An overview of the LFH

Bruno Pujos

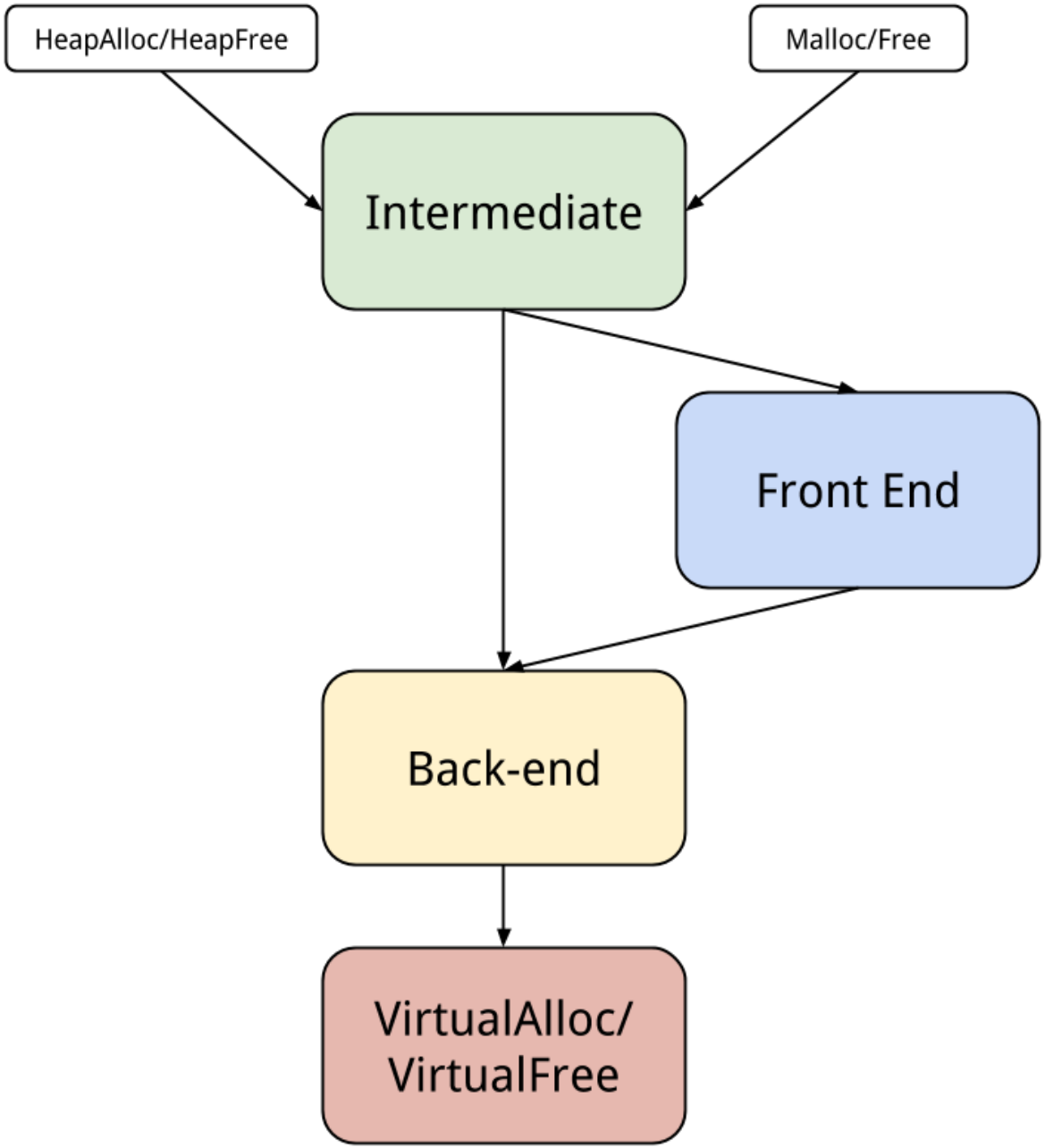
July 20, 2014

Plan

The LFH?

- Low Fragmentation Heap: Front End allocator
- Userland (sorry, no kernel this time...)
- Windows 8/8.1 32bit
- Why talk about it?
- Some details were left out to keep it simple

General Memory Management



A little bit of history

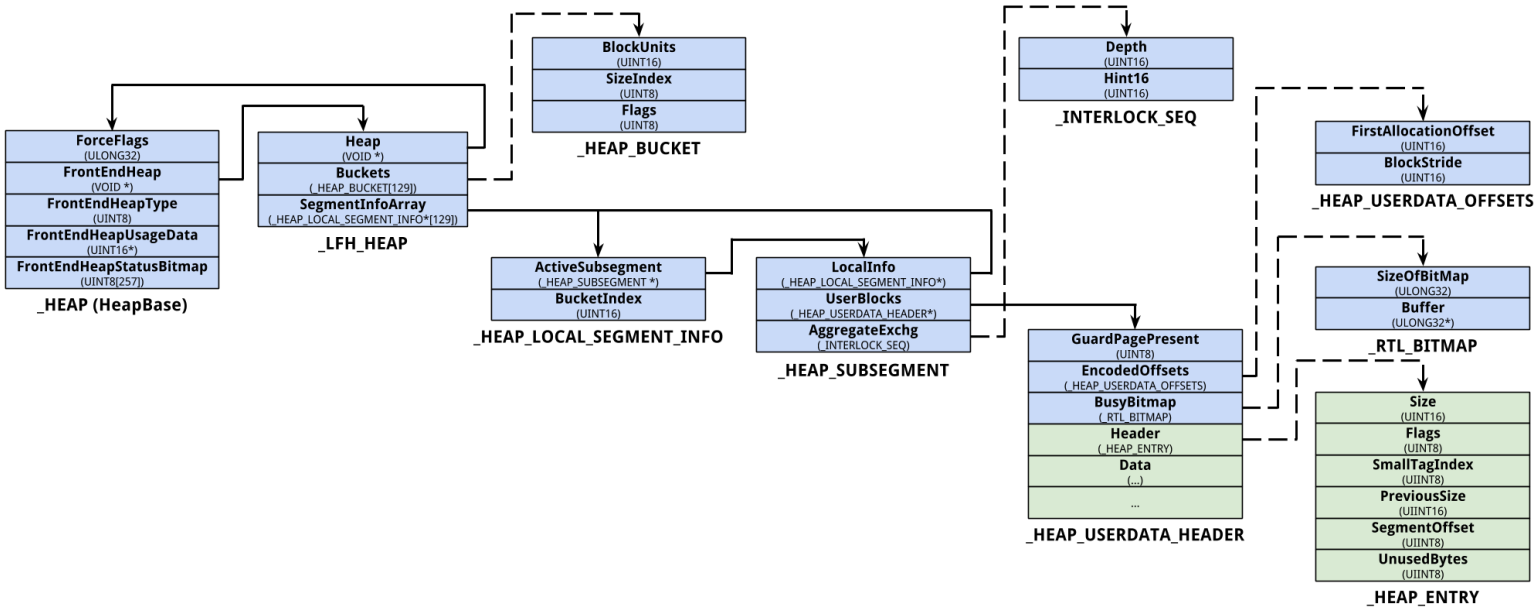
- LFH released with Windows XP (2001) but not enabled by default
- The Look-Aside-List was another Front End allocator at that time
- Since Vista, no more LAL, and LFH is enabled by default

Plan

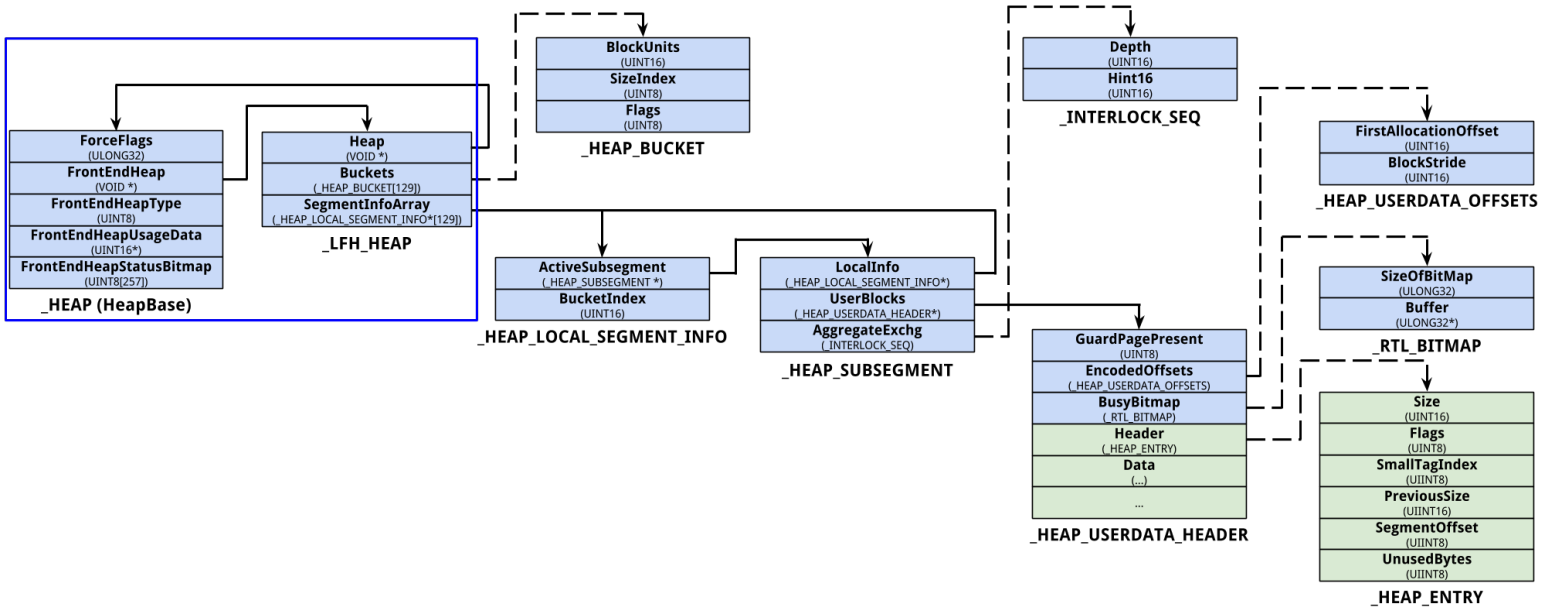
Plan

Plan

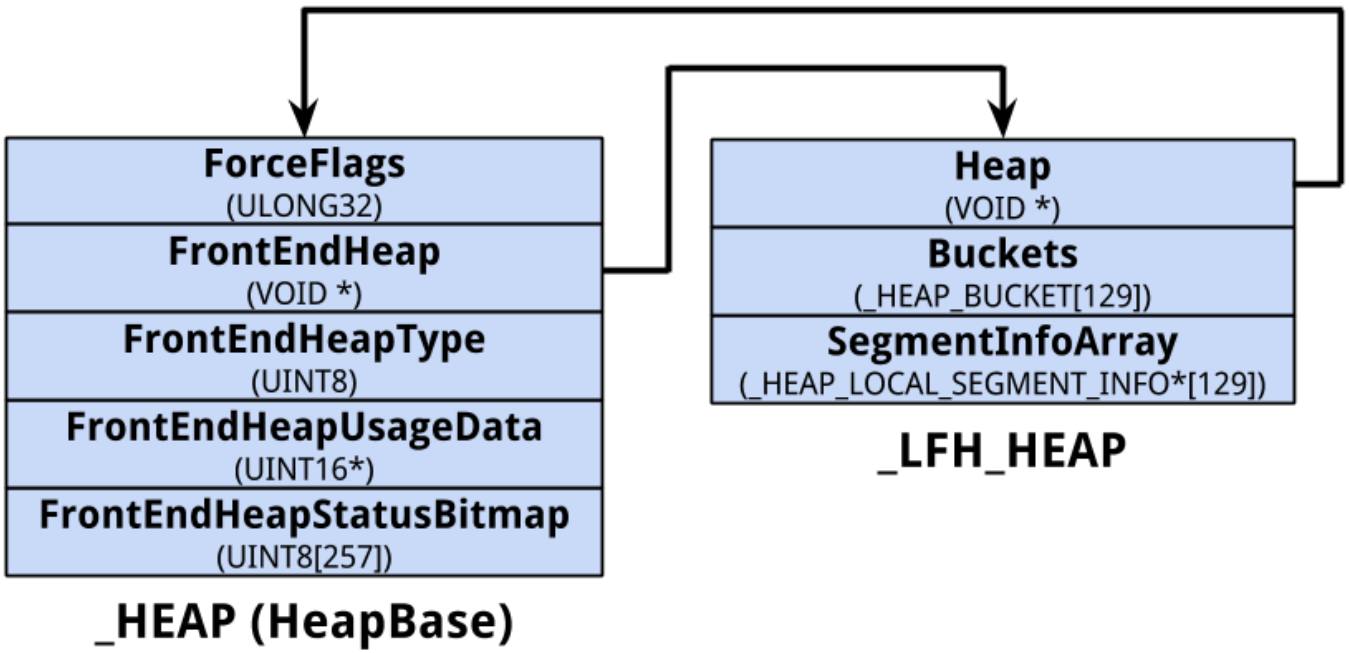
General Overview



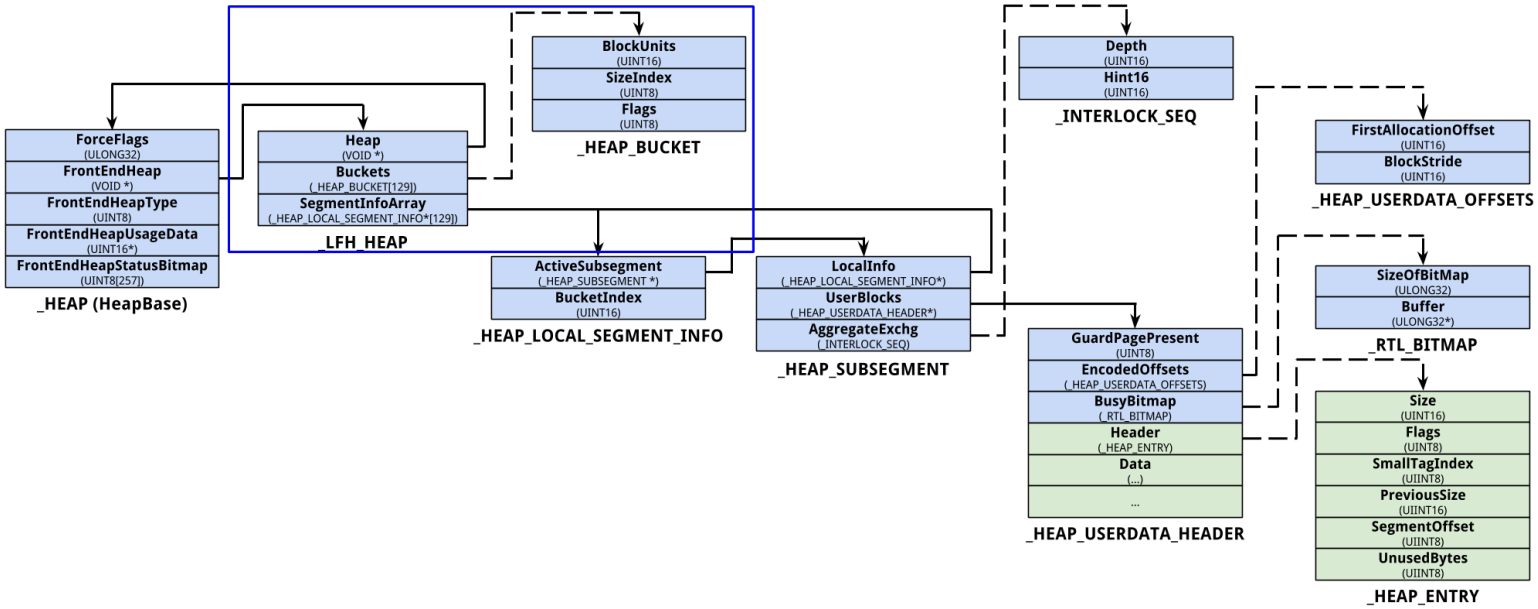
General Overview



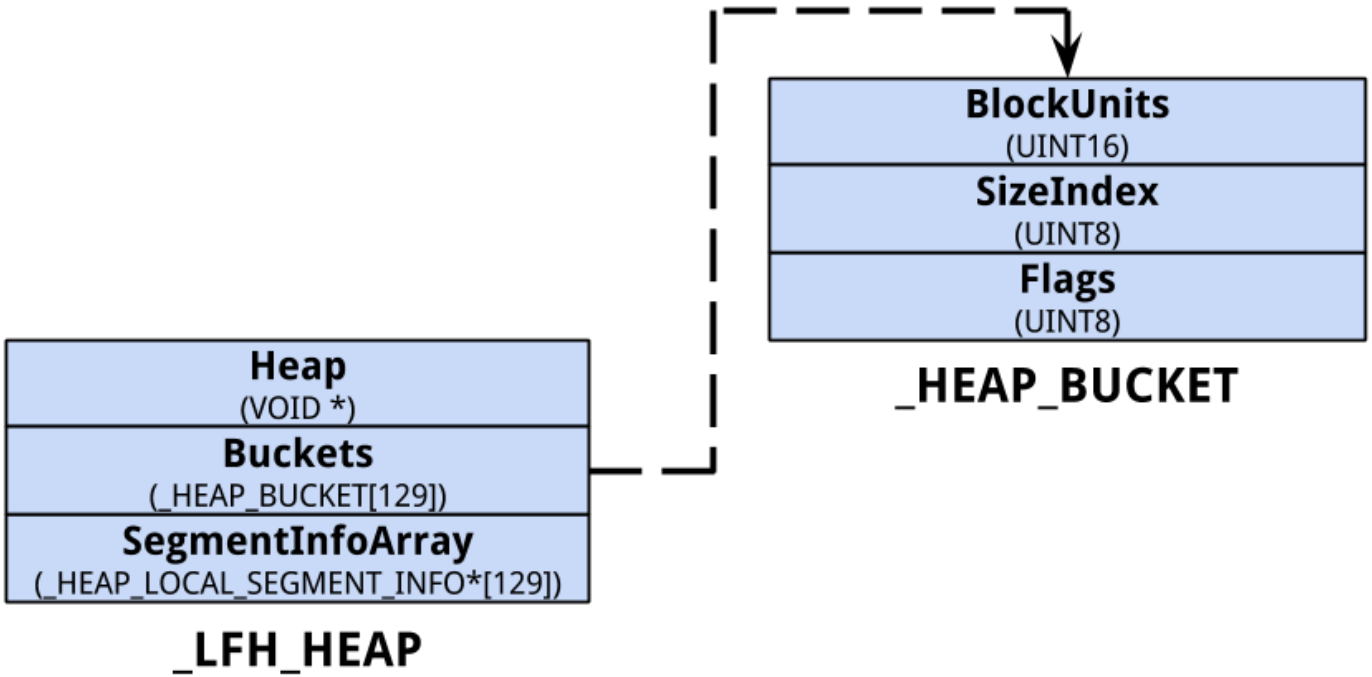
_HEAP & _LFH_HEAP



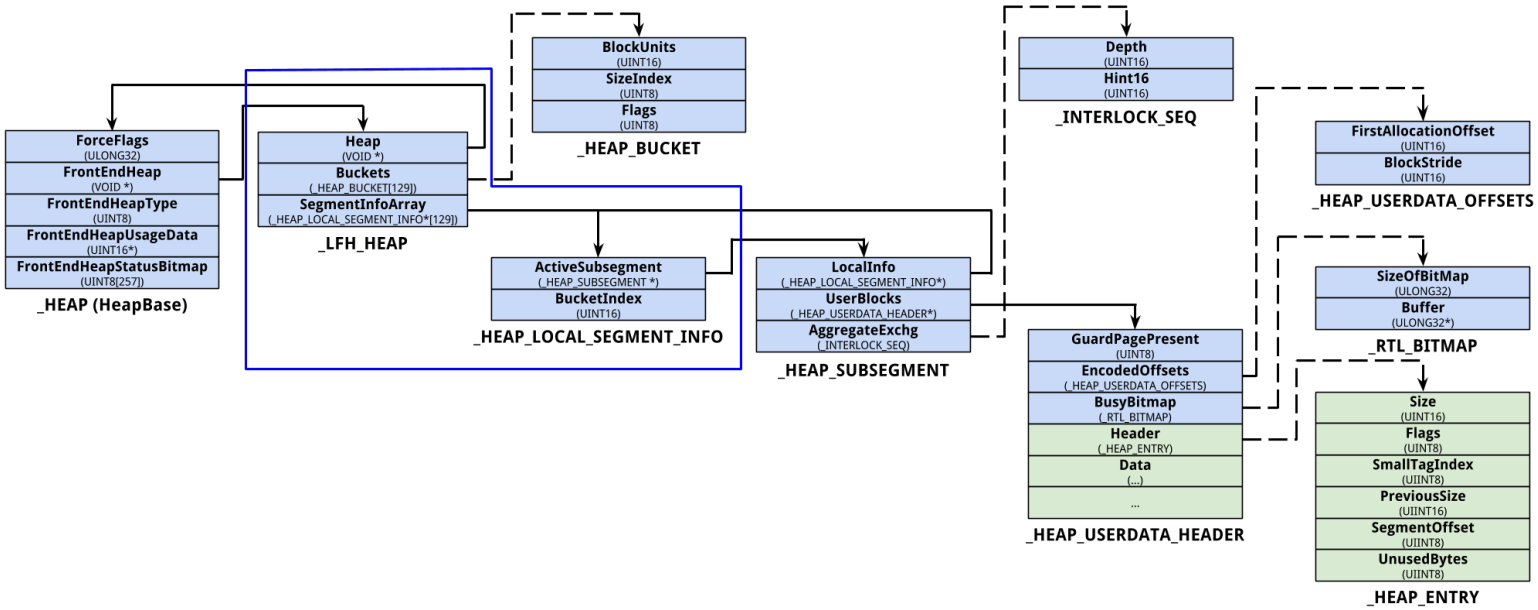
General Overview



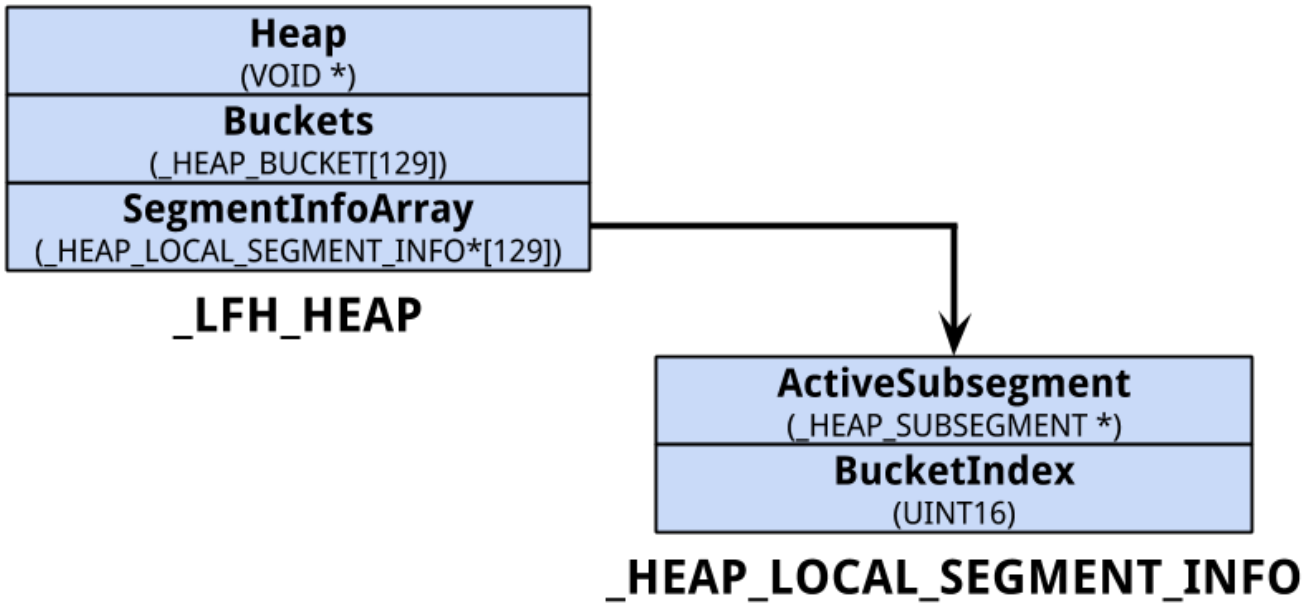
_HEAP_BUCKET


BlockUnits
(UINT16)
SizeIndex
(UINT8)
Flags
(UINT8)
_HEAP_BUCKET
Heap
(VOID *)
Buckets
(_HEAP_BUCKET[129])
SegmentInfoArray
(_HEAP_LOCAL_SEGMENT_INFO*[129])
_LFH_HEAP

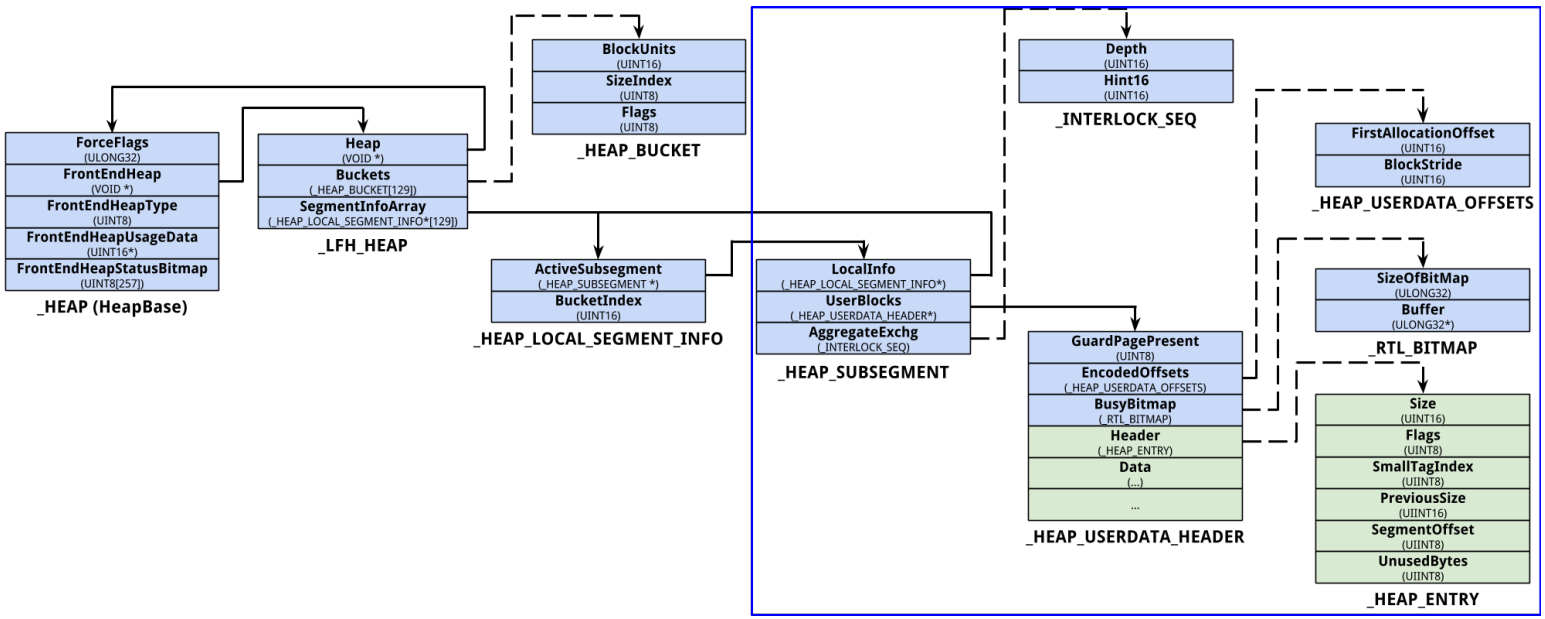
General Overview



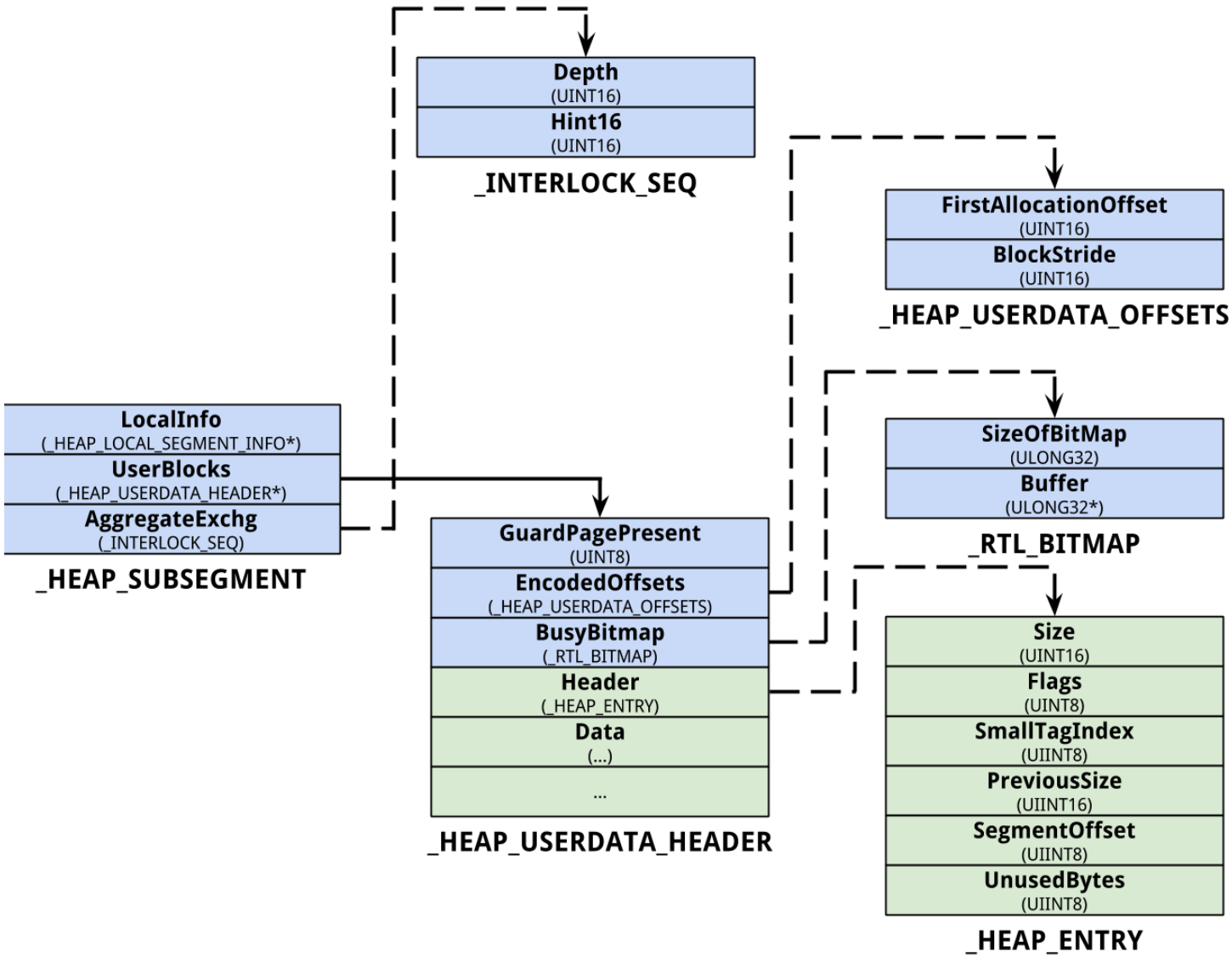
_HEAP_LOCAL_SEGMENT_INFO



General Overview

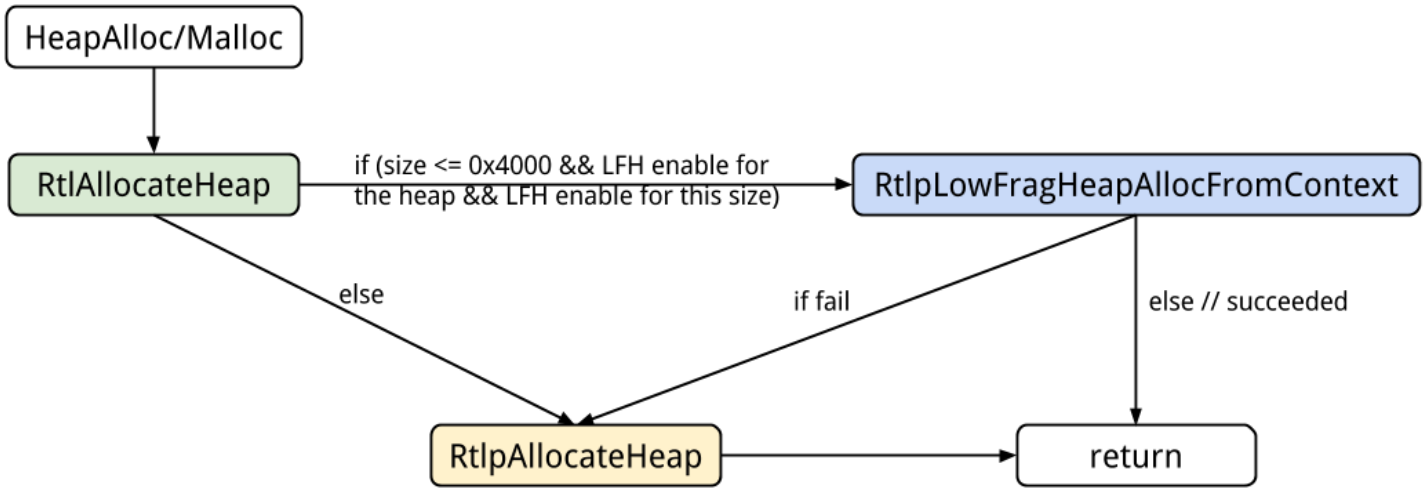


Subsegment & UserBlocks



Plan

Allocation Workflow



LFH activation & Back-end

- RtlpAllocateHeap(_HEAP *Heap, int Flags, int Size, unsigned int RoundedSize, _LIST_ENTRY *ListHint, int *RetCode)
- HEAP_NO_SERIALIZE
- Heap->CompatibilityFlags & 0x20000000: activation of the LFH needed
- RtlpPerformHeapMaintenance(_HEAP *Heap)

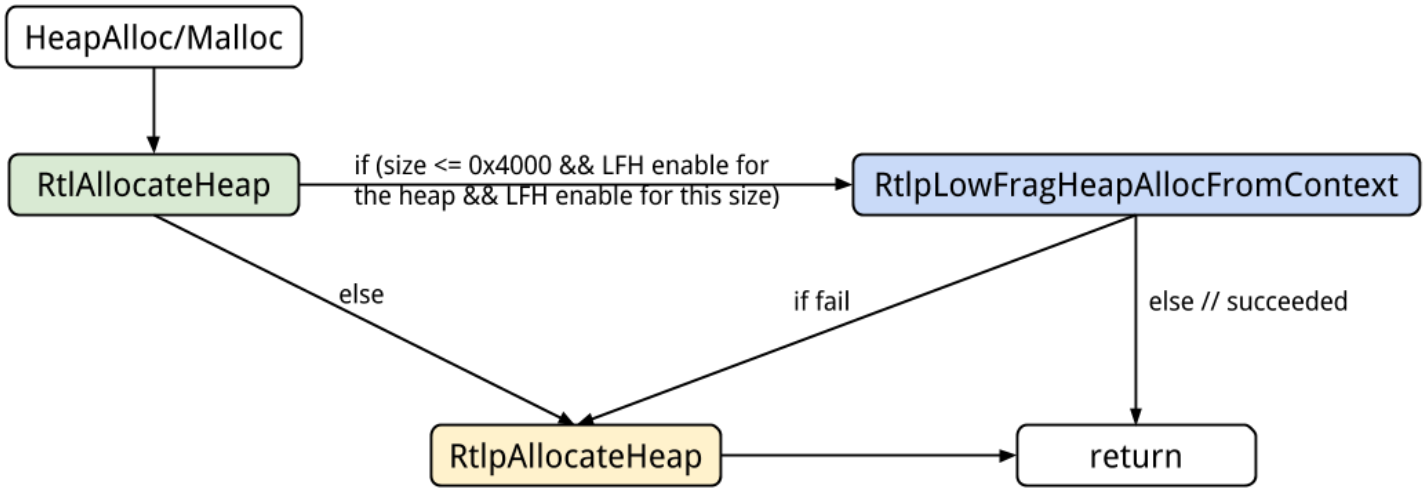
Allocation of size < 0x4000

- if the LFH is not activated: set the CompatibilityFlags
- if the LFH is not activated for this size:
  - add 0x21 in the Heap->FrontEndHeapUsageData[]
  - if 0x10 consecutive allocations or Heap->FrontEndHeapUsageData[] > 0xff00: activate for the next allocation of the same size

Activation for a given size

- set Heap->FrontEndHeapUsageData[] to the BucketIndex
- set Heap->FrontEndHeapStatusBitmap[] to 1 (activated)

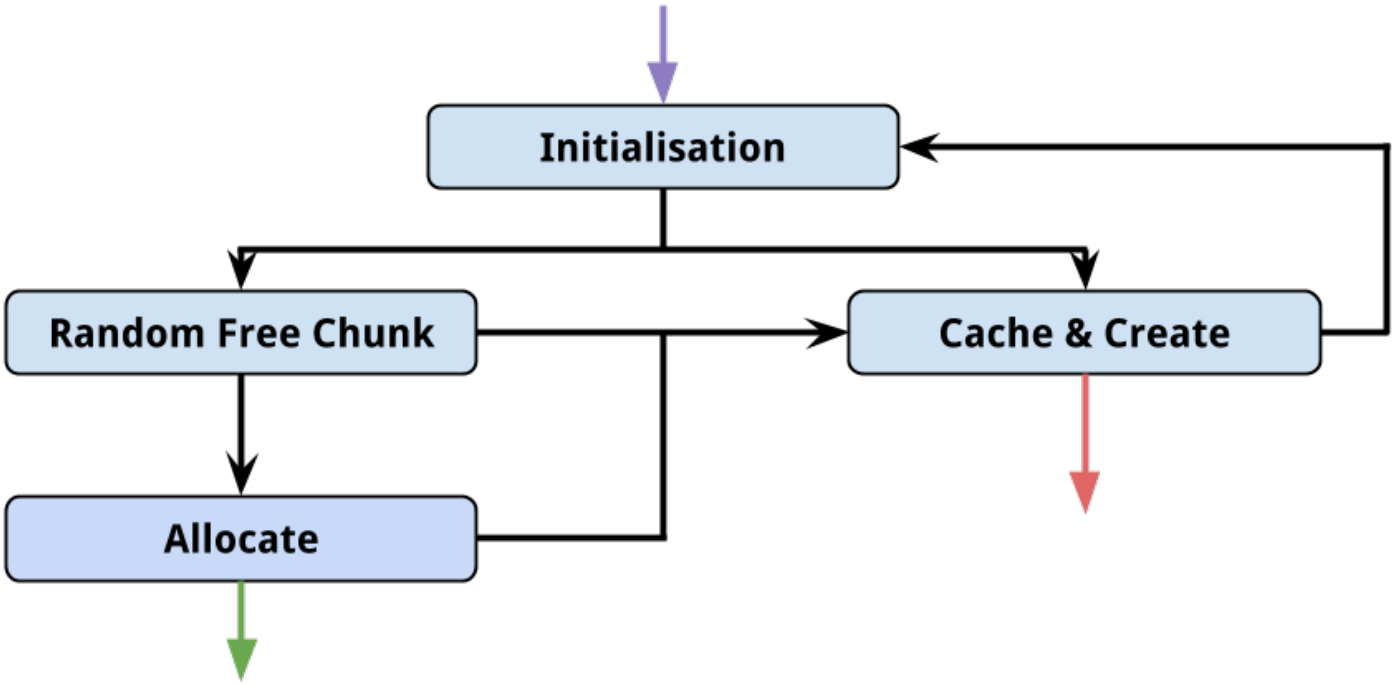
Allocation Workflow



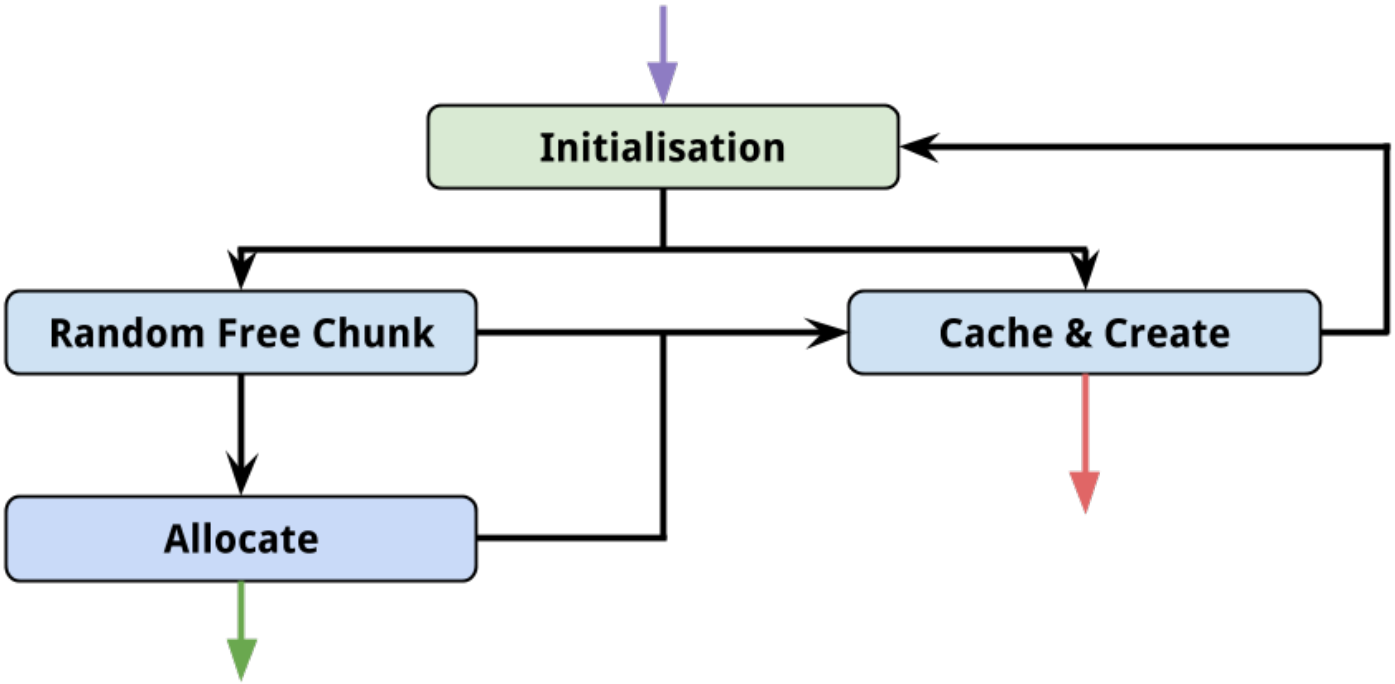
LFH Allocation

- size <= 0x4000
- HEAP_NO_SERIALIZE
- Heap->FrontEndHeapStatusBitmap == 1
- RtlpLowFragHeapAllocFromContext(_LFH_HEAP *LFH, unsigned short BucketIndex, int Size, char Flags)

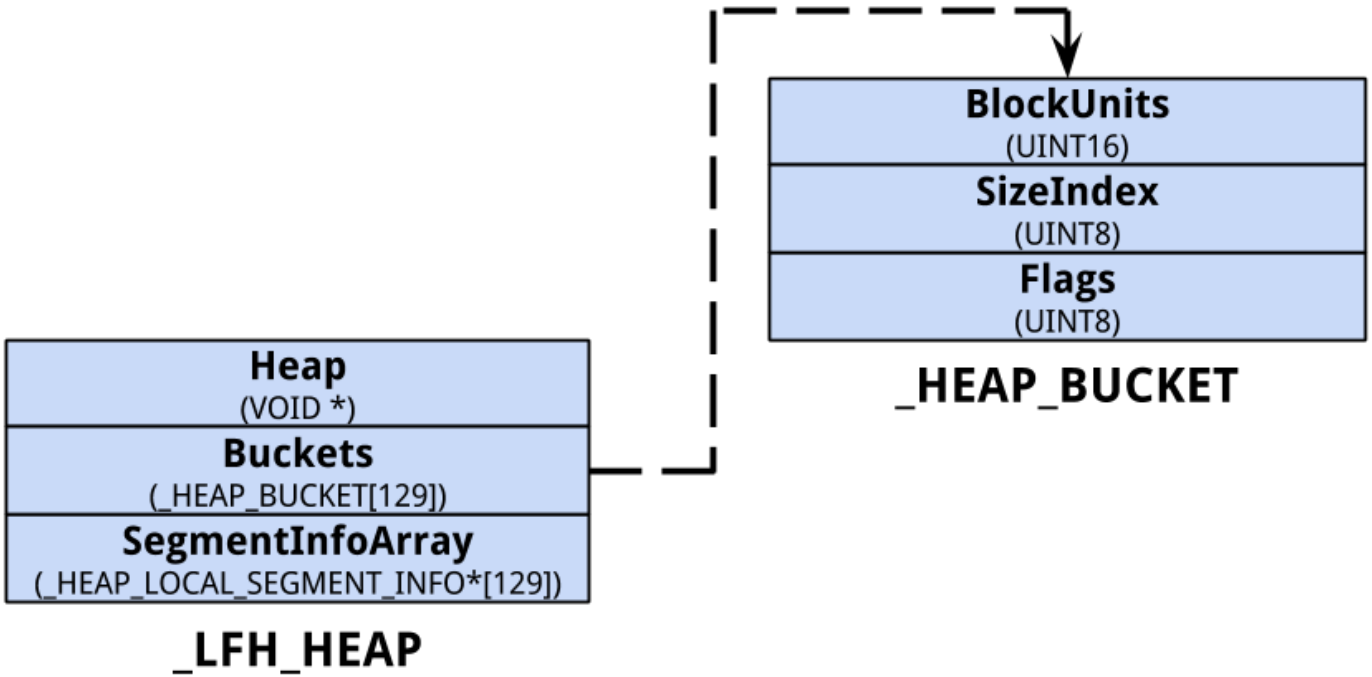
LFH Allocation Workflow



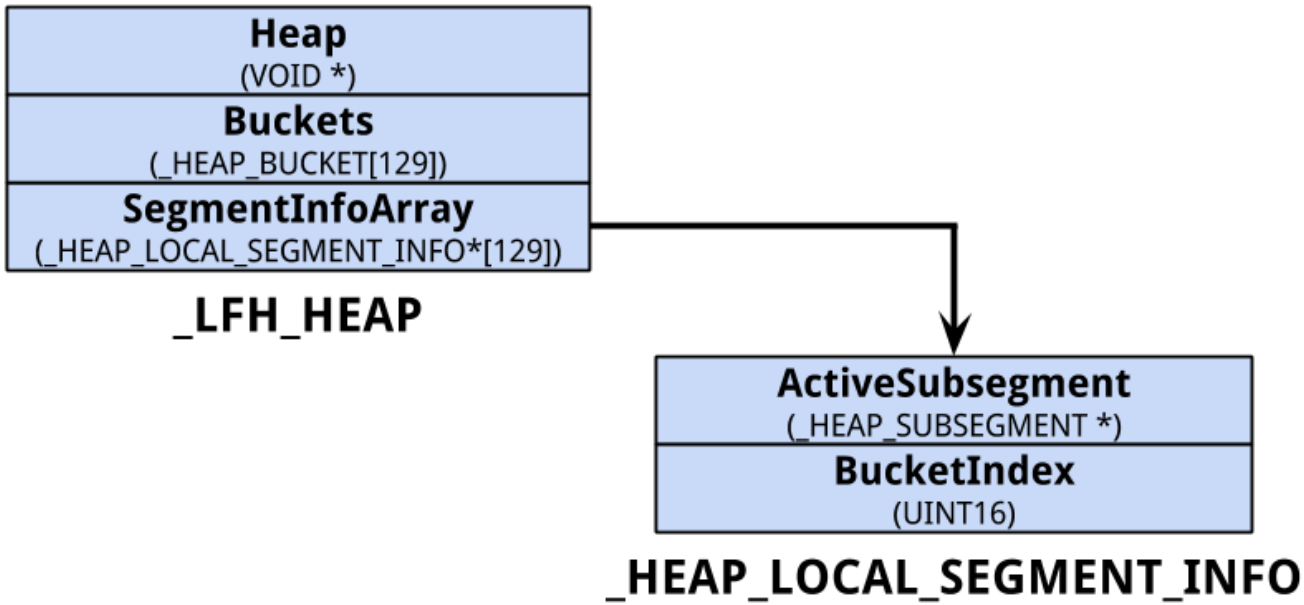
LFH Initialisation



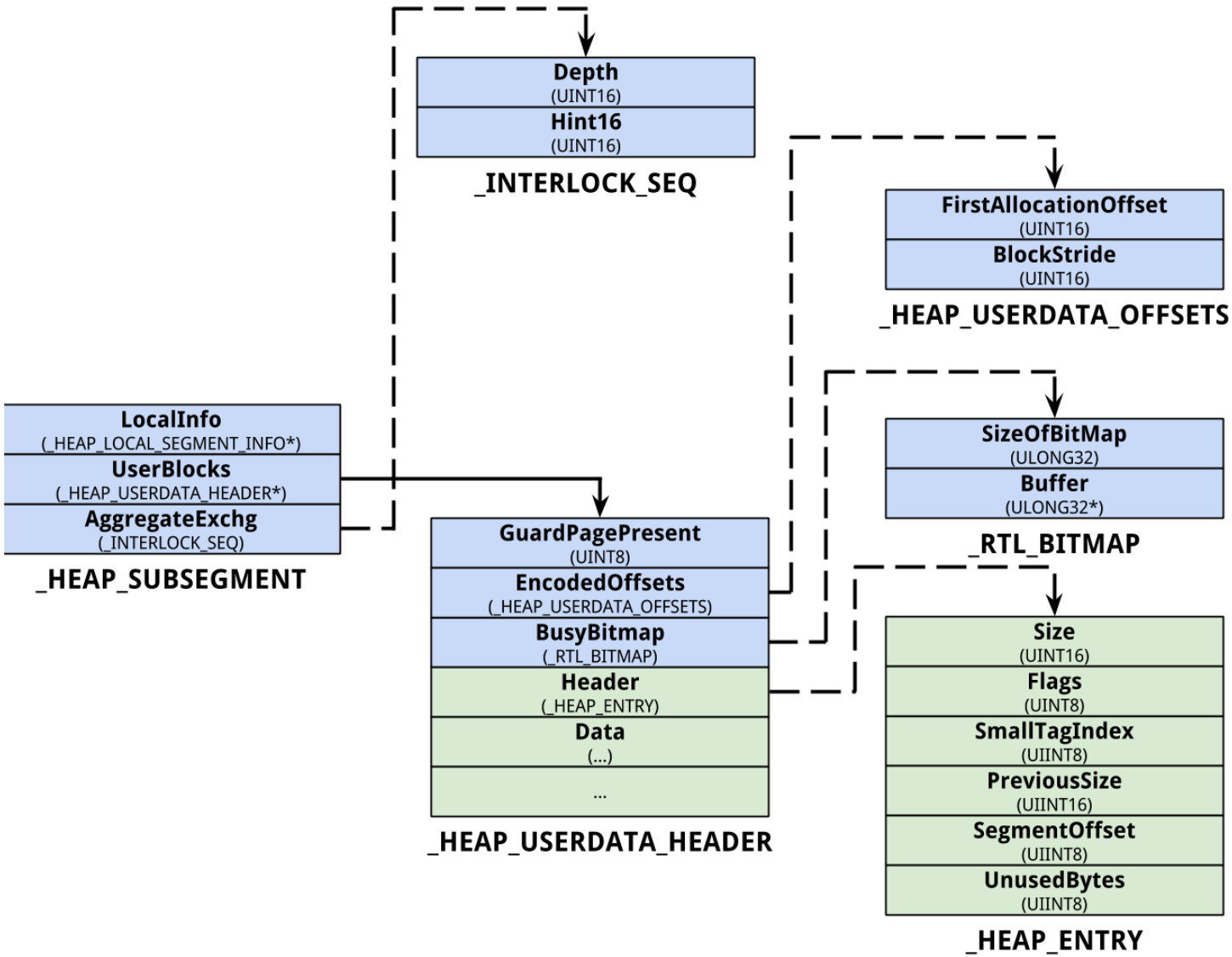
LFH Initialisation



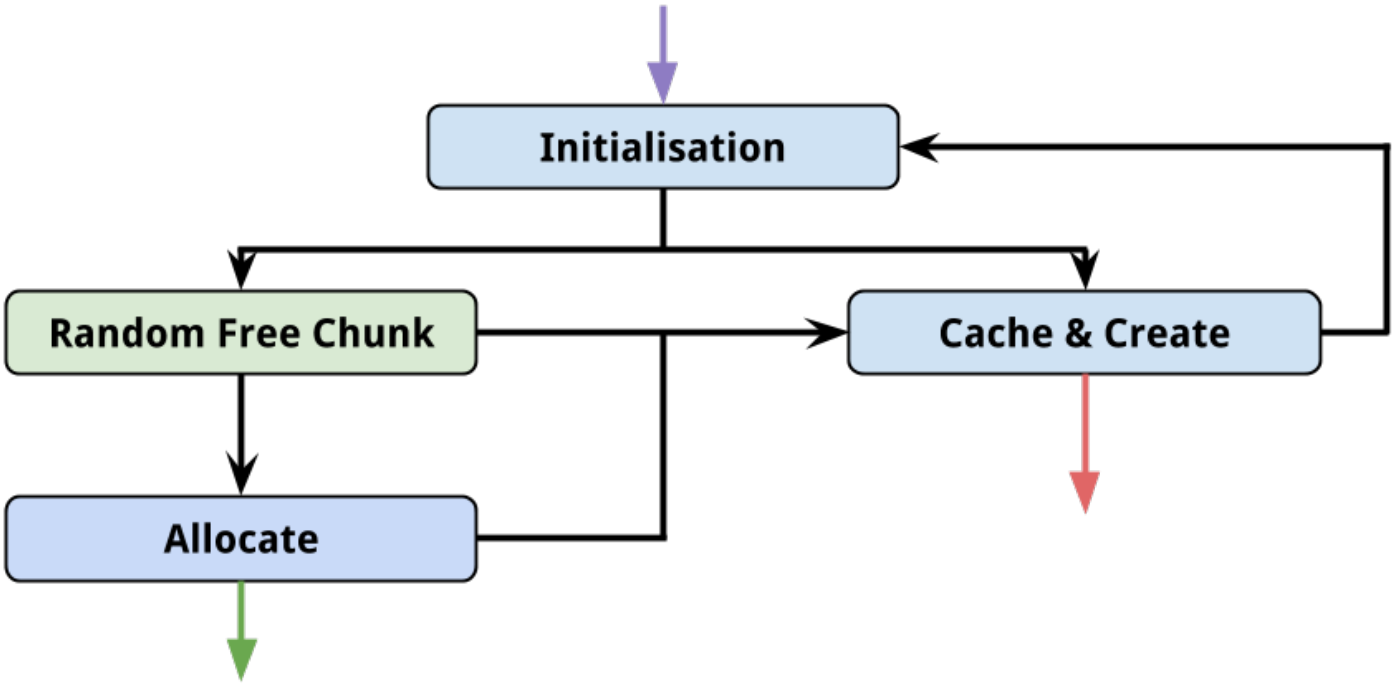
LFH Initialisation



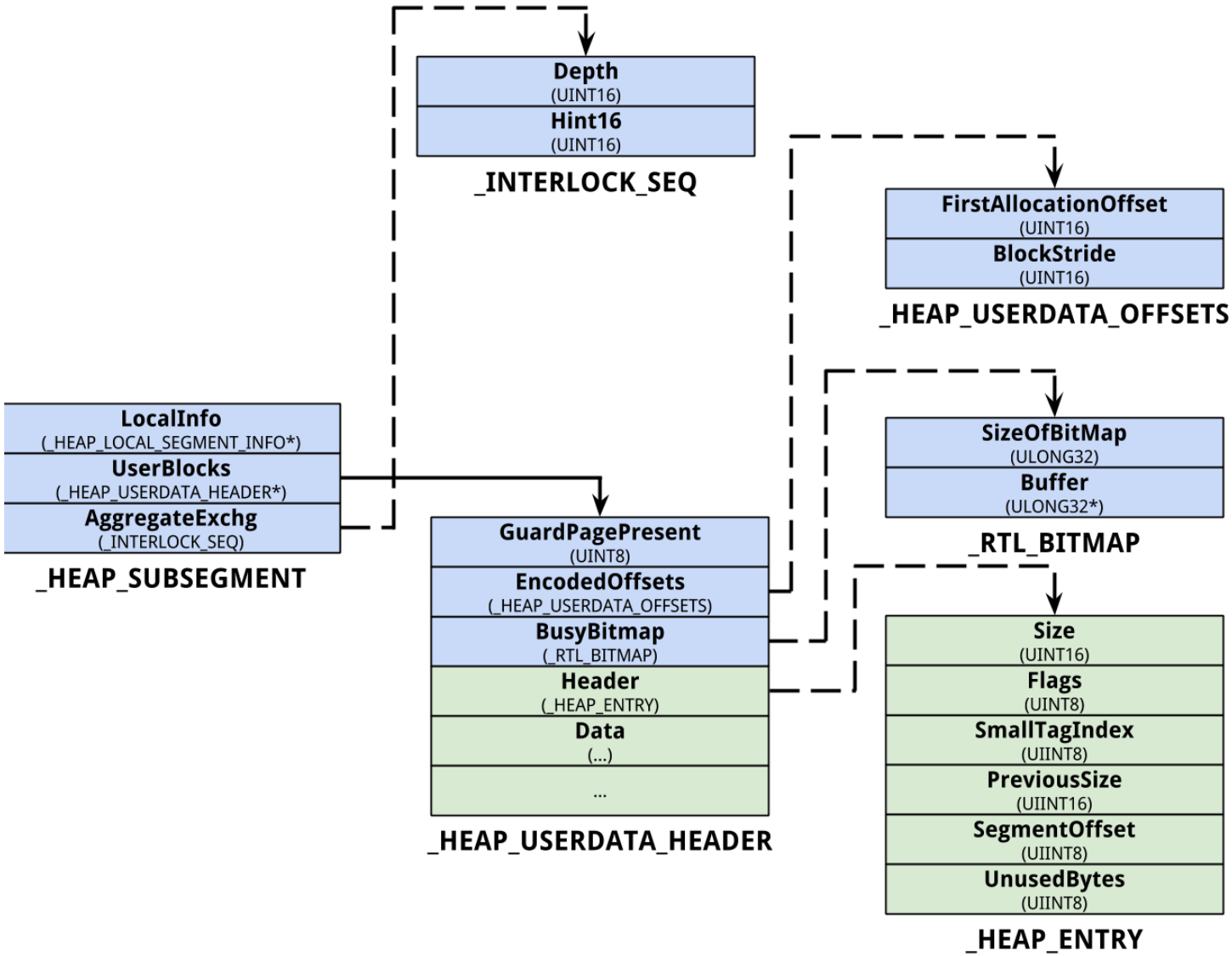
LFH Initialisation



LFH Randomization



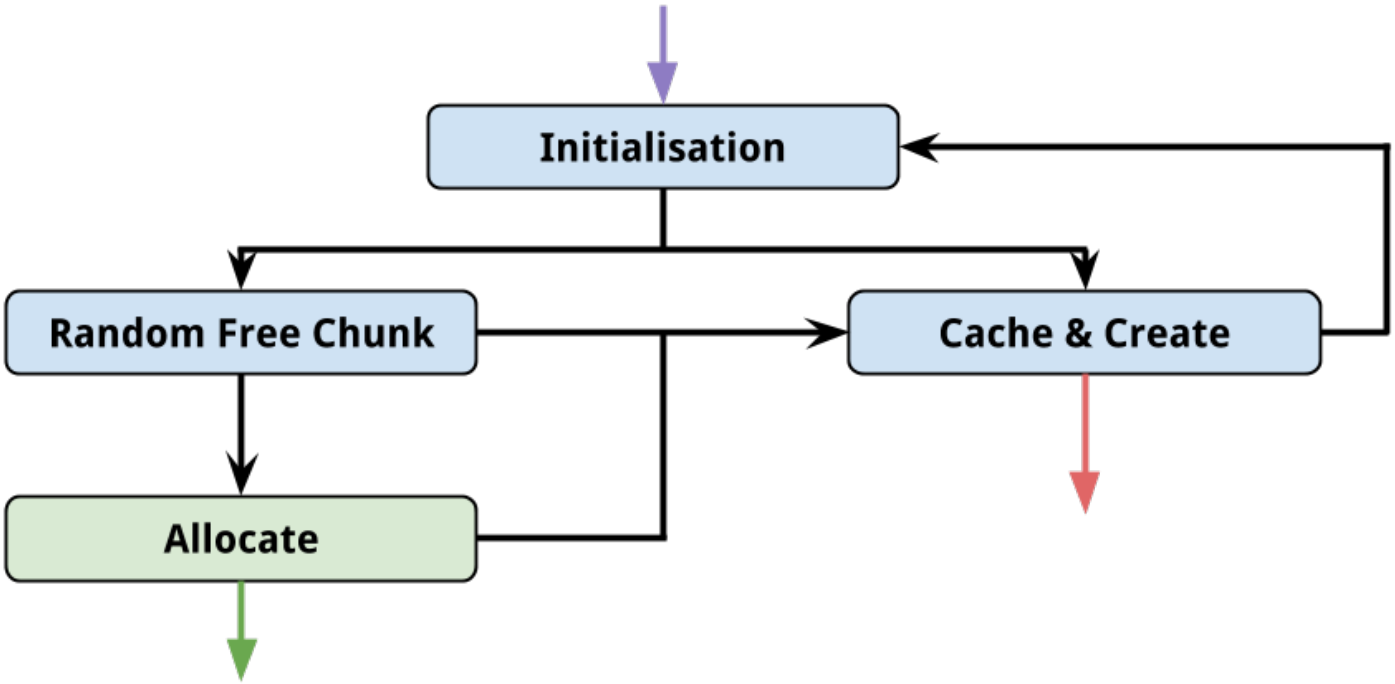
LFH Randomization



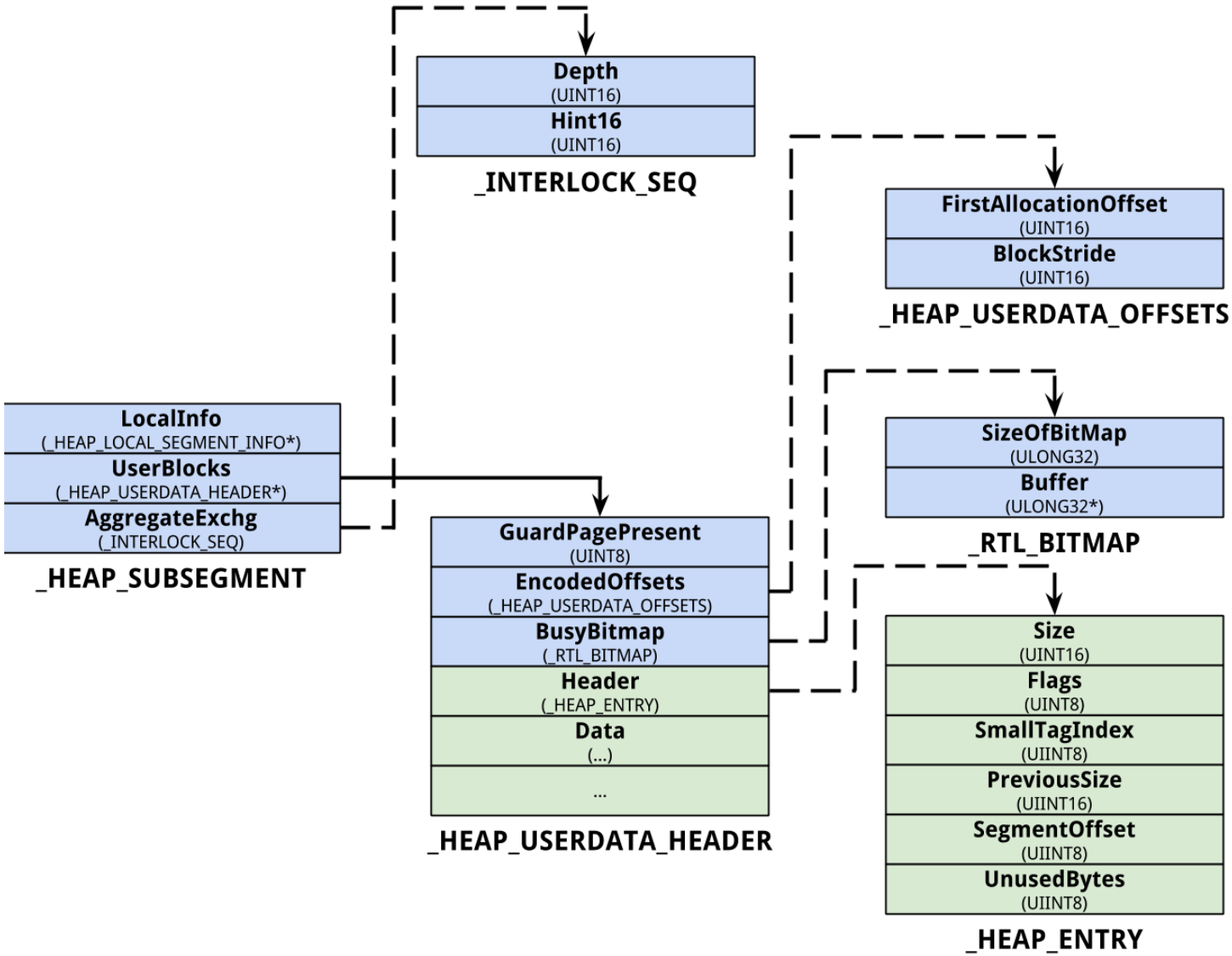
LFH Randomization

- RtlpLowFragHeapRandomData
- LowFragHeapDataSlot (in the TEB)

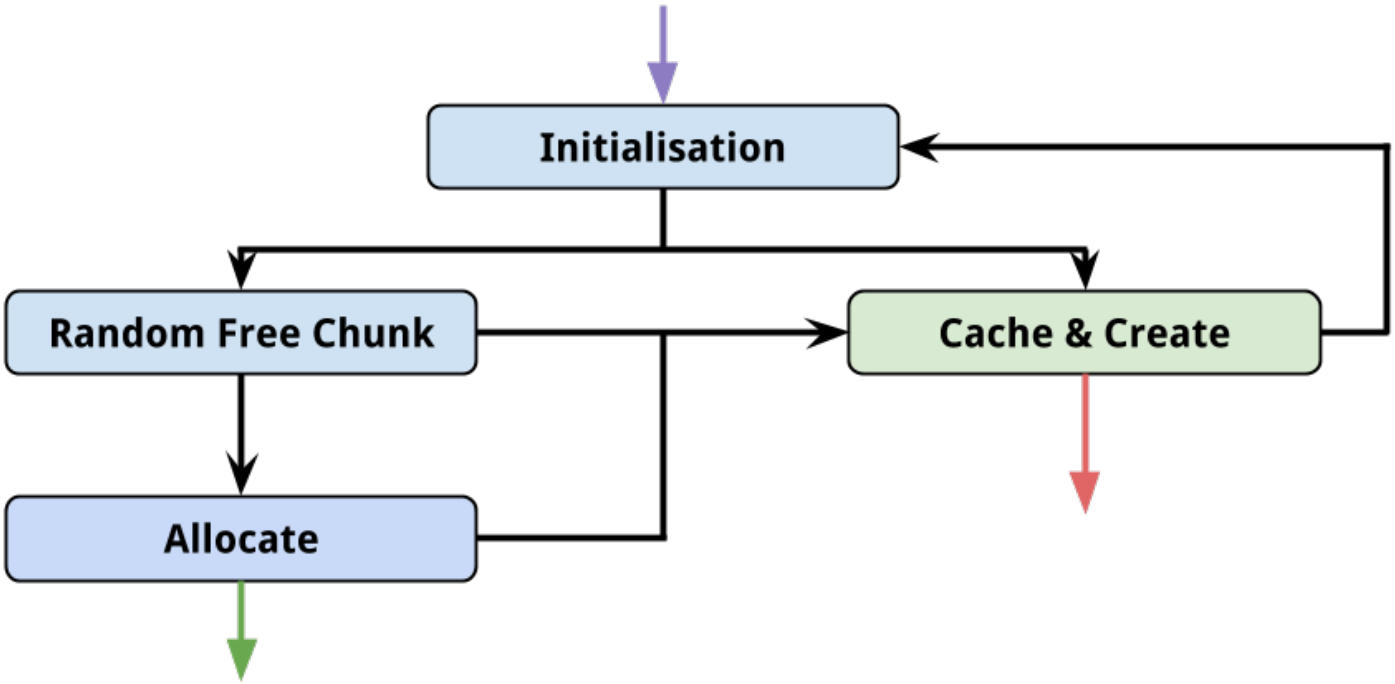
LFH Allocation



LFH Allocation



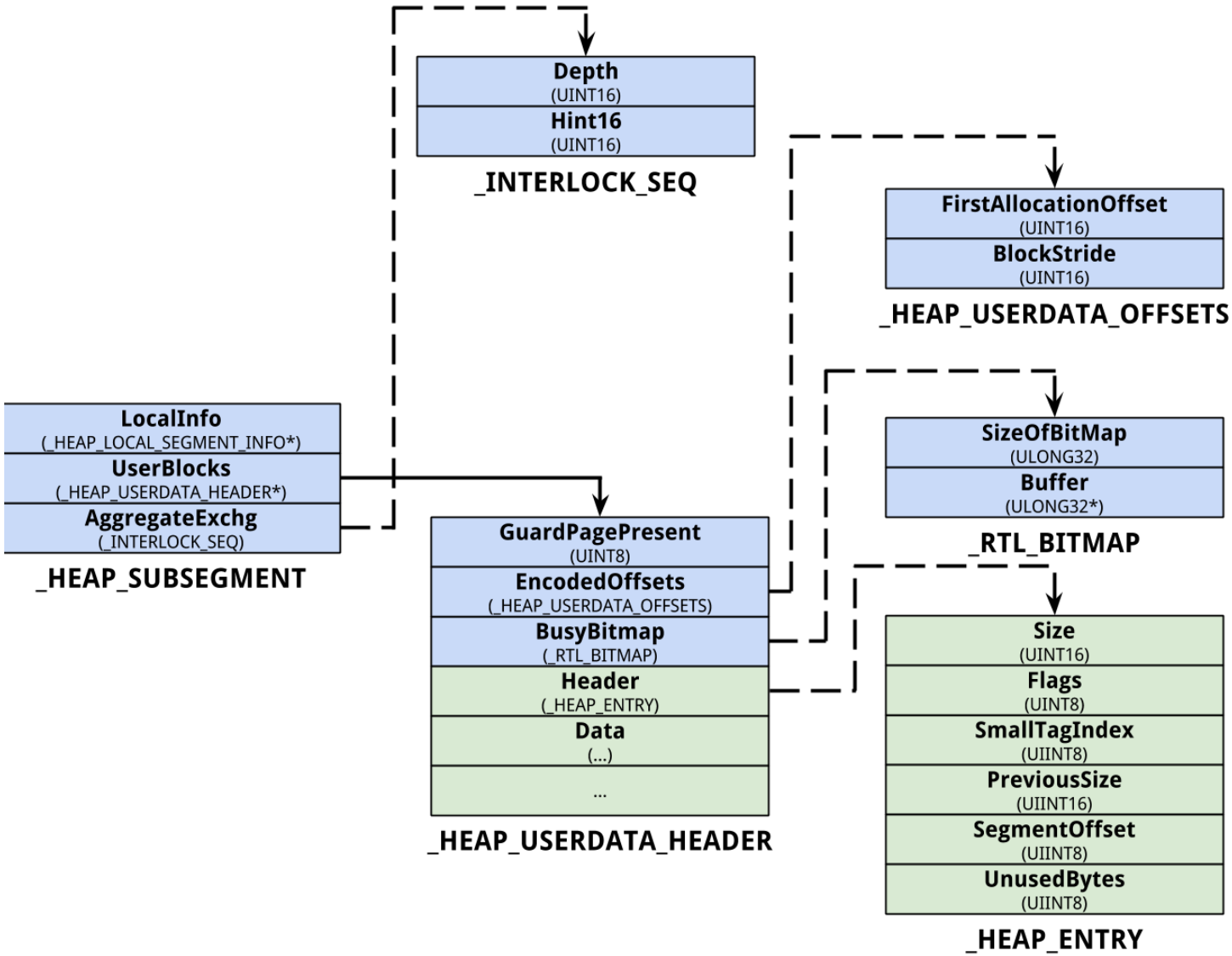
LFH Cache



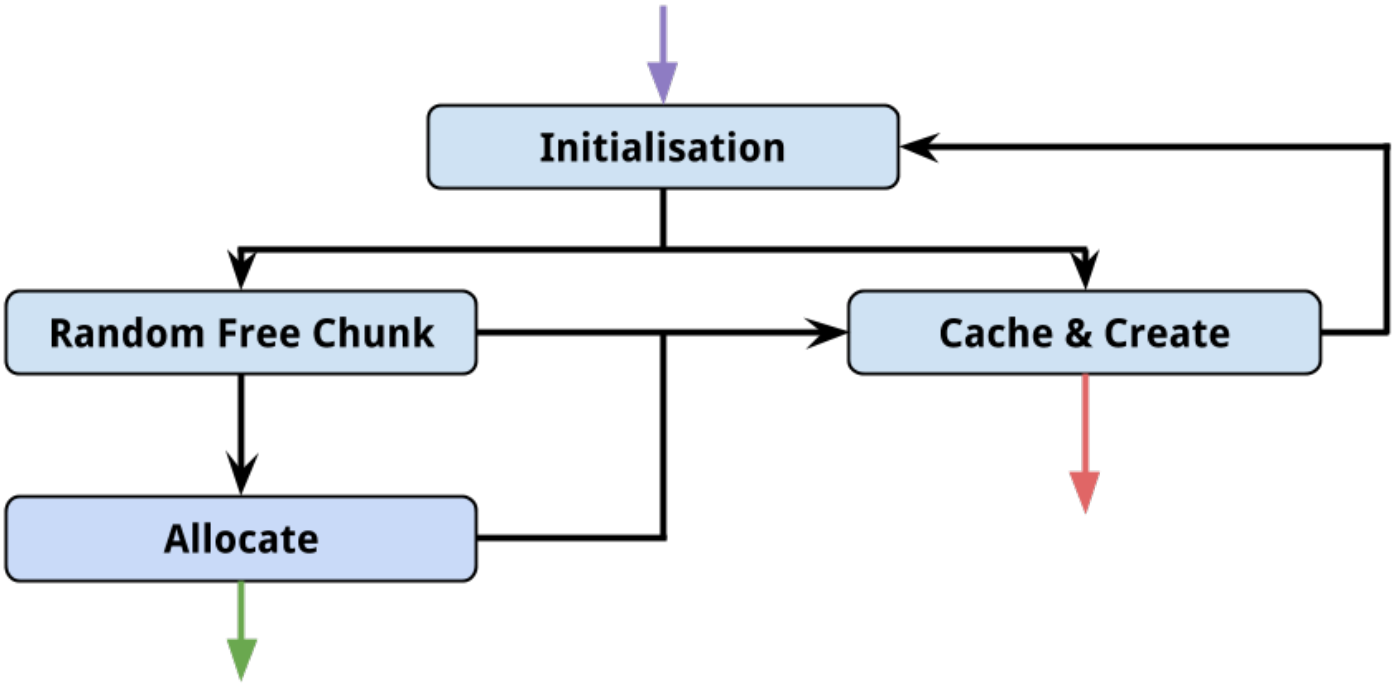
LFH Cache

- Check the cache
- Try to allocate UserBlocks and/or Subsegment
- Fail if RtlAllocateHeap fails
- Update RtlpLowFragHeapRandomData in Subsegment allocation

LFH Cache

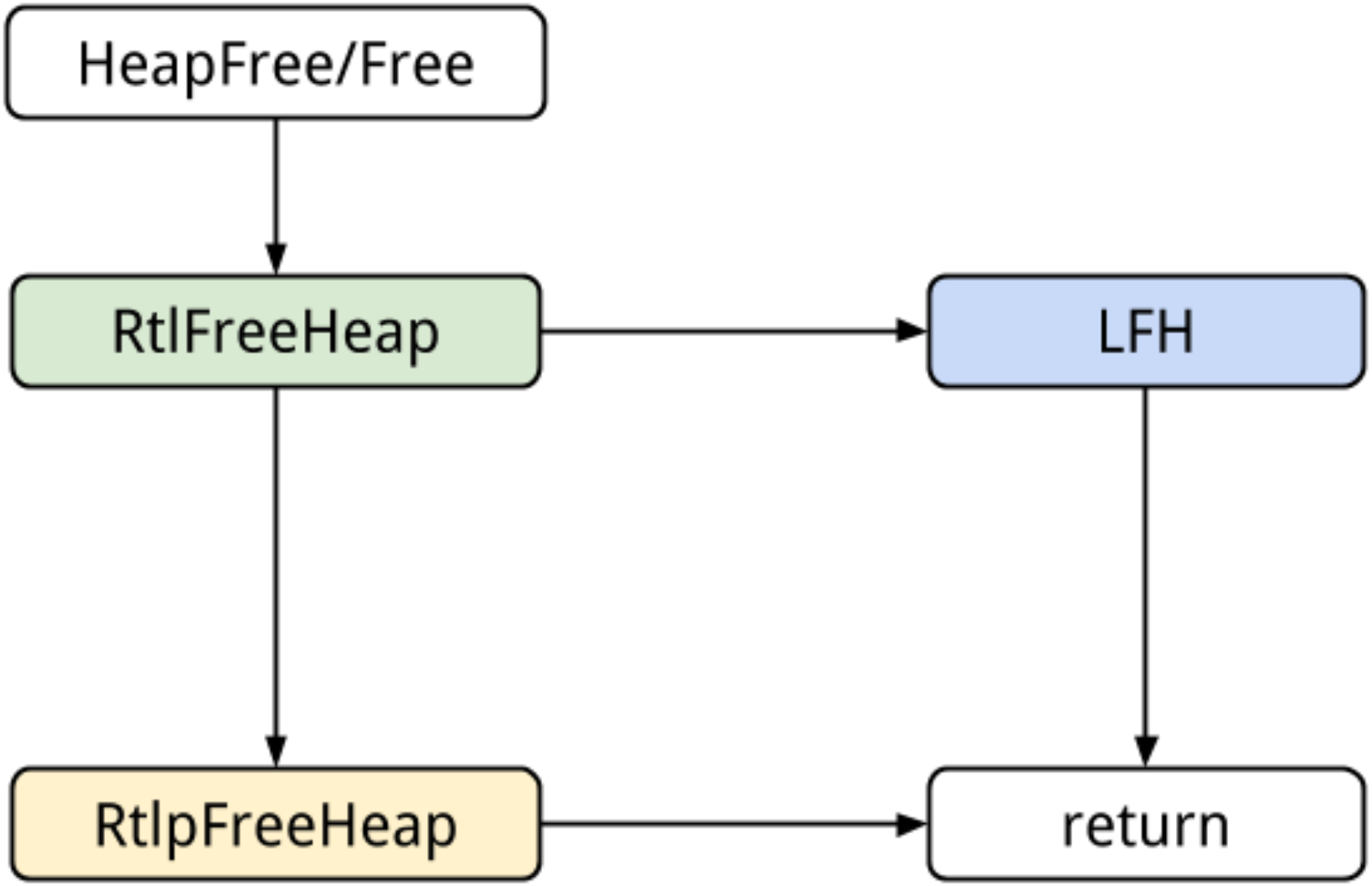


LFH Allocation Workflow



Plan

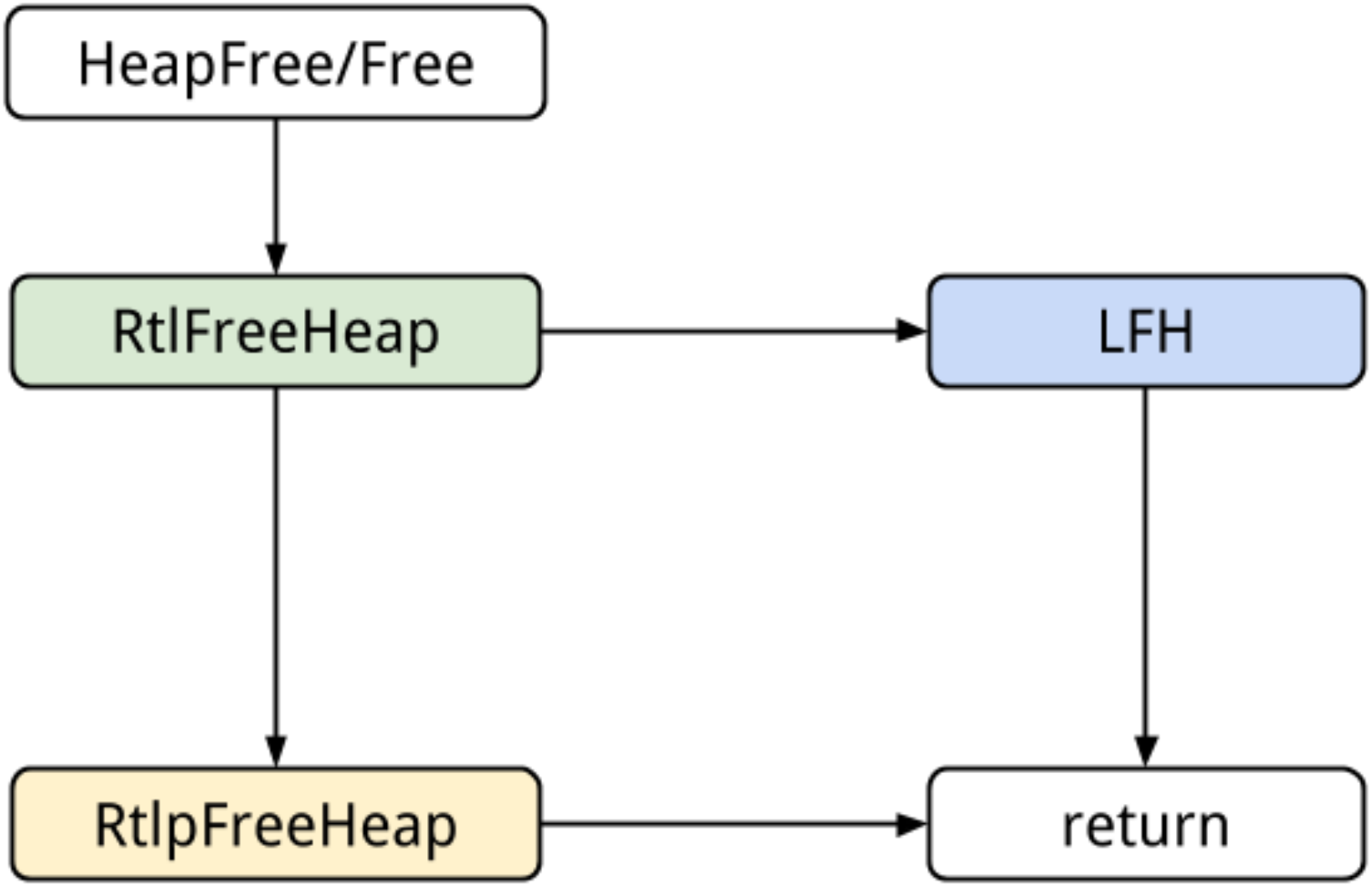
Free Workflow



Free Back-end

- RtlpFreeHeap(_HEAP *Heap, int Flags, _HEAP_ENTRY *Header, void *Chunk)
- Decrement the counter in Heap->FrontEndHeapUsageData[]

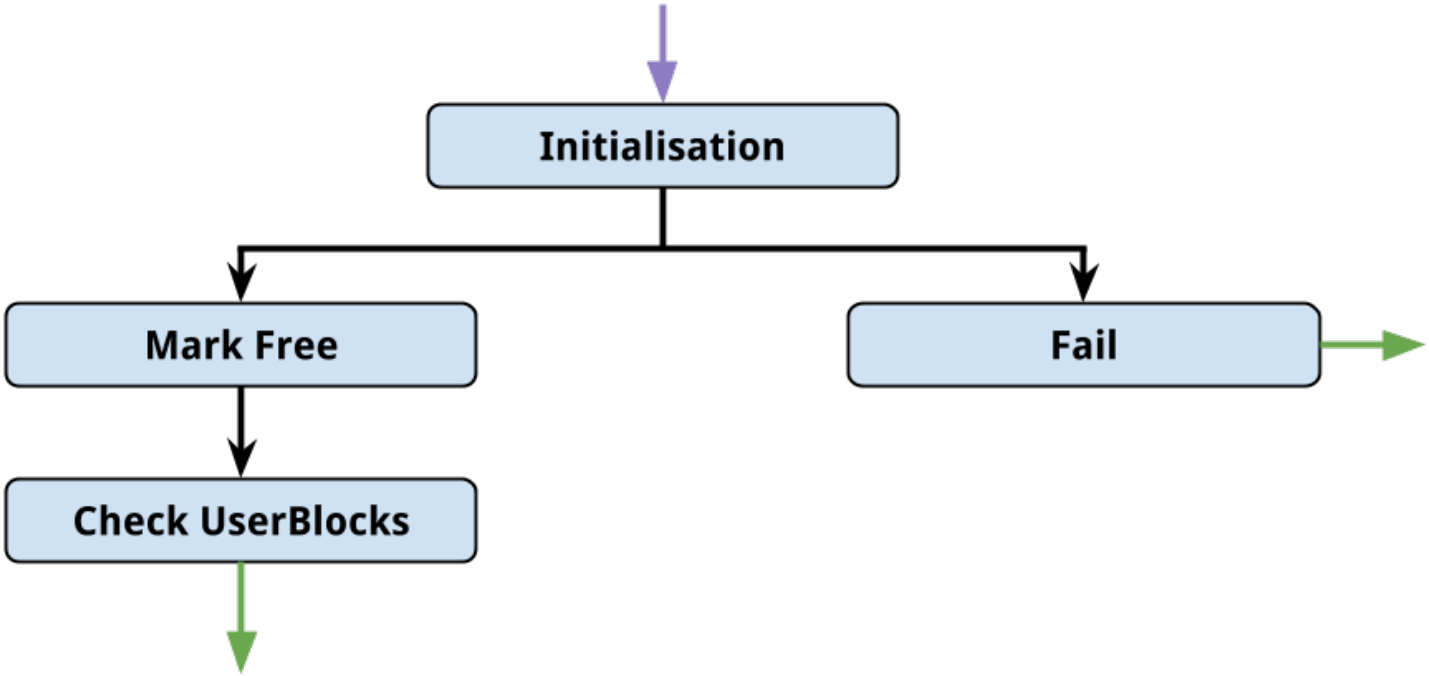
Free Workflow



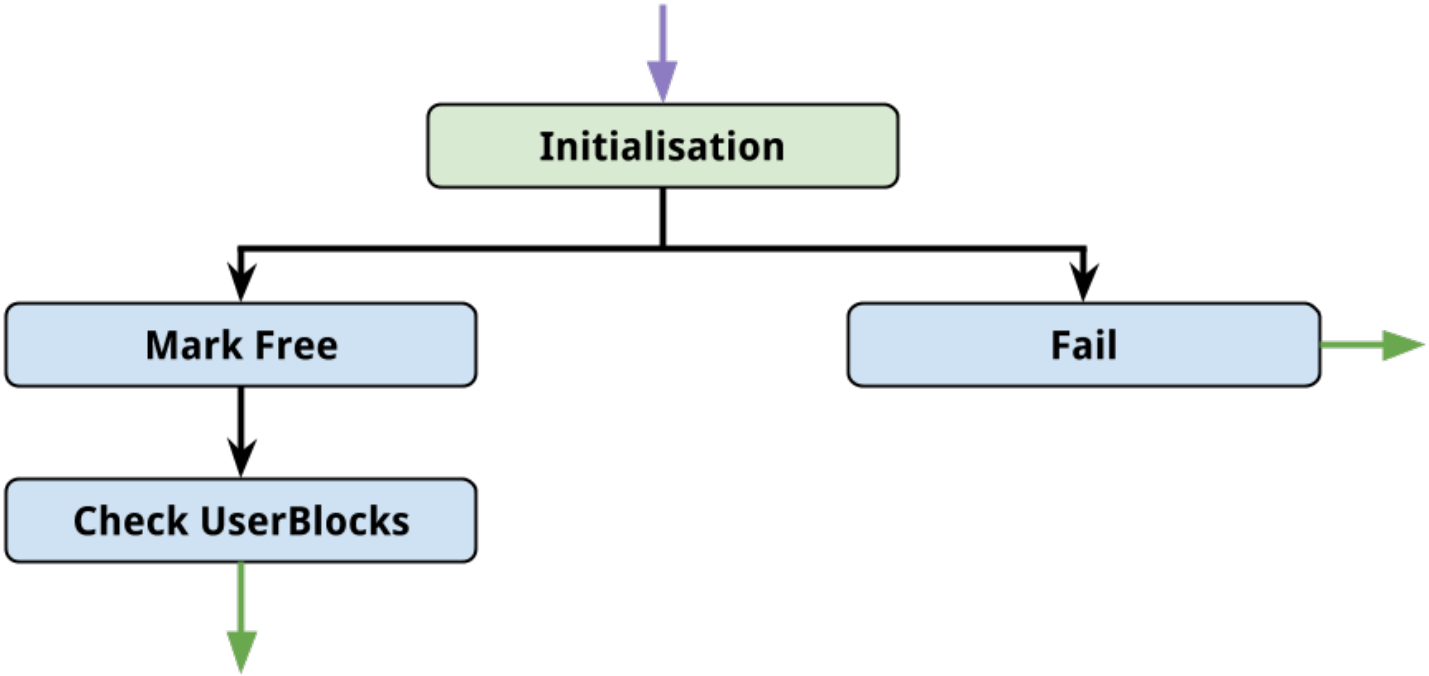
LFH Free

- No longer handled by RtlpLowFragHeapFree
- Same algorithm idea in Windows 8 and 8.1
- Header->UnusedBytes & 0x80
- Always returns true

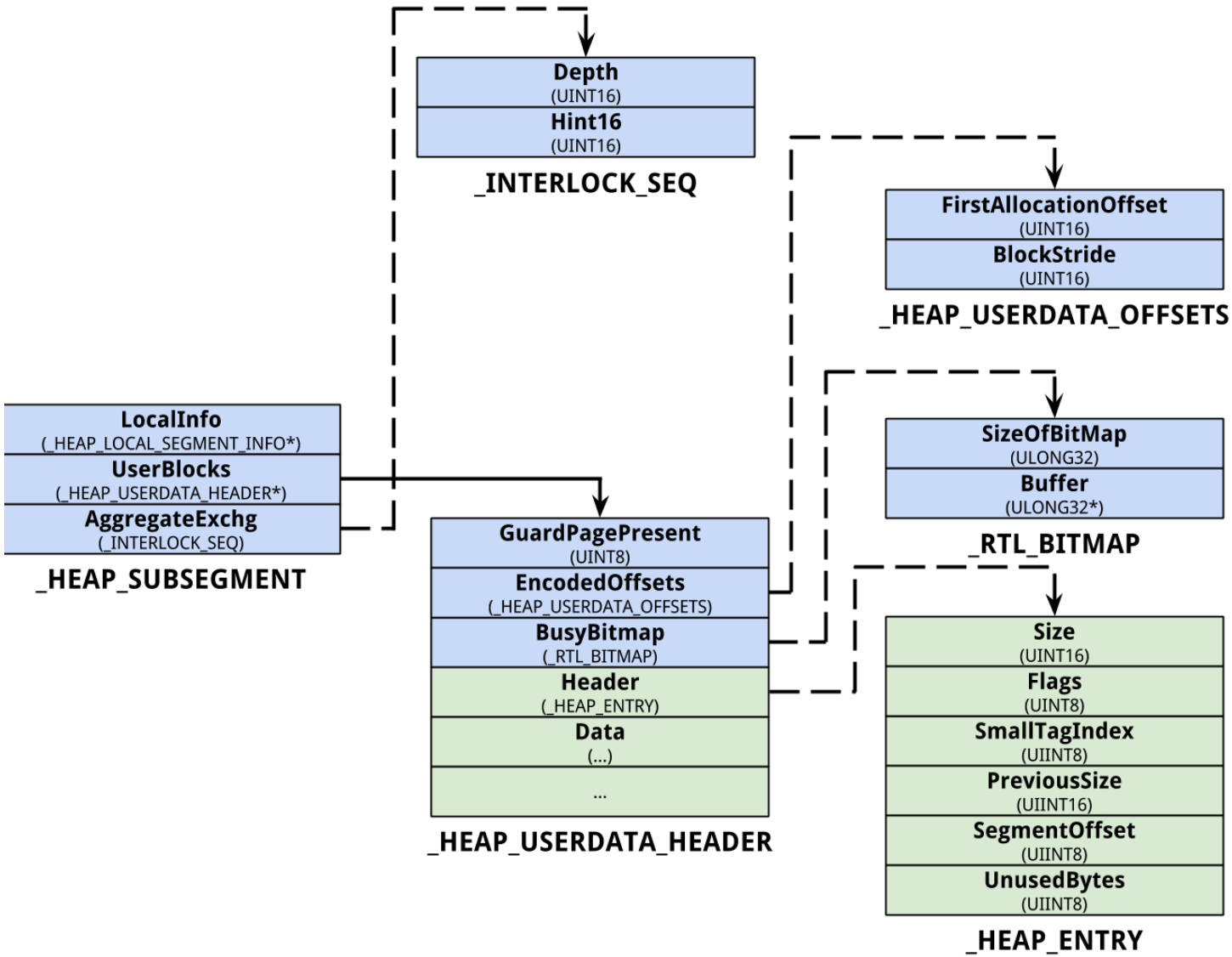
LFH Free Workflow



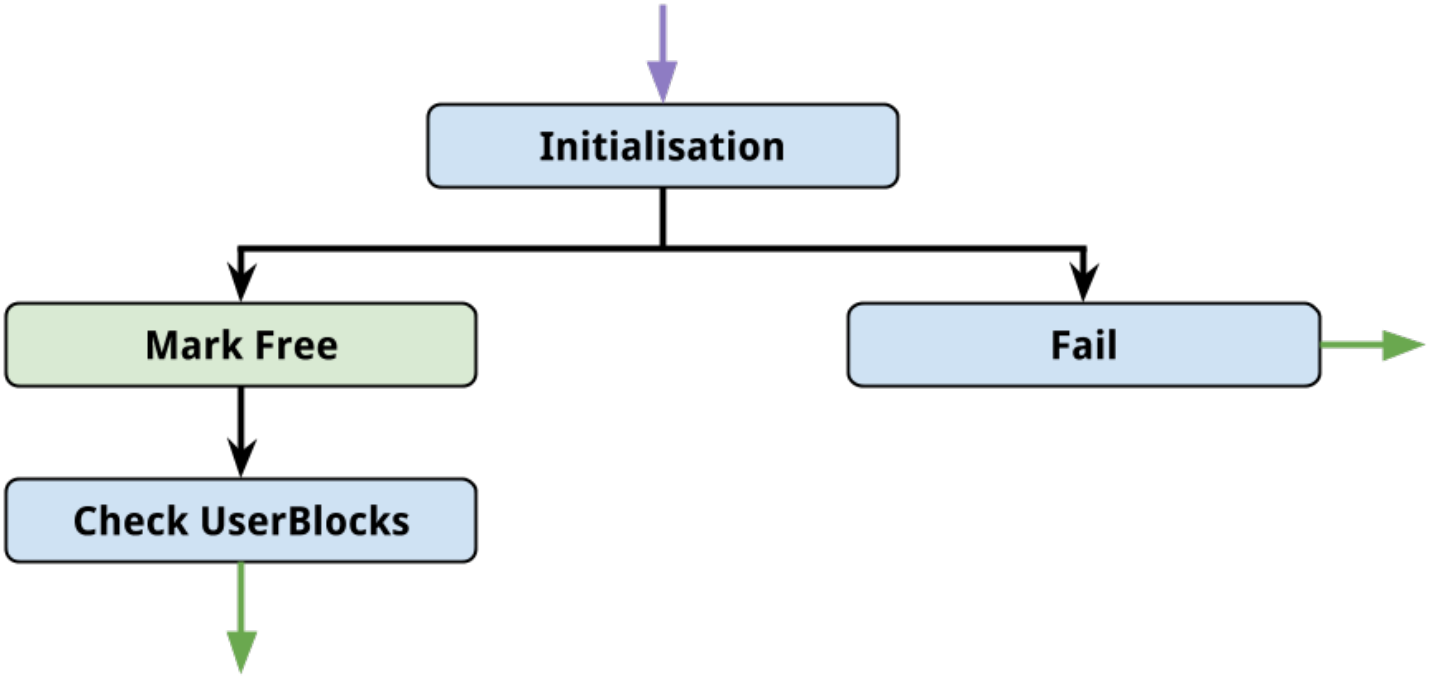
LFH Free Initialisation



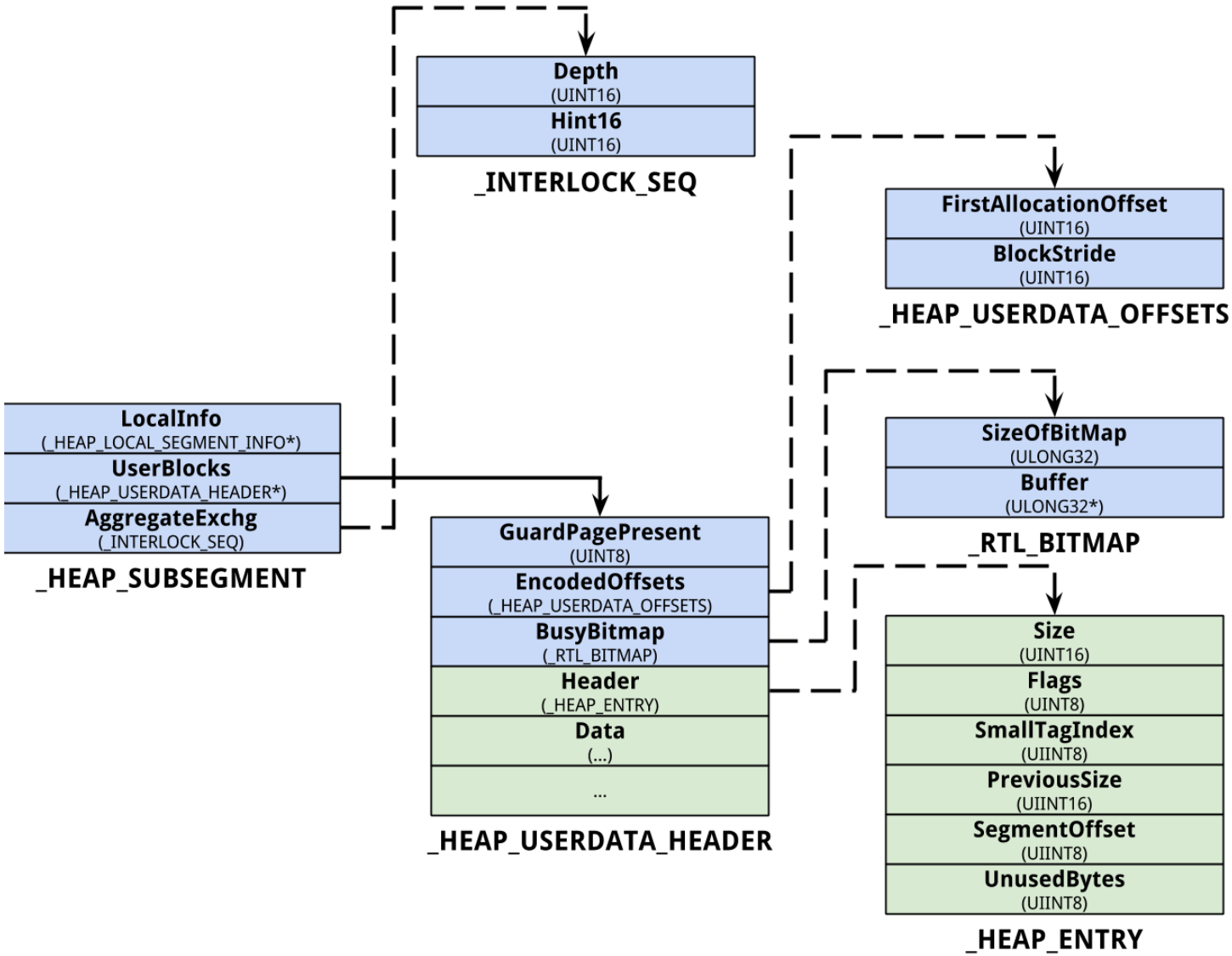
LFH Free Initialisation



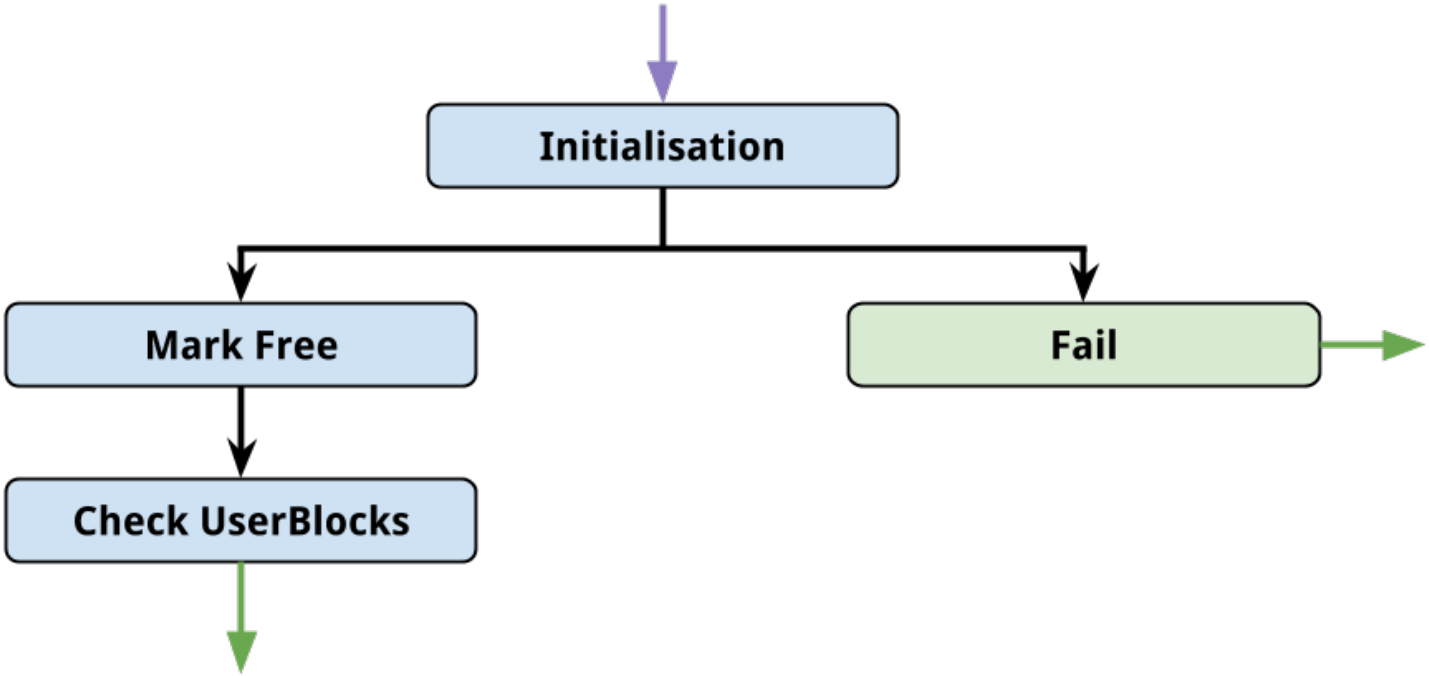
LFH Free Mark



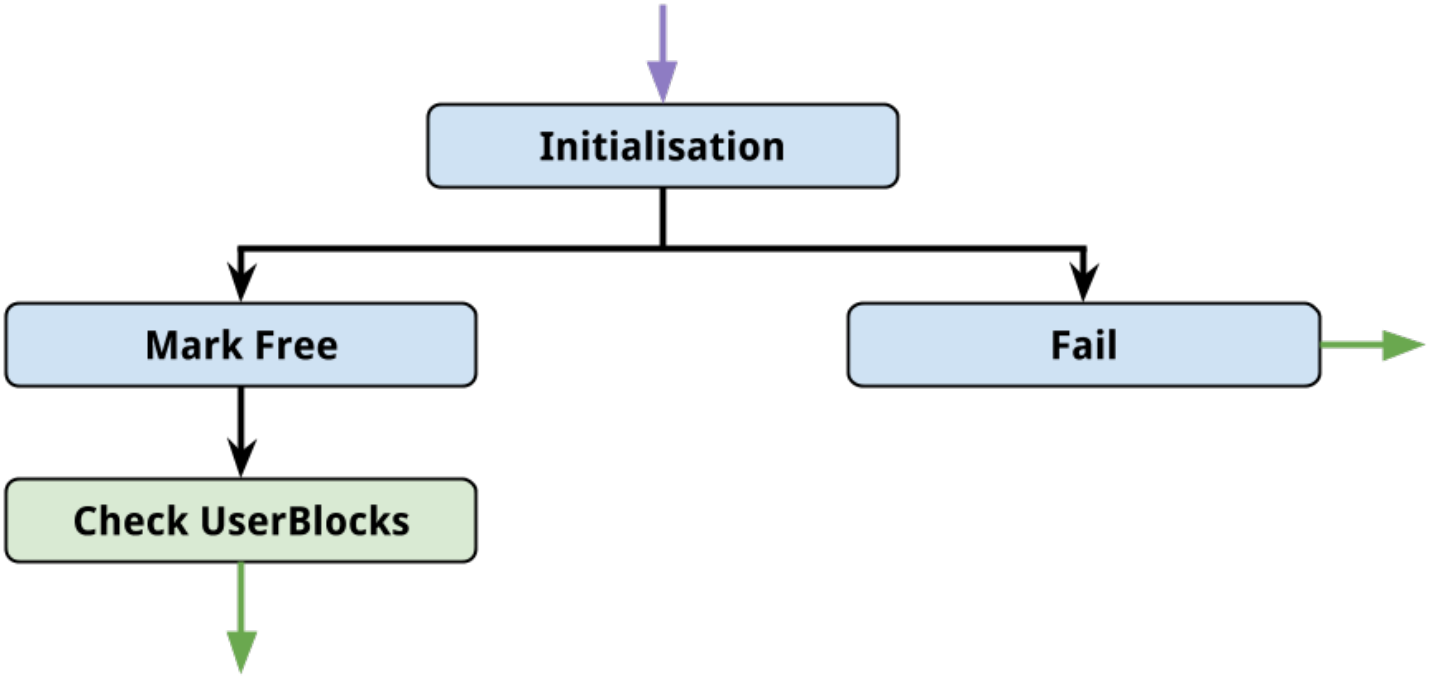
LFH Free Mark



LFH Free Fail



LFH Free Check UserBlocks



Attackers goal

- The goal is always to control eip
- For a ”generic” heap exploitation:
  - Arbitrary write
  - Trigger a free
  - ...

Safe Linking/Unlinking

- Moore 2005, but also in Linux malloc. . .
- The idea is to corrupt a double-linked list (stored in metadata)
- Could allow arbitrary 4-write
- Check introduced in Windows XP SP2, and generalized since then

Arbitrary Free

- Ben Hawkes 2008
- Before Windows 8:

```
if (Header->UnusedBytes == 0x5)
    Header -= 8 * Header->SegmentOffset;
```

- Overwrite of a _HEAP_ENTRY would allow a semi-arbitrary free
- Windows 8 introduces a check

- LFH overflow: structures have changed
  - FrontEndHeapUsageData
  - _RTL_BITMAP
- Off-by-one: encoded and changes the structure
- Heap Overflows: no more free of the heap

FreeEntryOffset Overwrite

- Chris Valasek 2010
- Before Windows 8, a free chunk would contain a NextOffset field of a free chunk in the first 2 bytes after the _HEAP_ENTRY
- Overwrite it so that a chunk will be allocated and allow a semi-controlled allocation (the next one)
- Rewrite data of another chunk :)
- NextOffset doesn't exist in Windows 8, use _HEAP_USERDATA_HEADER to locate free chunks

LFH GuardPages

- Since Windows 8
- Created during a UserBlocks allocation
- Protection against sequential overflow
- Protection to prevent UserBlocks overwrite
- PAGE_NOACCESS
- Possible to avoid triggering them by doing few allocations

Fast fail

- HeapEnableTerminationOnCorruption
- Fast fail is an interrupt (int 0x29) which halts the execution of the process

Non-deterministic virtual allocation

- For a request of a certain size (> VirtualMemoryThreshold), use NtAllocateVirtualMemory
- Before Windows 8: predictable memory layouts.
- Since Windows 8: the virtual allocation start at a random offset within the whole virtual chunk.

Non-deterministic allocation

- Already covered

_HEAP_USERDATA_HEADER Overwrite

- In Windows 8, the BlockStride and the FirstAllocationOffset are not encoded
- Header when allocating:

```
Header = (_HEAP_ENTRY) UserBlocks +
UserBlocks->FirstAllocationOffset +
(NewHint * UserBlocks->BlockStride);
```

- If we overwrite FirstAllocationOffset and/or BlockStride, a semi-arbitrary address is returned by the LFH
- Since Windows 8.1, FirstAllocationOffset and BlockStride are encoded: EncodedOffsets ^ UserBlocks ^ LFH ^ RtlpLFHKey

Plan

4 Observations

Determinism

- Non-deterministic allocations break a lot of things
- How to be determinist again?
- RtlpLowFragHeapRandomData are random but fix
- LowFragHeapDataSlot is a counter with a modulo 0x100
- Just allocate and free a 0x100 chunk to have the same value again from RtlpLowFragHeapRandomData
- Need to avoid subsegment allocation
- Need to be able to allocate and free the specific size we want

Determinism

- Not only useful for use-after-free, but also for overflow
  - Allocate the vulnerable chunk
  - Allocate and Free for 0x100 times
  - Allocate the chunk to overflow
  - Trigger the overflow
- Work only if we are sure that the chunk next to the vulnerable one is free (and we are lucky enough not to allocate the last chunk. . .)

Bitmap Flipping 2.0

- Valasek & Mandt 2012
- When a free is made, the offset of the byte to clear in the bitmap is determined by Header->Previous
- If we can overwrite the Heap->Previous it could to go out of the bitmap and set one bit to 0
- Problem: we need to overwrite the subsegment encoded with a good value
- Should still work if it can be triggered

Plan

5 Conclusion

Conclusion

”Application specific attacks are the future”
Ben Hawkes 2008

Link

- http://illmatics.com/Windows%208%20Heap%20Internals.pdf
- http://illmatics.com/Understanding_the_LFH.pdf
- https://media.blackhat.com/eu-13/briefings/Liu/bh-eu-13-liu-advanced-heap-WP.pdf
- https://www.lateralsecurity.com/downloads/hawkes_ruxcon-nov-2008.pdf
- http://sebug.net/paper/Meeting-Documents/hitbsecconf2012ams/D2T2%20-%20Steven%20Seeley%20-%20Ghost%20In%20the%20Windows%207%20Allocator.pdf
- http://www.blackhat.com/presentations/bh-usa-09/MCDONALD/BHUSA09-McDonald-WindowsHeap-PAPER.pdf
- ntdll.dll 6.3.9600.17031 (Windows 8.1)
- ntdll.dll 6.2.9200.16384 (Windows 8)