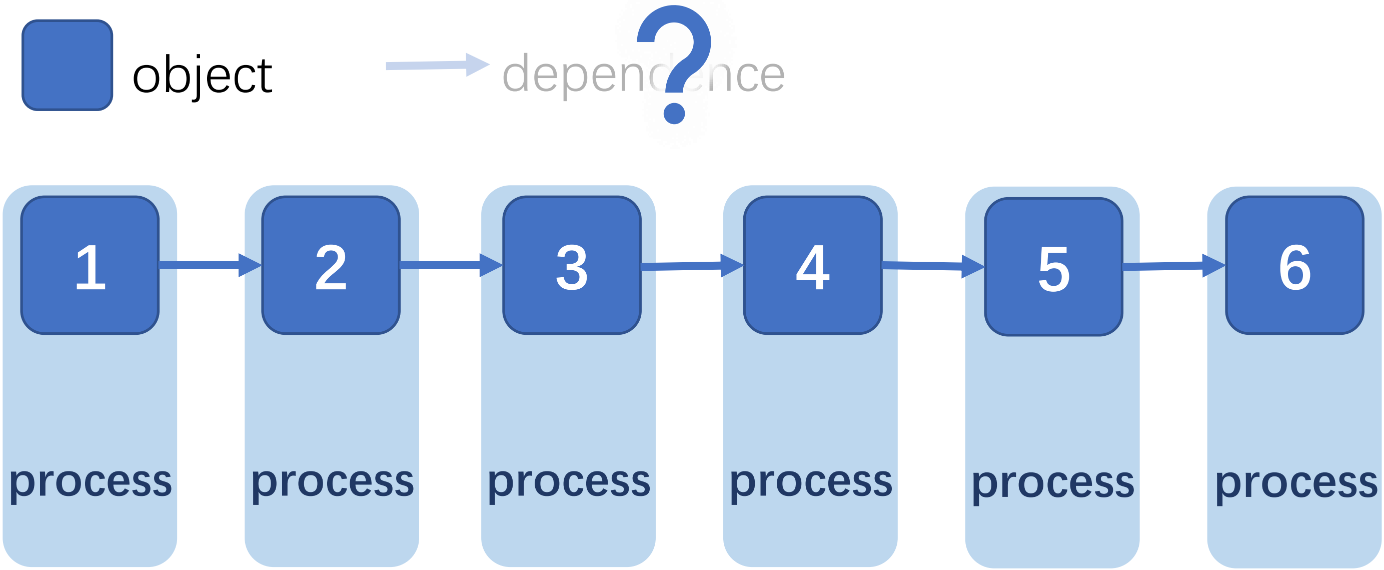
Many Sequential Iterative Algorithms Can Be Parallel and (Nearly) Work-efficient

Zheqi Shen, Zijin Wan, Yan Gu, Yihan Sun

UC Riverside

Sequential Iterative Algorithms

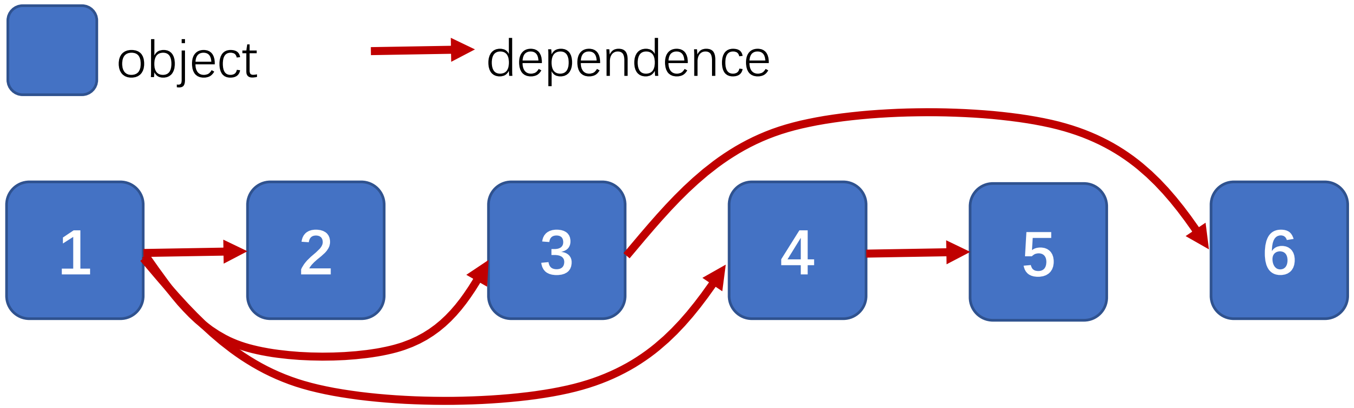
Process objects one by one in order




```
for i = 1 to n
    process object i
```

Parallelizing Sequential Iterative Algorithms

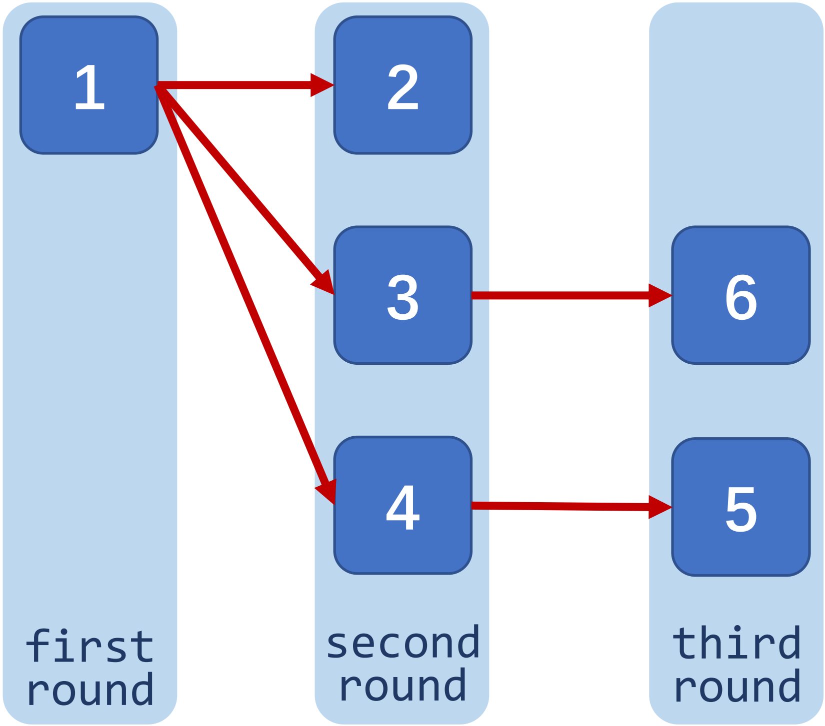
Identify the **dependences** among objects



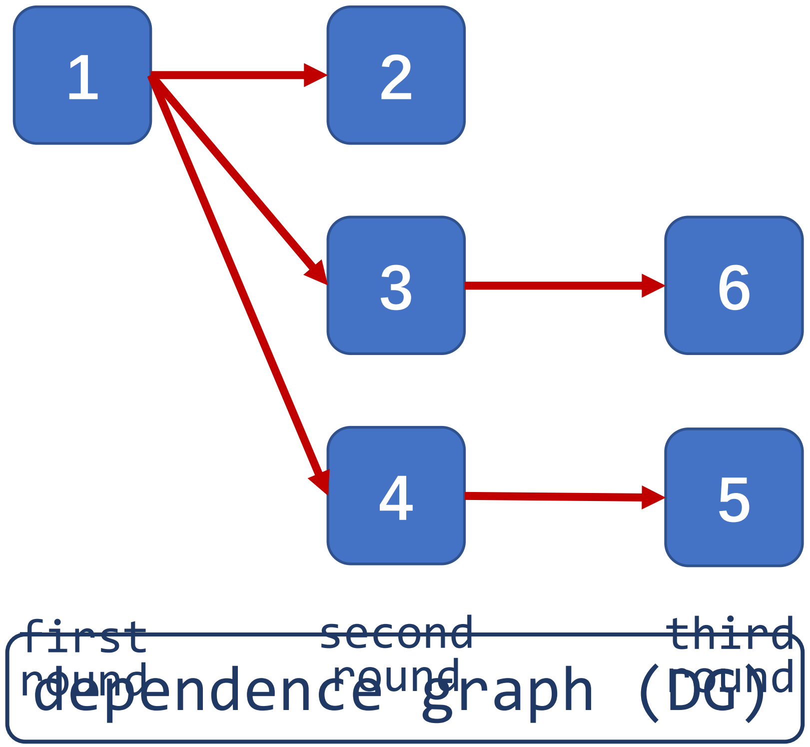

```
for i = 1 to n
    process object i
```

Parallelizing Sequential Iterative Algorithms

Obtain **parallelism**



Many Sequential Iterative Algorithms Can Be Parallel



G. Blelloch et al. (PPoPP 2012)

G. Blelloch et al. (SPAA 2012)

W. Hasenplaugh et al. (SPAA 2014)

X. Pan et al. (NIPS 2015)

J. Shun et al. (SODA 2015)

M. Fischer and A. Noever. (SODA 2018)

G. Blelloch et al. (JACM 2020)

G. Blelloch et al. (SPAA 2020)

G. Blelloch et al. (SPAA 2020)

...

Ideal Parallel Algorithm

- **Parallelize vertices as much as possible**
- **Process a vertex only when it is ready**

Ideal Parallel Algorithm

- Round-efficiency - Parallelize vertices as much as possible
- Work-efficiency - Process a vertex only when it is ready

Efficiently Parallelizing Algorithms

Analysis is based on the **binary-forking model** (with TAS)

- Work: total number of operations
- Span: length of the longest execution path

Keep the algorithms efficient

- **Round-efficiency**
- **Work-efficiency**

Efficiently Parallelizing Algorithms

Analysis is based on the **binary-forking model** (with TAS)

- Work: total number of operations
- Span: length of the longest execution path

Keep the algorithms efficient

- **Round-efficiency**: $\tilde{O}(D)$ span (D=longest path length of the given DG)
- **Work-efficiency**:

Efficiently Parallelizing Algorithms

Analysis is based on the **binary-forking model** (with TAS)

- Work: total number of operations
- Span: length of the longest execution path

Keep the algorithms efficient

- **Round-efficiency**: not equivalent to optimal span
- **Work-efficiency**:
- Near work-efficiency: important for practical performance

Some algorithms could not be parallelized efficiently

Example: Longest Increasing Subsequence (LIS)

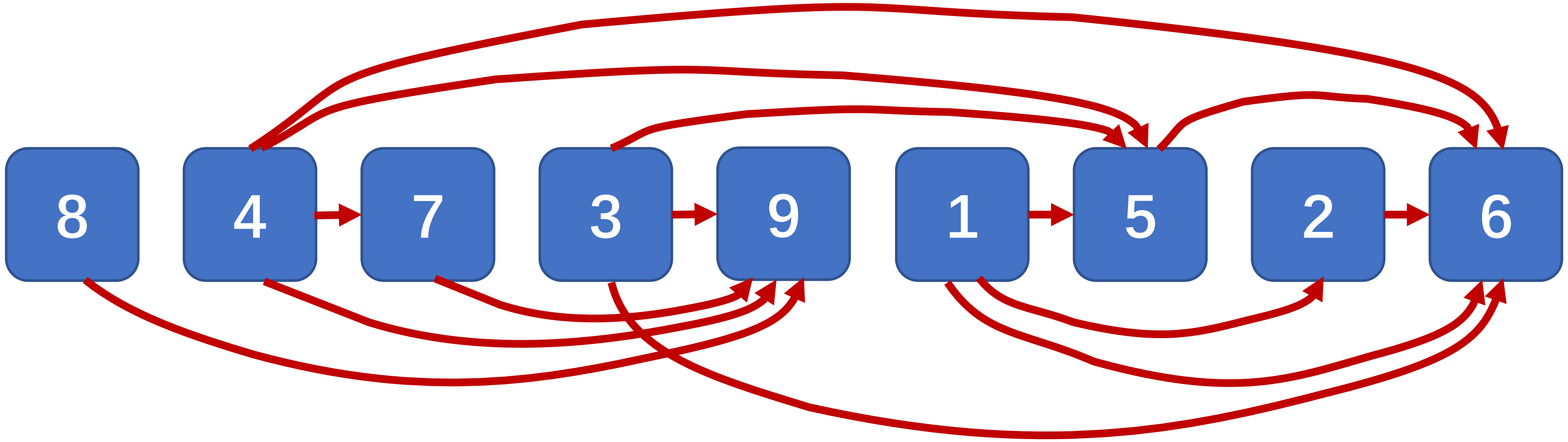
Given a sequence $s_1 \ldots s_n$

The LIS problem finds the longest subsequence s^* of s

Elements in s^* are strictly increasing

8	4	7	3	9	1	5	2	6

Example: Longest Increasing Subsequence (LIS)

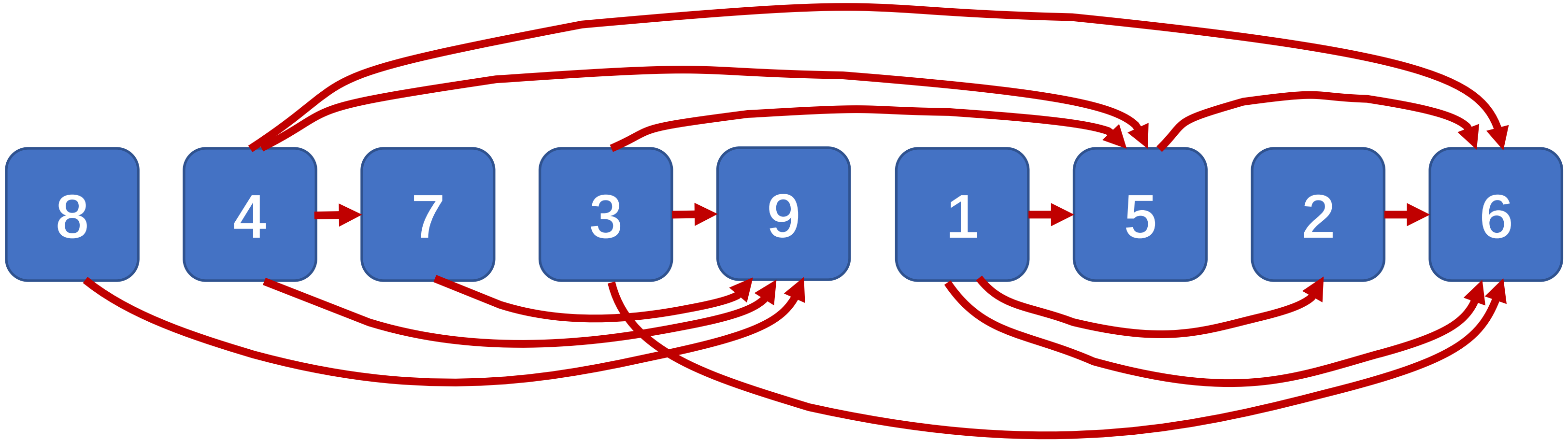


length of LIS ending with the i-th object

$$dp[i] = \max_{\boxed{j<i,\, s_j<s_i}} dp[j] + 1$$

Sequentially we can compute this in $\boldsymbol{O(n \log n)}$ cost

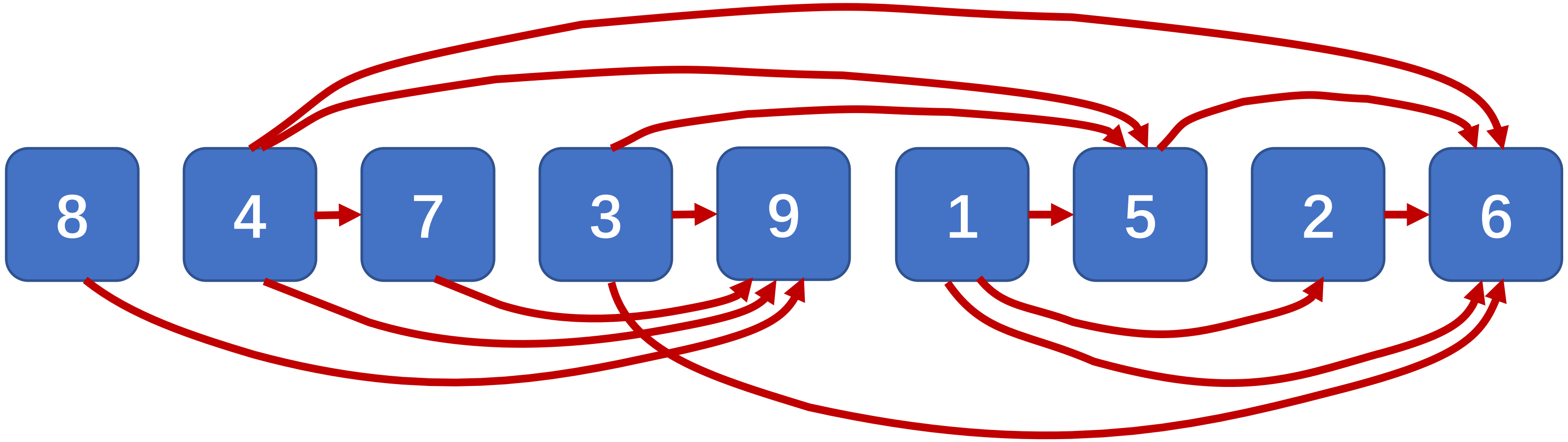
Example: Longest Increasing Subsequence (LIS)



Existing parallel algorithms are not nearly work-efficient or round efficient

- Galil et al.(Parallel Distrib 1994), Krusche, et al.(PPAM 2009), Nakashima et al. (ISPDC 2002), Semé, Thierry, et al. (ICCSA 2006), Thierry, et. al. (SPAA 2001) have $\Omega(n^{1.5})$ work
- Alam and Rahman's algorithm (IPL 2013) has $\Theta(n)$ span
- Krusche and Tiskin's algorithm (SPAA 2010) has $\tilde{O}(n \log^2 n)$ work and $\tilde{O}(n^{2/3})$ span

Example: Longest Increasing Subsequence (LIS)

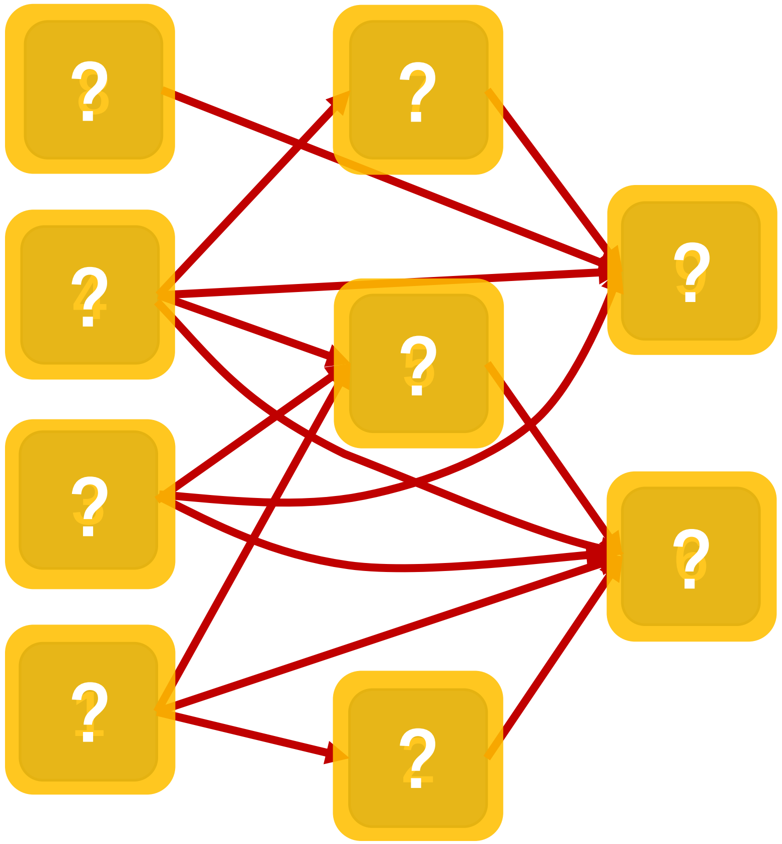


General approaches / frameworks

- Do not directly give efficient solutions to LIS

Deterministic Reservations

Proposed by Blelloch et al. (PPoPP 2012), used in algorithms from (Shun et al., SODA 2015)

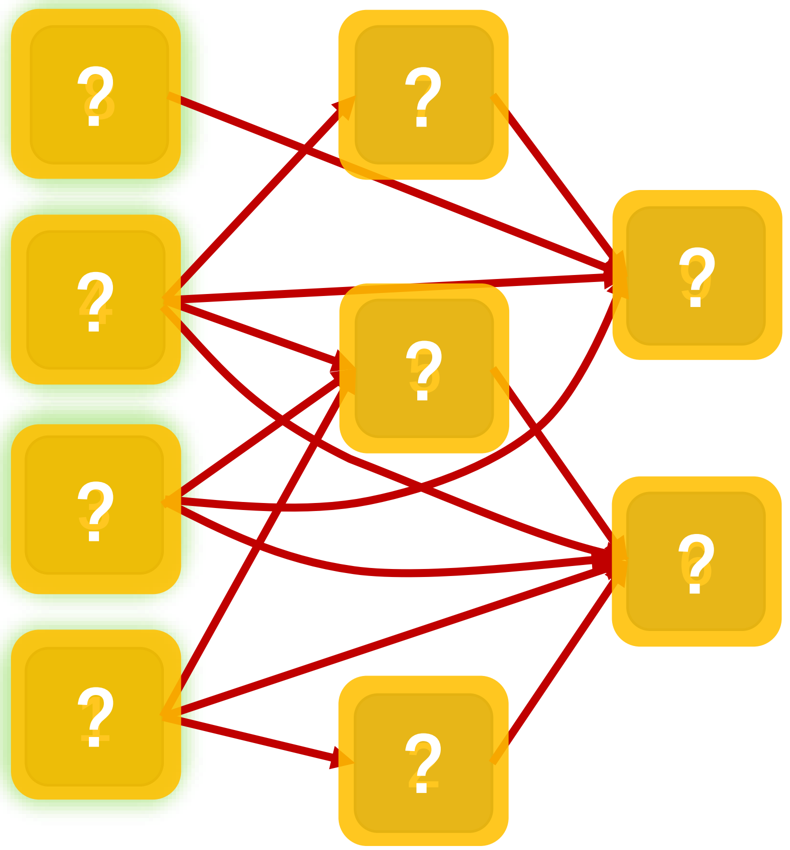


In a high level:

- Access all unprocessed objects each round

Deterministic Reservations

Proposed by Blelloch et al. (PPoPP 2012), used in algorithms from (Shun et al., SODA 2015)

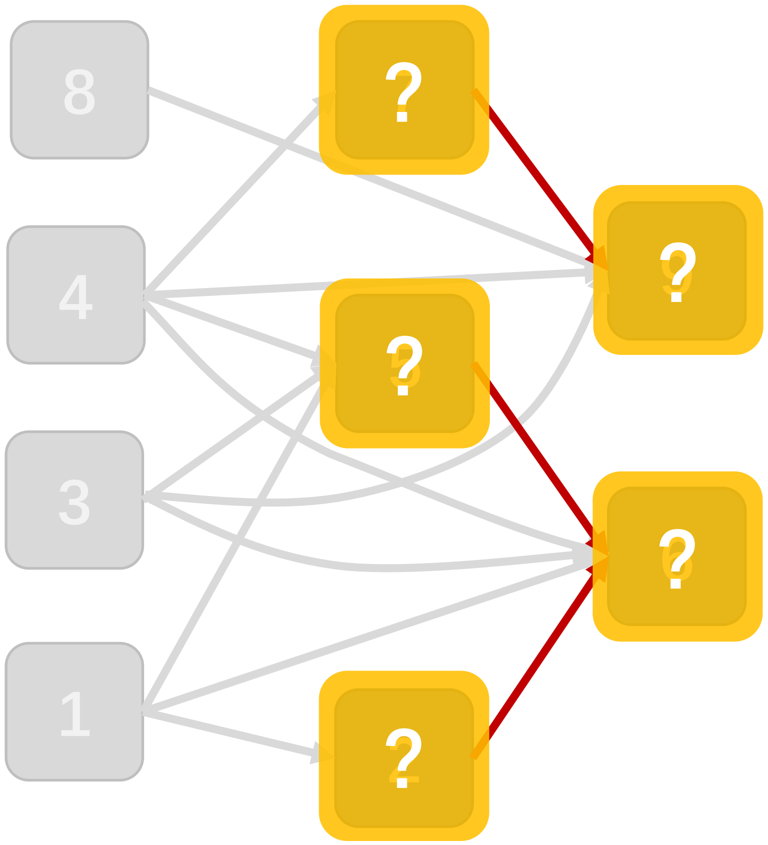


In a high level:

- Access all unprocessed objects each round
- Check their readiness

Deterministic Reservations

Proposed by Blelloch et al. (PPoPP 2012), used in algorithms from (Shun et al., SODA 2015)

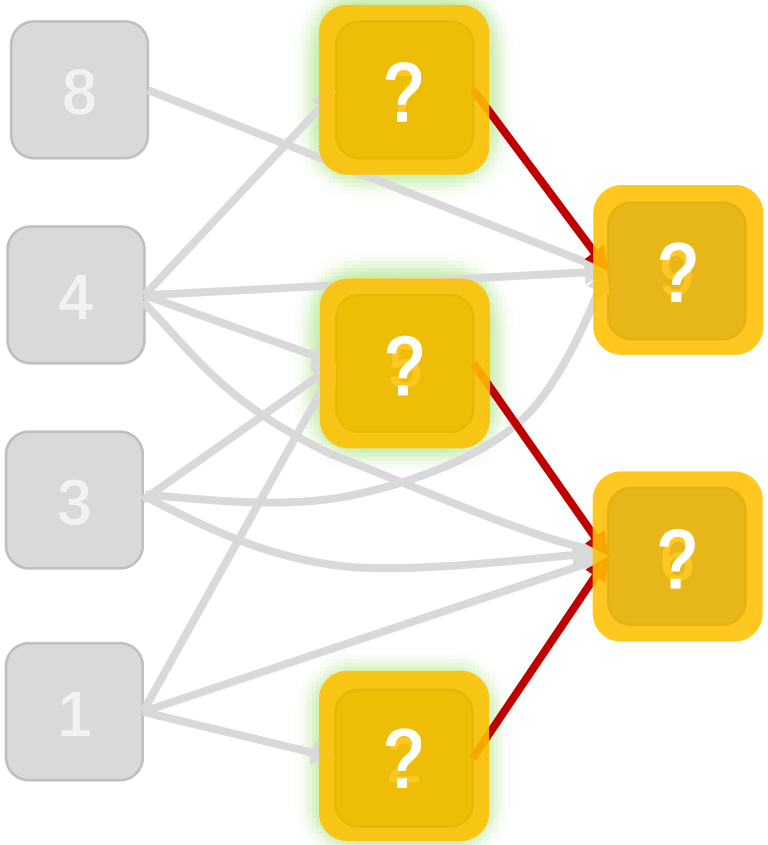


In a high level:

- Access all unprocessed objects each round
- Check their readiness
- Process the ready objects

Deterministic Reservations

Proposed by Blelloch et al. (PPoPP 2012), used in algorithms from (Shun et al., SODA 2015)

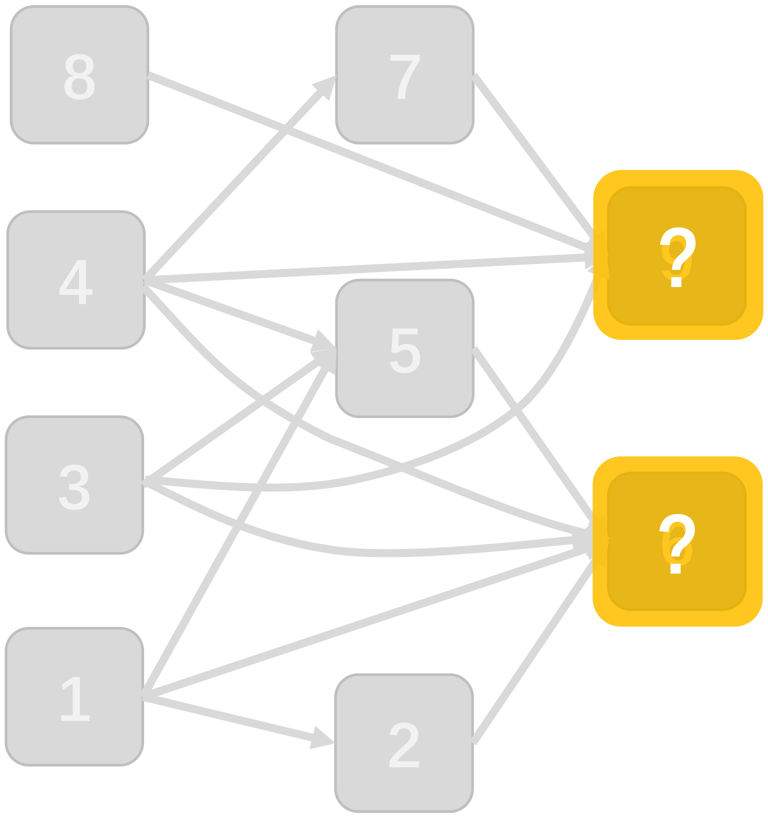


In a high level:

- Access all unprocessed objects each round
- Check their readiness
- Process the ready objects

Deterministic Reservations

Proposed by Blelloch et al. (PPoPP 2012), used in algorithms from (Shun et al., SODA 2015)

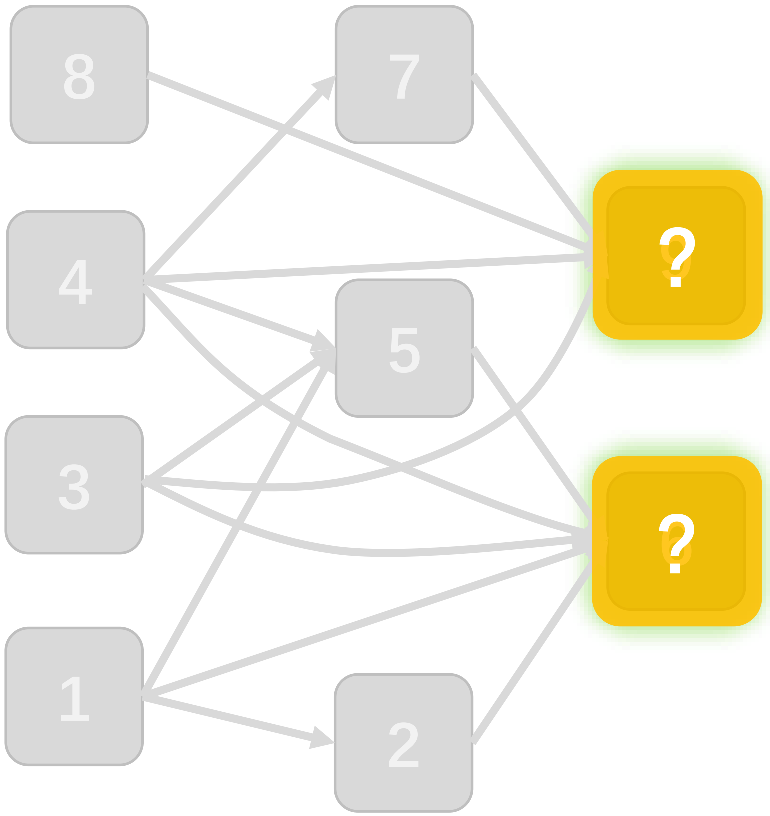


In a high level:

- Access all unprocessed objects each round
- Check their readiness
- Process the ready objects

Deterministic Reservations

Proposed by Blelloch et al. (PPoPP 2012), used in algorithms from (Shun et al., SODA 2015)

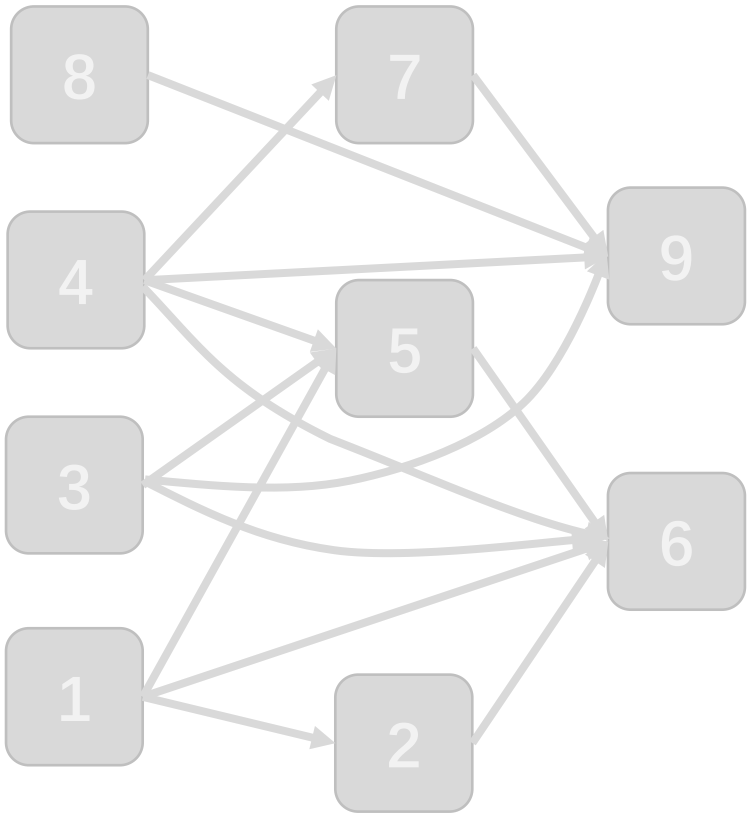


In a high level:

- Access all unprocessed objects each round
- Check their readiness
- Process the ready objects

Deterministic Reservations

Proposed by Blelloch et al. (PPoPP 2012), used in algorithms from (Shun et al., SODA 2015)



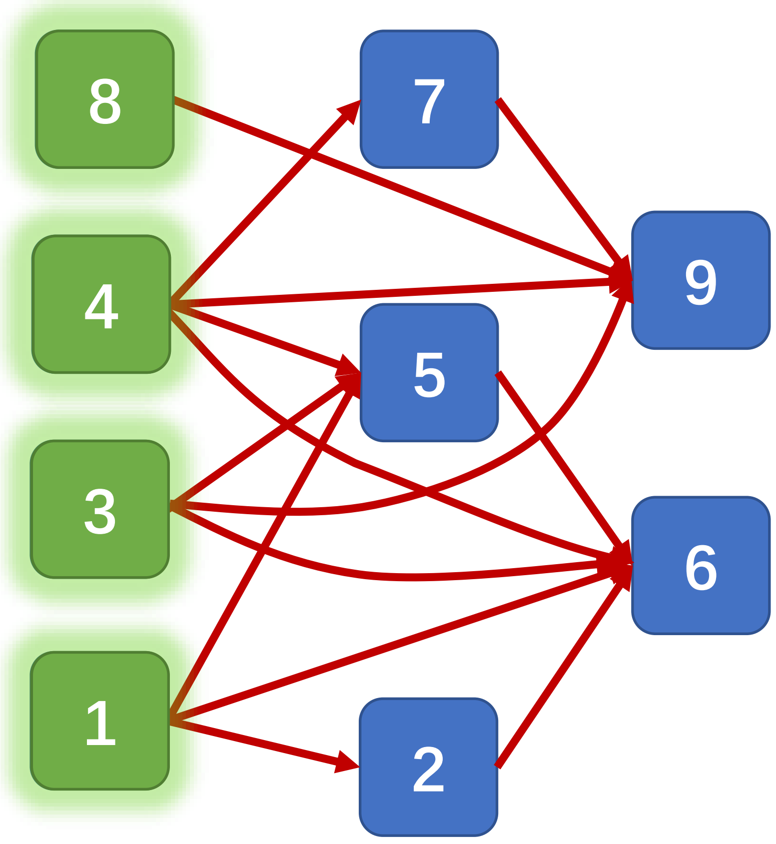
In a high level:

- Access all unprocessed objects each round
- Check their readiness
- Process the ready objects

- Work-efficient only when the work decreases geometrically in every round
- $\Theta(\mathbf{round} \cdot n)$ work for LIS
- $O(n^2)$ worst-case

Activation-based Approaches

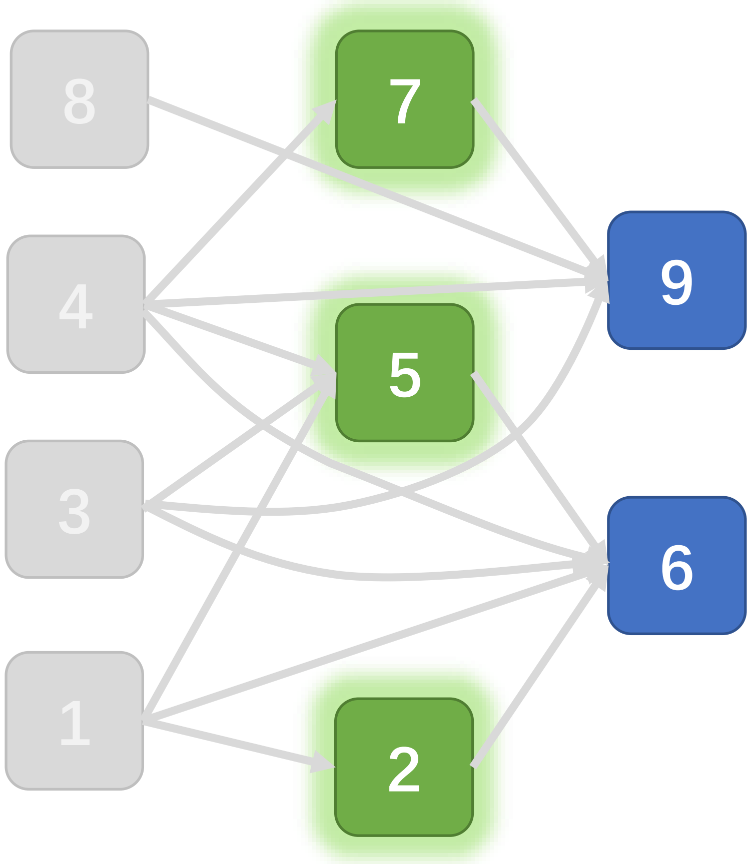
Used in Blelloch et al.(SPAA 2012, SPAA 2020), M. Fischer and A. Noever (SODA 2018) , generalized by Blelloch et al. (SPAA 2020)



- Activate some successors based on the edges in the DG

Activation-based Approaches

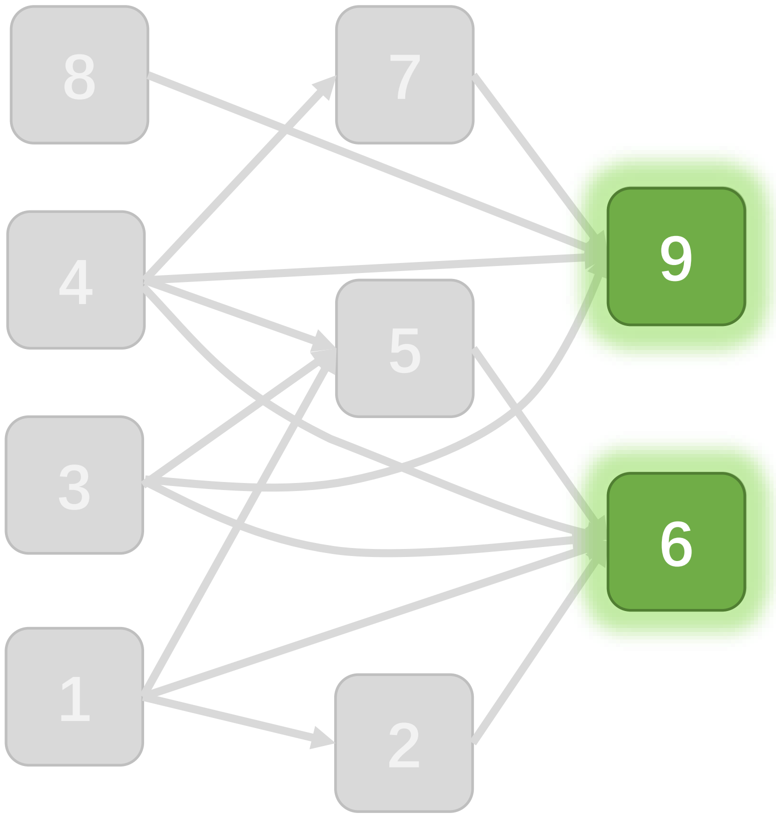
Used in Blelloch et al.(SPAA 2012, SPAA 2020), M. Fischer and A. Noever (SODA 2018) , generalized by Blelloch et al. (SPAA 2020)



- Activate some successors based on the edges in the DG
- Process the ready ones

Activation-based Approaches

Used in Blelloch et al.(SPAA 2012, SPAA 2020), M. Fischer and A. Noever (SODA 2018) , generalized by Blelloch et al. (SPAA 2020)



- Activate some successors based on the edges in the DG
- Process the ready ones

- Go through all the edges
- Take $\Theta(m)$ work for LIS (m =#edges in the DG)
- m can be up to $\Theta(n^2)$

	Cost of single readiness check	# candidates in next round
Deterministic reservations	can be fast	all the rest
Activation-based	$\Theta(\mathbf{deg})$	only successors

	Cost of single readiness check	# candidates in next round
Deterministic reservations	can be fast	all the rest
Activation-based	$\Theta(\text{deg})$	only successors

This Work

Phase-parallel framework to analyze dependences

- Core concept: $\boxed{\textbf{rank}}$
- Vertex-centric manner: avoid checking all the edges

Two general techniques to design algorithms in this framework

- Type 1 (more interesting problem in our paper)
- Type 2 (LIS problem is here)

Phase-parallel Framework

A general technique to parallelize the iterative algorithms

Formally define **rank** from the independence system $(S, \mathcal{F})$

- Feasible set: $\mathcal{F}(x) = \{E \in \mathcal{F} : E \subseteq x^{\downarrow}, x \in E\}$
- Maximum feasible set: $\mathrm{MFS}(x) = \arg\max_{E \in \mathcal{F}(x)} |E|$
- Rank: $\mathrm{rank}(x) = |\mathrm{MFS}(x)|$

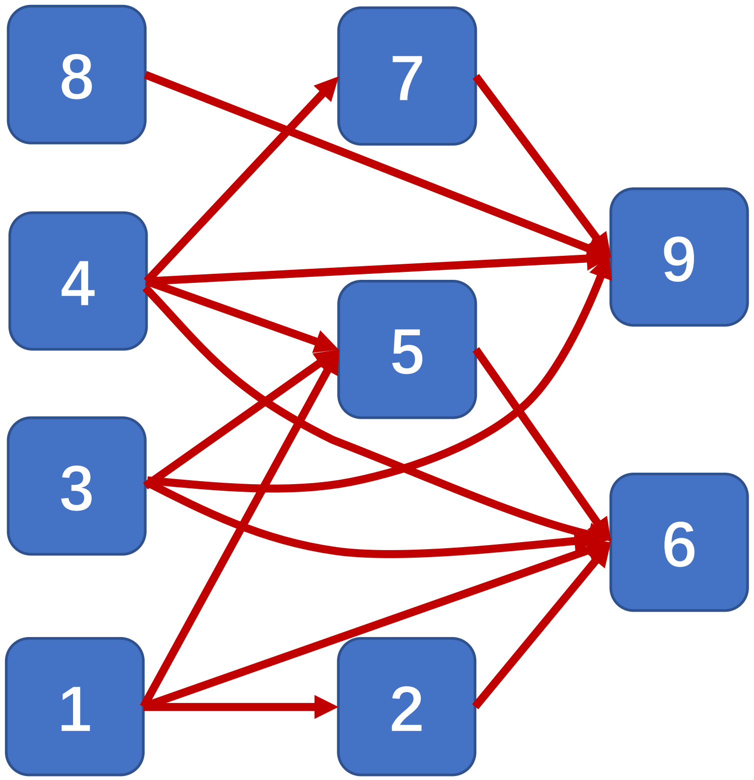
Apply to LIS

$$\mathcal{F}(x) = \{E \in \mathcal{F}: E \subseteq x^{\downarrow}, x \in E\}$$

$$\mathrm{MFS}(x) = \arg\max_{E \in \mathcal{F}(x)} |E|$$

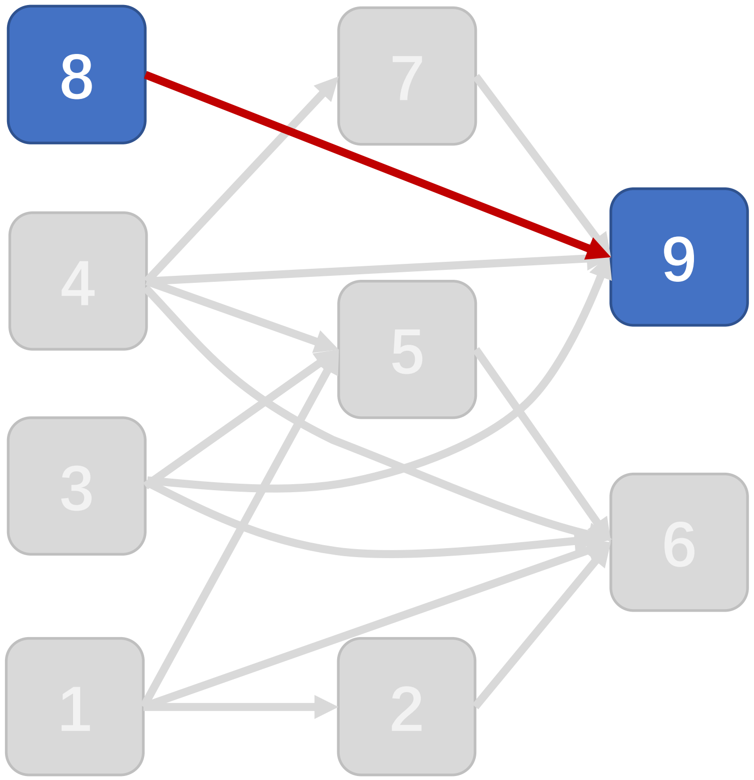
$$\mathrm{rank}(x) = |\mathrm{MFS}(x)|$$

Apply to LIS



- $\mathcal{F}(x) = \{E \in \mathcal{F}: E \subseteq x^{\downarrow}, x \in E\}$
- $\mathrm{MFS}(x) = \arg\max_{E \in \mathcal{F}(x)} |E|$
- $\mathrm{rank}(x) = |\mathrm{MFS}(x)|$

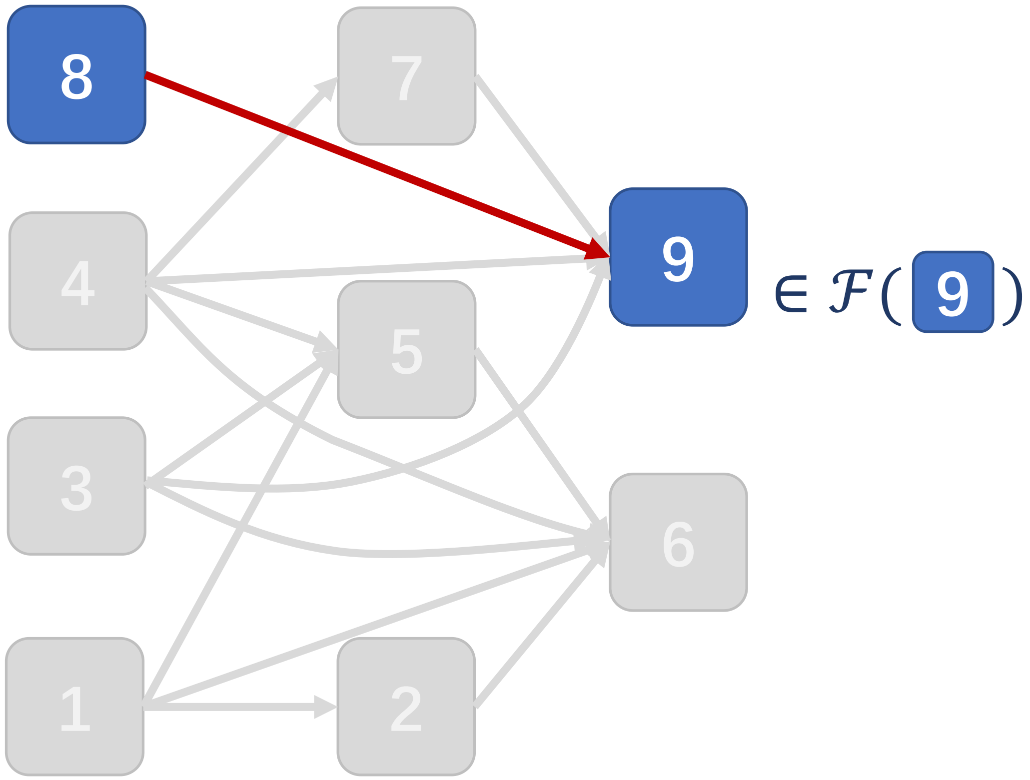
Apply to LIS



Given an object x

- $\mathcal{F}(x) = \{E \in \mathcal{F}: E \subseteq x^{\downarrow}, x \in E\}$
- $\mathrm{MFS}(x) = \arg\max_{E \in \mathcal{F}(x)} |E|$
- $\mathrm{rank}(x) = |\mathrm{MFS}(x)|$

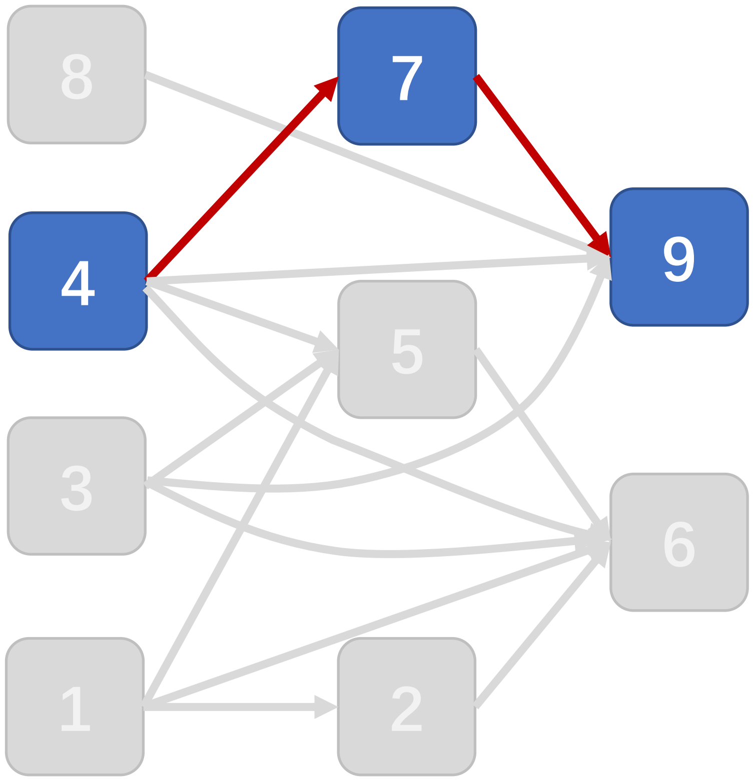
Apply to LIS



Given an object x

- $\mathcal{F}(x)$ increasing subsequence ending with object x
- $\mathrm{MFS}(x) = \arg\max_{E \in \mathcal{F}(x)} |E|$
- $\mathrm{rank}(x) = |\mathrm{MFS}(x)|$

Apply to LIS

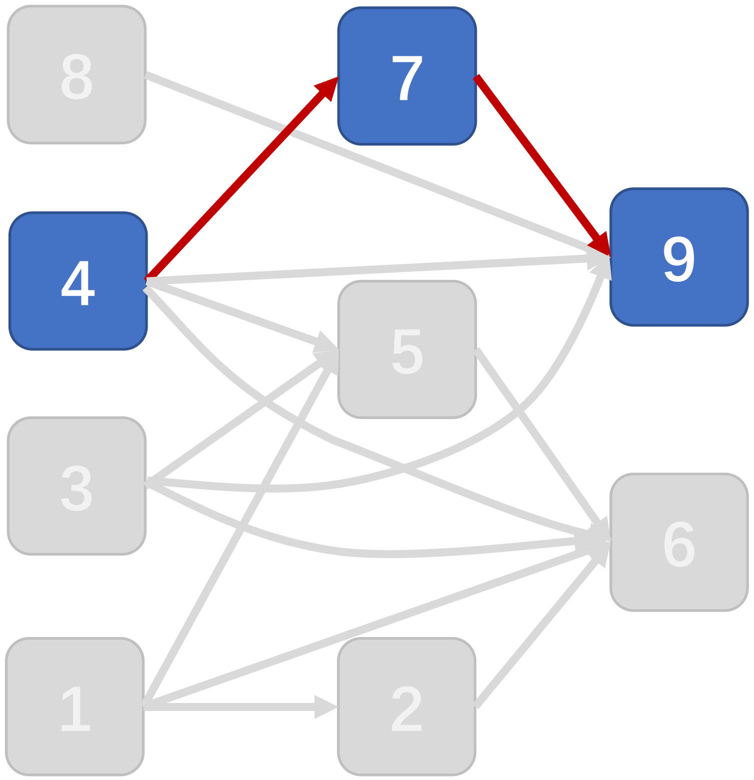


$= \mathrm{MFS}(\boxed{9})$

Given an object x

- $\mathcal{F}(x)$ — increasing subsequence ending with object x
- $\mathrm{MFS}(x)$ — LIS ending with object x
- $\mathrm{rank}(x) = |\mathrm{MFS}(x)|$

Apply to LIS

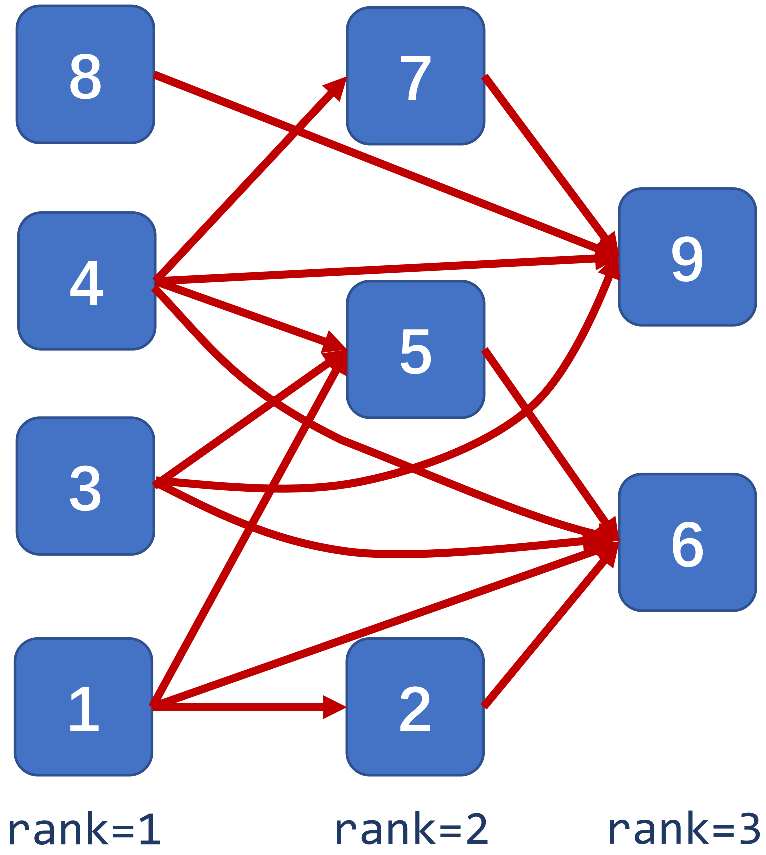


rank(9)=3

Given an object x

- $\mathcal{F}(x)$ — increasing subsequence ending with object x
- $\mathrm{MFS}(x)$ — LIS ending with object x
- $\mathrm{rank}(x)$ — the length of LIS

Apply to LIS



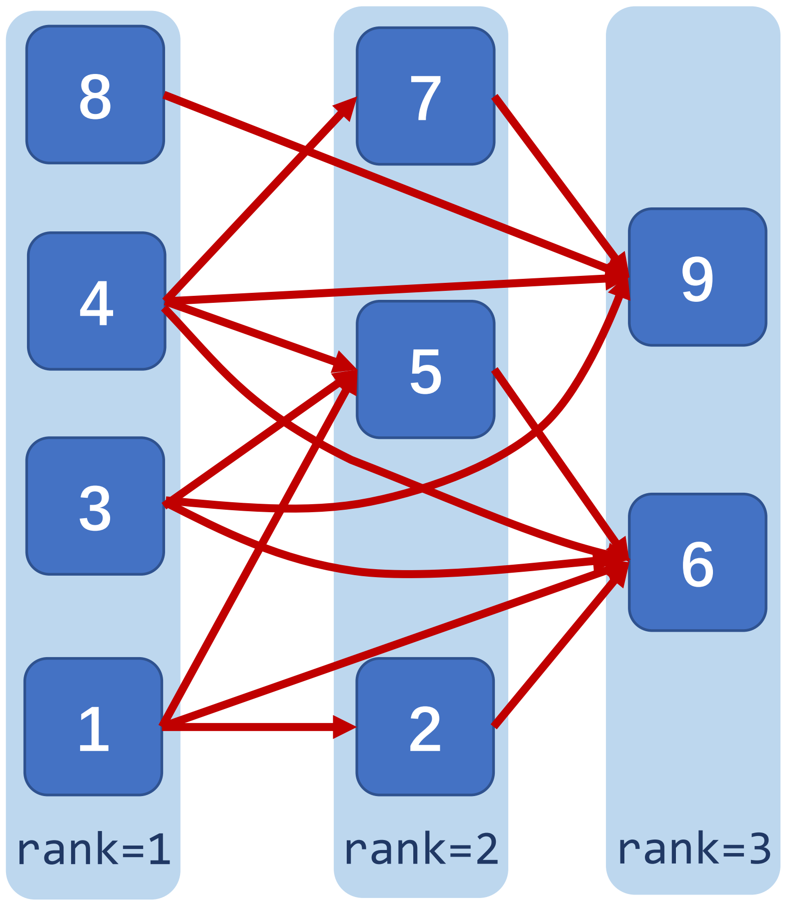
Given an object x

- $\mathcal{F}(x)$ — increasing subsequence ending with object x
- $\mathrm{MFS}(x)$ — LIS ending with object x
- $\mathrm{rank}(x)$ — the length of LIS

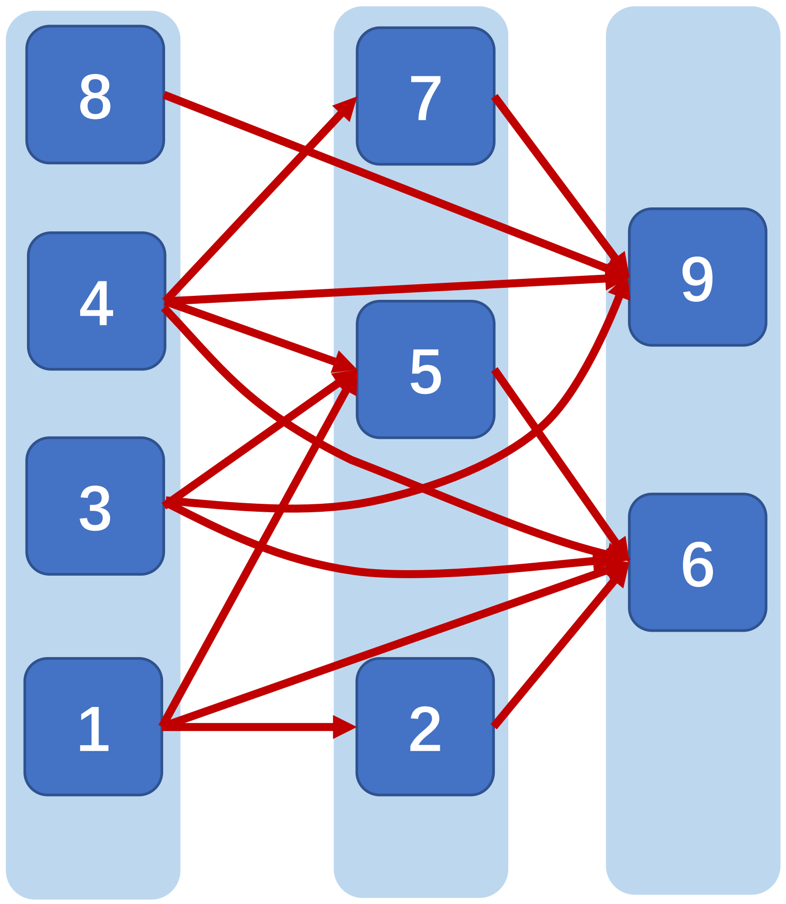
The **earliest "phase"** when an object is ready

With certain conditions, the **rank** of an object is its depth in DG

Processes objects in the order of ranks



Processes objects in the order of ranks



Challenge

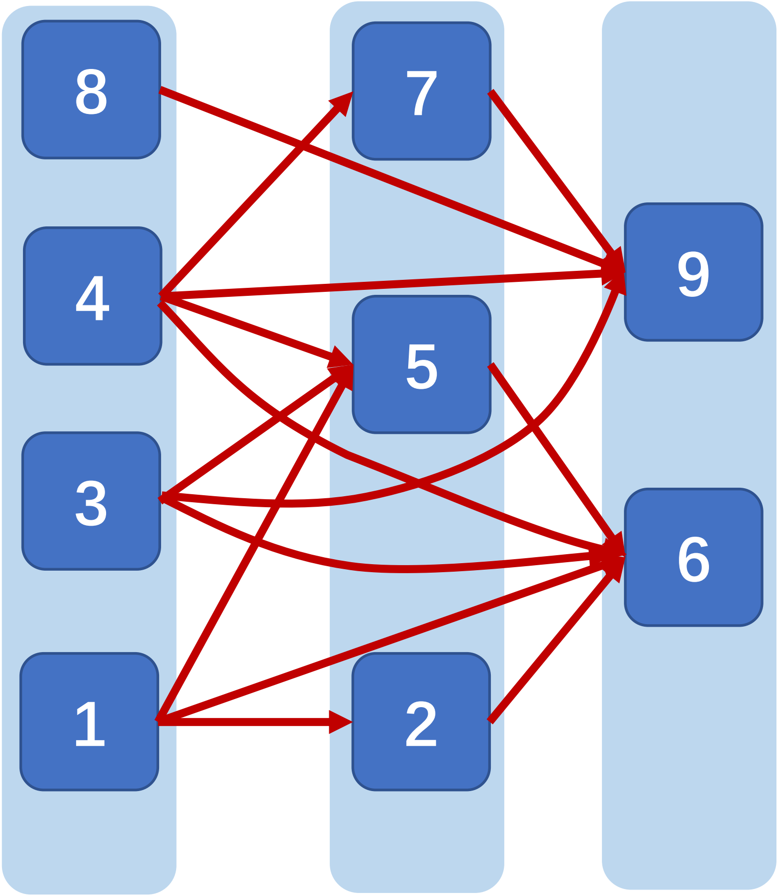
Find the ready objects in each round

Avoid visiting all edges

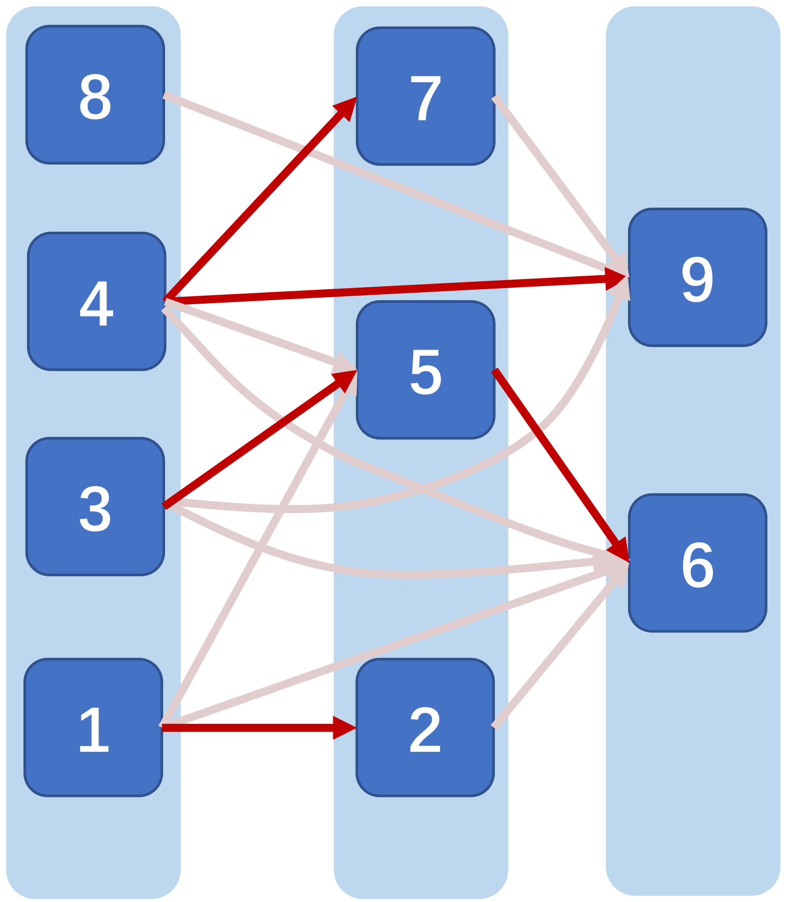
Solution

“Vertex-centric” approach

Core idea: select a "pivot" for every object



Core idea: select a "pivot" for every object

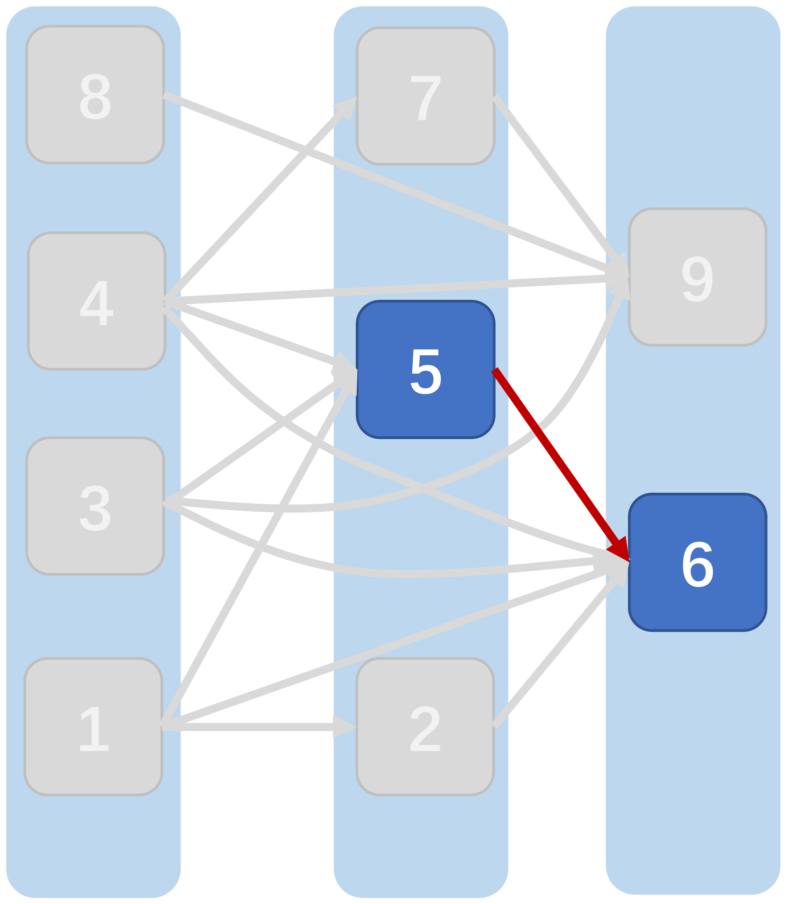


Pivot: an unfinished predecessor selected at uniformly random

If the pivot of an object hasn't finished, this object cannot be ready

We check readiness of an object **only when its pivot finishes**

Core idea: select a “pivot” for every object

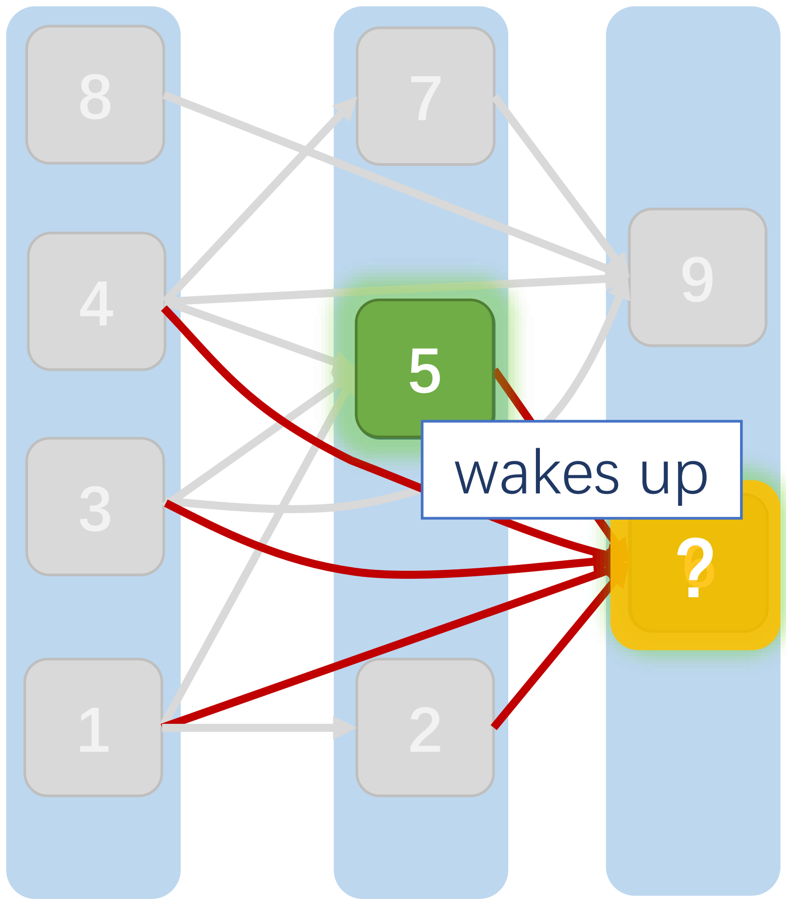


Pivot: an unfinished predecessor selected uniformly at random

If the pivot of an object haven’t finished, this object cannot be ready

We check readiness of an object **only when its pivot finishes**

Core idea: select a "pivot" for every object

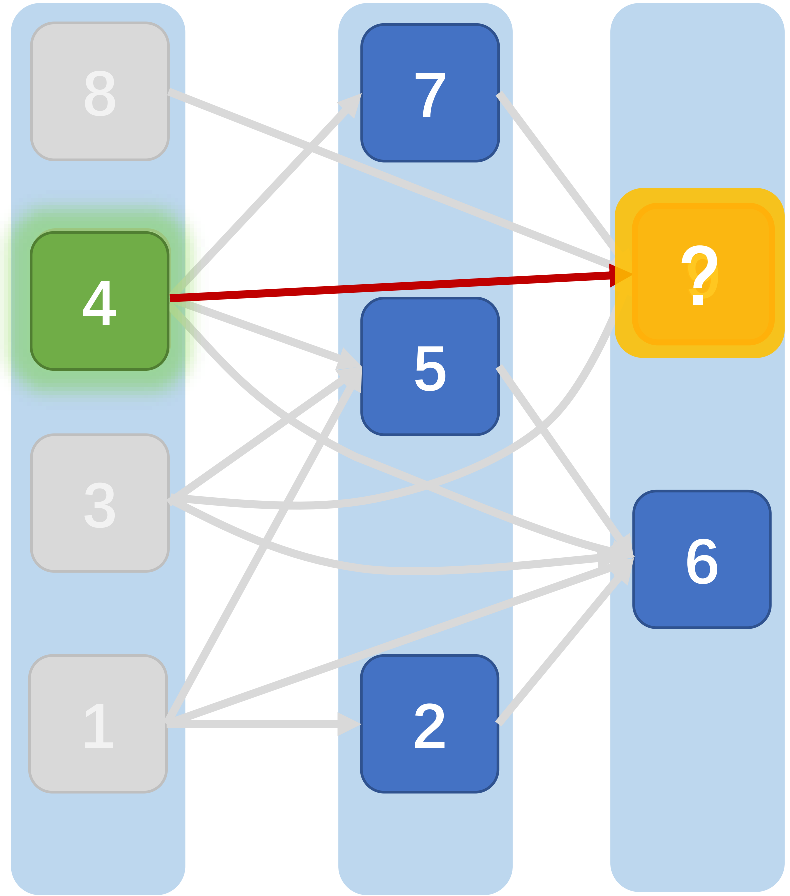


Pivot: an unfinished predecessor selected uniformly at random

If the pivot of an object haven't finished, this object cannot be ready

We check readiness of an object **only when its pivot finishes**

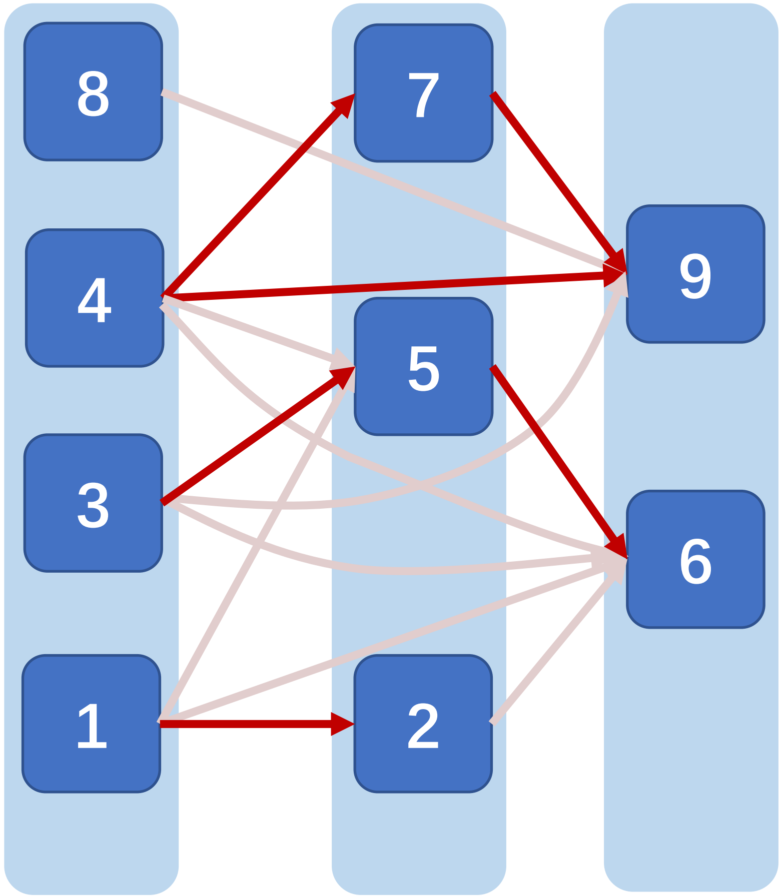
Wake-up and re-pivoting



If the object waked up is **not ready**

- Update pivot to another unprocessed objects
- Sleep until being waked up the next time

Wake-up and re-pivoting



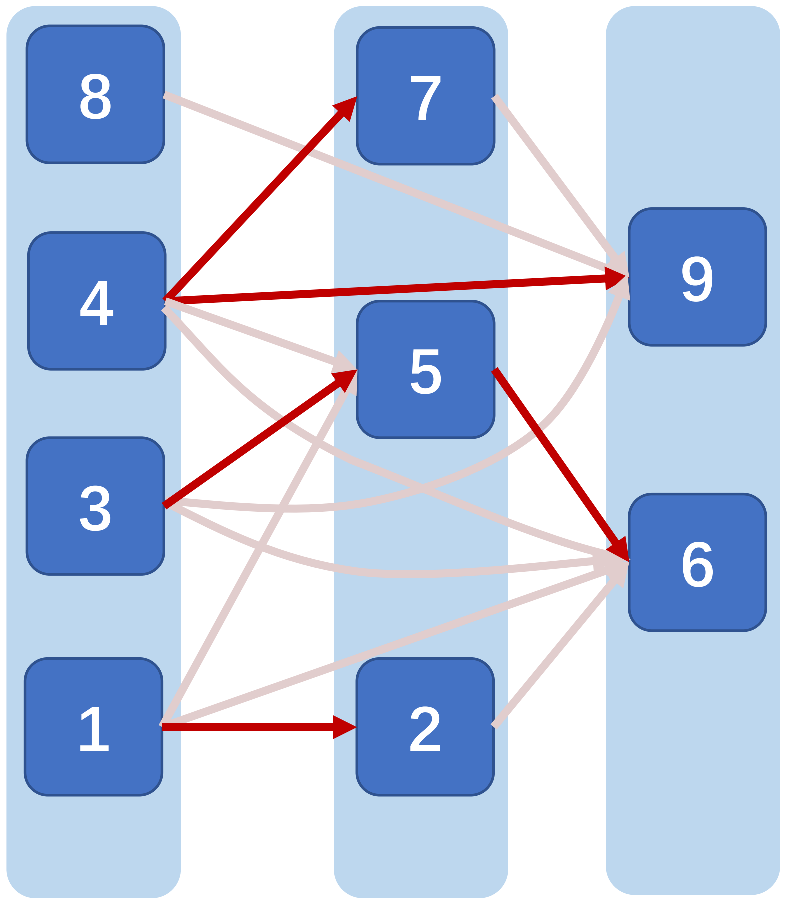
Assign a **pivot** to each object

Choose another pivot if not ready when being waked up

Save work as only a few edges are evaluated

If choosing pivots at uniformly random,
#evaluated edges is $O(n \log n)$ w.h.p.

Apply to the LIS algorithm



$\mathtt{F = \{ready\ at\ the\ beginning\}}$

while $F \neq \emptyset$

{

process F

$S_{woken} = \{\text{objects woken by F}\}$

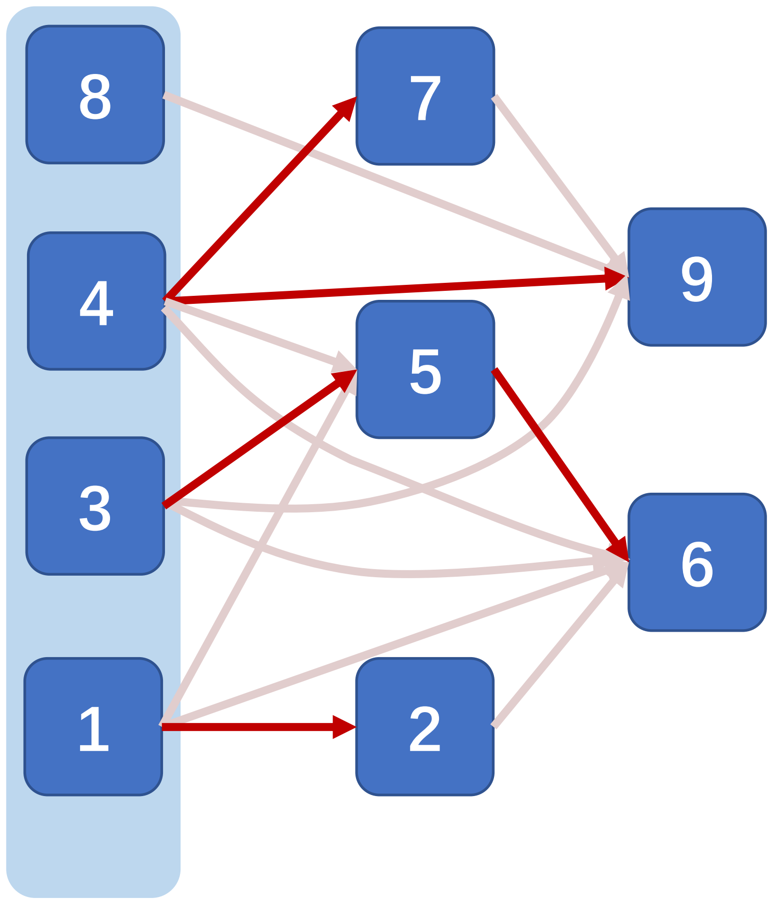
$S_{ready} = \{u \in S_{woken},\ u \text{ is ready}\}$

update pivots in $S_{woken} - S_{ready}$

$F = S_{ready}$

}

Apply to the LIS algorithm



F = {ready at the beginning}

while F $\neq \emptyset$

{

    process F

    S_{woken} = {objects woken by F}

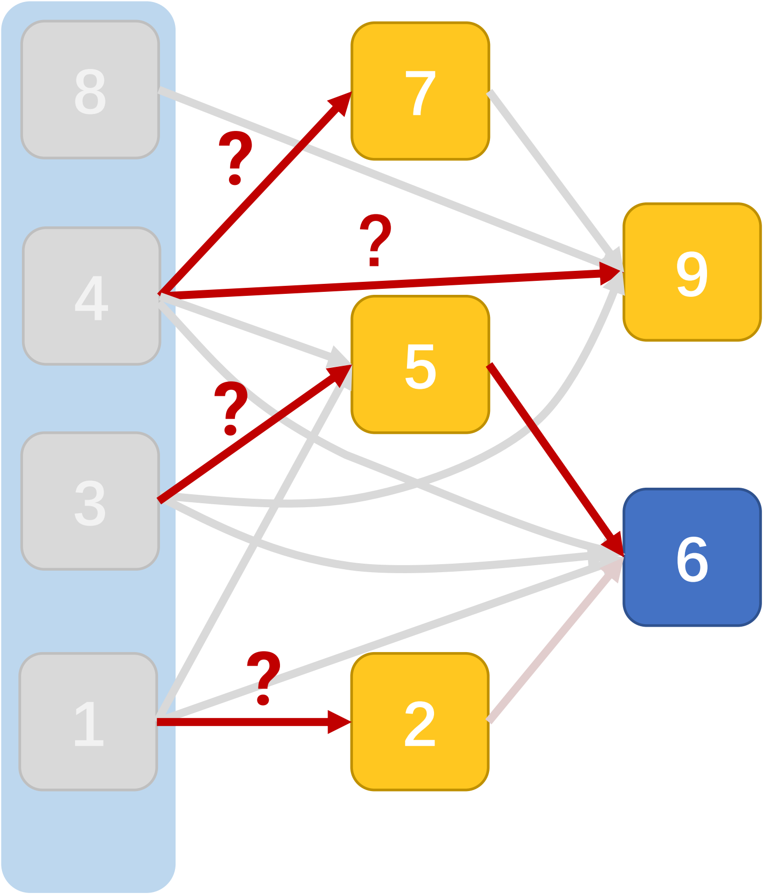
    S_{ready} = {$u \in S_{woken}$, u is ready}

    update pivots in $S_{woken} - S_{ready}$

    F = S_{ready}

}

Apply to the LIS algorithm




```
F = {ready at the beginning}

while F ≠ Ø
{
     process F
```

$S_{woken} = \{\text{objects woken by F}\}$

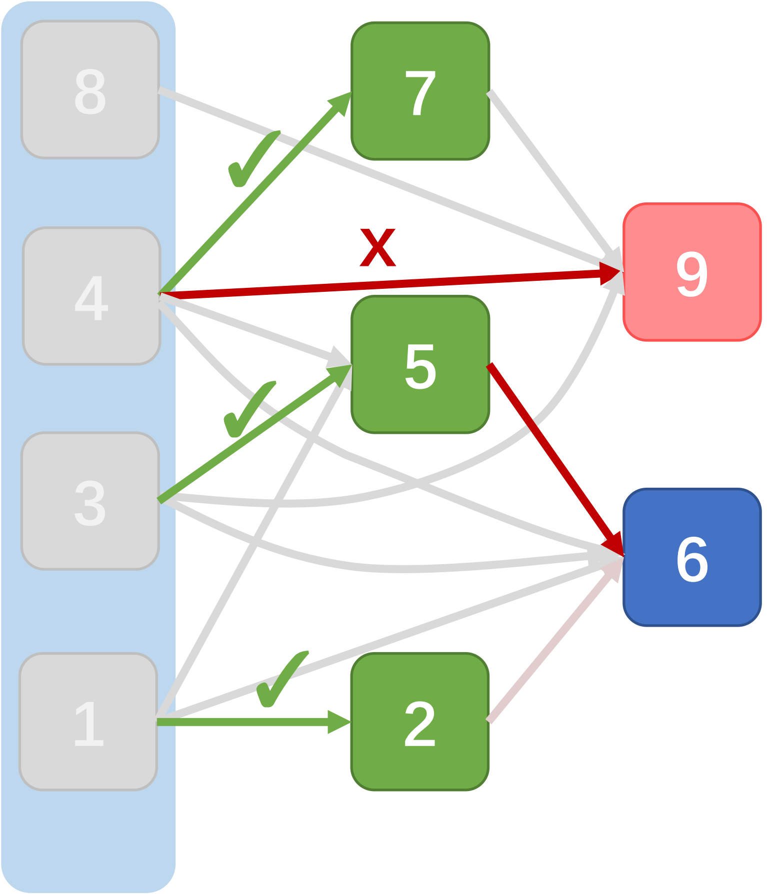
$S_{ready} = \{u \in S_{woken}, \text{ u is ready}\}$

```
     update pivots in
```
$S_{woken} - S_{ready}$

$F = S_{ready}$

```
}
```

Apply to the LIS algorithm



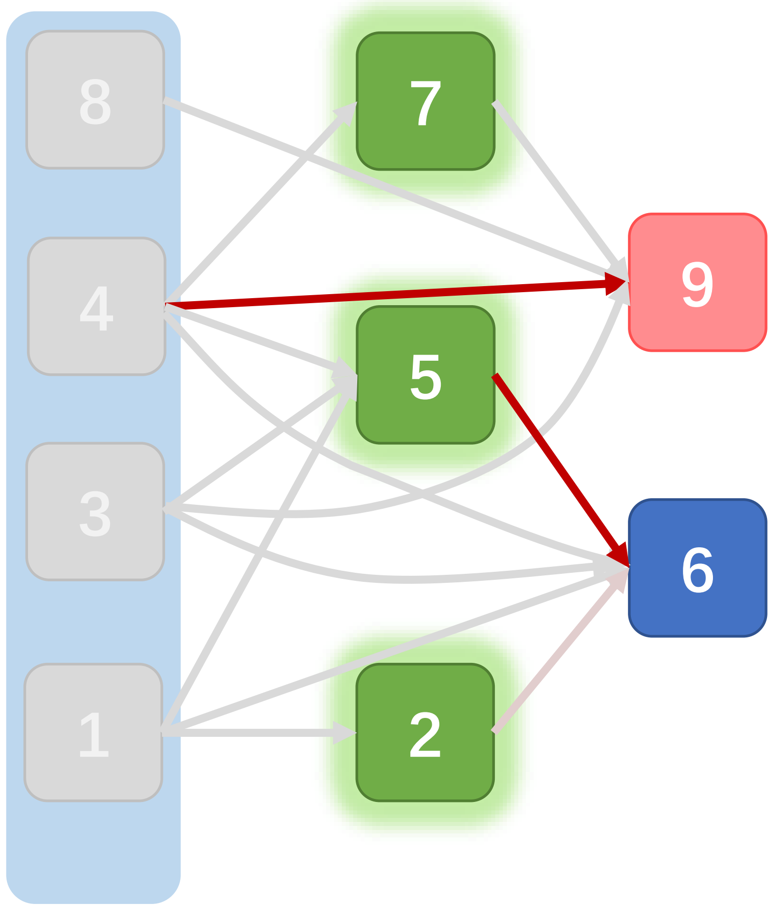
F = {ready at the beginning}

while $F \neq \emptyset$
{
    process F

    $S_{woken} = \{\text{objects woken by F}\}$
    $S_{ready} = \{u \in S_{woken},\ u \text{ is ready}\}$

    update pivots in $S_{woken} - S_{ready}$
    $F = S_{ready}$
}

Apply to the LIS algorithm



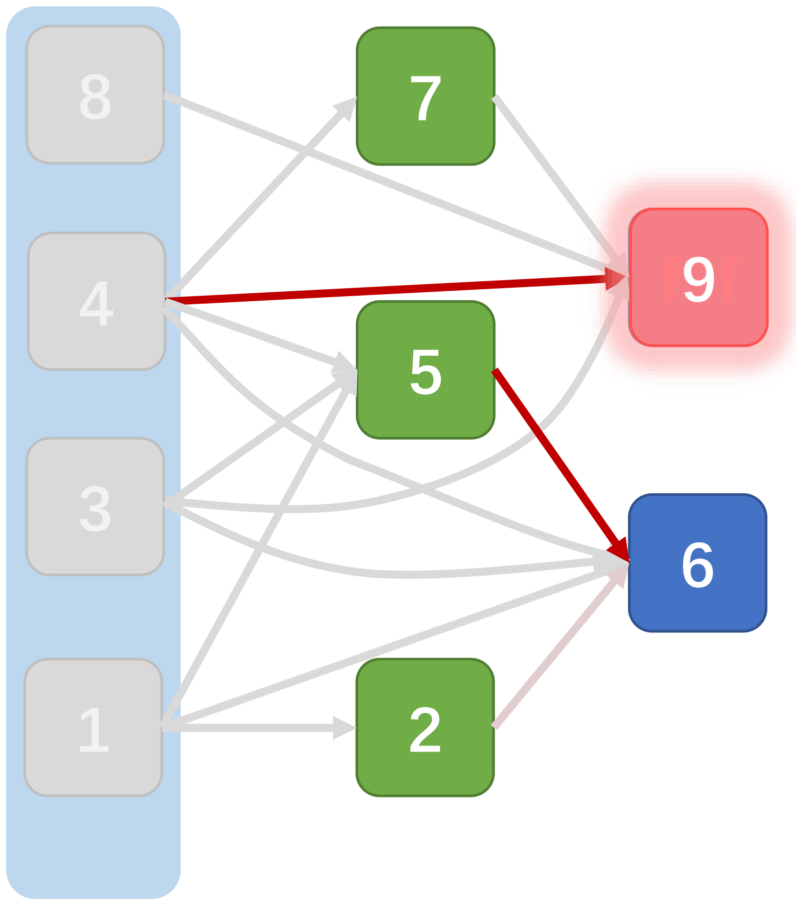
F = {ready at the beginning}

while $F \neq \emptyset$
{
    process F

    $S_{woken} = \{\text{objects woken by } F\}$
    $S_{ready} = \{u \in S_{woken},\ u \text{ is ready}\}$

    update pivots in $S_{woken} - S_{ready}$
    $F = S_{ready}$
}

Apply to the LIS algorithm



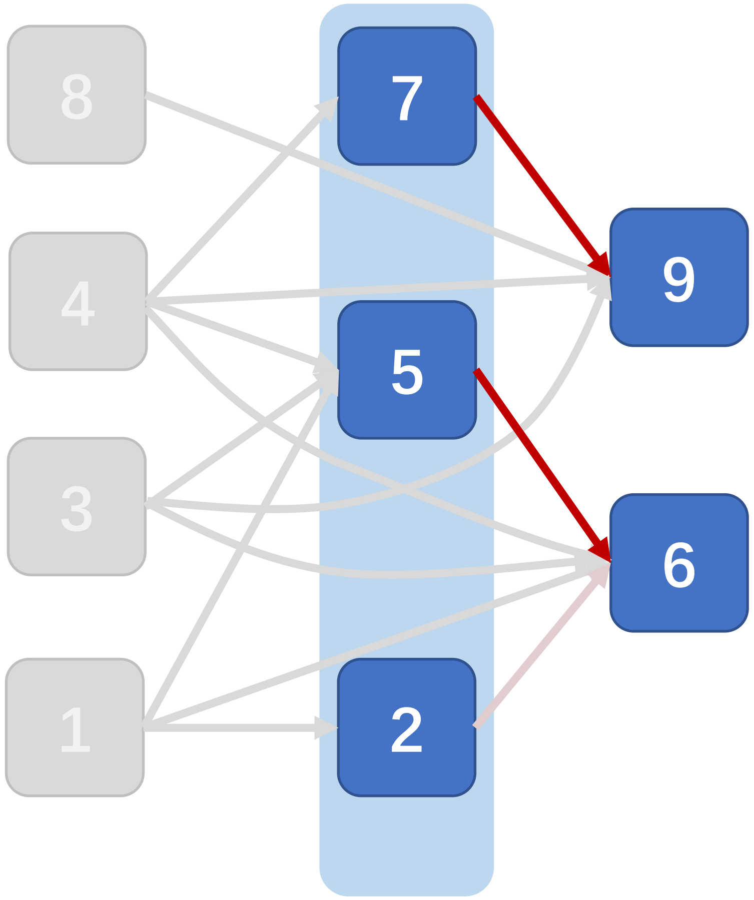
F = {ready at the beginning}

while $F \neq \emptyset$
{
    process F

    $S_{woken} = \{\text{objects woken by F}\}$
    $S_{ready} = \{u \in S_{woken},\ u \text{ is ready}\}$

    update pivots in $S_{woken} - S_{ready}$
    $F = S_{ready}$
}

Apply to the LIS algorithm



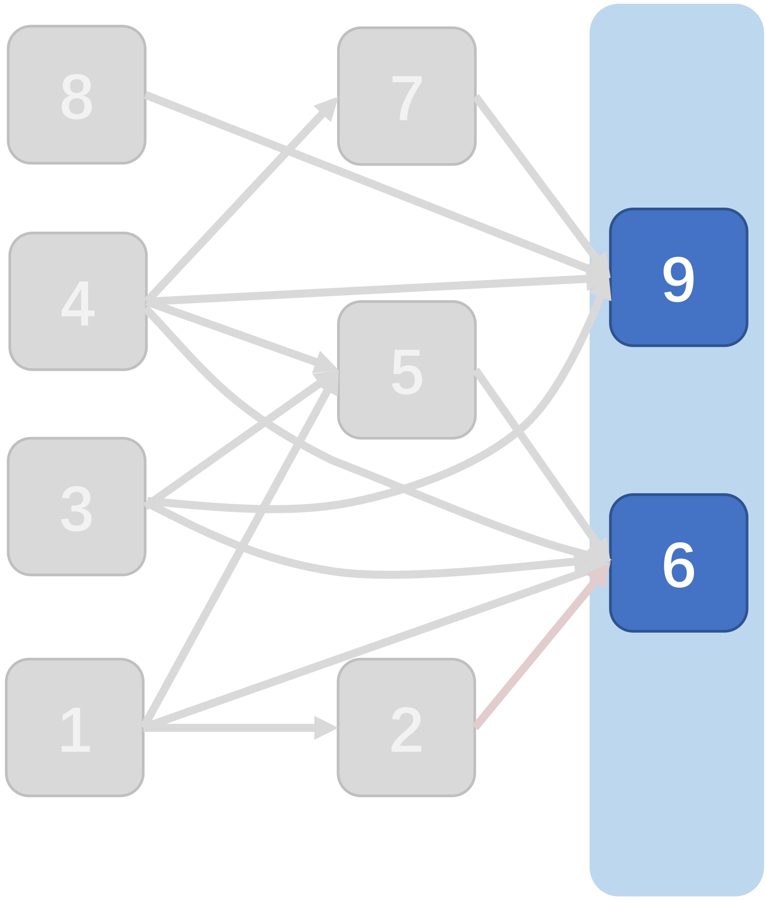
F = {ready at the beginning}

while $F \neq \emptyset$
{
    process F

    $S_{woken} = \{\text{objects woken by F}\}$
    $S_{ready} = \{u \in S_{woken},\ u \text{ is ready}\}$

    update pivots in $S_{woken} - S_{ready}$
    $F = S_{ready}$
}

Apply to the LIS algorithm



F = {ready at the beginning}

while $F \neq \emptyset$
{
    process F

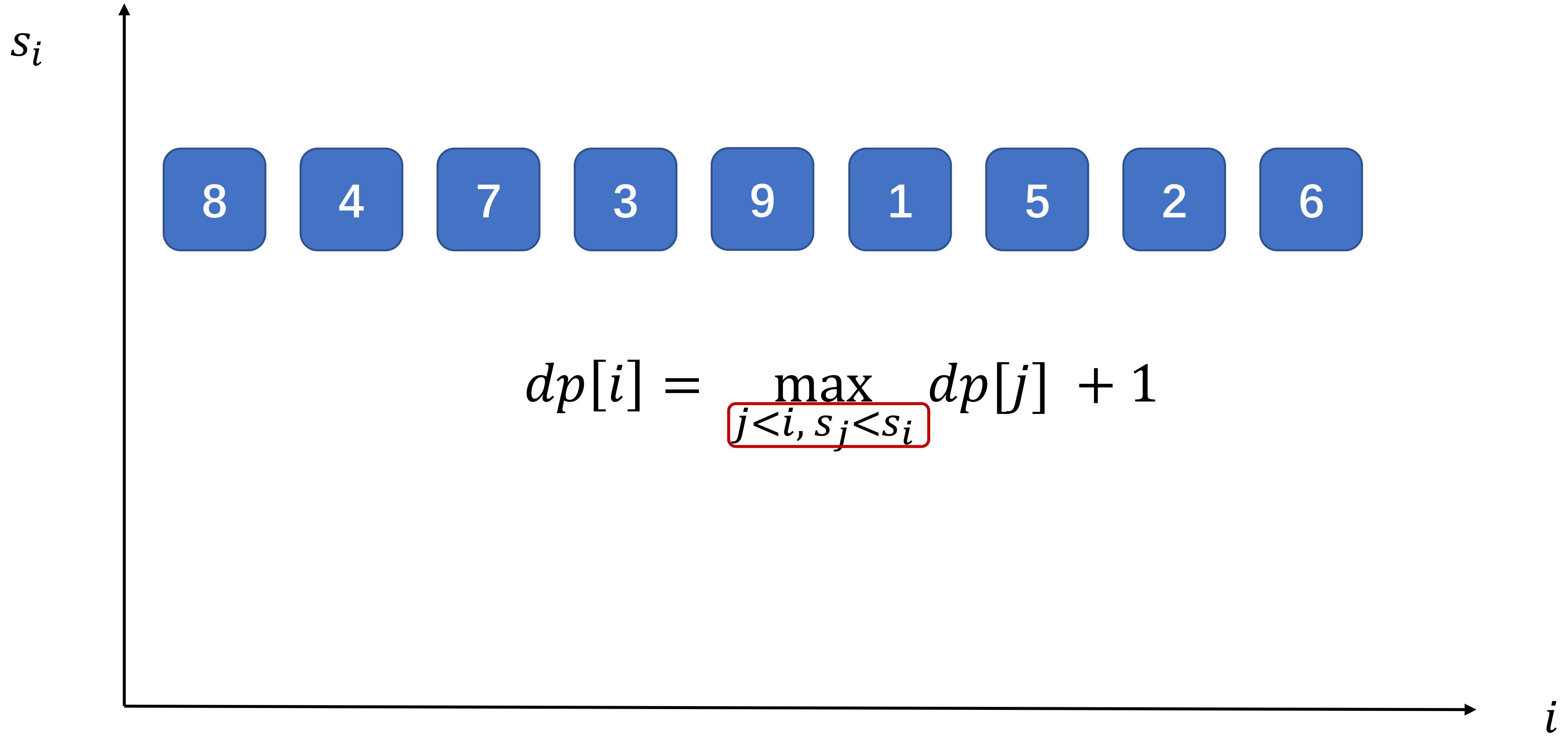
    $S_{woken} = \{\text{objects woken by F}\}$
    $S_{ready} = \{u \in S_{woken},\ \text{u is ready}\}$

    update pivots in $S_{woken} - S_{ready}$
    $F = S_{ready}$
}

Apply to the LIS algorithm

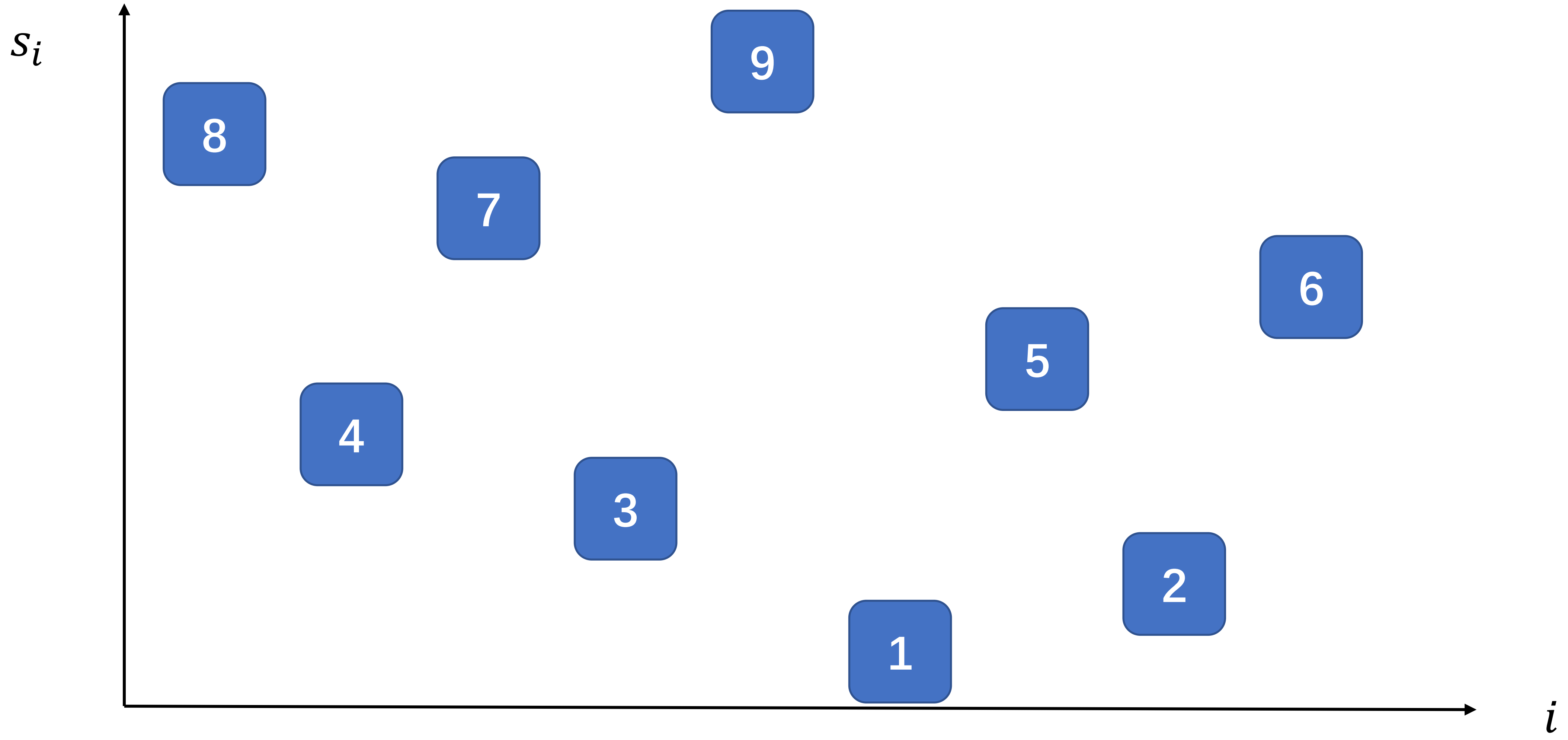
	F = {ready at the beginning}
	while $F \neq \emptyset$
	{
DP value computation	process F
	$S_{woken} = \{\text{objects woken by F}\}$
Readiness check	$S_{ready} = \{u \in S_{woken},\ u \text{ is ready}\}$
Pivot reselection	update pivots in $S_{woken} - S_{ready}$
	$F = S_{ready}$
	}

Vertex-centric Approach

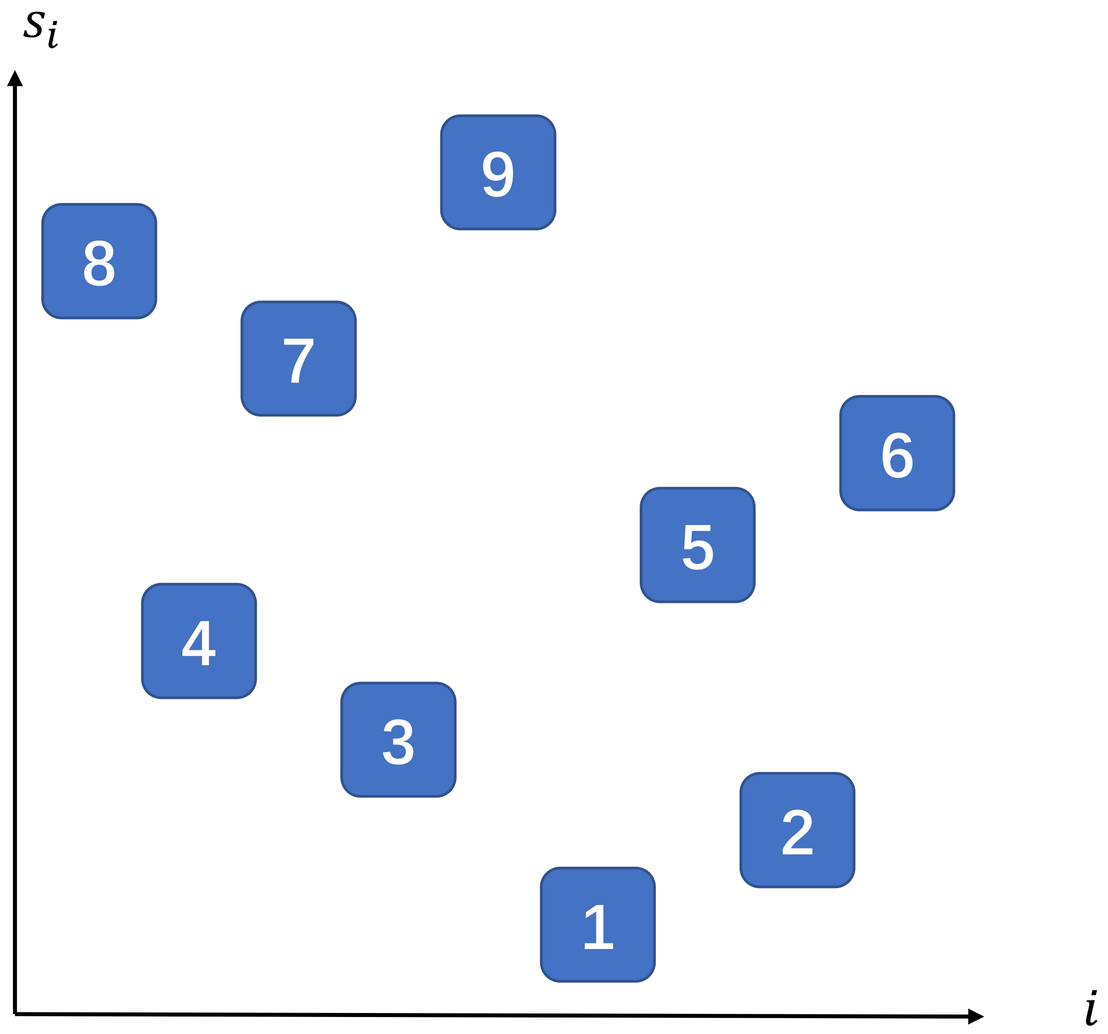


$$dp[i] = \max_{\boxed{j<i,\, s_j<s_i}} dp[j] + 1$$

Vertex-centric Approach

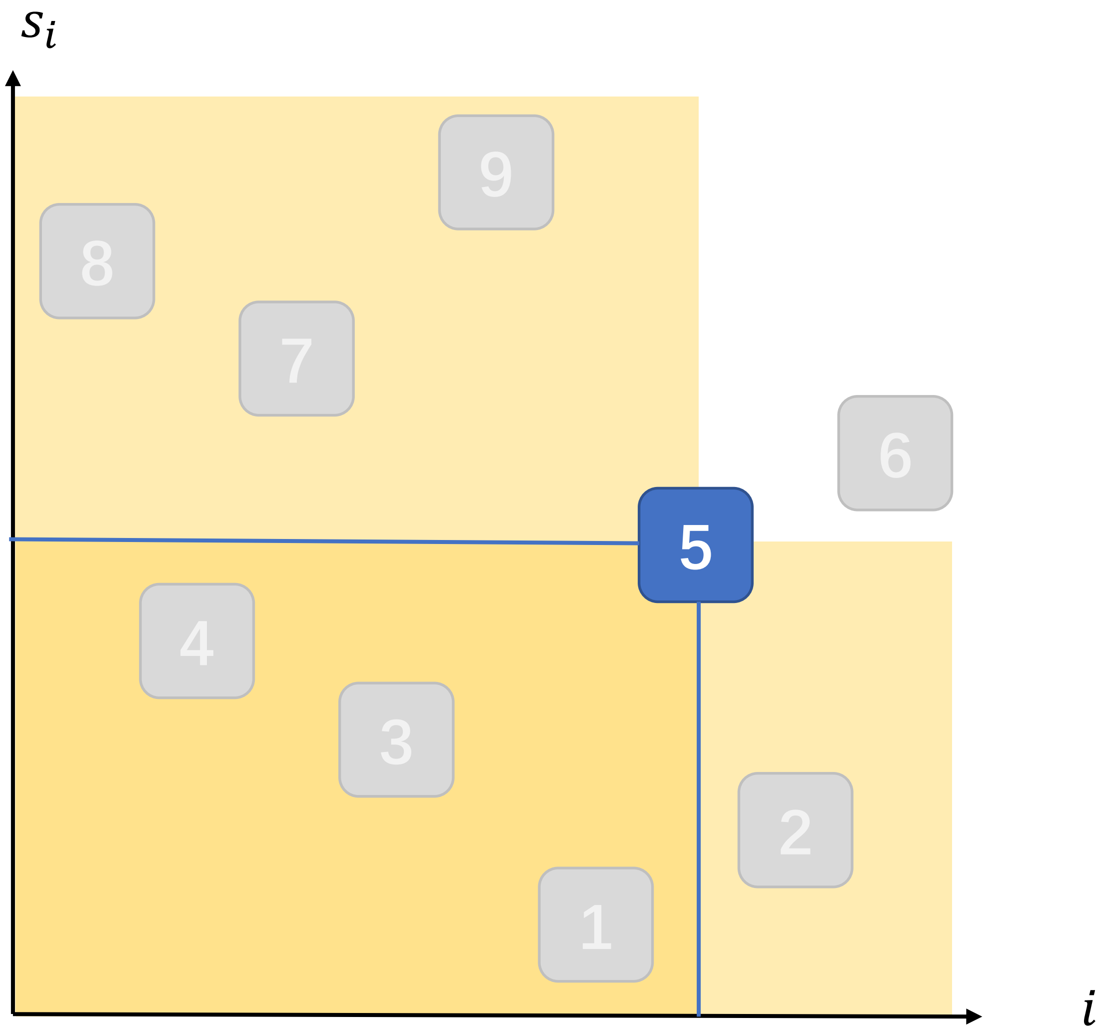


Vertex-centric Approach



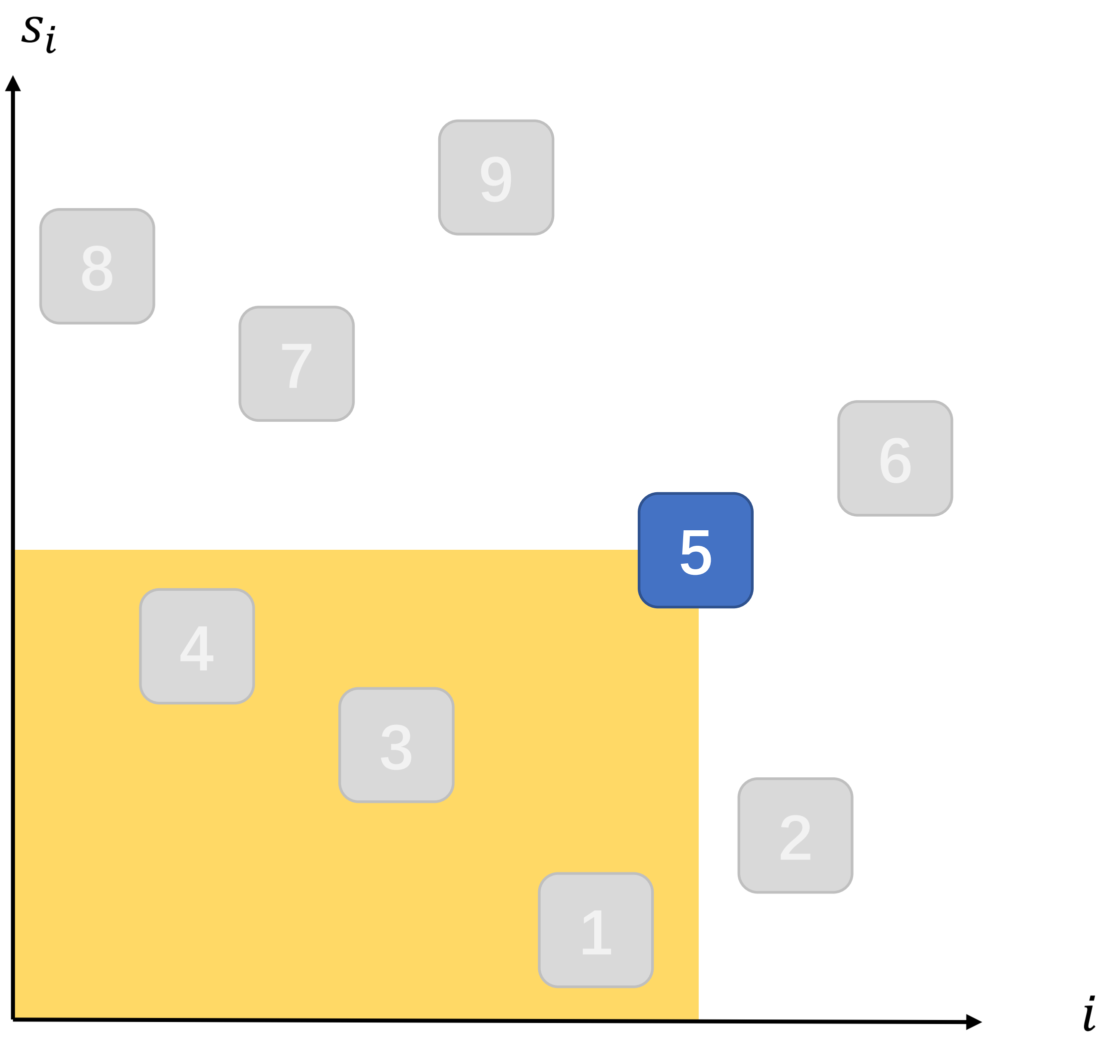
$$dp[i] = \max_{j<i,\, s_j<s_i} dp[j] + 1$$

Vertex-centric Approach



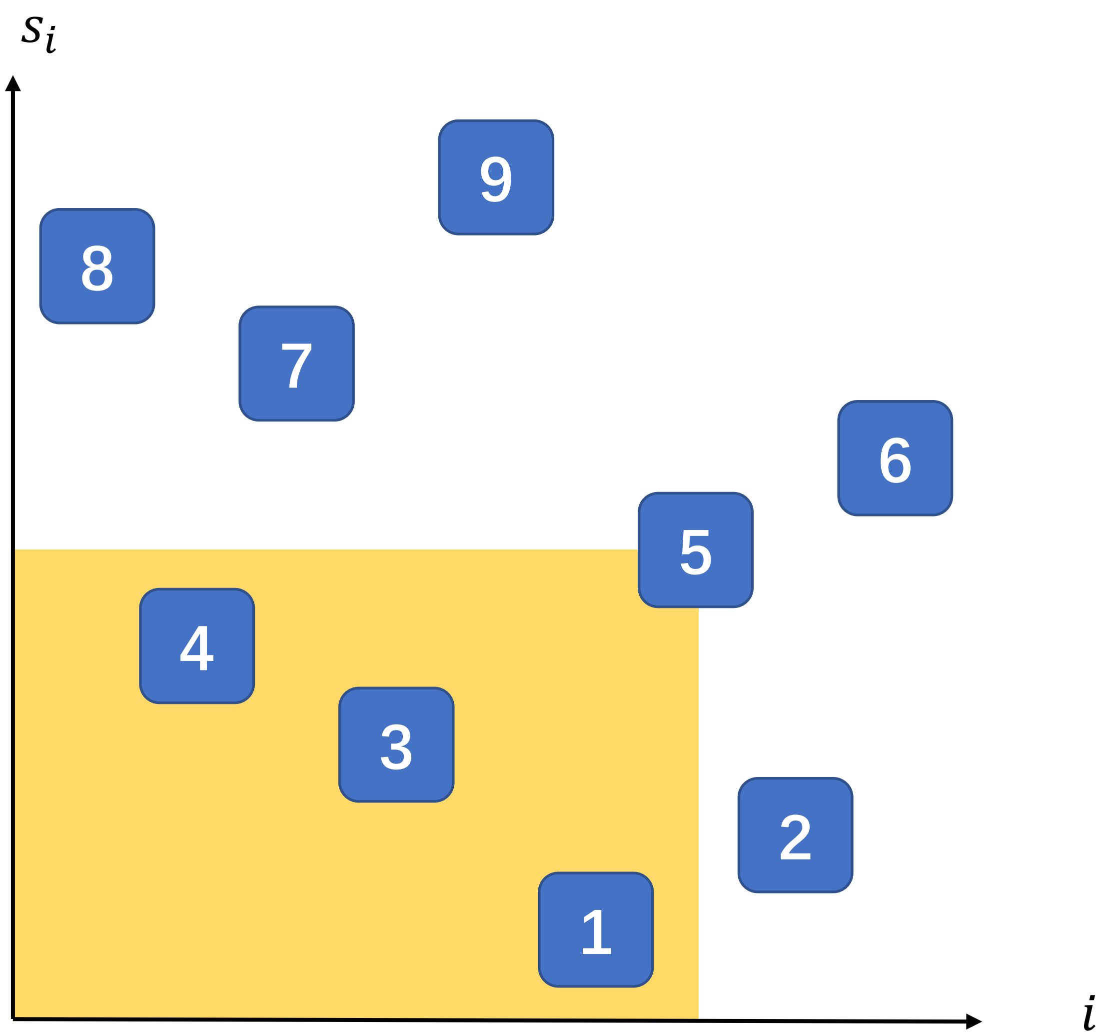
$$dp[i] = \max_{j<i,\, s_j<s_i} dp[j] + 1$$

Vertex-centric Approach



$$dp[i] = \max_{j<i,\, s_j<s_i} dp[j] + 1$$

Vertex-centric Approach



$$dp[i] = \max_{j<i,\, s_j<s_i} dp[j] + 1$$

Readiness check

DP value computation

Pivot re-selection

Above operations can be done by a 2D range tree in $O(\log^2 n)$

[Sun, et al. (PPoPP 2018)]

Complexity of our parallel LIS algorithm

`input`	n
`# wake-ups`	$O(n \log n)$ total wake-ups w.h.p.

`per wake-up`
`- Compute DP values`
`- Check readiness`
`- Re-select pivots`
	$O(\log^2 n)$ by range queries

Parallel LIS Algorithm

Using our phase-parallel framework and vertex-centric methods, we parallelize LIS algorithm with

Nearly work-efficient: $O(n \log^3 n)$ work w.h.p.

Round-efficient: $O(r \log^2 n)$ span

$n =$ input size

$r =$ the LIS length of the input

Key Techniques

Pivots

Wake-up strategy

Vertex-centric readiness check

More applications in **Type-2** approach

- Other range-search-based greedy or dynamic programming
- Maximal independent set (MIS) using wake-up strategy only

	Cost of single readiness check	# candidates in next round
Type-2 approach	Polylog (by range queries)	A few successors (by wake-up strategy)

	Cost of single readiness check	# candidates in next round
Type-1 approach	Trivial	Logarithmic (by 1-D range queries)
Type-2 approach	Polylog (by range queries)	A few successors (by wake-up strategy)

Improved Bounds

* with high probability

	Work	Span
LIS	$O(n \log^3 n)$ *	$O(r \log^2 n)$
MIS	$O(n + m)$	$O(\log^2 n)$ *
Activity Selection (weighted)	$O(n \log n)$	$O(r \log n)$
Activity Selection (unweighted)	$O(n \log n)$	$O(\log n)$ *

Improved Bounds

* with high probability

	Work	Span
LIS	$O(n \log^3 n)$ *	$O(r \log^2 n)$
MIS	$O(n + m)$	$O(\log^2 n)$ *
Activity Selection (weighted)	$O(n \log n)$	$O(r \log n)$
Activity Selection (unweighted)	$O(n \log n)$	$O(\log n)$ *

Improved Bounds

* with high probability

	Work	Span
LIS	$O(n \log^3 n)$ *	$O(r \log^2 n)$
MIS	$O(n + m)$	$O(\log^2 n)$ *
Activity Selection (weighted)	$O(n \log n)$	$O(r \log n)$
Activity Selection (unweighted)	$O(n \log n)$	$O(\log n)$ *

Improved Bounds

* with high probability

	Work	Span
LIS	$O(n \log^3 n)$ *	$O(r \log^2 n)$
MIS	$O(n + m)$	$O(\log^2 n)$ *
Activity Selection (weighted)	$O(n \log n)$	$O(r \log n)$
Activity Selection (unweighted)	$O(n \log n)$	$O(\log n)$ *

Improved Bounds

* with high probability

	Work	Span
LIS	$O(n \log^3 n)$ *	$O(r \log^2 n)$
MIS	$O(n + m)$	$O(\log^2 n)$ *
Activity Selection (weighted)	$O(n \log n)$	$O(r \log n)$
Activity Selection (unweighted)	$O(n \log n)$	$O(\log n)$ *

Huffman tree, graph coloring, SSSP, unlimited knapsack, and more in the paper

Experiment Setup

Hardware

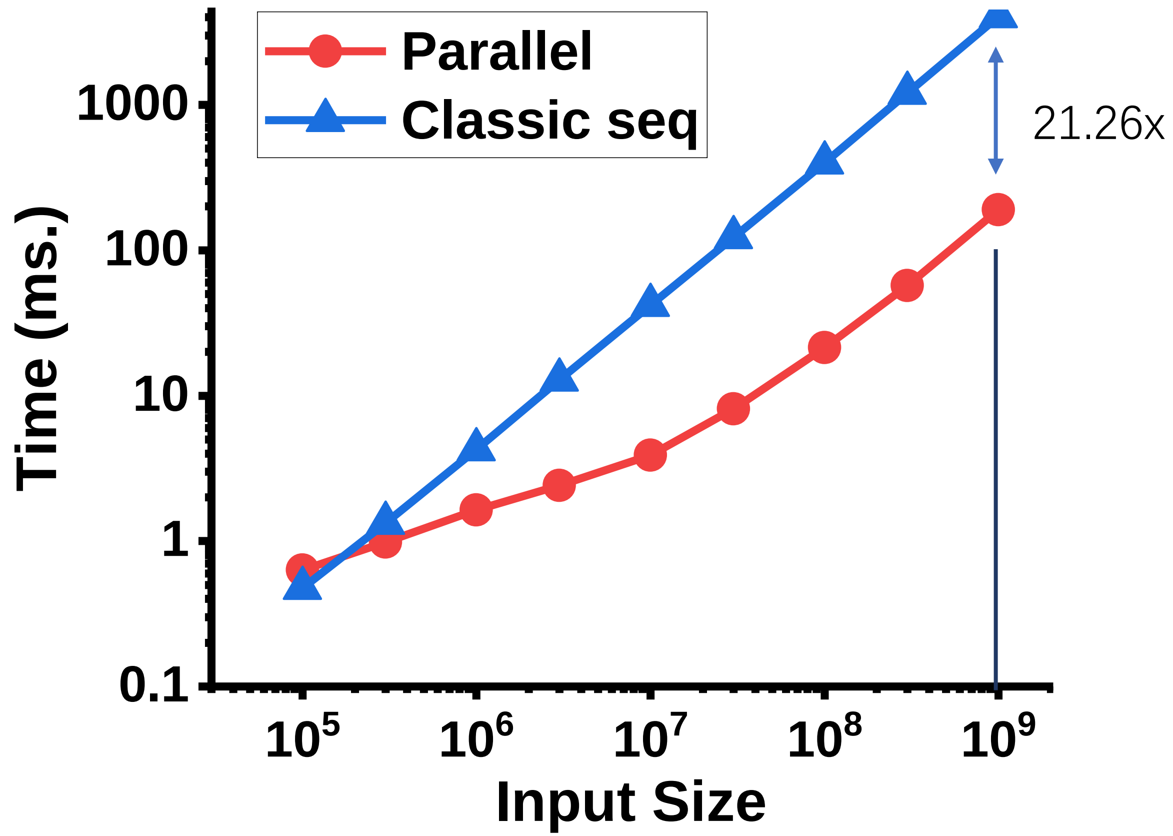
- 96 CPU cores (192 hyper-threads)
- 1.5 TiB of main memory

Parallelized Algorithms

- Huffman Tree
- Activity selection
- LIS
- …

Work-efficient algorithms generally perform well

Huffman Tree



Our algorithm is work-efficient

$\tilde{O}(r)$ span for rank r (tree height)

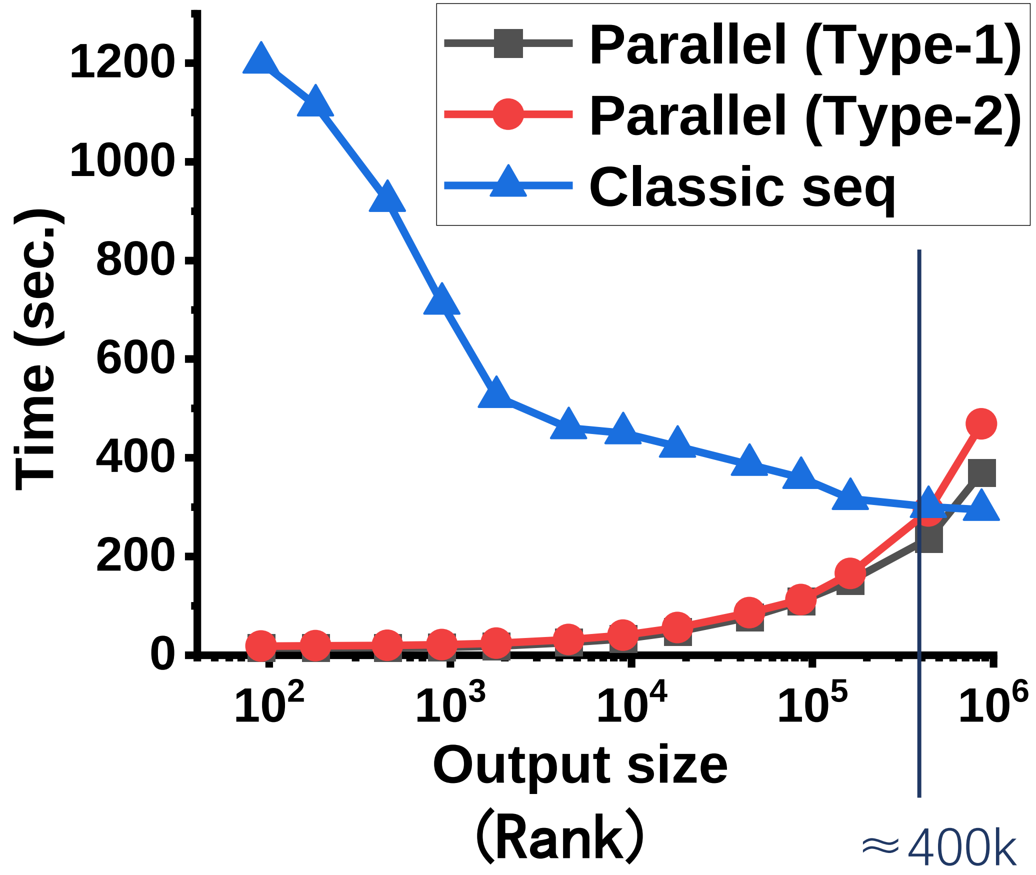
The rank of the test data is low

The algorithm performs well

- 21.26x speedup to the sequential

Activity Selection

$n = 10^9$

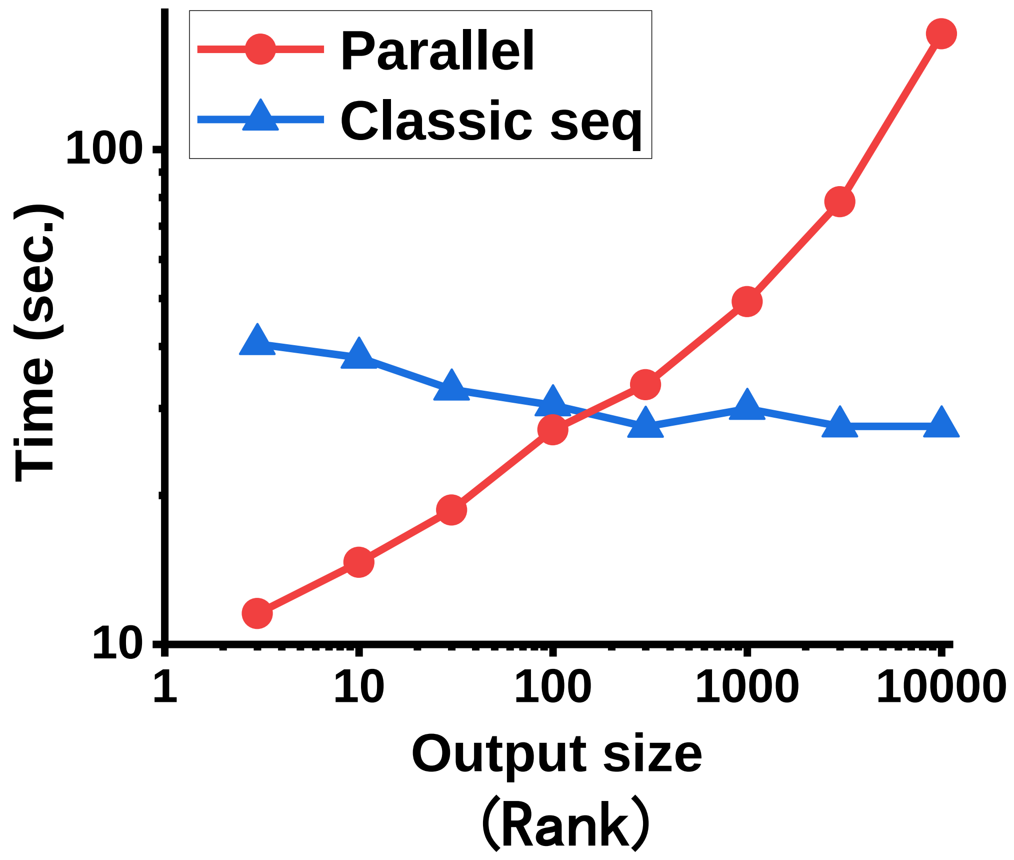


Implement with both Type-1 and -2

- Both are work-efficient: $O(n \log n)$
- Span is $\tilde{O}(r)$ for rank r

The algorithm performs well on a wide range of rank values

Longest Increasing Subsequence



Nearly work-efficient: $O(n \log^3 n)$

- $O(\log^2 n)$ overhead

Run fast on small ranks

- Overhead in work still limits its performance for large ranks

Open problem:

$O(n \log n)$ work with good parallelism?

Summary

Motivation

- Many existing sequential iterative algorithms can be highly parallel
- Parallelize these algorithms efficiently

New Techniques: general to many algorithms

- Phase-parallel framework
- Type-1 and Type-2 approaches

New algorithms (many improving best known bounds)

- LIS, activity selection, MIS, Huffman tree, SSSP, …

Future Work

Can LIS problem be solved in $O(n \log n)$ with good parallelism?

Can our techniques apply to other problems?

Thank you

Summary

Motivation

- Many existing sequential iterative algorithms can be highly parallel
- Parallelize these algorithms efficiently

New Techniques: general to many algorithms

- Phase-parallel framework
- Type-1 and Type-2 approaches

New algorithms (many improving best known bounds)

- LIS, activity selection, MIS, Huffman tree, SSSP, …