Aalto University

Research Institute for Collaborative Autonomous and Resilient Systems

Trustworthy & Accountable Function-as-a-Service

Fritz Alder, N. Asokan, Arseny Kurnikov, Andrew Paverd, Michael Steiner

Function-as-a-Service (FaaS)

Recent instantiation of “serverless computing”

- Customer specifies the function
- Service provider manages runtime, scaling, load-balancing etc.

Differences to Infrastructure-as-a-Service (IaaS)

- Relatively short-running function invocations
- Stateless functions (storage provided by separate service)

Motivation

FaaS is available from established cloud providers

Usual security concerns of cloud computing still apply:

- Confidentiality of data
- Integrity of computation

Motivation

The Register®
Biting the hand that feeds IT

DATA CENTRE | SOFTWARE | SECURITY | **DEVOPS** | BUSINESS | PERSONAL TECH | SCIENCE | EMERGENT TECH | BOOTNOTES | LECTURES

DevOps

Whisk-y business: How Apache OpenWhisk hole left IBM Cloud Functions at risk of hijacking

Now-patched vulnerability let attackers overwrite code

By Shaun Nichols in San Francisco 24 Jul 2018 at 13:00 3 SHARE ▼

IBM has patched a critical vulnerability in its Cloud Functions platform that would have allowed miscreants to remotely overwrite customers' code – and execute malicious commands to hijack services.

https://www.theregister.co.uk/2018/07/24/apache_ibm_cloud_vulnerable/

Motivation

FaaS is available from established cloud providers

Usual security concerns of cloud computing still apply:

- Confidentiality of data
- Integrity of computation

More accurate resource usage measurements required:

- Sub-second compute time measurements

Currently achieved via existing reputational trust, but can we do better?

Motivation

FaaS can also be provided by non-traditional service providers

- Data centres with spare capacity
- Individuals with powerful PCs (e.g. gamers)

Open source frameworks available

https://openwhisk.apache.org/

Multiple start-ups in this space

https://golem.network/

https://ankr.network

Motivation

FaaS can also be provided by non-traditional service providers

- Data centres with spare capacity
- Individuals with powerful PCs (e.g. gamers)

Heightened security concerns:

- Service provider identity/location may be unknown
- Service provider may not have security expertise

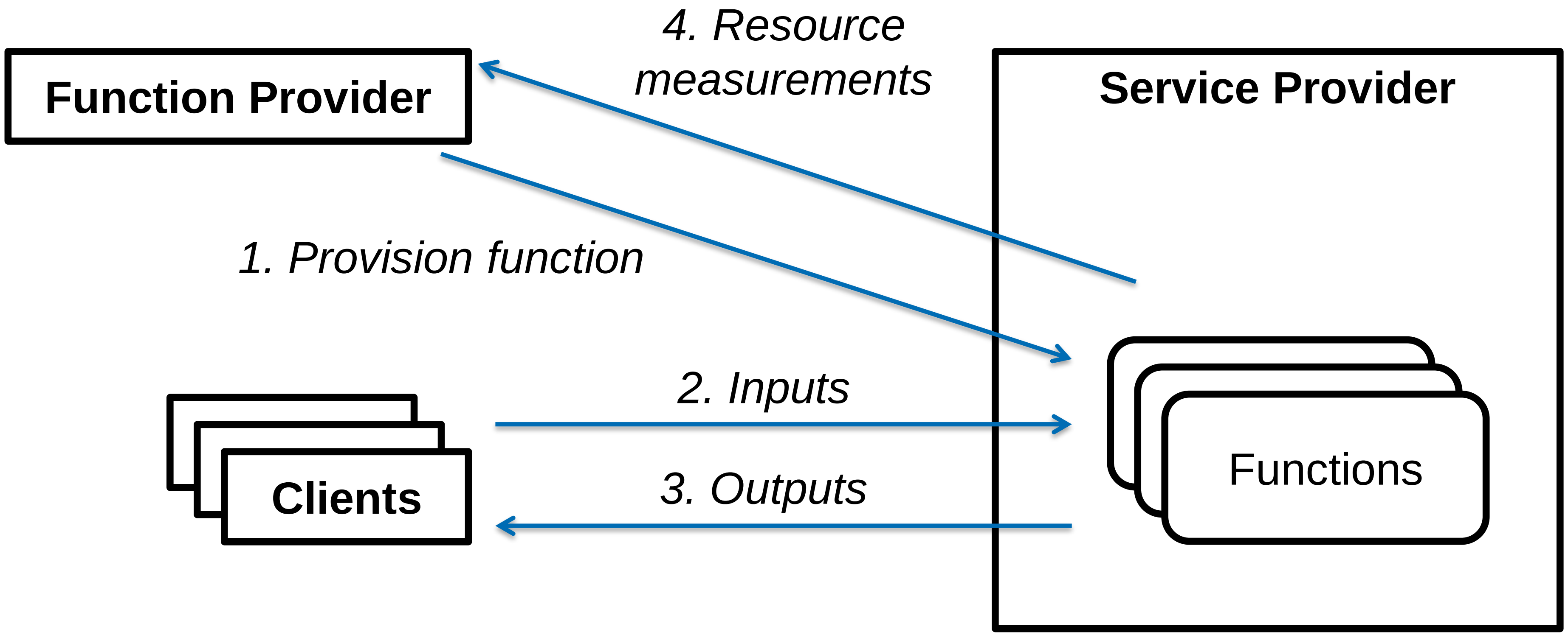
Very few disincentives for cheating:

- Malicious service provider might inflate resource usage measurements

No reputational trust has been established

System Model & Requirements

System model



Adversary model

Two types of adversaries:

Service provider

- Learn inputs and outputs of function invocations
- Modify inputs and outputs, or execute the function incorrectly
- Overcharge the function provider
  - Falsely inflate resource usage measurements
  - Create fake function invocations

Function provider

- Under-pay the service provider for resources used by the function

Requirements

R1 - Security

- Service provider cannot modify inputs or outputs of a function invocation
- Client assured that output is result of correct execution of intended function on supplied inputs

R2 - Privacy

- Service provider cannot learn inputs or outputs of a function invocation

R3 - Measurement accuracy

- Resource measurements must have sufficient accuracy for FaaS billing

R4 - Measurement veracity

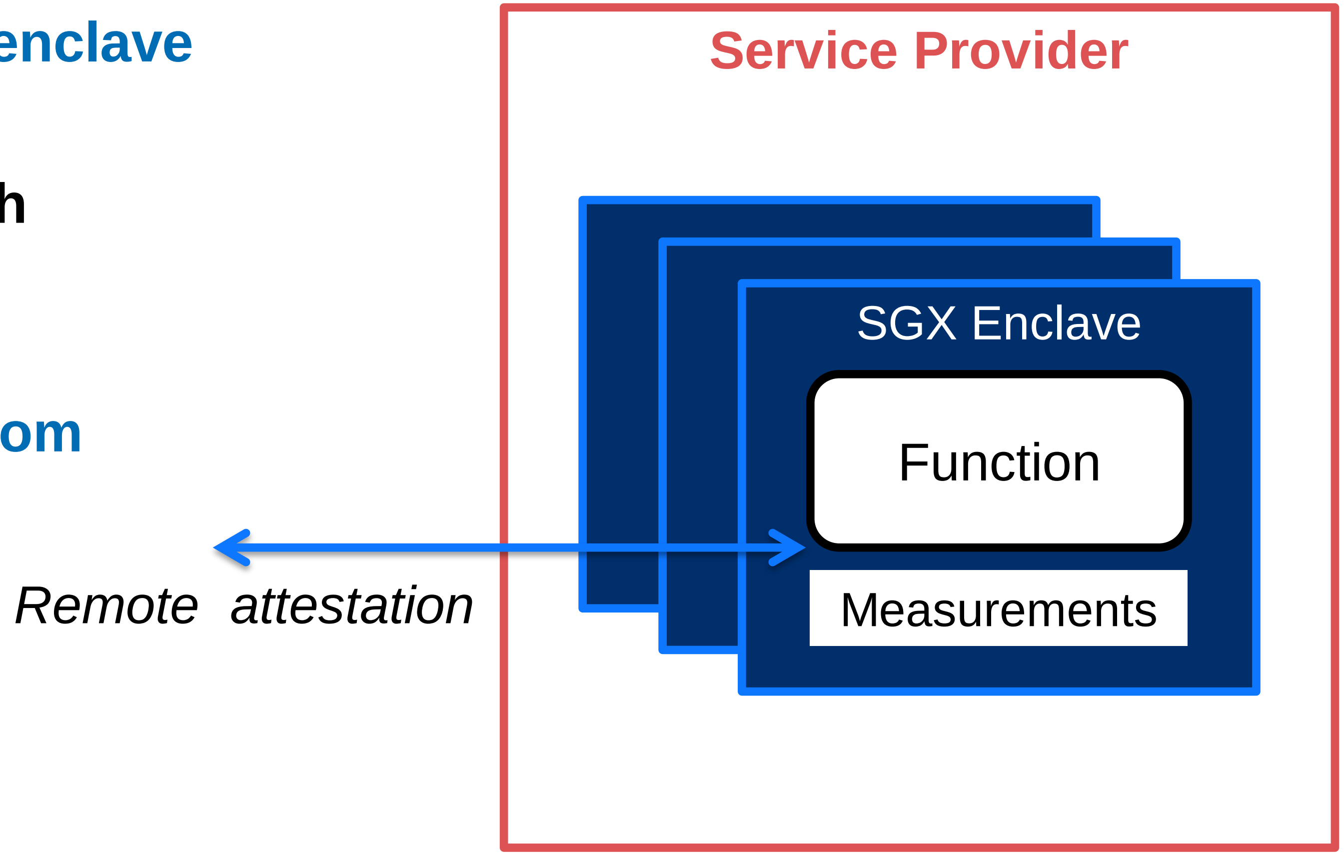
- All parties must be able to verify authenticity of resource measurements

Preliminary design

Execute each function in an SGX enclave

Use remote attestation to establish secure communication channels

Measure resource consumption from within the enclave

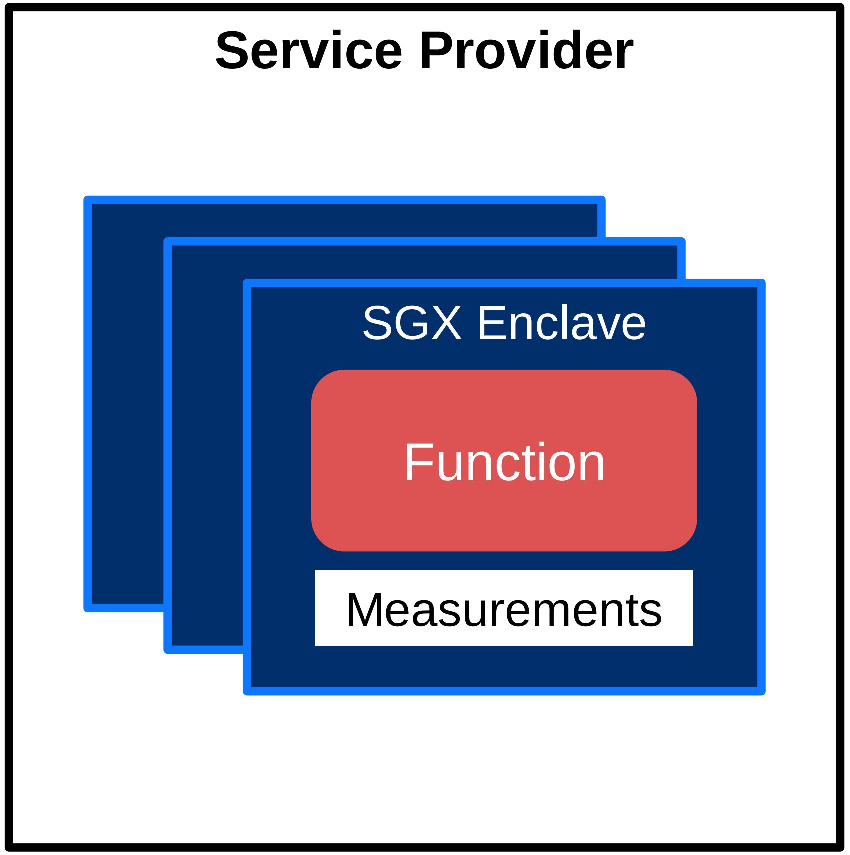


Design Challenges

Challenge: Sandboxing untrusted functions

Malicious function provider could attempt to reduce in-enclave measurements

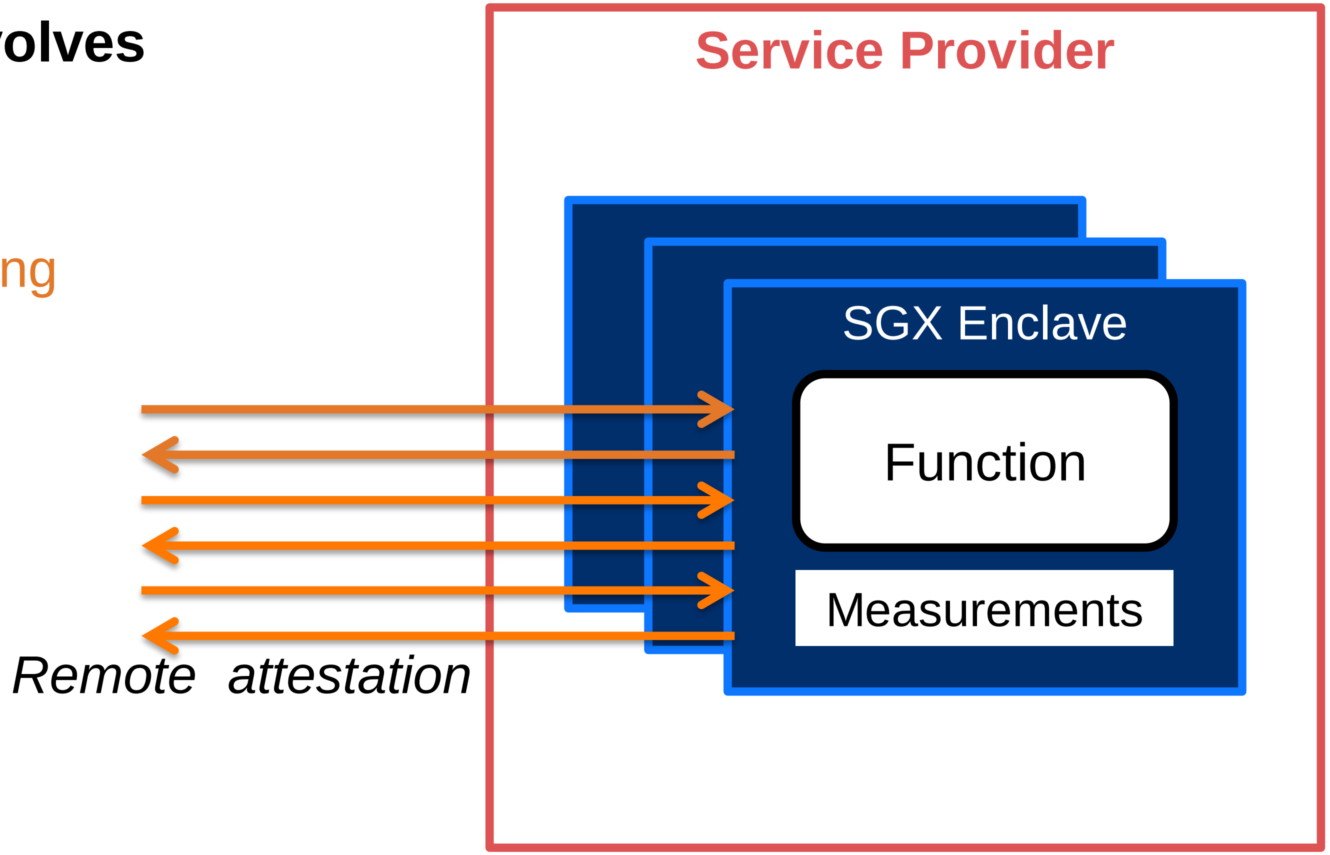
- No protection from code in the same enclave



Challenge: Attesting worker enclaves

Default SGX remote attestation involves multiple message round-trips

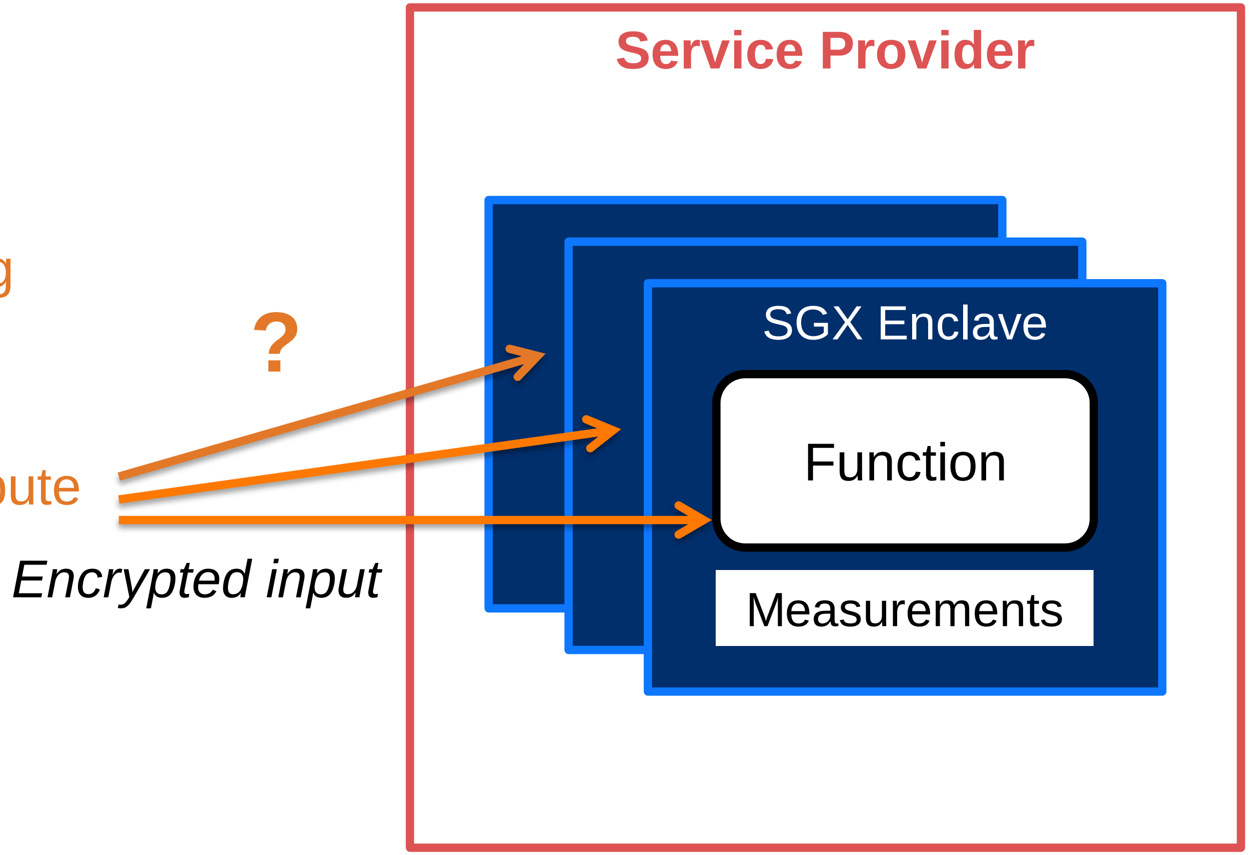
- Overhead and latency for short-running functions is too high
- Must be repeated for each enclave



Challenge: Encrypting client input

Function invocation is a one-shot message, including (encrypted) input

- Client must encrypt input *before* knowing which enclave will run the function
- Cannot rely on service provider to distribute keys to worker enclaves

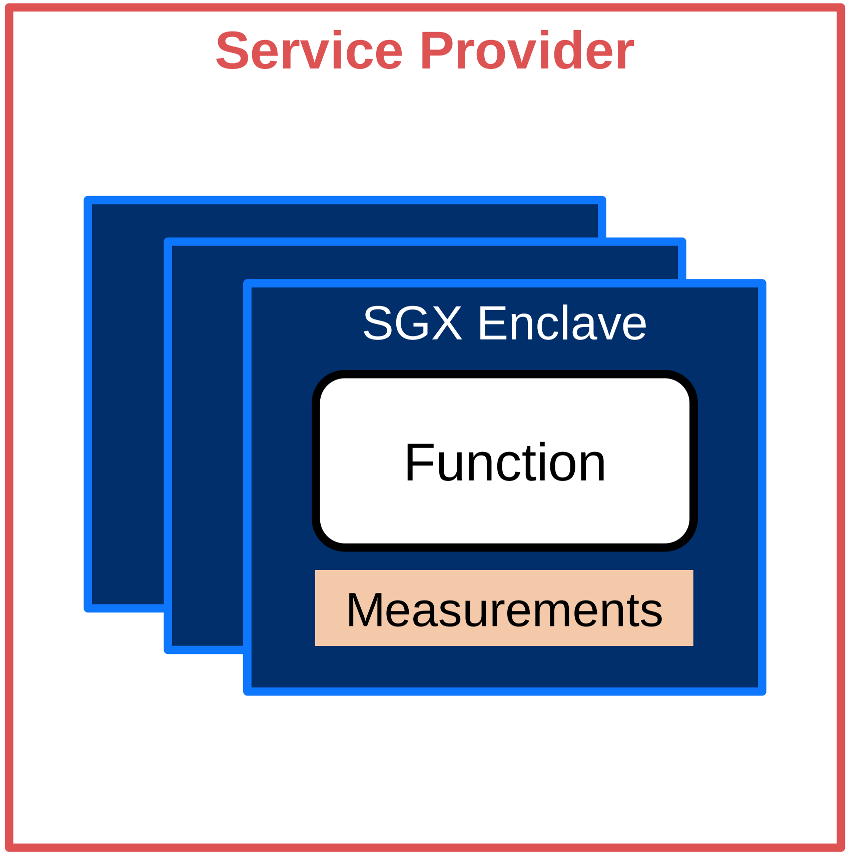


Challenge: Measuring time in enclaves

CPU instructions
RDTSC: read timestamp counter
AEX: asynchronous enclave exit
ERESUME: resume enclave

SGX enclave cannot reliably measure its own running time

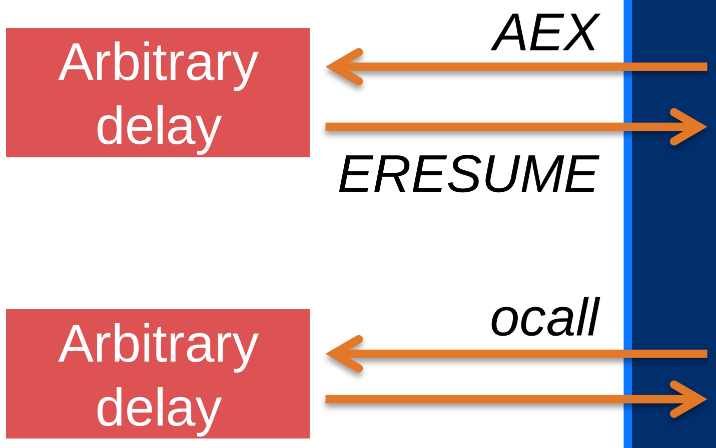
- `RDTSC` value can be manipulated by VMM
- `sgx_get_trusted_time()` can be arbitrarily delayed
- Enclaves can be transparently interrupted (AEX) and resumed (ERESUME)



Challenge: Measuring time in enclaves

VERICOUNT:

call `sgx_get_trusted_time()` at ecall start & end




```
ecall_to_measure()
{
    t1 = sgx_get_trusted_time();
        .
    [function code]
        .
        .
        .
        .
    t2 = sgx_get_trusted_time();
    time = t2 – t1;
}
```

Tople et al., “VeriCount: Verifiable Resource Accounting Using Hardware and Software Isolation”, ACNS 2018

S-FaaS Architecture

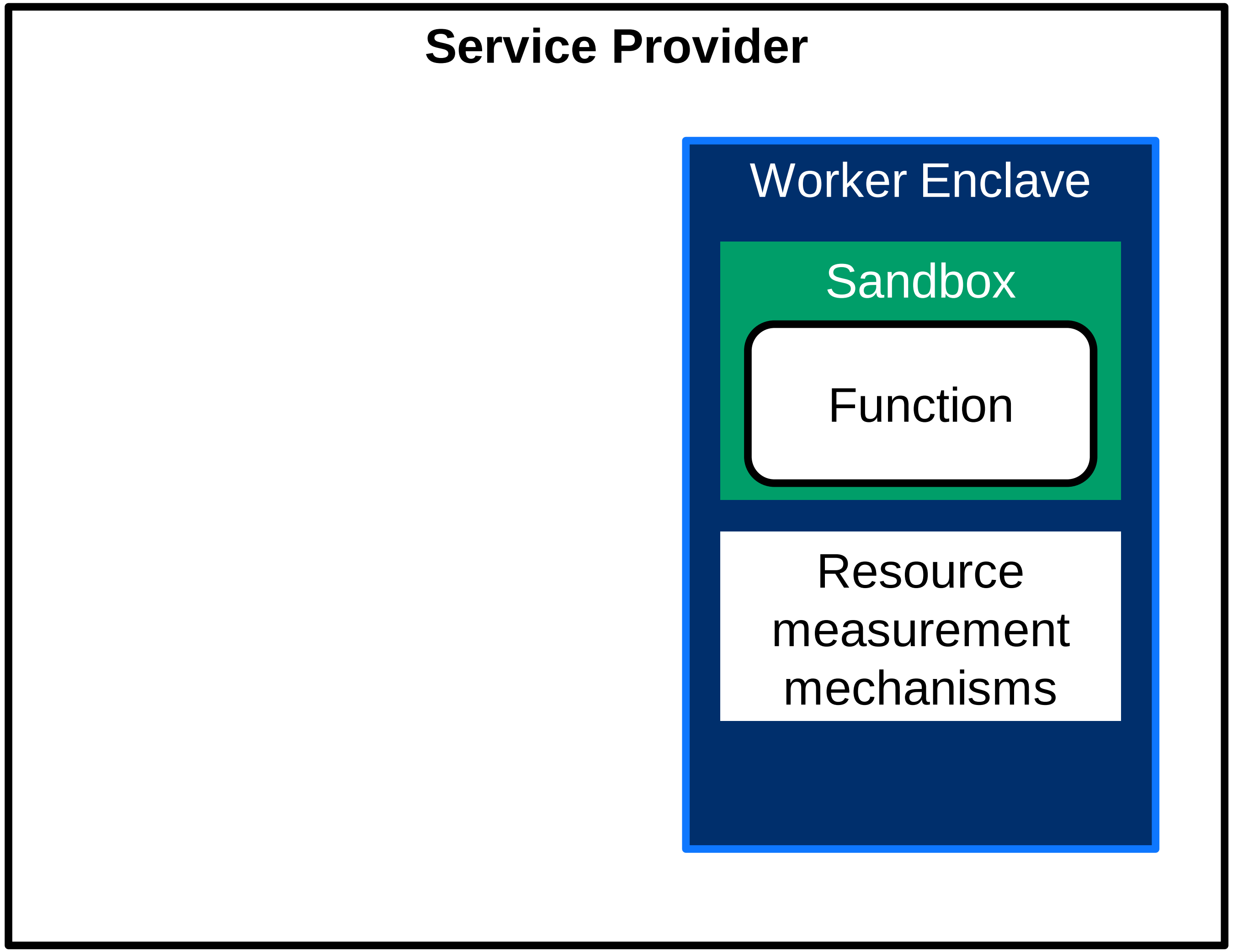
Architecture overview

Worker enclave runs function within a sandbox

- e.g. Ryoan
- sandboxing interpreters: e.g. for JavaScript

Challenges

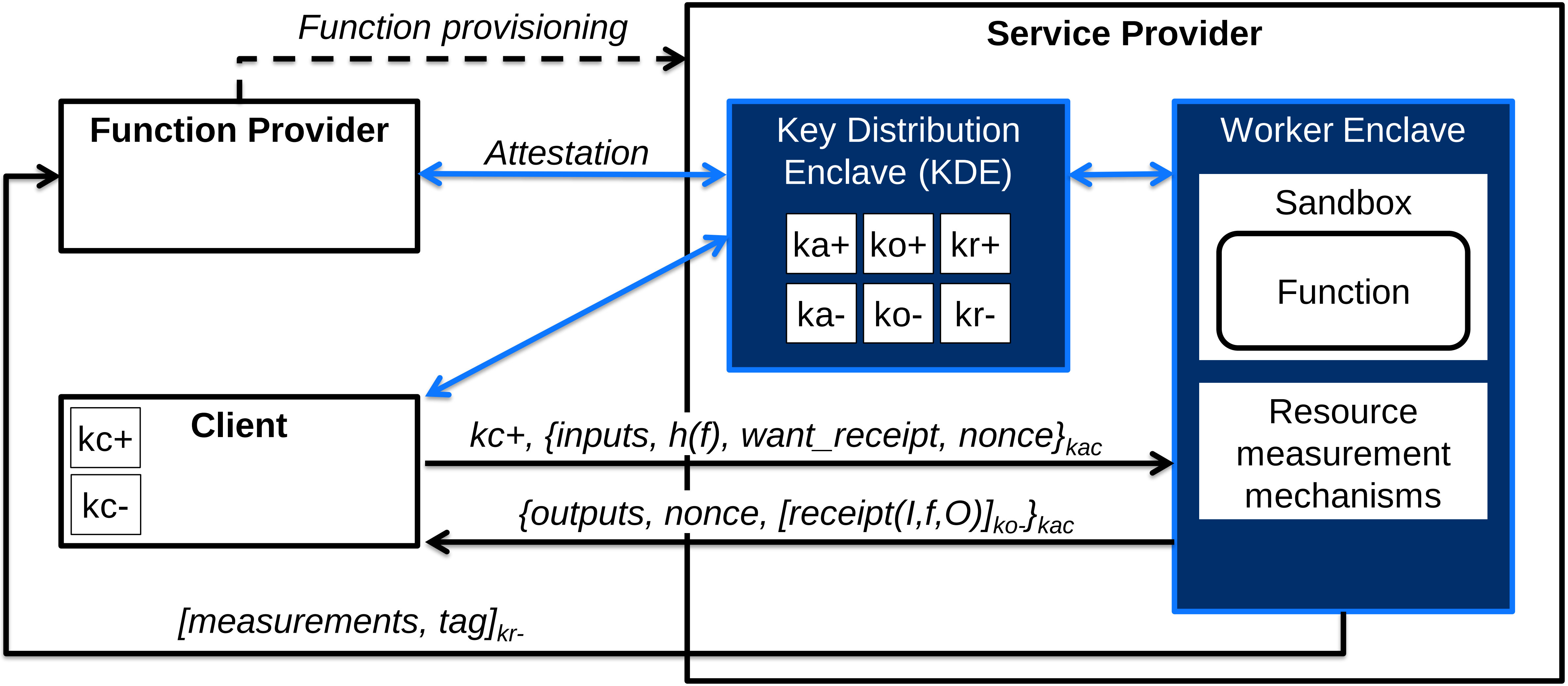
C1: Sandboxing
C2: Attesting enclaves
C3: Encrypting input
C4: Measuring time



Hunt et al., "Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data", OSDI 2016

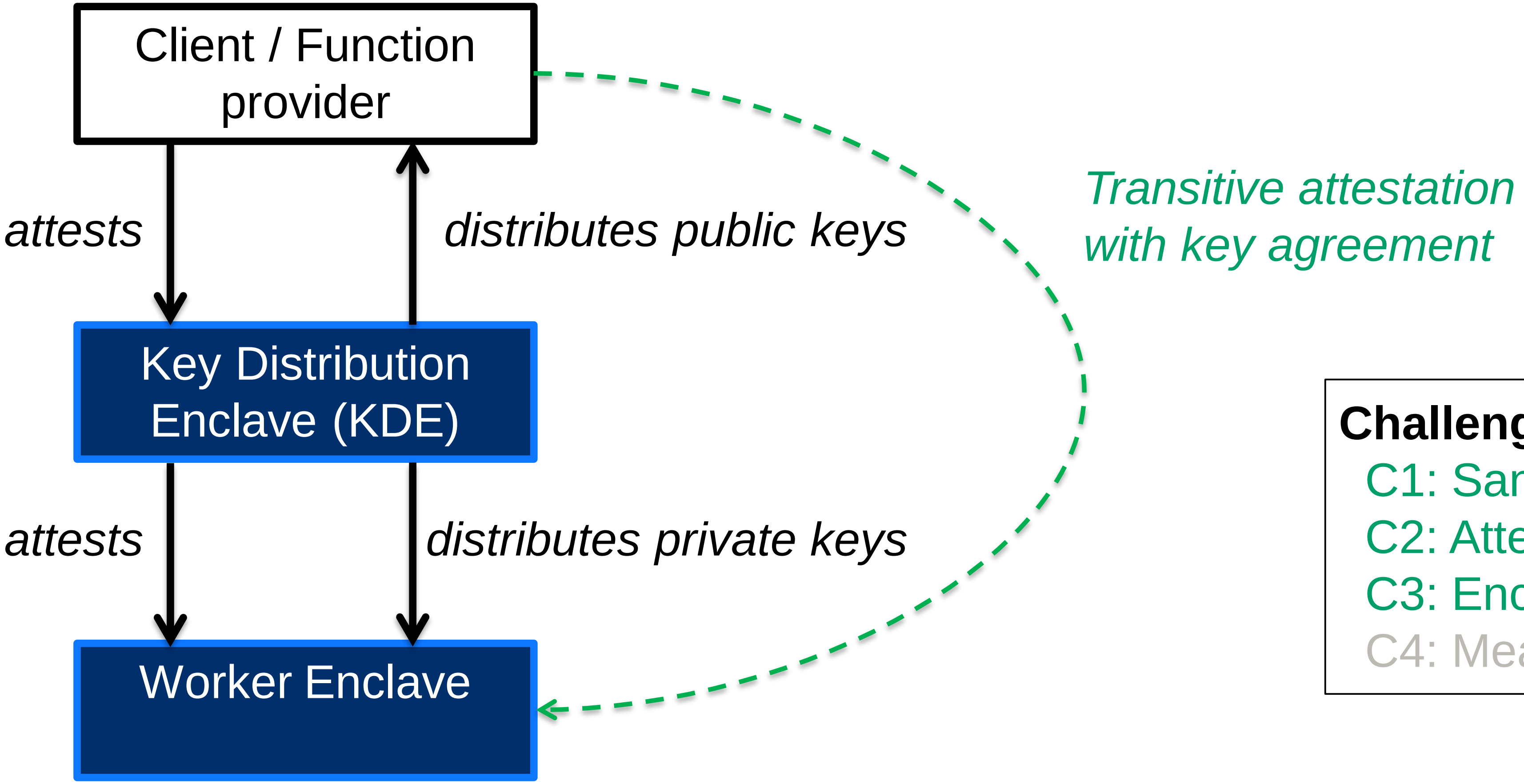
Architecture overview

ka: enclave's DH key
kc: client's DH key
ko: output key
kr: resource reporting key



Transitive attestation

Clients and function providers attest worker enclaves indirectly



Challenges
- C1: Sandboxing
- C2: Attesting enclaves
- C3: Encrypting input
- C4: Measuring time

Measuring Resource Usage in SGX

Motivation

FaaS is available from established cloud providers

Service	Invocations	Time (GHz-s)	Memory (GB-s)	Network (GB)
AWS Lambda	X	O	X	
Azure Functions	X	O	X	
Google Cloud Functions	X	X	X	X
IBM Cloud functions	X	O	X	

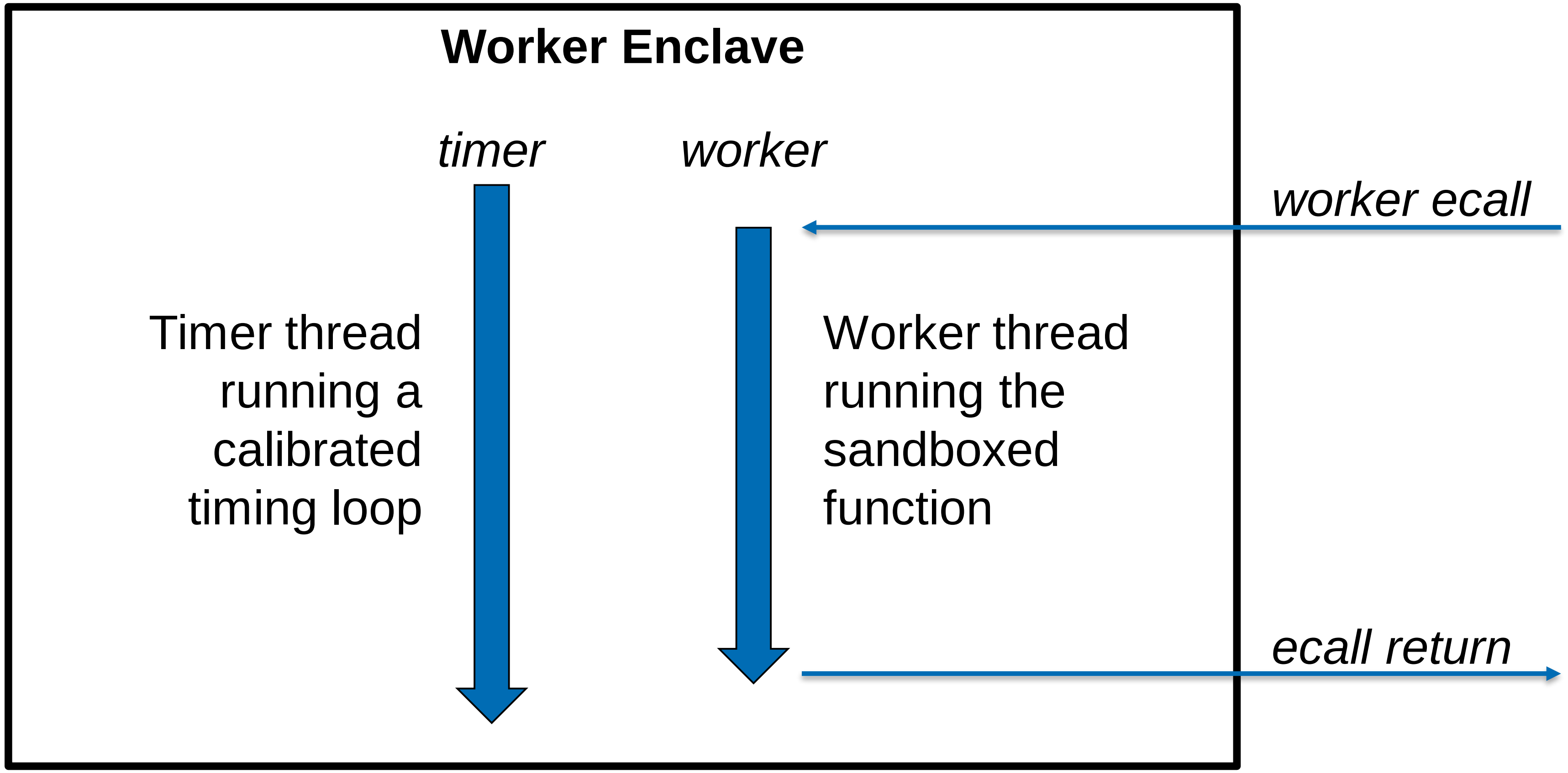
FaaS billing policies of established cloud providers *(X = explicit; O = implicit)*

Types of measurements

Symbol	Description	Units
t	Total compute time of the function	multiples of Ƭ
Ƭ	Duration of each tick in CPU cycles	GHz-s
$\mathbf{m_{int}}$	Time-integral of memory usage	GB-s
$\mathbf{m_{max}}$	Maximum memory used by the function	GB
net	Total number of network bytes sent and received	GB

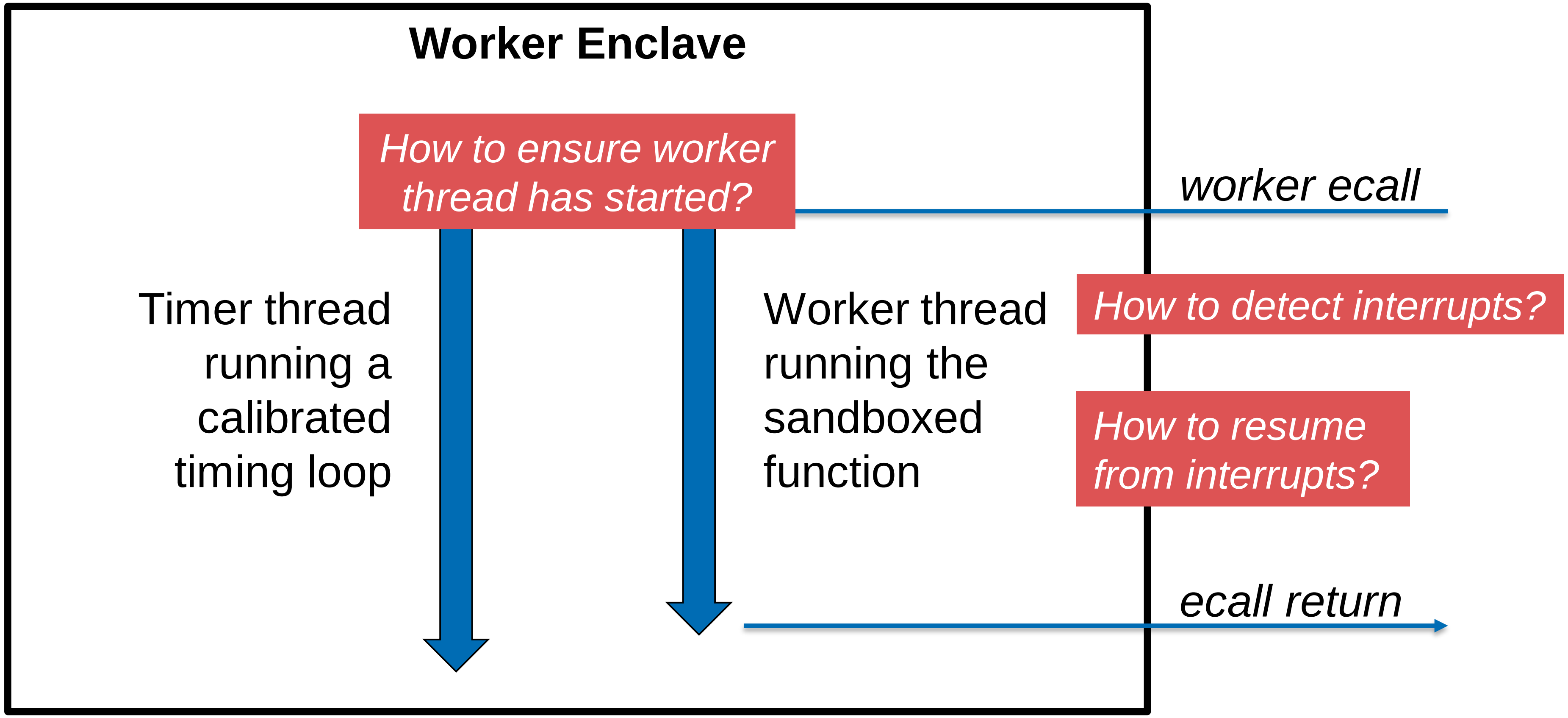
Measuring compute time

High level idea: two concurrent threads in the enclave (timer & worker)



Measuring compute time

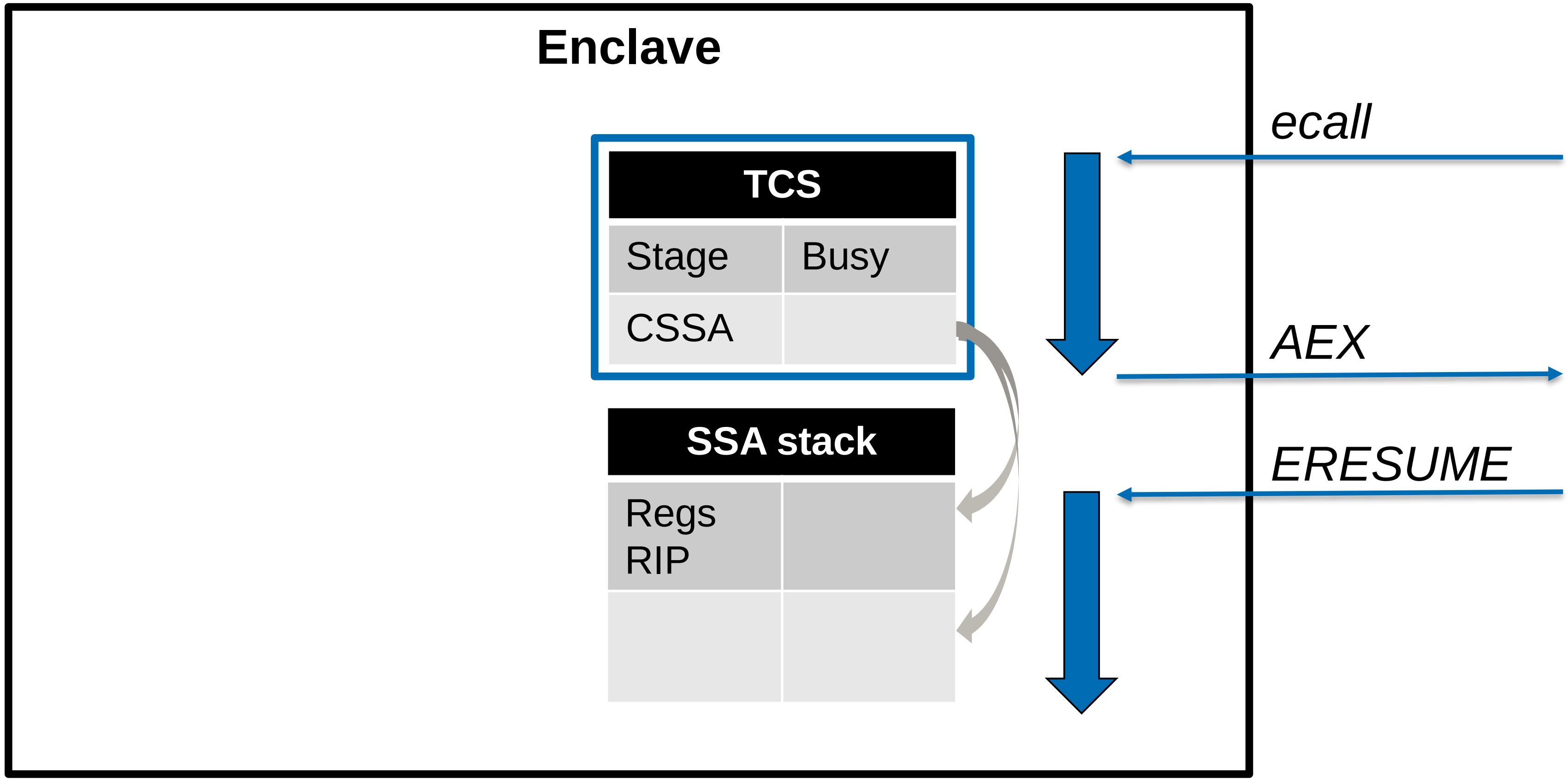
High level idea: two concurrent threads in the enclave (timer & worker)



Intel SGX internals

Enclave data structures
TCS: Thread Control Structure
(C)SSA: (Current) Save State Area

CPU Registers	
RAX	0xff…
RBX	
…	…
RSP	
RIP	0xff…



CPU Registers
RIP: Instruction Pointer
RSP: Stack Pointer

Intel Transactional Synchronization Extensions (TSX)

Special instructions enabling Hardware Lock Elision (HLE)

Read set

- Memory addresses read by the transaction (added upon access)
- Transaction will abort if address is concurrently written

Write set

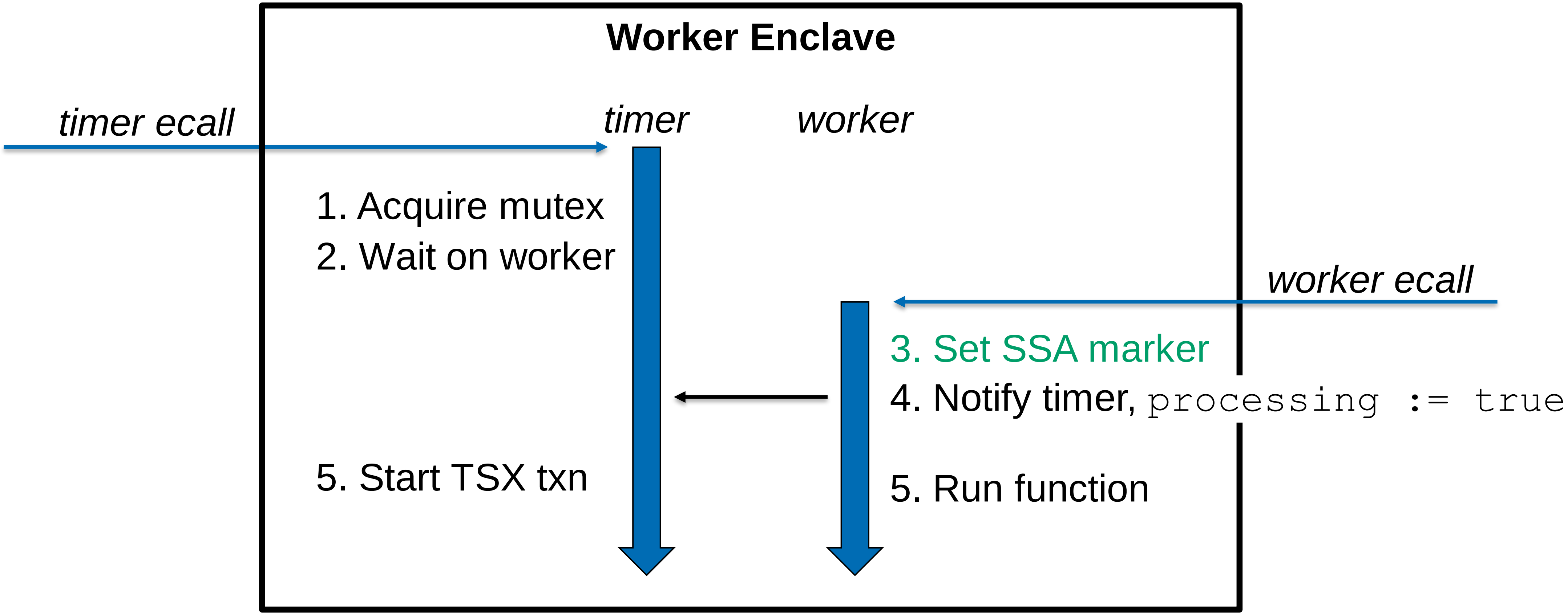
- Memory addresses written by the transaction
- Transaction will abort if address is concurrently read

Roll-back

- All operations since the beginning of the transaction are reverted

Starting a function

SSA stack	
Marker	0x12…

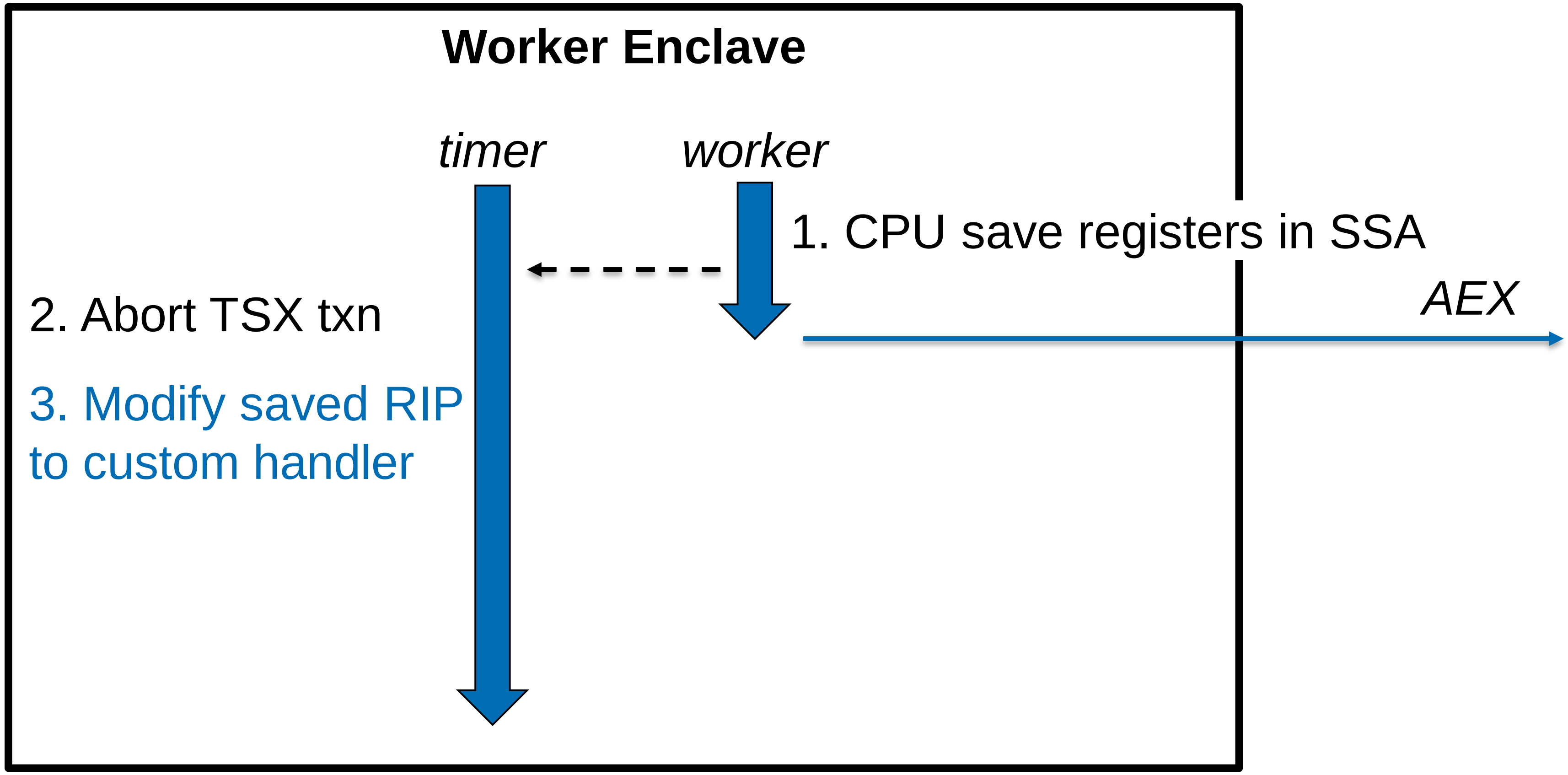


Timer thread algorithm

```
while(processing == true) {
    XBEGIN    // begin TSX txn
    if(worker.ssa == marker)  // add worker.ssa to txn read set
    {
        for(i=0; i<LOOP_COUNT; i++) // LOOP_COUNT depends on T
            nop;
        t_internal++;
    }
    XEND    // end TSX txn
    t_external = t_internal   // update external counter
}
```

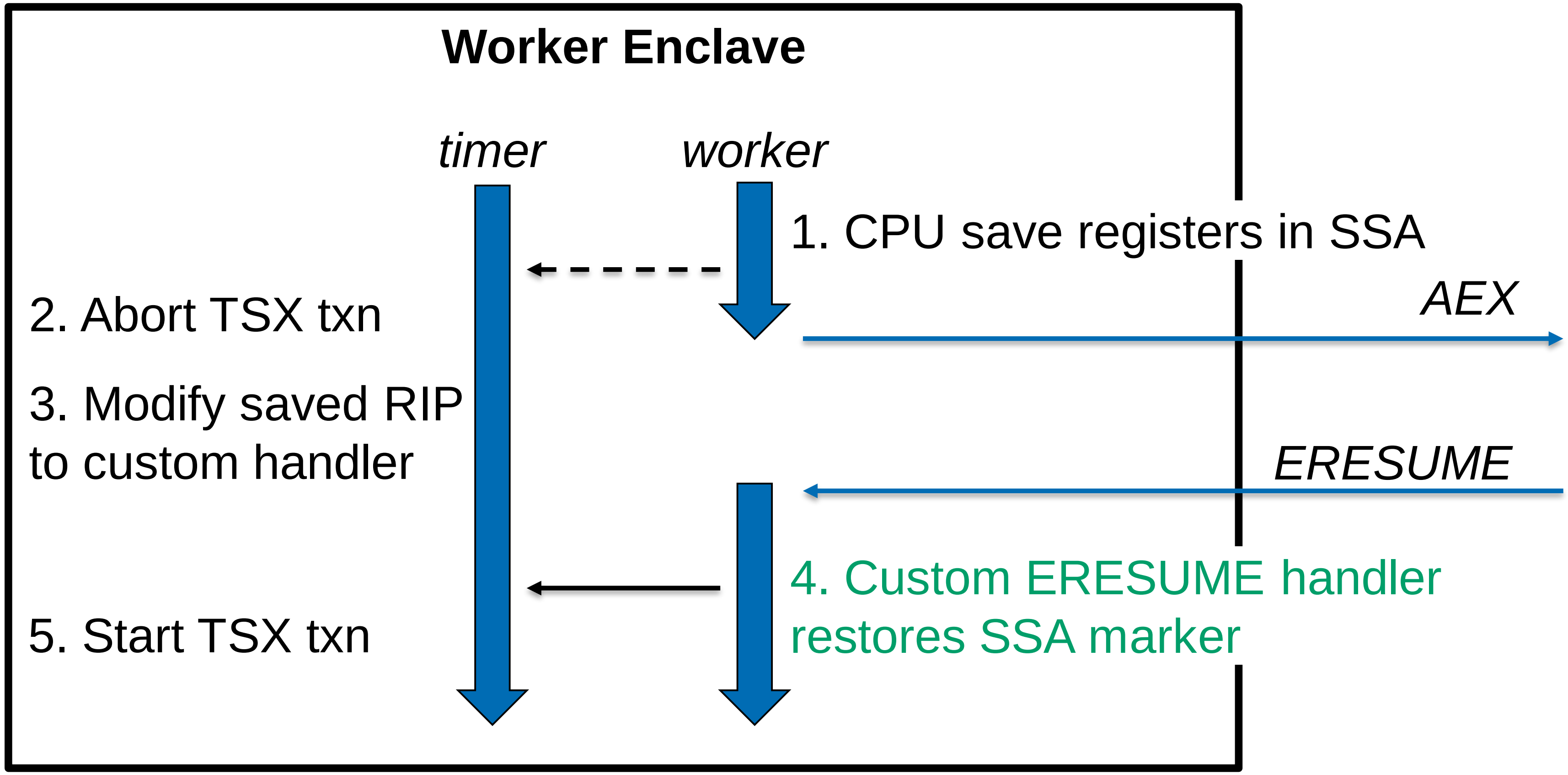
Worker thread interrupted

SSA stack	
Regs RIP	0x00… 0x89…



Worker thread resumed

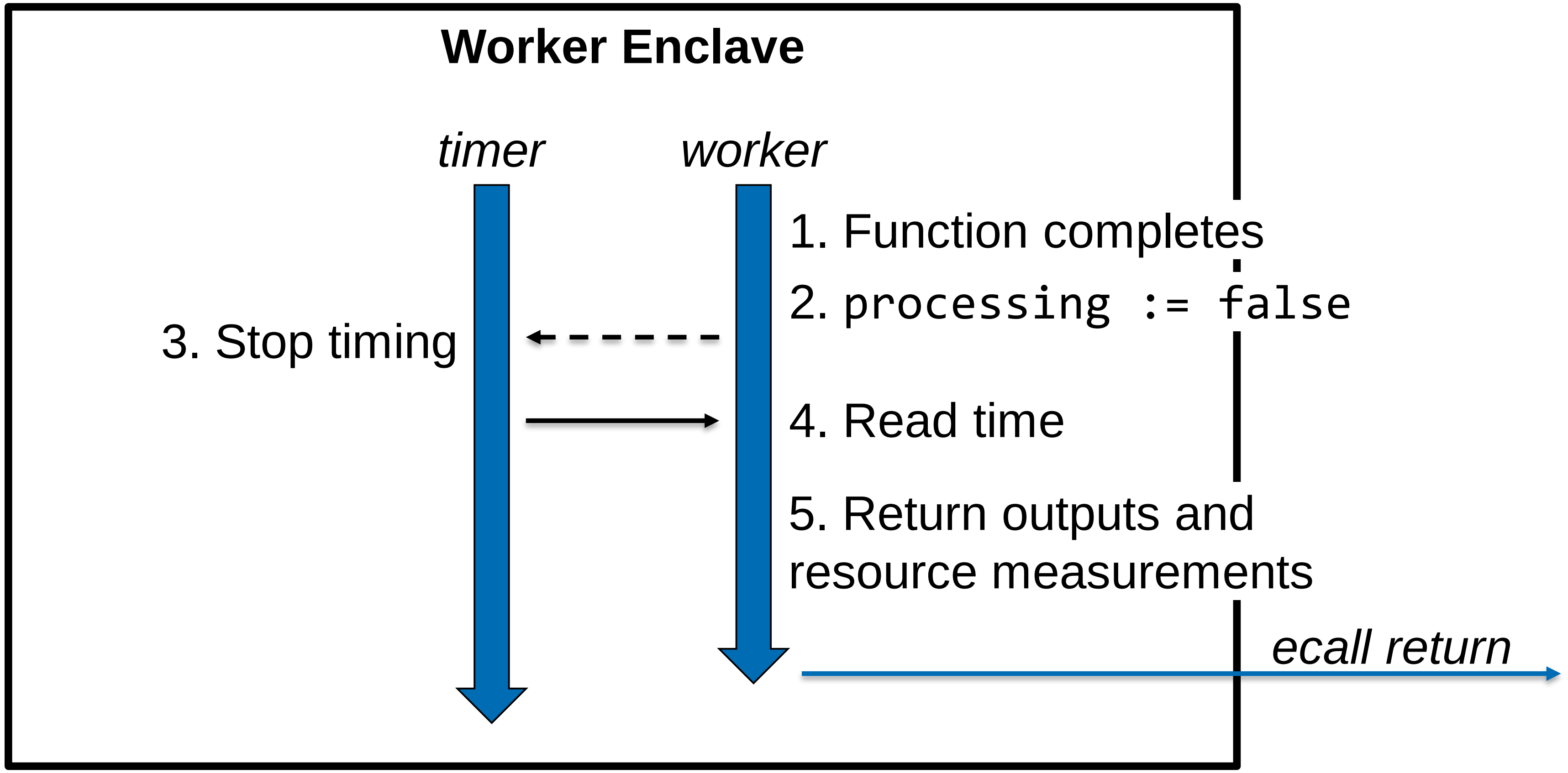
SSA stack	
Marker	0x12..



Custom ERESUME handler

```
.text
.globl custom_eresume_handler
.type custom_eresume_handler,@function
custom_eresume_handler:
   push %rax                              # Save registers
   push %rbx
   lea g_worker_ssa_gpr(%rip),%rax        # Load pointer
   mov (%rax),%rbx                        # Dereference pointer
   movl $12345,(%rbx)                     # Write SSA marker value
   pop %rbx                               # Restore registers
   pop %rax
   jmp *g_original_ssa_rip(%rip)          # Resume execution
```

Completing a function



Measuring Memory and Networking

Memory

- Instrumented allocators used by interpreter
- Measurements updated on every allocation/free

$\mathbf{m_{int}}$	Time-integral of memory usage
$\mathbf{m_{max}}$	Maximum memory used by the function

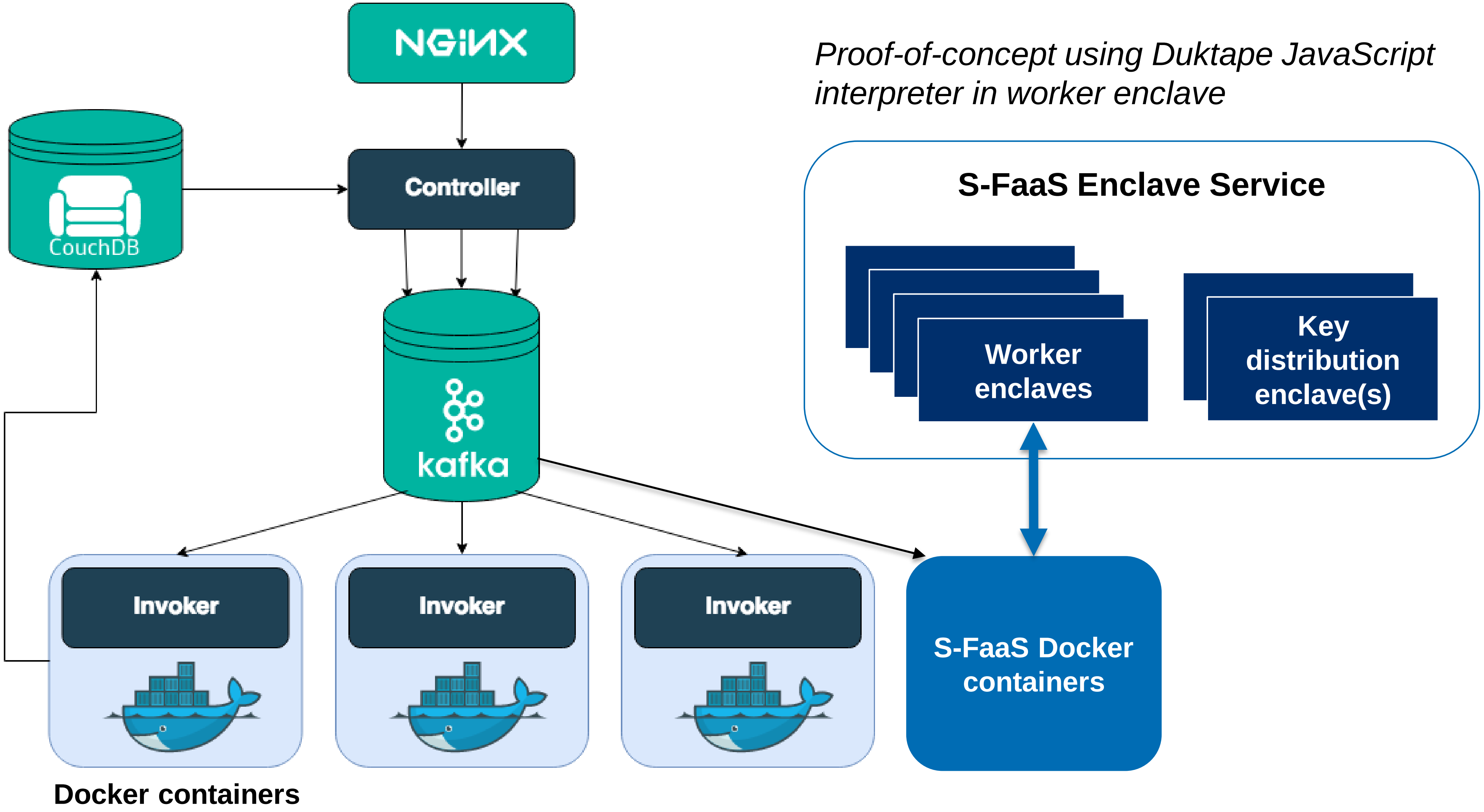
Network

- Payloads measured inside enclave

Integration with OpenWhisk

Integration with OpenWhisk

Proof-of-concept using Duktape JavaScript interpreter in worker enclave



https://openwhisk.apache.org/documentation.html

Evaluation

Evaluation: Accuracy

Synthetic function with well-defined compute and memory requirements

- `fibonacci(k)` calculates the first k numbers in the Fibonacci sequence

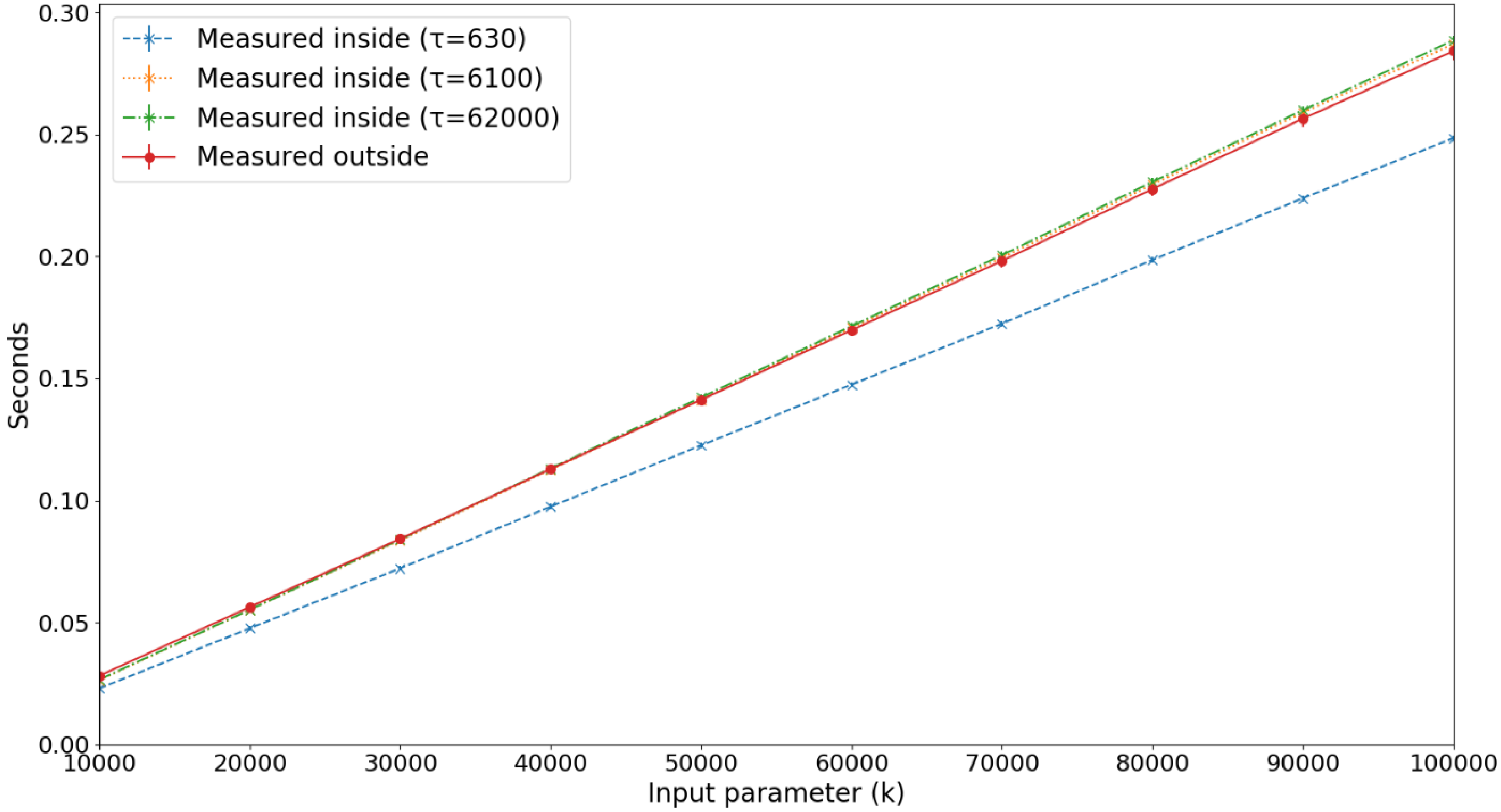
Compute time

- Expected to be linear in k
- Can be compared with measurement outside the enclave

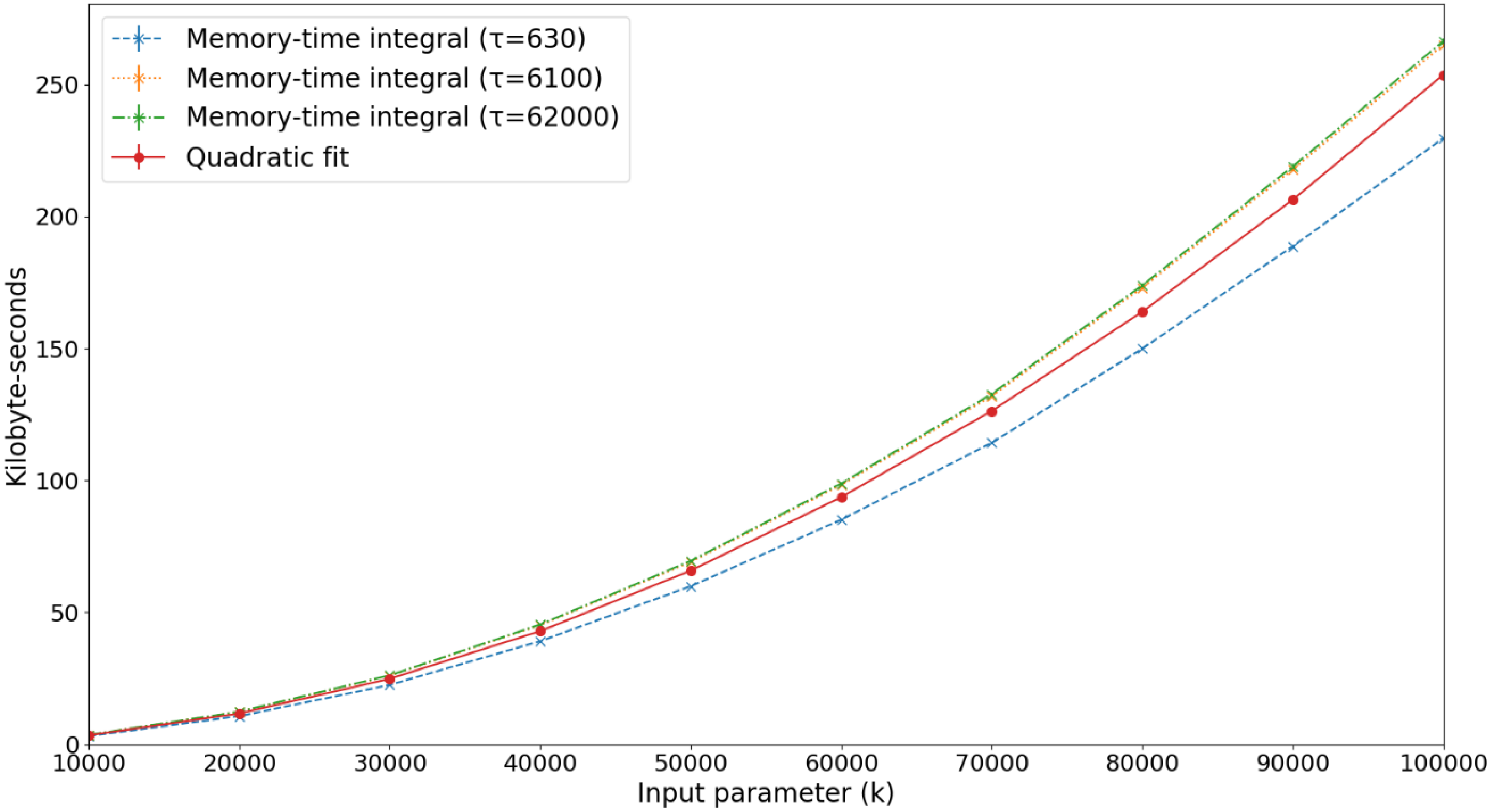
Memory time-integral

- Expected to be quadratic in k (k-element list pre-allocated at start of function)
- Harder to measure outside enclave

Evaluation: Accuracy



Evaluation: Accuracy



Evaluation: Performance

Pre-function latency

- Measure cold-start and warm-start latency
- Tested using an empty function to isolate pre-function latency
- Baseline: equivalent operation (same interpreter) without SGX

Resource measurement overhead

- Measure overhead of S-FaaS resource measurement mechanisms
- Octane JavaScript benchmarks (excluding graphical tests)
- Baseline: equivalent operation without resource measurement

Benchmark environment

- Core i5-6500, 8GB RAM, Ubuntu 16.04, Intel SGX SDK 2.2.1

Evaluation: Pre-function latency

Cold-start

1. Create Docker container
2. Create enclave
3. Provision function
4. Perform key-agreement
5. Return empty response

Baseline: 3179 ms ($\sigma = 40$ ms)

S-FaaS: 3249 ms ($\sigma = 38$ ms)

Latency increase: ~2%

Warm-start

1. Create Docker container
2. Create enclave
3. Provision function
4. Perform key-agreement
5. Return empty response

Baseline: 204 ms ($\sigma = 106$ ms)

S-FaaS: 210 ms ($\sigma = 149$ ms)

Latency increase: ~3%

Evaluation: Resource measurement overhead

Function	Baseline	S-FaaS					
		No encryption		Encryption		Encryption & receipt	
Box2D	3.019	3.118	3.3%	3.121	3.4%	3.135	3.8%
DeltaBlue	1.446	1.524	5.4%	1.529	5.7%	1.537	6.3%
NavierStokes	4.155	4.418	6.3%	4.447	7.0%	4.473	7.7%
RayTrace	0.779	0.848	8.9%	0.850	9.1%	0.852	9.4%
Richards	1.719	1.767	2.8%	1.767	2.8%	1.799	4.7%
Overall	**-**		***5.3%***		***5.6%***		***6.3%***

Trade-offs and limitations

Need for an additional thread

- State-of-the-art SGX side-channel defences(*) require control of both sibling hyperthreads

Timing granularity

- Choice of T affects extent of under- or over-reporting
- S-FaaS service providers can specify T for each function

Architecture-specific calibration

- Timing loop must be calibrated for different CPU architectures

(*) SGX side-channel defenses:
Cloak: Gruss et al., "Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory", Usenix SEC 2017
HyperRace: Chen et al., "Racing in Hyperspace: Closing Hyper-Threading Side Channels on SGX with Contrived Data Races", IEEE S&P 2018
Varys: Oleksenko et al., "Varys: Protecting SGX enclaves from practical side-channel attacks", Usenix ATC 2018

Suggested SGX enhancements

Secure tick counter

- Provide a trustworthy tick counter that can be accessed without leaving the enclave

Custom ERESUME handlers

- Allow enclaves to specify an in-enclave handler to be called on each ERESUME
- Could also be used to detect frequent AEX events indicative of side-channel attacks

Integration with distributed systems

Smart contracts to pay for outsourced computation

- S-FaaS function receipts and resource measurements can be verified in smart contracts
- Straight-forward integration with payment networks
  - Particularly beneficial to non-traditional service providers

Leader election based on useful work

- Similar to Resource-Efficient Mining for Blockchains (Zhang et al.)
- Uses “useful computation” to determine who mines next block

Zhang et al., “REM: Resource-Efficient Mining for Blockchains”, Usenix SEC 2017

Deployment considerations

Incremental deployment

- Initially, S-FaaS requires no changes on client-side (no client attestation or encryption)
- Clients can individually start to verify attestation and/or encrypt inputs

Implementations with other TEEs

- S-FaaS could be ported to e.g. ARM TrustZone
- TrustZone secure world still requires functions to run in a suitable sandbox, but timing would be simpler because secure world cannot be arbitrarily paused

Conclusions

FaaS increasingly popular with cloud providers and non-traditional service providers

- Requires strong security: data confidentiality and integrity of computation
- Requires accurate and trustworthy resource consumption measurement

S-FaaS demonstrates how to secure current FaaS architectures using SGX

- Transitive attestation
- In-enclave resource measurement mechanisms

Possibilities for future work

- Integration with distributed systems
- Measuring resource usage in other SGX applications

What if SGX is broken?

Back to current state of FaaS security and resource measurement

- **TEEs useful in two kinds of settings:**
  1. improving security
  2. improving other attributes while preserving security

  S-FaaS is Type 1. TEE compromise is a bigger concern in Type 2

- **Application-specific ways of detecting / mitigating effects of TEE compromise, e.g.,**
  - post-mortem auditing of signed receipts
  - statistical mechanisms like in PoET and Zhang et. al.

Trade-offs and limitations

Need for an additional thread

- Sibling hyperthreads disabled by some cloud providers due to shared L1 cache
- State-of-the-art SGX defenses (e.g. Cloak, HyperRace, and Varys) require control of both sibling hyperthreads to prevent cache-line side-channel attacks

Timing granularity

- Smaller values of T reduce time “sacrificed” by interrupts, but increase number of transactions
- Transaction setup times are not counted, so frequent transactions lead to under-reporting
- In S-FaaS, service providers can choose values of T for each function

Architecture-specific calibration

- Timing loop must be calibrated for different CPU architectures