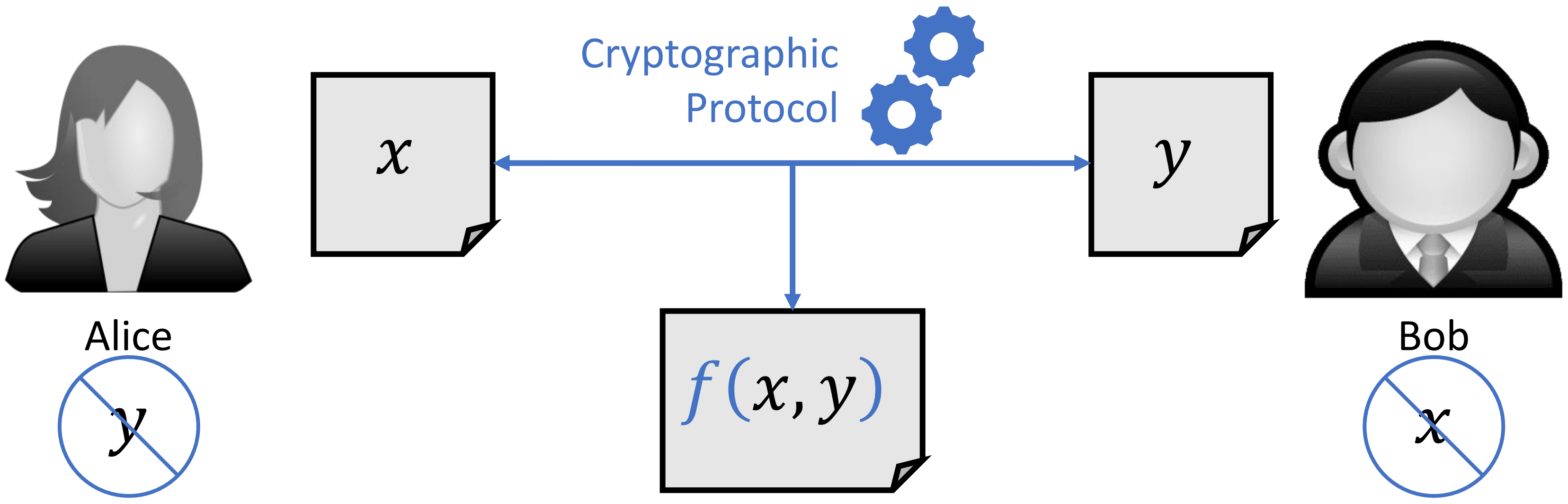
MAGE: Nearly Zero-Cost Virtual Memory for Secure Computation

Sam Kumar, David E. Culler, Raluca Ada Popa

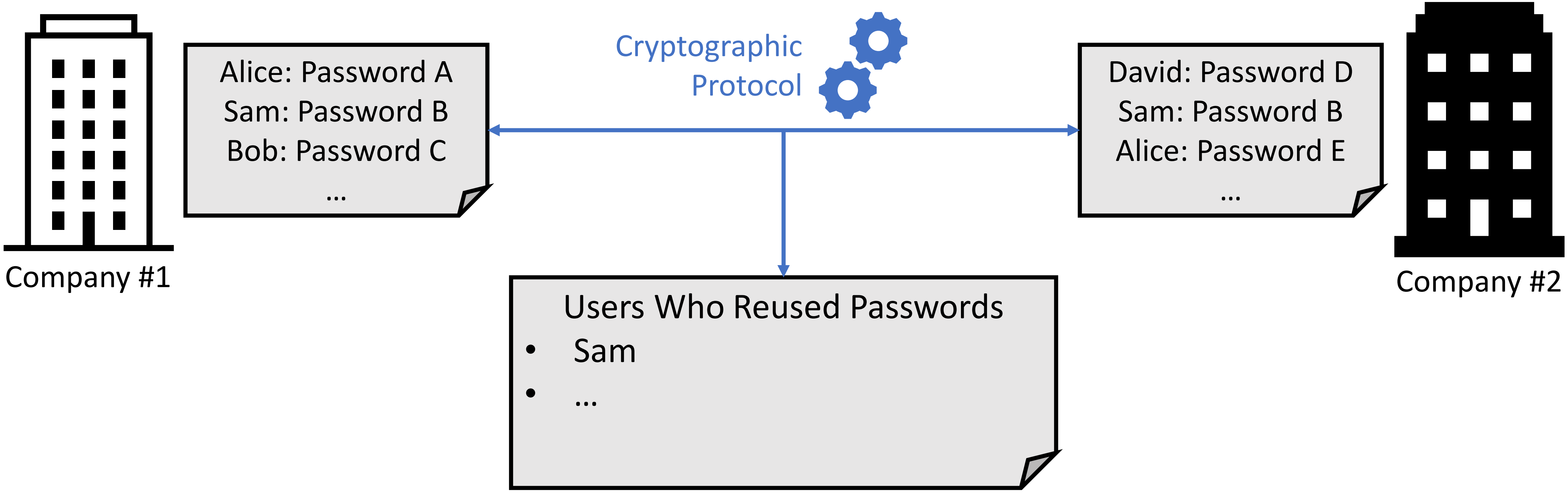
University of California, Berkeley

Secure Computation (SC)

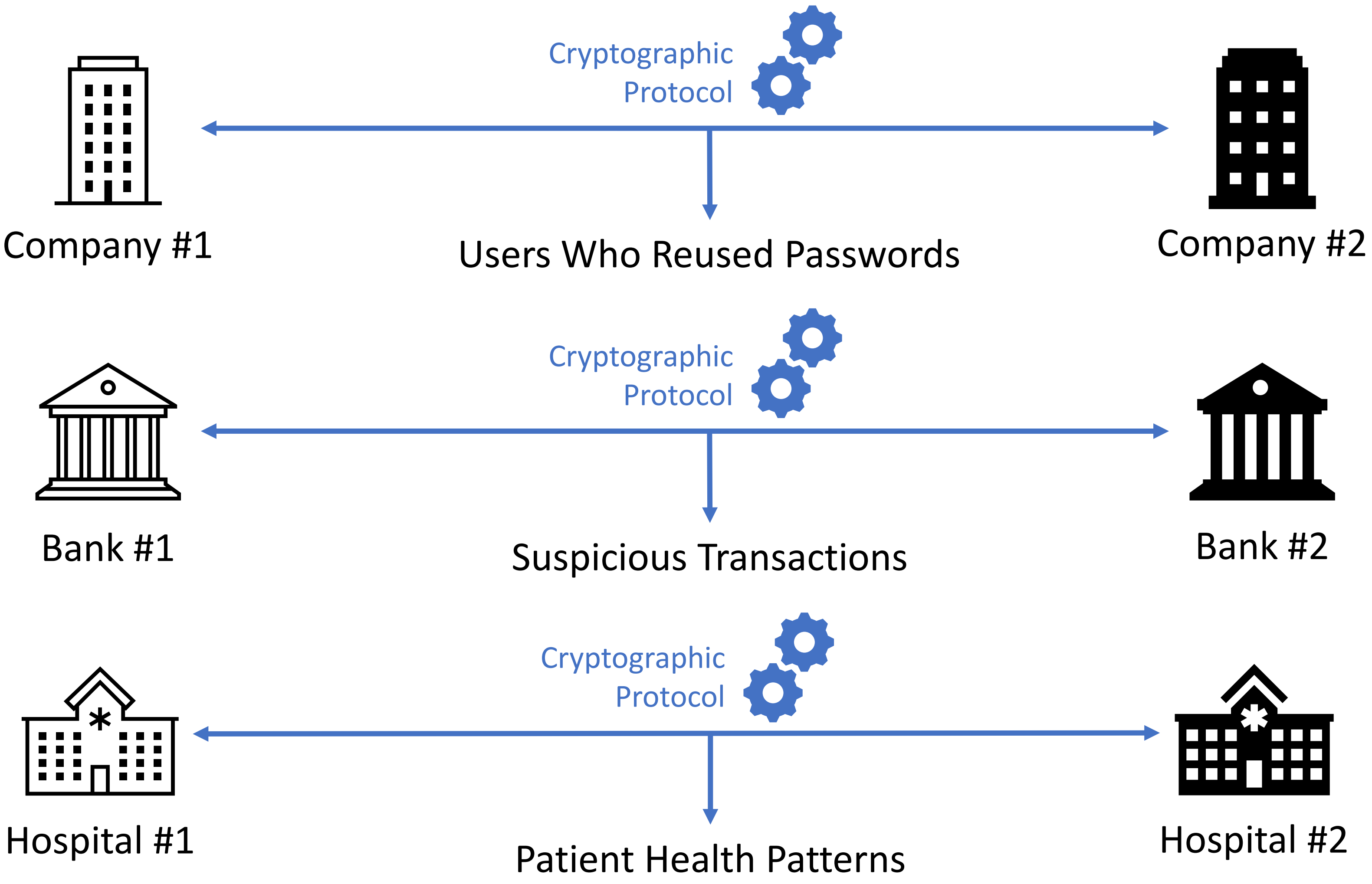


Secure Multi-Party Computation (SMPC) **[Yao86, GMW87]**

Application: Password Reuse Detection [WR19, PKY+21]



Potential SC Applications



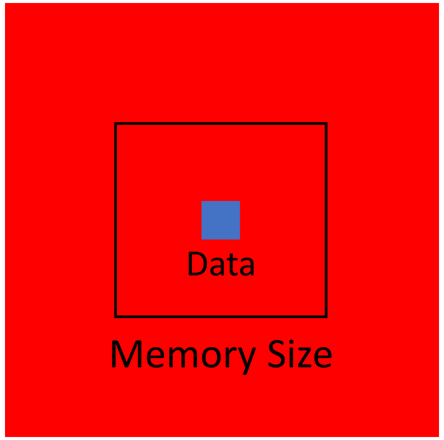
SC has high memory overhead for these applications

Our system, MAGE, addresses the SC memory bottleneck

Memory Overhead of Secure Computation

- SC requires computation on encrypted data (ciphertexts)

- Ciphertexts can be *much* larger than plaintexts
  - 128× expansion factor for garbled circuits (type of SMPC)



Encrypted Data

Memory Can Be a Bottleneck for SC

- "[SMPC] in practice only scales to a few thousand input records" [VSG+19]
  - Existing framework cannot even join *30,000 records* due to memory size
- 16-party set intersection (equi-join) scales to *10,000 integers* [PKY+21]
  - "We observed a stark increase in runtime … due to the exhaustion of available memory"

What Makes OS Virtual Memory Slow?

Based on heuristics
(doesn’t always work well)

Reactive procedure
(reacts to page faults)

Our system, **MAGE**, runs SC at nearly the same speed as if the machine had **unbounded physical memory**.

Key Observation: SC Programs are *Oblivious*

Normal Program (Unsuitable for SC)

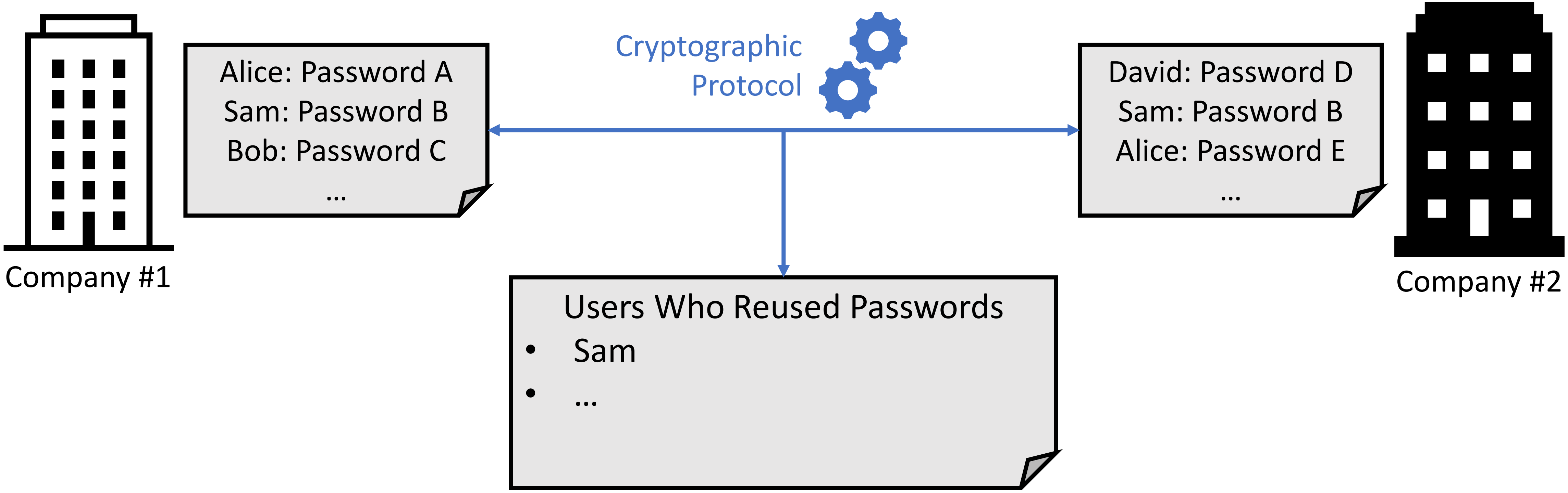
```
uint a = input("alice");
uint b = input("bob");
uint c;
if (a < b) {
    c = a;
} else {
    c = b;
}
```

Oblivious Program (Suitable for SC)

```
uint a = input("alice");
uint b = input("bob");
uint less = a < b;
uint cond = 0 - less;
uint c = ((a ^ b) & cond) ^ b;
```

- This property is **inherent to SC's privacy guarantees.**
- Given an SC program, we can ***pre-compute*** its memory access pattern to pre-plan memory management.

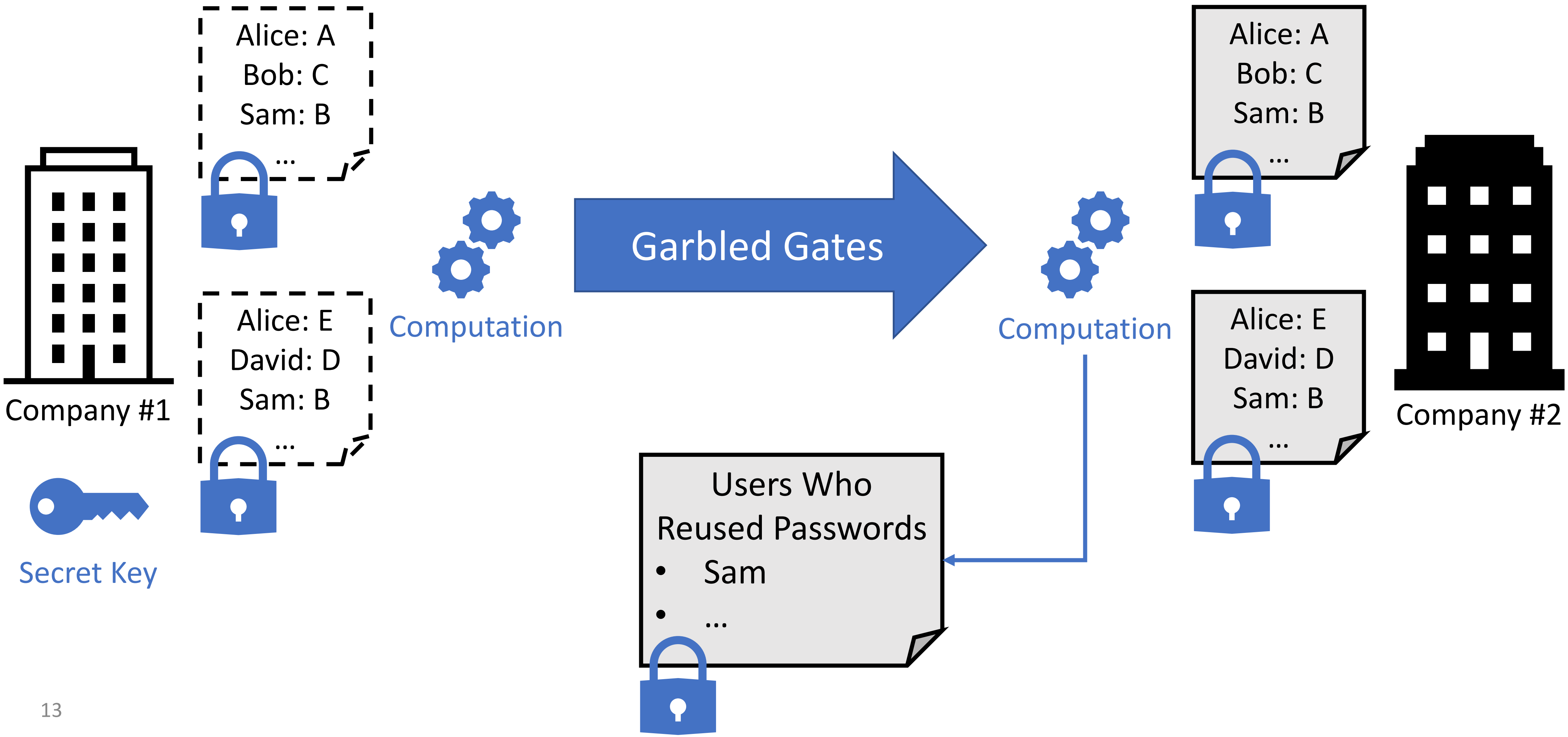
SC Example: Password Reuse Detection



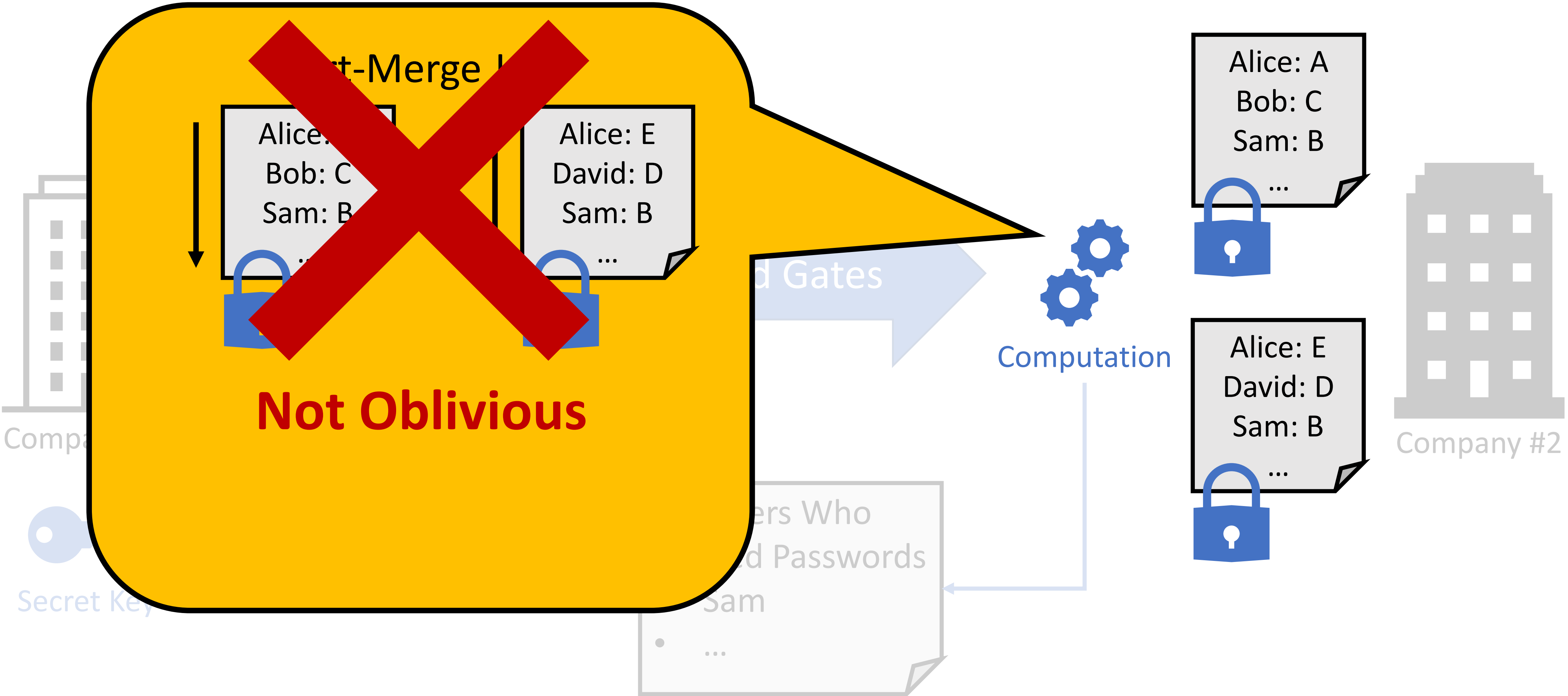
SC Example: Password Reuse Detection

SC Example: Password Reuse Detection

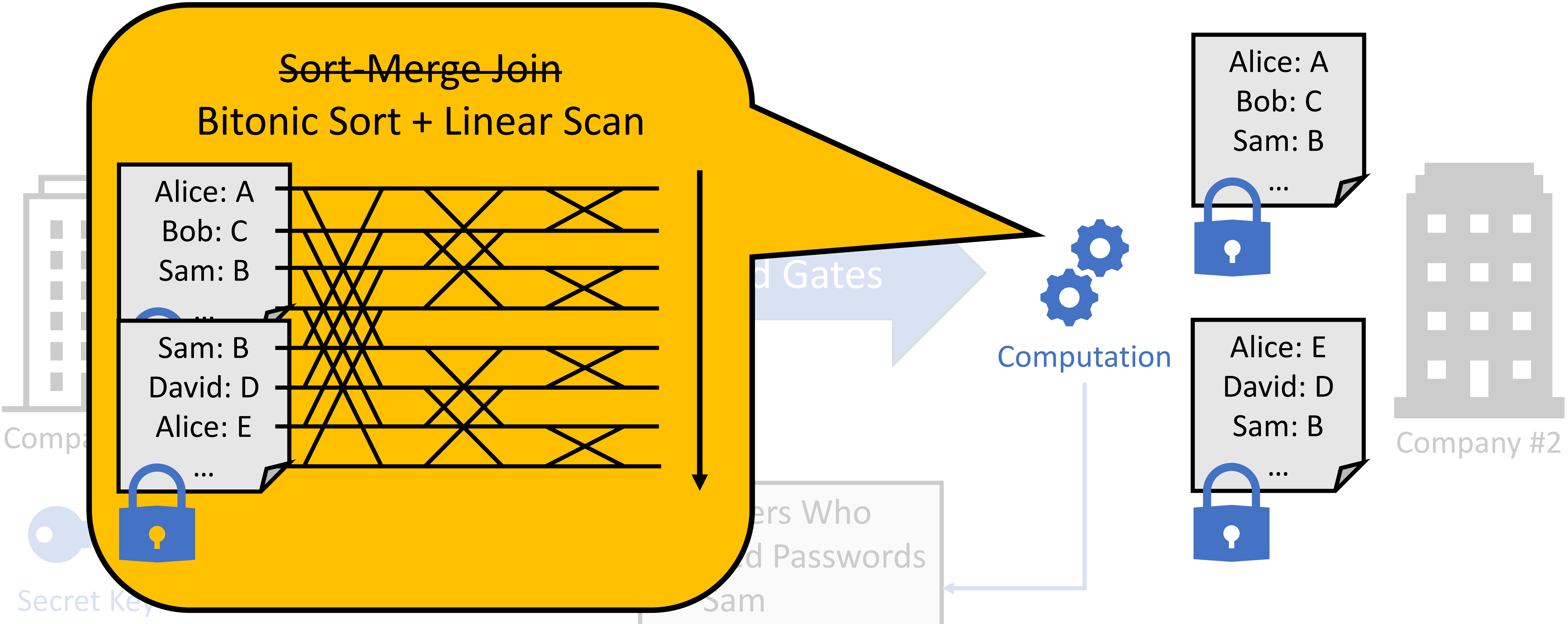
SC Example: Password Reuse Detection



SC Example: Password Reuse Detection

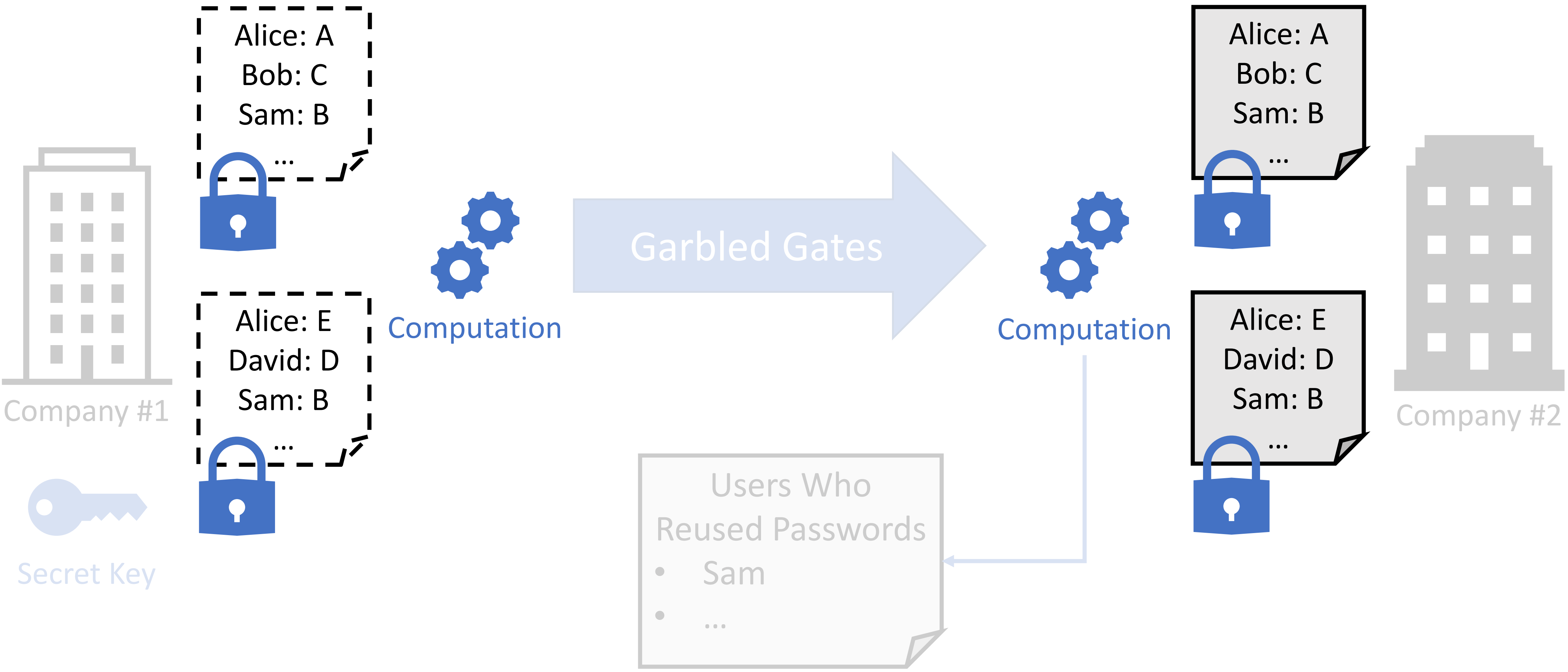


SC Example: Password Reuse Detection

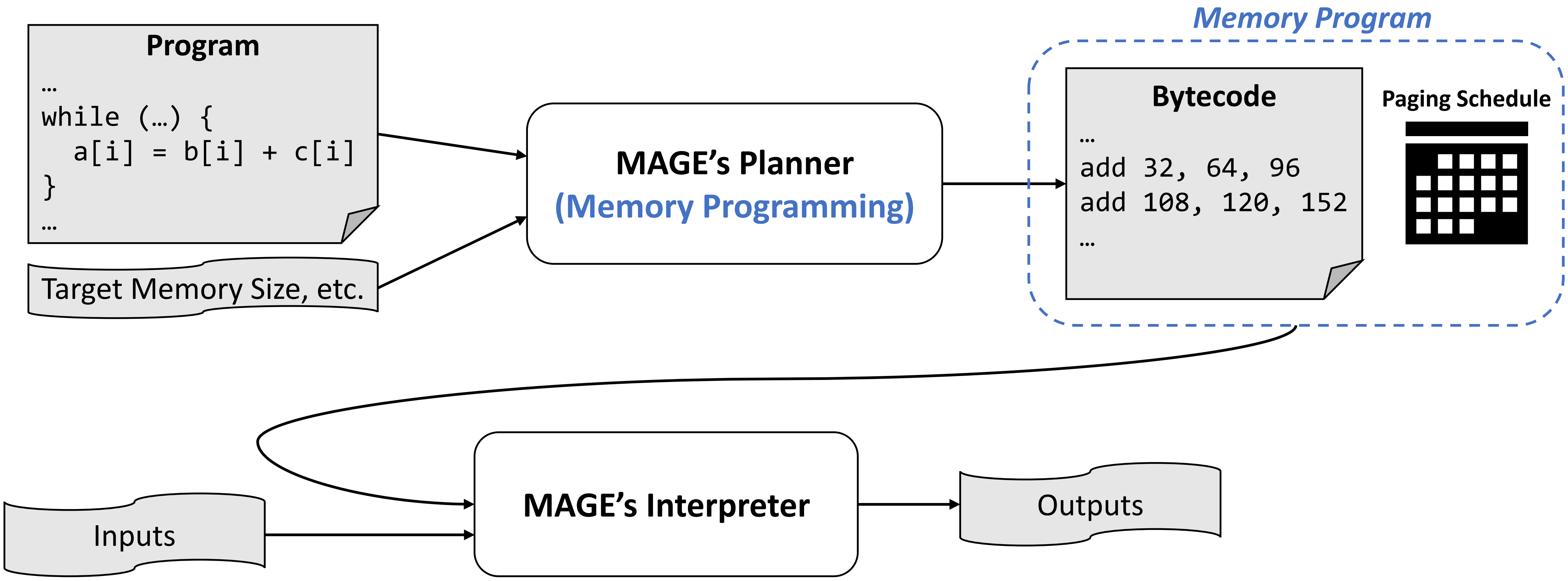


MAGE can predict the access pattern *in advance.*

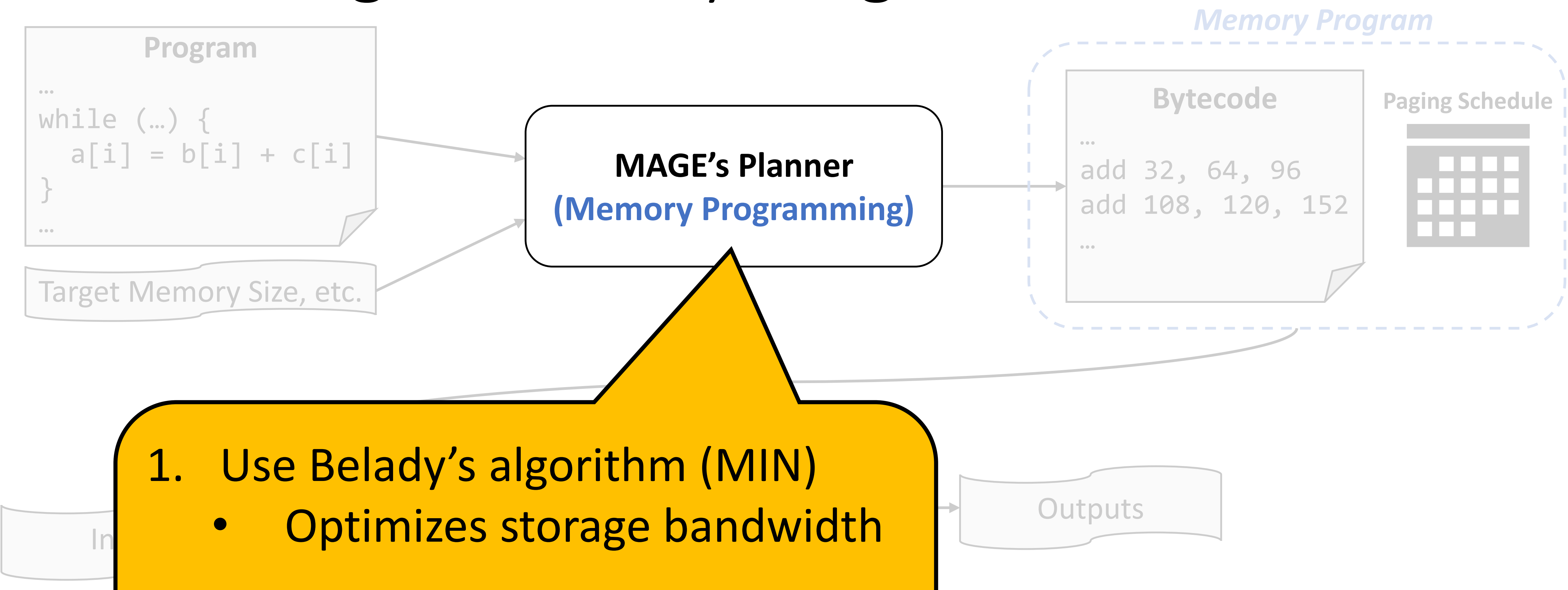
SC Example: Password Reuse Detection



MAGE's Workflow



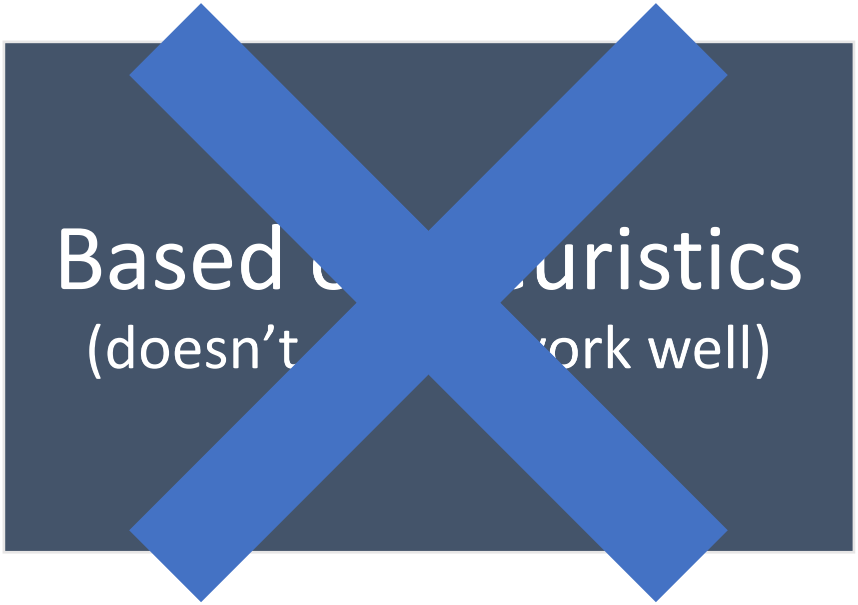
Generating a Memory Program



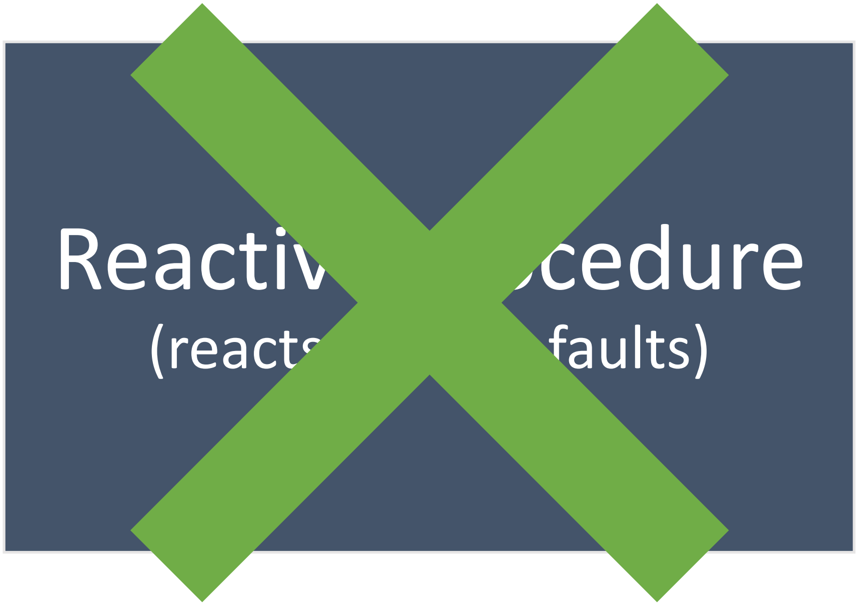
1. Use Belady's algorithm (MIN)
   - Optimizes storage bandwidth

2. Prefetch using the access pattern
   - Masks storage latency

What Makes OS Virtual Memory Slow?

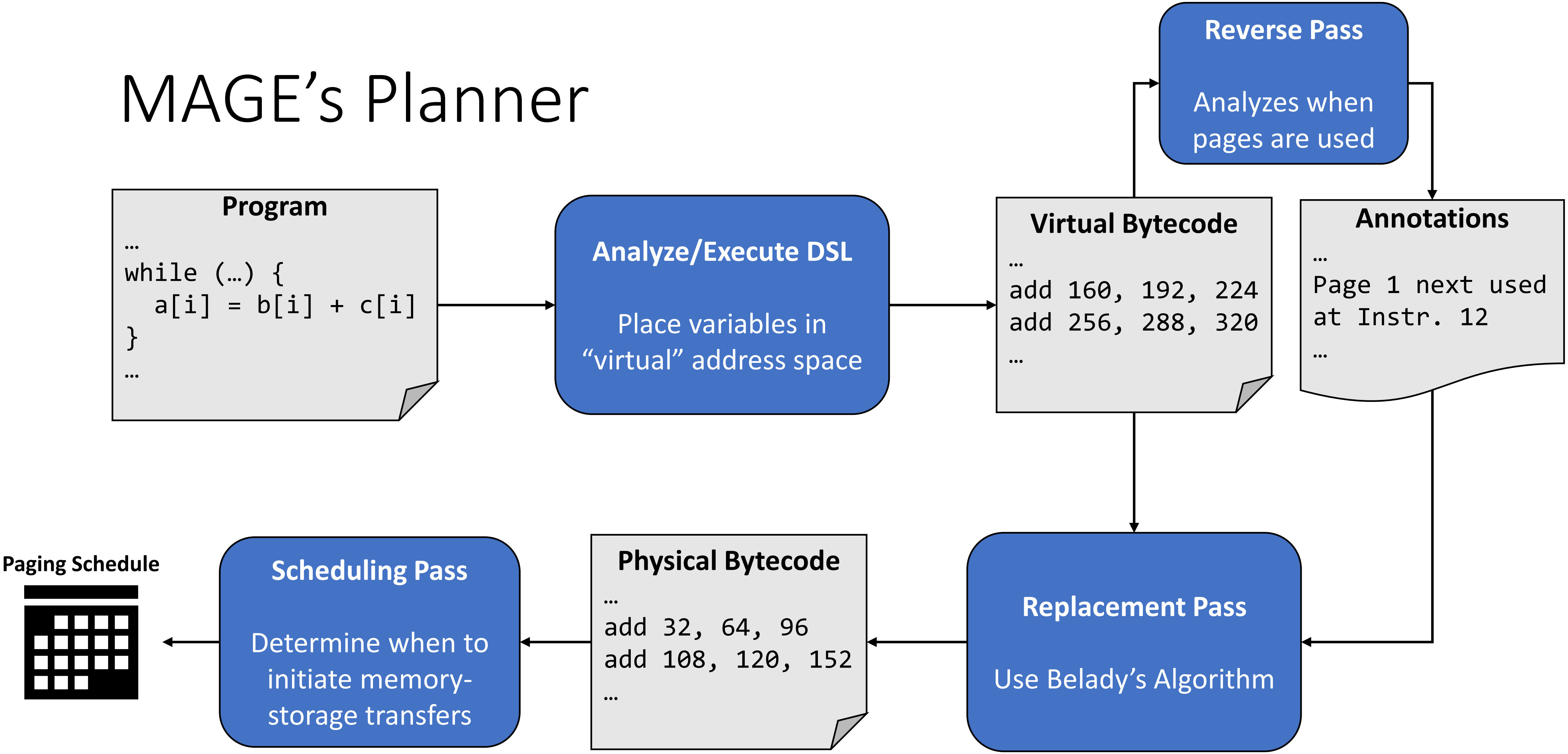


Belady’s algorithm (MIN)



Prefetch using the access pattern

MAGE's Planner



Additional Challenges

- How to cope with the size of the memory access pattern?
- How to extract the memory access pattern from the DSL?
- How to incorporate prefetching into Belady’s algorithm?
- How to parallelize/distribute the computation?
- How to extend MAGE with support for new SC protocols?

See paper for details

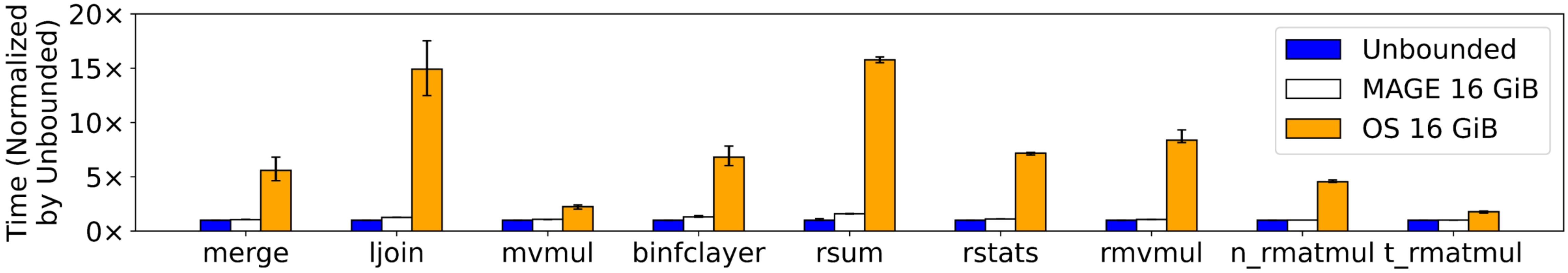
Implementation

- 11,000 lines of C++
- Supports:
  - Garbled circuits (type of SMPC)
  - CKKS (type of Fully Homomorphic Encryption)
- User program (runs on unmodified Linux)

Evaluation

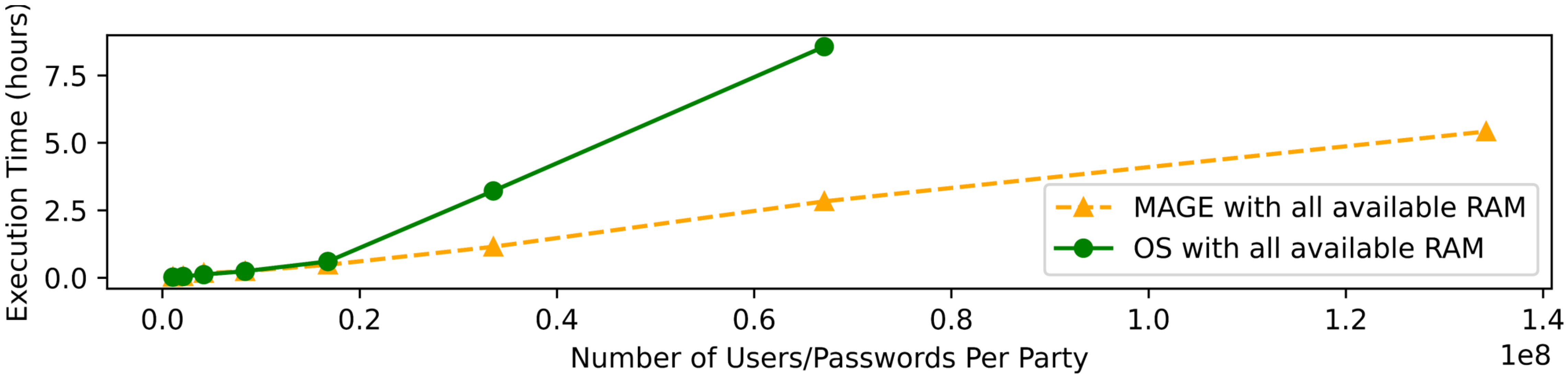
- We compare three setups:
  - **Unbounded**: Get enough memory to fit the entire computation
  - **MAGE**: Use our system MAGE
  - **OS**: Rely on OS swapping

Workloads



- For 7 workloads, MAGE performs within 10% of Unbounded
- For 7 workloads, MAGE outperforms OS by at least 4×
- MAGE outperforms OS by up to an order of magnitude

Password Reuse Application



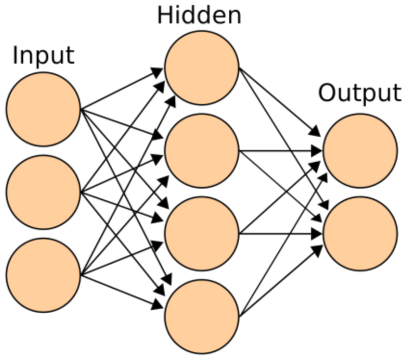
- MAGE can process up to 100 million user/password records
- For a given time budget, MAGE can handle a 3× larger problem

Conclusion

MAGE is a planner and runtime for secure computation. It:

- Leverages SC's obliviousness to rethink memory management for SC
- Pre-plans data transfers between storage and memory
- In many cases, runs SC at nearly in-memory speeds

MAGE's techniques could also potentially benefit:



Conclusion

MAGE is a planner and runtime for secure computation. It:

- Leverages SC's obliviousness to rethink memory management for SC
- Pre-plans data transfers between storage and memory
- In many cases, runs SC at nearly in-memory speeds

Thank you!

https://github.com/ucbrise/mage
https://github.com/ucbrise/mage-scripts

Sam Kumar
samkumar@cs.berkeley.edu

This material is based on work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1752814. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.