Makespan Scheduling of Unit Jobs with Precedence Constraints in $O(1.995^n)$ time

Jesper Nederlof

UU

Céline Swennenhuis

TU/e

Karol Węgrzycki

Saarland University

The results presented were obtained during the trimester on Discrete Optimization at Hausdorff Research Institute for Mathematics (HIM) in Bonn, Germany.

Department of Mathematics and Computer Science, Eindhoven University of Technology

$$P|prec, p_j = 1|C_{\max}$$

$P|prec, p_j = 1|C_{\max}$

Given:

- n jobs of length 1

$P|prec, p_j = 1|C_{\max}$

Given:

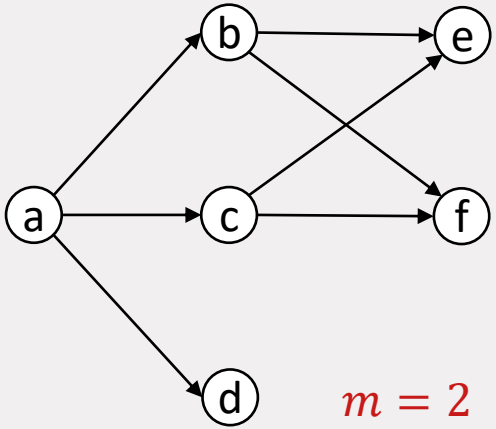
- n jobs of length 1
- m (identical, parallel) machines

$m = 2$

$P|prec, p_j = 1|C_{\max}$

Given:

- n jobs of length 1
- m (identical, parallel) machines
- A precedence graph G



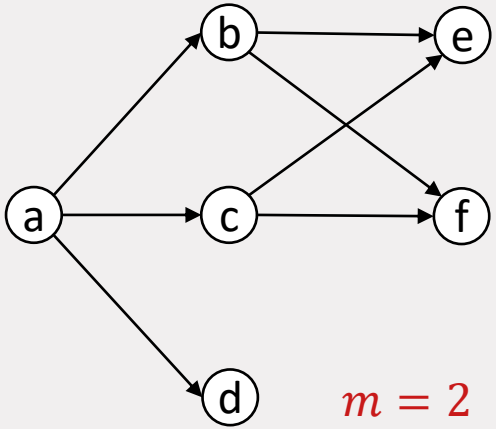
$m = 2$

$P|prec, p_j = 1|C_{\max}$

Given:

- n jobs of length 1
- m (identical, parallel) machines
- A precedence graph G
- $T \in \mathbb{N}$

Q: Is there a schedule of makespan T?



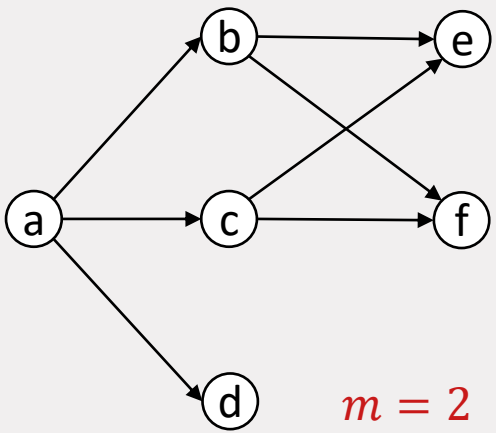
$m = 2$

$P|prec, p_j = 1|C_{\max}$

Given:

- n jobs of length 1
- m (identical, parallel) machines
- A precedence graph G
- $T \in \mathbb{N}$

Q: Is there a schedule of makespan T?



time →

	1	2	3	4
1	a	b	d	f
2		c	e	

Literature overview $P|prec, p_j = 1|C_{\max}$

Literature overview $P|prec, p_j = 1|C_{\max}$

- NP-complete[1] m = #machines given *as input*

[1]Jeffrey D. Ullman. *NP-complete scheduling problems.* Journal of Computer and System sciences, 10(3):384–393, 1975.

Literature overview $P|prec, p_j = 1|C_{\max}$

- NP-complete[1] m = #machines given *as input*

[1]Jeffrey D. Ullman. *NP-complete scheduling problems.* Journal of Computer and System sciences, 10(3):384–393, 1975.

- Poly-time solvable[2] for $m = 2$

[2]M. Fujii, T. Kasami, and K. Ninomiya. *Optimal sequencing of two equivalent processors*. SIAM Journal on Applied Mathematics, 17(4):784–789, 1969.

Literature overview $P|prec, p_j = 1|C_{\max}$

- NP-complete[1] m = #machines given *as input*

[1]Jeffrey D. Ullman. *NP-complete scheduling problems.* Journal of Computer and System sciences, 10(3):384–393, 1975.

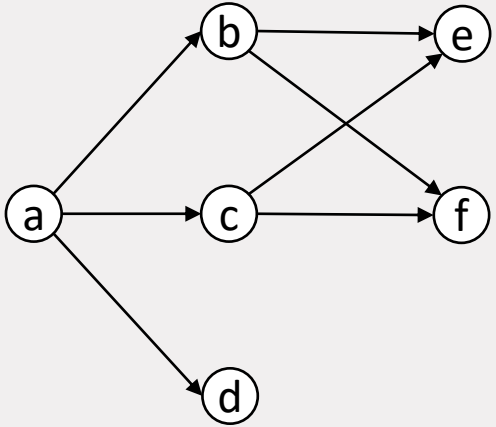
- Poly-time solvable[2] for $m = 2$

[2]M. Fujii, T. Kasami, and K. Ninomiya. *Optimal sequencing of two equivalent processors*. SIAM Journal on Applied Mathematics, 17(4):784–789, 1969.

- ???? for $m \geq 3$ **constant** **OPEN**[3]

[3]Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

Definitions

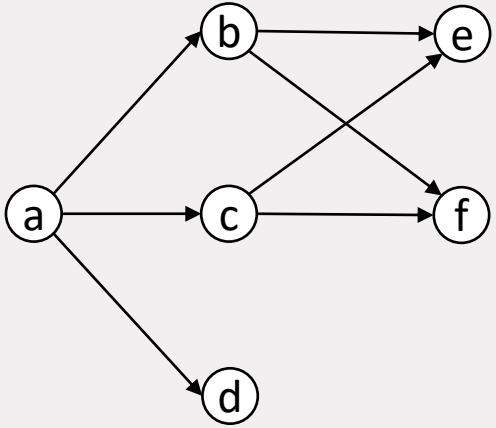


Precendence Constraints Graph G

Definitions

$G \Rightarrow$ partial order:

- $i \prec j$ if $(i, j) \in G$

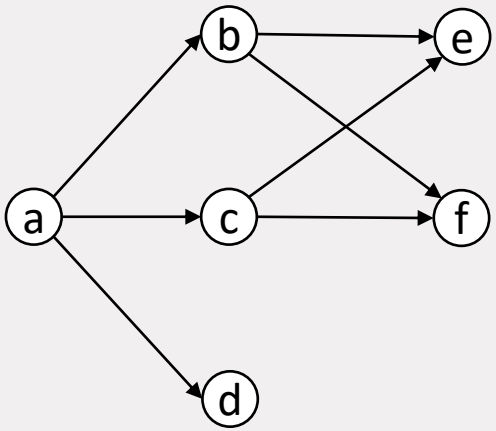


Precendence Constraints Graph G

Definitions

$G \Rightarrow$ partial order:

- $i \prec j$ if $(i,j) \in G$



Precendence Constraints Graph G

<u>Definition:</u> Let A be a set of jobs.

$pred[A] \quad = \{x \mid \exists\, a \in A \; s.t. \; x \preccurlyeq a\}$

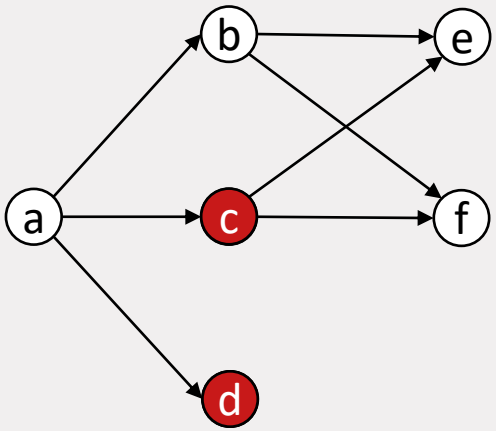
$succ[A] \quad = \{x \mid \exists\, a \in A \; s.t. \; x \succcurlyeq a\}$

$sinks(A) = \max\{A\}$

Definitions

$G \Rightarrow$ partial order:

- $i \prec j$ if $(i,j) \in G$



Precendence Constraints Graph G

<u>Definition:</u> Let A be a set of jobs.

$pred[A] \quad = \{x \mid \exists\, a \in A \; s.t. \; x \preccurlyeq a\}$

$succ[A] \quad = \{x \mid \exists\, a \in A \; s.t. \; x \succcurlyeq a\}$

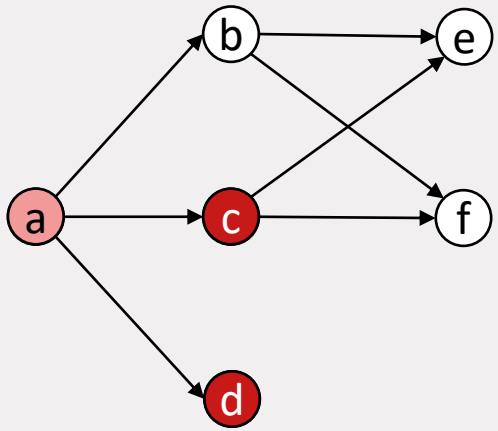
$sinks(A) = \max\{A\}$

Let $A = \{c, d\}$, then:

Definitions

$G \Rightarrow$ partial order:

- $i \prec j$ if $(i,j) \in G$



Precendence Constraints Graph G

Definition: Let A be a set of jobs.

$pred[A] \quad = \{x \mid \exists\, a \in A \; s.t. \; x \preccurlyeq a\}$

$succ[A] \quad = \{x \mid \exists\, a \in A \; s.t. \; x \succcurlyeq a\}$

$sinks(A) = \max\{A\}$

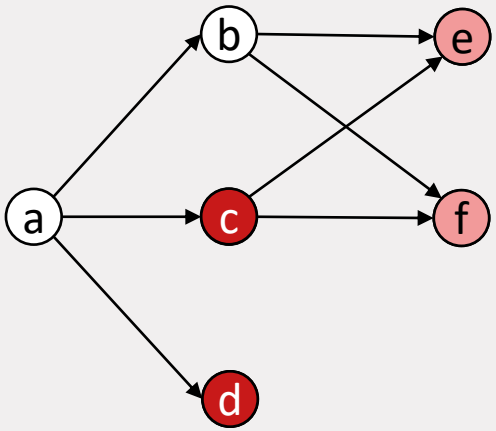
Let $A = \{c, d\}$, then:

- $pred[A] = \{a, c, d\}$

Definitions

$G \Rightarrow$ partial order:

- $i \prec j$ if $(i,j) \in G$



Precendence Constraints Graph G

Definition: Let A be a set of jobs.

$pred[A] \quad = \{x \mid \exists\, a \in A \; s.t. \; x \preccurlyeq a\}$

$succ[A] \quad = \{x \mid \exists\, a \in A \; s.t. \; x \succcurlyeq a\}$

$sinks(A) = \max\{A\}$

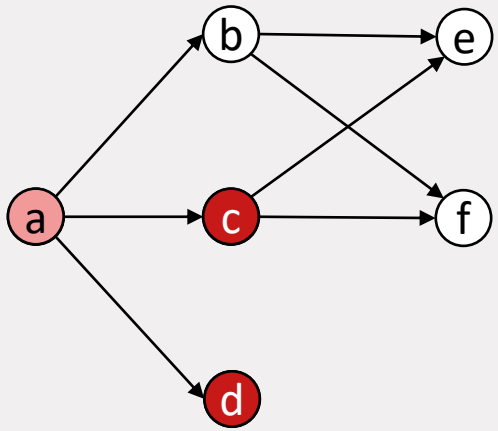
Let $A = \{c, d\}$, then:

- $pred[A] = \{a, c, d\}$
- $succ[A] = \{c, d, e, f\}$

Definitions

$G \Rightarrow$ partial order:

- $i \prec j$ if $(i,j) \in G$



Precendence Constraints Graph G

Definition: Let A be a set of jobs.

$pred[A] = \{x \mid \exists\, a \in A \; s.t. \; x \preccurlyeq a\}$

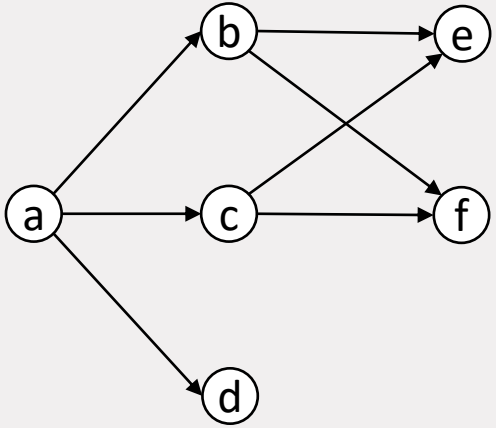
$succ[A] = \{x \mid \exists\, a \in A \; s.t. \; x \succcurlyeq a\}$

$sinks(A) = \max\{A\}$

Let $A = \{c, d\}$, then:

- $pred[A] = \{a, c, d\}$
- $succ[A] = \{c, d, e, f\}$
- $sinks(\{a, c, d\}) = \{c, d\}$

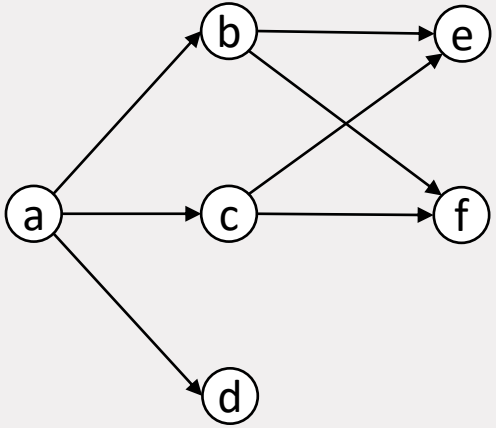
Definitions



Precendence Constraints Graph G

Definitions

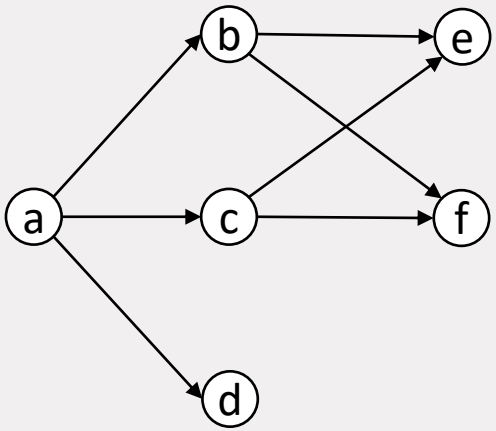
Def: An *antichain* is a set A whose elements are pairwise incomparable.



Precendence Constraints Graph G

Definitions

Def: An *antichain* is a set A whose elements are pairwise incomparable.



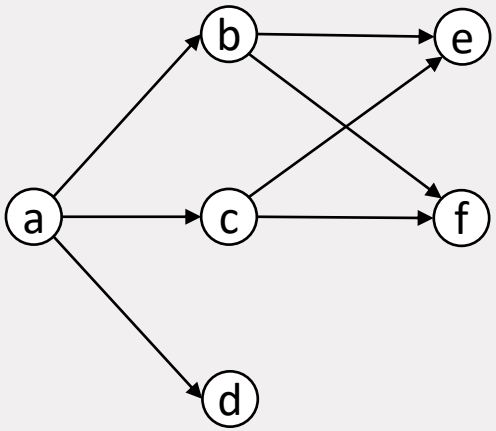
Precendence Constraints Graph G

Ex. of antichains in G

- ✓ $\{b, c, d\}$
- ✓ $\{b, c\}$
- ✓ $\{d, f\}$

Definitions

Def: An *antichain* is a set A whose elements are pairwise incomparable.



Precendence Constraints Graph G

There is a one-to-one relation between:

A = antichain of graph G

$pred[A]$

$sinks(X)$

X = possible set of jobs to schedule

DP Algorithm:

DP Algorithm:

$$T(A,t) = \begin{cases} \text{true}, & \text{if } pred[A] \text{ can be processed in } t \text{ time}, \\ \text{false}, & \text{otherwise.} \end{cases}$$

DP Algorithm:

$$T(A,t) = \begin{cases} \text{true}, & \text{if } pred[A] \text{ can be processed in } t \text{ time,} \\ \text{false}, & \text{otherwise.} \end{cases}$$

Antichain time

DP Algorithm:

$$T(A,t) = \begin{cases} \text{true}, & \text{if } pred[A] \text{ can be processed in } t \text{ time}, \\ \text{false}, & \text{otherwise.} \end{cases}$$

Antichain time

To compute $T(A,t)$:
try all $\binom{n}{m}$ possible jobs at time t

DP Algorithm:

$$T(A,t) = \begin{cases} \text{true}, & \text{if } pred[A] \text{ can be processed in } t \text{ time}, \\ \text{false}, & \text{otherwise.} \end{cases}$$

Antichain time

To compute $T(A,t)$:
try all $\binom{n}{m}$ possible jobs at time t

Runs in $poly(n) \cdot \#\mathrm{AC} \cdot \binom{\mathrm{n}}{\mathrm{m}}$ time

DP Algorithm:

$$T(A,t) = \begin{cases} \text{true}, & \text{if } pred[A] \text{ can be processed in } t \text{ time,} \\ \text{false}, & \text{otherwise.} \end{cases}$$

Antichain time

To compute $T(A,t)$:
try all $\binom{n}{m}$ possible jobs at time t

Runs in $poly(n) \cdot \#\mathrm{AC} \cdot \binom{n}{m}$ time

$\Rightarrow poly(n) \cdot 2^n \cdot \binom{n}{m}$ time algorithm

Before our work:
No $poly(n) \cdot 2^n$ time algorithm known!

$P|prec, p_j = 1|C_{\max}$

Our result:

$P|prec, p_j = 1|C_{\max}$ can be solved in $O(1.995^n)$ time.

$P|prec, p_j = 1|C_{\max}$

Our result:

$P|prec, p_j = 1|C_{\max}$ can be solved in $O(1.995^n)$ time.

Proof: Combination of three algorithms

$$P|prec, p_j = 1|C_{\max}$$

$P|prec, p_j = 1|C_{\max}$

- DP Algorithm
  - Dynamic Programming
  - $poly(n) \cdot \#AC \cdot \binom{n}{m}$ time

$P|prec, p_j = 1|C_{\max}$

- DP Algorithm
  - Dynamic Programming
  - $poly(n) \cdot \#AC \cdot \binom{n}{m}$ time

- Fast Subset Convolution Algorithm
  - Non-trivial application of
    Björklund, Husfeldt and Koivisto (SICOMP 2009)
  - $poly(n) \cdot (\#AC + 2^{n-m})$ time

$P|prec, p_j = 1|C_{\max}$

- DP Algorithm
  - Dynamic Programming
  - $poly(n) \cdot \#AC \cdot \binom{n}{m}$ time

- Fast Subset Convolution Algorithm
  - Non-trivial application of
    Björklund, Husfeldt and Koivisto (SICOMP 2009)
  - $poly(n) \cdot (\#AC + 2^{n-m})$ time

- Fixed-parameter Tractable Algorithm
  - Parameterized by *vertex cover of comparability graph*
  - $poly(n) \cdot 169^{|C|}$ time

$P|prec, p_j = 1|C_{\max}$

- DP Algorithm
  - Dynamic Programming
  - $poly(n) \cdot \#AC \cdot \binom{n}{m}$ time

- Fast Subset Convolution Algorithm
  - Non-trivial application of
    Björklund, Husfeldt and Koivisto (SICOMP 2009)
  - $poly(n) \cdot (\#AC + 2^{n-m})$ time

- Fixed-parameter Tractable Algorithm
  - Parameterized by *vertex cover of comparability graph*
  - $poly(n) \cdot 169^{|C|}$ time

Fast enough when
$|C| < n/7.5$

$P|prec, p_j = 1|C_{\max}$

- DP Algorithm
  - Dynamic Programming
  - $poly(n) \cdot \#AC \cdot \binom{n}{m}$ time
- Fast Subset Convolution Algorithm
  - Non-trivial application of Björklund, Husfeldt and Koivisto (SICOMP 2009)
  - $poly(n) \cdot (\#AC + 2^{n-m})$ time

Fast enough when

$|C| \geq n/7.5$

$\xRightarrow{\text{Dilworth's Theorem}}$

$\#AC \leq 1.94^n$

- Fixed-parameter Tractable Algorithm
  - Parameterized by *vertex cover of comparability graph*
  - $poly(n) \cdot 169^{|C|}$ time

Fast enough when

$|C| < n/7.5$

$P|prec, p_j = 1|C_{\max}$

- DP Algorithm
  - Dynamic Programming
  - $poly(n) \cdot \#AC \cdot \binom{n}{m}$ time
- Fast Subset Convolution Algorithm
  - Non-trivial application of Björklund, Husfeldt and Koivisto (SICOMP 2009)
  - $poly(n) \cdot (\#AC + 2^{n-m})$ time

Fast enough when $|C| \geq n/7.5$

$\xRightarrow{\text{Dilworth's Theorem}}$

$\#AC \leq 1.94^n$

- Fixed-parameter Tractable Algorithm
  - Parameterized by *vertex cover of comparability graph*
  - $poly(n) \cdot 169^{|C|}$ time

Fast enough when $|C| < n/7.5$

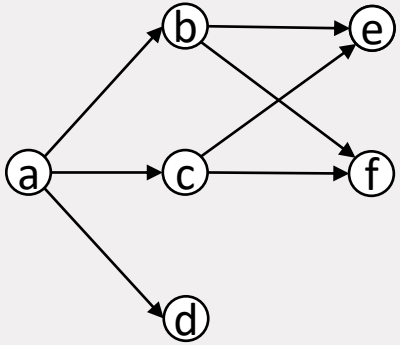
What is vertex cover of comparability graph?

What is vertex cover of comparability graph?

Let $G = (V, A)$ be input. Then $G^{\text{comp}} := (V, E)$

where $(v, w) \in E$ if v and w are *comparable*.

G

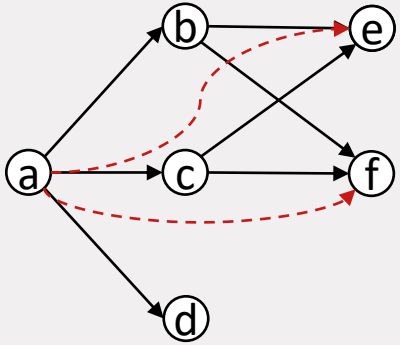


What is vertex cover of comparability graph?

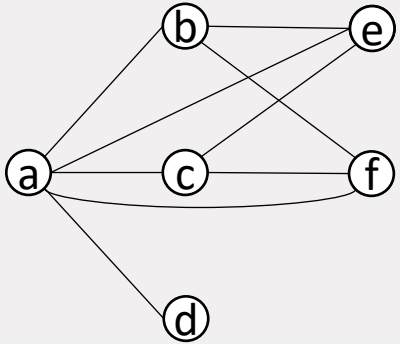
Let $G = (V, A)$ be input. Then $G^{\text{comp}} := (V, E)$

where $(v, w) \in E$ if v and w are *comparable*.

G



G^{comp}



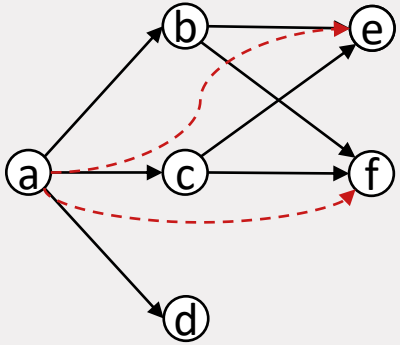
What is vertex cover of comparability graph?

Let $G = (V, A)$ be input. Then $G^{\text{comp}} := (V, E)$

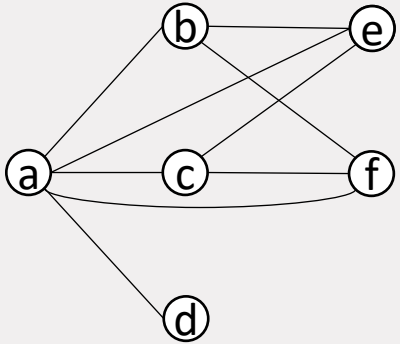
where $(v, w) \in E$ if v and w are *comparable*.

$C :=$ smallest vertex cover of G^{comp}

G



G^{comp}

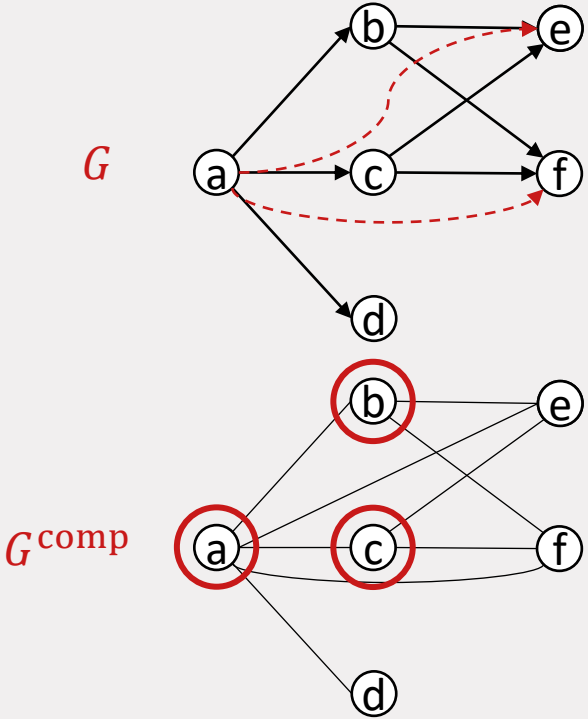


What is vertex cover of comparability graph?

Let $G = (V, A)$ be input. Then $G^{\text{comp}} := (V, E)$

where $(v, w) \in E$ if v and w are *comparable*.

$C :=$ smallest vertex cover of G^{comp}



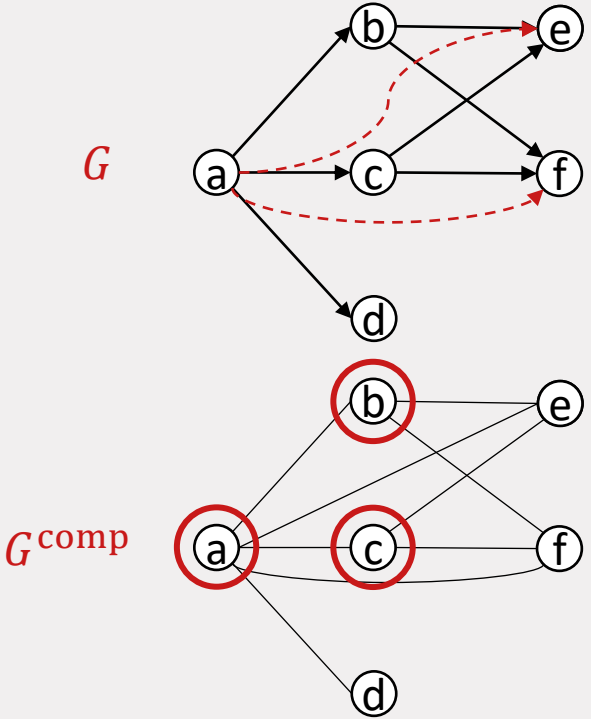
What is vertex cover of comparability graph?

Let $G = (V, A)$ be input. Then $G^{\text{comp}} := (V, E)$

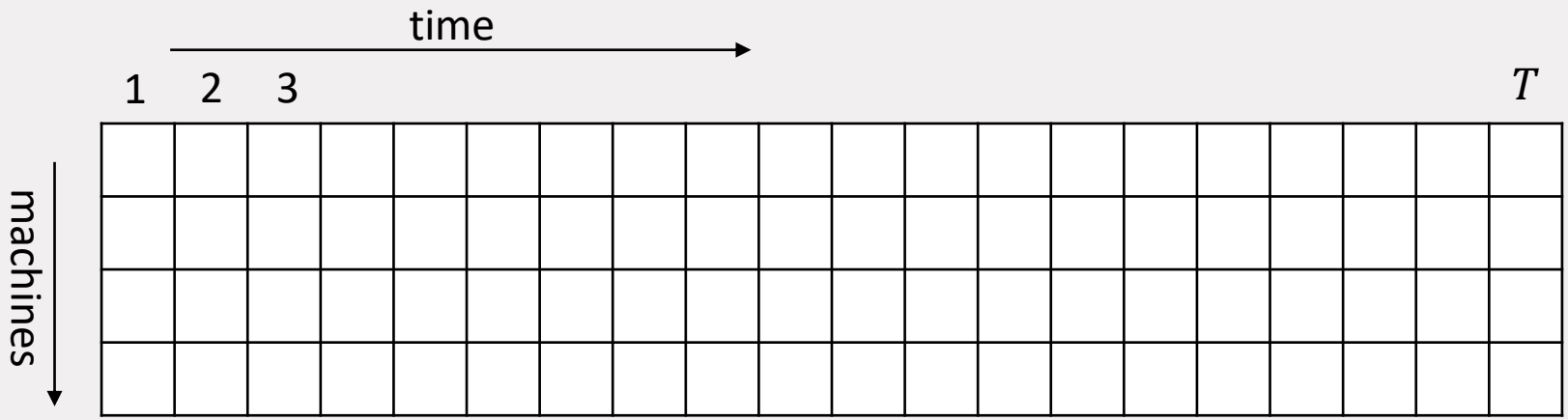
where $(v, w) \in E$ if v and w are *comparable*.

$C :=$ smallest vertex cover of G^{comp}

Claim: $V \setminus C$ is an antichain.

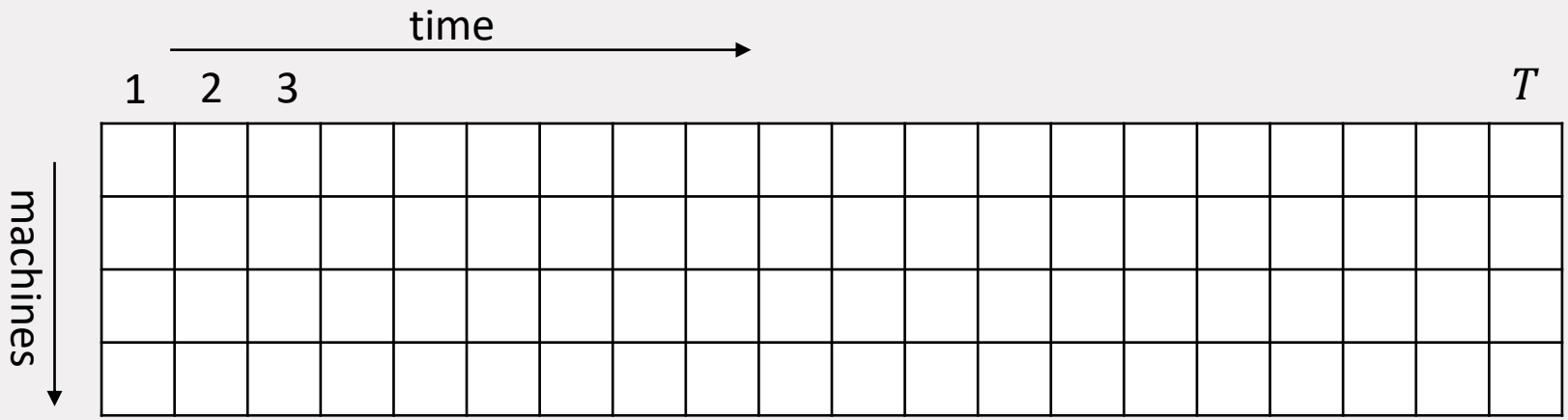


Sink-Adjusted Schedule



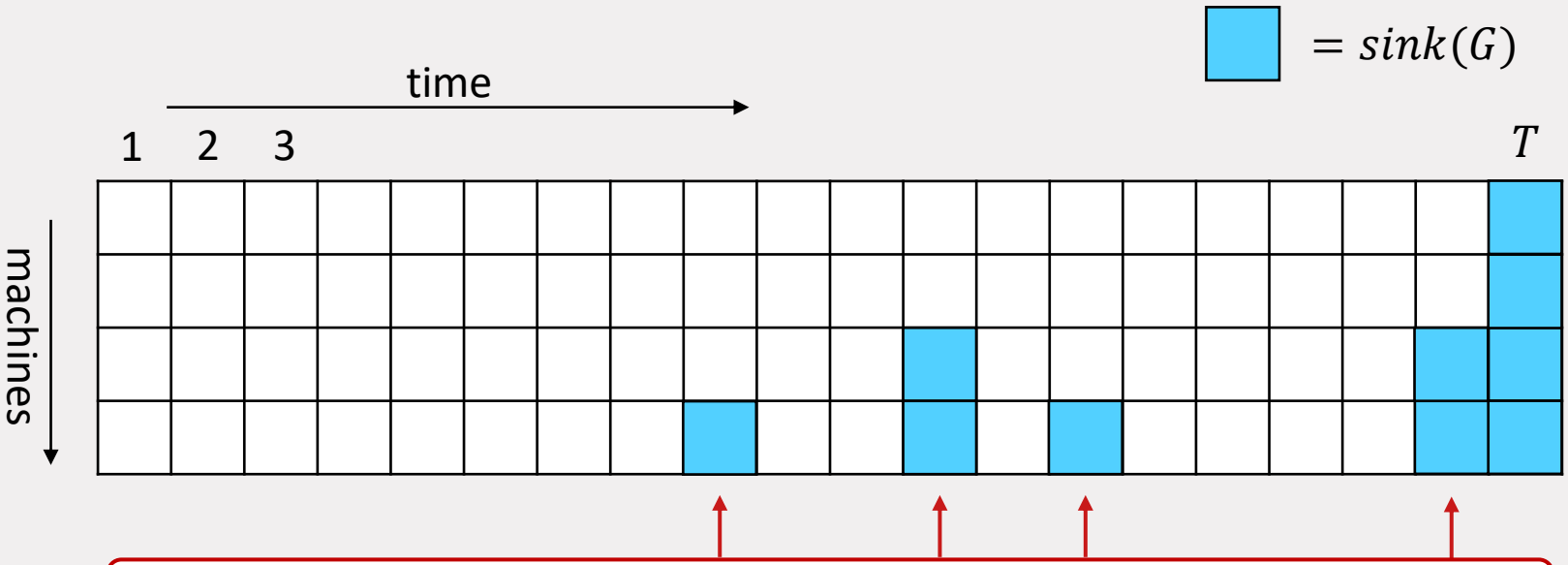
Sink-Adjusted Schedule

Assumption: $n = m \cdot T$



Sink-Adjusted Schedule

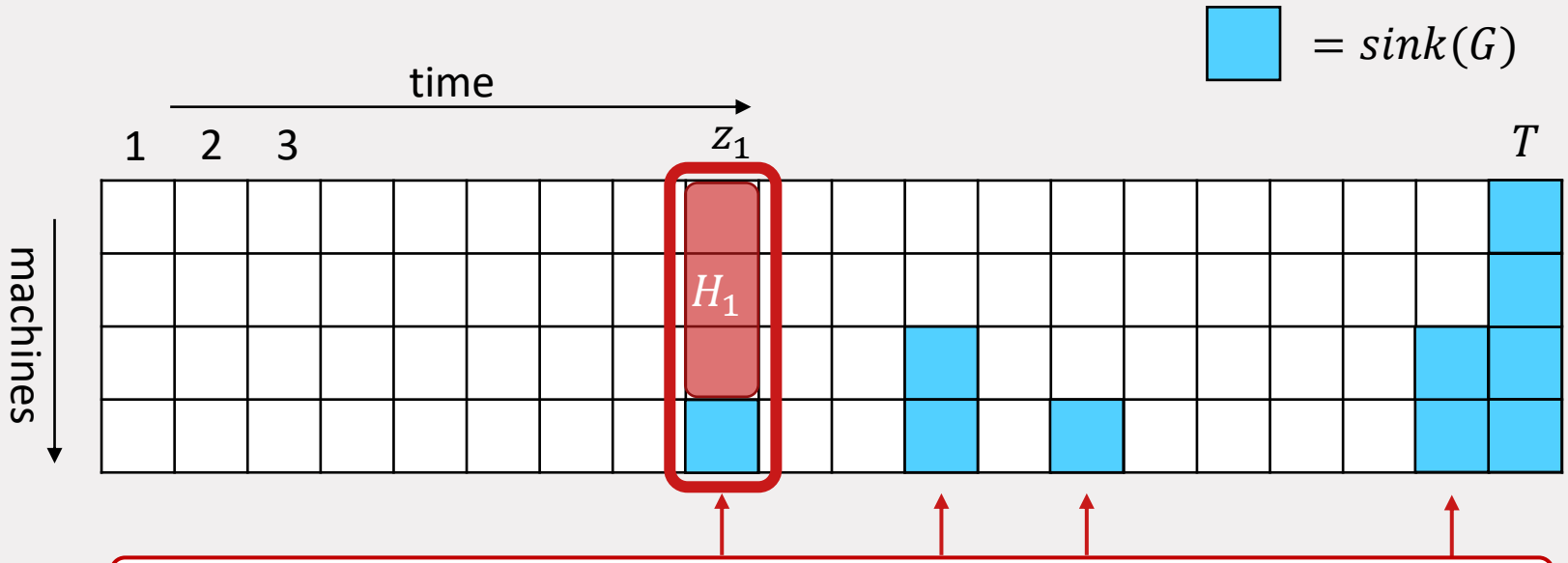
Assumption: $n = m \cdot T$



$z \in [1, T]$ is a ***sink moment*** if there are both sinks and non-sinks at time z.

Sink-Adjusted Schedule

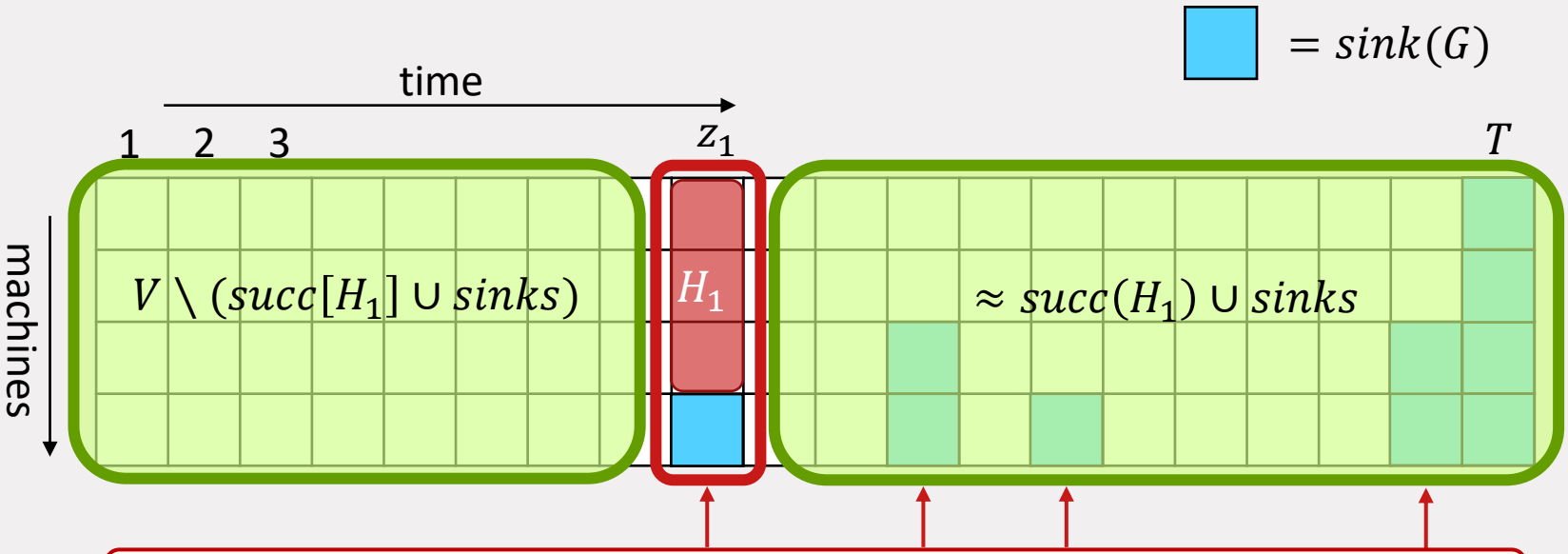
Assumption: $n = m \cdot T$



$z \in [1, T]$ is a ***sink moment*** if there are both sinks and non-sinks at time z.

Sink-Adjusted Schedule

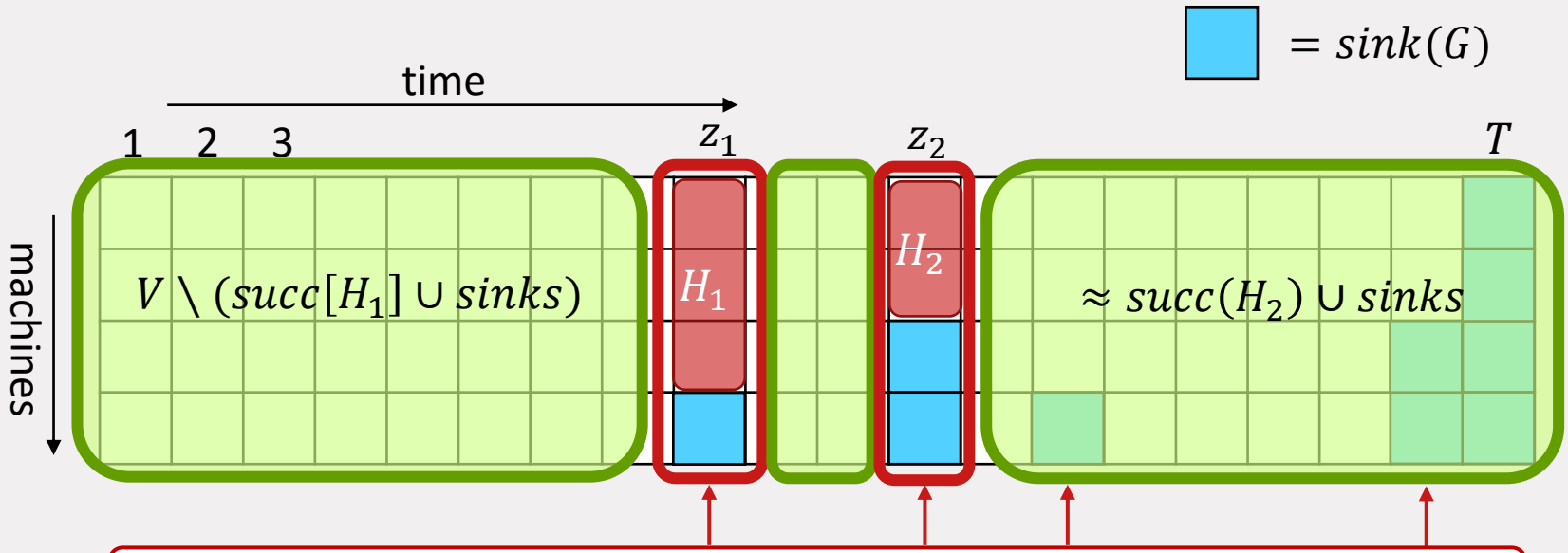
Assumption: $n = m \cdot T$



$z \in [1, T]$ is a ***sink moment*** if there are both sinks and non-sinks at time z.

Sink-Adjusted Schedule

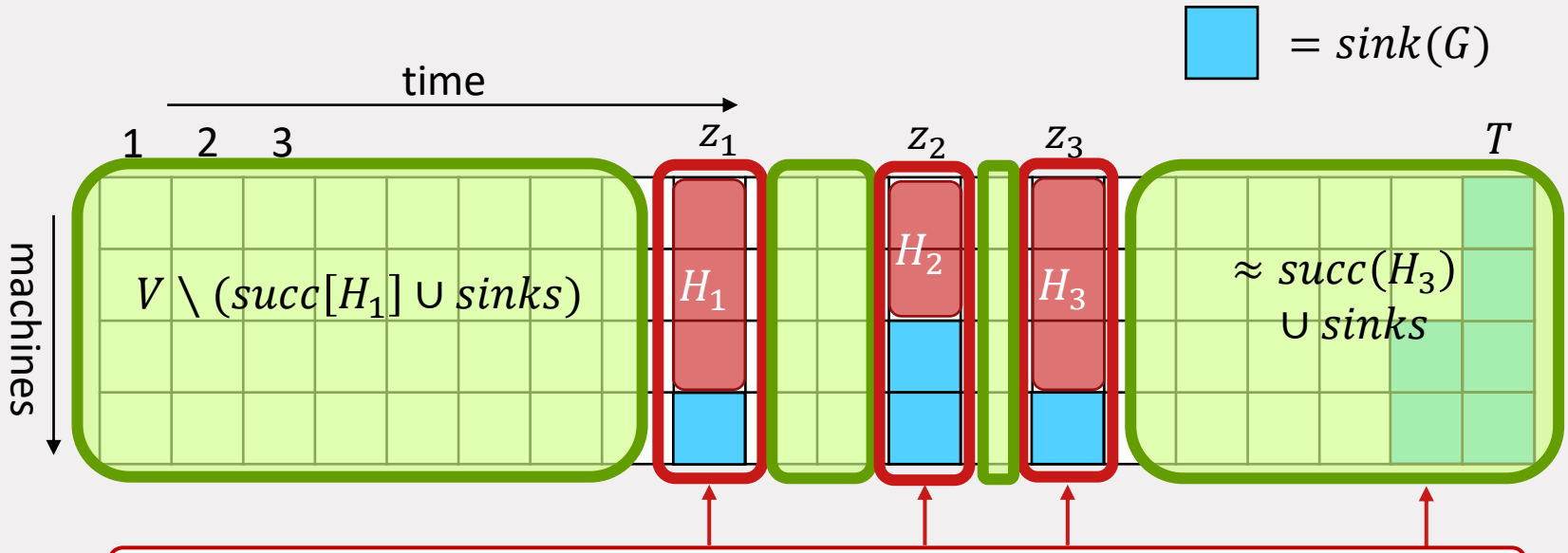
Assumption: $n = m \cdot T$



$z \in [1, T]$ is a ***sink moment*** if there are both sinks and non-sinks at time z.

Sink-Adjusted Schedule

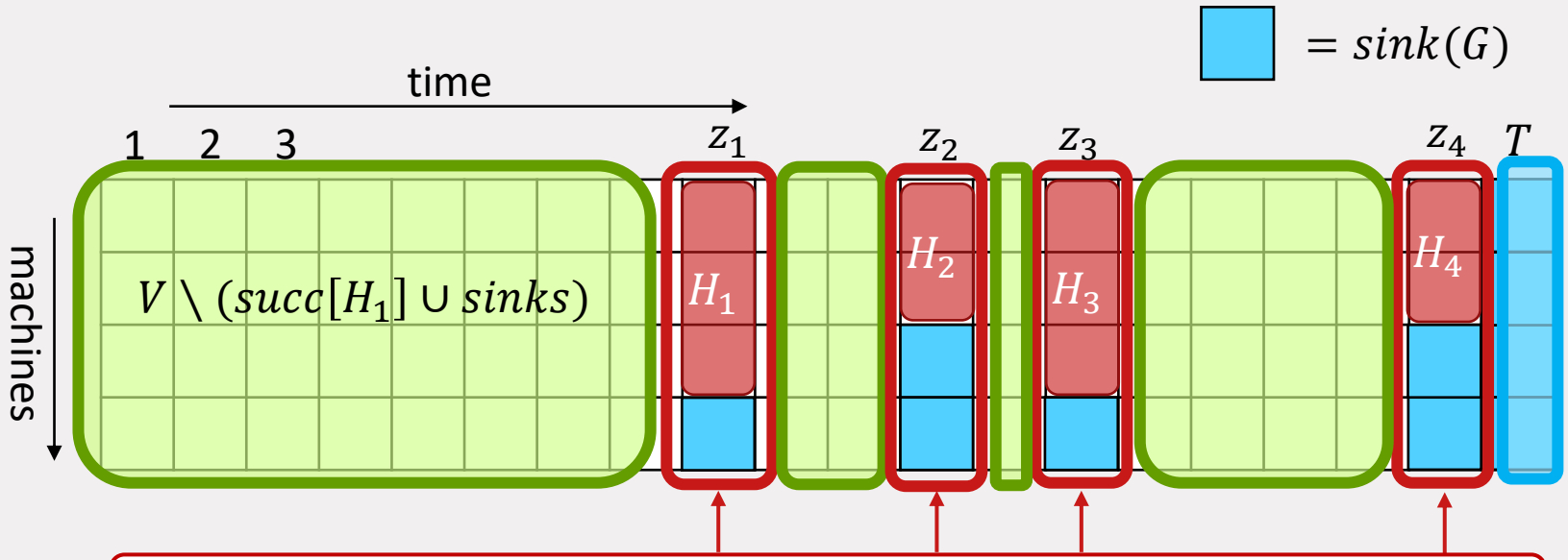
Assumption: $n = m \cdot T$



$z \in [1, T]$ is a ***sink moment*** if there are both sinks and non-sinks at time z.

Sink-Adjusted Schedule

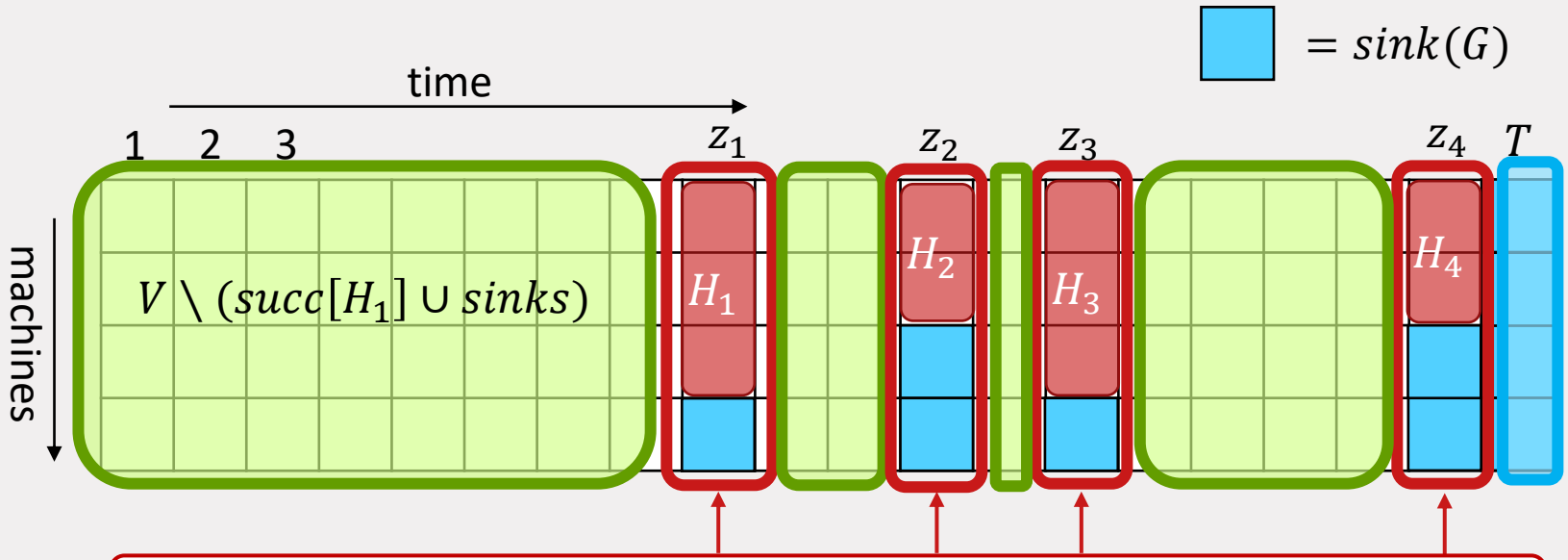
Assumption: $n = m \cdot T$



$z \in [1, T]$ is a ***sink moment*** if there are both sinks and non-sinks at time z.

Sink-Adjusted Schedule

Assumption: $n = m \cdot T$

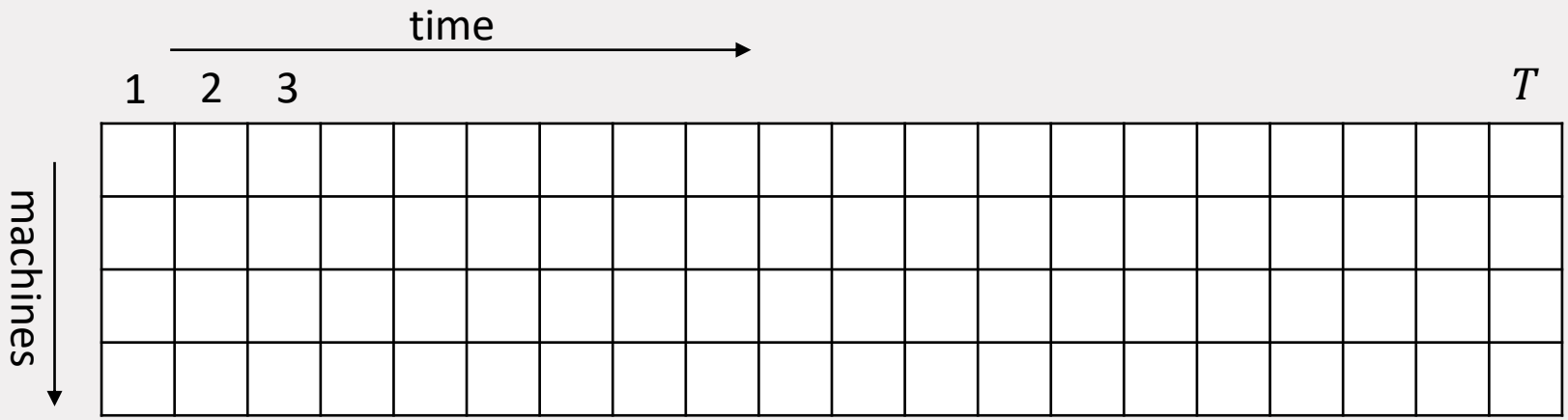


$z \in [1, T]$ is a ***sink moment*** if there are both sinks and non-sinks at time z.

No sinks in

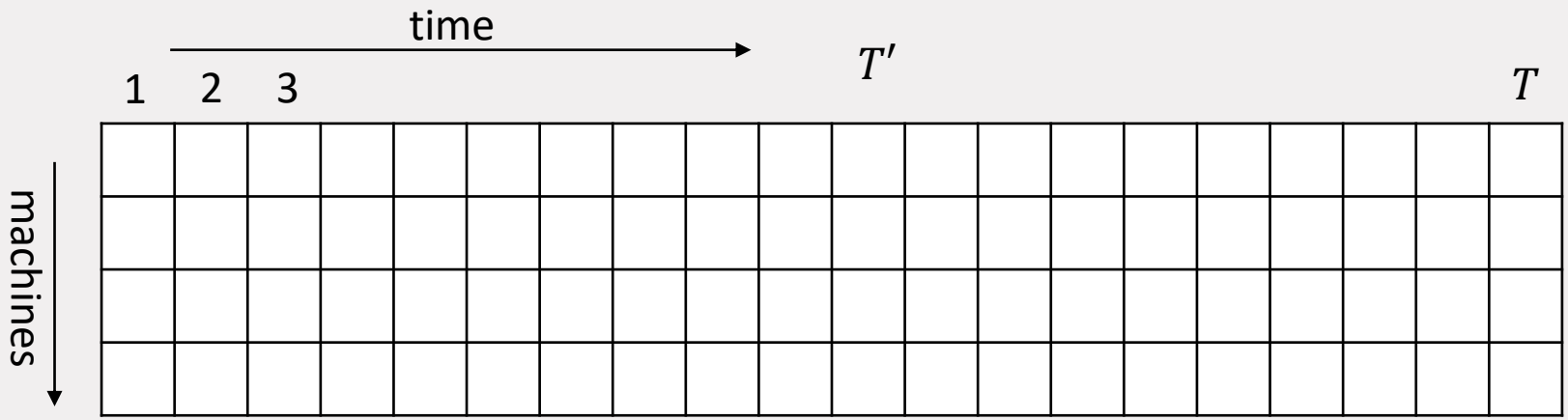
Middle-Adjusted Schedule

Assumption: $n = m \cdot T$



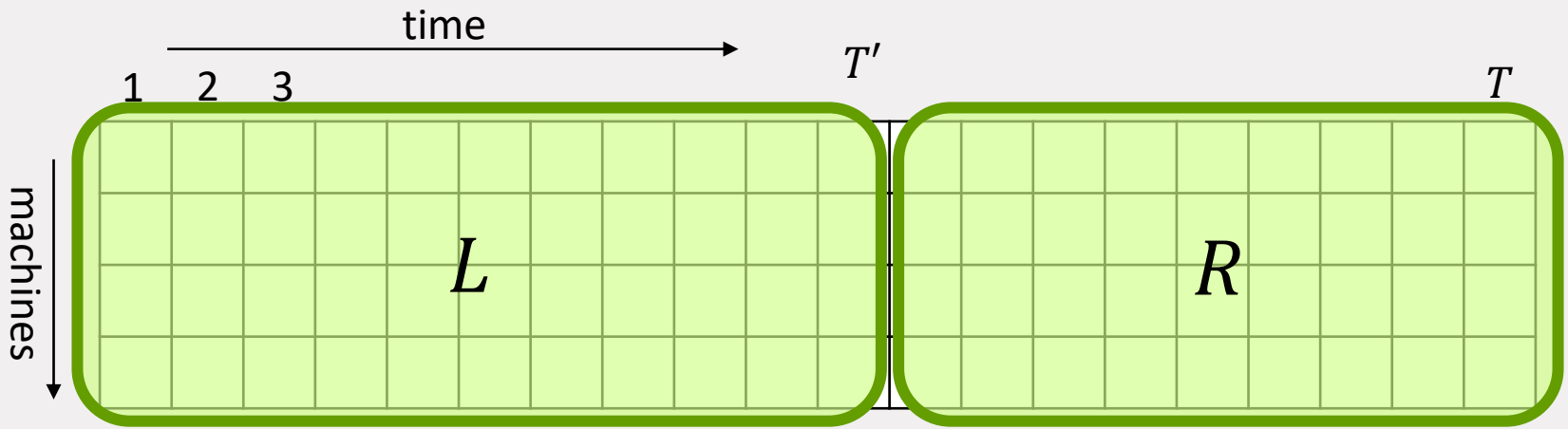
Middle-Adjusted Schedule

Assumption: $n = m \cdot T$



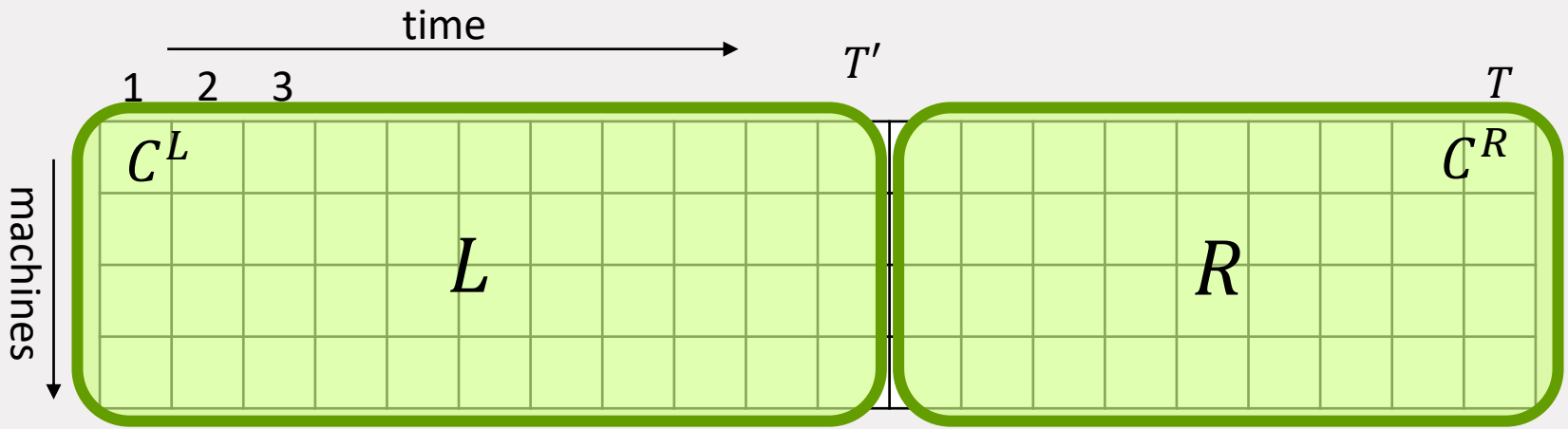
Middle-Adjusted Schedule

Assumption: $n = m \cdot T$



Middle-Adjusted Schedule

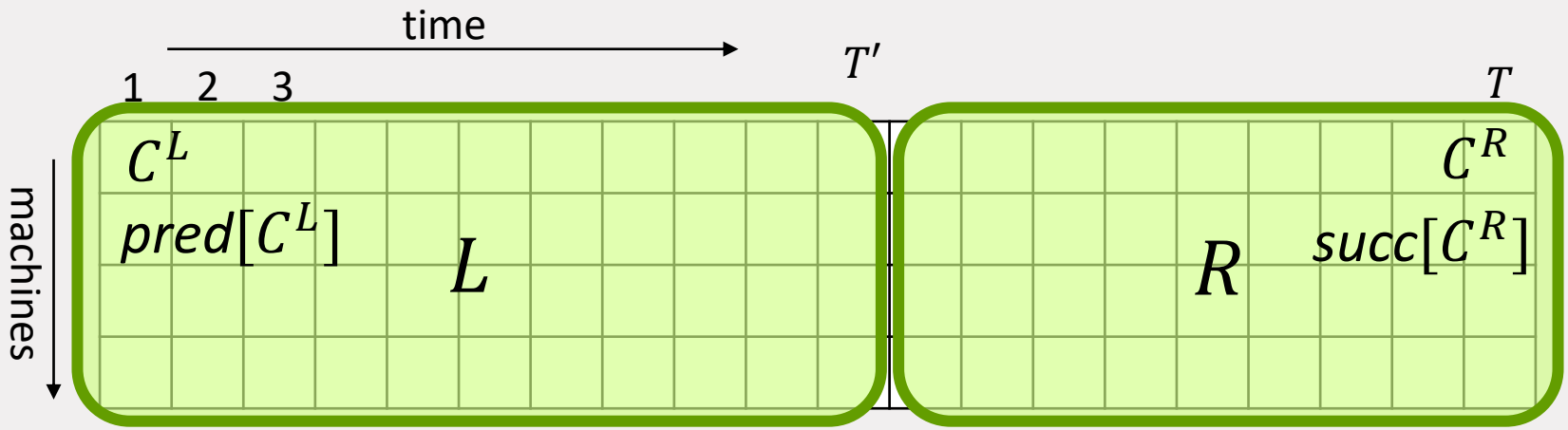
Assumption: $n = m \cdot T$



$$|C^L| \approx \frac{|C|}{2} \qquad |C^R| \approx \frac{|C|}{2}$$

Middle-Adjusted Schedule

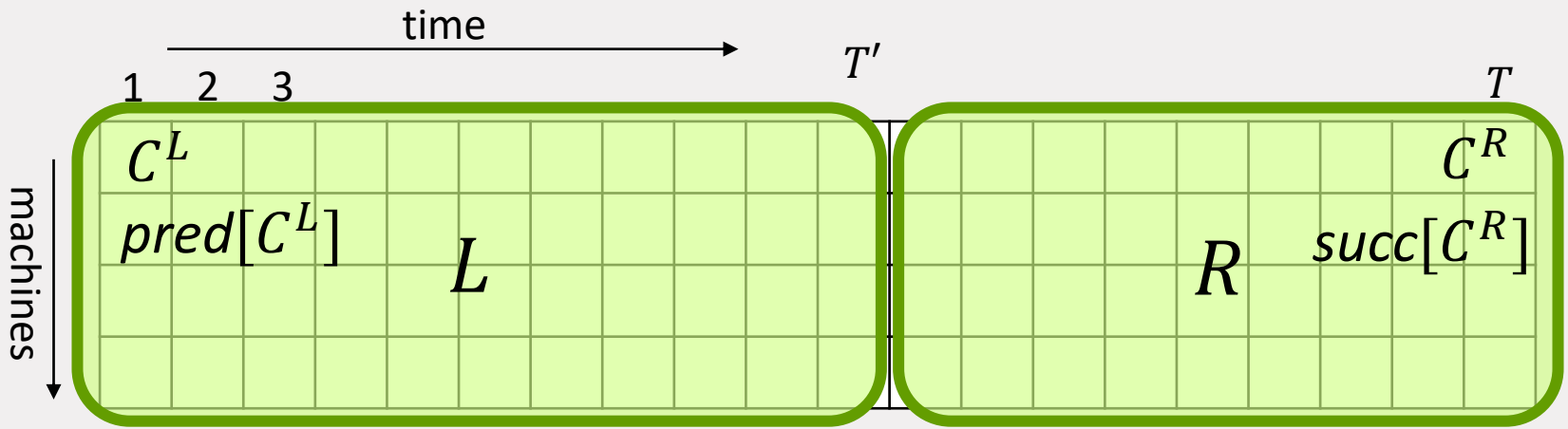
Assumption: $n = m \cdot T$



$$|C^L| \approx \frac{|C|}{2} \qquad |C^R| \approx \frac{|C|}{2}$$

Middle-Adjusted Schedule

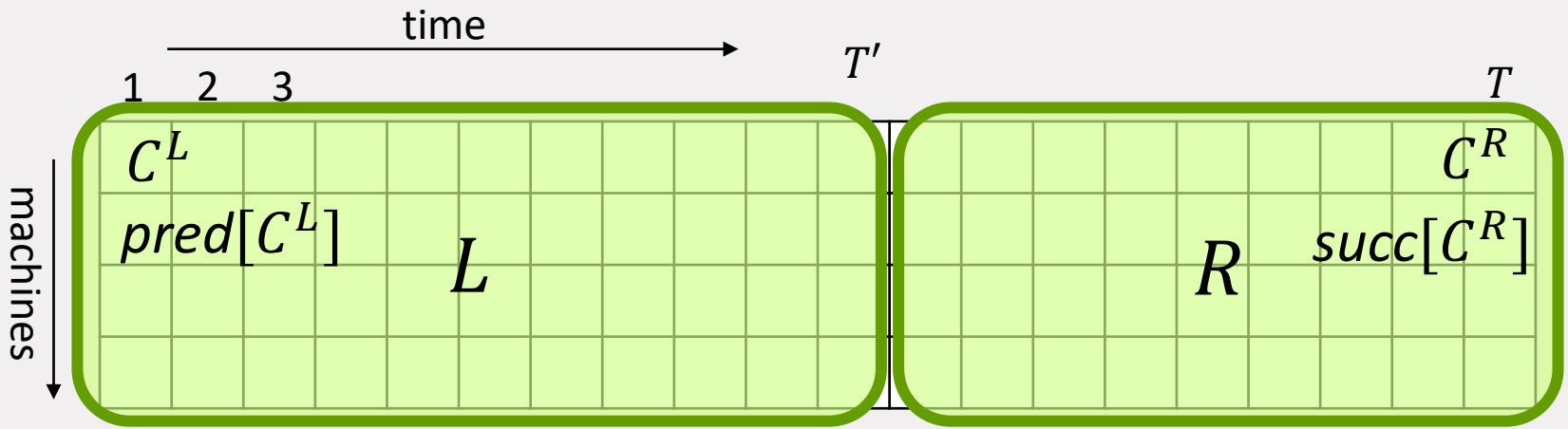
Assumption: $n = m \cdot T$



Undediced $U := V \setminus (pred[C^L] \cup succ[C^R])$

Middle-Adjusted Schedule

Assumption: $n = m \cdot T$



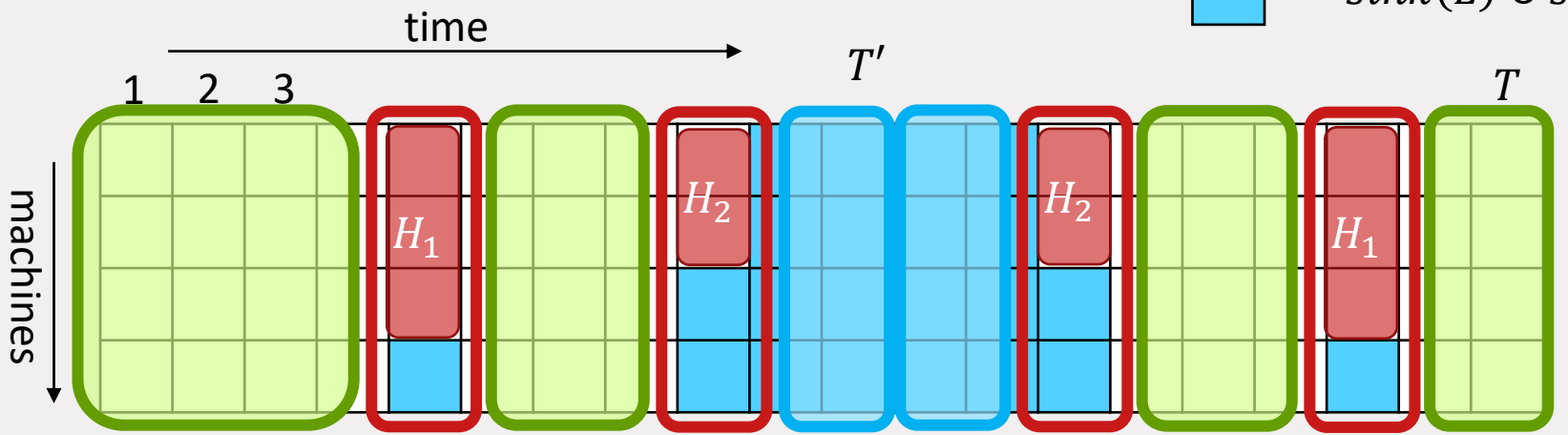
Undediced $U := V \setminus (pred[C^L] \cup succ[C^R])$

- U is antichain
- U are sinks in L
- U are sources in R

Algorithm

Assumption: $n = m \cdot T$

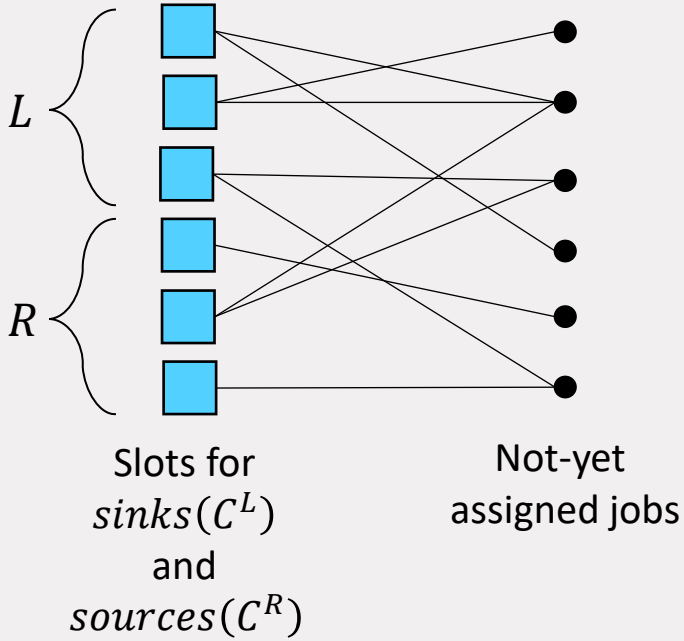
$= sink(L) \cup sink(R)$



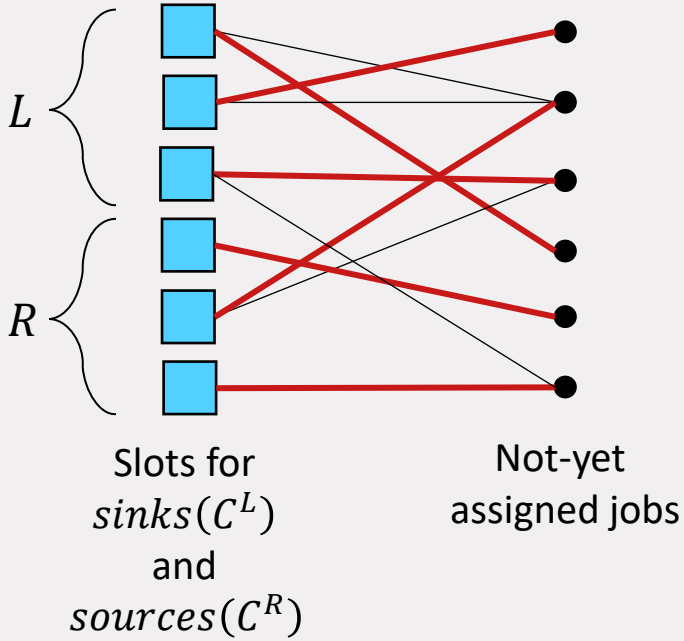
Undediced $U := V \setminus (pred[C^L] \cup succ[C^R])$

- U is antichain
- U are sinks in L
- U are sources in R

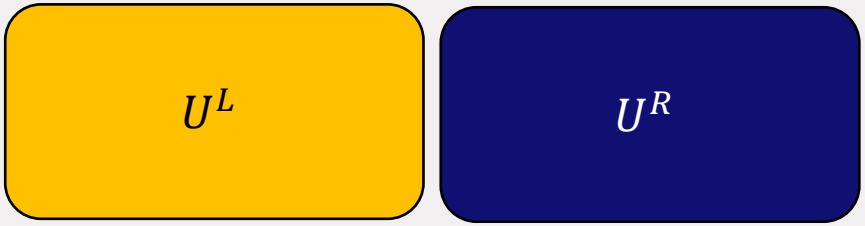
Algorithm



Algorithm

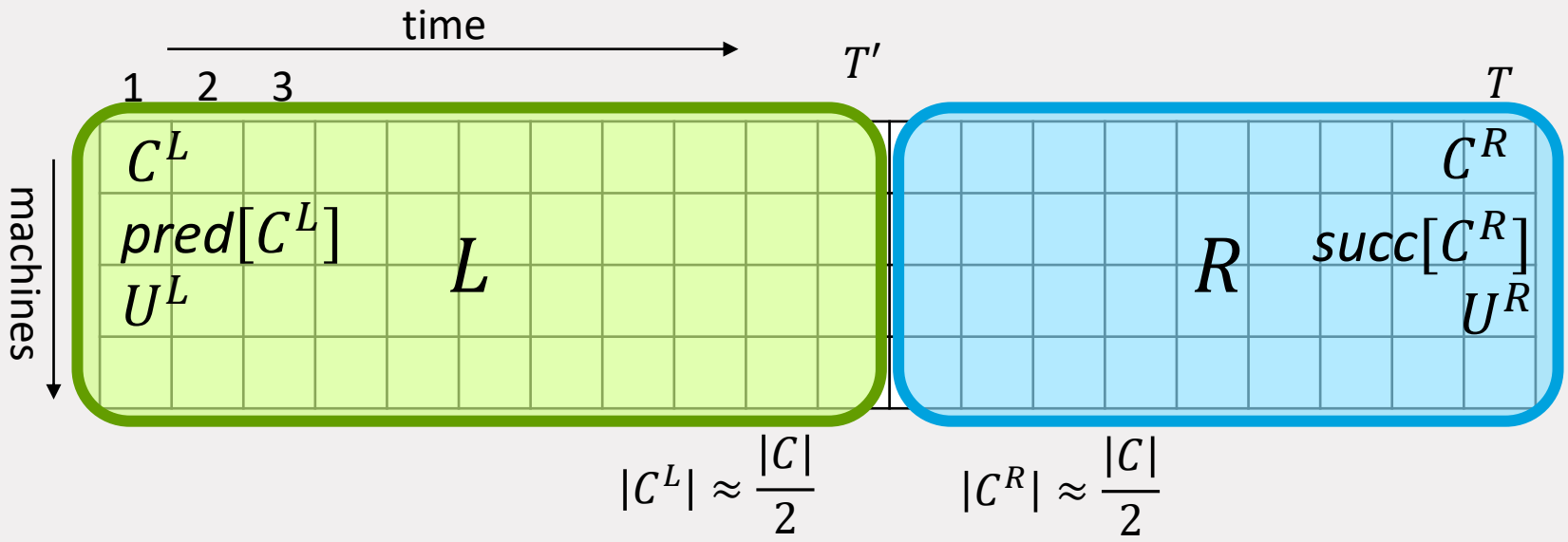


Find partition of U:



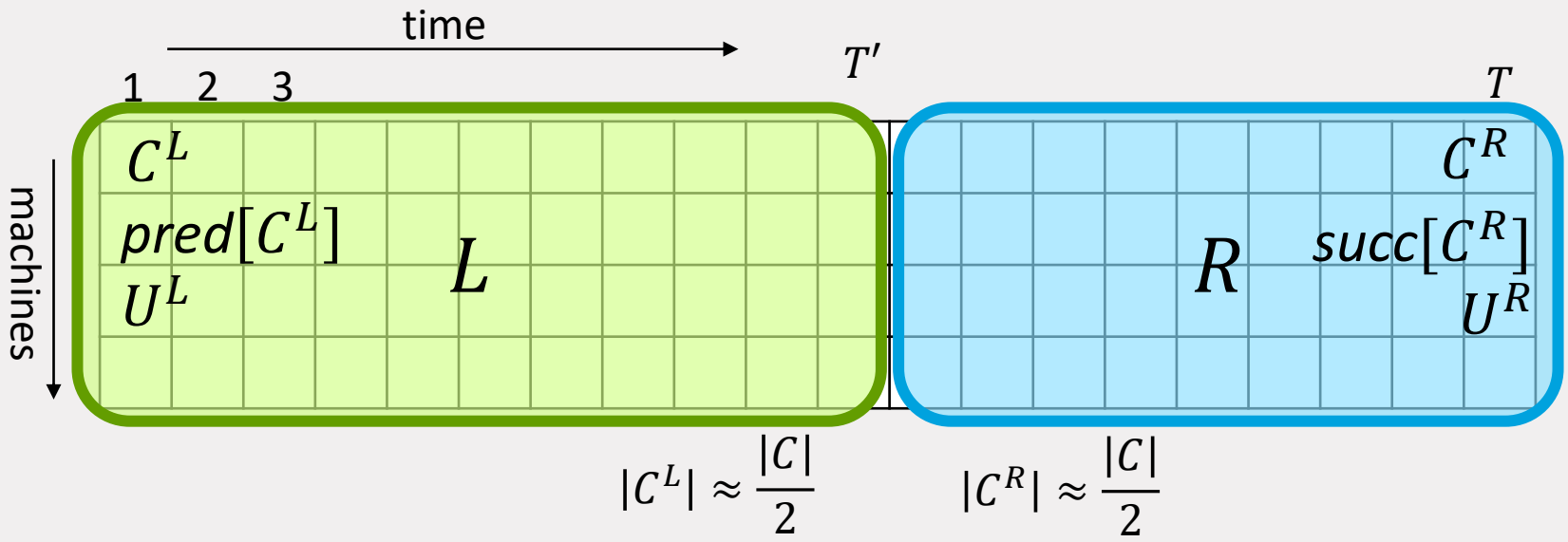
Algorithm

Assumption: $n = m \cdot T$



Algorithm

Assumption: $n = m \cdot T$



Guesses: $13^{|C|} \Rightarrow O(poly(n) \cdot 169^{|C|})$ time.

Conclusion

Conclusion

Main result:

$P|prec, p_j = 1|C_{\max}$ in $O(1.995^n)$ time.

Conclusion

Main result:

$P|prec, p_j = 1|C_{\max}$ in $O(1.995^n)$ time.

Key idea's:

- Tradeoff between $\#AC's$ and $|C|$.
- $O\left(poly(n) \cdot 169^{|C|}\right)$ time algorithm

Conclusion

Main result:

$P|prec, p_j = 1|C_{\max}$ in $O(1.995^n)$ time.

Key idea's:

- Tradeoff between $\#AC's$ and $|C|$.
- $O\left(poly(n) \cdot 169^{|C|}\right)$ time algorithm

Future Research:

$P3|prec, p_j = 1|C_{\max}$ in $2^{o(n)}$ time?

Conclusion

Main result:

$P|prec, p_j = 1|C_{\max}$ in $O(1.995^n)$ time.

Key idea's:

- Tradeoff between $\#AC's$ and $|C|$.
- $O\left(poly(n) \cdot 169^{|C|}\right)$ time algorithm

Future Research:

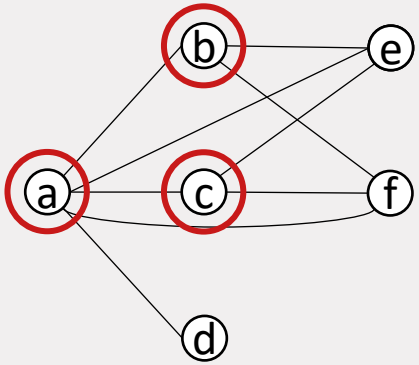
$P3|prec, p_j = 1|C_{\max}$ in $2^{o(n)}$ time?

Thanks for your attention!

Size of C vs #AC's

Claim: $|C| \geq n/7.5 \;\Rightarrow\; \#AC's \leq 1.94^n$

G^{comp}



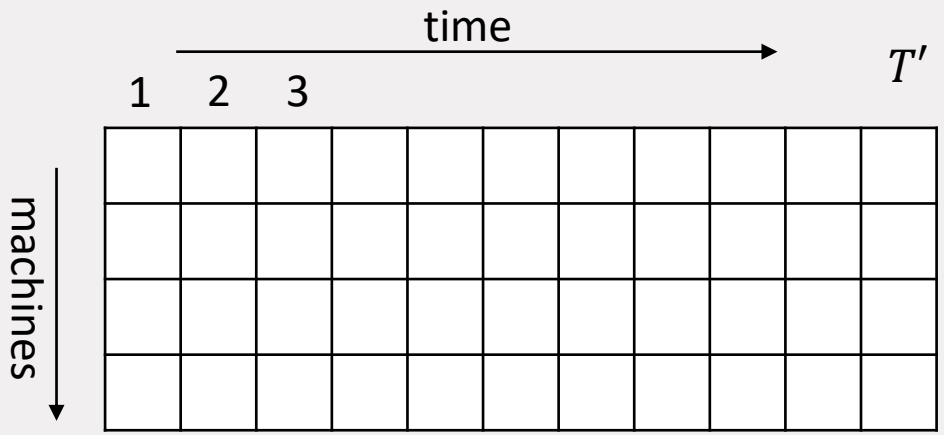
Proof. $|C| \geq n/7.5 \Rightarrow$ largest AC $\leq n - \frac{n}{7.5} = \alpha n$ with $\alpha = \left(1 - \frac{1}{7.5}\right)$.

Dilworth's Theorem: $\exists$ chains $C_1, \ldots, C_{\alpha n}$ that partition G.

$$\#\text{AC's} = \prod_{i=1}^{\alpha n} (|C_i| + 1) \leq \left(\frac{n}{\alpha n} + 1\right)^{\alpha n} = 1.945^n$$

Algorithm

Assumption: $n = m \cdot T$

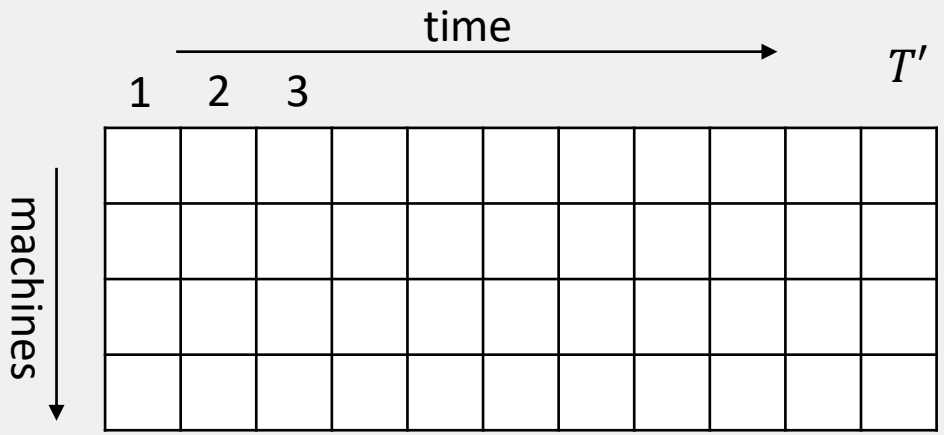


Undediced $U \coloneqq V \setminus (pred[C^L] \cup succ[C^R])$

- U is antichain
- U are sinks in L

Algorithm

Assumption: $n = m \cdot T$



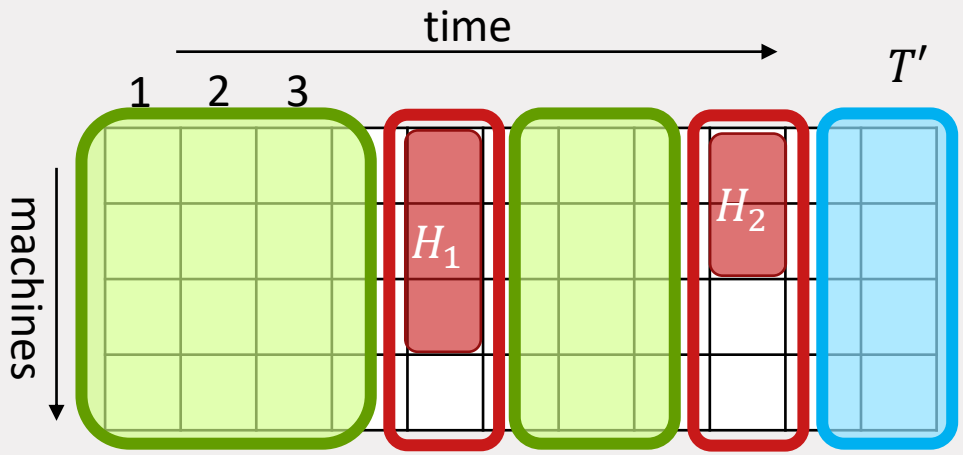
- Assume sink-adjusted

Undediced $U \coloneqq V \setminus (pred[C^L] \cup succ[C^R])$

- U is antichain
- U are sinks in L

Algorithm

Assumption: $n = m \cdot T$



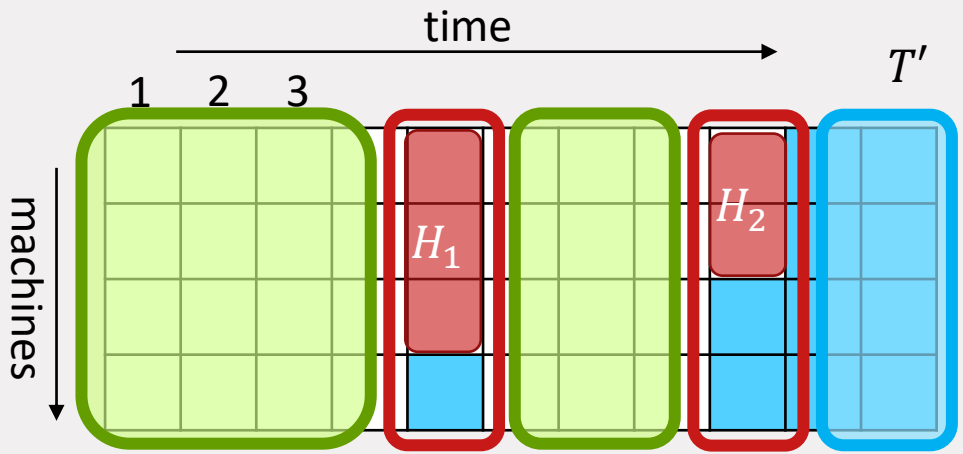
- Assume sink-adjusted

Undediced $U := V \setminus (pred[C^L] \cup succ[C^R])$

- U is antichain
- U are sinks in L

Algorithm

Assumption: $n = m \cdot T$

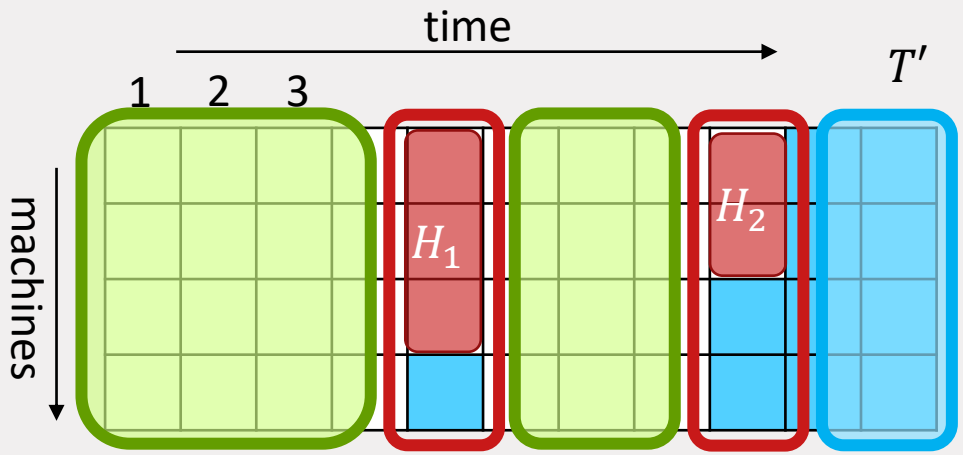


Undediced $U := V \setminus (pred[C^L] \cup succ[C^R])$

- U is antichain
- U are sinks in L

Algorithm

Assumption: $n = m \cdot T$



Undediced $U := V \setminus (pred[C^L] \cup succ[C^R])$

- U is antichain
- U are sinks in L

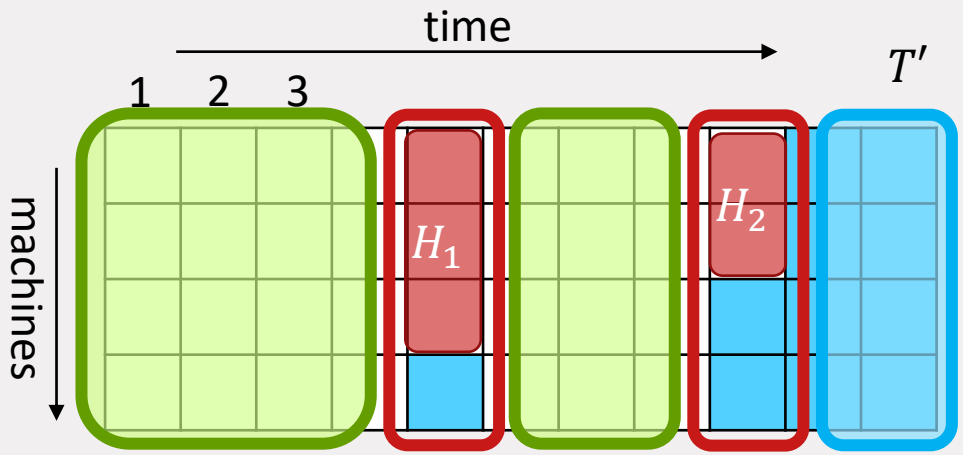
- Assume sink-adjusted

Assume that $H_1, H_2, \ldots \subseteq C^L$

- Guess jobs in H from C^L
- Reconstruct $H_1, H_2, \ldots$
- Reconstruct
- Distribute sinks

Algorithm

Assumption: $n = m \cdot T$



Undediced $U := V \setminus (pred[C^L] \cup succ[C^R])$

- U is antichain
- U are sinks in L

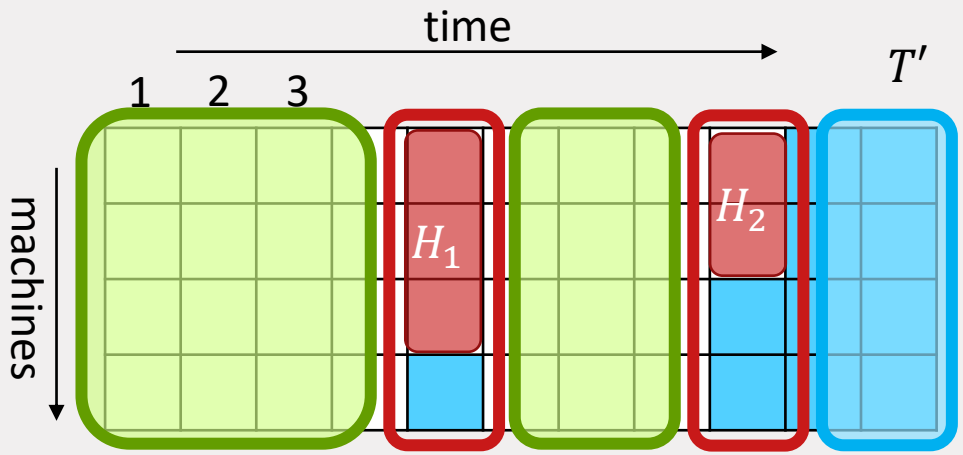
- Assume sink-adjusted

Assume that $H_1, H_2, \ldots \subseteq C^L$

- Guess jobs in H from C^L
- Reconstruct $H_1, H_2, \ldots$
- Reconstruct
- Distribute sinks

Algorithm

Assumption: $n = m \cdot T$



Undediced $U := V \setminus (pred[C^L] \cup succ[C^R])$

- U is antichain
- U are sinks in L

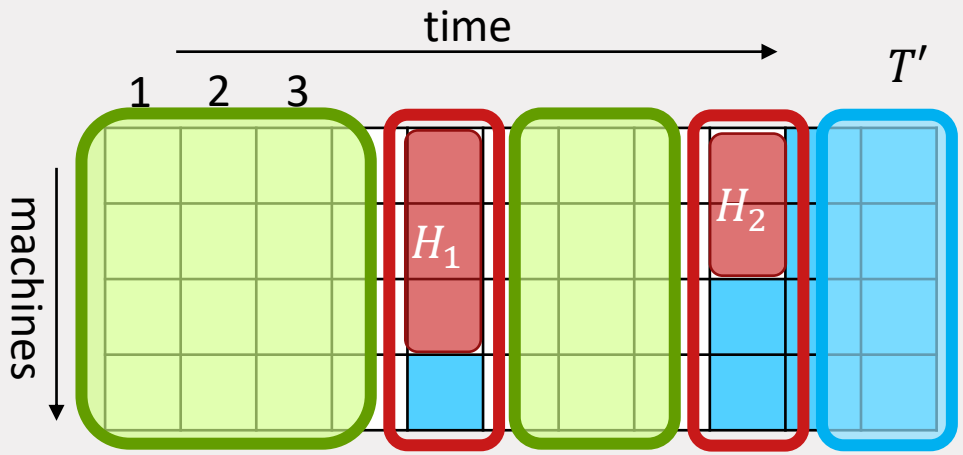
- Assume sink-adjusted

Assume that $H_1, H_2, \ldots \subseteq C^L$

- Guess jobs in H from C^L
- Reconstruct $H_1, H_2, \ldots$
- Reconstruct
- Distribute sinks

Algorithm

Assumption: $n = m \cdot T$



Undediced $U := V \setminus (pred[C^L] \cup succ[C^R])$

- U is antichain
- U are sinks in L

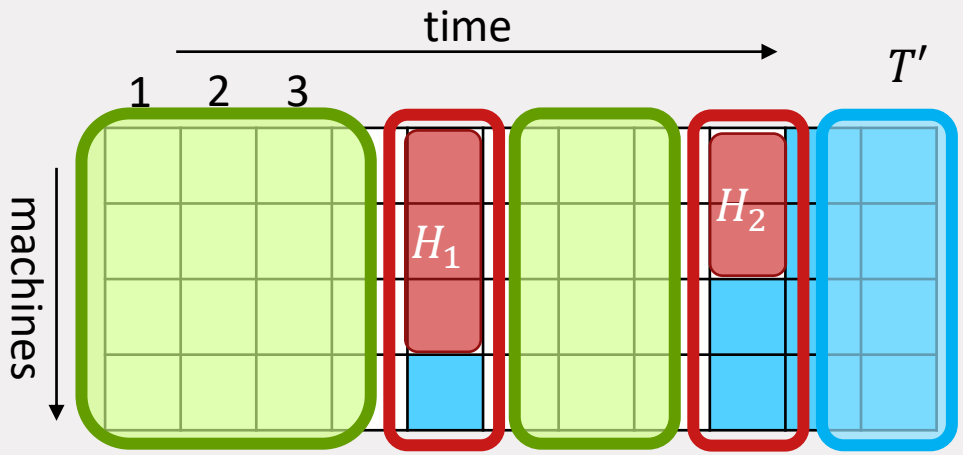
- Assume sink-adjusted

Assume that $H_1, H_2, \ldots \subseteq C^L$

- Guess jobs in H from C^L
- Reconstruct $H_1, H_2, \ldots$
- Reconstruct
- Distribute sinks

Algorithm

Assumption: $n = m \cdot T$



Undediced $U := V \setminus (pred[C^L] \cup succ[C^R])$

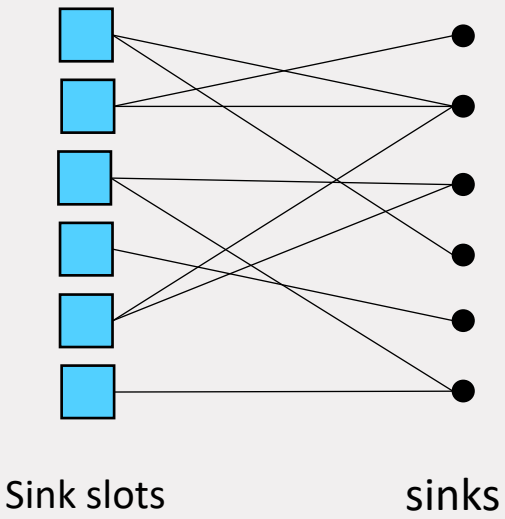
- U is antichain
- U are sinks in L

- Assume sink-adjusted

Assume that $H_1, H_2, \ldots \subseteq C^L$

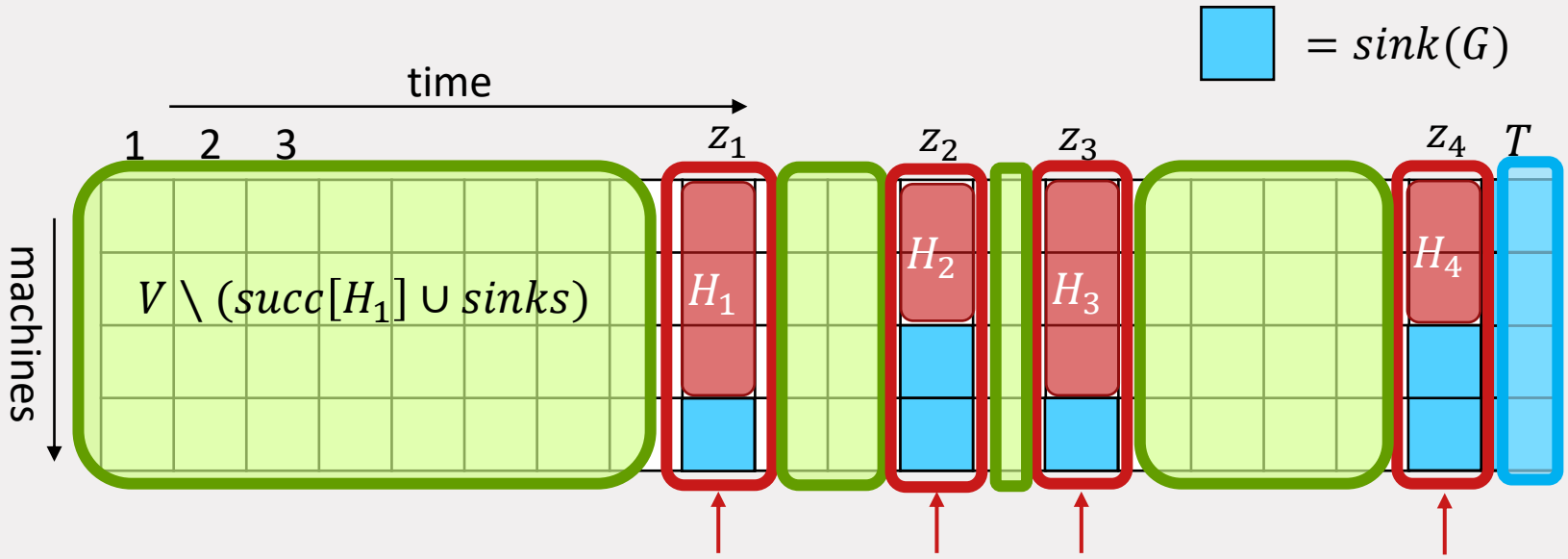
- Guess jobs in H from C^L
- Reconstruct $H_1, H_2, \ldots$
- Reconstruct
- Distribute sinks

Algorithm



Sink-Adjusted Schedule

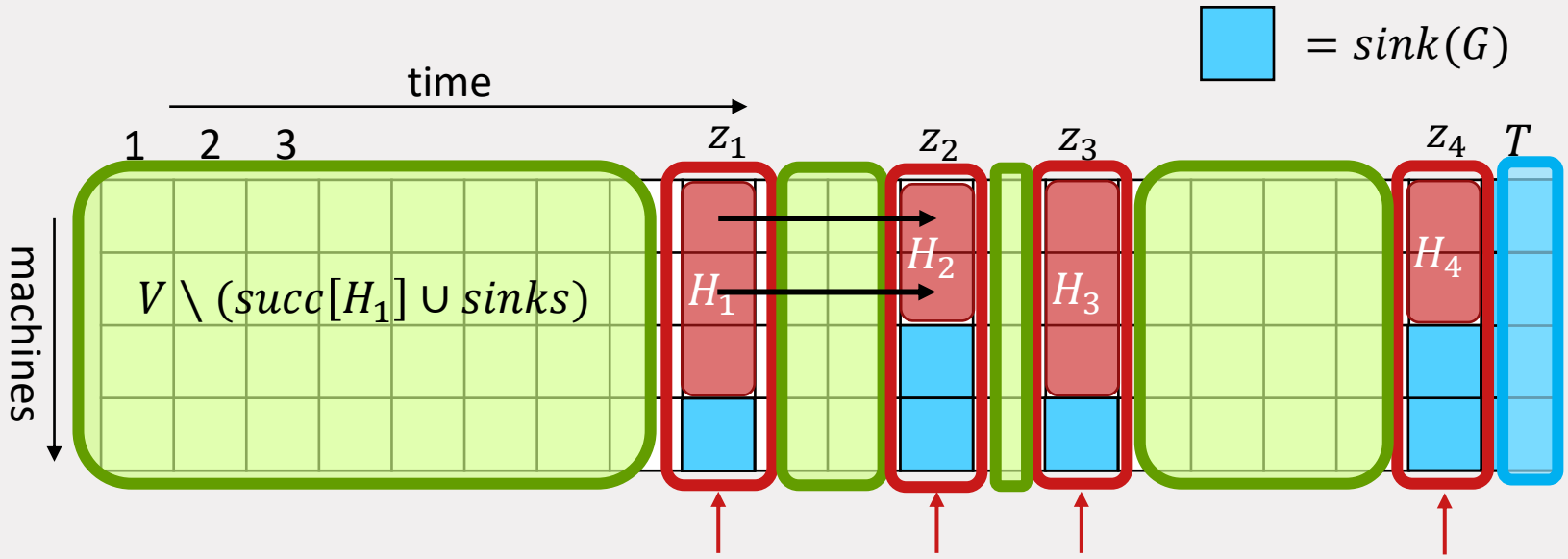
Assumption: $n = m \cdot T$



Claim: if we know $\cup_i H_i$, we can derive $H_1, H_2, \ldots, H_l$

Sink-Adjusted Schedule

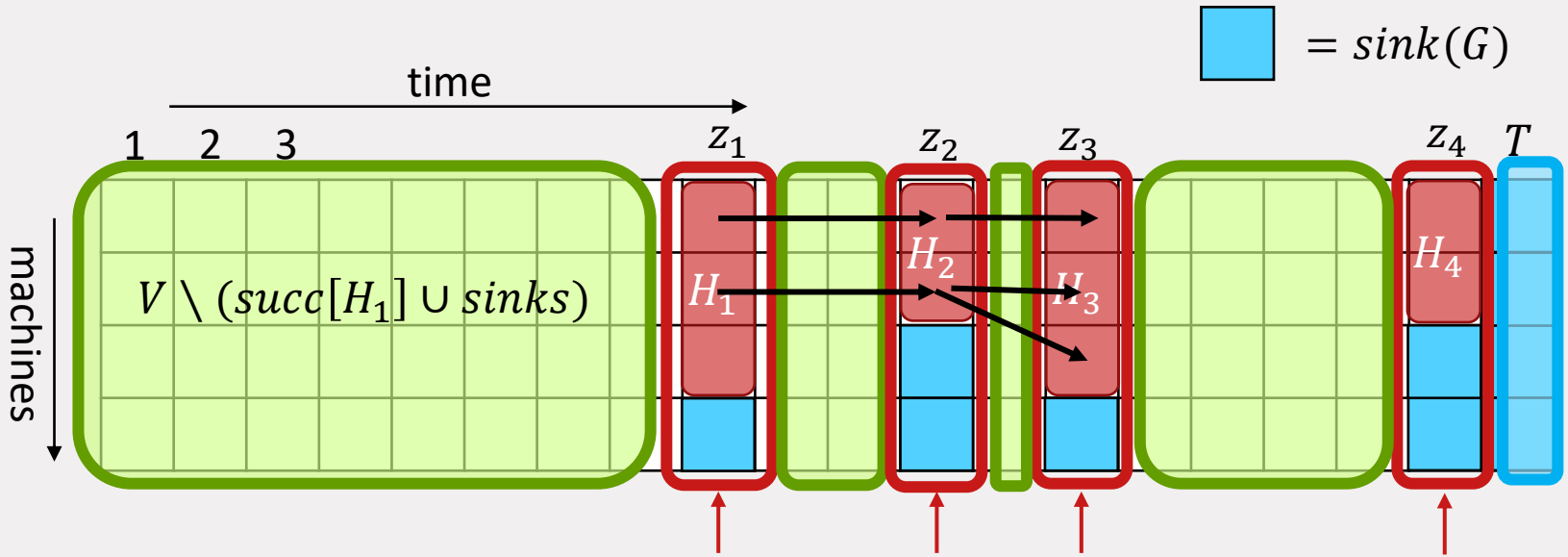
Assumption: $n = m \cdot T$



Claim: if we know $\cup_i H_i$, we can derive $H_1, H_2, \ldots, H_l$

Sink-Adjusted Schedule

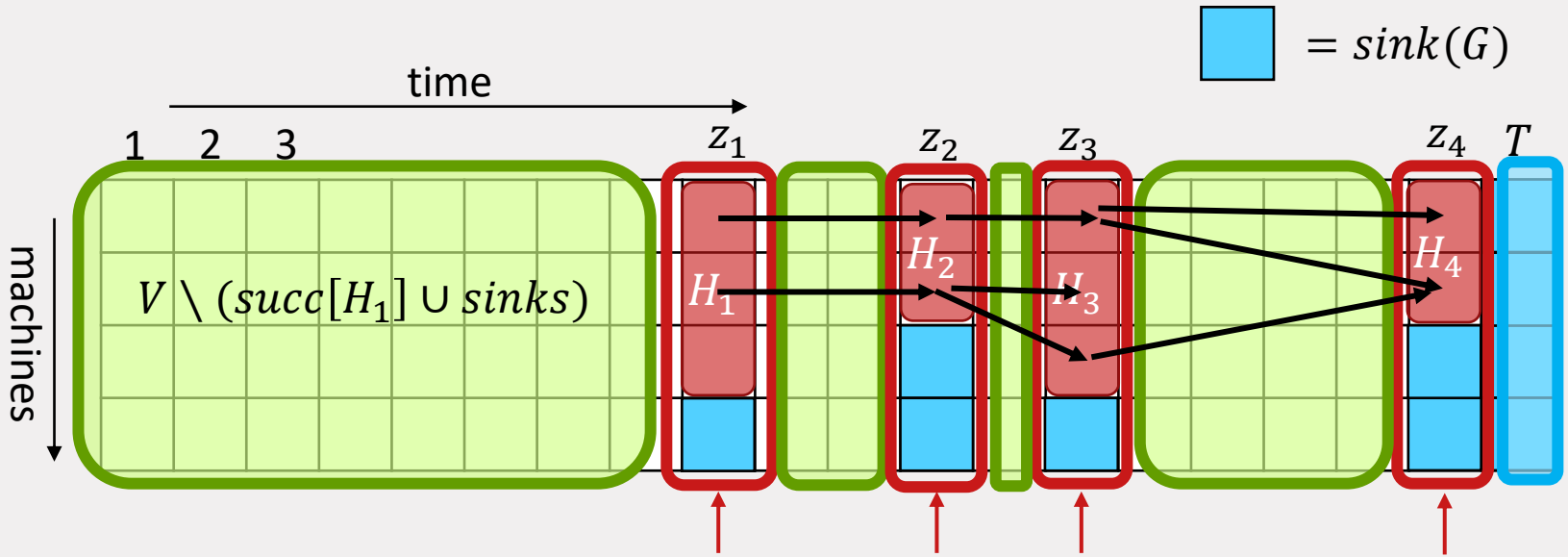
Assumption: $n = m \cdot T$



Claim: if we know $\cup_i H_i$, we can derive $H_1, H_2, \ldots, H_l$

Sink-Adjusted Schedule

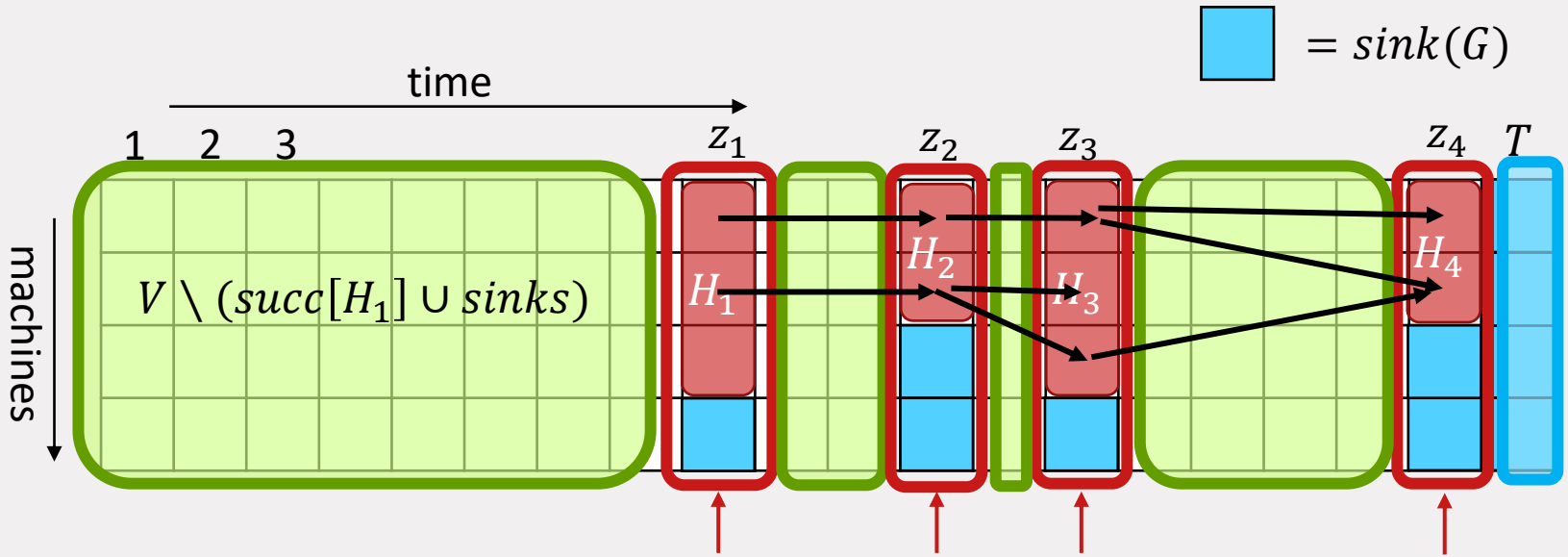
Assumption: $n = m \cdot T$



Claim: if we know $\cup_i H_i$, we can derive $H_1, H_2, \ldots, H_l$

Sink-Adjusted Schedule

Assumption: $n = m \cdot T$

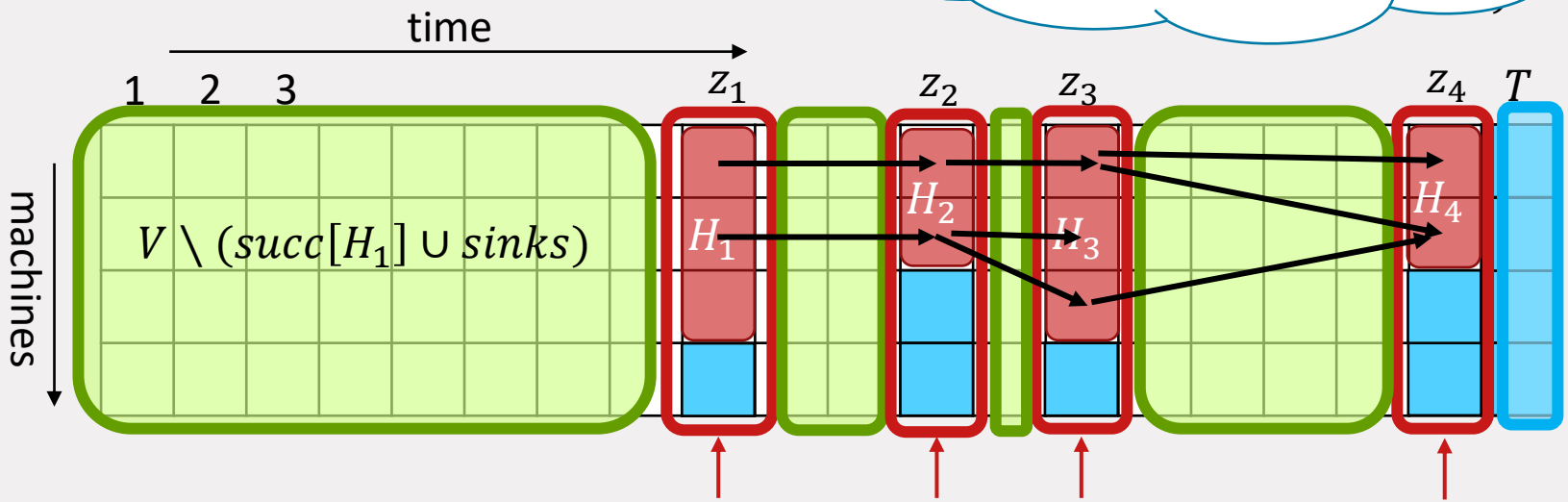


Claim: if we know $\cup_i H_i$, we can derive $H_1, H_2, \ldots, H_l$

$\Rightarrow$ for each we know which jobs are in there.

Sink-Adjusted Schedule

How to use this…?



Claim: if we know $\cup_i H_i$, we can derive $H_1, H_2, \ldots, H_l$

$\Rightarrow$ for each we know which jobs are in there.