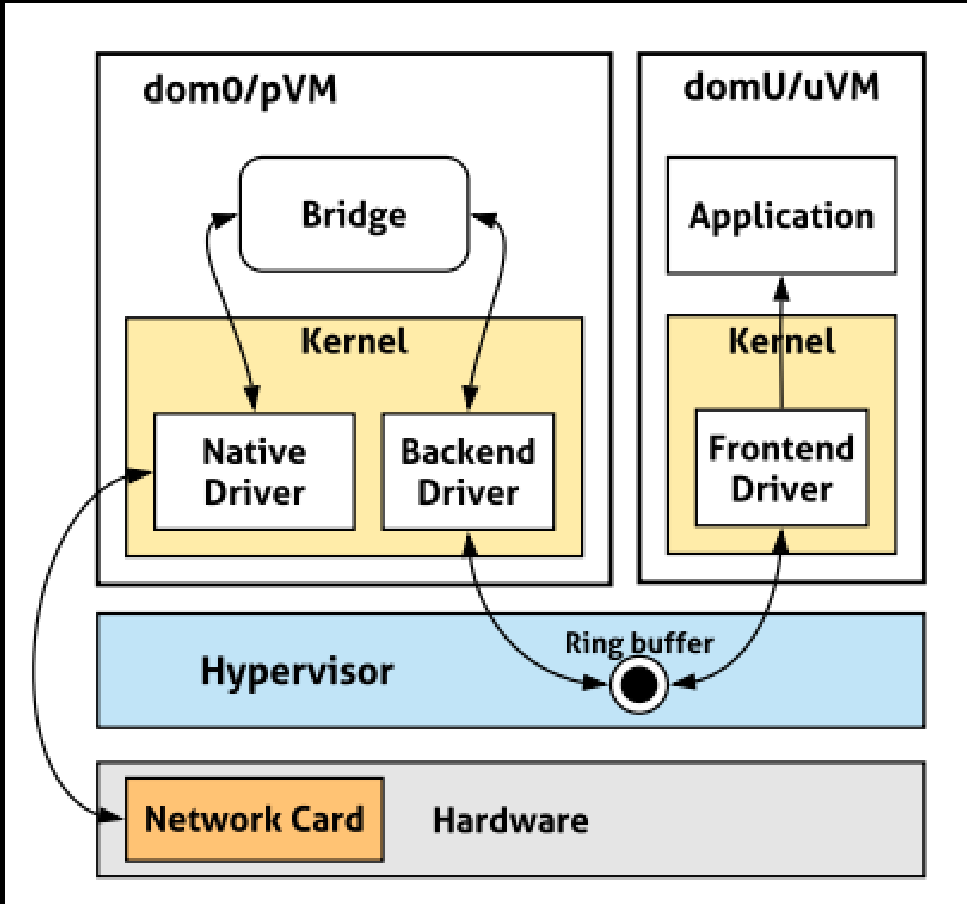
Revisiting memory to coin security

Djob Mvondo

Virtualization infrastructure

The **split driver model** is often used:
Frontend/Backend + Ring buffer idea



(a) Xen hypervisor

Résilience et dimensionnement des environnement virtualisés. Djob Mvondo. Dec 2020

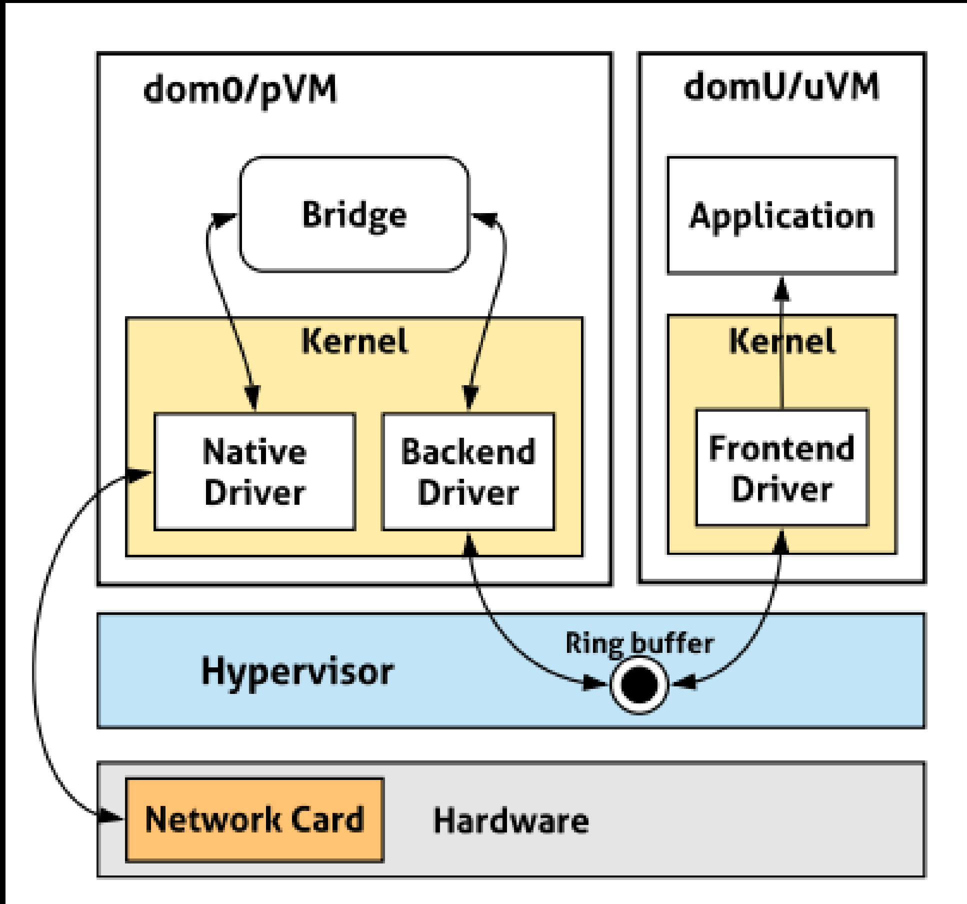
Virtualization infrastructure

The **split driver model** is often used:
Frontend/Backend + Ring buffer idea

Modularity

Performance

Existing code reuse



(a) Xen hypervisor

Résilience et dimensionnement des environnement virtualisés. Djob Mvondo. Dec 2020

Virtualization infrastructure

The **split driver model** is often used:
Frontend/Backend + Ring buffer idea

Modularity

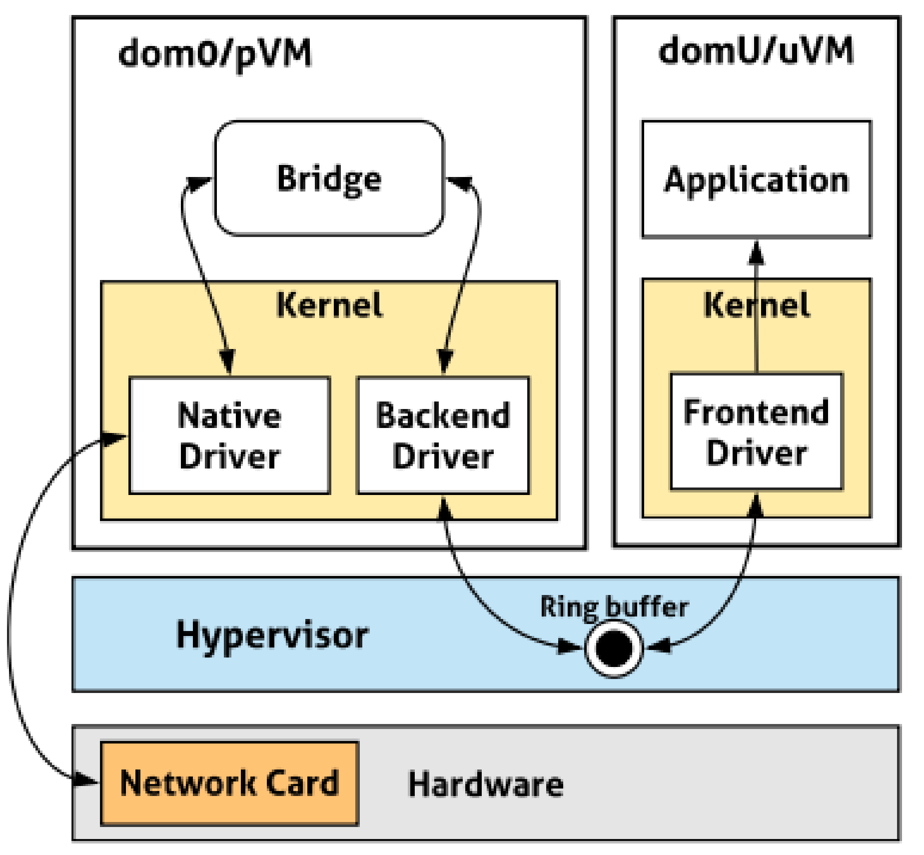
Performance

Existing code reuse

Single point of failure and bottleneck for the pVM

Bottleneck on the backend driver

Memory issues regarding ring buffers



(a) Xen hypervisor

Résilience et dimensionnement des environnement virtualisés. Djob Mvondo. Dec 2020

Single point of failure and bottleneck illustration

The **split driver model** is often used:
Frontend/Backend + Ring buffer idea

Modularity

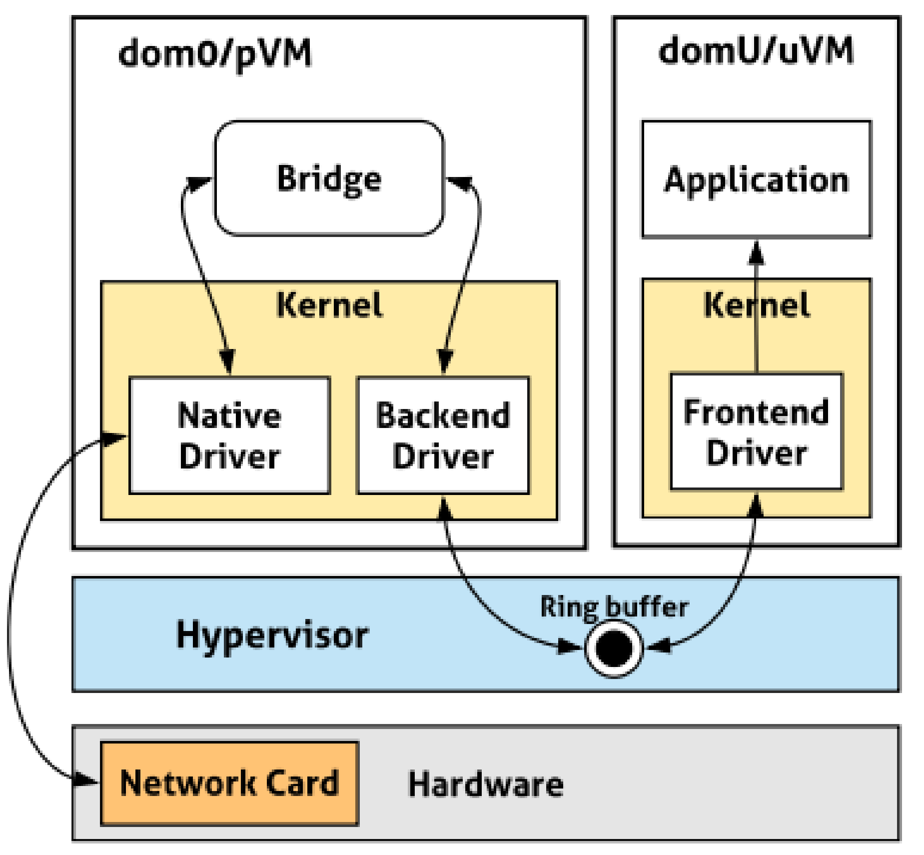
Performance

Existing code reuse

Single point of failure and bottleneck for the pVM

Bottleneck on the backend driver

Memory issues regarding ring buffers



(a) Xen hypervisor

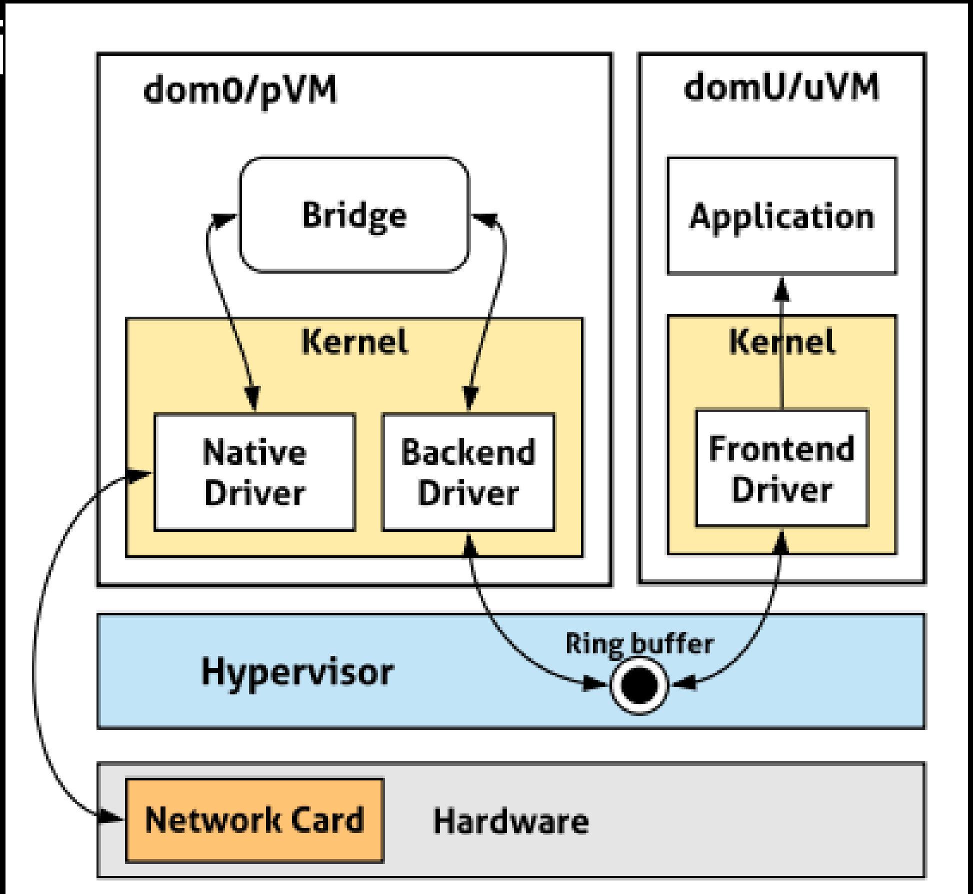
Résilience et dimensionnement des environnement virtualisés. Djob Mvondo. Dec 2020

Mitigating single point of failures

The key idea is to decompose the single poi
failure to reduce the **blast radius** in case of
problems.

Full replication[1]: *Replicate virtualized components across the data center*

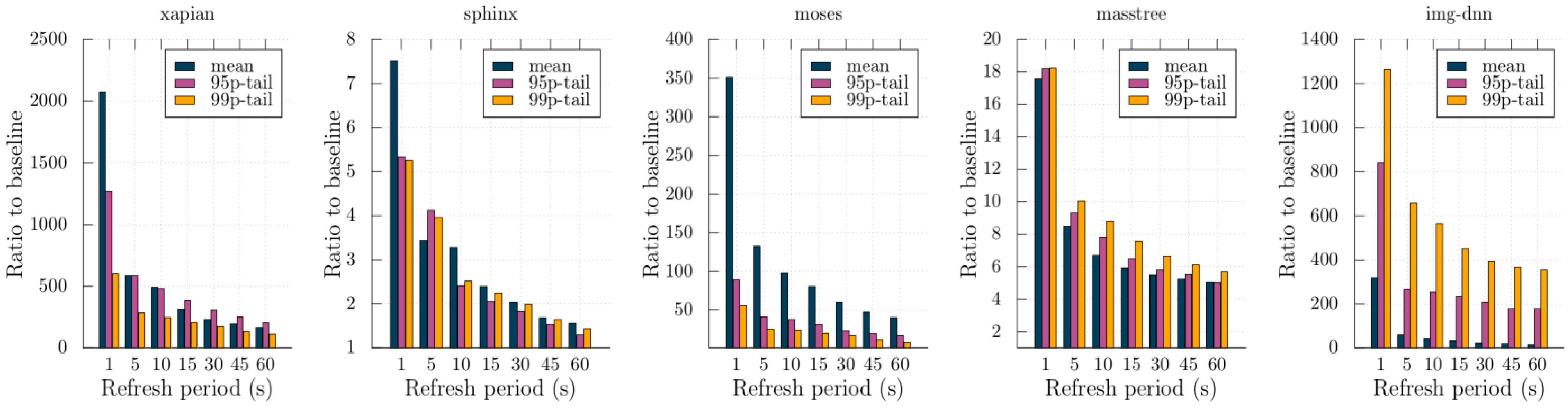
- *Resource consuming*
- *Synchronization across the different replicas*



(a) Xen hypervisor

[1] https://nutanixbible.com

Mitigating single point of failures



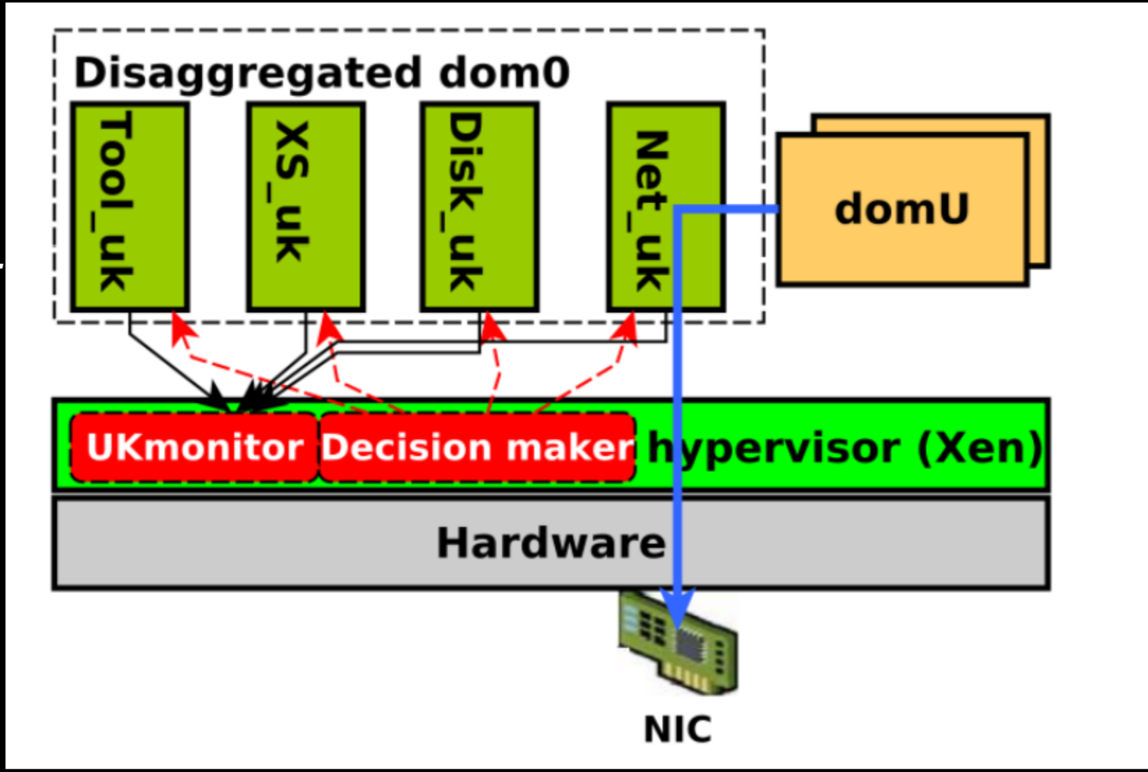
Djob Mvondo et al. Fine-Grained Fault Tolerance For Resilient pVM-based Virtual Machine Monitors. DSN'20

[2] Colp et al. Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor. SOSP’11

Mitigating single point of failures

The key idea is to decompose the single point of failure to reduce the **blast radius** in case of problems.

Disaggregation + Specialization + Pro-activity: *Reuse Xoar idea without the periodic reboot but introduce a tailored monitoring and recovery mechanism for each sub-component.*

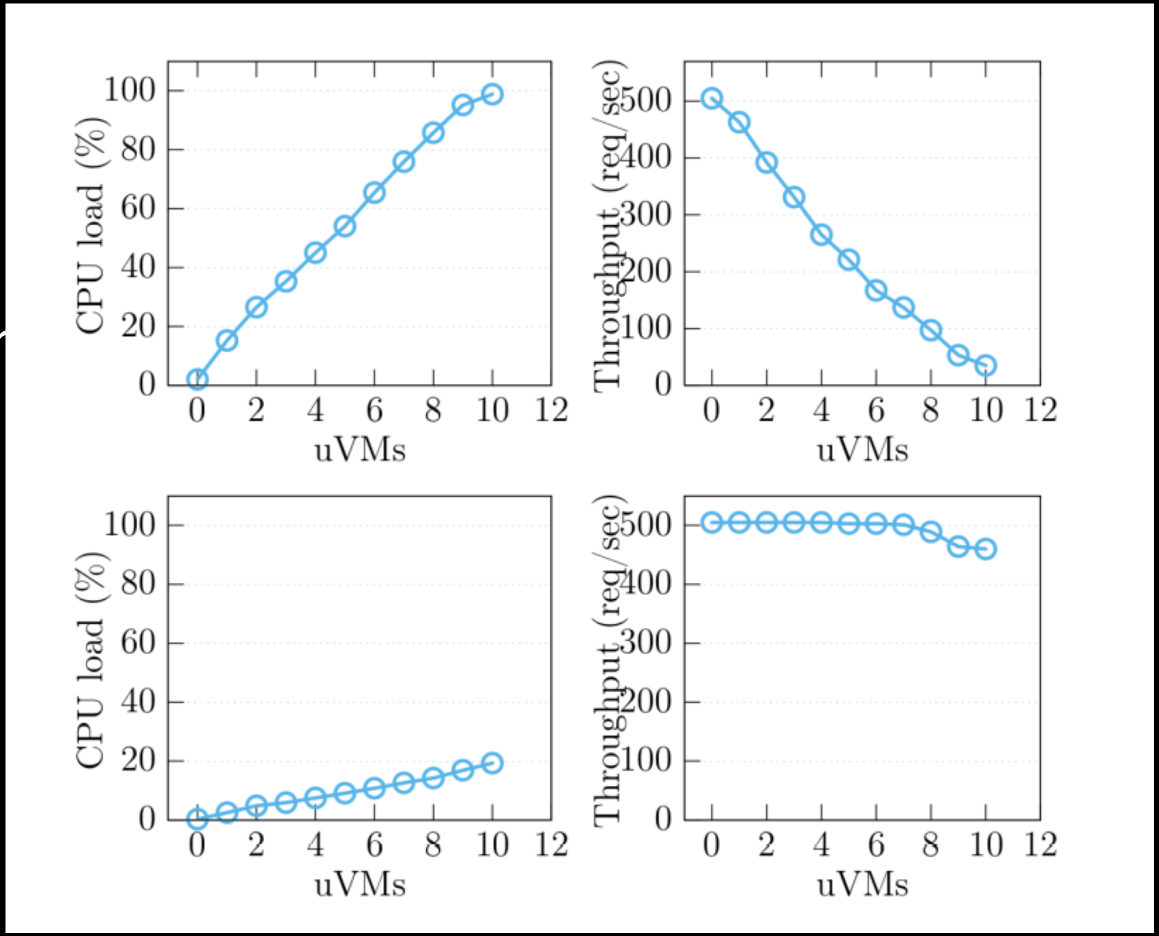


[1] Mike Swift et al. Recovering Device Drivers. OSDI'04

[2] Djob Mvondo et al. Fine-Grained Fault Tolerance For Resilient pVM-based Virtual Machine Monitors. DSN'20

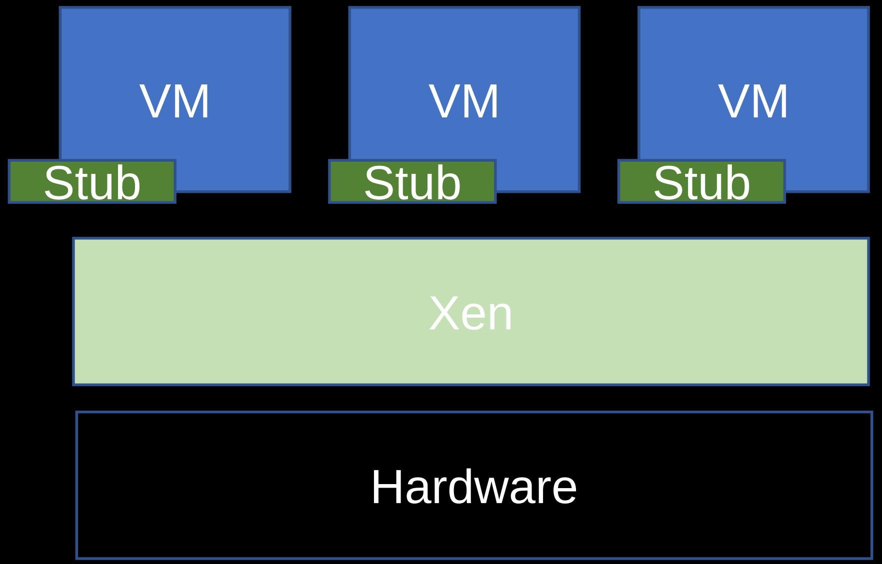
Mitigating bottlenecks

Bottlenecks can cause degradation on **application performance** and affect **response times**.



Djob Mvondo et al. Closer: A new design principle for the privileged virtual machine OS. MASCOTS 2019

Mitigating bottlenecks



Bottlenecks are mitigated by trying to **reduce the load** on the target component when **input load increases.**

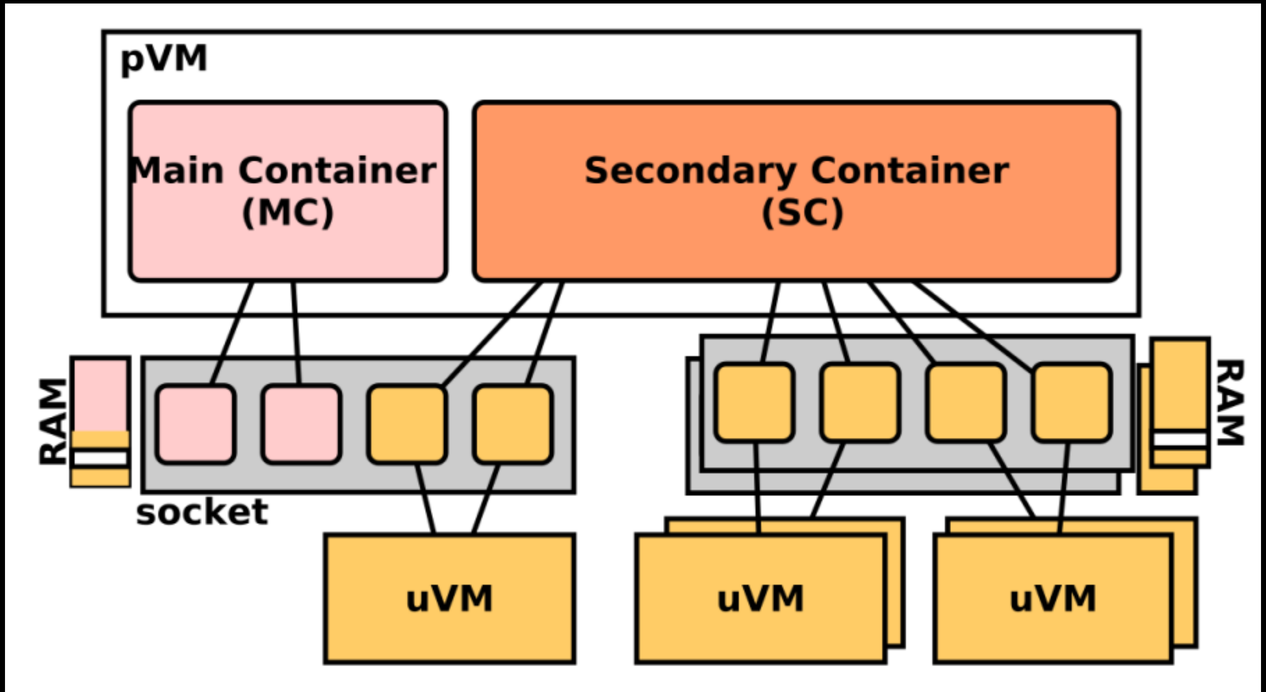
Stub-domains[1]: *Dedicate a specific component for each VM responsible to only help that VM.*

- *Quid of resource provisioning and positioning ?*

[1] Xen studdomains: https://xenproject.org

Mitigating bottlenecks

Bottlenecks are mitigated by trying **reduce the load** on the target component when **input load increases.**

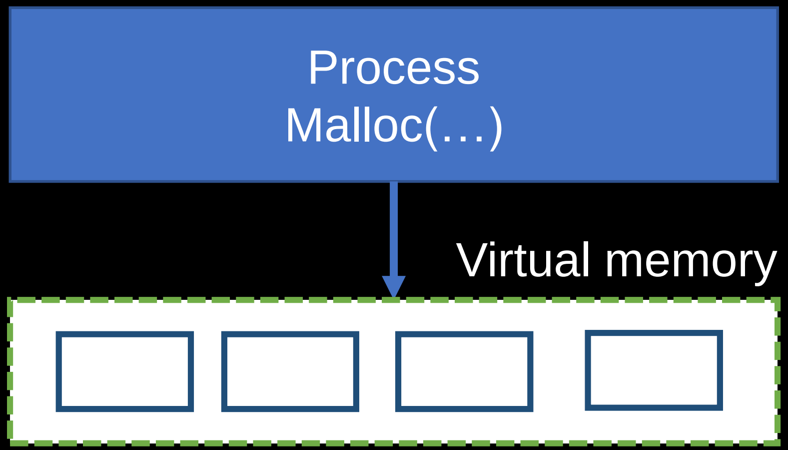


***Closer principle[1]:** Stubdomains provisioned automatically on VM allocated resources leaving out administration tasks.*

[1] Djob Mvondo et al. Closer: A new design principle for the privileged virtual machine OS. MASCOTS 2019

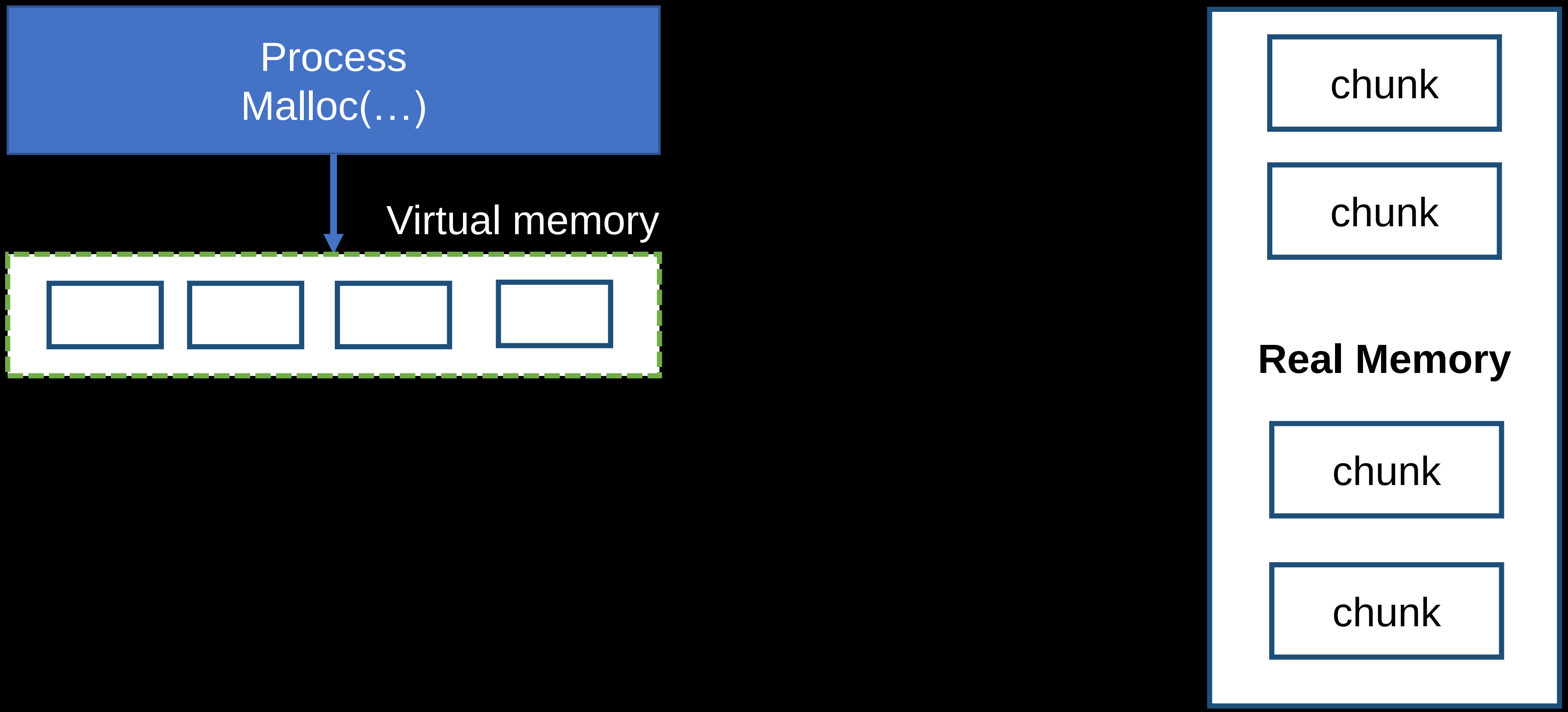
Memory issues: Corruption - access

To understand memory issues, we should understand how memory is allocated to a process/VM.



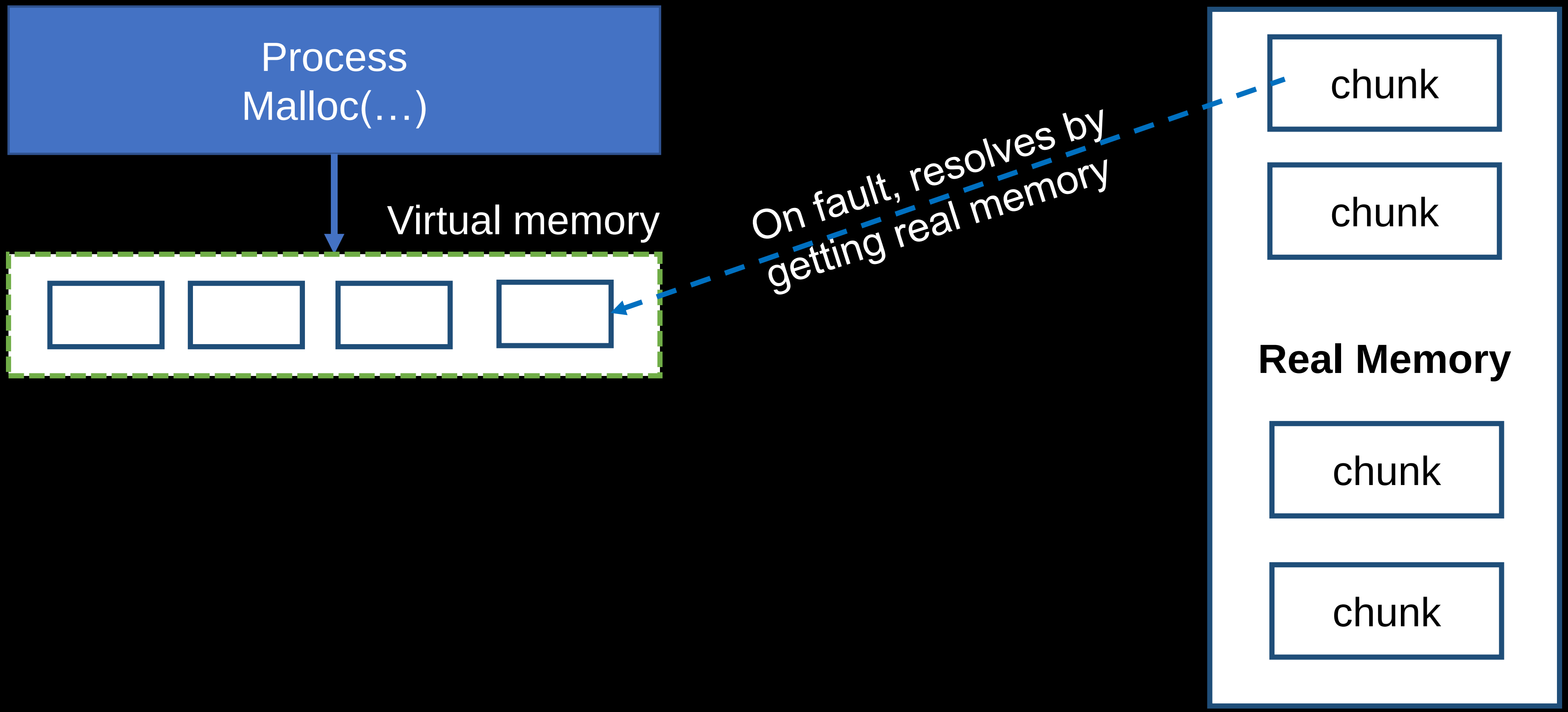
Memory issues: Corruption - access

To understand memory issues, we should understand how memory is allocated to a process/VM.



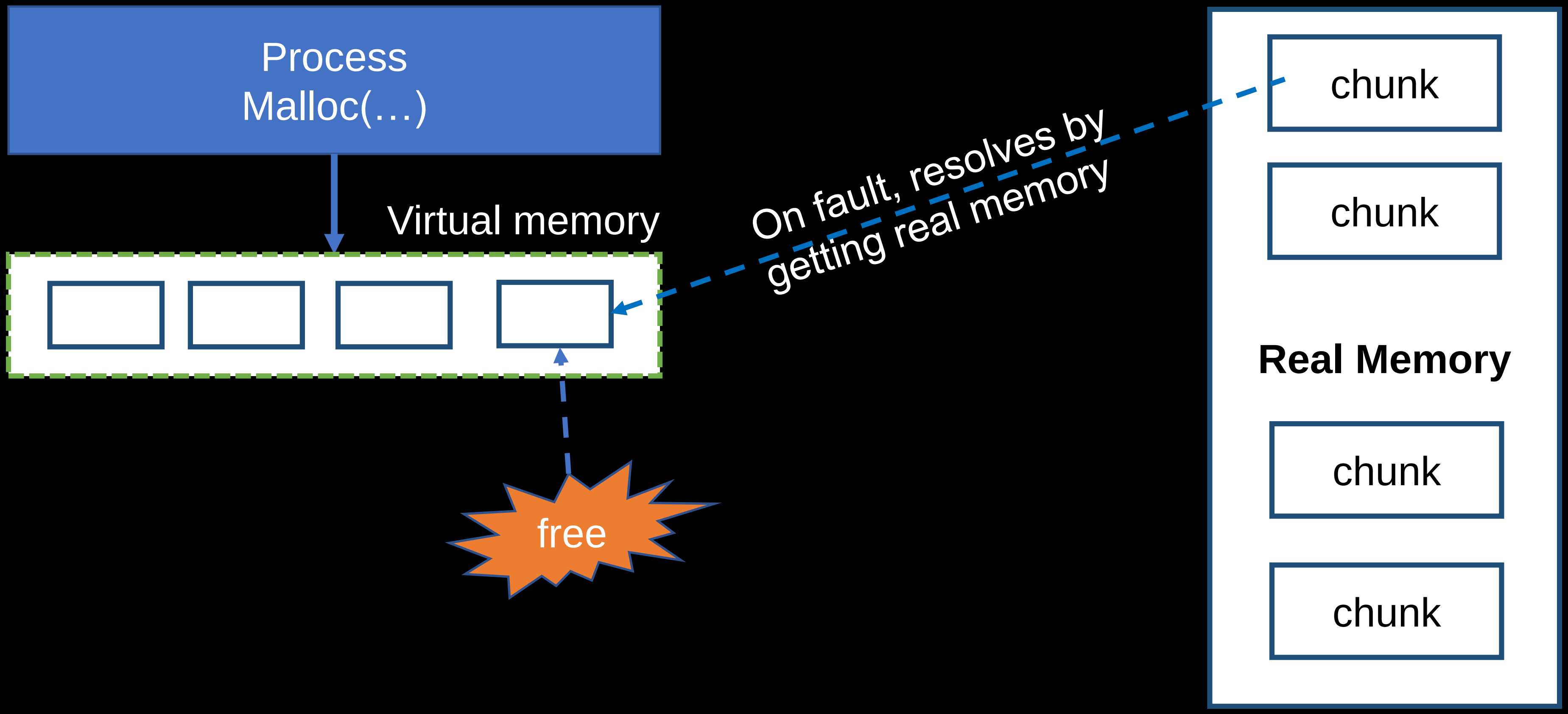
Memory issues: Corruption - access

To understand memory issues, we should understand how memory is allocated to a process/VM.



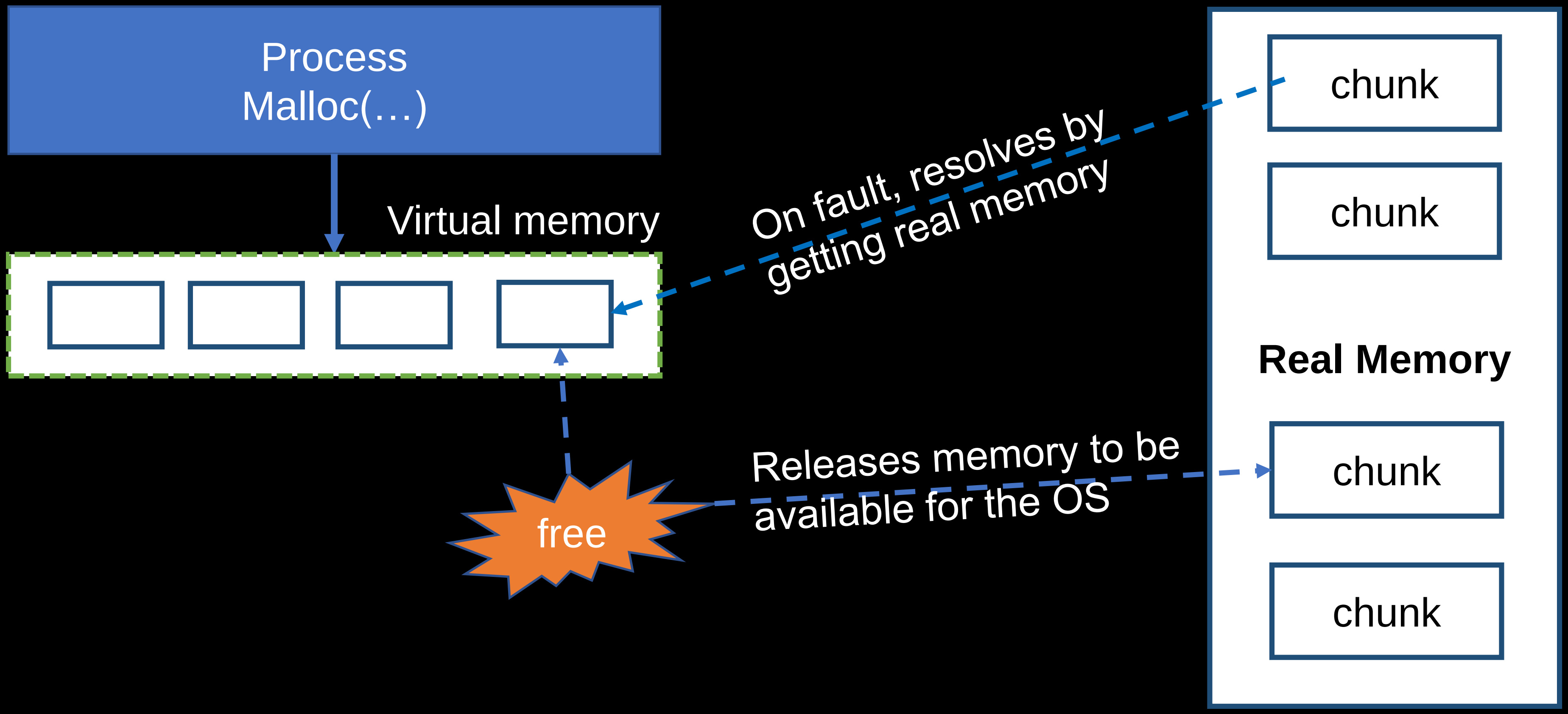
Memory issues: Corruption - access

To understand memory issues, we should understand how memory is allocated to a process/VM.



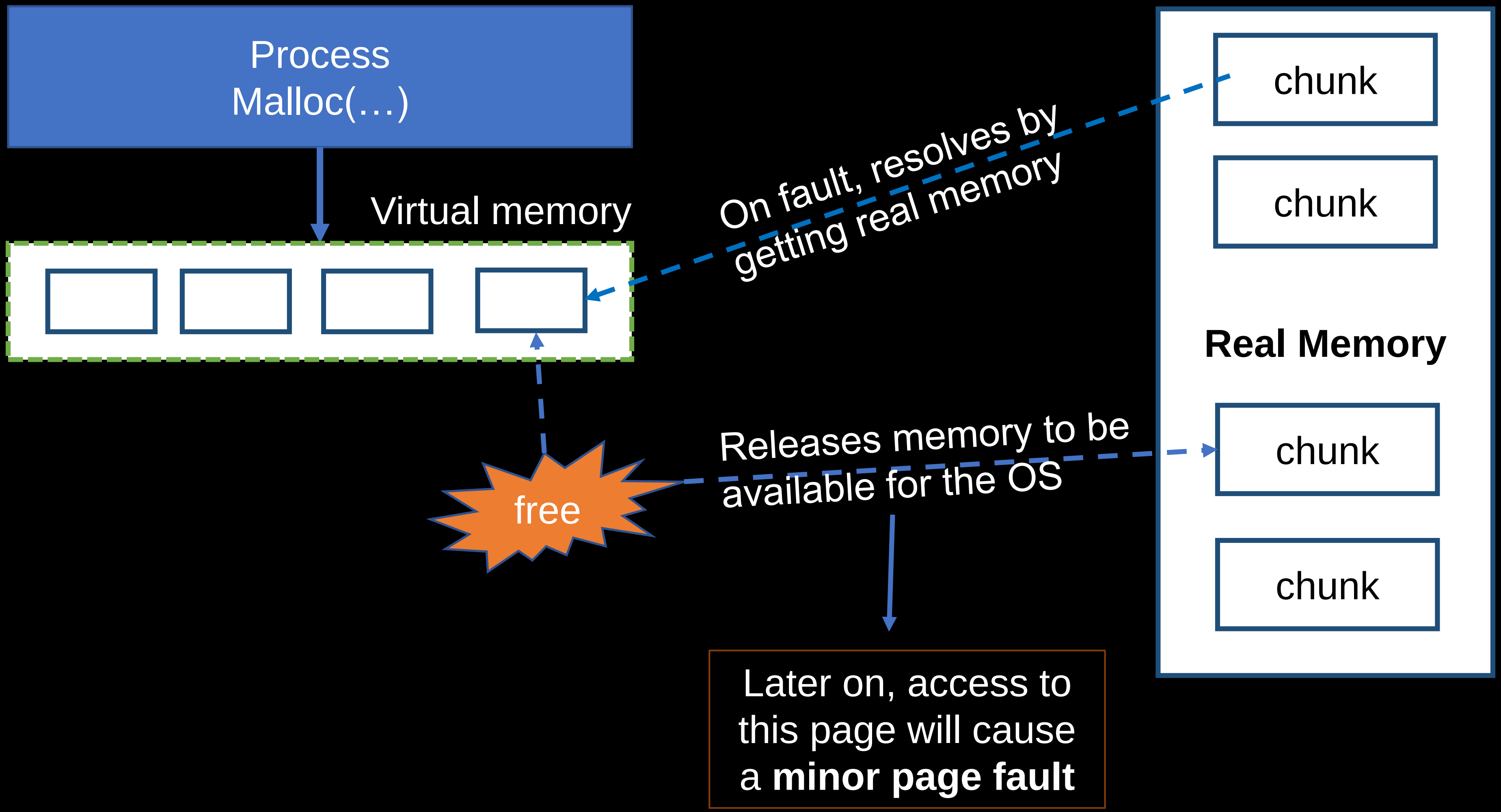
Memory issues: Corruption - access

To understand memory issues, we should understand how memory is allocated to a process/VM.



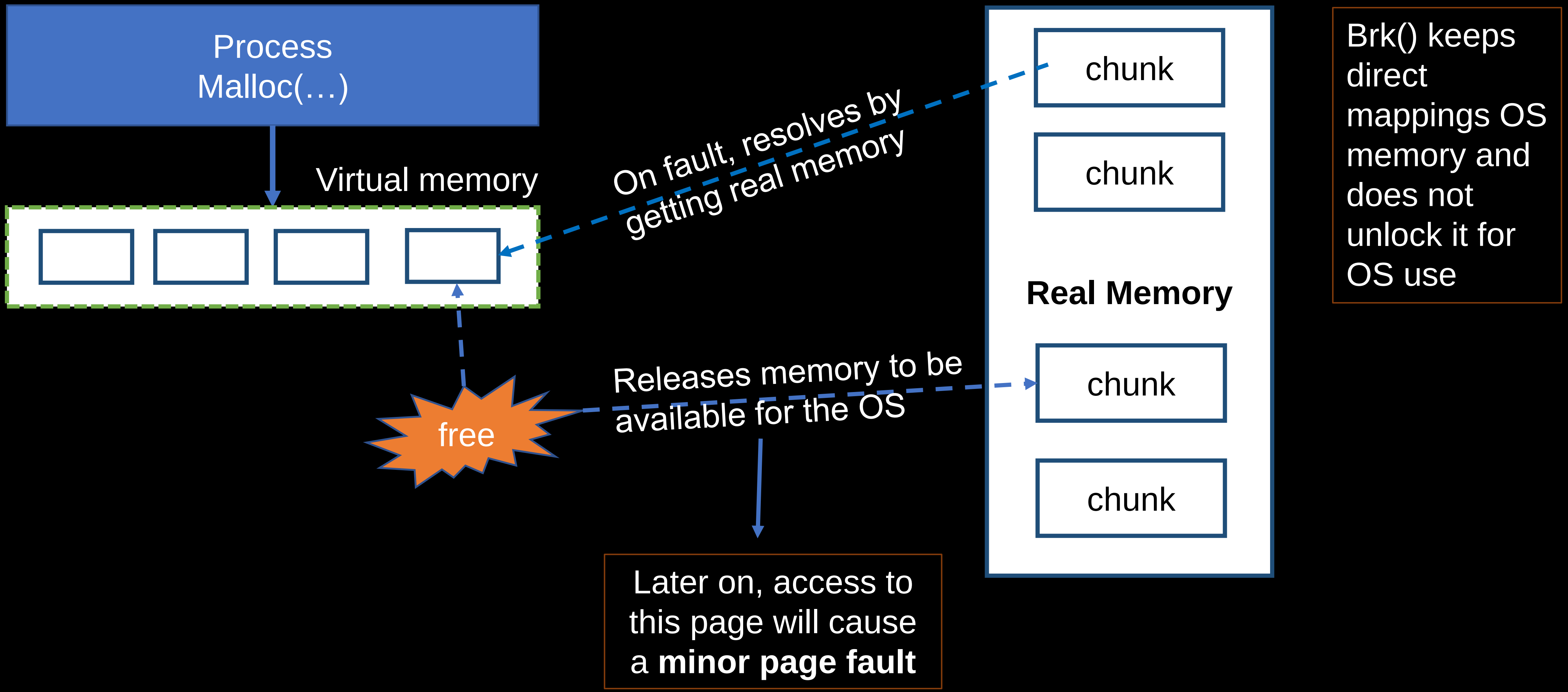
Memory issues: Corruption - access

To understand memory issues, we should understand how memory is allocated to a process/VM.



Memory issues: Corruption - access

To understand memory issues, we should understand how memory is allocated to a process/VM.



Memory issues: Corruption - access

Concretely, a process address space contains the **heap**, which keeps addresses towards code, data, and other libraries segments.

Each block has specific permissions

R: read

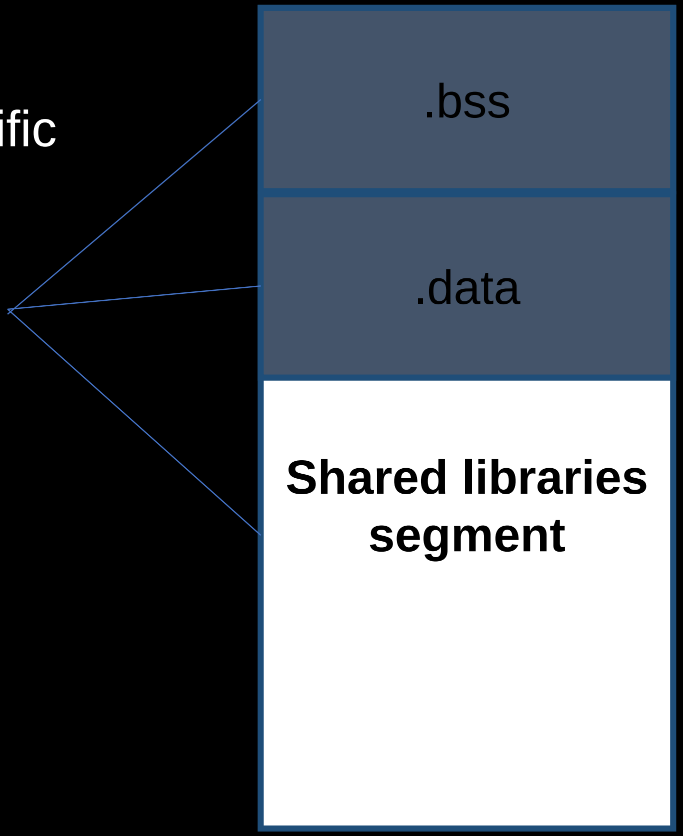
W: write

X: execution

P: protected

S: shared

Process heap



Using informations in the **private heap** of a process is the **prefered attack** mode when dealing with memory issues

Memory issues: Corruption - access

Step by step example to view how memory is allocated and how the OS is called

```c
#include <stdio.h> // standard io
#include <stdlib.h> // C standard library
#include <unistd.h> // unix standard library
#include <sys/types.h> // system types for linux

int main () {

    char * addr;
    printf("Welcome to this vm course ::%d\n", getpid());
    printf("Enter a sentence\n");
    getchar();
    addr = (char *) malloc(1000);
    free(addr);
    printf("Finished\n");

    return 0;
}
```

demo.c

gcc -odemo.o demo.c

strace ./demo.o

What do you observe ?

Memory issues: Corruption - access

Step by step example to view how memory is allocated and how the OS is called

```c
#include <stdio.h> // standard io
#include <stdlib.h> // C standard library
#include <unistd.h> // unix standard library
#include <sys/types.h> // system types for linux

int main () {

    char * addr;
    printf("Welcome to this vm course ::%d\n", getpid());
    printf("Enter a sentence\n");
    getchar();
    addr = (char *) malloc(1000);
    free(addr);
    printf("Finished\n");

    return 0;
}
```

demo.c

What do you observe ?

mmap()
mprotect()

What is their purpose ?

Memory issues: Corruption - access

Step by step example to view how memory is allocated and how the OS is called

mmap(): Maps a virtual memory region and defines the behavior when trying to access it (fetch on IO device or RAM).

Returns a pointer of the start address of the mapped region

mprotect(): Protects a memory region to prevent it from being allocated by the kernel.

Memory issues: Corruption - access

More details on the heap informated currently used by a process

/proc/<pid>/maps

```
7faa724da000-7faa72652000 r-xp 00025000 08:30 11971                      /usr/lib/x86_64-
linux-gnu/libc-2.31.so
7faa72652000-7faa7269c000 r--p 0019d000 08:30 11971                      /usr/lib/x86_64-
linux-gnu/libc-2.31.so
7faa7269c000-7faa7269d000 ---p 001e7000 08:30 11971                      /usr/lib/x86_64-
linux-gnu/libc-2.31.so
```

Provides information on the range of memory used, mappings, protections, and the type of underneath device.

Each line format :

<address start>-<address end> <mode> <offset> <major id:minor id> <inode id> <file path>

Memory issues: Corruption - access

Check the mappings for your C program and also **cat /proc/self/maps**

How can this information be maliciously used ?

Memory issues: Corruption - access

Check the mappings for your C program and also **cat /proc/self/maps**

How can this information be maliciously used ?

Buffer overflow

Memory issues: Corruption - access

Check the mappings for your C program and also **cat /proc/self/maps**

How can this information be maliciously used ?

Buffer overflow

Branch prediction

Memory issues: Corruption - access

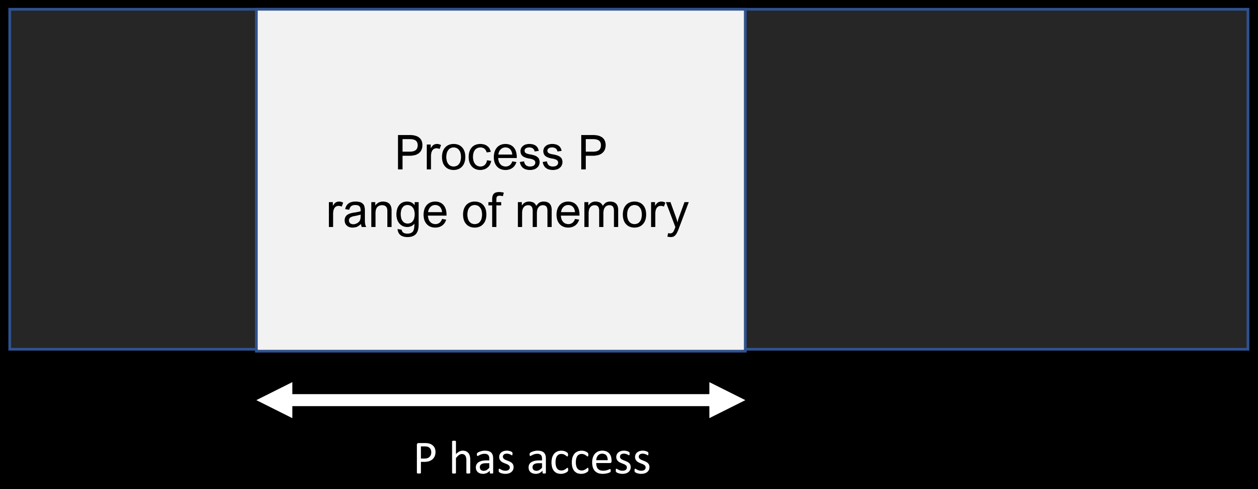
Check the mappings for your C program and also **cat /proc/self/maps**

How can this information be maliciously used ?

- Buffer overflow
- Branch prediction
- Admin privileges

Buffer overflow

Utiliser les zones de mémoires d'un processus pour essayer de lire les zones en dehors de sa juridiction.

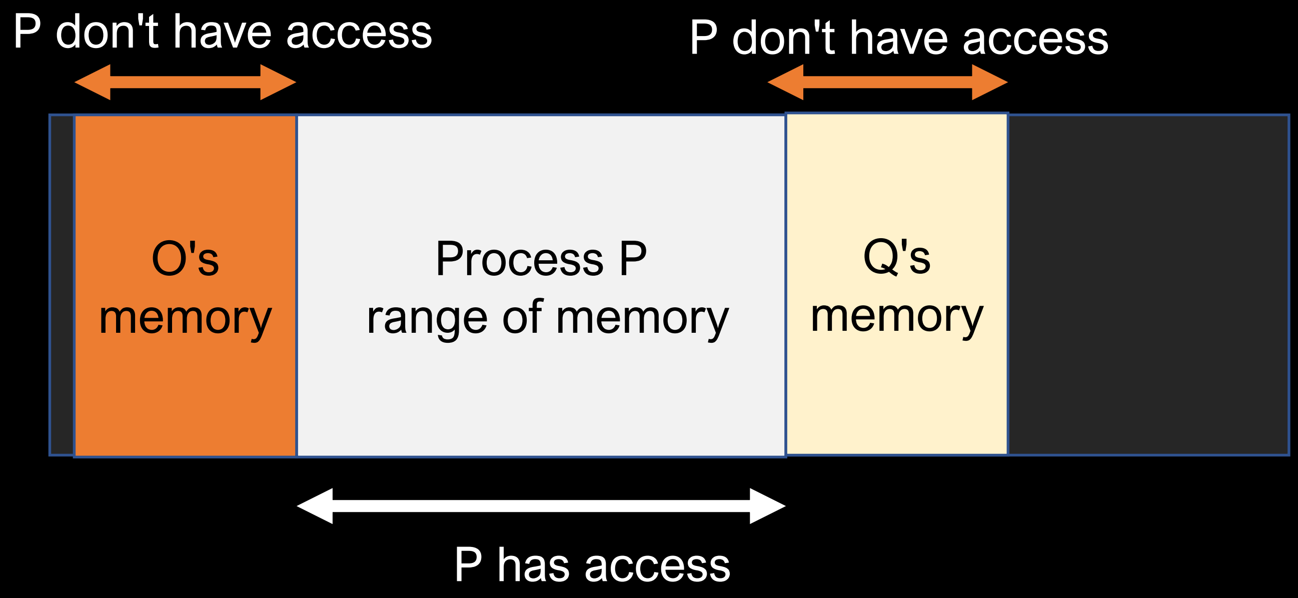


Branch prediction

Admin privileges

Buffer overflow

Utiliser les zones de mémoires d'un processus pour essayer de lire les zones en dehors de sa juridiction.

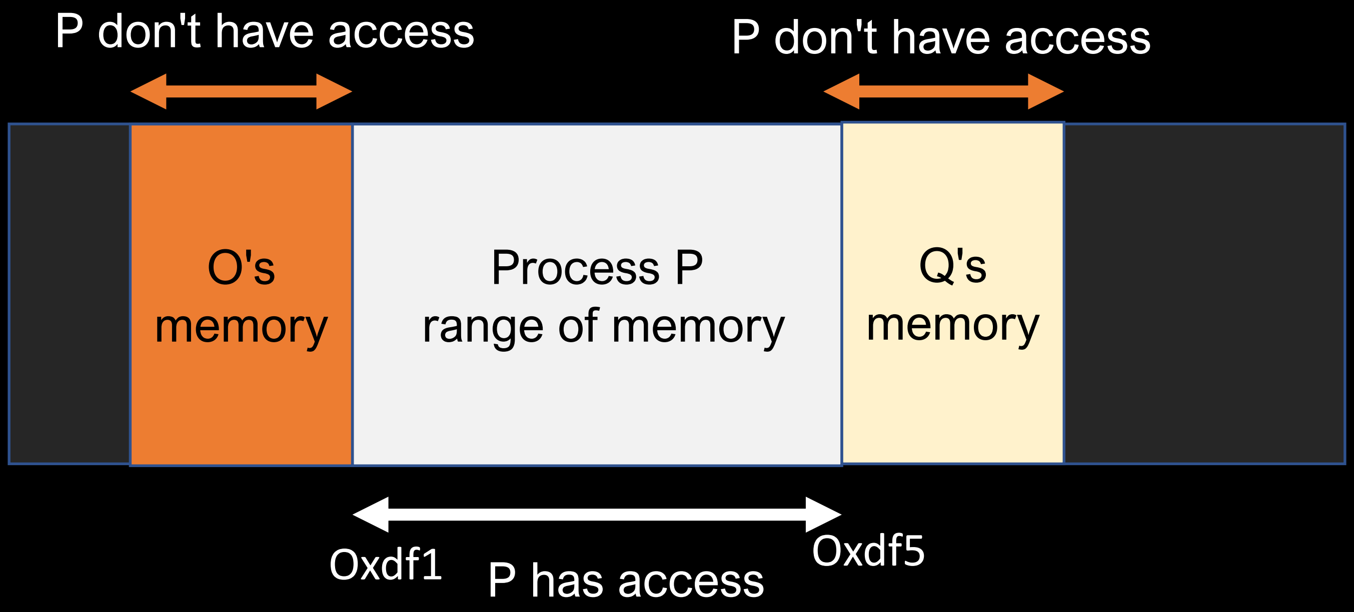


Branch prediction

Admin privileges

Buffer overflow

Utiliser les zones de mémoires d'un processus pour essayer de lire les zones en dehors de sa juridiction.



But what happens if Q's memory is not correctly protected?

Then P can go overbound (overflow). Any process can try to go overbound by manually triggering a read at a specific address.

Branch prediction

Admin privileges

Buffer overflow

```c
#include <stdio.h>
#include <string.h>

int main(void)
{
    char buff[15];
    int pass = 0;

    printf("\n Enter the password : \n");
    gets(buff);

    if(strcmp(buff, "ndoleplantain"))
        printf ("\n Wrong Password \n");
    else
    {
        printf ("\n Correct Password \n");
        pass = 1;
    }

    if(pass)
       /* Now Give root or admin rights to user*/
        printf ("\n Root privileges given to the user \n");

    return 0;
}
```

Exemple: Code d'authentification en C

gcc -ooverflow.o overflow.c –fno-stack-protector –zexecstatck –fno-pie

./overflow.o

What do you observe ?

Buffer overflow

Morris Worm: The Morris worm of 1988 was one of the first internet-distributed computer worms, and the first to gain significant mainstream media attention. It exploited a buffer overflow vulnerability in the Unix sendmail, finger, and rsh/rexec, infecting 10% of the

SQL Slammer: SQL Slammer is a 2003 computer worm that exploited a buffer overflow bug in

Adobe Flash Player: In 2016, a buffer overflow vulnerability was found in Adobe Flash Player for Windows, macOS, Linux and Chrome OS. The vulnerability was due to an error in Adobe Flash Player while parsing a specially crafted SWF (Shockwave Flash) file. Malicious

WhatsApp VoIP: In May 2019, Facebook announced a vulnerability associated with all of its WhatsApp products. The vulnerability exploited a buffer overflow weakness in WhatsApp's VOIP stack on smartphones. This allows remote code execution via a specially-crafted

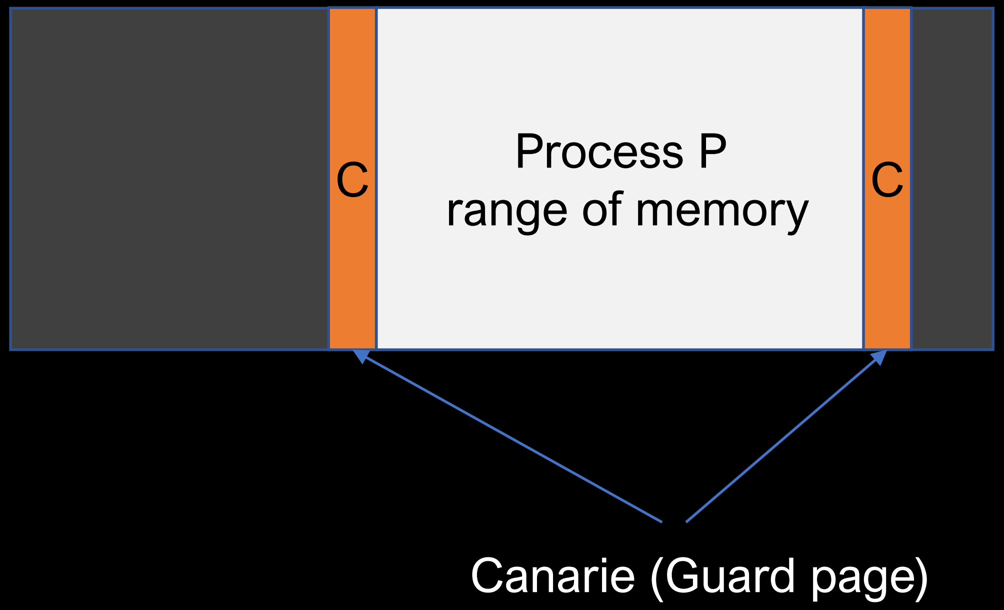
Branch prediction

Admin privileges

Se protéger contre Buffer overflow

Activer les canaries dans un système d'exploitation

Page mémoire placées à la fin d'une zone de mémoire afin de détecter des débordements



Branch prediction

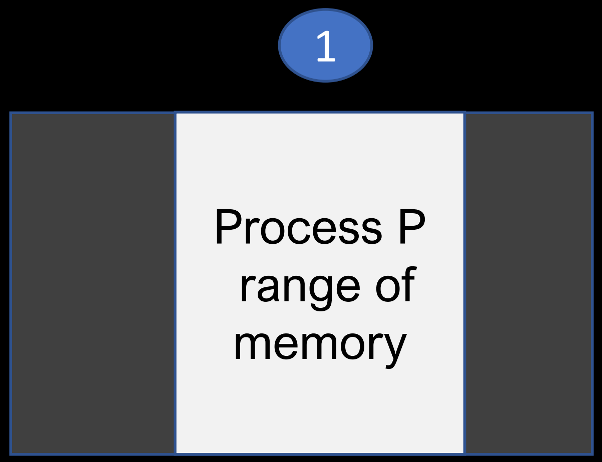
Admin privileges

Se protéger contre Buffer overflow

Le système va régulièrement changer l'emplacement des adresses de votre tas pour bloquer des attaques liés à l'ancien emplacement

Activer les canaries dans un système d'exploitation

(K)ASLR – (Kernel) Adress Space Layout Randomization



Page mémoire placées à la fin d'une zone de mémoire afin de détecter des débordements

Branch prediction

Admin privileges

Se protéger contre Buffer overflow

Le système va régulièrement changer l'emplacement des adresses de votre tas pour bloquer des attaques liés à l'ancien emplacement

Activer les canaries dans un système d'exploitation

(K)ASLR – (Kernel) Adress Space Layout Randomization

2

Process P range of memory

Page mémoire placées à la fin d'une zone de mémoire afin de détecter des débordements

Branch prediction

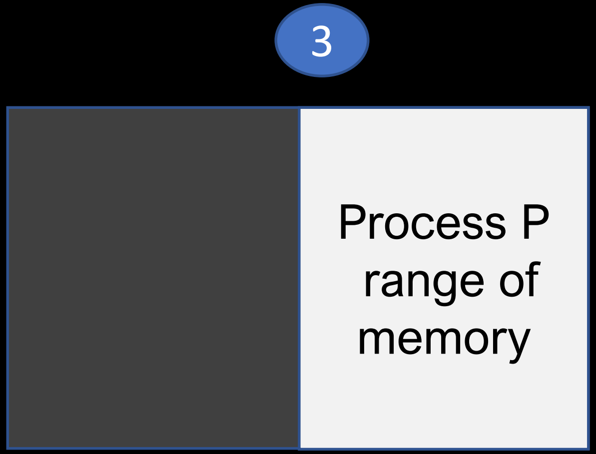
Admin privileges

Se protéger contre Buffer overflow

Le système va régulièrement changer l'emplacement des adresses de votre tas pour bloquer des attaques liés à l'ancien emplacement

Activer les canaries dans un système d'exploitation

(K)ASLR – (Kernel) Adress Space Layout Randomization



Page mémoire placées à la fin d'une zone de mémoire afin de détecter des débordements

Branch prediction

Admin privileges