Graph Evacuation Problems

Mordecai GOLIN
Hong Kong UST

CRM June, 2015

Joint Work with

- *Guru Prakash Arumugam*
- *John Augustine*
- *Di Chen*
- *Siu-Wing Cheng*
- *Yuya Higashikawa*
- *Naoki Katoh*
- *Guanqun Ni*
- *Bing Su*
- *Prashanth Srikanthan*
- *Yinfeng Xu*

Outline

- Dynamic Flow Networks
- Congestion in Dynamic Flows
- Evacuation Flows
  - Problem Definitions
  - Known Results
- Example Algorithm 1: k-Sink Evacuation on a Path
- Example Algorithm 2: 1-sink Min-Max Regret Evacuation on a Path with uniform capacity
- Open Problems

Evacuating Graphs

Evacuating Graphs

- Graph G=(V,E) represents structure
  - Vertices are rooms, Edges are Hallways
  - Vertices are Buildings, Edges are roads
  - Edge weight τ_e is transit time on edge
  - Edge capacity c_e is "width"

Evacuating Graphs

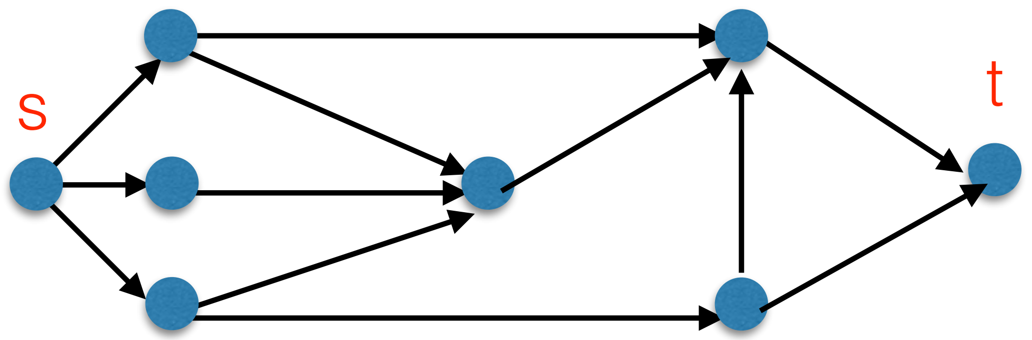
- Graph G=(V,E) represents structure
  - Vertices are rooms, Edges are Hallways
  - Vertices are Buildings, Edges are roads
  - Edge weight τ_e is transit time on edge
  - Edge capacity c_e is "width"
- Special vertices (sinks) are emergency exits
  - In case of emergency, want to evacuate everybody to exits as quickly as possible
  - Problem: Design Good Evacuation Protocols

Evacuating Graphs

- Graph G=(V,E) represents structure
  - Vertices are rooms, Edges are Hallways
  - Vertices are Buildings, Edges are roads
  - Edge weight τ_e is transit time on edge
  - Edge capacity c_e is “width”
- Special vertices (sinks) are emergency exits
  - In case of emergency, want to evacuate everybody to exits as quickly as possible
  - Problem: Design Good Evacuation Protocols
- Often Approached via Dynamic Flow Networks

Dynamic Flow Networks

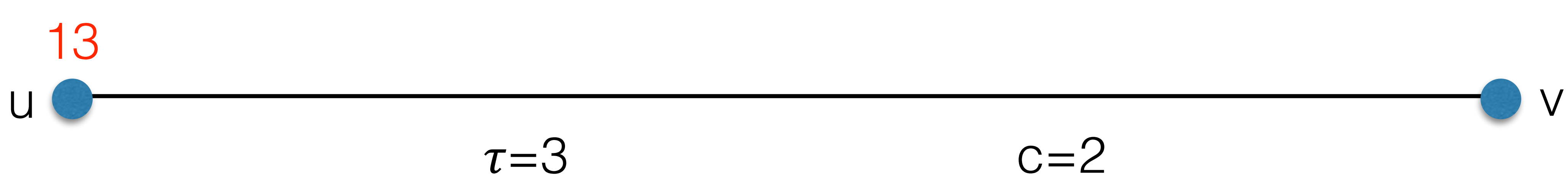
- *G=(V,E)*
- Edges have travel times τ_e and capacities c_e
- Distinguished source *s* and sink *t*



- **Max Flow Over Time Problem** *(input T)*
  How much flow can be pushed from s to t in time T?
  - Ford Fulkerson (1958)
  - Not polynomial (Constructs Static Max-Flow each timestep)
- **Quickest Flow Problem** *(input W)*
  How quickly can W items be moved from s to t?
  - Burkard, Dlasks and Klinz (1993)
  - Strongly Polynomial (uses parametric search)
- **Quickest Transhipment Problem**
  Like QF Problem but Multiple Sources/Sinks
  (with fixed supply/demands)
  - Hoppe & Tardos (2000)
  - Strongly Polynomial (but uses sub modular optimization)

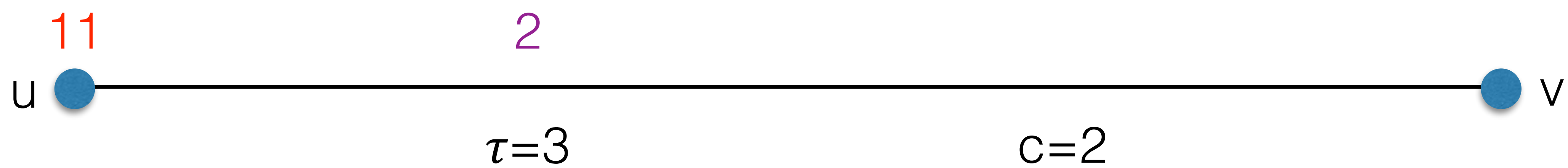
Edges have Capacities

- Original Flow Model is static. Doesn't model time
- Time required is function of both transit times and capacities
- c_e is edge capacity ("width")
  - At most c_e people can enter edge $e=(u,v)$ in one time unit. They travel together as a group on e
  - If more than c_e people at u, remainder need to **wait** to enter e
- τ_e is time for one group to traverse edge
- Start with W people at u
  How much time does take them all to reach v?

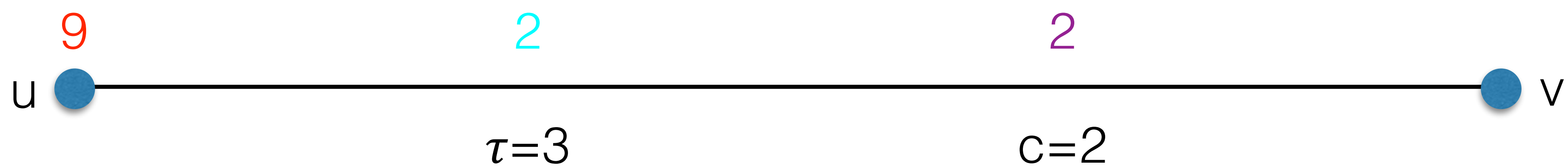



t=0
13
u
v
τ=3
c=2

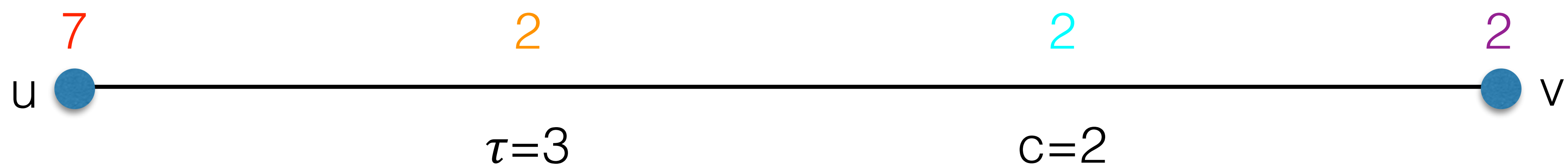
t=1



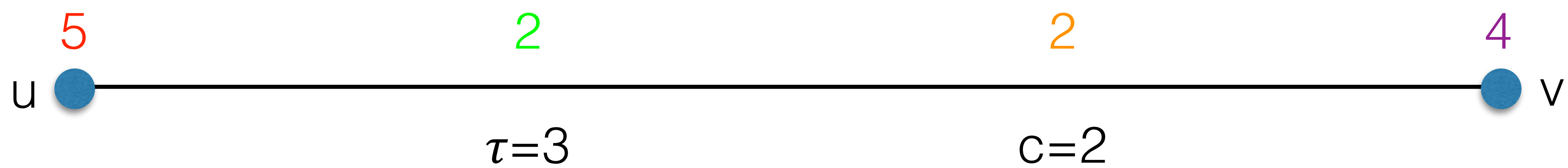
t=2



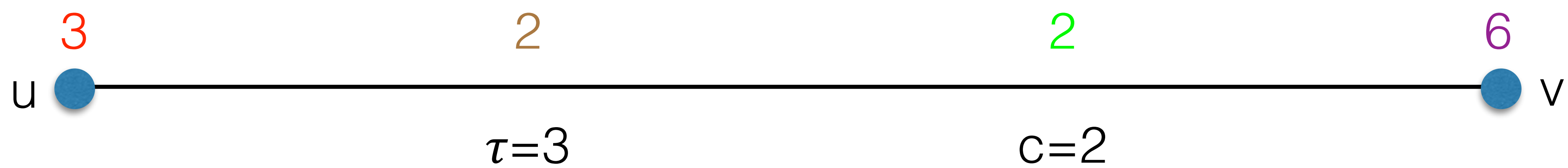
t=3



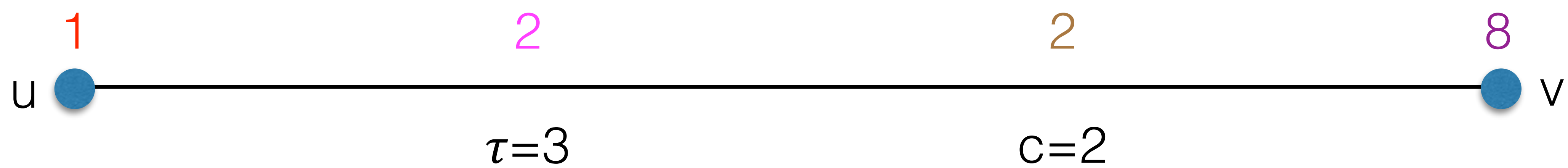
t=4



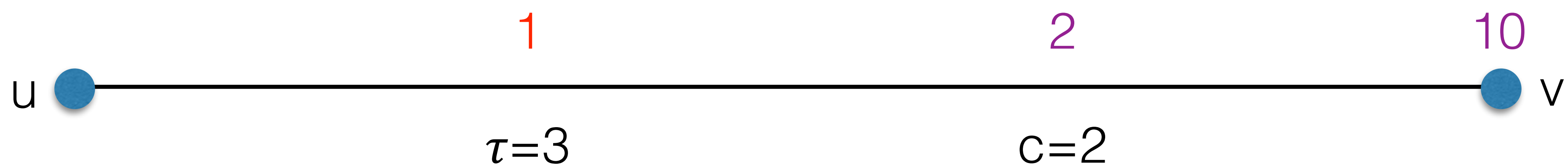
t=5



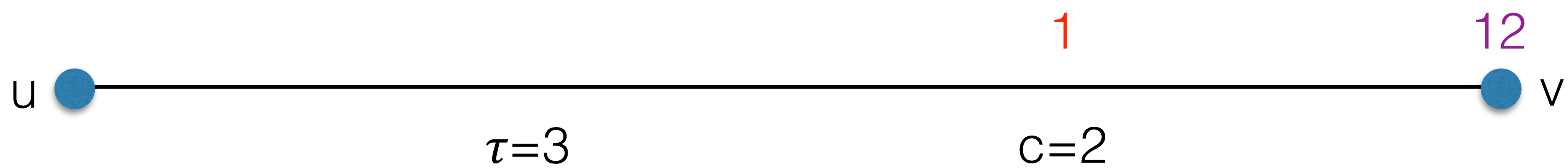
t=6



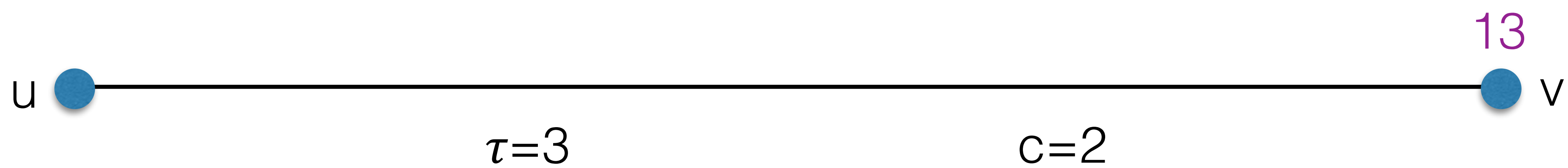
$t=7$



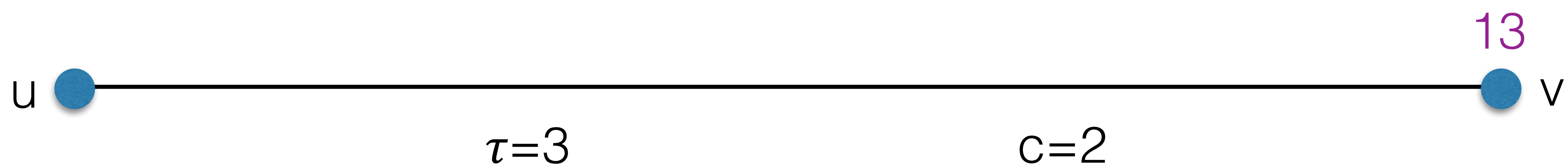
t=8



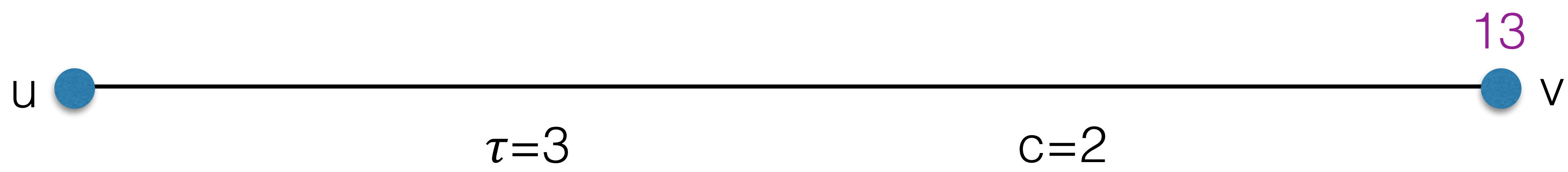
$t=9$



t=9

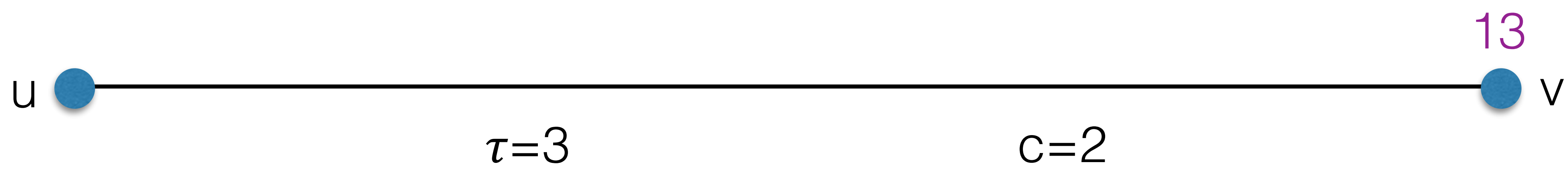


t=9



- 13 items split into $g = \lceil 13/2 \rceil = 7$ groups
- First group reached v at time $t = \tau = 3$
- Last group reached v at time $t = 3 + g - 1 = 9$

t=9

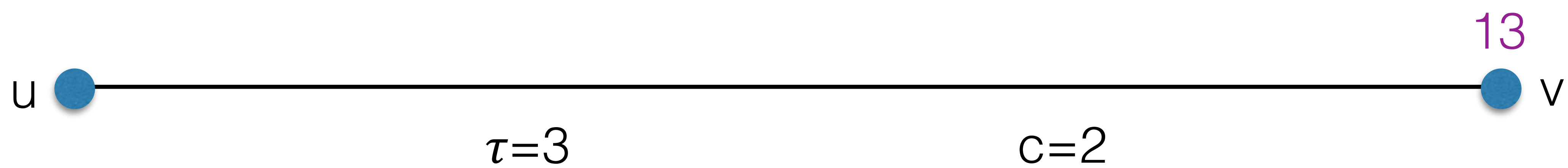


- 13 items split into $g = \lceil 13/2 \rceil = 7$ groups
- First group reached v at time $t = \tau = 3$
- Last group reached v at time $t = 3 + g - 1 = 9$

Discrete Model

- W people, Capacity c integral, Transit time τ
- All edge transit times integral
- Requires $\lceil W/c \rceil + \tau - 1$ time to move everyone from u to v

t=9



- 13 items split into $g = \lceil 13/2 \rceil = 7$ groups
- First group reached v at time $t = \tau = 3$
- Last group reached v at time $t = 3 + g - 1 = 9$

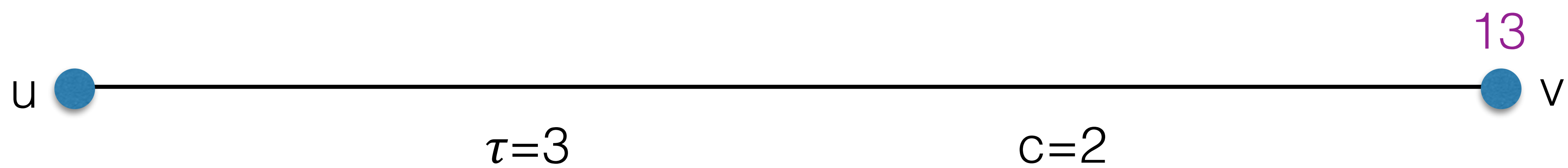
Discrete Model

- W people, Capacity c integral, Transit time τ
- All edge transit times integral
- Requires $\lceil W/c \rceil + \tau - 1$ time to move everyone from u to v

Continuous Model

- W units of non-quantized fluid. Fluid flows continuously
- c is rate: amount that can enter e in one unit of time
- Requires $W/c + \tau - 1$ time to move all fluid from u to v

t=9



- 13 items split into $g = \lceil 13/2 \rceil = 7$ groups
- First group reached v at time $t = \tau = 3$
- Last group reached v at time $t = 3 + g - 1 = 9$

Discrete Model

Default for this talk

- W people, Capacity c integral, Transit time τ
- All edge transit times integral
- Requires $\lceil W/c \rceil + \tau - 1$ time to move everyone from u to v

Continuous Model

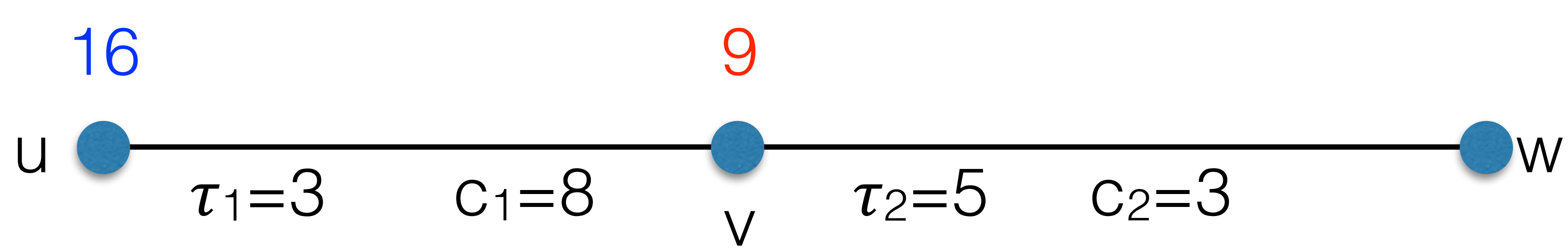
- W units of non-quantized fluid. Fluid flows continuously
- c is rate: amount that can enter e in one unit of time
- Requires $W/c + \tau - 1$ time to move all fluid from u to v

Outline

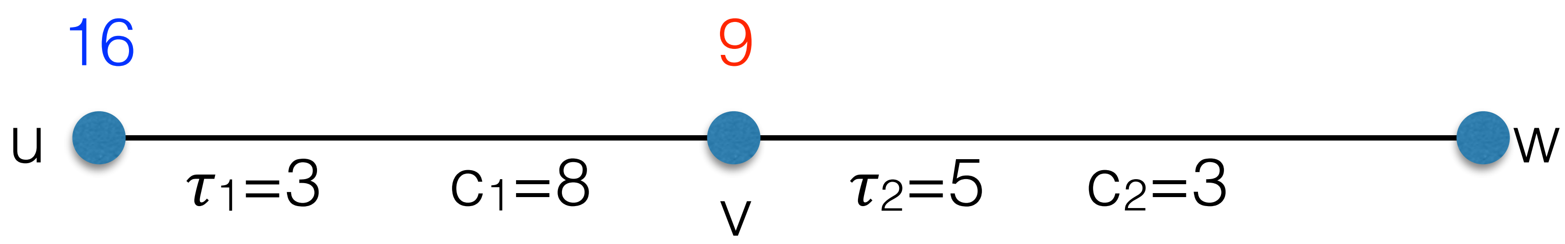
- Dynamic Flow Networks
- Congestion in Dynamic Flows
- Evacuation Flows
  - Problem Definitions
  - Known Results
- Example Algorithm 1: k-Sink Evacuation on a Path
- Example Algorithm 2: 1-sink Min-Max Regret Evacuation on a Path with uniform capacity
- Open Problems

Congestion Effects

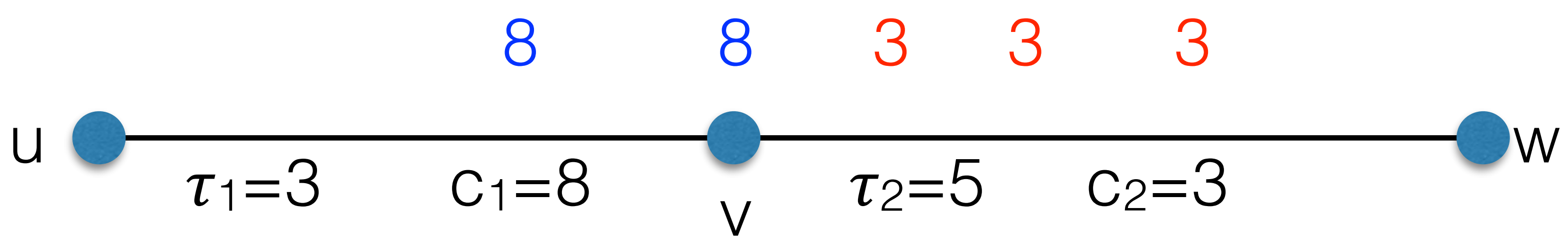
A major complication with dynamic flows is that they introduce congestion effects that can slow down transport time



Congestion Effects

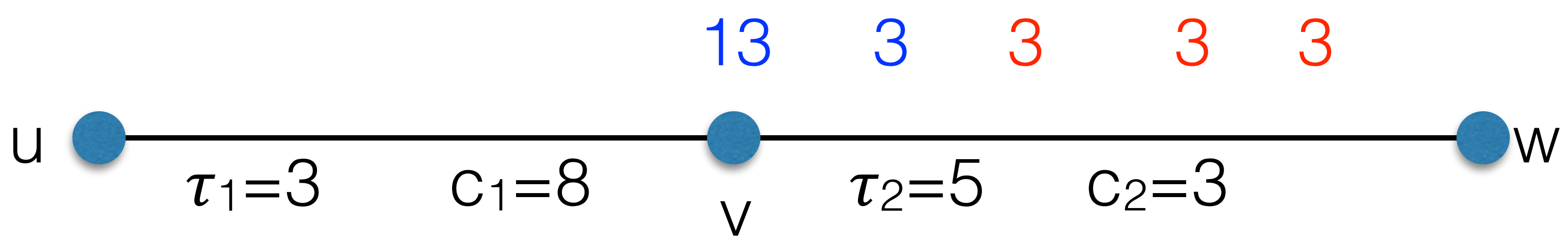


Congestion Effects



t=3: first 8 items from u reach v which is empty.
First 3 items pass through but others need to wait because $c_1 < c_2$
Congestion occurs.

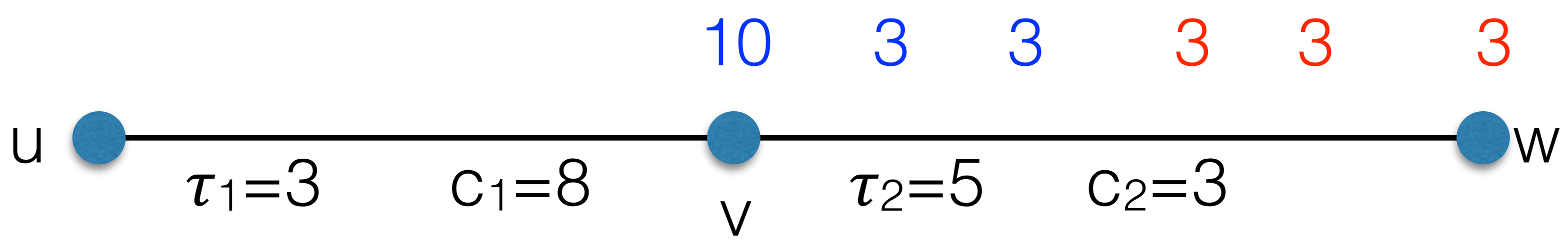
Congestion Effects



t=3: first 8 items from u reach v which is empty.
First 3 items pass through but others need to wait because $c_1<c_2$
Congestion occurs.

t=4

Congestion Effects

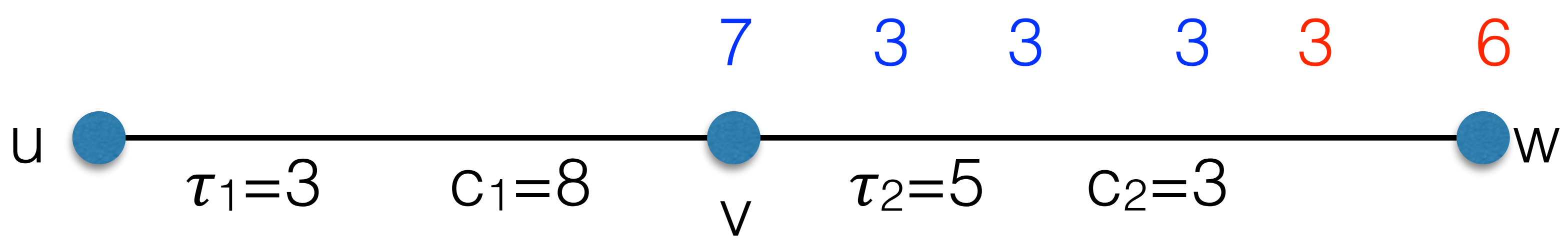


t=3: first 8 items from u reach v which is empty.
First 3 items pass through but others need to wait because $c_1 < c_2$
Congestion occurs.

t=4

t=5

Congestion Effects



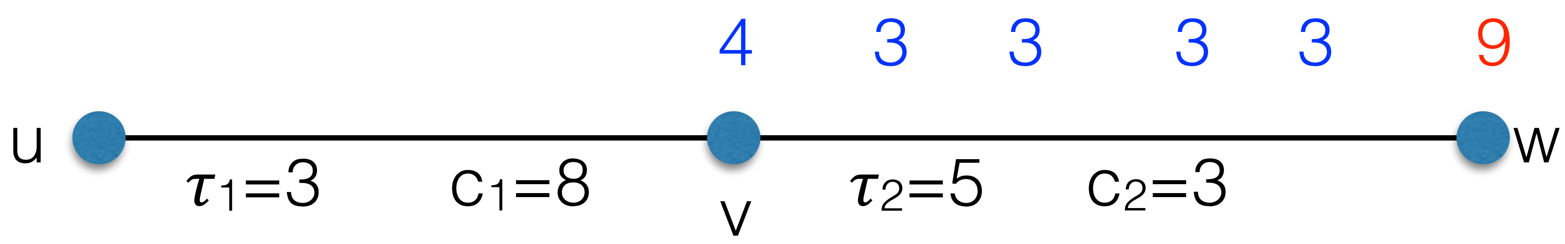
t=3: first 8 items from u reach v which is empty.
First 3 items pass through but others need to wait because $c_1<c_2$
Congestion occurs.

t=4

t=5

t=6

Congestion Effects



t=3: first 8 items from u reach v which is empty.
First 3 items pass through but others need to wait because $c_1 < c_2$
Congestion occurs.

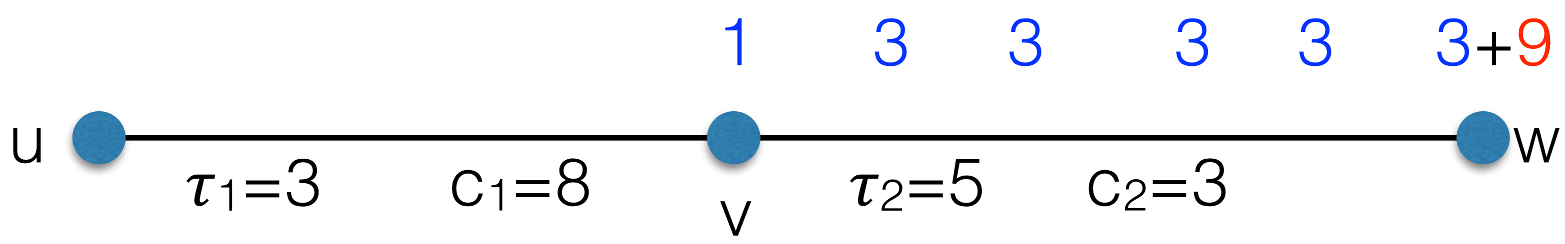
t=4

t=5

t=6

t=7

Congestion Effects



t=3: first 8 items from u reach v which is empty.
First 3 items pass through but others need to wait because $c_1 < c_2$
Congestion occurs.

t=4

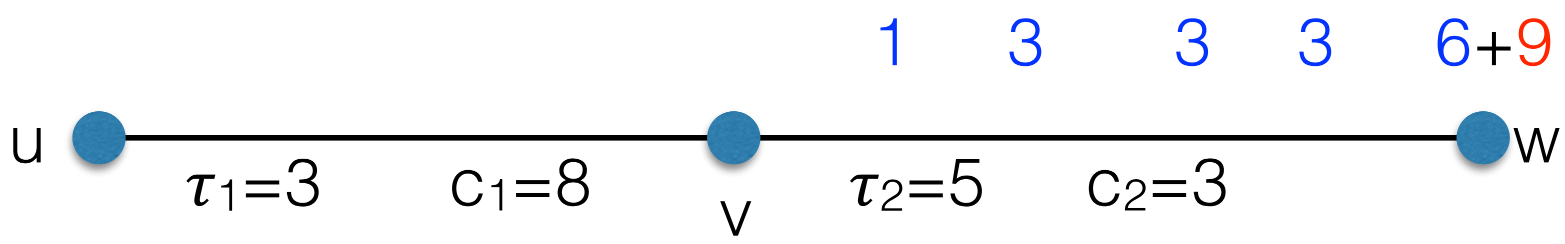
t=5

t=6

t=7

t=8

Congestion Effects



t=3: first 8 items from u reach v which is empty.
First 3 items pass through but others need to wait because $c_1 < c_2$
Congestion occurs.

t=4

t=5

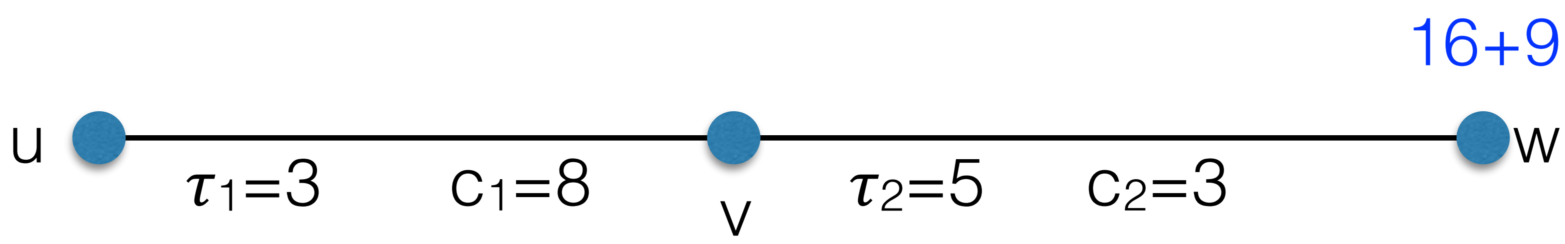
t=6

t=7

t=8

t=9

Congestion Effects



t=3: first 8 items from u reach v which is empty.
First 3 items pass through but others need to wait because $c_1<c_2$
Congestion occurs.

t=4

t=5

t=6

t=7

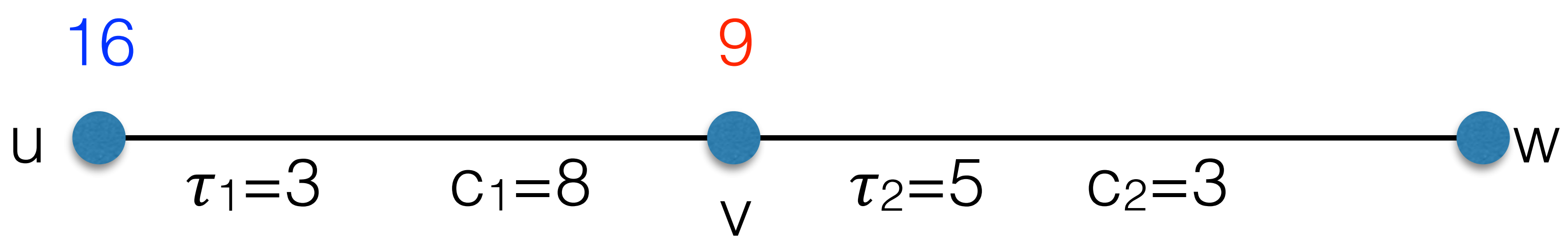
t=8

t=9

…

t=13: Last item arrives at w

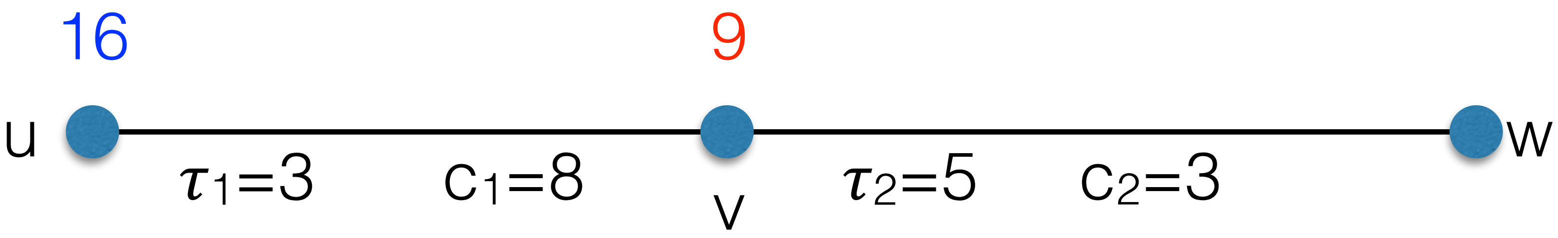
Congestion Effects



Congestion occurs because $c_1<c_2$

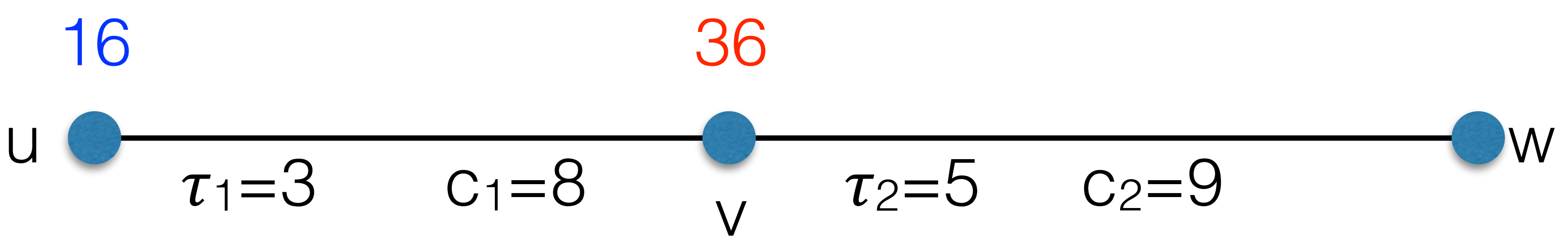
Full Evacuation at $t=13$

Congestion Effects

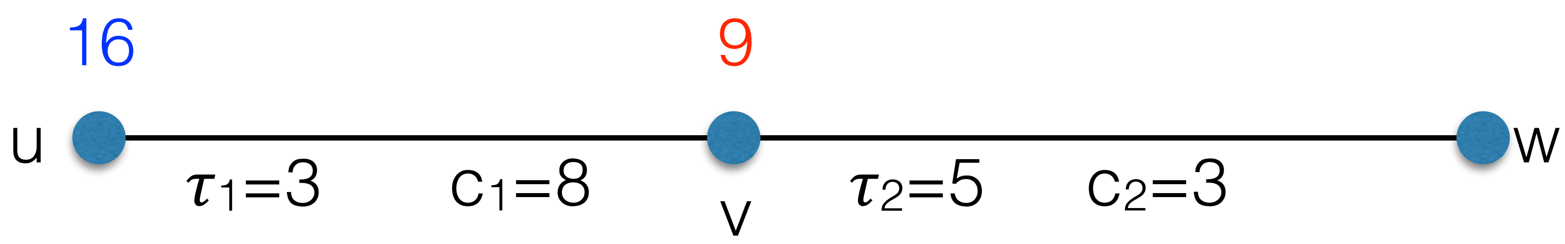


Congestion occurs because $c_1 < c_2$

Full Evacuation at t=13

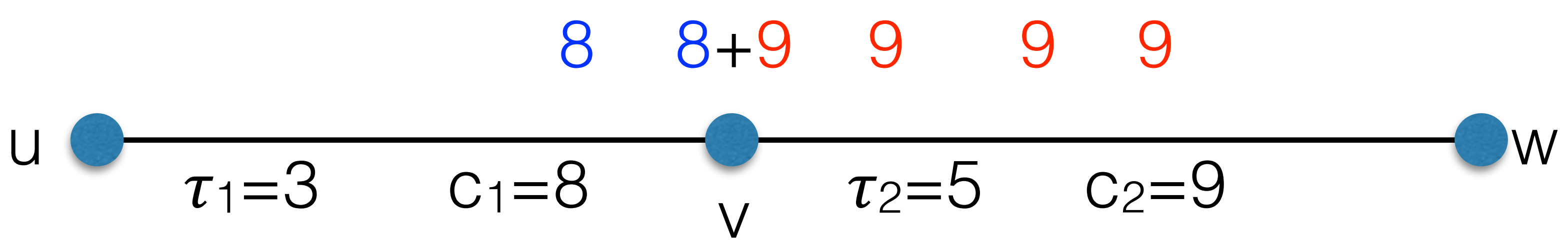


Congestion Effects



Congestion occurs because $c_1 < c_2$

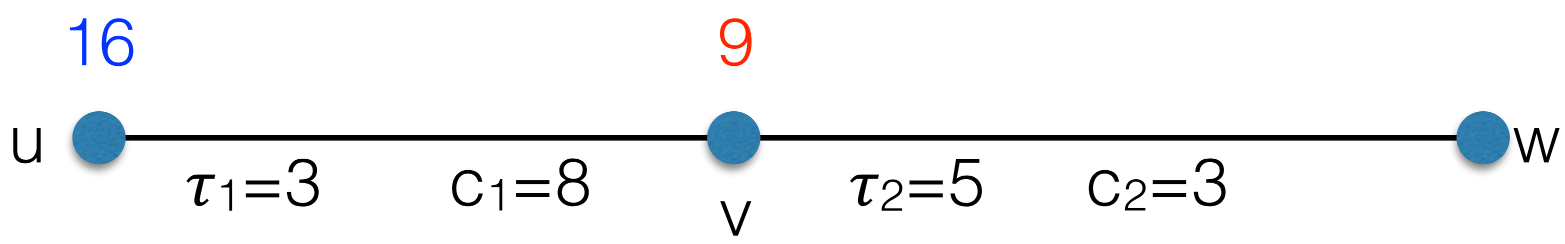
Full Evacuation at $t=13$



$t=3$: first 8 items from u reach v which still contains 9 items

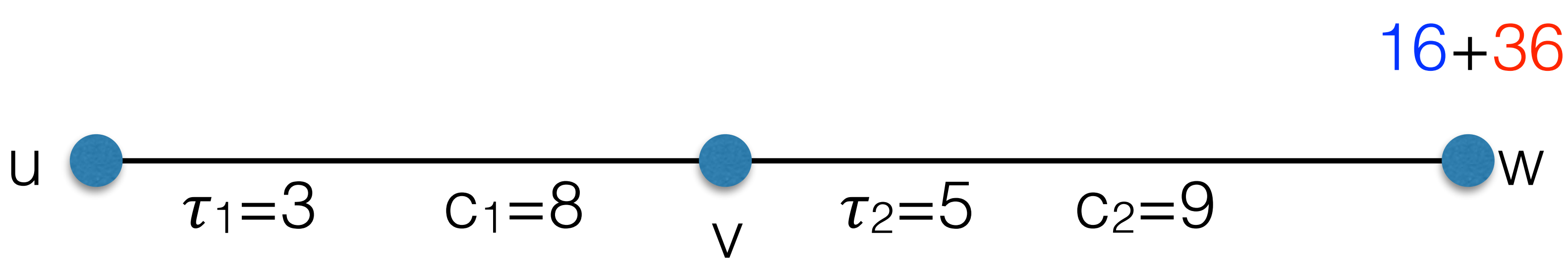
=> **Congestion occurs.**

Congestion Effects



Congestion occurs because $c_1<c_2$

Full Evacuation at $t=13$

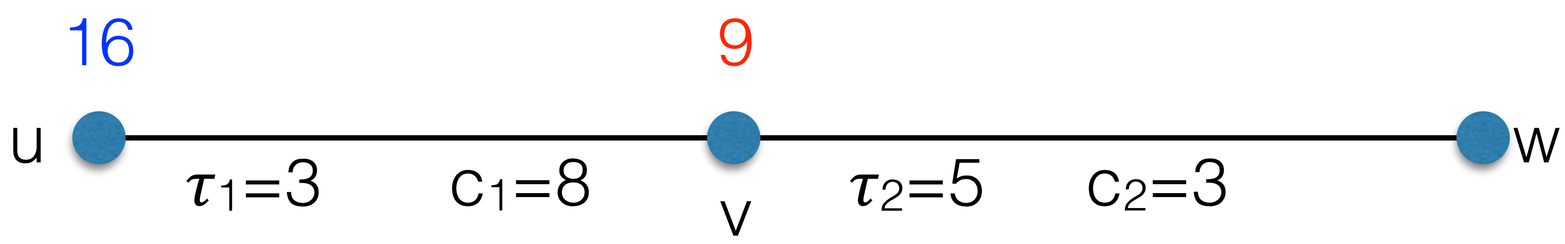


$t=3$: first 8 items from u reach v which still contains 9 items

=> **Congestion occurs.**

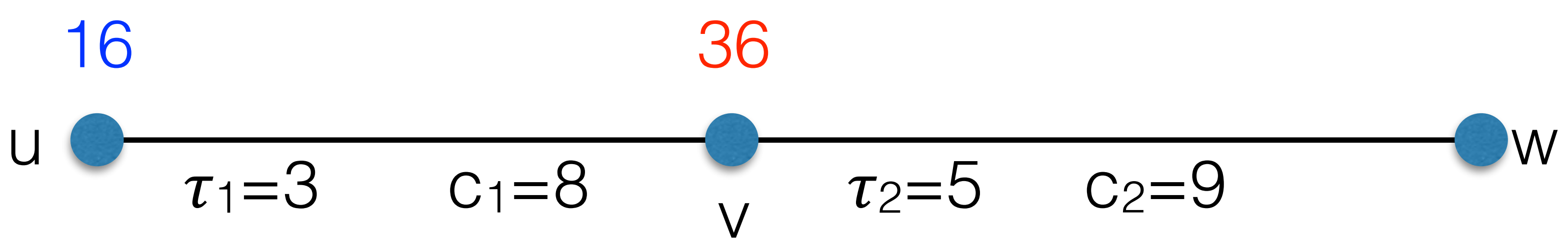
$t=12$: Last item arrives at w

Congestion Effects



Congestion occurs because $c_1<c_2$

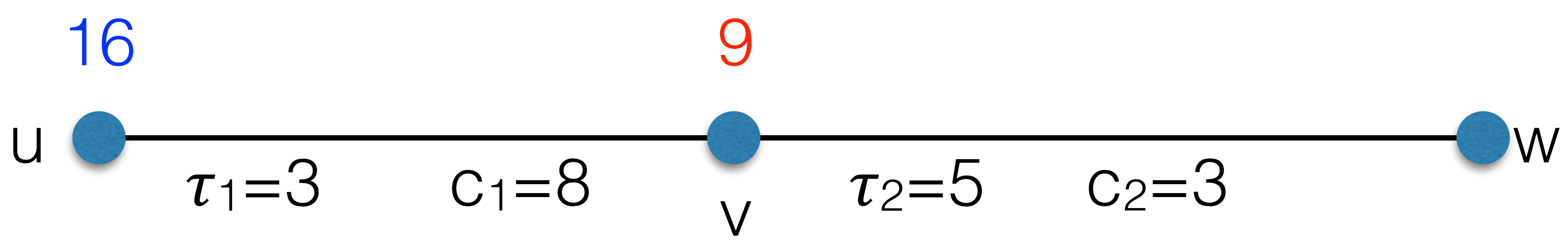
Full Evacuation at t=13



Congestion occurs because v not empty when first group arrives from u

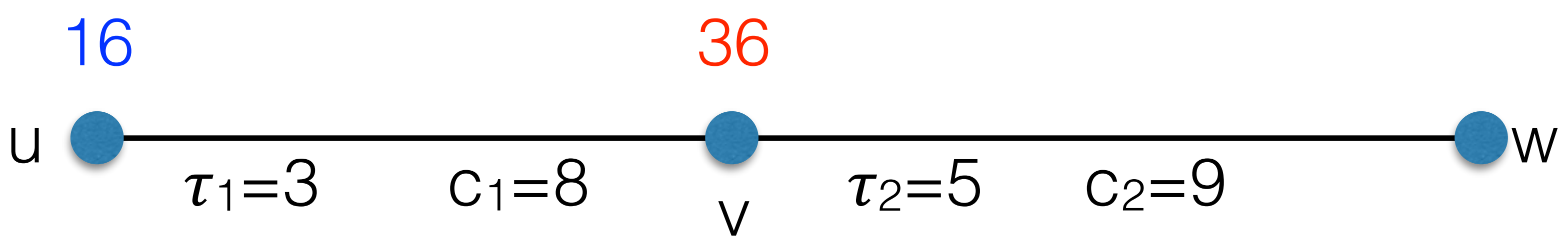
Full Evacuation at t=12

Congestion Effects



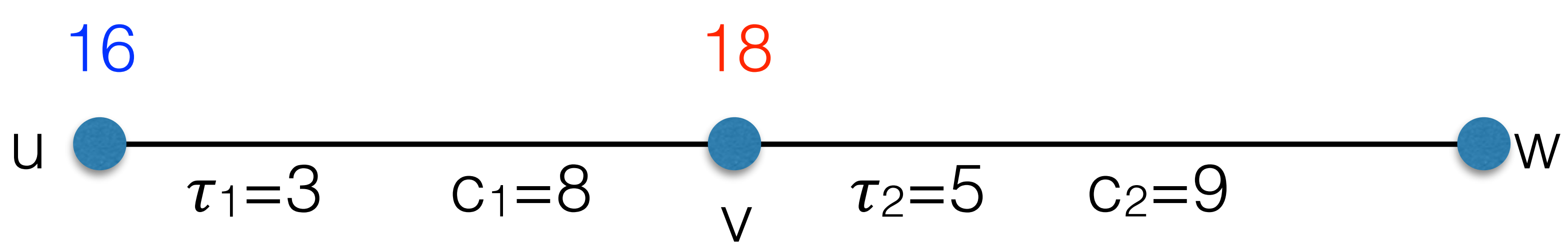
Congestion occurs because $c_1 < c_2$

Full Evacuation at t=13

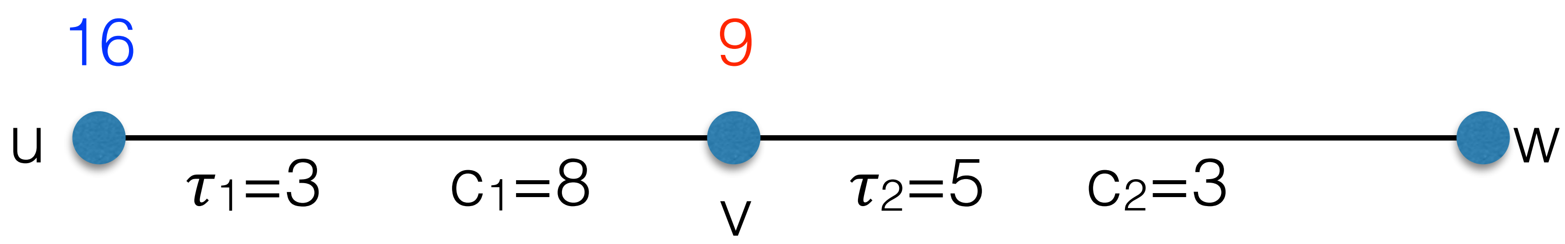


Congestion occurs because v not empty when first group arrives from u

Full Evacuation at t=12

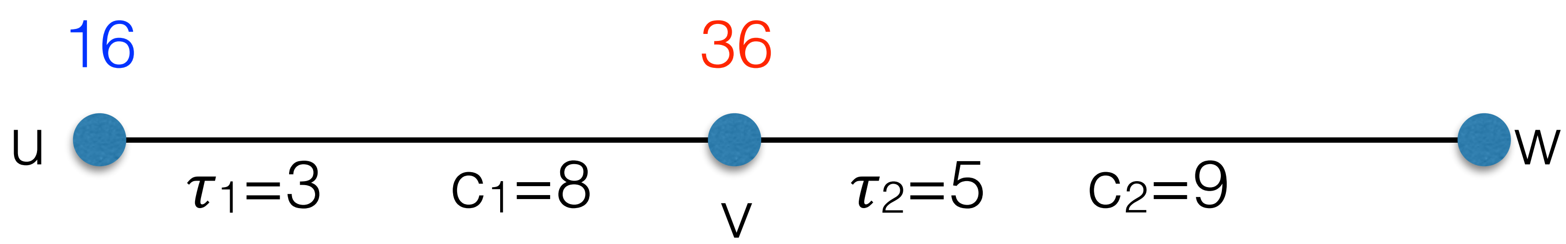


Congestion Effects



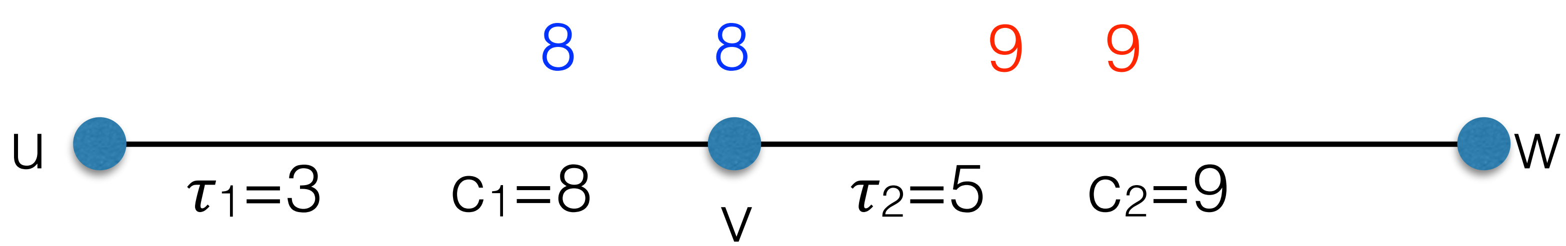
Congestion occurs because $c_1<c_2$

Full Evacuation at $t=13$



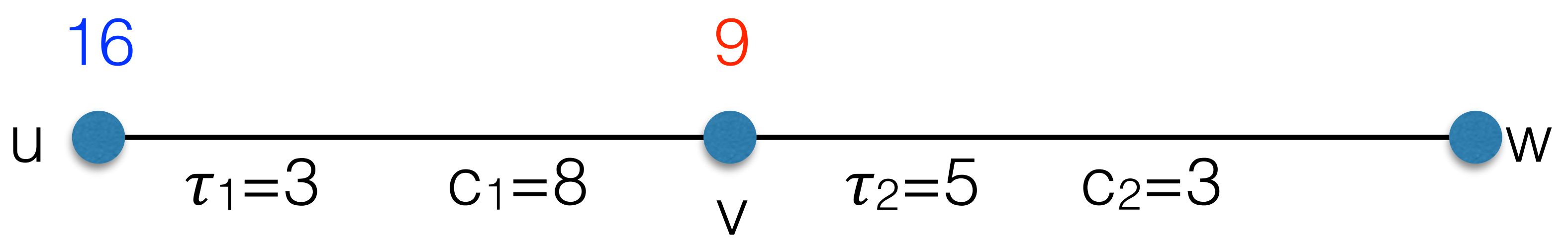
Congestion occurs because v not empty when first group arrives from u

Full Evacuation at $t=12$



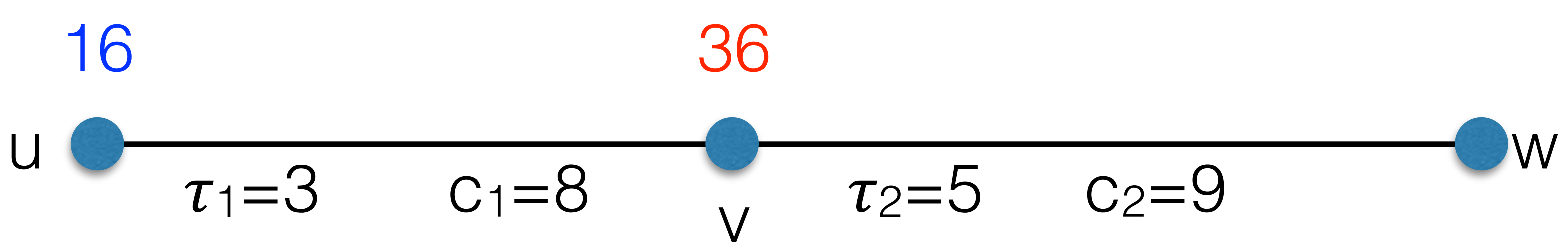
$t=3$: first 8 items from u reach v which is empty still contains 9 items => **NO Congestion occurs.**

Congestion Effects



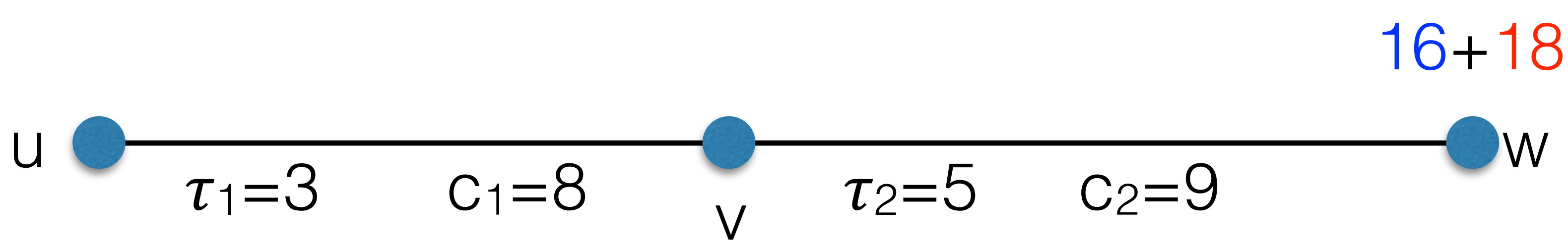
Congestion occurs because $c_1 < c_2$

Full Evacuation at $t=13$



Congestion occurs because v not empty when first group arrives from u

Full Evacuation at $t=12$

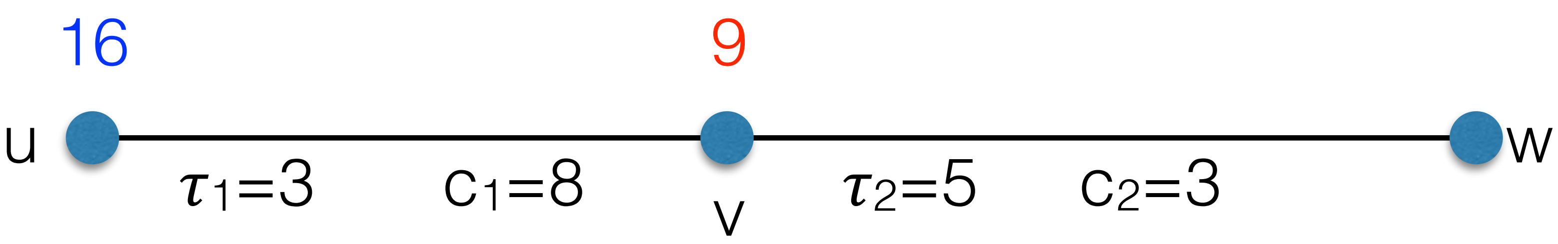


$t=3$: first 8 items from u reach v which is empty still contains 9 items

=> **NO Congestion occurs.**

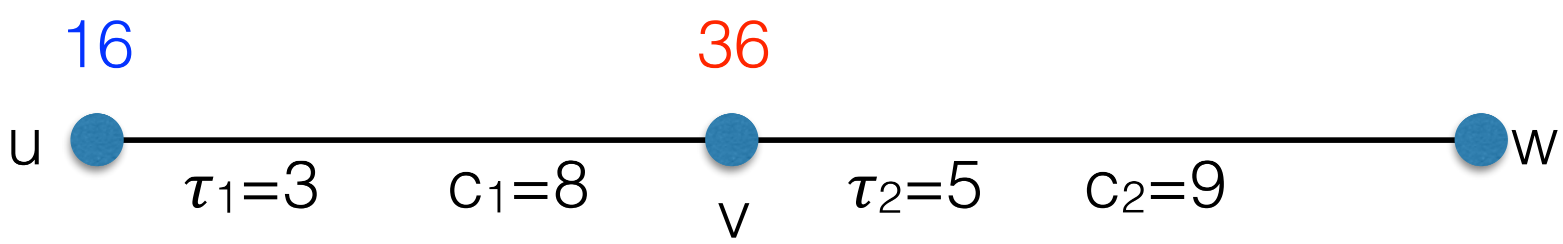
$t=9$: Last item arrives at w

Congestion Effects



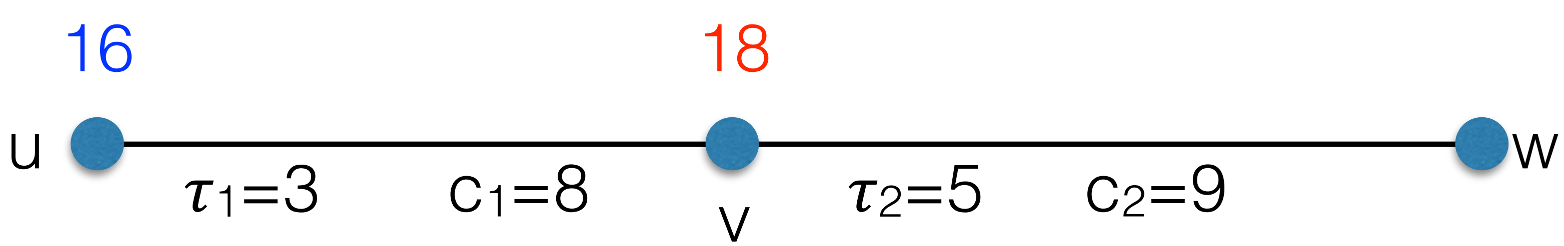
Congestion occurs because $c_1<c_2$

Full Evacuation at t=13



Congestion occurs because v not empty when first group arrives from u

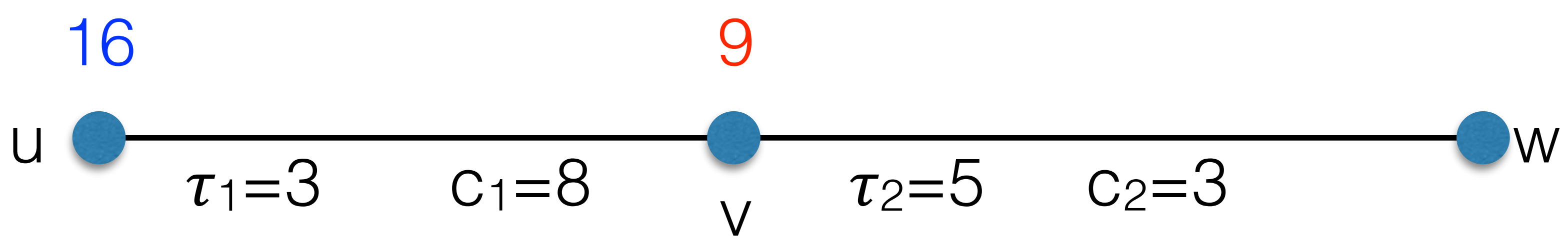
Full Evacuation at t=12



Items at u pass through v with **No Congestion occuring**

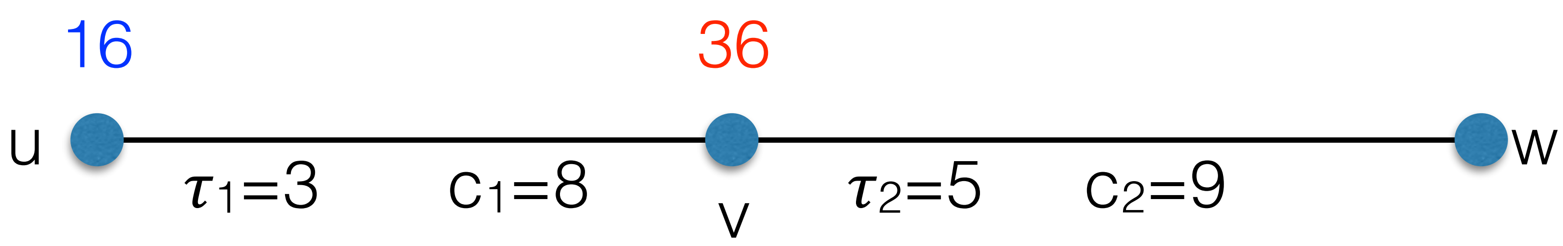
Full Evacuation at t=9

Congestion Effects



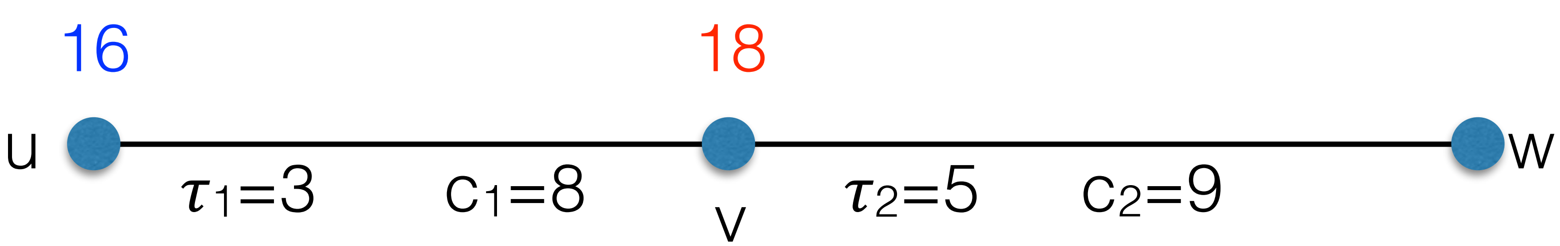
Congestion occurs because $c_1 < c_2$

Full Evacuation at $t=13$



Congestion occurs because v not empty when first group arrives from u

Full Evacuation at $t=12$

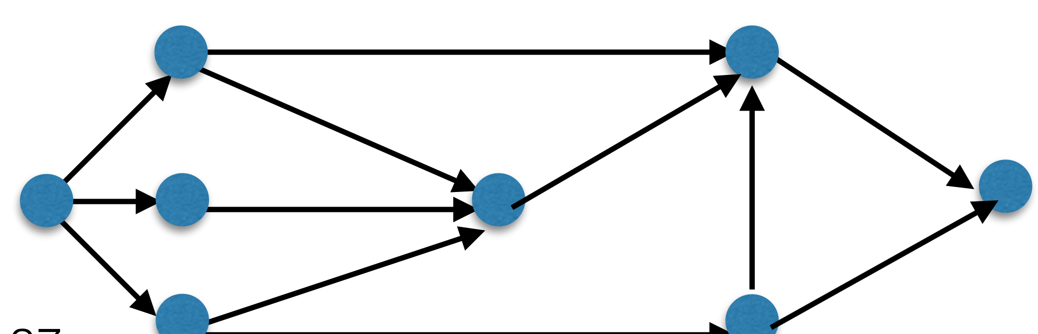


Items at u pass through v with **No Congestion occuring**

Full Evacuation at $t=9$

Analysis of Flow/Evacuation times must include congestion!!

Can be very complicated!



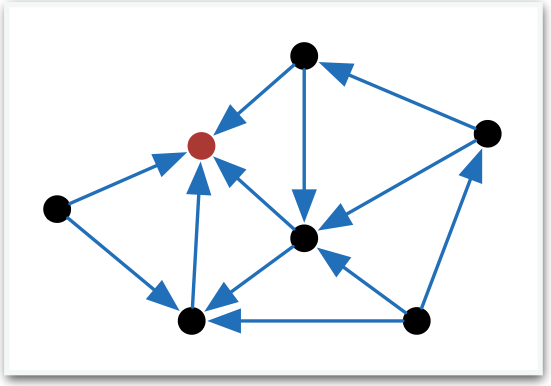
Outline

- Dynamic Flow Networks
- Congestion in Dynamic Flows
- Evacuation Flows
  - Problem Definitions
  - Known Results
- Example Algorithm 1: k-Sink Evacuation on a Path
- Example Algorithm 2: 1-sink Min-Max Regret Evacuation on a Path with uniform capacity
- Open Problems

Evacuation is NOT Flow

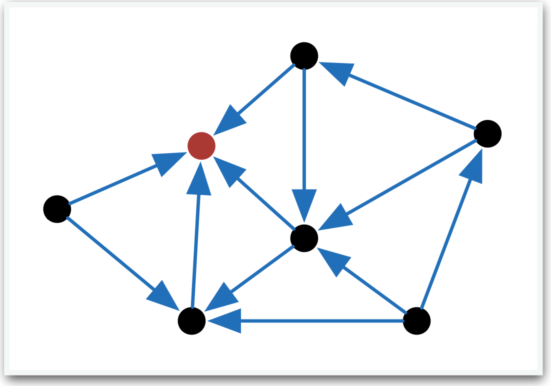
Evacuation is NOT Flow

- In Flow, different people from same vertex can follow different paths



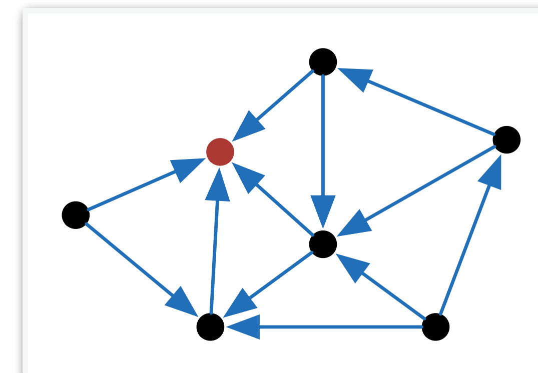
Evacuation is NOT Flow

- In Flow, different people from same vertex can follow different paths
- In Evacuation, want signs at vertices pointing **this way out =>**
  - Each vertex has unique **evacuation edge**.
  - Every person reaching that vertex must follow the evacuation edge to next vertex

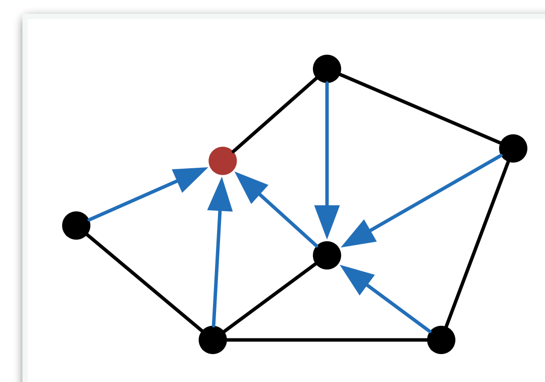


Evacuation is NOT Flow

- In Flow, different people from same vertex can follow different paths

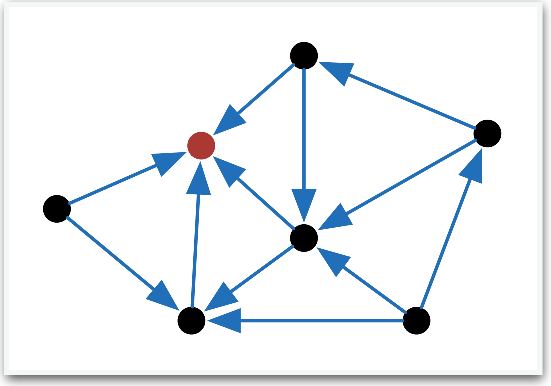


- In Evacuation, want signs at vertices pointing **this way out =>**
  - Each vertex has unique **evacuation edge**.
  - Every person reaching that vertex must follow the evacuation edge to next vertex

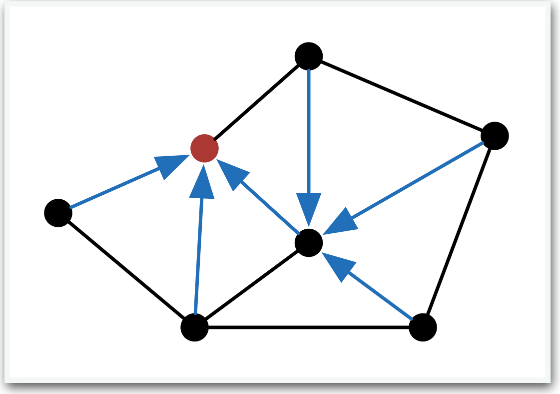


Evacuation is NOT Flow

- In Flow, different people from same vertex can follow different paths

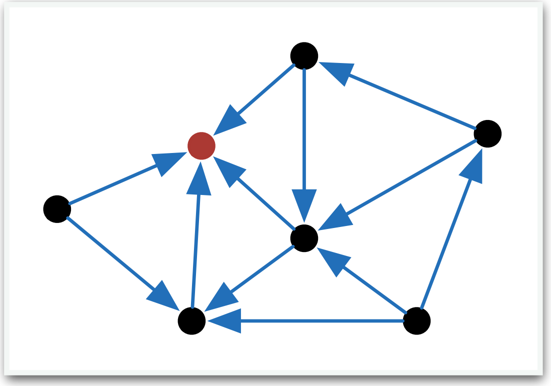


- In Evacuation, want signs at vertices pointing **this way out =>**
  - Each vertex has unique **evacuation edge**.
  - Every person reaching that vertex must follow the evacuation edge to next vertex
  - Evacuation edges partition vertices into directed forests moving toward sinks

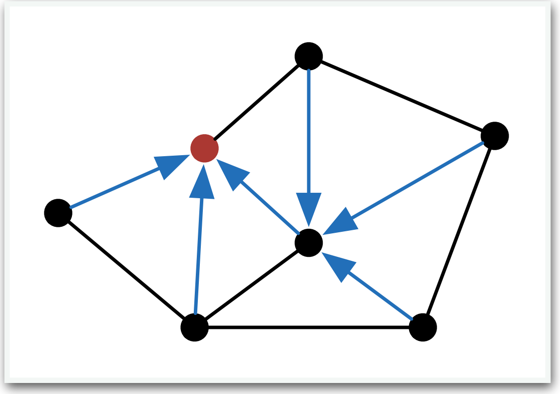


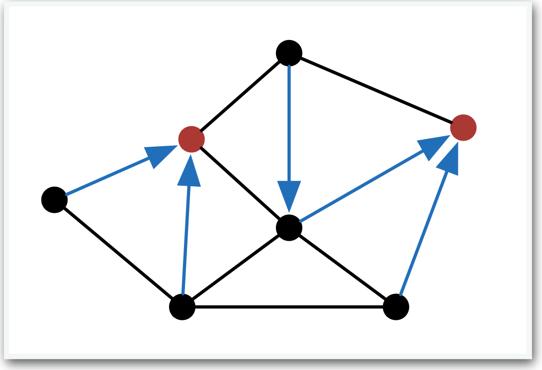
Evacuation is NOT Flow

- In Flow, different people from same vertex can follow different paths



- In Evacuation, want signs at vertices pointing **this way out =>**
  - Each vertex has unique **evacuation edge**.
  - Every person reaching that vertex must follow the evacuation edge to next vertex
  - Evacuation edges partition vertices into directed forests moving toward sinks





Graph Evacuation Problems

Graph Evacuation Problems

- **Input:** Graph $G=(V,E)$
  - τ_e, c_e: transit times *and capacities for each edge*
  - w_v: # of people starting on vertex v
  - Sinks: Either fixed set $K \subseteq V$ of sinks or a number k of sinks allowed

Graph Evacuation Problems

- **Input:** Graph G=(V,E)
  - τ_e, c_e: transit times *and capacities for each edge*
  - w_v: # of people starting on vertex *v*
  - Sinks: Either fixed set $K \subseteq V$ of sinks or a number k of sinks allowed
- **Output:** An Evacuation Protocol that minimizes maximum evacuation time
  - Evacuation Protocol
    - A unique evacuation edge for each vertex
    - If input is k, a set $K \subseteq V$ *of sinks with* $|K|=k$
  - Maximum Evacuation time
    - The evacuation time of a vertex is the earliest time by which ALL items from that vertex have reached a sink.
    - Maximum evacuation time is the maximum evacuation time over all vertices

Graph Evacuation Problems: Variations

Graph Evacuation Problems: Variations

- **Type of graph G: Path, Tree, General, ….**
  - For general G and k>1 problem is NP-Complete because it solves k-Center (if c_e set to be large)

Graph Evacuation Problems: Variations

- **Type of graph G: Path, Tree, General, ….**
  - For general G and k>1 problem is NP-Complete because it solves k-Center (if c_e set to be large)
- **Sink Input: Actual Sinks vs # of sinks**

Graph Evacuation Problems: Variations

- **Type of graph G: Path, Tree, General, ….**
  - For general G and k>1 problem is NP-Complete because it solves k-Center (if c_e set to be large)
- **Sink Input: Actual Sinks vs # of sinks**
- **Discrete vs Continuous flow**
  - Fleischer, Tardos (1998). D and C Dynamic Flow problems can often be solved using same algorithm

Graph Evacuation Problems: Variations

- **Type of graph G: Path, Tree, General, ….**
  - For general G and k>1 problem is NP-Complete because it solves k-Center (if c_e set to be large)
- **Sink Input: Actual Sinks vs # of sinks**
- **Discrete vs Continuous flow**
  - Fleischer, Tardos (1998). D and C Dynamic Flow problems can often be solved using same algorithm
- **Sink locations: anywhere or only on vertices**

Graph Evacuation Problems: Variations

- **Type of graph G: Path, Tree, General, ….**
  - For general G and k>1 problem is NP-Complete because it solves k-Center (if c_e set to be large)
- **Sink Input: Actual Sinks vs # of sinks**
- **Discrete vs Continuous flow**
  - Fleischer, Tardos (1998). D and C Dynamic Flow problems can often be solved using same algorithm
- **Sink locations: anywhere or only on vertices**
- **c_e: uniform (all the same) vs general (arbitrary)**

Graph Evacuation Problems: Variations

- **Type of graph G: Path, Tree, General, ….**
  - For general G and k>1 problem is NP-Complete because it solves k-Center (if c_e set to be large)
- **Sink Input: Actual Sinks vs # of sinks**
- **Discrete vs Continuous flow**
  - Fleischer, Tardos (1998). D and C Dynamic Flow problems can often be solved using same algorithm
- **Sink locations: anywhere or only on vertices**
- **c_e: uniform (all the same) vs general (arbitrary)**
- **Min-Max vs Min-Max Regret**
  - Robust solutions. MMR allows w_v, # of people on vertex, to be a range rather than a number. Find "best" solution for all allowable scenarios

Outline

- Dynamic Flow Networks
- Congestion in Dynamic Flows
- Evacuation Flows
  - Problem Definitions
  - Known Results
- Example Algorithm 1: k-Sink Evacuation on a Path
- Example Algorithm 2: 1-sink Min-Max Regret Evacuation on a Path with uniform capacity
- Open Problems

Known Results

	Min-max cost (DISCRETE/CONTINUOUS)			
	General capacity		Uniform capacity	
	1-sink	k-sink	1-sink	k-sink
Path	$O(n)$ [2]	$O(kn \log^2 n)$ [2]	$O(n)$	$O(kn)$ [6]
Tree	$O(n \log^2 n)$ [7]	$O(k^2 n \log^4 n)$ [3]	$O(n \log n)$ [4]	$O(k^2 n \log^3 n)$ [3]
General graph	Poly?	NP-Hard	Poly?	NP-Hard

	Min-max regret cost (DISCRETE/CONTINUOUS)			
	General capacity		Uniform capacity	
	1-sink	k-sink	1-sink	k-sink
Path	None		$O(n \log n)$ [5,9]	$O(kn^3 \log n)$ [1]
Tree			$O(n^2 \log^2 n)$ [4]	None
General graph			None	

References

[1] G.P. Arumugam, J. Augustine, M.J. Golin and P. Srikanthan, "A Polynomial Time Algorithm for Minimax-Regret Evacuation on a Dynamic Path", arXiv:1404.5448, 2014

[2] G.P. Arumugam, J. Augustine, M.J. Golin and P. Srikanthan, "Evacuation on Dynamic Paths with General Edge Capacities", document in preparation (2015)

[3] Di Chen and M.J. Golin, "Optimal Sink Location Problems in Dynamic Tree Networks", document in preparation (2015)

[4] Y. Higashikawa, M. J. Golin and N. Katoh, "Minimax Regret Sink Location Problem in Dynamic Tree Networks with Uniform Capacity", *Proc. WALCOM 2014*, LNCS 8344, pp. 125-137, 2014.

[5] Y. Higashikawa, J. Augustine, S. W. Cheng, N. Katoh, G. Ni, B. Su and Y. Xu, "Minimax Regret 1-Sink Location Problem in Dynamic Path Networks", *Theoretical Computer Science*, 2014.

[6] Y. Higashikawa, M. J. Golin and N. Katoh, "Multiple Sink Location Problems in Dynamic Path Networks", *Theoretical Computer Science* (to appear) 2015.

[7] S. Mamada, T. Uno, K. Makino and S. Fujishige, "An O(n $\log^2$ n) Algorithm for the Optimal Sink Location Problem in Dynamic Tree Networks", *Discrete Applied Mathematics*, 154(16), pp. 2387-2401, 2006.

[8] G. Ni, Y. Xu and Y. Dong, "Minimax regret k-sink location problem in dynamic path networks", *Proc. AAIM 2014*

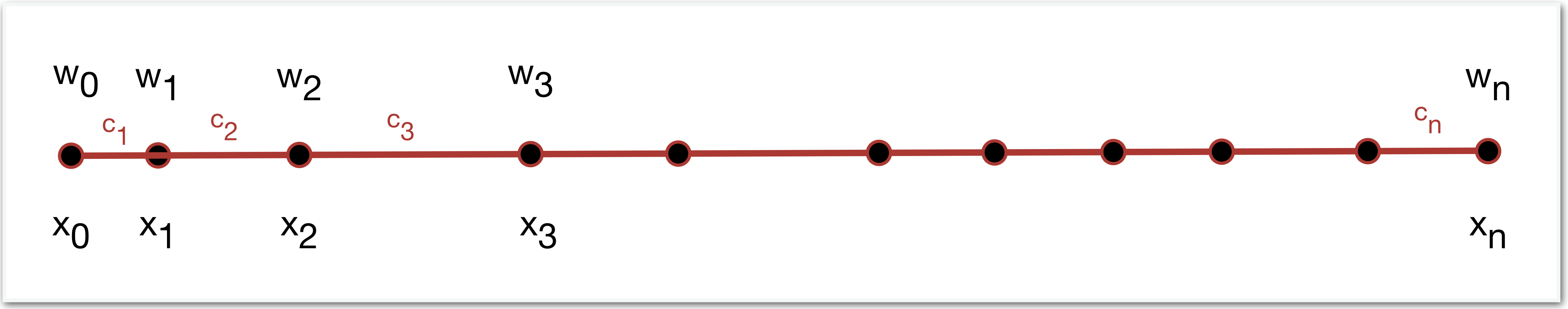
[9] H. Wang, "Minmax Regret 1-Facility Location on Uncertain Path Networks", *Proc. ISAAC 2013*, LNCS 8283, pp. 733-743, 2013.

Outline

- Dynamic Flow Networks
- Congestion in Dynamic Flows
- Evacuation Flows
  - Problem Definitions
  - Known Results
- Example Algorithm 1: k-Sink Evacuation on a Path
- Example Algorithm 2: 1-sink Min-Max Regret Evacuation on a Path with uniform capacity
- Open Problems

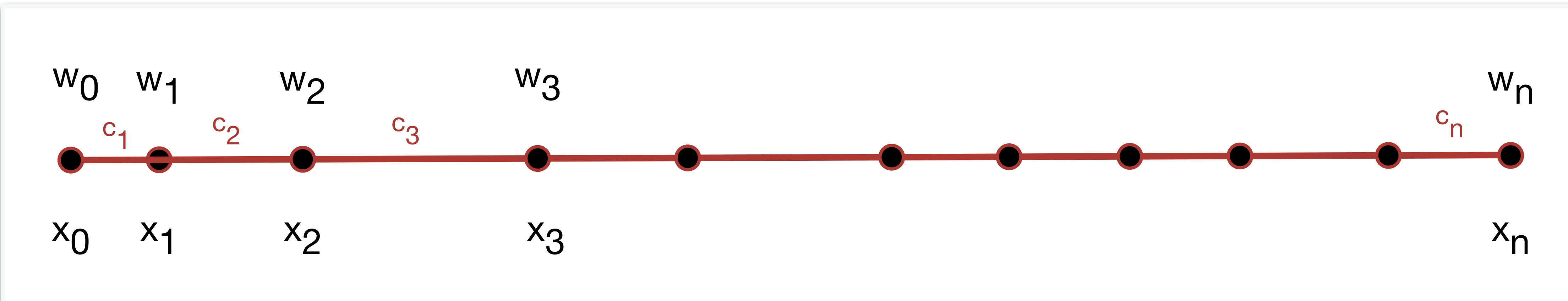
K-Sink Evacuation on a Path

Given a path, associated values c_e, τ_{e,W_v} and k, # of sinks,



K-Sink Evacuation on a Path

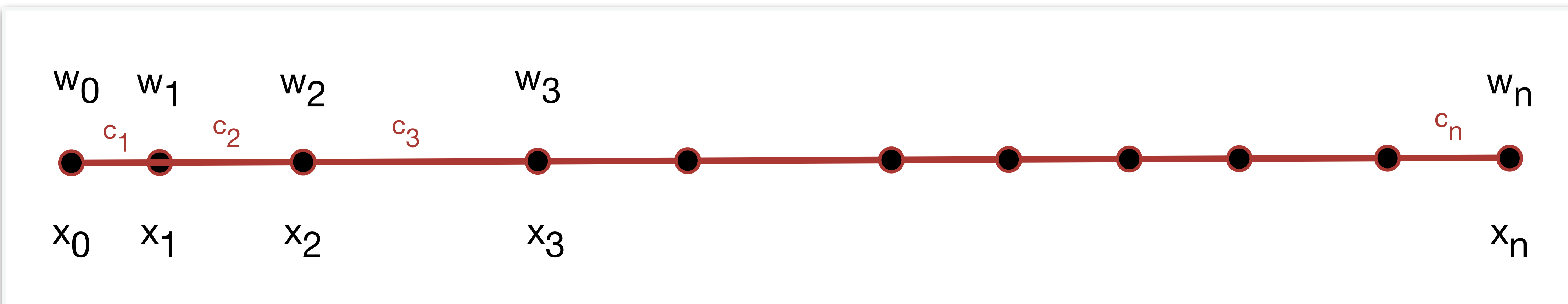
Given a path, associated values c_e, τ_{e,W_v} and k, # of sinks,



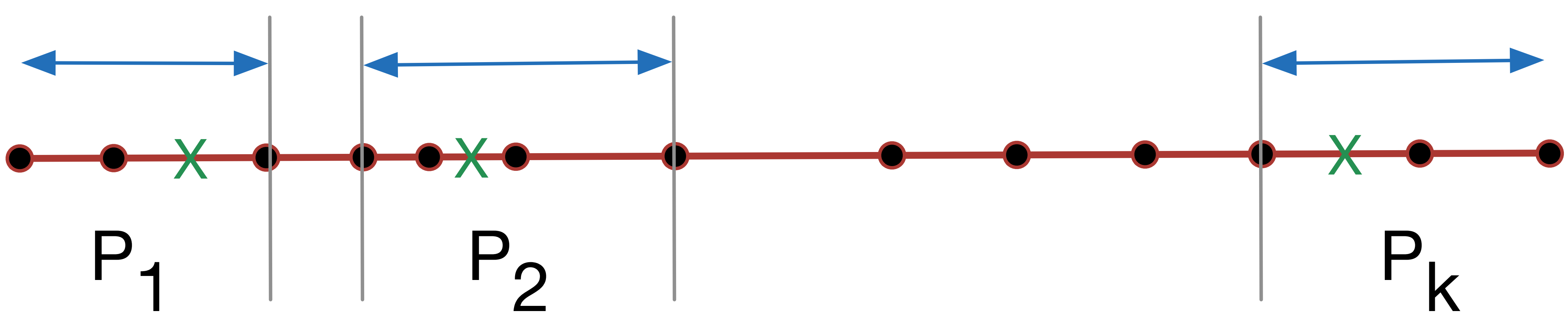
Find a partition into k-subpaths and a sink for each subpath, that minimizes the maximum evacuation time over all subpaths.

K-Sink Evacuation on a Path

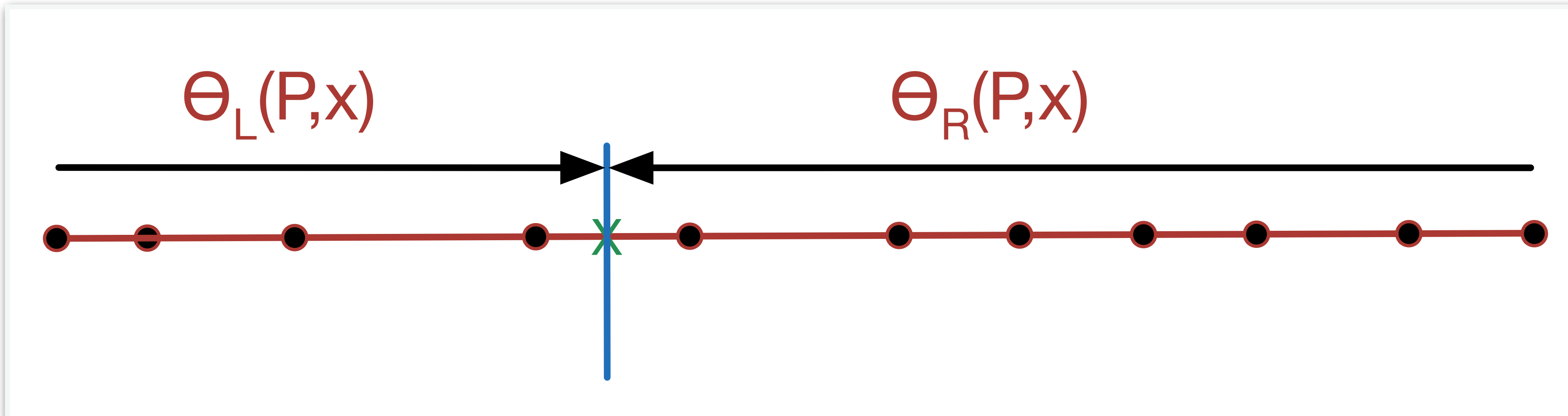
Given a path, associated values c_e, τ_{e,W_v} and k, # of sinks,



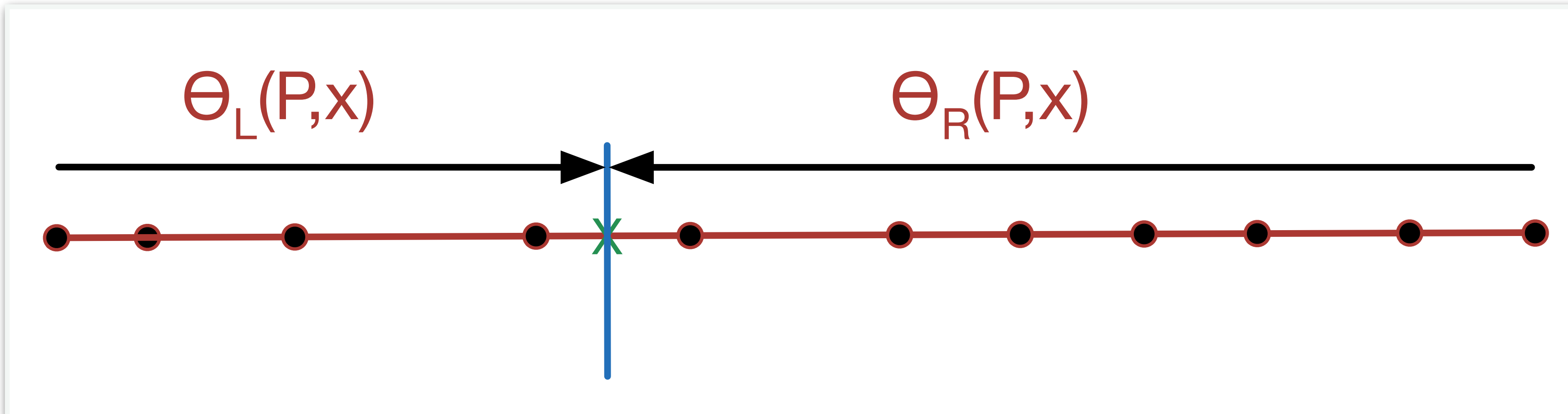
Find a partition into k-subpaths and a sink for each subpath, that minimizes the maximum evacuation time over all subpaths.



1-Sink Evacuation Notation



1-Sink Evacuation Notation



$\Theta_L(P,x)$ = Time to evacuate all nodes to left of x on P to x

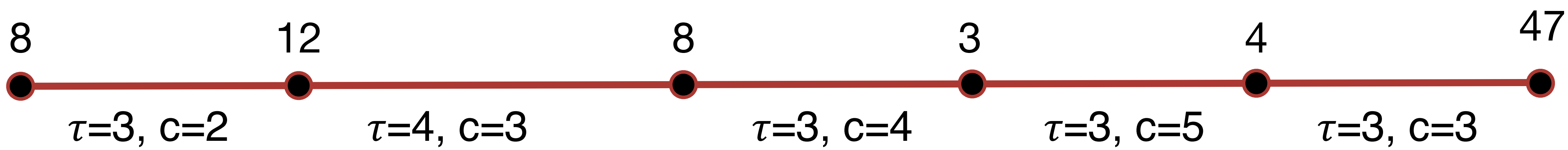
$\Theta_R(P,x)$ = Time to evacuate all nodes to right of x on P to x

$\Theta(P,x)$ = $\max(\Theta_L(P,x), \Theta_R(P,x))$
= Time to evacuate all nodes on P to x

$\Theta^1(P)$ = $\min_{\{x \in P\}} \Theta(P,x)$
= min evacuation time for P with one sink

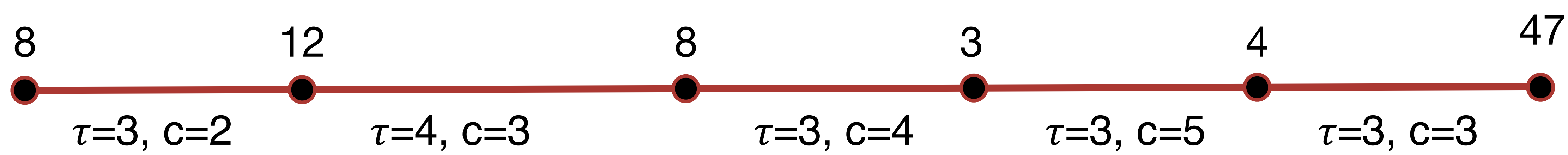
1-Sink Evacuation Example

Original Input:

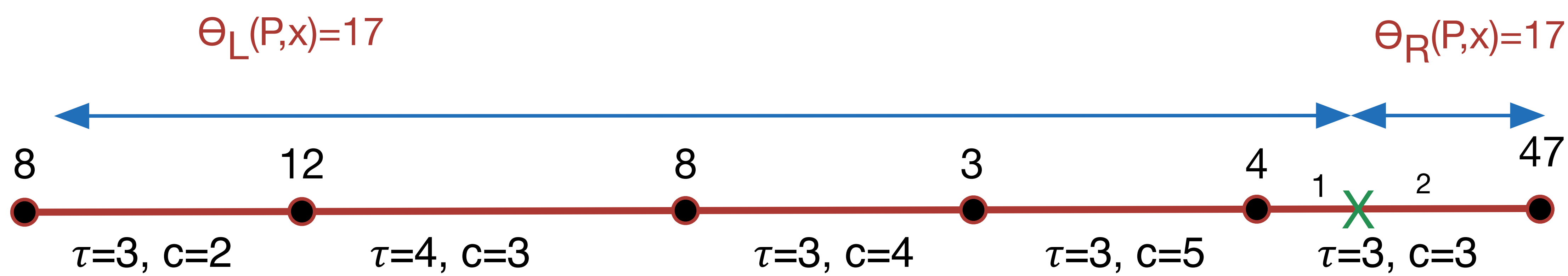


1-Sink Evacuation Example

Original Input:

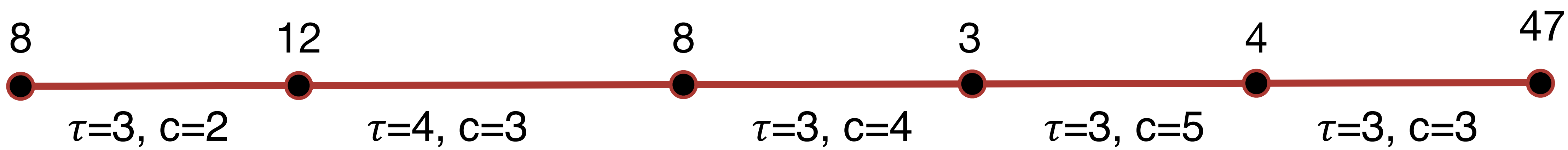


X is the sink location that minimizes Maximum Evacuation Time

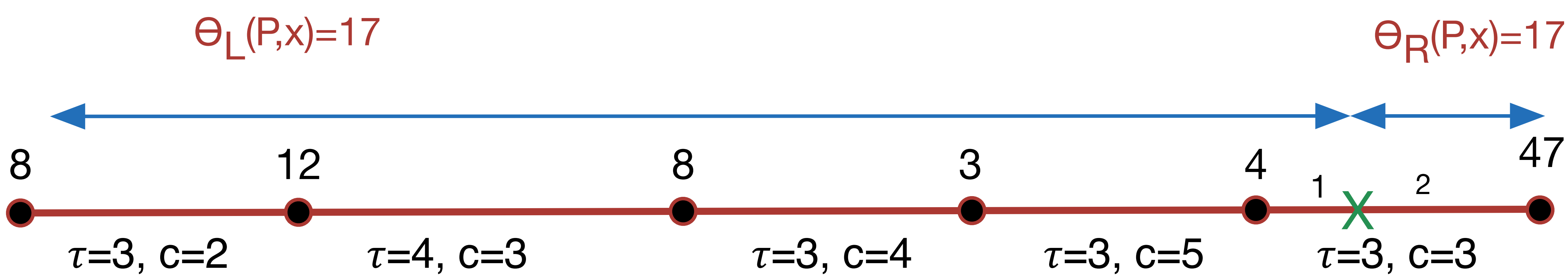


1-Sink Evacuation Example

Original Input:

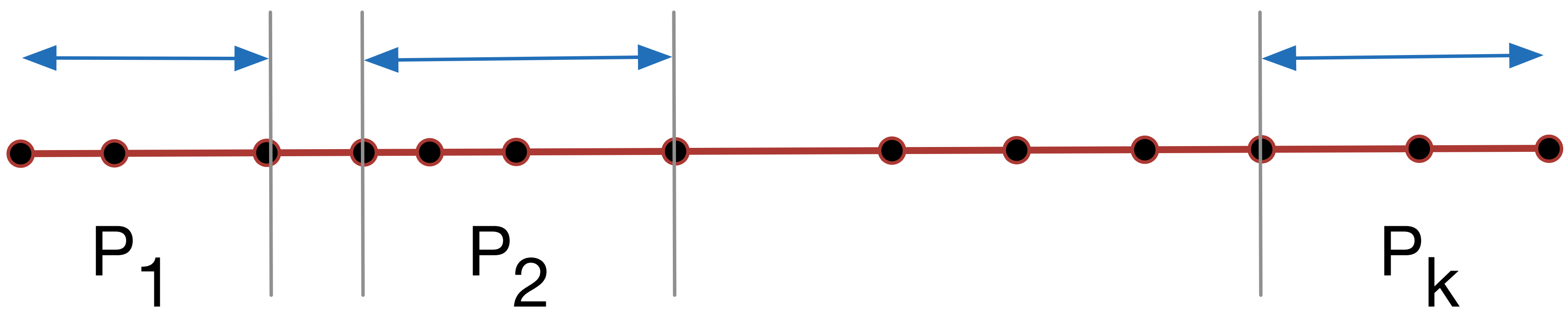


X is the sink location that minimizes Maximum Evacuation Time



Note: Min evac-time sink location is NOT an original vertex.
Can modify problem definition to require sink to be a vertex
Algorithms remain almost the same

k-Sink Evacuation Notation



- Given Path P and integer k
- $\mathbb{P} = \{P_1, P_2, \ldots, P_k\}$ is a partition of P into k-subpaths
- Given $\mathbb{P}$, the evacuation time of P is
  $$\max\left(\Theta^1(P_1), \Theta^1(P_2), \ldots, \Theta^1(P_k)\right)$$
- Want to find
  $$\Theta^k(P) = \min_{\mathbb{P}} \left(\max\left(\Theta^1(P_1), \Theta^1(P_2), \ldots, \Theta^1(P_k)\right) \right)$$
  $=$ Min k-sink evacuation time for P

Algorithm Development Sketch

1. Formulae for $\Theta_L(P,x)$ and $\Theta_L(P,x)$

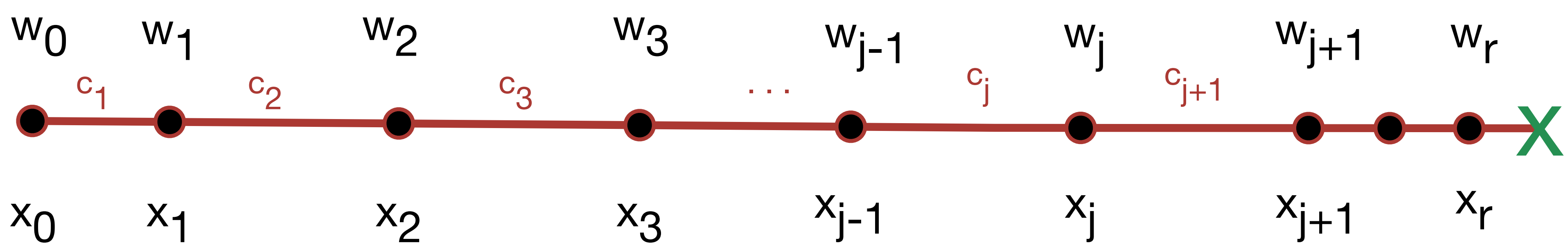
2. => $O(|P|)$ Algorithm for $\Theta_L(P,x)$, $\Theta_L(P,x)$

3. => $O(|P| \log |P|)$ Algorithm for $\Theta^1(P)$

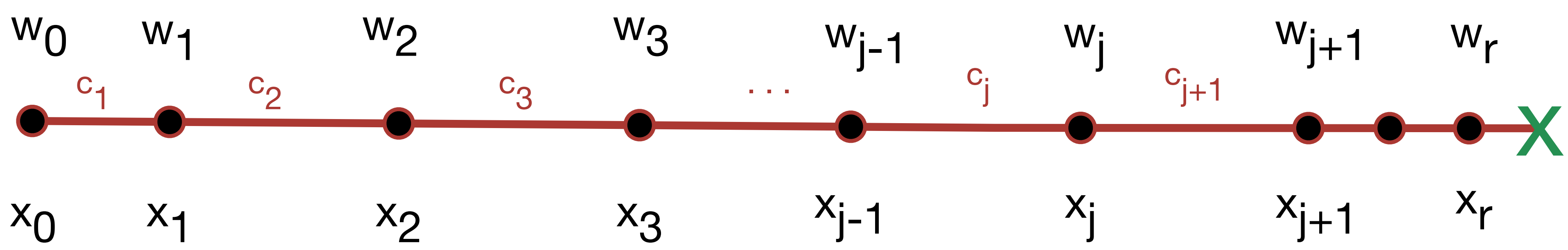
4. => $O(|P| \log |P|)$ Algorithm that $\forall \alpha > 0$
   tests whether $\Theta^k(P) \leq \alpha$

5. => $O(k|P| \log^2 |P|)$ Algorithm for $\Theta^k(P)$

Formulae for $\mathbf{\Theta}_L(P,x)$ and $\mathbf{\Theta}_R(P,x)$

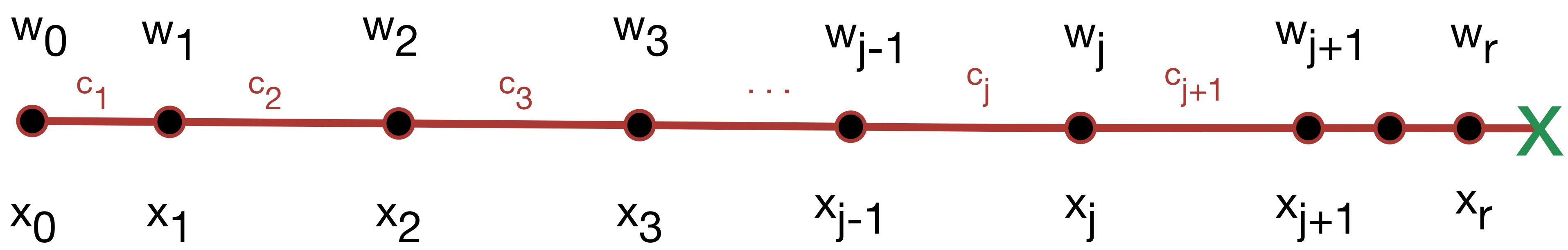


Formulae for $\Theta_L(P,x)$ and $\Theta_R(P,x)$



- Consider $\Theta_L(P,x)$ with sink at right

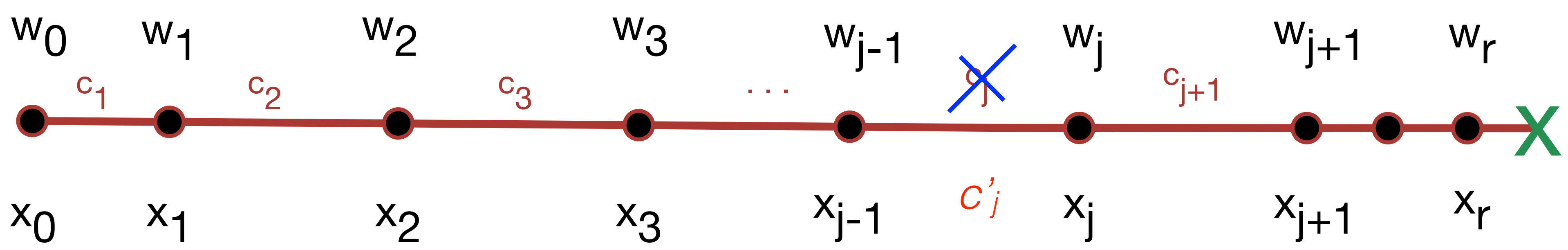
Formulae for $\Theta_L(P,x)$ and $\Theta_R(P,x)$



- Consider $\Theta_L(P,x)$ with sink at right

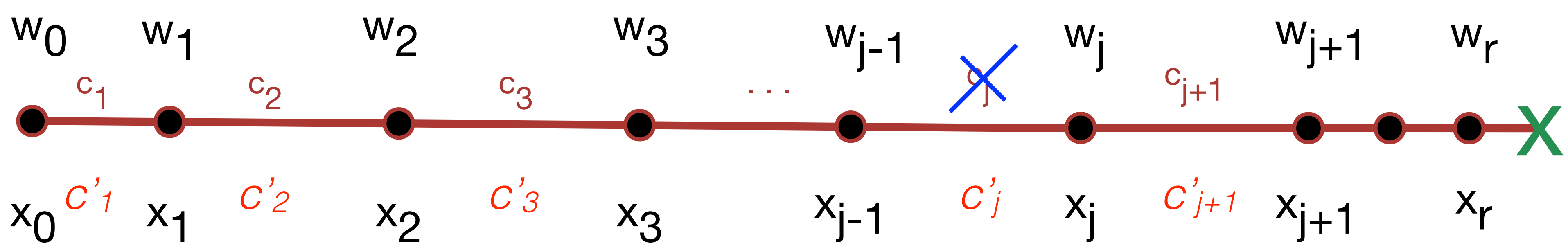
- **Lemma:** Suppose $c_j > c_{j+1}$.
  Create P' by replacing c_j with $c'_j = c_{j+1}$.
  => Then $\Theta_L(P,x) = \Theta_L(P',x)$

Formulae for $\Theta_L(P,x)$ and $\Theta_R(P,x)$



- Consider $\Theta_L(P,x)$ with sink at right
- **Lemma:** Suppose $c_j > c_{j+1}$.
  Create P' by replacing c_j with $c'_j = c_{j+1}$.
  =>Then $\Theta_L(P,x) = \Theta_L(P',x)$

Formulae for $\Theta_L(P,x)$ and $\Theta_R(P,x)$



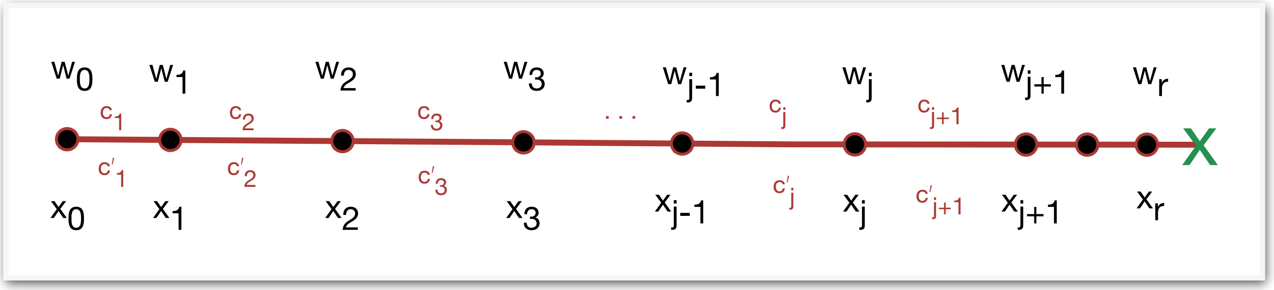
- Consider $\Theta_L(P,x)$ with sink at right

- **Lemma:** Suppose $c_j > c_{j+1}$.
  Create P' by replacing c_j with $c'_j = c_{j+1}$.
  => Then $\Theta_L(P,x) = \Theta_L(P',x)$

- **Corollary:** May replace capacities by

$$c'_1 \leq c'_2 \leq c'_3 \leq \ldots \leq c'_n \qquad c'_i = \min_{i \leq j \leq r+1} c_j$$

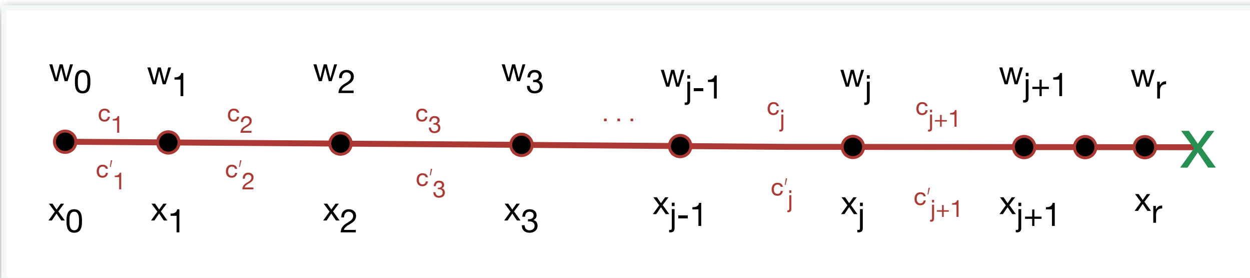
Formula for $\Theta_L(P,x)$



x_r is last vertex to right of x

Path with c_i has same evac time as Path with $c'_i = \min_{i \le j \le r+1} c_j$

Formula for $\Theta_L(P,x)$



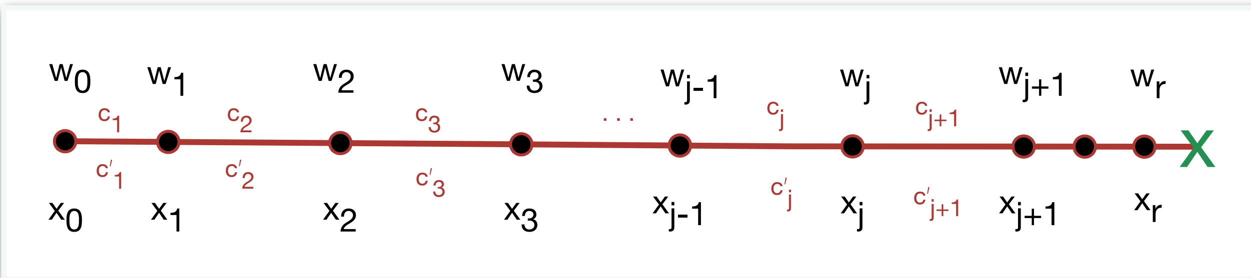
x_r is last vertex to right of x

Path with c_i has same evac time as Path with $c'_i = \min_{i \le j \le r+1} c_j$

Lemma:

$$\Theta_L(P,x) = \max_{0 \le t \le r} \left(\left((x - x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1 \right) \right) \qquad W_t = \sum_{0 \le j \le t} w_j$$

Formula for $\Theta_L(P,x)$



x_r is last vertex to right of x

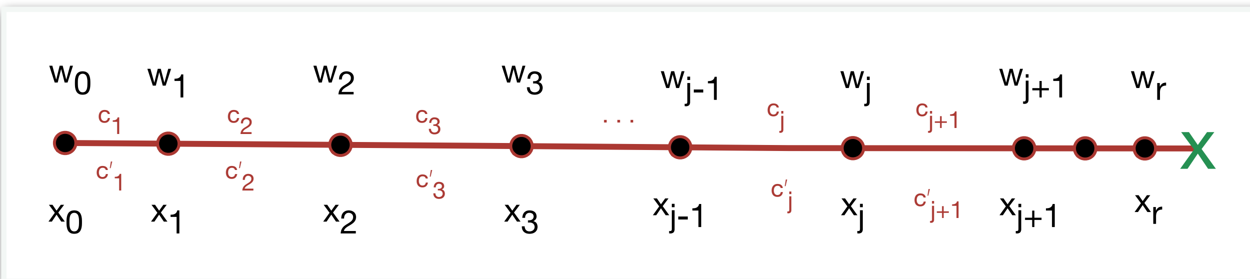
Path with c_i has same evac time as Path with $c'_i = \min_{i \le j \le r+1} c_j$

Lemma:

$$\Theta_L(P,x) = \max_{0 \le t \le r} \left(\left((x - x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1 \right) \right) \qquad W_t = \sum_{0 \le j \le t} w_j$$

Intuition: Analysis is on path P'

Formula for $\Theta_L(P,x)$



x_r is last vertex to right of x

Path with c_i has same evac time as Path with $c'_i = \min_{i \le j \le r+1} c_j$

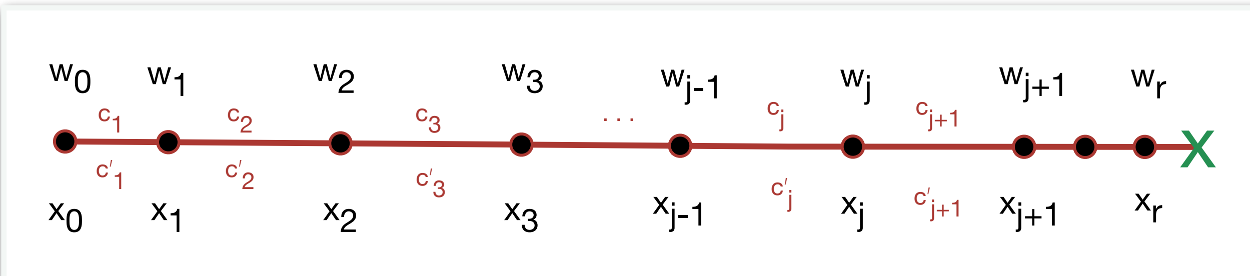
Lemma:

$$\Theta_L(P,x) = \max_{0 \le t \le r} \left(\left((x - x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1 \right) \right) \qquad W_t = \sum_{0 \le j \le t} w_j$$

Intuition: Analysis is on path P'

- Fix x_t. $x - x_t$ is uncongested travel time from x_t to x

Formula for $\Theta_L(P,x)$



x_r is last vertex to right of x

Path with c_i has same evac time as Path with $c'_i = \min_{i \le j \le r+1} c_j$

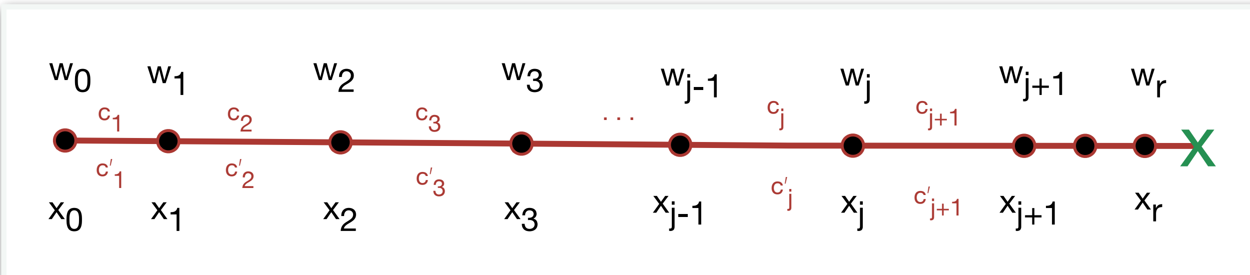
Lemma:

$$\Theta_L(P,x) = \max_{0 \le t \le r} \left(\left((x - x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1 \right) \right) \qquad W_t = \sum_{0 \le j \le t} w_j$$

Intuition: Analysis is on path P'

- Fix x_t. $x - x_t$ is uncongested travel time from x_t to x
- Remove all items to right of x_t.
  Move all items to left of x_t onto x_t. x_t's new weight is W_t

Formula for $\Theta_L(P,x)$



x_r is last vertex to right of x

Path with c_i has same evac time as Path with $c'_i = \min_{i \le j \le r+1} c_j$

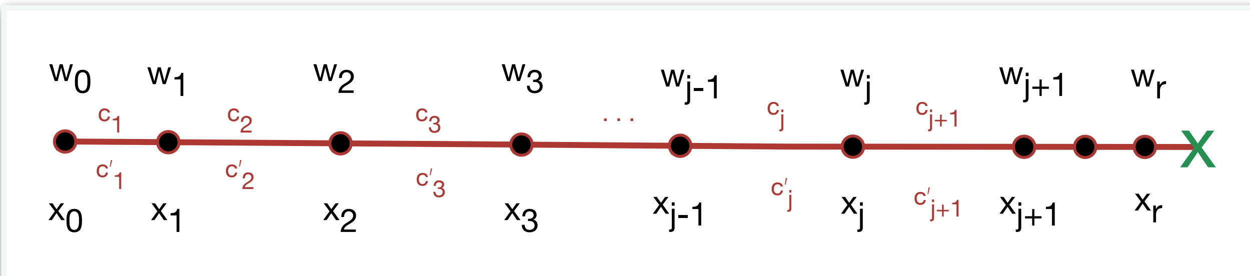
Lemma:

$$\Theta_L(P,x) = \max_{0 \le t \le r} \left(\left((x - x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1 \right) \right) \qquad W_t = \sum_{0 \le j \le t} w_j$$

Intuition: Analysis is on path P'

- Fix x_t. $x - x_t$ is uncongested travel time from x_t to x
- Remove all items to right of x_t.
  Move all items to left of x_t onto x_t. x_t's new weight is W_t
- # of groups leaving x_t is $g = \lceil W_t / c'_{t+1} \rceil$.
  No congestion on path to x.

Formula for $\Theta_L(P,x)$



x_r is last vertex to right of x

Path with c_i has same evac time as Path with $c'_i = \min_{i \le j \le r+1} c_j$

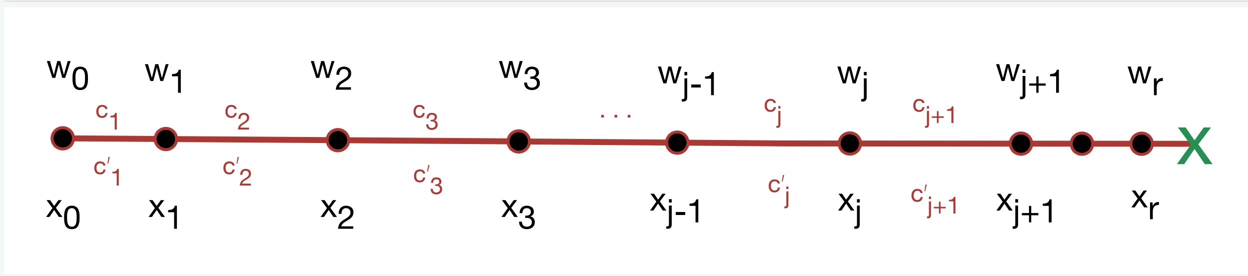
Lemma:

$$\Theta_L(P,x) = \max_{0 \le t \le r} \left(\left((x - x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1 \right) \right) \qquad W_t = \sum_{0 \le j \le t} w_j$$

Intuition: Analysis is on path P'

- Fix x_t. $x - x_t$ is uncongested travel time from x_t to x
- Remove all items to right of x_t.
  Move all items to left of x_t onto x_t. x_t's new weight is W_t
- # of groups leaving x_t is $g = \lceil W_t / c'_{t+1} \rceil$.
  No congestion on path to x.
- => $x - x_t + g - 1$ is the exact evacuation time for items on x_t

Formula for $\Theta_L(P,x)$

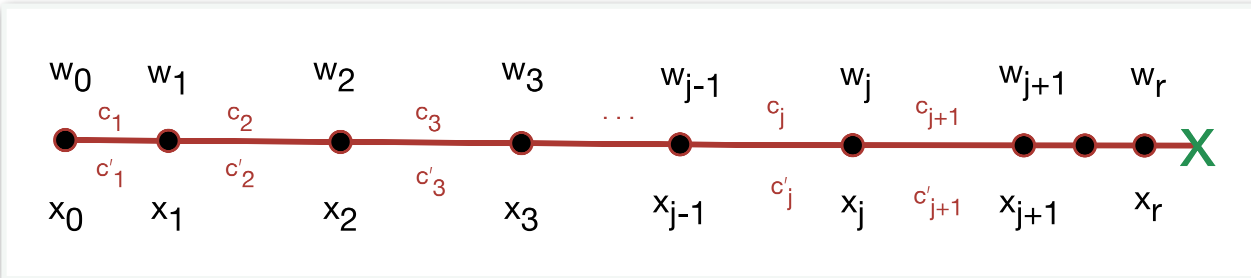


Path with c_i has same evac time as Path with $c'_i = \min_{i \le j \le r+1} c_j$

Lemma:

$$\Theta_L(P,x) = \max_{0 \le t \le r} \left(\left((x - x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1 \right) \right) \qquad W_t = \sum_{0 \le j \le t} w_j$$

Formula for $\Theta_L(P,x)$



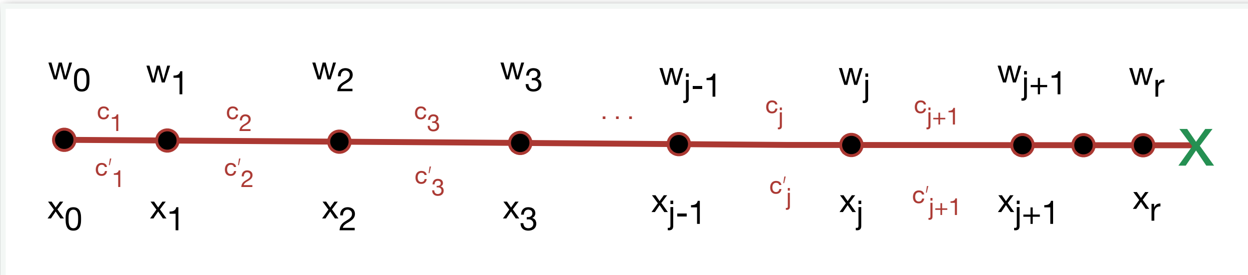
Path with c_i has same evac time as Path with $c'_i = \min_{i \le j \le r+1} c_j$

Lemma:

$$\Theta_L(P,x) = \max_{0 \le t \le r} \left(\left((x - x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1 \right) \right) \qquad W_t = \sum_{0 \le j \le t} w_j$$

- Fix vertex x_t and consider the W_t items passing through x_t

Formula for $\Theta_L(P,x)$



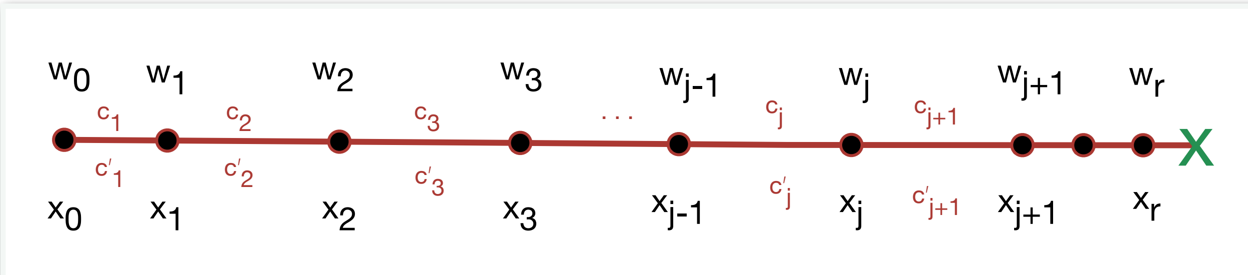
Path with c_i has same evac time as Path with $c'_i = \min_{i \le j \le r+1} c_j$

Lemma:

$$\Theta_L(P,x) = \max_{0 \le t \le r} \left(\left((x - x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1 \right) \right) \qquad W_t = \sum_{0 \le j \le t} w_j$$

- Fix vertex x_t and consider the W_t items passing through x_t
- => These W_t items leave x_t in $g \ge \lceil W_t / c'_{t+1} \rceil$ groups
  => Last group leaves x_t at time $\ge g-1$.

Formula for $\Theta_L(P,x)$



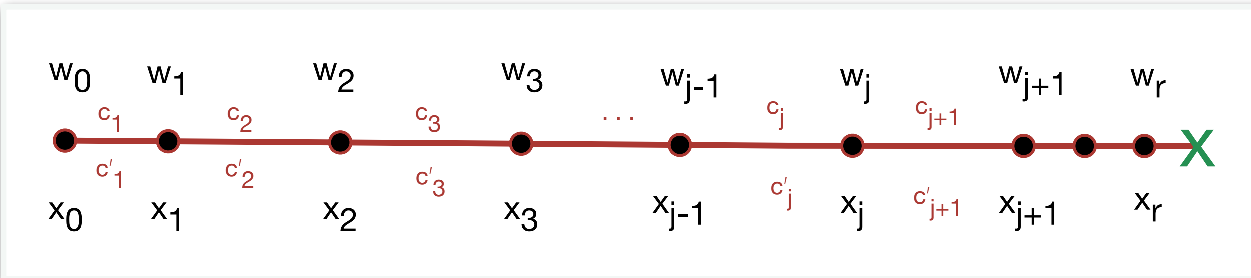
Path with c_i has same evac time as Path with $c'_i = \min_{i \le j \le r+1} c_j$

Lemma:

$$\Theta_L(P,x) = \max_{0 \le t \le r} \left(\left((x - x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1 \right) \right) \qquad W_t = \sum_{0 \le j \le t} w_j$$

- Fix vertex x_t and consider the W_t items passing through x_t
- => These W_t items leave x_t in $g \ge \lceil W_t / c'_{t+1} \rceil$ groups
  => Last group leaves x_t at time $\ge g-1$.
- Last item in last group requires at least $x - x_t$ time to move from x_t to x
  => final evacuation time $\ge$ $x - x_t + g - 1$

Formula for $\Theta_L(P,x)$



Path with c_i has same evac time as Path with $c'_i = \min_{i \le j \le r+1} c_j$

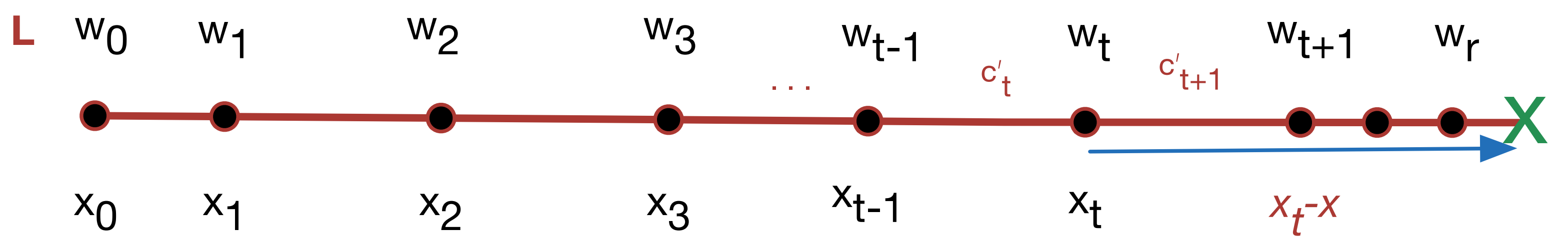
Lemma:

$$\Theta_L(P,x) = \max_{0 \le t \le r} \left(\left((x - x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1 \right) \right) \qquad W_t = \sum_{0 \le j \le t} w_j$$

- Fix vertex x_t and consider the W_t items passing through x_t
- => These W_t items leave x_t in $g \ge \lceil W_t/c'_{t+1} \rceil$ groups
  => Last group leaves x_t at time $\ge g-1$.
- Last item in last group requires at least $x - x_t$ time to move from x_t to x
  => final evacuation time $\ge$ $x - x_t + g-1$
- This is true for every t
  => have just proven **≥** direction of lemma

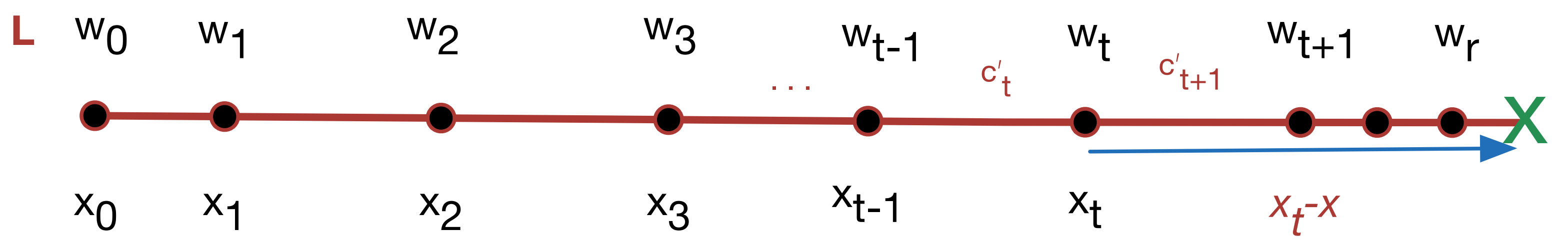
Lemma: $\Theta_L(P,x) = \max_{0\le t\le r}\left(\left((x-x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1\right)\right) \quad W_t = \sum_{0\le j\le t} w_j$

- Let L be last item on x_0 and x_t be last vertex at which L is congested (waits).
  (If L never experiences congestion set t=0.)
  => If L leaves x_t at time T′, L arrives at x at time T′+ x-x_t



Lemma: $\Theta_L(P,x) = \max_{0\le t\le r}\left(\left((x-x_t)+\left\lceil \frac{W_t}{c'_{t+1}}\right\rceil - 1\right)\right) \quad W_t = \sum_{0\le j\le t} w_j$

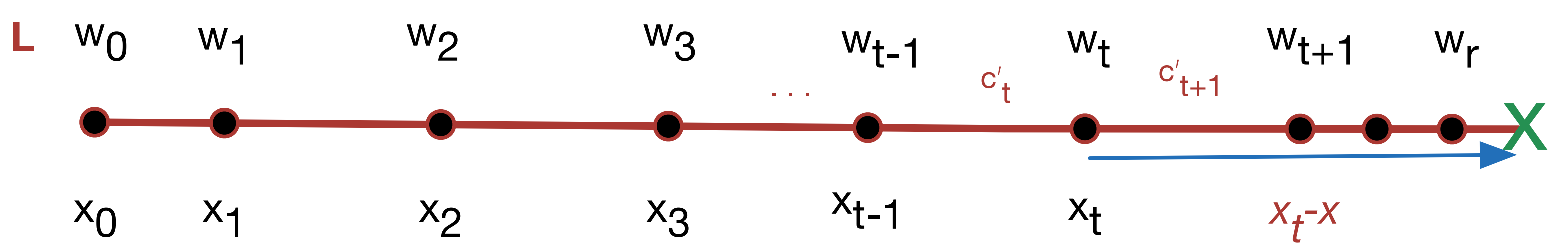
- Let L be last item on x_0 and x_t be last vertex at which L is congested (waits). *(If L never experiences congestion set t=0.)*
  => If L leaves x_t at time T′, L arrives at x at time T′+ x-x_t



- Note: # people arriving at x_t at any time T is $\le c'_t \le c'_{t+1}$

Lemma: $\Theta_L(P,x) = \max_{0\le t\le r}\left(\left((x-x_t)+\left\lceil \frac{W_t}{c'_{t+1}}\right\rceil - 1\right)\right)$ $\quad W_t = \sum_{0\le j\le t} w_j$

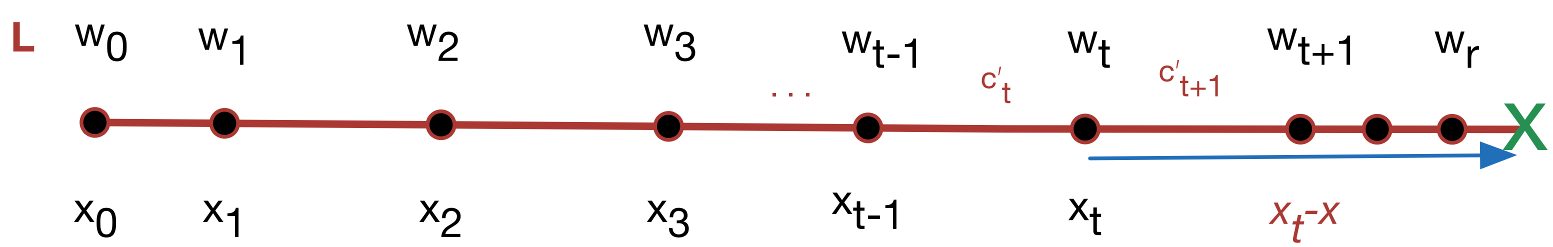
- Let L be last item on x_0 and x_t be last vertex at which L is congested (waits).
  (If L never experiences congestion set t=0.)
  => If L leaves x_t at time T′, L arrives at x at time T′+ x-x_t



- Note: # people arriving at x_t at any time T is $\le\ c'_t \le c'_{t+1}$
- Suppose ∃ timestep T≥0 at which < c'_{t+1} items leave x_t.
  => no one is left waiting at x_t.
  => at T+1 the $\le c'_{t+1}$ people arriving at x_t all pass through without waiting at x_t
  => repeating; no one left waiting at x_t at T+2, T+3, etc.
  => L passes through x_t without waiting, contradicting choice of t.

Lemma: $\Theta_L(P,x) = \max_{0\le t\le r}\left(\left((x-x_t)+\left\lceil \frac{W_t}{c'_{t+1}}\right\rceil - 1\right)\right) \quad W_t = \sum_{0\le j\le t} w_j$

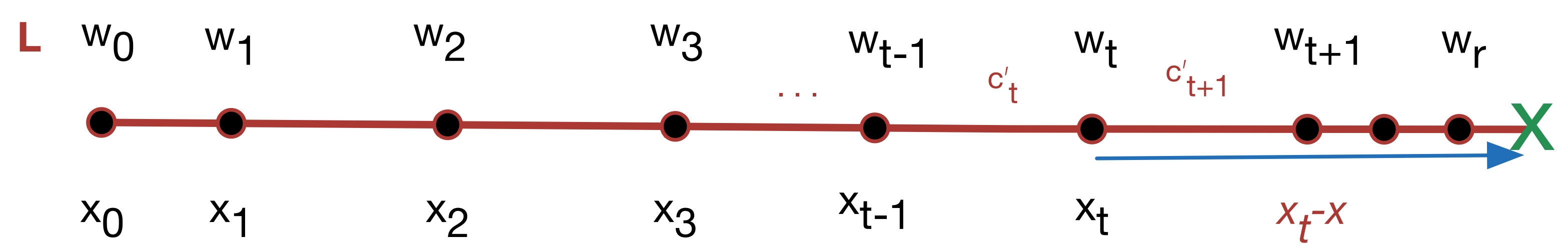
- Let L be last item on x_0 and x_t be last vertex at which L is congested (waits).
  (If L never experiences congestion set t=0.)
  => If L leaves x_t at time T′, L arrives at x at time T′+ x-x_t



- Note: # people arriving at x_t at any time T is $\le c'_t \le c'_{t+1}$
- Suppose ∃ timestep T≥0 at which < c'_{t+1} items leave x_t.
  => no one is left waiting at x_t.
  => at T+1 the $\le c'_{t+1}$ people arriving at x_t all pass through without waiting at x_t
  => repeating; no one left waiting at x_t at T+2, T+3, etc.
  => L passes through x_t without waiting, contradicting choice of t.
- => At every time step exactly c'_{t+1} people leave x_t

Lemma: $\Theta_L(P,x) = \max_{0\le t\le r}\left(\left((x-x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1\right)\right) \quad W_t = \sum_{0\le j\le t} w_j$

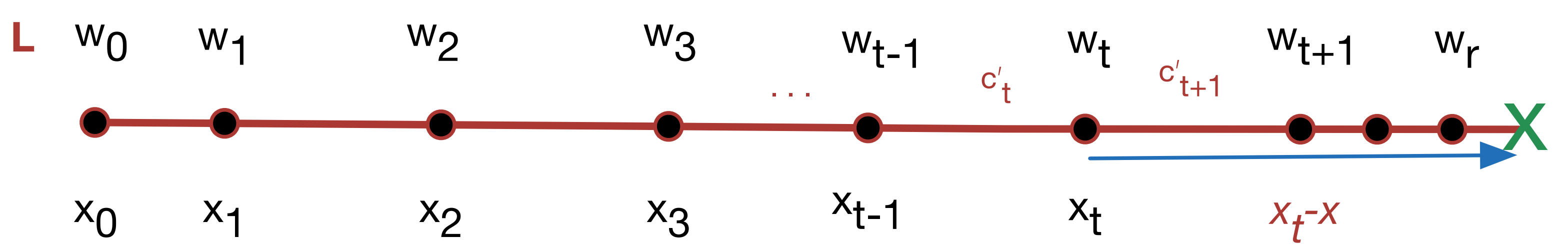
- Let L be last item on x_0 and x_t be last vertex at which L is congested (waits).
  (If L never experiences congestion set t=0.)
  => If L leaves x_t at time T′, L arrives at x at time T′+ x-x_t



- Note: # people arriving at x_t at any time T is $\le\ c'_t \le c'_{t+1}$
- Suppose ∃ timestep T≥0 at which < c'_{t+1} items leave x_t.
  => no one is left waiting at x_t.
  => at T+1 the $\le c'_{t+1}$ people arriving at x_t all pass through without waiting at x_t
  => repeating; no one left waiting at x_t at T+2, T+3, etc.
  => L passes through x_t without waiting, contradicting choice of t.
- => At every time step exactly c'_{t+1} people leave x_t
- => L leaves x_t in group g= $\lceil W_t/c'_{t+1} \rceil$ at time g -1

Lemma: $\Theta_L(P,x) = \max_{0\le t\le r}\left(\left((x-x_t) + \left\lceil \frac{W_t}{c'_{t+1}} \right\rceil - 1\right)\right) \quad W_t = \sum_{0\le j\le t} w_j$

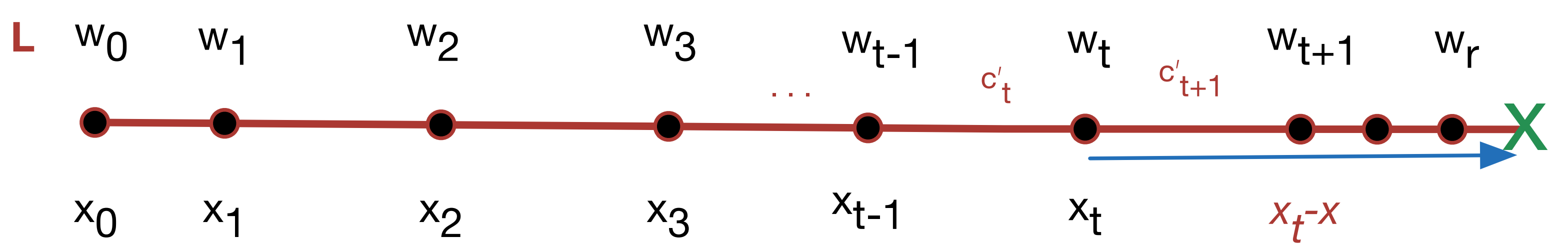
- Let L be last item on x_0 and x_t be last vertex at which L is congested (waits).
  (If L never experiences congestion set t=0.)
  => If L leaves x_t at time T′, L arrives at x at time T′+ x-x_t



- Note: # people arriving at x_t at any time T is $\le c'_t \le c'_{t+1}$
- Suppose ∃ timestep T≥0 at which < c'_{t+1} items leave x_t.
  => no one is left waiting at x_t.
  => at T+1 the $\le c'_{t+1}$ people arriving at x_t all pass through without waiting at x_t
  => repeating; no one left waiting at x_t at T+2, T+3, etc.
  => L passes through x_t without waiting, contradicting choice of t.
- => At every time step exactly c'_{t+1} people leave x_t
- => L leaves x_t in group g= $\lceil W_t/c'_{t+1} \rceil$ at time g -1
- => L arrives at x at time x- x_t + g-1

Lemma: $\Theta_L(P,x) = \max_{0\le t\le r}\left(\left((x-x_t)+\left\lceil \frac{W_t}{c'_{t+1}}\right\rceil - 1\right)\right) \quad W_t = \sum_{0\le j\le t} w_j$

- Let L be last item on x_0 and x_t be last vertex at which L is congested (waits).
  (If L never experiences congestion set t=0.)
  => If L leaves x_t at time T′, L arrives at x at time T′+ $x-x_t$

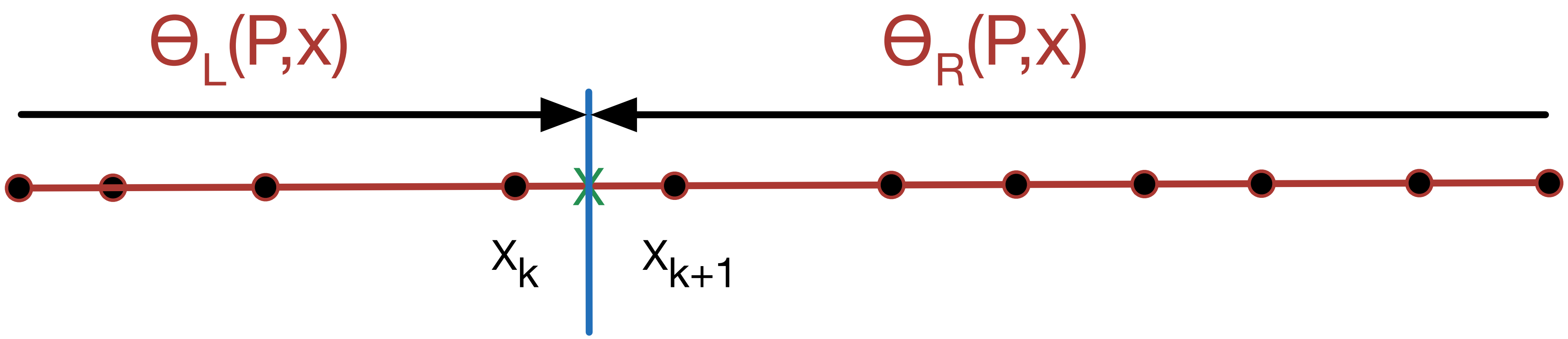


- Note: # people arriving at x_t at any time T is $\le c'_t \le c'_{t+1}$
- Suppose ∃ timestep T≥0 at which < c'_{t+1} items leave x_t.
  => no one is left waiting at x_t.
  => at T+1 the $\le c'_{t+1}$ people arriving at x_t all pass through without waiting at x_t
  => repeating; no one left waiting at x_t at T+2, T+3, etc.
  => L passes through x_t without waiting, contradicting choice of t.
- => At every time step exactly c'_{t+1} people leave x_t
- => L leaves x_t in group $g = \lceil W_t/c'_{t+1} \rceil$ at time g -1
- => L arrives at x at time x- x_t + g-1
- => have just proven **≤** direction of lemma

Algorithm Development Sketch

1. Formulae for $\Theta_L(P,x)$ and $\Theta_L(P,x)$
2. => $O(|P|)$ Algorithm for $\Theta_L(P,x)$, $\Theta_L(P,x)$
3. => $O(|P| \log |P|)$ Algorithm for $\Theta^1(P)$
4. => $O(|P| \log |P|)$ Algorithm that $\forall \alpha > 0$ tests whether $\Theta^k(P) \leq \alpha$
5. => $O(k|P| \log^2 |P|)$ Algorithm for $\Theta^k(P)$

Formulas for $\Theta_L(P,x)$ and $\Theta_R(P,x)$

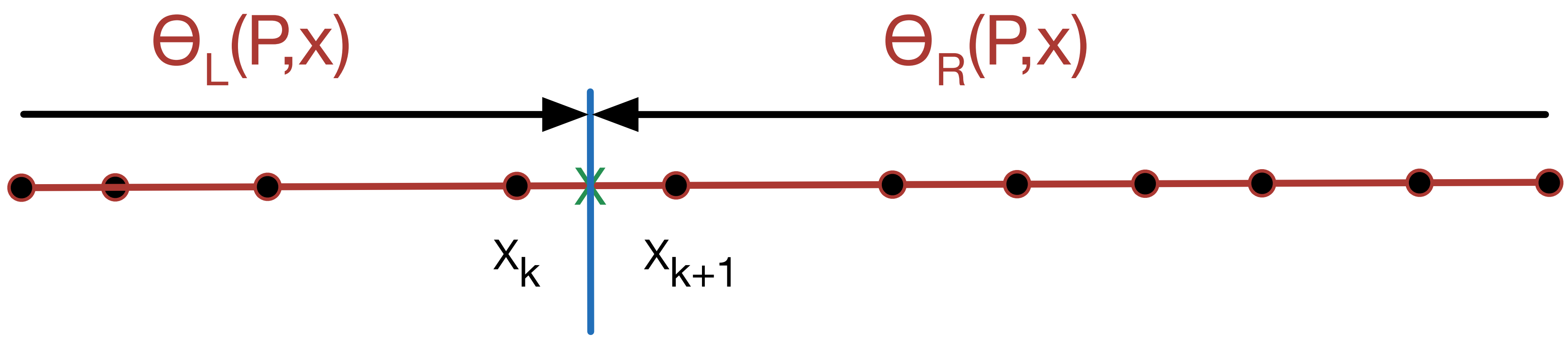


Theorem: Let k be s.t. $x_k < x \leq x_{k+1}$. Then

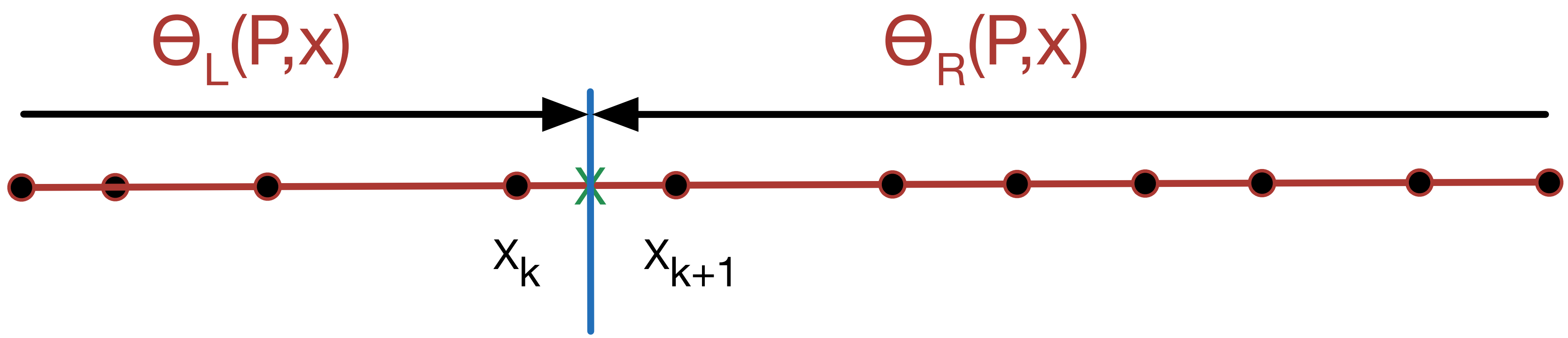
$$\Theta_L(P,x) = \max_{x_i < x}\left((x - x_i) + \left\lceil \frac{\sum_{0\leq j\leq t} w_j}{\min_{i+1\leq j\leq k+1} c_j} \right\rceil + 1\right) \qquad \Theta_R(P,x) = \max_{x_i > x}\left((x_i - x) + \left\lceil \frac{\sum_{i\leq j\leq n} w_j}{\min_{k+1\leq j\leq n} c_j} \right\rceil + 1\right)$$

Corollary: $\Theta_L(P,x)$ and $\Theta_R(P,x)$ can be computed in $O(|P|)$ time

Formulas for $\Theta_L(P,x)$ and $\Theta_R(P,x)$

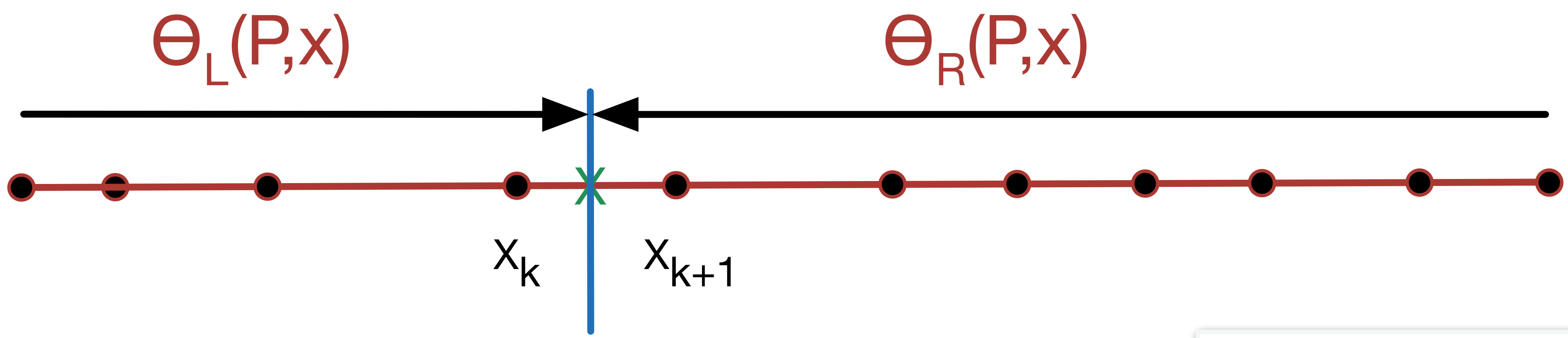


Formulas for $\Theta_L(P,x)$ and $\Theta_R(P,x)$

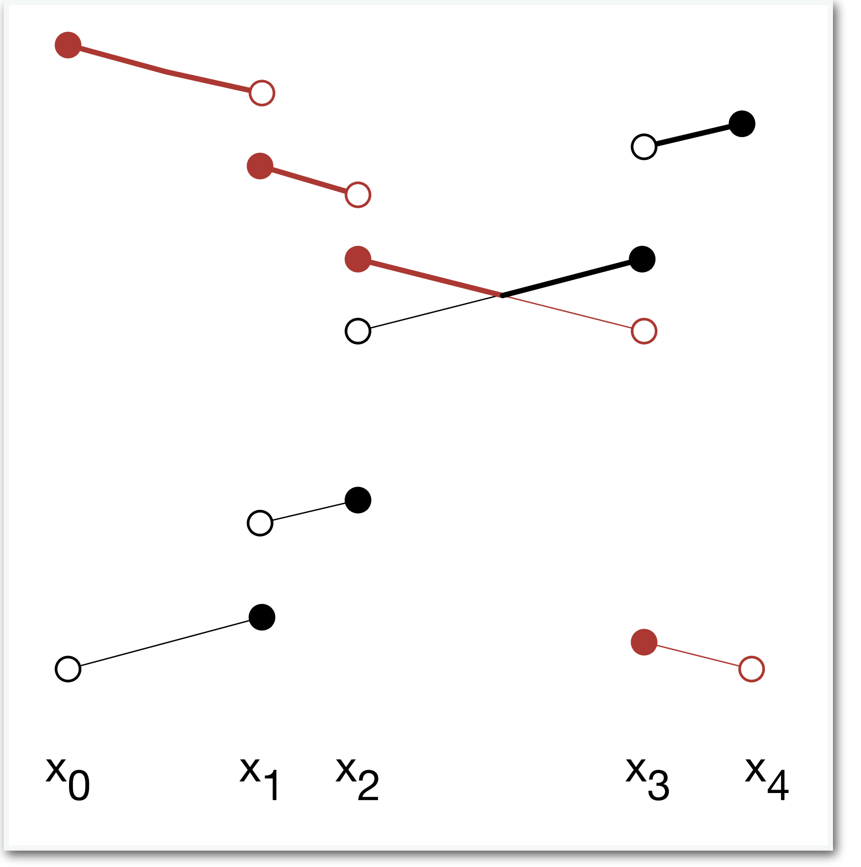


Claim 1: $\Theta_L(P,x)$ ($\Theta_R(P,x)$) is a monotonically increasing (decreasing) piecewise linear function in x.

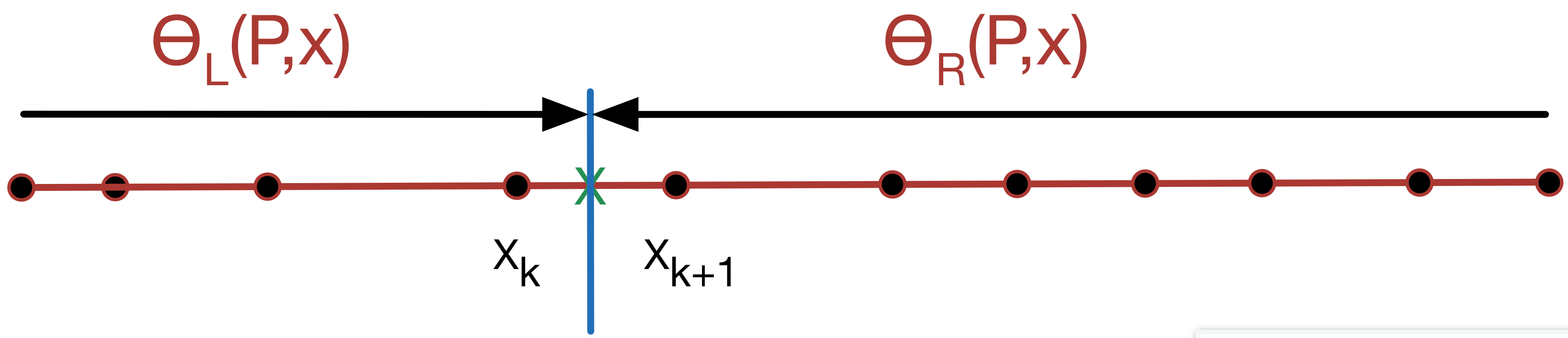
Formulas for $\Theta_L(P,x)$ and $\Theta_R(P,x)$

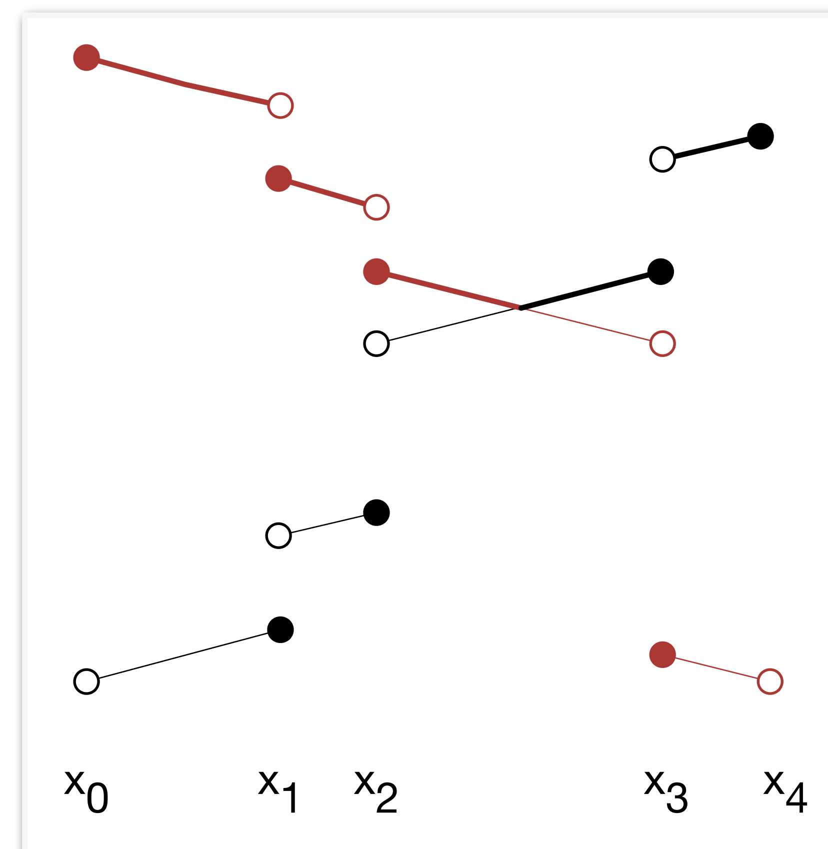


Claim 1: $\Theta_L(P,x)$ ($\Theta_R(P,x)$) is a monotonically increasing (decreasing) piecewise linear function in x.



Formulas for $\Theta_L(P,x)$ and $\Theta_R(P,x)$





Claim 1: $\Theta_L(P,x)$ ($\Theta_R(P,x)$) is a monotonically increasing (decreasing) piecewise linear function in x.

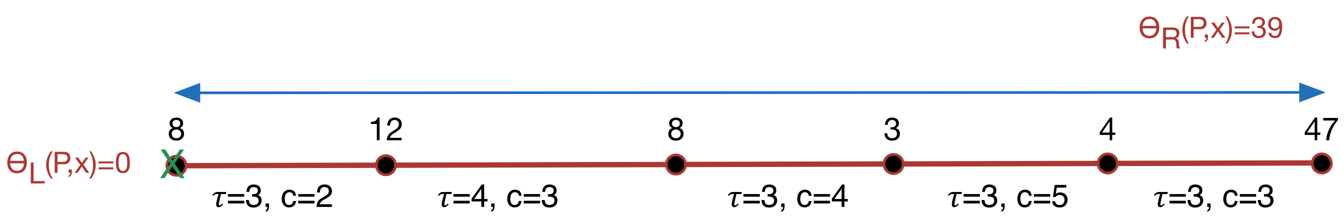
Claim 2: $\Theta(P,x) = \max(\Theta_L(P,x), \Theta_R(P,x))$ is a unimodal function. It decreases, achieves a unique minimum and then increases

Algorithm Development Sketch

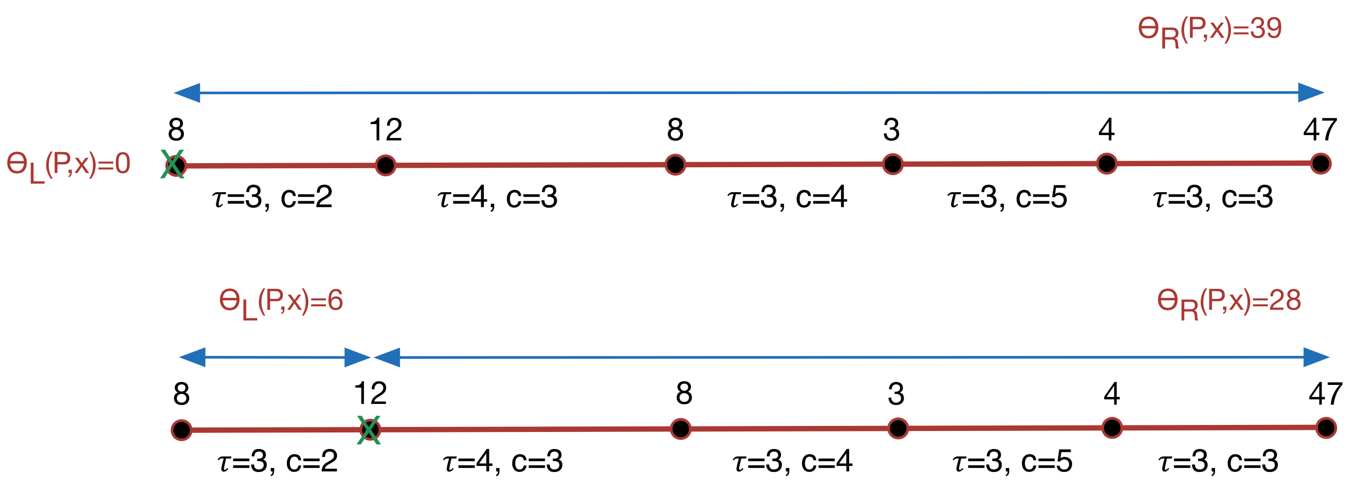
1. Formulae for $\Theta_L(P,x)$ and $\Theta_L(P,x)$
2. => $O(|P|)$ Algorithm for $\Theta_L(P,x)$, $\Theta_L(P,x)$
3. => $O(|P| \log |P|)$ Algorithm for $\Theta^1(P)$
4. => $O(|P| \log |P|)$ Algorithm that $\forall \alpha > 0$ tests whether $\Theta^k(P) \leq \alpha$
5. => $O(k|P| \log^2 |P|)$ Algorithm for $\Theta^k(P)$

An $O(|P| \log|P|)$ Algorithm for $\Theta^1(P)$

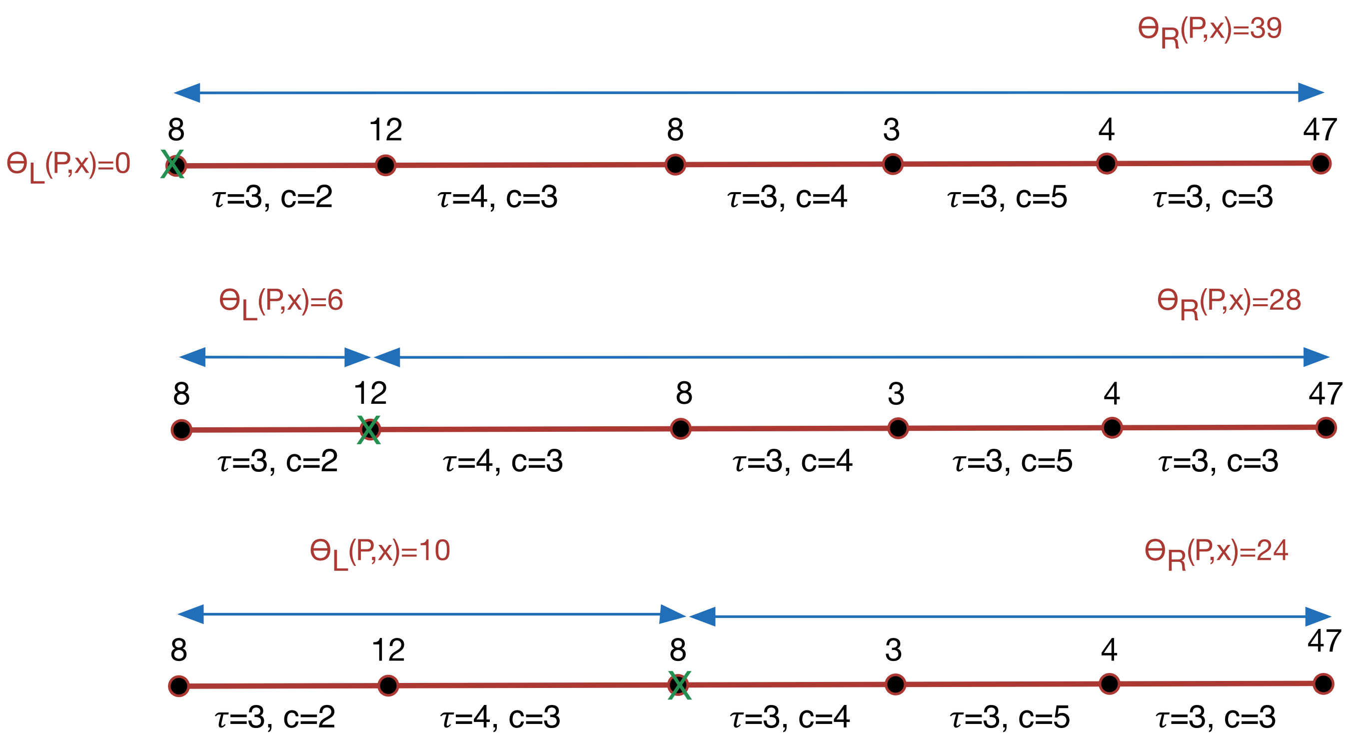
An O(|P| log|P|) Algorithm for $\Theta^1(P)$



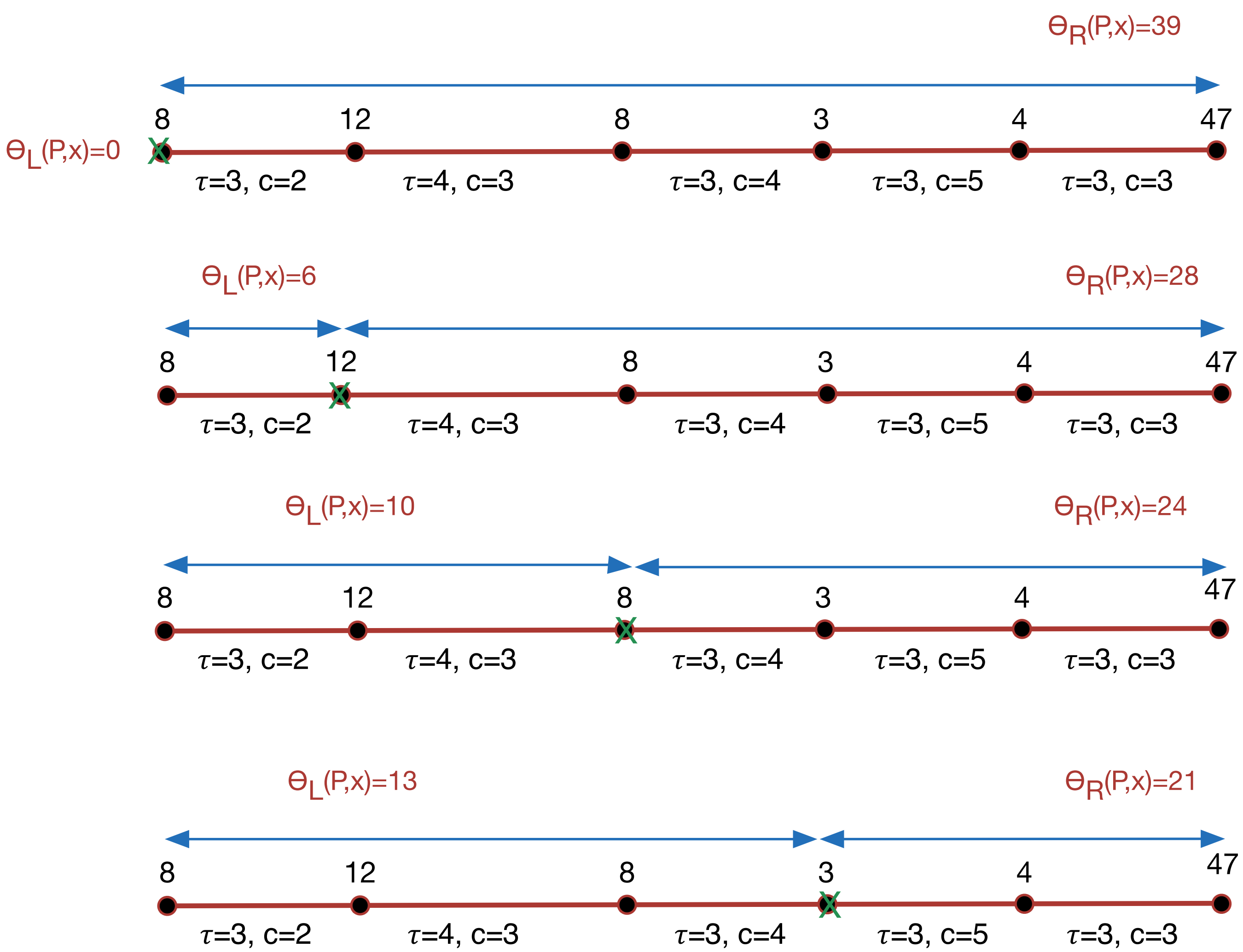
An $O(|P| \log|P|)$ Algorithm for $\Theta^1(P)$



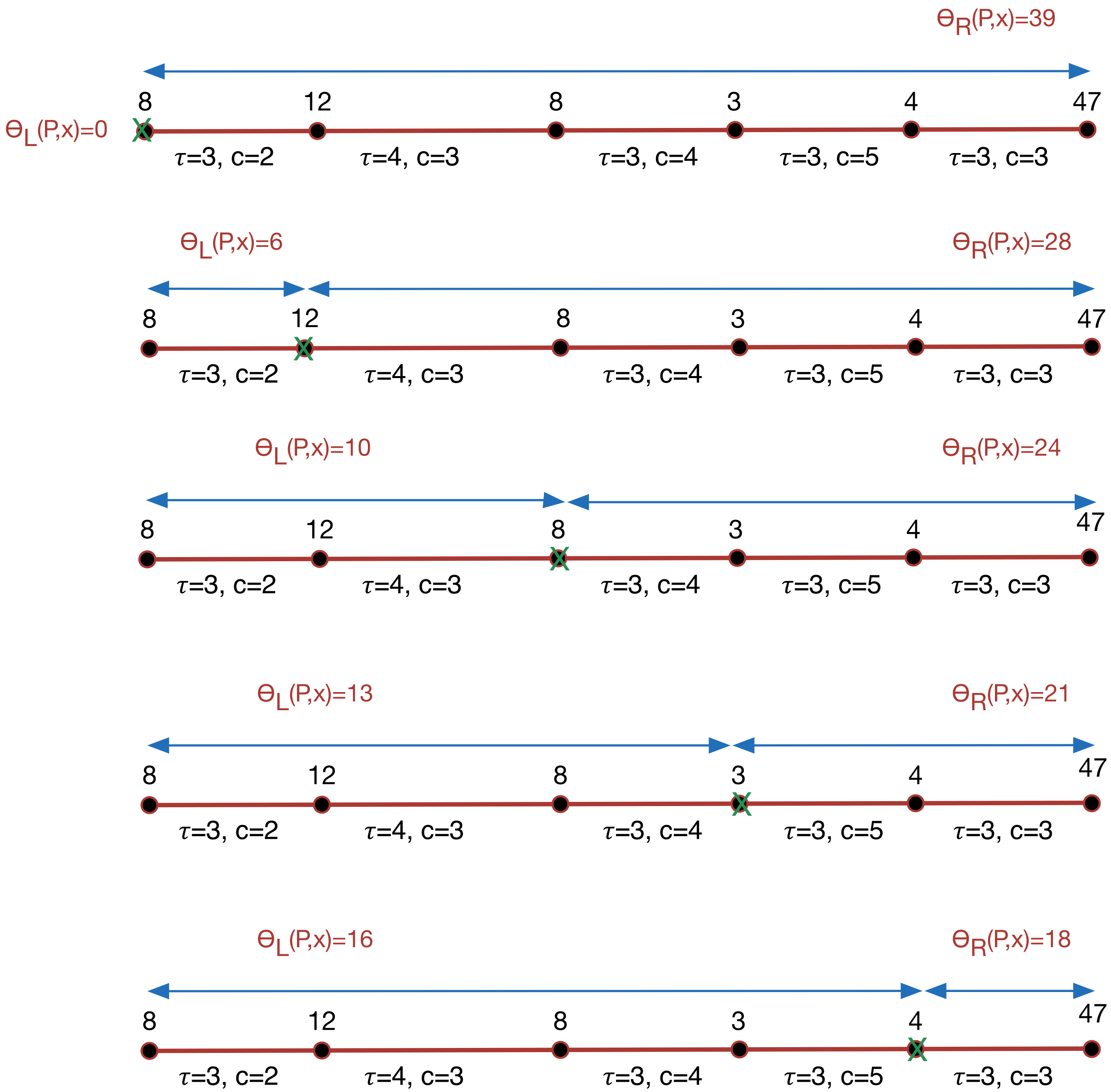
An $O(|P| \log|P|)$ Algorithm for $\Theta^1(P)$



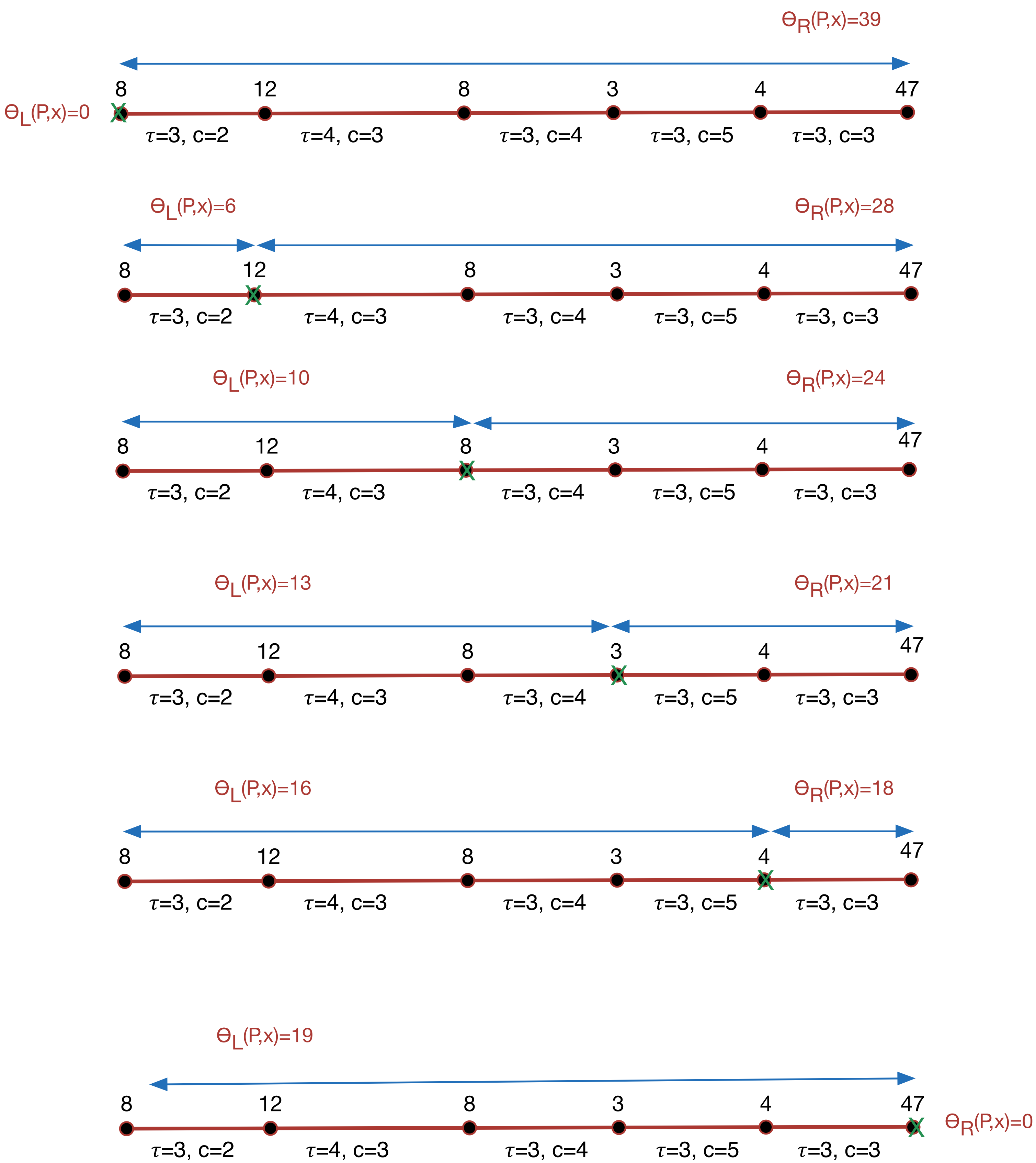
An O(|P| log|P|) Algorithm for $\Theta^1(P)$



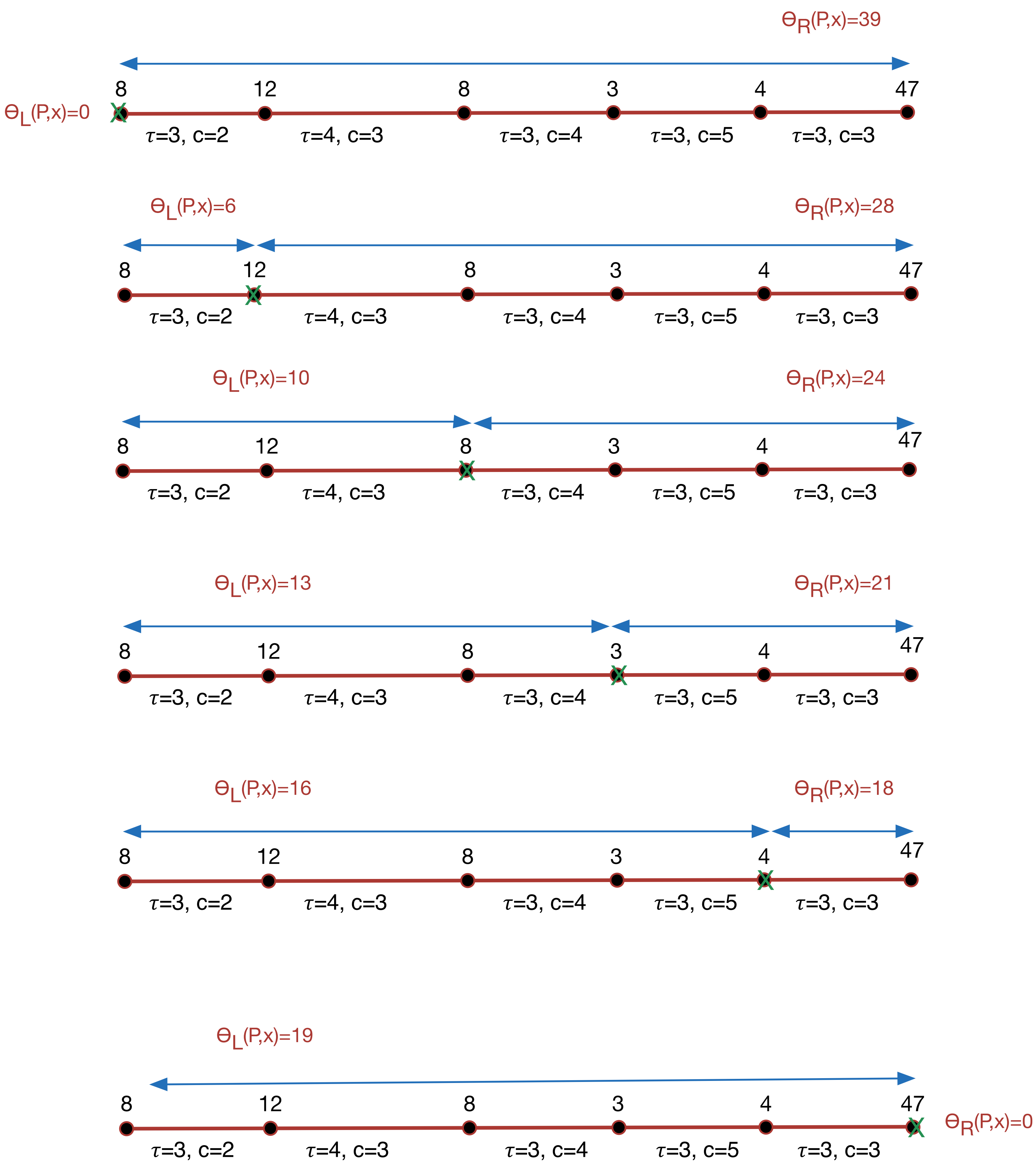
An O(|P| log|P|) Algorithm for $\Theta^1(P)$



An O(|P| log|P|) Algorithm for $\Theta^1(P)$



An O(|P| log|P|) Algorithm for $\Theta^1(P)$



Search for where

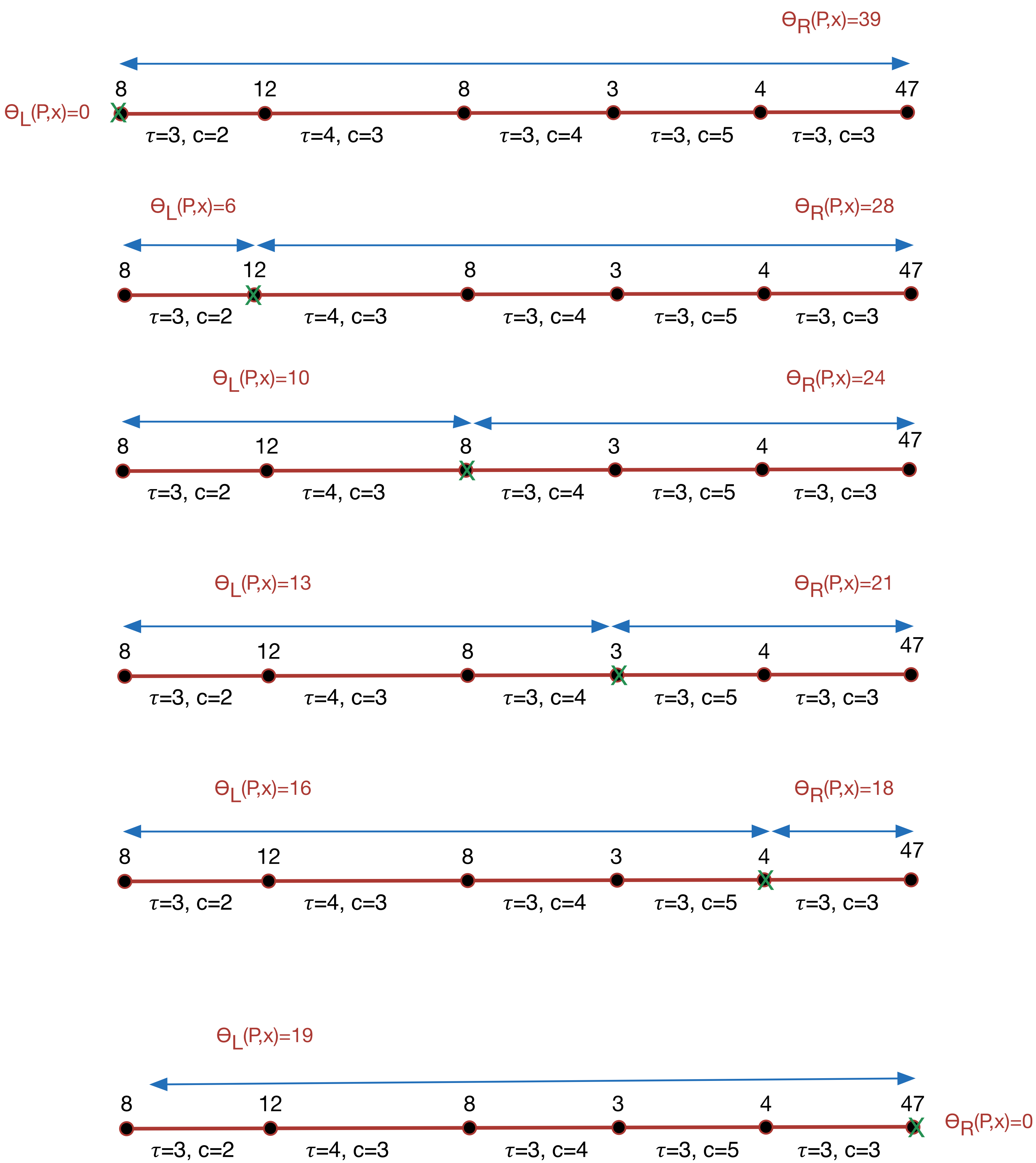
$\Theta_L(P,x_i) < \Theta_R(P,x_i)$

switches to

$\Theta_L(P,x_i) > \Theta_R(P,x_i)$.

Optimum sink x is in the interval where the switch occurs

An O(|P| log|P|) Algorithm for $\Theta^1(P)$



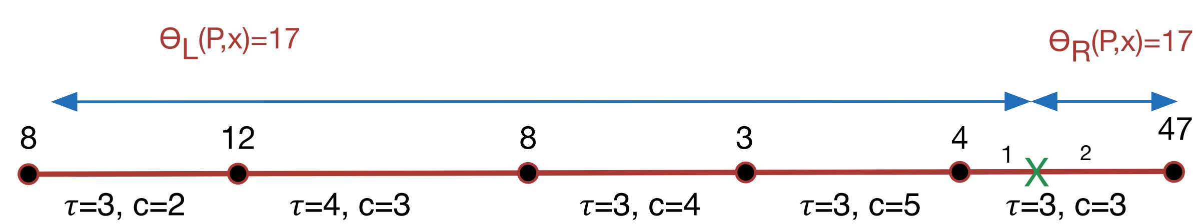
Search for where

$\Theta_L(P,x_i) < \Theta_R(P,x_i)$

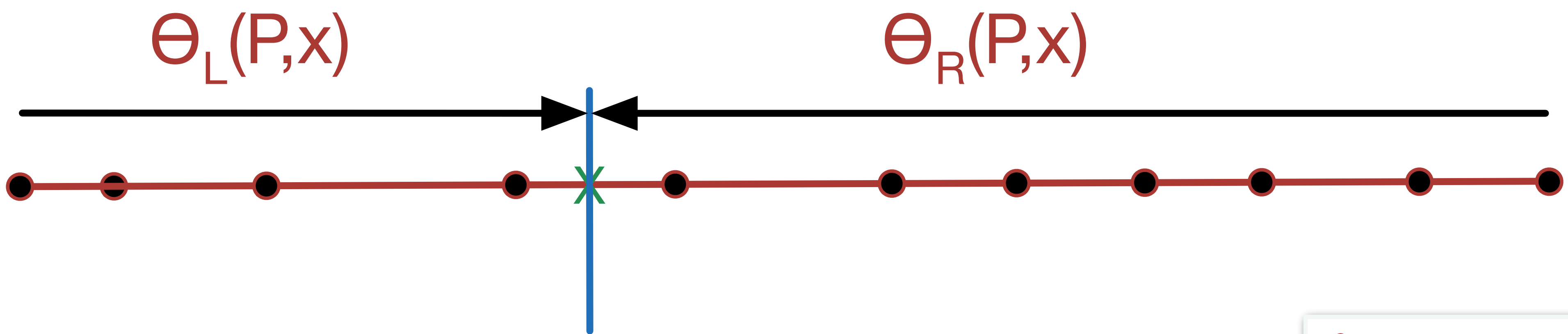
switches to

$\Theta_L(P,x_i) > \Theta_R(P,x_i)$.

Optimum sink x is in the interval where the switch occurs

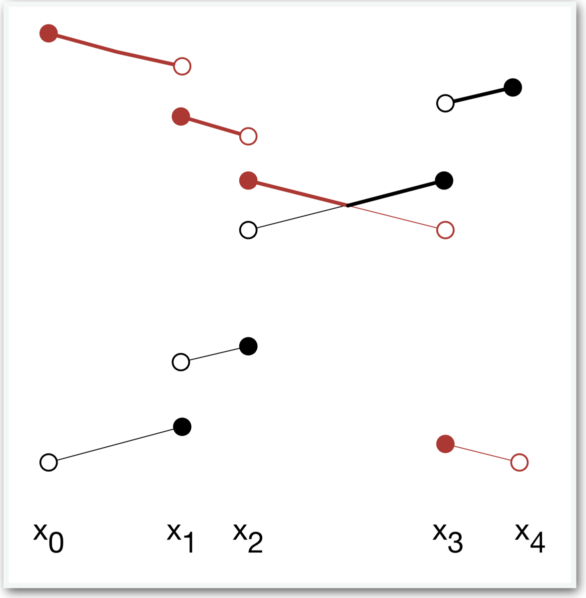


An O(|P| log|P|) Algorithm for $\Theta^1(P)$

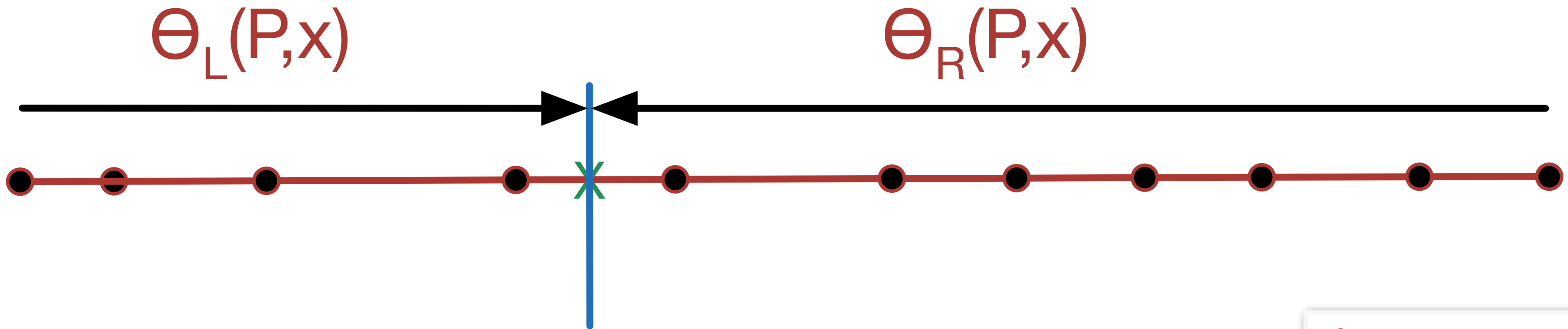


Corollary: For fixed x, $\Theta_L(P,x)$, $\Theta_R(P,x)$ can be computed in O(|P|) time

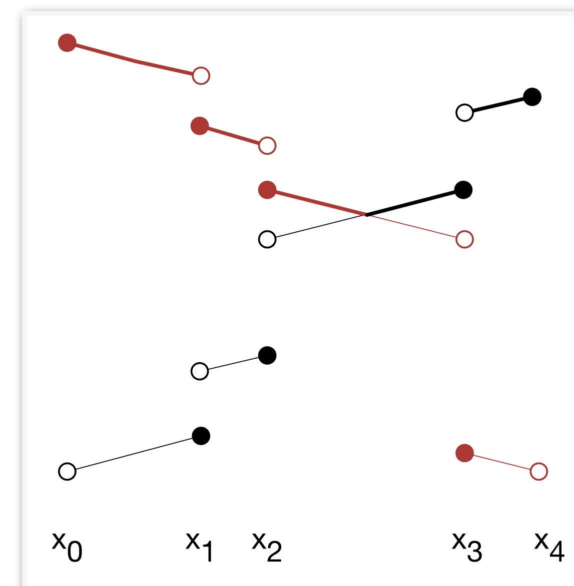
Claim 2: $\Theta(P,x) = \max(\Theta_L(P,x), \Theta_R(P,x))$ is a unimodal function.



An $O(|P| \log|P|)$ Algorithm for $\Theta^1(P)$



Corollary: For fixed x, $\Theta_L(P,x)$, $\Theta_R(P,x)$ can be computed in $O(|P|)$ time



Claim 2: $\Theta(P,x) = \max(\Theta_L(P,x), \Theta_R(P,x))$ is a unimodal function.

Algorithm: Using $O(|P| \log|P|)$ time binary search

Find x_t s.t $\Theta^1(P) = \Theta(P,x)$ satisfying $x_t < x \leq x_{t+1}$.

Gives $\Theta_L(P, x_t)$, $\Theta_R(P, x_t)$, $\Theta_L(P, x_{t+1})$, $\Theta_R(P, x_{t+1})$

In $O(1)$ time do a linear interpolation to find x.

Algorithm Development Sketch

1. Formulae for $\Theta_L(P,x)$ and $\Theta_L(P,x)$
2. => $O(|P|)$ Algorithm for $\Theta_L(P,x)$, $\Theta_L(P,x)$
3. => $O(|P| \log |P|)$ Algorithm for $\Theta^1(P)$
4. => $O(|P| \log |P|)$ Algorithm that $\forall \alpha > 0$ tests whether $\Theta^k(P) \leq \alpha$
5. => $O(k|P| \log^2 |P|)$ Algorithm for $\Theta^k(P)$

An O(|P| log|P|) Testing Algorithm for $\Theta^k(P)$ [1]

Set $P_{i,j}$ to be path from x_i to x_j and $P_{i,x}$ path from x_i to x.
Set |P| to be # of vertices in P.

An O(|P| log|P|) Testing Algorithm for $\Theta^k(P)$ [1]

Set $P_{i,j}$ to be path from x_i to x_j and $P_{i,x}$ path from x_i to x.
Set |P| to be # of vertices in P.

Thm: $\forall \alpha > 0$, k>0 and i,j can test **if $\Theta^k(P_{i,n}) \leq \alpha$**

in ***$O(|P_{i,n}| \log |P_{i,n}|)$*** time

<u>An O(|P| log|P|) Testing Algorithm for $\Theta^k(P)$ [1]</u>

Set $P_{i,j}$ to be path from x_i to x_j and $P_{i,x}$ path from x_i to x.
Set |P| to be # of vertices in P.

<u>Thm:</u> $\forall \alpha > 0$, $k>0$ and i,j can test **if $\Theta^k(P_{i,n}) \leq \alpha$**
in ***$O(|P_{i,n}| \log |P_{i,n}|)$*** time

<u>Lemma:</u> $\forall \alpha > 0$, and i can find **maximum j s.t. $\Theta^1(P_{i,j}) \leq \alpha$**
in ***$O(|P_{i,j}| \log |P_{i,j}|)$*** time

An $O(|P| \log|P|)$ Testing Algorithm for $\Theta^k(P)$ [1]

Set $P_{i,j}$ to be path from x_i to x_j and $P_{i,x}$ path from x_i to x.
Set $|P|$ to be # of vertices in P.

<u>Thm:</u> $\forall \alpha > 0$, $k>0$ and i,j can test **if $\Theta^k(P_{i,n}) \leq \alpha$**
in ***$O(\,|P_{i,n}| \;\log |P_{i,n}|\,)$*** time

<u>Lemma:</u> $\forall \alpha > 0$, and i can find **maximum j s.t. $\Theta^1(P_{i,j}) \leq \alpha$**
in ***$O(\,|P_{i,j}| \;\log |P_{i,j}|\,)$*** time

<u>Proof Idea (Lemma):</u>
In ***$O(|P_{i,x}| \;\log |P_{i,x}|)$*** use linear formula for $\Theta_L(P_{i,n},x)$ &
doubling search technique to find max x s.t. **$\Theta_L(P_{i,n},x) \leq \alpha$**.

An O(|P| log|P|) Testing Algorithm for $\Theta^k(P)$ [2]

Set $P_{i,j}$ to be path from x_i to x_j and $P_{i,x}$ path from x_i to x.
Set |P| to be # of vertices in P.

Thm: $\forall \alpha > 0$, $k>0$ and i,j can test **if $\Theta^k(P_{i,n}) \le \alpha$**
in ***$O(|P_{i,n}| \log |P_{i,n}|)$*** time

Lemma: $\forall \alpha > 0$, and i can find **maximum j s.t. $\Theta^1(P_{i,j}) \le \alpha$**
in ***$O(|P_{i,j}| \log |P_{i,j}|)$*** time

Proof Idea (Lemma):
In ***$O(|P_{i,x}| \log |P_{i,x}|)$*** use linear formula for $\Theta_L(P_{i,n},x)$ &
doubling search technique to find max x s.t. **$\Theta_L(P_{i,n},x) \le \alpha$**.

Similarly, in ***$O(|P_{x,j}| \log |P_{x,j}|)$***, find max j s.t. **$\Theta_R(P_{i,n},x) \le \alpha$**

An $O(|P| \log|P|)$ Testing Algorithm for $\Theta^k(P)$ [3]

Set $P_{i,j}$ to be path from x_i to x_j and $P_{i,x}$ path from x_i to x.
Set $|P|$ to be # of vertices in P.

Thm: $\forall\alpha > 0$, $k>0$ and i,j can test **if $\Theta^k(P_{i,n}) \leq \alpha$** in **$O(|P_{i,n}| \log |P_{i,n}|)$** time

Lemma: $\forall\alpha > 0$, and i can find **maximum j s.t. $\Theta^1(P_{i,j}) \leq \alpha$** in **$O(|P_{i,j}| \log |P_{i,j}|)$** time

An $O(|P| \log|P|)$ Testing Algorithm for $\Theta^k(P)$ [3]

Set $P_{i,j}$ to be path from x_i to x_j and $P_{i,x}$ path from x_i to x.
Set $|P|$ to be # of vertices in P.

Thm: $\forall \alpha > 0$, $k>0$ and i,j can test **if $\Theta^k(P_{i,n}) \le \alpha$** in **$O(|P_{i,n}| \log |P_{i,n}|)$** time

Lemma: $\forall \alpha > 0$, and i can find **maximum j s.t. $\Theta^1(P_{i,j}) \le \alpha$** in **$O(|P_{i,j}| \log |P_{i,j}|)$** time

---

Proof Sketch (Thm): Use Lemma to peel off, from left side of $P_{i,j}$, k max-length subpaths that can each be 1-evacuated in α time. If this covers all $P_{i,j}$, then YES. Otherwise NO.

An O(|P| log|P|) Testing Algorithm for $\Theta^k(P)$ [3]

Set $P_{i,j}$ to be path from x_i to x_j and $P_{i,x}$ path from x_i to x.
Set |P| to be # of vertices in P.

Thm: $\forall \alpha > 0$, k>0 and i,j can test **if $\Theta^k(P_{i,n}) \le \alpha$** in **$O(|P_{i,n}| \log |P_{i,n}|)$** time

Lemma: $\forall \alpha > 0$, and i can find **maximum j s.t. $\Theta^1(P_{i,j}) \le \alpha$** in **$O(|P_{i,j}| \log |P_{i,j}|)$** time

---

Proof Sketch (Thm): Use Lemma to peel off, from left side of $P_{i,j}$, k max-length subpaths that can each be 1-evacuated in α time. If this covers all $P_{i,j}$, then YES. Otherwise NO.

An O(|P| log|P|) Testing Algorithm for $\Theta^k(P)$ [4]

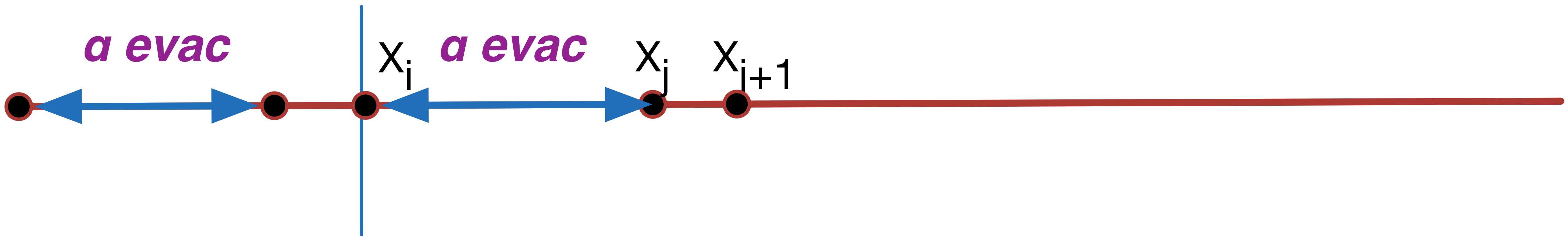
Set $P_{i,j}$ to be path from x_i to x_j and $P_{i,x}$ path from x_i to x.
Set |P| to be # of vertices in P.

Thm: $\forall \alpha > 0$, k>0 and i,j can test **if $\Theta^k(P_{i,n}) \le \alpha$** in **$O(|P_{i,n}| \log |P_{i,n}|)$** time

Lemma: $\forall \alpha > 0$, and i can find **maximum j s.t. $\Theta^1(P_{i,j}) \le \alpha$** in **$O(|P_{i,j}| \log |P_{i,j}|)$** time

---

Proof Sketch (Thm): Use Lemma to peel off, from left side of $P_{i,j}$, k max-length subpaths that can each be 1-evacuated in α time. If this covers all $P_{i,j}$, then YES. Otherwise NO.



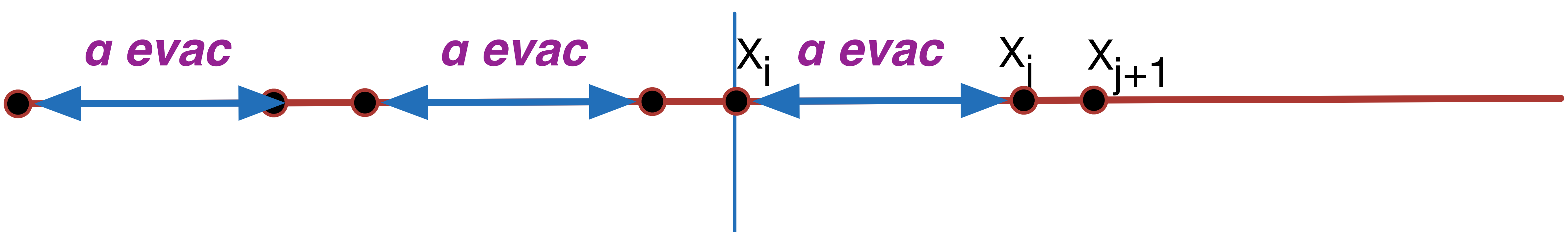
An $O(|P| \log|P|)$ Testing Algorithm for $\Theta^k(P)$ [5]

Set $P_{i,j}$ to be path from x_i to x_j and $P_{i,x}$ path from x_i to x.
Set $|P|$ to be # of vertices in P.

Thm: $\forall \alpha > 0$, $k>0$ and i,j can test **if $\Theta^k(P_{i,n}) \leq \alpha$**
in **$O(|P_{i,n}| \log |P_{i,n}|)$** time

Lemma: $\forall \alpha > 0$, and i can find **maximum j s.t. $\Theta^1(P_{i,j}) \leq \alpha$**
in **$O(|P_{i,j}| \log |P_{i,j}|)$** time

Proof Sketch (Thm): Use Lemma to peel off, from left side of $P_{i,j}$, k max-length subpaths that can each be 1-evacuated in α time. If this covers all $P_{i,j}$, then YES. Otherwise NO.



Algorithm Development Sketch

1. Formulae for $\Theta_L(P,x)$ and $\Theta_L(P,x)$

2. => $O(|P|)$ Algorithm for $\Theta_L(P,x)$, $\Theta_L(P,x)$

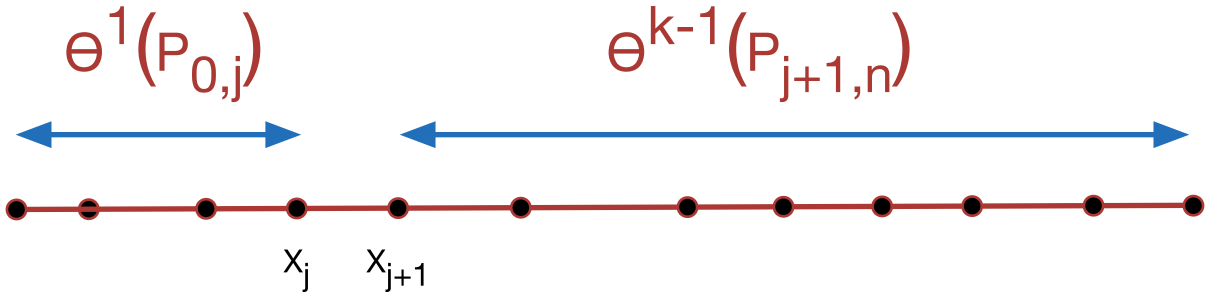
3. => $O(|P| \log |P|)$ Algorithm for $\Theta^1(P)$

4. => $O(|P| \log |P|)$ Algorithm that $\forall \alpha > 0$
   tests whether $\Theta^k(P) \leq \alpha$

5. => $O(k|P| \log^2 |P|)$ Algorithm for $\Theta^k(P)$

An O(k |P| log² |P|) Algorithm for Θᵏ(P) [1]

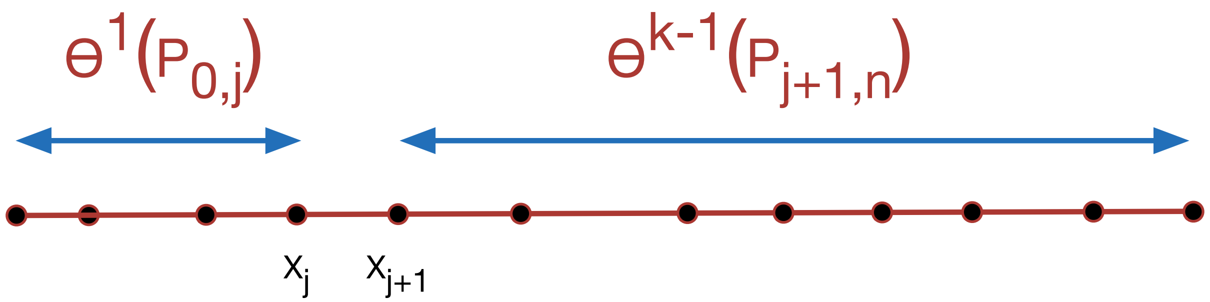
- $\Theta^k(P) = \Theta_j^k = \min_j (\max(\Theta^1(P_{0,j}), \Theta^{k-1}(P_{j+1,n})))$



- $\Theta^1(P_{0,j})$ ($\Theta^{k-1}(P_{j+1,n})$) is non decreasing (increasing) in j
- Θ_j^k = is "unimodal" in j

An $O(k\,|P|\,\log^2|P|)$ Algorithm for $\Theta^k(P)$ [1]

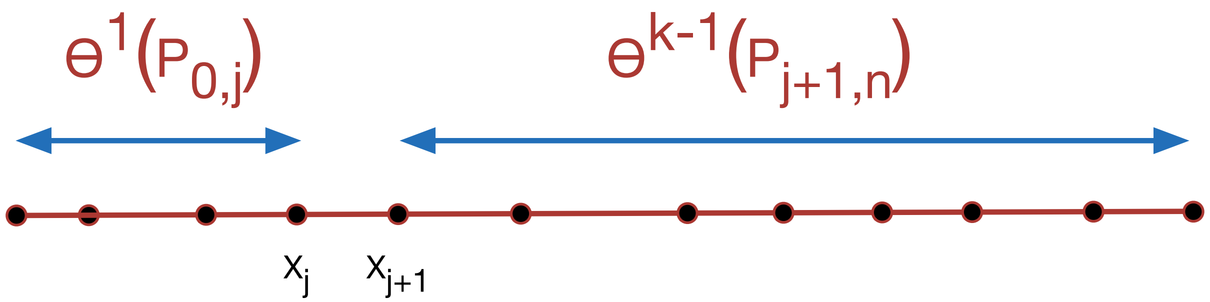
- $\Theta^k(P) = \Theta_j^k = \min_j\left(\max(\Theta^1(P_{0,j}), \Theta^{k-1}(P_{j+1,n})) \right)$



- $\Theta^1(P_{0,j})$ ($\Theta^{k-1}(P_{j+1,n})$) is non decreasing (increasing) in j
- Θ_j^k = is "unimodal" in j
- $\Theta^{k-1}(P_{j+1,n}) \leq \Theta^1(P_{0,j})$ can be tested in ***O(|P| log |P|)*** time

An $O(k\,|P|\,\log^2|P|)$ Algorithm for $\Theta^k(P)$ [1]

- $\Theta^k(P) = \Theta_j^k = \min_j (\ \max(\Theta^1(P_{0,j}),\ \Theta^{k-1}(P_{j+1,n}))\ )$



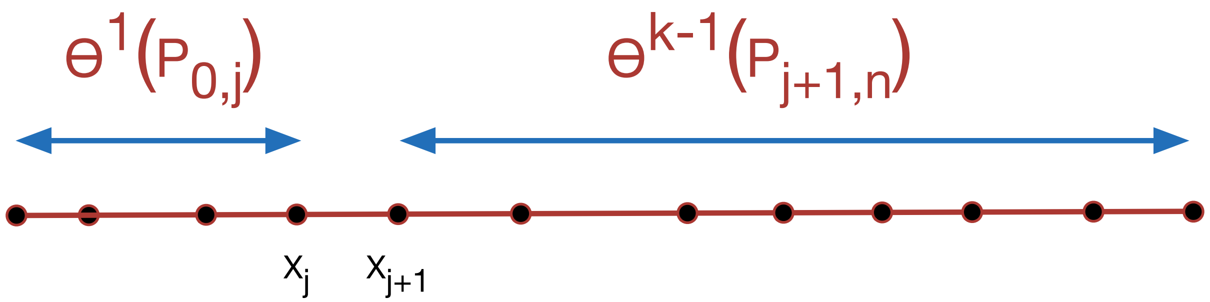
- $\Theta^1(P_{0,j})$ ($\Theta^{k-1}(P_{j+1,n})$) is non decreasing (increasing) in j

- Θ_j^k = is "unimodal" in j

- $\Theta^{k-1}(P_{j+1,n}) \leq \Theta^1(P_{0,j})$ can be tested in ***O(|P| log |P|)*** time
  - Using previous algorithms for k=1 and testing

An $O(k\,|P|\,\log^2|P|)$ Algorithm for $\Theta^k(P)$ [1]

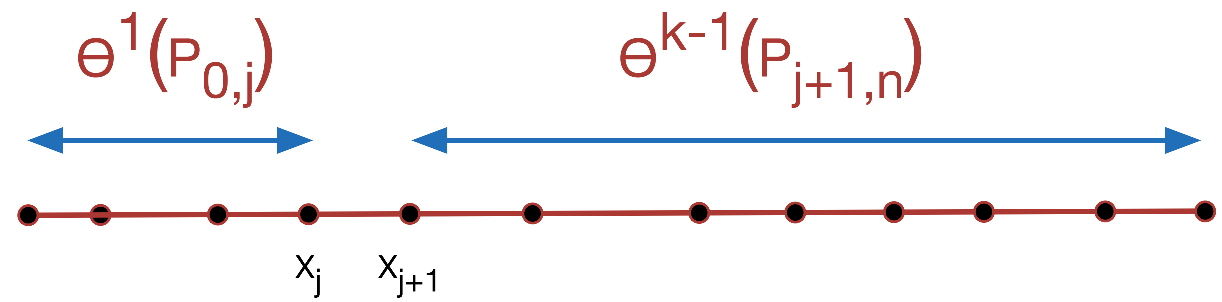
- $\Theta^k(P) = \Theta_j^k = \min_j \left(\max(\Theta^1(P_{0,j}), \Theta^{k-1}(P_{j+1,n})) \right)$



- $\Theta^1(P_{0,j})$ ($\Theta^{k-1}(P_{j+1,n})$) is non decreasing (increasing) in j
- Θ_j^k = is "unimodal" in j
- $\Theta^{k-1}(P_{j+1,n}) \leq \Theta^1(P_{0,j})$ can be tested in ***O(|P| log |P|)*** time
  - Using previous algorithms for k=1 and testing
  - $O(|P_{0,j}| \log |P_{0,j}|) + O(|P_{j+1,n}| \log |P_{j+1,n}|) = O(|P| \log |P|)$

An $O(k\,|P|\,\log^2|P|)$ Algorithm for $\Theta^k(P)$ [1]

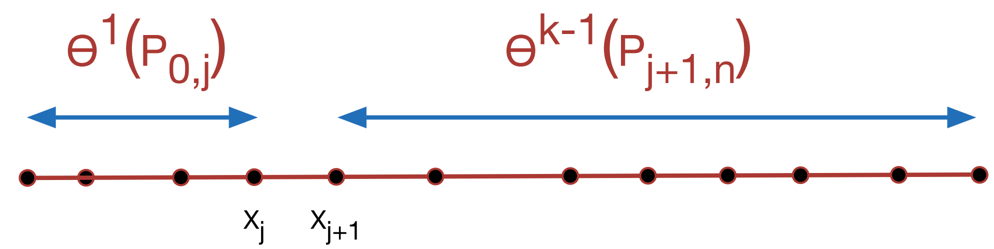
- $\Theta^k(P) = \Theta_j^k = \min_j \left(\max(\Theta^1(P_{0,j}), \Theta^{k-1}(P_{j+1,n})) \right)$



- $\Theta^1(P_{0,j})$ ($\Theta^{k-1}(P_{j+1,n})$) is non decreasing (increasing) in j
- Θ_j^k = is "unimodal" in j
- $\Theta^{k-1}(P_{j+1,n}) \leq \Theta^1(P_{0,j})$ can be tested in ***$O(|P| \log |P|)$*** time
  - Using previous algorithms for k=1 and testing
  - $O(|P_{0,j}| \log |P_{0,j}|) + O(|P_{j+1,n}| \log |P_{j+1,n}|) = O(|P| \log |P|)$
- Binary search to find largest j s.t. $\Theta^{k-1}(P_{j+1,n}) > \Theta^1(P_{0,j})$
  - ***$O(|P| \log^2|P|)$ time***

An $O(k\,|P|\,\log^2|P|)$ Algorithm for $\Theta^k(P)$ [2]

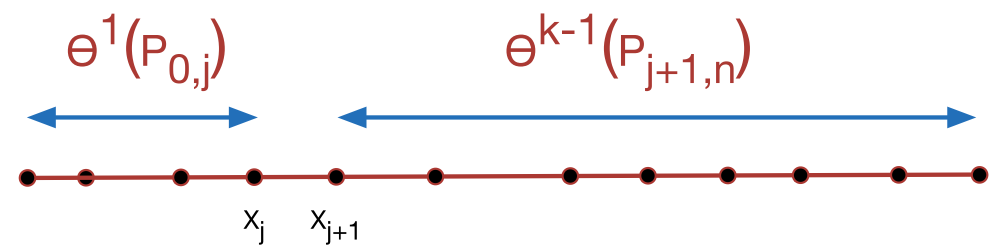
- $\Theta^k(P) = \Theta^k_j = \min_j (\Theta^1(P_{0,j}), \Theta^{k-1}(P_{j+1,n}))$
  $\Theta^1(P_{0,j})$, $\Theta^{k-1}(P_{j+1,n})$ increase/decrease in j
  Θ^k_j "unimodal" in j



- => **$O(|P|\log^2|P|)$** time Binary search
  to find largest j s.t $\Theta^{k-1}(P_{j+1,n}) > \Theta^1(P_{0,j})$

An $O(k\,|P|\,\log^2|P|)$ Algorithm for $\Theta^k(P)$ [2]

- $\Theta^k(P) = \Theta^k_j = \min_j (\Theta^1(P_{0,j}), \Theta^{k-1}(P_{j+1,n}))$
  $\Theta^1(P_{0,j}), \Theta^{k-1}(P_{j+1,n})$ increase/decrease in j
  Θ^k_j "unimodal" in j

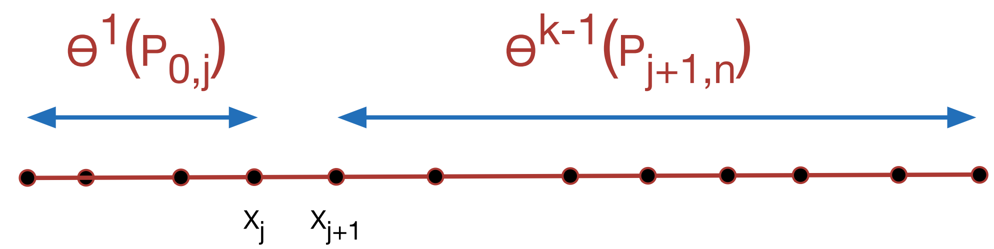


- => **$O(|P|\log^2|P|)$** time Binary search
  to find largest j s.t $\Theta^{k-1}(P_{j+1,n}) > \Theta^1(P_{0,j})$

- **$\Theta^k(P)$ is min of $\Theta^1(P_{0,j+1})$ and $\Theta^{k-1}(P_{j+1,n})$**

An $O(k\,|P|\,\log^2|P|)$ Algorithm for $\Theta^k(P)$ [2]

- $\Theta^k(P) = \Theta^k_j = \min_j\,(\Theta^1(P_{0,j}),\ \Theta^{k-1}(P_{j+1,n}))$
  $\Theta^1(P_{0,j})$, $\Theta^{k-1}(P_{j+1,n})$ increase/decrease in j
  Θ^k_j "unimodal" in j



- => **$O(|P|\log^2|P|)$** time Binary search
  to find largest j s.t $\Theta^{k-1}(P_{j+1,n}) > \Theta^1(P_{0,j})$

- **$\Theta^k(P)$ is min of $\Theta^1(P_{0,j+1})$ and $\Theta^{k-1}(P_{j+1,n})$**

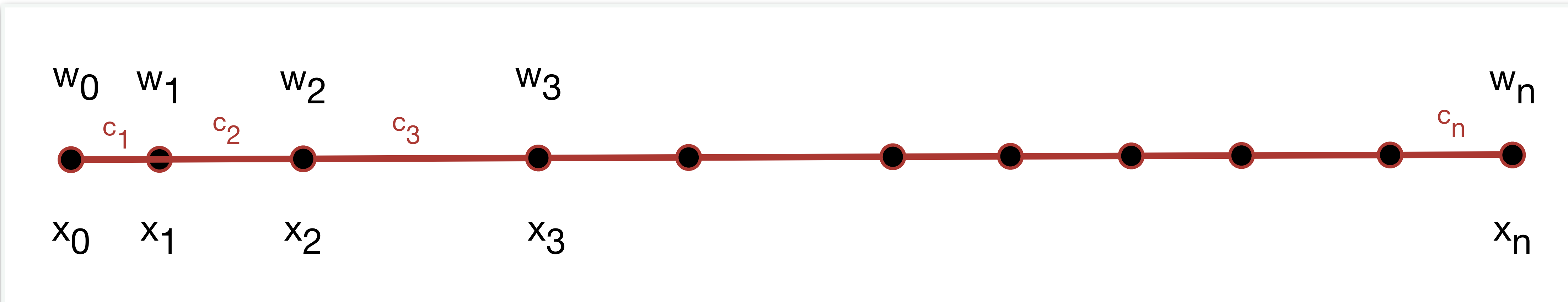
- $\Theta^{k-1}(P_{j+1,n})$ can be found recursively
  - stop when k=1 (know how to solve)
  - Total algorithm is **$k\,O(|P|\log^2|P|) = O(k\,|P|\log^2|P|)$**

Outline

- Dynamic Flow Networks
- Congestion in Dynamic Flows
- Evacuation Flows
  - Problem Definitions
  - Known Results
- Example Algorithm 1: k-Sink Evacuation on a Path
- Example Algorithm 2: 1-sink Min-Max Regret Evacuation on a Path with uniform capacity
- Open Problems

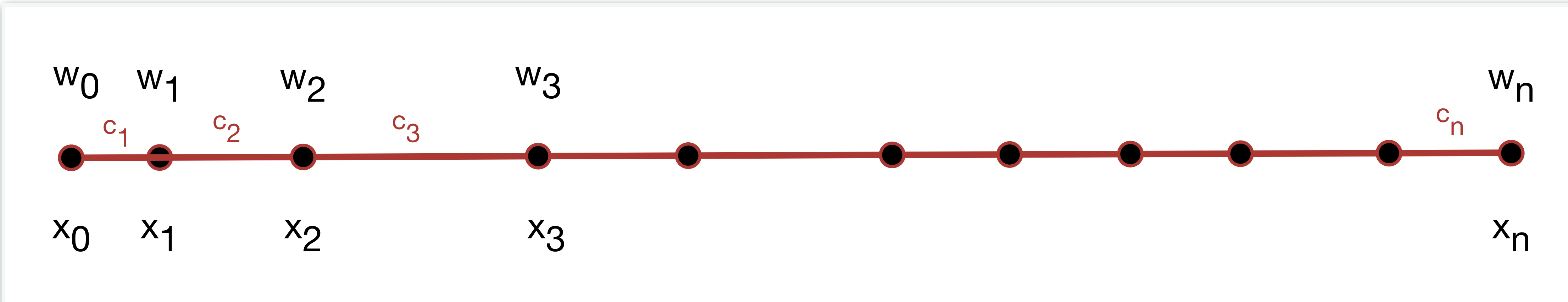
Min-Max Regret Evacuation on a Path

In the regret version of the problem, input still provides c_e, τ_e, k

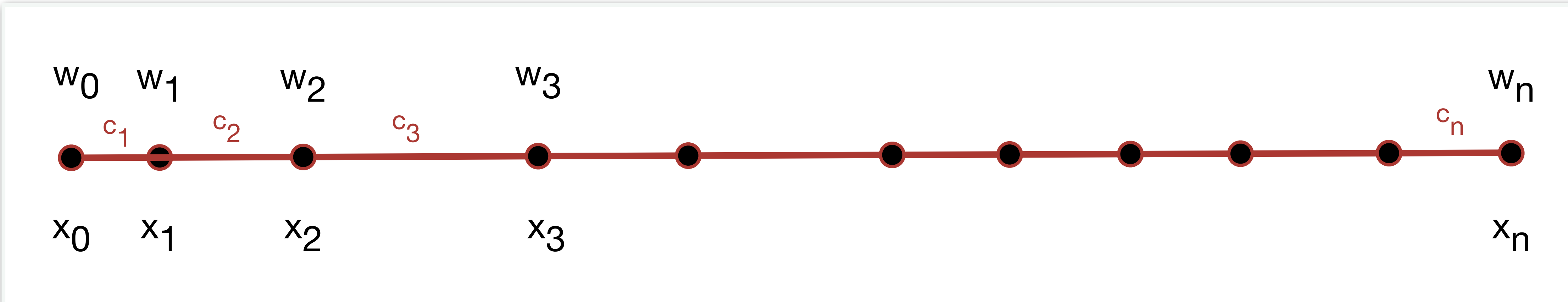


- But w_v is no longer explicitly input.
  Instead, each vertex has an input range $w_v \in [w'_v, W'_v]$
- Algorithm needs to find **robust** evacuation protocol that works least badly against adversarial input.
- Min-Max Regret is one standard way of modelling robustness

Min-Max Regret Evacuation on a Path

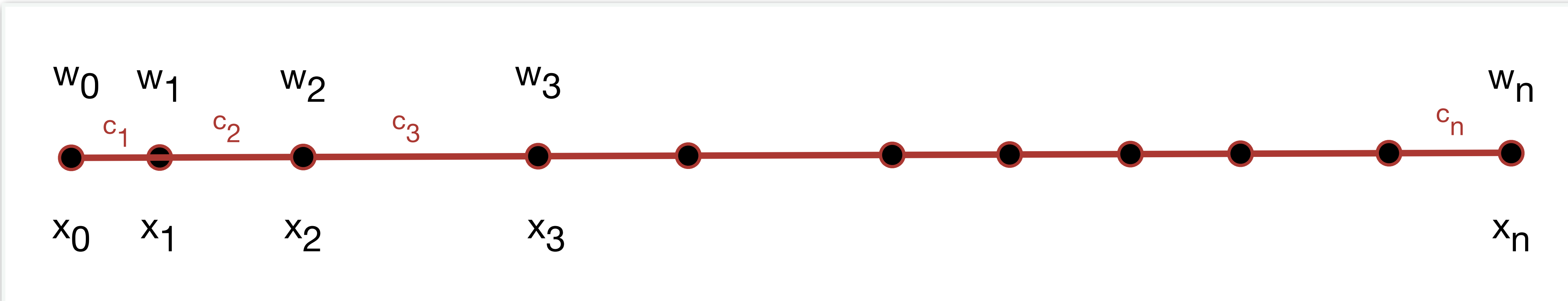


Min-Max Regret Evacuation on a Path



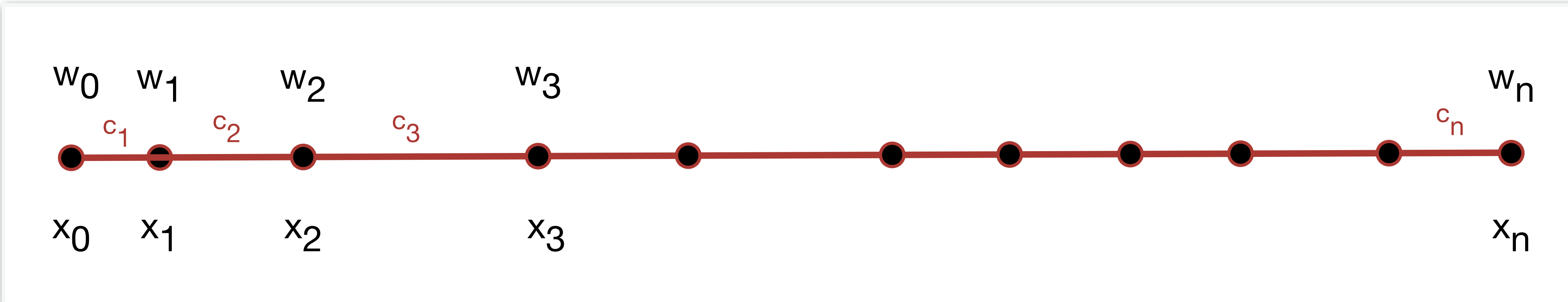
- $S = \prod_v [w'_v, W'_v]$ is the set of all feasible scenarios.
  An $s \in S$ is of the form $s = (w_1, \ldots, w_n)$

Min-Max Regret Evacuation on a Path



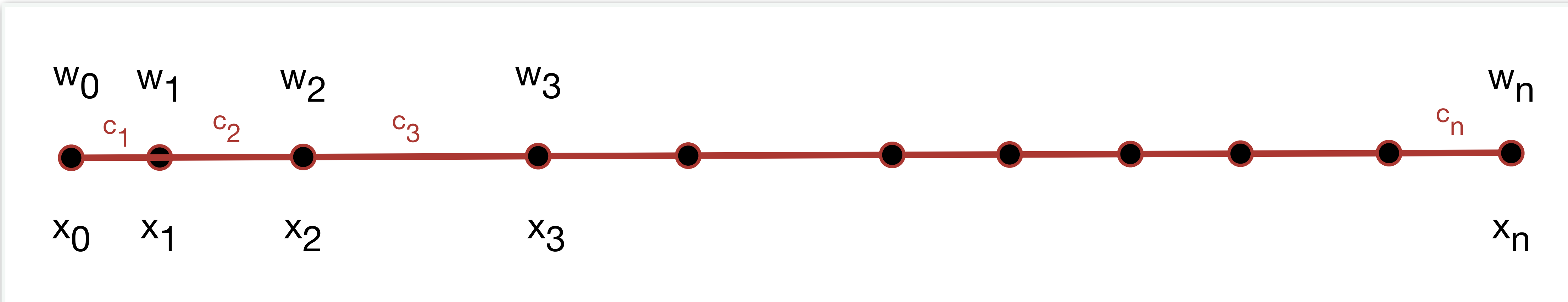
- $S = \prod_v [w'_v, W'_v]$ is the set of all feasible scenarios.
  An $s \in S$ is of the form $s = (w_1, \ldots, w_n)$
- $\Theta(P,x,s)$ = evacuation time of P to x in scenario s
- $\Theta^1(P,s)$ = min evacuation time of P in scenario s

Min-Max Regret Evacuation on a Path



- $S = \prod_v [w'_v, W'_v]$ is the set of all feasible scenarios.
  An $s \in S$ is of the form $s = (w_1, \ldots, w_n)$
- $\Theta(P,x,s)$ = evacuation time of P to x in scenario s
- $\Theta^1(P,s)$ = min evacuation time of P in scenario s
- **$R(x,s)$** = Regret of x under scenario s = **$\Theta(P,x,s) - \Theta^1(P,s)$**
- $R(x)$ = Max regret of x = $\max_s R(x,s)$

Min-Max Regret Evacuation on a Path



- $S = \prod_v [w'_v, W'_v]$ is the set of all feasible scenarios.
  An $s \in S$ is of the form $s = (w_1, \ldots, w_n)$

- $\Theta(P,x,s)$ = evacuation time of P to x in scenario s

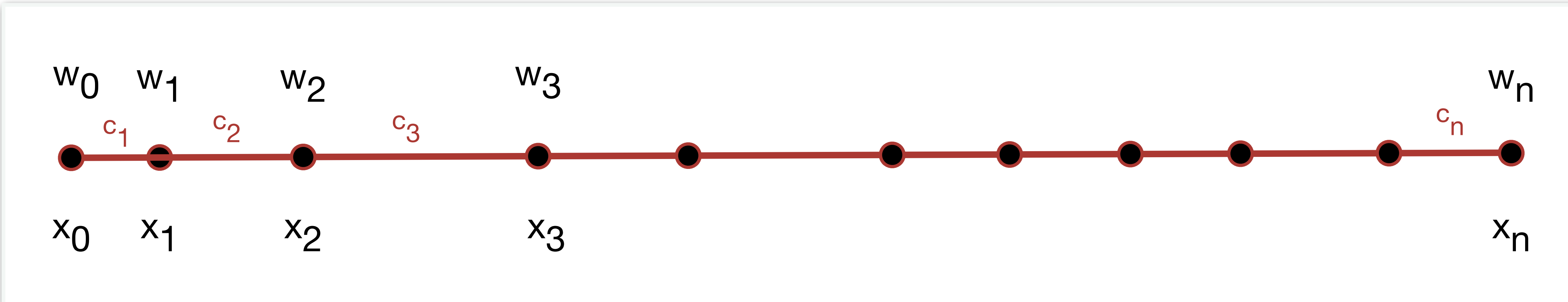
- $\Theta^1(P,s)$ = min evacuation time of P in scenario s

- **$R(x,s)$** = Regret of x under scenario s = **$\Theta(P,x,s) - \Theta^1(P,s)$**

- $R(x)$ = Max regret of x = $\max_s R(x,s)$

- The Min-max regret of P is minimum regret over all x
  $MMR(P) = \min_x R(x)$

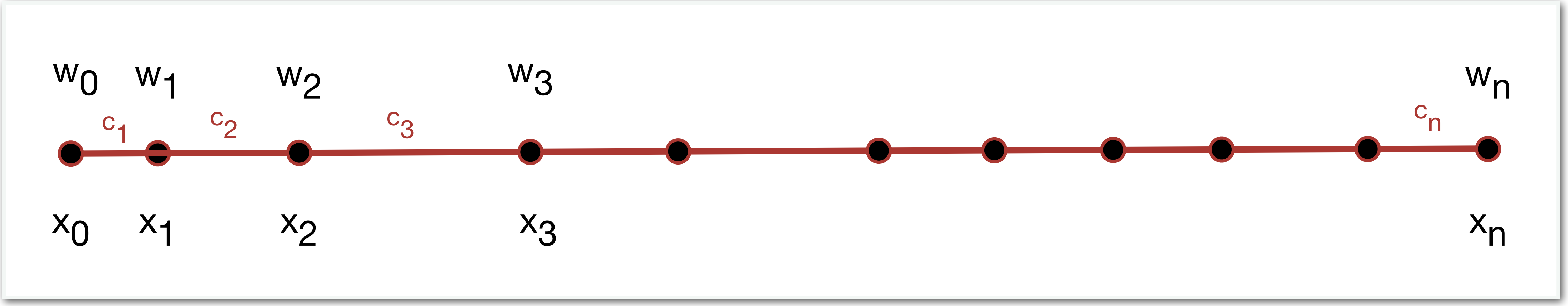
Min-Max Regret Evacuation on a Path



$R(x,s) = \Theta(P,x,s) - \Theta^1(P,s)$ $\qquad R(x) = \mathrm{Max}_s\, R(x,s)$

$MMR(P) = \mathrm{Min}_x\, R(x) = \mathrm{Min}_x\, \mathrm{Max}_s\, \{\Theta(P,x,s) - \Theta^1(P,s)\}$

Min-Max Regret Evacuation on a Path

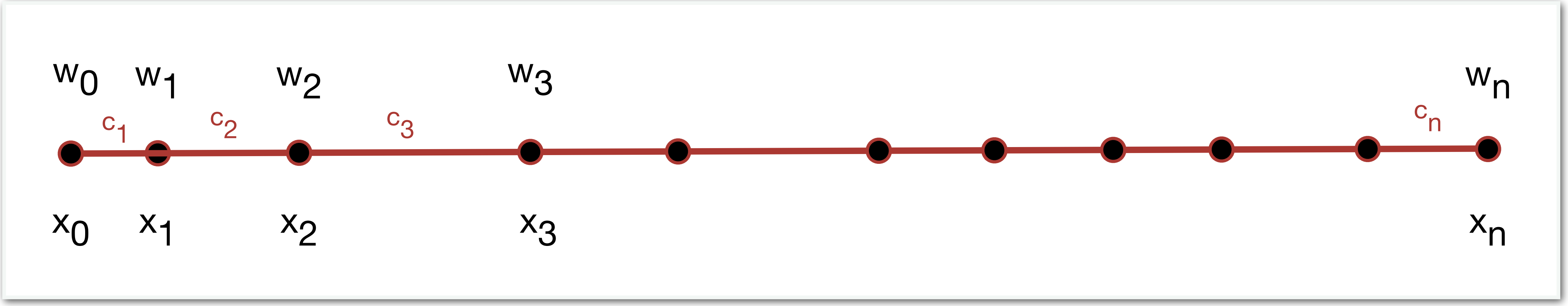


$R(x,s) = \Theta(P,x,s) - \Theta^1(P,s) \qquad R(x) = \mathrm{Max}_s\, R(x,s)$

$\mathrm{MMR}(P) = \mathrm{Min}_x\, R(x) = \mathrm{Min}_x\ \mathrm{Max}_s\, \{\Theta(P,x,s) - \Theta^1(P,s)\}$

- A-Priori, it isn't obvious that this can be calculated efficiently.

Min-Max Regret Evacuation on a Path

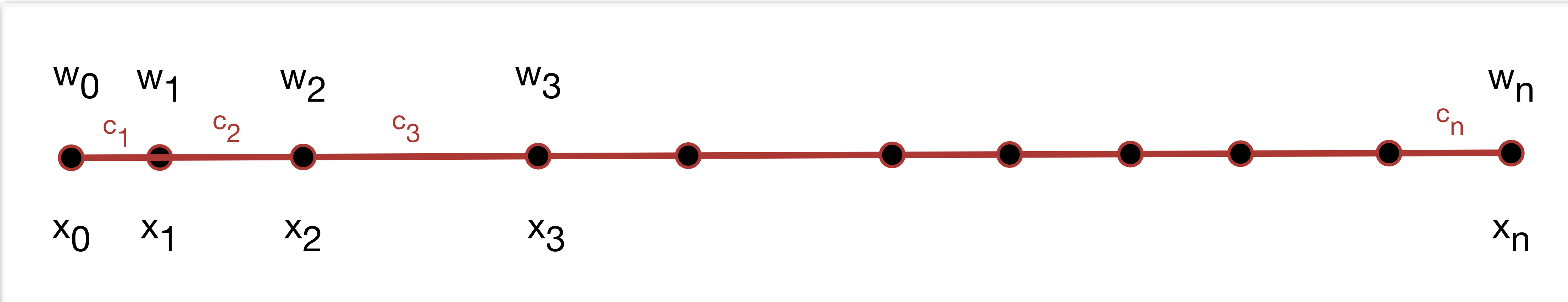


$R(x,s) = \Theta(P,x,s) - \Theta^1(P,s)$ $\quad$ $R(x) = \mathrm{Max}_s\, R(x,s)$

$MMR(P) = \mathrm{Min}_x\, R(x) = \mathrm{Min}_x\, \mathrm{Max}_s \{\Theta(P,x,s) - \Theta^1(P,s)\}$

- A-Priori, it isn't obvious that this can be calculated efficiently.
- Can show that, for uniform capacities, there are only O(n) scenarios *s* at which any R(x) attains maximum

Min-Max Regret Evacuation on a Path

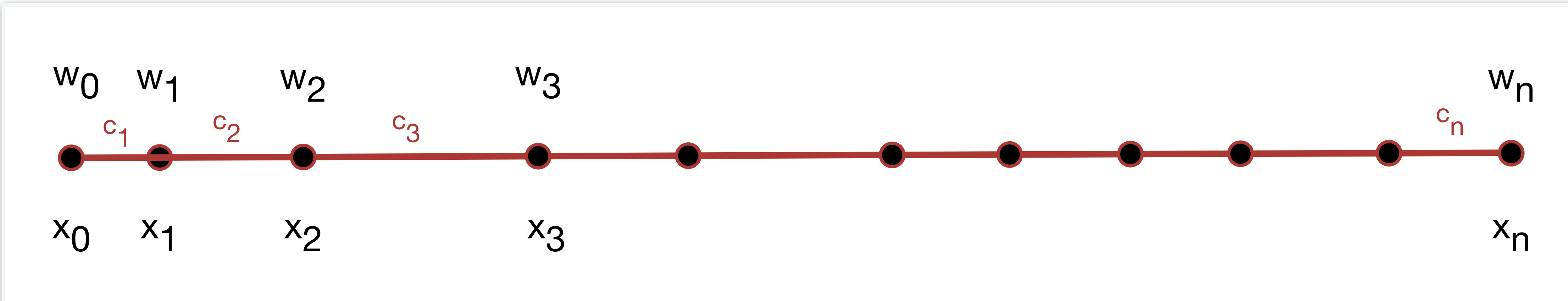


$R(x,s) = \Theta(P,x,s) - \Theta^1(P,s)$ $\qquad$ $R(x) = \mathrm{Max}_s\, R(x,s)$

$MMR(P) = \mathrm{Min}_x\, R(x) = \mathrm{Min}_x\, \mathrm{Max}_s \{\Theta(P,x,s) - \Theta^1(P,s)\}$

- A-Priori, it isn't obvious that this can be calculated efficiently.
- Can show that, for uniform capacities, there are only O(n) scenarios *s* at which any R(x) attains maximum
- This, permits evaluating MMR(P) in polynomial time
  - further observations reduce this to O(n log n)

Min-Max Regret Evacuation on a Path



$R(x,s) = \Theta(P,x,s) - \Theta^1(P,s)$ $\quad$ $R(x) = \mathrm{Max}_s\, R(x,s)$

$\mathrm{MMR}(P) = \mathrm{Min}_x\, R(x) = \mathrm{Min}_x\, \mathrm{Max}_s\, \{\Theta(P,x,s) - \Theta^1(P,s)\}$

- A-Priori, it isn't obvious that this can be calculated efficiently.
- Can show that, for uniform capacities, there are only O(n) scenarios *s* at which any R(x) attains maximum
- This, permits evaluating MMR(P) in polynomial time
  - further observations reduce this to O(n log n)
- Existence of O(n) scenarios not totally surprising
  - Same phenomenon arises in MMR for medians on a line

Min-Max Regret Evacuation on a Path

$R(x,s) = \Theta(P,x,s) - \Theta^1(P,s)$ $\qquad$ $R(x) = \text{Max}_s\, R(x,s)$

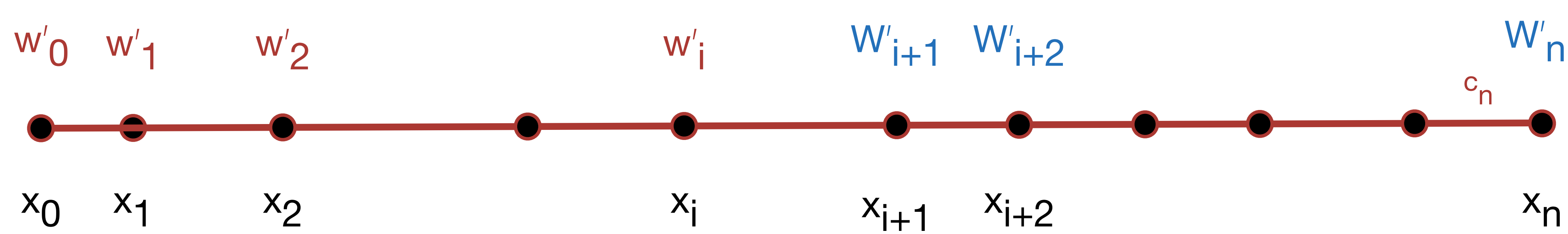
$\text{MMR}(P) = \text{Min}_x\, R(x) = \text{Min}_x\ \text{Max}_s\ \{\Theta(P,x,s) - \Theta^1(P,s)\}$

There are 2n scenarios at which $R(s,x)$ attains max.

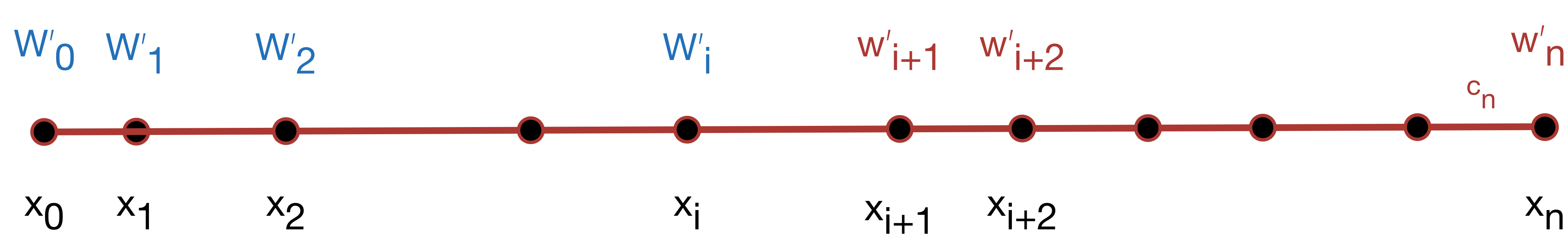
These are s_i in which $w_j = w'_j$ for $j \le i$ & $w_j = W'_j$ for $i > j$

and s'_i in which $w_j = W'_j$ for $j \le i$ & $w'_j = w'_j$ for $i > j$

S_i



S'_i



Min-Max Regret Evacuation on a Path

$R(s,x) = \Theta(P,x,s) - \Theta^1(P,s)$ $\quad$ $R(x) = \mathrm{Max}_s\, R(s,x)$

$MMR(P) = \mathrm{Min}_x\, R(x) = \mathrm{Min}_x\, \mathrm{Max}_s\, \{\Theta(P,x,s) - \Theta^1(P,s)\}$

There are 2n scenarios at which R(s,x) attains max.
These are s_i in which $w_j = w'_j$ for $j < i$ & $w_j = W'_j$ for $i > j$
and s'_i in which $w_j = W'_j$ for $j < i$ & $w'_j = w'_j$ for $i > j$

- k-sink uniform capacity on path have $O(n^3)$ worst case scenarios => $O(kn^3 \log n)$ time time algorithm
- 1-sink uniform capacity on tree have $O(n^2)$ worst case MMR scenarios => $O(n^2 \log^2 n)$ time algorithm
- NOTHING is known about any other cases.
  In particular, even on path no structure for MMR solution for 1-sink gen cap problem => no polynomial time alg

Outline

- Dynamic Flow Networks
- Congestion in Dynamic Flows
- Evacuation Flows
  - Problem Definitions
  - Known Results
- Example Algorithm 1: k-Sink Evacuation on a Path
- Example Algorithm 2: 1-sink Min-Max Regret Evacuation on a Path with uniform capacity
- Open Problems

Open Frontier Problems

- G a General Graph, $k>1$ (NP Hard)
  - Find approximation algorithm or PTAS
- ~~G a General Graph, $k=1$~~
  - ~~Solve exactly or prove NP-Hard~~
  - ~~Even if the one sink is given~~
- G a tree with uniform capacities, $k>1$
  - solve min-max regret k-sink problem
- G a path (tree) tree with general capacities, $k=1$
  - solve min-max regret 1-sink problem
- For Robust Computation
  - Replace Min-Max-Regret by size distribution on nodes and find sink(s) that minimize expected evacuation time.