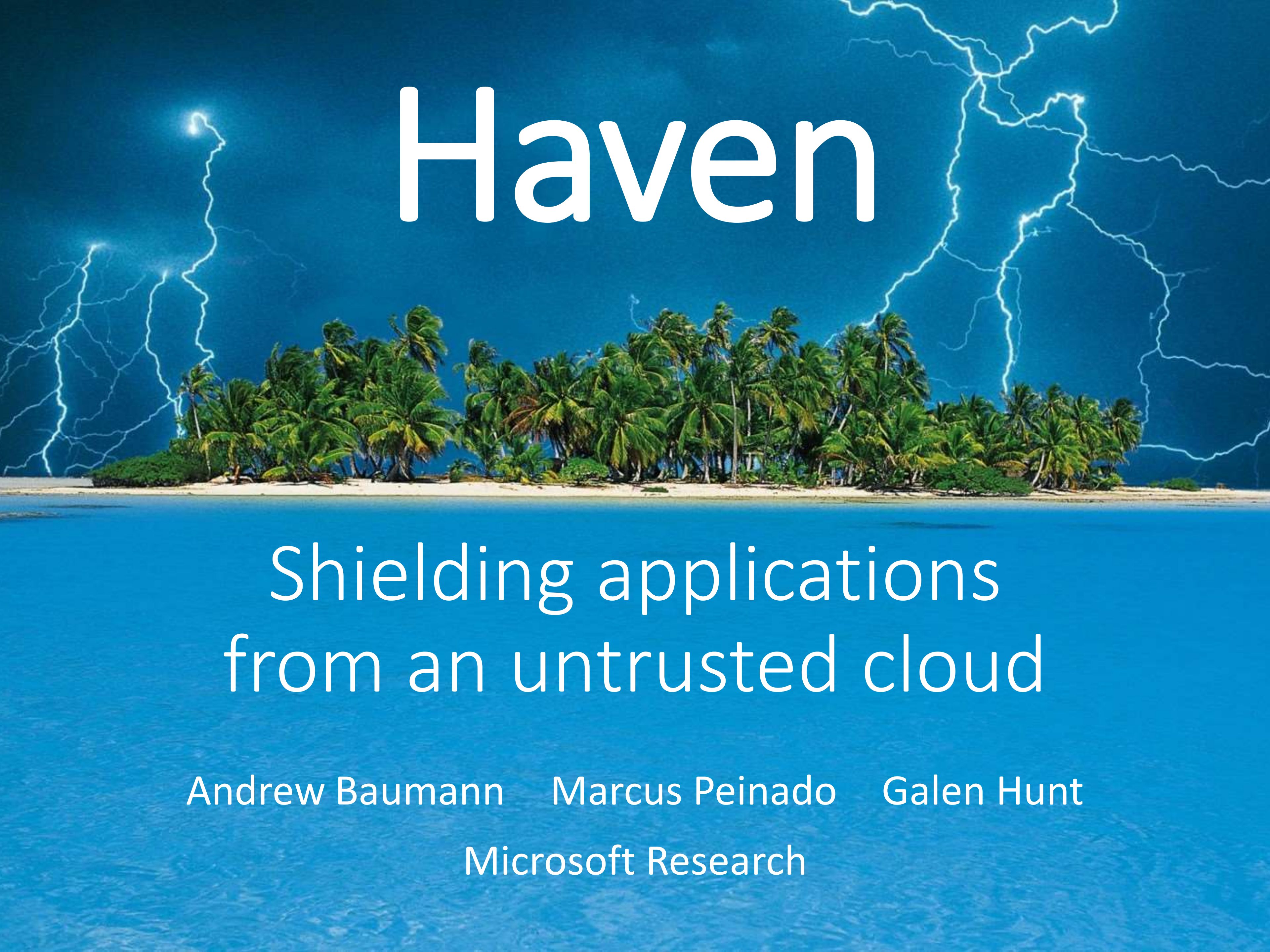


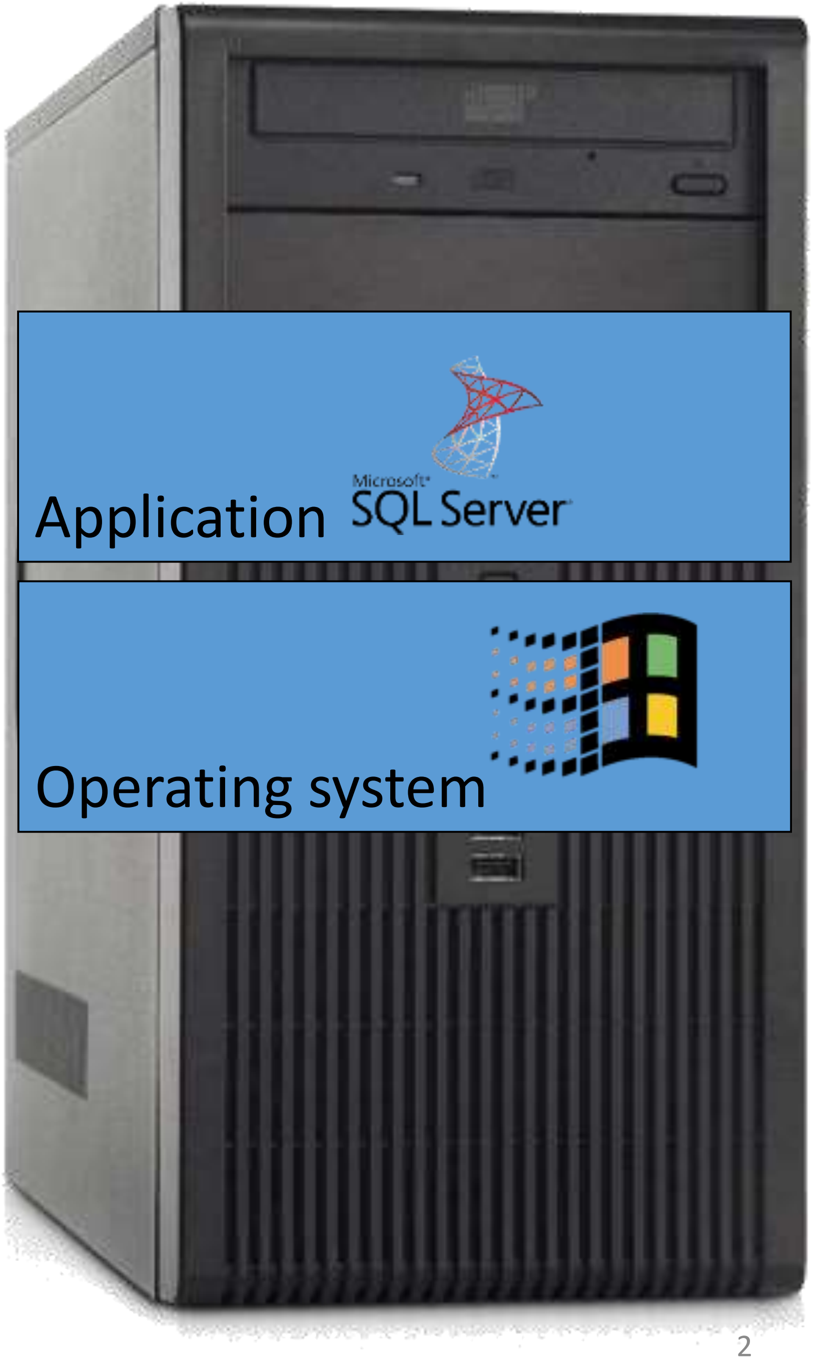
Haven

Shielding applications from an untrusted cloud

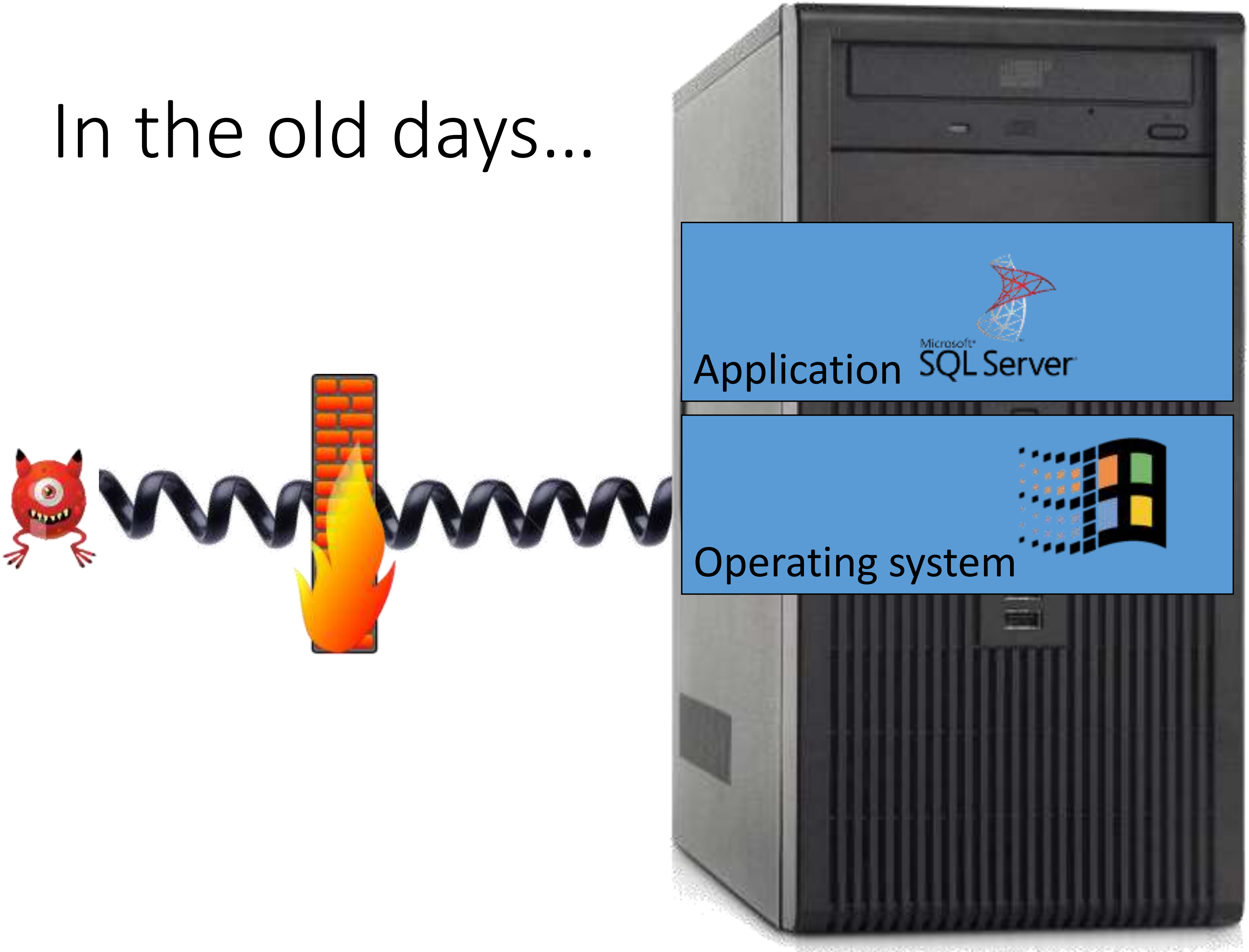
Andrew Baumann Marcus Peinado Galen Hunt

Microsoft Research

In the old days…



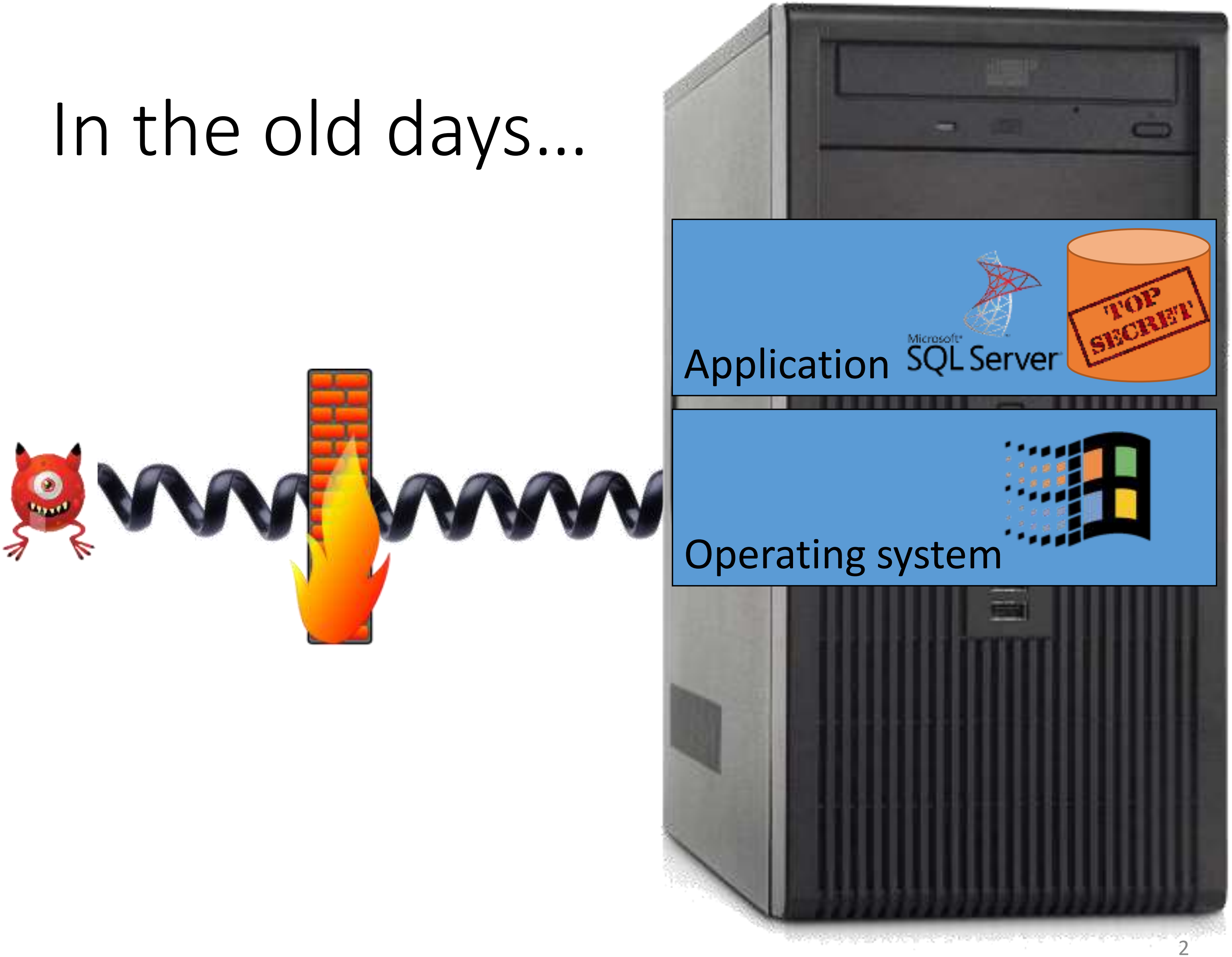
In the old days…



Application

Operating system

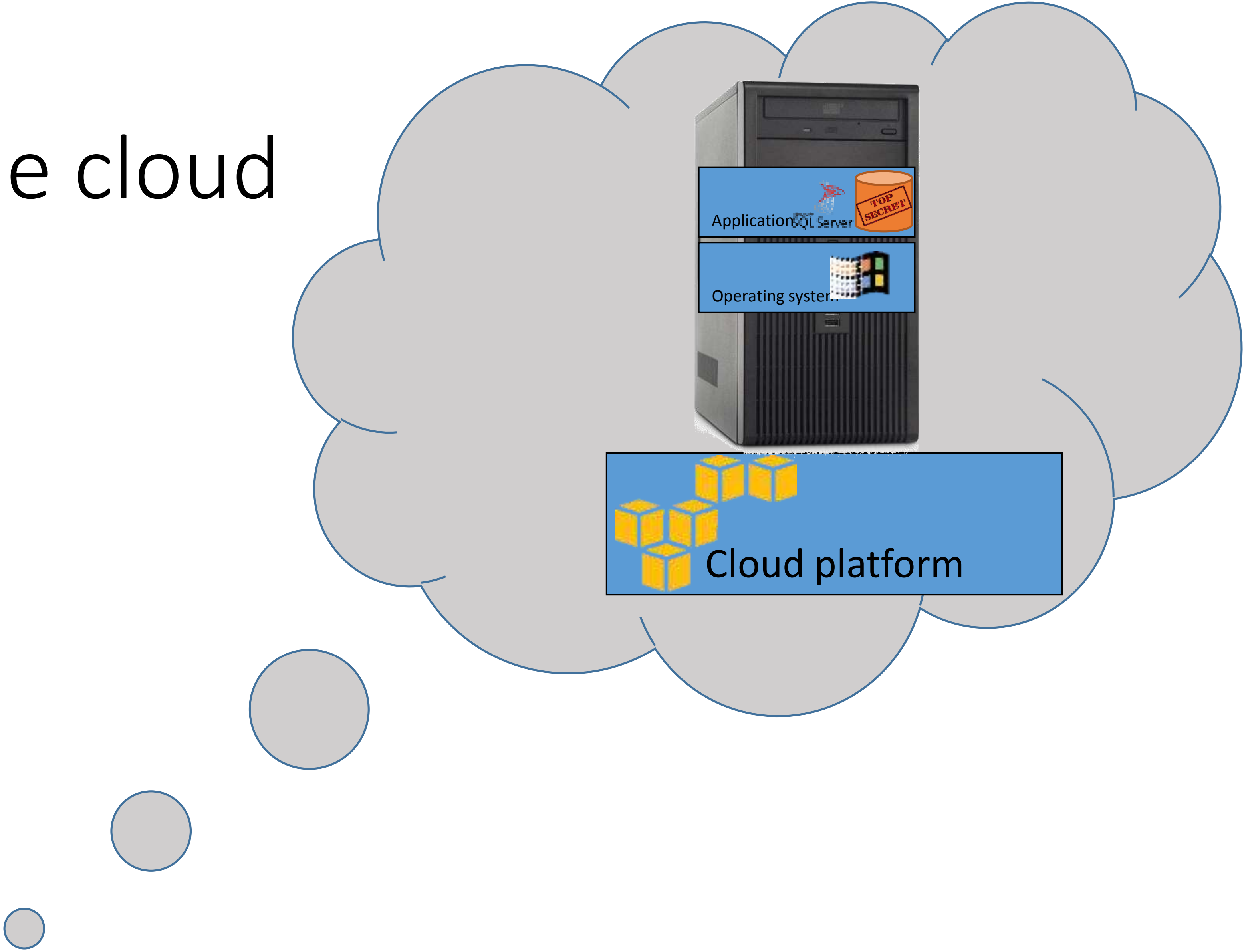
In the old days…



Application

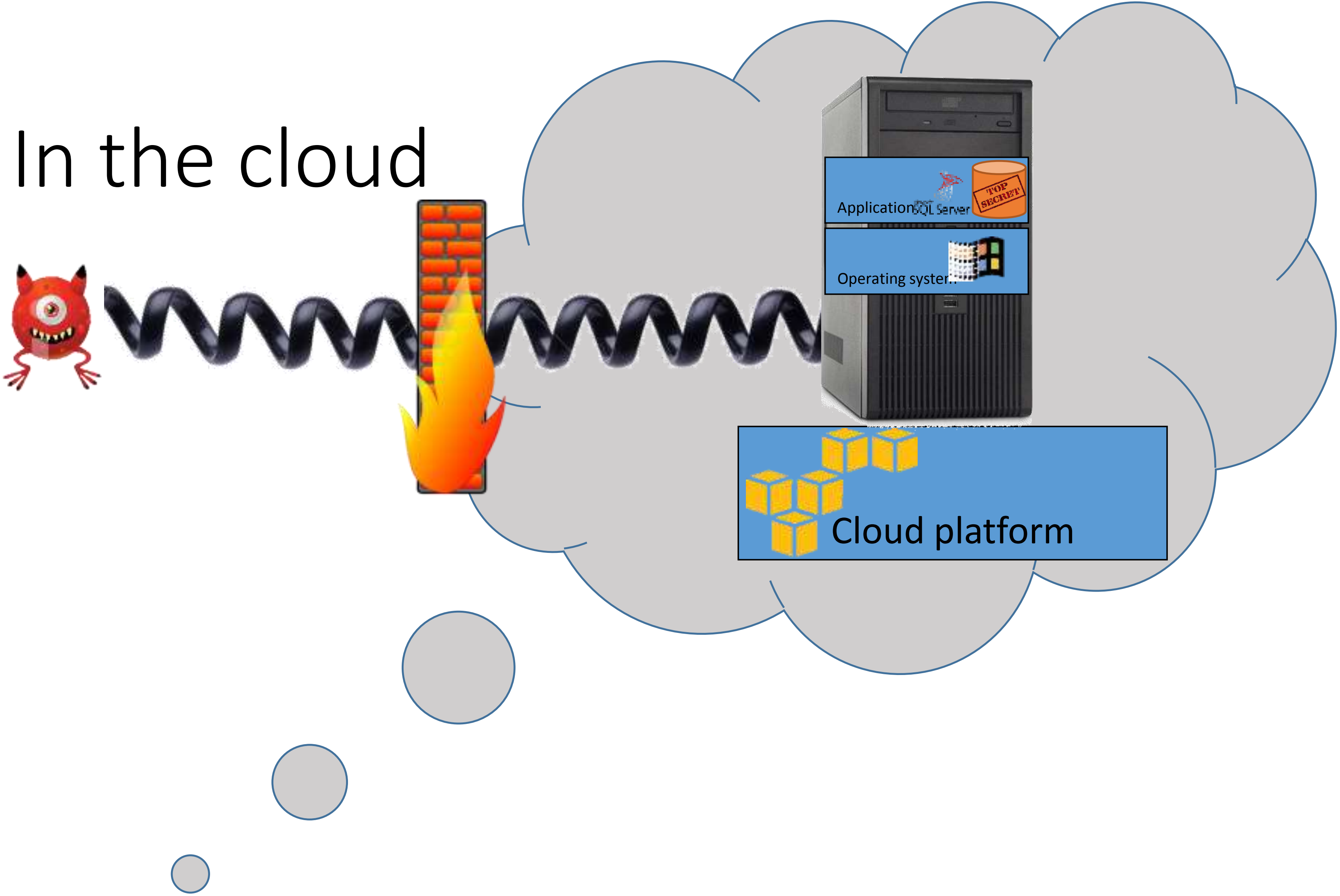
Operating system

In the cloud



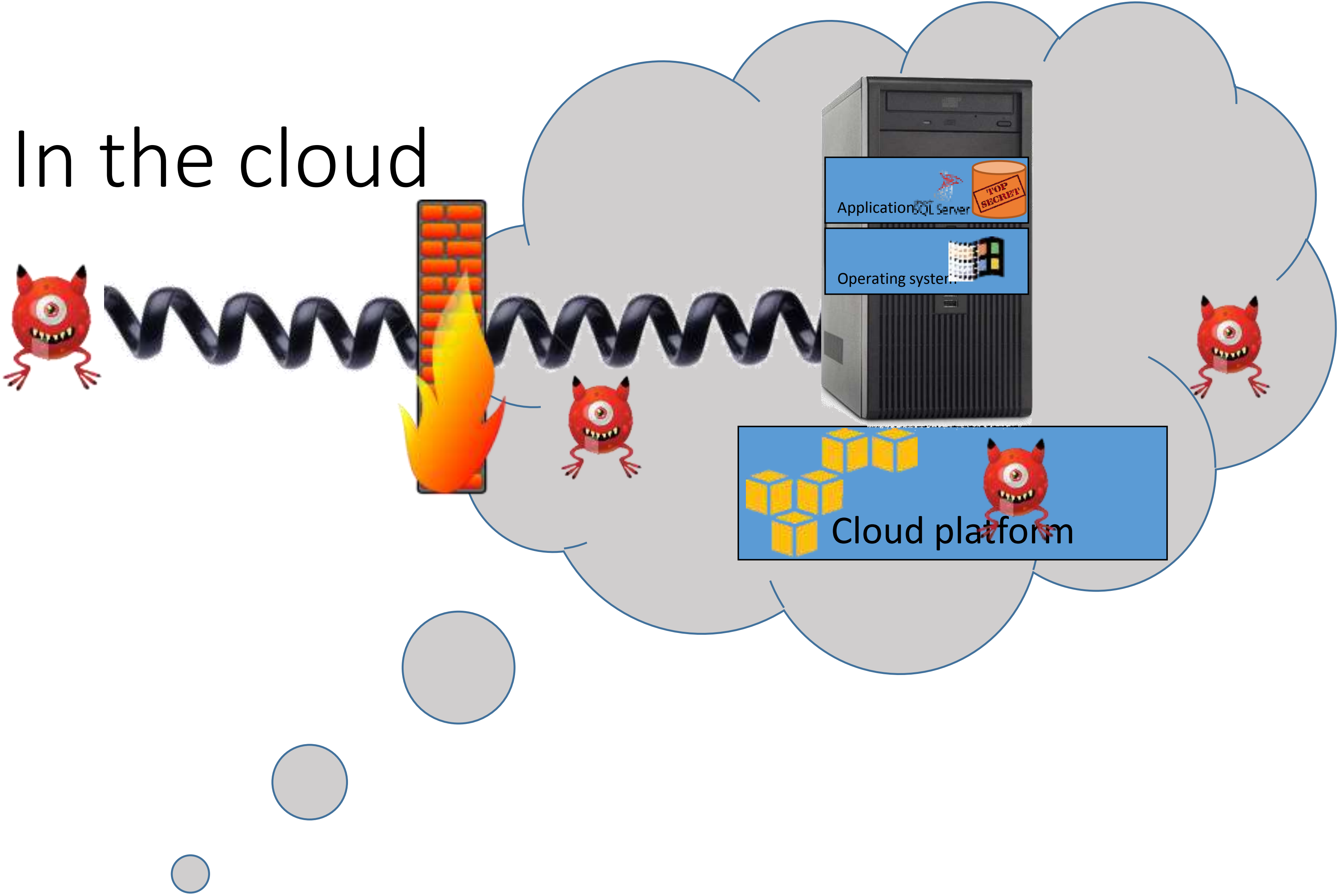
Trust…?

In the cloud



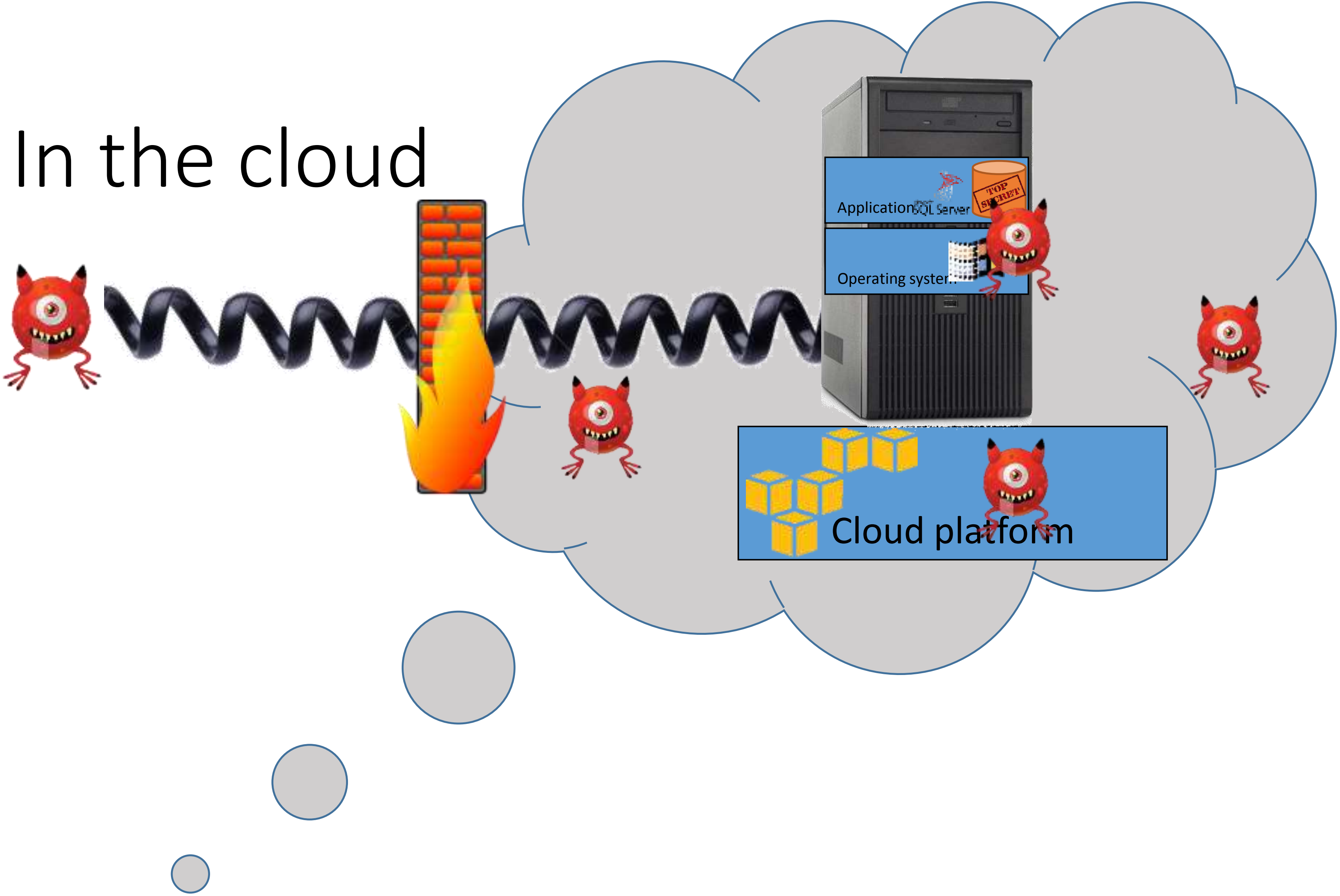
Trust…?

In the cloud



Trust…?

In the cloud



Trust…?

Our goals for Haven

Secure, private execution

of unmodified applications

(bugs and all)

in an untrusted cloud

on commodity hardware

(Intel SGX)

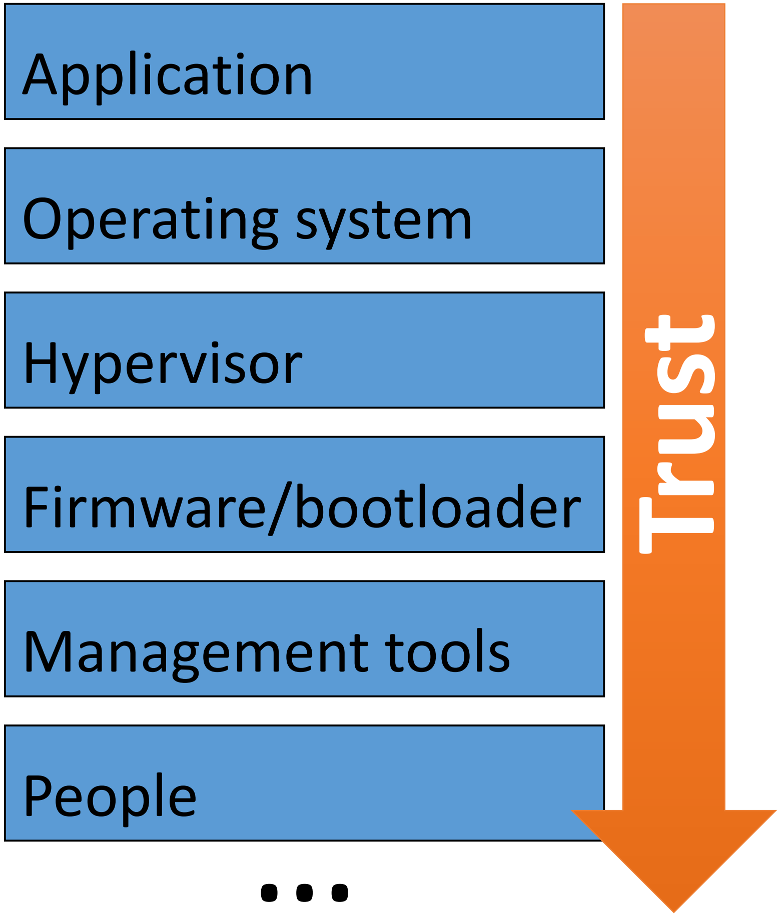
Microsoft SQL Server

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

- Use a small, secure, hypervisor
- 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

- Trusted hardware: 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

Secure colo provides:

Power and cooling

Network access

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.

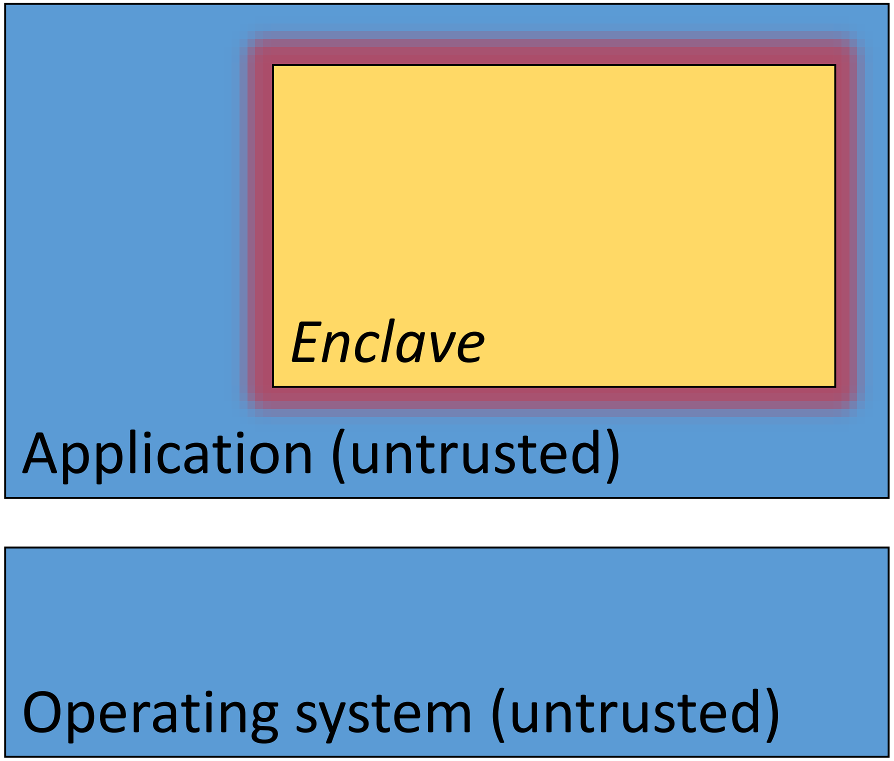
  → Input and output may be encrypted
- Host may deny service, cannot alter behaviour

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

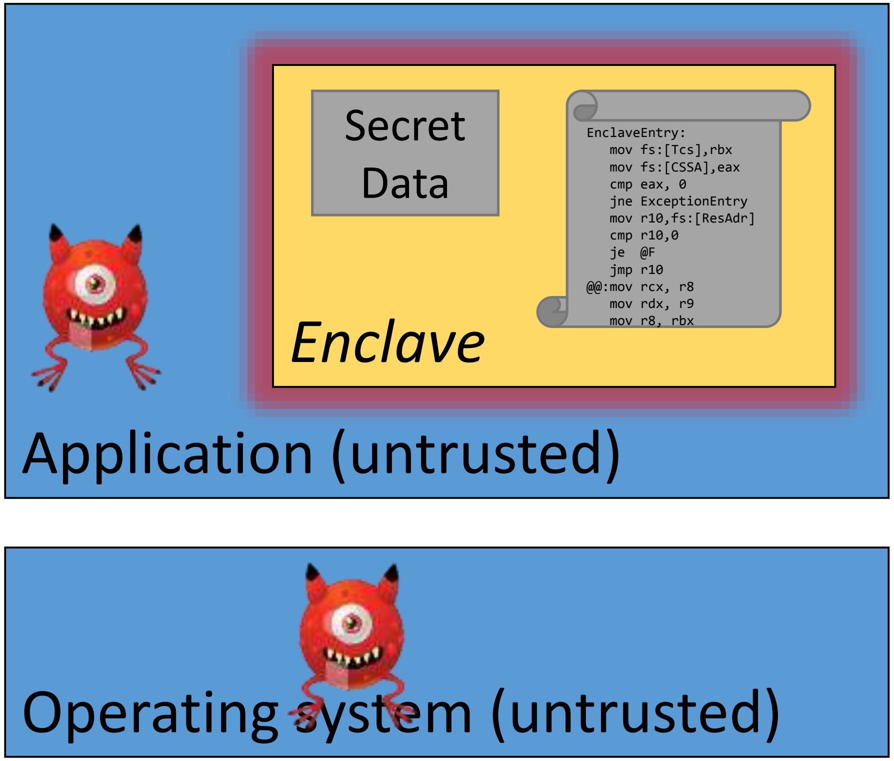
- We do not prevent:
  - Denial-of-service (don’t pay!)
  - Side-channel attacks

Intel SGX


Enclave
Application (untrusted)
Operating system (untrusted)

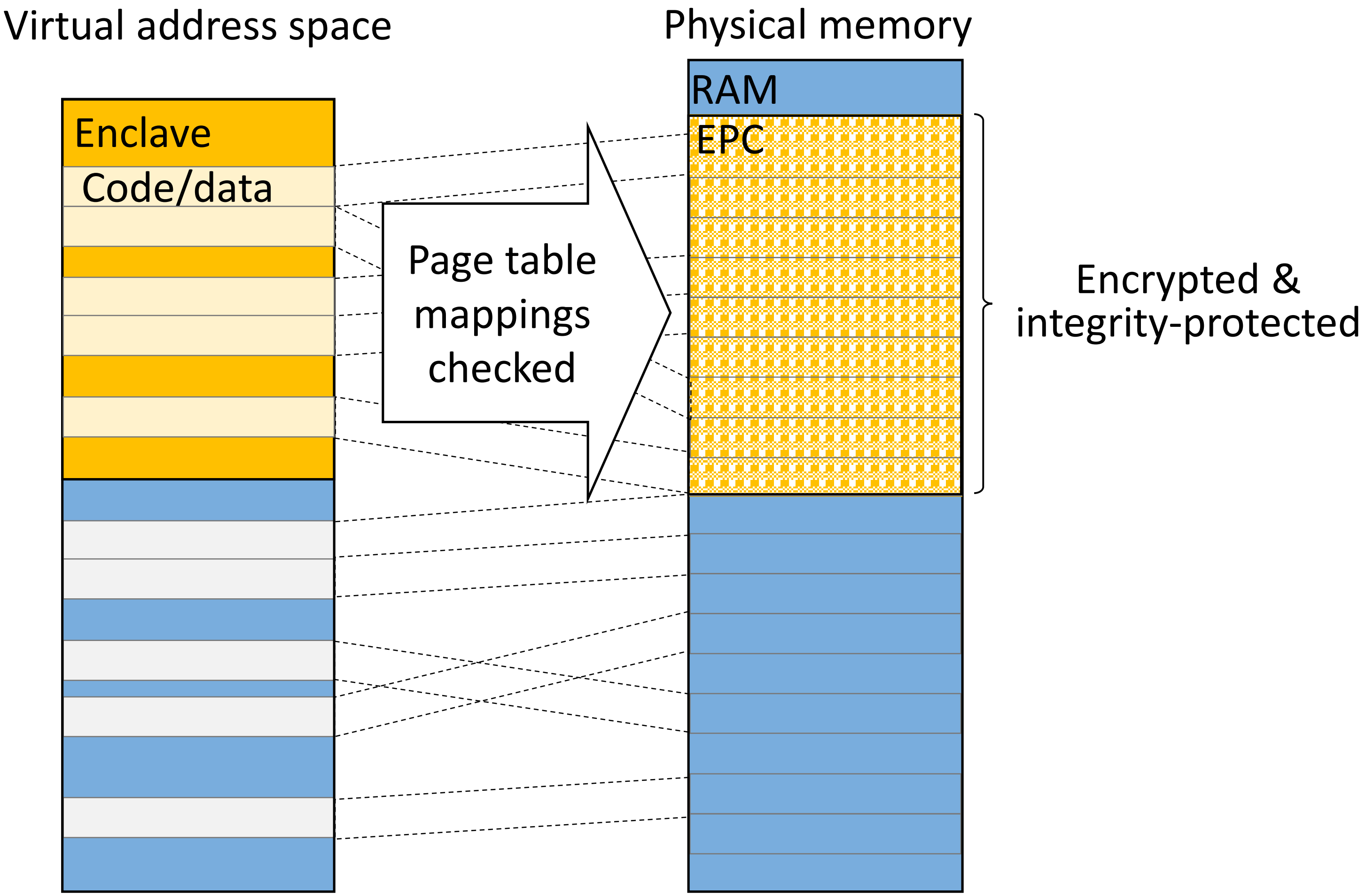
Intel SGX

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

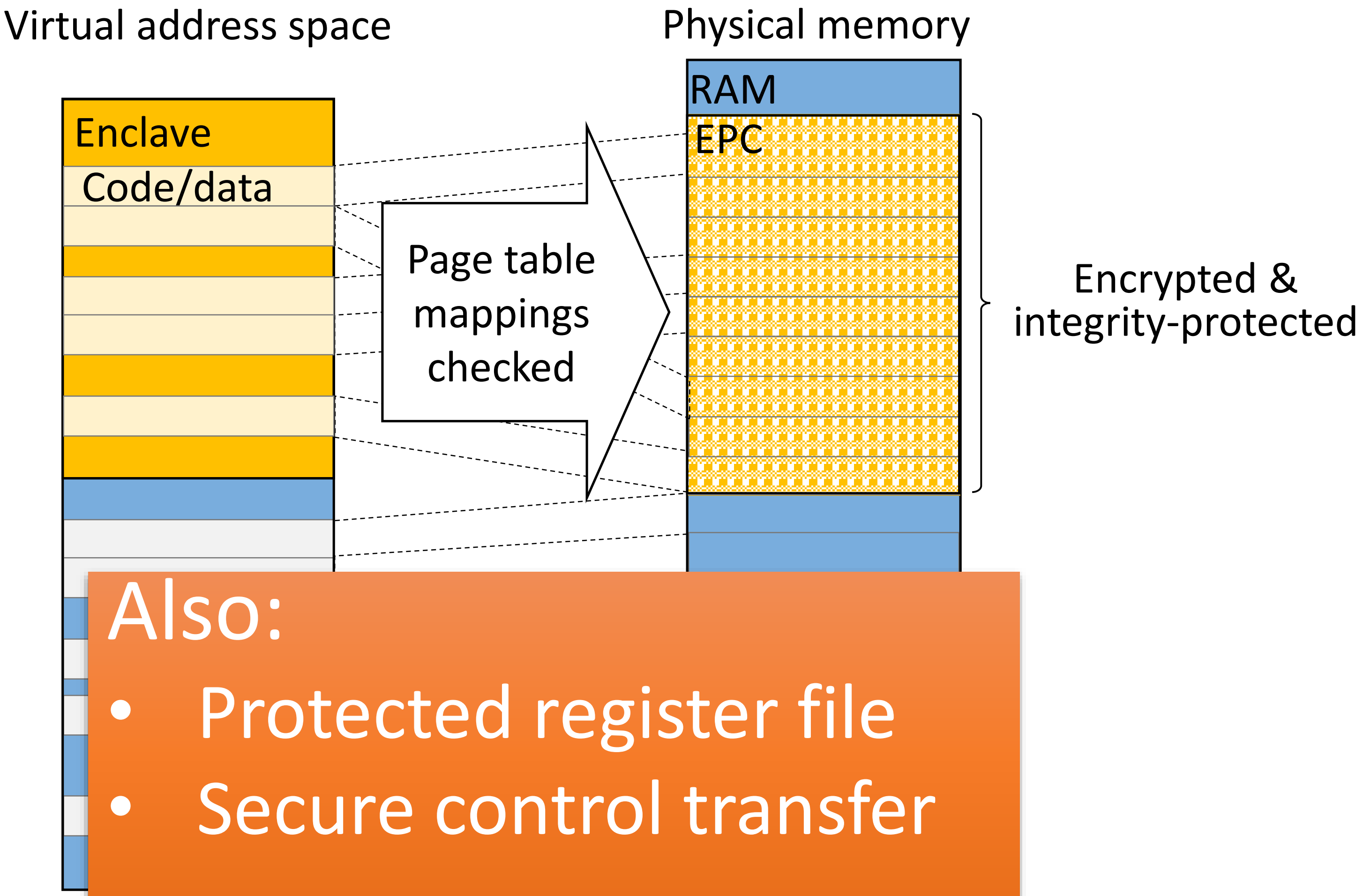


```
EnclaveEntry:
    mov fs:[Tcs],rbx
    mov fs:[CSSA],eax
    cmp eax, 0
    jne ExceptionEntry
    mov r10,fs:[ResAdr]
    cmp r10,0
    je  @F
    jmp r10
@@:mov rcx, r8
    mov rdx, r9
    mov r8, rbx
```

SGX at the hardware level



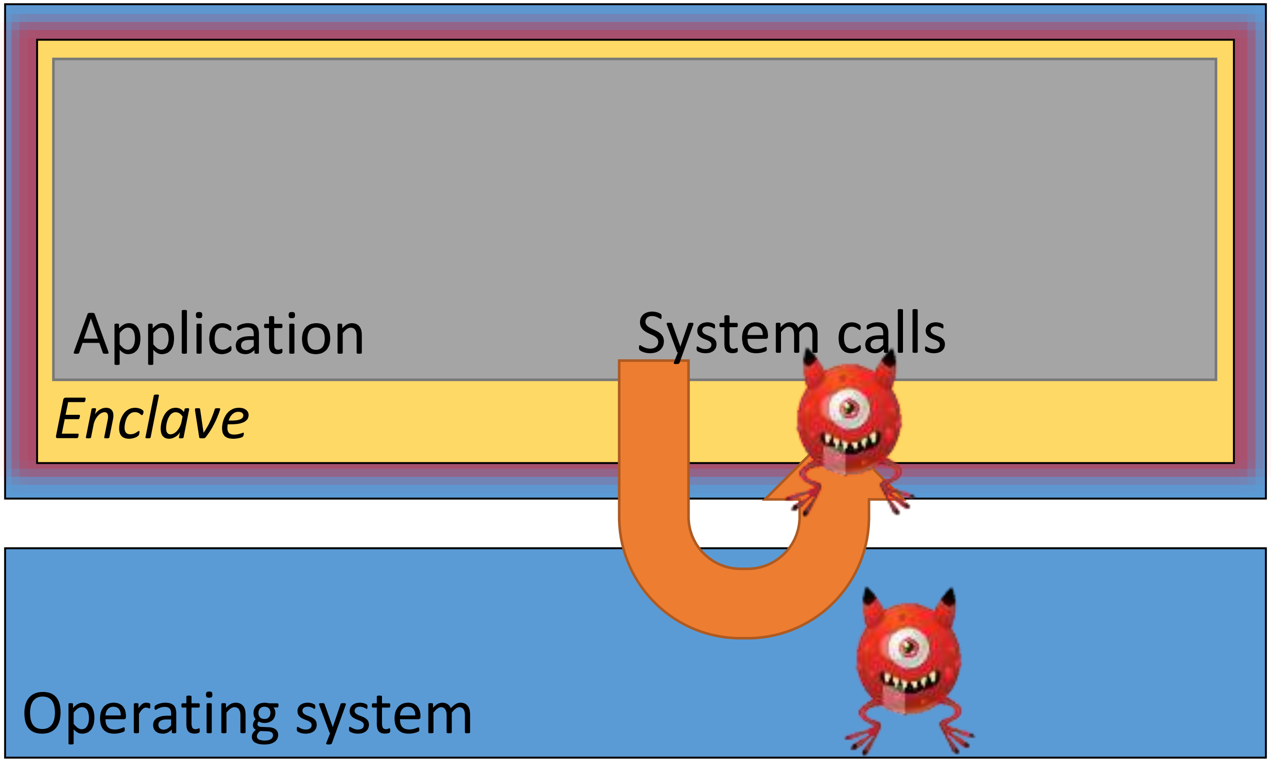
SGX at the hardware level



Also:

- Protected register file
- Secure control transfer

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:

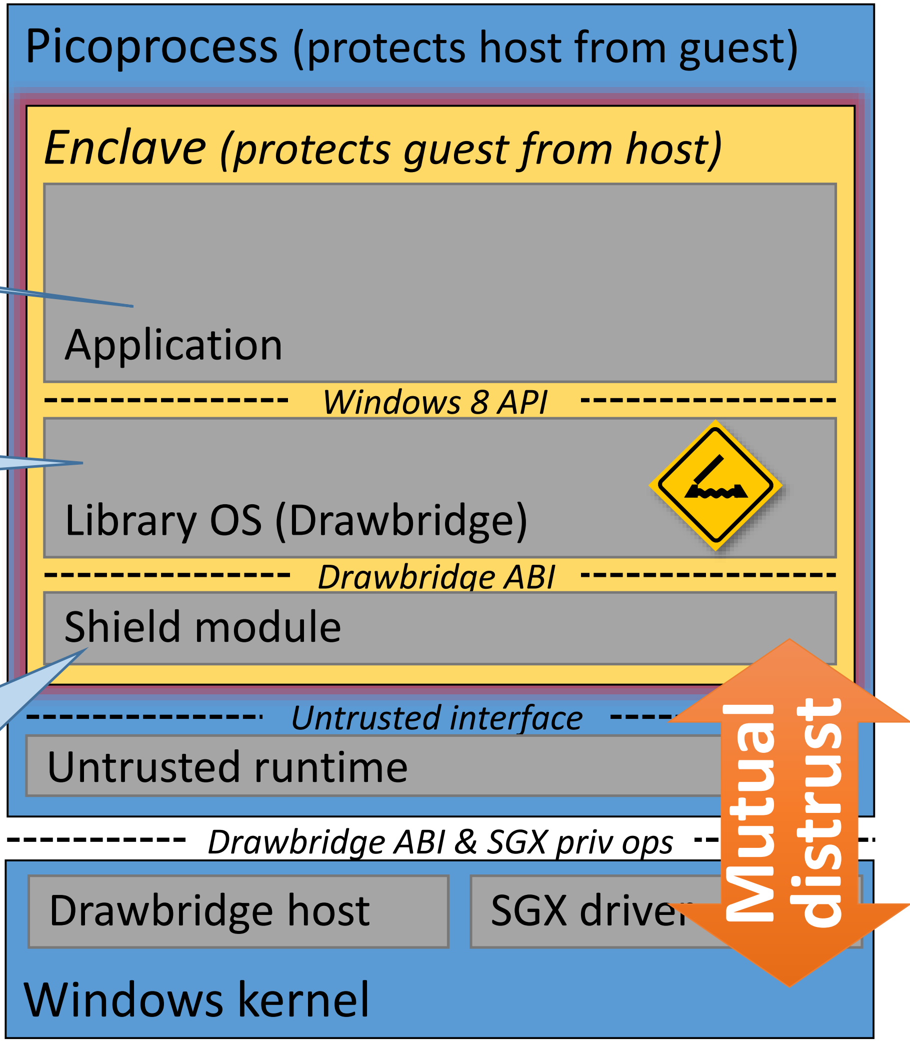
- Don’t try to check them all
- Admit OS into trusted computing base

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

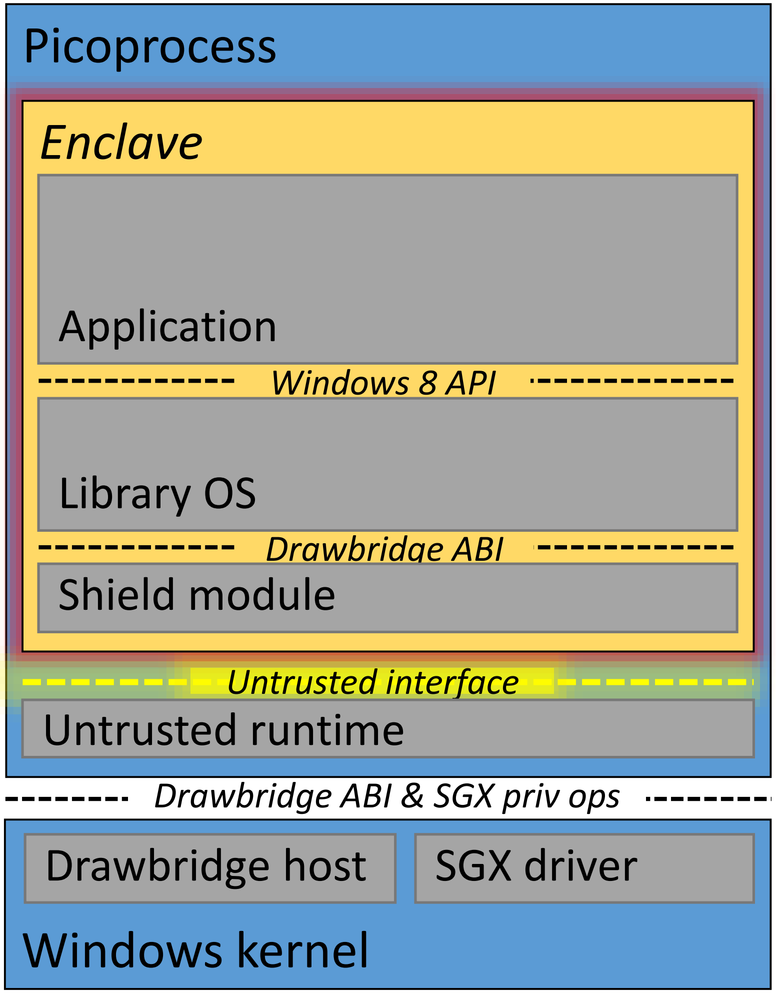


Untrusted interface

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

    Virtual address allocation, threads
  - Physical resource policy in host

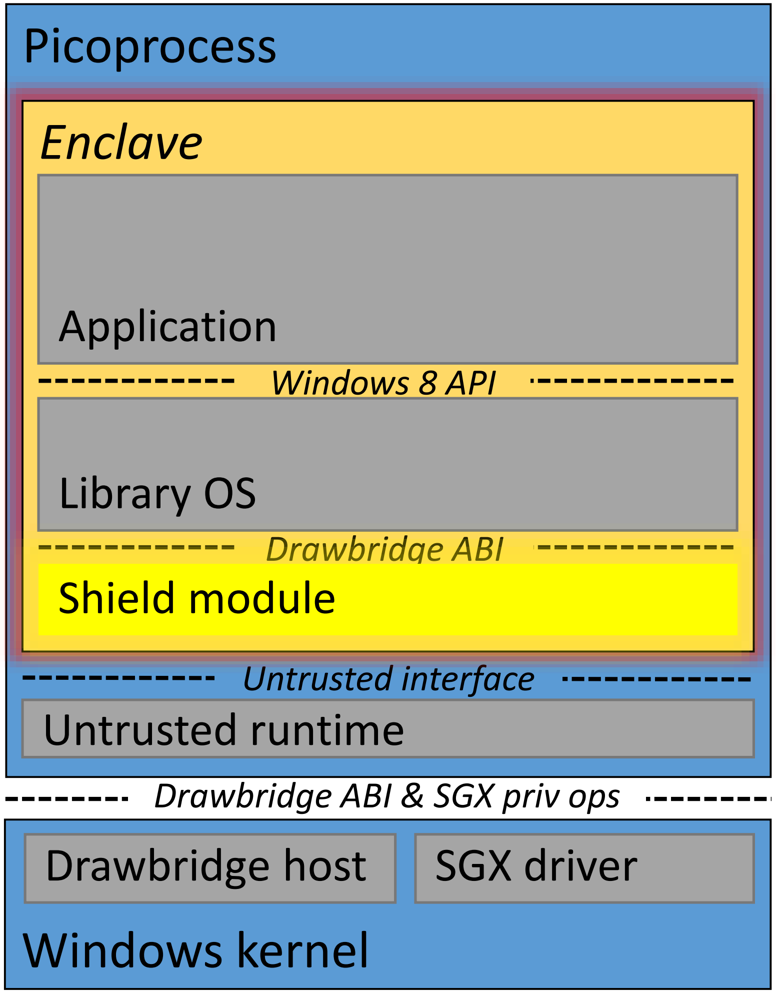
    Physical pages, VCPUs
- ~20 calls, restricted semantics



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 doesn’t report page faults or GPFs to the enclave
3. Permitted instructions

- RDTSC/RDTSCP needed, for practicality and performance

1. Thread-local storage
    - Can’t reliably switch FS and GS

SGX limitations

1. Dynamic memory allocation and protection
   - New instructions needed
2. Exception handling
   - SGX d enclave
3. Permi

- RDTSC/ formance

1. 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

   1.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!

What’s next?

- Rollback of persistent storage
  - Requires more hardware or communication
- Untrusted time
  - Network time sync, RDTSC
- Cloud management
  - Suspend / resume / migrate applications
  - Encrypted VLANs

Conclusion

- Closer to a true “utility computing” model
  - Utility provides raw resources
  - Doesn’t care what you do with them
- Why trust the cloud when you don’t have to?

Thanks!

baumann@microsoft.com