CPSC 768: Scalable and Private Graph Algorithms

Lecture 1: Intro

Quanquan C. Liu
quanquan.liu@yale.edu

“Practical” Algorithms

Heuristics

“Practical” Algorithms

Heuristics

Engineering

“Practical” Algorithms

Heuristics

Engineering

Math Tools

"Practical" Algorithms

Heuristics

Engineering

Tools from ML

Math Tools

“Practical” Algorithms

Heuristics

Engineering

Tools from ML

Math Tools

“Practical” Algorithms

Provably efficient algorithms

Models that **consider modern challenges** and **computing environments**

Goal: **Data structures** with best theoretical guarantees

Implementable in practice

Key Modern Challenge: **Massive** Datasets

MASSIVE graph data sets

ClueWeb

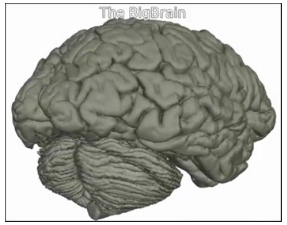
Google

~ 27 billion comments

~ 10 billion edges

~ 6 trillion edges

Web Data Commons Hyperlink Graph



~ 128 billion hyperlinks

~ 2 billion follows

~ 300 million neurons

Main Takeaway of the Class

- This class is focused on **research**
  - Use the **techniques you learn** outside of this class
  - Start **conducting research in these topics** and **more**
  - Exposure to practice problems and **open problems**
  - **Complete a final project**

Logistics

- Course website: https://quanquancliu.com/cpsc768
  - Everything important!
  - Syllabus
  - Recommended topics
  - Class schedule
  - Links to a bunch of papers
  - Class notes (posted as a batch for the week on Sunday)
- **Open problem session scheduling** + survey: https://forms.gle/CsKnH5DvuVu5XKXQA

Format and Workload

- **Two class presentations** on subjects of their choice related to the topics of the course. *50% of grade.*
  - Finalize dates and topics for presentations before **April 1: due Feb. 5**.
  - Finalize dates and topics for presentations on and **after April 1: due Feb. 26**.
- **One final project** (individual or with partner). *50% of grade.*

Format and Workload

- **One final project** (individual or with partner). *50% of grade.*
  - *Project proposal (1 page): due Feb. 26.*
  - *Progress report (2-3 pages): due March 27.*
  - *In-class presentation: last two weeks of class.*
  - *Final report (at least 8 pages, less than 20): April 24*

Final Project Details (in the context of SODA/SOSA/ALENEX)

- **Reading project (e.g. SOSA paper):**
  - Read 2-3 papers on the same project and survey on key ideas
  - Service to the community for hard to read papers!

Final Project Details (in the context of SODA/SOSA/ALENEX)

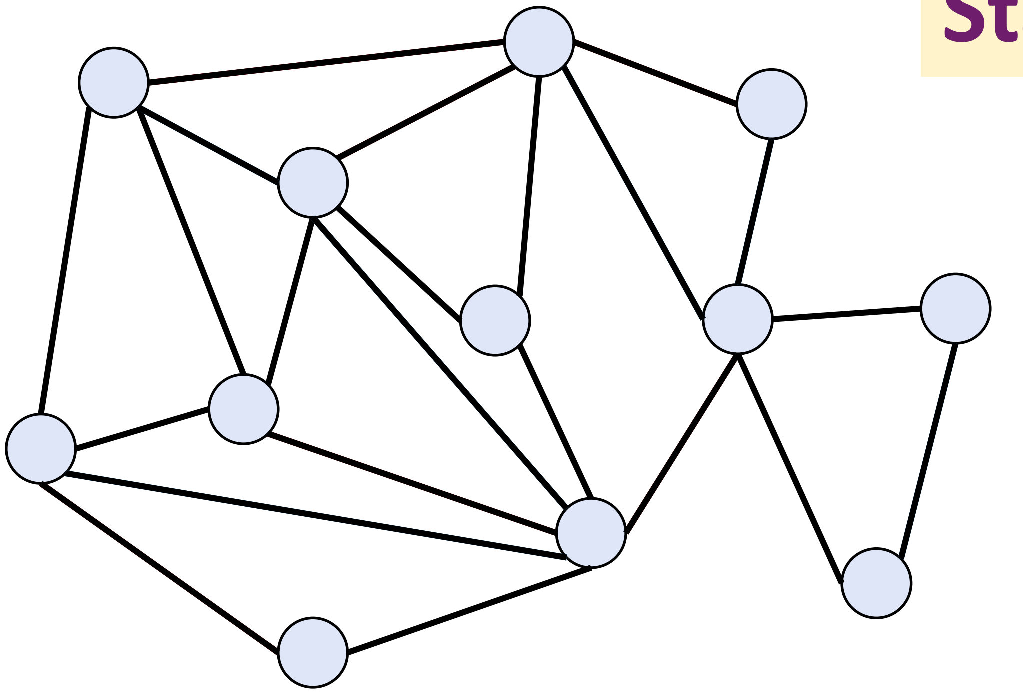
- **Reading project (e.g. SOSA paper):**
  - Read 2-3 papers on the same project and survey on key ideas
  - Service to the community for hard to read papers!
- **Theory project (e.g. SODA paper):**
  - Make an improvement over previous result
  - Solve an open problem

Final Project Details (in the context of SODA/SOSA/ALENEX)

- **Reading project (e.g. SOSA paper):**
  - Read 2-3 papers on the same project and survey on key ideas
  - Service to the community for hard to read papers!
- **Theory project (e.g. SODA paper):**
  - Make an improvement over previous result
  - Solve an open problem
- **Implementation project (e.g. ALENEX paper):**
  - Algorithm engineering implementation of an algorithm
  - Give a *more implementable* solution with same or better theory guarantees
- More details given in the syllabus

A Brief Overview of Some Models Covered in This Class

A Key Focus on Graphs



Static or Dynamic

A Brief Overview of Models We'll Cover

How to represent graph:
• On one machine • Distributed across many machines

A Brief Overview of Models We'll Cover

How to represent graph:	What resources to process:
• On one machine • Distributed across many machines	• Multiple cores • Communication network over machines

A Brief Overview of Models We'll Cover

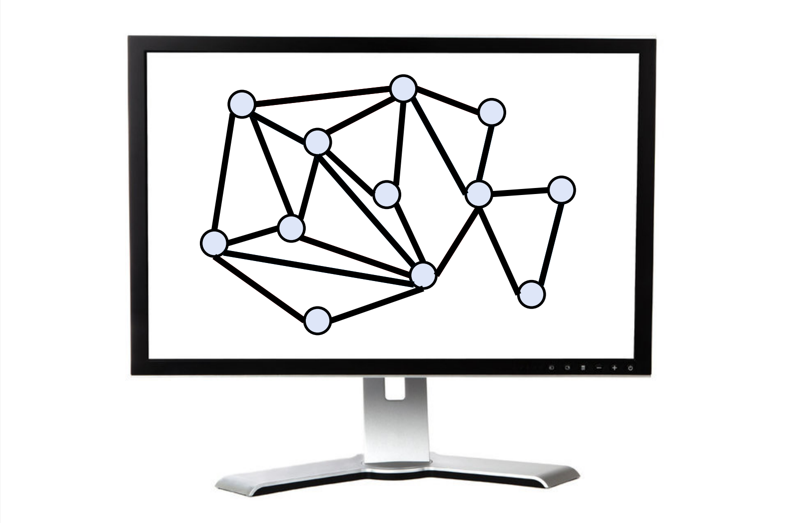
How to represent graph:	What resources to process:
• On one machine • Distributed across many machines	• Multiple cores • Communication network over machines
How updates are given:	
• In batches of multiple updates • As a stream of updates	

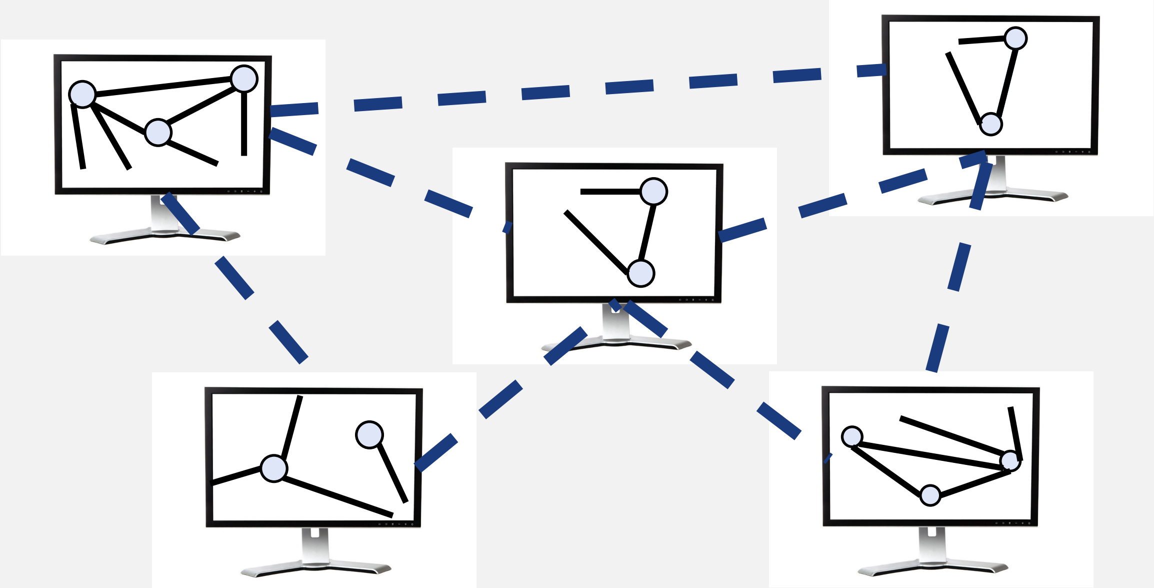
A Brief Overview of Models We'll Cover

How to represent graph:	What resources to process:
• On one machine • Distributed across many machines	• Multiple cores • Communication network over machines
How updates are given:	**Adversarial models:**
• In batches of multiple updates • As a stream of updates	• Adaptive/oblivious • Privacy violating • Central/Local

How to Represent the Graph

- On one machine
- Distributed across many machines





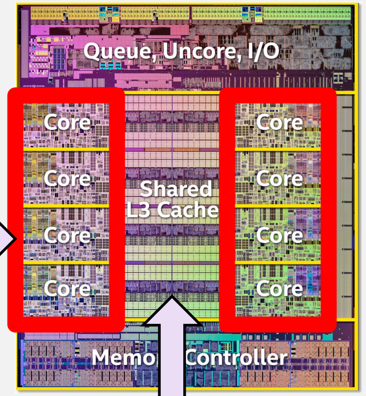
What Resources to Use to Process Graph

- Multiple cores and processors
- Communication network over machines

What Resources to Use to Process Graph

- **Multiple cores and processors**
- Communication network over machines

Cores: parallel processing

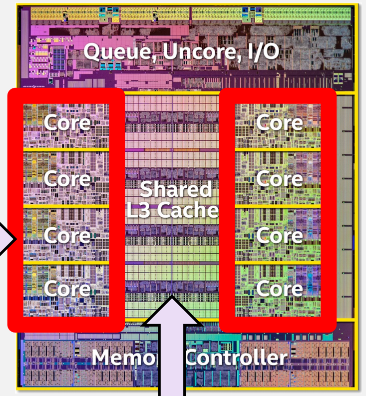


Shared-memory: read and write from same memory

What Resources to Use to Process Graph

- **Multiple cores and processors**
- Communication network over machines

Cores: parallel processing



Shared-memory: read and write from same memory

Shared-memory work-depth model

- **Work:**
  - Total number of operations
- **Depth/Span:**
  - Longest chain of sequential dependencies in algorithm

What Resources to Use to Process Graph

- Multiple cores and processors
- **Communication network over machines**

Massively Parallel Computation (MPC) Model

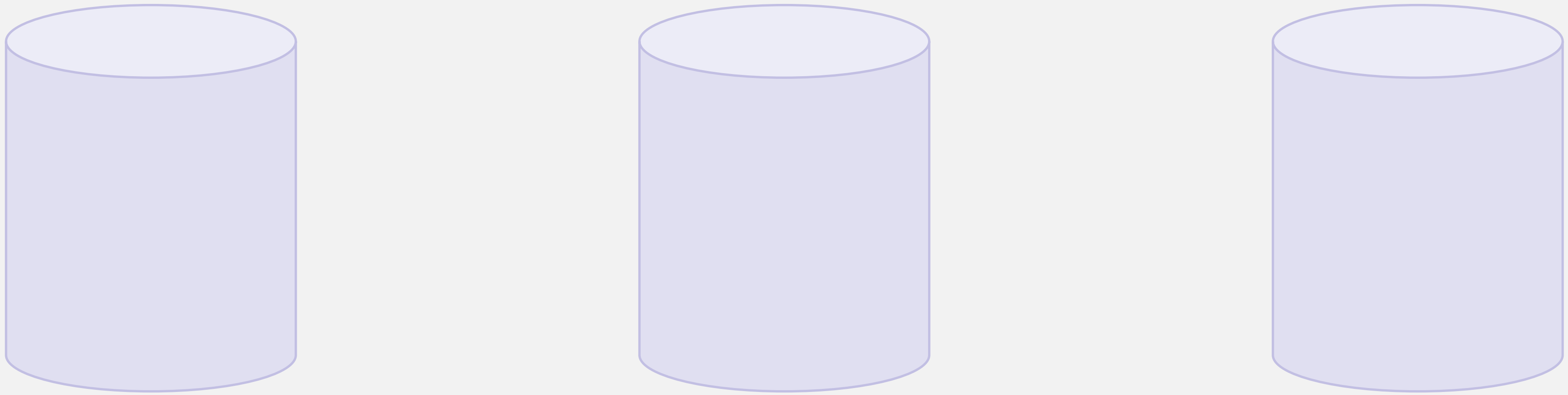
- M machines
- Synchronous rounds

What Resources to Use to Process Graph

- Multiple cores and processors
- **Communication network over machines**

Massively Parallel Computation (MPC) Model

- *M* machines
- Synchronous rounds

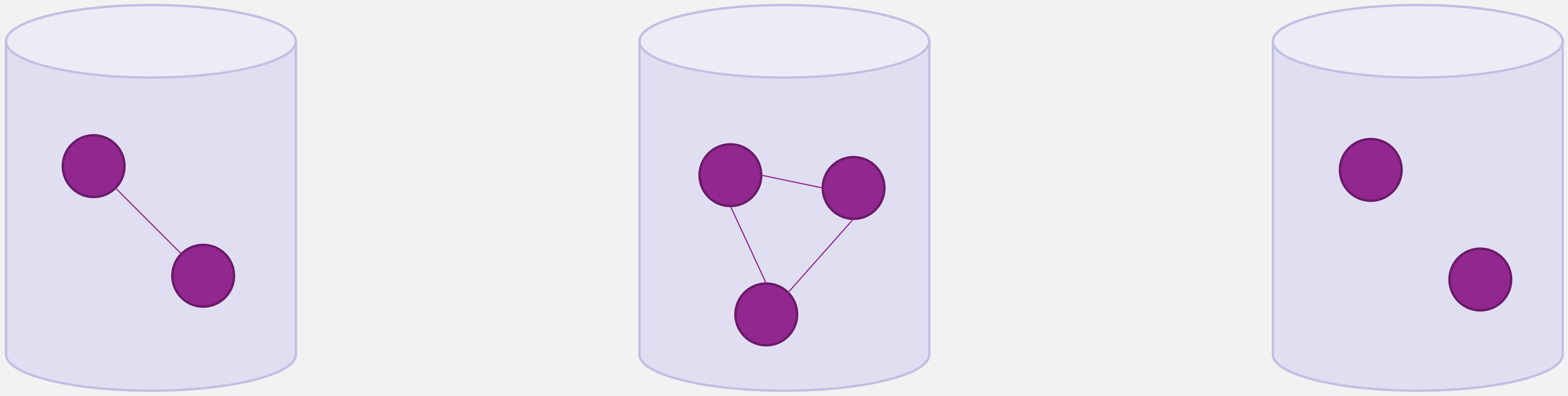


What Resources to Use to Process Graph

- Multiple cores and processors
- **Communication network over machines**

Massively Parallel Computation (MPC) Model

- M machines
- Synchronous rounds

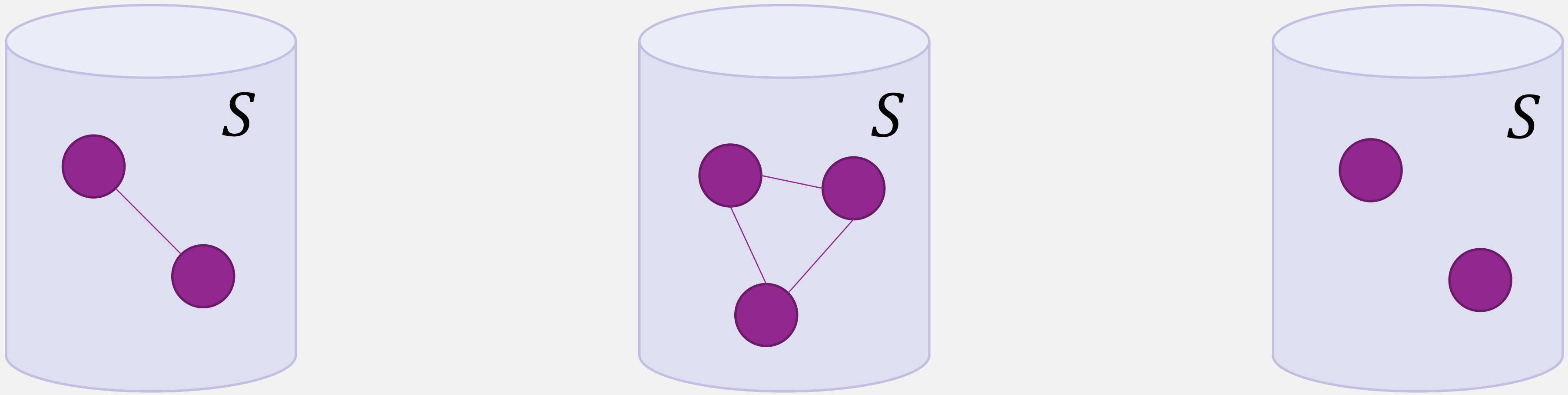


What Resources to Use to Process Graph

- Multiple cores and processors
- **Communication network over machines**

Massively Parallel Computation (MPC) Model

- M machines
- Synchronous rounds
- S space per machine

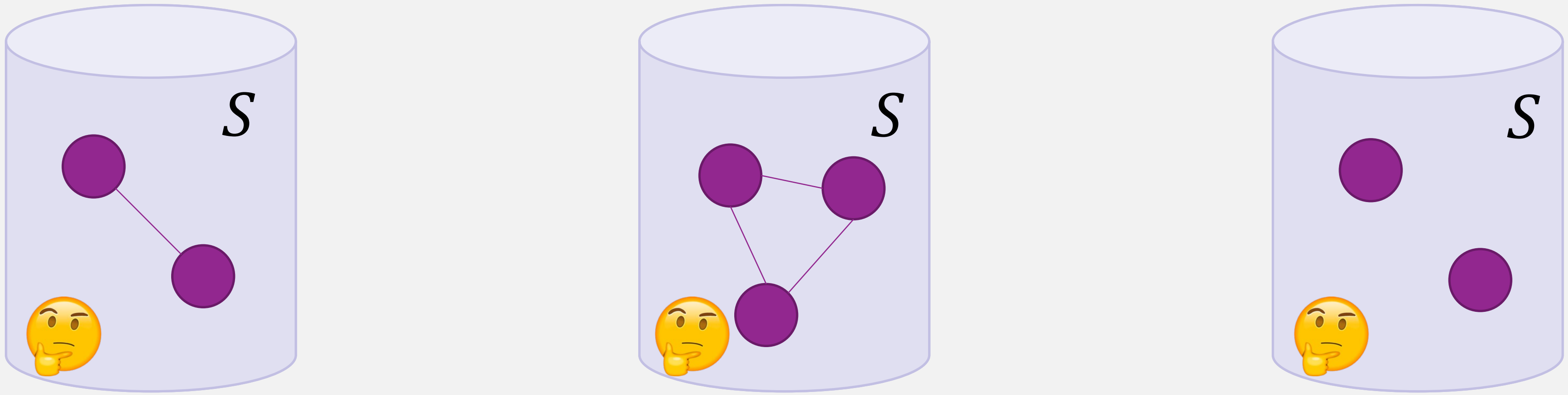


What Resources to Use to Process Graph

- Multiple cores and processors
- **Communication network over machines**

Massively Parallel Computation (MPC) Model

- M machines
- Synchronous rounds
- S space per machine

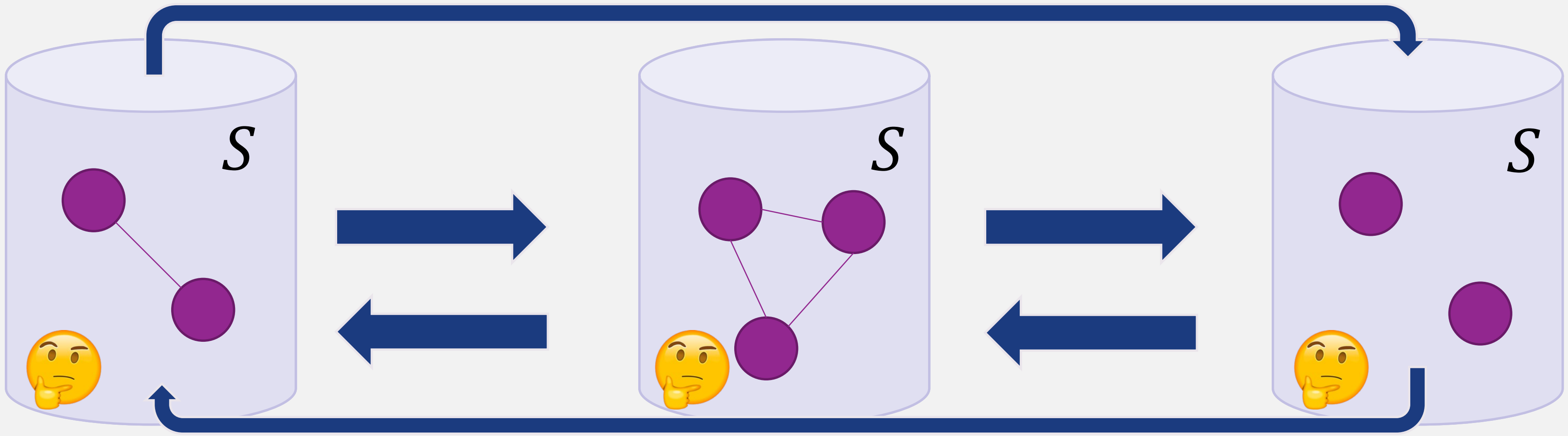


What Resources to Use to Process Graph

- Multiple cores and processors
- **Communication network over machines**

Massively Parallel Computation (MPC) Model

- M machines
- Synchronous rounds
- S space per machine



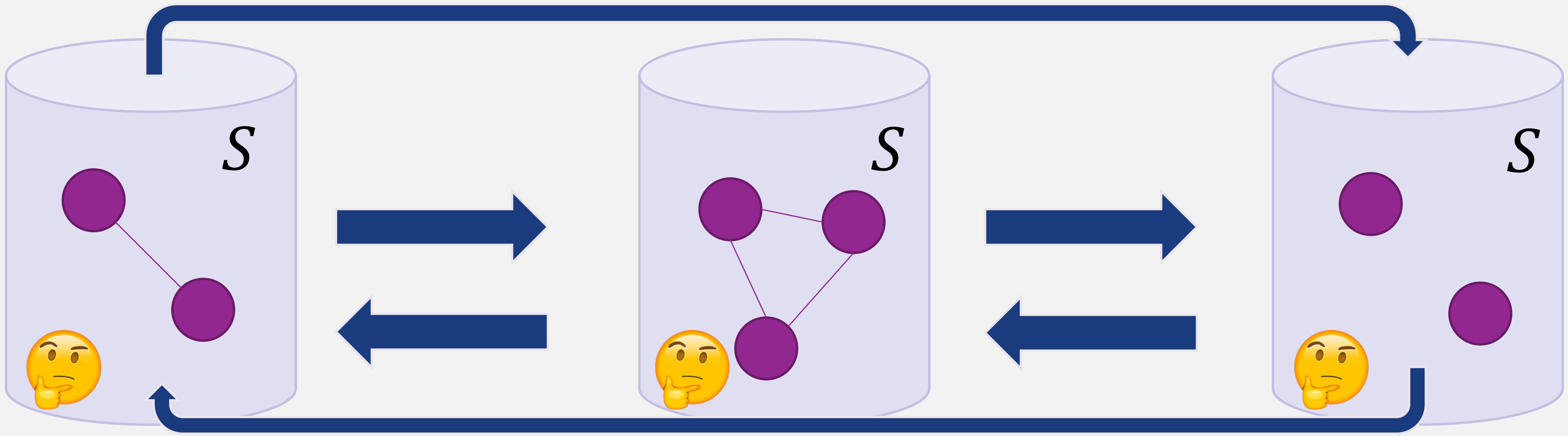
What Resources to Use to Process Graph

- Multiple cores and processors
- **Communication network over machines**

Massively Parallel Computation (MPC) Model

- M machines
- Synchronous rounds
- S space per machine

Total Space: $\boldsymbol{M \cdot S}$



What Resources to Use to Process Graph

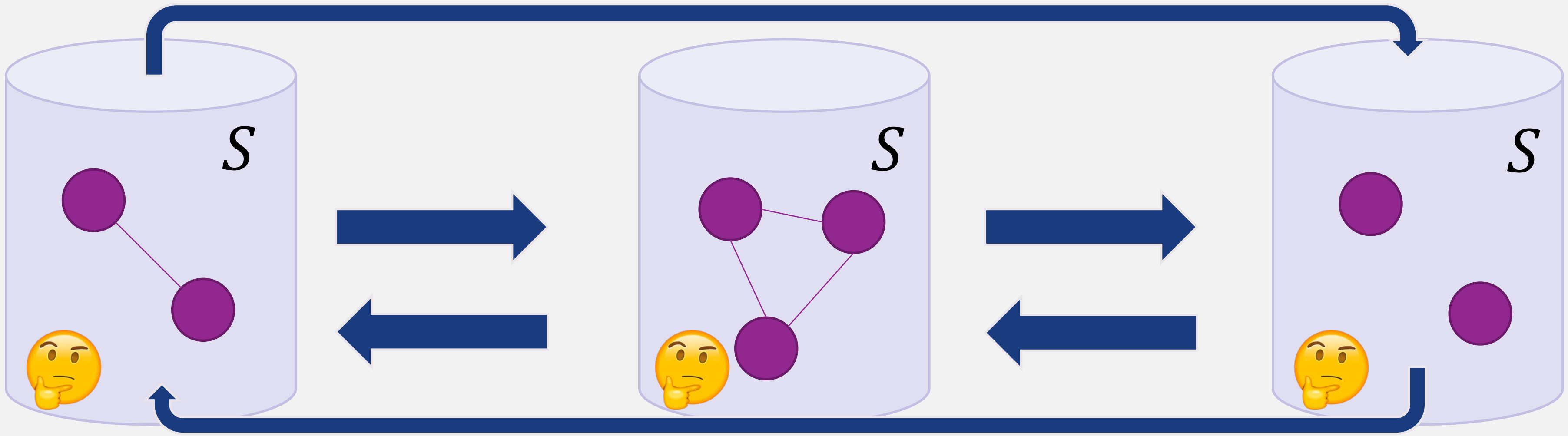
- Multiple cores and processors
- **Communication network over machines**

Complexity measures:
- **Total Space**
- **Space Per Machine**
- **Rounds of communication**

Massively Parallel Computation (MPC) Model

- *M* machines
- Synchronous rounds
- *S* space per machine

Total Space: $\boldsymbol{M \cdot S}$

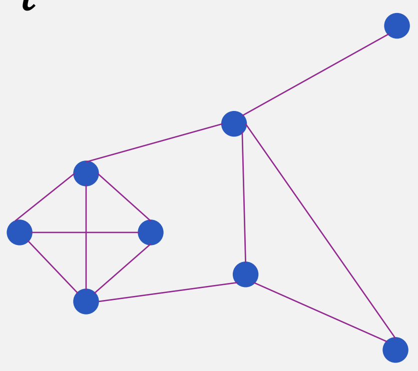


How Updates are Given

- **In batches of multiple updates**
- As a stream of updates

Batch-dynamic model

G_i

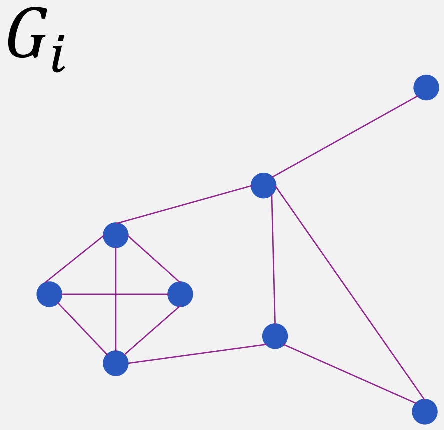


Old Graph

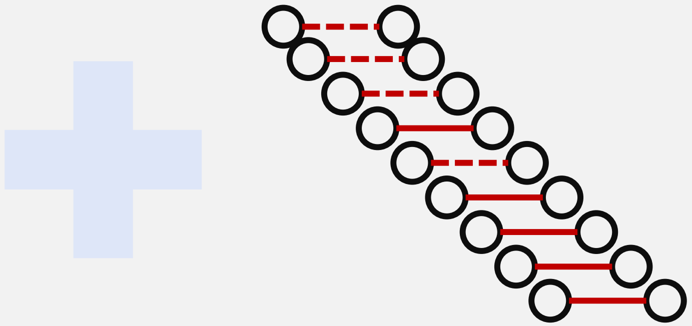
How Updates are Given

- **In batches of multiple updates**
- As a stream of updates

Batch-dynamic model



Old Graph

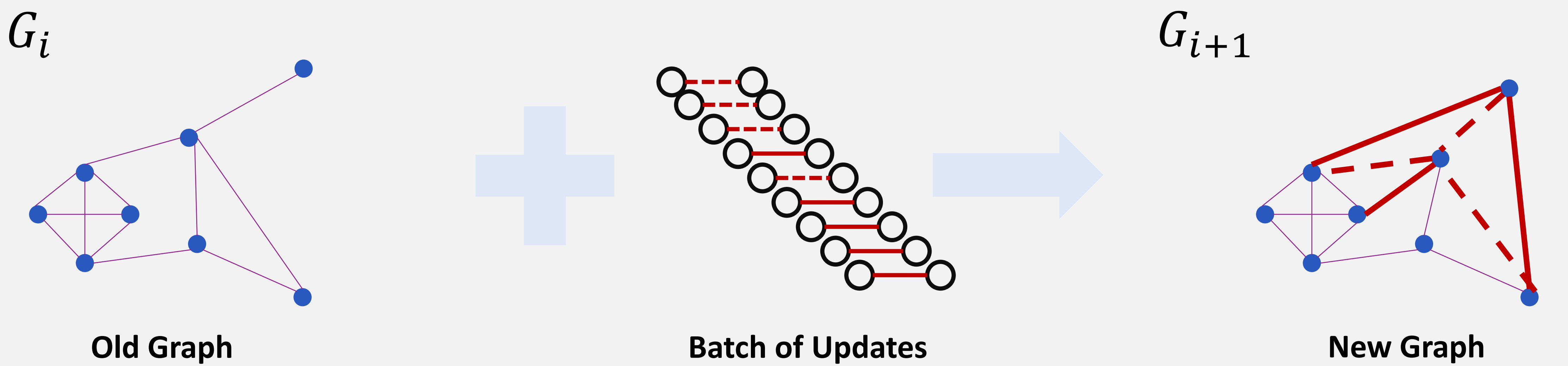


Batch of Updates

How Updates are Given

- **In batches of multiple updates**
- As a stream of updates

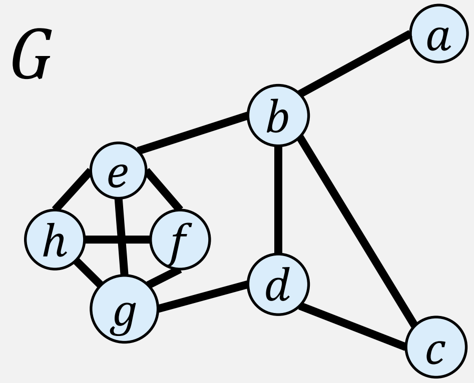
Batch-dynamic model



How Updates are Given

- In batches of multiple updates
- **As a stream of updates**

Streaming Model

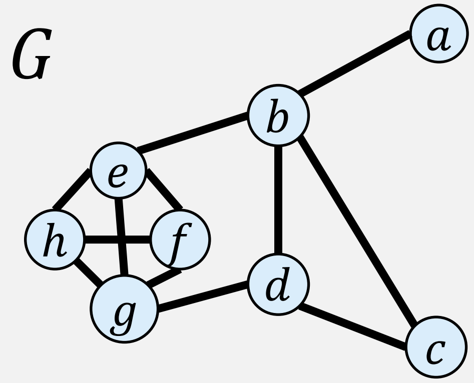


Input Graph

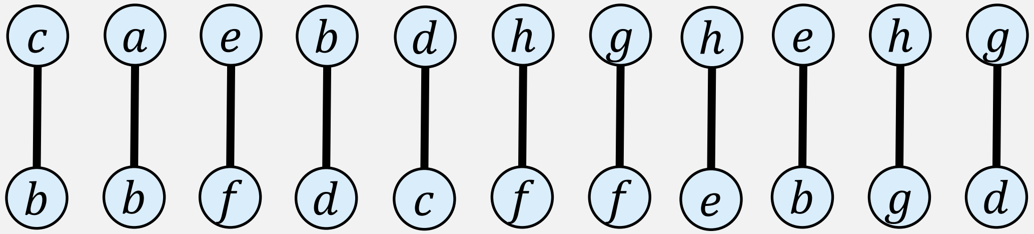
How Updates are Given

- In batches of multiple updates
- **As a stream of updates**

Streaming Model



Input Graph

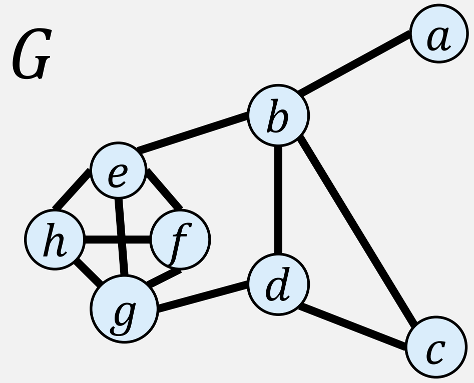


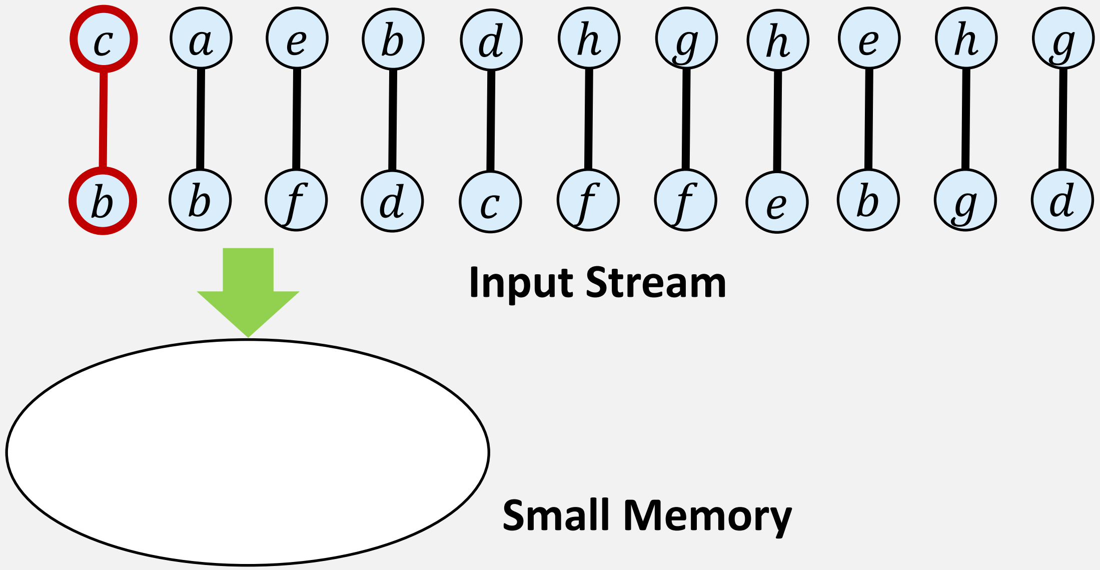
Input Stream

How Updates are Given

- In batches of multiple updates
- **As a stream of updates**

Streaming Model

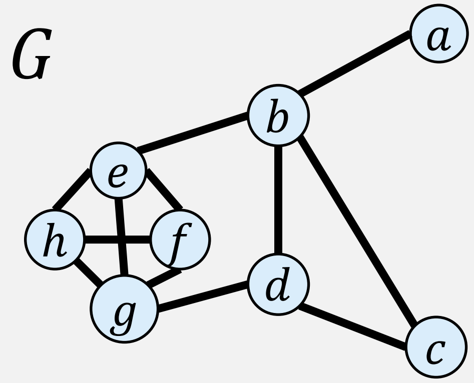




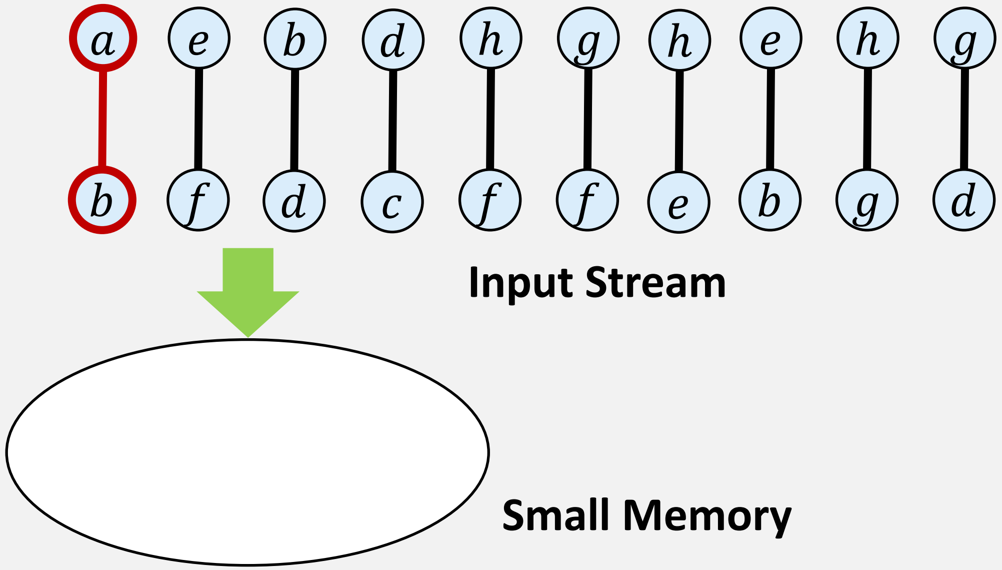
How Updates are Given

- In batches of multiple updates
- **As a stream of updates**

Streaming Model



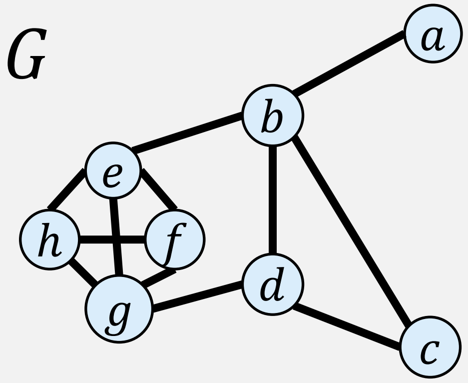
Input Graph



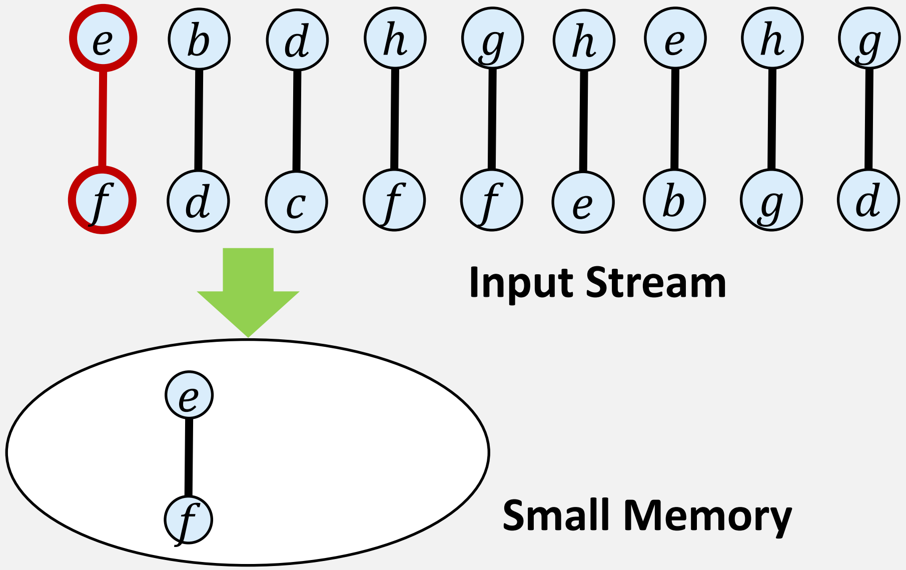
How Updates are Given

- In batches of multiple updates
- **As a stream of updates**

Streaming Model

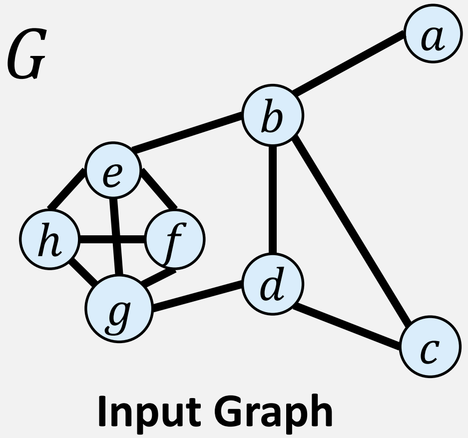


Input Graph

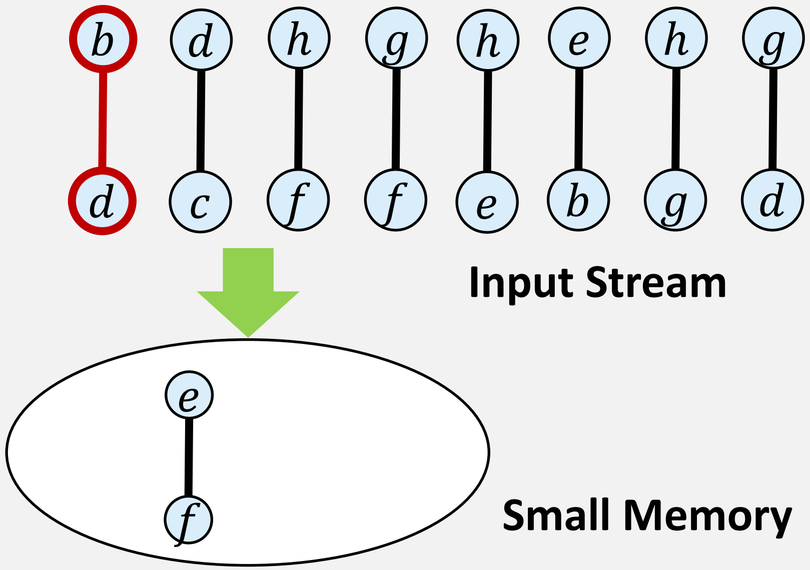


How Updates are Given

- In batches of multiple updates
- **As a stream of updates**



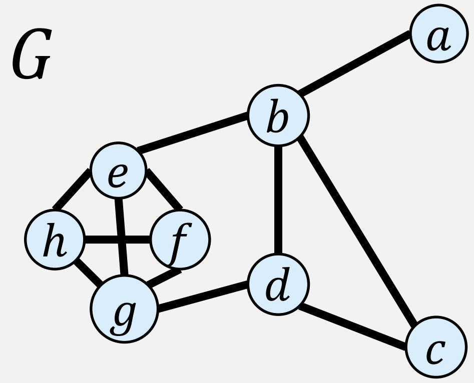
Streaming Model



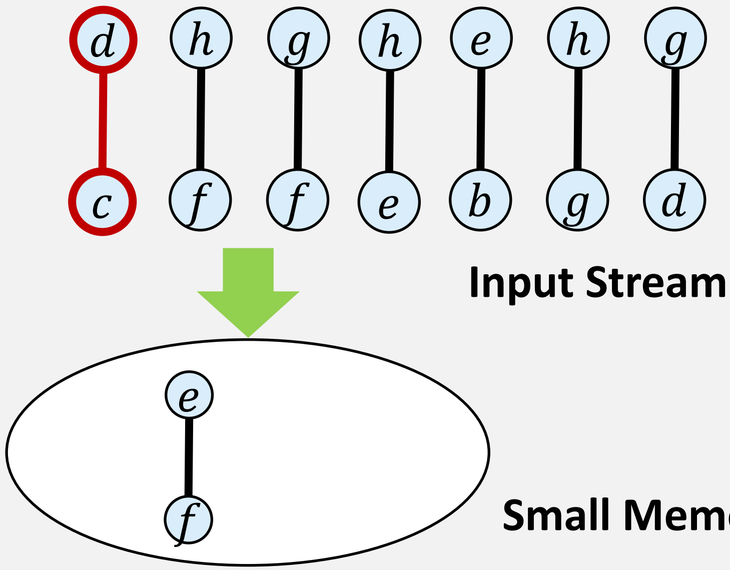
How Updates are Given

- In batches of multiple updates
- **As a stream of updates**

Streaming Model



Input Graph



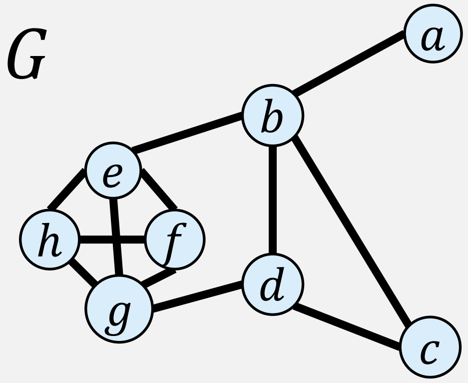
Input Stream

Small Memory

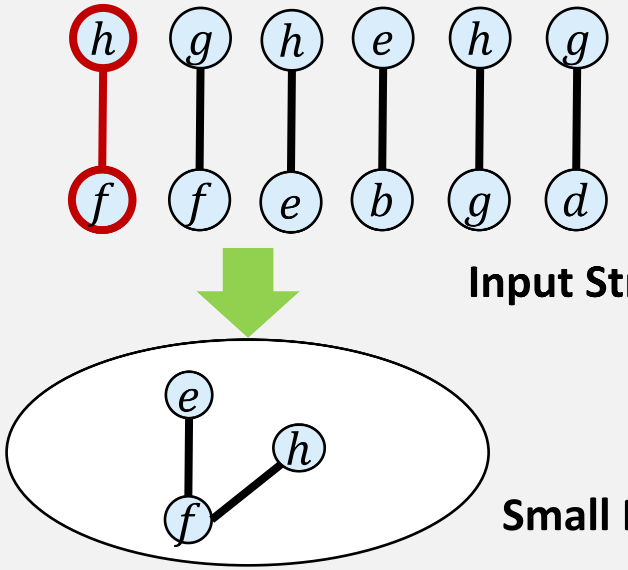
How Updates are Given

- In batches of multiple updates
- **As a stream of updates**

Streaming Model



Input Graph

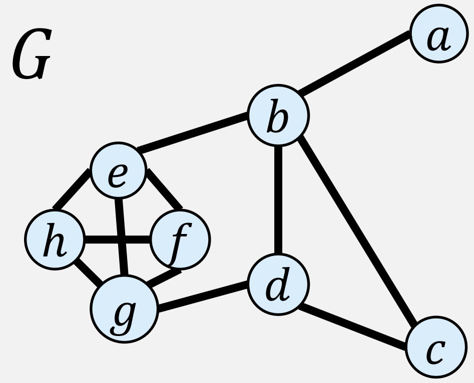


Input Stream

Small Memory

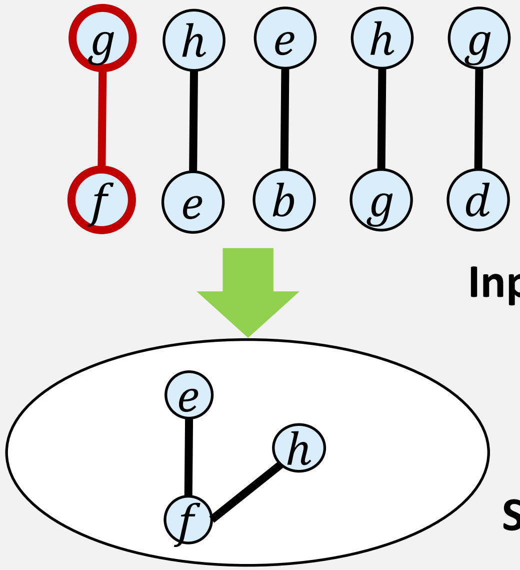
How Updates are Given

- In batches of multiple updates
- **As a stream of updates**



Input Graph

Streaming Model

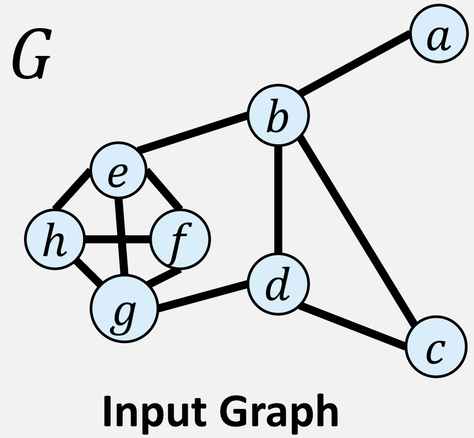


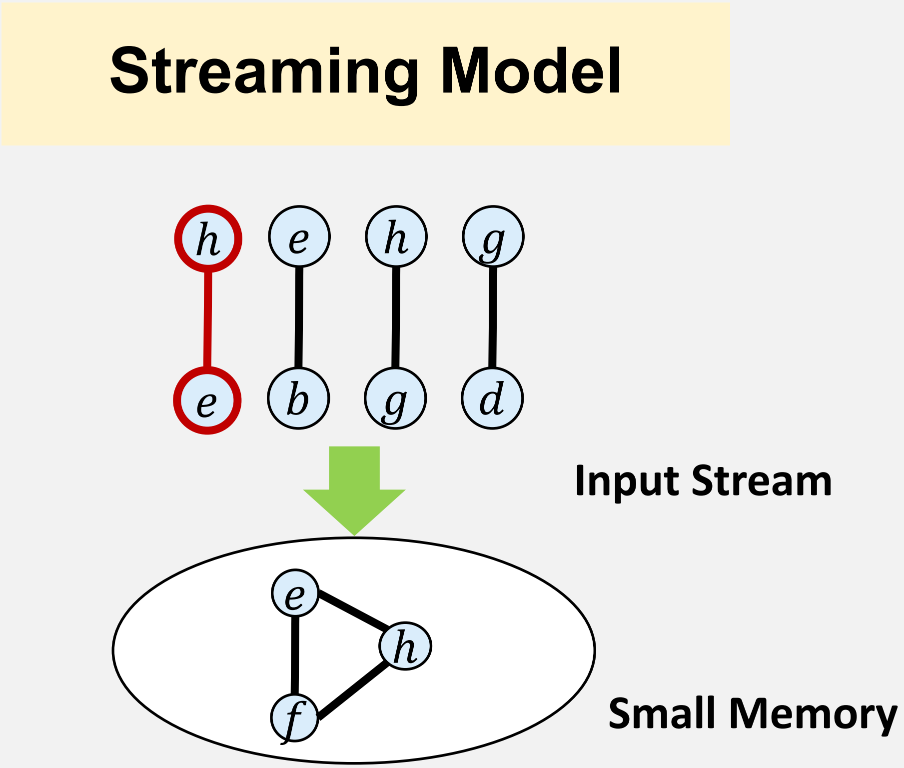
Input Stream

Small Memory

How Updates are Given

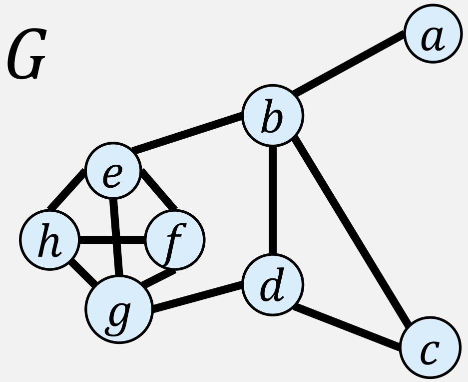
- In batches of multiple updates
- **As a stream of updates**





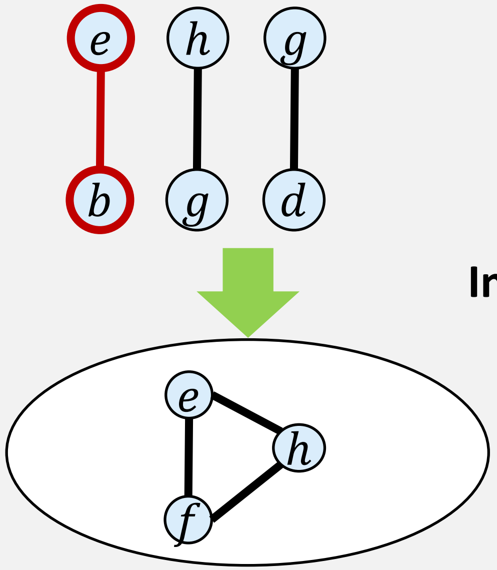
How Updates are Given

- In batches of multiple updates
- **As a stream of updates**



Input Graph

Streaming Model



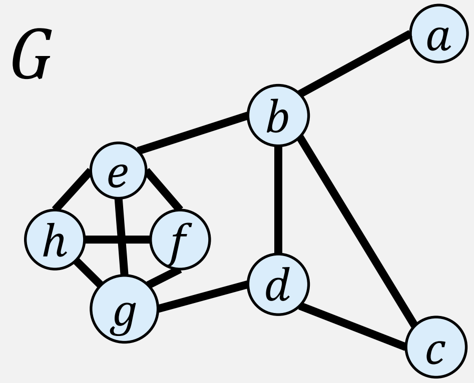
Input Stream

Small Memory

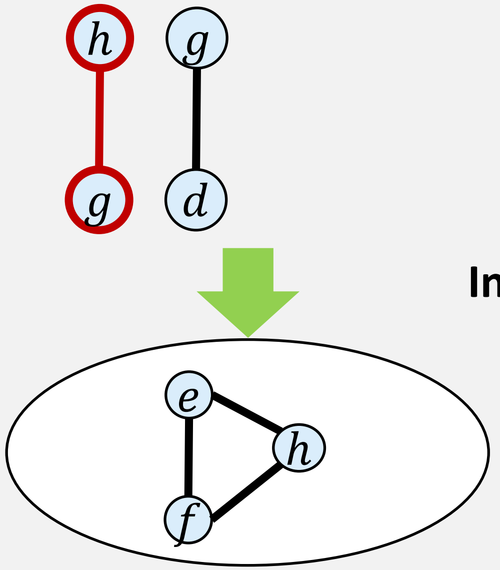
How Updates are Given

- In batches of multiple updates
- **As a stream of updates**

Streaming Model



Input Graph



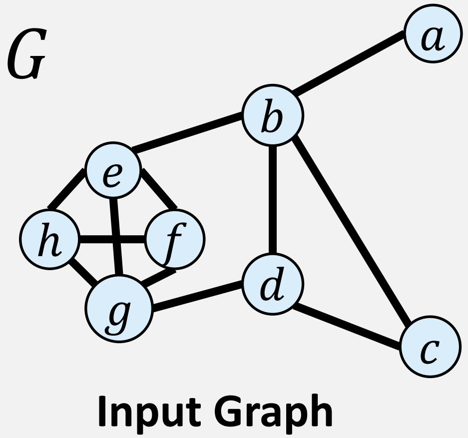
Input Stream

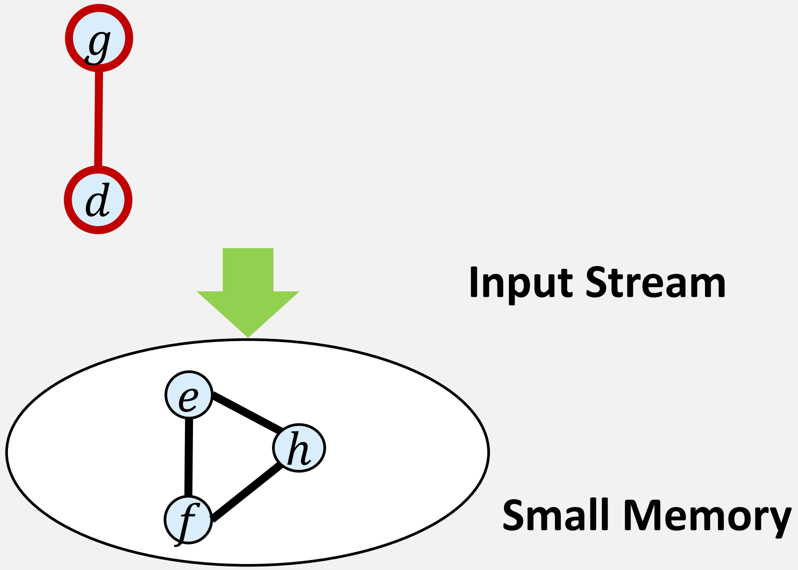
Small Memory

How Updates are Given

- In batches of multiple updates
- **As a stream of updates**

Streaming Model

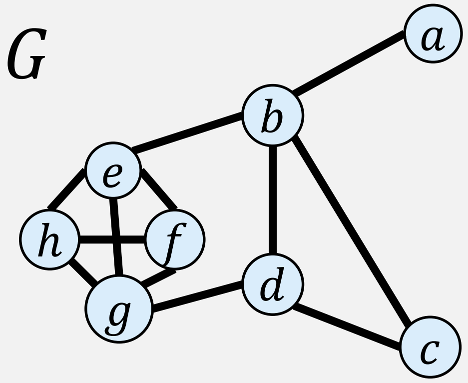




How Updates are Given

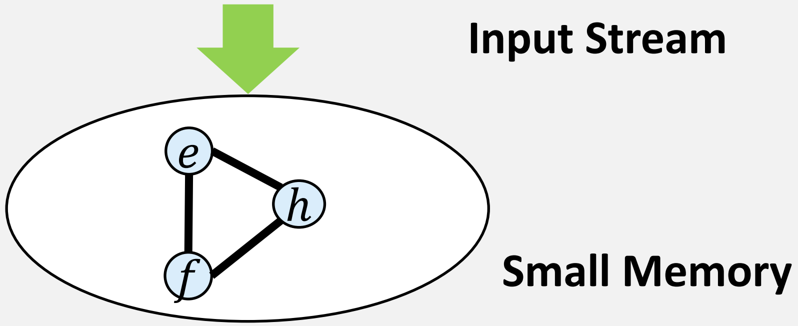
- In batches of multiple updates
- **As a stream of updates**

Streaming Model



Input Graph

Complexity measures:
- Memory size
- Number of passes



Adversarial Models

- Adaptive/oblivious
- Privacy violating

Adversarial Models

- **Adaptive/oblivious**
- Privacy violating

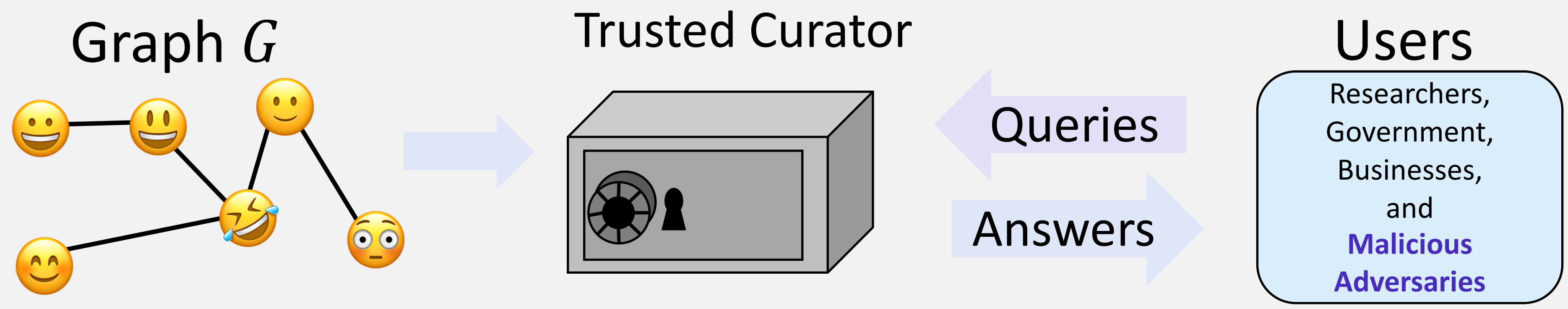
Oblivious: no knowledge on algorithm outputs
Adaptive: can see previous algorithm outputs
(sometimes internal random bits)

Adversarial Models

- Adaptive/oblivious
- **Privacy violating**

Oblivious: no knowledge on algorithm outputs
Adaptive: can see previous algorithm outputs (sometimes internal random bits)

Differential Privacy: Central Model (DP)



Adversarial Models

- Adaptive/oblivious
- **Privacy violating**

Oblivious: no knowledge on algorithm outputs
Adaptive: can see previous algorithm outputs (sometimes internal random bits)

Differential Privacy: Central Model (DP)

Differential Privacy

An algorithm $\mathcal{A}$ is **ε-differentially private** if for all pairs of neighbors G and G' and all sets of possible outputs S:

$$\Pr[\mathcal{A}(G) \in S] \leq e^{\varepsilon} \cdot \Pr[\mathcal{A}(G') \in S].$$

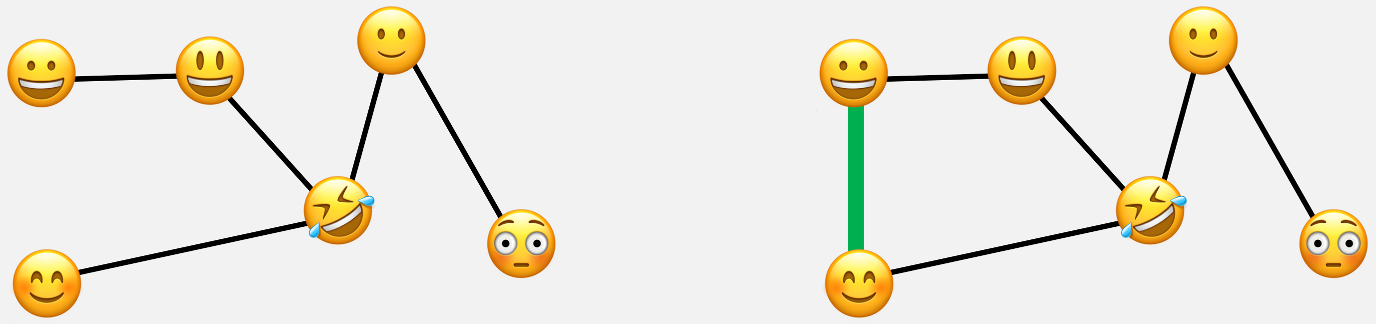
Neighboring inputs differ in some information we'd like to hide

Adversarial Models

- Adaptive/oblivious
- **Privacy violating**

Oblivious: no knowledge on algorithm outputs
Adaptive: can see previous algorithm outputs (sometimes internal random bits)

Neighboring Inputs



Edge-neighboring graphs: differ in **one edge**

Adversarial Models

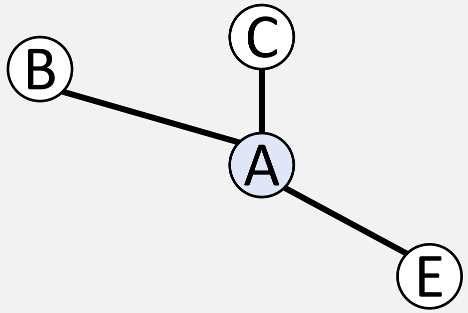
- Adaptive/oblivious
- **Privacy violating**

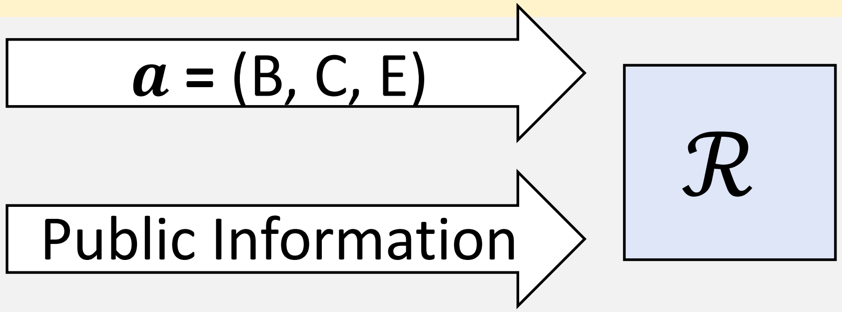
Oblivious: no knowledge on algorithm outputs
Adaptive: can see previous algorithm outputs (sometimes internal random bits)

Differential Privacy: Local Model (LDP)

Local Randomizer

An **ε-local randomizer** $\mathcal{R}$ is an ε-differentially private algorithm that takes as input an adjacency list $\boldsymbol{a}$ and public information.





Combinations of Models

Batch-Dynamic + {**Shared-memory work-depth, MPC**}

Combinations of Models

Batch-Dynamic + {**Shared-memory work-depth, MPC**}

Dynamic + **Differential Privacy** = **Continual Release**

Combinations of Models

Batch-Dynamic + {**Shared-memory work-depth, MPC**}

Dynamic + **Differential Privacy** = **Continual Release**

Distributed + **Differential Privacy** = **Local Model**

Combinations of Models

Batch-Dynamic + {Shared-memory work-depth, MPC}

Dynamic + Differential Privacy = Continual Release

Distributed + Differential Privacy = Local Model

Incremental/Fully Dynamic + Streaming

Combinations of Models

Batch-Dynamic + {**Shared-memory work-depth, MPC**}

Dynamic + **Differential Privacy** = **Continual Release**

Distributed + **Differential Privacy** = **Local Model**

Incremental/Fully Dynamic + **Streaming**

And many more…

Combinations of Models

Each model is great for a **different** real-world system!
But there are many!

Batch-Dynamic + {**Shared-memory work-depth, MPC**}

Dynamic + **Differential Privacy** = **Continual Release**

Distributed + **Differential Privacy** = **Local Model**

Incremental/Fully Dynamic + **Streaming**

And many more…

Combinations of Models

Each model is great for a **different** real-world system!
But there are many!

Batch-Dynamic + {**Shared-memory work-depth, MPC**}

Dynamic + **Differential Privacy** = **Continual Release**

Distributed + **Differential Privacy** = **Local Model**

Incremental/Fully Dynamic + **Streaming**

What problems can we study where solutions can be implemented in many models without many changes?