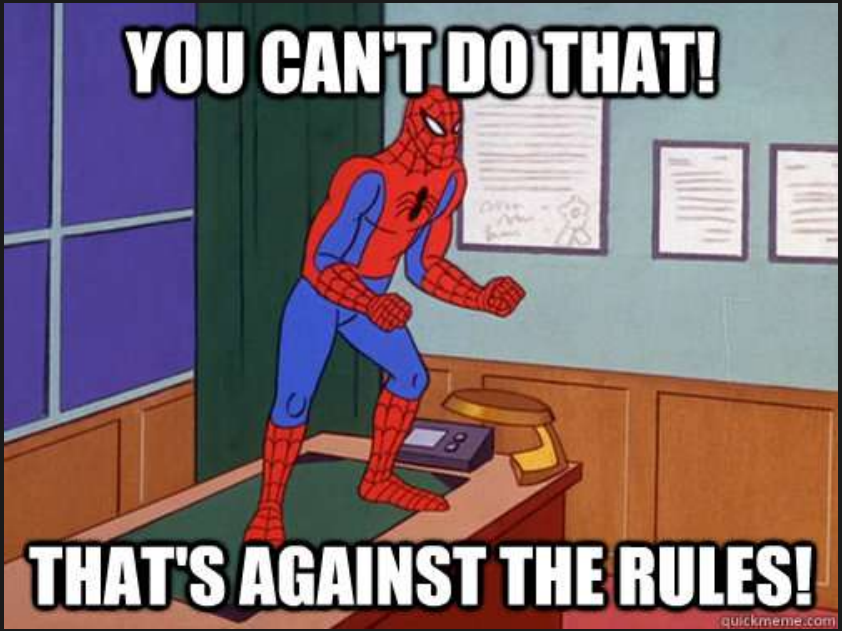


[1] Michael Schwarz et al. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.

...allows to clearly see the bits of the exponent[1]



[1] Michael Schwarz et al. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.

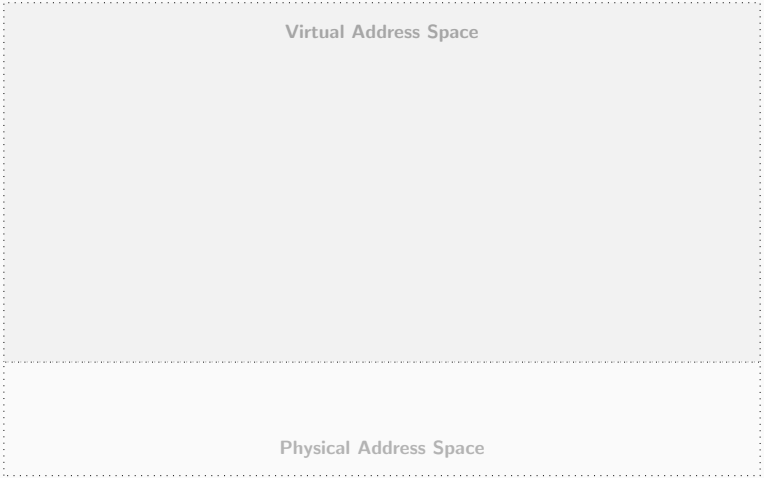

YOU CAN'T DO THAT!
THAT'S AGAINST THE RULES!
quickmeme.com

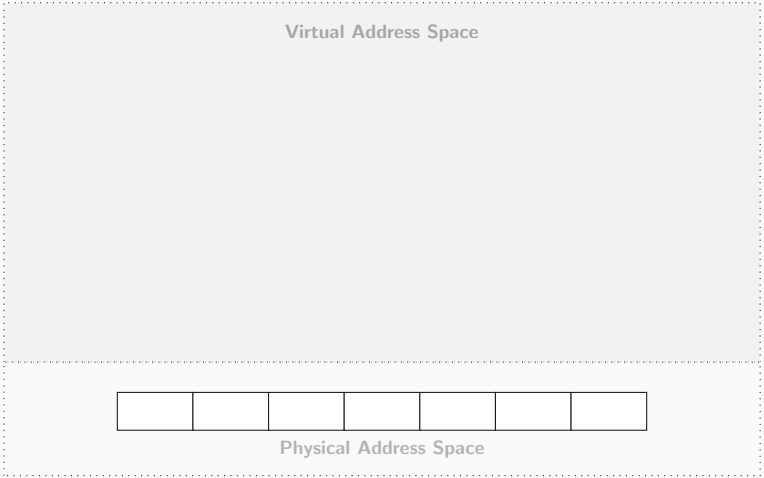
Yet another cache: Linux page cache

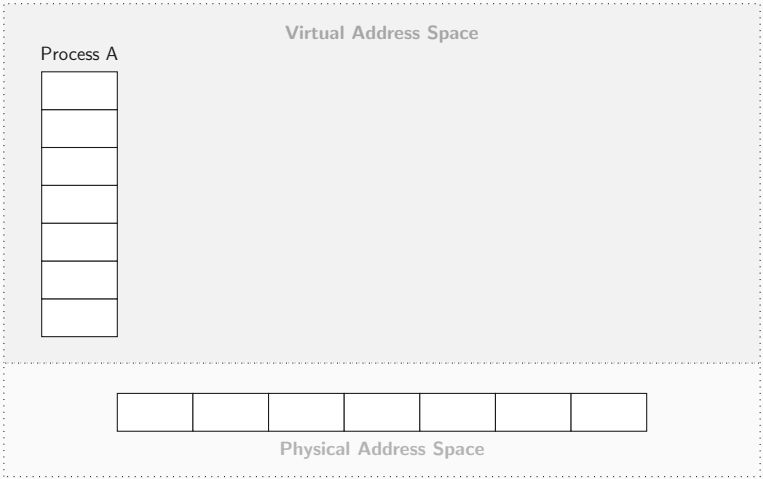
- Files buffered page-wise in “page cache”
- Lower access time for frequently accessed data

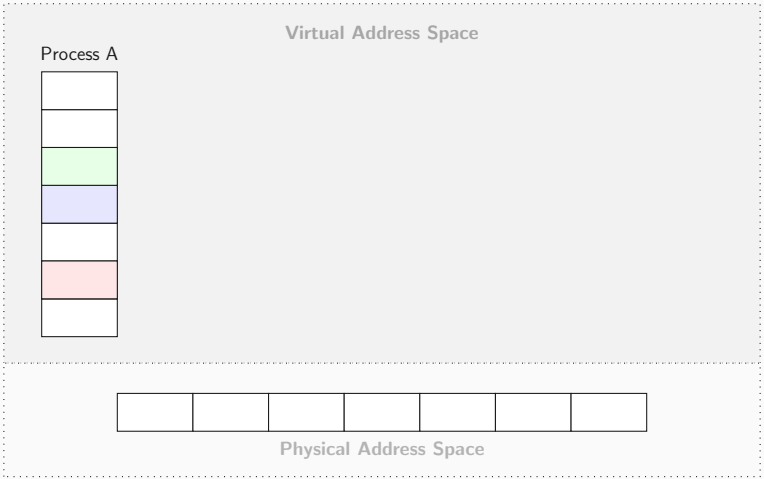
Yet another cache: Linux page cache

- Files buffered page-wise in “page cache”
- Lower access time for frequently accessed data
- Use up all the memory
- Pages are freed on demand
- Deduplicate pages (copy-on-write)

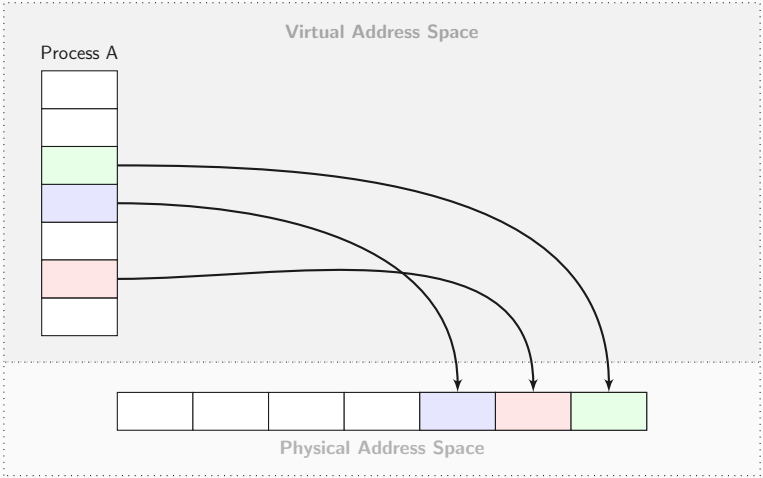

Virtual Address Space
Physical Address Space

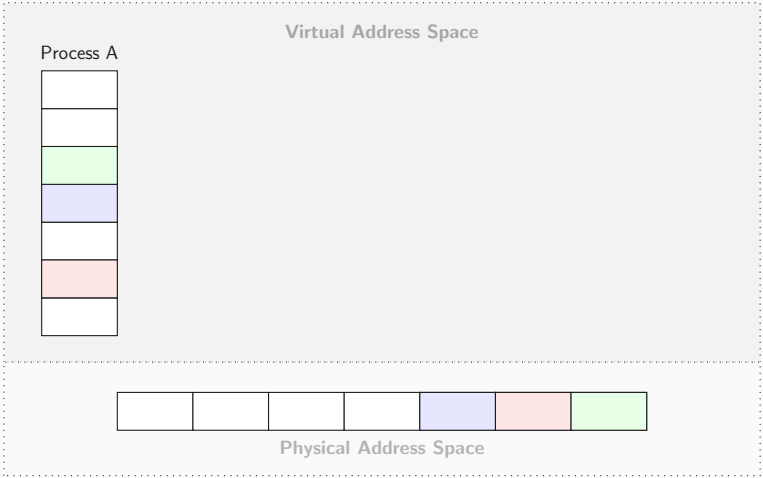

Virtual Address Space
Physical Address Space


Virtual Address Space
Process A
Physical Address Space

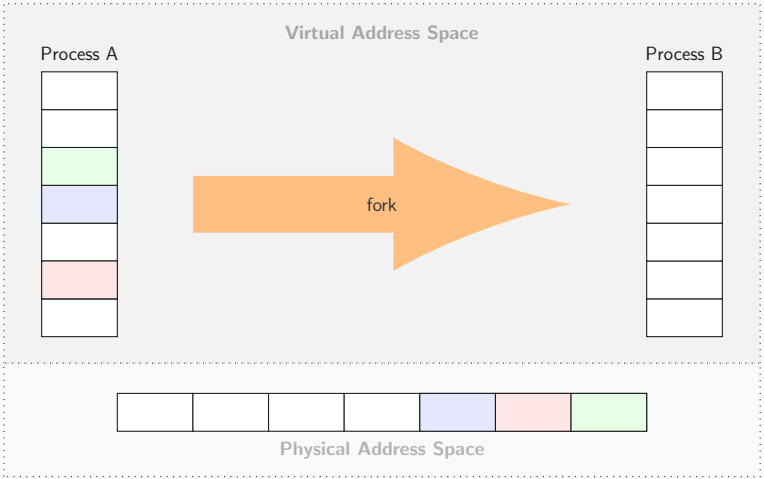

Virtual Address Space
Process A
Physical Address Space

Copy-on-Write

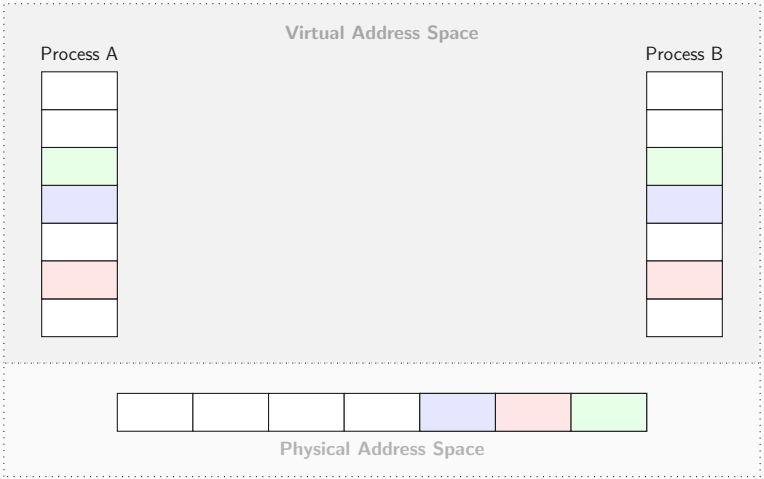



Virtual Address Space
Process A
Physical Address Space

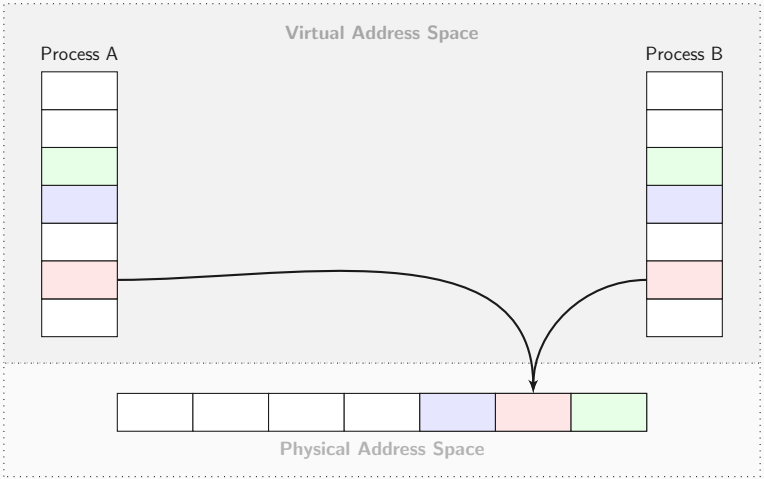
Copy-on-Write



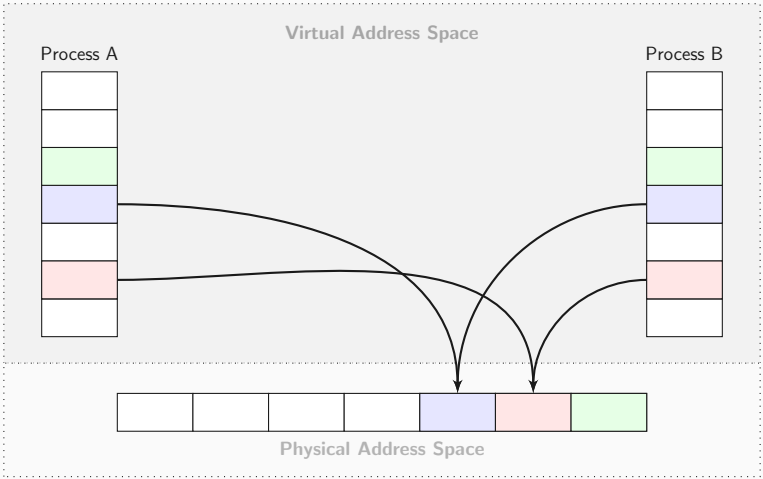
Copy-on-Write



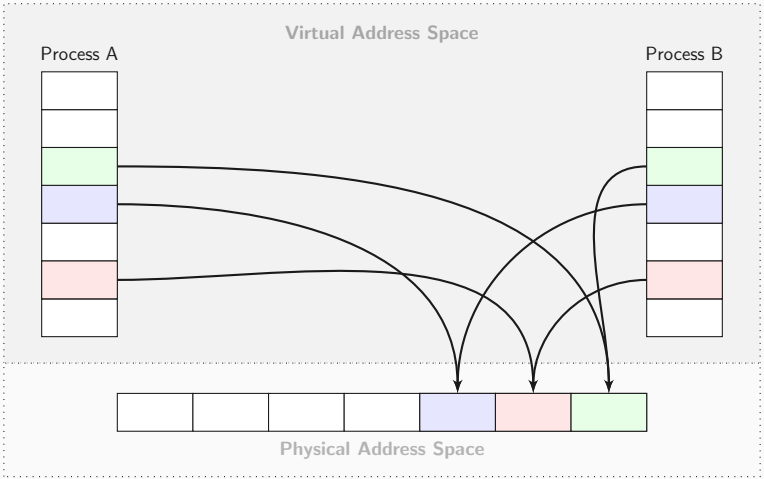
Copy-on-Write



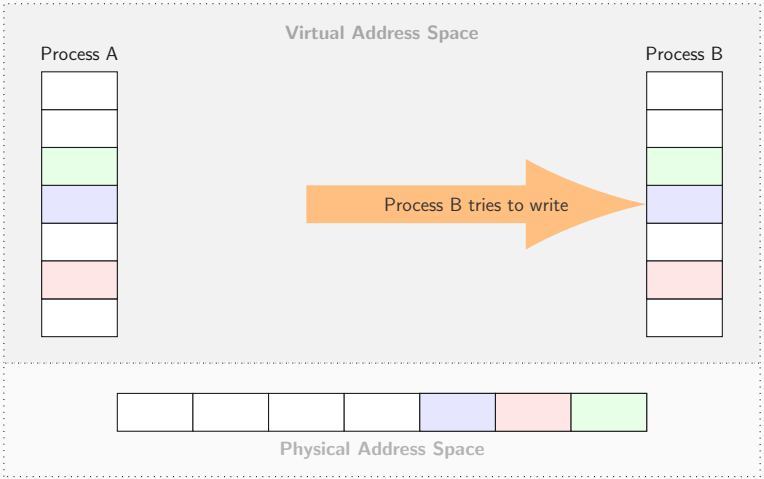
Copy-on-Write



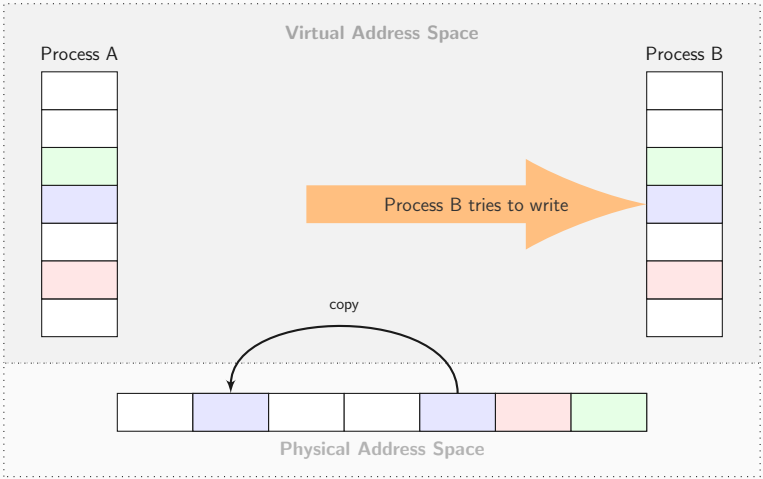
Copy-on-Write



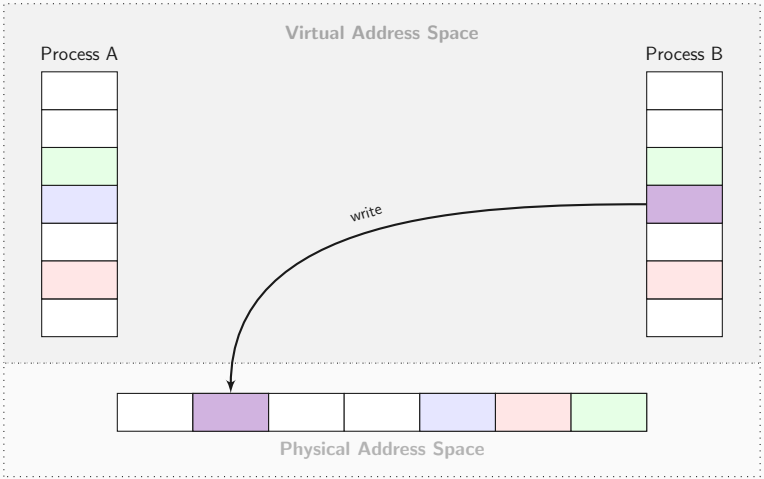
Copy-on-Write



Copy-on-Write



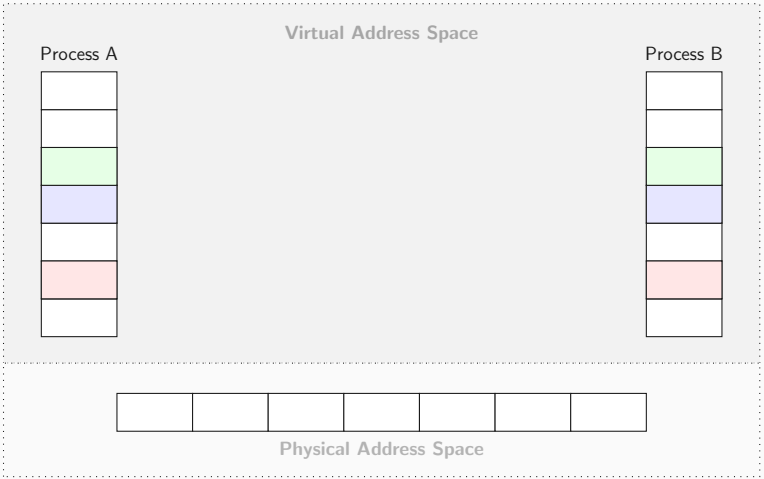
Copy-on-Write



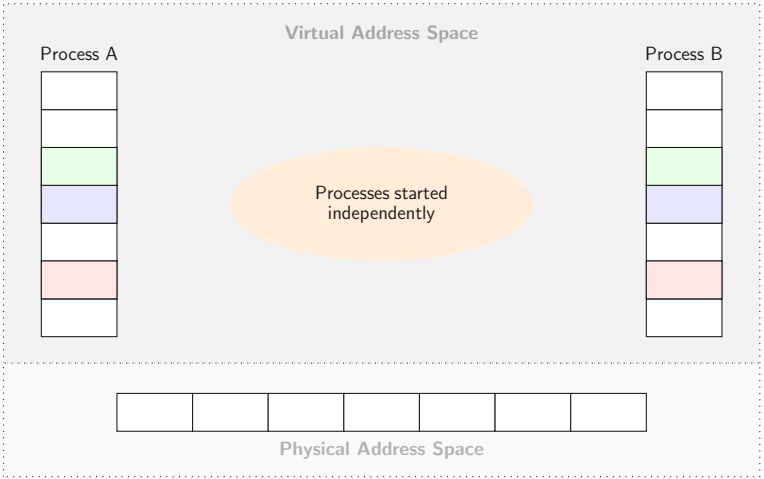
Write vs. Copy-on-Write

- Regular write access $< 0.2\mu s$
- Write access with copy-on-write pagefault $> 3.0\mu s$
- Clearly distinguishable

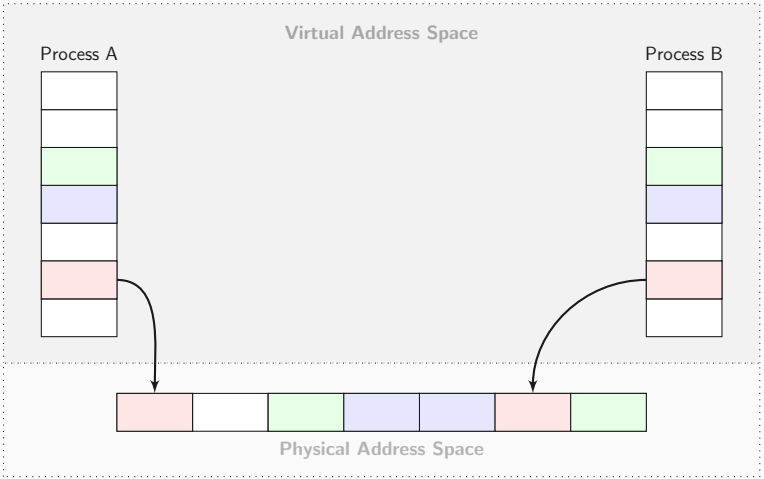
Page Deduplication



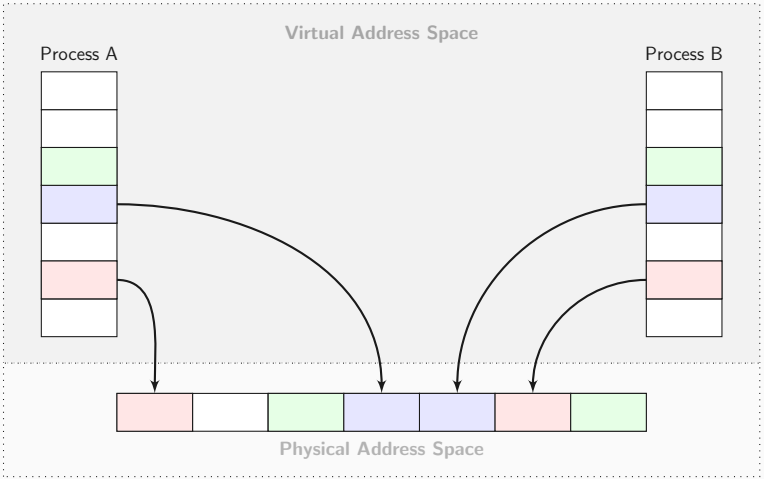
Page Deduplication



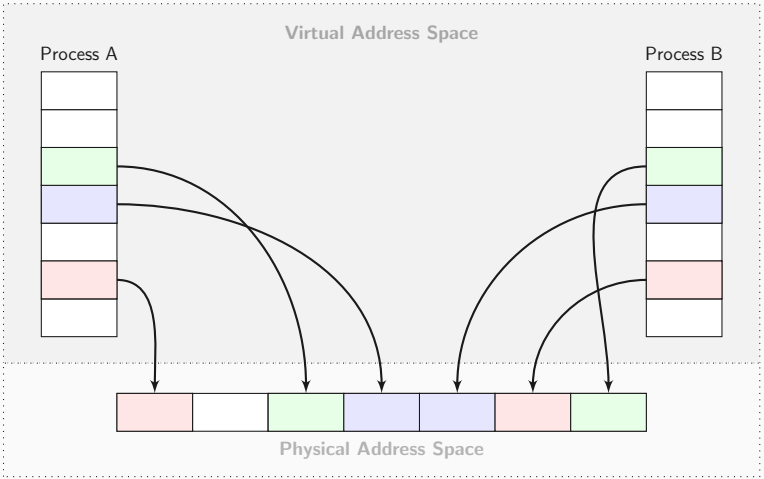
Page Deduplication



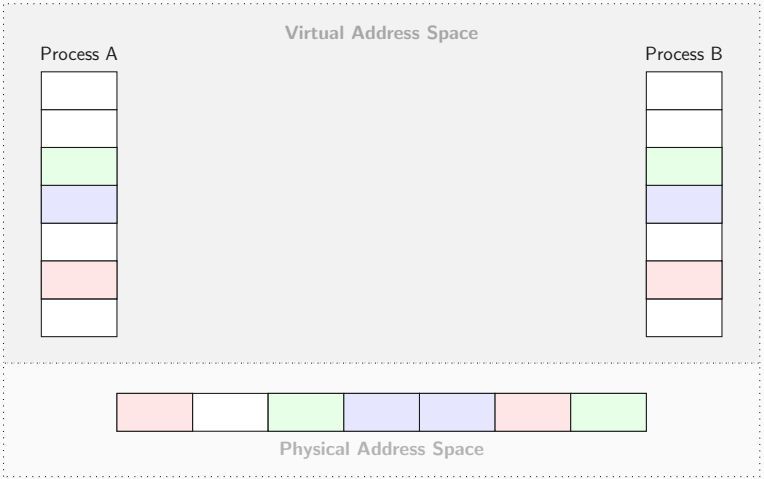
Page Deduplication



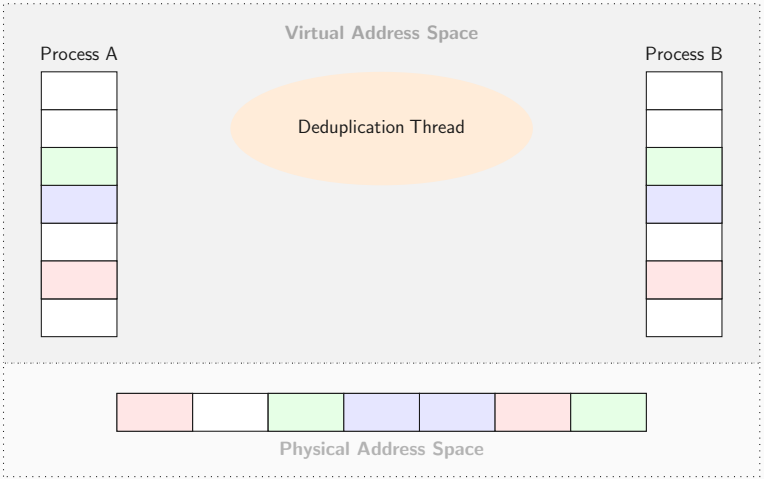
Page Deduplication



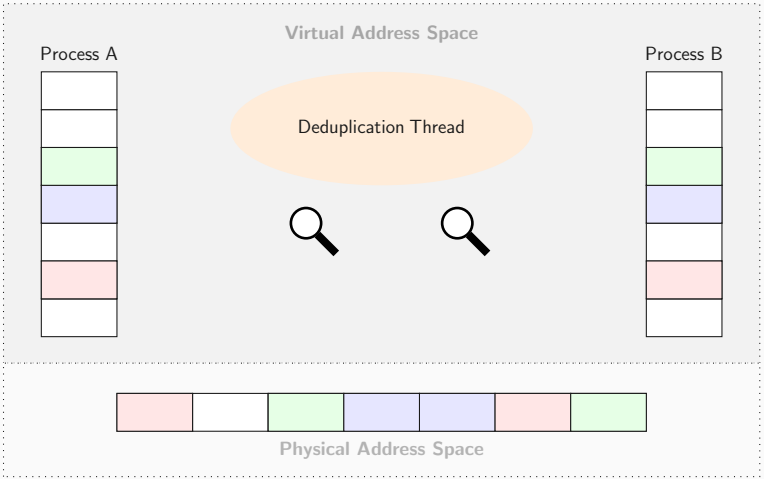
Page Deduplication



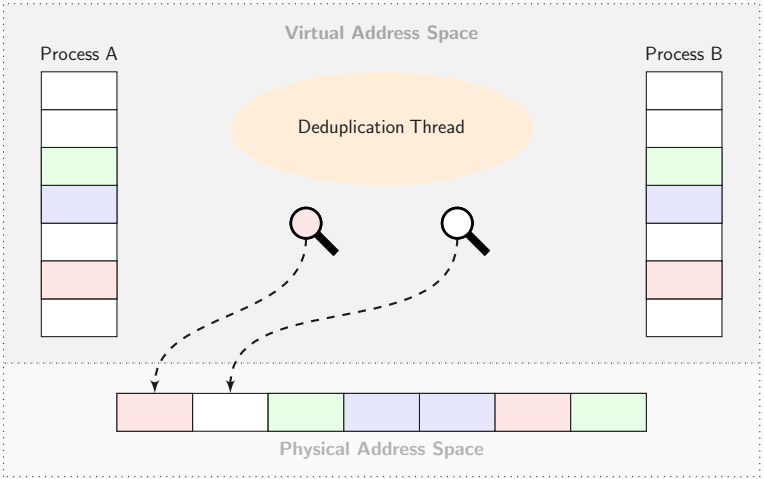
Page Deduplication



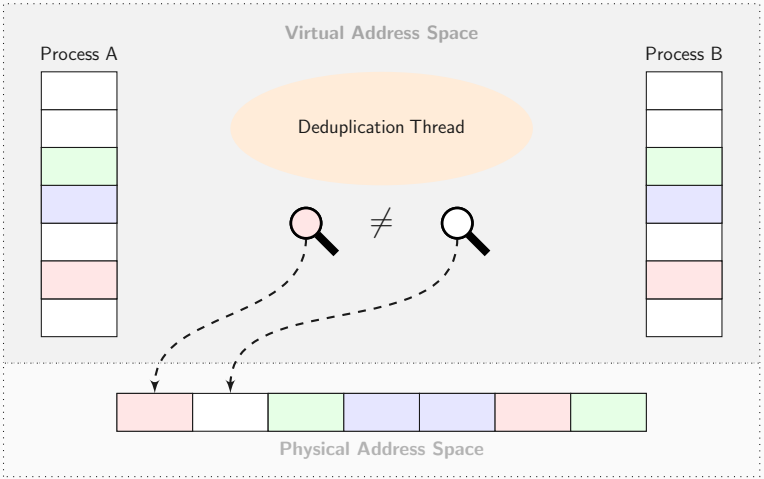
Page Deduplication

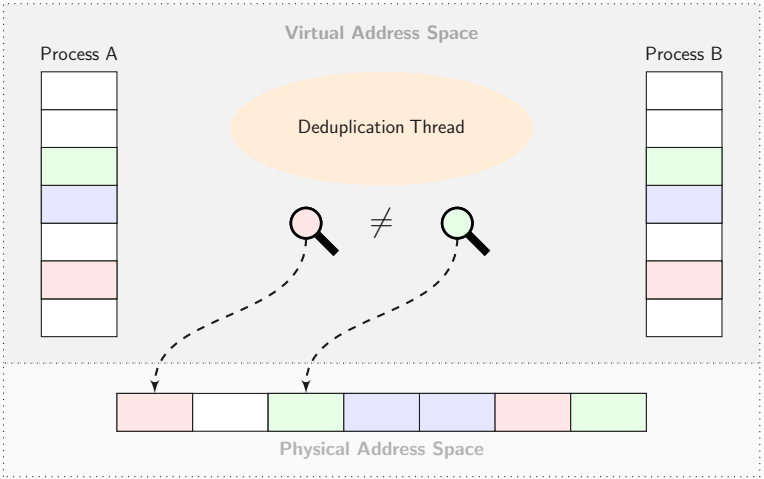


Page Deduplication

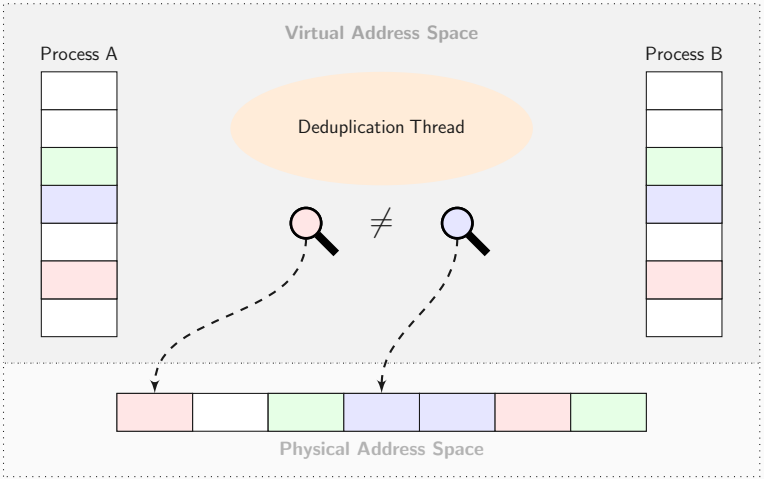


Page Deduplication

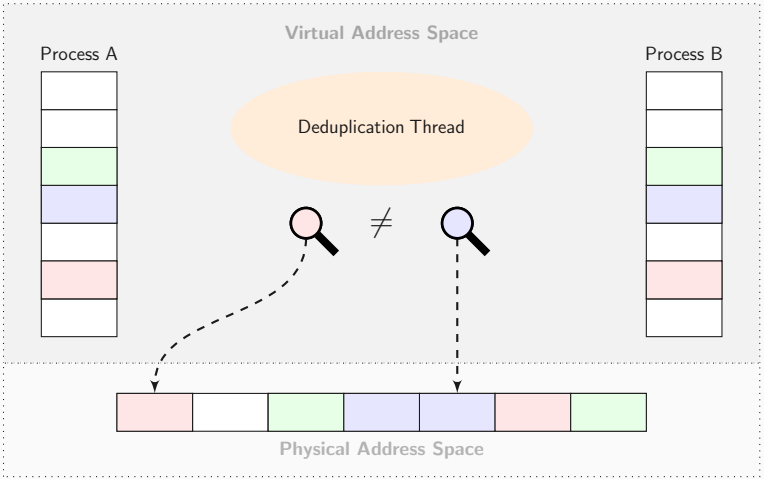


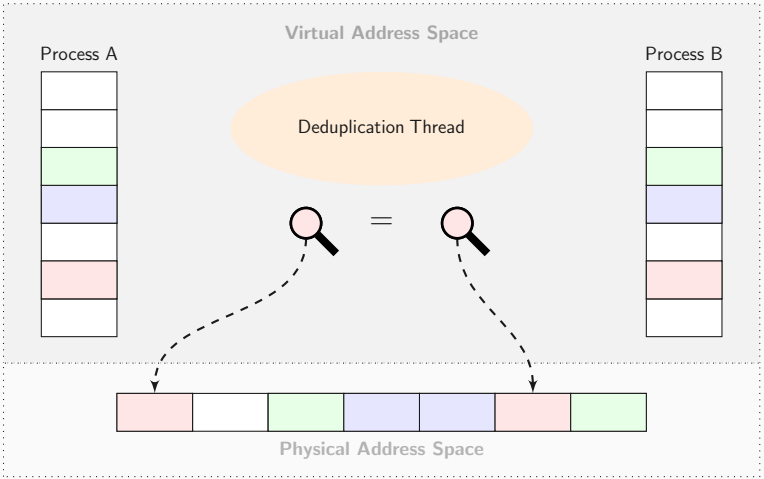

Virtual Address Space
Process A
Process B
Deduplication Thread
≠
Physical Address Space

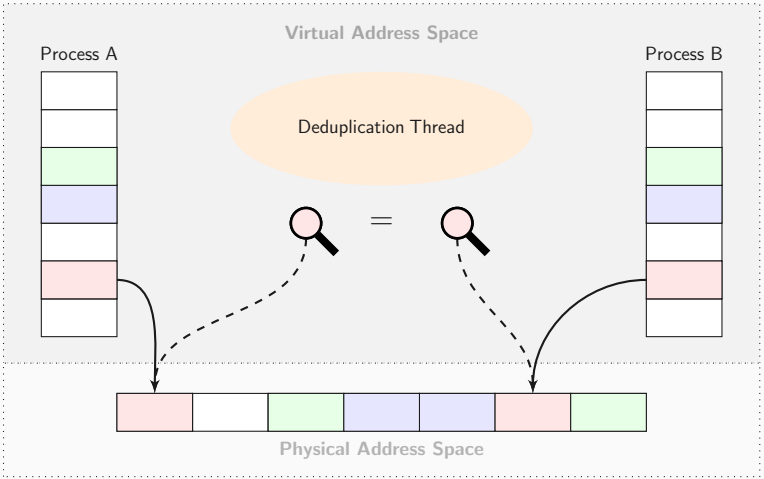
Page Deduplication

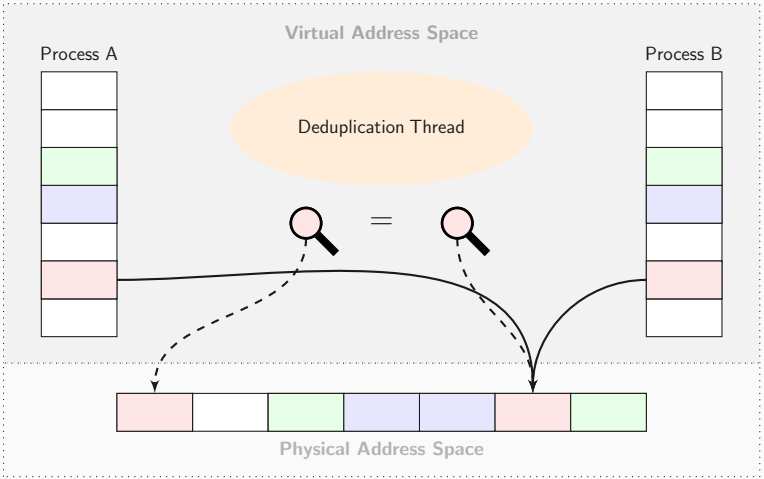


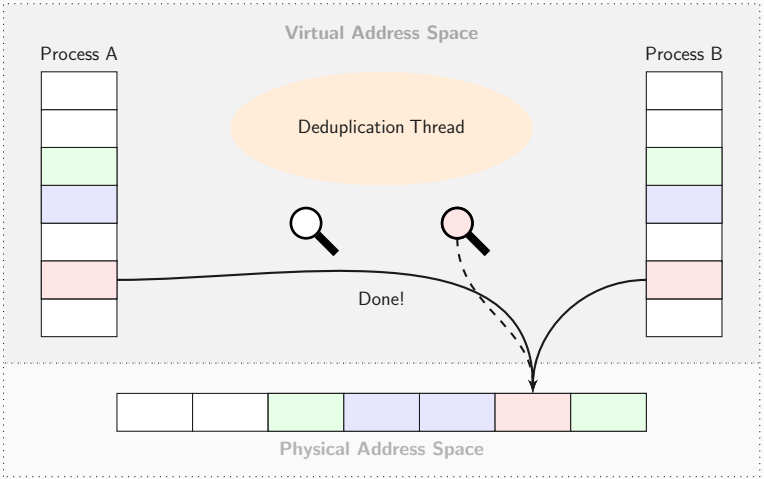
Page Deduplication



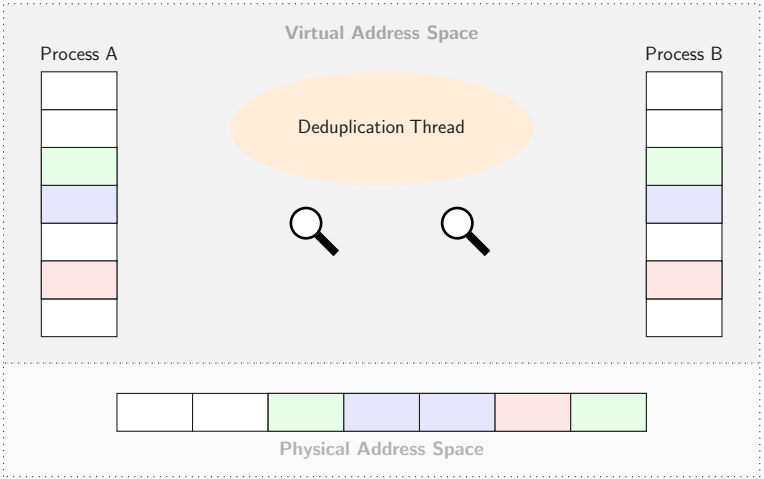

Virtual Address Space
Process A
Process B
Deduplication Thread
=
Physical Address Space


Virtual Address Space
Process A
Process B
Deduplication Thread
=
Physical Address Space


Virtual Address Space
Process A
Process B
Deduplication Thread
=
Physical Address Space


Virtual Address Space
Process A
Process B
Deduplication Thread
Done!
Physical Address Space

Page Deduplication

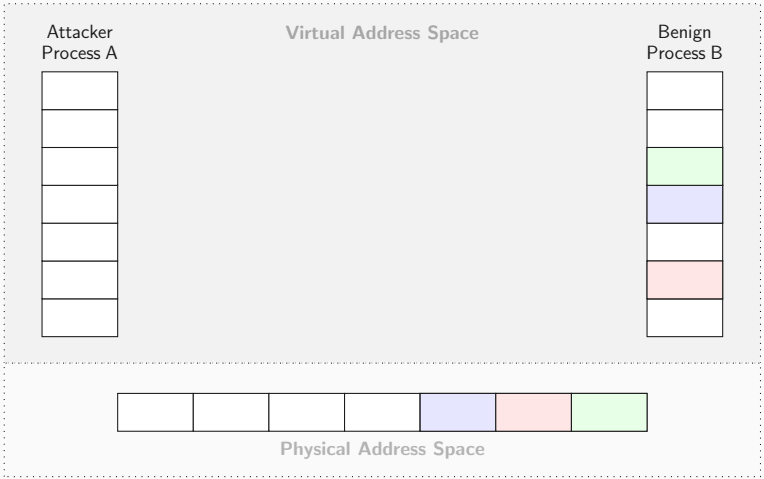


- Deduplication between processes:
  1. in same OS instance (Android, Windows)
  2. in different VMs (KVM, VMWare, ...)

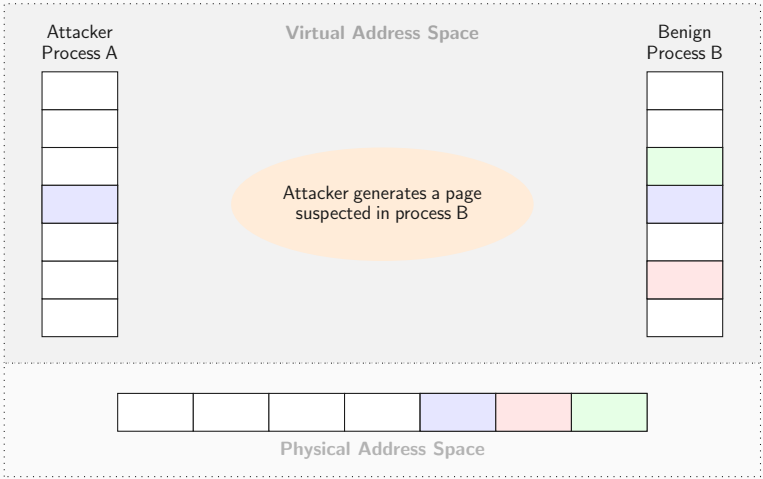
Page Deduplication (without fork)

- Deduplication between processes:
  1. in same OS instance (Android, Windows)
  2. in different VMs (KVM, VMWare, ...)
- Code pages, data pages - even kernel pages
- Time until deduplication 2-45 minutes
  - depends on system configuration

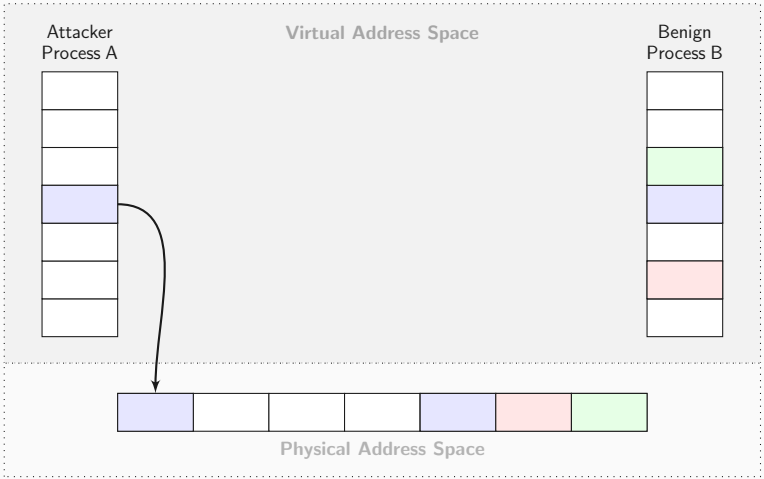
Page Deduplication Attack



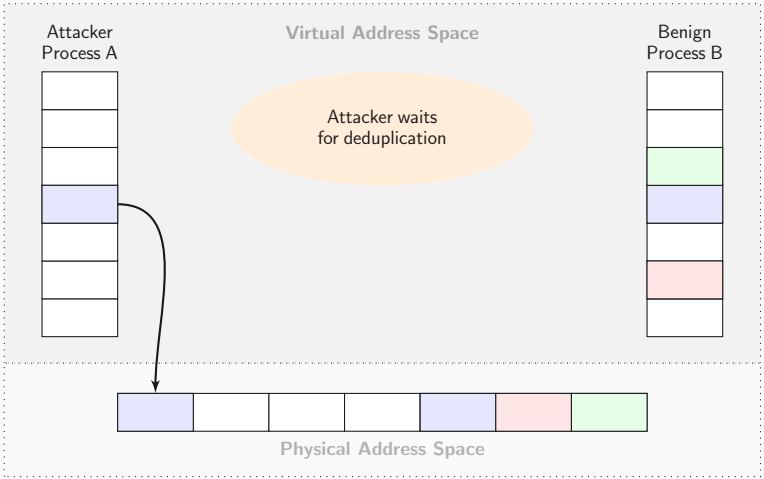
Page Deduplication Attack



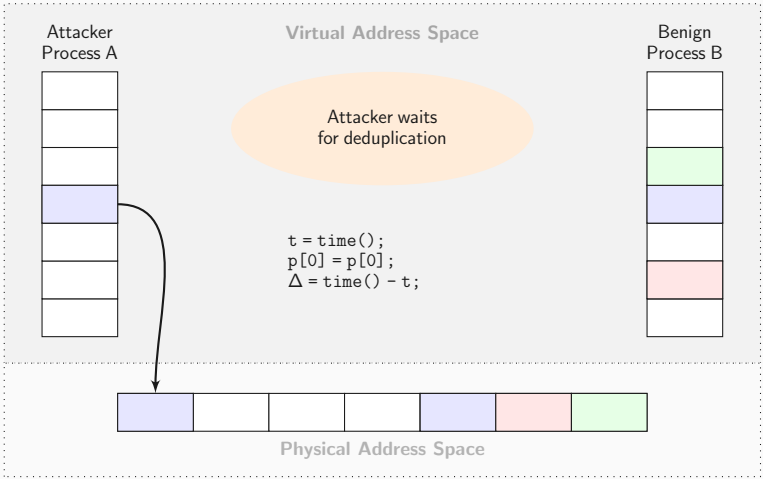
Page Deduplication Attack



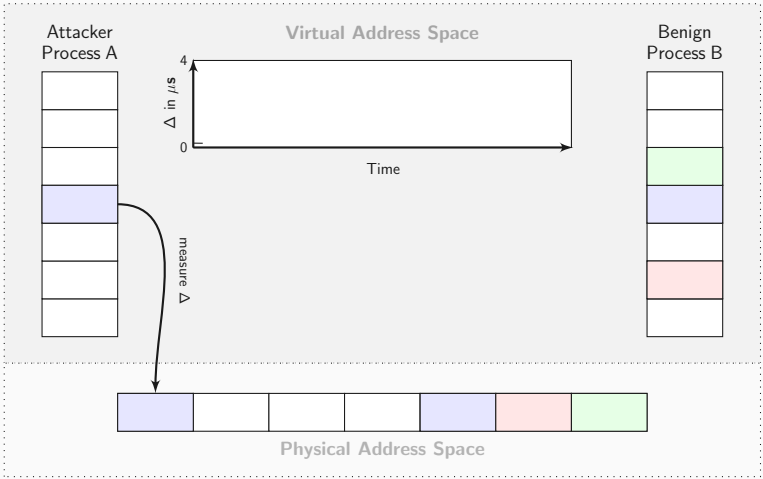
Page Deduplication Attack



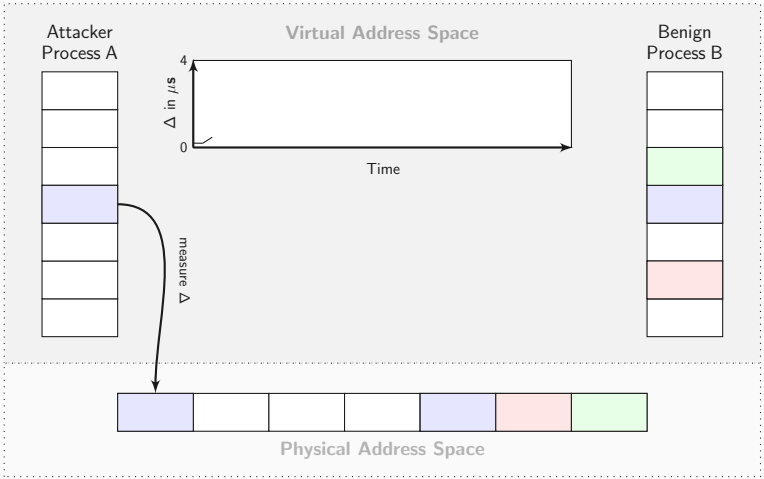
Page Deduplication Attack



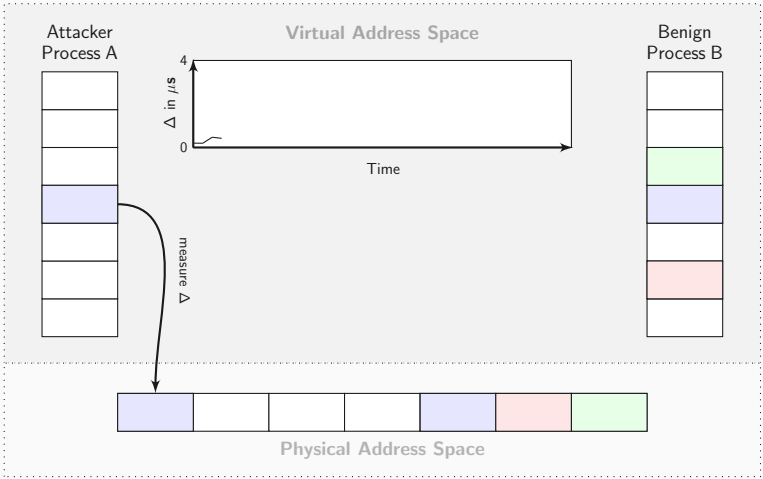
Page Deduplication Attack



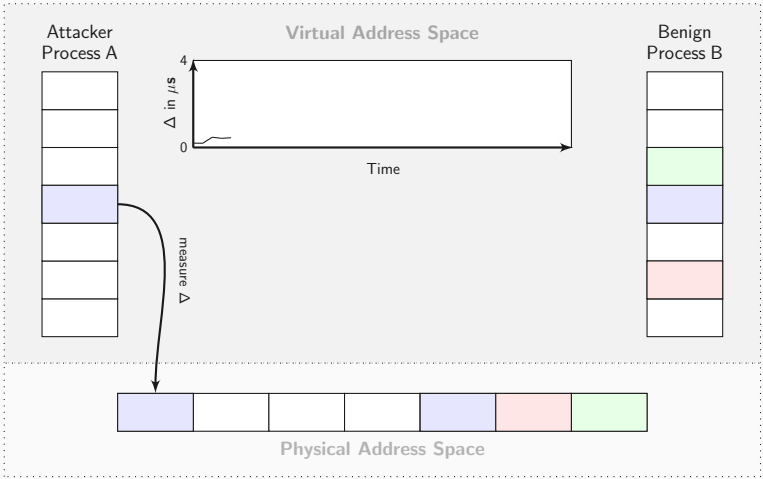
Page Deduplication Attack



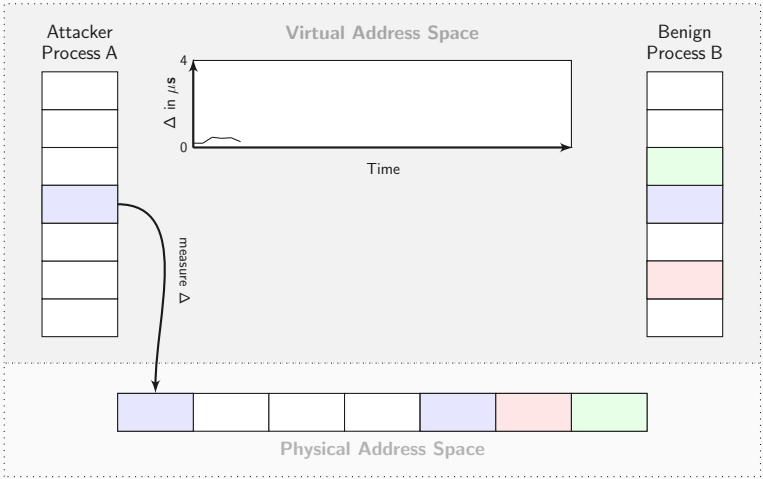
Page Deduplication Attack



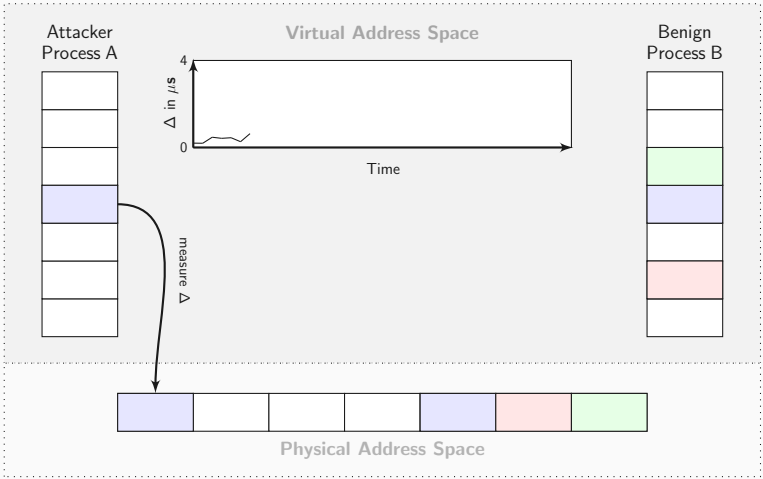
Page Deduplication Attack



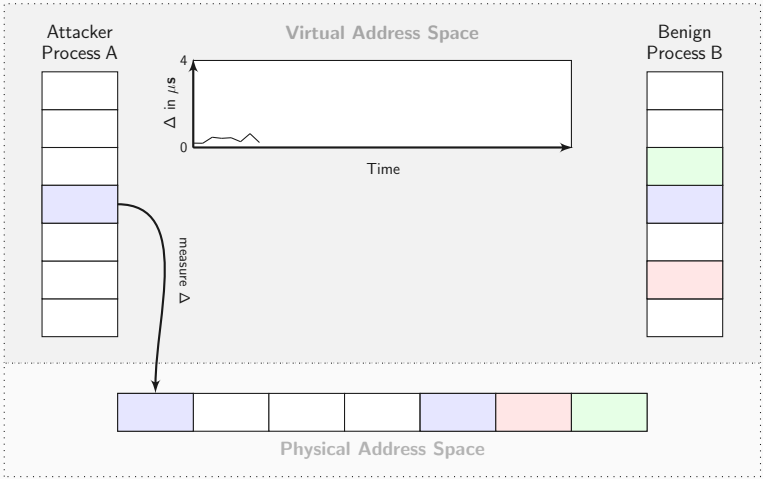
Page Deduplication Attack



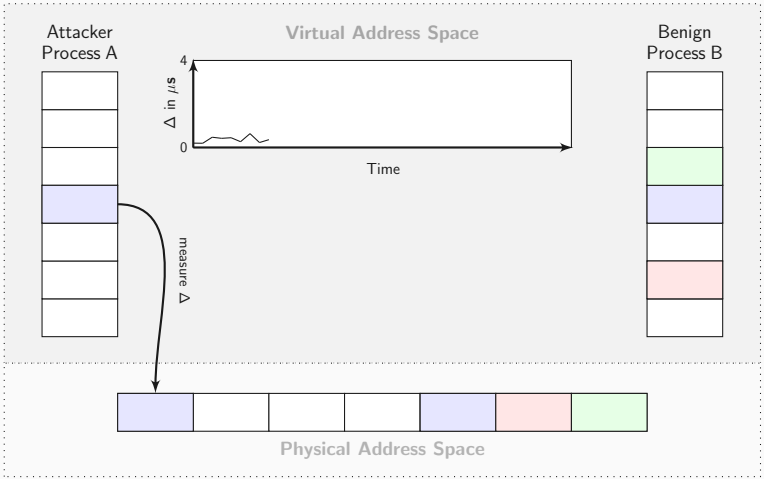
Page Deduplication Attack



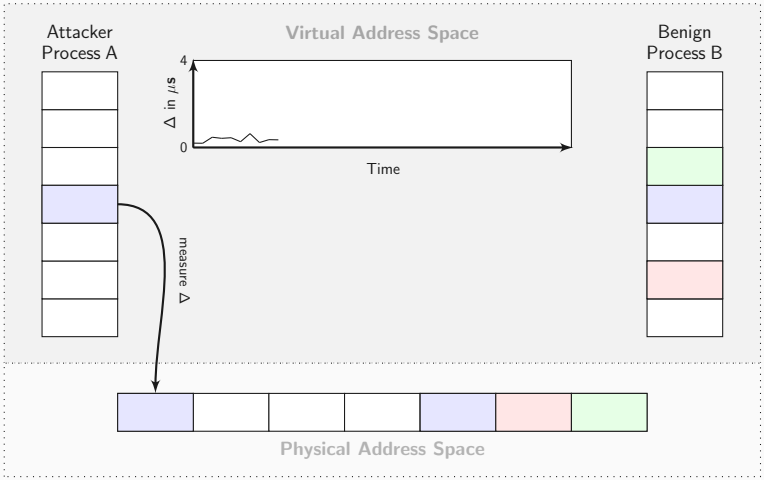
Page Deduplication Attack



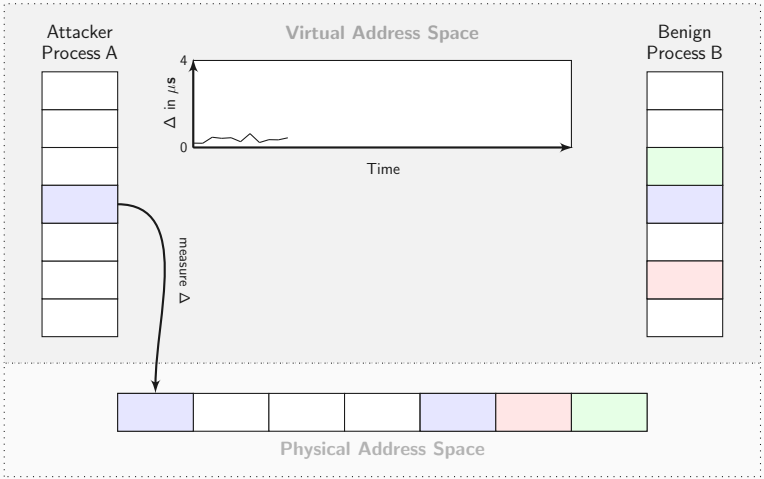
Page Deduplication Attack



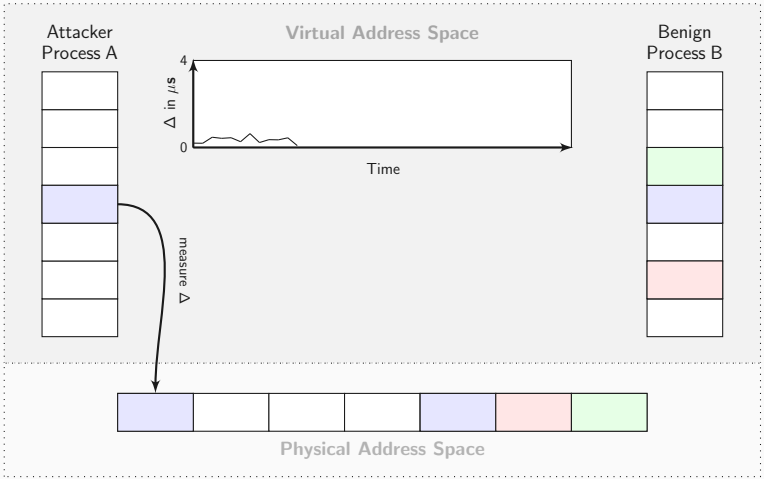
Page Deduplication Attack



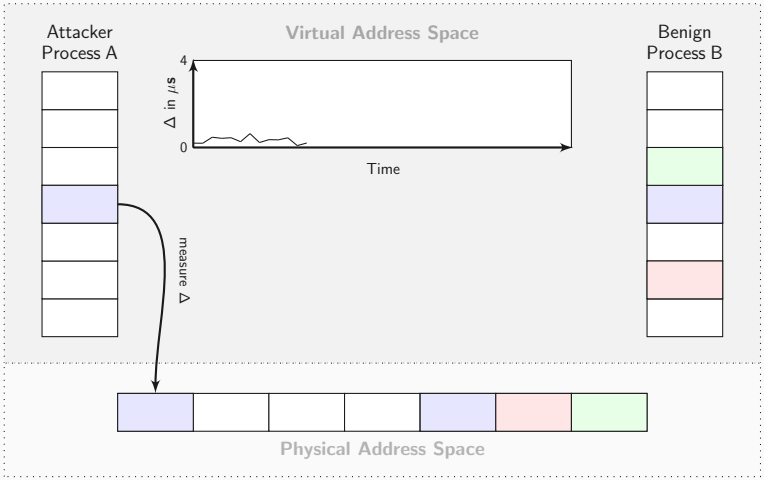
Page Deduplication Attack



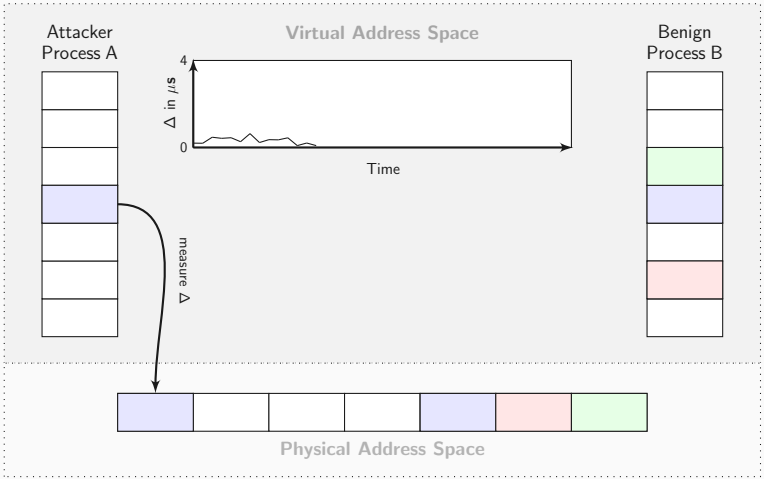
Page Deduplication Attack



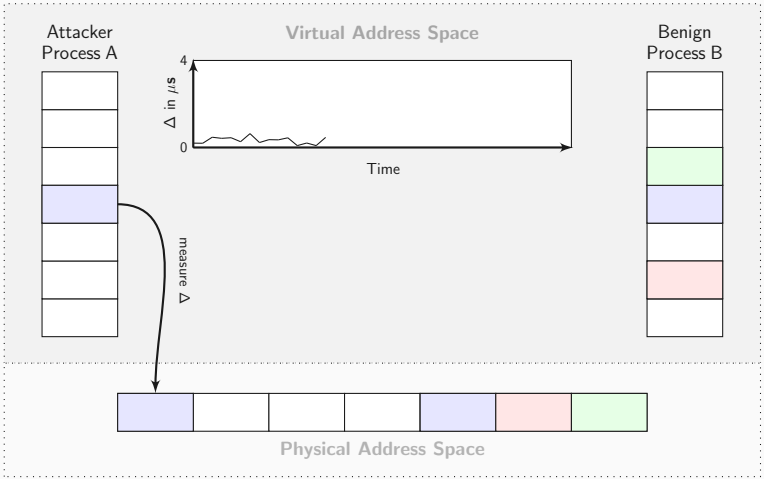
Page Deduplication Attack



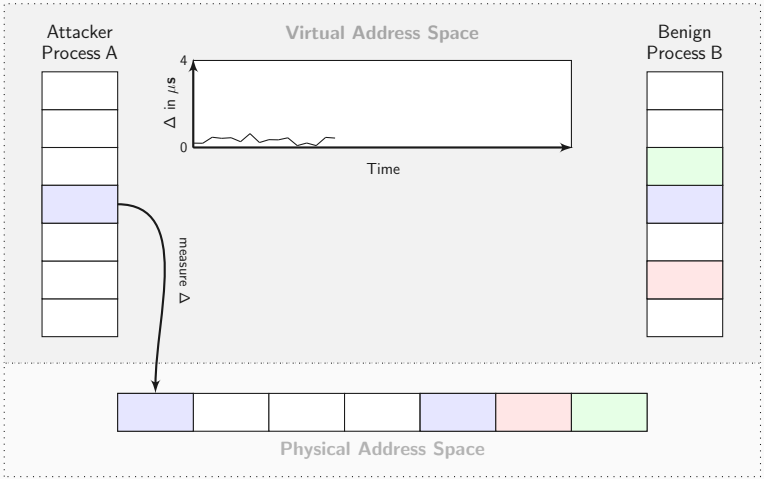
Page Deduplication Attack



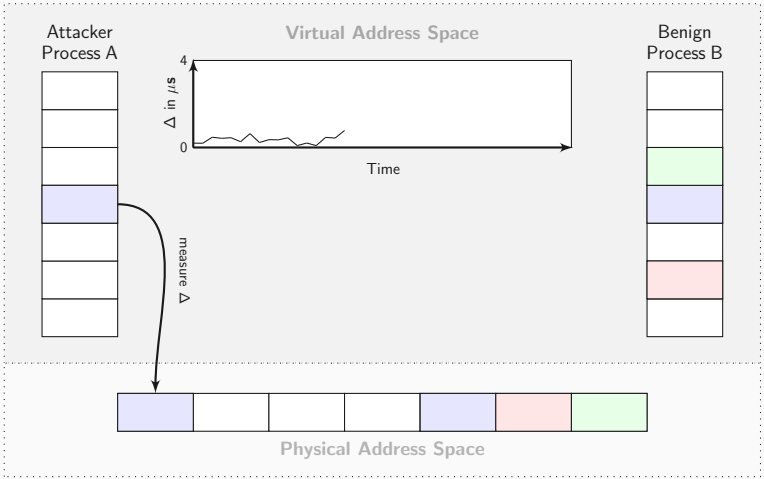
Page Deduplication Attack



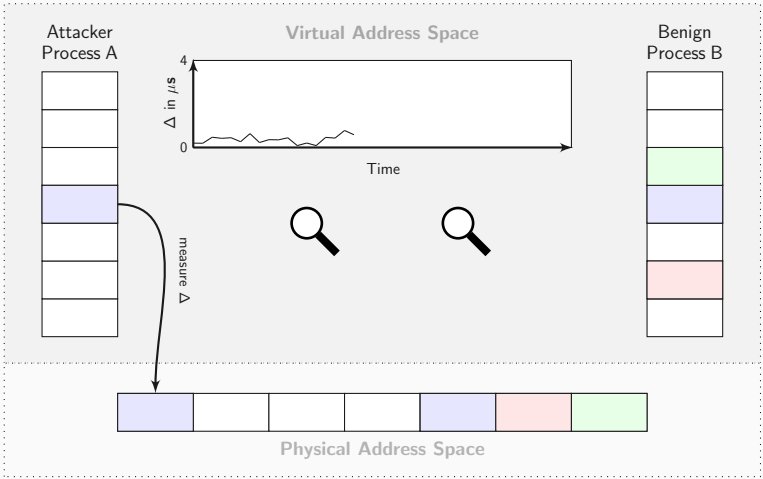
Page Deduplication Attack



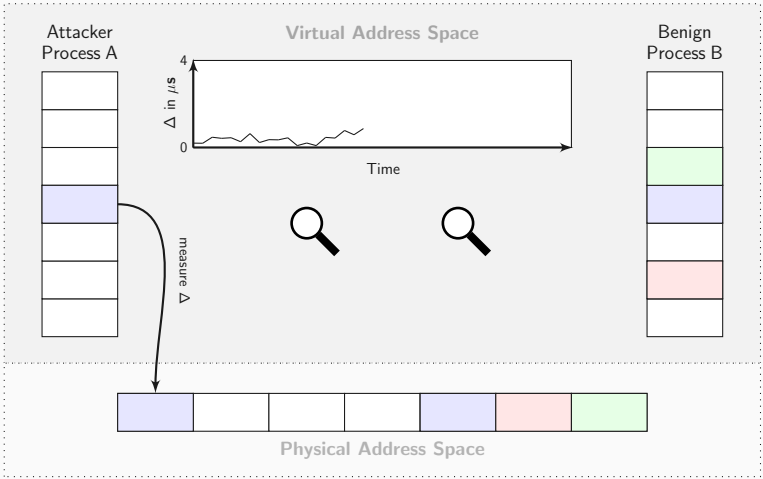
Page Deduplication Attack



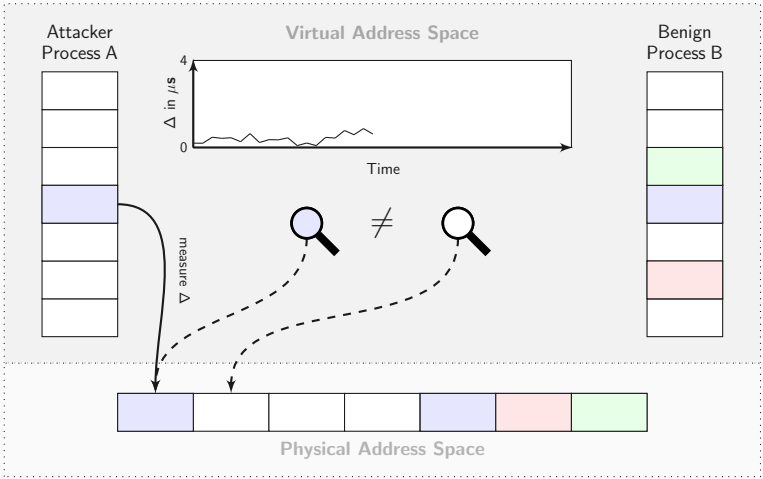
Page Deduplication Attack



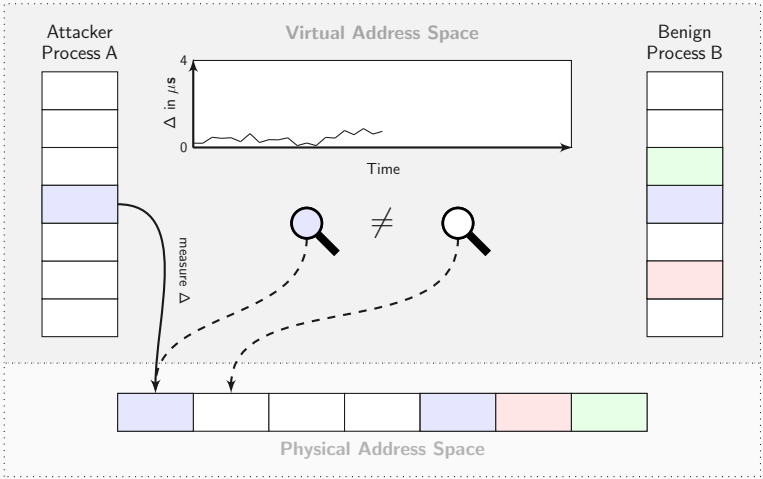
Page Deduplication Attack



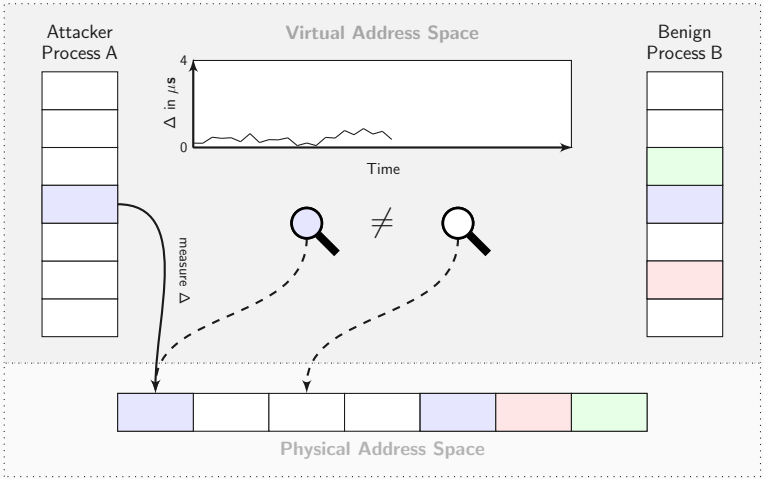
Page Deduplication Attack



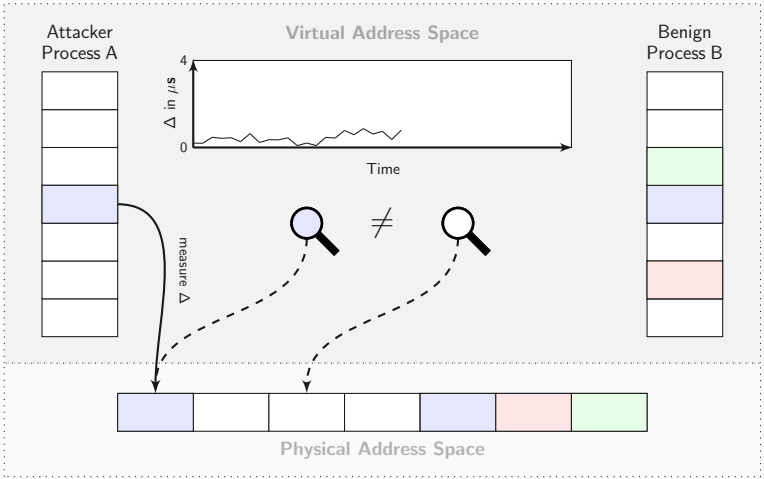
Page Deduplication Attack



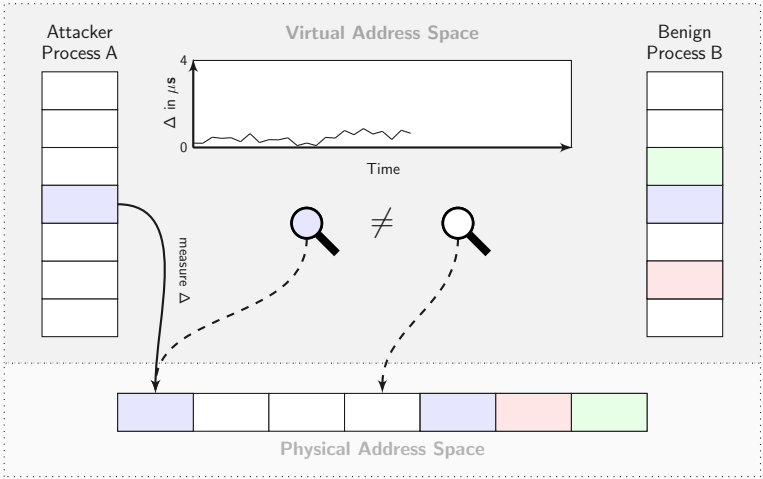
Page Deduplication Attack



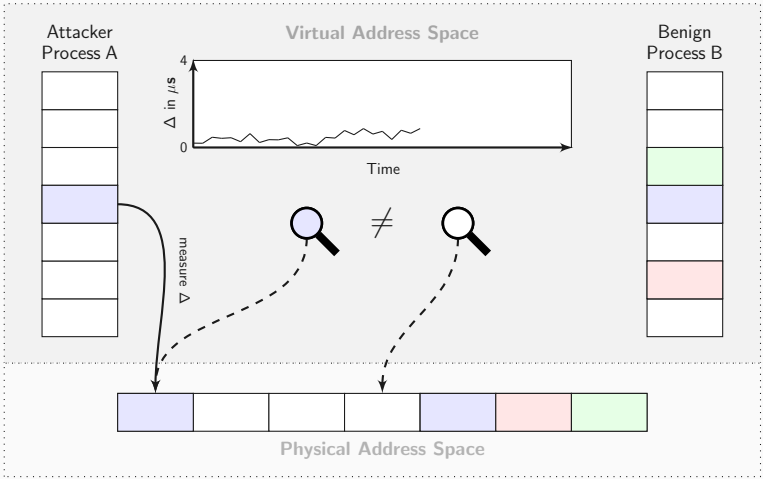
Page Deduplication Attack



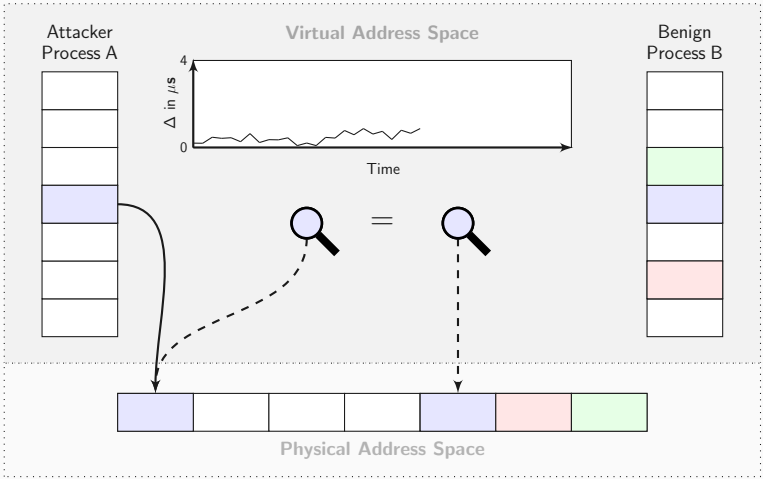
Page Deduplication Attack



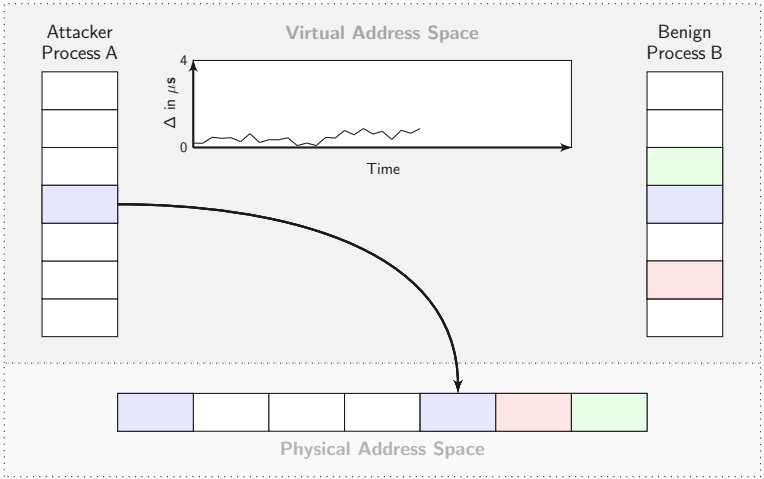
Page Deduplication Attack



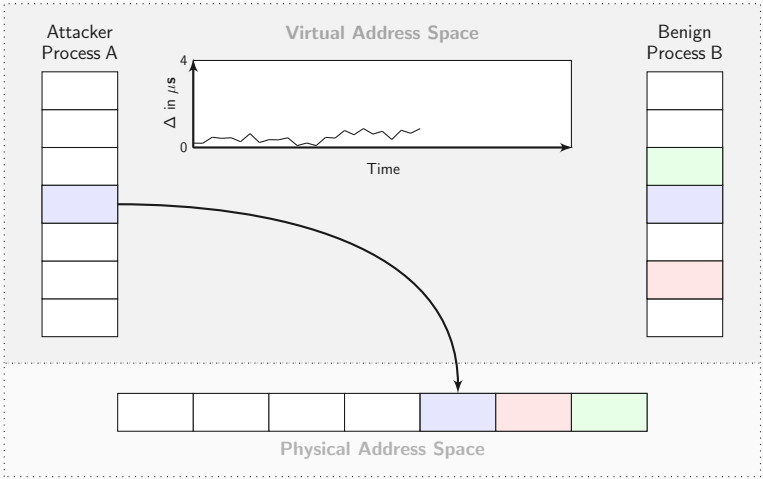
Page Deduplication Attack



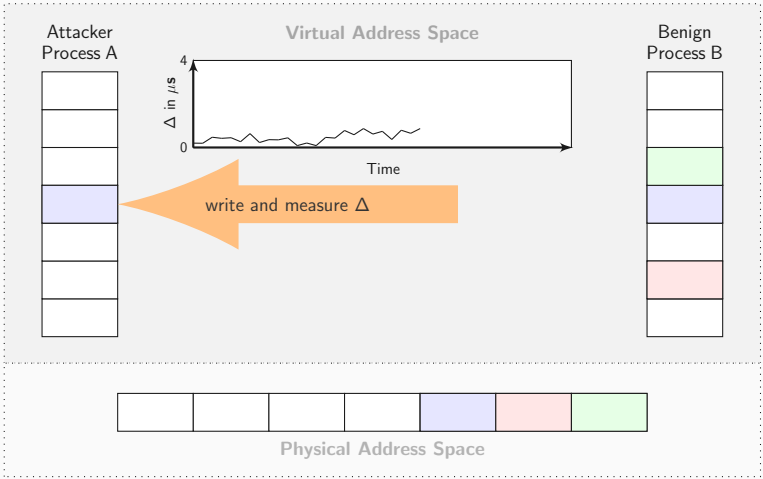
Page Deduplication Attack



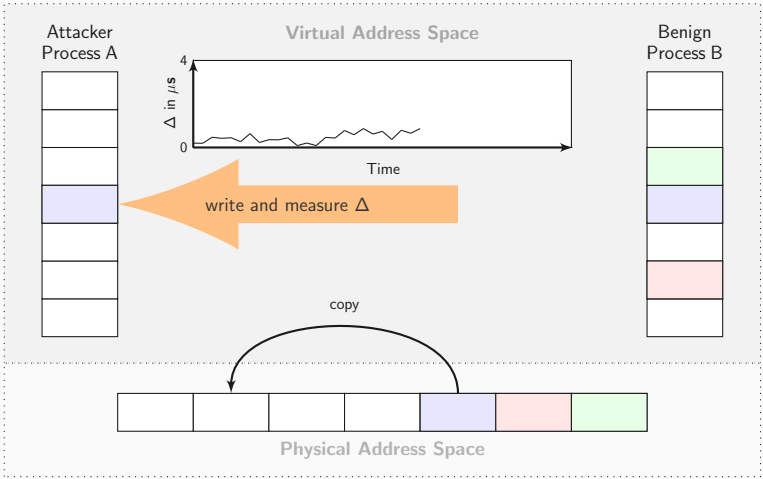
Page Deduplication Attack



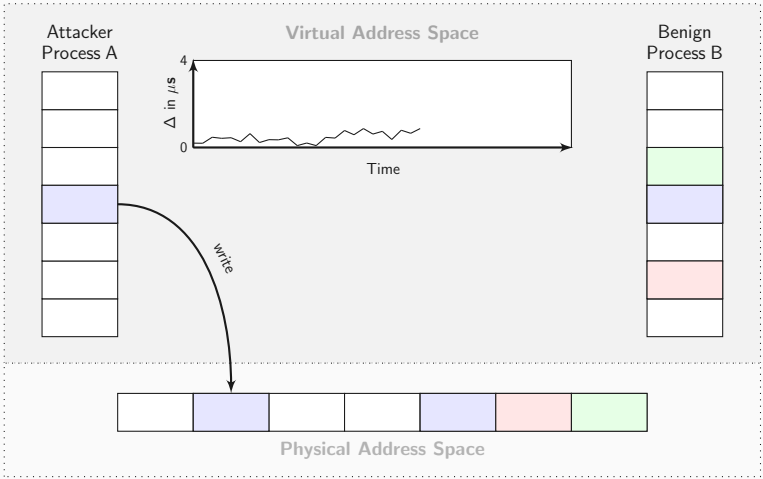
Page Deduplication Attack



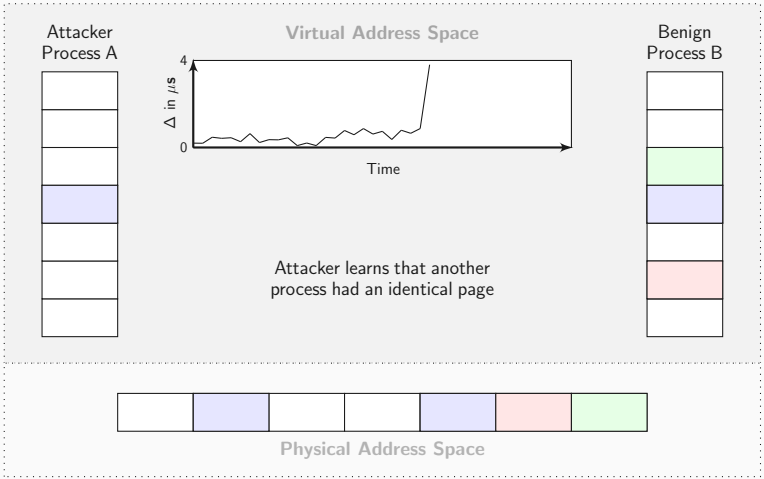
Page Deduplication Attack



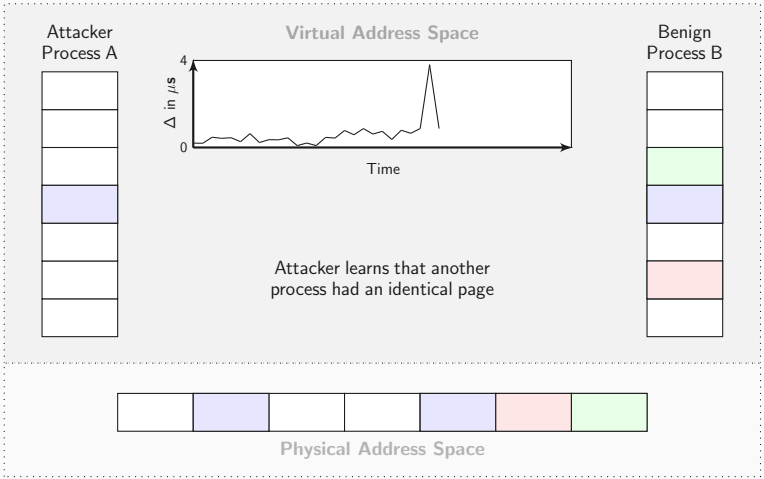
Page Deduplication Attack



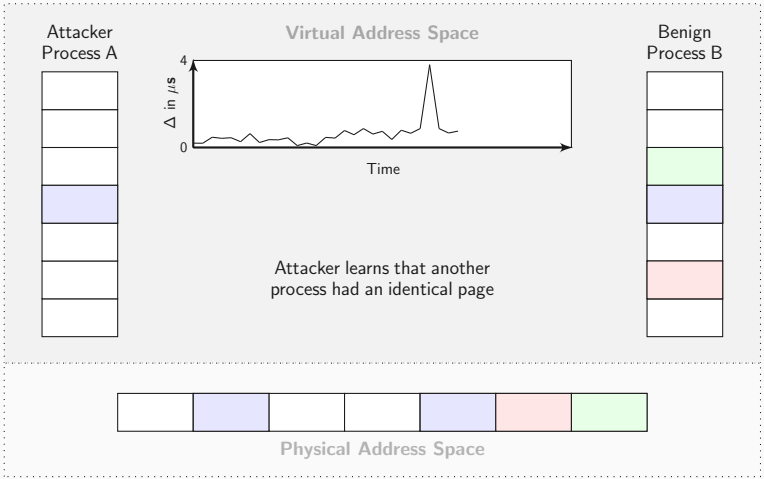
Page Deduplication Attack



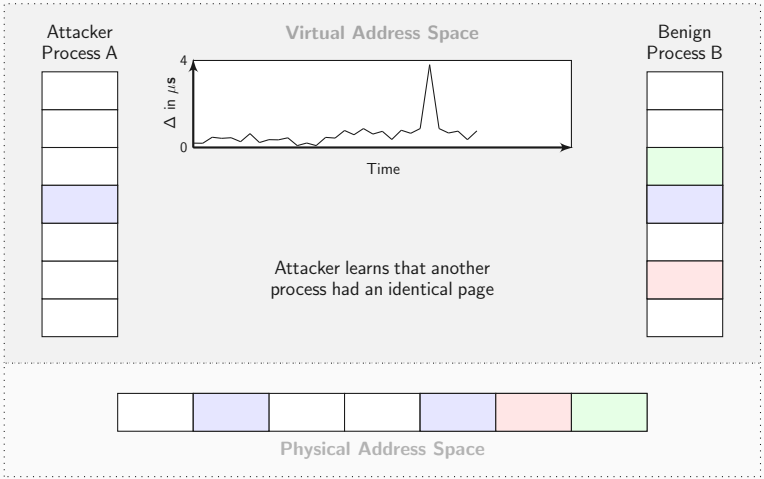
Page Deduplication Attack



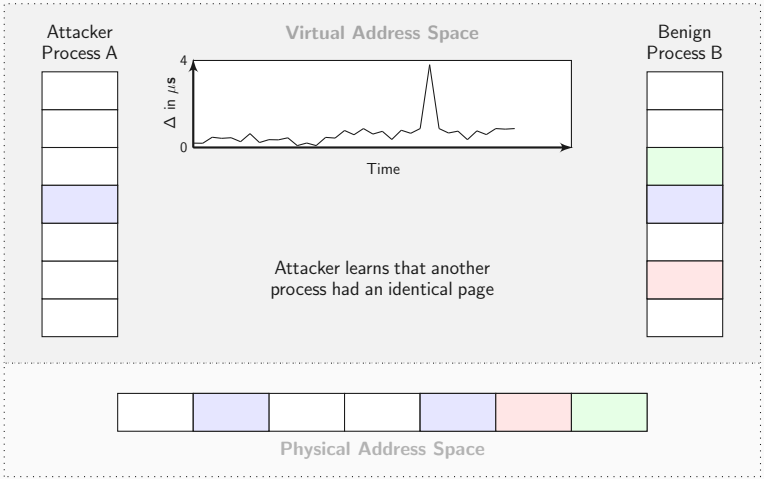
Page Deduplication Attack



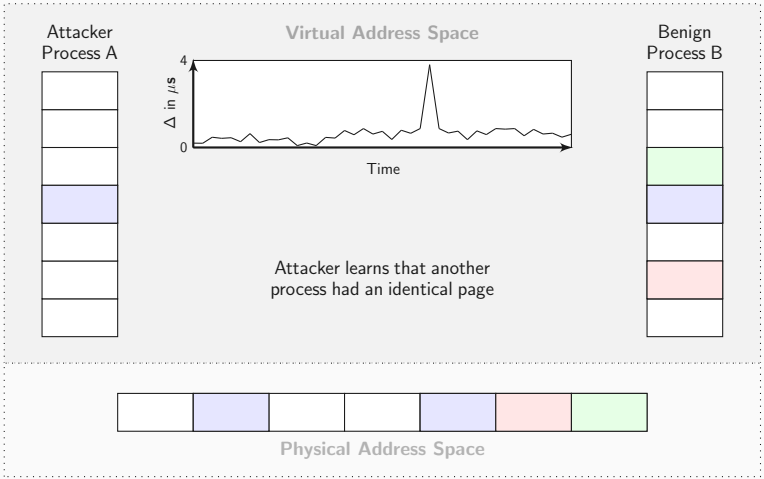
Page Deduplication Attack



Page Deduplication Attack



Page Deduplication Attack



What can be attacked?

- Detect binary versions in co-located VMs
- Detect downloaded image in Firefox under certain conditions

→ Attacks on hypervisors

- Native code only

What can be attacked?

- Detect CSS files and images of opened websites
  - Chrome, Firefox and Internet Explorer
- Perform the attack in JavaScript

$\rightarrow$ Attacks on KVM, Windows 8.1 and Android

- Images and CSS files are page-aligned in memory
- Load them into memory for all websites of interest
- Detect deduplication

→ Malicious ad networks: alternative to tracking pixels?

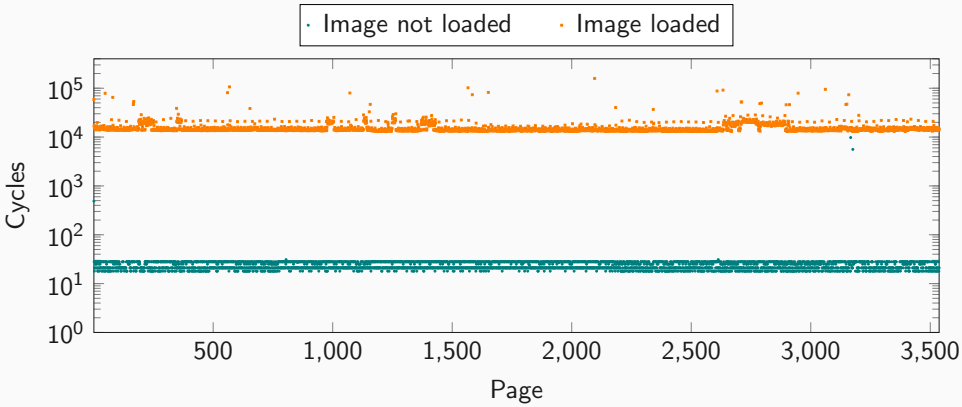


Image not loaded
Image loaded
Cycles
10^5
10^4
10^3
10^2
10^1
10^0
500
1,000
1,500
2,000
2,500
3,000
3,500
Page

- No cycle counting (`rdtsc`)
- No access to virtual addresses

- Only require microsecond accuracy
  - `performance.now()` is accurate enough
  - Can even work with millisecond accuracy
    - Accumulate time difference
    - Only possible with enough image/CSS data
- Large typed arrays are allocated page-aligned


Image not loaded
Image loaded
Nanoseconds
10^5
10^4
10^3
10^2
500
1,000
1,500
2,000
2,500
3,000
3,500
Page

- Attacker chosen set of websites
- Load website images and CSS files into arrays
- Reuse HTTP headers of system under attack

JavaScript:

- Reduce timer accuracy?
- Prevent page-aligned arrays?
- Website diversification?
- Prevent control over full pages
  - Every n-th byte not part of JavaScript array

JavaScript:

- Reduce timer accuracy?
- Prevent page-aligned arrays?
- Website diversification?
- Prevent control over full pages
  - Every n-th byte not part of JavaScript array

Generic:

- Disable page deduplication (for writable pages)

- Can we mount an attack without page deduplication?

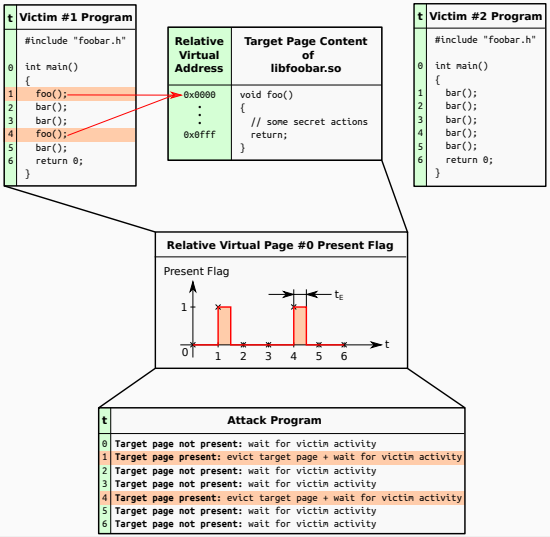
- Can we mount an attack without page deduplication?
- Shared pages are in the page cache

- Can we mount an attack without page deduplication?
- Shared pages are in the page cache
- Non-shared pages too

Where to get the signal from?

```
MINCORE(2)                Linux Programmer's Manual               MINCORE(2)

NAME        top

       mincore - determine whether pages are resident in memory

SYNOPSIS        top

       #include <unistd.h>
       #include <sys/mman.h>

       int mincore(void *addr, size_t length, unsigned char *vec);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       mincore():
           Since glibc 2.19:
                _DEFAULT_SOURCE
           Glibc 2.19 and earlier:
                _BSD_SOURCE || _SVID_SOURCE

DESCRIPTION        top

       mincore() returns a vector that indicates whether pages of the
       calling process's virtual memory are resident in core (RAM), and so
       will not cause a disk access (page fault) if referenced.  The kernel
       returns residency information about the pages starting at the address
       addr, and continuing for length bytes.
```

Attack Idea / Overview



Challenge: Flush / Evict

- No unprivileged flush system call

Challenge: Flush / Evict

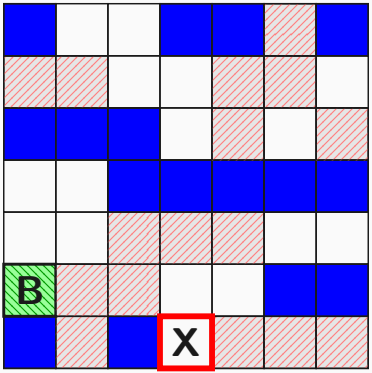
- No unprivileged flush system call
- Eviction? Well...

- No unprivileged flush system call
- Eviction? Well...
- Filling your entire memory with garbage takes long, makes your system unstable and laggy

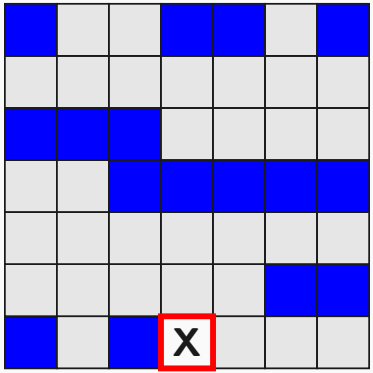
- No unprivileged flush system call
- Eviction? Well...
- Filling your entire memory with garbage takes long, makes your system unstable and laggy
- Got down to around 2–10 seconds with that approach in 2015.

- No unprivileged flush system call
- Eviction? Well...
- Filling your entire memory with garbage takes long, makes your system unstable and laggy
- Got down to around 2–10 seconds with that approach in 2015.
- Idea: use page cache pages for eviction

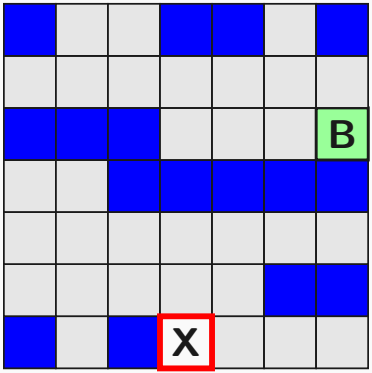
(1) Start



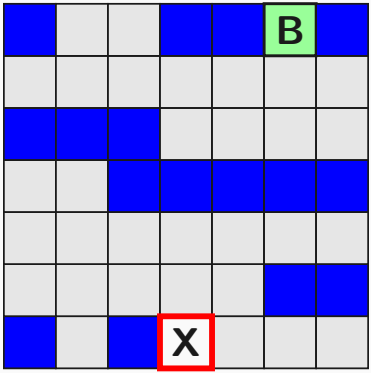
(2) Evict Page Cache



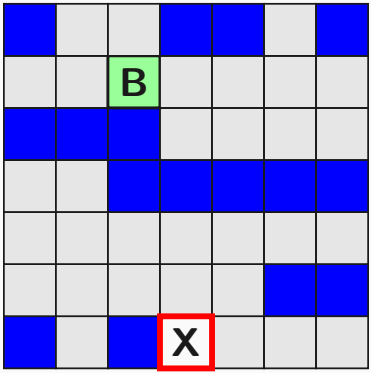
(3) Access Binary



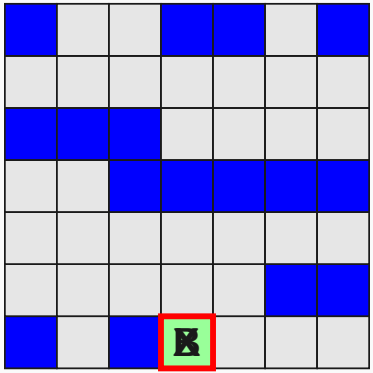
(4) Evict + Access



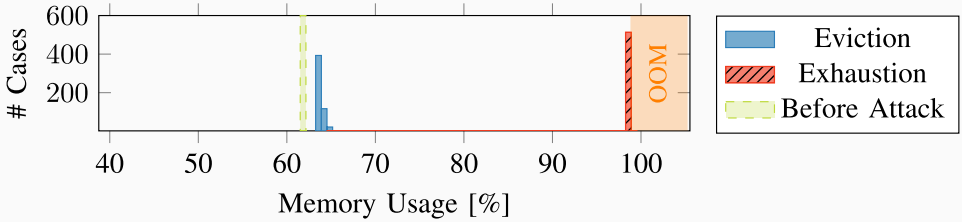
(5) Evict + Access



(6) Stop if target reached



- Great advantage over memory exhaustion: only negligible memory footprint



But...

- 2.68s for one eviction, that's still too slow for most use cases

But...

- 2.68s for one eviction, that's still too slow for most use cases
- Idea: Let's build the eviction set more cleverly

- Ideally contains all pages that are currently / most often in the page cache

- Ideally contains all pages that are currently / most often in the page cache

$\rightarrow$ Eviction will load useful pages into the page cache + not evict these

- Ideally contains all pages that are currently / most often in the page cache

$\rightarrow$ Eviction will load useful pages into the page cache + not evict these

- Add some pages which are not needed + rarely accessed $\rightarrow$ evict these first

- Ideally contains all pages that are currently / most often in the page cache

→ Eviction will load useful pages into the page cache + not evict these

- Add some pages which are not needed + rarely accessed → evict these first
- Add memory pressure to reduce the page cache size for faster eviction

- Ideally contains all pages that are currently / most often in the page cache

→ Eviction will load useful pages into the page cache + not evict these

- Add some pages which are not needed + rarely accessed → evict these first
- Add memory pressure to reduce the page cache size for faster eviction

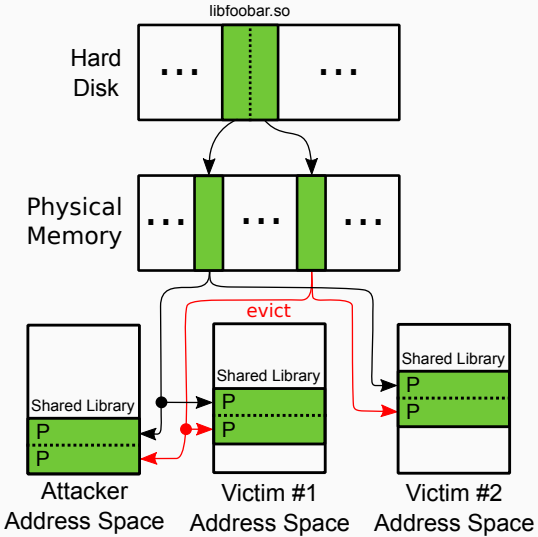
- Ideally contains all pages that are currently / most often in the page cache

→ Eviction will load useful pages into the page cache + not evict these

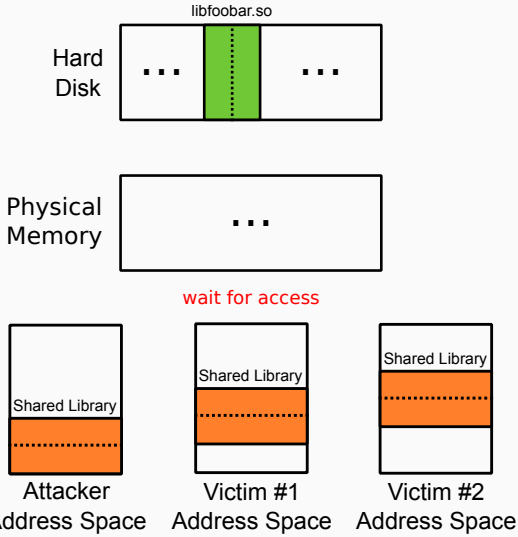
- Add some pages which are not needed + rarely accessed → evict these first
- Add memory pressure to reduce the page cache size for faster eviction

→ 149 ms for one eviction

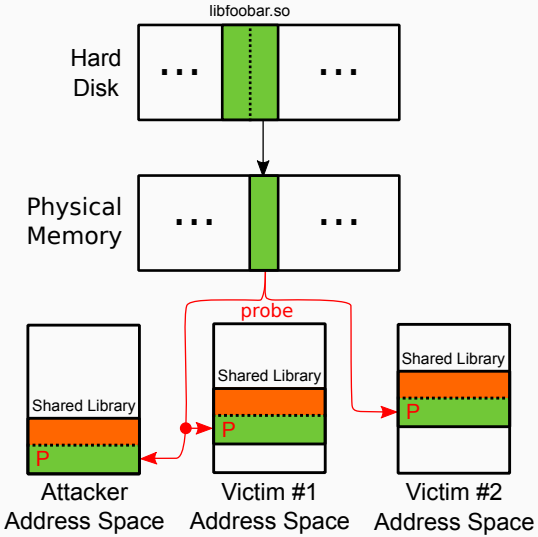
Attack Overview



Attack Overview



Attack Overview



What about Windows?

- `QueryWorkingSetEx` instead of mincore

What about Windows?

- `QueryWorkingSetEx` instead of mincore
  - `PROCESS_QUERY_LIMITED_INFORMATION`? → can even attack non-shared pages (heap, stack, etc.)

What about Windows?

- `QueryWorkingSetEx` instead of mincore
  - `PROCESS_QUERY_LIMITED_INFORMATION`? → can even attack non-shared pages (heap, stack, etc.)
  - ShareCount → requires no permissions + works on shared pages

What about Windows?

- `QueryWorkingSetEx` instead of mincore
  - `PROCESS_QUERY_LIMITED_INFORMATION`? → can even attack non-shared pages (heap, stack, etc.)
  - ShareCount → requires no permissions + works on shared pages
- Page cache eviction really inefficient

What about Windows?

- `QueryWorkingSetEx` instead of mincore
  - `PROCESS_QUERY_LIMITED_INFORMATION`? → can even attack non-shared pages (heap, stack, etc.)
  - ShareCount → requires no permissions + works on shared pages
- Page cache eviction really inefficient
- Working set eviction isntead:

What about Windows?

- `QueryWorkingSetEx` instead of mincore
  - `PROCESS_QUERY_LIMITED_INFORMATION`? → can even attack non-shared pages (heap, stack, etc.)
  - ShareCount → requires no permissions + works on shared pages
- Page cache eviction really inefficient
- Working set eviction isntead:
  - `VirtualUnlock` flushes pages

What about Windows?

- `QueryWorkingSetEx` instead of mincore
  - `PROCESS_QUERY_LIMITED_INFORMATION`? → can even attack non-shared pages (heap, stack, etc.)
  - ShareCount → requires no permissions + works on shared pages
- Page cache eviction really inefficient
- Working set eviction isntead:
  - `VirtualUnlock` flushes pages
  - `SetProcessWorkingSetSize` to a minimum (52 KB) + self-eviction in victim

What about Windows?

- `QueryWorkingSetEx` instead of mincore
  - `PROCESS_QUERY_LIMITED_INFORMATION`? → can even attack non-shared pages (heap, stack, etc.)
  - ShareCount → requires no permissions + works on shared pages
- Page cache eviction really inefficient
- Working set eviction isntead:
  - `VirtualUnlock` flushes pages
  - `SetProcessWorkingSetSize` to a minimum (52 KB) + self-eviction in victim

What about Windows?

- `QueryWorkingSetEx` instead of mincore
  - `PROCESS_QUERY_LIMITED_INFORMATION`? → can even attack non-shared pages (heap, stack, etc.)
  - ShareCount → requires no permissions + works on shared pages
- Page cache eviction really inefficient
- Working set eviction isntead:
  - `VirtualUnlock` flushes pages
  - `SetProcessWorkingSetSize` to a minimum (52 KB) + self-eviction in victim

→ 4.48 ms for one eviction

- Cross-container / cross-sandbox covert channel:

Attack Examples

- Cross-container / cross-sandbox covert channel:
  - 7 KB/s on Linux

Attack Examples

- Cross-container / cross-sandbox covert channel:
  - 7 KB/s on Linux
  - 273 KB/s on Windows 10

Attack Examples

- Cross-container / cross-sandbox covert channel:
  - 7 KB/s on Linux
  - 273 KB/s on Windows 10
- ASLR break: $<$ 1 minute to locate stack, heap, and binary (in other process on Windows)

Attack Examples

- Cross-container / cross-sandbox covert channel:
  - 7 KB/s on Linux
  - 273 KB/s on Windows 10
- ASLR break: $<$ 1 minute to locate stack, heap, and binary (in other process on Windows)
- Observe when user plays a video in Firefox, distinguish codecs (and by that sites), when a PNG is rendered, various user input

Attack Examples

- Cross-container / cross-sandbox covert channel:
  - 7 KB/s on Linux
  - 273 KB/s on Windows 10
- ASLR break: $<$ 1 minute to locate stack, heap, and binary (in other process on Windows)
- Observe when user plays a video in Firefox, distinguish codecs (and by that sites), when a PNG is rendered, various user input
- UI redress attack

Attack Examples

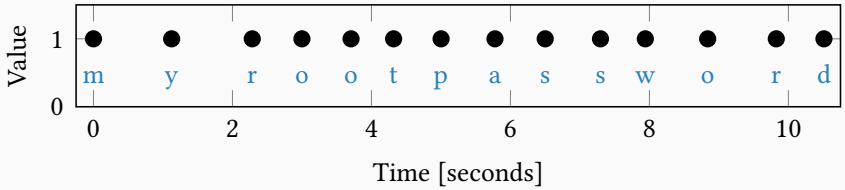
- Cross-container / cross-sandbox covert channel:
  - 7 KB/s on Linux
  - 273 KB/s on Windows 10
- ASLR break: $<$ 1 minute to locate stack, heap, and binary (in other process on Windows)
- Observe when user plays a video in Firefox, distinguish codecs (and by that sites), when a PNG is rendered, various user input
- UI redress attack
- Keystroke timing attack

Attack Examples

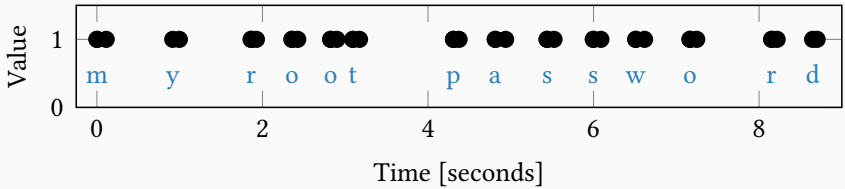
- Cross-container / cross-sandbox covert channel:
  - 7 KB/s on Linux
  - 273 KB/s on Windows 10
- ASLR break: $<$ 1 minute to locate stack, heap, and binary (in other process on Windows)
- Observe when user plays a video in Firefox, distinguish codecs (and by that sites), when a PNG is rendered, various user input
- UI redress attack
- Keystroke timing attack
- PHP password generation

Attack Examples

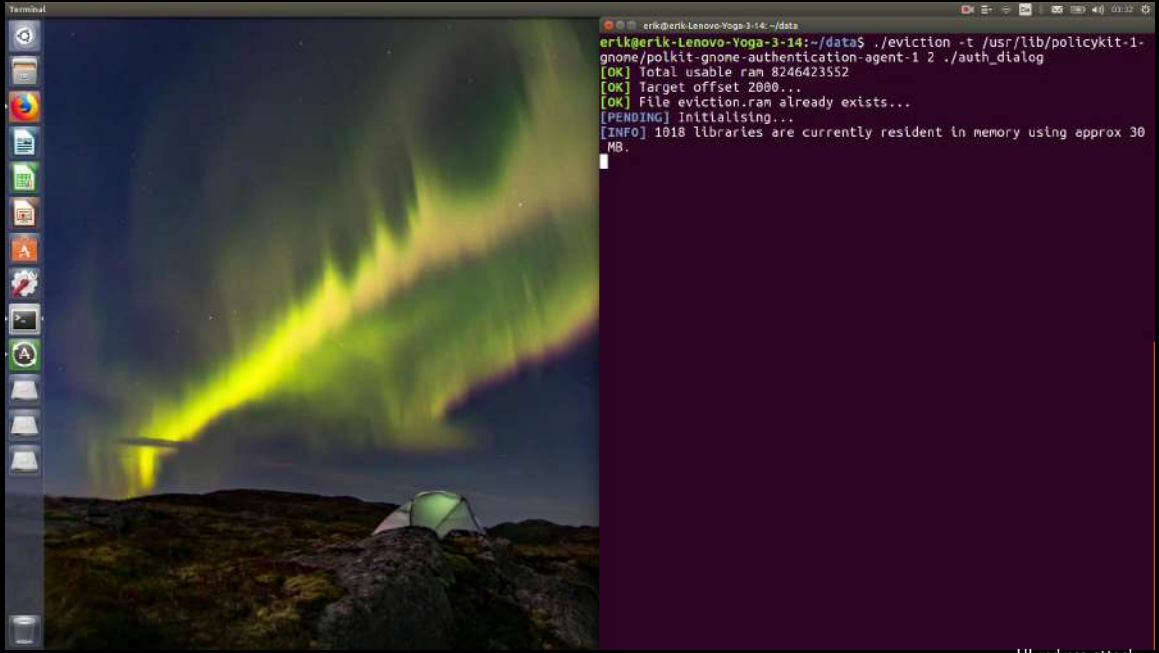
- Cross-container / cross-sandbox covert channel:
  - 7 KB/s on Linux
  - 273 KB/s on Windows 10
- ASLR break: $<$ 1 minute to locate stack, heap, and binary (in other process on Windows)
- Observe when user plays a video in Firefox, distinguish codecs (and by that sites), when a PNG is rendered, various user input
- UI redress attack
- Keystroke timing attack
- PHP password generation
- Oracle attacks



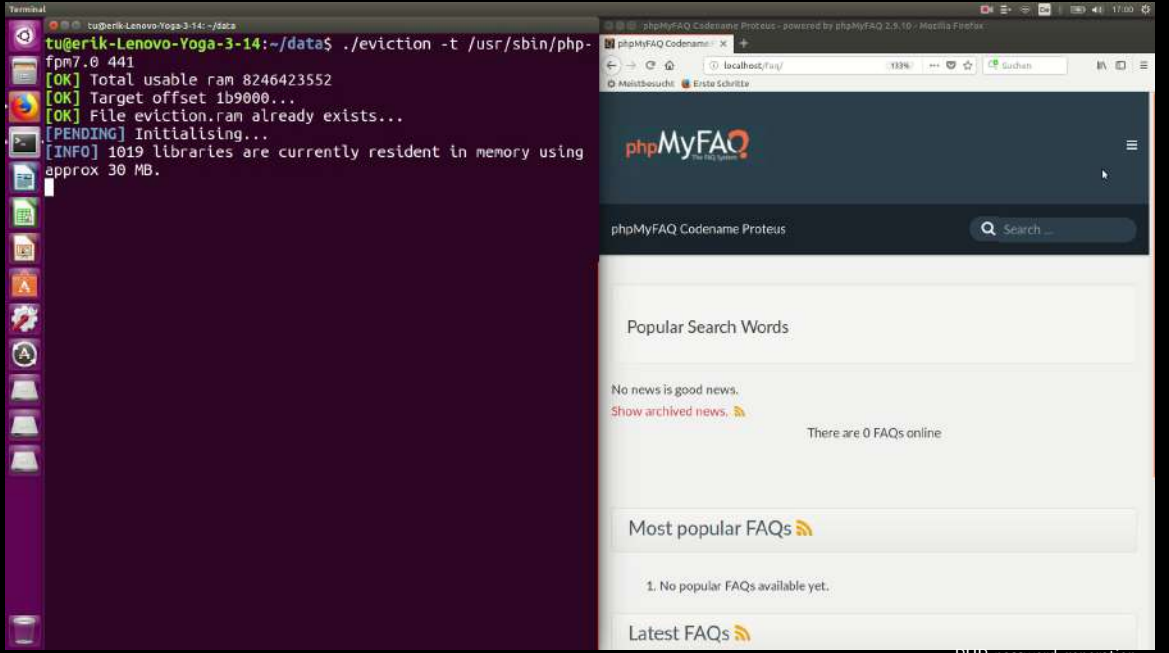
(Linux)



(Windows)



UI redress attack


Terminal
tu@erik-Lenovo-Yoga-3-14: ~/data
tu@erik-Lenovo-Yoga-3-14:~/data$ ./eviction -t /usr/sbin/php-fpm7.0 441
[OK] Total usable ram 8246423552
[OK] Target offset 1b9000...
[OK] File eviction.ram already exists...
[PENDING] Initialising...
[INFO] 1019 libraries are currently resident in memory using approx 30 MB.
phpMyFAQ Codename Proteus - powered by phpMyFAQ 2.9.10 - Mozilla Firefox
phpMyFAQ Codename
localhost/faq/
Meistbesucht
Erste Schritte
phpMyFAQ
phpMyFAQ Codename Proteus
Search ...
Popular Search Words
No news is good news.
Show archived news.
There are 0 FAQs online
Most popular FAQs
1. No popular FAQs available yet.
Latest FAQs

- generic cache attacks also exist without hardware caches

Conclusions

- generic cache attacks also exist without hardware caches
- potentially better suited for malware (hardware-agnostic)

Conclusions

- generic cache attacks also exist without hardware caches
- potentially better suited for malware (hardware-agnostic)
- difficult to mitigate entirely

Microarchitectural Attacks and Beyond

Daniel Gruss

February 21, 2019

Graz University of Technology