ZebRAM: Comprehensive and Compatible Software Protection against Rowhammer Attacks

Radhesh Krishnan Konoth, Marco Oliverio, Andrei Tatar, Dennis Andriesse, Herbert Bos, Cristiano Giuffrida and Kaveh Razavi

VU UNIVERSITY AMSTERDAM

UNIVERSITÀ DELLA CALABRIA

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
    - Even if there is no software bug (and formally verified)

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
  - Even if there is no software bug (and formally verified)
  - 87% of DDR3 DIMMs are vulnerable (Kim et al. ISCA'14)

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
  - Even if there is no software bug (and formally verified)
  - 87% of DDR3 DIMMs are vulnerable (Kim et al. ISCA'14)
  - DDR4 also contain this bug (Van der Veen et al. CCS'17)

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
  - Even if there is no software bug (and formally verified)
  - 87% of DDR3 DIMMs are vulnerable (Kim et al. ISCA'14)
  - DDR4 also contain this bug (Van der Veen et al. CCS'17)

- Existing defenses are ineffective

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
  - Even if there is no software bug (and formally verified)
  - 87% of DDR3 DIMMs are vulnerable (Kim et al. ISCA’14)
  - DDR4 also contain this bug (Van der Veen et al. CCS’17)

- Existing defenses are ineffective
  - Hardware solutions like ECC, TRR are found to ineffective

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
  - Even if there is no software bug (and formally verified)
  - 87% of DDR3 DIMMs are vulnerable (Kim et al. ISCA'14)
  - DDR4 also contain this bug (Van der Veen et al. CCS'17)

- Existing defenses are ineffective
  - Hardware solutions like ECC, TRR are found to ineffective (Cojocar et. al S&P'19, Gruss et al. Blackhat'18)

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
  - Even if there is no software bug (and formally verified)
  - 87% of DDR3 DIMMs are vulnerable (Kim et al. ISCA’14)
  - DDR4 also contain this bug (Van der Veen et al. CCS’17)

- Existing defenses are ineffective
  - Hardware solutions like ECC, TRR are found to ineffective (Cojocar et. al S&P’19, Gruss et al. Blackhat’18)
  - ANVIL - CPU performance counters to detect Rowhammer attack (AWEKE et. al ASPLOS’16)

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
    - Even if there is no software bug (and formally verified)
    - 87% of DDR3 DIMMs are vulnerable (Kim et al. ISCA’14)
    - DDR4 also contain this bug (Van der Veen et al. CCS’17)

- Existing defenses are ineffective
    - Hardware solutions like ECC, TRR are found to ineffective (Cojocar et. al S&P’19, Gruss et al. Blackhat’18)
    - ANVIL - fails against DMA-based attacks   (Van der Veen et al. CCS’17)

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
  - Even if there is no software bug (and formally verified)
  - 87% of DDR3 DIMMs are vulnerable (Kim et al. ISCA'14)
  - DDR4 also contain this bug (Van der Veen et al. CCS'17)

- Existing defenses are ineffective
  - Hardware solutions like ECC, TRR are found to ineffective (Cojocar et. al S&P'19, Gruss et al. Blackhat'18)
  - ANVIL - fails against DMA-based attacks (Van der Veen et al. CCS'17)
  - CATT - isolates different security domains using guard rows (Brasser et al. SEC'17)

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
  - Even if there is no software bug (and formally verified)
  - 87% of DDR3 DIMMs are vulnerable (Kim et al. ISCA'14)
  - DDR4 also contain this bug (Van der Veen et al. CCS'17)

- Existing defenses are ineffective
  - Hardware solutions like ECC, TRR are found to ineffective (Cojocar et. al S&P'19, Gruss et al. Blackhat'18)
  - ANVIL - fails against DMA-based attacks (Van der Veen et al. CCS'17)
  - CATT - fails because different security domains share memory (Gruss et al. S&P'18)

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
    - Even if there is no software bug (and formally verified)
    - 87% of DDR3 DIMMs are vulnerable (Kim et al. ISCA'14)
    - DDR4 also contain this bug (Van der Veen et al. CCS'17)

- Existing defenses are ineffective
    - Hardware solutions like ECC, TRR are found to ineffective (Cojocar et. al S&P'19, Gruss et al. Blackhat'18)
    - ANVIL - fails against DMA-based attacks (Van der Veen et al. CCS'17)
    - CATT - fails because different security domains share memory (Gruss et al. S&P'18)

- ZebRAM

Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
  - Even if there is no software bug (and formally verified)
  - 87% of DDR3 DIMMs are vulnerable (Kim et al. ISCA'14)
  - DDR4 also contain this bug (Van der Veen et al. CCS'17)

- Existing defenses are ineffective
  - Hardware solutions like ECC, TRR are found to ineffective (Cojocar et. al S&P'19, Gruss et al. Blackhat'18)
  - ANVIL - fails against DMA-based attacks (Van der Veen et al. CCS'17)
  - CATT - fails because different security domains share memory (Gruss et al. S&P'18)

- ZebRAM
  - The first **comprehensive** and **compatible software-based** solution ...

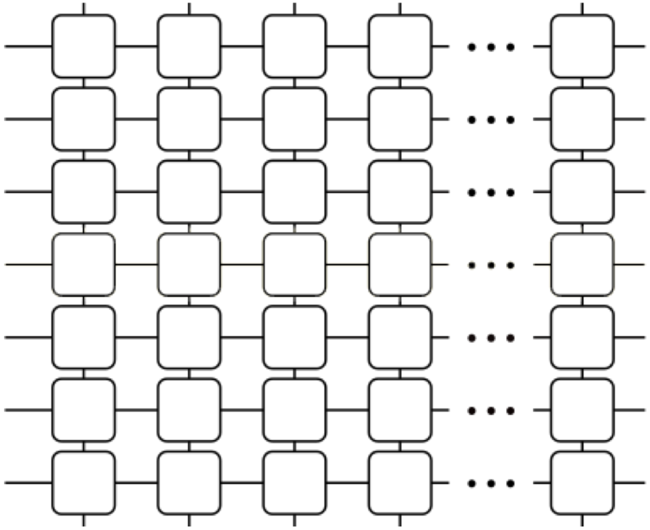
Motivation

- Rowhammer -- a DRAM defect that allows an attacker to exploit a system
  - Even if there is no software bug (and formally verified)
  - 87% of DDR3 DIMMs are vulnerable (Kim et al. ISCA'14)
  - DDR4 also contain this bug (Van der Veen et al. CCS'17)

- Existing defenses are ineffective
  - Hardware solutions like ECC, TRR are found to ineffective (Cojocar et. al S&P'19, Gruss et al. Blackhat'18)
  - ANVIL - fails against DMA-based attacks (Van der Veen et al. CCS'17)
  - CATT - fails because different security domains share memory (Gruss et al. S&P'18)

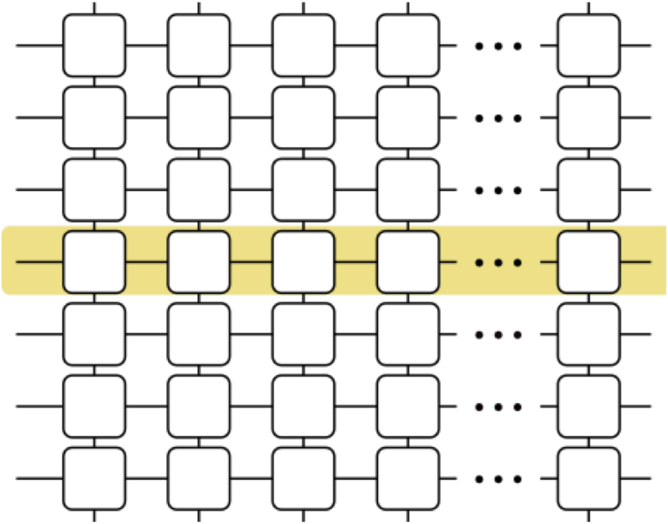
- ZebRAM
  - The first **comprehensive** and **compatible software-based** solution ...
  - … to defend against this hardware bug.

Rowhammer bug



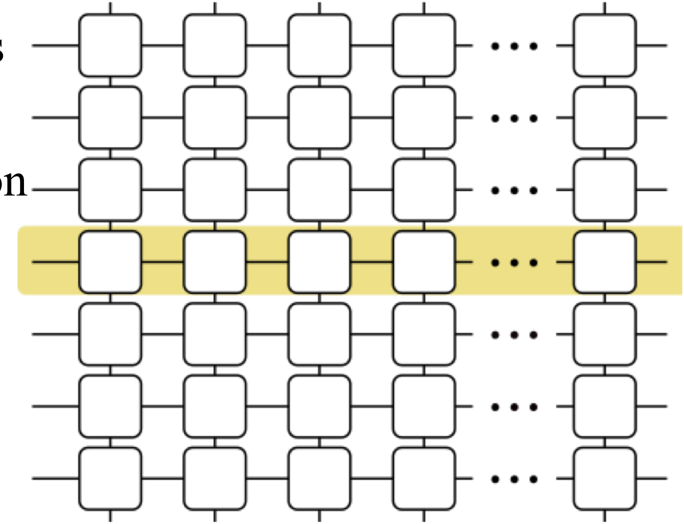
Rowhammer bug

- DRAM rows consists of DRAM cells



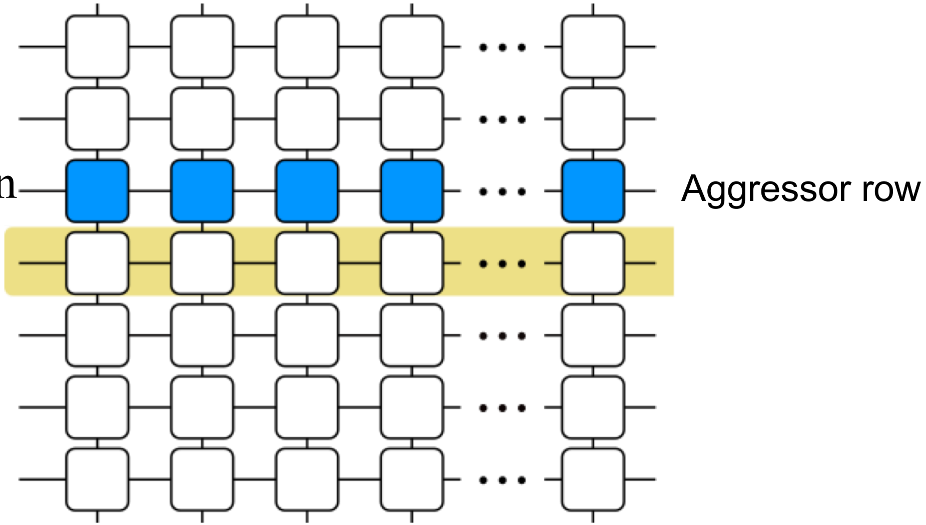
Rowhammer bug

- DRAM rows consists of DRAM cells
- Each cell can store one bit information



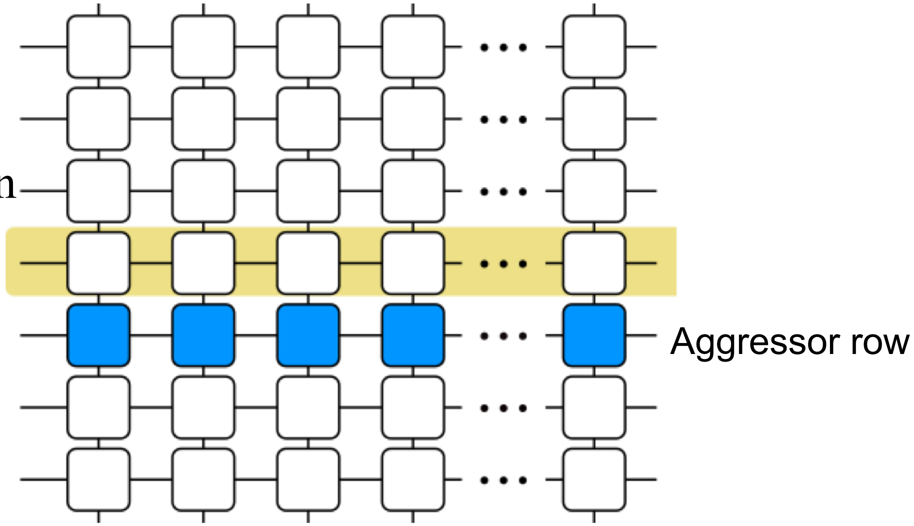
Rowhammer bug

- DRAM rows consists of DRAM cells
- Each cell can store one bit information
- Up on proximate access, DRAM cells leak charge to **neighbouring cells …**



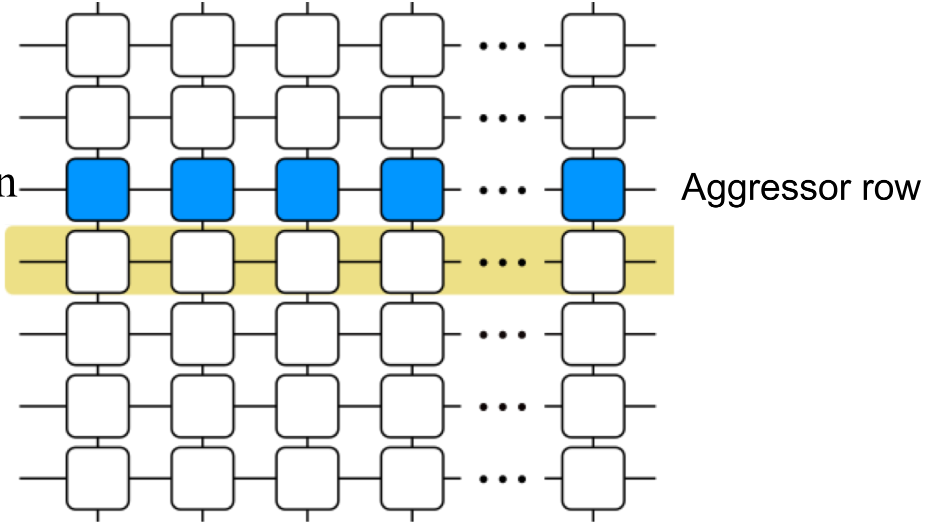
Rowhammer bug

- DRAM rows consists of DRAM cells
- Each cell can store one bit information
- Up on proximate access, DRAM cells leak charge to **neighbouring cells …**



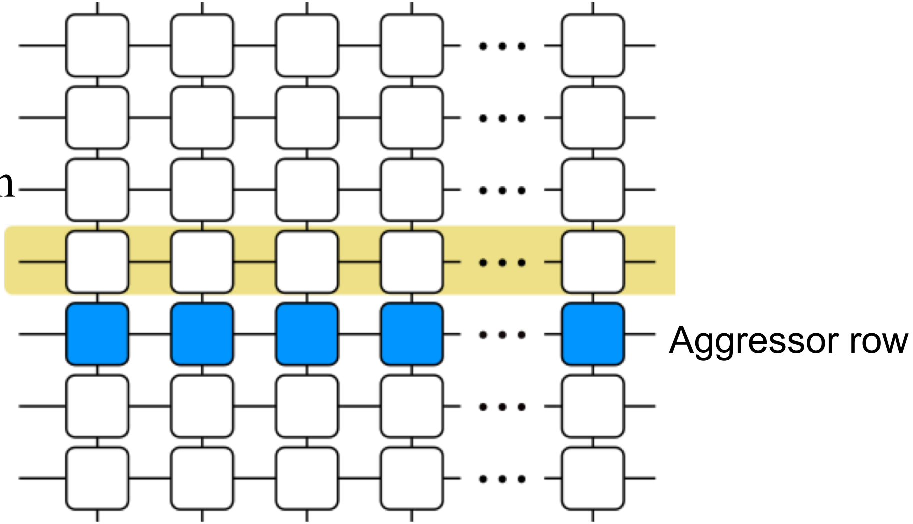
Rowhammer bug

- DRAM rows consists of DRAM cells
- Each cell can store one bit information
- Up on proximate access, DRAM cells leak charge to **neighbouring cells …**



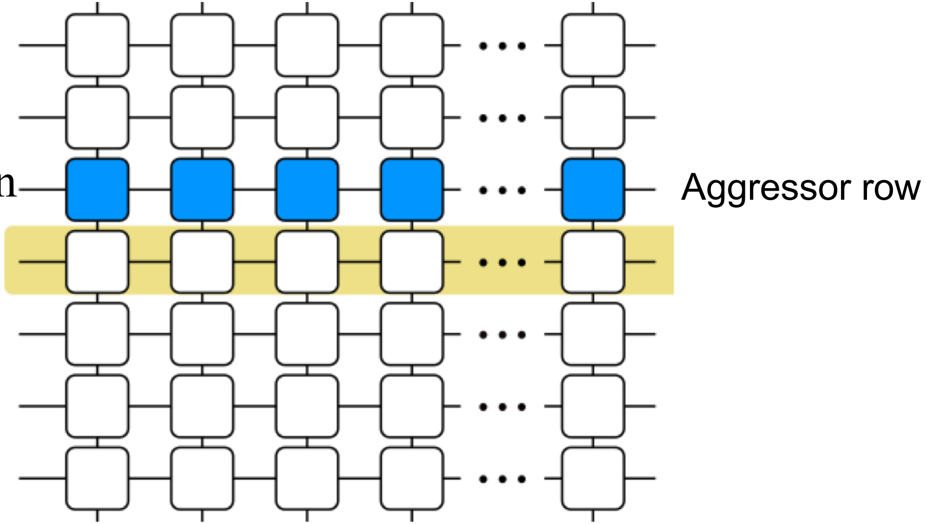
Rowhammer bug

- DRAM rows consists of DRAM cells
- Each cell can store one bit information
- Up on proximate access, DRAM cells leak charge to **neighbouring cells …**



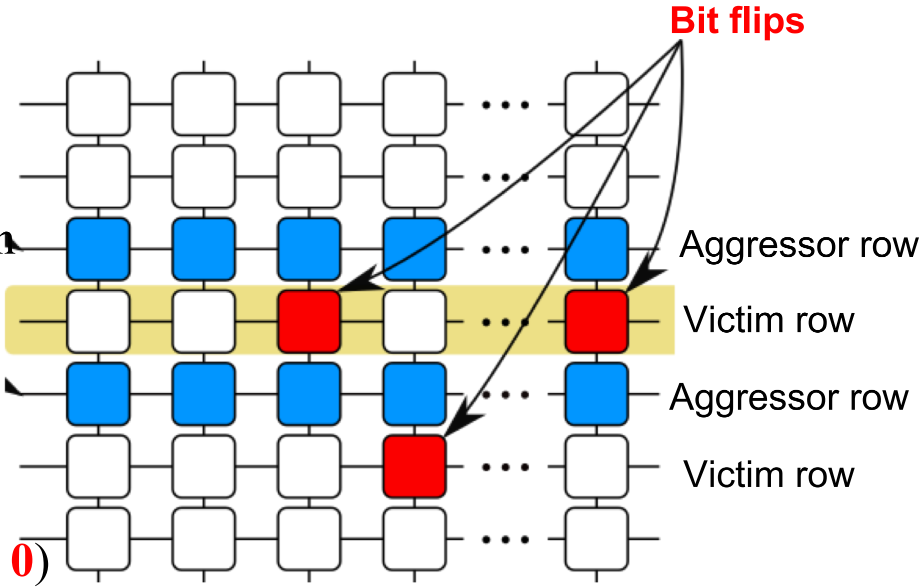
Rowhammer bug

- DRAM rows consists of DRAM cells
- Each cell can store one bit information
- Up on proximate access, DRAM cells leak charge to **neighbouring cells …**



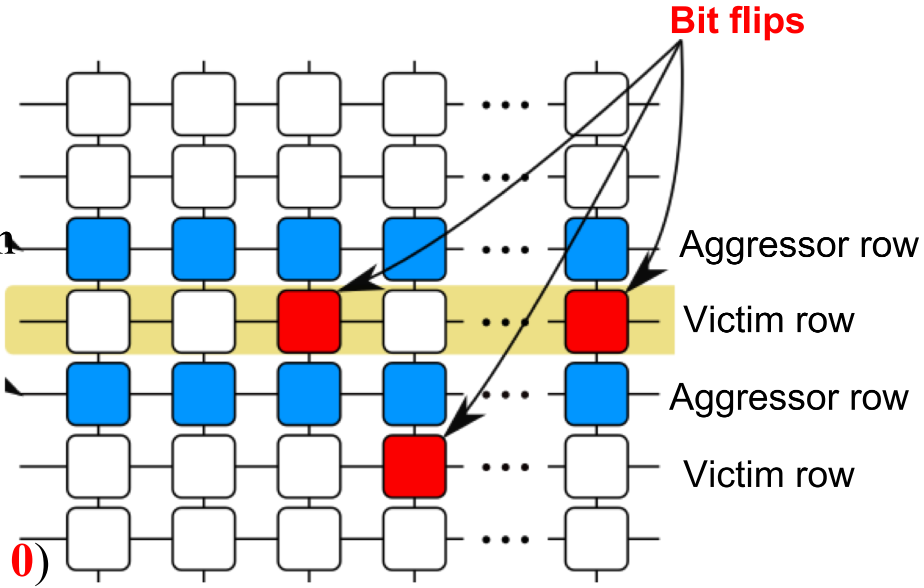
Rowhammer bug

- DRAM rows consists of DRAM cells
- Each cell can store one bit information
- Up on proximate access, DRAM cells leak charge to **neighbouring cells …**
- … and induce **bit flips** in them: (**1 => 0**) or (**0 => 1**)



Rowhammer bug

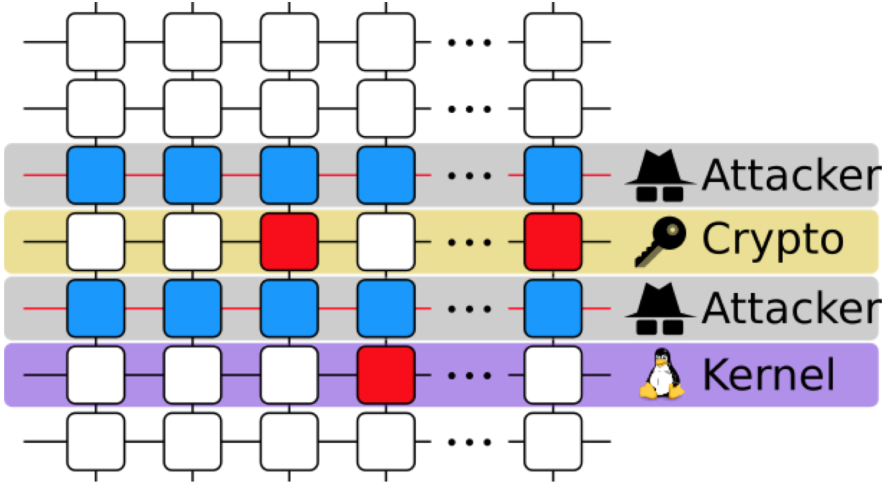
- DRAM rows consists of DRAM cells
- Each cell can store one bit information
- Up on proximate access, DRAM cells leak charge to **neighbouring cells …**
- … and induce **bit flips** in them: (**1 => 0**) or (**0 => 1**)
- **Rowhammer bug**



How is this a security problem?

An attacker can flips a bit in:

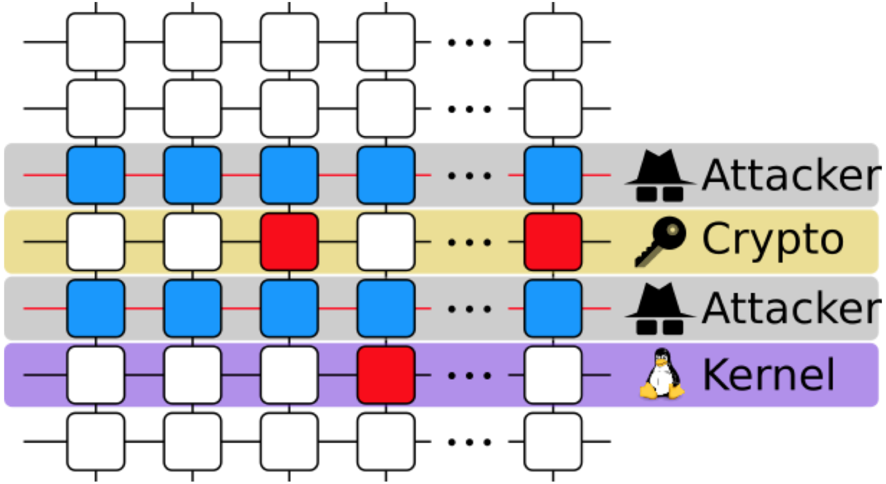
- Cryptographic key, page table entry in kernel e.t.c.
- … to compromise the system.



How is this a security problem?

An attacker can flips a bit in:

- Cryptographic key, page table entry in kernel e.t.c.
- … to compromise the system.



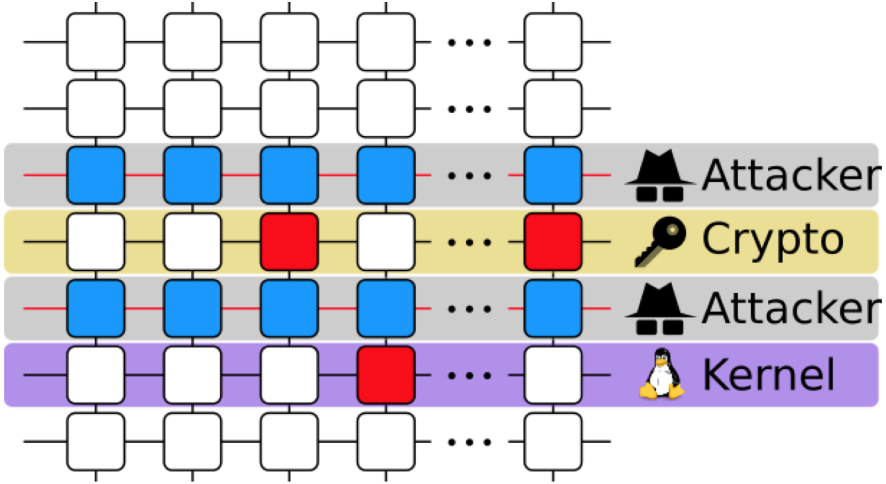
Two important points to note:

1. Attacker should able to read **very fast**

How is this a security problem?

An attacker can flips a bit in:

- Cryptographic key, page table entry in kernel e.t.c.
- … to compromise the system.



Two important points to note:

1. Attacker should able to read **very fast**
2. Can flip a bit on its **neighboring** row

Solution for many security problems

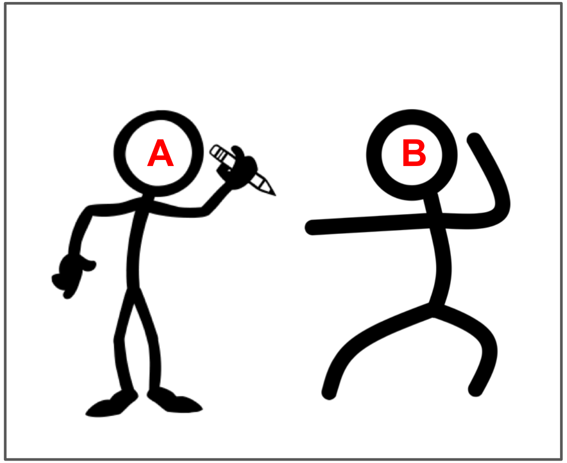
Solution for many security problems

Isolation

Solution for many security problems

Isolation

To protect a process A from writing to process B's memory:

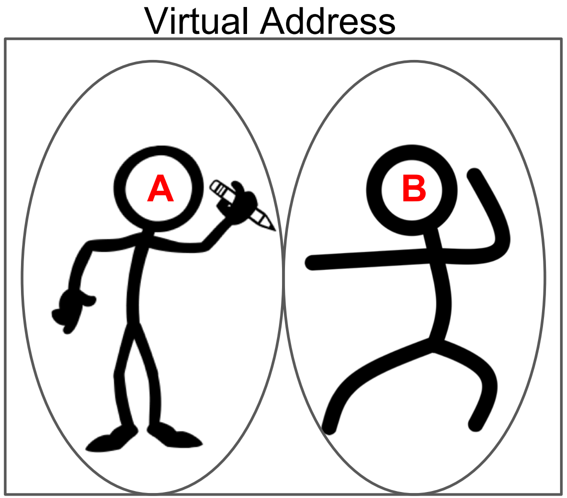


Solution for many security problems

Isolation

To protect a process *A* from writing to process *B*'s memory:

- ➢ We isolate them using virtual address space

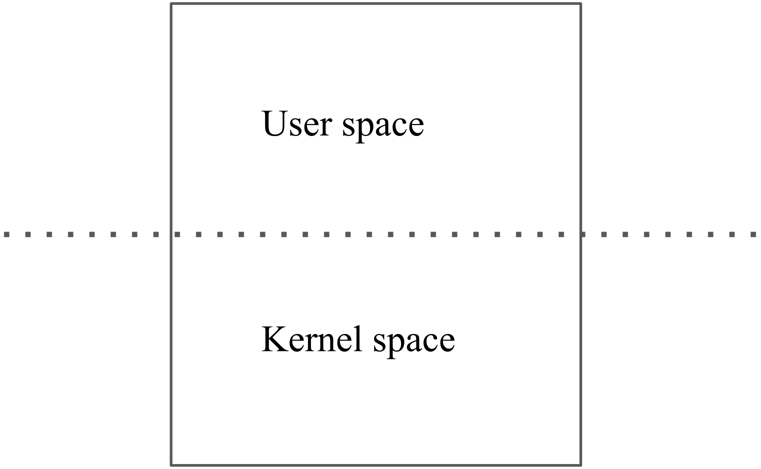


Isolation approach 1

1. **Separate security domains using guard rows**

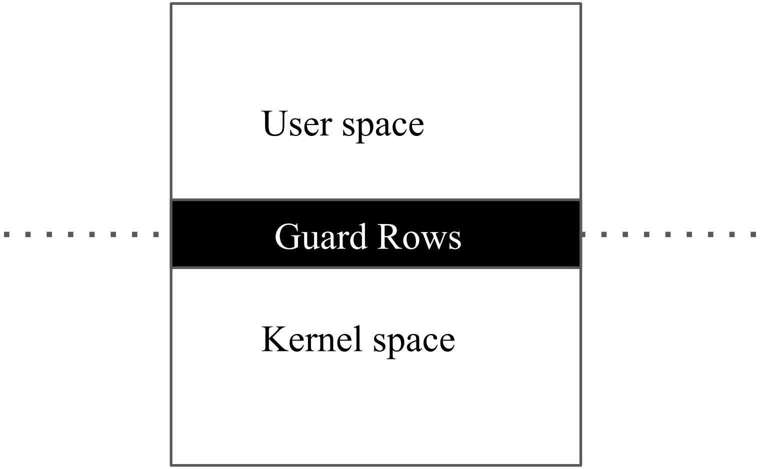
Isolation approach 1

1. **Separate security domains using guard rows**



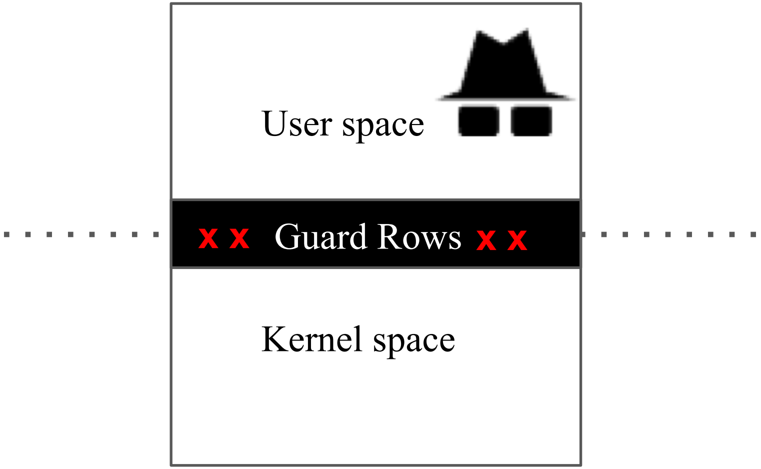
Isolation approach 1

1. **Separate security domains using guard rows**



Isolation approach 1

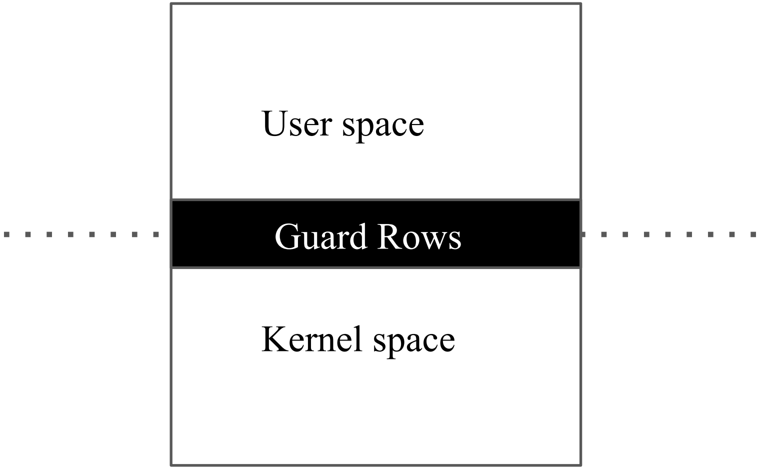
1. **Separate security domains using guard rows**



Isolation approach 1

1. **Separate security domains using guard rows**

CATT uses this approach (Brasser et al. SEC’17)



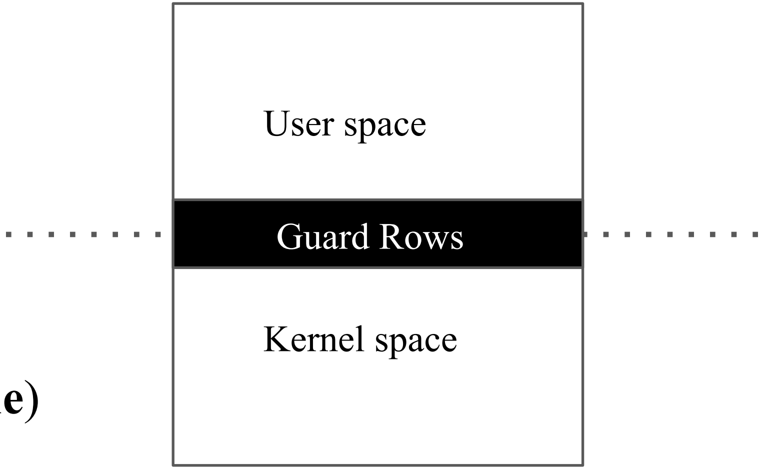
Isolation approach 1

1. **Separate security domains using guard rows**

CATT uses this approach (Brasser et al. SEC'17)

Limitation :

➢ Security domains share memory (**pagecache**) (Gruss et al. S&P'18)



Isolation approach 2

1. Separate security domains using guard rows
2. **Isolate security sensitive data using guard rows**

An application can use a custom memory allocator:

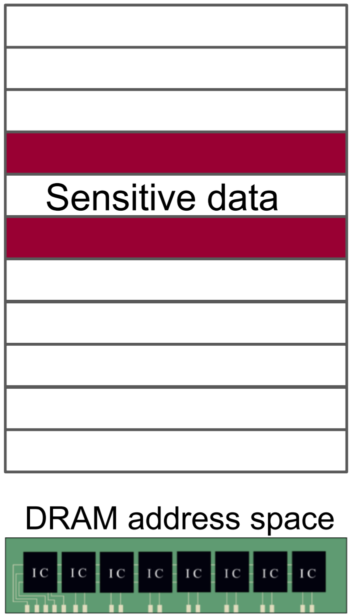
- ➢ Allocate memory protected by guard rows

Isolation approach 2

1. Separate security domains using guard rows
2. **Isolate security sensitive data using guard rows**

An application can use a custom memory allocator:

- ➢ Allocate memory protected by guard rows
- ➢ for storing sensitive data (Tatar et al. ATC'18)



Isolation approach 2

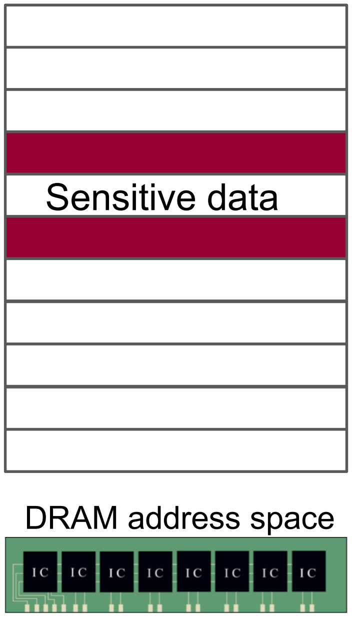
1. Separate security domains using guard rows
2. **Isolate security sensitive data using guard rows**

An application can use a custom memory allocator:

- ➢ Allocate memory protected by guard rows
- ➢ for storing sensitive data (Tatar et al. ATC'18)

Limitation:

- ➢ Application specific defense



ZebRAM

Protect the **whole** system **transparently**..

ZebRAM

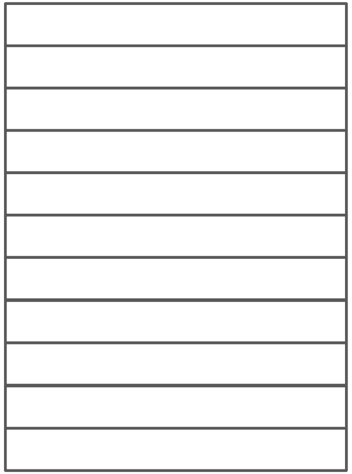
Protect the **whole** system **transparently**..

...by placing **guard row** between every **data row**!

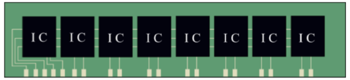
ZebRAM

Protect the **whole** system **transparently**..

...by placing **guard row** between every **data row**!



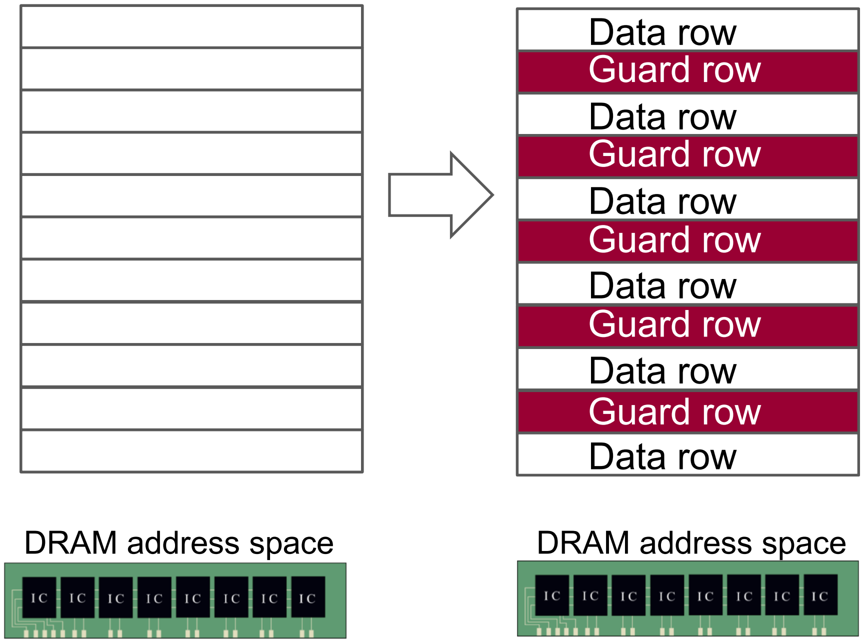
DRAM address space



ZebRAM

Protect the **whole** system **transparently**..

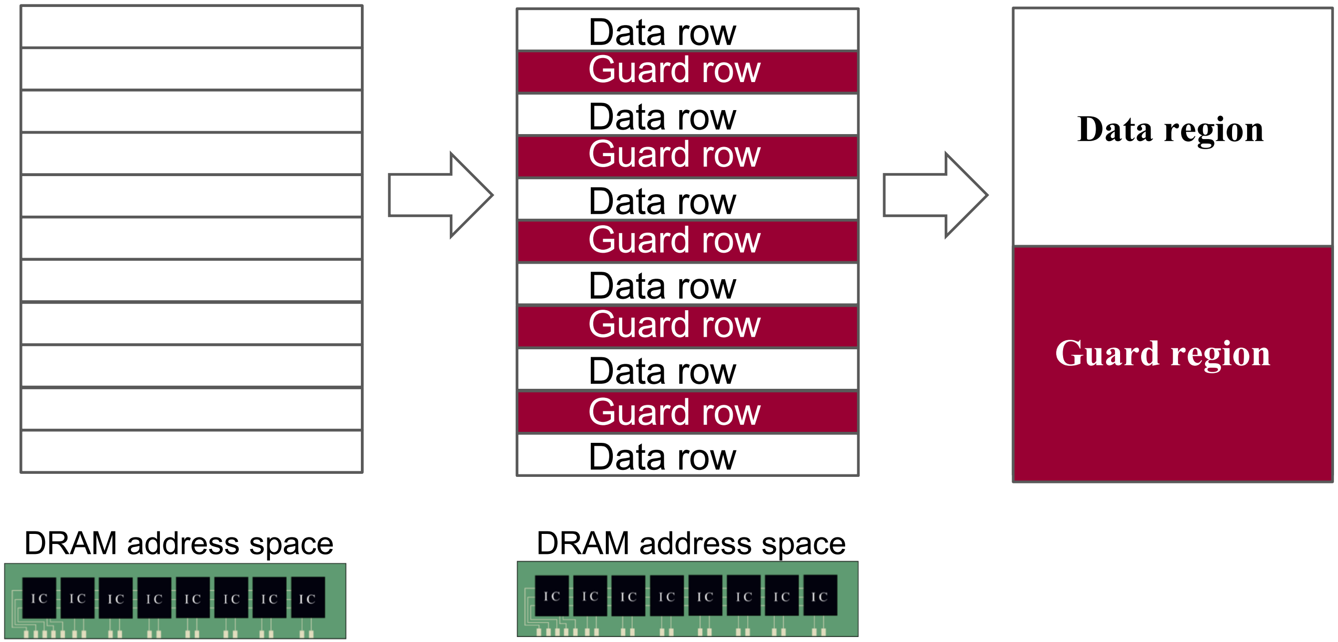
...by placing **guard row** between every **data row**!



ZebRAM

Protect the **whole** system **transparently**..

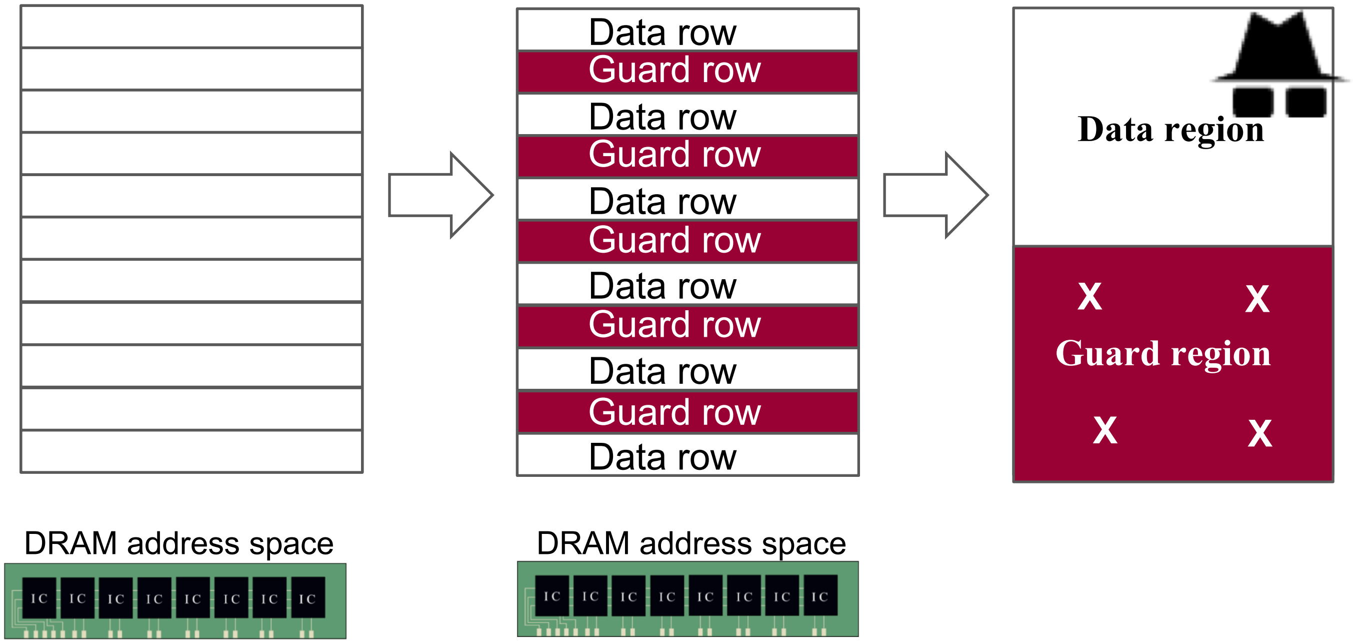
...by placing **guard row** between every **data row**!



ZebRAM

Protect the **whole** system **transparently**..

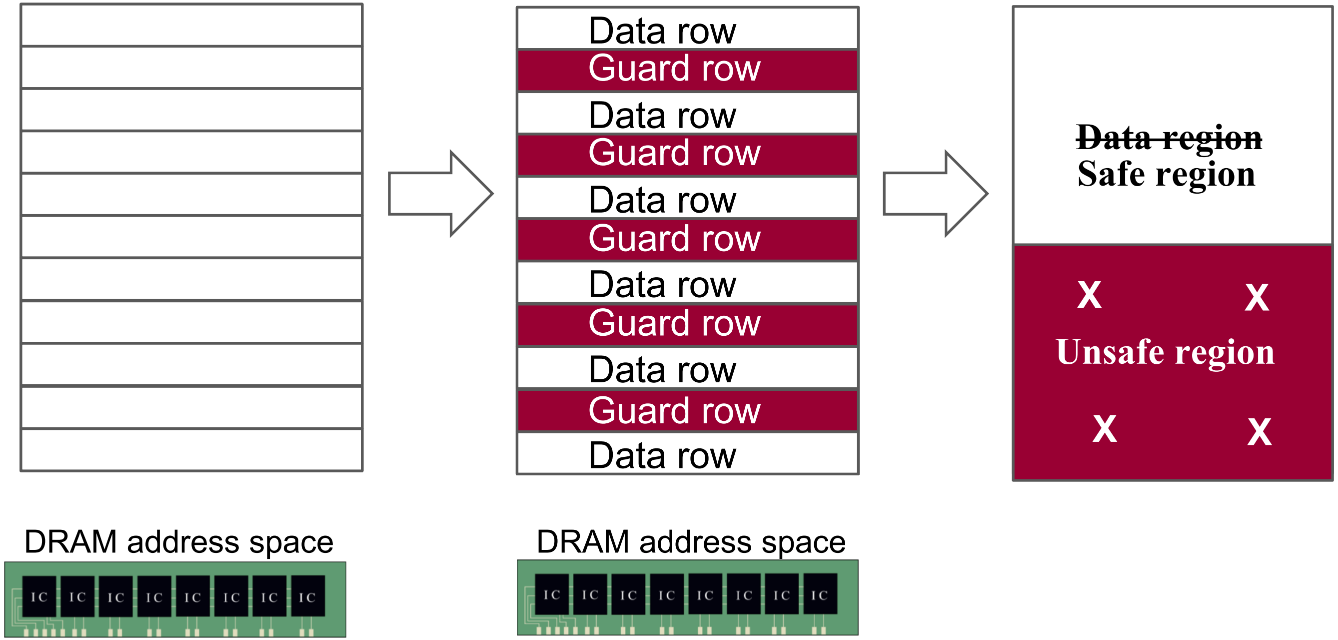
...by placing **guard row** between every **data row**!



ZebRAM

Protect the **whole** system **transparently**..

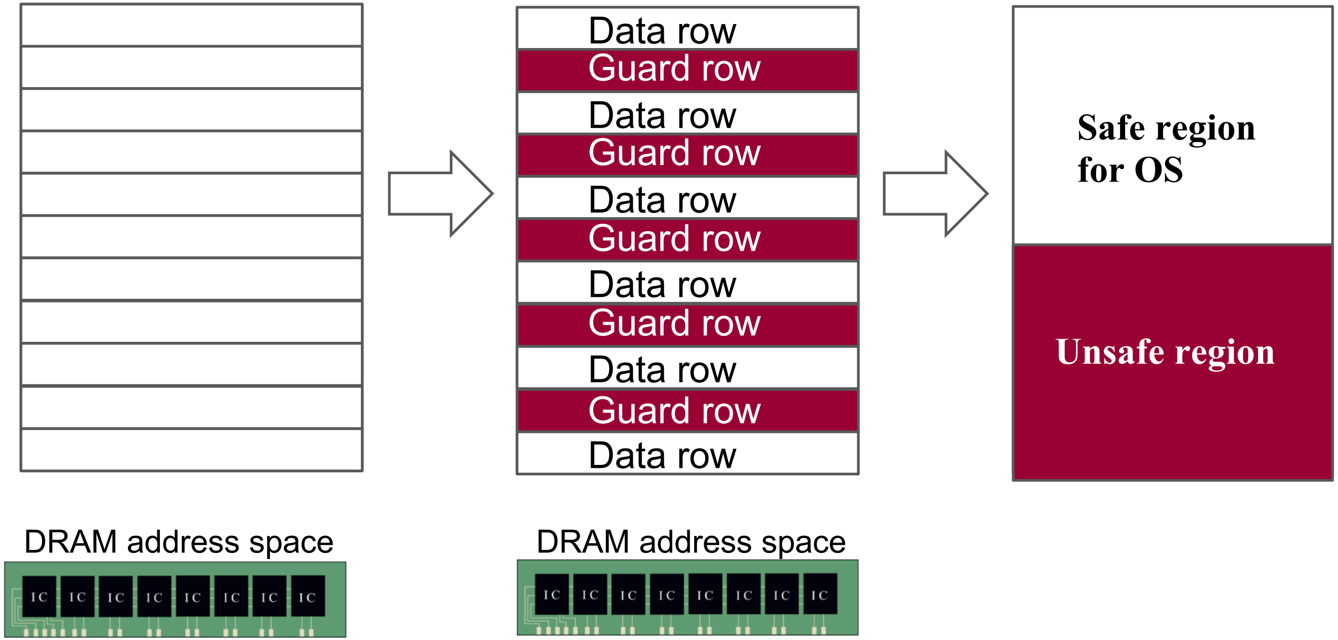
...by placing **guard row** between every **data row**!



ZebRAM

Protect the **whole** system **transparently**..

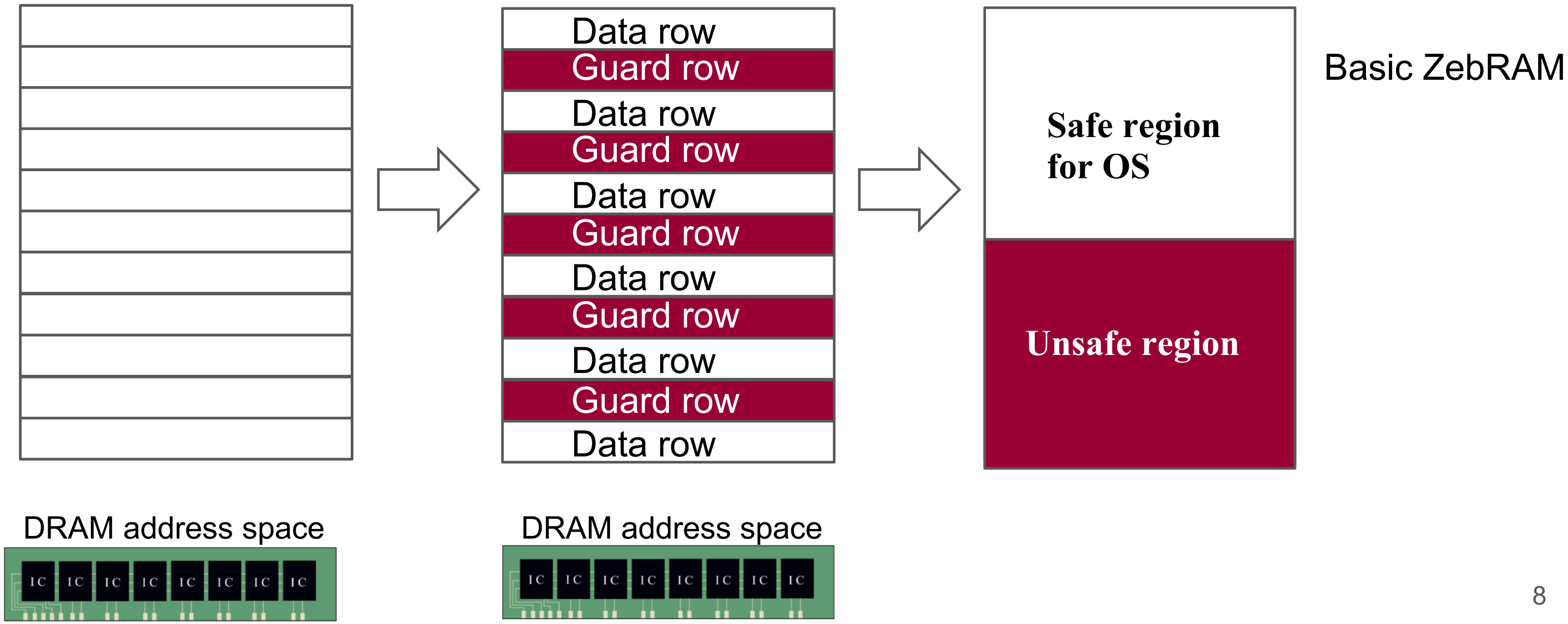
...by placing **guard row** between every **data row**!



ZebRAM

Protect the **whole** system **transparently**..

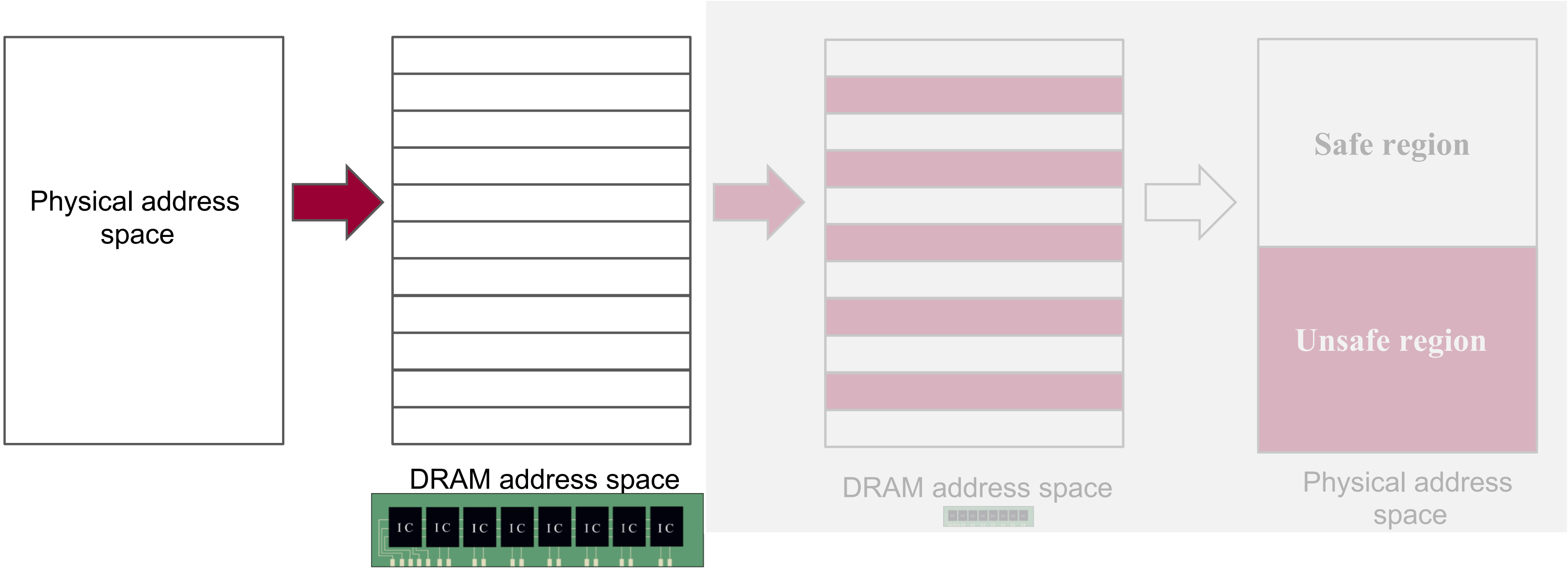
...by placing **guard row** between every **data row**!



How do we achieve these?

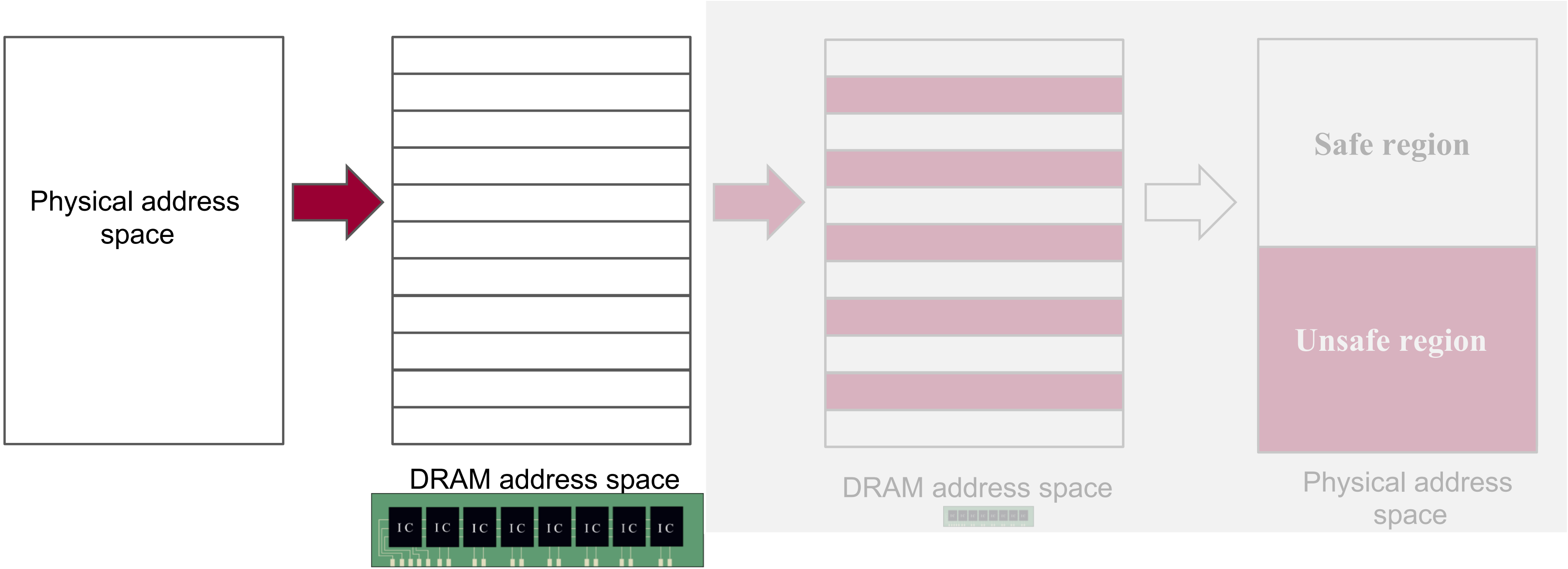
ZebRAM Challenge 1

1. We want to isolate every row in DRAM using guard rows



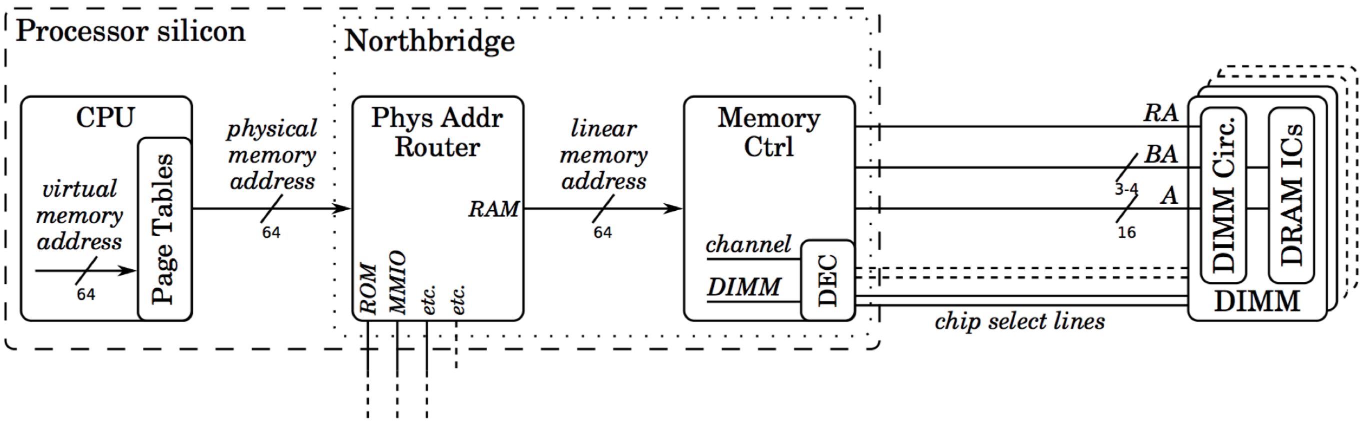
ZebRAM Challenge 1

1. We want to isolate every row in DRAM using guard rows
   - Map physical address to its location in DRAM (DRAM address)



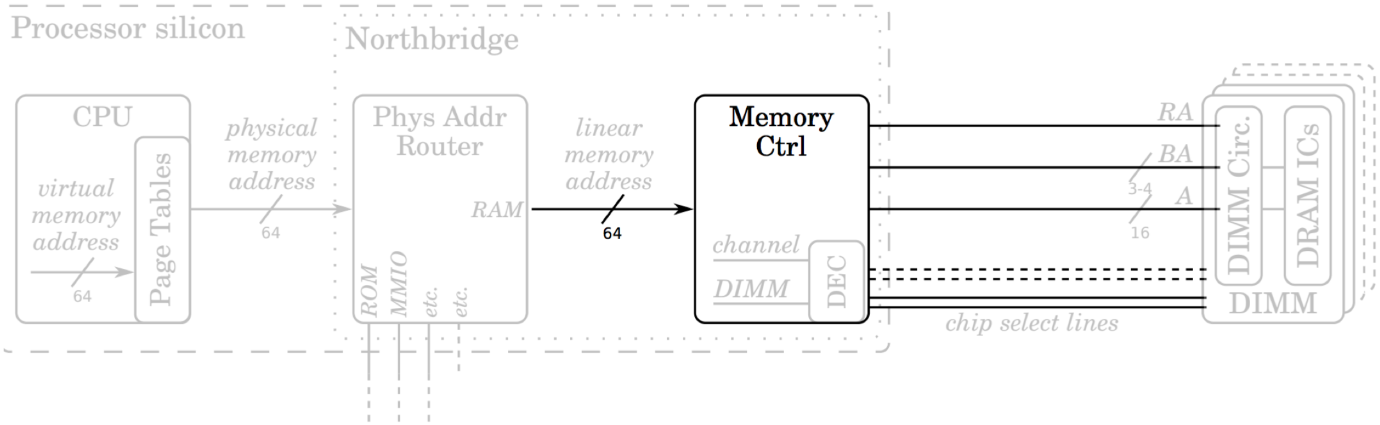
Challenge 1 : Physical address to DRAM address

Virtual address to Physical address:



Challenge 1 : Physical address to DRAM address

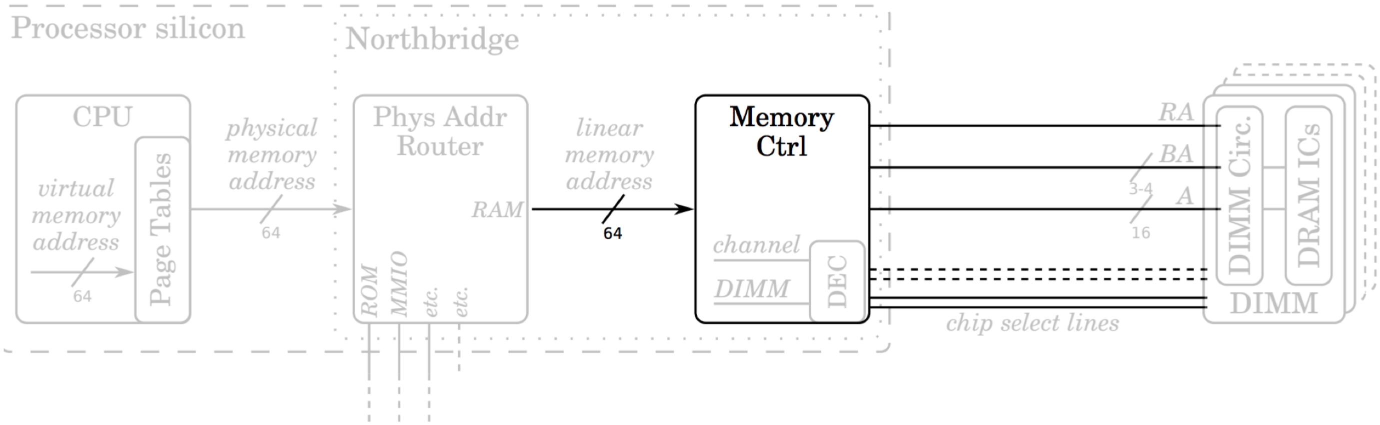
Physical address to DRAM address



Challenge 1 : Physical address to DRAM address

➢ DRAM organized in:

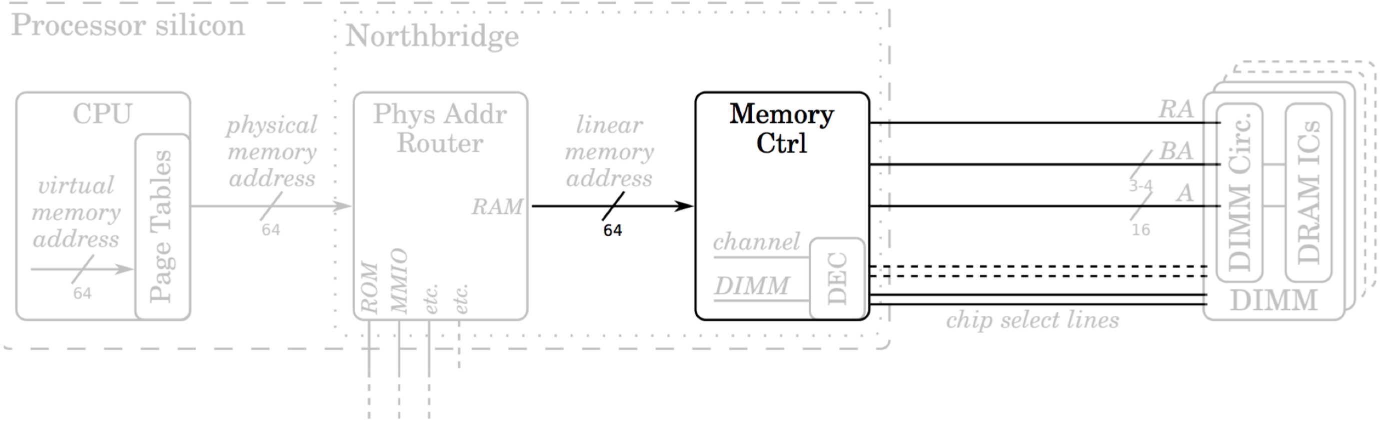
channel



Challenge 1 : Physical address to DRAM address

- DRAM organized in:

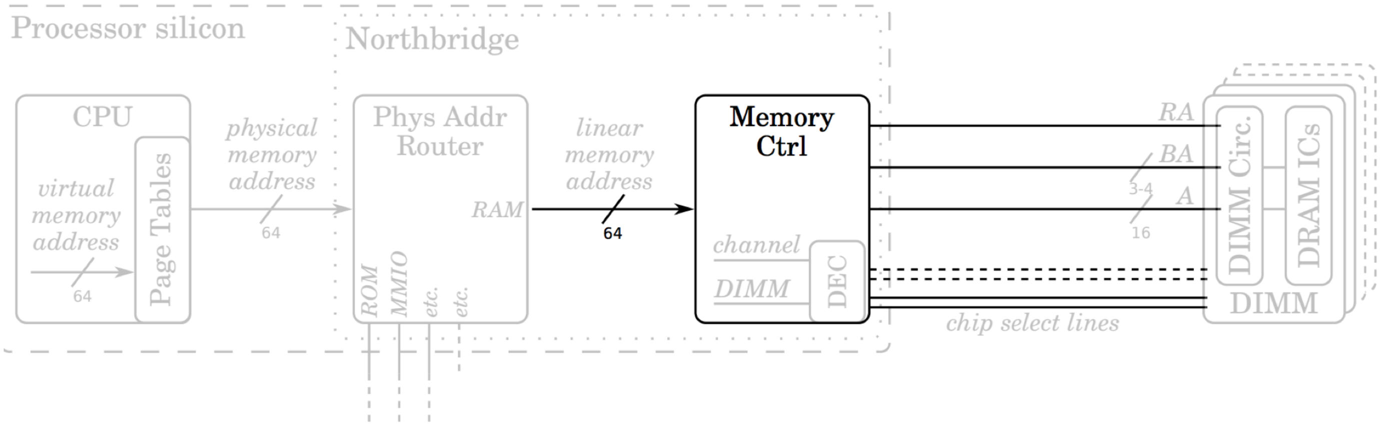
  channel, DIMM



Challenge 1 : Physical address to DRAM address

- DRAM organized in:

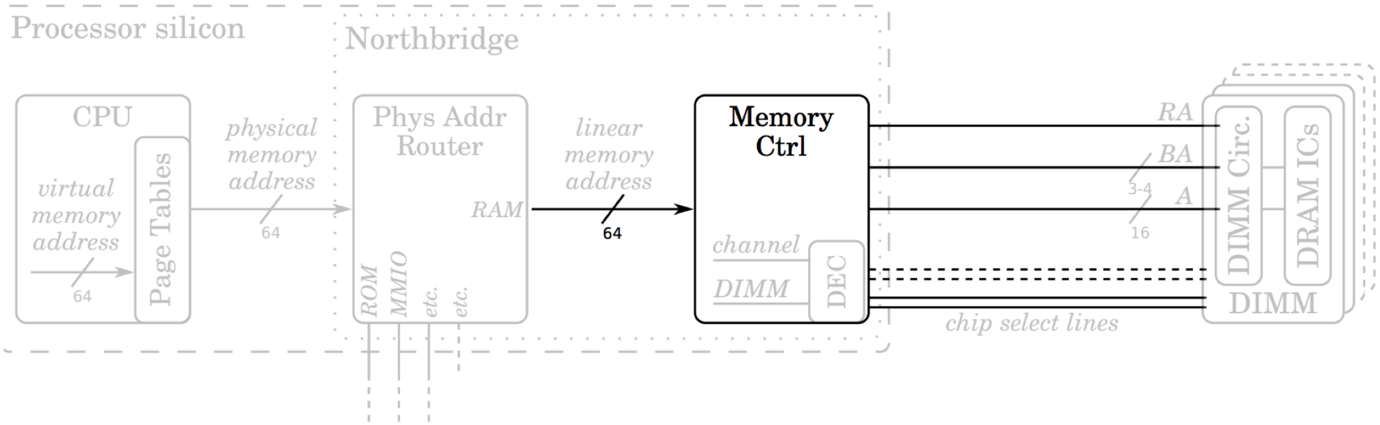
  channel, DIMM, rank



Challenge 1 : Physical address to DRAM address

➢ DRAM organized in:

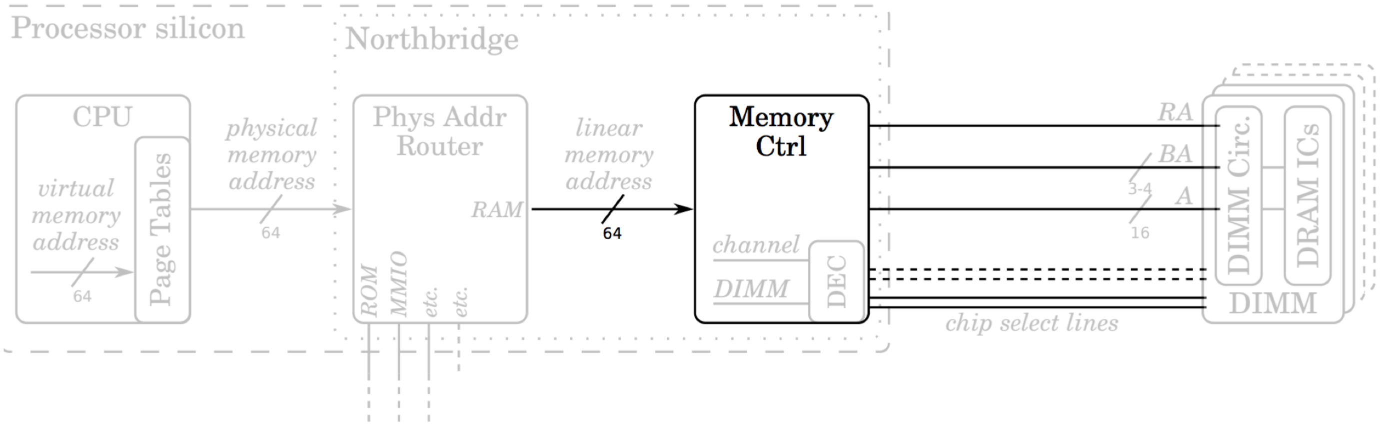
channel, DIMM, rank, bank



Challenge 1 : Physical address to DRAM address

- DRAM organized in:

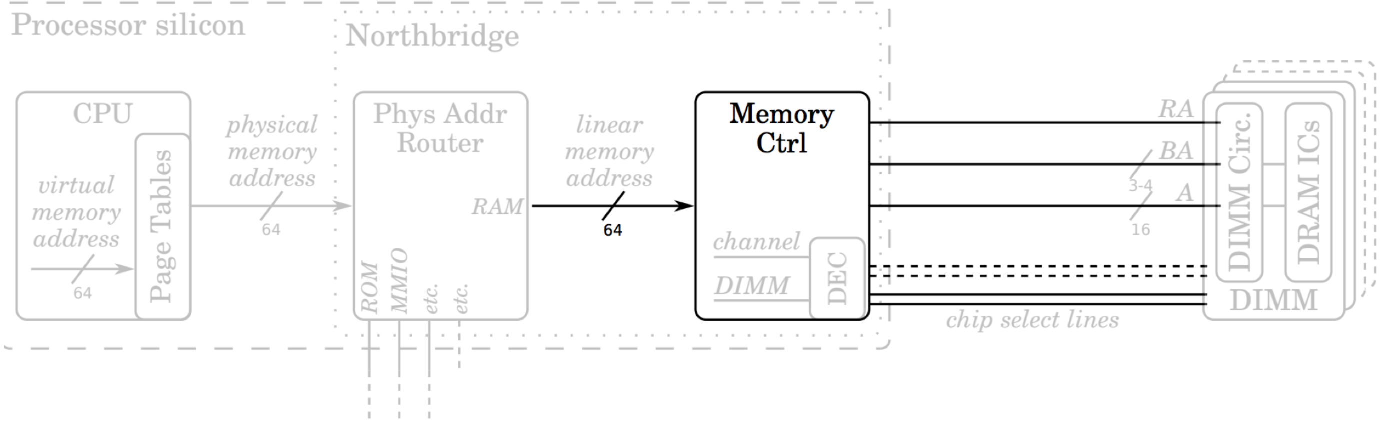
  channel, DIMM, rank, bank, row



Challenge 1 : Physical address to DRAM address

- DRAM organized in:

  channel, DIMM, rank, bank, row, column



Challenge 1 : Physical address to DRAM address

To understand this mapping:

Challenge 1 : Physical address to DRAM address

To understand this mapping:

- ➢ Previous reverse-engineering work (Pessl et al. SEC’16)

Challenge 1 : Physical address to DRAM address

To understand this mapping:

- ➢ Previous reverse-engineering work (Pessl et al. SEC'16)
- ➢ More reverse engineering

Challenge 1 : Physical address to DRAM address

To understand this mapping:

- ➢ Previous reverse-engineering work (Pessl et al. SEC’16)
- ➢ More reverse engineering

DRAM address translation library, **RAMSES**

Challenge 1 : Physical address to DRAM address

To understand this mapping:

- ➢ Previous reverse-engineering work (Pessl et al. SEC'16)
- ➢ More reverse engineering

DRAM address translation library, **RAMSES**

Memory allocator, **ALIS** (Tatar et al. ATC'18)

Challenge 1 : Physical address to DRAM address

To understand this mapping:

- ➢ Previous reverse-engineering work (Pessl et al. SEC'16)
- ➢ More reverse engineering

DRAM address translation library, **RAMSES**

Memory allocator, **ALIS** (Tatar et al. ATC'18)

Challenge 1 : Physical address to DRAM address

To understand this mapping:

- ➢ Previous reverse-engineering work (Pessl et al. SEC'16)
- ➢ More reverse engineering

DRAM address translation library, **RAMSES**

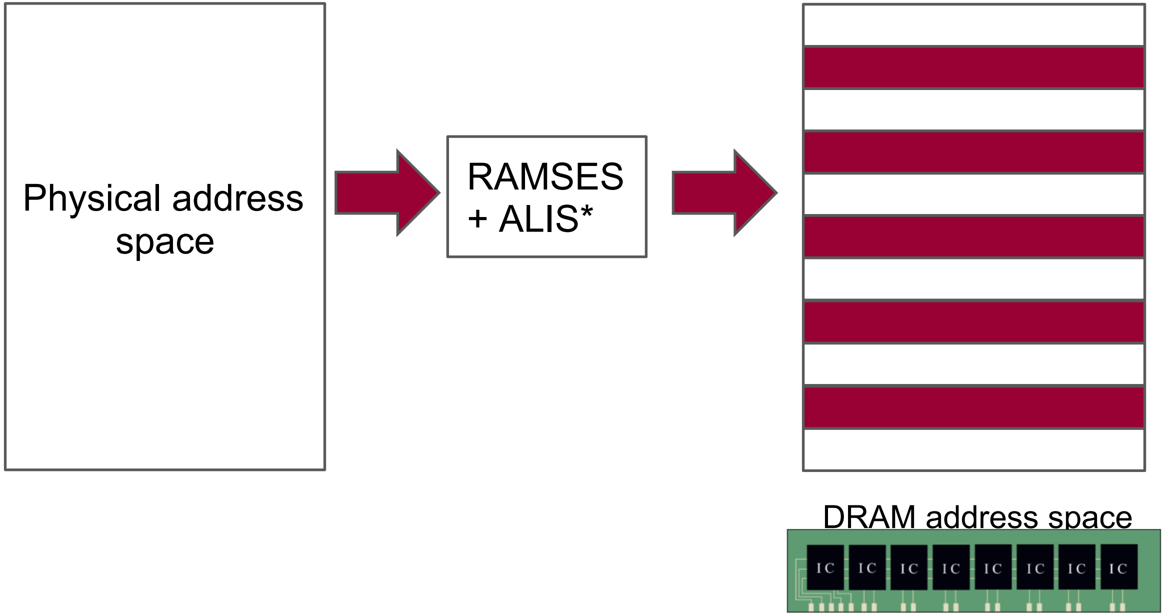
Memory allocator, **ALIS** (Tatar et al. ATC'18)

For ZebRAM, we extended ALIS…

...to allocate memory in zebra pattern.

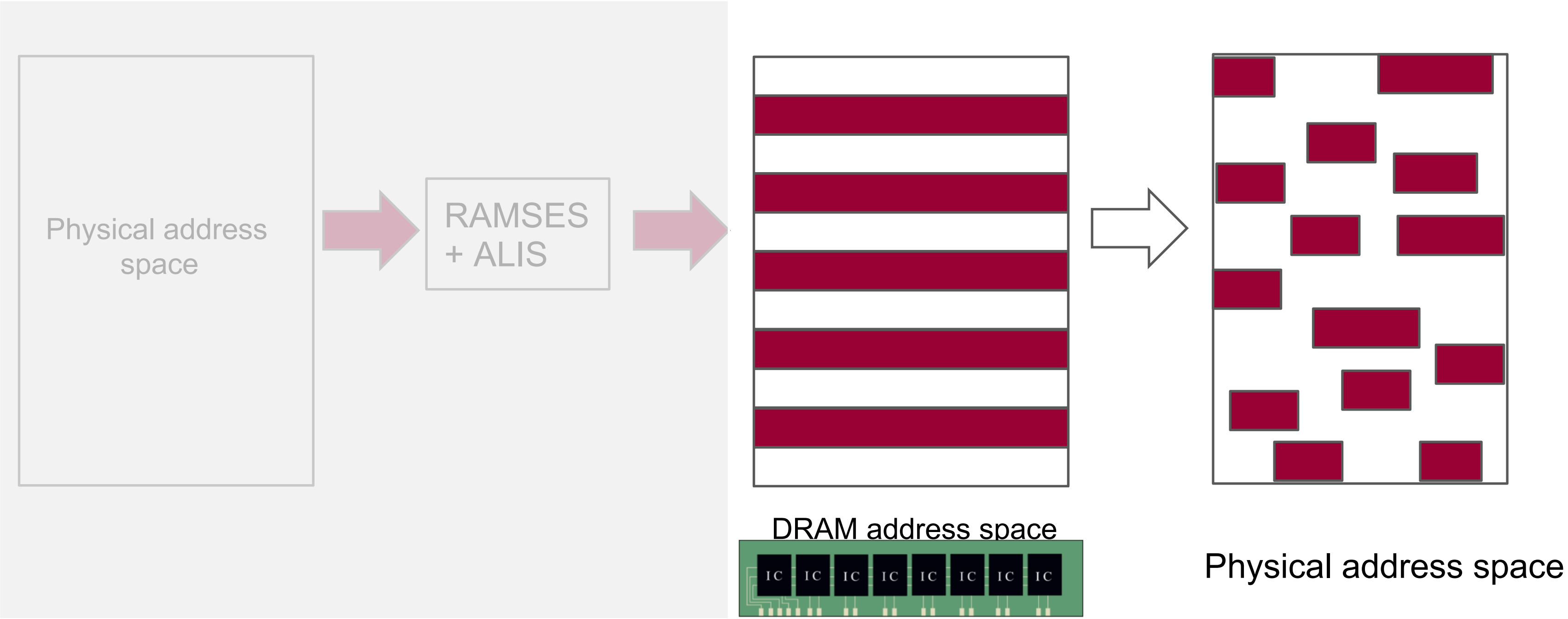
ZebRAM Challenge 1

1. Translating physical addresses to DRAM addresses and placing guard rows



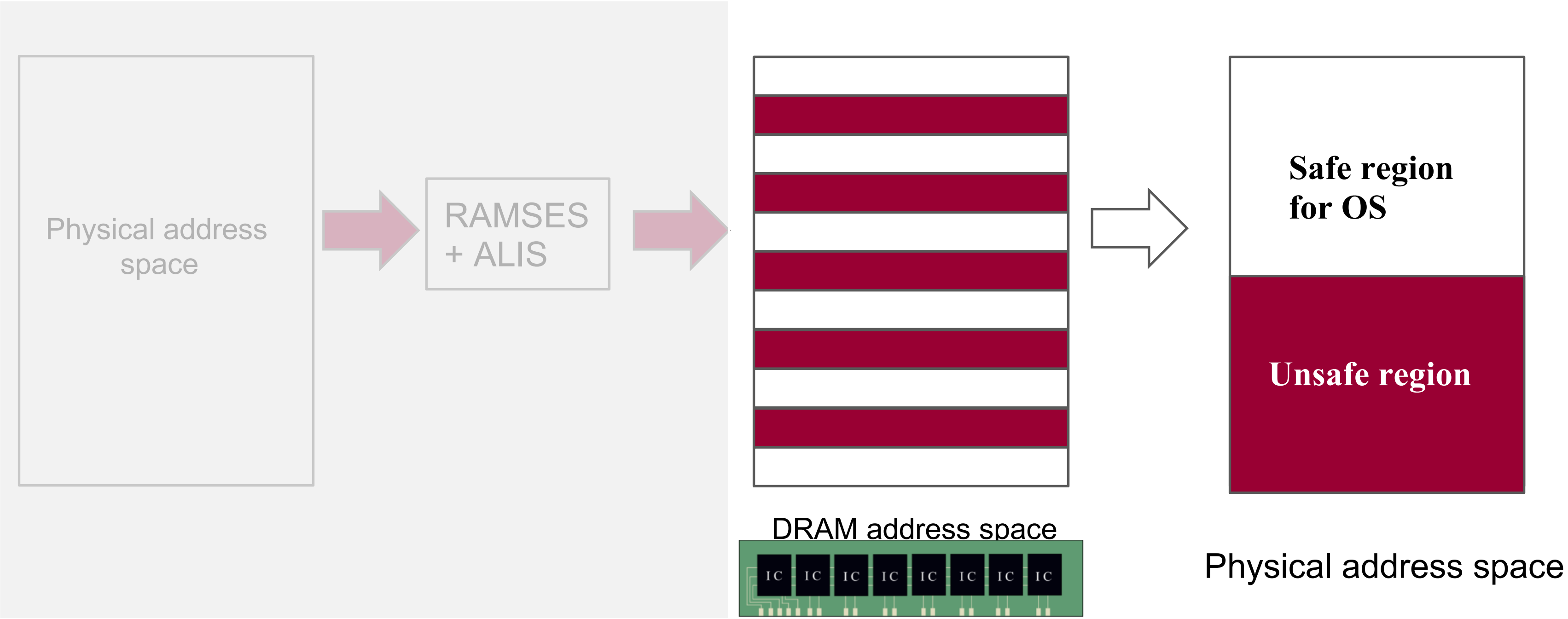
Challenge 2 : Re-mapping physical address space

2. Transparently re-map the data rows and guard rows as two contiguous memory region



Challenge 2 : Re-mapping physical address space

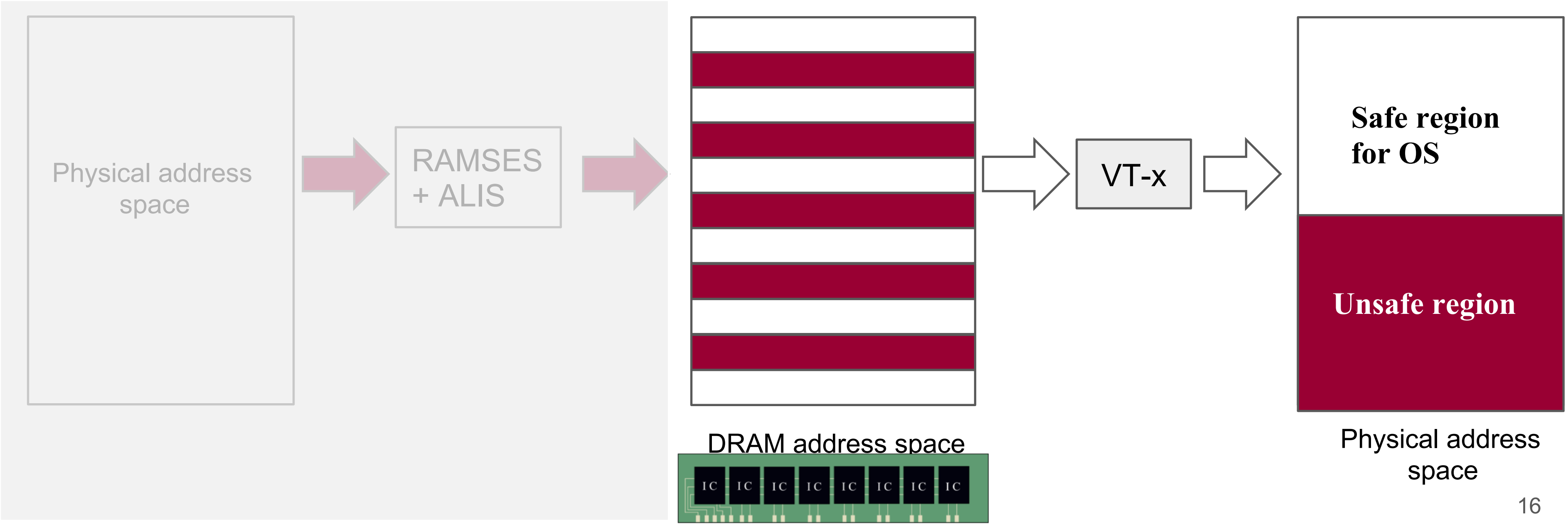
2. Transparently re-map the data rows and guard rows as two contiguous memory region



Challenge 2 : Re-mapping physical address space

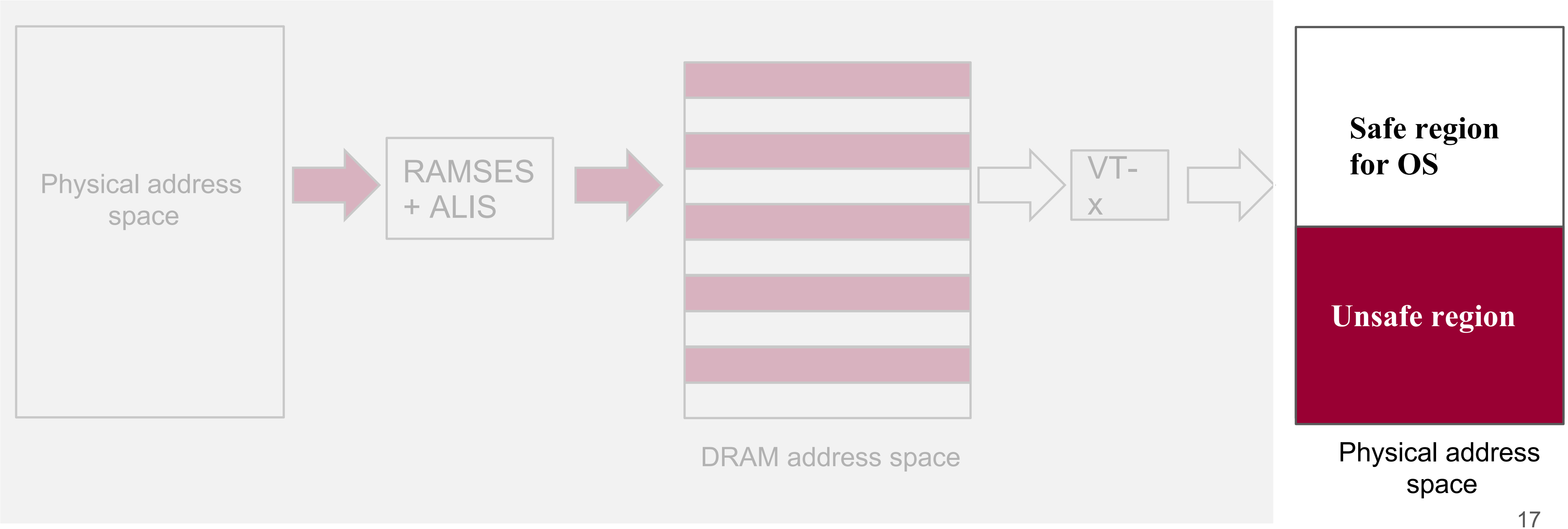
We use virtualization feature like Intel (VT-x) …

…to **transparently** re-map the guard and data rows as two contiguous memory region



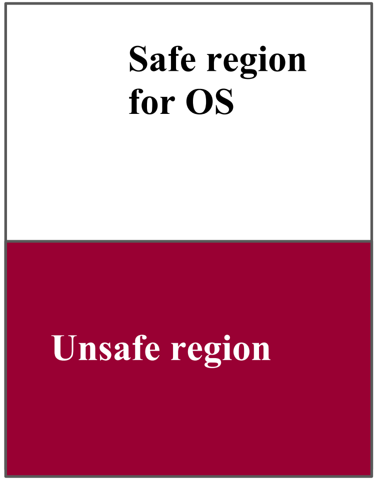
ZebRAM Challenge 3

3. Utilizing the unsafe region **securely** and **efficiently**



Challenge 3 : Utilizing unsafe region

Securely means two things here :

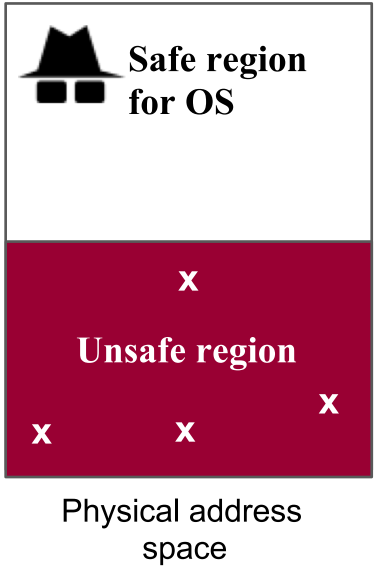


Physical address space

Challenge 3 : Utilizing unsafe region

Securely means two things here :

1. **Handle bit flips that may occur on unsafe region**



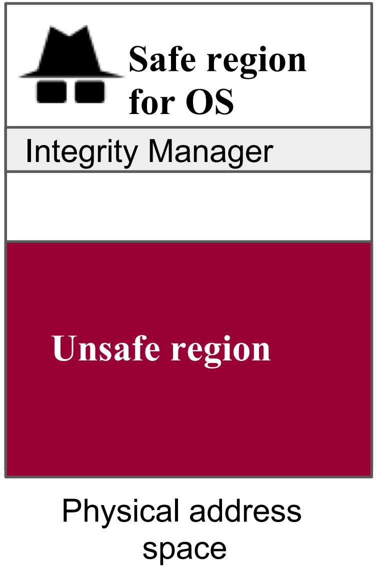
Challenge 3 : Utilizing unsafe region

Securely means two things here :

1. **Handle bit flips that may occur on unsafe region**

ZebRAM implements a **integrity manager** that uses:

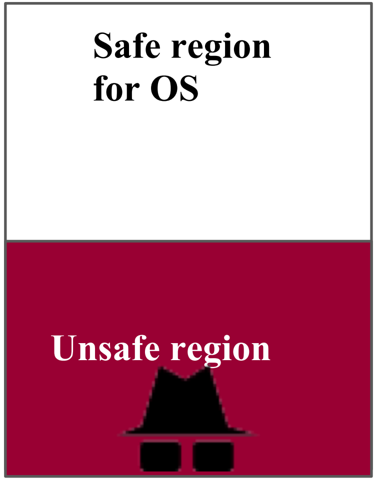
1. Hash verification (SHA-256)
2. Error correction code (ECC)



Challenge 3 : Utilizing unsafe region

Securely means two things:

1. Handle bit flips that may occur on unsafe region
2. **Protect the unsafe region from illegal bit flips**

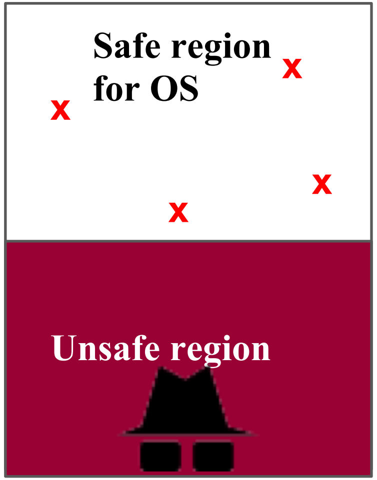


Physical address space

Challenge 3 : Utilizing unsafe region

Securely means two things:

1. Handle bit flips that may occur on unsafe region
2. **Protect the unsafe region from illegal bit flips**



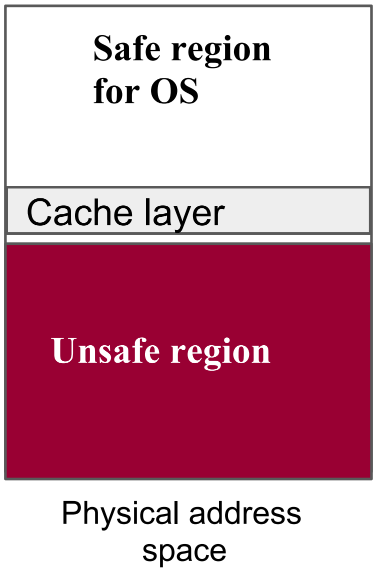
Physical address space

Challenge 3 : Utilizing unsafe region

Securely means two things:

1. Handle bit flips that may occur on unsafe region
2. **Protect the unsafe region from illegal bit flips**

ZebRAM slows down the consecutive accesses to the same location in the unsafe region:



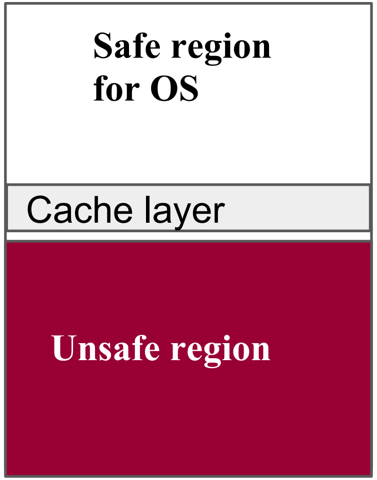
Challenge 3 : Utilizing unsafe region

Securely means two things:

1. Handle bit flips that may occur on unsafe region
2. **Protect the unsafe region from illegal bit flips**

ZebRAM slows down the consecutive accesses to the same location in the unsafe region:

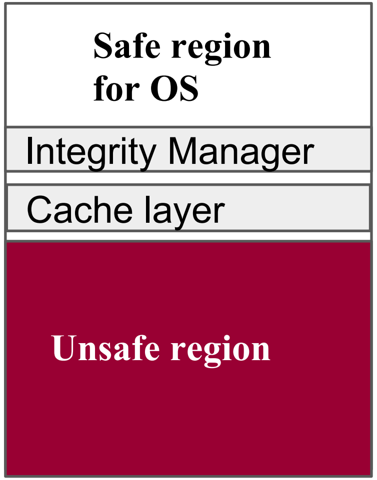
1. By implements a **cache layer** using safe memory
2. Enforcing **Least-recently-added** eviction policy



Physical address space

Challenge 3 : Utilizing unsafe region

Efficiently:

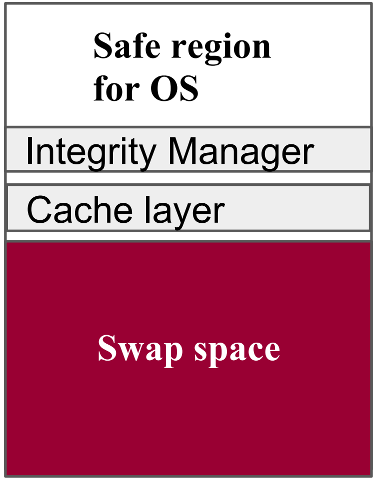


Physical address space

Challenge 3 : Utilizing unsafe region

Efficiently:

- ➢ Exposes the unsafe region as **swap space** to the OS

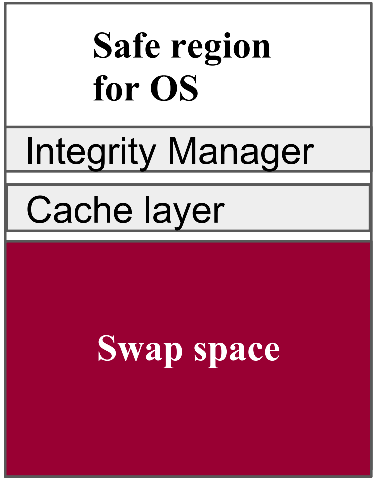


Physical address space

Challenge 3 : Utilizing unsafe region

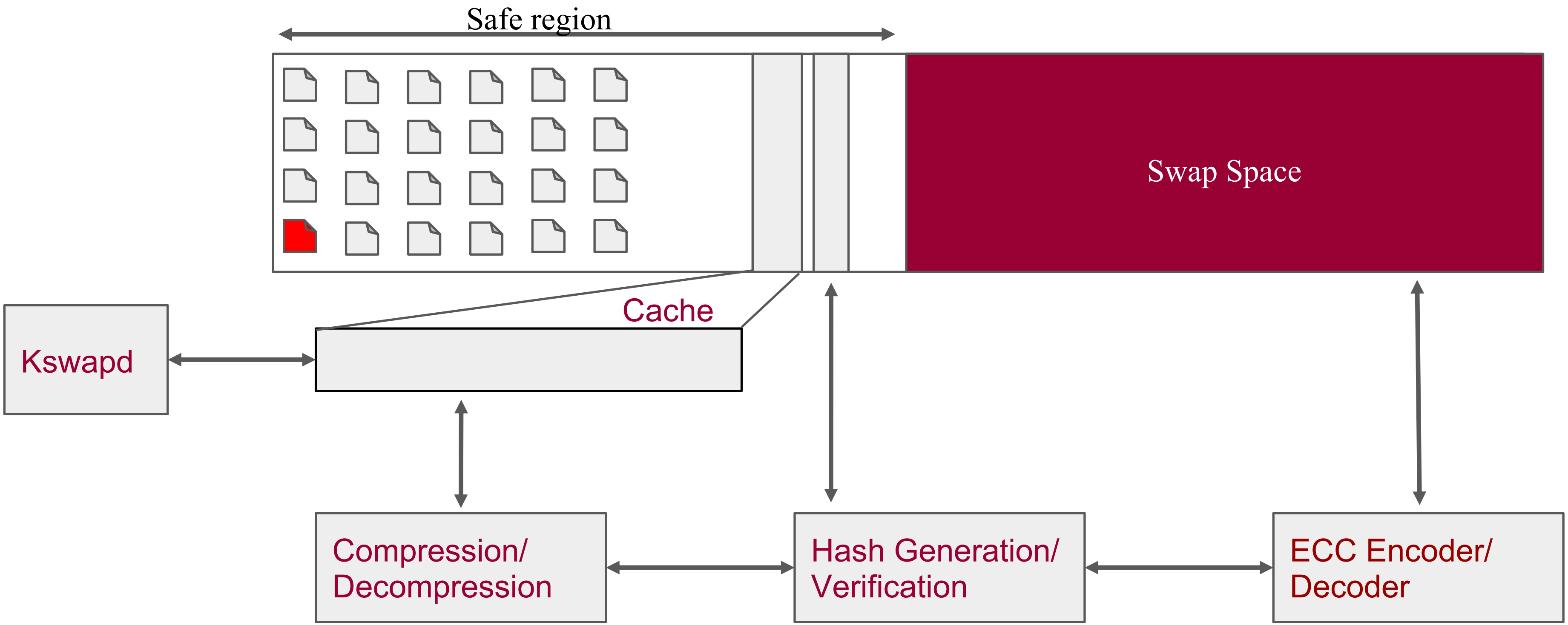
Efficiently:

- ➢ Exposes the unsafe region as **swap space** to the OS
- ➢ Helps to utilize **efficient page replacement policies** in commodity OS

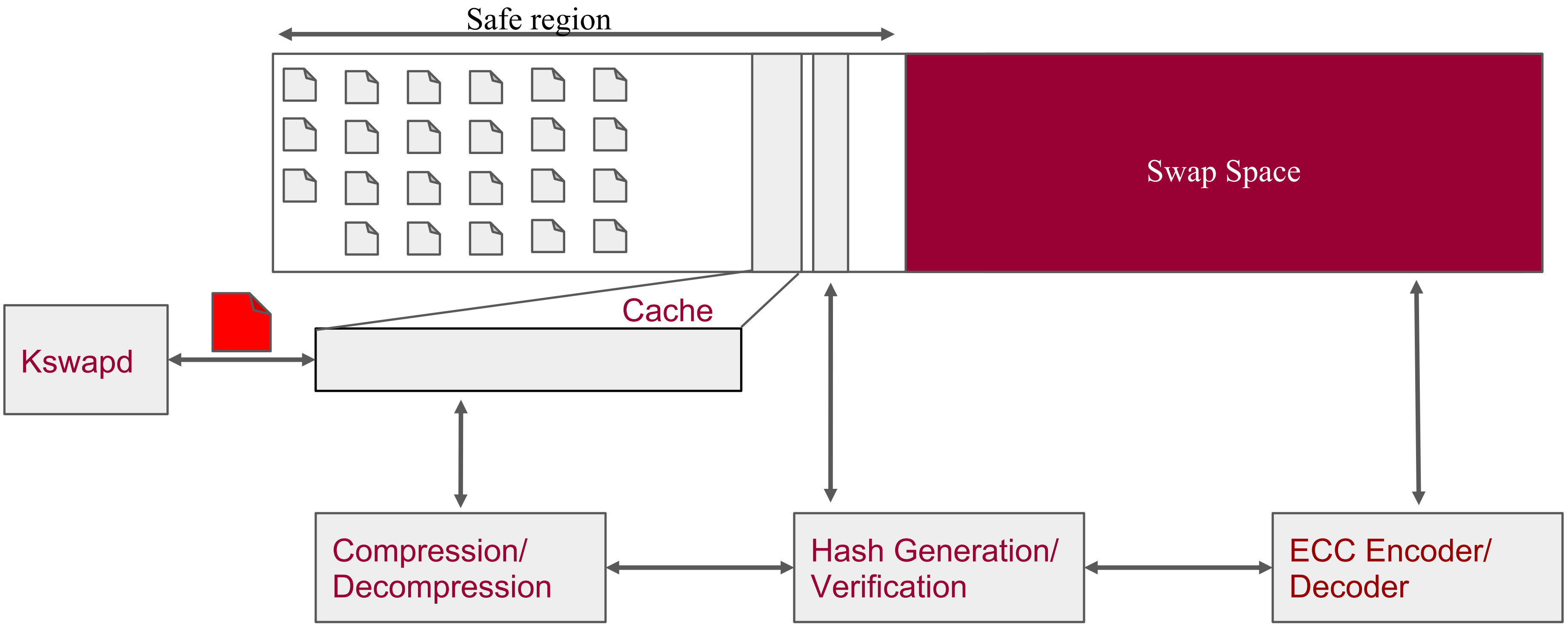


Physical address space

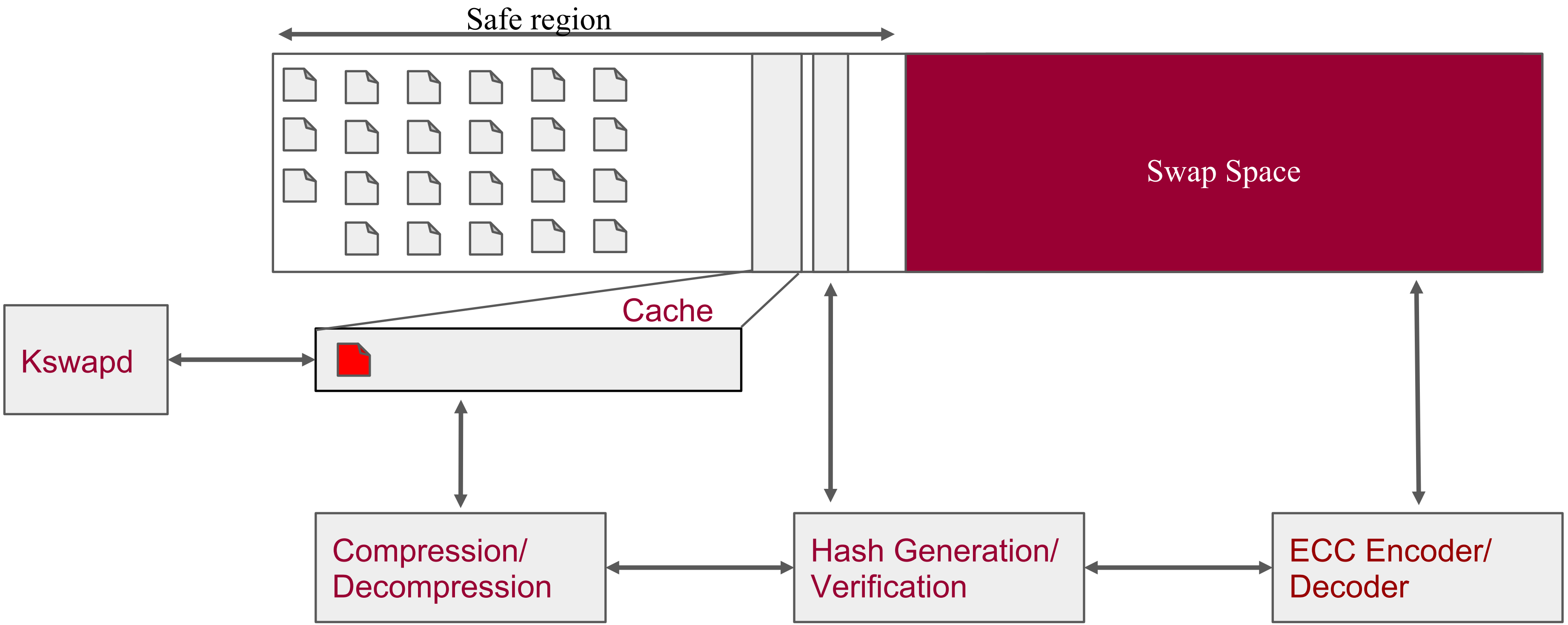
Life of a page in ZebRAM world



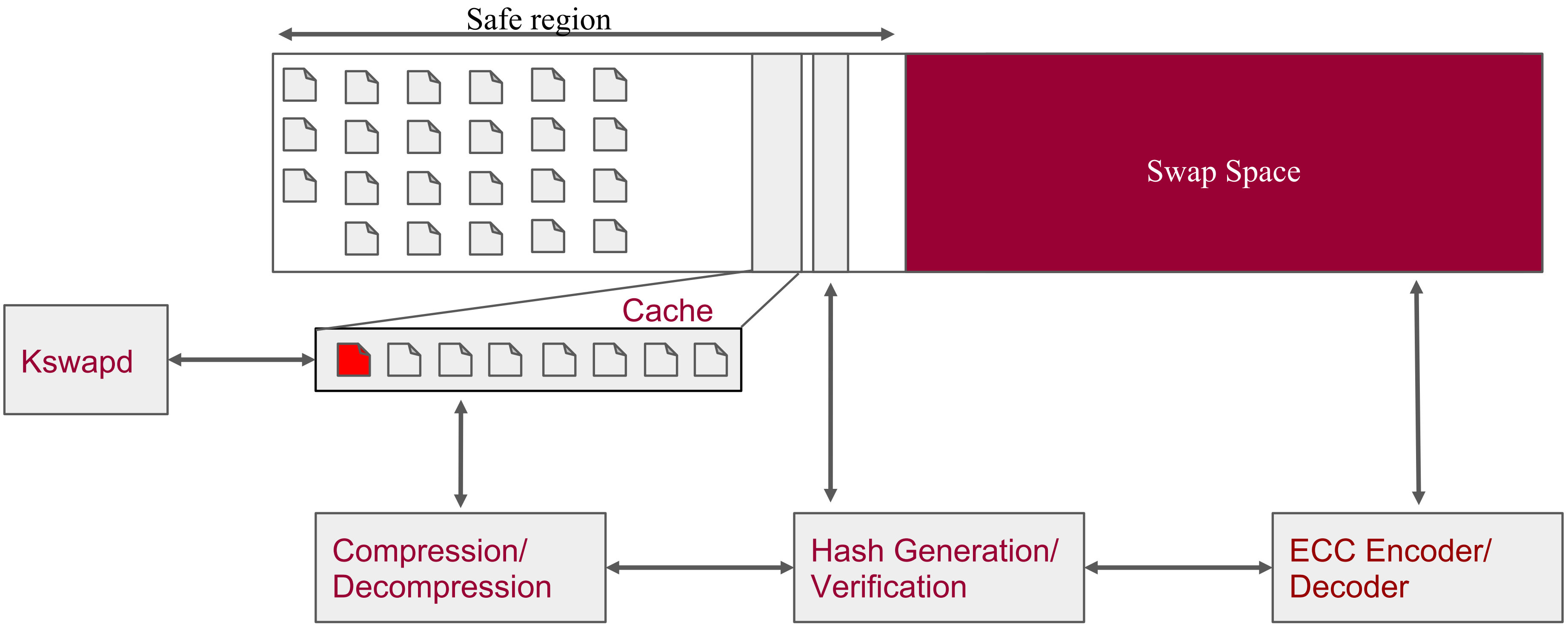
Life of a page in ZebRAM world



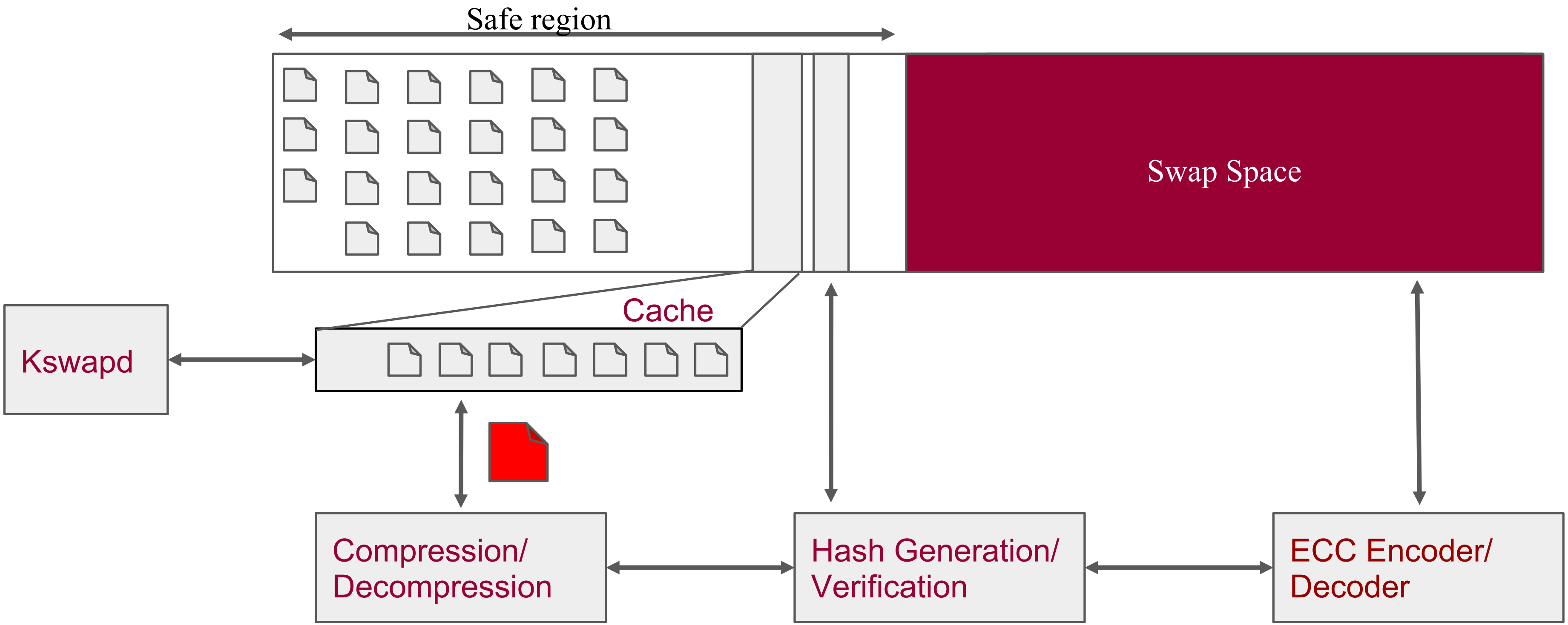
Life of a page in ZebRAM world



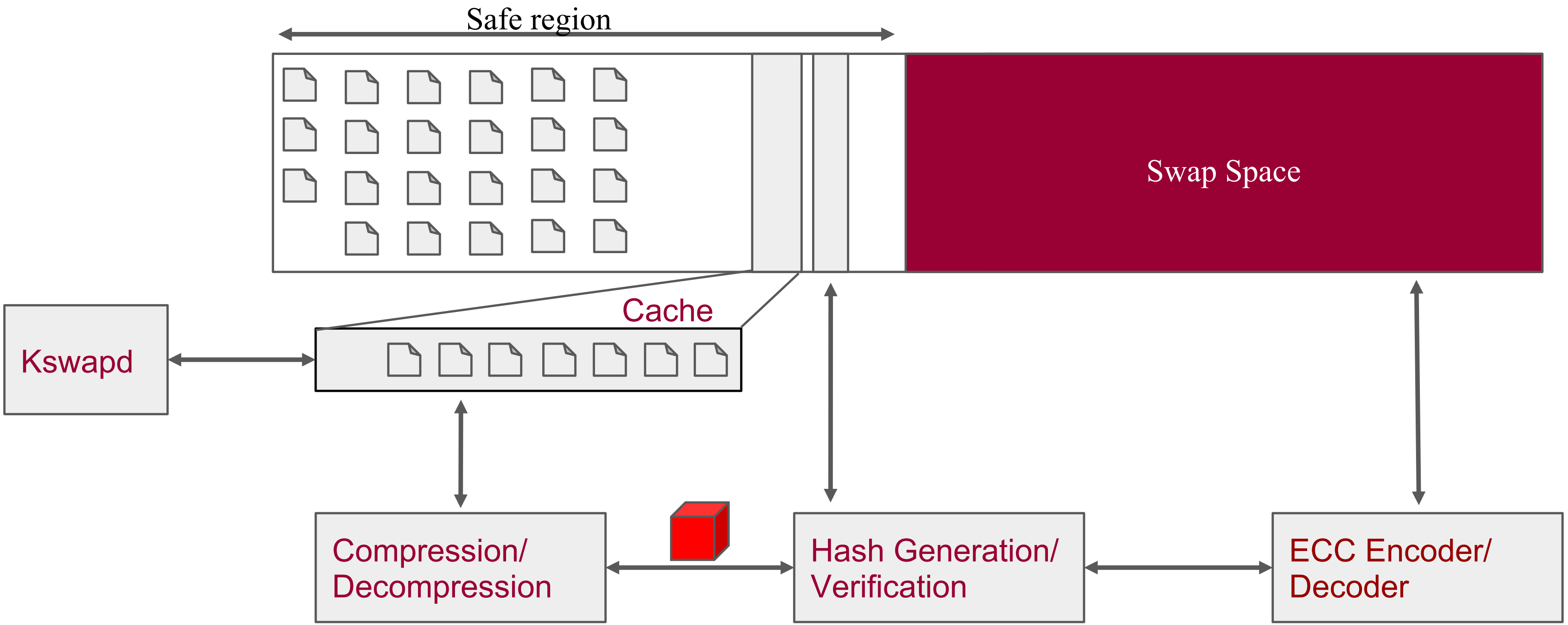
Life of a page in ZebRAM world



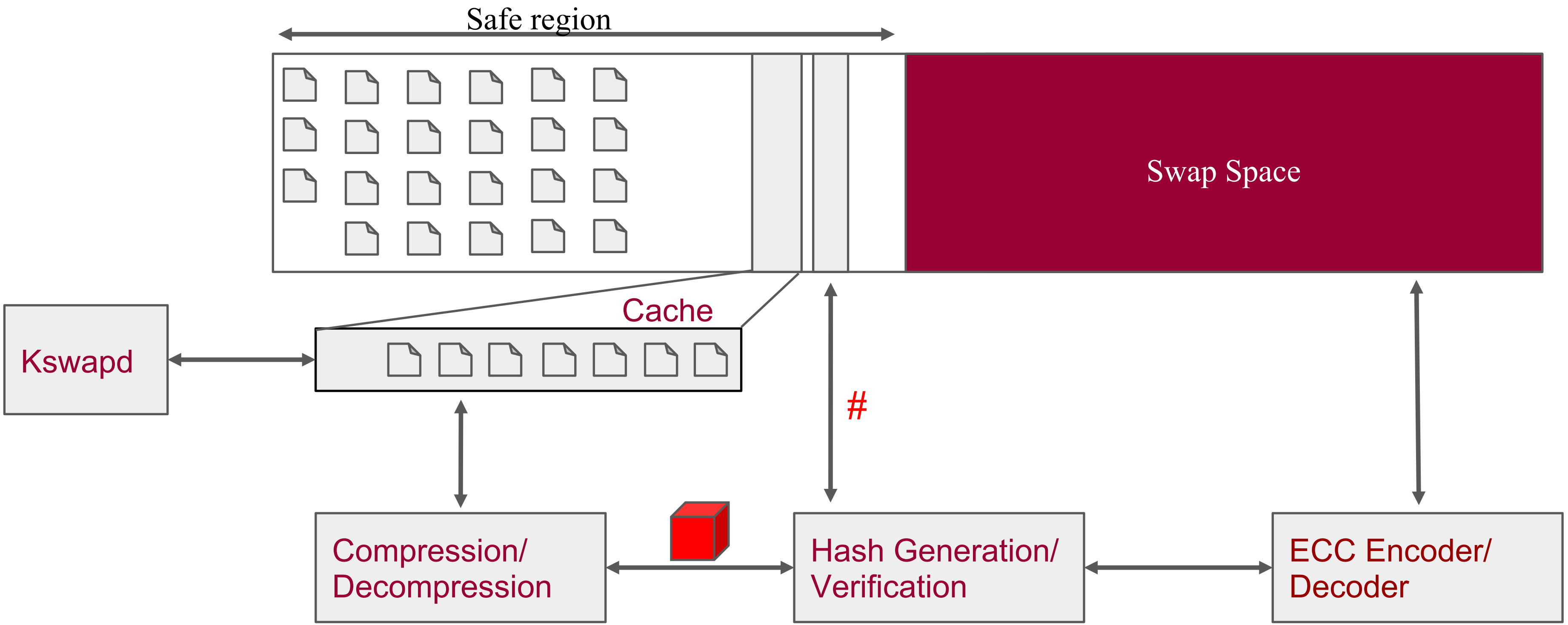
Life of a page in ZebRAM world



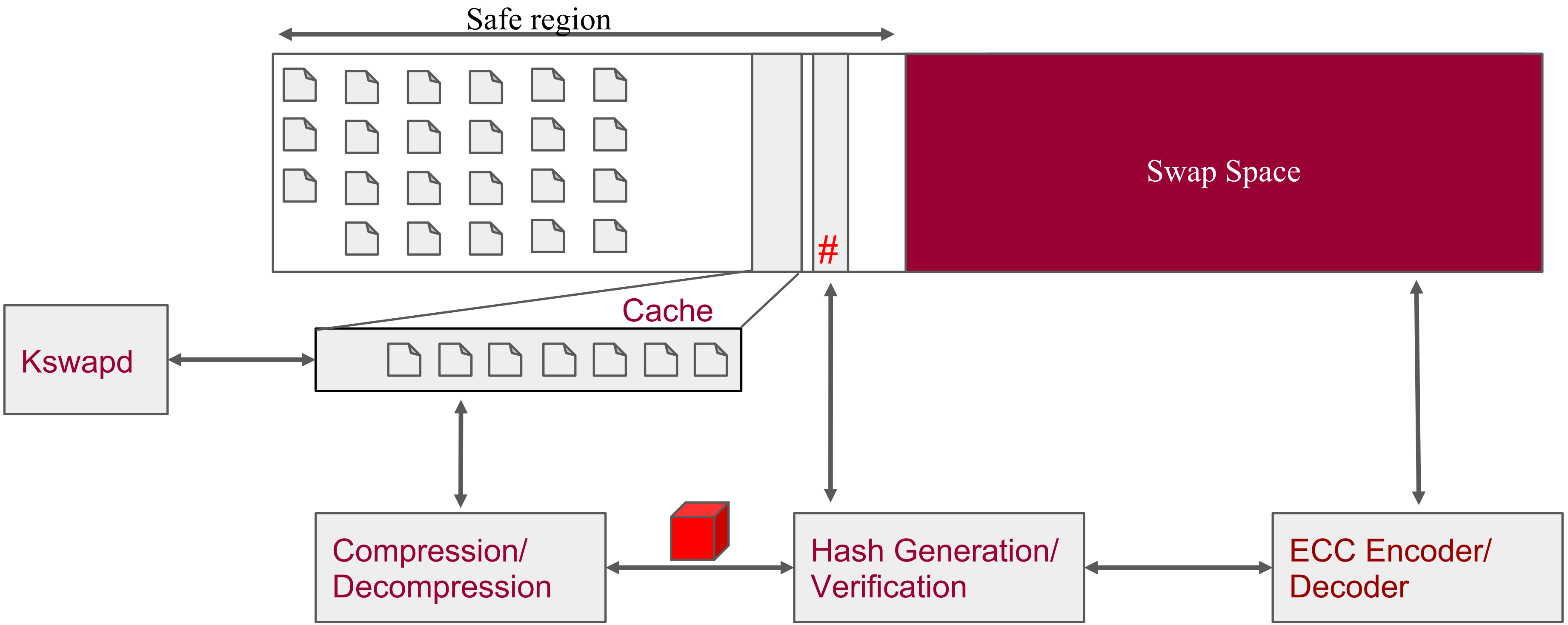
Life of a page in ZebRAM world



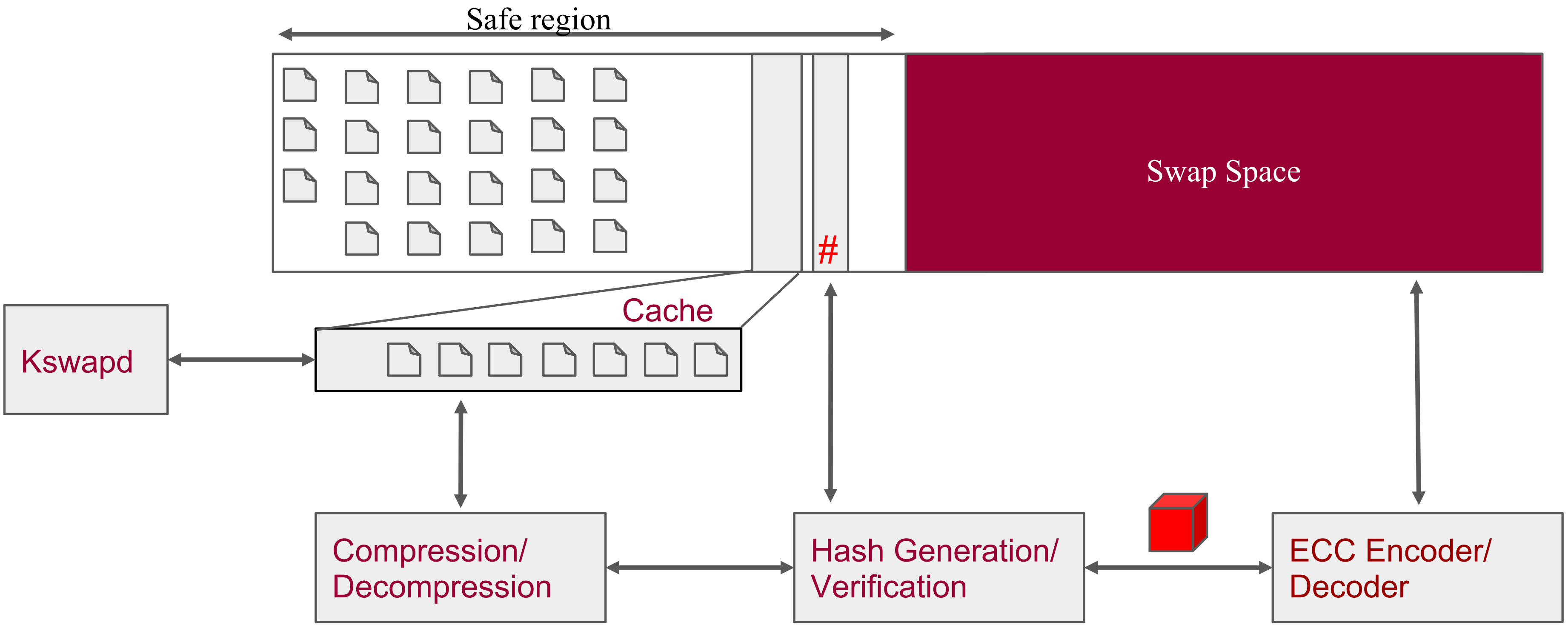
Life of a page in ZebRAM world



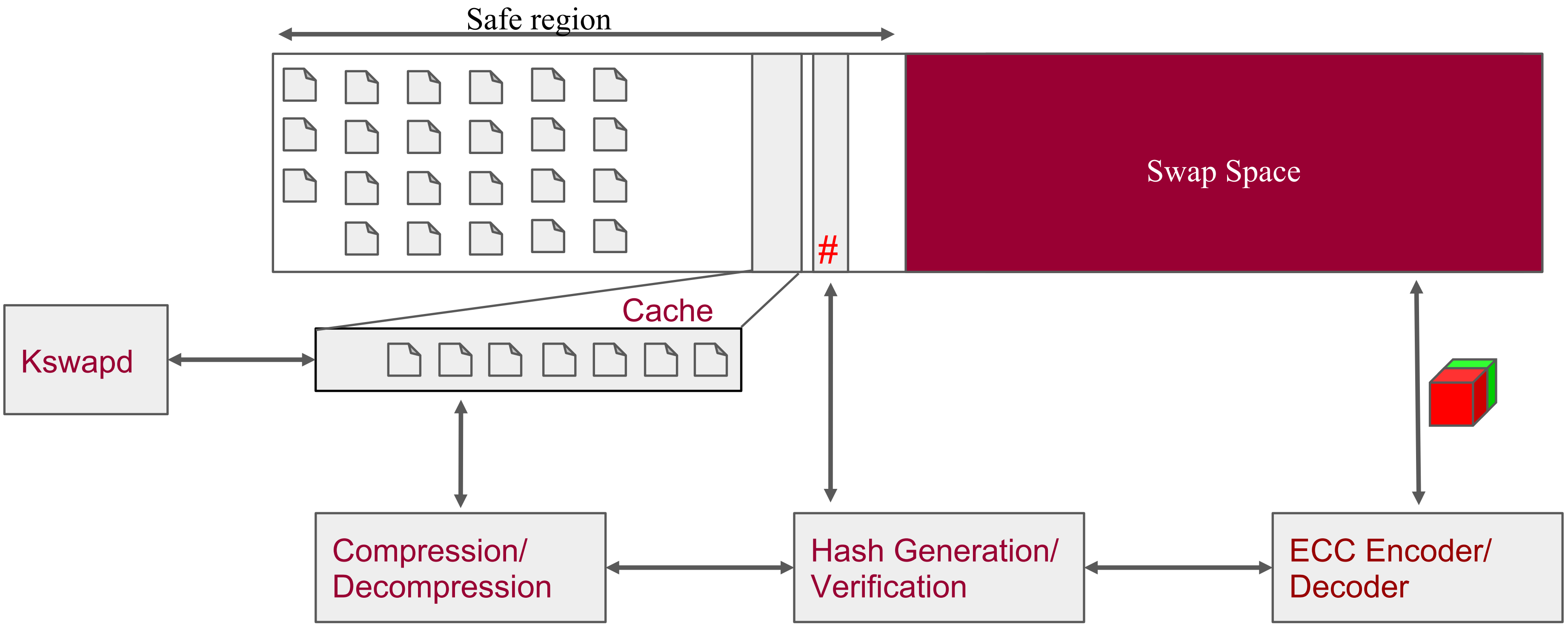
Life of a page in ZebRAM world



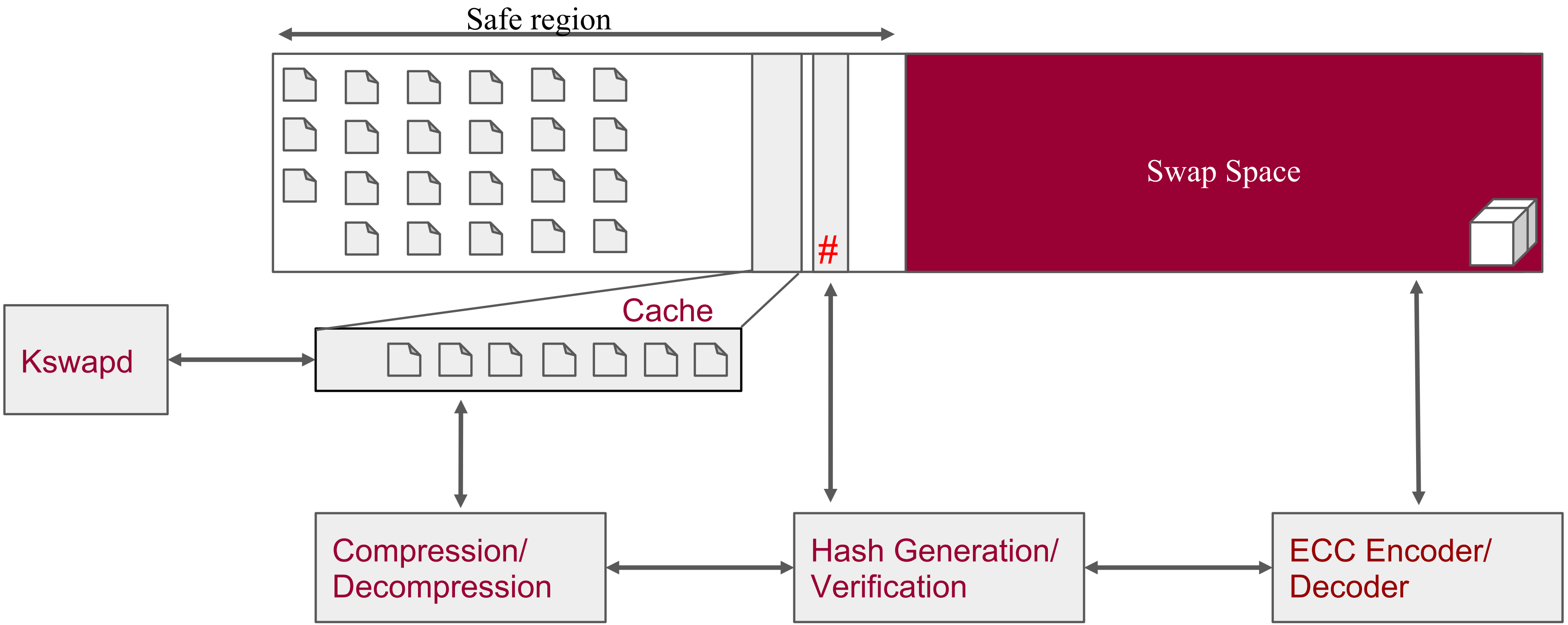
Life of a page in ZebRAM world



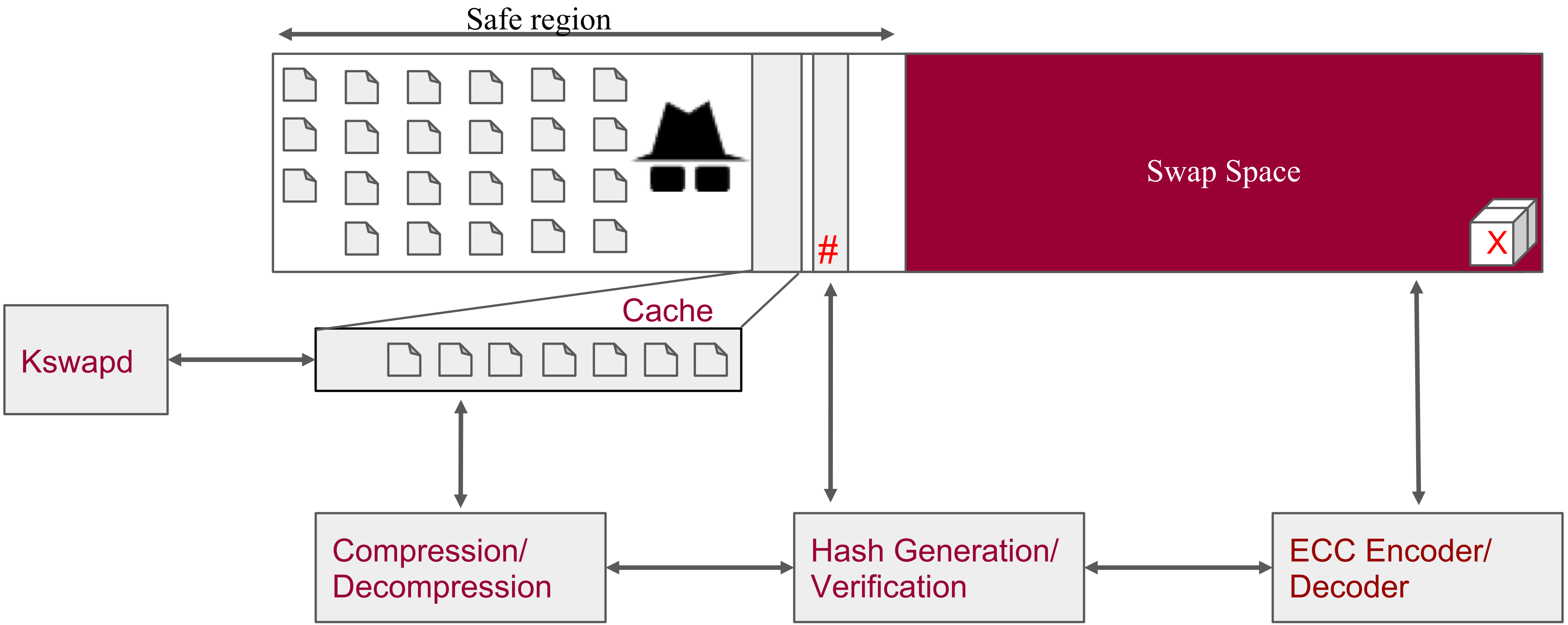
Life of a page in ZebRAM world



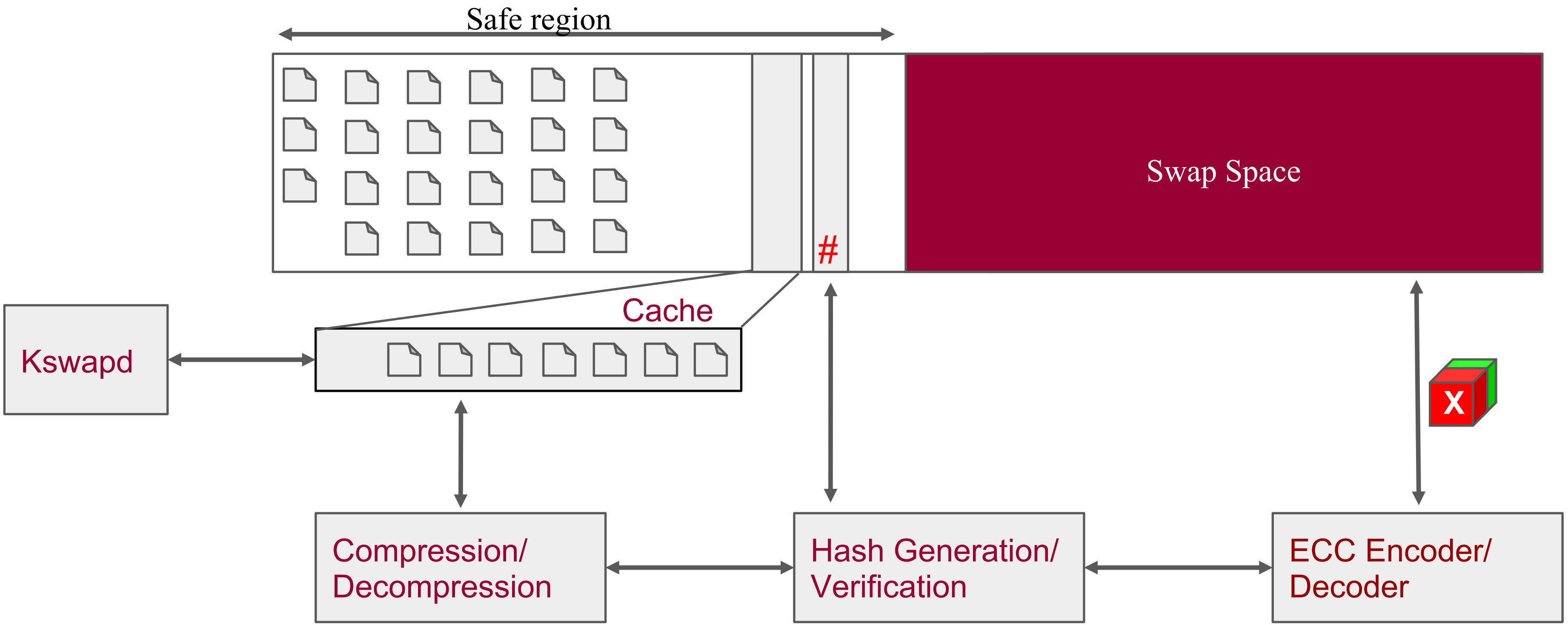
Life of a page in ZebRAM world



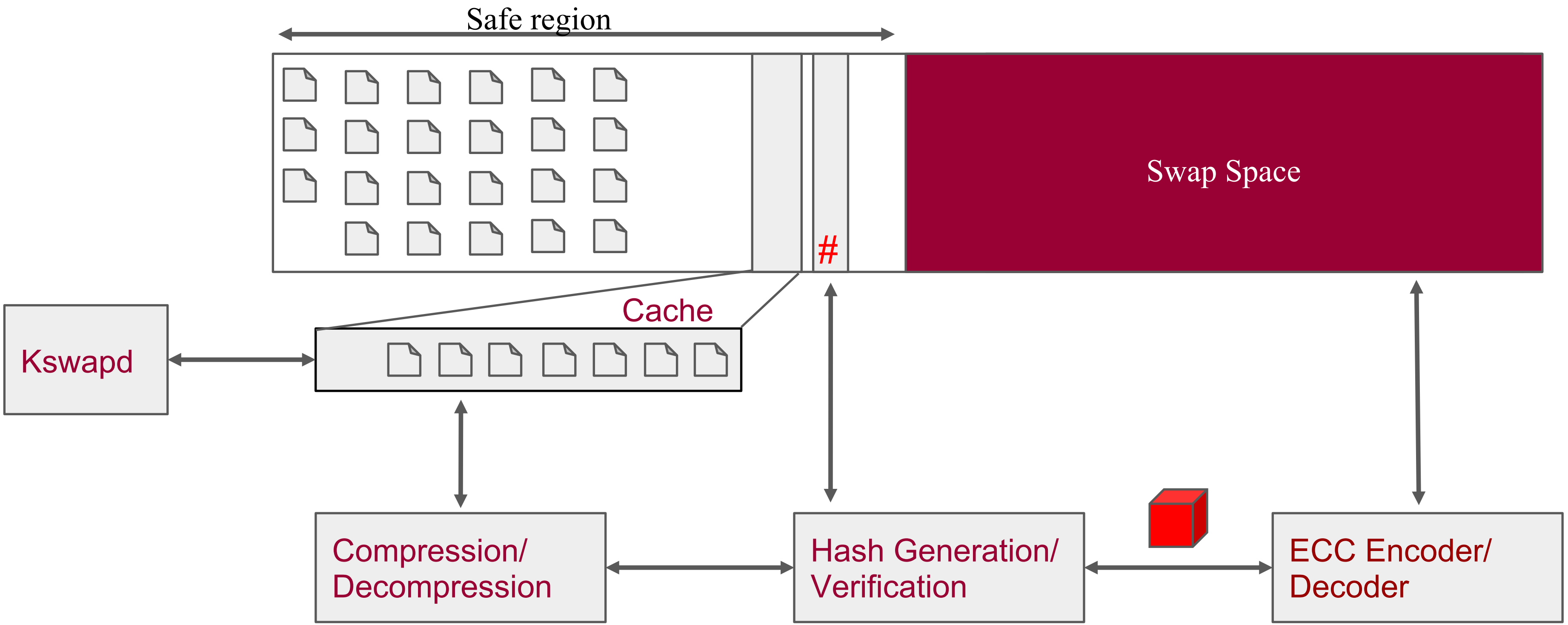
Life of a page in ZebRAM world



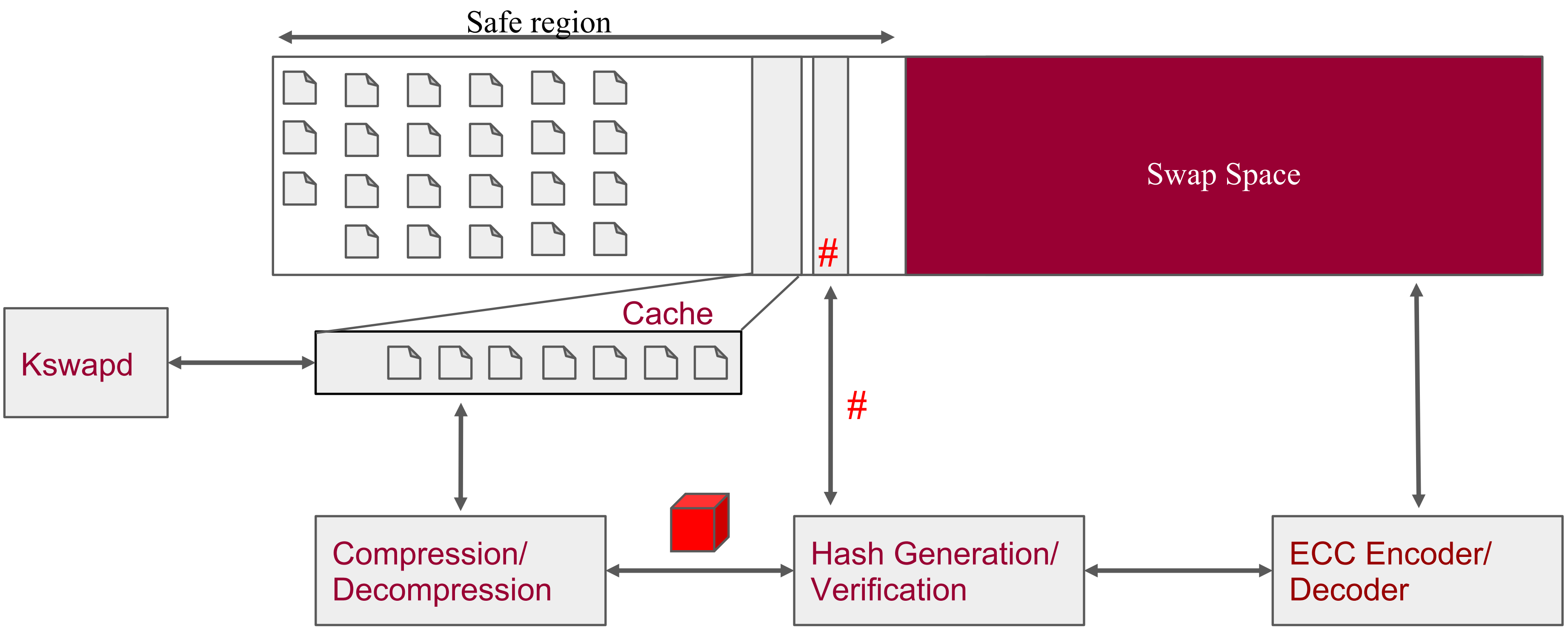
Life of a page in ZebRAM world



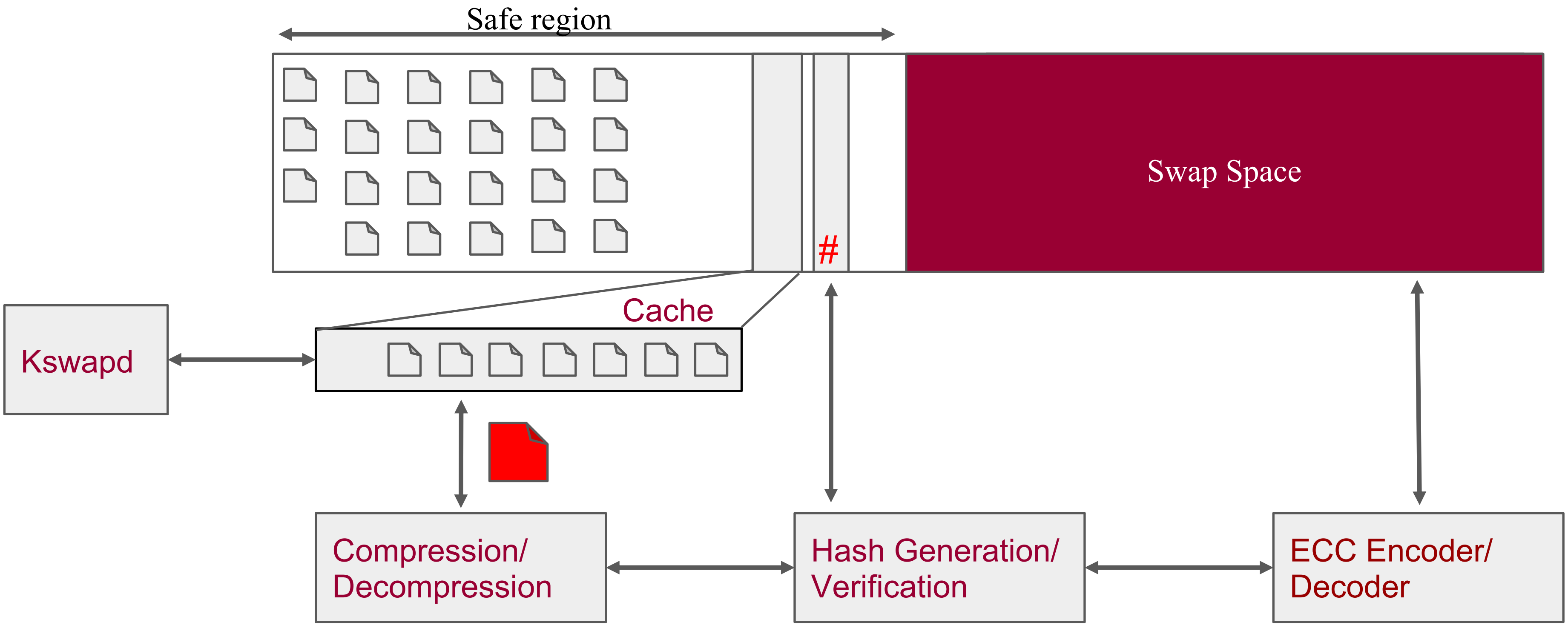
Life of a page in ZebRAM world



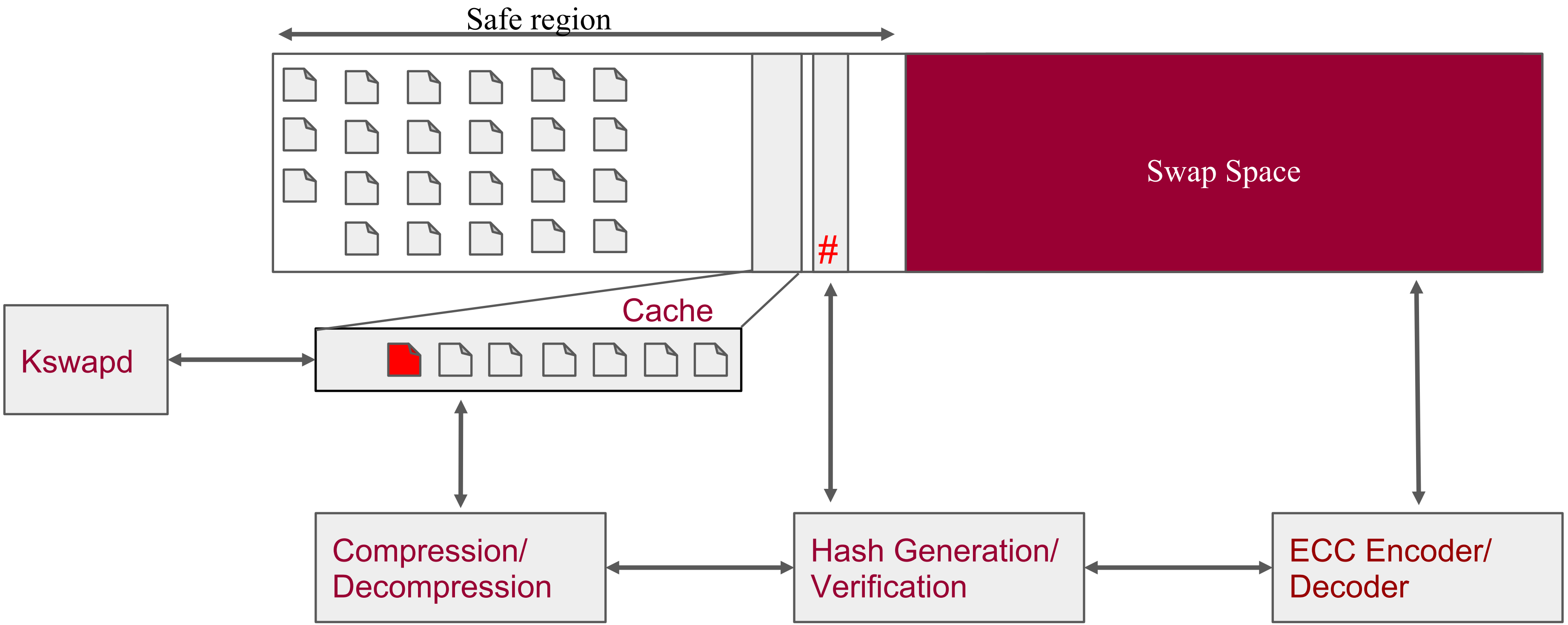
Life of a page in ZebRAM world



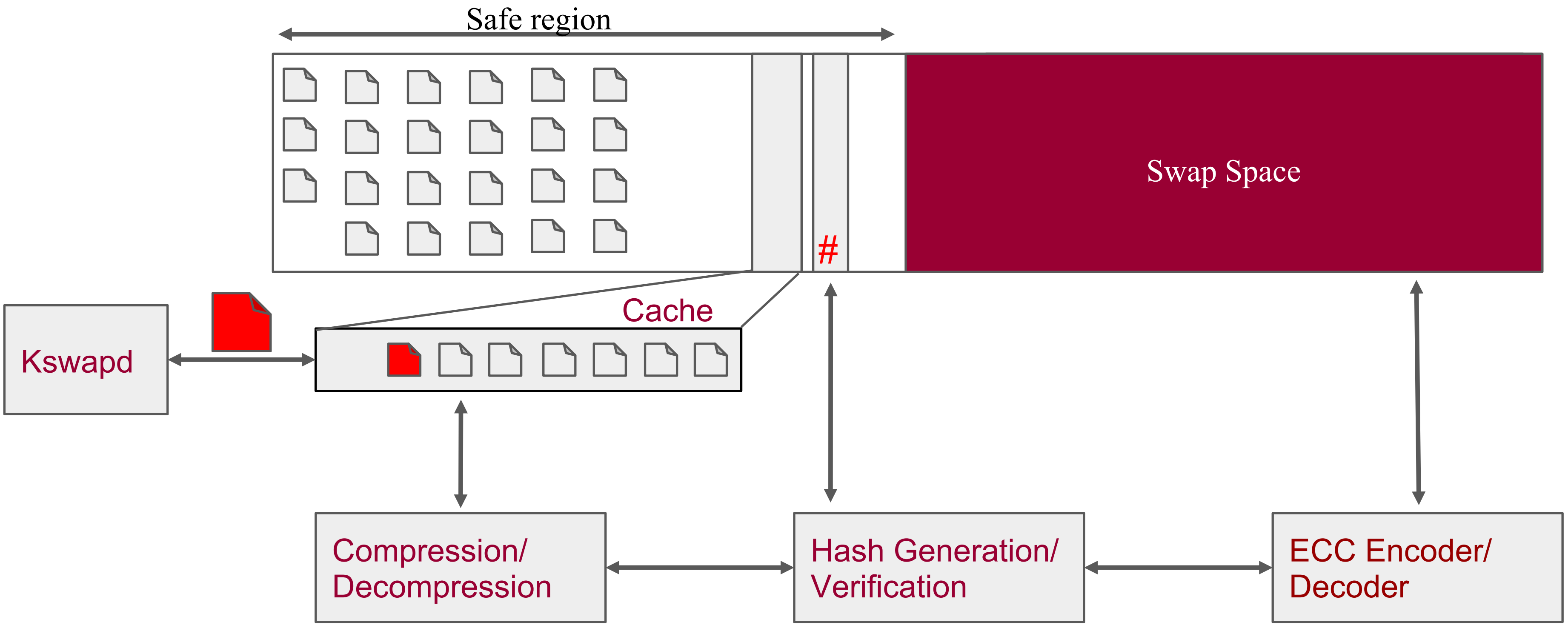
Life of a page in ZebRAM world



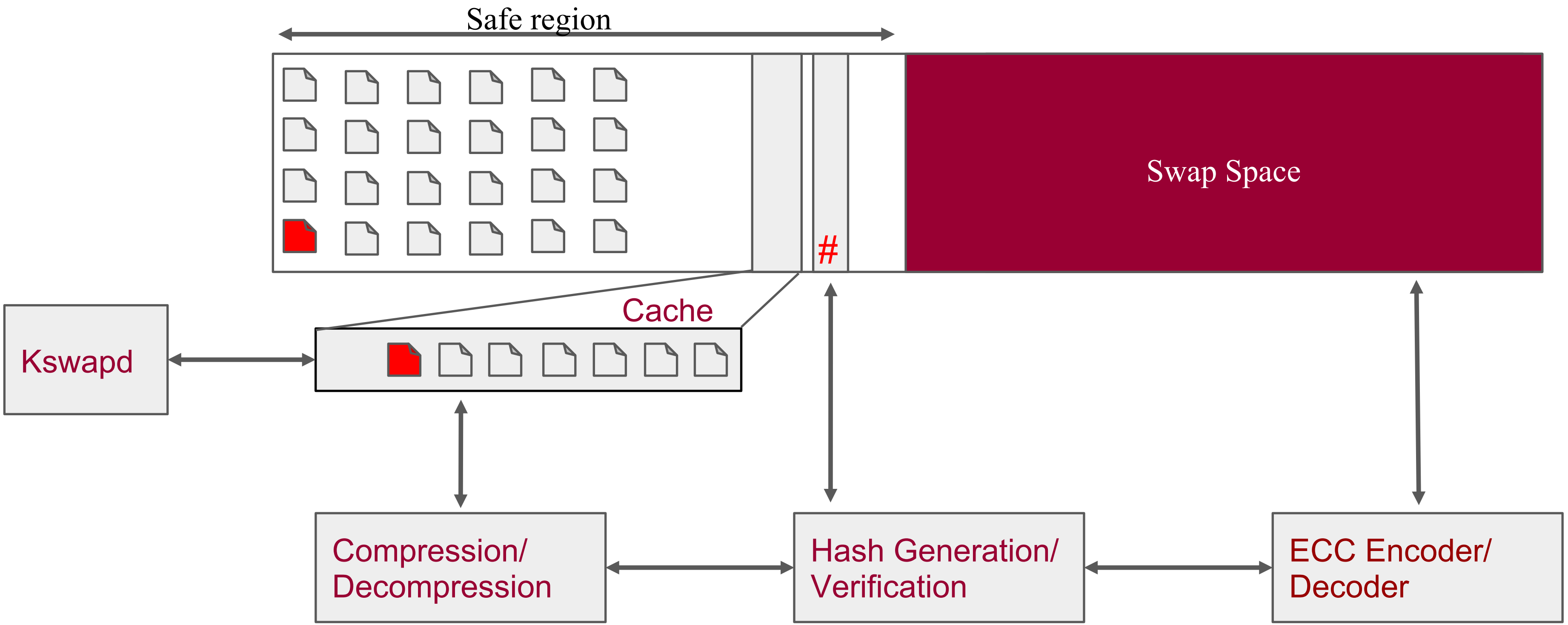
Life of a page in ZebRAM world



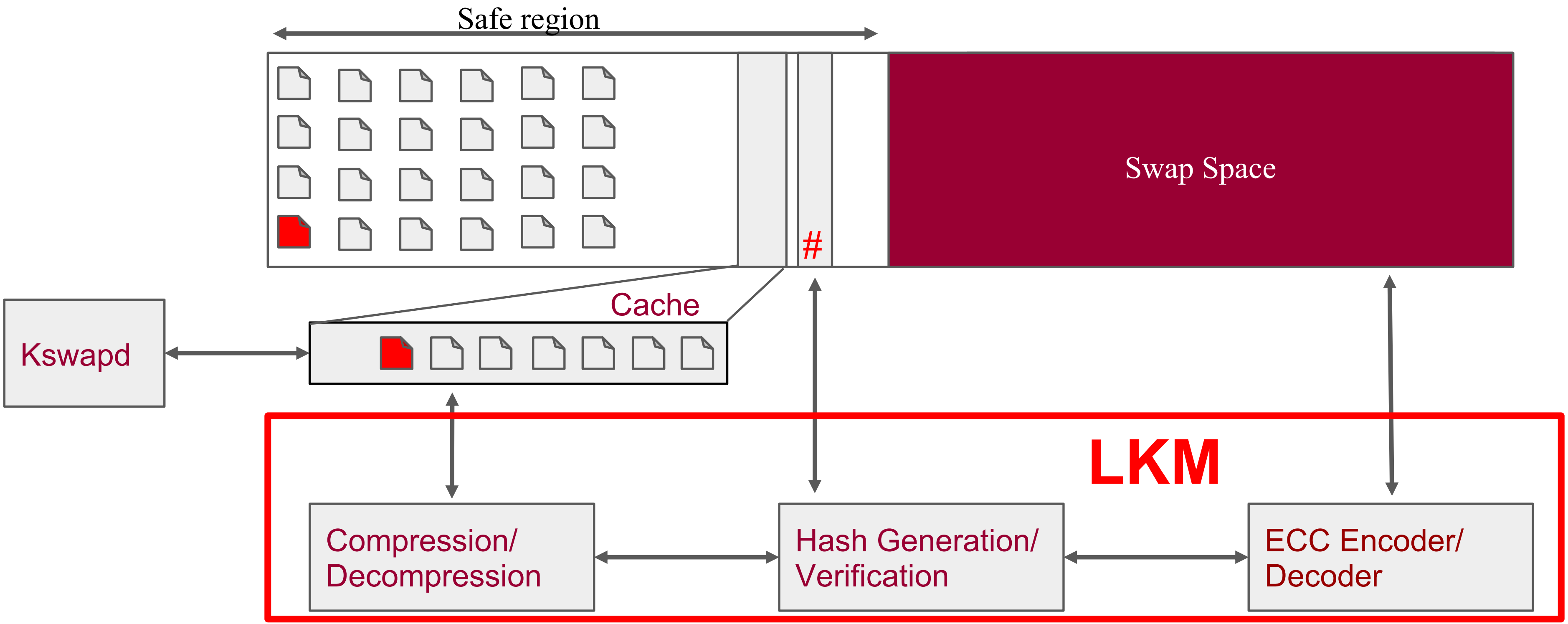
Life of a page in ZebRAM world



Life of a page in ZebRAM world



Implementation



Evaluation setup

- Haswell i7-4790 machine
- Qemu-KVM hypervisor to run ZebRAM protected OS
- Ubuntu 16.04 64-bit OS
- 100Gbit/s link

Evaluation setup

- Haswell i7-4790 machine
- Qemu-KVM hypervisor to run ZebRAM protected OS
- Ubuntu 16.04 64-bit OS
- 100Gbit/s link

- SECURITY
- SPEC
- APACHE
- NGINX
- Micro benchmarks
- REDIS

Evaluation setup

- Haswell i7-4790 machine
- Qemu-KVM hypervisor to run ZebRAM protected OS
- Ubuntu 16.04 64-bit OS
- 100Gbit/s link

- **SECURITY**
- **SPEC**
- APACHE
- NGINX
- Micro benchmarks
- **REDIS**

Security Evaluation

We ran the Rowhammer exploit on the ZebRAM protected OS

Run no.	1 bit flip in 64 bits	2 bit flips in 64 bits	Total bit flips	ZebRAM detection performance Detected bit flips	ZebRAM detection performance Corrected bit flips
1	4,698	2	4,702	4,702	4,698
2	5,132	0	5,132	5,132	5,132
3	2,790	0	2,790	2,790	2,790
4	4,216	1	4,218	4,218	4,216
5	3,554	0	3,554	3,554	3,554

Security Evaluation

We ran the Rowhammer exploit on the ZebRAM protected OS

Run no.	1 bit flip in 64 bits	2 bit flips in 64 bits	Total bit flips	ZebRAM detection performance Detected bit flips	Corrected bit flips
1	4,698	2	4,702	4,702	4,698
2	5,132	0	5,132	5,132	5,132
3	2,790	0	2,790	2,790	2,790
4	4,216	1	4,218	4,218	4,216
5	3,554	0	3,554	3,554	3,554

Security Evaluation

We ran the Rowhammer exploit on the ZebRAM protected OS

Run no.	1 bit flip in 64 bits	2 bit flips in 64 bits	Total bit flips	ZebRAM detection performance Detected bit flips	Corrected bit flips
1	4,698	2	4,702	= 4,702 100%	4,698
2	5,132	0	5,132	5,132	5,132
3	2,790	0	2,790	2,790	2,790
4	4,216	1	4,218	4,218	4,216
5	3,554	0	3,554	3,554	3,554

Security Evaluation

We ran the Rowhammer exploit on the ZebRAM protected OS

Run no.	1 bit flip in 64 bits	2 bit flips in 64 bits	Total bit flips	ZebRAM detection performance Detected bit flips	Corrected bit flips	
1	4,698	2	4,702	4,702	4,698	99.9%
2	5,132	0	5,132	5,132	5,132	
3	2,790	0	2,790	2,790	2,790	
4	4,216	1	4,218	4,218	4,216	
5	3,554	0	3,554	3,554	3,554	

Security Evaluation

We ran the Rowhammer exploit on the ZebRAM protected OS

Run no.	1 bit flip in 64 bits	2 bit flips in 64 bits	Total bit flips	ZebRAM detection performance Detected bit flips	Corrected bit flips
1	4,698	2	4,702	4,702	4,698
2	5,132	0	5,132	5,132	5,132
3	2,790	0	2,790	2,790	2,790
4	4,216	1	4,218	4,218	4,216
5	3,554	0	3,554	3,554	3,554

Take away:

- ECC module alone **detected 100%** the bit flips
- ECC module **corrected 99.97 %** of the bit flips

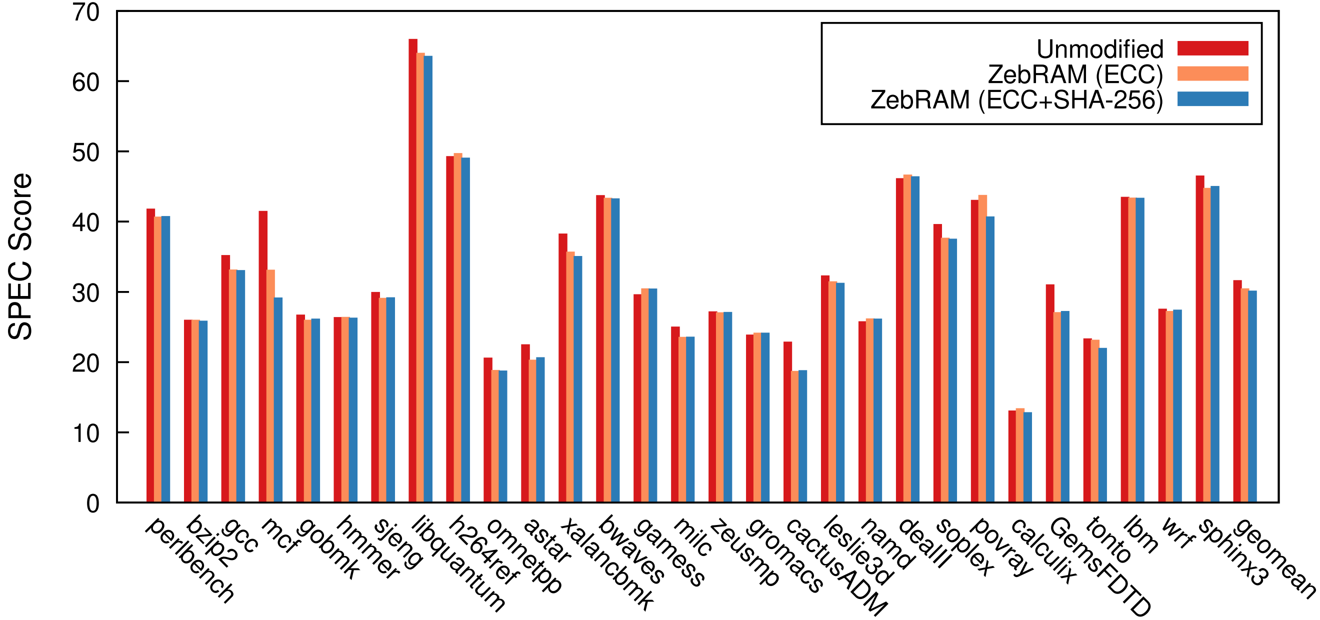
Performance Evaluation

We ran spec 2006 on three different setup:

- Baseline (unmodified Linux) with 4GB memory
- ZebRAM (ECC only)
- ZebRAM (ECC + SHA-256)

Performance Evaluation

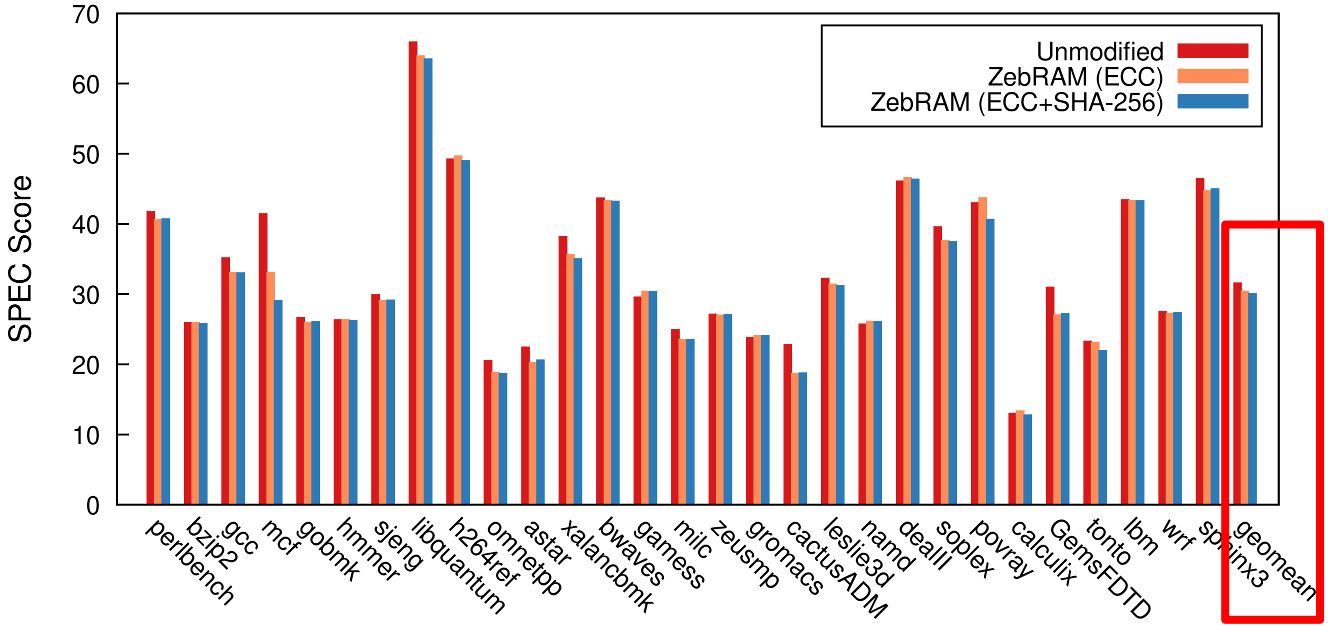
Spec 2006 benchmark shows ...



Performance Evaluation

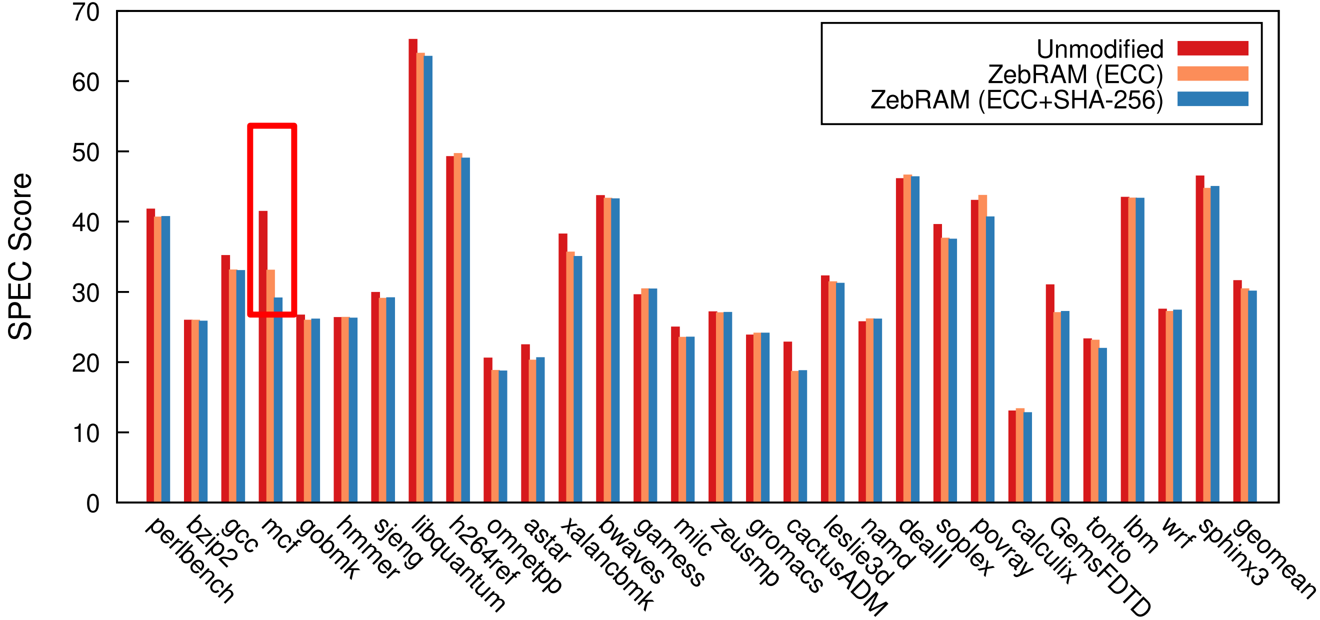
Spec 2006 benchmark shows …

… 5% (geometric mean) overhead from unavailability of transparent huge page



Performance Evaluation

MCF benchmark shows more than 5% performance overhead



Performance Evaluation : Working Set Size

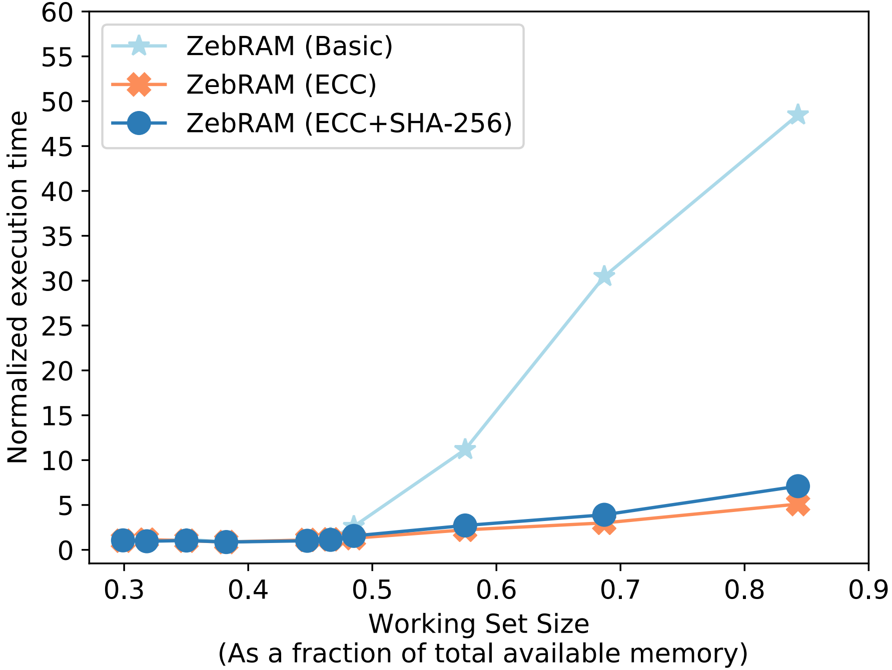
YCSB to generate the load and induce different working set size ...

… for **redis** (4.0.8) key-value store

We ran experiments on different setups:

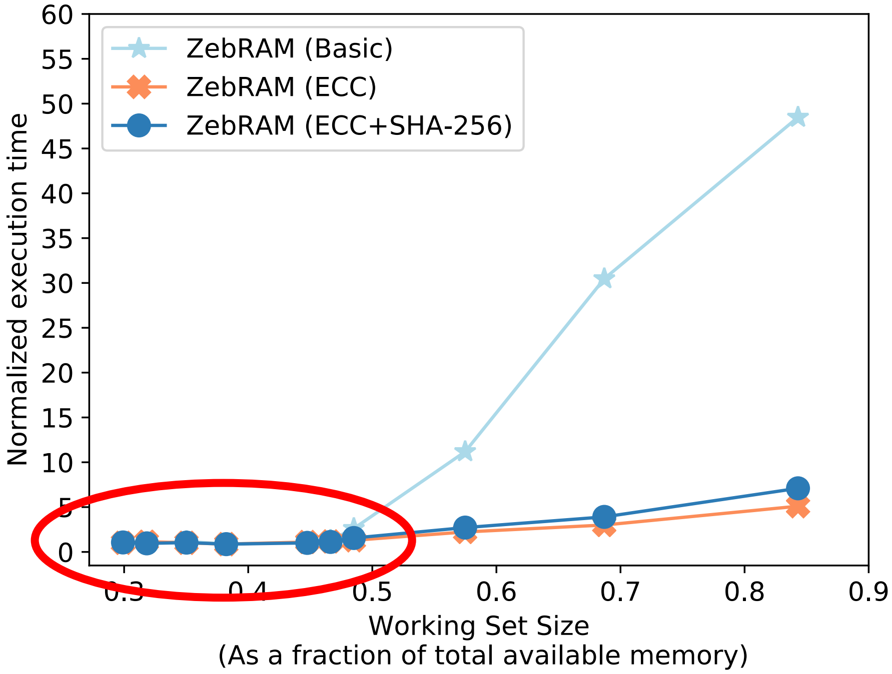
- ZebRAM Basic – uses only safe region and swaps out to SSD
- ZebRAM (ECC only)
- ZebRAM (ECC + SHA-256)
- Baseline

Performance Evaluation



Performance Evaluation

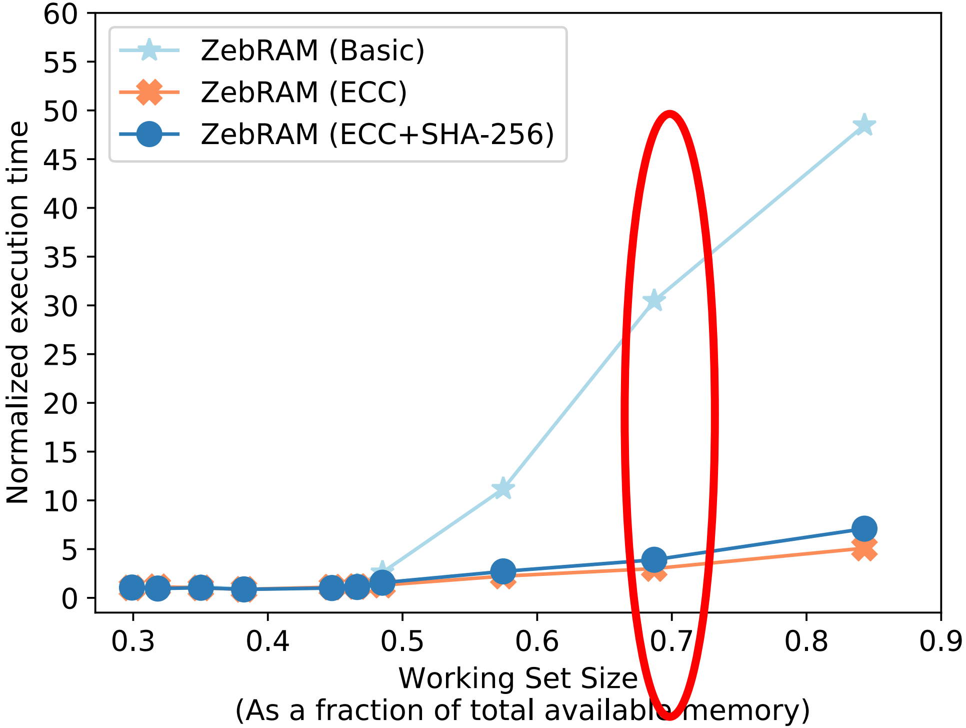
1.05x performance overhead till it starts using swap



Performance Evaluation

When active working set is using 70% of the memory:

- ZebRAM (Basic) = 30x
- ZebRAM (ECC) = 3x
- ZebRAM (ECC + SHA-256) = 3.9x



Summary

- The ZebRAM is the first solution to provide complete protection against Rowhammer attacks
- Performance overhead:
  - Minimal when the active working set fits in the safe region
  - Function of the active working set size when it does not fit in the safe region
- Code for ZebRAM will be available soon at https://github.com/vusec