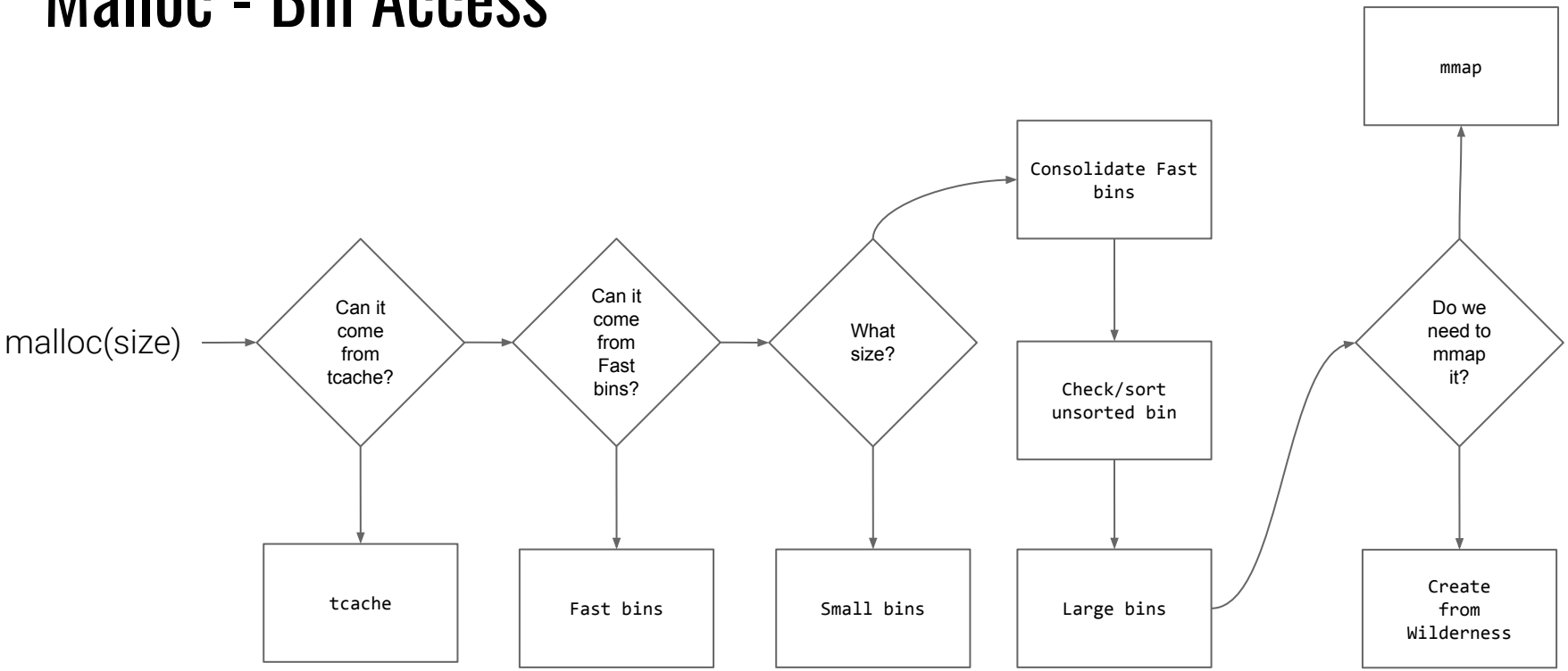
Module: Dynamic Allocator Misuse II

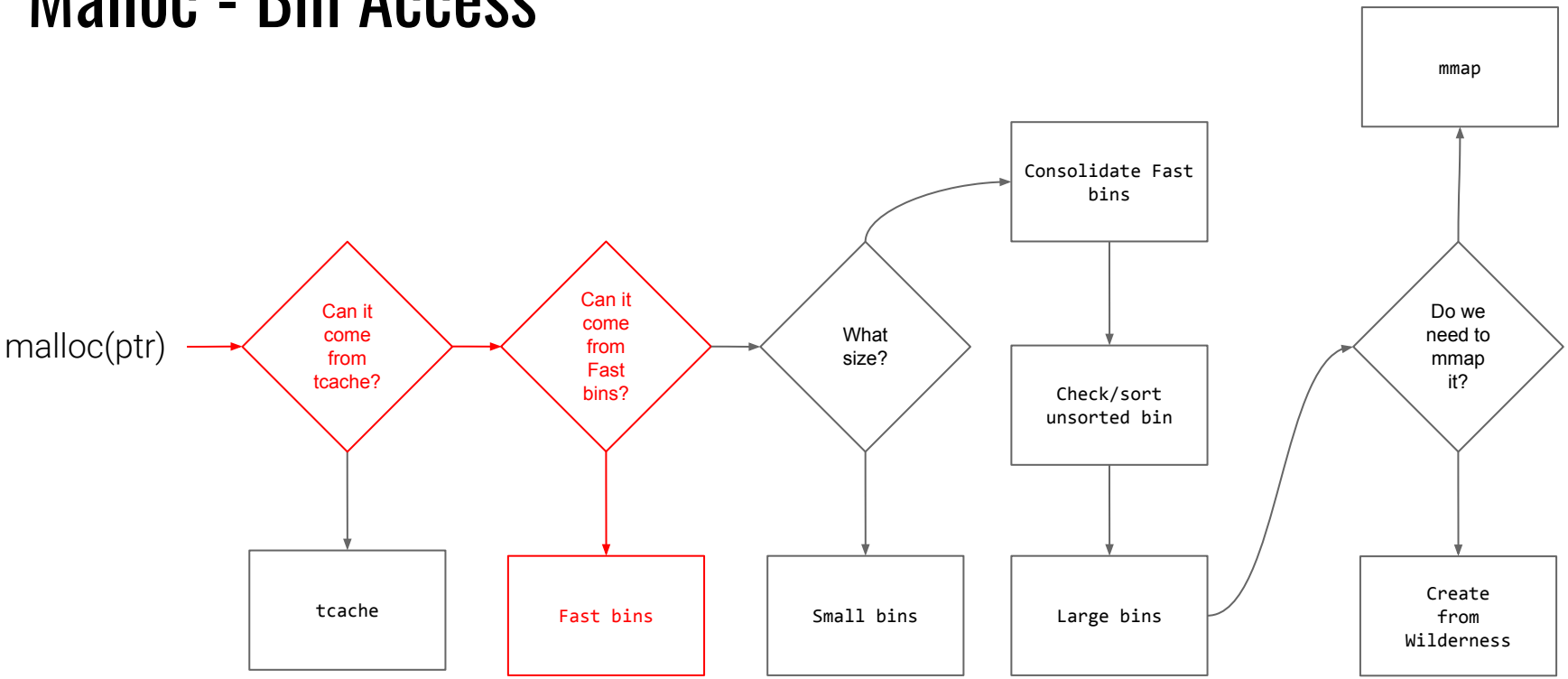
The Bins

Robert Wasinger
Arizona State University

Malloc - Bin Access



Malloc - Bin Access

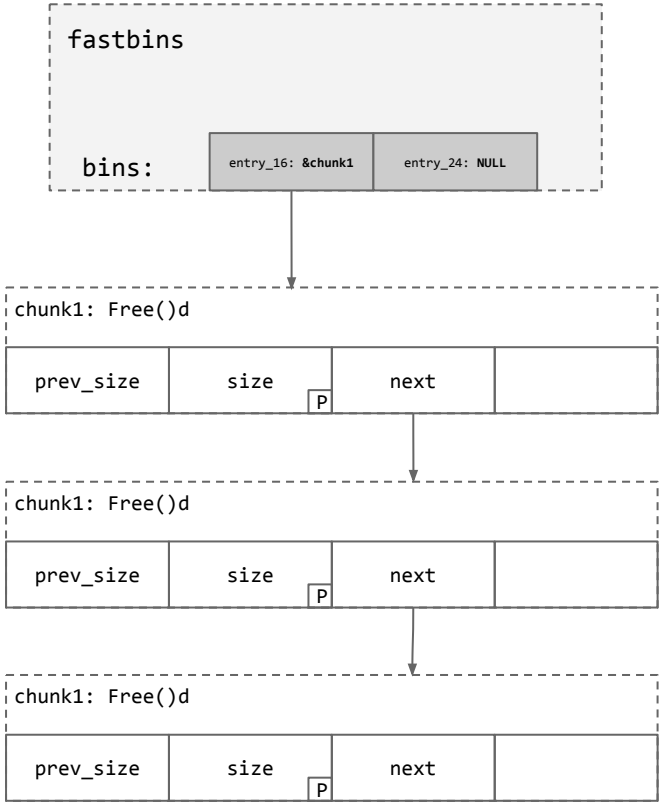


Fast Bins

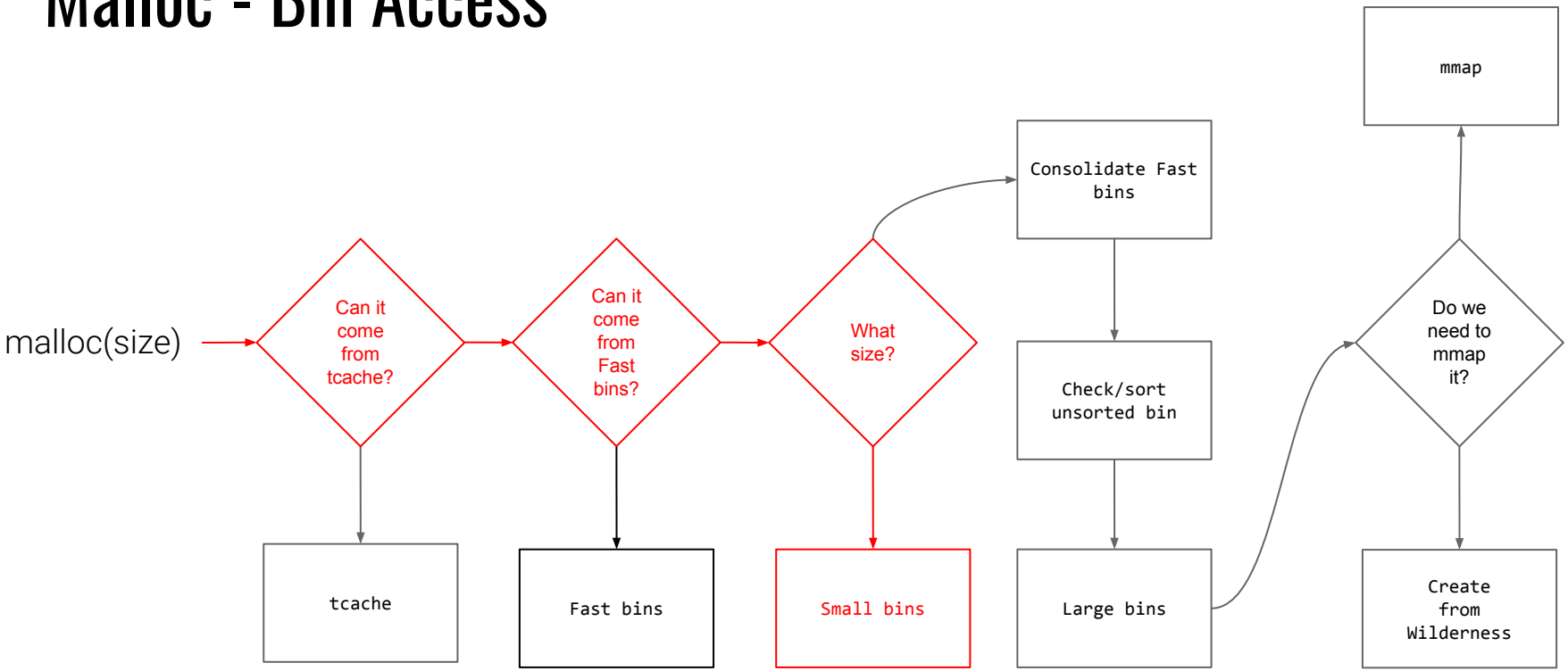
- Singly linked list with safe-linking - similar to tcache
- Bin lists grow to unlimited length
- Bins of constant size up to 88 bytes
- P bit is never cleared for chunks in the fast bin
- Only checks top chunk for double-free

https://elixir.bootlin.com/glibc/latest/source/malloc/malloc.c#L1728

Fast Bins - List



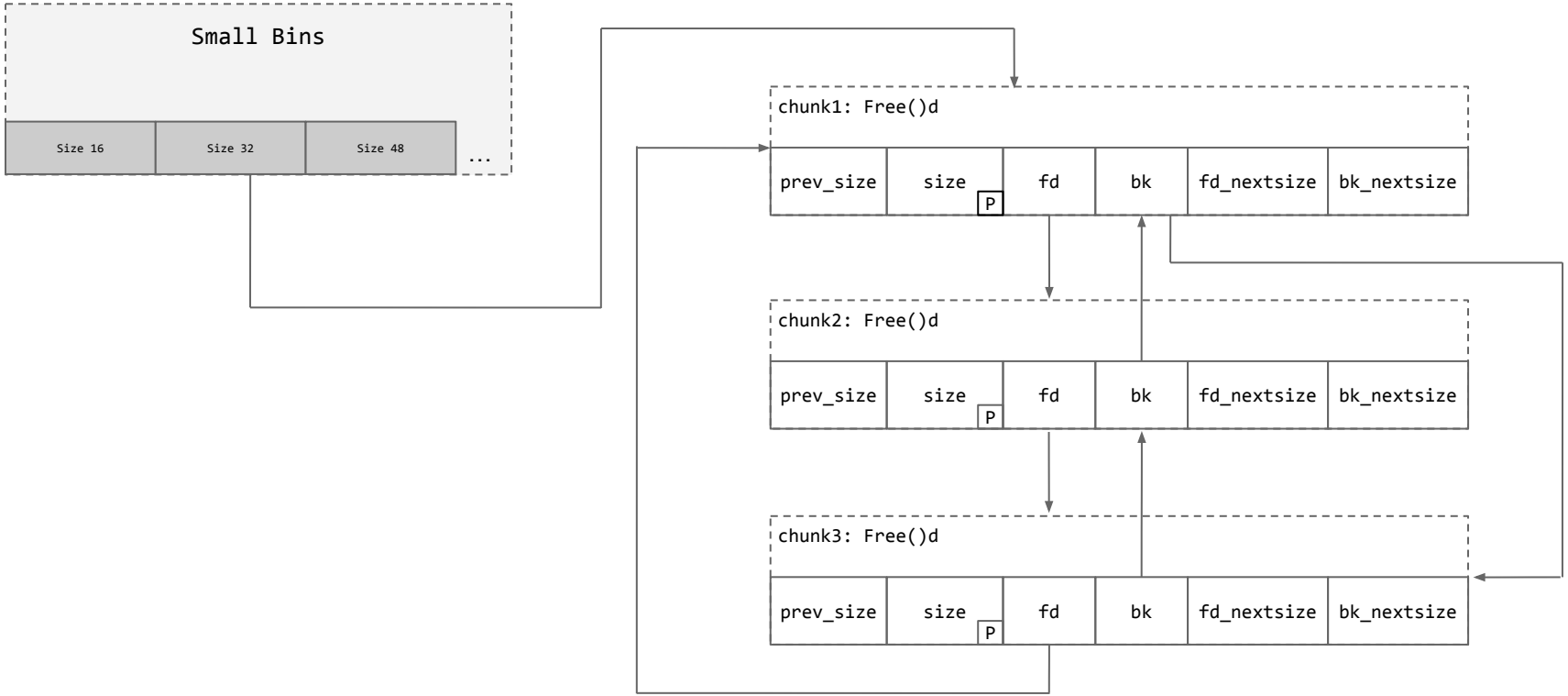
Malloc - Bin Access



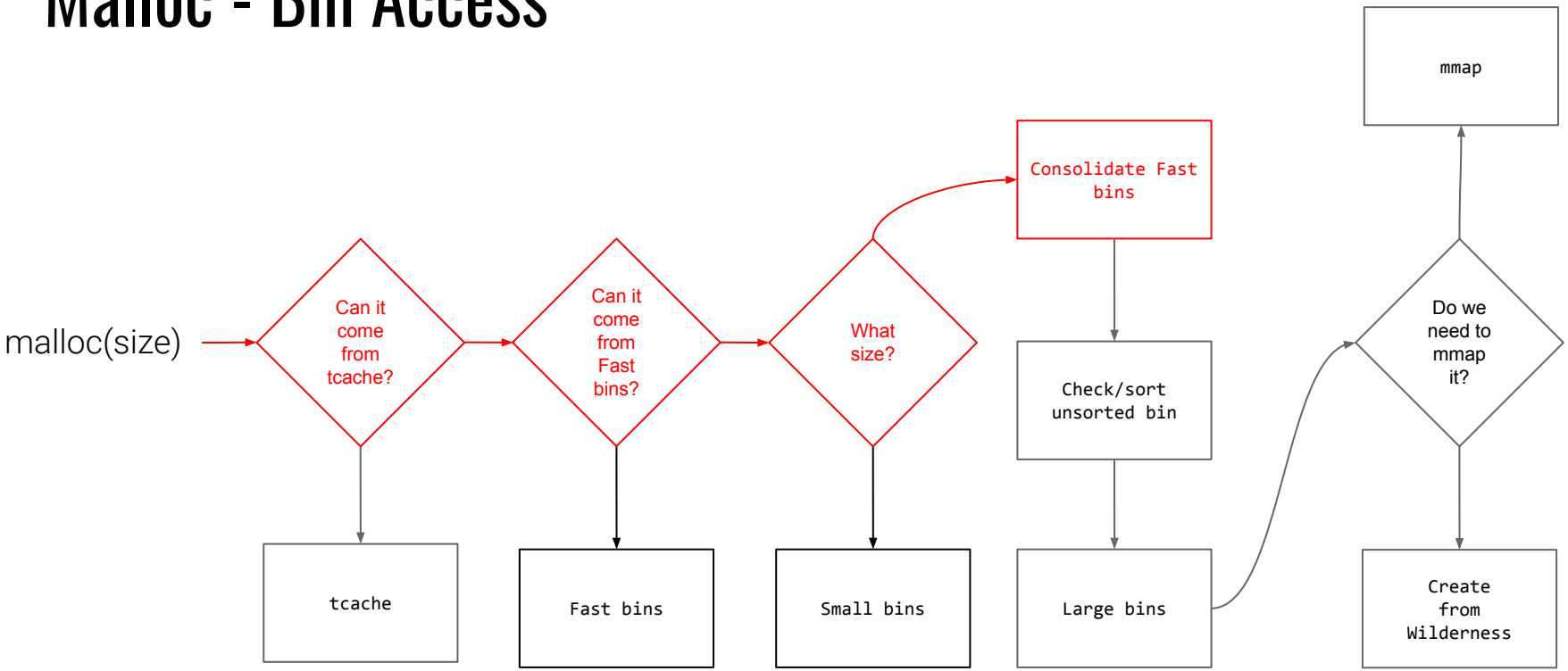
Small Bins

- Doubly linked lists
- Bins of constant size up to 1024 bytes
- Fast access, but capable of consolidating

Small Bins


Small Bins
Size 16
Size 32
Size 48
...
chunk1: Free()d
prev_size
size
P
fd
bk
fd_nextsize
bk_nextsize
chunk2: Free()d
prev_size
size
P
fd
bk
fd_nextsize
bk_nextsize
chunk3: Free()d
prev_size
size
P
fd
bk
fd_nextsize
bk_nextsize

Malloc - Bin Access



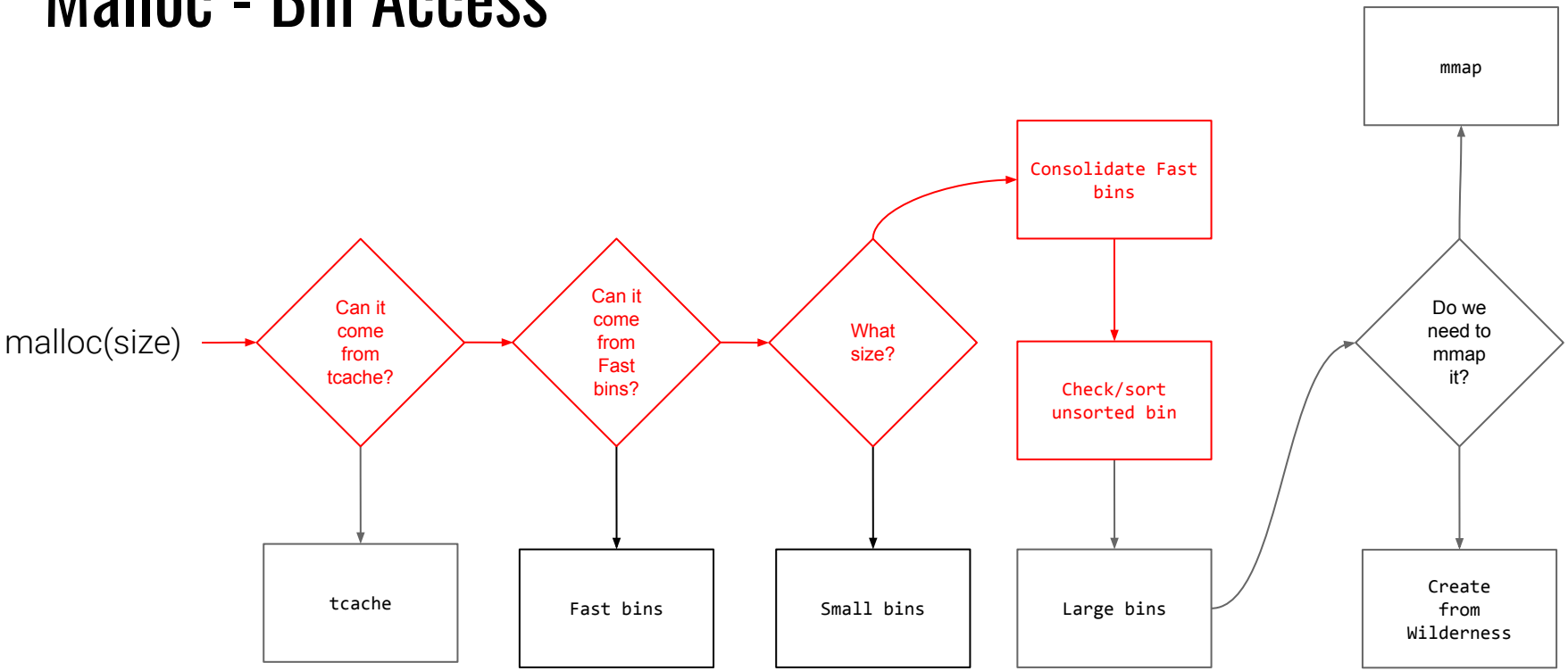
https://elixir.bootlin.com/glibc/latest/source/malloc/malloc.c#L3937

Fast Bins - Clearing and Consolidating

- If a chunk is malloc'd over 1024 bytes in size
  - This is to prevent fragmentation
- If a chunk is freed around ~ 65 KB in size (this was chosen heuristically)
  - https://elixir.bootlin.com/glibc/glibc-2.37/source/malloc/malloc.c#L1743

https://elixir.bootlin.com/glibc/latest/source/malloc/malloc.c#L4700

Malloc - Bin Access



https://elixir.bootlin.com/glibc/latest/source/malloc/malloc.c#L3937

Unsorted Bin

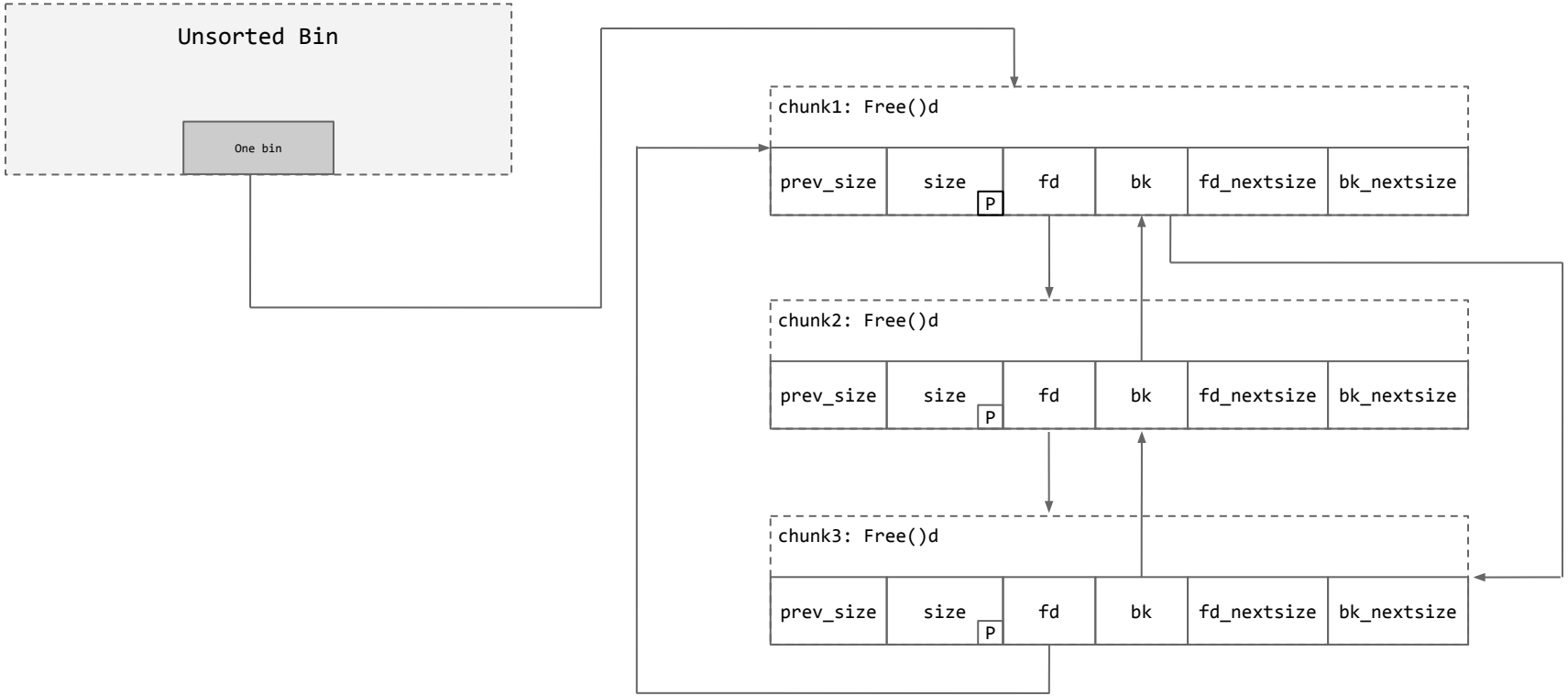
- Doubly linked list
- Holds large and small bin values (anything that cannot go in fast bins)

On Malloc:

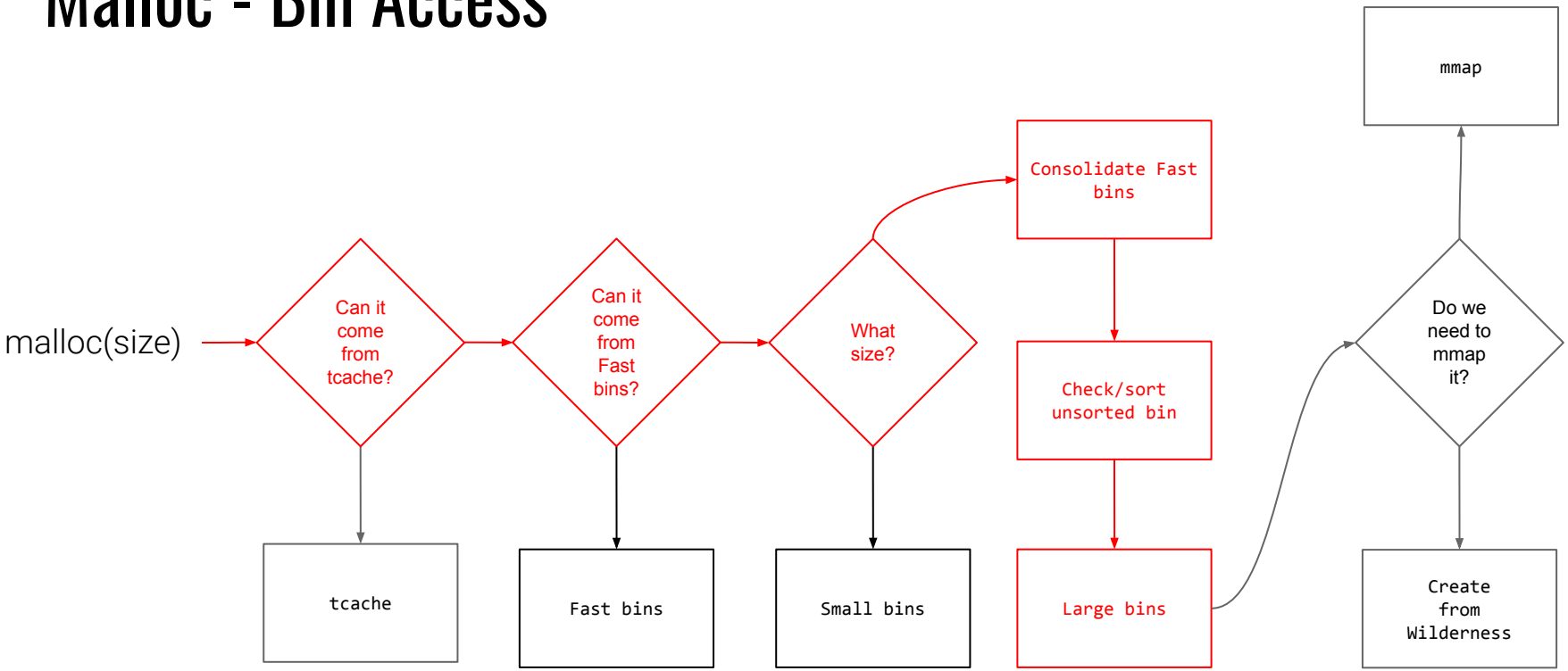
- Unsorted bin chunks are checked
- If chunk does not satisfy malloc, it is placed in appropriate small/large bin

https://elixir.bootlin.com/glibc/latest/source/malloc/malloc.c#L3980

Unsorted Bin

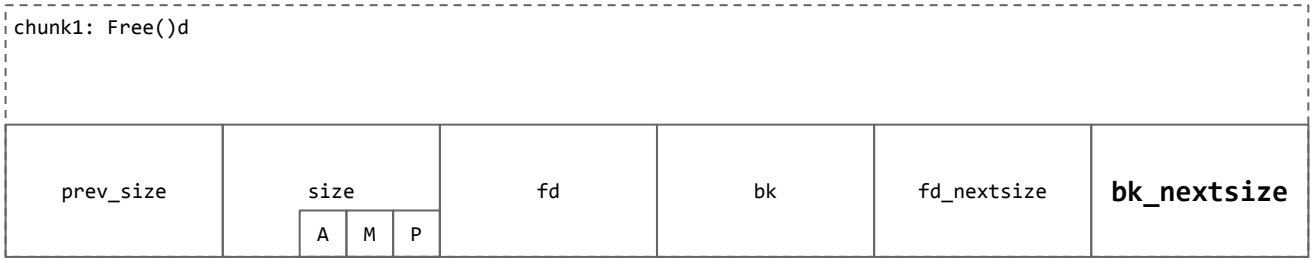


Malloc - Bin Access



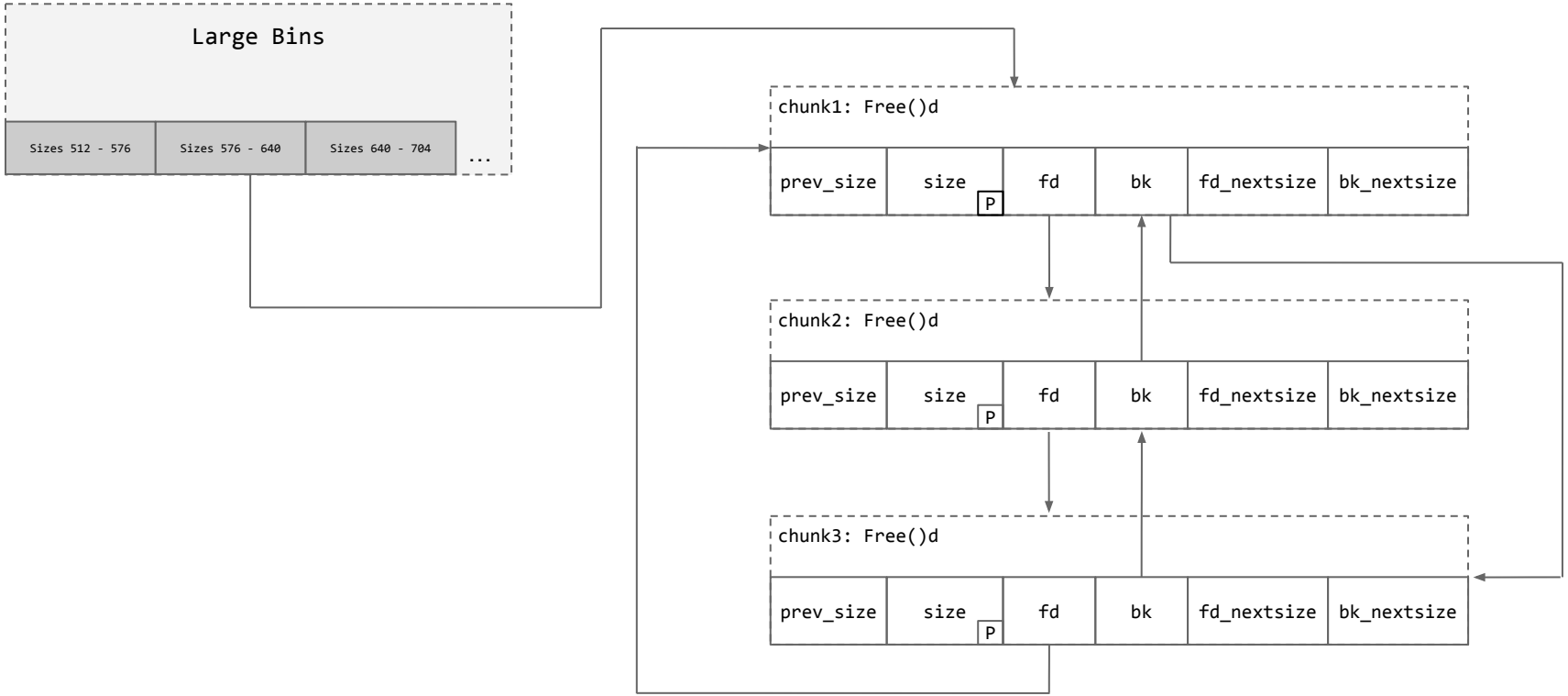
Large Bins

- Doubly linked lists
- Bins consist of a range of sizes
- Chunks in each bin are sorted by size, largest first
- `bk_nextsize` is used “jump up” in size category quickly

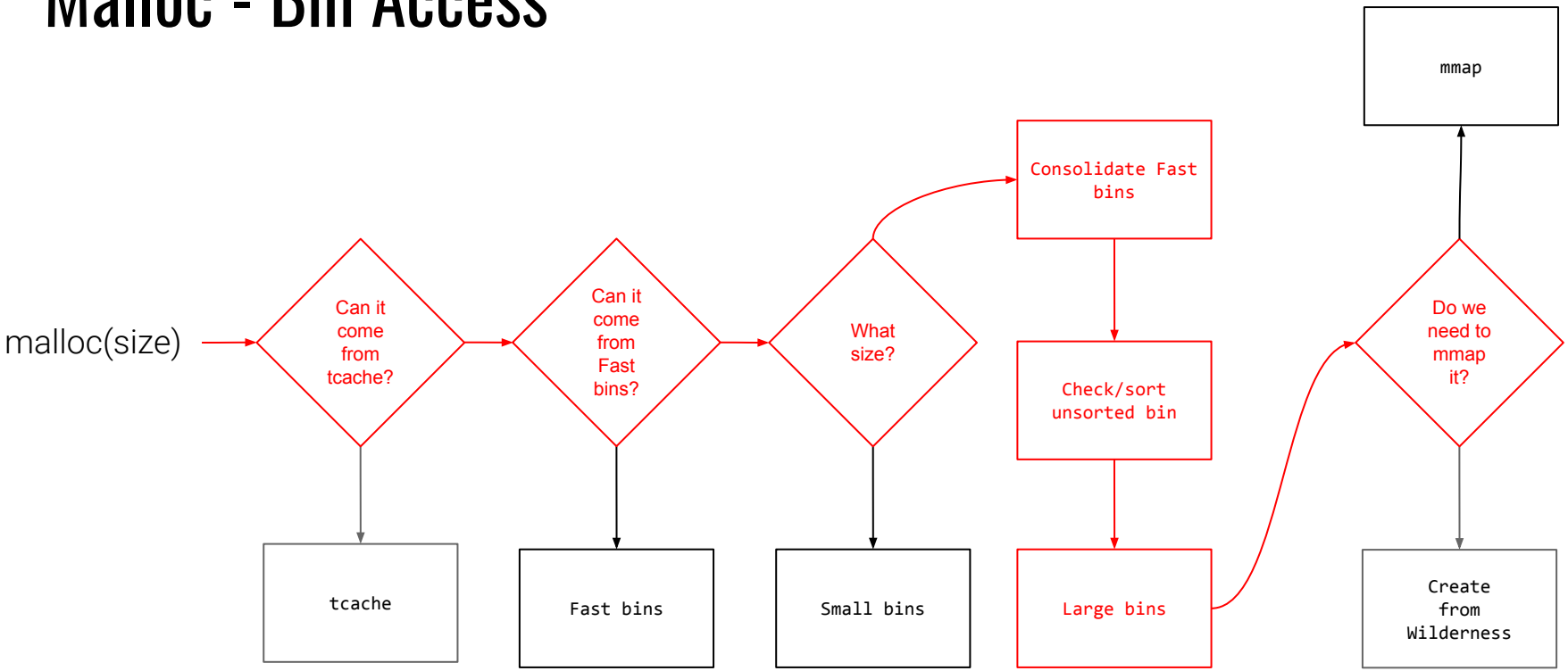


https://elixir.bootlin.com/glibc/latest/source/malloc/malloc.c#L4169

Large Bins


Large Bins
Sizes 512 - 576
Sizes 576 - 640
Sizes 640 - 704
...
chunk1: Free()d
prev_size
size
P
fd
bk
fd_nextsize
bk_nextsize
chunk2: Free()d
prev_size
size
P
fd
bk
fd_nextsize
bk_nextsize
chunk3: Free()d
prev_size
size
P
fd
bk
fd_nextsize
bk_nextsize

Malloc - Bin Access



What if a satisfactory chunk still is not found?

- A chunk may be formed from the wilderness
- An extremely large chunk may be be created via `mmap`

https://elixir.bootlin.com/glibc/latest/source/malloc/malloc.c#L4339

The M Bit

For extremely large calls to `malloc` the dynamic allocator may decide to mmap a dedicated region of memory for the allocation.

The `M` bit in the size field notes an allocation was created via `mmap`.