Efficient Memory Integrity Verification and Encryption for Secure Processors

G. Edward Suh, Dwaine Clarke,
Blaise Gassend, Marten van Dijk,
Srinivas Devadas

Massachusetts Institute of Technology

New Security Challenges

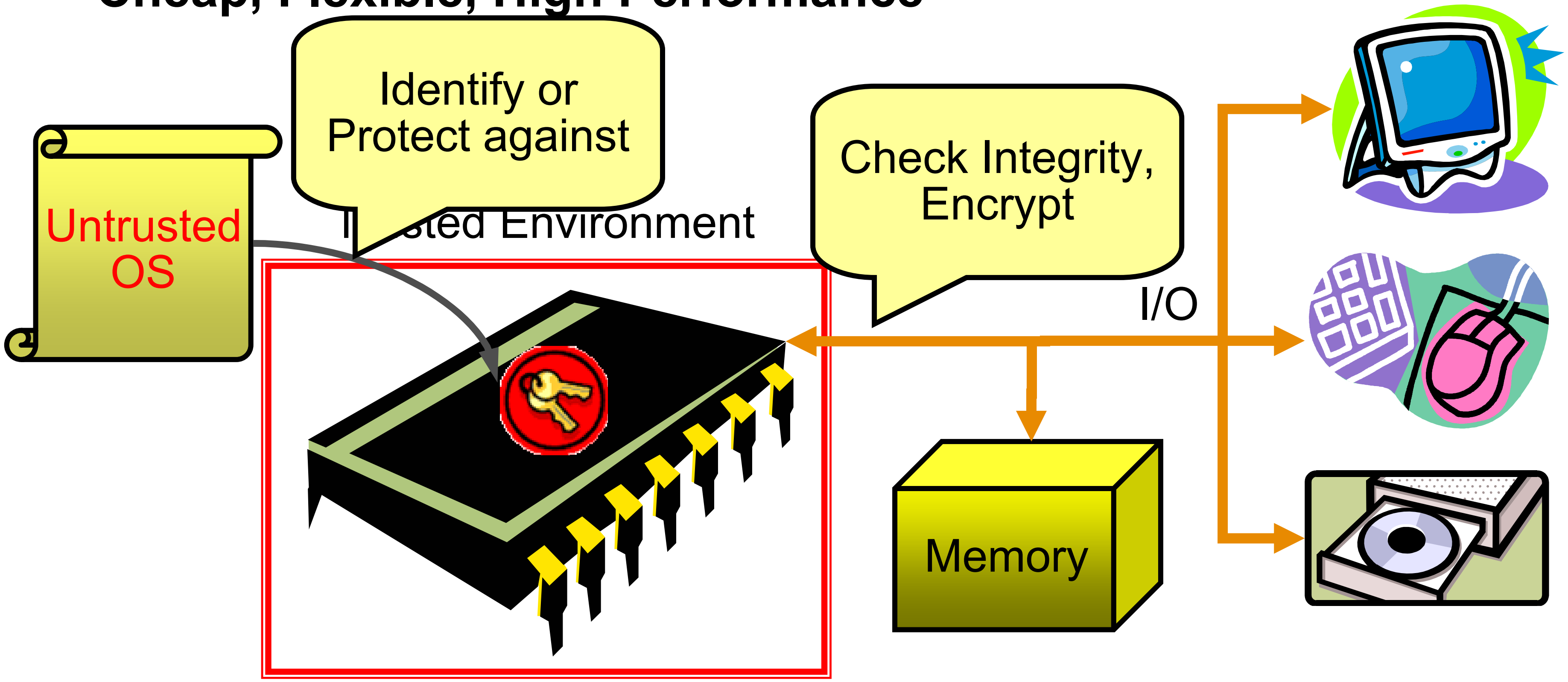
- Current computer systems have a large Trusted Computing Base (TCB)
  - Trusted hardware: processor, memory, etc.
  - Trusted operating systems, device drivers
- Future computers should have a much smaller TCB
  - Untrusted OS
  - Physical attacks → Without additional protection, components cannot be trusted
- Why smaller TCB?
  - Easier to verify and trust
  - Enables new applications

Applications

- Emerging applications require TCBs that are secure even from an owner
- Distributed computation on Internet/Grid computing
  - SETI@home, distributed.net, and more
  - Interact with a random computer on the net → how can we trust the result?
- Software licensing
  - The owner of a system is an attacker
- Mobile agents
  - Software agents on Internet perform a task on behalf of you
  - Perform sensitive transactions on a remote (untrusted) host

Single-Chip AEGIS Secure Processors

- Only trust a single chip: tamper-resistant
  - Off-chip memory: verify the integrity and encrypt
  - Untrusted OS: identify a core part or protect against OS attacks
- Cheap, Flexible, High Performance



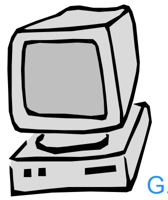
Secure Execution Environments

- Tamper-Evident (TE) environment
  - Guarantees a valid execution and the identity of a program; no privacy
  - Any software or physical tampering to alter the program behavior should be detected

  → Integrity verification

- Private Tamper-Resistant (PTR) environment
  - TE environment + privacy
  - Assume programs do not leak information via memory access patterns

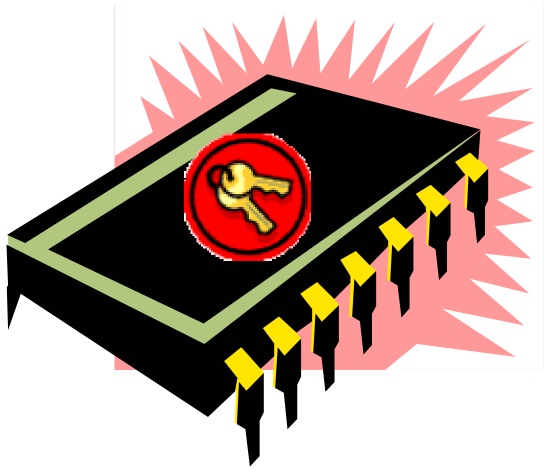
  → Encryption + Integrity verification



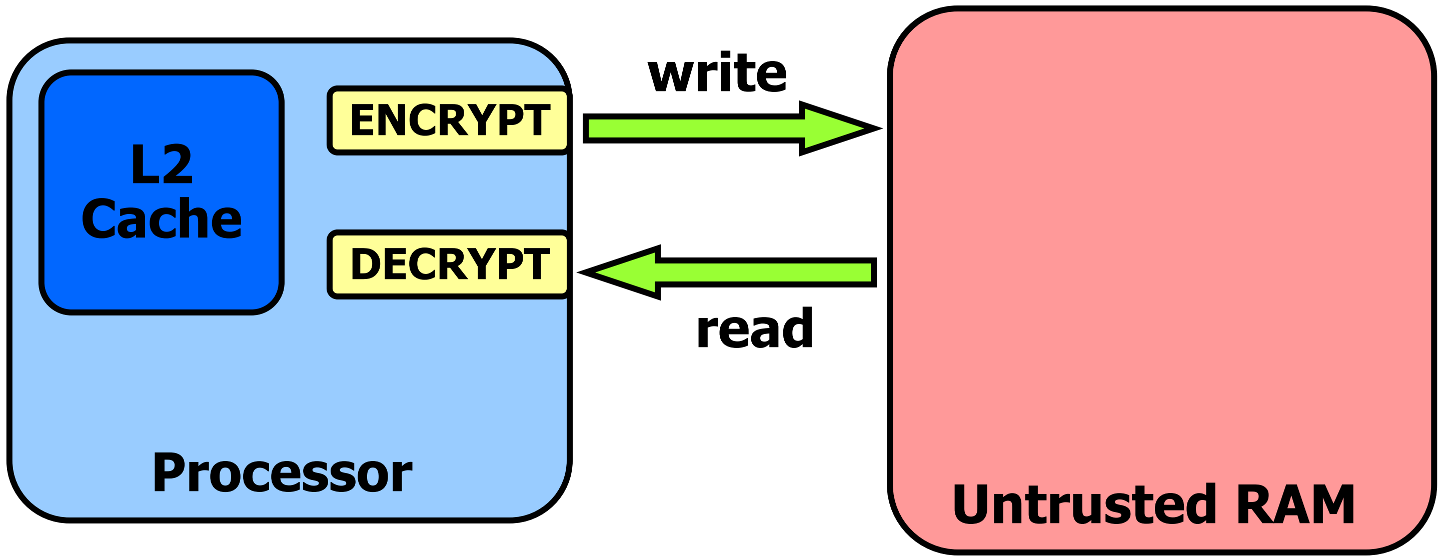
Other Trusted Computing Platforms

- IBM 4758 cryptographic coprocessor
  - Entire system (processor, memory, and trusted software) in a tamper-proof package
  - Expensive, requires continuous power
- XOM (eXecution Only Memory): David Lie et al
  - Stated goal: Protect integrity and privacy of code and data
  - Memory integrity checking does not prevent replay attacks
  - Always encrypt off-chip memory
- Palladium/NGSCB: Microsoft
  - Stated goal: Protect from software attacks
  - Memory integrity and privacy are assumed (only software attacks)

Memory Encryption

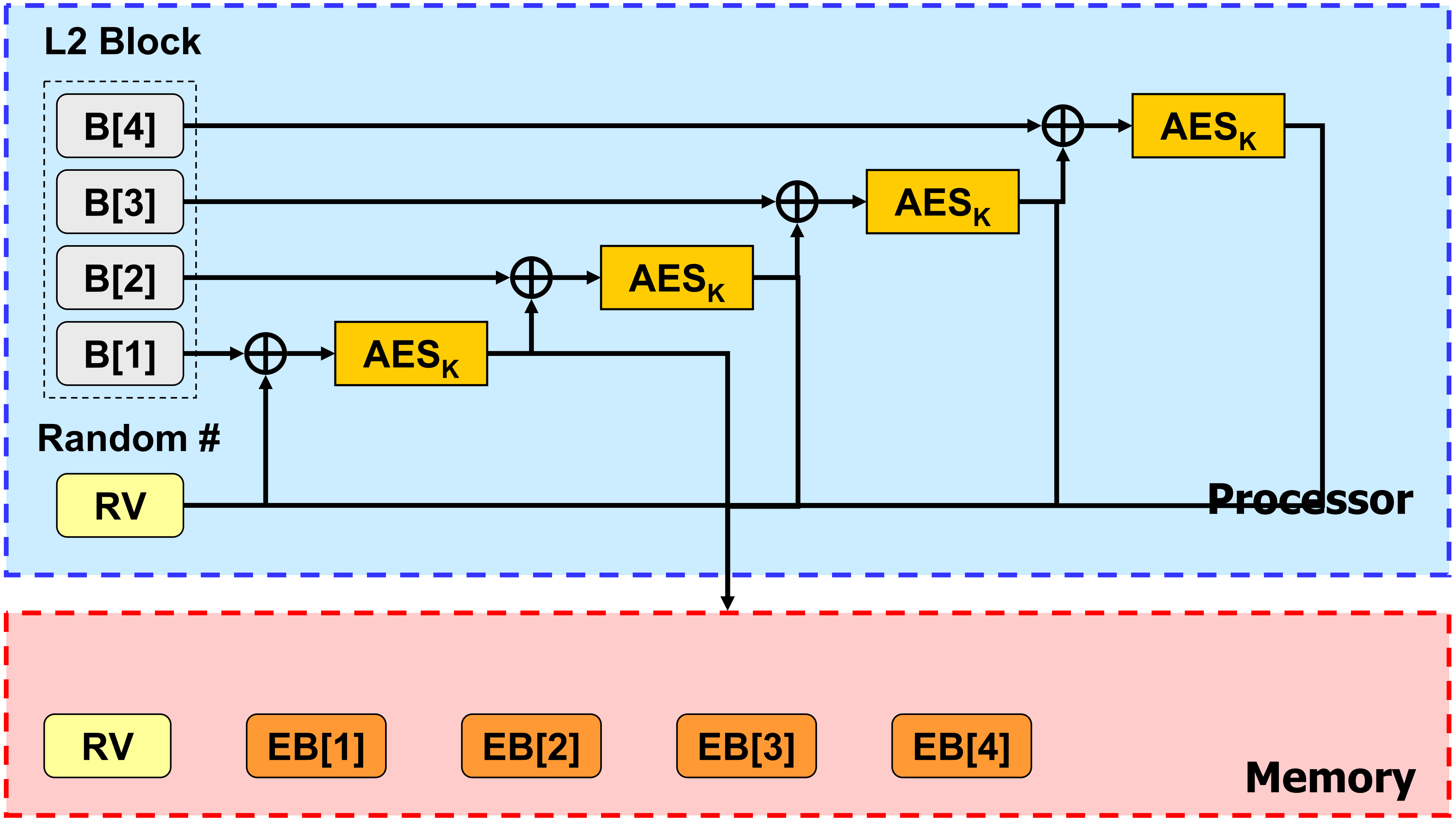


Memory Encryption

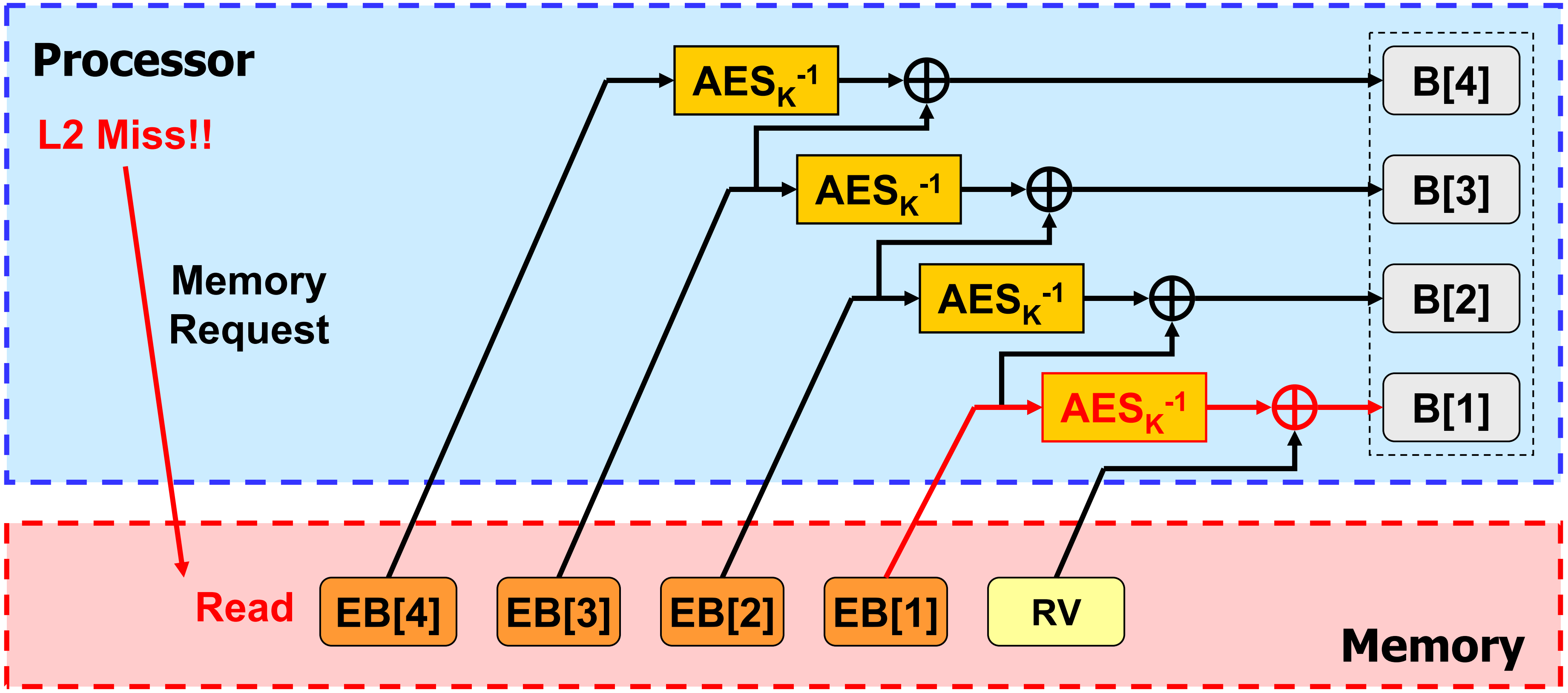


- Encrypt on an L2 cache block granularity
  - Use symmetric key algorithms (AES, 16 Byte chunks)
  - Should be randomized to prevent comparing two blocks
  - Adds decryption latency to each memory access

Direct Encryption (CBC mode): encrypt



Direct Encryption (CBC mode): decrypt

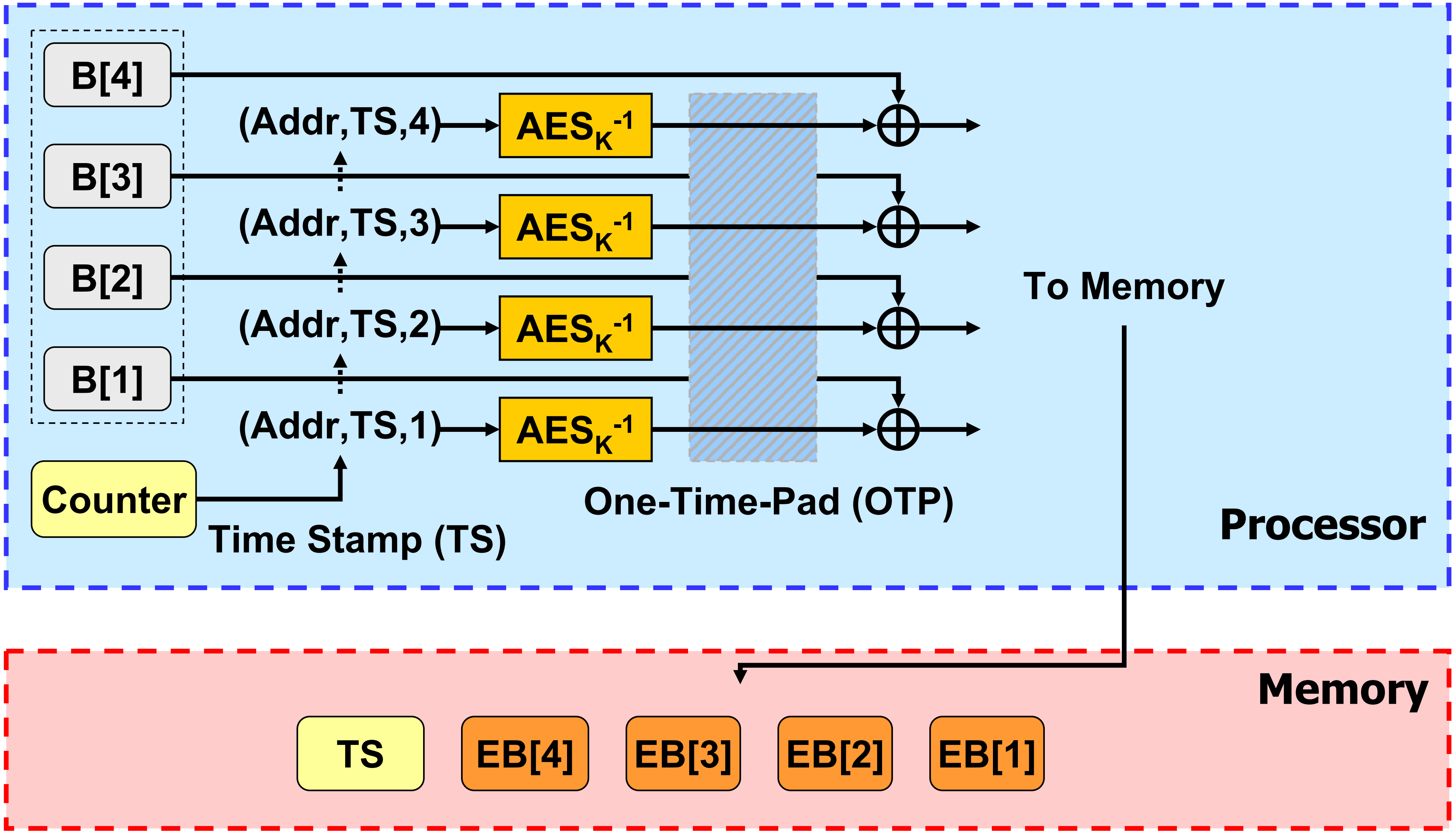


- **Off-chip access latency**

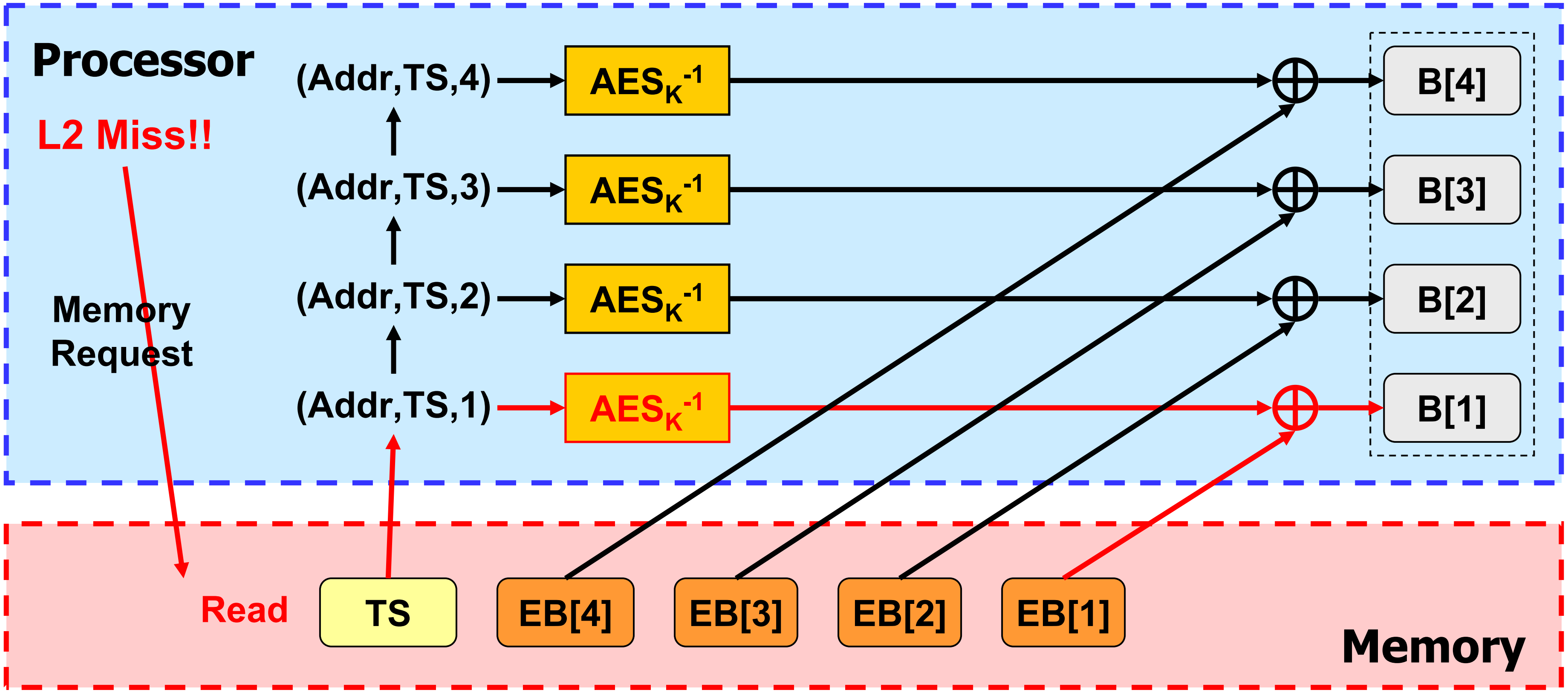
  = latency for the last chunk of an L2 block + AES + XOR

  → Decryption directly impacts off-chip latency

One-Time-Pad Encryption (OTP): encrypt



One-Time-Pad Encryption (OTP): decrypt



- Off-chip access latency = MAX(latency for the time stamp + AES, latency for an L2 block) + XOR

  → Overlap the decryption with memory accesses

Effects of Encryption on Performance

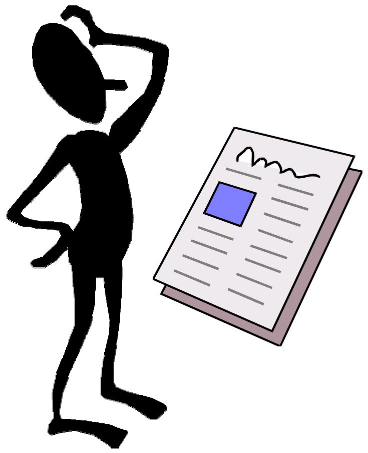
- Simulations based on the SimpleScalar tool set
  - 9 SPEC CPU2000 benchmarks
  - 256-KB, 1-MB, 4-MB L2 caches with 64-B blocks
  - 32-bit time stamps and random vectors → No caching!
  - Memory latency: 80/5, decryption latency: 40
- Performance degradation by encryption

	Direct (CBC)	One-Time-Pad
Worst Case	25%	18%
Average	13%	8%

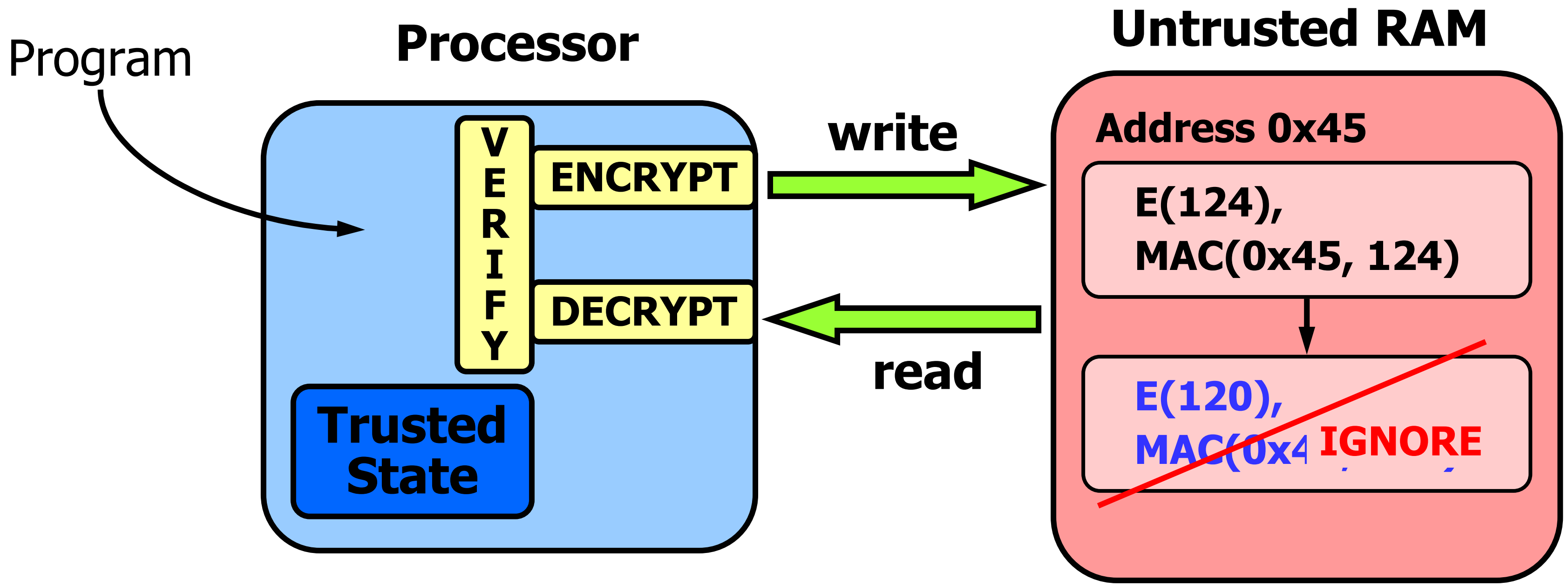
Security and Optimizations

- The security of the OTP is at least as good as the conventional CBC scheme
  - OTP is essentially a counter-mode (CTR) encryption
- Further optimizations are possible
  - For static data such as instructions, time stamps are not required → completely overlap the AES computations with memory accesses
  - Cache time stamps on-chip, or speculate the value
- Will be used for instruction encryption of Philips media processors

Integrity Verification



Difficulty of Integrity Verification

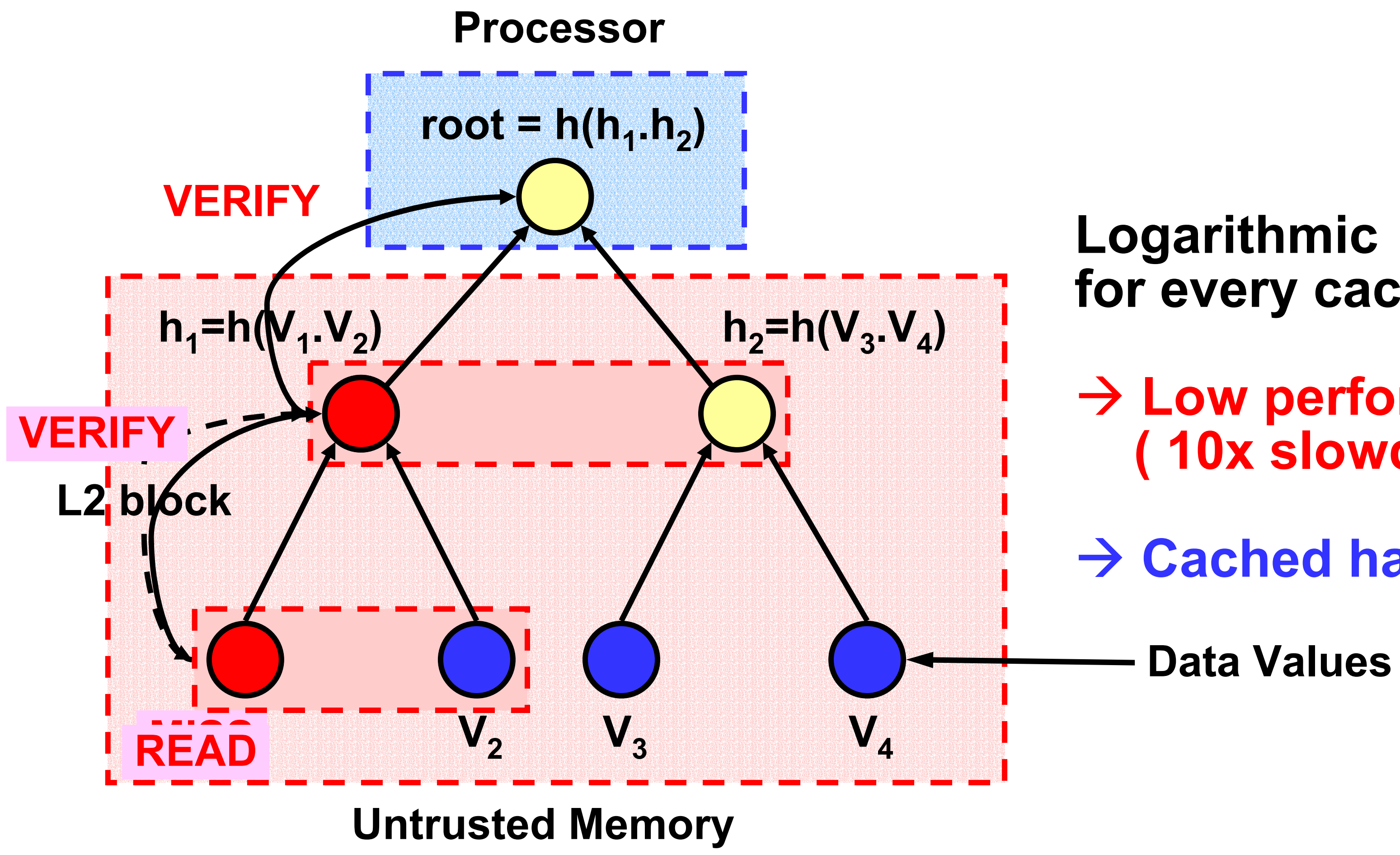


Cannot simply MAC on writes and check the MAC on reads

→ Replay attacks

Hash trees for integrity verification

Hash Trees

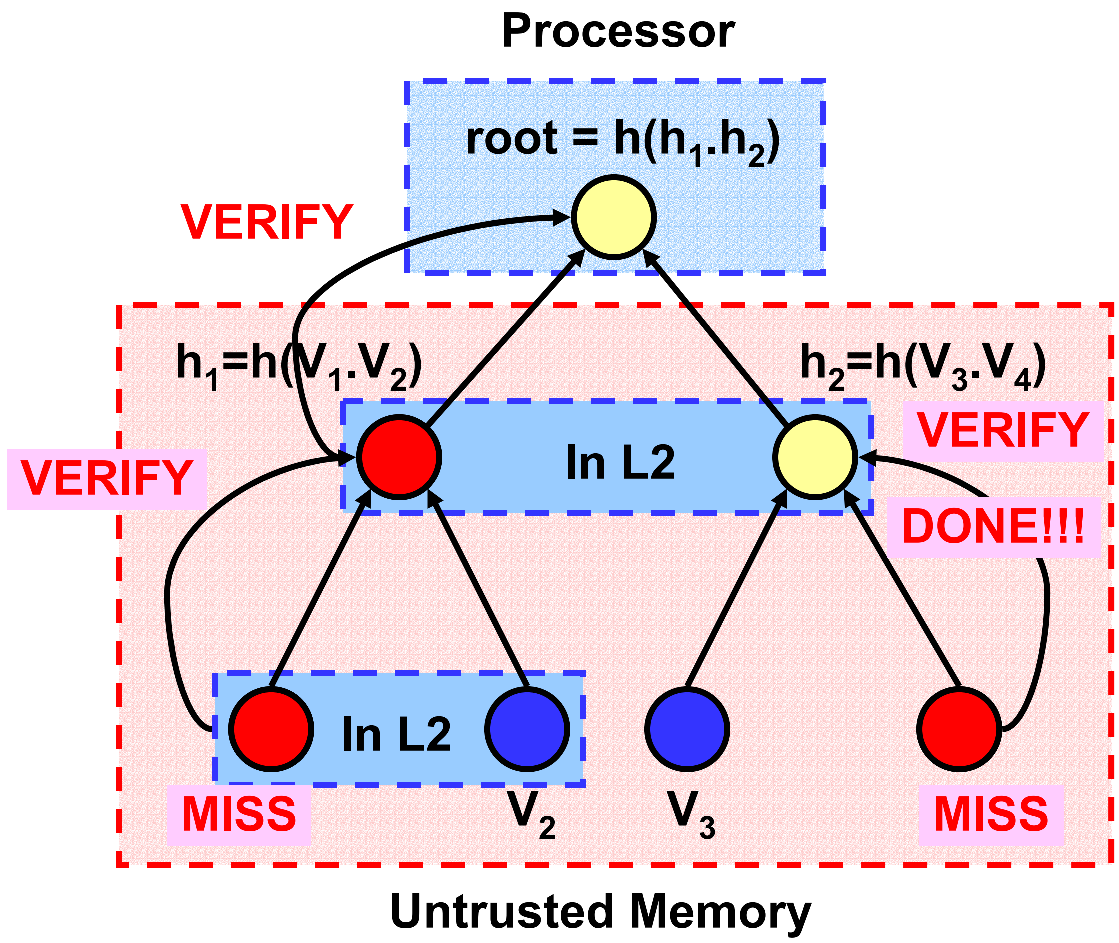


Logarithmic overhead for every cache miss

→ Low performance (10x slowdown)

→ Cached hash trees

Cached Hash Trees (HPCA'03)



Cache hashes in L2

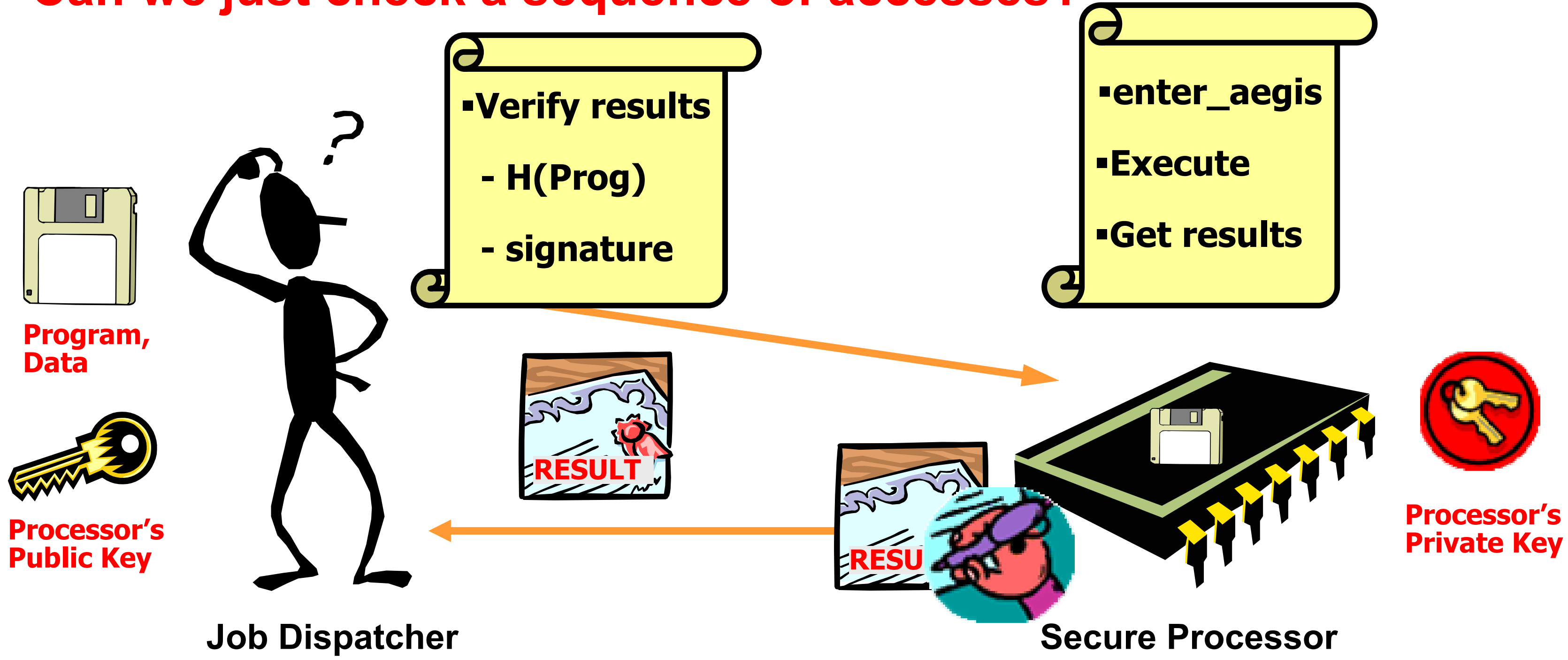
✓ L2 is trusted
✓ Stop checking earlier

→ Less overhead (22% average, 51% worst case)

→ Still expensive

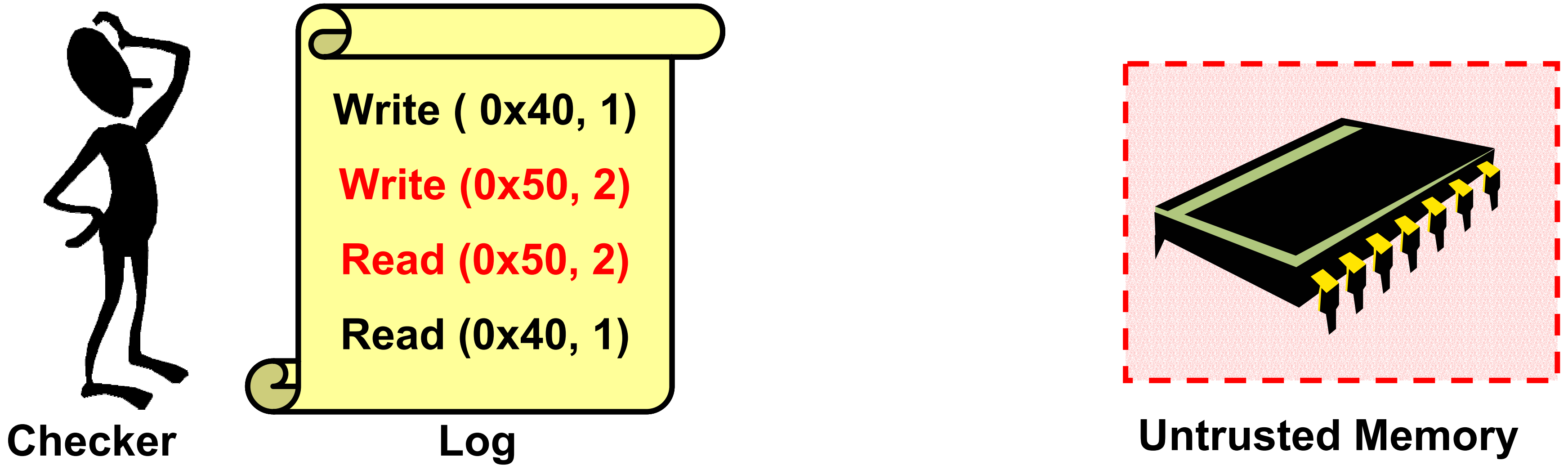
Can we do better?

- Some applications only require to verify memory accesses after a long execution
  - Distributed computation
  - No need to check after each memory access
- Can we just check a sequence of accesses?



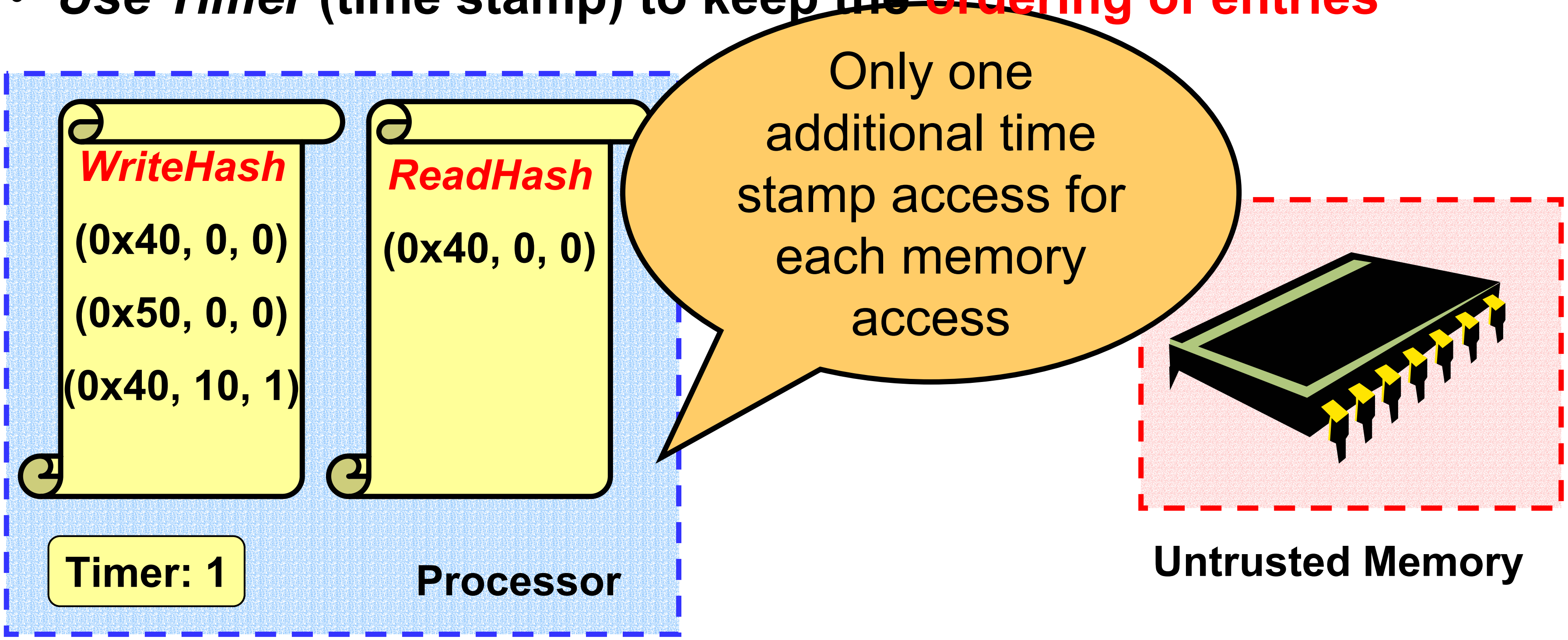
Log Hash Integrity Verification: Idea

- At run-time, maintain a log of reads and writes
  - **Reads: make a 'read' note with (address, value) i**
  - **Writes: make a 'write' note with (address, value)**
- **check**: go thru log, check each read has the most recent value written to the address
- Problem!!: Log grows → use cryptographic hashes



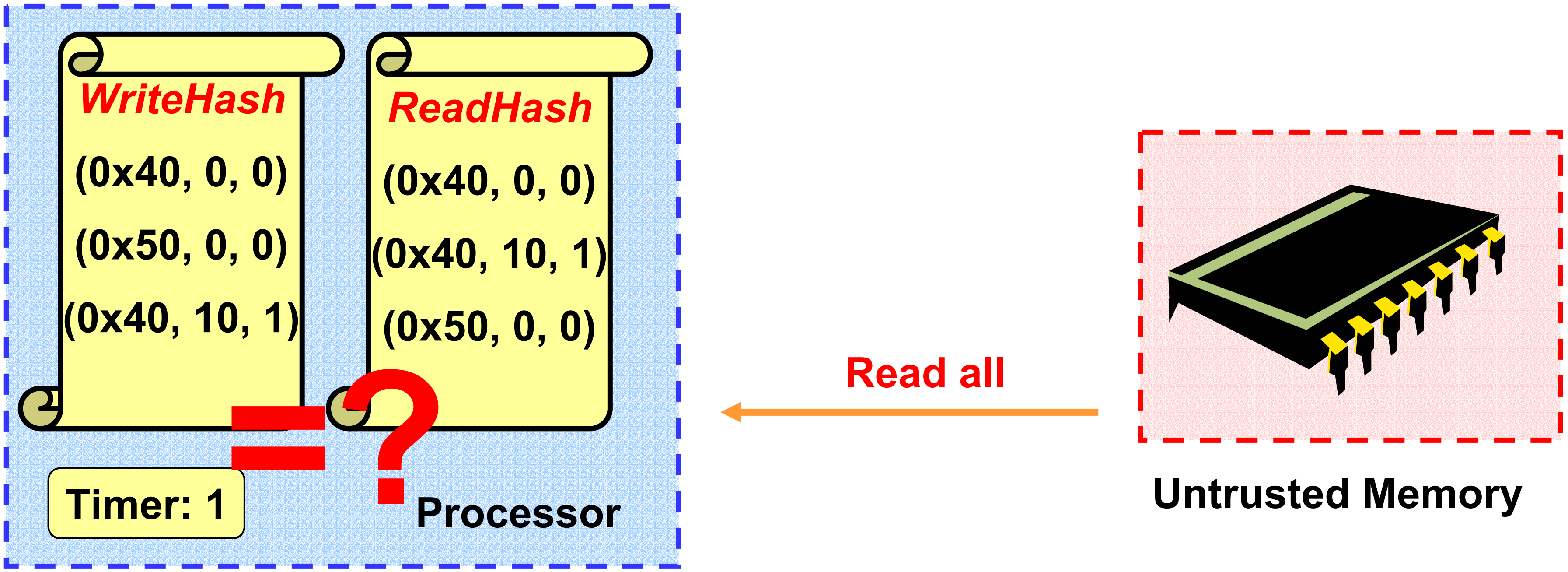
Log Hash Algorithms: Run-Time

- Use set hashes as compressed logs
  - Set hash: maps a set to a fixed length string
  - *ReadHash*: a set of read entries (addr, val, time) in the log
  - *WriteHash*: a set of write entries (addr, val, time) in the log
- *Use Timer* (time stamp) to keep the ordering of entries



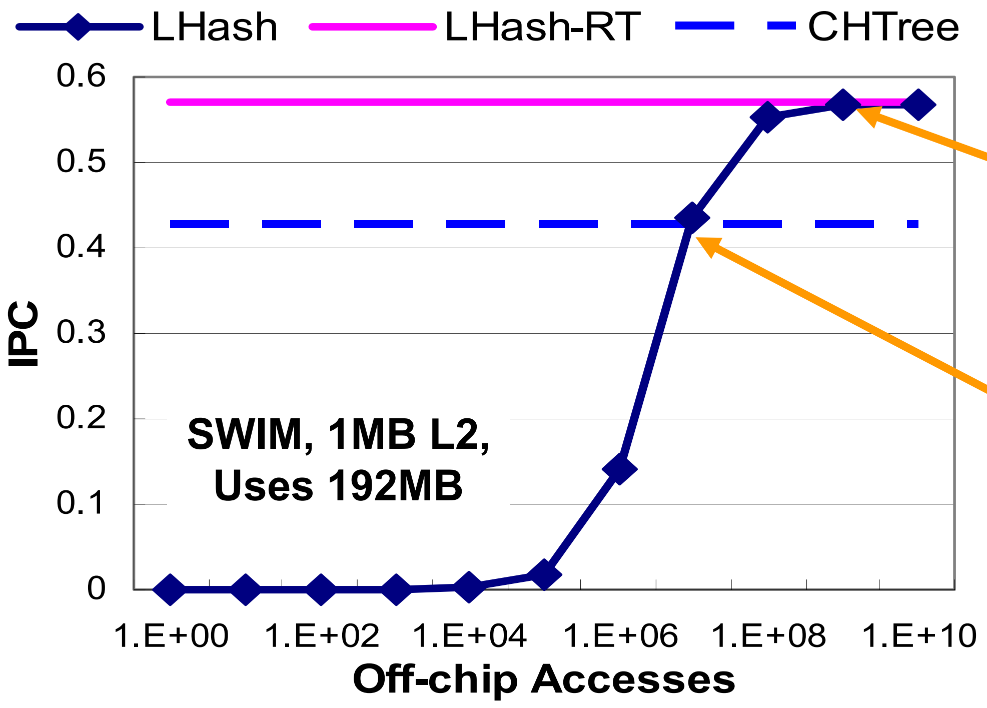
Log Hash Algorithms: Integrity Check

- **Read all the addresses** that are not in a cache
- Compare *ReadHash* and *WriteHash* (same set?)



Checking Overhead of Log Hash Scheme

- Integrity check requires reading the entire memory space being used
  - Cost depends on the size and the length of an application
- For long programs, the checking overhead is negligible
  - Amortized over a long execution time



Check overhead is negligible for programs w/ more than a billion accesses

Better than Hash Trees for programs w/ more than 10 million accesses

Performance Comparisons

- Overhead for TE environments
  - Integrity verification

	CHTree	LHash
Worst Case	52%	15%
Average	22%	4%

- Overhead for PTR environments
  - Integrity verification + encryption

	CHTree + CBC	LHash + OTP
Worst Case	59%	23%
Average	31%	10%

Summary

- Untrusted owners are becoming more prevalent
  - Untrusted OS, physical attacks → requires a small TCB
- Single-chip secure processors require off-chip protection mechanisms: Integrity verification and Encryption
- OTP encryption scheme reduces the overhead of encryption in all cases
  - Allows decryption to be overlapped with memory accesses
  - Cache or speculate time stamps to further hide decryption latency
- Log Hash scheme significantly reduces the overhead of integrity verification for certified execution when programs are long enough

Questions?

More Information at www.csg.lcs.mit.edu