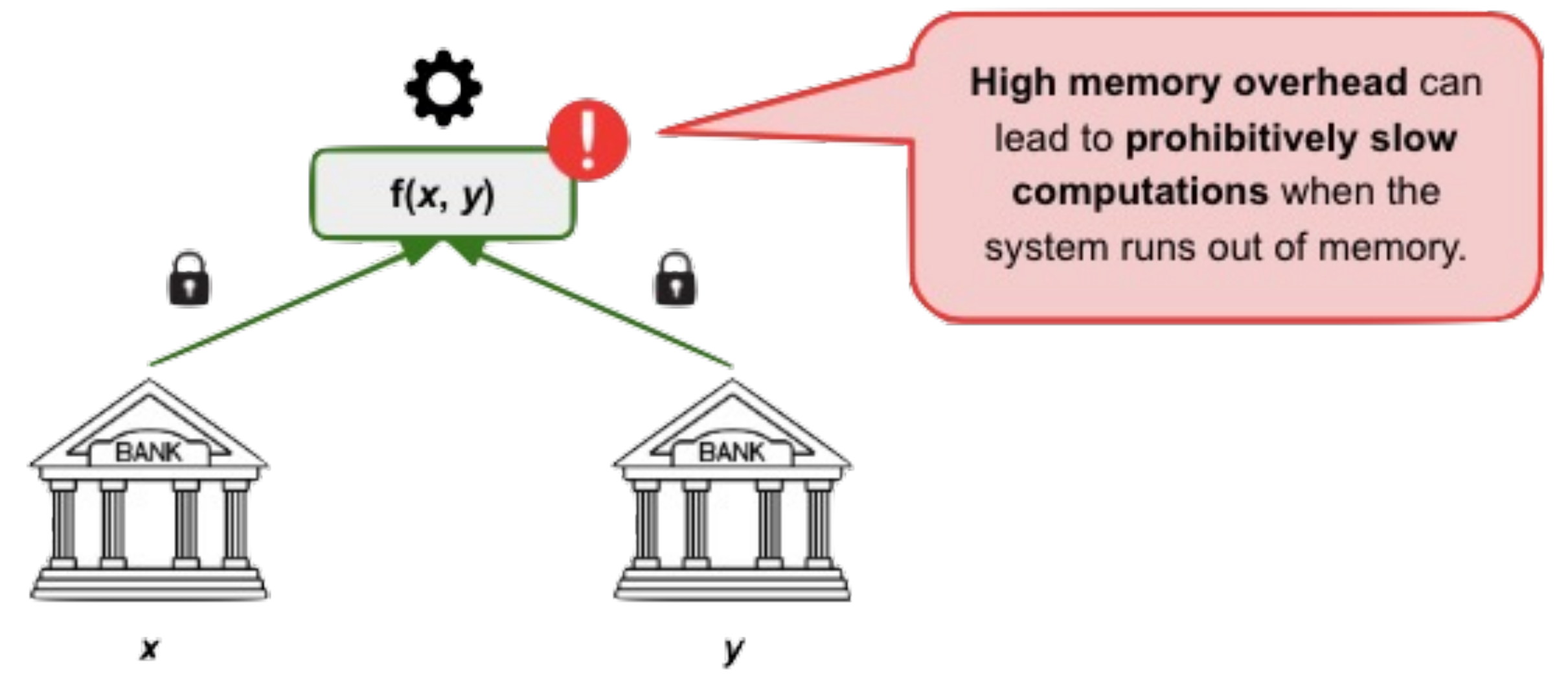
Speculative Memory Programming for Secure Computation

Alice Yeh
University of California, Berkeley
External collaborator: Sam Kumar

Introduction

Secure computation enables **computing on encrypted data**, with a popular use case being secure multi-party computation, which allows parties to collaboratively compute on their shared data without any party learning the private inputs of another. However, one major barrier to its adoption is the **high memory overhead**, which can lead to **prohibitively slow computation** once systems run out of memory.


f(x, y)
High memory overhead can lead to **prohibitively slow computations** when the system runs out of memory.
BANK
BANK
x
y

One case of secure computation is in fraud detection, where banks collaboratively compute on shared data to discover fraud.

Background

Speculative Execution

Speculative execution is an optimization where the CPU operates on instructions out of order to fetch and compute data that may be needed later. Previous works have used speculative execution to parallelize data fetching to speed up operations [1].

MAGE [KCP21]

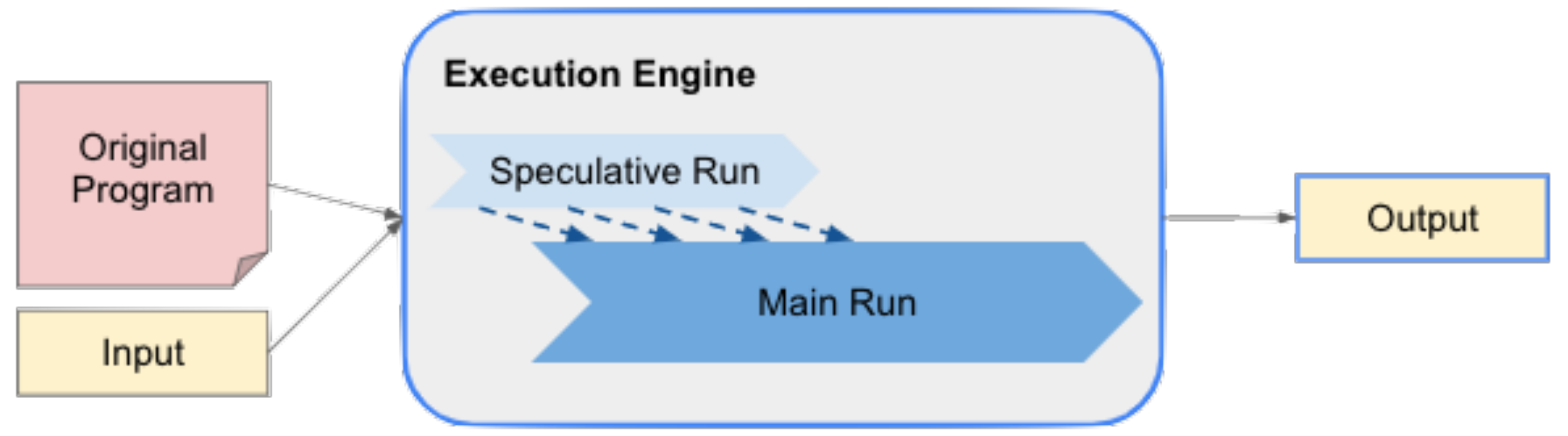
MAGE is an execution engine that can efficiently run SCs at speeds close to machines with unbounded physical memory. However, MAGE requires DSL programs for its planner and planning itself takes time proportional to program execution [2].

3PO [BGK+22]

3PO builds on MAGE's idea in the context of far memory. A prominent idea is an in-kernel tracer that generates a "tape" of page accesses, which can be used to determine pages to prefetch [3].

Design & Implementation

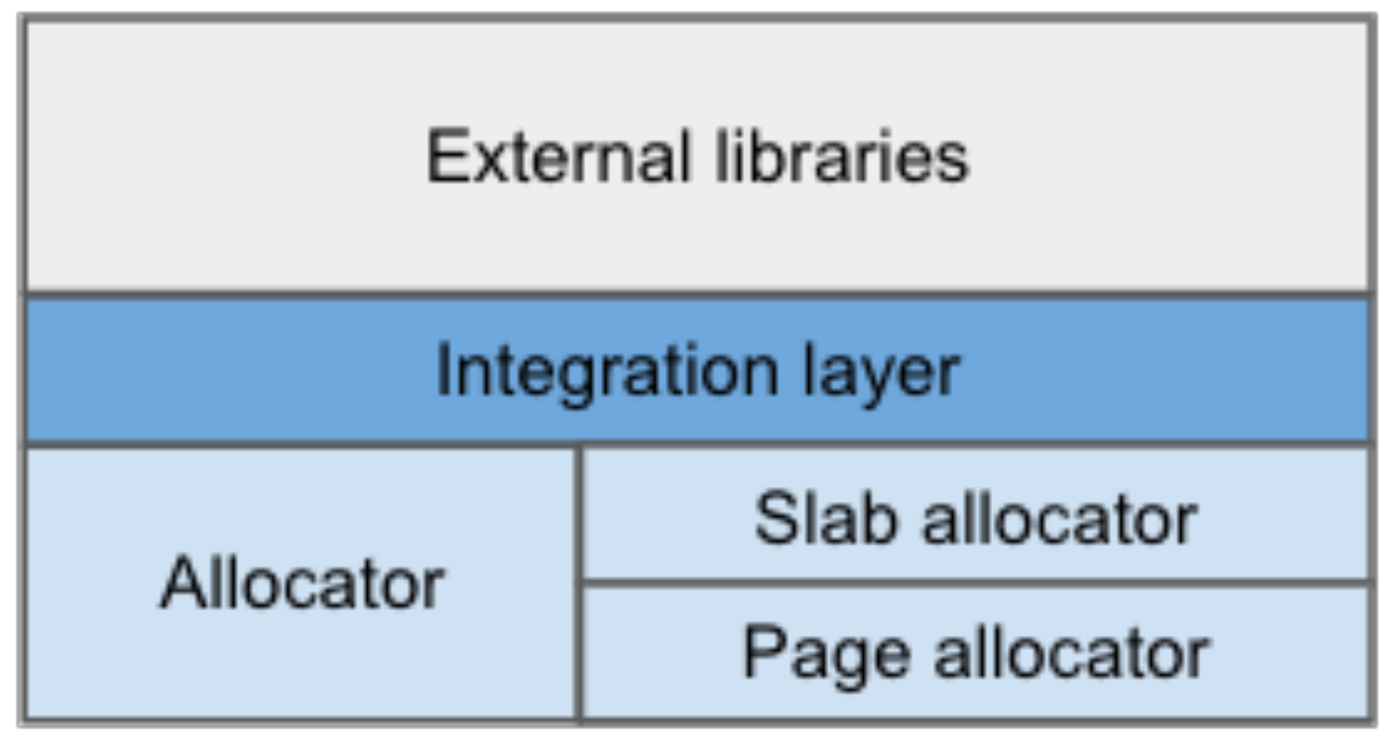
Architecture



Single pass execution on the **original program** using ***speculative execution*** to get memory access patterns and ***hint-passing to the OS*** about future memory accesses.

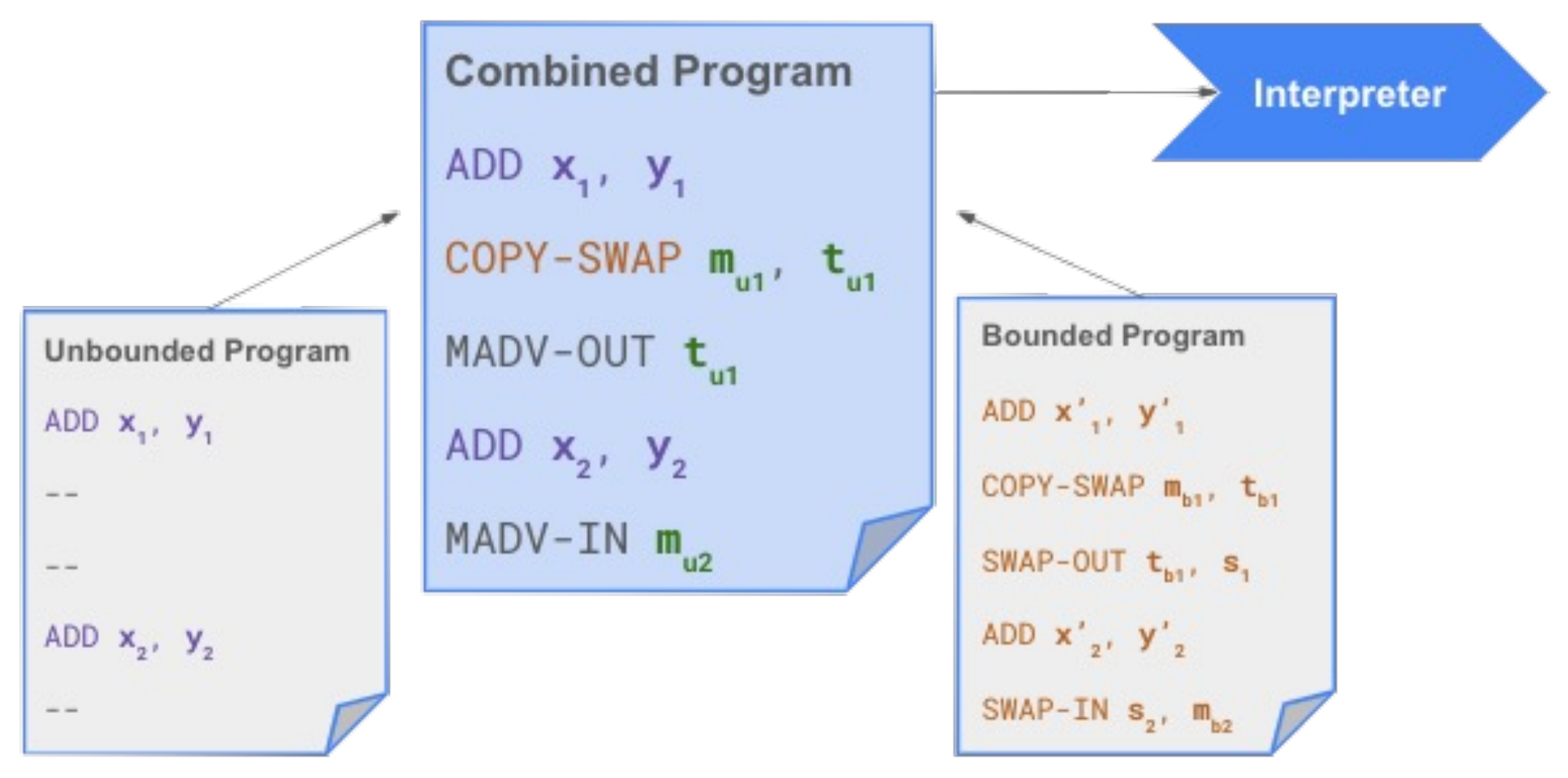
Making Speculation Efficient Using Oblivious Allocation

In order to be efficient, the speculative thread needs to have a **small memory footprint** and to **run quickly**. We observe, however, that it *doesn't* need to be correct, as long as its memory access patterns are consistent. Therefore, our idea is to create a **content-oblivious region for ciphertext allocation** in the speculative run and map this region to just one physical page.



Oblivious allocation stack, consisting of the oblivious allocator and an integration layer that allows developers to easily annotate their libraries to mark objects for oblivious allocation.

Passing Hints to the OS about Future Page Accesses



```
Unbounded Program
ADD x1, y1
--
--
ADD x2, y2
--
```

```
Combined Program
ADD x1, y1
COPY-SWAP mu1, tu1
MADV-OUT tu1
ADD x2, y2
MADV-IN mu2
```

```
Bounded Program
ADD x'1, y'1
COPY-SWAP mb1, tb1
SWAP-OUT tb1, s1
ADD x'2, y'2
SWAP-IN s2, mb2
```

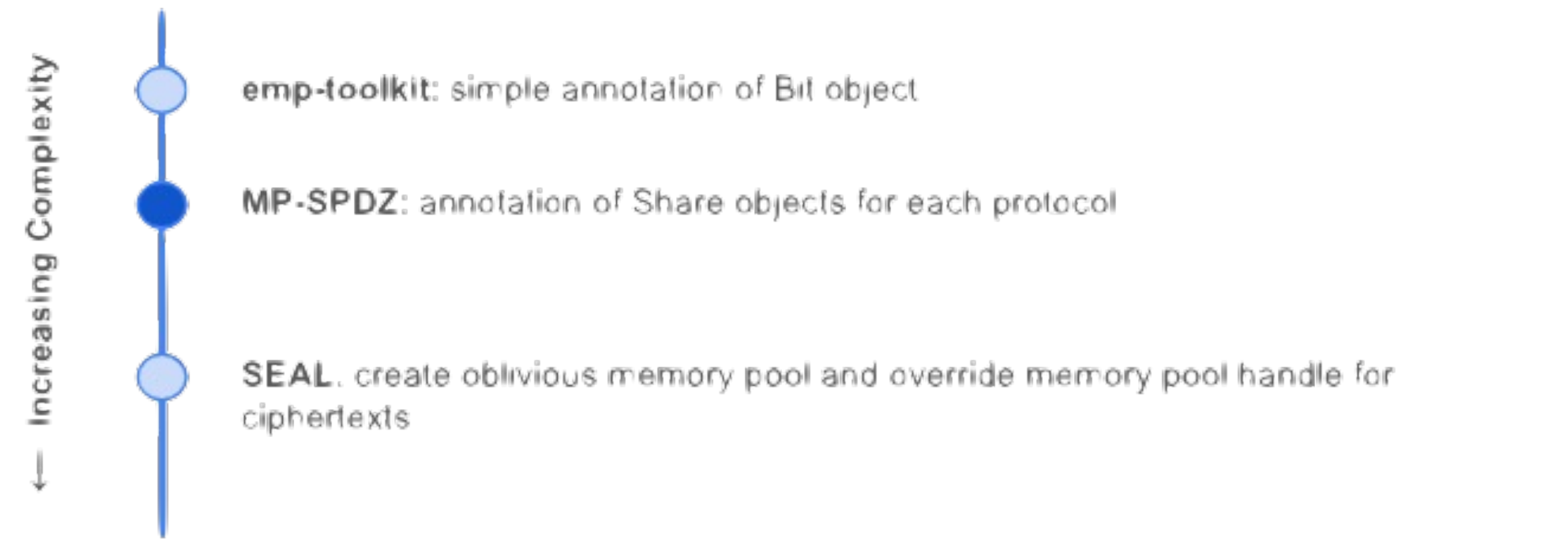
Running a memory program with madvise syscalls through MAGE's interpreter to benchmark execution times.

In the speculative run, we record a "tape" of page accesses, and in the main run, we **pass hints to the OS about these page accesses using the madvise syscall**. This informs to the OS to read in pages that will be needed and to mark pages as unused when they are no longer needed. We validated that hint-passing using madvise **does not significantly worse performance** using MAGE.

Results & Evaluation

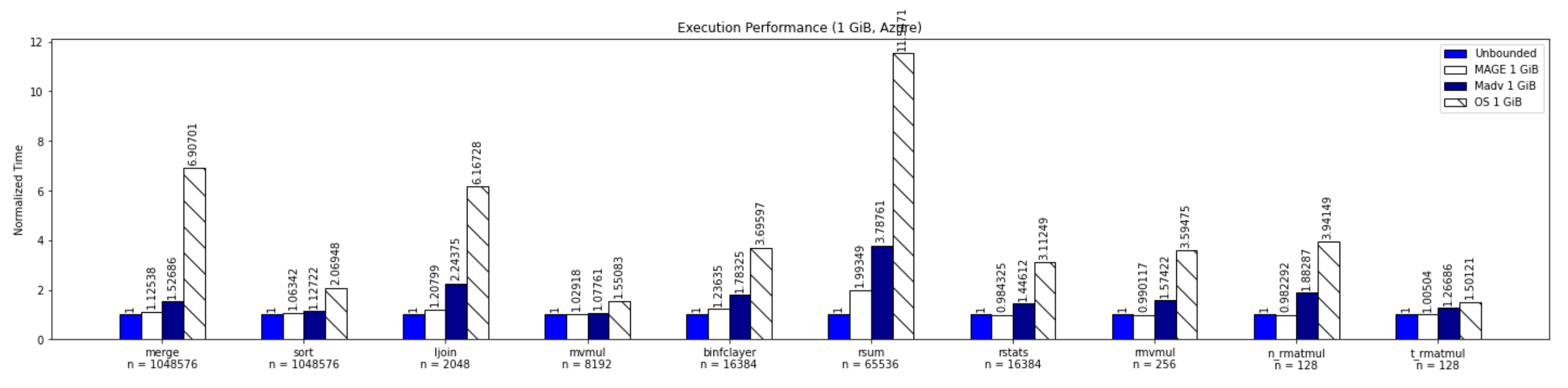
Complexity of Oblivious Annotations

We implemented oblivious annotations in three major crypto libraries [4, 5, 6] and evaluated the complexity of modifying these libraries to support oblivious allocation.

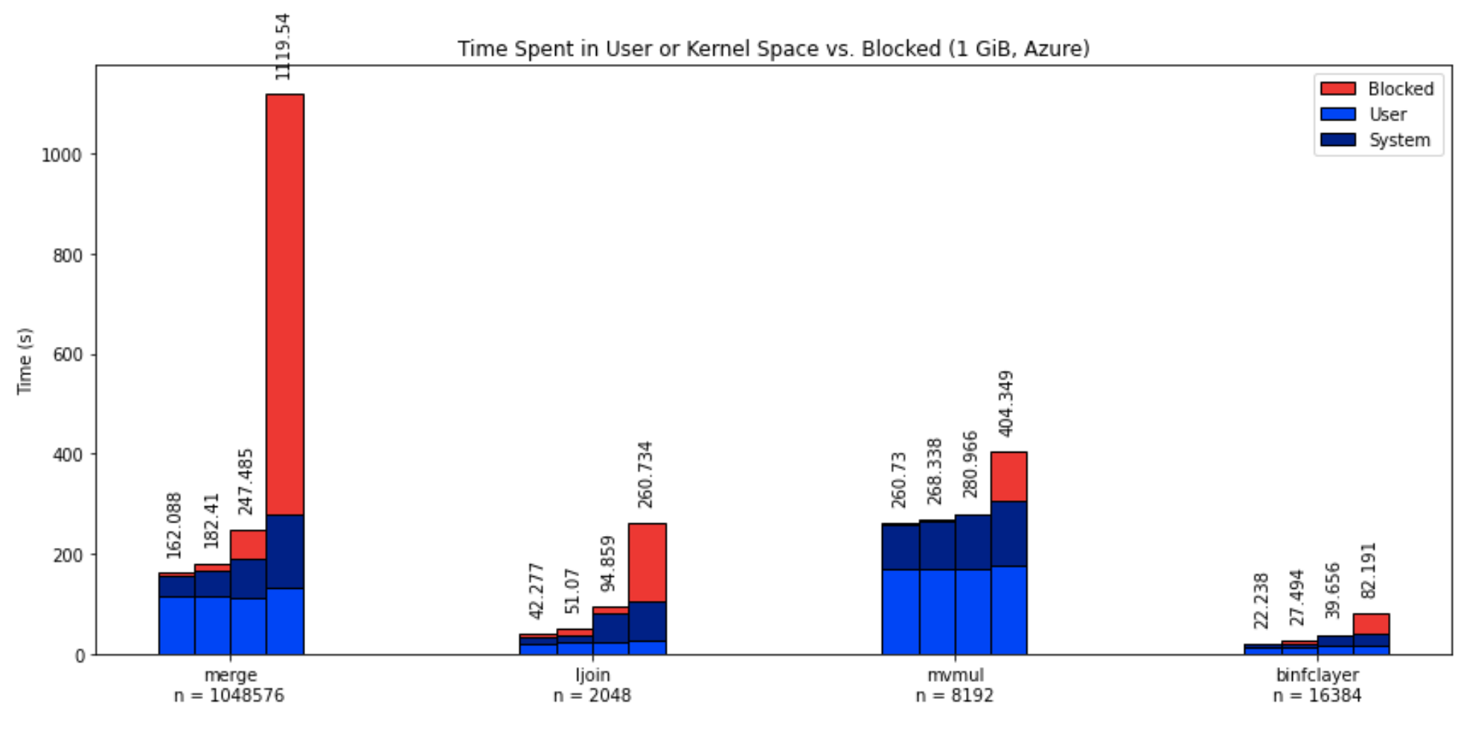

Increasing Complexity
emp-toolkit: simple annotation of Bit object
MP-SPDZ: annotation of Share objects for each protocol
SEAL: create oblivious memory pool and override memory pool handle for ciphertexts

Effect of Hint-Passing on Execution Time

We benchmarked execution time for hint-passing, alongside a system with unbounded memory, MAGE, and a system relying on classical OS paging.



For all workloads, the hint-passing implementation performs within 2x of MAGE. 6 of the workloads perform within 1.5x of MAGE.



Hint-passing successfully reduces the time spent blocked, compared to classical OS paging.

References

[1] F. Chang and G. A. Gibson. Automatic I/O hint generation through speculative execution. 3rd USENIX Symposium on Operating Systems Design and Implementation, 1999.
[2] S. Kumar, D. E. Culler, and R. A. Popa. MAGE: Nearly Zero-Cost Virtual Memory for Secure Computation. 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21), pages 367–385, 2021.
[3] S. K. E. A. A. O. A. P. S. R. S. S. Christopher Branner-Augmon, Narek Galstyan. 3PO: Programmed Far-Memory Prefetching for Oblivious Applications. 2022.
[4] X. Wang, A. J. Malozemoff, and J. Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.
[5] M. Keller. MP-SPDZ: A versatile framework for multi-party computation. Cryptology ePrint Archive, Paper 2020/521, 2020.
[6] Microsoft SEAL. https://github.com/Microsoft/SEAL, 2020.