What's Eating my RAM?

JIANFEI PAN

Cppcon
The C++ Conference

2024
September 15 - 20

What’s Eating My RAM?

Bloomberg

Engineering

CppCon 2024
September 17, 2024

Jianfei Pan
Software Engineer, Portfolio/Risk Analytics

TechAtBloomberg.com

A story

🚨 **90% Memory Used:** What’s that alarm?

🔍**Back to basics:** How does my code impact memory usage?

👷 **Memory allocation troubleshooting & tools:** Leak & Fragmentation

🚨 **90% Memory Used:**
What’s that alarm?

What’s that alarm?

🚨 90% Memory Used

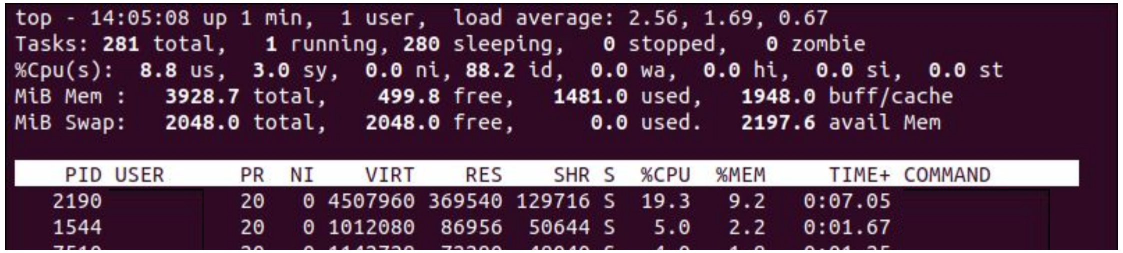
Consequences:

- **Swap**: performance degradation
- **Out-of-memory (OOM) killer**: service disruption
- **Multi-tenant environment**: resources are shared by different processes

What’s that alarm?

🚨 90% Memory Used

~$ top -o RES



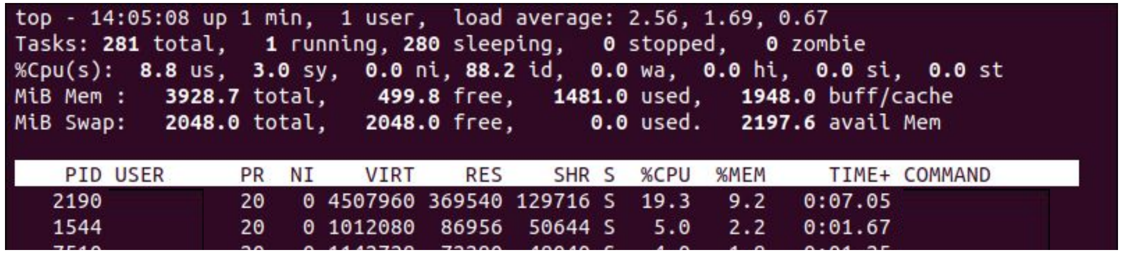
```
top - 14:05:08 up 1 min,  1 user,  load average: 2.56, 1.69, 0.67
Tasks: 281 total,   1 running, 280 sleeping,   0 stopped,   0 zombie
%Cpu(s):  8.8 us,  3.0 sy,  0.0 ni, 88.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3928.7 total,    499.8 free,   1481.0 used,   1948.0 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   2197.6 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   2190           20   0 4507960 369540 129716 S  19.3   9.2   0:07.05
   1544           20   0 1012080  86956  50644 S   5.0   2.2   0:01.67
```

What’s that alarm?

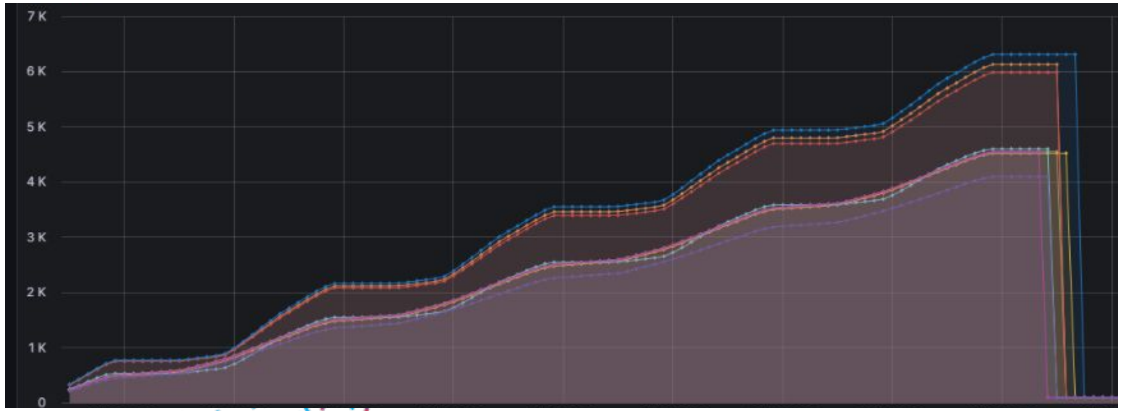
🚨 90% Memory Used

~$ top -o RES



```
top - 14:05:08 up 1 min,  1 user,  load average: 2.56, 1.69, 0.67
Tasks: 281 total,   1 running, 280 sleeping,   0 stopped,   0 zombie
%Cpu(s):  8.8 us,  3.0 sy,  0.0 ni, 88.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3928.7 total,    499.8 free,   1481.0 used,   1948.0 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   2197.6 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   2190 [redacted] 20   0 4507960 369540 129716 S  19.3   9.2   0:07.05 [redacted]
   1544           20   0 1012080  86956  50644 S   5.0   2.2   0:01.67
```



🔍Back to basics:

How does my code impact memory usage?

How does my code impact memory usage?

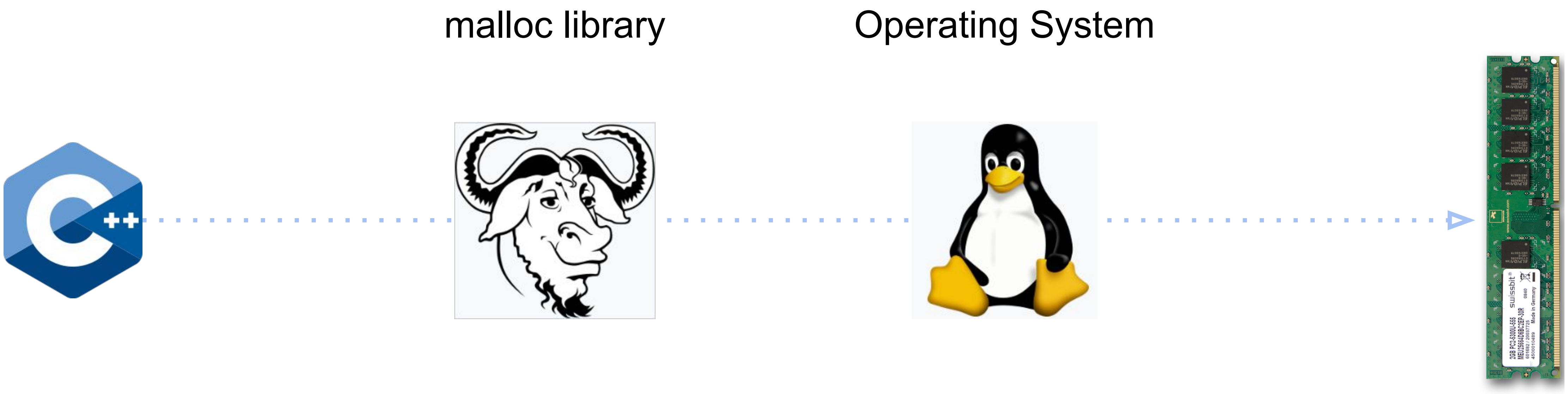
```
new std::string("Hello");
```

?

90 %

How does my code impact memory usage?

How does my code impact memory usage?

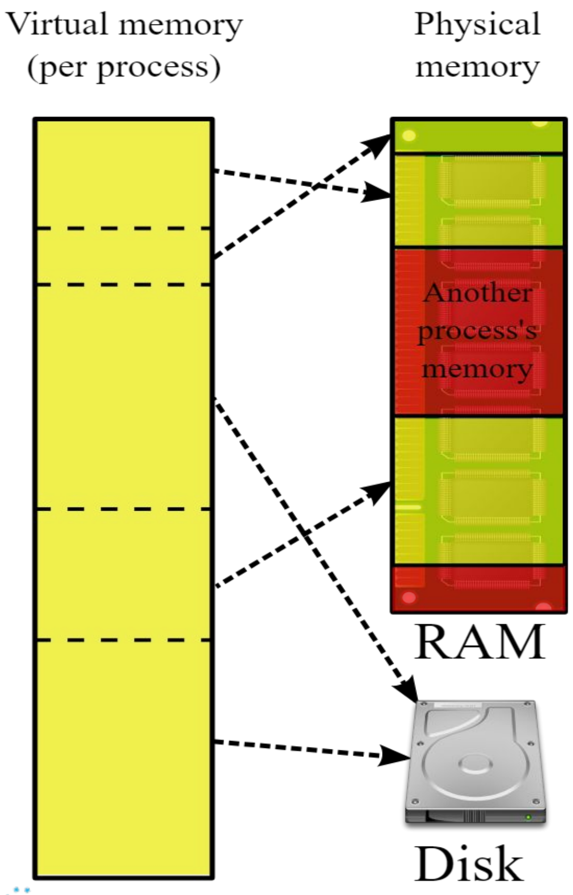


malloc library

Operating System

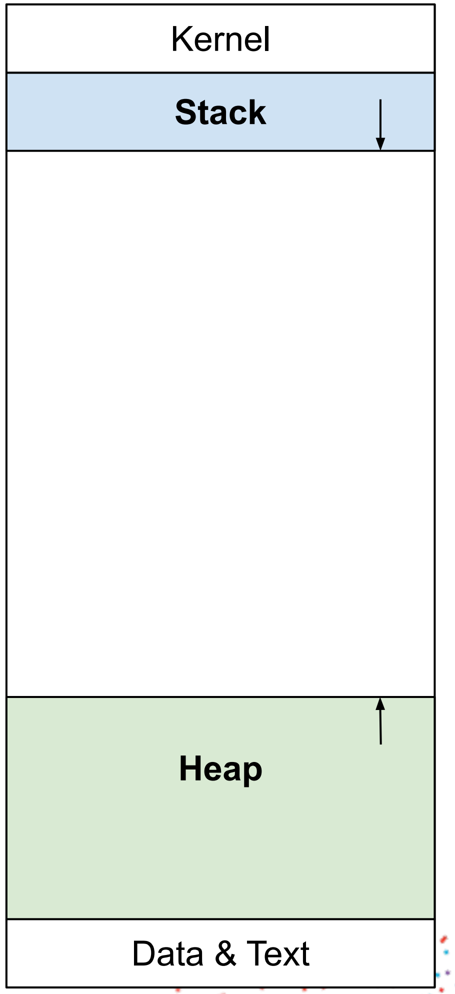
How does my code impact memory usage?

Operating System

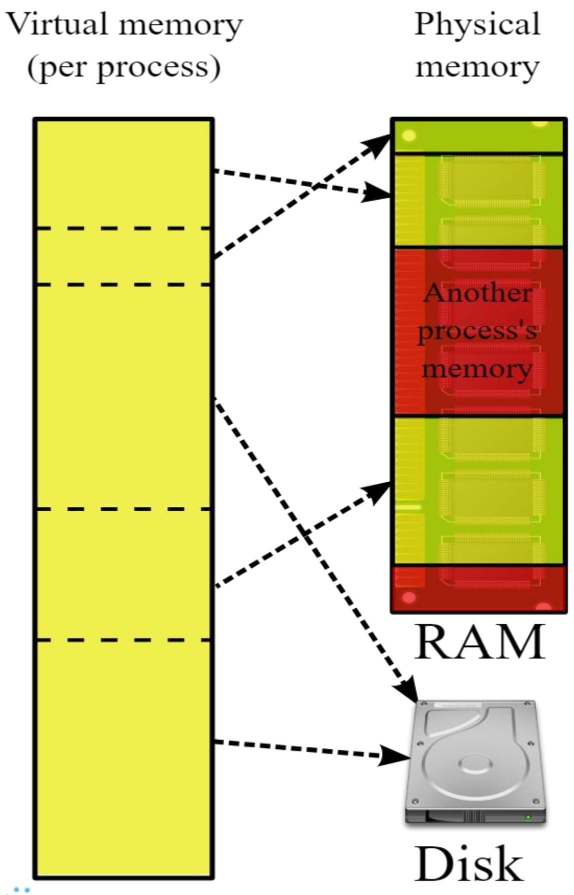


How does my code impact memory usage?

glibc's malloc library



Operating System

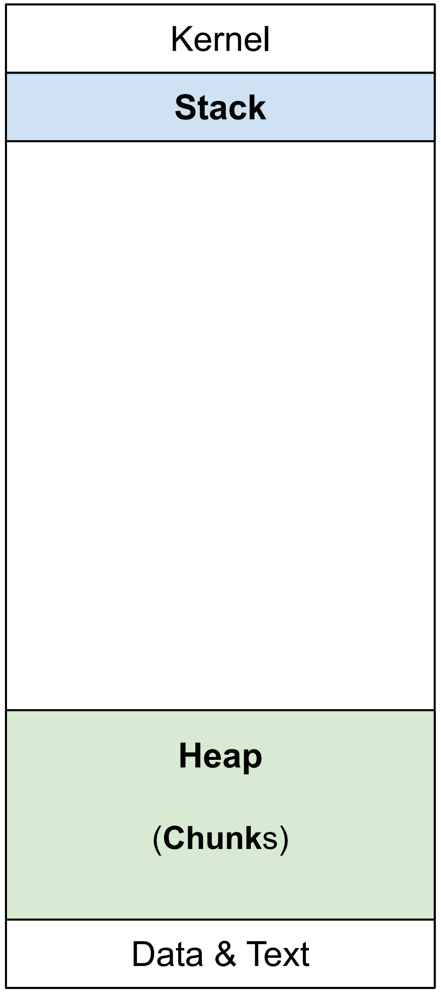


How does my code impact memory usage?

malloc

Heap:
*a **contiguous** region of memory subdivided into chunks*

Chunk:
a range of memory of various sizes *allocated to the application*



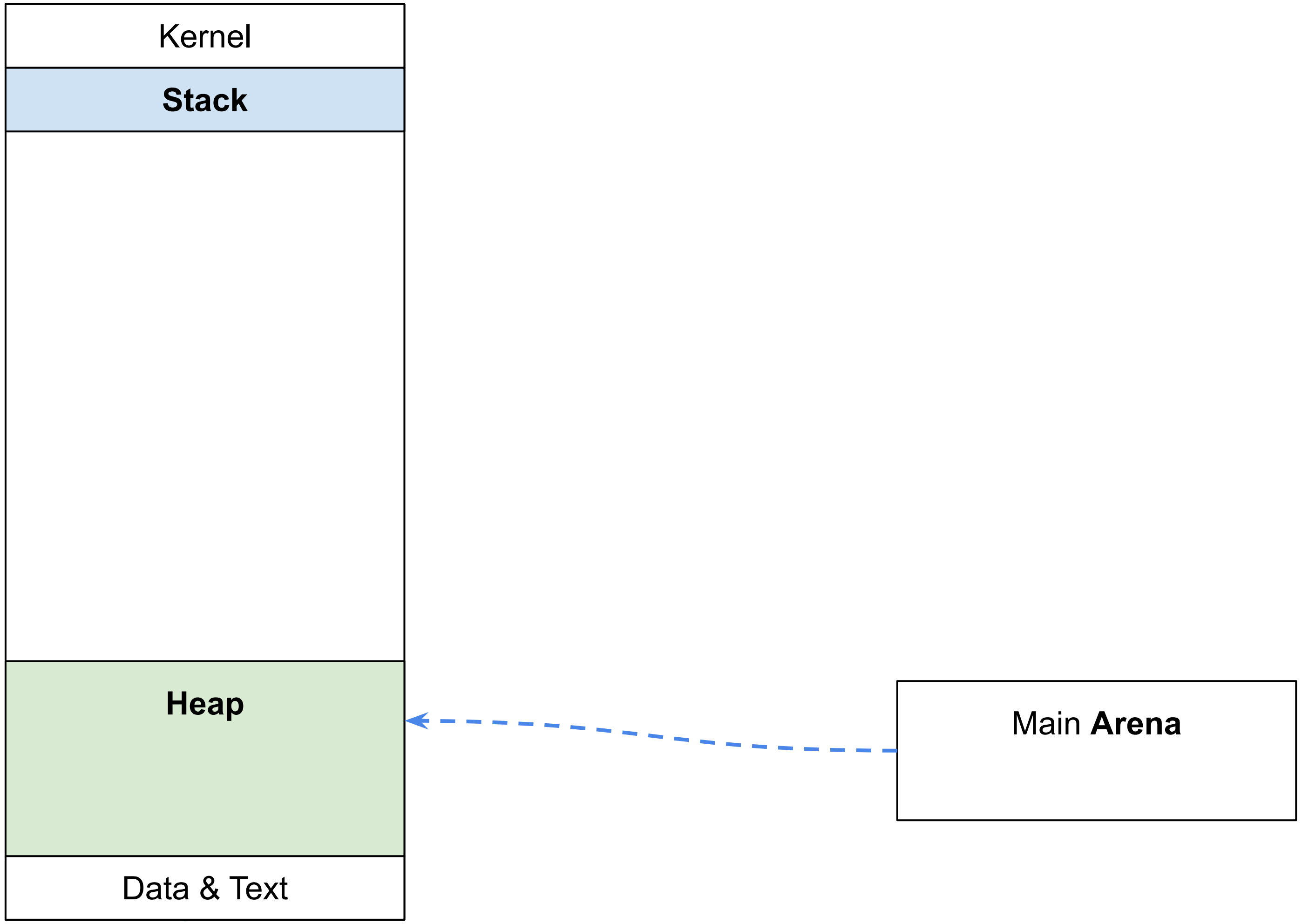
How does my code impact memory usage?

malloc

Arena:
a structure that is shared among one or more threads

Heap:
*a **contiguous** region of memory subdivided into chunks*

Chunk:
a range of memory of various sizes *allocated to the application*



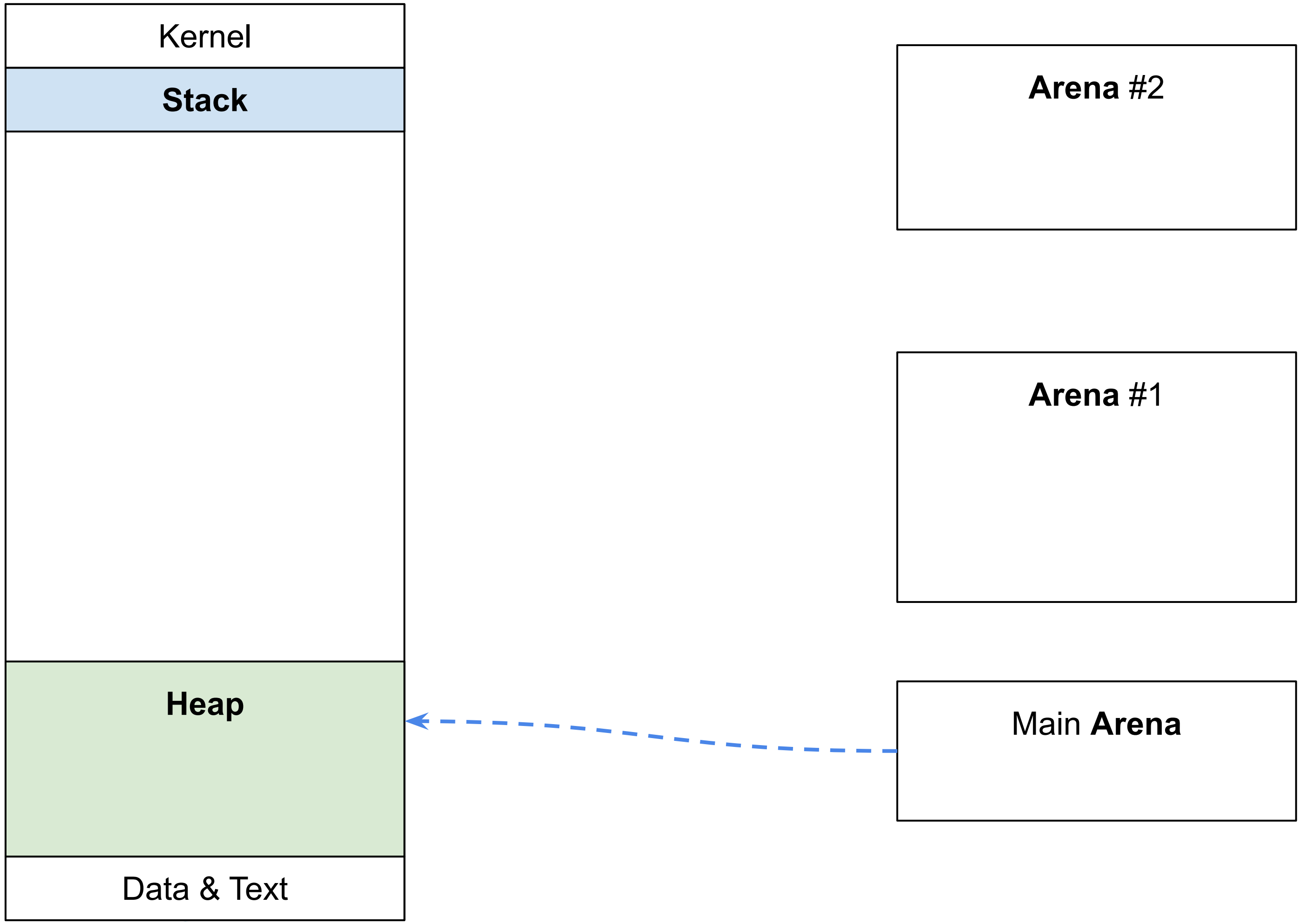
How does my code impact memory usage?

malloc

Arena:
a structure that is shared among one or more threads

Heap:
a ***contiguous*** *region of memory subdivided into chunks*

Chunk:
a range of memory of various sizes *allocated to the application*



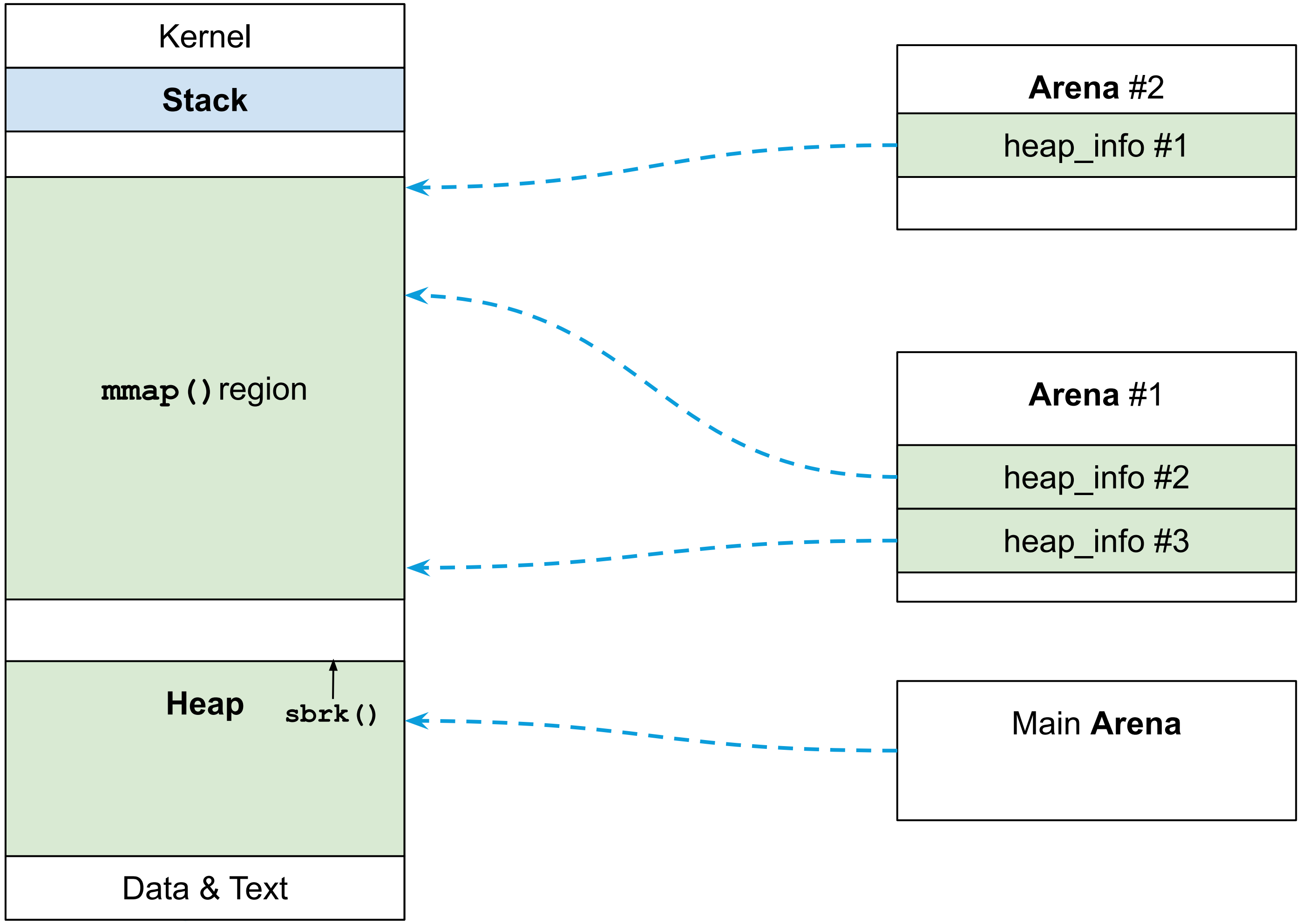
How does my code impact memory usage?

malloc

Arena:
a structure that is shared among one or more threads

Heap:
a ***contiguous*** *region of memory subdivided into chunks*

Chunk:
a range of memory of various sizes *allocated to the application*



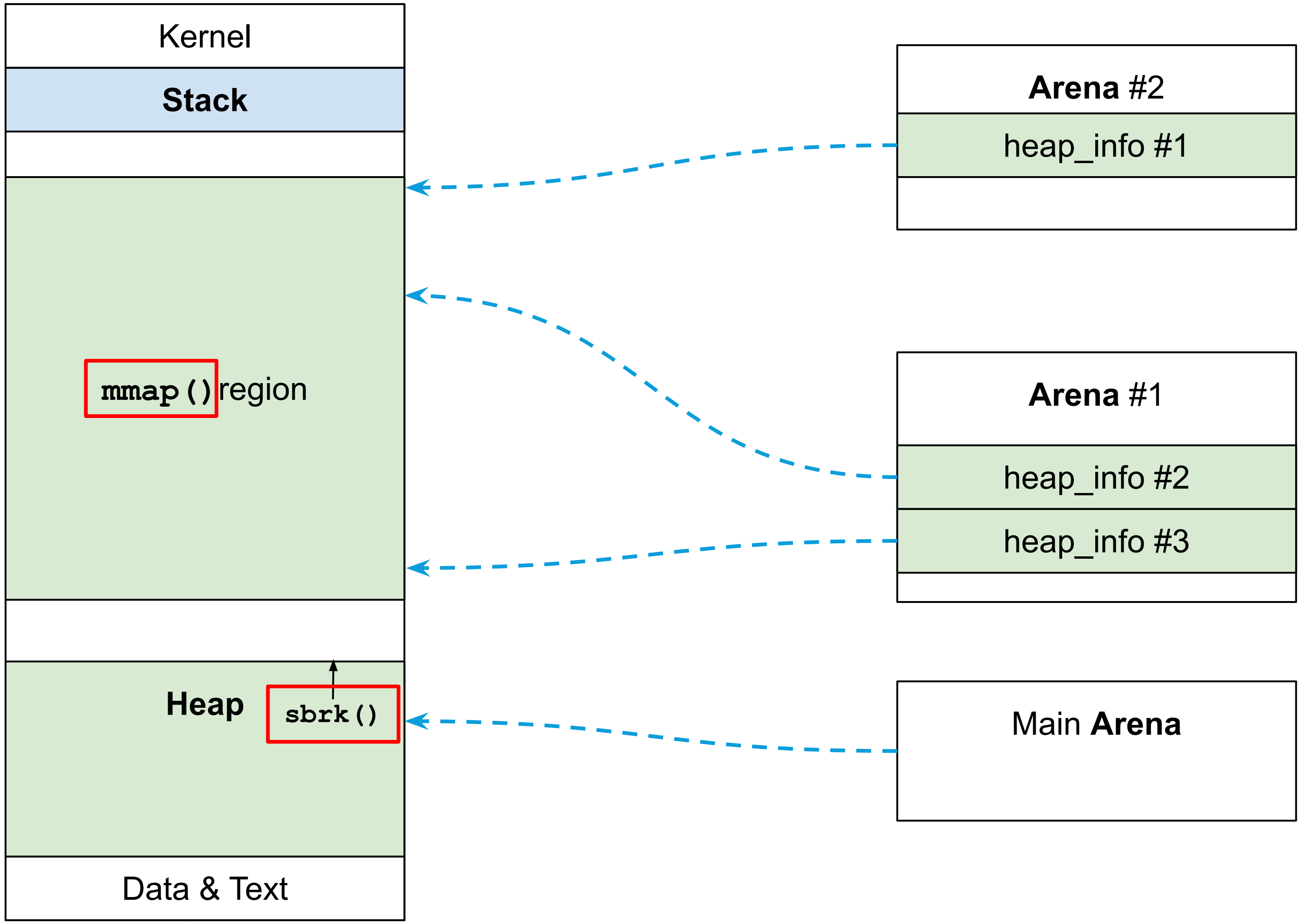
How does my code impact memory usage?

malloc

Arena:
a structure that is shared among one or more threads

Heap:
a ***contiguous*** *region of memory subdivided into chunks*

Chunk:
a range of memory of various sizes *allocated to the application*

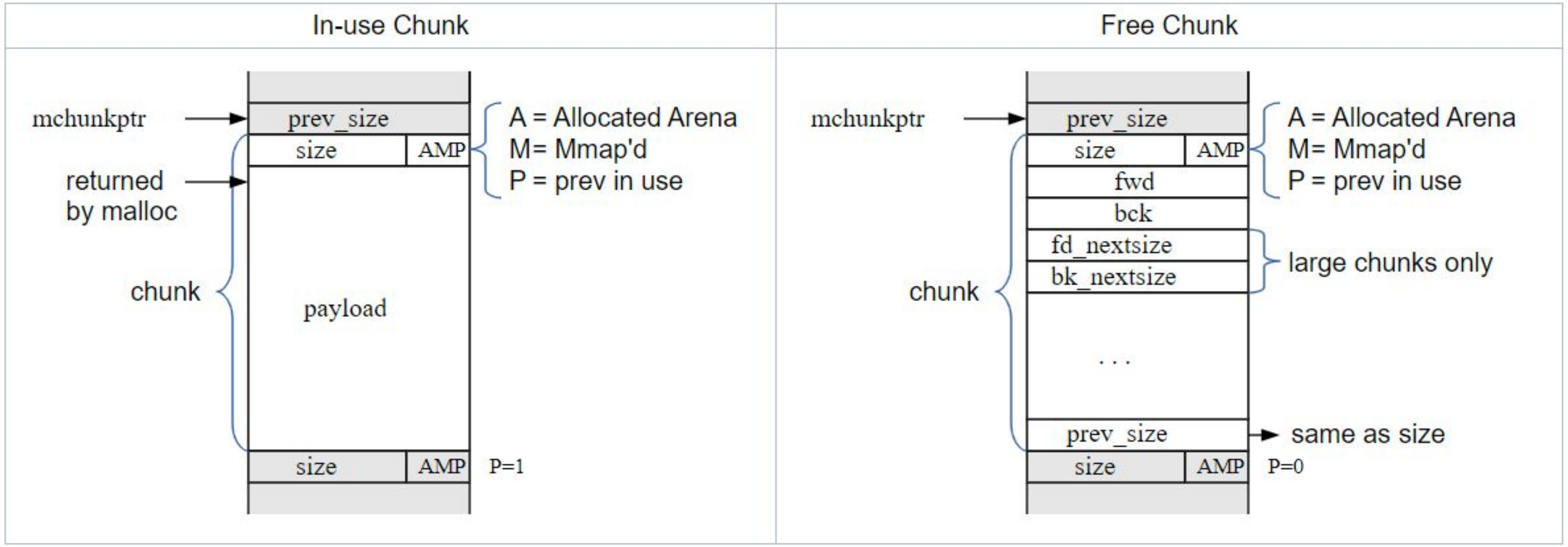


How does my code impact memory usage?

Chunk

can be:

- allocated: in-use chunk
- freed: (marked as) free chunk
- combined with adjacent free chunks

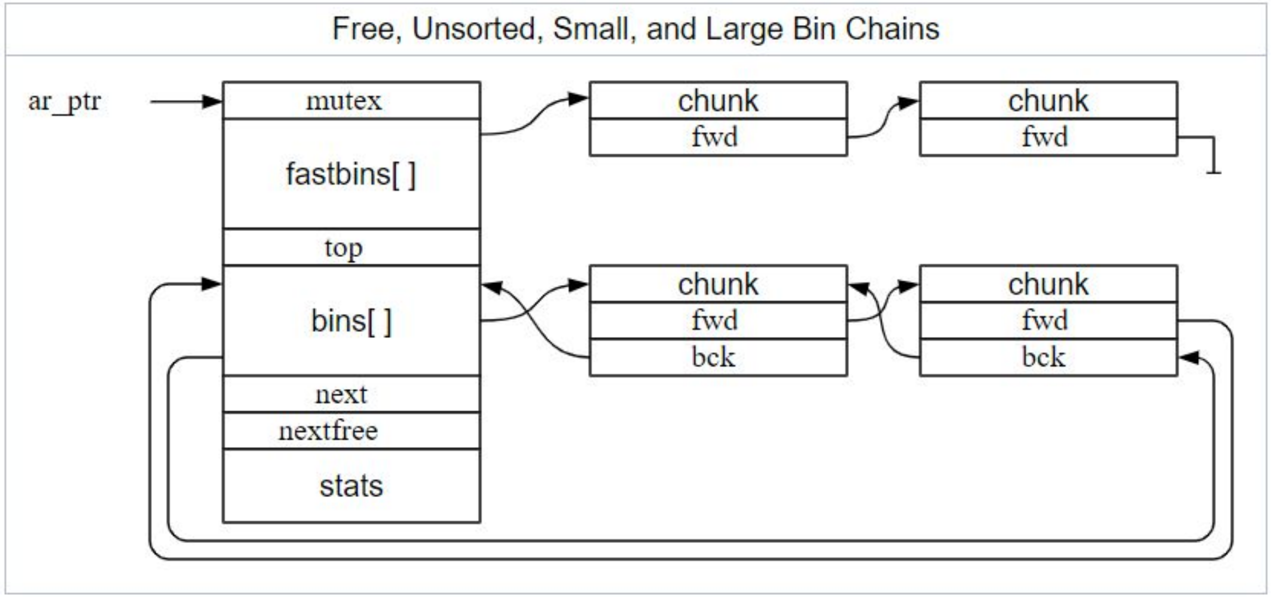


Sourced from https://sourceware.org/glibc/wiki/MallocInternals.

How does my code impact memory usage?

***`free()`** marks a chunk as "free to be reused",*
from the OS' point of view: the memory still "belongs" to the application.

Bins manage free chunks of different sizes



malloc.c

```
    64 bins of size       8   bytes
    32 bins of size      64   bytes
    16 bins of size     512   bytes
     8 bins of size    4096   bytes
     4 bins of size   32768   bytes
     2 bins of size  262144   bytes
     1 bin  of size what's left
```

How does my code impact memory usage?

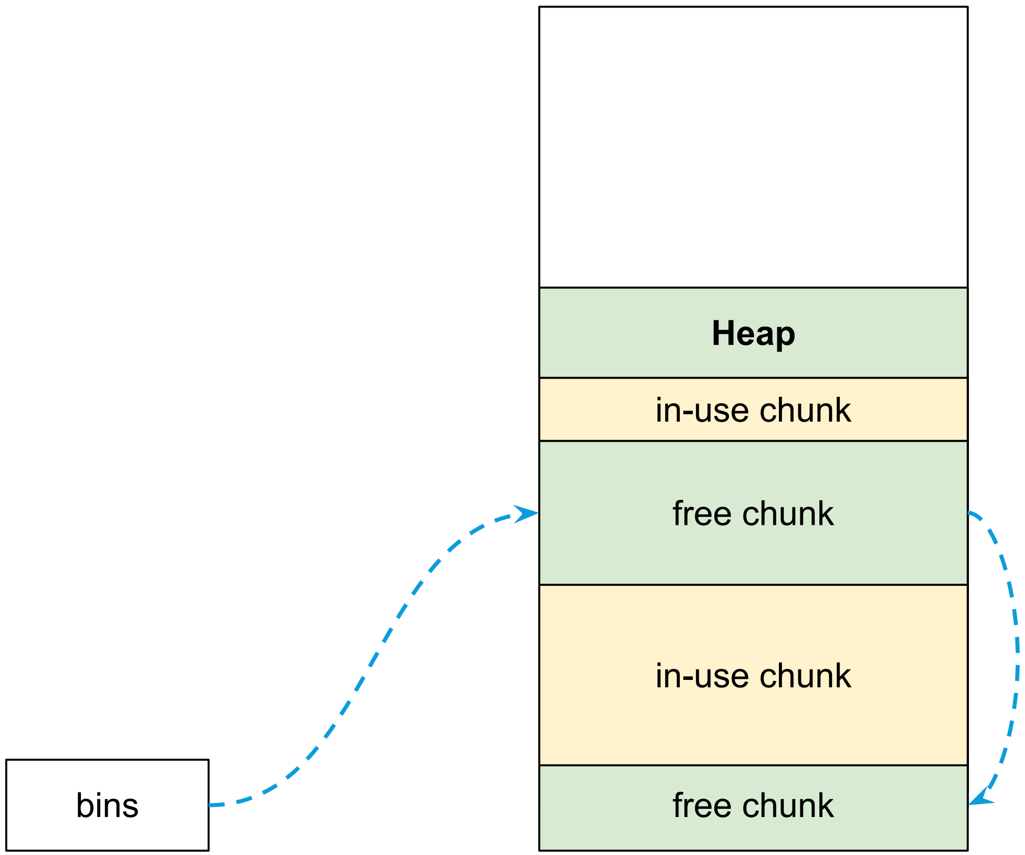
malloc

malloc algorithm:

-
-
-

Free algorithm:

-
-
-



How does my code impact memory usage?

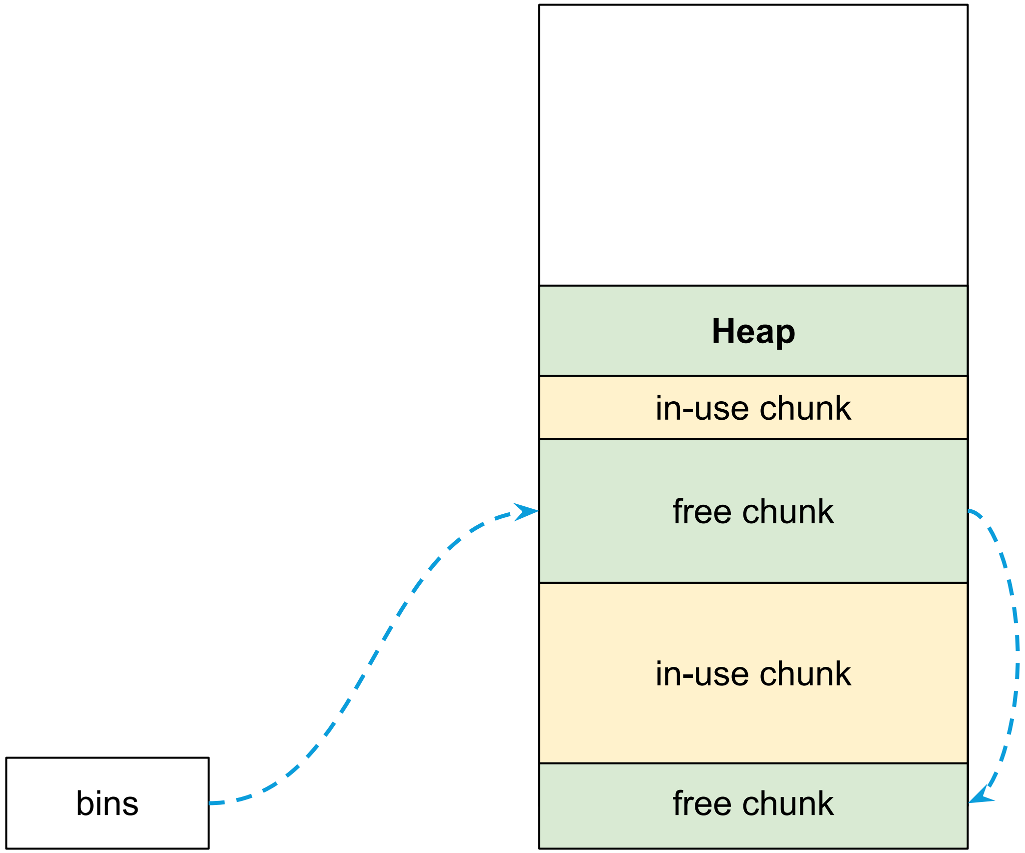
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it,
-
-

Free algorithm:

-
-
-



How does my code impact memory usage?

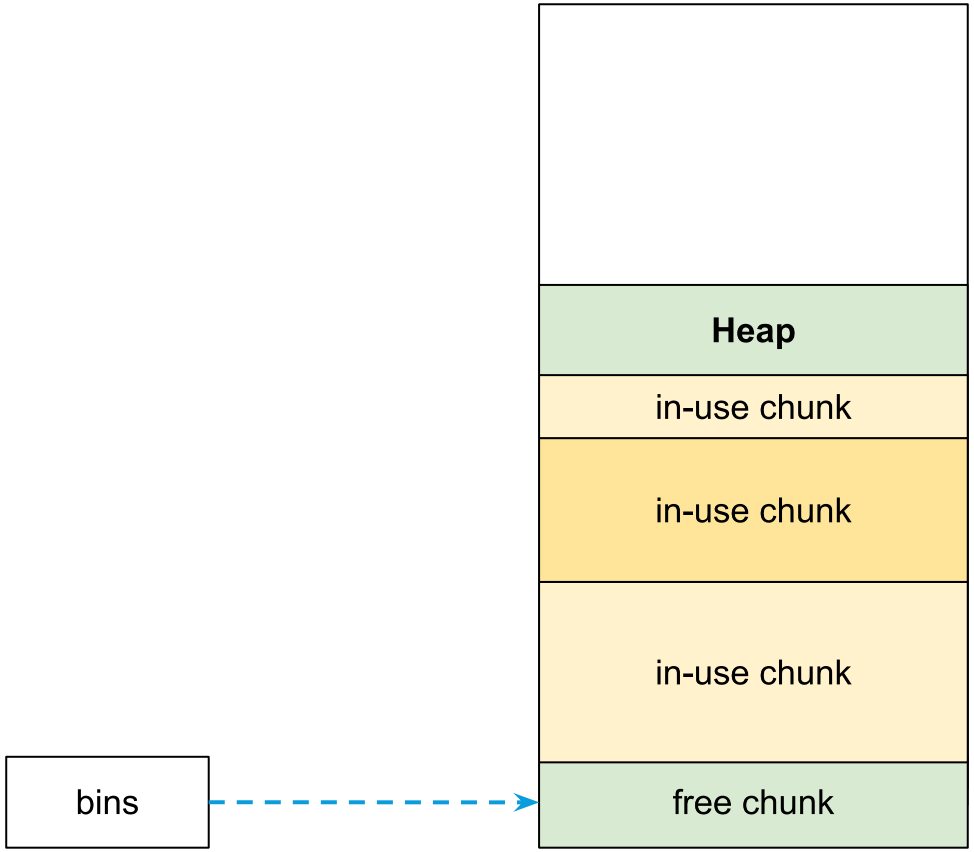
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it, use that
-
-

Free algorithm:

-
-
-



How does my code impact memory usage?

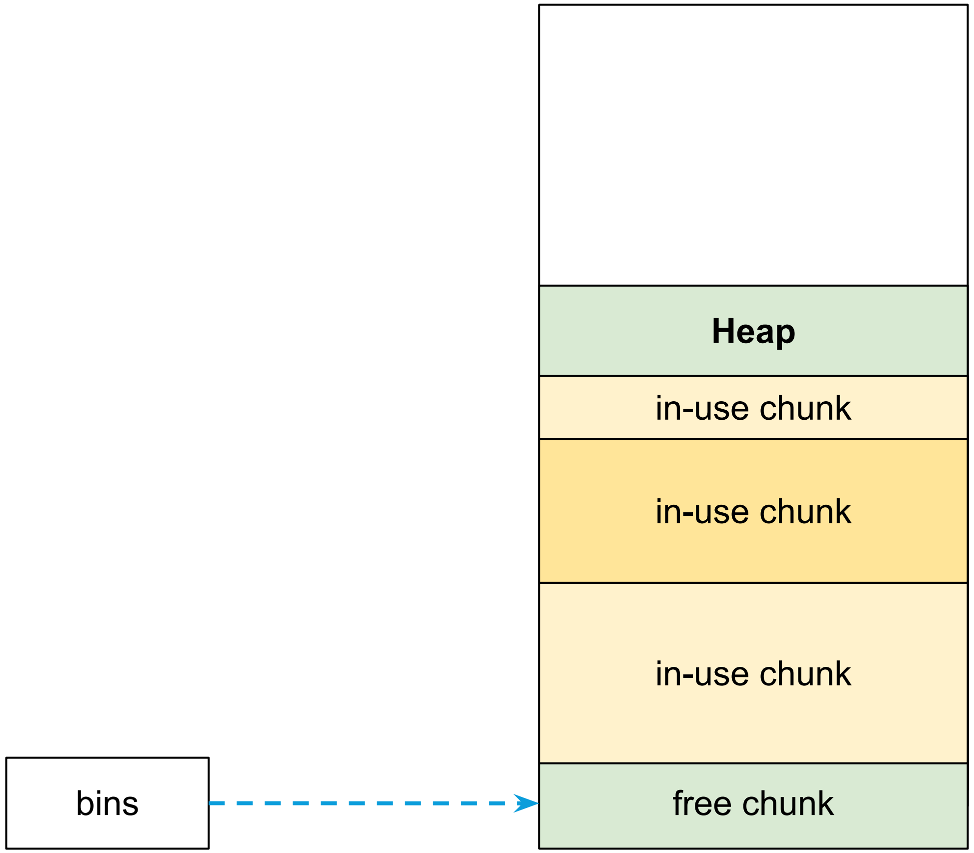
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it, use that

Free algorithm:

- **Place the free chunk in the appropriate bin**



How does my code impact memory usage?

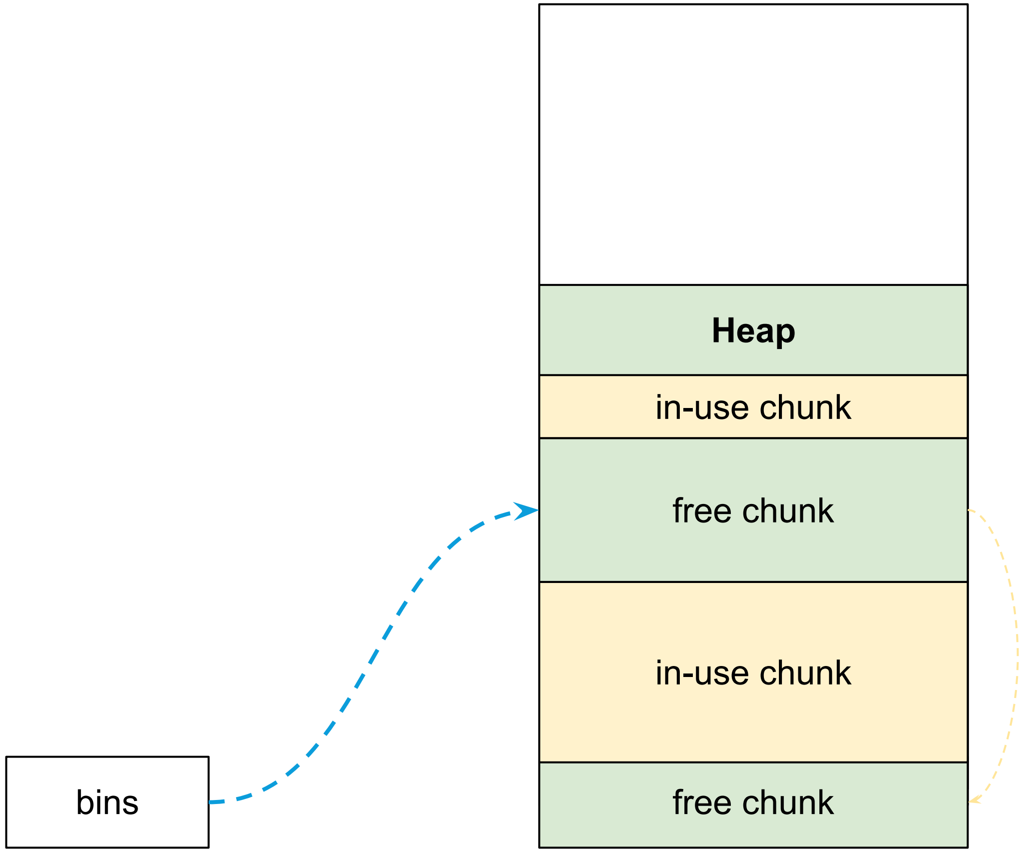
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it, use that
-
-

Free algorithm:

- **Place the free chunk in the appropriate bin**
-
-



How does my code impact memory usage?

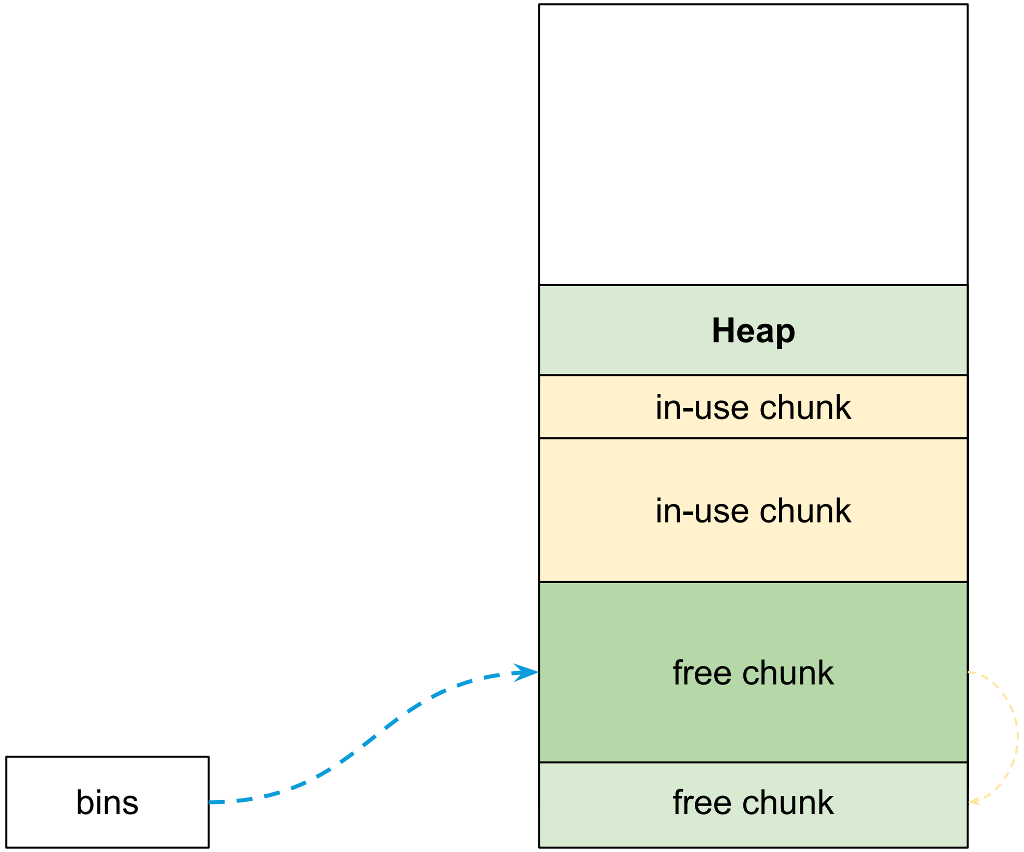
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it, use that
-
-

Free algorithm:

- Place the free chunk in the appropriate bin
- **If this chunk is adjacent to another free chunk,**
-



How does my code impact memory usage?

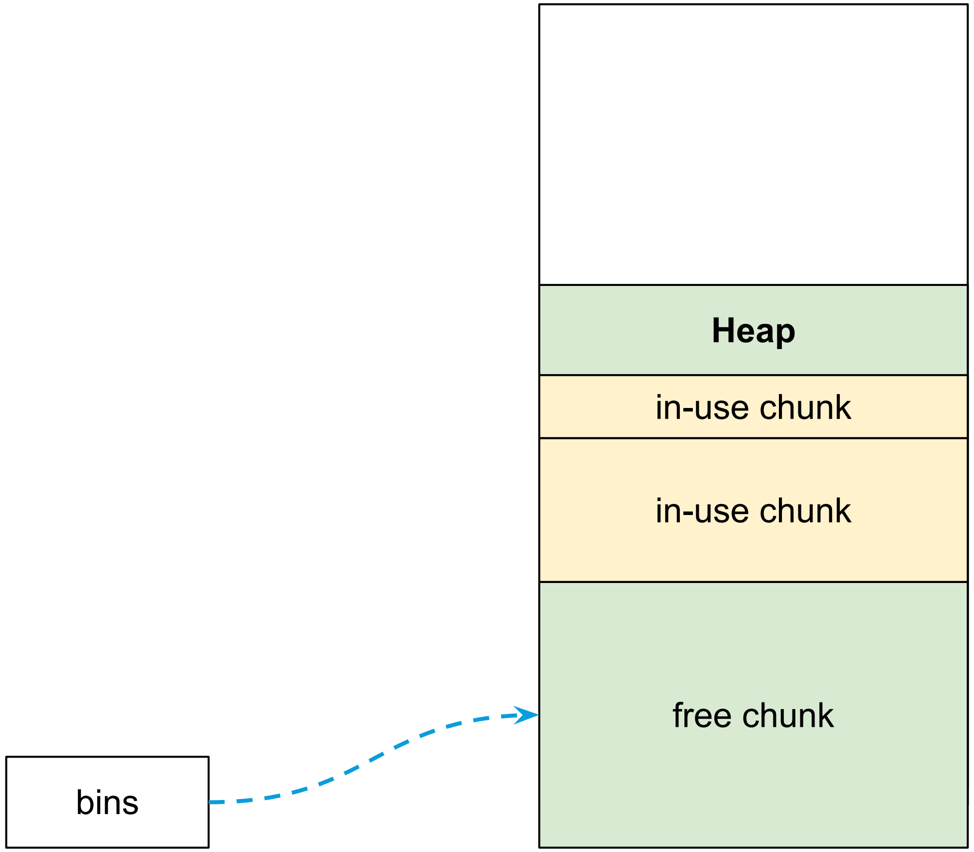
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it, use that
-
-

Free algorithm:

- Place the free chunk in the appropriate bin
- **If this chunk is adjacent to another free chunk, combine**
-



How does my code impact memory usage?

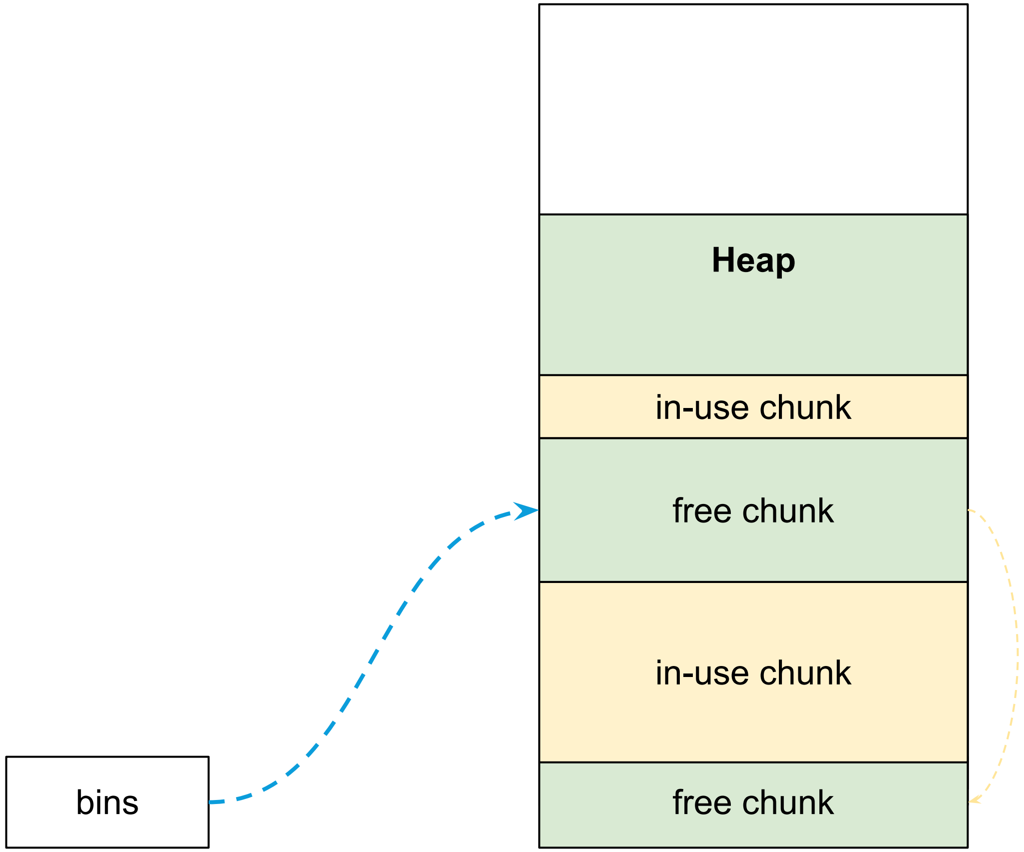
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it, use that
- **If no chunk is available,**
-

Free algorithm:

- Place the free chunk in the appropriate bin
- If this chunk is adjacent to another free chunk, combine
-



How does my code impact memory usage?

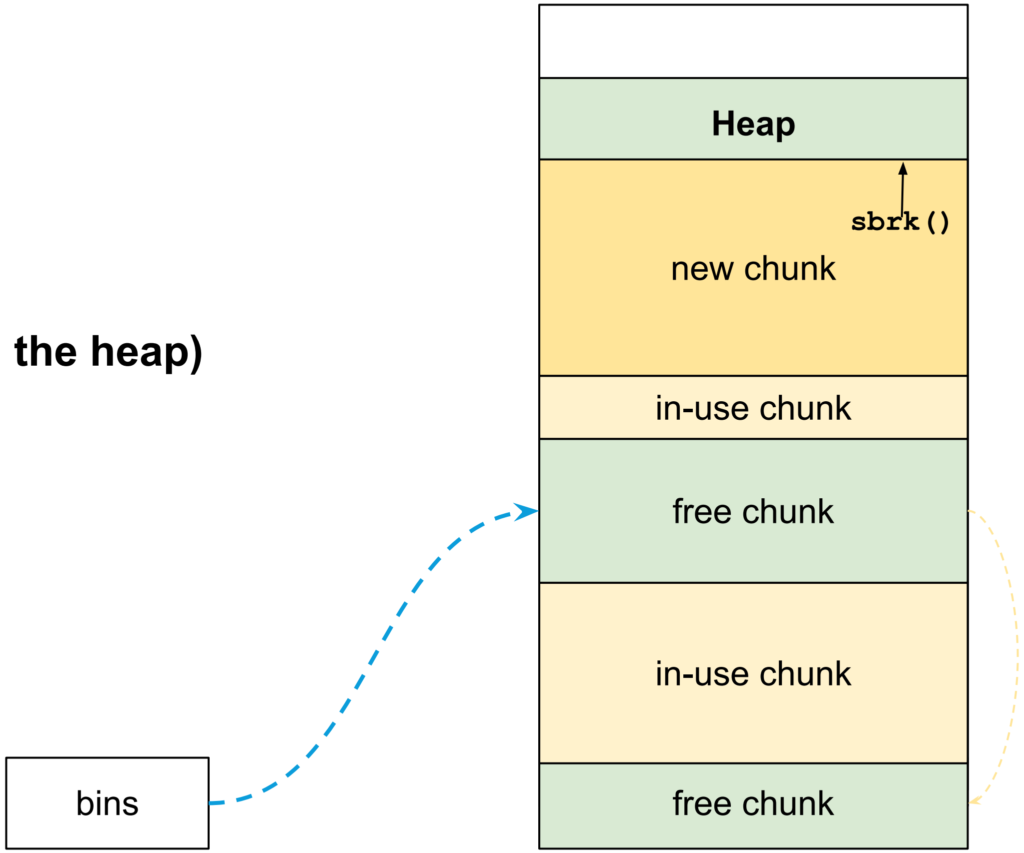
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it, use that
- **If no chunk is available, create a new chunk(`sbrk()`: extends the heap)**
-

Free algorithm:

- Place the free chunk in the appropriate bin
- If this chunk is adjacent to another free chunk, combine
-



How does my code impact memory usage?

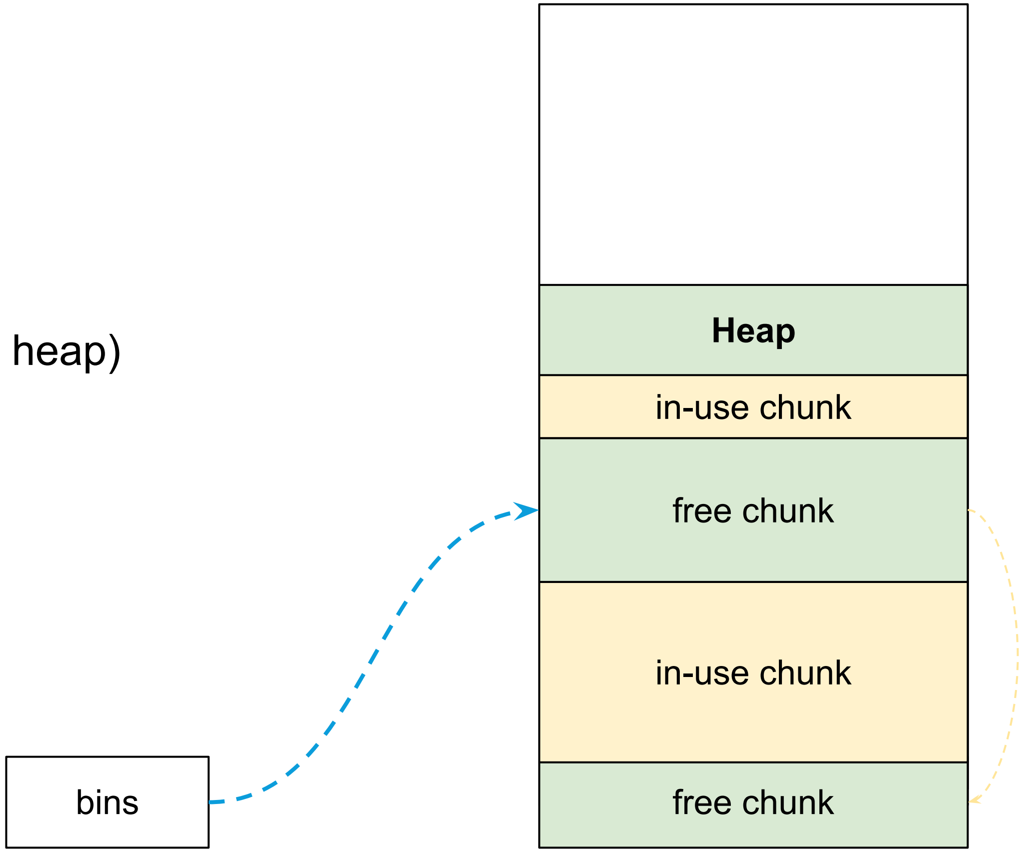
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it, use that
- If no chunk is available, create a new chunk(**`sbrk()`**: extends the heap)
- ● **If the request is large enough (`M_MMAP_THRESHOLD`):**

Free algorithm:

- Place the free chunk in the appropriate bin
- If this chunk is adjacent to another free chunk, combine
-



How does my code impact memory usage?

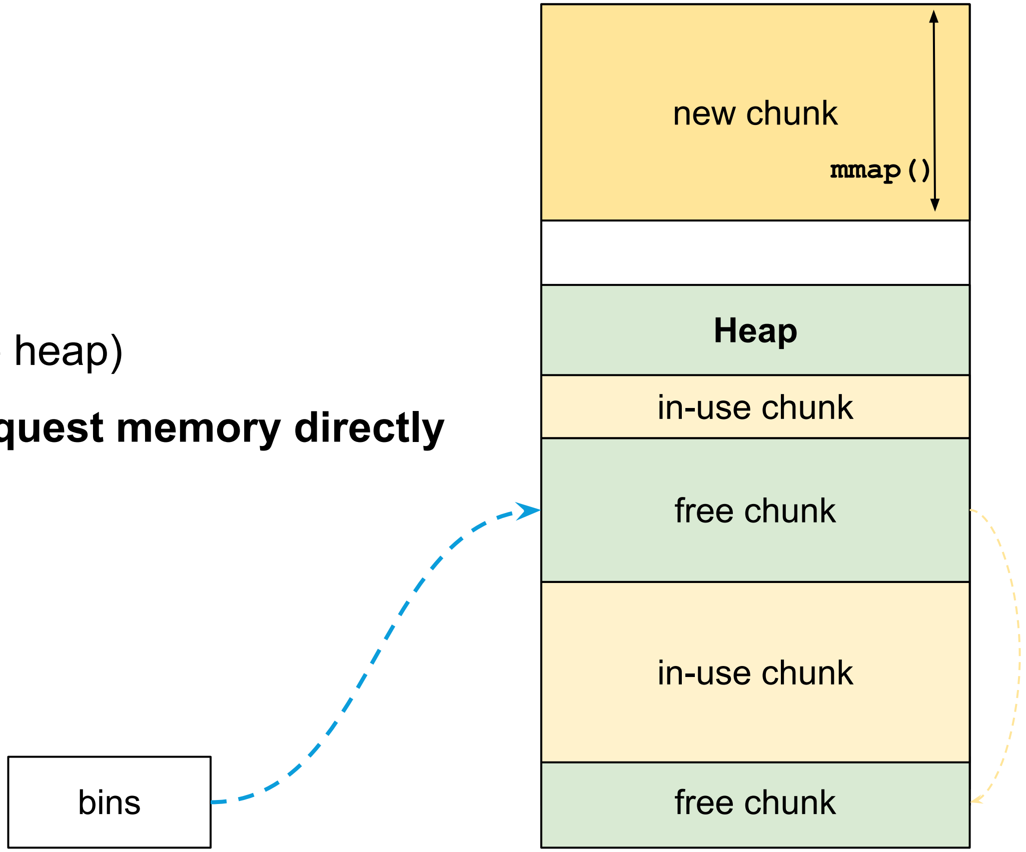
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it, use that
- If no chunk is available, create a new chunk(**`sbrk()`**: extends the heap)
- 🟢 **If the request is large enough (`M_MMAP_THRESHOLD`): `mmap` request memory directly from OS**

Free algorithm:

- Place the free chunk in the appropriate bin
- If this chunk is adjacent to another free chunk, combine
-



How does my code impact memory usage?

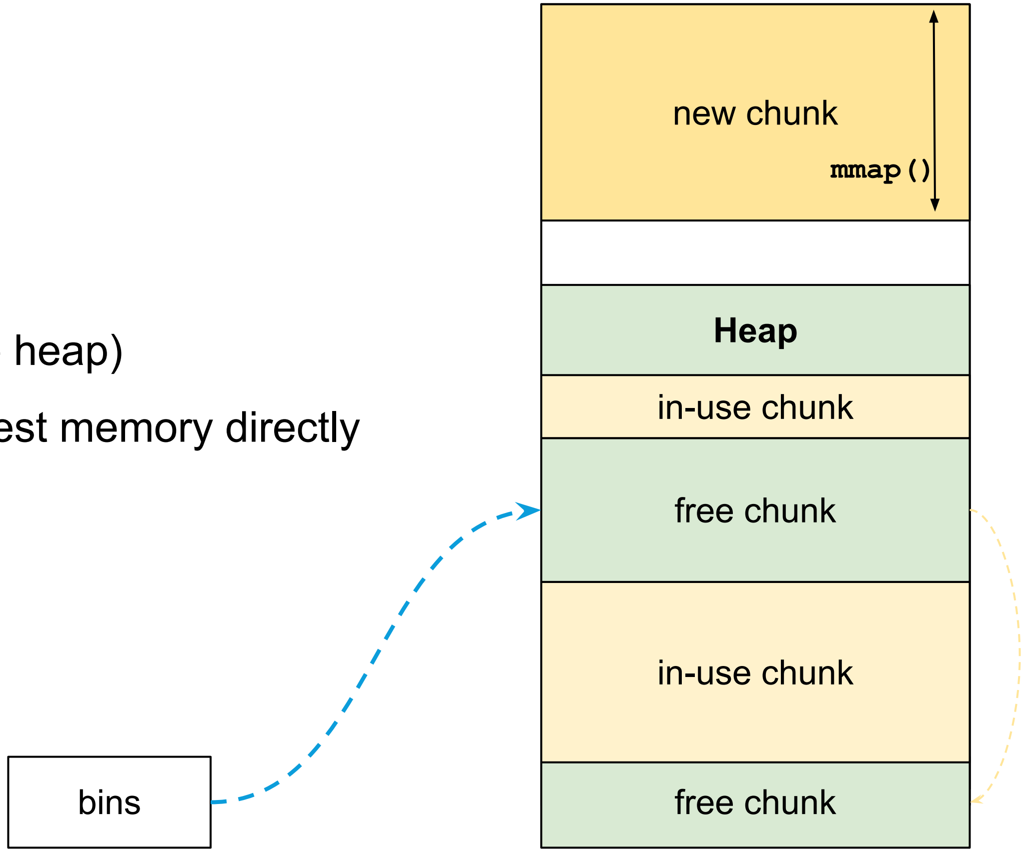
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it, use that
- If no chunk is available, create a new chunk(**`sbrk()`**: extends the heap)
- If the request is large enough (**`M_MMAP_THRESHOLD`**): **`mmap`** request memory directly from OS

Free algorithm:

- Place the free chunk in the appropriate bin
- If this chunk is adjacent to another free chunk, combine
- **If this chunk is mapped: `munmap`**



How does my code impact memory usage?

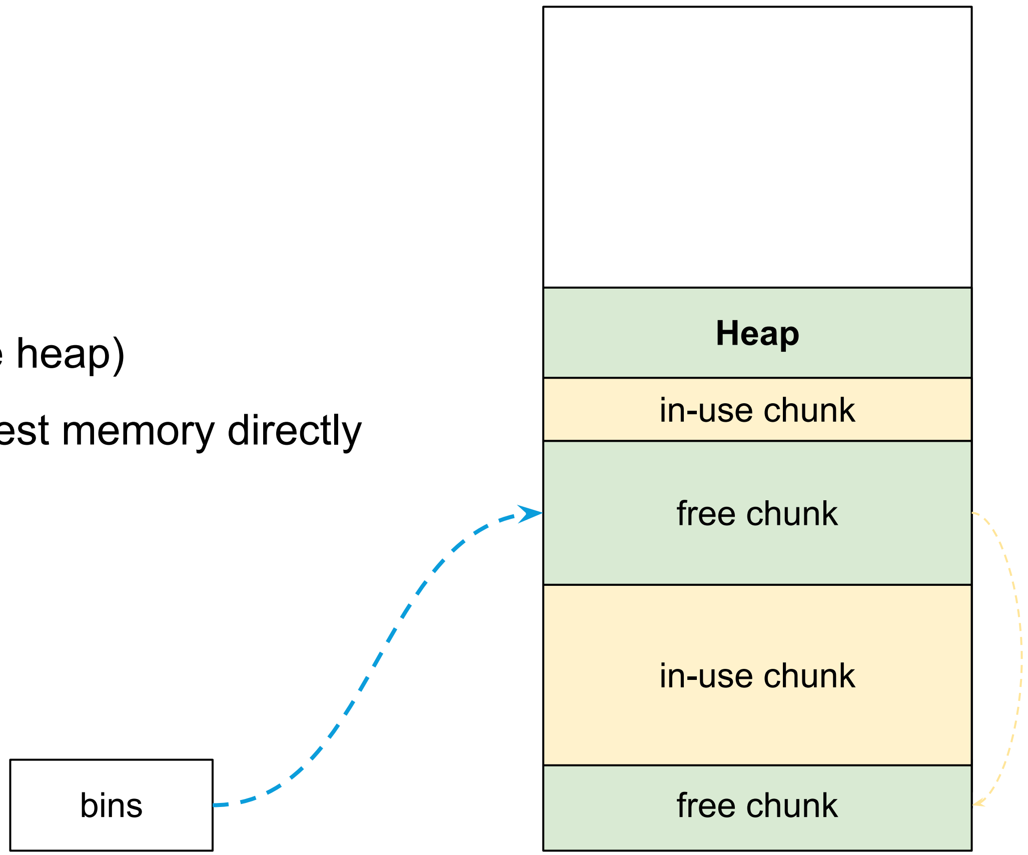
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it, use that
- If no chunk is available, create a new chunk(**`sbrk()`**: extends the heap)
- If the request is large enough (**`M_MMAP_THRESHOLD`**): **`mmap`** request memory directly from OS

Free algorithm:

- Place the free chunk in the appropriate bin
- If this chunk is adjacent to another free chunk, combine
- **If this chunk is mapped: `munmap`**



How does my code impact memory usage?

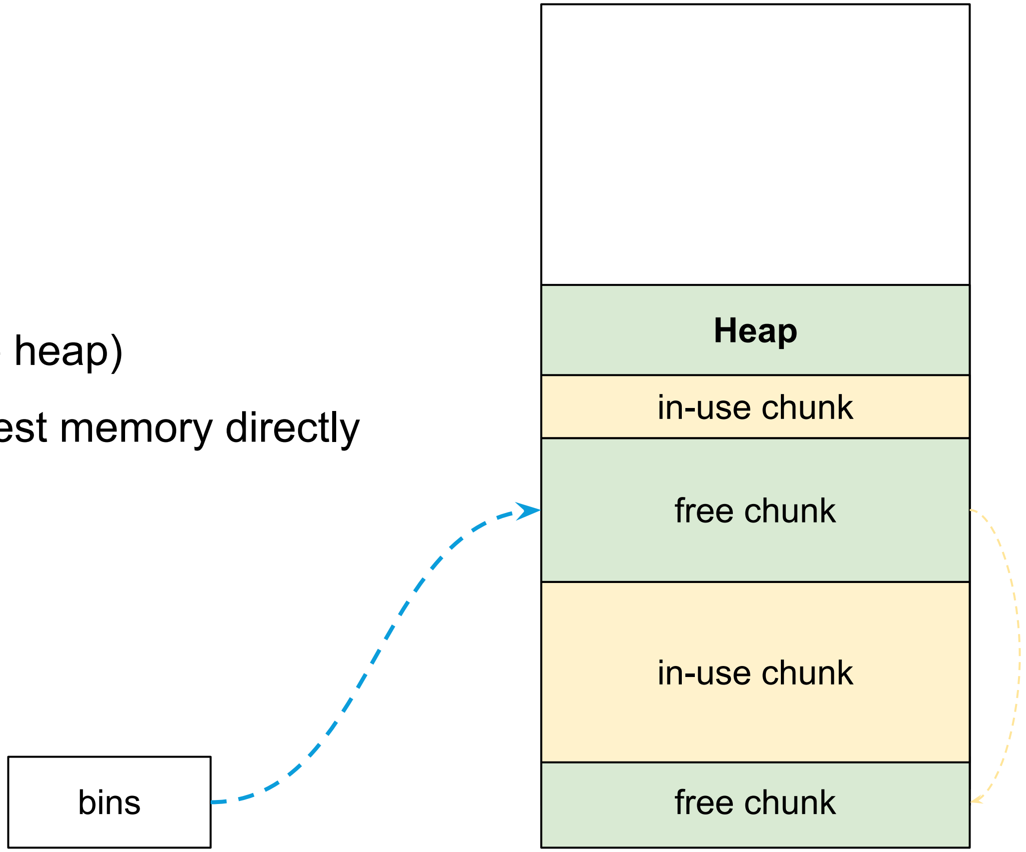
malloc

malloc algorithm:

- If the appropriate bin has a chunk in it, use that
- If no chunk is available, create a new chunk(**`sbrk()`**: extends the heap)
- If the request is large enough (**`M_MMAP_THRESHOLD`**): **`mmap`** request memory directly from OS

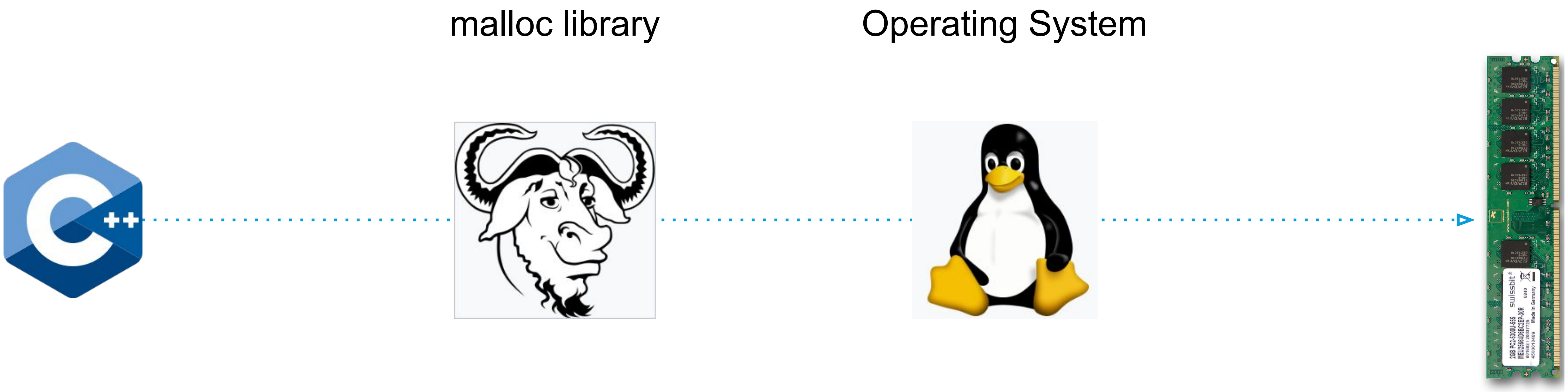
Free algorithm:

- Place the free chunk in the appropriate bin
- If this chunk is adjacent to another free chunk, combine
- If this chunk is mapped: **`munmap`**



(tcache, topmost chunk, shrink…)

How does my code impact memory usage?



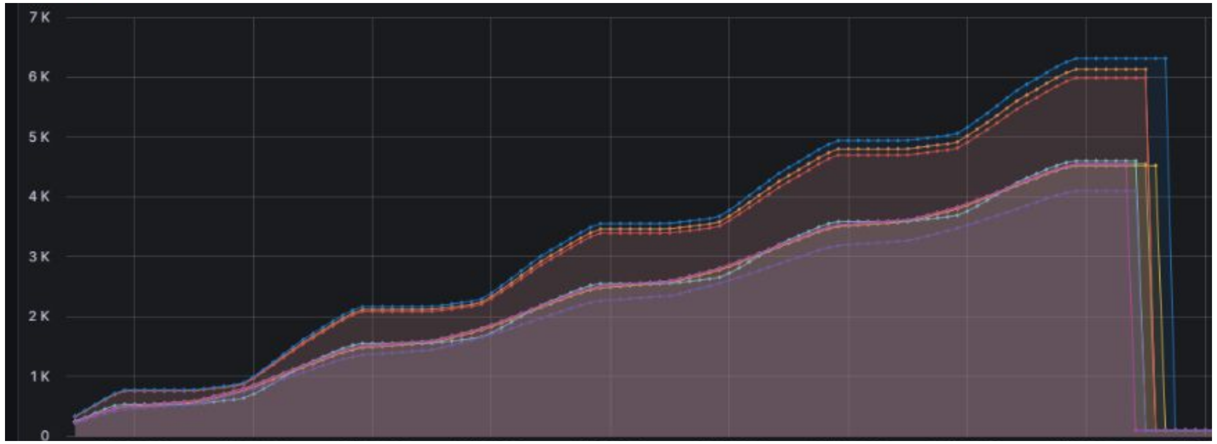
malloc library

Operating System

👷 **Memory allocation troubleshooting & tools:**
Leak & Fragmentation

Memory Leak

Memory which is no longer needed is not released



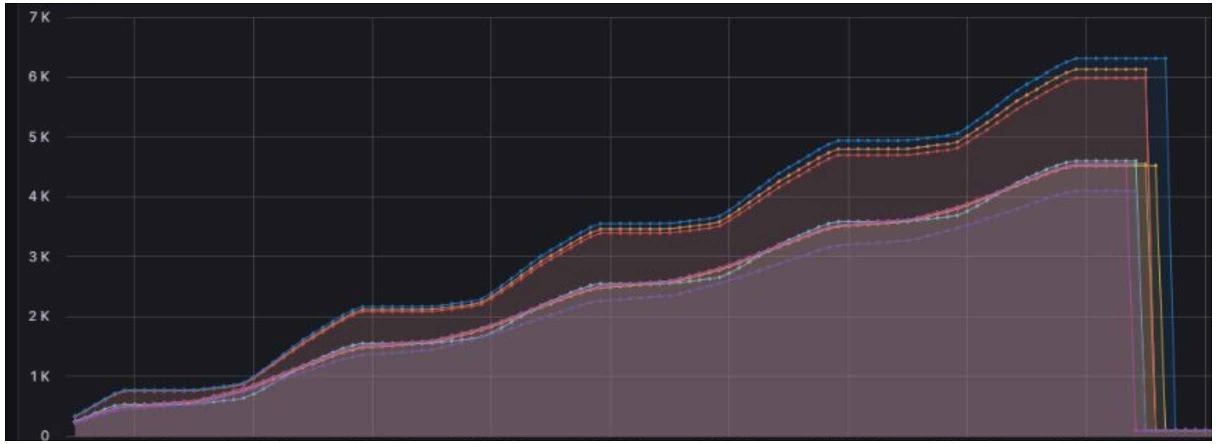
Memory Leak

Memory which is no longer needed is not released

1. **`new()`** it, then forget it

Resource acquisition is initialization (RAII)

“no object leaks, no resource leaks”



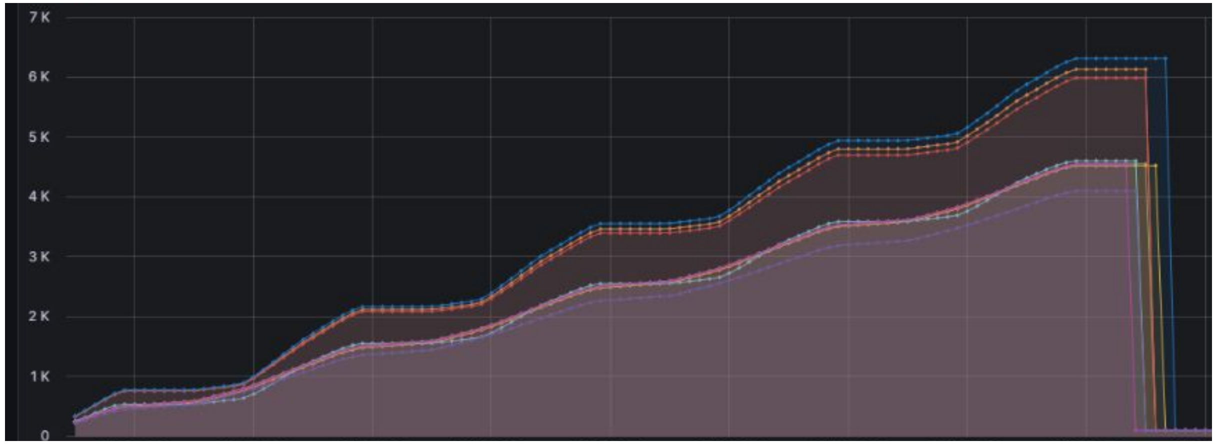
Memory Leak

Memory which is no longer needed is not released

1. **`new()`** it, then forget it
2. Keep entries that are no longer needed

keep pushing entries into a container

but never clean it

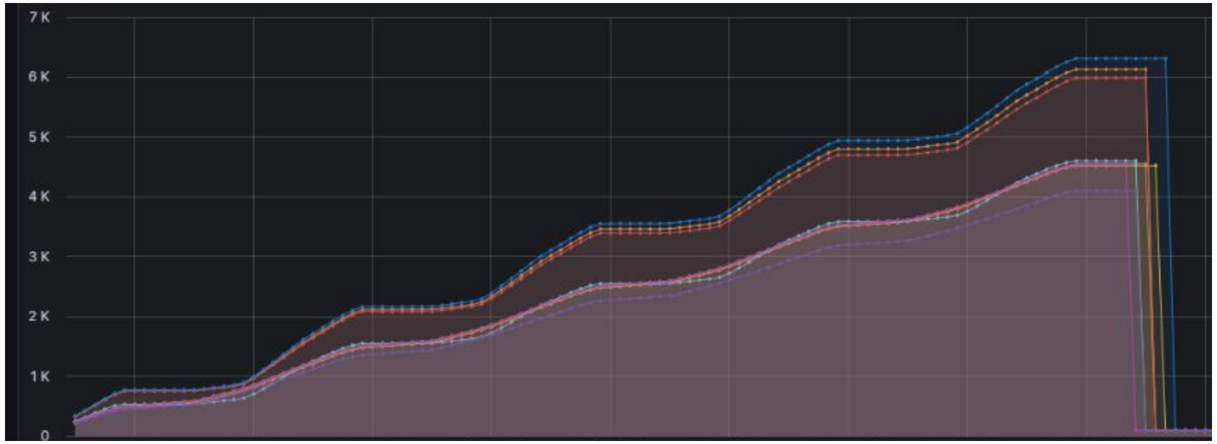


Memory Leak

Memory which is no longer needed is not released

1. **`new()`** it, then forget it
2. Keep entries that are no longer needed
3. Missing virtual ~Base()

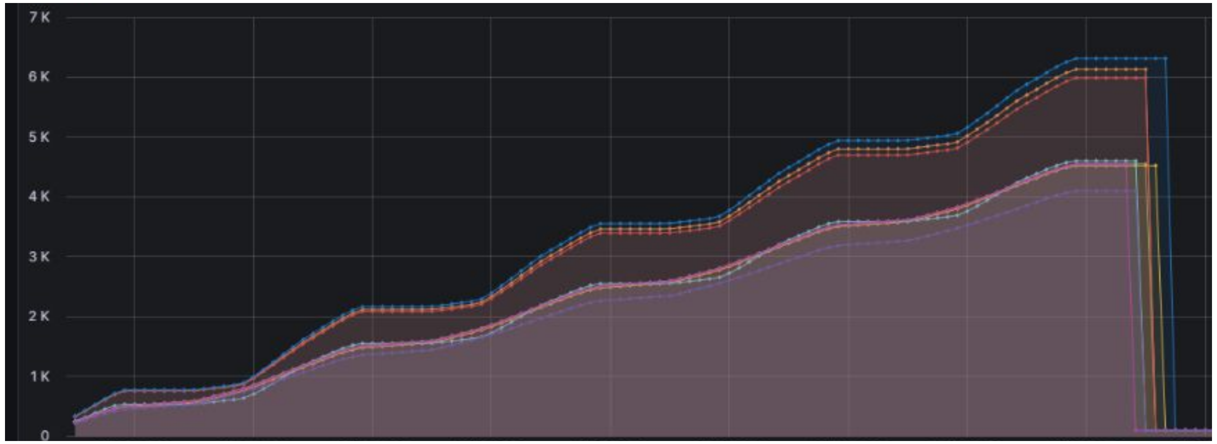
Base class' dtor isn't called

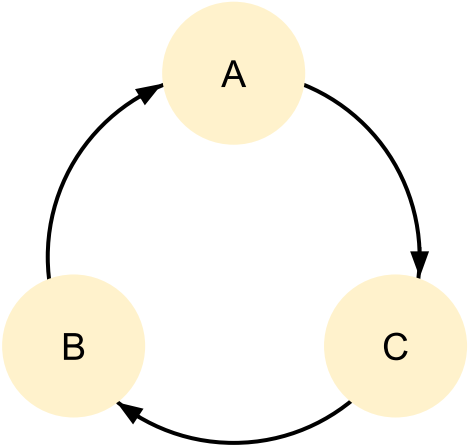


Memory Leak

Memory which is no longer needed is not released

1. **`new()`** it, then forget it
2. Keep entries that are no longer needed
3. Missing virtual ~Base()
4. Circular reference





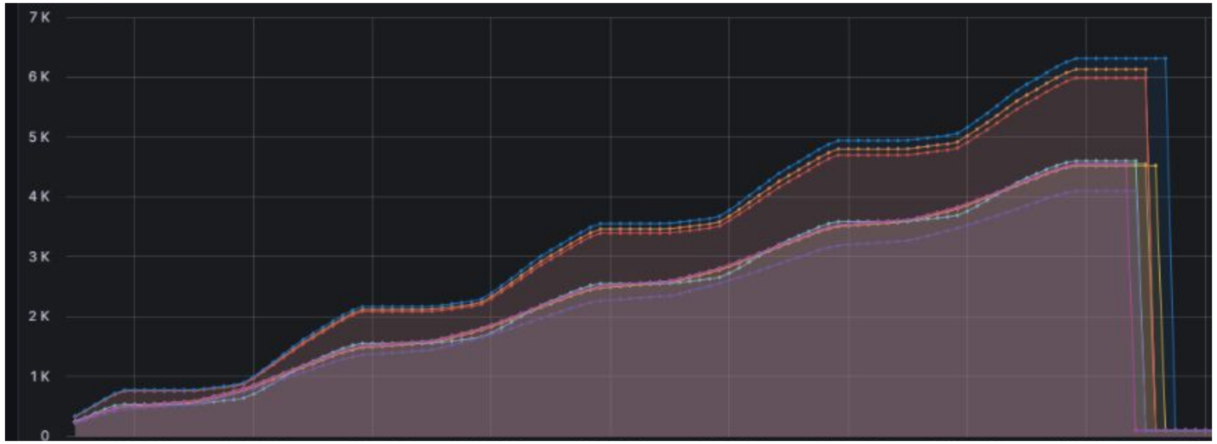
Reference counting: maintain a count of the smart pointers that point to the same object

Memory Leak

Memory which is no longer needed is not released

1. **`new()`** it, then forget it
2. Keep entries that are no longer needed
3. Missing virtual ~Base()
4. Circular reference

   …



🔍How to detect memory leaks?

Memory Leaks: Tools

Tool	What	How
bslma::TestAllocator	Allocator for detecting memory error	Inject the allocator, compile & link

Allocators: handle all the requests for allocation and deallocation of memory for a given container

“…library containers independent of the underlying memory model”
“...all of the STL container interfaces had to be rewritten to accept allocators”
C++98: stateless allocators
C++03: stateful allocators
C++17: PMR allocators: flexibility at run time

Memory Leaks: Tools

Tool	What	How
bslma::TestAllocator	Allocator for detecting memory error	Inject the allocator, compile & link

Allocators: handle all the requests for allocation and deallocation of memory for a given container

```
template<
    class T,
    class Allocator = std::allocator<T>
> class vector;
```

“…library containers independent of the underlying memory model”
“...all of the STL container interfaces had to be rewritten to accept allocators”
C++98: stateless allocators
C++03: stateful allocators
C++17: PMR allocators: flexibility at run time

Memory Leaks: Tools

Tool	What	How
bslma::TestAllocator	Allocator for detecting memory error	Inject the allocator, compile & link

Allocators: handle all the requests for allocation and deallocation of memory for a given container

```
template<
    class T,
    class Allocator = std::allocator<T>
> class vector;
```

“…library containers independent of the underlying memory model”
“...all of the STL container interfaces had to be rewritten to accept allocators”
C++98: stateless allocators
C++03: stateful allocators
C++17: PMR allocators: flexibility at run time

😃

- fast
- small overhead
- scoped

🙁

- code change required
- compile & link required

Memory Leaks: Tools

Tool	What	How
AddressSanitizer	Memory error detector	compile & link **`-fsanitize=address`**

google/sanitizers

a compiler instrumentation module

a runtime library which replaces the **`malloc`** function

Memory Leaks: Tools

Tool	What	How
AddressSanitizer	Memory error detector	compile & link **`-fsanitize=address`**

google/sanitizers

a compiler instrumentation module

a runtime library which replaces the **`malloc`** function

😃

- no code change required
- fast (50%-100% slower)

🙁

- compile & link required
- extra memory cost

Memory Leaks: Tools

Tool	What	How
valgrind memcheck	memory error detector	valgrind --tool=memcheck <prog>
valgrind massif	heap profiler	valgrind --tool=massif <prog>

Valgrind

runs your application in a "sandbox"
insert its own instructions to do advanced debugging and profiling

Memory Leaks: Tools

Tool	What	How
valgrind memcheck	memory error detector	valgrind --tool=memcheck <prog>
valgrind massif	heap profiler	valgrind --tool=massif <prog>

Valgrind

runs your application in a "sandbox"
insert its own instructions to do advanced debugging and profiling

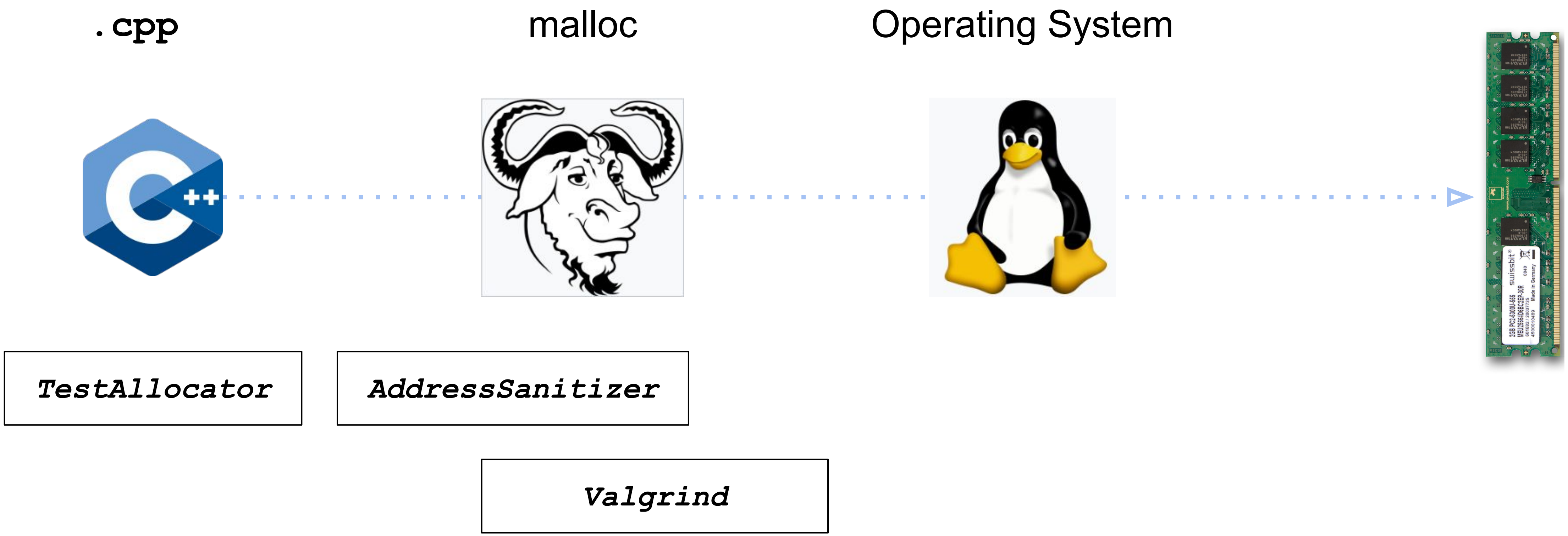
😃

- no code change required
- no compile & link required

🙁

- slow (10-30 times slower)
- extra memory cost

Memory Leaks: Tools



`.cpp`

malloc

Operating System

`TestAllocator`

`AddressSanitizer`

`Valgrind`

Memory Leaks: Tools

snapshot in the end *VS.* *snapshots over time*

TestAllocator / AddressSanitizer / memcheck

massif

Memory Leaks: Tools

snapshot in the end *VS.* *snapshots over time*

TestAllocator / AddressSanitizer / memcheck

massif

```
 LEAK SUMMARY:
    definitely lost: 921,664 bytes in 730 blocks
    indirectly lost: 59,250,966 bytes in 53,741 blocks
      possibly lost: 23,493,535 bytes in 19,707 blocks
    still reachable: 85,768,287 bytes in 88,669 blocks

<call stacks>
```

Memory Leaks: Tools

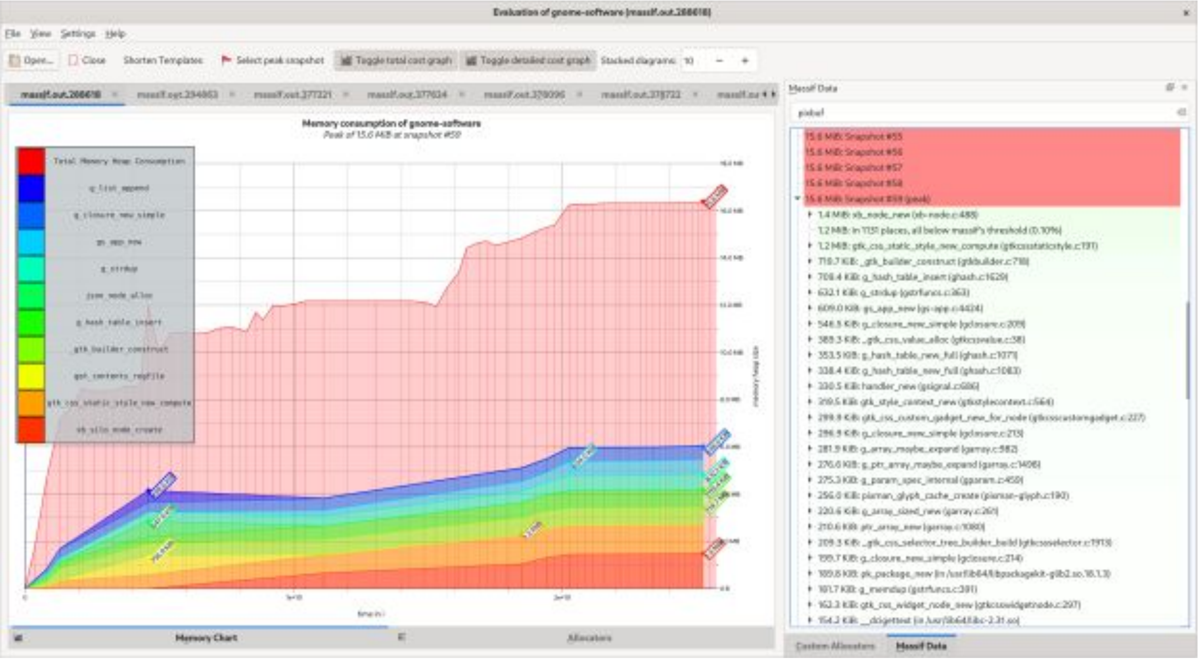
snapshot in the end *VS.* *snapshots over time*

TestAllocator / AddressSanitizer / memcheck

massif

```
 LEAK SUMMARY:
    definitely lost: 921,664 bytes in 730 blocks
    indirectly lost: 59,250,966 bytes in 53,741 blocks
      possibly lost: 23,493,535 bytes in 19,707 blocks
    still reachable: 85,768,287 bytes in 88,669 blocks

<call stacks>
```


Evaluation of gnome-software (massif.out.288618)
File View Settings Help
Open... Close Shorten Templates Select peak snapshot Toggle total cost graph Toggle detailed cost graph Stacked diagrams: 10 − +
massif.out.288618 massif.out.294863 massif.out.377221 massif.out.377634 massif.out.378096 massif.out.378722 massif.ou
Memory consumption of gnome-software
Peak of 15.6 MiB at snapshot #59
Total Memory Heap Consumption
g_list_append
g_closure_new_simple
gs_app_new
g_strdup
json_node_alloc
g_hash_table_insert
gtk_builder_construct
gph_contents_regfile
gtk_css_static_style_new_compute
xb_silo_node_create
time in i
memory heap size
Memory Chart
Allocators
Massif Data
pixbuf
Custom Allocators
Massif Data

```
15.6 MiB: Snapshot #55
15.6 MiB: Snapshot #56
15.6 MiB: Snapshot #57
15.6 MiB: Snapshot #58
15.6 MiB: Snapshot #59 (peak)
  1.4 MiB: xb_node_new (xb-node.c:488)
  1.2 MiB: in 1131 places, all below massif's threshold (0.10%)
  1.2 MiB: gtk_css_static_style_new_compute (gtkcssstaticstyle.c:131)
  718.7 KiB: _gtk_builder_construct (gtkbuilder.c:718)
  708.4 KiB: g_hash_table_insert (ghash.c:1629)
  632.1 KiB: g_strdup (gstrfuncs.c:363)
  609.0 KiB: gs_app_new (gs-app.c:4424)
  546.5 KiB: g_closure_new_simple (gclosure.c:209)
  388.3 KiB: _gtk_css_value_alloc (gtkcssvalue.c:38)
  353.5 KiB: g_hash_table_new_full (ghash.c:1077)
  338.4 KiB: g_hash_table_new_full (ghash.c:1083)
  330.5 KiB: handler_new (gsignal.c:686)
  319.5 KiB: gtk_style_context_new (gtkstylecontext.c:564)
  299.8 KiB: gtk_css_custom_gadget_new_for_node (gtkcsscustomgadget.c:227)
  296.9 KiB: g_closure_new_simple (gclosure.c:213)
  281.9 KiB: g_array_maybe_expand (garray.c:582)
  276.6 KiB: g_ptr_array_maybe_expand (garray.c:1406)
  275.3 KiB: g_param_spec_internal (gparam.c:450)
  256.0 KiB: pixman_glyph_cache_create (pixman-glyph.c:190)
  220.6 KiB: g_array_sized_new (garray.c:261)
  210.6 KiB: ptr_array_new (garray.c:1080)
  209.3 KiB: _gtk_css_selector_tree_builder_build (gtkcssselector.c:1913)
  199.7 KiB: g_closure_new_simple (gclosure.c:214)
  189.8 KiB: pk_package_new (in /usr/lib64/libpackagekit-glib2.so.18.1.3)
  181.7 KiB: g_memdup (gstrfuncs.c:381)
  162.3 KiB: gtk_css_widget_node_new (gtkcsswidgetnode.c:297)
  154.2 KiB: __dcigettext (in /usr/lib64/libc-2.31.so)
```

Sourced from https://github.com/KDE/massif-visualizer.

Memory Leaks

Tips:

- “Static” leaks may hide the real issue – we need enough traffic for profiling.
- They are all good tools, but for different cases.
- Catch the problem in earlier stages – Integrate AddressSanitizer in CI.
- Install the tools so we can start profiling easily.
- Care about the lifecycle and ownership of what we allocate.

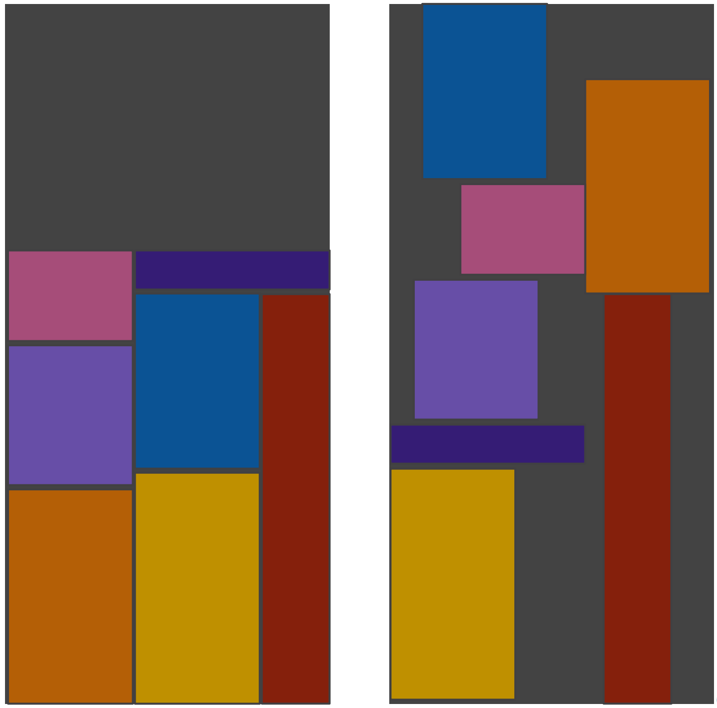
Memory Leaks

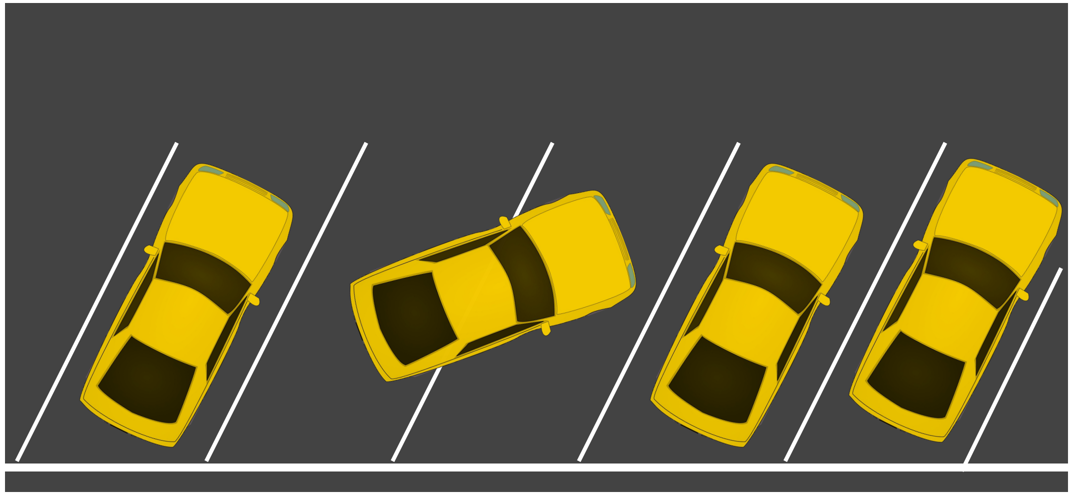
Memory leak fixed.

😕 What else?

Fragmentation

You try to allocate a big block and you can't, even though you appear to have enough memory free

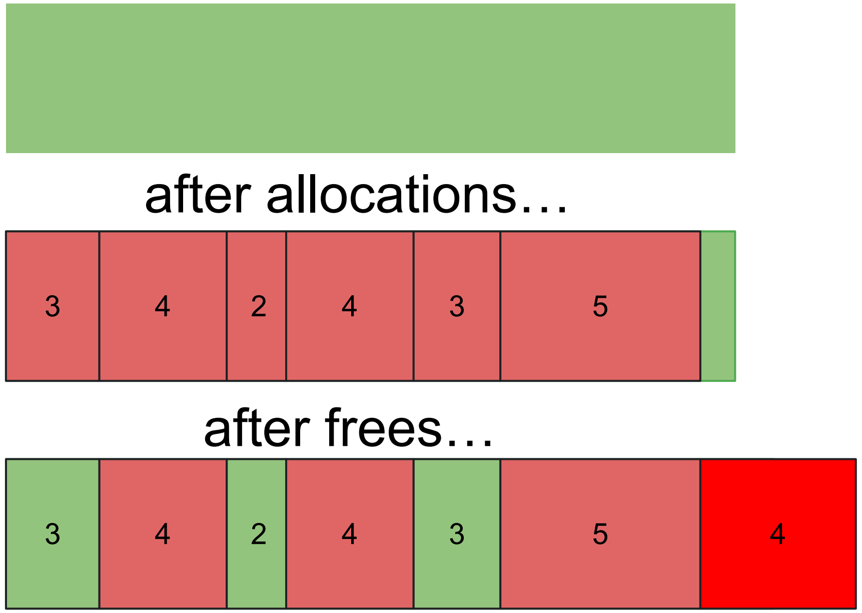




Source: Pixabay

Fragmentation

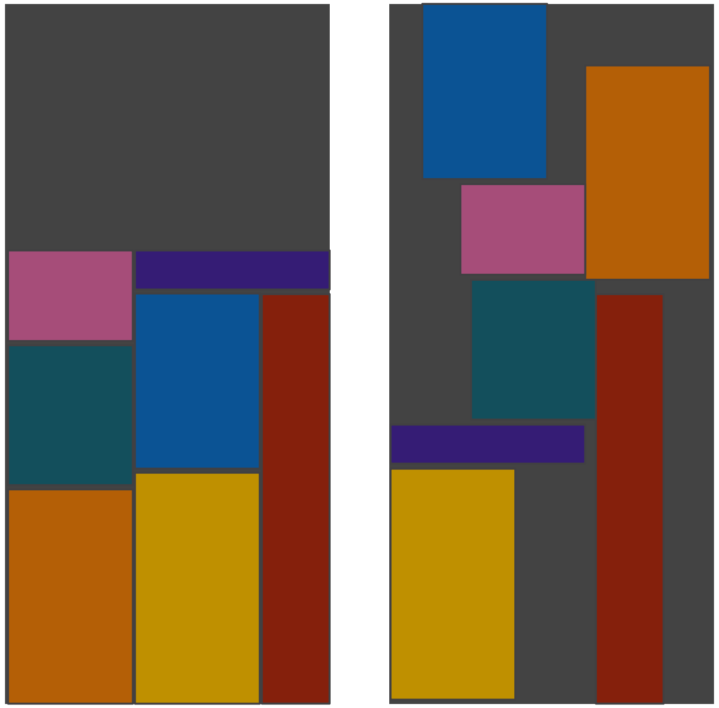
External fragmentation



after allocations…

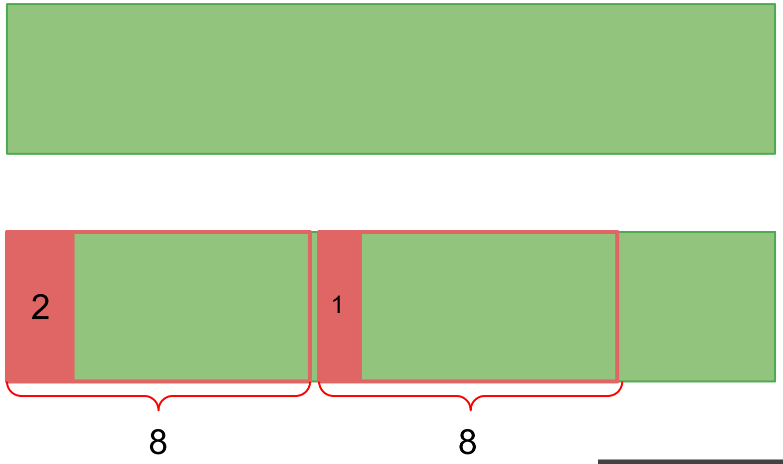
after frees…

I have free spaces, but I need to extend the heap

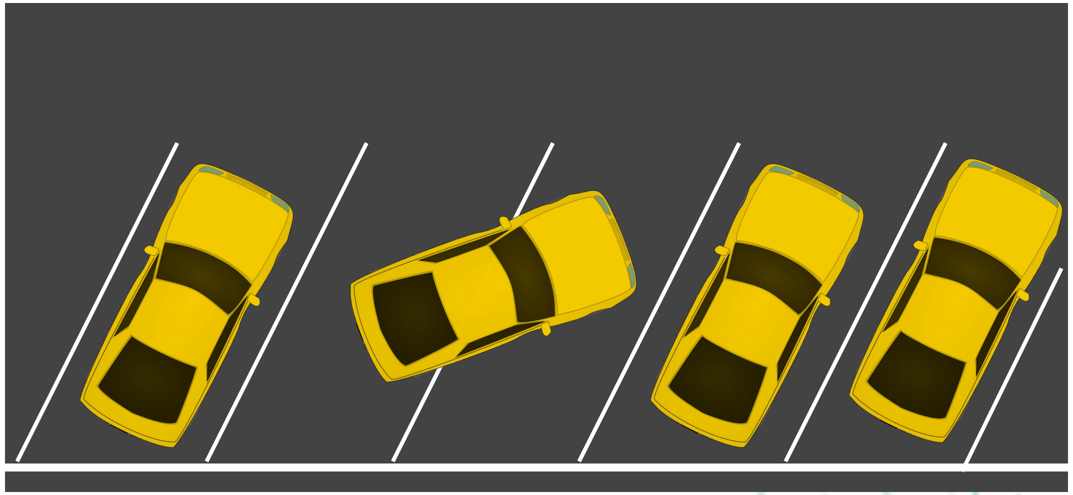


Fragmentation

Internal fragmentation



I allocate a large chunk for a small size



🔍How to estimate Fragmentation?

How to estimate Fragmentation?

External:

External Fragmentation = 1 - LargestAllocableBlock / TotalFreeMemory

Most allocations can be done with chunks in bins.

How to estimate Fragmentation?

External:

External Fragmentation = 1 - LargestAllocableBlock / TotalFreeMemory

Most allocations can be done with chunks in bins.

Internal:

Internal Fragmentation = 1 - AccessedBytes / TotalAllocatedBytes

Avoid useless or underused allocations.

How to estimate Fragmentation?

External Fragmentation = 1 - LargestAllocableBlock / TotalFreeMemory

mallinfo(3)

mallinfo service A:

Total bytes (arena):	244 355 072
Total in-use allocations:	**31 650 352**
Total free space:	**212 704 720**
Largest allocable block:	**63 248**

External Fragmentation = 0.9997

How to estimate Fragmentation?

External Fragmentation = 1 - LargestAllocableBlock / TotalFreeMemory

mallinfo(3)

mallinfo service A:

Total bytes (arena):	244 355 072
Total in-use allocations:	**31 650 352**
Total free space:	**212 704 720**
Largest allocable block:	**63 248**

External Fragmentation = 0.9997

mallinfo service B:

Total bytes	(arena):	33 292 288
Total in-use allocations:		**30 316 416**
Total free space:		**2 975 872**
Largest allocable block:		**120 928**

External Fragmentation = 0.9593

How to estimate Fragmentation?

External Fragmentation = 1 - LargestAllocableBlock / TotalFreeMemory

mallinfo(3)

mallinfo service A:

Total bytes (arena):	244 355 072
Total in-use allocations:	**31 650 352**
Total free space:	**212 704 720**
Largest allocable block:	**63 248**

External Fragmentation = 0.9997

1 day - 400MB

mallinfo service B:

Total bytes	(arena):	33 292 288
Total in-use allocations:		**30 316 416**
Total free space:		**2 975 872**
Largest allocable block:		**120 928**

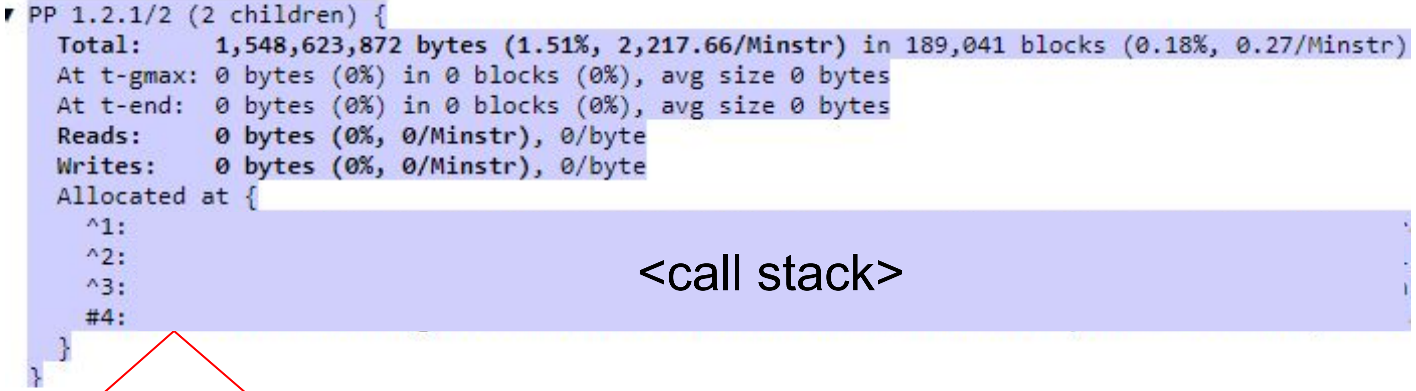
External Fragmentation = 0.9593

7 day - 200MB

How to estimate Fragmentation?

Internal Fragmentation = 1 - AccessedBytes / TotalAllocatedBytes

Valgrind --tool=dhat



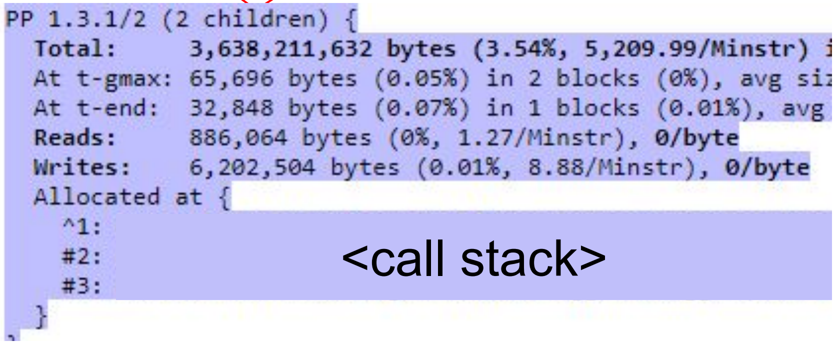
```
PP 1.2.1/2 (2 children) {
  Total:     1,548,623,872 bytes (1.51%, 2,217.66/Minstr) in 189,041 blocks (0.18%, 0.27/Minstr)
  At t-gmax: 0 bytes (0%) in 0 blocks (0%), avg size 0 bytes
  At t-end:  0 bytes (0%) in 0 blocks (0%), avg size 0 bytes
  Reads:     0 bytes (0%, 0/Minstr), 0/byte
  Writes:    0 bytes (0%, 0/Minstr), 0/byte
  Allocated at {
    ^1:
    ^2:                   <call stack>
    ^3:
    #4:
  }
}
```

Zero-access

Total:	1, 548, 632, 872	bytes
Reads:	0	bytes
Writes:	0	bytes

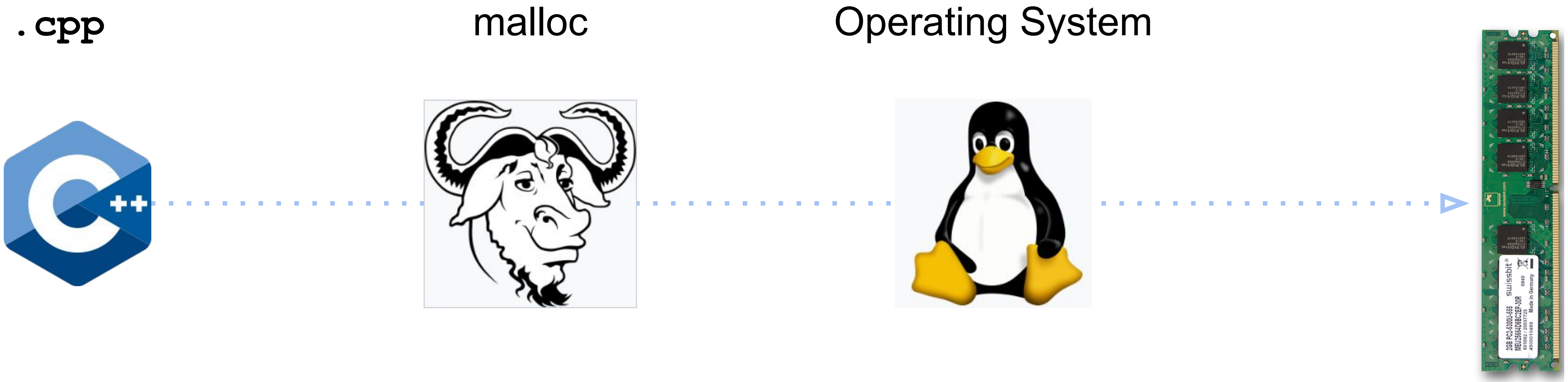
Low-access

Total:	3, 638, 211, 632	bytes
Reads:	886, 064	bytes
Writes:	86, 202, 504	bytes



```
PP 1.3.1/2 (2 children) {
  Total:     3,638,211,632 bytes (3.54%, 5,209.99/Minstr) i
  At t-gmax: 65,696 bytes (0.05%) in 2 blocks (0%), avg si
  At t-end:  32,848 bytes (0.07%) in 1 blocks (0.01%), avg
  Reads:     886,064 bytes (0%, 1.27/Minstr), 0/byte
  Writes:    6,202,504 bytes (0.01%, 8.88/Minstr), 0/byte
  Allocated at {
    ^1:
    #2:              <call stack>
    #3:
  }
}
```

Defragmentation

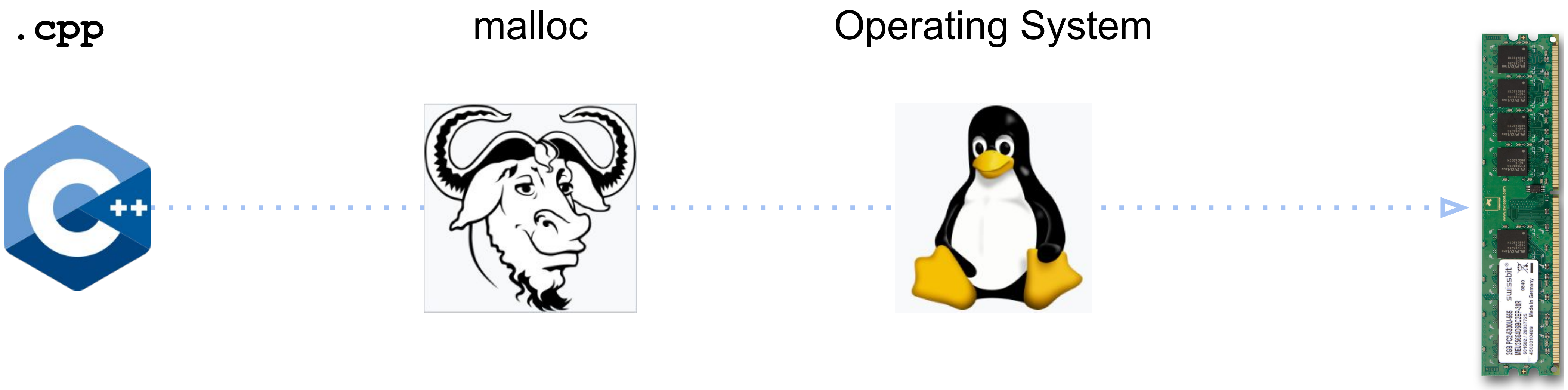


`.cpp`

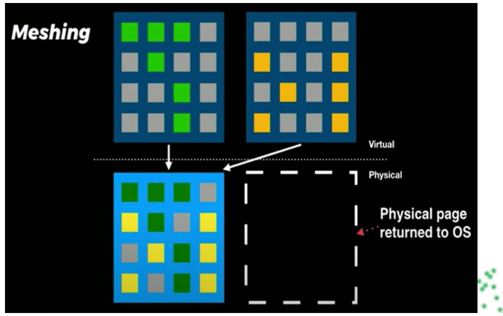
malloc

Operating System

Defragmentation

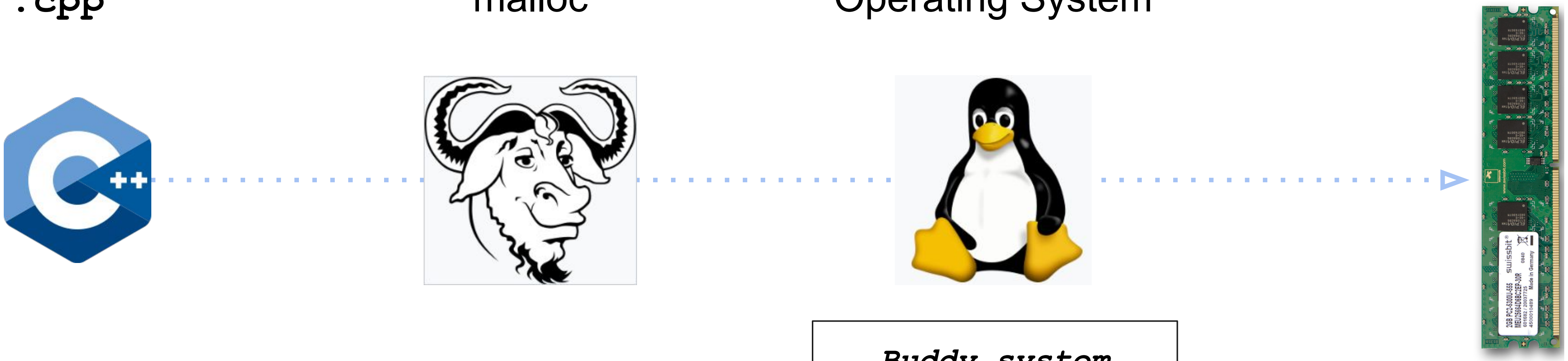


MESH



CppCon 2019: Emery Berger “Mesh: Automatically Compacting”

Defragmentation

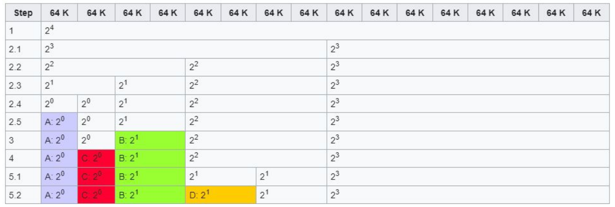


`.cpp`

malloc

Operating System

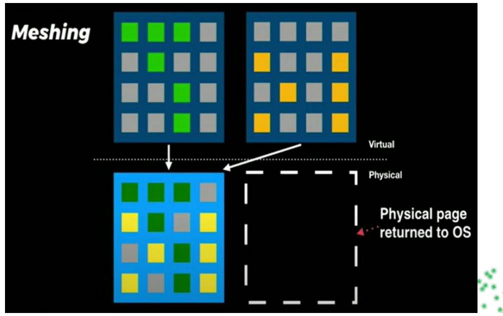
`Buddy system`



Step	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K
1	2^4															
2.1	2^3								2^3							
2.2	2^2				2^2				2^3							
2.3	2^1		2^1		2^2				2^3							
2.4	2^0	2^0	2^1		2^2				2^3							
2.5	A: 2^0	2^0	2^1		2^2				2^3							
3	A: 2^0	2^0	B: 2^1		2^2				2^3							
4	A: 2^0	C: 2^0	B: 2^1		2^2				2^3							
5.1	A: 2^0	C: 2^0	B: 2^1		2^1		2^1		2^3							
5.2	A: 2^0	C: 2^0	B: 2^1		D: 2^1		2^1		2^3							

Source: Wikipedia licensed under CC BY-SA 4.0

`MESH`



CppCon 2019: Emery Berger "Mesh: Automatically Compacting"

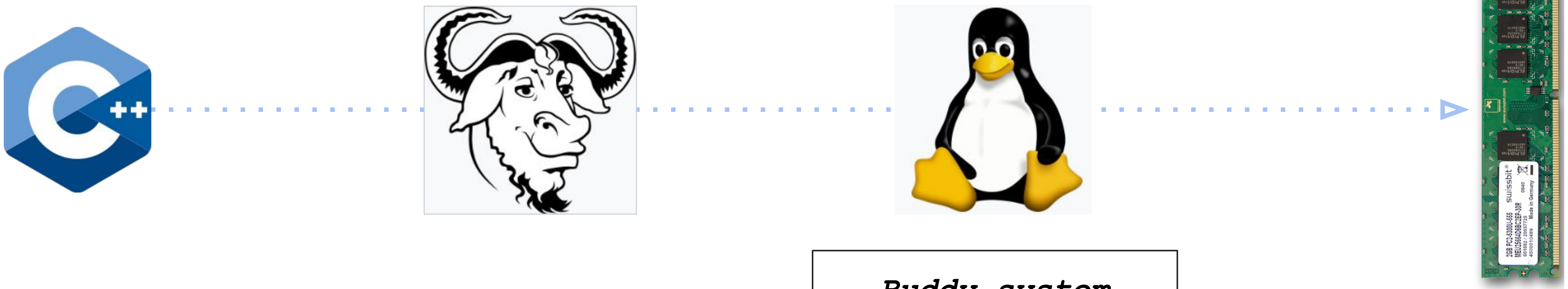
Defragmentation

Malloc Tunable Parameters

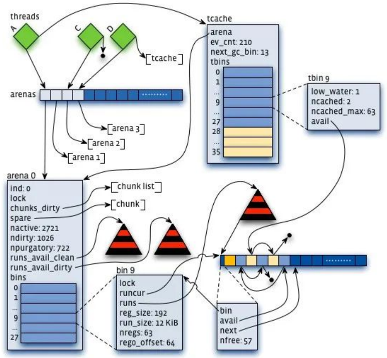
.cpp

malloc

Operating System

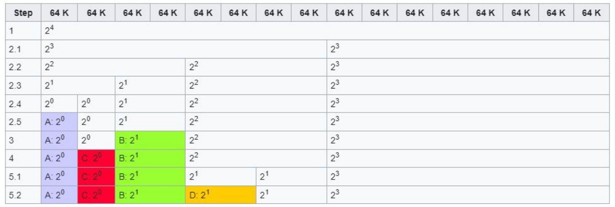


jemalloc



Source: scalable-memory-allocation-using-jemalloc

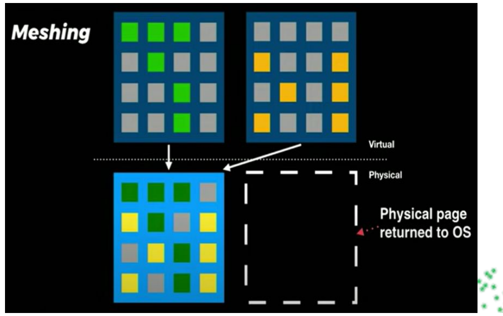
Buddy system



Step	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K
1	2^4															
2.1	2^3								2^3							
2.2	2^2				2^2				2^3							
2.3	2^1		2^1		2^2				2^3							
2.4	2^0	2^0	2^1		2^2				2^3							
2.5	A: 2^0	2^0	2^1		2^2				2^3							
3	A: 2^0	2^0	B: 2^1		2^2				2^3							
4	A: 2^0	C: 2^0	B: 2^1		2^2				2^3							
5.1	A: 2^0	C: 2^0	B: 2^1		2^1		2^1		2^3							
5.2	A: 2^0	C: 2^0	B: 2^1		D: 2^1		2^1		2^3							

Source: Wikipedia licensed under CC BY-SA 4.0

MESH



CppCon 2019: Emery Berger "Mesh: Automatically Compacting"

Defragmentation

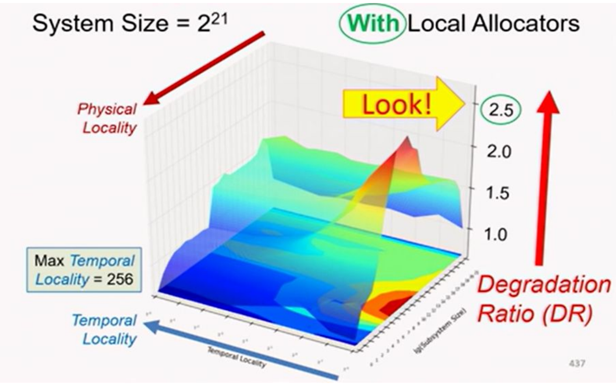
Malloc Tunable Parameters

.cpp

malloc

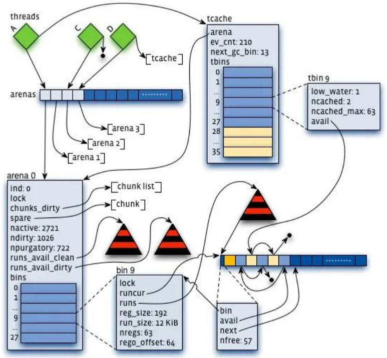
Operating System

local allocators



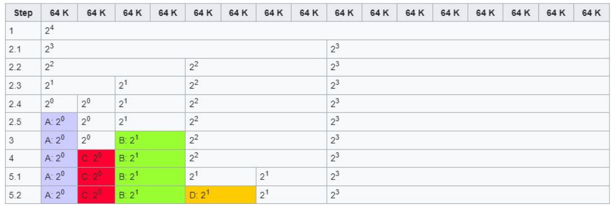
John Lakos "Local(Arena) memory allocators" - CppCon 2017

jemalloc



Source: scalable-memory-allocation-using-jemalloc

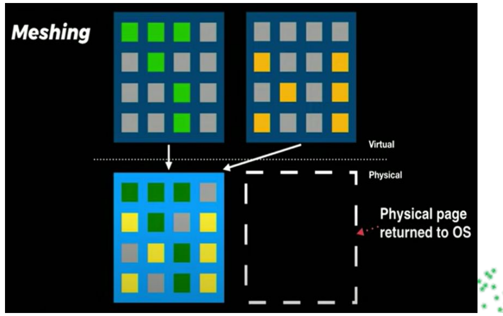
Buddy system



Step	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K	64 K
1	2^4															
2.1	2^3								2^3							
2.2	2^2				2^2				2^3							
2.3	2^1		2^1		2^2				2^3							
2.4	2^0	2^0	2^1		2^2				2^3							
2.5	A: 2^0	2^0	2^1		2^2				2^3							
3	A: 2^0	2^0	B: 2^1		2^2				2^3							
4	A: 2^0	C: 2^0	B: 2^1		2^2				2^3							
5.1	A: 2^0	C: 2^0	B: 2^1		2^1		2^1		2^3							
5.2	A: 2^0	C: 2^0	B: 2^1		D: 2^1		2^1		2^3							

Source: Wikipedia licensed under CC BY-SA 4.0

MESH



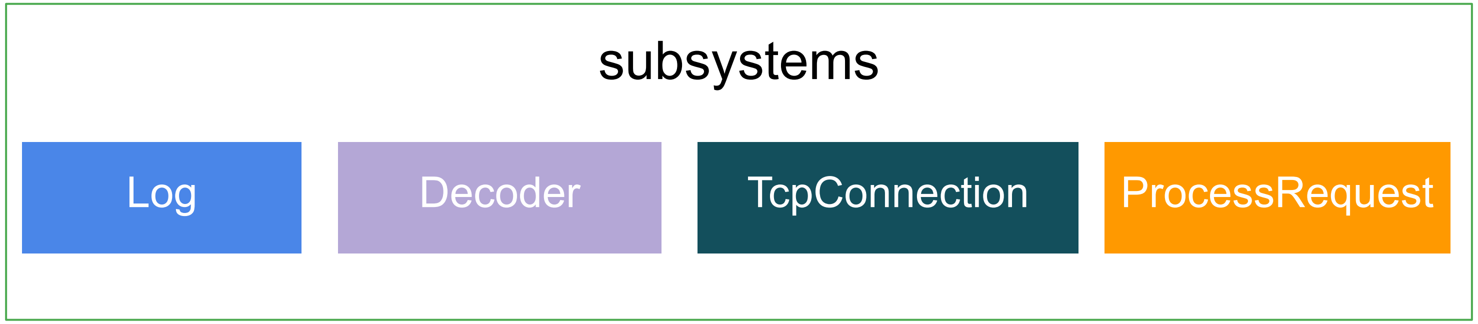
CppCon 2019: Emery Berger "Mesh: Automatically Compacting"

Defragmentation

Long-running stateless system

- start and run for a long time
- receive requests and process them, no state

Defragmentation



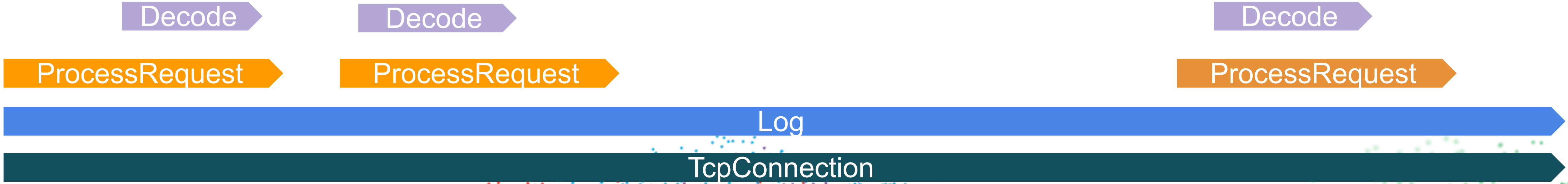
subsystems

Log

Decoder

TcpConnection

ProcessRequest



Decode

Decode

Decode

ProcessRequest

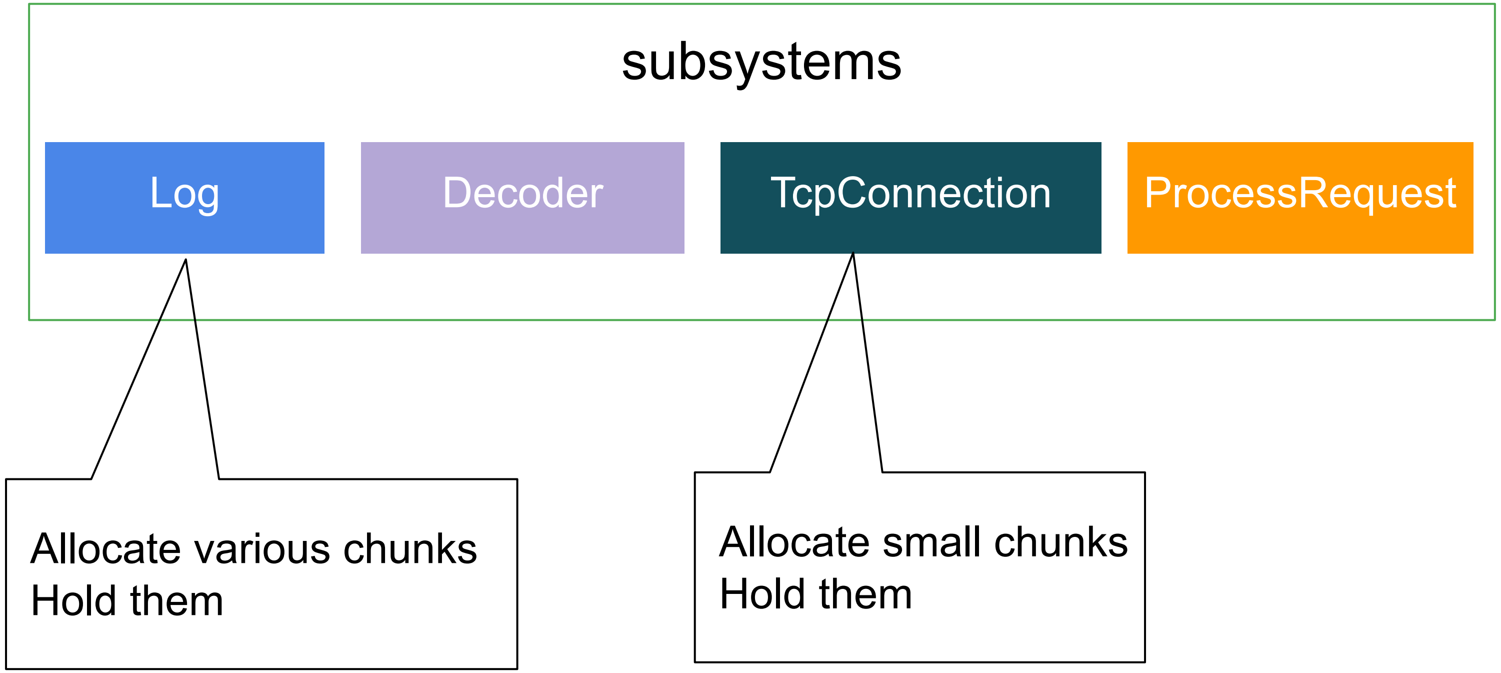
ProcessRequest

ProcessRequest

Log

TcpConnection

Defragmentation



subsystems

Log

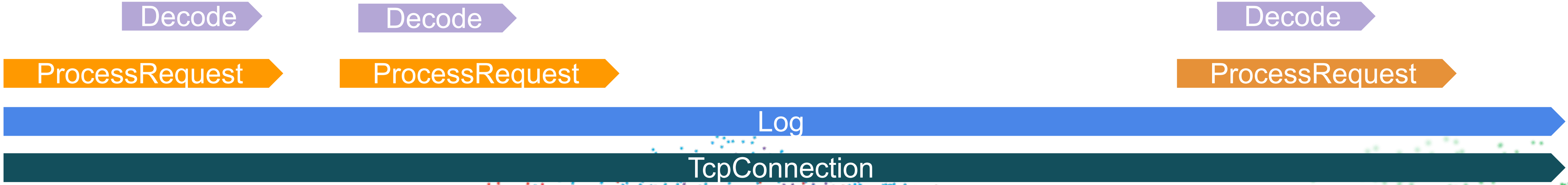
Decoder

TcpConnection

ProcessRequest

Allocate various chunks
Hold them

Allocate small chunks
Hold them



Decode

Decode

Decode

ProcessRequest

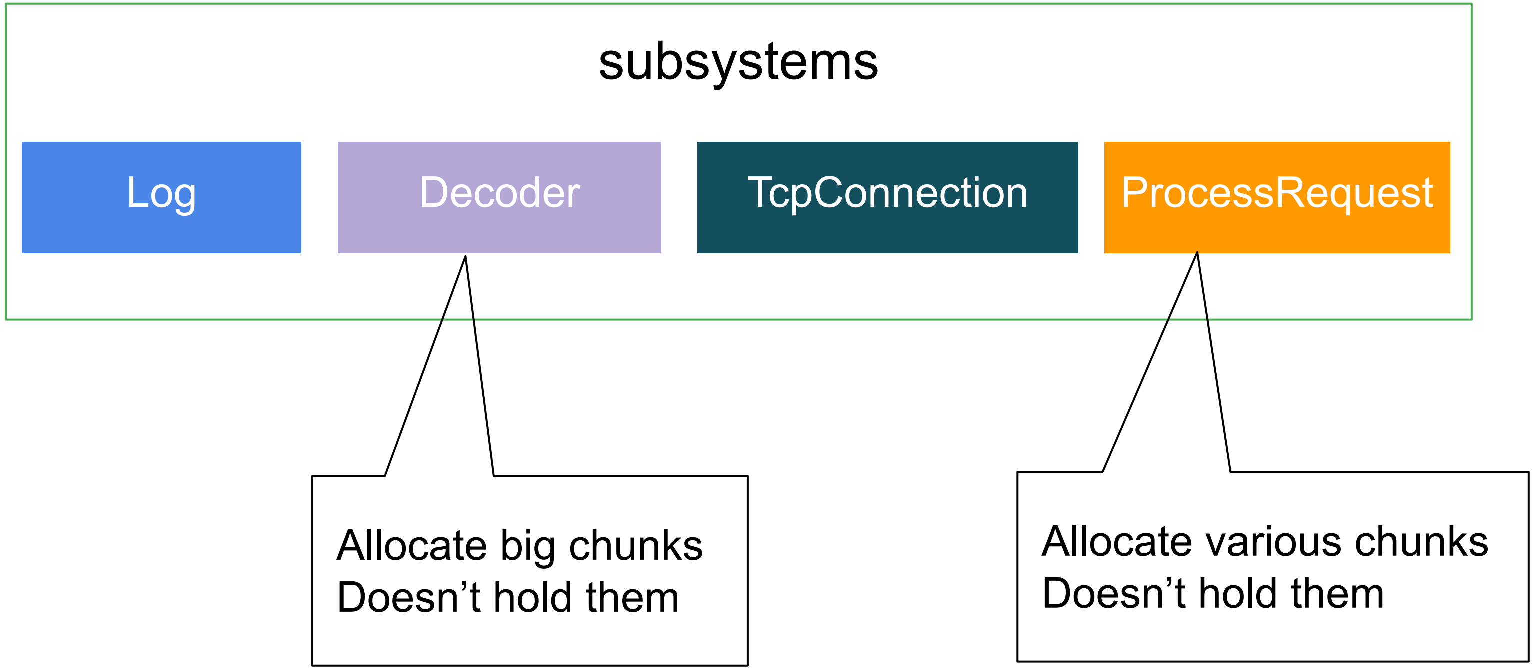
ProcessRequest

ProcessRequest

Log

TcpConnection

Defragmentation



subsystems

Log

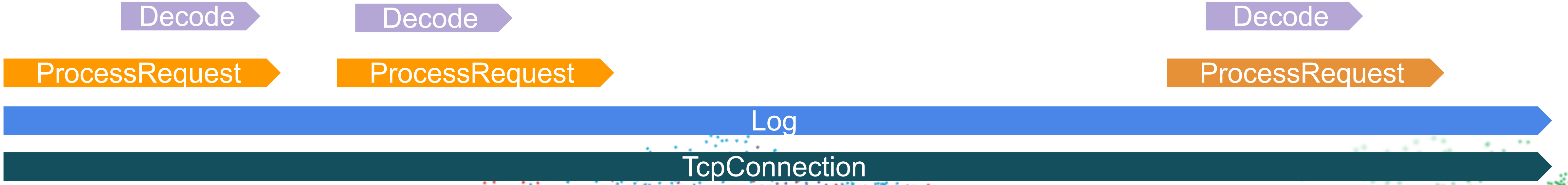
Decoder

TcpConnection

ProcessRequest

Allocate big chunks
Doesn’t hold them

Allocate various chunks
Doesn’t hold them



Decode

Decode

Decode

ProcessRequest

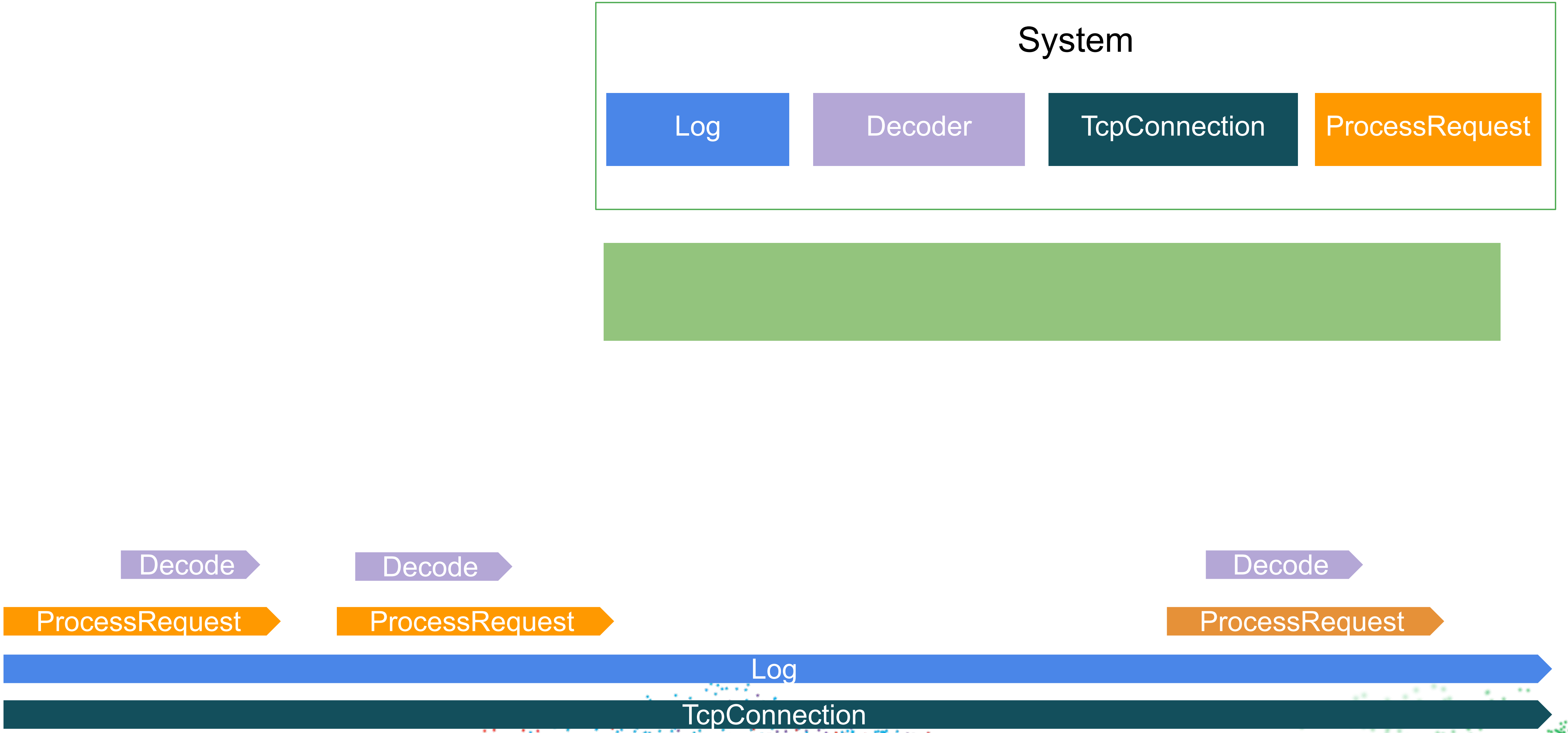
ProcessRequest

ProcessRequest

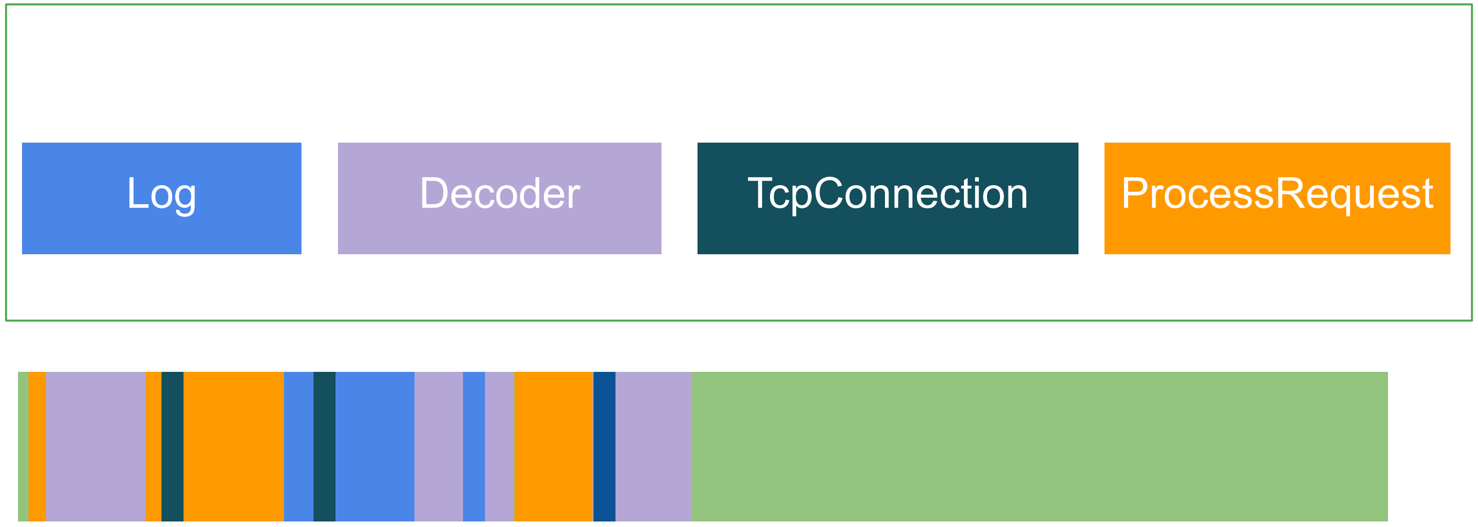
Log

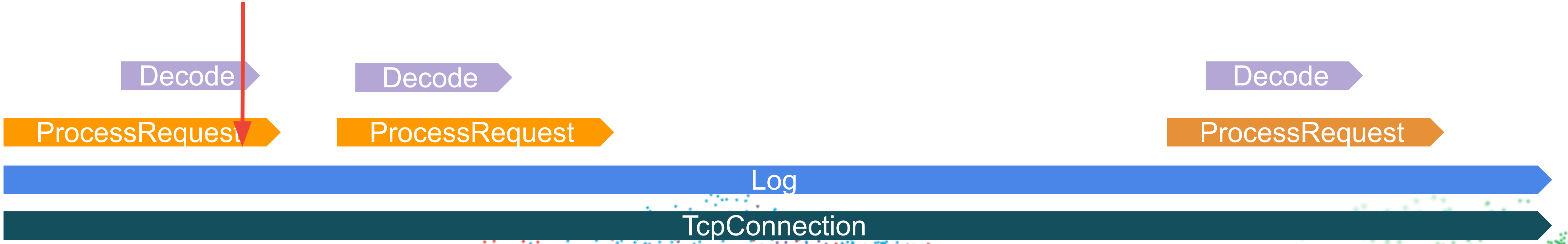
TcpConnection

Defragmentation

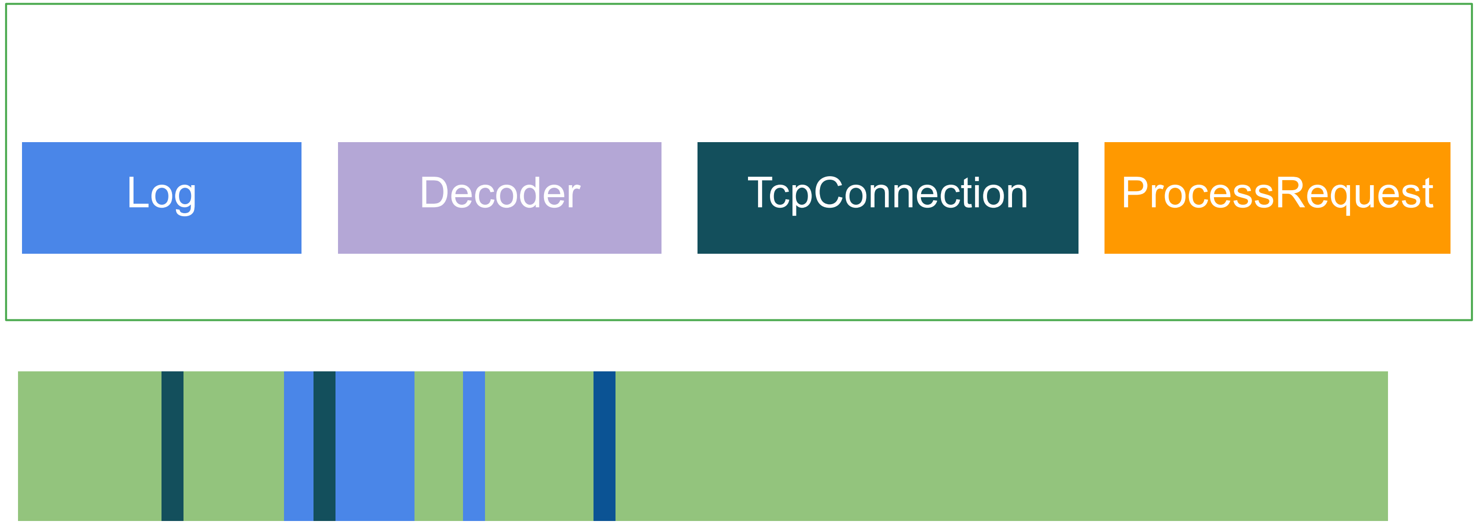


Defragmentation

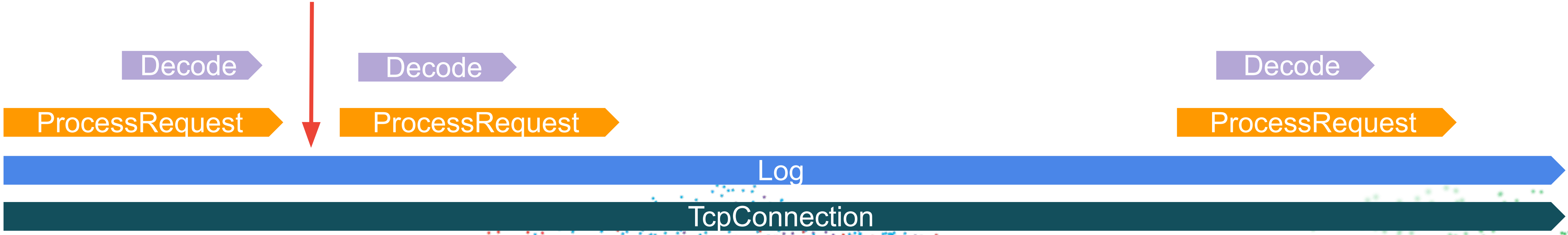




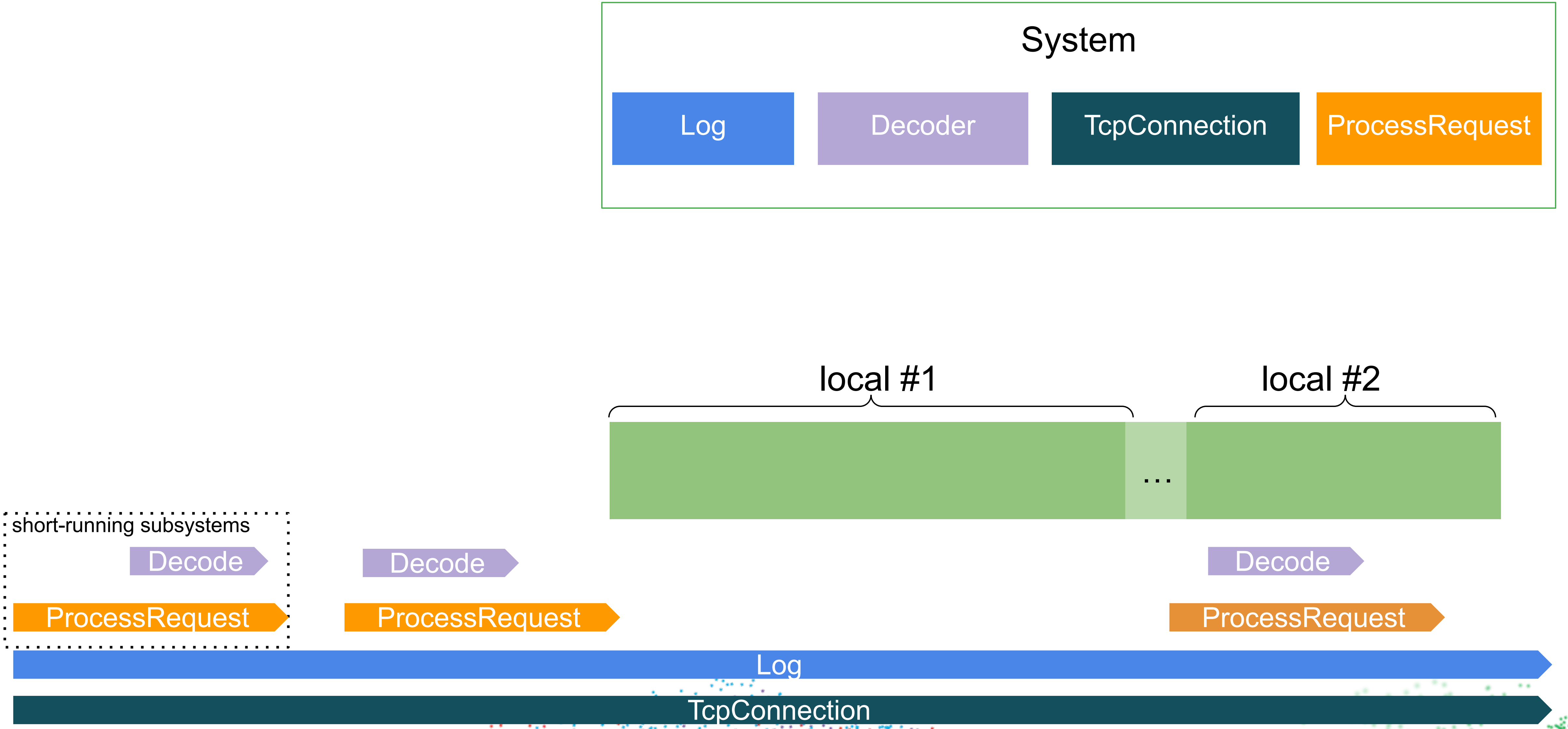
Defragmentation



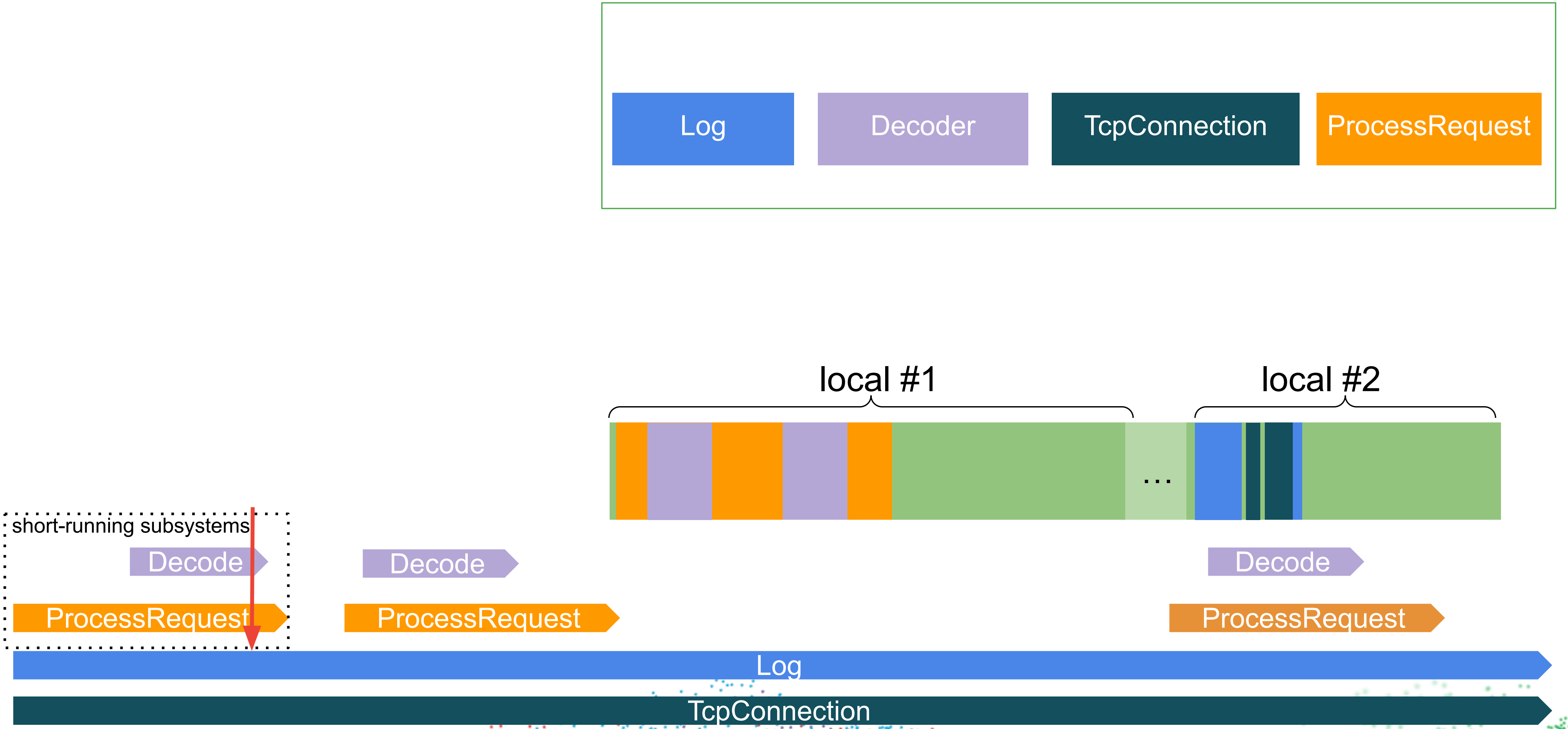
Fragmentation!



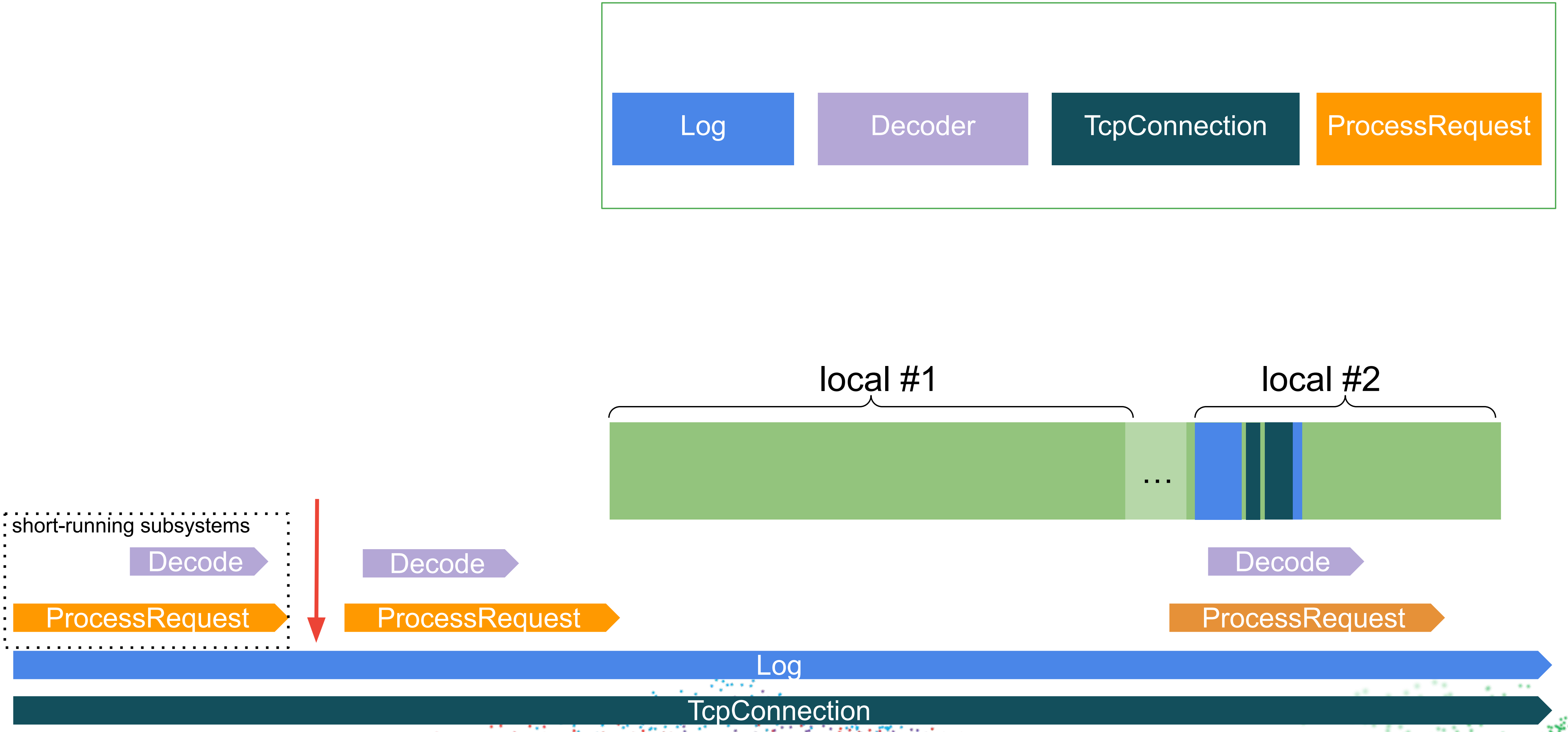
Defragmentation



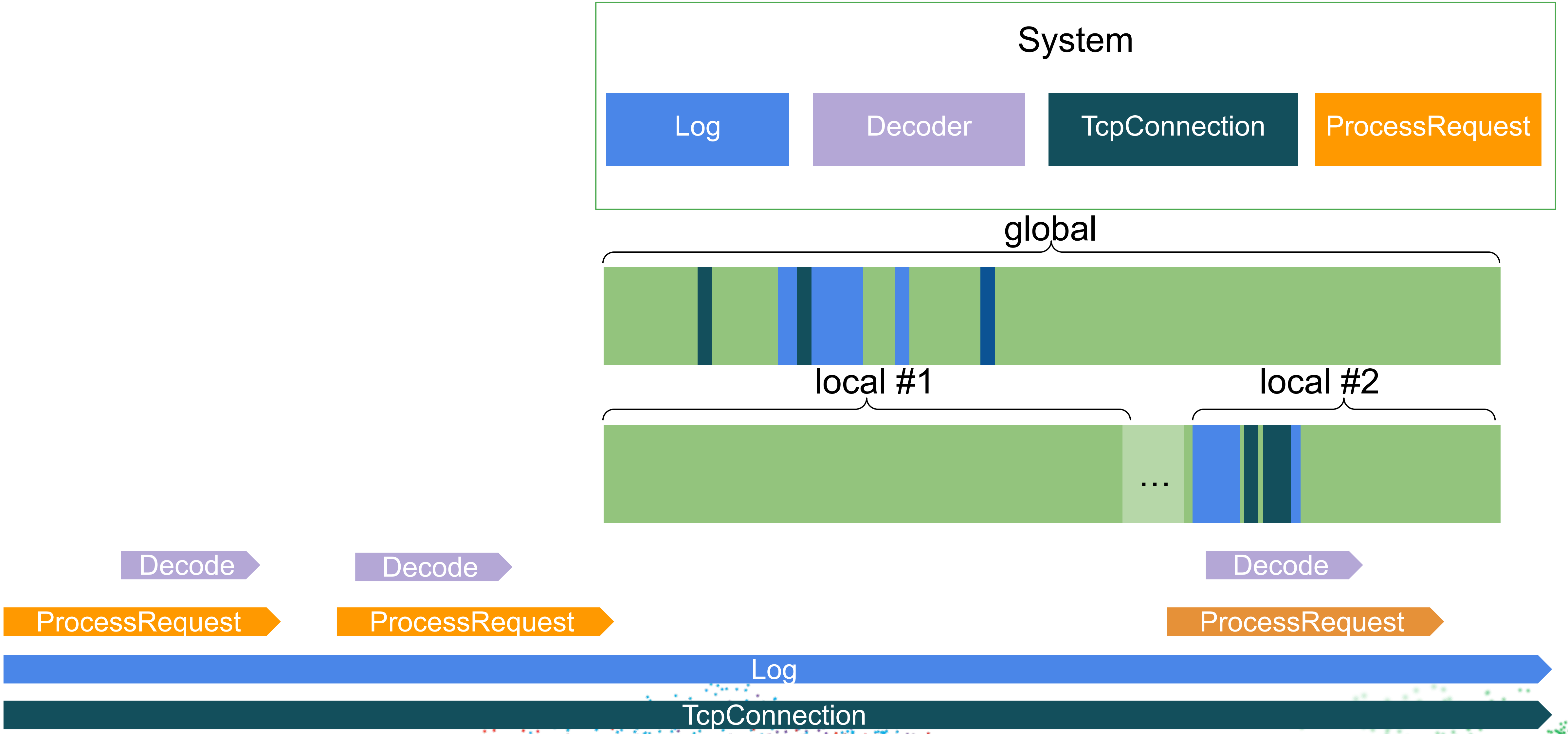
Defragmentation



Defragmentation

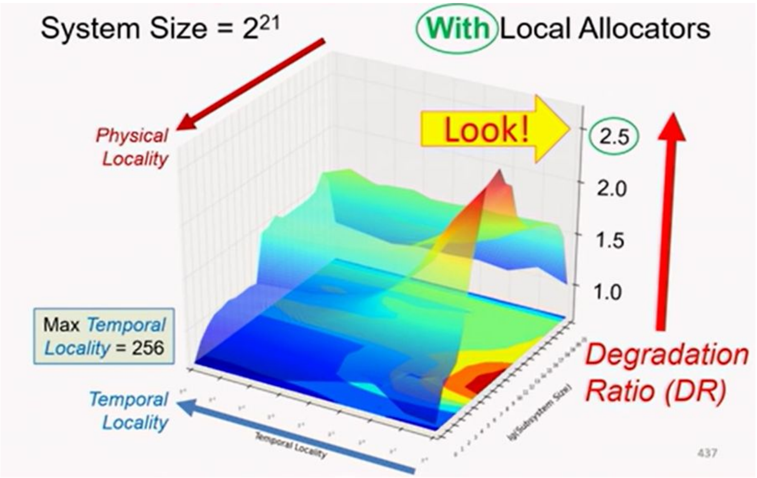


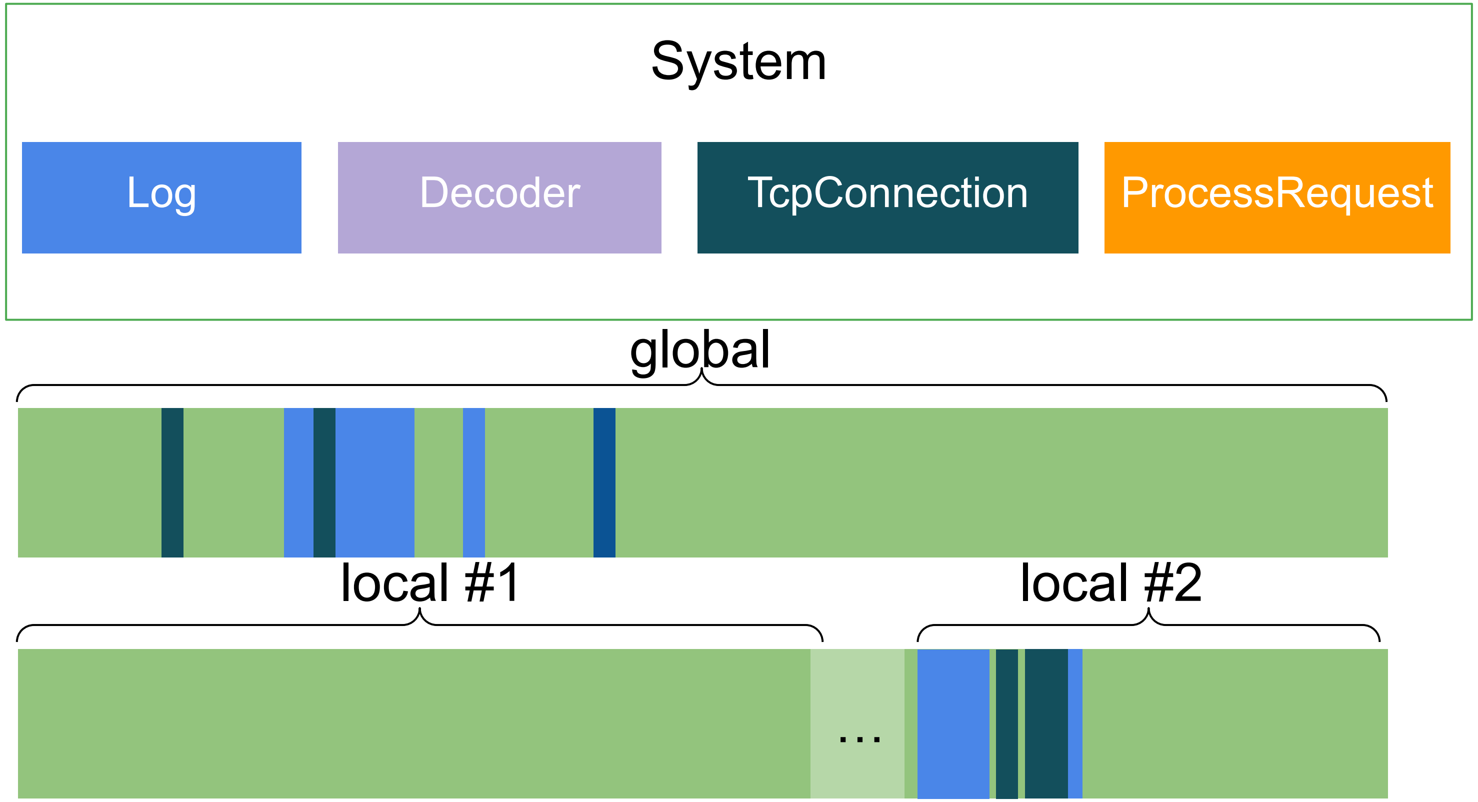
Defragmentation

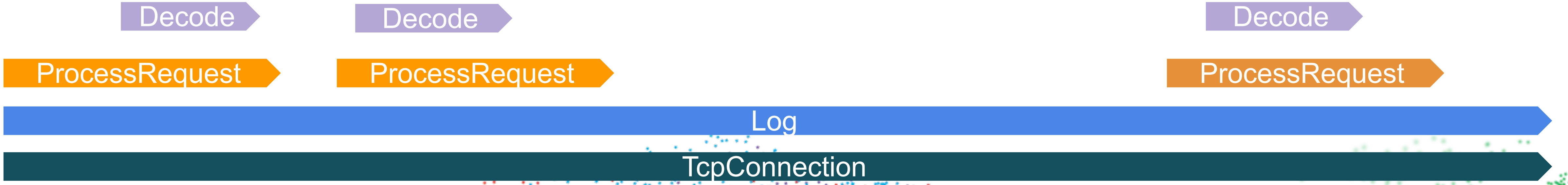


Defragmentation

`locality`







Using local allocator

```
void execution(int request_number,
               bslma::Allocator* allocatorL,
               bslma::Allocator* allocatorS){
    Subsystem longLiveSubsystem(allocatorL);
    Subsystem shortLiveSubsystem(allocatorS);

    for(int i = 1; i<=request_number; ++i){
        longLiveSubsystem.allocate();
        shortLiveSubsystem.allocate();

        longLiveSubsystem.allocate();
        shortLiveSubsystem.allocate();

        longLiveSubsystem.allocate();
        shortLiveSubsystem.allocate();

        shortLiveSubsystem.deallocate();
    }
}
```

```
cout<< "global allocator" <<endl;
execution(request_number, 0, 0);
```

```
cout<< "local allocator" <<endl;
bdlma::SequentialAllocator bsa_s;
bdlma::SequentialAllocator bsa_l;
execution(request_number, &bsa_l, &bsa_s);
```

Using local allocator

global allocator

Total non-mmapped bytes :	89 346 048
Total allocated space :	8 224
Total free space :	89 337 824
LargestAllocableBlock	48
# Free chunks	2 789 300

External Fragmentation = 0.99999946

Elapsed(ms) = 5049

local allocator

Total non-mmapped bytes :	135 168
Total allocated space :	8 224
Total free space :	126 944
LargestAllocableBlock	126 944
# Free chunks	1

External Fragmentation = 0

Elapsed(ms) = 4026

Additional Reading

1. glibc’s malloc

2. Tech talks:
    a. C++ allocators: Local(Arena) memory allocators - John Lakos - CppCon 2017
    b. What Programmers Should Know About Memory Allocation - S. Al Bahra, H. Sowa, P. Khuong - CppCon 2019
    c. Emery Berger “Mesh: Automatically Compacting Your C++ Application's Memory”- CppCon 2019
    d. Getting Allocators out of Our Way - Alisdair Meredith & Pablo Halpern - CppCon 2019

KEEP
CALM
AND
PROFILE
the MEMORY USAGE

Thank you!

https://TechAtBloomberg.com/cplusplus

https://www.bloomberg.com/careers

Bloomberg Engineering

TechAtBloomberg.com