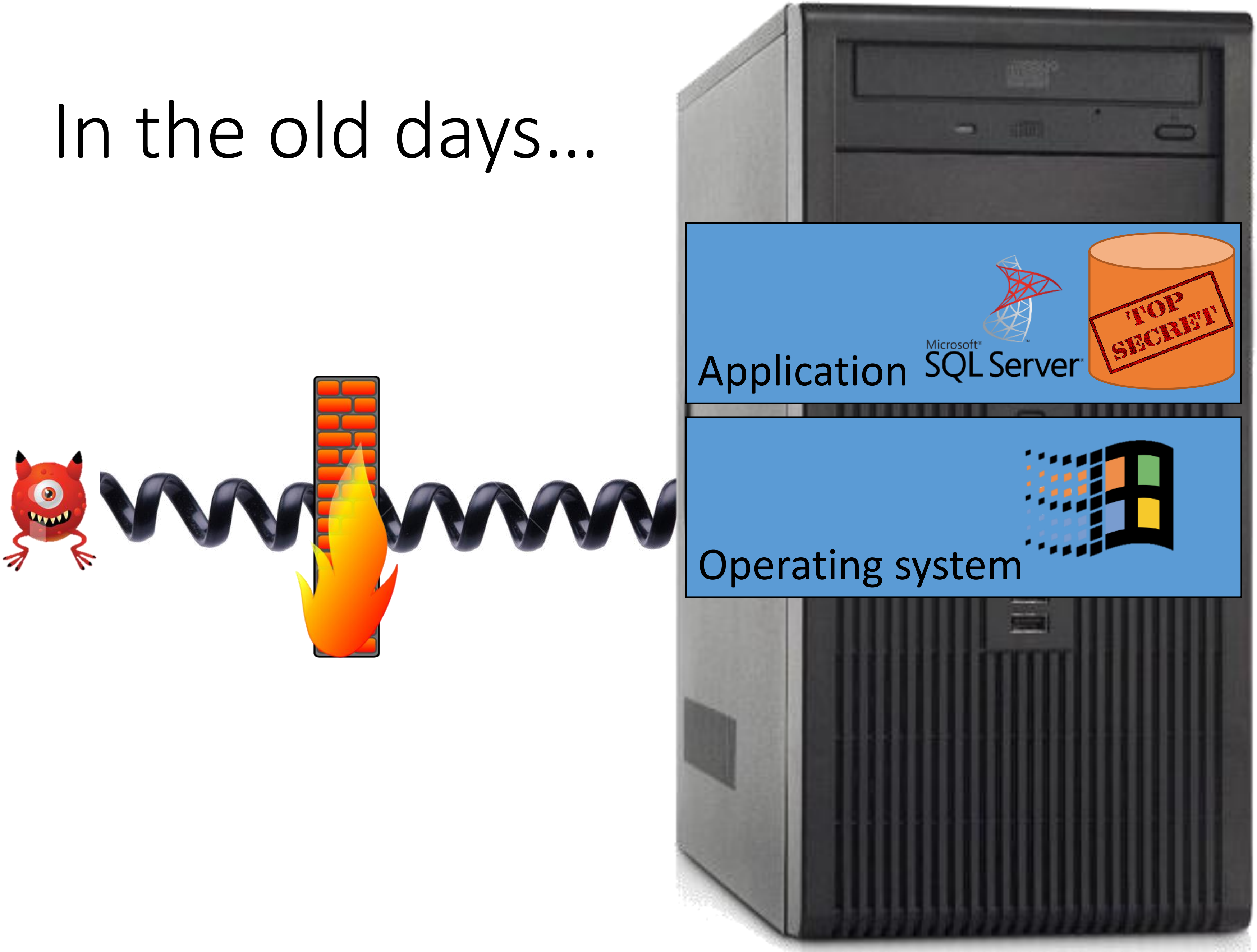
Haven

Shielding applications from an untrusted cloud

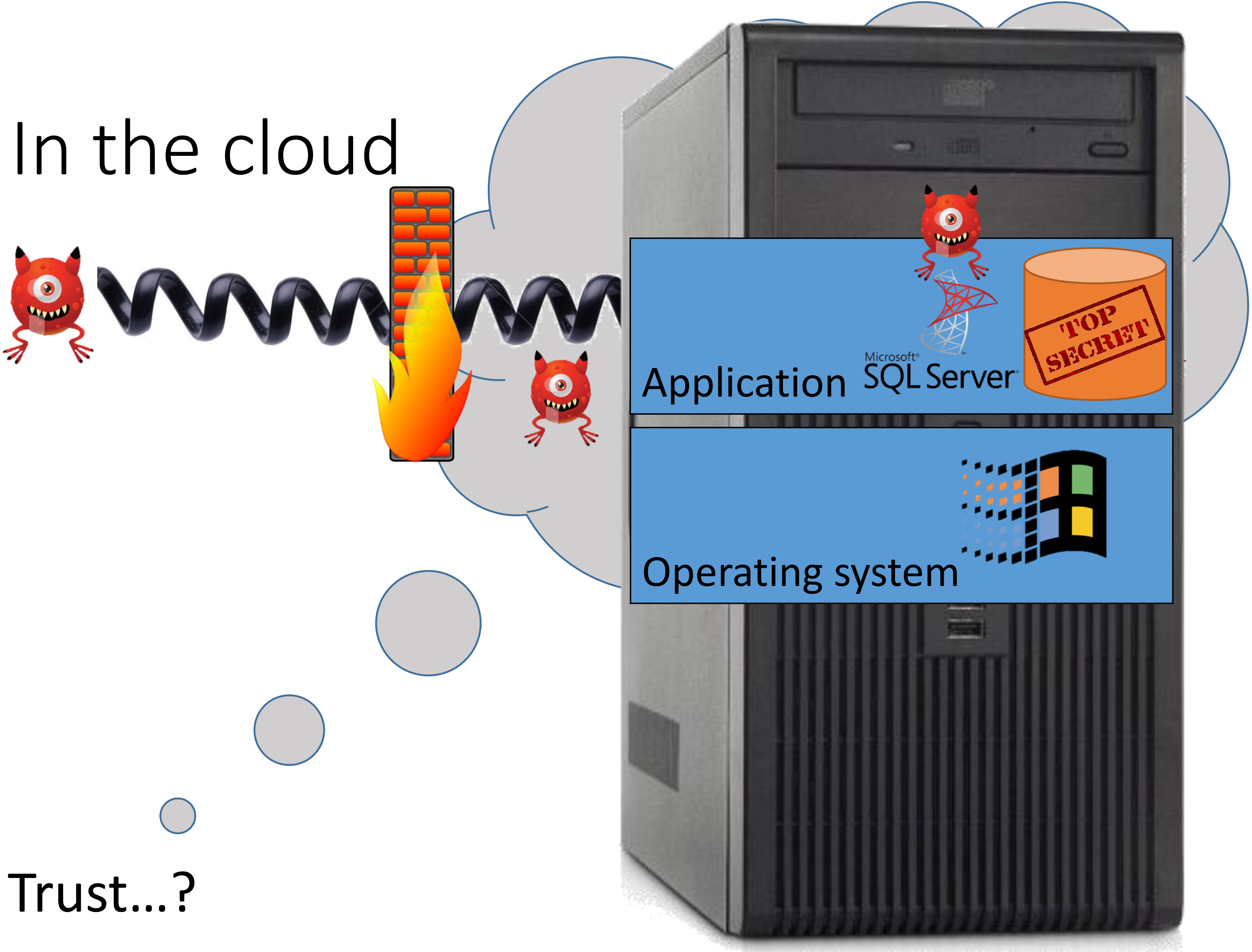
Andrew Baumann Marcus Peinado Galen Hunt

Microsoft Research

In the old days...



In the cloud



Trust…?

Hypervisor vulnerabilities are real

- November ’13: Privilege escalation in Hyper-V
- October ’14: Xen guest may read other VM’s data
- May ’15: “Venom” privilege escalation in Xen, KVM
- …

Our goals for Haven

Secure, private execution

of unmodified applications

(bugs and all)

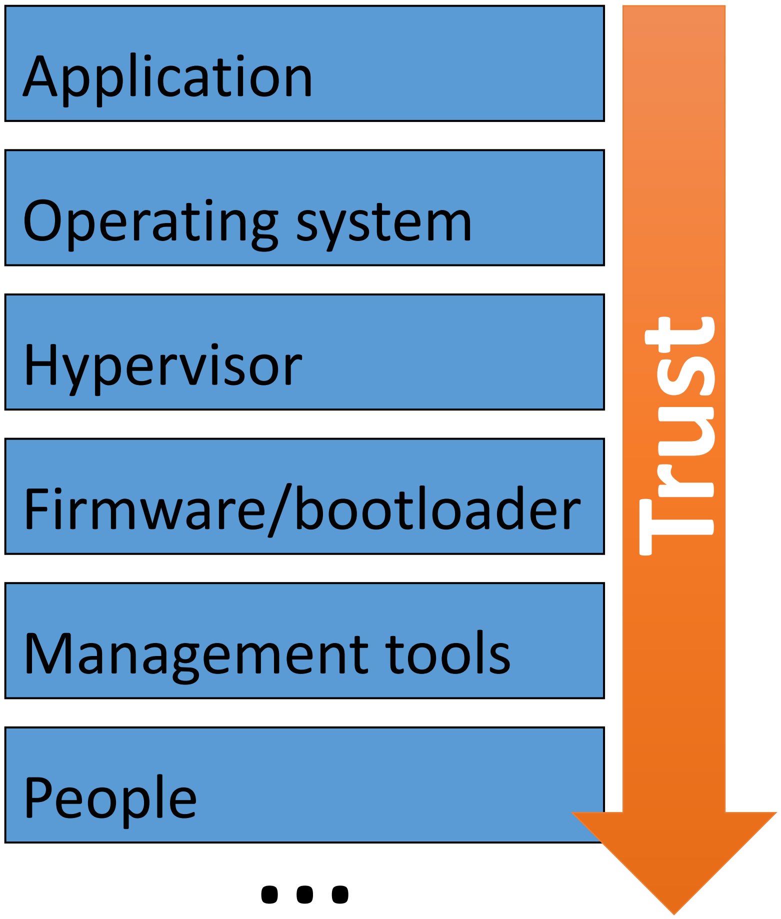
in an untrusted cloud

on commodity hardware

(Intel SGX)

Can you trust the cloud?

- Huge trusted computing base
  - Privileged software
    Hypervisor, firmware, ...
  - Management stack
  - Staff
    Sysadmins, cleaners, security, ...
  - Law enforcement
- Hierarchical security model
  - Observe or modify any data
  - Even if encrypted on disk / net



Current approaches

Hardware Security Modules

- Dedicated crypto hardware
  - Expensive
- Limited set of APIs
  - Key storage
  - Crypto operations
- Protects the “crown jewels”, not general-purpose

Trusted hypervisors

- Hardware root of trust (e.g., TPM or TrustZone)
- Small, secure, hypervisor
  - Multiplexes hardware
  - Ensures basic security, such as strong isolation

Problem #1: system administrators

Problem #2: physical attacks (e.g. memory snooping)

Problem #3: tampering with hypervisor ✓

Remote attestation

- For example, using a TPM chip
- Basic idea:
  - Signed measurement (hash) of privileged software
  - Remote user checks measurement
  - Incorrect attestation → compromised software
- Problem: what is the expected measurement?
  - Cloud provider applies patches and updates
  - Must trust provider for current hash value

What do we really want?

Secure colo provides:

Power and cooling

Network access

Raw resources

Untrusted I/O

Shielded execution

- Protection of specific program from rest of system
  - cf. protection, isolation, sandboxing, etc.
  - New term (older concept)
- Program unmodified, naïve to threats
- Confidentiality and integrity of:
  - The program
  - Its intermediate state, control flow, etc.

  → Input and output may be encrypted
- Host may deny service, cannot alter behaviour

Threat model

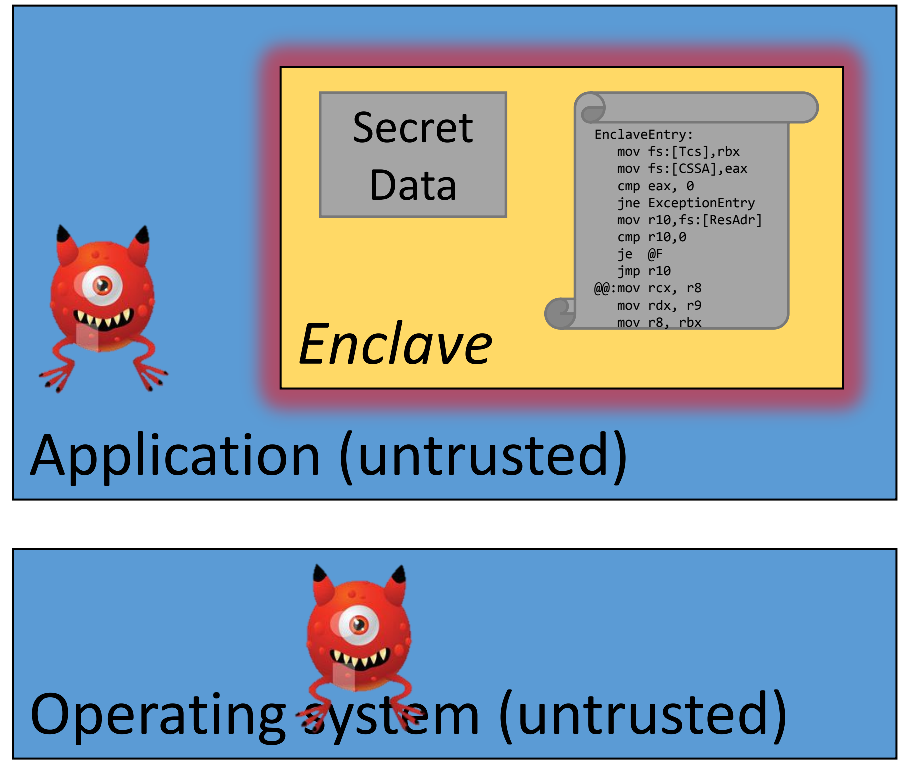
- **We assume a malicious cloud provider**
  - Convenient proxy for real threats
- All the provider’s software is malicious
  - Hypervisor, firmware, management stack, etc.
- All hardware besides the CPU is untrusted
  - DMA attacks, DRAM snooping, cold boot

- We do not prevent:
  - Denial-of-service (don’t pay!)
  - Side-channel attacks (open problem)

Background: Intel SGX

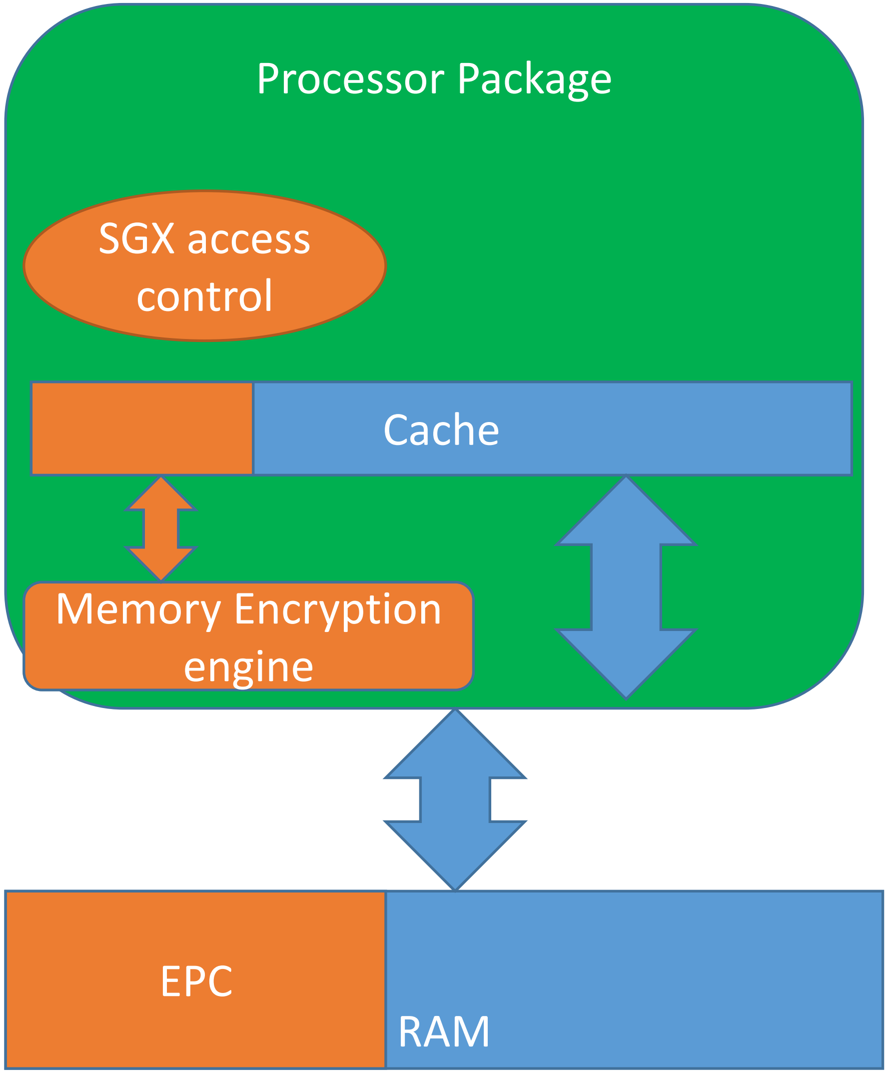
Intel SGX

- Hardware isolation for an *enclave*
  - New instructions to establish, protect
  - Call gate to enter
- Remote attestation

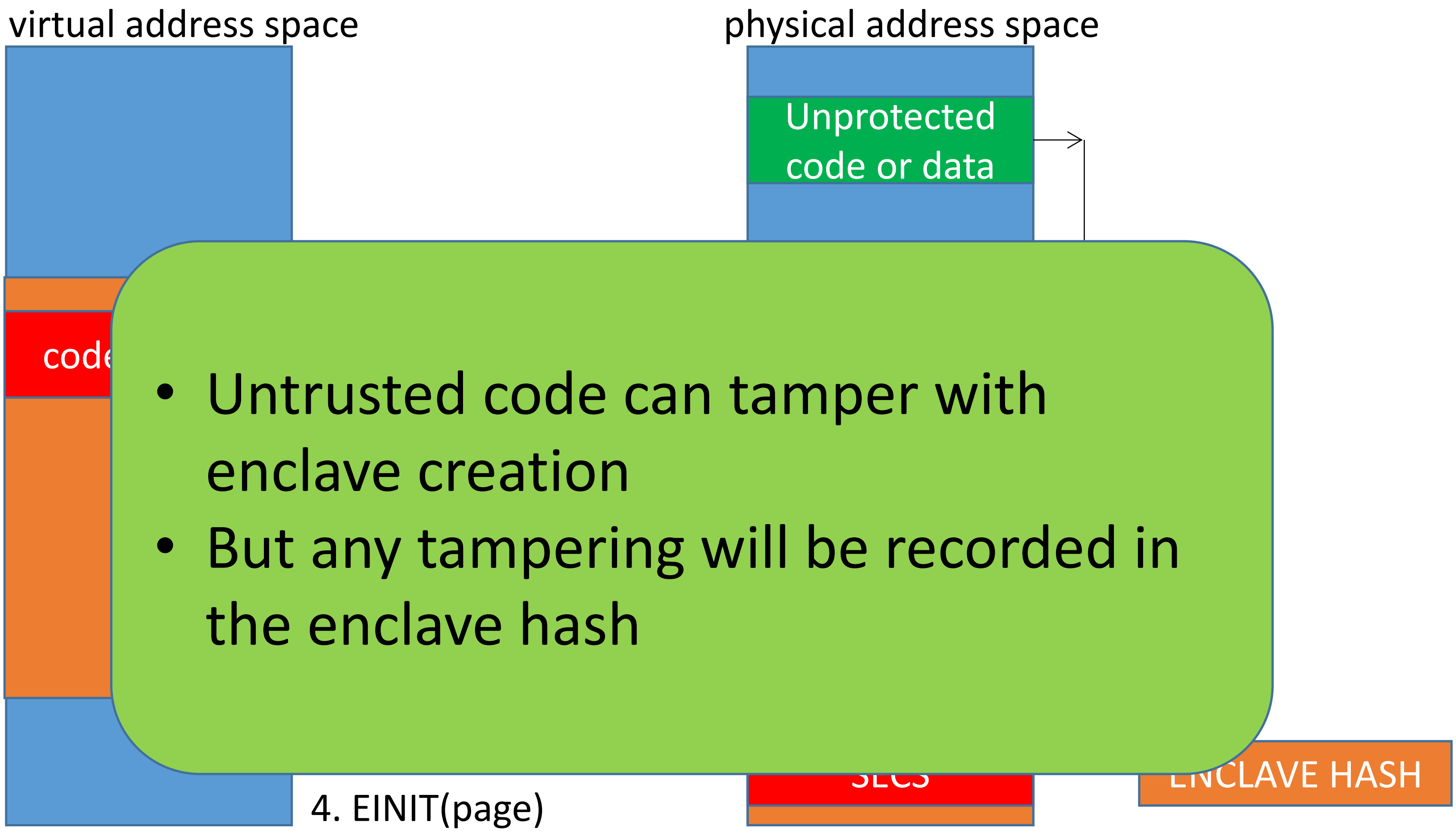


Enclave Memory

- Processor designates physical memory range as EPC memory
  - Specified by BIOS at boot time.
- EPC RAM is encrypted and integrity protected.
  - Applied by processor as cache lines travel between the LLC and RAM
  - RAM and memory buses are now outside the HW TCB.
- SGX access controls protect enclave memory inside the processor.
  - Only code running in an enclave can access this enclave's memory.



Building an Enclave



- Untrusted code can tamper with enclave creation
- But any tampering will be recorded in the enclave hash

Executing an Enclave

- EENTER: jumps to a fixed enclave address
    - Defined during enclave construction
- EEXIT: jumps to any address outside the enclave
- Asynchronous exit due to interrupts, exceptions etc.
    - Save and scrub processor state
- ERESUME: Resume enclave execution after an asynchronous exit.

Other features

- Sealed storage
  - EGETKEY: Enclave can obtain persistent keys as a function of its enclave hash or author

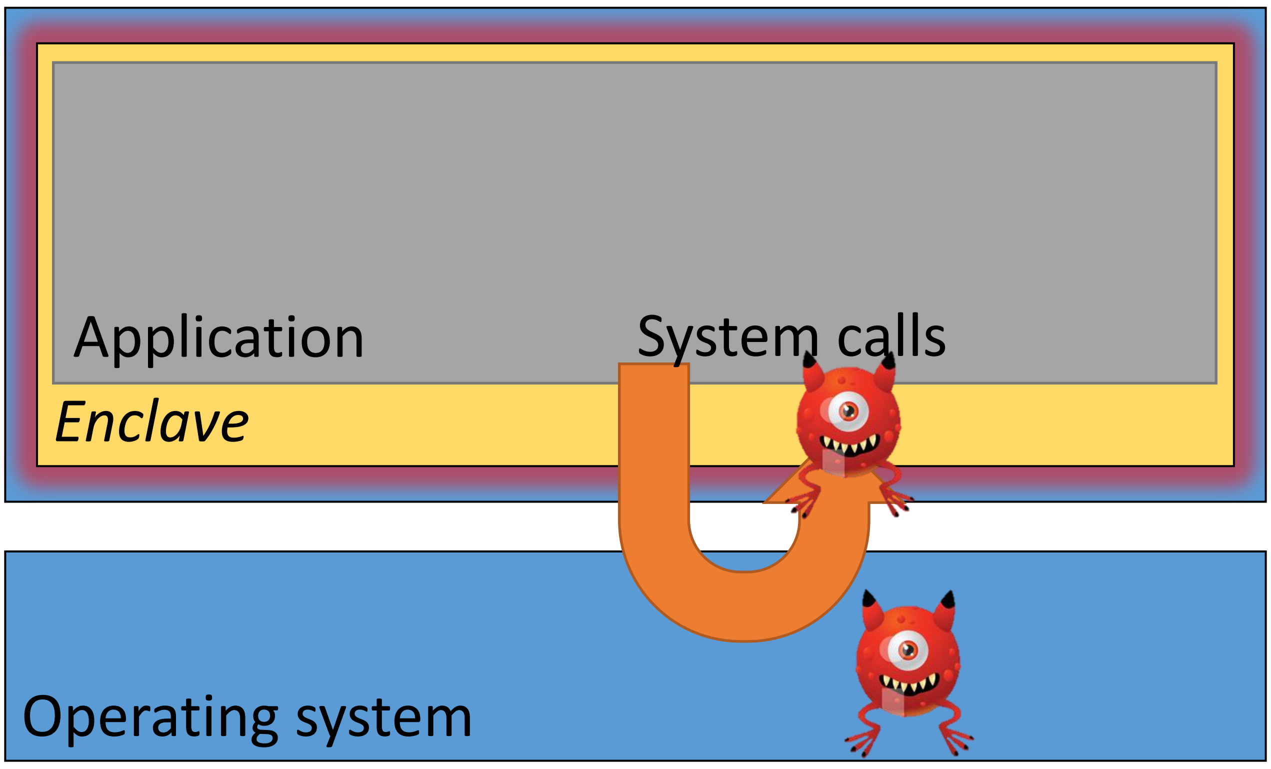
- Attestation
  - EPID group signature scheme
  - Implemented in a special “Quoting Enclave”

SGX: what's new?
(over prior trusted hardware)

- Doesn't rely on any trusted software
  - *Untrusted* OS performs scheduling/multiplexing
  - Paging support
  - (Practically) unlimited number of distrusting enclaves
- Hardware TCB = CPU package
  - Encrypted and integrity-protected RAM
  - CPU-based attestation
  - High level of physical security

Haven Design

Design challenge: Iago attacks



Iago attacks

- `malloc()` returns pointer to user's stack
- Scheduler allows two threads to race in a mutex
- System has 379,283 cores and -42MB of RAM
- `read()` fails with EROFS
- …

Our approach:

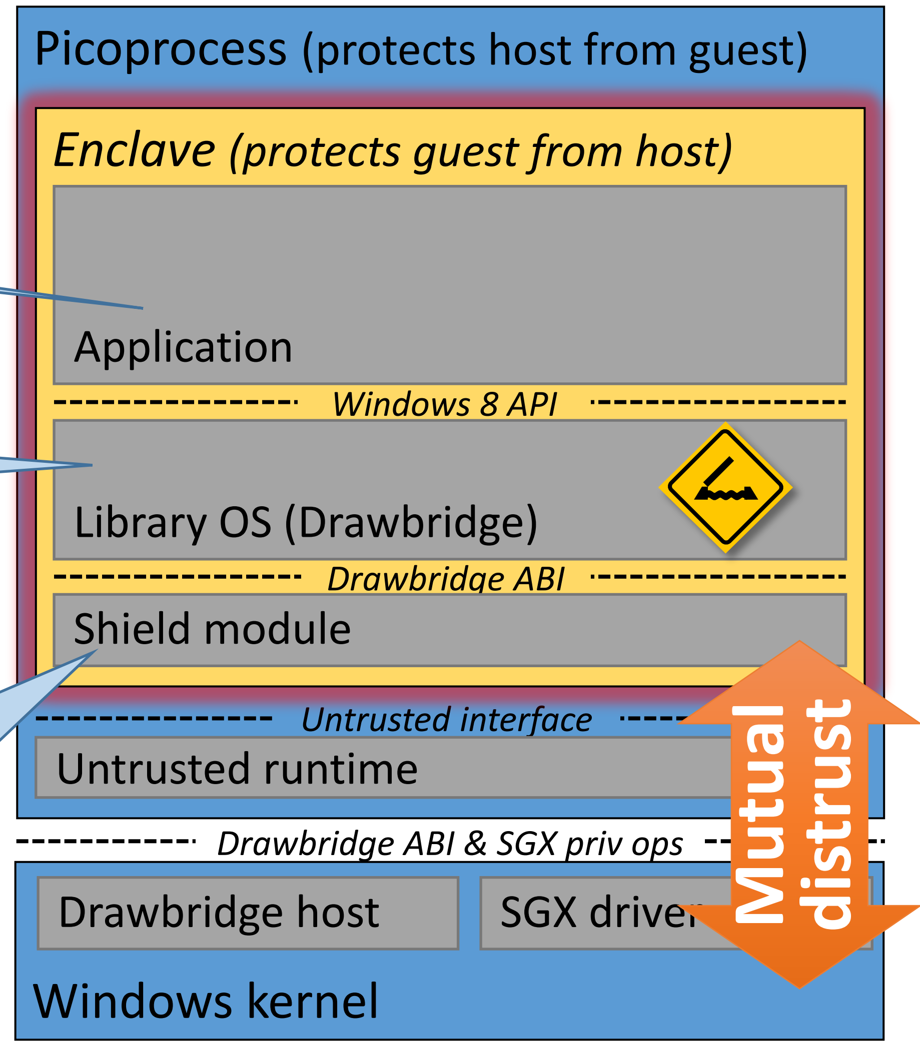
- Don't try to check them all
- Admit OS into trusted computing base

Haven

- Unmodified binaries

- Subset of Windows, enlightened to run in-process

- Shields LibOS from Iago attacks
- Includes typical kernel functionality
  - Scheduling, VM, file system
- Untrusted interface with host

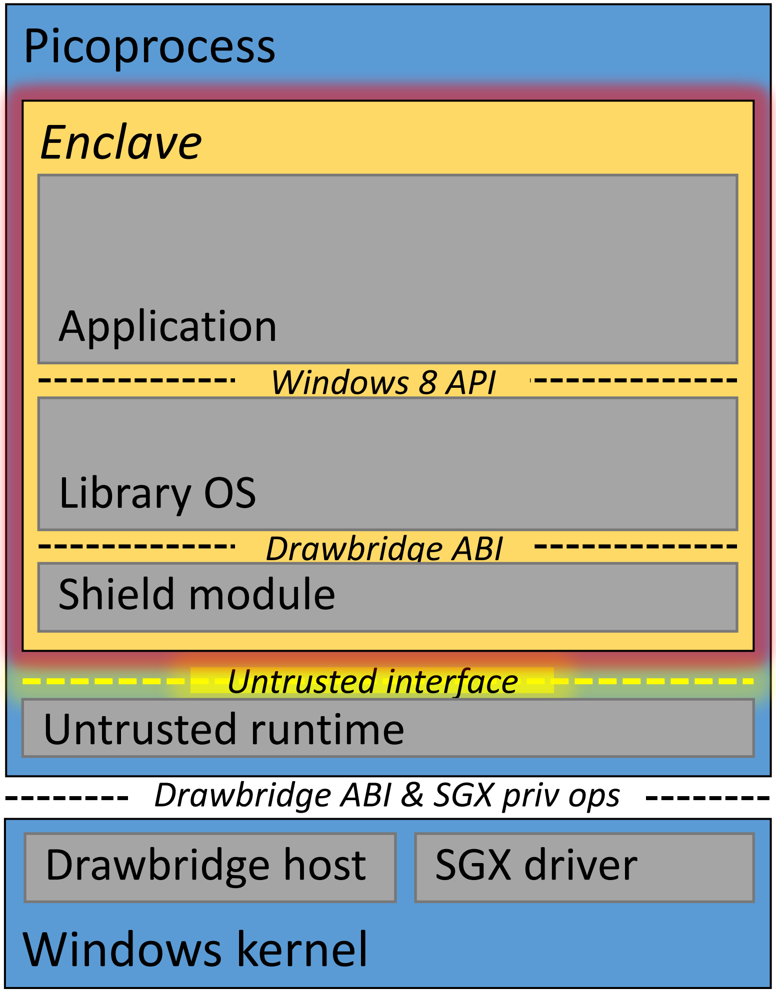


Untrusted interface

- Host/guest mutual distrust
- Policy/mechanism with a twist
  - Virtual resource policy in guest

    Virtual address allocation, threads
  - Physical resource policy in host

    Physical pages, VCPUs
- ~20 calls, restricted semantics



Untrusted interface

Upcalls:

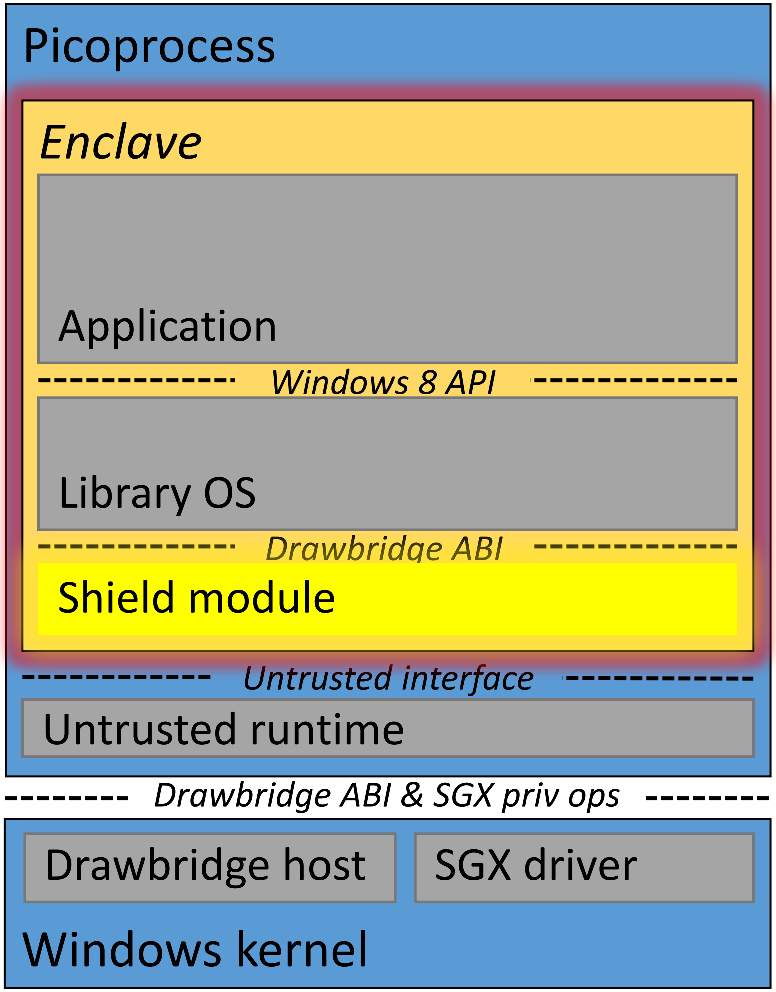
```
ExceptionDispatch(ExceptionInfo)
ThreadEntry()
```

Downcalls:

```
AsyncCancel(AsyncHandle)
AsyncPoll(AsyncHandle) → Results
DebugStringPrint(Message)
EventClear(EventHandle)
EventSet(EventHandle)
ObjectClose(Handle)
ObjectsWaitAny(Num, Handles, Timeout) → Idx
ProcessExit(ExitCode)
StreamAttributesQueryByHandle(StreamHandle) → Attrs
StreamFlush(StreamHandle)
StreamGetEvent(StreamHandle, EventId) → EventHandle
StreamOpen(URI, Options) → StreamHandle
StreamRead(StreamHandle, Off, Sz, Bf) → AsyncHandle
StreamWrite(StreamHandle, Off, Sz, Bf) → AsyncHandle
SystemTimeQuery() → Time
ThreadCreate(Tcs) → ThreadHandle
ThreadExit()
ThreadInterrupt(ThreadHandle)
ThreadYieldExecution()
VirtualMemoryCommit(Addr, Size, Prot)
VirtualMemoryFree(Addr, Size)
VirtualMemoryProtect(Addr, Size, Prot)
```

Shield module

- Memory allocator, region manager
  - Host commits/protects specific pages
  - No address allocation
- Private file system
  - Encrypted, integrity-protected VHD
- Scheduler
  - Don’t trust host to schedule threads
- Exception handler
  - Emulation of some instructions
- Sanity-check of untrusted inputs
  - Anything wrong → panic!
- 23 KLoC (half in file system)



SGX limitations

1. Dynamic memory allocation and protection
   - New instructions needed
2. Exception handling
   - SGX d … enclave
3. Permi…
   - RDTSC … erformance
4. Thread-local storage
   - Can’t reliably switch FS and GS

Good news!

These are fixed in SGX v2

Performance evaluation

- Implemented and tested using SGX emulator
  - Thanks, Intel!
- Problem: no SGX implementation yet
- Solution: model for SGX performance

1. TLB flush on Enclave crossings
2. Variable spin-delay for critical SGX instructions
   - Enclave crossings
   - Dynamic memory allocation, protection
3. Penalty for access to encrypted memory
   - Slow overall system DRAM clock

Performance summary

- Depends on model parameters, details in paper
- 35% (Apache) – 65% (SQL Server) slowdown vs. VM
    - Assumes 10k+ cycles SGX instructions, 30% slower RAM
- … and you don't have to trust the cloud!

SGX wish-list

- Exception handling overhead is high
  - `IRET`, `ERESUME` require enclave exits
  - A single application exception (e.g. stack growth) results in two exceptions and *eight* enclave crossings
- Demand loading
  - Fixed in spec (EACCEPTCOPY), but we haven't tested it
- Shielded VMs
  - Everybody wants this
  - Not trivial: can't trap-and-emulate in hypervisor

What's next for Haven?

- Rollback of persistent storage
  - Requires more hardware, or more communication
- Untrusted time
  - Network time sync, RDTSC
- Cloud management
  - Suspend / migrate applications [Lorch et al, NSDI 2015]
  - Encrypted VLANs
- Side-channel defences
  - Open problem [Xu et al, IEEE Security & Privacy 2015]

Conclusion

- Haven is closer to a true “utility computing” model
  - Utility provides raw resources
  - Doesn’t care what you do with them
- Hope that SGX will be the first step in widespread hardware support for shielded execution

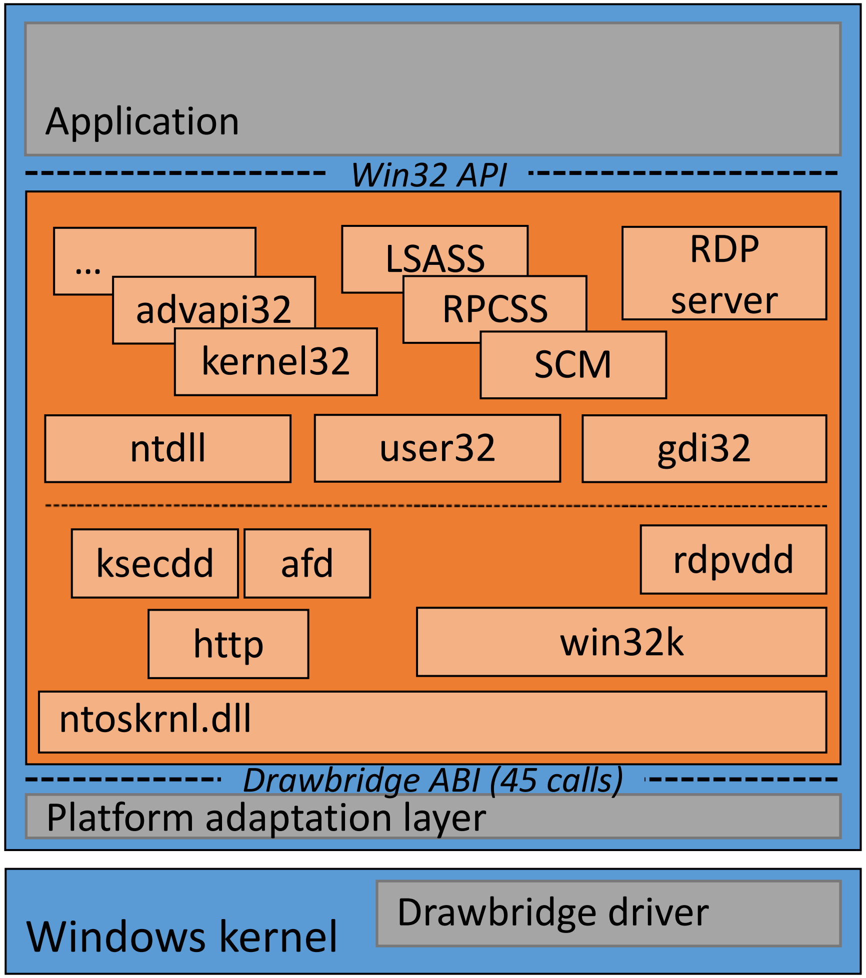
Backup

Related work

- TPM-based systems [Flicker, TrustVisor, Credo, Nexus, MiniBox]
  - Vulnerable to simple physical attacks
  - Typically relies on trusted hypervisor
  - Prohibitively expensive otherwise [Flicker]
- Protecting user memory from the OS [Overshadow, SP^3, CloudVisor, SecureME, InkTag, Virtual Ghost]
  - Relies on trusted hypervisor or compiler [Virtual Ghost]
  - Vulnerable to Iago attacks at syscall interface
- Homomorphic encryption
  - ✓ No hardware in the TCB
  - ✓ Suitable for some applications [CryptDB, Cipherbase, MrCrypt]
  - Intolerable overheads for general-purpose computation

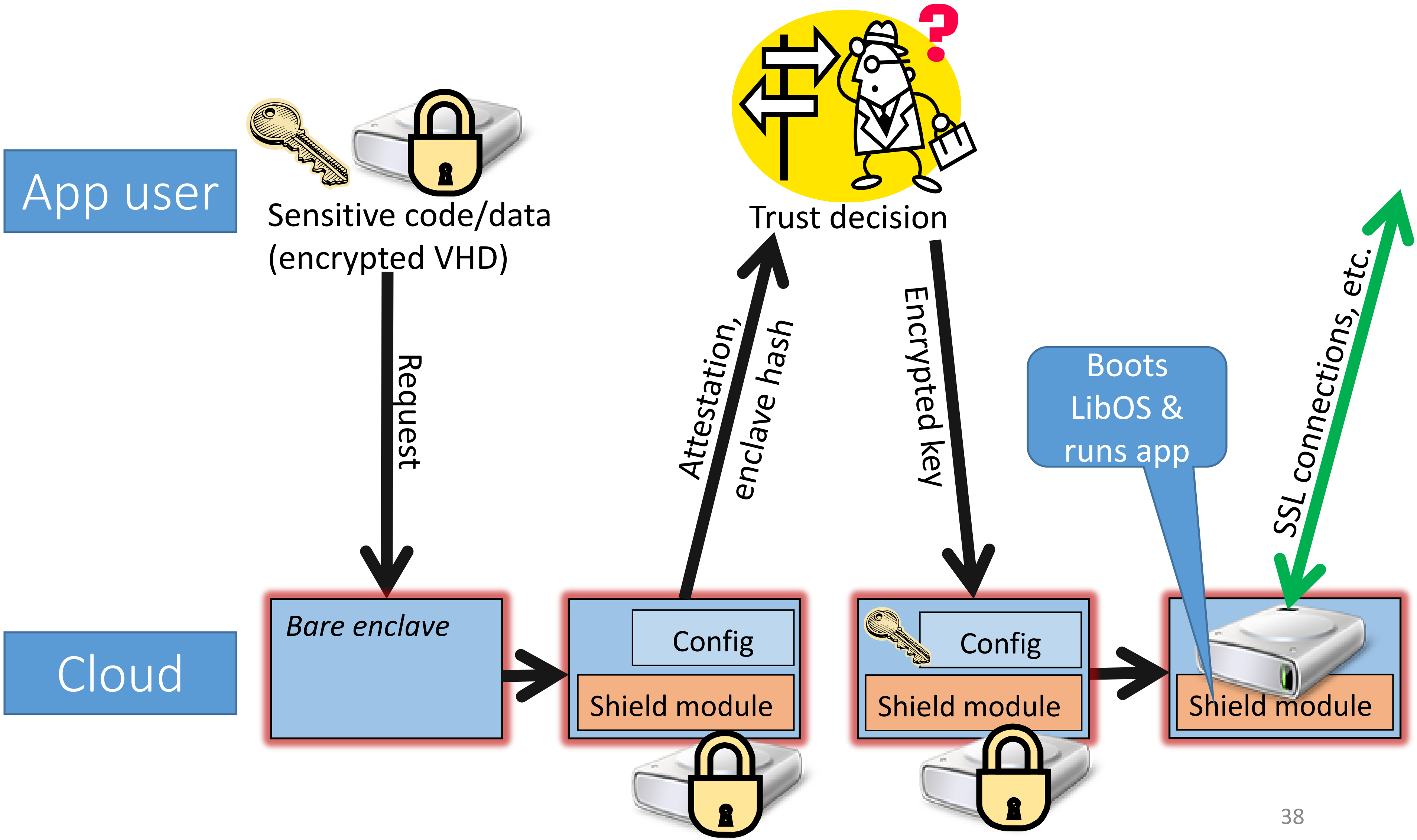
Background: Drawbridge

- Secure isolation of existing Windows applications
- **Picoprocess** for confinement
  - Secure isolation container
  - Low overhead (vs. a VM)
- **Library OS** for compatibility
  - Enlightened Windows
  - Strong app compatibility

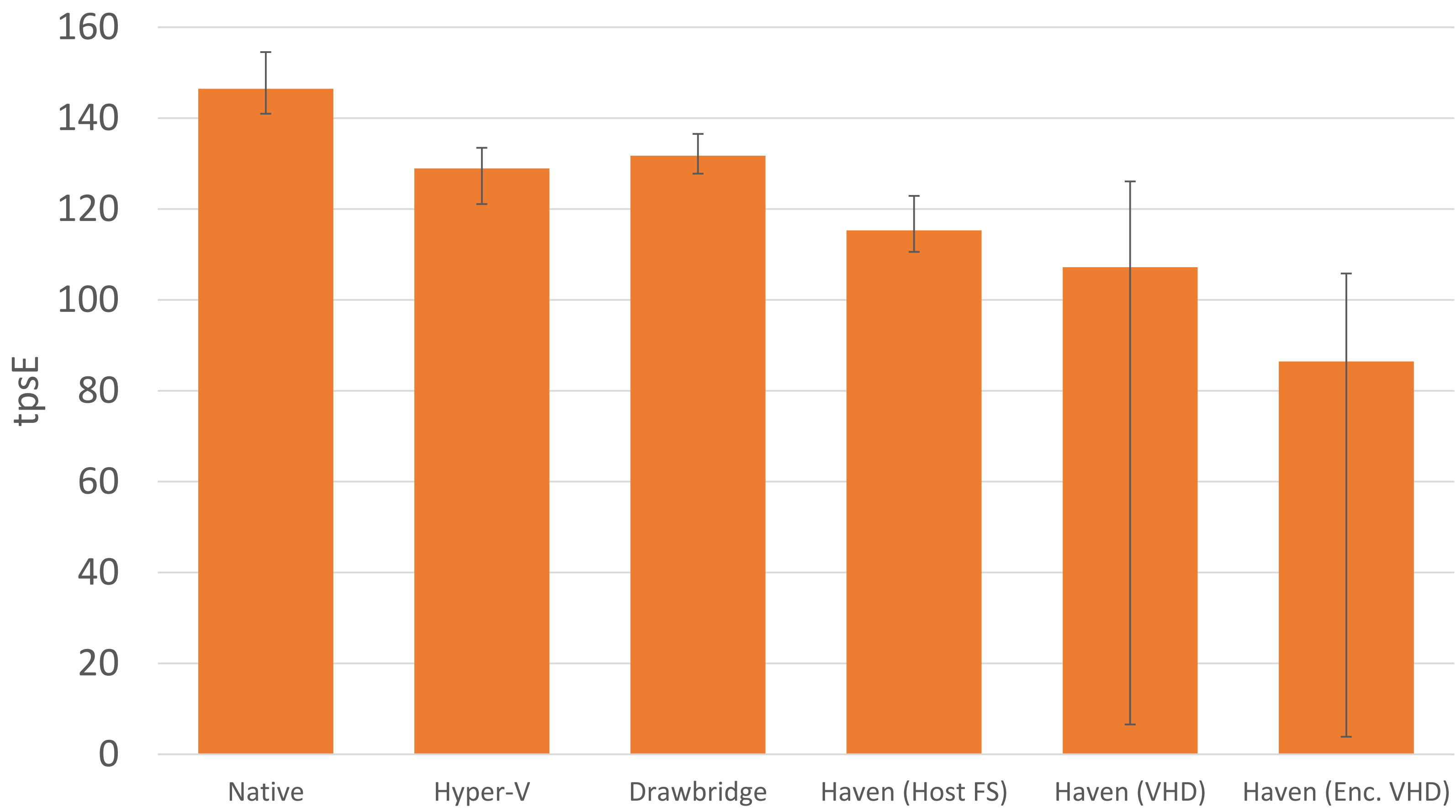


D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, G. C. Hunt, Rethinking the library OS from the top down, Proc. ASPLOS'11

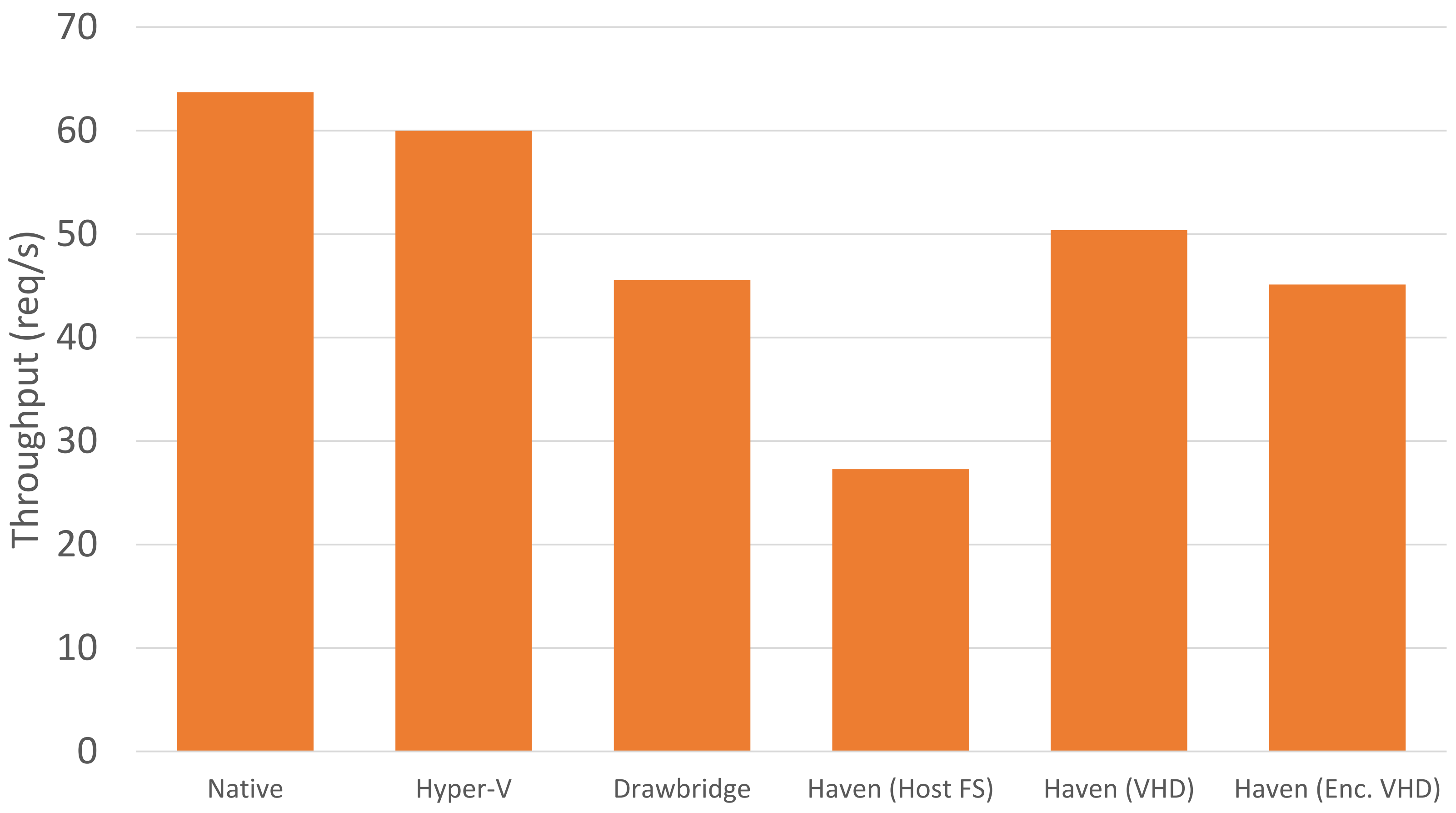
Provisioning



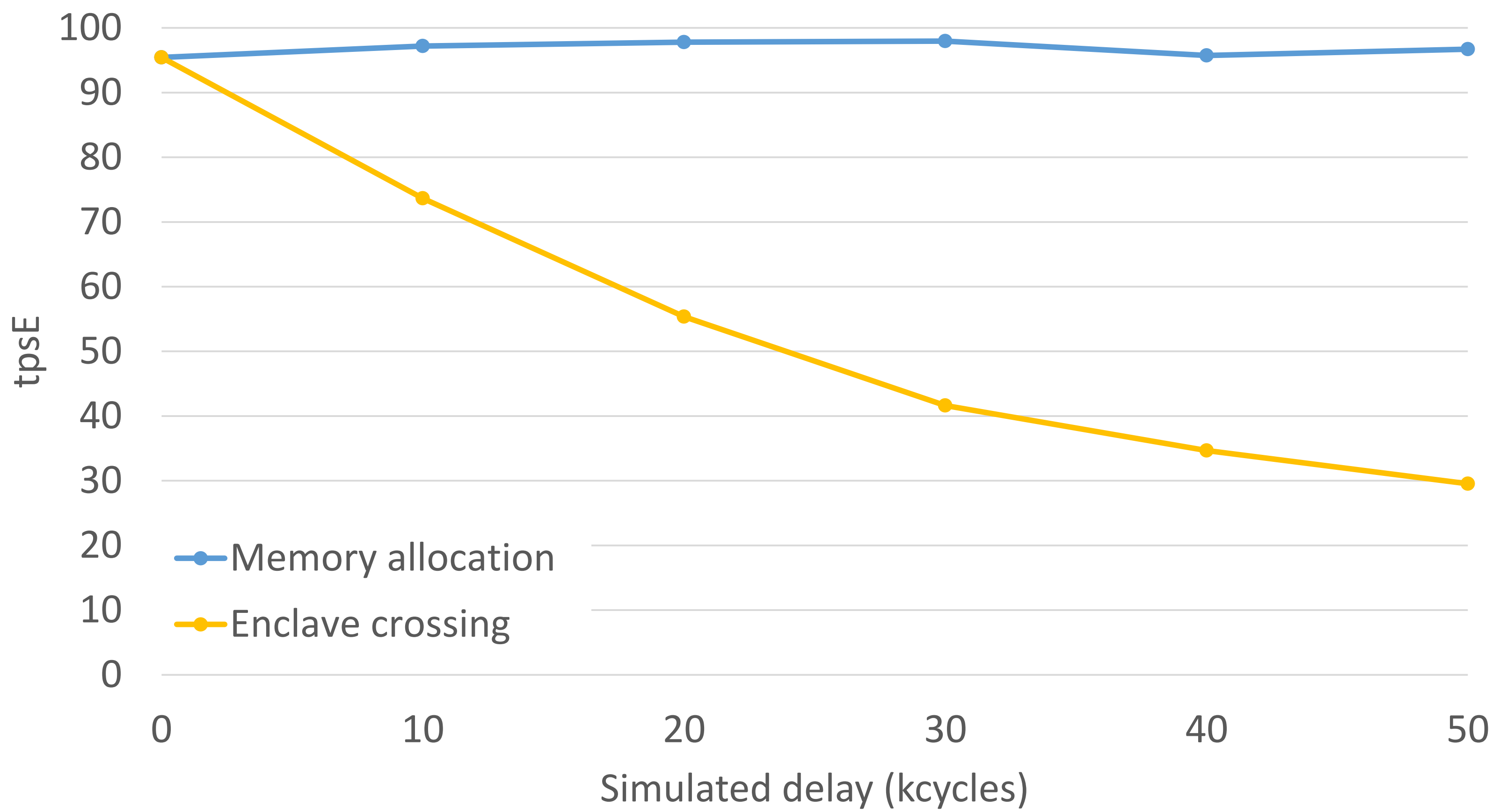
Performance: SQL Server



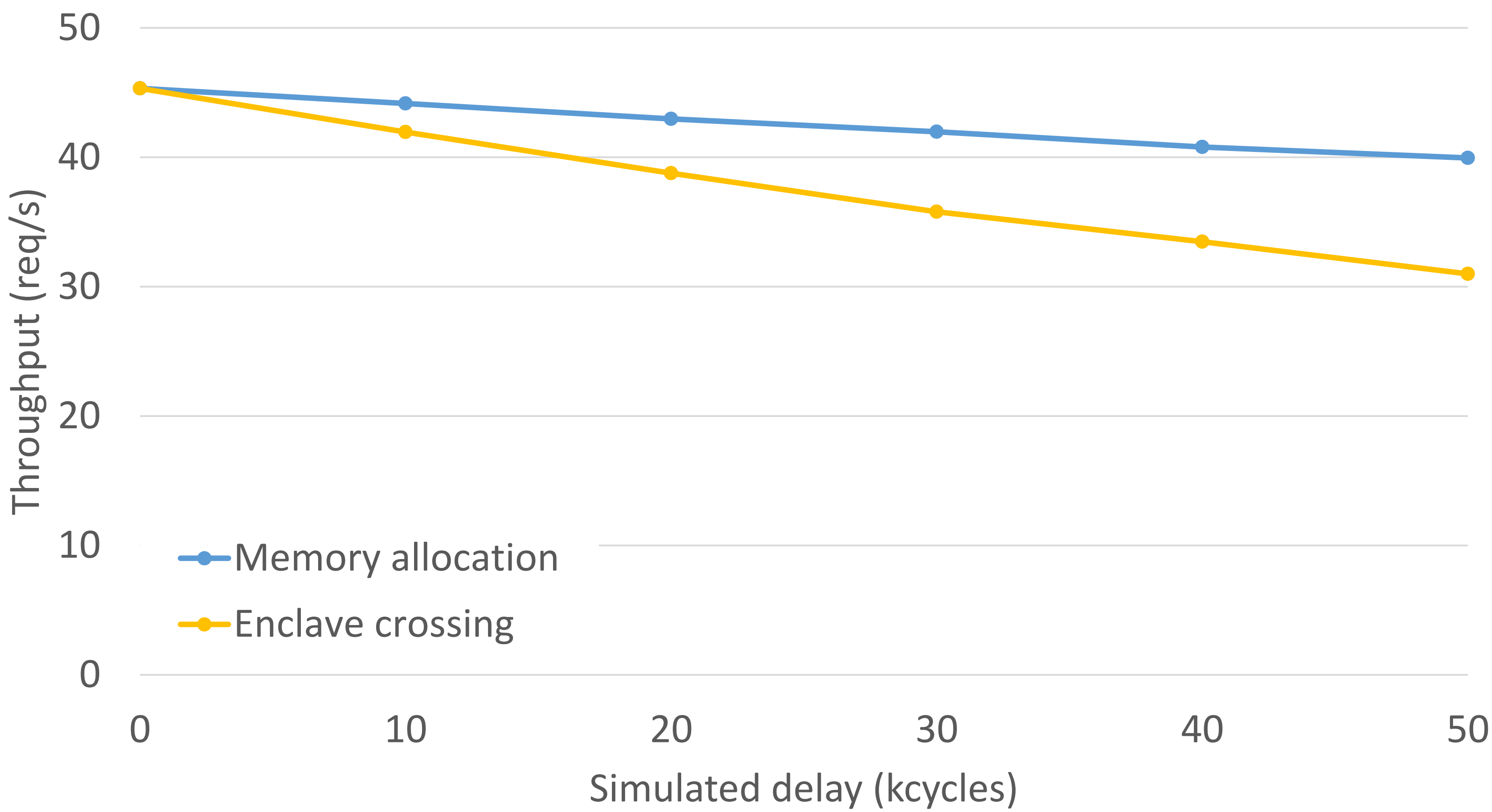
Performance: Apache/MediaWiki



Sensitivity to SGX cost: SQL Server



Sensitivity to SGX cost: Apache



Sensitivity to memory slowdown

- Hard to simulate: not many options
- Scaled down overall DRAM bandwidth and latencies by 33%
  - SQL Server TPC-E throughput reduced by 21%
  - Apache / MediaWiki throughput reduced by 7%
  - Over-estimate: some accesses would be outside the enclave