PartitionAlloc Design

Zia

What is PartitionAlloc

- PA is an allocator developed by google w/ security and performance on browser platforms in mind.
- Used in:
    - Chromium's rendering engine 'Blink' (Originally)
    - Rest of Chromium (Later)
    - Edge
    - Opera
    - Parts of V8

Purpose

- A) Unify memory allocation system across platforms (Windows, Android, Linux, ...)
- B) Target the 'lowest memory footprint' (Reduced fragmentation, memory decommitment, isolation)
- C) Optimize performance and memory usage of Chrome on the client side instead of server.

Glossary

- Partition: A heap that is separated from other partitions and from non-PA memory. Each partition holds multiple buckets.
- Buckets: A collection of memory regions in a partition that can hold similarly sized objects.
- Slot: An indivisible allocation unit; backs buckets.
- Slot Span: A number of contiguous same-size slots. Will always be a multiple of partition page size.
- Extent: Run of consecutive super pages
- Pool: Large contiguous virtual address region

How it works

Hooks calls to “malloc & friends” symbol definitions and overwriting their backend.

This is called the ‘Unified Allocator Shim’.

```
Overview of the unified allocator shim The allocator shim consists of three stages

+-------------------------+    +-----------------------+    +----------------+
|     malloc & friends    | -> |       shim layer      | -> |   Routing to   |
|    symbols definition   |    |     implementation    |    |    allocator   |
+-------------------------+    +-----------------------+    +----------------+
| - libc symbols (malloc, |    | - Security checks     |    | - glibc        |
|   calloc, free, ...)    |    | - Chain of dispatchers|    | - Android      |
| - C++ symbols (operator |    |   that can intercept  |    |   bionic       |
|   new, delete, ...)     |    |   and override        |    | - WinHeap      |
| - glibc weak symbols    |    |   allocations         |    | - Partition    |
|   (__libc_malloc, ...)  |    +-----------------------+    |   Alloc        |
+-------------------------+                                 +----------------+
```

What is it?

- Core concepts:
  - PartitionAlloc is built around partitions (super pages), slot spans, slots, and size buckets
- Partitions:
  - 2 MiB–aligned superpages, partially commit-able, with guard pages at the start and end
- Metadata placement:
  - Slot-span metadata lives in a reserved region within the first guard page
- Slot spans & buckets:
  - Each slot span holds same-sized slots; allocations are size-segregated to reduce type confusion
- Address space discipline:
  - Once a region is assigned to a partition/bucket, it is never repurposed

Buckets

- Allocations are mapped to size buckets, each defining a fixed slot size
- Same-bucket allocations are served from size-segregated slot spans
- Enables:
  - Fast address → size mapping
  - Low metadata overhead
  - Improved cache locality
  - Reduced external fragmentation

Types:

- kNeutral
  - Fewer buckets (coarser granularity)
  - ↓ partially-filled slot spans
  - ↑ per-allocation internal fragmentation (larger size rounding)
- kDenser
  - ~2× number of buckets vs Neutral
  - ↓ internal fragmentation (closer size fit)
  - ↑ risk of partially-filled slot spans

16 Bytes Alignment (Typically 64-bit Systems)

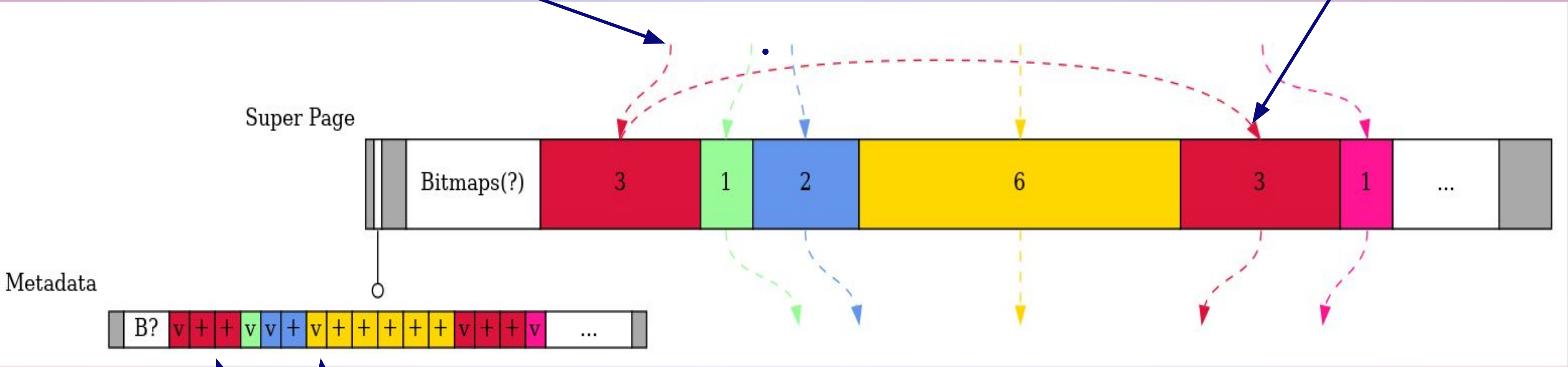
Index	Size	Bucket Distribution	Originating Formula
0	16	kNeutral and kDenser	linear [16 x 1]
1	32	kNeutral and kDenser	linear [16 x 2]
2	48	kNeutral and kDenser	linear [16 x 3]
3	64	kNeutral and kDenser	linear [16 x 4]
4	80	kNeutral and kDenser	linear [16 x 5]
5	96	kNeutral and kDenser	linear [16 x 6]
6	112	kNeutral and kDenser	linear [16 x 7]
7	128	kNeutral and kDenser	linear [16 x 8] yet exponential [2^7 x (1 + 0)]
8	144	kNeutral and kDenser	linear [16 x 9] yet exponential [2^7 x (1 + ⅛)]
9	160	kNeutral and kDenser	linear [16 x 10] yet exponential [2^7 x (1 + ¼)]
10	176	kNeutral and kDenser	linear [16 x 11] yet exponential [2^7 x (1 + ⅜)]
11	192	kNeutral and kDenser	linear [16 x 12] yet exponential [2^7 x (1 + ½)]
12	208	kNeutral and kDenser	linear [16 x 13] yet exponential [2^7 x (1 + ⅝)]

	Order-Index 0	Order-Index 1	Order-Index 2	Order-Index 3	Order-Index 4	Order-Index 5	Order-Index 6	Order-Index 7
Order 8 (2^7)	121-128	129-144	145-160	161-176	177-192	193-208	209-224	225-240
Order 9 (2^8)	241-256	257-288	289-320	321-352	353-384	385-416	417-448	449-480
Order 10 (2^9)	481-512	513-576	577-640	641-704	705-768	769-832	833-896	897-960

Slot span A of size 3 belonging to bucket X

Bucket X
Size = 256 Bytes

Slot span B of size 3 belonging to bucket X



V’s = SlotSpanMetadata
+’s = SubsequentPageMetadata

SuperPages & Metadata

Heavy metadata usage: PA maintains extensive metadata to improve performance and security

Central metadata page: A small system page inside the leading guard page stores core heap metadata

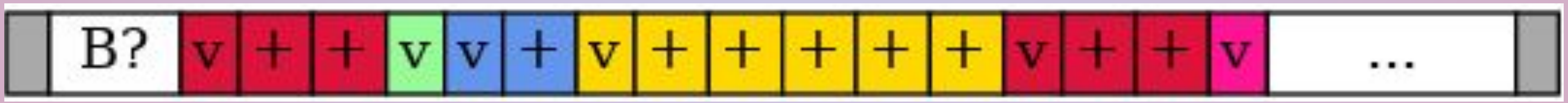
PartitionPageMetadata: Each partition page is tracked by a **32-byte struct** in this system page

SlotSpanMetadata (v):

- One per slot span, located at the start of the span
- Tracks slot size, freelist head, alloc/free counts
- Records span state (empty / active / full)
- Points back to the owning size bucket

SubsequentPageMetadata (+):

- One per partition page within a slot span
- Tracks owning slot span, commit state, and page type (guard, empty, etc.)



Lists

There exists an active, empty, and decommitted list (no full).

Free Lists

- Track free slots *within a slot span*
- Singly-linked, stored inside freed slots
- One freelist per slot span (per bucket)
- Performance-critical, security-hardened

Hardening

- Encoded freelist pointers
- Shadow pointer verification
- Fail-fast on corruption

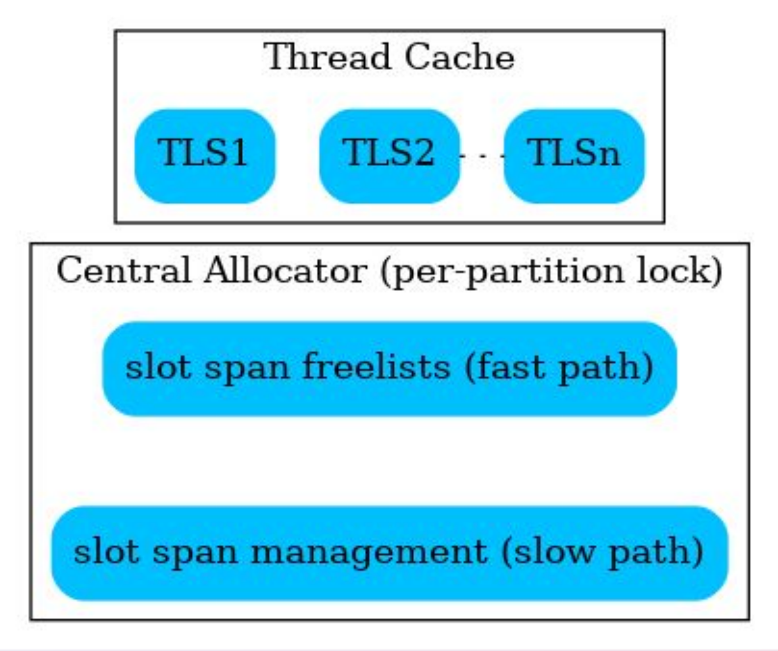
```
Transform(ptr offset) :
    if BIG_ENDIAN:
        uintptr_t transformed = ~offset
    else:
        uintptr_t transformed = ReverseBytes(offset)

    return transformed
```

Decommitment

- When PA still owns the vmaddr space, but the space isn't backed by physical memory

Thread-local caching

- Layer 1: lockless, per-thread cache → improves cache locality & performance
- Central allocator: slab-based, per-partition lock → memory efficiency & safety
- If thread cache is empty, allocate from slot span's freelist; if that's empty, create a new span or request from OS.
- Tuned for minimal fast-path operations → fewer locks, cache-line fetches, branches



Pointer Compression

```
// Example: heap base: 0x4b0'ffffffff
//  - g_base: 0x4b3'ffffffff (lower 34 bits set)
//  - normal pointer: 0x4b2'a08b6480
//    - compression:
//      - shift right by 3:         0x96'54116c90
//      - truncate:                    0x54116c90
//      - mark MSB:                    0xd4116c90
//    - decompression:
//      - sign-extend:        0xffffffff'd4116c90
//      - shift left by 3:    0xfffffffe'a08b6480
//      - 'and' with g_base:  0x000004b2'a08b6480
```

Why V8 does and doesn't use it

Historically (pre-sandbox):

- V8 largely avoided PartitionAlloc
- Used custom allocators (e.g. Zone, PagedSpace, NewSpace, OldSpace)
- Tight control over GC layout, object lifetimes, and pointer compression
- PA was seen as unnecessary overhead for a VM with specialized needs

Shift with the V8 Sandbox:

- Security model changed: memory safety > allocator autonomy
- Sandbox requires strong spatial isolation and robust metadata
- PartitionAlloc adopted as the backing allocator for sandboxed memory

Today:

- V8 still keeps its logical heap & GC, but physical memory comes from PA
- PA provides:
    - Guarded, size-segregated allocations
    - Precise bounds + metadata for sandbox checks
    - Hardening primitives (GigaCage-style isolation, OOB resistance)
- Result: V8-on-PA

Thank you bye bye:

1. glossary
2. Design Docs
3. chromium blog
4. buckets
5. src: PartitionPageMetadata
6. src: SlotSpanMetadata
7. src: Base & Config
8. src: Address Transformation