System-wide Memory Management for Linux Embedded Systems

Revision 1.0

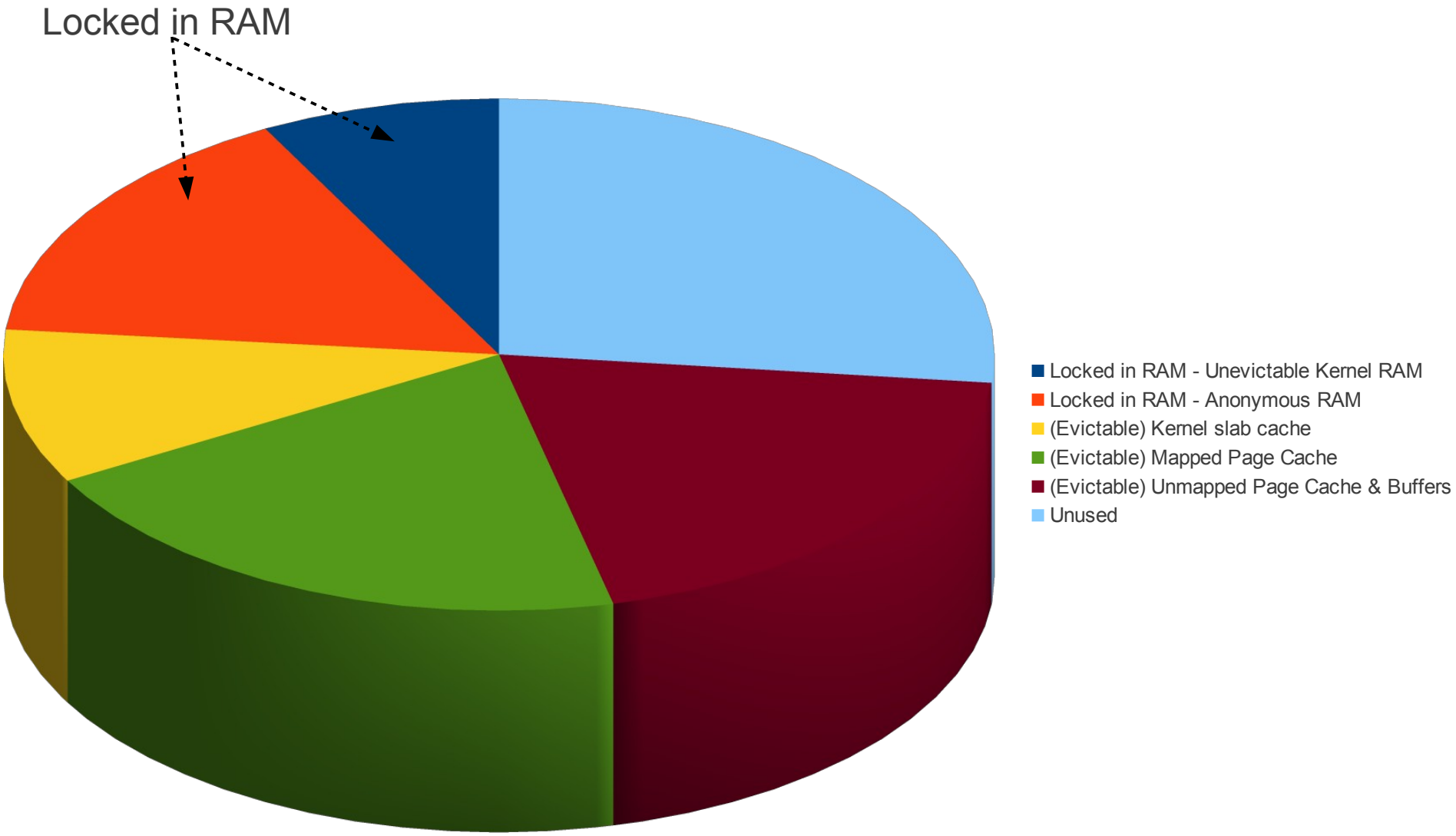
Presented by:
Howard Cochran
cochran@lexmark.com
Lexmark International
at:
Embedded Linux Conference
February, 2013

Agenda

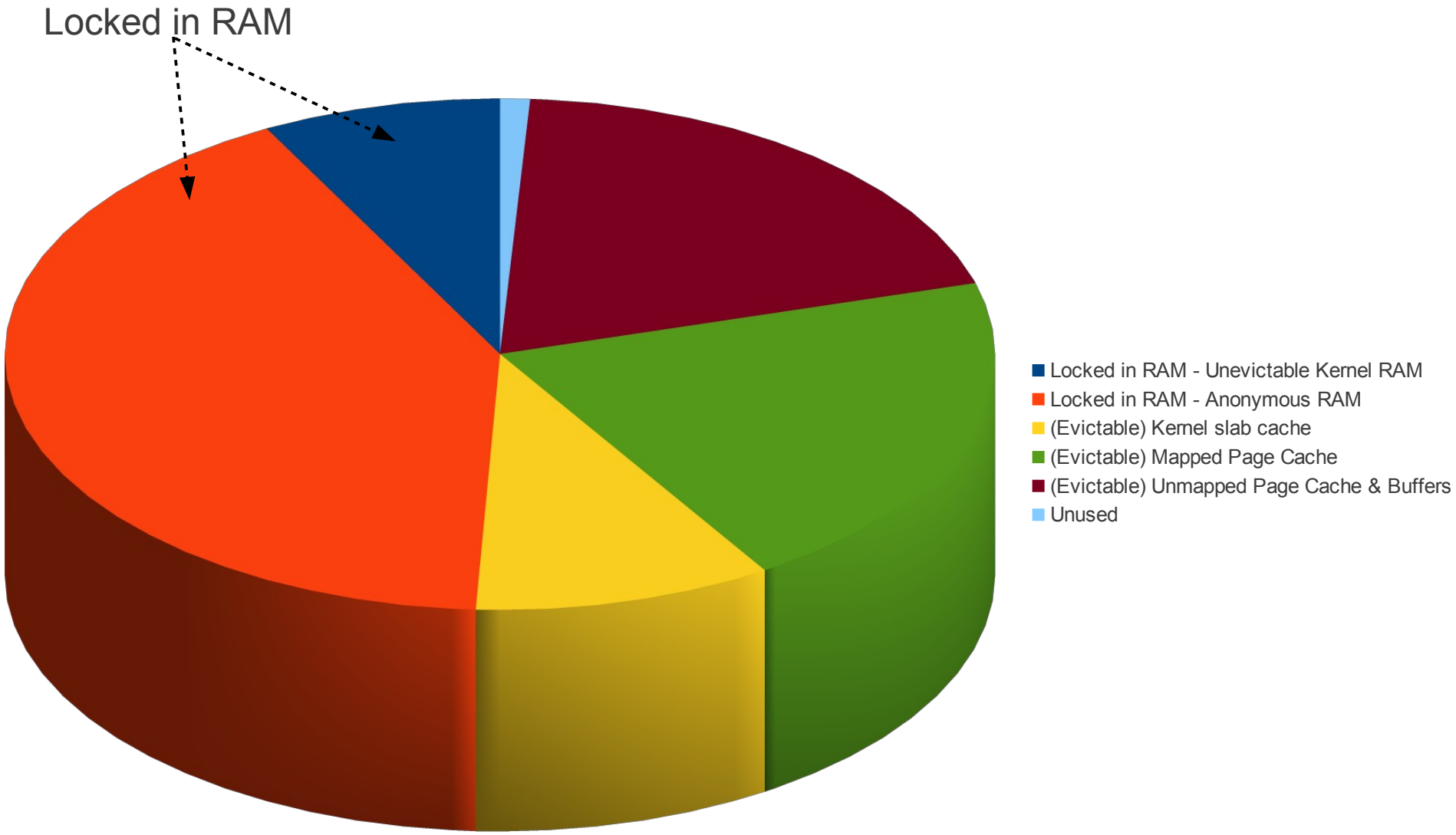
- The Problem:
  - How Linux kernel manages memory
  - Memory challenges in Lexmark devices
  - Limitations of glibc malloc for embedded
- Our Solution:
  - membroker service
  - ANR malloc & gmalloc
  - track_fs
  - Configuration
- Q&A

We're publishing these as Free (Libre) Software!

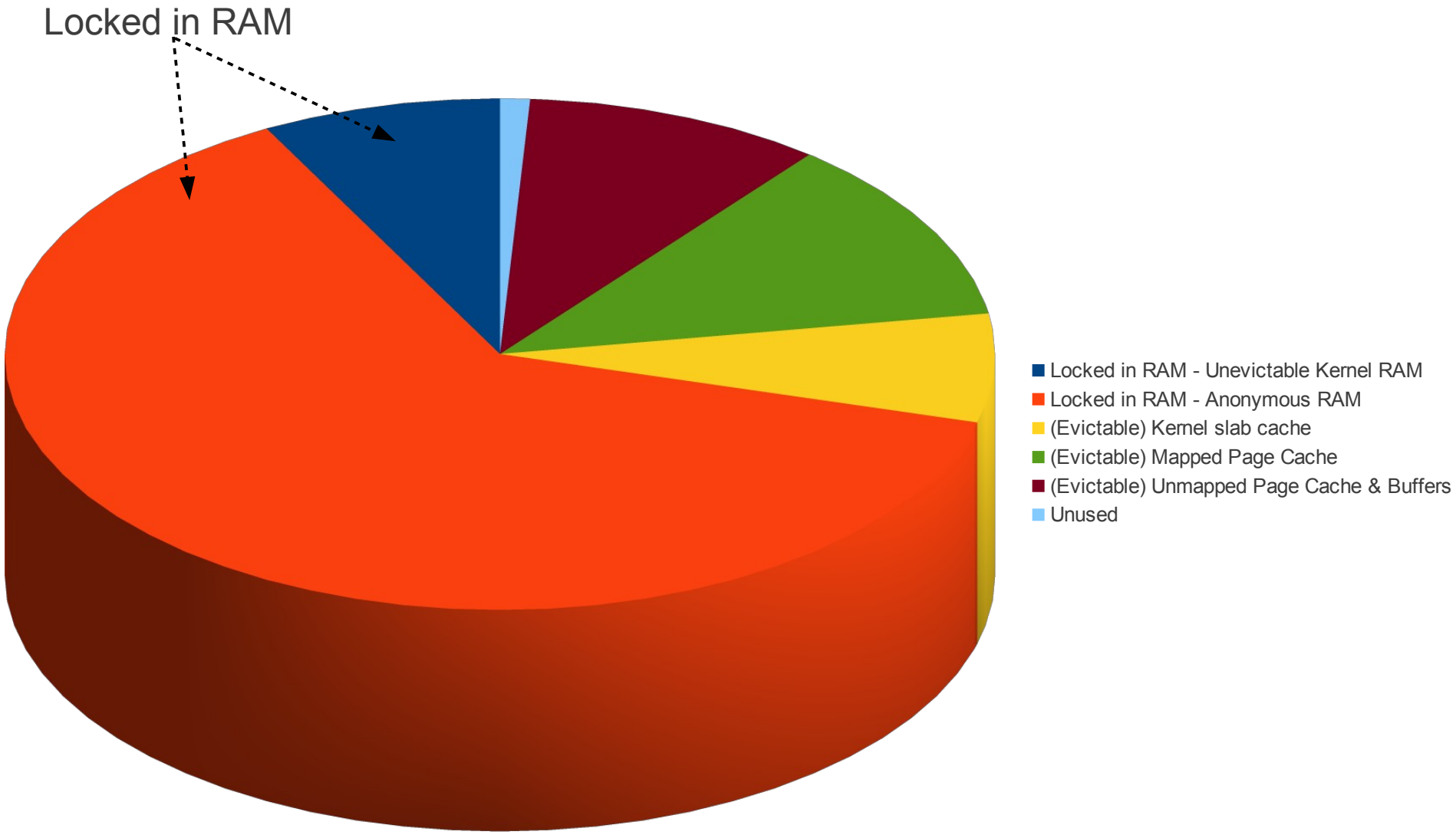
Memory Use – No memory pressure



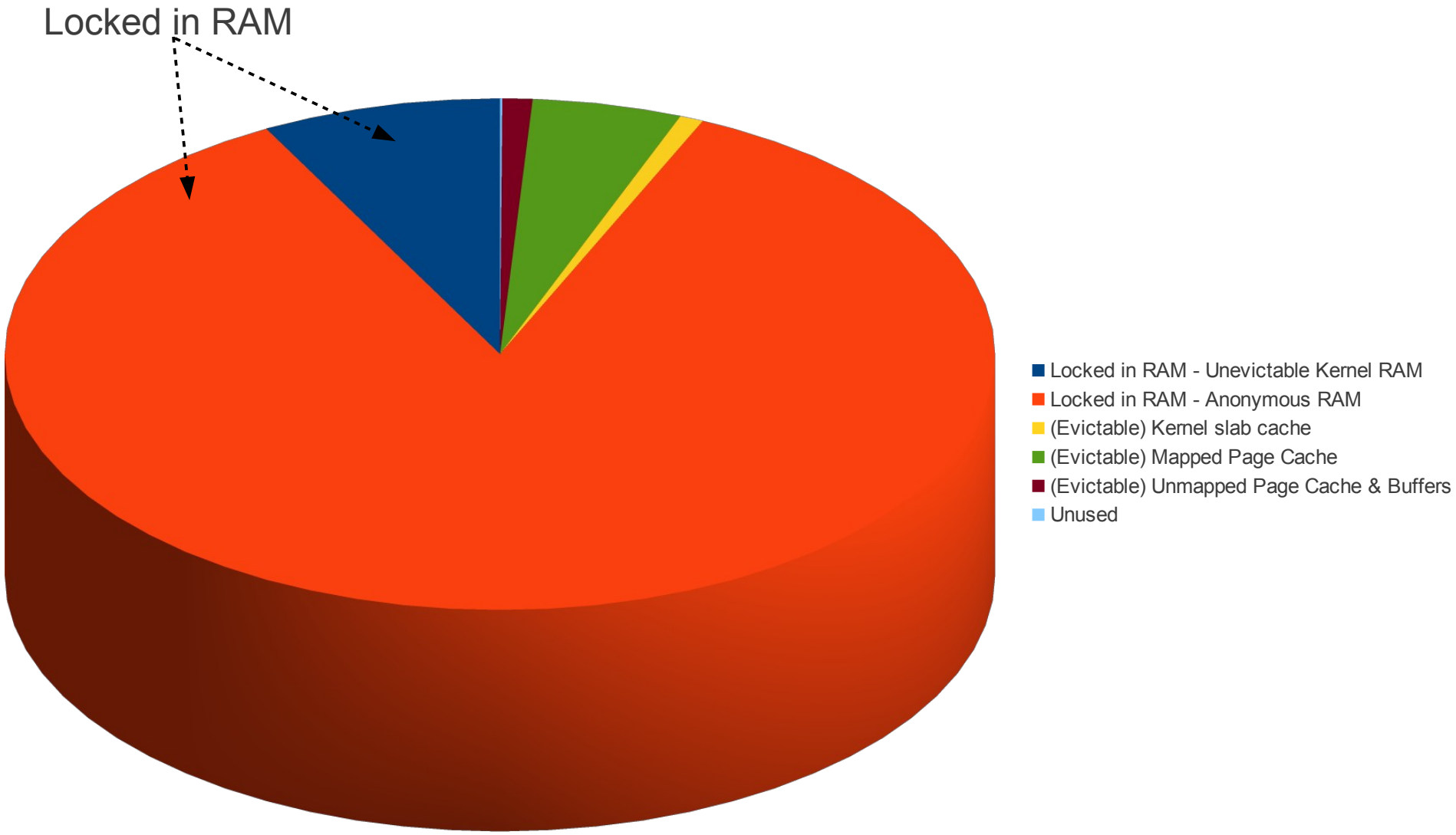
Memory Use – Light Pressure



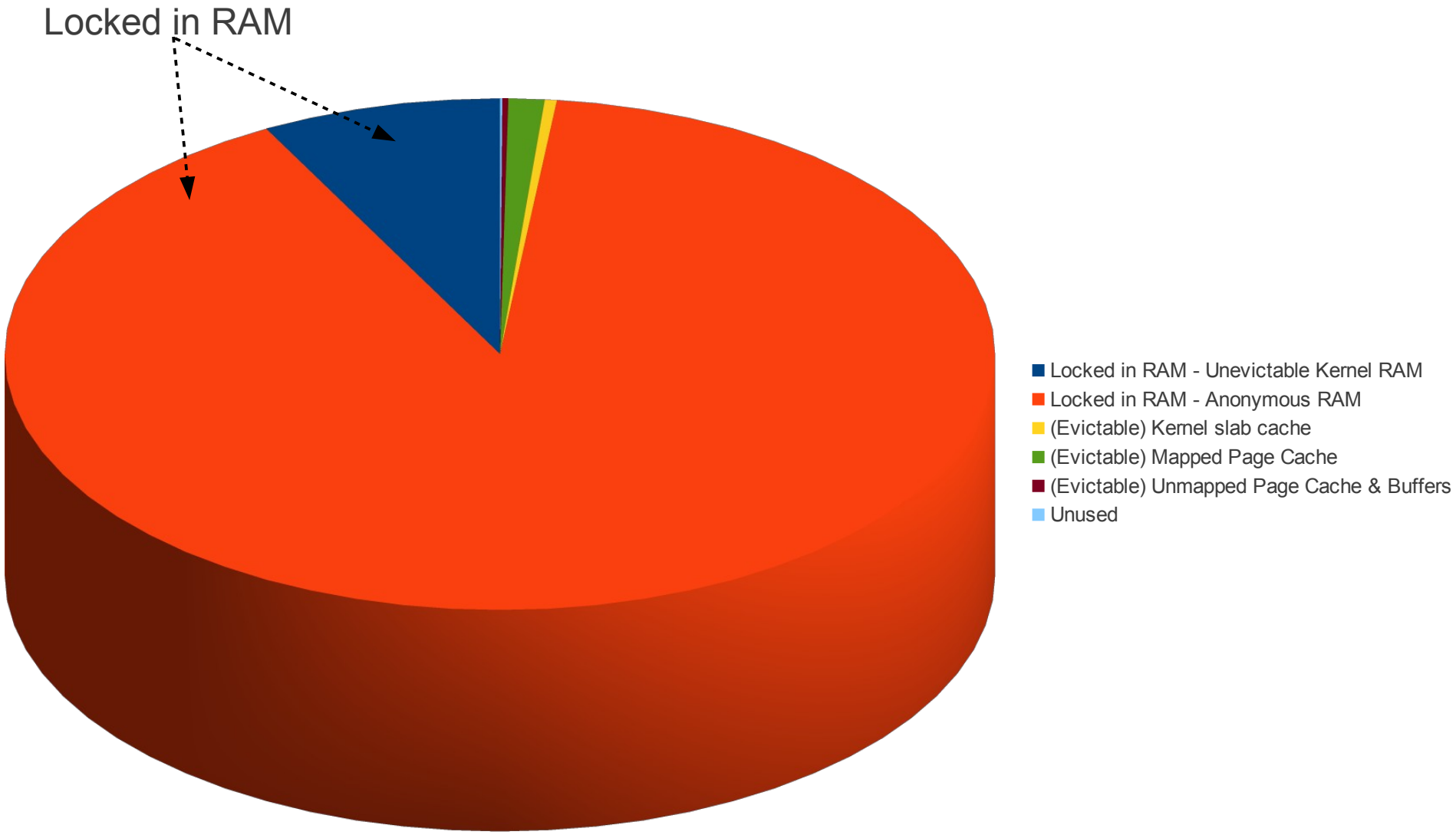
Memory Use – Anonymous Growing



Heavy Pressure – Thrashing



Extreme Pressure – Crash!



Linux without swap

- The ramp-up from "memory pressure" to "hard crash" is steep.
- out_of_memory killer - Kills an arbitrary process
  - There are /proc tunables to influence who OOM killer chooses
  - For embedded, ANY invocation of OOM killer could be fatal

Anonymous Memory

- **Locked into RAM** (unless you have swap)
- Are the majority, in my experience
  - Often 80+%
- Workstations use swap to provide virtually endless supply of anonymous pages.
- Developers get lazy - no one expects malloc() to fail

Agenda

- The Problem:
  - How Linux kernel manages memory
  - Memory challenges in Lexmark devices
  - Limitations of glibc malloc for embedded
- Our Solution:
  - membroker service
  - ANR malloc & gmalloc
  - track_fs
  - Configuration
- Q&A

My problem space:

My problem space

- A few “large” processes for core functionality:
  - Many threads.
  - Some threads are REALTIME ~10ms deadlines
  - Non-RT threads use lots of anonymous RAM
  - “GiveBack”: A malloc() can stall while other subsystems free caches or wait for previous processing to complete.
  - All allocations can fail.
- Many smaller processes in system
  - Typically off-the-shelf, built on glibc malloc
  - Some are long-running services
  - Occasional spikes in memory usage.
  - Code usually cannot tolerate malloc returning NULL.

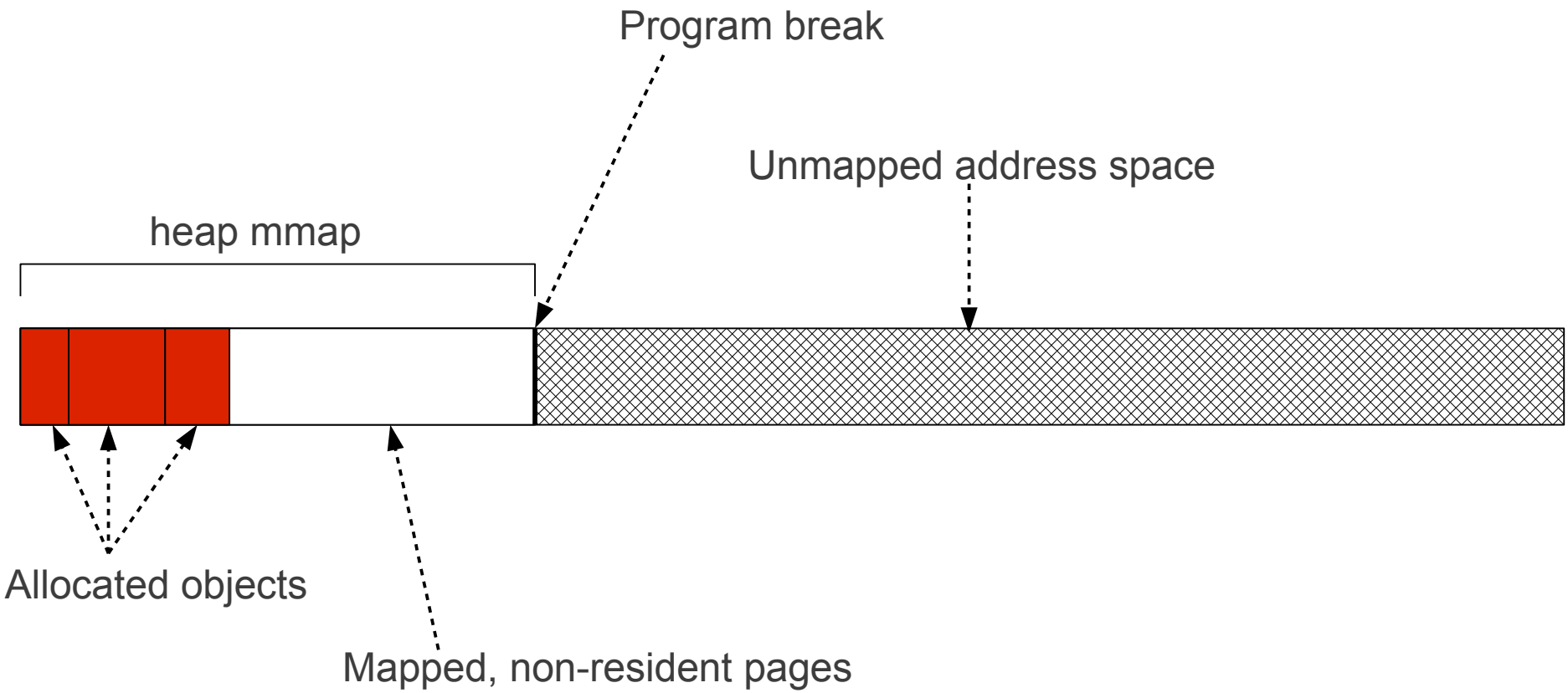
Themes

- Return pages to kernel when possible
- Coordinate anonymous memory between processes
- Avoid Real-time failures related to mmap_sem

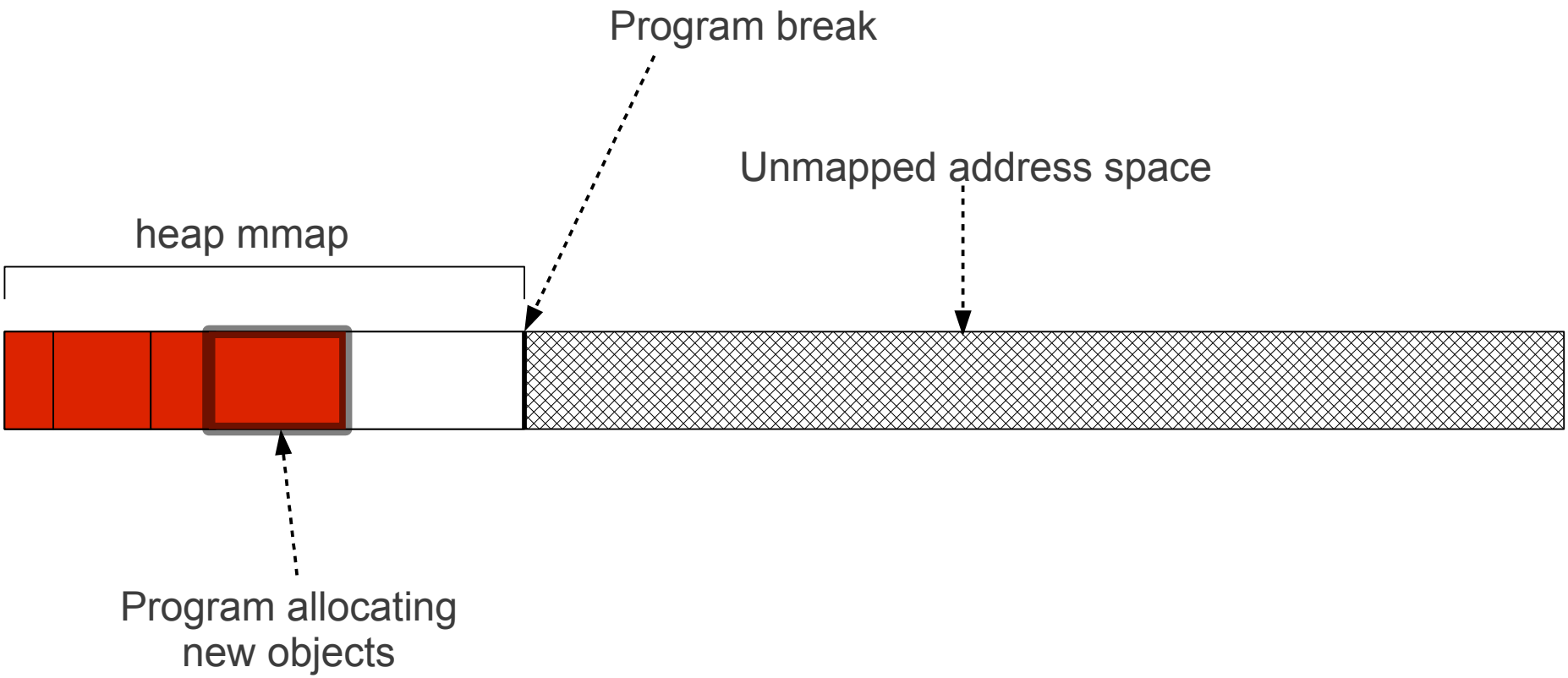
Agenda

- The Problem:
  - How Linux kernel manages memory
  - Memory challenges in Lexmark devices
  - Limitations of glibc malloc for embedded
- Our Solution:
  - membroker service
  - ANR malloc & gmalloc
  - track_fs
  - Configuration
- Q&A

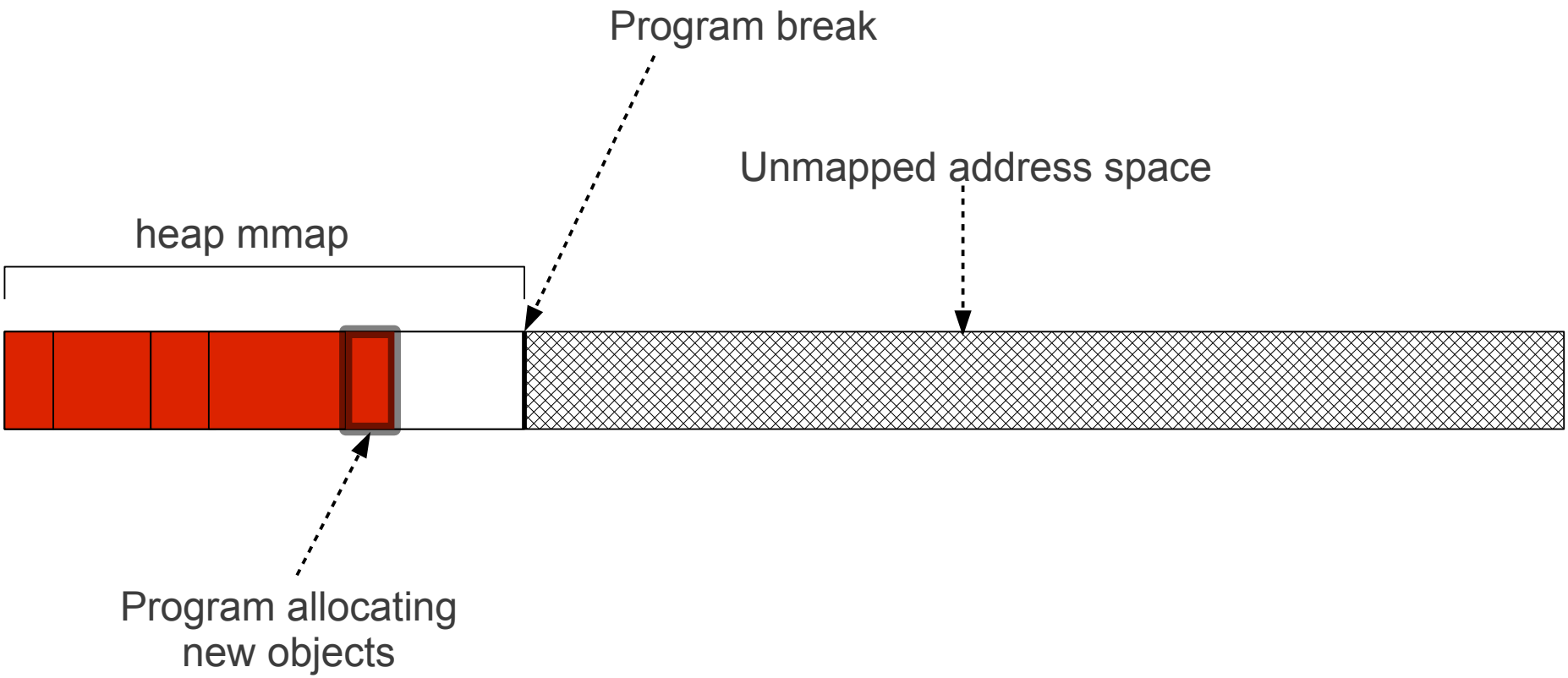
glibc heap


Program break
Unmapped address space
heap mmap
Allocated objects
Mapped, non-resident pages

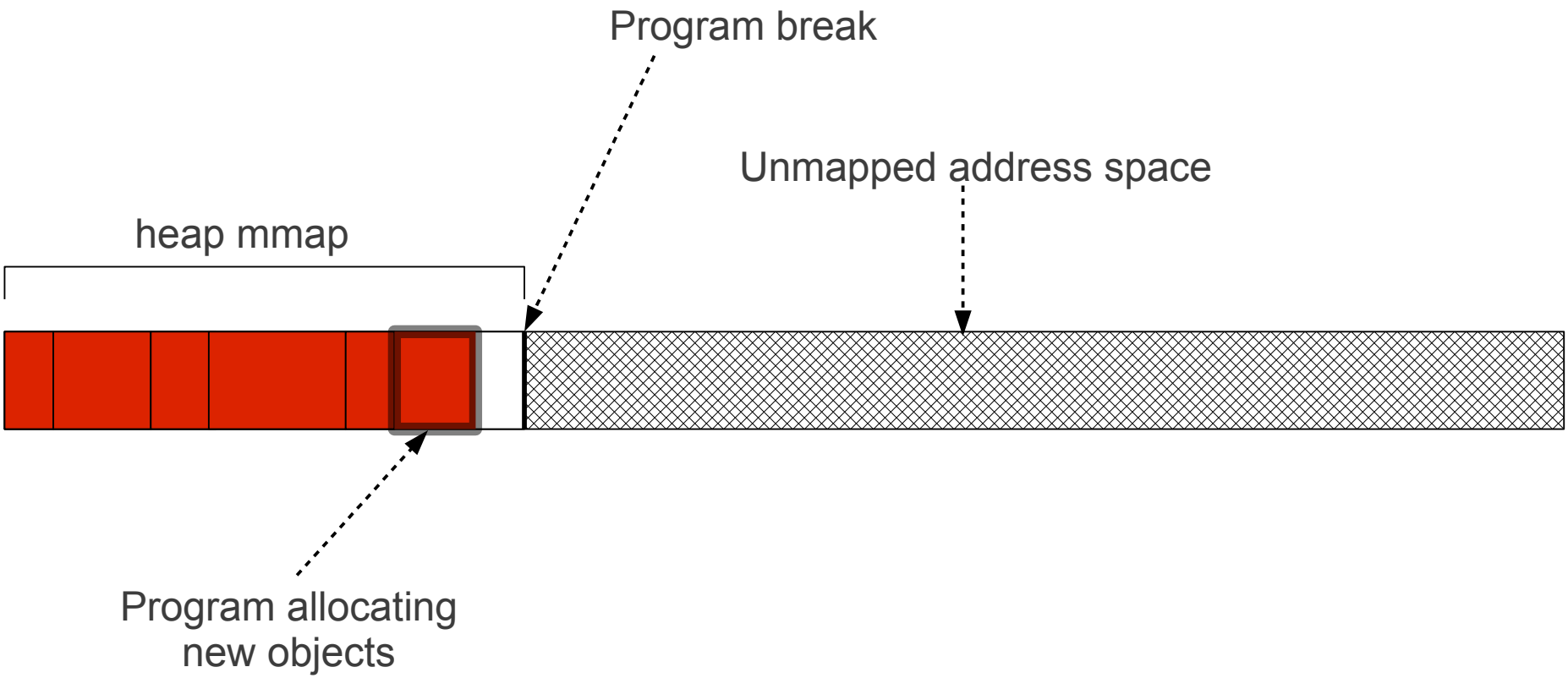
glibc heap - allocating



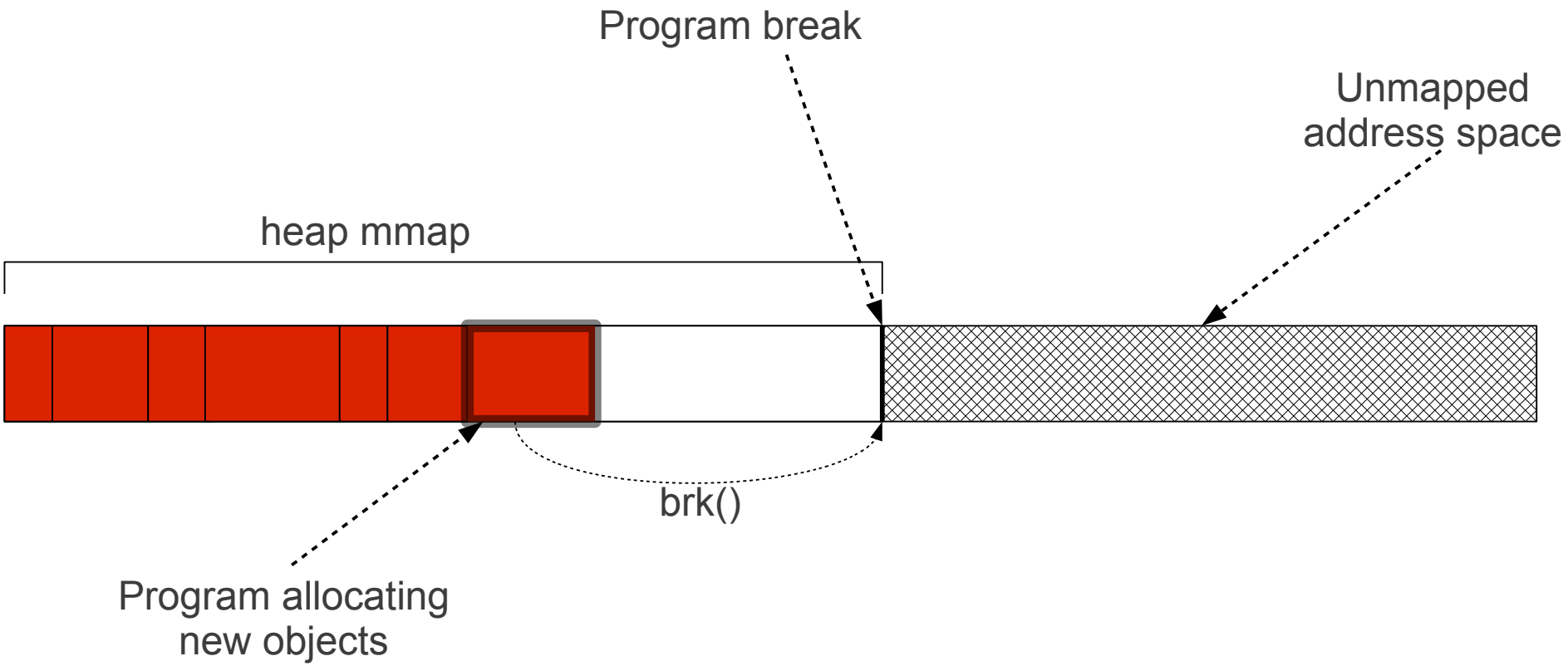
glibc heap – allocating



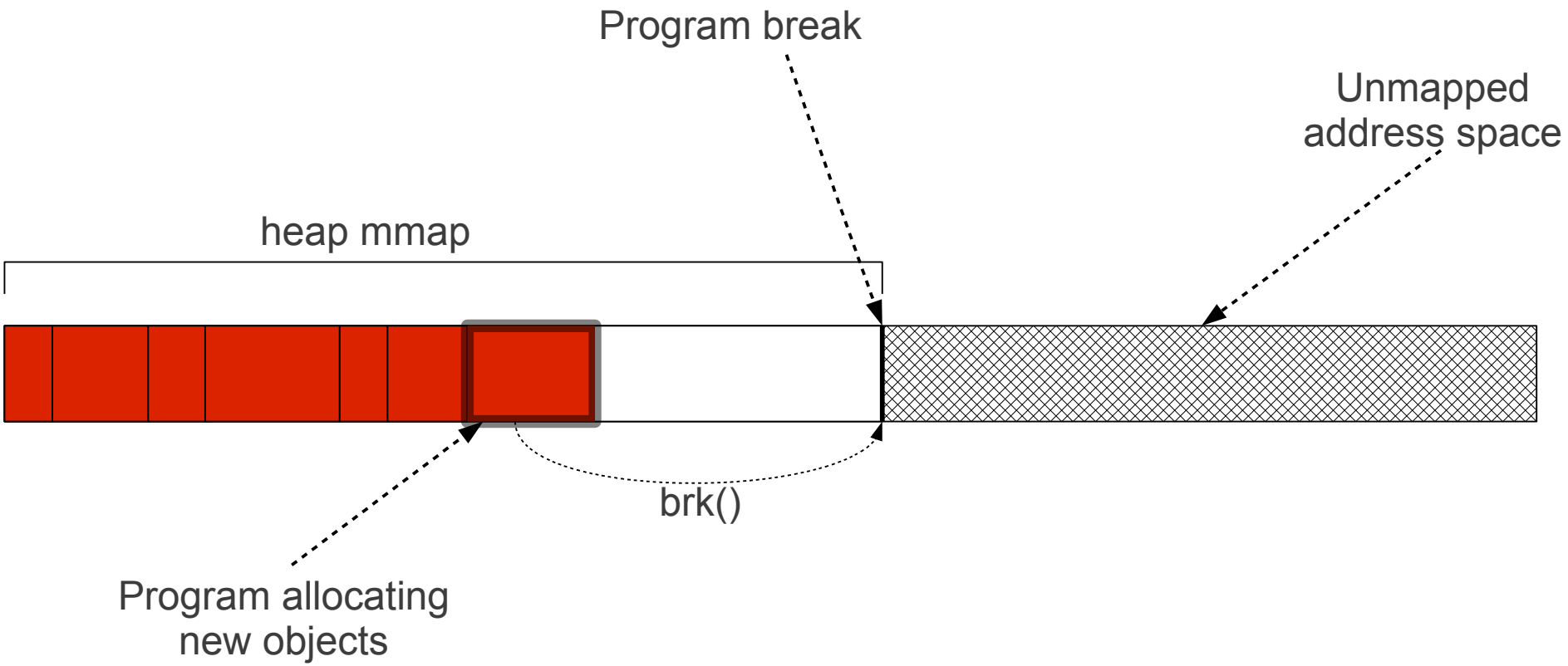
glibc heap – allocating



glibc heap – grows heap via brk()



brk() is bad for other realtime threads



brk() modifies a kernel vma.
down_write(current->mm->mmap_sem)
In-process Realtime threads fail deadlines!

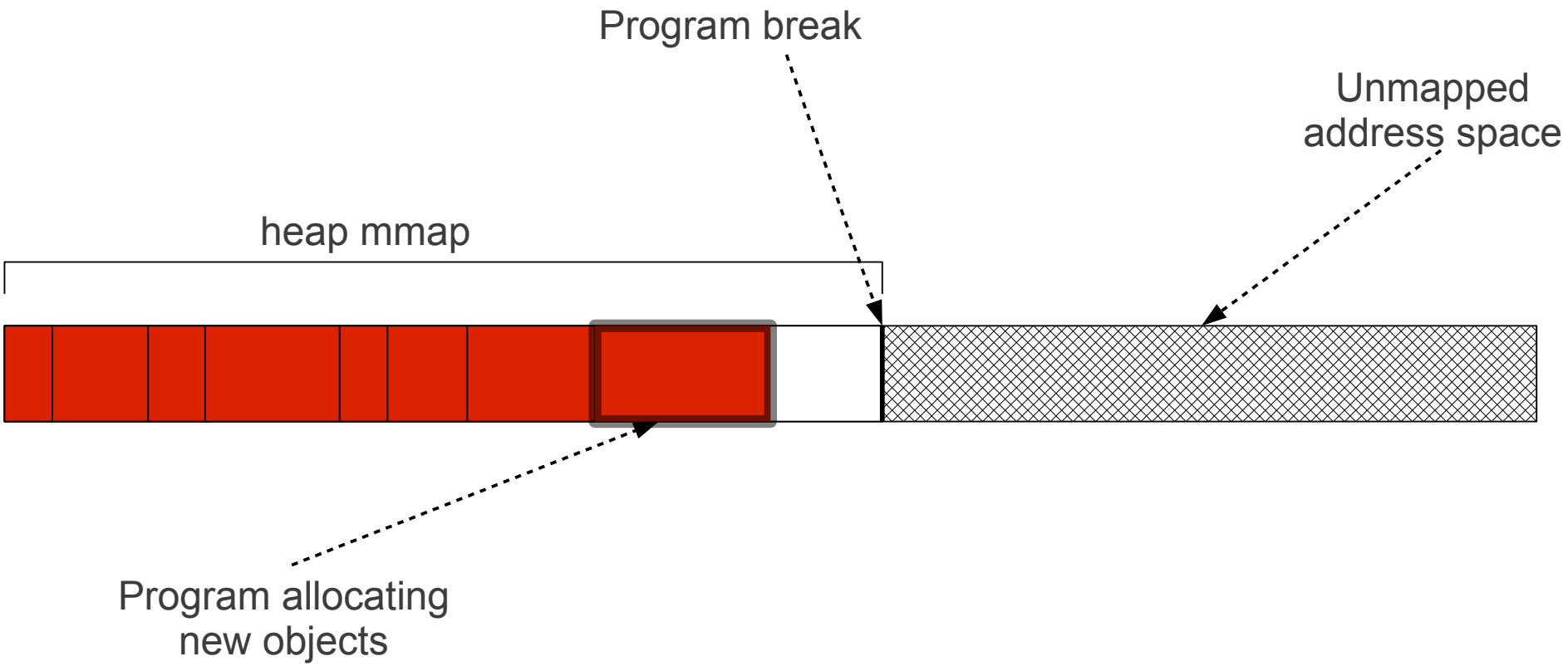
mmap_sem

- Is a read/write semaphore in the kernel
  - Can have many concurrent readers
  - Can have only one writer.
  - When writer has a lock, must be no readers
  - If a writer is waiting to acquire lock, new attempts to get read lock will block, even if reader is realtime.
- It is per thread group (i.e. per-mm)
- **No priority inheritance –** causes priority inversion
- Need write-lock to modify process's mmap

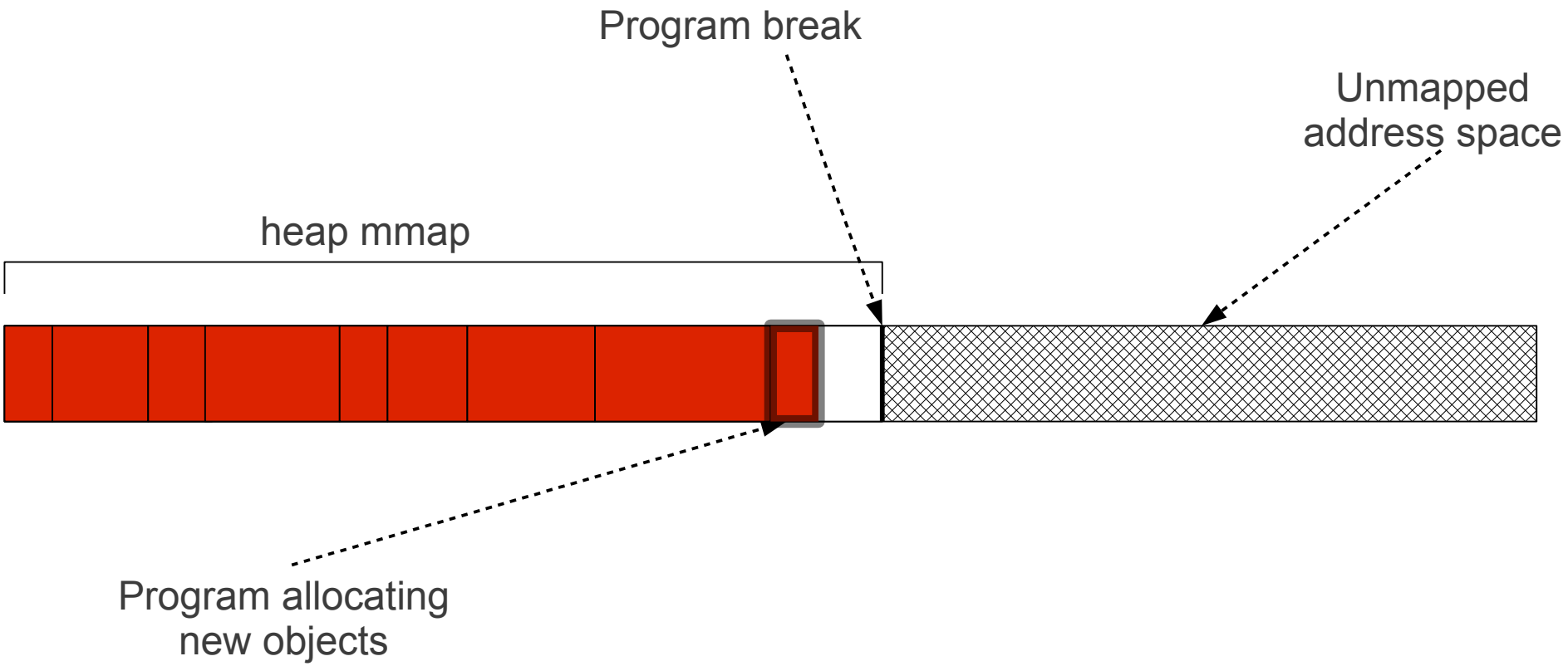
mmap_sem (continued)

- Realtime threads often need read-lock
  - Minor page faults
  - do_cache_op()
  - get_user_pages()
  - copy_to_user – some implementations
  - signal delivery

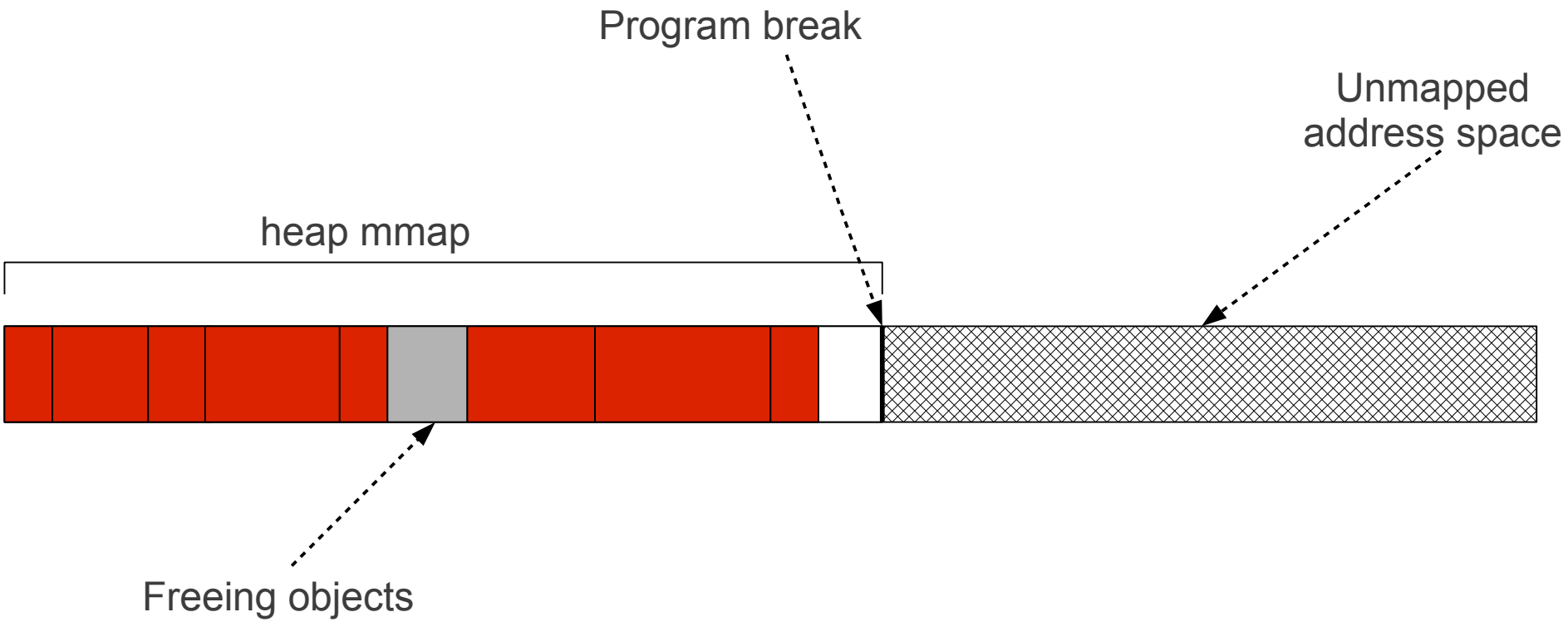
glibc heap – allocating



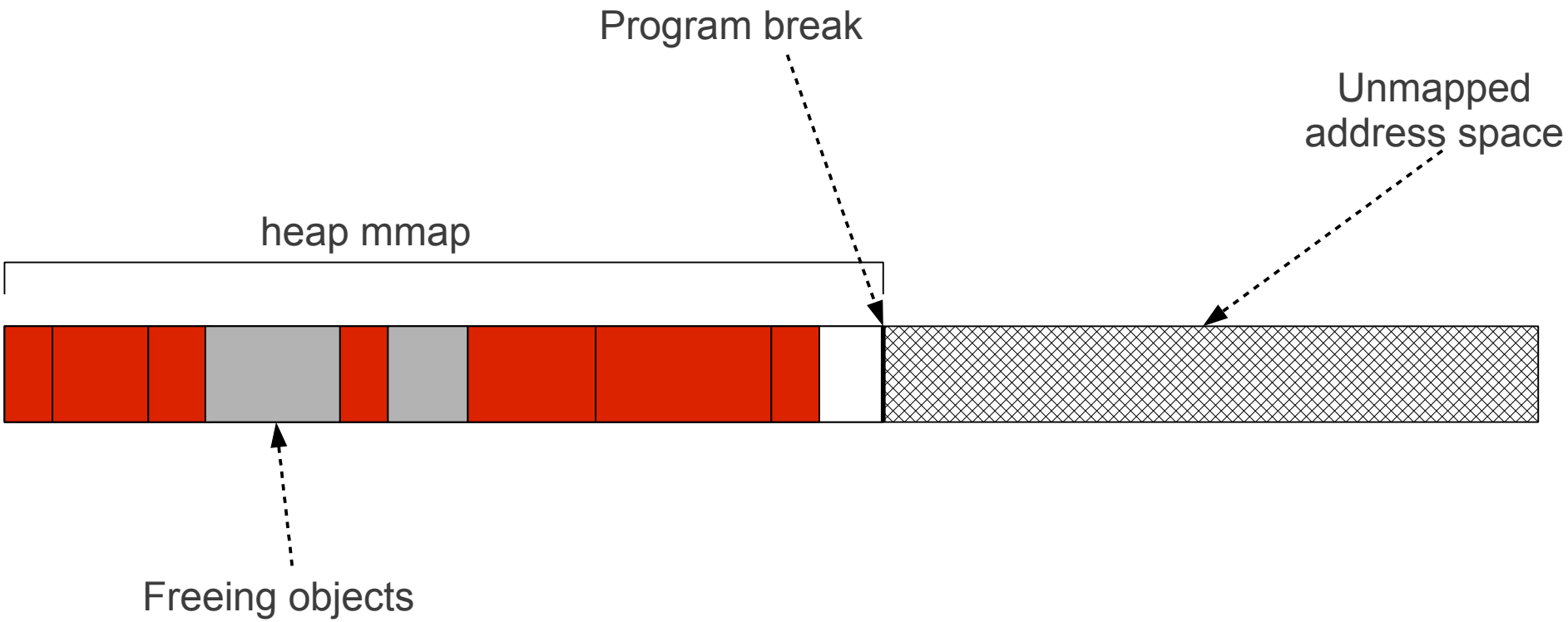
glibc heap – allocating



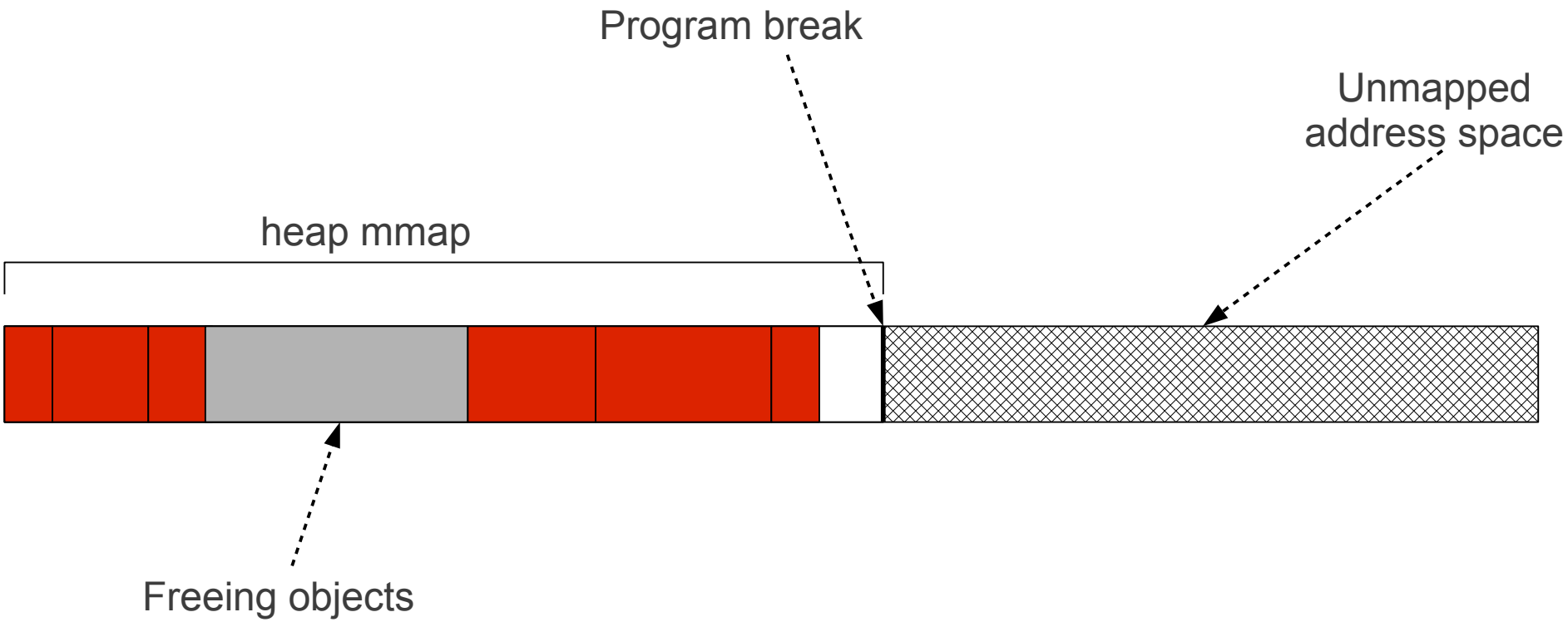
glibc heap – freeing



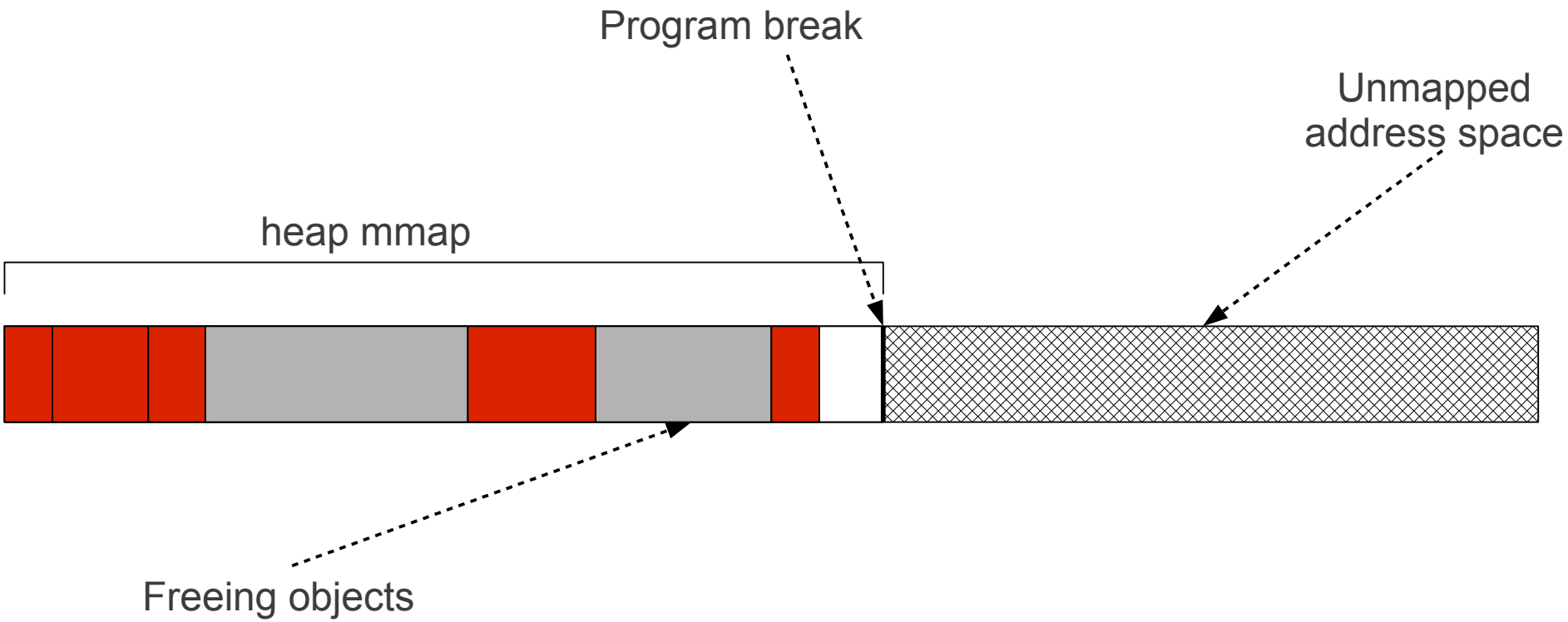
glibc heap – freeing



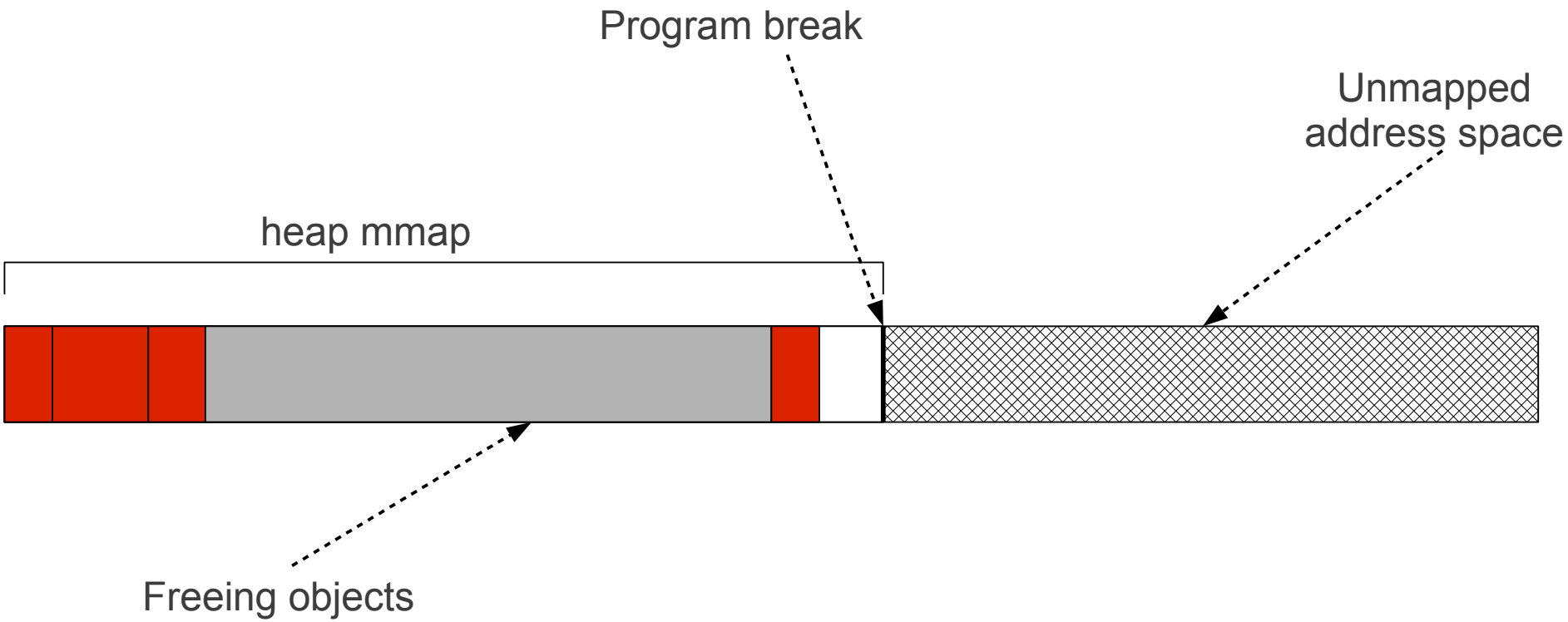
glibc heap – freeing



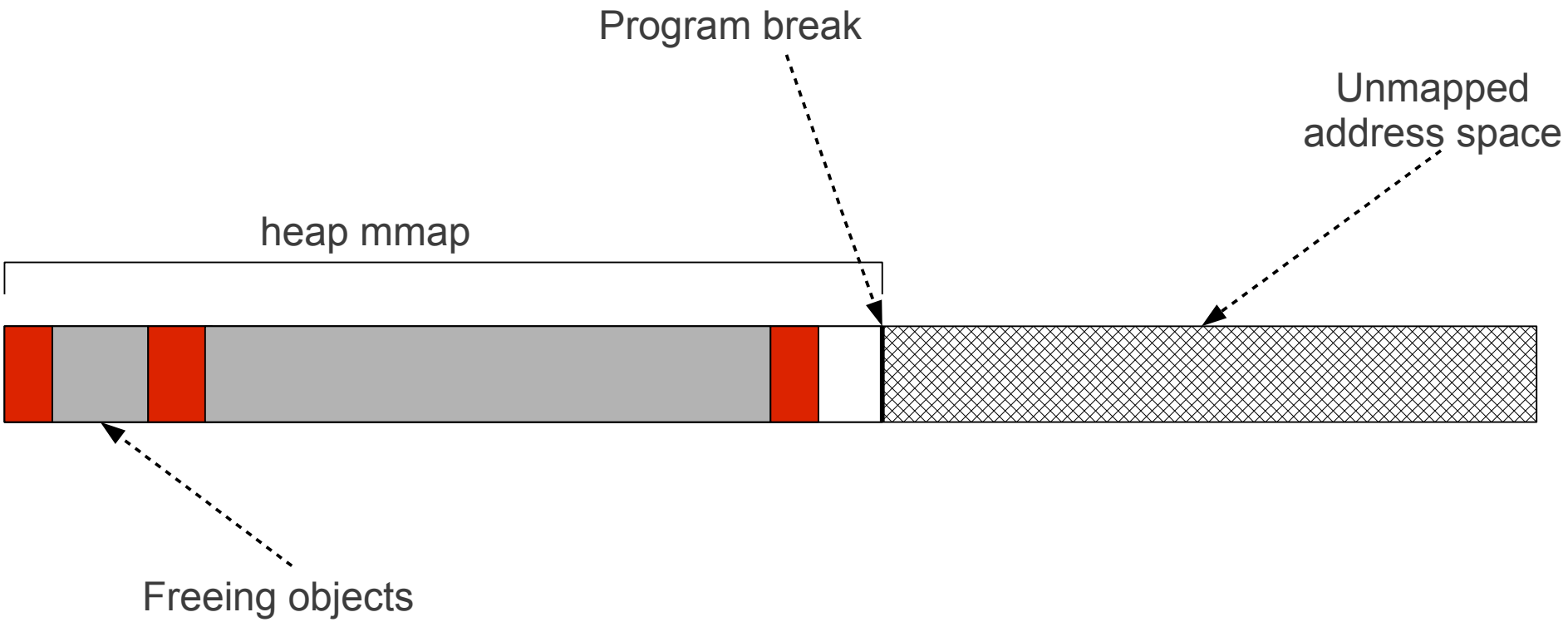
glibc heap – freeing



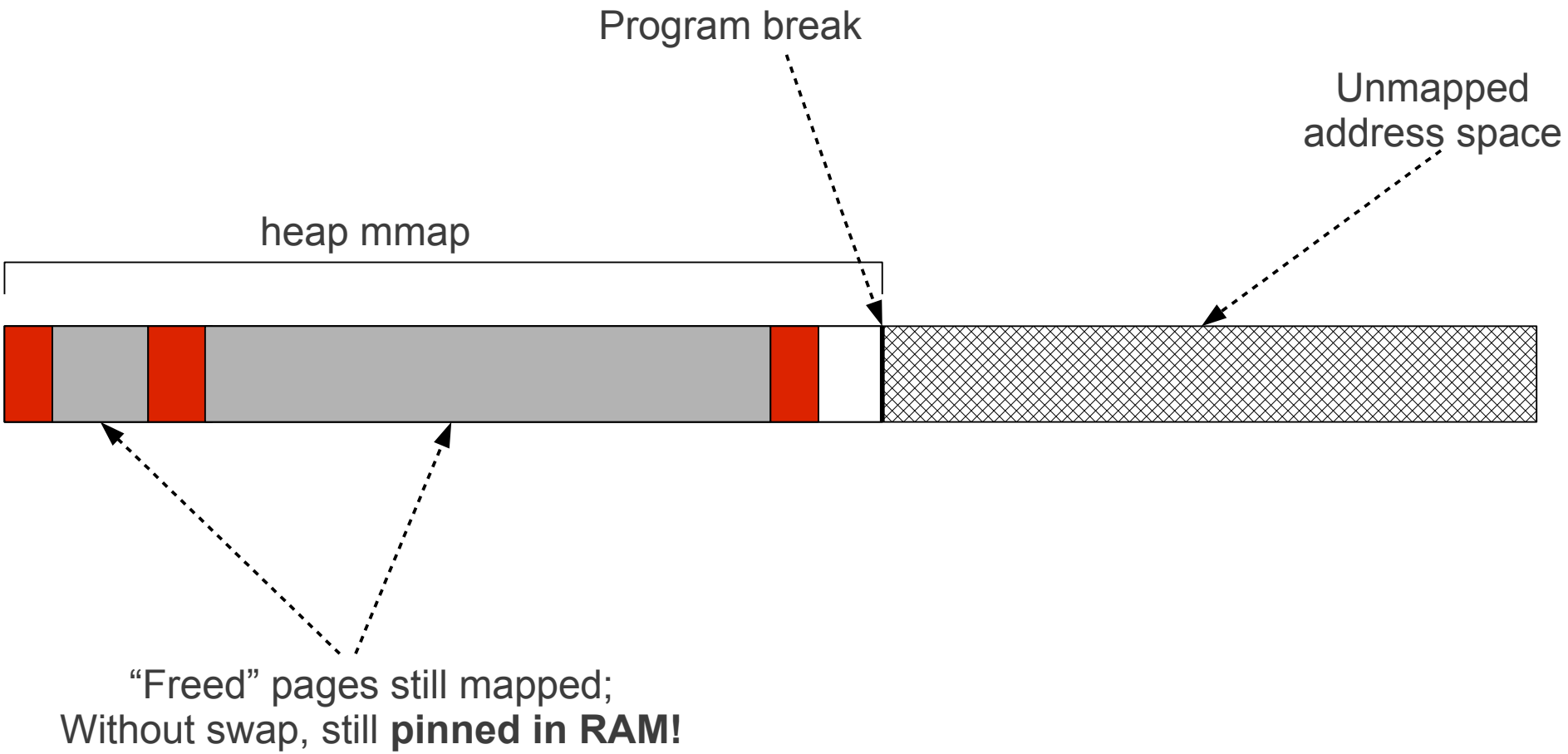
glibc heap – freeing



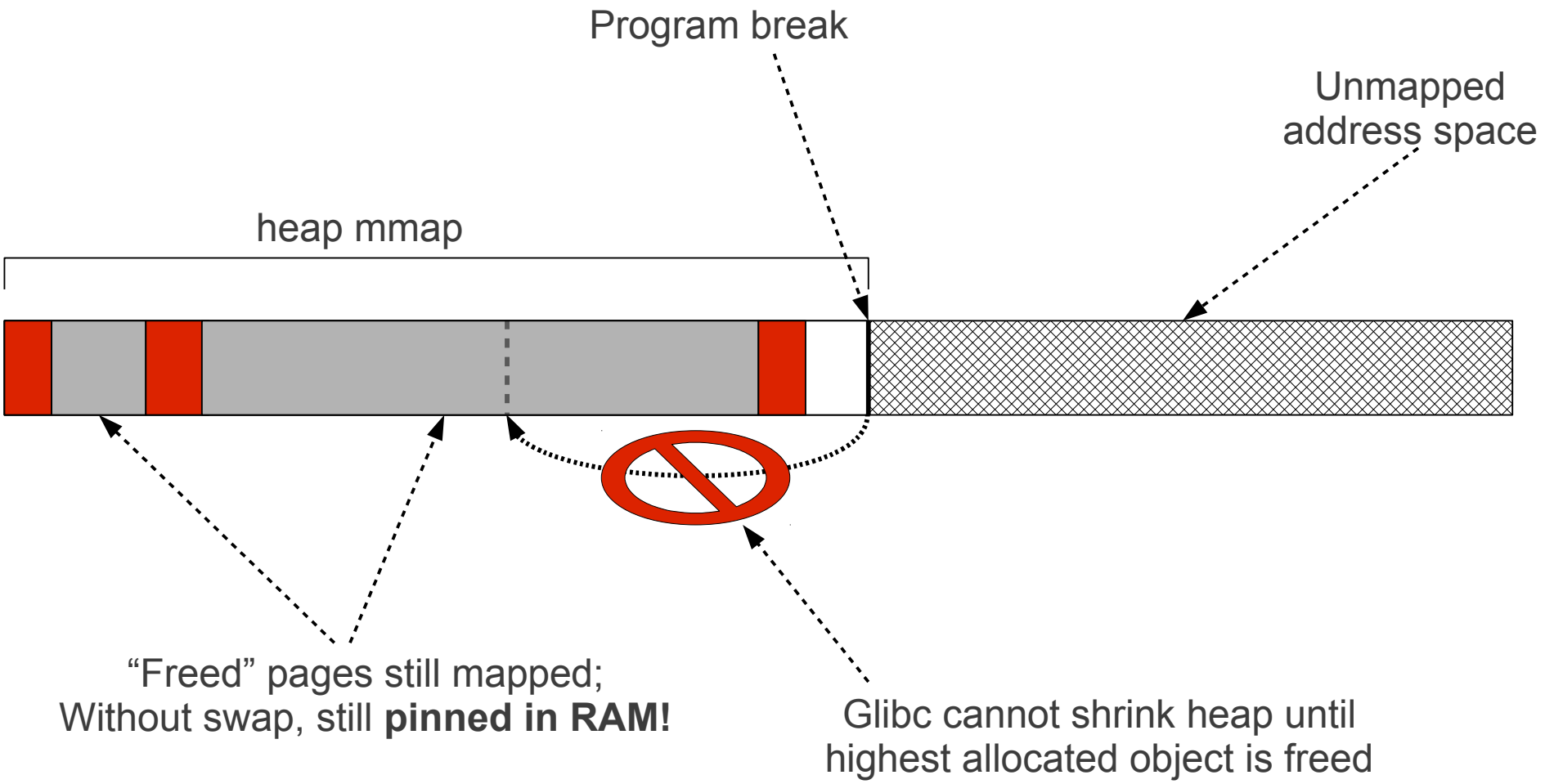
glibc heap – freeing



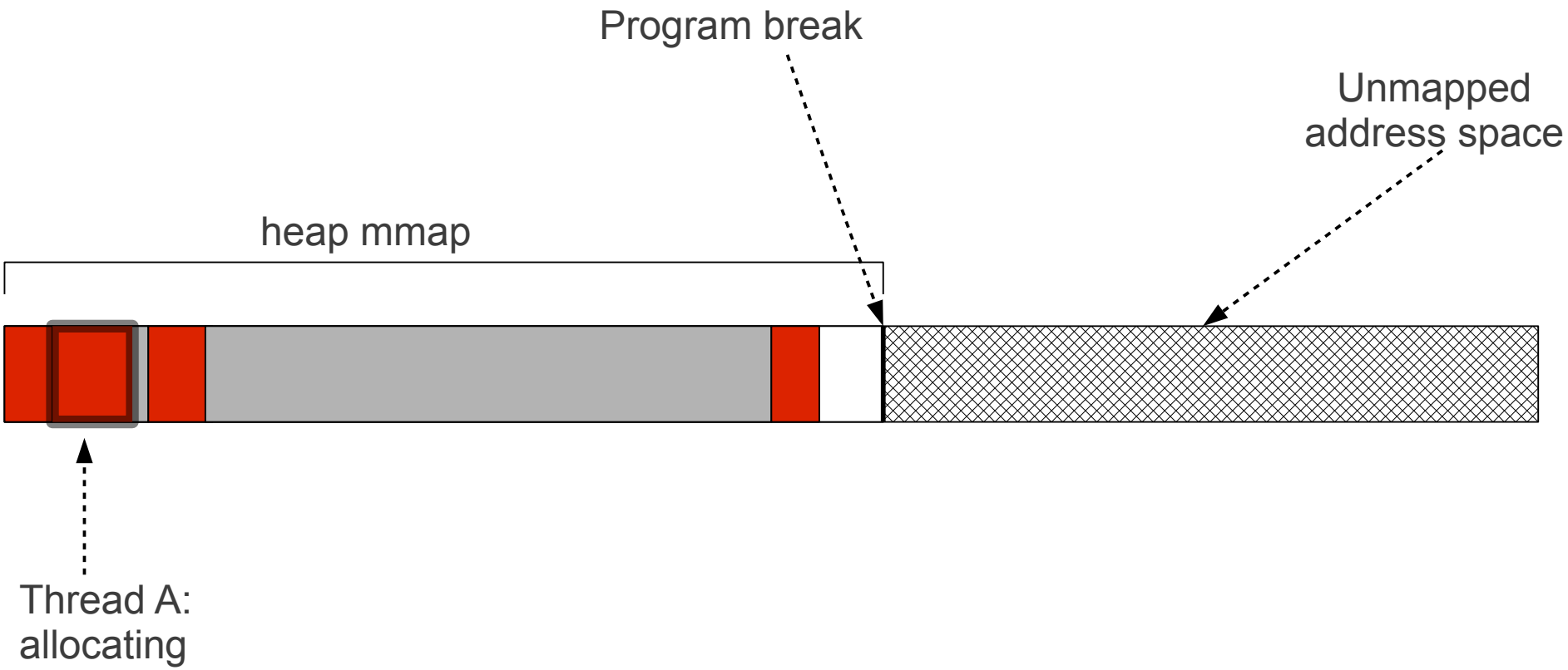
glibc heap – freed pages still pinned!



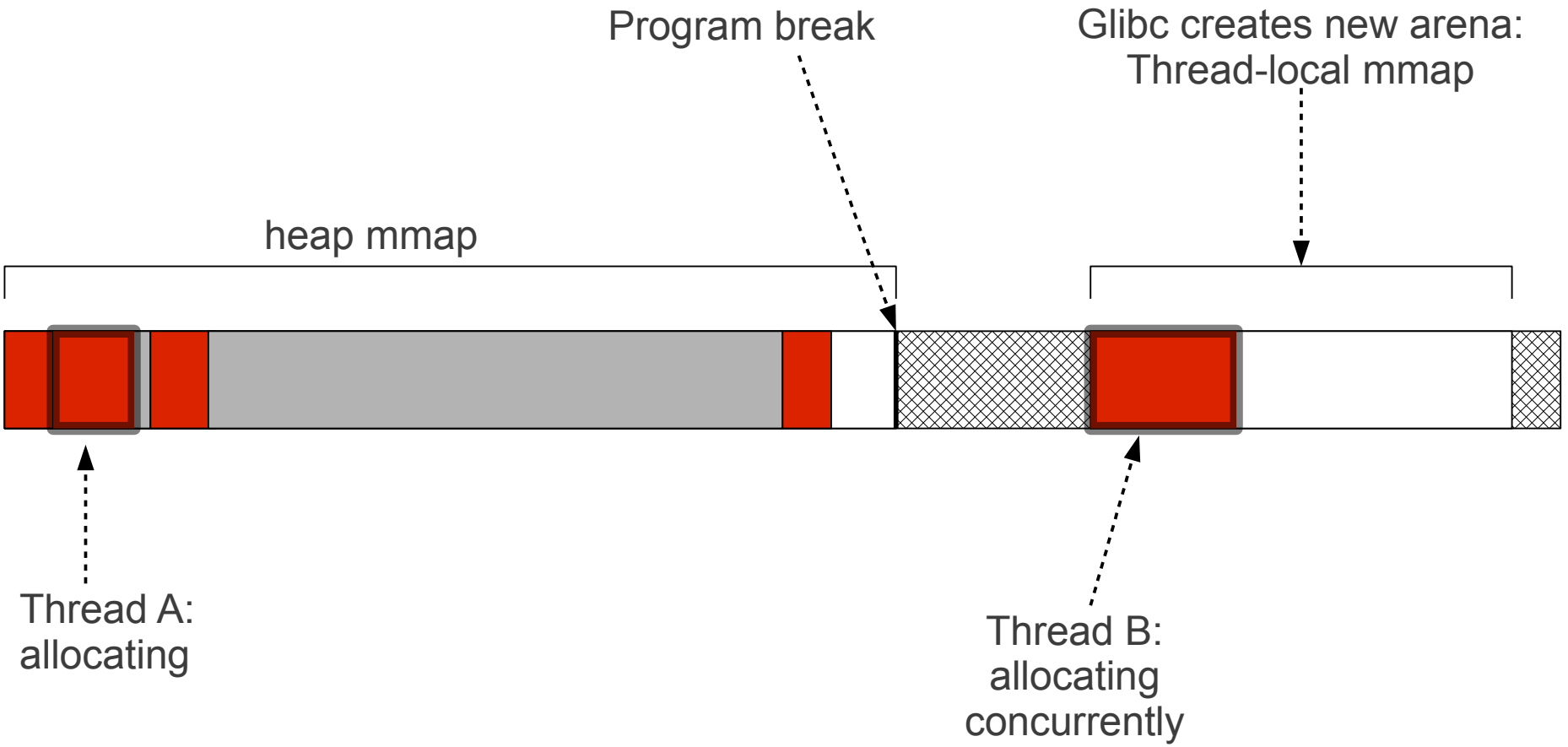
glibc heap – cannot shrink heap



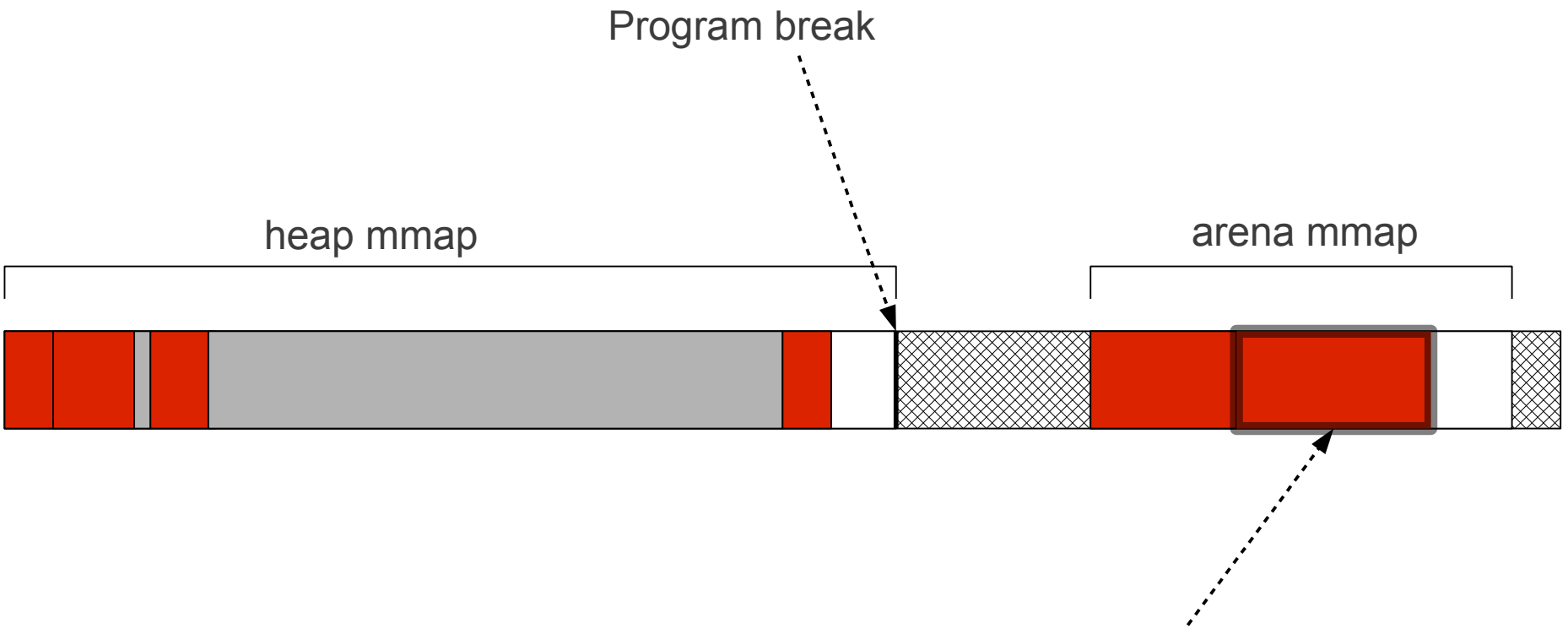
glibc heap – concurrent allocation



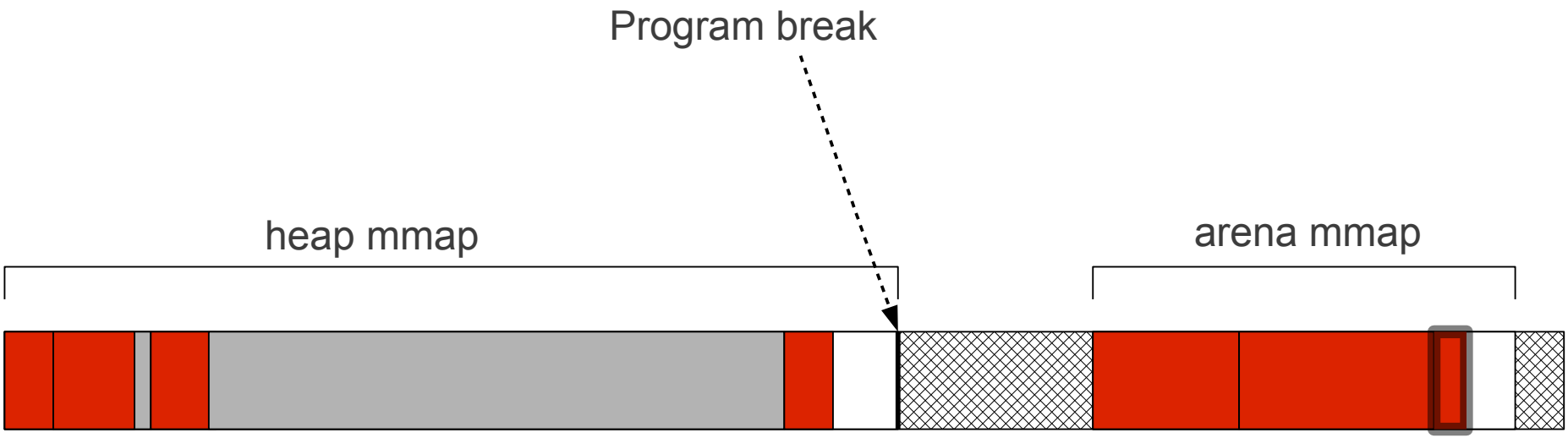
glibc heap – concurrent allocation



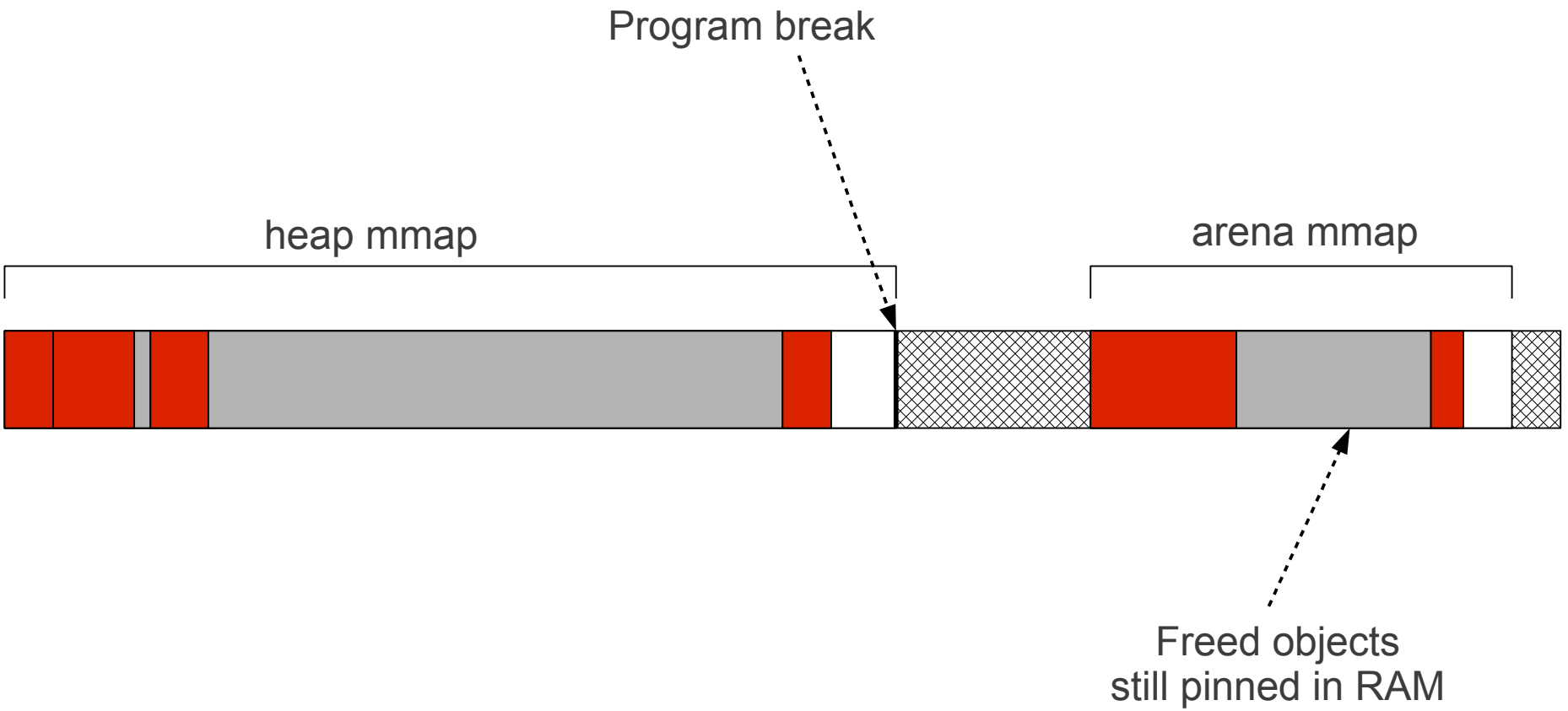
glibc heap – alloc from arena



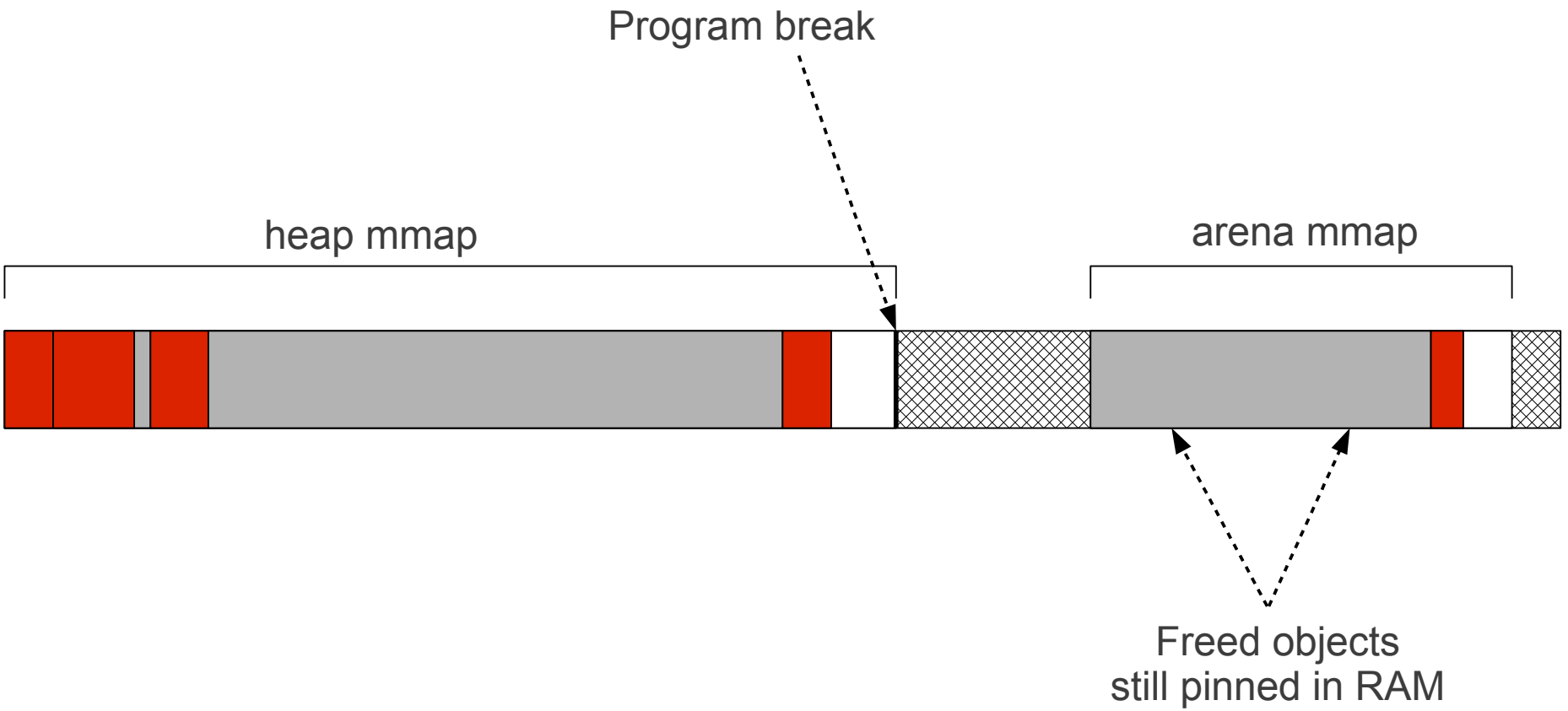
glibc heap – alloc from arena



glibc heap – free within arena



glibc heap – free within arena (pinned!)



Agenda

- The Problem:
  - How Linux kernel manages memory
  - Memory challenges in Lexmark devices
  - Limitations of glibc malloc for embedded
- Our Solution:
  - membroker service
  - ANR malloc & gmalloc
  - track_fs
  - Configuration
- Q&A

membroker

- Service from which apps can cooperatively negotiate memory "quota" system-wide.
- Does not physically manage memory
  - kernel does that.
- Reads from .conf file, which sets limit
  - For Lexmark, the .conf file is generated by a script, based on static tuning and amount of installed RAM.

Membroker is published under LGPL 2.1
https://github.com/lxkiwatkins/membroker.git

membroker – components

- mbserver – long-running service
- Clients communicate via socket – simple protocol
- libmbclient.so – Client library handles communication
  - libmbclient does no memory allocations

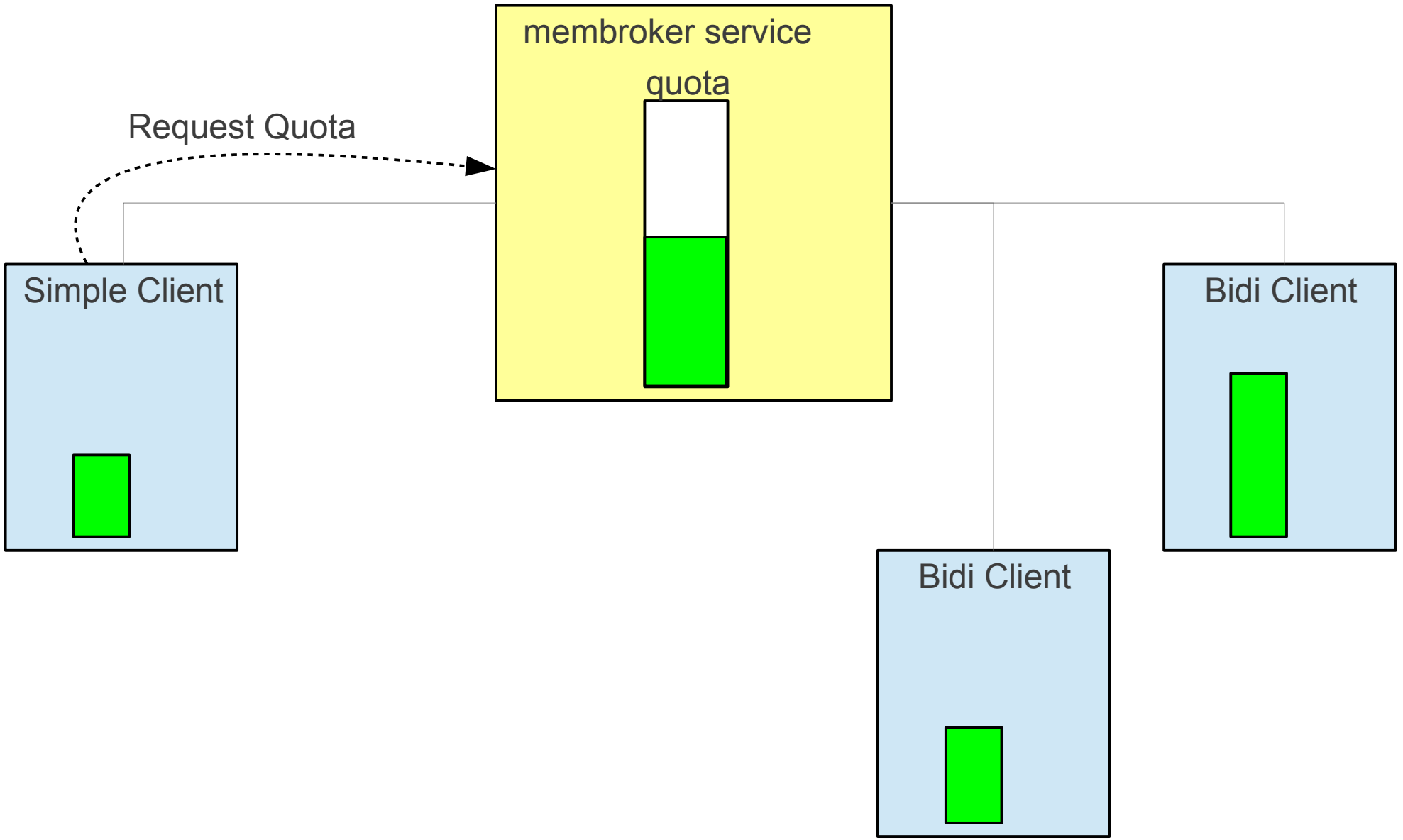
membroker - interactions

- Membroker maintains:
  - Global “available quota”
  - A list of currently connected clients
  - How much quota is owned by each client
- Client asks for quota
  - Membroker decrements available quota and gives to client
  - If quota not available:
    - Ask other clients if they can give back quota
    - Can stall client request until gets enough or gives up and rejects

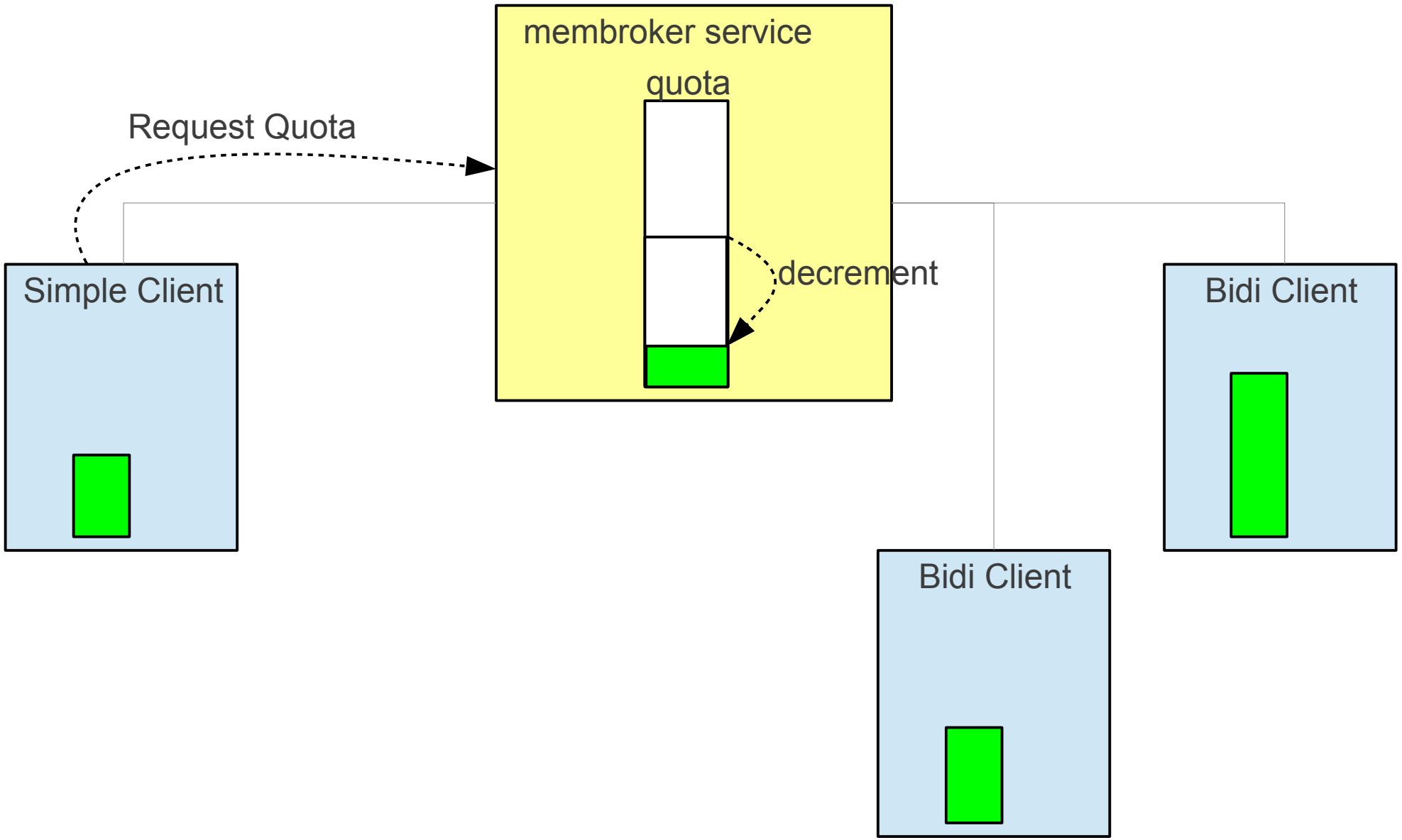
membroker – two types of clients

- Simple client
  - Only client can initiate a membroker transaction
  - Two kinds of transactions:
    - Ask for quota
    - Return quota when no longer needed
- Bidirectional client
  - Can also receive a request from membroker to “give back”
  - Give back request can normal or urgent.
  - Client may free caches, do a garbage collection, wait for some operation to finish, etc.

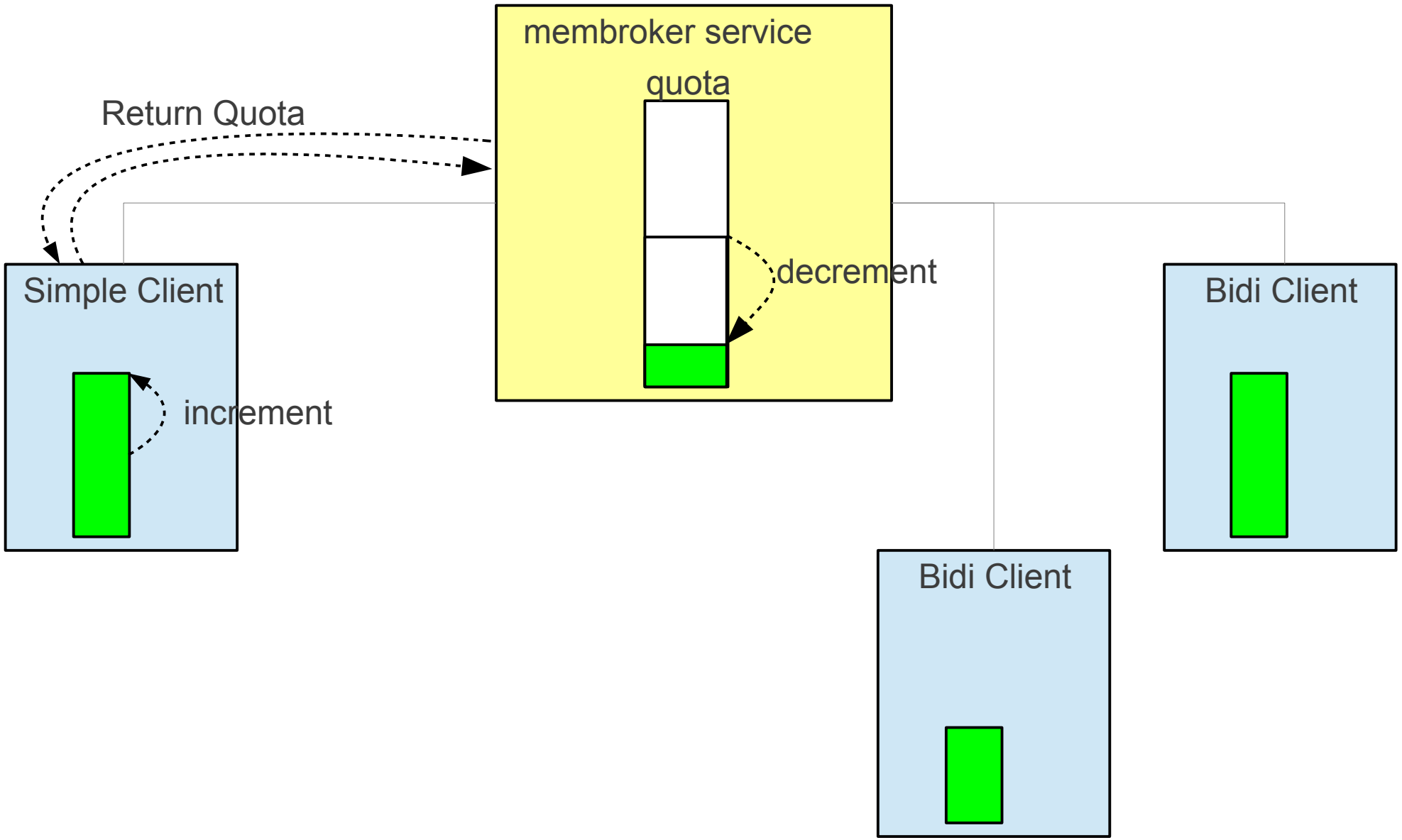
membroker in action



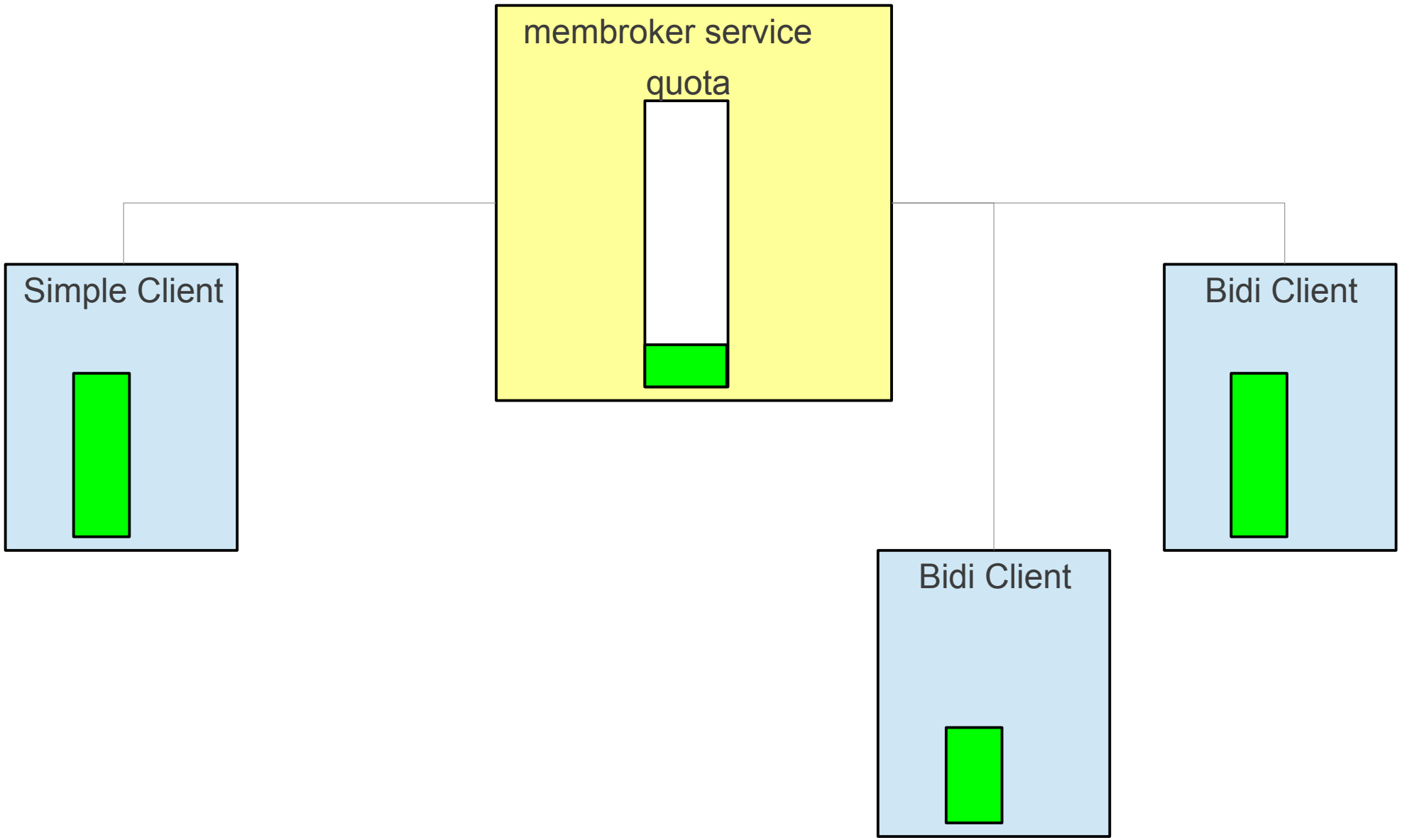
membroker in action



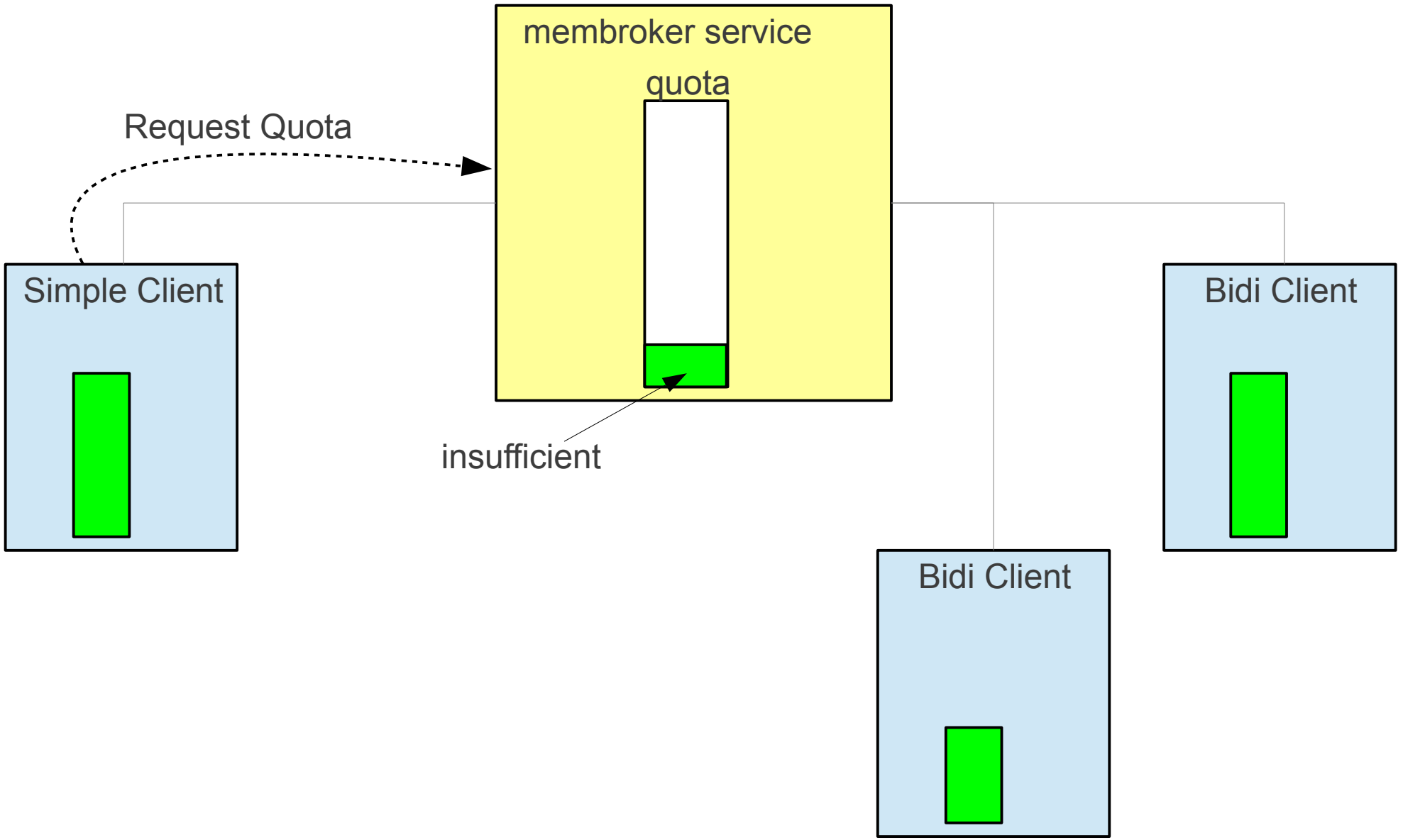
membroker in action



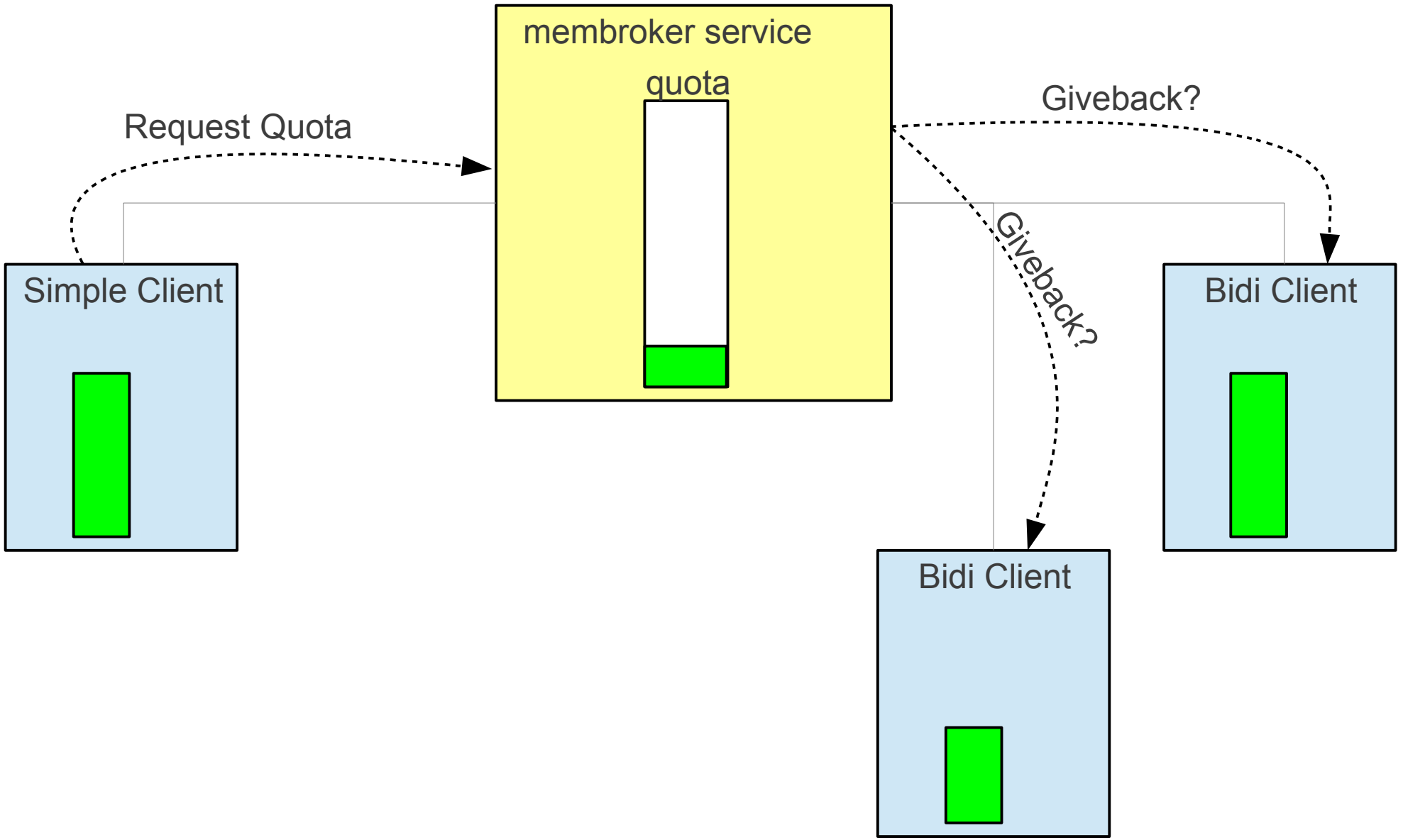
membroker in action



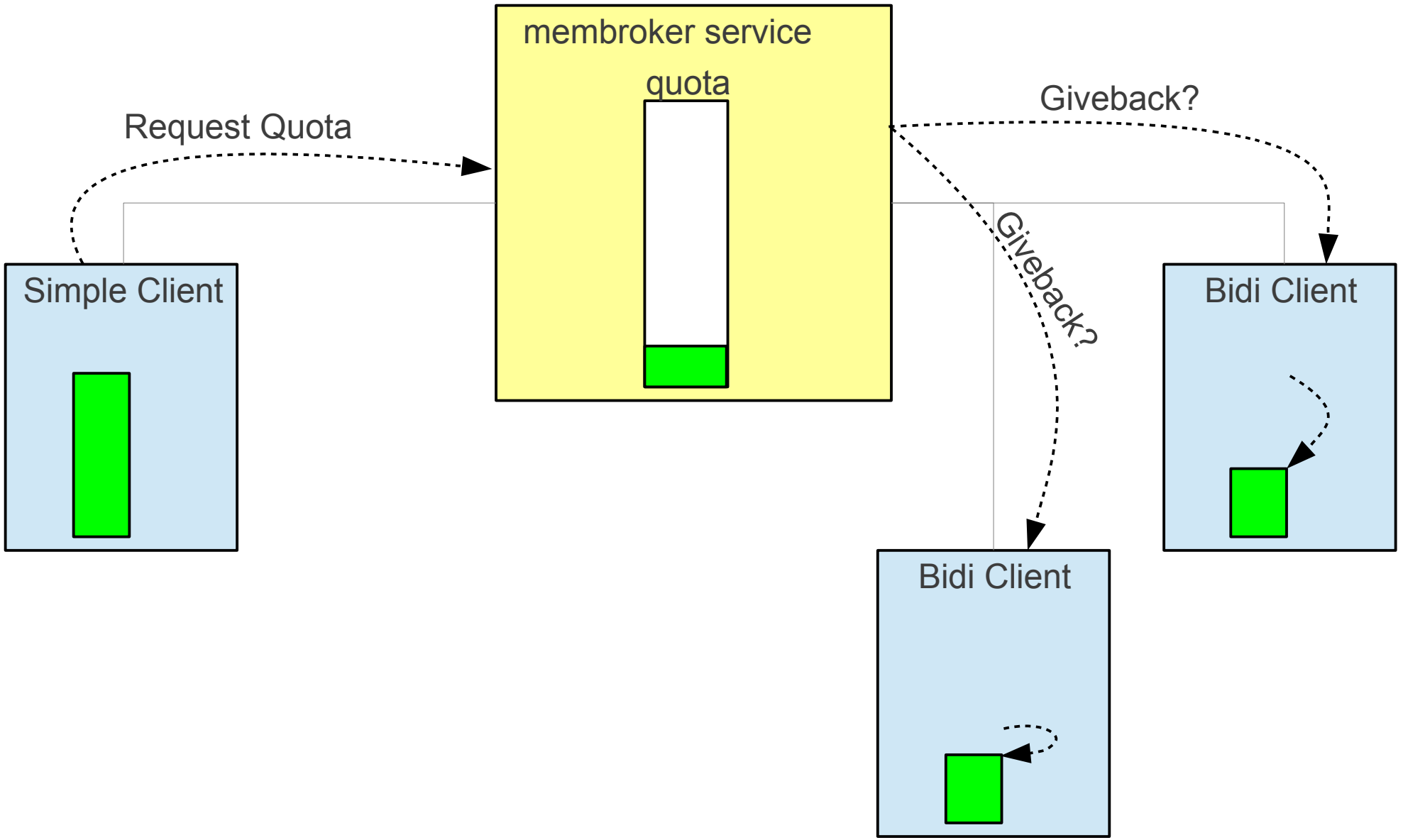
membroker in action



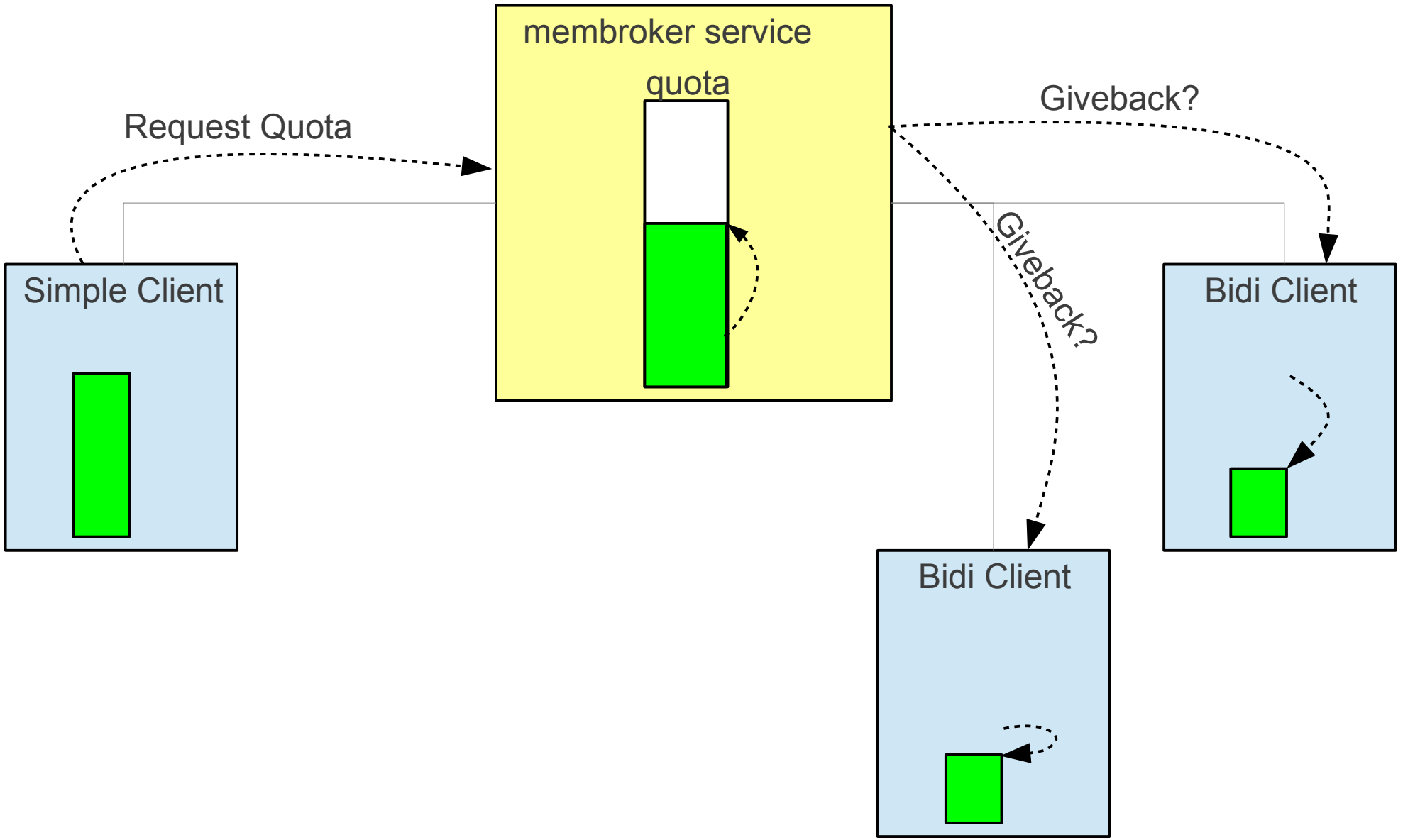
membroker in action



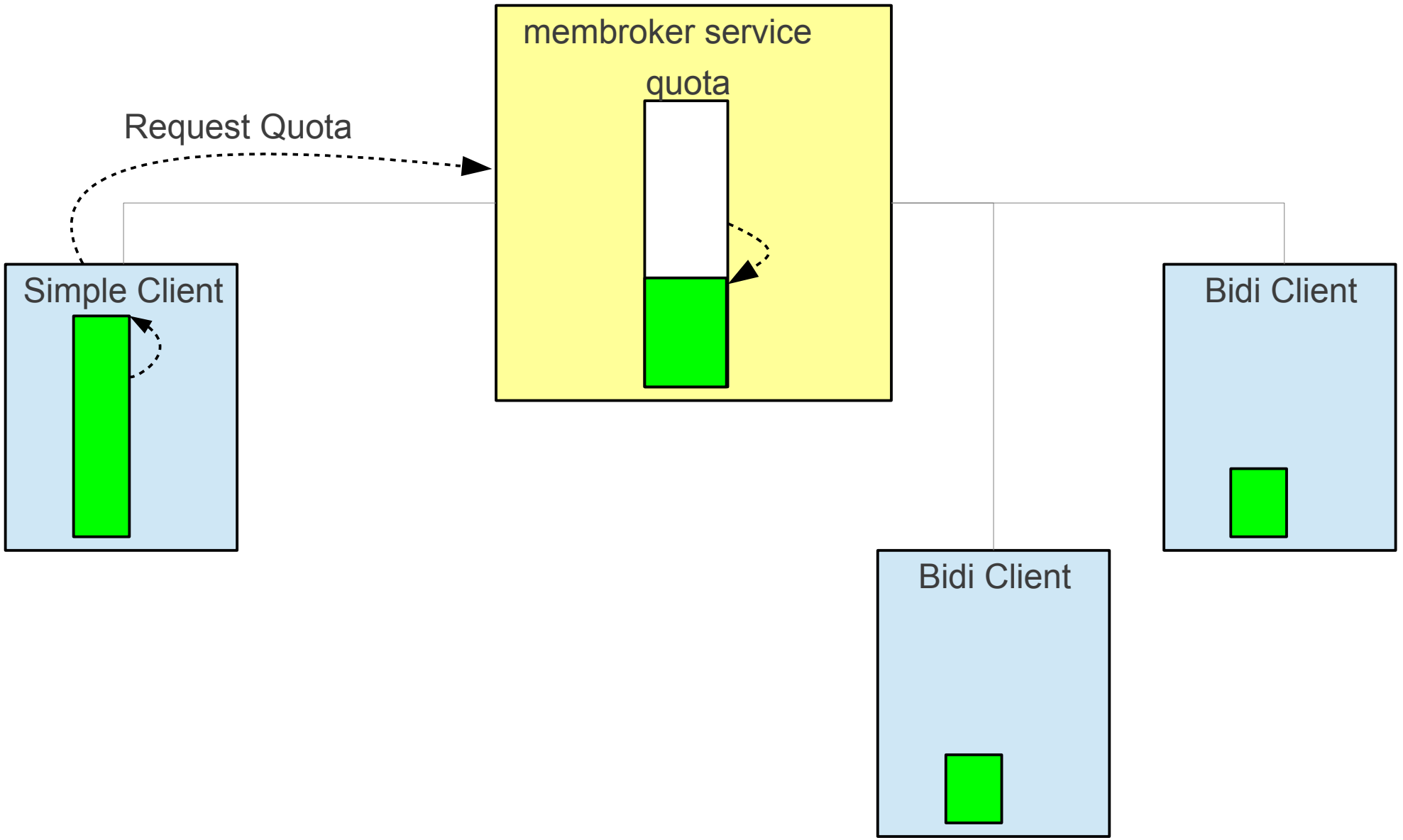
membroker in action



membroker in action



membroker in action



membroker – two levels of urgency

Low urgency (aka “REQUEST”)

- May block while membroker sends Low-urgency giveback request to bidi clients.
- Does not wait indefinitely.
- May return fewer pages than requested.

High urgency (aka “RESERVE”)

- May block indefinitely.
- Will send giveback “RESERVE” message to bidi clients.
- Only fails if all bidi clients refuse giveback.
- Returns either all of requested quota or none of it.

Agenda

- The Problem:
  - How Linux kernel manages memory
  - Memory challenges in Lexmark devices
  - Limitations of glibc malloc for embedded
- Our Solution:
  - membroker service
  - ANR malloc & gmalloc
  - track_fs
  - Configuration
- Q&A

Acknowledgement

membroker and anrmalloc written by:

Ian Watkins

iwatkins@lexmark.com

ANR malloc - A new allocator

Crucial "embedded-friendly" features:

- Return unused pages to system when possible
  - madvise(MADV_DONTNEED)
- Avoid changes to process's memory map
  - i.e. Friendly to real-time threads in same process
- Client can set limit on memory use.

anrmalloc is published under LGPL 2.1
https://github.com/lxkiwatkins/anrmalloc.git

ANR malloc – Why the name “ANR”?

Author, **Ian Watkins**, didn't want to call it “iwmalloc”, so...

The three letters most commonly occurring in the last names of its designers are A, N, and R...

- Ian Watkins
- Scott Arrington
- Steven Walter
- Howard Cochran

ANR malloc - Additional Features

Good space efficiency

- Small allocations use slabs
  - avg 2-3 bits overhead
- Larger allocations use DL-malloc-like algorithm
  - ~1 word overhead
- Slab sizes are controlled by a .conf file

Good speed

- In malloc-heavy real app tests, indistinguishable from glibc.
- In synthetic, pure malloc test, noticeably slower

Optional mark & sweep garbage collection

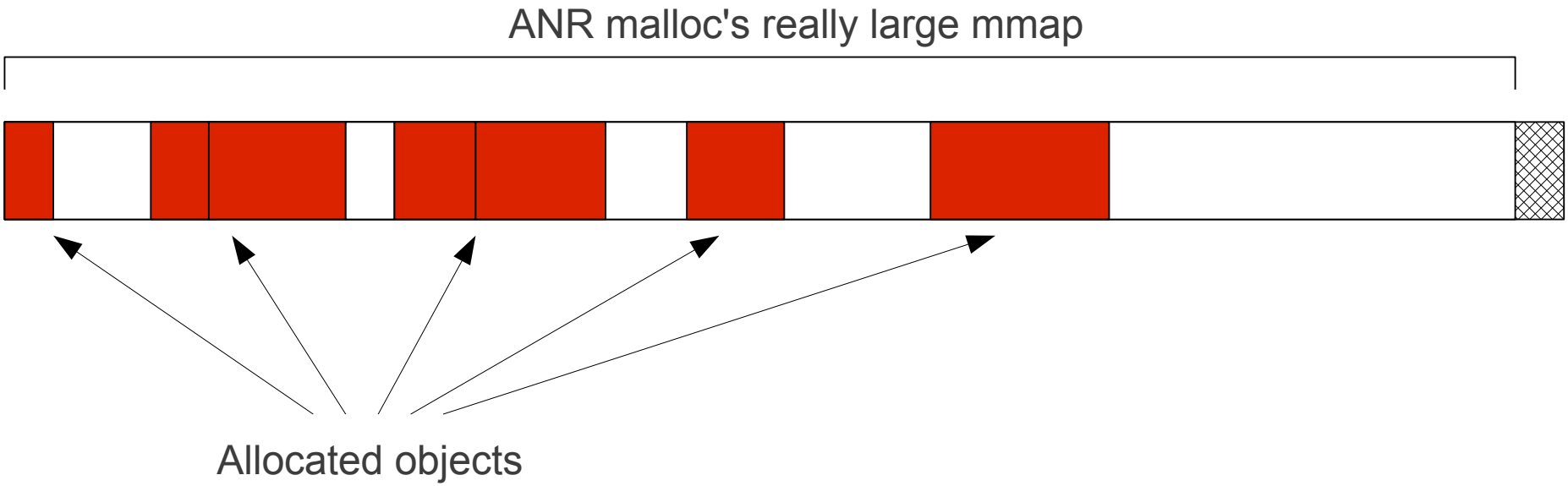
ANR malloc – Debug Features

- All of these are optional and run-time configurable:
- “Flight Data Recorder” - last N client operations
- Guard word at top and bottom of each object to detect overruns
- Store N levels of call stack with each allocation
- Allow client code to stuff additional debug info in each allocated object.
- Debug socket to dump debug data
- Compatible with valgrind on x86
- Fill-with-trash on free

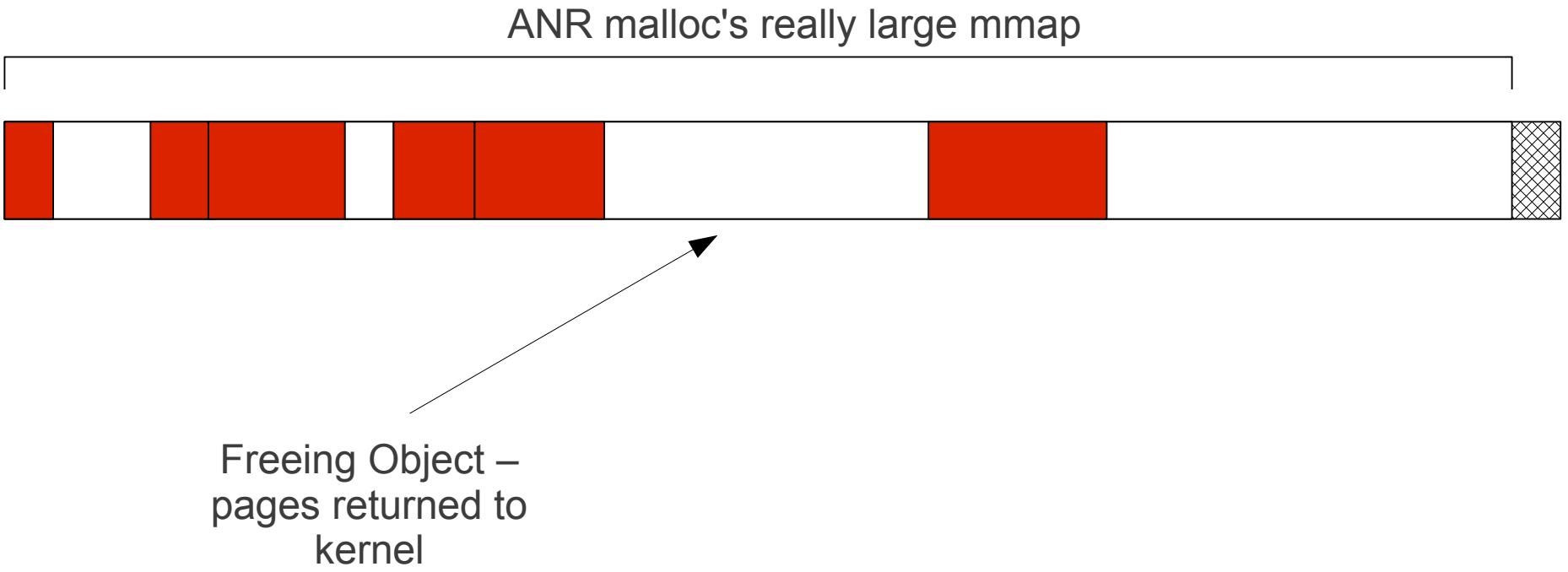
ANR malloc - Returns RAM to linux

- Maintains a bitmap of which pages in its mapping are in use.
- During free:
  - If possible, combine with adjacent free area
  - Mark in bitmap any pages newly made unused.
  - When a threshold of freeable pages reached:
    - Use madvise(MADV_DONTNEED) to return them to the kernel
    - Hysteresis avoids thrashing madvise + page fault
- Volatile Ranges?
  - anrmalloc might become a basis for taking advantage of upcoming Volatile Range kernel support.
  - See http://lwn.net/Articles/518130/

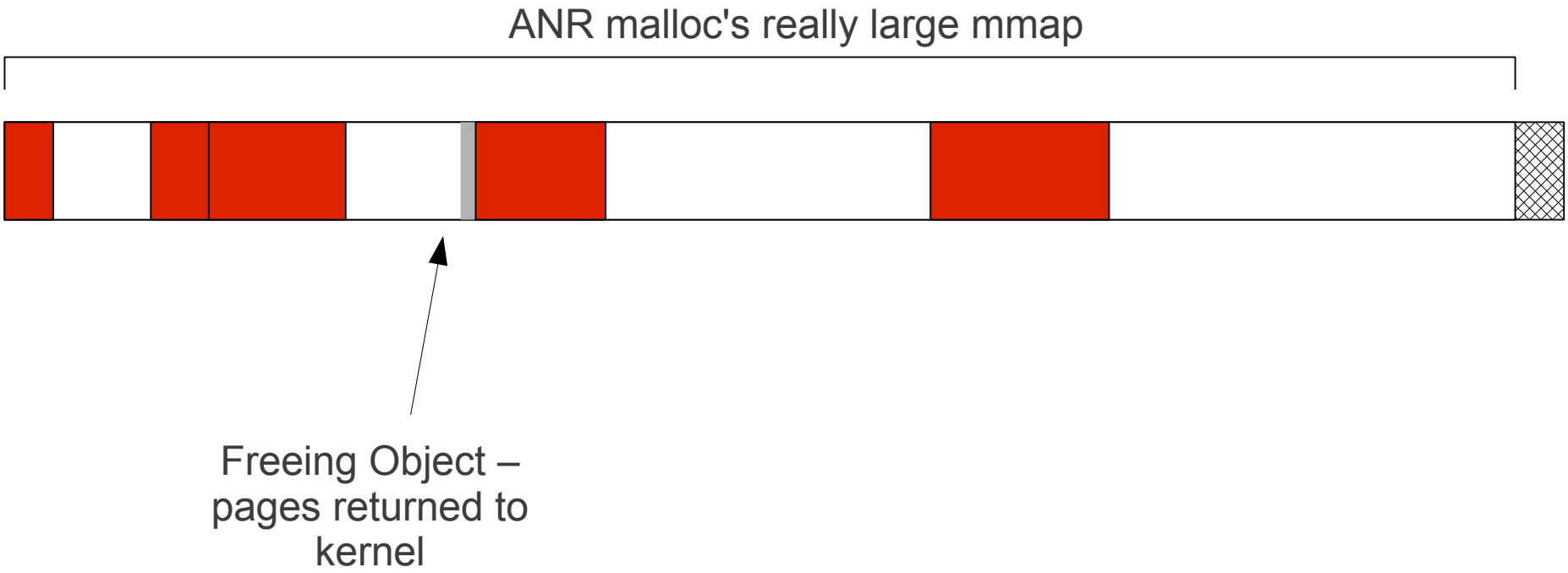
ANR malloc



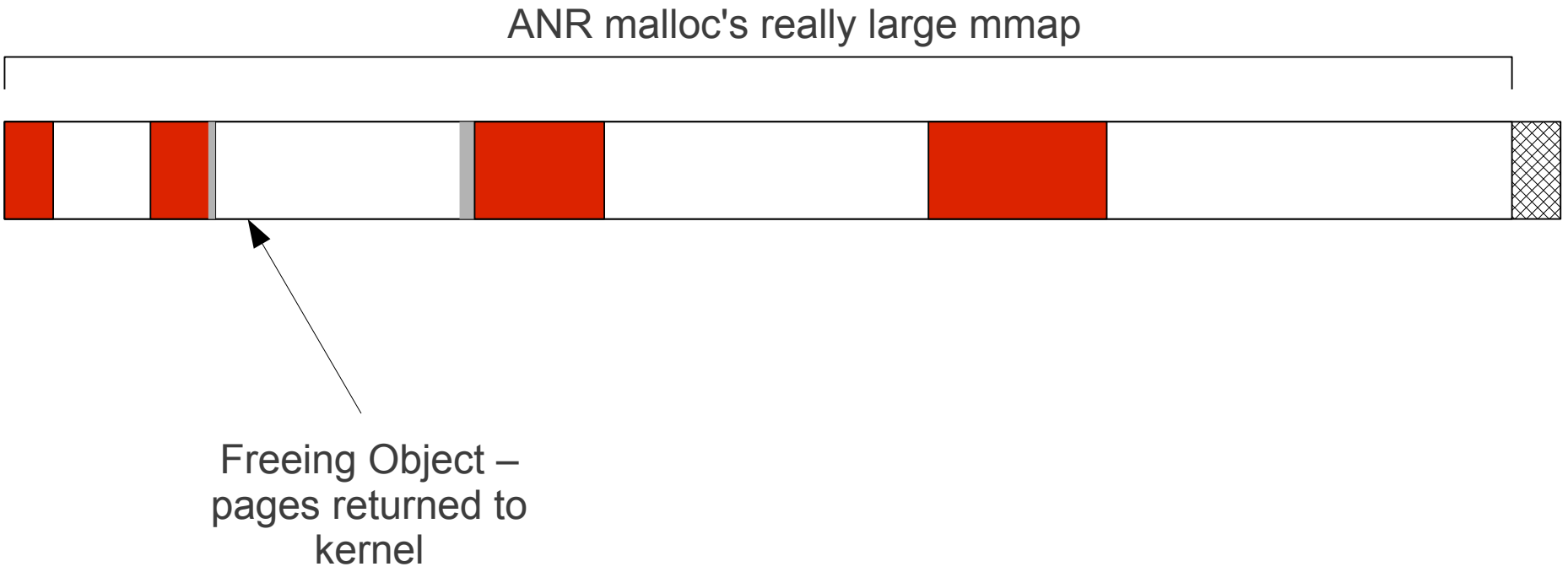
ANR malloc


ANR malloc's really large mmap
Freeing Object –
pages returned to
kernel

ANR malloc


ANR malloc's really large mmap
Freeing Object –
pages returned to
kernel

ANR malloc


ANR malloc's really large mmap
Freeing Object –
pages returned to
kernel

ANR malloc - Antifeature!

Access is serialized via mutex.

- Why? To avoid having to dynamically create "arena" mapping when concurrent allocation occurs
  - Would cause RT misses in other threads
- free() will not block
  - If lock held, actual free is deferred.
- It is still thread-safe.

ANR Malloc - Two APIs

gmalloc

- A transparent replacement for glibc malloc

anr_malloc

- "Full featured" API for clients who need everything.

anr_core

- "Base class" for both of the above.

gmalloc - Replace glibc malloc

- Use dynamic linker to override glibc functions
  - malloc, free, calloc, realloc, posix_memalign
  - Can use LD_PRELOAD to affect unmodified app
  - Reads configuration from .conf file or the environment
- What you get:
  - Gives freed pages back to kernel
  - Never modifies process's memory map behind your back (for Realtime safety)
  - Automatic membroker integration (Optional)
    - Block malloc while ask membroker for quota.
    - Returns unused quota to membroker (w/ hysteresis)

gmalloc

- Configurable fixed max limit
  - Will not exceed, even if membroker could provide quota.
- Can force abort when app exceeds limit.
  - Or just return NULL
- Can enable call-stack debug data per allocation

gmalloc – What you don't get

- Does not allow membroker to ask it to “give back”
  - i.e. not a “bidi” client
- Does not provide garbage collection support
- App cannot make allocations with different level of urgency

anr_malloc interface (1/3)

- Much more control; must customize app
- App must call anr_* functions instead of malloc, free.
- Optional support for mark & sweep Garbage Collection:
  - App initiates G.C. when desired.
  - All subsystems that use anr_malloc register for a “mark your stuff” callback
  - During G.C., anr_malloc calls all registered marking functions.
  - When done, any objects not marked are garbage:
    - Optionally, it can free free them.
    - Optionally generate a “leak report”

anr_malloc interface (2/3)

- Register for “need more memory” callback
  - This callback may do any of these:
    - Free cached objects.
    - Initiate an ANR garbage collection.
    - Ask membroker for quota
  - OK to stall indefinitely.

anr_malloc interface (3/3)

- Variations of malloc:
  - anr_malloc() – normal.
    - May call “need more memory” callback.
    - May block indefinitely.
    - If fails, returns NULL.
  - anr_malloc_if_available() –
    - Succeeds if quota readily available, or fails fast.
    - Not allowed to call “need more memory” callback.

Agenda

- The Problem:
  - How Linux kernel manages memory
  - Memory challenges in Lexmark devices
  - Limitations of glibc malloc for embedded
- Our Solution:
  - membroker service
  - ANR malloc & gmalloc
  - track_fs
  - Configuration
- Q&A

track_fs – membroker aware tmpfs

- FUSE filesystem
- Layers on top of tmpfs
- During write, compares quota with what devstat says
- If needed, requests more quota from membroker
- Returns ENOSPC if membroker cannot give quota
- Very simple – file ops serialized via mutex
- During unlink, gives quota back to membroker
- Modularized – could membroker interaction is in separate shared object easily replaced.
- Does not support mmap.

track_fs is published under GPLv2
https://github.com/...

Agenda

- The Problem:
  - How Linux kernel manages memory
  - Memory challenges in Lexmark devices
  - Limitations of glibc malloc for embedded
- Our Solution:
  - membroker service
  - ANR malloc & gmalloc
  - track_fs
  - Configuration
- Q&A

Typical Configuration

- Make mapping for anr_malloc or gmalloc be twice as large as the most quota that you expect to have.
  - Internal limit (quota) can change, but can keep mapping the same.
  - Reduces fragmentation.
  - Extra large mapping is almost free due to madvise().
  - .conf file or app controls mapping size.

Typical Configuration – membroker

- Set membroker's quota to most of RAM
  - Exclude page cache working set, kernel pages, DMA buffers, thread stacks.
  - Tune this limit by trial and error
    - Stress test device
    - Watch /proc/meminfo, /proc/vmstat – pgmajfault for signs of heavy memory pressure
    - Slowly give membroker more quota until see pressure, then back off a bit.

Thank You!

- Acknowledgements:
  - **Ian Watkins**, iwatkins@lexmark.com
    - Author, anrmalloc & membroker
  - **Scott Arrington, Steven Walter** – design
  - **Randy Witt** – author, track_fs.
- Contact:
  - Howard Cochran cochran@lexmark.com
- Source Code:
  - https://github.com/lxkiwatkins/membroker.git
  - https://github.com/lxkiwatkins/anrmalloc.git
  - https://github.com/zedian/track_fs

Bonus Slides...

Future Work

- membroker:
  - Different classes of clients with independent quotas
  - Utilize low-memory notification from kernel
- anr_core
  - Support 64-bit architecture
  - Optional “arena” concurrency for apps with no RT code
- Other
  - Provide library to add membroker integration for anrmalloc client.
- track_fs
  - Add concurrency
  - Support mmap
  - Optional “non-urgent”-only membroker integration.

Alternative – RT safety only

- glibc malloc doesn't call brk() directly, but a __morecore() function that your app can override.
  - App can make a large brk() call at startup, and never grow it.
- Avoid arena creation
  - Use dynamic linker to override all glibc malloc functions
  - Simpler wrapper
  - Lock a mutex, then call the real glibc function, then unlock.
  - Use mallopt(M_MMAP_MAX, 0) to prevent separate mmap for large allocations.

Alternative – Tracking without gmalloc

- What if?:
  - You want to use glibc malloc, not a custom allocator.
  - You only need to roughly track memory usage
  - You don't care about returning unused pages to linux
- Override glibc's __morecore
  - Normally __morecore is sbrk()
  - Your __morecore could:
    - Request or return quota from membroker
    - Call sbrk()
- Use techniques on previous slide to prevent mmap or arena creation in glibc malloc.

Handling OOM Crash

- Out of Memory failures are hard to debug
  - For our devices, any actual OOM task kill is a crash.
    - We must reboot
  - Generally, shell is unresponsive.
  - Cannot fork() tools to read information from /proc, etc.

Handling OOM – Kernel Hack Method

- Hack out_of_memory():
  - Make out_of_memory() simply return unless it has been called many times in short period (i.e. “try harder).
  - Make it panic instead of kill a process
  - Make panic path to dump some info to flash storage:
    - /proc/meminfo
    - For all /proc/pid/smaps, dump sums of writable private dirty (i.e. unevectable) pages
    - Dump maps for the “largest” process in the system
- Disadvantages:
  - Hacky, maintenance burden
  - limited to only what you chose ahead of time to collect

Handling OOM – Sacrificial Lamb Method

- Force kernel to select a specific process to kill
  - Use /proc/pid/oom* nodes to accomplish this
- Parent is debug monitor – receives SIGCHLD
  - Parent can use freezer control groups to halt most activity
  - Let parent be SCHED_FIFO
  - Parent may collect more debug data
  - May allow debug shell to operate for leisurely debug

Memory Control Groups, oom_notify

- Put most processes in a large memory control group
- Have debug processes live in a smaller control group
- A debug process can listen for oom_notify event
  - Kernel will suspend allocations from main group for you
  - Debug process can leisurely collect debug data
  - Might could even implement a non-fatal recovery strategy
- We have not tried memory controllers yet:
  - The code appears fairly heavy-weight
  - It broke the “lumpy reclaim” algorithm on ARM
    - Lumpy reclaim was crucial for us
    - No longer an issue since Mel Gorman replaced lumpy reclaim with page COMPACTION.

Recap: Problems with glibc malloc

- Uses sbrk() to increase size of heap mapping
- The only way it can return heap to kernel is by shrinking the heap mapping via sbrk().
  - Even one extant allocation near end of mapping prevents shrinking.
  - Cannot return unused pages in middle of heap
- Achieves thread concurrency by creating a new mapping for second thread (arena).
  - These are sticky; may never unmap these
- Not Realtime-safe: Use of mmap, sbrk will break any RT threads within the process!