Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

by

A bunch of pasty faced sad sack nerds sitting in a basement want to sound cool and tough, like they've just done a tour in 'Nam. [slashdot]

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Victor van der Veen[1], *Yanick Fratantonio*[2], *Martina Lindorfer*[2], *Daniel Gruss*[3], *Clémentine Maurice*[3], *Giovanni Vigna*[2], *Herbert Bos*[1], *Kaveh Razavi*[1], *and Cristiano Giuffrida*[1]

[1]Vrije Universiteit Amsterdam, [2]UC Santa Barbara, [3]TU Graz

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Your takeaway message of today

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Your takeaway message of today

Rowhammer on ARM

Drammer:
Deterministic Rowhammer Attacks on Mobile Platforms

Your takeaway message of today

Rowhammer on ARM

Deterministic exploitation

Drammer:
Deterministic Rowhammer Attacks on Mobile Platforms

Your takeaway message of today

Rowhammer on ARM

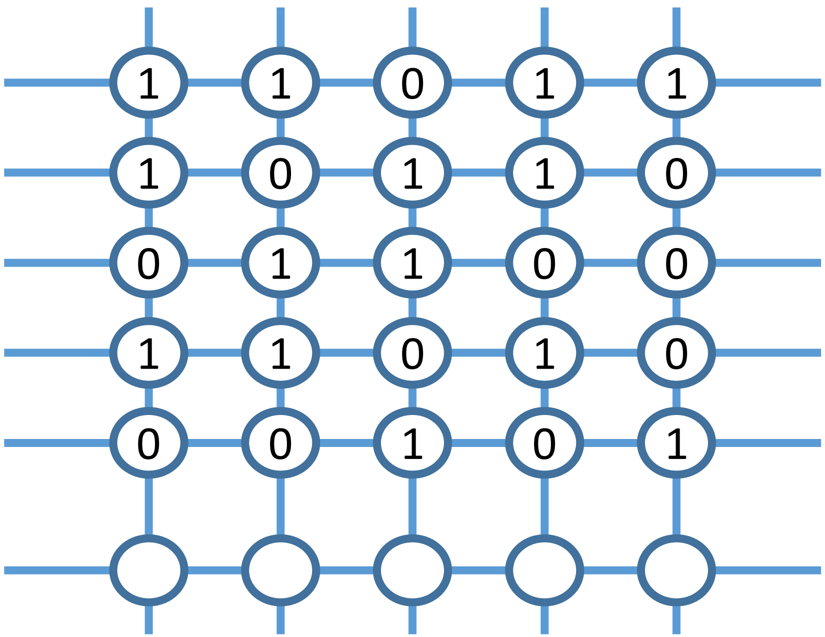
Deterministic exploitation

Works on a Google Pixel

Drammer: Deterministic **Rowhammer** Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors



Aggressor row

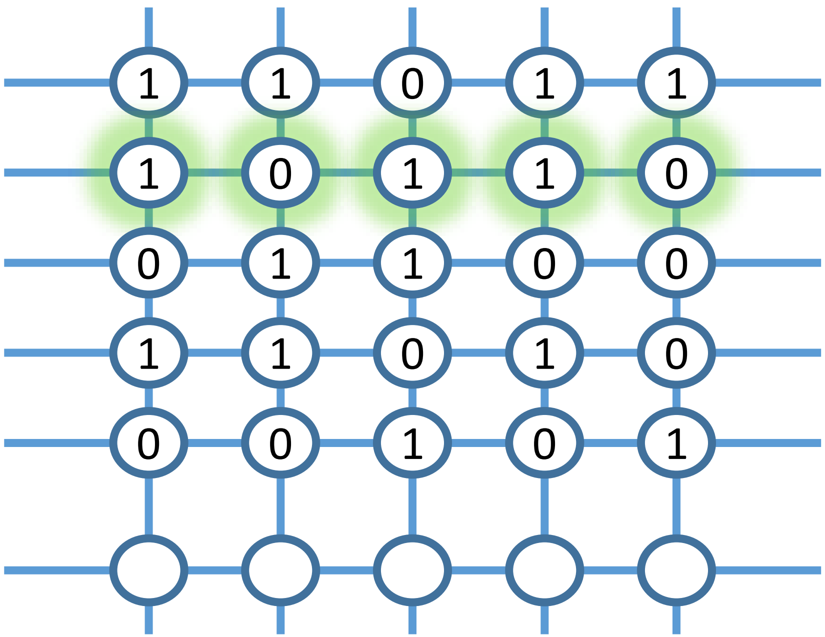
Victim row

Aggressor row

Drammer: Deterministic **Rowhammer** Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors



Aggressor row

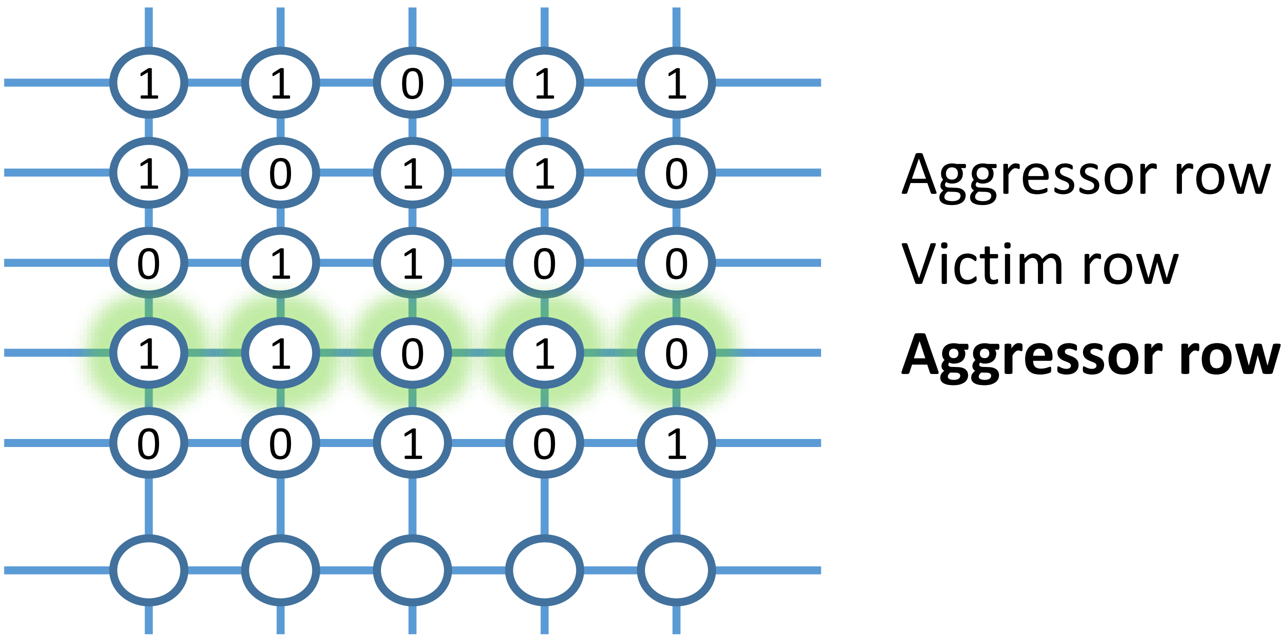
Victim row

Aggressor row

Drammer: Deterministic **Rowhammer** Attacks on Mobile Platforms

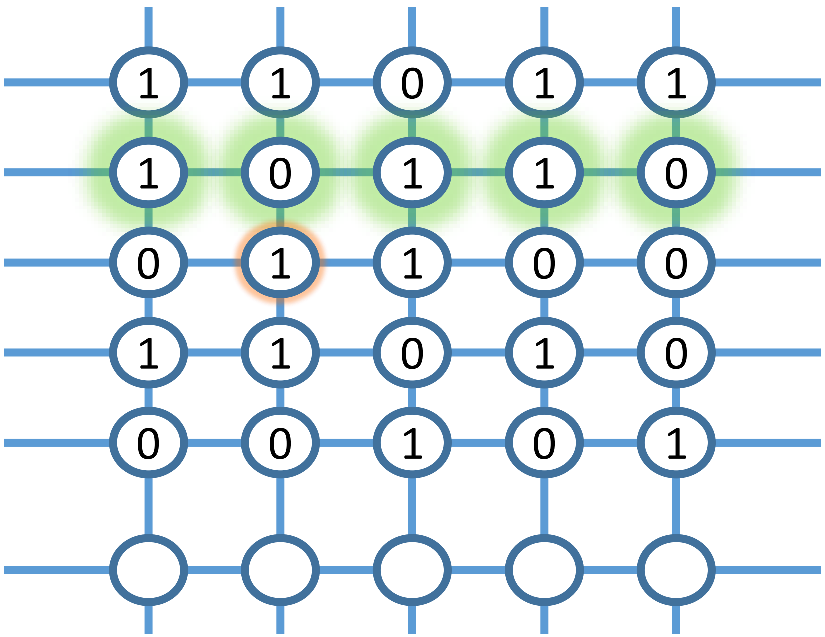
Flipping bits in memory

DRAM hardware glitch causing disturbance errors



Flipping bits in memory

DRAM hardware glitch causing disturbance errors



Aggressor row

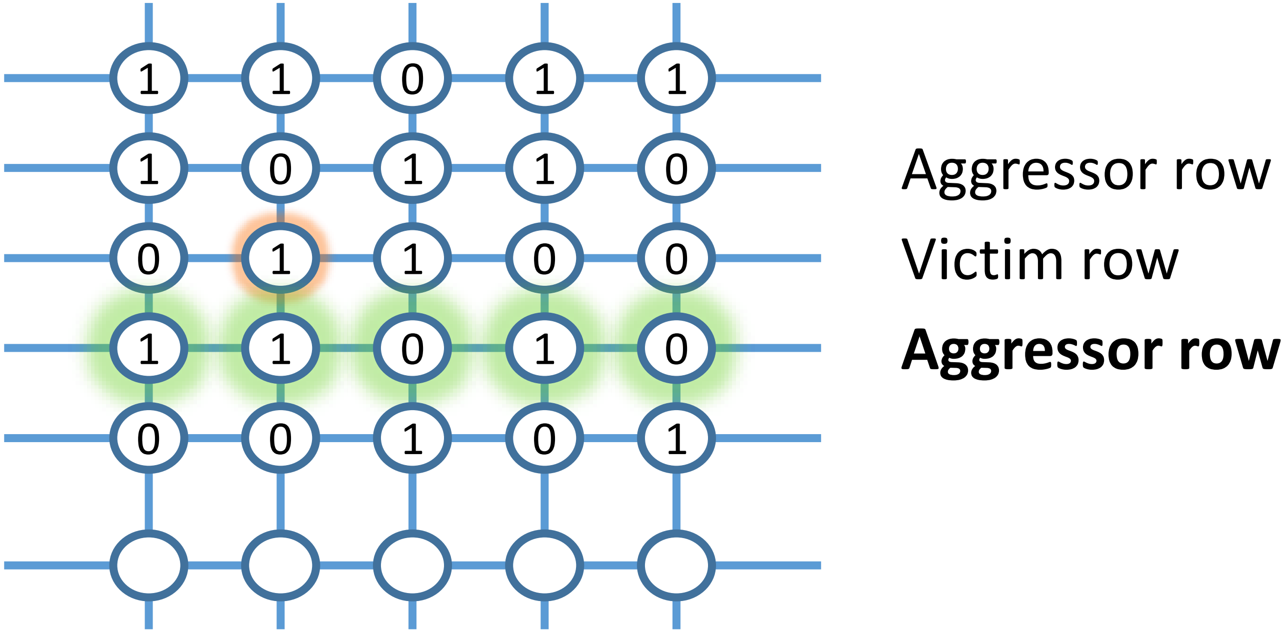
Victim row

Aggressor row

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Flipping bits in memory

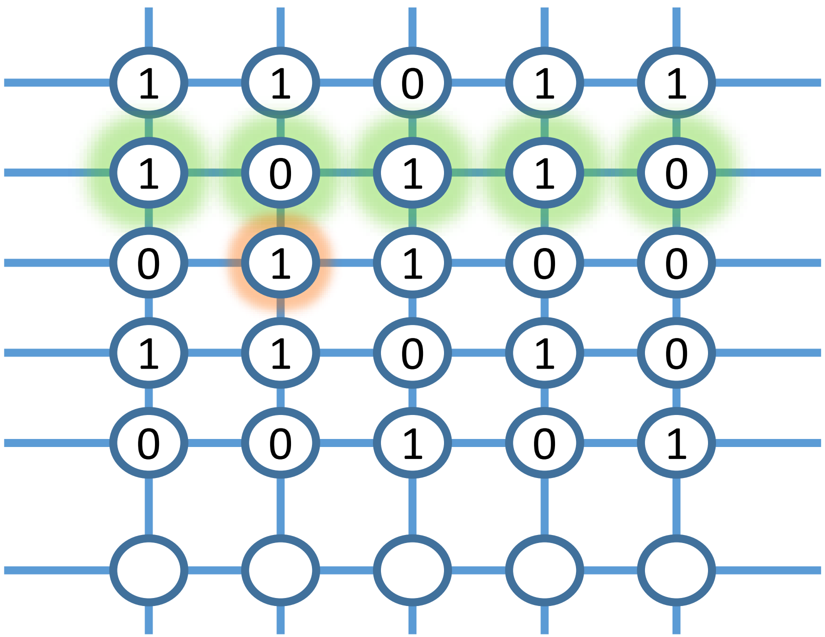
DRAM hardware glitch causing disturbance errors



Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors



Aggressor row

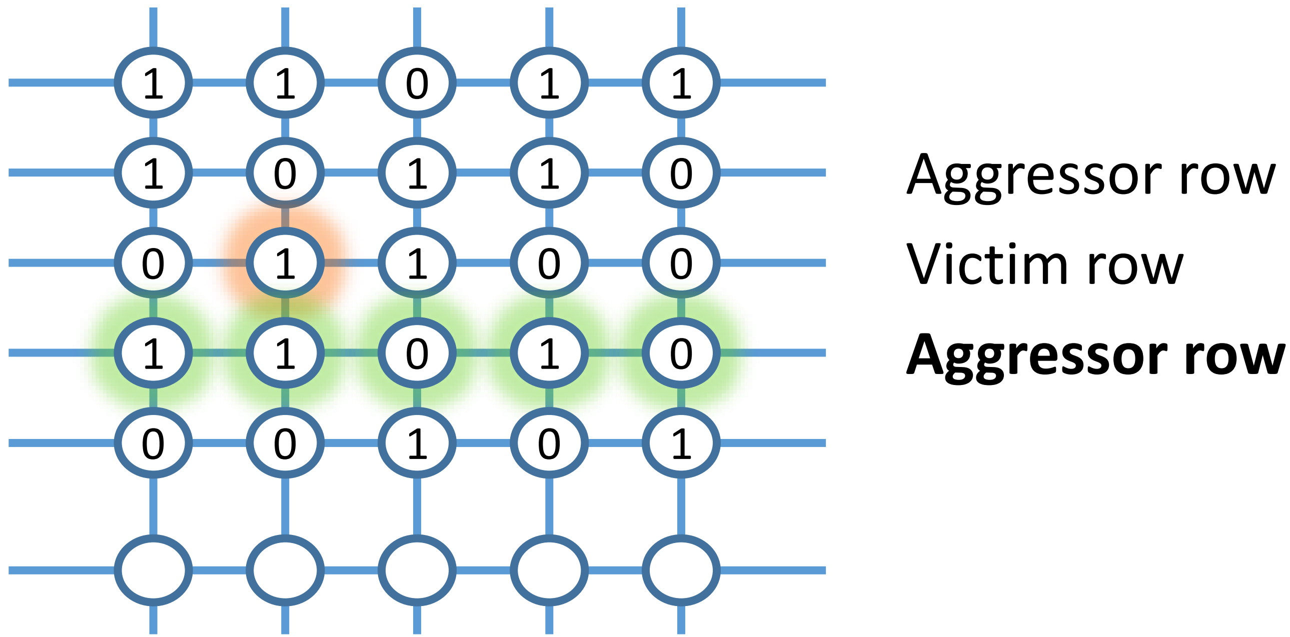
Victim row

Aggressor row

Drammer: Deterministic **Rowhammer** Attacks on Mobile Platforms

Flipping bits in memory

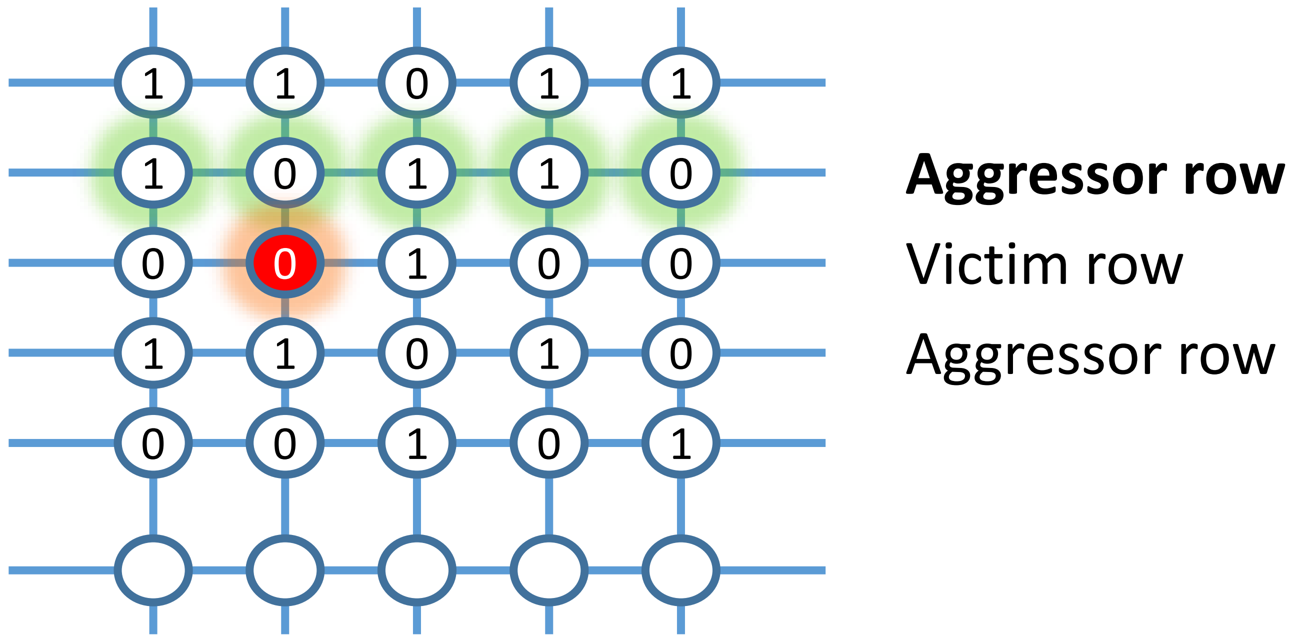
DRAM hardware glitch causing disturbance errors



Drammer: Deterministic **Rowhammer** Attacks on Mobile Platforms

Flipping bits in memory

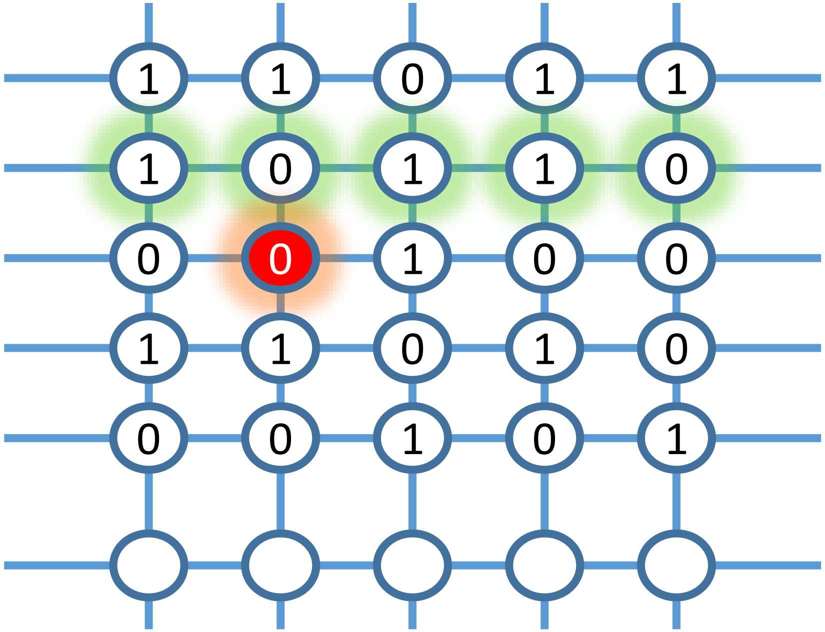
DRAM hardware glitch causing disturbance errors



Drammer: Deterministic **Rowhammer** Attacks on Mobile Platforms

Flipping bits in memory

DRAM hardware glitch causing disturbance errors



Aggressor row

Victim row

Aggressor row

- Not every bit may flip
- Once a bit flips, we can reproduce it

Overview

1. Memory Templating

   Scan memory for useful bit flips

Overview

1. Memory Templating

   Scan memory for useful bit flips

Overview

1. ## Memory Templating
   Scan memory for useful bit flips
2. ## Land sensitive data
   Store a crucial data structure on a vulnerable page

Overview

1. **Memory Templating**
   Scan memory for useful bit flips
2. **Land sensitive data**
   Store a crucial data structure on a vulnerable page
3. **Reproduce the bit flip**
   Modify the data structure and get root acces

Overview

1. Memory Templating

   Scan memory for useful bit flips

Templating

Uncached memory access

- `clflush`
- cache eviction
- non-temporal access instructions

Determining the physical addresses aggressor/victim rows

- `/proc/self/pagemap`
- 2MB huge pages (relative)

Templating

Uncached memory access

- **`clflush`**
- cache eviction
- non-temporal access instructions

Determining the physical addresses aggressor/victim rows

- **`/proc/self/pagemap`**
- 2MB huge pages (relative)

But does it work on ARM?

Templating

Uncached memory access

- **`clflush`**
- cache eviction
- non-temporal access instructions

Determining the physical addresses aggressor/victim rows

- **`/proc/self/pagemap`**
- 2MB huge pages (relative)

But does it work on ARM?

Nope

Templating

Uncached memory access

- **`clflush`**
- cache eviction
- non-temporal access instructions

Determining the physical addresses aggressor/victim rows

- **`/proc/self/pagemap`**
- 2MB huge pages (relative)

But does it work on ARM?

None of them

Templating

Uncached memory access

- **`clflush`**
- cache eviction
- non-temporal access instructions

Determining the physical addresses aggressor/victim rows

- **`/proc/self/pagemap`**
- 2MB huge pages (relative)

But does it work on ARM?

(and we tried)

Templating on ARM

DMA

Direct Memory Access

Android's DMA memory allocator provides everything we need:

- Uncached memory (no **`clflush`** required)
- Physically contiguous memory

Templating on ARM

DMA

Direct Memory Access

Android's DMA memory allocator provides everything we need:

- Uncached memory (no **`clflush`** required)
- Physically contiguous memory

Physical memory:

Templating on ARM

DMA

Direct Memory Access

Android’s DMA memory allocator provides everything we need:

- Uncached memory (no **`clflush`** required)
- Physically contiguous memory

Physical memory:

DMA ALLOCATED CHUNK

Templating on ARM

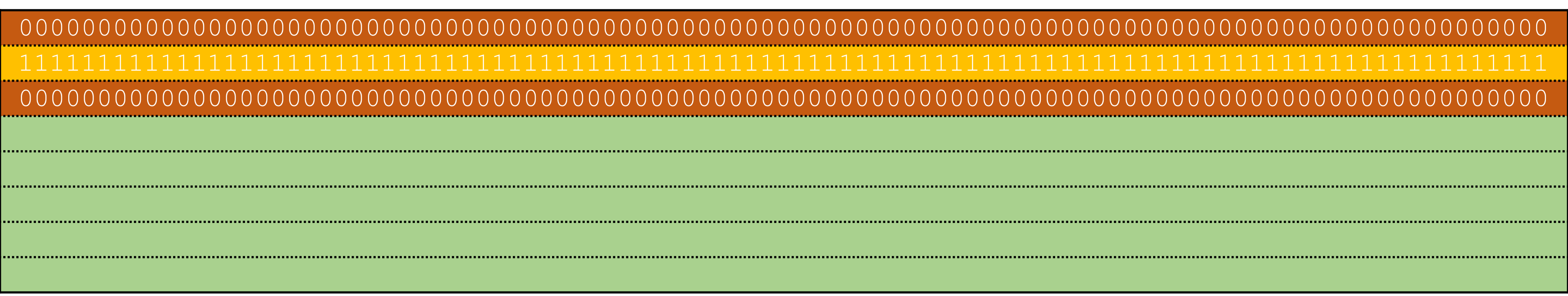
DMA

Direct Memory Access

Android's DMA memory allocator provides everything we need:

- Uncached memory (no **`clflush`** required)
- Physically contiguous memory

Physical memory:



Templating on ARM

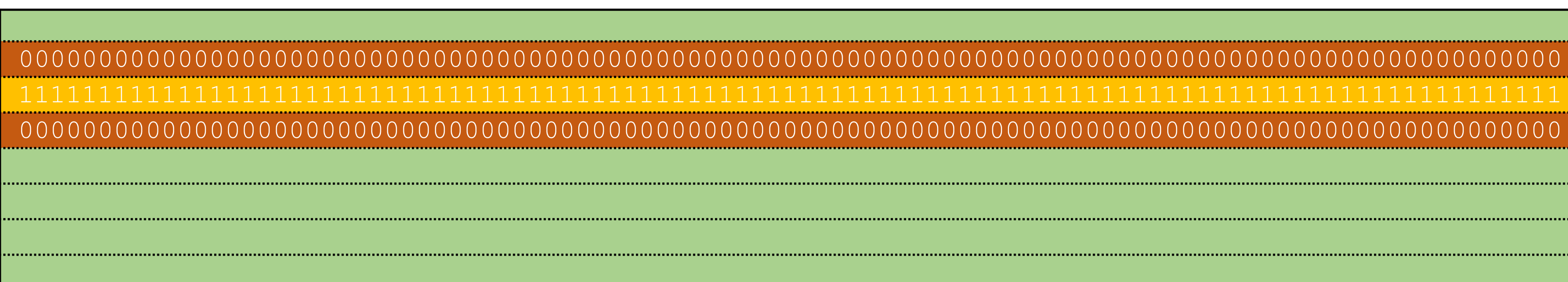
DMA

Direct Memory Access

Android’s DMA memory allocator provides everything we need:

- Uncached memory (no `clflush` required)
- Physically contiguous memory

Physical memory:



Templating on ARM

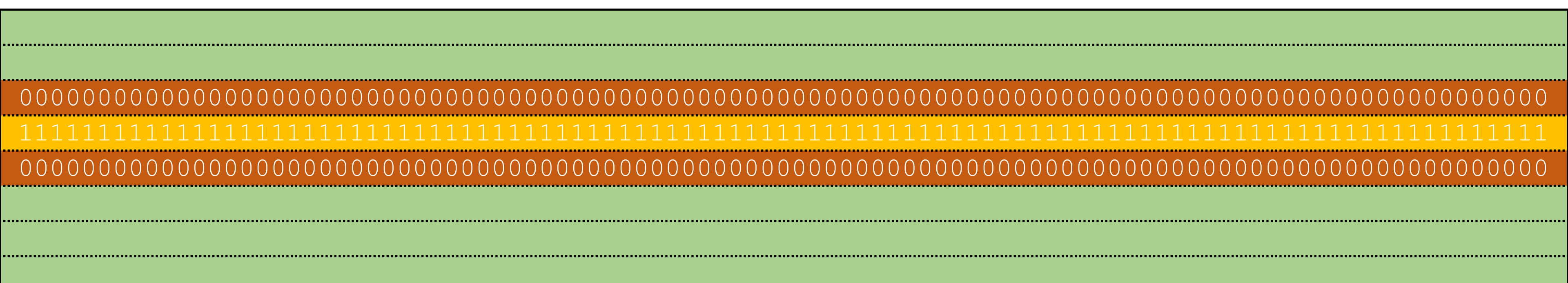
DMA

Direct Memory Access

Android's DMA memory allocator provides everything we need:

- Uncached memory (no **`clflush`** required)
- Physically contiguous memory

Physical memory:



Templating on ARM

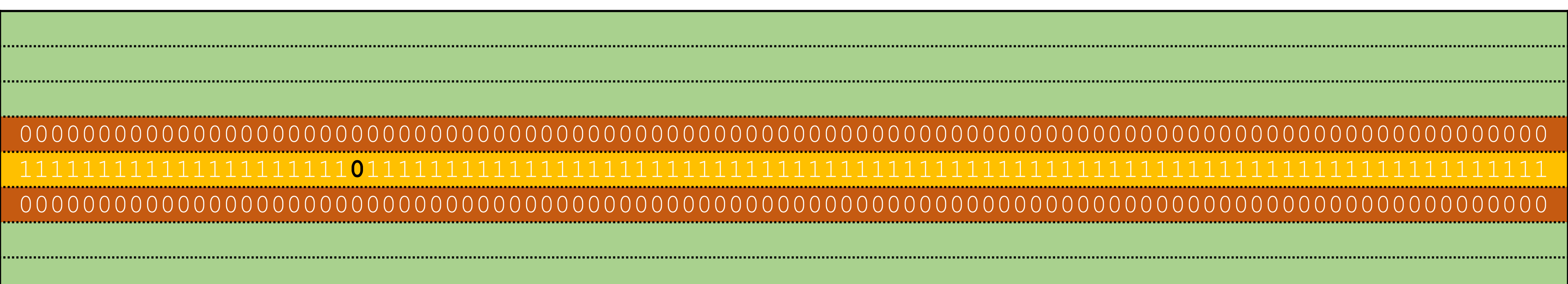
DMA

Direct Memory Access

Android's DMA memory allocator provides everything we need:

- Uncached memory (no **`clflush`** required)
- Physically contiguous memory

Physical memory:



Templating on ARM

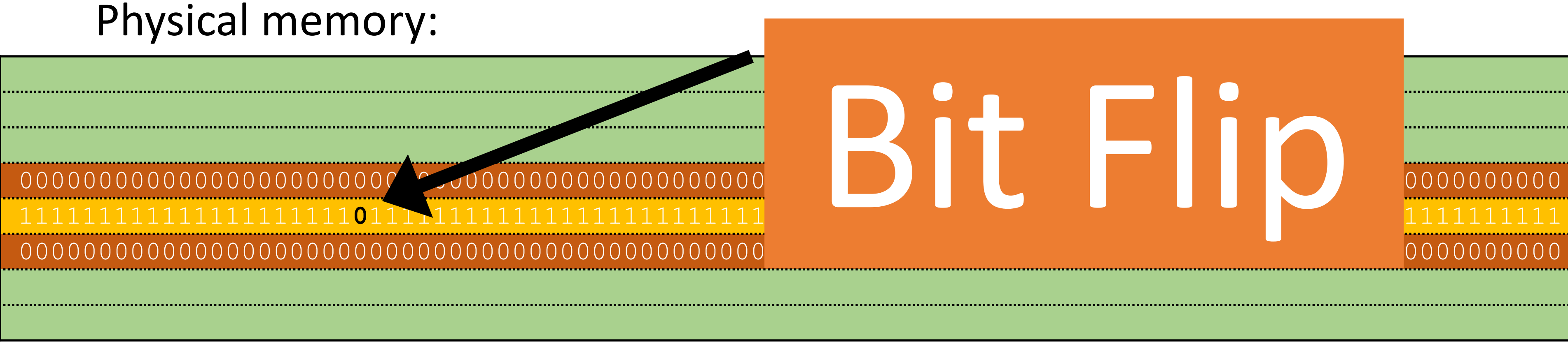
DMA

Direct Memory Access

Android's DMA memory allocator provides everything we need:

- Uncached memory (no **`clflush`** required)
- Physically contiguous memory

Physical memory:



Overview

1. Memory Templating
   Scan memory for useful bit flips

2. Land sensitive data
   Store a crucial data structure on a vulnerable page

3. Reproduce the bit flip
   Modify the data structure and get root acces

Overview

1. Memory Templating
   Scan memory for useful bit flips

2. Land sensitive data
   Store a crucial data structure on a vulnerable page

Overview

1. Memory Templating
   Scan memory for useful bit flips
2. Land a Page Table
   Store a page table on a vulnerable page

But why?

Page Tables

Mapping virtual addresses to physical addresses

Page Tables

Mapping virtual addresses to physical addresses

Example lookup for input virtual address `0xb6a5717f`

1 0 1 1	0 1 1 0	1 0 1 0	0 1 0 1	0 1 1 1	0 0 0 1	0 1 1 1	1 1 1 1

Page Tables

Mapping virtual addresses to physical addresses

Example lookup for input virtual address `0xb6a5717f`

1 0 1 1	**0 1 1 0**	**1 0 1 0**	0 1 0 1	0 1 1 1	0 0 0 1	0 1 1 1	1 1 1 1

- Highest 12 bits: *level 1 table index* (*Translation Table Base Register*)

Page Tables

Mapping virtual addresses to physical addresses

Example lookup for input virtual address `0xb6a5717f`

1 0 1 1	0 1 1 0	1 0 1 0	**0 1 0 1**	**0 1 1 1**	0 0 0 1	0 1 1 1	1 1 1 1

- Highest 12 bits: *level 1 table index* (*Translation Table Base Register*)
- Middle 8 bits: *level 2 table index*

Page Tables

Mapping virtual addresses to physical addresses

Example lookup for input virtual address `0xb6a5717f`

1 0 1 1	0 1 1 0	1 0 1 0	0 1 0 1	0 1 1 1	0 0 0 1 0 1 1 1 1 1 1 1

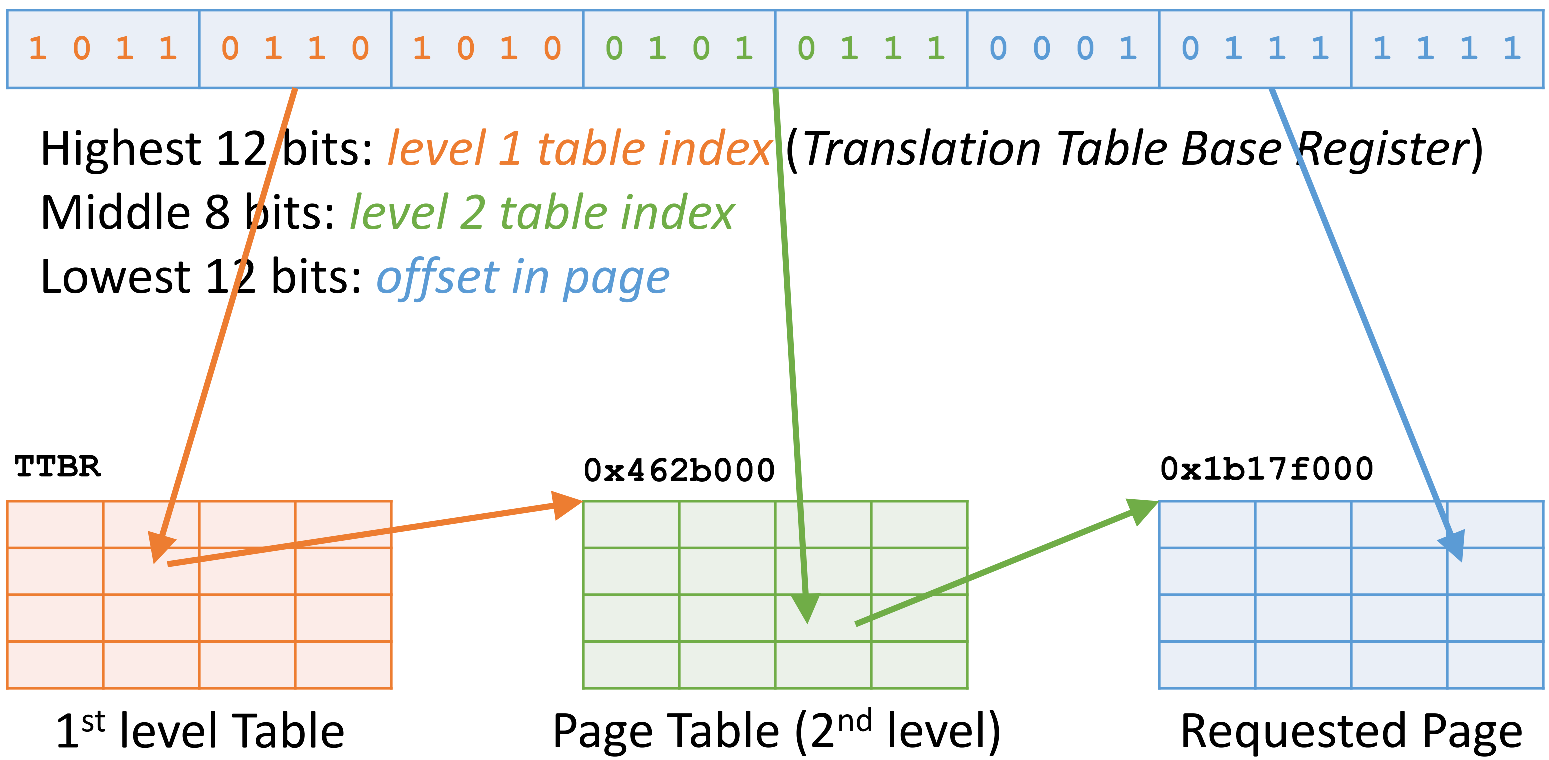
- Highest 12 bits: *level 1 table index* (*Translation Table Base Register*)
- Middle 8 bits: *level 2 table index*
- Lowest 12 bits: *offset in page*

Page Tables

Mapping virtual addresses to physical addresses

Example lookup for input virtual address `0xb6a5717f`

- Highest 12 bits: *level 1 table index* (*Translation Table Base Register*)
- Middle 8 bits: *level 2 table index*
- Lowest 12 bits: *offset in page*



Page Table Entries

Entry in the (2nd level) Page Table

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 1 1	x x x x	x x x x	x x x x

Page Table Entries

Entry in the (2nd level) Page Table

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 1 1	x x x x	x x x x	x x x x

- 12 bits of *properties*

Page Table Entries

Entry in the (2nd level) Page Table

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 1 1	x x x x	x x x x	x x x x

- 12 bits of *properties*
- 20 bits for the *page base address*

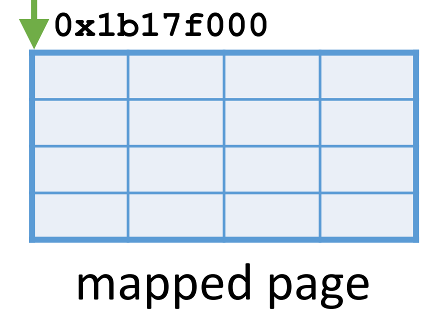
Page Table Entries

Entry in the (2nd level) Page Table

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 1 1	x x x x	x x x x	x x x x

`0x1b17f << 12`

- 12 bits of *properties*
- 20 bits for the *page base address*



What if we flip a bit in the entry?

Page Table Entries

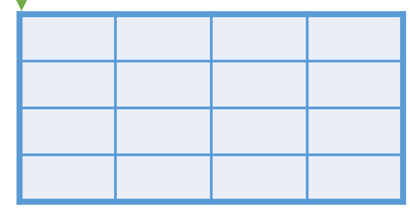
Entry in the (2^{nd} level) Page Table

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 1 1	x x x x	x x x x	x x x x

`0x1b17f << 12`

- 12 bits of *properties*
- 20 bits for the *page base address*

`0x1b17f000`



mapped page

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 1 0	x x x x	x x x x	x x x x

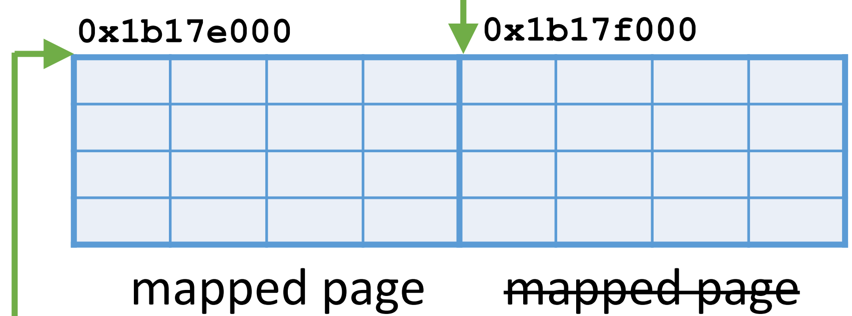
Rowhammer Attacks on
Page Table Entries

Entry in the (2^{nd} level) Page Table

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 1 1	x x x x	x x x x	x x x x

`0x1b17f << 12`

- 12 bits of *properties*
- 20 bits for the *page base address*



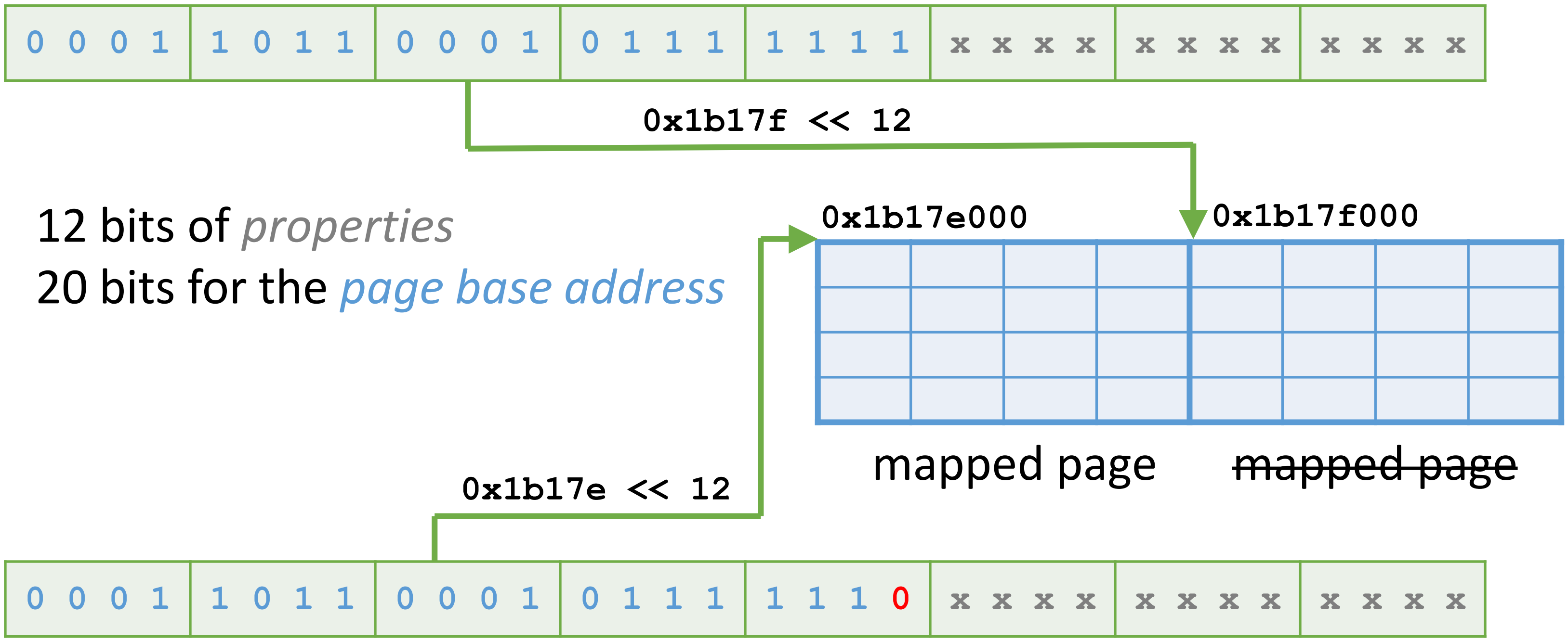
`0x1b17e << 12`

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 1 0	x x x x	x x x x	x x x x

Rowhammer Attacks on
Page Table Entries

Entry in the (2^{nd} level) Page Table

- 12 bits of *properties*
- 20 bits for the *page base address*



A 1-to-0 flip moves the mapping ‘to the left’

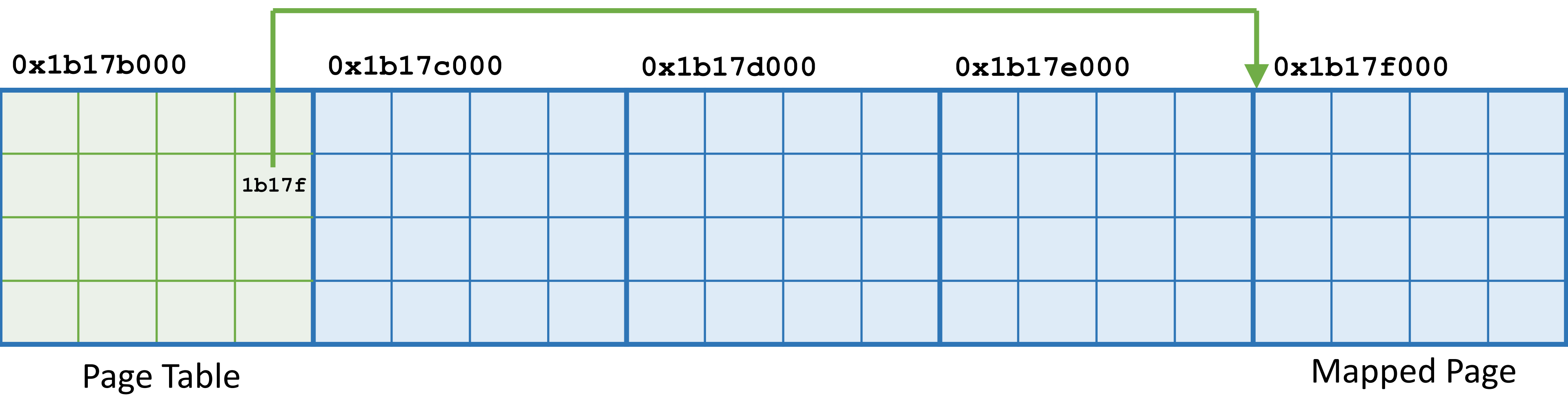
- Flip offset 0: –1 page
- Flip offset 1: –2 pages
- Flip offset 2: –4 pages
- Flip offset *n*: -2^n pages

Page Table Entries

1. Map a page 4 pages ‘away’ from its page table

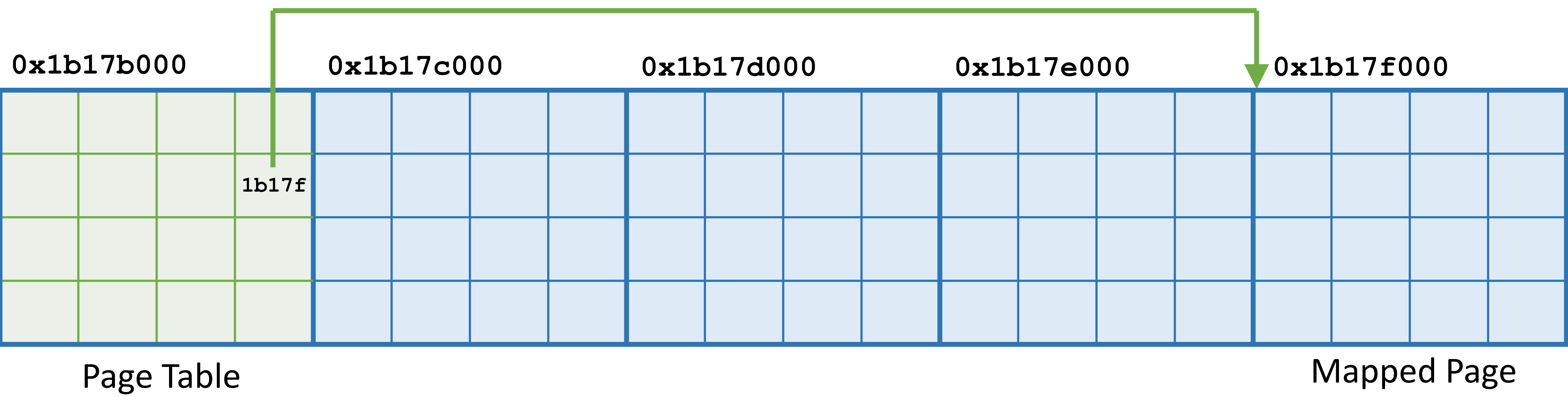
Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages 'away' from its page table



Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages 'away' from its page table



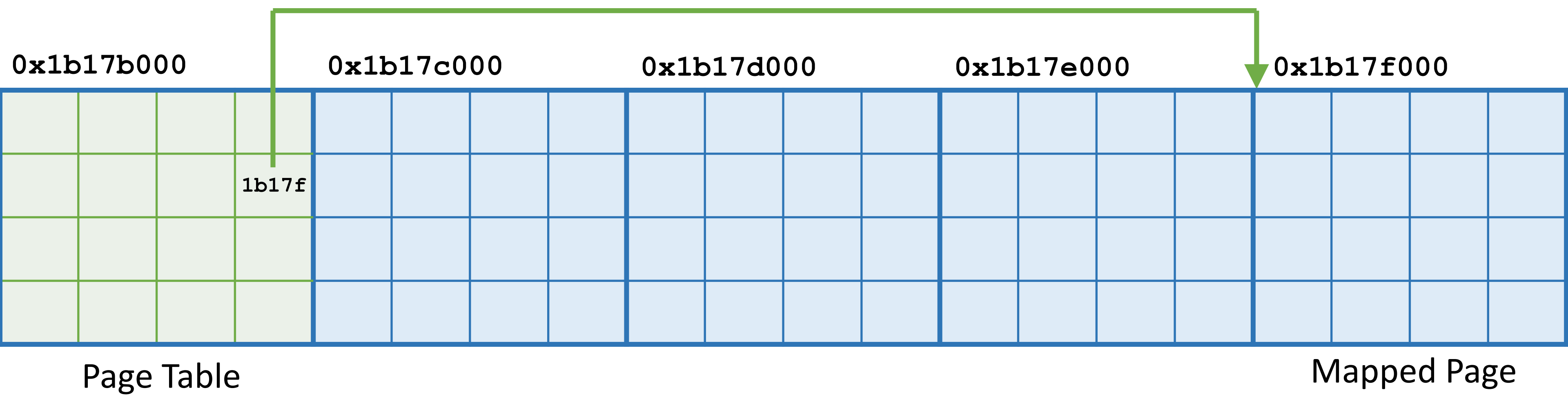
Virtual address `0xb6a57000` maps to Page Table Entry:

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 1 1	x x x x	x x x x	x x x x

which translates to physical page **`0x1b17f000`**

Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages 'away' from its page table
2. Flip bit 2 in the page table entry



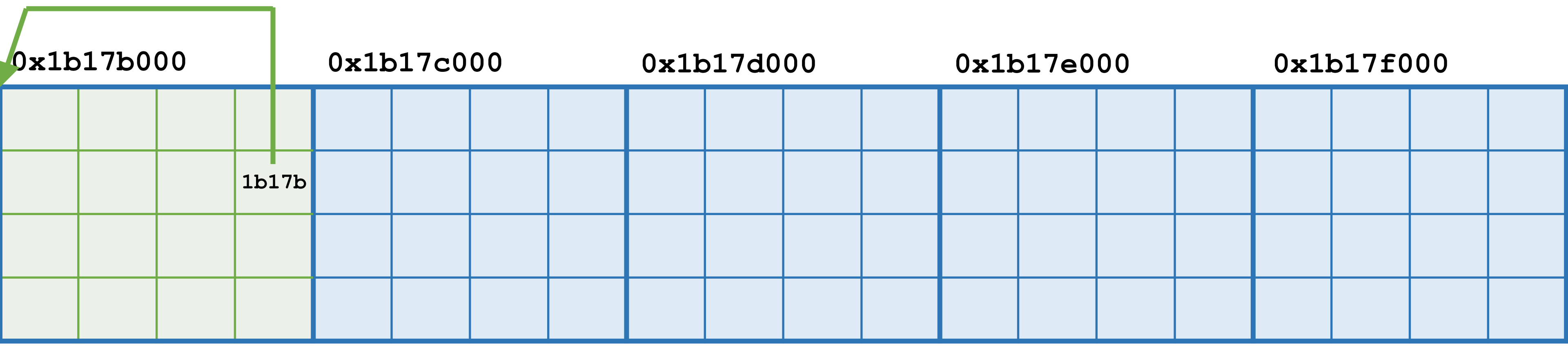
Virtual address `0xb6a57000` maps to Page Table Entry:

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 1 1	x x x x	x x x x	x x x x

which translates to physical page **0x1b17f000**

Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages 'away' from its page table
2. Flip bit 2 in the page table entry



Mapped Page Table

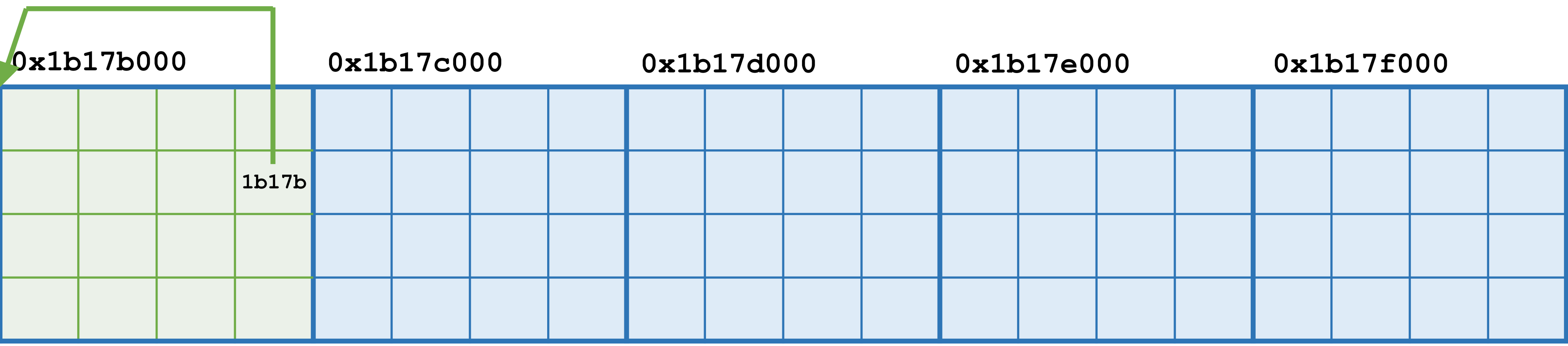
Virtual address `0xb6a57000` maps to Page Table Entry:

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 0 1	x x x x	x x x x	x x x x

which translates to physical page **0x1b17b000**

Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages 'away' from its page table
2. Flip bit 2 in the page table entry
3. Write page table entries



Mapped Page Table

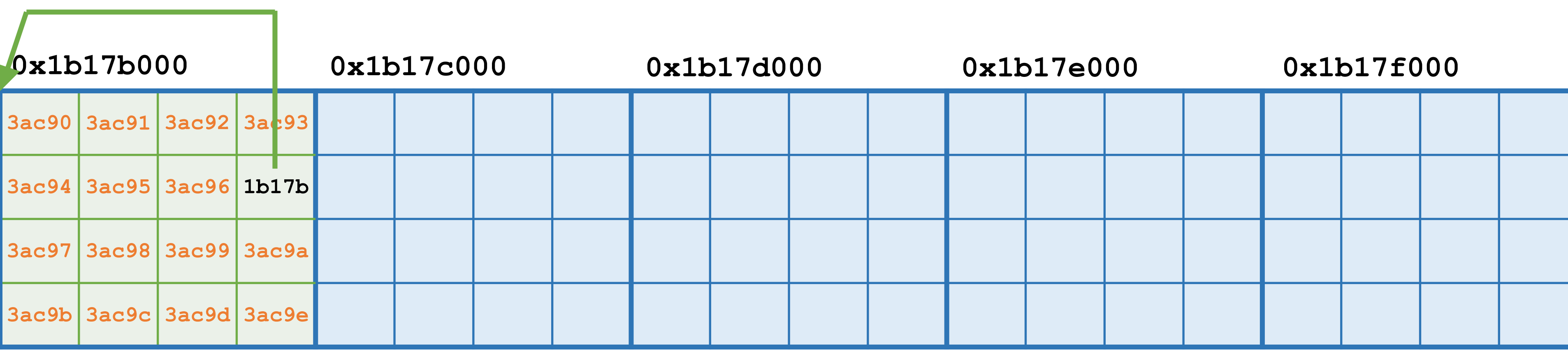
Virtual address `0xb6a57000` maps to Page Table Entry:

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 0 1	x x x x	x x x x	x x x x

which translates to physical page **0x1b17b000**

Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry
3. Write page table entries



Mapped Page Table

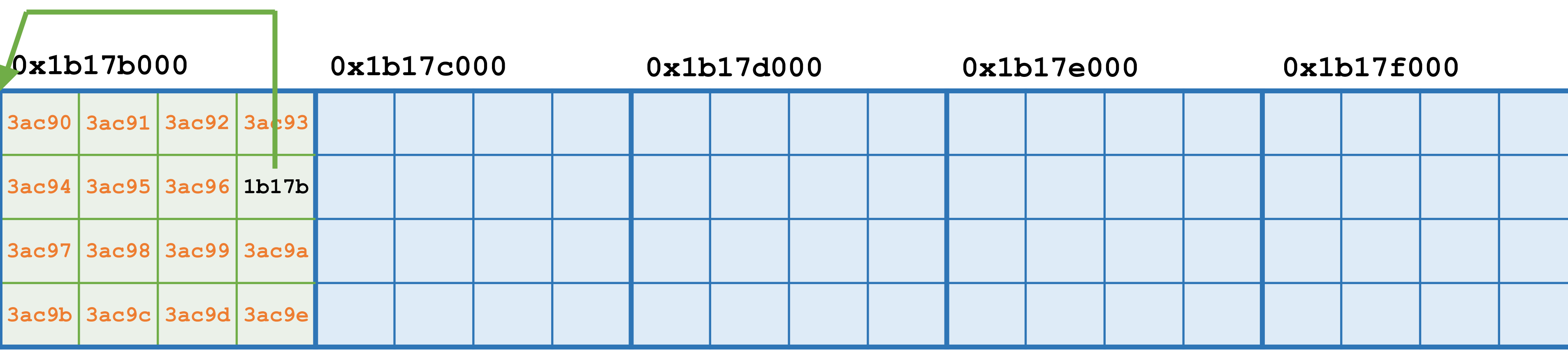
Virtual address `0xb6a57000` maps to Page Table Entry:

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 0 1	x x x x	x x x x	x x x x

which translates to physical page **0x1b17b000**

Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry
3. Write page table entries
4. Read/write kernel memory



Mapped Page Table

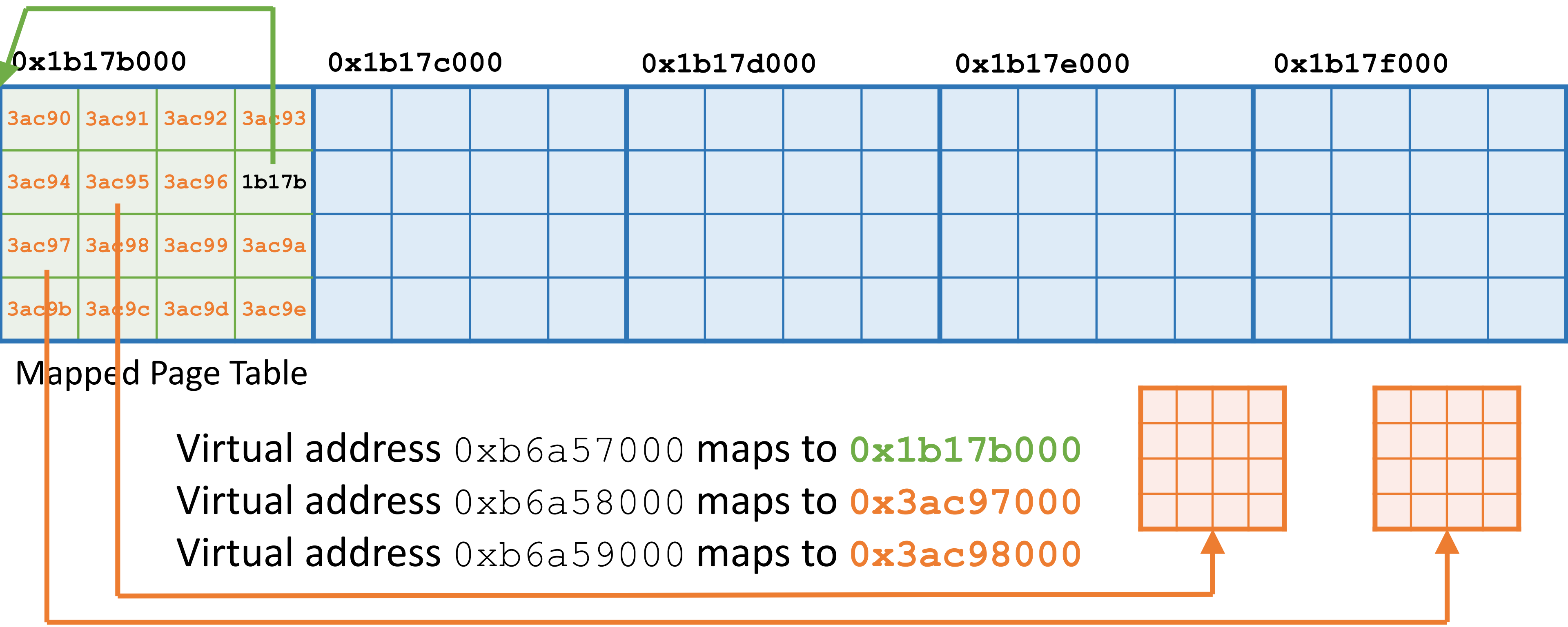
Virtual address `0xb6a57000` maps to Page Table Entry:

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 0 1	x x x x	x x x x	x x x x

which translates to physical page **0x1b17b000**

Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry
3. Write page table entries
4. Read/write kernel memory



Virtual address `0xb6a57000` maps to **0x1b17b000**

Virtual address `0xb6a58000` maps to **0x3ac97000**

Virtual address `0xb6a59000` maps to **0x3ac98000**

Overview

1. Memory Templating
   Scan memory for useful bit flips

2. Land a Page Table
   Store a page table on a vulnerable page

But how?

Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

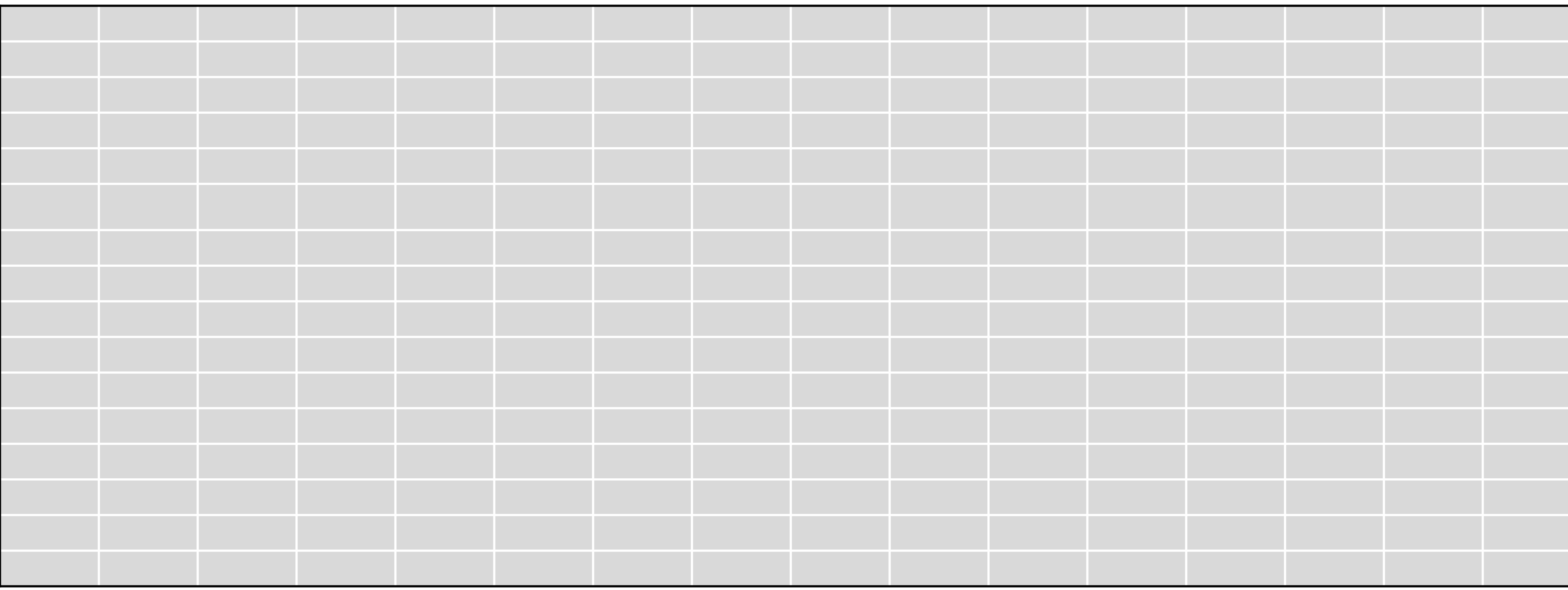
Phys Feng Shui

Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui

Physical memory:

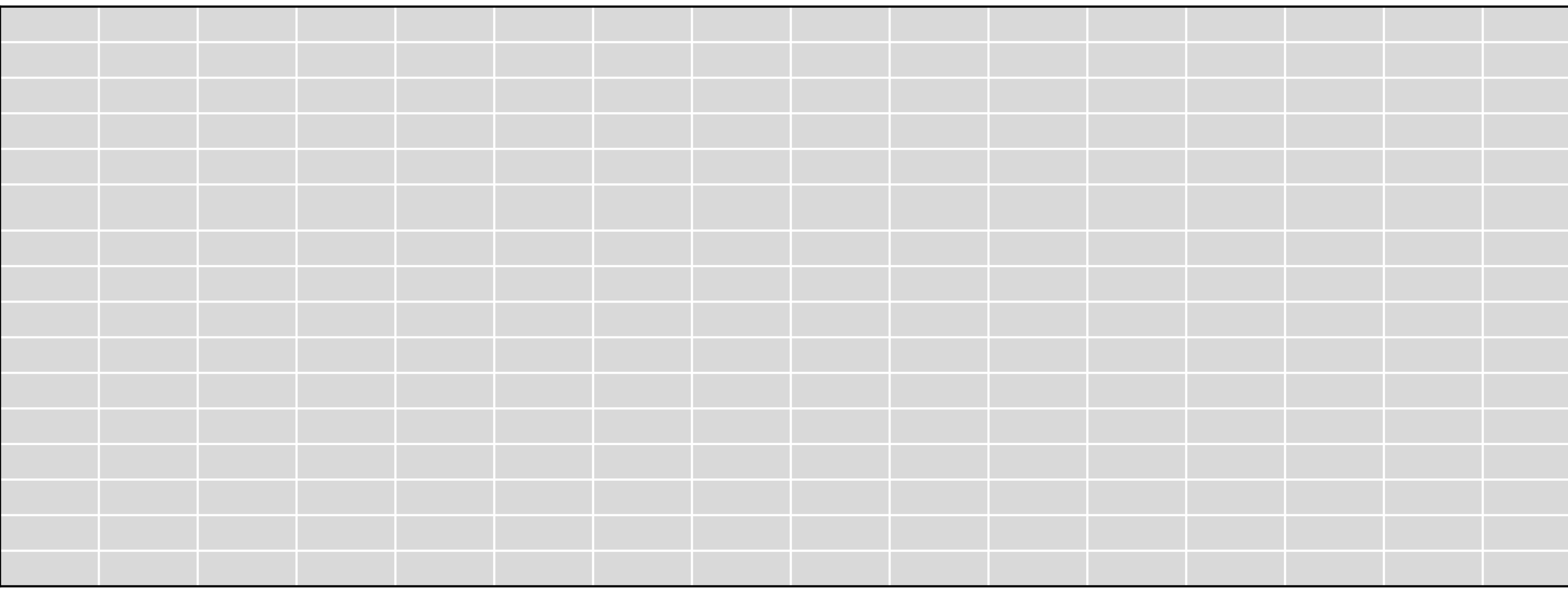


Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui

Physical memory:



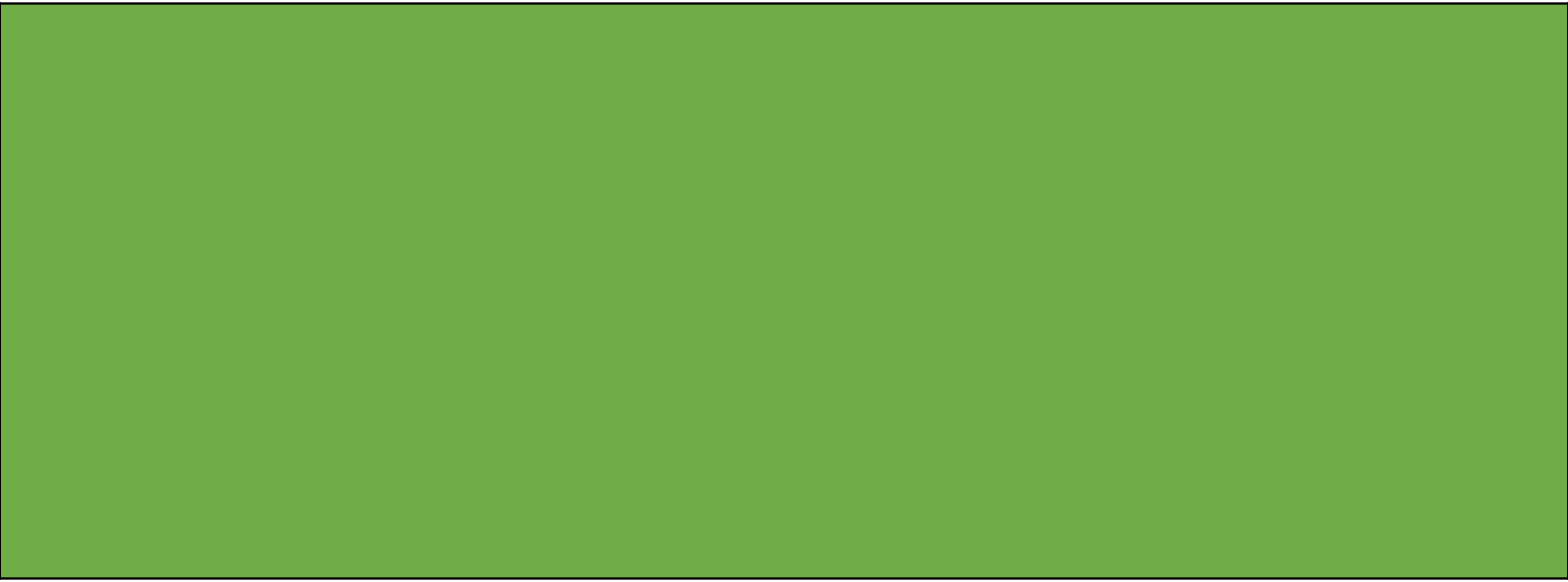
Exhaust all memory

Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui

Physical memory:



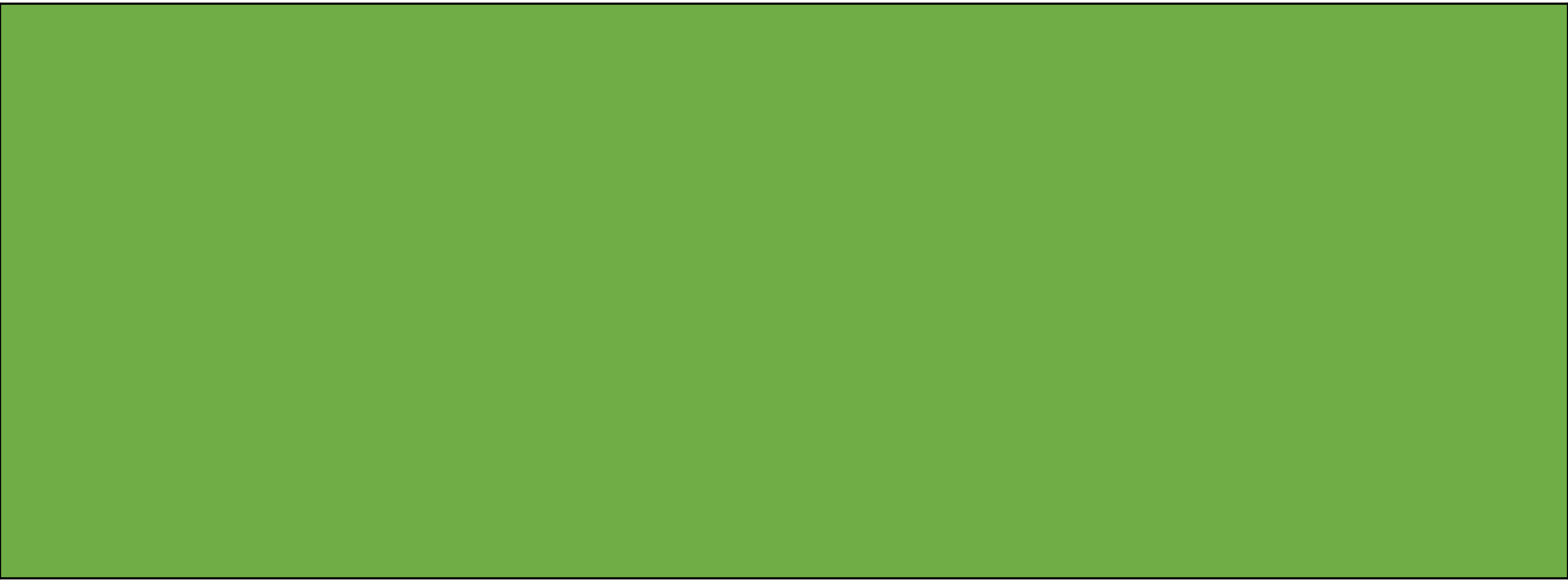
Exhaust all memory

Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui

Physical memory:



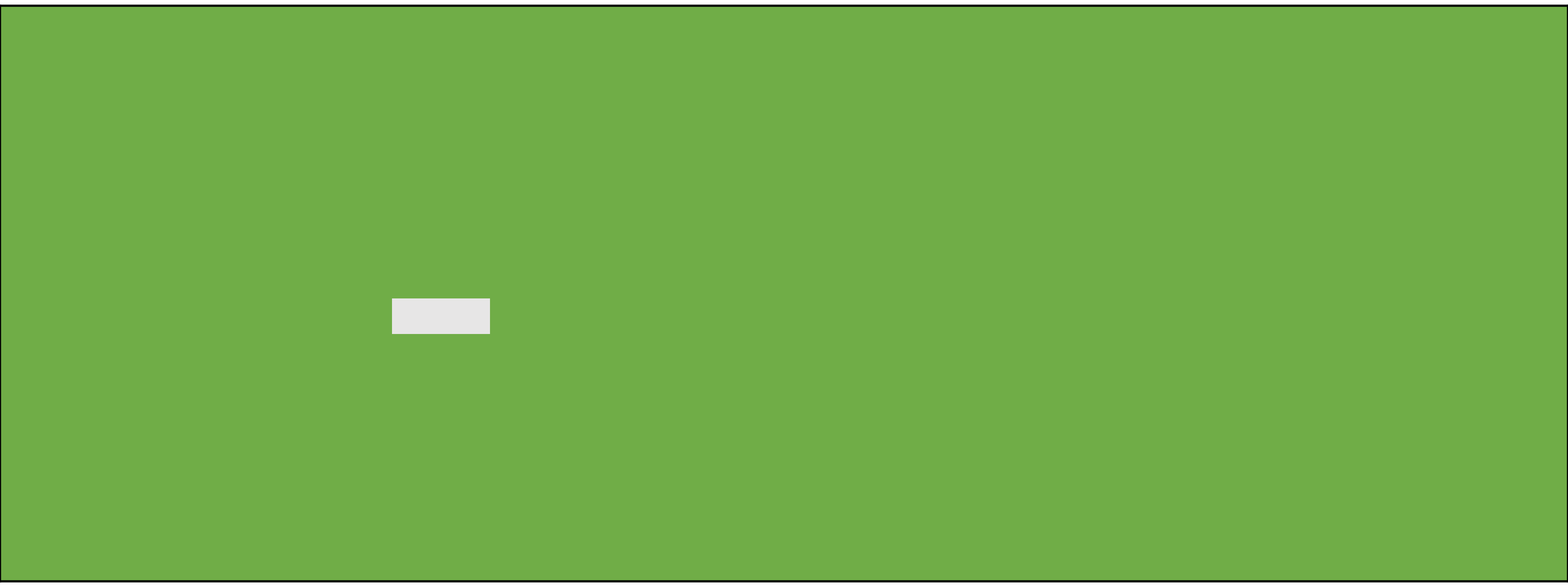
Release the vulnerable page

Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui

Physical memory:



Release the vulnerable page

Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui

Physical memory:

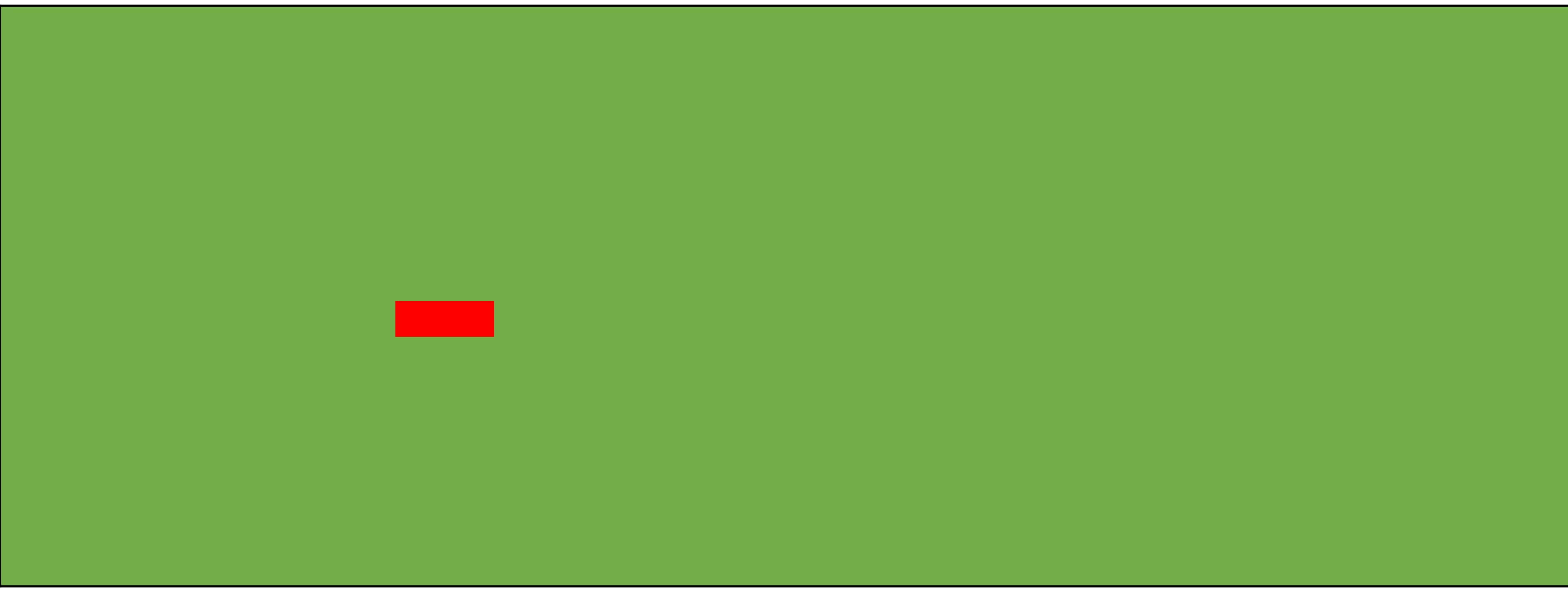
Trigger a Page Table Allocation

Landing a Page Table

- No access to `pagemap` (virtual – physical address mapping)
- No fancy memory management features (deduplication)

Phys Feng Shui

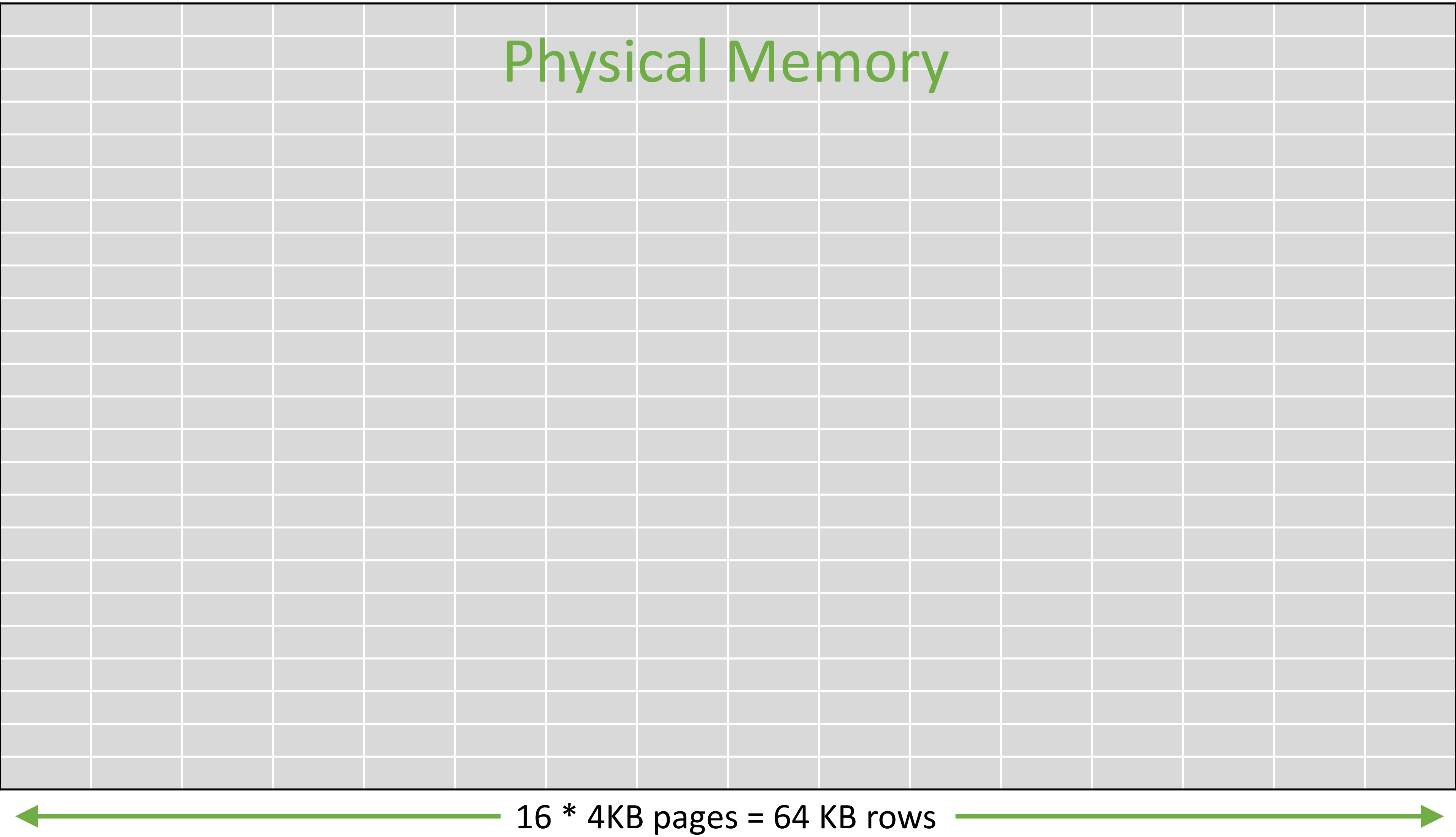
Physical memory:



Trigger a Page Table Allocation

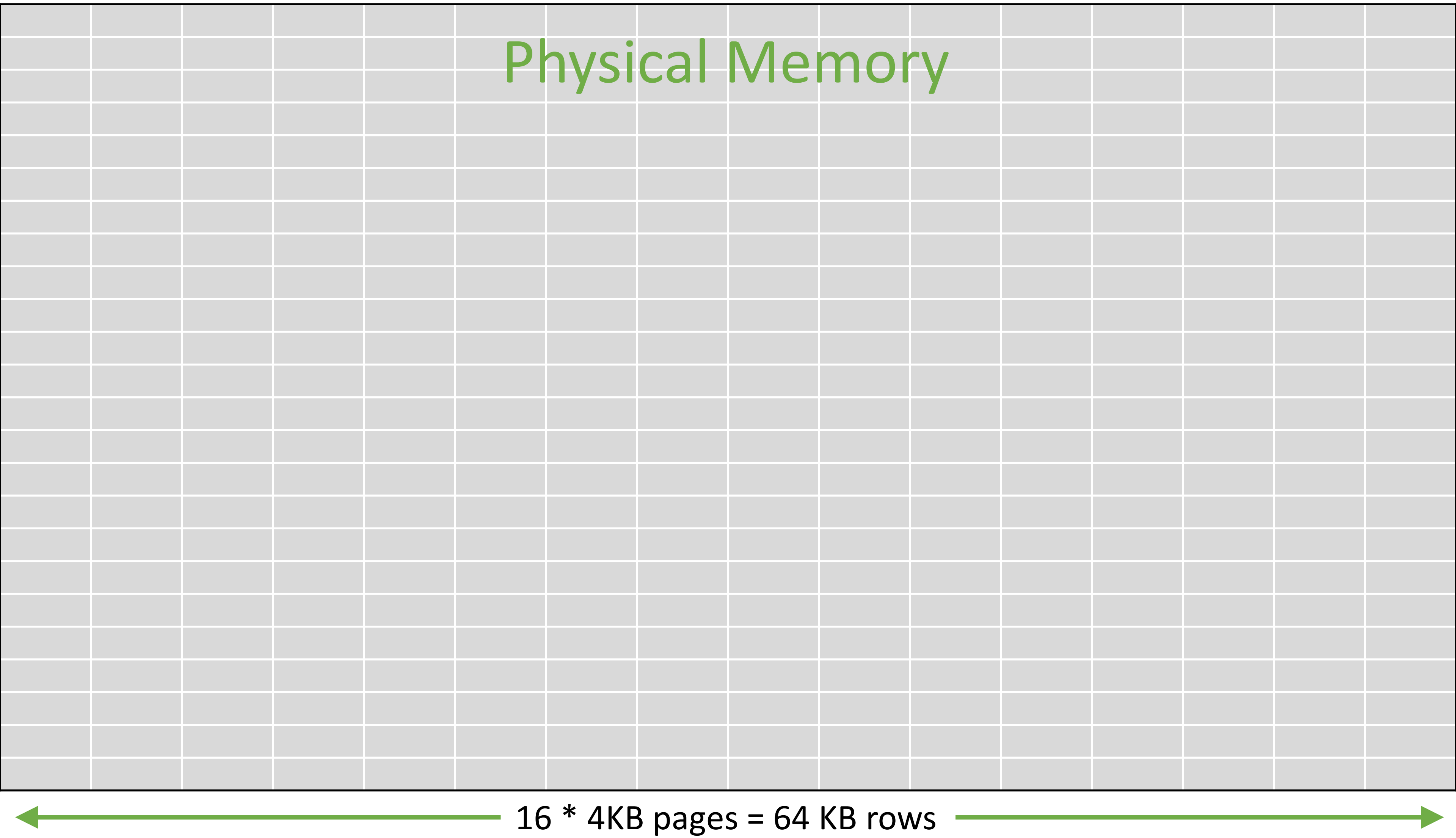
Phys Feng Shui

Exploit the predictable behavior of the Buddy Allocator



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies*)



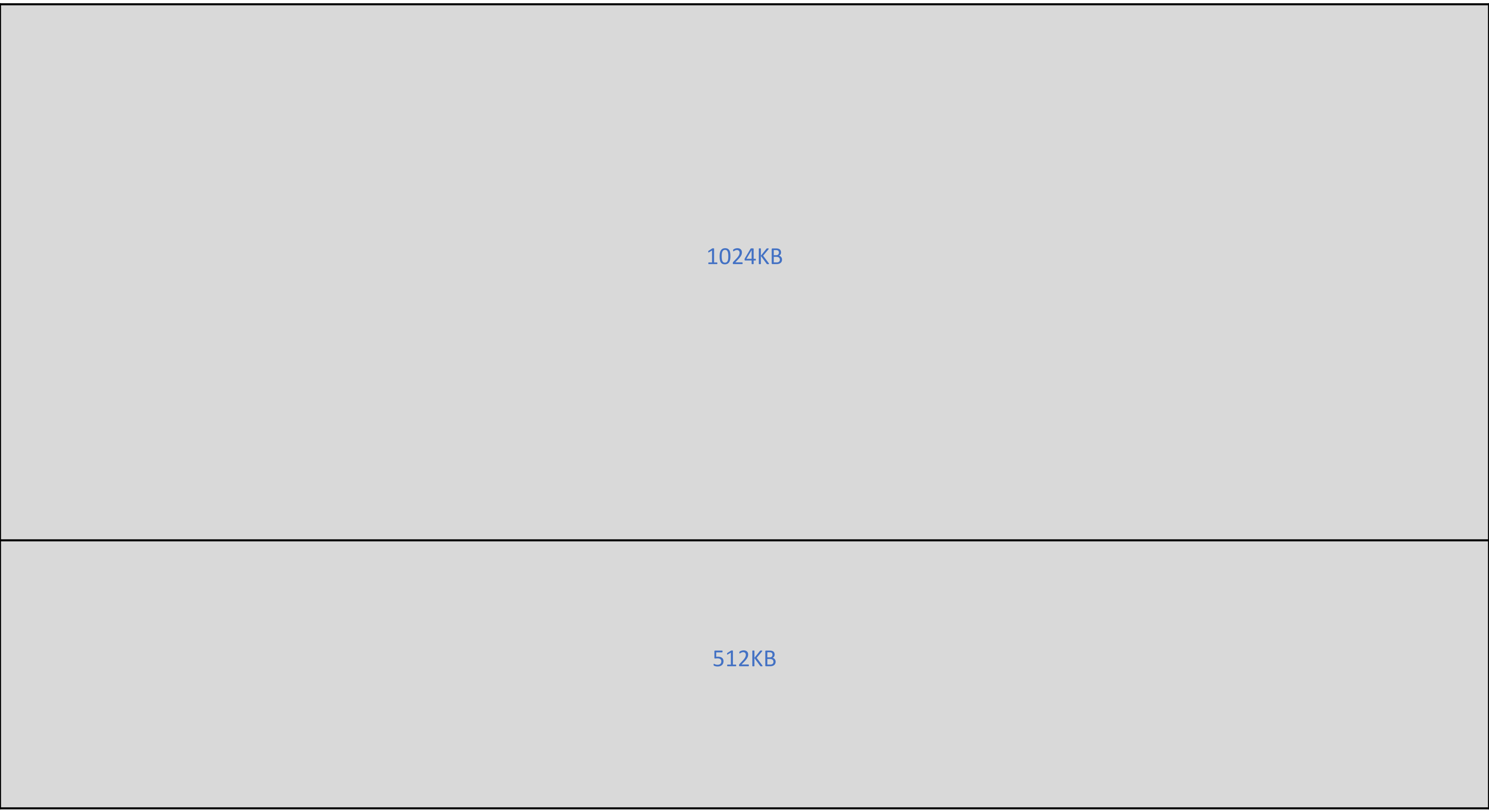
Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

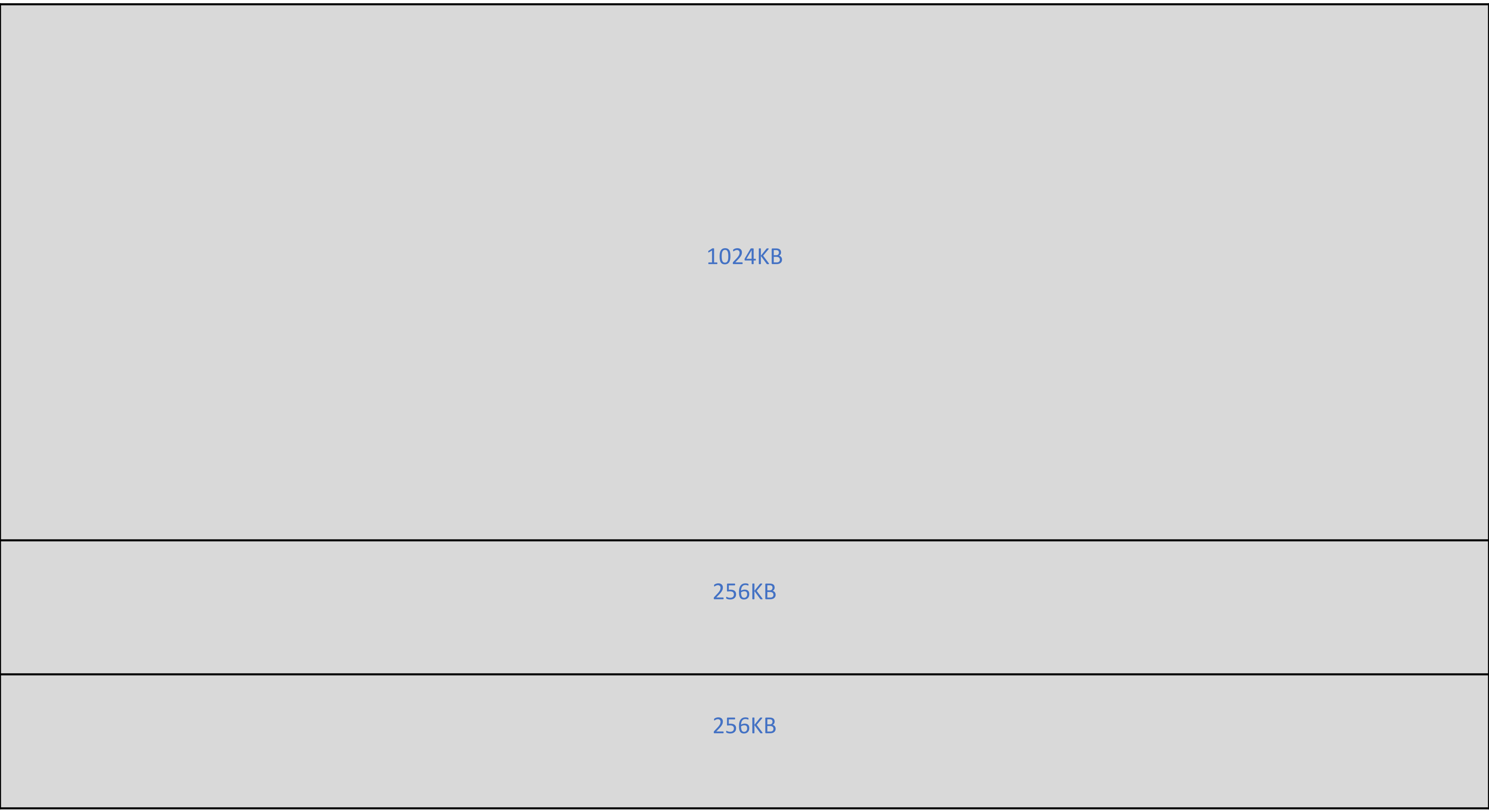
X1 = __get_free_pages(flags, 6); // get 2^6 = 64KB of memory



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

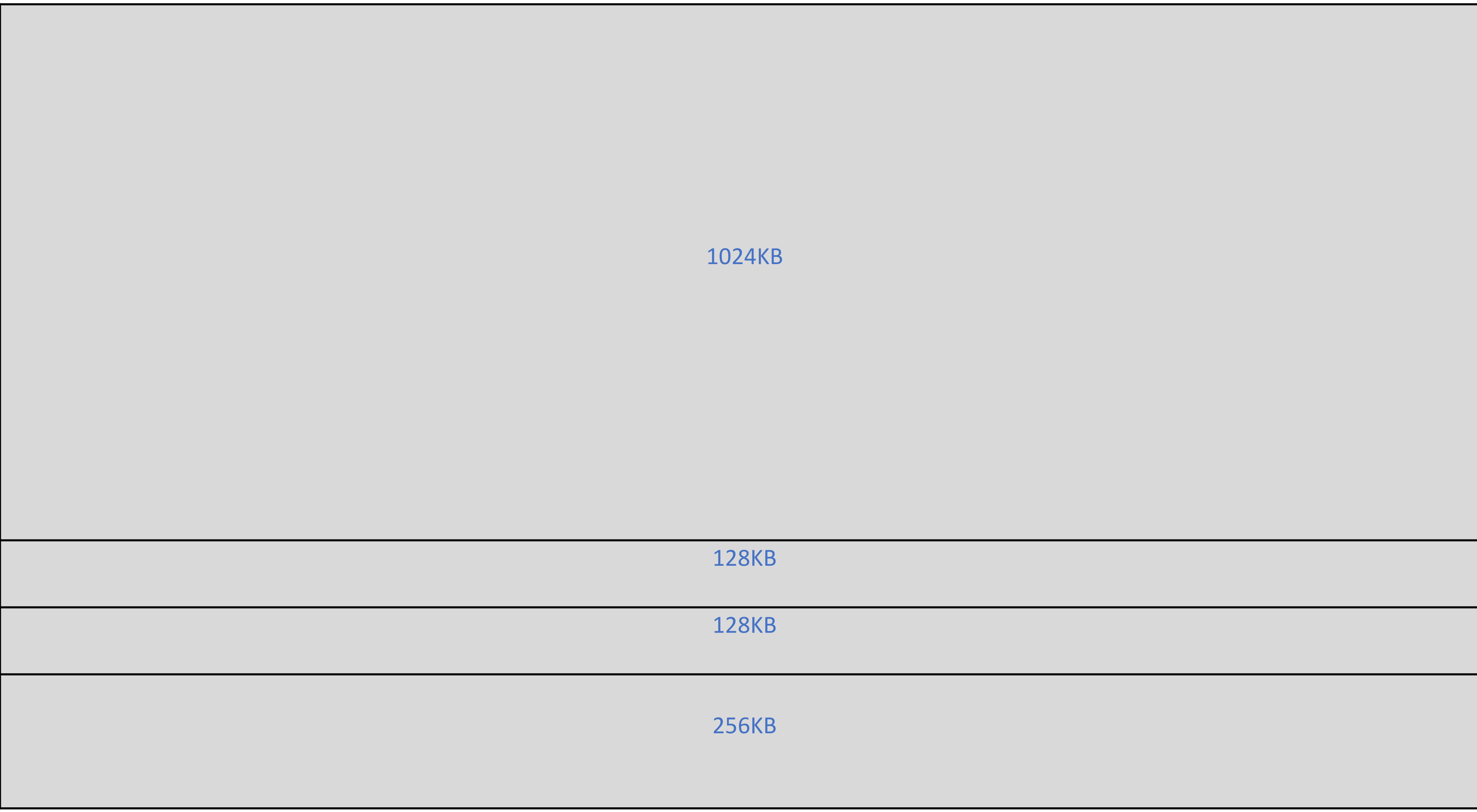
```
X1 = __get_free_pages(flags, 6); // get 2^6 = 64KB of memory
```



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

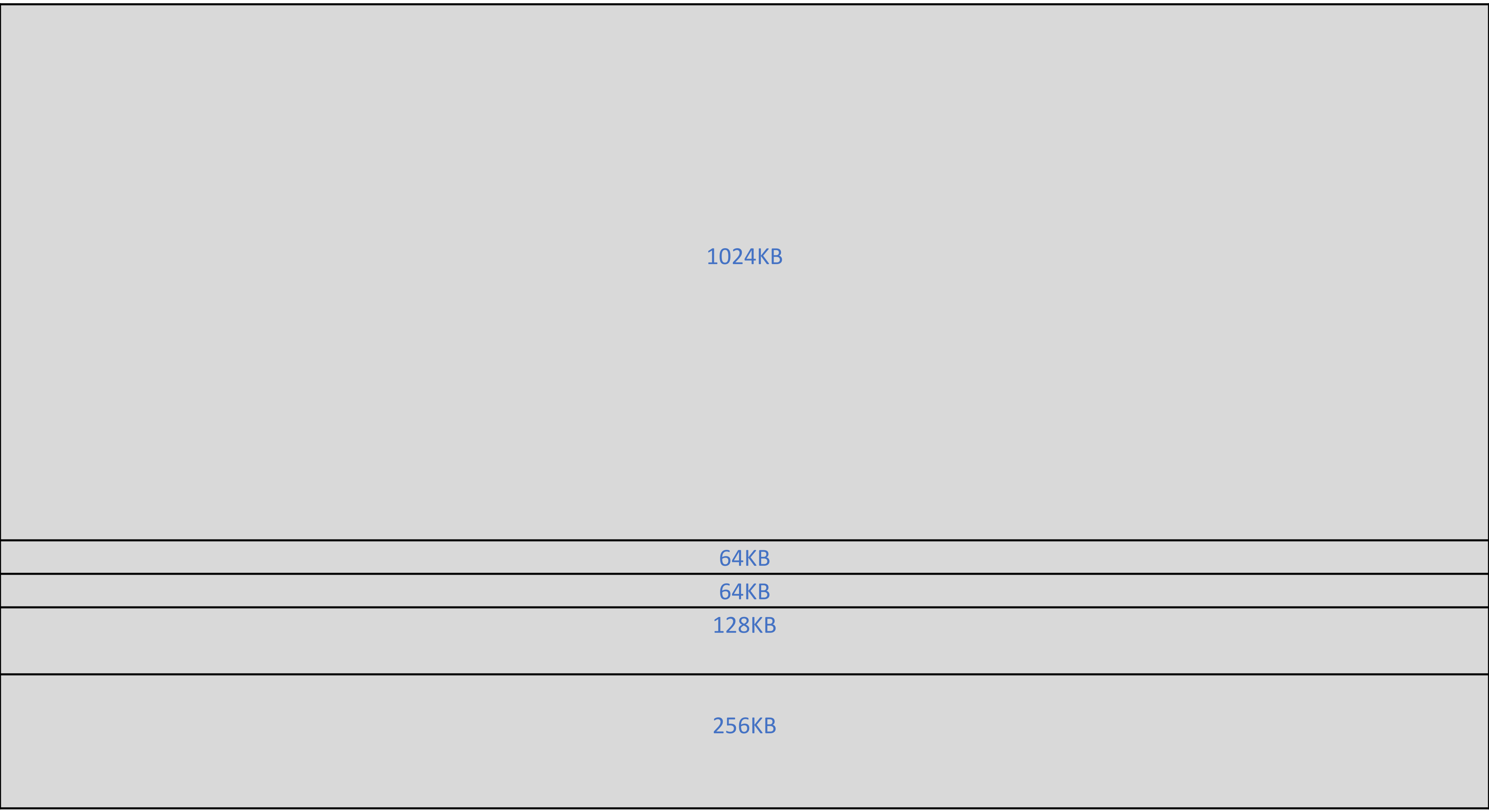
```
X1 = __get_free_pages(flags, 6); // get 2^6 = 64KB of memory
```



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

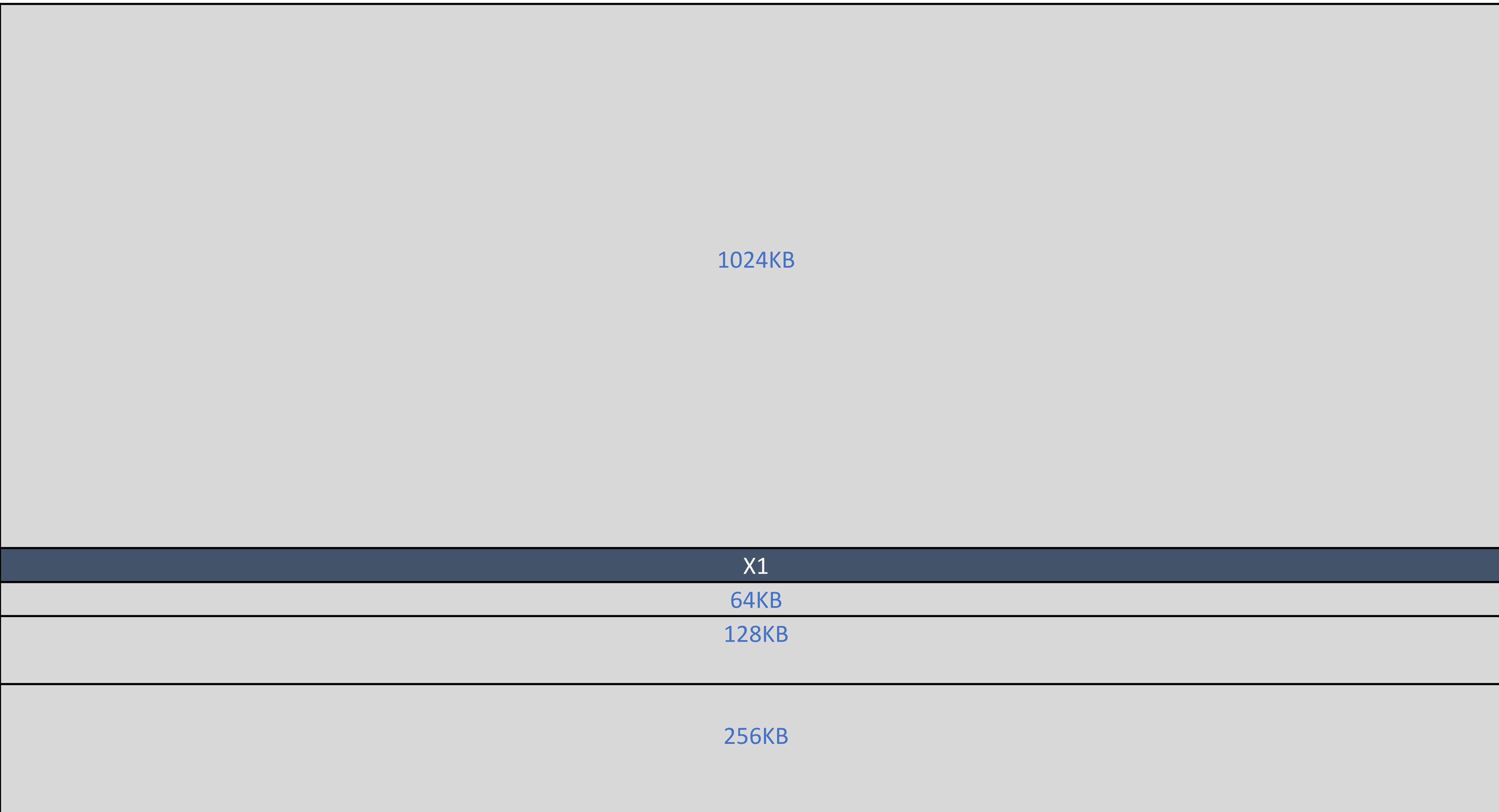
`X1 = __get_free_pages(flags, 6); // get 2^6 = 64KB of memory`



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

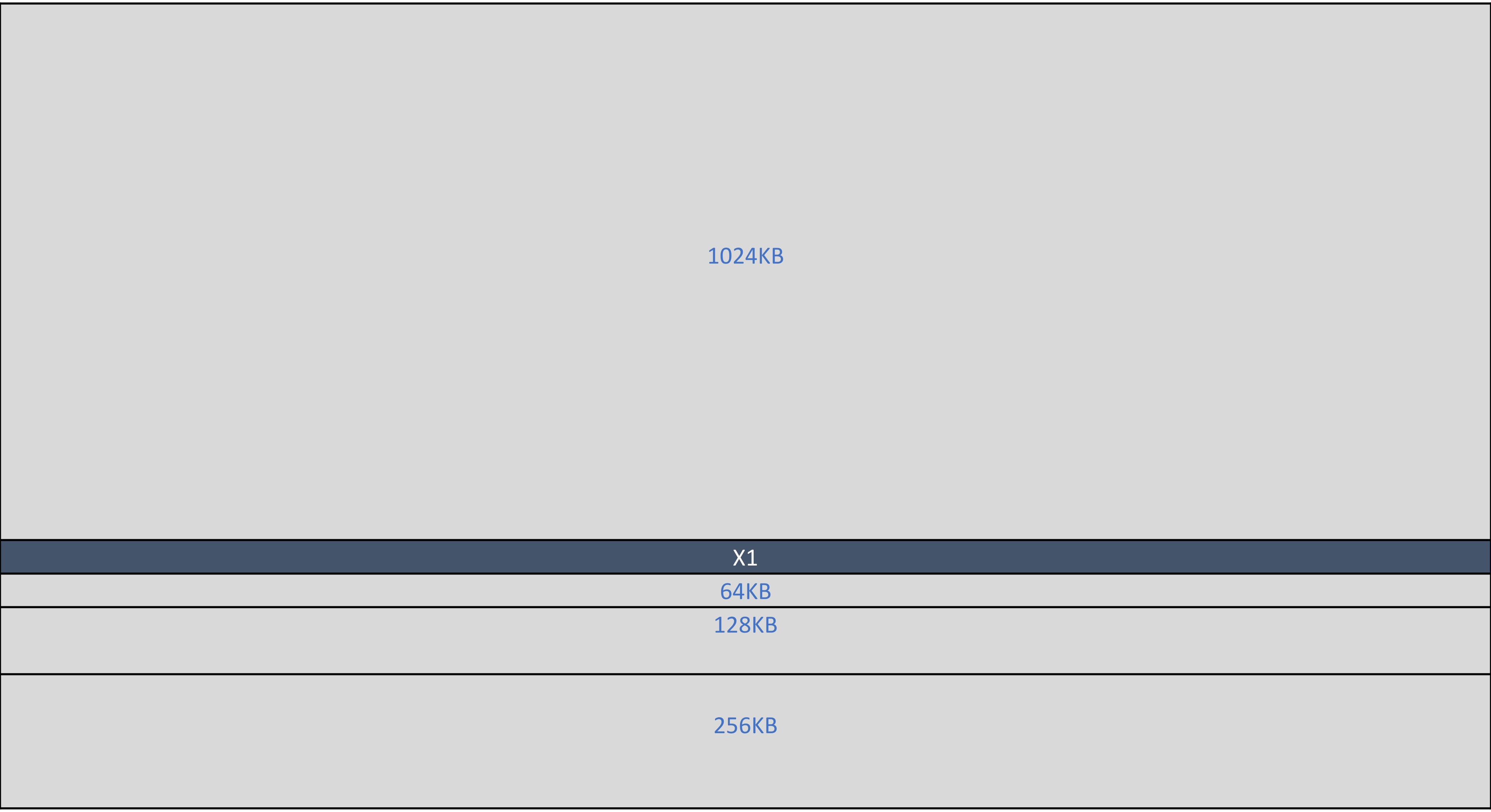
```
X1 = __get_free_pages(flags, 6); // get 2^6 = 64KB of memory
```



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

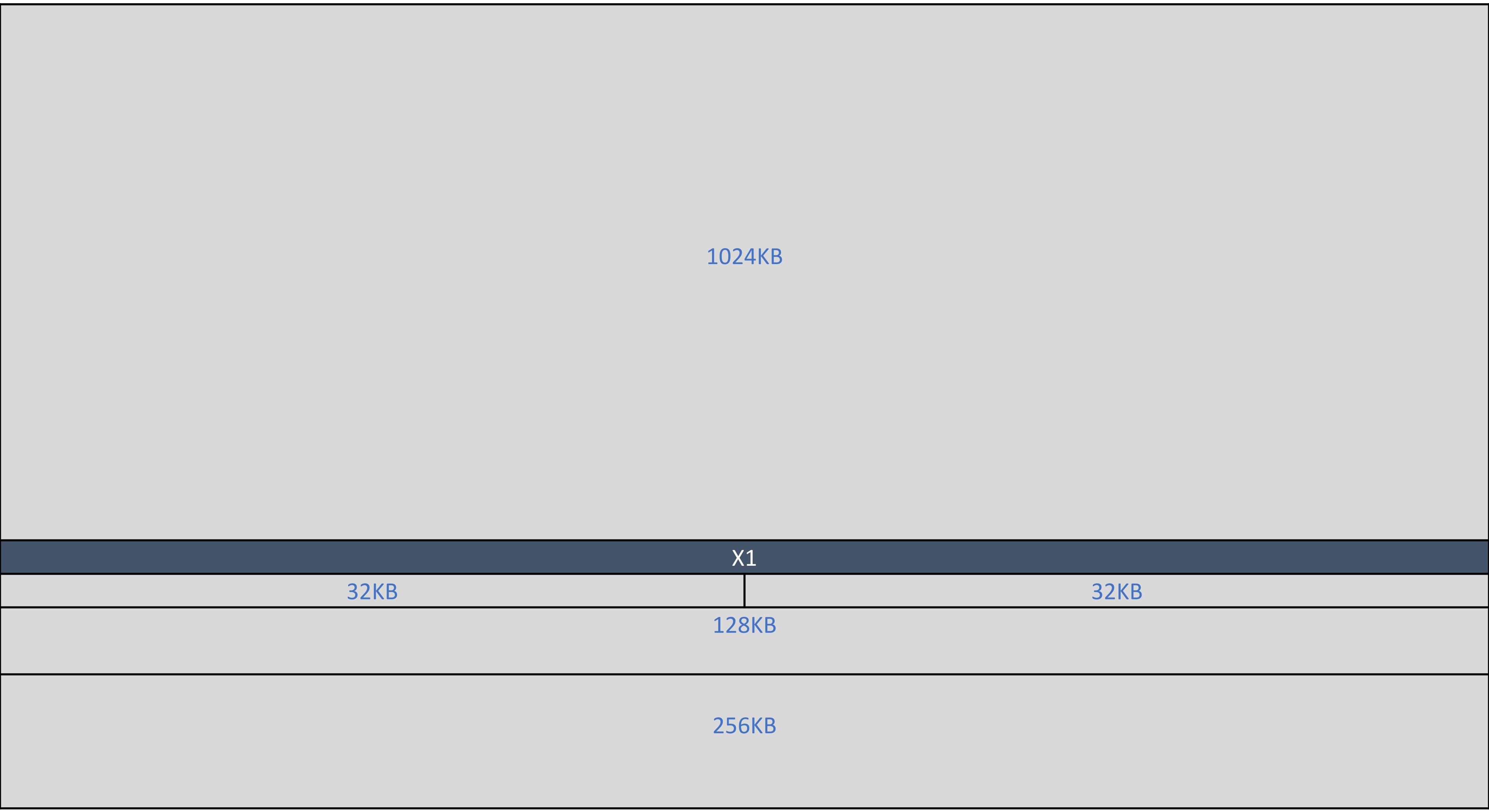
```
X2 = __get_free_pages(flags, 3); // get 2^3 = 8KB of memory
```



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

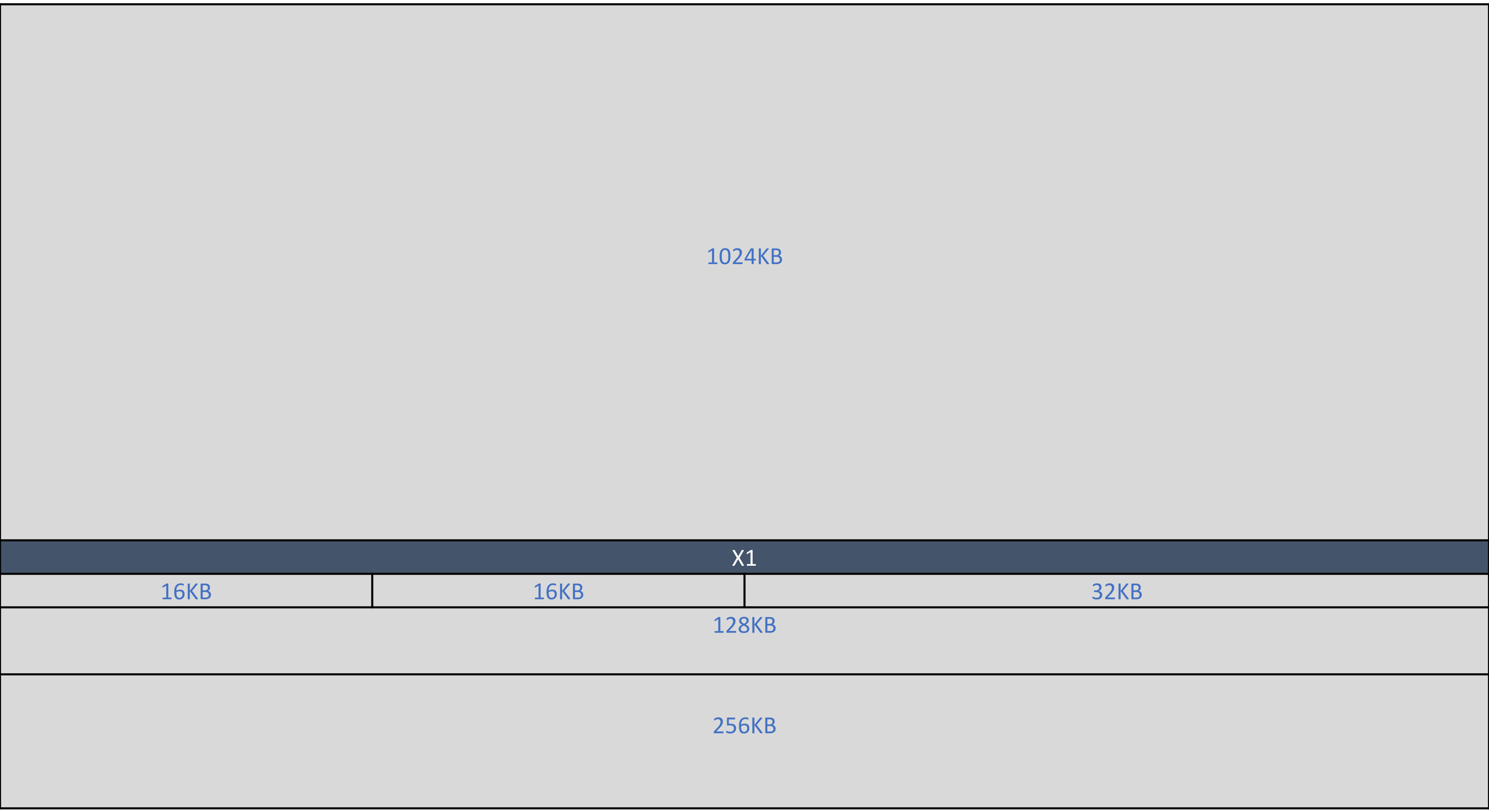
```
X2 = __get_free_pages(flags, 3); // get 2^3 = 8KB of memory
```



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

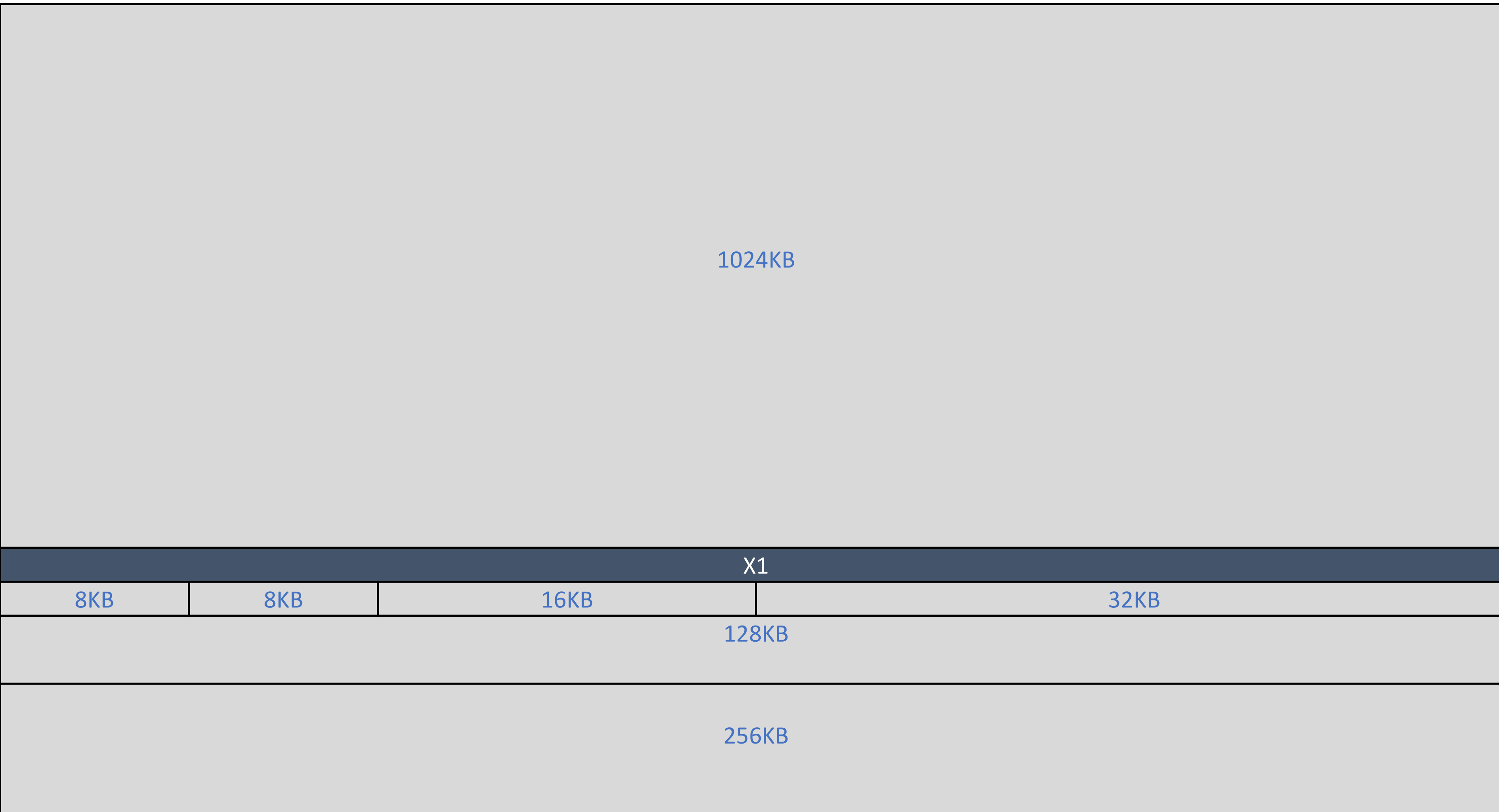
```
X2 = __get_free_pages(flags, 3); // get 2^3 = 8KB of memory
```



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

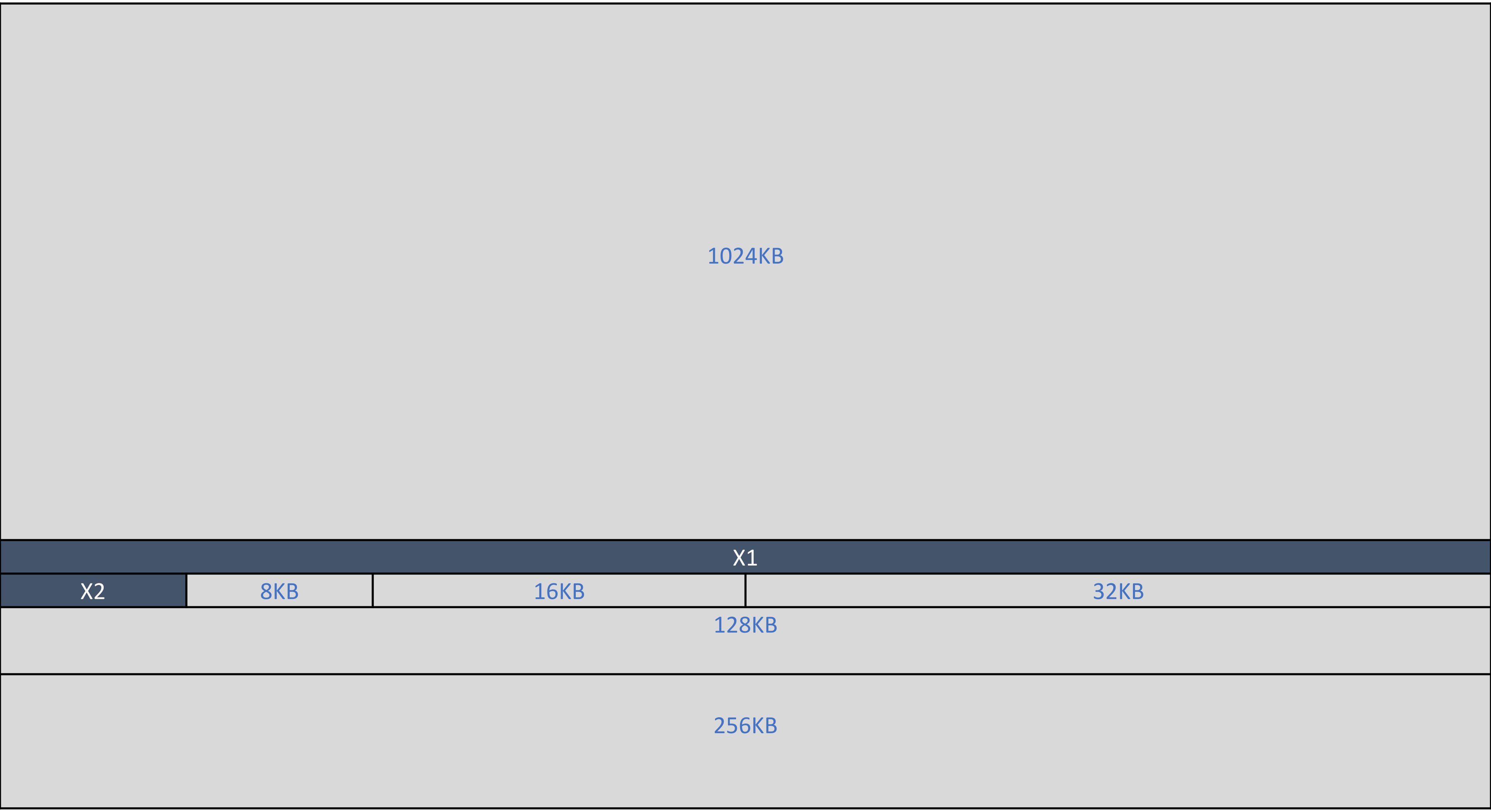
```
X2 = __get_free_pages(flags, 3); // get 2^3 = 8KB of memory
```



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

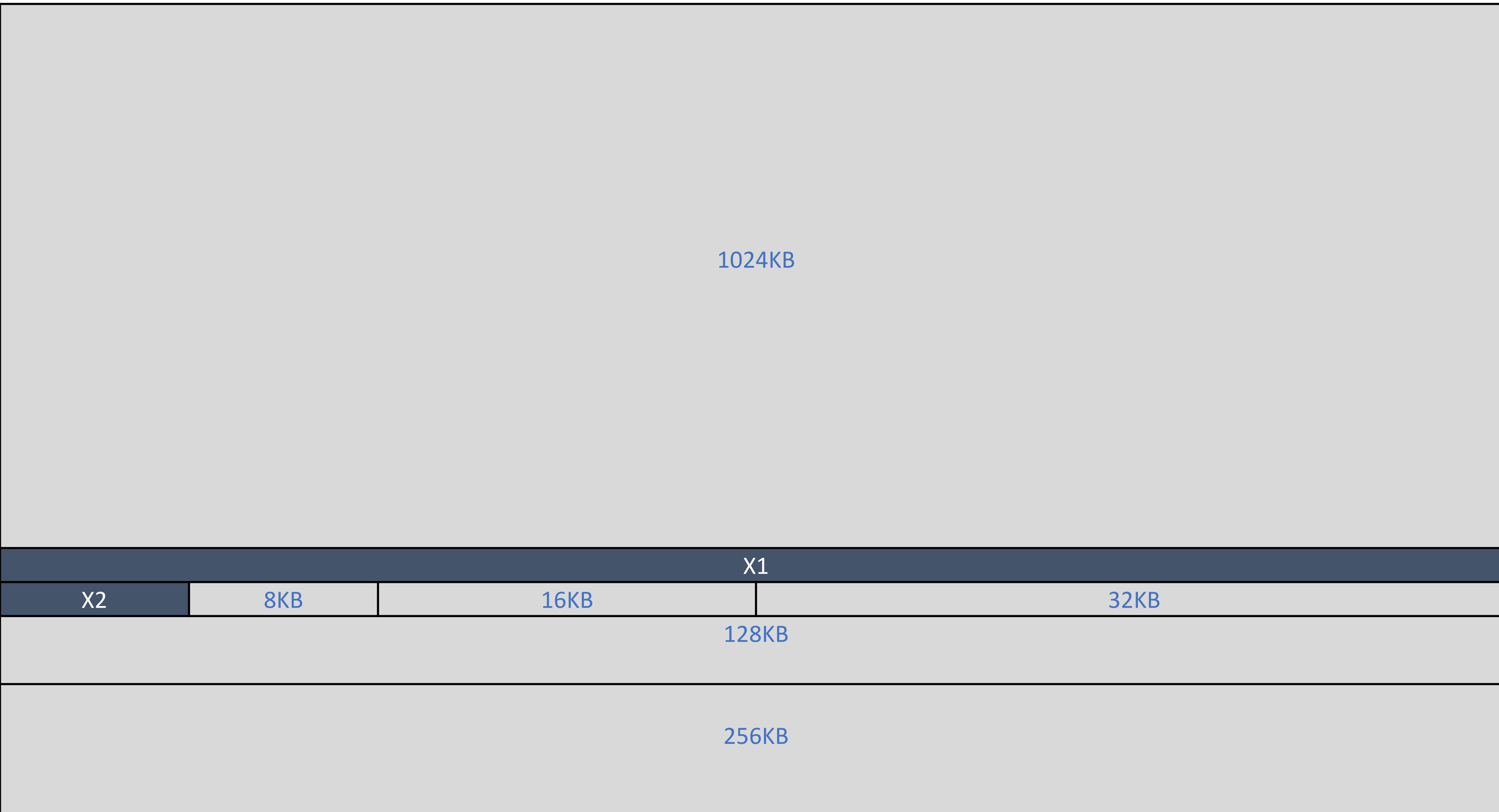
```
X2 = __get_free_pages(flags, 3); // get 2^3 = 8KB of memory
```



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

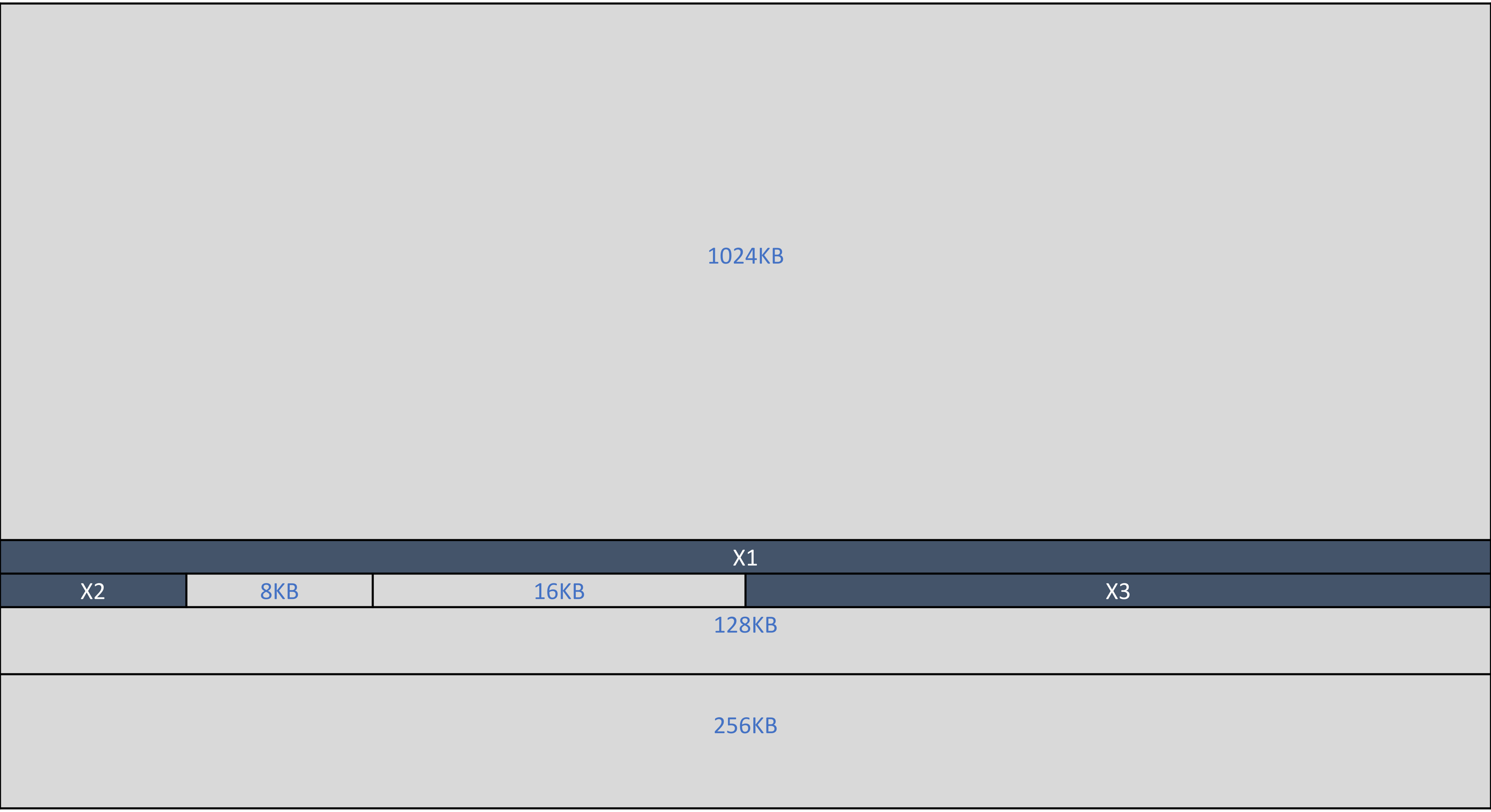
```
X3 = __get_free_pages(flags, 5); // get 2^3 = 32KB of memory
```



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

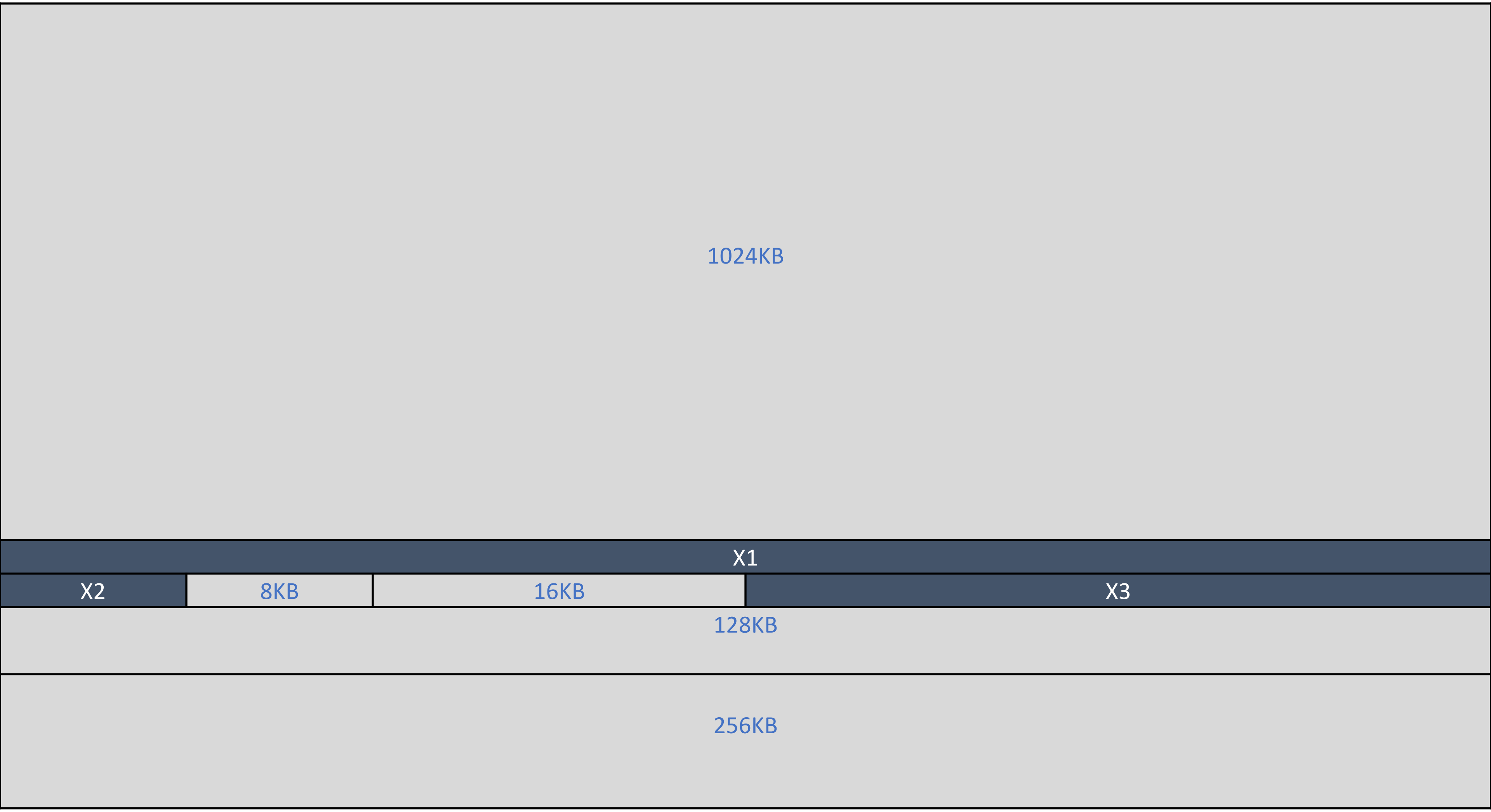
`P3 = __get_free_pages(flags, 5); // get 2^3 = 32KB of memory`



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

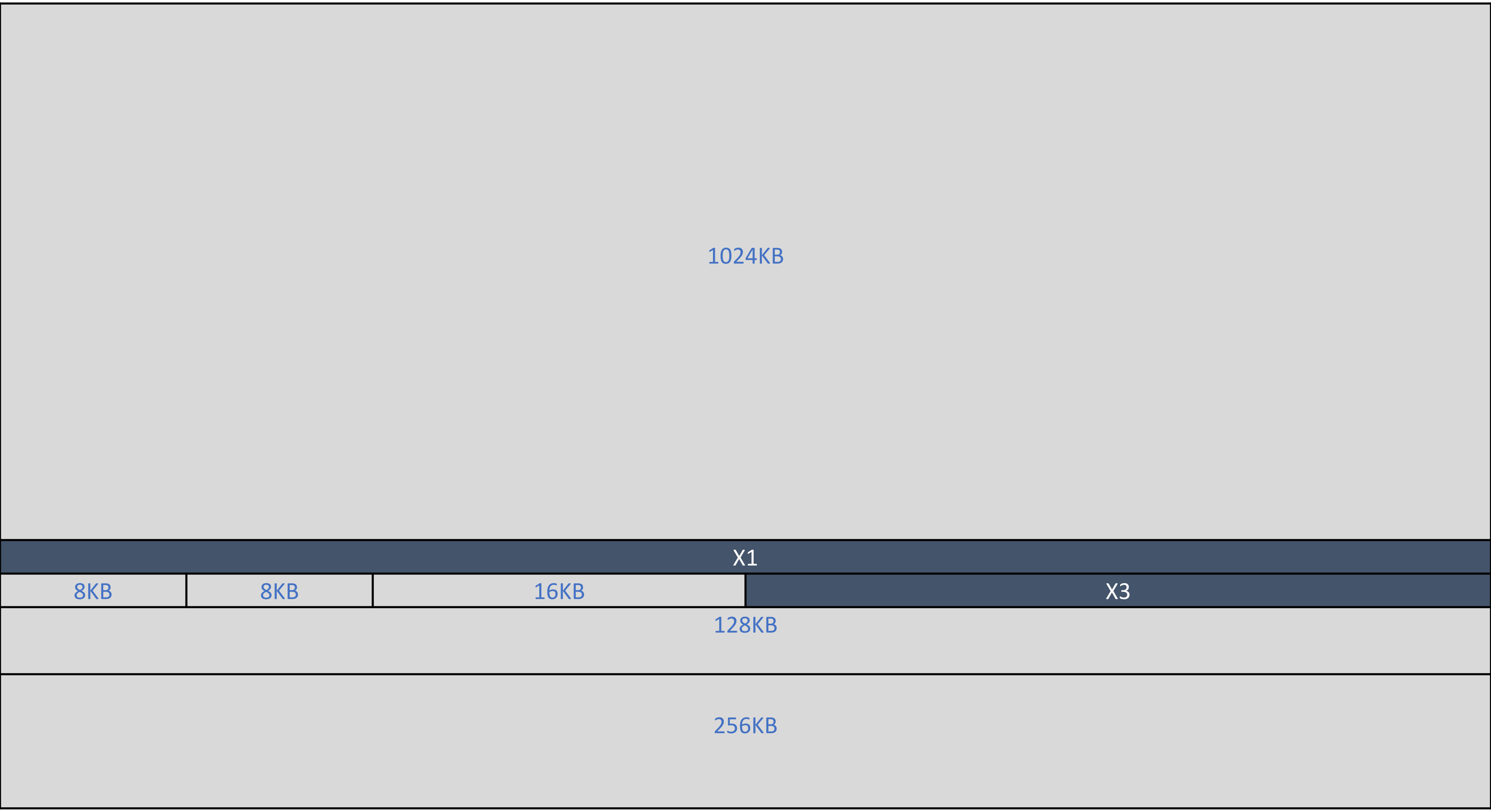
```
free_pages(X2, 3); // free X2
```



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

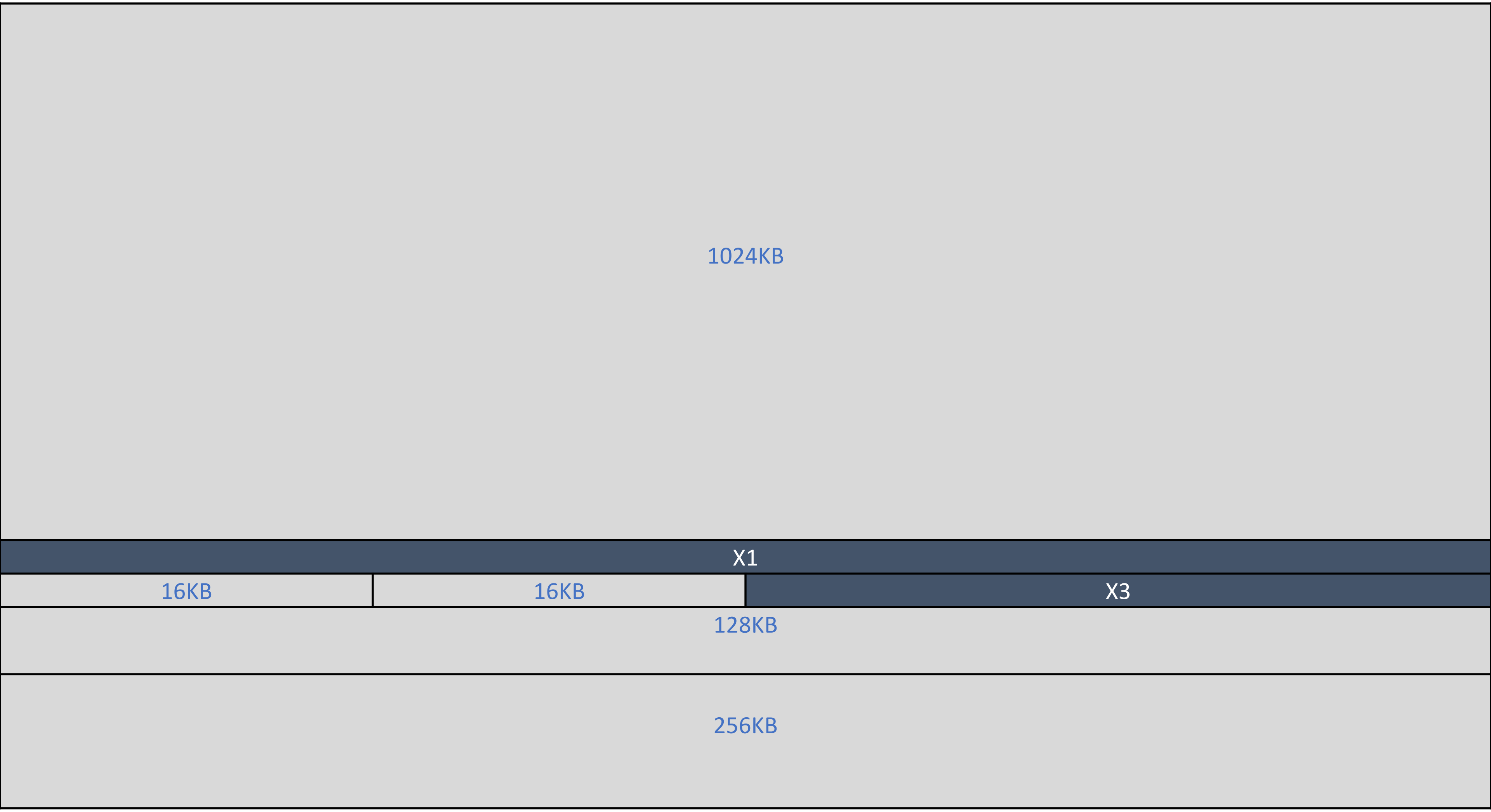
```
free_pages(X2, 3); // free X2
```



Phys Feng Shui – Buddy Allocator

Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

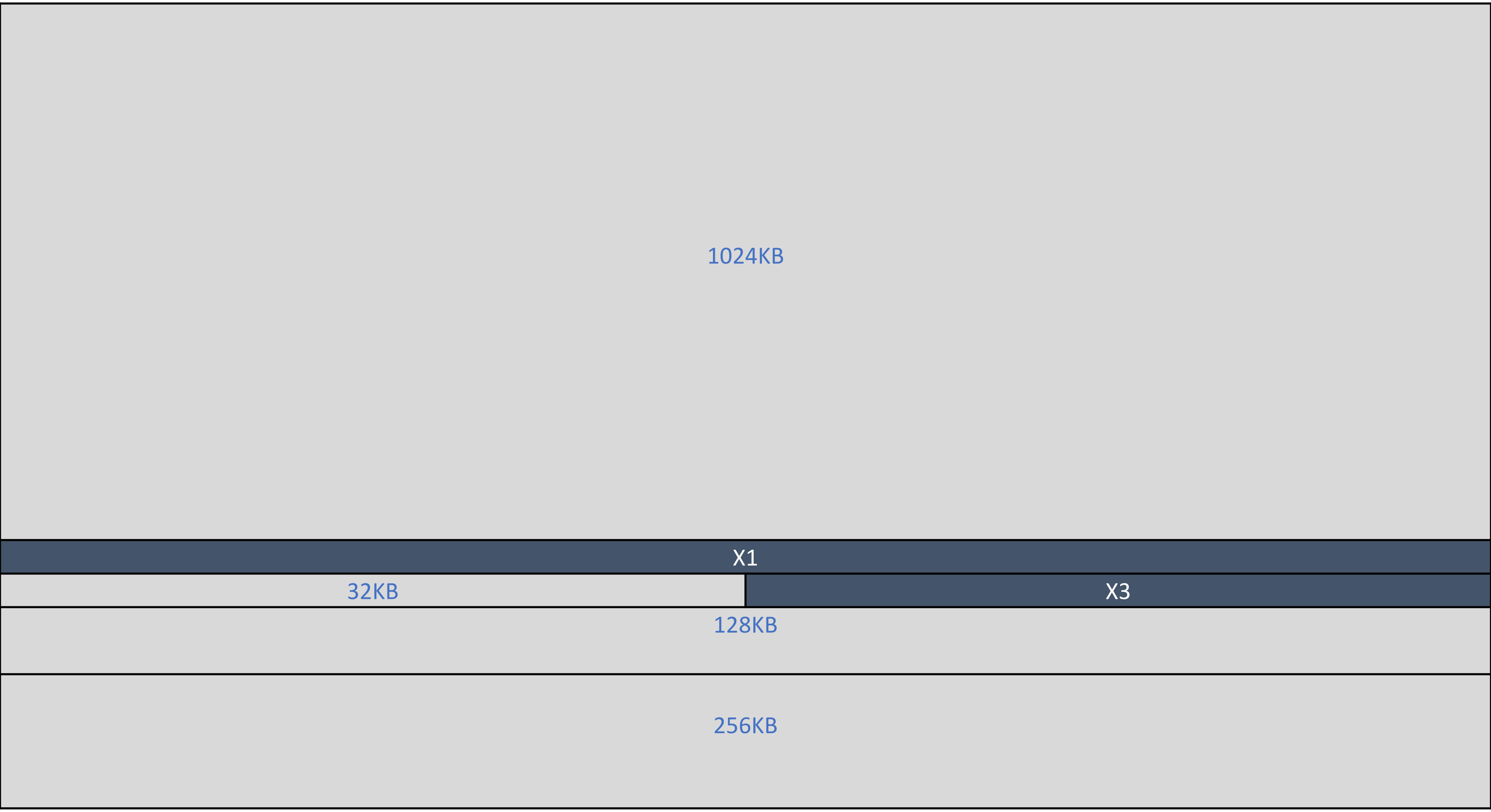
```
free_pages(X2, 3); // free X2
```



Phys Feng Shui – Buddy Allocator

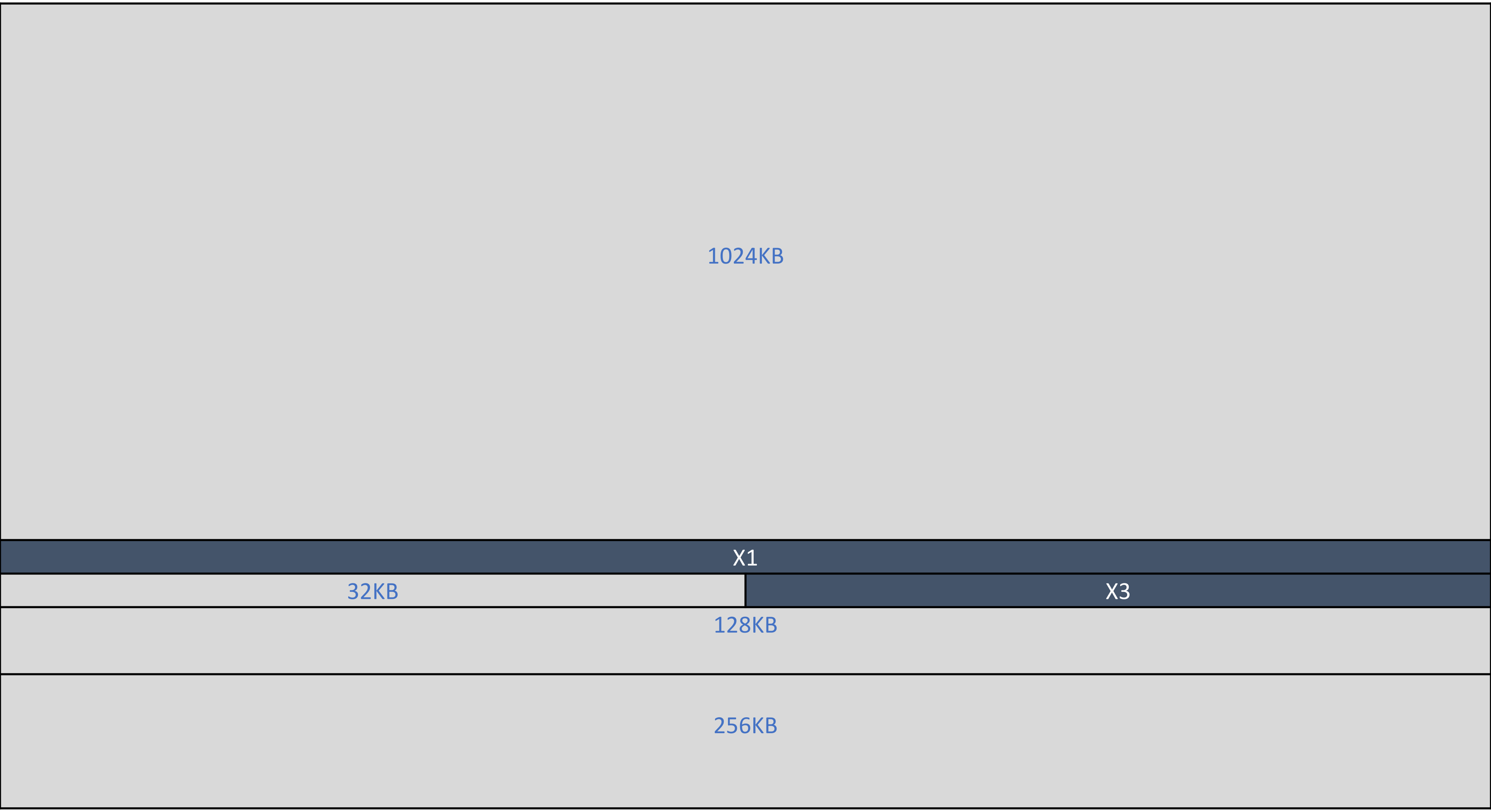
Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

```
free_pages(X2, 3); // free X2
```



Phys Feng Shui

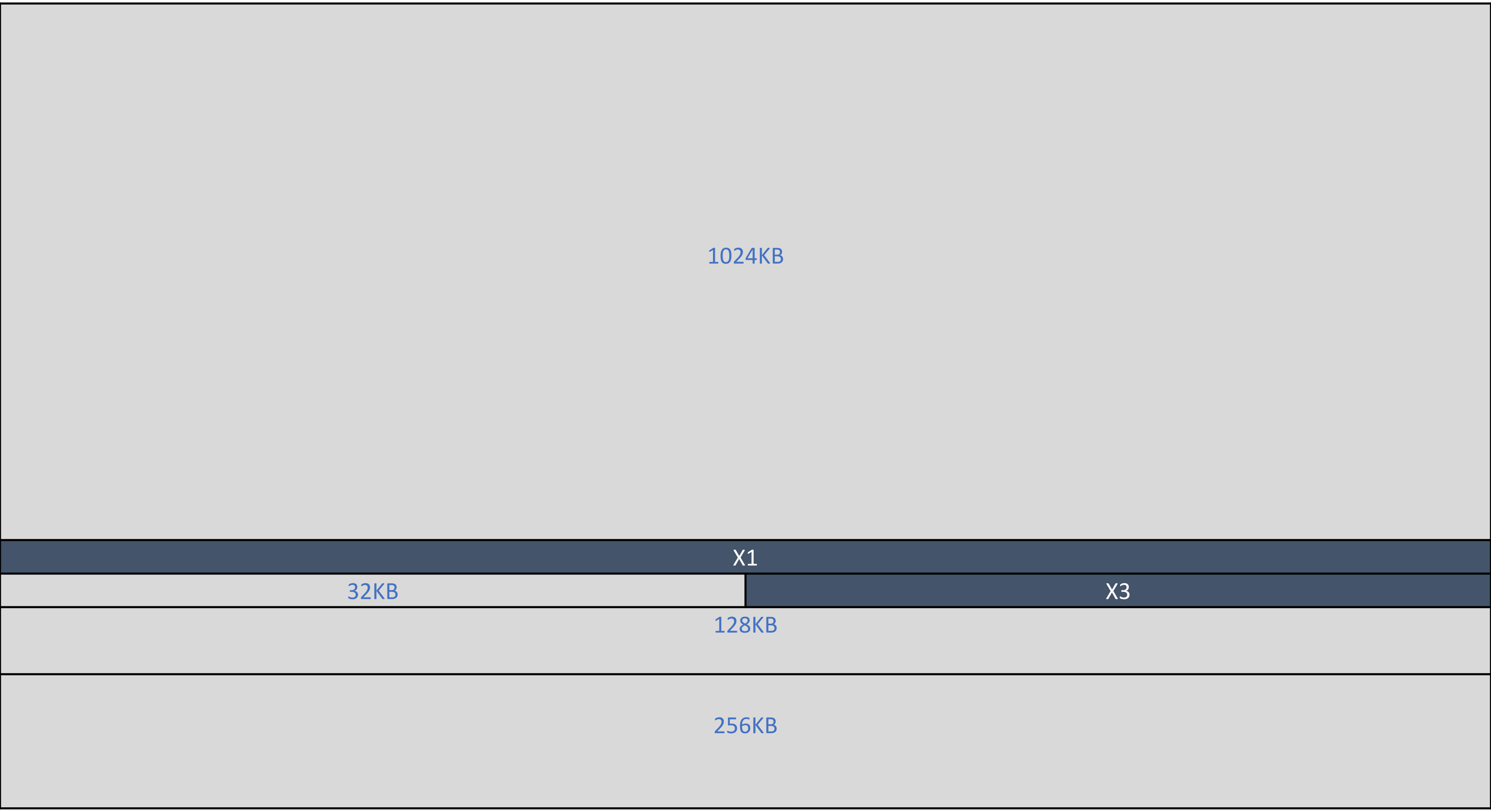
Deterministic Rowhammer exploitation in 8 steps



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

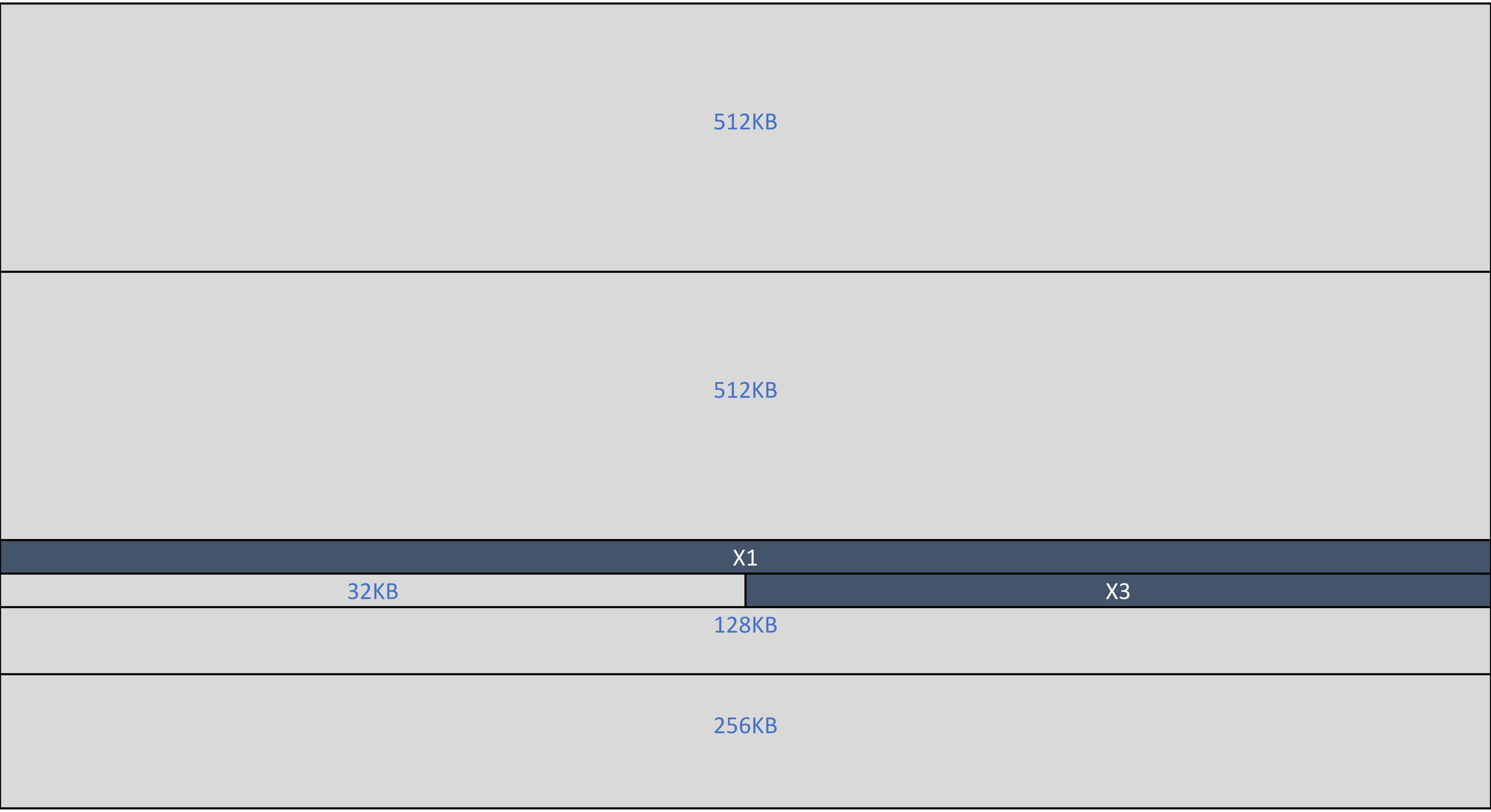
```
L1, L2, …, Ln = exhaust(L);
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

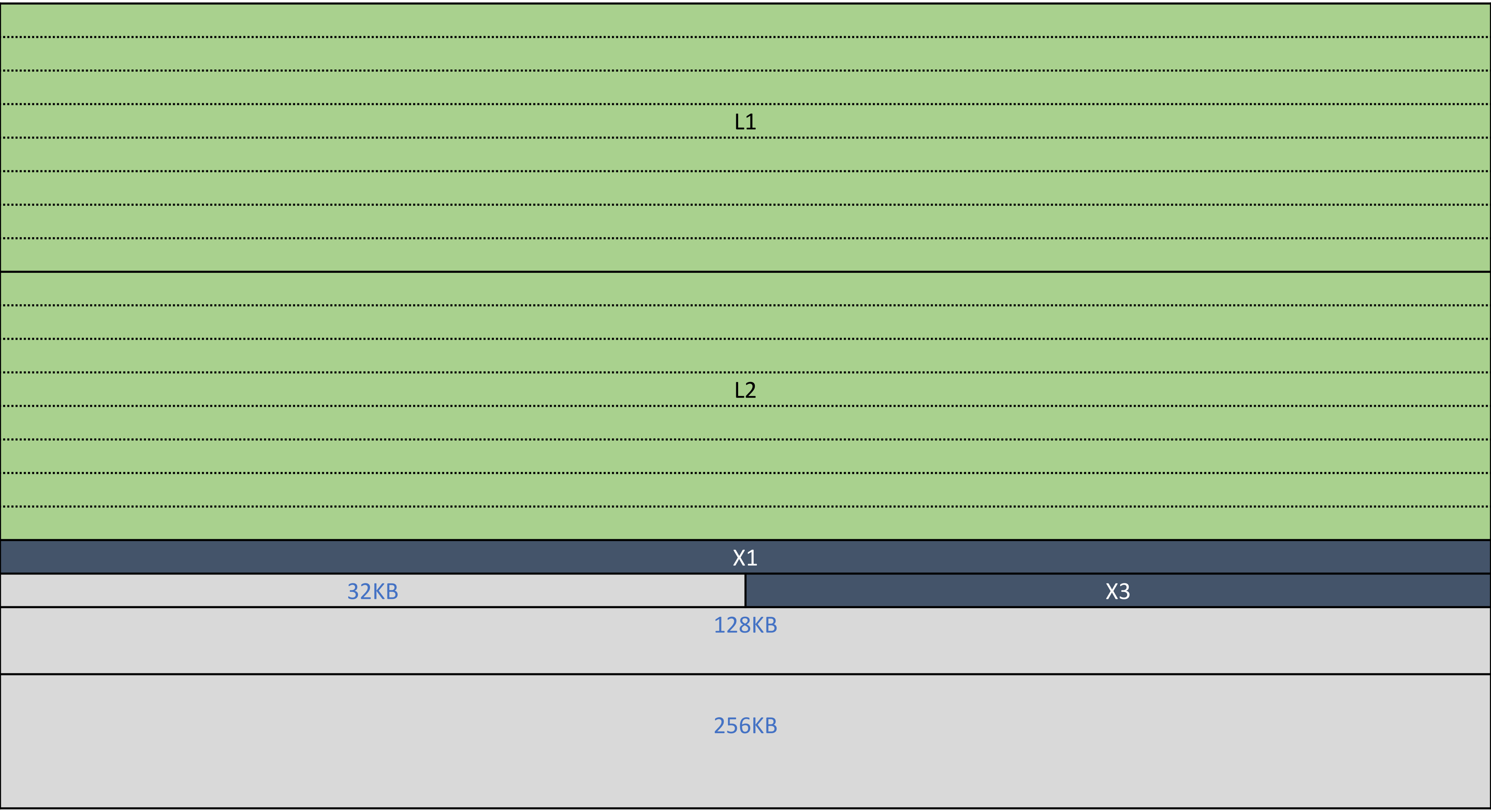
```
L1, L2, …, Ln = exhaust(9); // get all 2^9 = 512KB chunks
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

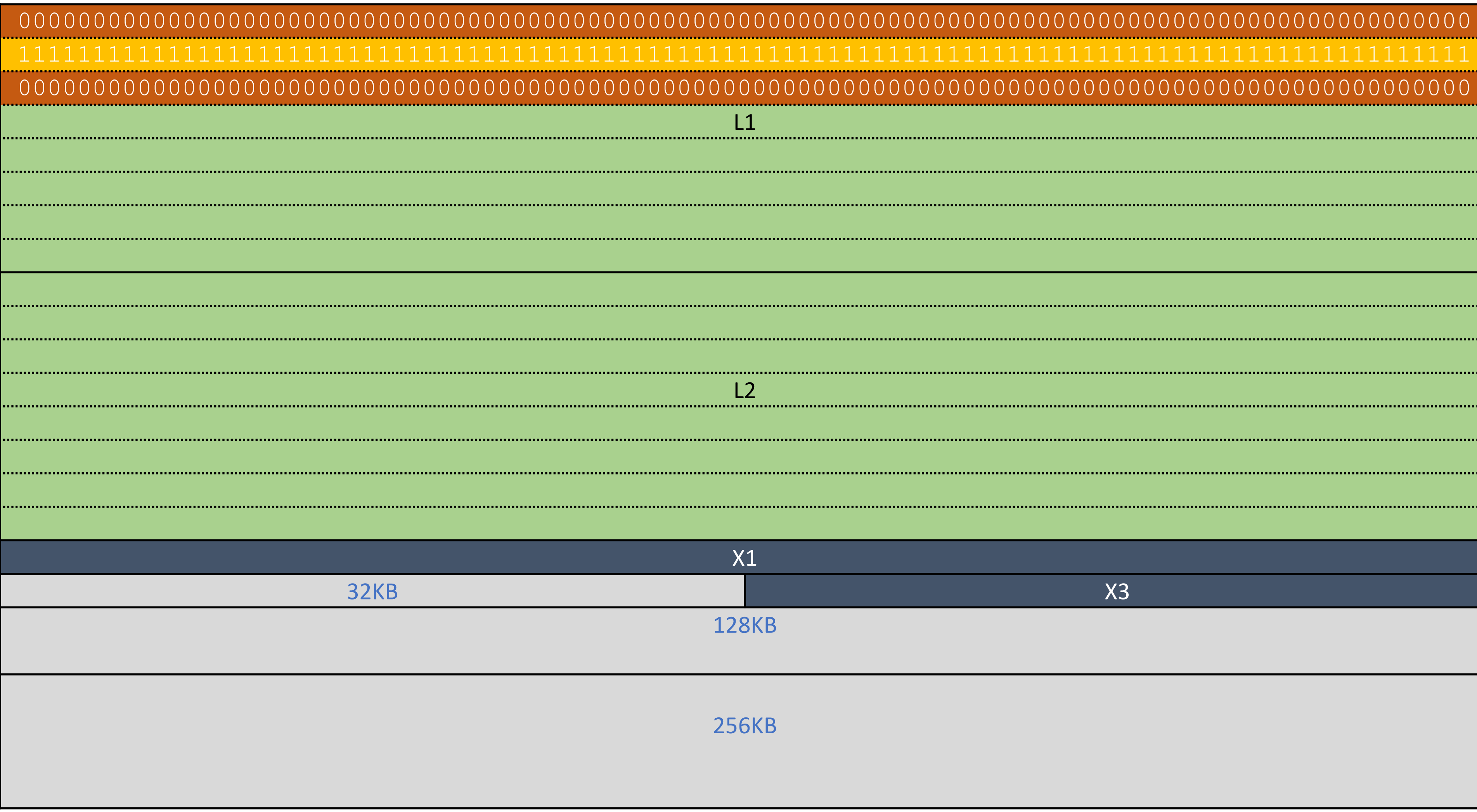
```
L1, L2, …, Ln = exhaust(L); // get all 2^9 = 512KB chunks
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

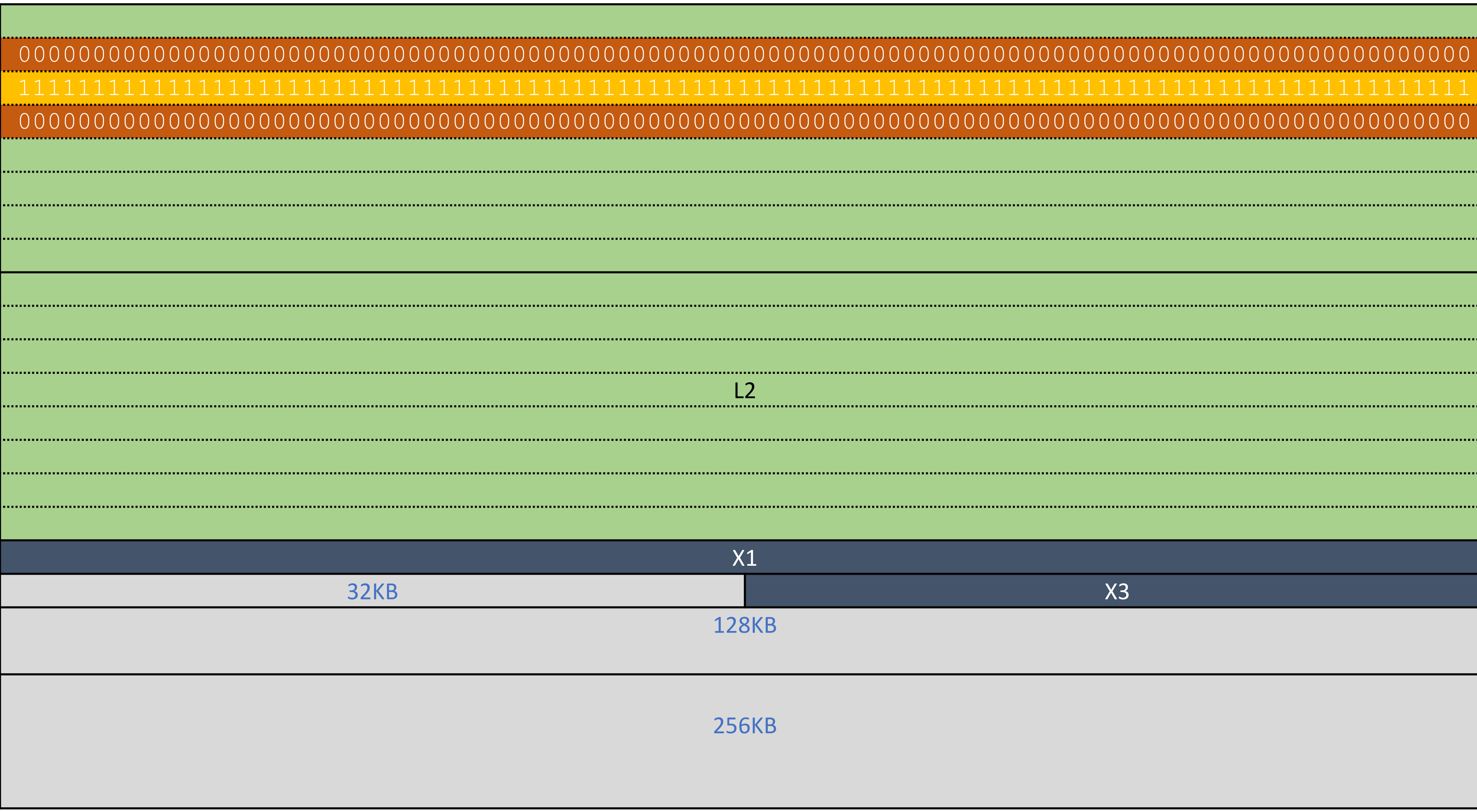
`Hammer(L1, 2);` `// hammer row 2 of chunk L1`



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

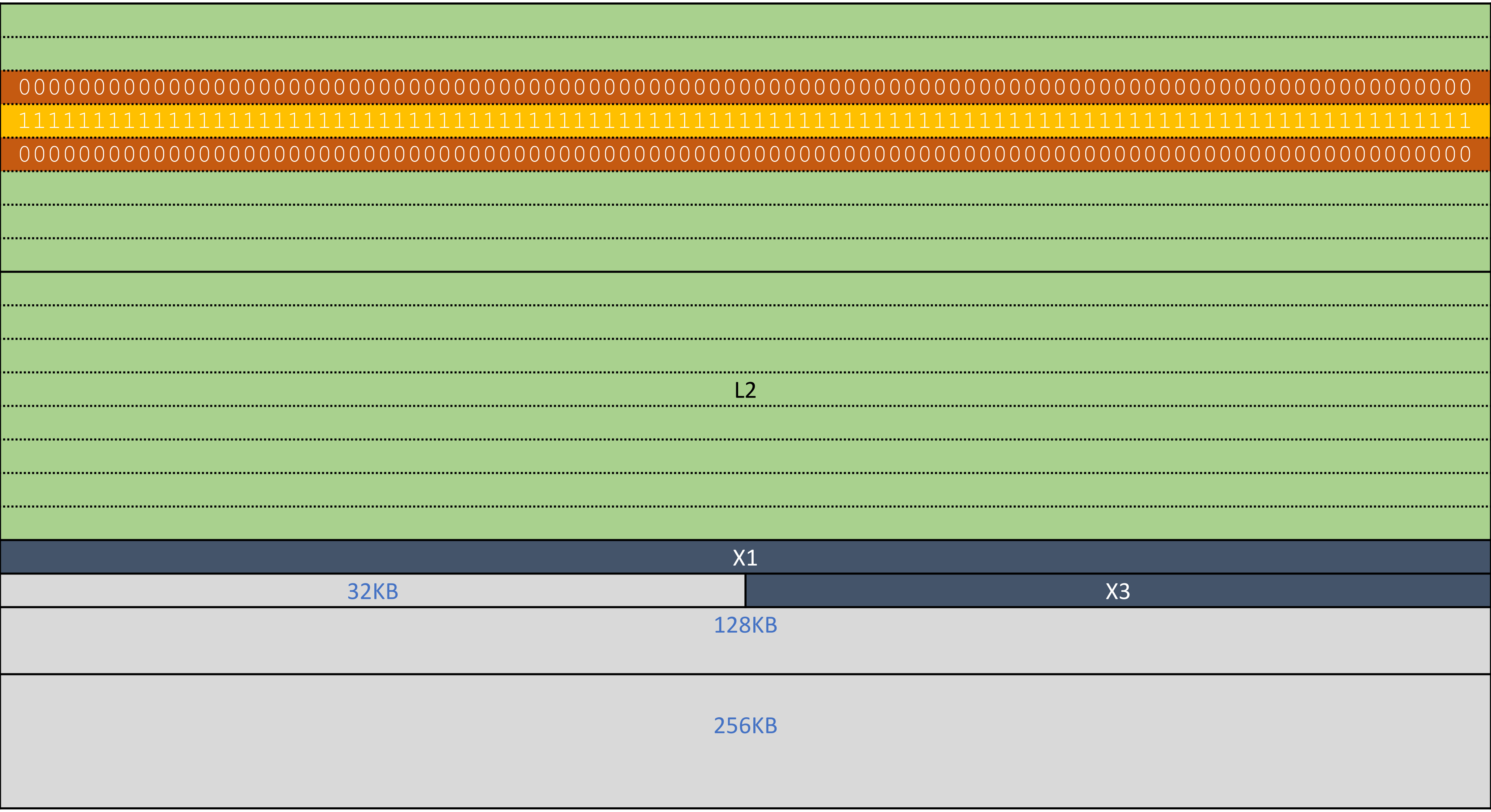
`Hammer(L1, 3);` `// hammer row 3 of chunk L1`



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

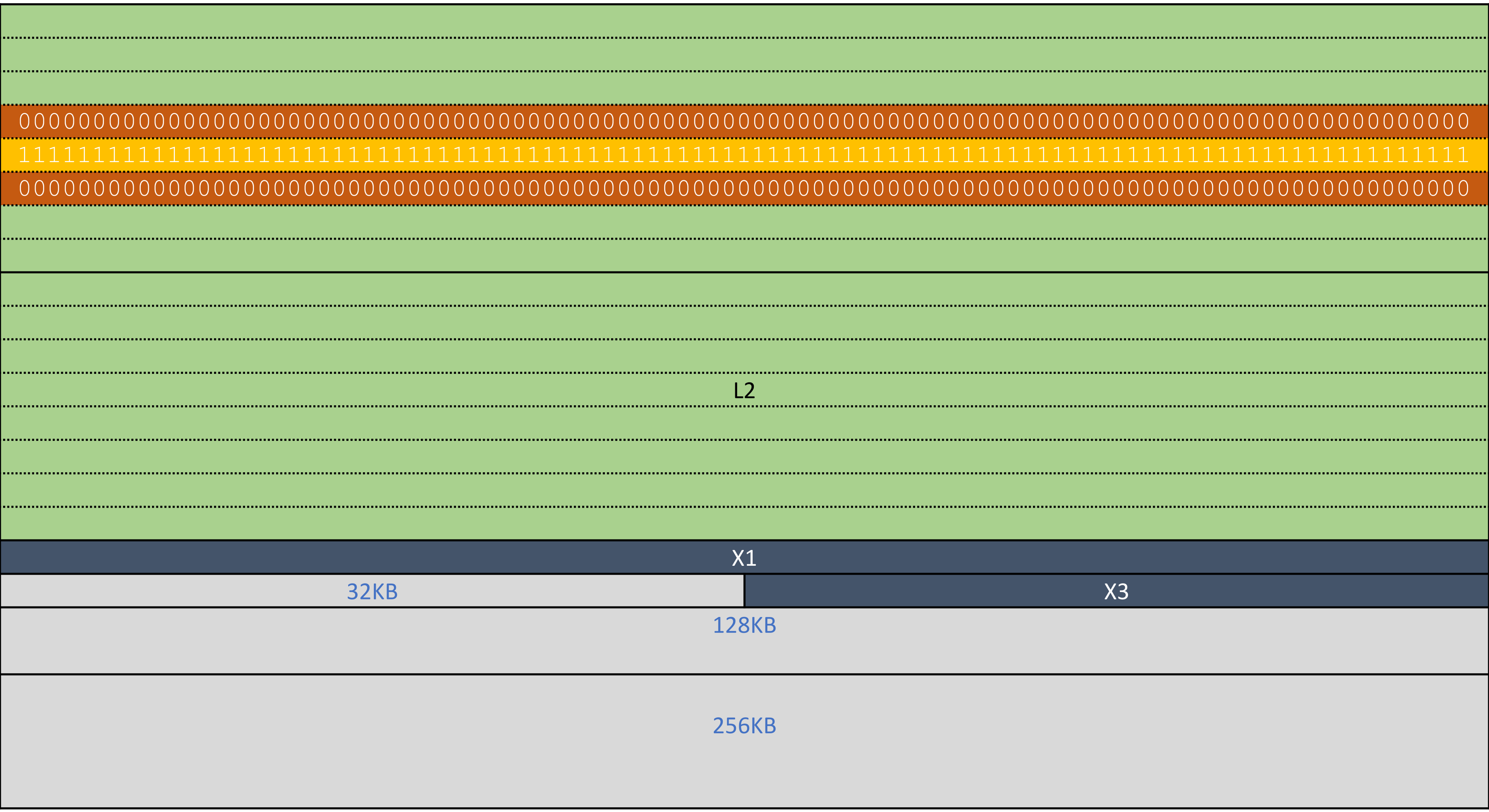
```
Hammer(L1, 4); // hammer row 4 of chunk L1
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

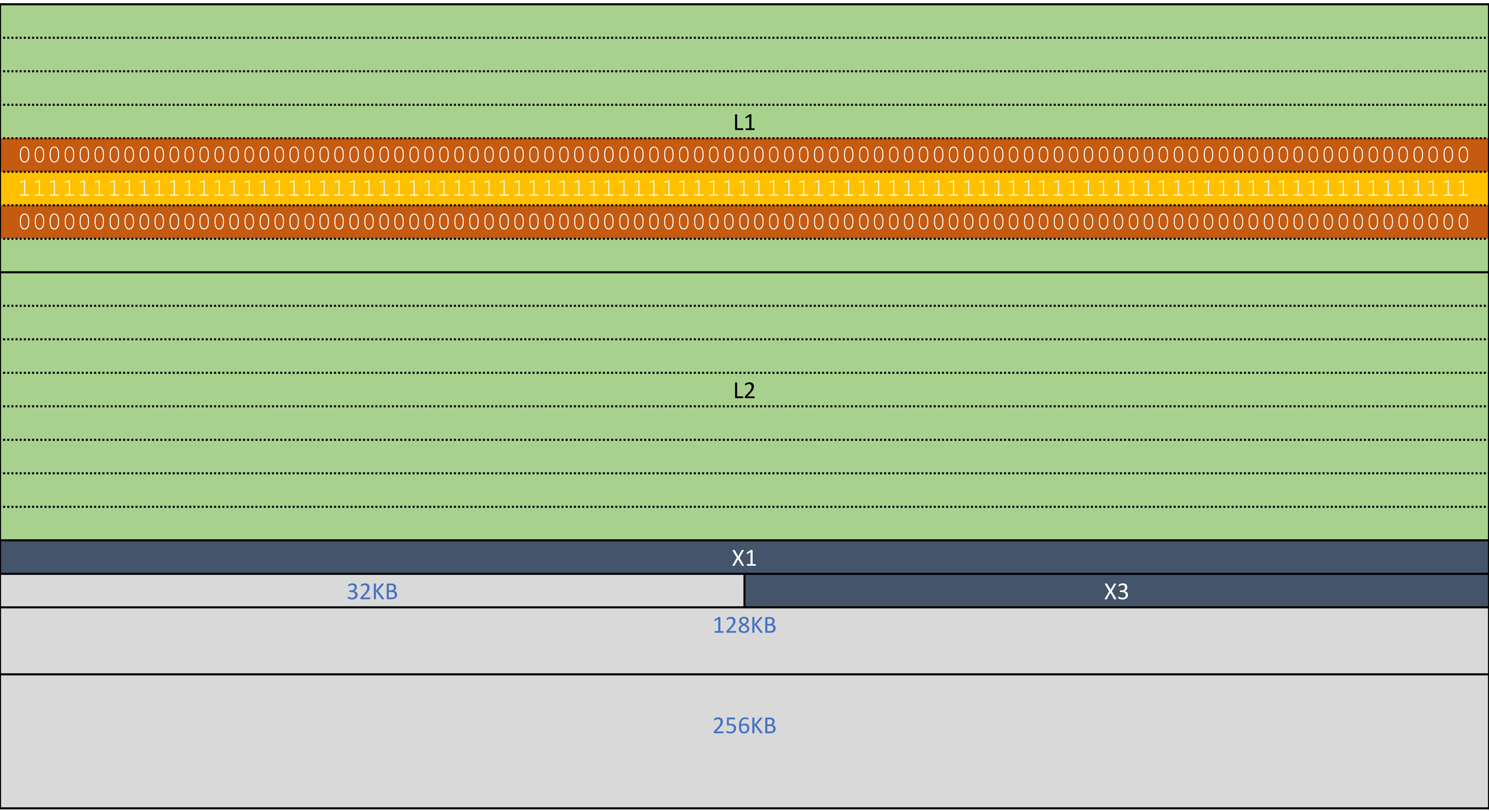
`Hammer(L1, 5);` `// hammer row 5 of chunk L1`



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

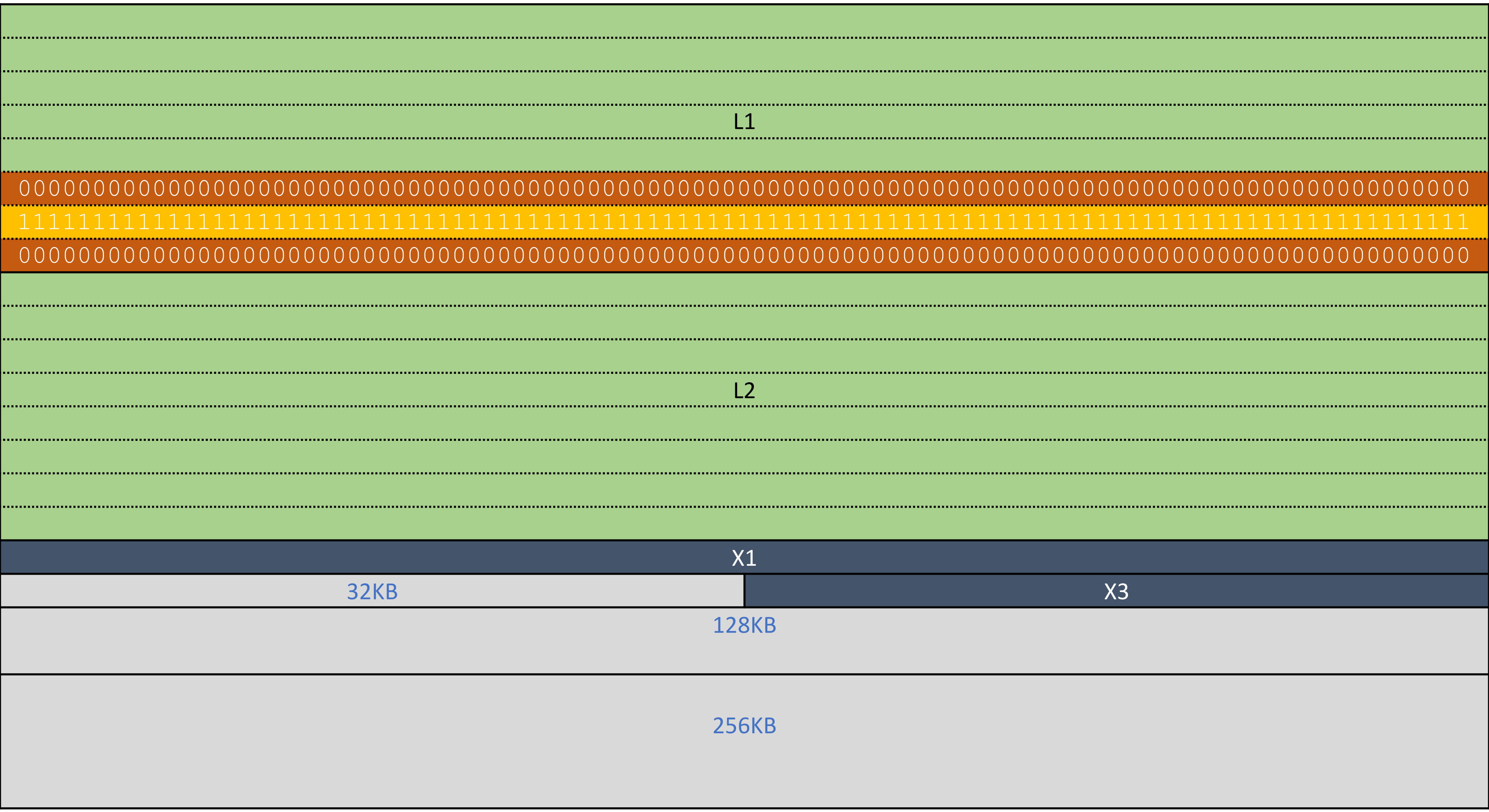
```
Hammer(L1, 6); // hammer row 6 of chunk L1
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

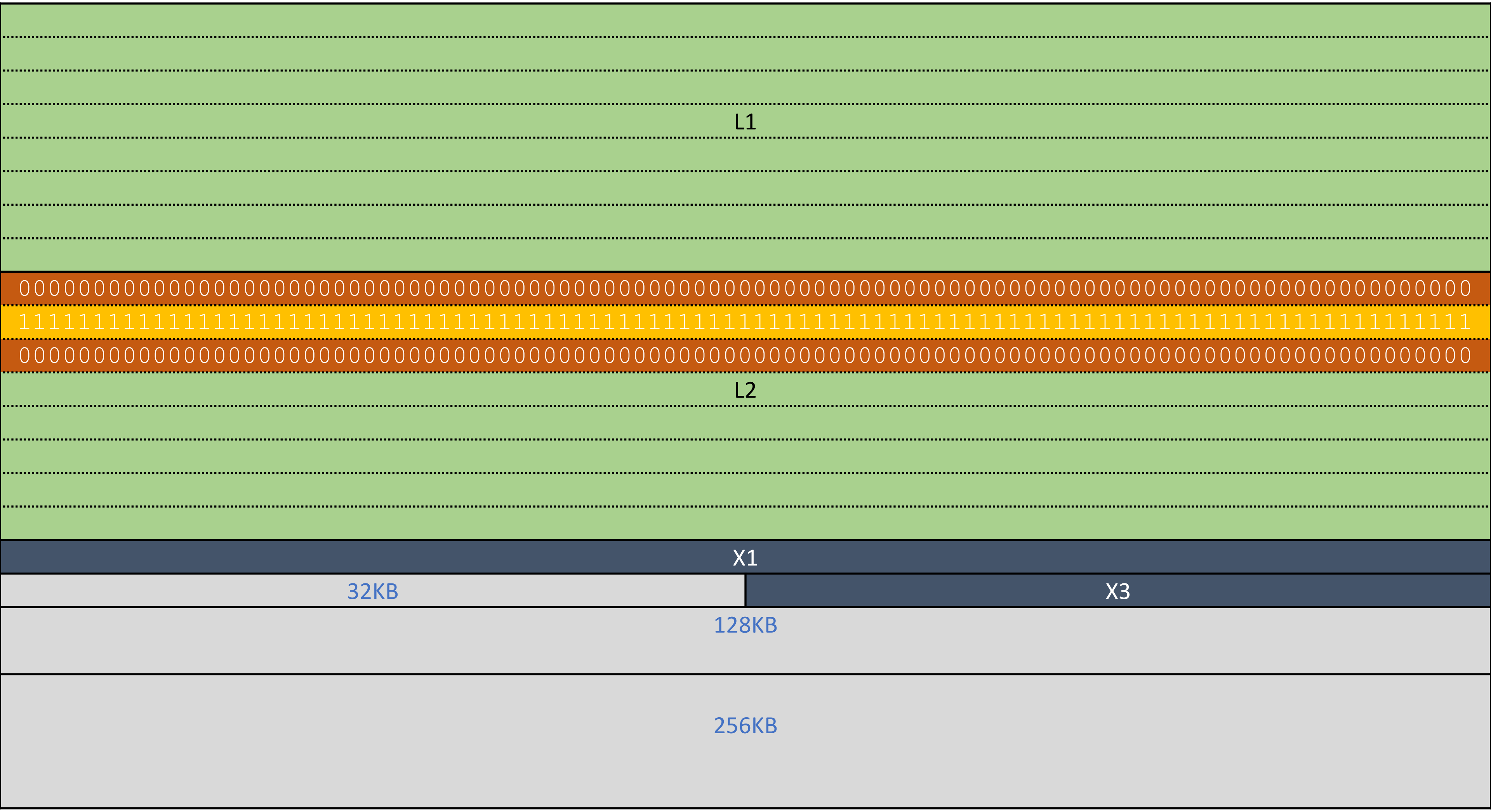
```
Hammer(L1, 7); // hammer row 7 of chunk L1
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

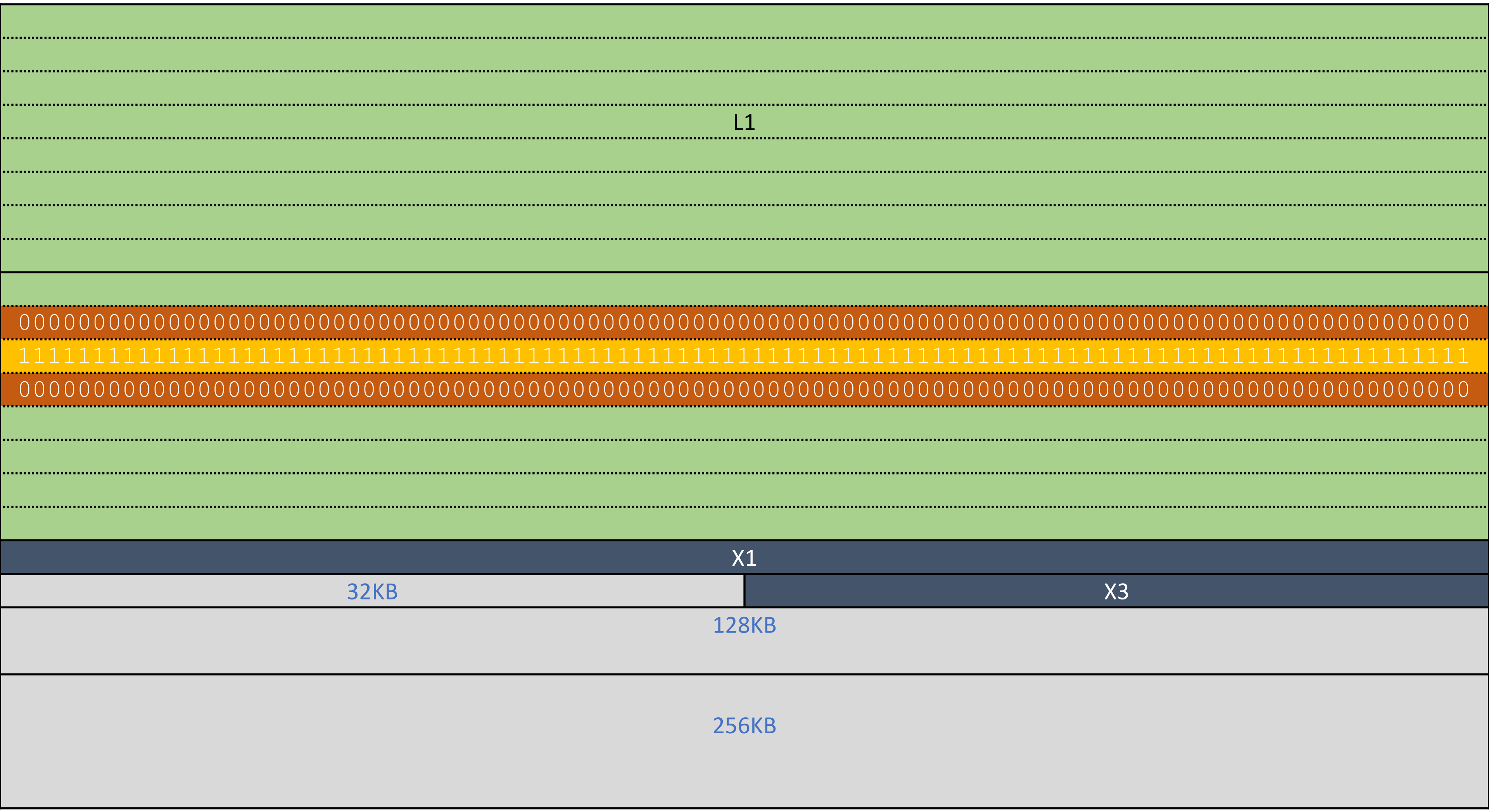
```
Hammer(L2, 2); // hammer row 2 of chunk L2
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

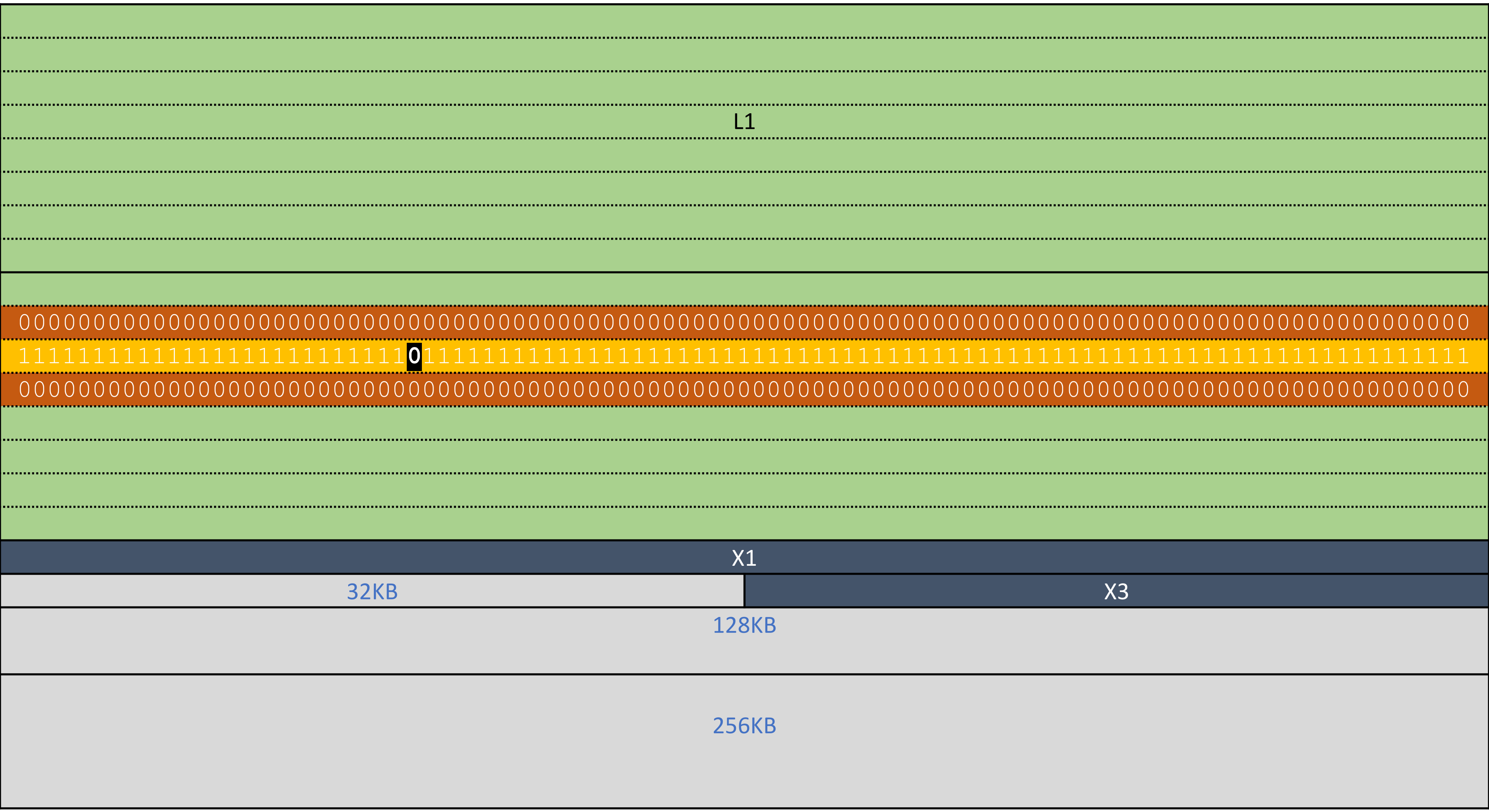
```
Hammer(L2, 3); // hammer row 3 of chunk L2
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

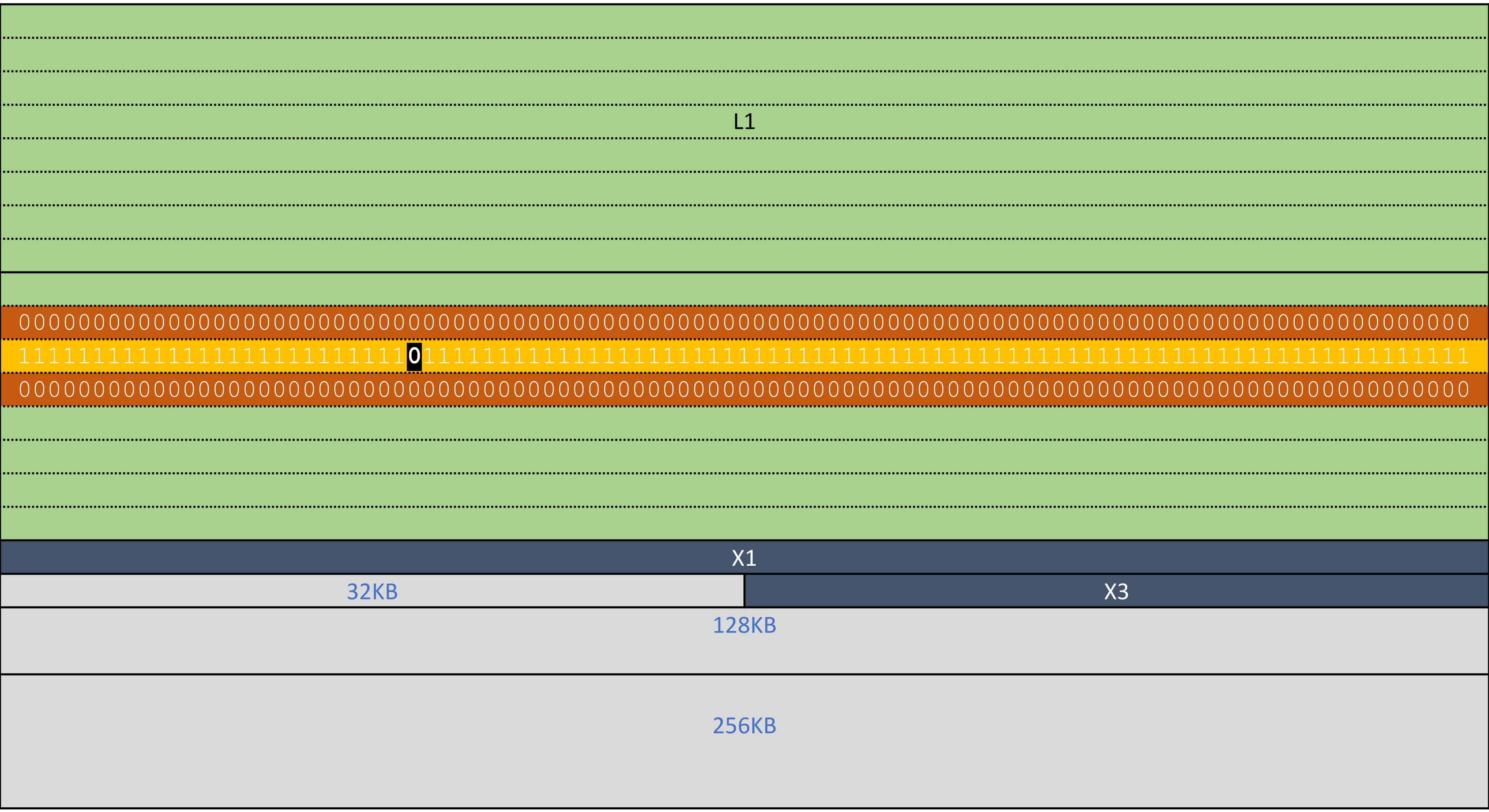
`“exploitable flip found in page 5 of virtual row 3 of L2!”`



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

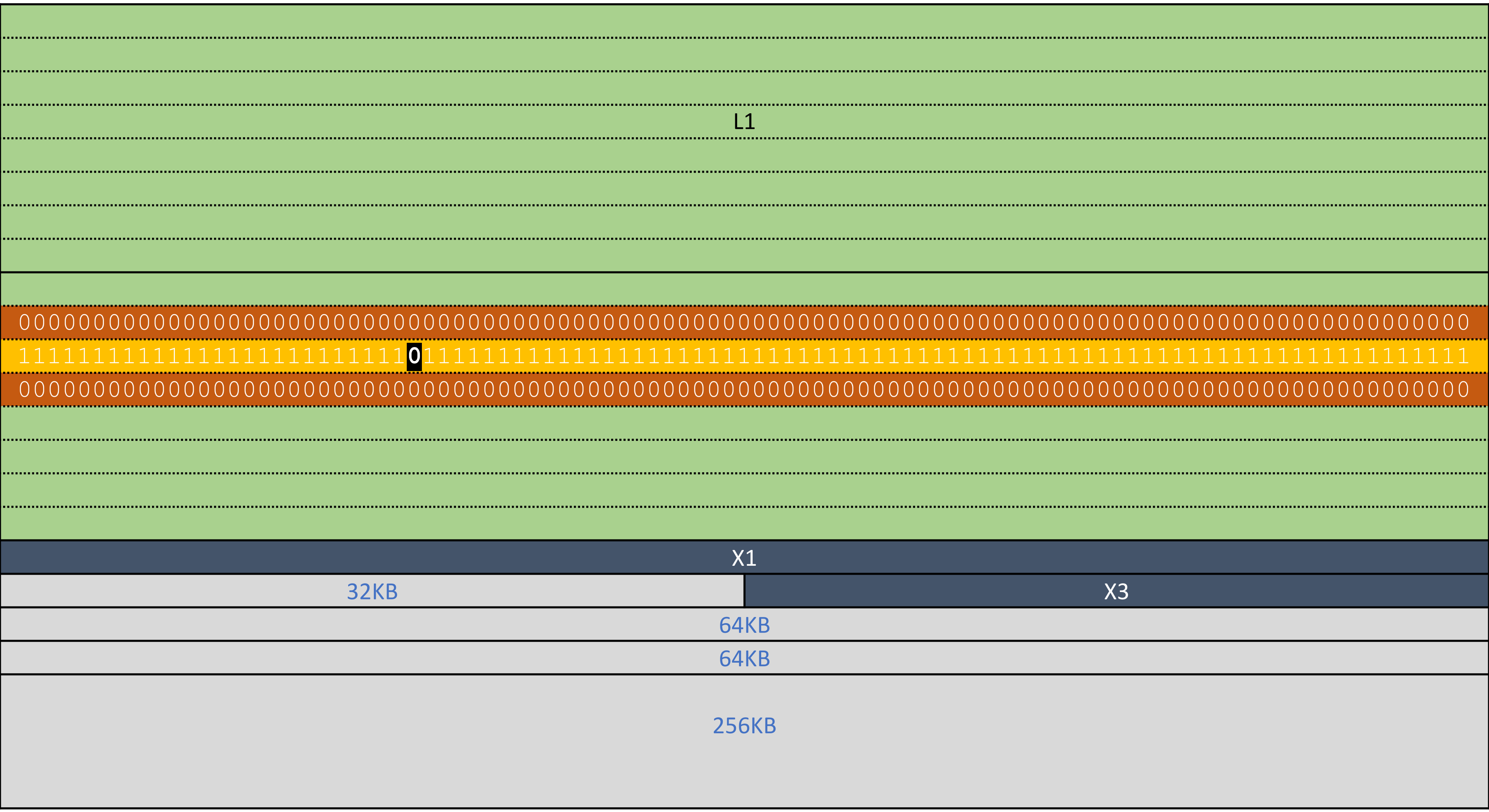
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

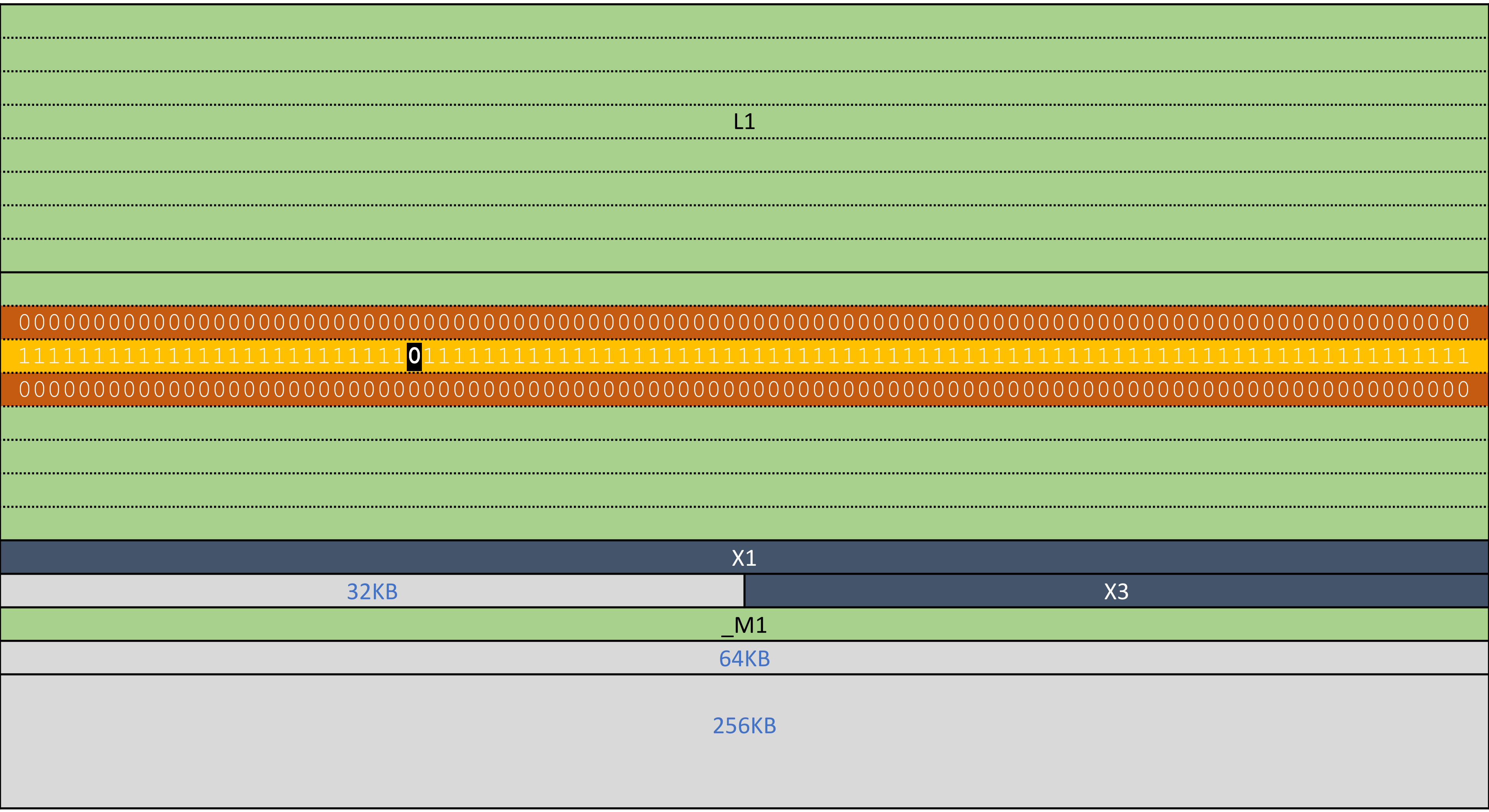
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

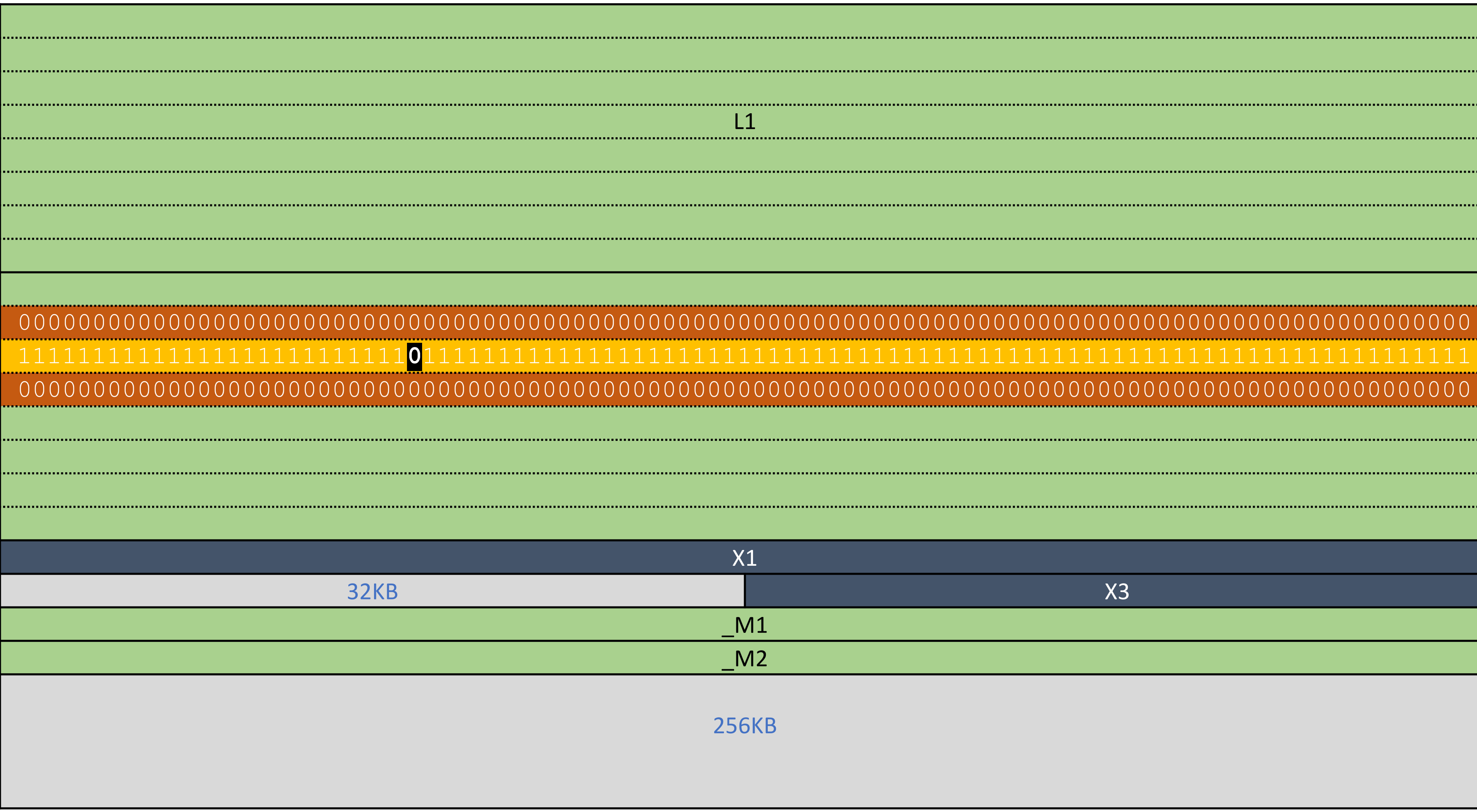
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

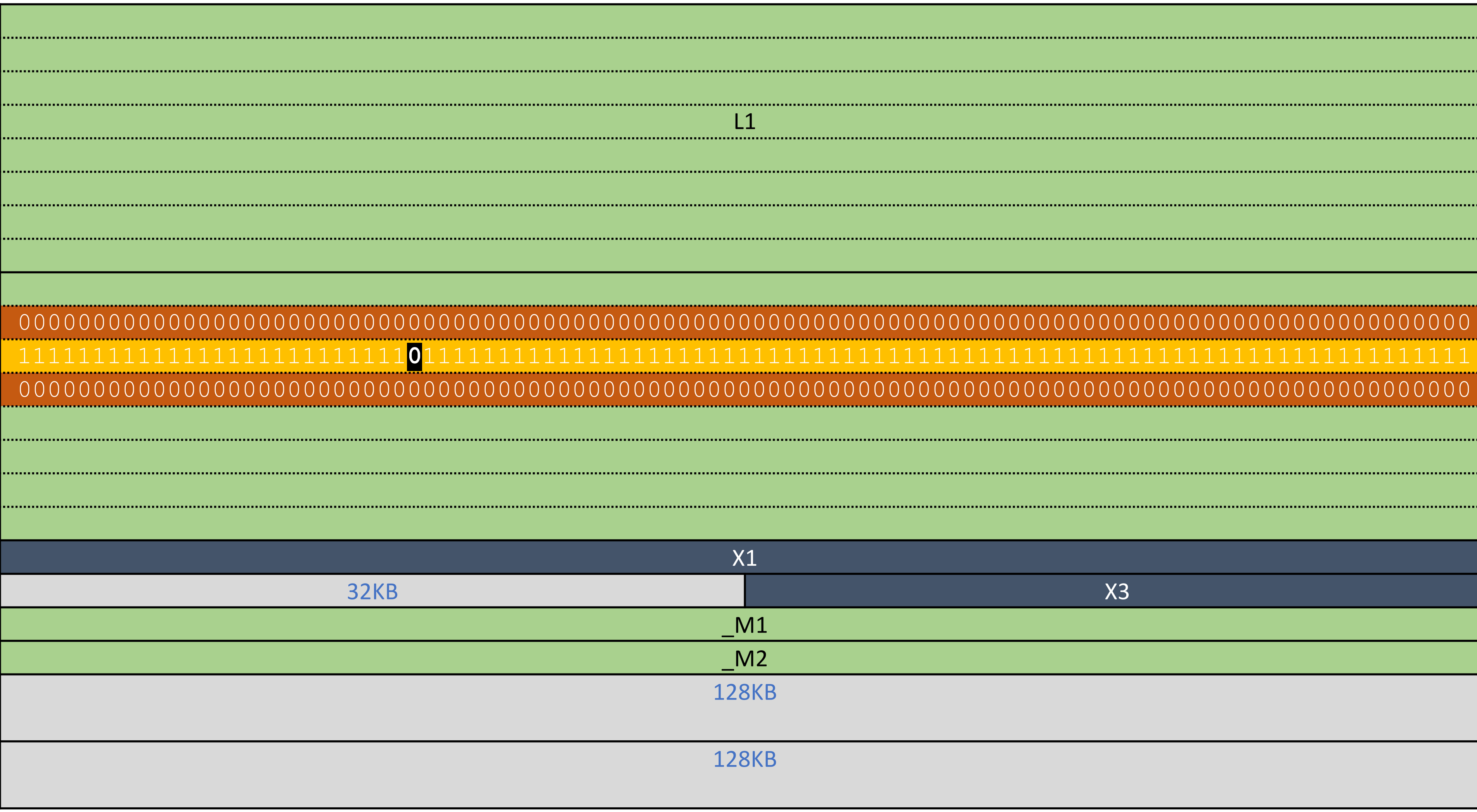
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

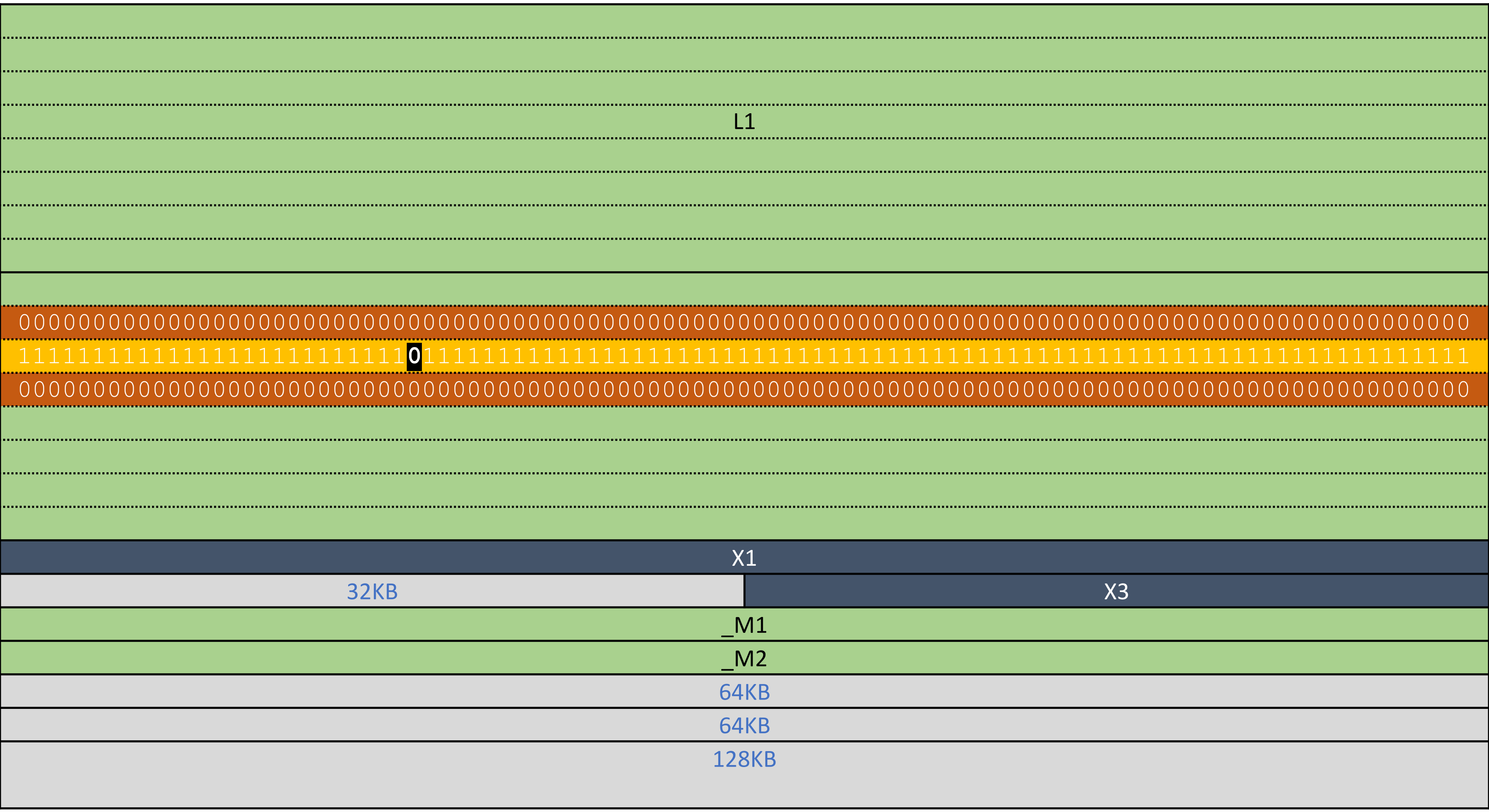
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

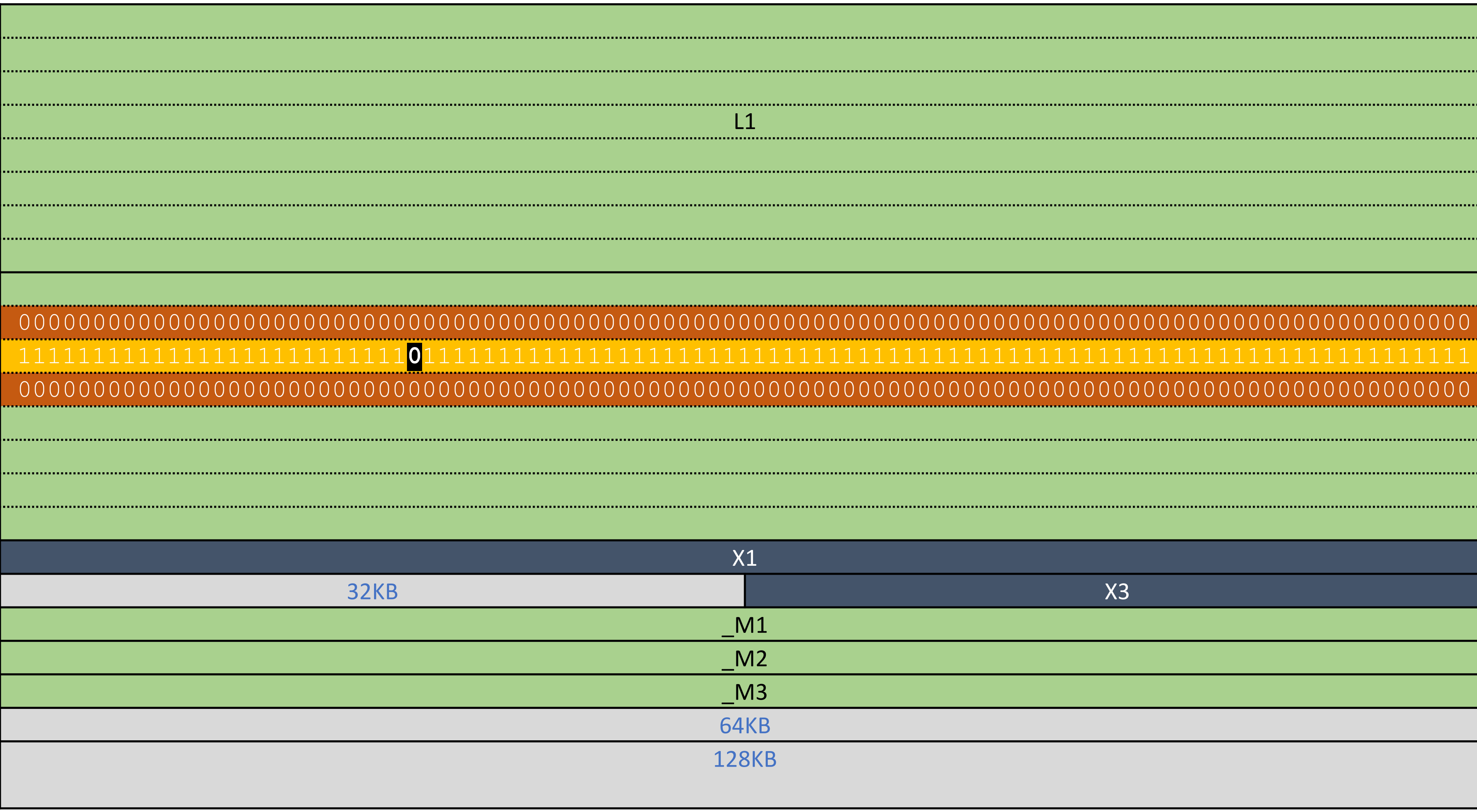
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

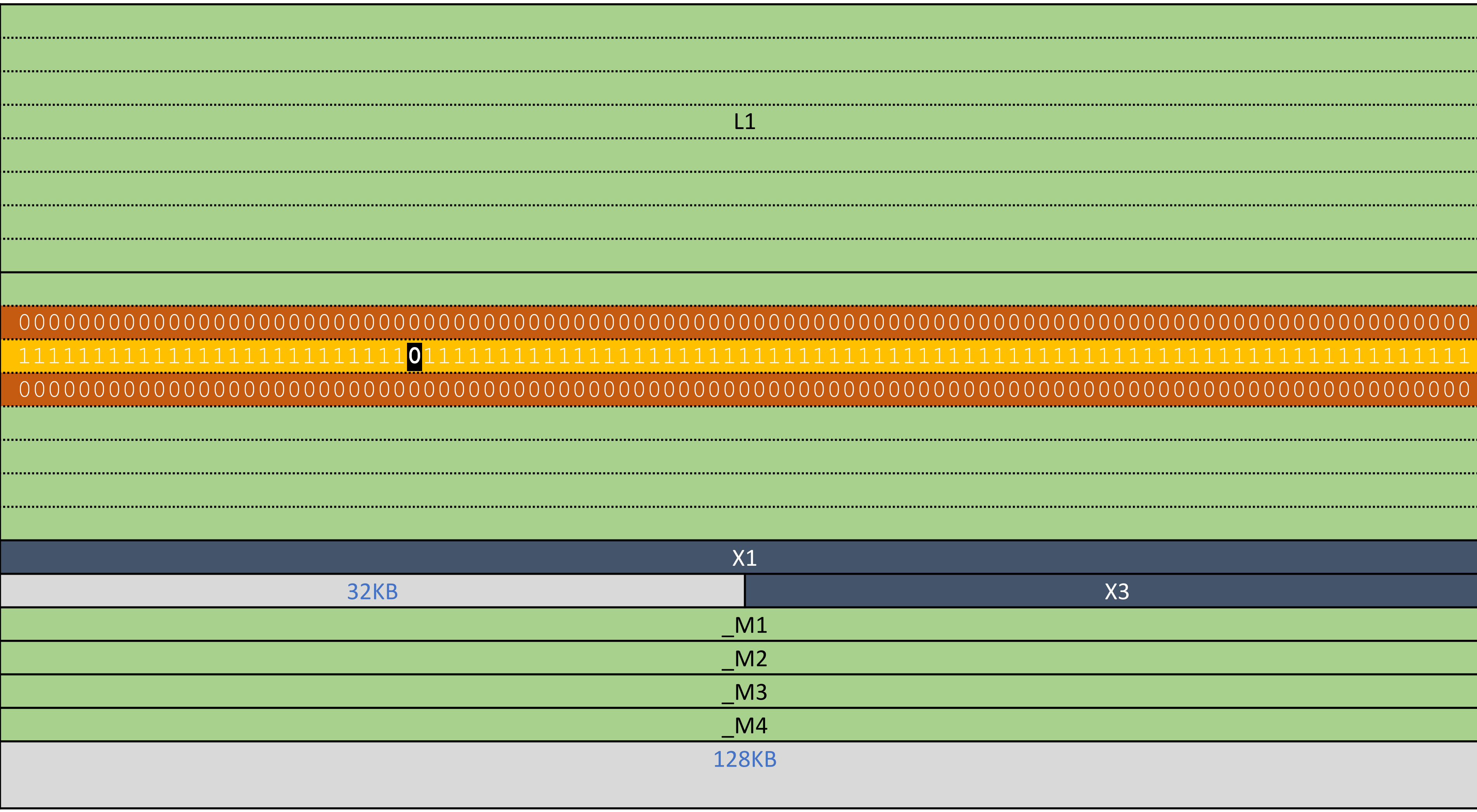
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

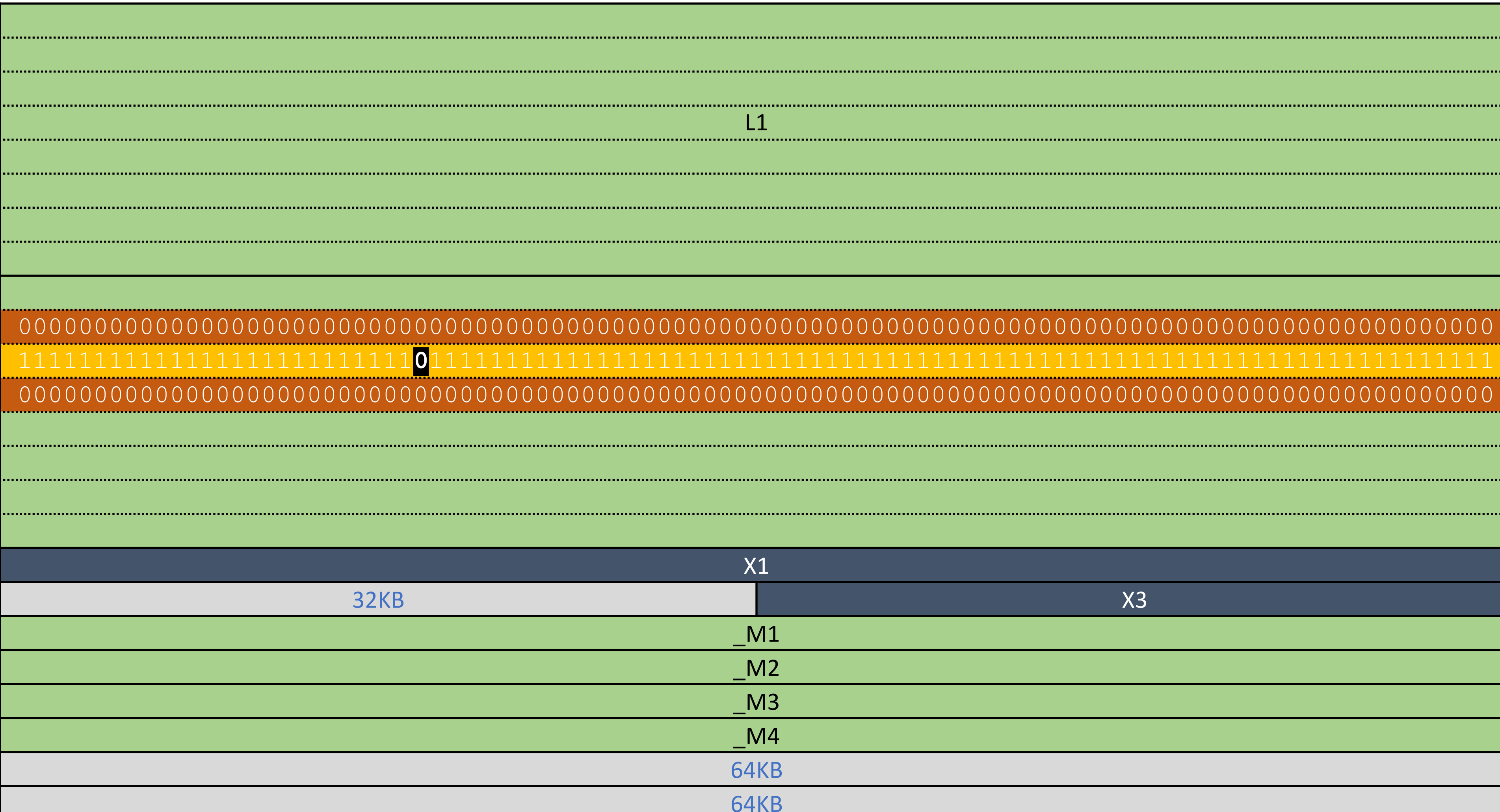
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

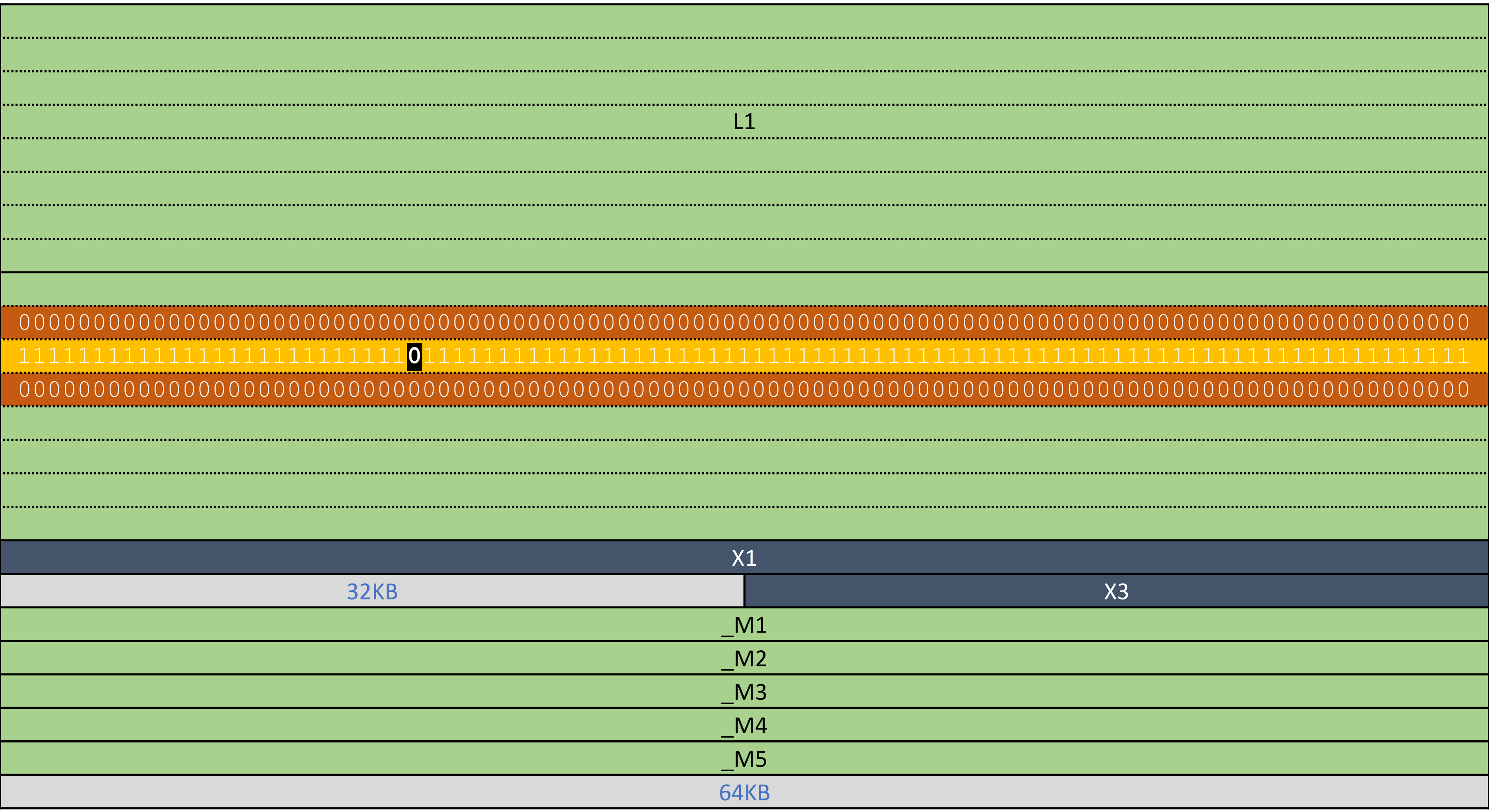
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

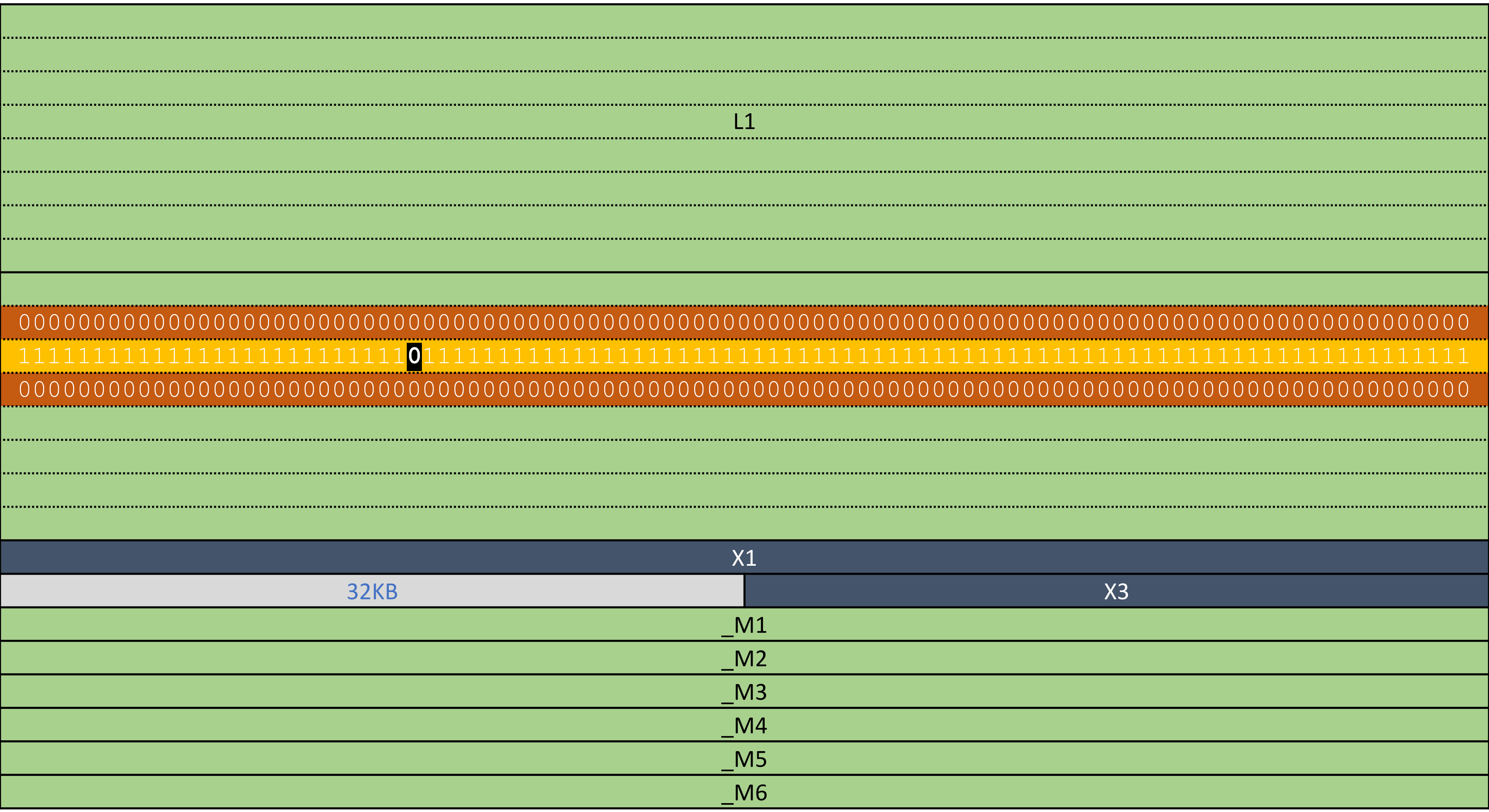
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

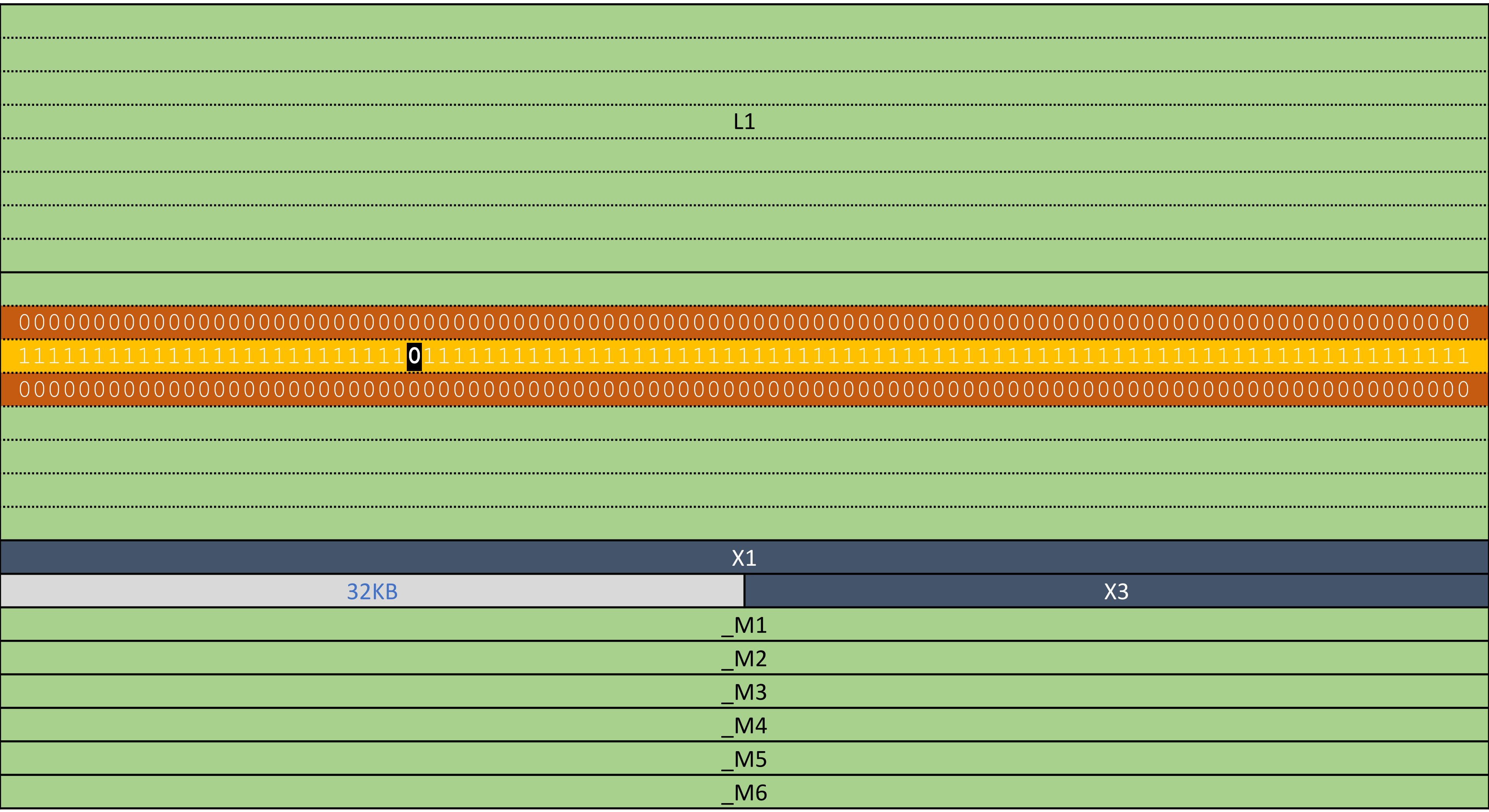
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 3/8

Release *Large* chunk with vulnerable page

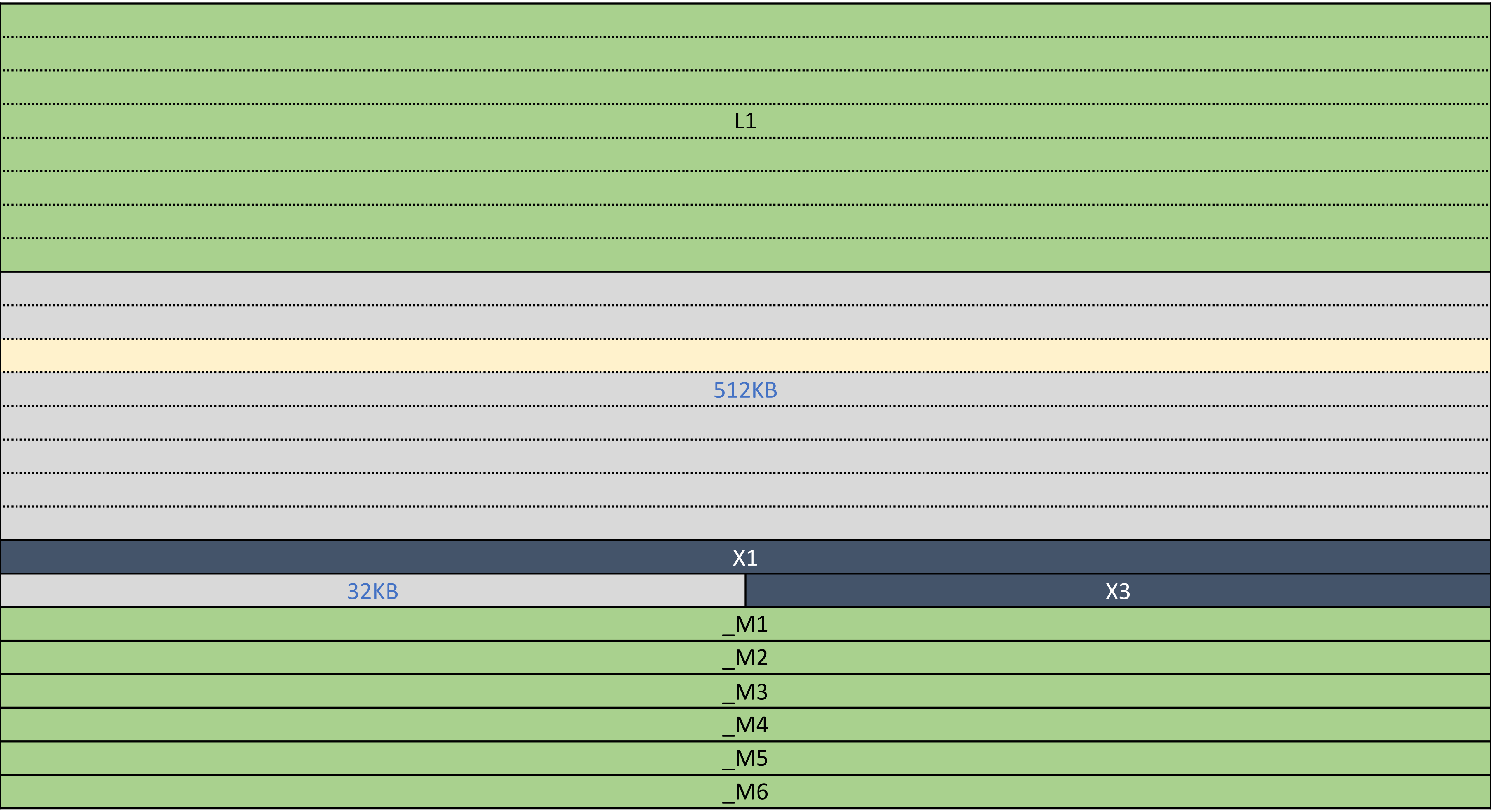
```
Release(L2); // L chunk with vulnerable page
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

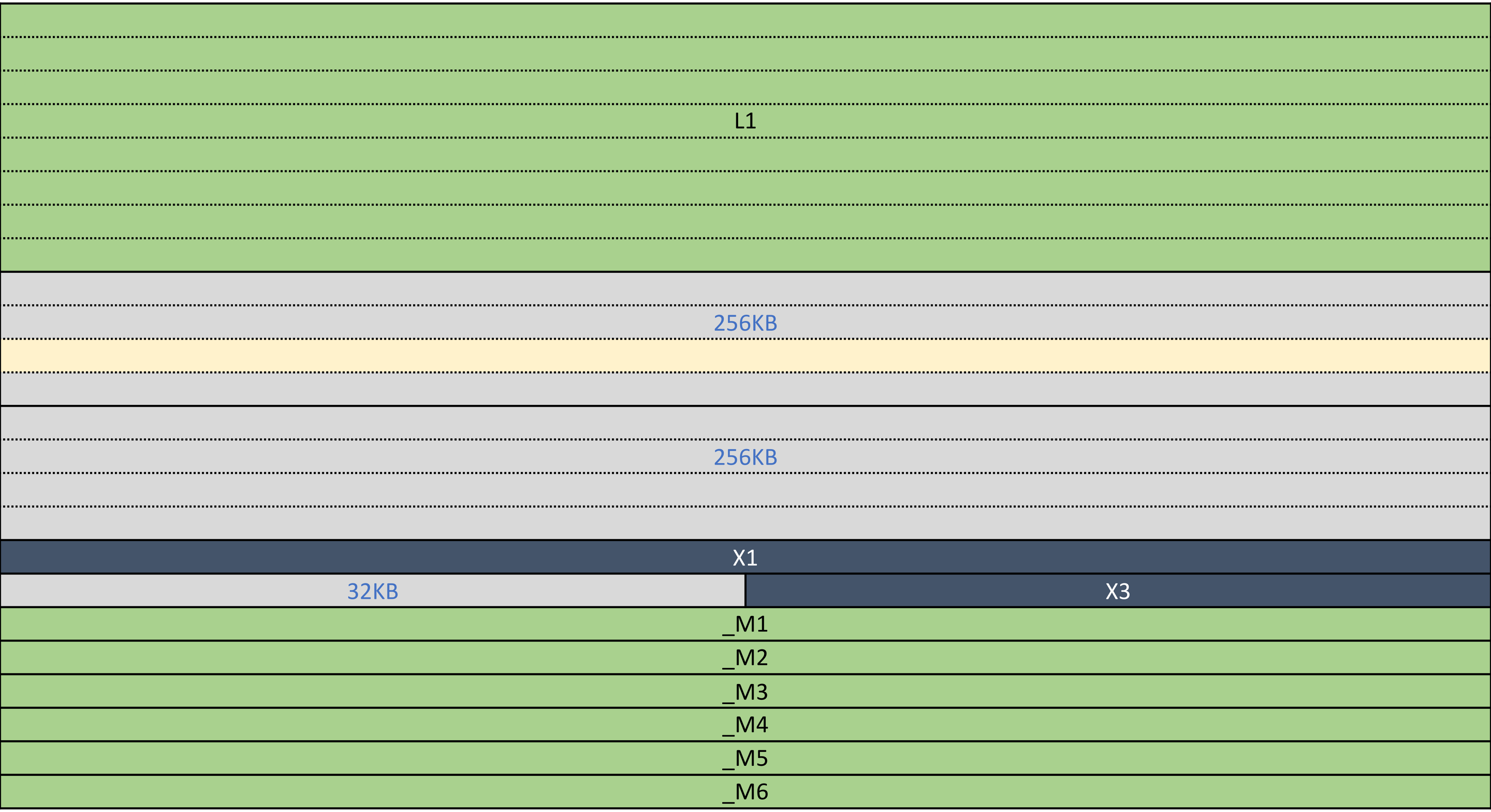
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

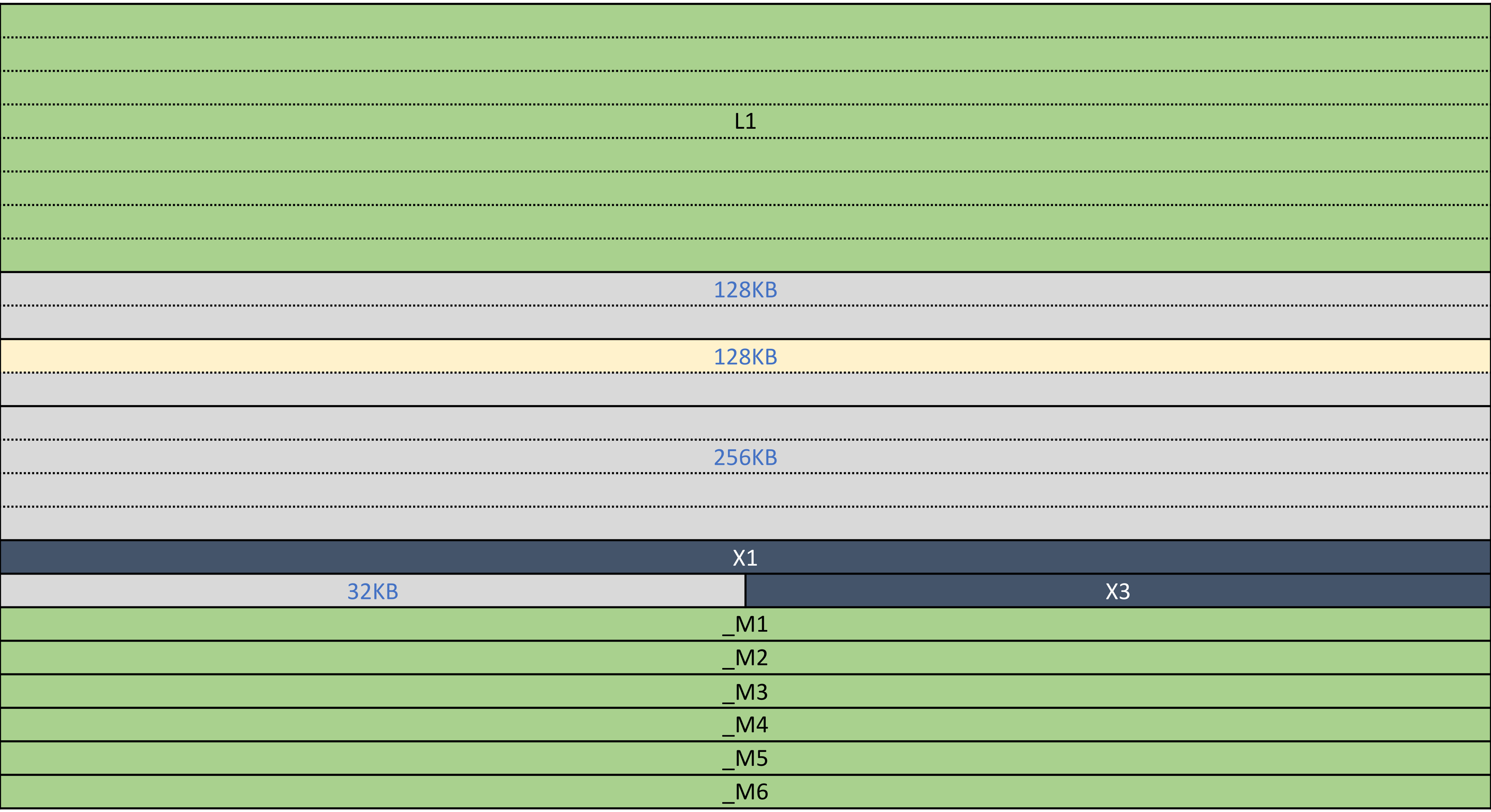
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

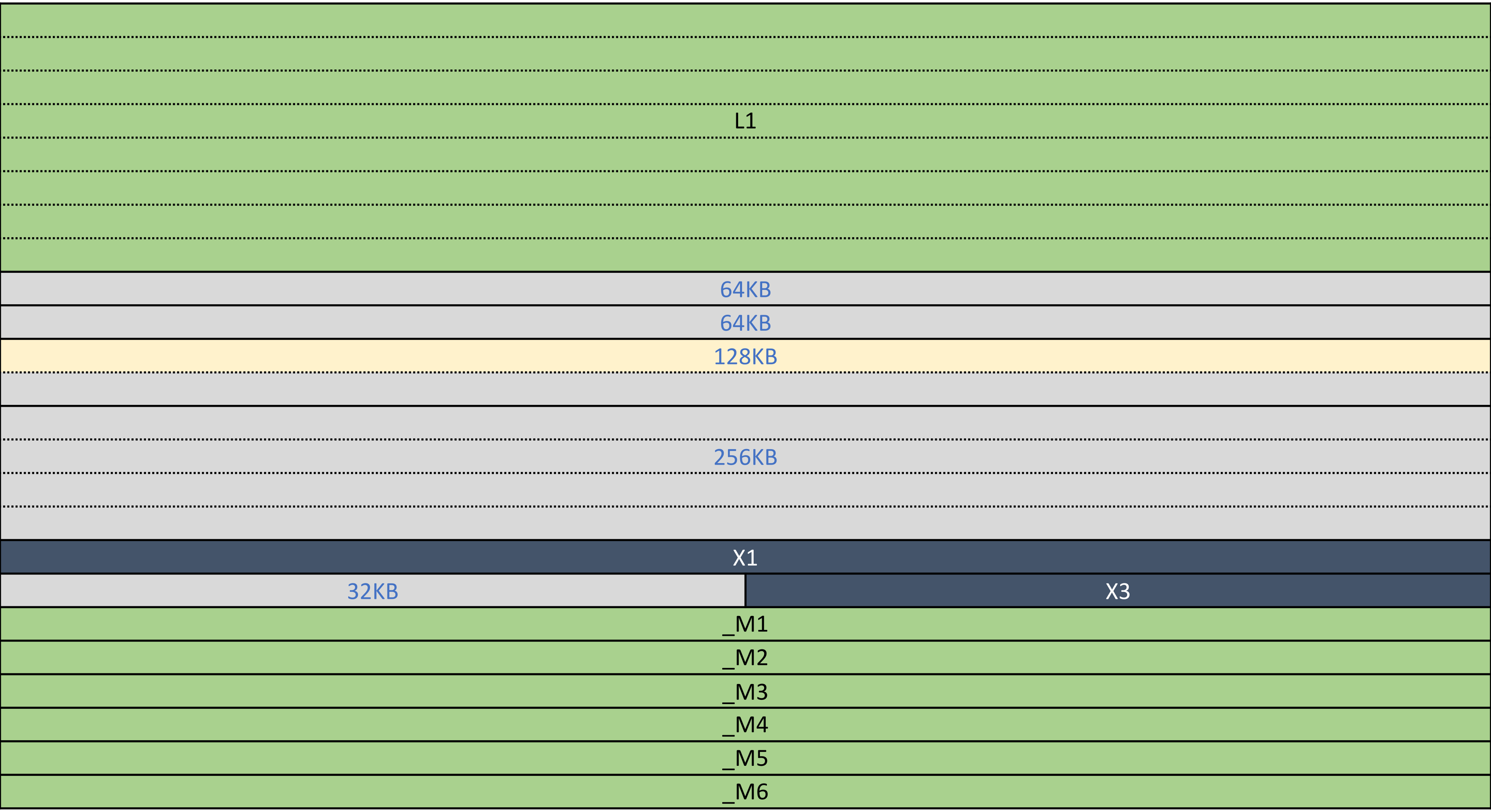
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

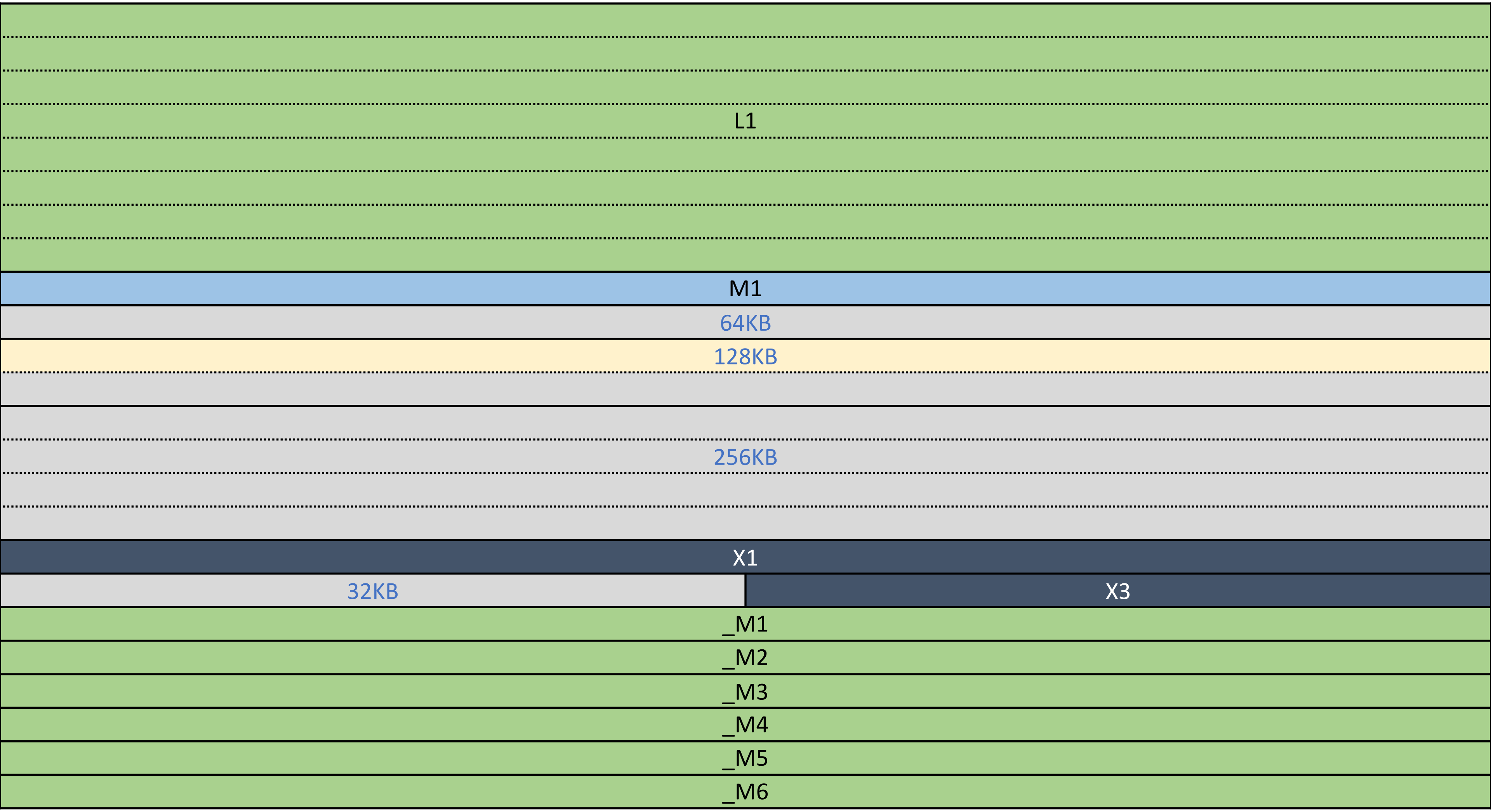
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

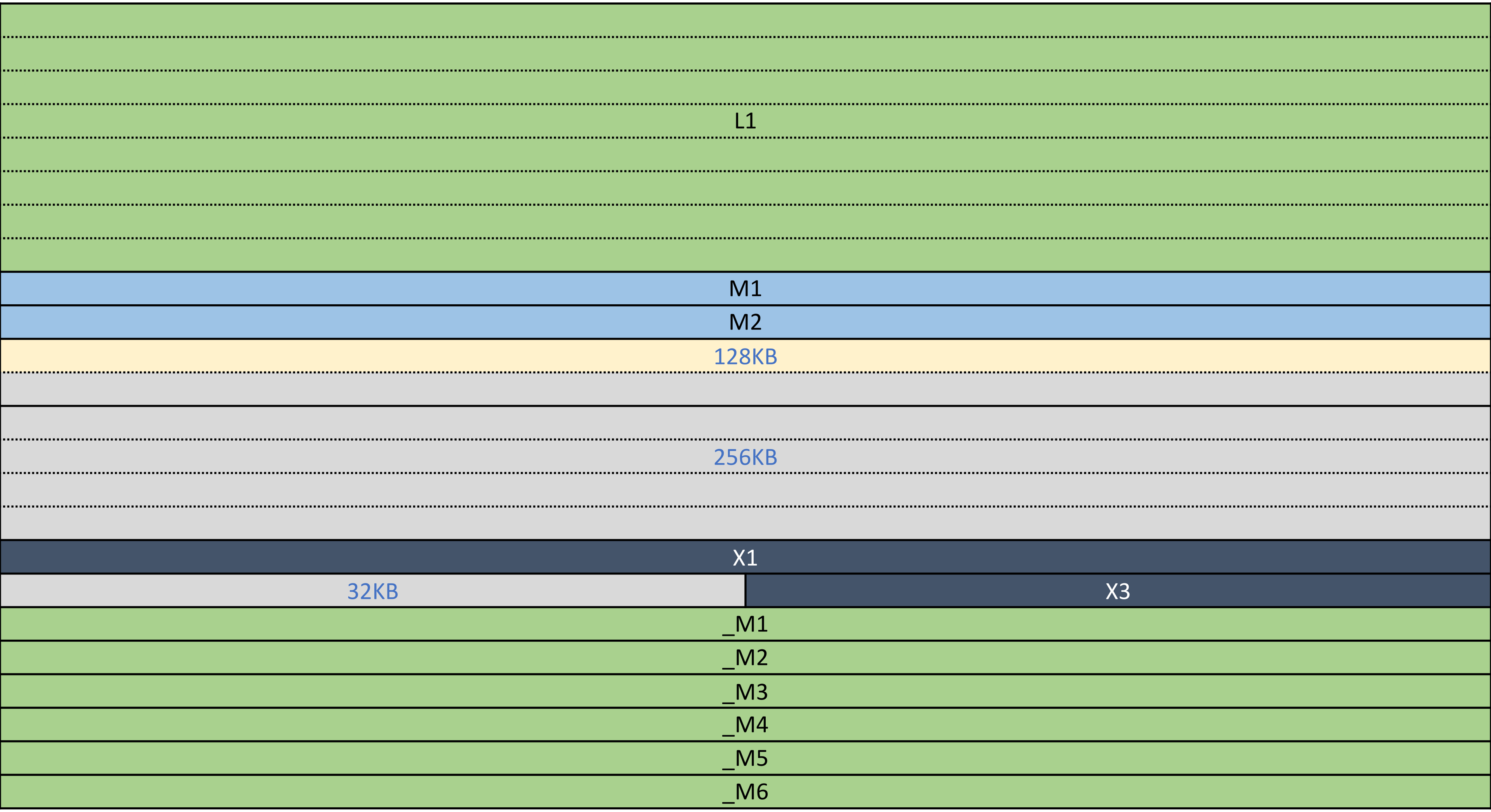
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

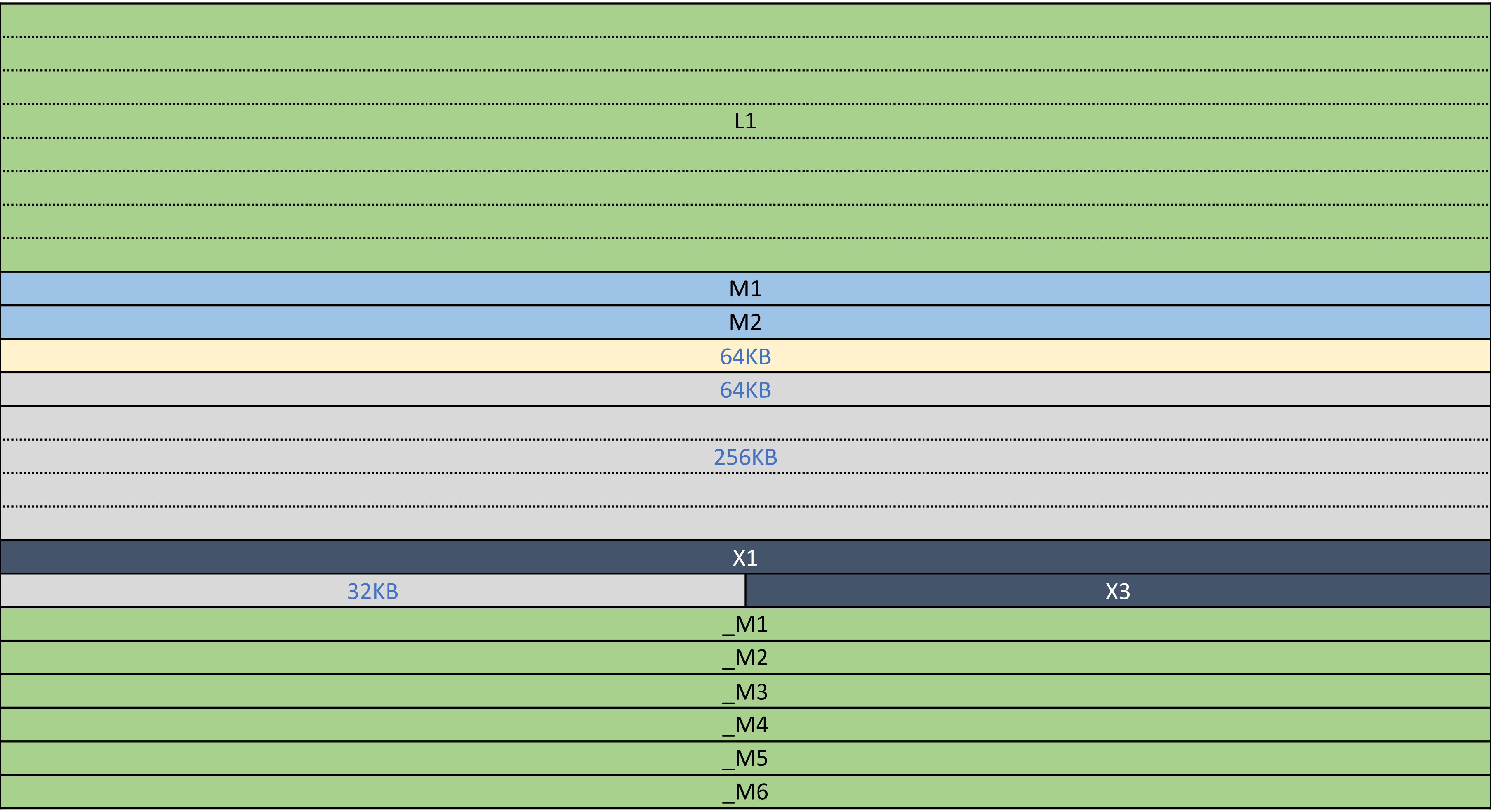
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

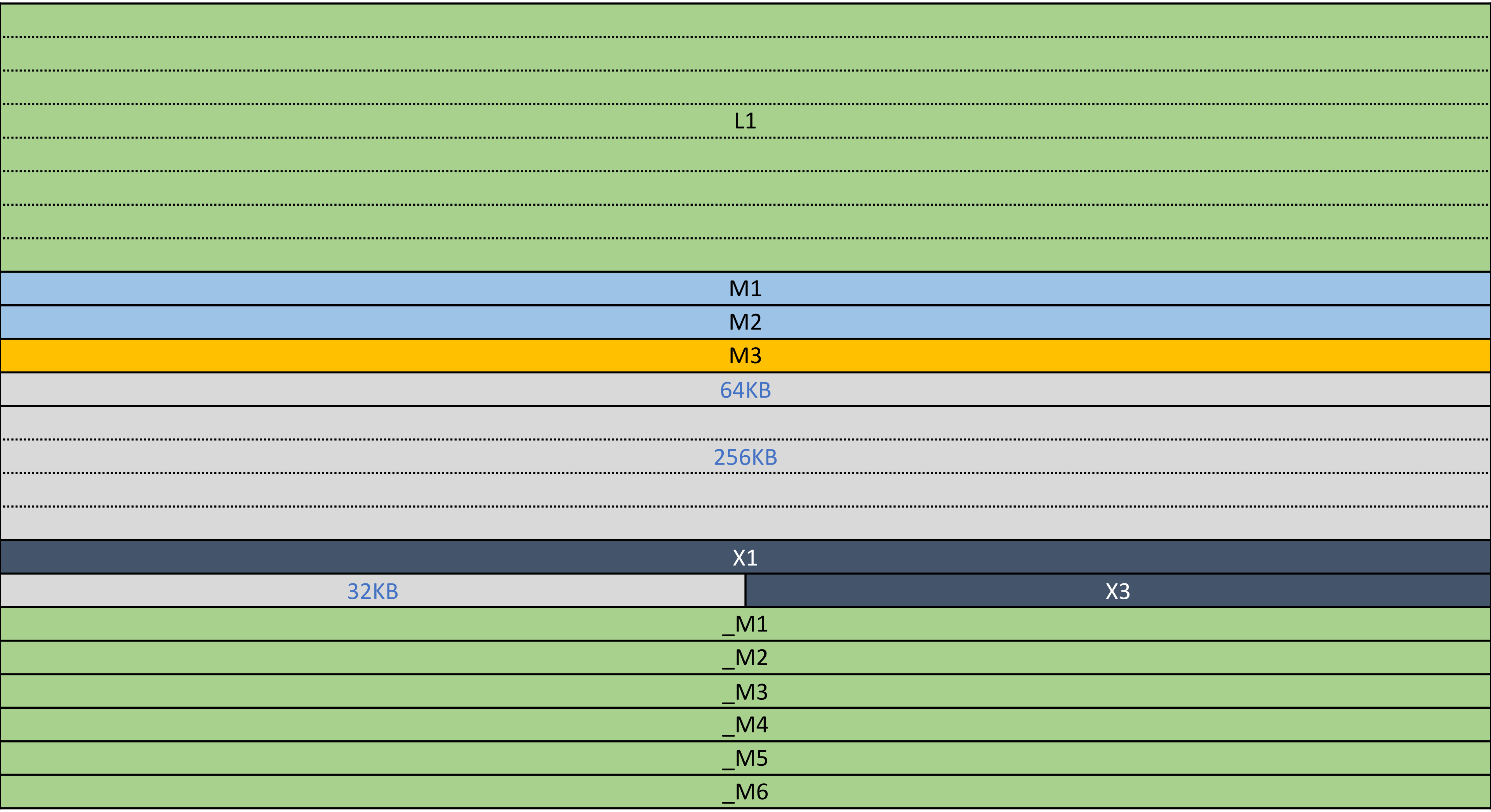
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

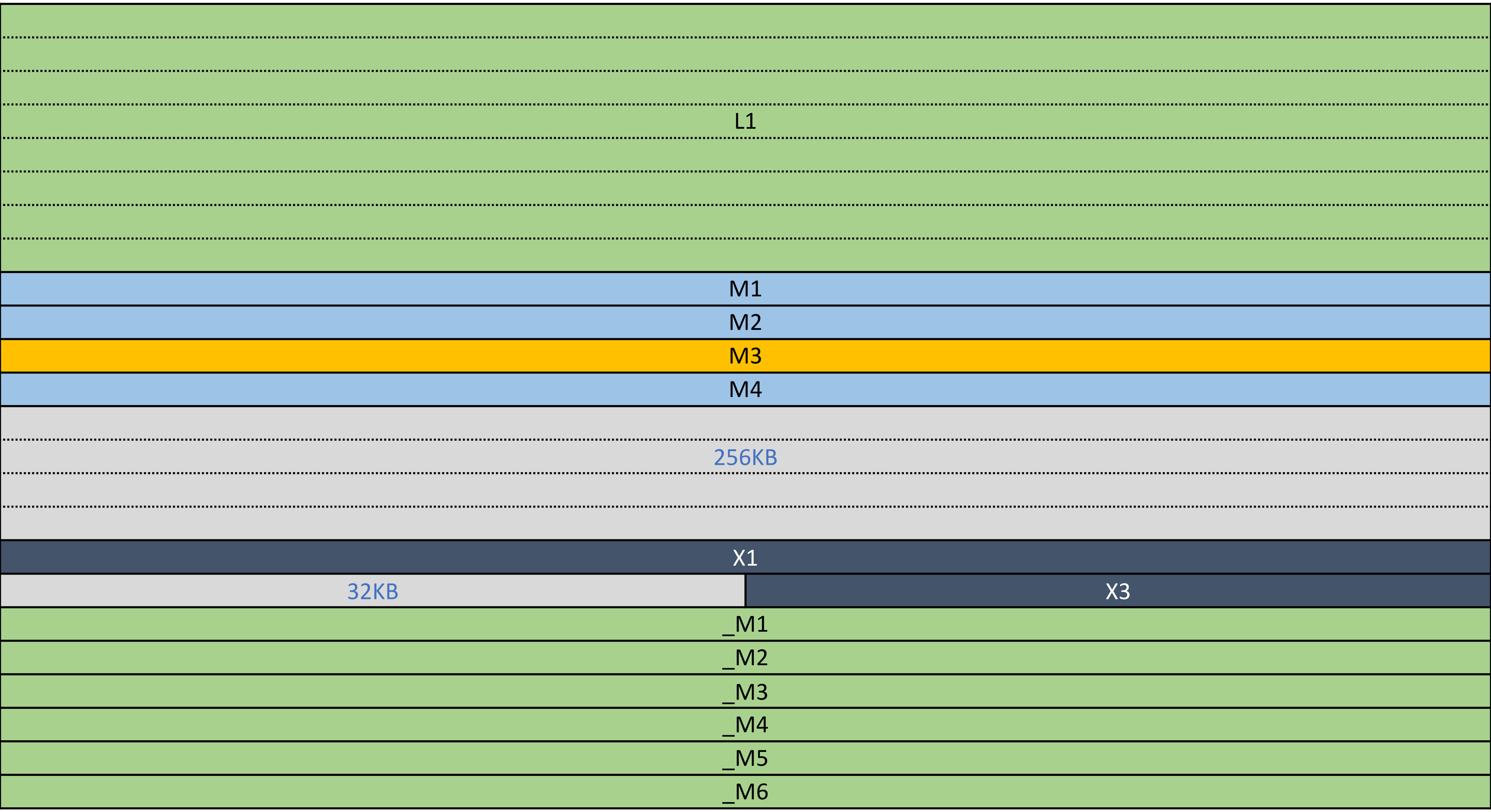
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

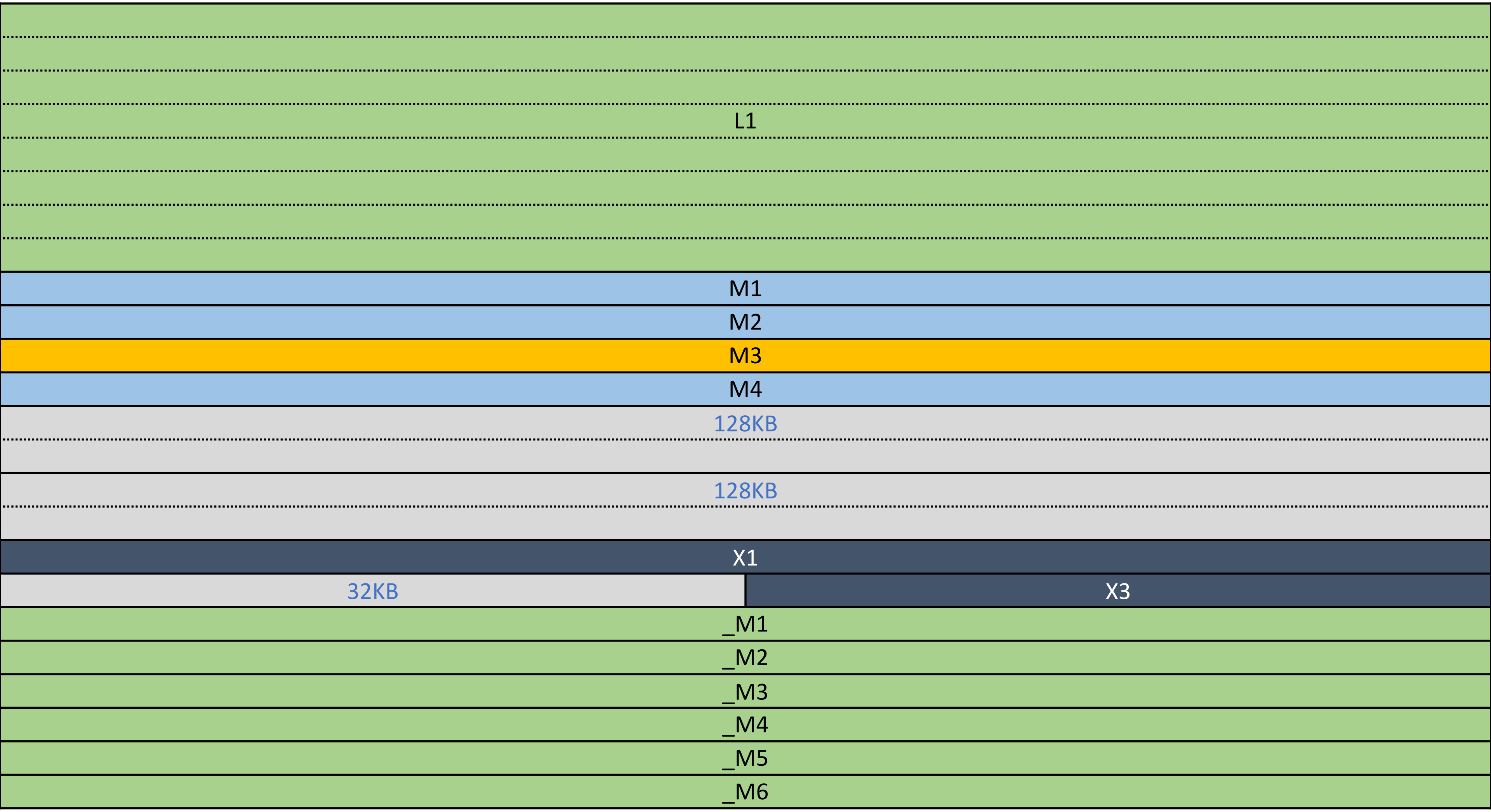
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

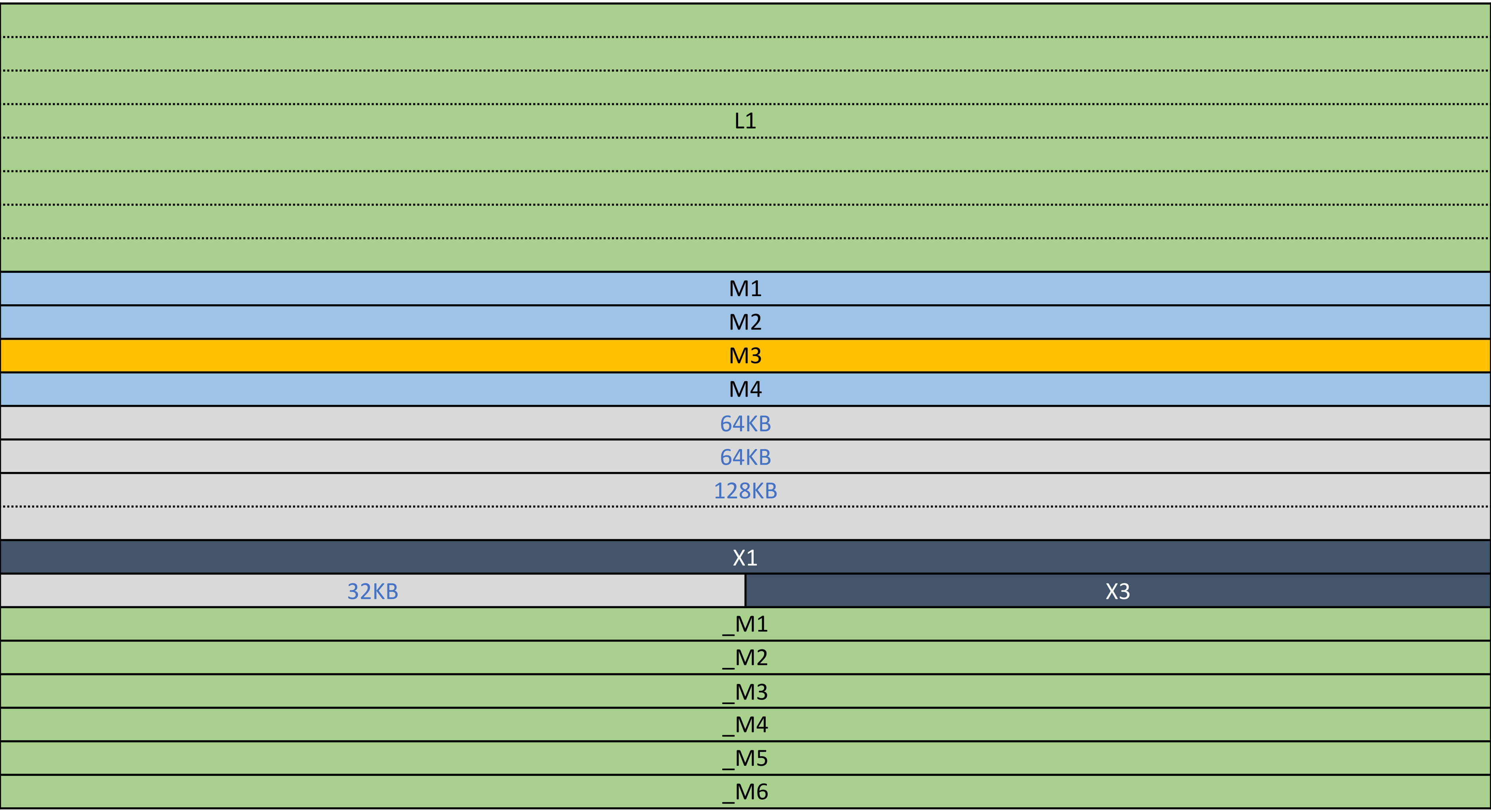
`M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks`



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

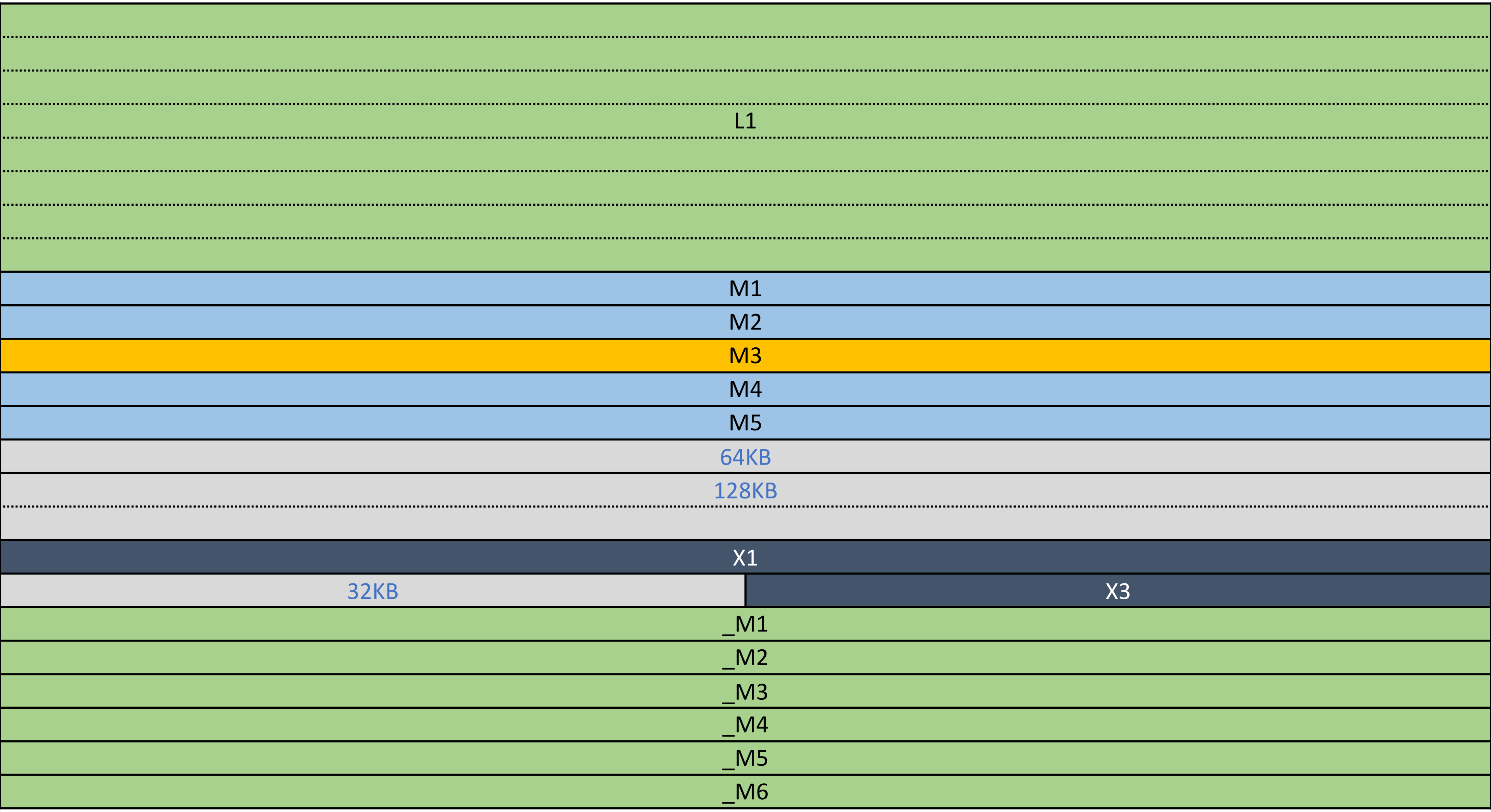
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

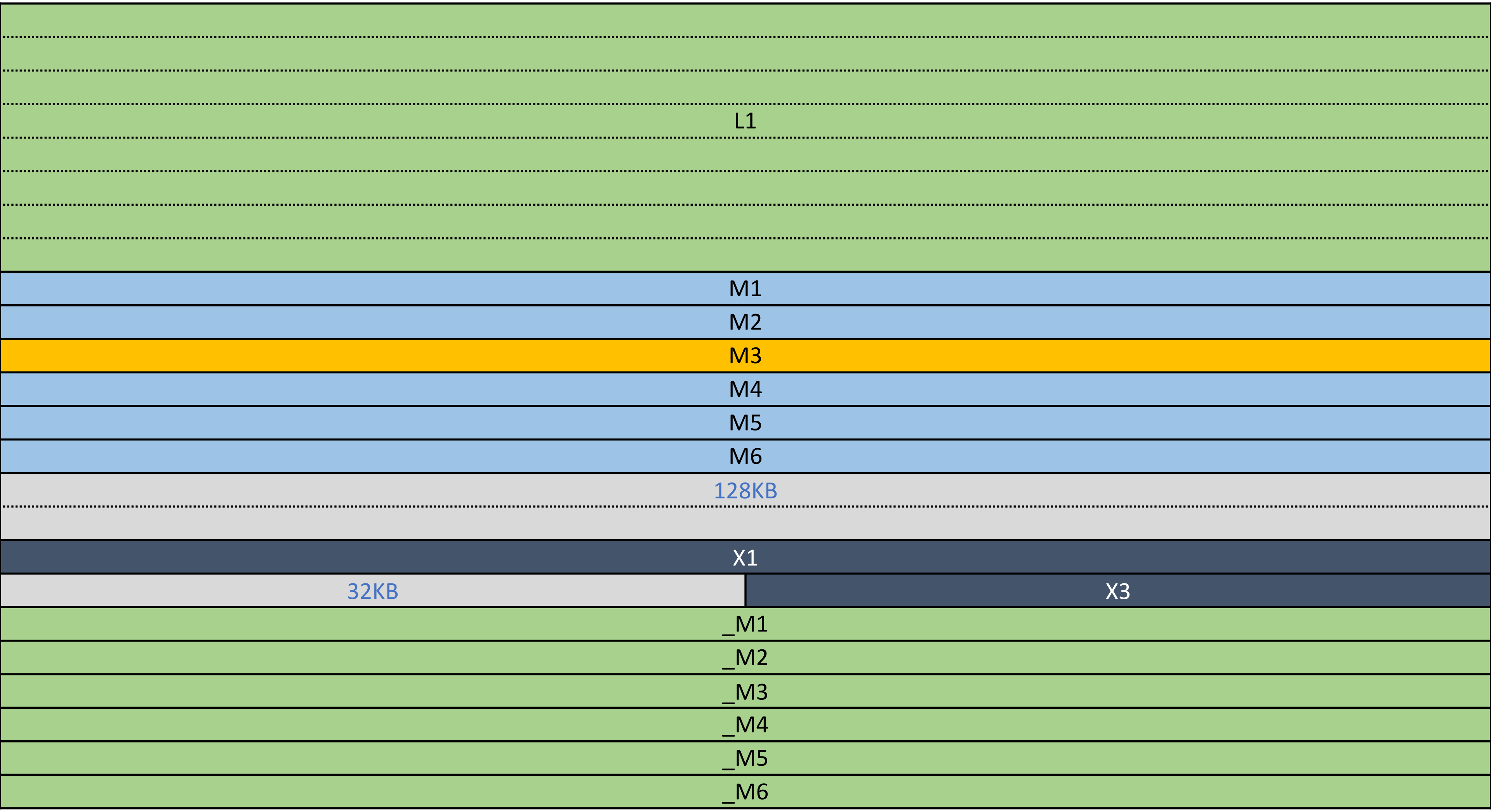
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

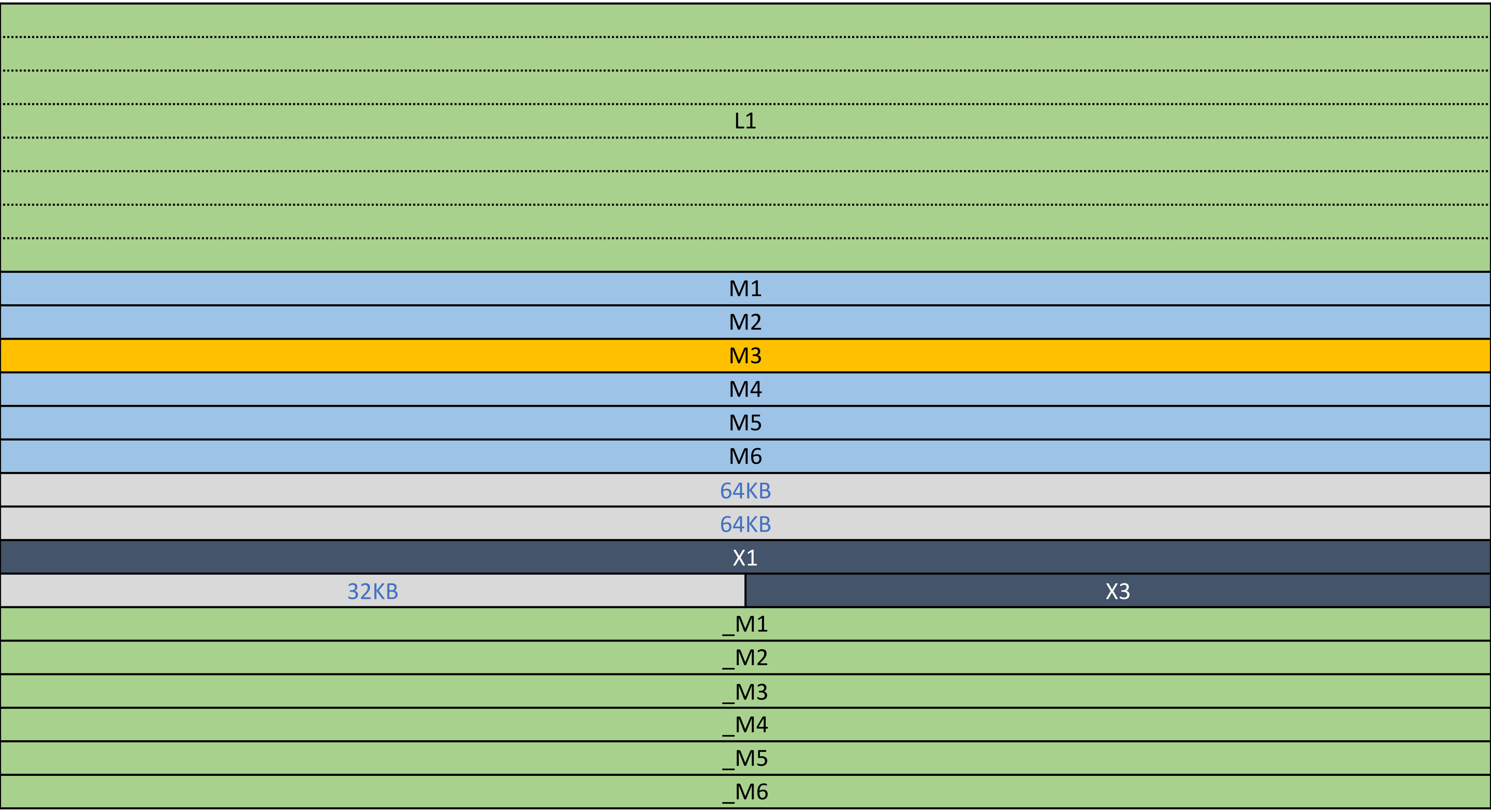
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

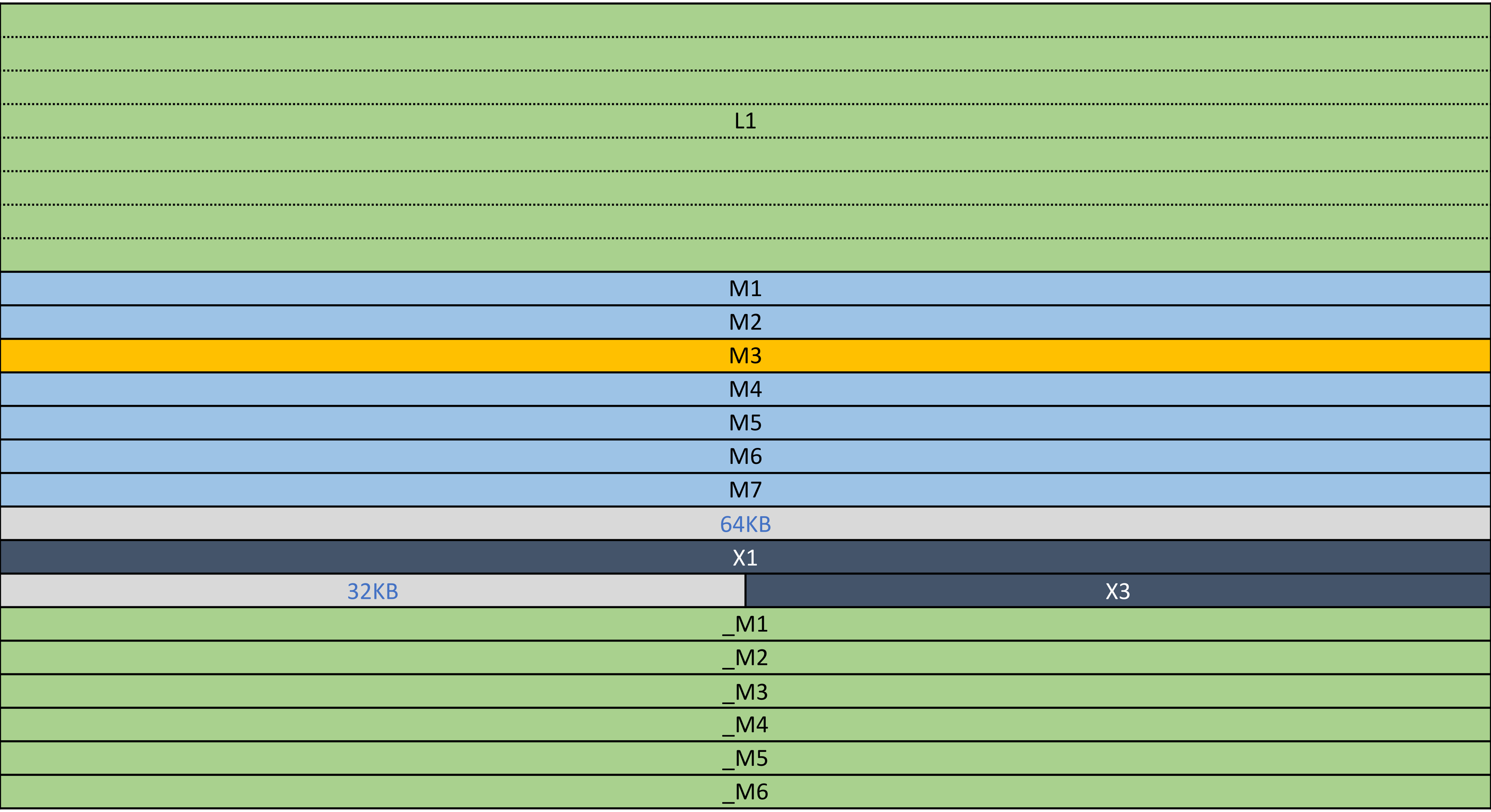
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

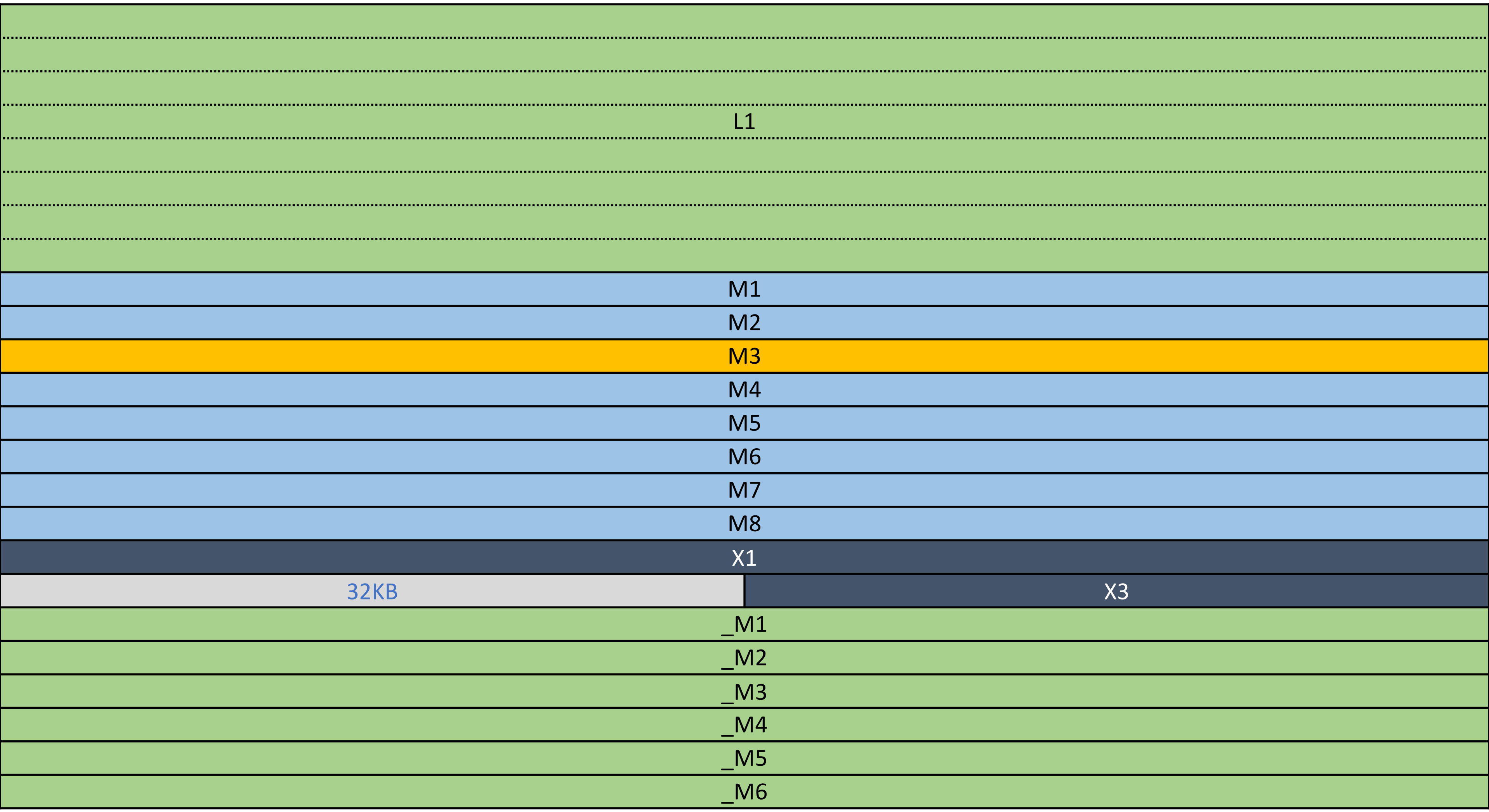
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

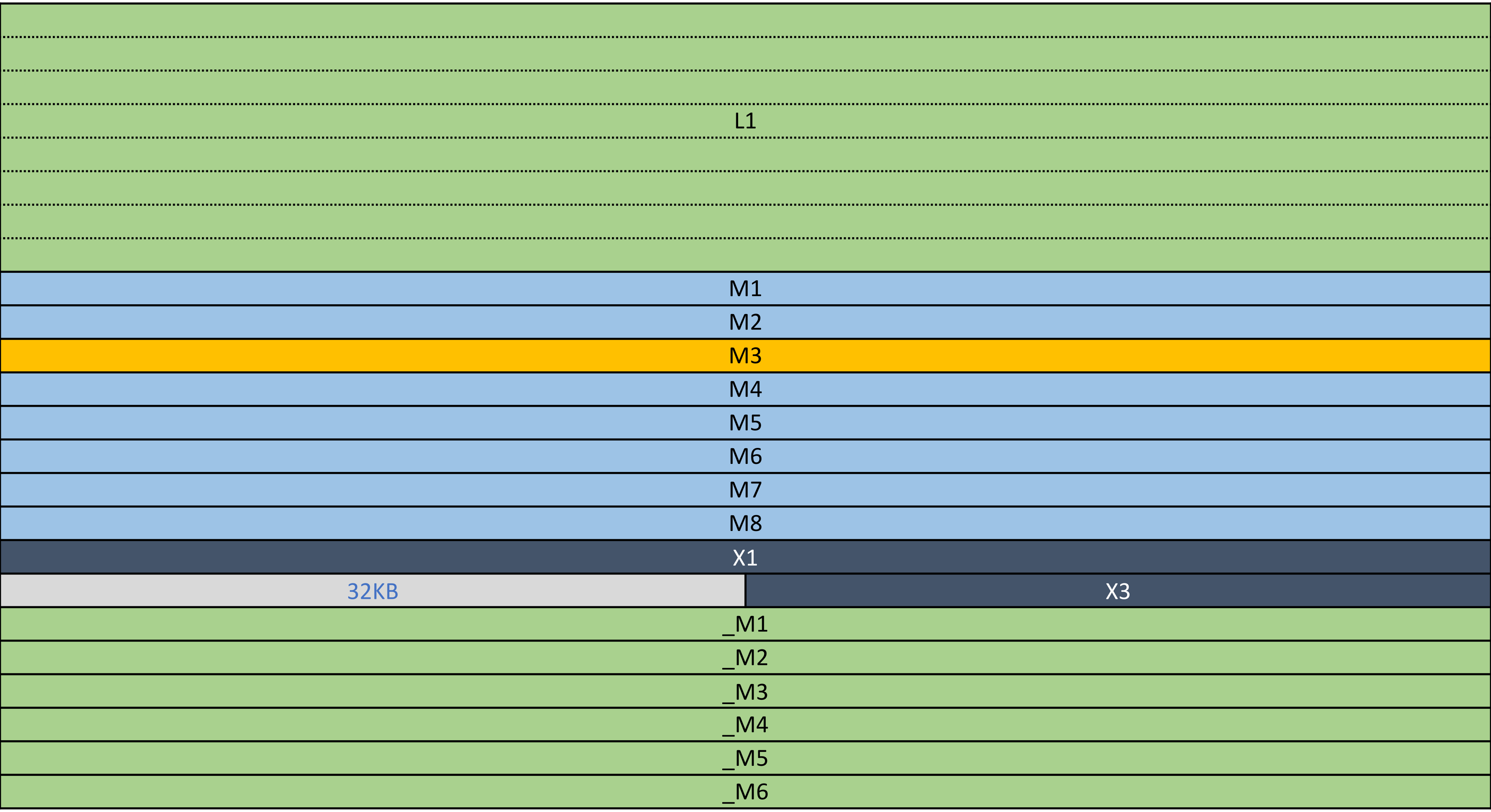
Exhaust *Medium-sized* chunks (again)

```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 5/8

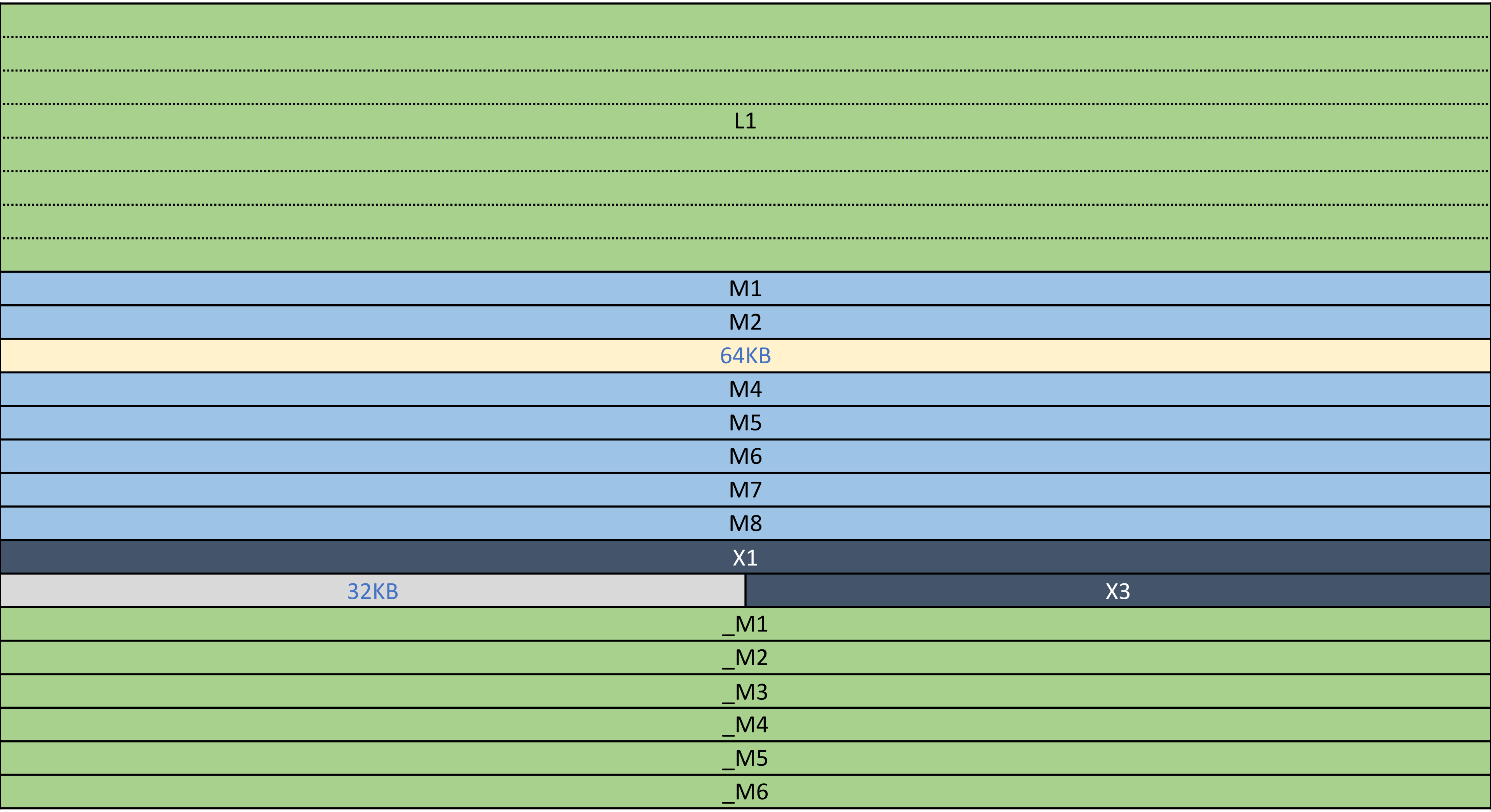
Release vulnerable *Medium-sized* chunk + Release all Large chunks



Phys Feng Shui step 5/8

Release vulnerable *Medium-sized* chunk + Release all Large chunks

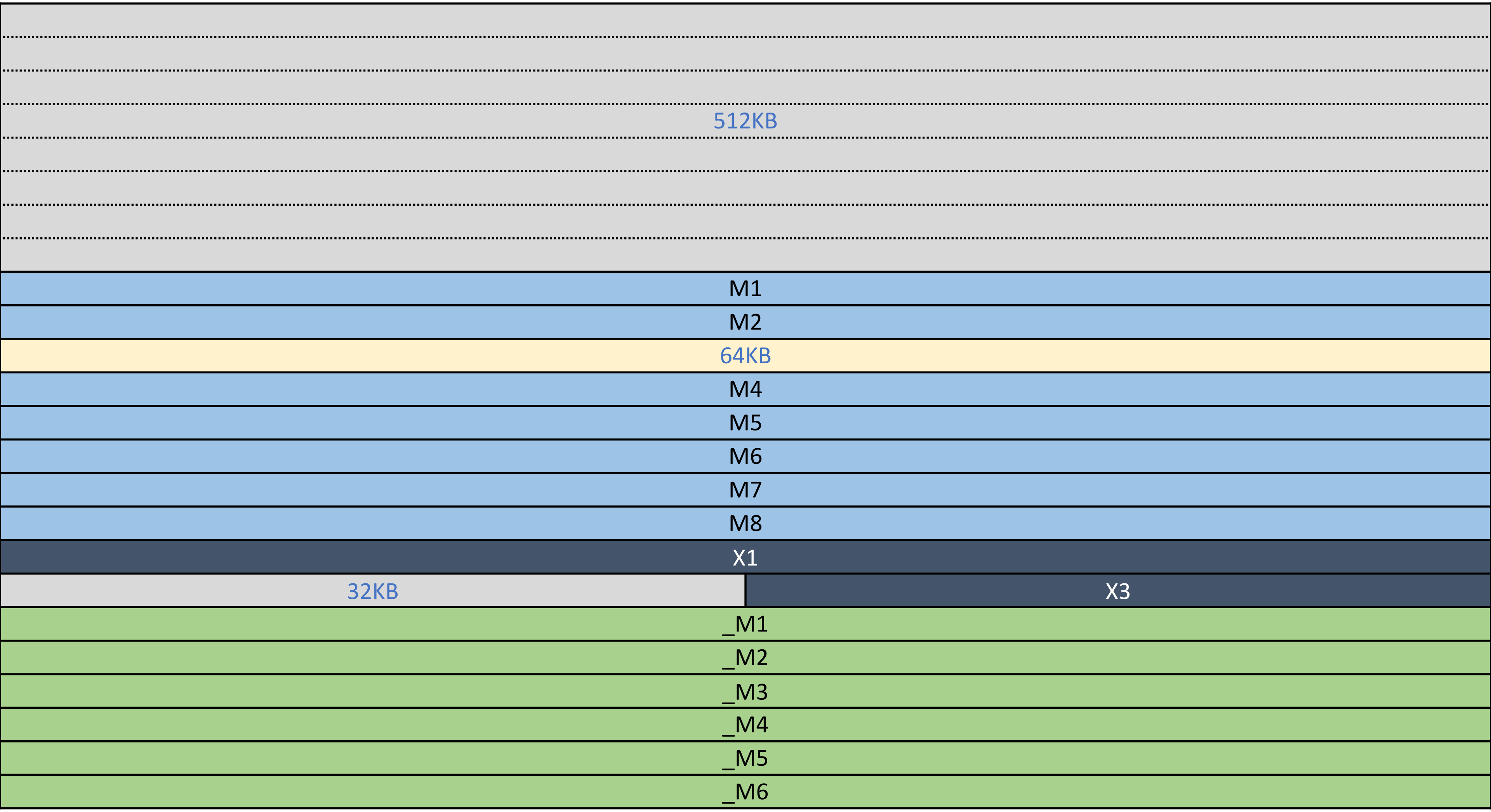
```
Release(M3); // releases the vulnerable row
```



Phys Feng Shui step 5/8

Release vulnerable *Medium-sized* chunk + Release all Large chunks

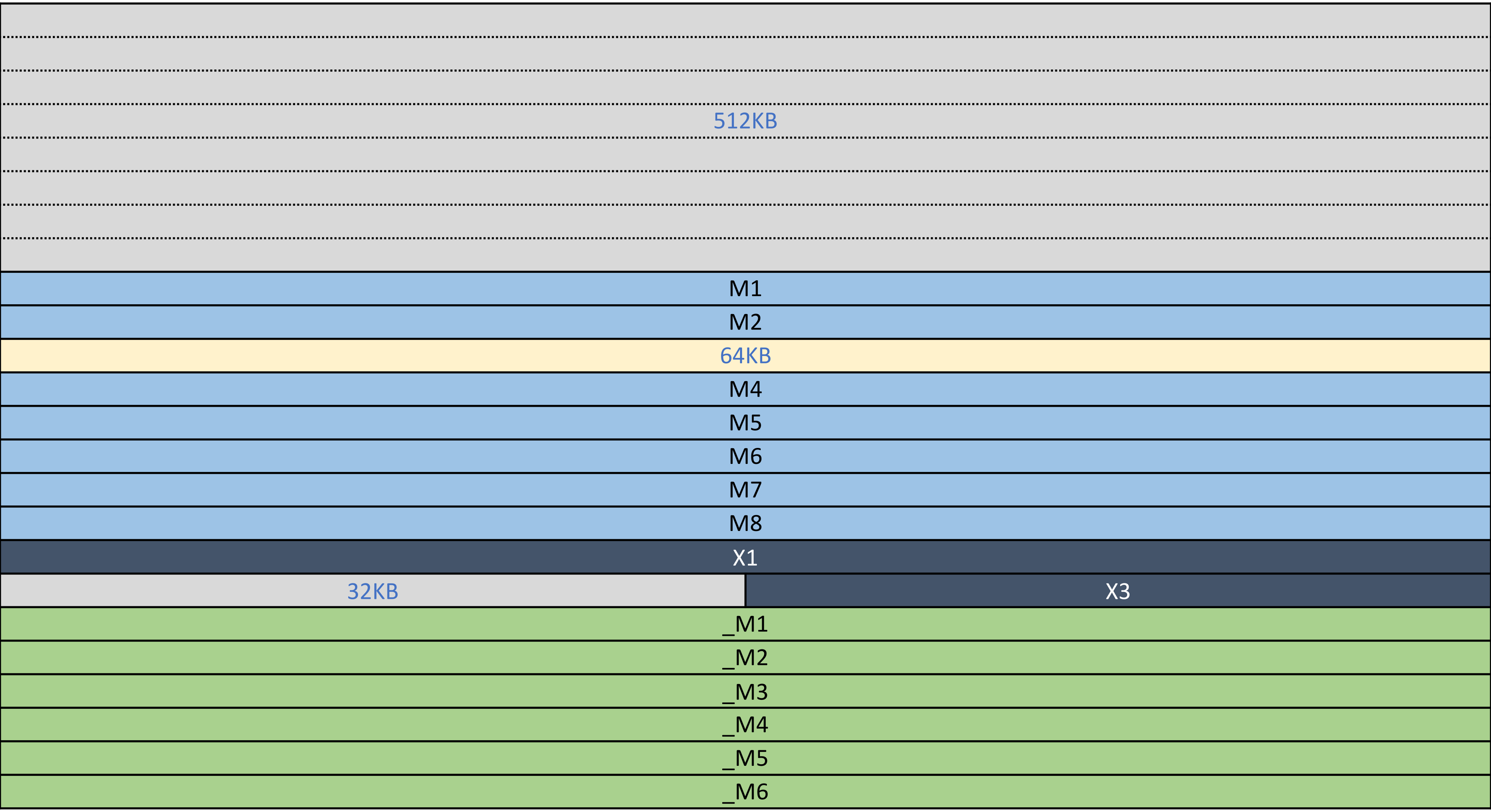
```
ReleaseAll(L); // to avoid going out-of-memory later
```



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

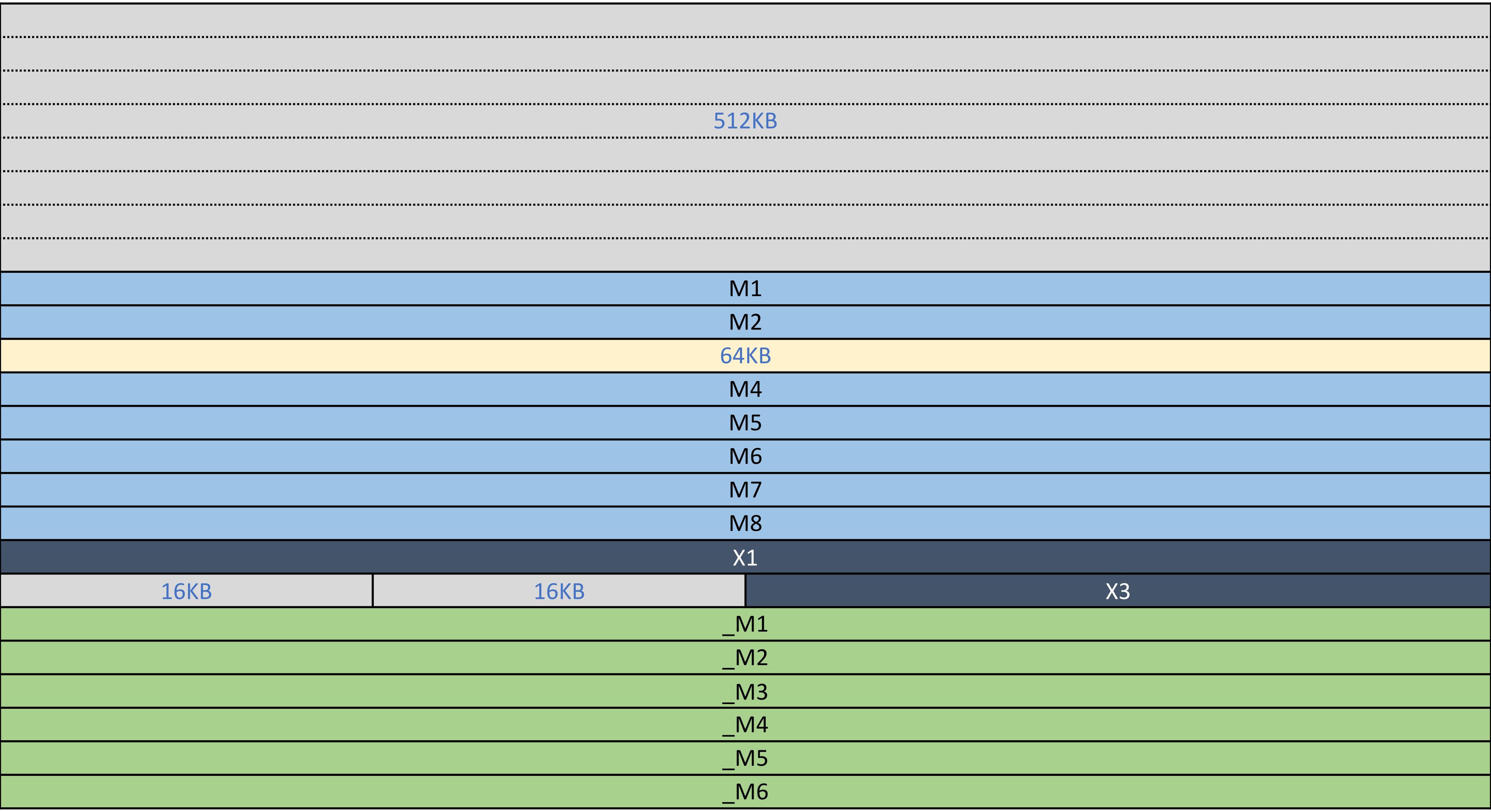
`Land(S); // allocate 4KB pages until the 64KB is used`



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

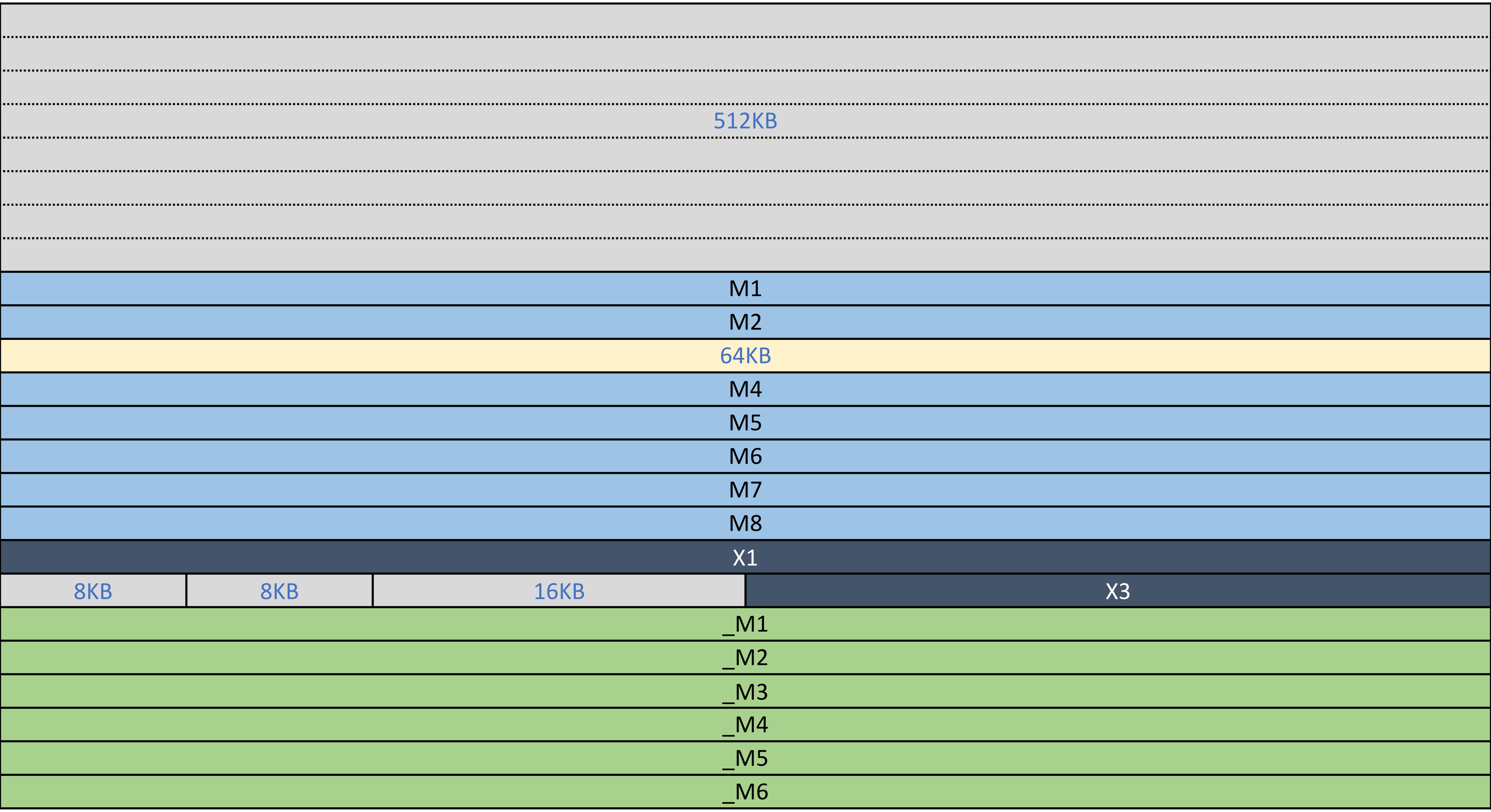
Land(S); // allocate 4KB pages until the 64KB is used



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

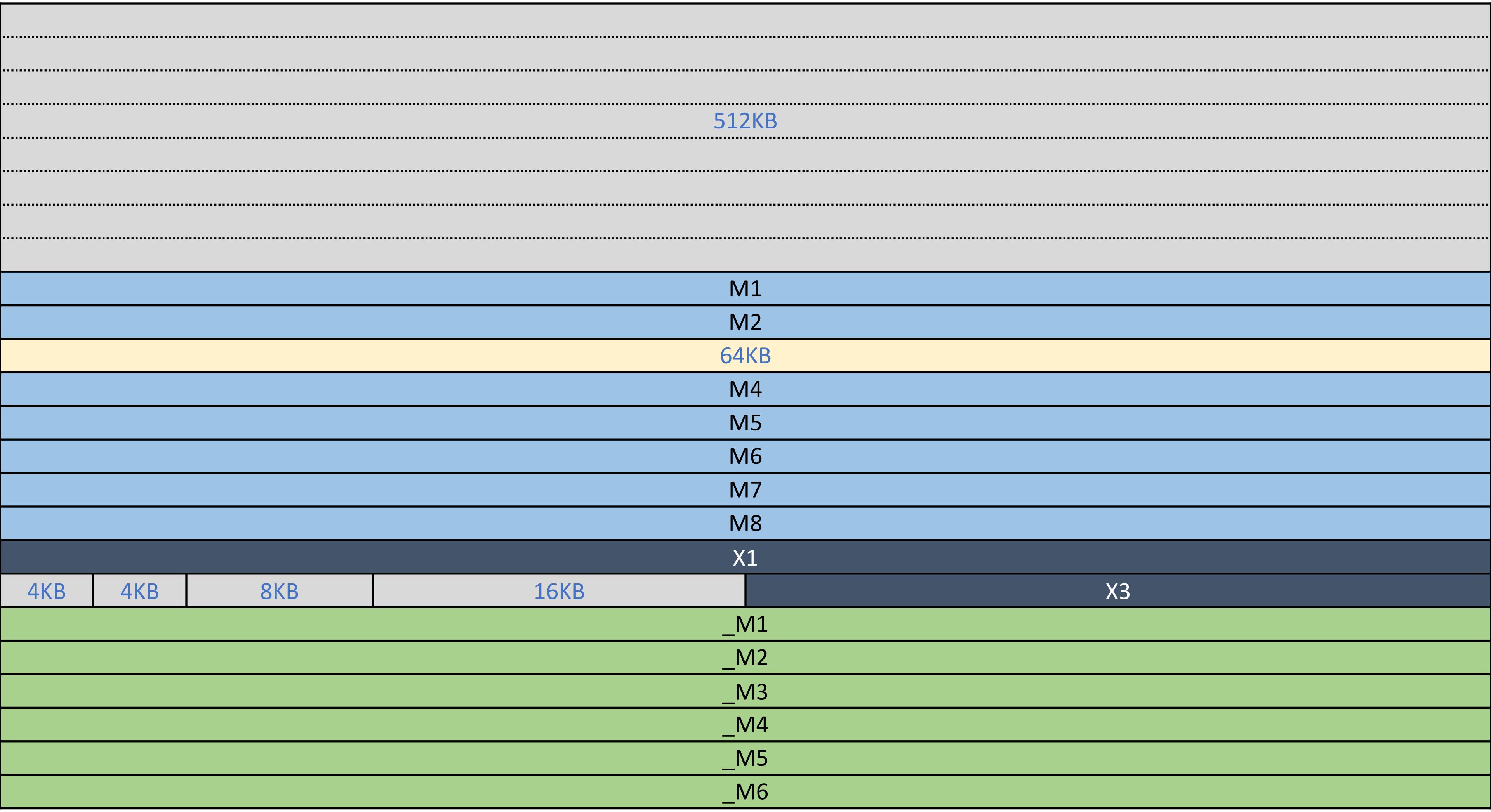
`Land(S); // allocate 4KB pages until the 64KB is used`



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

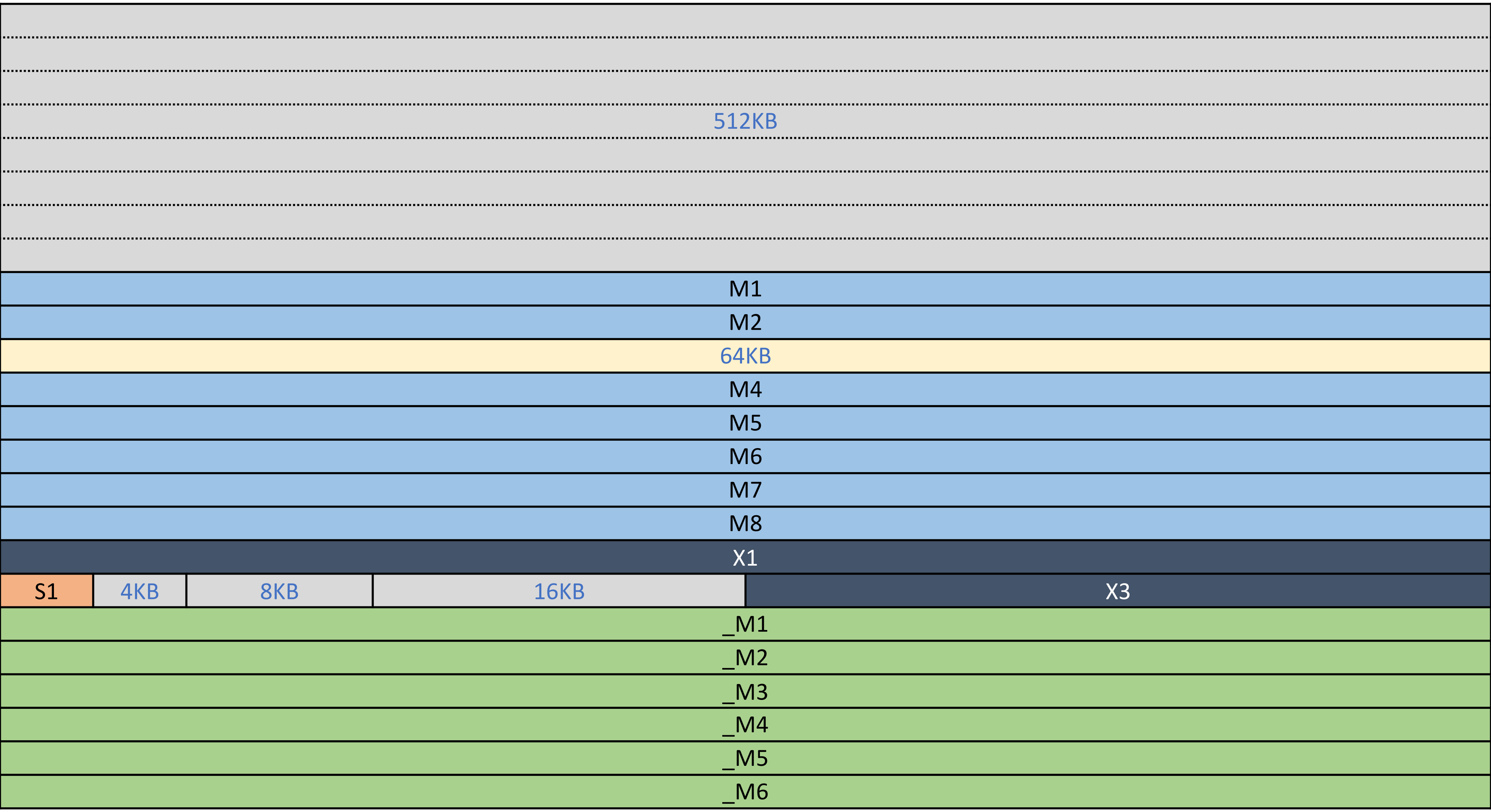
`Land(S); // allocate 4KB pages until the 64KB is used`



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

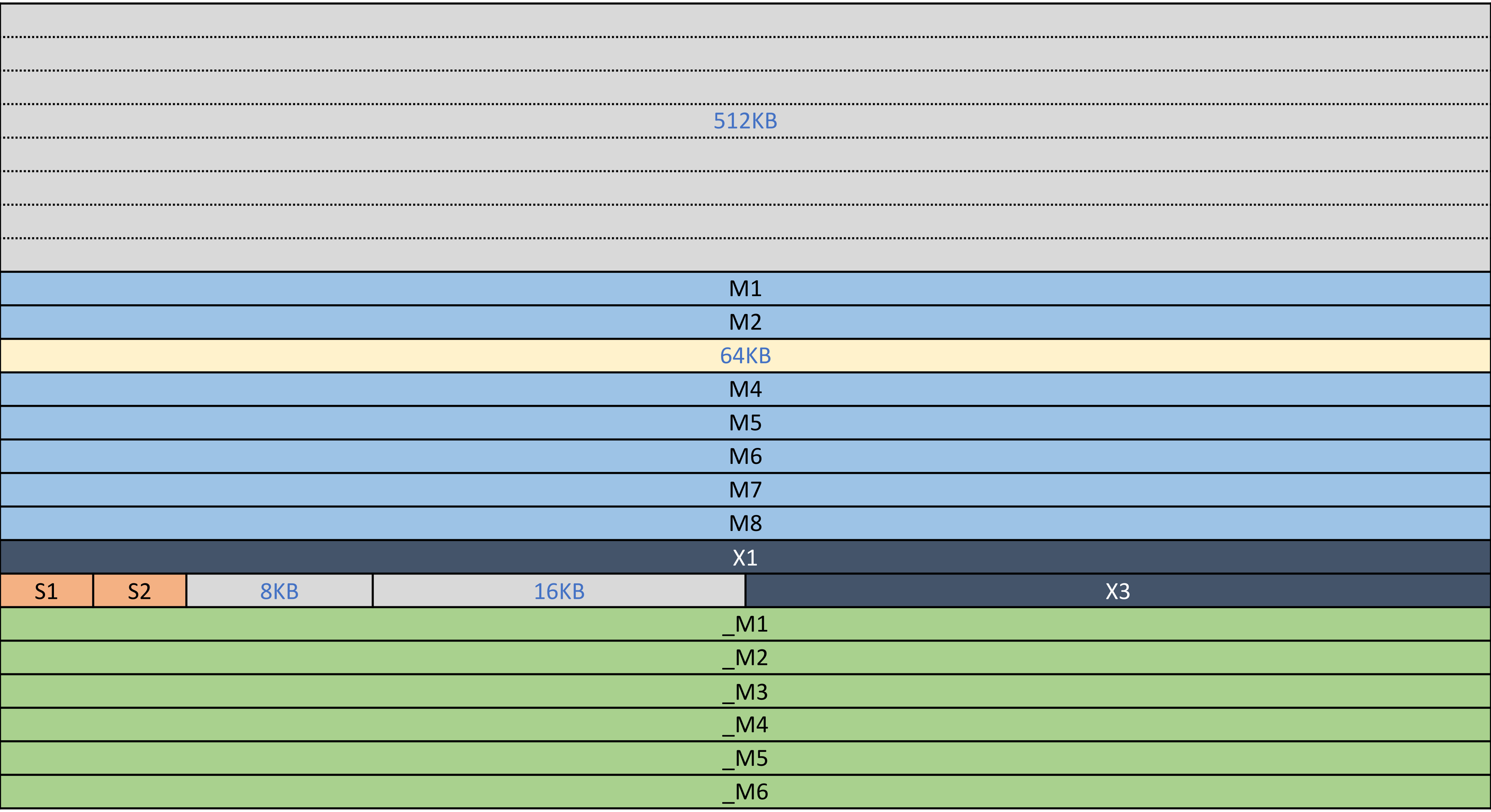
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

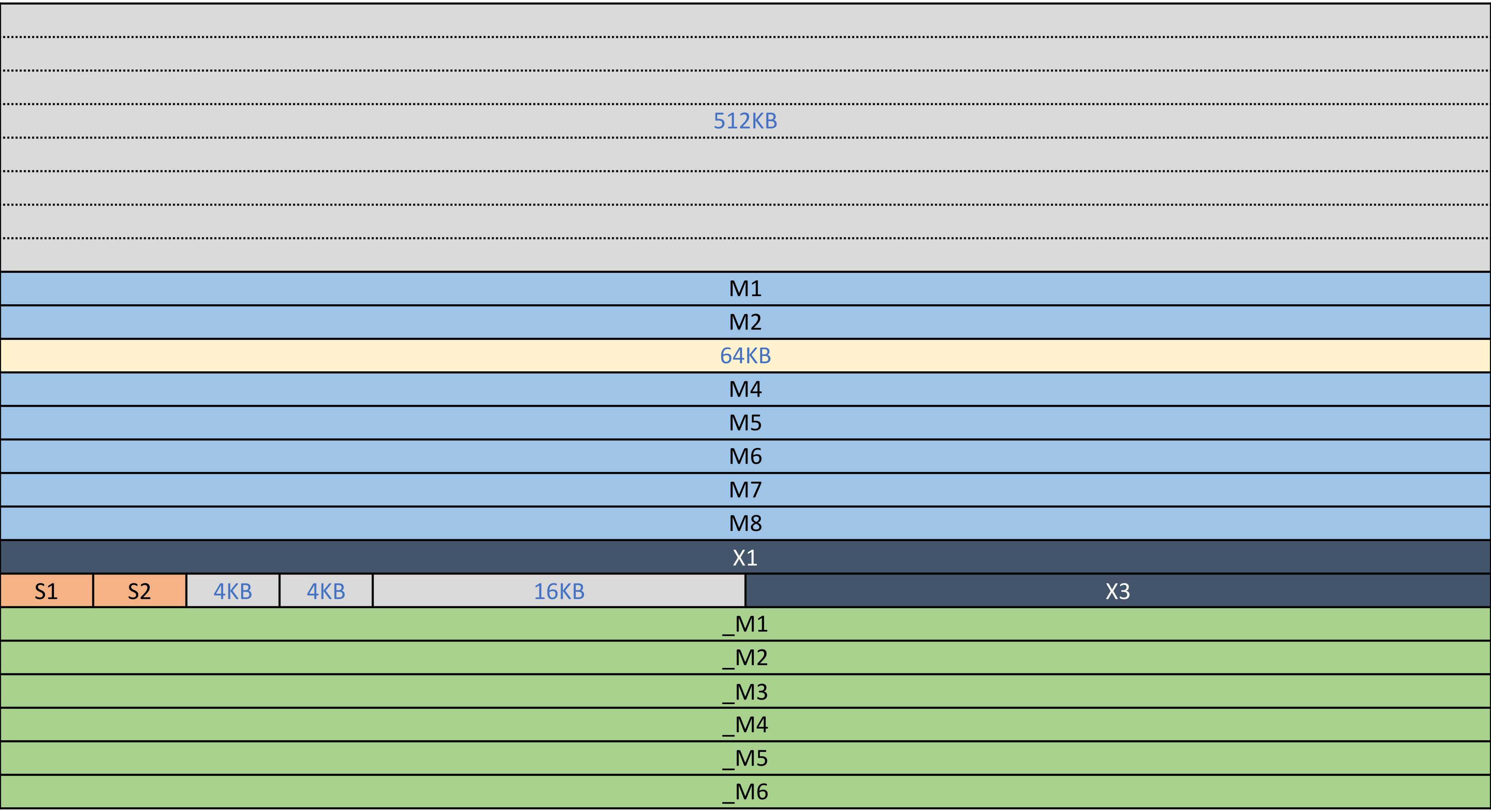
`Land(S); // allocate 4KB pages until the 64KB is used`



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

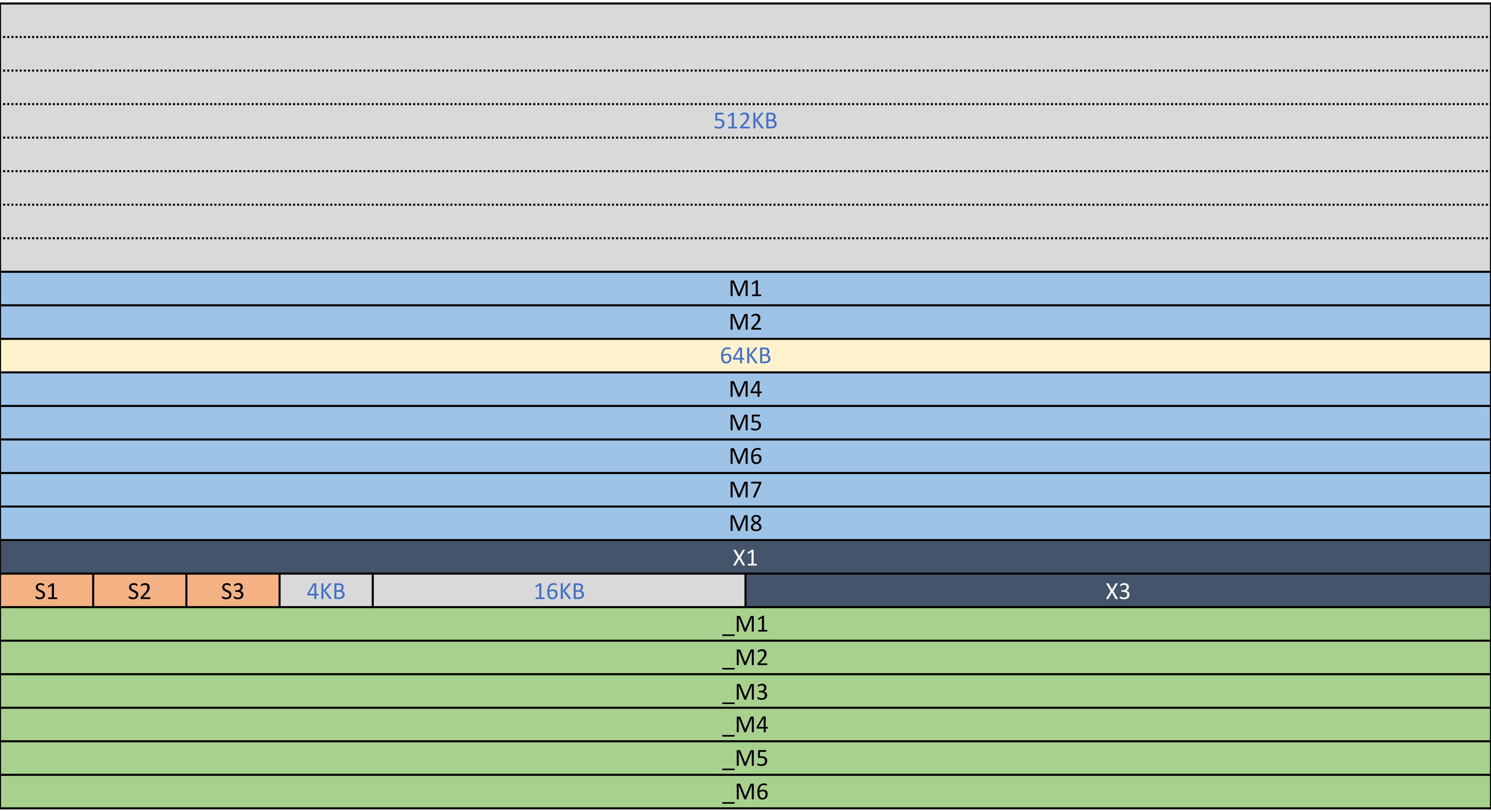
`Land(S); // allocate 4KB pages until the 64KB is used`



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

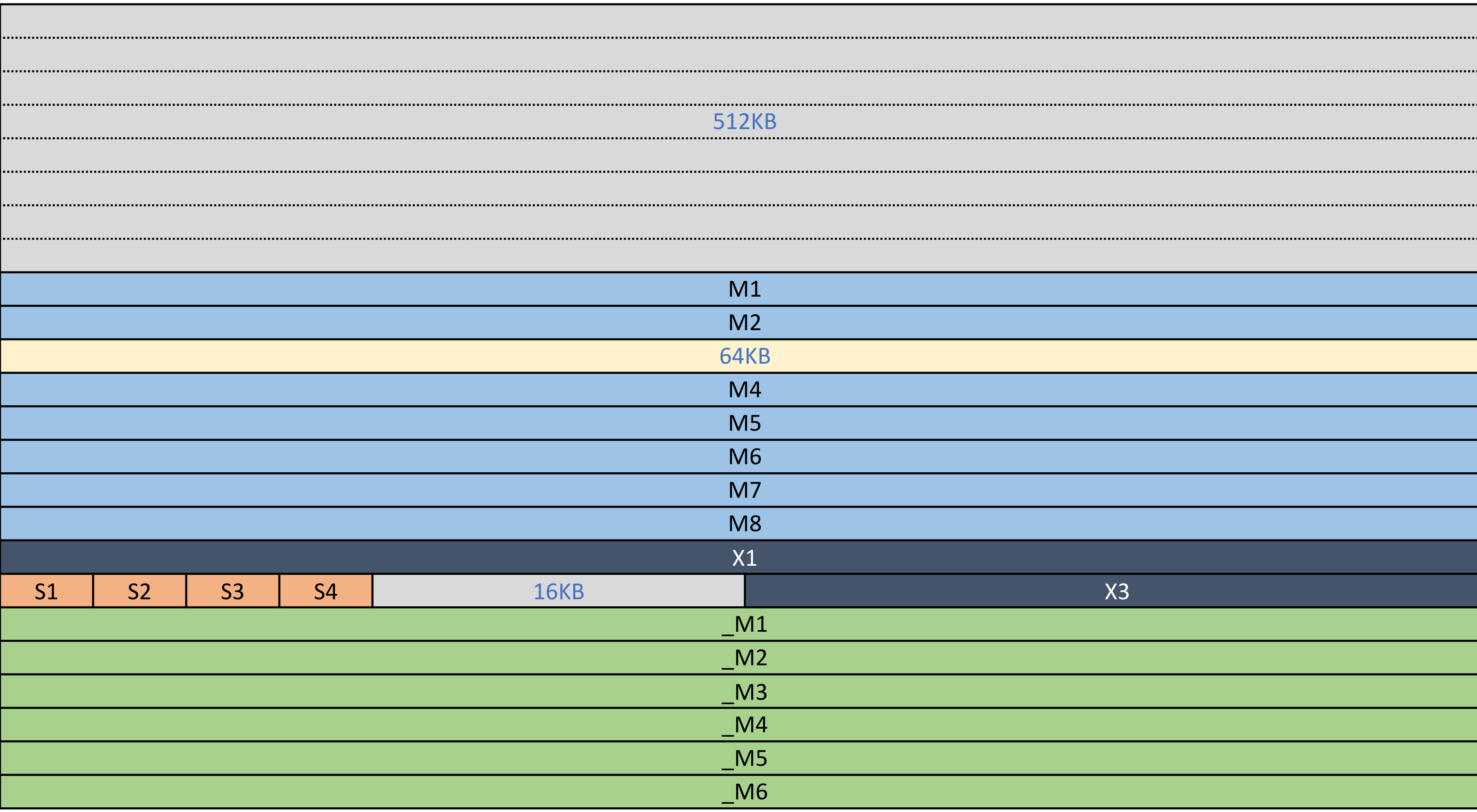
`Land(S); // allocate 4KB pages until the 64KB is used`



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

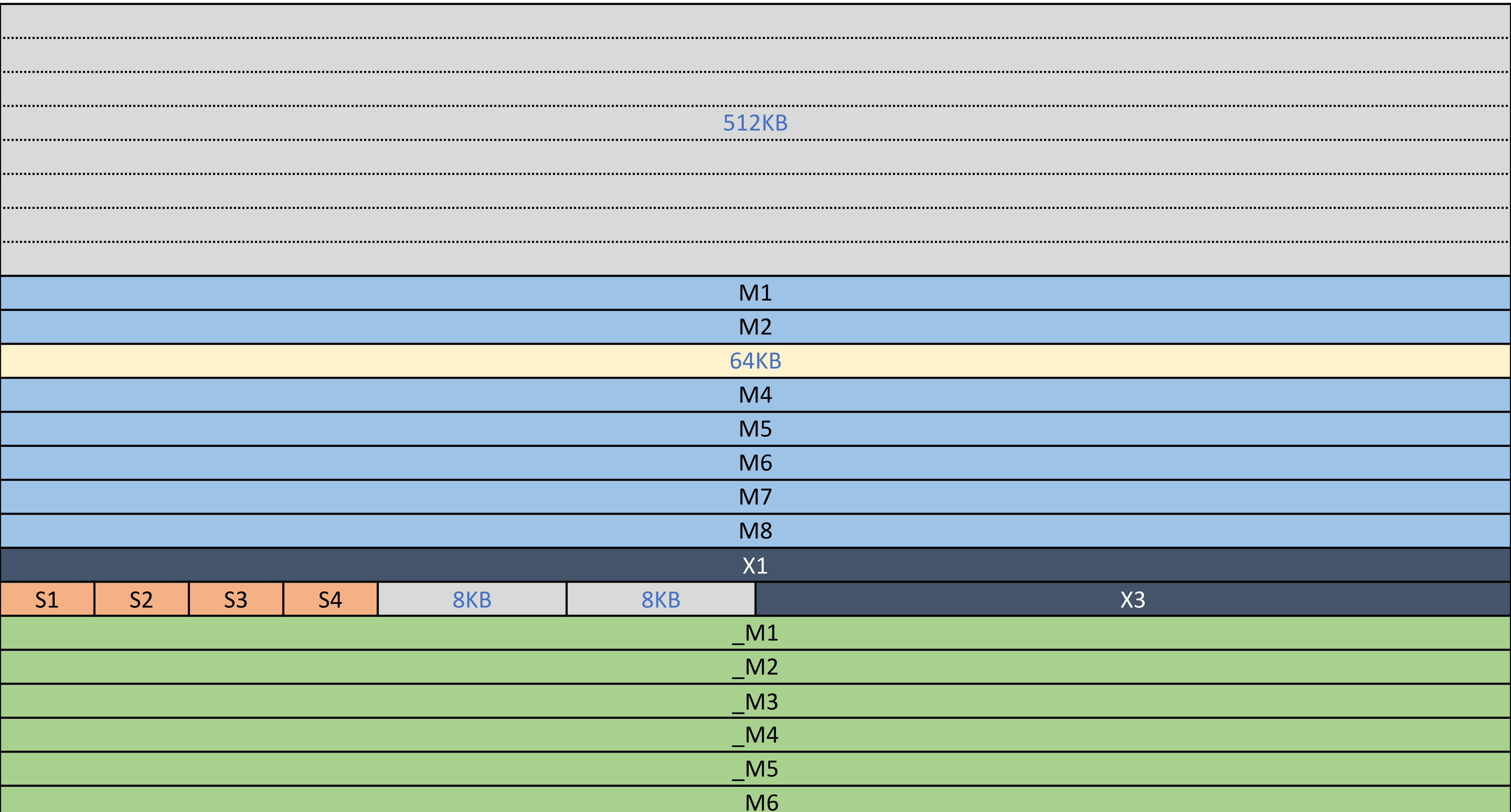
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

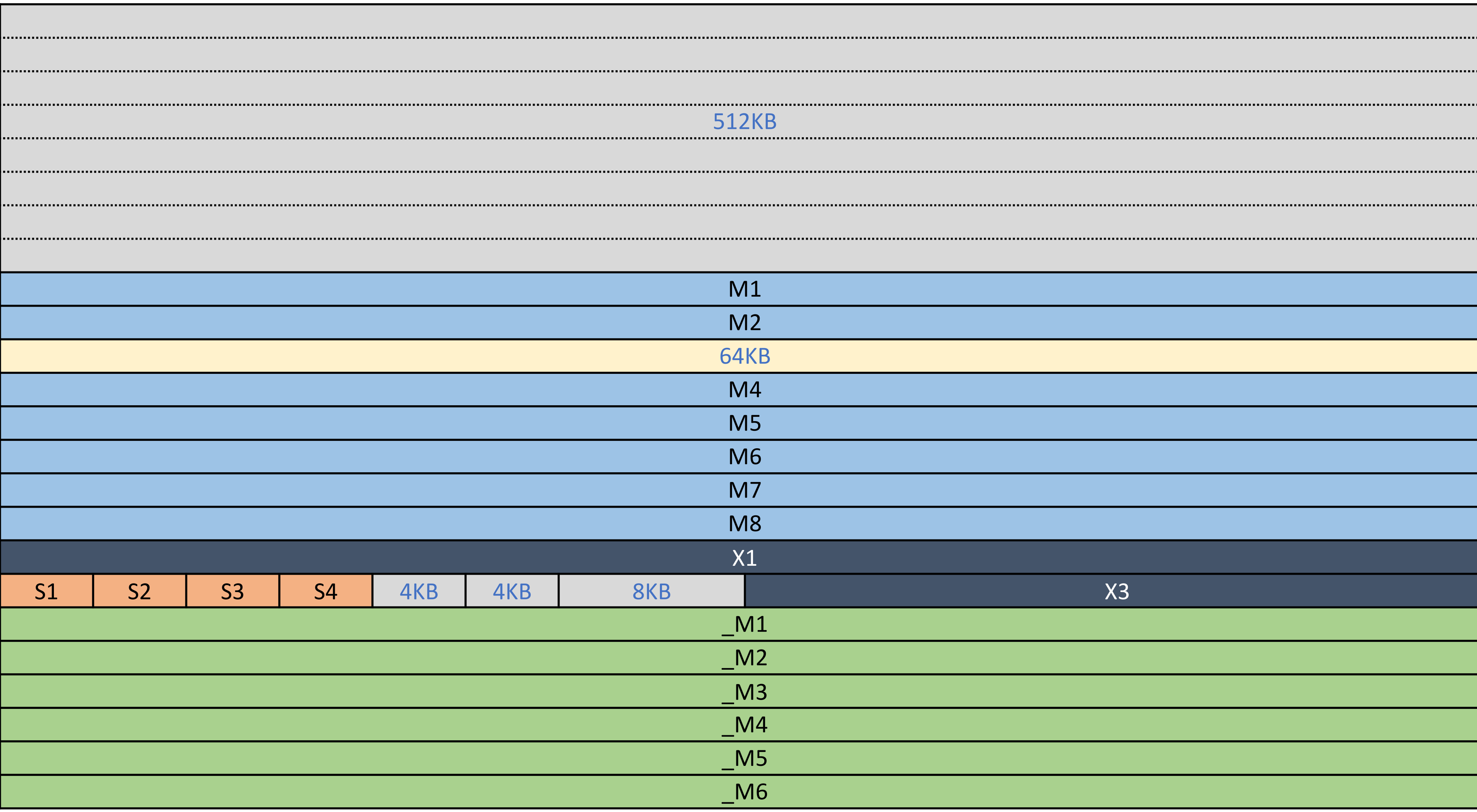
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

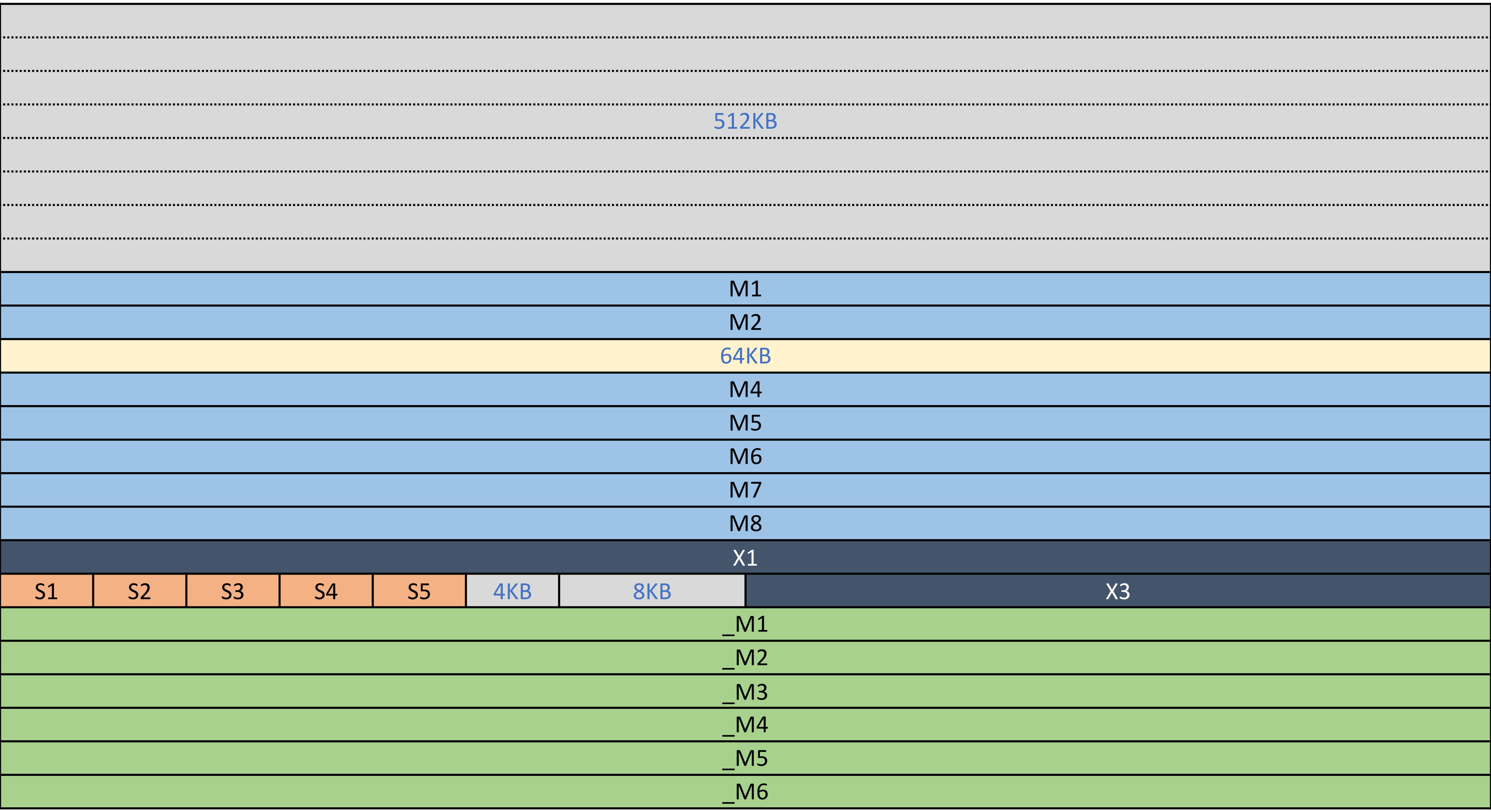
`Land(S);` `// allocate 4KB pages until the 64KB is used`



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

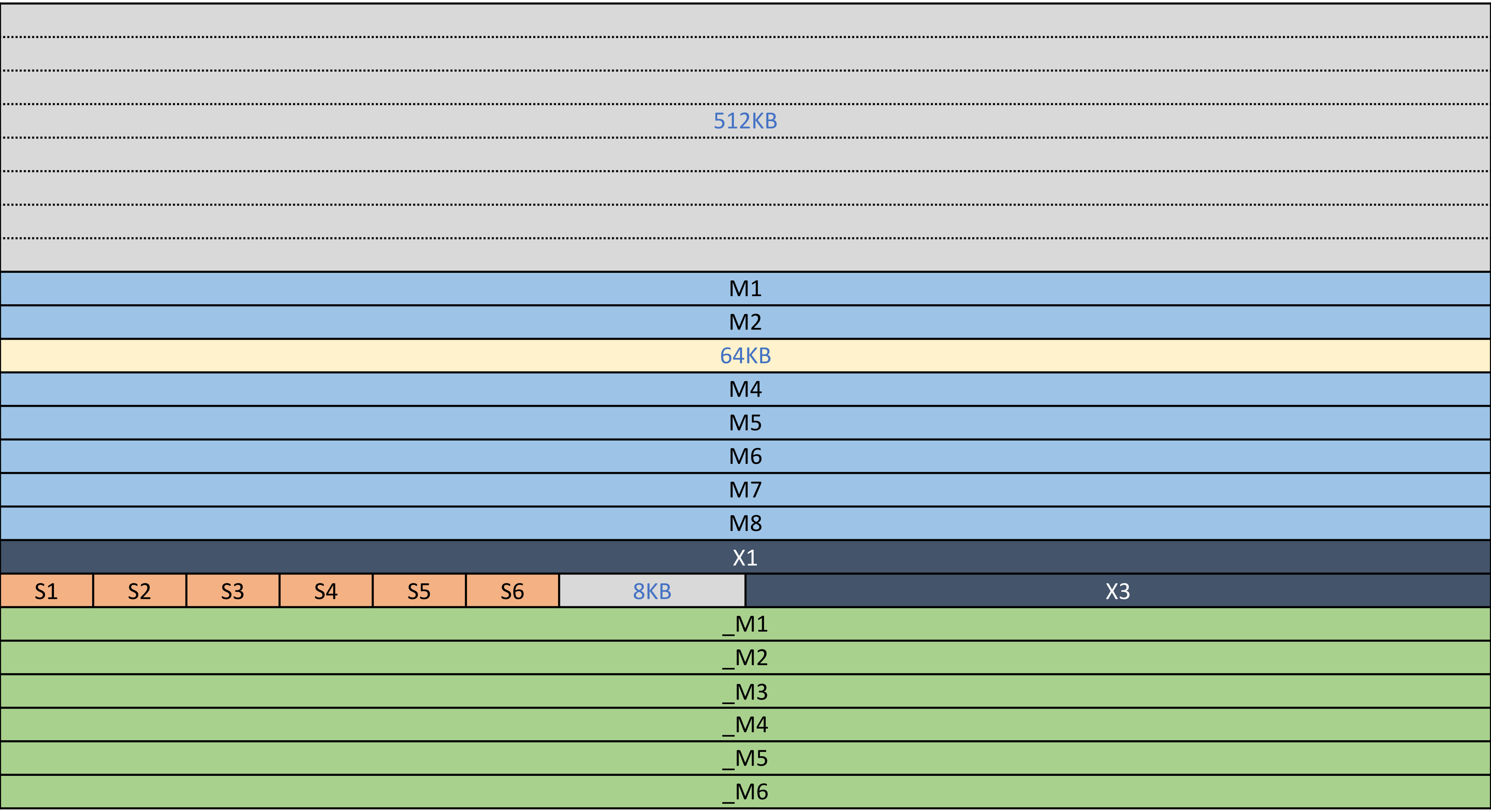
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

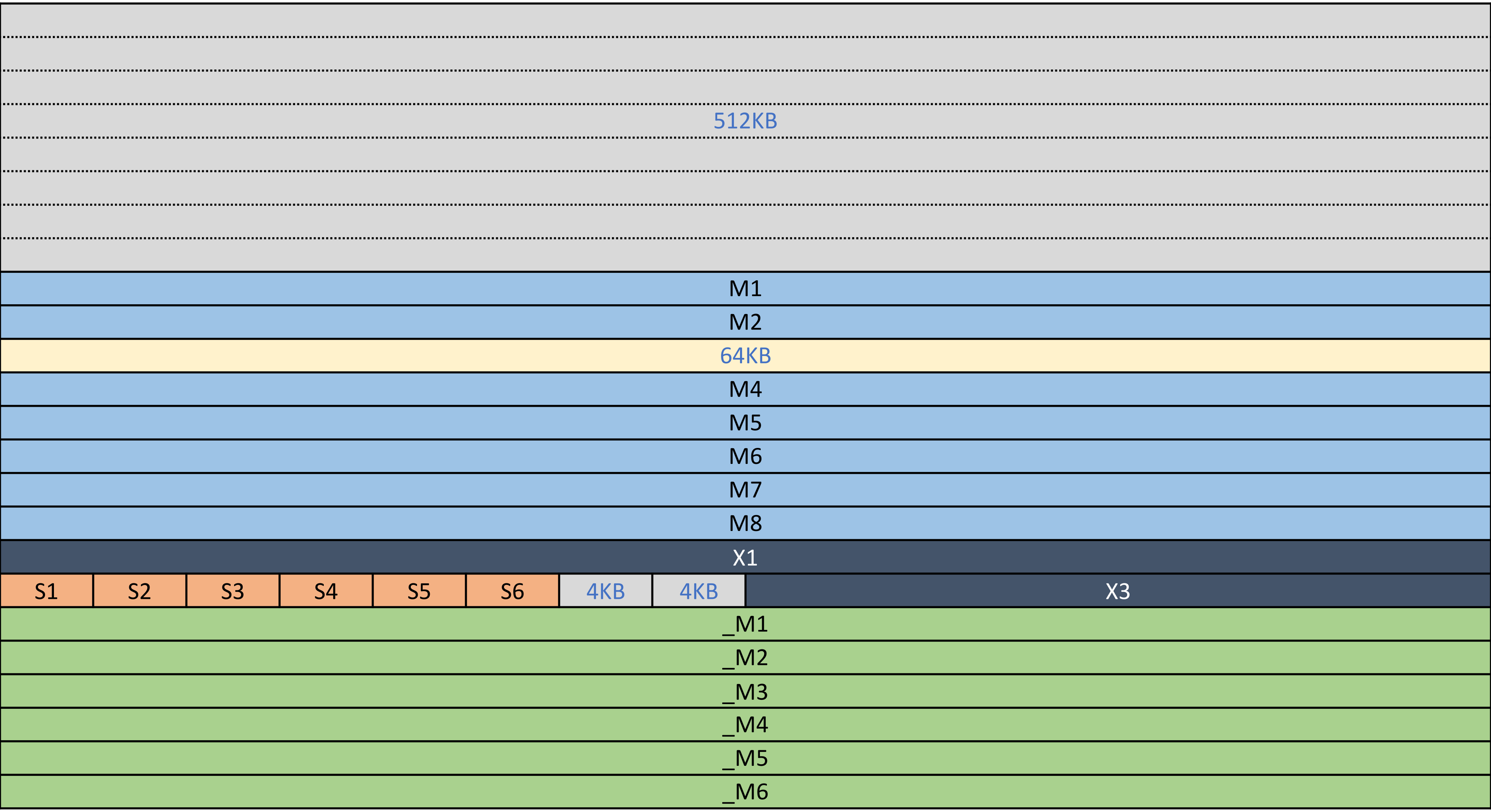
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

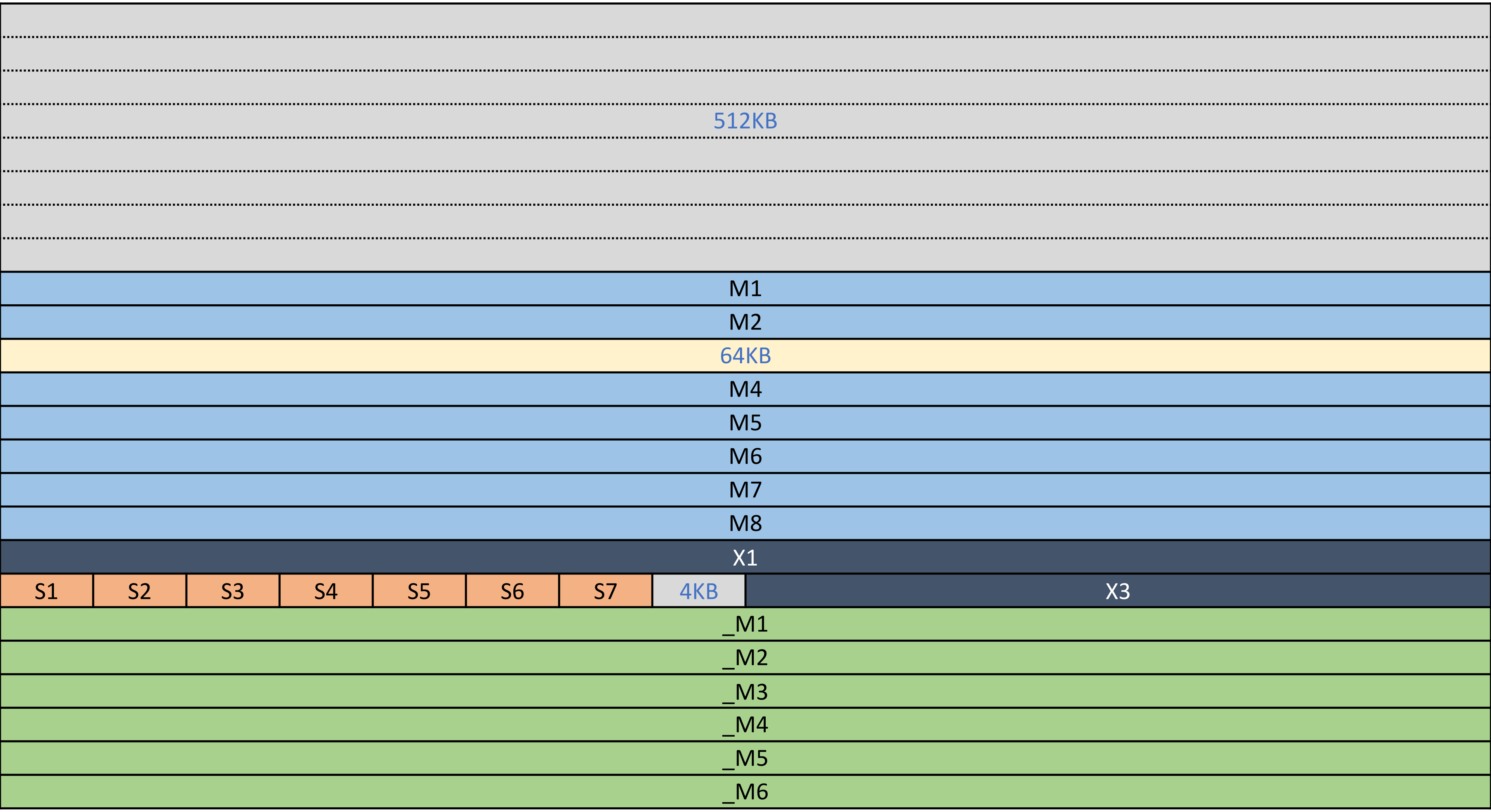
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

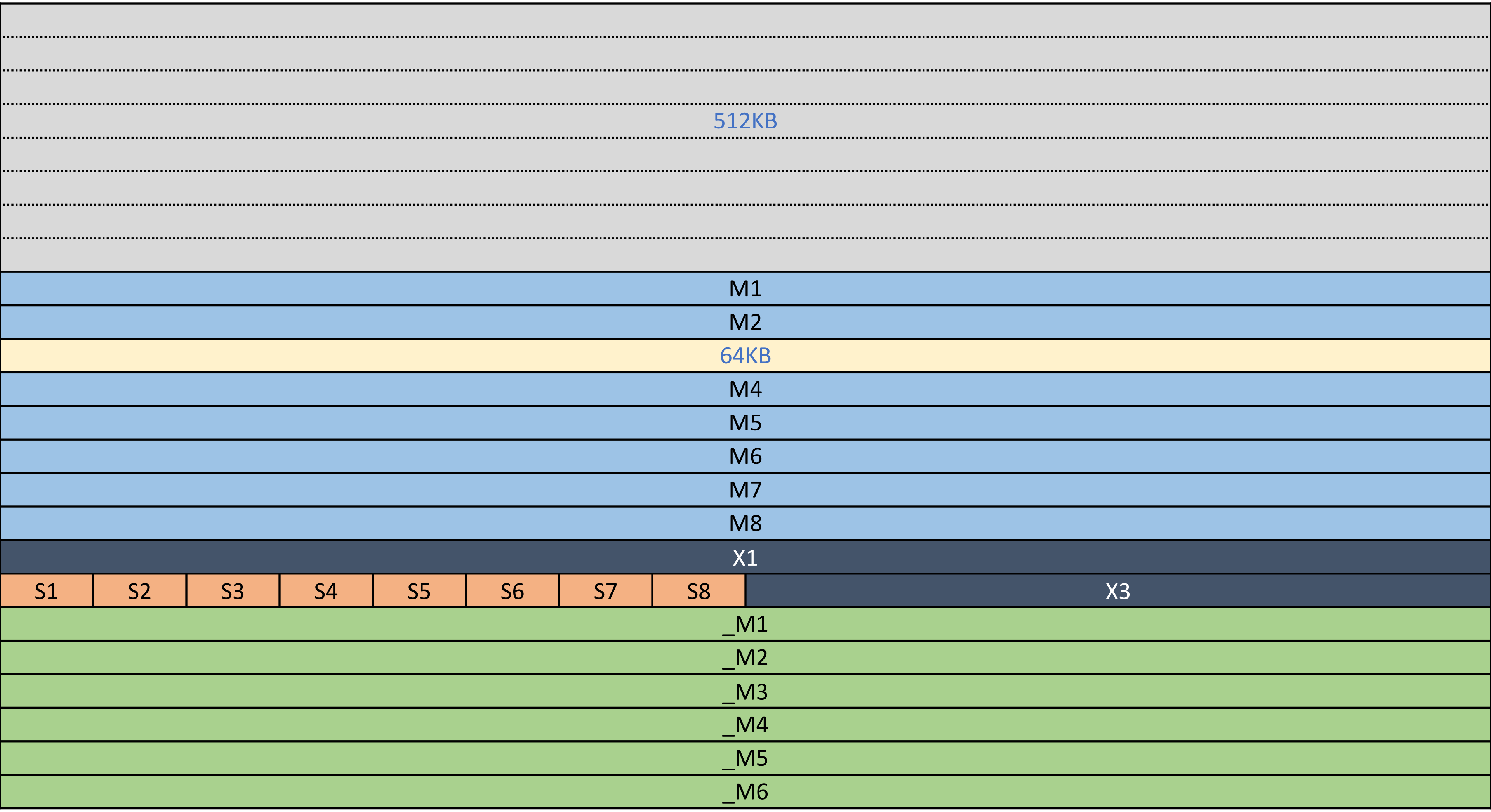
`Land(S); // allocate 4KB pages until the 64KB is used`



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

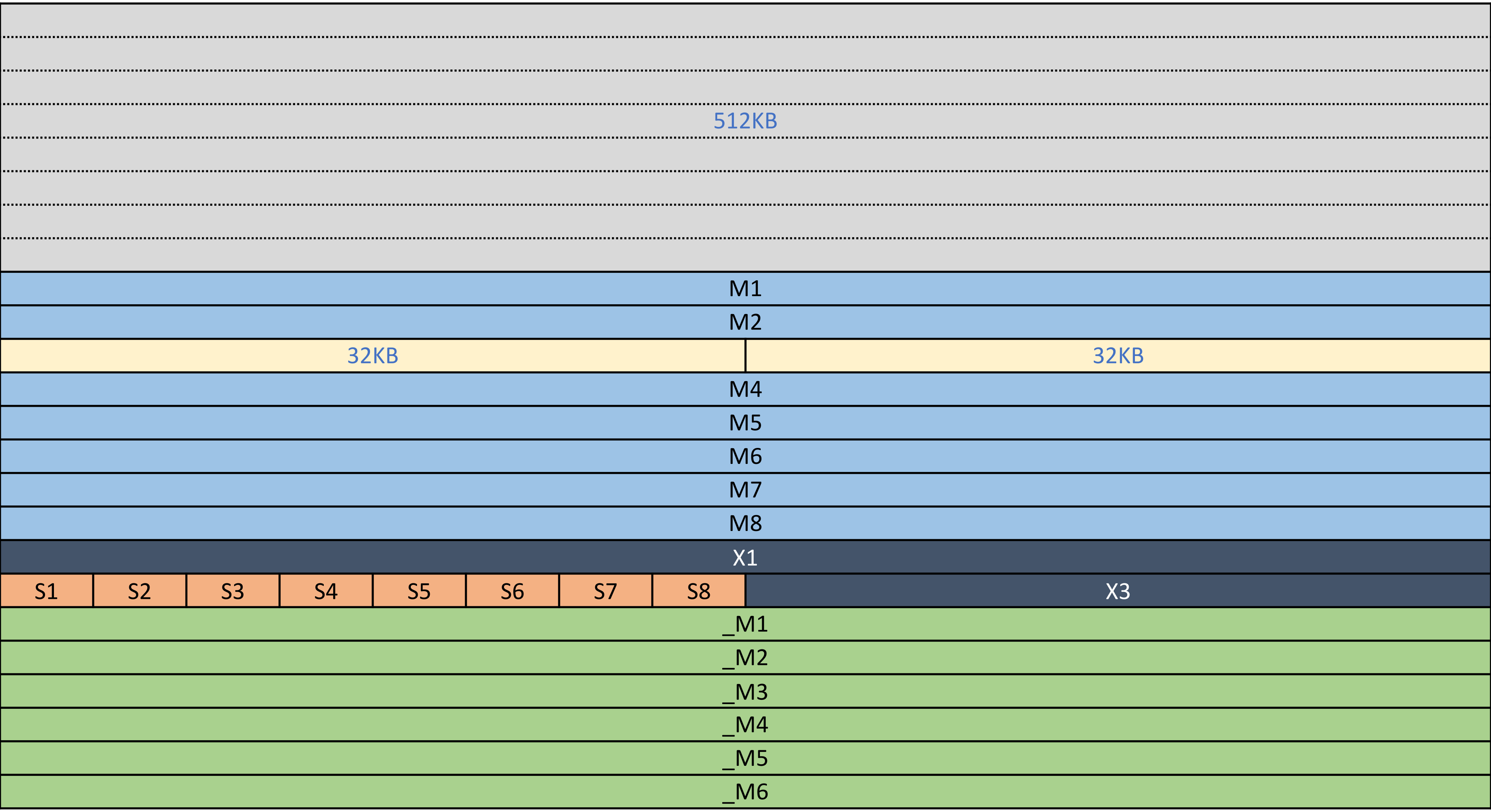
`Land(S); // allocate 4KB pages until the 64KB is used`



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

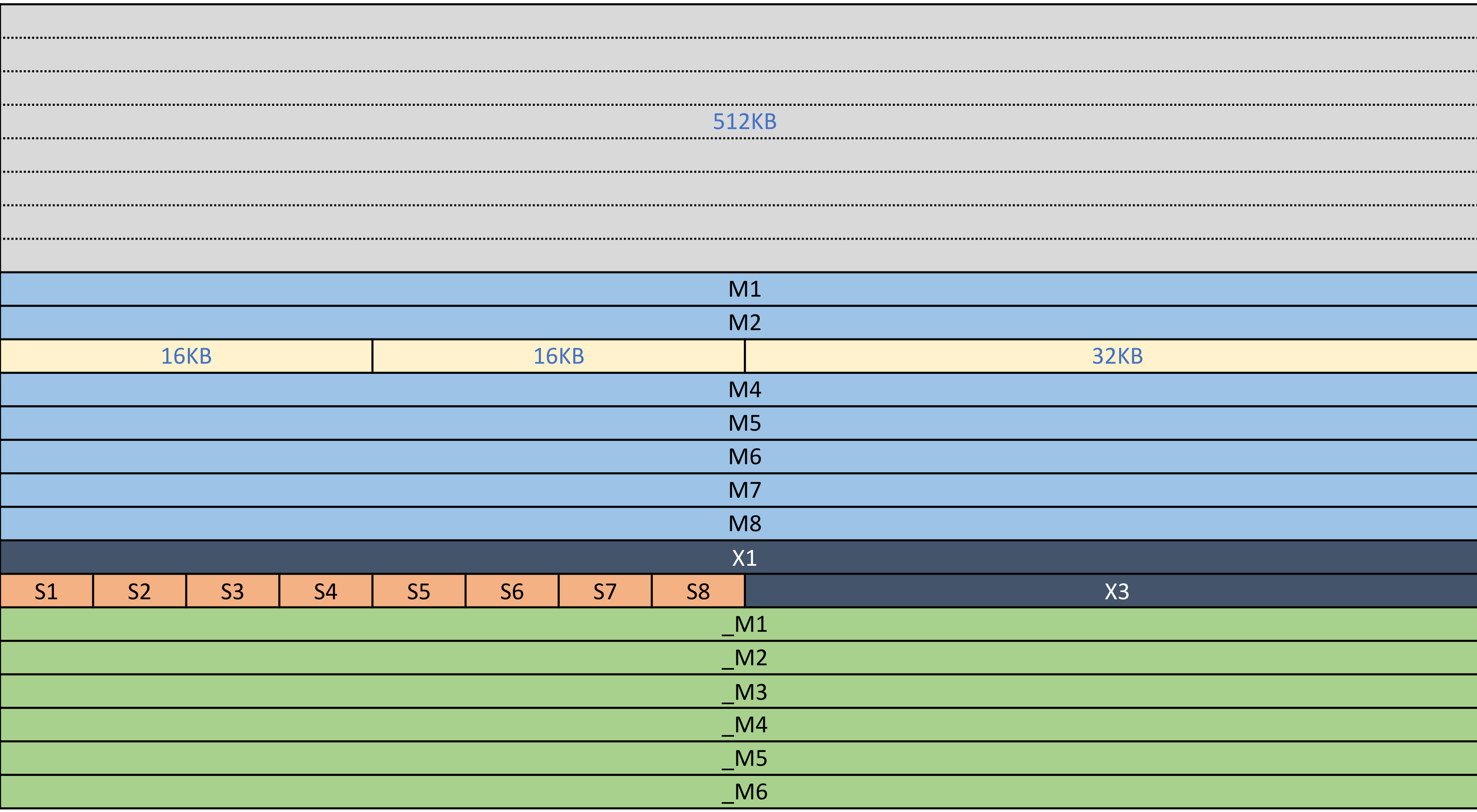
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

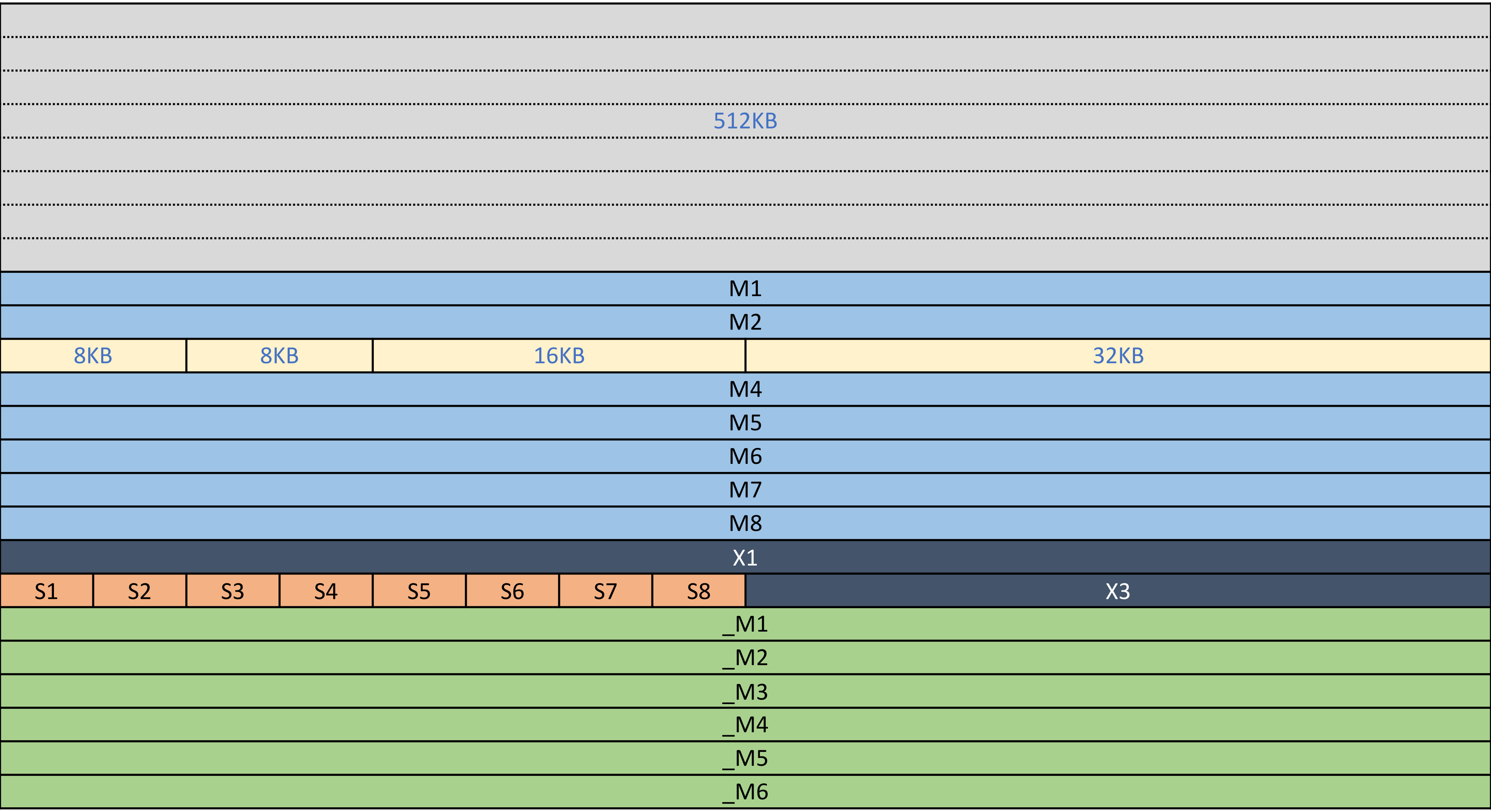
`Land(S); // allocate 4KB pages until the 64KB is used`



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

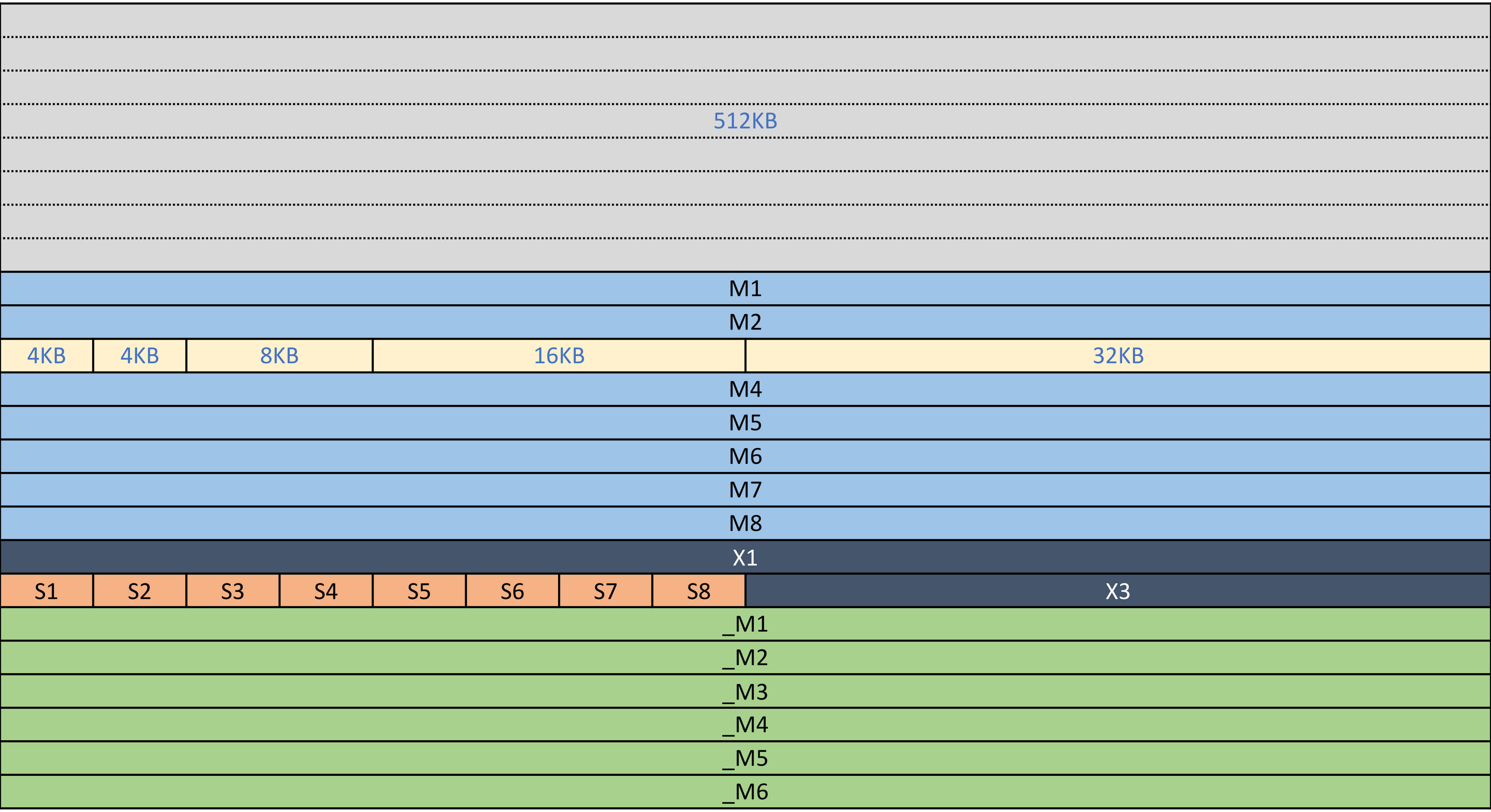
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

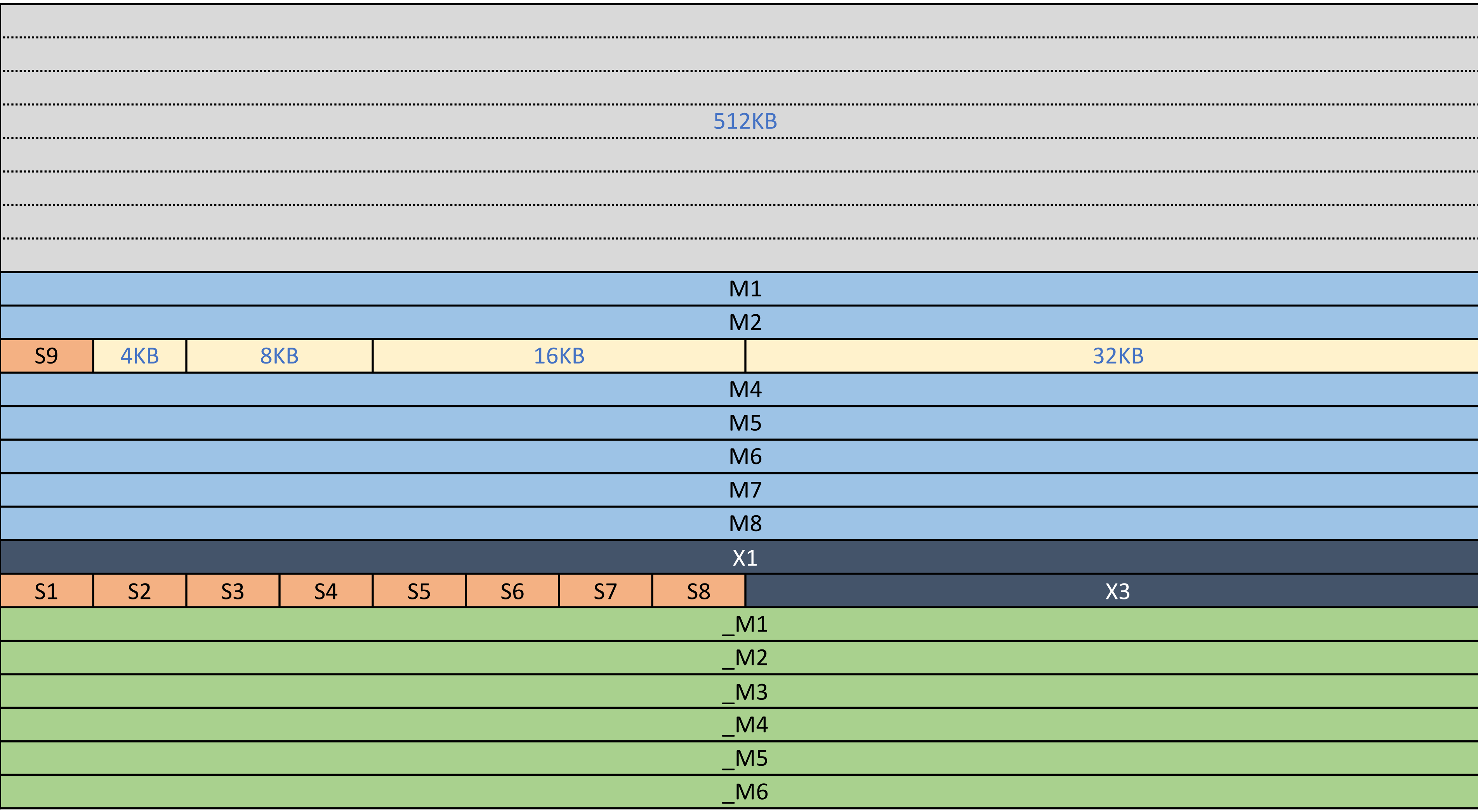
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

Land a *small* chunk in the vulnerable 64 KB row

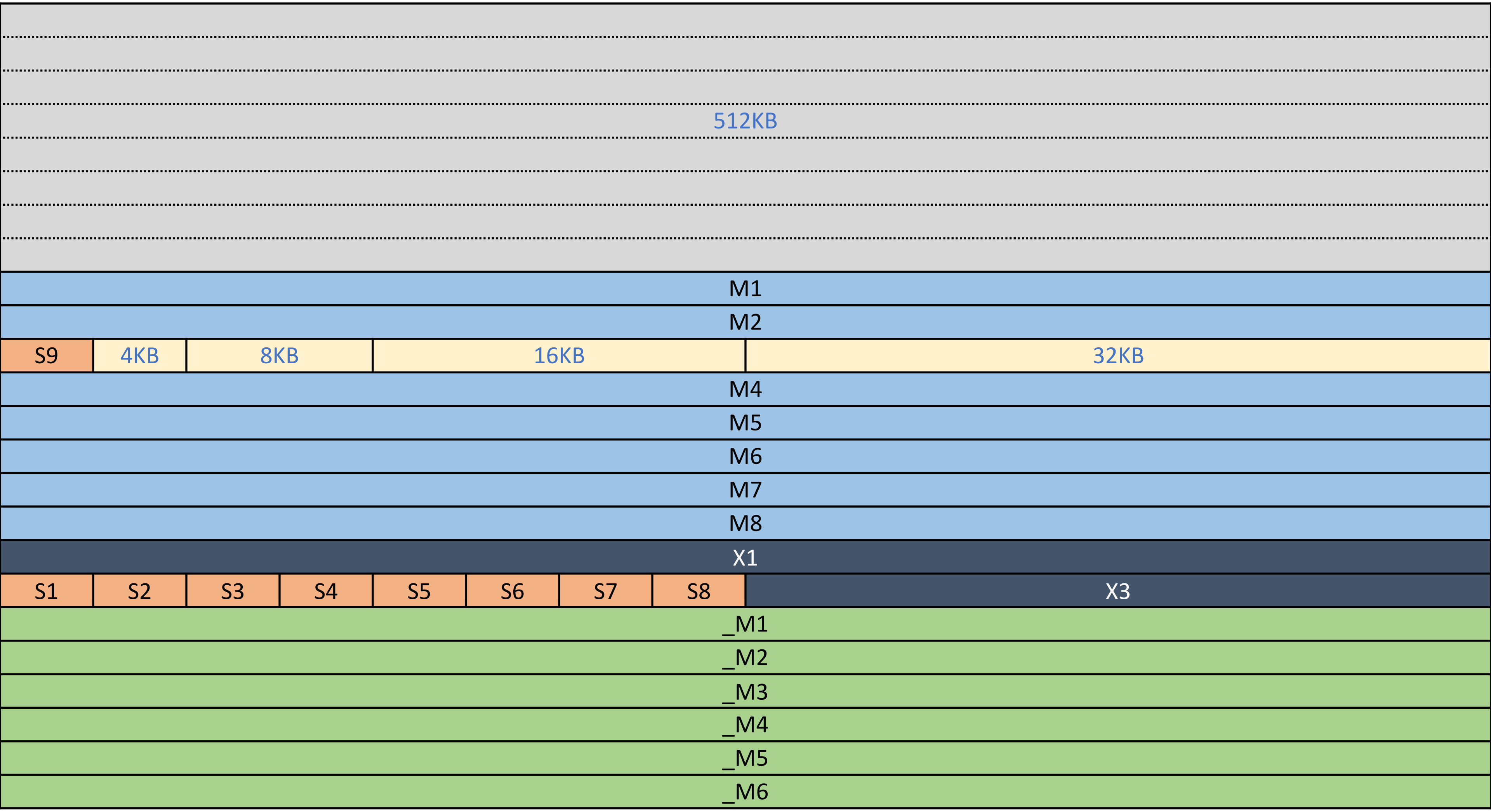
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 7/8

Pad *small* chunks until the vulnerable page

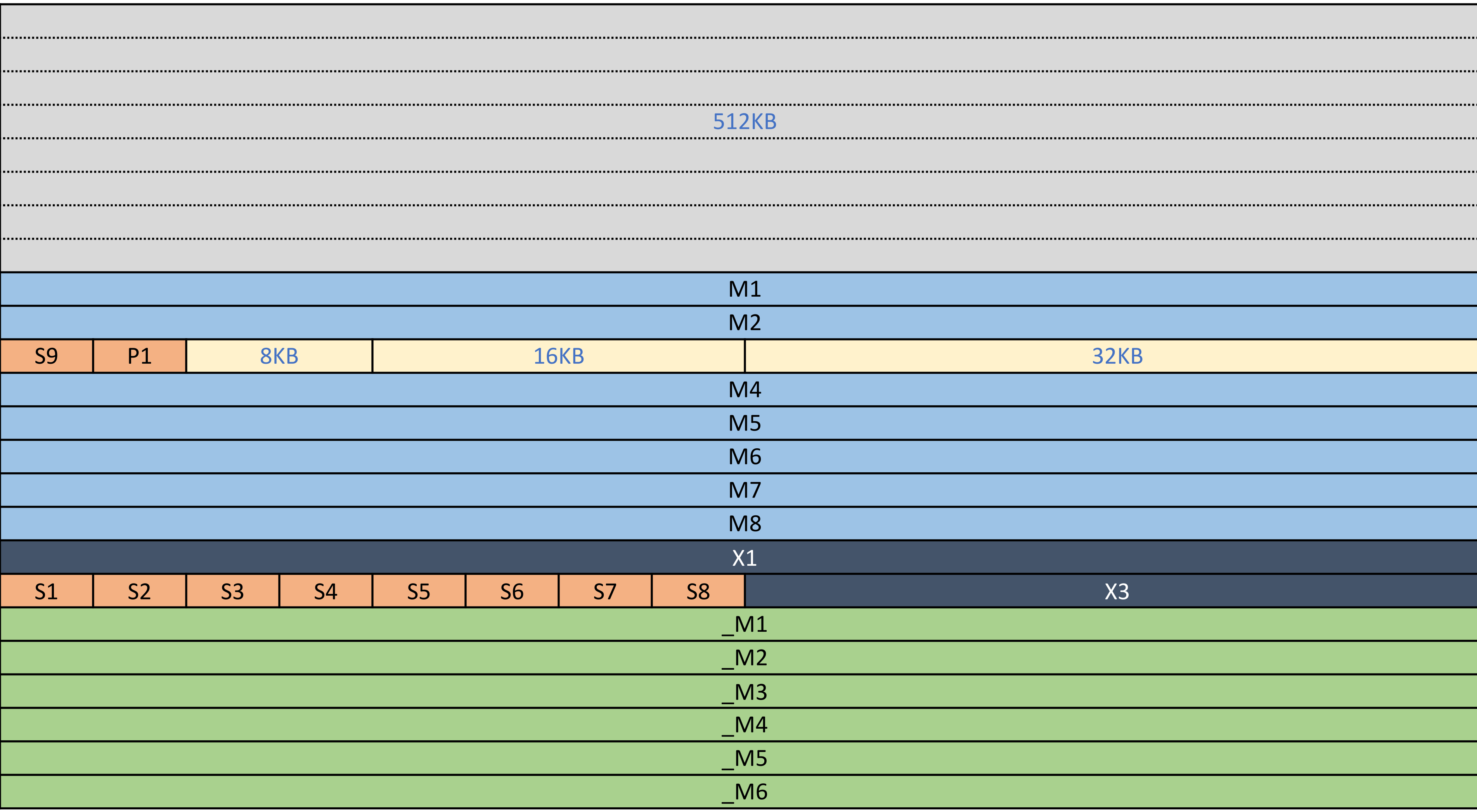
Pad(P); // insert padding until vulnerable page



Phys Feng Shui step 7/8

Pad *small* chunks until the vulnerable page

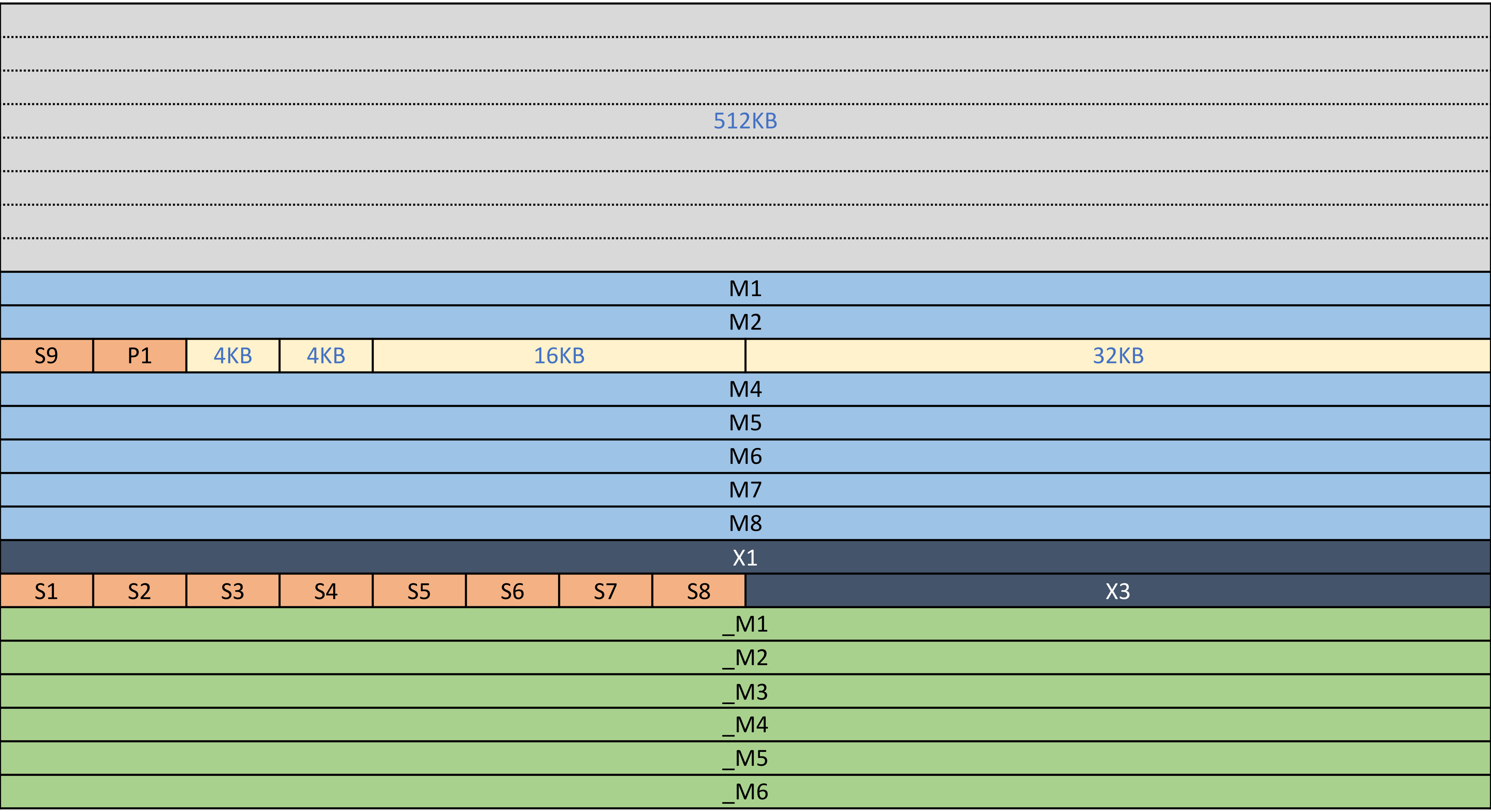
```
Pad(P); // insert padding until vulnerable page
```



Phys Feng Shui step 7/8

Pad *small* chunks until the vulnerable page

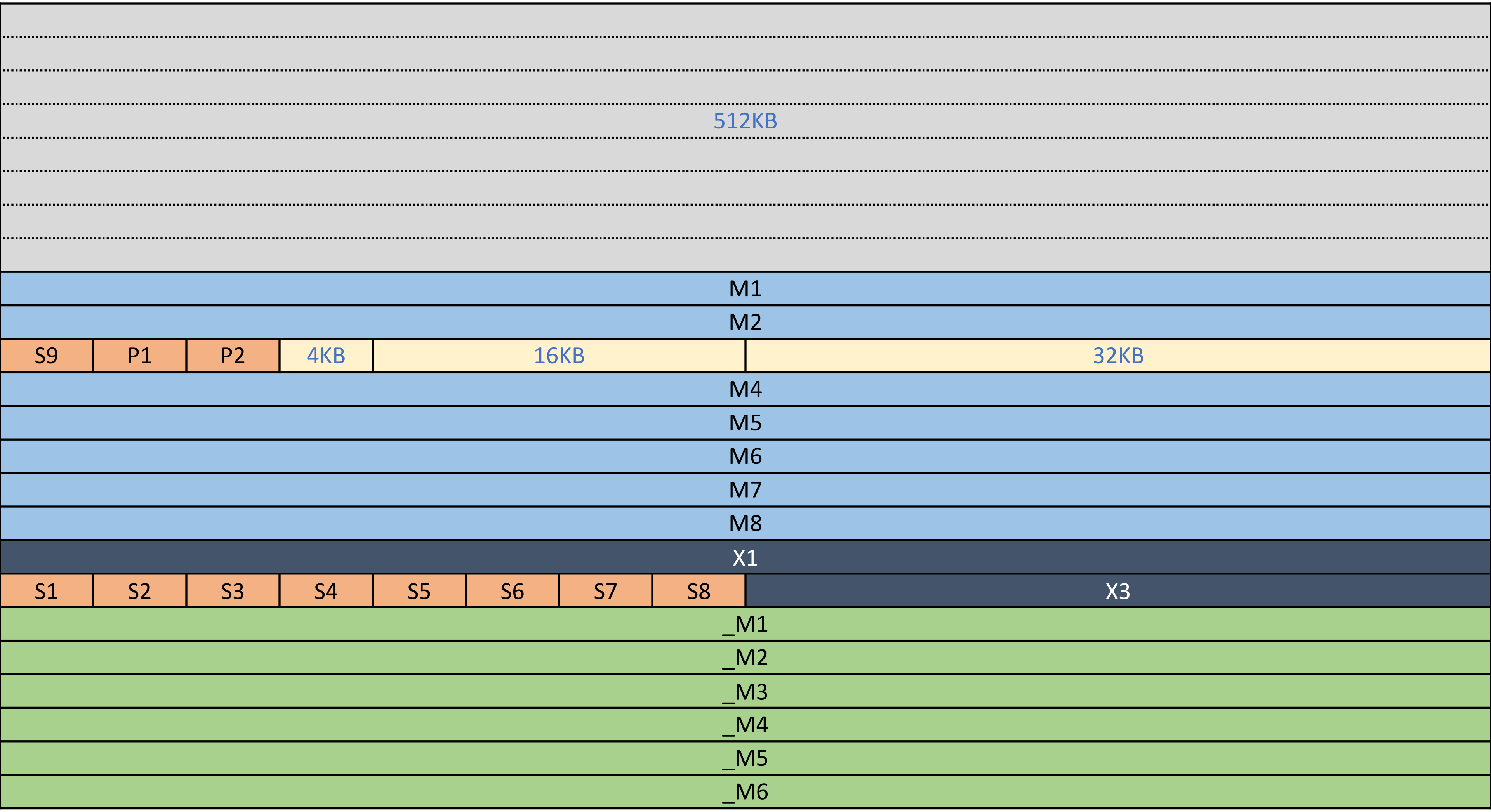
```
Pad(P); // insert padding until vulnerable page
```



Phys Feng Shui step 7/8

Pad *small* chunks until the vulnerable page

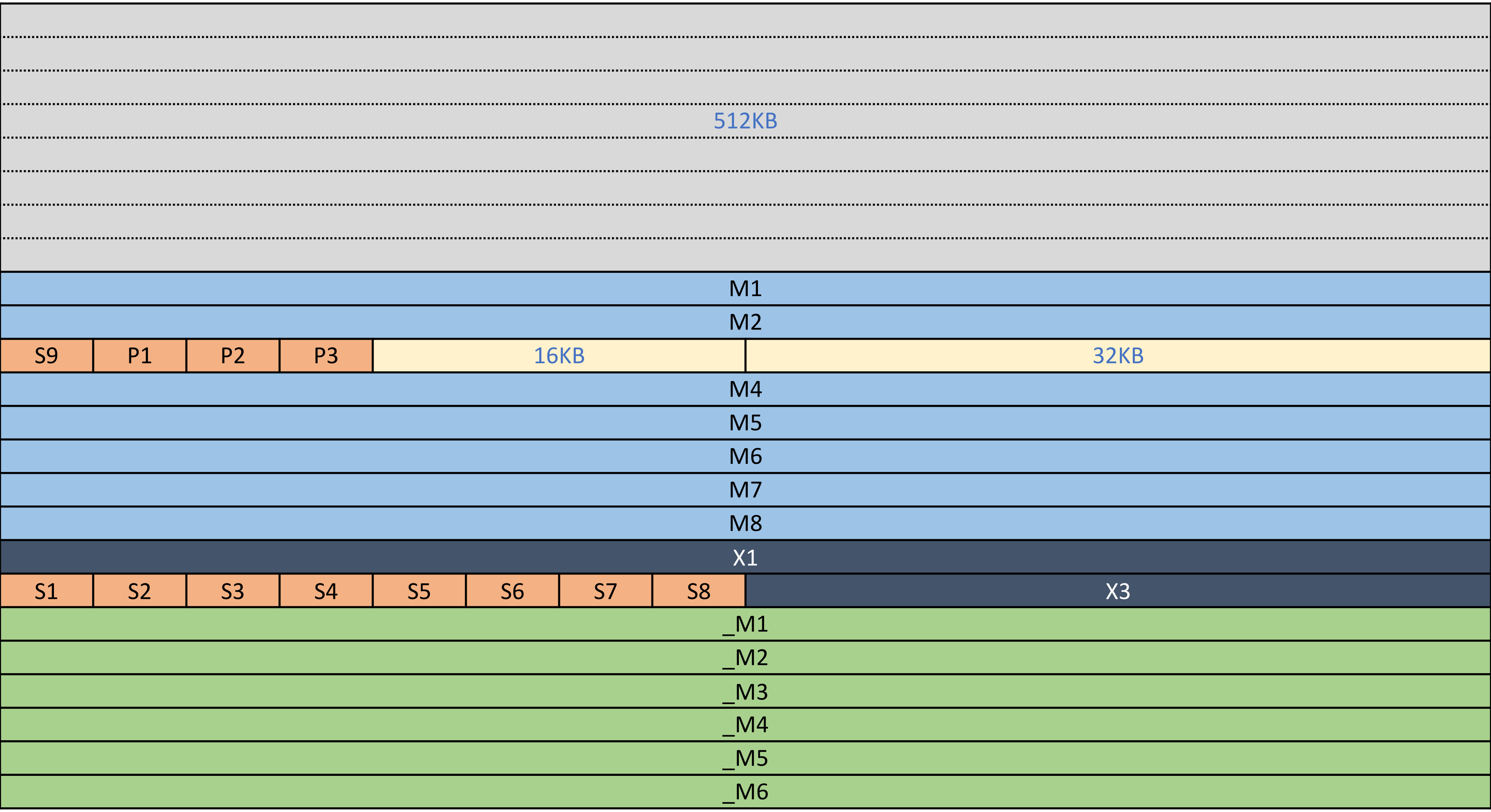
```
Pad(P); // insert padding until vulnerable page
```



Phys Feng Shui step 7/8

Pad *small* chunks until the vulnerable page

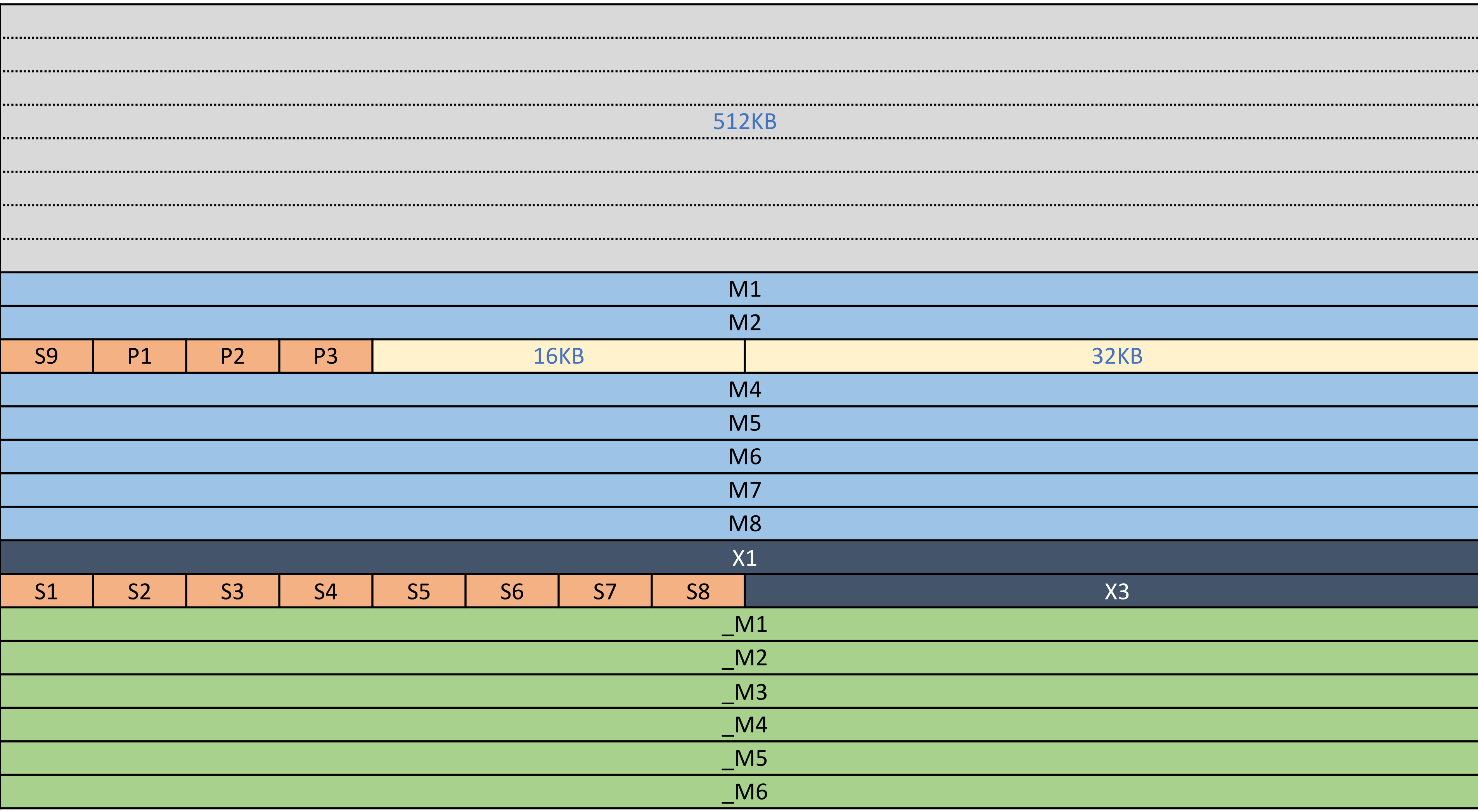
```
Pad(P); // insert padding until vulnerable page
```



Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE

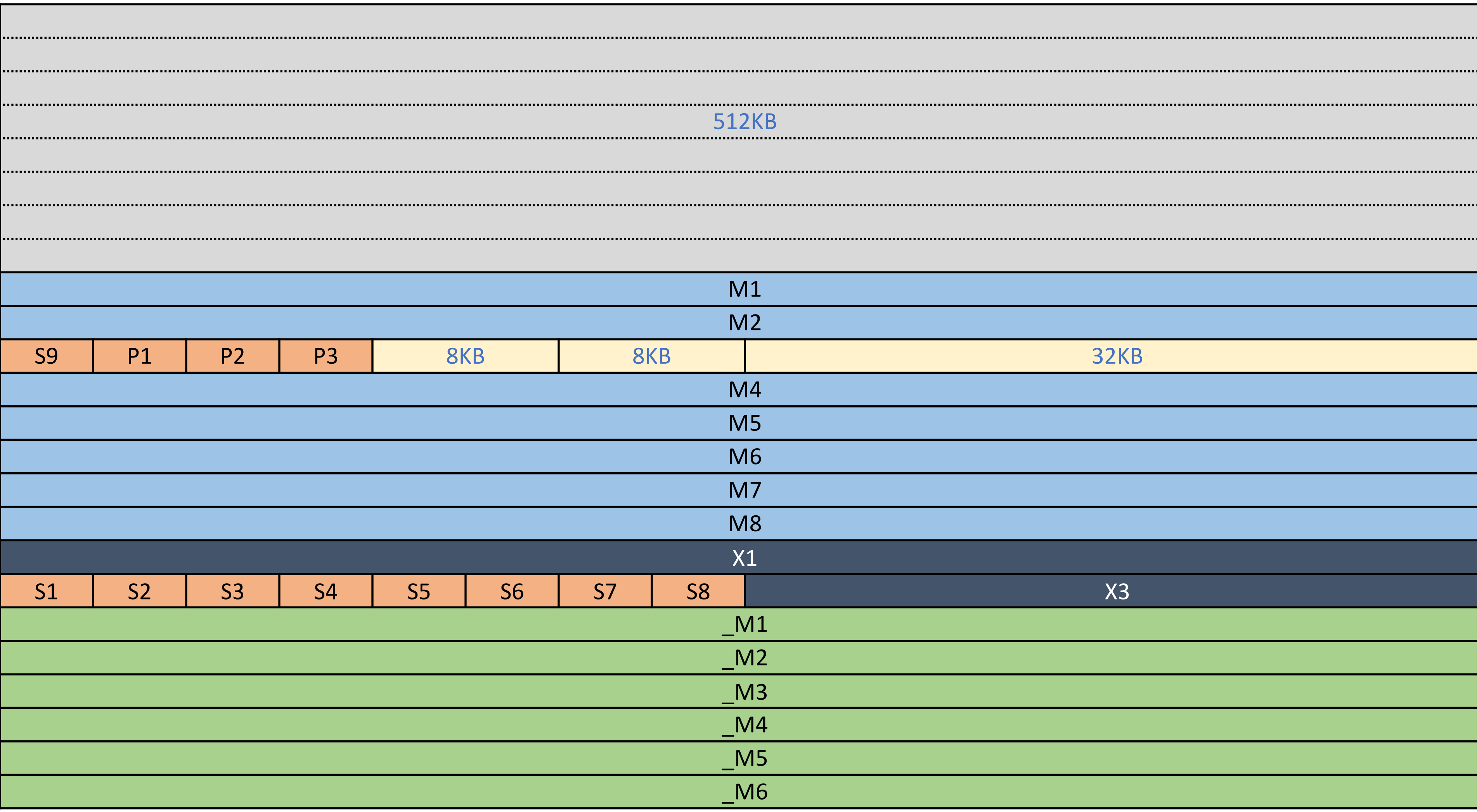
```
PT = mmap(MAP_FIXED); // Force a Page Table allocation
```



Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE

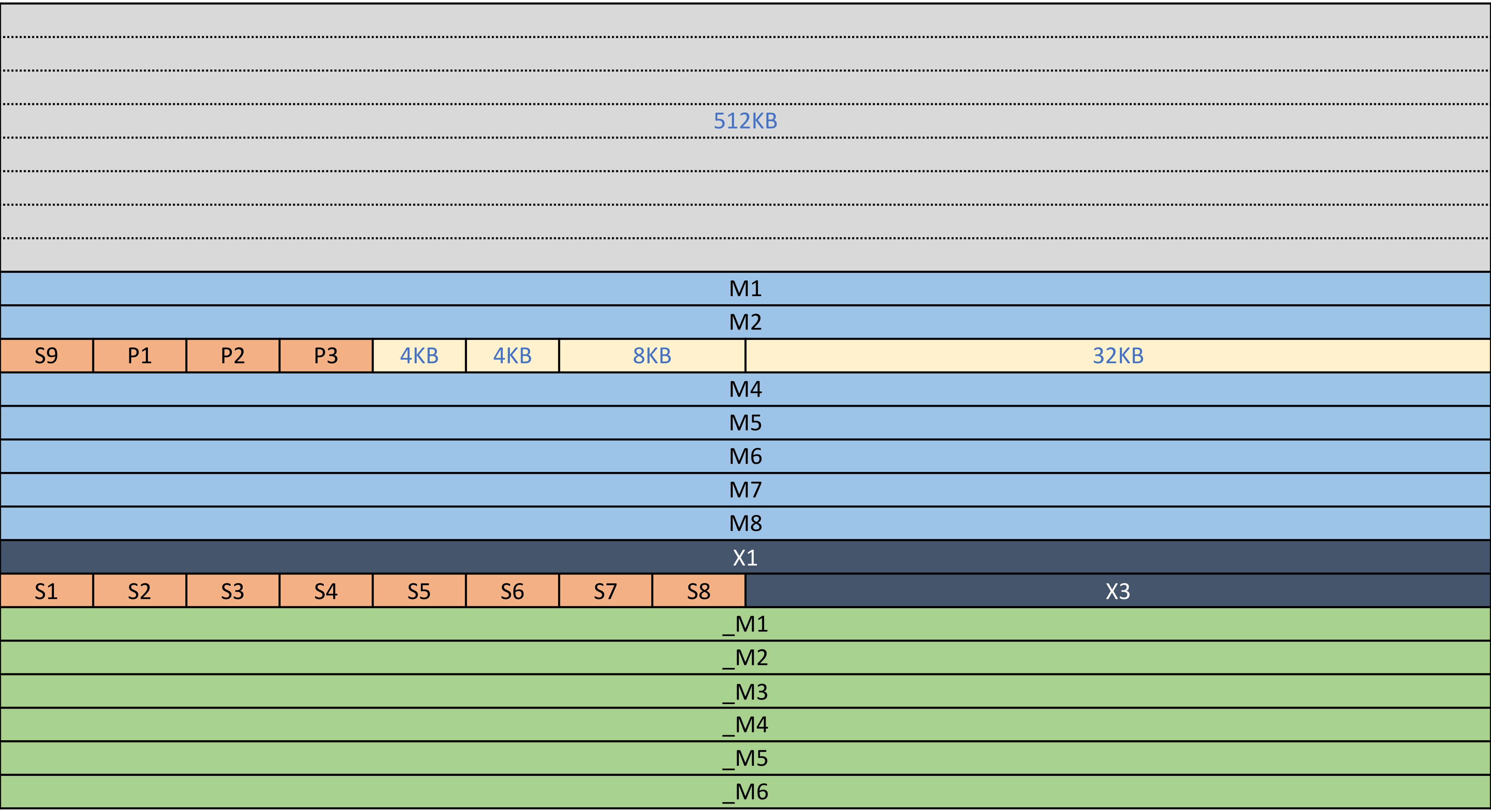
```
PT = mmap(MAP_FIXED); // Force a Page Table allocation
```



Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE

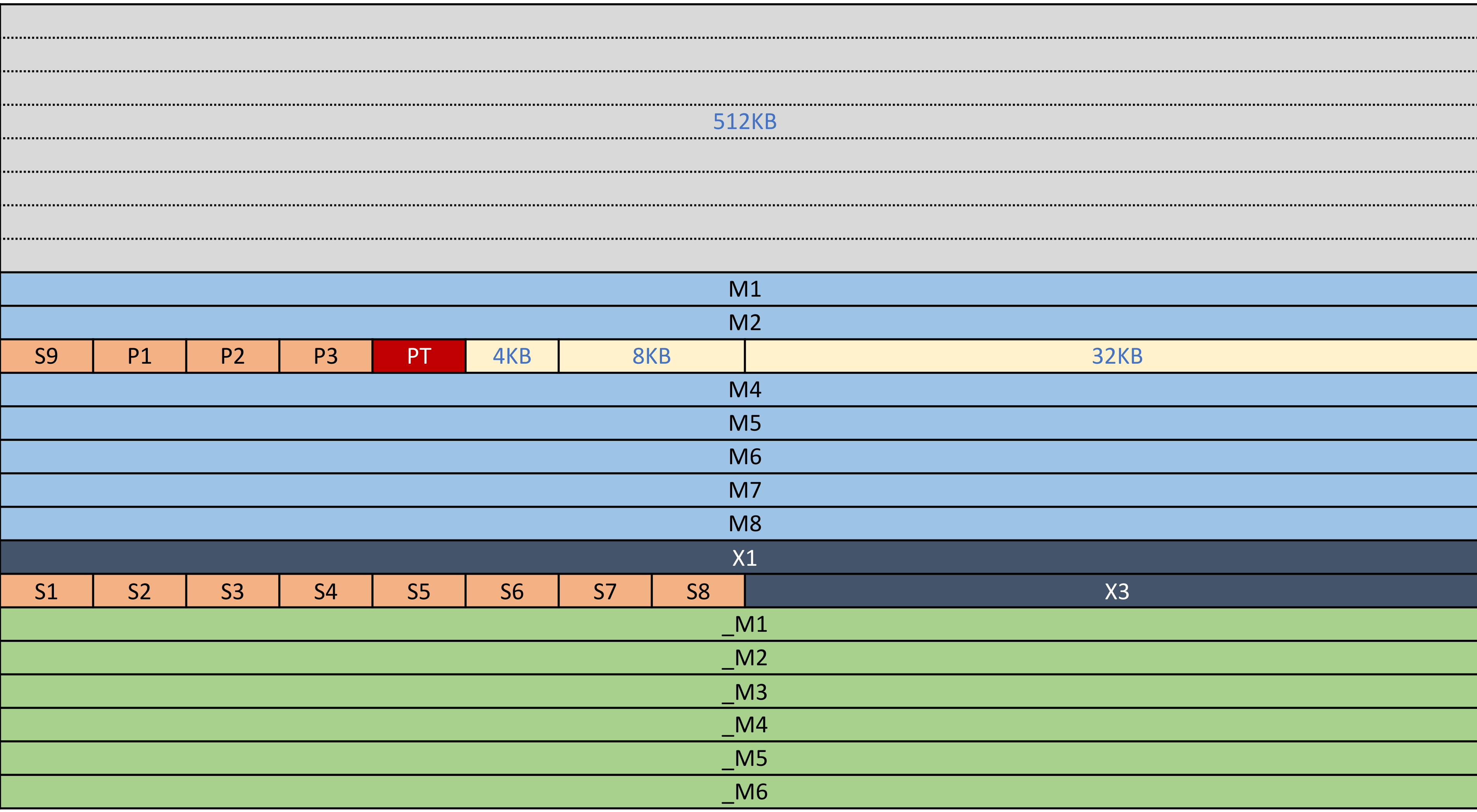
```
PT = mmap(MAP_FIXED); // Force a Page Table allocation
```



Phys Feng Shui step 8/8

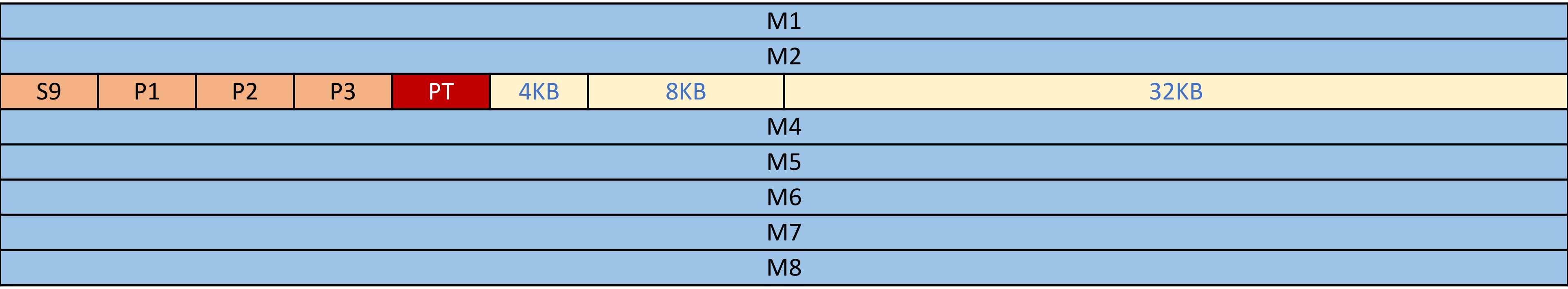
Force a Page Table allocation + map the vulnerable PTE

```
PT = mmap(MAP_FIXED); // Force a Page Table allocation
```



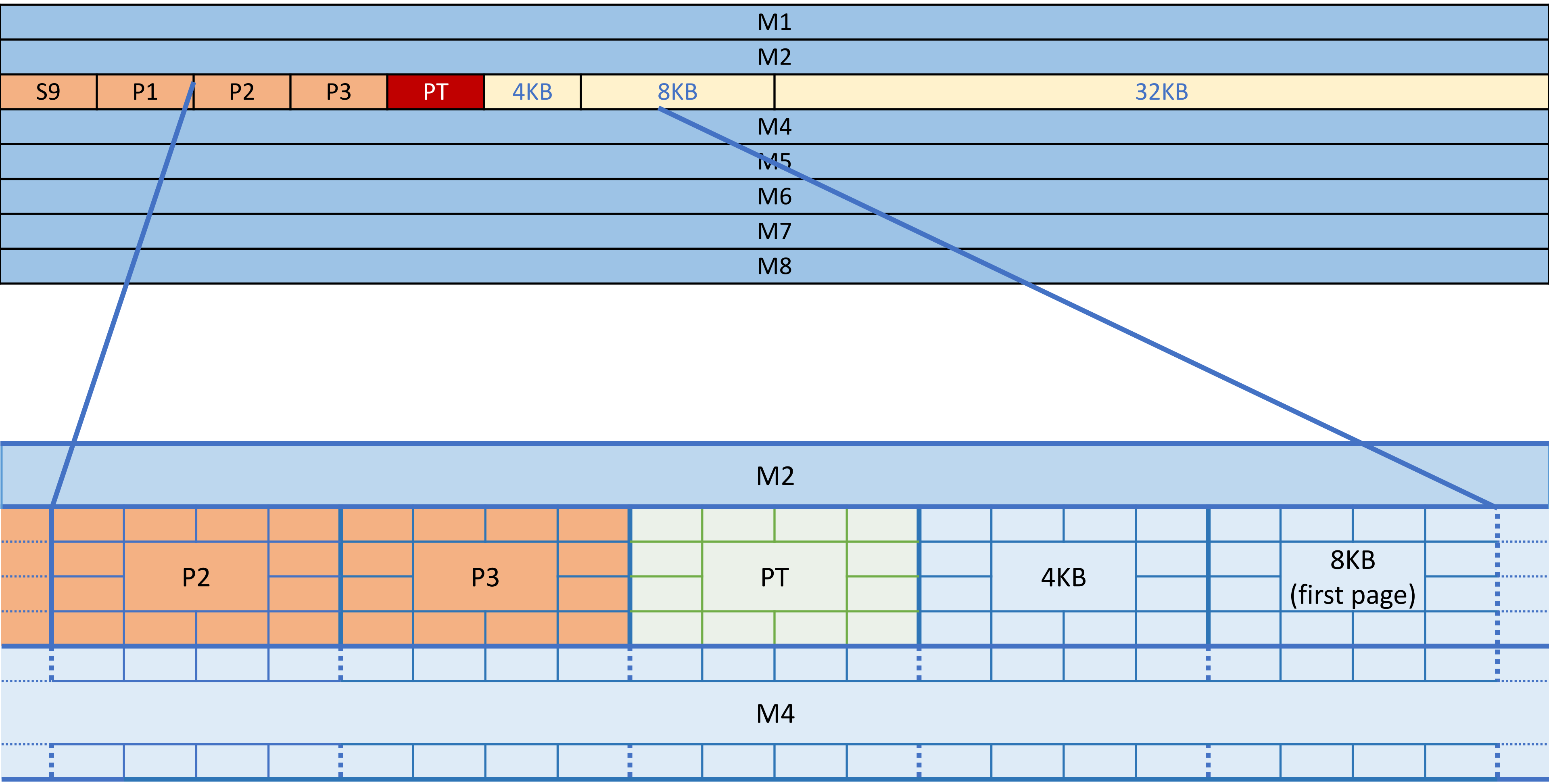
Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE



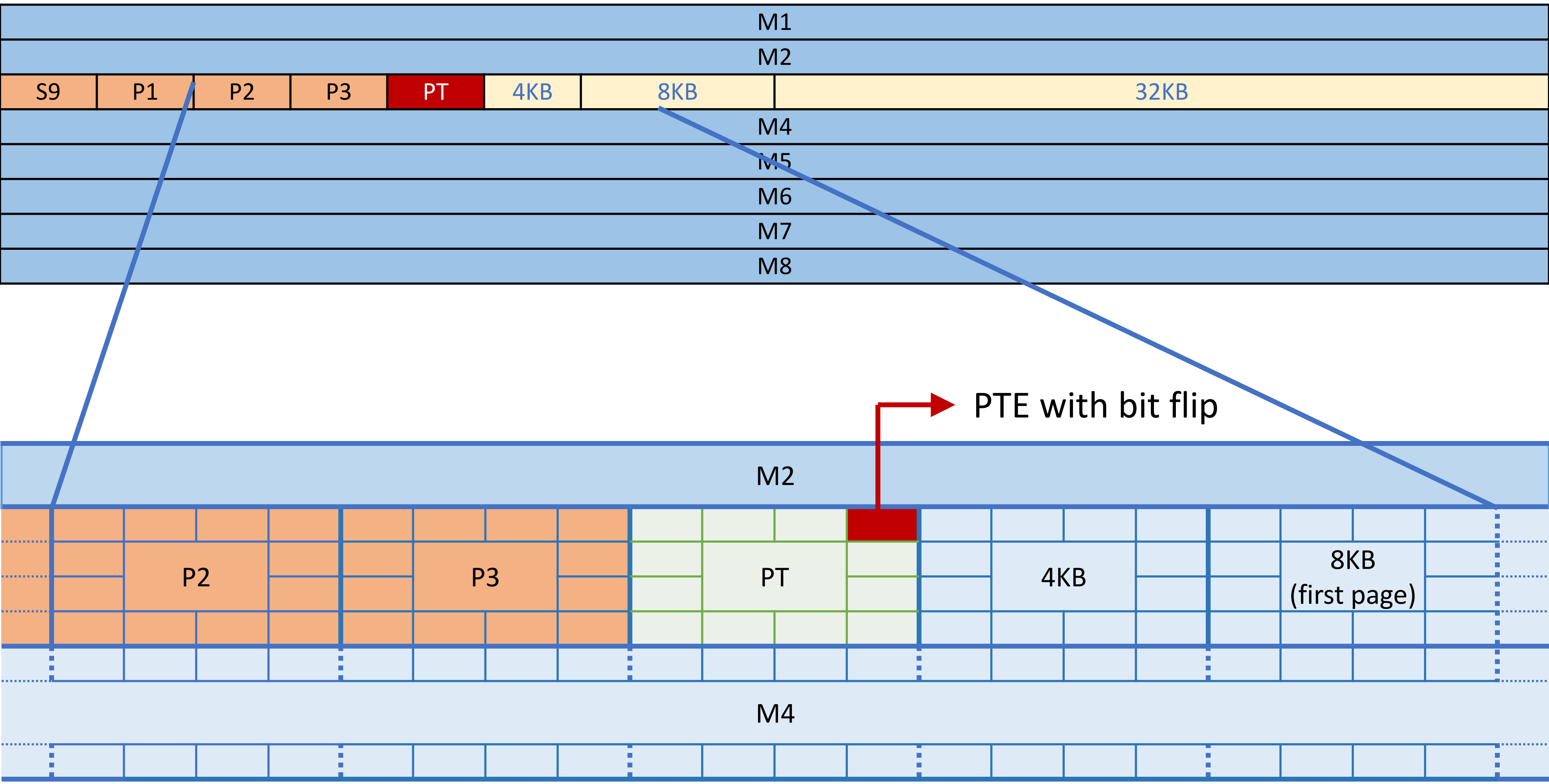
Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE



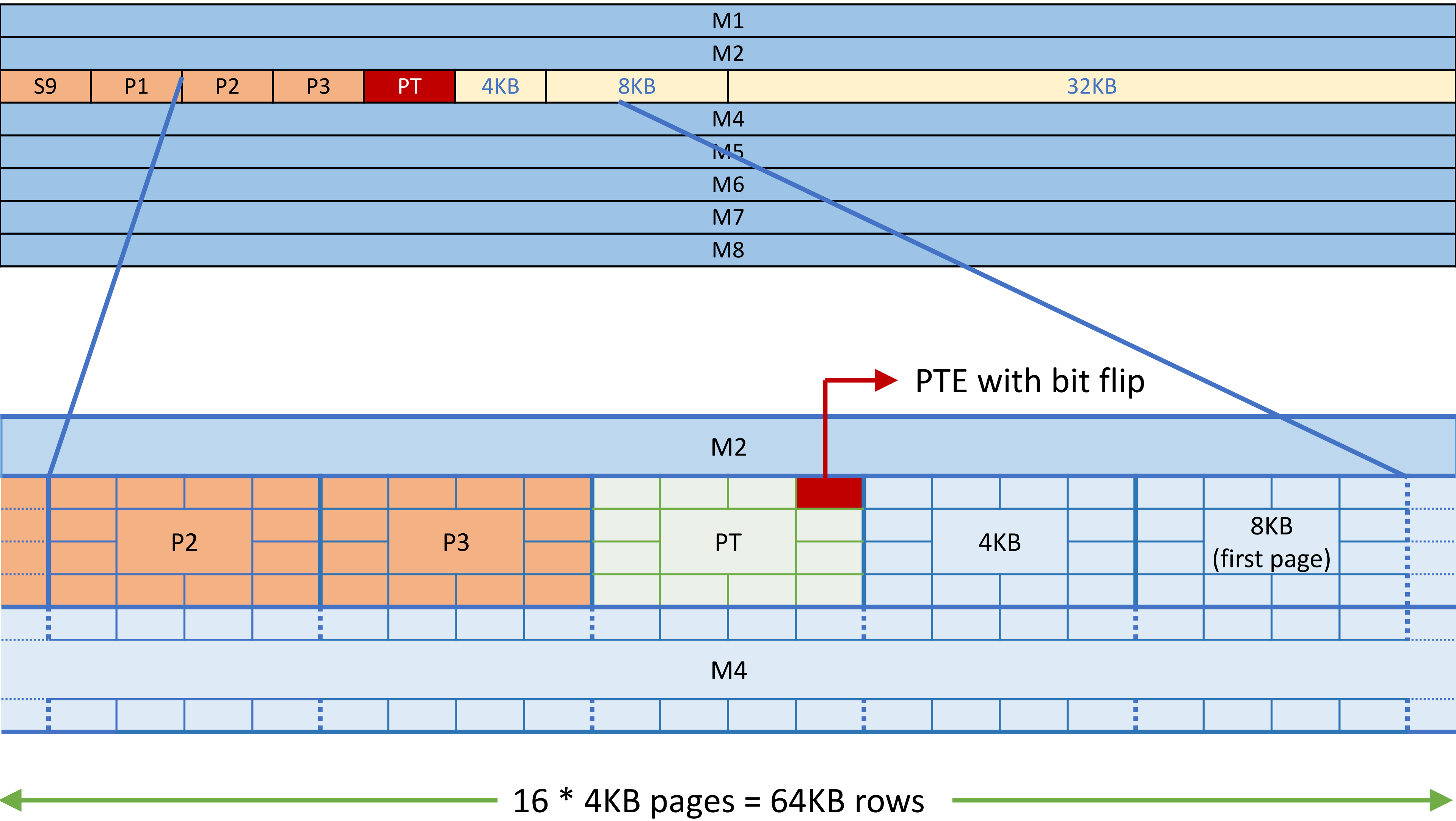
Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE



Phys Feng Shui step 8/8

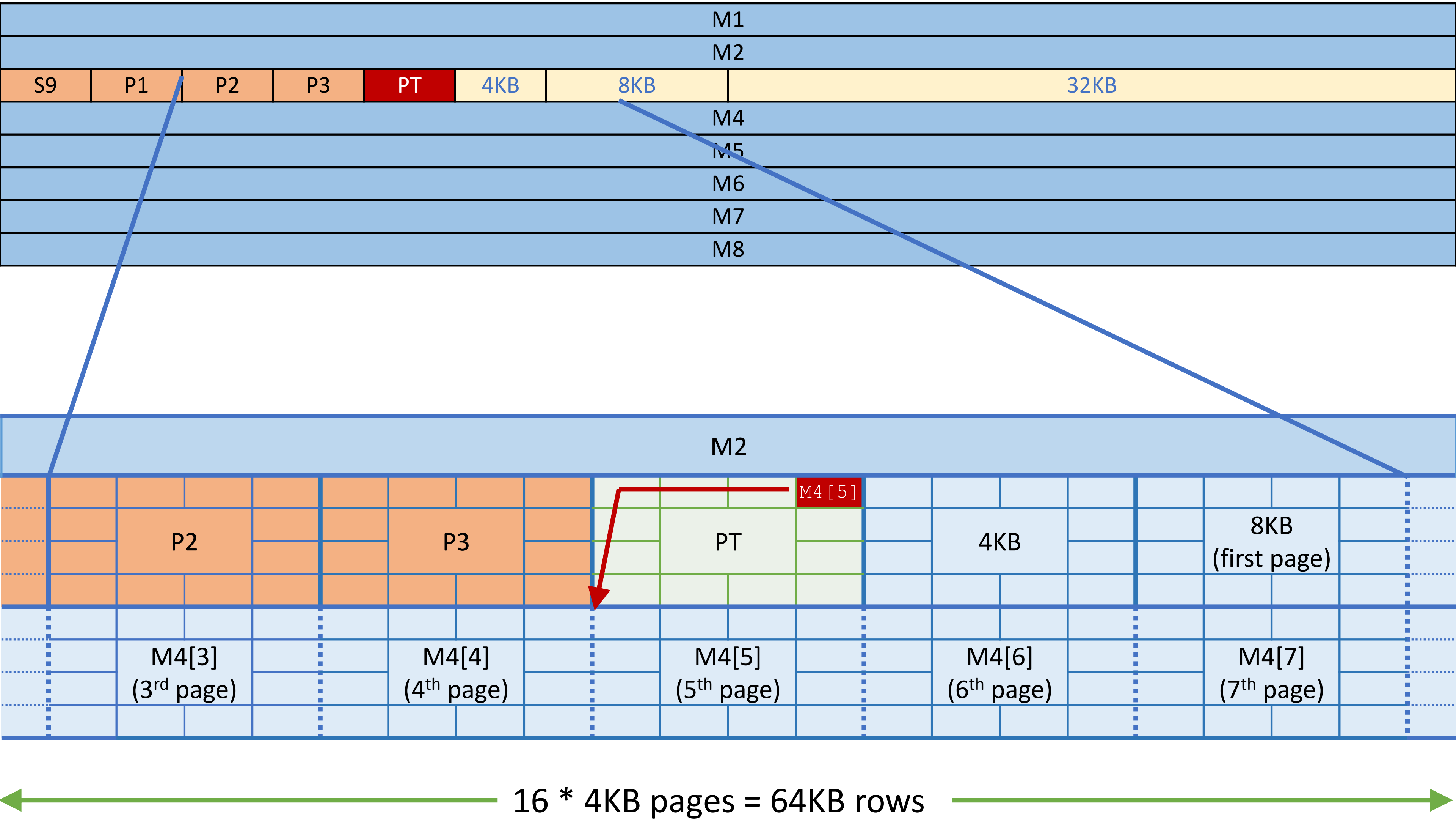
Force a Page Table allocation + map the vulnerable PTE



Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE

```
mmap(M4[5], MAP_FIXED); // map vulnerable PTE 64KB 'away'
```

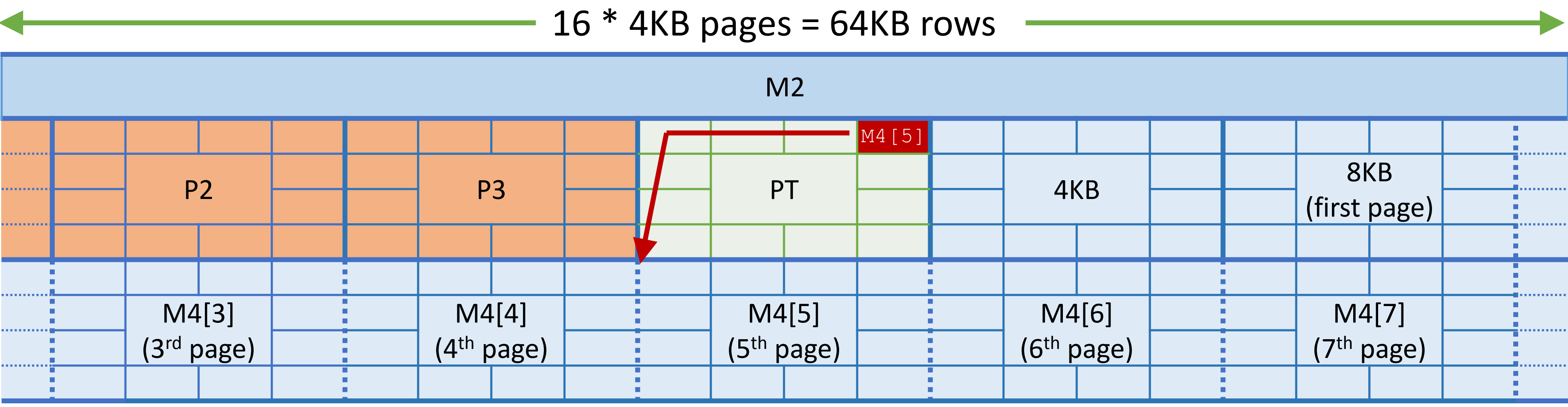


Overview

1. Memory Templating
   Scan memory for useful bit flips
2. Land a Page Table
   Store a page table on a vulnerable page
3. Reproduce the bit flip
   Modify the data structure and get root acces

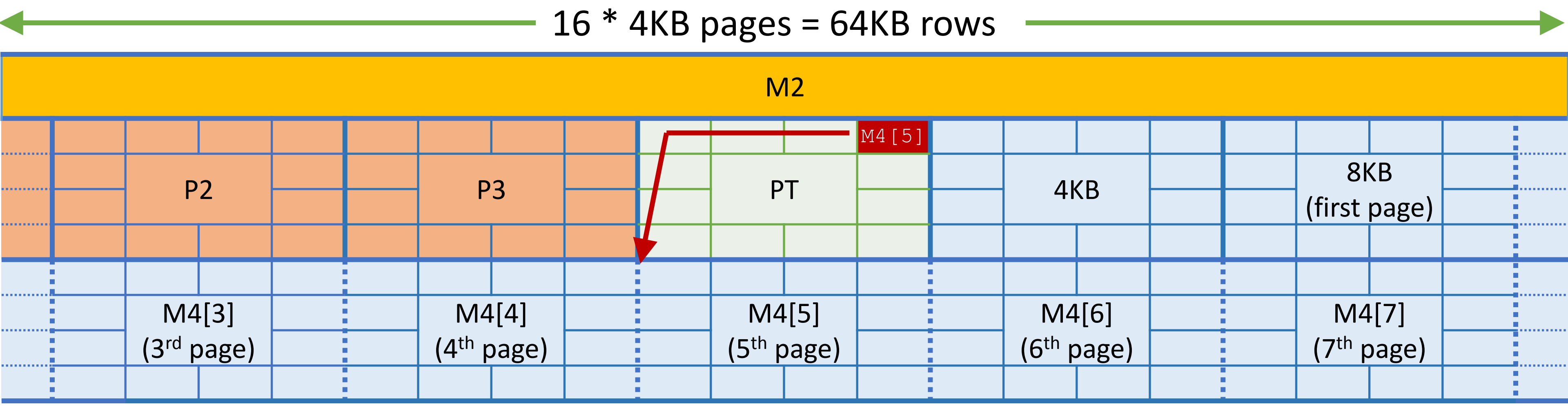
Drammer

Perform double-sided rowhammer to flip a bit in the PTE



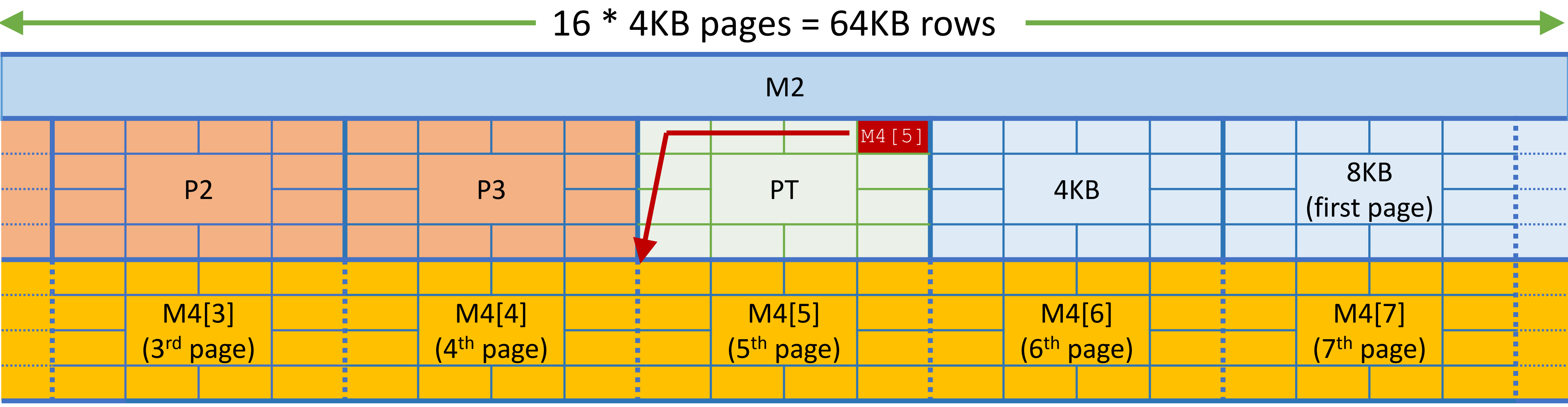
Drammer

Perform double-sided rowhammer to flip a bit in the PTE



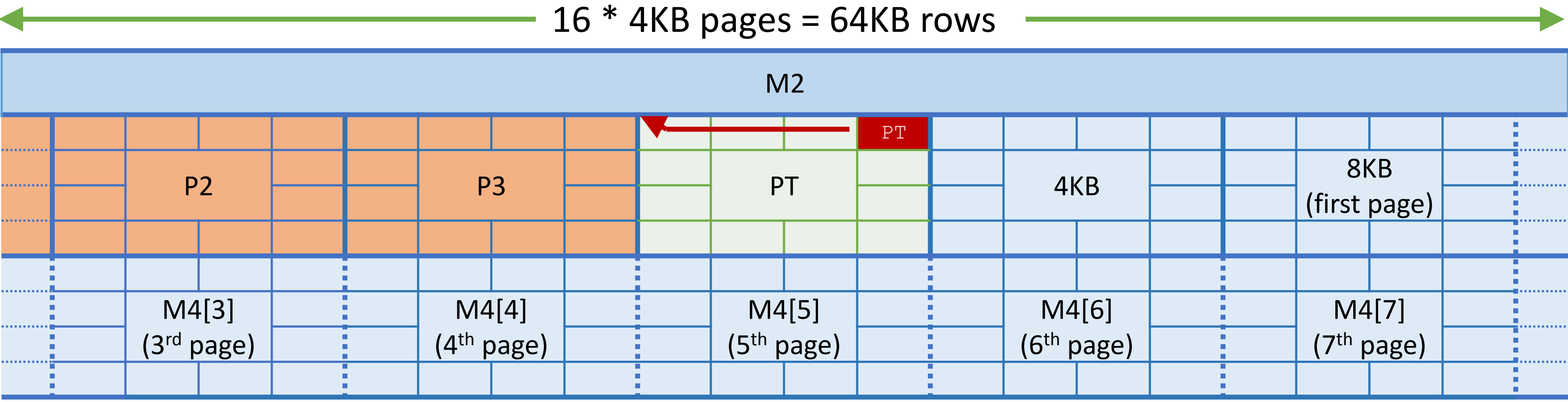
Drammer

Perform double-sided rowhammer to flip a bit in the PTE



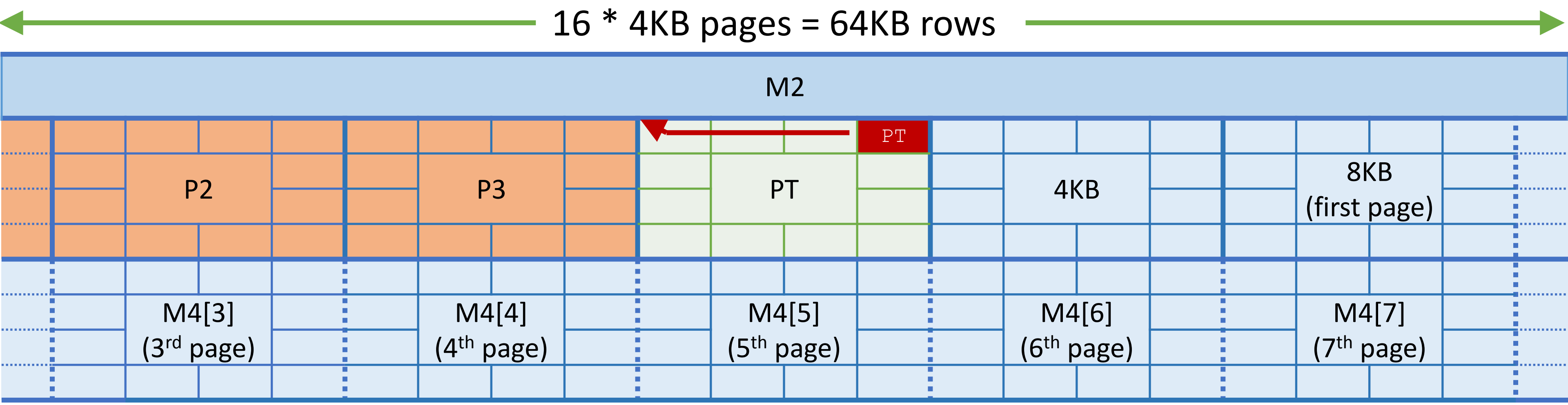
Drammer

Write access to a Page Table



Drammer

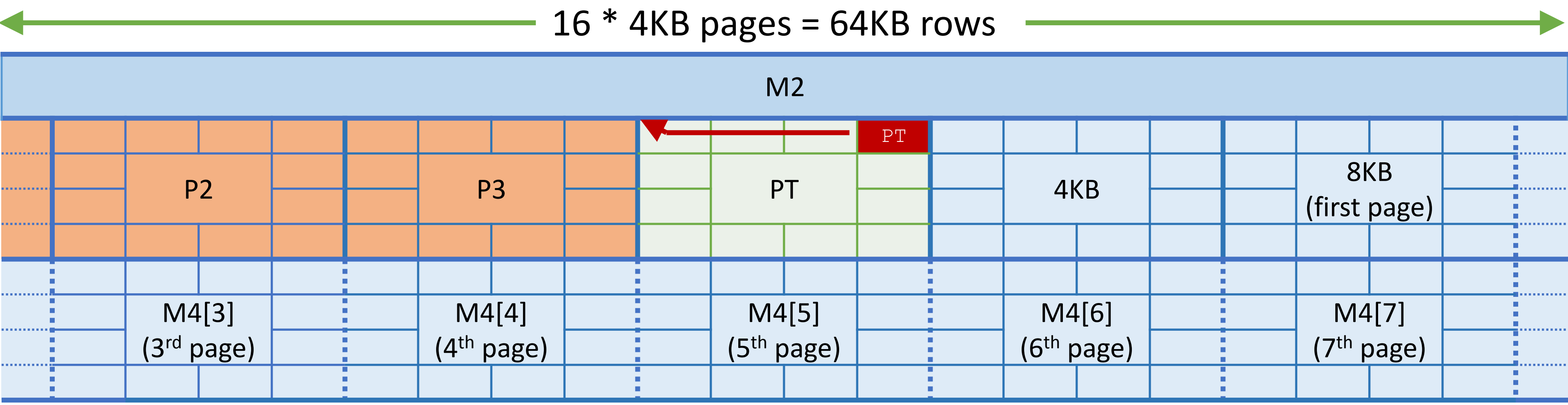
Write access to a Page Table



1. Fill PT with Page Table Entries to kernel memory

Drammer

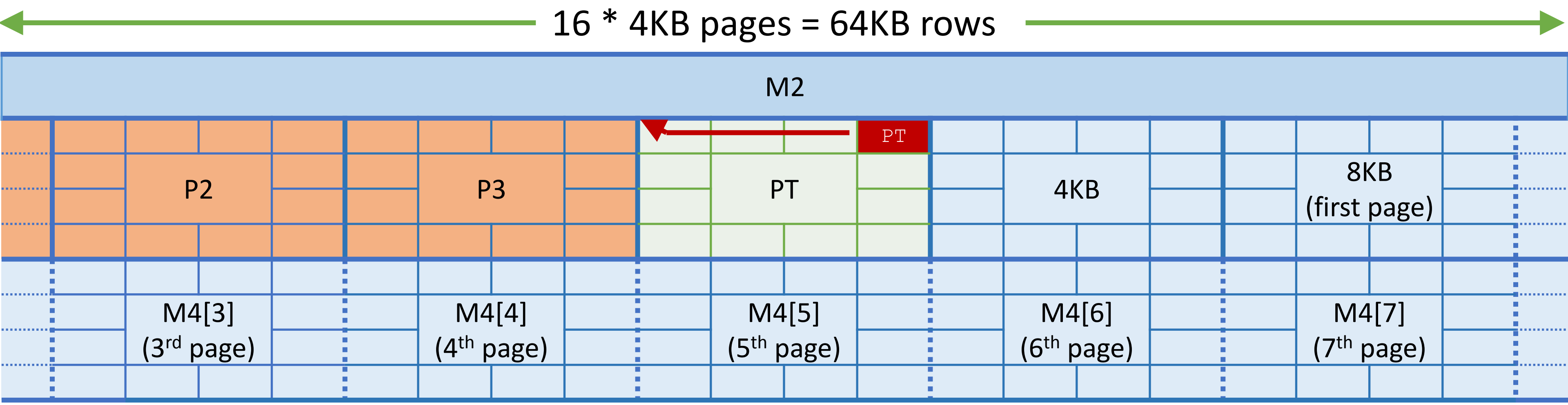
Write access to a Page Table



1. Fill PT with Page Table Entries to kernel memory
2. Search kernel memory for our **`struct cred`**

Drammer

Write access to a Page Table



1. Fill PT with Page Table Entries to kernel memory
2. Search kernel memory for our **`struct cred`**
3. Overwrite our **`uid`** and **`gid`** to get root privileges

Evaluation

Device	#flips	1st exploitable flip after
LG Nexus 5[1]	1058	116s
LG Nexus 5[4]	0	-
LG Nexus 5[5]	747,013	1s
LG Nexus 4	1,328	7s
OnePlus One	3,981	942s
Motorola Moto G (2013)	429	441s
LG G4 (ARMv8 – 64-bit)	117,496	5s

Bit flips on 18 out of 27 tested devices

Evaluation

Device	#flips	1st exploitable flip after
LG Nexus 5[1]	1058	116s
LG Nexus 5[4]	0	-
LG Nexus 5[5]	747,013	1s
LG Nexus 4	1,328	7s
OnePlus One	3,981	942s
Motorola Moto G (2013)	429	441s
LG G4 (ARMv8 – 64-bit)	117,496	5s

Evaluation

Device	#flips	1st exploitable flip after
LG Nexus 5[1]	1058	116s
LG Nexus 5[4]	0	-
LG Nexus 5[5]	747,013	1s
LG Nexus 4	1,328	7s
OnePlus One	3,981	942s
Motorola Moto G (2013)	429	441s
LG G4 (ARMv8 – 64-bit)	117,496	5s

Evaluation

Device	#flips	1st exploitable flip after
LG Nexus 5[1]	1058	116s
LG Nexus 5[4]	0	-
LG Nexus 5[5]	747,013	1s
LG Nexus 4	1,328	7s
OnePlus One	3,981	942s
Motorola Moto G (2013)	429	441s
LG G4 (ARMv8 – 64-bit)	117,496	5s

Evaluation

Device	#flips	1st exploitable flip after
LG Nexus 5[1]	1058	116s
LG Nexus 5[4]	0	-
LG Nexus 5[5]	747,013	1s
LG Nexus 4	1,328	7s
OnePlus One	3,981	942s
Motorola Moto G (2013)	429	441s
LG G4 (ARMv8 – 64-bit)	117,496	5s

After the 1st exploitable flip, exploitation takes at most 22 seconds

Evaluation

Device	#flips	1^{st} exploitable flip after
LG Nexus 5[1]	1058	116s
LG Nexus 5[4]	0	-
LG Nexus 5[5]	747,013	1s
LG Nexus 4	1,328	7s
OnePlus One	3,981	942s
Motorola Moto G (2013)	429	441s
LG G4 (ARMv8 – 64-bit)	117,496	5s

After the 1^{st} exploitable flip, exploitation takes at most 22 seconds

Drammer test app reported bit flips on:

Google Pixel, OnePlus 3, Galaxy Note 7, HTC One M8, …

Disclosure

Contacted Google with a list of suggested mitigations on July 25

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

"Can you publish at another conference, later this year?"

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

"Can you publish at another conference, later this year?"

"What if we support you financially?"

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Ok, could you then perhaps obfuscate some parts of the paper?”

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

"Ok, could you then perhaps obfuscate some parts of the paper?"

Rewarded $4000 for a *critical* issue

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

"Ok, could you then perhaps obfuscate some parts of the paper?"

Rewarded $4000 for a *critical* issue

(because *"it doesn't work on the devices in our Reward Program"*)

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Ok, could you then perhaps obfuscate some parts of the paper?”

Rewarded $4000 for a *critical* issue

(because *“it doesn’t work on the devices in our Reward Program”*)

But now it does

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

"Ok, could you then perhaps obfuscate some parts of the paper?"

Rewarded $4000 for a *critical* issue

Partial hardening in November's updates

"We will continue to work on a longer term solution"

Conclusion

- Deterministic Rowhammer exploitation
- **No special memory management features required (e.g., deduplication)**
- ARM memory controllers are fast enough to do Rowhammer
- LPDDR* found vulnerable
- No easy software fix

Conclusion

- Deterministic Rowhammer exploitation
- No special memory management features required (e.g., deduplication)
- ARM memory controllers are fast enough to do Rowhammer
- LPDDR* found vulnerable
- No easy software fix
- **Using DMA bypasses state-of-the-art defenses (e.g., ANVIL)**

Conclusion

- Deterministic Rowhammer exploitation
- No special memory management features required (e.g., deduplication)
- ARM memory controllers are fast enough to do Rowhammer
- LPDDR* found vulnerable
- No easy software fix
- Using DMA bypasses state-of-the-art defenses (e.g., ANVIL)

- More details
  - Demos, statistics and test app:
    https://vusec.net/projects/drammer
  - Open source:
    https://github.com/vusec/drammer