



CREATING A SPIDER GOAT: USING TRANSACTIONAL MEMORY SUPPORT FOR SECURITY

Igor Muttik | PhD

Alex Nayshtut | CISSP-ISSAP

Roman Dementiev | PhD

Introducing Spider Goat...



Spider Pig

+ Sacrificial Goat

= Spider Goat

Agenda

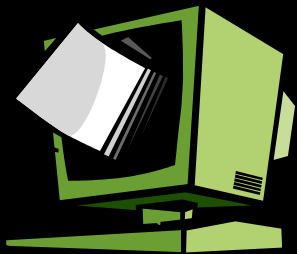
- Monitoring memory
- Transaction synchronization problem
- What is TSX?
- Can we use TSX for security?
- Demo – detecting unauthorized memory changes
- Potential applications: advantages and challenges
- Memory monitoring approaches
- Conclusions



Monitoring Memory for Changes

1) Reading protected RAM cells in a loop



- Malicious code (e.g. shellcode) may have time to disable security
- CPU intensive
- No reverting of changes

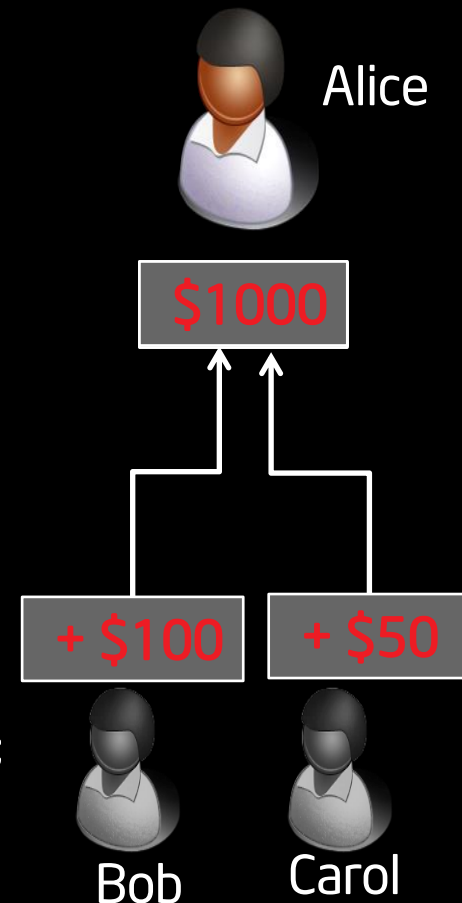


2) Setting up exceptions on memory pages (4k)

- Mark protected pages read-only – any write creates a page fault
- Software then has to check what exactly was changed
- Reverting changes requires storing original data (up to 2*RAM)

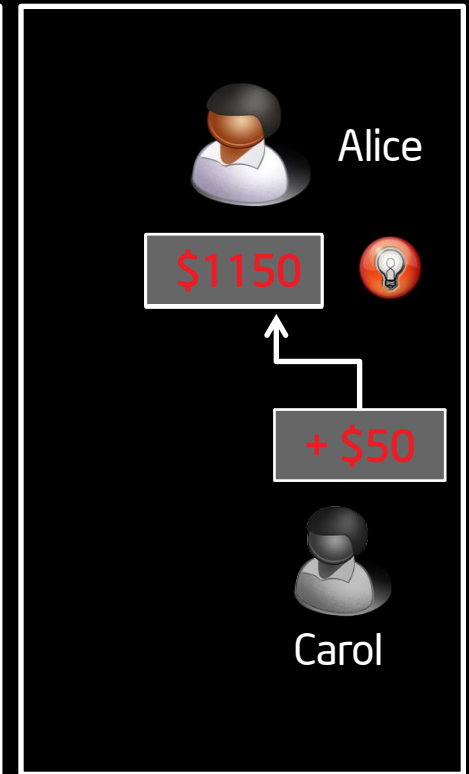
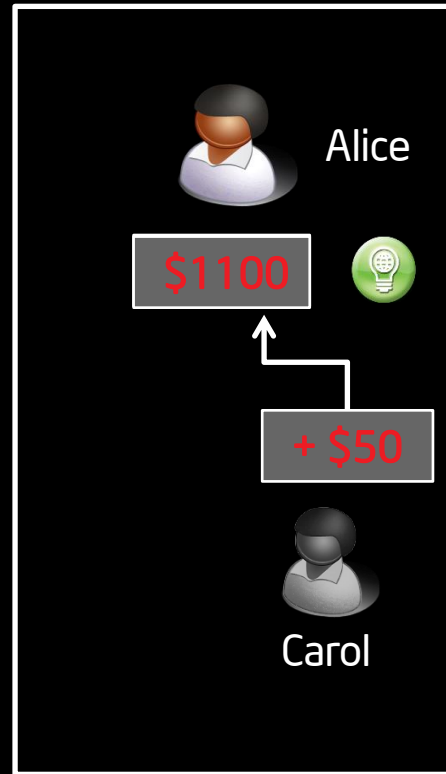
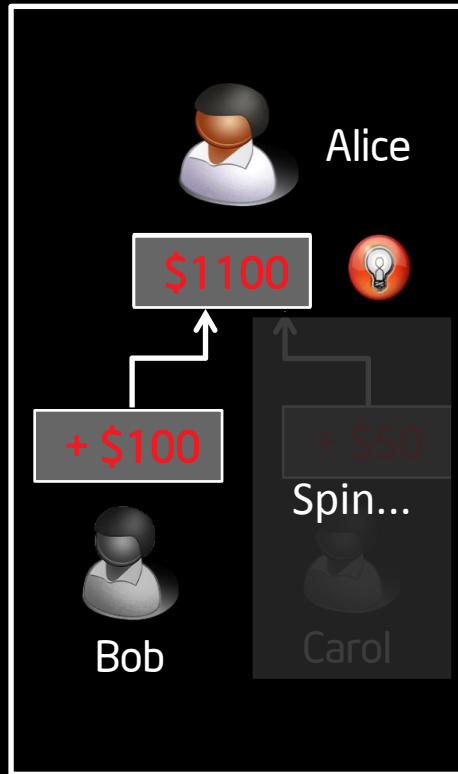
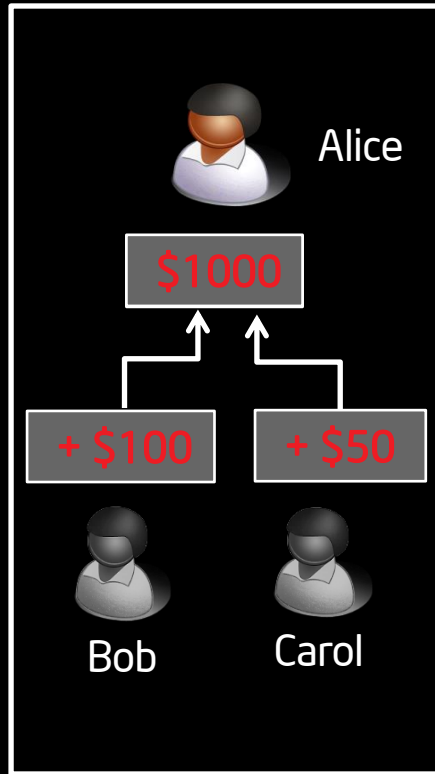
Transaction Synchronization Problem

- Multiple threads accessing the same database record is a scenario very common in database software!
- Example:
 - Alice has \$1000 in a database record in her account as a total value
 - Bob wants to transfer \$100 to Alice
 - Carol also wants to transfer to Alice but \$50
- Depending on timing of reads and writes the end result may be either **\$1100** or **\$1050** or **\$1150**
- Typical solution – software lock (like a semaphore ; other threads just wait for a “green light” )
 - bad for multi-core CPUs



Read/write conflict of 2+ threads on a memory cell is the root problem

Using Locks for Transactions (steps)



A Bit of Context

The basic idea was simple: if hardware suffers more transient failures as it gets smaller, why not allow software to detect erroneous computations and re-execute them? This idea seemed promising until John realized THAT IT WAS THE WORST IDEA EVER. Modern software barely works when the hardware is correct, so relying on software to correct hardware errors is like asking Godzilla to prevent Mega-Godzilla from terrorizing Japan. THIS DOES NOT LEAD TO RISING PROPERTY VALUES IN TOKYO.



Source: "The Slow Winter" by James Mickens
https://www.usenix.org/system/files/1309_14-17_mickens.pdf

Today, if a person uses a desktop or laptop, she is justifiably angry if she discovers that her machine is doing a non-trivial amount of work. If her hard disk is active for more than a second per hour, or if her CPU utilization goes above 4%, she either has a computer virus, or she made the disastrous decision to run a Java program. Either way, it's not your fault: you brought the fire down from Olympus, and the mortals do with it what they will.

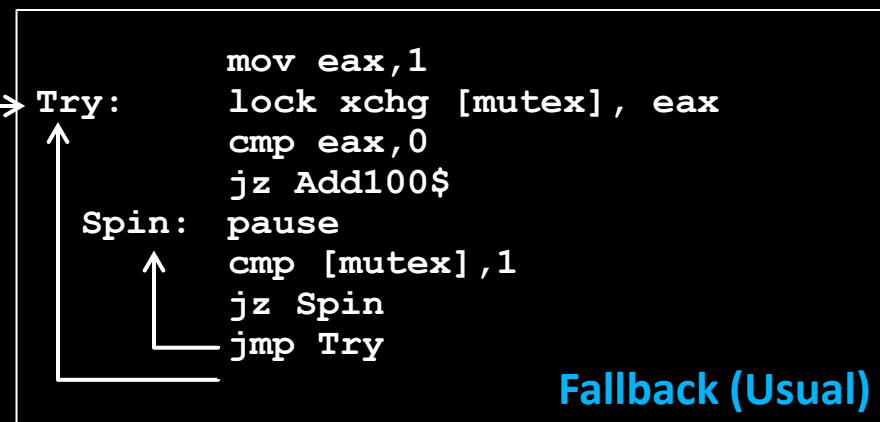
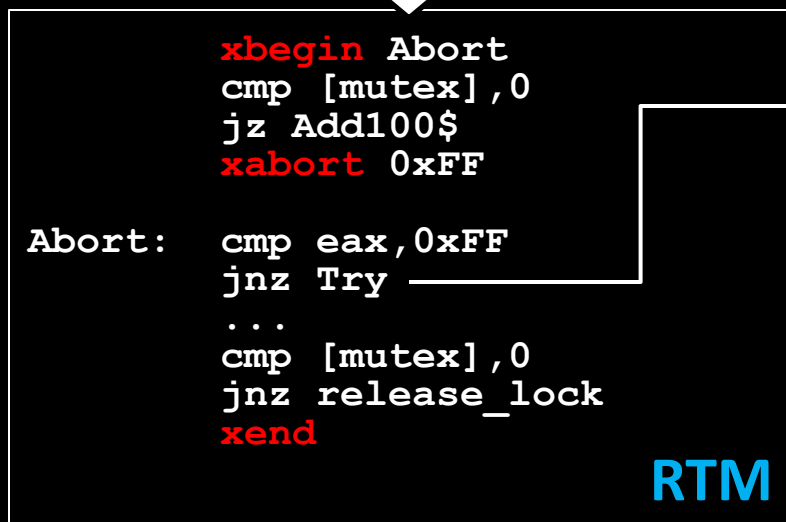
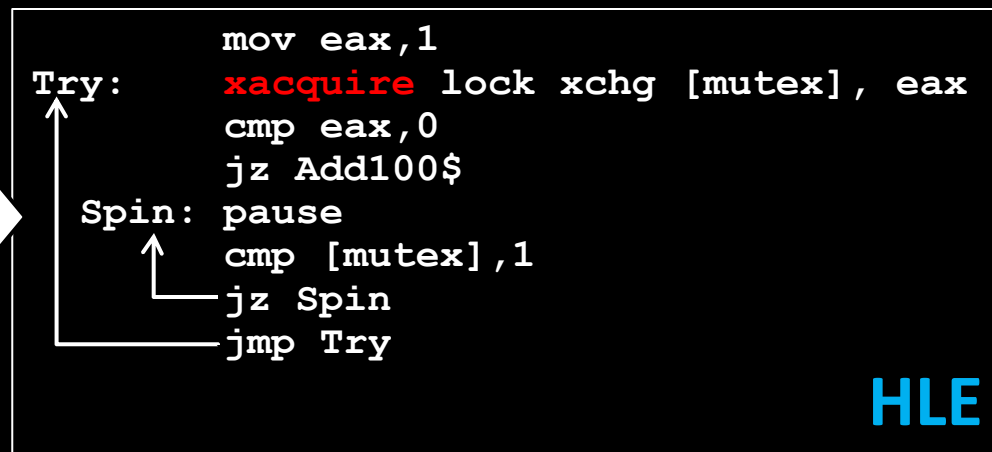
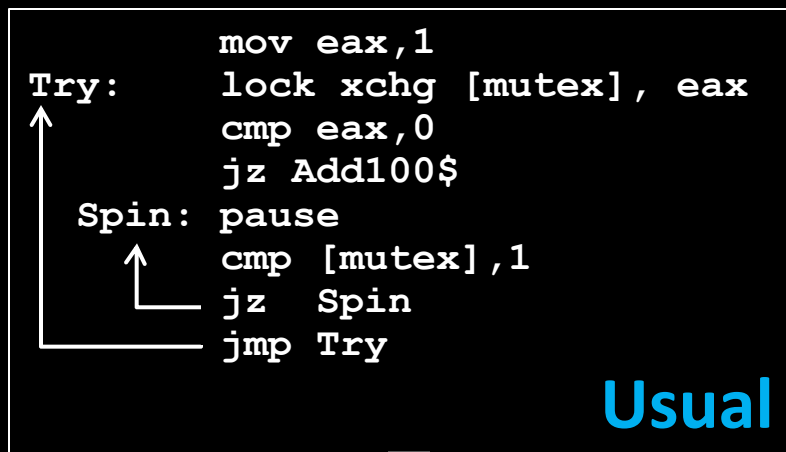


Transactional Synchronization Hardware

- Commercial transactional memory
 - 2011 – IBM Blue Gene/Q
 - 2012 – IBM zEC12 mainframe
 - 2013 – Intel® TSX (4th gen CPUs: Haswell+)
 - 2014 – IBM Power8
- Intel® Transactional Synchronization Extensions (TSX) is an optimization to resolve threads' conflicts. Two kinds:
 - Hardware Lock Elision (HLE - legacy compatible extension)
 - Restricted Transactional Memory (RTM - new instructions)
- New instructions:
 - HLE: XACQUIRE, XRELEASE (instruction prefixes)
 - RTM: XBEGIN, XEND (instructions)
 - Auxiliary XABORT, XTEST
 - CPUID check for TSX RTM support: `(EAX=07H, ECX=0H).EBX.RTM[bit 11]==1`



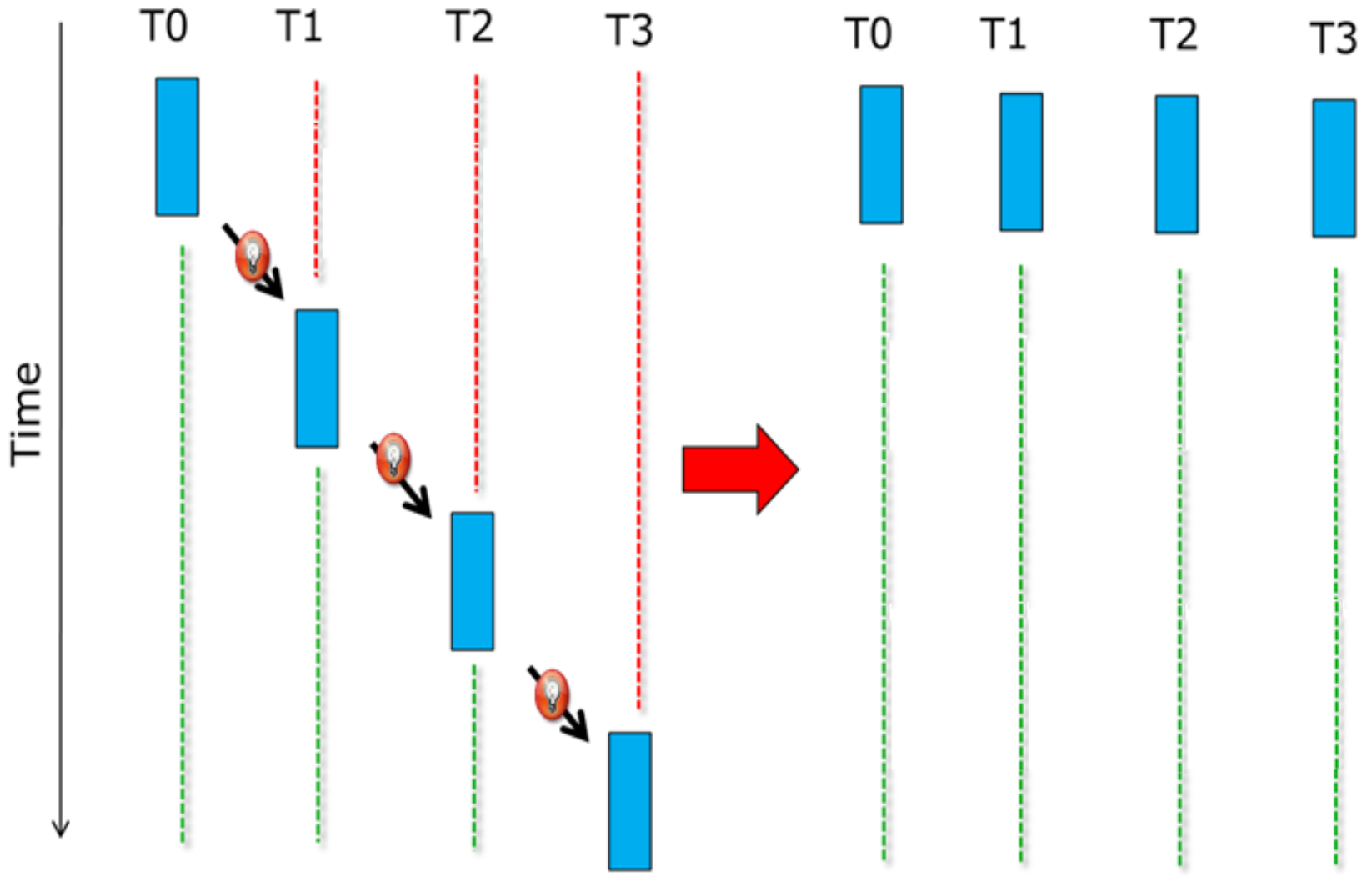
Coding Transactions with TSX



Code example for illustration only.

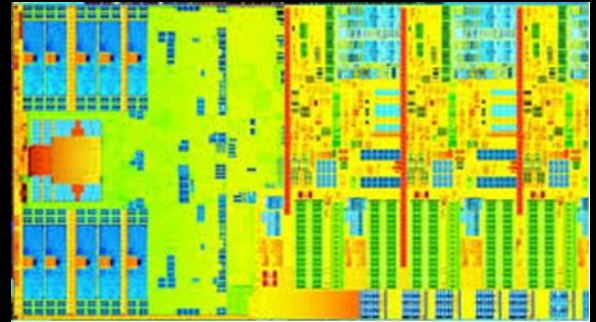
Source: http://en.wikipedia.org/wiki/Transactional_Synchronization_Extensions.

The Effect of TSX



Mechanics of TSX

- What TSX is NOT:
 - Not related to virtualization
 - Not based on RAM page tables (neither PT or EPT)
- Transactional support is piggybacking on cache
 - So its granularity is a cache line (x86: 64 bytes)
 - Has its own dedicated memory (32kb+ per physical core)
 - Existing cache coherency protocols detect conflicts

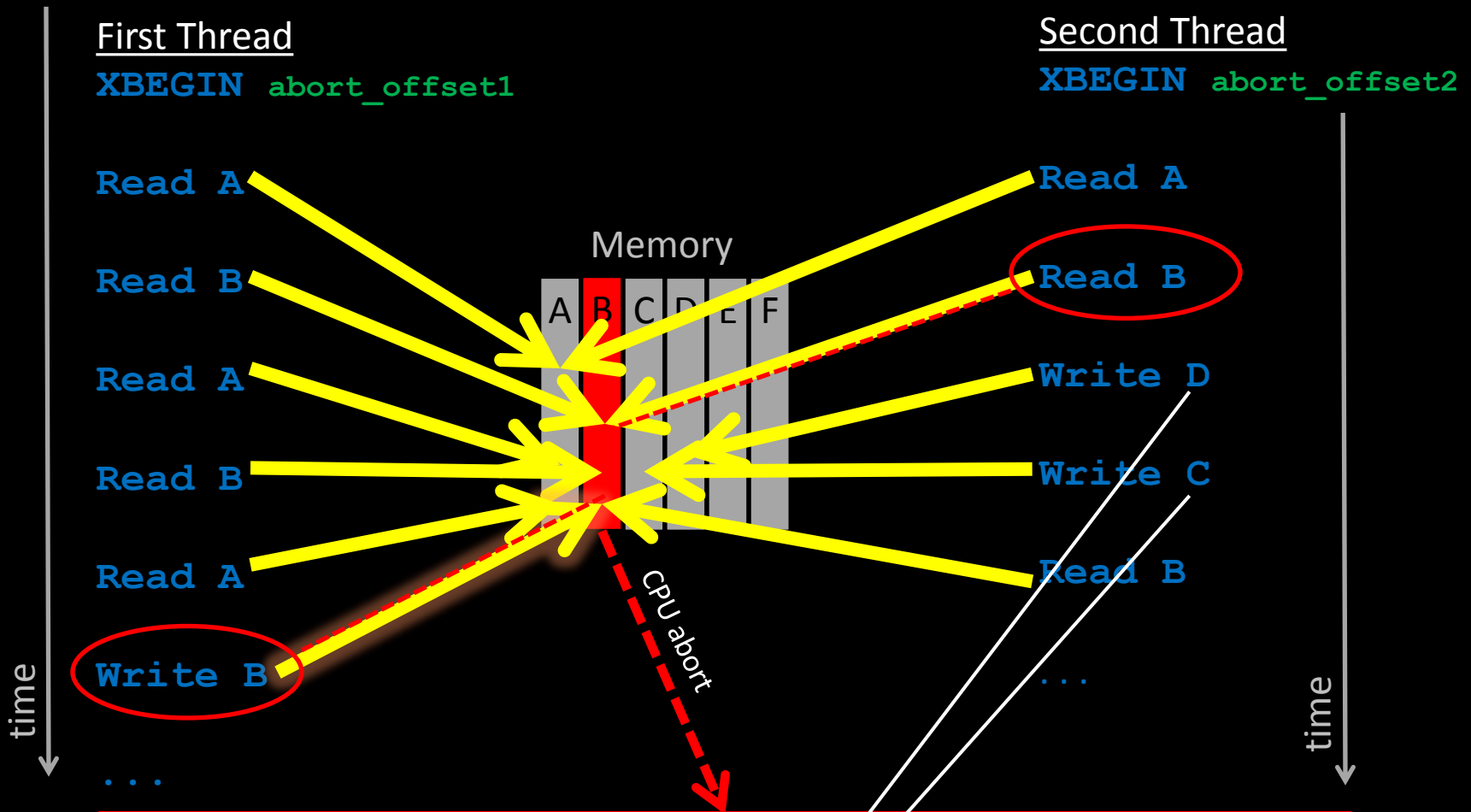


Detecting Conflicts between Threads

- Works for threads in different cores and for hyper-threads
- Modified data is not visible to other threads before a commit (XEND)
- Requires a programmer to mark transactions in source code
- Read/write conflict is resolved by the abort handler (only for RTM)
 - Similar to an exception handler (defined in XBEGIN parameter)
 - May re-run the transaction several times
 - May call a fallback routine (e.g. with the software lock)

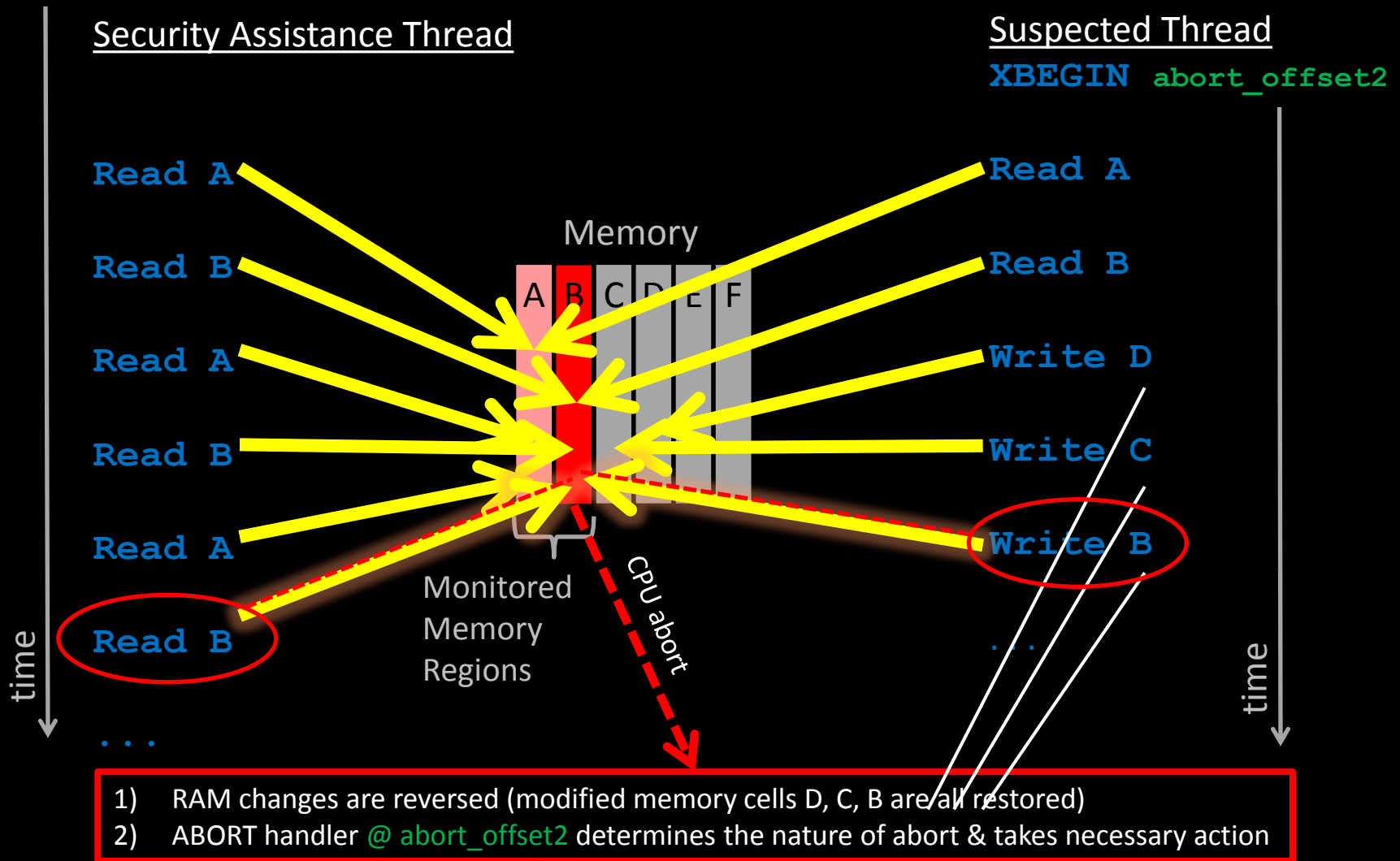


Typical TSX use

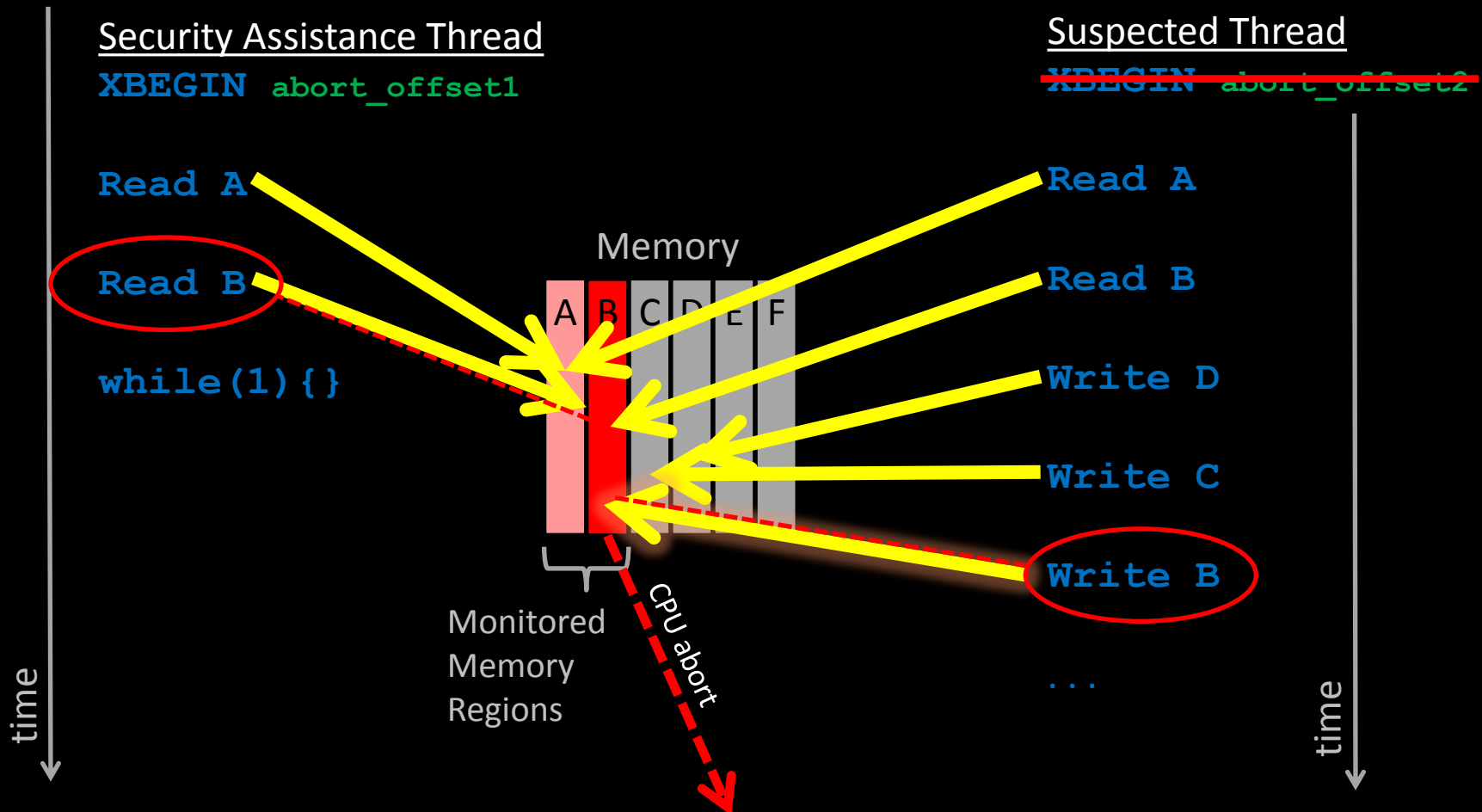


- 1) RAM changes are reversed (modified memory cells D, C are all restored)
- 2) ABORT handler @ `abort_offset2` determines the nature of abort & takes necessary action

Reverting RAM Modification with TSX



Detecting RAM Modification with TSX



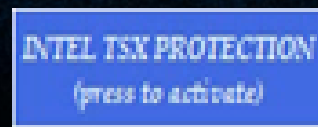
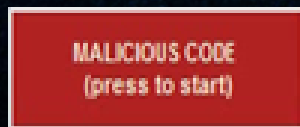
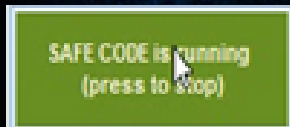
- 1) No RAM changes are reversed (security thread had no writes)
- 2) ABORT handler @ `abort_offset1` determines the nature of abort & takes necessary action

Live Demo

- Runs on a real TSX-capable notebook (vPro)
- The timeline is expanded for human eye to see
- Two systems



- Controls (big buttons)



Potential Applications

- Memory monitoring and protection (VMM, OS vendors)
 - Protecting **kernel code/data** (like Microsoft's PatchGuard)
 - Reverting unwanted memory changes in Guest OS (**hypervisor**)
 - Protecting critical data (e.g. **System Call Table**)
- Software self-protection (ISVs)
 - Can detect patching of software
 - Raises the bar for attackers



Advantages

- Creates a hardware-supported short-lived memory sandbox defined by RAM reads
 - Works equally well with physical and virtual RAM
- RAM granularity is better than pages (64 bytes vs 4k)
- Instant reaction to RAM modification
 - No window for malicious code to disable security
- Automatic roll-back of RAM changes is possible



Challenges



- Aborts due to the OS activities
 - Context switches (practically any API)
 - Interrupts create “aborts noise”
 - Requires short threads or thread management
- Injecting XBEGIN into a malicious thread (e.g. via hypervisor)
- Malware operating in kernel (or in hypervisor) may stop or modify the TSX security thread
- DoS attacks on TSX buffers (32kb+ per physical core)
 - Bogus transactions may create random & frequent capacity aborts
 - TSX buffers are susceptible to cache attacks

Memory Monitoring Approaches



	Tight RAM reading loop	Page-based exceptions	TSX RTM
Implementation	Software	Hardware	Hardware
Granularity	1 byte (or less)	4k	64 bytes
Intercept level	Hypervisor, OS, user mode	Hypervisor, OS	Hypervisor, OS, user mode
Speed	Slow	Medium	Fast
RAM coverage	Any amount	Any amount	Small amount
Response	Delayed	Instant	Instant
Time span	Unlimited	Unlimited	Short

Conclusions

- Sensitive, unusual and novel security method
- There are quirks
- A bit of a Spider Goat!
- Further research is ongoing...



