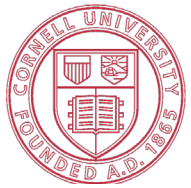
SoftVN: Efficient Memory Protection via Software-Provided Version Numbers

Muhammad Umar[1], Weizhe Hua[1], Zhiru Zhang[1], G. Edward Suh[1,2]
Cornell University[1]; Meta AI[2]

ISCA 2022 Session 3A: Security II
Monday, June 20, 2022



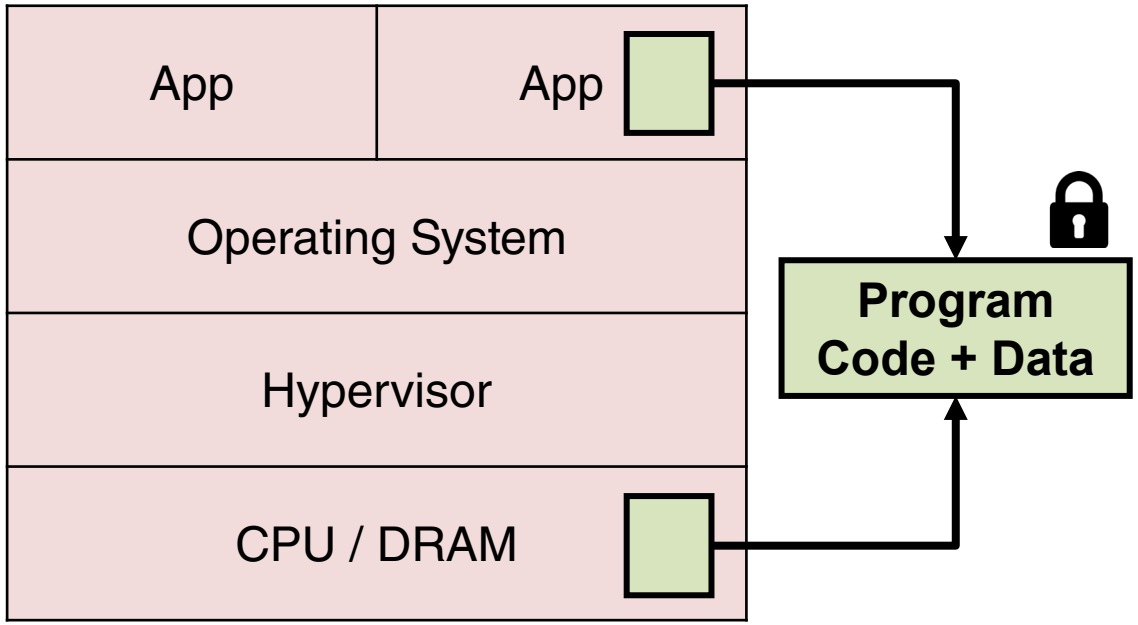
Cornell University

Meta AI

CSL

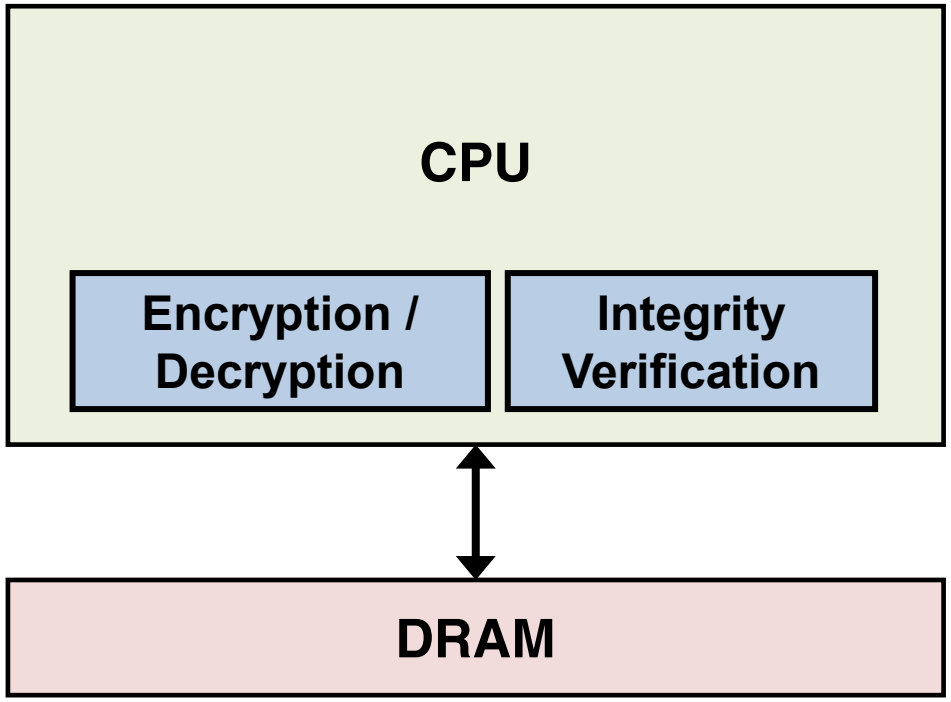
TEEs (Trusted Execution Environments)

- Secure hardware compartments that provide strong protection for confidentiality and integrity of code and data, even from low-level software and physical tampering
- Also called ‘enclaves’ or ‘secure processors’
- Examples: Intel SGX, AMD SEV, ARM TrustZone, RISC-V KeyStone



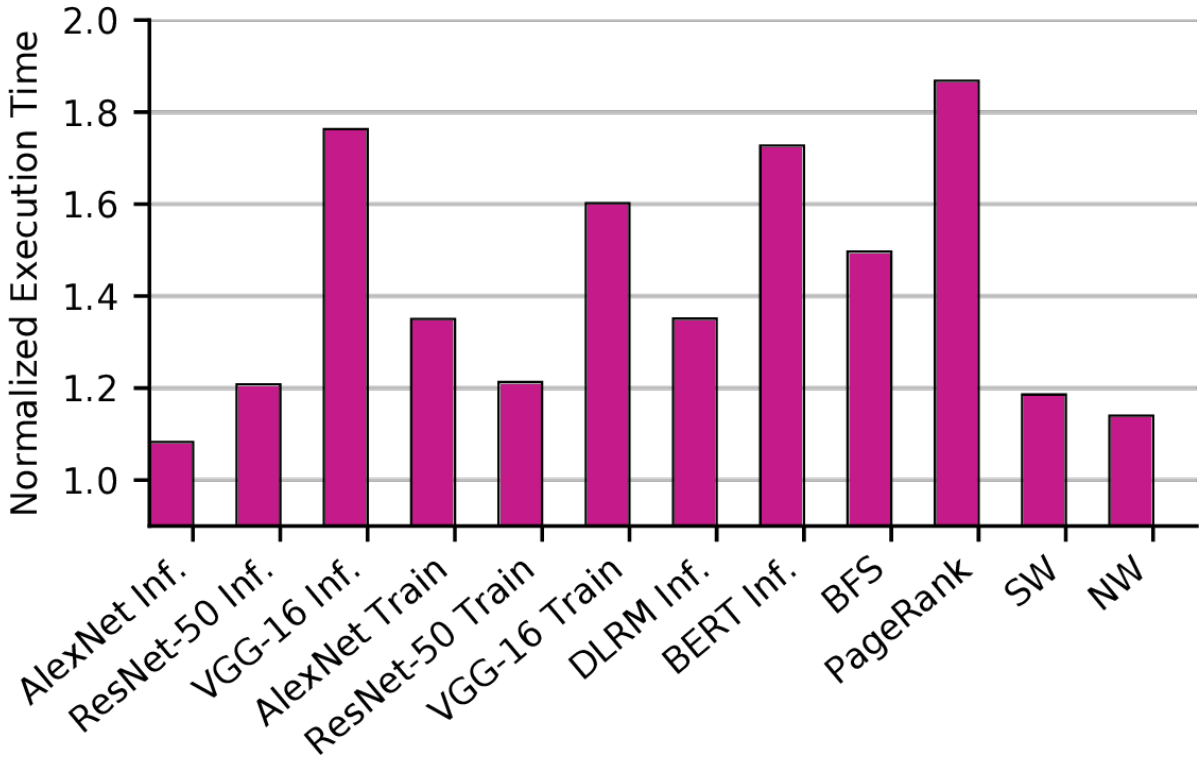
Problem: Memory Protection Overhead

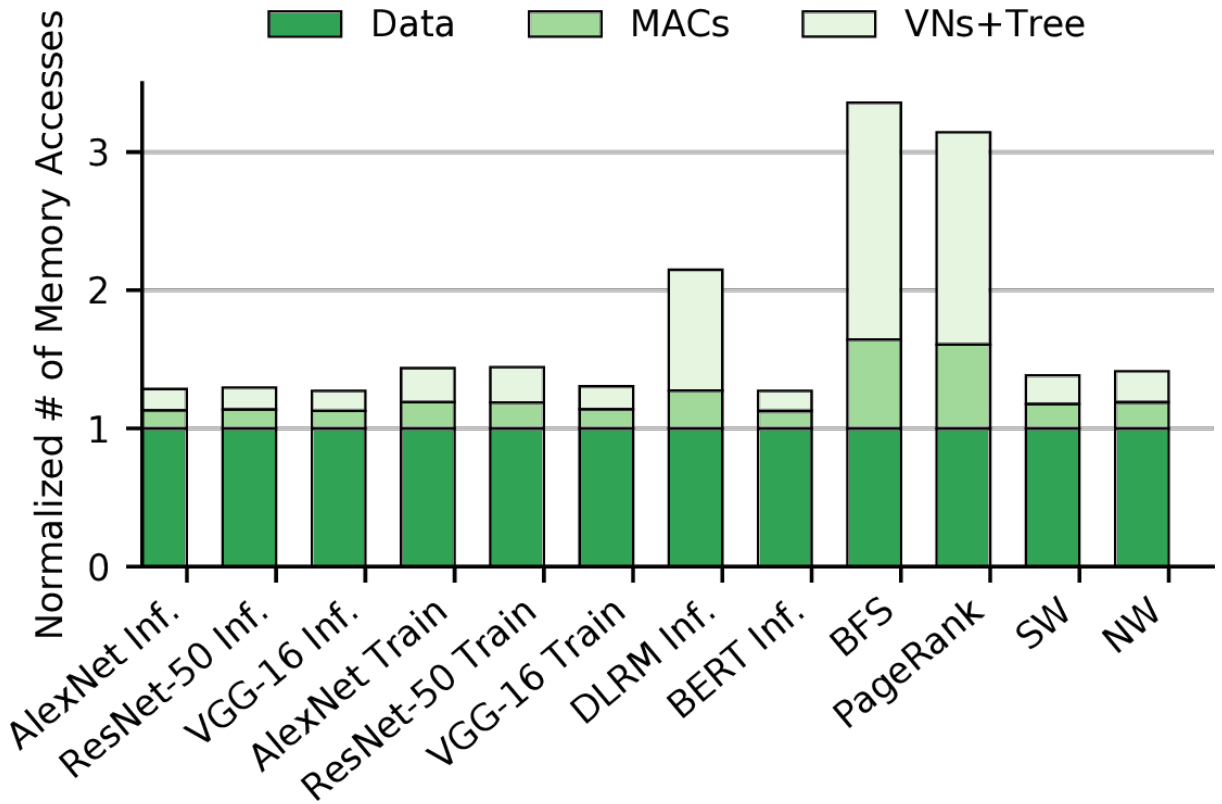
- Off-chip memory protection in TEEs
  - For **confidentiality**, need to encrypt data (cache blocks) in memory
  - For **integrity**, need to check if a memory read returns most recent value that is stored at the address by a processor
- Main source of overhead in TEEs
  - Especially for **memory-intensive workloads**



Memory Protection Overhead for Memory-Intensive Workloads

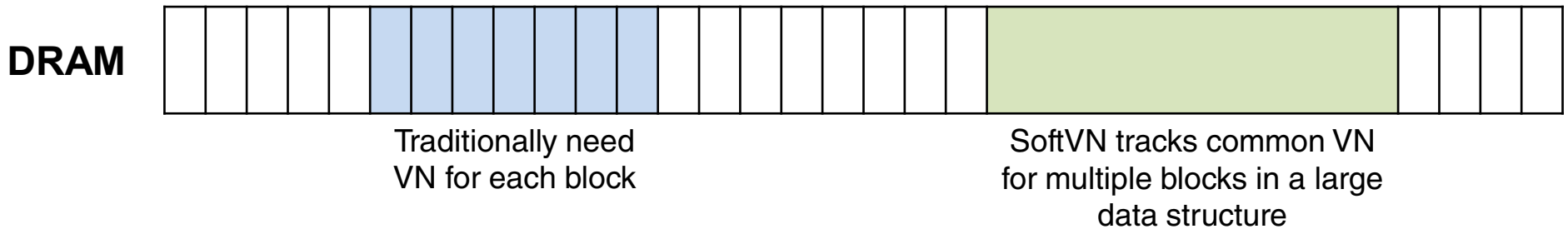
- Significant performance overhead and memory traffic increase for memory-intensive workloads





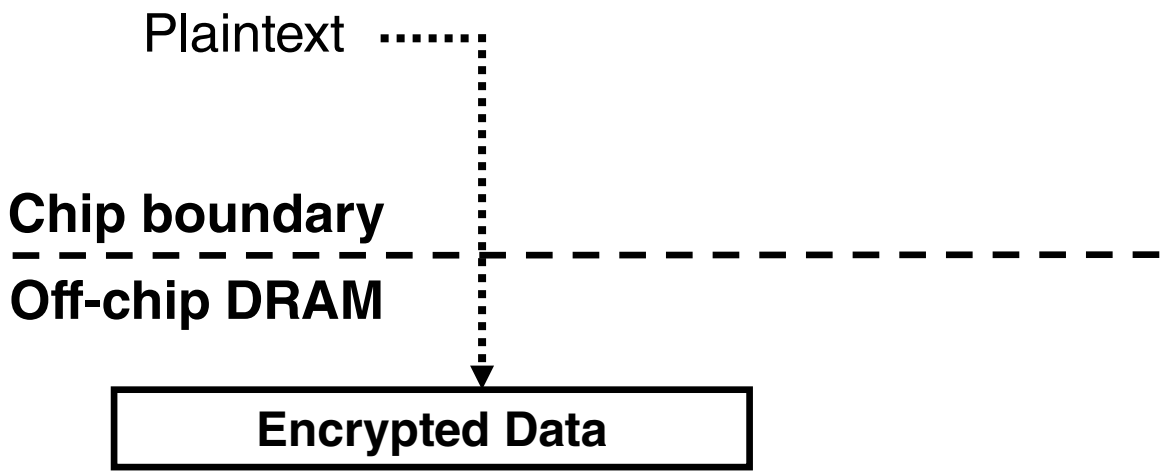
SoftVN Memory Protection - Overview

- Exploit application-specific characteristics
  - Uniform write pattern to many elements in a data structure
- Intuition: common version number (VN) for all elements in a data structure
- Track VN metadata in software
  - Software provides VN for reads' decryption (instead of off-chip)
- Enable encryption + integrity verification with low overhead



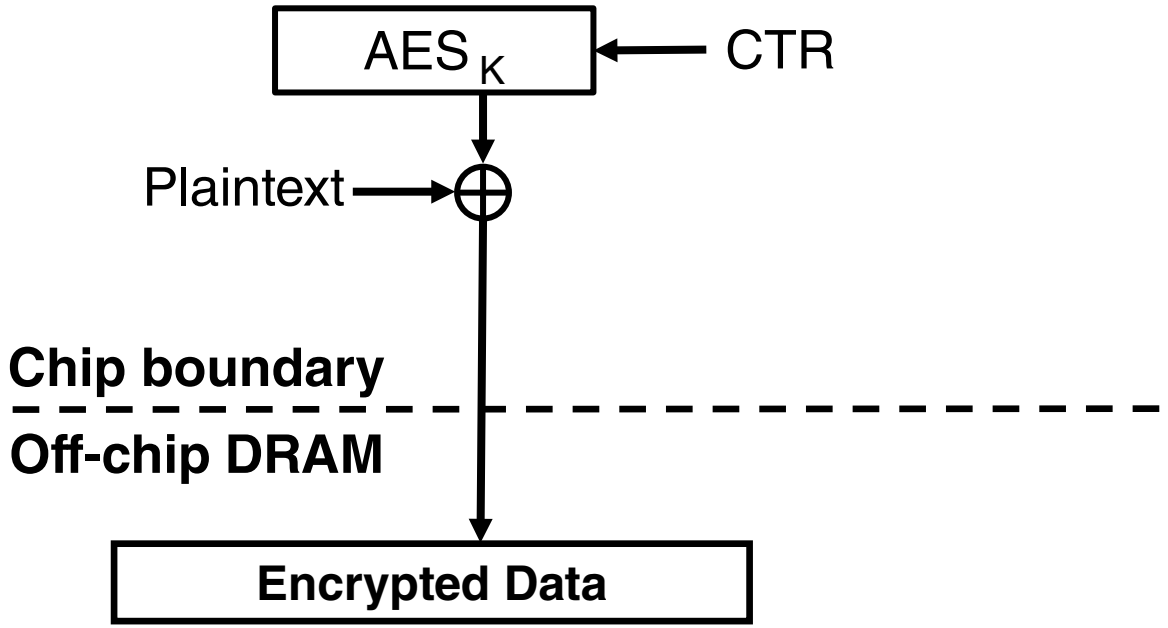
Today's Off-Chip Memory Protection

- **Confidentiality**: Encrypted data



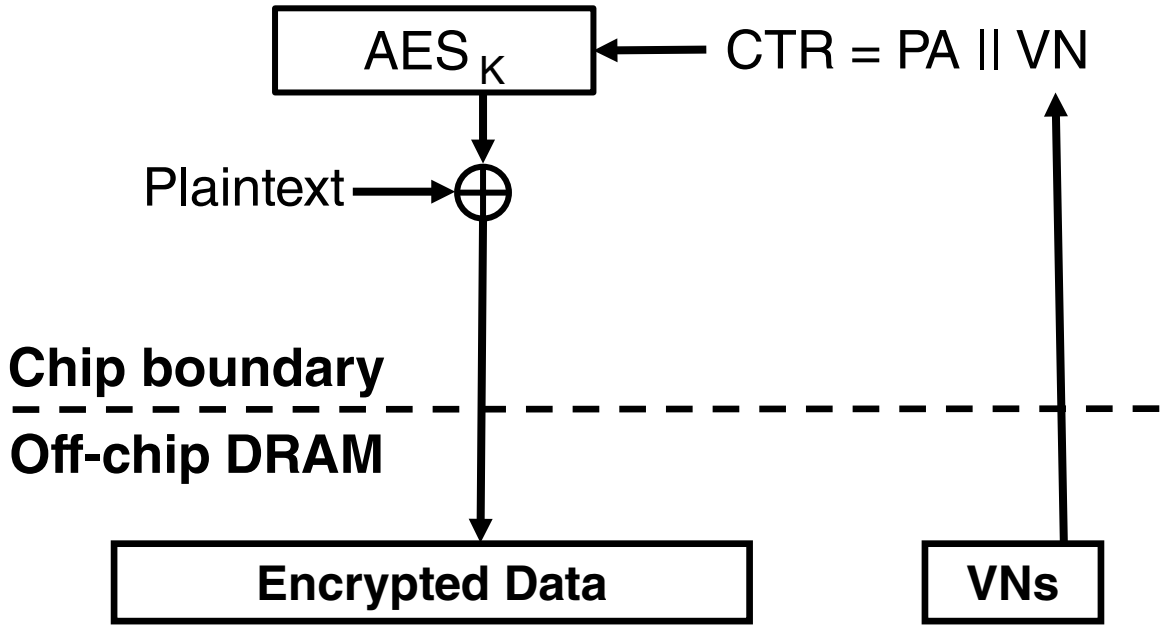
Today's Off-Chip Memory Protection

- **Confidentiality**: Encrypted data using AES
  - CTR mode hides the encryption/decryption latency, CTR unique spatially and temporally



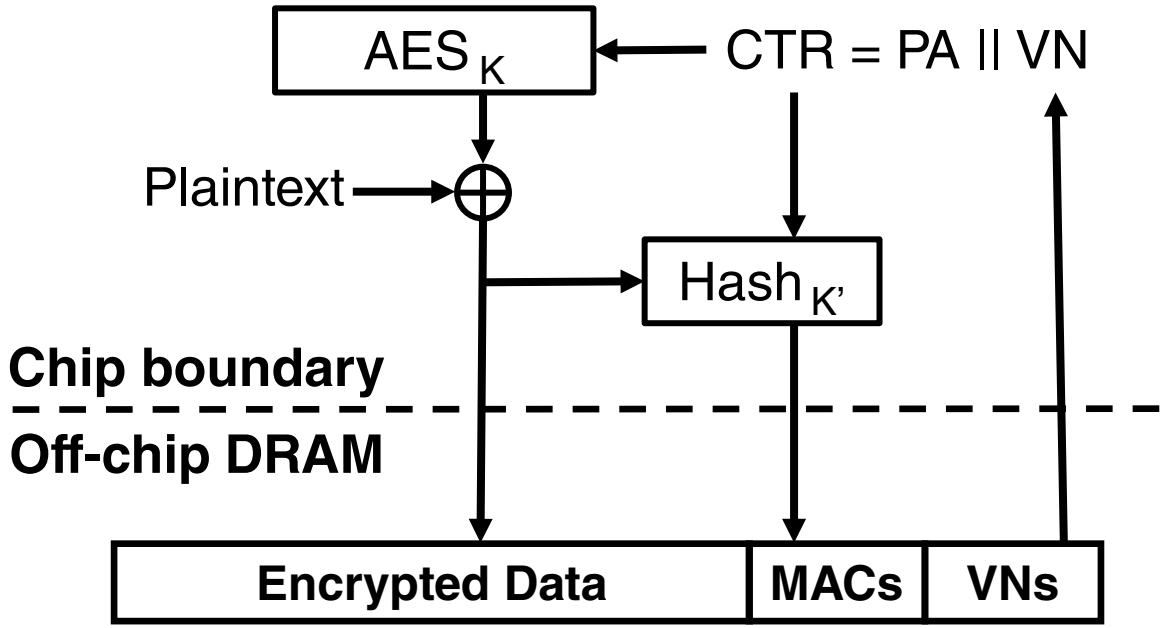
Today's Off-Chip Memory Protection

- **Confidentiality**: Encrypted data using AES
  - CTR mode hides the encryption/decryption latency, CTR unique spatially and temporally
  - Per cache-block **version numbers** (VNs) are stored in the off-chip memory
  - PA: physical address



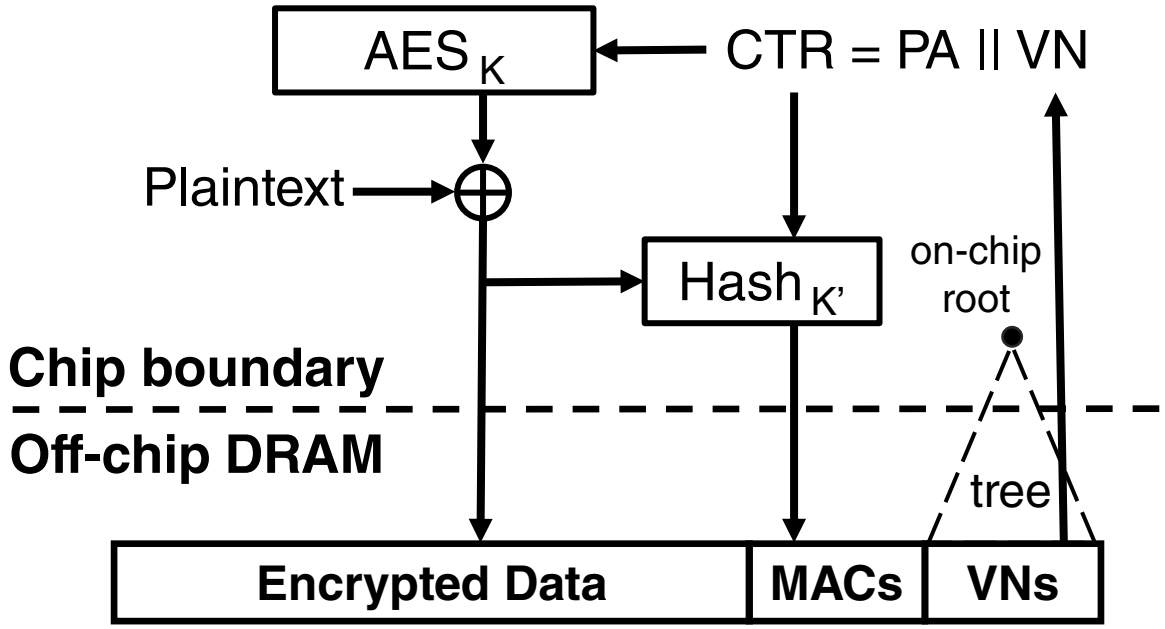
Today's Off-Chip Memory Protection

- **Confidentiality**: Encrypted data using AES
  - CTR mode hides the encryption/decryption latency, CTR unique spatially and temporally
  - Per cache-block **version numbers** (VNs) are stored in the off-chip memory
  - PA: physical address
- **Integrity**: verify using per-block Message Authentication Codes (MACs)
  - MACs using keyed hashes, for example



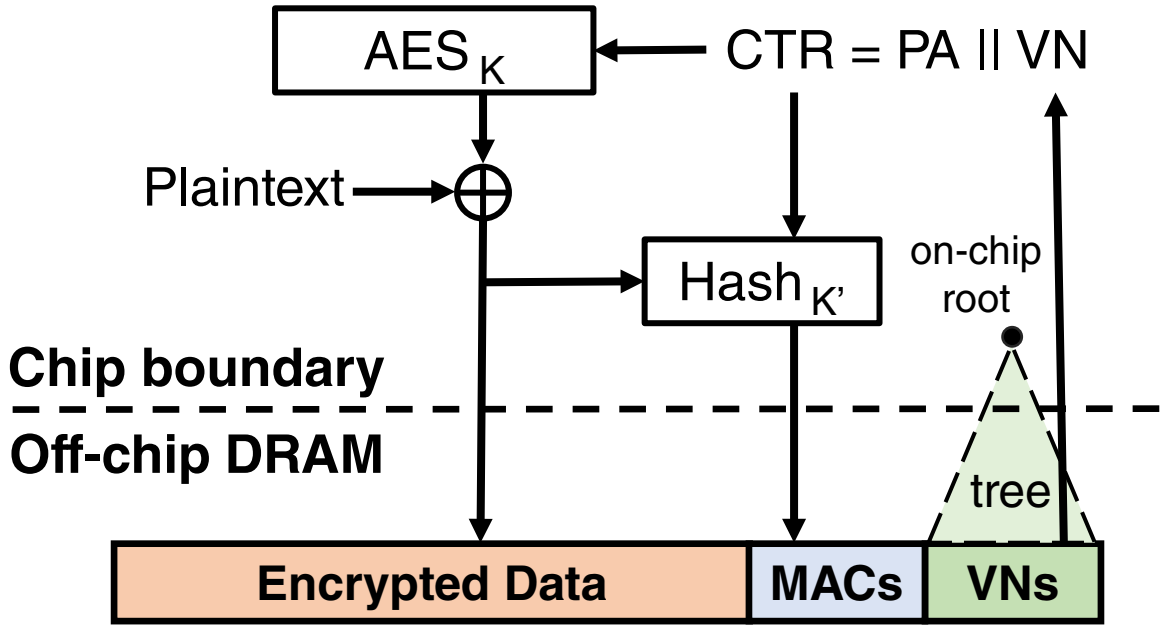
Today's Off-Chip Memory Protection

- **Confidentiality**: Encrypted data using AES
  - CTR mode hides the encryption/decryption latency, CTR unique spatially and temporally
  - Per cache-block **version numbers** (VNs) are stored in the off-chip memory
  - PA: physical address
- **Integrity**: verify using per-block Message Authentication Codes (MACs)
  - MACs using keyed hashes, for example
  - For replay protection, need integrity tree over VNs, with root on-chip



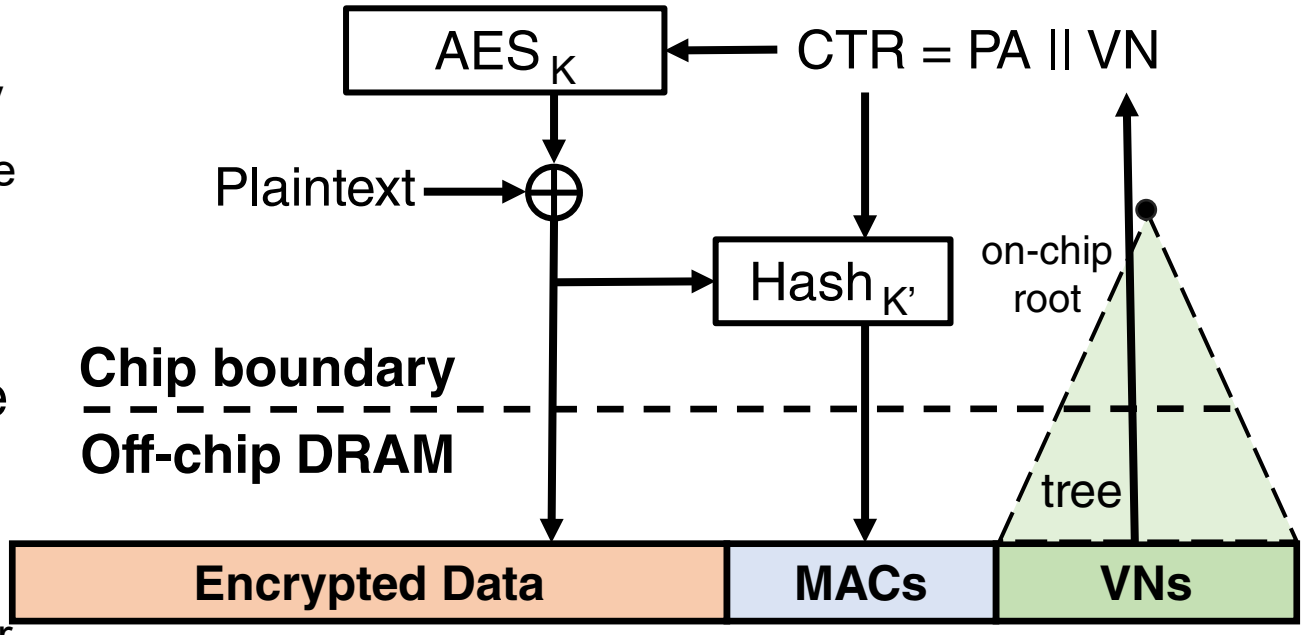
Today's Off-Chip Memory Protection

- **Confidentiality**: Encrypted data using AES
  - CTR mode hides the encryption/decryption latency, CTR unique spatially and temporally
  - Per cache-block **version numbers** (VNs) are stored in the off-chip memory
  - PA: physical address
- **Integrity**: verify using per-block Message Authentication Codes (MACs)
  - MACs using keyed hashes, for example
  - For replay protection, need integrity tree over VNs, with root on-chip
- Incurs **high performance overhead**



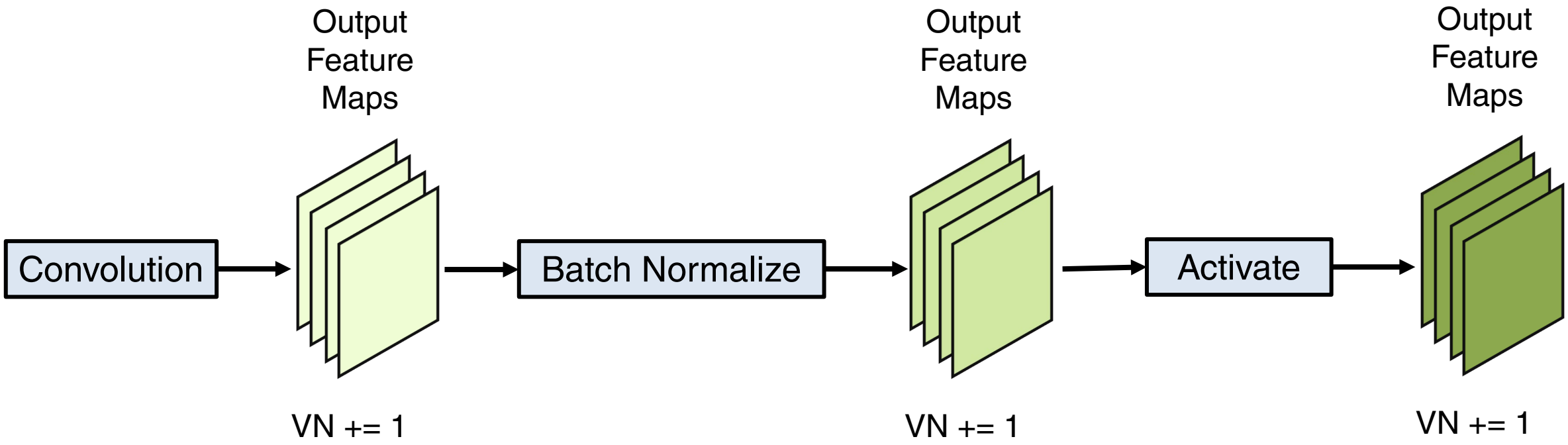
Today's Off-Chip Memory Protection

- **Confidentiality**: Encrypted data using AES
  - CTR mode hides the encryption/decryption latency, CTR unique spatially and temporally
  - Per cache-block **version numbers** (VNs) are stored in the off-chip memory
  - PA: physical address
- **Integrity**: verify using per-block Message Authentication Codes (MACs)
  - MACs using keyed hashes, for example
  - For replay protection, need integrity tree over VNs, with root on-chip
- Incurs **high performance overhead** for memory intensive applications



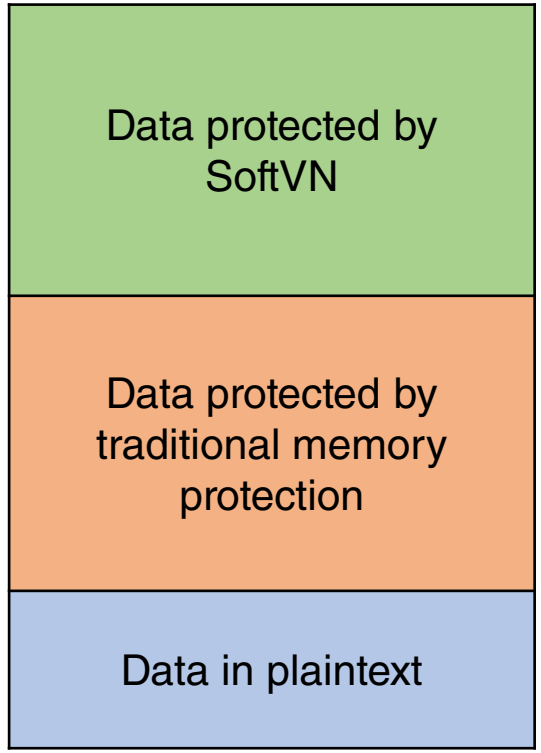
SoftVN Intuition

Example: DNN Inference



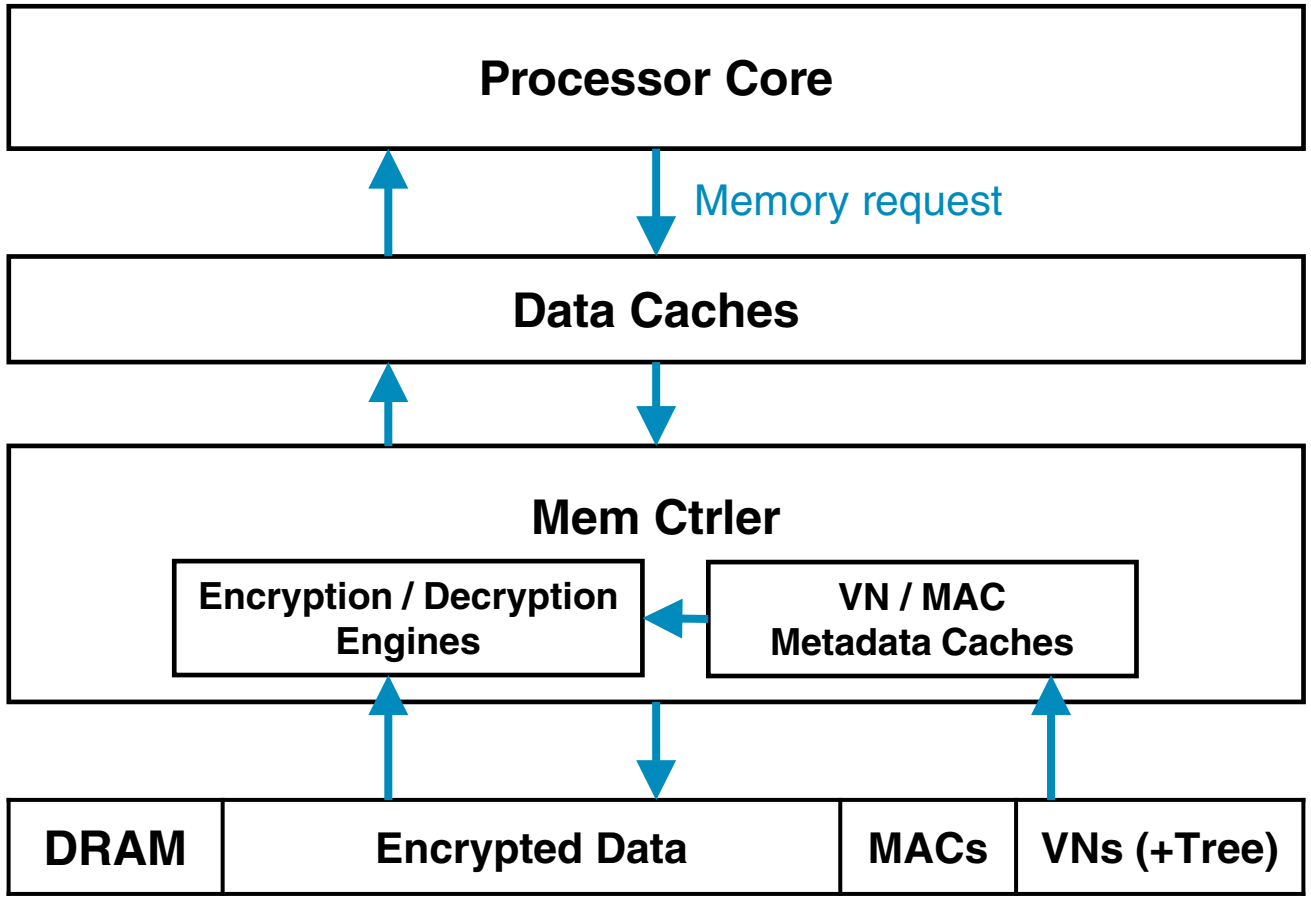
SoftVN Memory Protection

- Extends current memory protection schemes
  - New region for **large** structures only
- Allows application software to provide **VNs for memory reads**

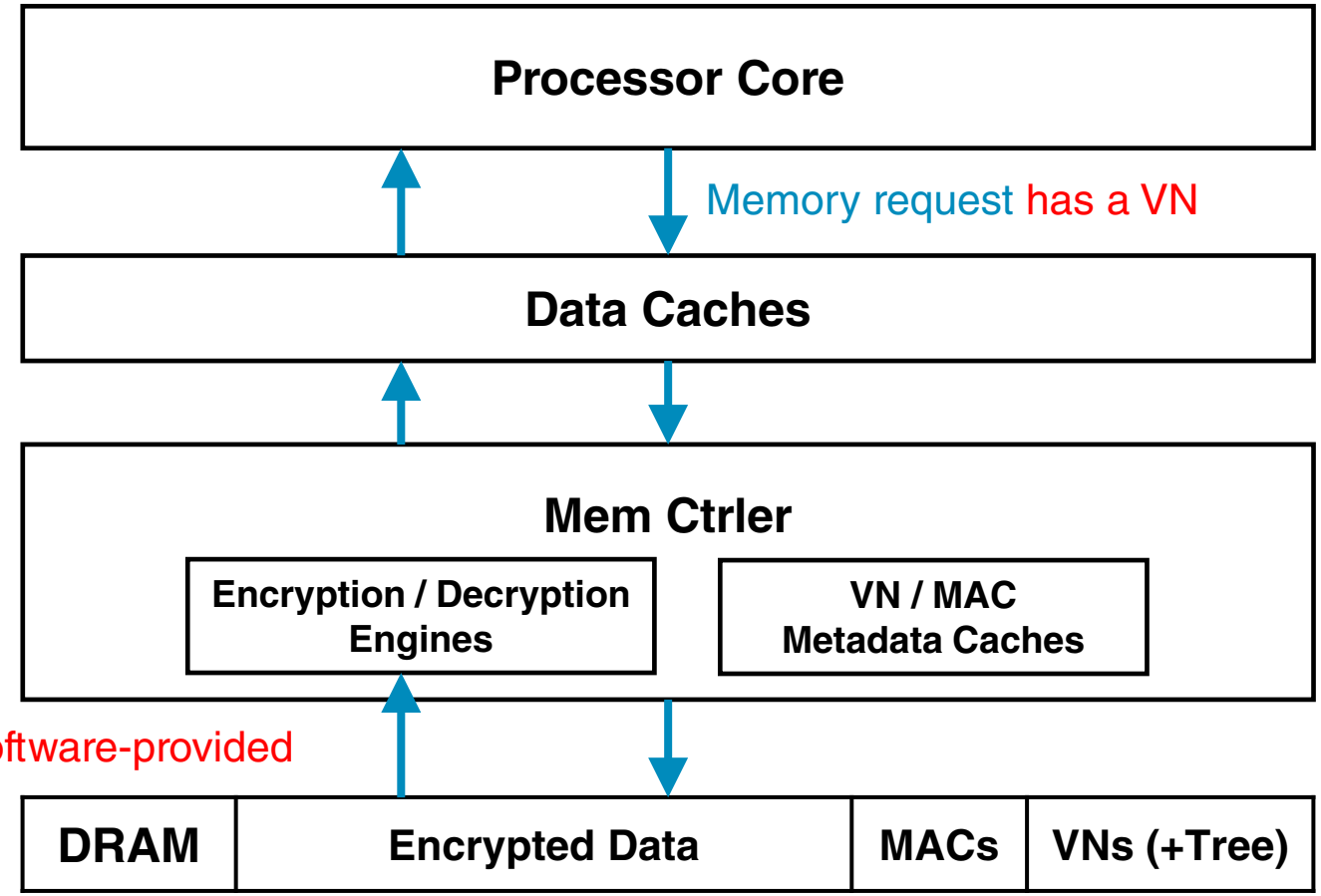


Virtual Memory Layout of an Application

Read Decryption in Traditional Memory Protection

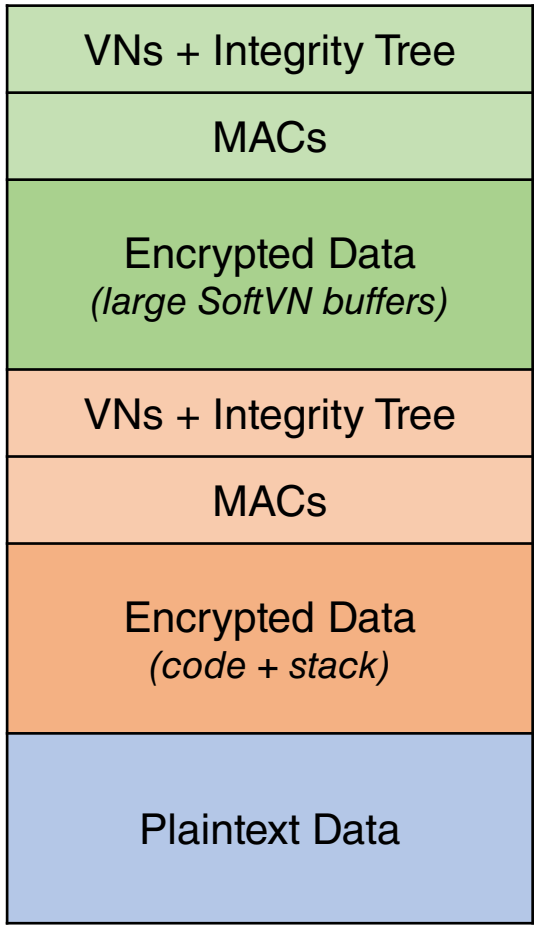


Read Decryption with Software-provided Version Numbers



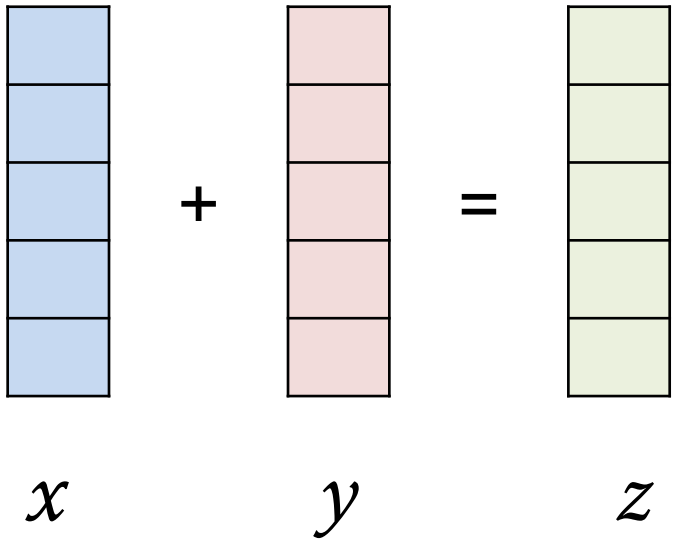
SoftVN Memory Protection

- Extends current memory protection schemes
  - New region for **large** structures only
- Allows software to provide **VNs for memory reads** to reduce **latency**
- **Still stores** software-provided **VNs off-chip** and updates them *in the background*
  - Use for transparent processor mechanisms e.g. cache block eviction, paging and prefetching.



Physical Memory Layout

Example: Vector-Vector Addition

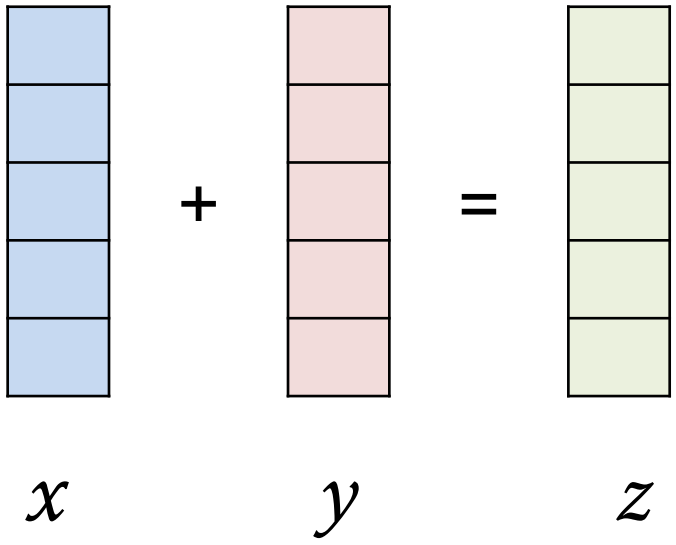


Three data buffers in the SoftVN region

Vector-Vector Addition Kernel

```
for(int i = 0; i < N; i++)
   z[i] = x[i] + y[i];
```

Challenge 1: Software Overhead to Specify VNs

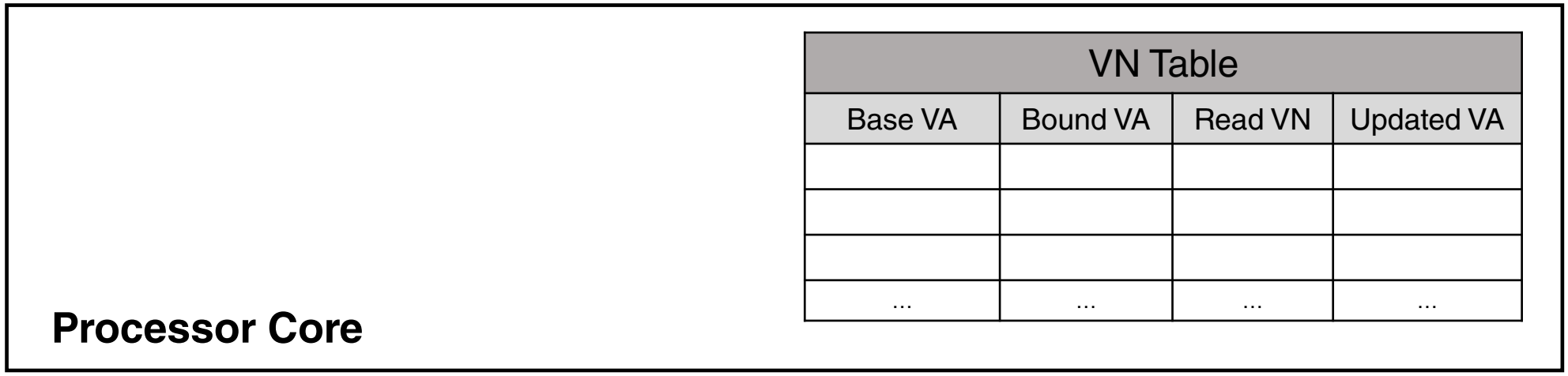


Kernel with Modifications

```
for(int i = 0; i < N; i++)
    SoftVN instructions
    z[i] = x[i] + y[i];
```

Software needs to efficiently provide VNs for each memory request in the SoftVN region

Solution: VN Table



HW Mechanism – Set VN Table

`SoftVN_SetVN (Table_ID, VA, Length, Read_VN)`

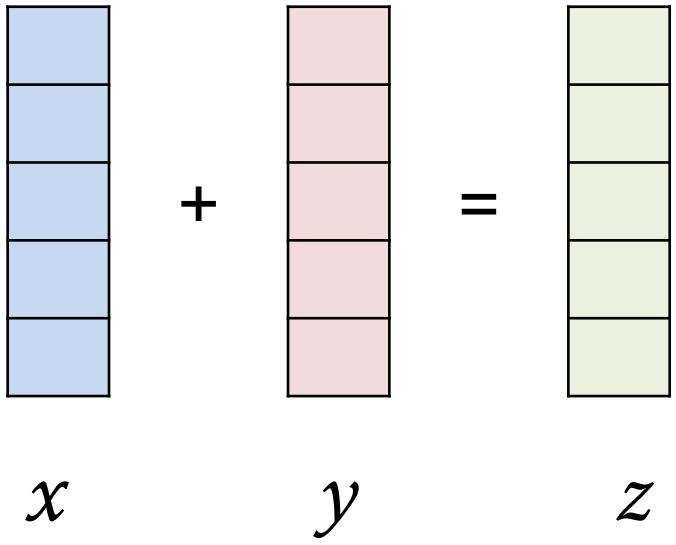
Processor Core

VN Table				
Base VA	Bound VA	Read VN	Updated VA	
0xaaa	0xaff	2	0	*x*
0xbbb	0xbff	2	0	*y*
0xccc	0xcff	1	0	*z*
...	...	...	...	

Set all data buffers entries in VN table

`SoftVN_InvalidateVN (Table_ID)`

Example: Vector-Vector Addition



Three SoftVN region data buffers

Kernel with Modifications

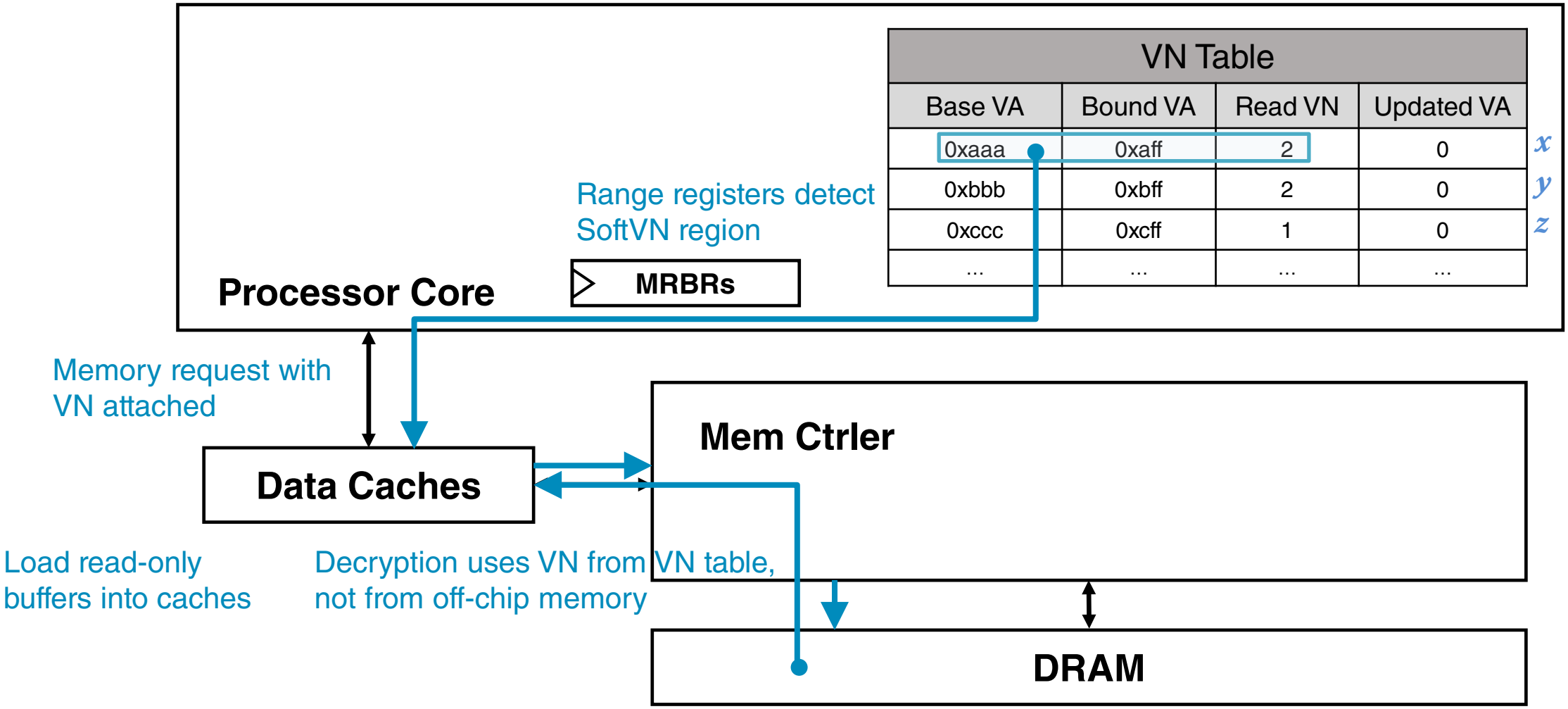
```
Set VN Table (x)
Set VN Table (y)
Set VN Table (z)
```

```
for(int i = 0; i < N; i++)
   z[i] = x[i] + y[i];
```

```
Invalidate Table (x)
Invalidate Table (y)
Invalidate Table (z)
```

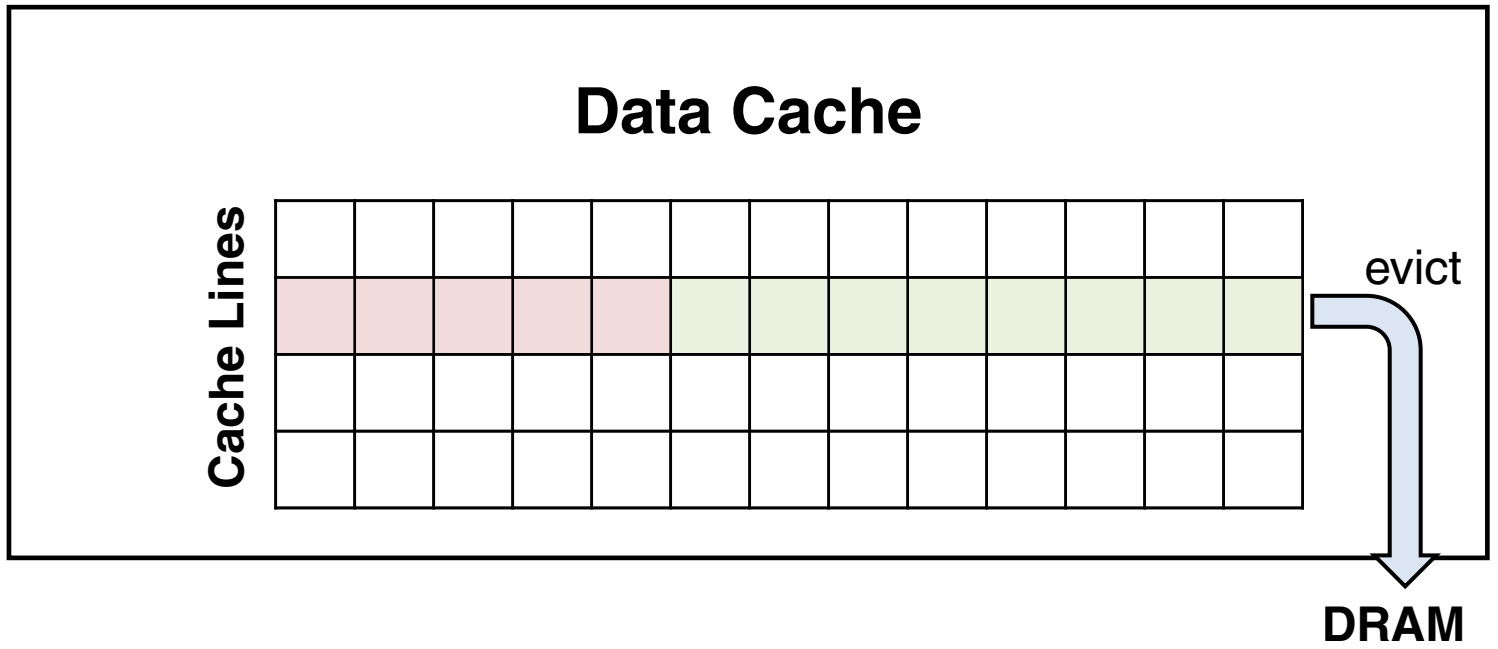
HW Mechanism – Loading SoftVN buffers

```
for(int i = 0; i < N; i++)
    z[i] = x[i] + y[i];
```

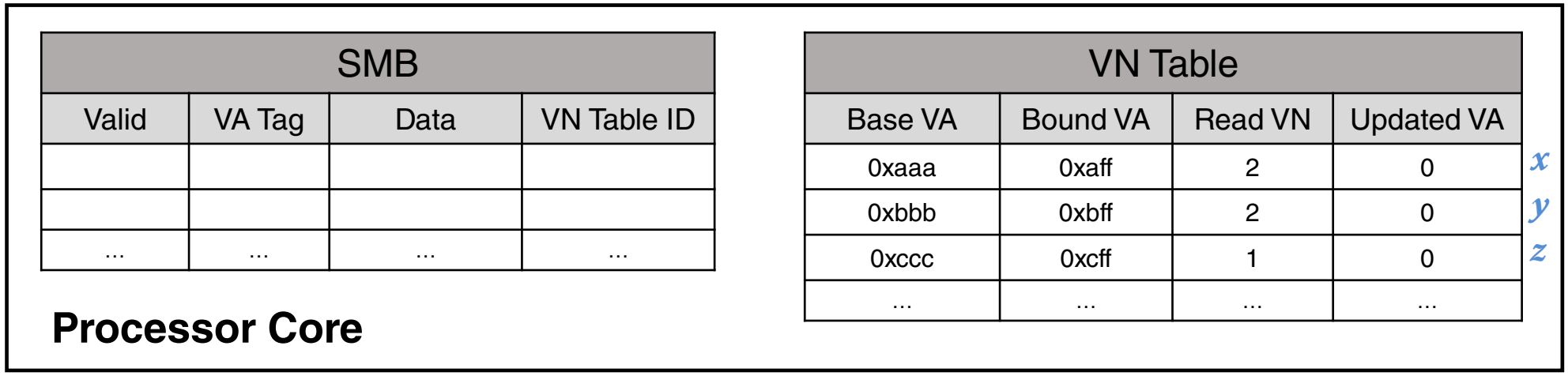


Challenge 2: Granularity Mismatch

- Load/store instructions write a word at a time
- Off-chip accesses are per cache line granularity e.g. to evict and writeback cache lines
  - VNs are per cache line

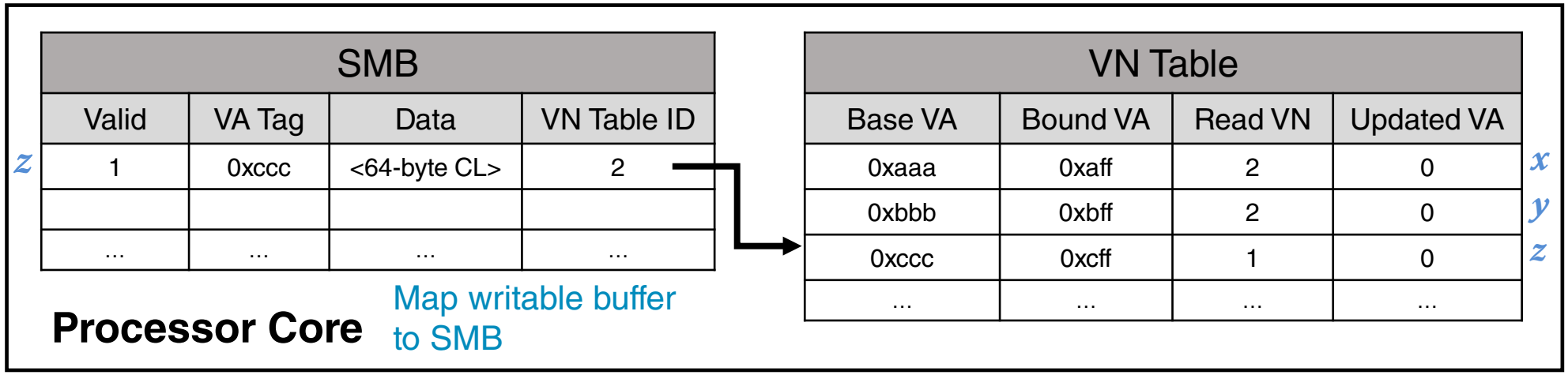


Solution: Secure Memory Buffer (SMB) for Writes to SoftVN Region



HW Mechanism – Map to SMB

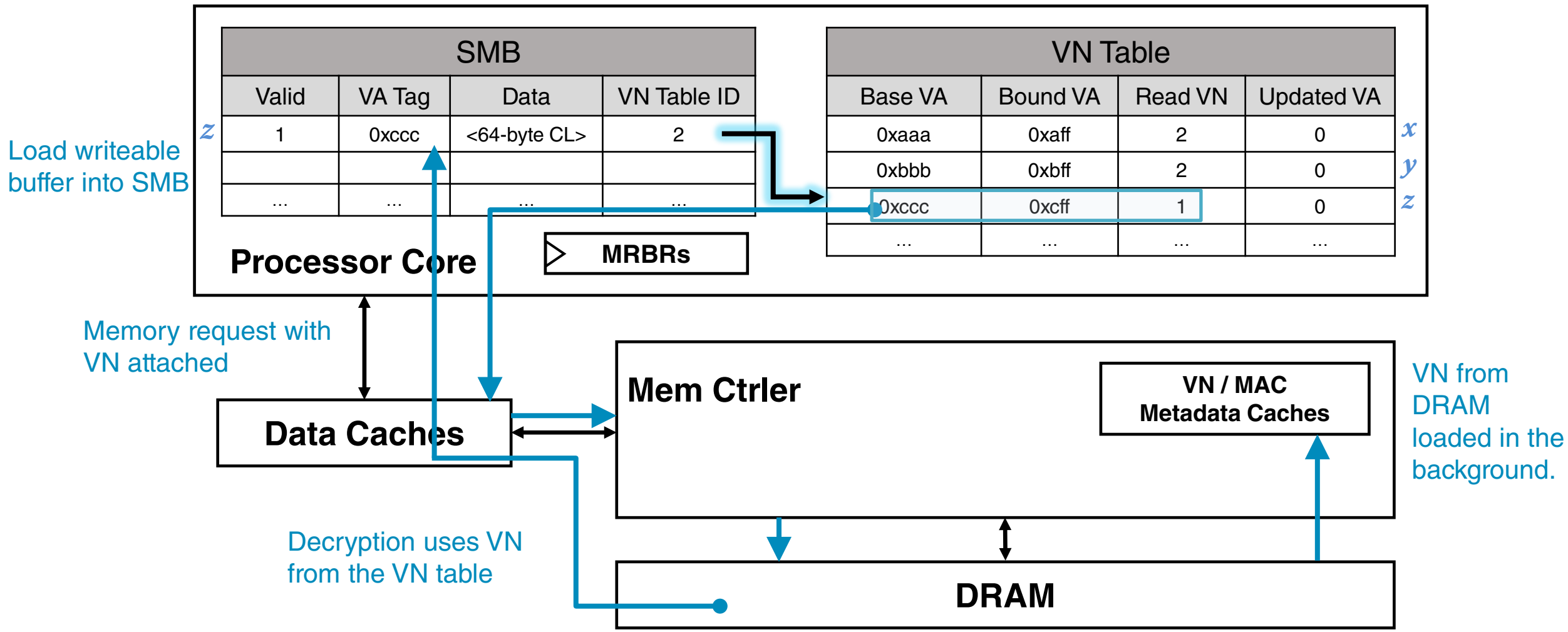
```
for(int i = 0; i < N; i++)
    z[i] = x[i] + y[i];
```




```
SoftVN_Map (SMB_Slot, Table_ID)
```

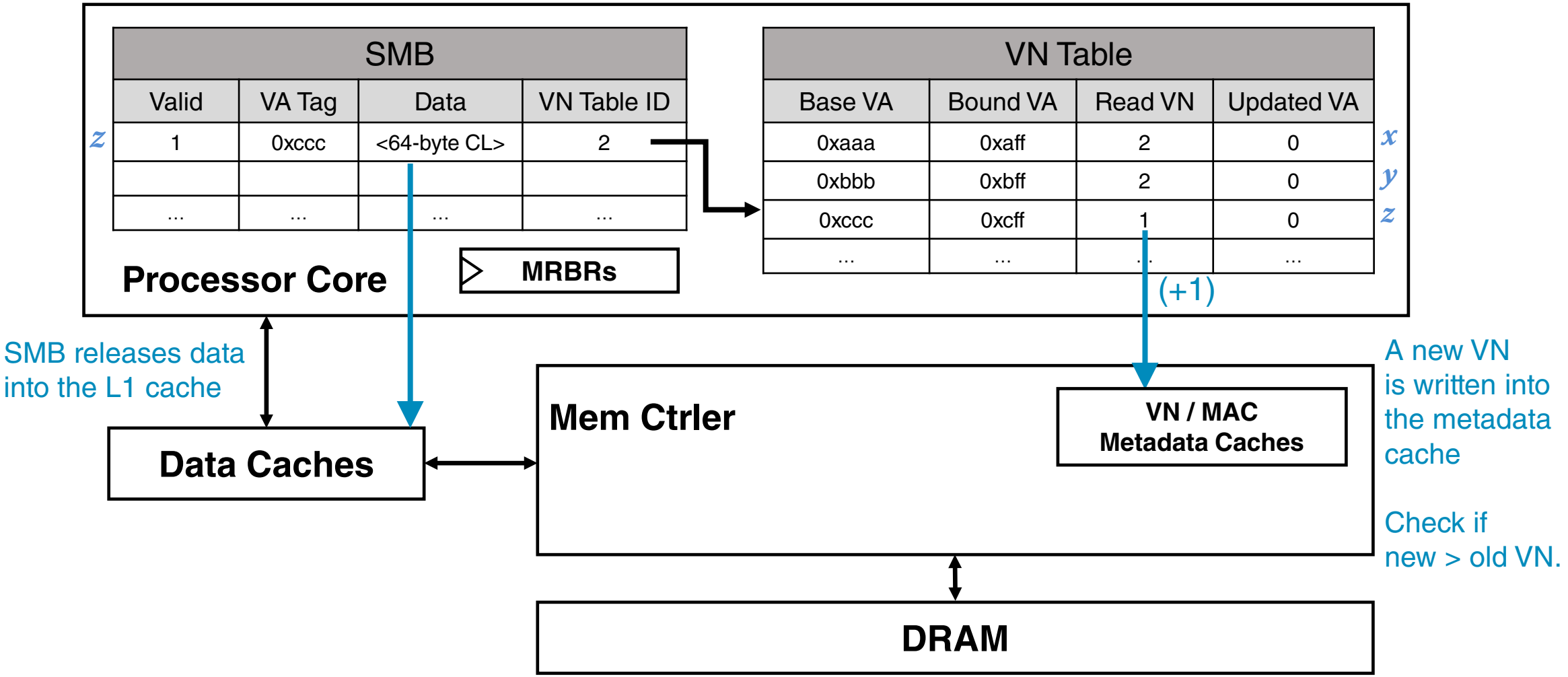
HW Mechanism – Loading to SMB

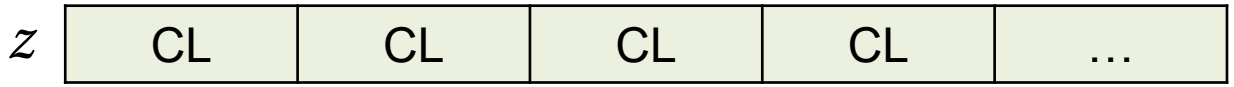
```
for(int i = 0; i < N; i++)
    z[i] = x[i] + y[i];
```



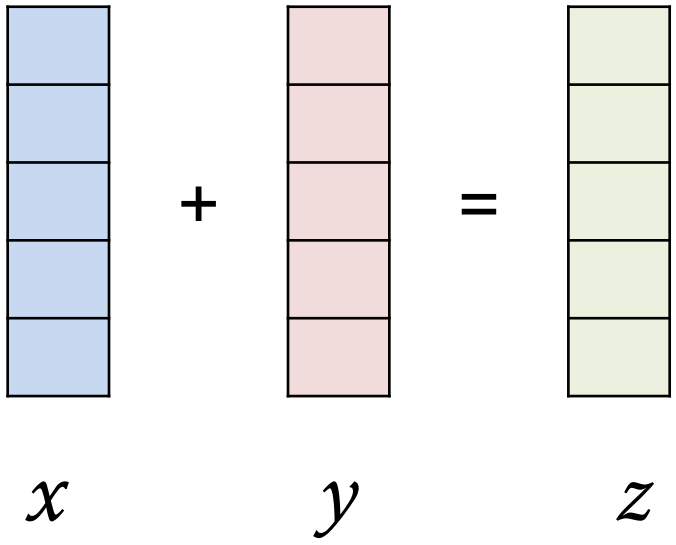
HW Mechanism – Releasing from SMB

```
for(int i = 0; i < N; i++)
    z[i] = x[i] + y[i];
```





Example: Vector-Vector Addition



Three SoftVN region data buffers

Kernel with Modifications

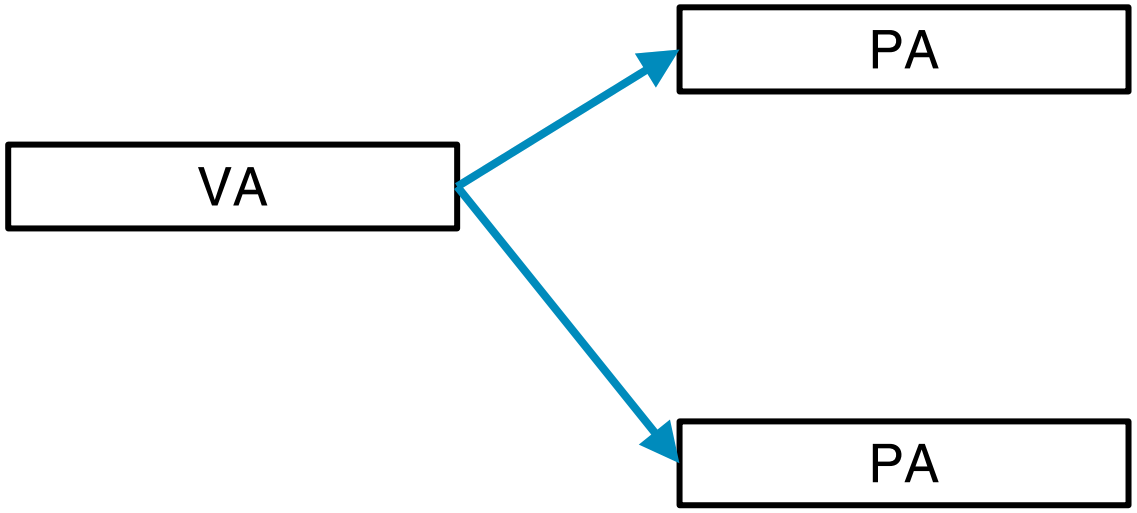
```
Set VN Table (x)
Set VN Table (y)
Set VN Table (z)
Map to SMB (z)
```

```
for(int i = 0; i < N; i++)
   z[i] = x[i] + y[i];
```

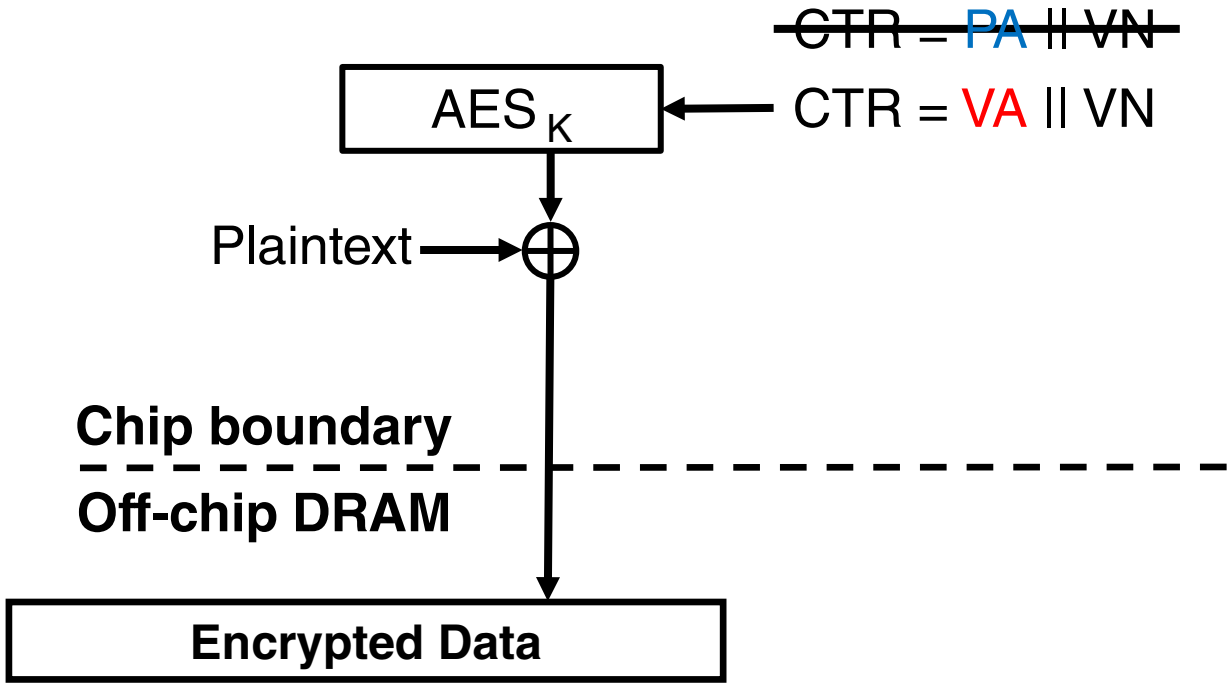
```
Invalidate Table (x)
Invalidate Table (y)
Invalidate Table (z)
```

Challenge 3: VA-PA mapping

- Virtual-to-Physical address mapping controlled by OS (as in Intel SGX)
  - Paging can relocate blocks → VN changes
  - Can no longer track VN in software

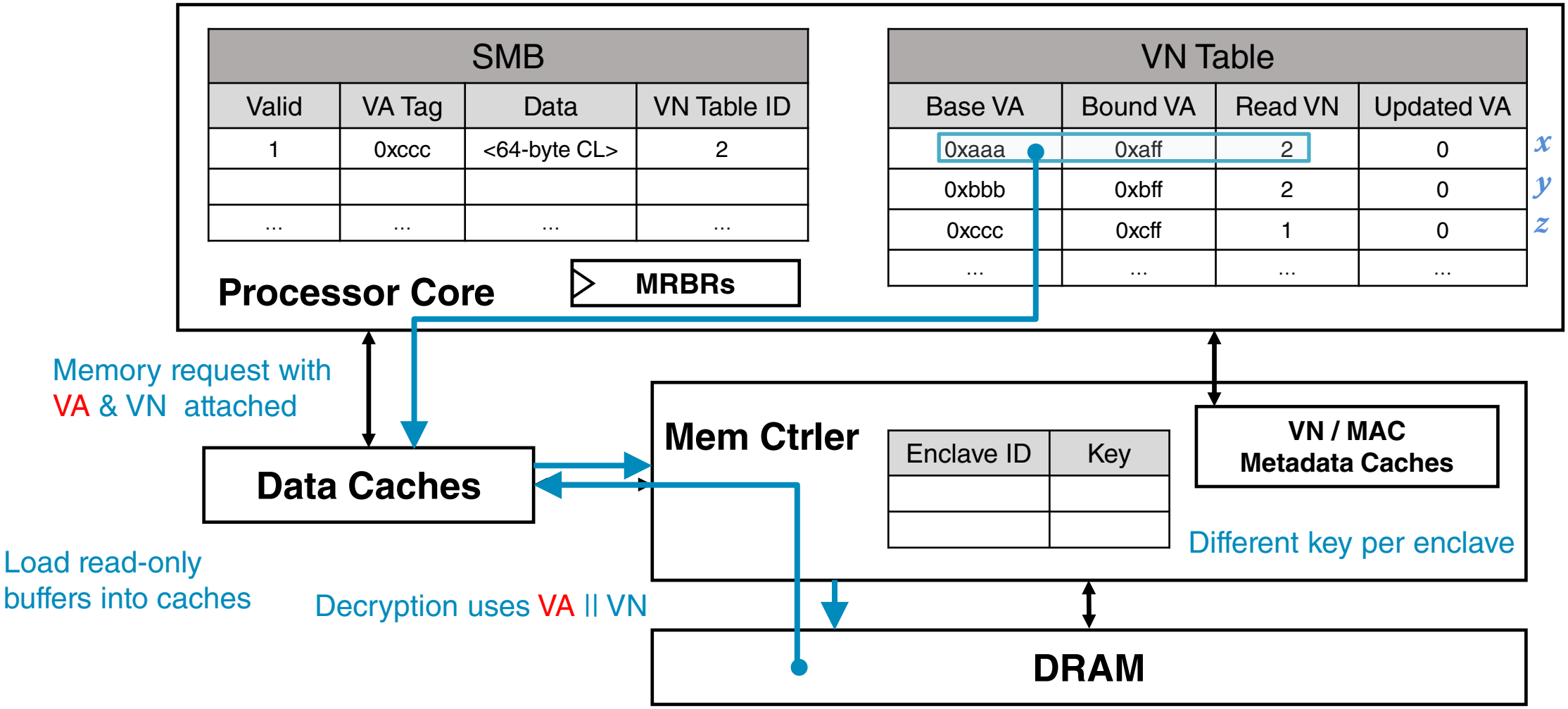


Solution: Use Virtual instead of Physical Addresses

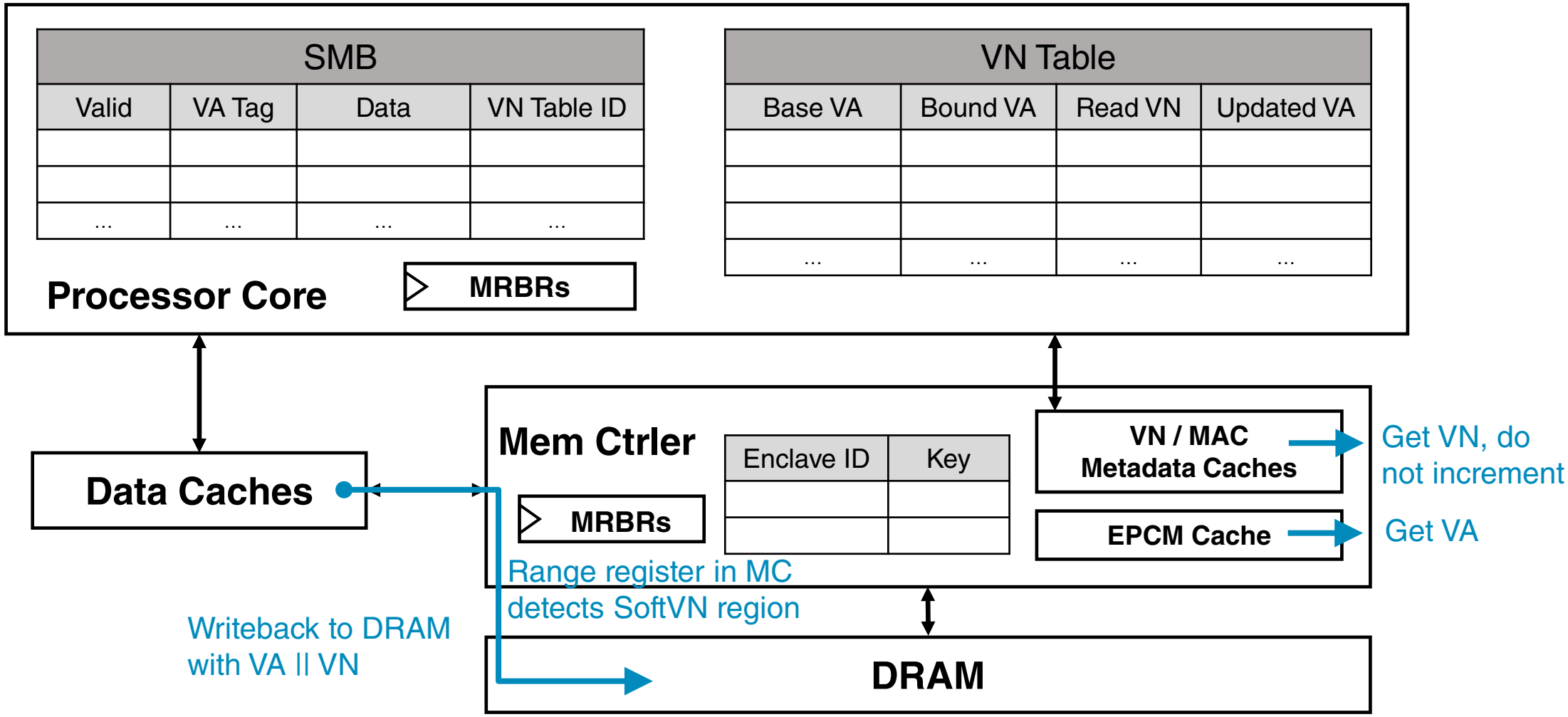


HW Mechanism – Loading SoftVN buffers

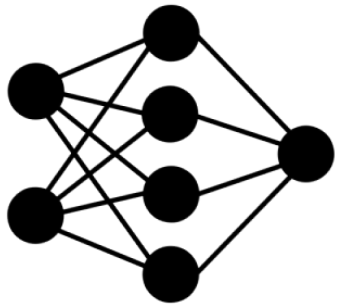
```
for(int i = 0; i < N; i++)
    z[i] = x[i] + y[i];
```



HW Mechanism – Writeback to DRAM



Applications



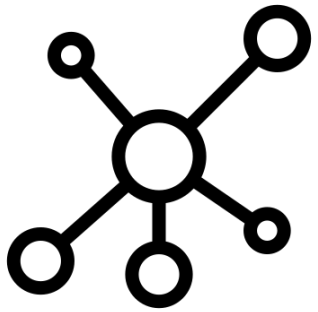
Deep Neural Networks

Image / Language / Recommender Models

28 kernels

6 LoC added on avrg.

max. 6/16 VN table & 1/4 SMB entries



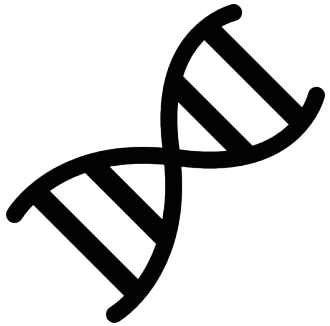
Graphs

BFS / PageRank

2 kernels

10 LoC added on avrg.

max. 5/16 VN table & 1/4 SMB entries



Bioinformatics

Global / Local Alignment

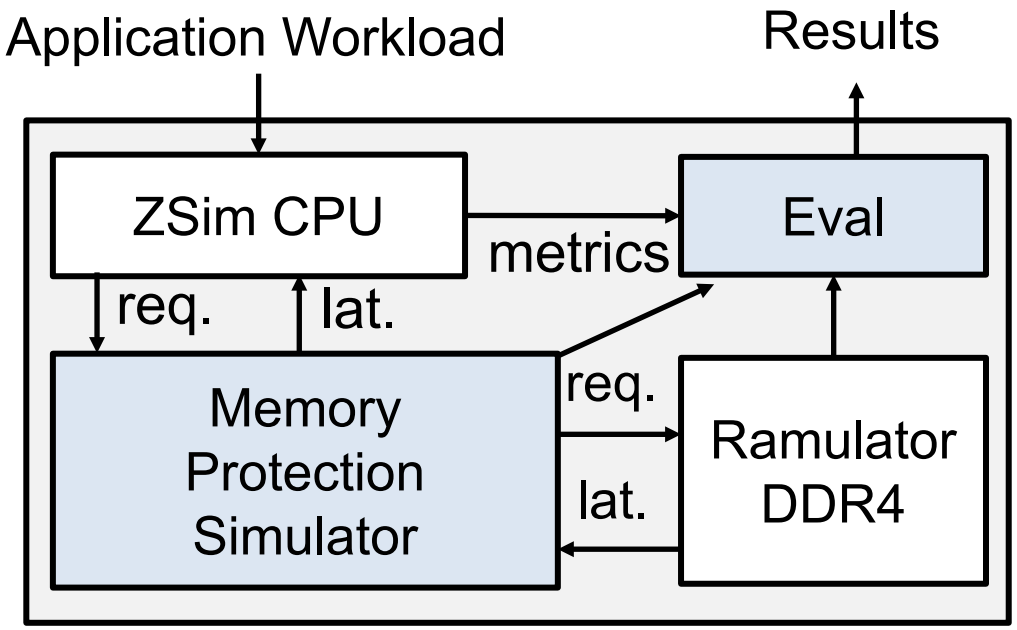
2 kernels

10 LoC added on avrg.

max. 4/16 VN table & 2/4 SMB entries

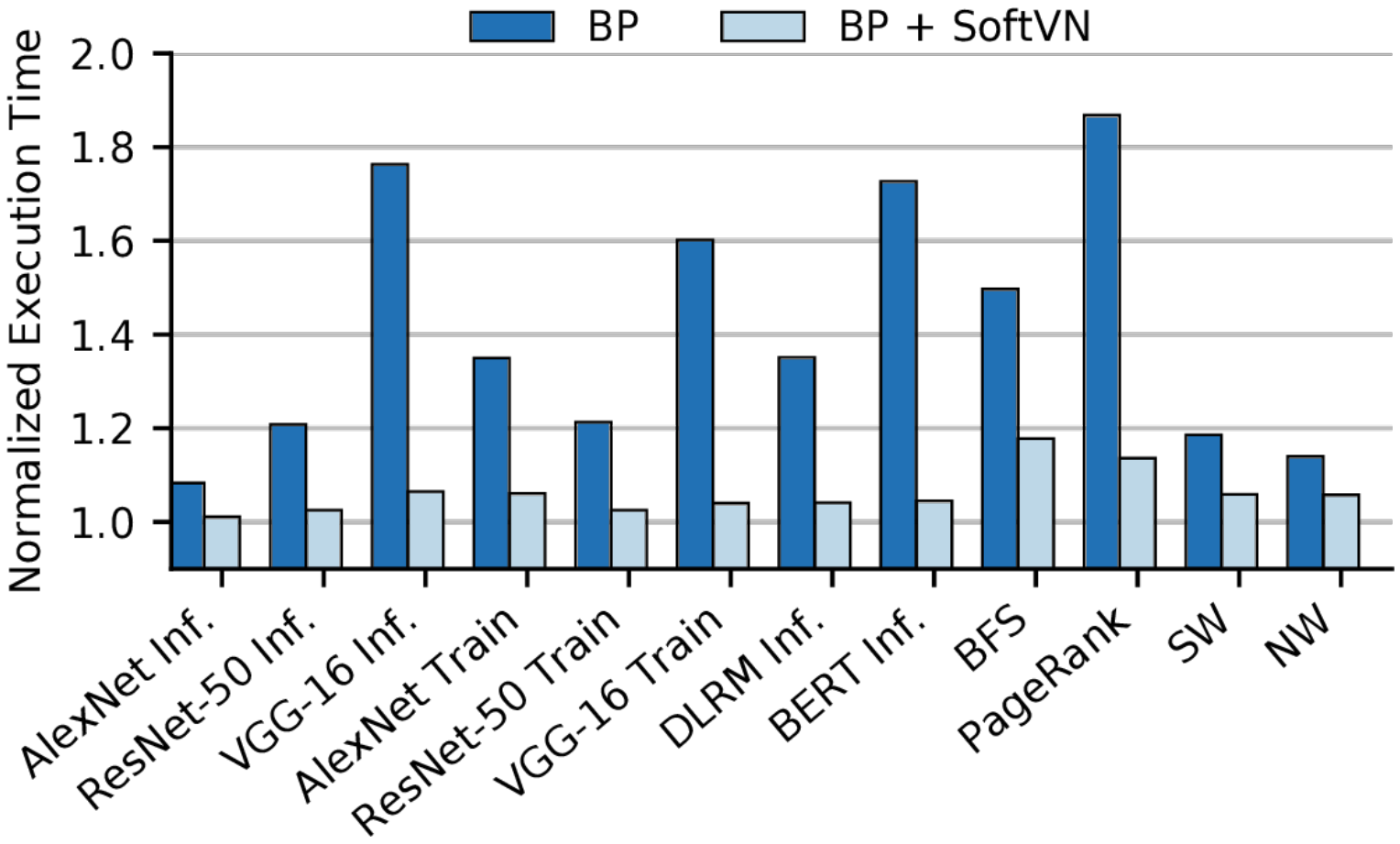
Evaluation Methodology

- ZSim [ISCA '13] OoO-Core Simulator & Ramulator [CAL '15] DRAM Simulator
- Baseline: Intel SGX-like protection
- 16 GB memory each for traditional protection and SoftVN regions
- C/C++ applications/kernels with modifications



Simulation Results

- SoftVN incurs only a **5%** perf. overhead on average, compared to 35% of baseline.
  - **33%** perf. improvement
  - **82%** overhead reduction
- SoftVN reduces the VN metadata accesses by **66%**.

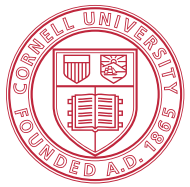


Summary

- SoftVN greatly lowers the overhead for memory protection in processor enclaves, by allowing software to track VNs for large data buffers and providing them for memory reads.

- More in the paper…
  - Details of hardware mechanisms
  - Details of applications
  - Security analysis
  - More results including sensitivity analysis

Thank you!
Questions?



Cornell University

Meta AI

CSL