Drammer: Flip Feng Shui Goes Mobile

Victor van der Veen
@vvdveen

VUSec.net

Binary Armoring
Malware Analysis
Hardware Vulnerabilities

Mobile Security
Software Reliability
Software Testing

Drammer

Your takeaway message of today

Rowhammer on ARM

Reliable exploitation

Also on a Google Pixel

Flip Feng Shui in 2016

The art of turning bit flips into a reliable compromise…

… of the cloud

- Hammering a Needle in the Software Stack @ USENIX Security

… of the browser

- Dedup Est Machina @ IEEE Security & Privacy

… of the mobile

- Drammer @ ACM CCS

Dutch high quality research at top venues in computer security

Best research group in the world in 2016 (better than MIT!)

Flip Feng Shui
Worldwide impact

ArsTechnica, WIRED, The Register, Infoworld, Slashdot, The Stack, Softpedia, Science Daily, CORDIS, Security Now!, Daily Mail, Tech Times, Fossbytes, The Inquirer, Hack Read, Threatpost

Argentina (Segu-info), **Austria** (Der Standard), **Belgium** (DeMorgen), **China** (Freebuff, Sohu, EEPW), **Czech Republic** (Svět Androida), **Denmark** (Version2), **France** (Silicon, Le Monde Informatique, Informanews), **Germany** (Der Spiegel, Golem.de, Pro-Linux, Crn.de, JAXenter, Computer Bild, t3n Magazine, Netzwelt.de), **Hungary** (HWSW), **Italy** (Repubblica.it, Punto Informatico, Gadgetblog.it, Tutto Android), **Mexico** (PCWorld Mexico), **The Netherlands** (NU.nl, Tweakers.net, Crimesite), **Norway** (Digi.no), **Poland** (eGospodarka, Softonet, PCLab.pl, Dobreprogramy, PC Format, Telix.pl), **Russia** (Хакер, Securitylab.ru), **Slovakia** (Živé.sk), Spain (López Dóriga, CSO, El Android libre), **Switzerland** (Neue Zürcher Zeitung), **Taiwan** (iThome), **Turkey** (Teknokulis, CHIP, Webtekno), and **Ukraine** (KO).

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]

Flip Feng Shui
Worldwide impact

ArsTechnica, WIRED, The Register, Infoworld, Slashdot, The Stack, Softpedia, Science Daily, CORDIS, Security Now!, Daily Mail, Tech Times, Fossbytes, The Inquirer, Hack Read, Threatpost

Argentina (Segu-info), **Austria** (Der Standard), **Belgium** (DeMorgen), **China** (Freebuff, Sohu, EEPW), **Czech Republic** (Svět Androida), **Denmark** (Version2), **France** (Silicon, Le Monde Informatique, Informanews), **Germany** (Der Spiegel, Golem.de, Pro-Linux, Crn.de, JAXenter, Computer Bild, t3n Magazine, Netzwelt.de), **Hungary** (HWSW), **Italy** (Repubblica.it, Punto Informatico, Gadgetblog.it, Tutto Android), **Mexico** (PCWorld Mexico), **The Netherlands** (NU.nl, Tweakers.net, Crimesite), **Norway** (Digi.no), **Poland** (eGospodarka, Softonet, PCLab.pl, Dobreprogramy, PC Format, Telix.pl), **Russia** (Хакер, Securitylab.ru), **Slovakia** (Živé.sk), Spain (López Dóriga, CSO, El Android libre), **Switzerland** (Neue Zürcher Zeitung), **Taiwan** (iThome), **Turkey** (Teknokulis, CHIP, Webtekno), and **Ukraine** (KO).

- *Powerful Bit-Flipping Attack*
  Bruce Schneier
- *This is a significant finding, thanks for Researching it*
  Mark Seaborn, Google
- *This is an issue that will affect almost any modern computer system*
  Hugo Vincent, ARM Research
- *We appreciate your assistance in helping to maintain and improve the security of our products.*
  Cristina Formaini, Apple

INTRODUCTION

Flip Feng Shui Requirements

1. A *reproducible* hardware bit flip

   Rowhammer – a widespread DRAM issue

2. Ability to target the bit flip

   Predictable memory management behavior

Rowhammer

A DRAM hardware glitch

Memory cells (capacitors) have a natural discharge rate

refresh every 64ms

Activating neighboring cells increases the discharge rate

Rowhammer

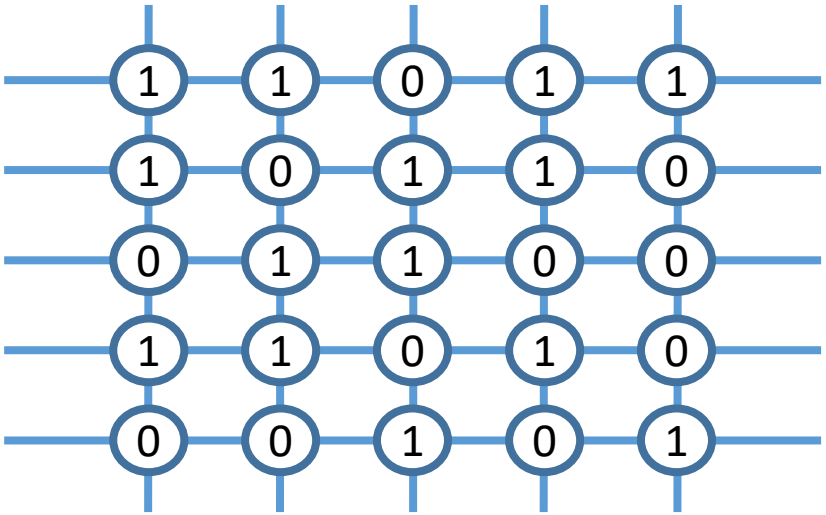
- Victim cell is charged to represent 1
- Neighboring cells are accessed frequently
- Victim cell leaks charge below a certain threshold
- When read, victim cell is interpreted 0

Disturbance error!

Rowhammer

A DRAM hardware glitch

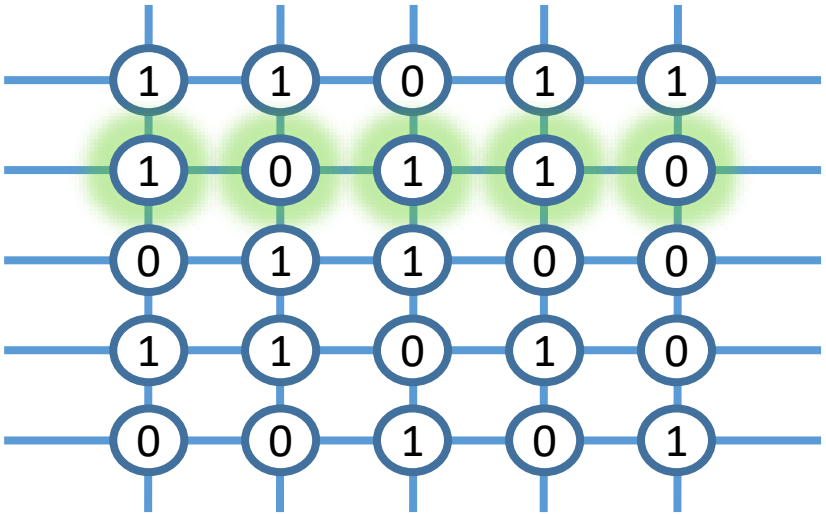
Single-Sided Rowhammer



Rowhammer

A DRAM hardware glitch

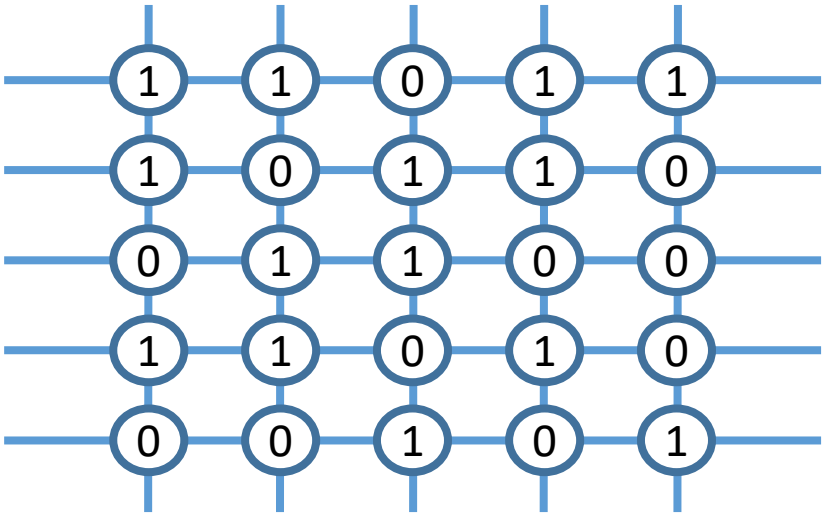
Single-Sided Rowhammer



Rowhammer

A DRAM hardware glitch

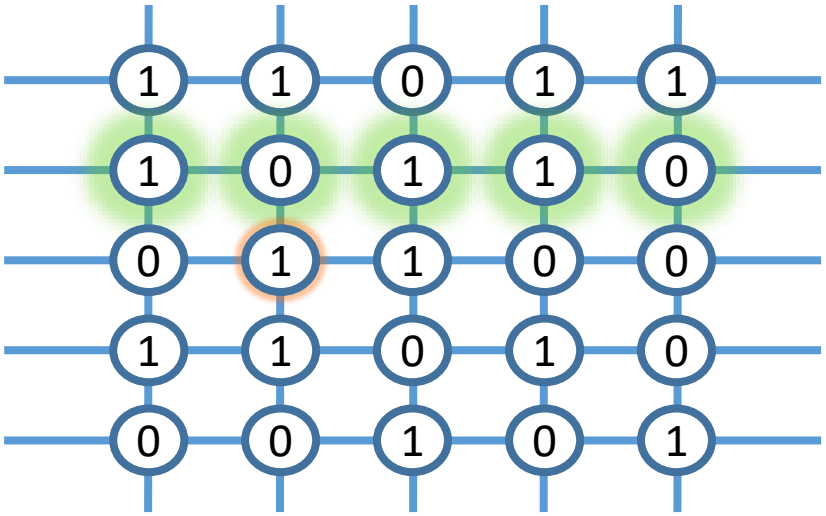
Single-Sided Rowhammer



Rowhammer

A DRAM hardware glitch

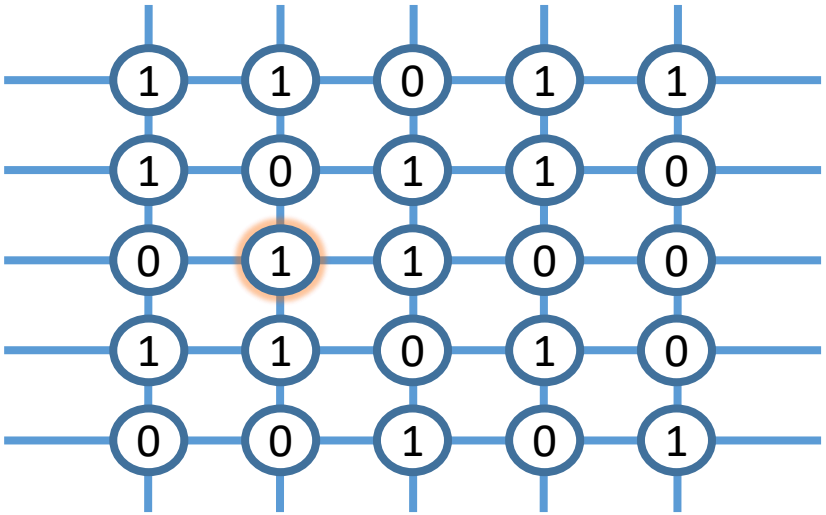
Single-Sided Rowhammer



Rowhammer

A DRAM hardware glitch

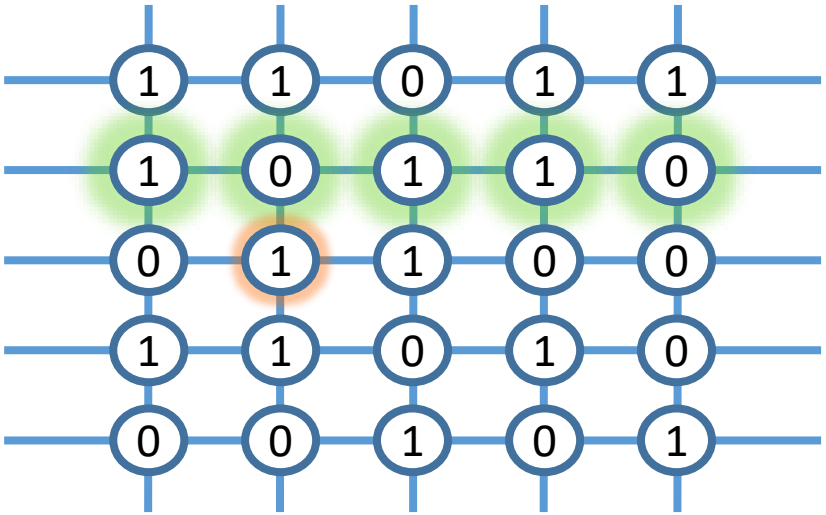
Single-Sided Rowhammer



Rowhammer

A DRAM hardware glitch

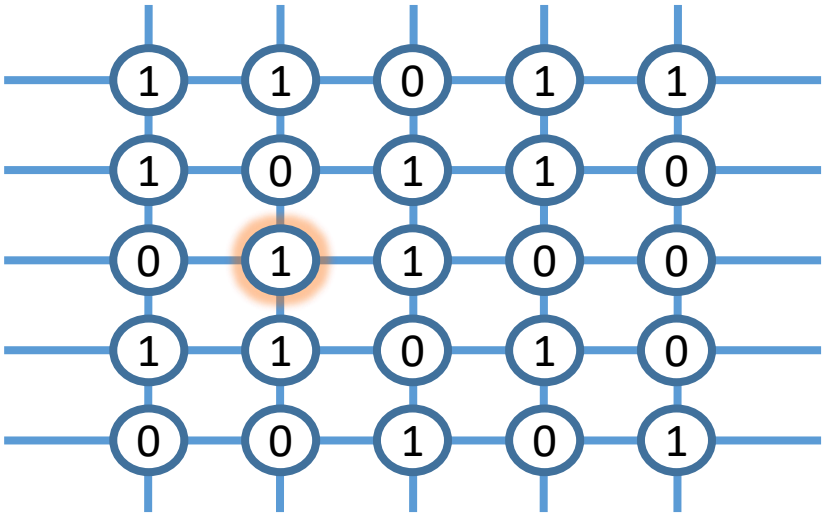
Single-Sided Rowhammer



Rowhammer

A DRAM hardware glitch

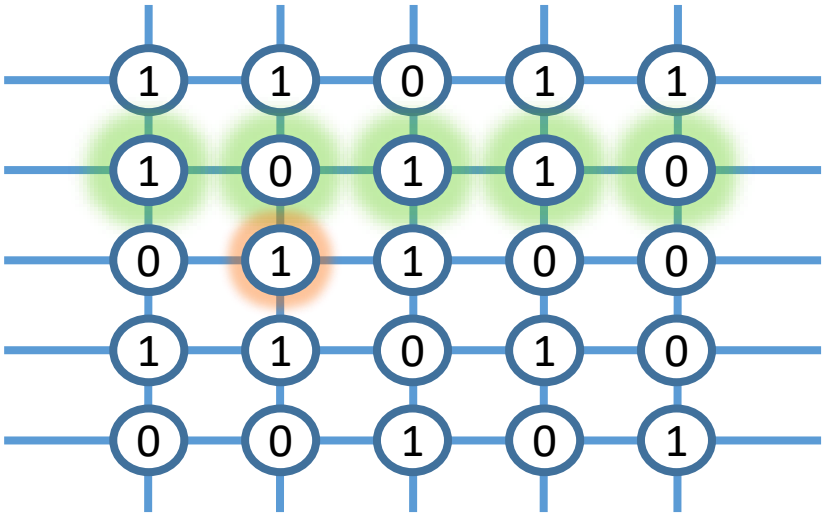
Single-Sided Rowhammer



Rowhammer

A DRAM hardware glitch

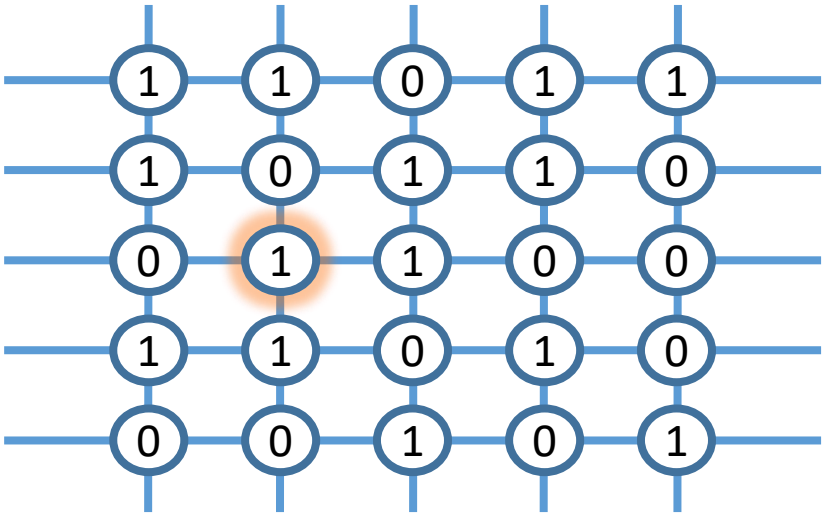
Single-Sided Rowhammer



Rowhammer

A DRAM hardware glitch

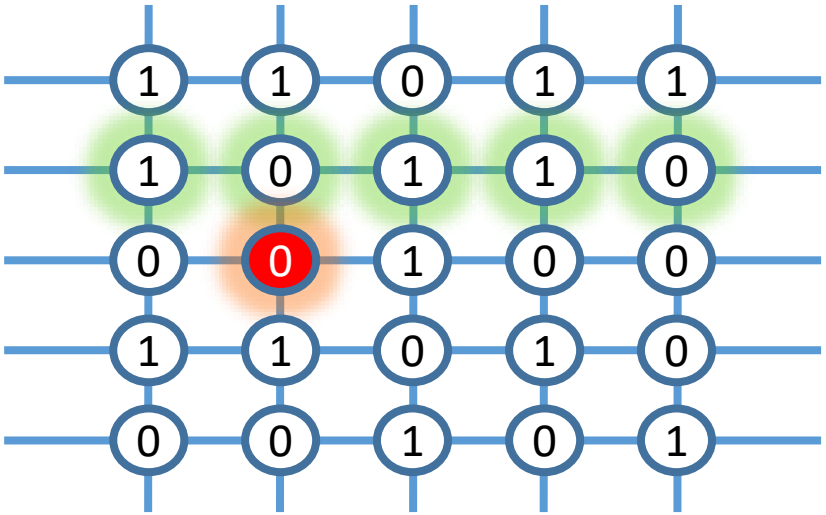
Single-Sided Rowhammer



Rowhammer

A DRAM hardware glitch

Single-Sided Rowhammer

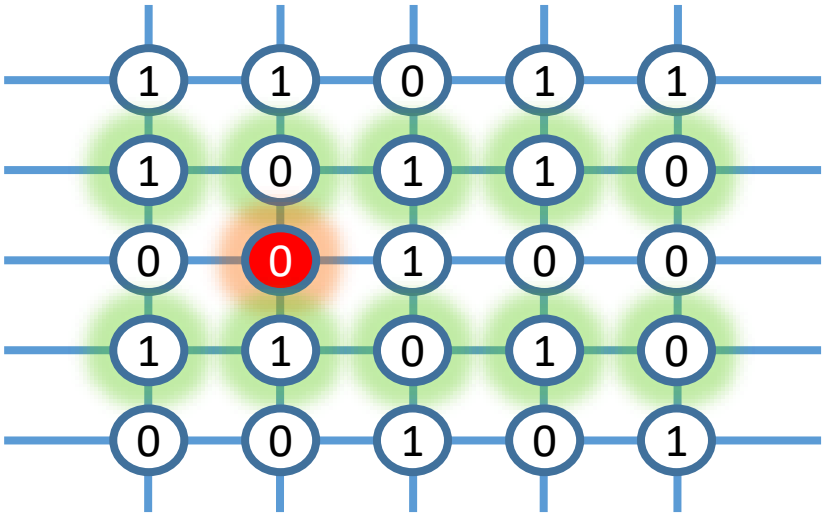


- Not every bit may flip
- Once a bit flips, we can often reproduce it

Rowhammer

A DRAM hardware glitch

~~Single~~Double-Sided Rowhammer

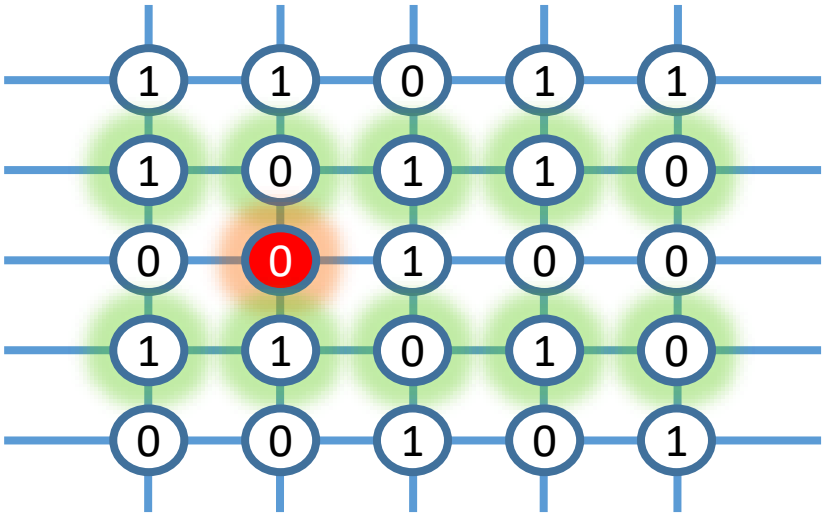


- Sandwich the victim row by alternating accesses
- Increased chance of flipping a bit
- Harder requirements

Rowhammer

A DRAM hardware glitch

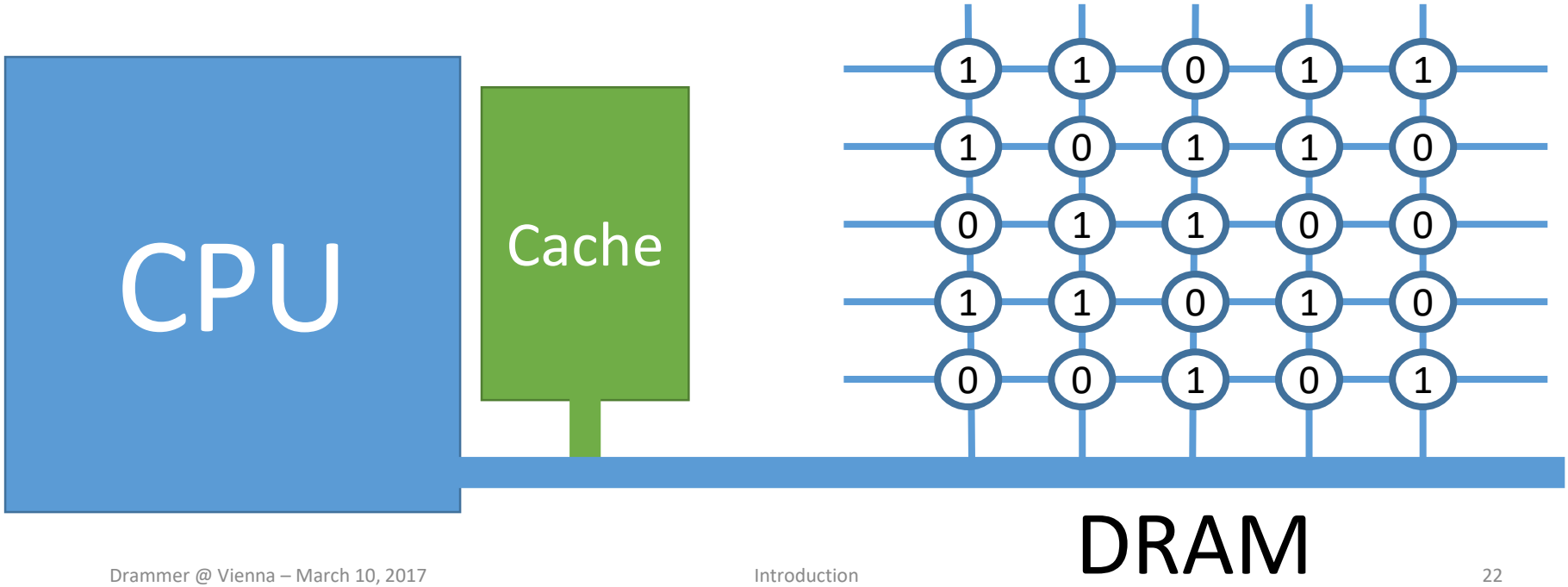
Challenges for an attacker



1. **Bypass the CPU Cache**
   Most memory accesses are served by the CPU cache
2. **Select the Aggressor Rows (for double-sided Rowhammer)**
   How do we know from which address to read from?

Bypass the CPU cache

- DRAM accesses are slow
- CPU caches data for fast access
- To Rowhammer, we must read many times from DRAM



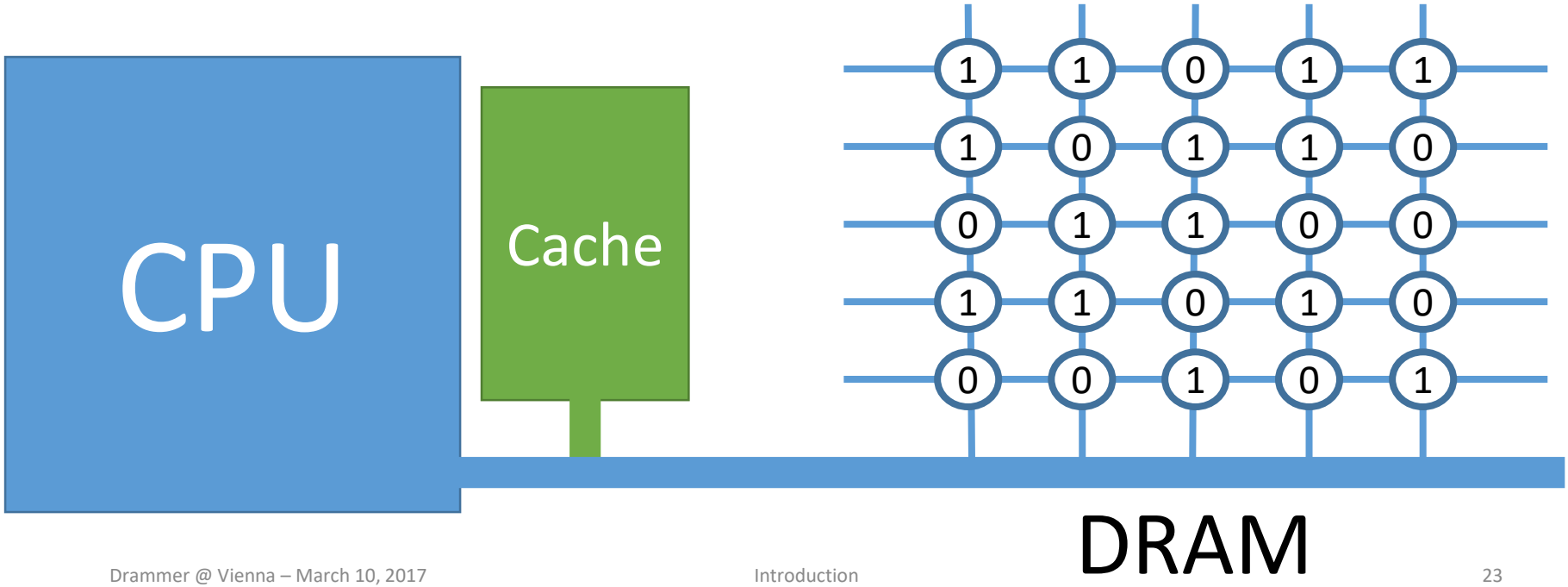
Bypass the CPU cache

Solution: cache flush

Tell the CPU to remove data from the cache

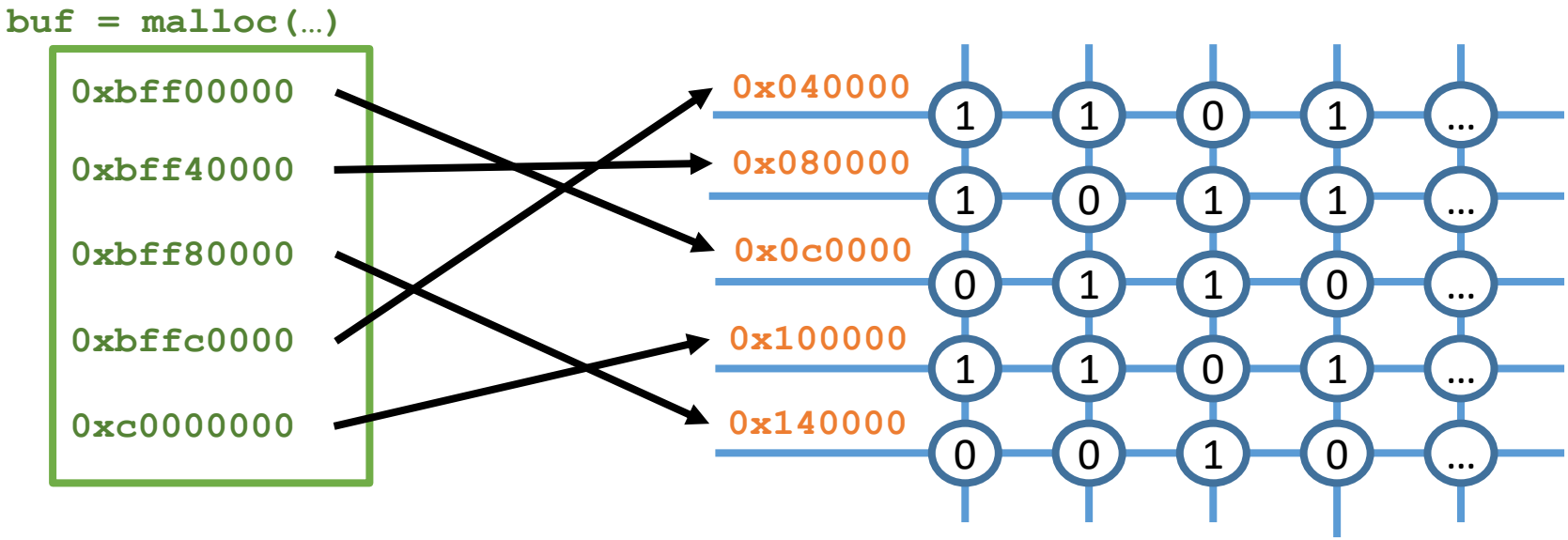
Solution: cache eviction

Read more data until the cache is full



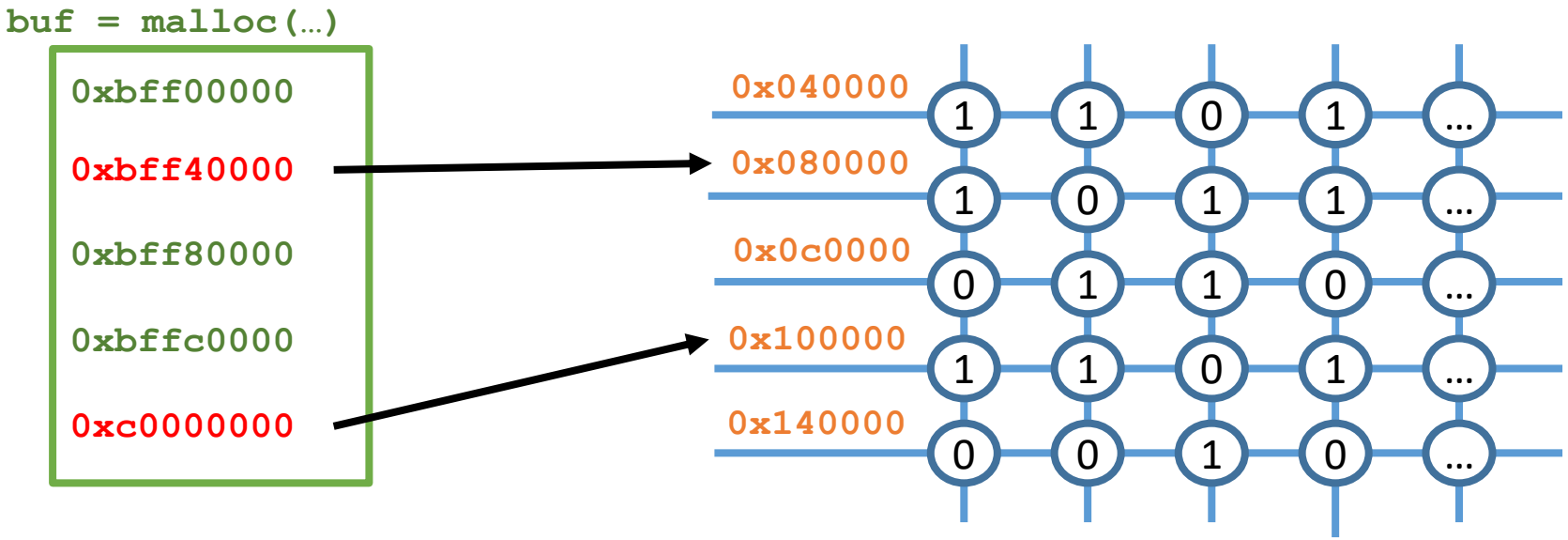
Select the Aggressor Rows

- DRAM works with **physical memory addresses**
- Software works with **virtual addresses**
- Unpredictable mapping of virtual to physical addresses



Select the Aggressor Rows

- DRAM works with **physical memory addresses**
- Software works with **virtual addresses**
- Unpredictable mapping of virtual to physical addresses
- How do we select the addresses to read from?

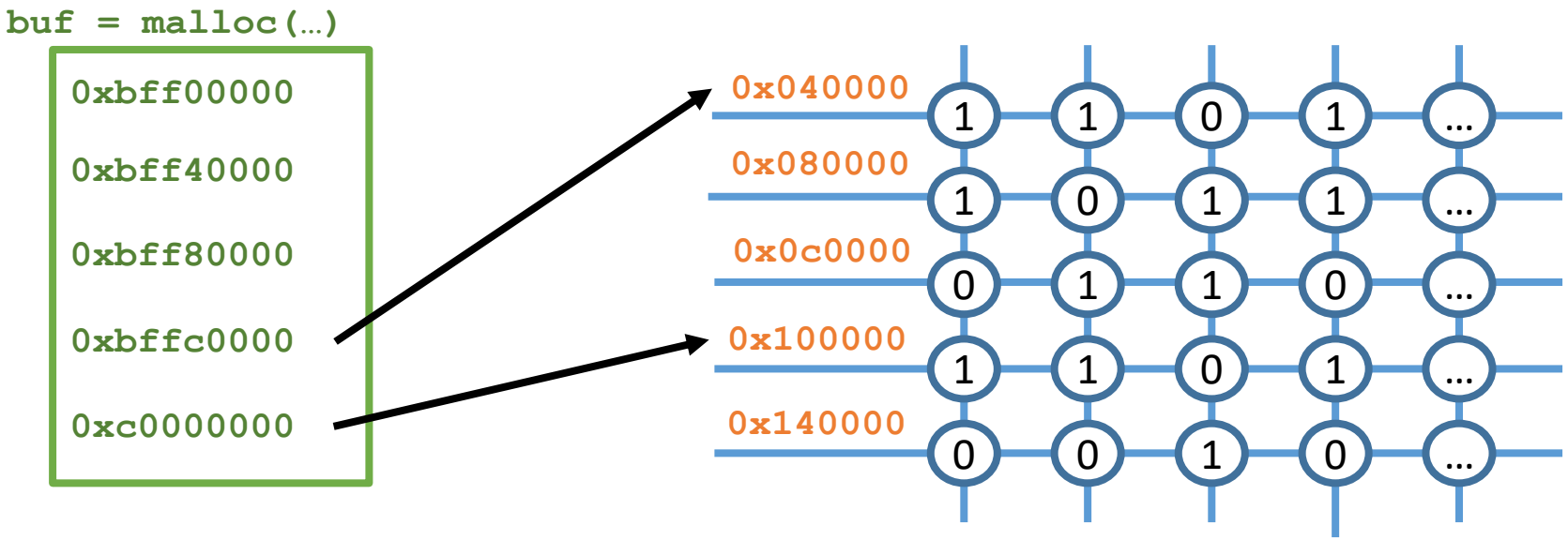


Select the Aggressor Rows

Solution: single sided Rowhammer

Pick any two addresses and ‘just’ hammer

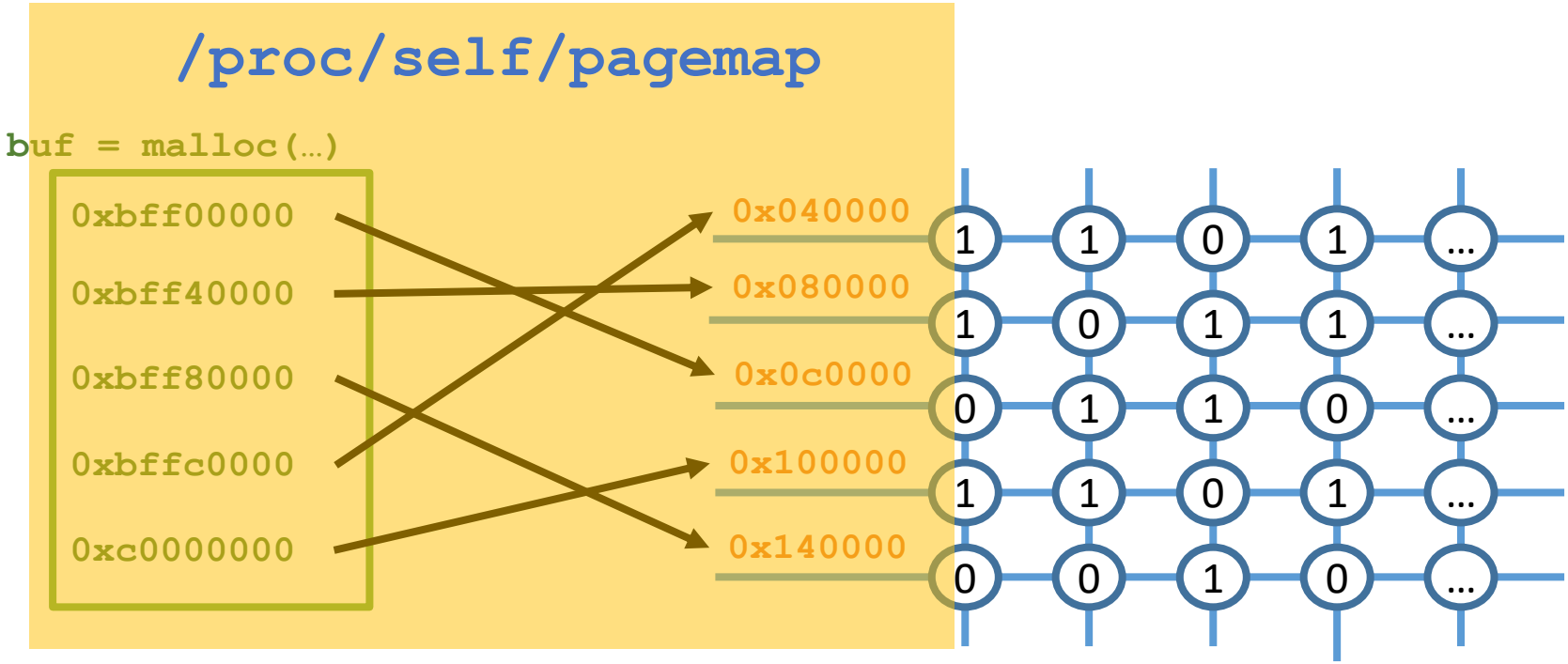
- Less interference between rows, so less bit flips
- Still good enough with buggy DRAM



Select the Aggressor Rows

Solution: pagemap

Special file that holds the mapping of virtual to physical addresses



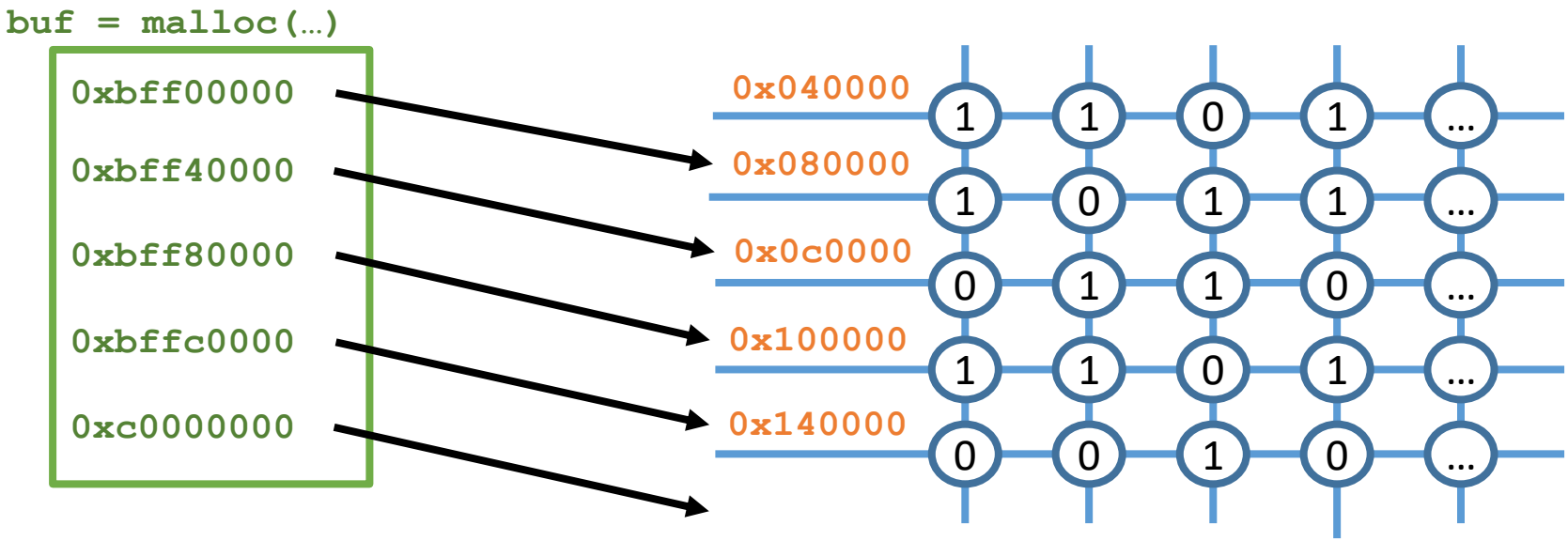
Select the Aggressor Rows

Solution: pagemap

Special file that holds the mapping of virtual to physical addresses

Solution: huge pages

Virtual allocations mapped to **physically contiguous** memory



Flip Feng Shui Mechanics

1. Memory templating

   Scan memory for useful bit flips

2. Land sensitive data

   Store sensitive data on a vulnerable location

3. Reproduce the bit flip

   Modify the sensitive data to compromise the system

Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos

FLIP FENG SHUI

USENIX Security
August 2016

Hammering a Needle in the Software Stack

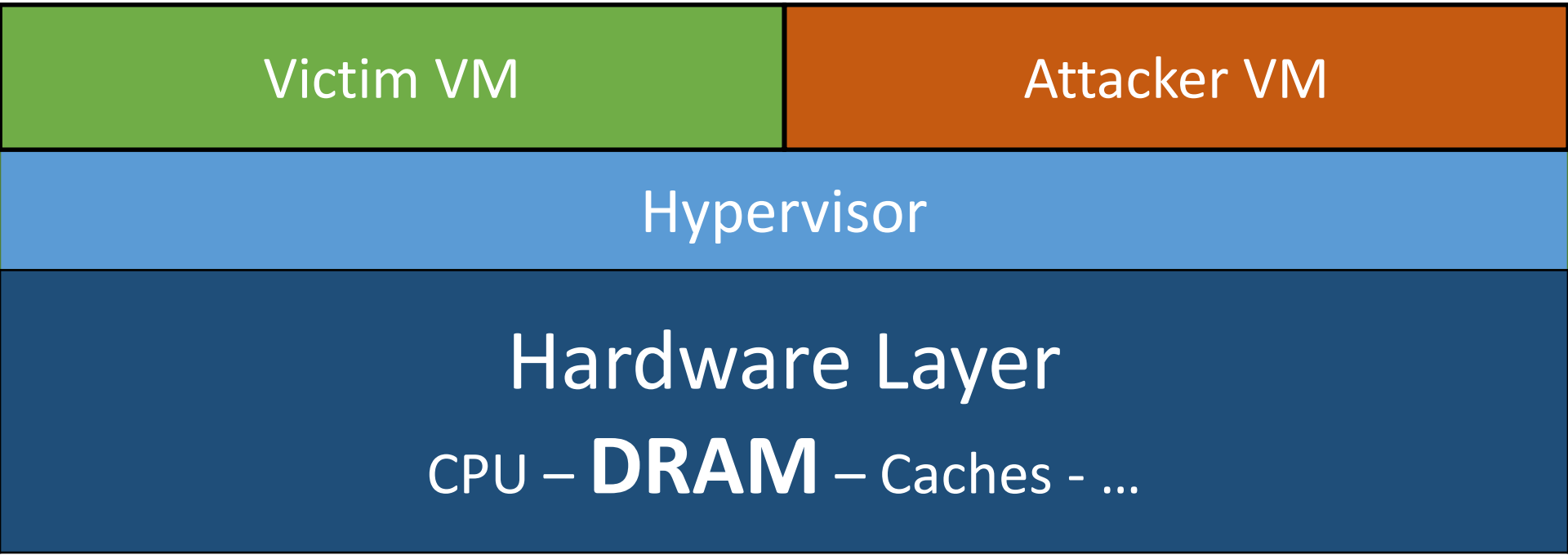
- Malicious VM in the cloud compromising neighbors
- Rowhammer + Memory Deduplication
- You can flip a bit in another VM

What would YOU flip?

Memory templating

Victim and Attacker share hardware

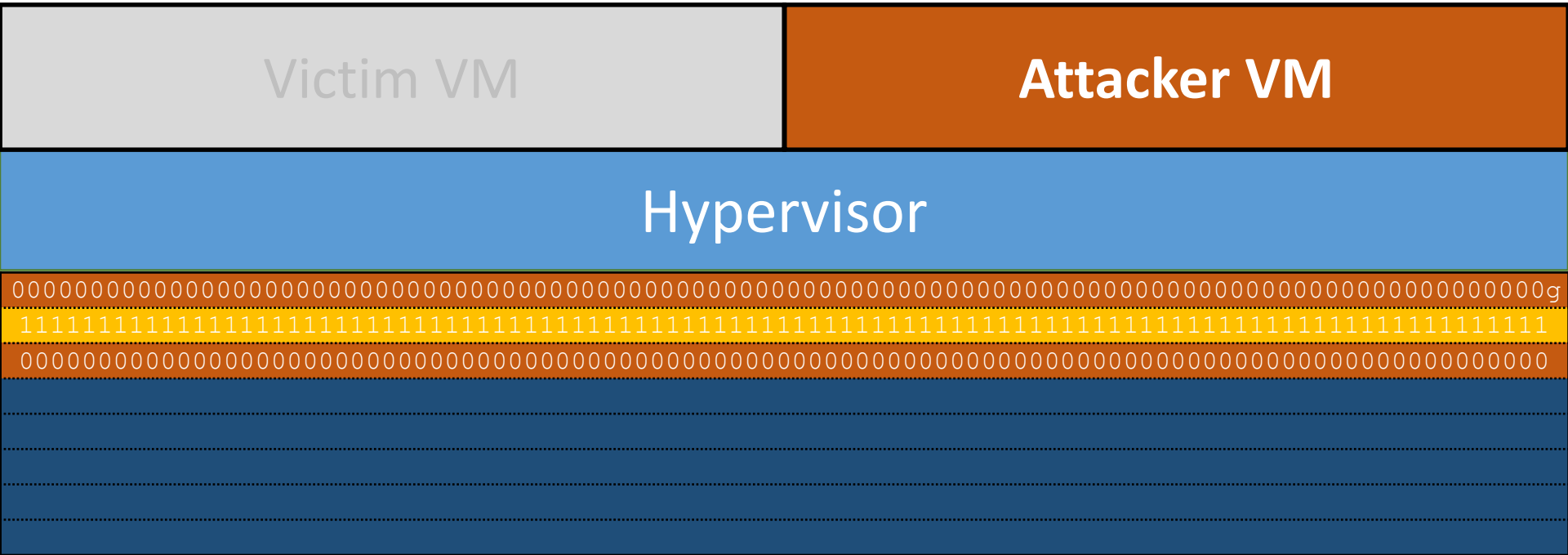
- Attacker has access to the **same** physical memory as the victim
- Attacker is root on his own VM: use **cache flush** to Rowhammer
- Use **huge pages** to select aggressor rows



Memory templating

Victim and Attacker share hardware

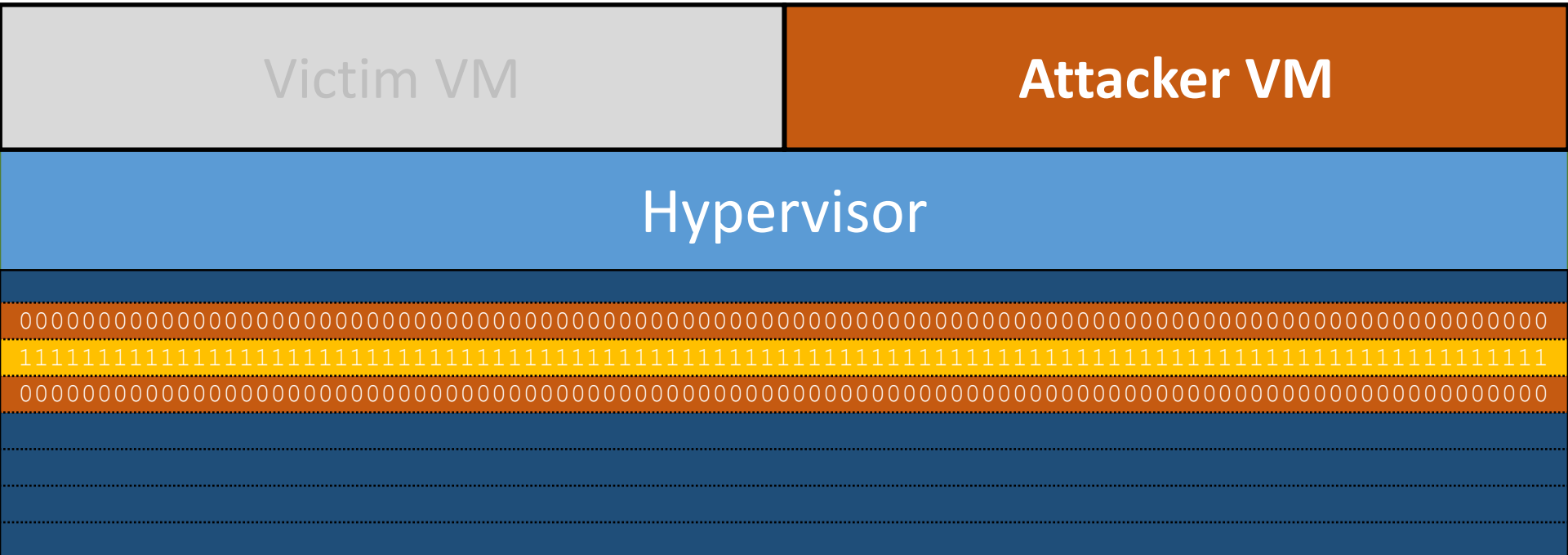
- Attacker has access to the **same** physical memory as the victim
- Attacker is root on his own VM: use **cache flush** to Rowhammer
- Use **huge pages** to select aggressor rows



Memory templating

Victim and Attacker share hardware

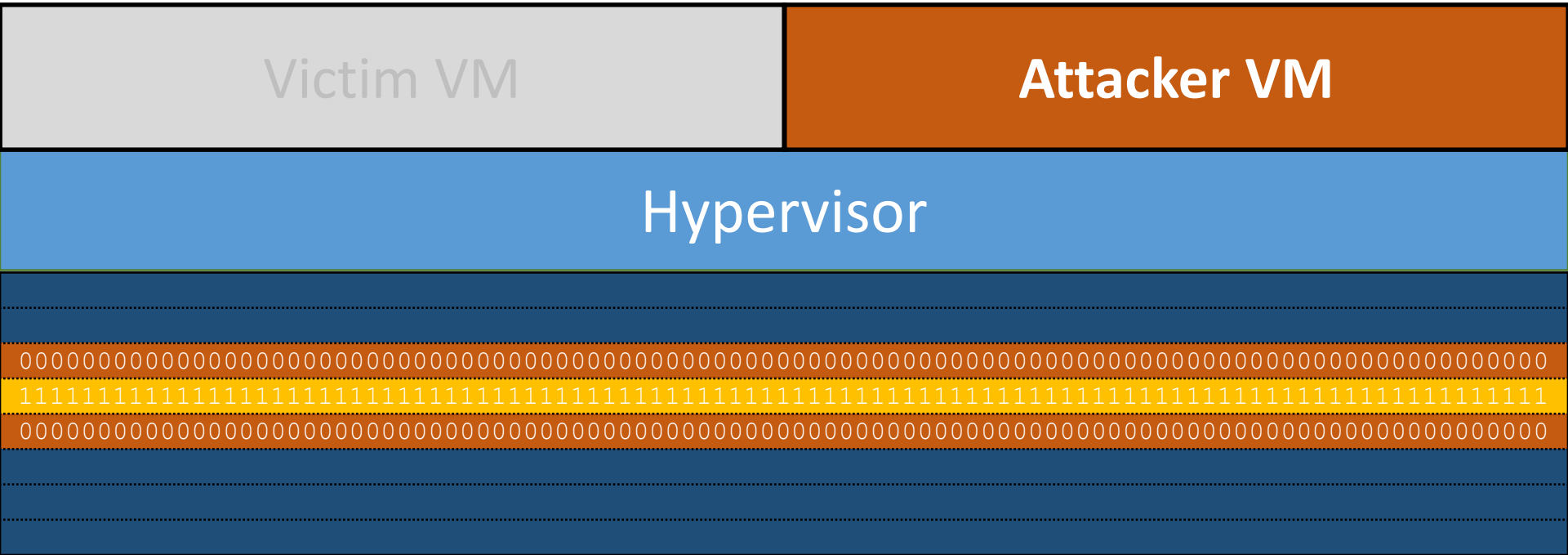
- Attacker has access to the **same** physical memory as the victim
- Attacker is root on his own VM: use **cache flush** to Rowhammer
- Use **huge pages** to select aggressor rows



Memory templating

Victim and Attacker share hardware

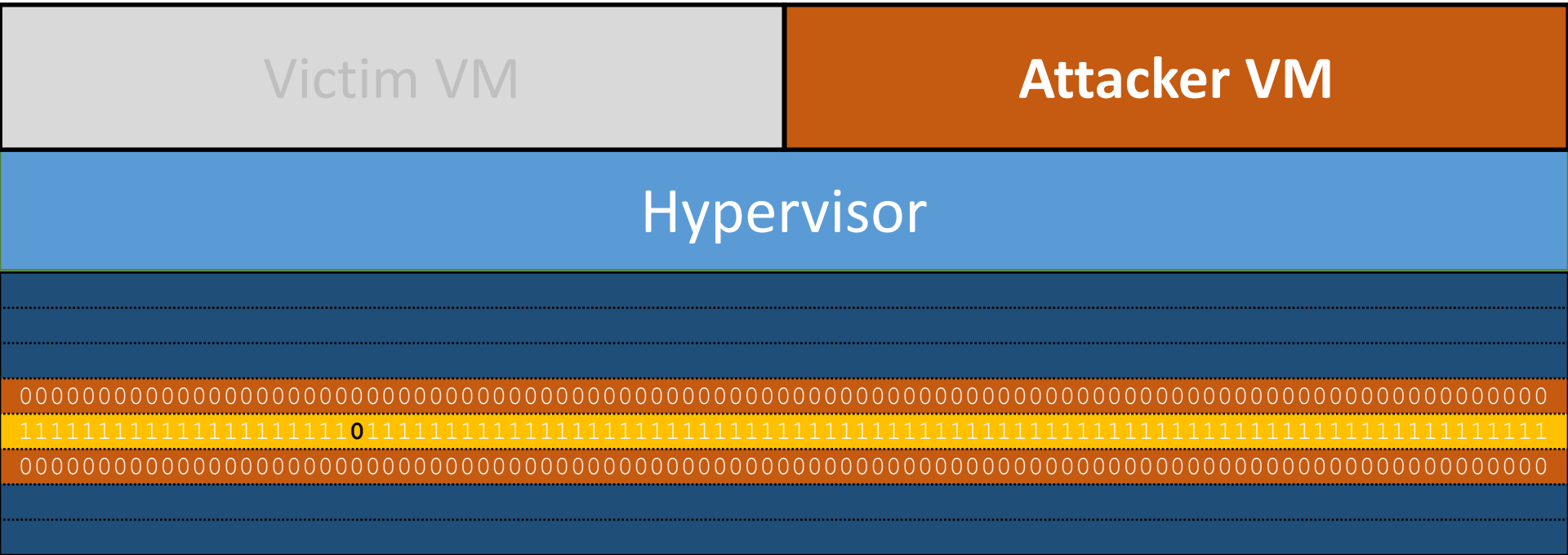
- Attacker has access to the **same** physical memory as the victim
- Attacker is root on his own VM: use **cache flush** to Rowhammer
- Use **huge pages** to select aggressor rows



Memory templating

Victim and Attacker share hardware

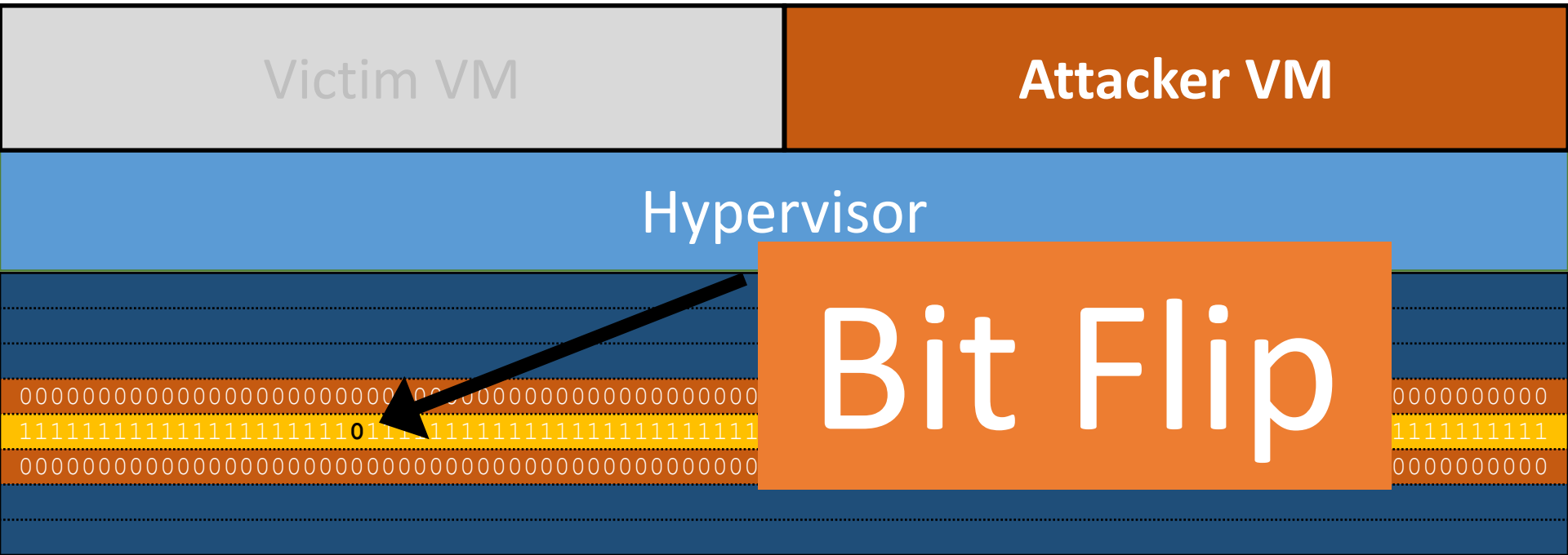
- Attacker has access to the **same** physical memory as the victim
- Attacker is root on his own VM: use **cache flush** to Rowhammer
- Use **huge pages** to select aggressor rows



Memory templating

Victim and Attacker share hardware

- Attacker has access to the **same** physical memory as the victim
- Attacker is root on his own VM: use **cache flush** to Rowhammer
- Use **huge pages** to select aggressor rows

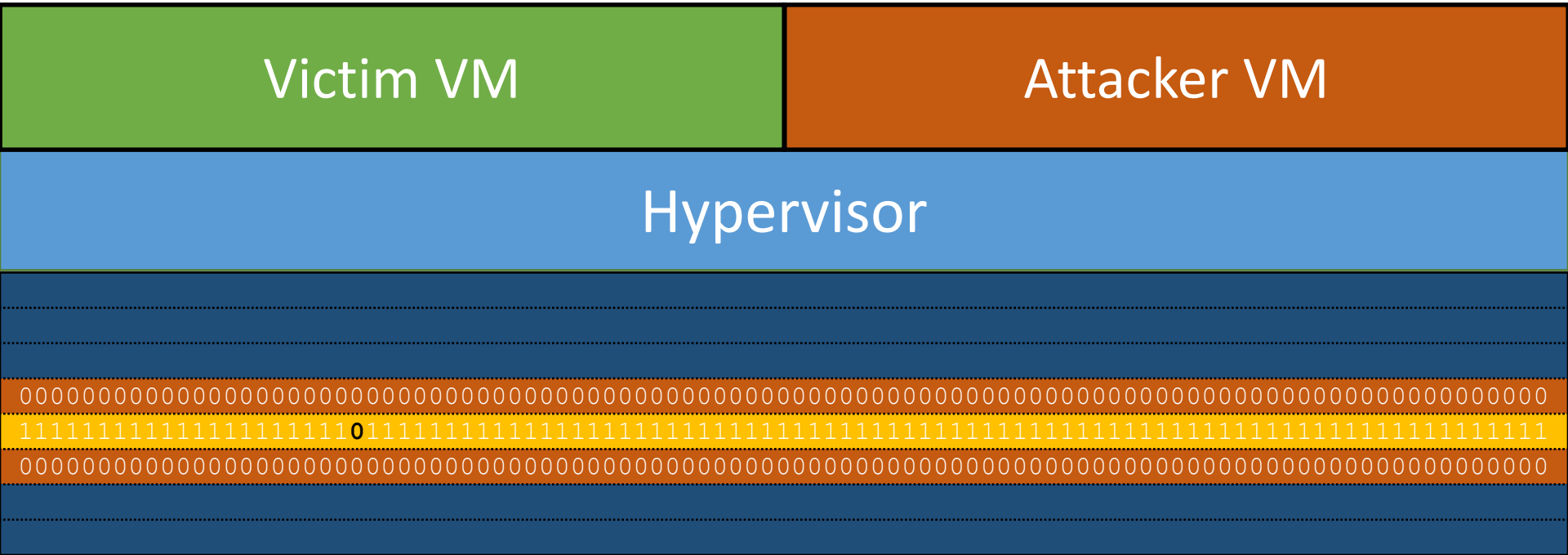


Land sensitive data

Given a bit flip in the Attacker VM

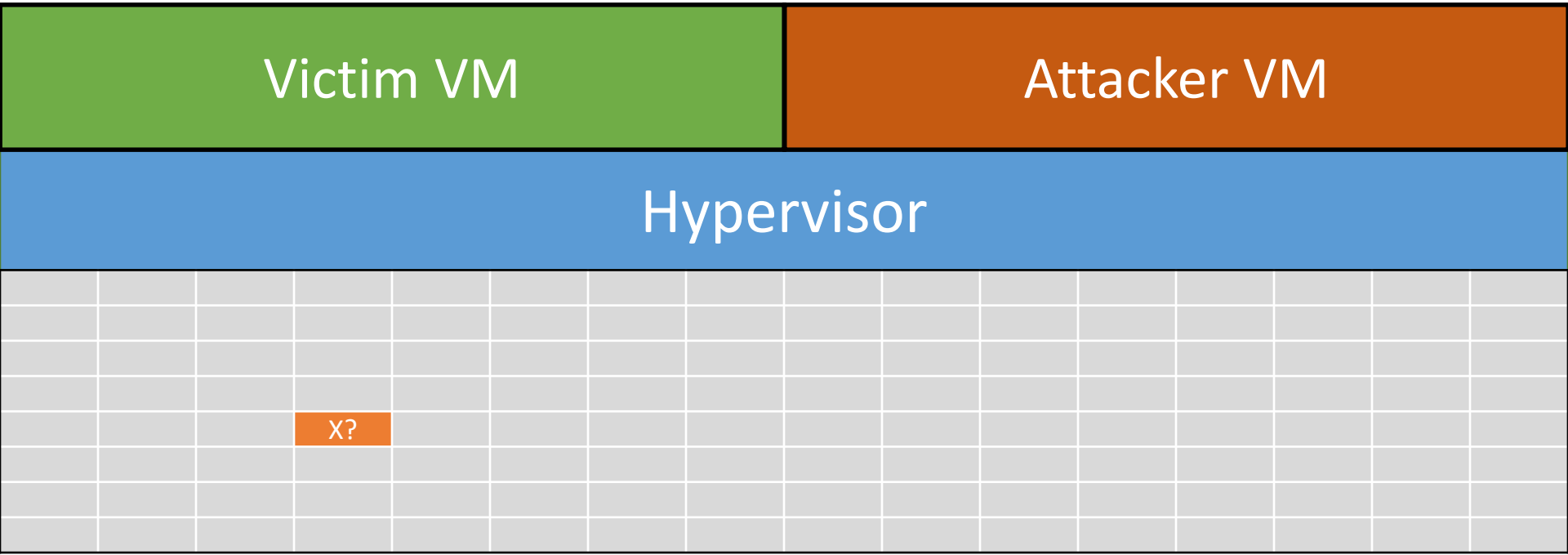
How do we force the Victim VM to store sensitive data here?

Memory Deduplication



Land sensitive data

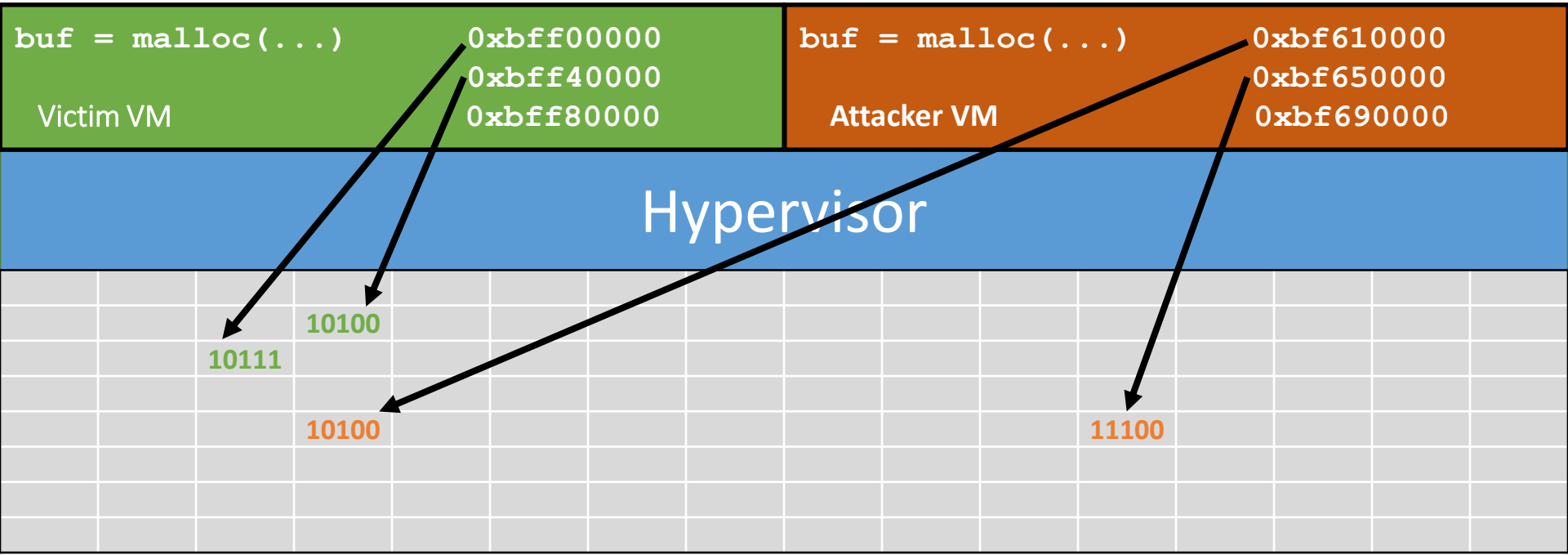
Memory deduplication



Land sensitive data

Memory deduplication

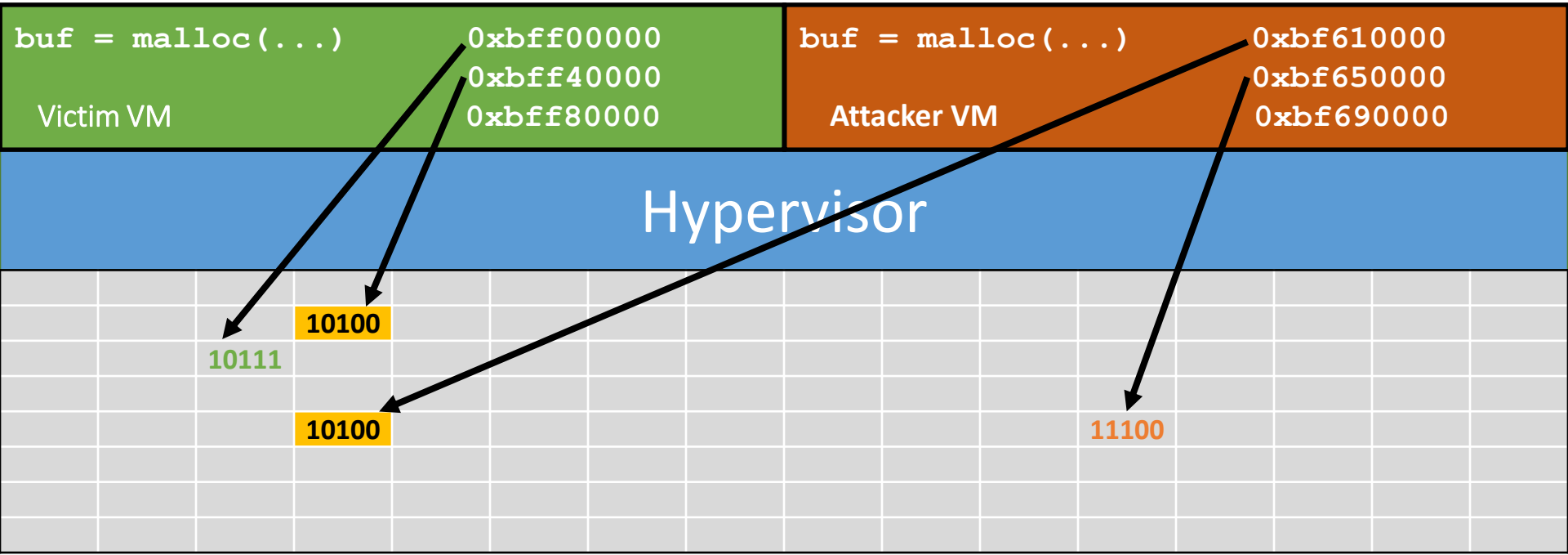
- Operating System / Hypervisor searches for *duplicate* pages



Land sensitive data

Memory deduplication

- Operating System / Hypervisor searches for *duplicate* pages

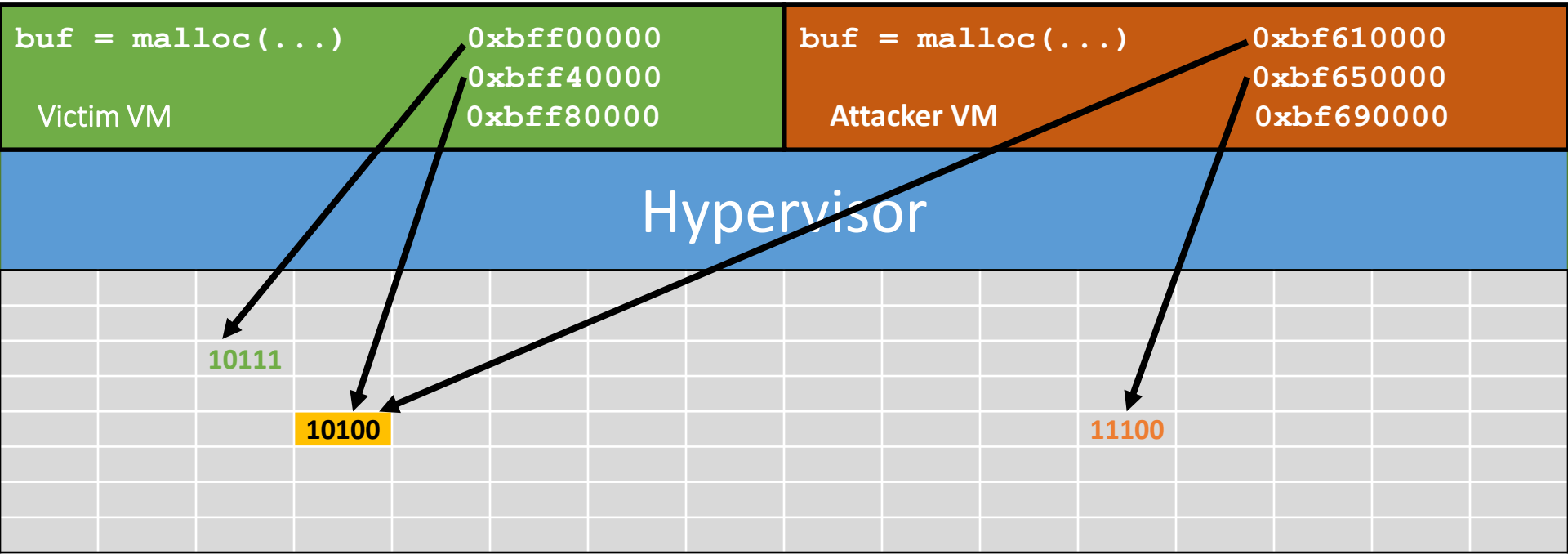


Land sensitive data

Memory deduplication

- Operating System / Hypervisor searches for *duplicate* pages
- Frees the duplicated bytes from physical memory
- Updates the virtual address mapping

But **what** should we flip?



Land sensitive data

What should we land?

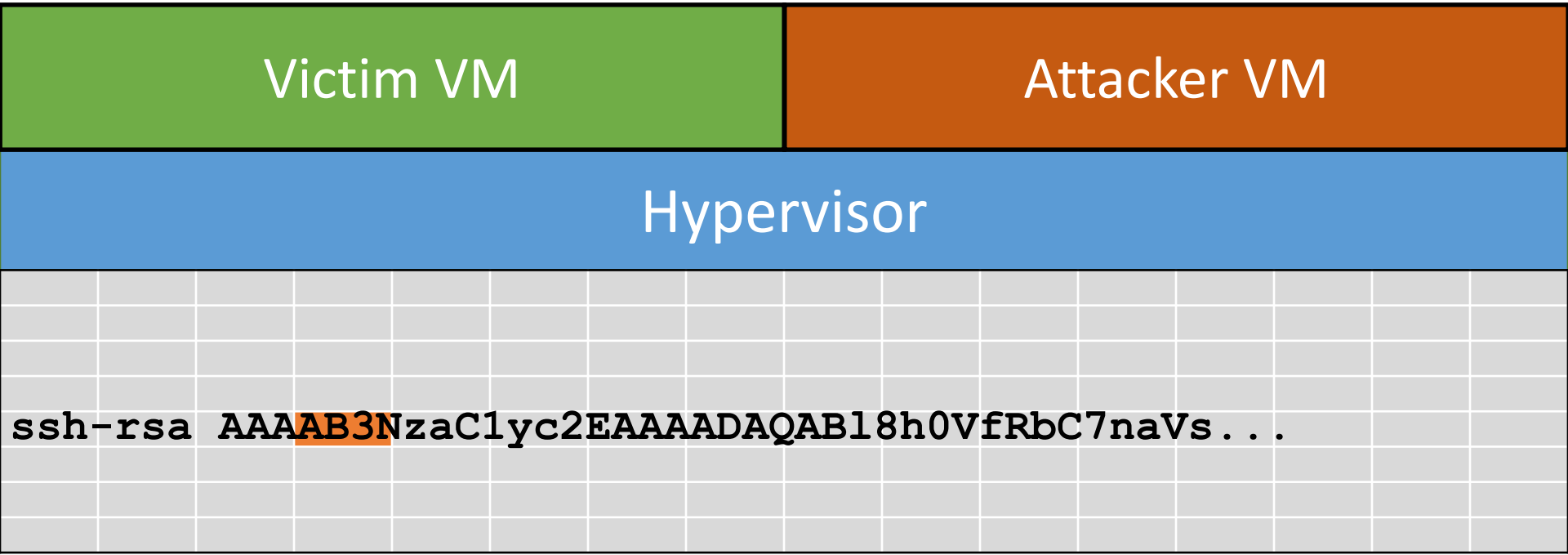
Cryptographic keys

+ Domain names

Reproduce the bit flip

Cryptographic keys

Public RSA key from **`.ssh/authorized_keys`**

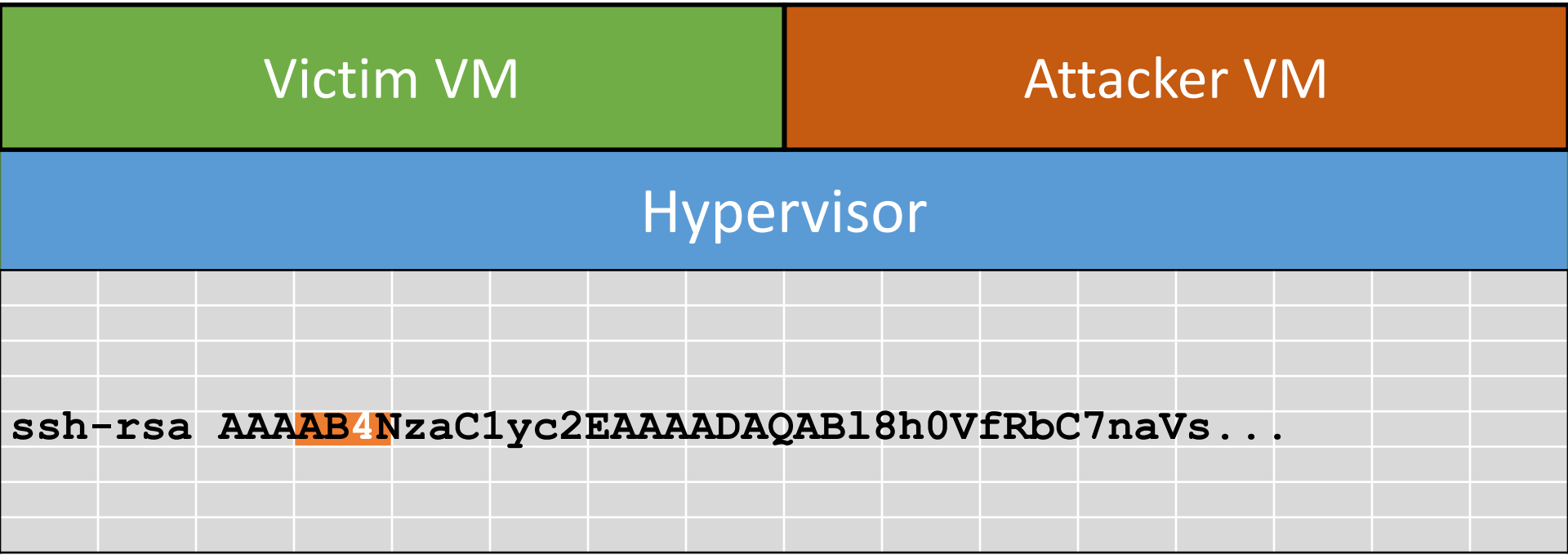


Reproduce the bit flip

Cryptographic keys

Public RSA key from `.ssh/authorized_keys`

- Flipping a bit changes the public/private key pair

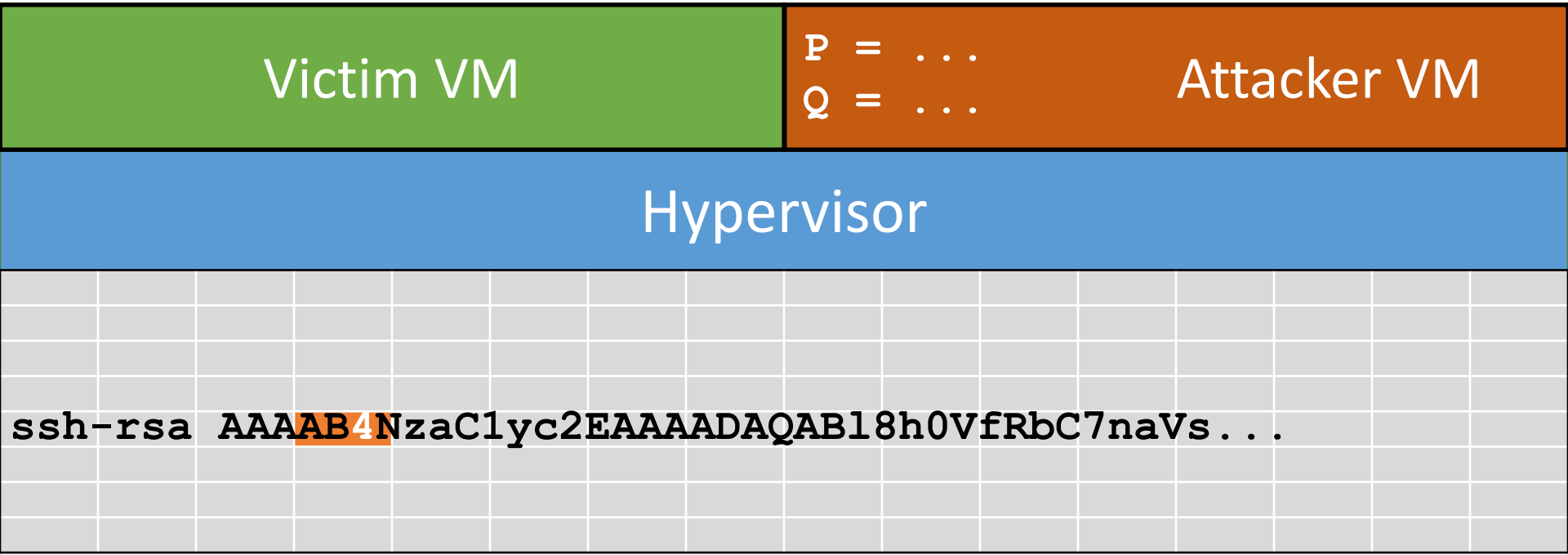


Reproduce the bit flip

Cryptographic keys

Public RSA key from `.ssh/authorized_keys`

- Flipping a bit changes the public/private key pair
- New public key is *easy* to factorize

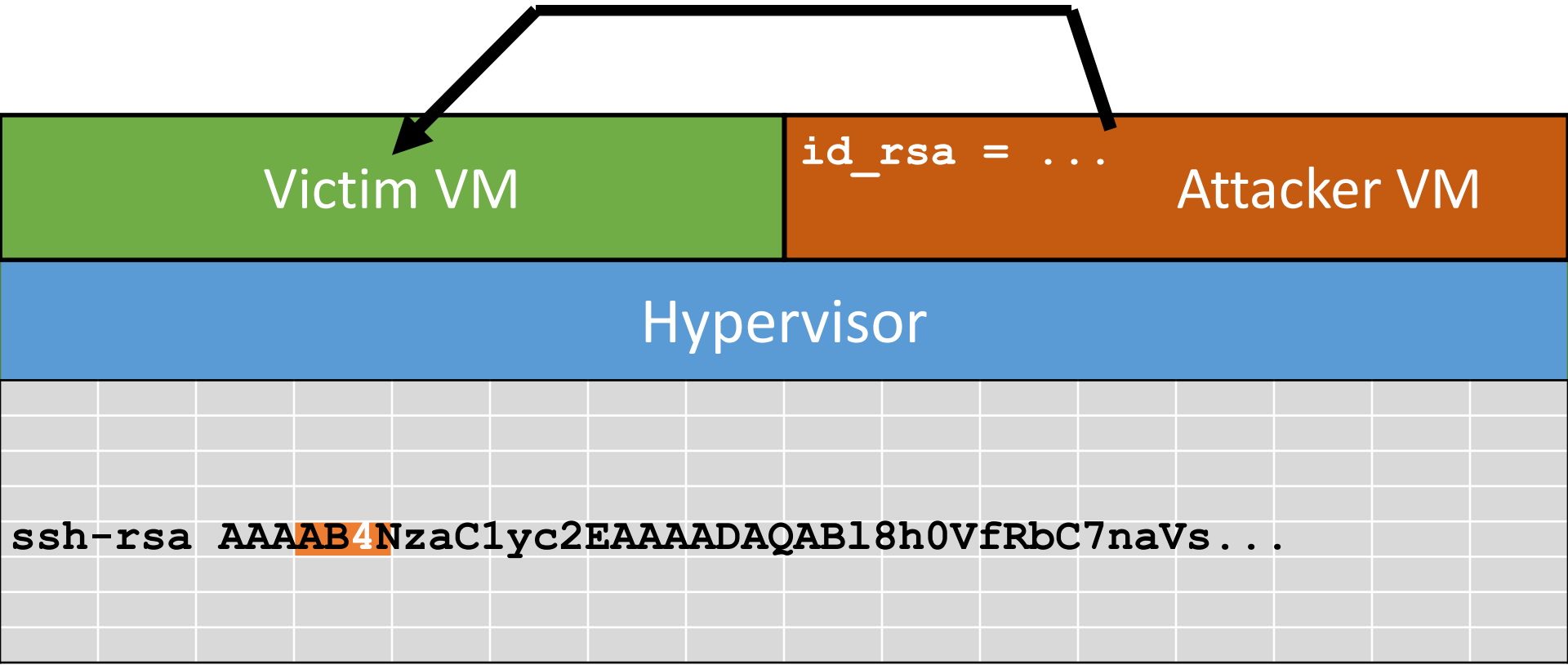


Reproduce the bit flip

Cryptographic keys

Public RSA key from `.ssh/authorized_keys`

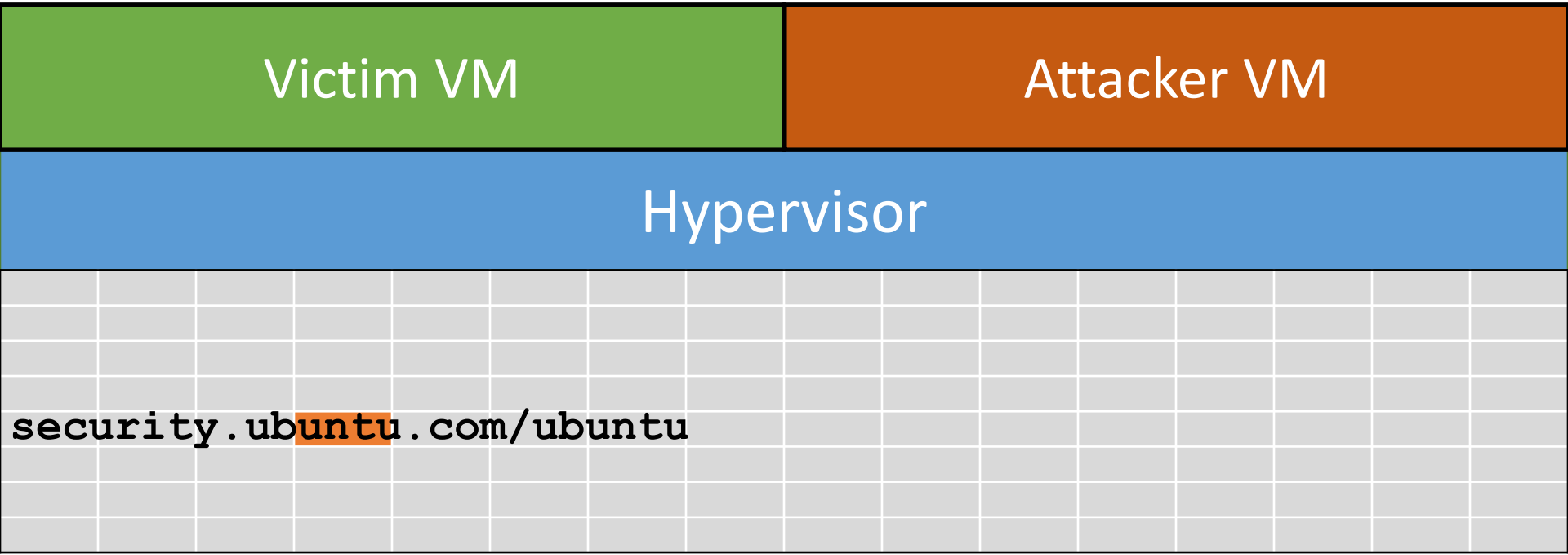
- Flipping a bit changes the public/private key pair
- New public key is *easy* to factorize
- Attacker computes the new private key and gets SSH access



Reproduce the bit flip

Cryptographic keys + domain names

Domain name used for **`apt-get upgrade`**

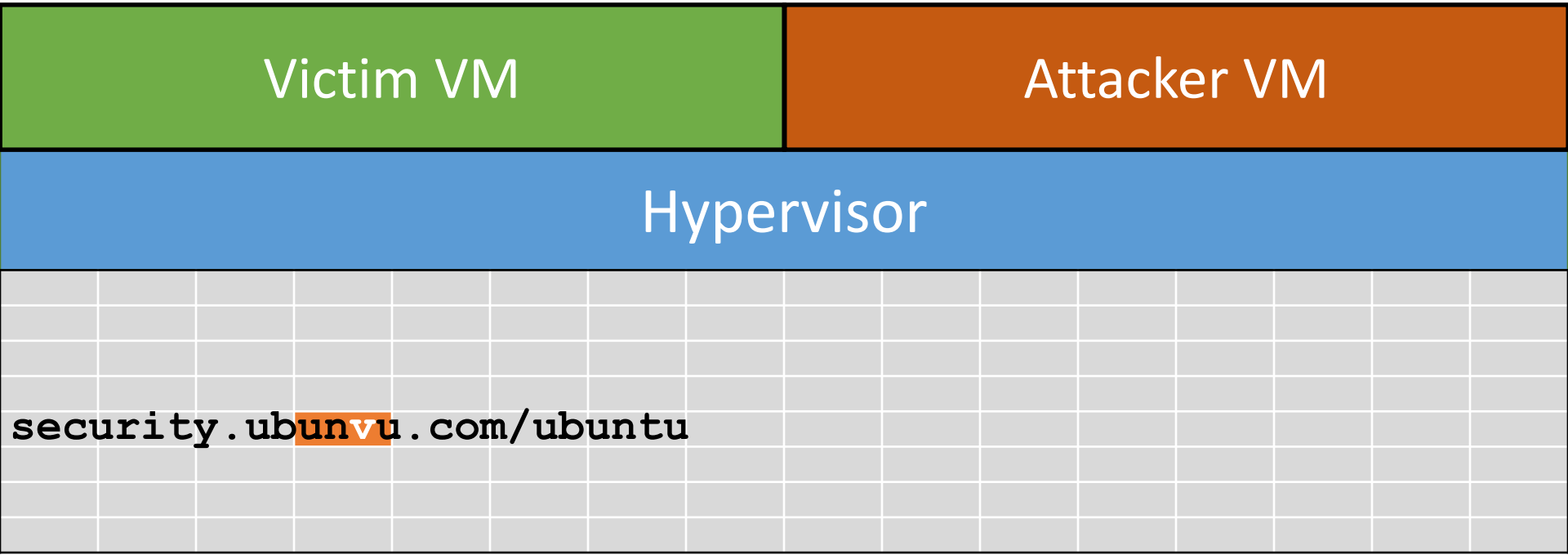


Reproduce the bit flip

Cryptographic keys + domain names

Domain name used for **`apt-get upgrade`**

- Flipping a bit changes the domain name used to pull updates
- Attacker hosts malicious **`ls`** command at **`ubunvu.com`**

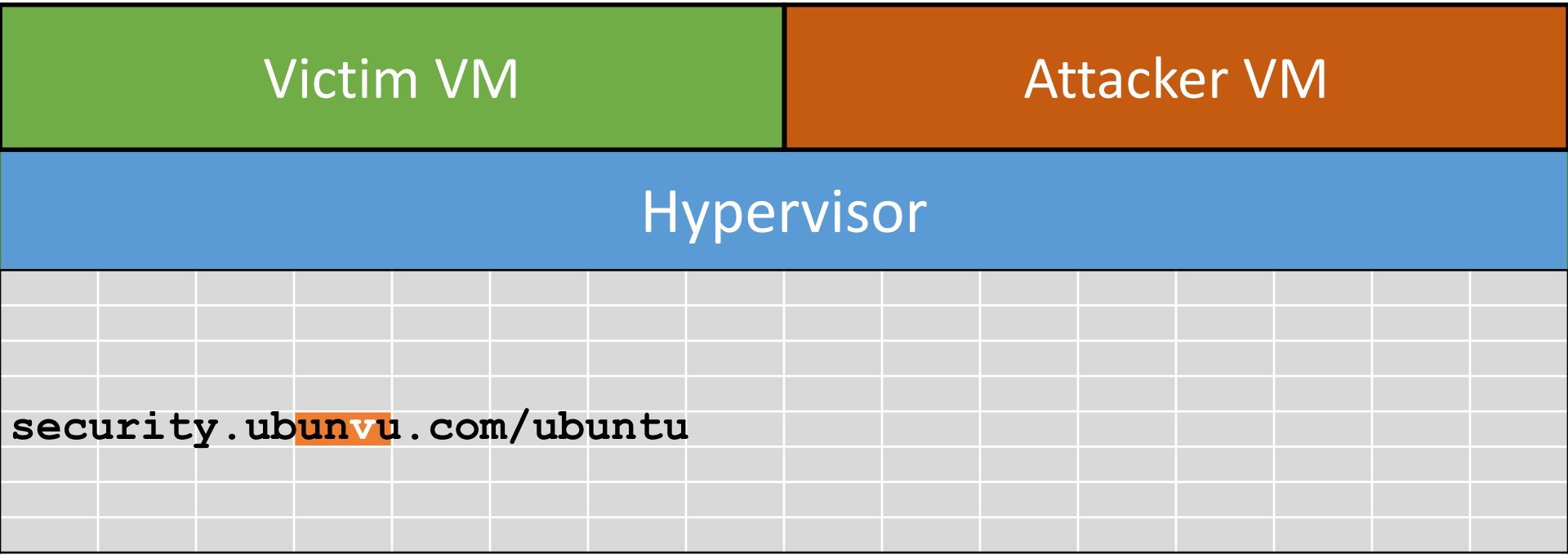


3. Reproduce the bit flip

Cryptographic keys + domain names

Domain name used for **`apt-get upgrade`**

- Flipping a bit changes the domain name used to pull updates
- Attacker hosts malicious **`ls`** command at **`ubunvu.com`**
- Packages are signed: also flip a bit in the GPG keychain

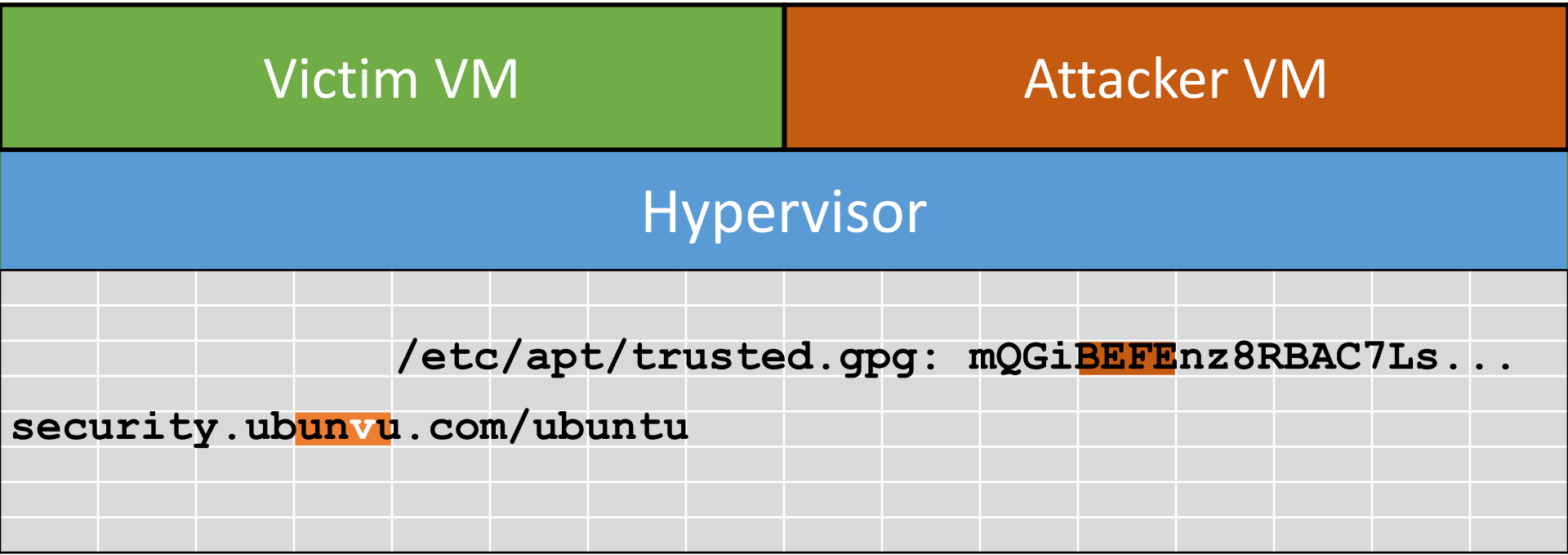


Reproduce the bit flip

Cryptographic keys + domain names

Domain name used for **`apt-get upgrade`**

- Flipping a bit changes the domain name used to pull updates
- Attacker hosts malicious **`ls`** command at **`ubunvu.com`**
- Packages are signed: also flip a bit in the GPG keychain

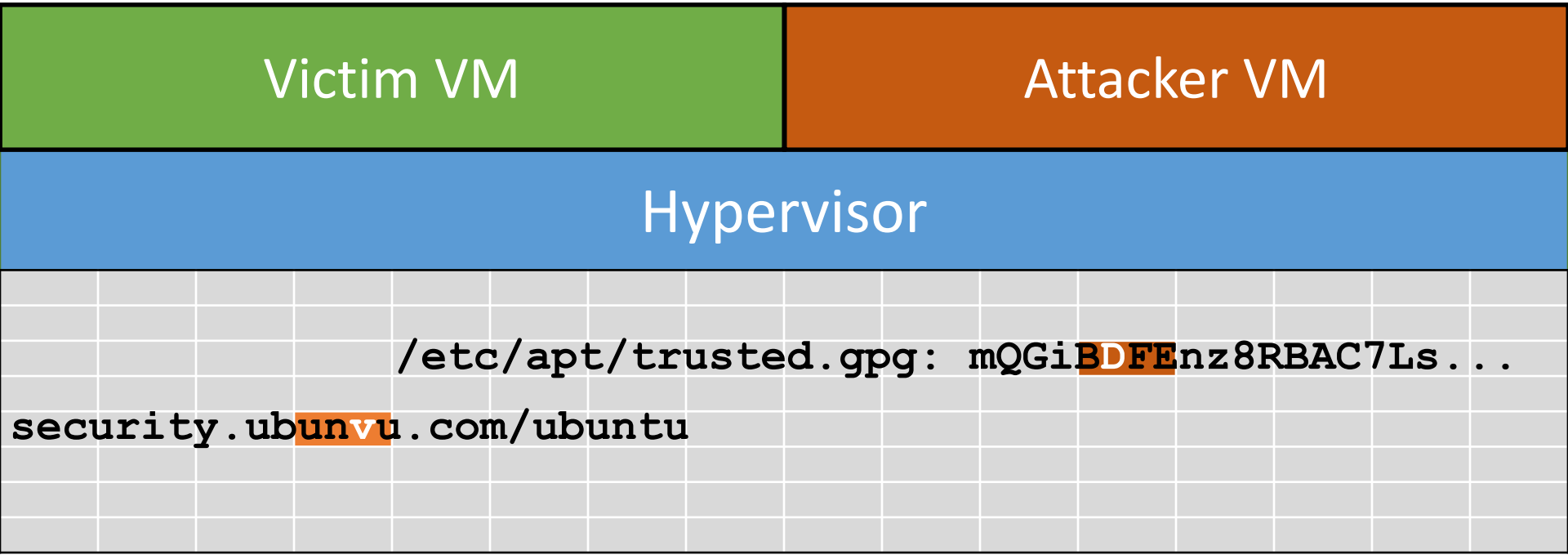


Reproduce the bit flip

Cryptographic keys + domain names

Domain name used for **`apt-get upgrade`**

- Flipping a bit changes the domain name used to pull updates
- Attacker hosts malicious **`ls`** command at **`ubunvu.com`**
- Packages are signed: also flip a bit in the GPG keychain

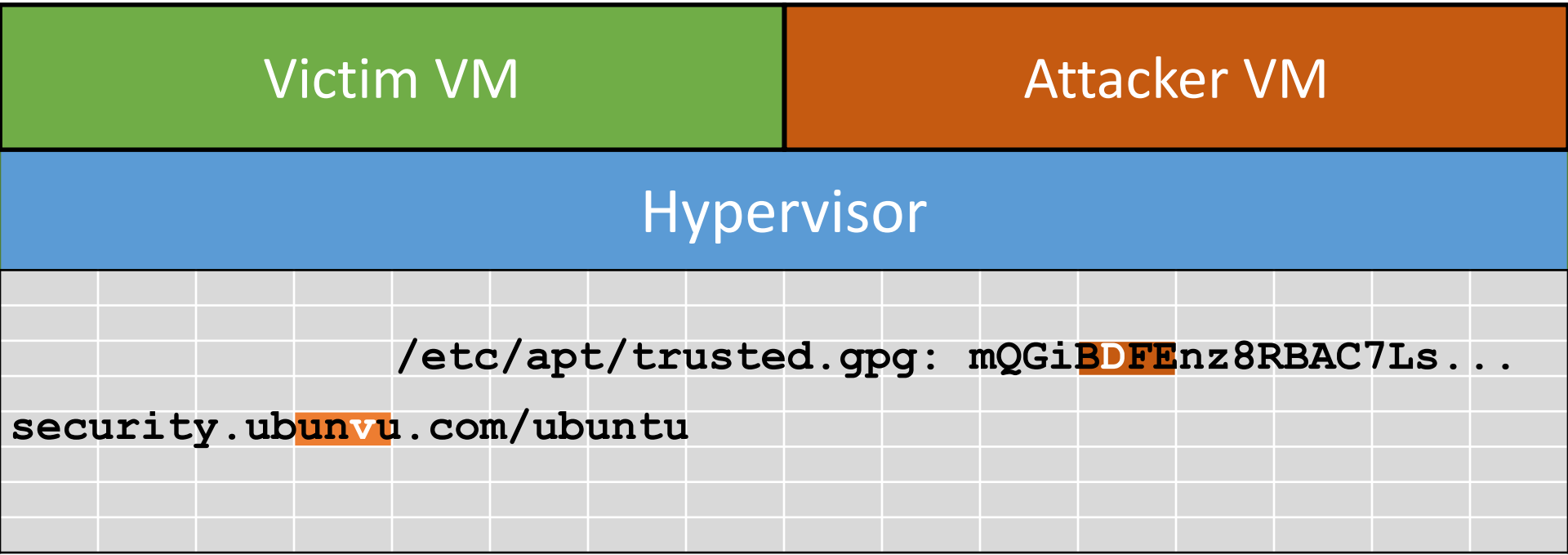


Reproduce the bit flip

Cryptographic keys + domain names

Domain name used for **`apt-get upgrade`**

- Flipping a bit changes the domain name used to pull updates
- Attacker hosts malicious **`ls`** command at **`ubunvu.com`**
- Packages are signed: also flip a bit in the GPG keychain
- Attacker computes the new GPG key and signs his backdoored **`ls`**

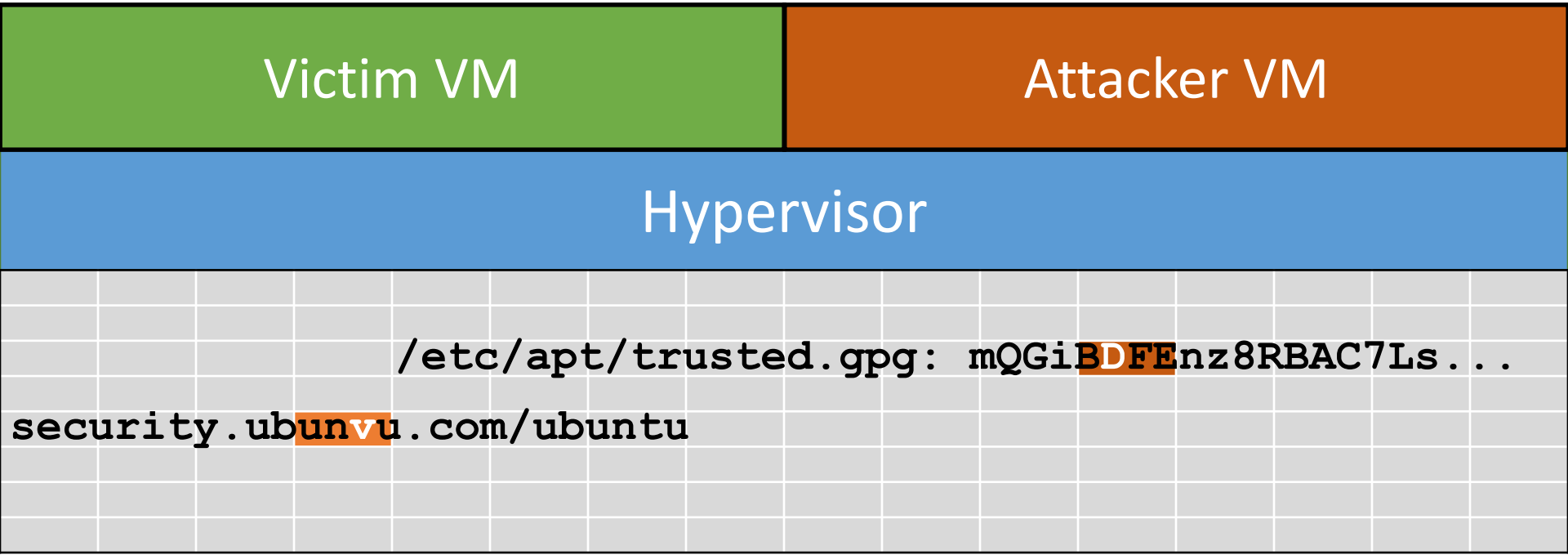


Reproduce the bit flip

Cryptographic keys + domain names

Domain name used for **`apt-get upgrade`**

- Flipping a bit changes the domain name used to pull updates
- Attacker hosts malicious **`ls`** command at **`ubunvu.com`**
- Packages are signed: also flip a bit in the GPG keychain
- Attacker computes the new GPG key and signs his backdoored **`ls`**
- Wait for victim to apt-get upgrade



Disclosure

Contacted NCSC on June 9, 2016

- Good job on contacting vendors and affected parties!
- GPG Patch so that self-signed keys are no longer valid

Bought some domains for ubuntu/debian

- ***ubunvu.com***
- ***ubuftu.com***
- ***ubunte.com***
- ***ubuntt.com***
- ***ubuntw.com***
- ***ubun4u.com***
- ***ubunuu.com***
- ***dabian.org***
- ***ddbian.org***
- ***debiaf.org***
- ***debicn.org***
- ***debial.org***
- ***debiqn.org***
- ***debien.org***

And more…

Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovani Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida

DRAMMER

ACM CCS
October 2016

Drammer

- Can we use Rowhammer to root Android phones?
- Targeting ARM instead of x86
- Android devices do not have memory deduplication
- Exploit behavior of underlying memory allocator

Memory Templating

Challenges (recap for x86)

1. Bypass the CPU cache (hard requirement)
   - Cache eviction
   - Explicit cache flush (clflush instruction)
2. Select the aggressor rows (soft requirement)
   - **`/proc/self/pagemap`**
   - Huge pages

Does this work on ARM?

Memory Templating

Challenges (recap for x86)

1. Bypass the CPU cache (hard requirement)
   - Cache eviction — Too slow
   - Explicit cache flush (clflush instr — Privileged
2. Select the aggressor rows (~~soft~~hard requirement)
   - `/proc/self/pagemap` — Privileged
   - Huge pages — Not Available

Does this work on ARM?

Of course not

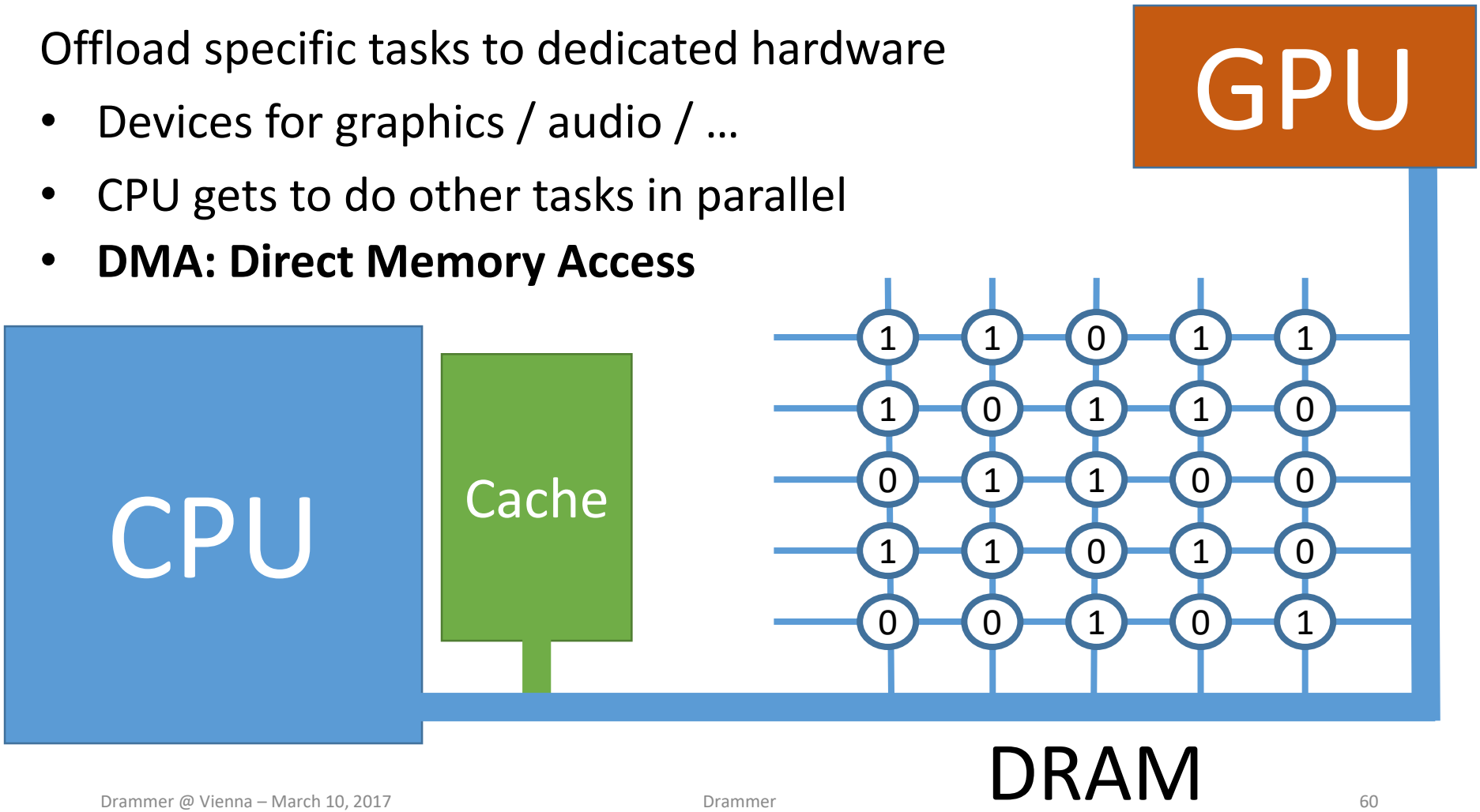
Memory templating

Bypass the CPU cache

Sometimes CPU caches are detrimental

Offload specific tasks to dedicated hardware

- Devices for graphics / audio / …
- CPU gets to do other tasks in parallel
- **DMA: Direct Memory Access**

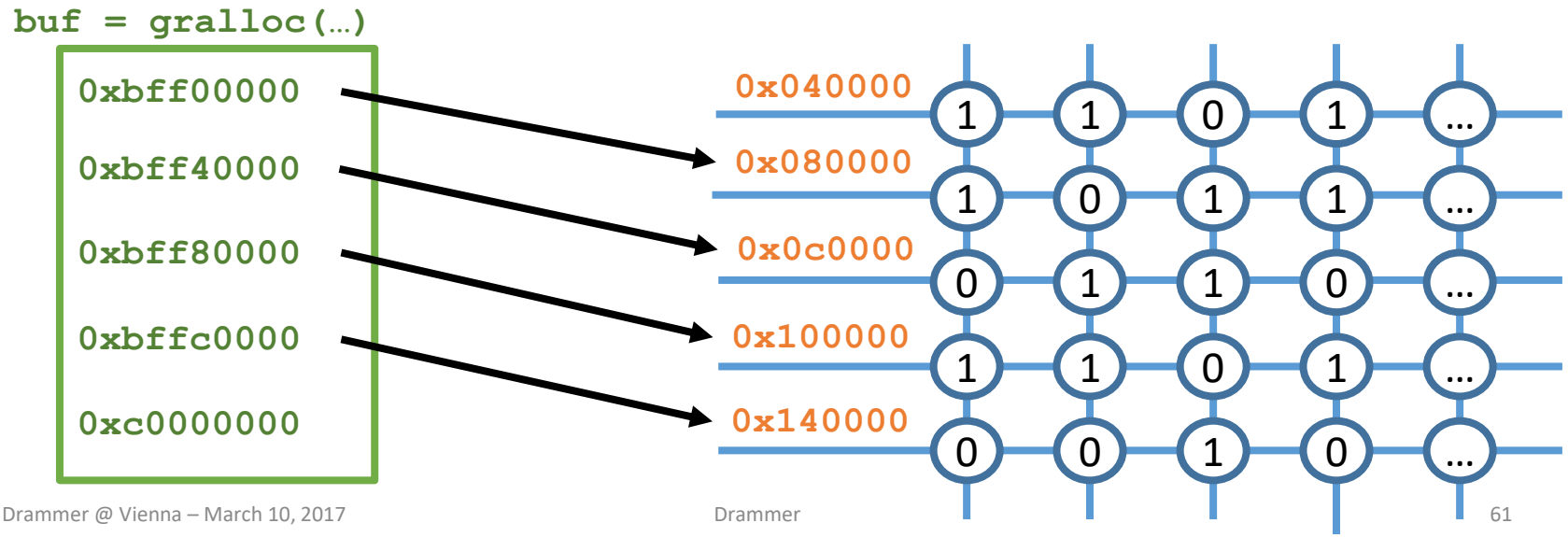


Memory templating
Select the aggressor rows

Dedicated hardware devices might be *stupid*

- Simple devices cannot translate virtual to physical mappings
- Shared memory must be **contiguous**

DMA can behave very much like **Huge Pages**



Land sensitive data

Threat-Model

- Recent Android OS without memory deduplication (too costly)
- Unprivileged app wants to get root access

Goal: read/write access to anywhere in memory

- Find administrative kernel data structures for our process
- Overwrite our own uid to get root access

Approach

1. Land a Page Table at the vulnerable location
2. Flip a bit in a Page Table Entry!

ATTACK PRINCIPLES

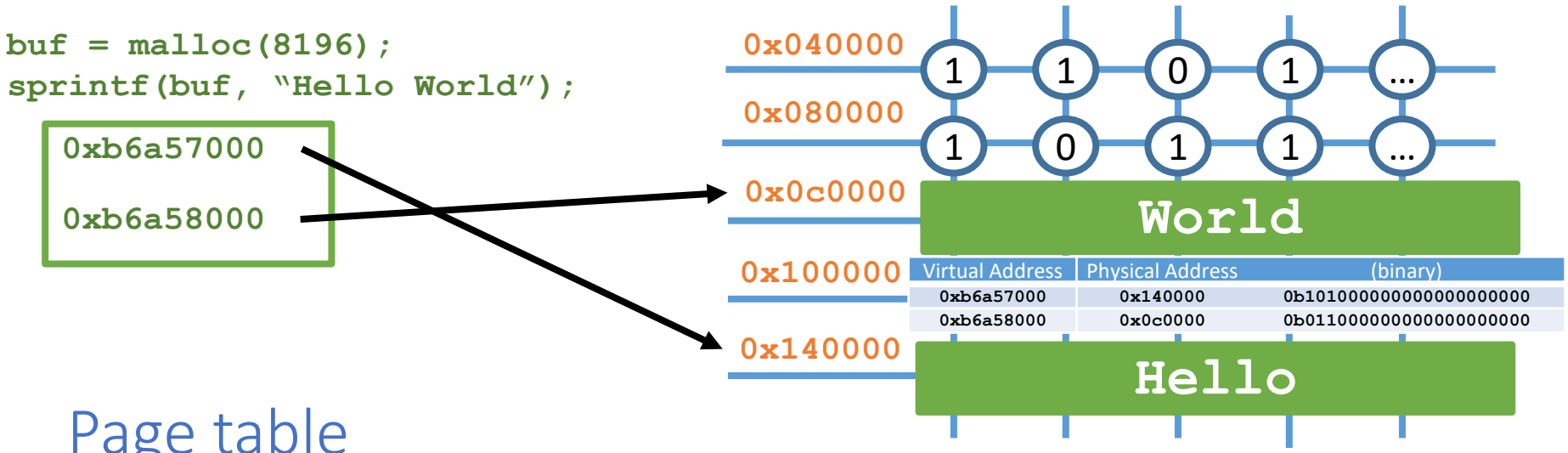
An Intuition

Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

```
buf = malloc(8196);
sprintf(buf, "Hello World");
```



Page table

Virtual Address	Physical Address	(binary)
0xb6a57000	0x140000	0b10100000000000000000
0xb6a58000	0x0c0000	0b01100000000000000000

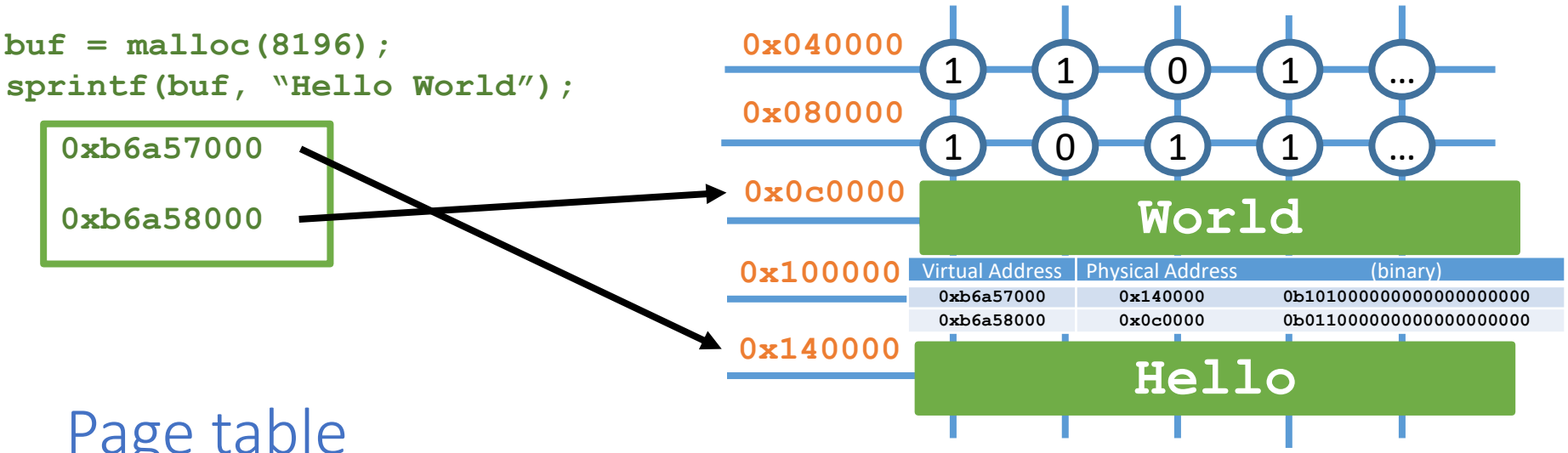
Where are page tables stored? In memory!

Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

```
buf = malloc(8196);
sprintf(buf, "Hello World");
```



Page table

Virtual Address	Physical Address	(binary)
0xb6a57000	0x140000	0b10100000000000000000
0xb6a58000	0x0c0000	0b01100000000000000000

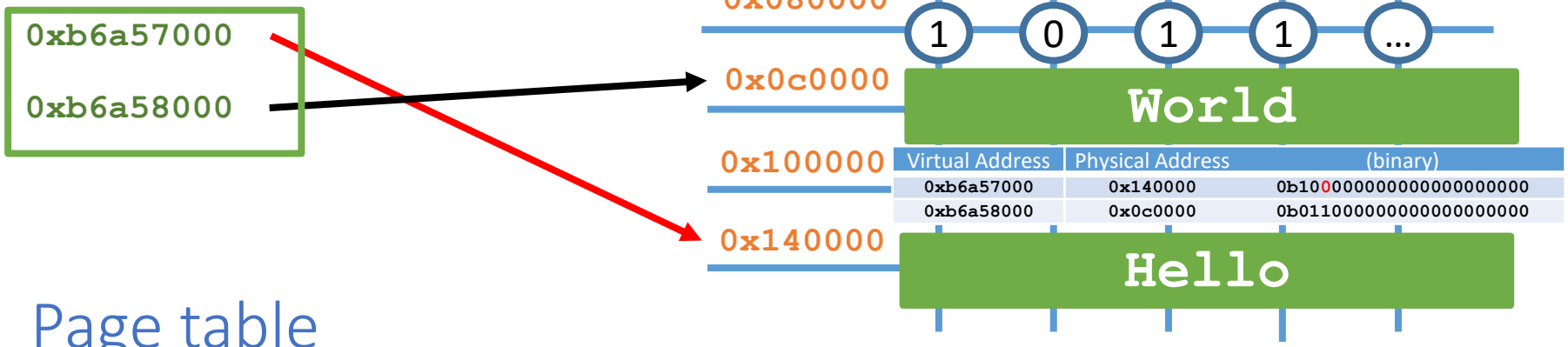
What if we flip a bit in a page table? Modify mappings!

Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

```
buf = malloc(8196);
sprintf(buf, "Hello World");
```



Page table

Virtual Address	Physical Address	(binary)
0xb6a57000	0x140000	0b10000000000000000000
0xb6a58000	0x0c0000	0b01100000000000000000

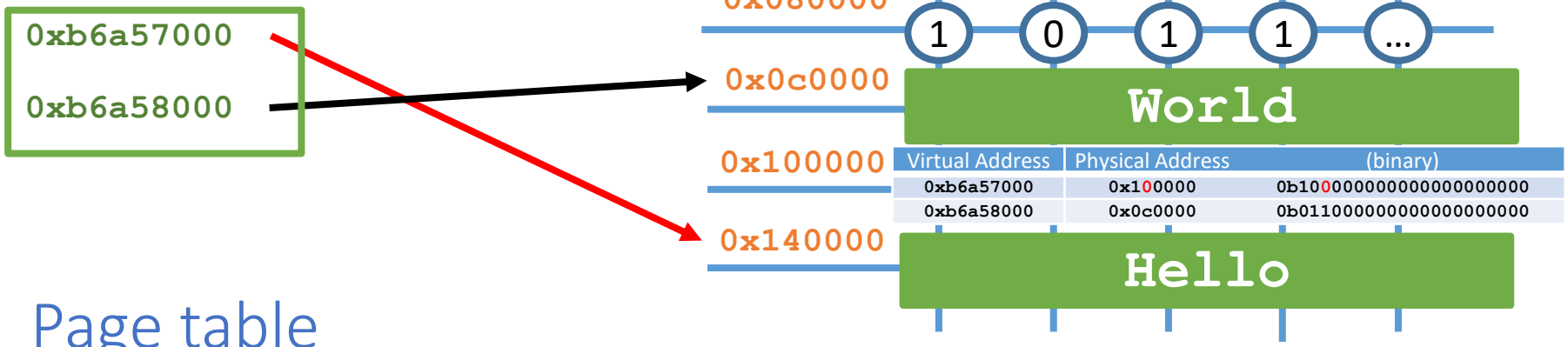
What if we flip a bit in a page table? Modify mappings!

Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

```
buf = malloc(8196);
sprintf(buf, "Hello World");
```



Page table

Virtual Address	Physical Address	(binary)
0xb6a57000	0x100000	0b10000000000000000000
0xb6a58000	0x0c0000	0b01100000000000000000

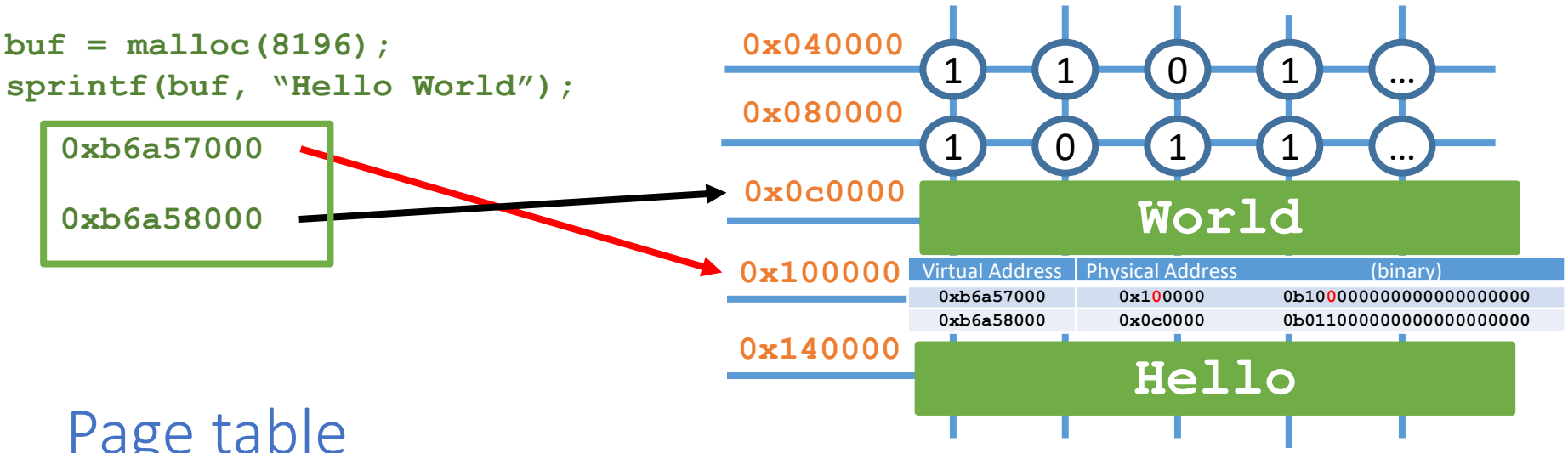
What if we flip a bit in a page table? Modify mappings!

Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

```
buf = malloc(8196);
sprintf(buf, "Hello World");
```



Page table

Virtual Address	Physical Address	(binary)
0xb6a57000	0x100000	0b10000000000000000000
0xb6a58000	0x0c0000	0b01100000000000000000

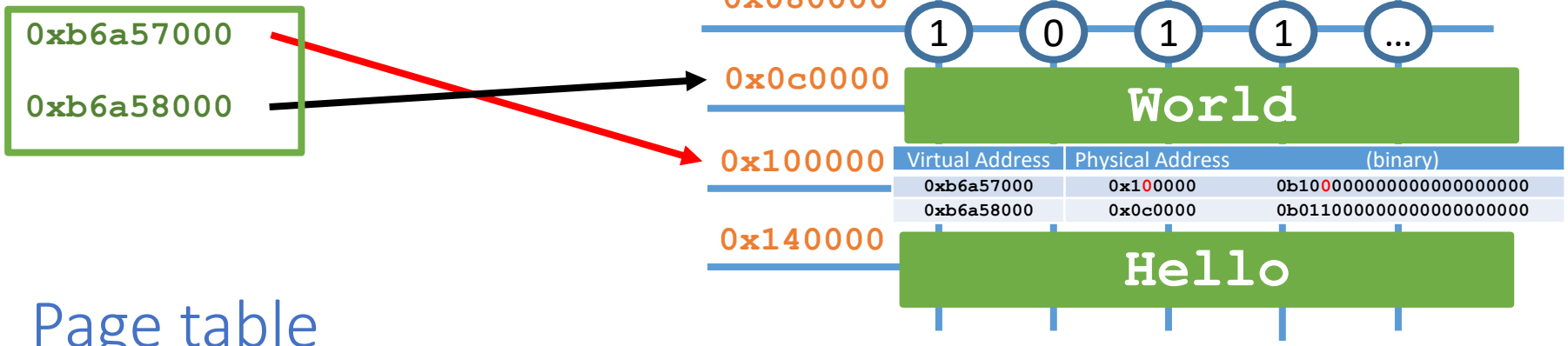
What if we flip a bit in a page table? Modify mappings!

Land sensitive data

Page tables

Mapping virtual addresses to physical addresses

```
buf = malloc(8196);
sprintf(buf, "Hello World");
```



Page table

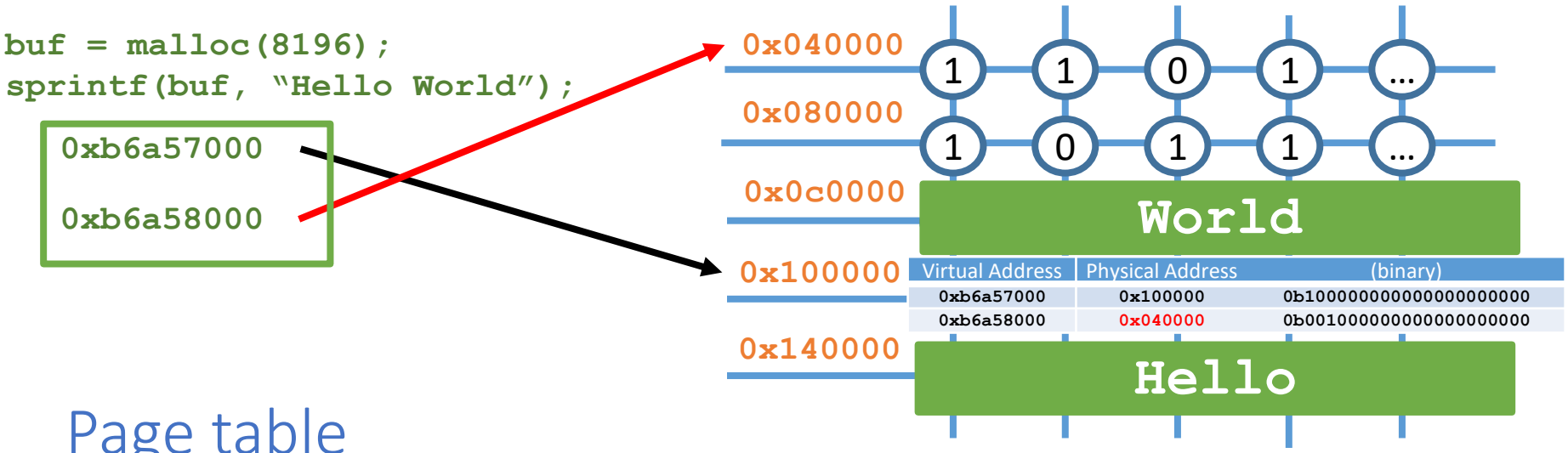
Virtual Address	Physical Address	(binary)
0xb6a57000	0x100000	0b10000000000000000000
0xb6a58000	0x0c0000	0b01100000000000000000

Page tables give us read/write access to anywhere in memory

Land sensitive data

Page tables

Mapping virtual addresses to physical addresses



Page table

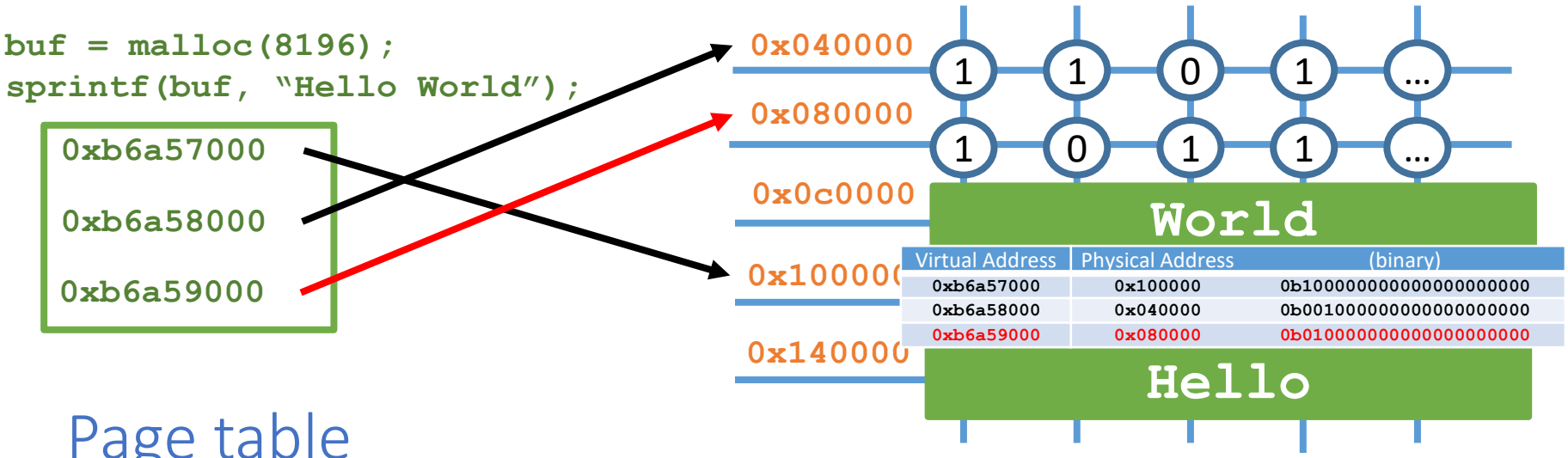
Virtual Address	Physical Address	(binary)
0xb6a57000	0x100000	0b10000000000000000000
0xb6a58000	0x040000	0b00100000000000000000

By modifying existing entries…

Land sensitive data

Page tables

Mapping virtual addresses to physical addresses



Page table

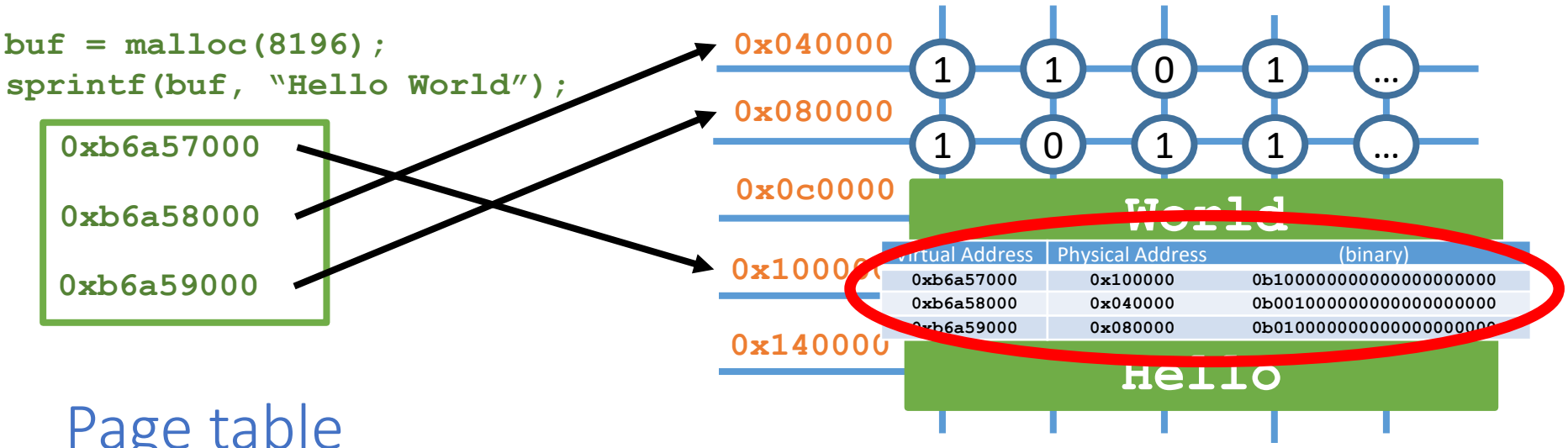
Virtual Address	Physical Address	(binary)
0xb6a57000	0x100000	0b10000000000000000000
0xb6a58000	0x040000	0b00100000000000000000
0xb6a59000	0x080000	0b01000000000000000000

By modifying existing entries… or by adding new ones

Land sensitive data

Page tables

Mapping virtual addresses to physical addresses



Page table

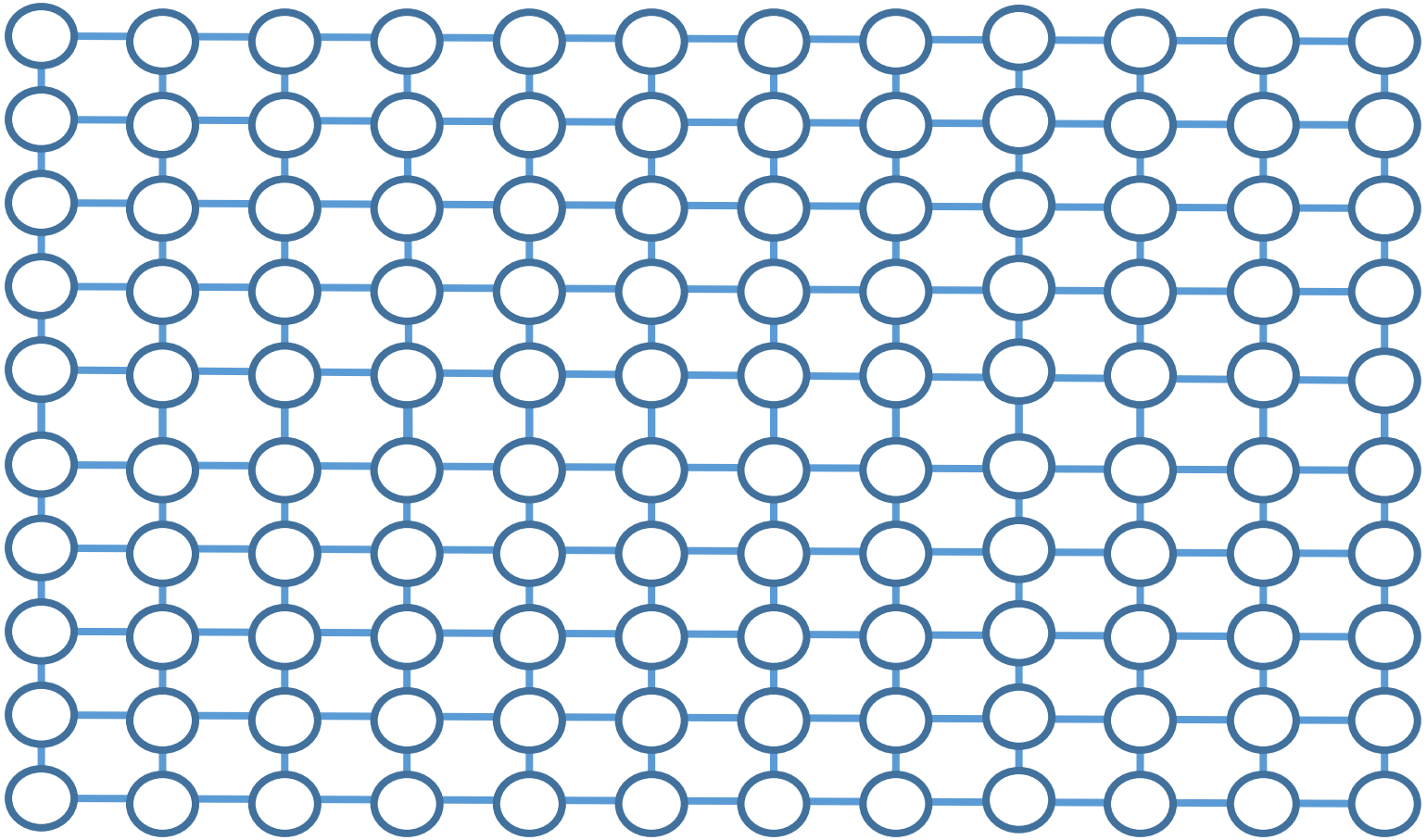
Virtual Address	Physical Address	(binary)
0xb6a57000	0x100000	0b10000000000000000000
0xb6a58000	0x040000	0b00100000000000000000
0xb6a59000	0x080000	0b01000000000000000000

By modifying existing entries… or by adding new ones

Land sensitive data

Phys Feng Shui (intuition)

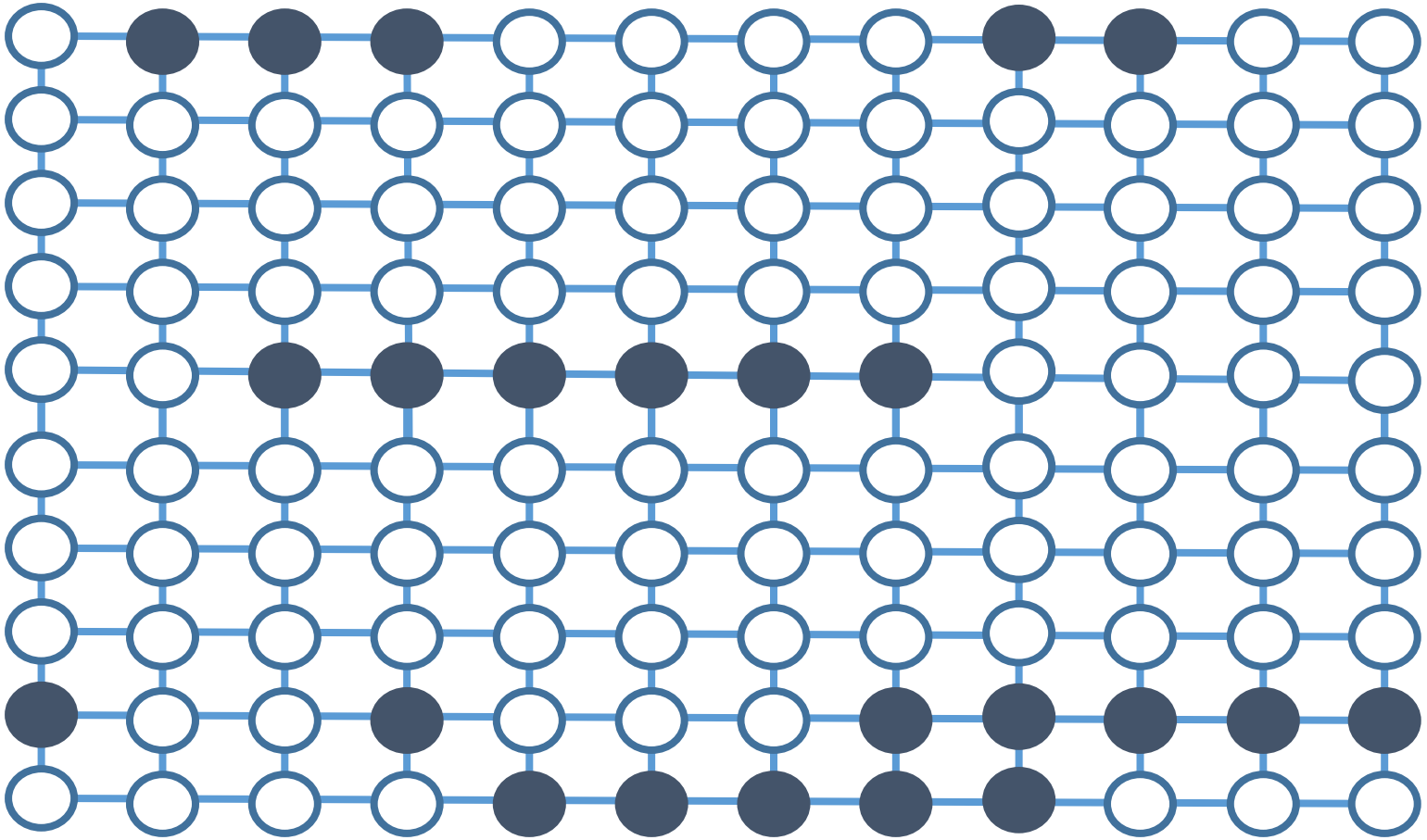
Physical memory



Land sensitive data

Phys Feng Shui (intuition)

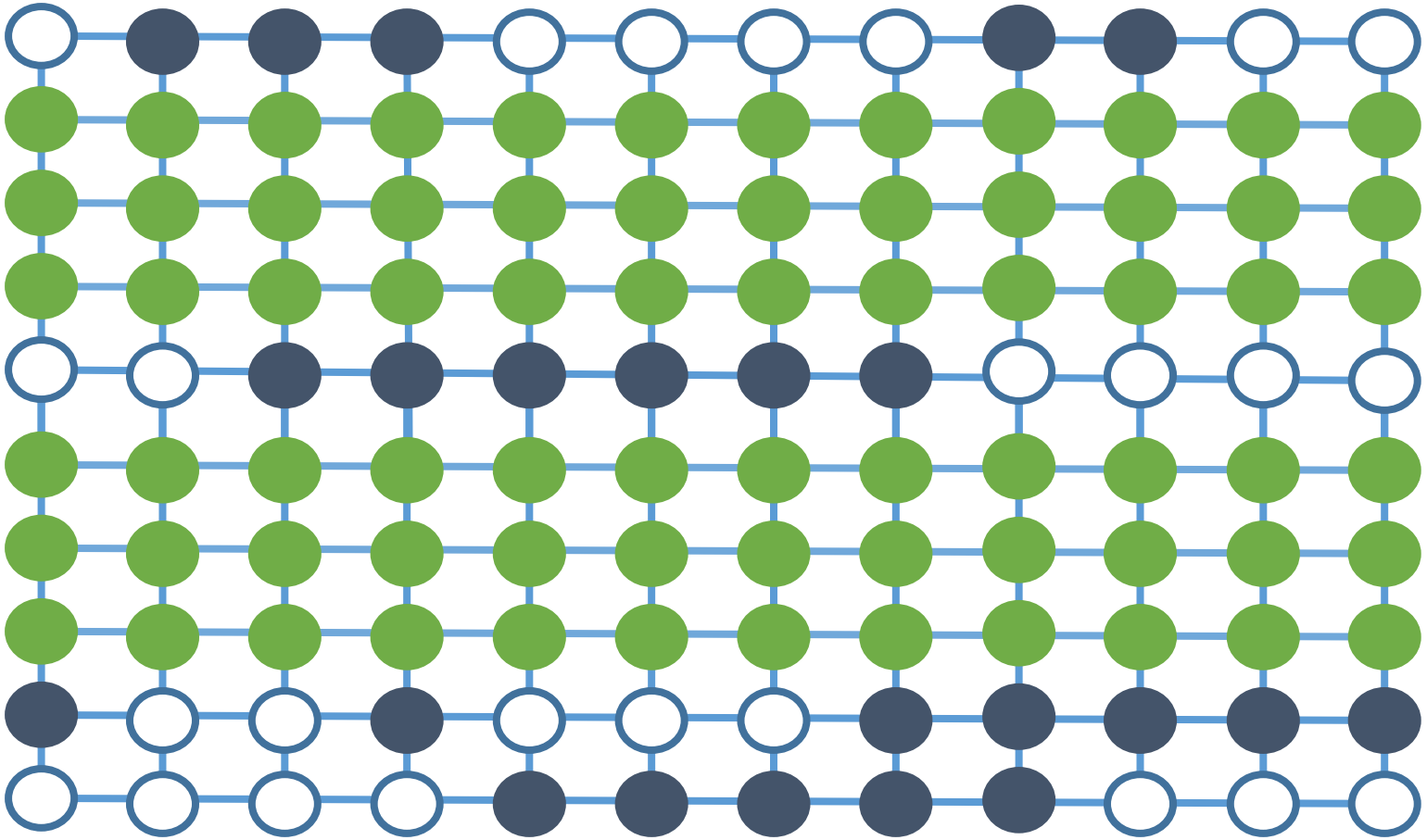
Running applications before Drammer



Land sensitive data

Phys Feng Shui (intuition)

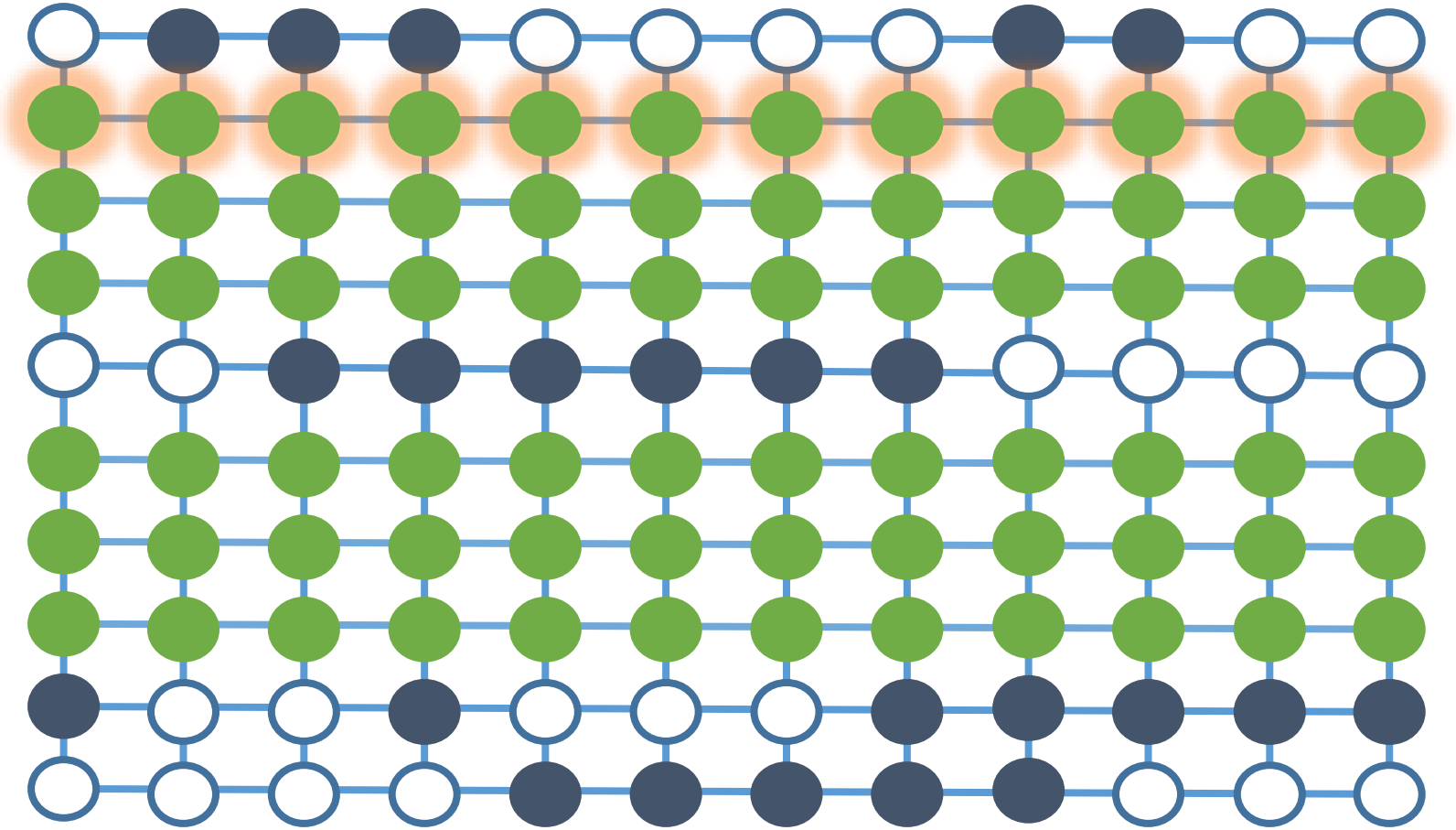
Step 1. Allocate large chunks of memory



Land sensitive data

Phys Feng Shui (intuition)

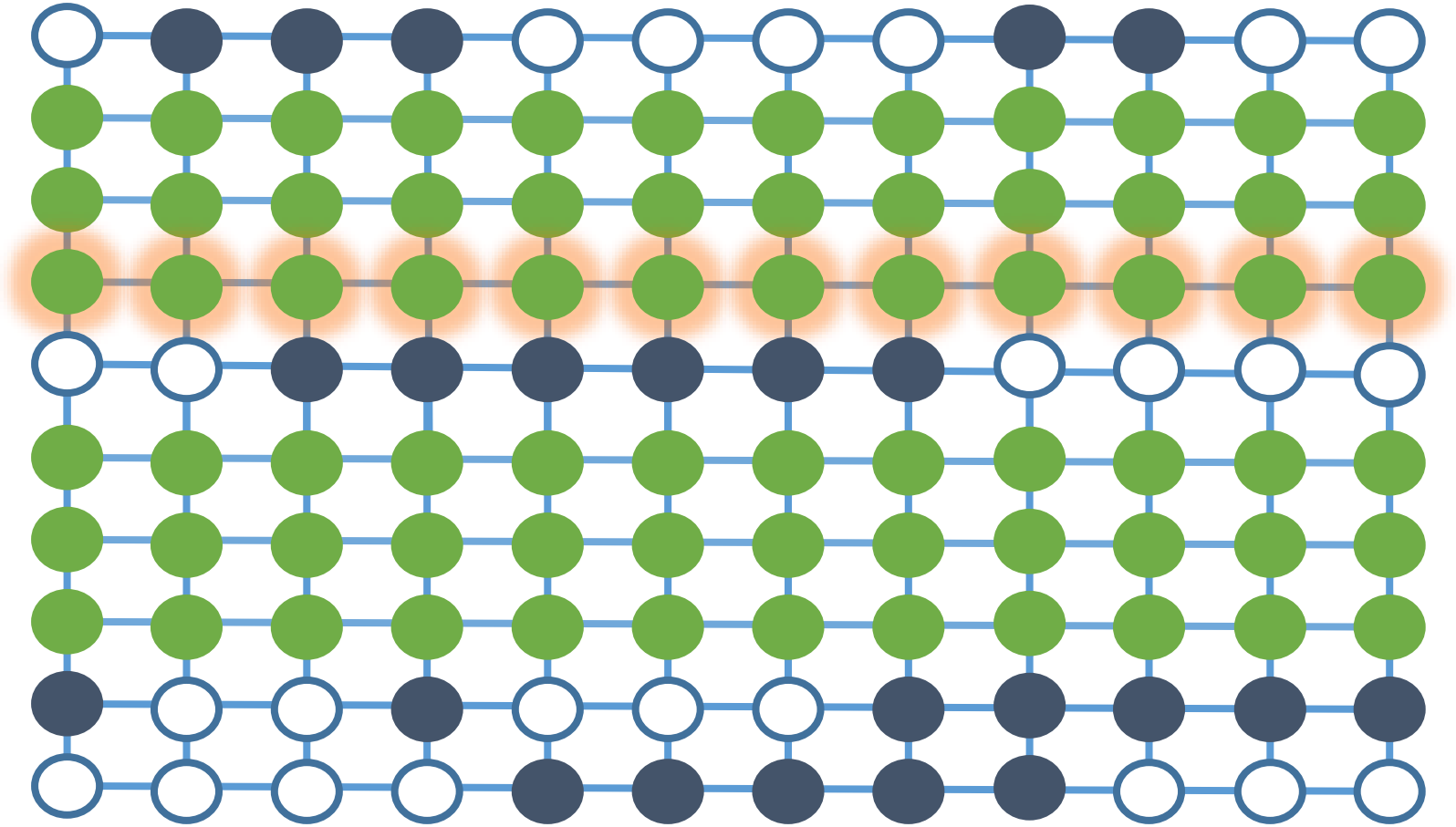
Step 2. Search for bit flips



Land sensitive data

Phys Feng Shui (intuition)

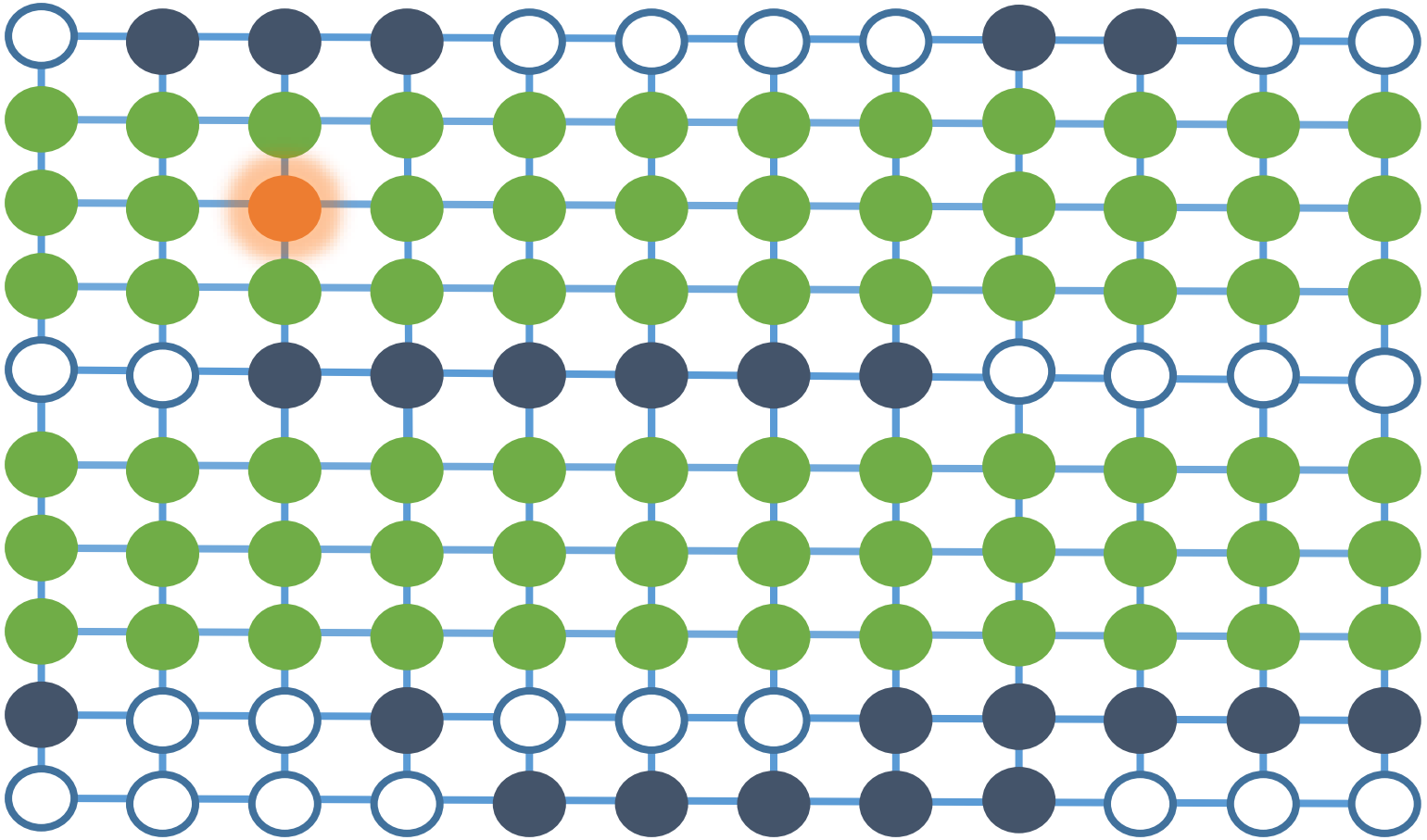
Step 2. Search for bit flips



Land sensitive data

Phys Feng Shui (intuition)

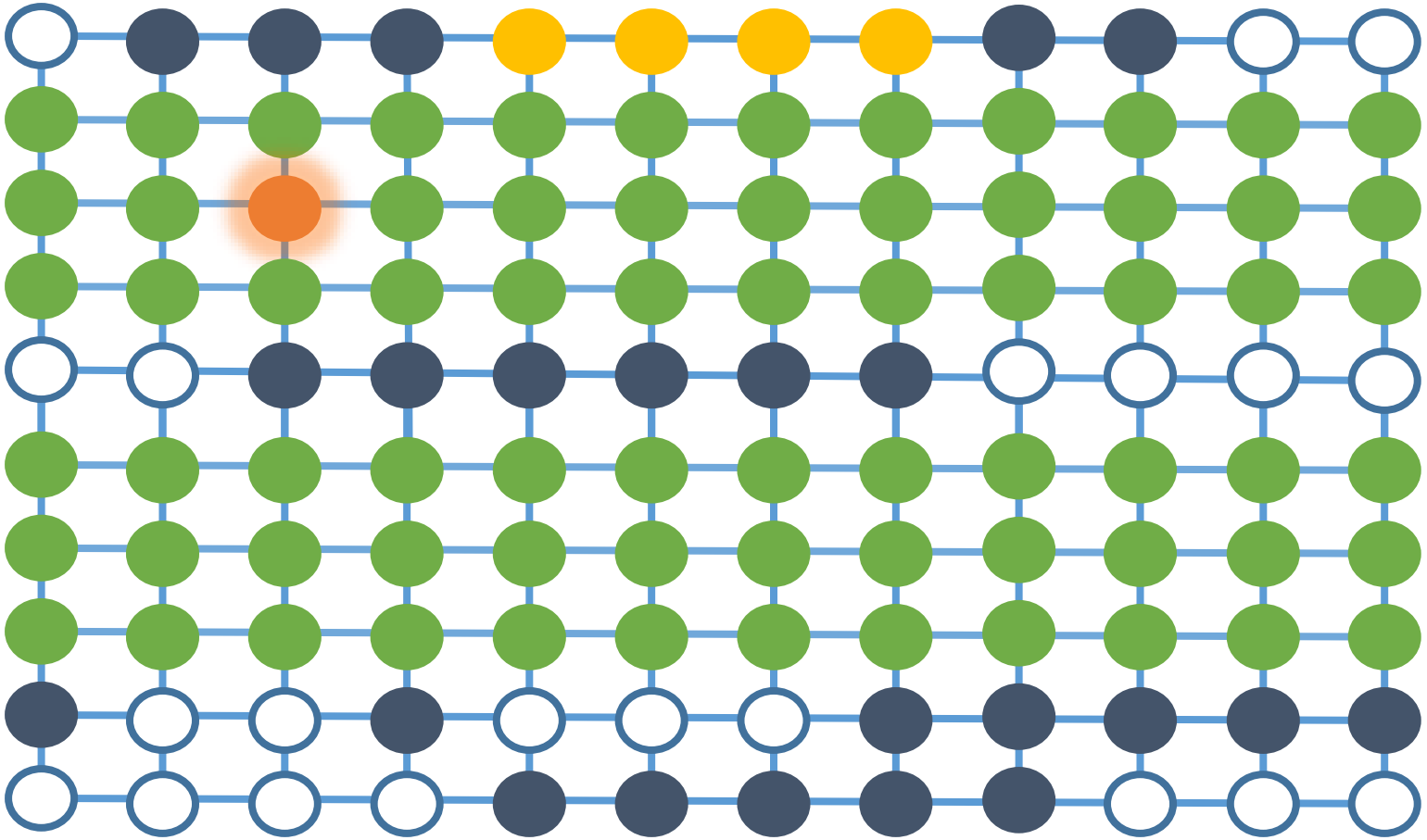
Step 2. Search for bit flips (got one!)



Land sensitive data

Phys Feng Shui (intuition)

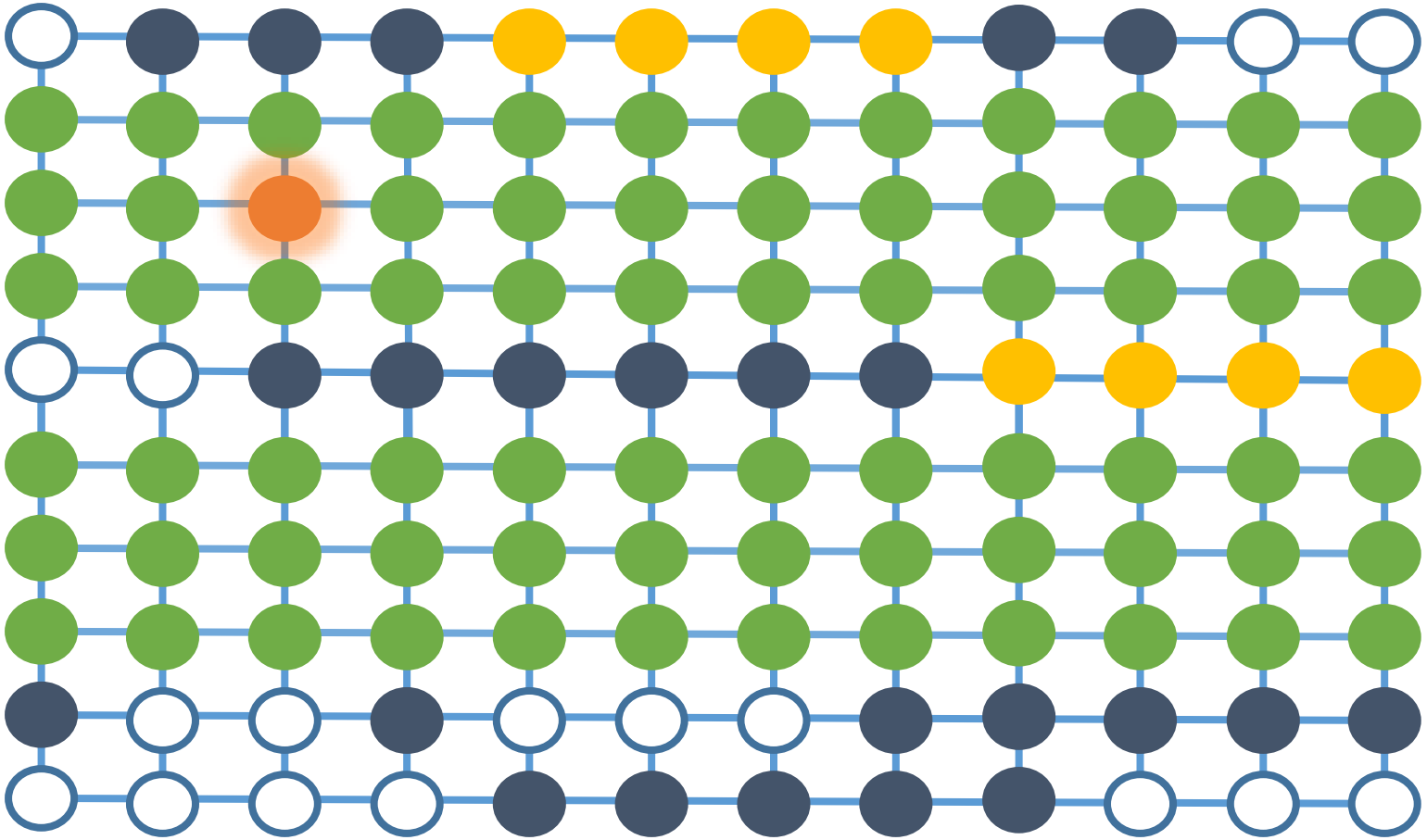
Step 3. Allocate small chunks



Land sensitive data

Phys Feng Shui (intuition)

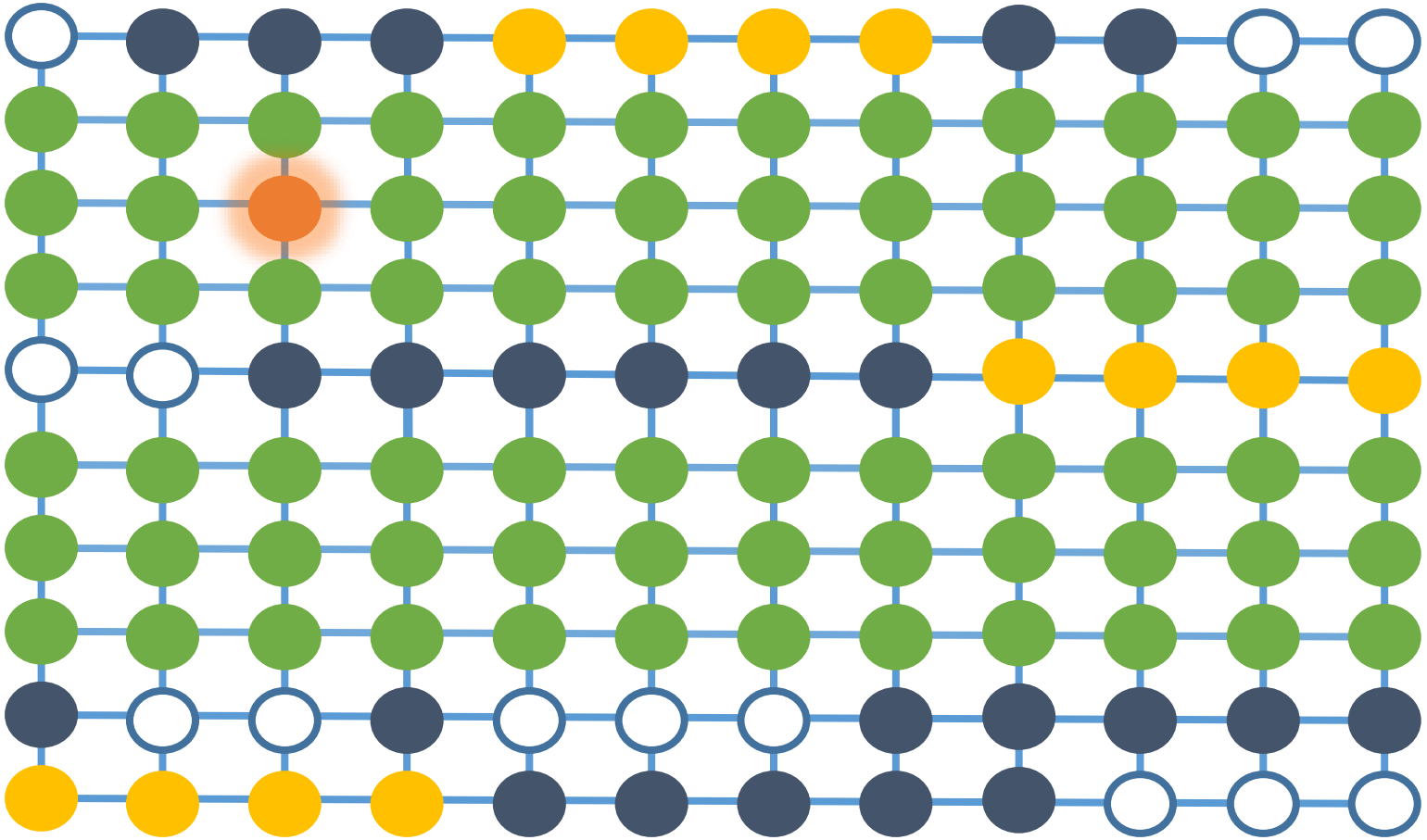
Step 3. Allocate small chunks



Land sensitive data

Phys Feng Shui (intuition)

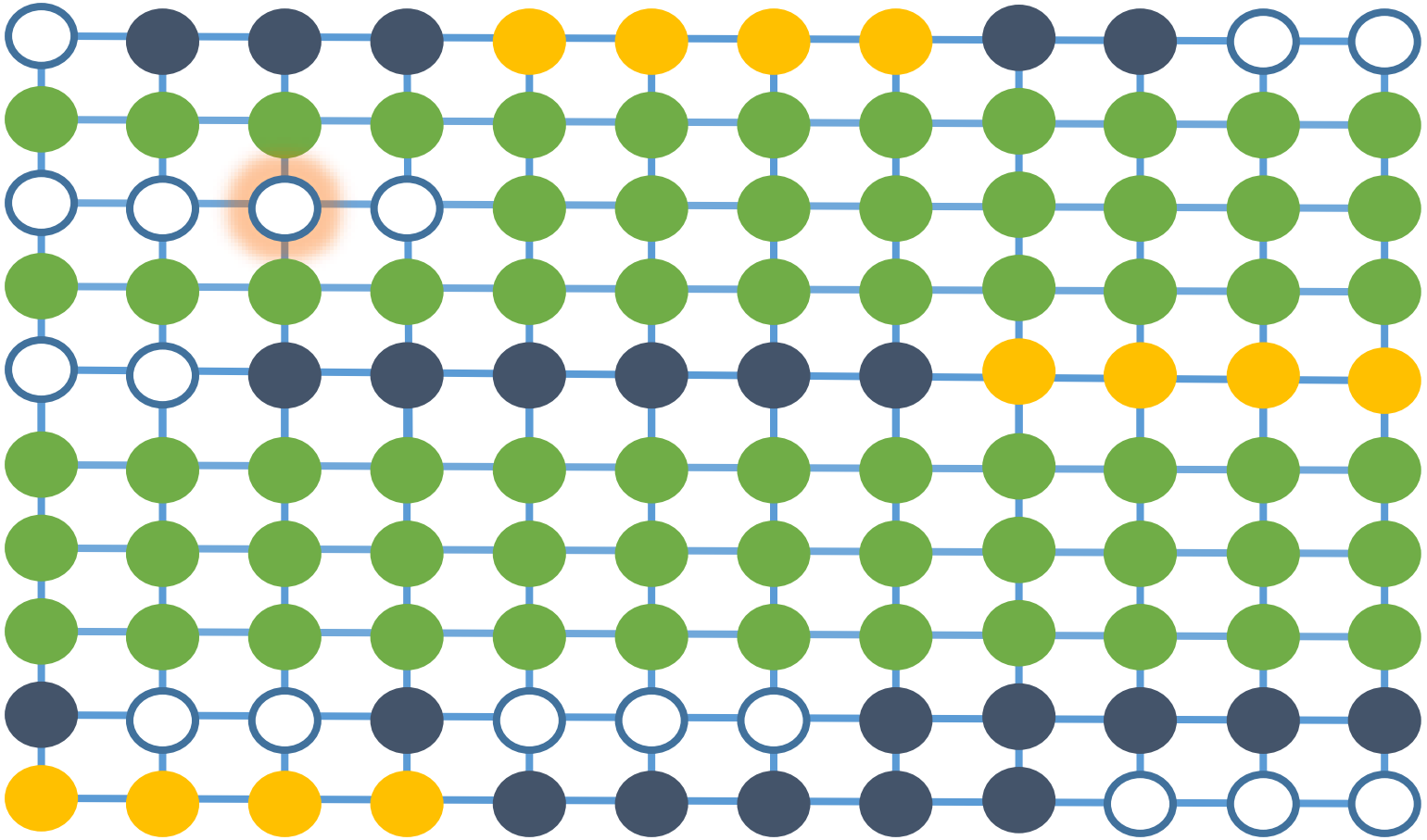
Step 3. Allocate small chunks



Land sensitive data

Phys Feng Shui (intuition)

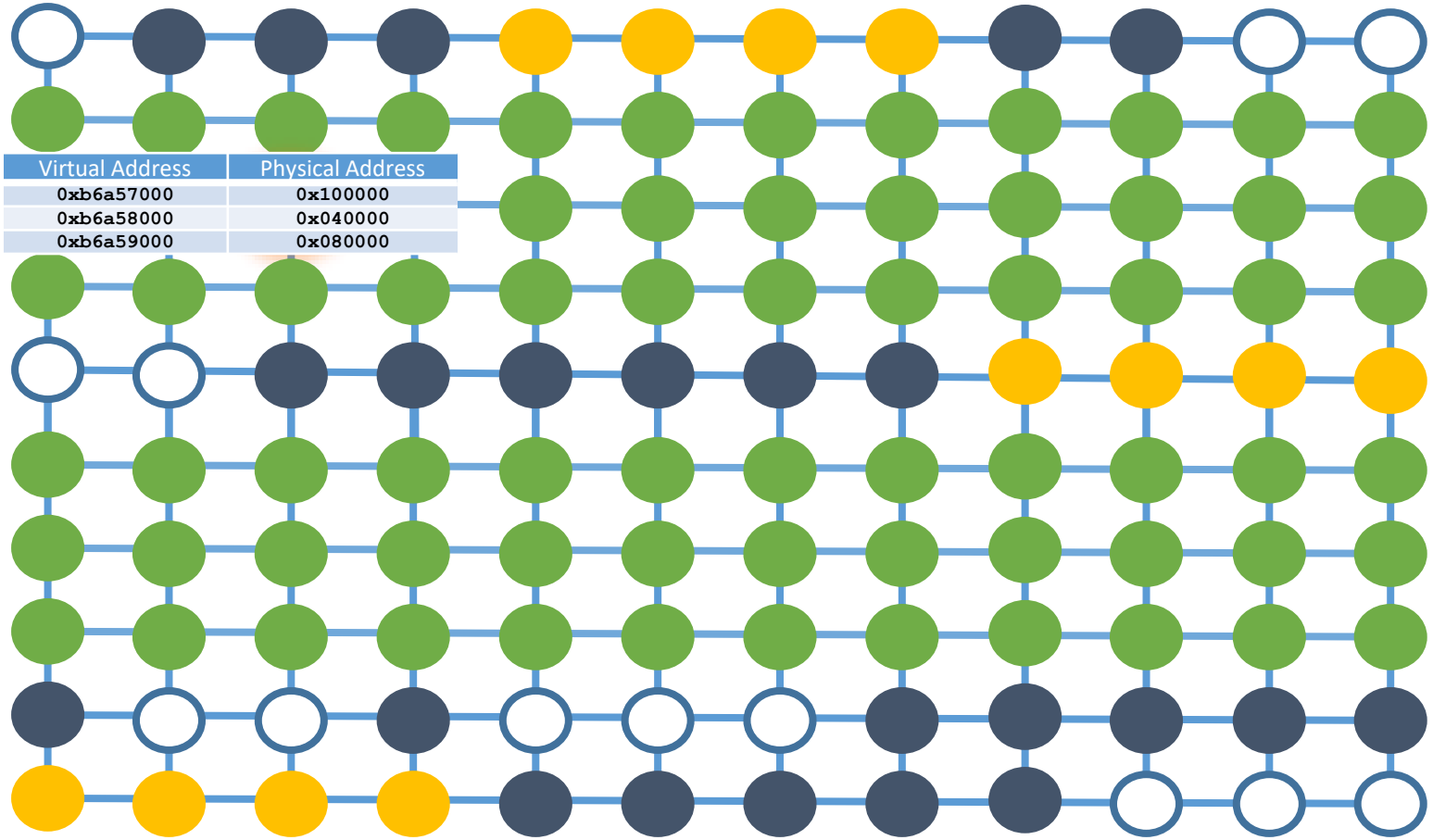
Step 4. Release vulnerable chunk



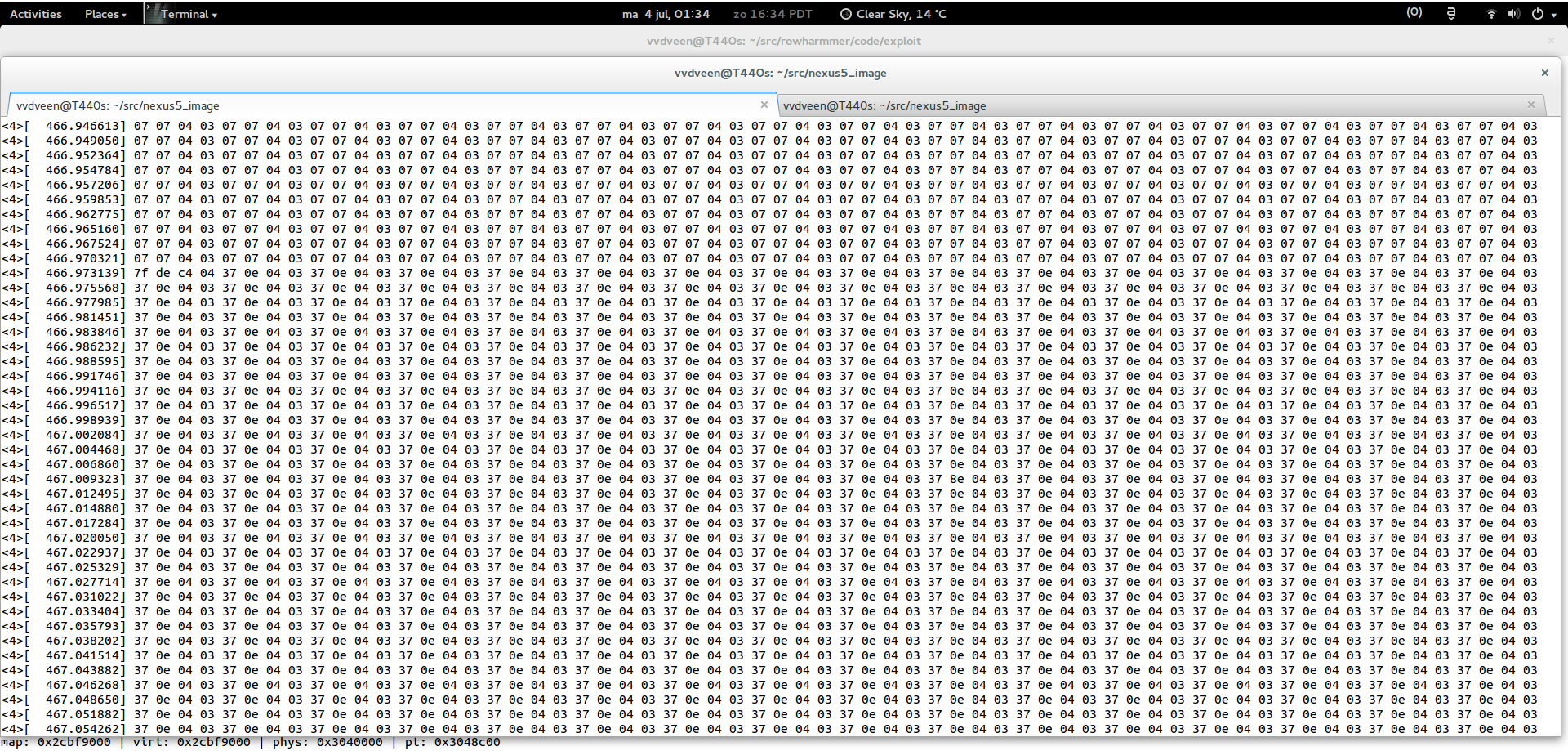
Land sensitive data

Phys Feng Shui (intuition)

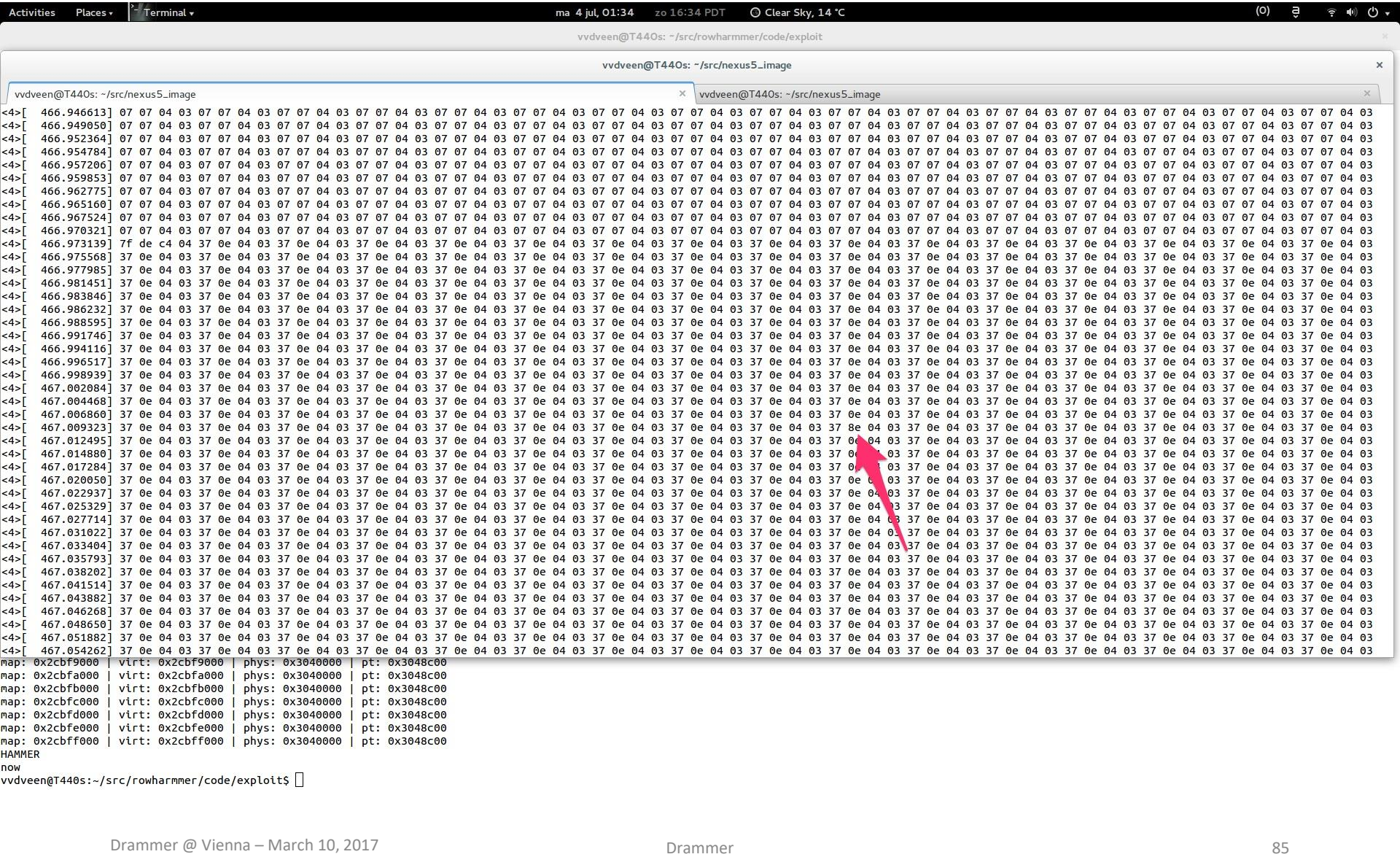
Step 5. Generate a new page table



Reproduce the bit flip



Reproduce the bit flip



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	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

Drammer test app reported bit flips on:

Google Pixel, OnePlus 3, Galaxy Note 7, …

Disclosure

Contacted Google with suggested mitigations on July 25, 2016

(91 days before #CCS16)

“Can you publish at another conference, later this year?”

“What if we support you financially?”

Disclosure

Contacted Google with suggested mitigations on July 25, 2016

(91 days before #CCS16)

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

Rewarded $4000

Partial hardening in November's updates

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

CONCLUSION

Now what?

Drammer

Widespread Rowhammer exploitation

- Cloud > Browser > **Mobile**
- Bit flips on LPDDR2, LPDDR3, LPDDR4 modules
- Reliable exploitation without custom OS features

Yes, we are also working on software defenses…

Software was never designed to deal with bit flips

- Proposed defenses are in early stages and easily bypassed
- Even hardware mitigations are currently still **optional**

You cannot fix Rowhammer

- More research is necessary

More details, demos, statistics, and **Drammer test app**

https://vusec.net/projects/drammer

open source: https://github.com/vusec/drammer

@vvdveen

BACKUP SLIDES

Did you not have enough yet?

ATTACK DETAILS

Determinism

Page Tables

Mapping virtual addresses to physical addresses

Page Tables

Mapping virtual addresses to physical addresses

Example lookup for input virtual address `0xb6a5717f`

Page Tables

Mapping virtual addresses to physical addresses

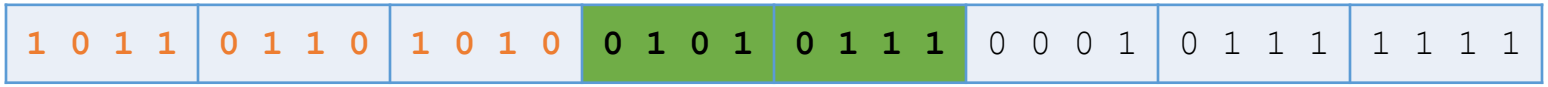
Example lookup for input virtual address `0xb6a5717f`

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

Page Tables

Mapping virtual addresses to physical addresses

Example lookup for input virtual address `0xb6a5717f`

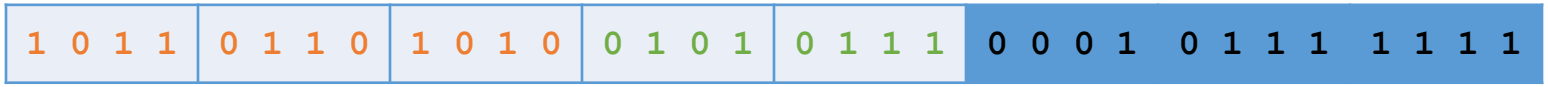


- Highest 12 bits: *level 1 table index* (*Translation Table Base Register*)
- Middle 8 bits: *level 2 table index*

Page Tables

Mapping virtual addresses to physical addresses

Example lookup for input virtual address `0xb6a5717f`

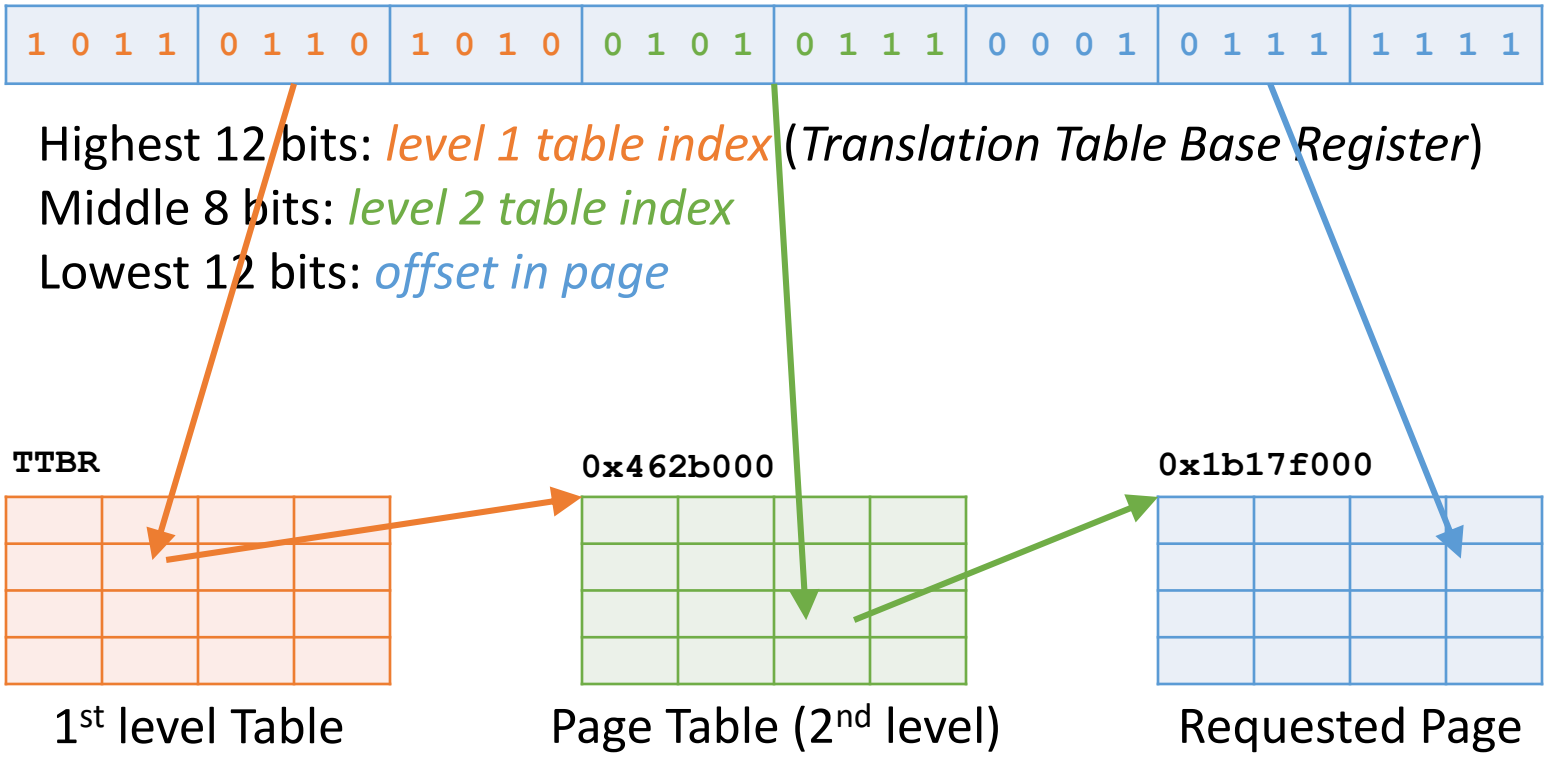


- Highest 12 bits: *level 1 table index* (*Translation Table Base Register*)
- Middle 8 bits: *level 2 table index*
- Lowest 12 bits: *offset in page*

Page Tables

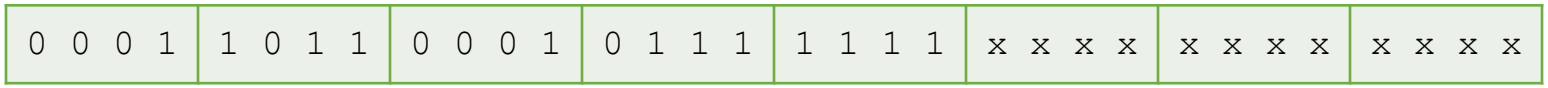
Mapping virtual addresses to physical addresses

Example lookup for input virtual address `0xb6a5717f`



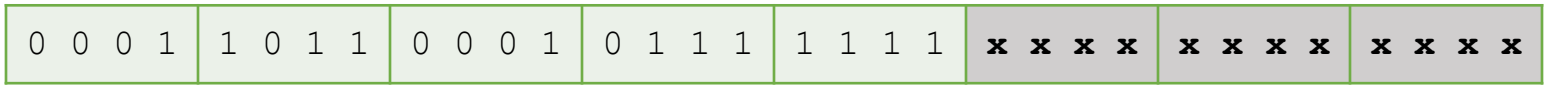
Page Table Entries

Entry in the (2nd level) Page Table



Page Table Entries

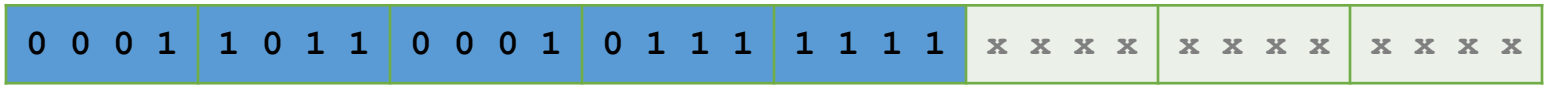
Entry in the (2nd level) Page Table



- 12 bits of *properties*

Page Table Entries

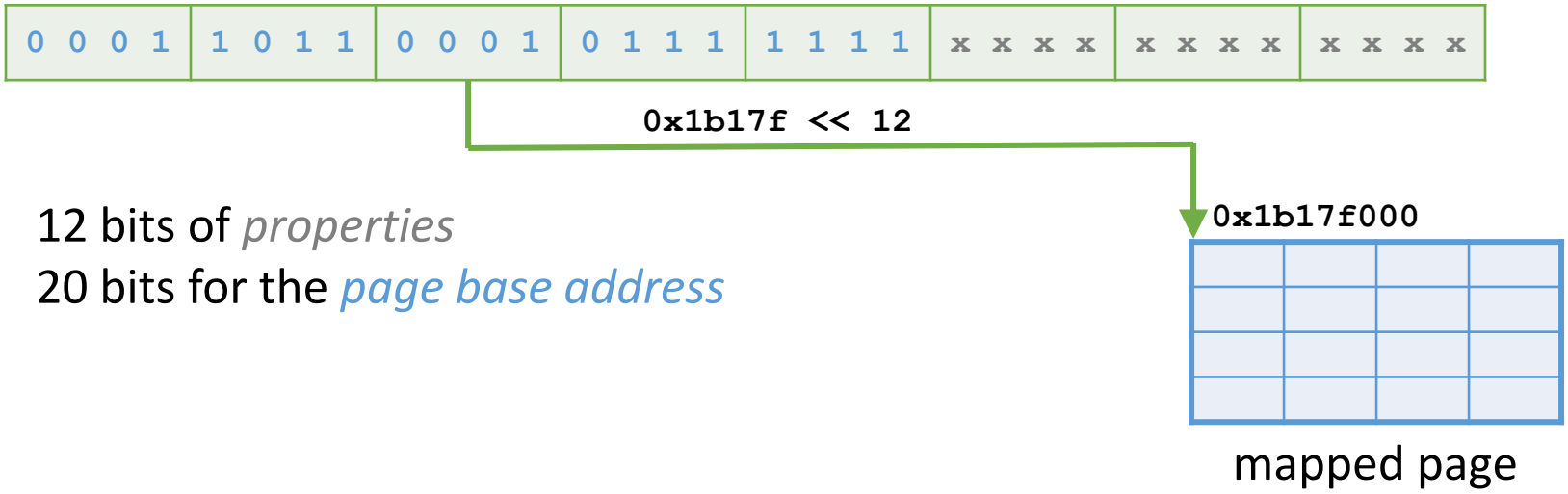
Entry in the (2nd level) Page Table



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

Page Table Entries

Entry in the (2nd level) Page Table

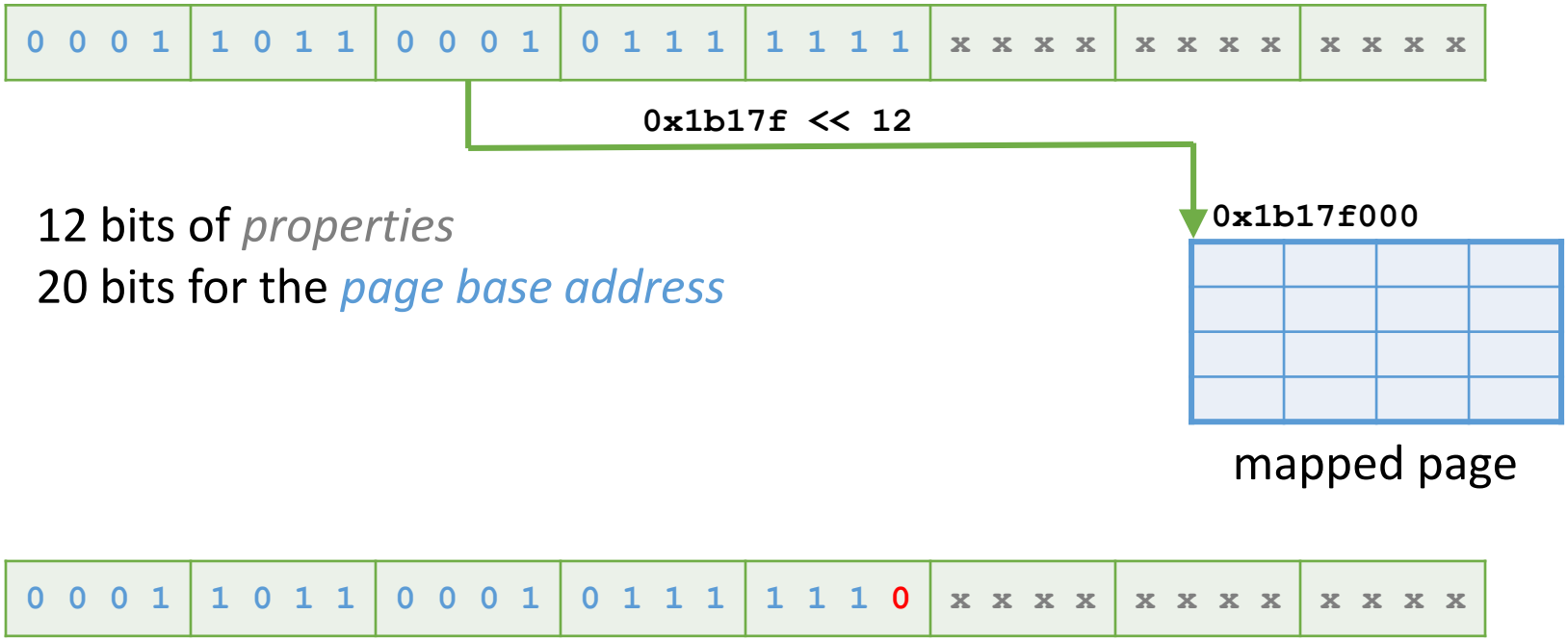


- 12 bits of *properties*
- 20 bits for the *page base address*

What if we flip a bit in the entry?

Flipping Page Table Entries

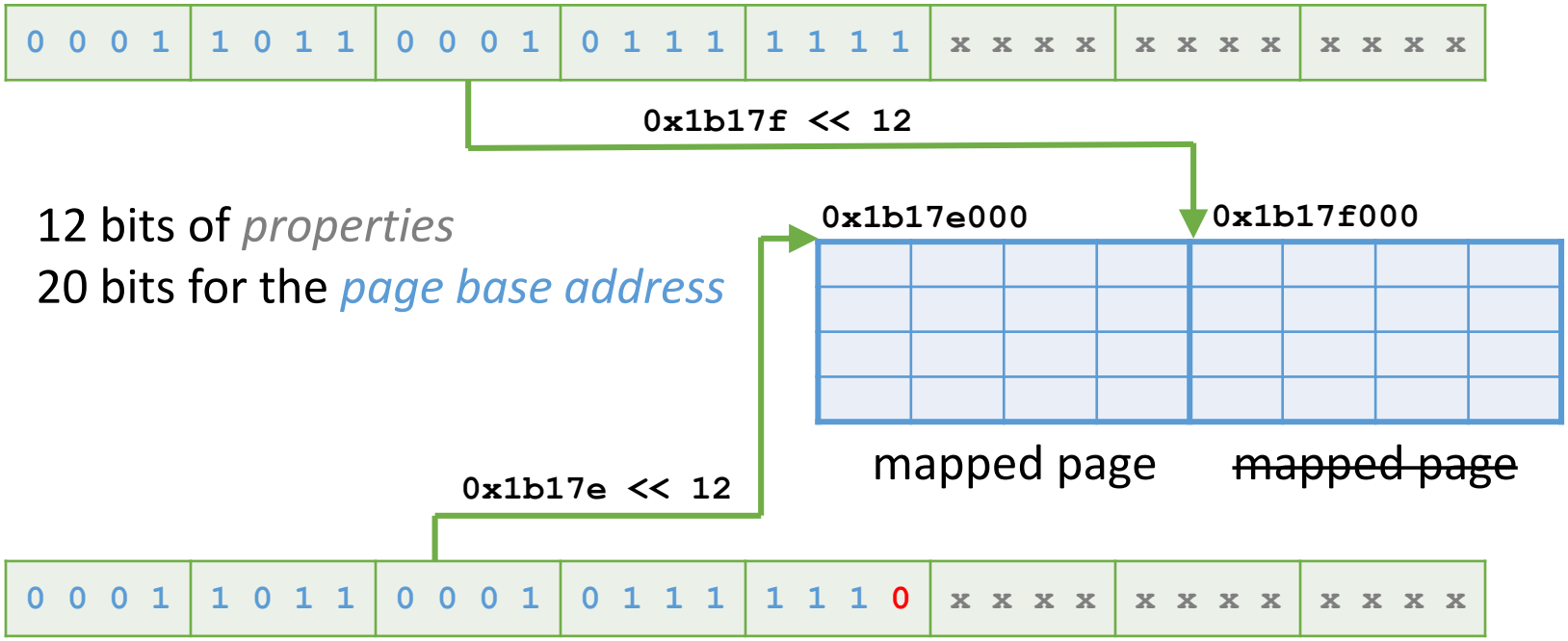
Entry in the (2nd level) Page Table



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

Flipping Page Table Entries

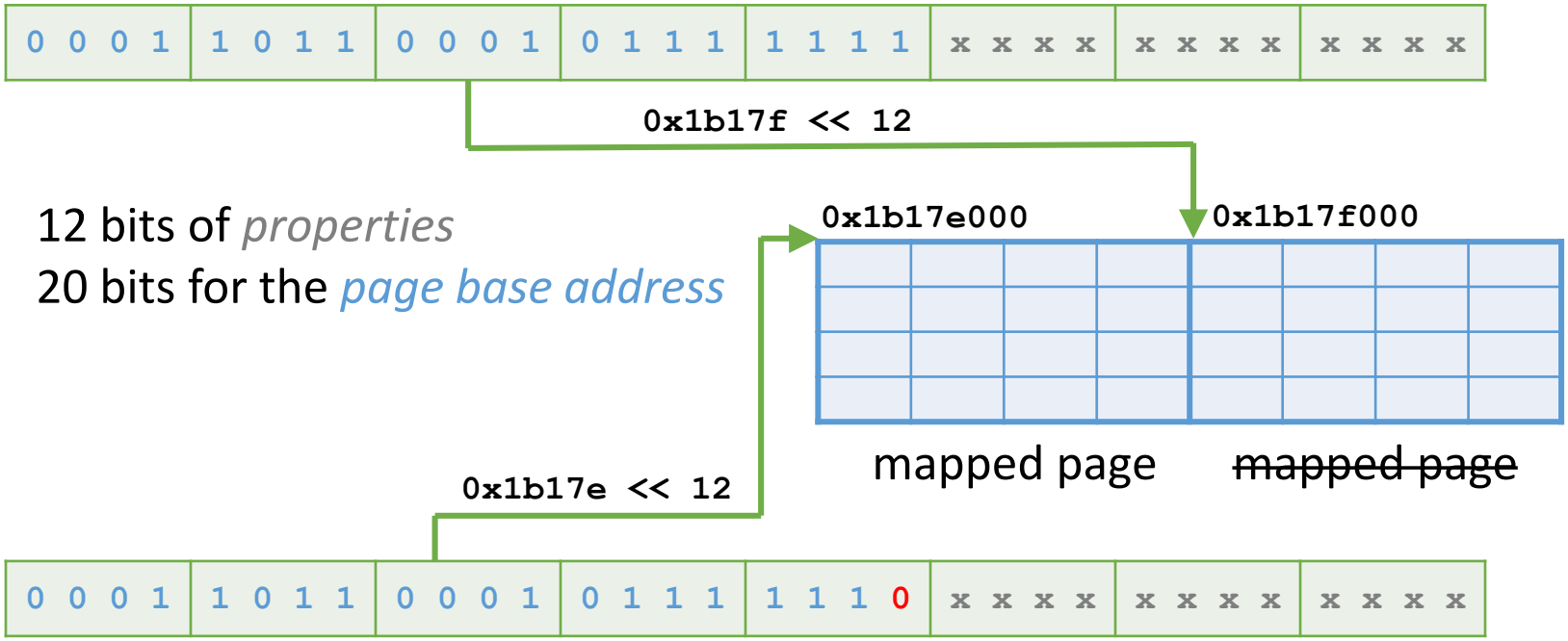
Entry in the (2nd level) Page Table



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

Flipping Page Table Entries

Entry in the (2nd 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’

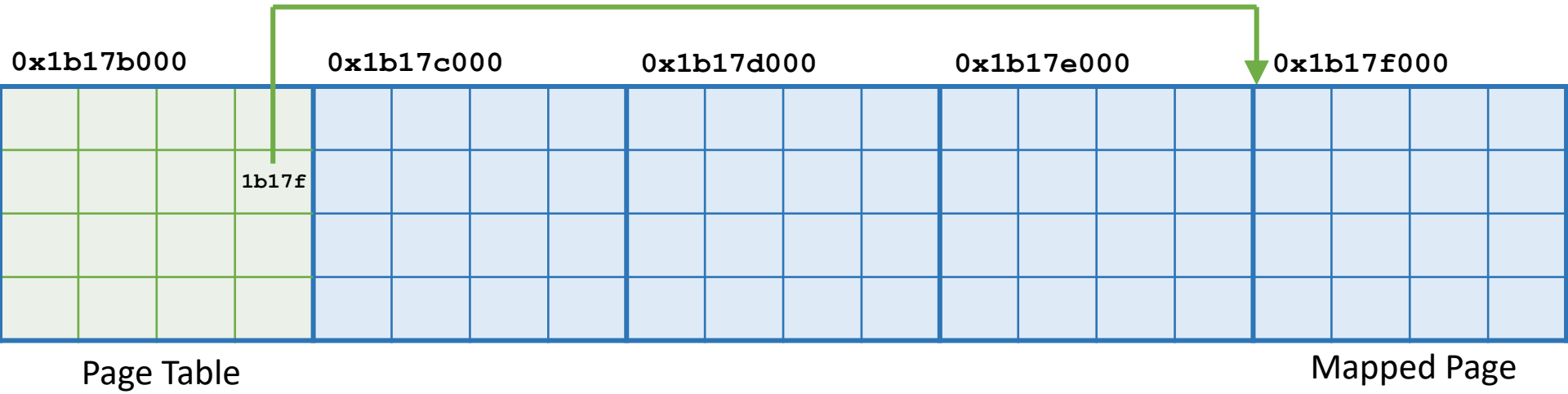
- Flip offset 0: –1 page
- Flip offset 1: –2 pages
- Flip offset 2: –4 pages
- Flip offset *n*: -2^n pages

Flipping Page Table Entries

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

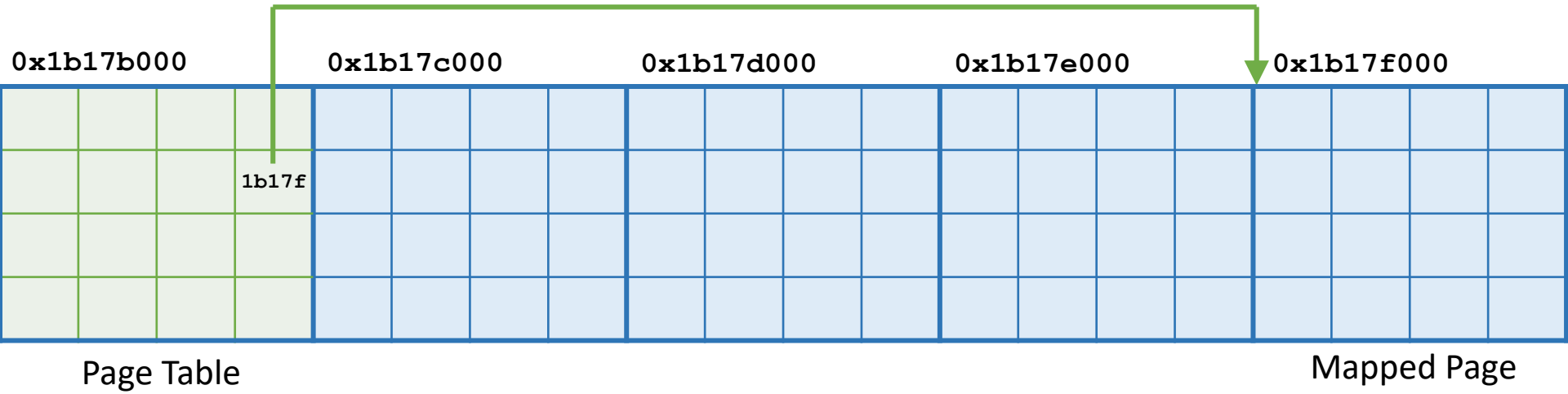
Flipping Page Table Entries

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



Flipping Page Table Entries

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



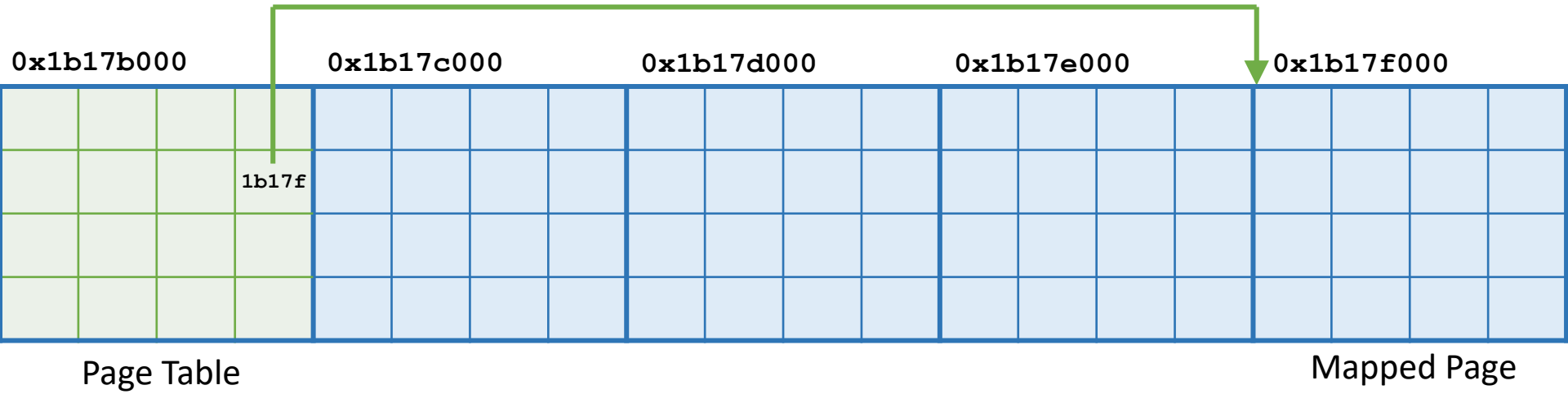
Virtual address `0xb6a57000` maps to Page Table Entry:

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`**

Flipping Page Table Entries

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



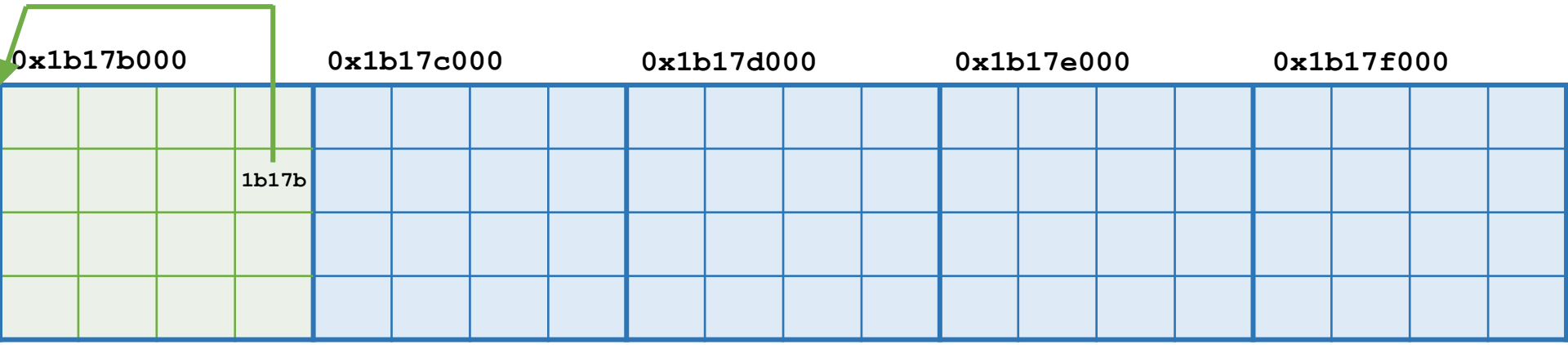
Virtual address `0xb6a57000` maps to Page Table Entry:

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`**

Flipping 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

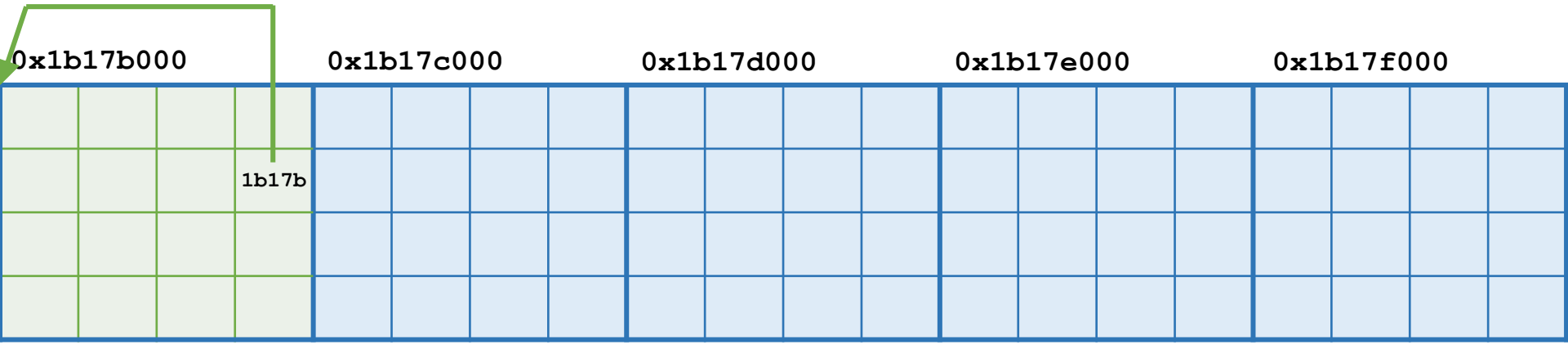
Virtual address `0xb6a57000` maps to Page Table Entry:

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**

Flipping 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

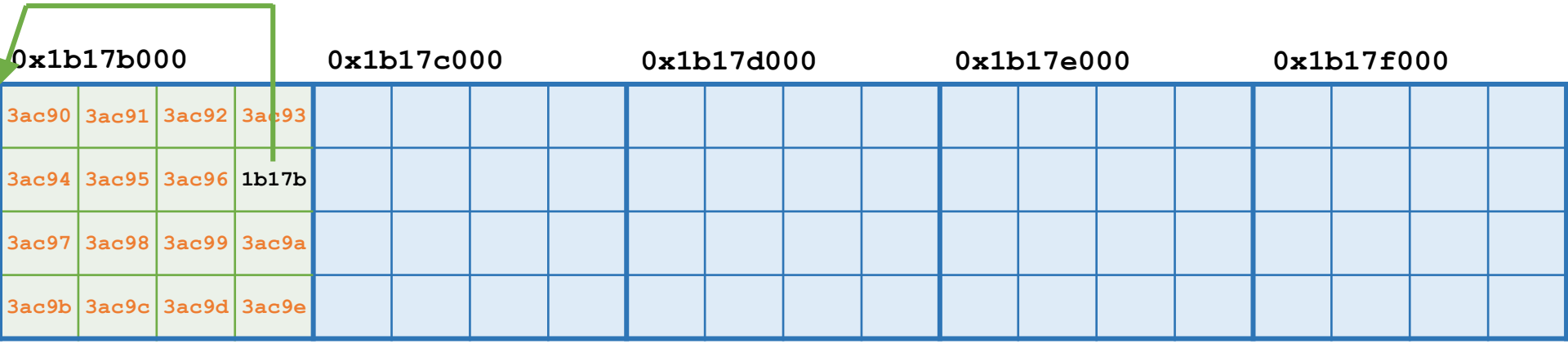
Virtual address `0xb6a57000` maps to Page Table Entry:

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**

Flipping 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

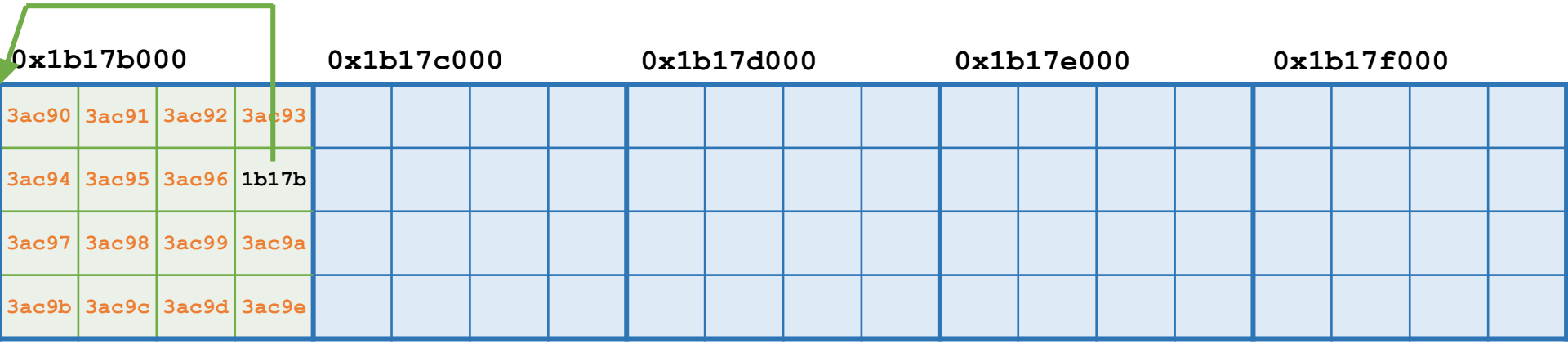
Virtual address `0xb6a57000` maps to Page Table Entry:

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**

Flipping 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

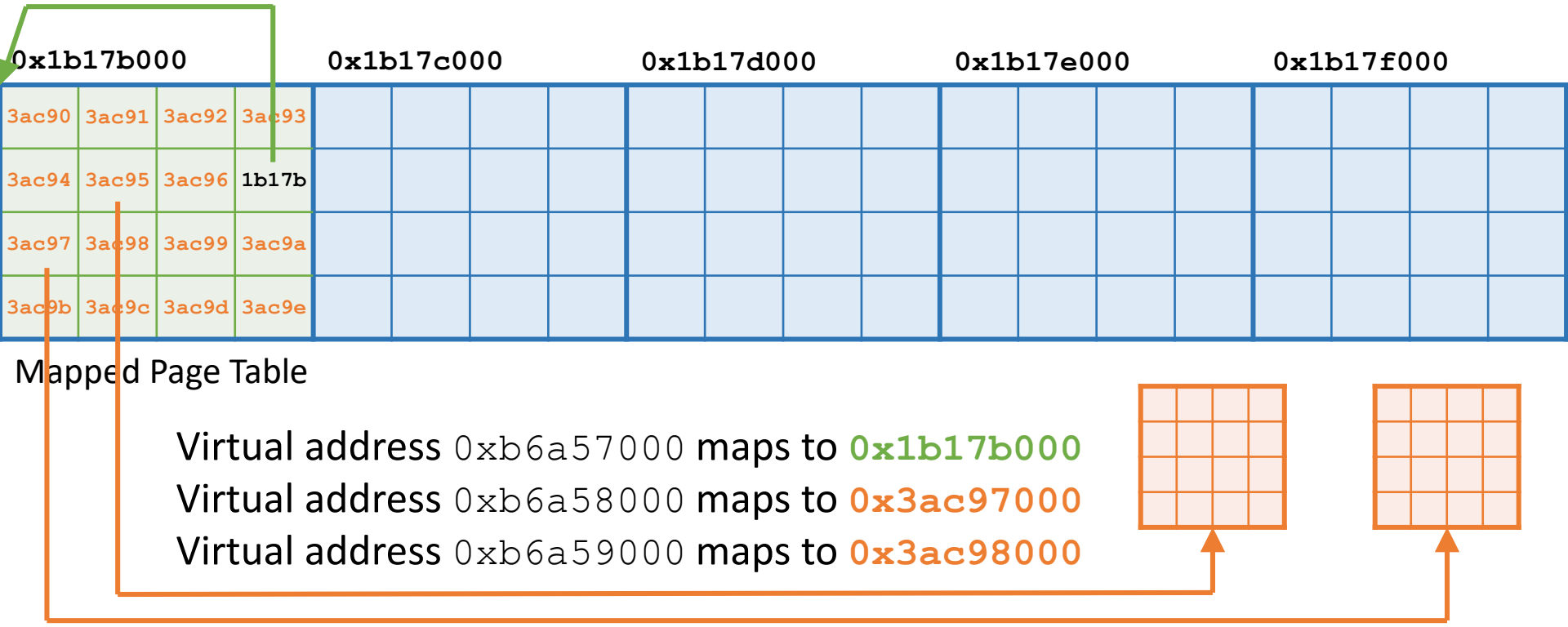
Virtual address `0xb6a57000` maps to Page Table Entry:

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**

Flipping 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



Landing Page Tables

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

Landing Page Tables

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

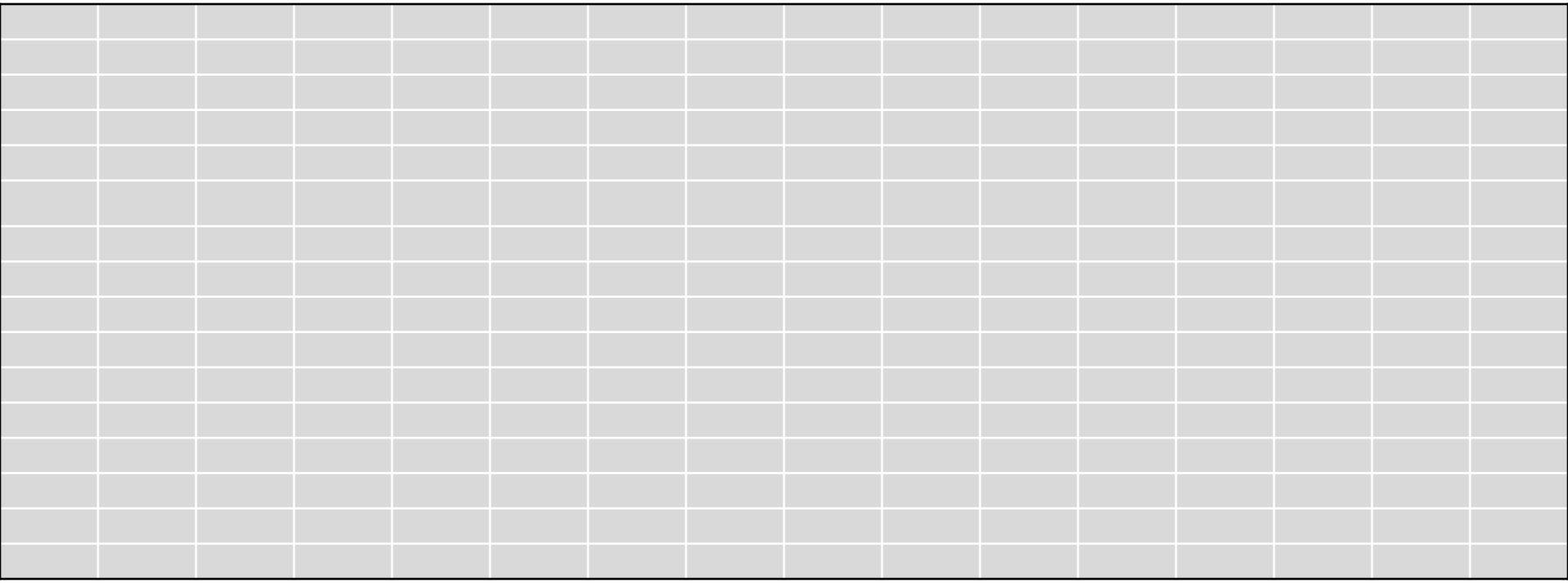
Phys Feng Shui

Landing Page Tables

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

Phys Feng Shui

Physical memory:

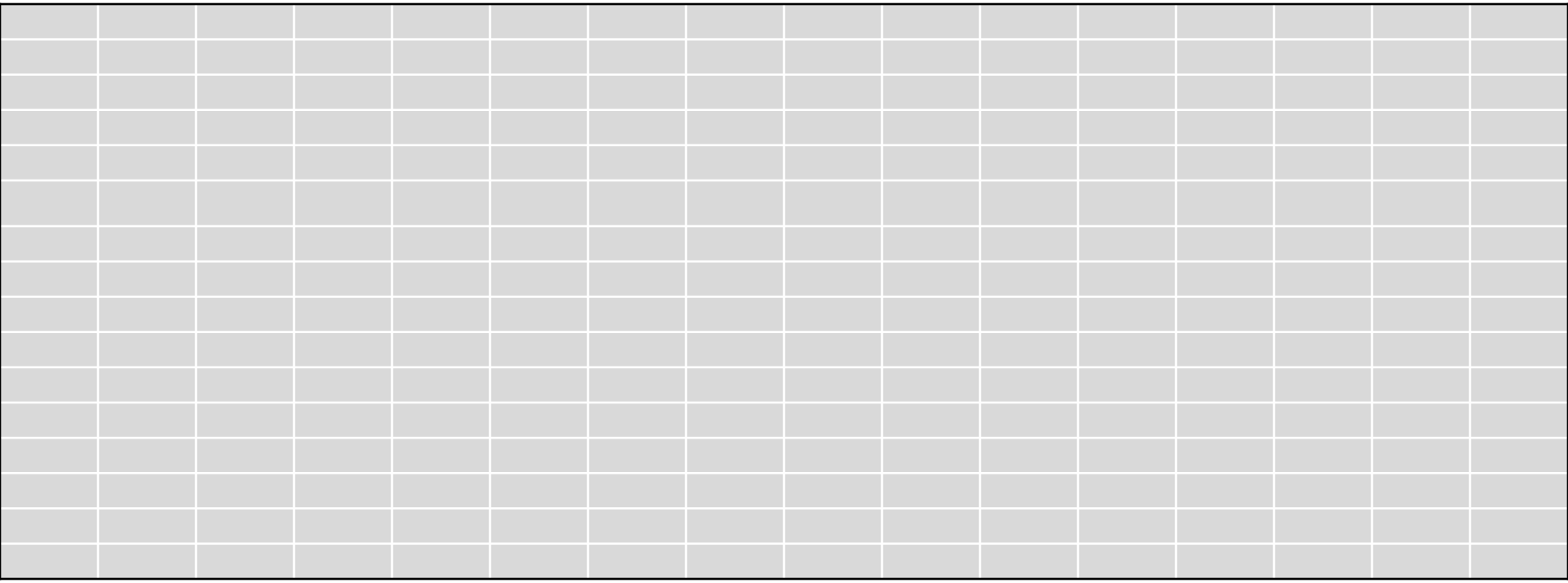


Landing Page Tables

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

Phys Feng Shui

Physical memory:



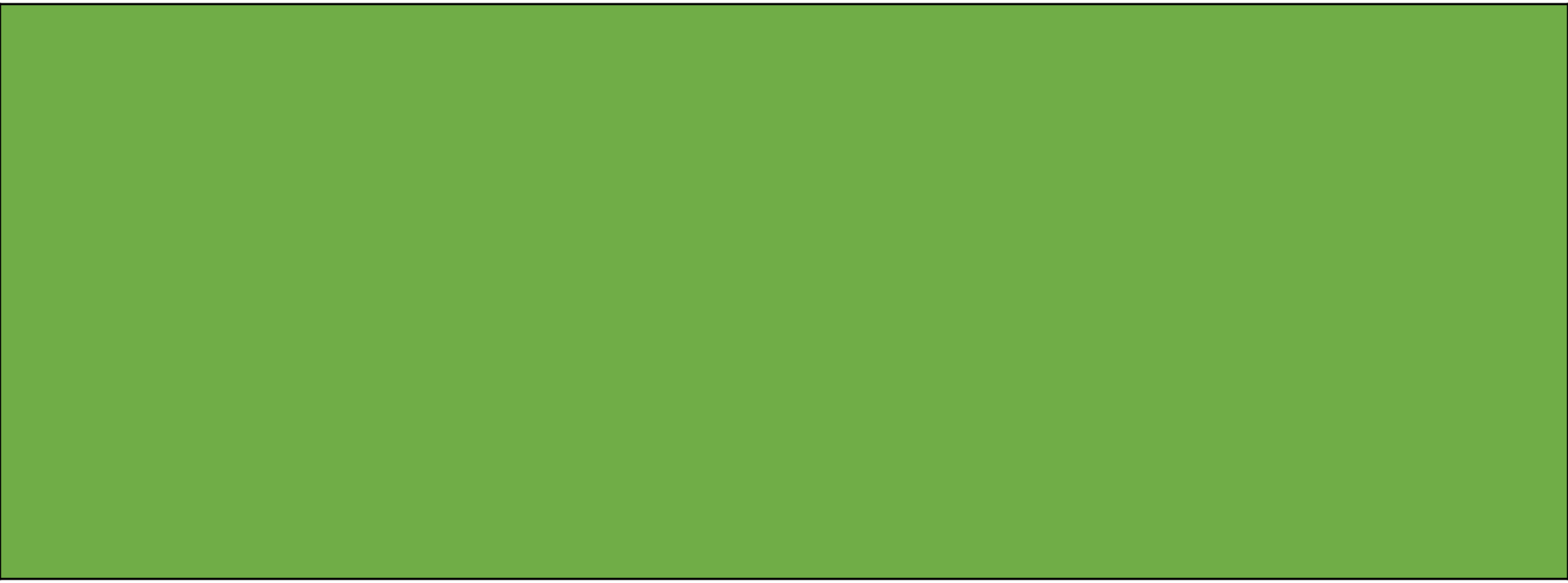
Exhaust all memory

Landing Page Tables

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

Phys Feng Shui

Physical memory:



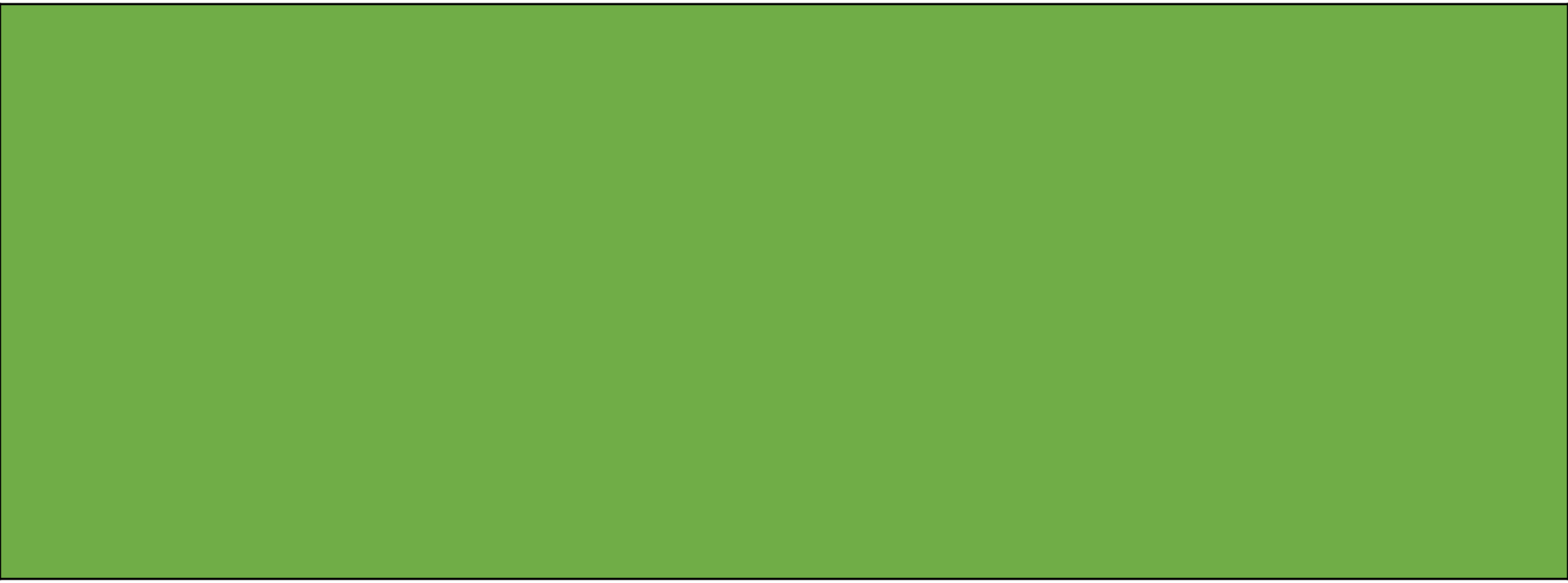
Exhaust all memory

Landing Page Tables

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

Phys Feng Shui

Physical memory:



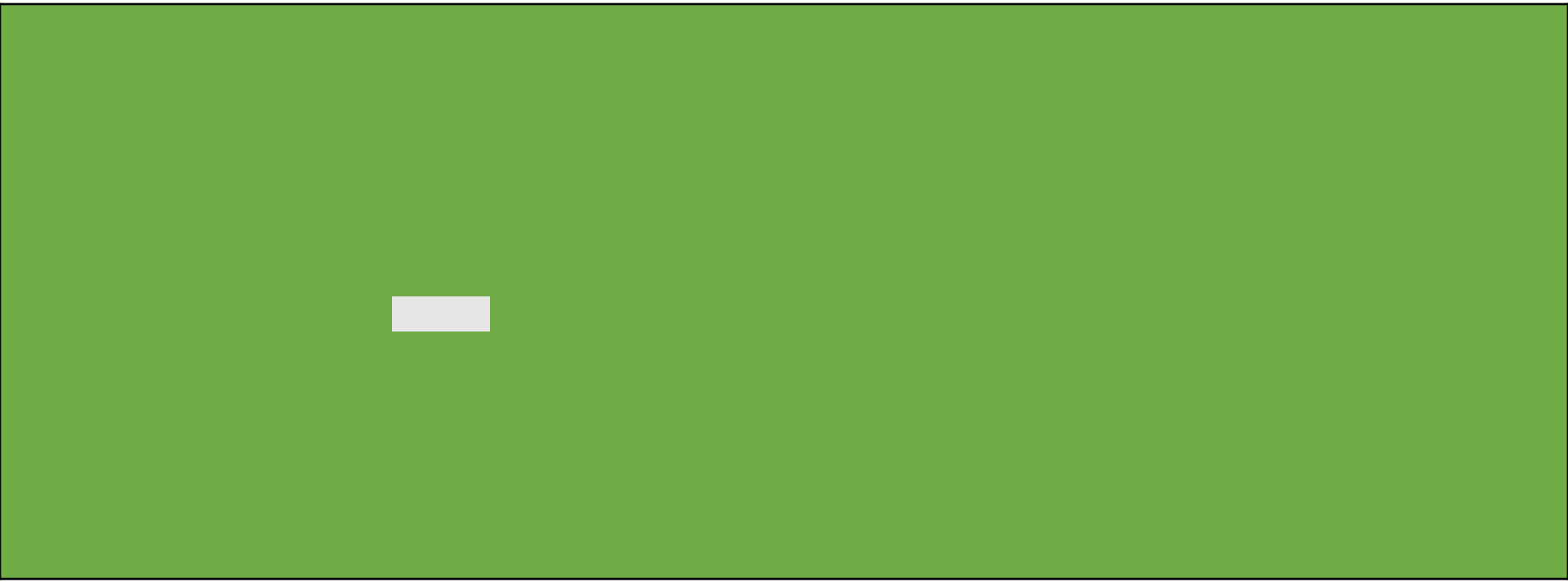
Release the vulnerable page

Landing Page Tables

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

Phys Feng Shui

Physical memory:



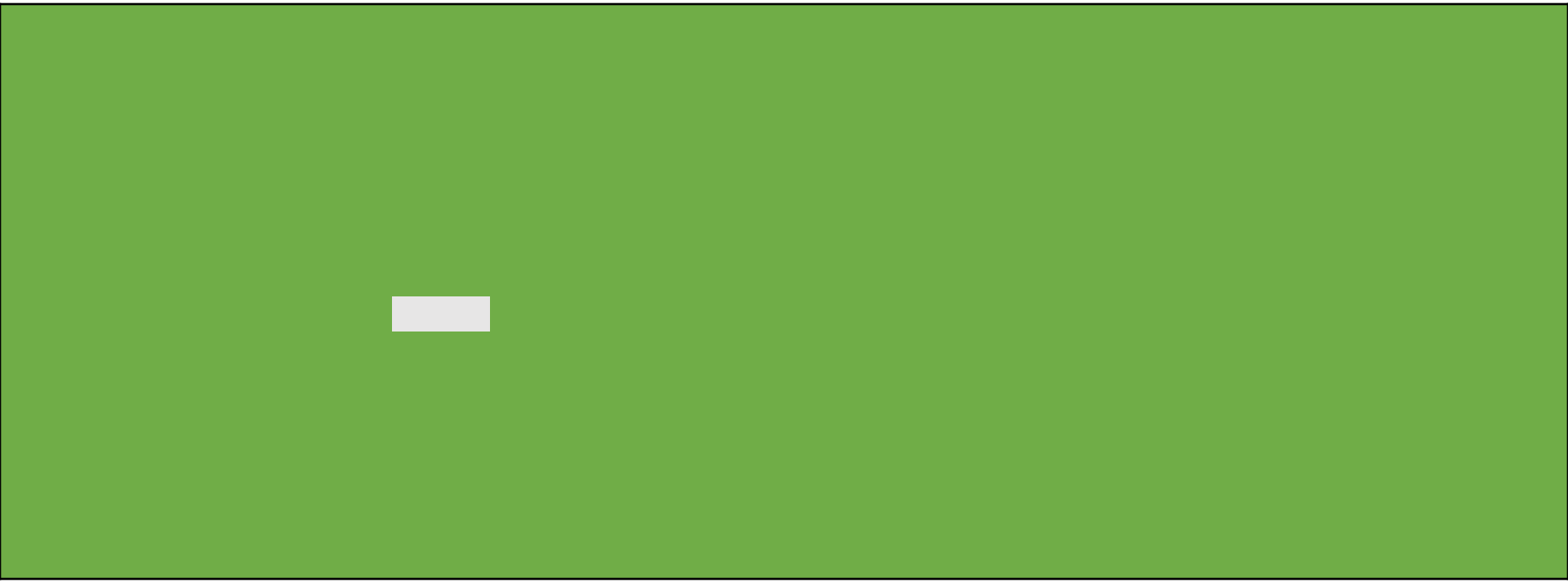
Release the vulnerable page

Landing Page Tables

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

Phys Feng Shui

Physical memory:



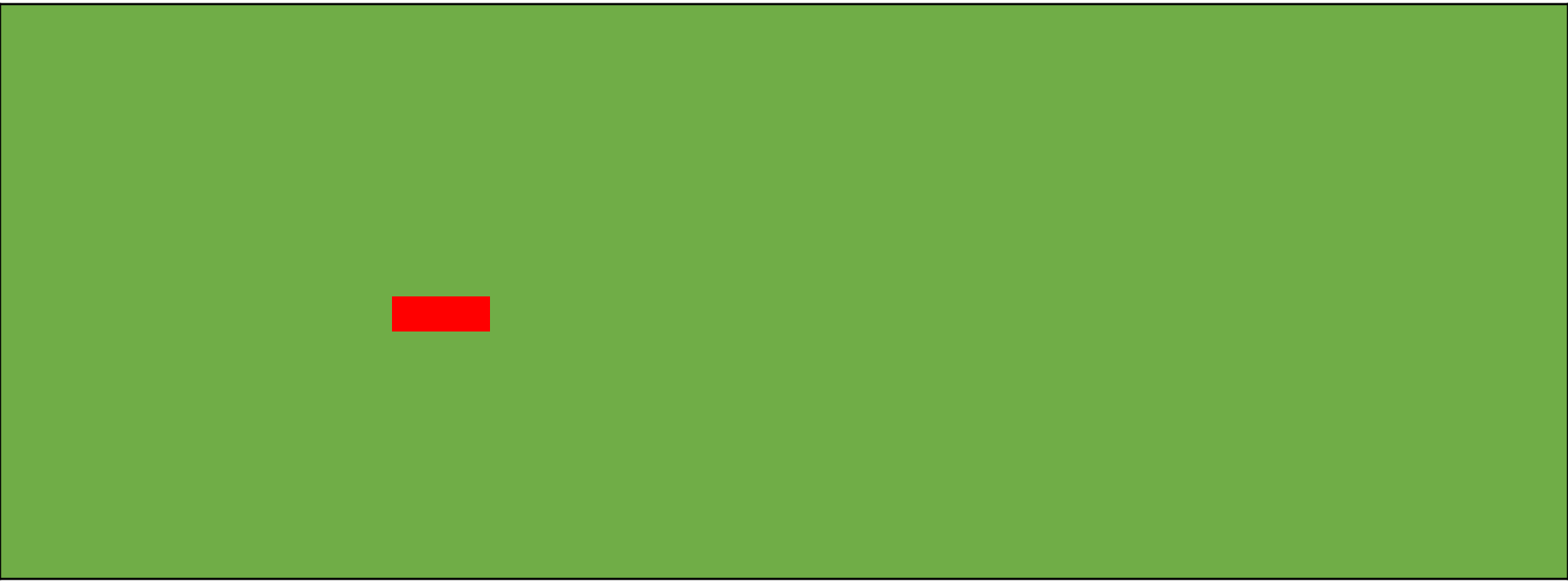
Trigger a Page Table Allocation

Landing Page Tables

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

Phys Feng Shui

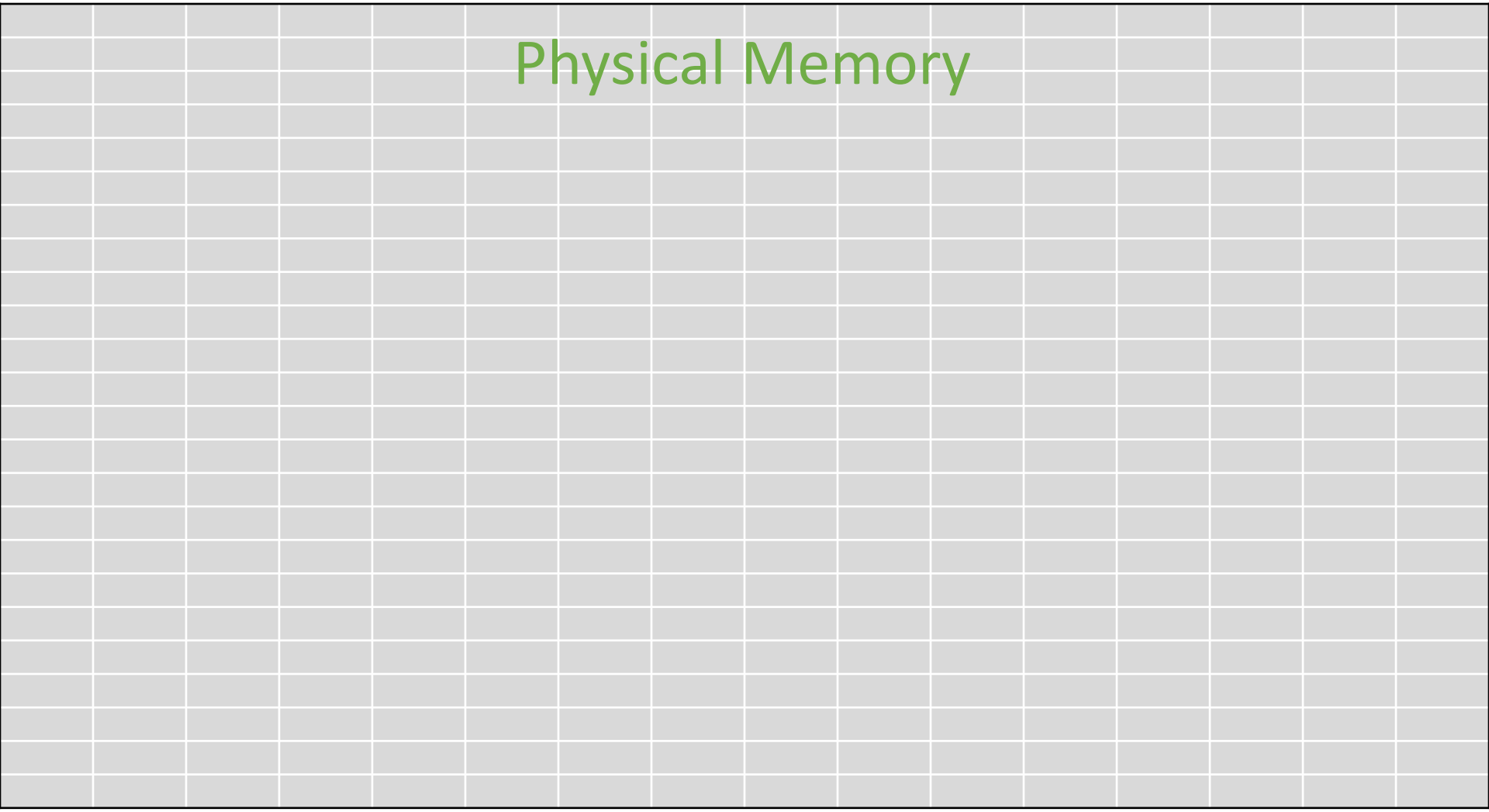
Physical memory:



Trigger a Page Table Allocation

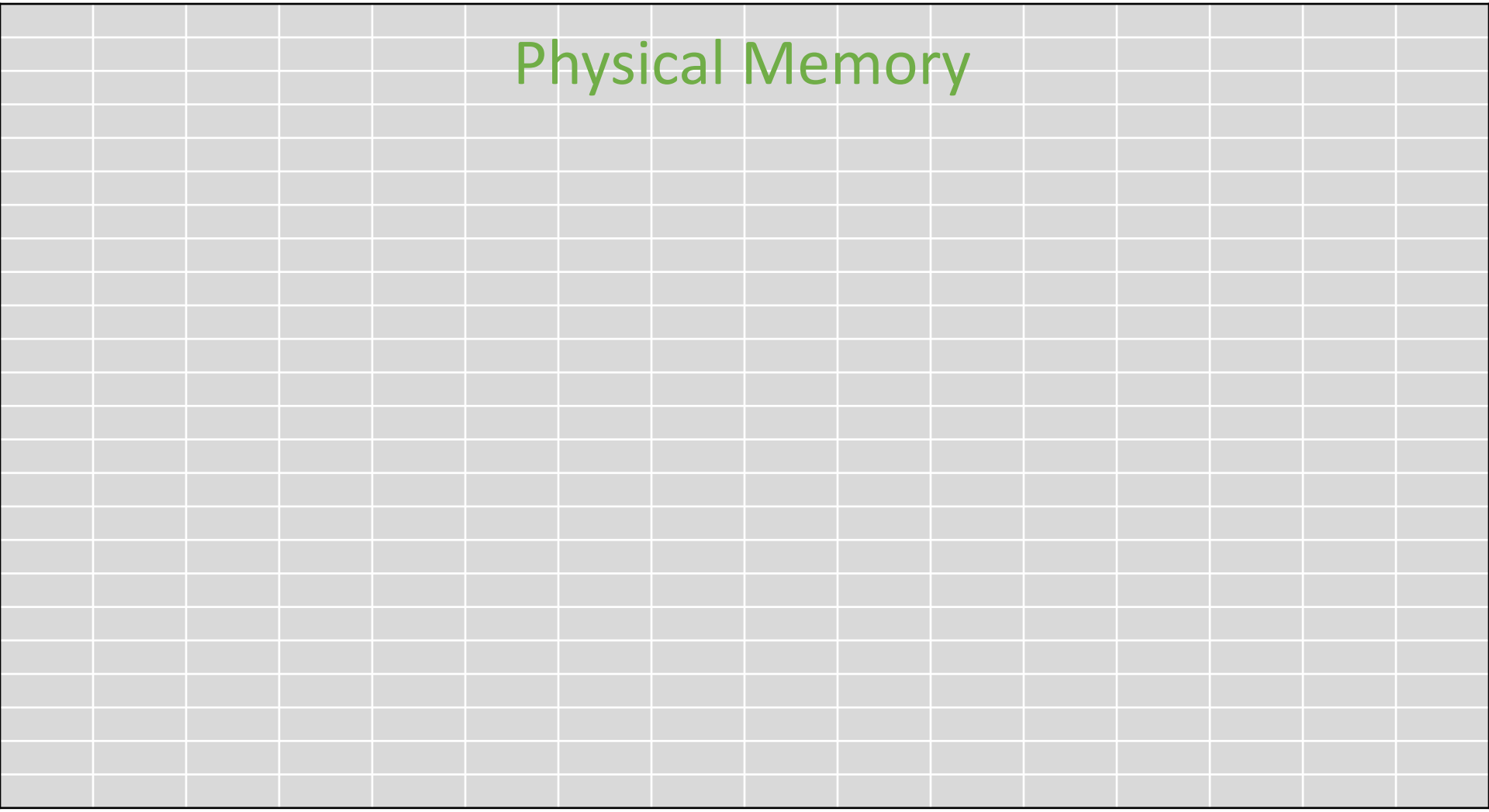
Phys Feng Shui

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

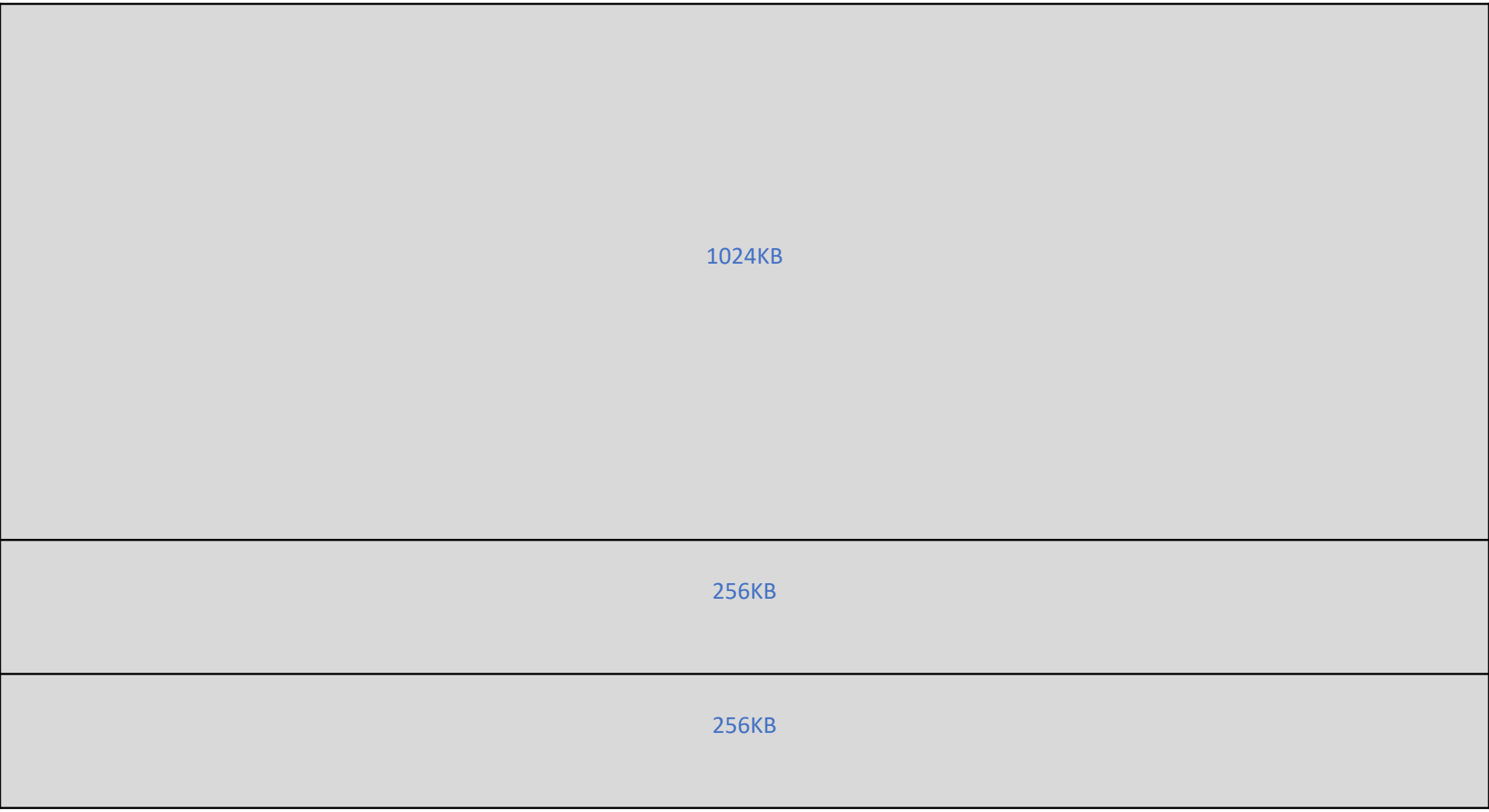
Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

```
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)*

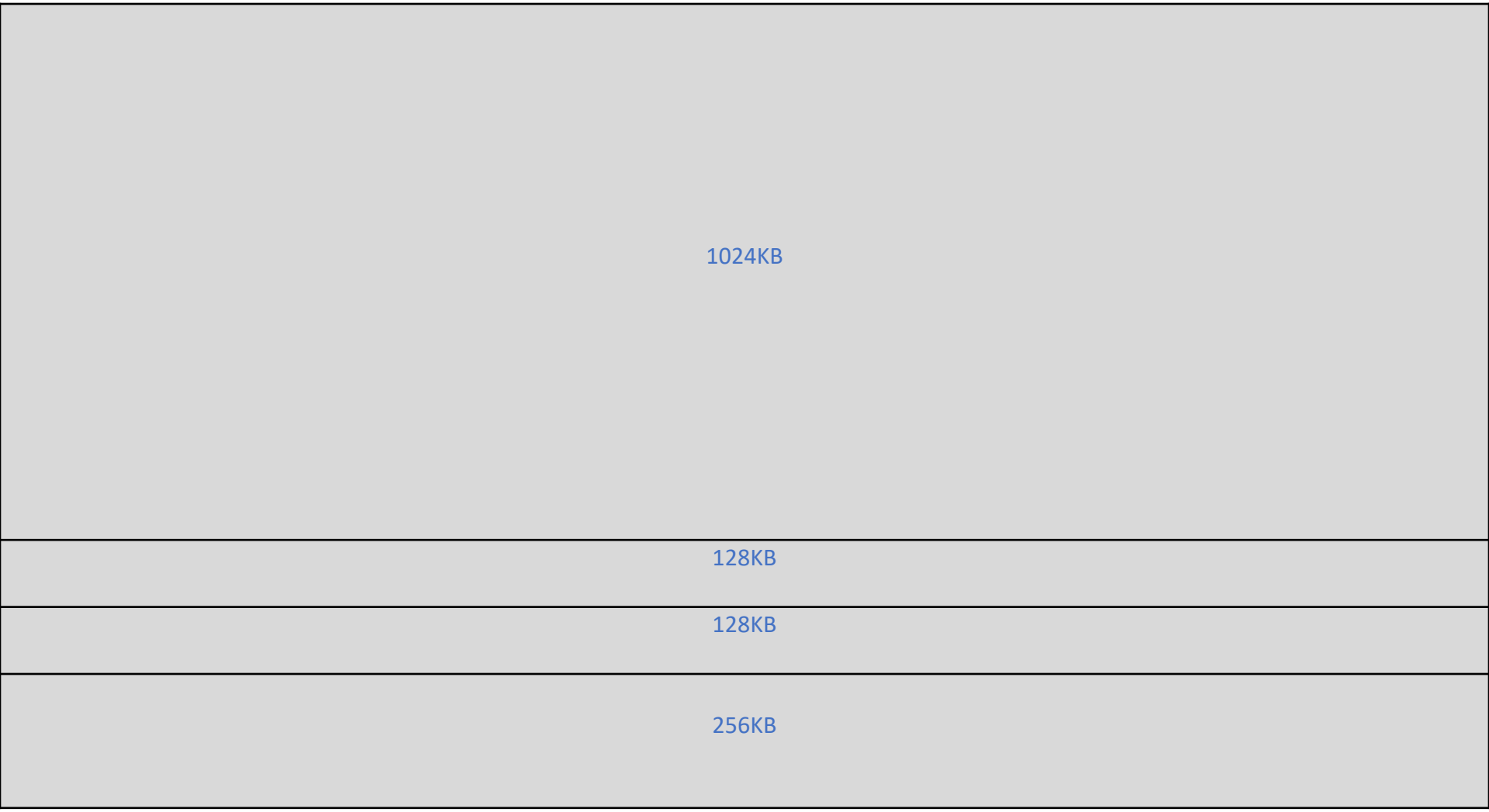
```
X1 = __get_free_pages(flags, 6); // get 2^6 = 64KB of memory
```



Phys Feng Shui – Buddy Allocator

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

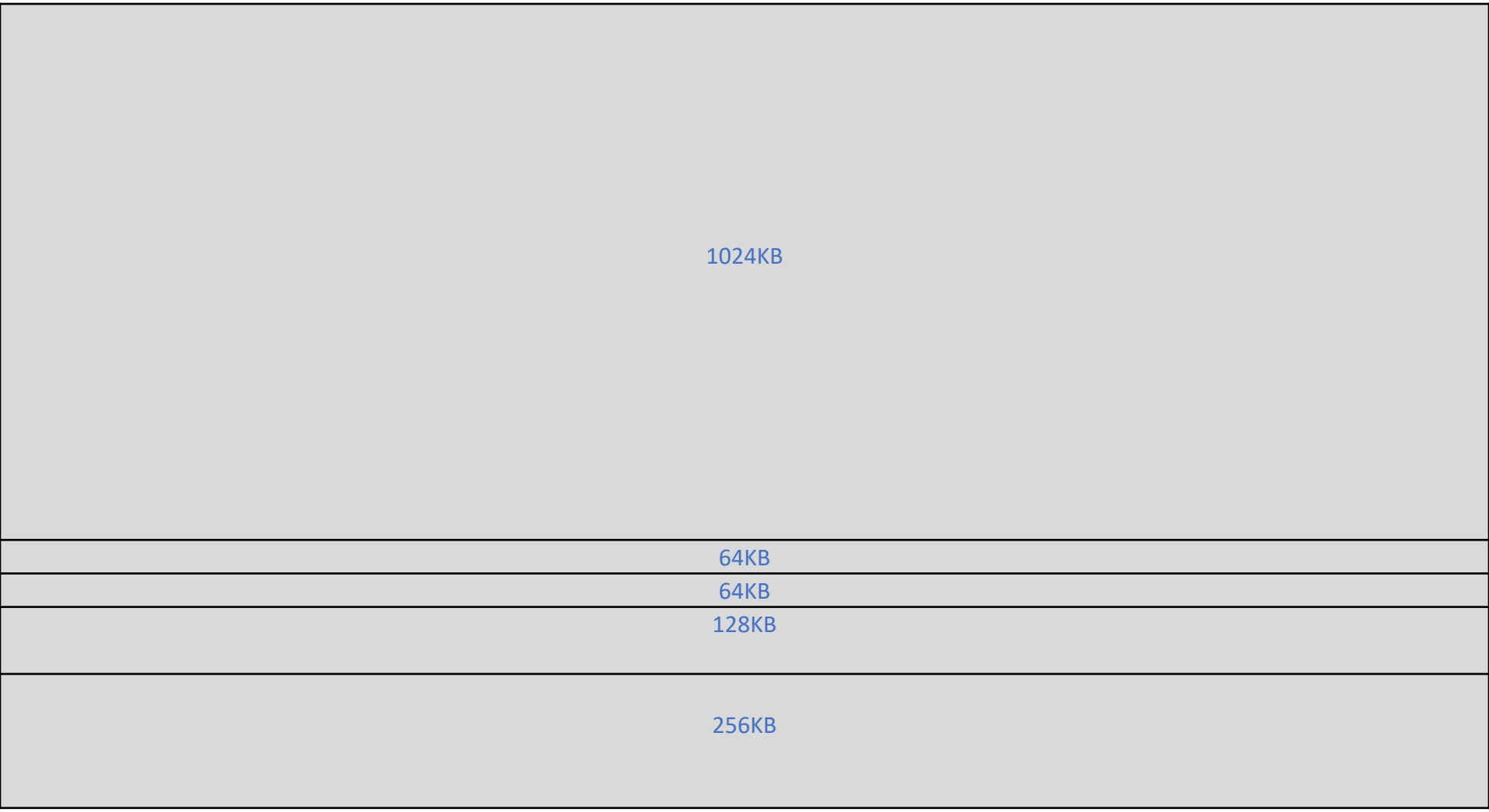
```
X1 = __get_free_pages(flags, 6); // get 2^6 = 64KB of memory
```



Phys Feng Shui – Buddy Allocator

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

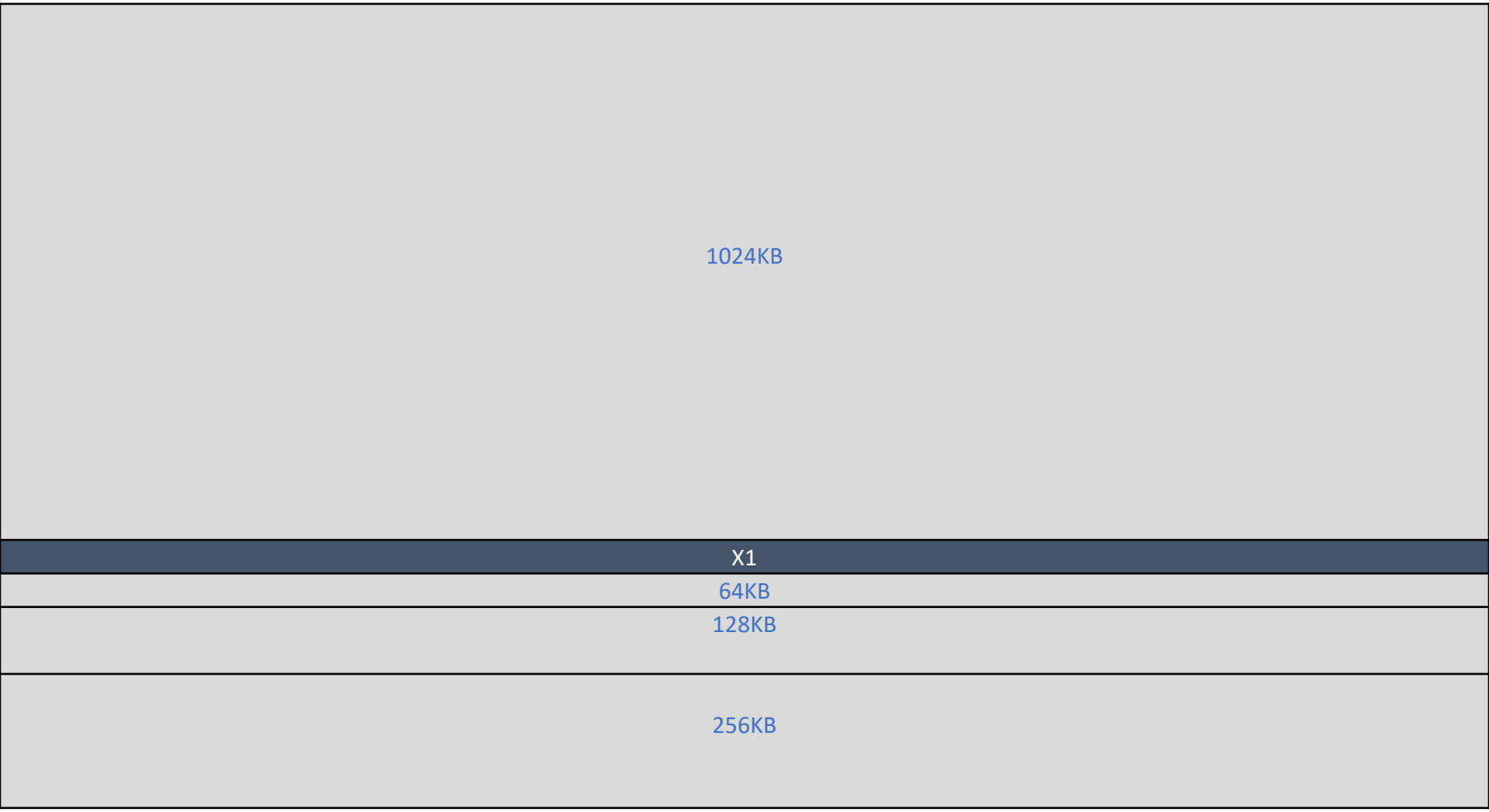
```
X1 = __get_free_pages(flags, 6); // get 2^6 = 64KB of memory
```



Phys Feng Shui – Buddy Allocator

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

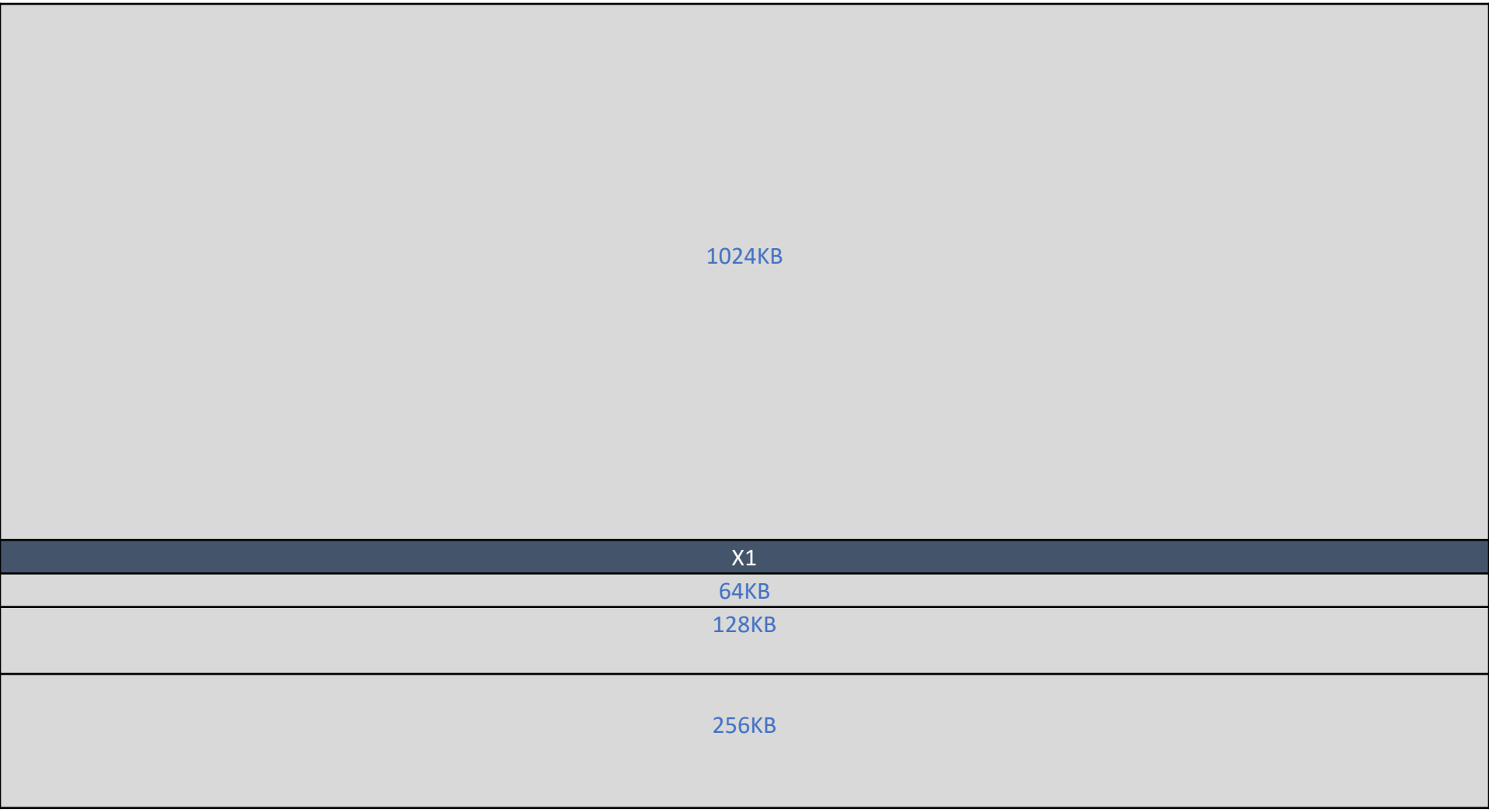
```
X1 = __get_free_pages(flags, 6); // get 2^6 = 64KB of memory
```



Phys Feng Shui – Buddy Allocator

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

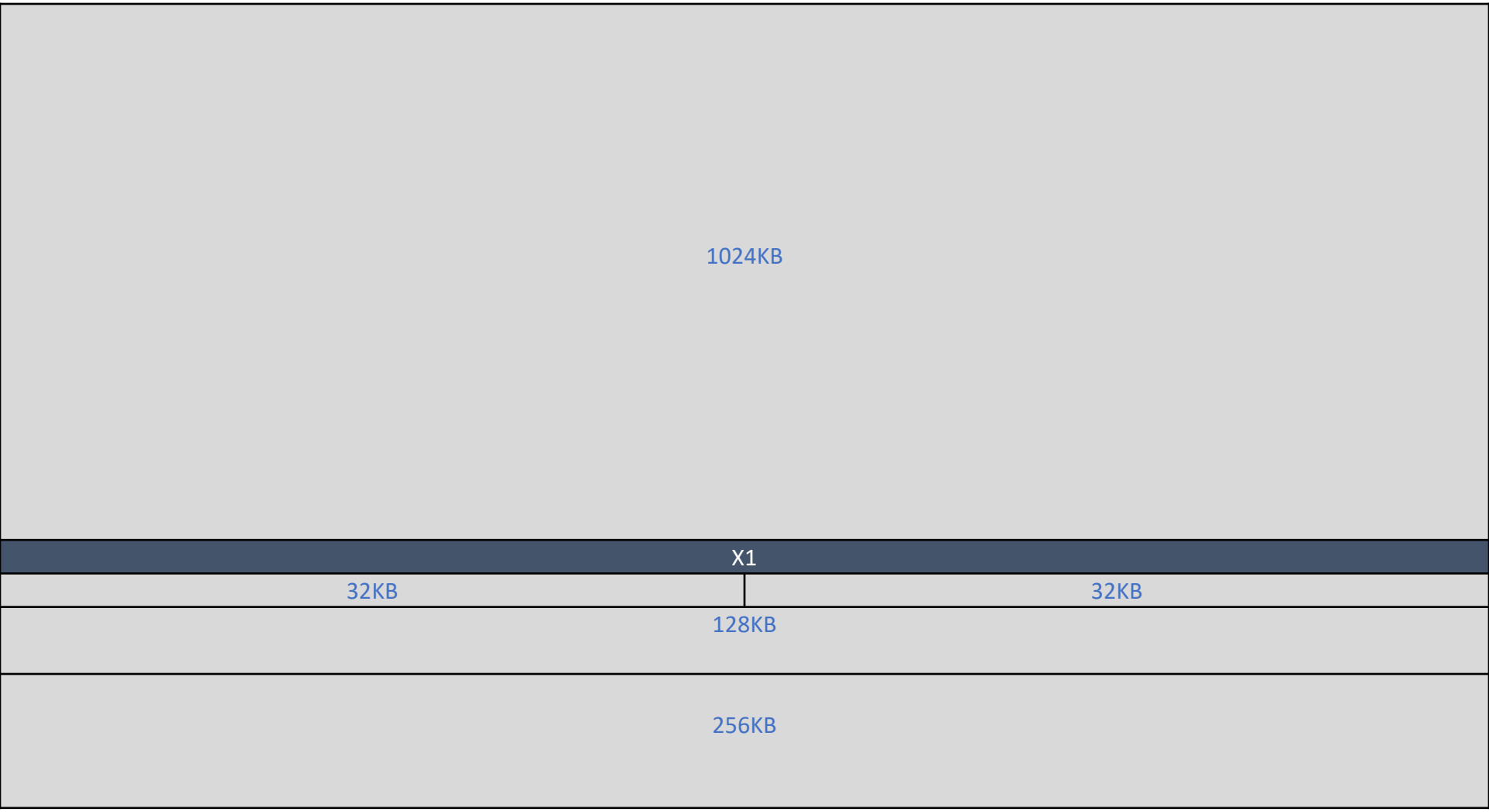
```
X2 = __get_free_pages(flags, 3); // get 2^3 = 8KB of memory
```



Phys Feng Shui – Buddy Allocator

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

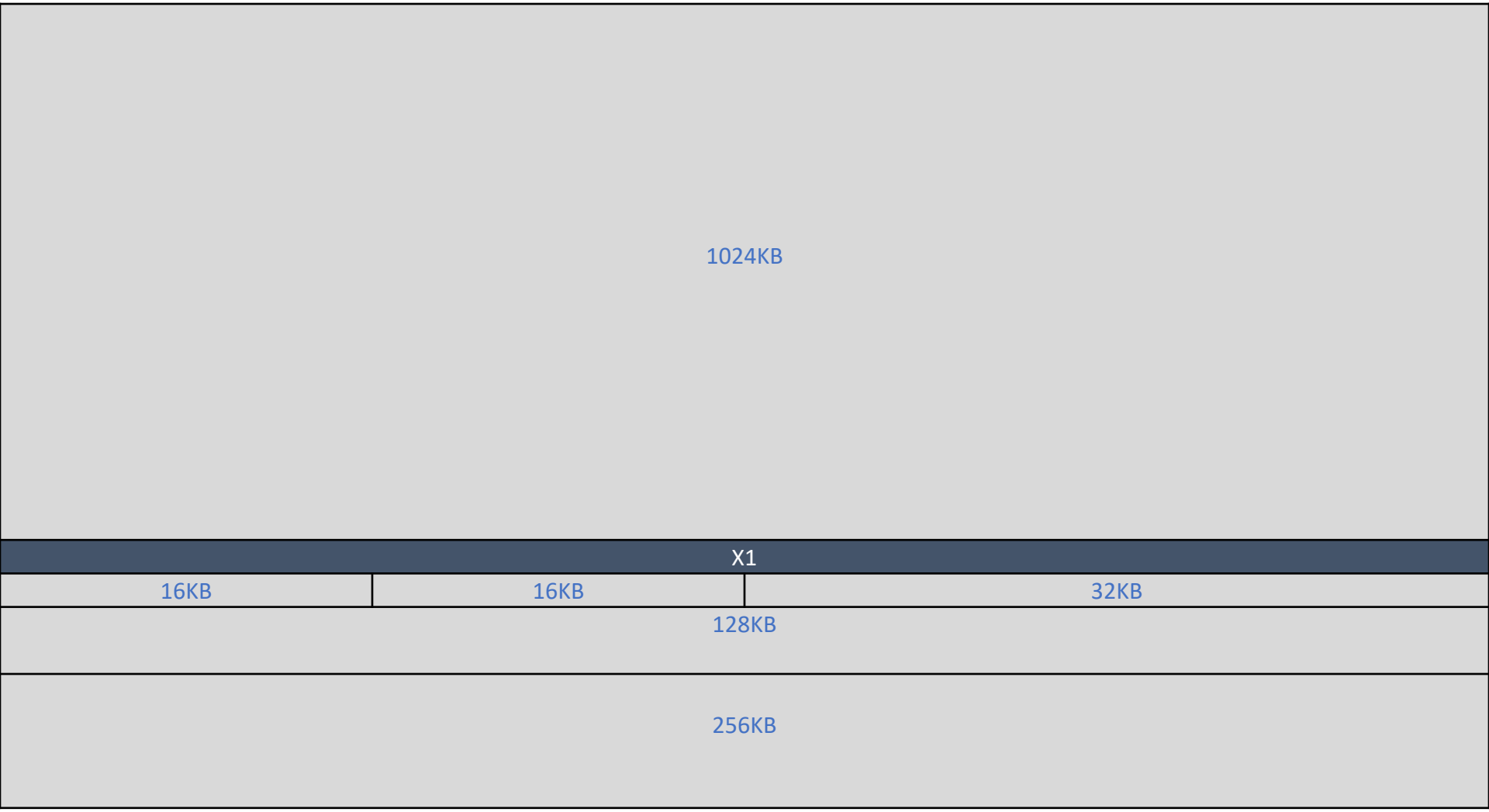
```
X2 = __get_free_pages(flags, 3); // get 2^3 = 8KB of memory
```



Phys Feng Shui – Buddy Allocator

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

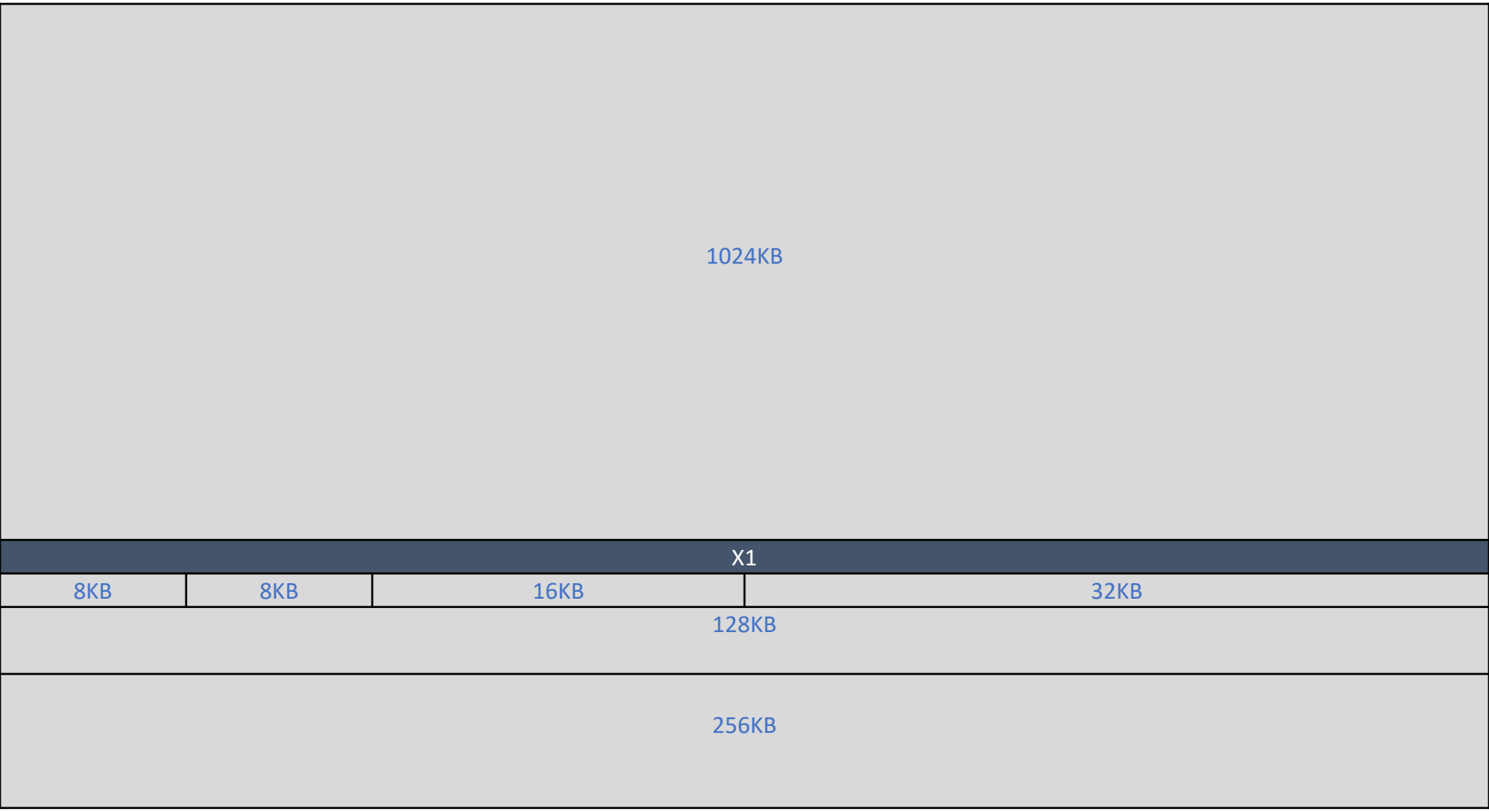
```
X2 = __get_free_pages(flags, 3); // get 2^3 = 8KB of memory
```



Phys Feng Shui – Buddy Allocator

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

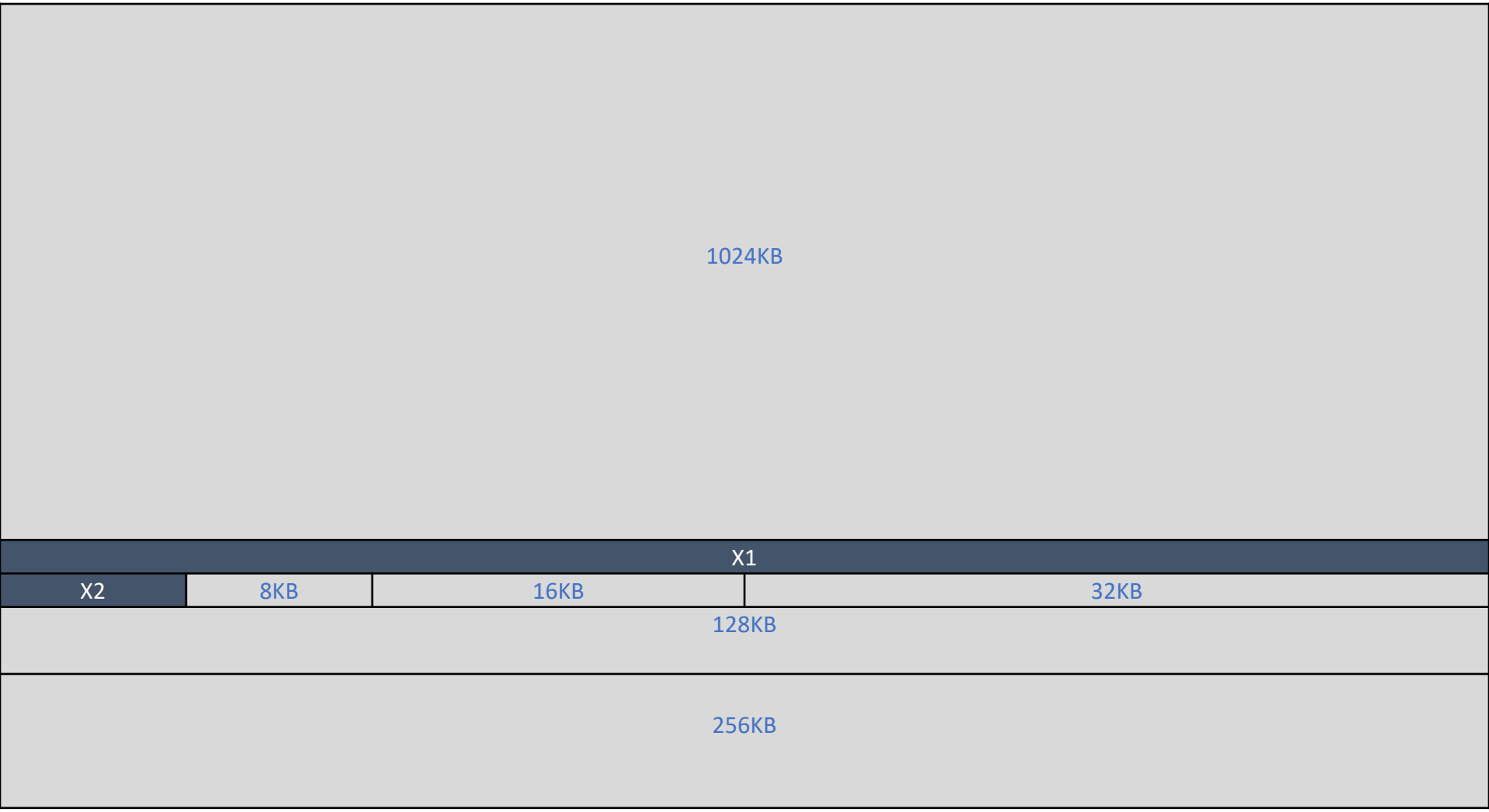
```
X2 = __get_free_pages(flags, 3); // get 2^3 = 8KB of memory
```



Phys Feng Shui – Buddy Allocator

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

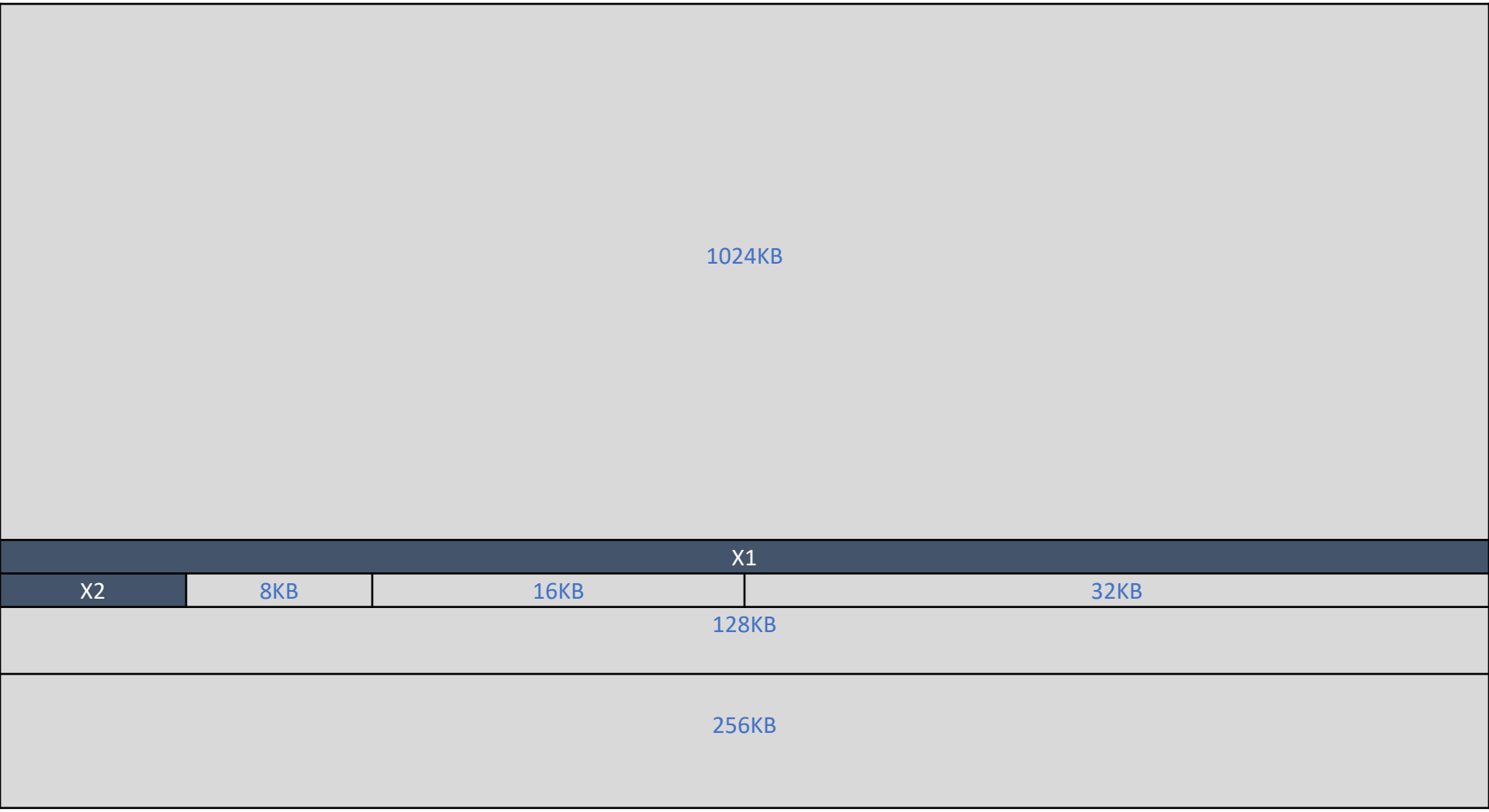
```
X2 = __get_free_pages(flags, 3); // get 2^3 = 8KB of memory
```



Phys Feng Shui – Buddy Allocator

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

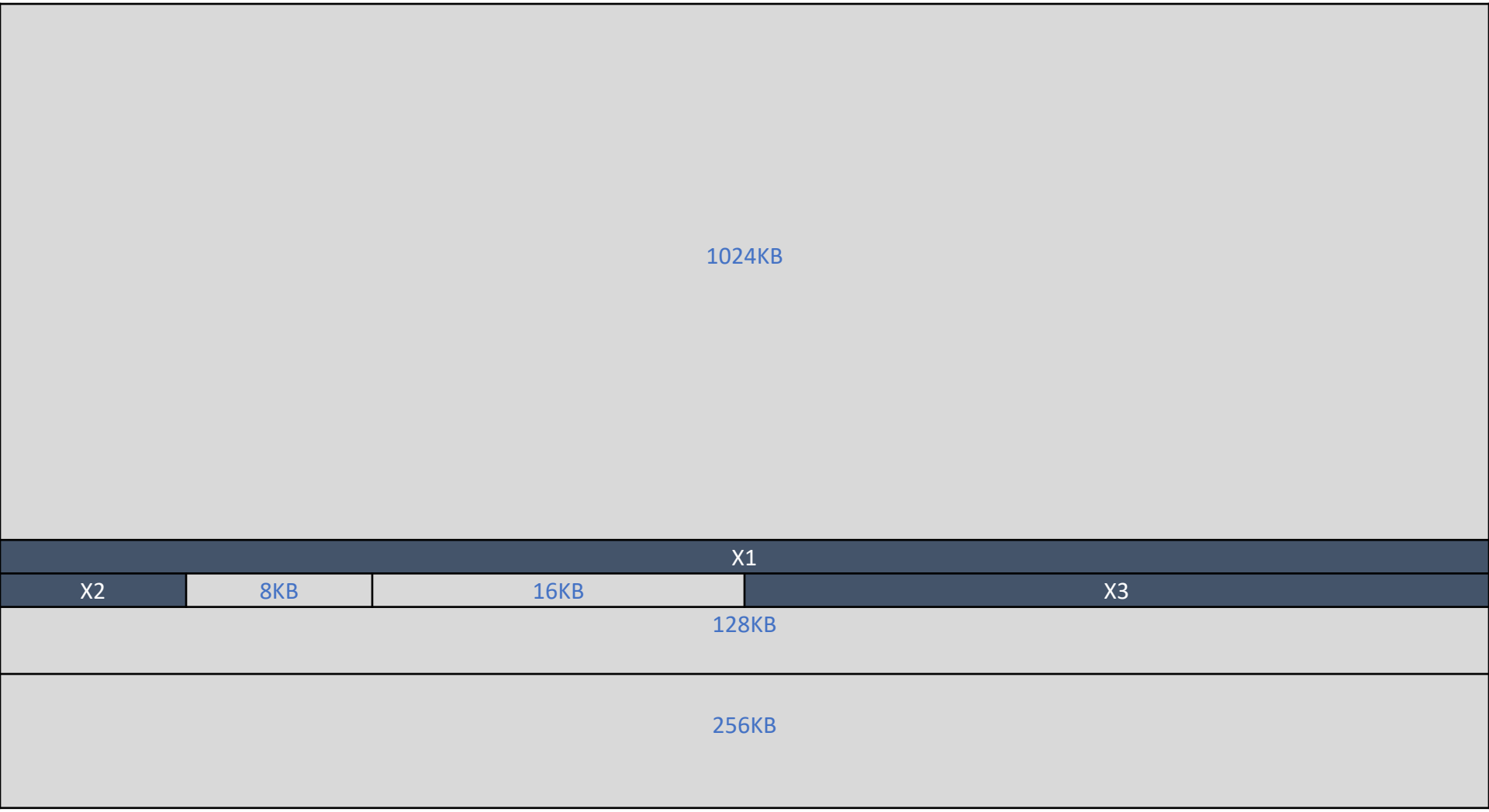
```
X3 = __get_free_pages(flags, 5); // get 2^3 = 32KB of memory
```



Phys Feng Shui – Buddy Allocator

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

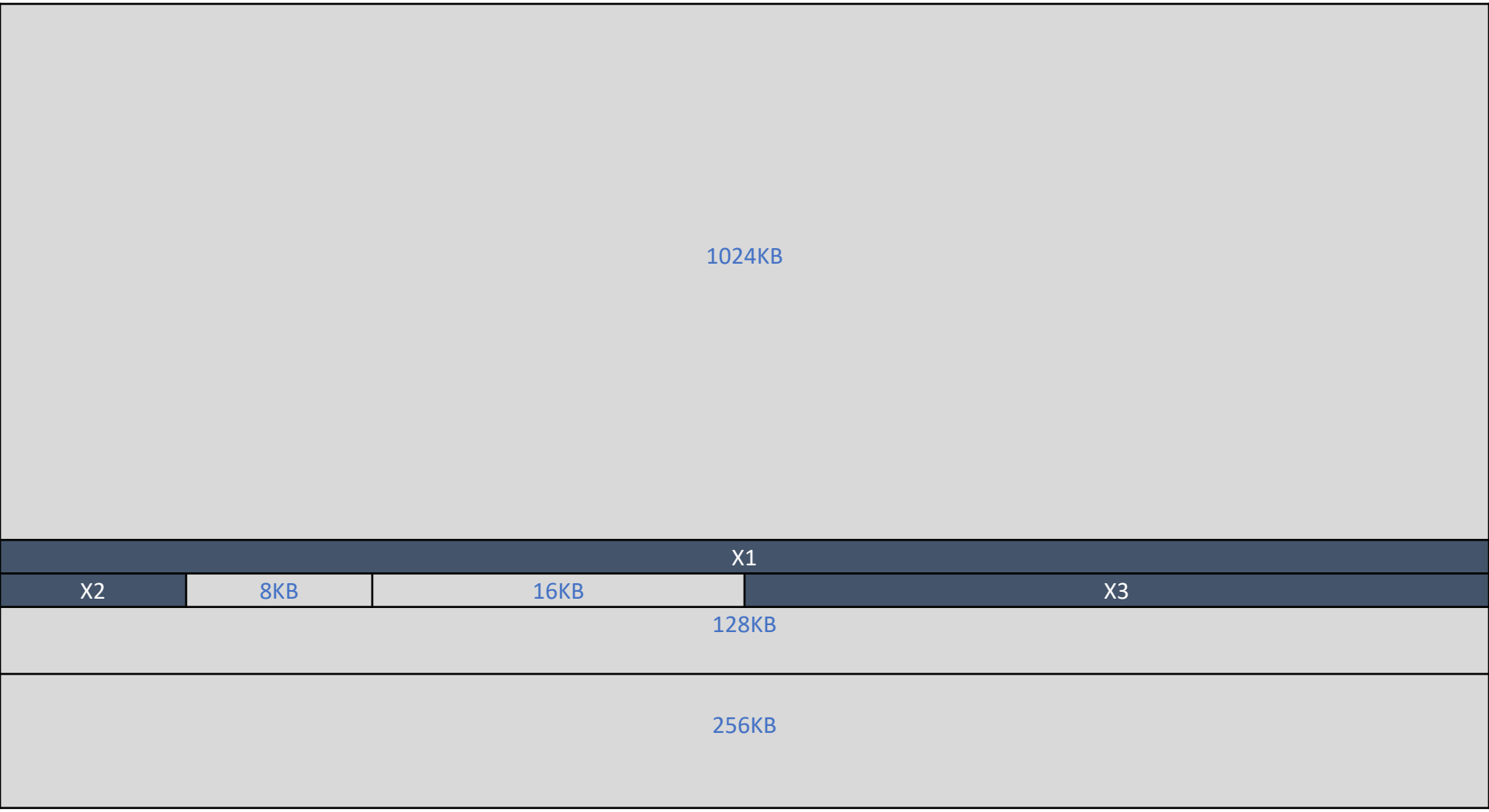
```
P3 = __get_free_pages(flags, 5); // get 2^3 = 32KB of memory
```



Phys Feng Shui – Buddy Allocator

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

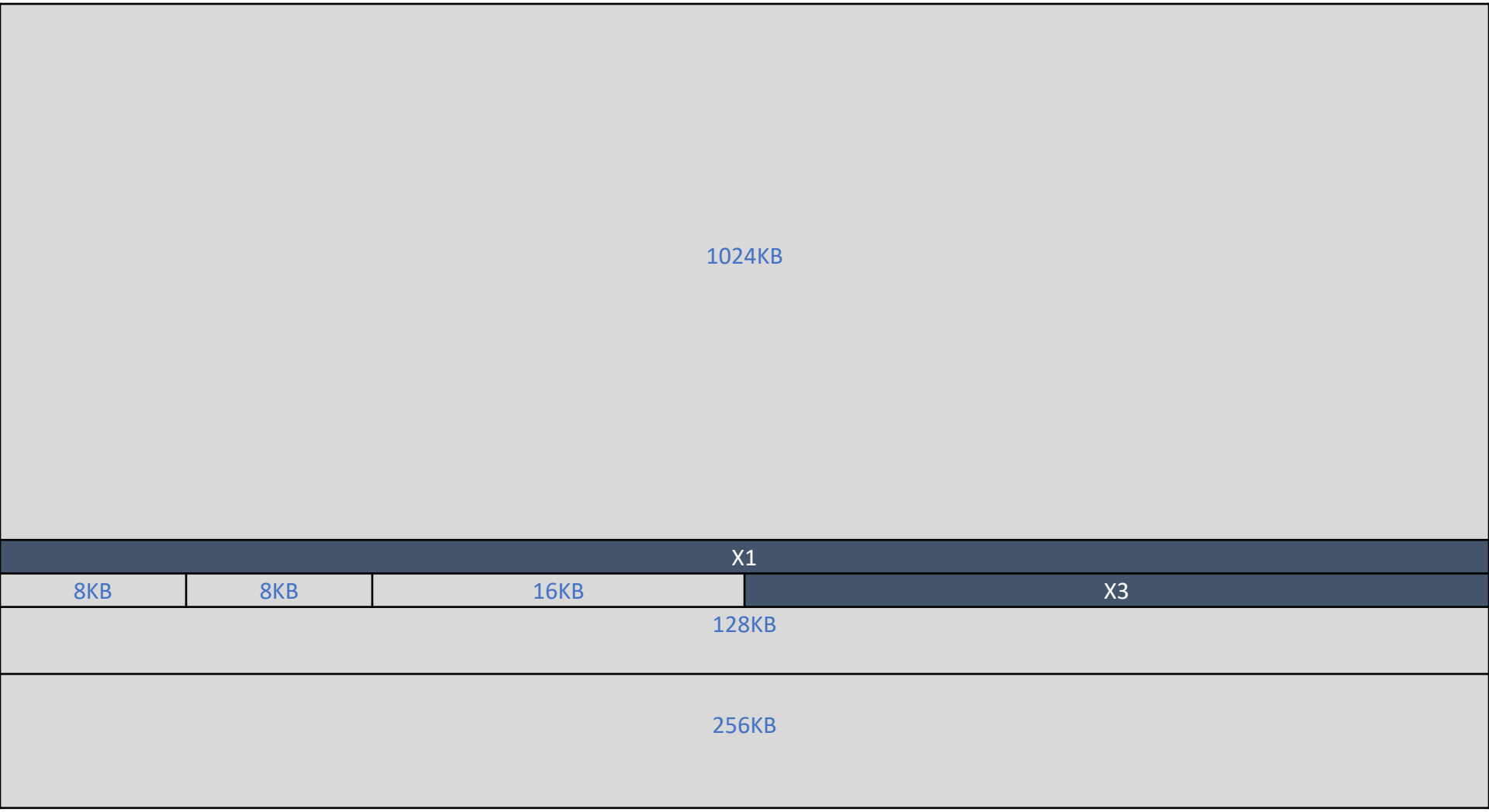
```
free_pages(X2, 3); // free X2
```



Phys Feng Shui – Buddy Allocator

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

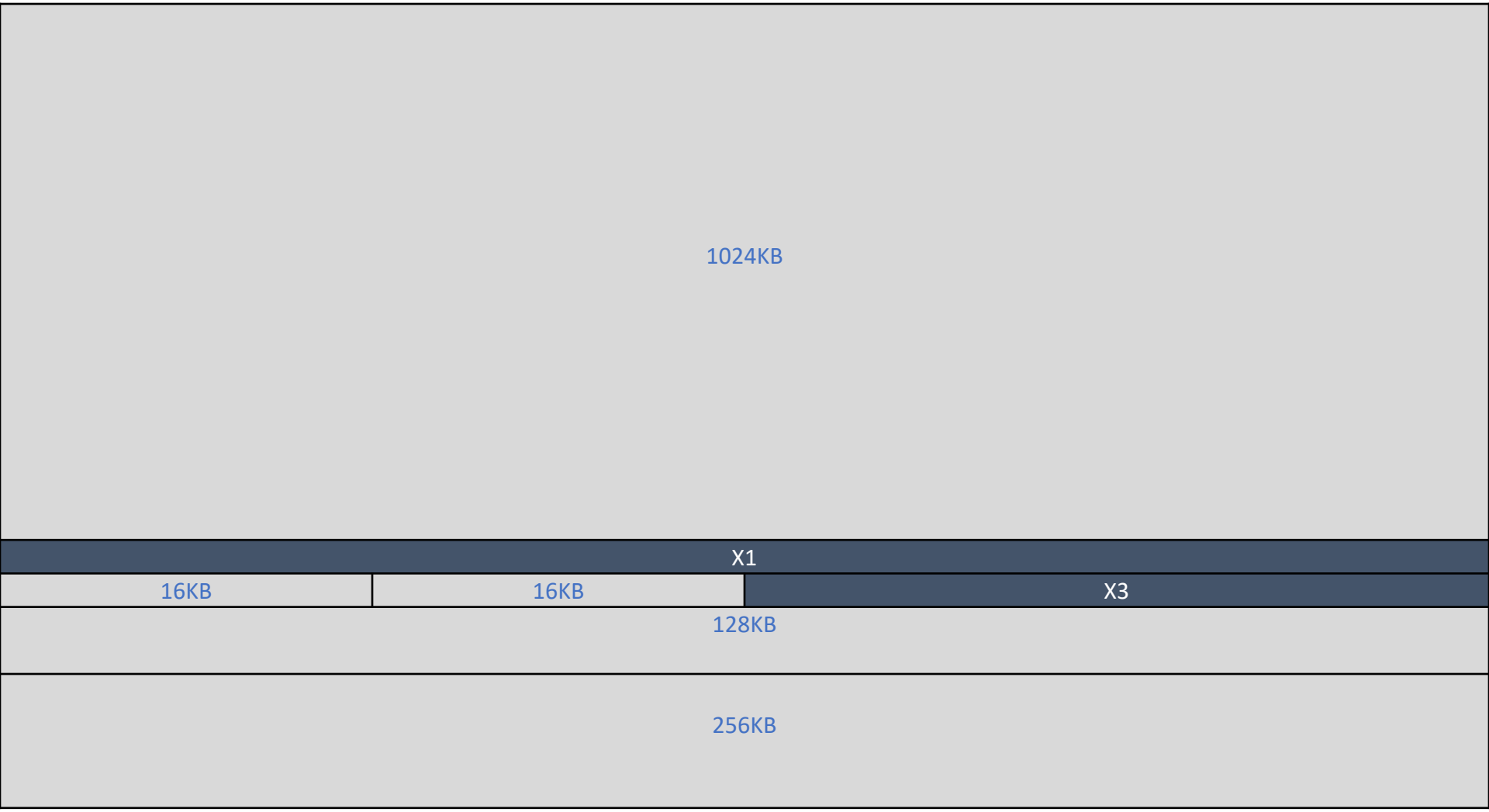
```
free_pages(X2, 3); // free X2
```



Phys Feng Shui – Buddy Allocator

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

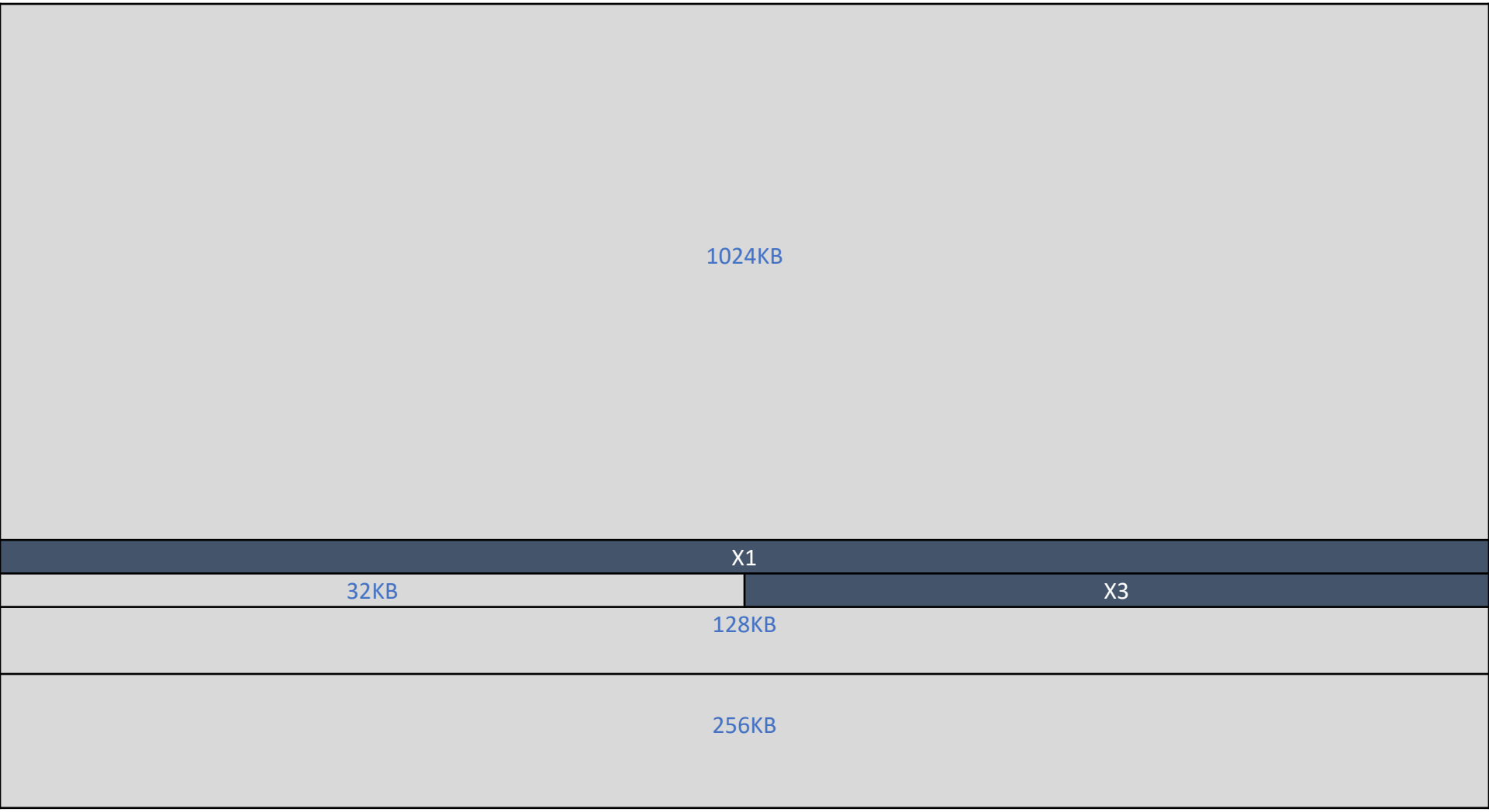
```
free_pages(X2, 3); // free X2
```



Phys Feng Shui – Buddy Allocator

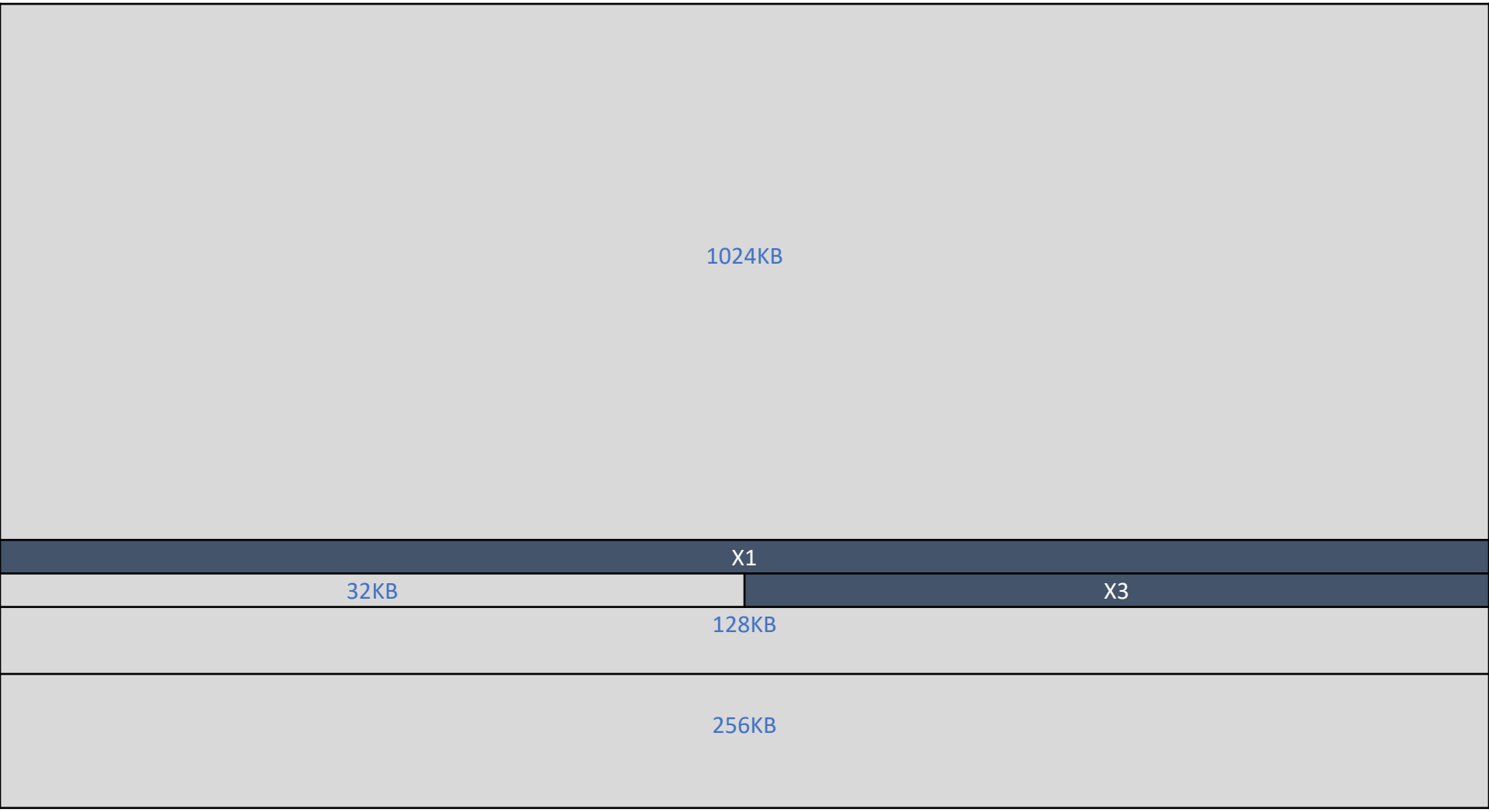
Avoid fragmentation by keeping track of same-size memory chunks (*buddies)*

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



Phys Feng Shui – Buddy Allocator

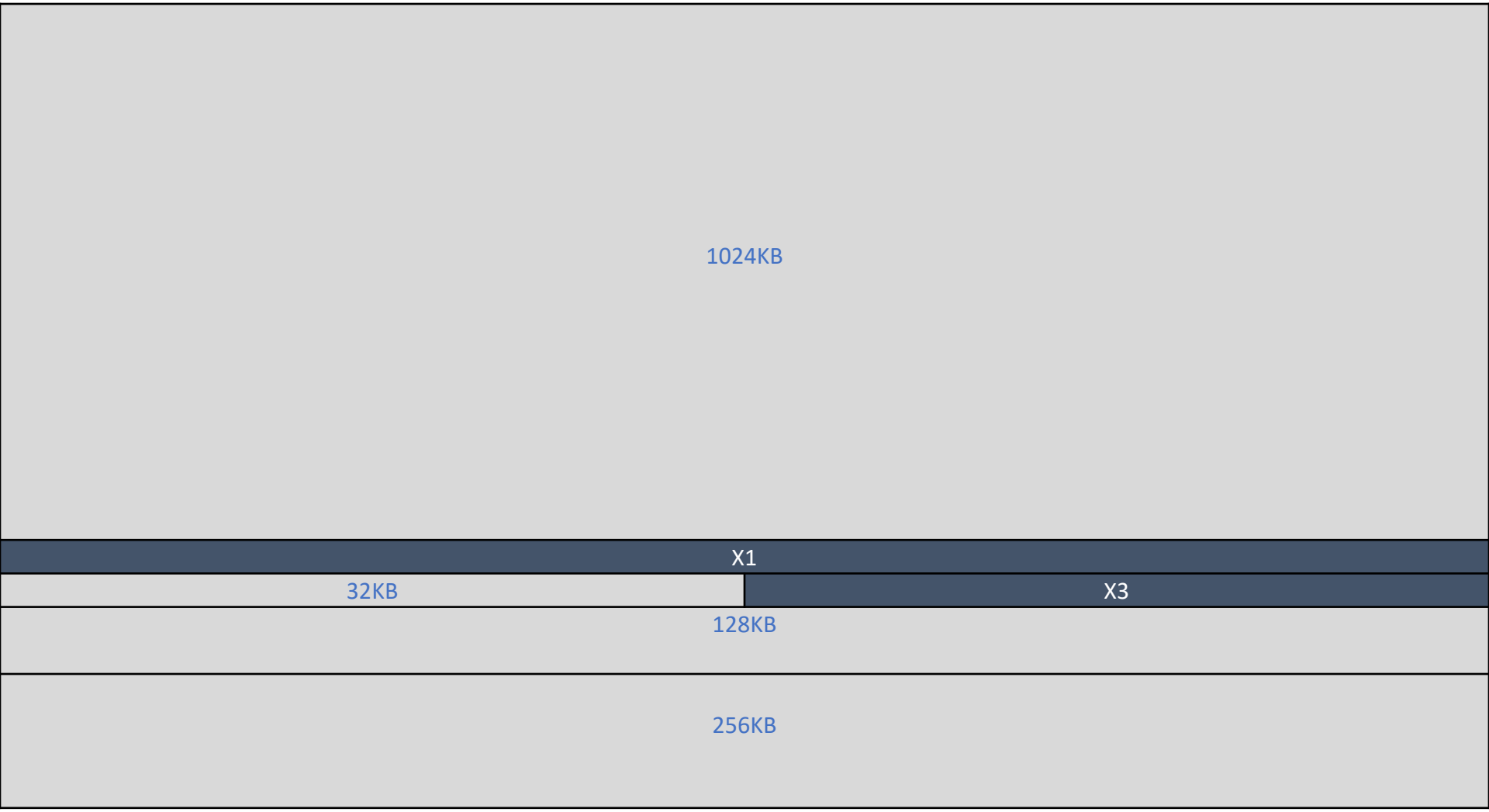
Deterministic Rowhammer exploitation in 8 steps



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

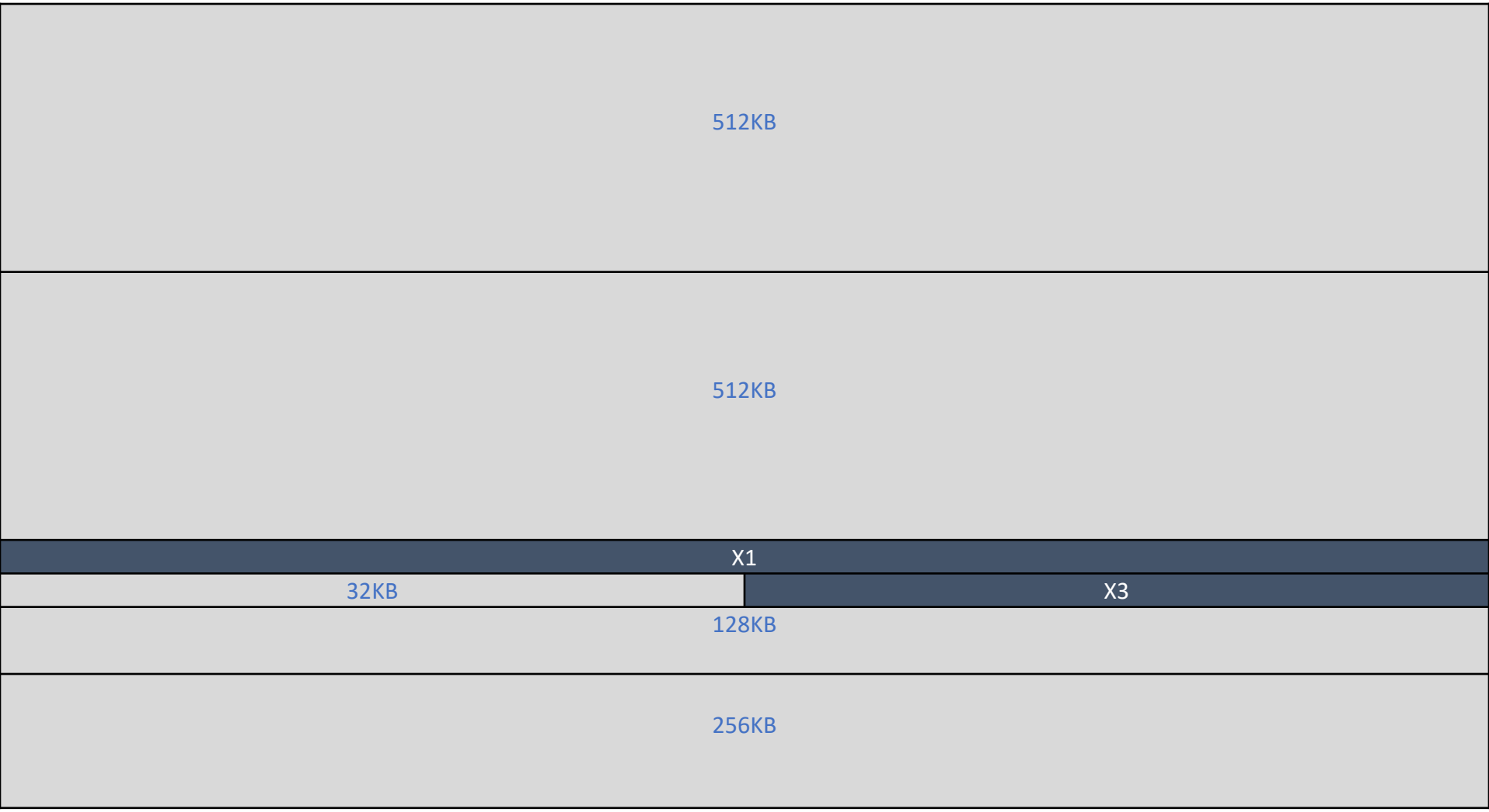
```
L1, L2, …, Ln = exhaust(L);
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

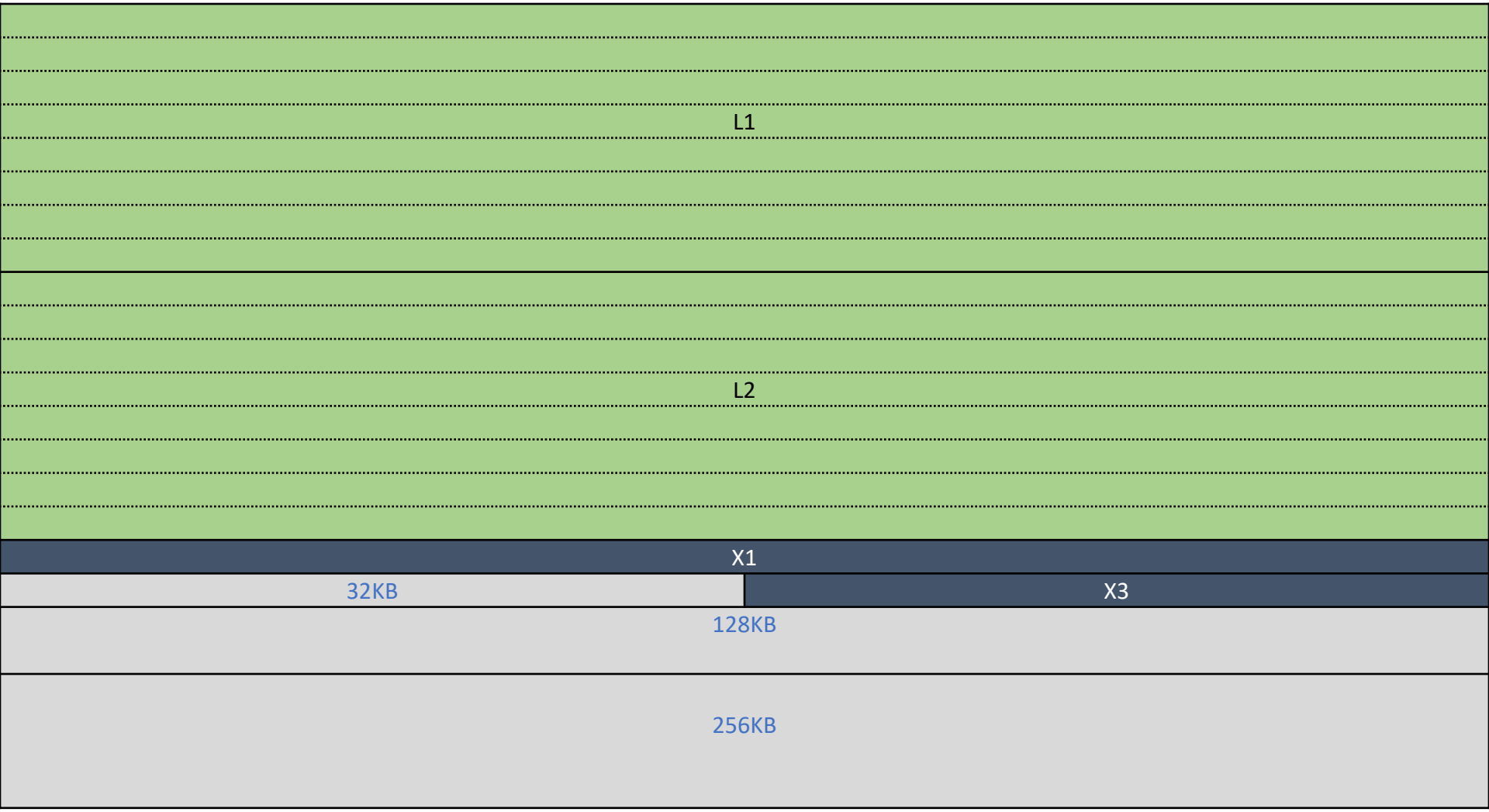
```
L1, L2, …, Ln = exhaust(9); // get all 2^9 = 512KB chunks
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

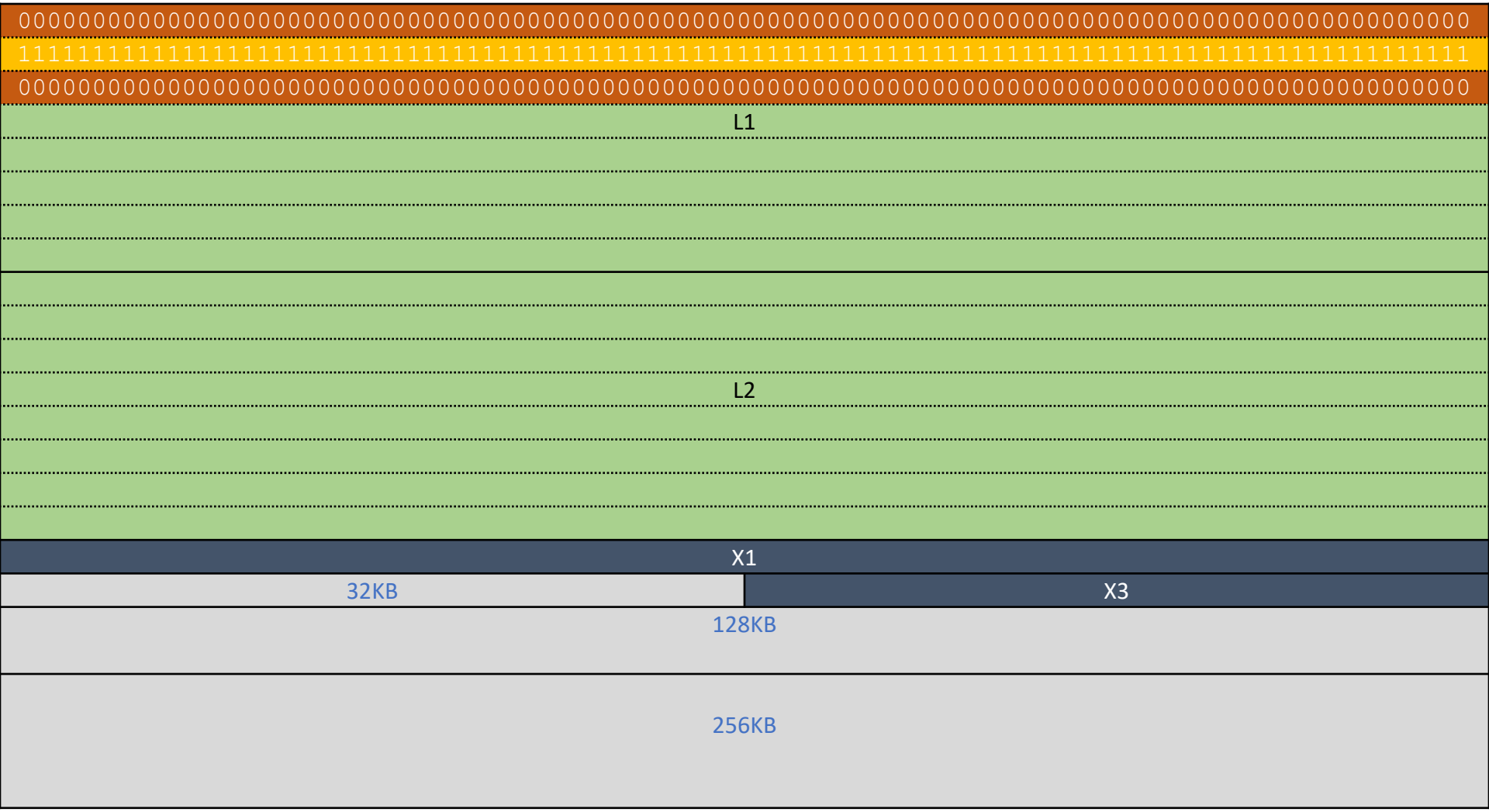
```
L1, L2, …, Ln = exhaust(L); // get all 2^9 = 512KB chunks
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

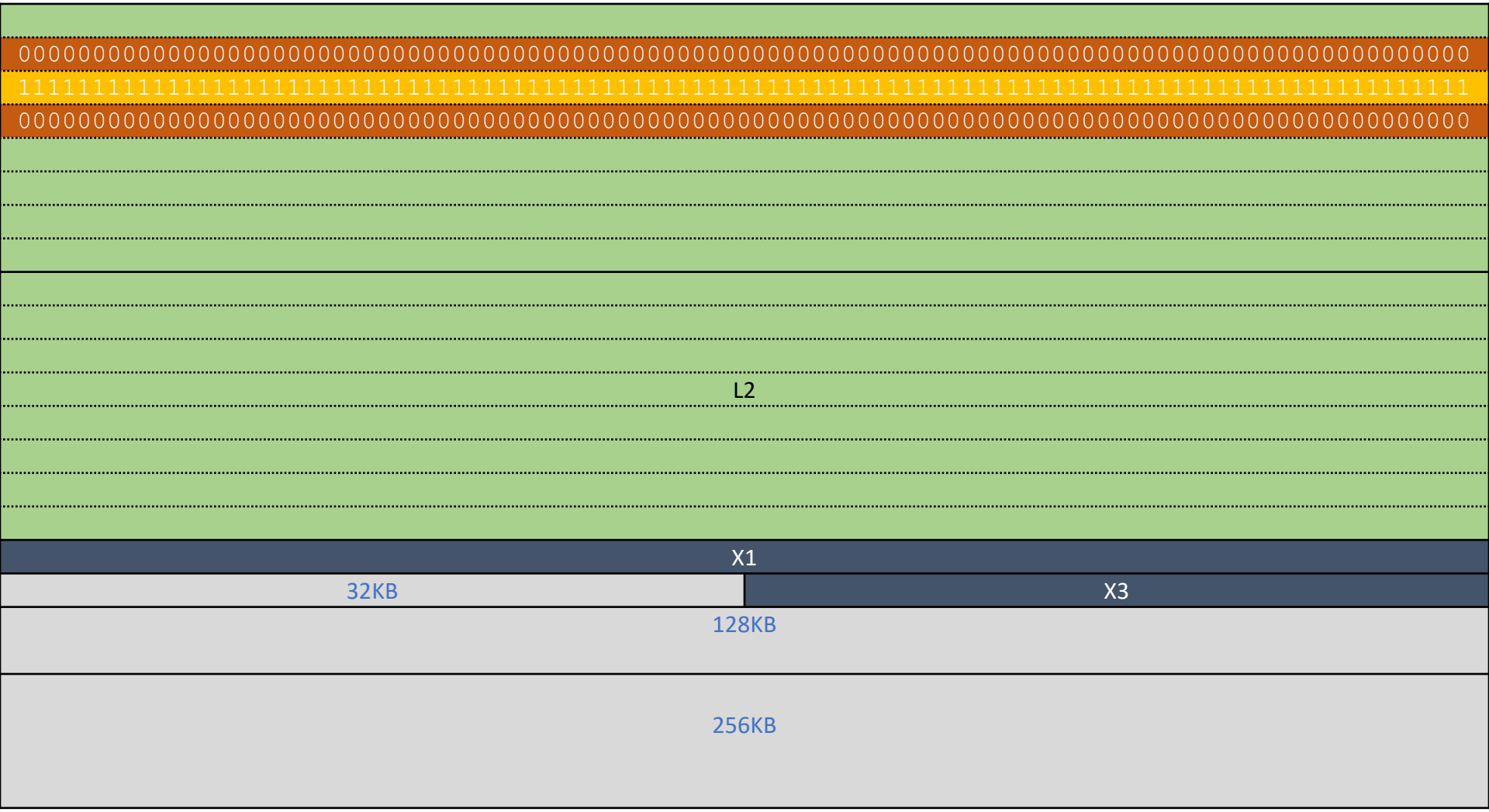
```
Hammer(L1, 2); // hammer row 2 of chunk L1
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

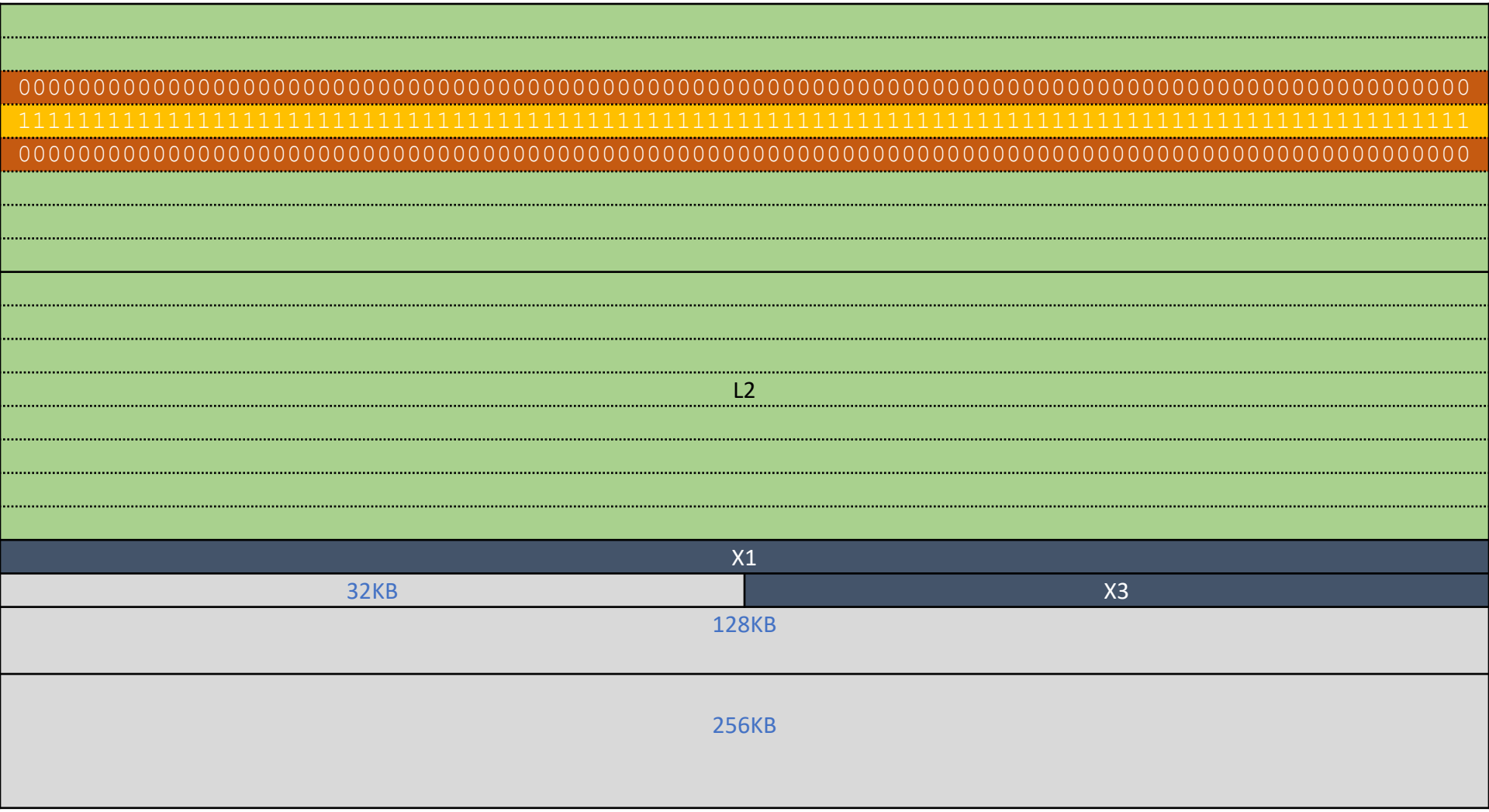
```
Hammer(L1, 3); // hammer row 3 of chunk L1
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

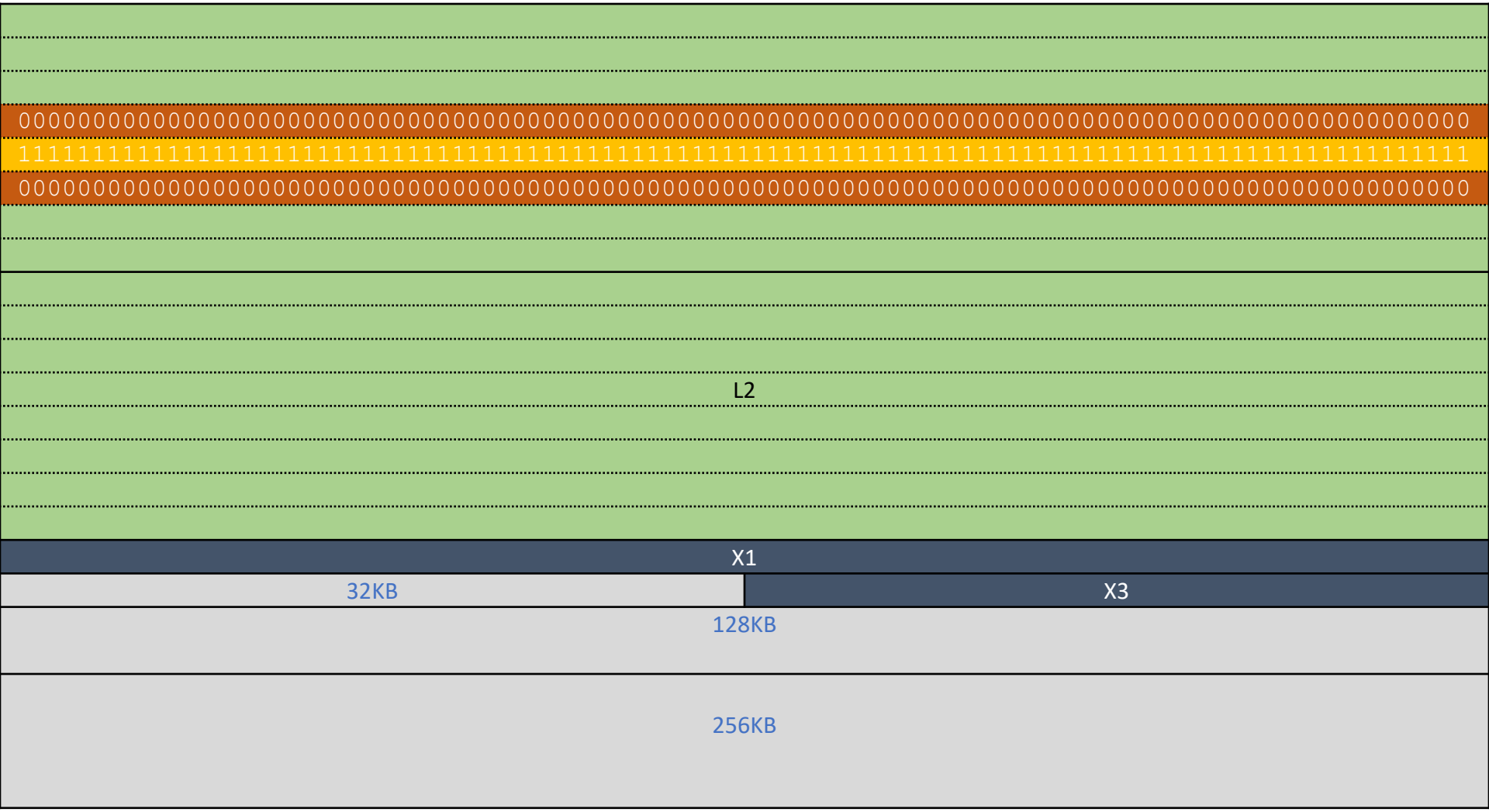
```
Hammer(L1, 4); // hammer row 4 of chunk L1
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

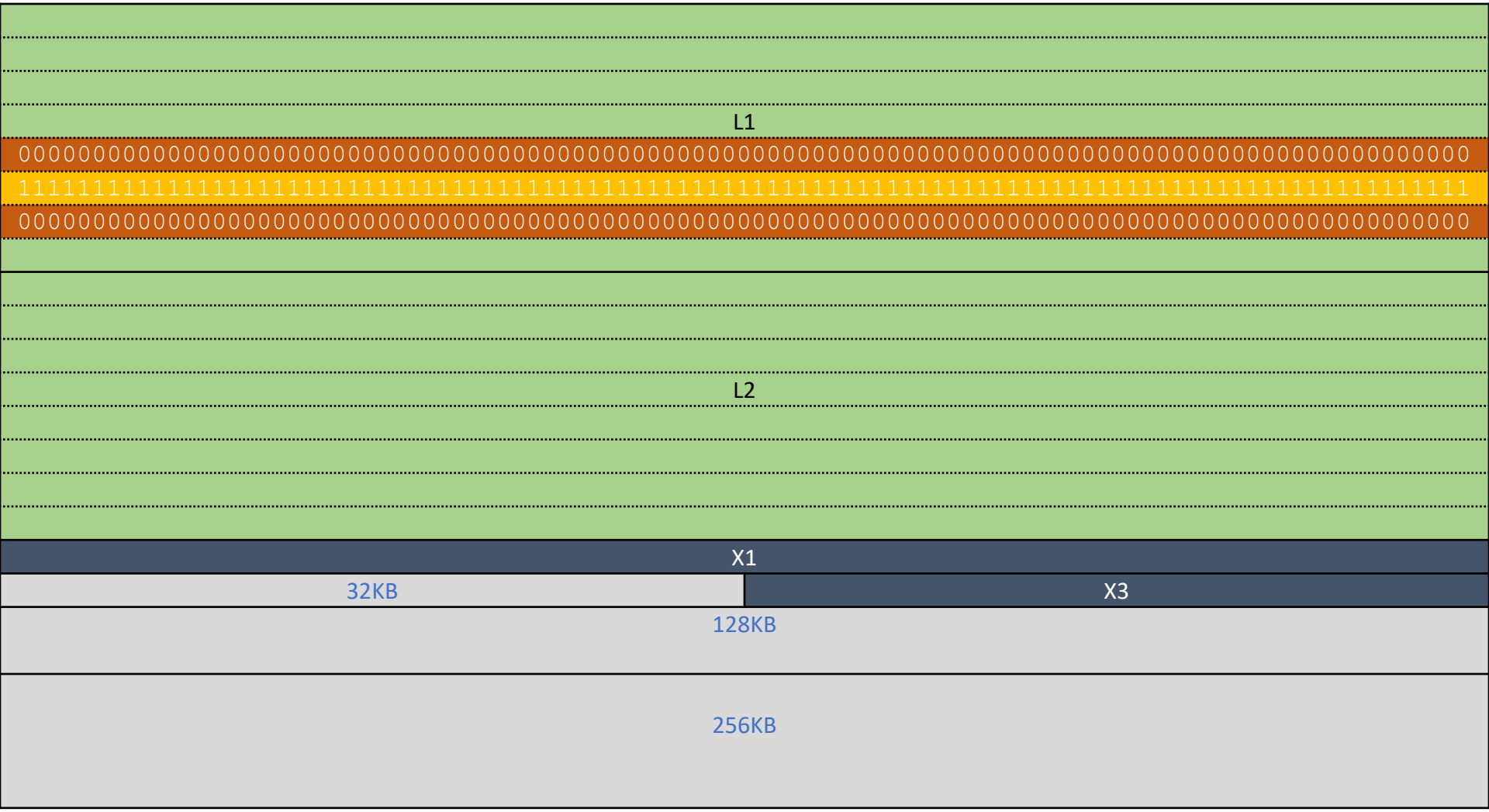
```
Hammer(L1, 5); // hammer row 5 of chunk L1
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

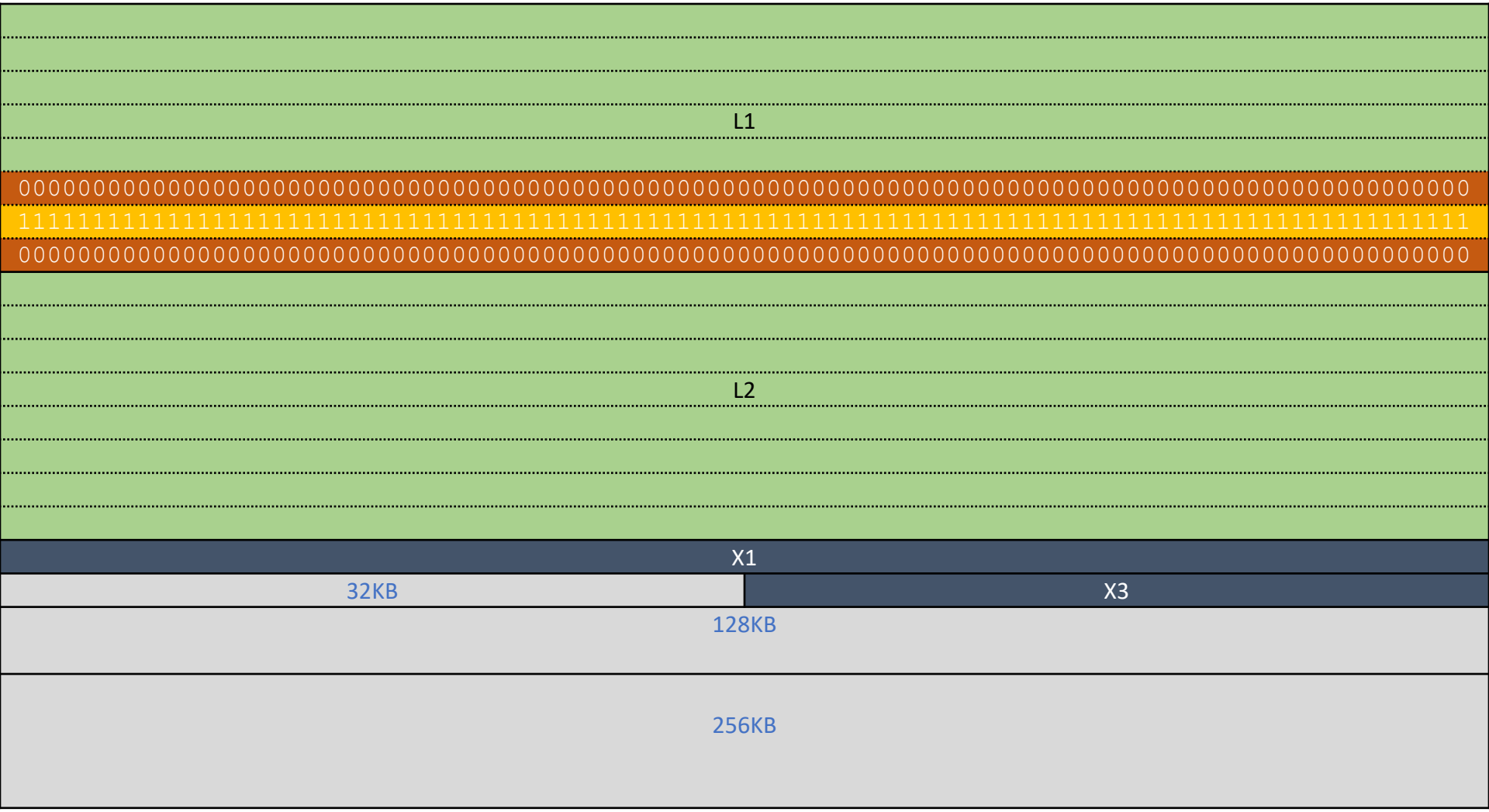
```
Hammer(L1, 6); // hammer row 6 of chunk L1
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

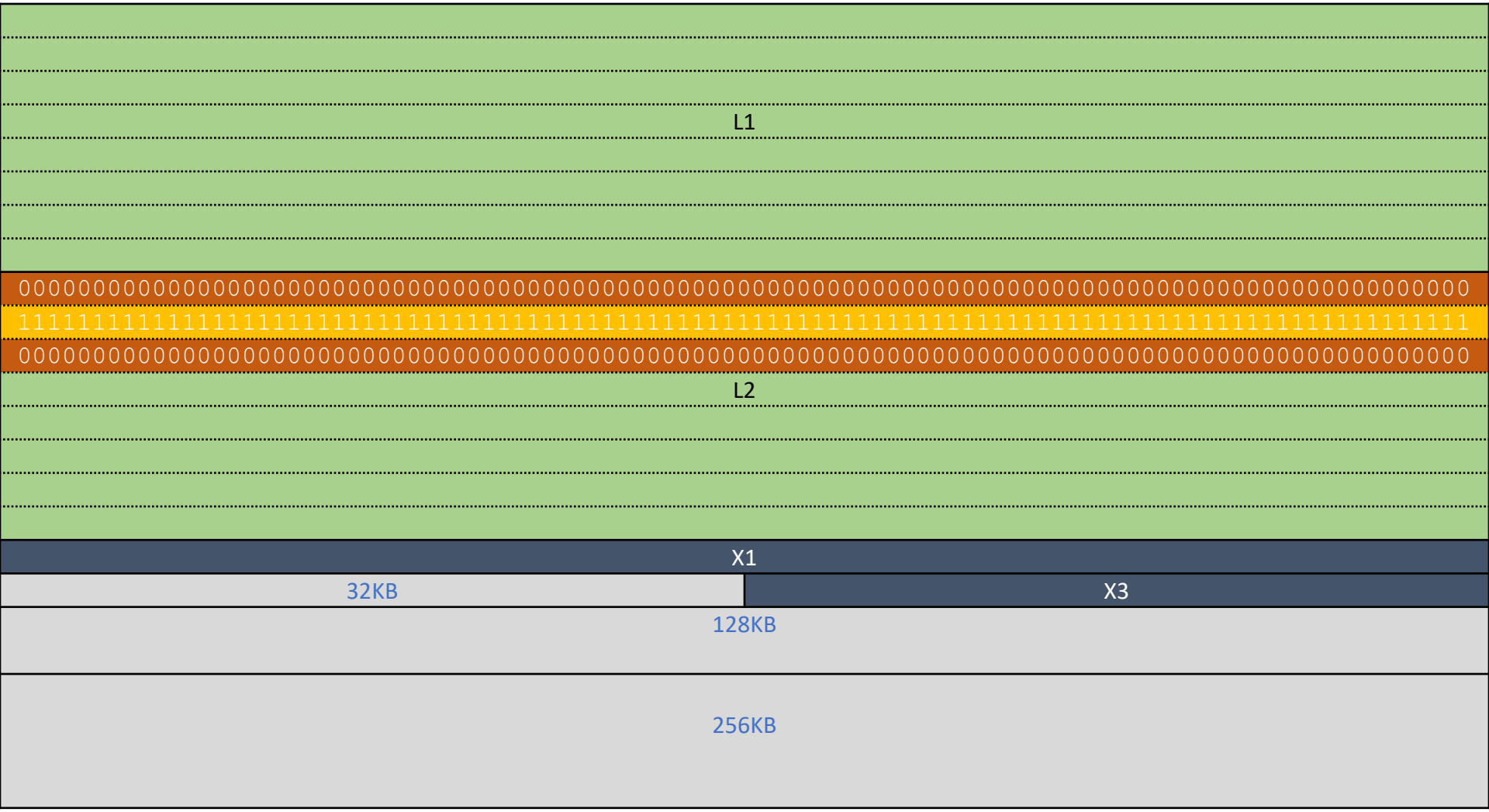
```
Hammer(L1, 7); // hammer row 7 of chunk L1
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

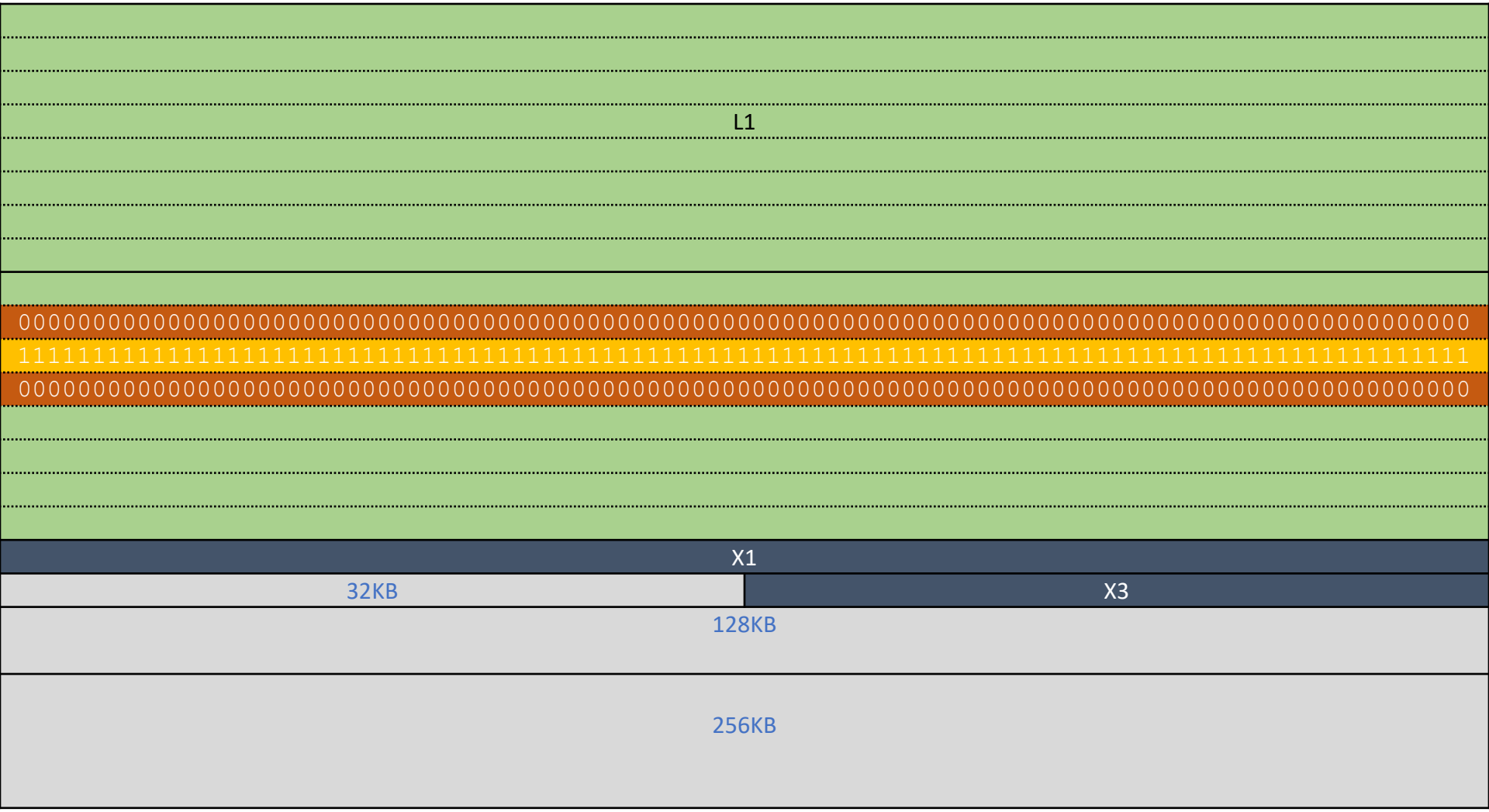
```
Hammer(L2, 2); // hammer row 2 of chunk L2
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

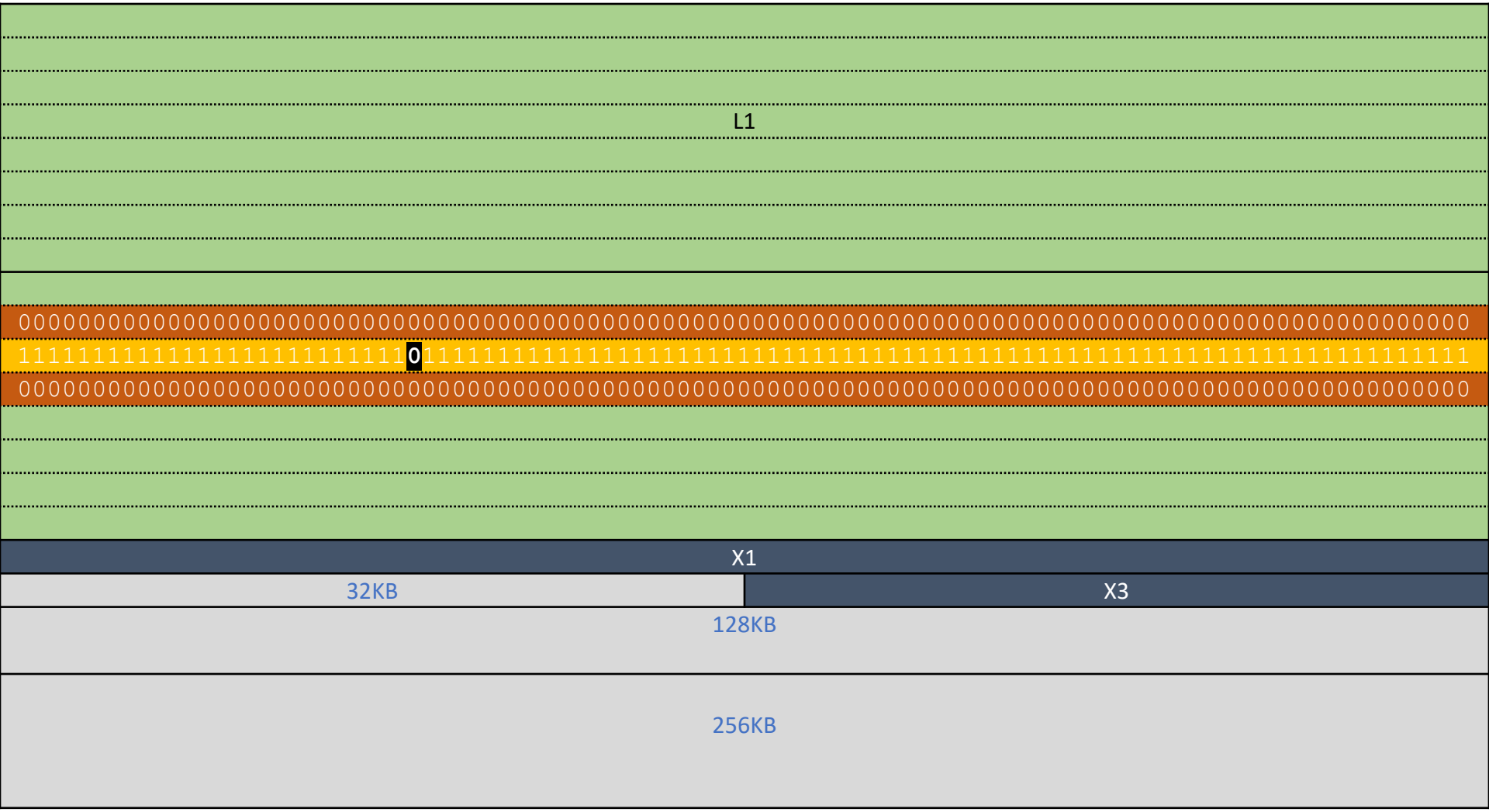
```
Hammer(L2, 3); // hammer row 3 of chunk L2
```



Phys Feng Shui step 1/8

Exhaust + Template *Large* chunks

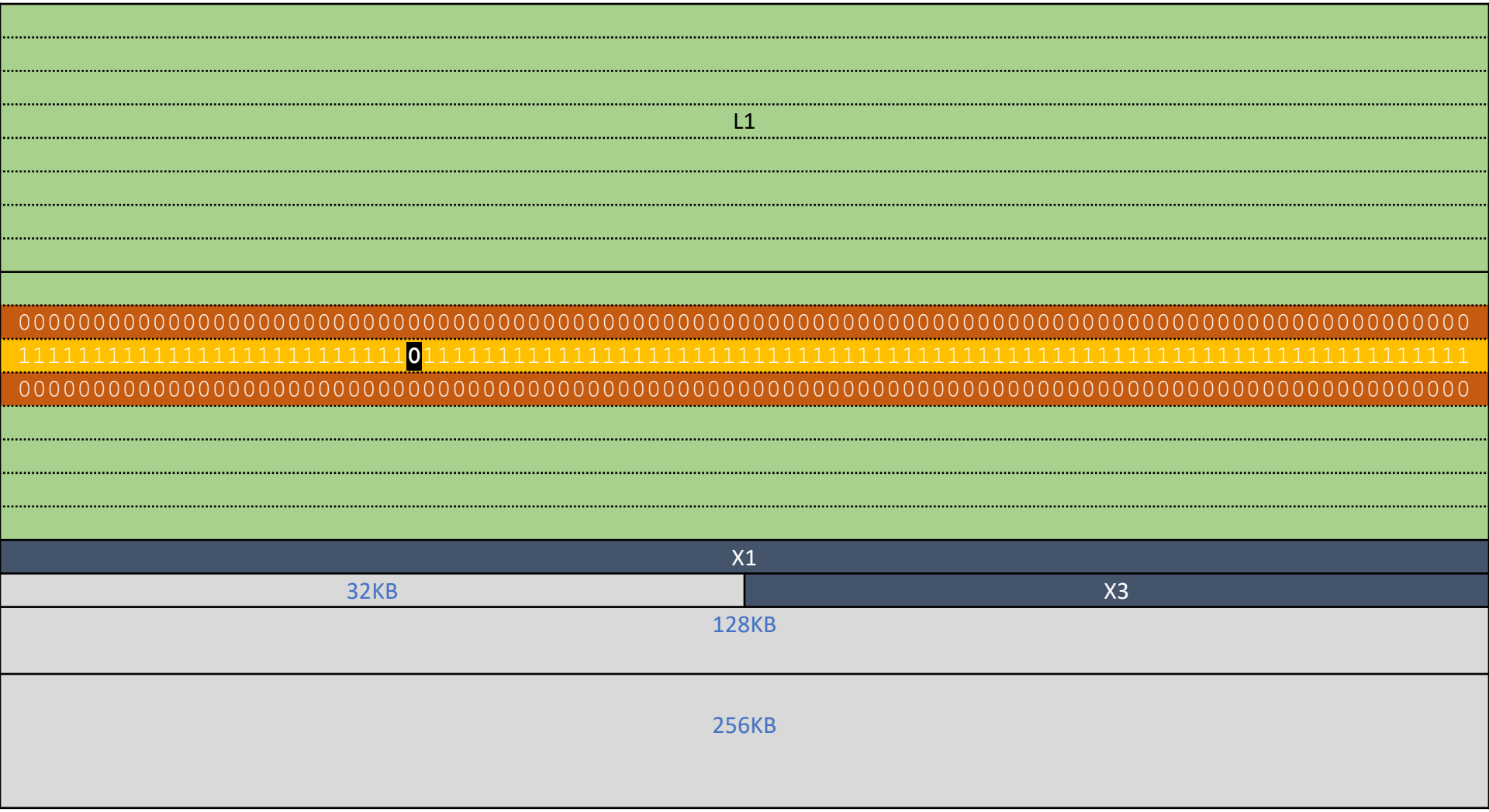
"exploitable flip found in page 5 of virtual row 3 of L2!"



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

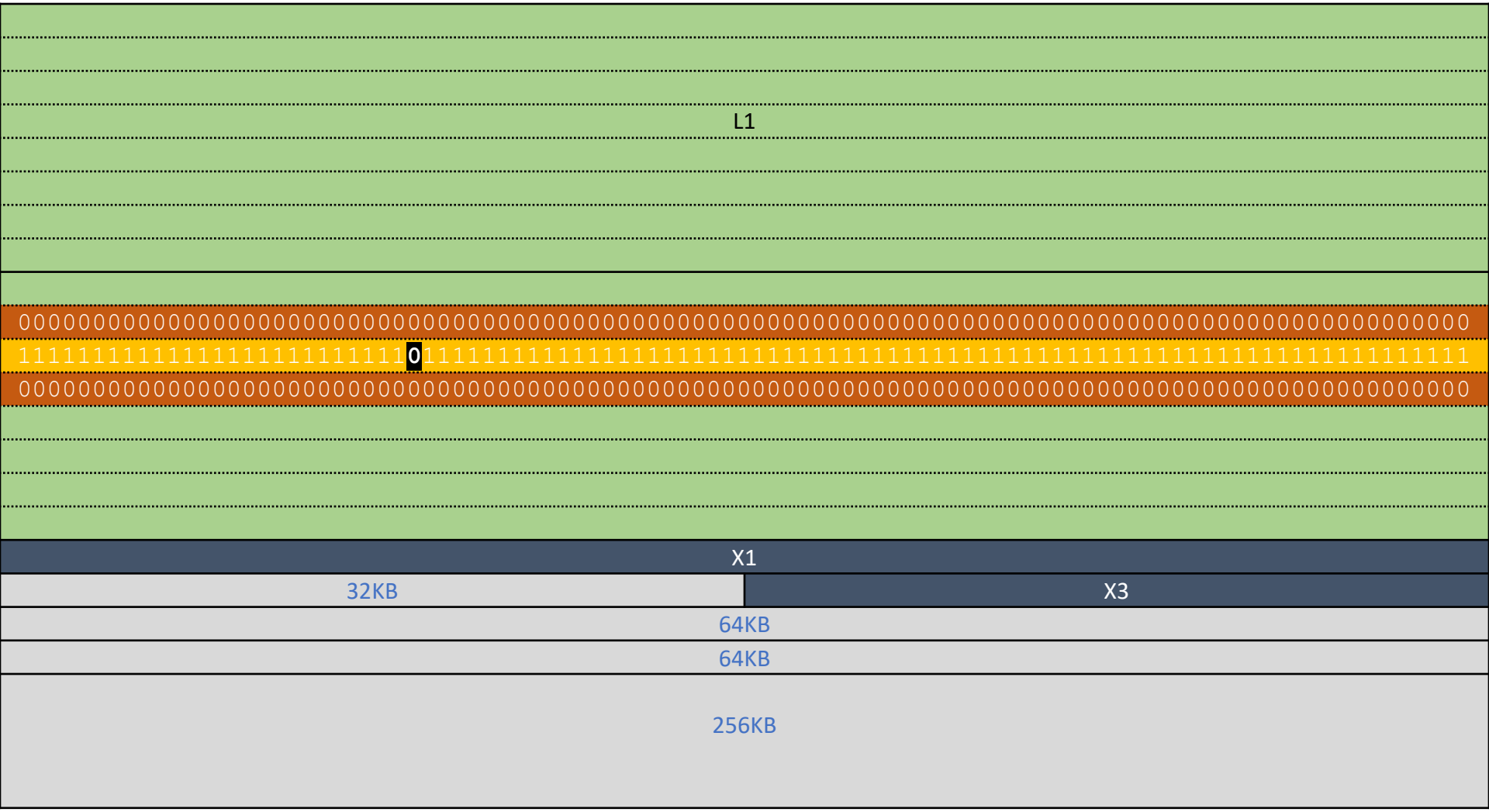
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

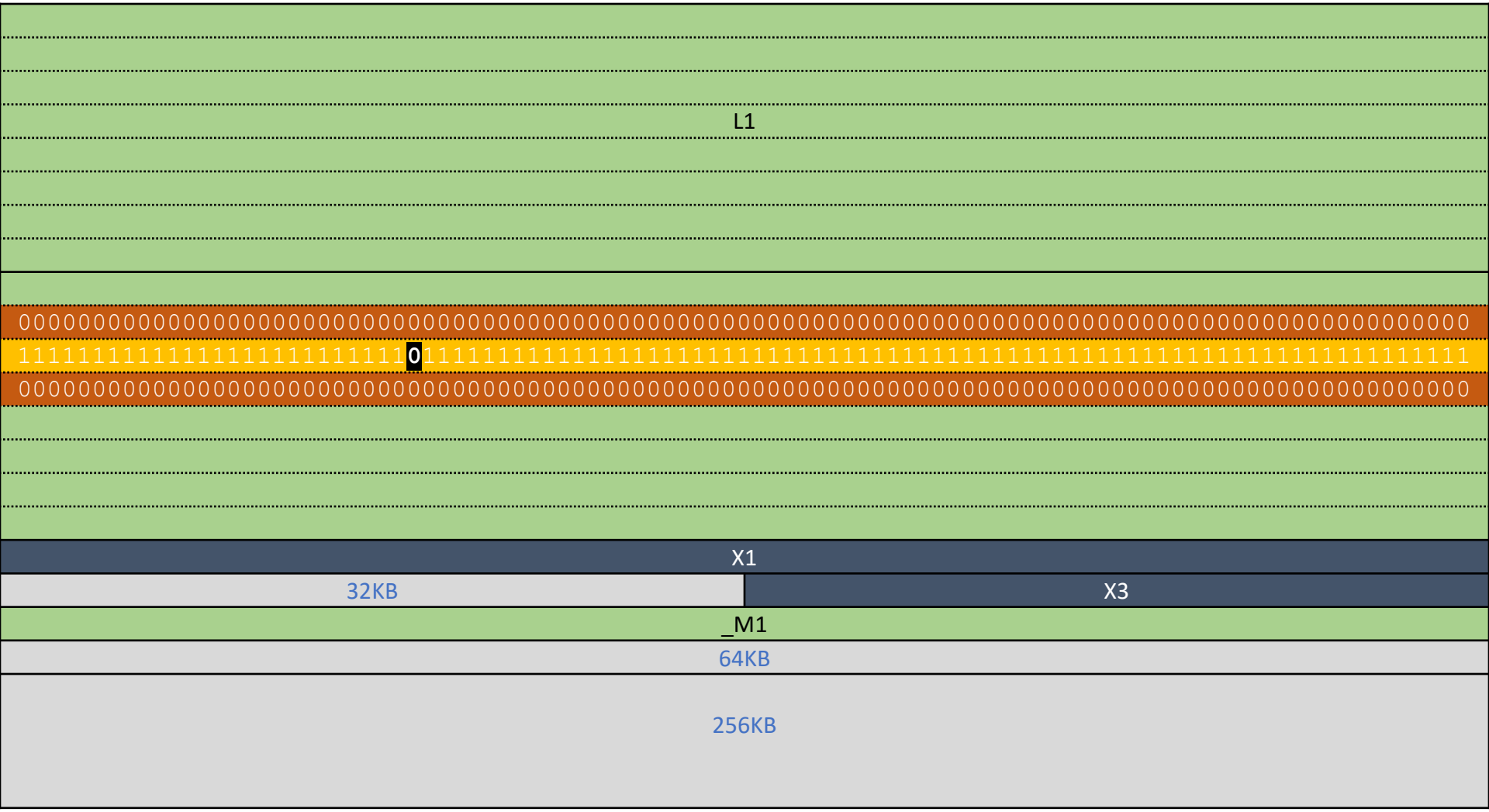
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

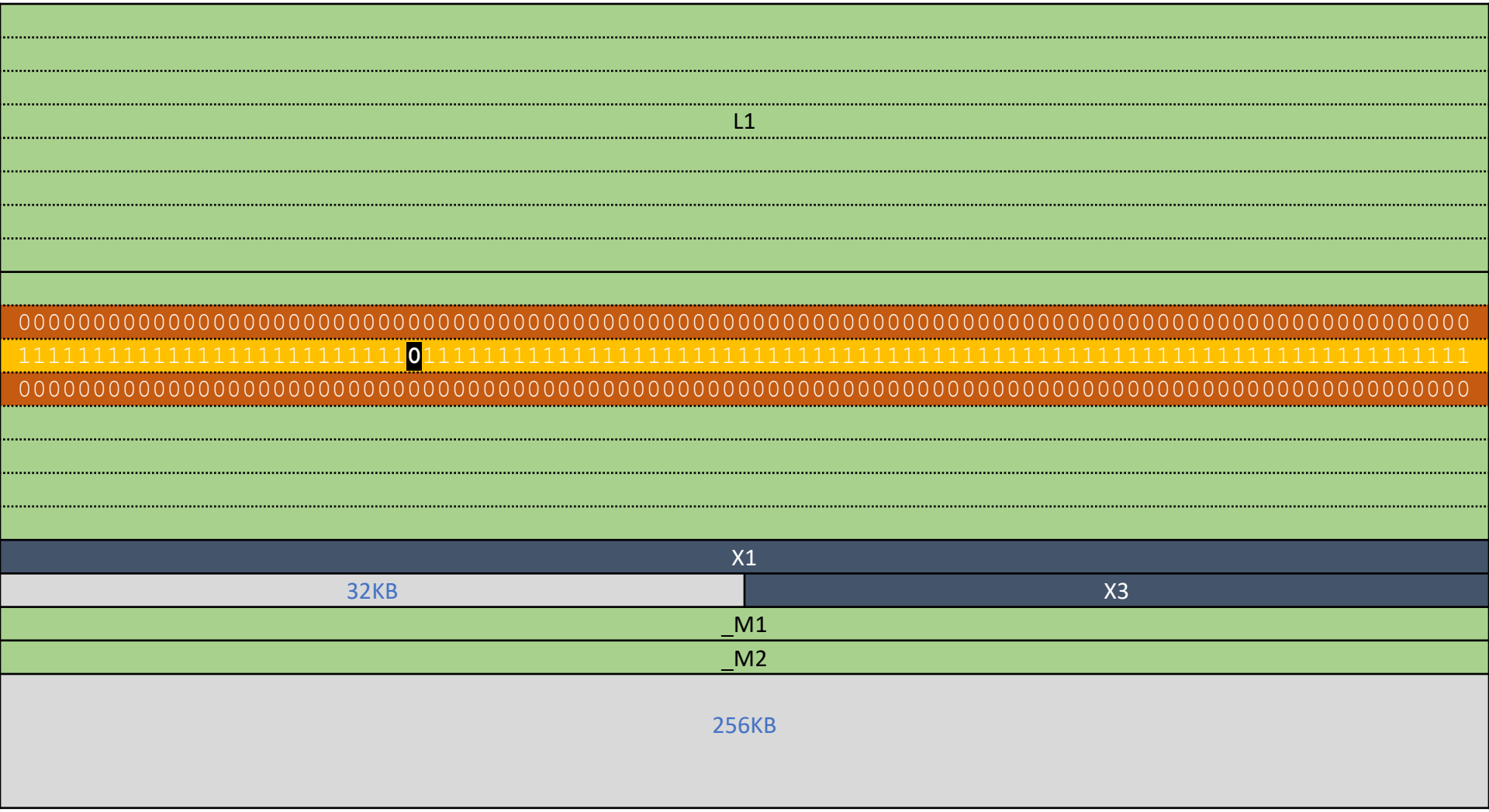
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

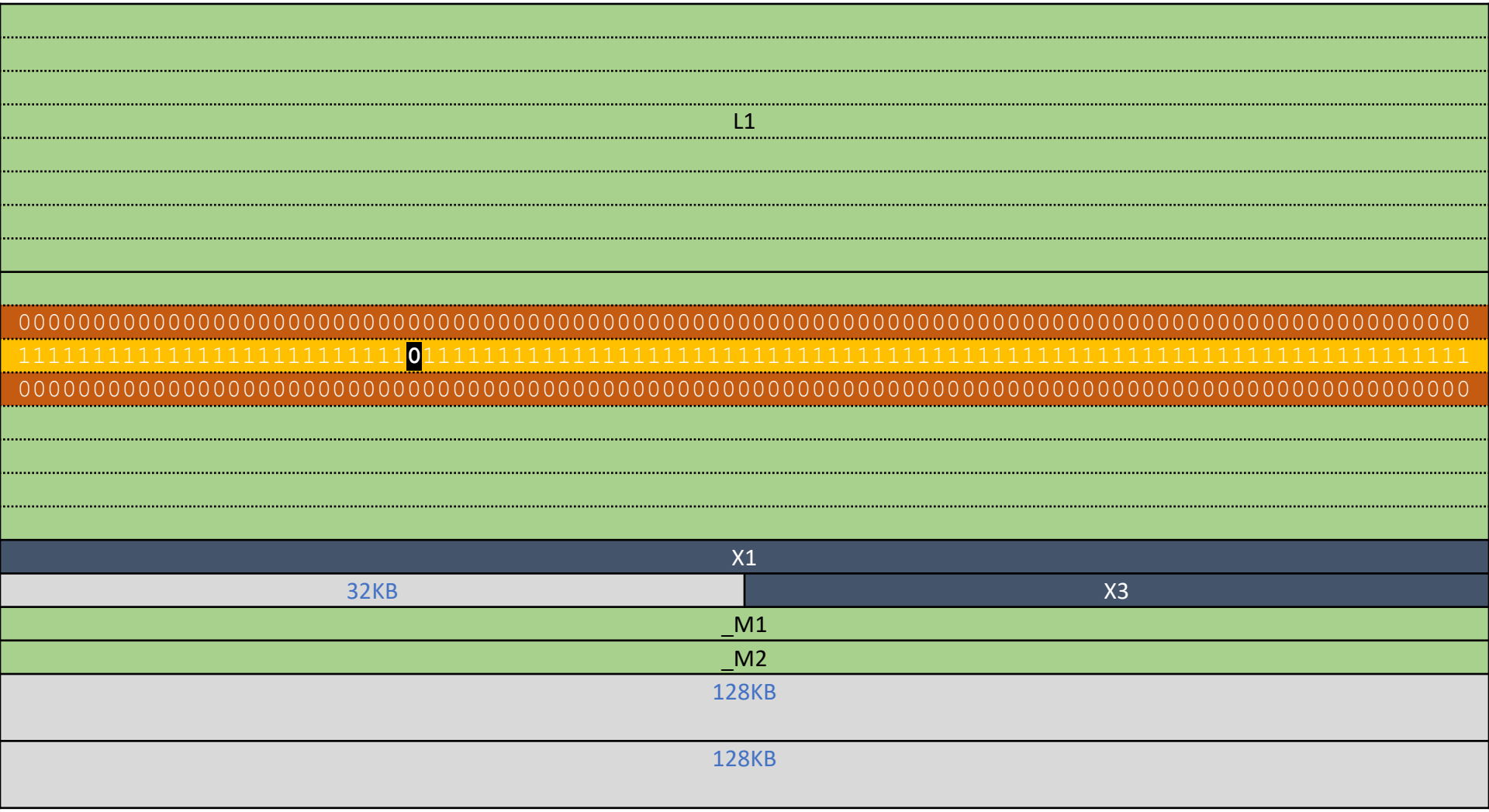
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

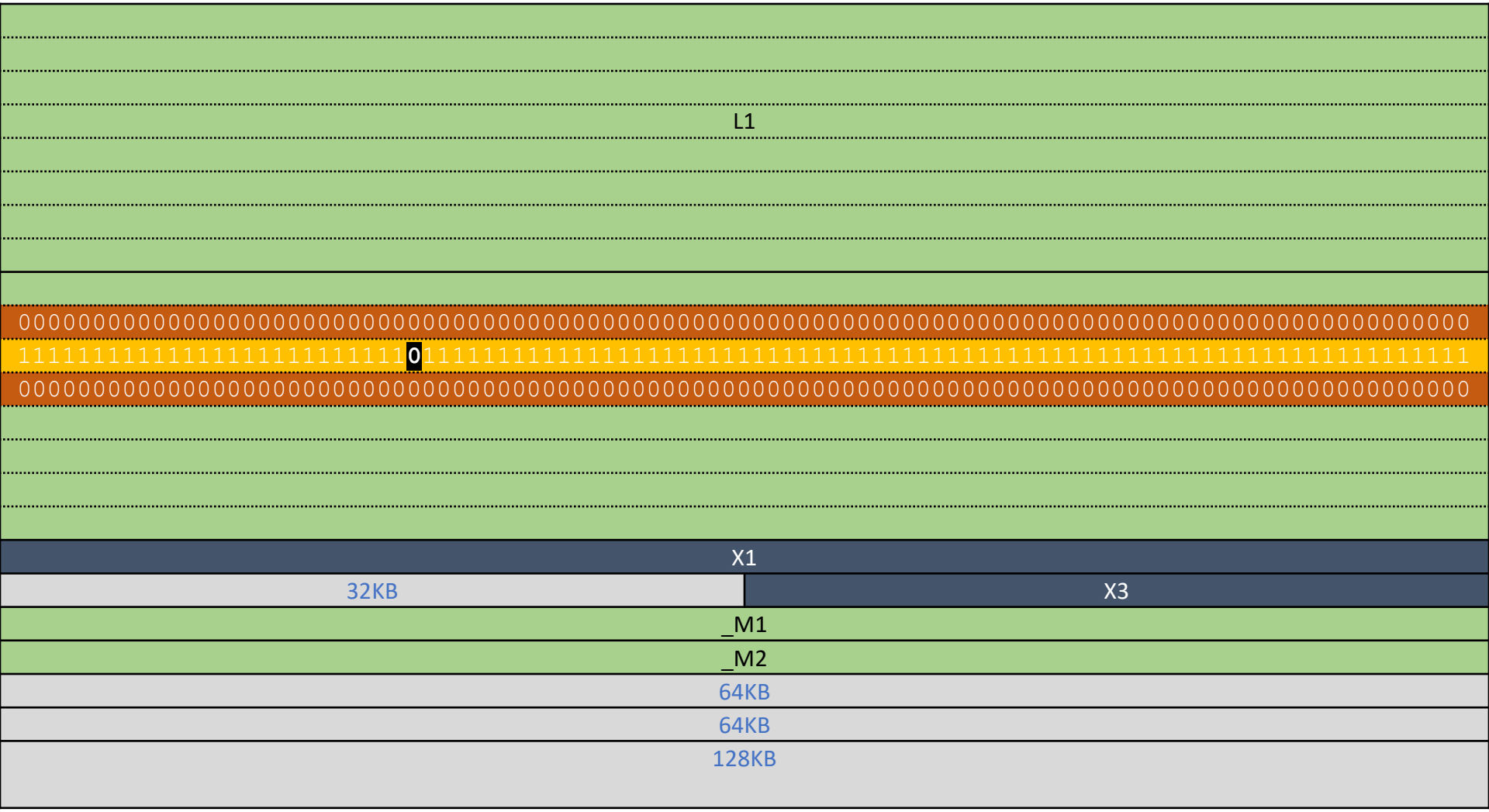
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

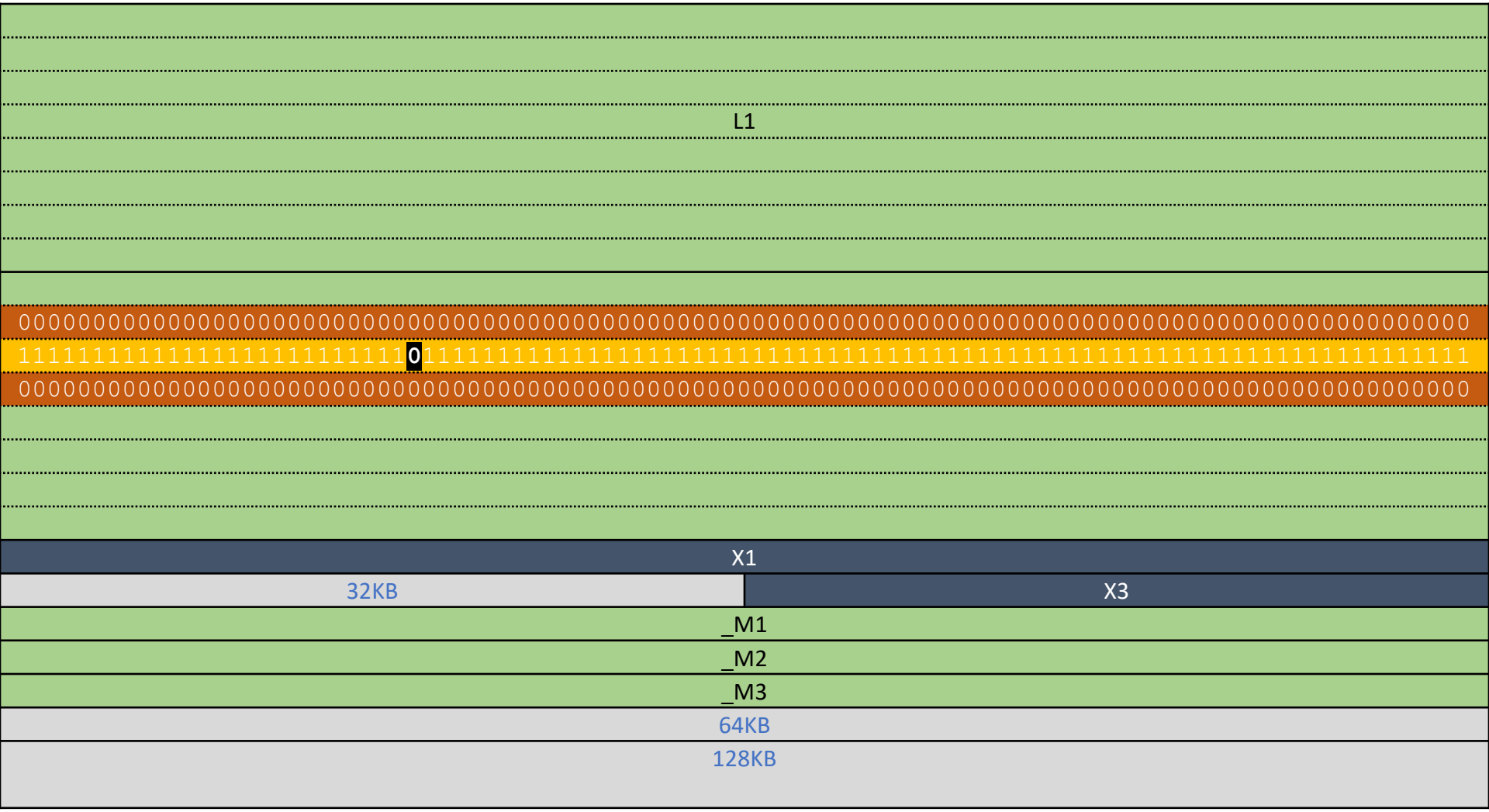
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

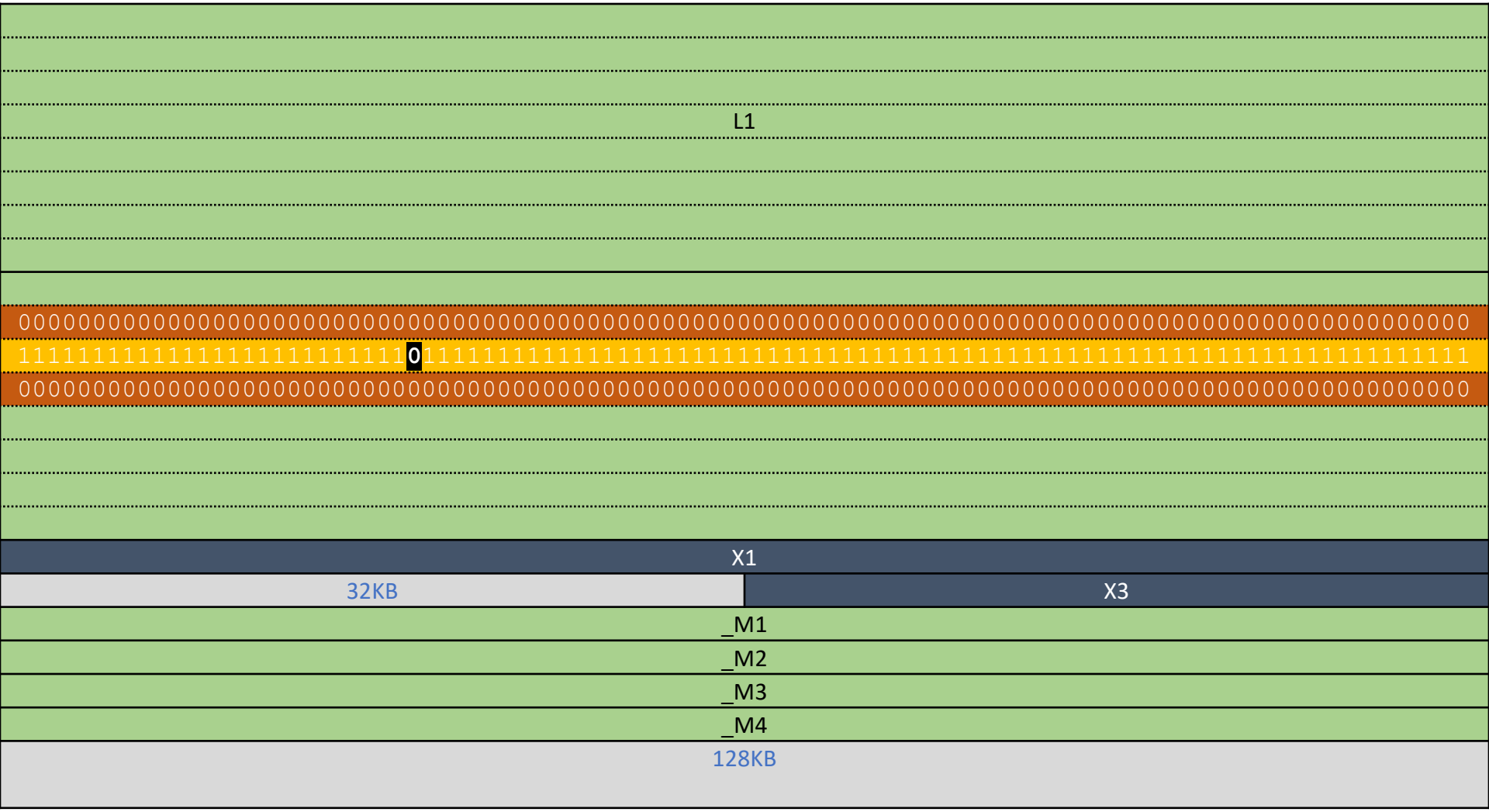
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

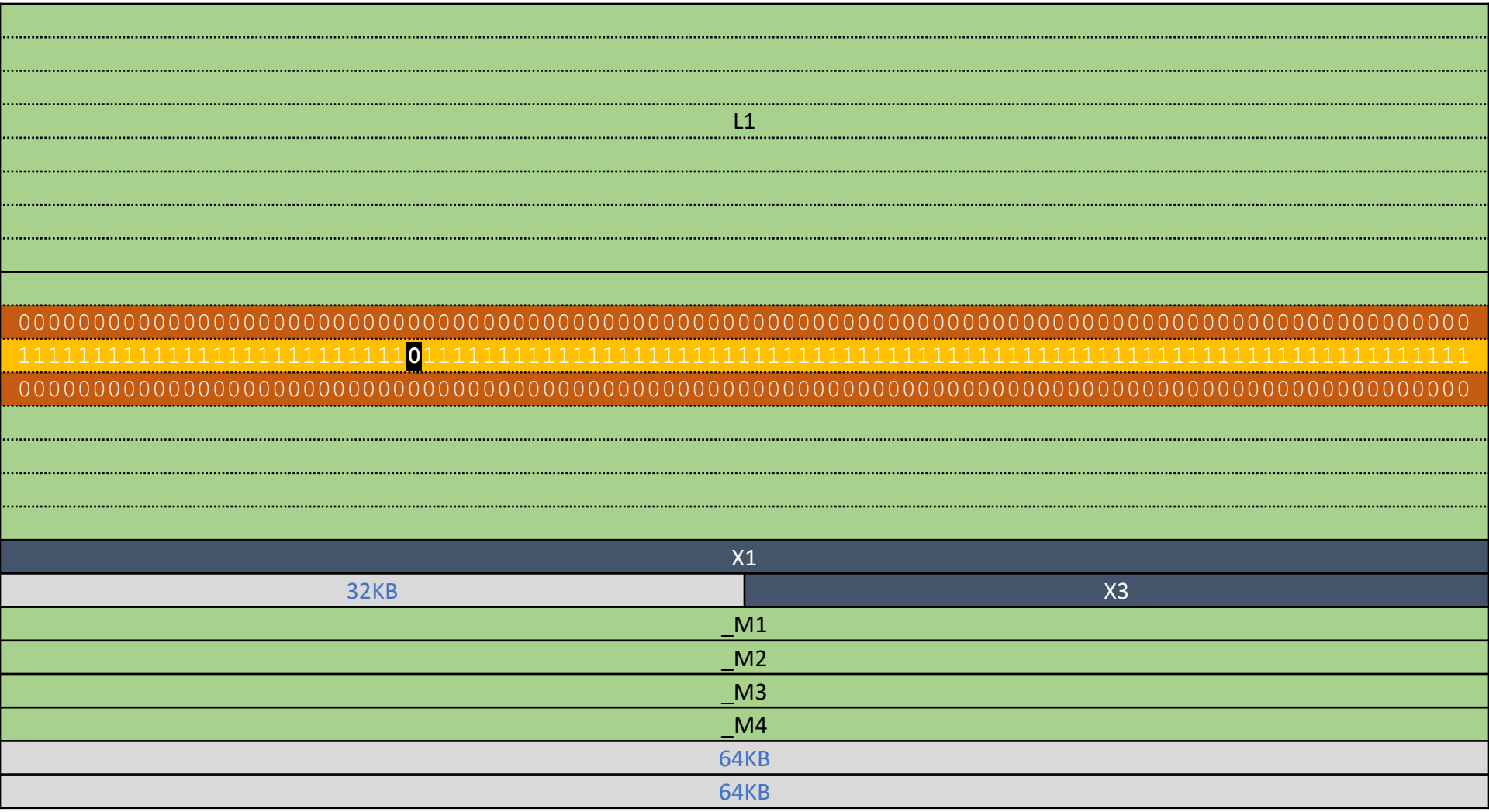
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

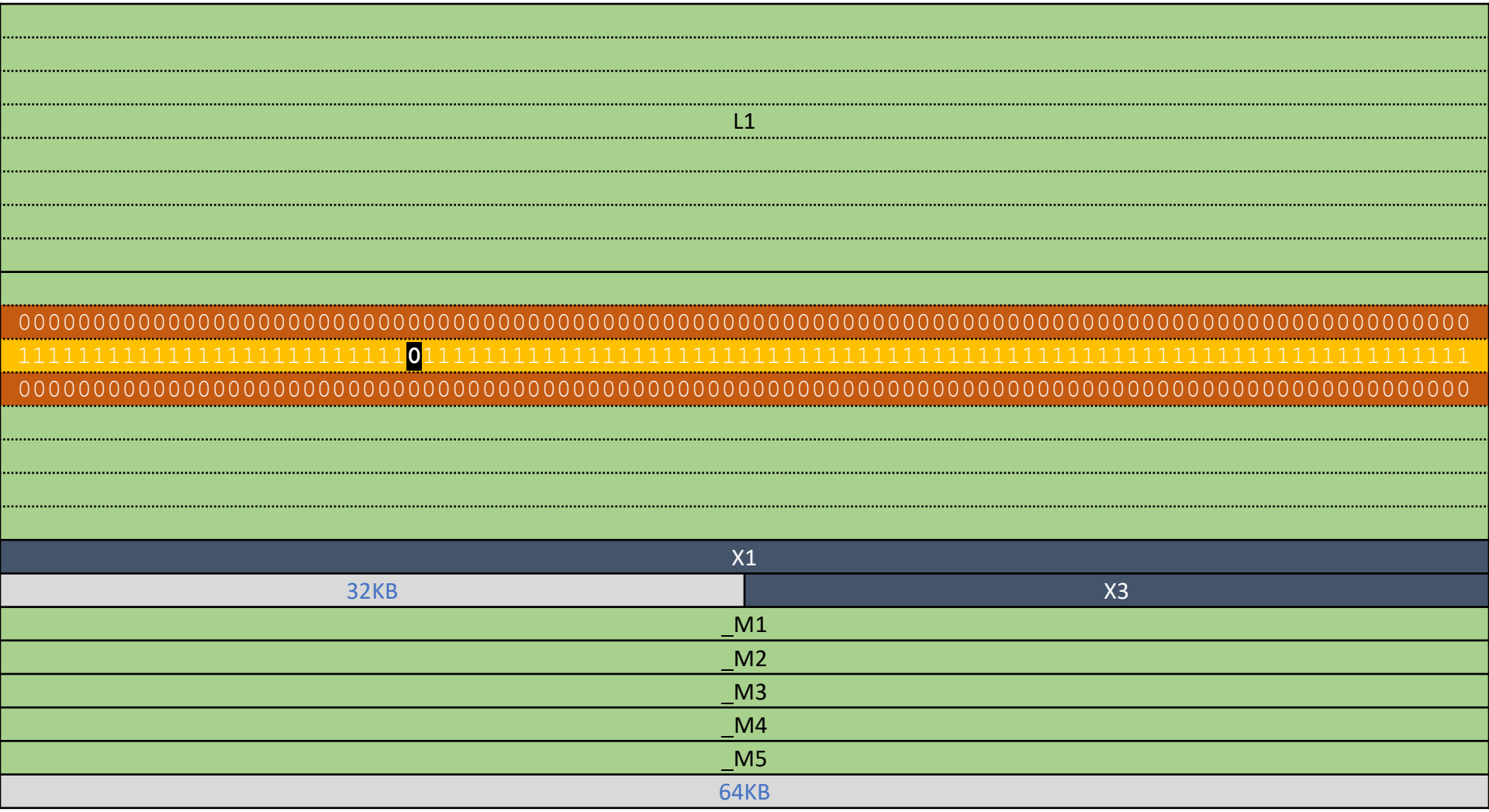
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

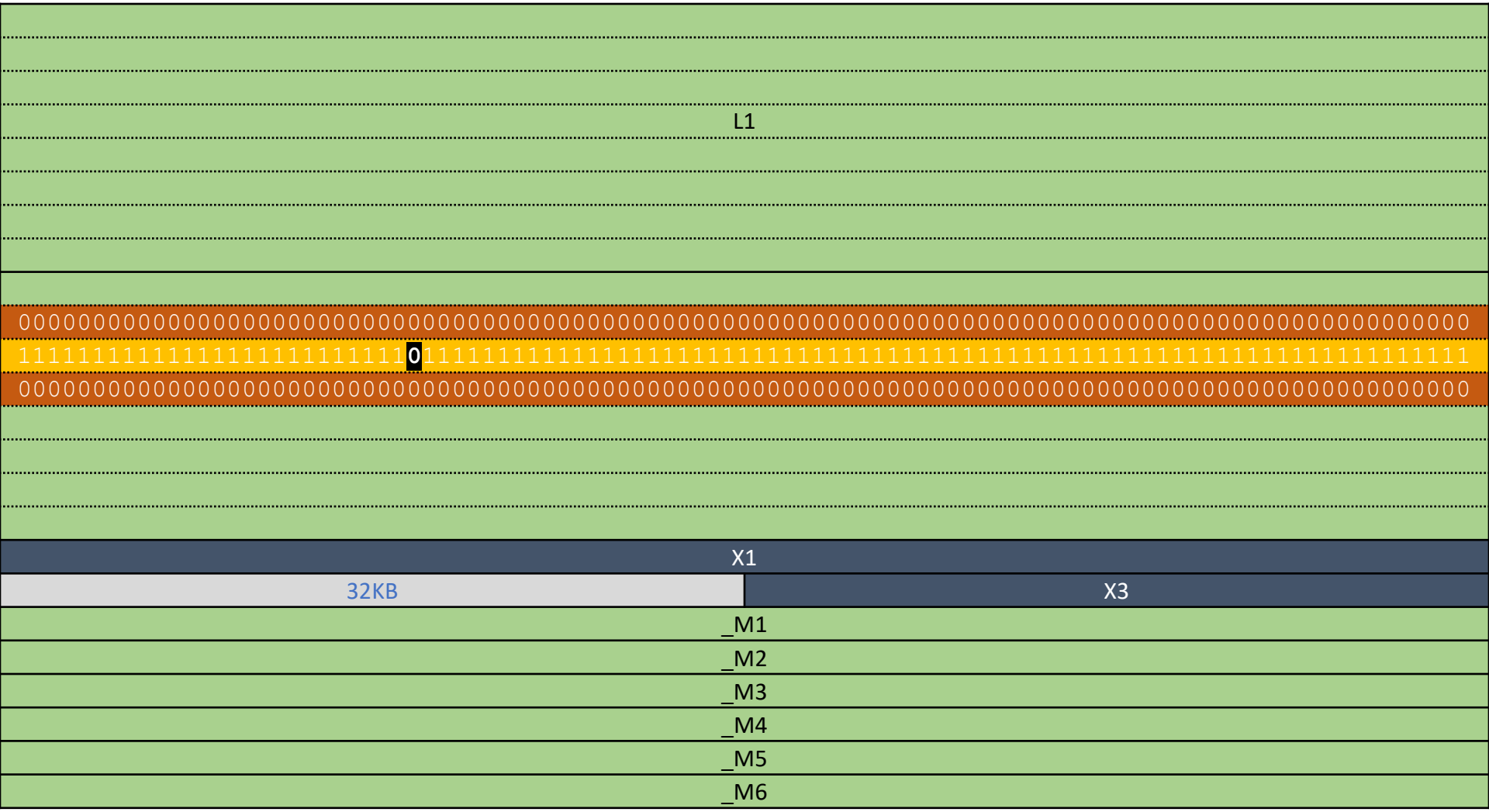
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 2/8

Exhaust *Medium-sized* chunks

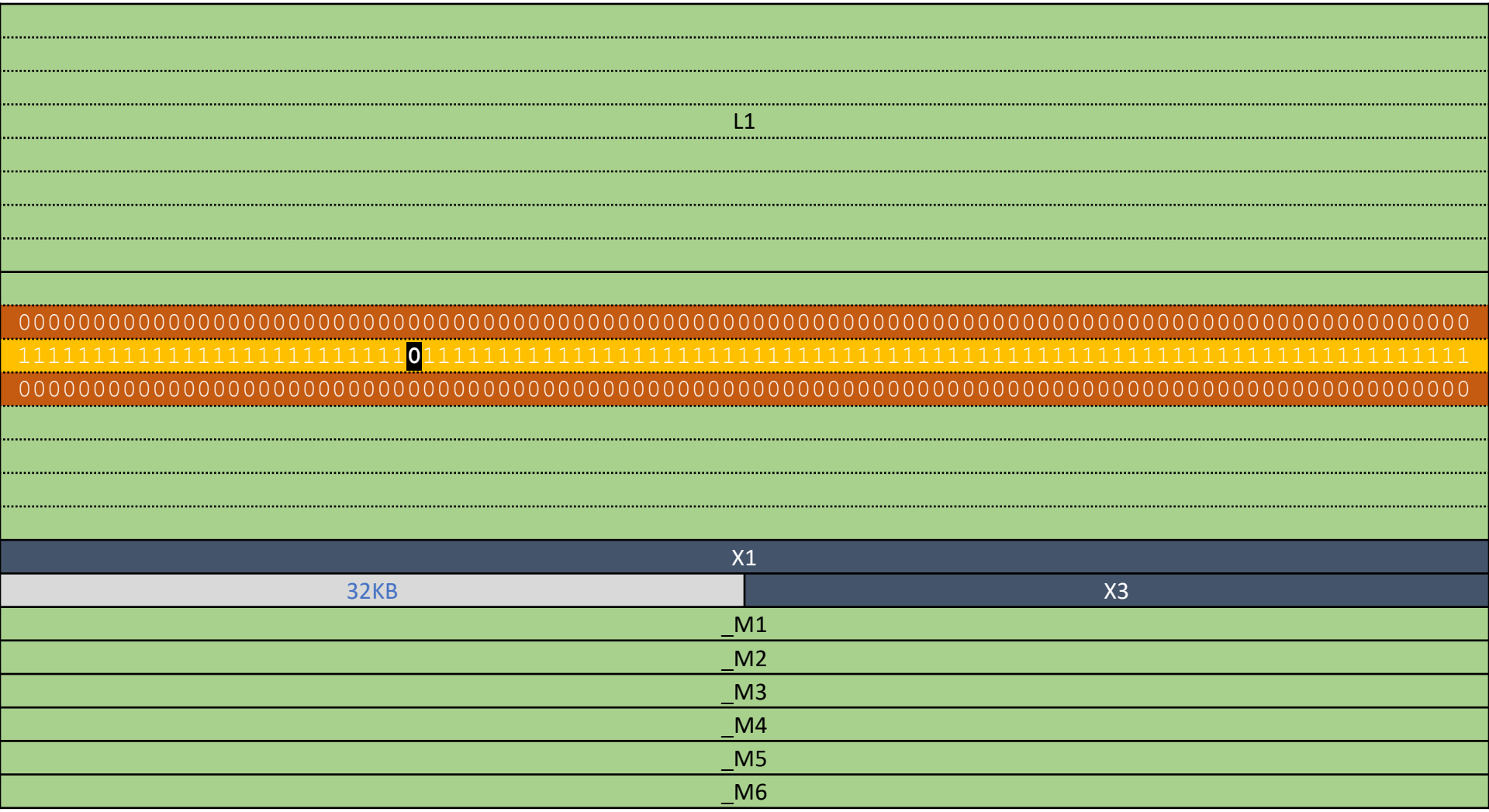
```
_M1, _M2, …, _Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 3/8

Release *Large* chunk with vulnerable page

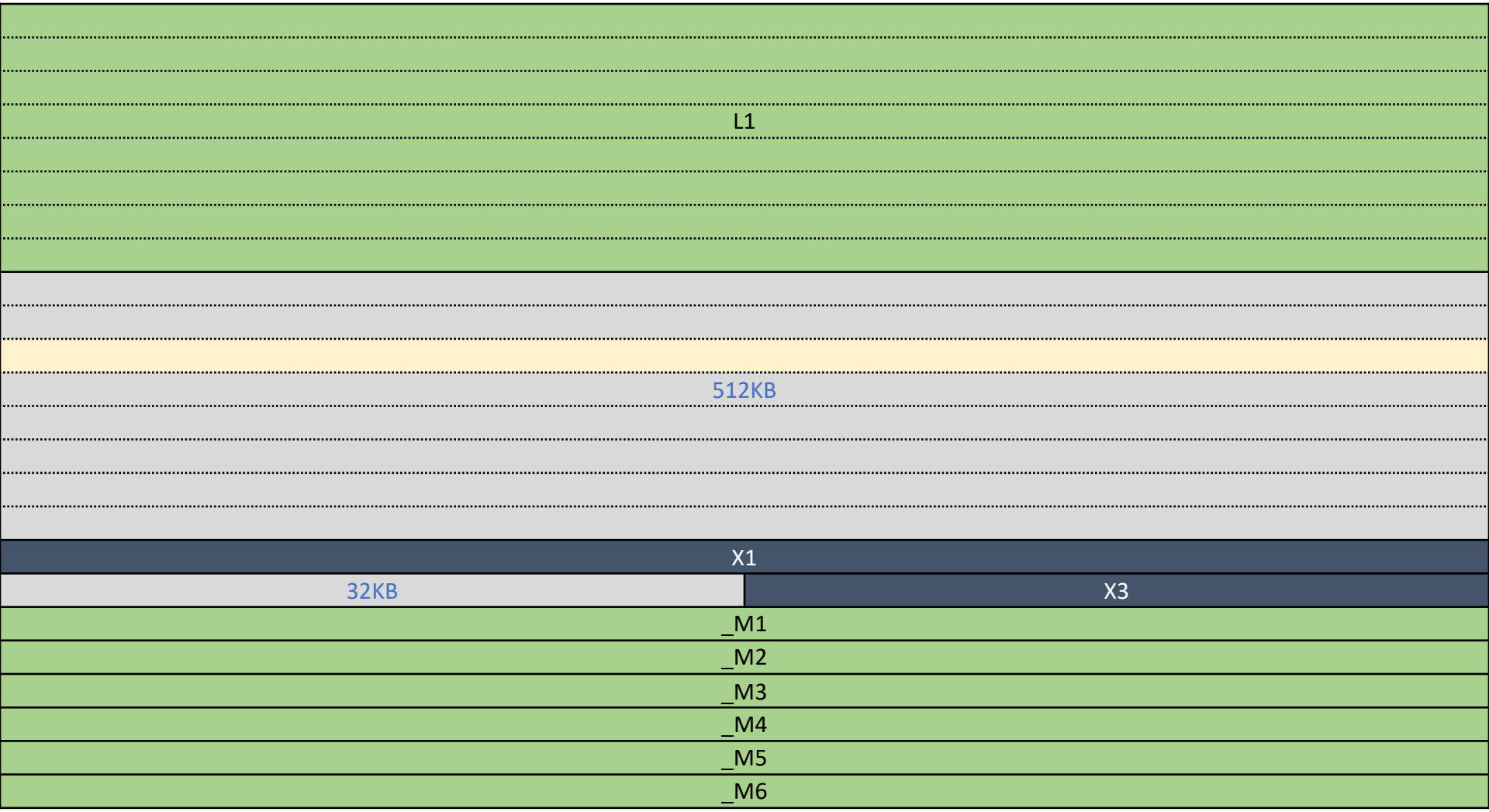
```
Release(L2); // L chunk with vulnerable page
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

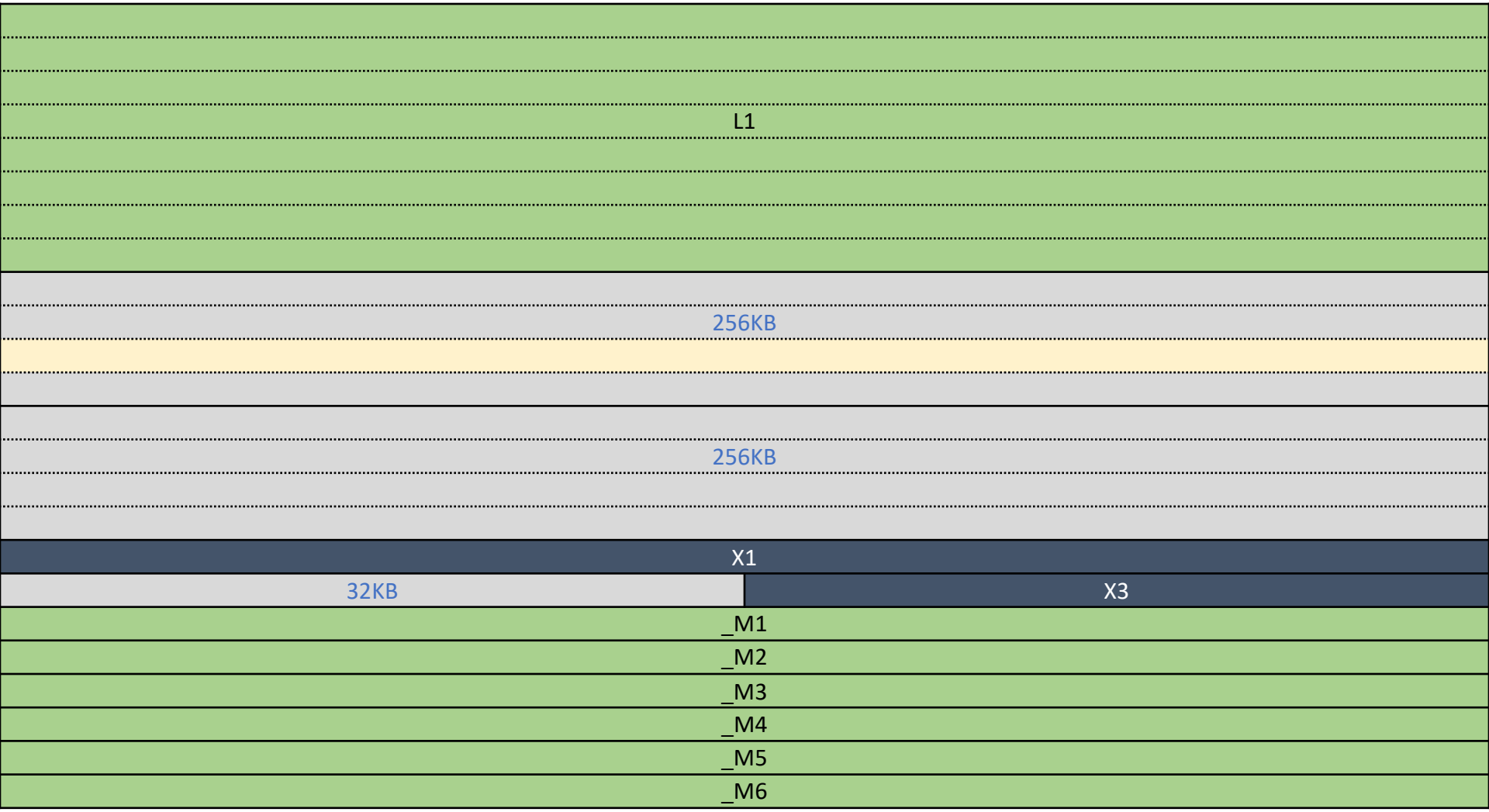
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

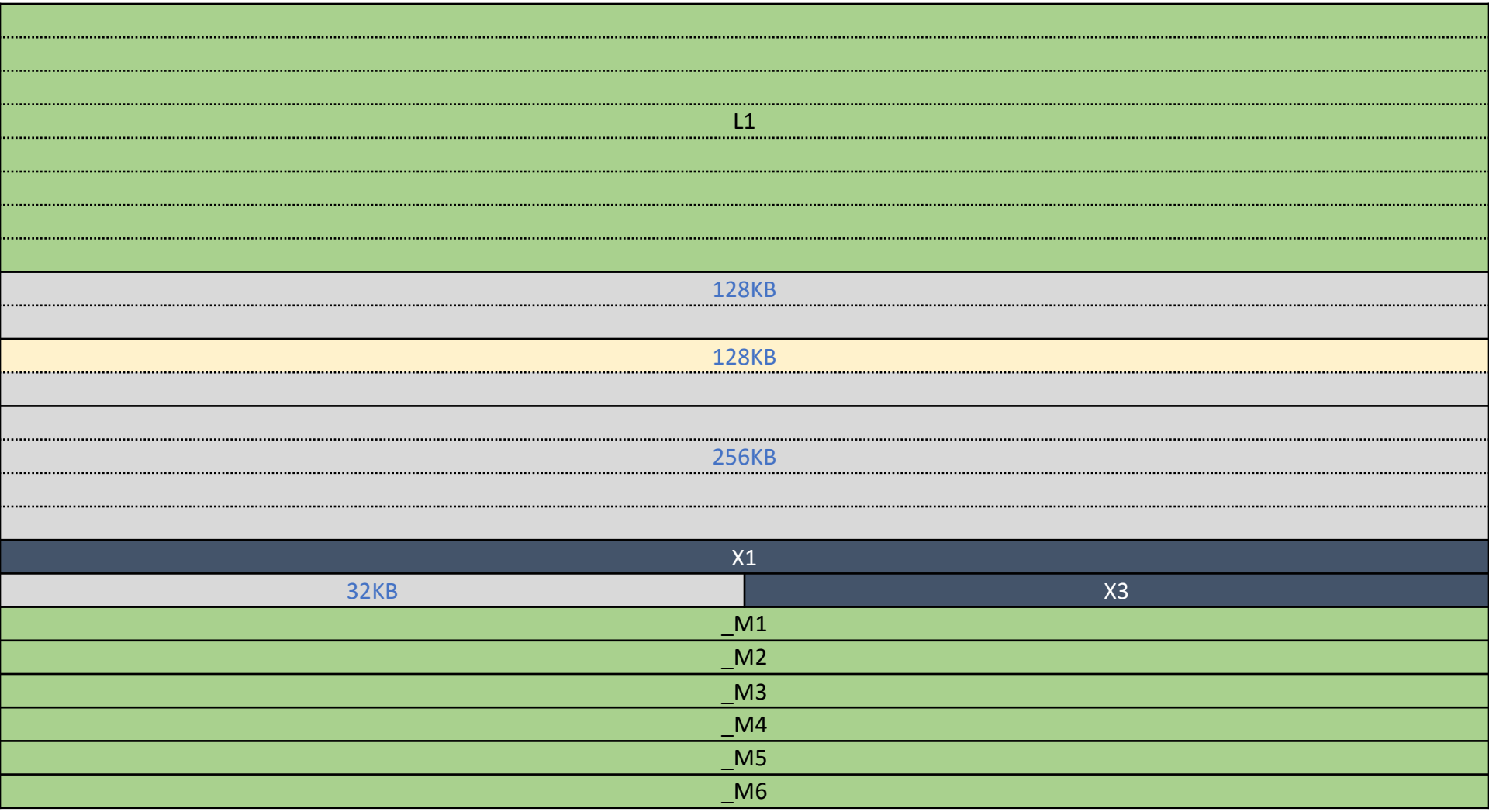
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

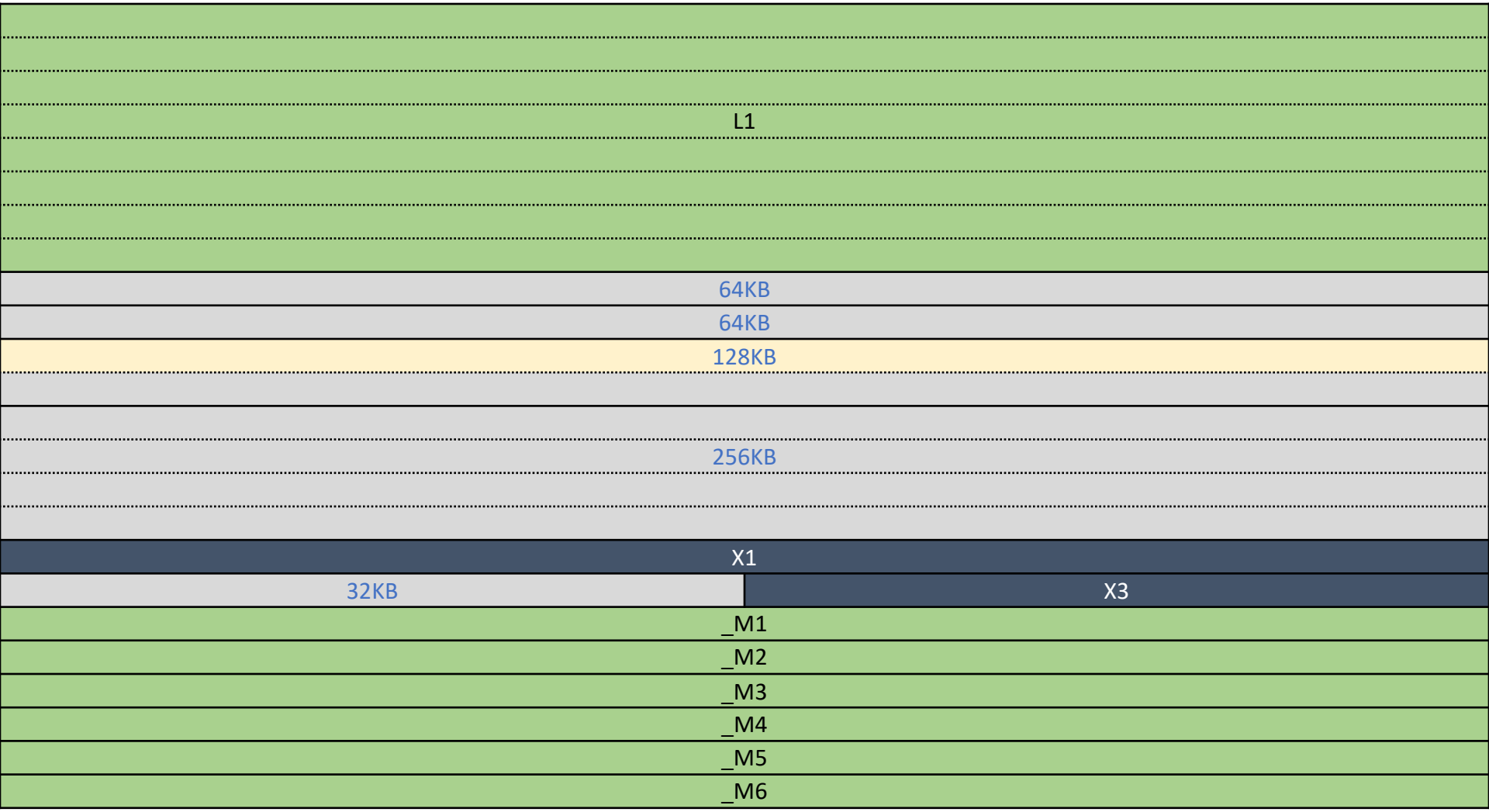
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

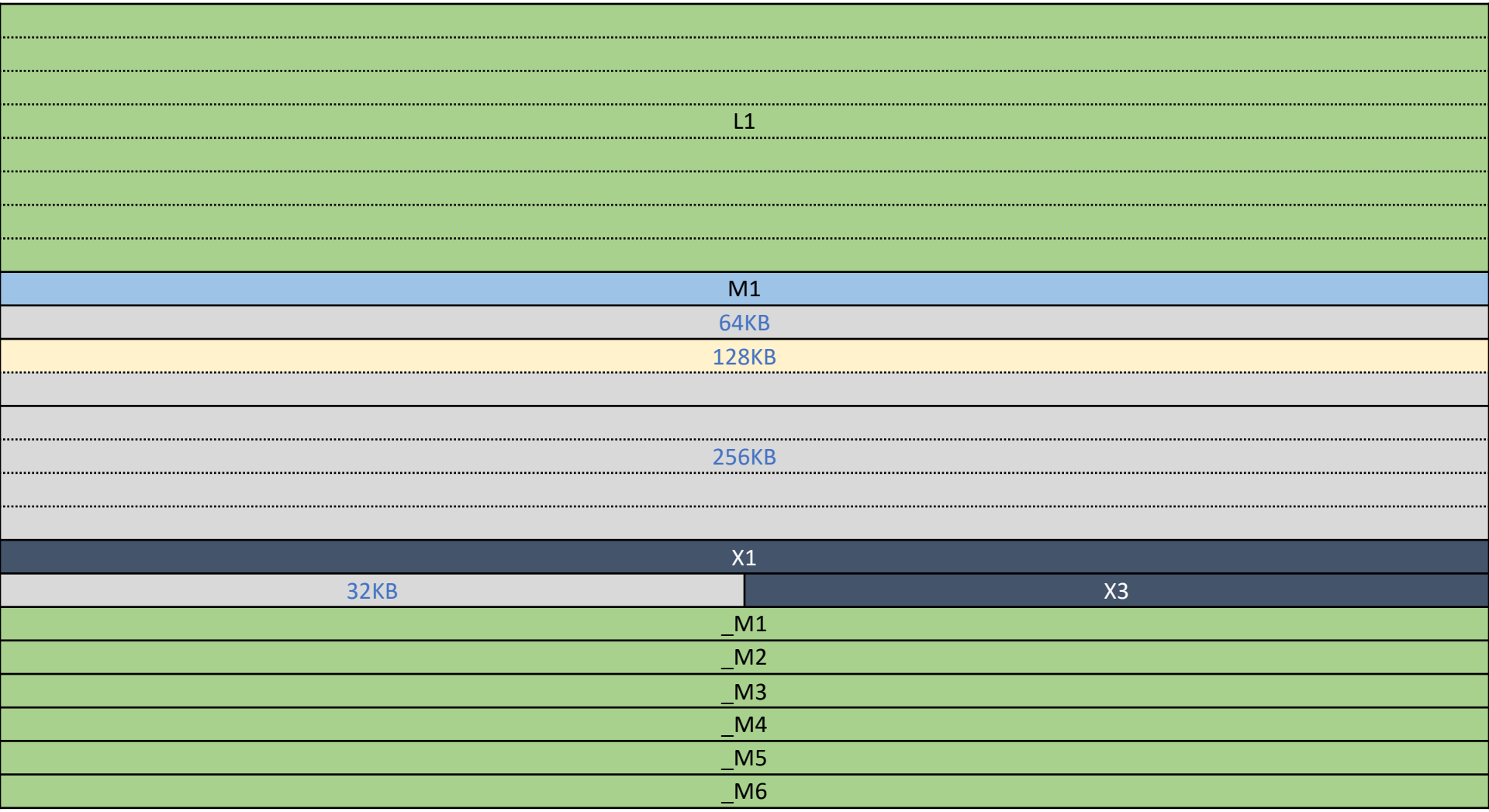
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

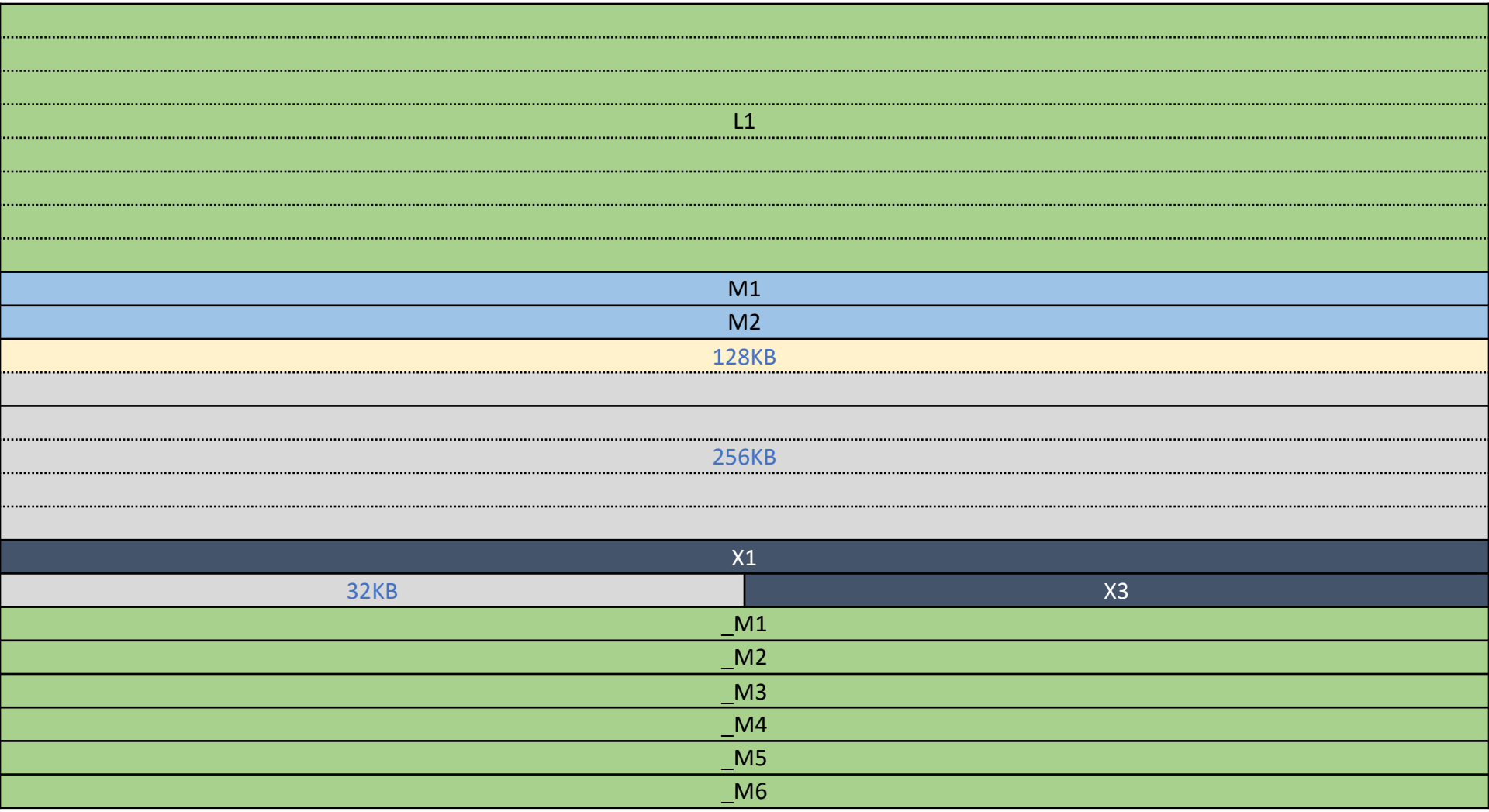
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

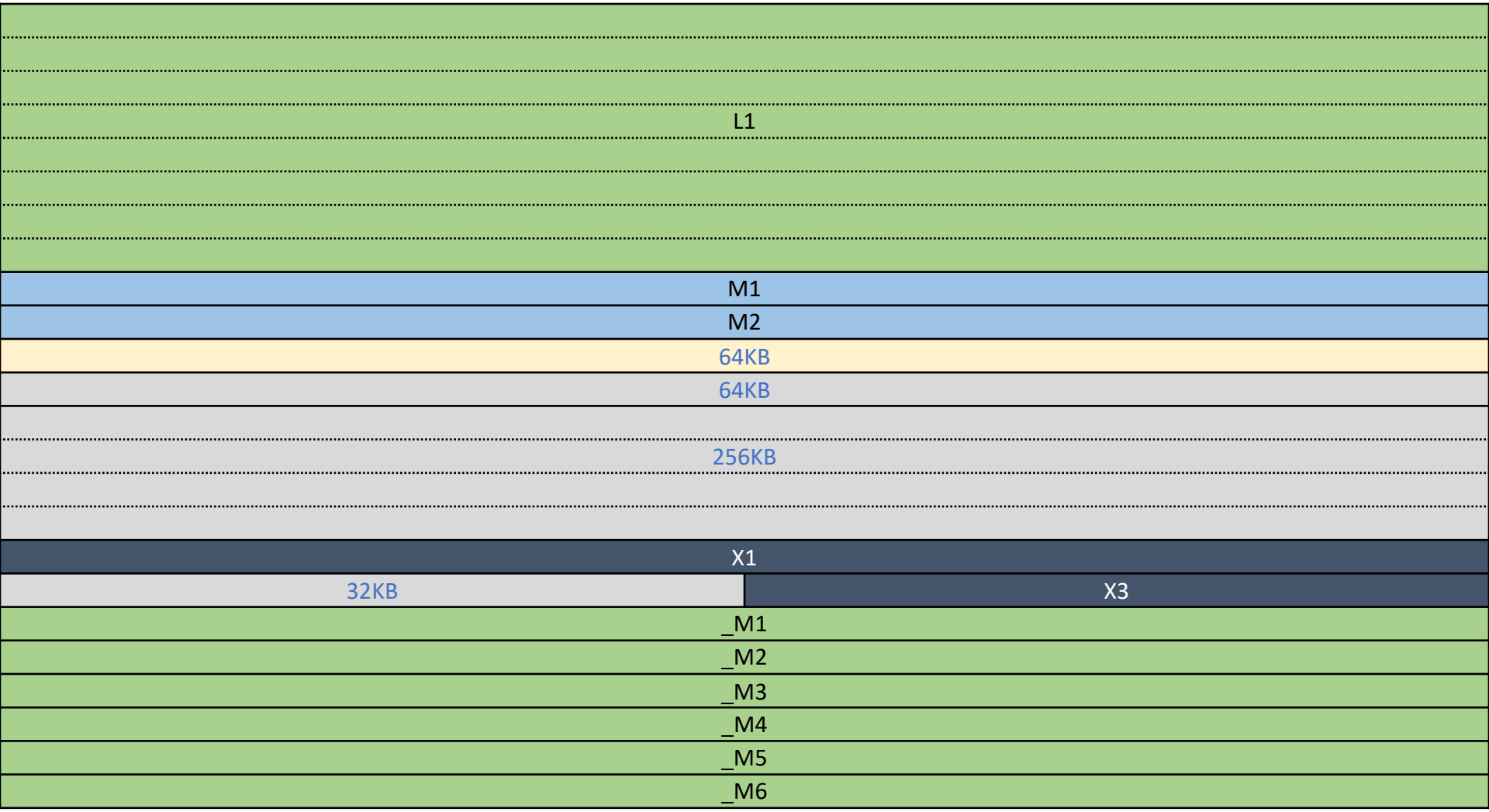
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

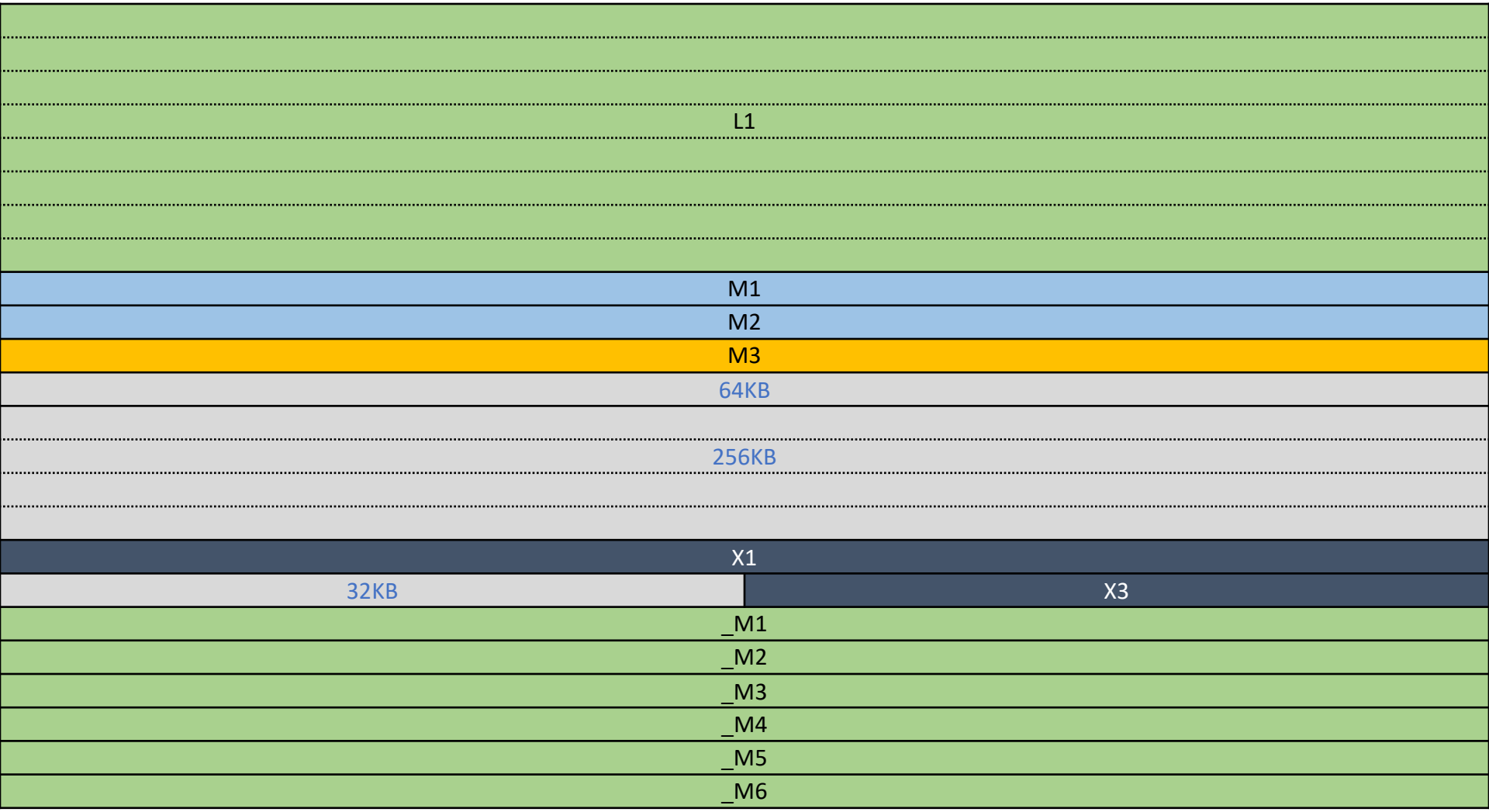
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

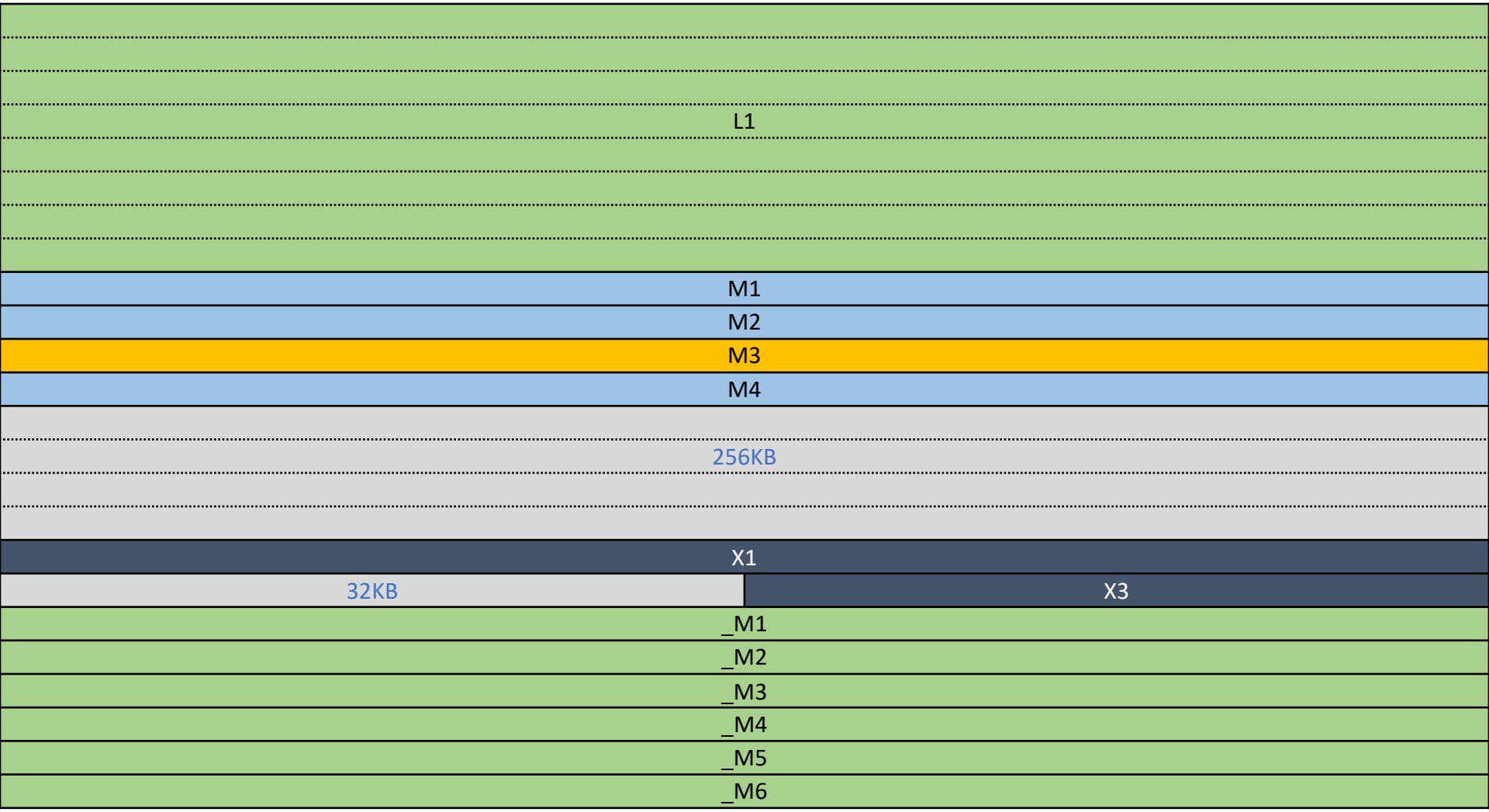
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

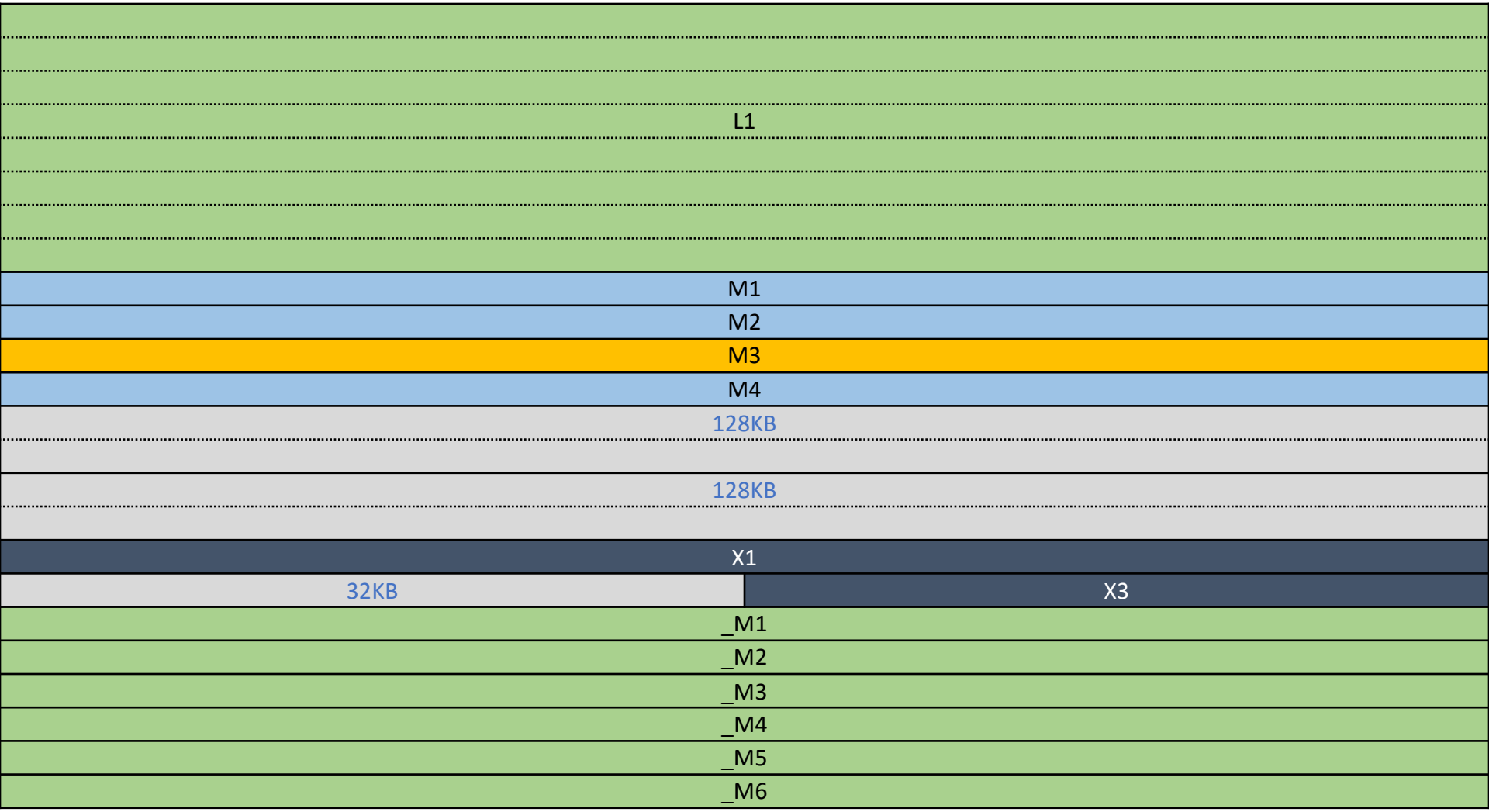
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

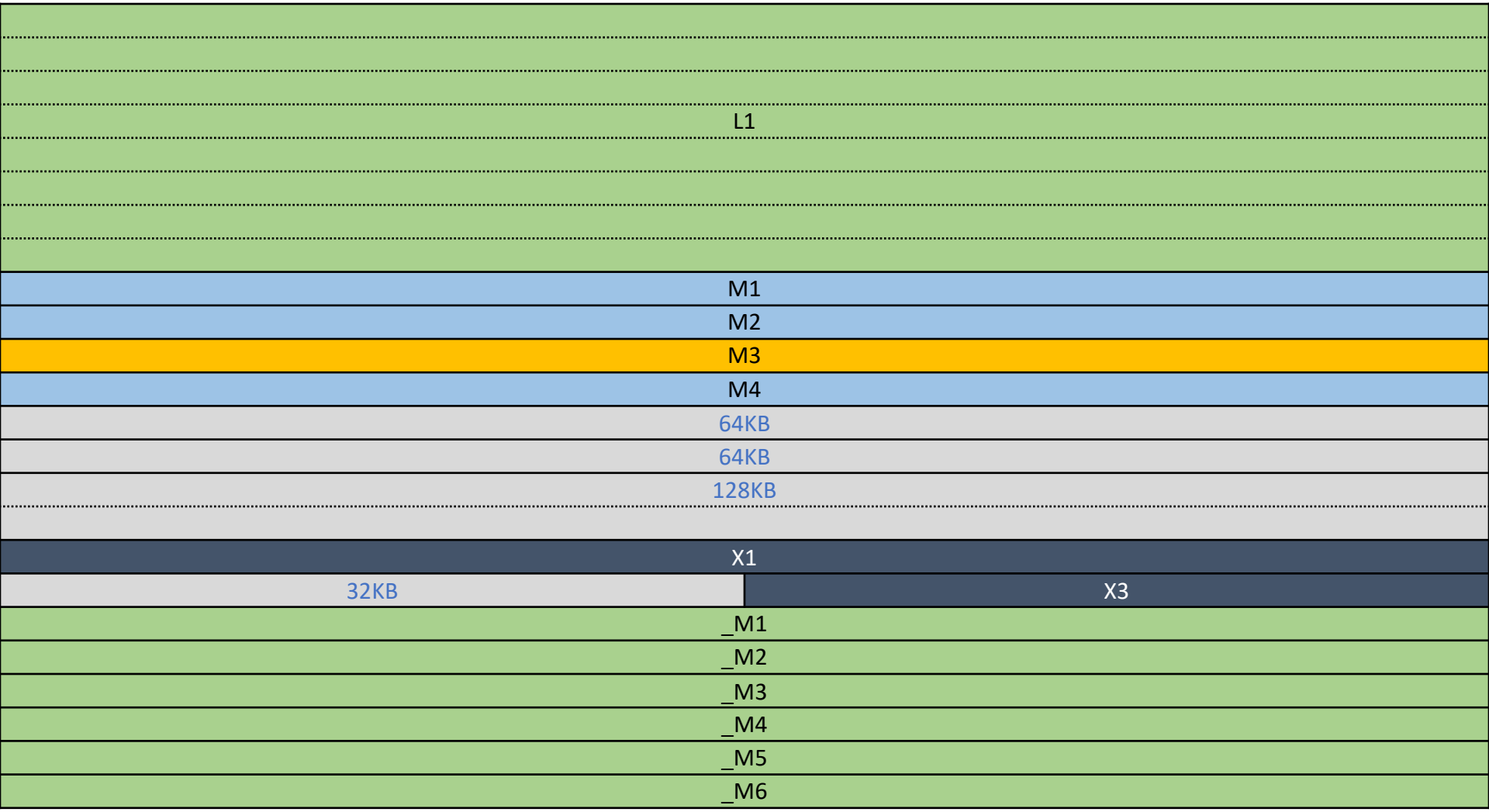
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

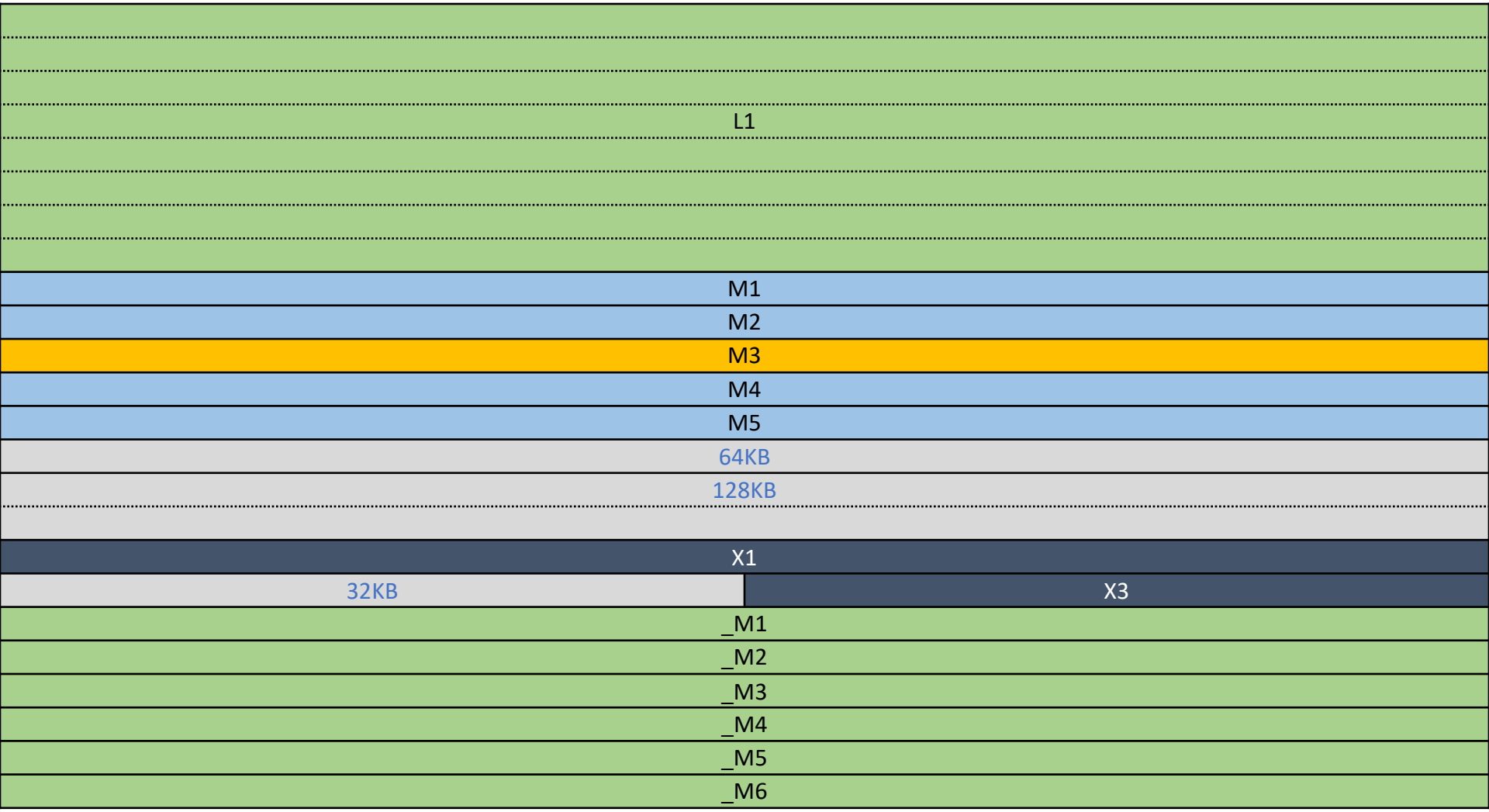
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

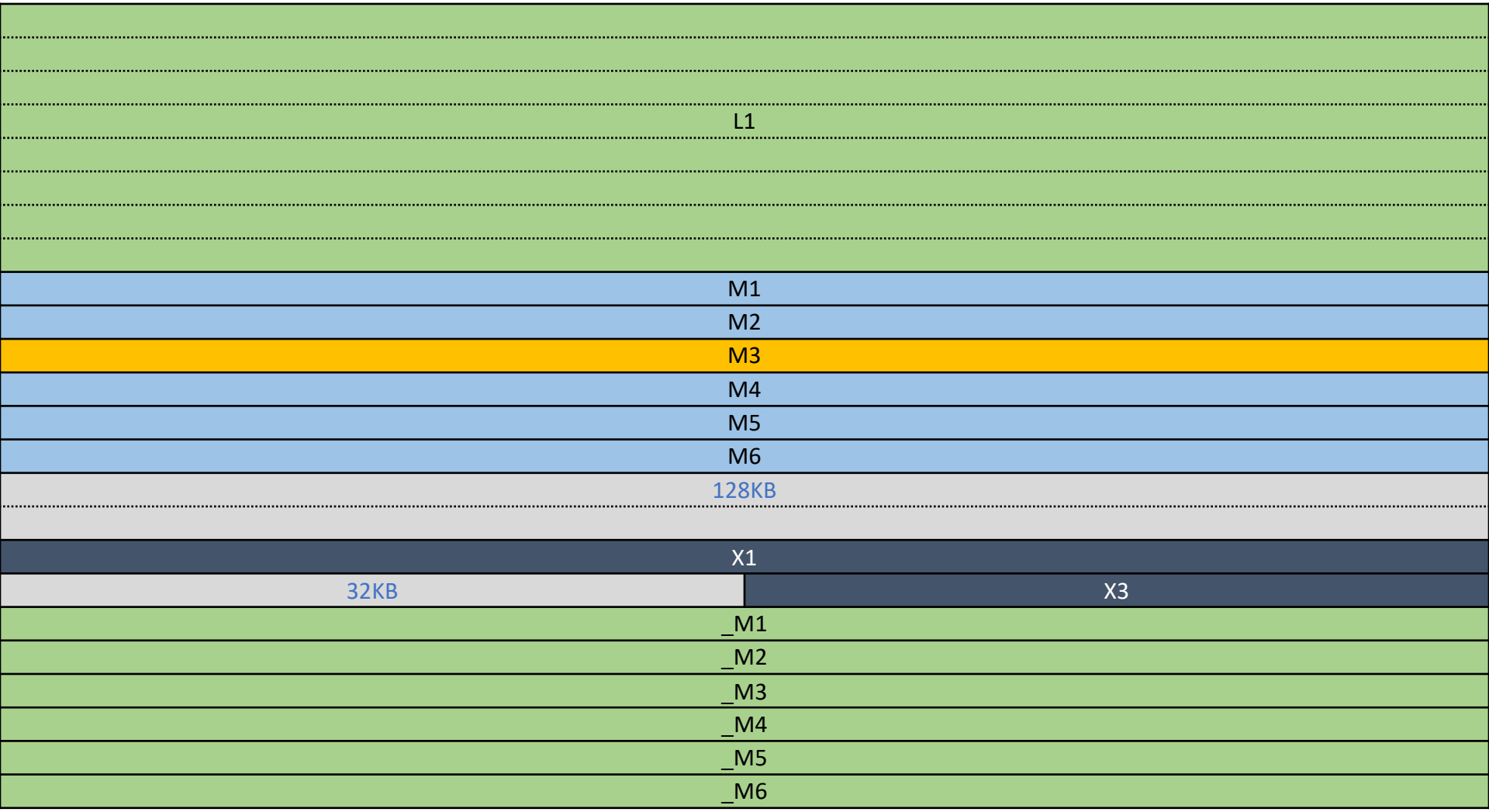
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

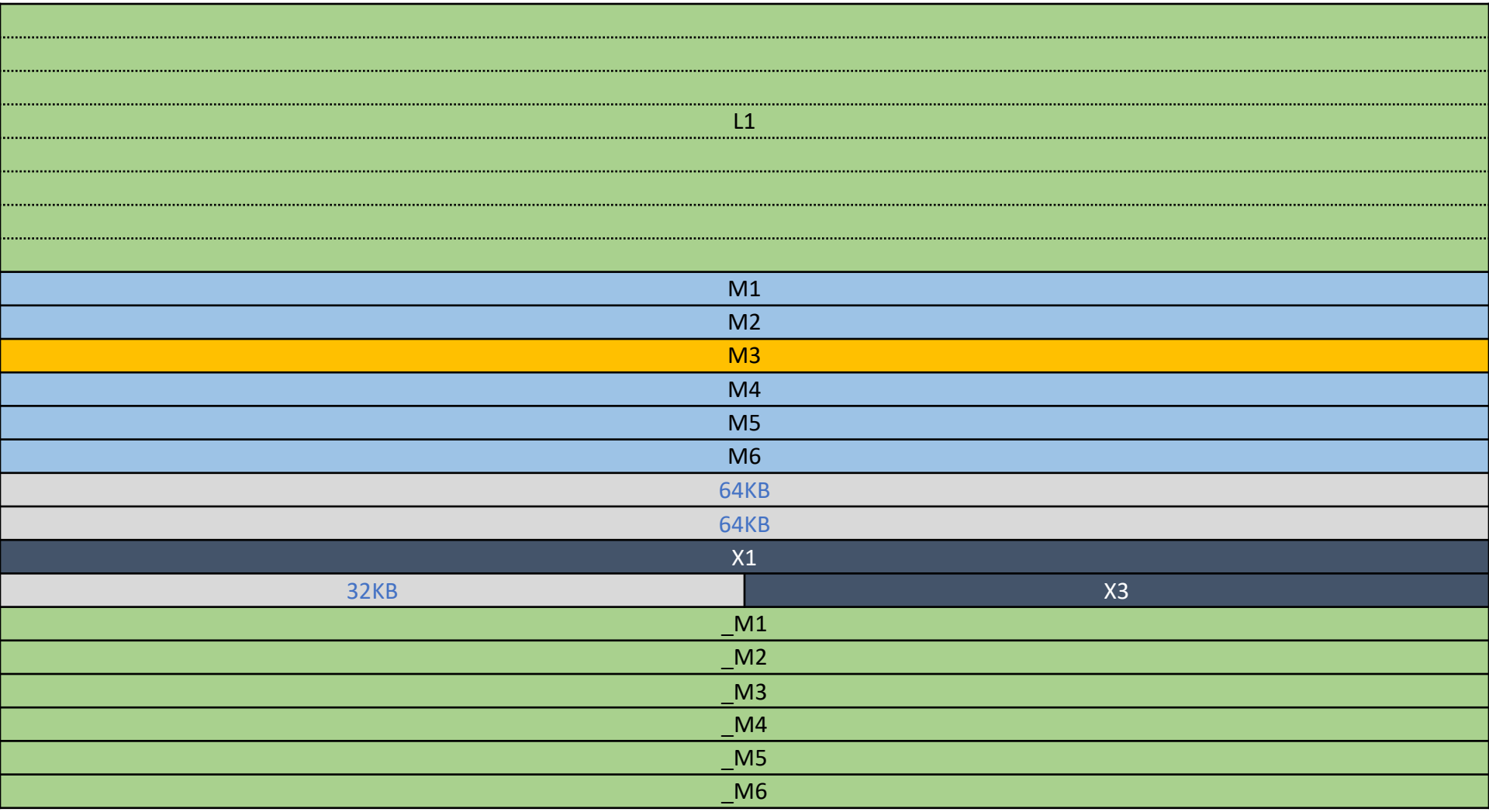
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

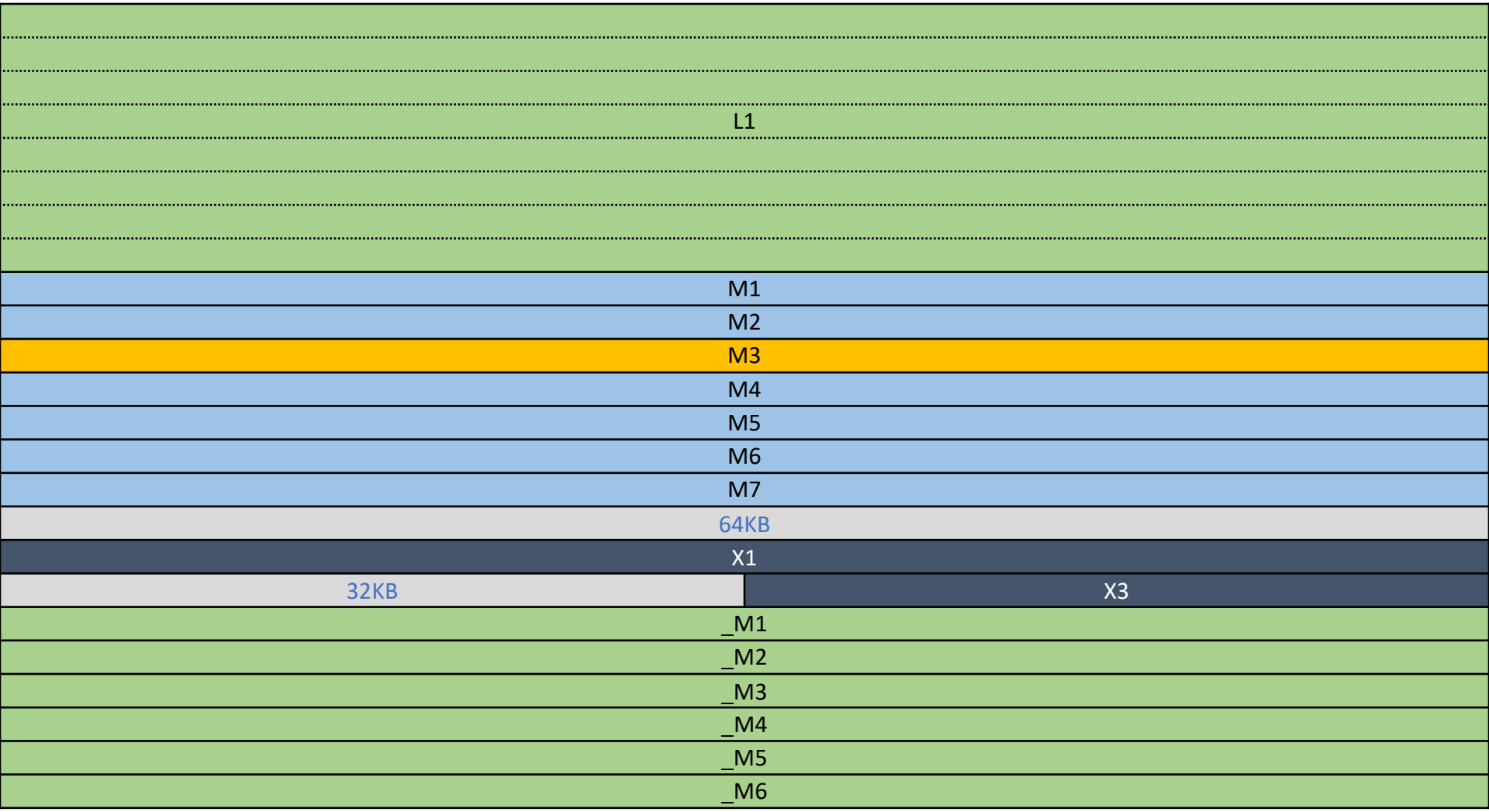
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

Exhaust *Medium-sized* chunks (again)

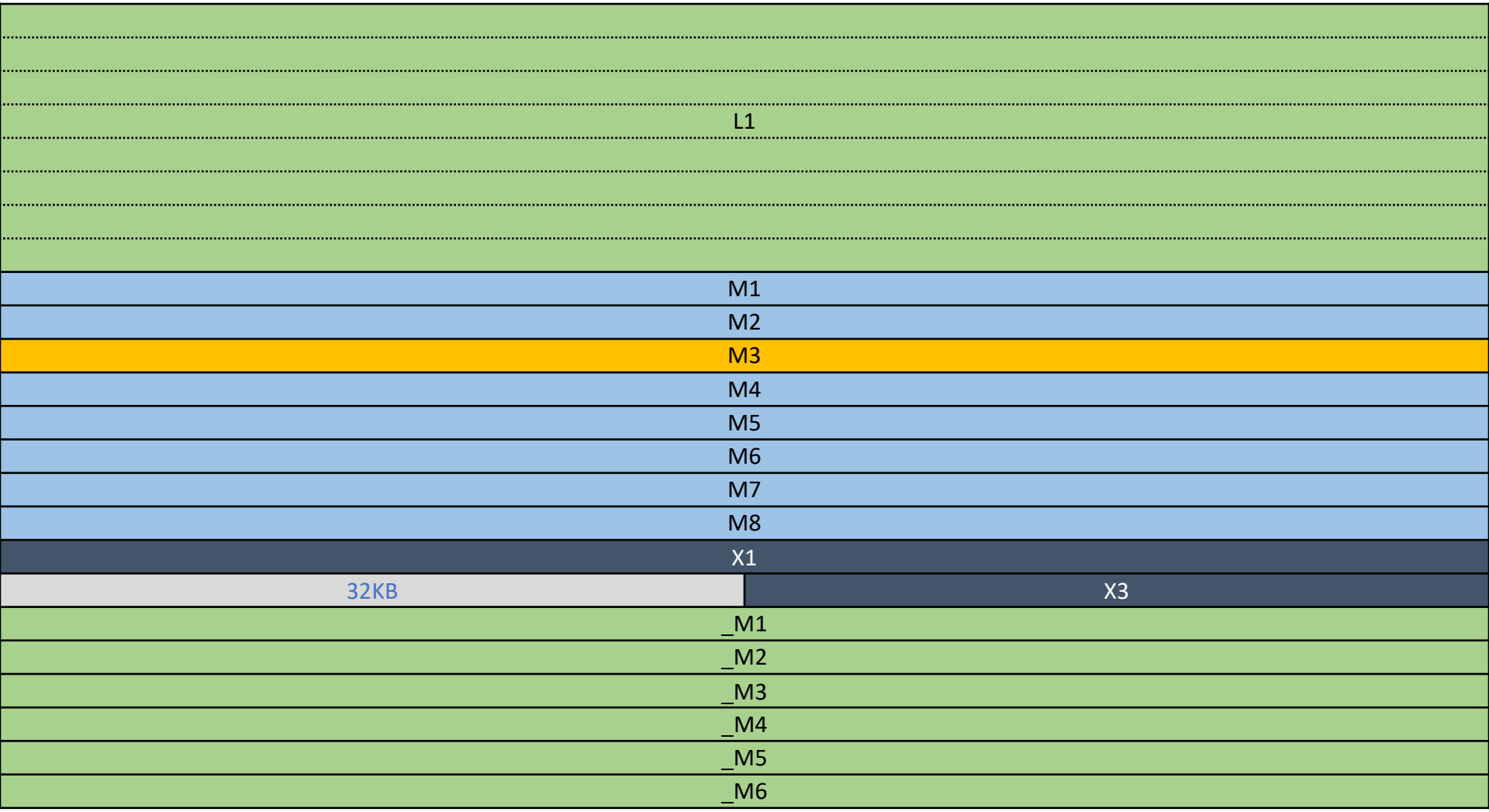
```
M1, M2, …, Mn = exhaust(6); // get all 2^6 = 64KB chunks
```



Phys Feng Shui step 4/8

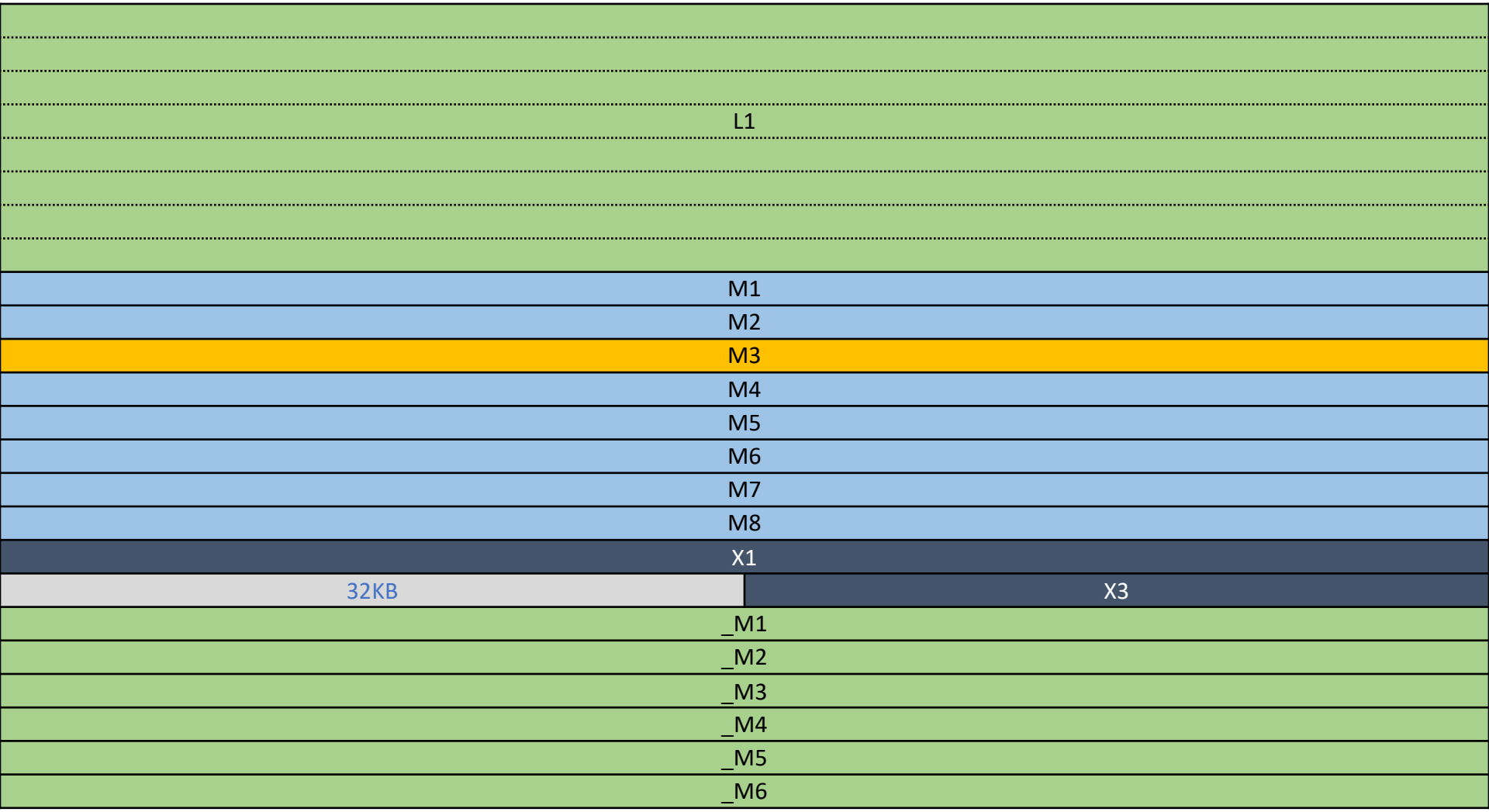
Exhaust *Medium-sized* chunks (again)

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



Phys Feng Shui step 5/8

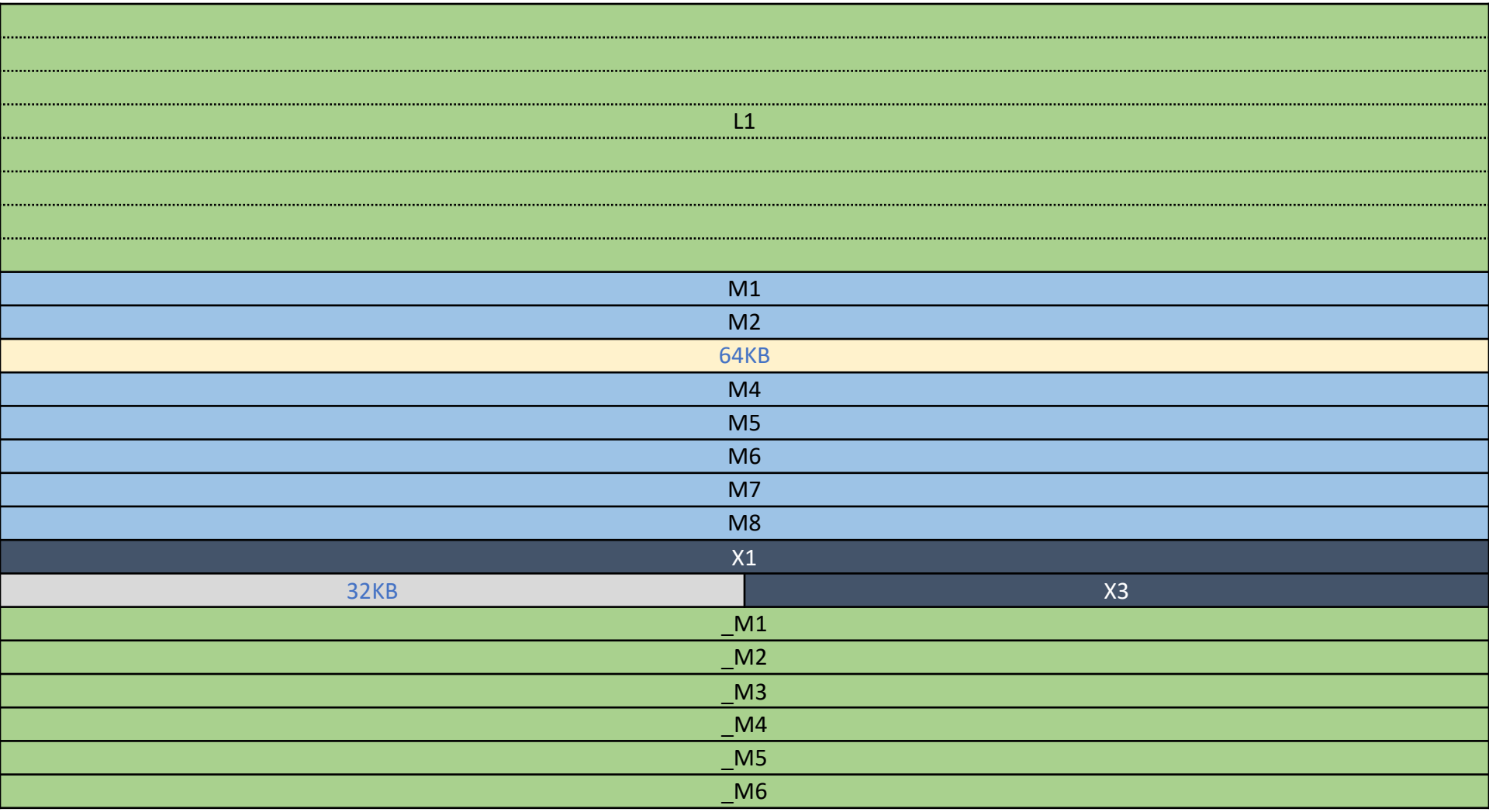
Release vulnerable *Medium-sized* chunk + Release all Large chunks



Phys Feng Shui step 5/8

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

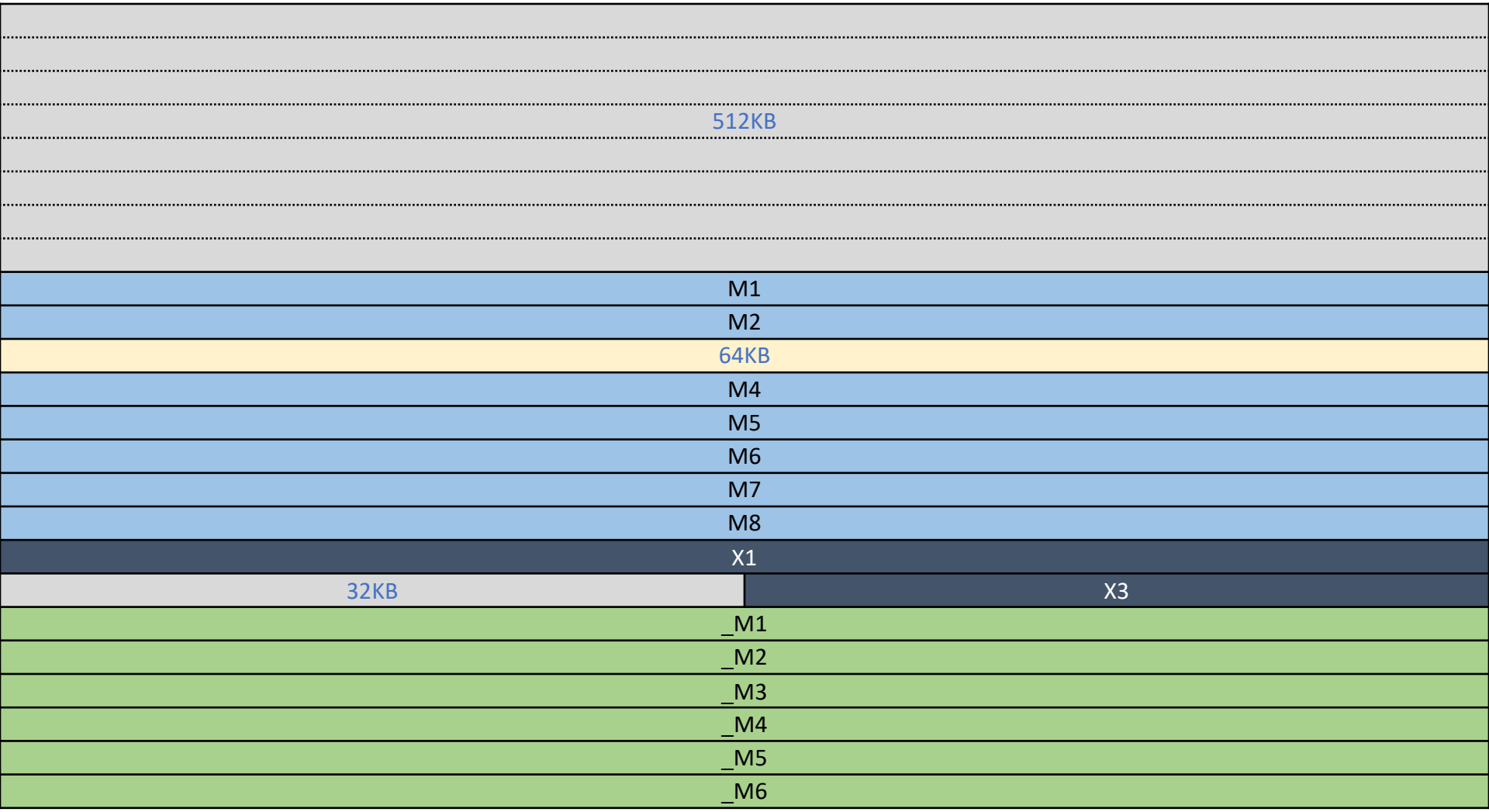
```
Release(M3); // releases the vulnerable row
```



Phys Feng Shui step 5/8

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

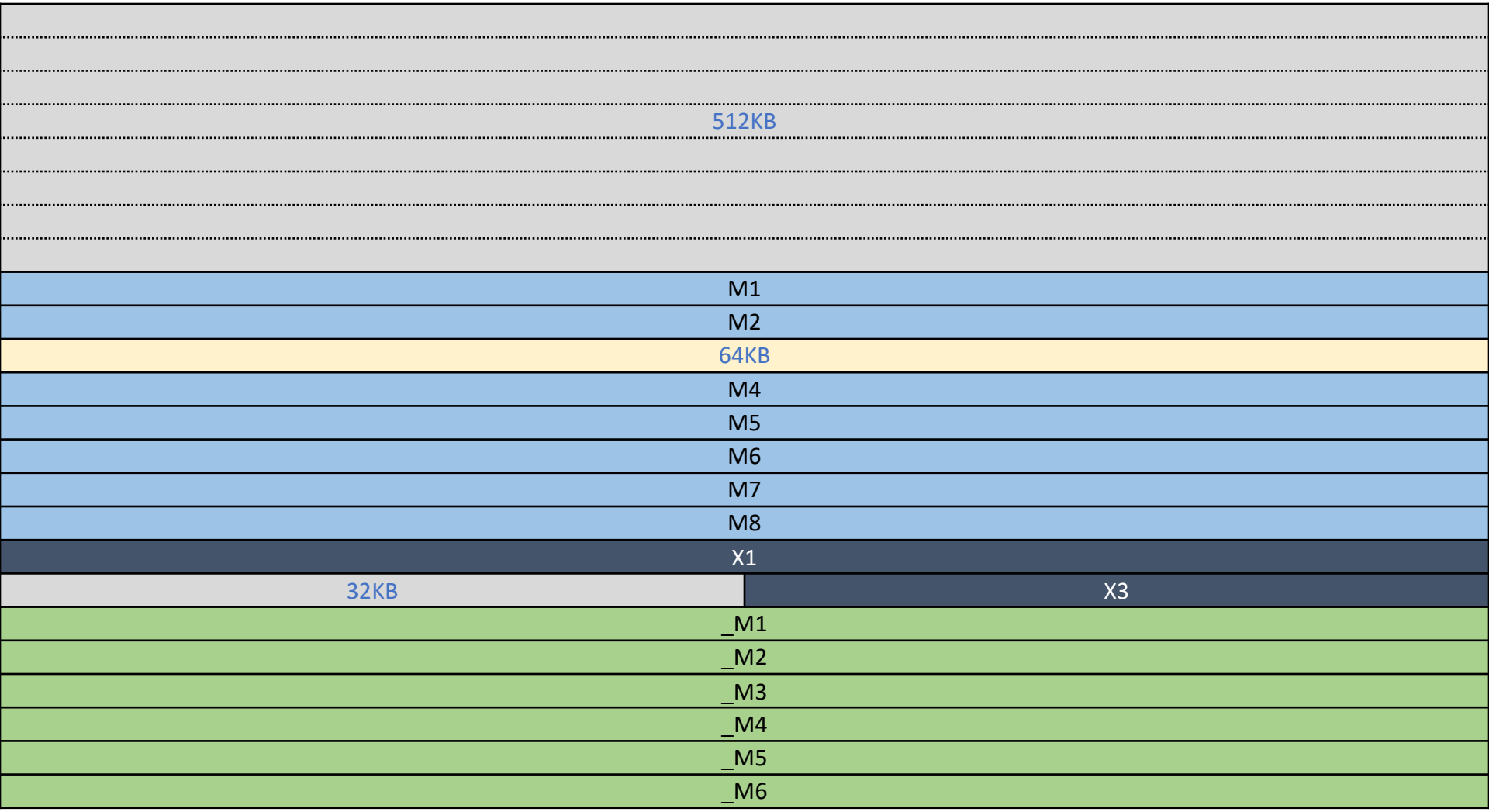
```
ReleaseAll(L); // to avoid going out-of-memory later
```



Phys Feng Shui step 6/8

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

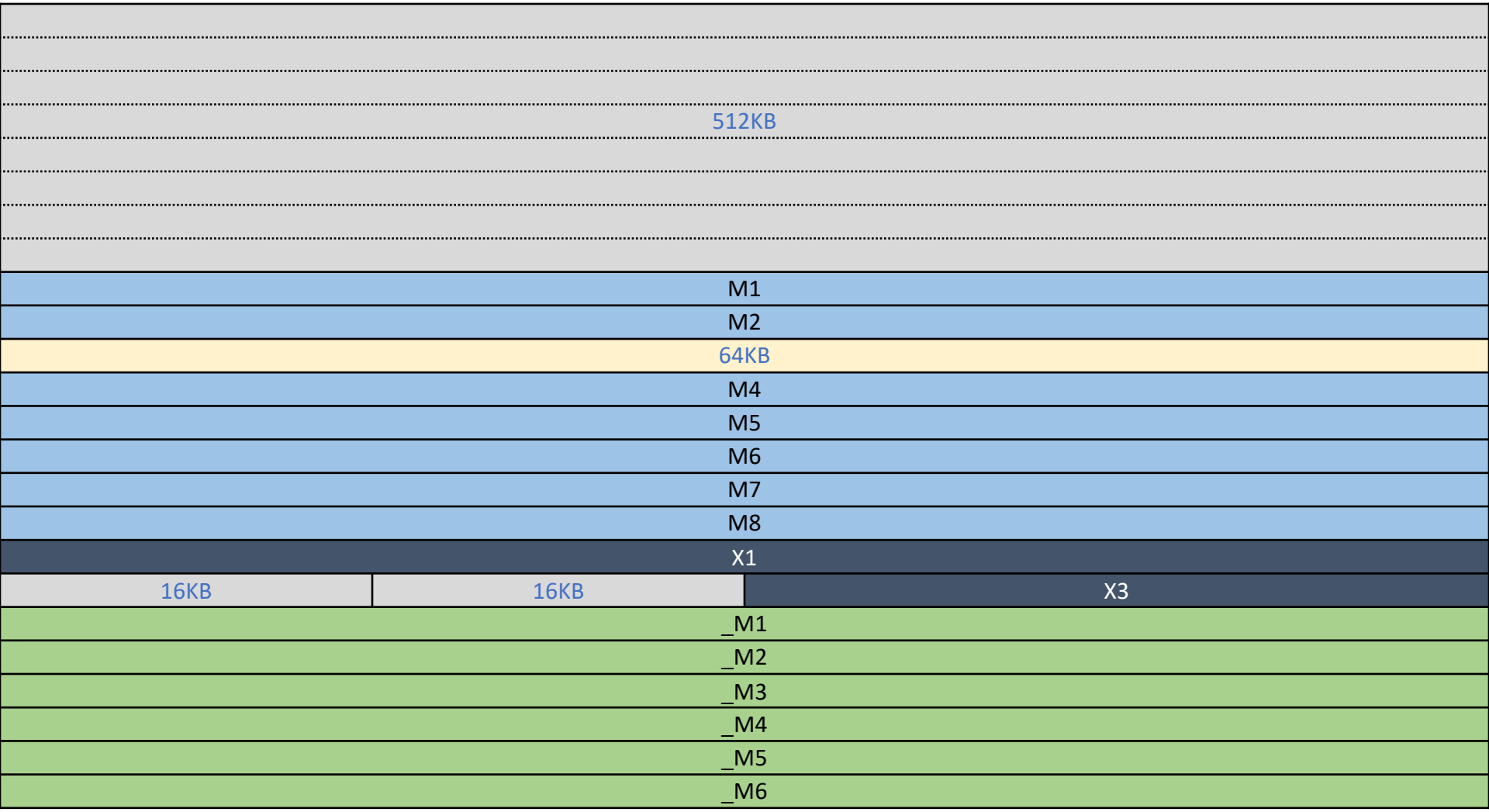
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

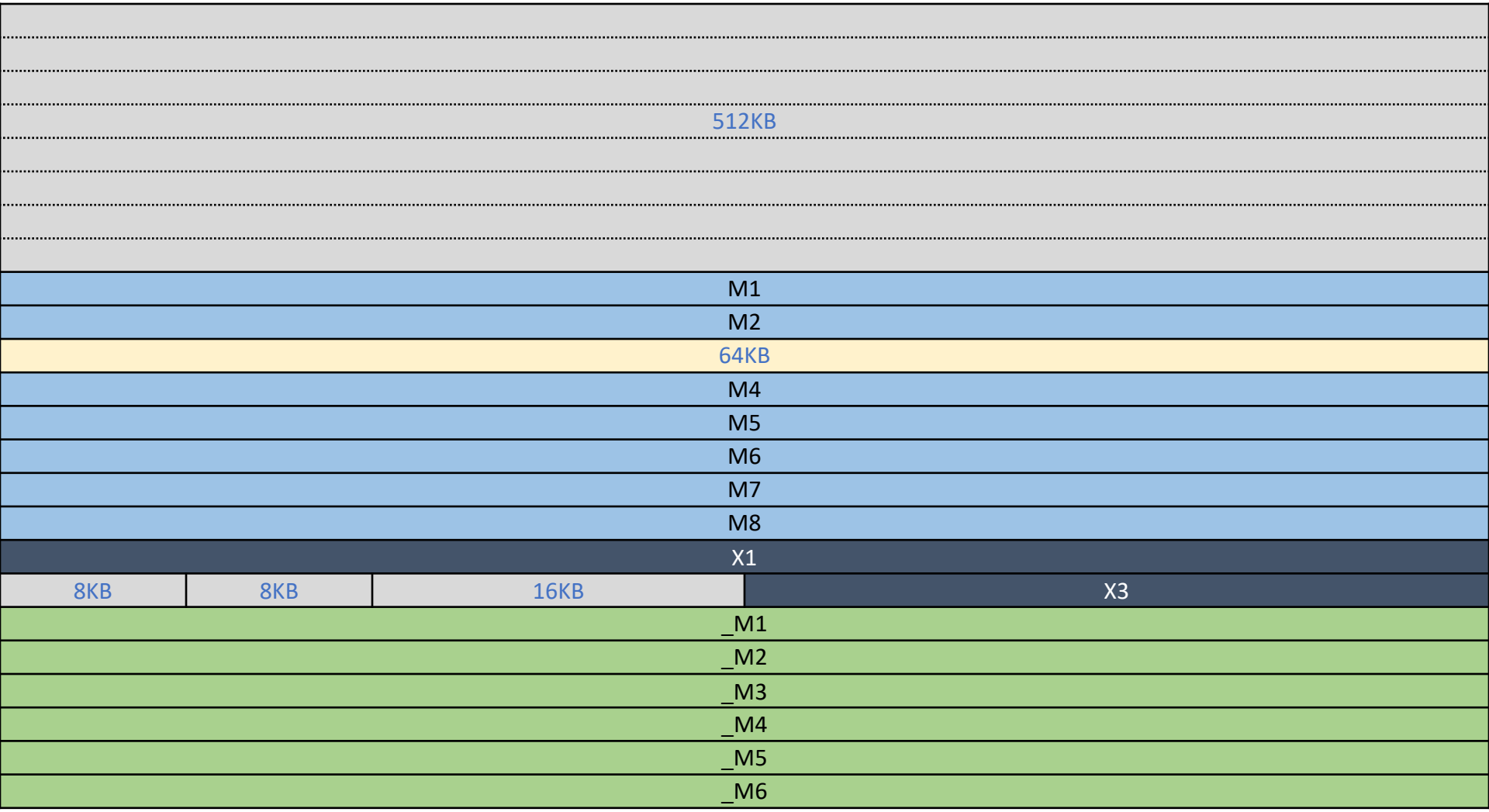
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

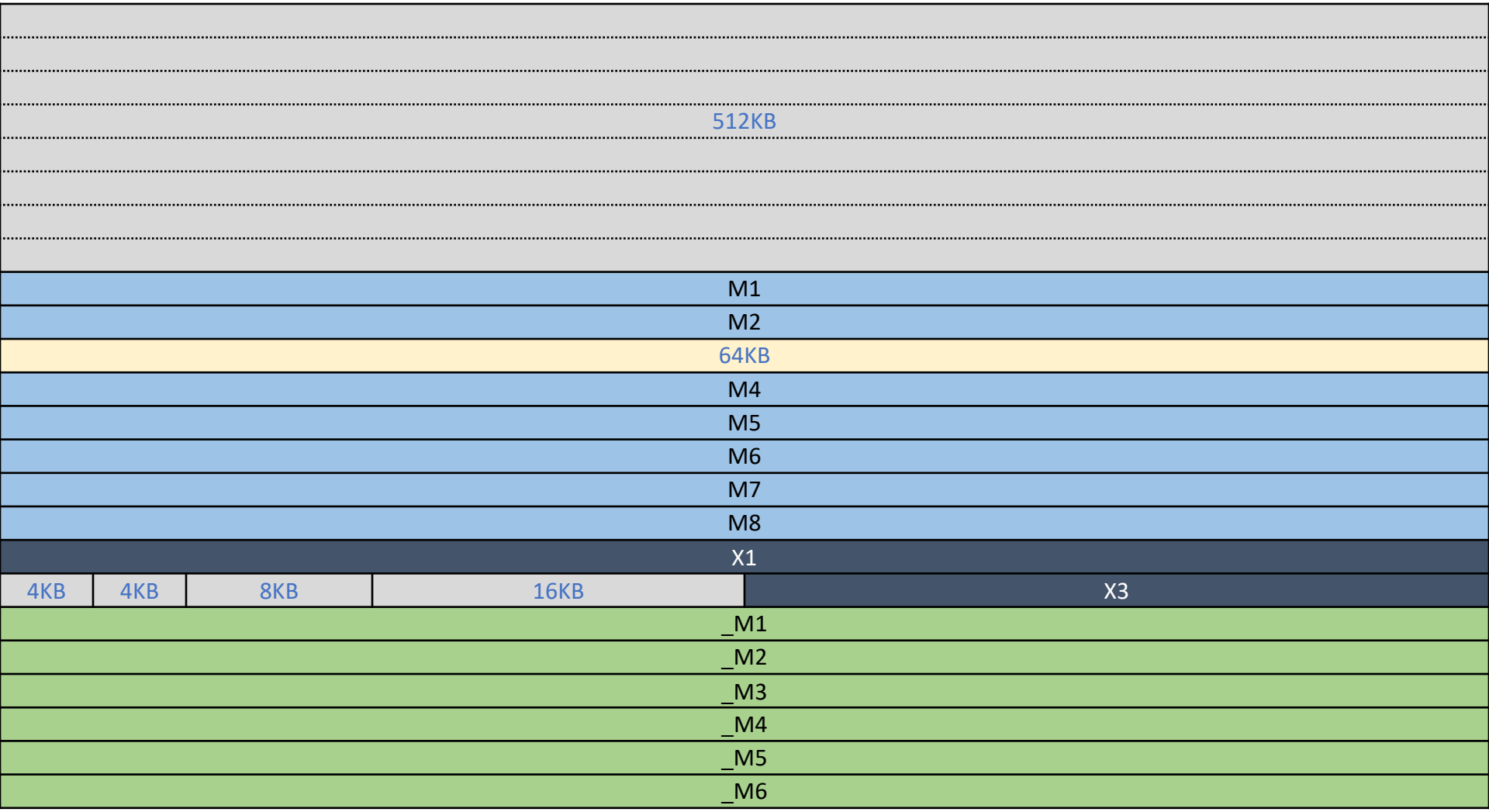
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

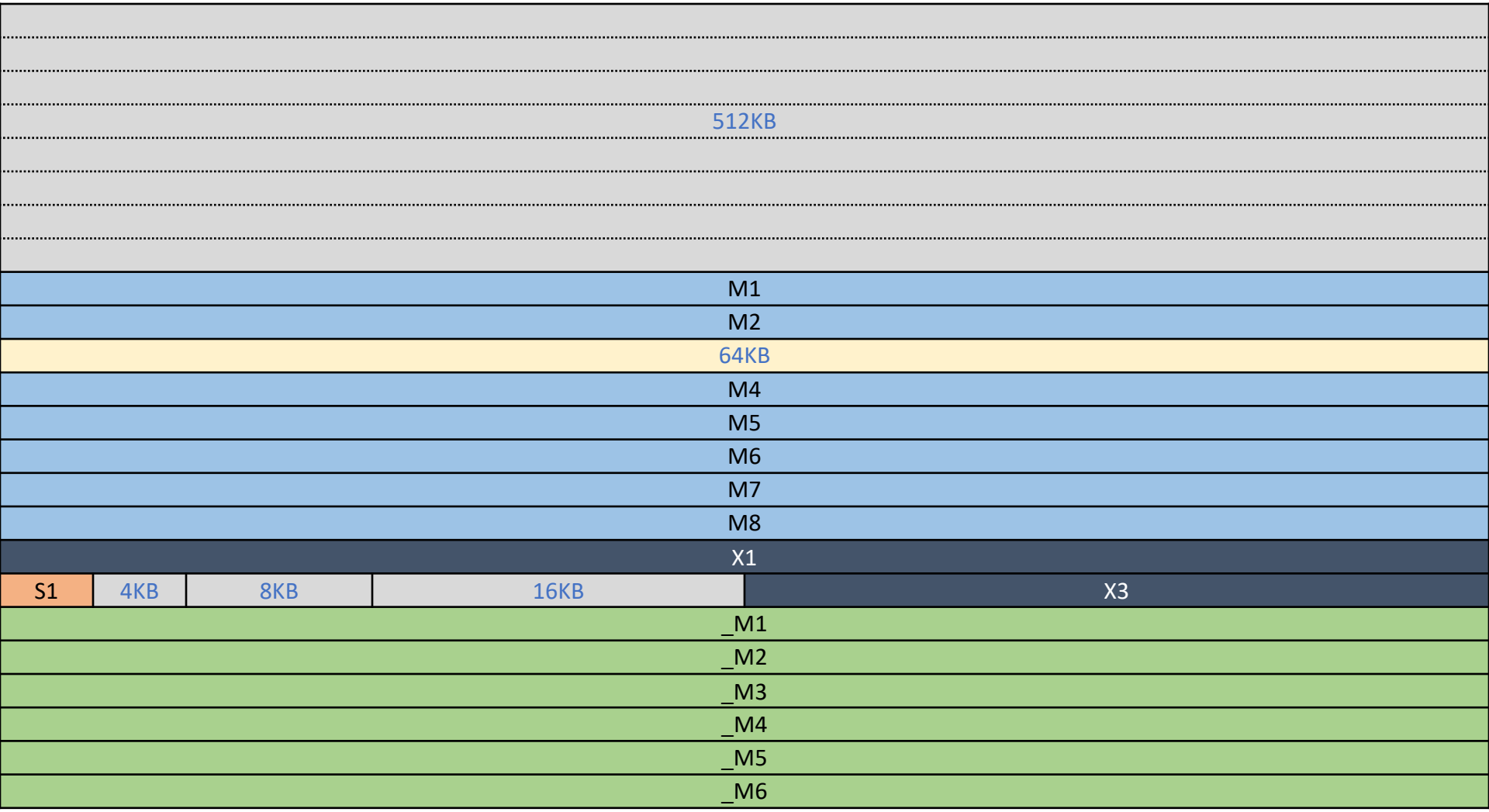
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

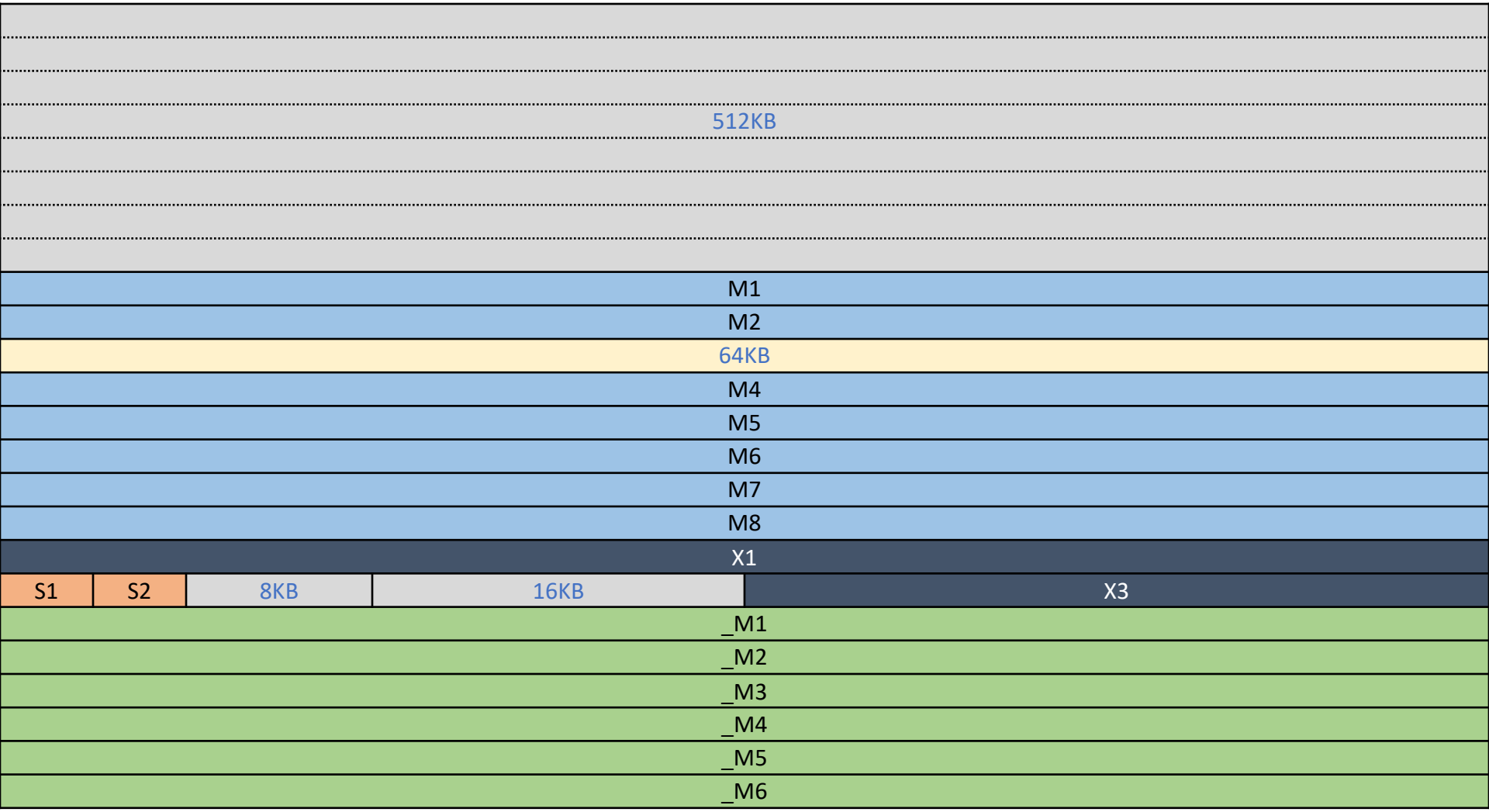
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

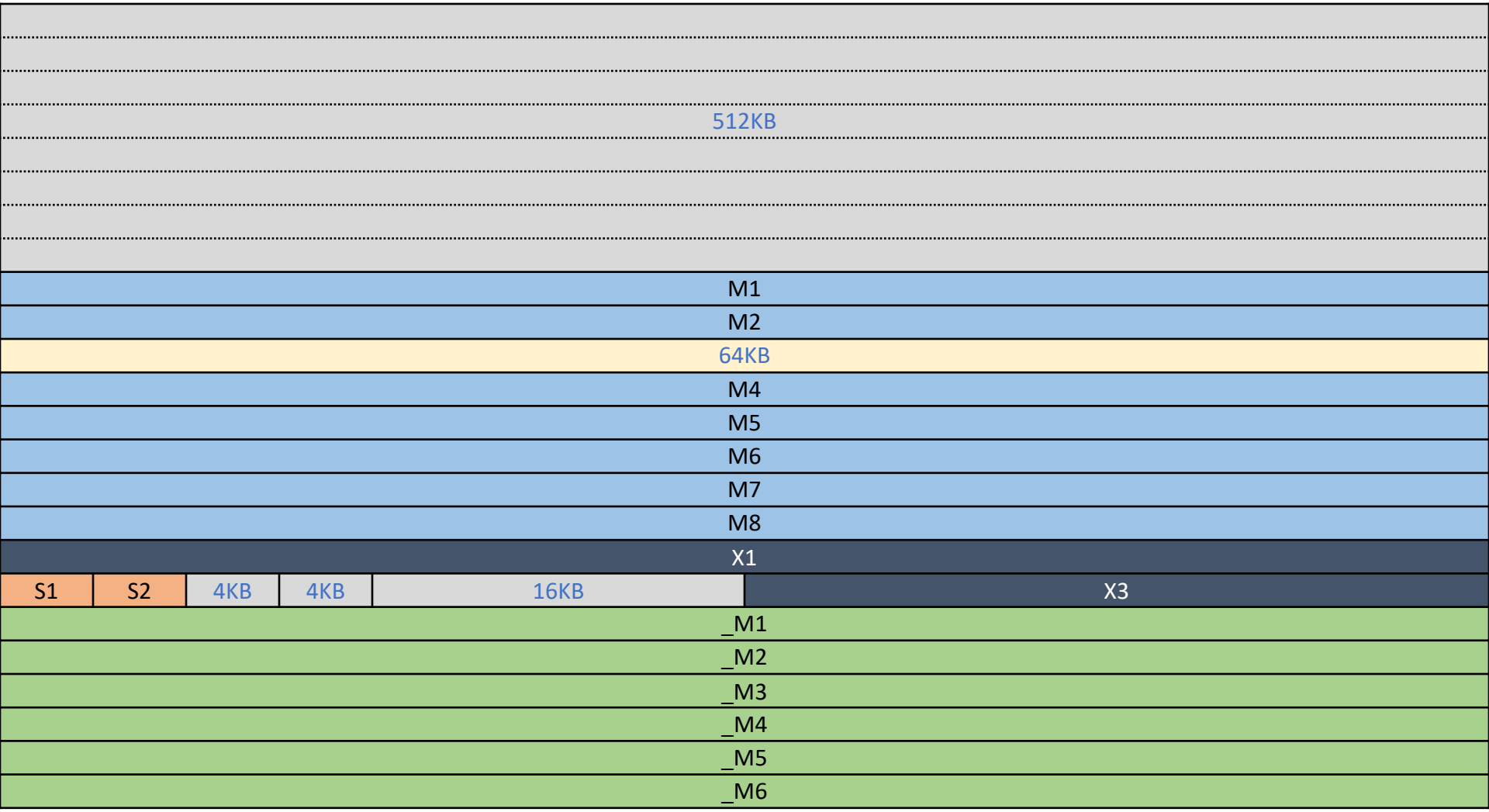
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

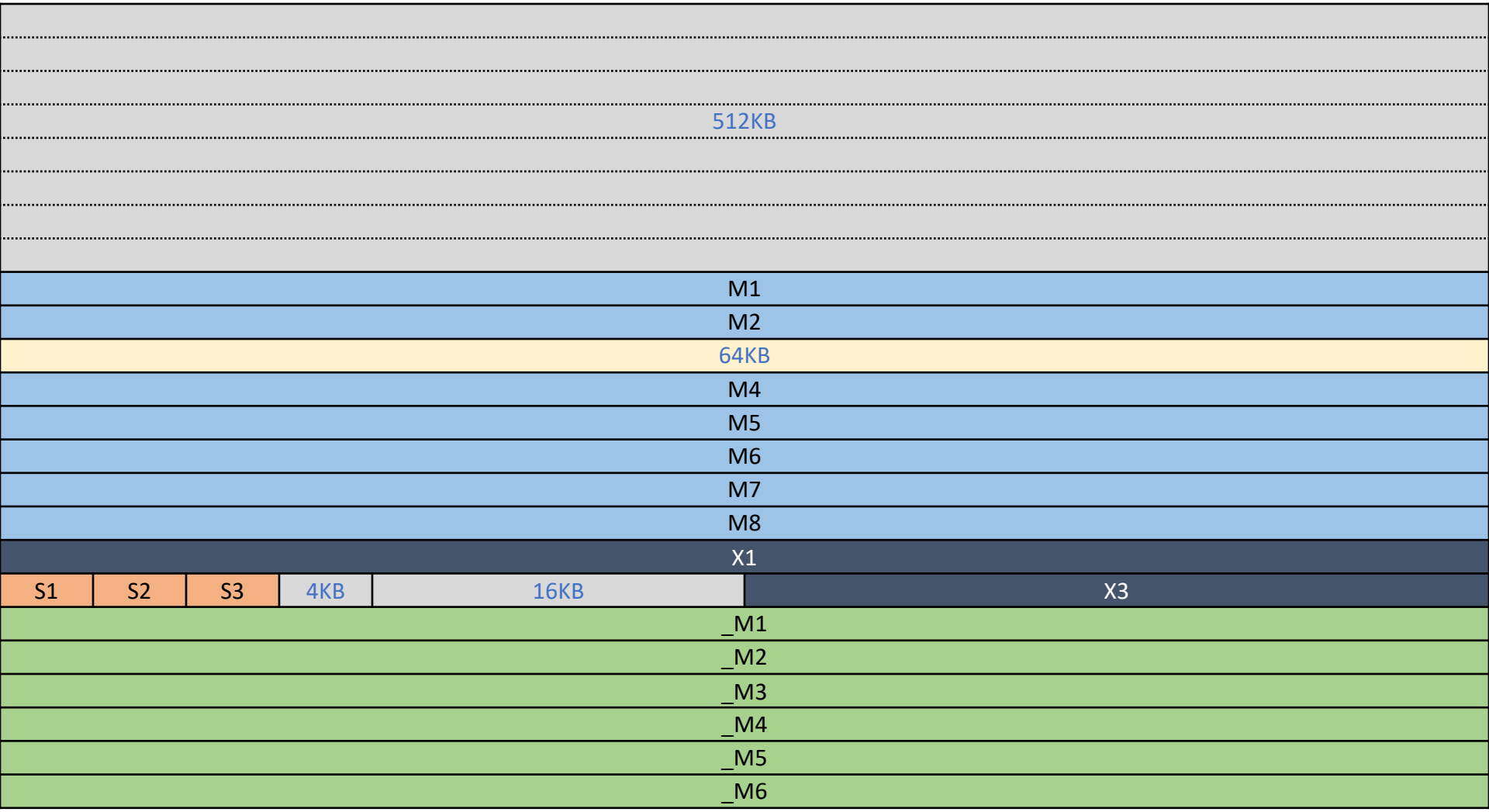
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

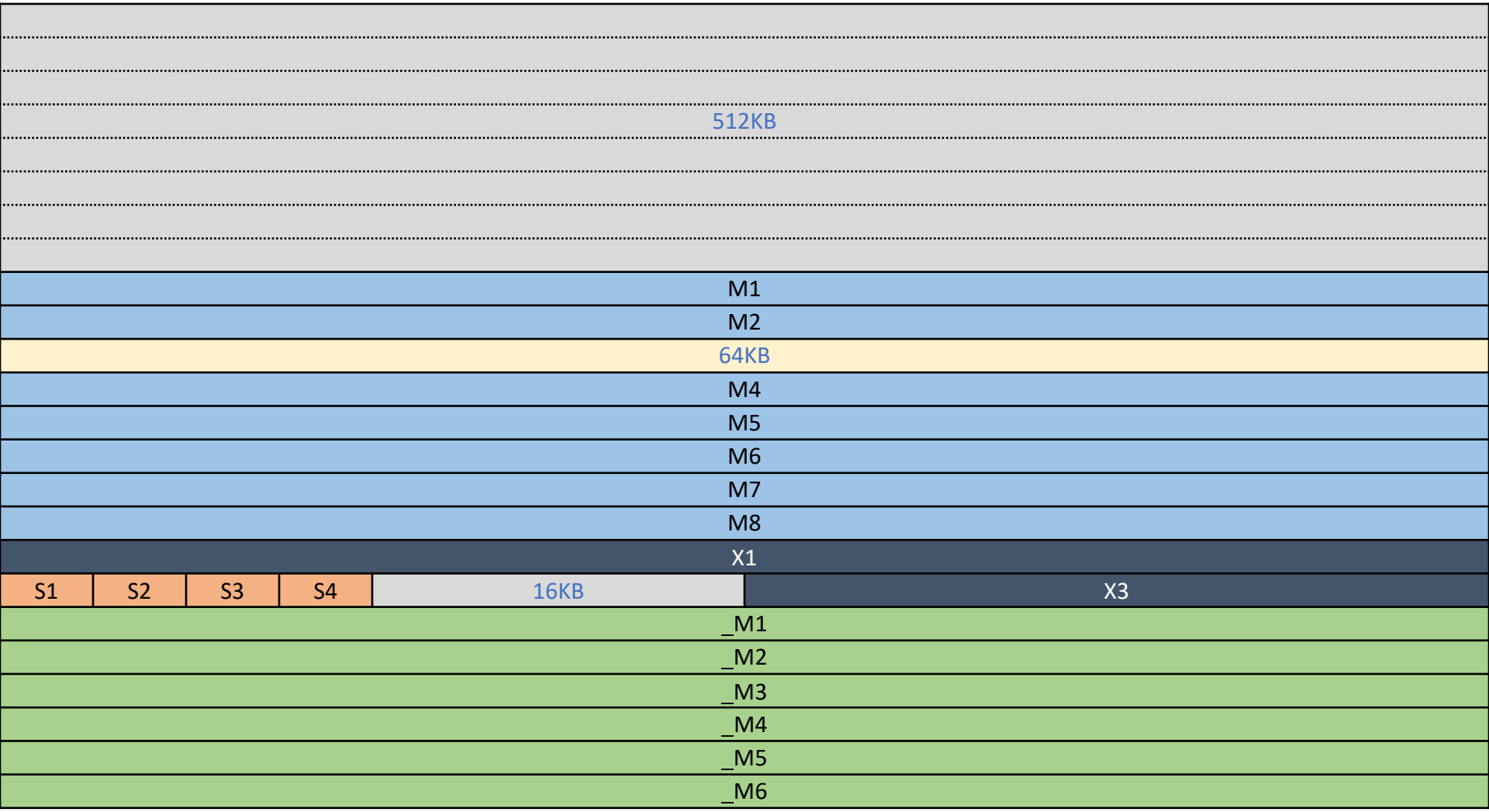
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

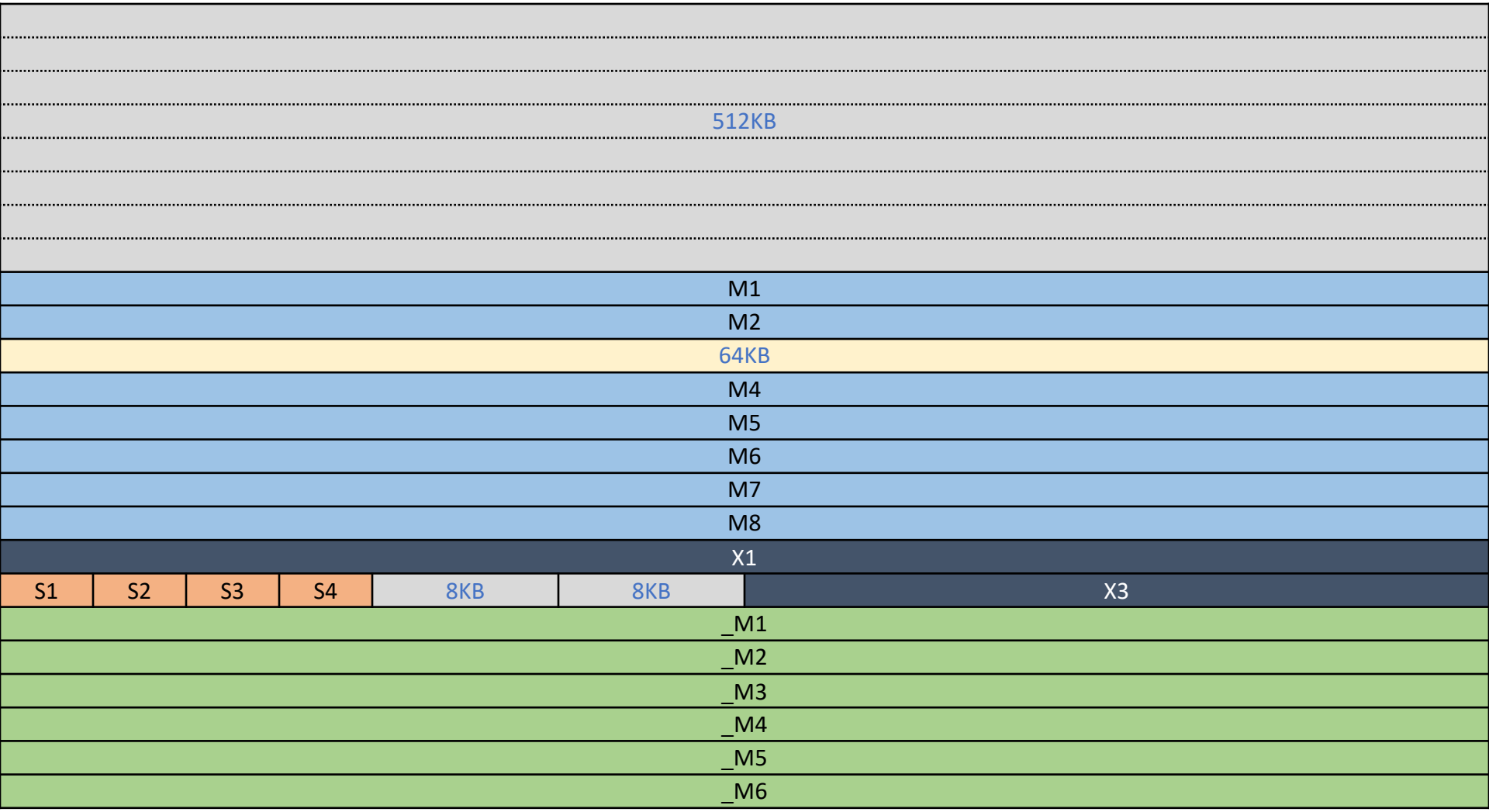
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

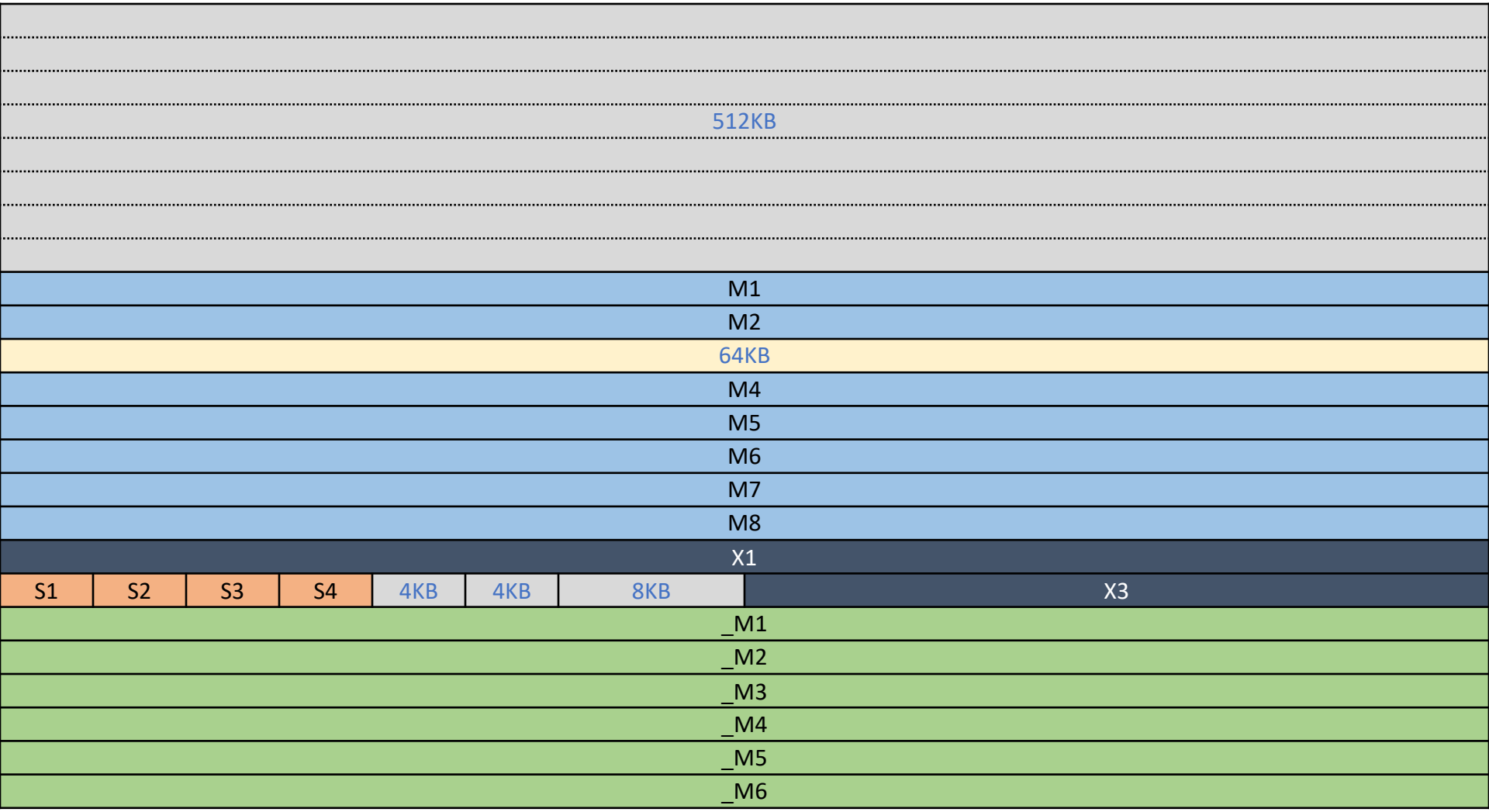
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

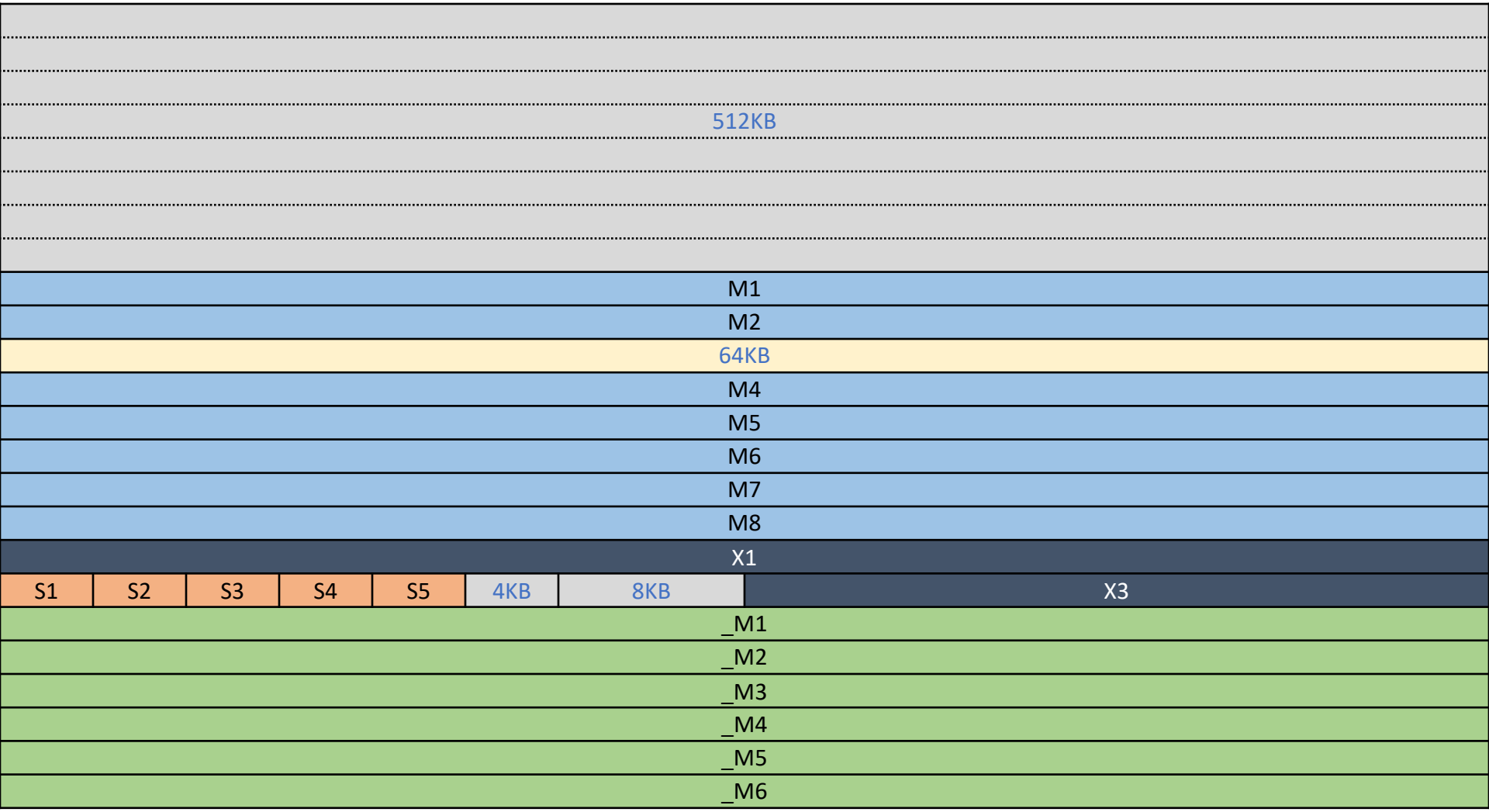
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

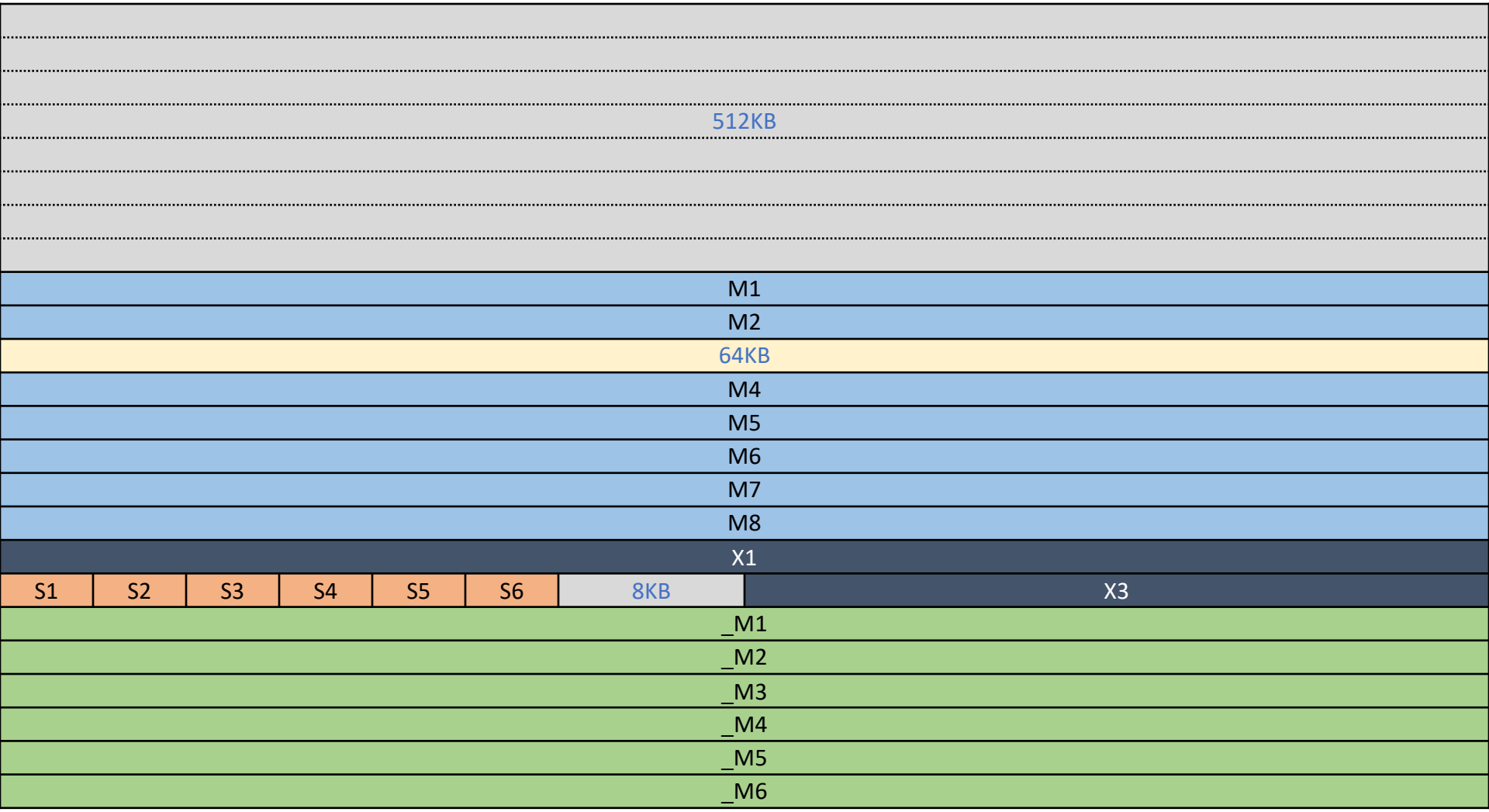
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

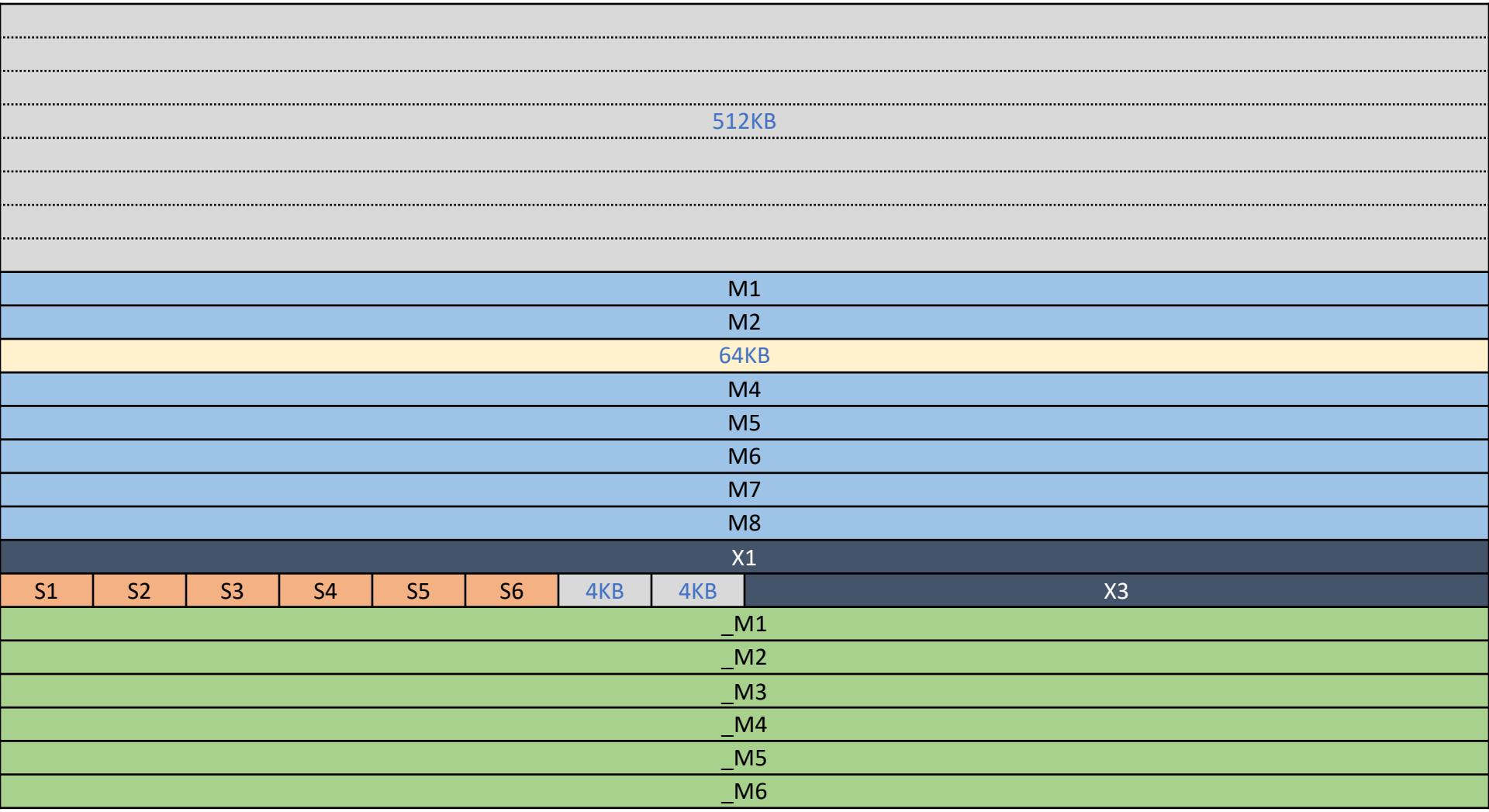
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

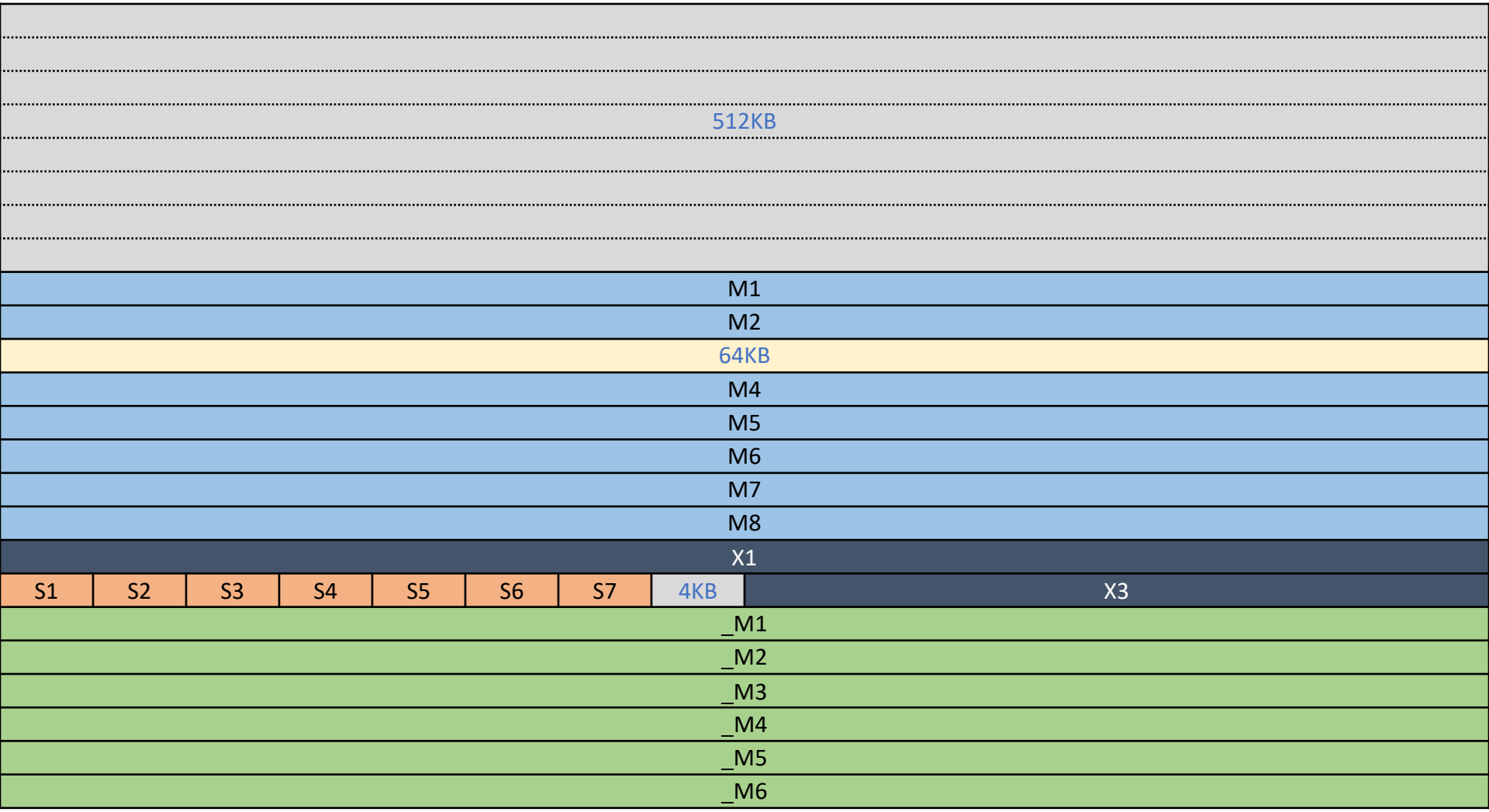
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

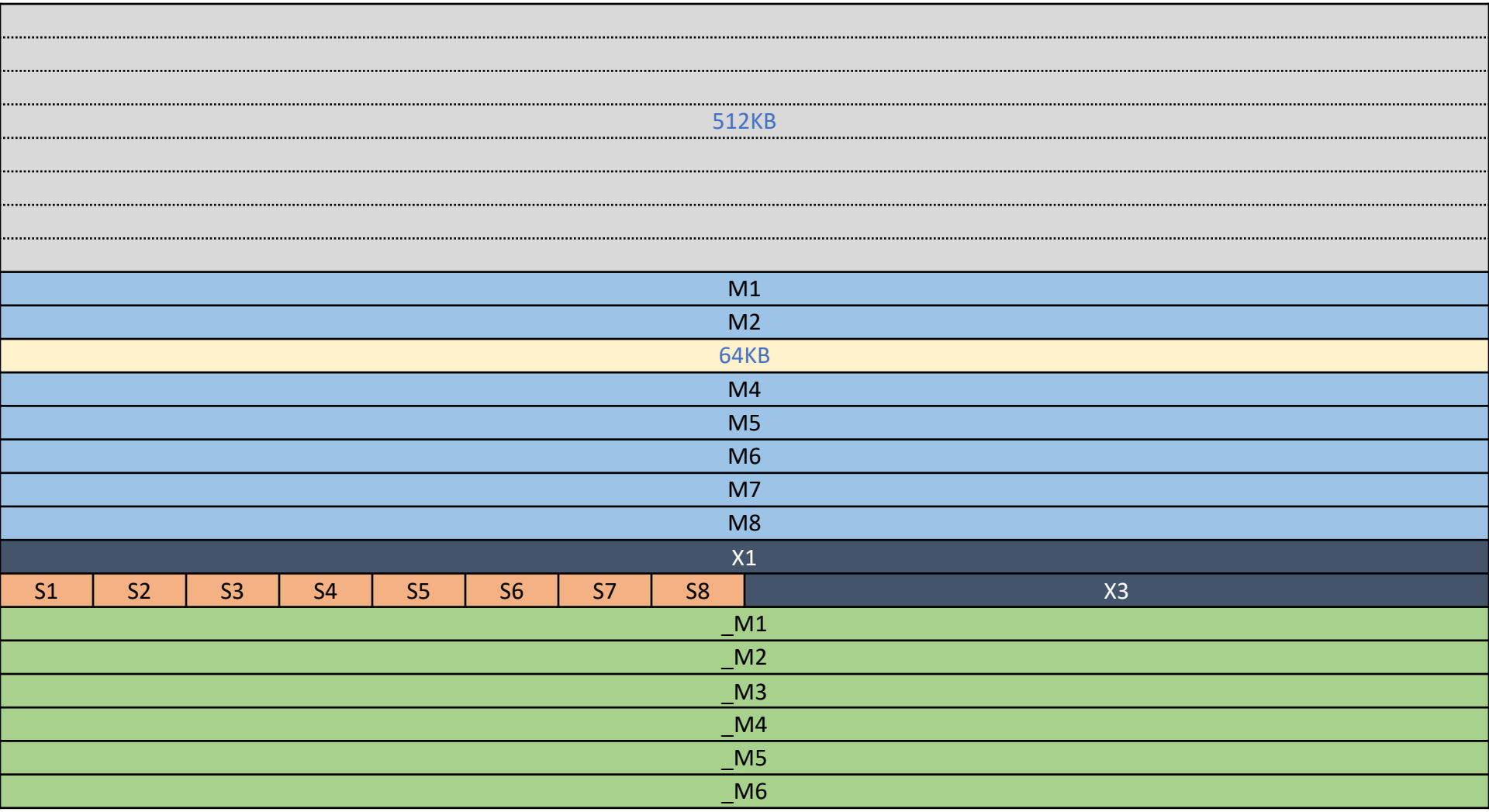
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

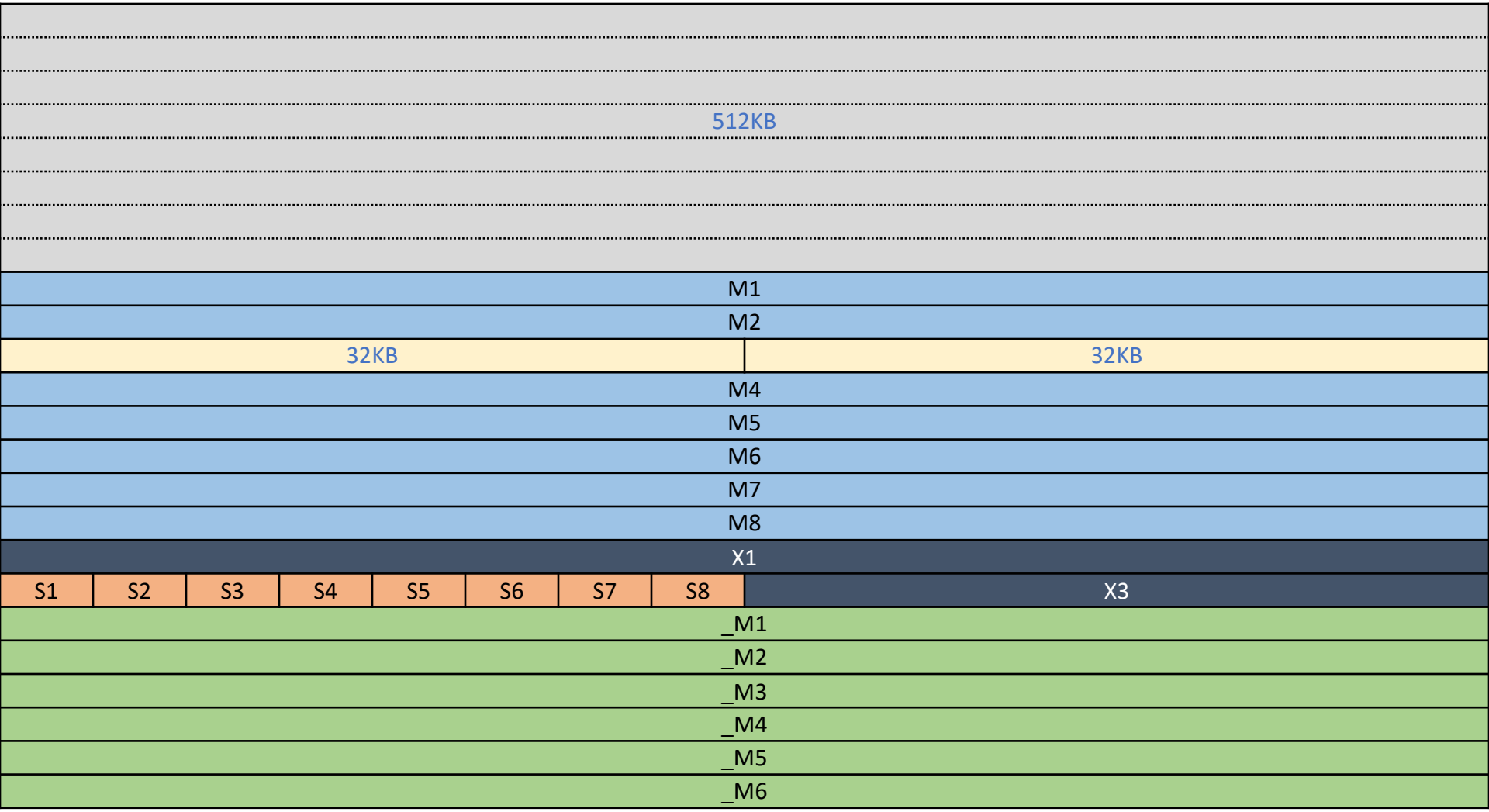
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

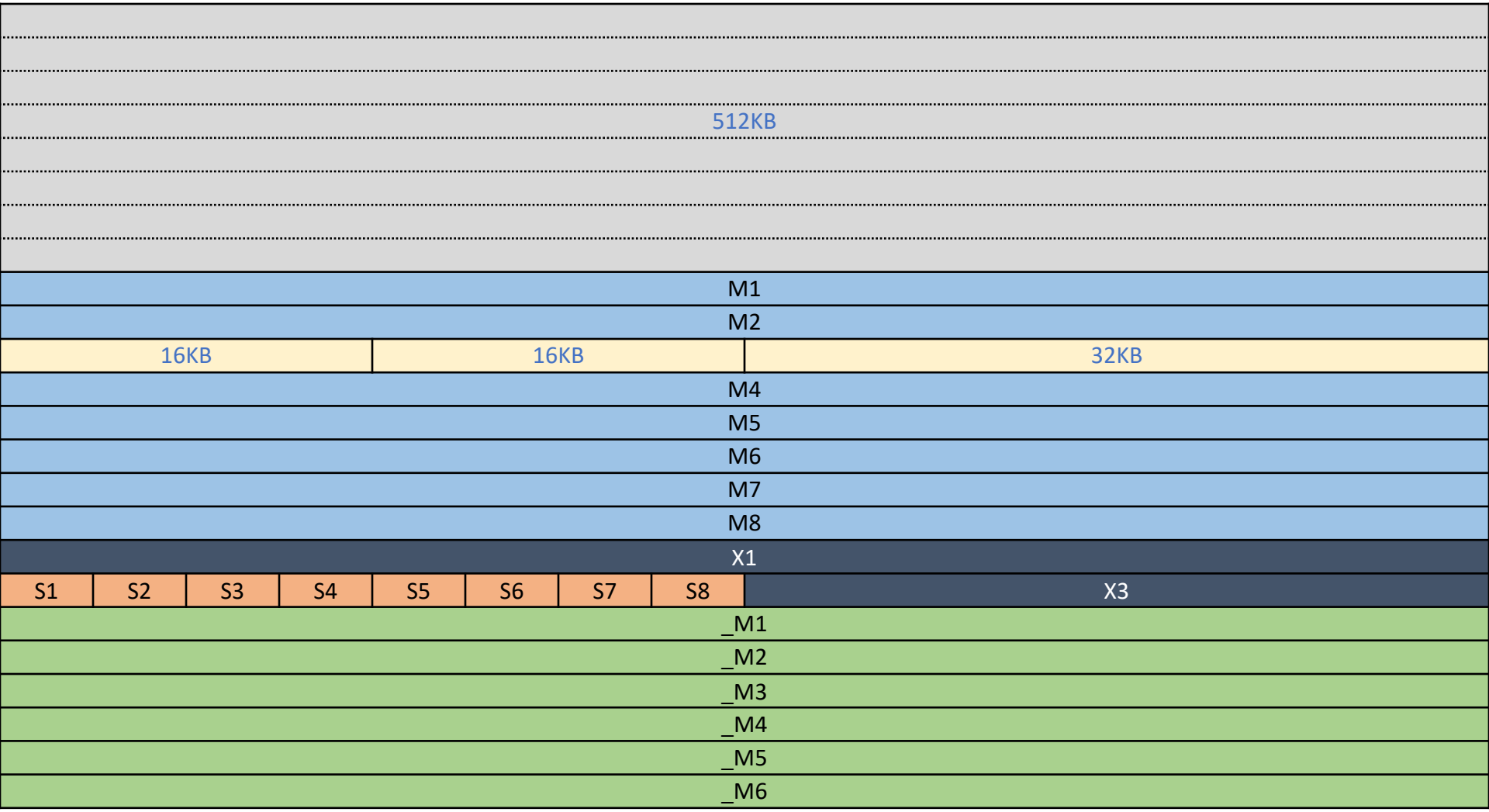
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

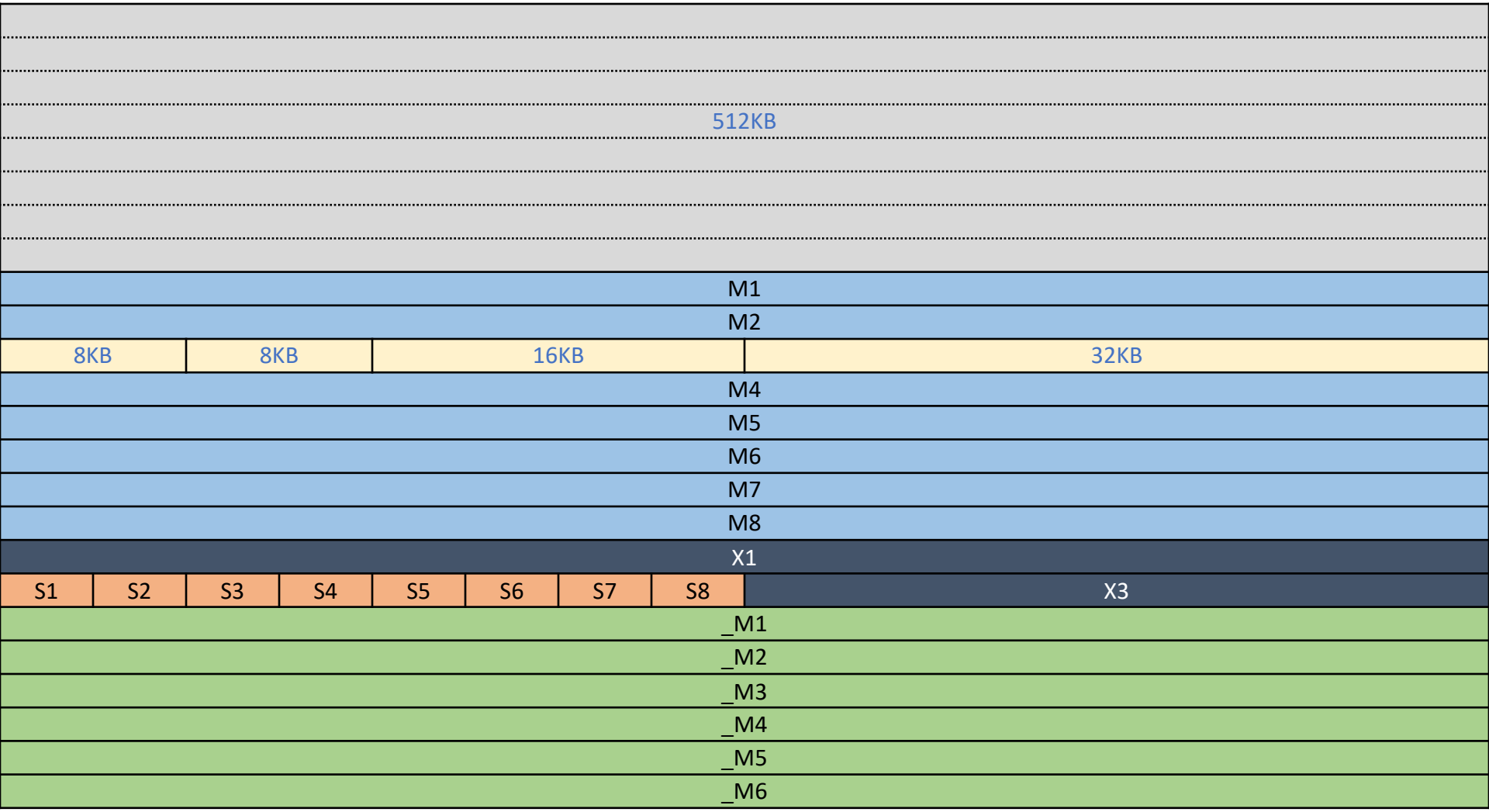
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

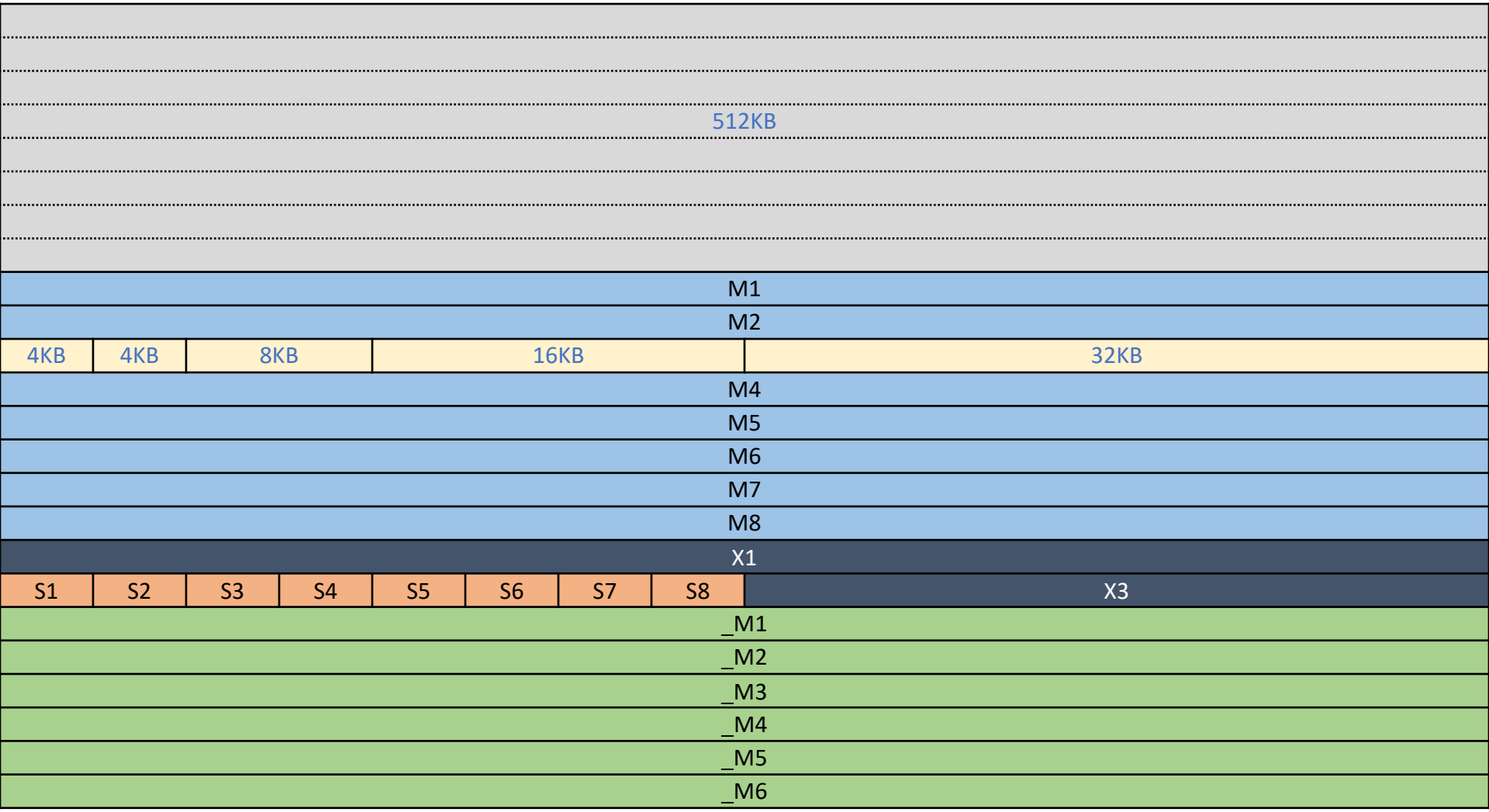
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

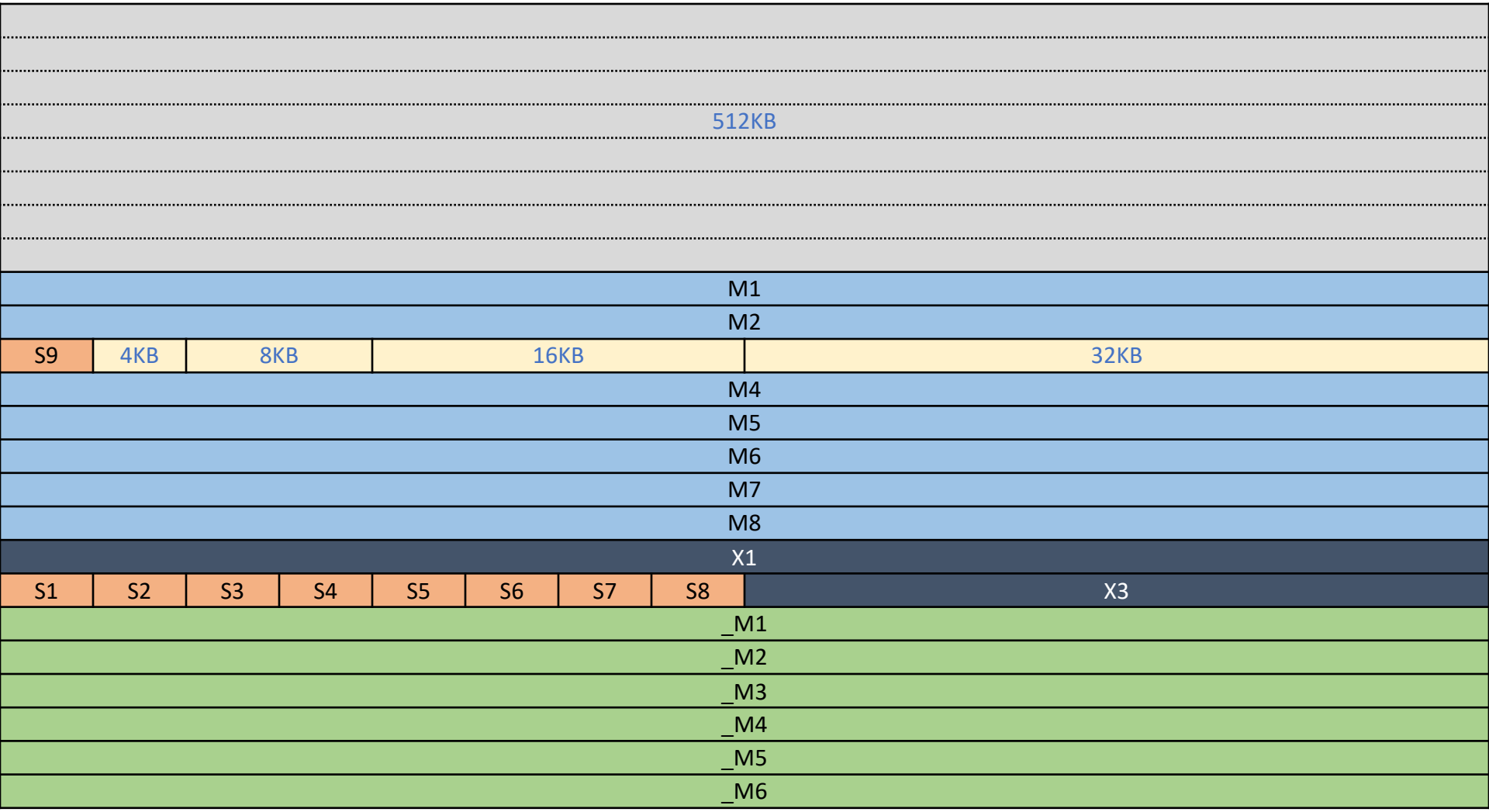
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 6/8

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

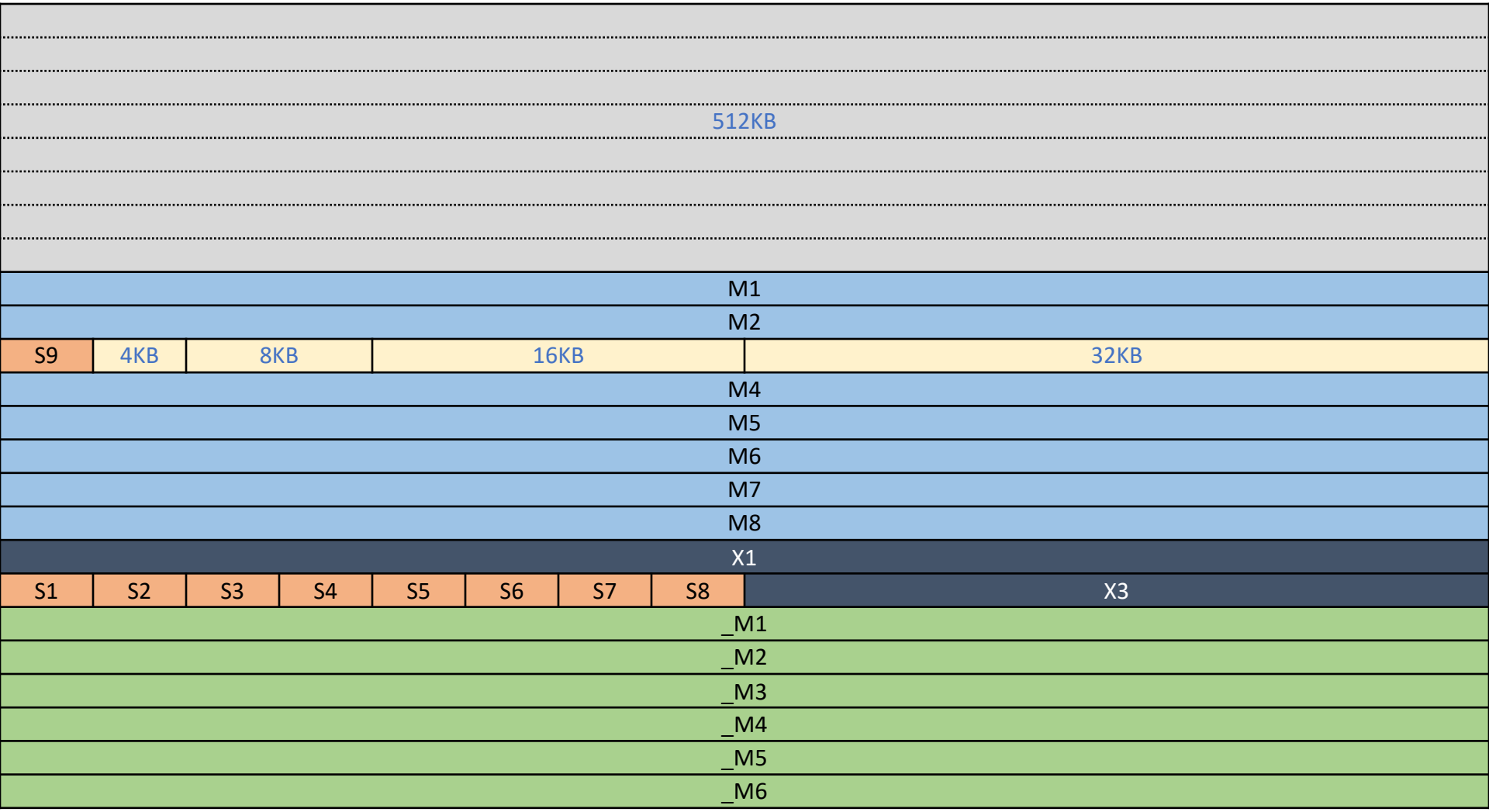
```
Land(S); // allocate 4KB pages until the 64KB is used
```



Phys Feng Shui step 7/8

Pad *small* chunks until the vulnerable page

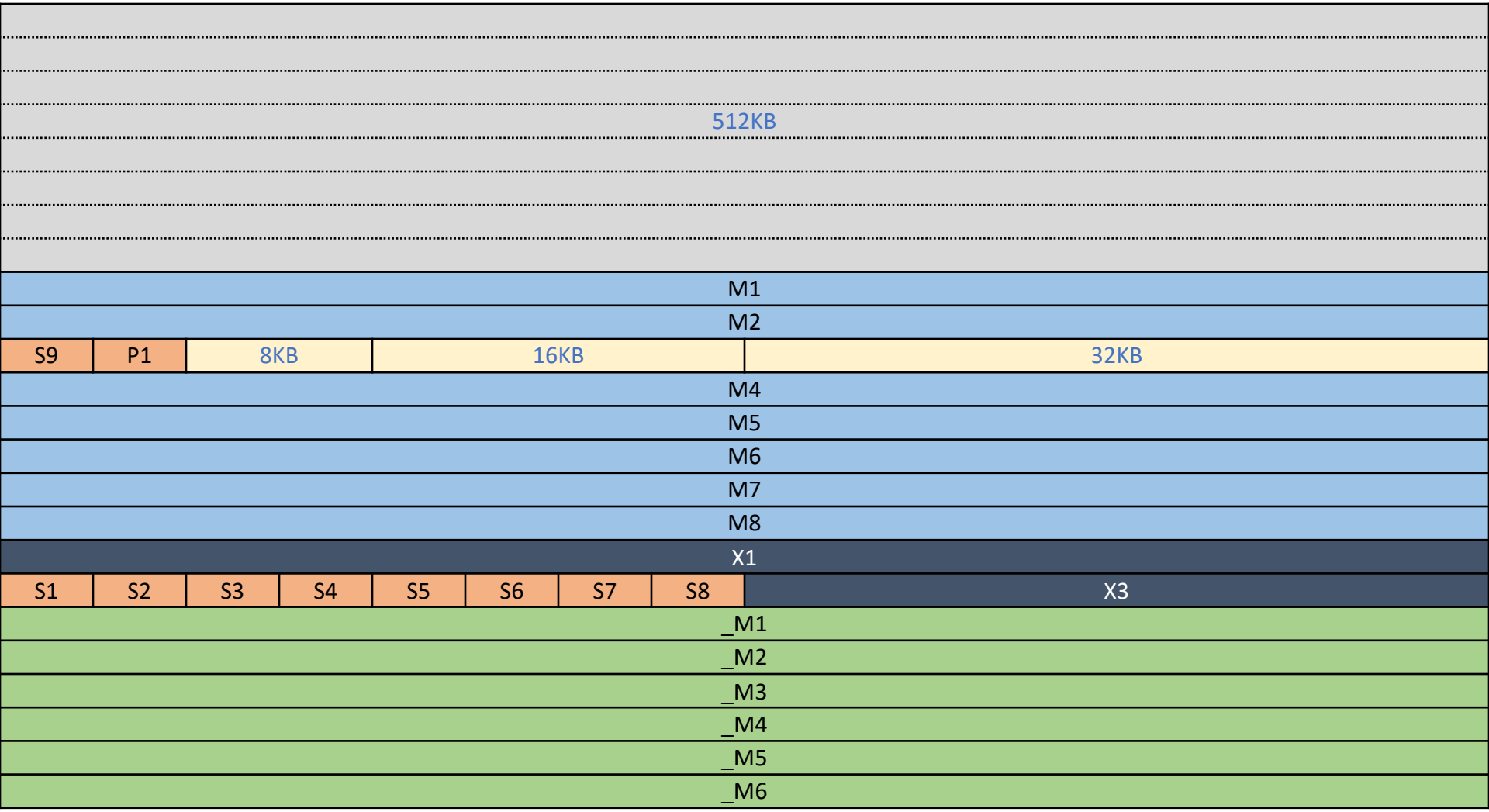
```
Pad(P); // insert padding until vulnerable page
```



Phys Feng Shui step 7/8

Pad *small* chunks until the vulnerable page

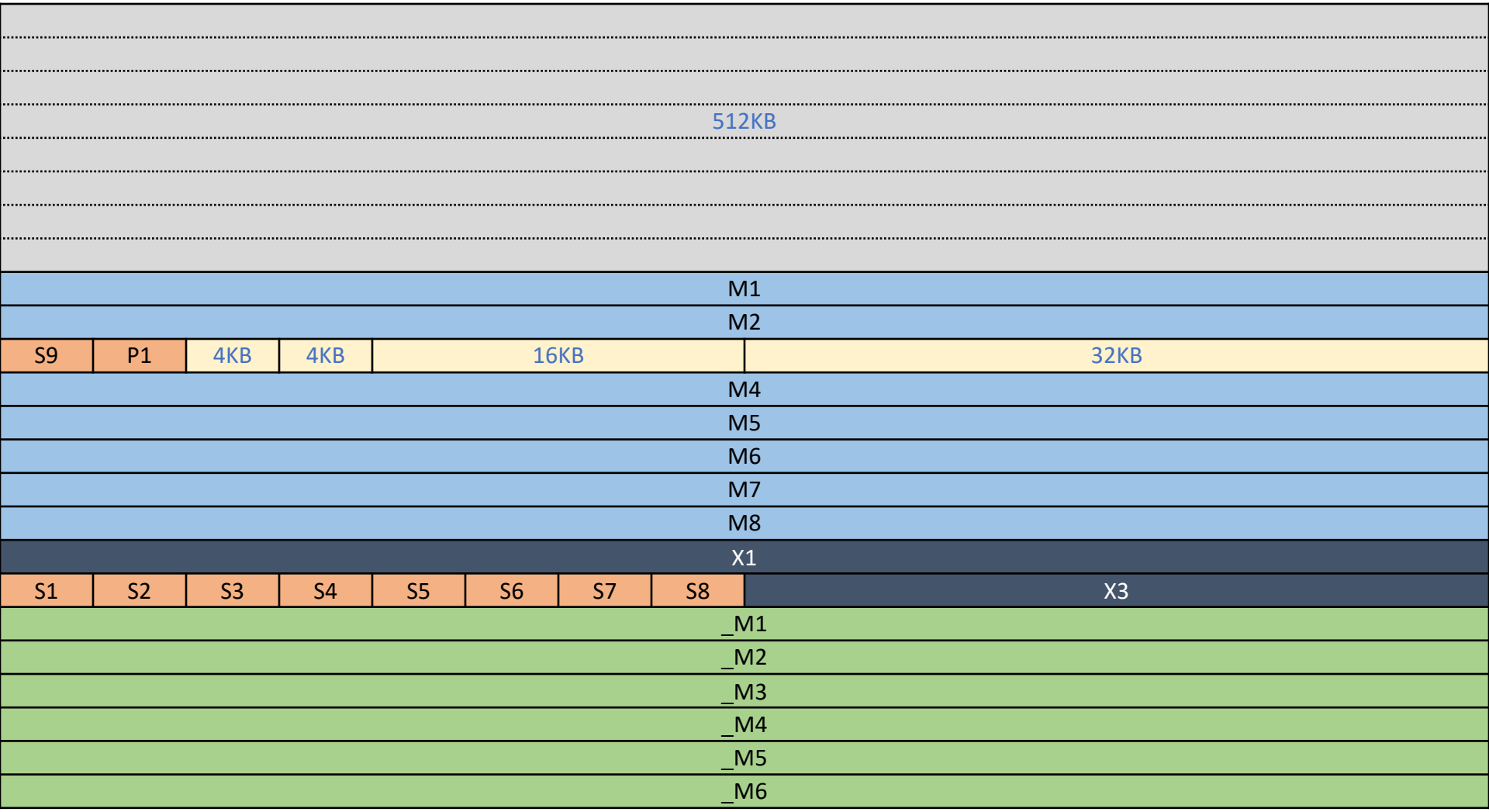
```
Pad(P); // insert padding until vulnerable page
```



Phys Feng Shui step 7/8

Pad *small* chunks until the vulnerable page

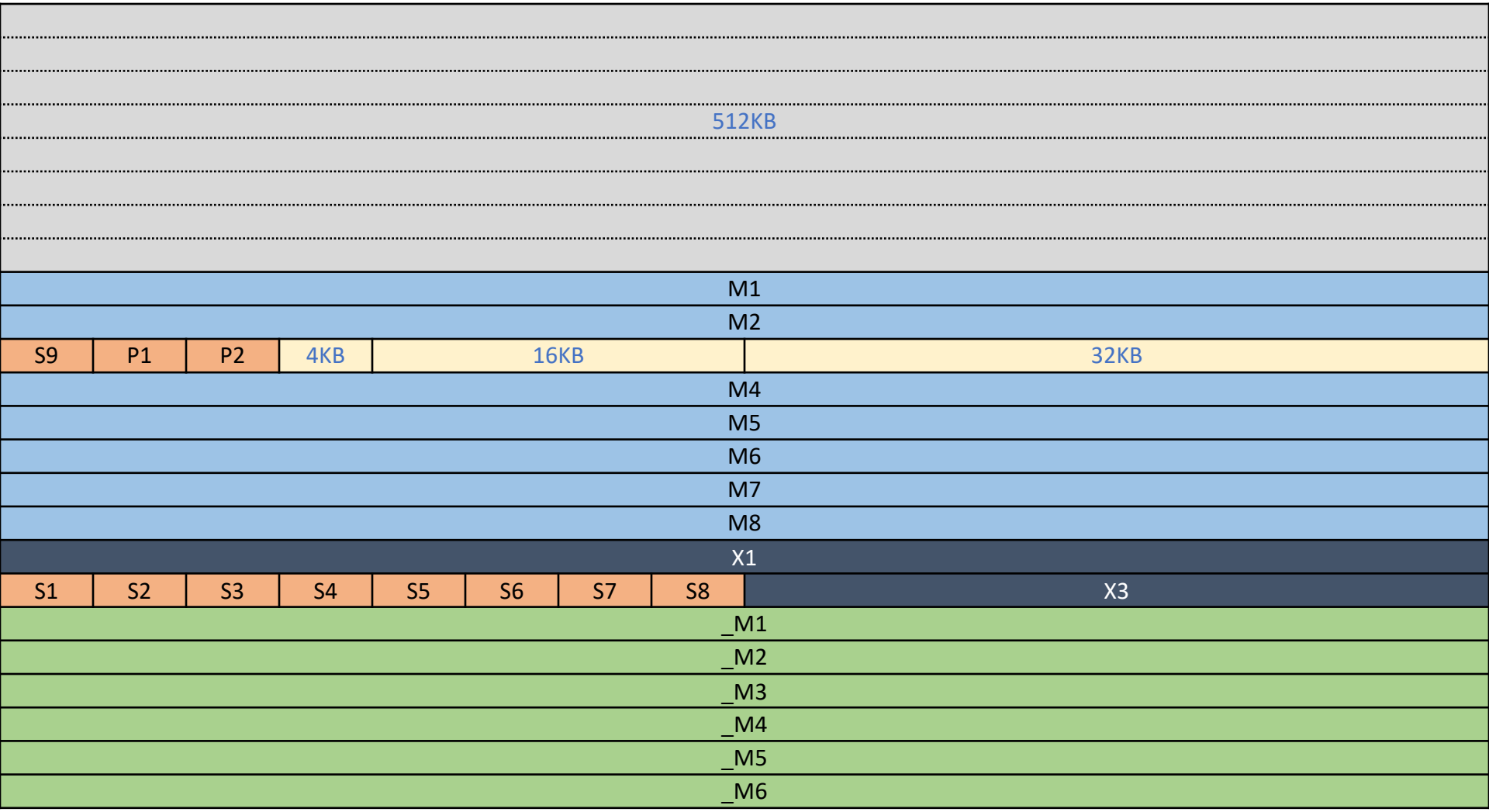
```
Pad(P); // insert padding until vulnerable page
```



Phys Feng Shui step 7/8

Pad *small* chunks until the vulnerable page

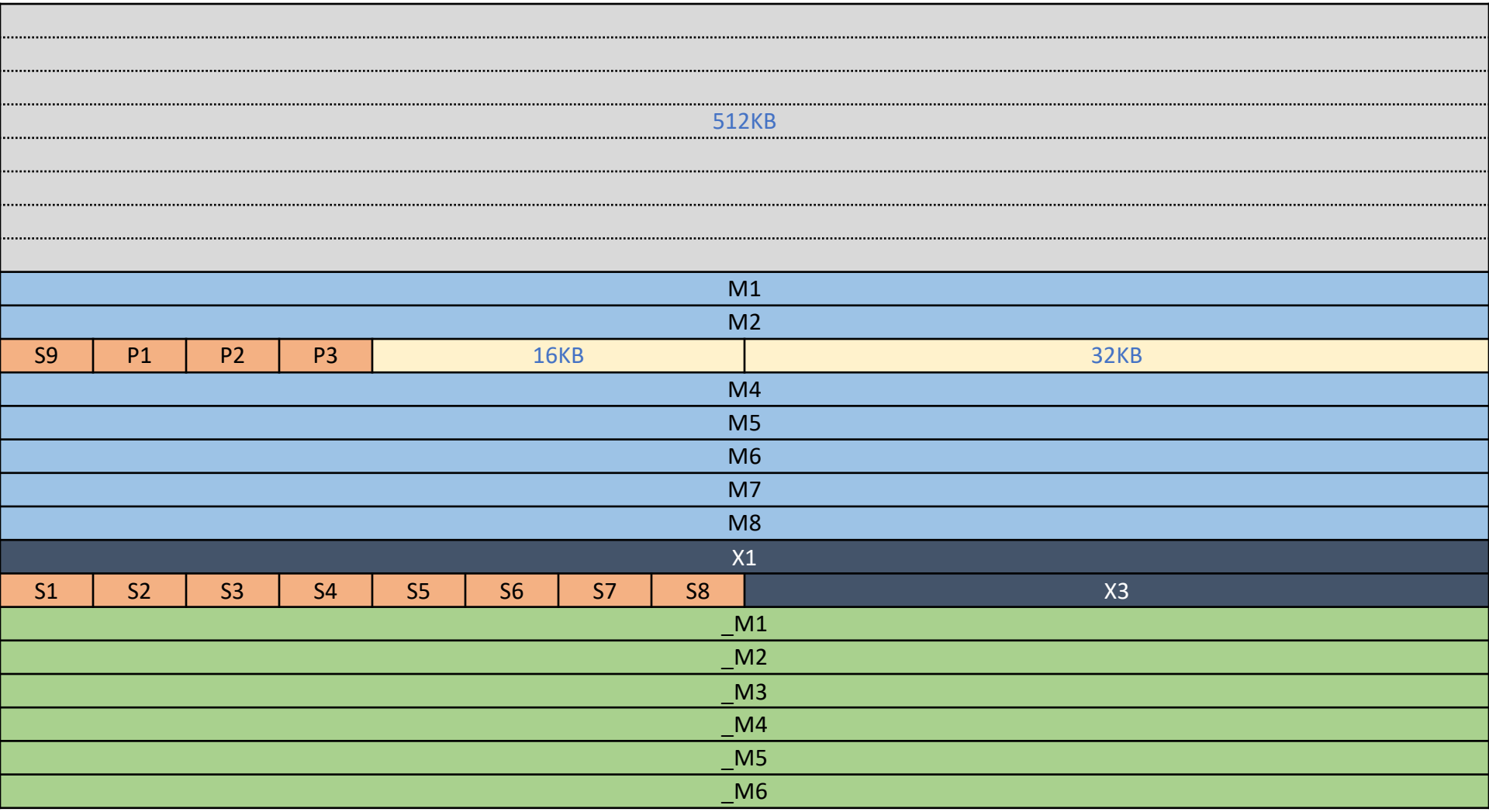
```
Pad(P); // insert padding until vulnerable page
```



Phys Feng Shui step 7/8

Pad *small* chunks until the vulnerable page

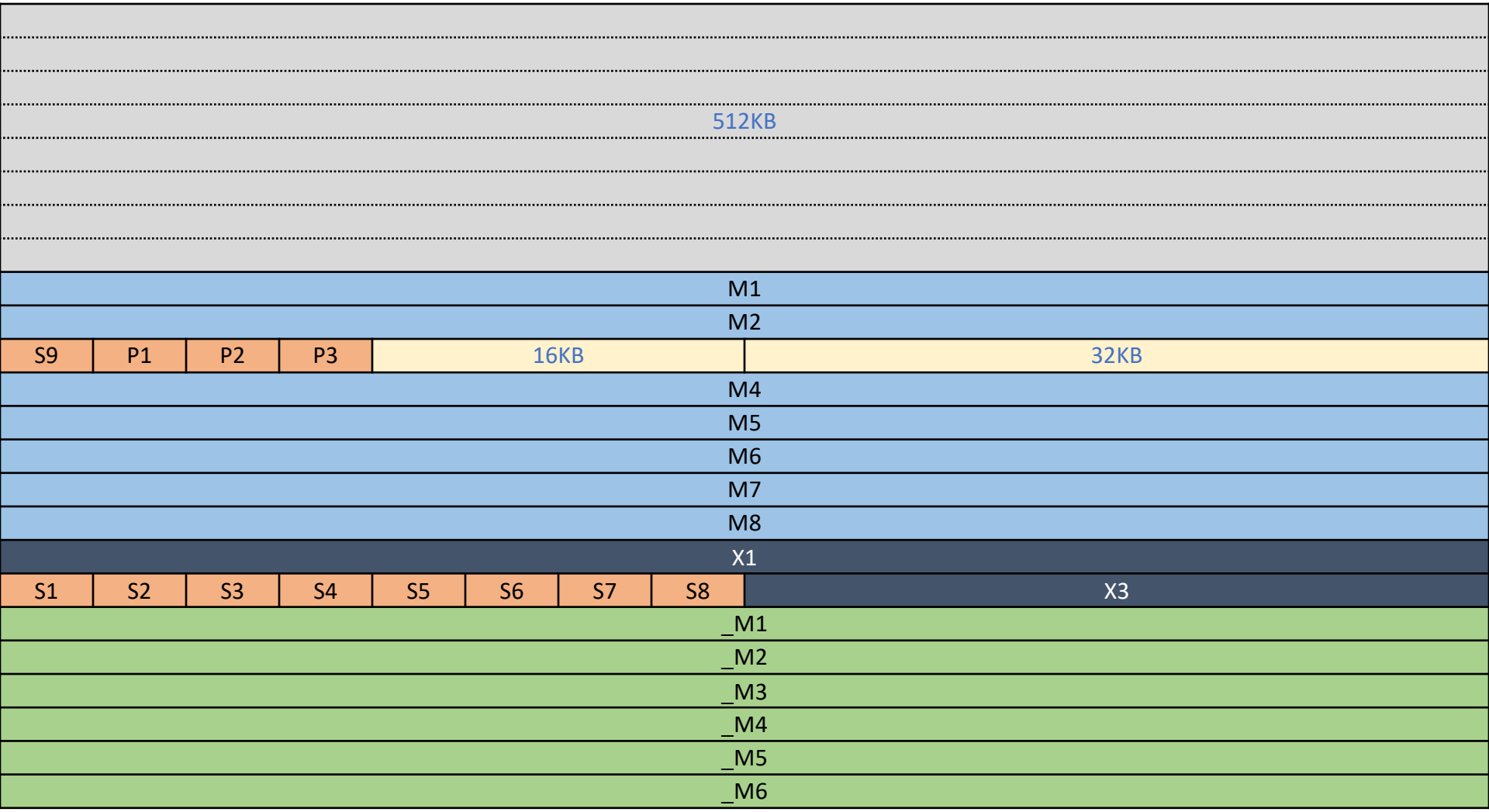
```
Pad(P); // insert padding until vulnerable page
```



Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE

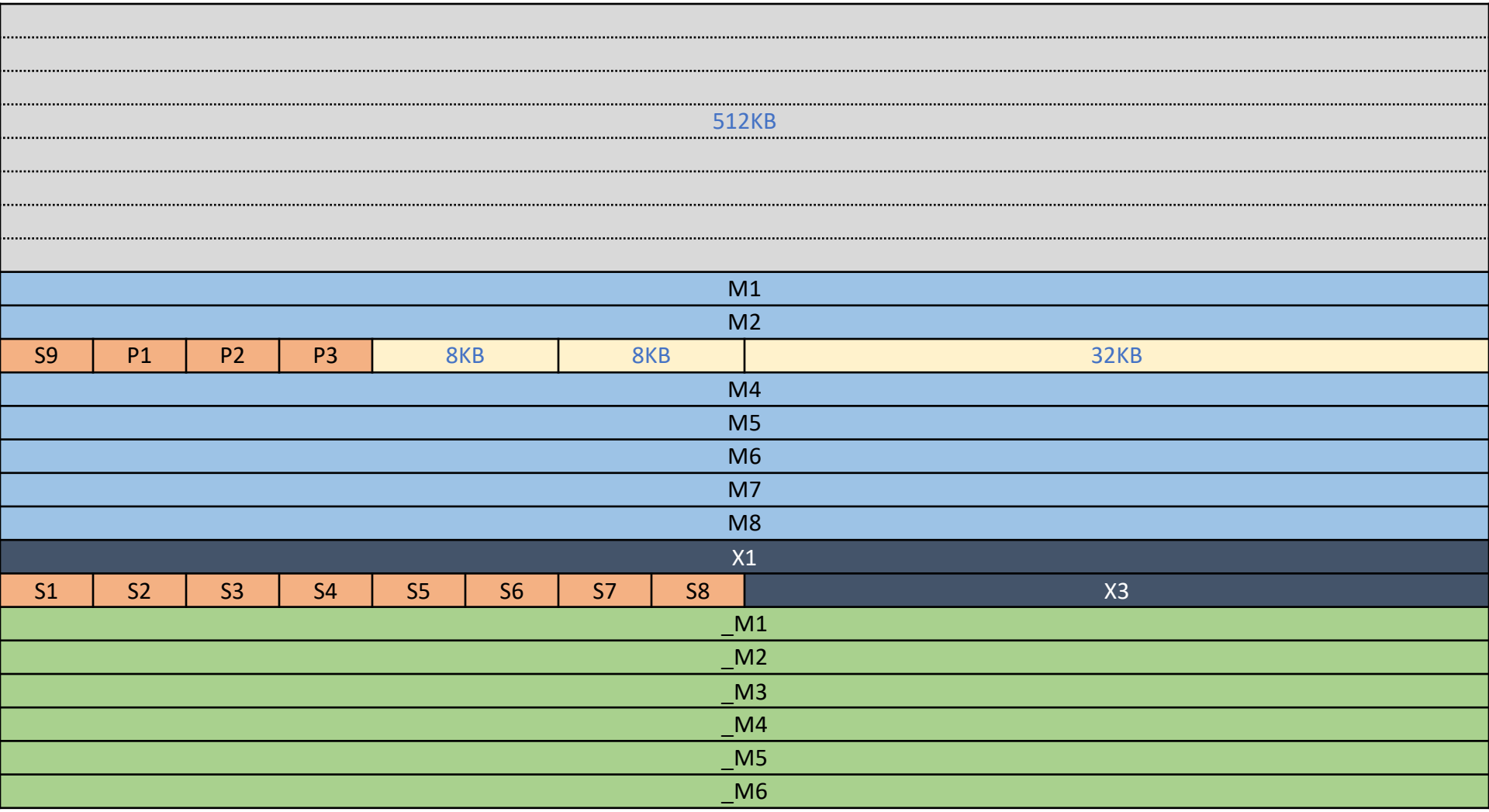
```
PT = mmap(MAP_FIXED); // Force a Page Table allocation
```



Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE

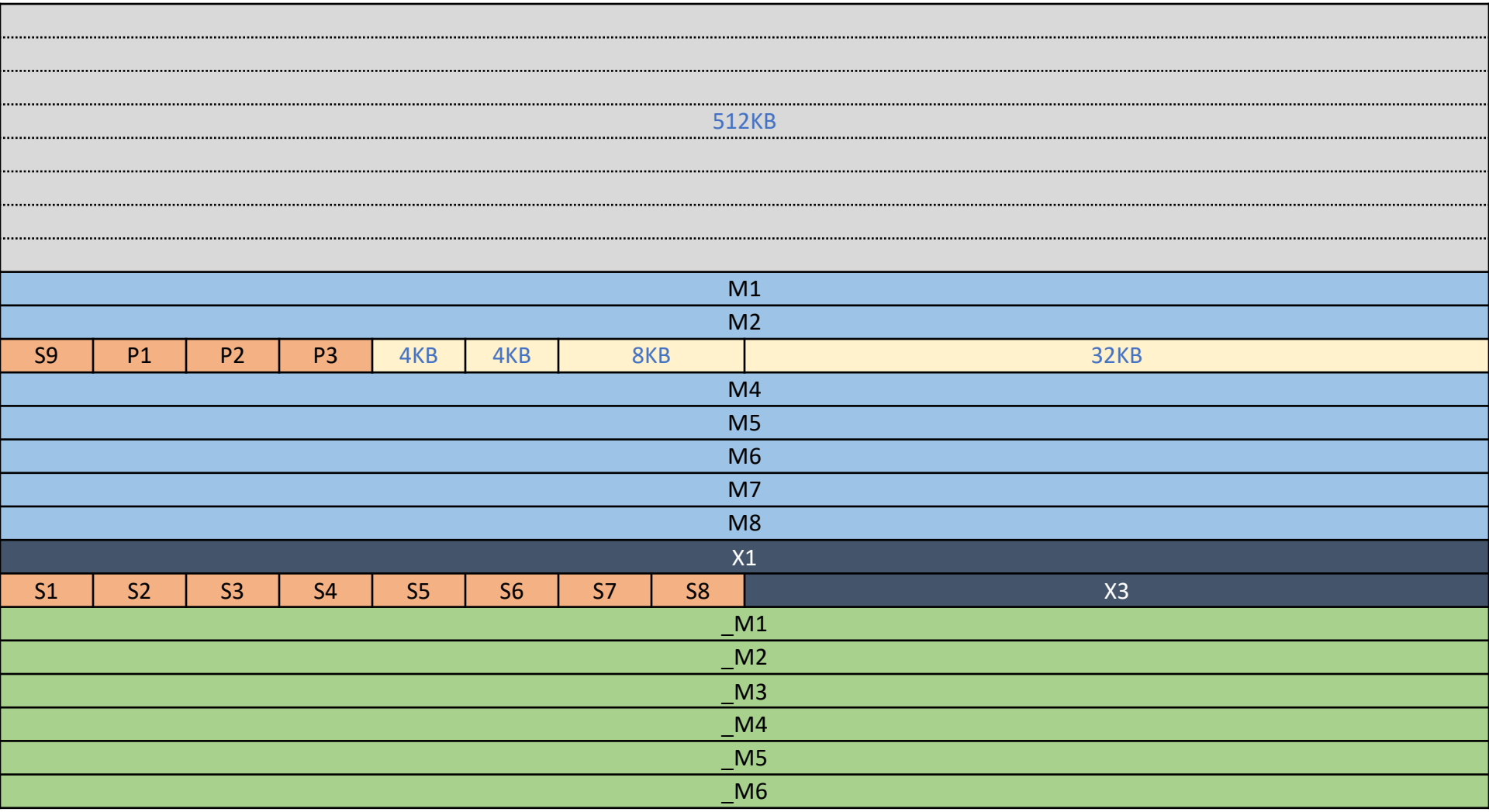
```
PT = mmap(MAP_FIXED); // Force a Page Table allocation
```



Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE

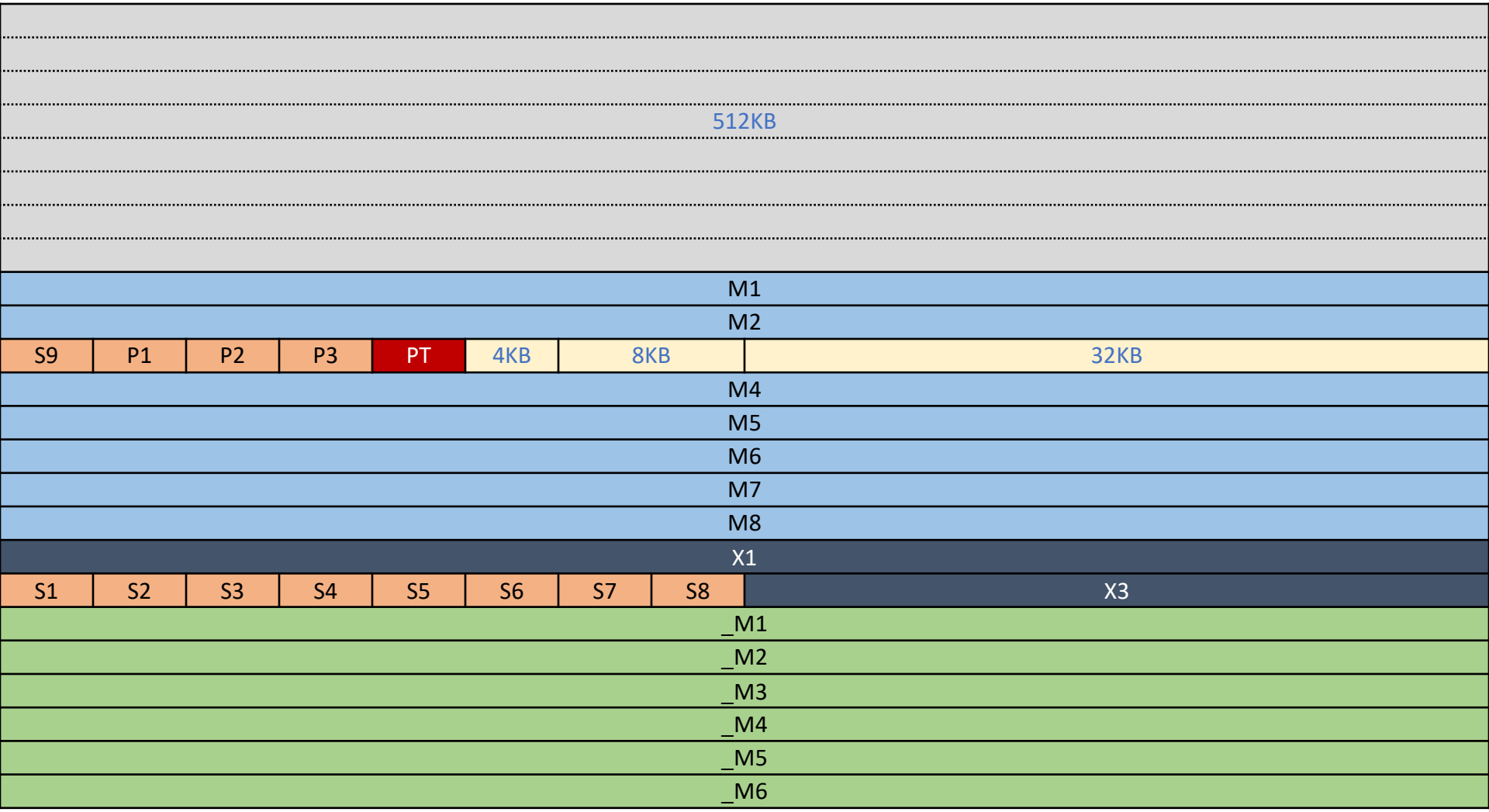
```
PT = mmap(MAP_FIXED); // Force a Page Table allocation
```



Phys Feng Shui step 8/8

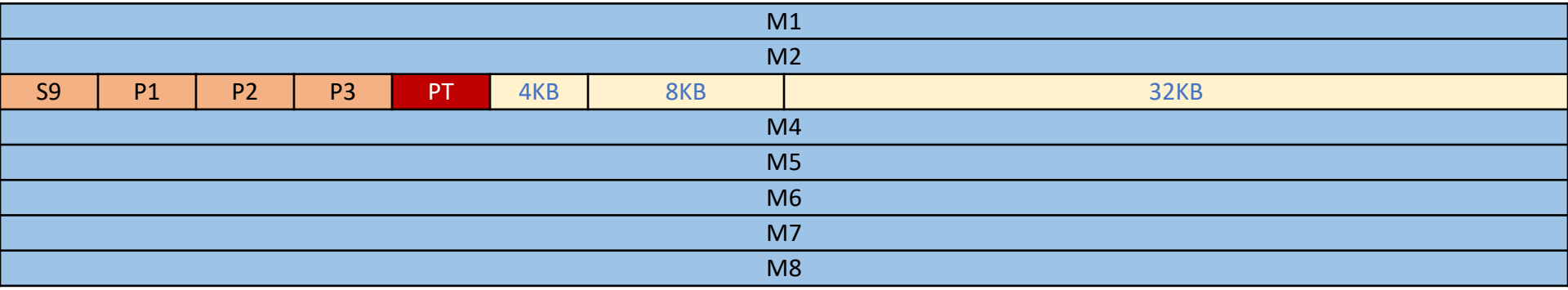
Force a Page Table allocation + map the vulnerable PTE

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



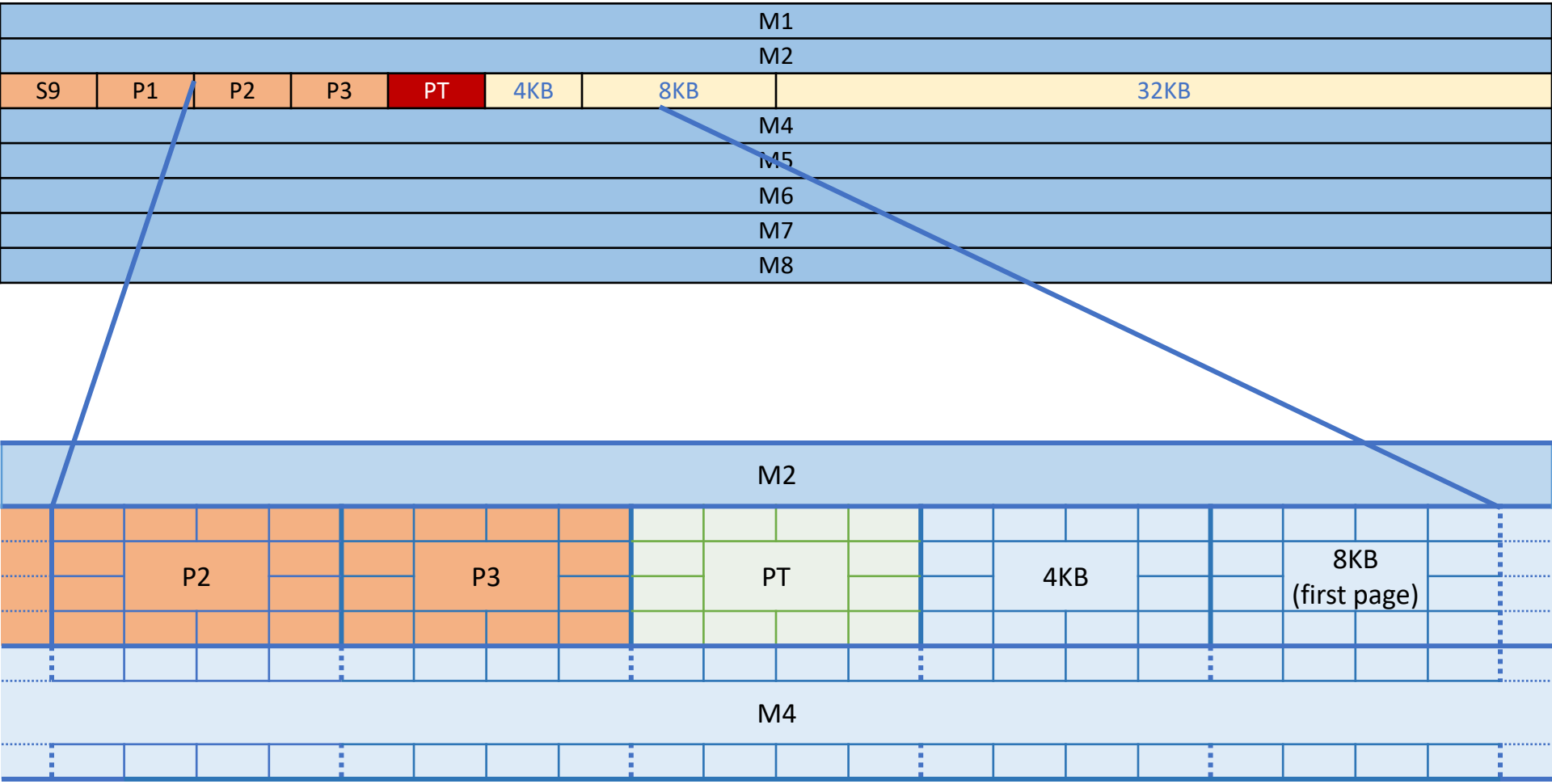
Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE



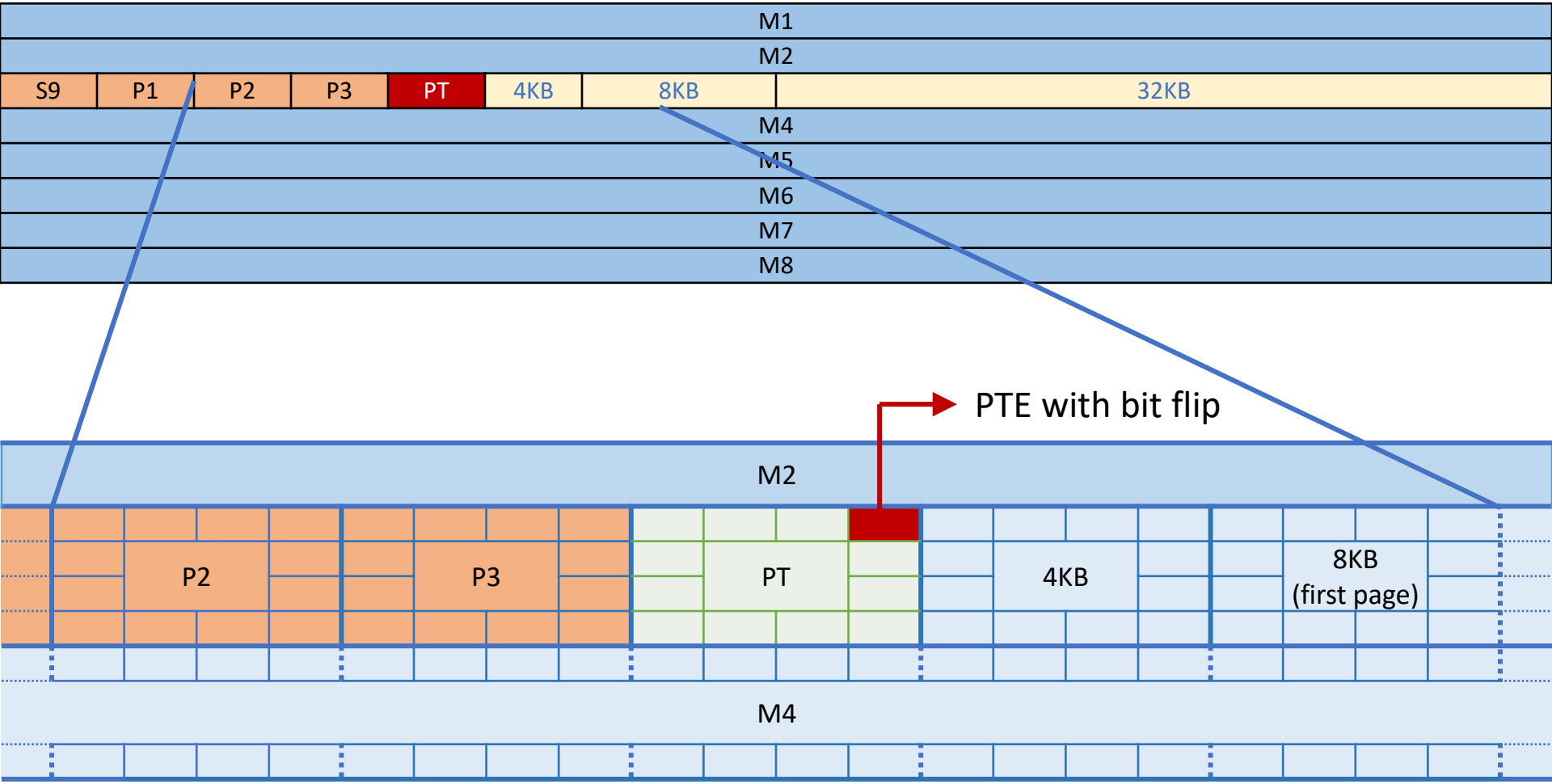
Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE



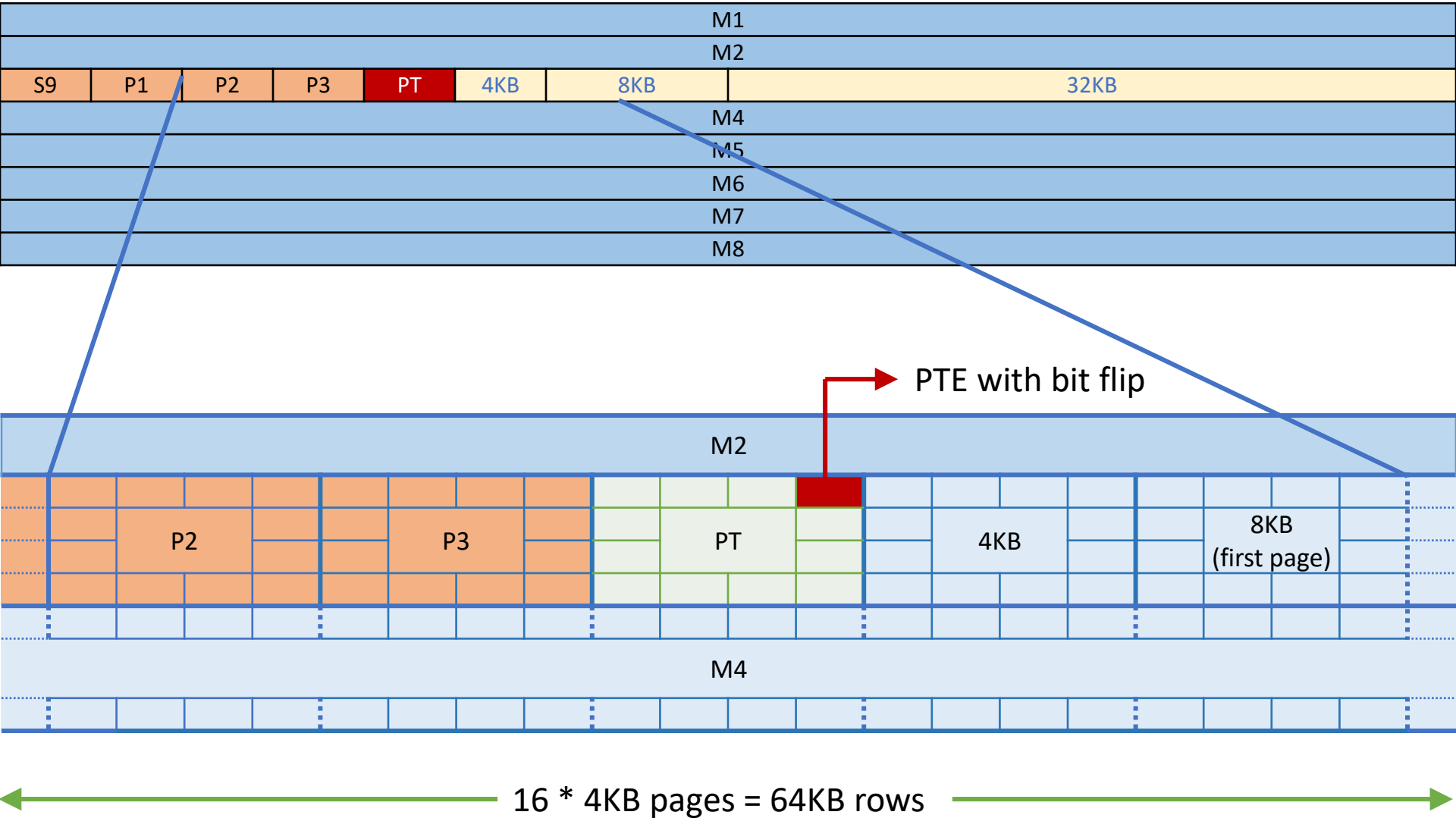
Phys Feng Shui step 8/8

Force a Page Table allocation + map the vulnerable PTE



Phys Feng Shui step 8/8

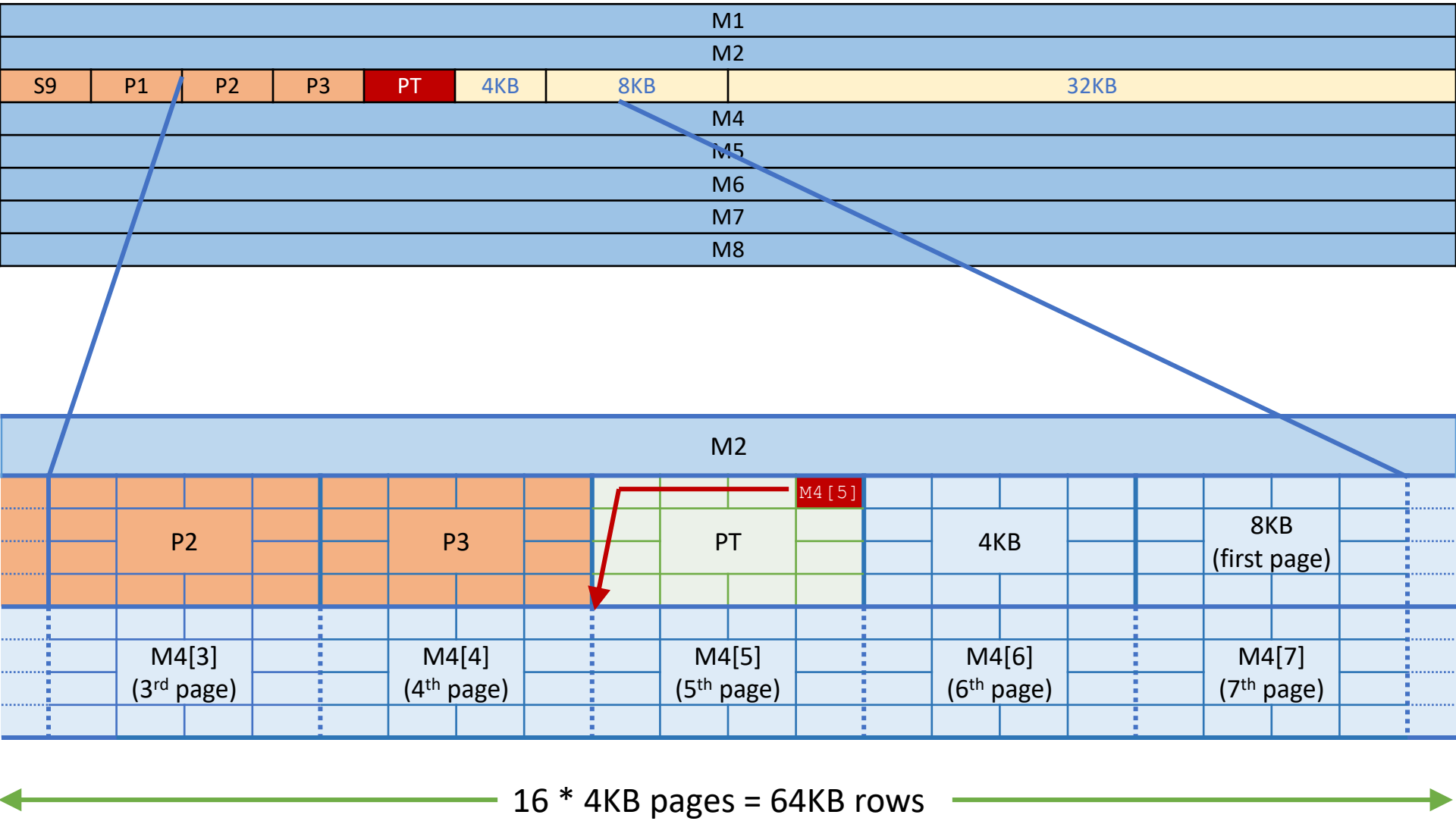
Force a Page Table allocation + map the vulnerable PTE



Phys Feng Shui step 8/8

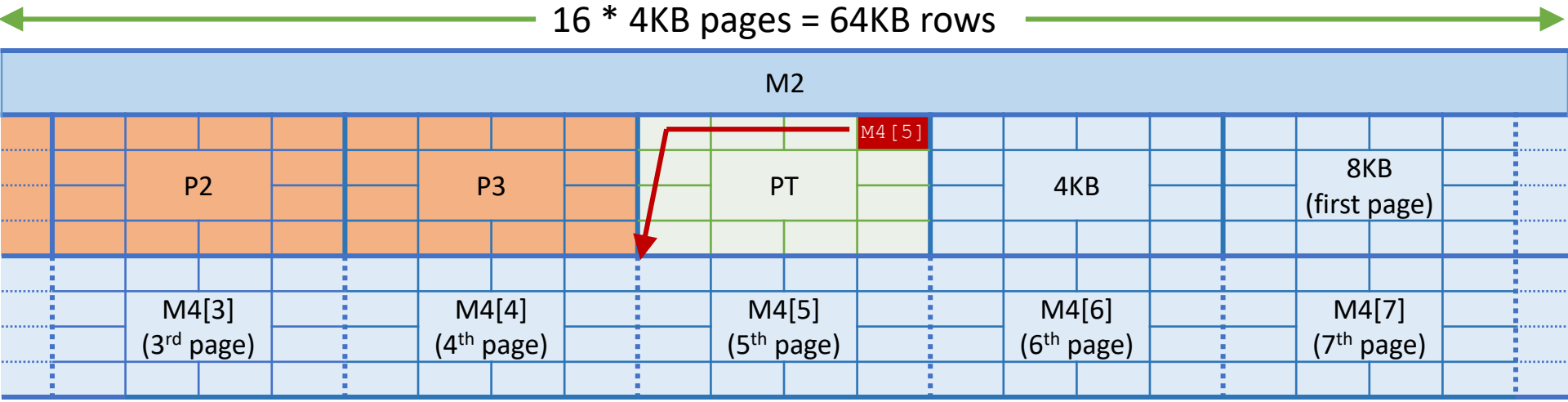
Force a Page Table allocation + map the vulnerable PTE

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



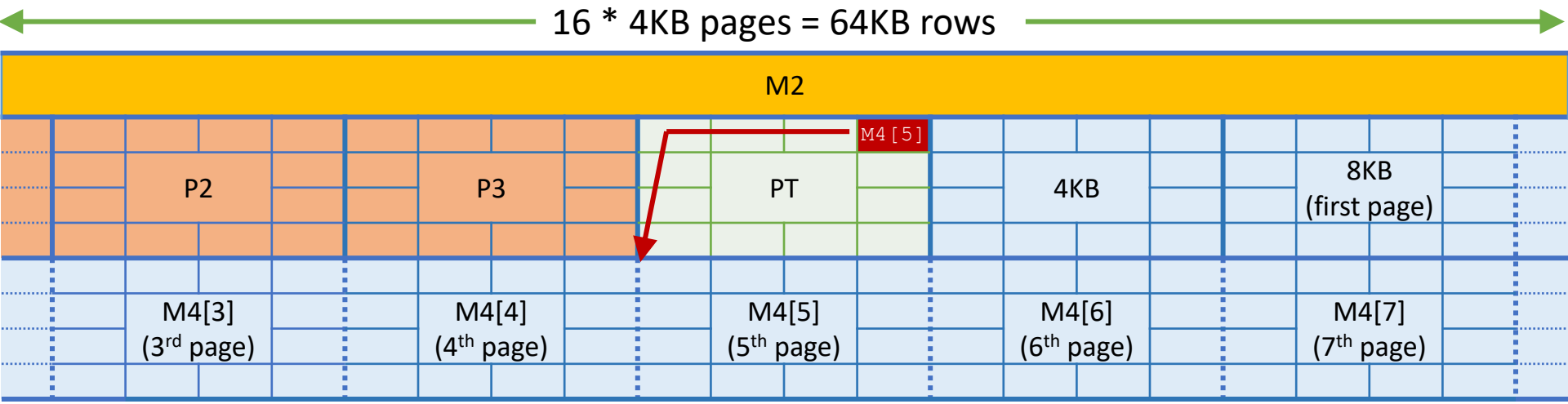
Drammer

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



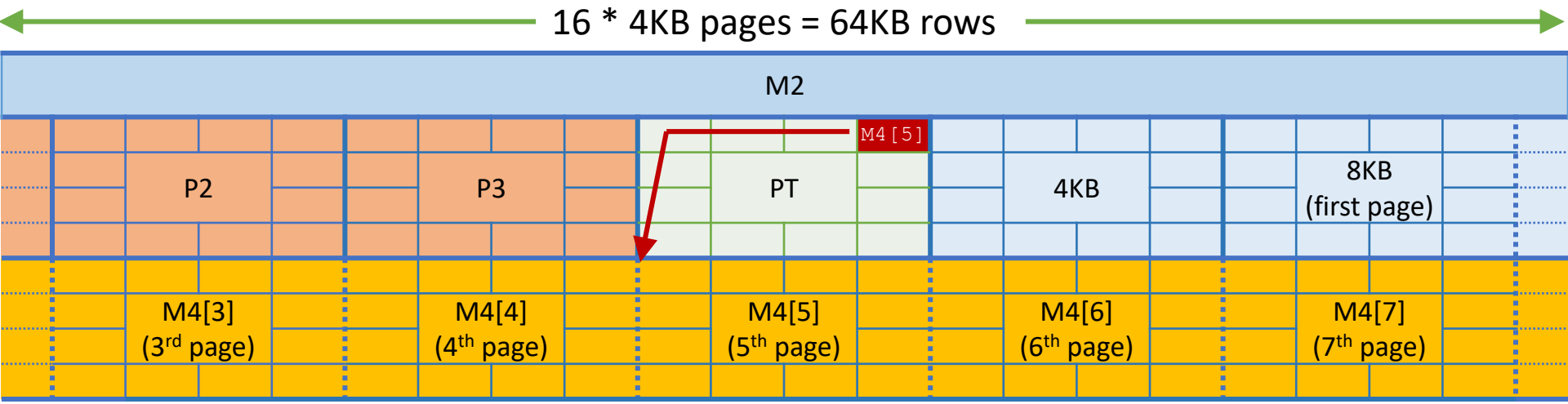
Phys Feng Shui step 8/8

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

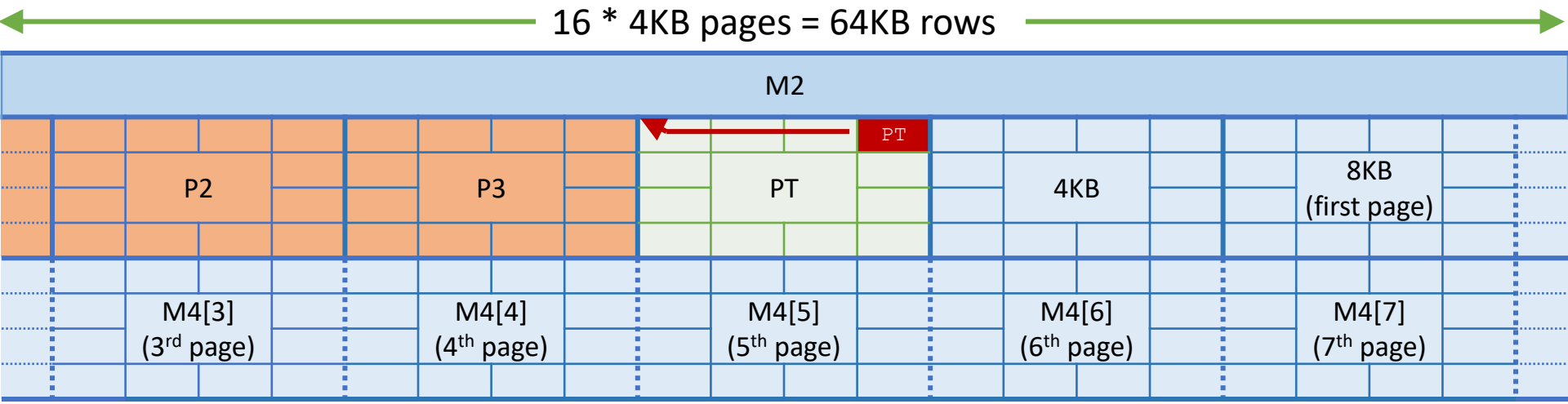


Phys Feng Shui step 8/8

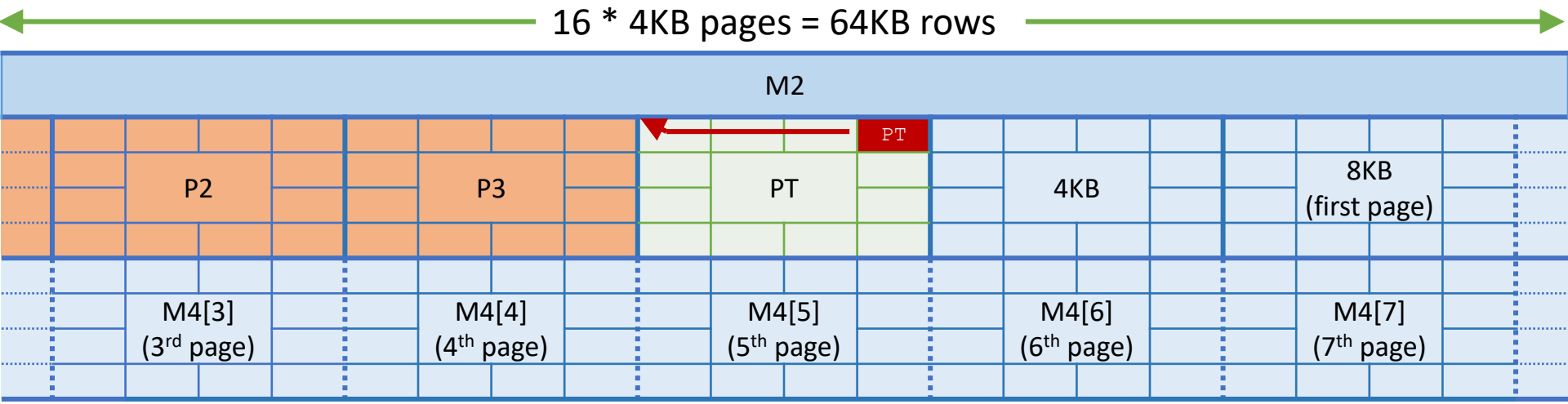
Perform double-sided rowhammer to flip a bit in the PTE



Phys Feng Shui step 8/8

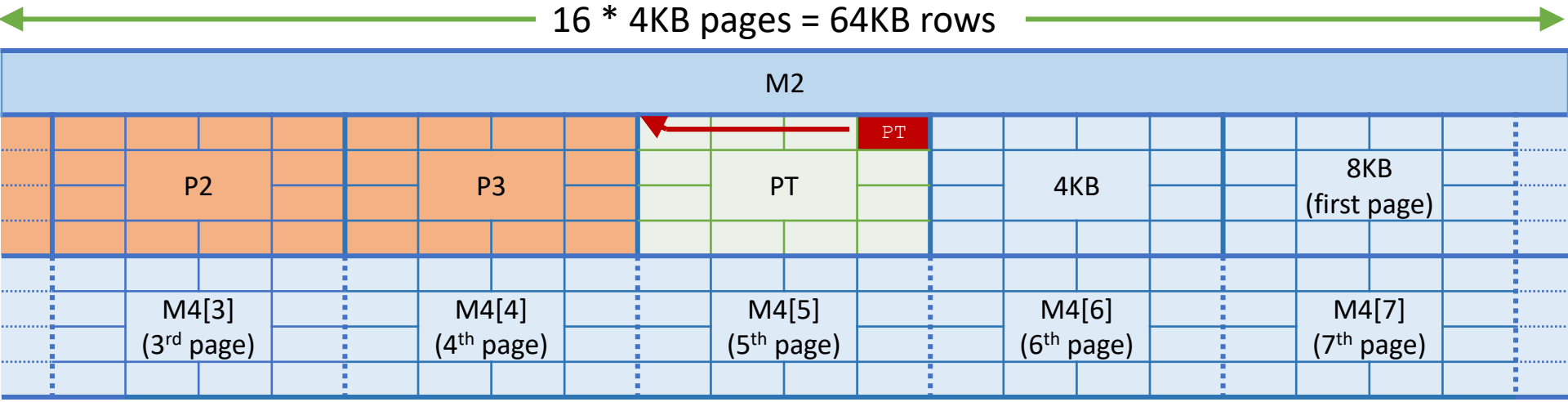


Phys Feng Shui step 8/8



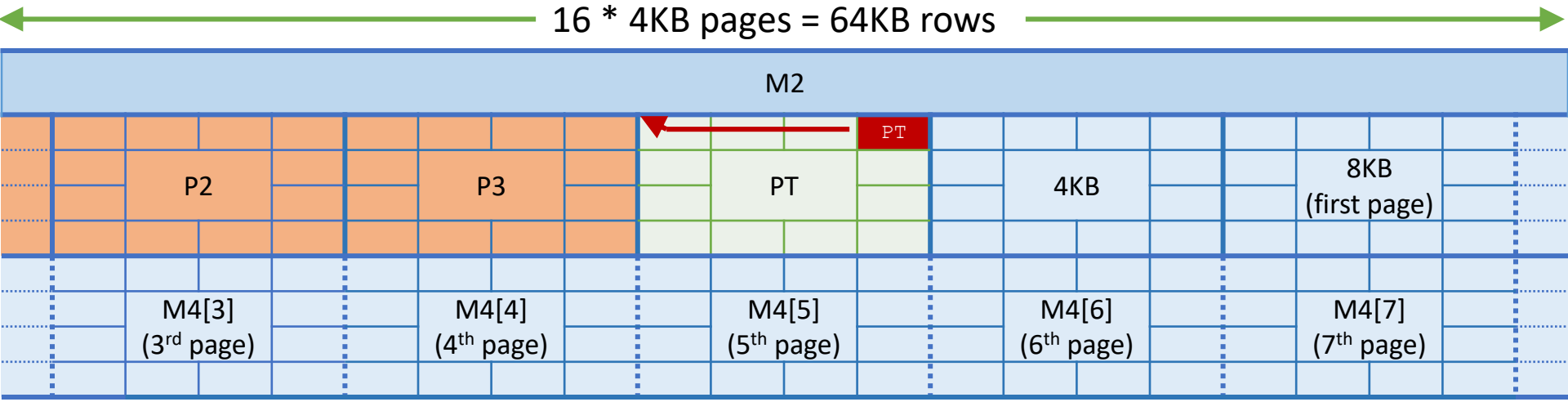
1. Fill PT with Page Table Entries to kernel memory

Phys Feng Shui step 8/8



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

Phys Feng Shui step 8/8



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