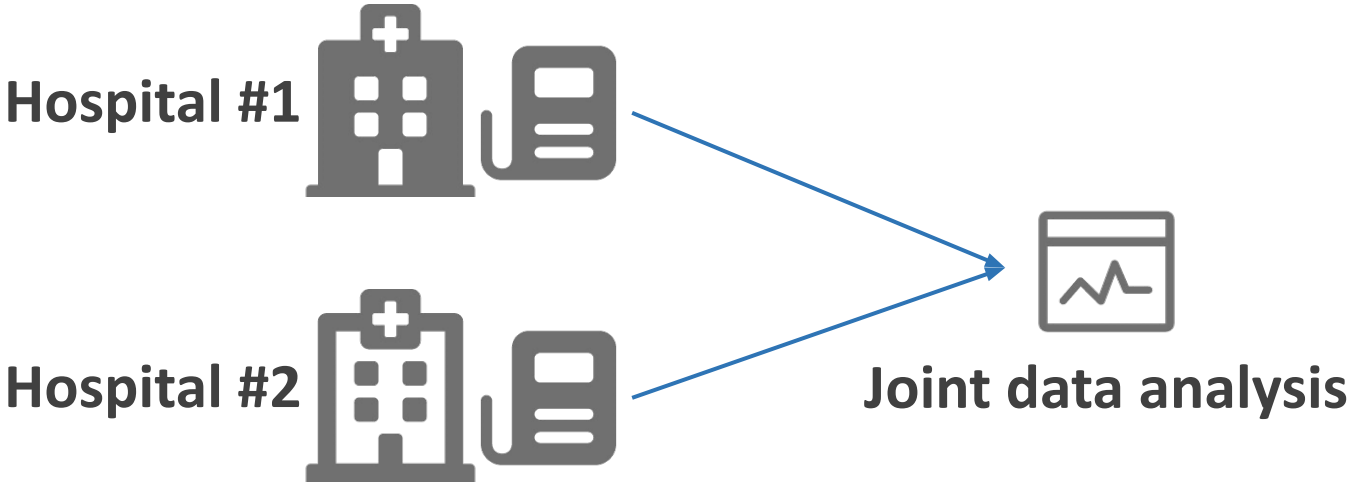
Osprey: Transparent and Efficient Virtual Memory for Secure Computation

Yicheng Liu, Alice Yeh, Harry Xu, Raluca Ada Popa, Sam Kumar

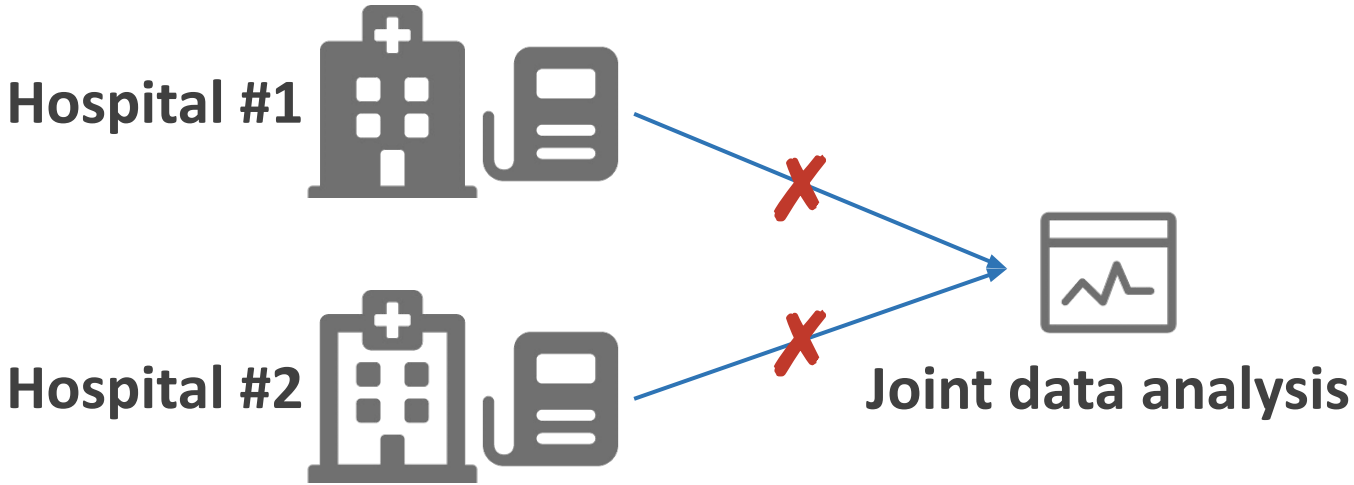
UCLA

Berkeley
UNIVERSITY OF CALIFORNIA

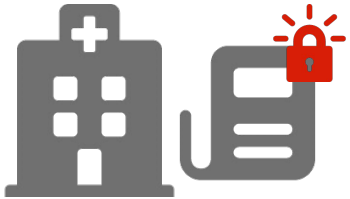
Data is Most Valuable When Combined...



…But Combining It is Exactly What Privacy Forbids

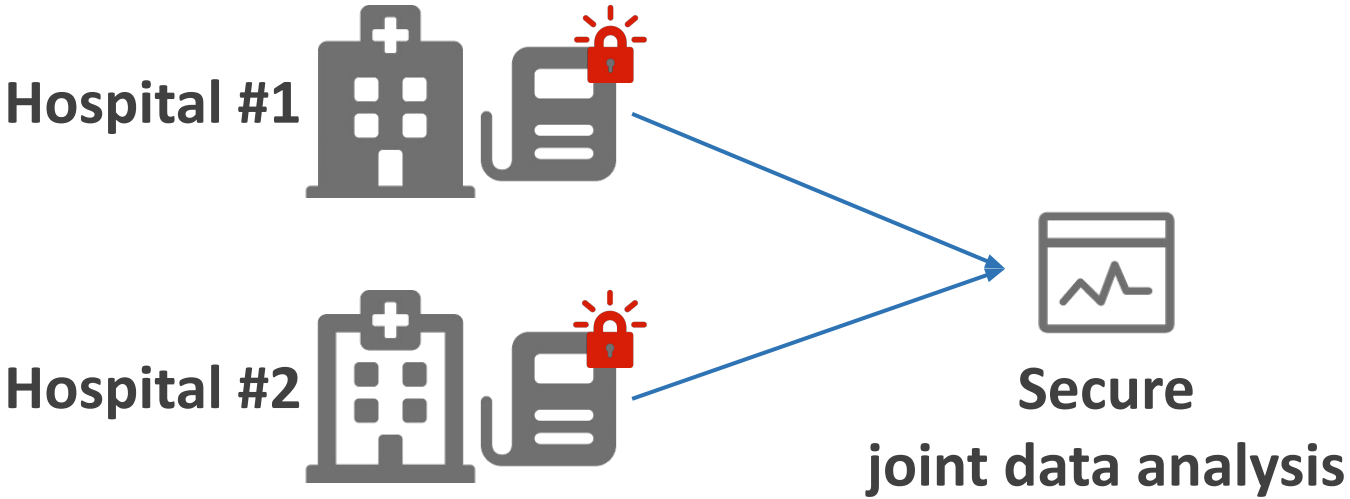


Secure Computation (SC) is a Way Out

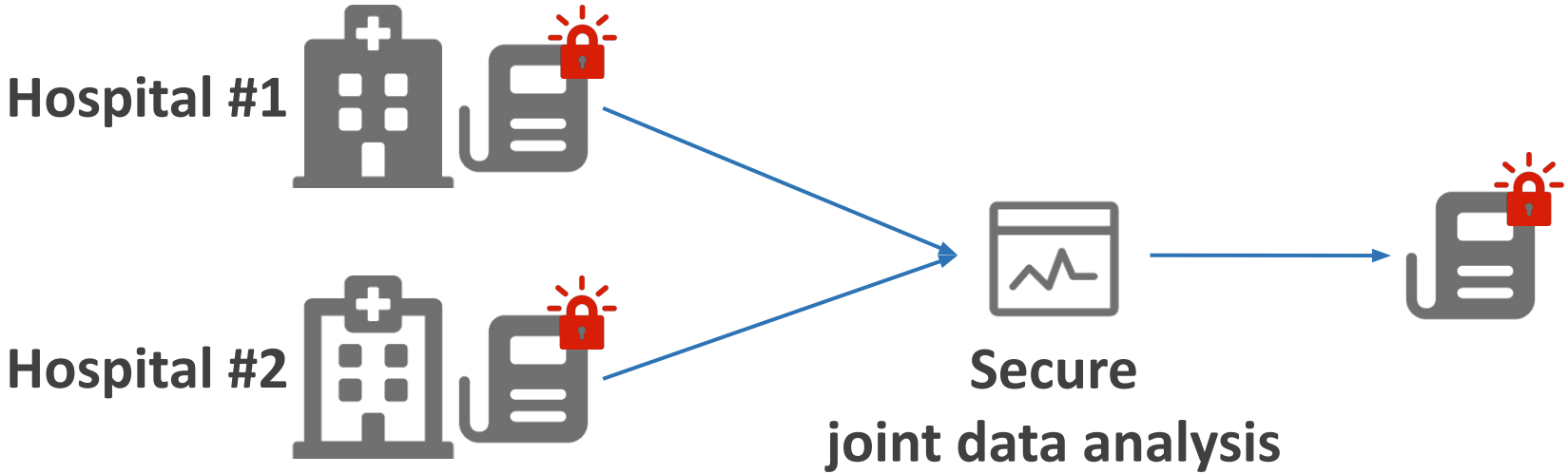
Hospital #1 

Hospital #2 

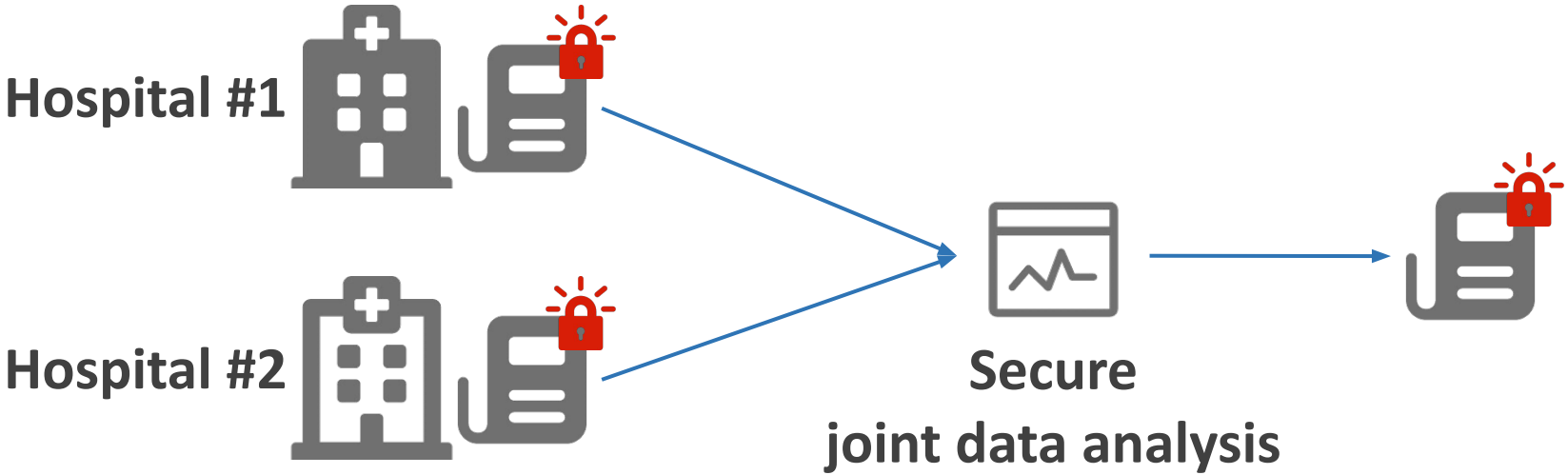
Secure Computation (SC) is a Way Out



Secure Computation (SC) is a Way Out



Secure Computation (SC) is a Way Out

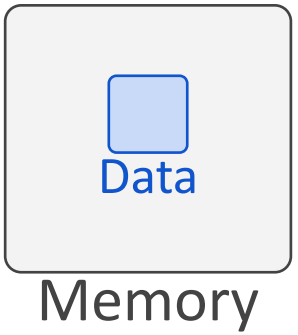


But with large memory overhead…

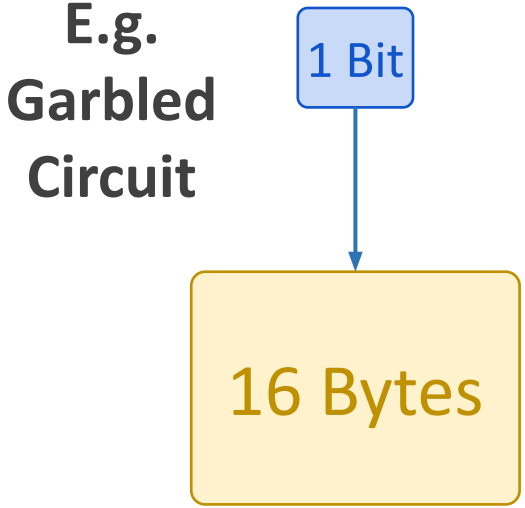
Memory Overhead of Secure Computation



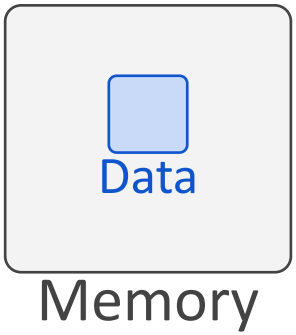
Memory Overhead of Secure Computation



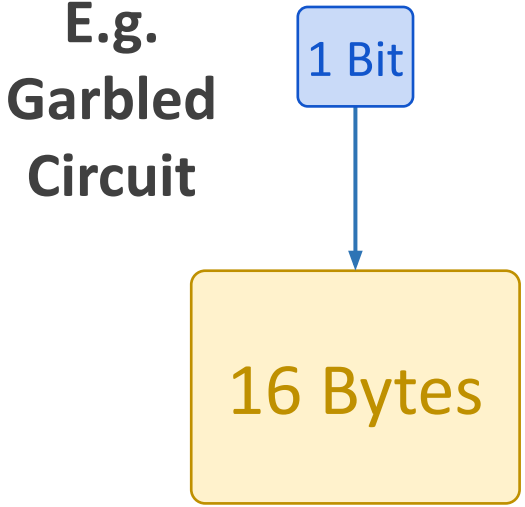
Memory Overhead of Secure Computation



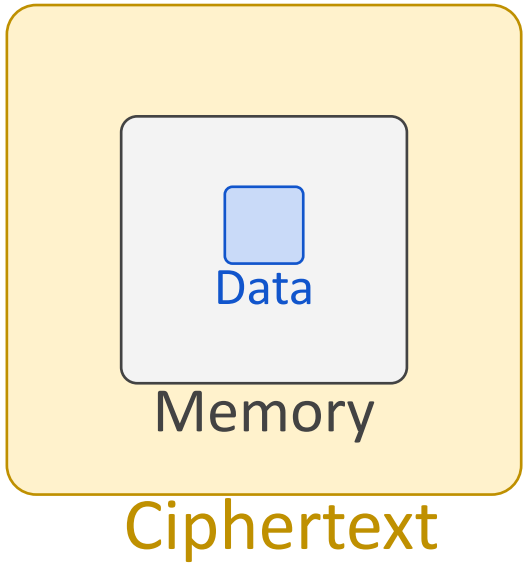
128× Memory Inflation



Memory Overhead of Secure Computation

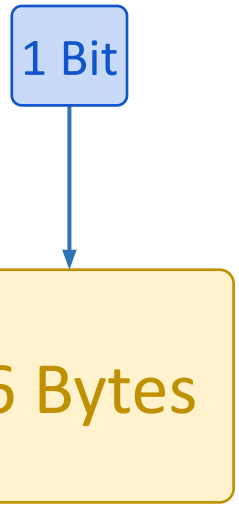


128× Memory Inflation

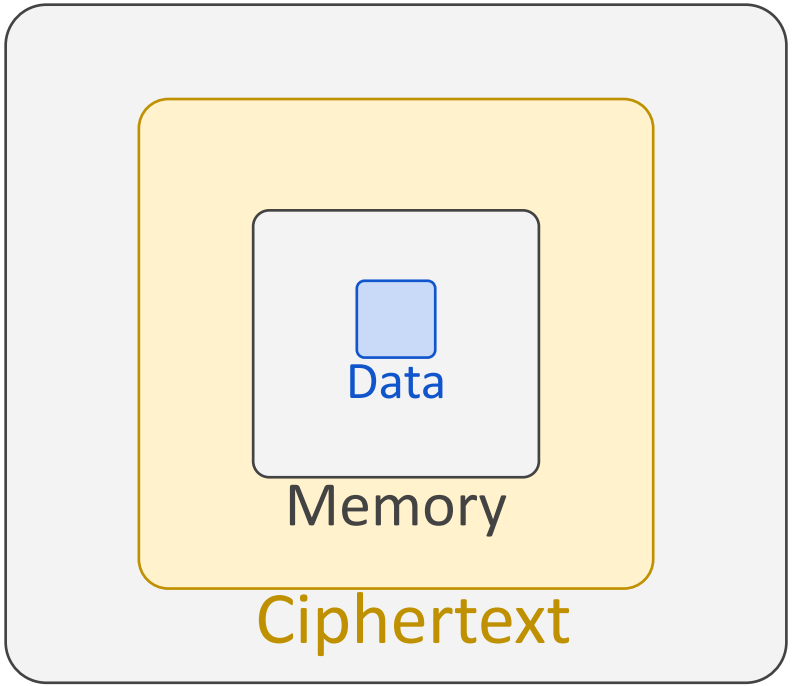


Memory Overhead of Secure Computation

E.g. Garbled Circuit

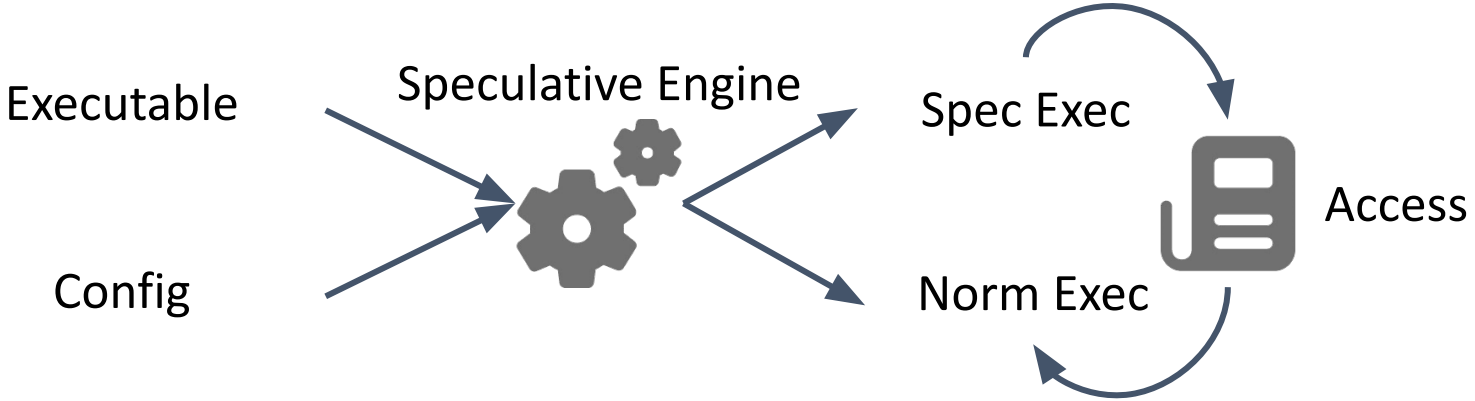


128× Memory Inflation



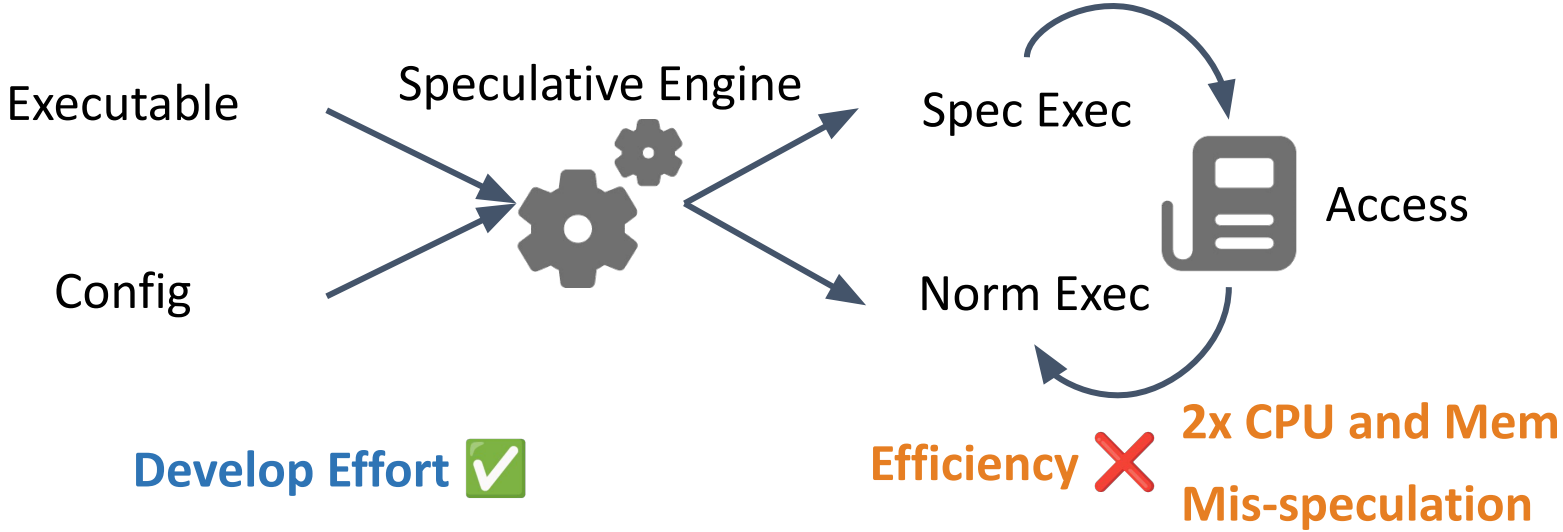
Runtime Solution: Speculative Execution [Fraser03]

- Insight: Get access ahead of time by speculative execution



Runtime Solution: Speculative Execution [Fraser03]

- Insight: Get access ahead of time by speculative execution



Compile-Time Solution: Memory Programming [Kumar21]

- Observation: Secure computation is input-oblivious ← Secure requirement
  - Memory access is independent from input

Input-Obliviousness

- Observation: Secure computation is input-oblivious ← Secure requirement
  - Memory access is independent from input

```
// Program: Find min
uint a = input("alice");
uint b = input("bob");
uint c = min(a, b);
```

Input-Obliviousness

- Observation: Secure computation is input-oblivious ← Secure requirement
  - Memory access is independent from input

```
// Program: Find min
uint a = input("alice");
uint b = input("bob");
uint c = min(a, b);
```

```
// Non-oblivious
uint min(uint a, uint b) {
    if (a < b) return a;
    return b;
}
```

Input-Obliviousness

- Observation: Secure computation is input-oblivious ← Secure requirement
  - Memory access is independent from input

```
// Program: Find min
uint a = input("alice");
uint b = input("bob");
uint c = min(a, b);
```

```
// Non-oblivious
uint min(uint a, uint b) {
    if (a < b) return a;
    return b;
}
```

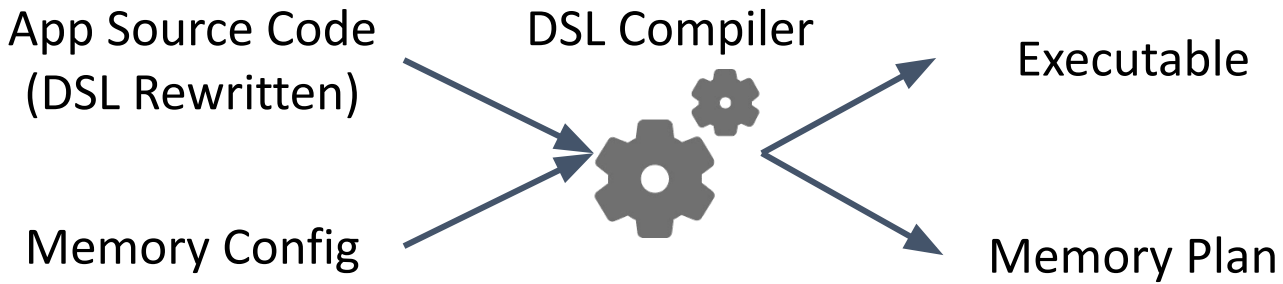
```
// Oblivious
uint min(uint a, uint b) {
    uint less = a < b;
    uint cond = 0 - less;
    return ((a ^ b) & cond) ^ b;
}
```

Compile-Time Solution: Memory Programming [Kumar21]

- Observation: Secure computation is input-oblivious ← Secure requirement
  - Memory access is independent from input → **Preplan**

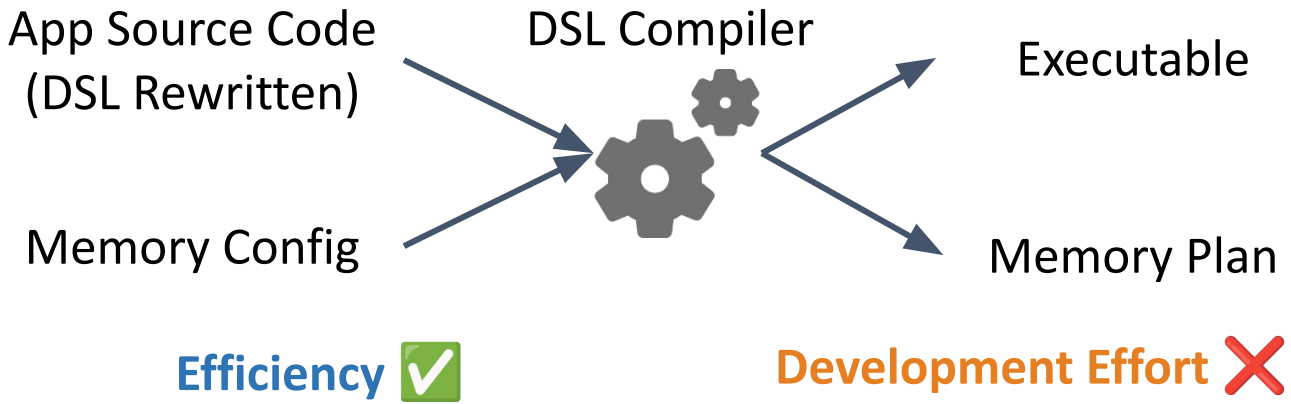
Compile-Time Solution: Memory Programming [Kumar21]

- Observation: Secure computation is input-oblivious ← Secure requirement
  - Memory access is independent from input → **Preplan**



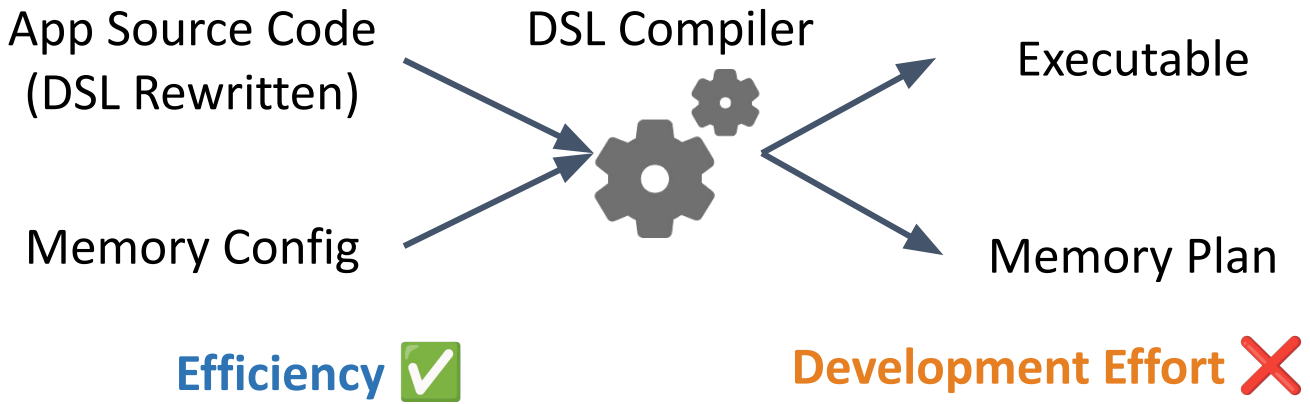
Compile-Time Solution: Memory Programming [Kumar21]

- Observation: Secure computation is input-oblivious ← Secure requirement
  - Memory access is independent from input → **Preplan**



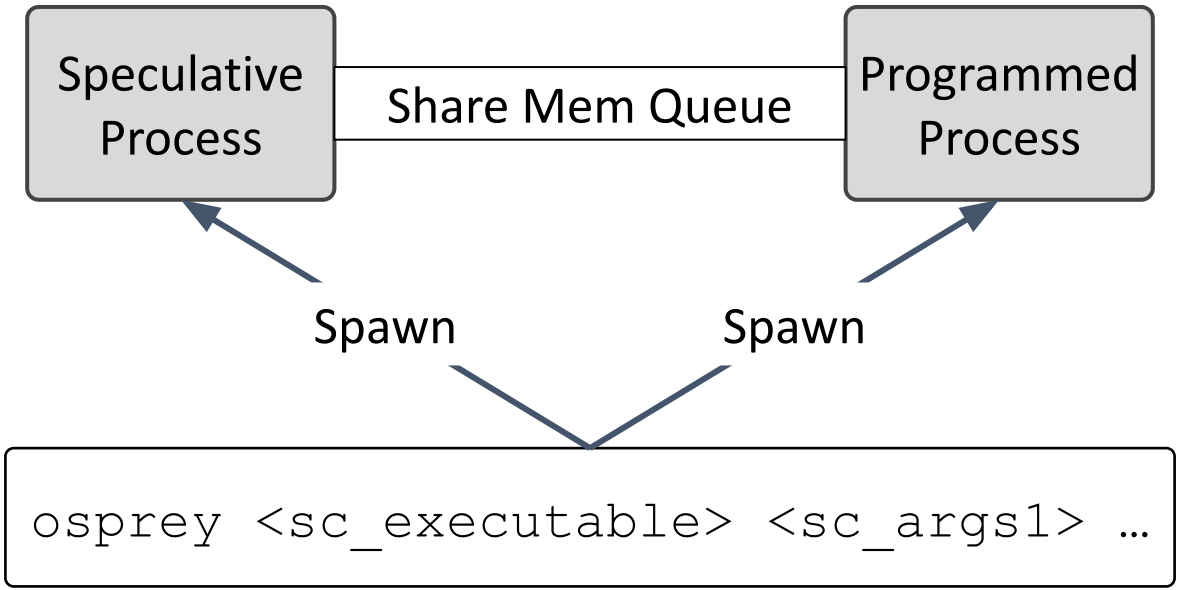
Compile-Time Solution: Memory Programming [Kumar21]

- Observation: Secure computation is input-oblivious ← Secure requirement
  - Memory access is independent from input → **Preplan**

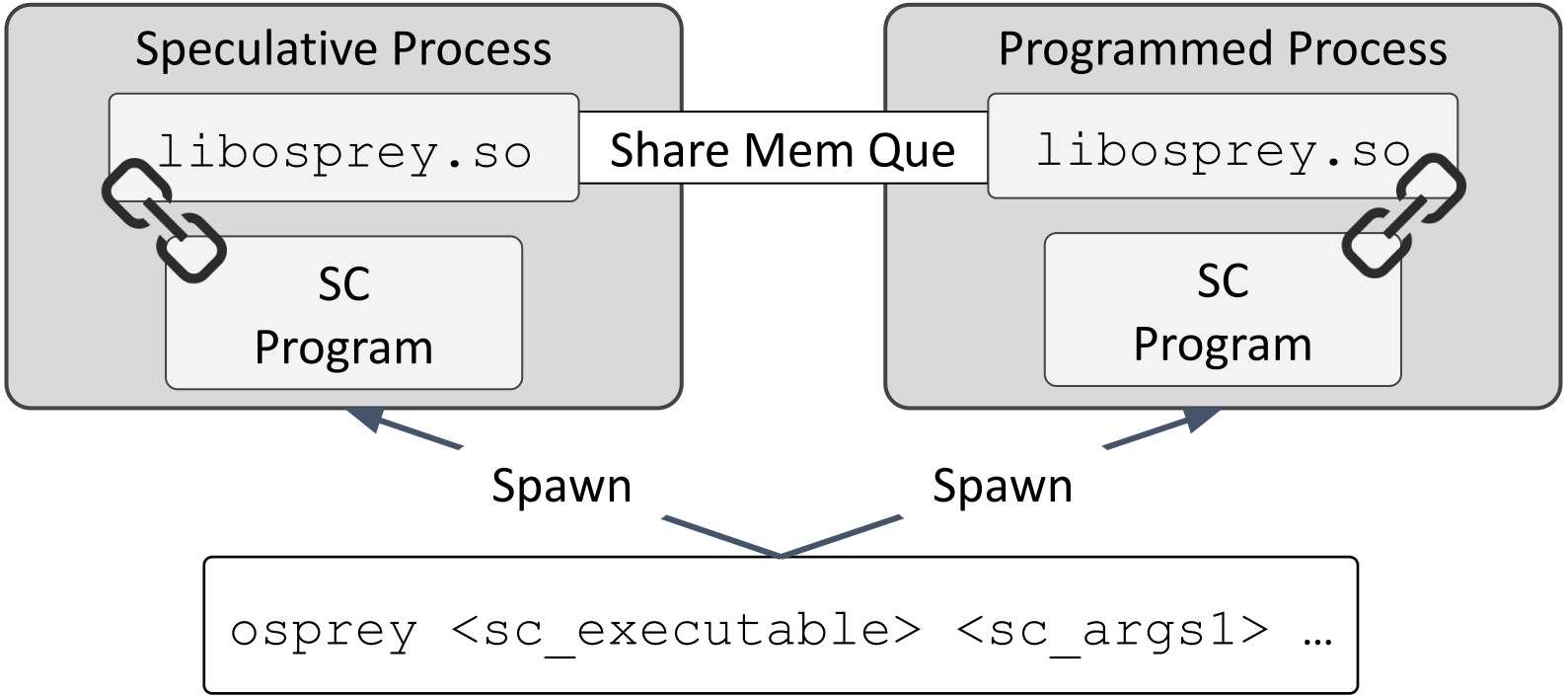


Can we get low develop effort and high efficiency altogether?

Osprey: Oblivious Speculation for PRogramming mEmorY

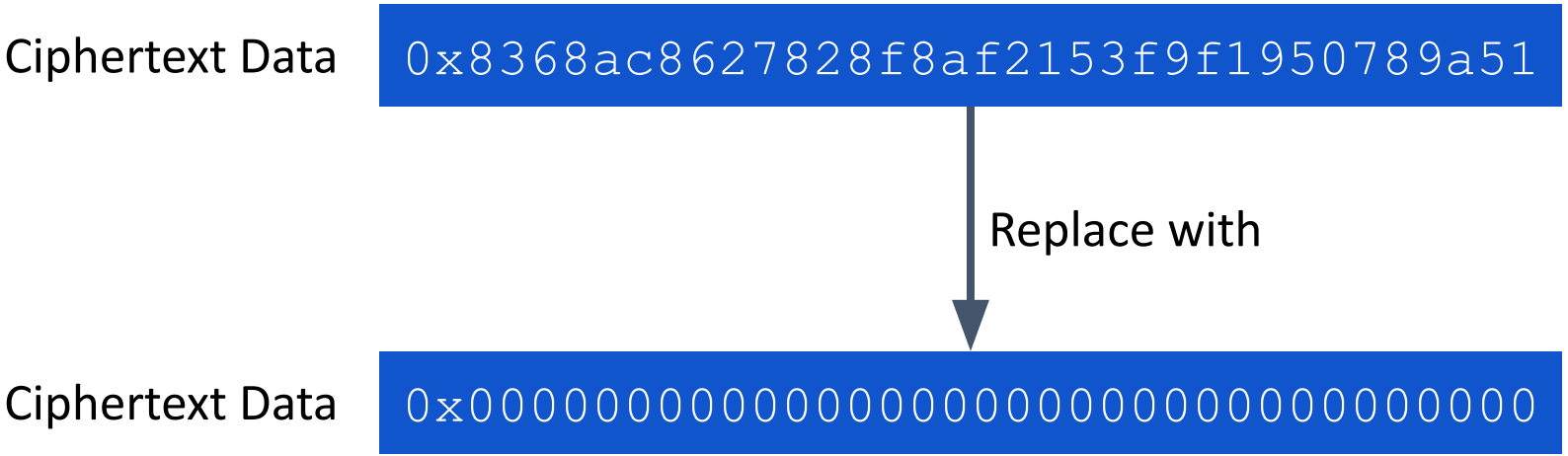


Osprey: Oblivious Speculation for PRogramming mEmorY



Our Insight: Ciphertext Data is Content-Oblivious (CO)

Scramble ciphertext data at the runtime won't change memory accesses

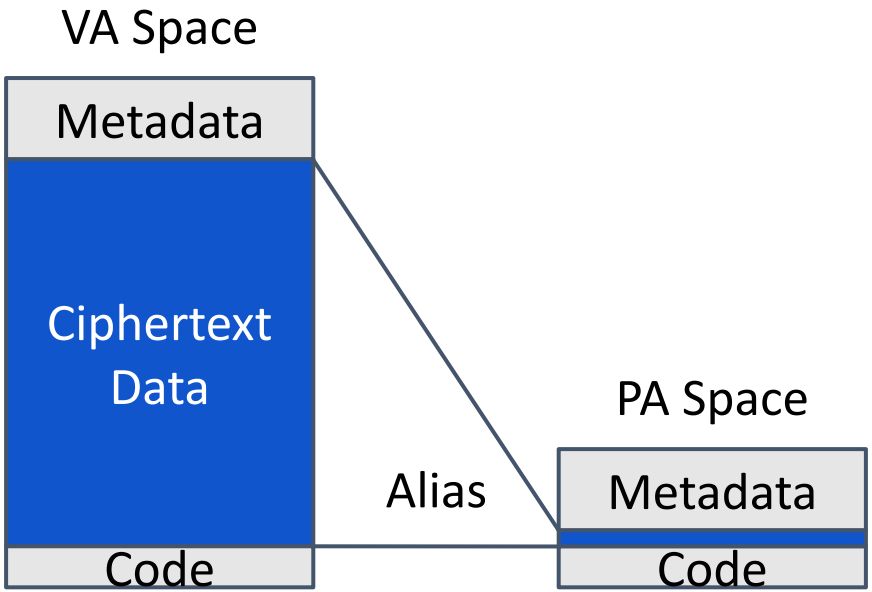


Leverage 1: Don't Store Ciphertext Data

Scramble ciphertext data at the runtime won't change memory accesses

Why do we bother **store** ciphertext data?

--> **Alias** all ciphertext content to a single physical page

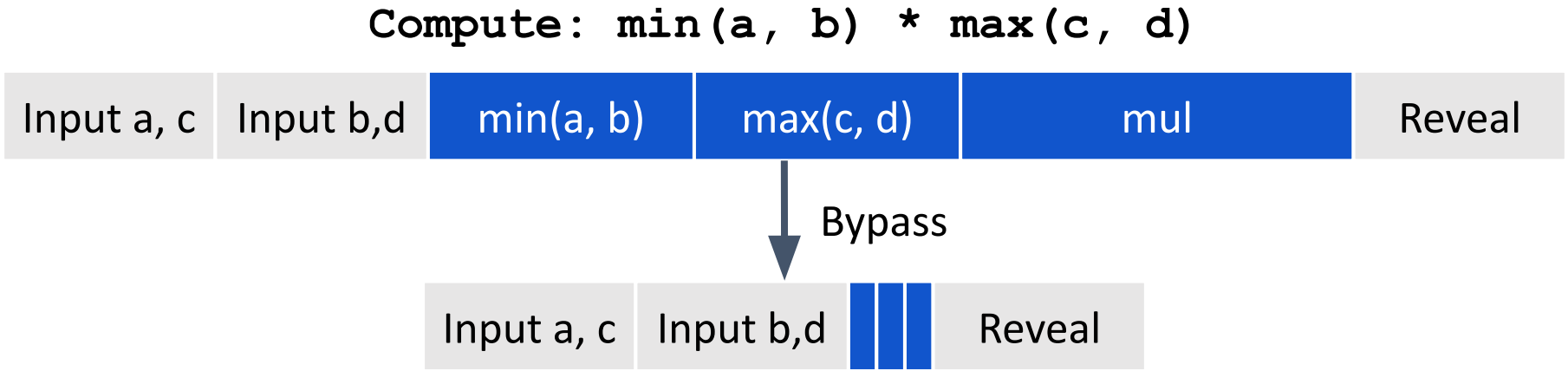


Leverage 2: Don't Compute Ciphertext Data

Scramble ciphertext data at the runtime won't change memory accesses

Why do we bother **compute** ciphertext data?

--> **Bypass** all ciphertext primitives and only record accesses

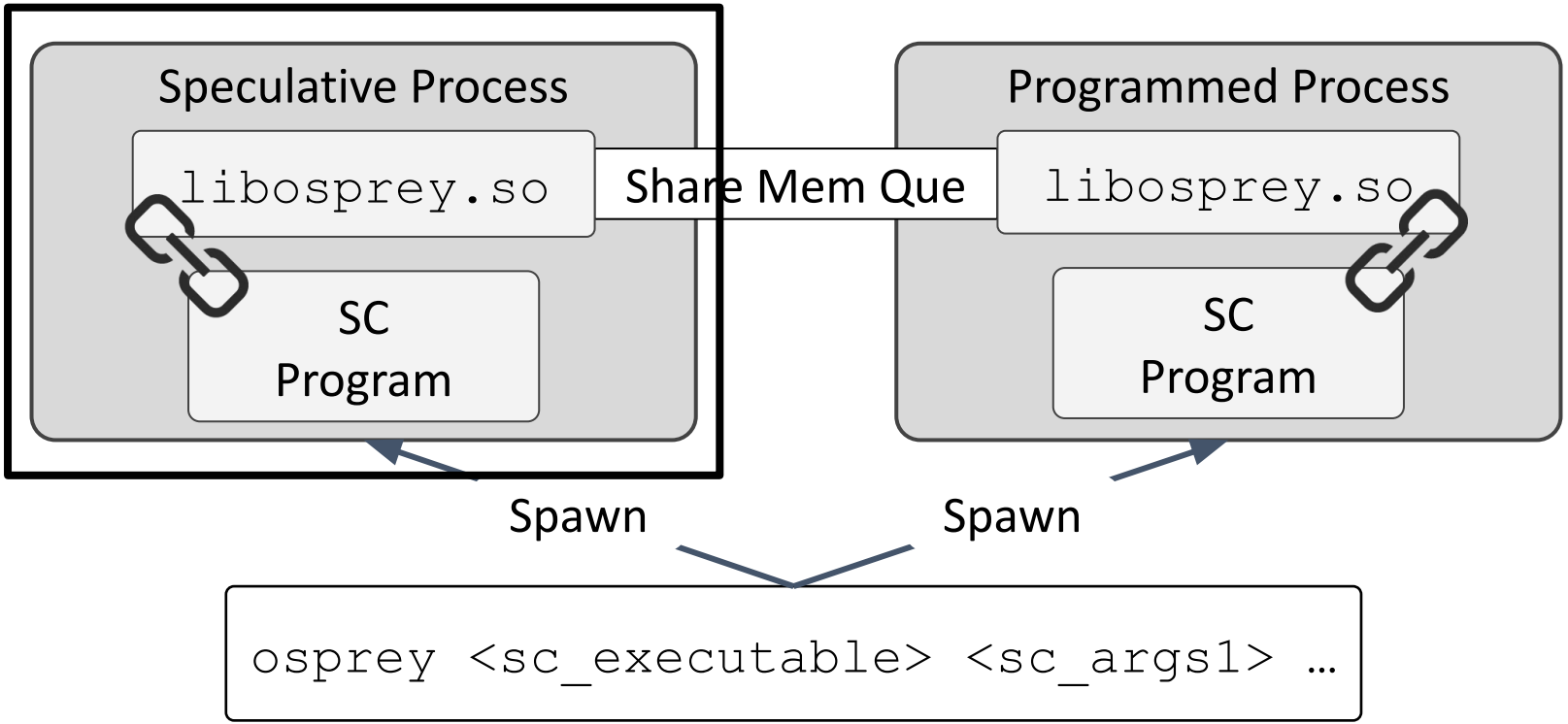


Annotate CO Memory and Crypto Primitives in SC Library

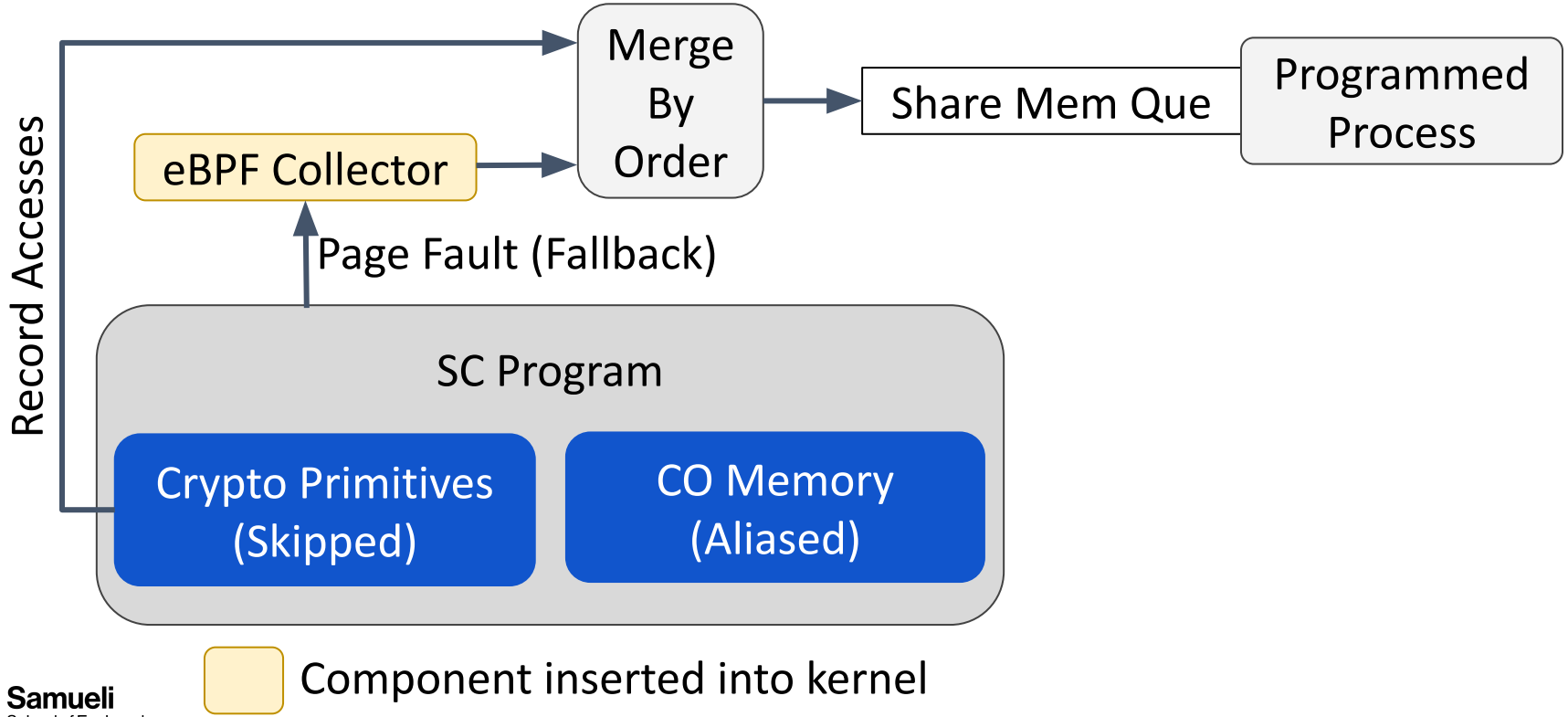
```
// Annotate CO Memory
class Integer {
private:
    std::vector<block, COAllocator> bits;
    ...
}


// Annotate Crypto Operation
Integer add(Integer a, Integer b) {
    Integer res;
    OSPREY_TOUCH(a, b, res);
    ...Slow Execution...
    return res;
}
```

The speculative process becomes **nearly free** after memory and computation overhead removed

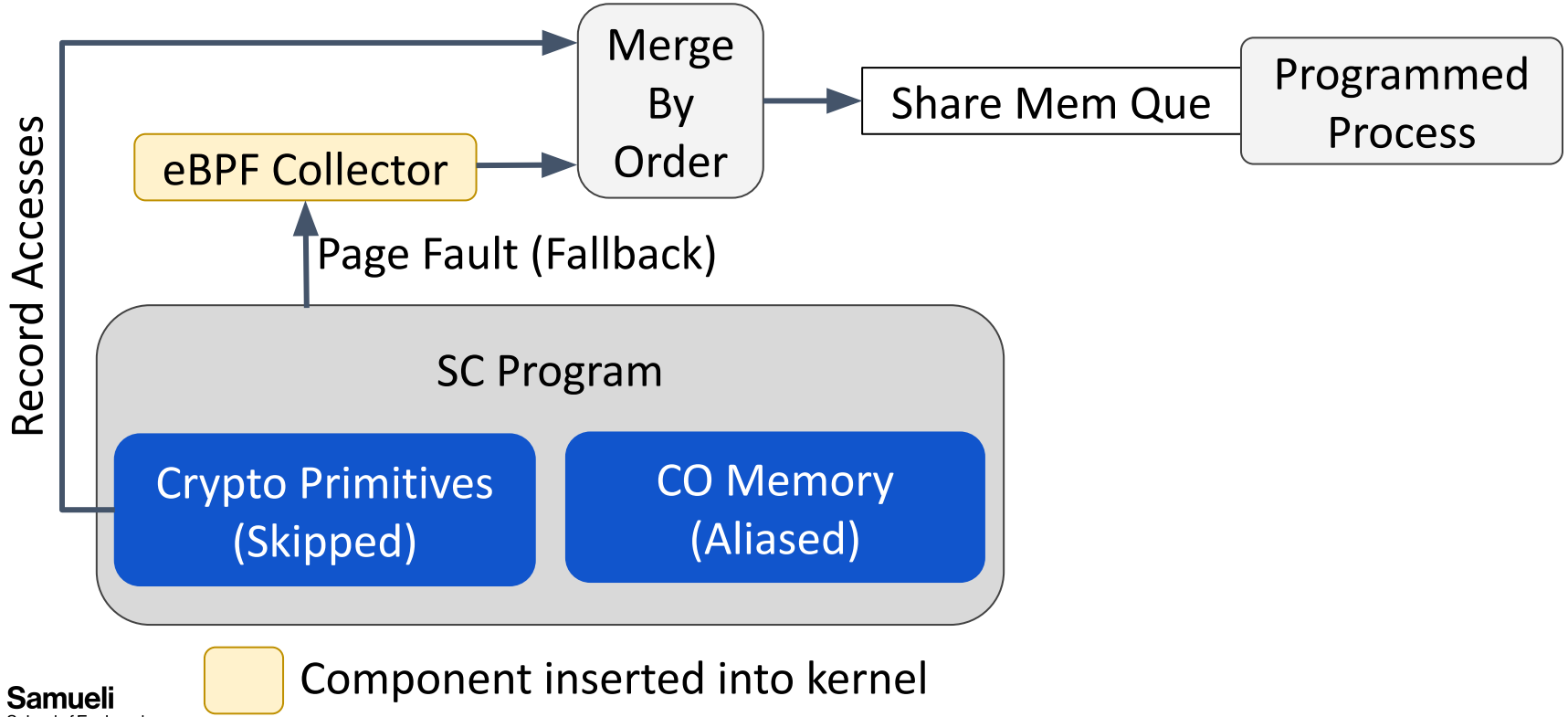
How Does Speculative Process Work?



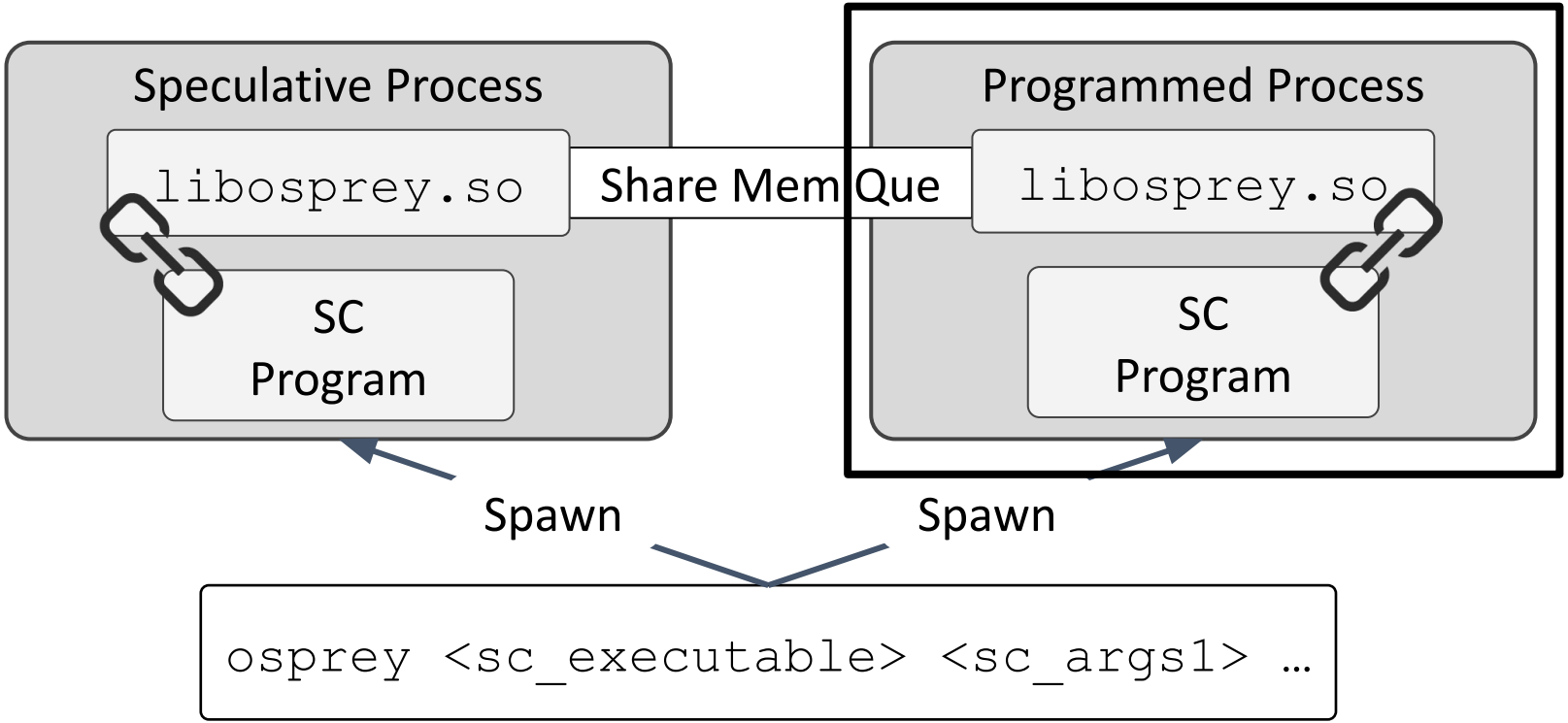
Speculative Process



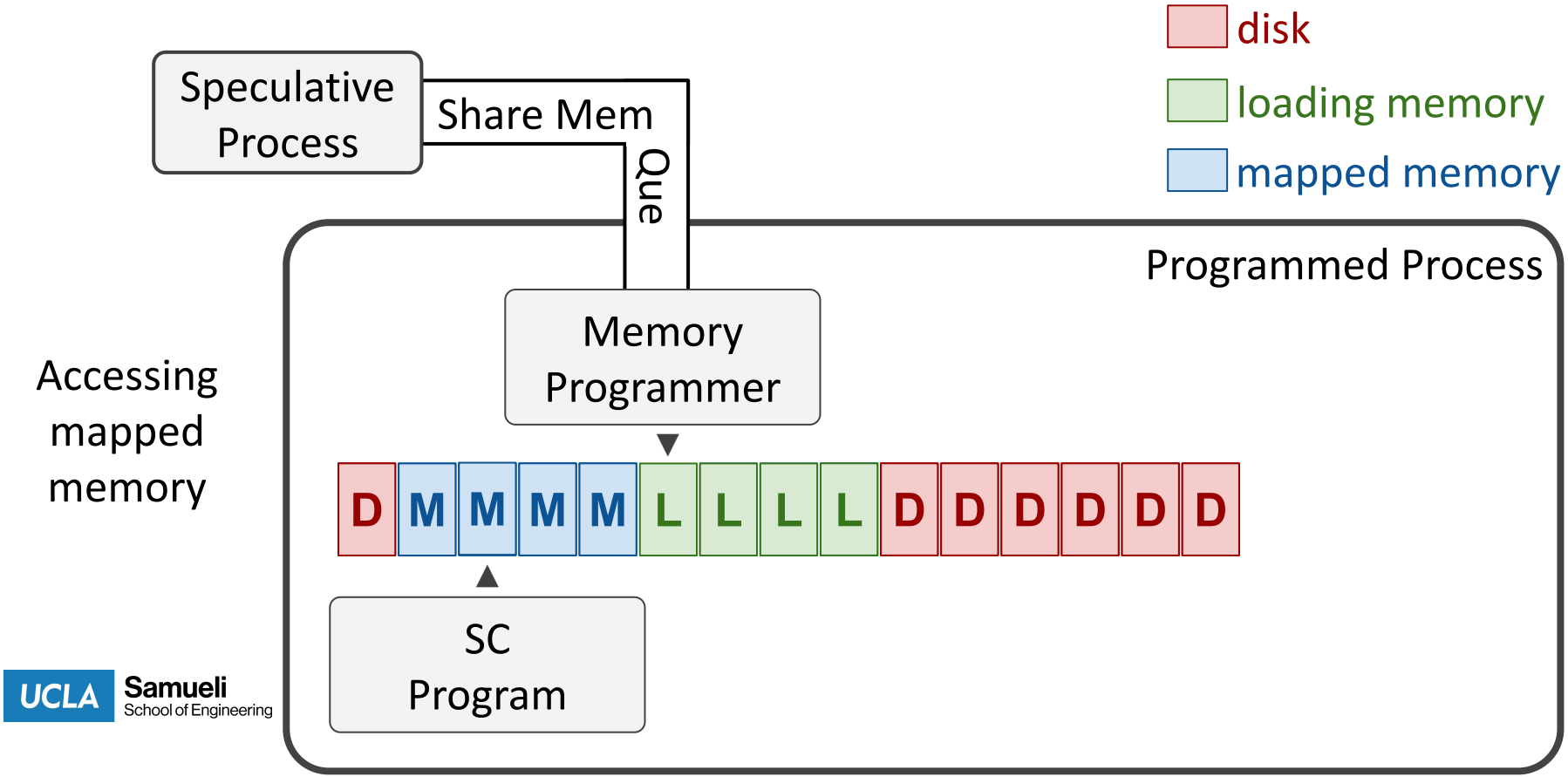
Speculative Process



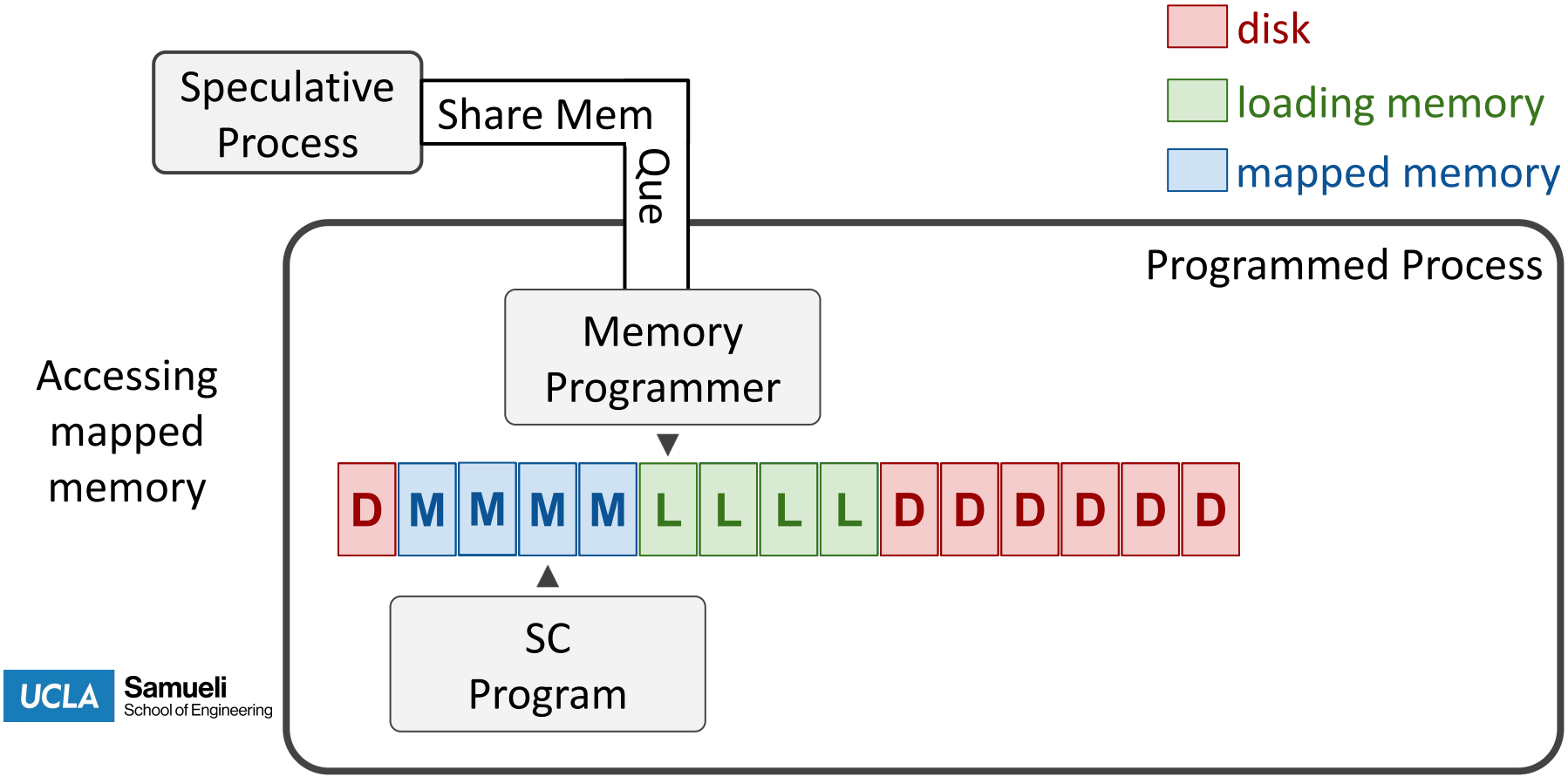
How Does Programmed Process Work?



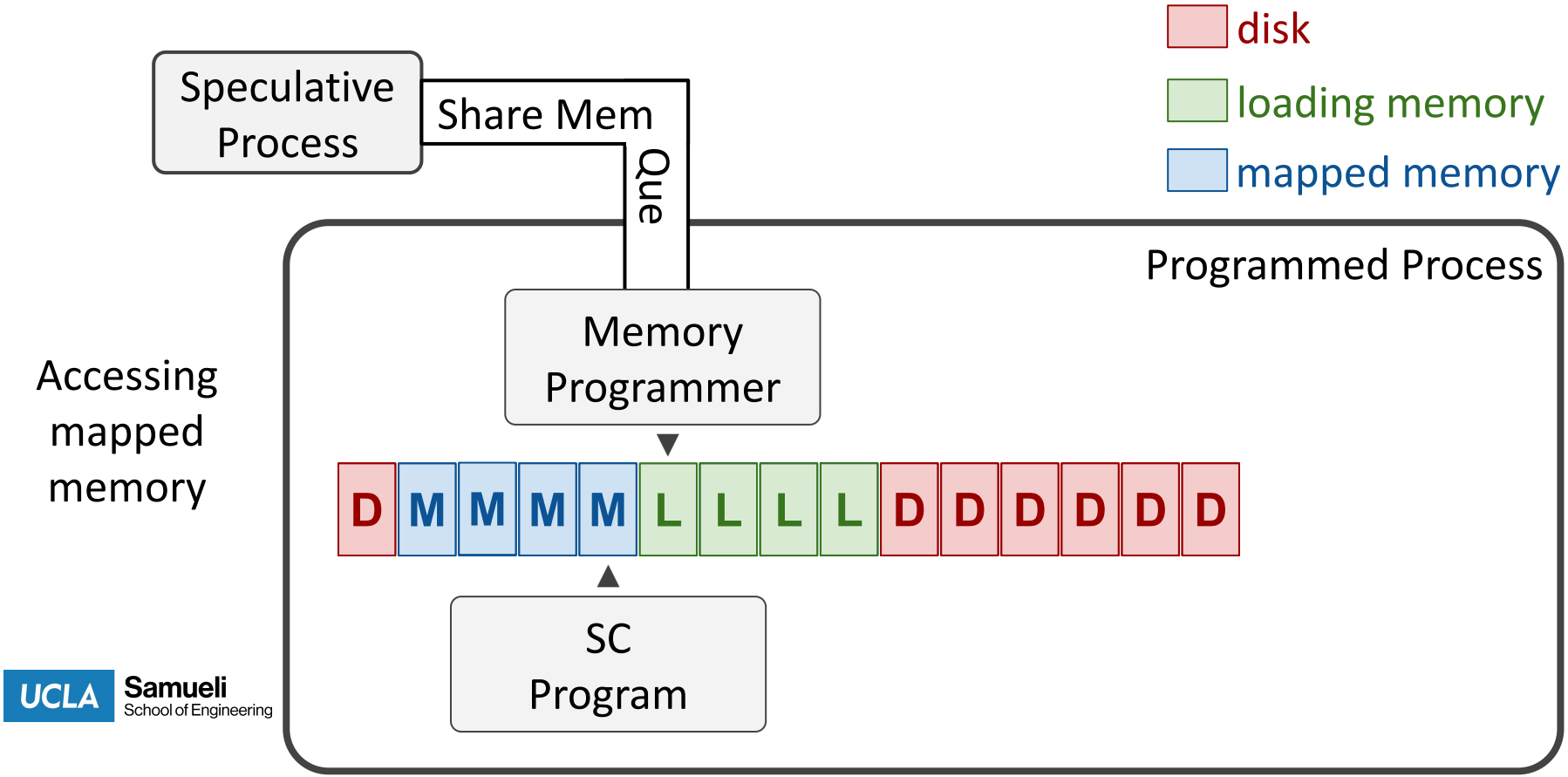
Programmed Process



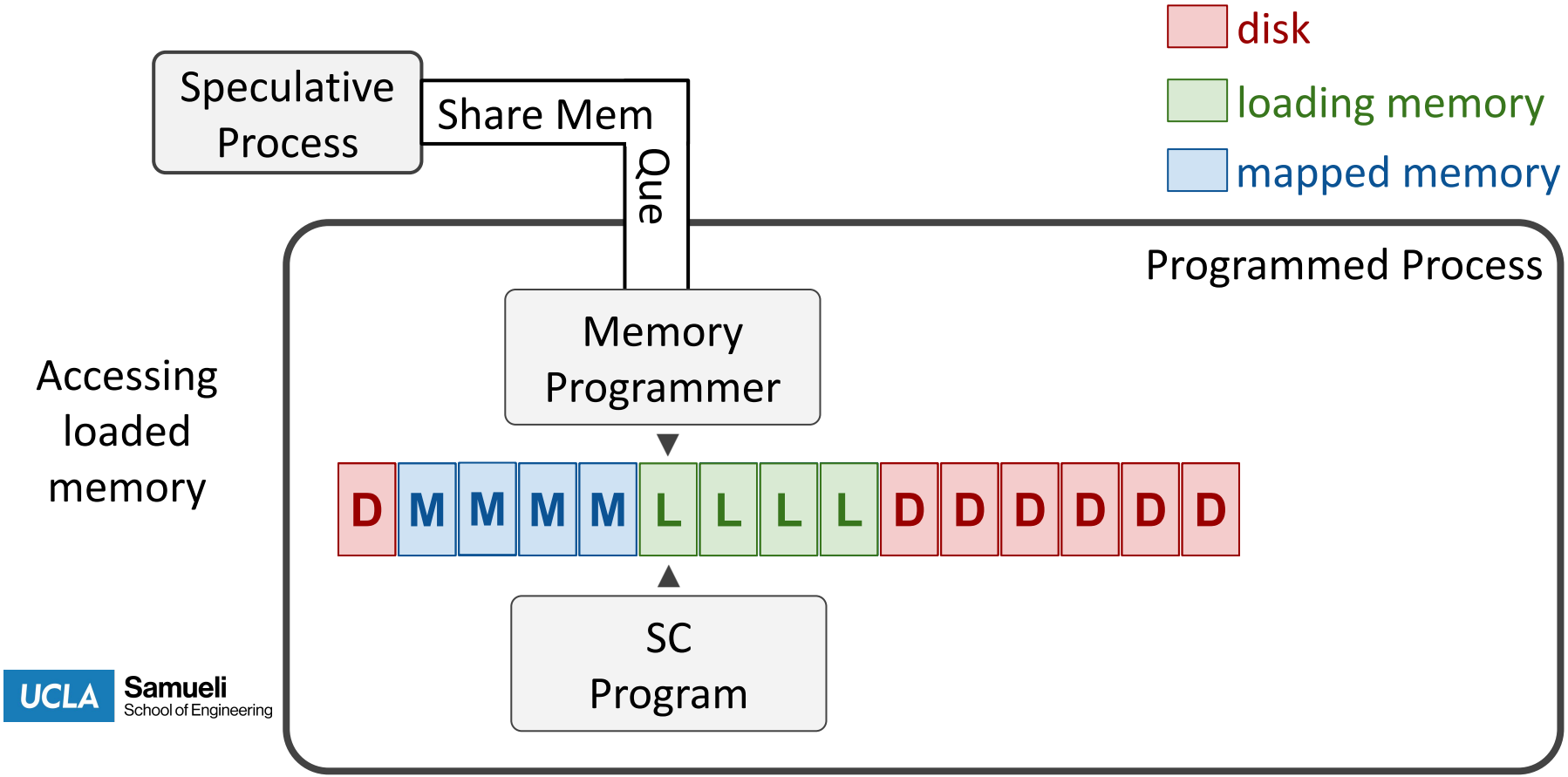
Programmed Process



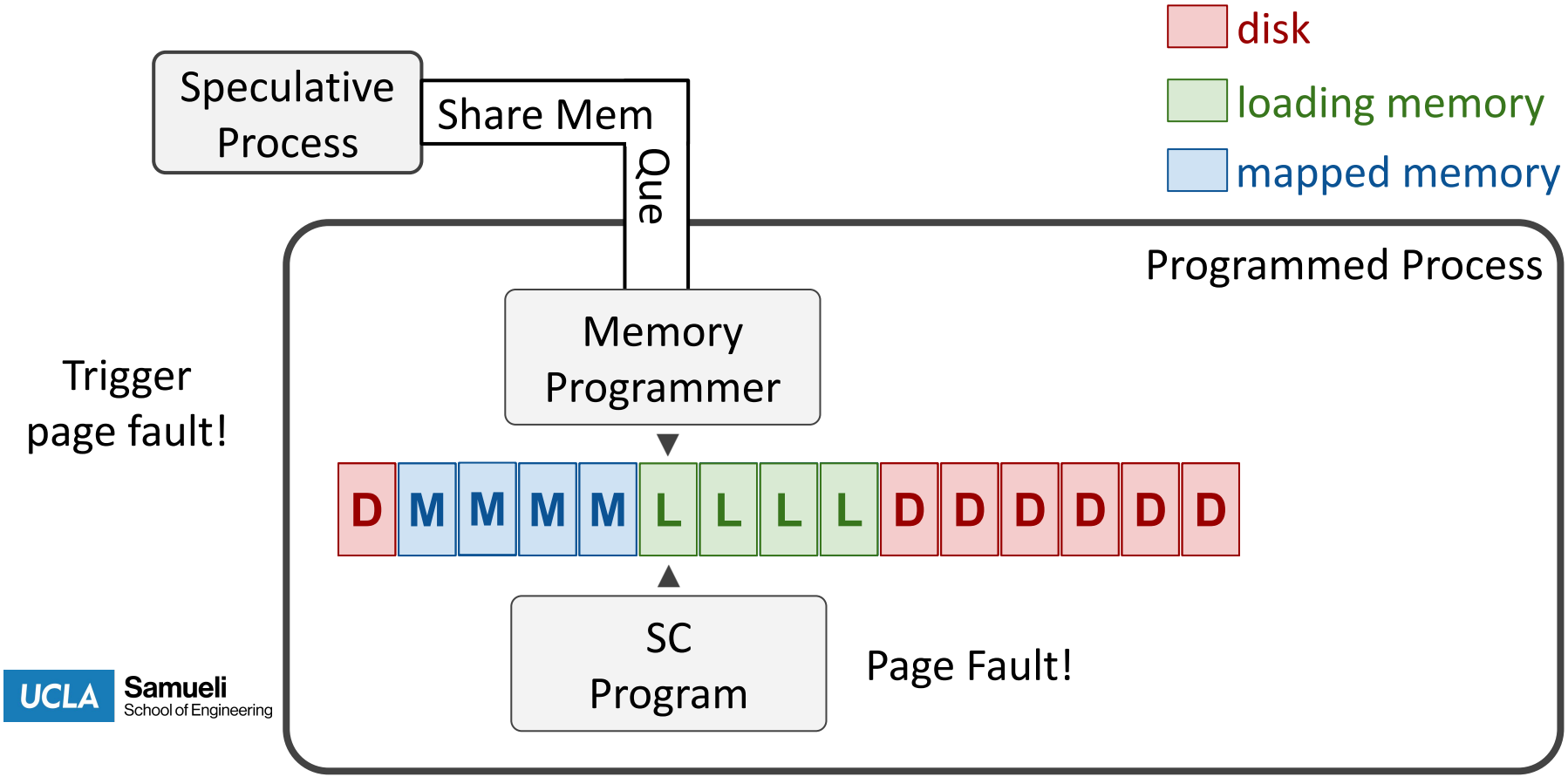
Programmed Process



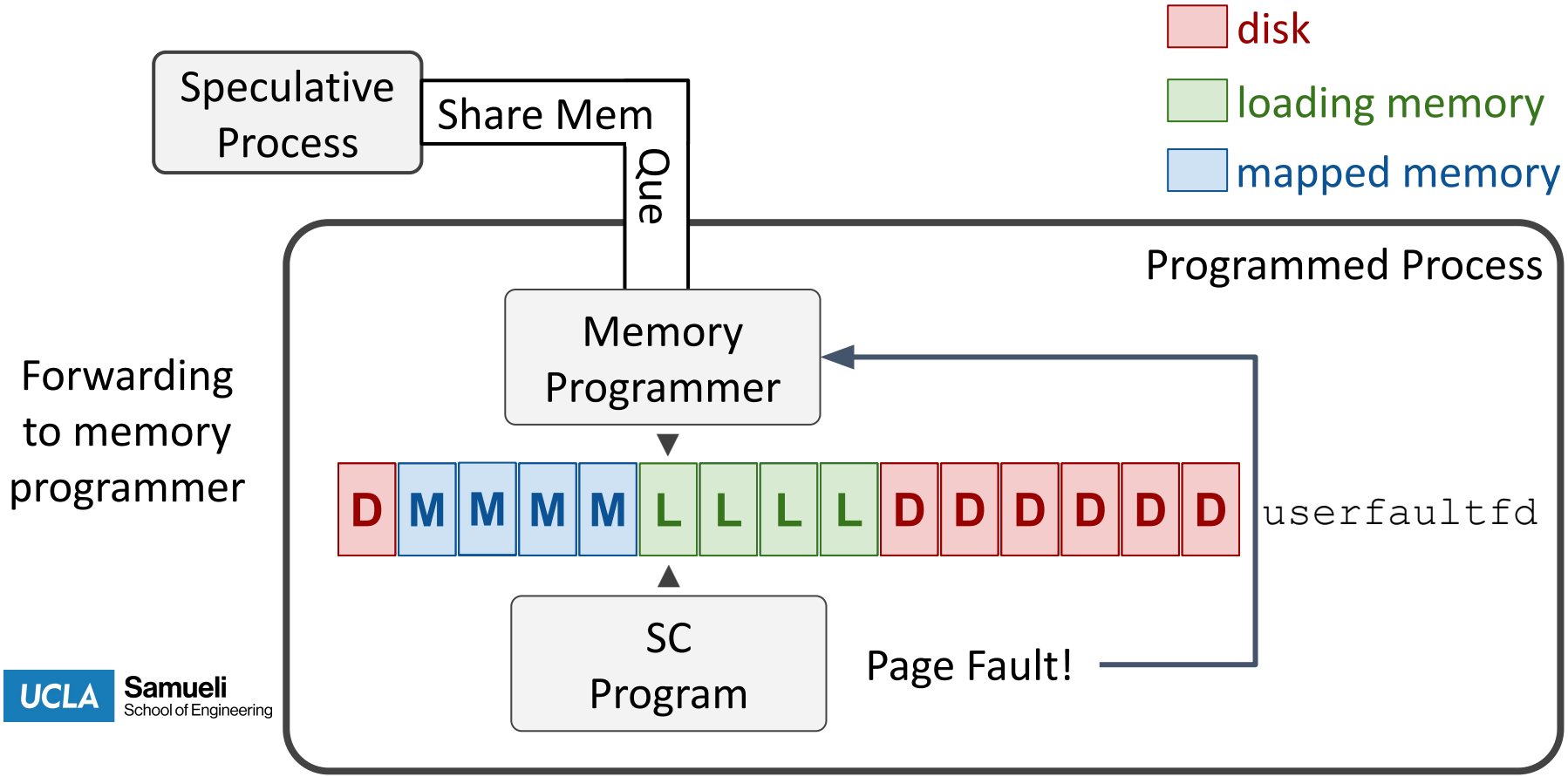
Programmed Process



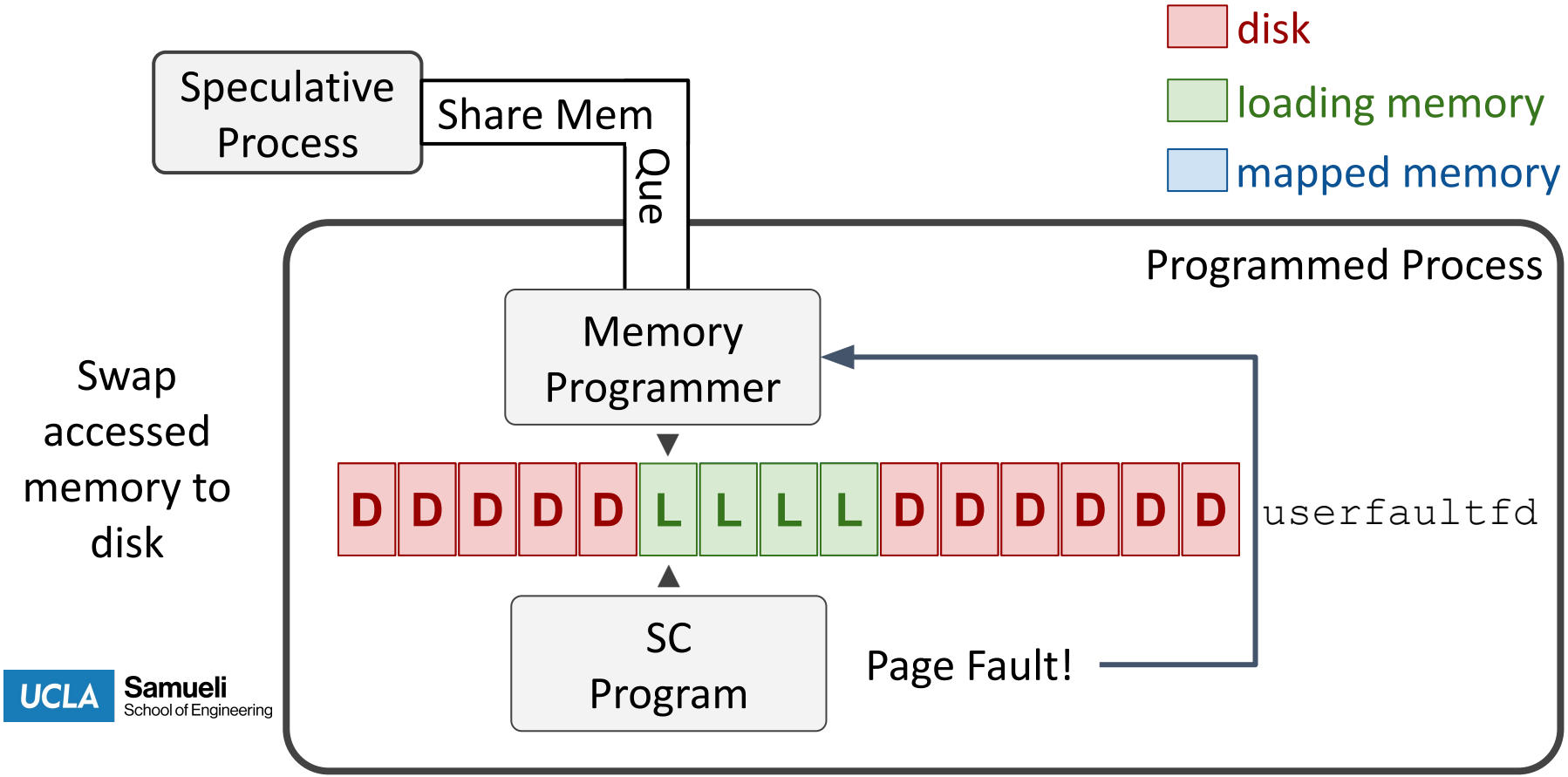
Programmed Process



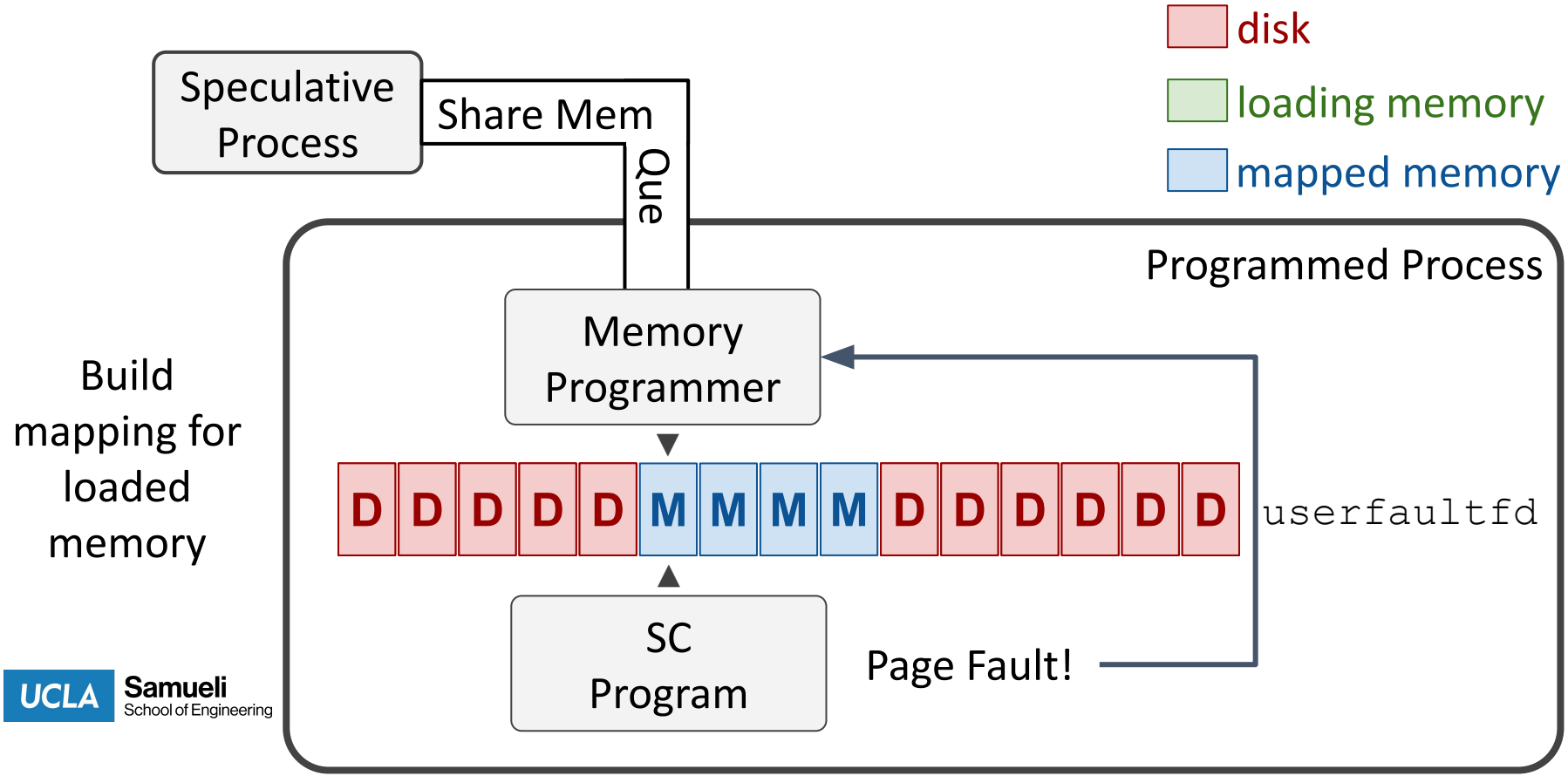
Programmed Process



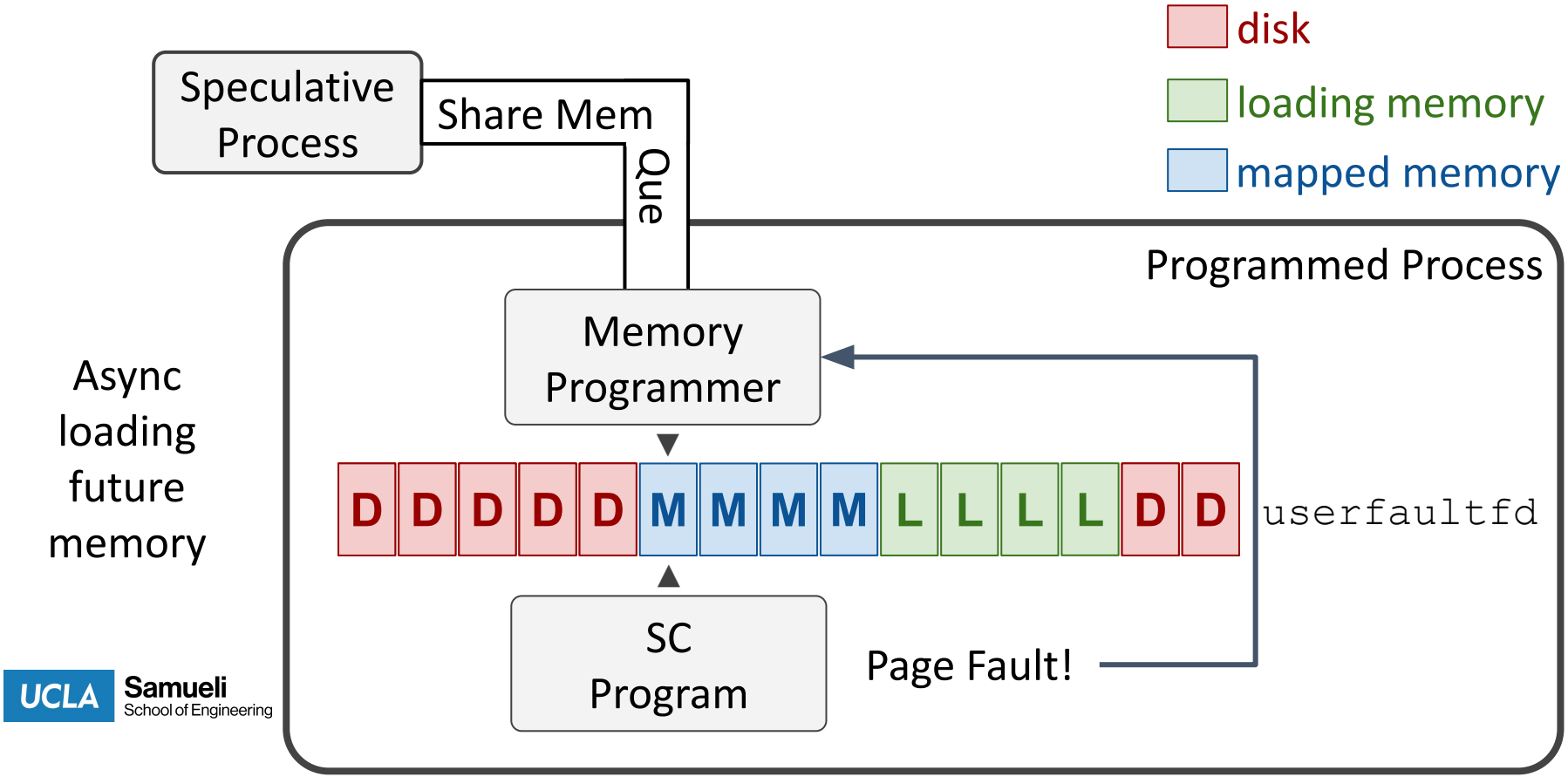
Programmed Process



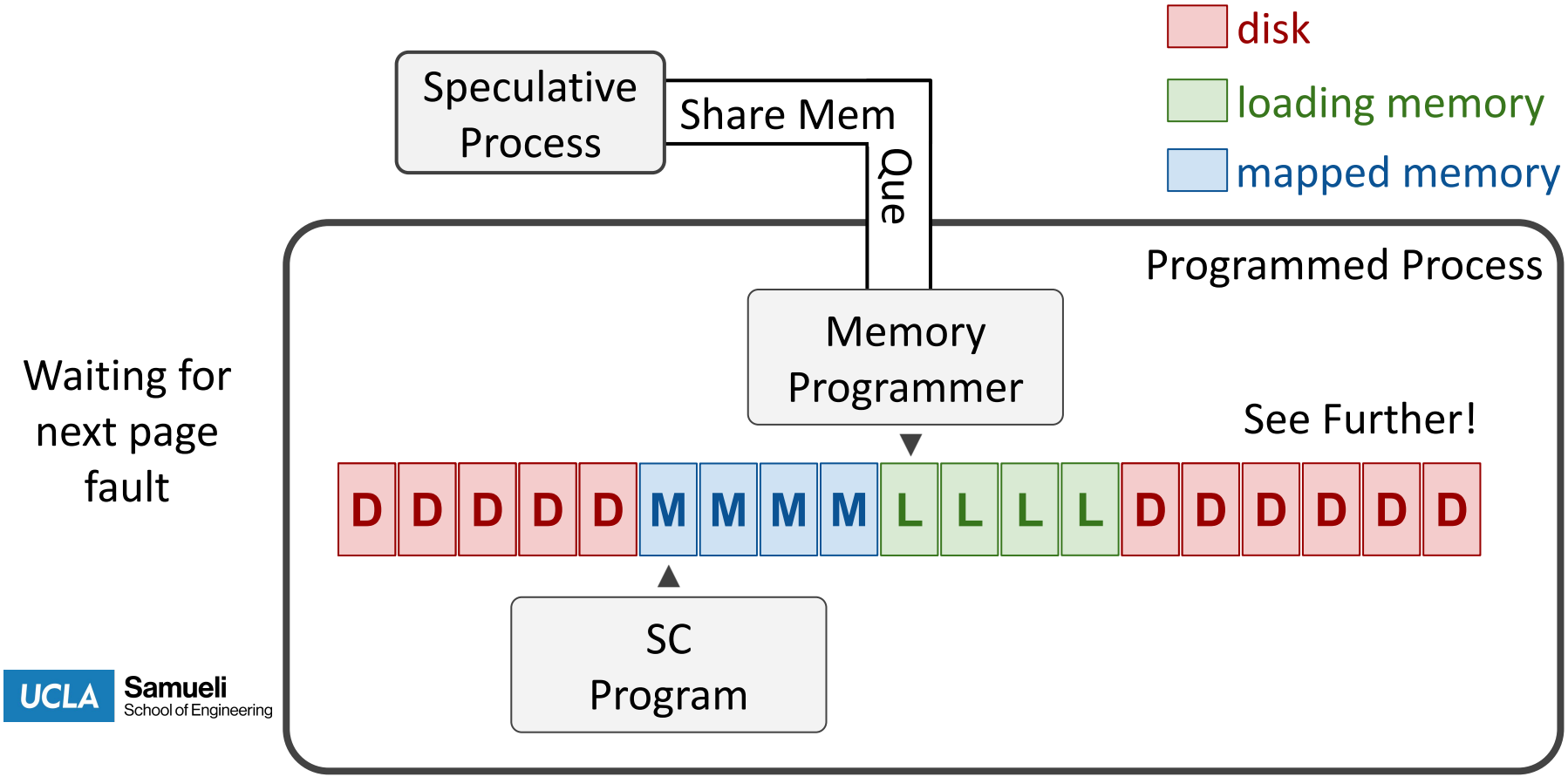
Programmed Process



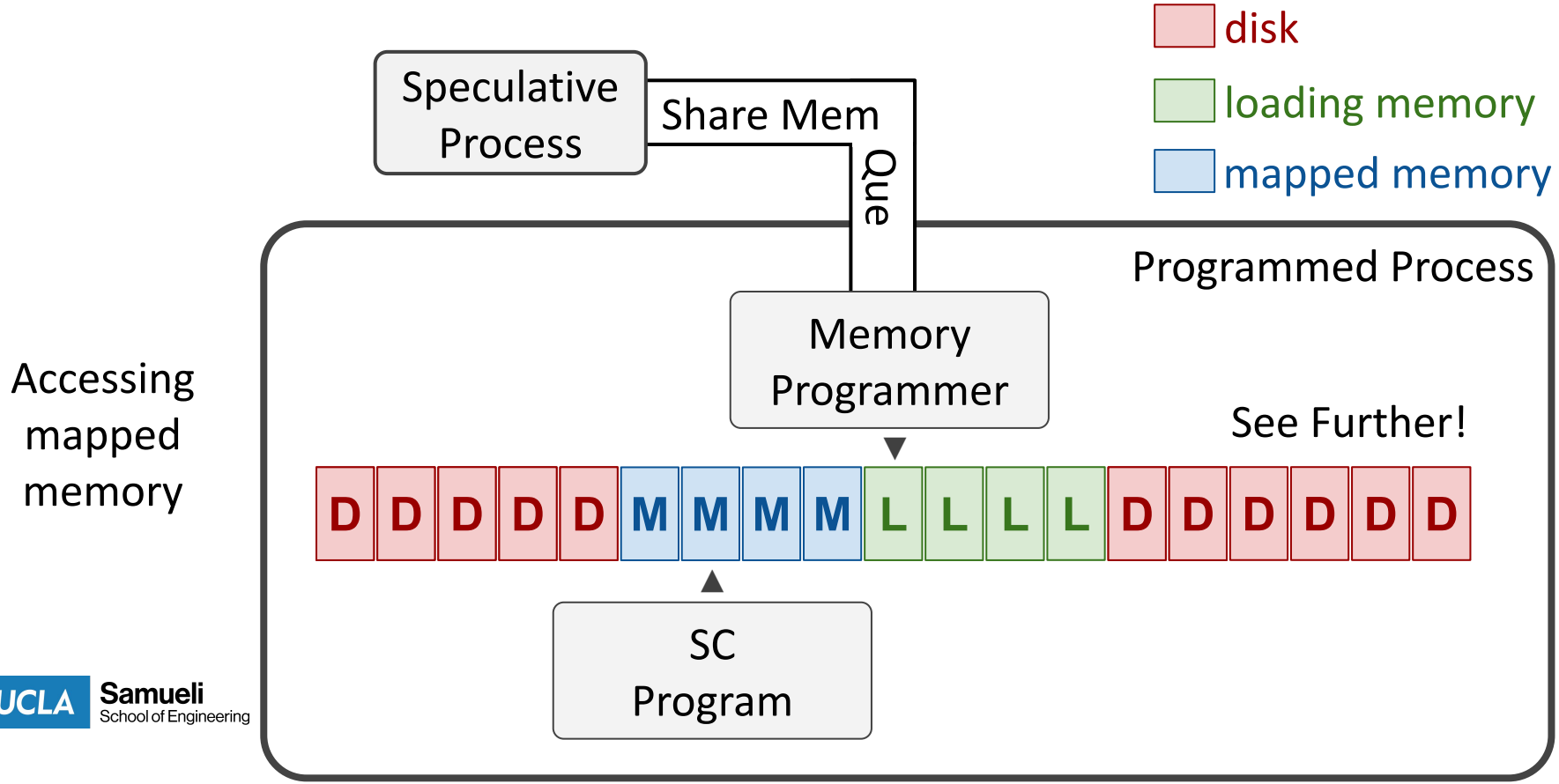
Programmed Process



Programmed Process



Programmed Process



Check Out Our Paper To See...

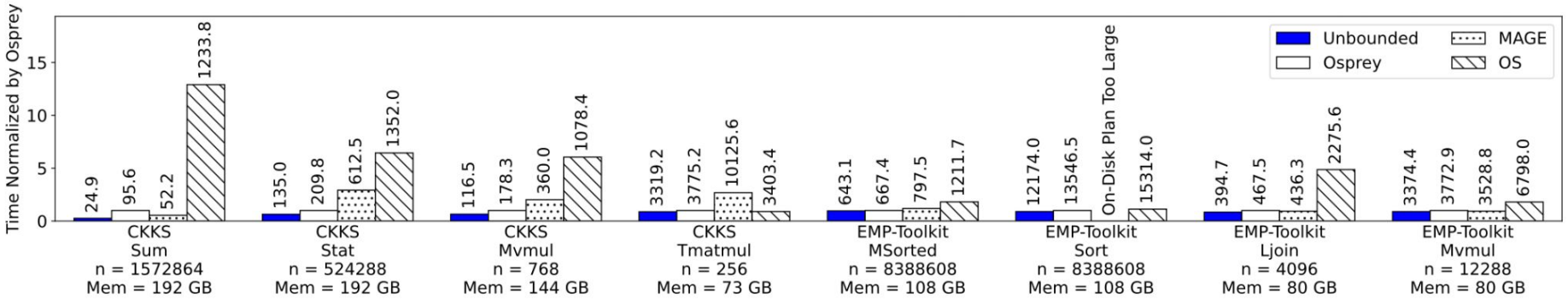
- How do we minimize and generalize Linux kernel modification
- How we use MPK to avoid PF contention in multi-threading

...and more

Evaluation Setup

- Platform
  - CPU: INTEL(R) XEON(R) GOLD 5520+ (Max 4.0 Ghz)
  - SSD: Micron 7450 PRO NVMe (SEQ R/W 6.8/5.6 GB/s)
- Cryptographic Protocols:
  - Garbled Circuits
  - CKKS Homomorphic Encryption
- Baselines:
  - Unbounded: Physical Upper Bound
  - MAGE [OSDI 21]: Policy Upper Bound
  - OS Swapping: Baseline

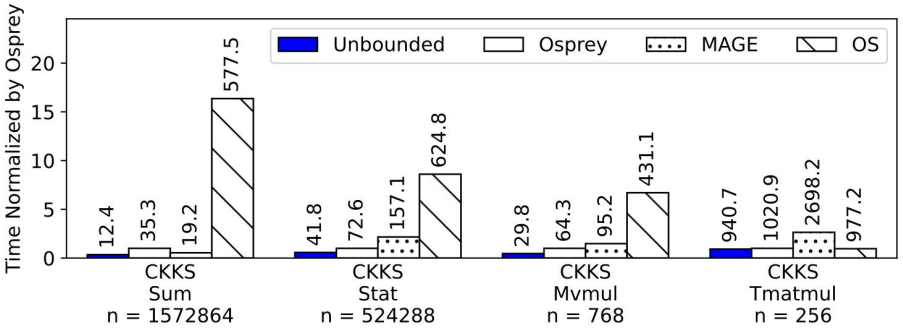
Performance of Workloads in Data Analytics Applications

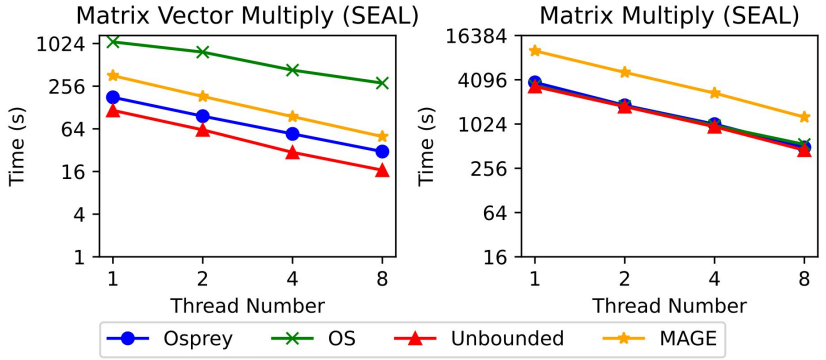


Matches or beats MAGE on 7/8 workloads — without a DSL rewrite

6 of 8 within 60% of unbounded performance

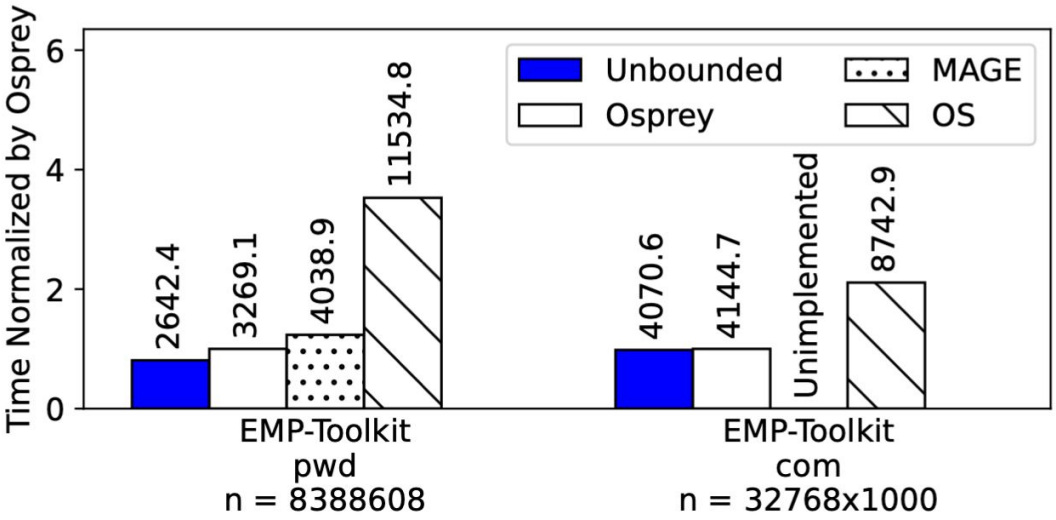
Multithreaded Performance





MPK enables Osprey to scale with thread count

End-to-End Application Performance



pwd: password reuse detection

com: comorbidity analysis

Conclusion

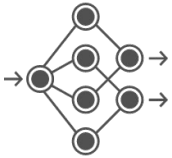
Osprey is a transparent and efficient virtual memory system for secure computation. It:

- Exploits content-obliviousness to make speculation nearly free
- Pre-fetches and evicts pages using a live, streamed access trace
- Runs SC at near in-memory speeds with zero application changes

Osprey's techniques could also potentially benefit:



Oblivious Database



ML Inference

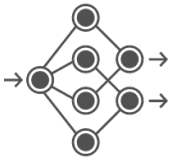
Conclusion

Osprey is a transparent and efficient virtual memory system for secure computation. It:

Osprey's techniques could also potentially benefit:



Oblivious Database



ML Inference

Thank you!

Yicheng Liu
easonliu@ucla.edu

Github:
https://github.com/mpc-systems/osprey-project.git

UCLA Samueli School of Engineering