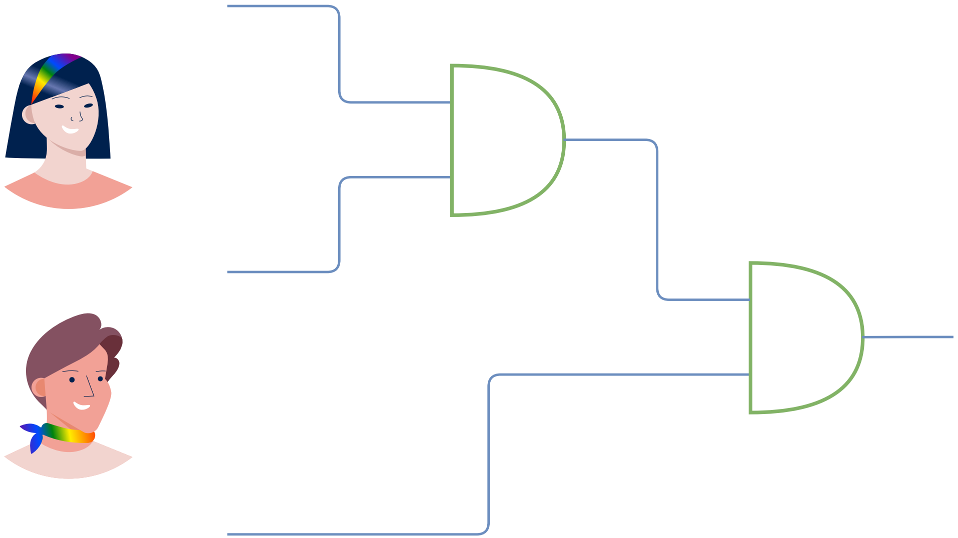
SEEC: Memory Safety Meets Efficiency in Secure Two-Party Computation

Robin Hundt Nora Khayata Thomas Schneider

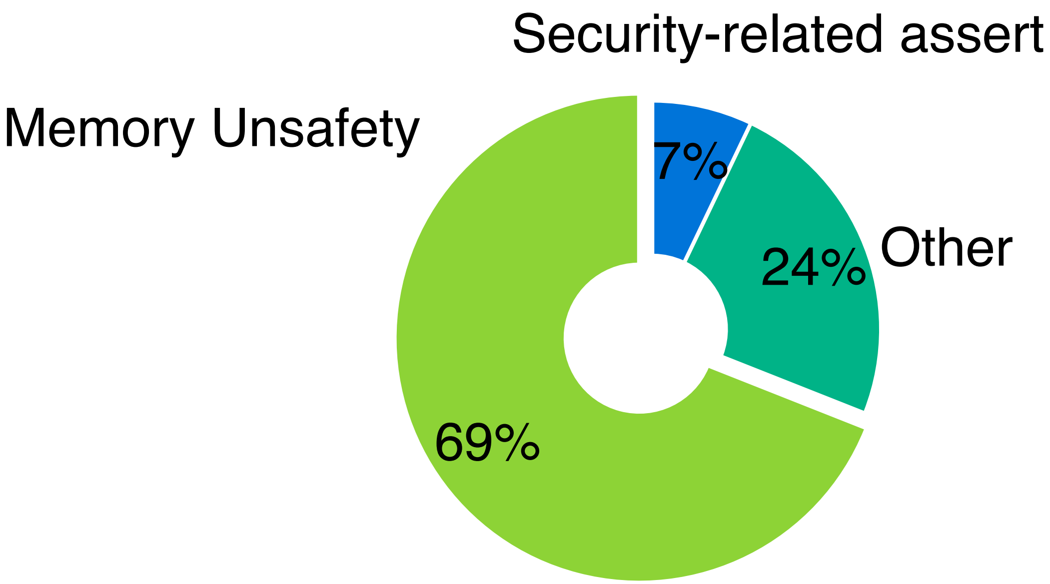
ENCRYPTO CRYPTOGRAPHY AND PRIVACY ENGINEERING



TECHNISCHE UNIVERSITÄT DARMSTADT

Motivation

Safety

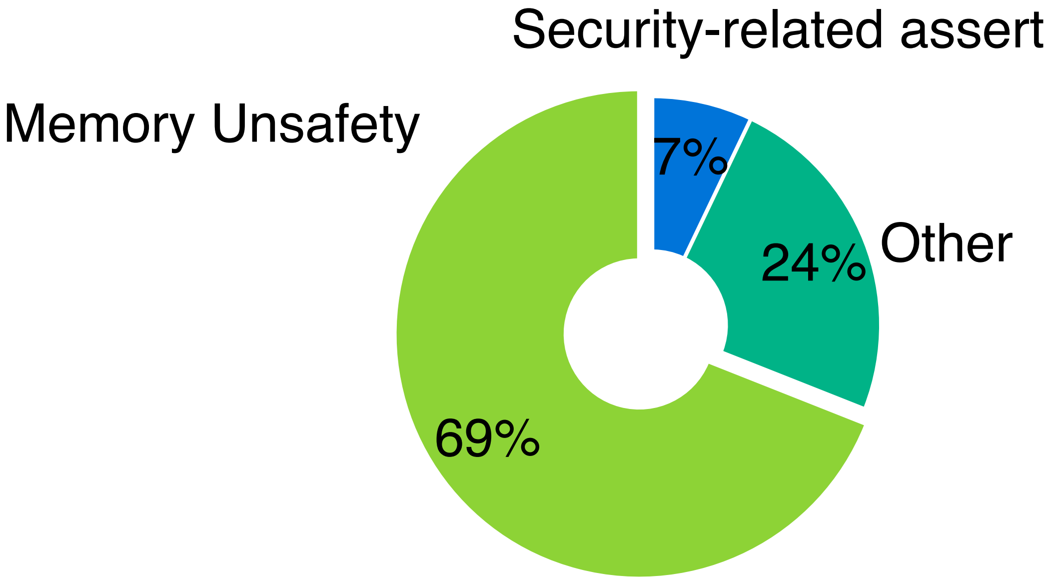


Source: The Chromium Projects - Memory Safety

Motivation

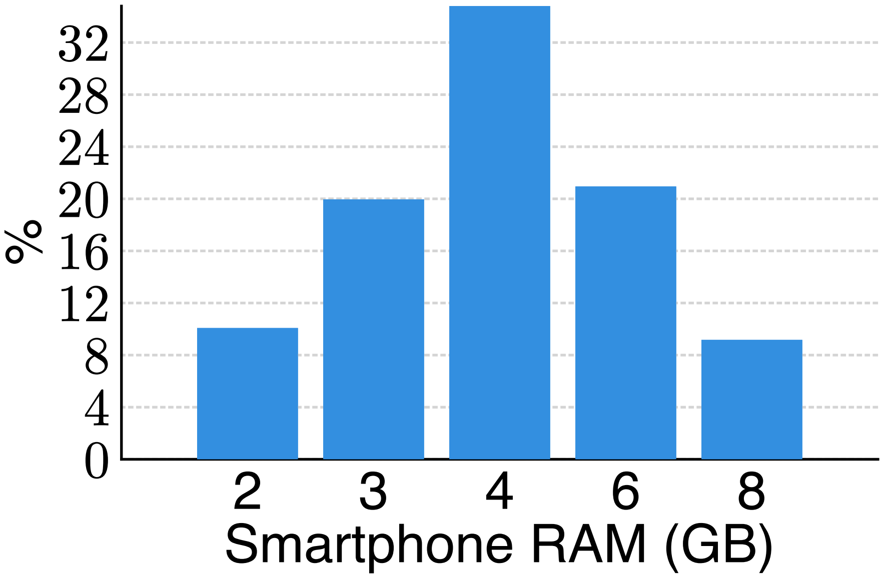
ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

Safety



Source: The Chromium Projects - Memory Safety

Efficiency



Source: scientiamobile, 2022

[SEEC Executes Enormous Circuits (SEEC)]

- High-Level eDSL / FUSE [BHK+23]
- (SIMD) Sub-Circuits
- Function (In-)Dependent Setup
- Extensibility w/o forking
- Cross-Platform

[SEEC Executes Enormous Circuits (SEEC)]

2PC GMW (A/B) [GMW87,Bea92]
2PC GMW (A+B) [DSZ15]
ASTRA (B*) [CCPS19]
ABY2.0 (B*) [PSSY21]
OT: [ALSZ13], Silent OT [BCG+19]

encryptogroup/mpc-bench

* Partial Implementation

Functions in Traditional Programs

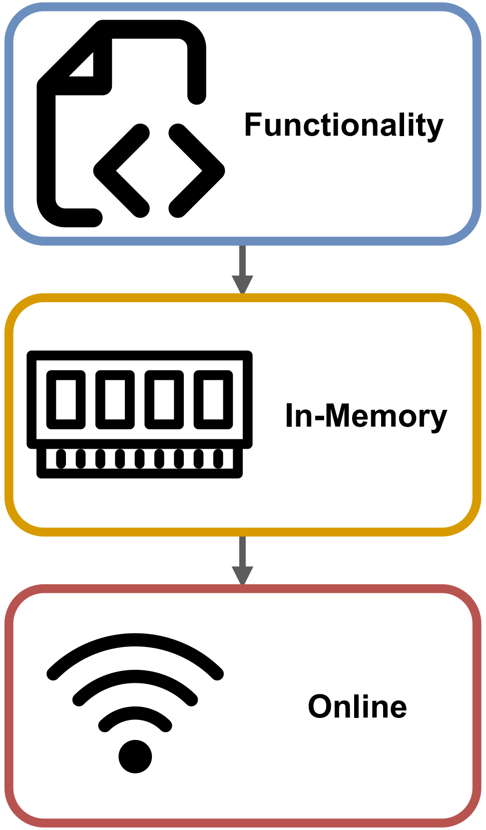
```
fn func(args: [bool; 2]) -> bool {
  // ... calculate return
}

let [a, b, c] = read_data();

let result_0 = func([a, b]);
// use result_0 for next func call
let result_1 = func([result_0, c]);
```

Circuit Reuse in Secure Programs

```
fn func(args: [bool; 2]) -> bool {
  // ... calculate return
}

let [a, b, c] = read_data();

let result_0 = func([a, b]);
// use result_0 for next func call
let result_1 = func([result_0, c]);
```

```
fn func(args: [SBool; 2]) -> Sbool {
  // ... calculate return
}

let [a, b, c] = read_data();

let result_0 = func([a, b]);
// use result_0 for next func call
let result_1 = func([result_0, c]);
```

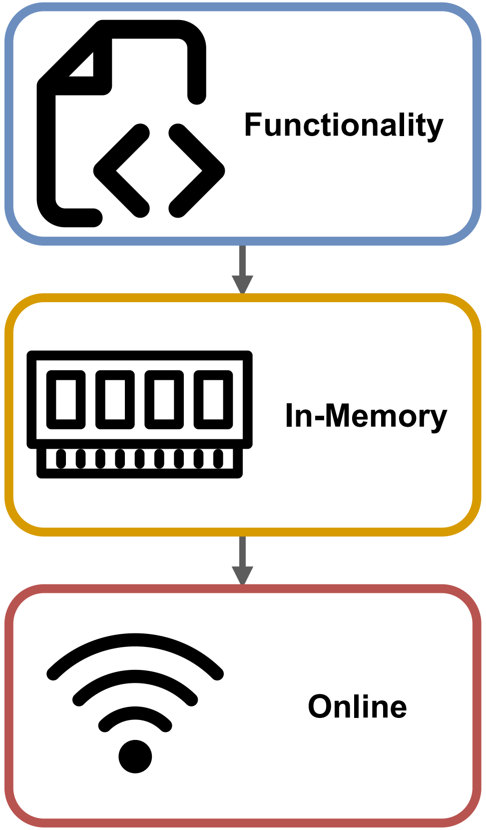
Sub-Circuits in GMW: Challenges




```
func(a);
func(b);
```

`a` and `b` are independent

Sub-Circuits in GMW: Challenges



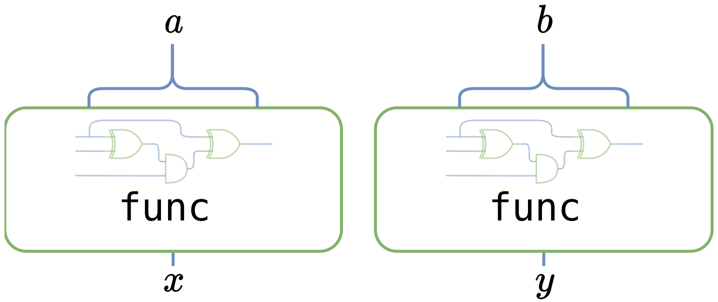

```
func(a);
func(b);
```

a and b are independent

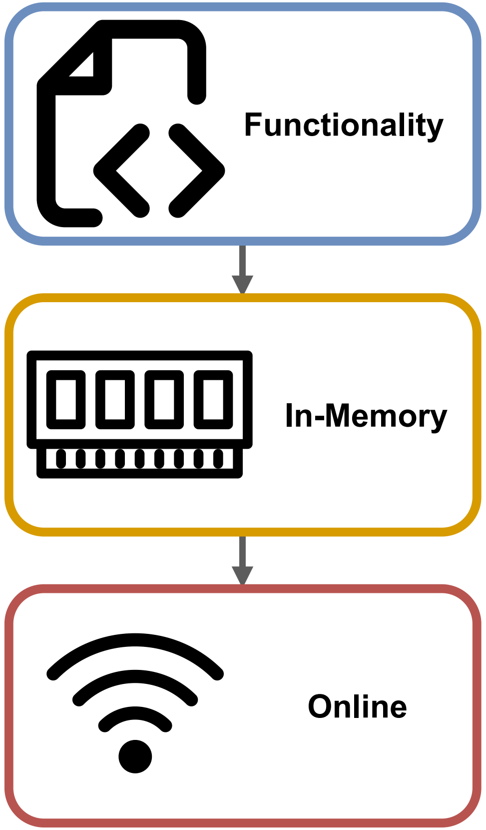
Bytecode VM

```
func:
  # ...
  ret

call func
call func
```

Graph based



Sub-Circuits in GMW: Challenges



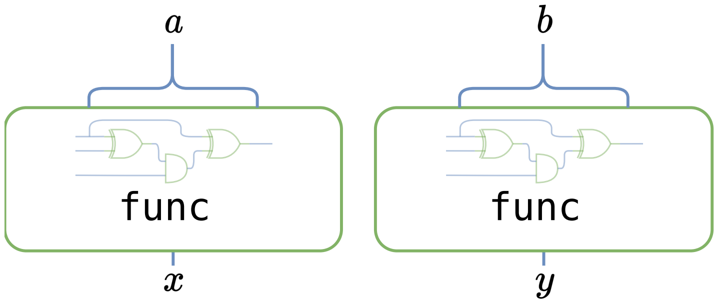

```
func(a);
func(b);
```

`a` and `b` are independent

Bytecode VM

```
func:
  # ...
  ret

call func
call func
```

Graph based



- → Increased rounds
- → `func` only once in memory

- → Concurrent evaluation
- → Increased Memory

SEEC: eDSL Enables Efficient Circuit Reuse

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

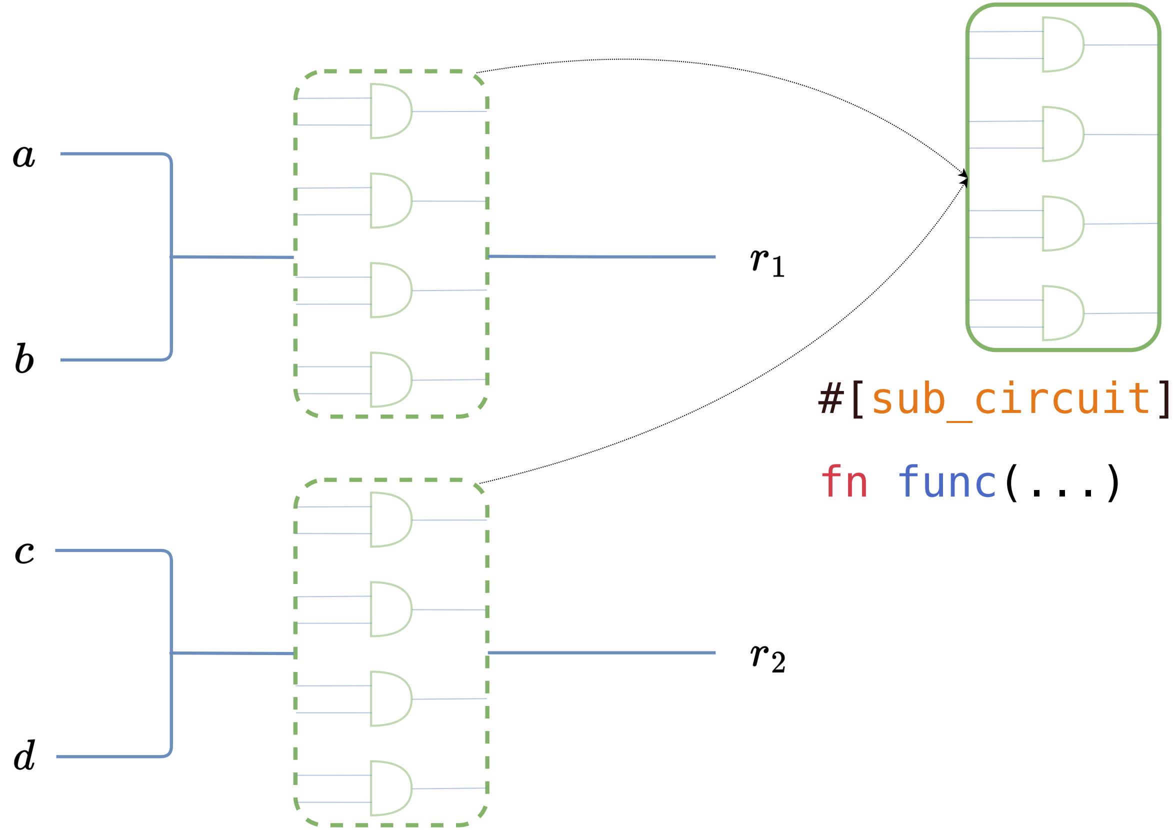
```
#[sub_circuit]
fn func(a: Vec<Secret>, b: Vec<Secret>)
    -> Vec<Secret> {
  a.into_iter().zip(b).map(|(el_a, el_b)| {

    el_a & el_b

  }).collect()
}
```

SEEC: eDSL Enables Efficient Circuit Reuse

```rust
#[sub_circuit]
fn func(a: Vec<Secret>, b: Vec<Secret>)
    -> Vec<Secret> {
  a.into_iter().zip(b).map(|(el_a, el_b)| {

    el_a & el_b

  }).collect()
}
```

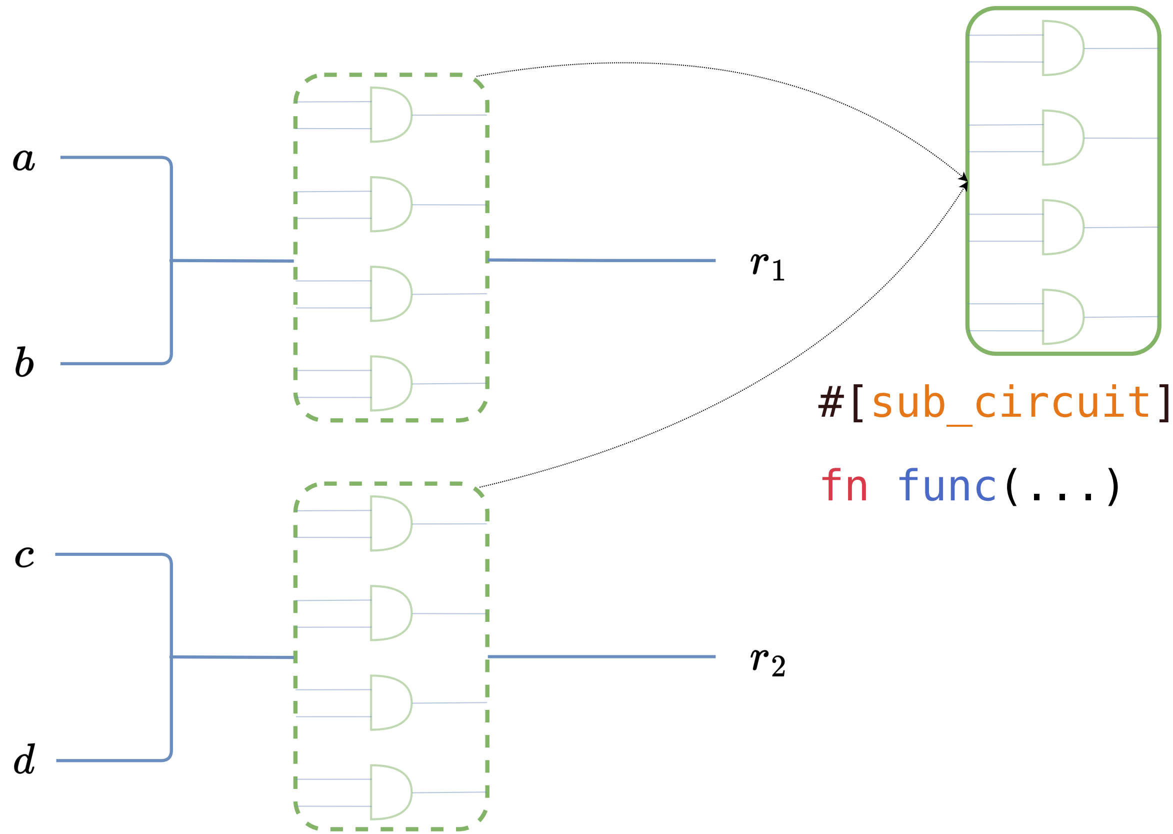
```rust
let (a, b, c, d) = init_data();
// func is called as normal
function.
let r1 = func(a, b);

let r2 = func(c, d);
```

SEEC: Sub-Circuits Are Not Inlined




```
#[sub_circuit]
fn func(...)
```

SEEC: Sub-Circuits Are Not Inlined

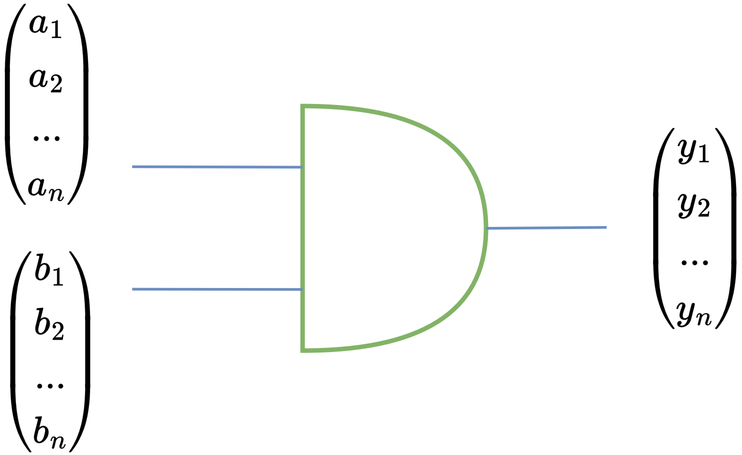



```
#[sub_circuit]
fn func(...)
```

Online

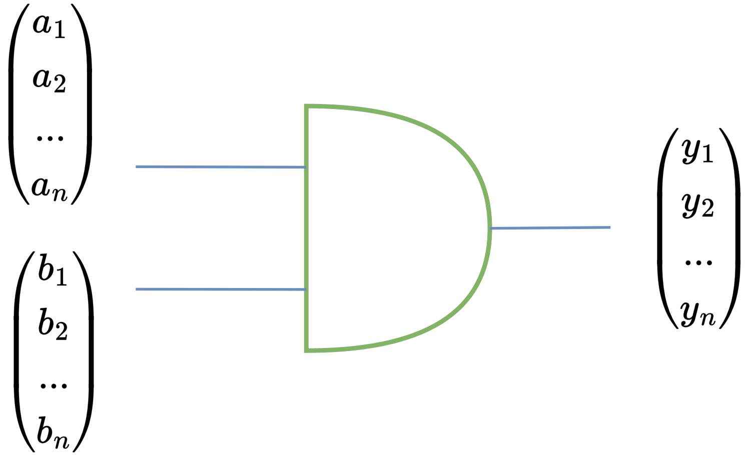
- Layer iteration **as if** inlined (DL)
  - No increase in depth
- Partial and concurrent evaluation

Single Instruction, Multiple Data (SIMD)

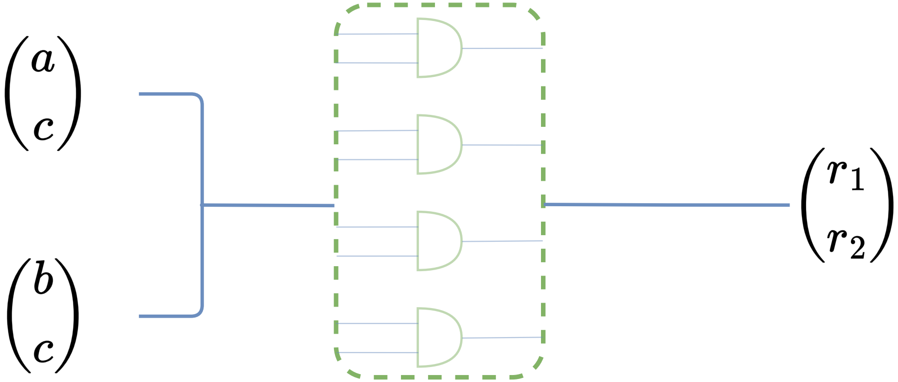


Traditional SIMD, e.g., in MOTION [BDST22].

Single Instruction, Multiple Data (SIMD)



Traditional SIMD, e.g., in MOTION [BDST22].



SIMD Sub-Circuits in SEEC.

SEEC: Optimizations

Static Layers (SL)

- Transforms Dynamic Layer (DL) representation
- Layers are precomputed for every call site
- Precomputed layers are stored deduplicated

Early Deallocation (ED)

Streaming MTs (SMT)

SEEC: Optimizations

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

Static Layers (SL)

- Transforms Dynamic Layer (DL) representation
- Layers are precomputed for every call site
- Precomputed layers are stored deduplicated

Early Deallocation (ED)

- Unneeded gate outputs are freed
- Only applies to SIMD circuits

Streaming MTs (SMT)

SEEC: Optimizations

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

Static Layers (SL)

- Transforms Dynamic Layer (DL) representation
- Layers are precomputed for every call site
- Precomputed layers are stored deduplicated

Early Deallocation (ED)

- Unneeded gate outputs are freed
- Only applies to SIMD circuits

Streaming MTs (SMT)

- Multiplication Triples (MTs) are precomputed and stored in a file
- Online: read on-demand in batches from the file

Evaluation

Frameworks

- ABY [DSZ15]
- MP-SPDZ [Kel20]
- MOTION [BDST22]
- SEEC

Environment

LAN-0.25ms / LAN-1.25ms
WAN-100ms

$1, 2, \ldots, 32$ Threads

Heaptrack[1]

[1] https://github.com/KDE/heaptrack

Evaluation

Frameworks

- ABY [DSZ15]
- MP-SPDZ [Kel20]
- MOTION [BDST22]
- SEEC

Environment

LAN-0.25ms / LAN-1.25ms
WAN-100ms

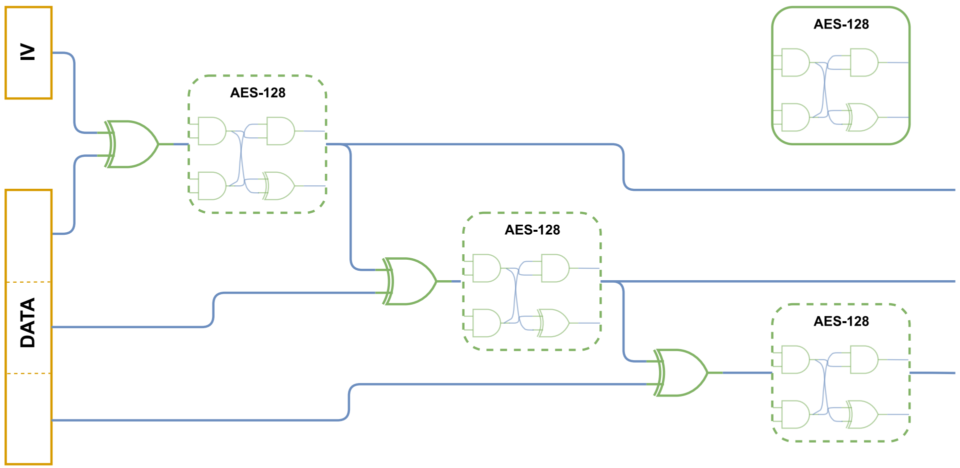
$1, 2, \ldots, 32$ Threads

Heaptrack[1]

[1] https://github.com/KDE/heaptrack

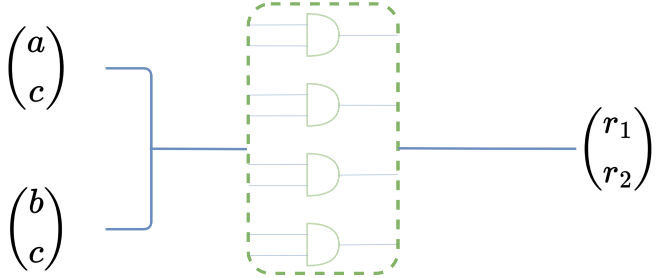
Circuits

Sub-Circuit

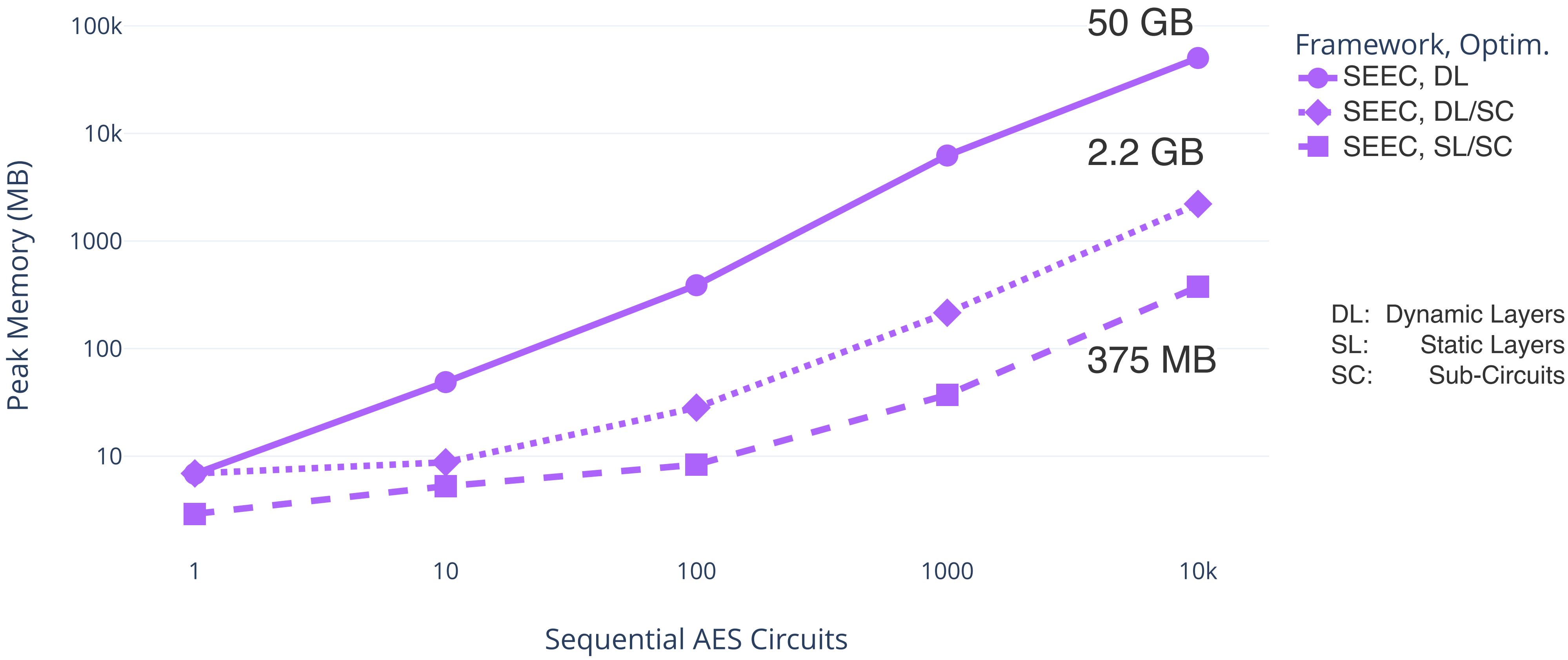


SIMD

- AES-128
- SHA-256

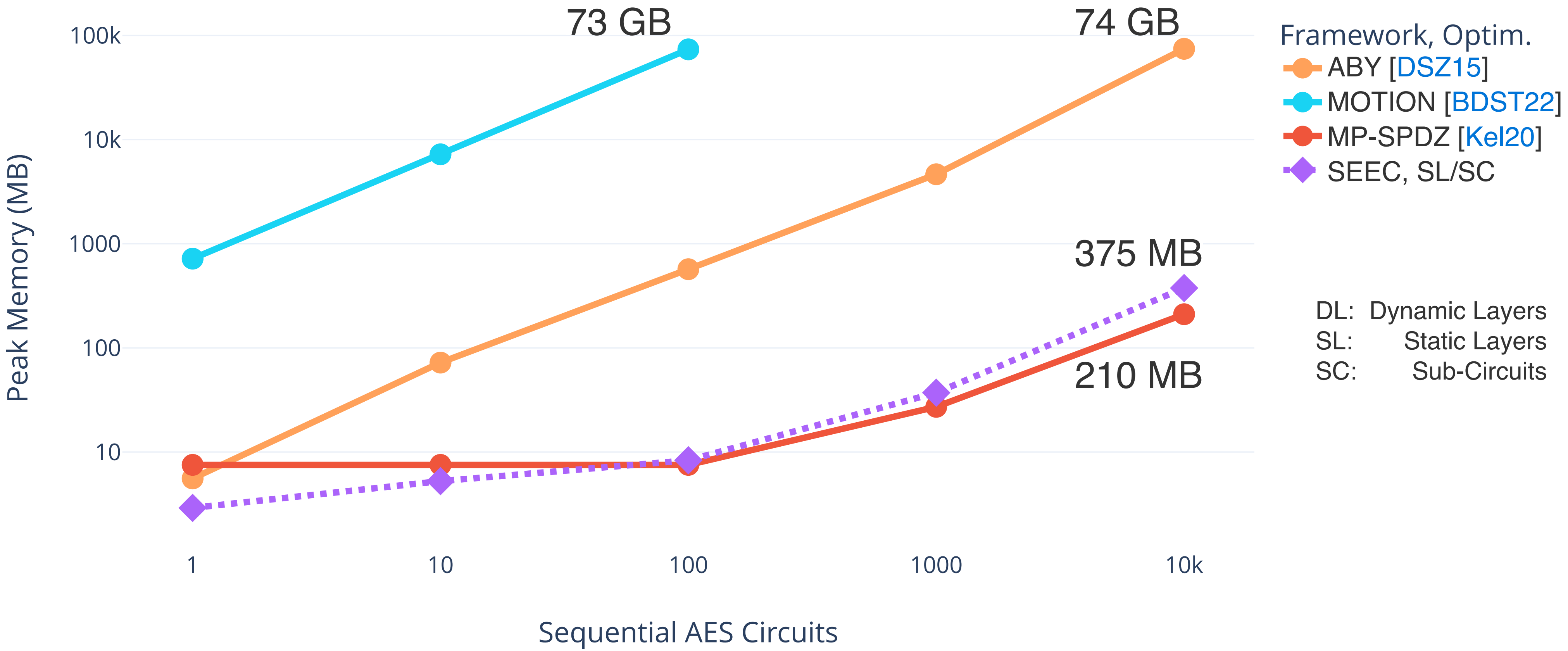


AES-CBC: Reduced Memory via Sub-Circuits



DL: Dynamic Layers
SL: Static Layers
SC: Sub-Circuits

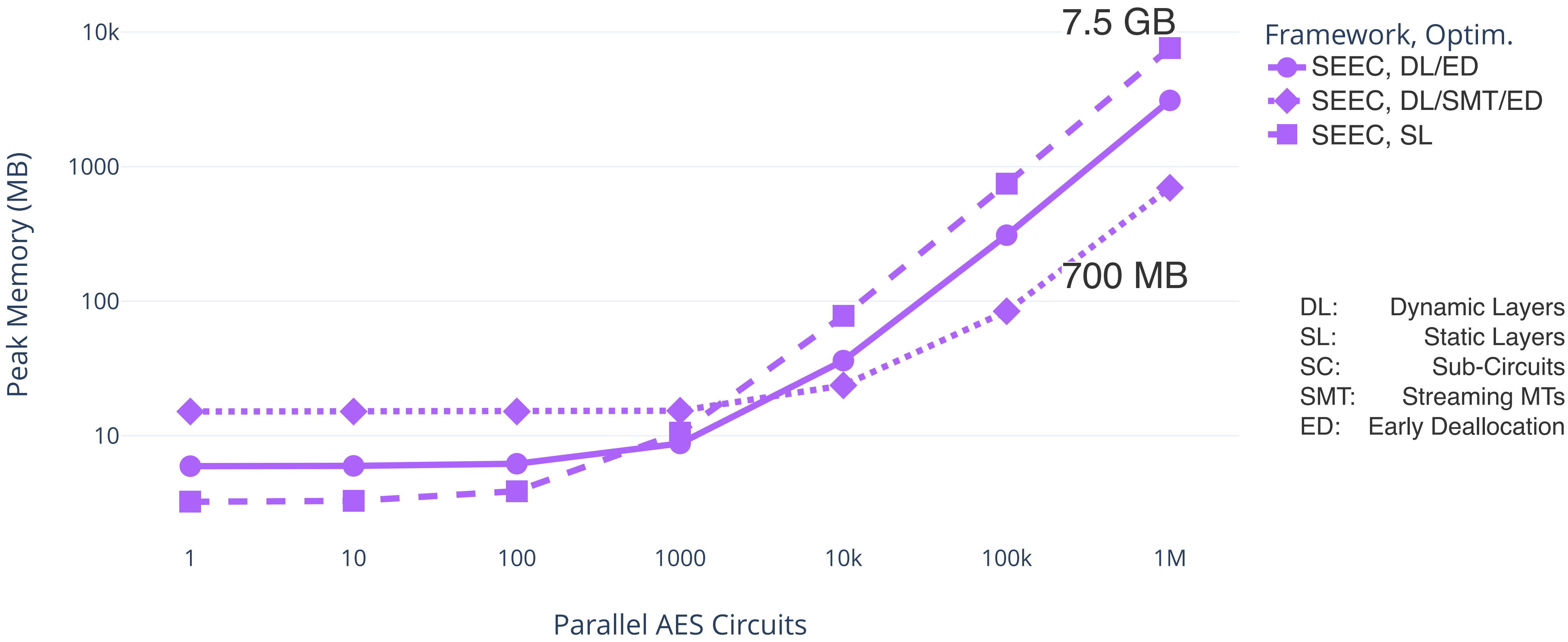
AES-CBC: Reduced Memory via Sub-Circuits



DL: Dynamic Layers
SL: Static Layers
SC: Sub-Circuits

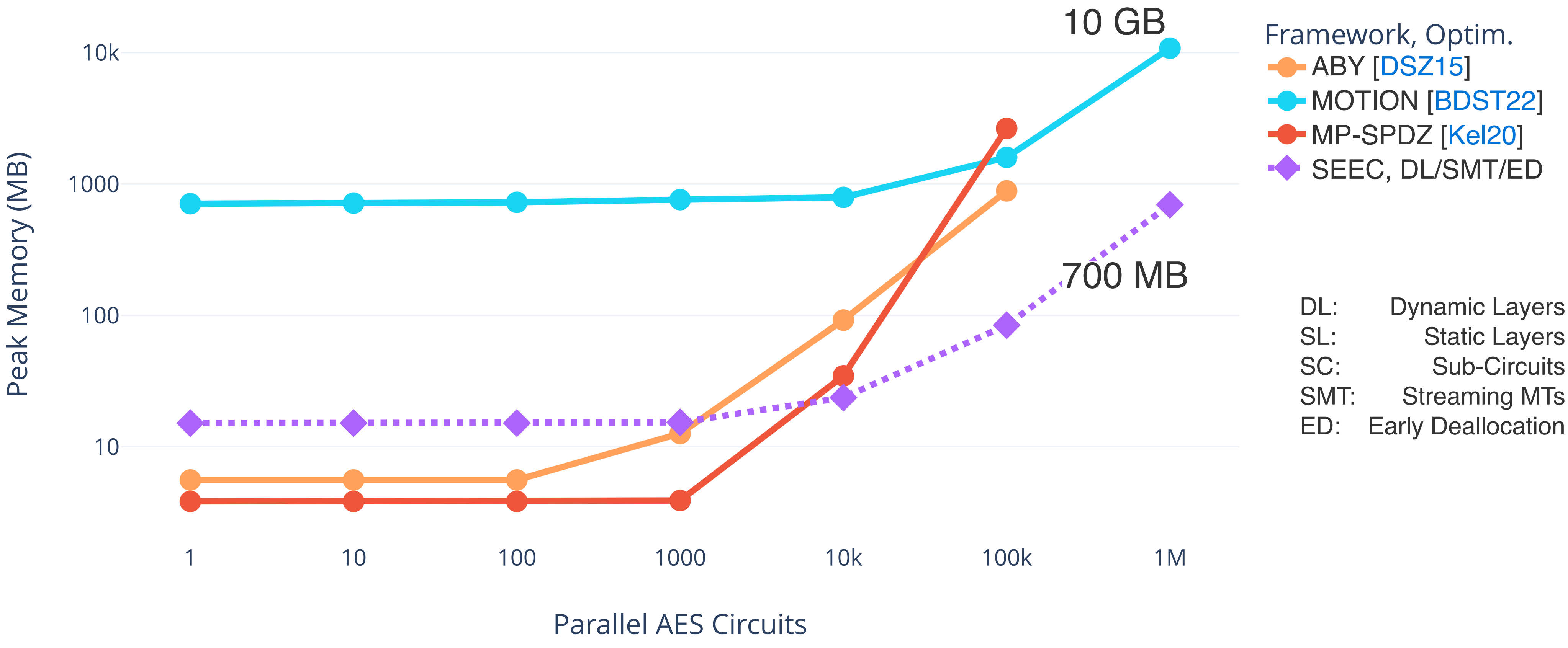
AES: Reduced SIMD Memory Usage

ENCRYPTO CRYPTOGRAPHY AND PRIVACY ENGINEERING



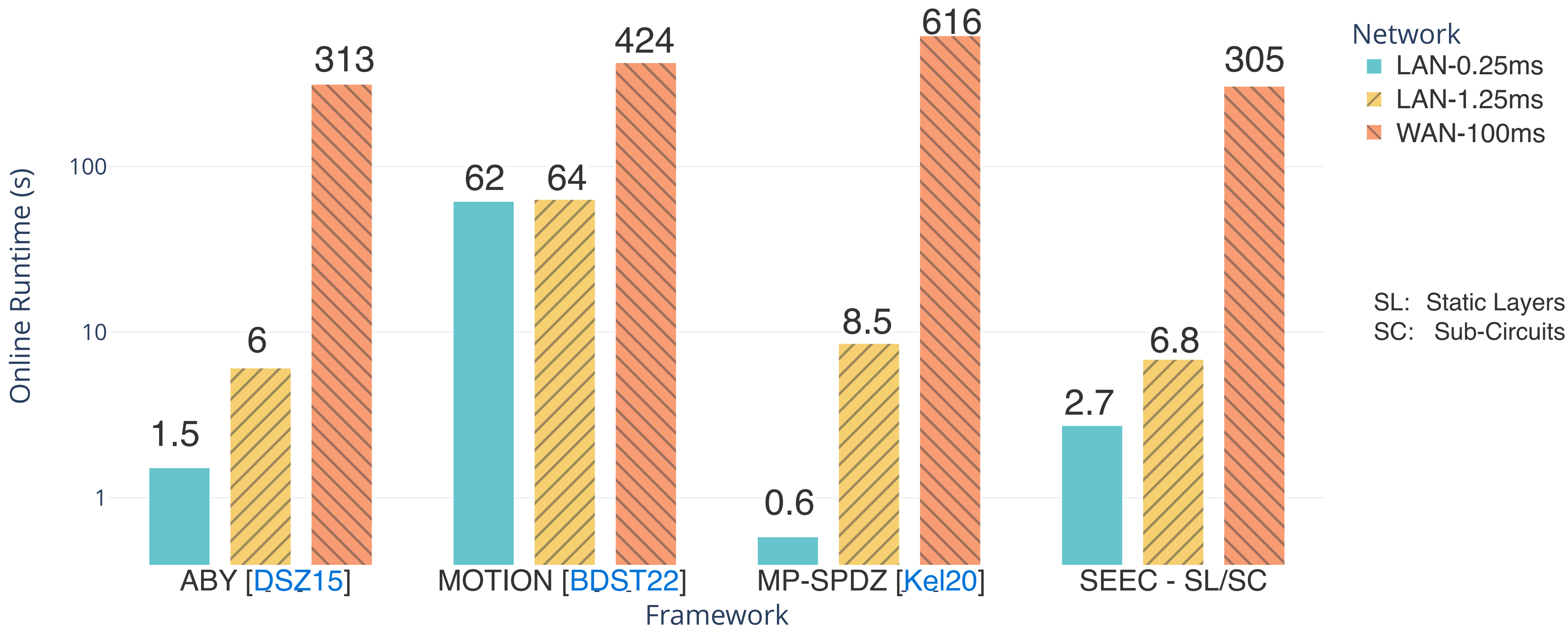
DL: Dynamic Layers
SL: Static Layers
SC: Sub-Circuits
SMT: Streaming MTs
ED: Early Deallocation

AES: Reduced SIMD Memory Usage



DL: Dynamic Layers
SL: Static Layers
SC: Sub-Circuits
SMT: Streaming MTs
ED: Early Deallocation

AES-CBC Runtime: Effect of Latency



SL: Static Layers
SC: Sub-Circuits

Summary

Memory-Safety

&

Memory-Efficiency

Sub-Circuits

```
#[sub_circuit]
fn process(...)
```

SIMD

Up to 15× - 1,983× less memory than MOTION [BDST22].

	Predictability	Reliability
ABY	✓	✗
MP-SPDZ	✗	✓
MOTION	✗	✓
SEEC	✓	✓

Questions?

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

github.com/encryptogroup/SEEC

Made with

typst draw.io SVG REPO

References

[ALSZ13] G. ASHAROV, Y. LINDELL, T. SCHNEIDER, M. ZOHNER. “More Efficient Oblivious Transfer and Extensions for Faster Secure Computation”. In: CCS, 2013.

[BCG+19] E. BOYLE, G. COUTEAU, N. GILBOA, Y, ISHAI, L. KOHL, P. RINDAL, and P. SCHOLL. “Efficient two-round OT extension and silent non-interactive secure computation.” In CCS, 2019.

[GMW87] O. GOLDREICH, S. MICALI, A. WIGDERSON. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In STOC, 1987.

[Bea92] D. BEAVER. “Efficient Multiparty Protocols Using Circuit Randomization”. In CRYPTO, 1992.

[DSZ15] D. DEMMLER, T. SCHNEIDER, M. ZOHNER. “ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation”. In NDSS, 2015.

[CCPS19] ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction”. In: CCSW@CCS, 2019.

[Kel20] M. KELLER. “MP-SPDZ: A Versatile Framework for Multi-Party Computation”. In CCS, 2020.

[PSSY21] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. “ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation”. In USENIX Security, 2021.

[BDST22] L. BRAUN, D. DEMMLER, T. SCHNEIDER, O. TKACHENKO. “MOTION - A Framework for Mixed-Protocol Multi-Party Computation”. In TOPS, 2022.

[BHK+23] L. BRAUN, M. HUPPERT, N. KHAYATA, T. SCHNEIDER, O. TKACHENKO. “FUSE - Flexible File Format and Intermediate Representation for Secure Multi- Party Computation”. In AsiaCCS 2023.

Appendix

Future Work

- Expanding `Secret` API
- SIMD `#[sub_circuit]` macro
- Usability improvements

- Protocol composability
- Optional register storage
- Sub-Circuit SIMD-vectorization

- Sub-Circuit output deallocation

- OT-based interleaved setup
- Interleaved function dependent preprocessing

- Asynchronous Evaluation
- QUIC Channels
- Multi-Party + Malicious Protocols

Benchmarking Tool

```
net_settings = ["RESET", "LAN", "WAN"]
repeat = 5

[[bench]]
framework = "SEEC"
target = "bristol"
tag = "seec_aes_ctr_no_setup"
compile_flags = ["../../../circuits/
advanced/aes_128.bristol"]
flags = ["--insecure-setup"]
cores = [0,1]
[bench.compile_args]
"--simd" = ["1", "10", "100", "1000",
"10000", "100000", "1000000"]
```

```
[[bench]]
framework = "MOTION"
tag = "motion_aes_no_setup"
target = "aes128"
flags = ["--insecure-setup"]
cores = [0,1]
[bench.args]
"--num-simd" = ["1", "10", "100",
"1000", "10000", "100000", "1000000"]
```

encryptogroup/mpc-bench

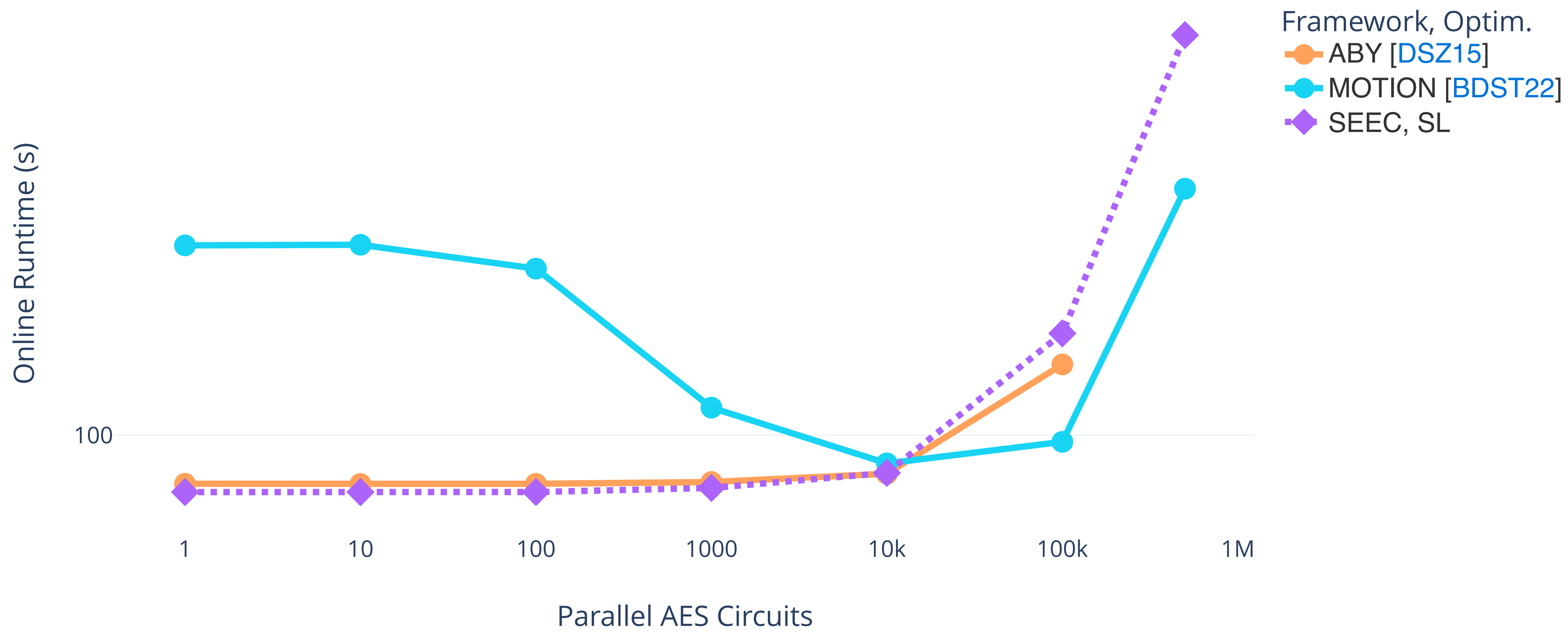
Sub-Circuit Iteration


```
class BaseLayerIter:
  graph: Graph,
  inputs_needed: [int]
  next_layer: Queue<Id>
  current_layer: Queue<Id>
  prev_interactive: Queue<Id>

  def __init__(self, i, G, I, O):
    self.graph = G
    self.inputs_needed = [len(pred(v)) for v in G.V]
    self.next_layer = Queue()
    self.current_layer = Queue()
    self.prev_interactive = Queue()

  # decrement successors' inputs_needed of v and add to queue if
  # no more inputs are needed
  def add_ready_successors(self, v, queue):
    for s in succ(v):
      self.inputs_needed[s] -= 1
      if self.inputs_needed[s] == 0:
        queue.push_back(s)
```

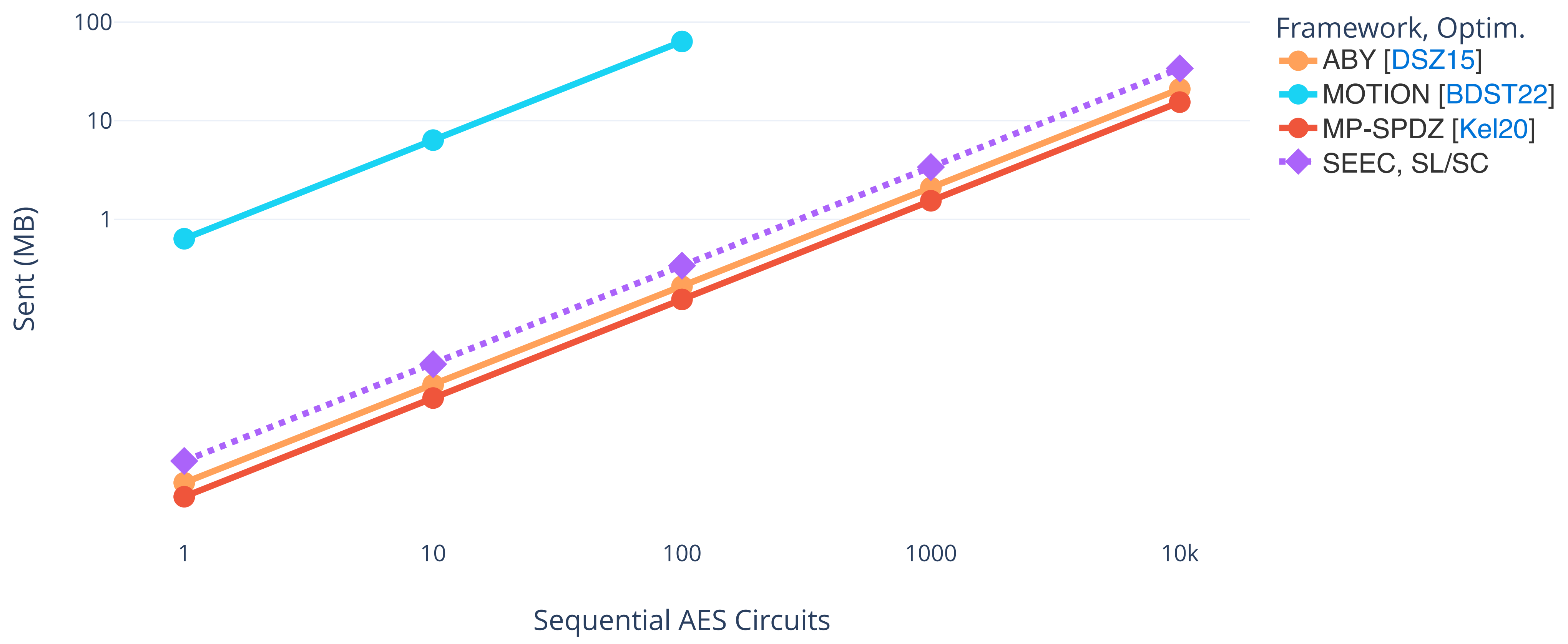
Sub-Circuit Iteration

```
  # returns next layer in topological order or None
  def next(self):
    # next layer queue becomes the current layer
    swap(next_layer, current_layer)
    # current layer is empty, as we popped all elements
    # in previous iteration
    layer = Layer()

    # check previous interactive gates successors for current
layer
    while v = prev_interactive.pop_front():
      self.add_ready_successors(v, inputs_needed,
current_layer)

    # pop from the front of the queue until empty
    while v = current_layer.pop_front():
      if G.is_interactive(v):
        layer.push_interactive(v)
        # consider successors in **next** iteration
        self.prev_interactive.push_back(v)
      else:
        layer.push_non_interactive(v)
        # potentially add successors of non-interactive gate
to
        # **current layer**
        self.add_ready_successors(v, inputs_needed,
current_layer)

    if layer.is_empty():
      # we have yielded all gates and this iterator is
exhausted
      return None
    else:
      # this layer can be evaluated in one round
      return layer
```

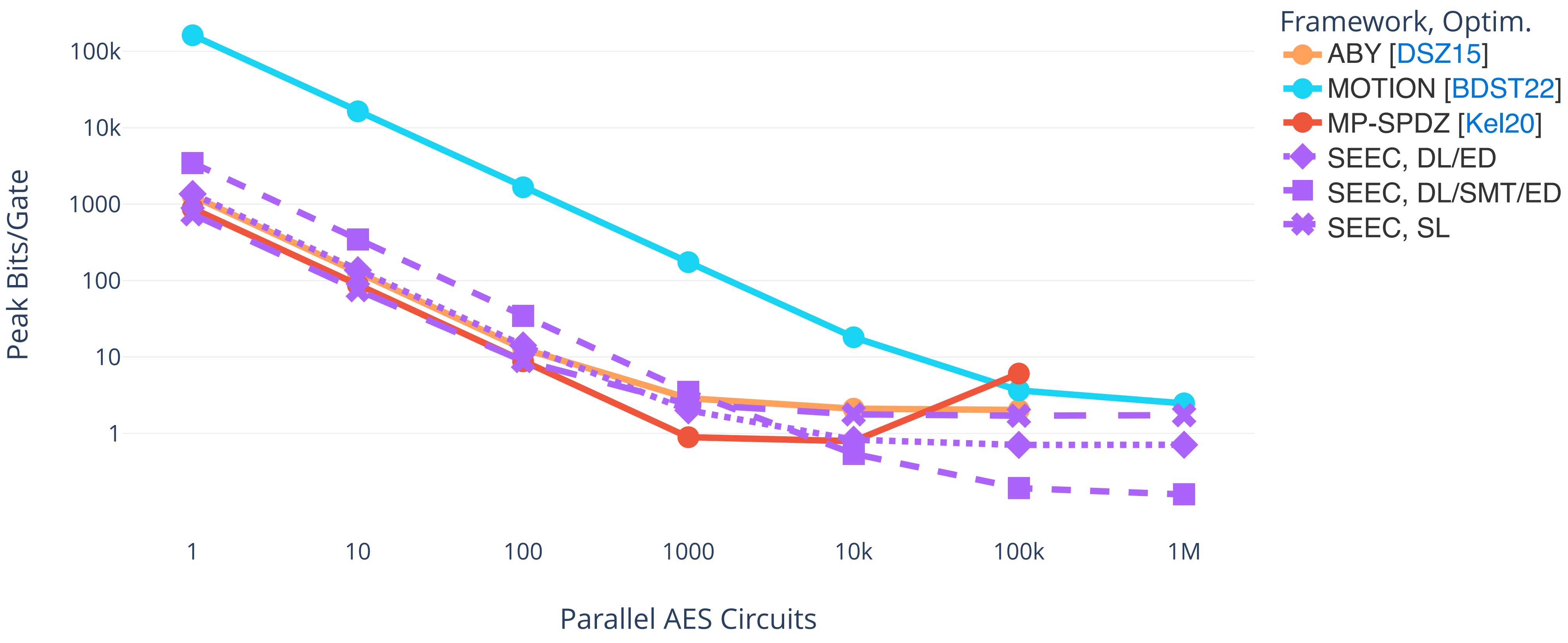
SHA-256: Effect of Nagle's Algorithm



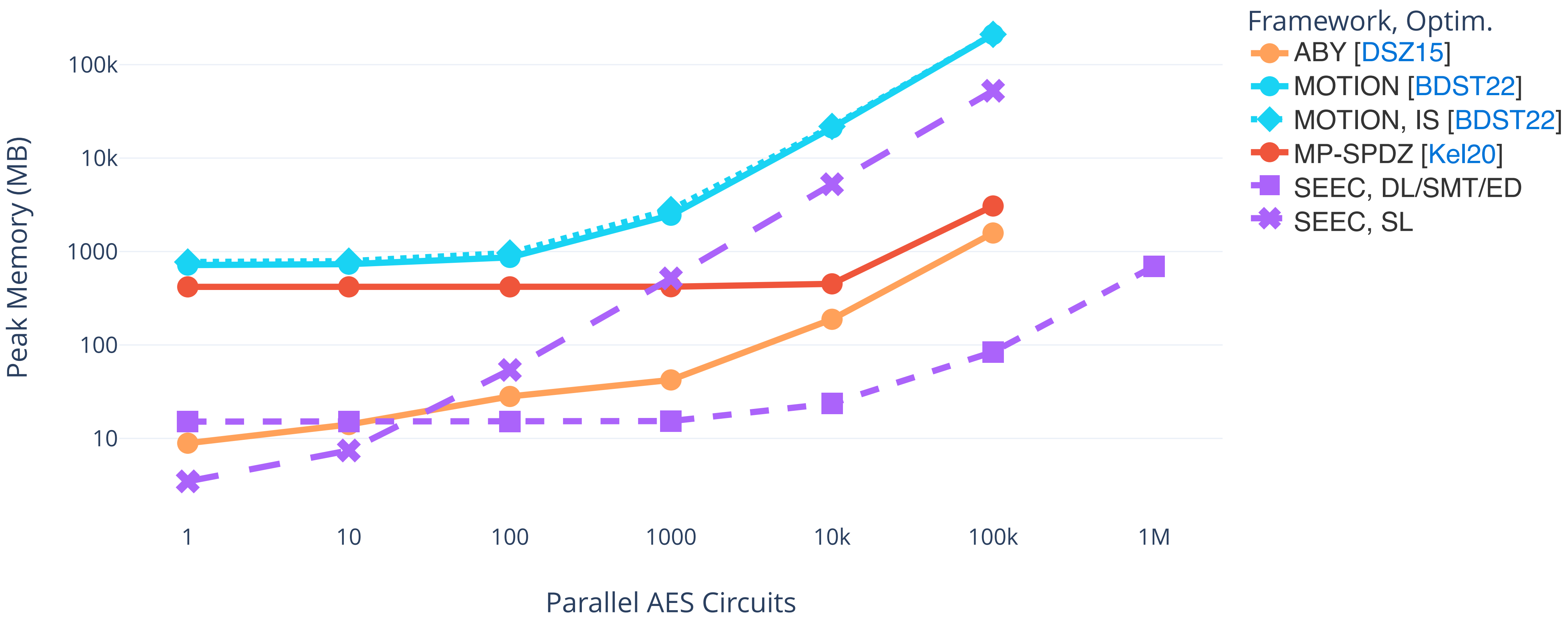
AES-CBC: Async. Communication Overhead



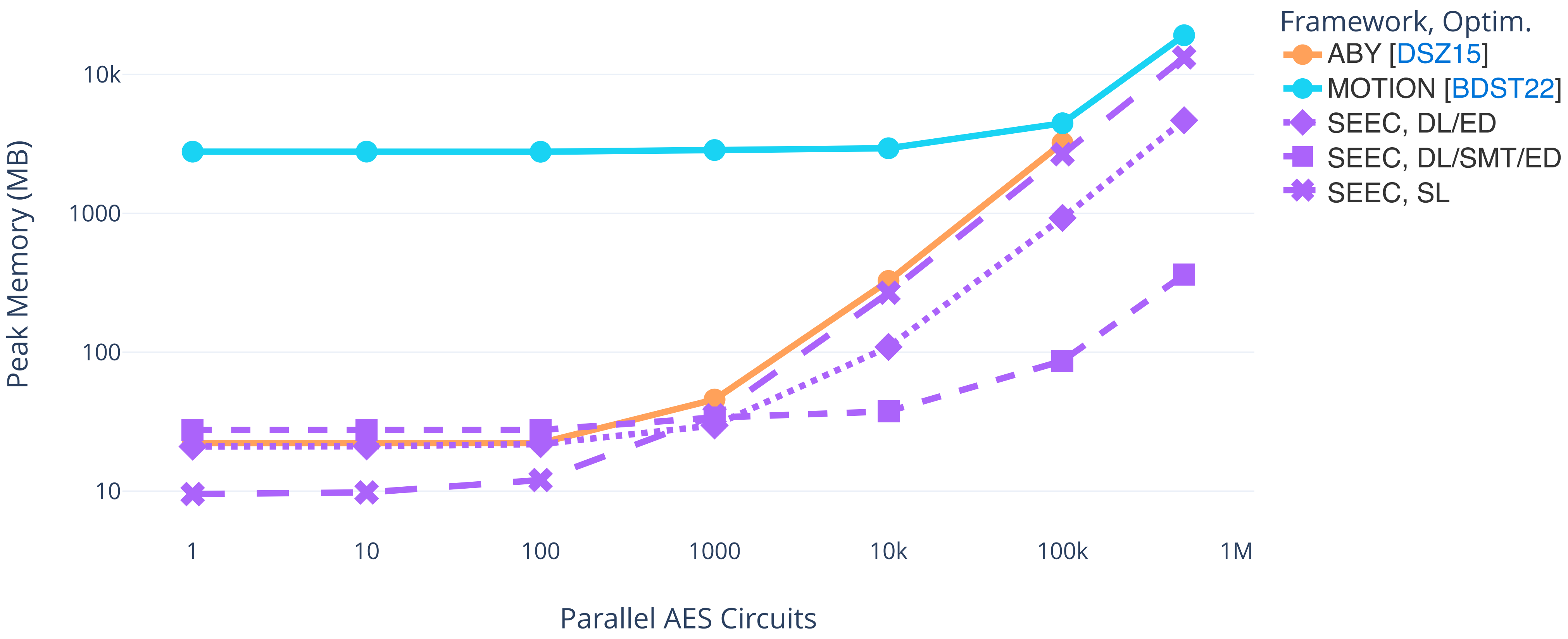
SIMD AES: Peak Bits per Gate



SIMD AES: Impact of Setup



SHA-256: Reduced SIMD Memory Usage



SEEC: System Architecture (slightly outdated)