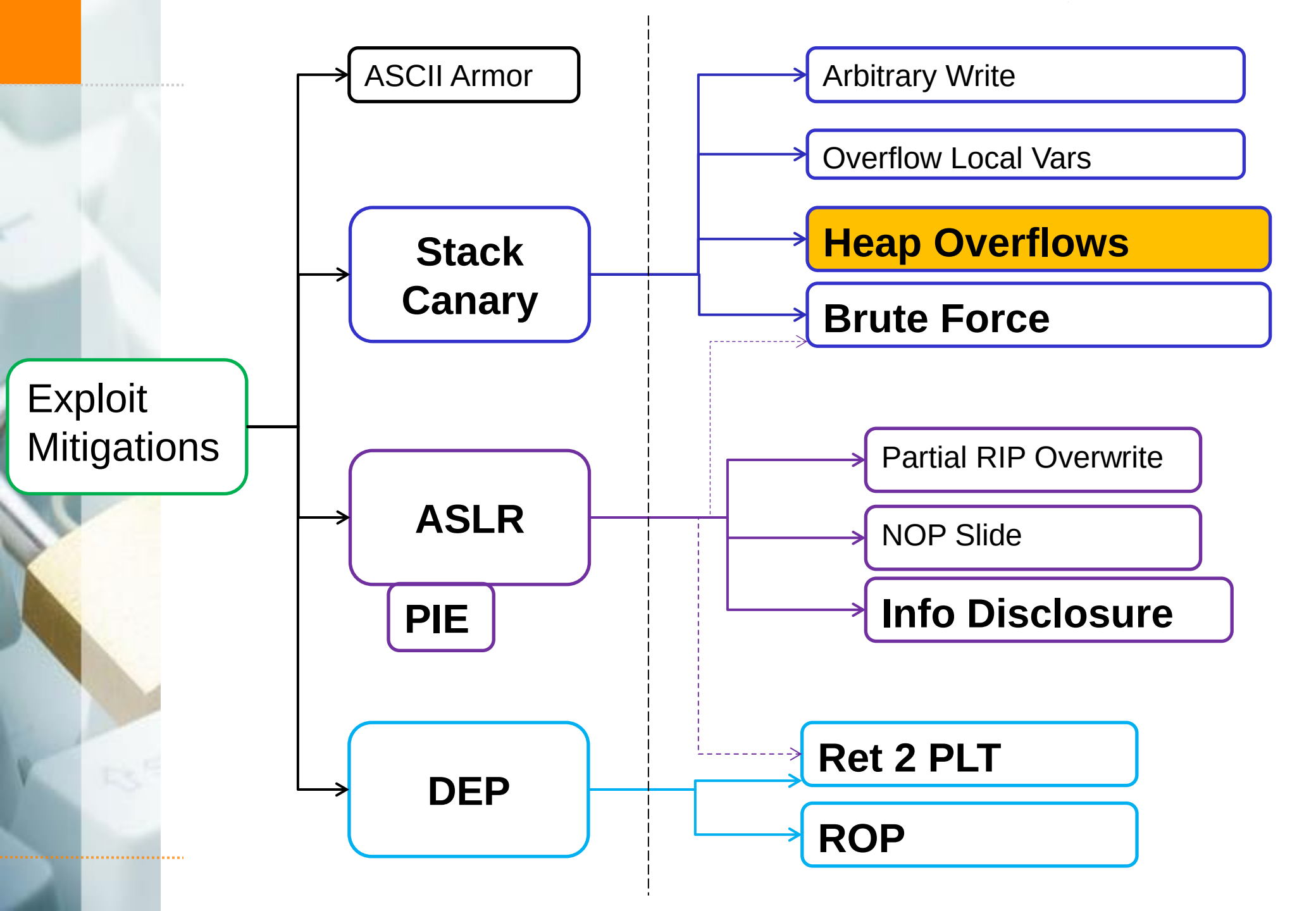
Defeat Exploit Mitigation
Heap Intro

HEAP

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch


Exploit Mitigations
ASCII Armor
Stack Canary
ASLR
PIE
DEP
Arbitrary Write
Overflow Local Vars
Heap Overflows
Brute Force
Partial RIP Overwrite
NOP Slide
Info Disclosure
Ret 2 PLT
ROP

Heap Exploitation

This slidedeck is not completely technically accurate

Should give an overview of heap exploitation concepts

Heap Introduction

What is a heap?

- malloc() allocations
- Fullfill allocating and deallocating of memory regions

Heap usage:

- Global variables (live longer than a function)
- Can be big (several kilobytes or even megabytes)

Reminder: Stack usage:

- Function-local variables
- Relatively small (usually <100 or <1000 bytes)

Heap Introduction

Heap:

- Dynamic memory (allocations at runtime)
- Objects, big buffers, structs, persistence, large things
- Slow, manually

Stack:

- Fixed memory allocations (known at compile time)
- Local variables, return addresses, function args
- Fast, automatic

Heap Introduction

Userspace/OS can implement his own memory allocator

- Linux: ptmalloc2 (previously dlmalloc)
- Samba: talloc
- FreeBSD and Firefox: jemalloc
- Google: tcmalloc
- Solaris: libumem

- Simplest: mmap() a memory block and manage it

Heap in Linux

- Heap implementation is usually implemented in GLIBC
- Current Heap allocator implementation: **ptmalloc2**
  - Based on dlmalloc
  - From GLIBC 2.4 onwards

- *Previous / Old:*
  - *Doug Lea's memory allocator*
  - *Dlmalloc*
  - *Note: If you research heap exploits, check what allocator is assumed to be used*

Heap introduction

malloc(): Get a memory region

free(): Release a memory region

We only cover manual allocations

- Not: Automatic garbage collection
- (Garbage collection is just an automatic free() by using reference counting)

How does heap work?

void *ptr;

ptr = malloc(len)

- Allocated “len” size memory block
- Returns a pointer to this memory block

free(ptr)

- Tells the memory allocator that the memory block can now be re-used
- Note: ptr is NOT NULL after a free()

What is a heap allocator doing?

- Allocate big memory pages from the OS
- Manage this pages

- Split the pages into smaller chunks
- Make these chunks available to the program

Heap – Simplified Example

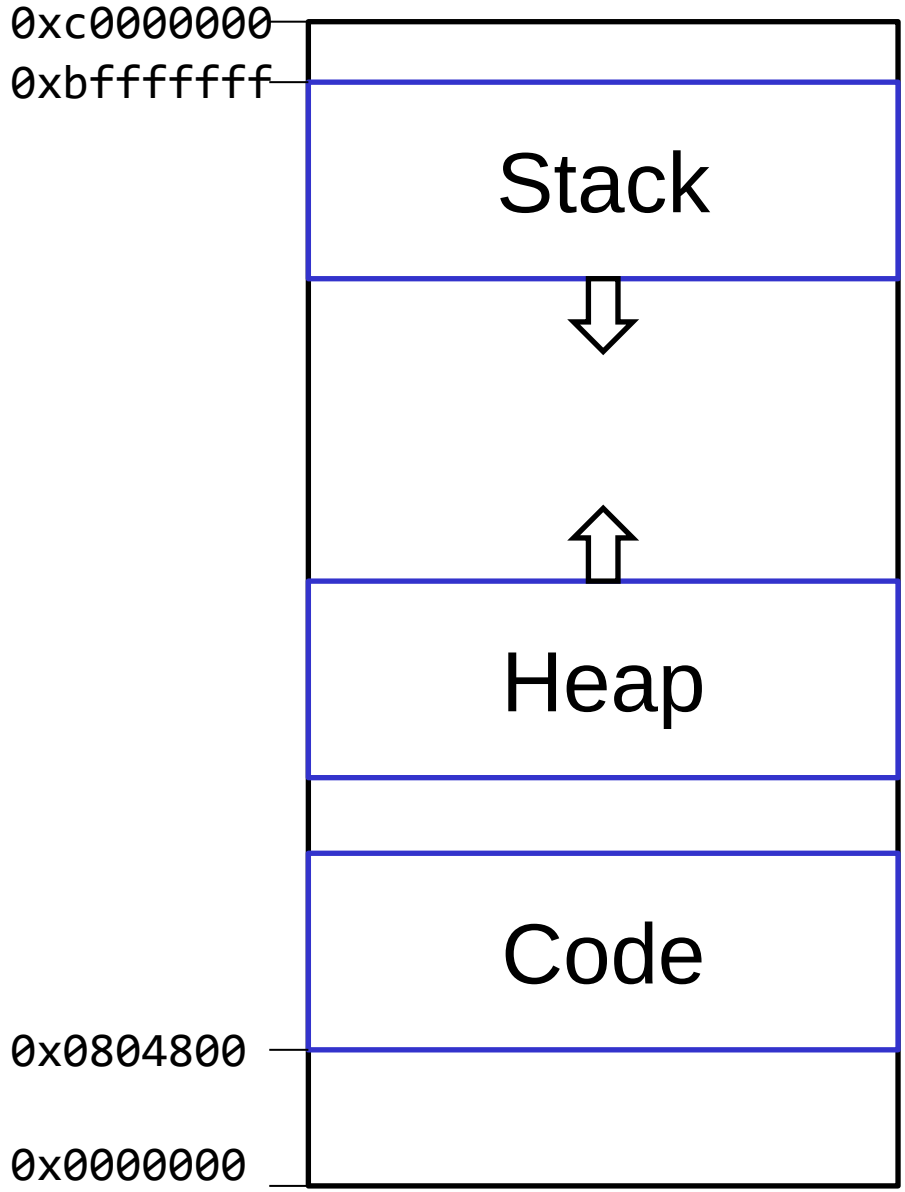
Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

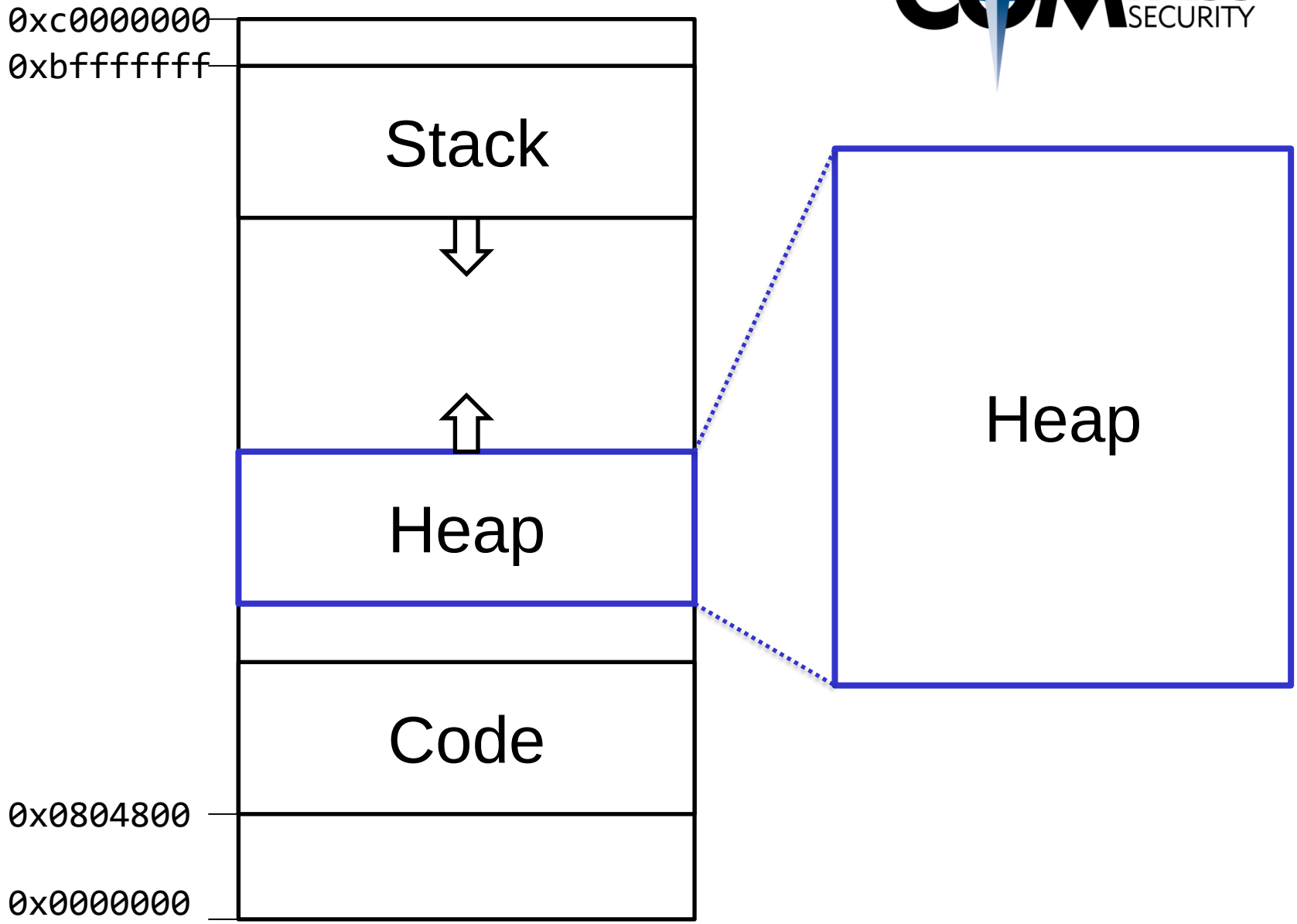
How is this implemented?

- The heap implementation gets a (big) **block of flat/unstructured memory (page)**
- Partition the heap/page into bin's
- A bin has **chunks** of the same size

Heap: Memory Layout



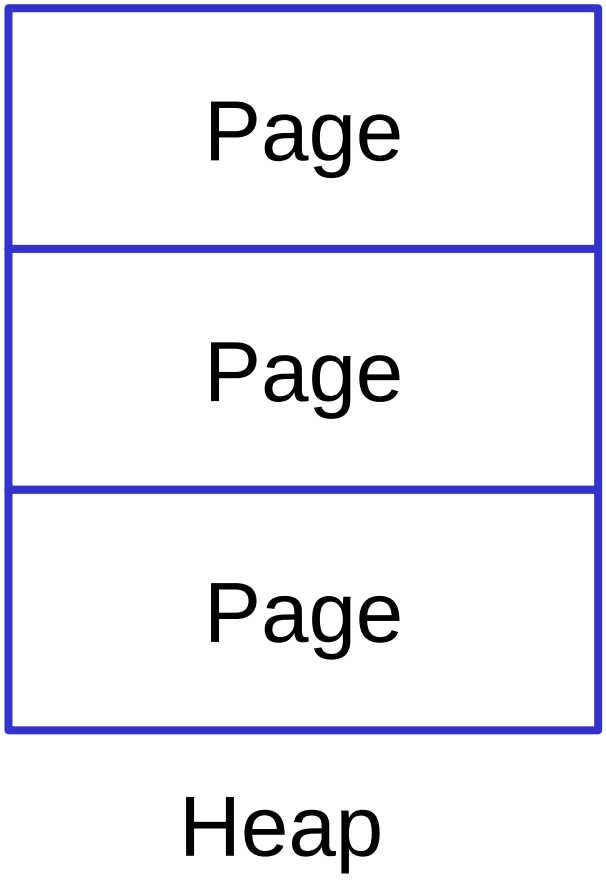
Heap: Memory Layout



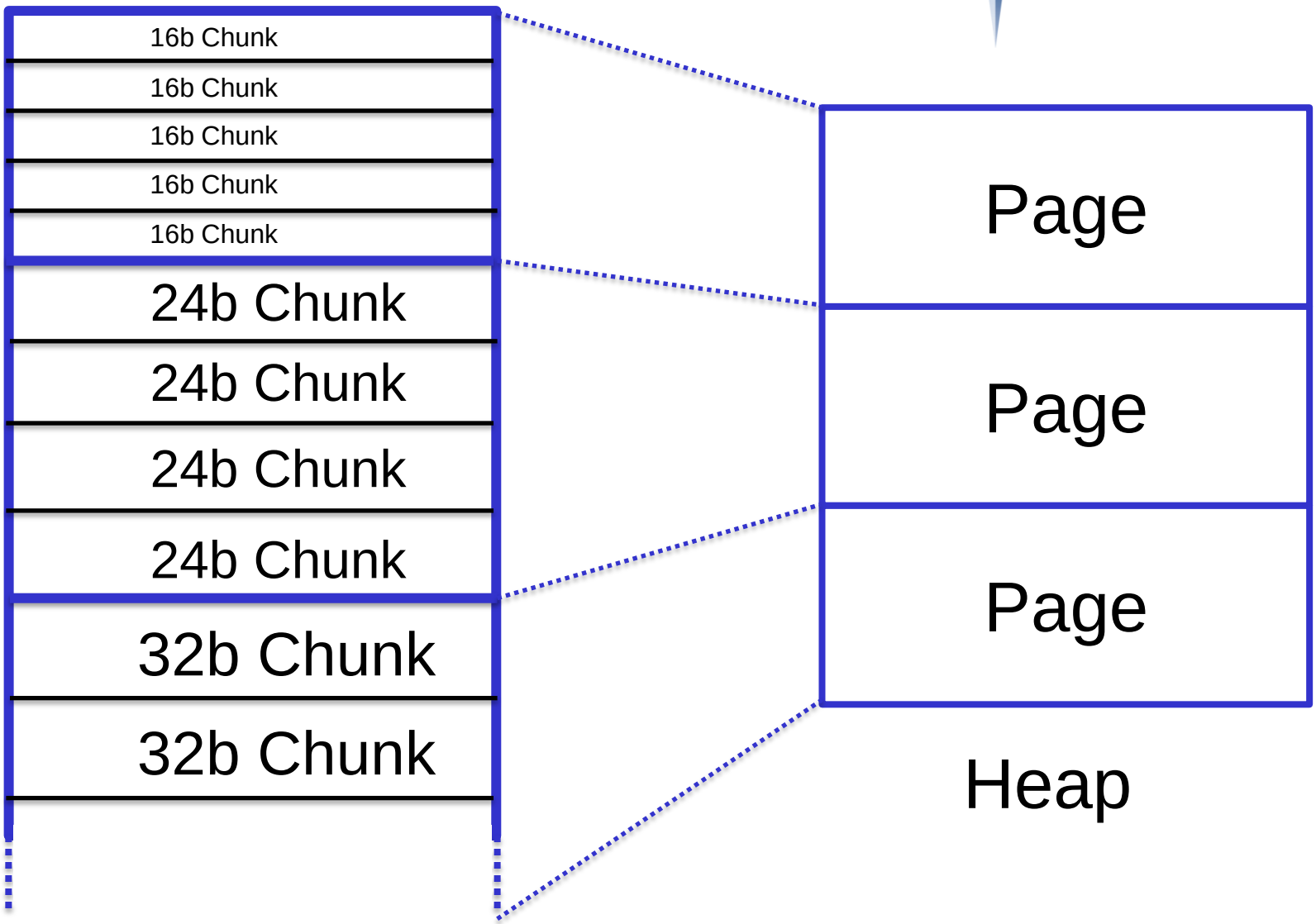
Heap: Memory Layout

Page:

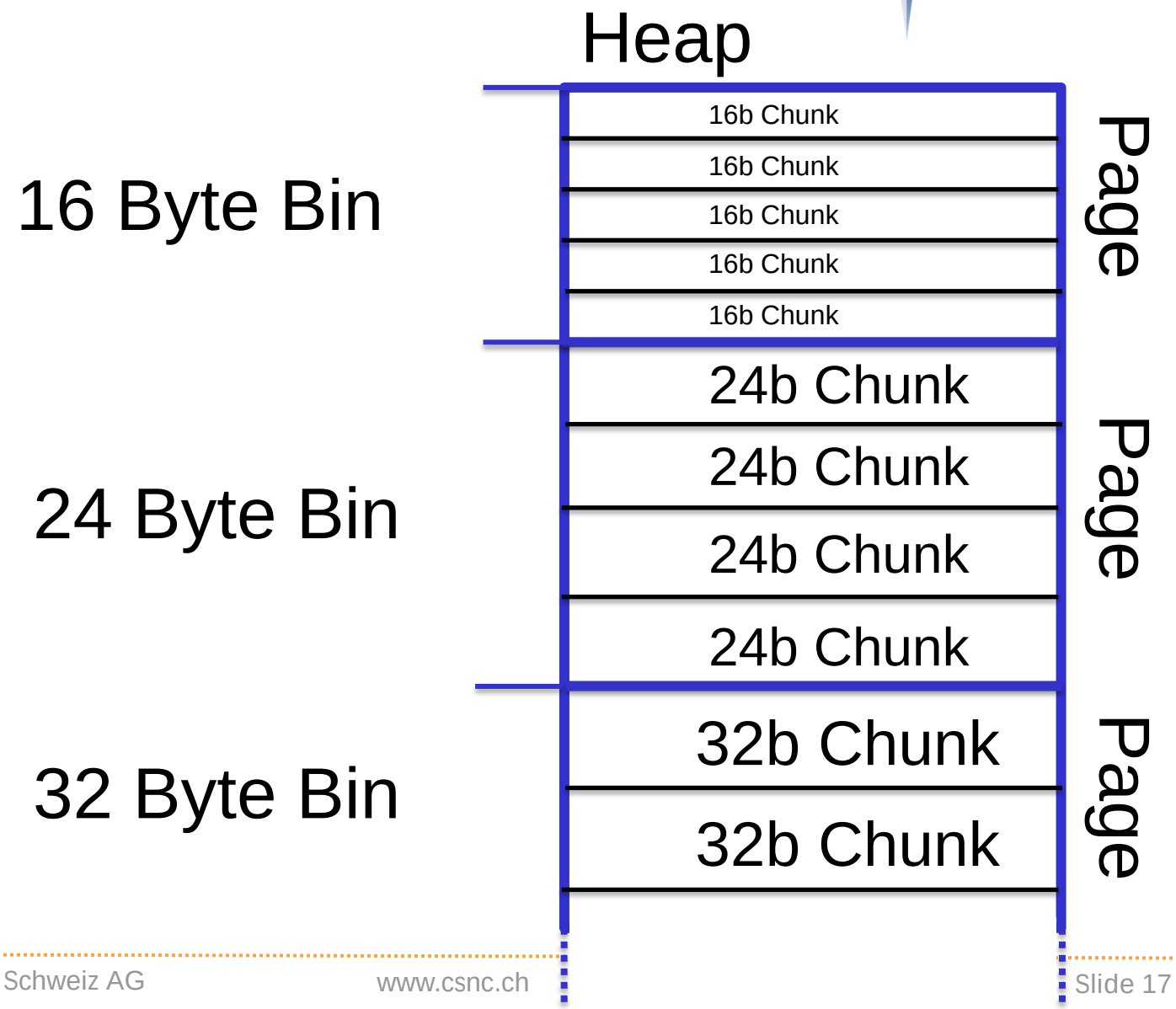
- A memory page
- Usually 4k
- Can also be 2 Megabytes or other
- Allocated via sbrk() or mmap()



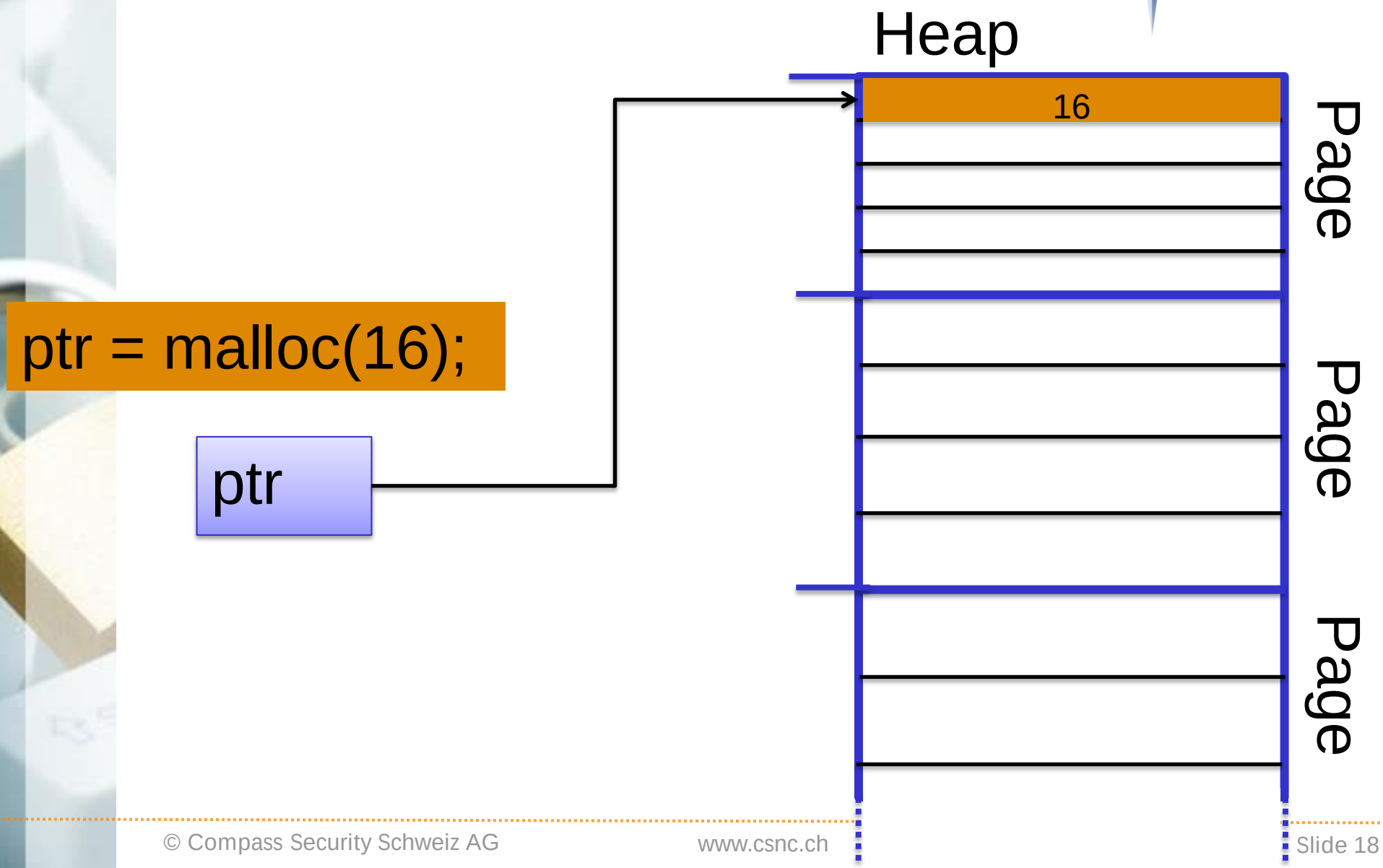
Heap: Memory Layout



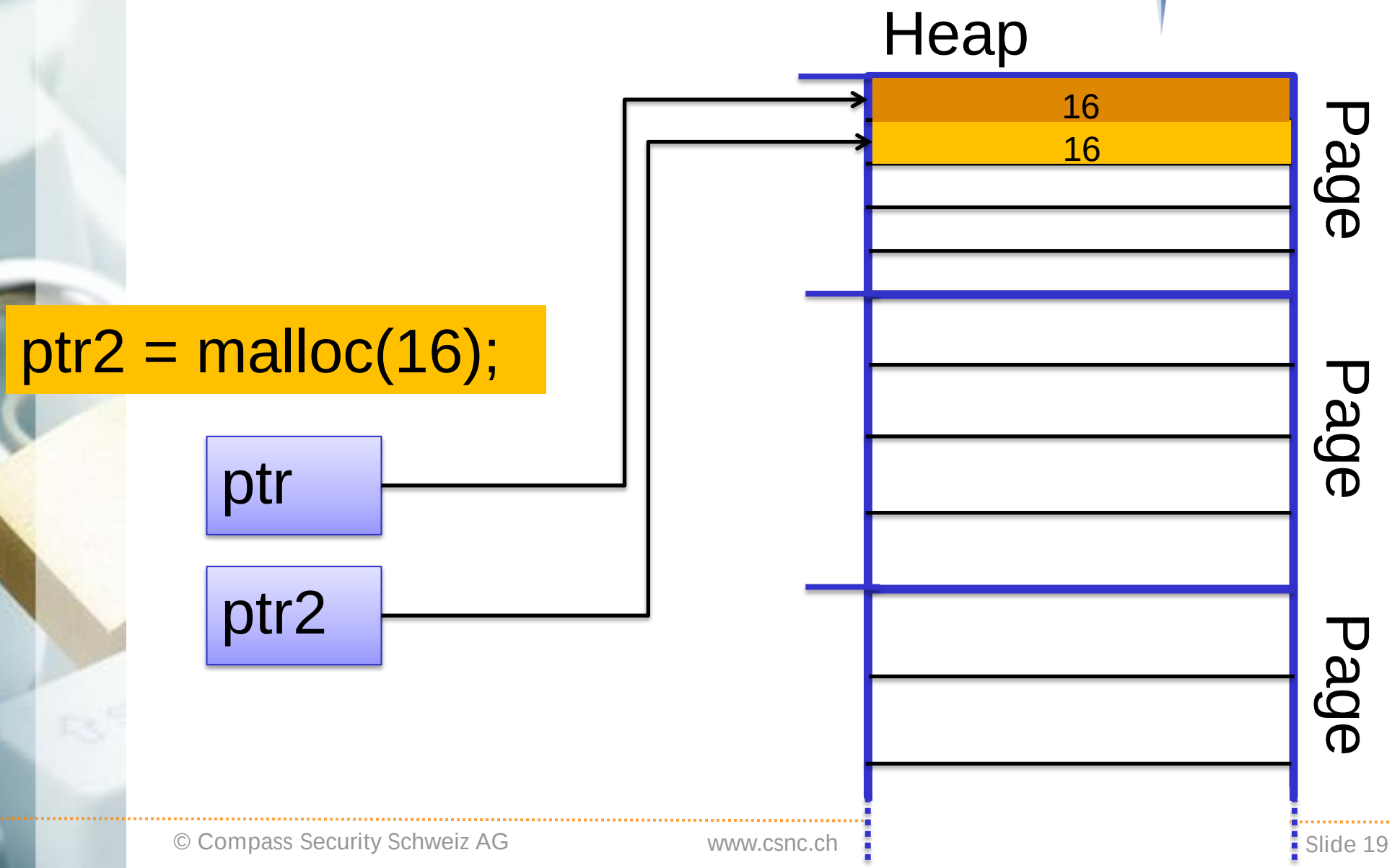
Heap: Oversimplified example



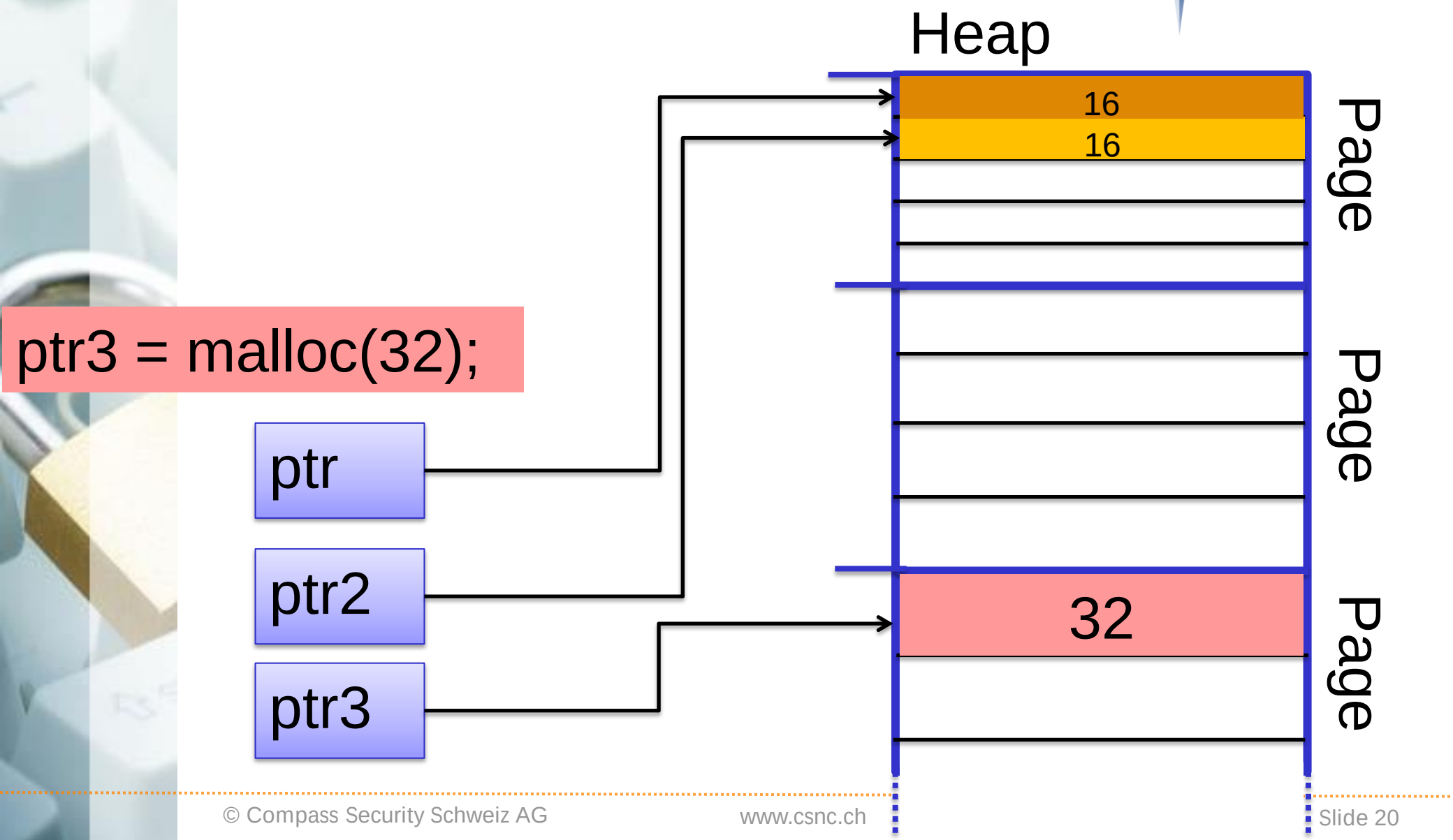
Heap: Oversimplified example



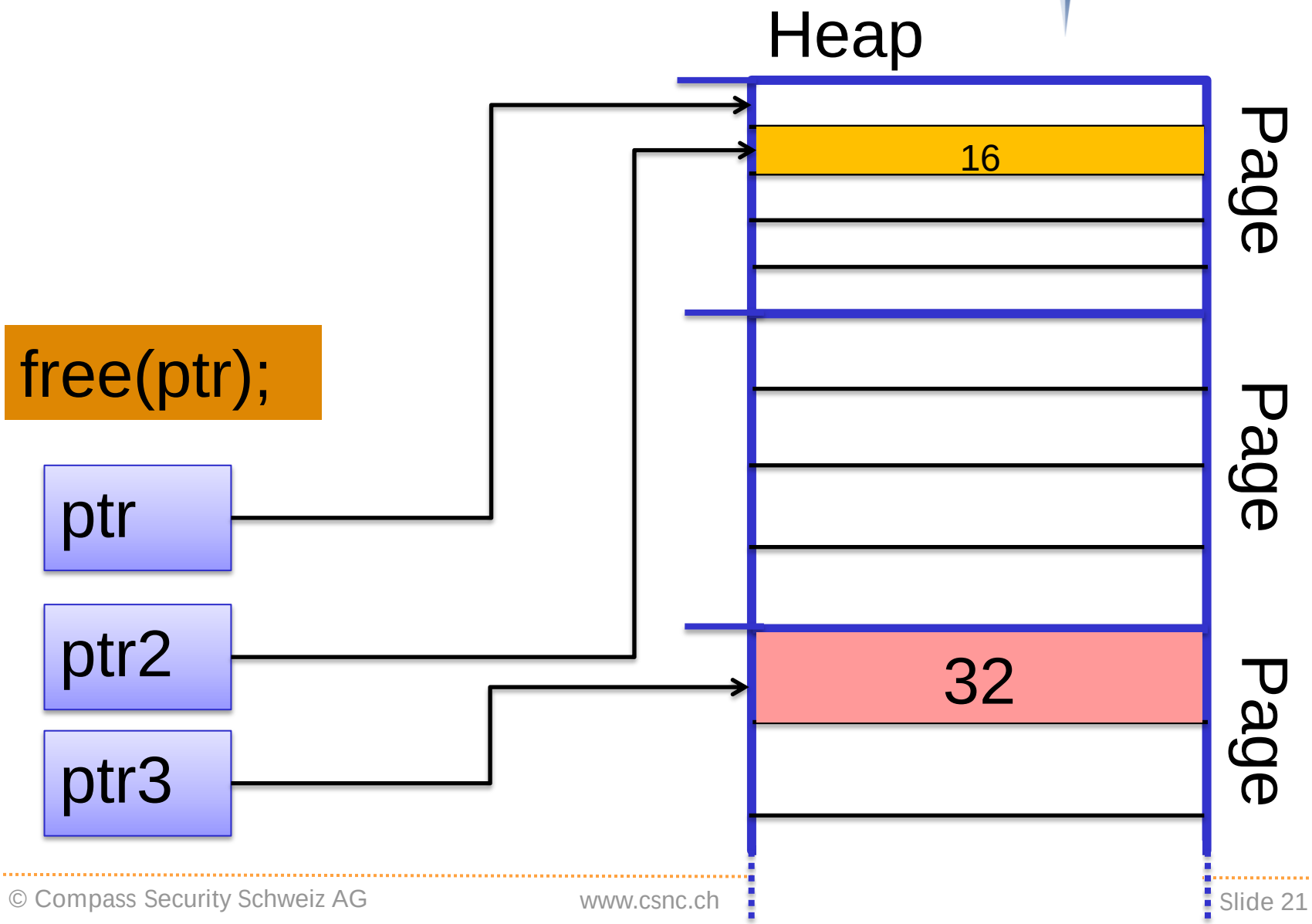
Heap: Oversimplified example



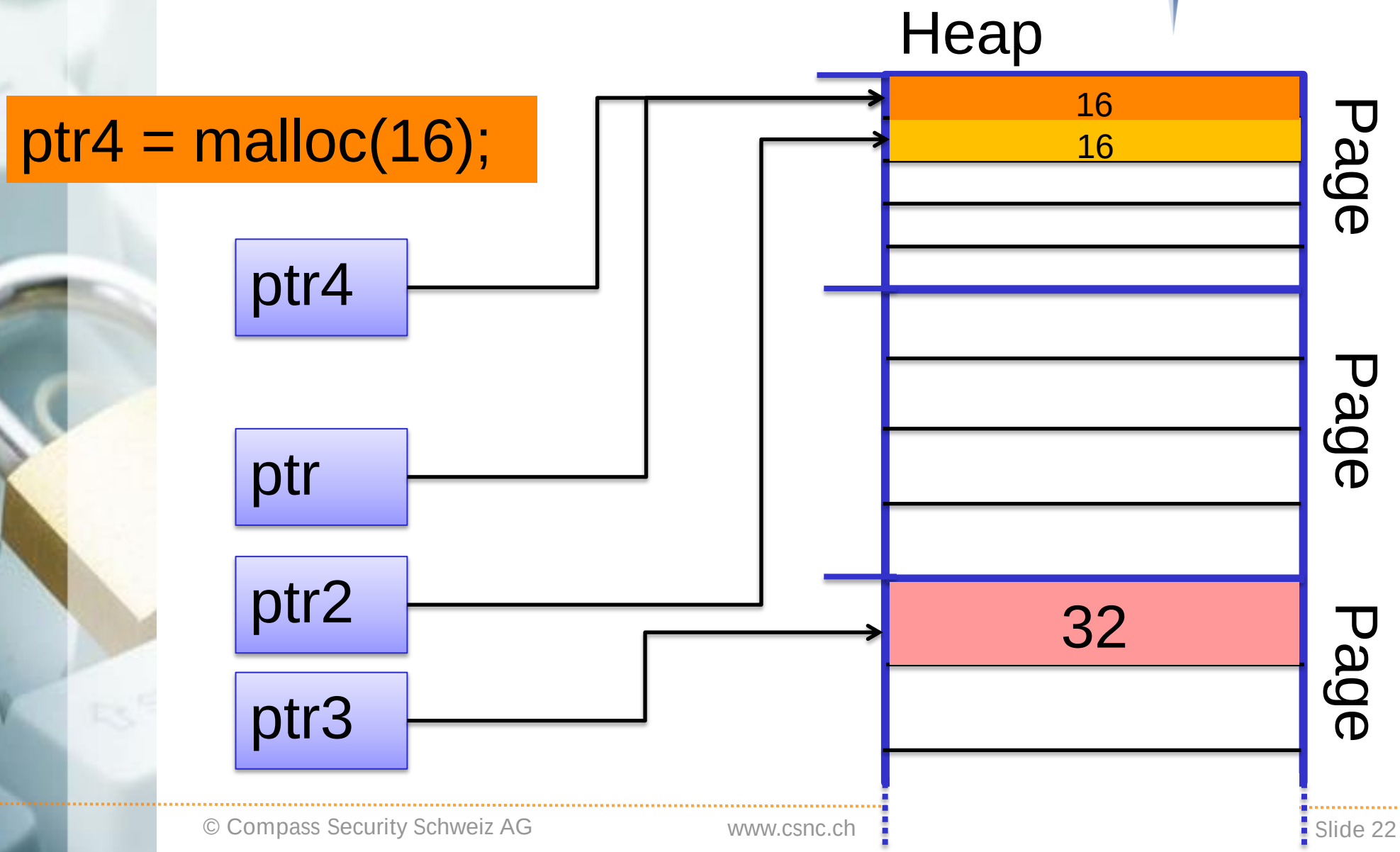
Heap: Oversimplified example



Heap: Oversimplified example



Heap: Oversimplified example



Heap - Recap

Recap:

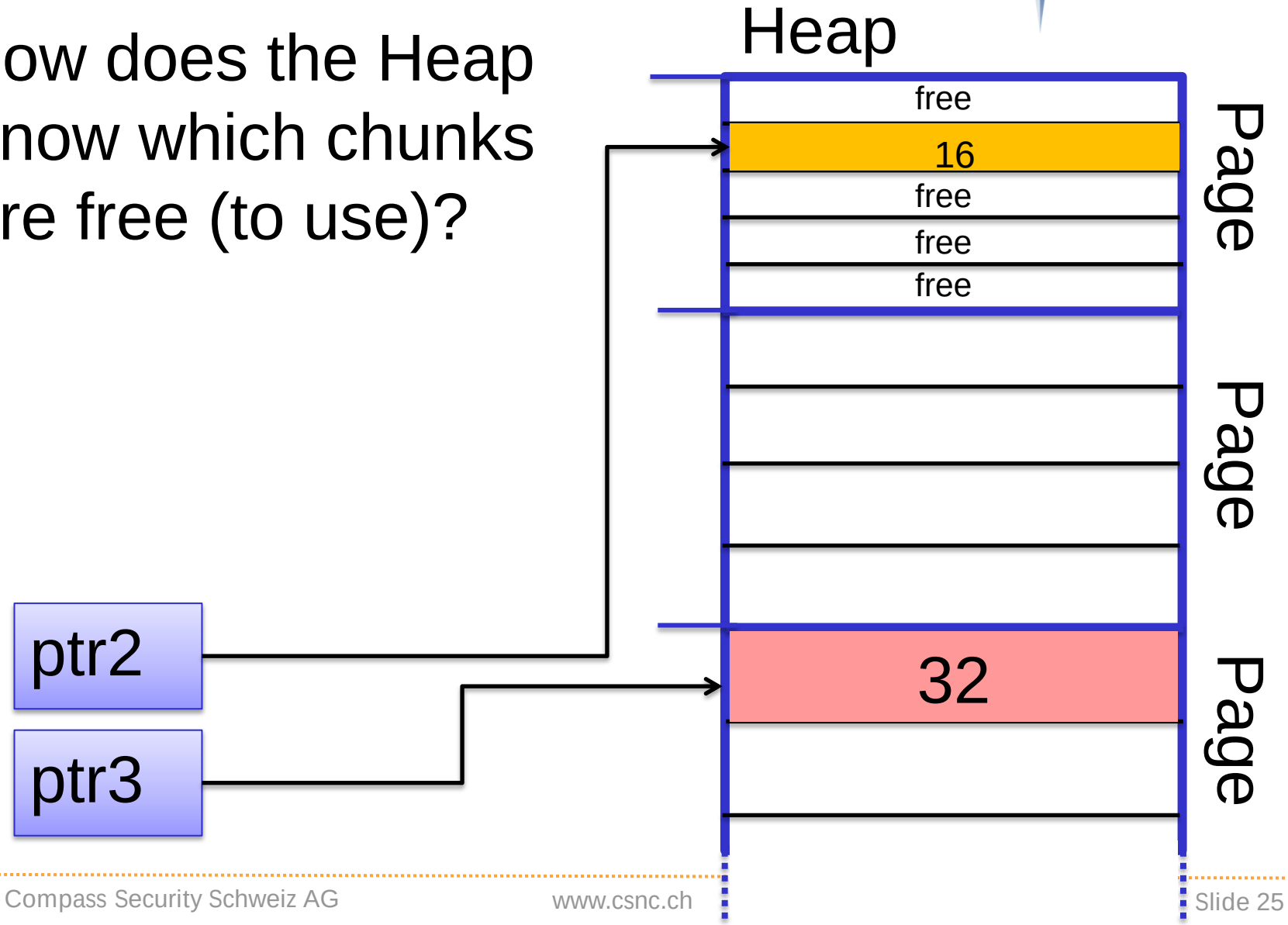
- Heap divides big memory pages into smaller chunks
- Heap gives these chunks to the program on request

Heap Memory Management

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

How does the Heap Know which chunks Are free (to use)?



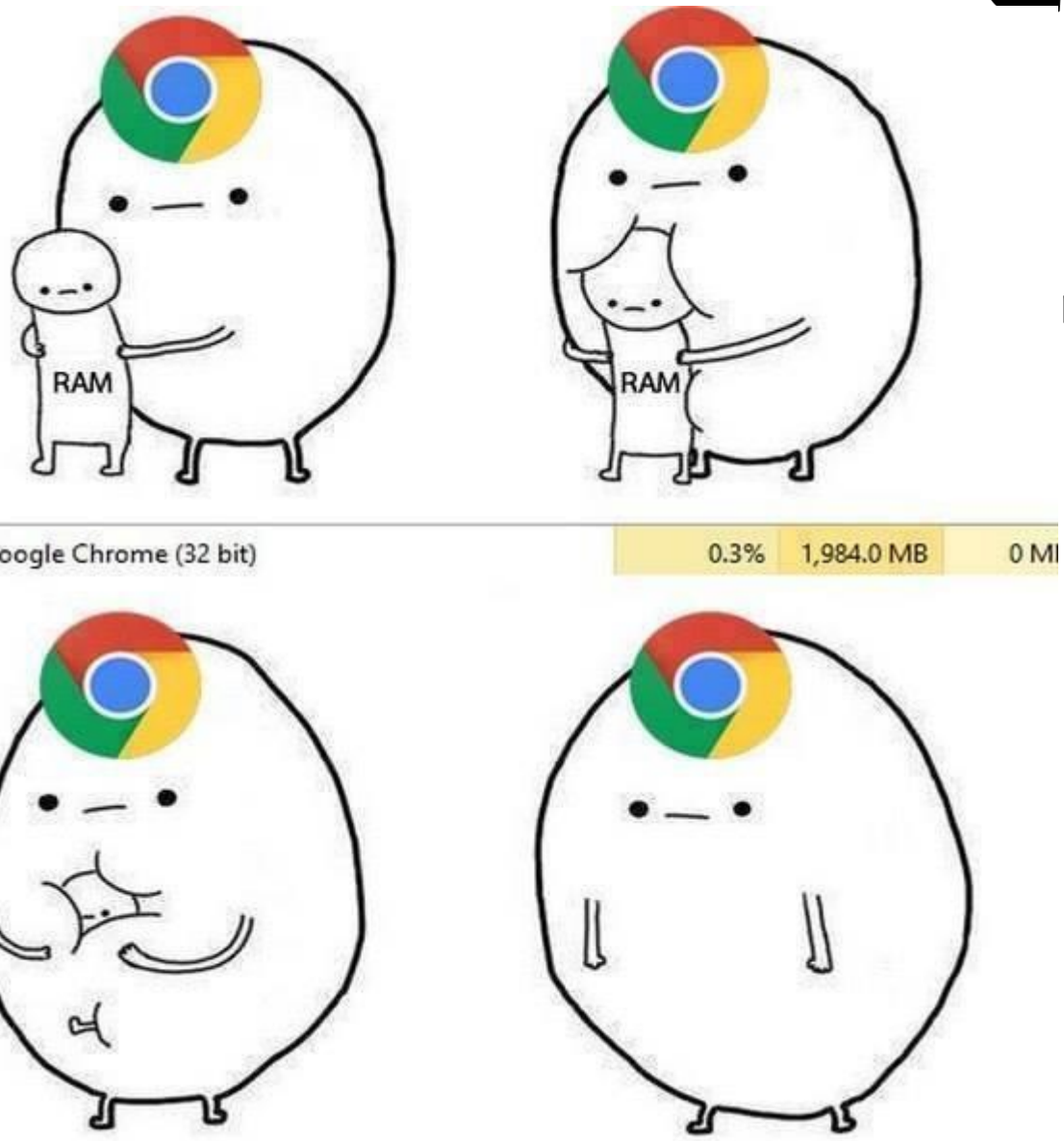
Heap allocator requirements:

- Should be quick to fulfill malloc() and free()
- Should not waste memory by managing memory

- Also: No bugs, correct, low-fragmentation, etc.

Heap Memory Management

Hea



ry

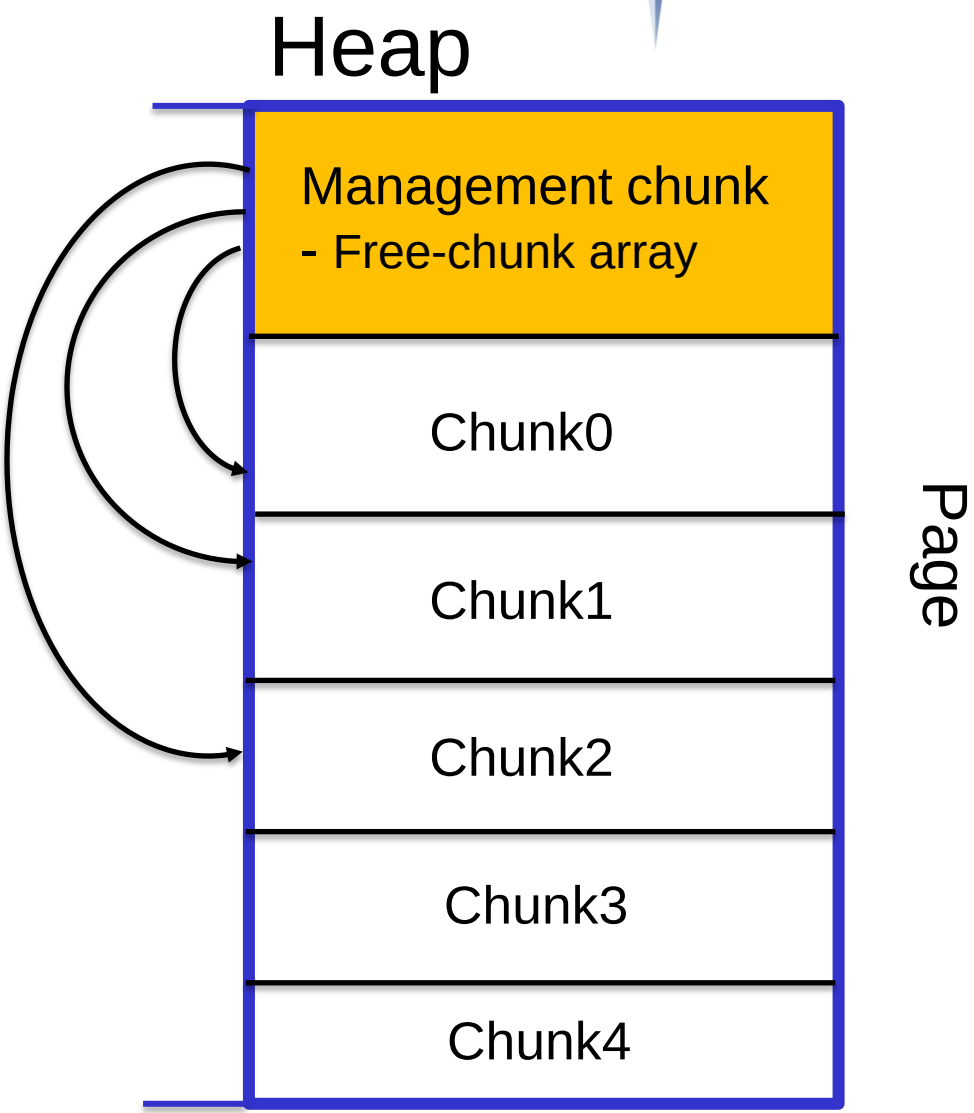
Heap Memory Management

One possibility:

PHP7 – emalloc

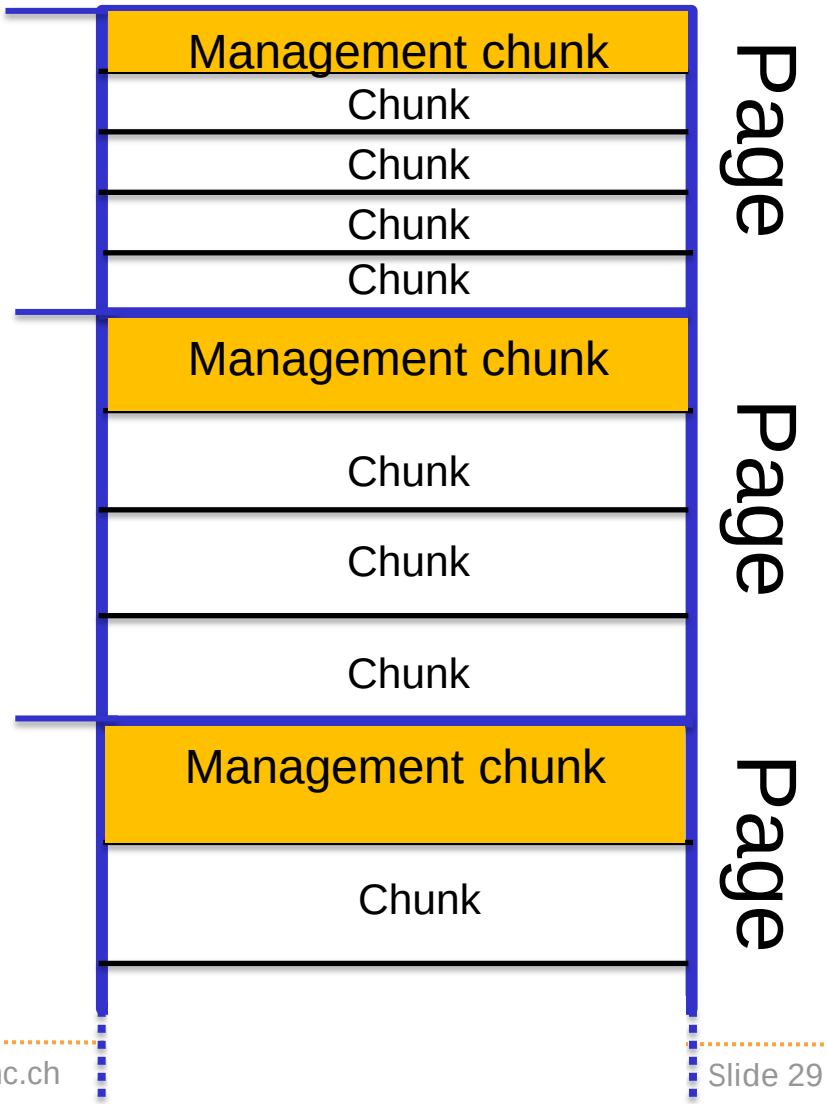
- First chunk has management information
- Management chunk describes other chunks
- Which are free, how big are they etc.

- *(ok, emalloc allocates chunks from the OS, divides them into pages - so the oppositive naming convention. That's a detail).*



Heap Memory Management

Heap could look like this:



Heap Memory Management

But wait, there's more!

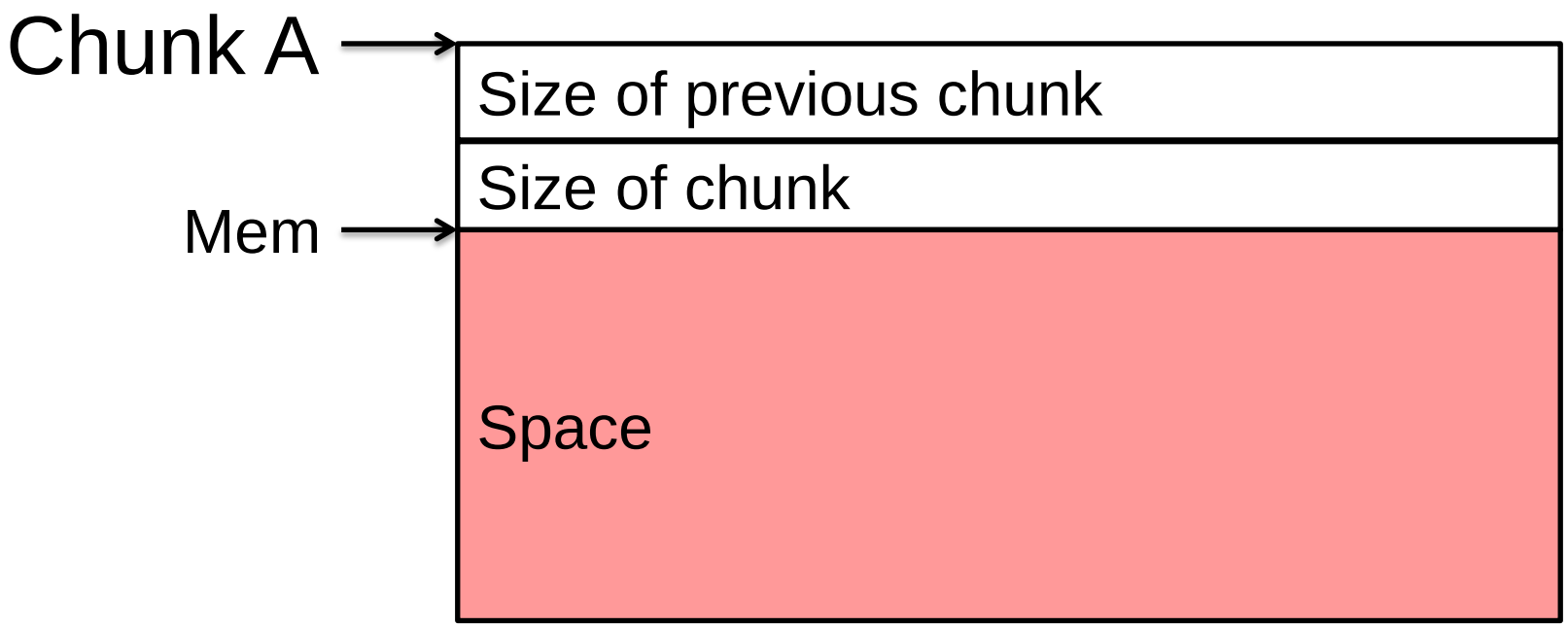
Heap Chunks

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

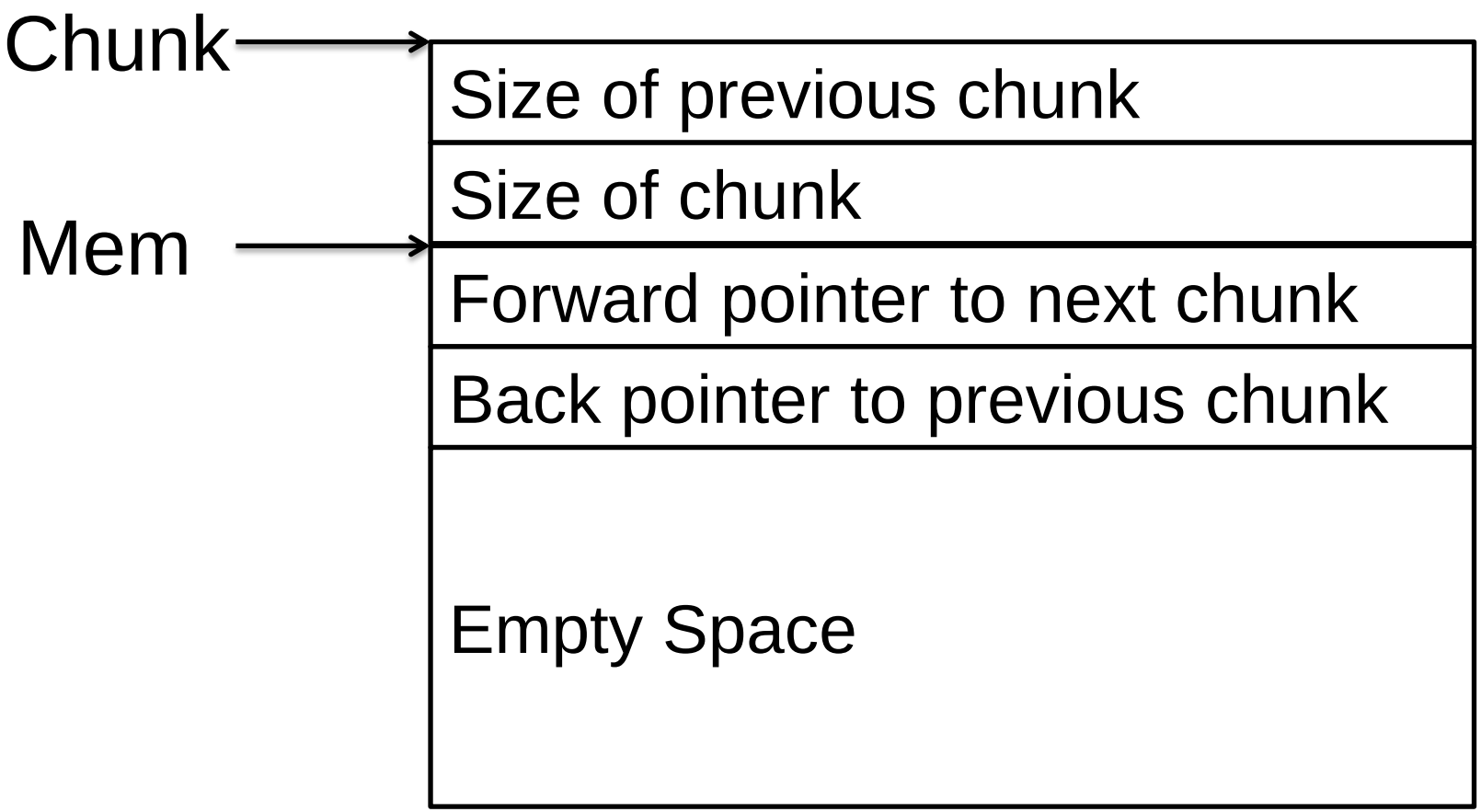
Chunk

Ptmalloc2 chunk:



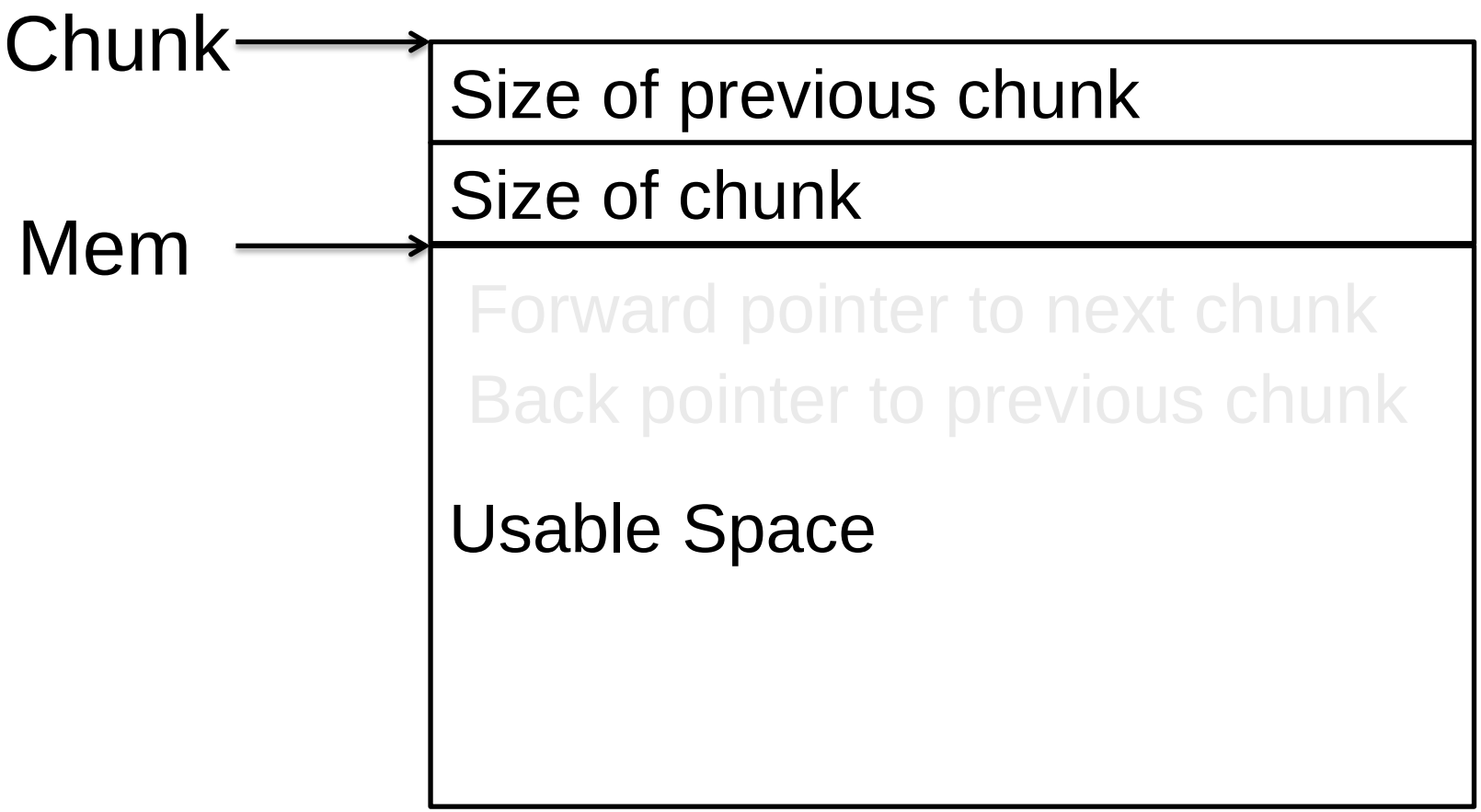
Heap Chunks

Ptmalloc2 FREE chunk:



Heap Chunks

Ptmalloc2 ALLOCATED chunk:



Heap Chunks

Free chunks close to each other get merged

Heap attacks

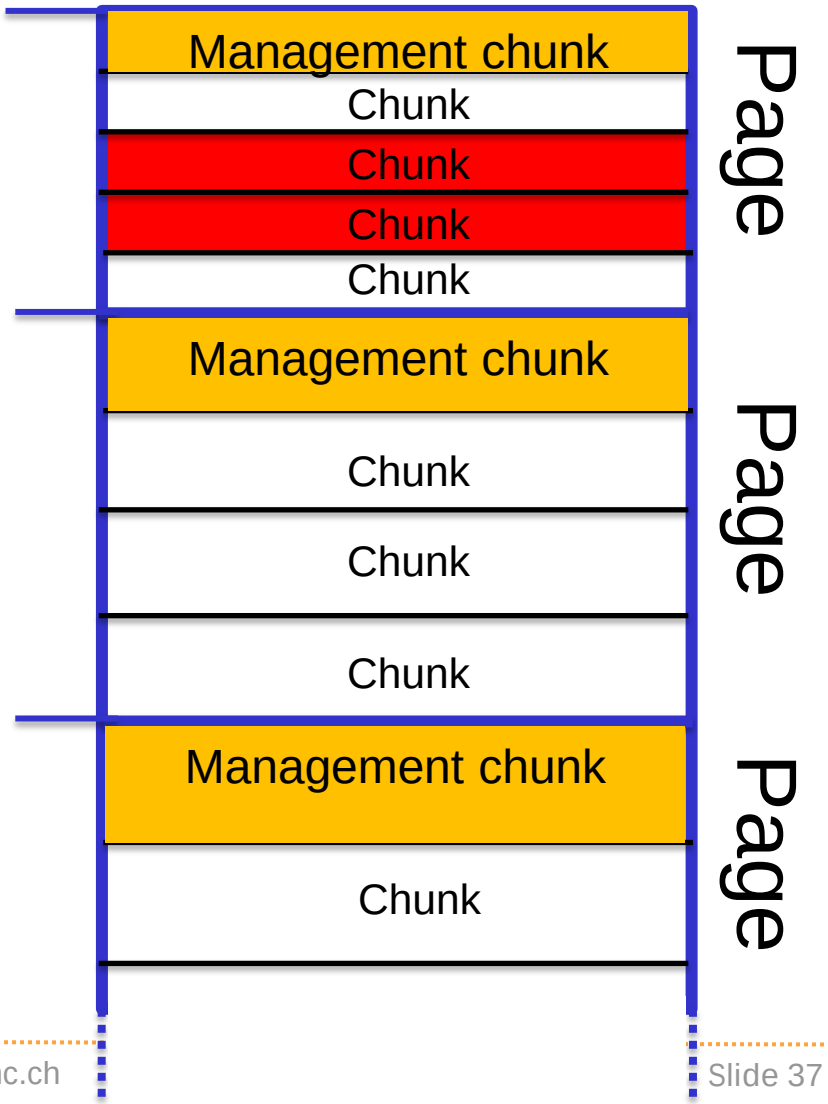
Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

Heap Attacks: Buffer overflow

Heap attack:

Inter-chunk overflow



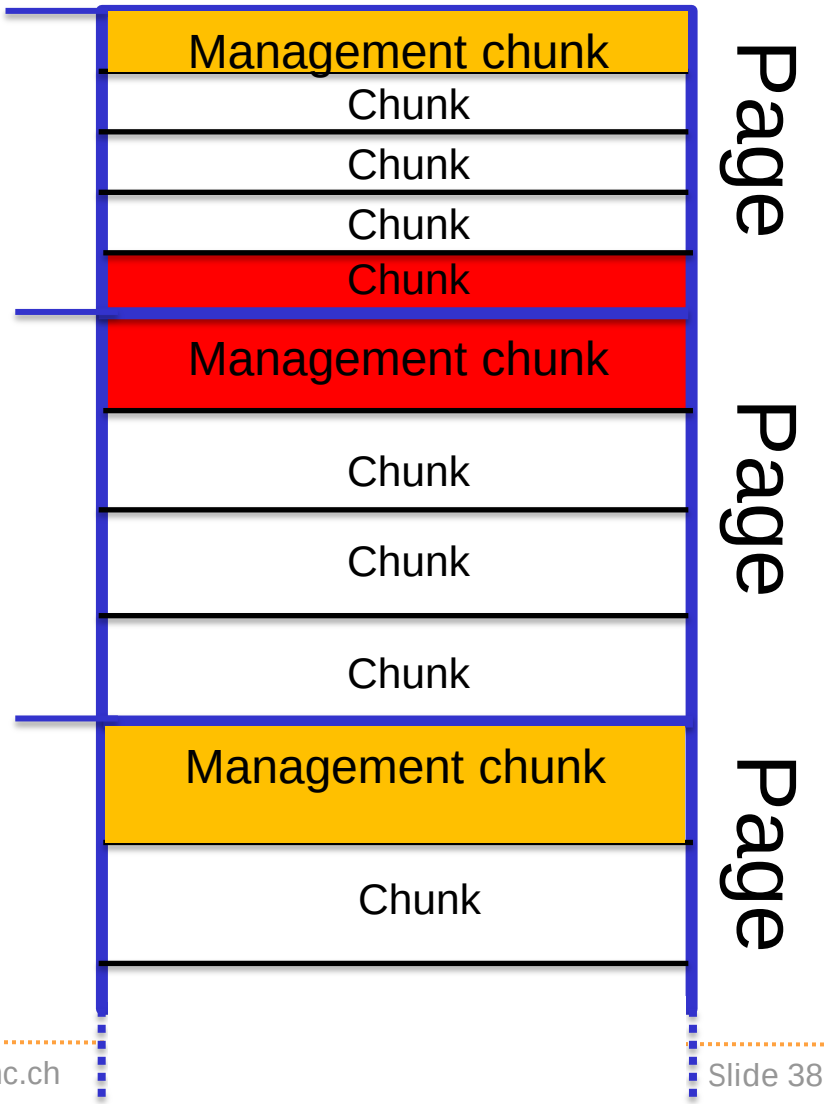
Heap Attacks: Buffer overflow

Heap attack:

Inter-chunk overflow with management chunk

Problem:

- In-band signalling (again)
- Can modify management data of heap allocator
- Therefore, can modify behaviour of heap allocator



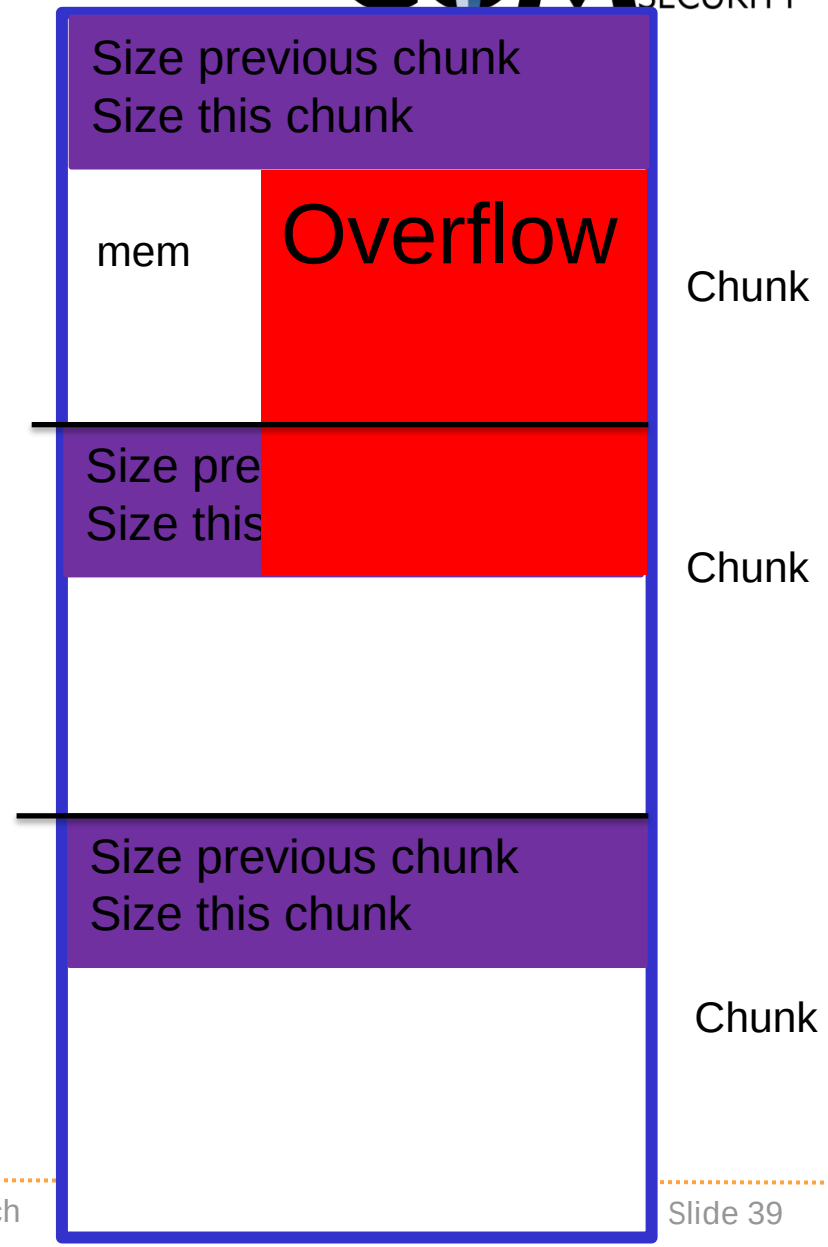
Heap Attacks: Buffer overflow

Heap attack:

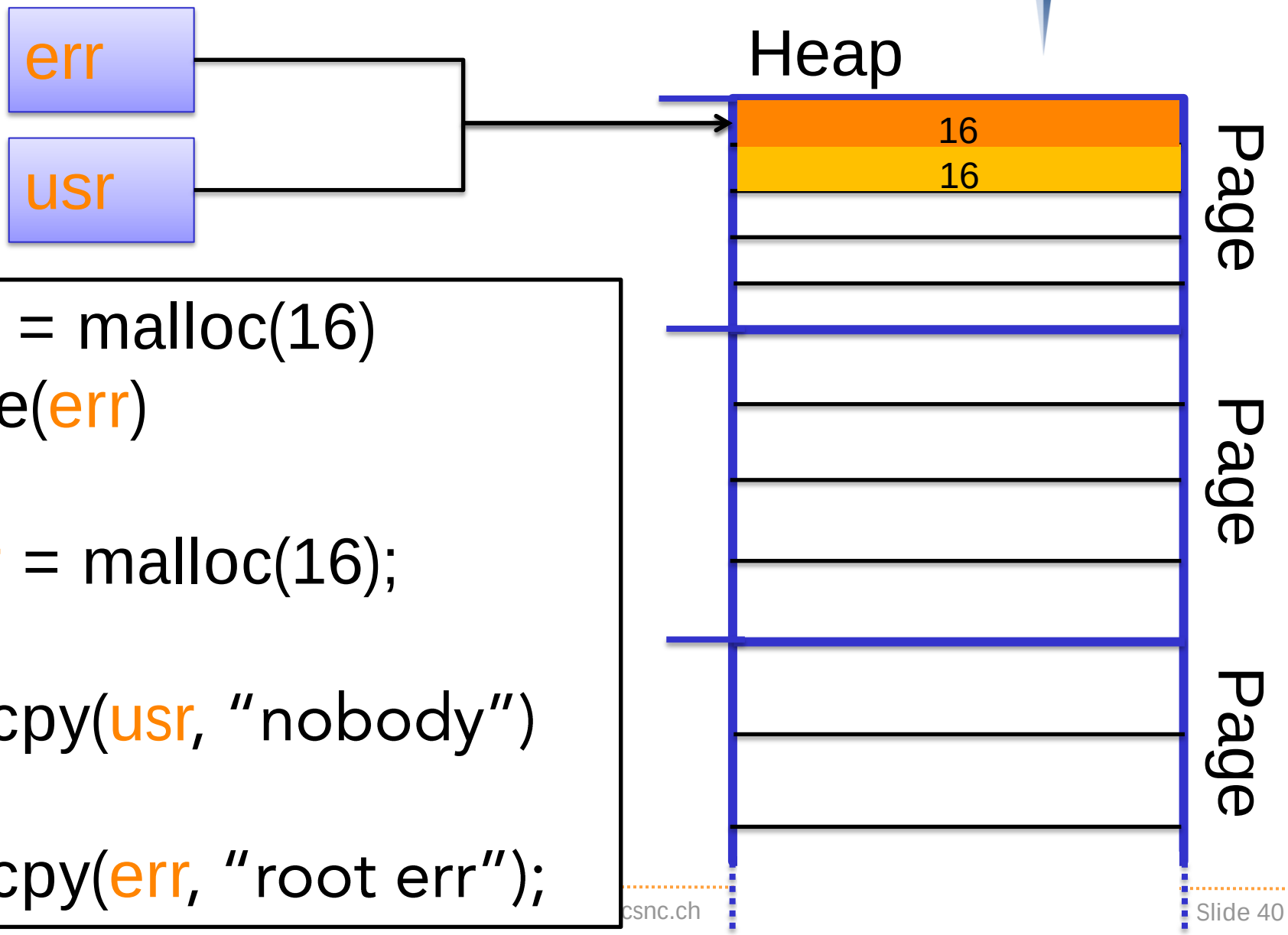
Inter-chunk overflow with chunk metadata

Problem:

- In-band signalling (again)
- Can modify management data of heap allocator
- Therefore, can modify behaviour of heap allocator
  - Create fake chunks
  - Ptmalloc2: Write what where upon free



Heap Attacks: Use after free (UAF)



Use-after-free

```
err = malloc(16)
free(err)

usr = malloc(16);
…
strcpy(usr, “nobody”)
…
strcpy(err, “root err”);
```

Recap:

- A buffer overflow on the heap can modify other buffers on the heap
- A buffer overflow on the heap can influence memory allocator management data structures (junks etc.)

References

Resources:

http://homes.soic.indiana.edu/yh33/Teaching/I433-2016/lec13-HeapAttacks.pdf

http://www.pwntester.com/blog/2014/03/23/codegate-2k14-4stone-pwnable-300-write-up/