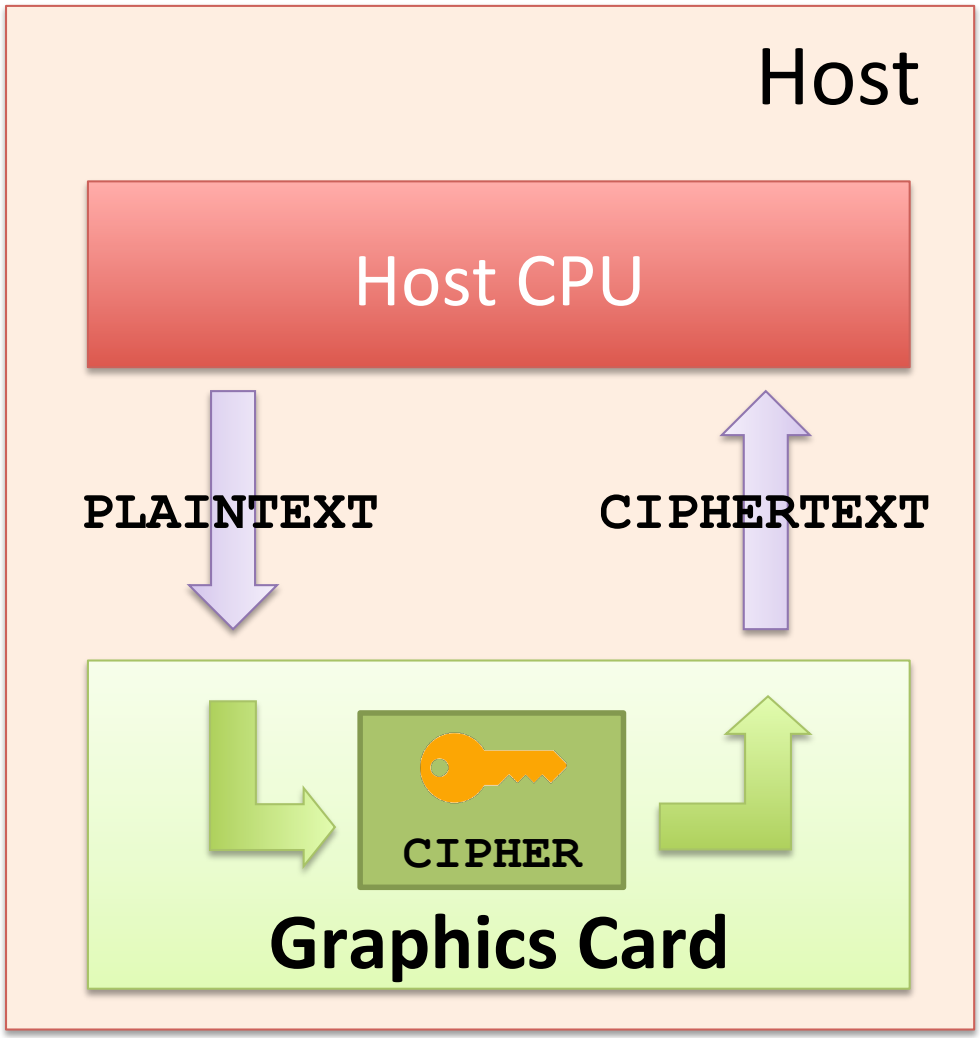
PixelVault: Using GPUs for Securing Cryptographic Operations

Giorgos Vasiliadis gvasil@ics.forth.gr

Motivation

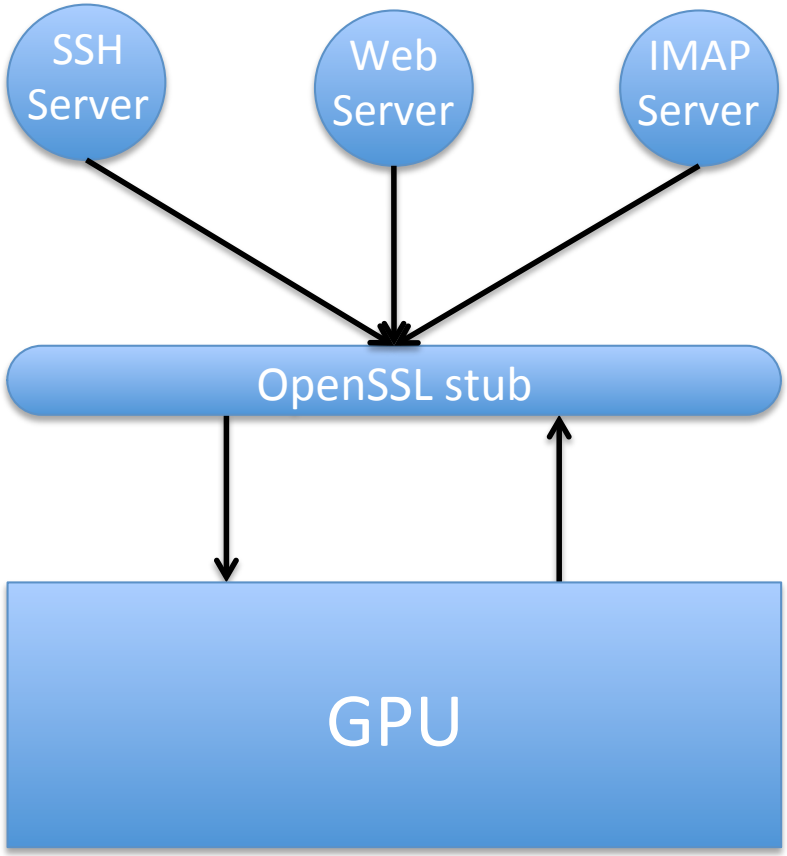
- Secret keys may remain unencrypted in CPU Registers, RAM, etc.
  - Memory disclosure attacks
    - Heartbleed
  - DMA/Firewire attacks
  - Physical attacks
    - Cold-boot attacks
  - …

PixelVault Overview



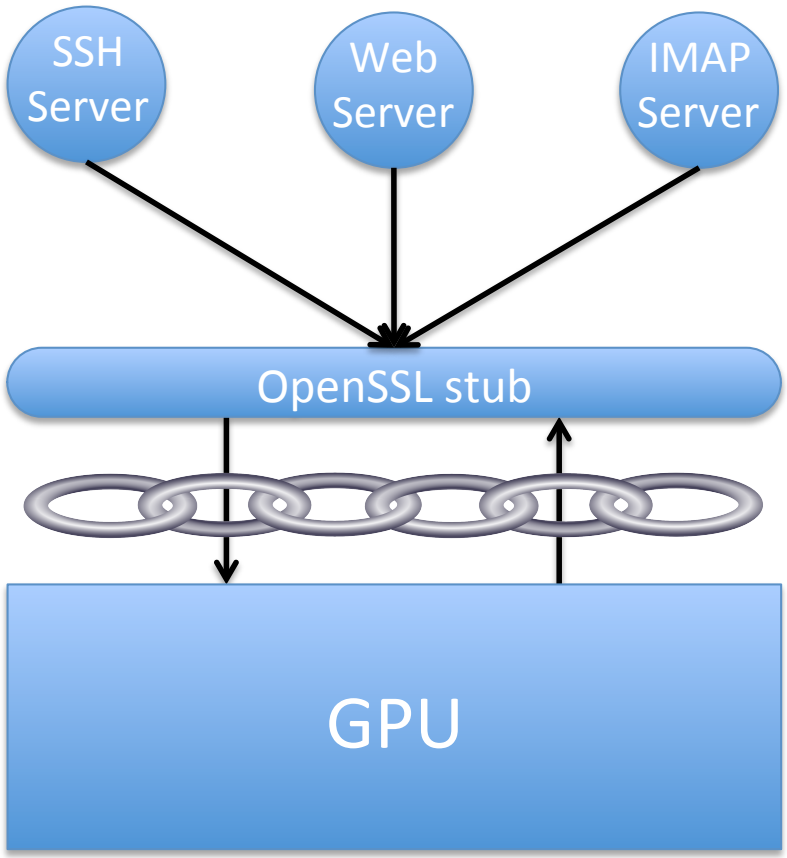
- Runs encryption securely outside CPU/ RAM
- Only on-chip memory of GPU is used as storage
- Secret keys are never observed from host

Cryptographic Processing with GPUs



- GPU-accelerated SSL
  - [CryptoGraphics, CT-RSA'05]
  - [Harrison et al., Sec'08]
  - [SSLShader, NSDI'11]
  - …
- High-performance
- Cost-effective

Cryptographic Processing with GPUs



- GPU-accelerated SSL
  - [CryptoGraphics, CT-RSA'05]
  - [Harrison et al., Sec'08]
  - [SSLShader, NSDI'11]
  - …
- High-performance
- Cost-effective

Can we also make it secure?

Implementation Challenges

- How to isolate GPU execution?
- Who holds the keys?
- Where is the code?

Implementation Challenges

- How to isolate GPU execution?
- Who holds the keys?
- Where is the code?

GPU as a coprocessor

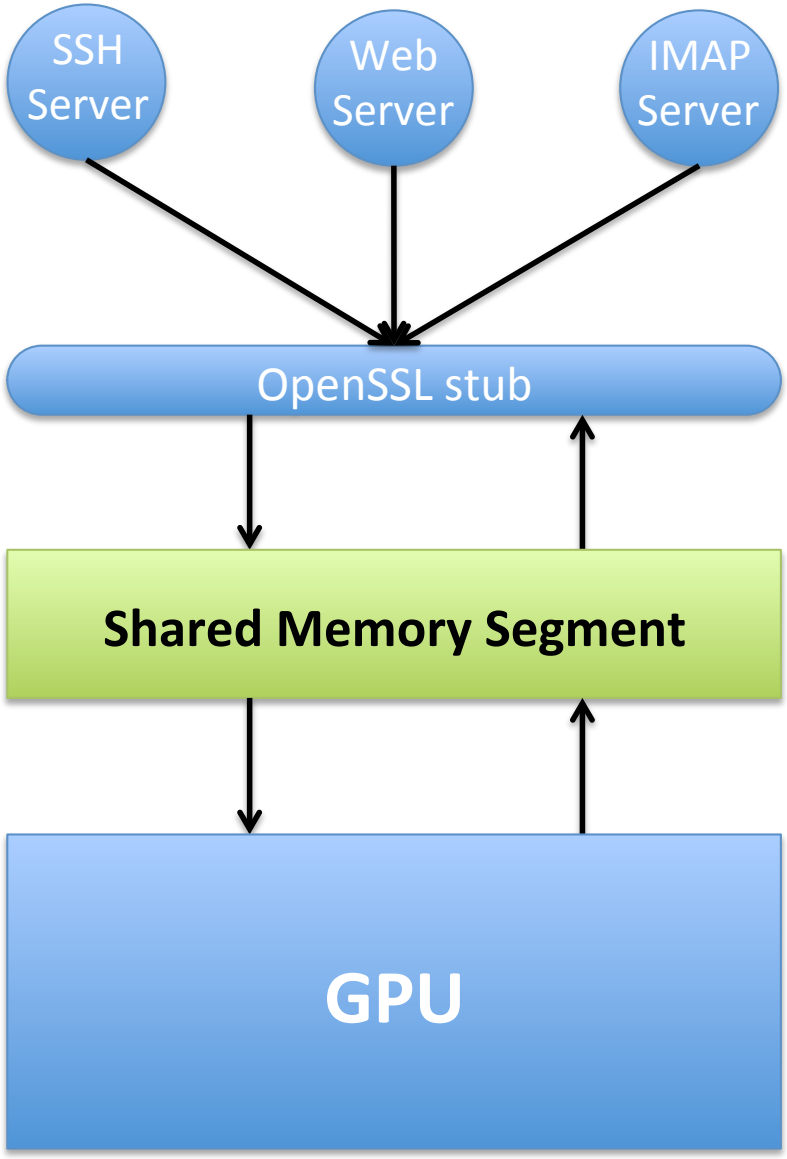
- Typically handled by the host
  - Load parameters, launch GPU program, transfer data, etc.

- Not secure for our purposes
  - Crypto keys have to be transferred every time

Autonomous GPU execution

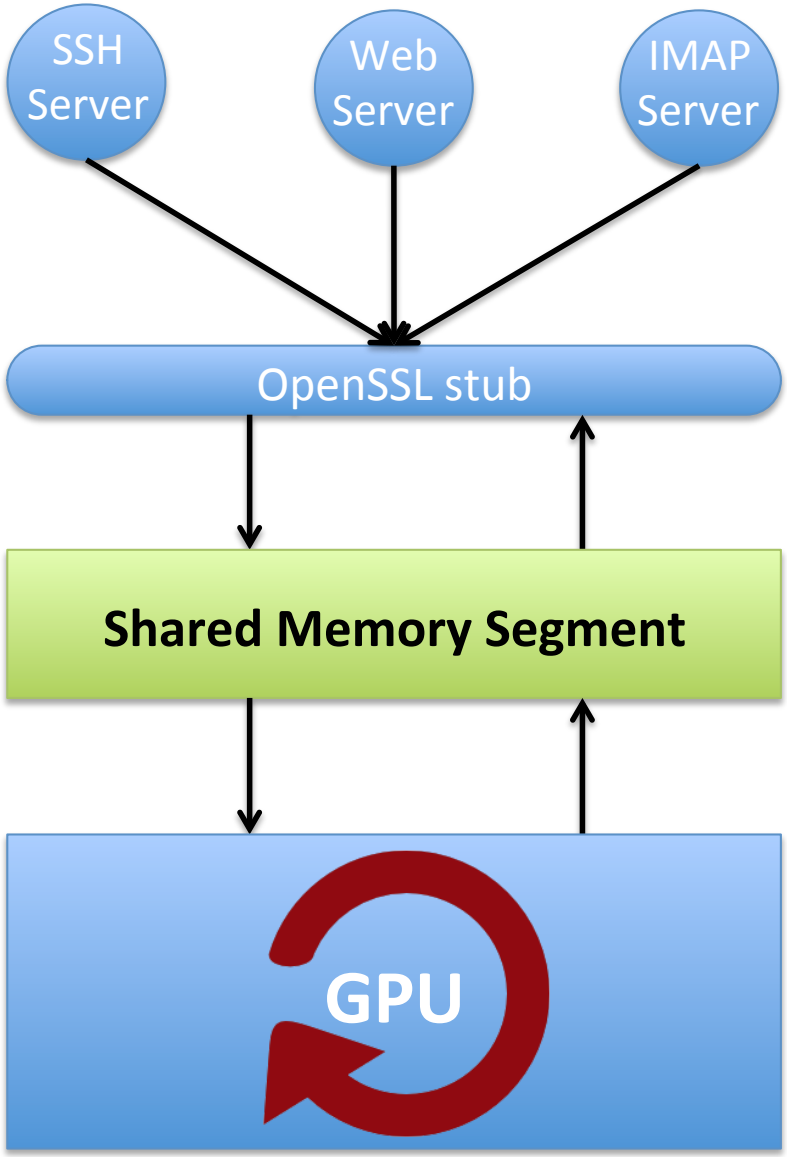
- Force GPU program to run indefinitely
  - i.e., using an infinite **`while`** loop
- GPUs are non-preemptive
  - No other program can run at the same time
- We use a shared memory segment for communication between the CPU and the GPU

Shared Memory between CPU/GPU



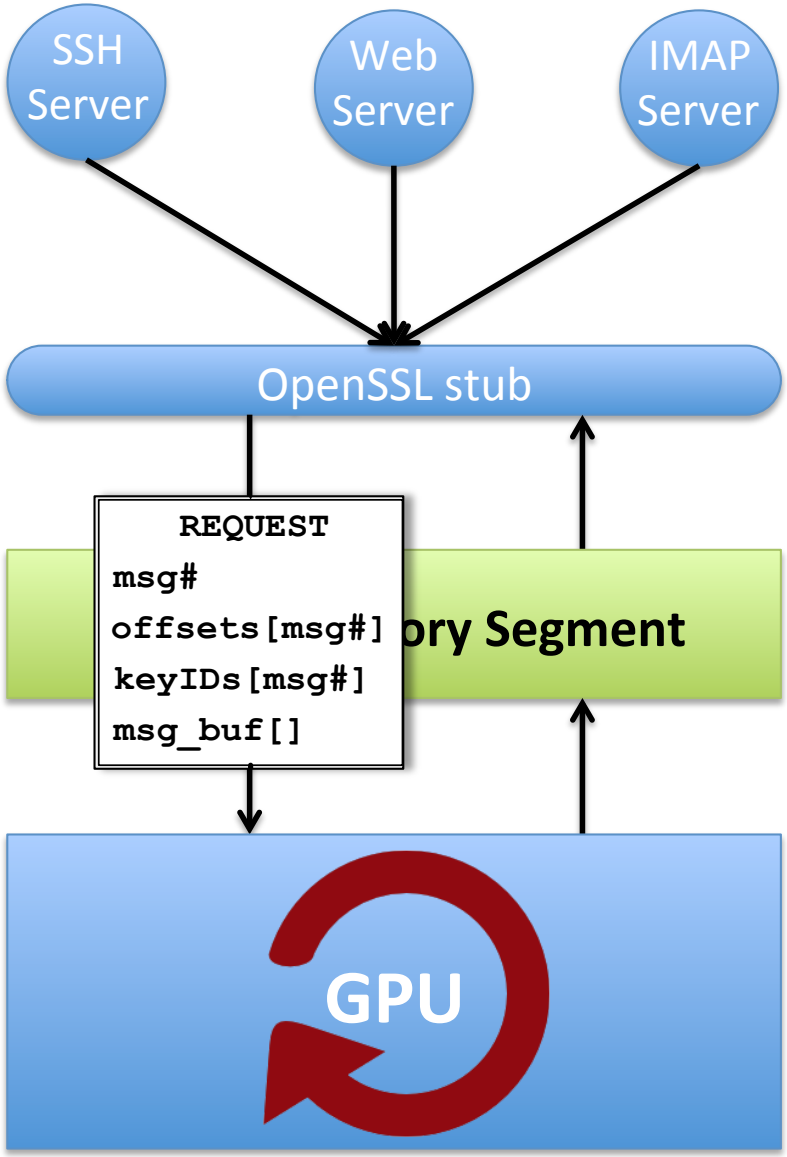
- *Page-locked* memory
  - Accessed by the GPU directly, via DMA
  - Cannot be swapped to disk
- Processing requests are issued through this shared memory space

Shared Memory between CPU/GPU



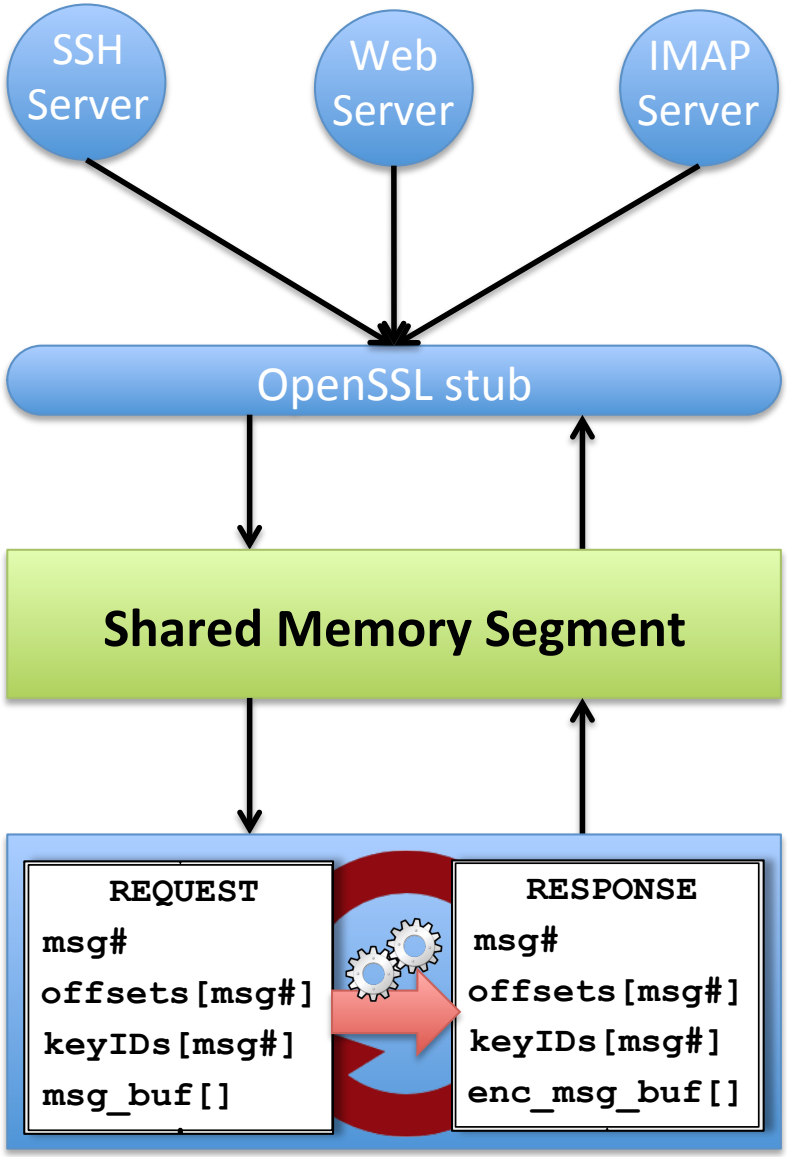
- GPU continuously monitors the shared space for new requests

Shared Memory between CPU/GPU



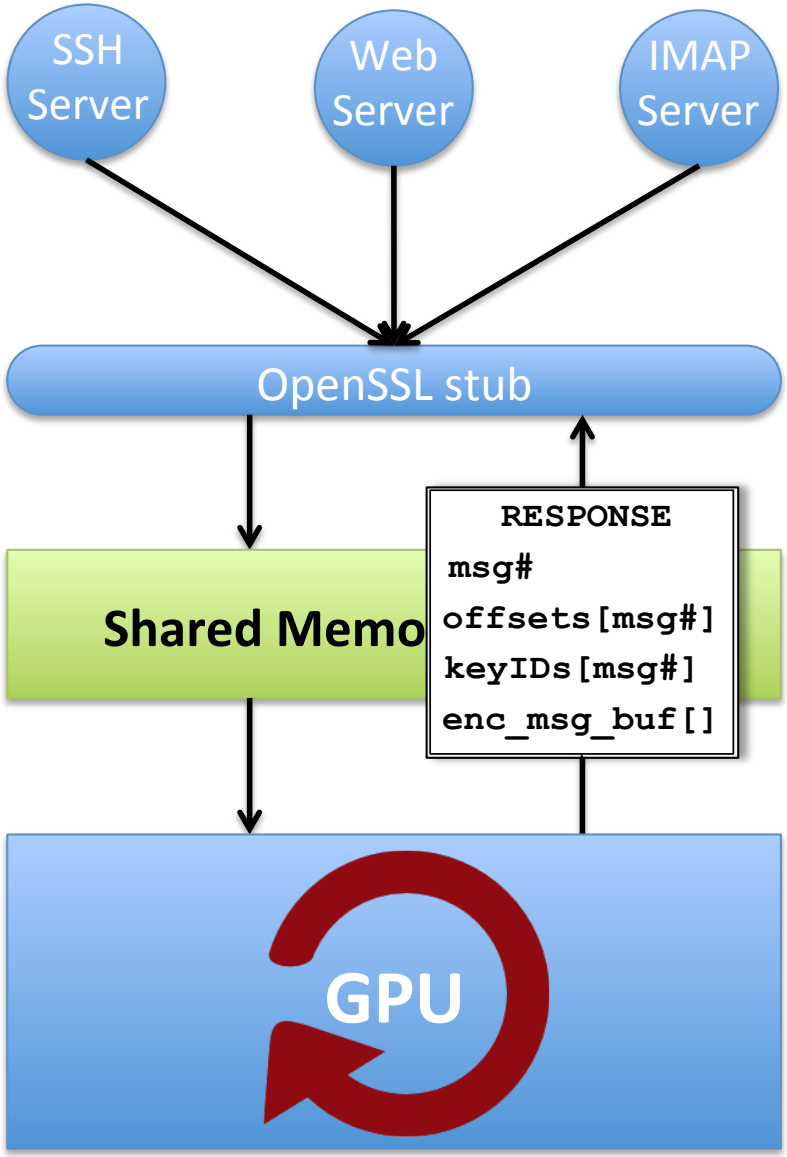
- When a new request is available, it is transferred to the memory space of the GPU

Shared Memory between CPU/GPU



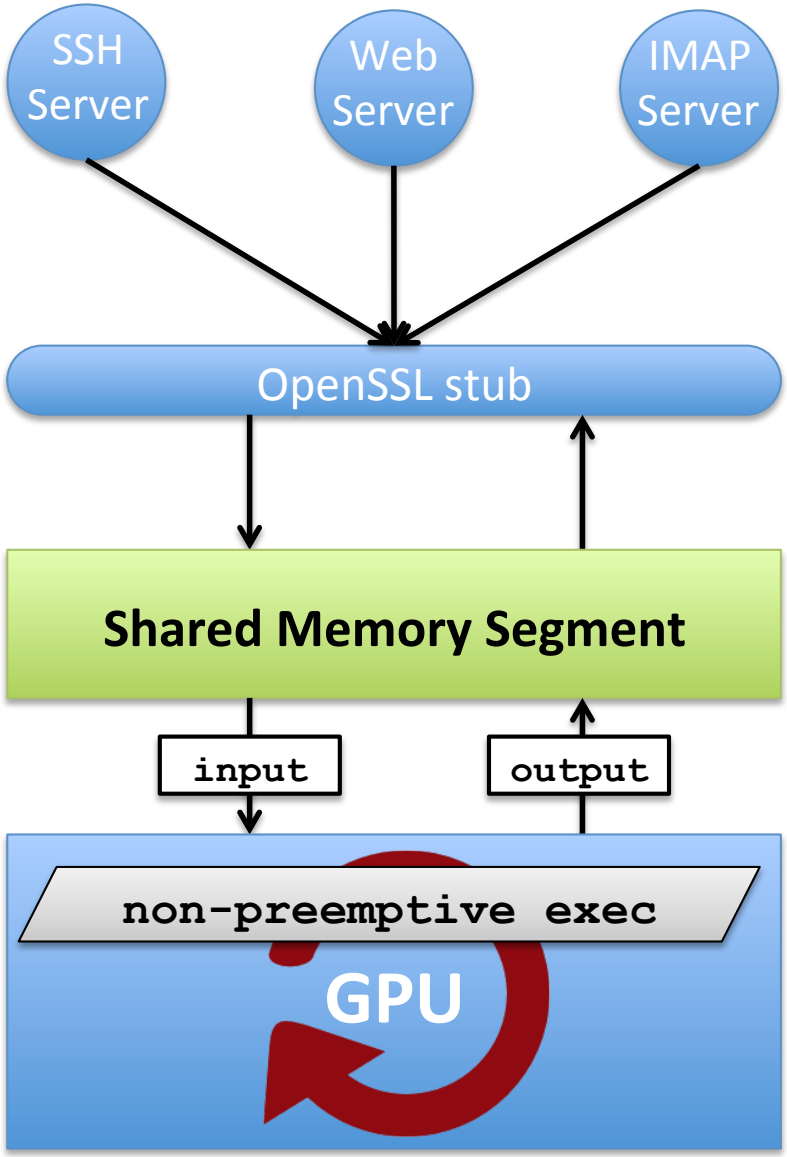
- The request is processed by the GPU

Shared Memory between CPU/GPU



- When processing is finished, the host is notified by setting the response parameter fields accordingly

Autonomous GPU execution

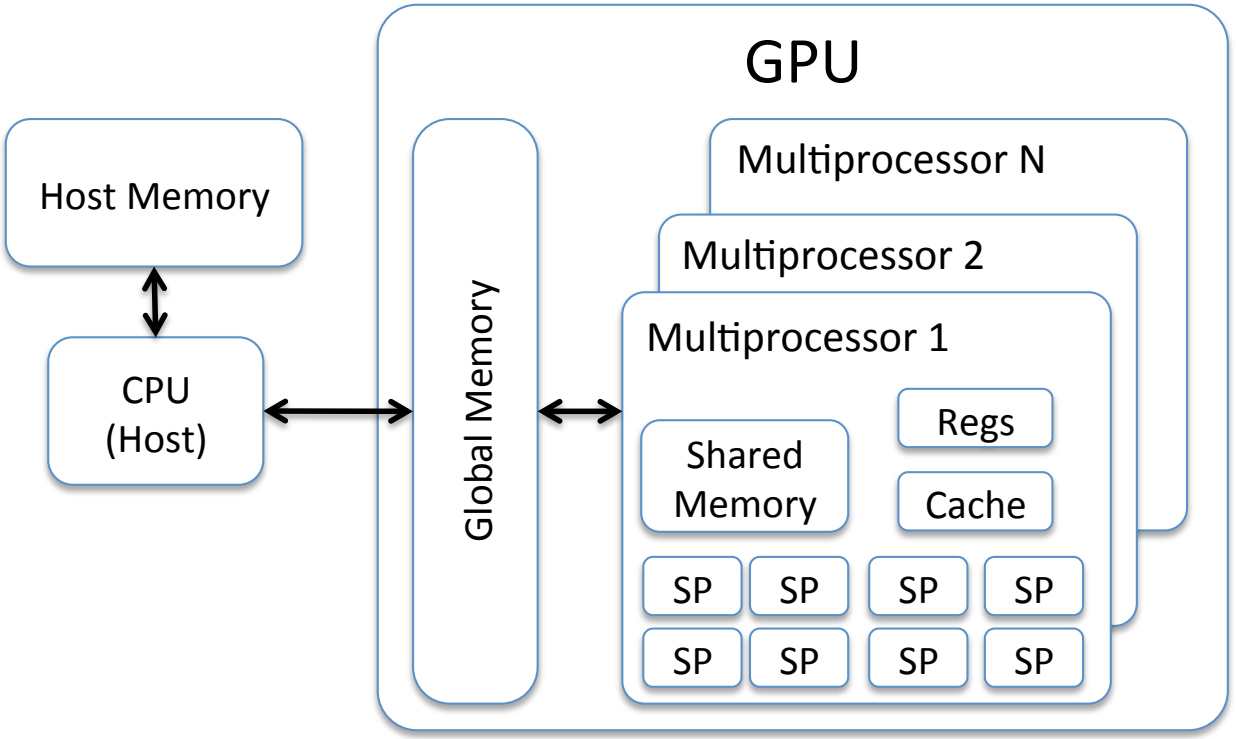


- Non-preemptive execution
- Only the output block is being written back to host memory

Implementation Challenges

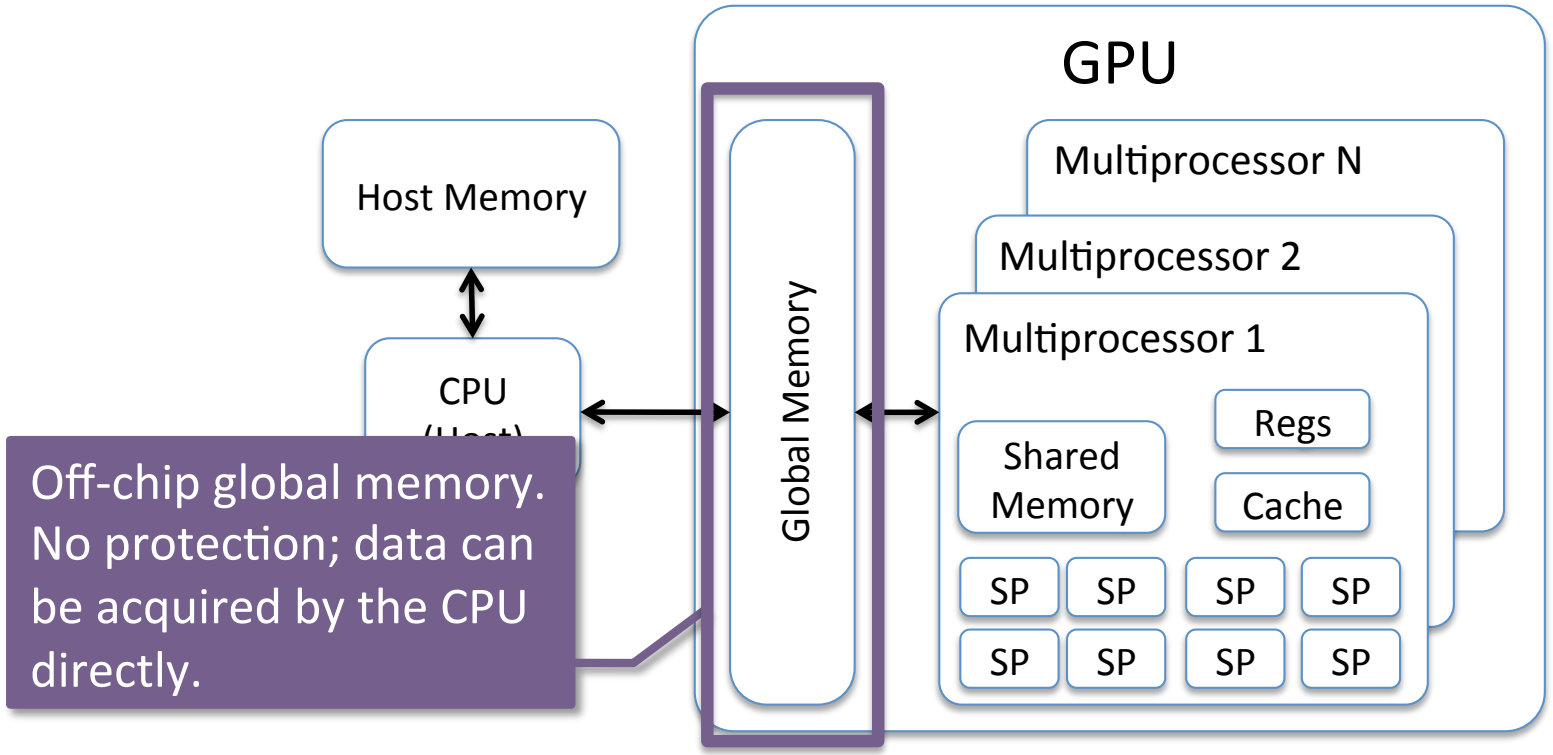
- How to isolate GPU execution?
- Who holds the keys?
- Where is the code?

Who holds the keys?



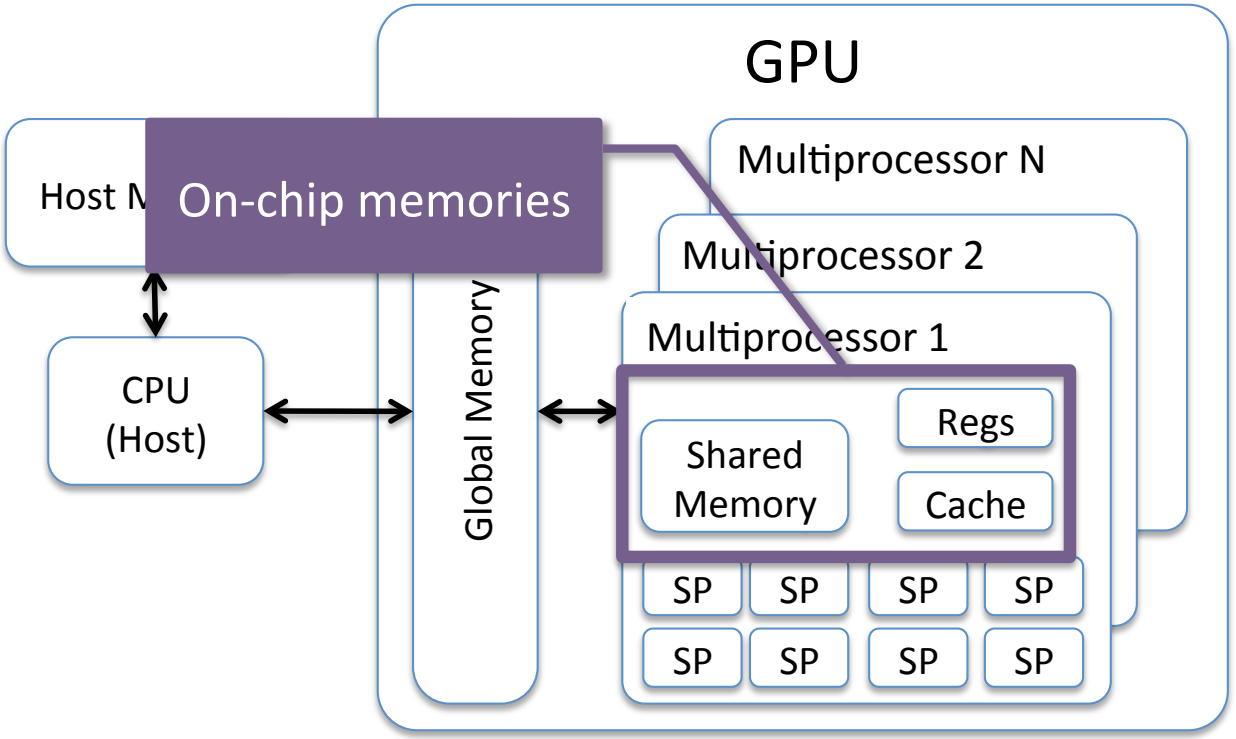
- GPUs contain different memory hierarchies of …
  - different sizes, and …
  - different characteristics

Who holds the keys?



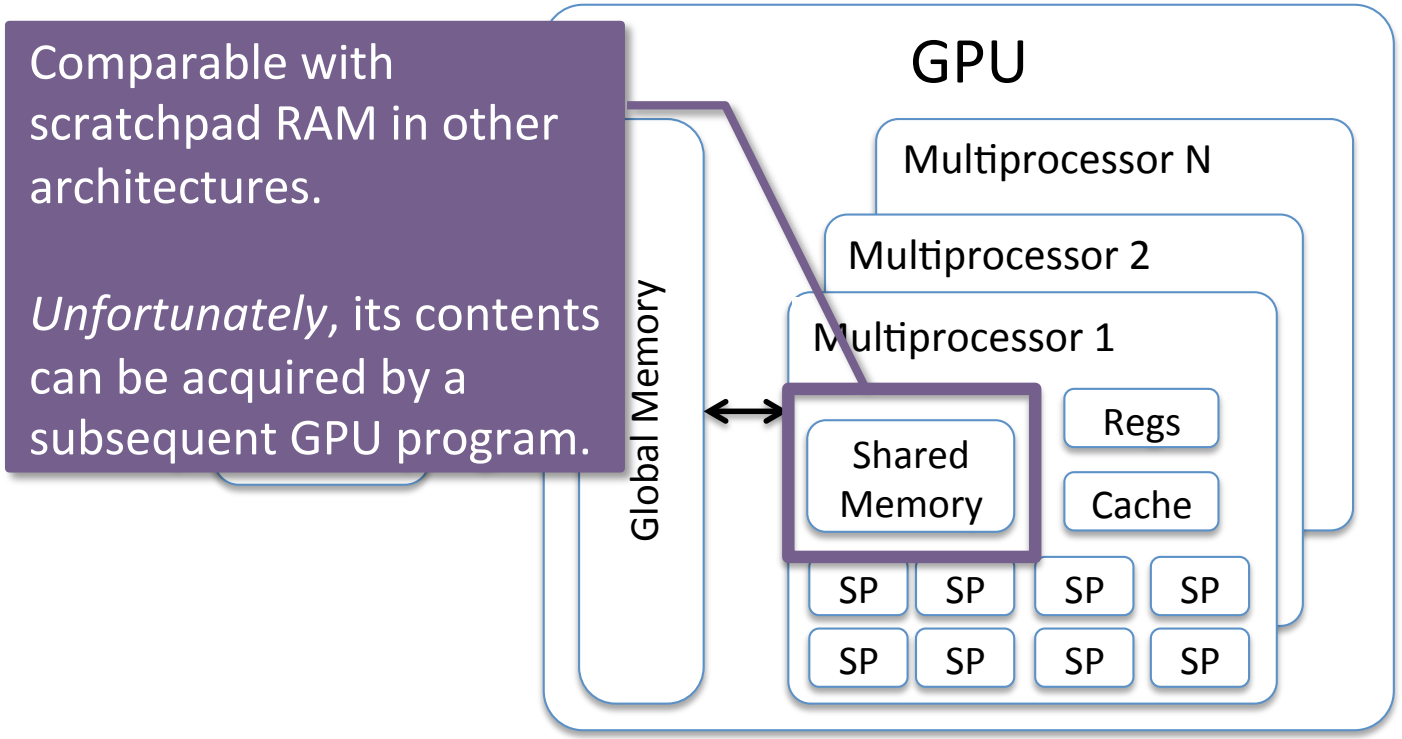
- GPUs contain different memory hierarchies of …
  - different sizes, and …
  - different characteristics

Who holds the keys?



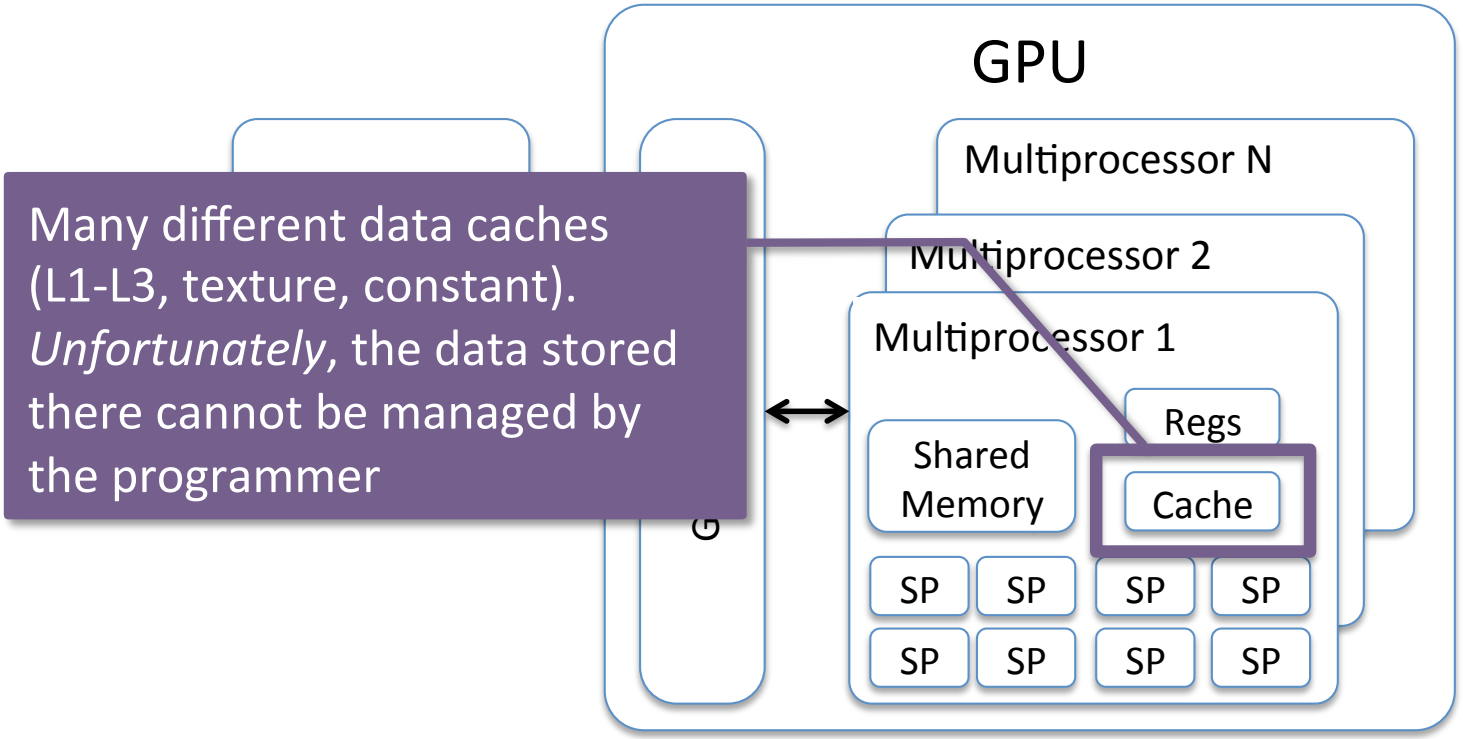
- GPUs contain different memory hierarchies of …
  - different sizes, and …
  - different characteristics

Who holds the keys?



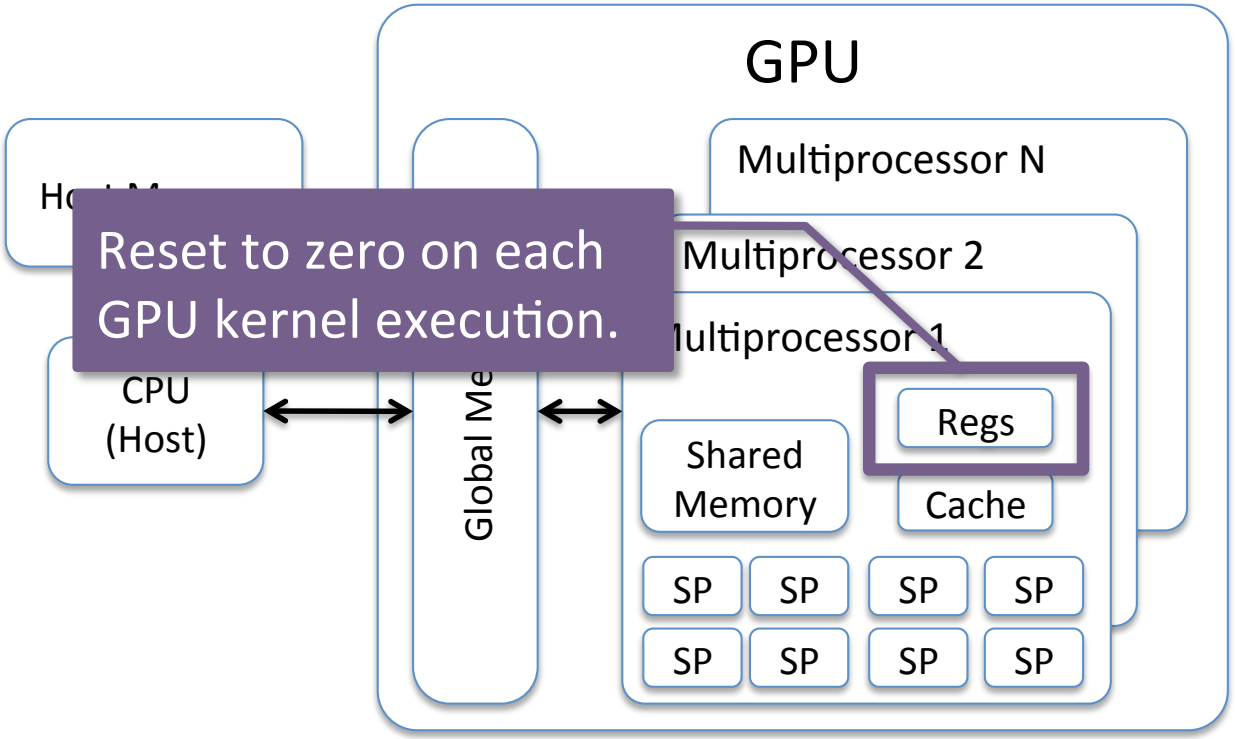
- GPUs contain different memory hierarchies of …
  - different sizes, and …
  - different characteristics

Who holds the keys?



- GPUs contain different memory hierarchies of …
  - different sizes, and …
  - different characteristics

Who holds the keys?



- GPUs contain different memory hierarchies of …
  - different sizes, and …
  - different characteristics

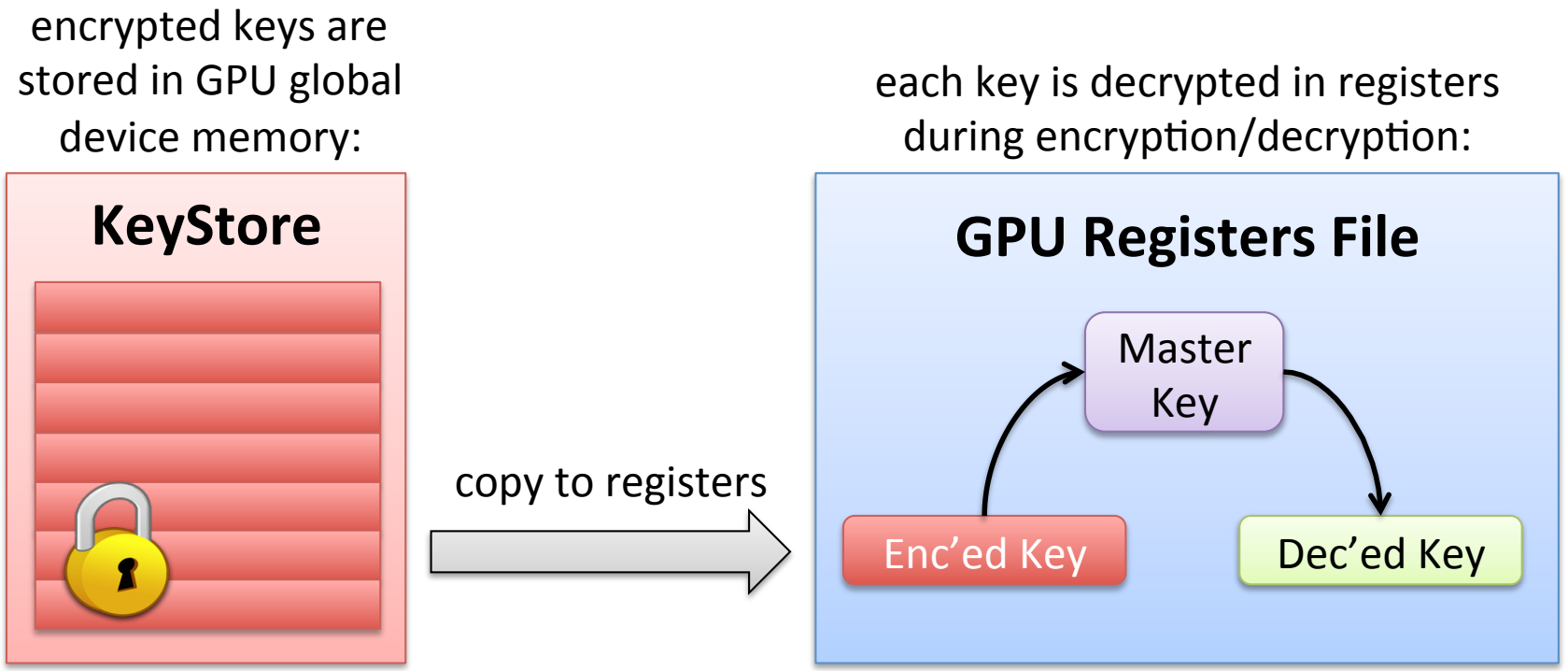
Keeping secrets on GPU registers

- Secret keys are loaded on GPU registers at an early stage of the bootstrapping phase
  - Remain there as long as the autonomous GPU program is running

- Unfortunately, the number of available registers in current GPU models is small
  - Enough for a single/few secret keys, but *what about if we want to store more?*

Support for an arbitrary number of keys

- We can use a separate KeyStore array that holds an arbitrary number of secret keys



Implementation Challenges

- How to isolate GPU execution?
- Who holds the keys?
- Where is the code?

```
mov.u32 %r2, 0;
setp.le.s32 %p1, %r1, %r2;
mov.s32 %r5, %r4;
add.u32 %r6, %r1, %r4;
@%p1 bra $Lt_0_1282;
mov.s32 %r8, %r3;
xor.b32 %r10, %r7, %r9;
st.global.u8 [%r5+0], %r10;
add.u32 %r5, %r5, 1;
setp.ne.s32 %p2, %r5, %r
```

Where is the code?

- GPU code is initially stored in global device memory for the GPU to execute it
  - An adversary could replace it with a malicious version

Global Device Memory

```
mov.u32 %r2, 0;
setp.le.s32 %p1, %r1, %r2;
mov.s32 %r5, %r4;
add.u32 %r6, %r1, %r4;
@%p1 bra $Lt_0_1282;
mov.s32 %r8, %r3;
xor.b32 %r10, %r7, %r9;
st.global.u8 [%r5+0], %r10;
add.u32 %r5, %r5, 1;
setp.ne.s32 %p2, %r5, %r
```

Prevent GPU code modification attacks

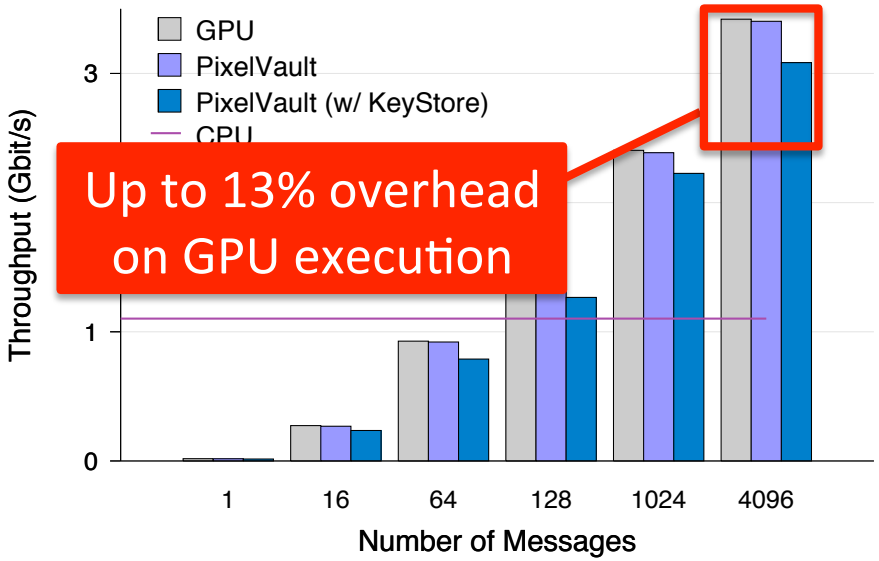
- Three levels of instruction caching (icache)
  - 4KB, 8KB, and 32KB, respectively
  - Hardware-managed

- **Opportunity:** Load the code to the icache, and then erase it from global device memory
  - The code runs indefinitely from the icache
  - Not possible to be flushed or modified

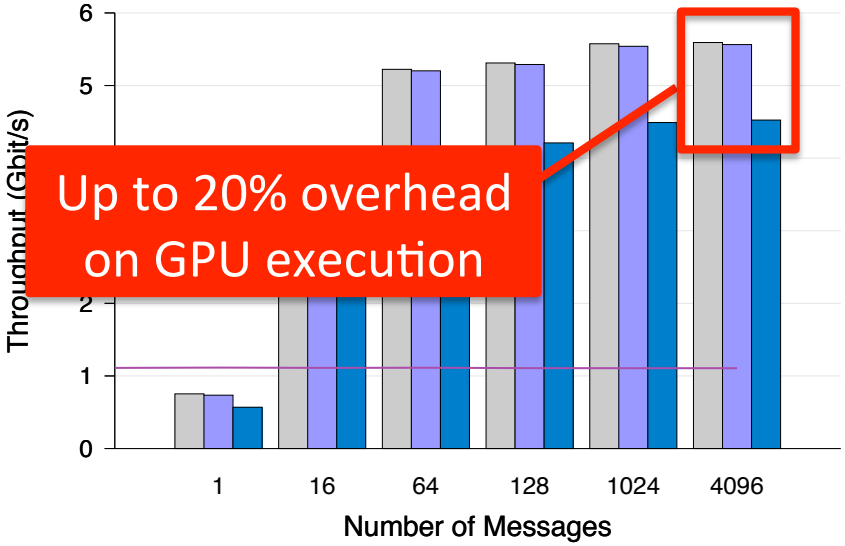
PixelVault Crypto Suite

- Currently implemented algorithms
  - AES-128
  - RSA-1024
- Implemented completely using on-chip memory (i.e. registers, scratchpad memory)
  - The only data that is written back to global, off-chip device memory is the output block

AES-128 CBC Performance

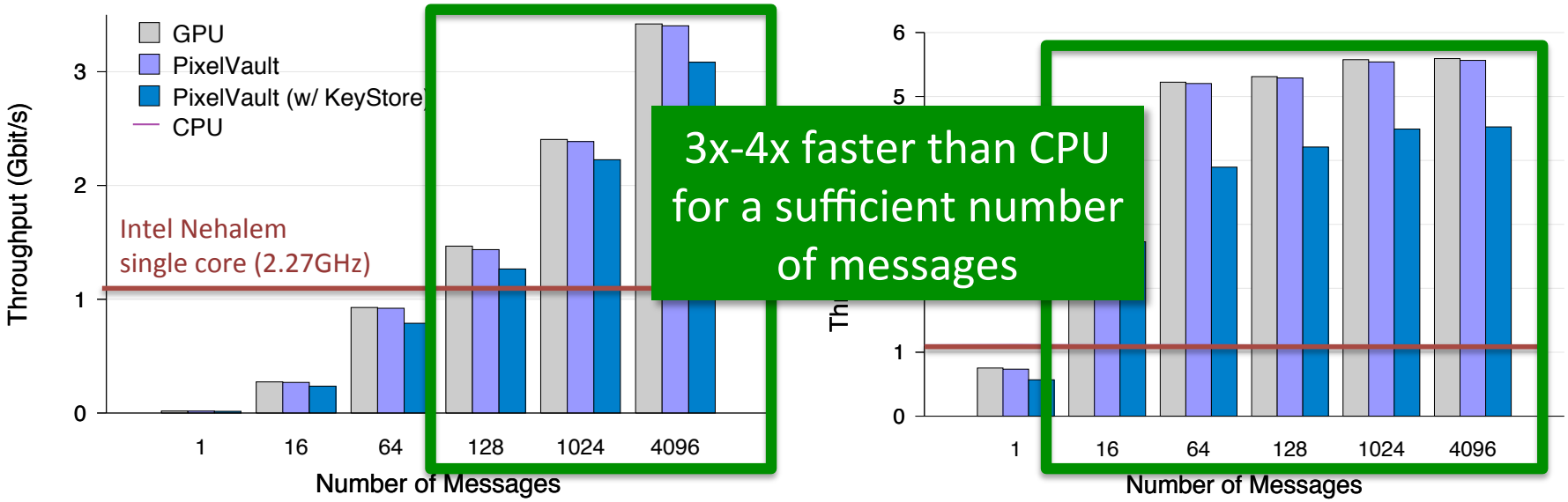


Encryption



Decryption

AES-128 CBC Performance



Encryption

Decryption

RSA 1024-bit Decryption

#Msgs	CPU	GPU [25]	PixelVault	PixelVault (w/ KeyStore)
1	1632.7	15.5	15.3	14.3
16	1632.7	242.2	240.4	239.2
64	1632.7	954.9	949.9	939.6
112	1632.7	1659.5	1652.4	1630.3
128	1632.7	1892.3	1888.3	1861.7
1024	1632.7	10643.2	10640.8	9793.1
4096	1632.7	17623.5	17618.3	14998.8
8192	1632.7	24904.2	24896.1	21654.4

- PixelVault adds an 1%-15% overhead over the default GPU-accelerated RSA

RSA 1024-bit Decryption

#Msgs	CPU	GPU [25]	PixelVault	PixelVault (w/ KeyStore)
1	1632.7	15.5	15.3	14.3
16	1632.7	242.2	240.4	239.2
64	1632.7	954.9	949.9	939.6
112	1632.7	1659.5	1652.4	1630.3
128	1632.7	1892.3	1888.3	1861.7
1024	1632.7	10643.2	10640.8	9793.1
4096	1632.7	17623.5	17618.3	14998.8
8192	1632.7	24904.2	24896.1	21654.4

- Still faster than CPU when batch processing >128 messages

Conclusions

- Cryptography on the GPU is not only fast …
- … *but* also **secure!**
  - Preserves the secrecy of keys even when the base system is fully compromised

- More technical details
  - See our ACM CCS'2014 paper

    PixelVault: Using GPUs for Securing Cryptographic Operations"

PixelVault: Using GPUs for Securing Cryptographic Operations

thank you!

Giorgos Vasiliadis gvasil@ics.forth.gr