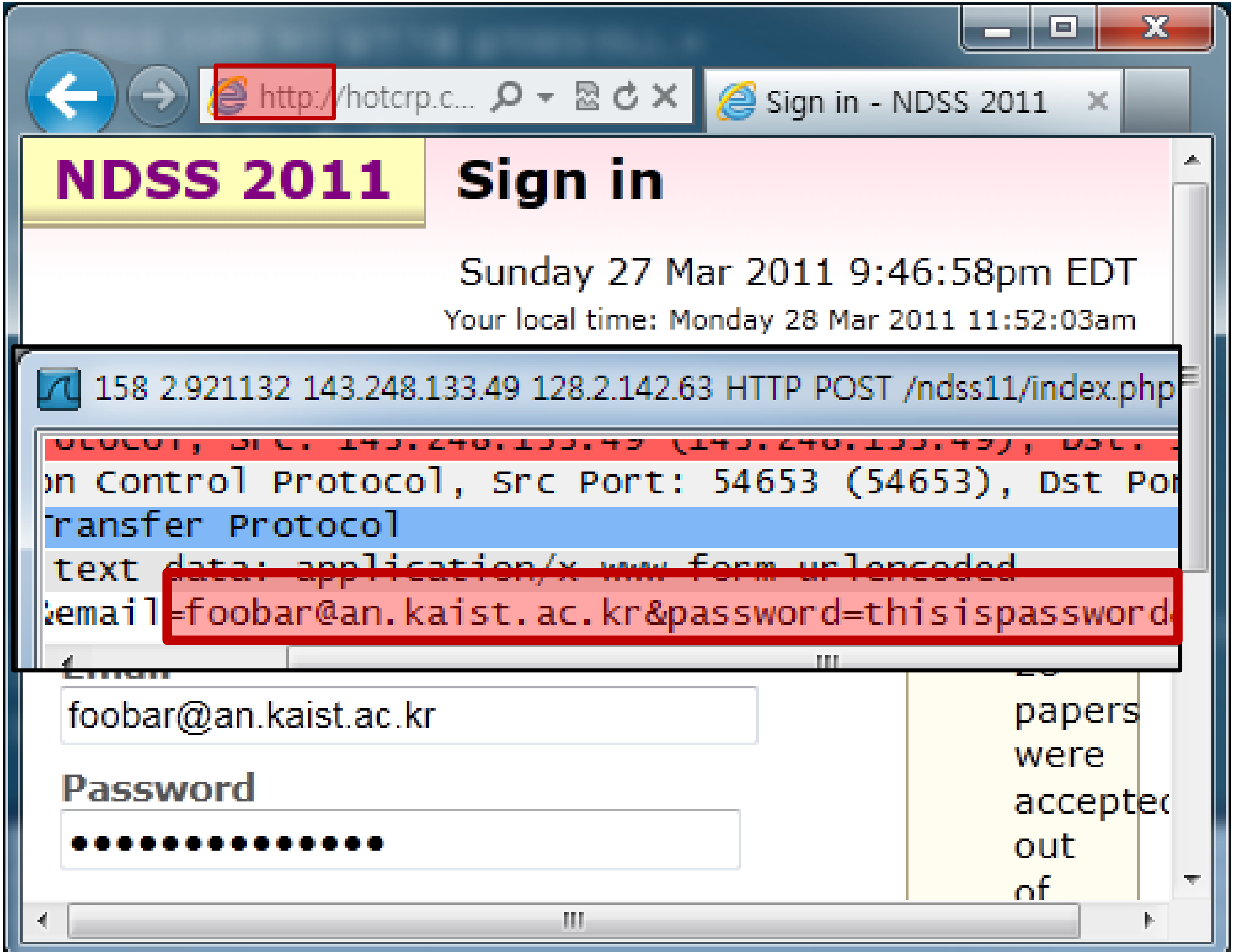
SSLShader: Cheap SSL Acceleration with Commodity Processors

Keon Jang[+], Sangjin Han[+], Seungyeop Han[*],
Sue Moon[+], and KyoungSoo Park[+]

KAIST[+] and University of Washington[*]

Security of Paper Submission Websites

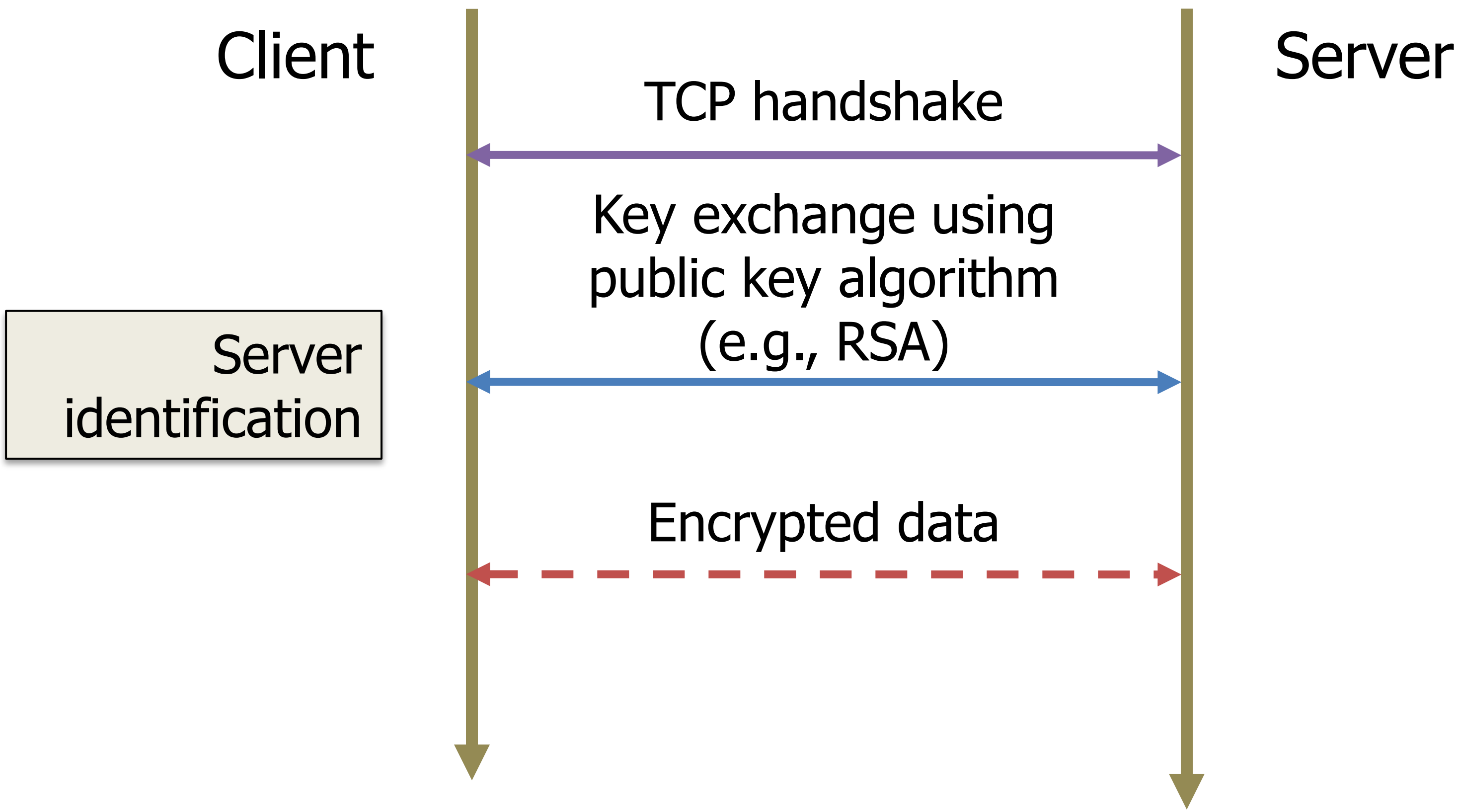


Security Threats in the Internet

- Public WiFi without encryption
  - Easy target that requires almost no effort
- Deep packet inspection by governments
  - Used for censorship
  - In the name of national security
- NebuAd’s targeted advertisement
  - Modify user’s Web traffic in the middle

Secure Sockets Layer (SSL)

- A de-facto standard for secure communication
  - Authentication, Confidentiality, Content integrity



SSL Deployment Status

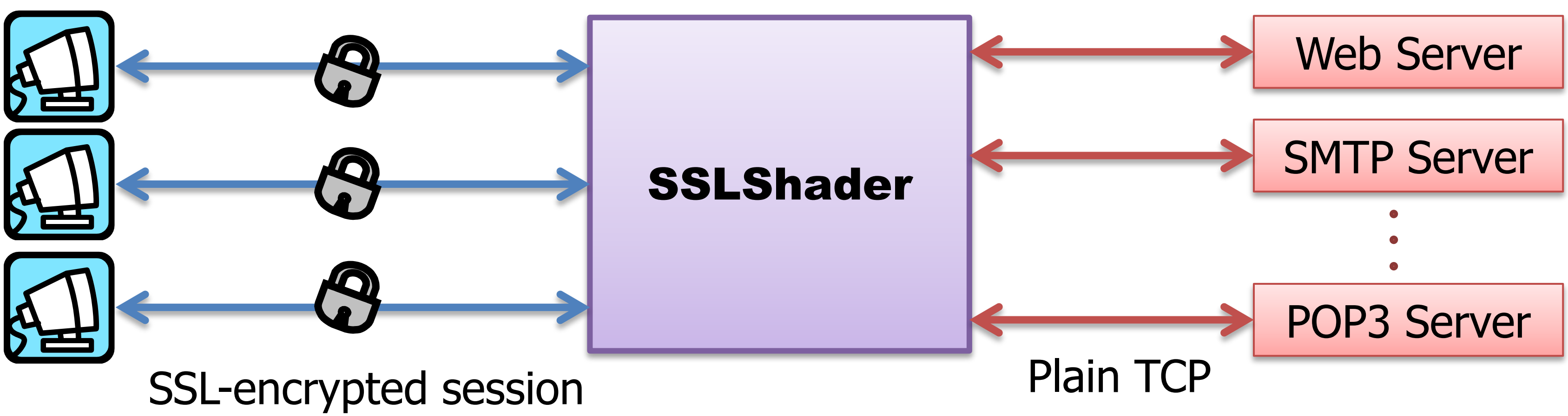
- Most of Web-sites are not SSL-protected
  - Less than 0.5%
    - [NETCRAFT Survey Jan '09]
- Why is SSL not ubiquitous?
  - Small sites: lack of recognition, manageability, etc.
  - Large sites: cost
    - SSL requires lots of computation power

SSL Computation Overhead

- Performance overhead (HTTPS vs. HTTP)
  - Connection setup 22x
  - Data transfer 50x
- Good privacy is expensive
  - More servers
  - H/W SSL accelerators
- Our suggestion:
  - Offload SSL computation to GPU

SSLShader

- SSL-accelerator leveraging GPU
  - High-performance
  - Cost-effective
- SSL reverse proxy
  - No modification on existing servers



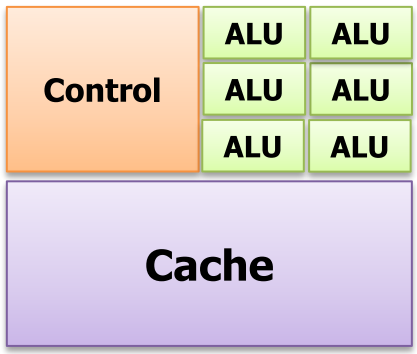
Our Contributions

- GPU cryptography optimization
  - The fastest RSA on GPU
  - Superior to high-end hardware accelerators
  - Low latency

- SSLShader
  - Complete system exploiting GPU for SSL processing
    - Batch processing
    - Pipelining
    - Opportunistic offloading
    - Scaling with multiple cores and NUMA nodes

CRYPTOGRAPHIC PROCESSING WITH GPU

How GPU Differs From CPU?



Intel Xeon 5650 CPU:
6 cores

NVIDIA GTX580 GPU:
512 cores

Instructions / sec

62×10^9 $<$ 870×10^9

Single Instruction Multiple Threads (SIMT)

Example code: vector addition (C = A + B)

CPU code

```
void VecAdd(
int *A, int *B, int *C, int N)
{
    //iterate over N elements
    for(int i = 0; i < N; i++)
        C[i] = A[i] + B[i]
}

VecAdd(A, B, C, N);
```

GPU code

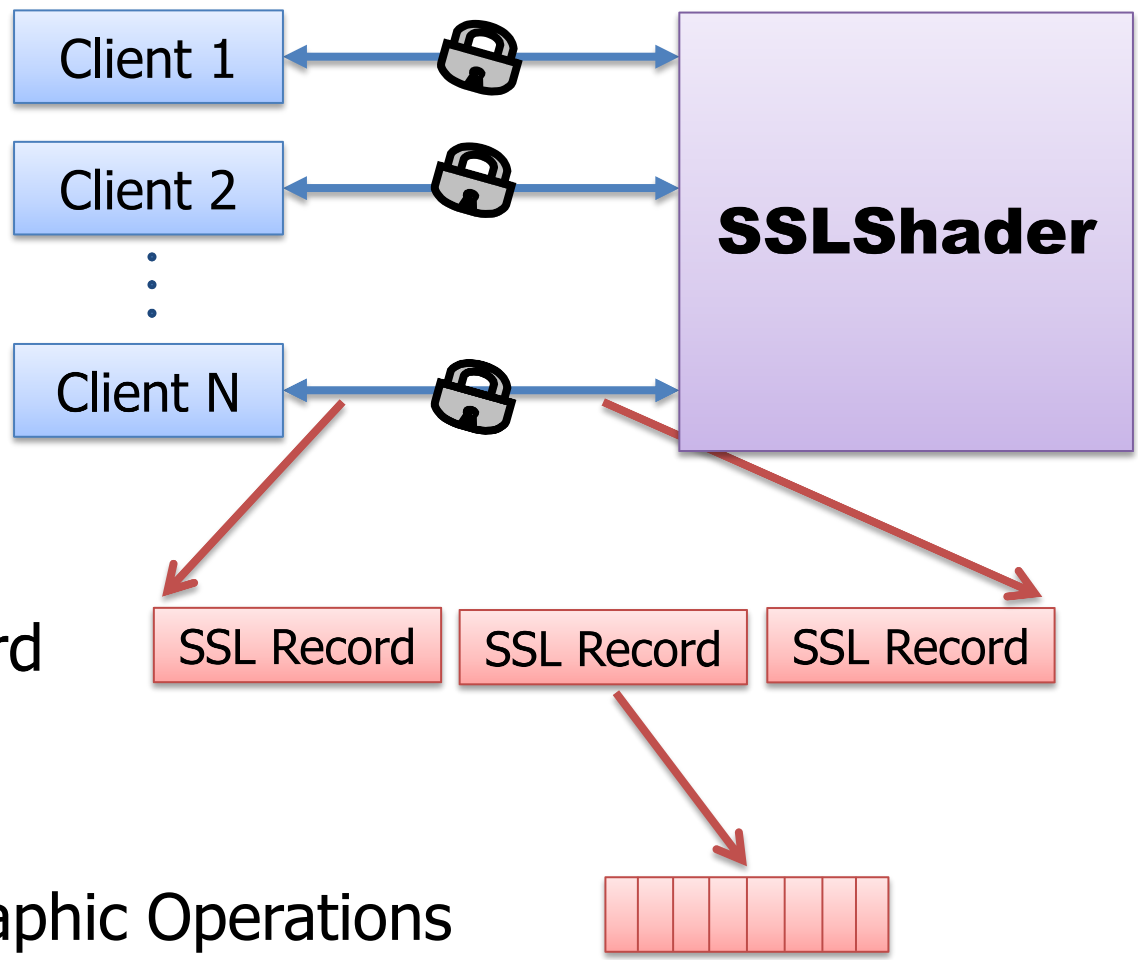
```
__global__ void VecAdd(
int *A, int *B, int *C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i]
}

//Launch N threads
VecAdd<<<1, N>>>(A, B, C);
```

Parallelism in SSL Processing

1. Independent Sessions

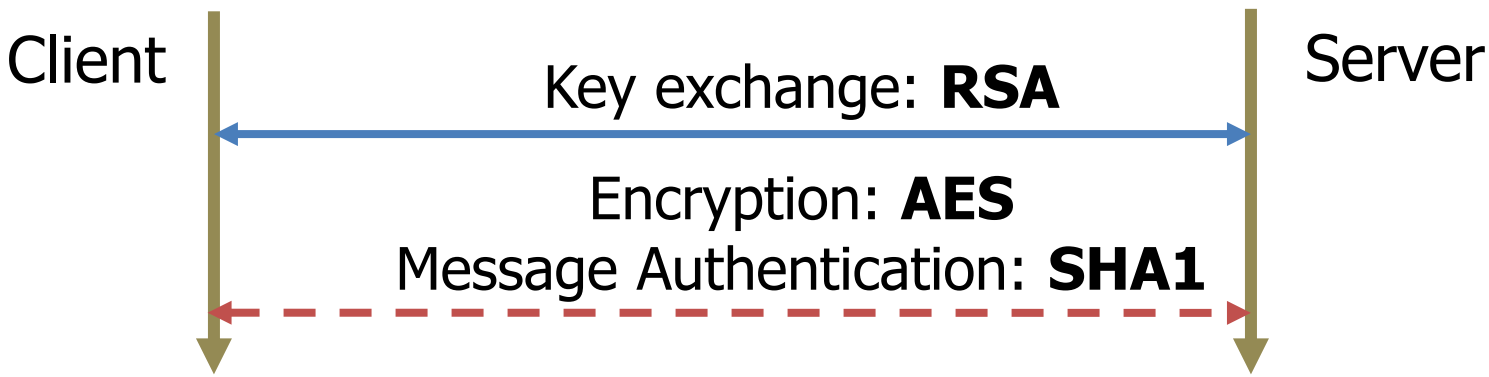
2. Independent SSL Record

3. Parallelism in Cryptographic Operations



Our GPU Implementation

- Choices of cipher-suite



- Optimization of GPU algorithms
  - Exploiting massive parallel processing
    - Parallelization of algorithms
    - Batch processing
  - Data copy overhead is significant
    - Concurrent copy and execution

Basic RSA Operations

- M: plain-text, C: cipher-text
- (e, n): public key, (d, n): private key

- Encryption:
  →Client

Small number: 3, 17, 65537

$$C = M^{e} \bmod n$$

1024/2048 bits integer (300 ~ 600 digits)

- Decryption:
  →Server

$$M = C^{d} \bmod n$$

Exponentiation → many multiplications

Breakdown of Large Integer Multiplication

Schoolbook multiplication

```
        649
x       627
----------
         63
        280
       4200
        180
        800
      12000
       5400
      32000
+    360000
----------
     406923
```

3 x 3 = 9 multiplications
9 addition of 6-digits integers

Accumulation is difficult to parallelize due to

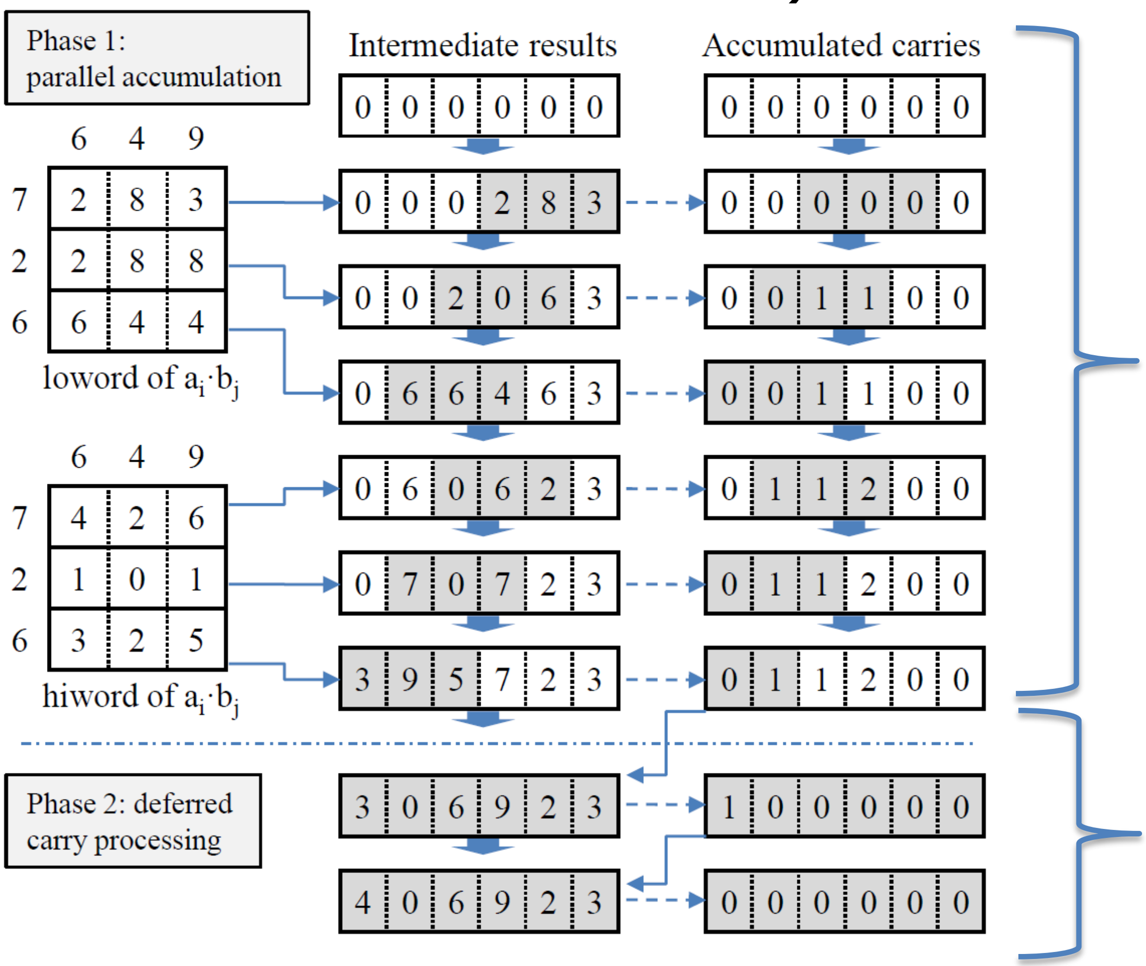
"overlapping digits"

"carry propagation"

$O(s)$ Parallel Multiplications

s = # of words in a large integer
(E.g., 1024-bits = 16 x 64 bits word)

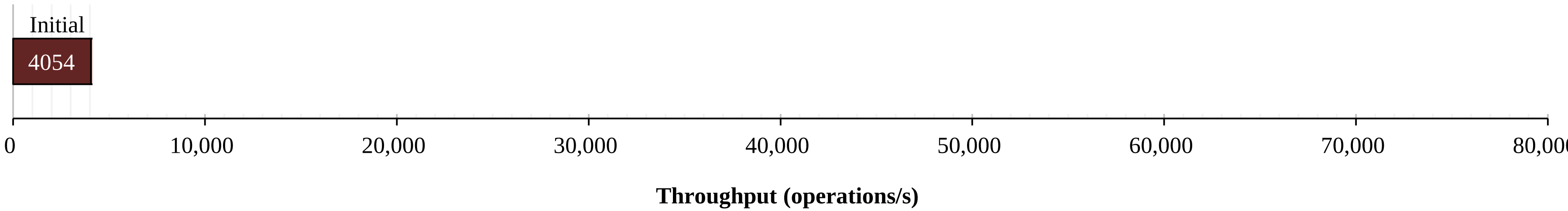
Example of
649 x 627 = 406,923



2s steps

1 or 2 steps
(s – 1 worst case)

More Optimizations on RSA



- Common optimizations for RSA
  - Chinese Remainder Theorem (CRT)
  - Montgomery Multiplication
  - Faster Calculation of M×n
  - Interleaving of T + M×n
  - Mixed-Radix Conversion Offloading
- GPU specific optimizations
  - Warp Utilization
  - Loop Unrolling
  - Elimination of Divergence
  - Avoiding Bank Conflicts
  - Instruction-Level Optimization

Read our paper for details ☺

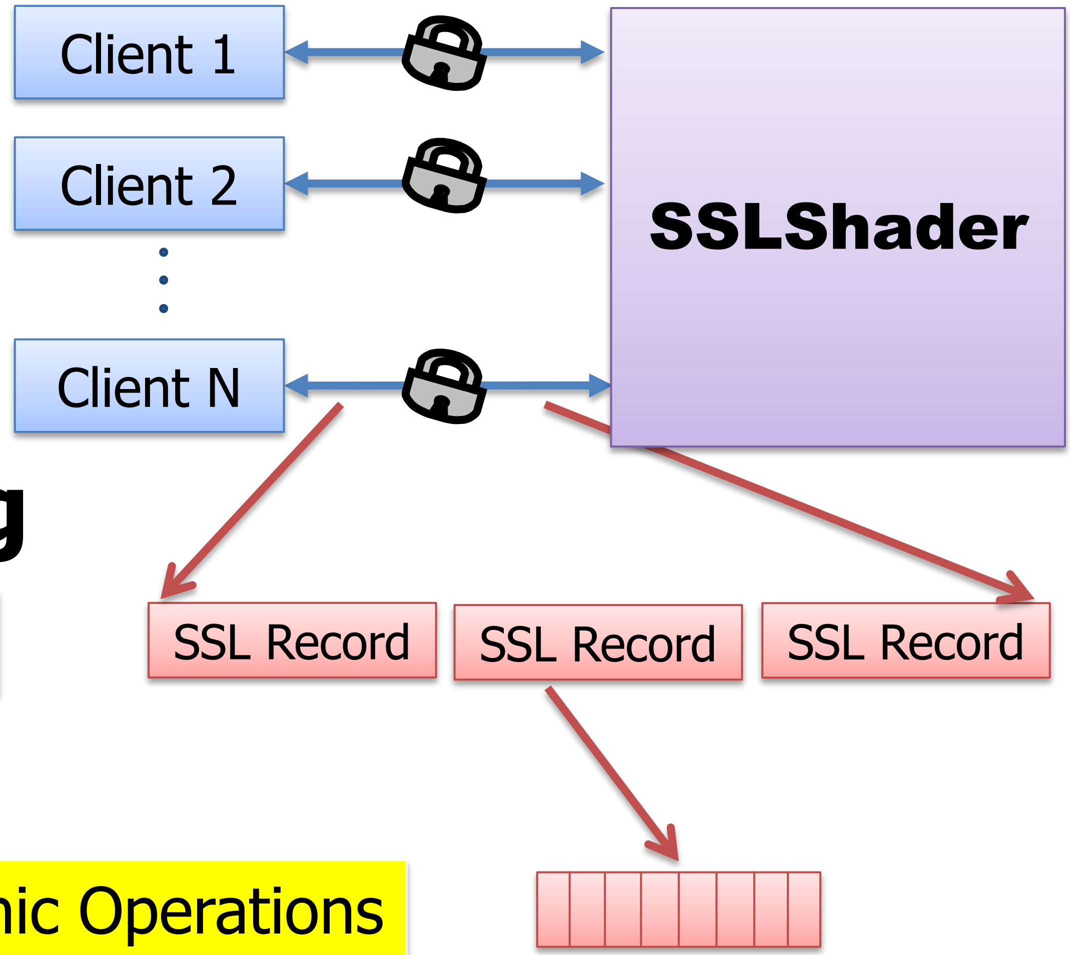
Parallelism in SSL Processing

1. Independent Sessions

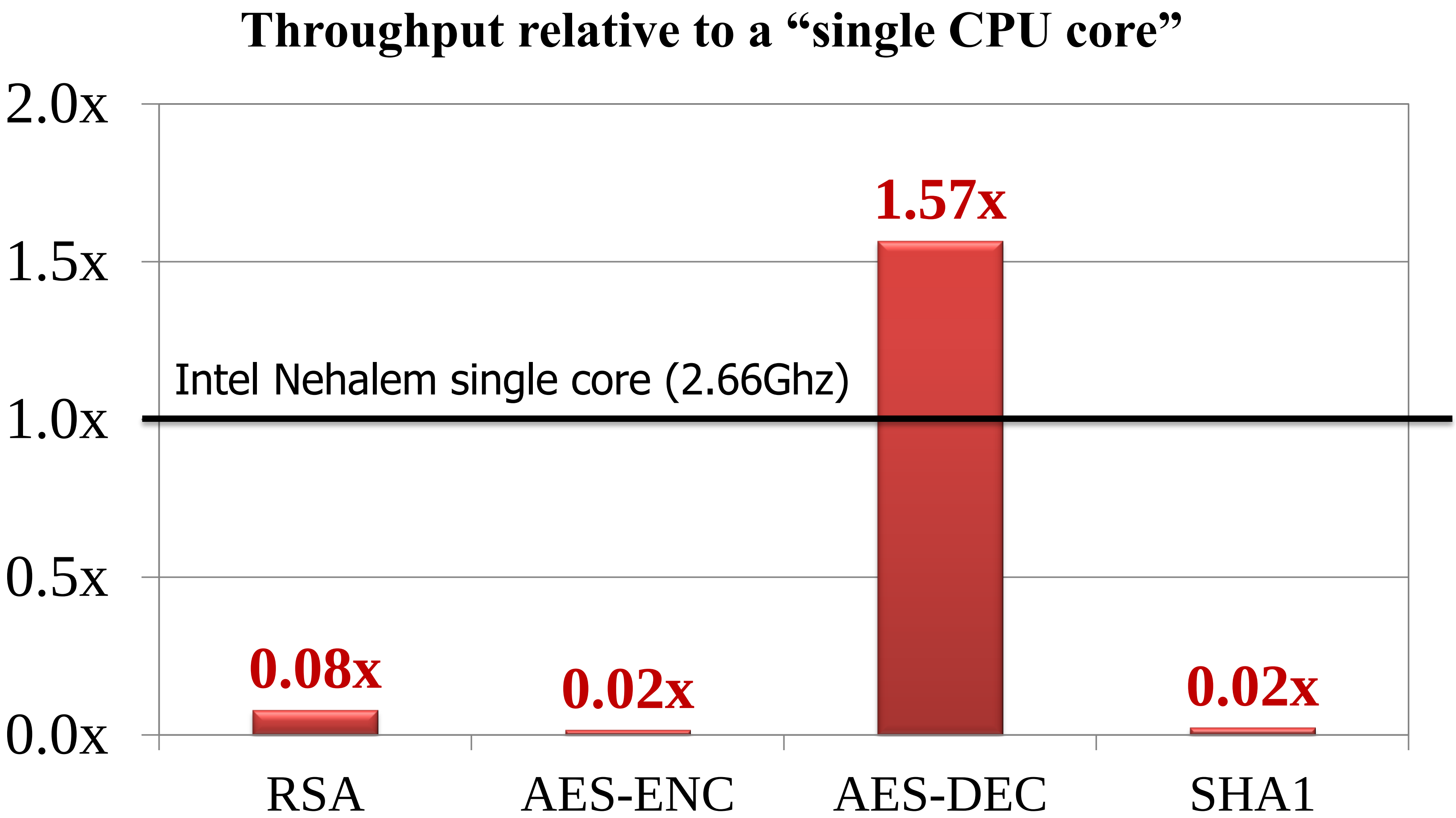
Batch Processing

2. Independent SSL Record

3. Parallelism in Cryptographic Operations

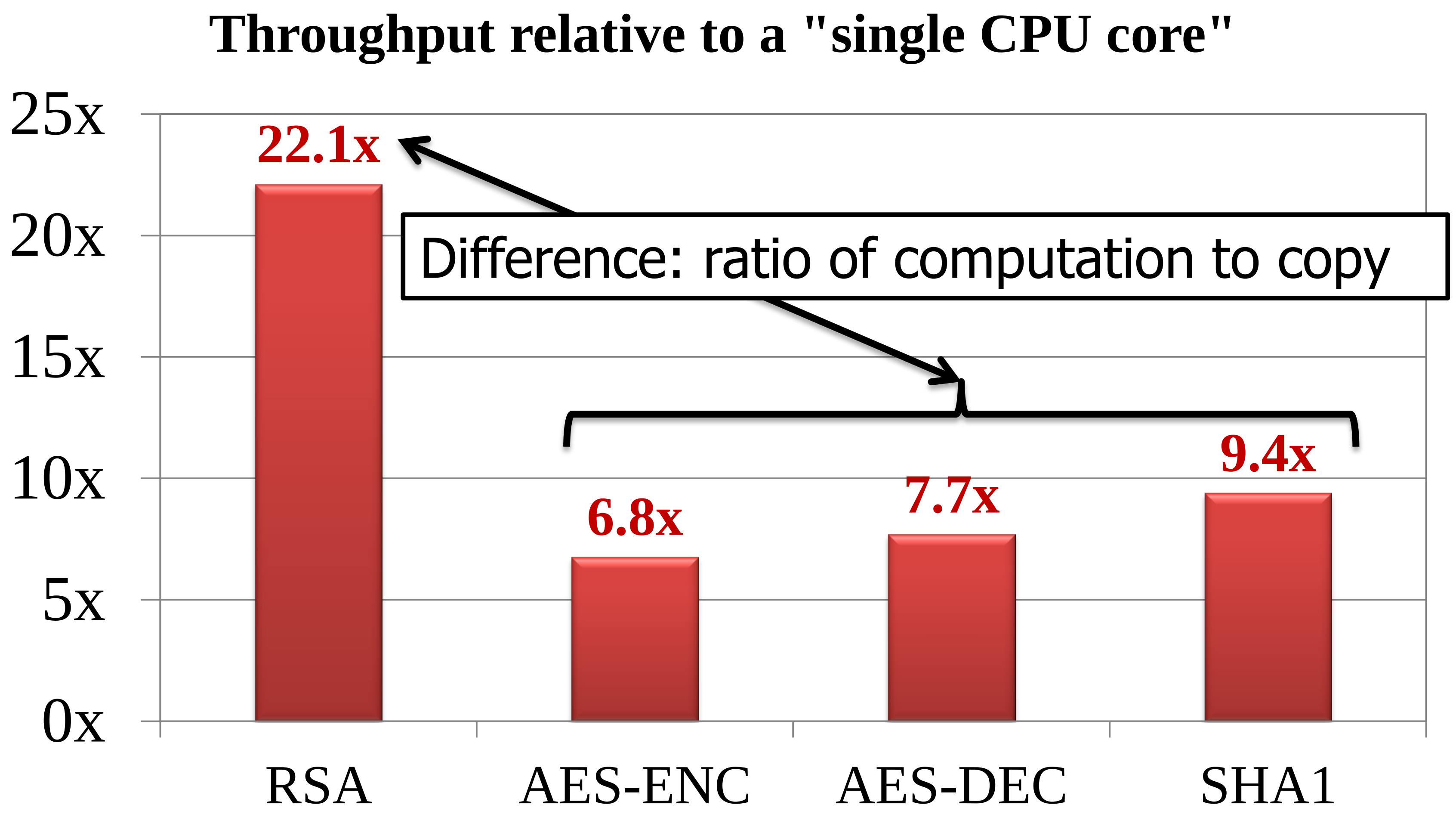


GTX580 Throughput w/o Batching



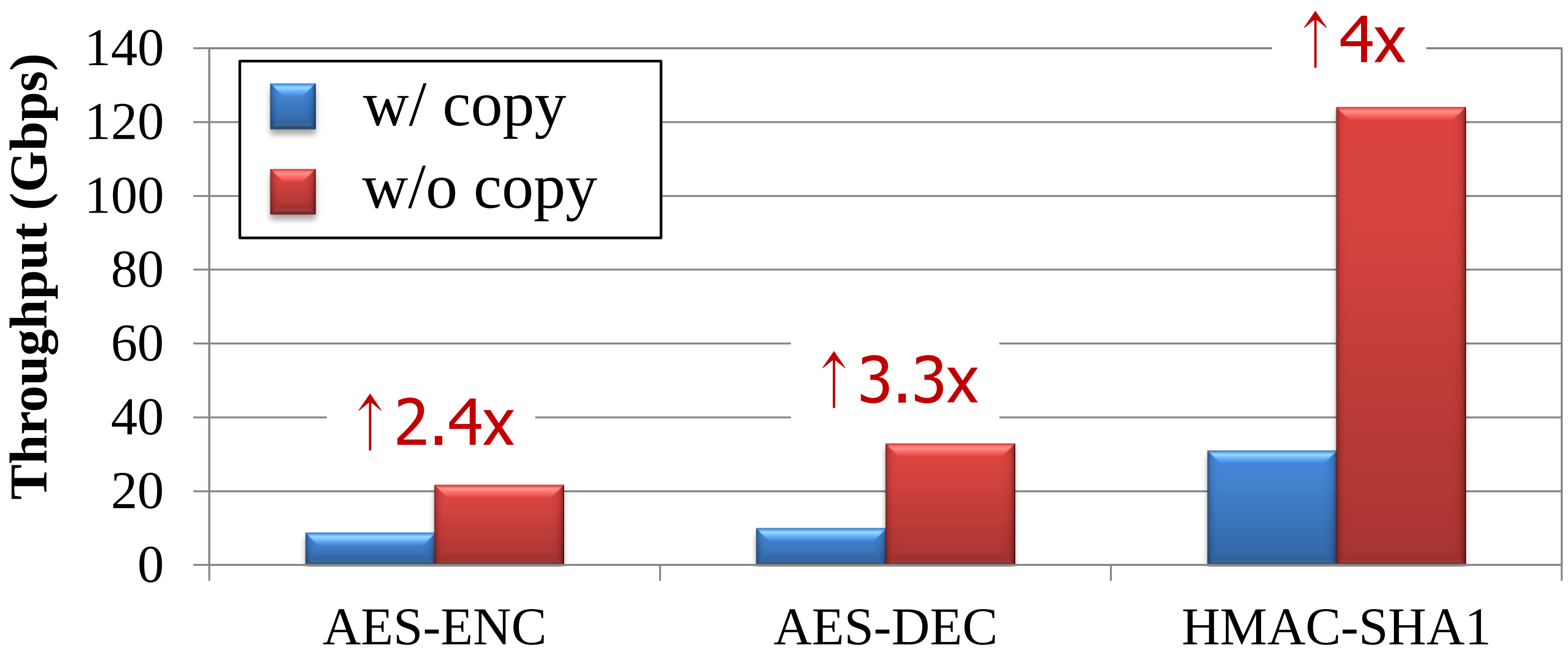
GTX580 Throughput w/ Batching

Batch size: **32~4096** depending on the algorithm



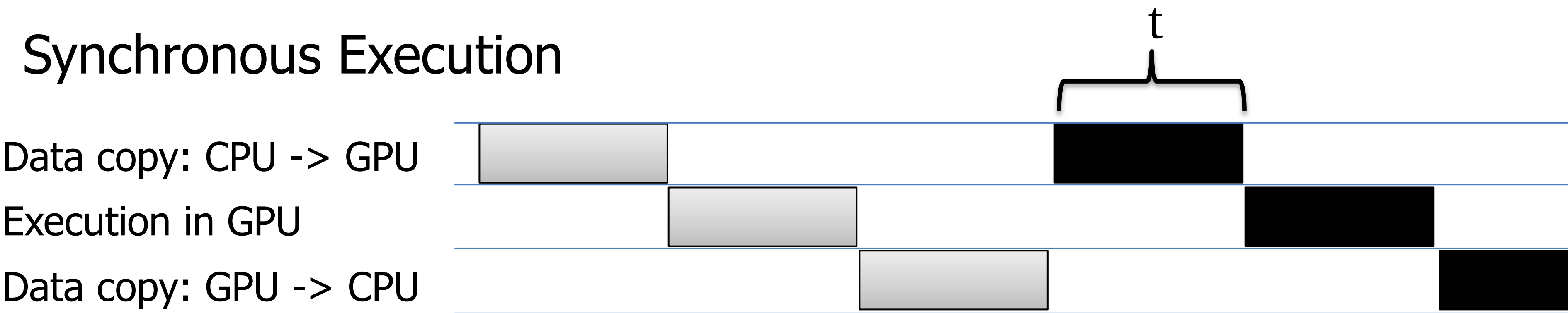
Copy Overhead in GPU Cryptography

- GPU processing works by
  - Data copy: CPU → GPU
  - Execution in GPU
  - Data copy: GPU -> CPU



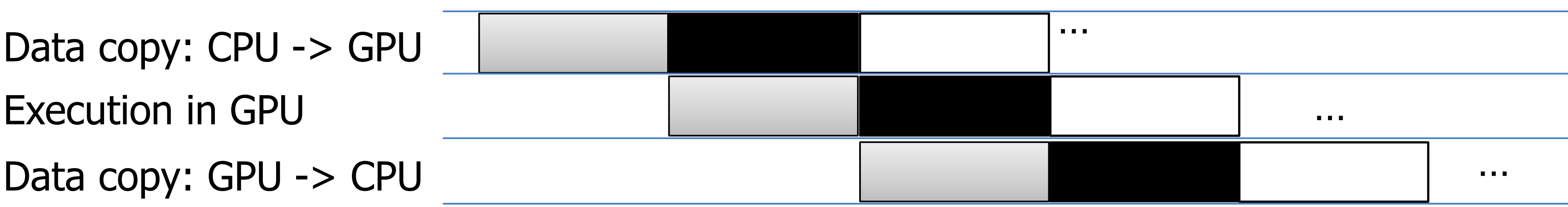
Hiding Copy Overhead

Synchronous Execution



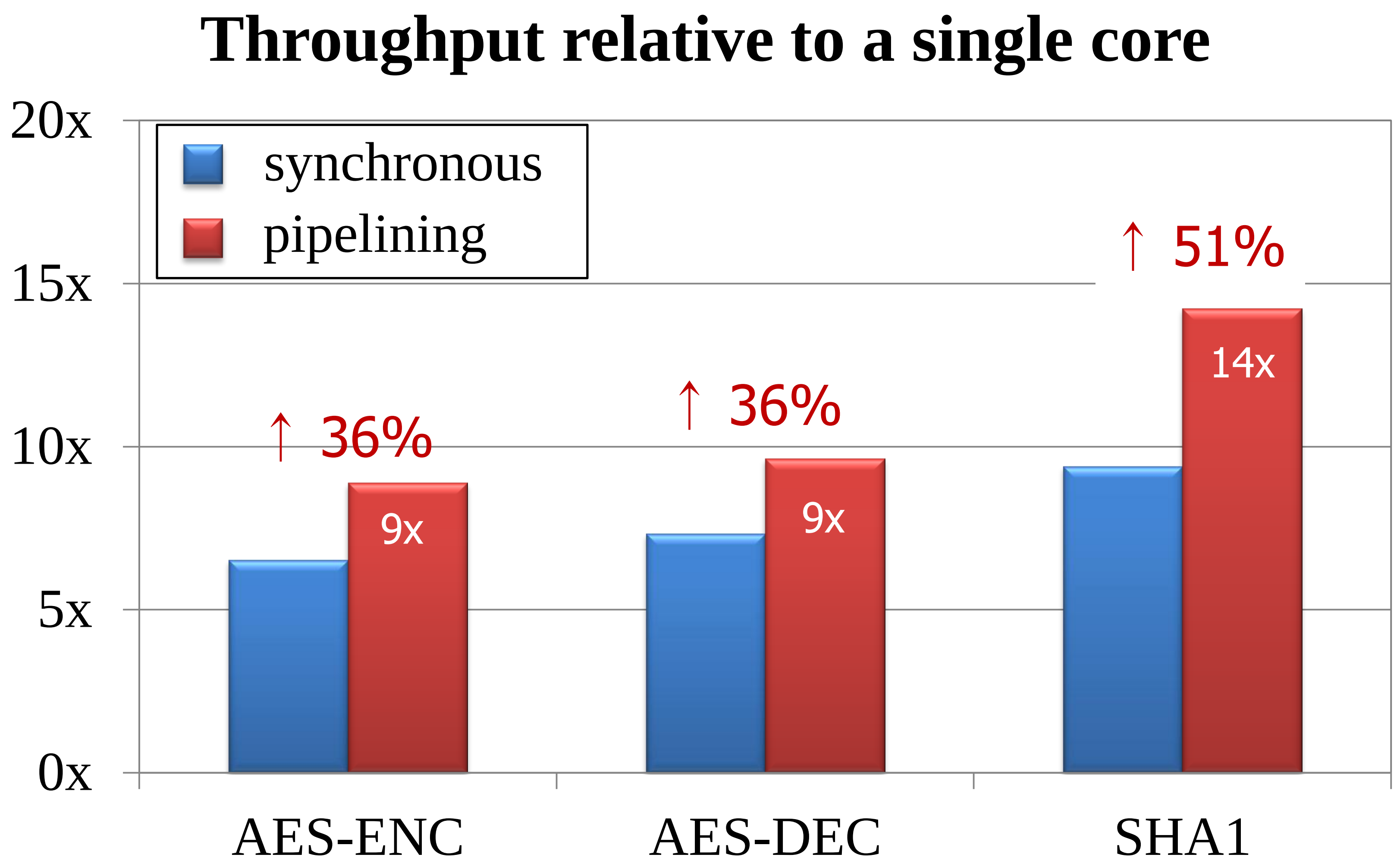
Processing time : 3t

Pipelining



Amortized processing time : t

GTX580 Performance w/ Pipelining



Summary of GPU Cryptography

- Performance gain from GTX580
  - GPU performs as fast as 9 ~ 28 CPU cores
  - Superior to high-end hardware accelerators

	RSA-1024 (ops/sec)	AES-ENC (Gbps)	AES-DEC (Gbps)	SHA1 (Gbps)
GTX580	91.9K	11.5	12.5	47.1
CPU core	3.3K	1.3	1.3	3.3

- Lessons
  - Batch processing is essential to fully utilize a GPU
  - AES and SHA1 are bottlenecked by data copy
    - PCIe 3.0
    - Integrated GPU and CPU

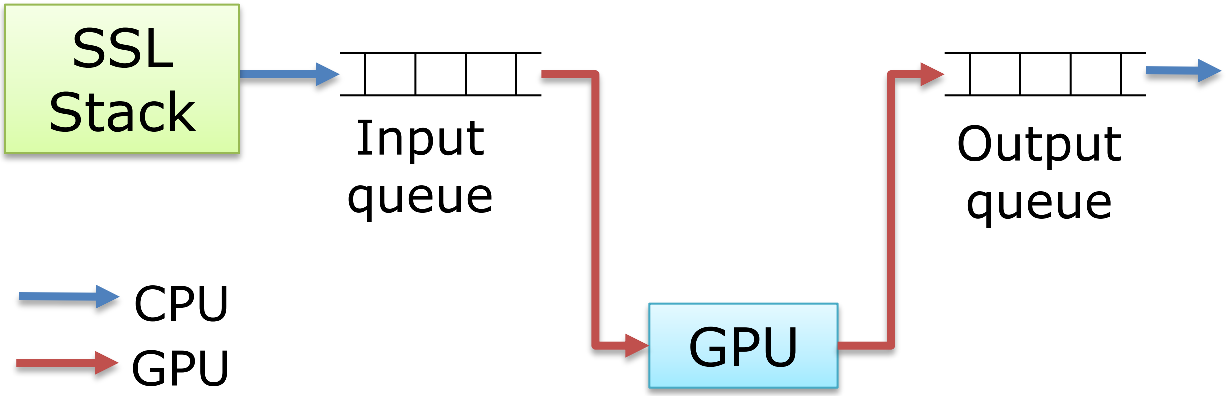
BUILDING SSL-PROXY THAT LEVERAGES GPU

SSLShader Design Goals

- Use existing application without modification
  - SSL reverse proxy
- Effectively leverage GPU
  - Batching cryptographic operations
  - Load balancing between CPU and GPU
- Scale performance with architecture evolution
  - Multi-core CPUs
  - Multiple NUMA nodes

Batching Crypto Operations

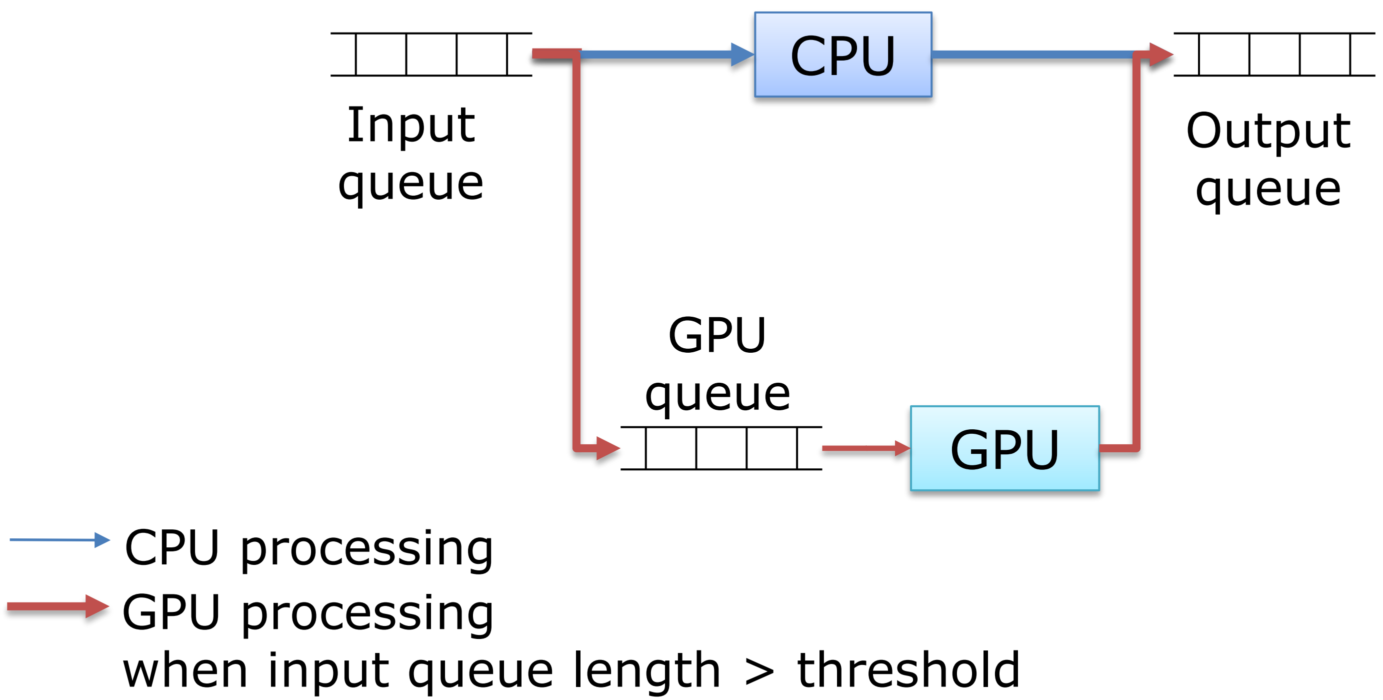
- Network workloads vary over time
  - Waiting for fixed batch size doesn't work



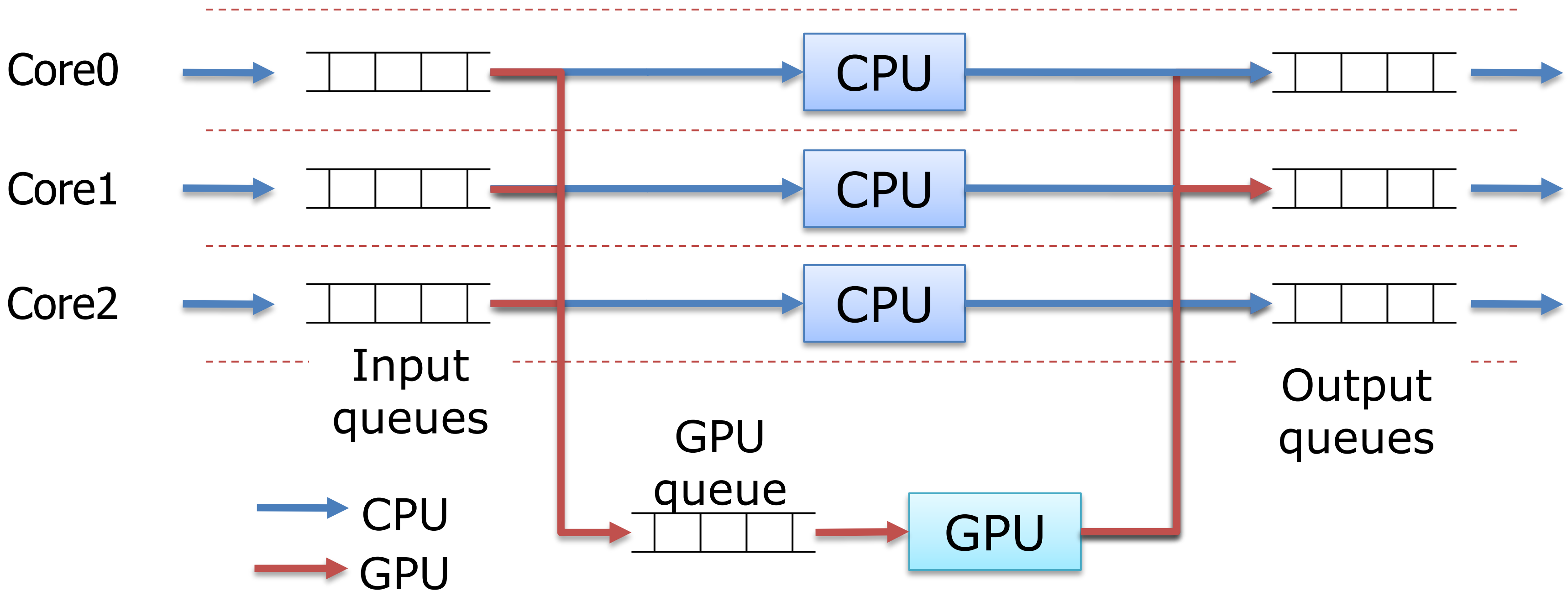
- Batch size is dynamically adjusted to queue length

Balancing Load Between CPU and GPU

- For small batch, CPU is faster than GPU
  - Opportunistic offloading

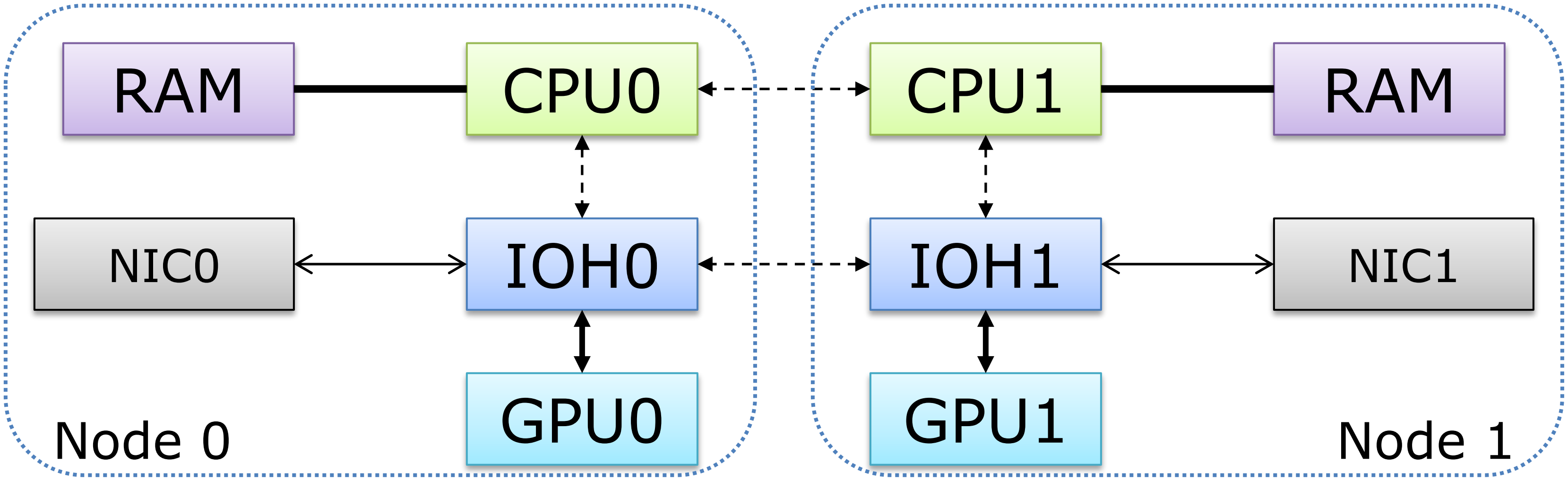


Scaling with Multiple Cores



- Per-core worker threads
  - Network I/O, cryptographic operation
- Sharing a GPU with multiple cores
  - More parallelism with larger batch size

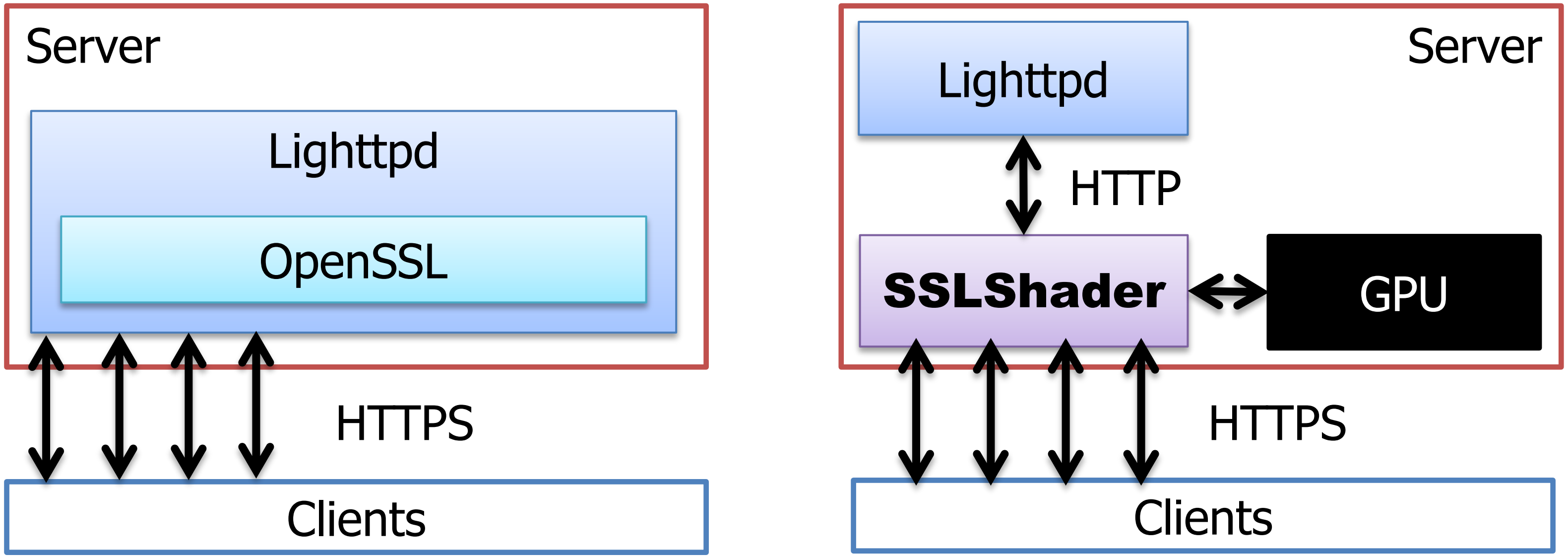
Scaling with NUMA systems



- A process = worker threads + a GPU thread
  - Separate process per NUMA node
  - Minimizes data sharing across NUMA nodes

Evaluation

- Experimental configurations



Server Specification

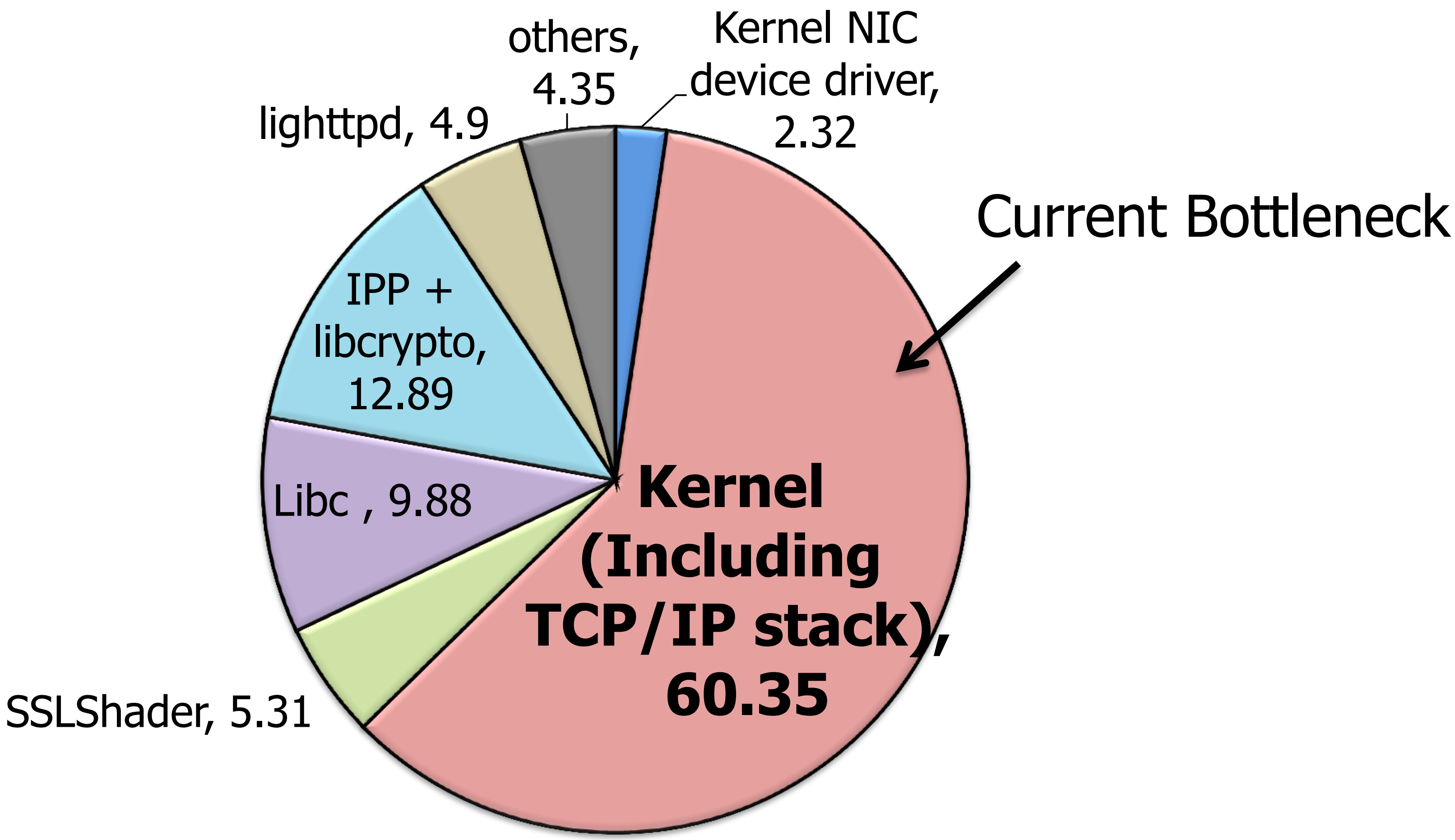
	Model	Spec	Qty
CPU	Intel X5650	2.66Ghz x 6 croes	2
GPU	NVIDIA GTX580	1.5Ghz x 512 cores	2
NIC	Intel X520-DA2	10GbE x 2	2

Evaluation Metrics

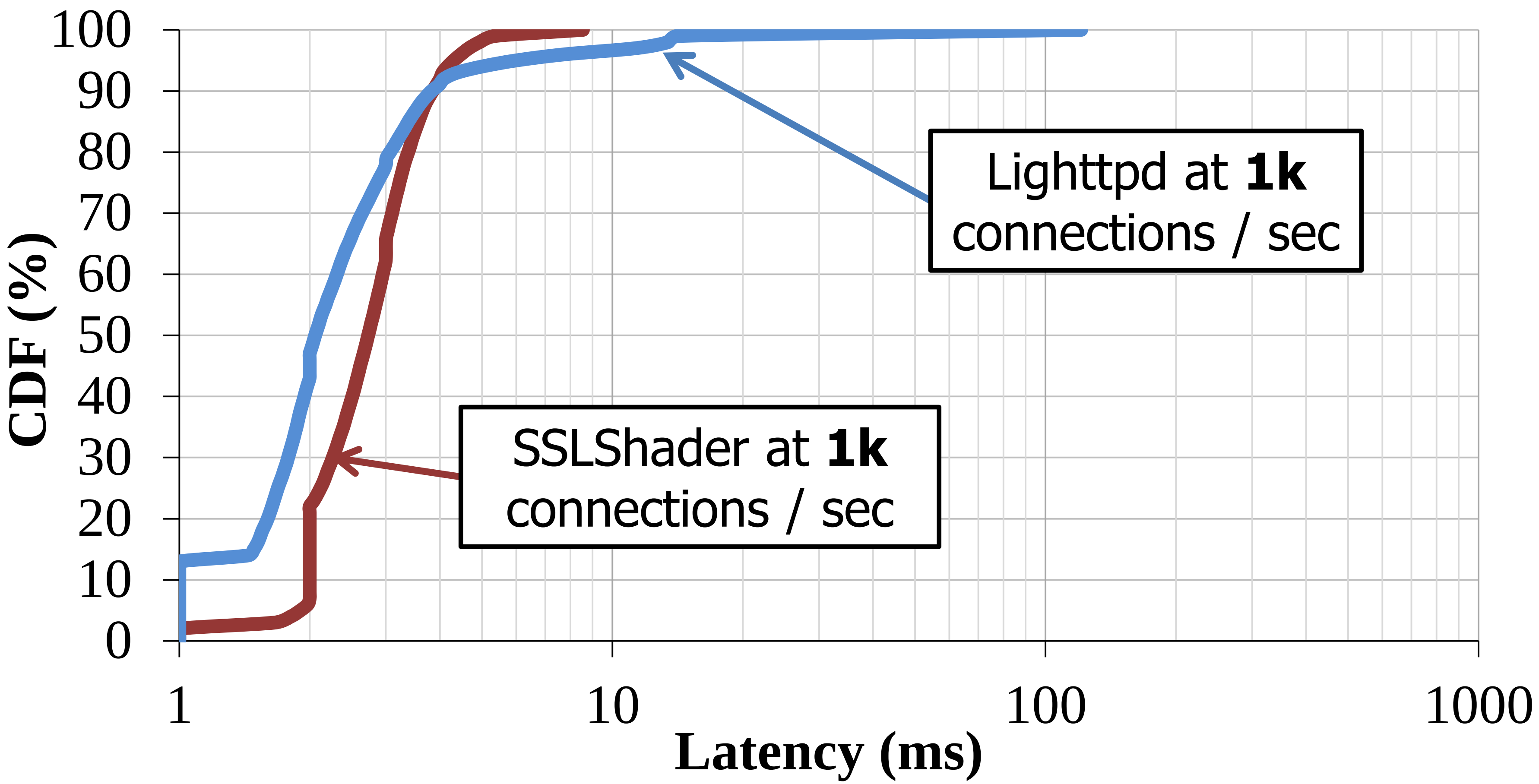
- HTTPS connection handling performance
  - Use small content size
  - Stress on RSA computation
- Latency distribution at different loads
  - Test opportunistic offloading
- Data transfer rate at various content size

HTTPS Connection Rate

CPU Usage Breakdown (RSA 1024)

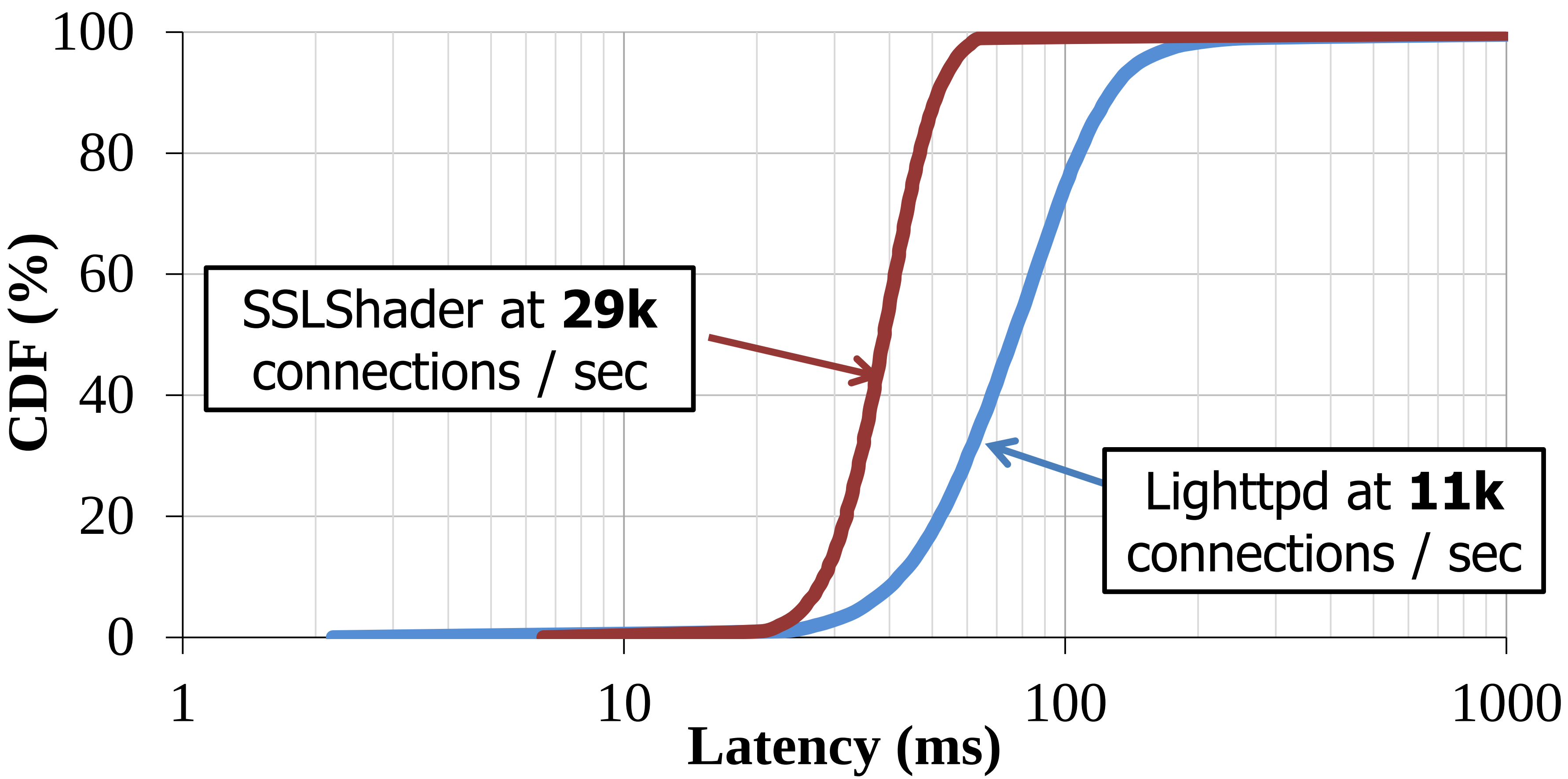


Latency at Light Load



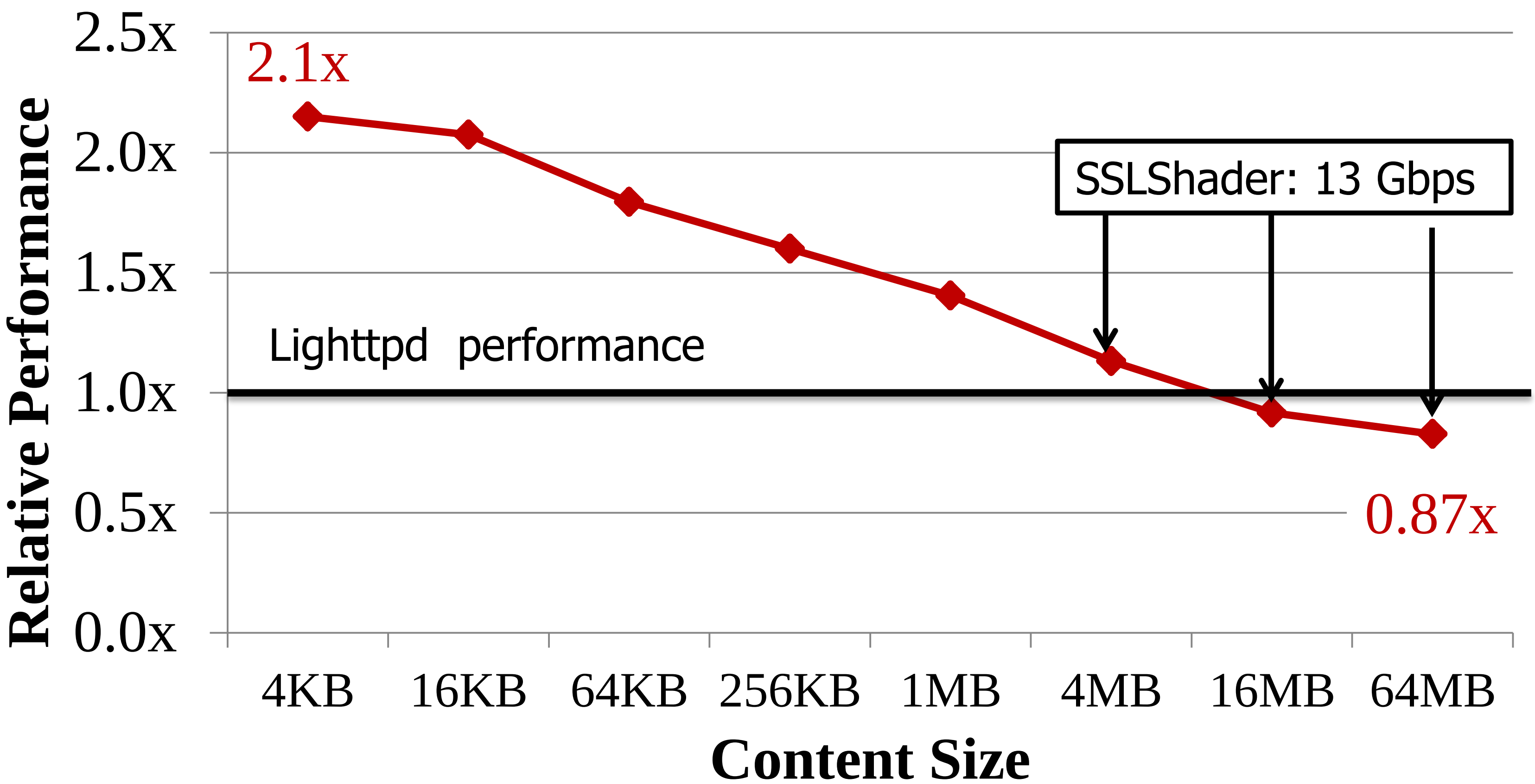
Similar latency at light load

Latency at Heavy Load



Lower latency and higher throughput at heavy load

Data Transfer Performance



Typical web content size is under 100KB

CONCLUSIONS

Summary

- Cryptographic algorithms in GPU
  - Fast RSA, AES, and SHA1
  - Superior to high-end hardware accelerators

- **SSLShader**
  - Transparent integration
  - Effective utilization of GPU for SSL processing
    - Up to 6x connections / sec
    - 13 Gbps throughput

Linux network stack performance

Copy overhead

For more details

https://shader.kaist.edu/sslshader

QUESTIONS?

THANK YOU!