Oblivious Priority Queue and SSSP in the External Memory Setting

Arya Maheshwari* and **Elaine Shi°**

*CMU → MIT

°CMU

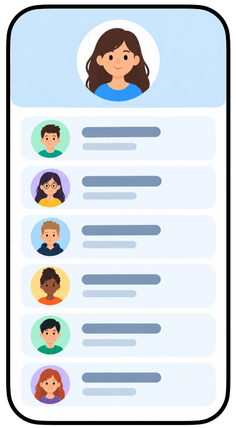
08.19.26

CRYPTO'26

- **A Motivating Application**
- Definitions and Model
- Our Setting and Results
- Some Technical Highlights

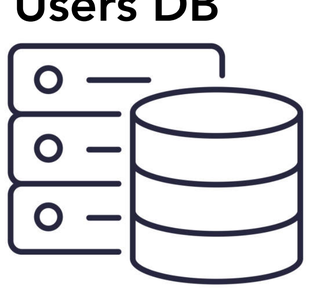
Private Contact Discovery in Signal

Alice



Server

Users DB

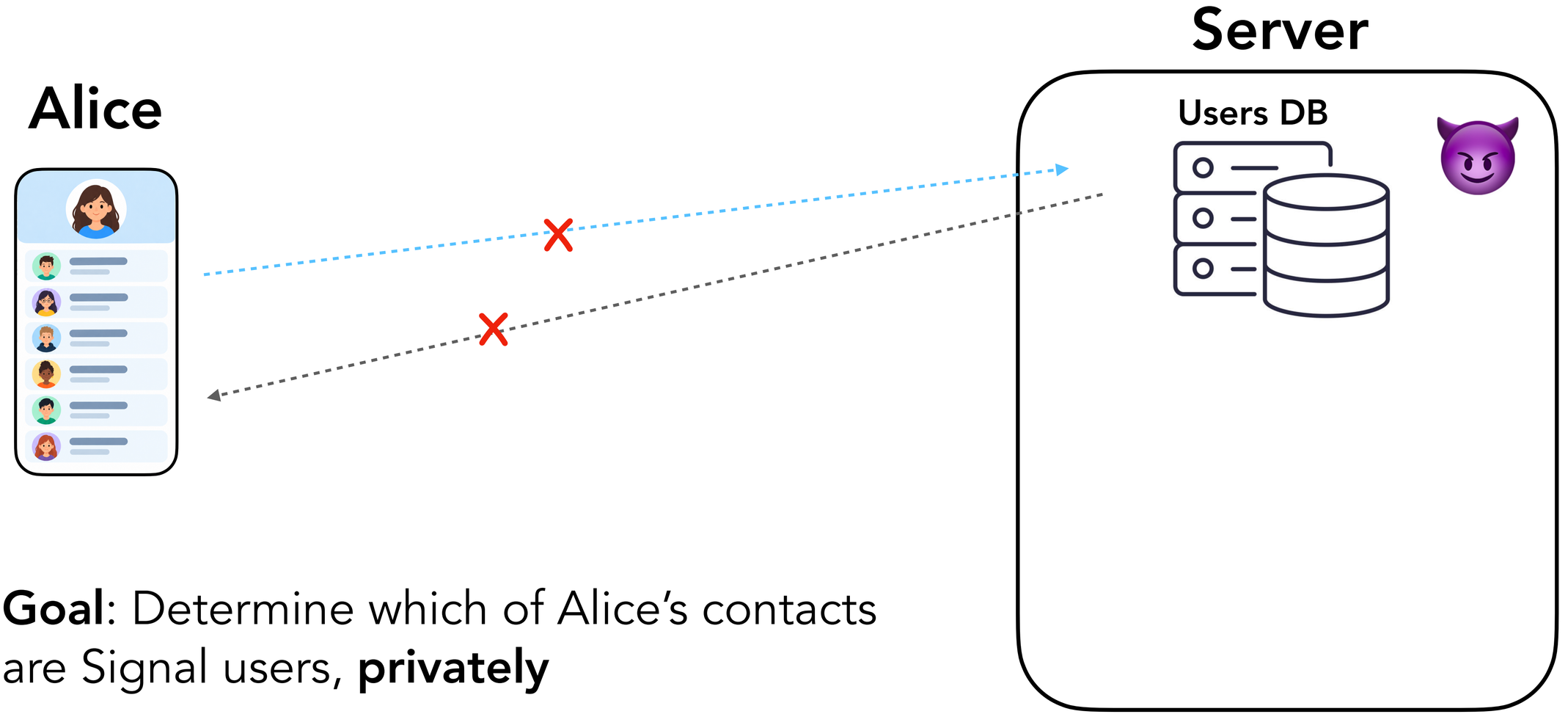




Goal: Determine which of Alice's contacts are Signal users, **privately**

Based on Signal blog posts, "Private contact discovery for Signal" and "Building a Faster ORAM Layer for Enclaves"

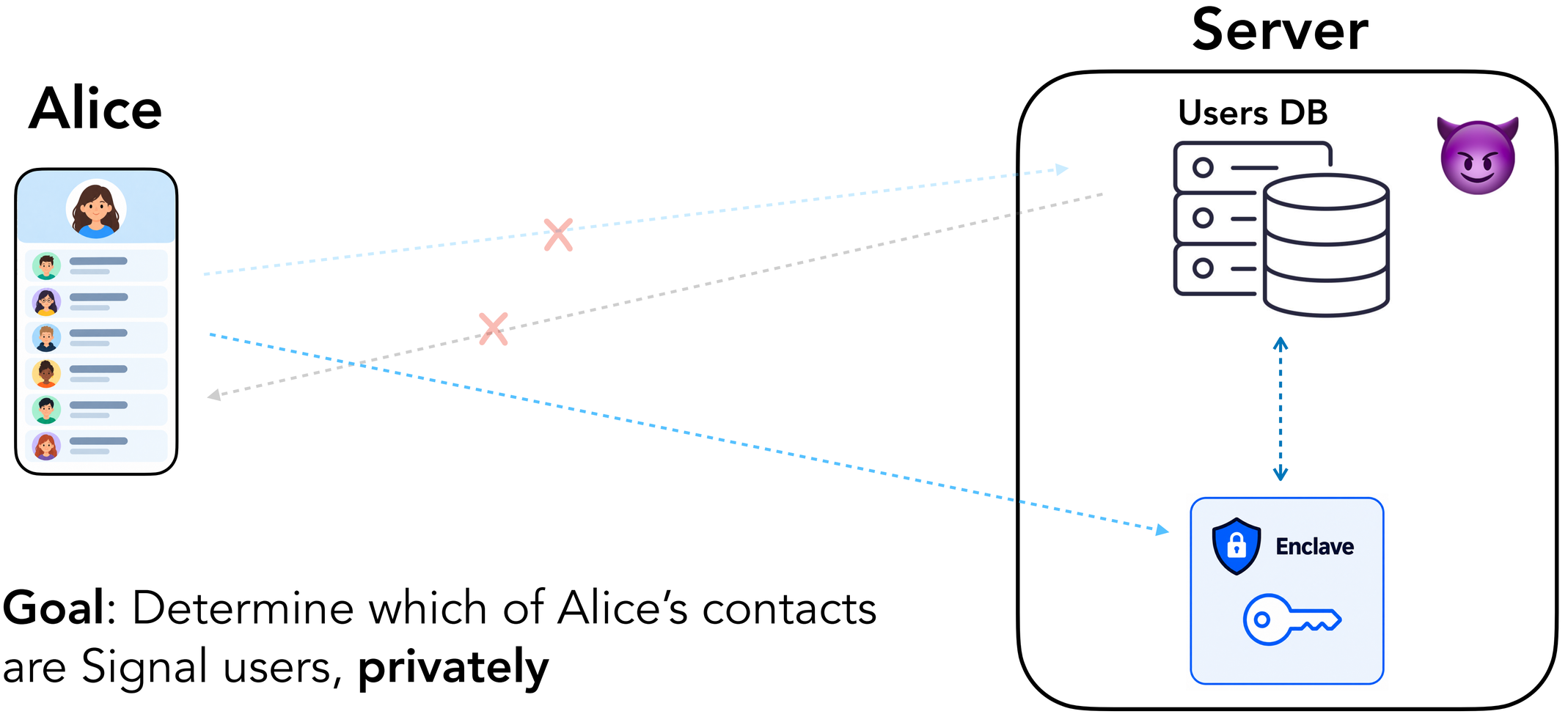
Private Contact Discovery in Signal



Goal: Determine which of Alice's contacts are Signal users, **privately**

Based on Signal blog posts, "Private contact discovery for Signal" and "Building a Faster ORAM Layer for Enclaves"

Private Contact Discovery in Signal

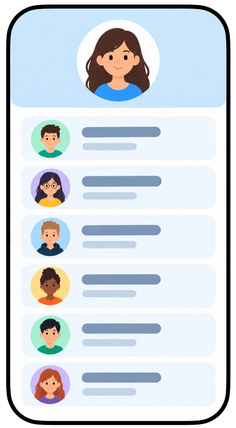


Goal: Determine which of Alice's contacts are Signal users, **privately**

Based on Signal blog posts, "Private contact discovery for Signal" and "Building a Faster ORAM Layer for Enclaves"

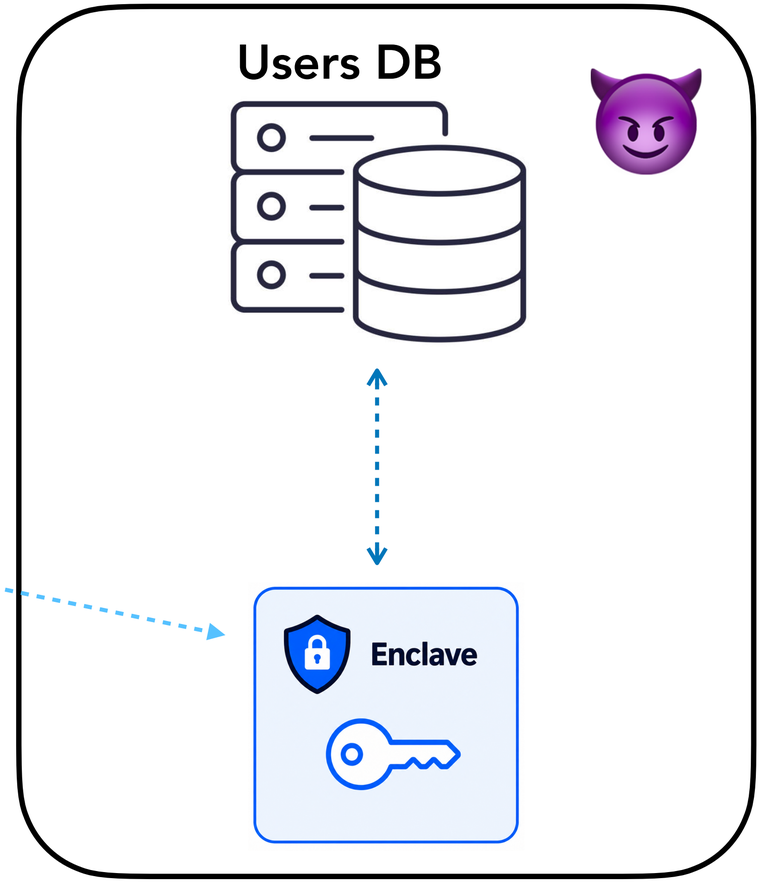
Private Contact Discovery in Signal

Alice



Problem: **memory access patterns** still leak information about private data!

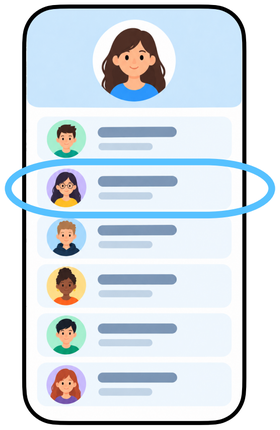
Server



Based on Signal blog posts, “Private contact discovery for Signal” and “Building a Faster ORAM Layer for Enclaves”

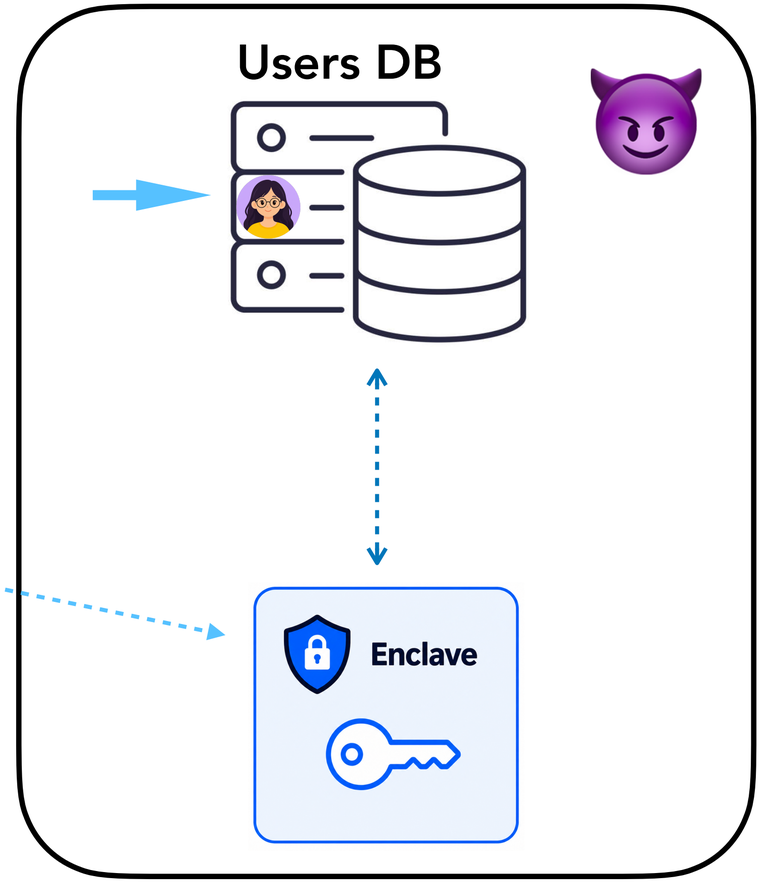
Private Contact Discovery in Signal

Alice



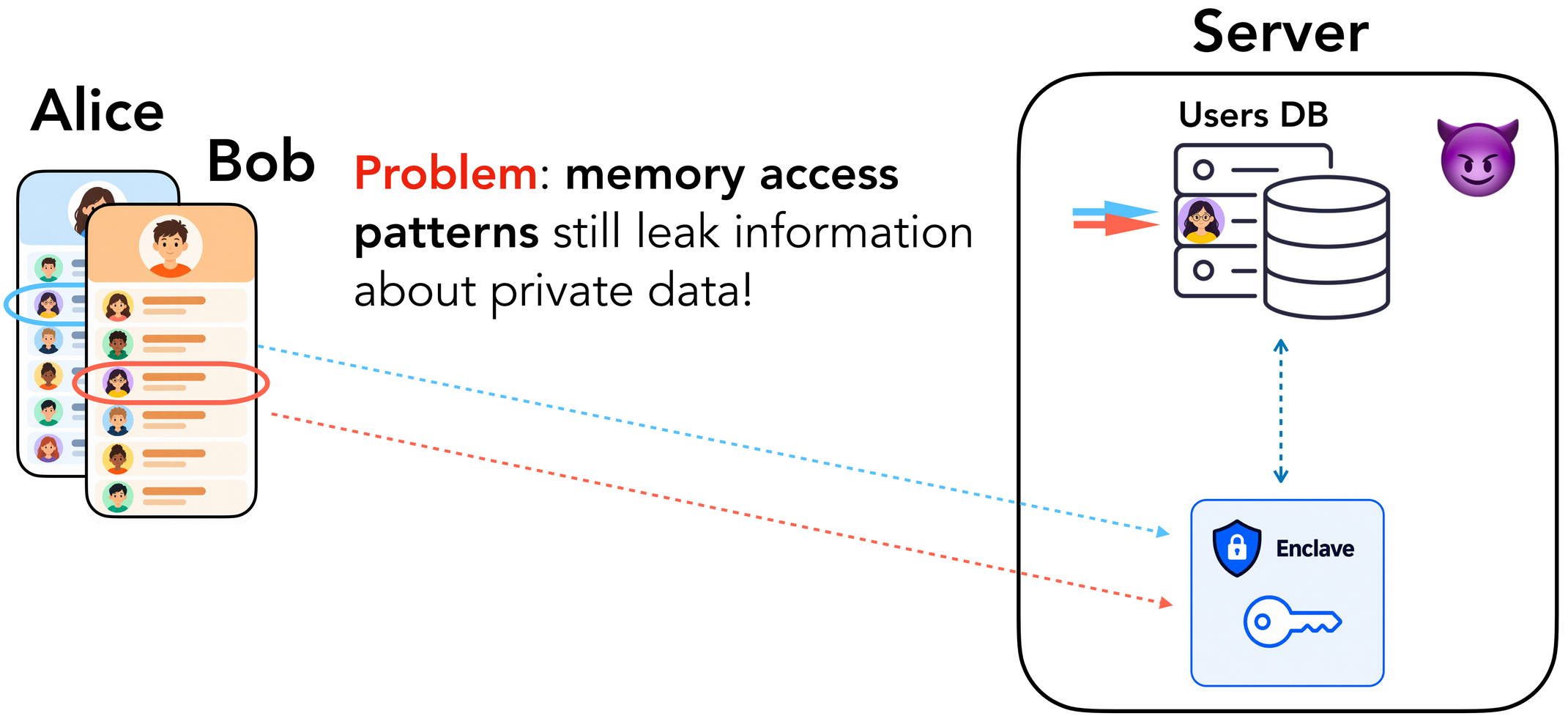
Problem: **memory access patterns** still leak information about private data!

Server



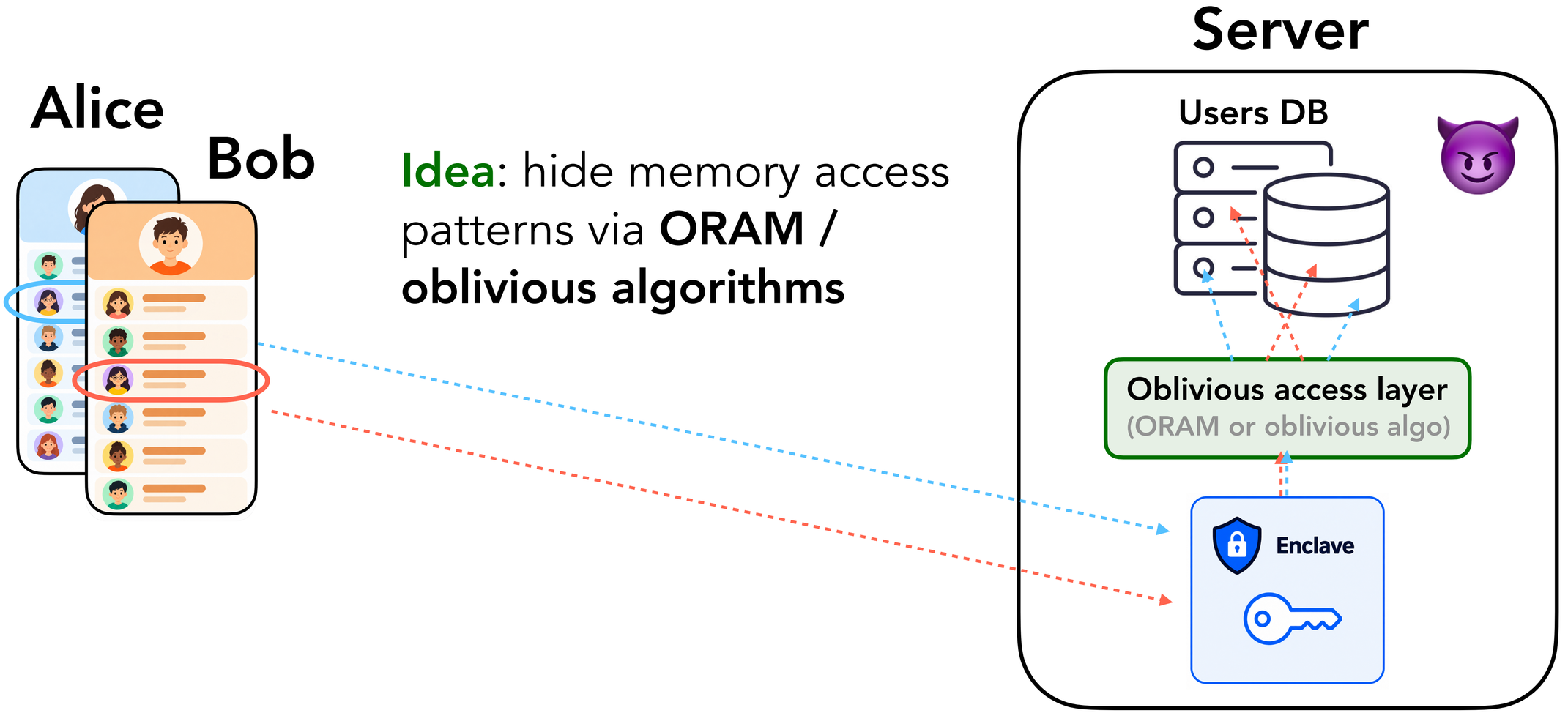
Based on Signal blog posts, “Private contact discovery for Signal” and “Building a Faster ORAM Layer for Enclaves”

Private Contact Discovery in Signal



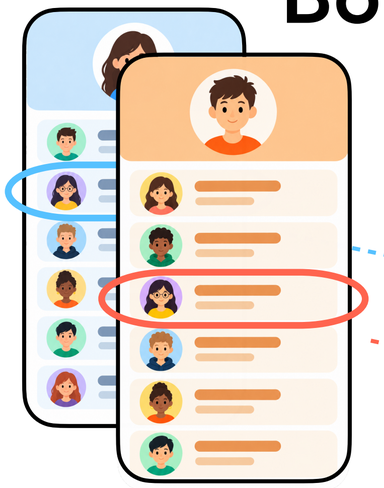
Based on Signal blog posts, “Private contact discovery for Signal” and “Building a Faster ORAM Layer for Enclaves”

Private Contact Discovery in Signal

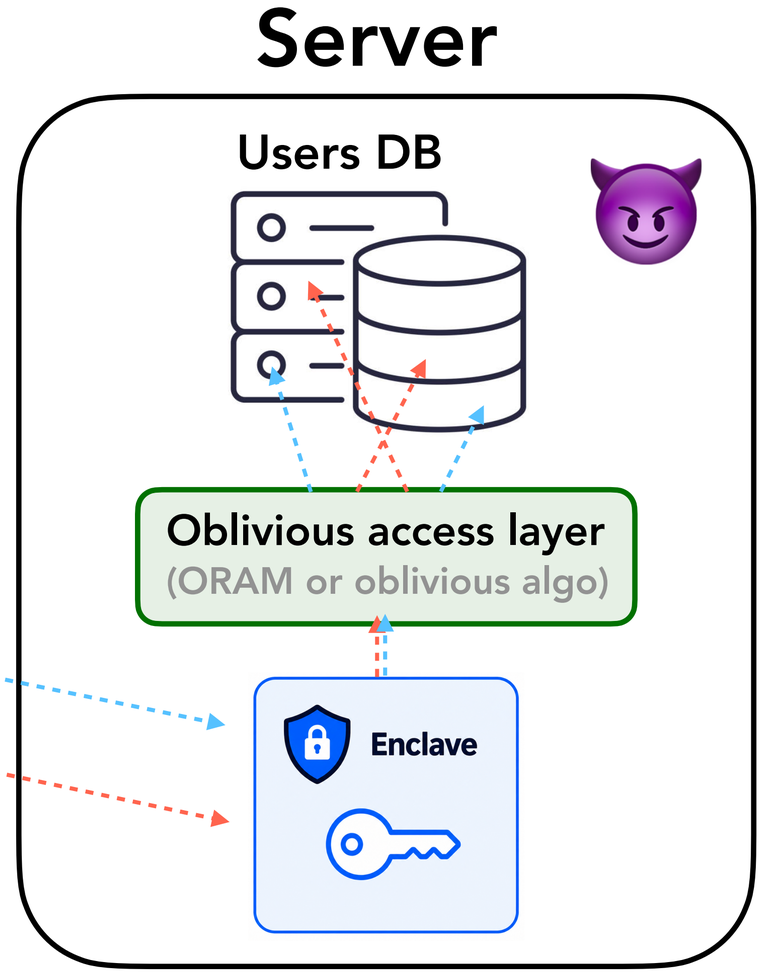


Based on Signal blog posts, “Private contact discovery for Signal” and “Building a Faster ORAM Layer for Enclaves”

Private Contact Discovery in Signal



Idea: hide memory access patterns via **ORAM / oblivious algorithms**



Desiderata

1. **Security**: Obliviousness
2. **Efficiency:** External-Memory Model

Based on Signal blog posts, “Private contact discovery for Signal” and “Building a Faster ORAM Layer for Enclaves”

- A Motivating Application
- **Definitions and Model**
- Our Setting and Results
- Some Technical Highlights

Security: **Obliviousness**

An algorithm A is **oblivious** if ∀ inputs x_1, x_2 of the same size,

$$\text{AccessPattern}(\, A(x_1)\,) \approx \text{AccessPattern}(\, A(x_2)\,)$$

Security: **Obliviousness**

An algorithm A is **oblivious** if ∀ inputs x_1, x_2 of the same size,

$$\mathrm{AccessPattern}(\, A(x_1)\,) \approx \mathrm{AccessPattern}(\, A(x_2)\,)$$

i.e., ∃ PPT simulator Sim s.t. $\forall x$, $\mathrm{AccessPattern}(\, A(x)\,) \approx \mathrm{Sim}(|x|)$

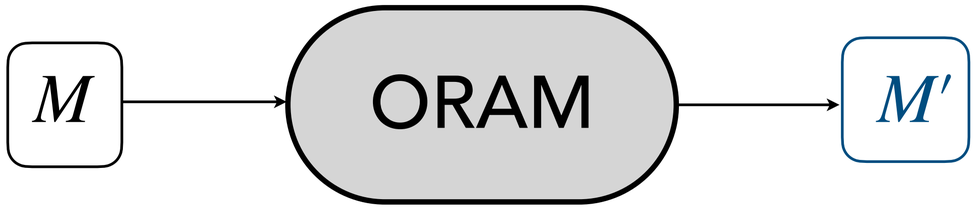
Security: **Obliviousness**

An algorithm A is **oblivious** if ∀ inputs x_1, x_2 of the same size,

$$\text{AccessPattern}(\, A(x_1)\,) \approx \text{AccessPattern}(\, A(x_2)\,)$$

i.e., ∃ PPT simulator Sim s.t. $\forall x$, $\text{AccessPattern}(\, A(x)\,) \approx \text{Sim}(|x|)$

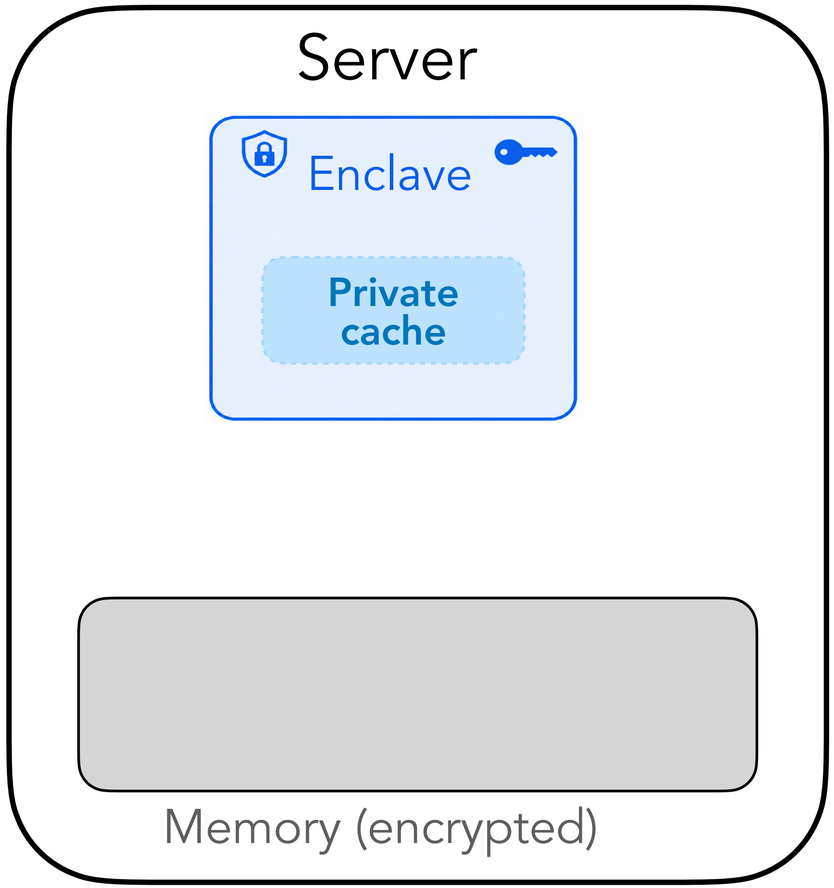
Oblivious RAM (ORAM): generic compiler [Gol87, GO96, SS13, SvDS+18, AKL+20, AEKL25, …]



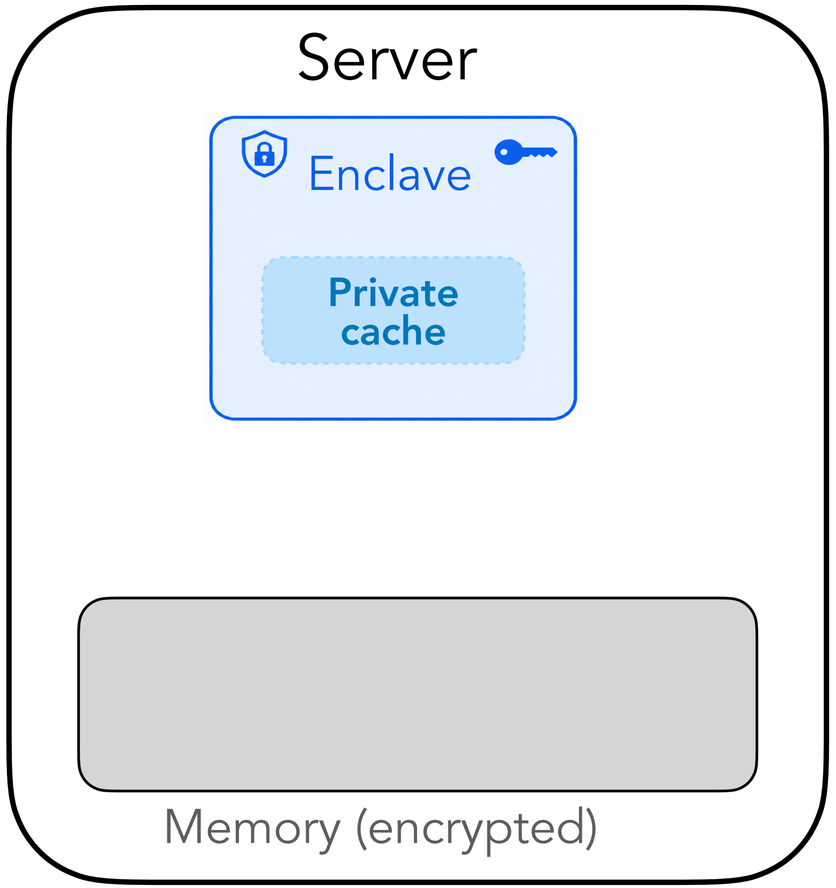
such that

1. M' is oblivious
2. $\forall x,\ M'(x) \approx M(x)$

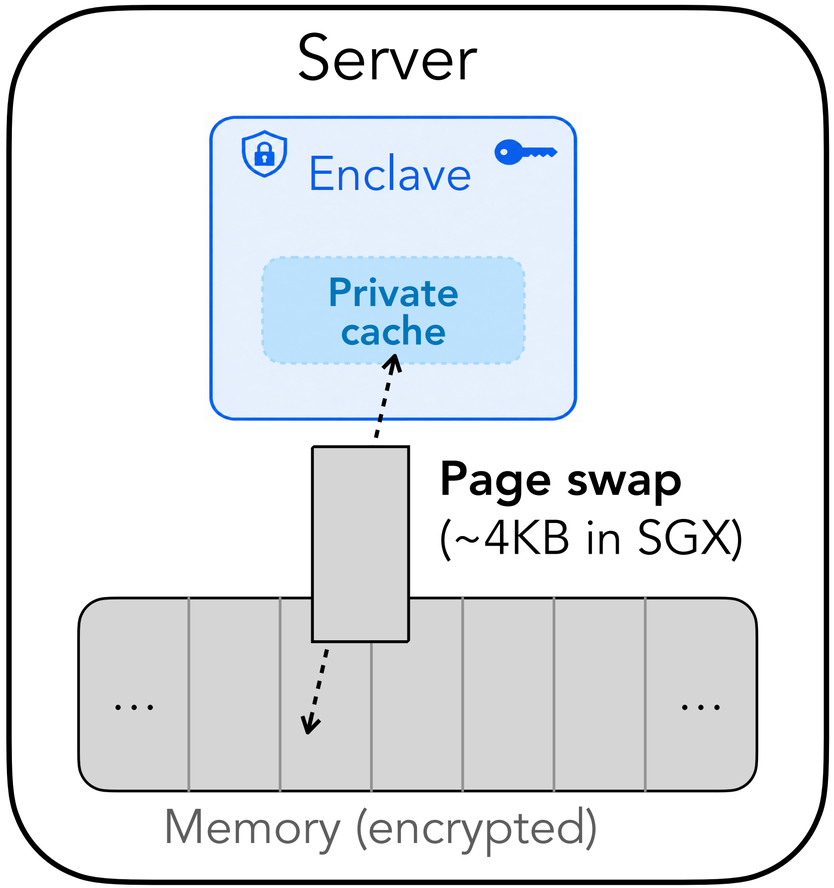
Efficiency: **The External-Memory (EM) Model**



Efficiency: **The External-Memory (EM) Model**



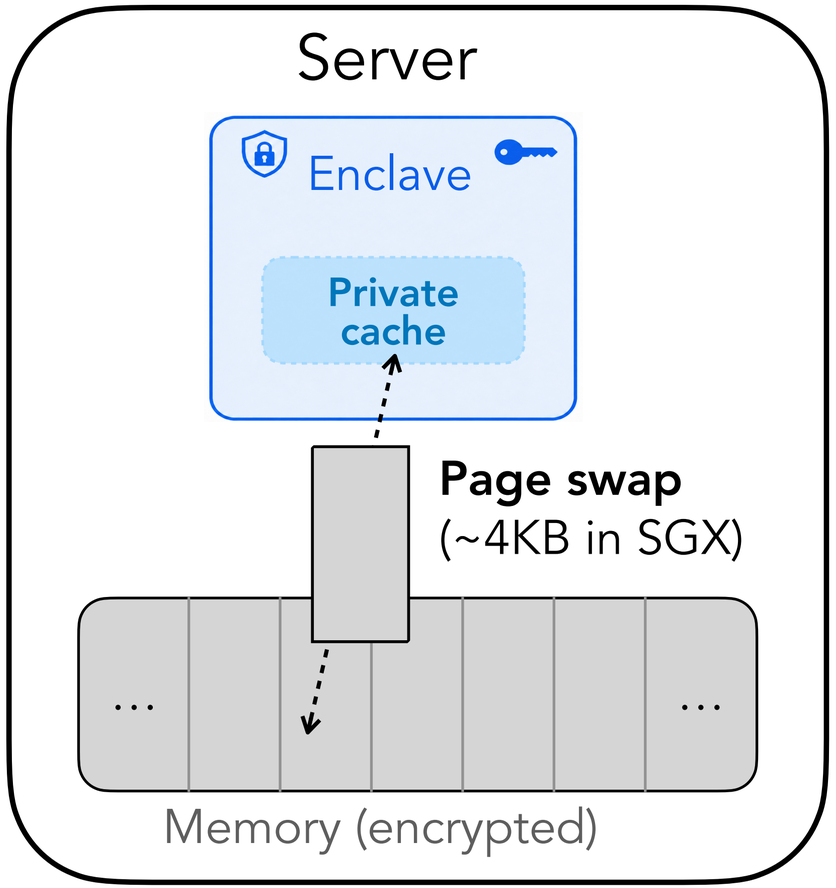
Efficiency: **The External-Memory (EM) Model**





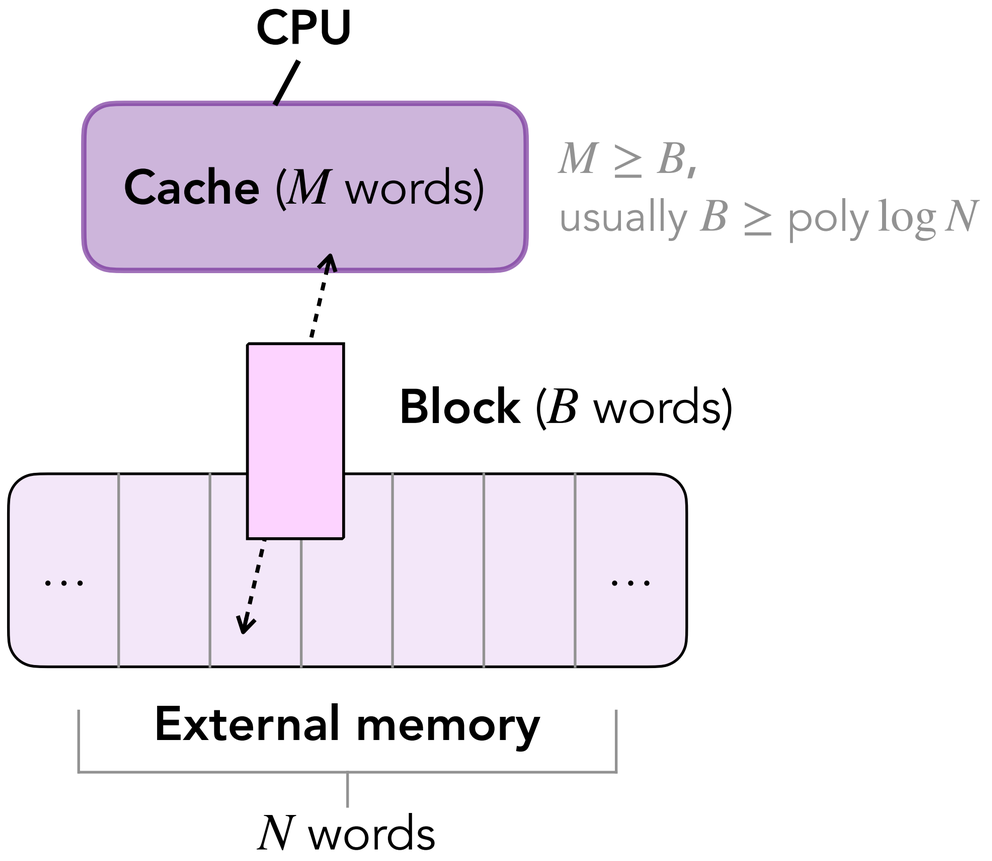



Efficiency: **The External-Memory (EM) Model**



model

External-Memory Model



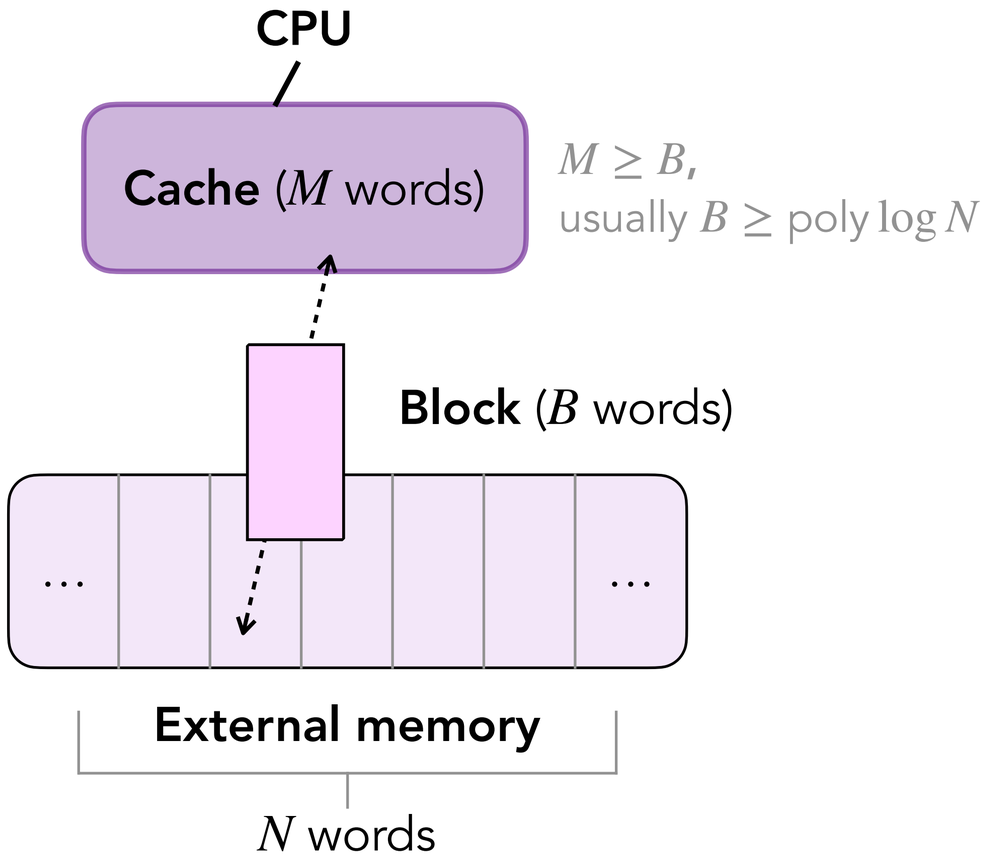
[AV88, Dem02, …]

Efficiency: **The External-Memory (EM) Model**

Efficiency metrics:

- **Work**: total computation
- **I/O cost**: # of block transfers

External-Memory Model



[AV88, Dem02, …]

Efficiency: **The External-Memory (EM) Model**

Efficiency metrics:

- **Work**: total computation
- **I/O cost**: # of block transfers

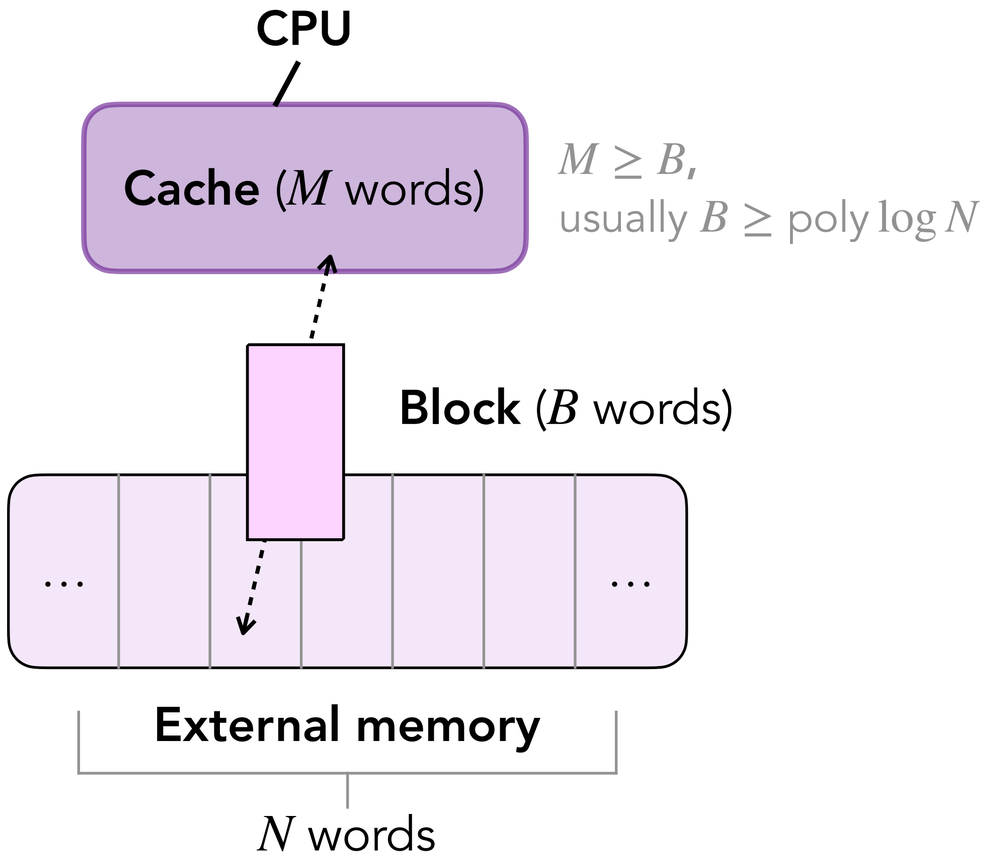
⇒ Exploit **locality** to minimize I/O cost

vs. standard **word RAM** model:

$$M, B = \Theta(1)$$

work = I/O cost = "running time"

External-Memory Model



[AV88, Dem02, …]

Generic Simulation via ORAM

Arbitrary non-private algorithm A

Generic oblivious compiler
via ORAM [AEKL25]

Oblivious algorithm A' on an EM machine

Generic Simulation via ORAM

Arbitrary non-private algorithm A

↓

Generic oblivious compiler
via ORAM [AEKL25]

↓

Oblivious algorithm A' on an EM machine

! Price of generality: logarithmic overhead is unavoidable for generic simulation [GO96, LN18, KL21]

Generic Simulation via ORAM

Arbitrary non-private algorithm A

↓

Generic oblivious compiler
via ORAM [AEKL25]

↓

Oblivious algorithm A' on an EM machine

! Price of generality: logarithmic overhead is unavoidable for **generic** simulation [GO96, LN18, KL21]

Generic Simulation via ORAM

Arbitrary non-private algorithm A

↓

Generic oblivious compiler
via ORAM [AEKL25]

↓

Oblivious algorithm A' on an EM machine

❗ **Price of generality:** logarithmic overhead is unavoidable for **generic** simulation [GO96, LN18, KL21]

💡Instead, what about **customized algorithms for specific tasks?**

Not bound by lower bounds for generic simulation!

Question:

Can **customized oblivious algorithms match non-private baselines** in the external-memory model?

A Nascent Field…

Classic (Non-Private) EM Algorithms

- Sorting ✓
- Priority Queue ✓
- Trees (MST, etc) ✓
- Graph Traversals ✓
- Standard Data Structures ✓
- … ✓

[AV88, Dem02, …]

Oblivious EM Algorithms

- Sorting ✓ [LSX19, RS21, GWT+25]
- Priority Queue ✓ [JLS21] (*kind of)
- ORAM (generic) ✓ [AEKL25]



A Nascent Field…

Classic (Non-Private) EM Algorithms

- Sorting ✓
- Priority Queue ✓
- Trees (MST, etc) ✓
- Graph Traversals ✓
- Standard Data Structures ✓
- … ✓

[AV88, Dem02, …]

Oblivious EM Algorithms

- Sorting ✓ [LSX19, RS21, GWT+25]
- Priority Queue ✓ [JLS21] (*kind of)
- ORAM (generic) ✓ [AEKL25]

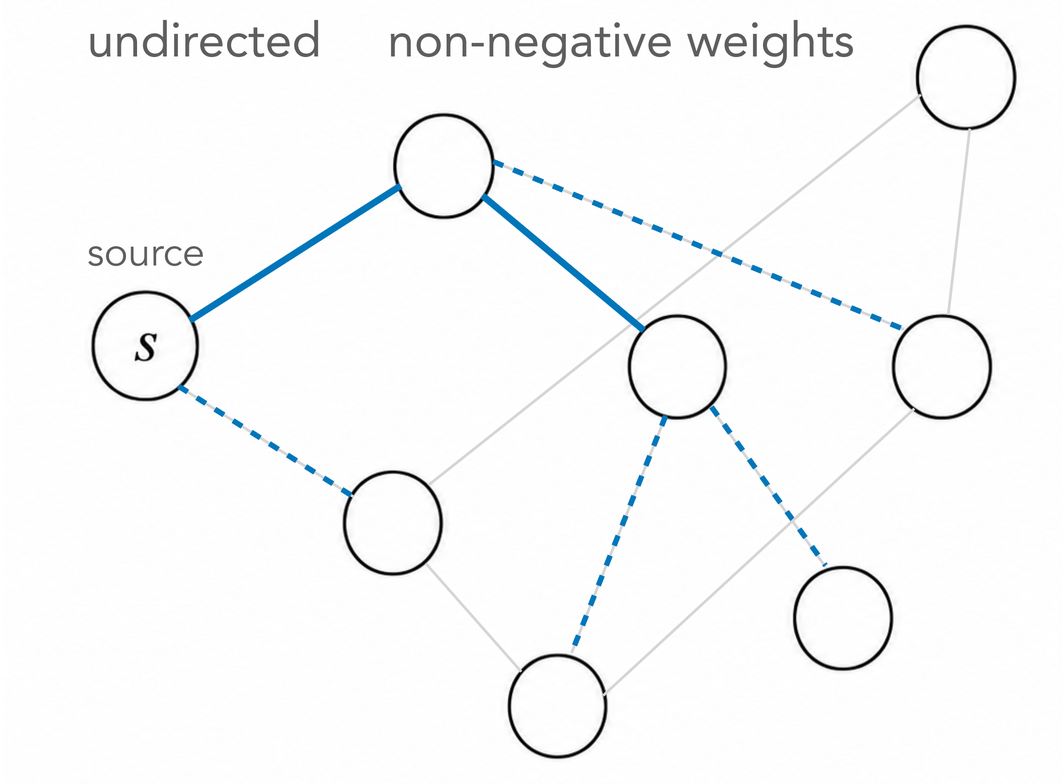
Our Work:
SSSP ✅
+
Priority Queue (revisited) ✅

↓

Long term: “Oblivious STL”

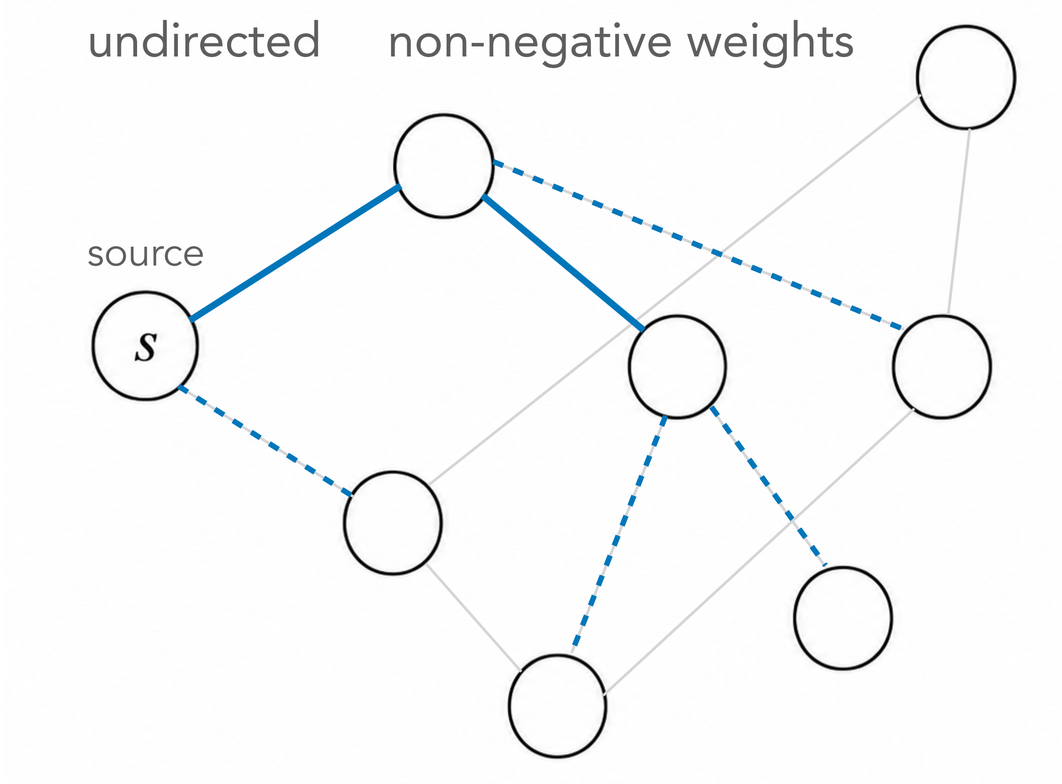
- A Motivating Application
- Definitions and Model
- **Our Setting and Results**
- Some Technical Highlights

Our Problem: Single-Source Shortest Path (SSSP)



V = # vertices E = # edges

Our Problem: Single-Source Shortest Path (SSSP)



V = # vertices E = # edges

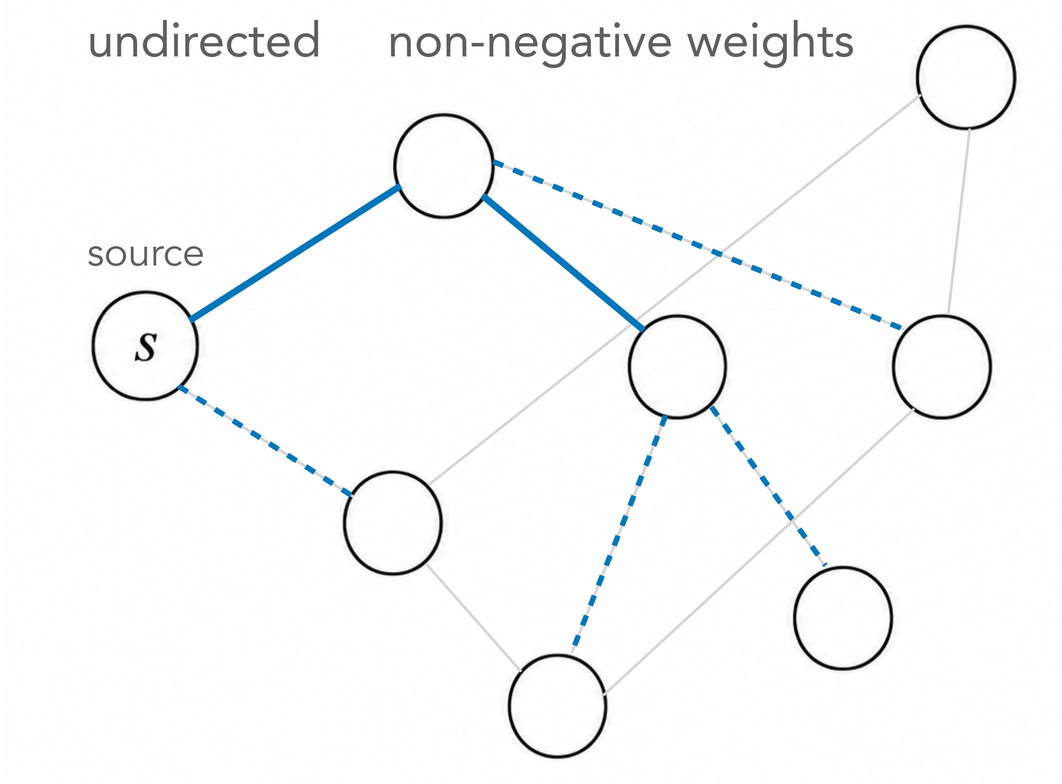
Standard (word RAM) algorithm:

$O(V \log V + E)$ work

via Dijkstra + Fibonacci heap

$\Rightarrow$ up to $O(V \log V + E)$ I/O cost

Our Problem: Single-Source Shortest Path (SSSP)



V = # vertices E = # edges

B = block size,
M = cache size
(both in words)

Standard (word RAM) algorithm:

$O(V \log V + E)$ work

via Dijkstra + Fibonacci heap

$\Rightarrow$ up to $O(V \log V + E)$ I/O cost

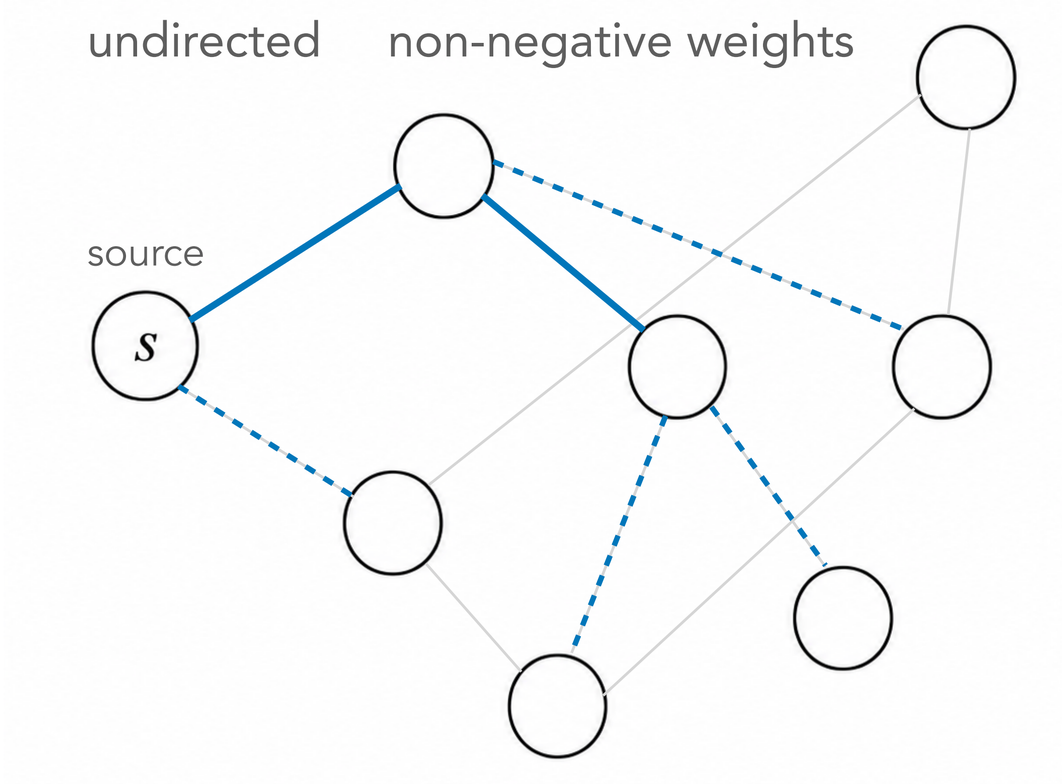
External memory algorithm:

$O(V + \frac{E}{B \log\log E} \log \frac{E}{M})$ I/O cost and

$O(V + E \log E)$ work

via [KS96 + JL19]

Our Problem: Single-Source Shortest Path (SSSP)



V = # vertices E = # edges

B = block size,
M = cache size
(both in words)

Standard (word RAM) algorithm:

$O(V \log V + E)$ work

via Dijkstra + Fibonacci heap

$\Rightarrow$ up to $O(V \log V + E)$ I/O cost

External memory algorithm:

$O(V + \frac{E}{B \log \log E} \log \frac{E}{M})$ I/O cost and

$O(V + E \log E)$ work

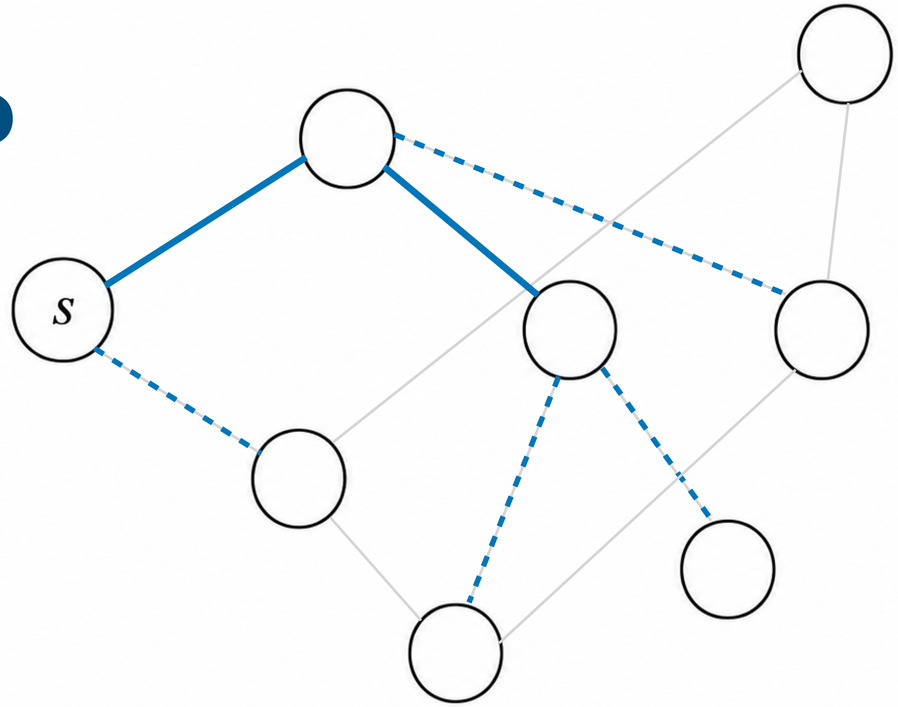
via [KS96 + JL19]

Main Result: Oblivious SSSP

There is an **oblivious** SSSP algorithm

that incurs $O(V + \frac{E}{B} \log \frac{E}{M})$ **I/O cost**

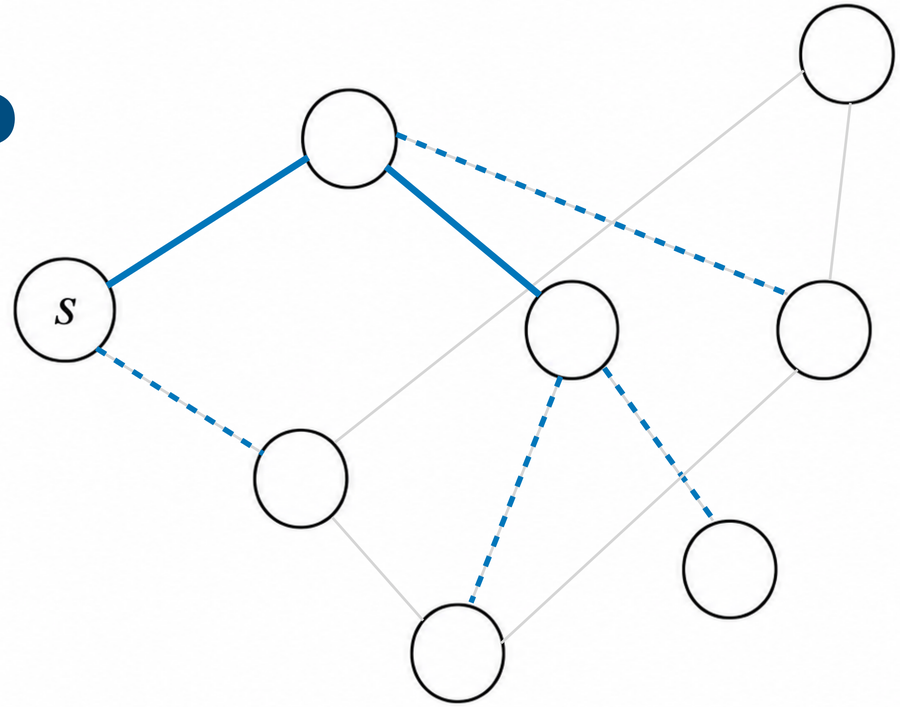
and $O(E \log E)$ **total work.***



*assuming the existence of PRFs and standard tall cache assumptions ($M \geq B^2$)

*in the standard case where $E = \Omega(V)$

Main Result: Oblivious SSSP



There is an **oblivious** SSSP algorithm that incurs $O(V + \frac{E}{B} \log \frac{E}{M})$ **I/O cost** and $O(E \log E)$ **total work.***

matches up to $\log \log E$ factor!

Non-private baseline:

$O(V + \frac{E}{B \log \log E} \log \frac{E}{M})$ I/Os,

$O(E \log E)$ work

*assuming the existence of PRFs and standard tall cache assumptions ($M \geq B^2$)

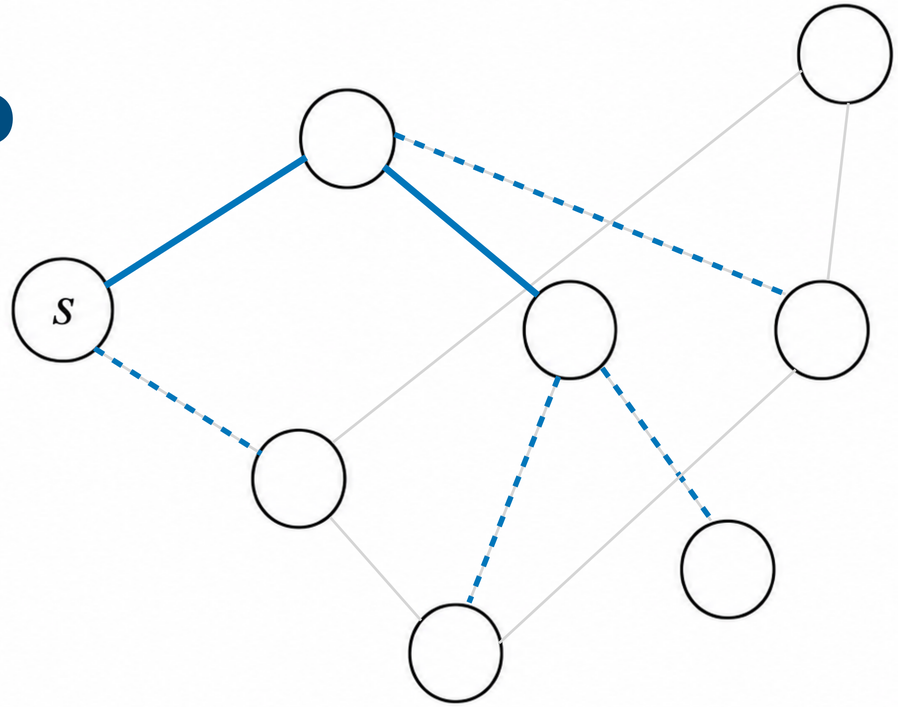
*in the standard case where $E = \Omega(V)$

Main Result: Oblivious SSSP

There is an **oblivious** SSSP algorithm

that incurs $O(V + \frac{E}{B} \log \frac{E}{M})$ **I/O cost**

and $O(E \log E)$ **total work.***



Non-private baseline:

$O(V + \frac{E}{B \log \log E} \log \frac{E}{M})$ I/Os,

$O(E \log E)$ work

Oblivious baseline

- Best word-RAM O-SSSP:

  $O(V + \frac{E}{\log B} \log E)$ I/Os

  [AHR24] + van emde Boas layout

- (more baseline comparisons in paper)

*assuming the existence of PRFs and tall cache assumptions ($M \geq B^2$, etc)

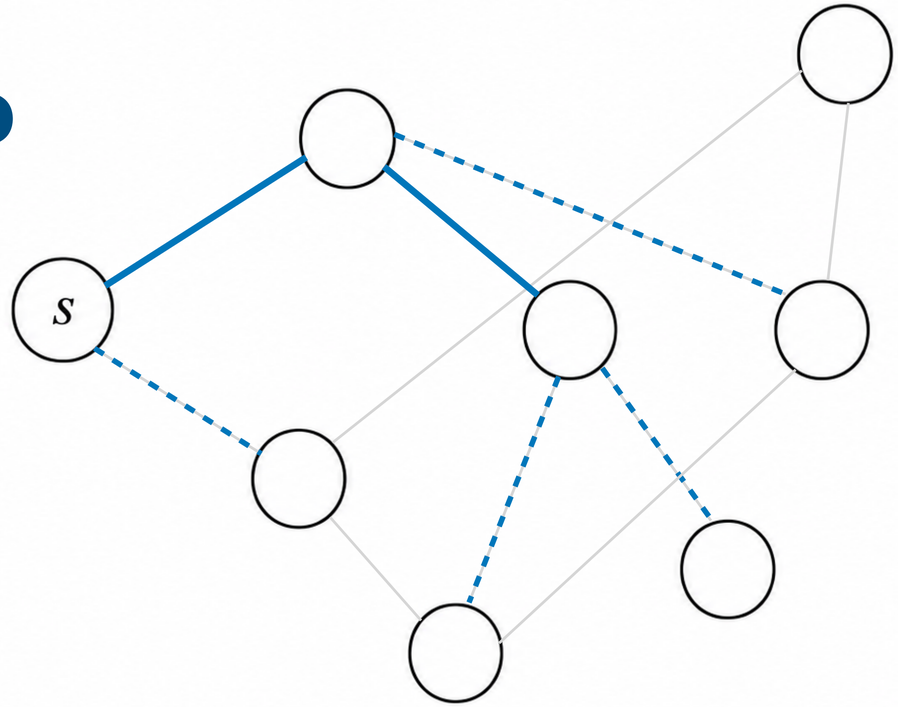
*in the standard case where $E = \Omega(V)$

Main Result: Oblivious SSSP

There is an **oblivious** SSSP algorithm

that incurs $O(V + \frac{E}{B} \log \frac{E}{M})$ **I/O cost**

and $O(E \log E)$ **total work.***



Non-private baseline:

$O(V + \frac{E}{B \log \log E} \log \frac{E}{M})$ I/Os,

$O(E \log E)$ work

Oblivious baseline

- Best word-RAM O-SSSP:

  $O(V + \frac{E}{\log B} \log E)$ I/Os

  [AHR24] + van emde Boas layout

- (more baseline comparisons in paper)

*assuming the existence of PRFs and tall cache assumptions ($M \geq B^2$, etc)

*in the standard case where $E = \Omega(V)$

The Bigger Picture: Algorithms ∩ Crypto



Techniques & models from algorithms literature enable more efficient crypto on real-world architectures

External-Memory Algorithms ⟷ Oblivious Simulation

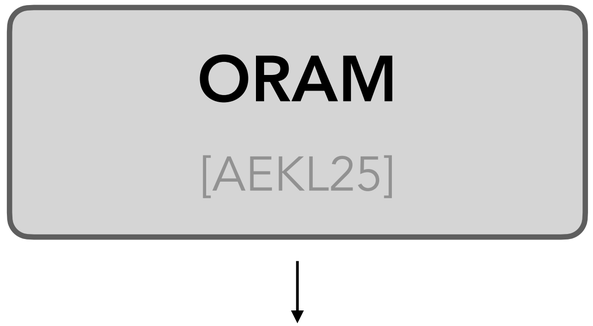
- A Motivating Application
- Definitions and Model
- Our Setting and Results
- **Some Technical Highlights**

Oblivious SSSP: Strawman

Target: $O\left(V + \frac{E}{B}\log\frac{E}{M}\right)$ I/Os

Dijkstra's + Fibonacci heap

$O(E + V \log V)$



Oblivious SSSP

1. Insert $(s, 0)$ into PQ
2. While PQ is not empty:
    1. Extract minimum (u, d) from PQ
    2. Mark u as visited; set its distance to d
    3. For all neighbors v of u:
       If v not visited, insert $(v, d + w_{uv})$ into PQ

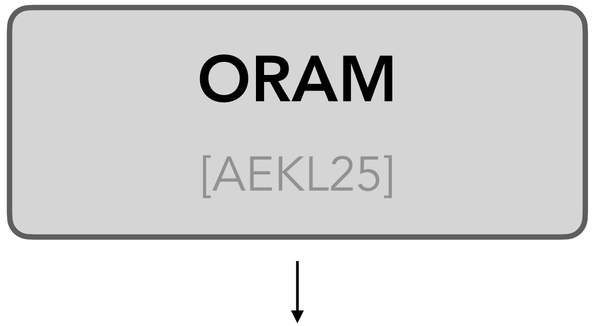
*assuming standard case where $E = \Omega(V)$

Oblivious SSSP: Strawman

Target: $O\left(V + \frac{E}{B}\log\frac{E}{M}\right)$ I/Os

Dijkstra's + Fibonacci heap

$O(E + V \log V)$



Oblivious SSSP

1. Insert $(s, 0)$ into PQ
2. While PQ is not empty:
   1. Extract minimum (u, d) from PQ
   2. Mark u as visited; set its distance to d
   3. For all neighbors v of u:
      If v not visited, insert $(v, d + w_{uv})$ into PQ

Problem: E accesses to `visited` array in Dijkstra

$\Rightarrow \Omega(E \cdot \log_B(E/M))$ I/O cost

*assuming standard case where $E = \Omega(V)$

Oblivious SSSP: Strawman

Target: $O\left(V + \frac{E}{B}\log\frac{E}{M}\right)$ I/Os

Dijkstra's + Fibonacci heap

$O(E + V \log V)$

↓

ORAM

[AEKL25]

↓

Oblivious SSSP

1. Insert $(s, 0)$ into PQ
2. While PQ is not empty:
   1. Extract minimum (u, d) from PQ
   2. Mark u as visited; set its distance to d
   3. For all neighbors v of u:
      If v not visited, insert $(v, d + w_{uv})$ into PQ

Problem: E accesses to `visited` array in Dijkstra

$\Rightarrow \Omega(E \cdot \log_B(E/M))$ I/O cost

External-memory SSSP (Kumar & Schwabe '96)**:**
Replaces `visited` array with **2nd PQ**!

$$O\left(V + \frac{E}{B}\log\frac{E}{M}\right) \text{ I/Os*}$$

*assuming standard case where $E = \Omega(V)$

*additional $\log\log E$ improvement via recent PQ construction [JL19]

Oblivious SSSP: **Algorithm**

Starting point: EM algorithm → make it oblivious in a tailored way

- PQ → Oblivious PQs ✓
- Adjacency list access ?

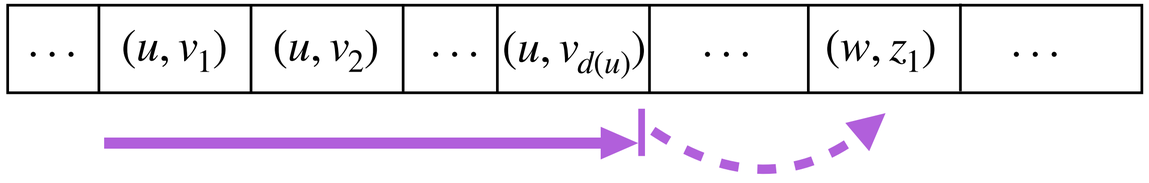
Oblivious SSSP: **Algorithm**

Starting point: EM algorithm → make it oblivious in a tailored way

- PQ → Oblivious PQs ✓
- Adjacency list access ?

Problem: "for all neighbors v of u…" →

! Access pattern reveals degrees of vertices



Oblivious SSSP: **Algorithm**

Starting point: EM algorithm → make it oblivious in a tailored way

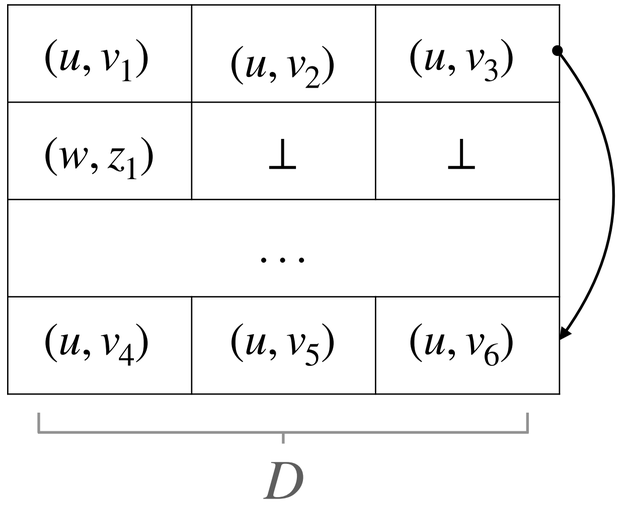
- PQ → Oblivious PQs ✓
- Adjacency list access **?**

Solution: **oblivious adjacency list iterator**

inspired by construction of [Ost24]

Key idea: **"regularize"** the adjacency list

- Store list in segments of size D, where $D = E/V$ is a public size
- ⇒ access pattern depends only on public D, E, V



Oblivious SSSP: **Algorithm**

Starting point: EM algorithm → make it oblivious in a tailored way

- PQ → Oblivious PQs ✓
- Adjacency list access → Oblivious adjacency list iterator ✓
- Some additional tricks to put everything together

En Route: Oblivious PQ

Oblivious PQ, revisited

Operations:

- `FindMin`
- `ExtractMin`
- `IsEmpty`
- `DecreaseKey`
- `Insert`
- `Delete`

En Route: Oblivious PQ

Oblivious PQ, revisited

Operations:

- `FindMin`
- `ExtractMin`
- `IsEmpty`
- `DecreaseKey`
- `Insert`
- `Delete`

	I/O cost	Total work
[JLN19] lower bound	$\Omega\left(\frac{1}{B}\log\frac{N}{M}\right)$	$\Omega(\log N)$
[JLS21]	$O\left(\frac{1}{B}\log\frac{N}{M}\right)$	$O(\log N)$

En Route: Oblivious PQ

😈 can observe both block-level accesses *and word-level accesses within block*

Oblivious PQ, revisited

	I/O cost	Total work	Strongly oblivious?
[JLN19] lower bound	$\Omega\left(\frac{1}{B}\log\frac{N}{M}\right)$	$\Omega(\log N)$	–
[JLS21]	$O\left(\frac{1}{B}\log\frac{N}{M}\right)$	$O(\log N)$	✗

Operations:

- `FindMin`
- `ExtractMin`
- `IsEmpty`
- `DecreaseKey`
- `Insert`
- `Delete`

En Route: Oblivious PQ

😈 can observe both block-level accesses *and word-level accesses within block*

Oblivious PQ, revisited

There is a **strongly oblivious** PQ where each operation incurs amortized $O(\frac{1}{B}\log\frac{N}{M})$ **I/O cost** & $O(\log n)$ **work.***

Operations:

- `FindMin`
- `ExtractMin`
- `IsEmpty`
- `DecreaseKey`
- `Insert`
- `Delete`

	I/O cost	Total work	Strongly oblivious?
[JLN19] lower bound	$\Omega\left(\frac{1}{B}\log\frac{N}{M}\right)$	$\Omega(\log N)$	–
[JLS21]	$O\left(\frac{1}{B}\log\frac{N}{M}\right)$	$O(\log N)$	❌
Ours	$O\left(\frac{1}{B}\log\frac{N}{M}\right)$	$O(\log N)$	✅

*assuming the existence of PRFs and that $M \geq \log^c \lambda$ for suitable constant c

Current & Future Work

Directed SSSP: Current work on $(\log \log E)$-overhead algorithm, using new **oblivious buffered repository tree** construction.

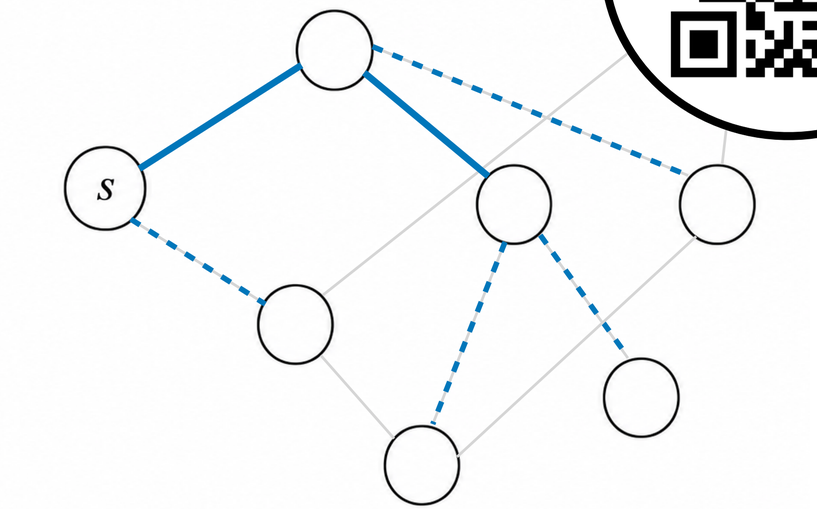
Future Directions:

- Develop more of "Oblivious STL" (e.g. BST, additional graph algos)
- Extend to related real-world architecture models (e.g. parallel [RS21])

See the paper for…

- Full details on oblivious PQ and oblivious adjacency list iterator
- Comparison to additional oblivious baselines
- Two approaches to generic oblivious simulation in the external-memory setting
- All proofs

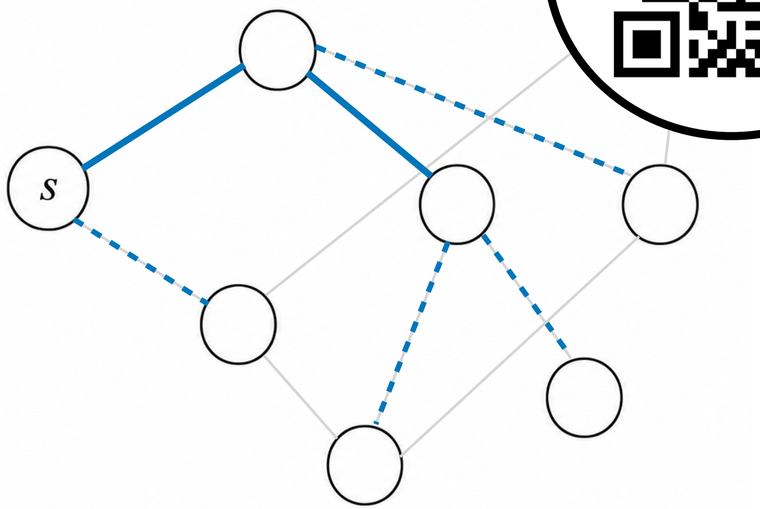
Paper:



See the paper for…

- Full details on oblivious PQ and oblivious adjacency list iterator
- Comparison to additional oblivious baselines
- Two approaches to generic oblivious simulation in the external-memory setting
- All proofs

Paper:



Thank you!

amahesh@mit.edu

Our Results

Theorem 1: Oblivious SSSP

Scheme	I/O cost	Total work
EM-OSim of SSSP [AEKL25] + [FT87]	$\log_B(E/M)\cdot O(E+V\log V)$	$O(B^\epsilon\cdot\log E\cdot\log_B E\cdot(E+V\log V))$
OSim of EM-SSSP [AKL+22] + [JL19]	$\log\frac{E}{M}\cdot O\left(V+\frac{E}{B\log\log E}\log\frac{E}{B}\right)$	$\log\frac{E}{B}\cdot O\left(V+\frac{E\log(E/B)}{B\log\log E}\right)+\log M\cdot O\left(V+E\log E\right)$
Word-RAM O-SSSP [AHR24] + [vEBKZ76]	$O(E\log_B E)$	$O(E\log E)$
Ours	$O(V+\frac{E}{B}\log\frac{E}{M})$	$O(E\log E)$
Non-private [JL19]	$O(V+\frac{E}{B\log\log E}\log\frac{E}{M})$	$O(E\log E)$