ACDA 2025 Mini-tutorial on Learning-Augmented Algorithms
Part 2

Algorithmic Ideas from Learning-Augmented Algorithms

Vaidehi Srinivas
Northwestern University

Recap

Recap

Learning-Augmented Algorithms (a.k.a. Algorithms with Predictions):

Take advantage of **unreliable predictions**, without sacrificing worst-case guarantees

Recap

Learning-Augmented Algorithms (a.k.a. Algorithms with Predictions):

Take advantage of **unreliable predictions**, without sacrificing worst-case guarantees

Consistency
If prediction is good, take advantage!

Recap

Learning-Augmented Algorithms (a.k.a. Algorithms with Predictions):

Take advantage of **unreliable predictions**, without sacrificing worst-case guarantees

Consistency
If prediction is good, take advantage!

Robustness
If prediction is bad, revert to worst-case guarantee

Recap

Learning-Augmented Algorithms (a.k.a. Algorithms with Predictions):

Take advantage of **unreliable predictions**, without sacrificing worst-case guarantees

Consistency
If prediction is good, take advantage!

Graceful Degradation
in magnitude of prediction error

Robustness
If prediction is bad, revert to worst-case guarantee

Recap

Learning-Augmented Algorithms (a.k.a. Algorithms with Predictions):

Take advantage of **unreliable predictions**, without sacrificing worst-case guarantees

Consistency
If prediction is good, take advantage!

Graceful Degradation
in magnitude of prediction error

Robustness
If prediction is bad, revert to worst-case guarantee

Ok great, but...

Recap

Learning-Augmented Algorithms (a.k.a. Algorithms with Predictions):

Take advantage of **unreliable predictions**, without sacrificing worst-case guarantees

Consistency
If prediction is good, take advantage!

Graceful Degradation
in magnitude of prediction error

Robustness
If prediction is bad, revert to worst-case guarantee

Ok great, but...

- Can we actually achieve this for interesting problems?

Recap

Learning-Augmented Algorithms (a.k.a. Algorithms with Predictions):

Take advantage of **unreliable predictions**, without sacrificing worst-case guarantees

Consistency
If prediction is good, take advantage!

Graceful Degradation
in magnitude of prediction error

Robustness
If prediction is bad, revert to worst-case guarantee

Ok great, but...

- Can we actually achieve this for interesting problems?
- Does it help develop **new** algorithmic ideas?

Recap

Learning-Augmented Algorithms (a.k.a. Algorithms with Predictions):

Take advantage of **unreliable predictions**, without sacrificing worst-case guarantees

Consistency	**Graceful Degradation**	**Robustness**
If prediction is good, take advantage!	in magnitude of prediction error	If prediction is bad, revert to worst-case guarantee

Ok great, but…

- Can we actually achieve this for interesting problems?
- Does it help develop **new** algorithmic ideas?

Yes and yes!

Part 2

Part 2

How to utilize extra **unreliable** information in algorithm design?

Part 2

How to utilize extra **unreliable** information in algorithm design?

Worst-case analysis
Pessimistically ignore anything unreliable

Part 2

How to utilize extra **unreliable** information in algorithm design?

Worst-case analysis	**Heuristic**
Pessimistically ignore anything unreliable	Optimistically assume information is good, because it often is

Part 2

How to utilize extra **unreliable** information in algorithm design?

Worst-case analysis
Pessimistically ignore anything unreliable

Heuristic
Optimistically assume information is good, because it often is

Learning-Augmented Algorithms
Do both simultaneously!

Part 2

How to utilize extra **unreliable** information in algorithm design?

Worst-case analysis
Pessimistically ignore anything unreliable

Heuristic
Optimistically assume information is good, because it often is

Learning-Augmented Algorithms
Do both simultaneously!

Goal of talk:

Part 2

How to utilize extra **unreliable** information in algorithm design?

Worst-case analysis
Pessimistically ignore anything unreliable

Heuristic
Optimistically assume information is good, because it often is

Learning-Augmented Algorithms
Do both simultaneously!

Goal of talk:

- Highlight a few interesting examples where **new theoretical frameworks** and abstractions lead to **new algorithmic ideas**

Part 2

How to utilize extra **unreliable** information in algorithm design?

Worst-case analysis
Pessimistically ignore anything unreliable

Heuristic
Optimistically assume information is good, because it often is

Learning-Augmented Algorithms
Do both simultaneously!

Goal of talk:

- Highlight a few interesting examples where **new theoretical frameworks** and abstractions lead to **new algorithmic ideas**
- Focus on data structures and optimizing runtime (though there's lots of exciting work in other relevant areas, like streaming, online algorithms, and more!)

Part 2

How to utilize extra **unreliable** information in algorithm design?

Worst-case analysis
Pessimistically ignore anything unreliable

Heuristic
Optimistically assume information is good, because it often is

Learning-Augmented Algorithms
Do both simultaneously!

Goal of talk:

- Highlight a few interesting examples where **new theoretical frameworks** and abstractions lead to **new algorithmic ideas**
- Focus on data structures and optimizing runtime (though there's lots of exciting work in other relevant areas, like streaming, online algorithms, and more!)
- Exciting time to get involved!

Three ideas

Three ideas

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

Three ideas

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

Three ideas

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

(3) **Randomized Algorithms:** Randomized algorithms and data structures can be hedged to take advantage of extra information by incorporating a prior

Three ideas

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

(3) **Randomized Algorithms:** Randomized algorithms and data structures can be hedged to take advantage of extra information by incorporating a prior

Sequences of Instances

Sequences of Instances

Setting: solve a **sequence** of instances $I_1, \ldots, I_T$ of some algorithmic problem

Sequences of Instances

Setting: solve a **sequence** of instances $I_1, \ldots, I_T$ of some algorithmic problem

Example: max-flow

Sequences of Instances

Setting: solve a **sequence** of instances $I_1, \ldots, I_T$ of some algorithmic problem

Example: max-flow

Goal: minimize the **total runtime** to solve all of them

Sequences of Instances

Setting: solve a **sequence** of instances $I_1, \ldots, I_T$ of some algorithmic problem

Example: max-flow

Goal: minimize the **total runtime** to solve all of them

Challenge: If could solve sequences asymptotically faster than one at a time, would get an asymptotically faster worst-case algorithm

Sequences of Instances

Setting: solve a **sequence** of instances $I_1, \ldots, I_T$ of some algorithmic problem

Example: max-flow

Goal: minimize the **total runtime** to solve all of them

Challenge: If could solve sequences asymptotically faster than one at a time, would get an asymptotically faster worst-case algorithm

Hope: Adapt to **structure** in the sequence, when it exists!

Sequences of Instances

Setting: solve a **sequence** of instances $I_1, \ldots, I_T$ of $\underbrace{\text{some algorithmic problem}}_{\textbf{Example: } \text{max-flow}}$

Goal: minimize the **total runtime** to solve all of them

Challenge: If could solve sequences asymptotically faster than one at a time, would get an asymptotically faster worst-case algorithm

Hope: Adapt to **structure** in the sequence, when it exists!

(like dynamic algorithms, but want to impose less structure)

Sequences of Instances

Setting: solve a **sequence** of instances $I_1, \ldots, I_T$ of $\underbrace{\text{some algorithmic problem}}_{}$

Example: max-flow

Goal: minimize the **total runtime** to solve all of them

Challenge: If could solve sequences asymptotically faster than one at a time, would get an asymptotically faster worst-case algorithm

Hope: Adapt to **structure** in the sequence, when it exists!

(like dynamic algorithms, but want to impose less structure)

Example: suppose flow instances came from a predictable traffic network

Sequences of Instances

Setting: solve a **sequence** of instances $I_1, \ldots, I_T$ of some algorithmic problem

Example: max-flow

Goal: minimize the **total runtime** to solve all of them

Challenge: If could solve sequences asymptotically faster than one at a time, would get an asymptotically faster worst-case algorithm

Hope: Adapt to **structure** in the sequence, when it exists!

(like dynamic algorithms, but want to impose less structure)

Example: suppose flow instances came from a predictable traffic network

Formalism: Learning-Augmented **"warm starts"** help us design and reason about algorithms for sequences of instances

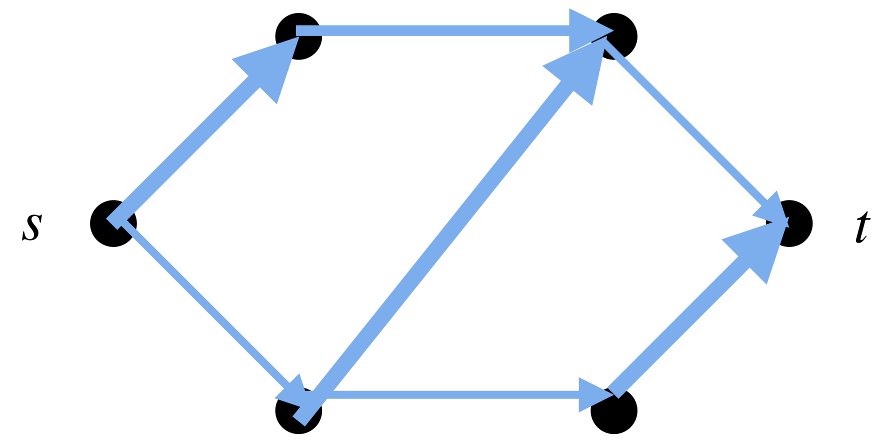
Warm Start Example: Max-Flow via Ford-Fulkerson

Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

Warm Start Example: Max-Flow via Ford-Fulkerson

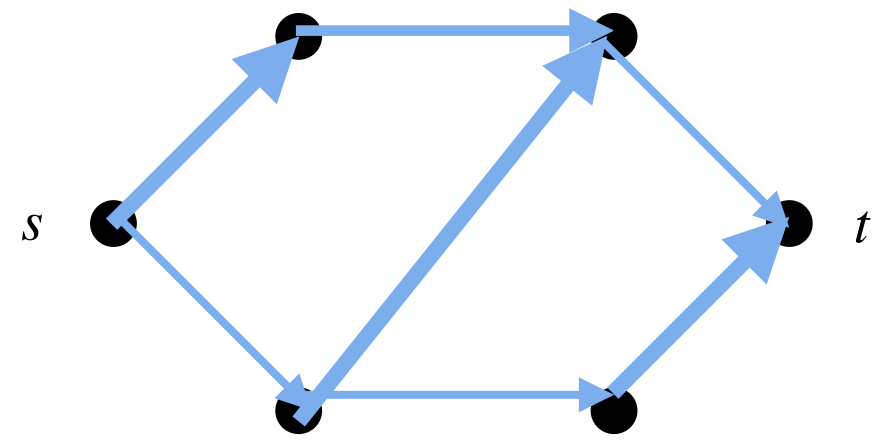
Recall Ford-Fulkerson:



Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

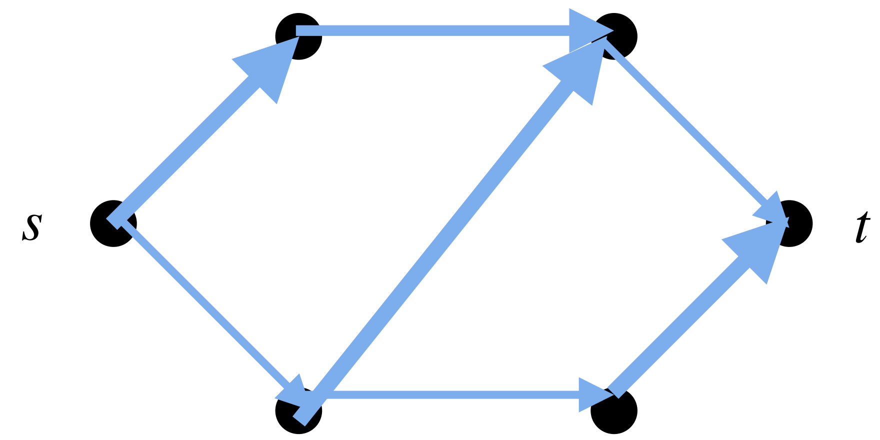
- Start with the all-0 flow



Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

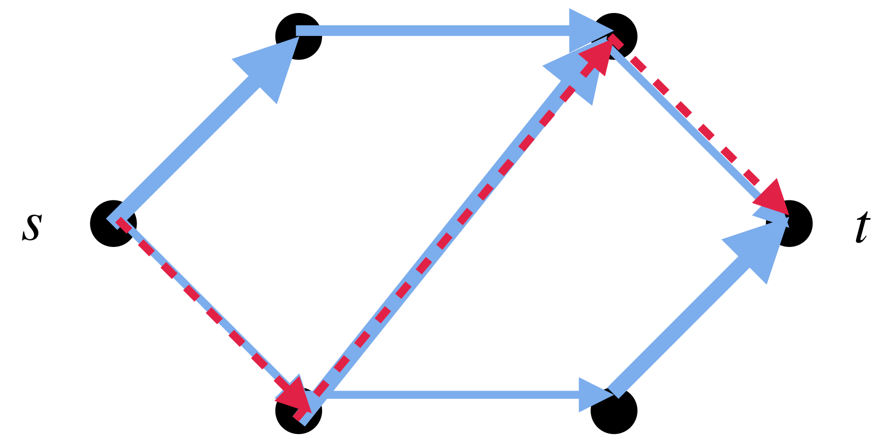
- Start with the all-0 flow
- Each iteration: find **augmenting path** to increase flow in residual graph by 1, in time $O(|E|)$



Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

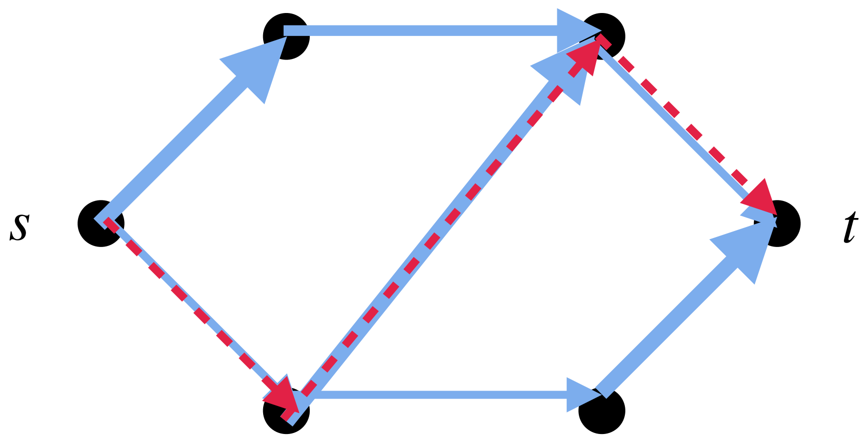
- Start with the all-0 flow
- Each iteration: find **augmenting path** to increase flow in residual graph by 1, in time $O(|E|)$



Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

- Start with the all-0 flow
- Each iteration: find **augmenting path** to increase flow in residual graph by 1, in time $O(|E|)$
- # of iterations: bounded by max flow in graph f

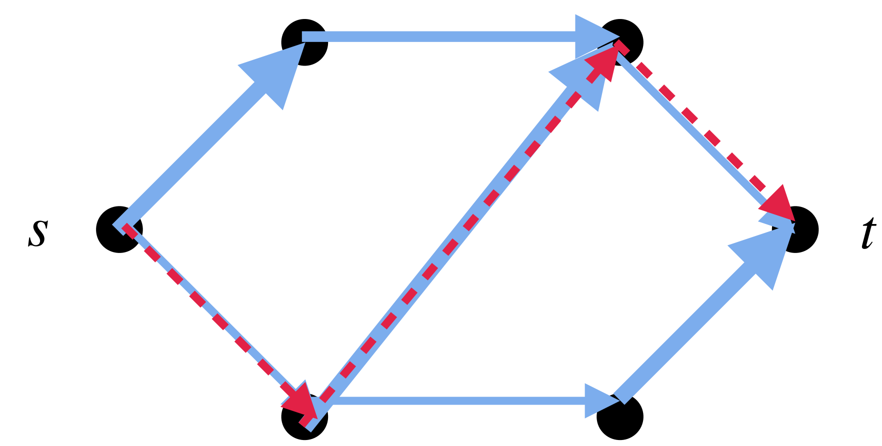


Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

- Start with the all-0 flow
- Each iteration: find **augmenting path** to increase flow in residual graph by 1, in time $O(|E|)$
- # of iterations: bounded by max flow in graph f

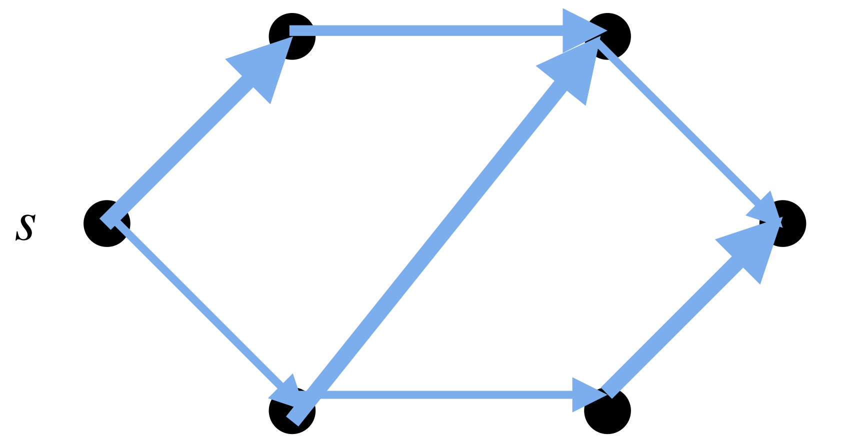
Total cost: $O(|E|f)$



Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

- Start with the all-0 flow
- Each iteration: find **augmenting path** to increase flow in residual graph by 1, in time $O(|E|)$
- # of iterations: bounded by max flow in graph f

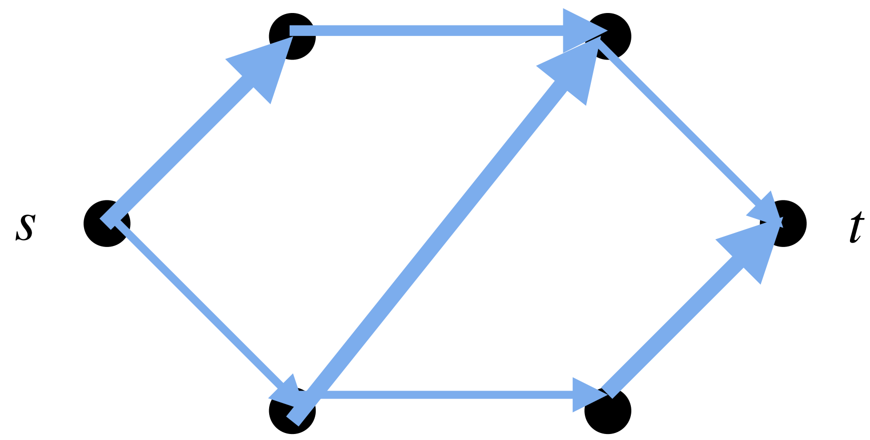


Total cost: $O(|E|f)$

Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

- Start with the all-0 flow
- Each iteration: find **augmenting path** to increase flow in residual graph by 1, in time $O(|E|)$
- # of iterations: bounded by max flow in graph f



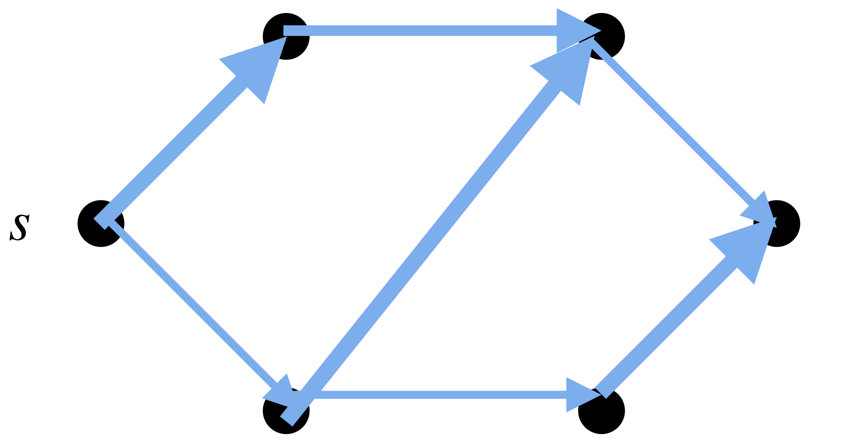
Total cost: $O(|E|f)$

Warm start algorithm [Davies Moseley Vassilvitskii Wang '23]

Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

- Start with the all-0 flow
- Each iteration: find **augmenting path** to increase flow in residual graph by 1, in time $O(|E|)$
- # of iterations: bounded by max flow in graph f



Total cost: $O(|E|f)$

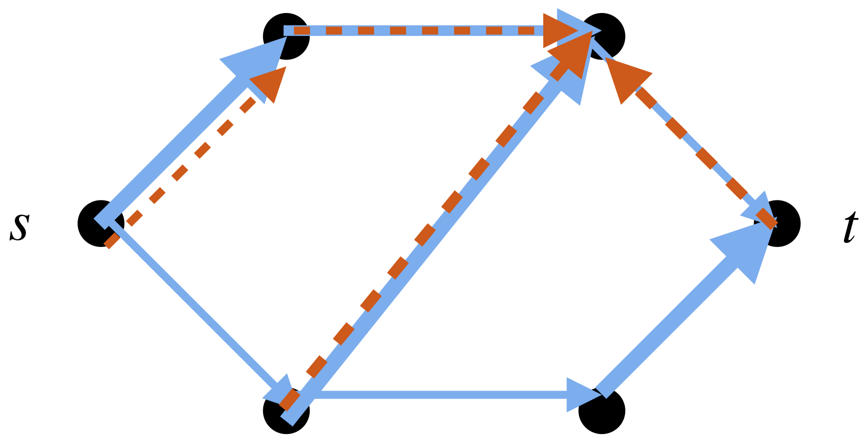
Warm start algorithm [Davies Moseley Vassilvitskii Wang '23]

- Start with a potentially **infeasible prediction** p for the flow on every edge

Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

- Start with the all-0 flow
- Each iteration: find **augmenting path** to increase flow in residual graph by 1, in time $O(|E|)$
- # of iterations: bounded by max flow in graph f



Total cost: $O(|E|f)$

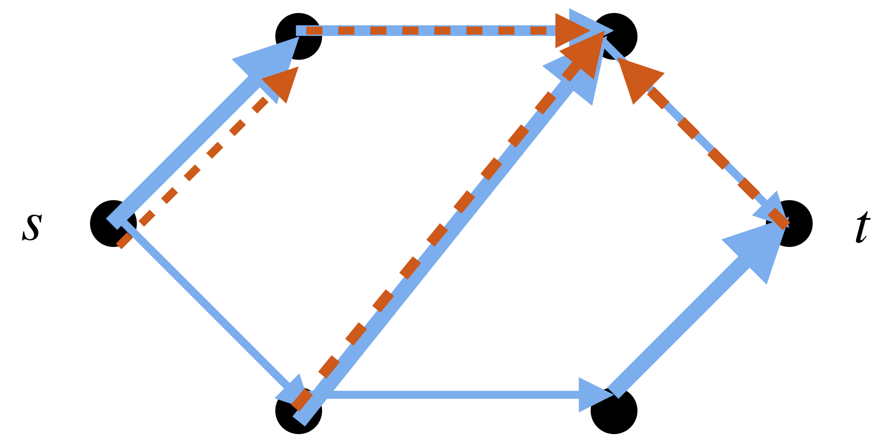
Warm start algorithm [Davies Moseley Vassilvitskii Wang '23]

- Start with a potentially **infeasible prediction** p for the flow on every edge

Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

- Start with the all-0 flow
- Each iteration: find **augmenting path** to increase flow in residual graph by 1, in time $O(|E|)$
- # of iterations: bounded by max flow in graph f



Total cost: $O(|E|f)$

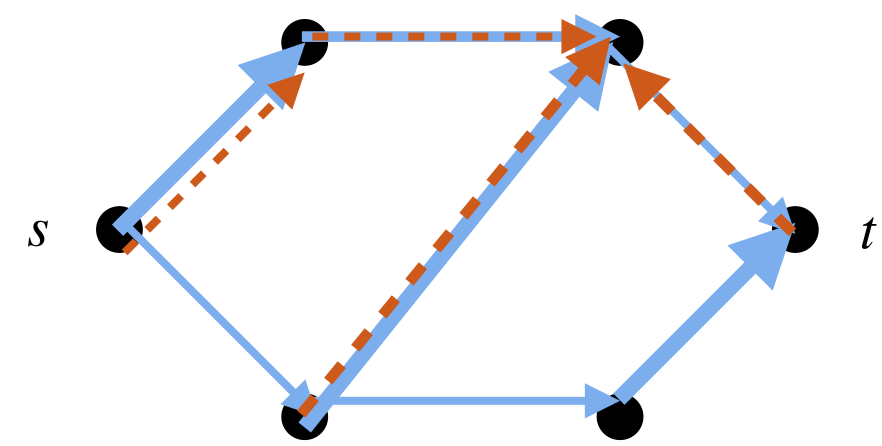
Warm start algorithm [Davies Moseley Vassilvitskii Wang '23]

- Start with a potentially **infeasible prediction** p for the flow on every edge
- Each iteration: augmenting-path like procedure in time $O(|E|)$

Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

- Start with the all-0 flow
- Each iteration: find **augmenting path** to increase flow in residual graph by 1, in time $O(|E|)$
- # of iterations: bounded by max flow in graph f



Total cost: $O(|E|f)$

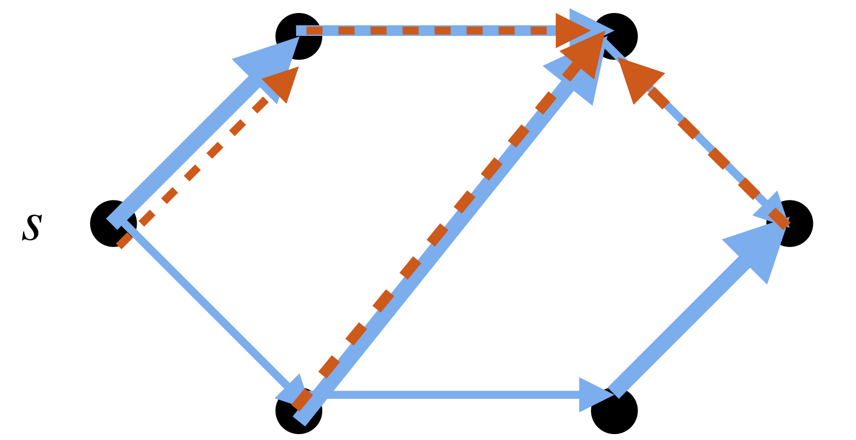
Warm start algorithm [Davies Moseley Vassilvitskii Wang '23]

- Start with a potentially **infeasible prediction** p for the flow on every edge
- Each iteration: augmenting-path like procedure in time $O(|E|)$
- # of iterations: bounded by $\eta = O(\sum_{e \in E} \left| p(e) - f(e) \right|)$, where $f(e)$ is the true flow on every edge (ℓ_1 error)

Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

- Start with the all-0 flow
- Each iteration: find **augmenting path** to increase flow in residual graph by 1, in time $O(|E|)$
- # of iterations: bounded by max flow in graph f



Total cost: $O(|E|f)$

Warm start algorithm [Davies Moseley Vassilvitskii Wang '23]

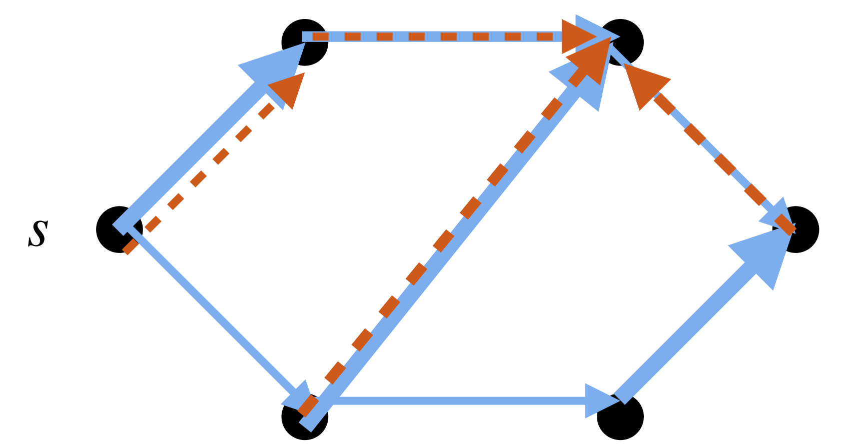
- Start with a potentially **infeasible prediction** p for the flow on every edge
- Each iteration: augmenting-path like procedure in time $O(|E|)$
- # of iterations: bounded by $\eta = O(\sum_{e \in E} \left| p(e) - f(e) \right|)$, where $f(e)$ is the true flow on every edge (ℓ_1 error)

Total cost: $O(|E| \cdot \min\{\eta, f\})$

Warm Start Example: Max-Flow via Ford-Fulkerson

Recall Ford-Fulkerson:

- Start with the all-0 flow
- Each iteration: find **augmenting path** to increase flow in residual graph by 1, in time $O(|E|)$
- # of iterations: bounded by max flow in graph f



Total cost: $O(|E|f)$

Warm start algorithm [Davies Moseley Vassilvitskii Wang '23]

- Start with a potentially **infeasible prediction** p for the flow on every edge
- Each iteration: augmenting-path like procedure in time $O(|E|)$
- # of iterations: bounded by $\eta = O(\sum_{e \in E} \left| p(e) - f(e) \right|)$, where $f(e)$ is the true flow on every edge (ℓ_1 error)

Total cost: $O(|E| \cdot \min\{\eta, f\})$

Always outputs correct answer!
Can we optimize runtime?

What to do with a warm start algorithm?

What to do with a warm start algorithm?

Challenge: a warm-start algorithm is only as good as the prediction!

What to do with a warm start algorithm?

Challenge: a warm-start algorithm is only as good as the prediction!

For new instance, must predict the optimal flow

What to do with a warm start algorithm?

Challenge: a warm-start algorithm is only as good as the prediction!

For new instance, must predict the optimal flow

$\longrightarrow$ very high-dimensional learning problem

What to do with a warm start algorithm?

Challenge: a warm-start algorithm is only as good as the prediction!

For new instance, must predict the optimal flow

$\longrightarrow$ very high-dimensional learning problem

Simple idea: For a **sequence** of related instances, use

"yesterday's solution as today's prediction"

What to do with a warm start algorithm?

Challenge: a warm-start algorithm is only as good as the prediction!

For new instance, must predict the optimal flow

$\longrightarrow$ very high-dimensional learning problem

Simple idea: For a **sequence** of related instances, use

"yesterday's solution as today's prediction"

[DMVW '23] Experiments on image segmentation tasks from frames of a video

What to do with a warm start algorithm?

Challenge: a warm-start algorithm is only as good as the prediction!

For new instance, must predict the optimal flow

$\longrightarrow$ very high-dimensional learning problem

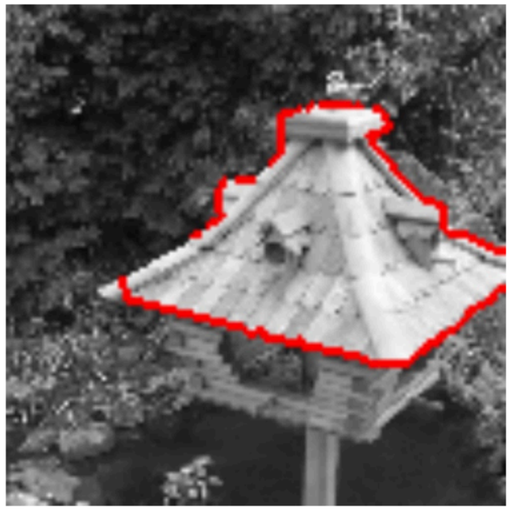
Simple idea: For a **sequence** of related instances, use

"yesterday's solution as today's prediction"

[DMVW '23] Experiments on image segmentation tasks from frames of a video

(a) Image 1

(b) Image 5



(c) Image 10

Figure 4: Cuts (red) on the first, fifth, and last images from the 120×120 pixels BIRDHOUSE sequence.

Theoretical Interpretation

Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm
[DMVW '23]:

Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges

Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges

Solution space

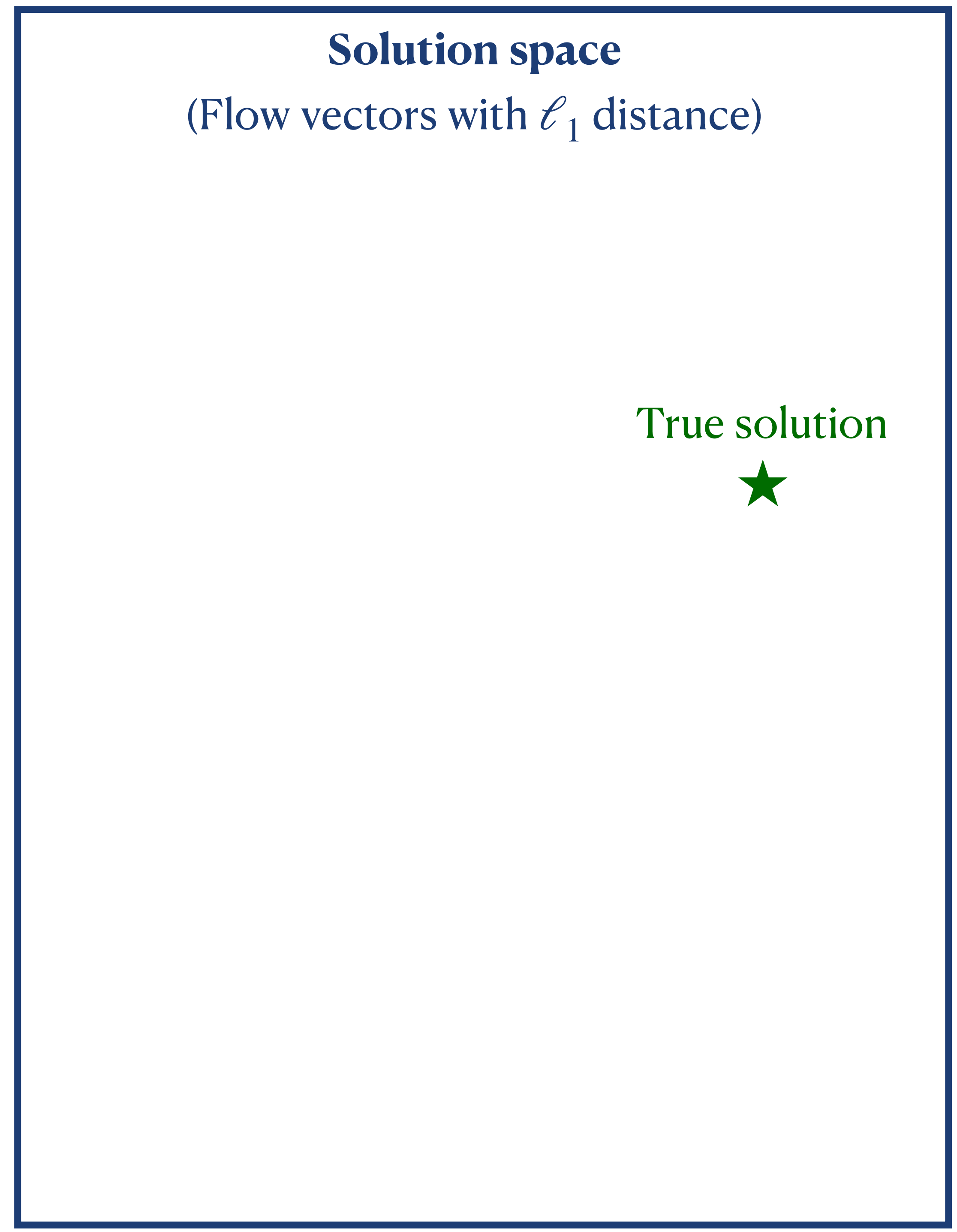
(Flow vectors with ℓ_1 distance)

Theoretical Interpretation

Is “yesterday’s solution as today’s prediction” theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW ’23]:

- Flow solution is a vector indexed by edges

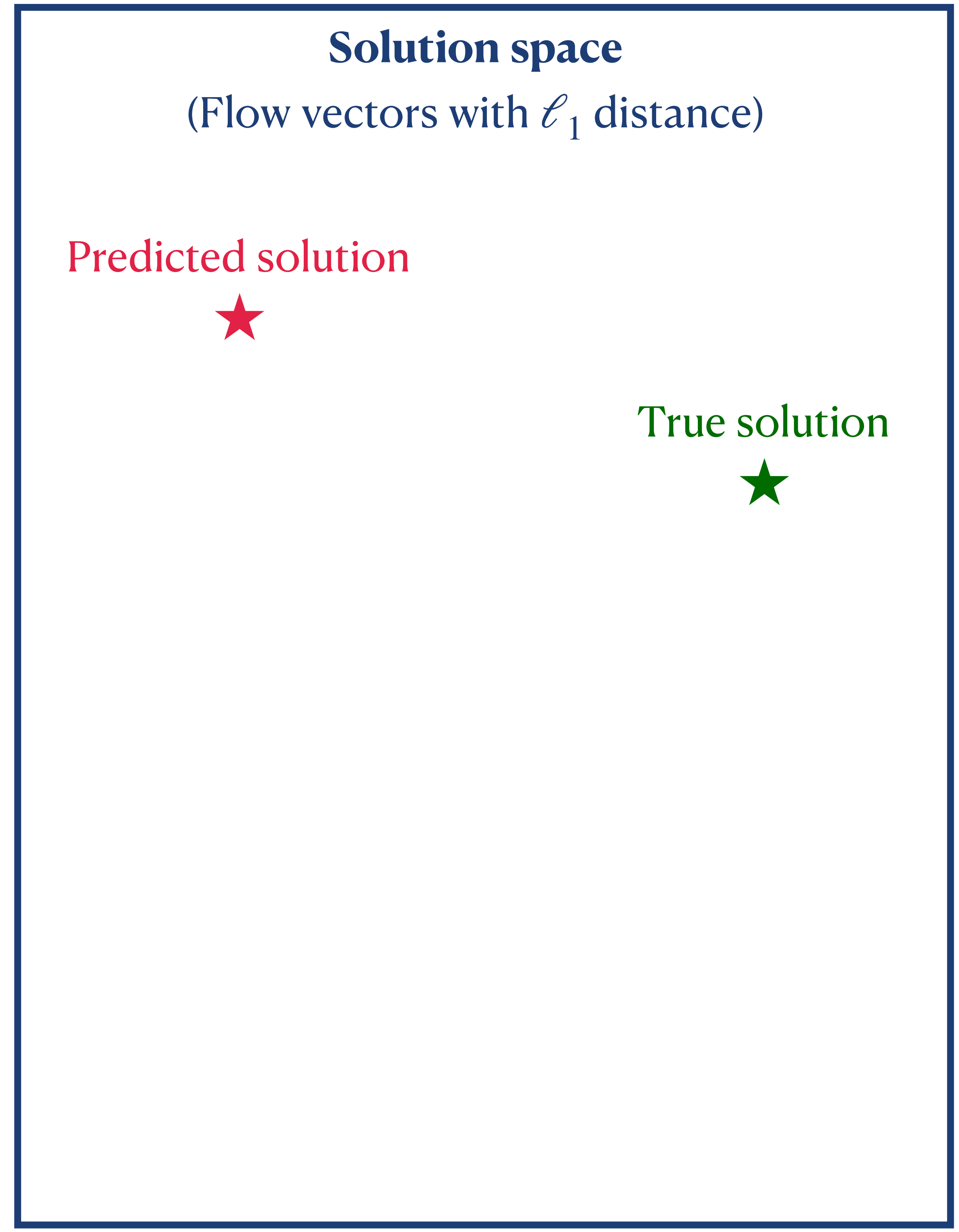


Theoretical Interpretation

Is “yesterday’s solution as today’s prediction” theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW ’23]:

- Flow solution is a vector indexed by edges

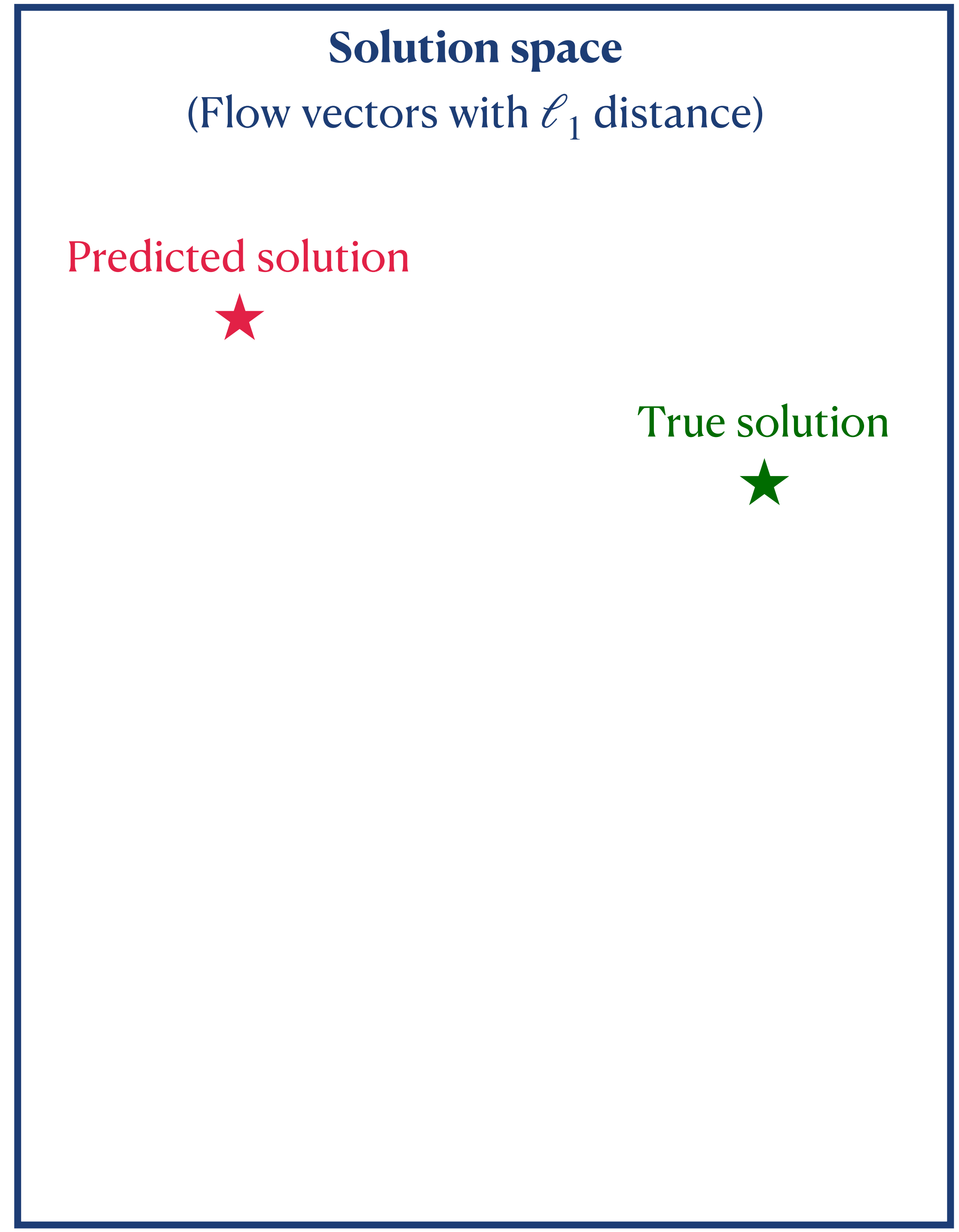


Theoretical Interpretation

Is “yesterday’s solution as today’s prediction” theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW ’23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

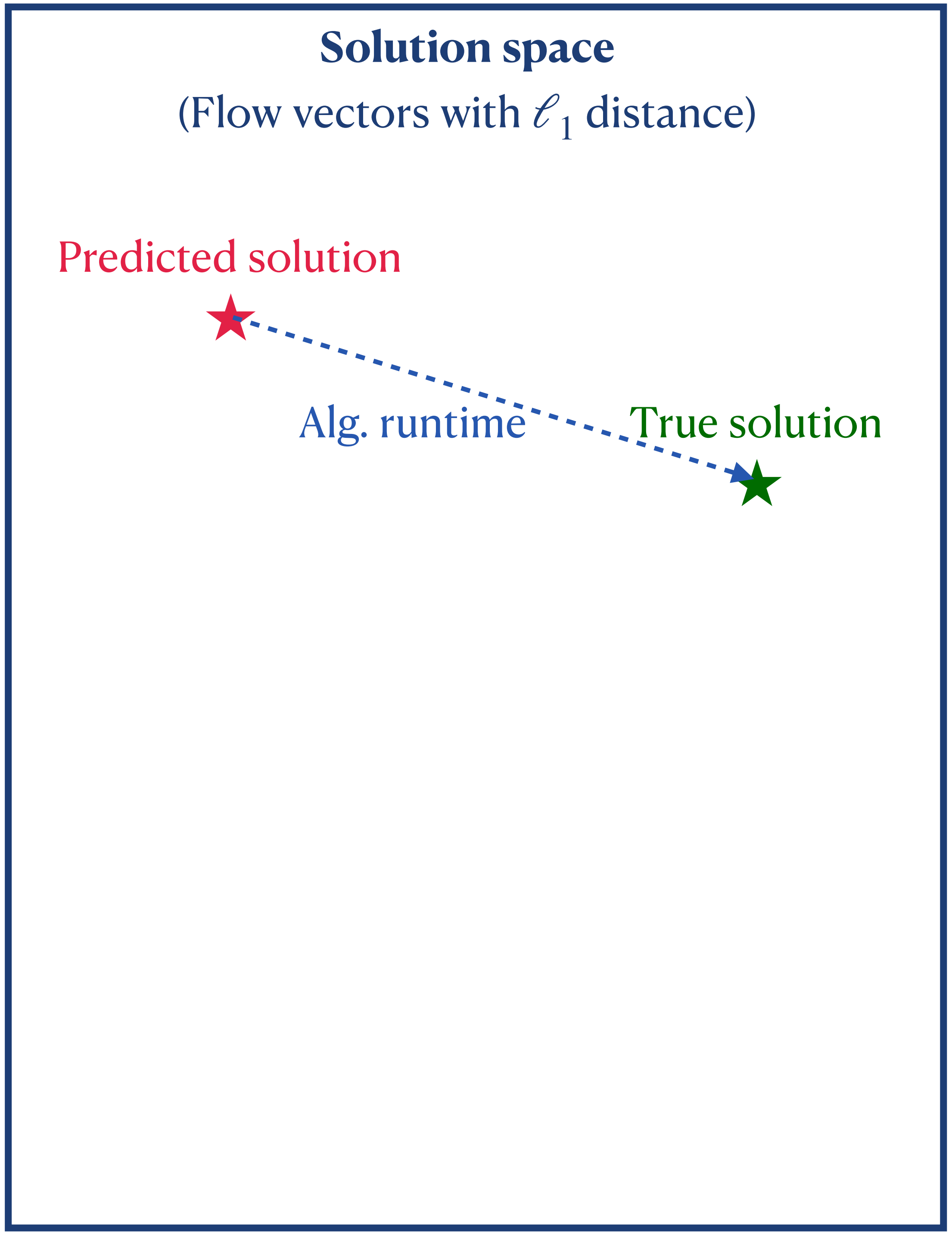


Theoretical Interpretation

Is “yesterday’s solution as today’s prediction” theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW ’23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution



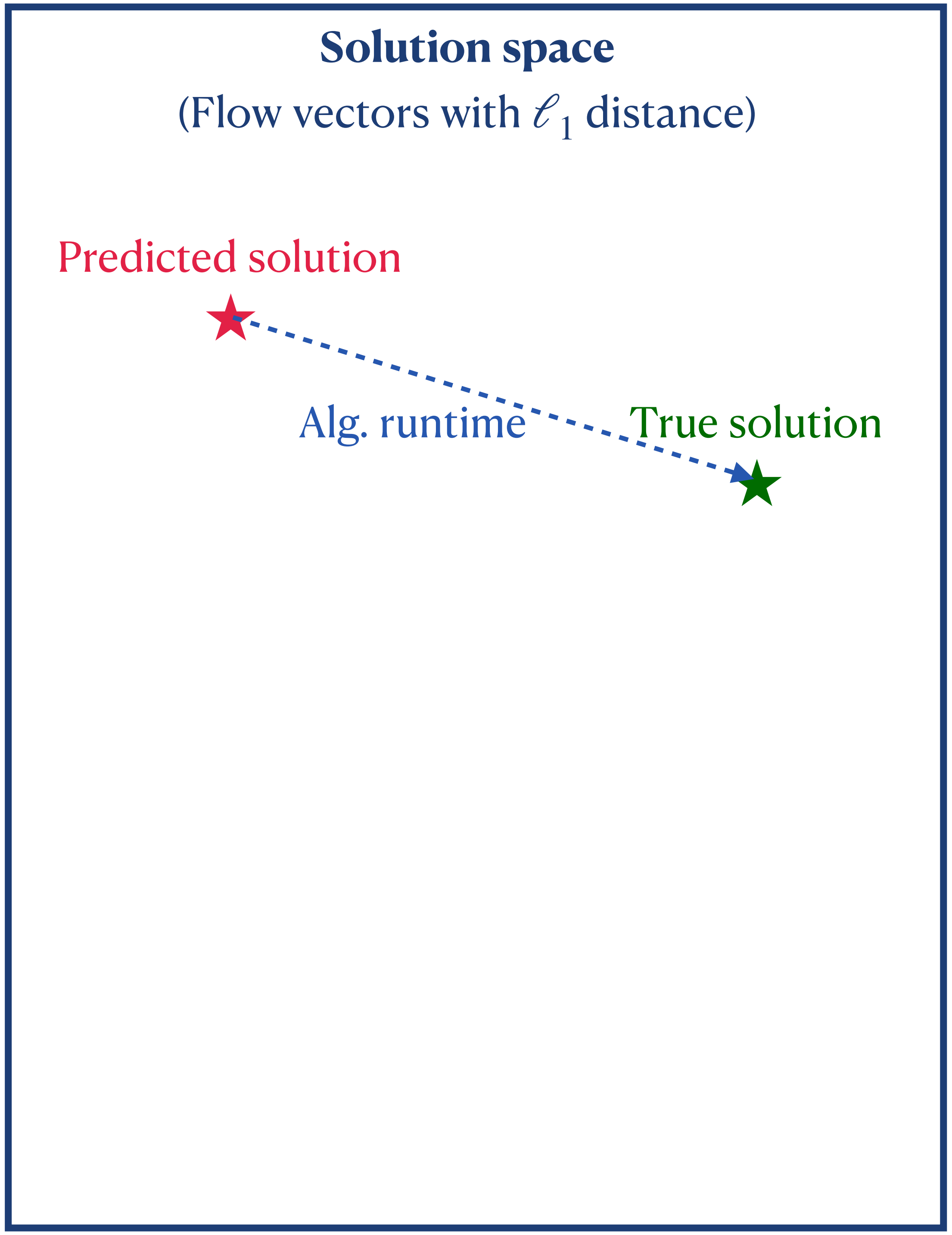
Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:



Theoretical Interpretation

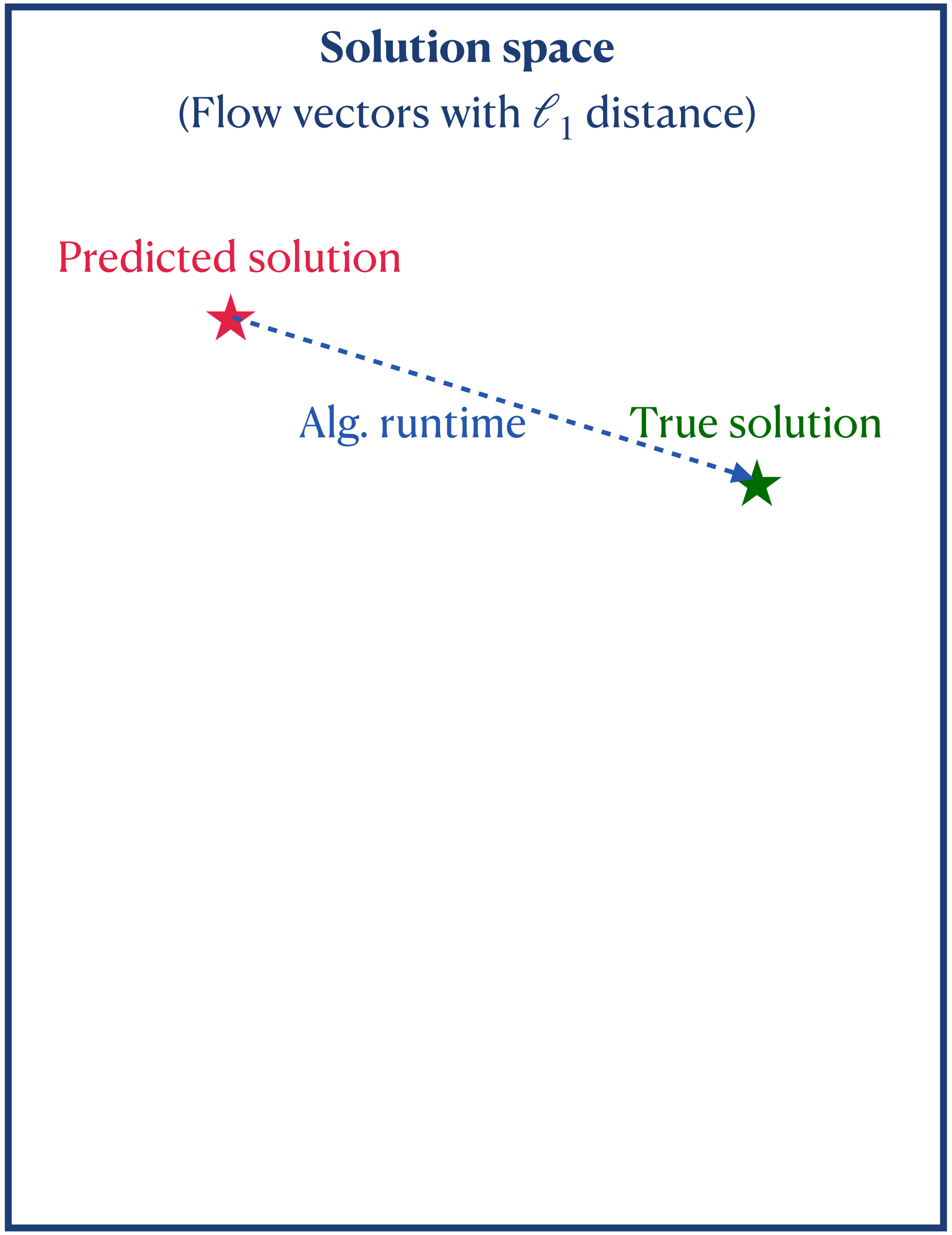
Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space



Theoretical Interpretation

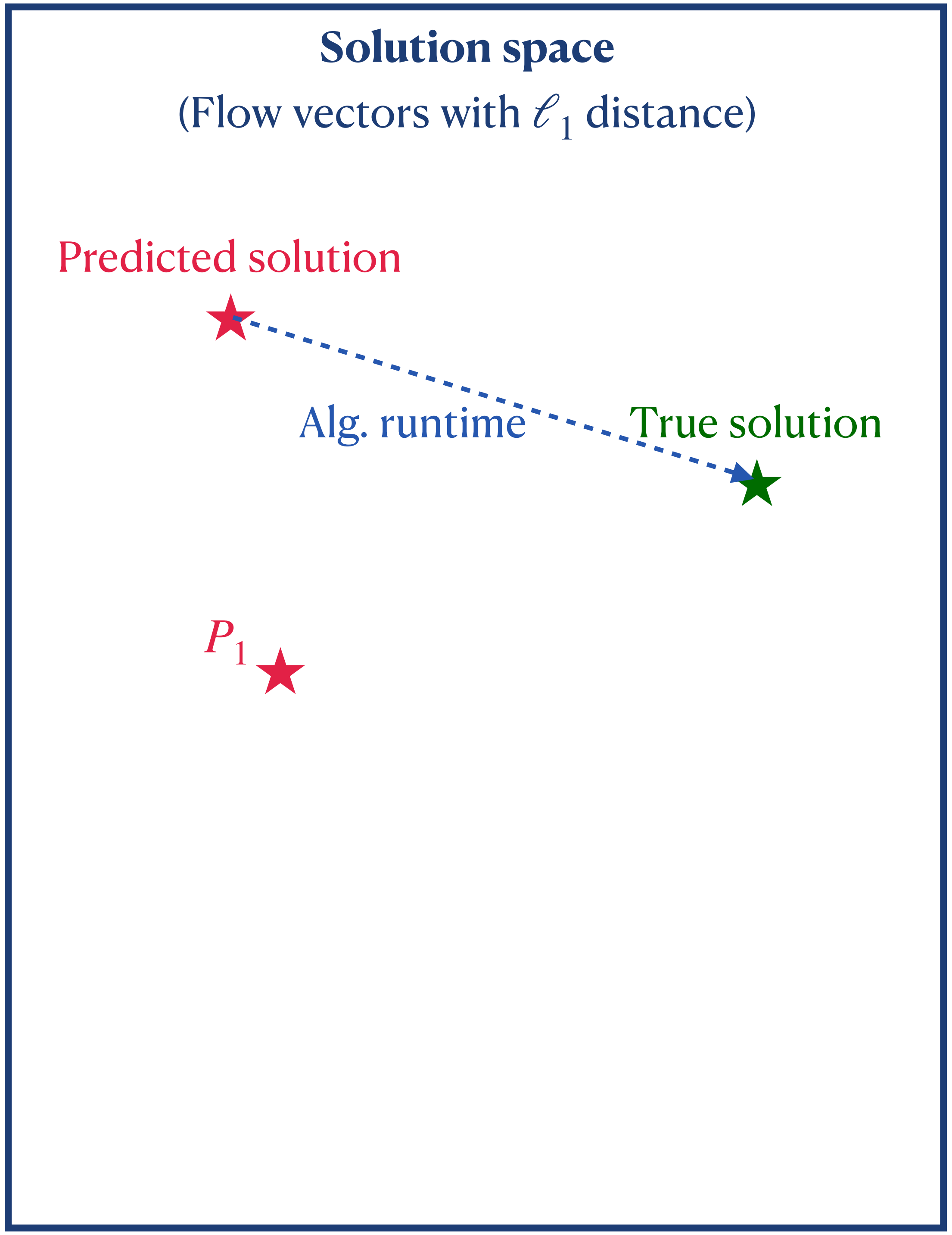
Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space



Theoretical Interpretation

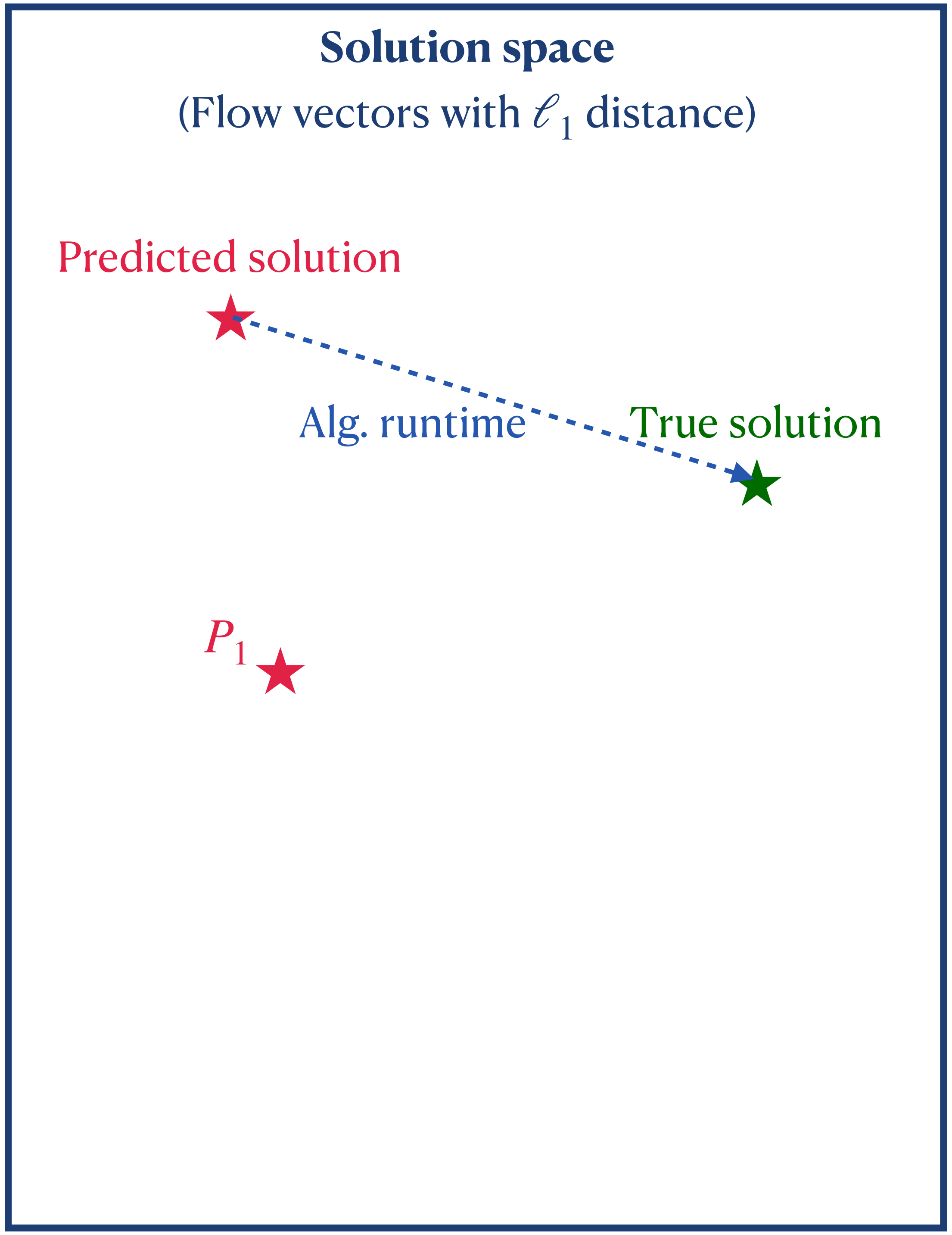
Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space
- True solution S_t is revealed



Theoretical Interpretation

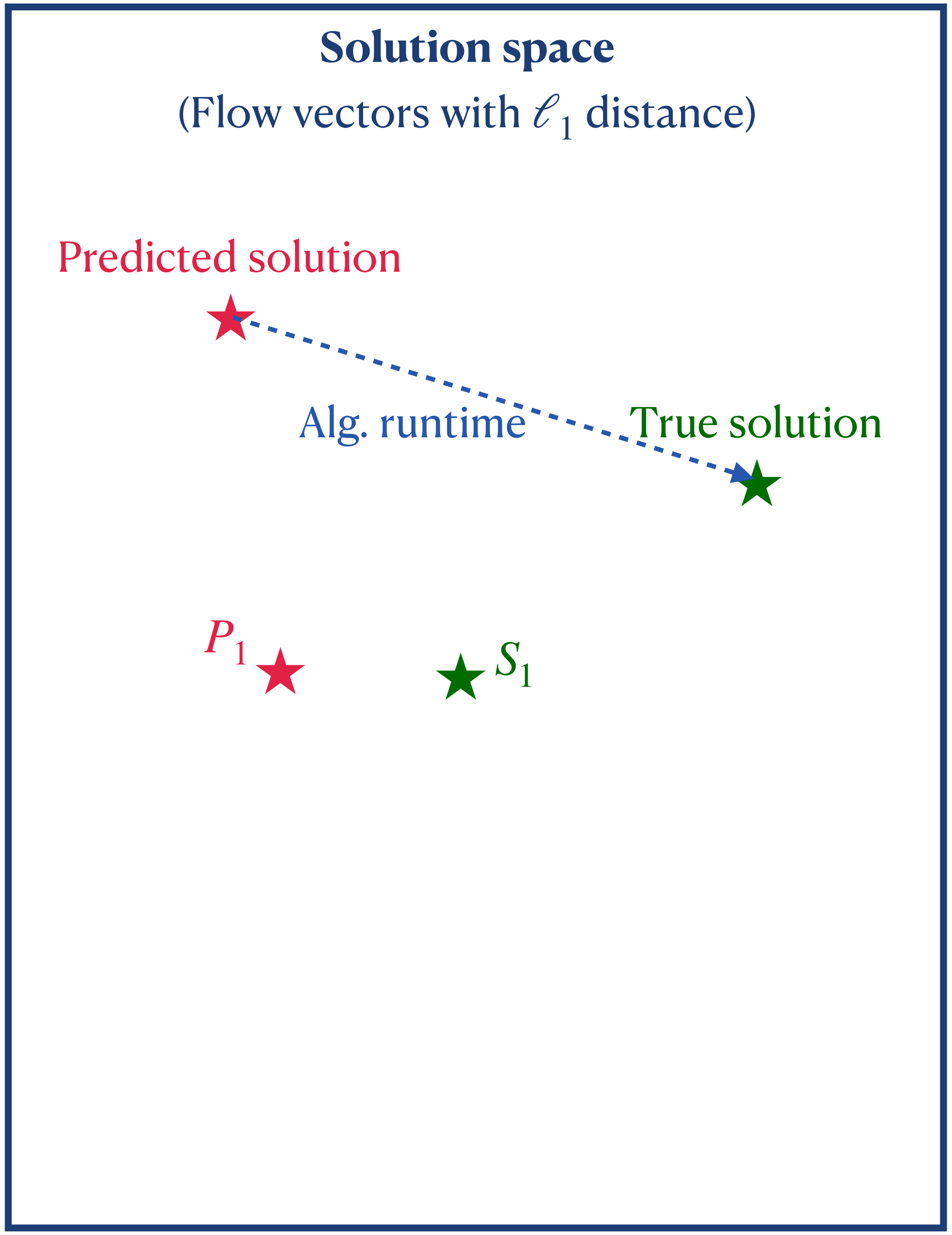
Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space
- True solution S_t is revealed



Theoretical Interpretation

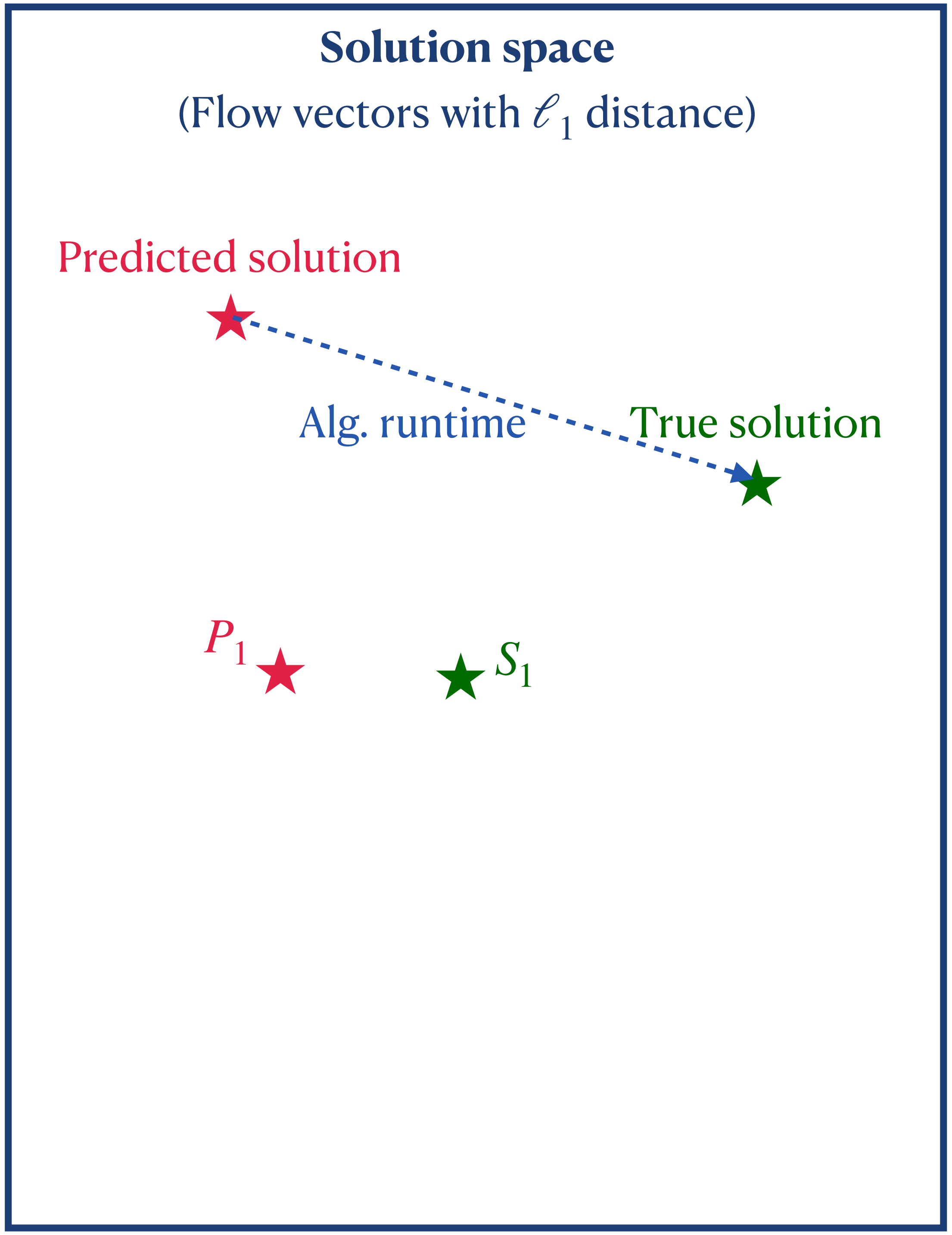
Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space
- True solution S_t is revealed
- Algorithm pays $d(P_t, S_t)$



Theoretical Interpretation

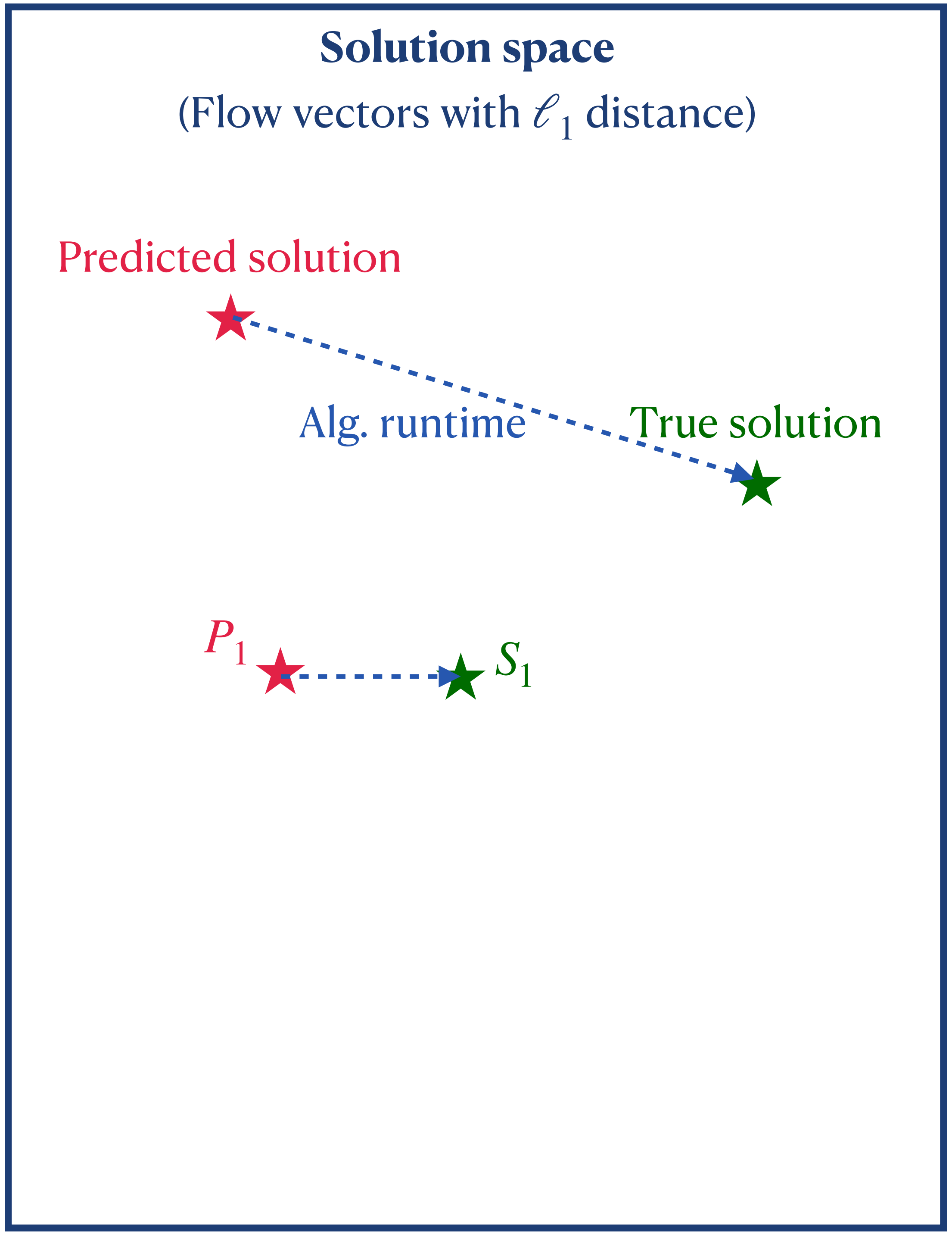
Is “yesterday’s solution as today’s prediction” theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW ’23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space
- True solution S_t is revealed
- Algorithm pays $d(P_t, S_t)$



Theoretical Interpretation

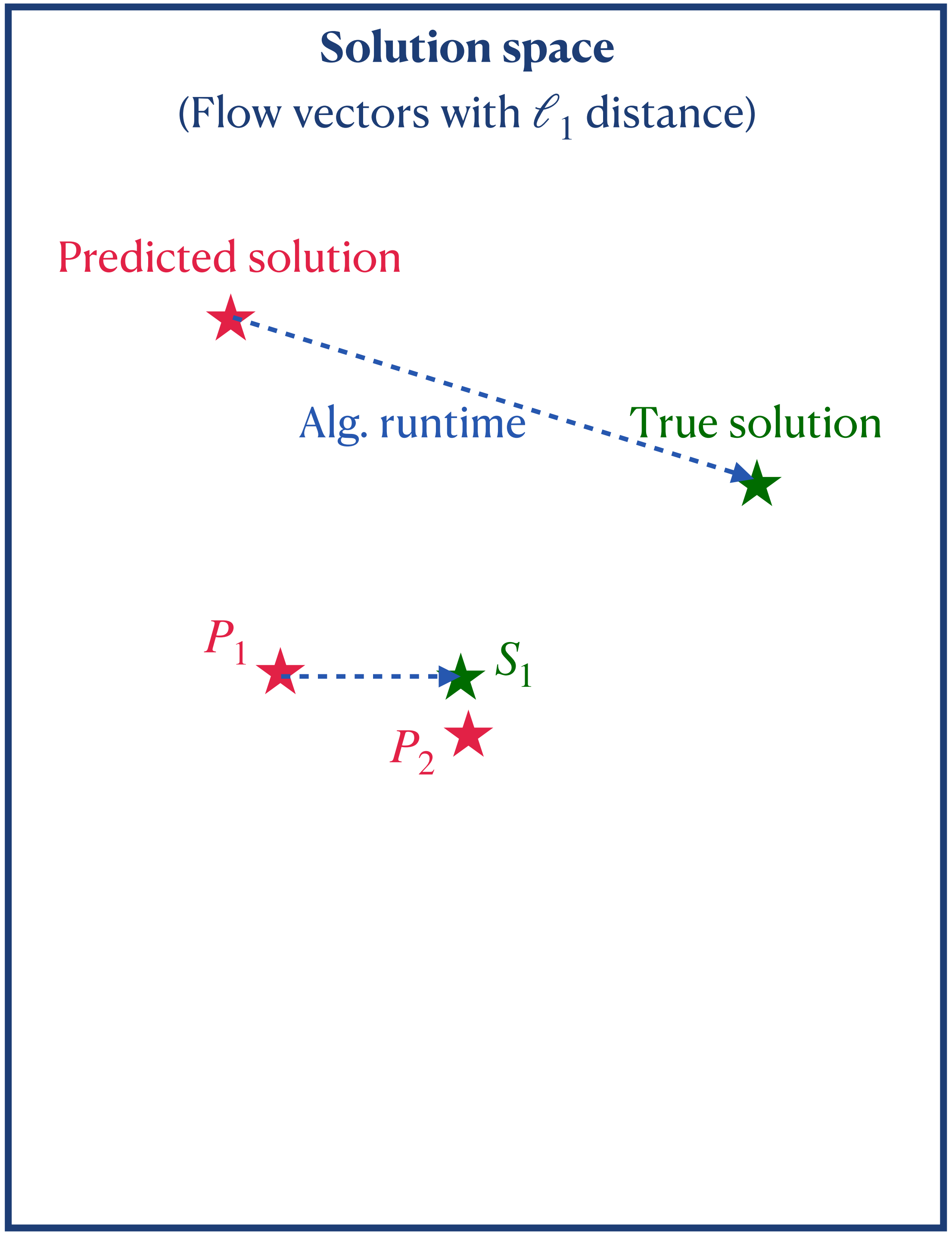
Is “yesterday’s solution as today’s prediction” theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW ’23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space
- True solution S_t is revealed
- Algorithm pays $d(P_t, S_t)$



Theoretical Interpretation

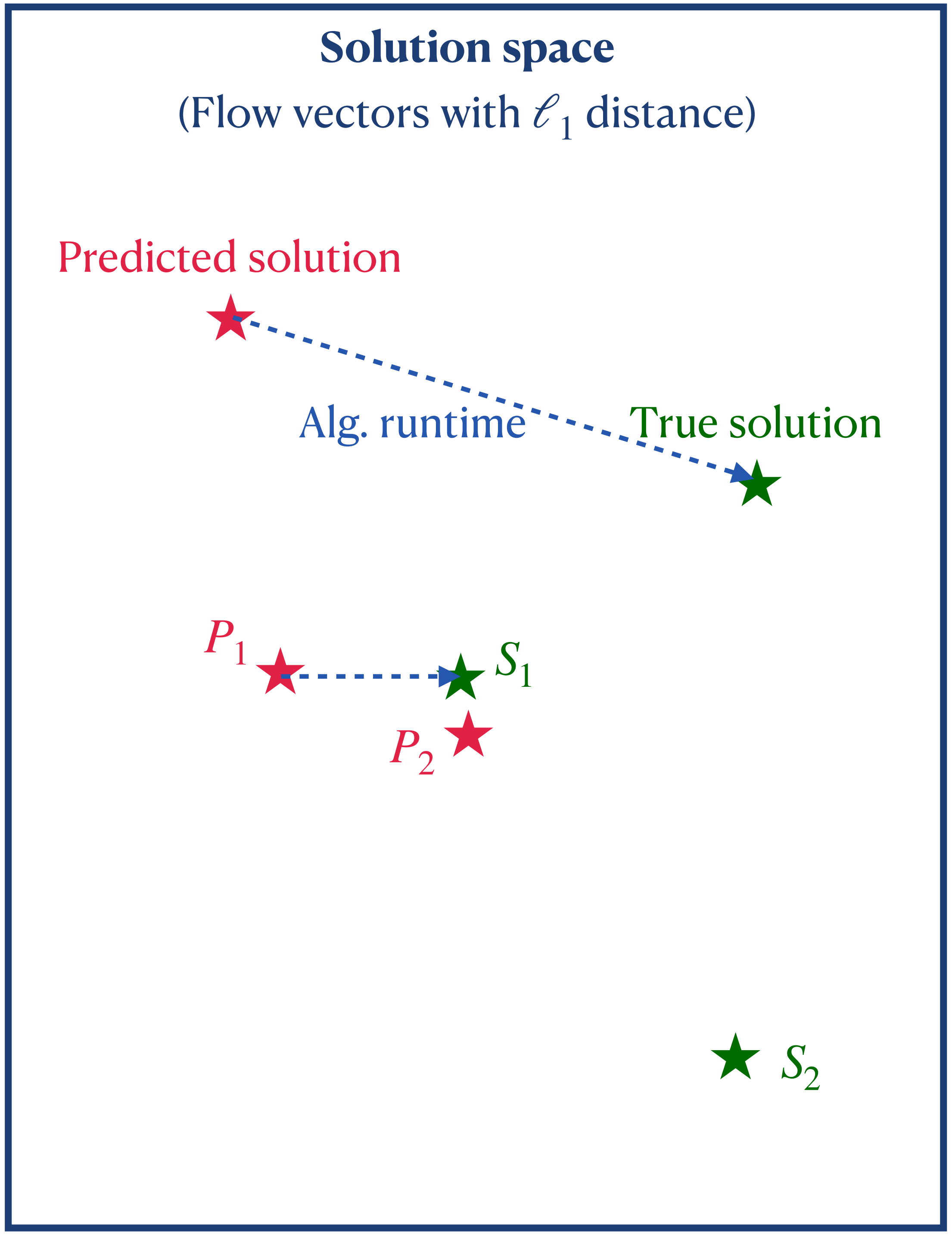
Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space
- True solution S_t is revealed
- Algorithm pays $d(P_t, S_t)$



Theoretical Interpretation

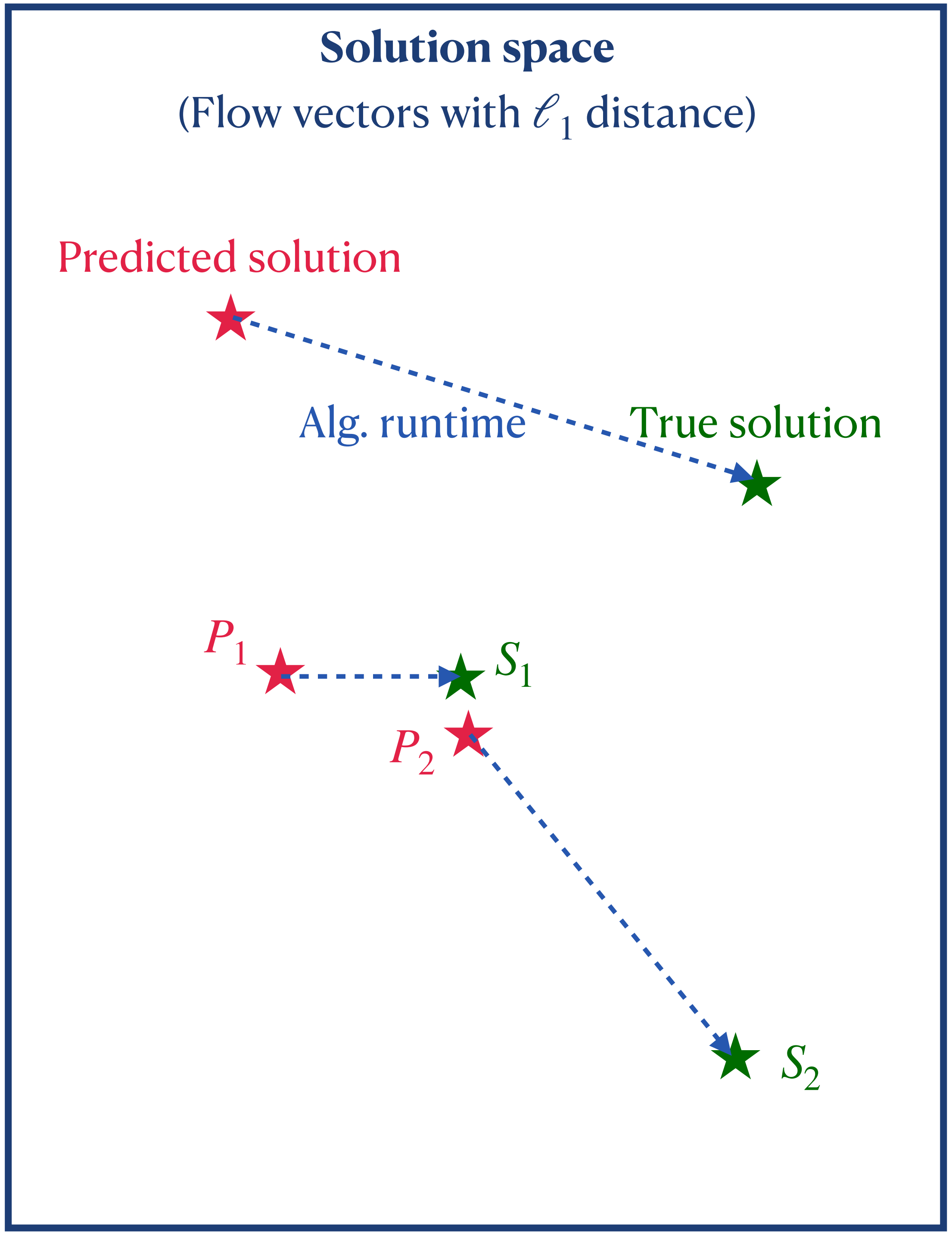
Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space
- True solution S_t is revealed
- Algorithm pays $d(P_t, S_t)$



Theoretical Interpretation

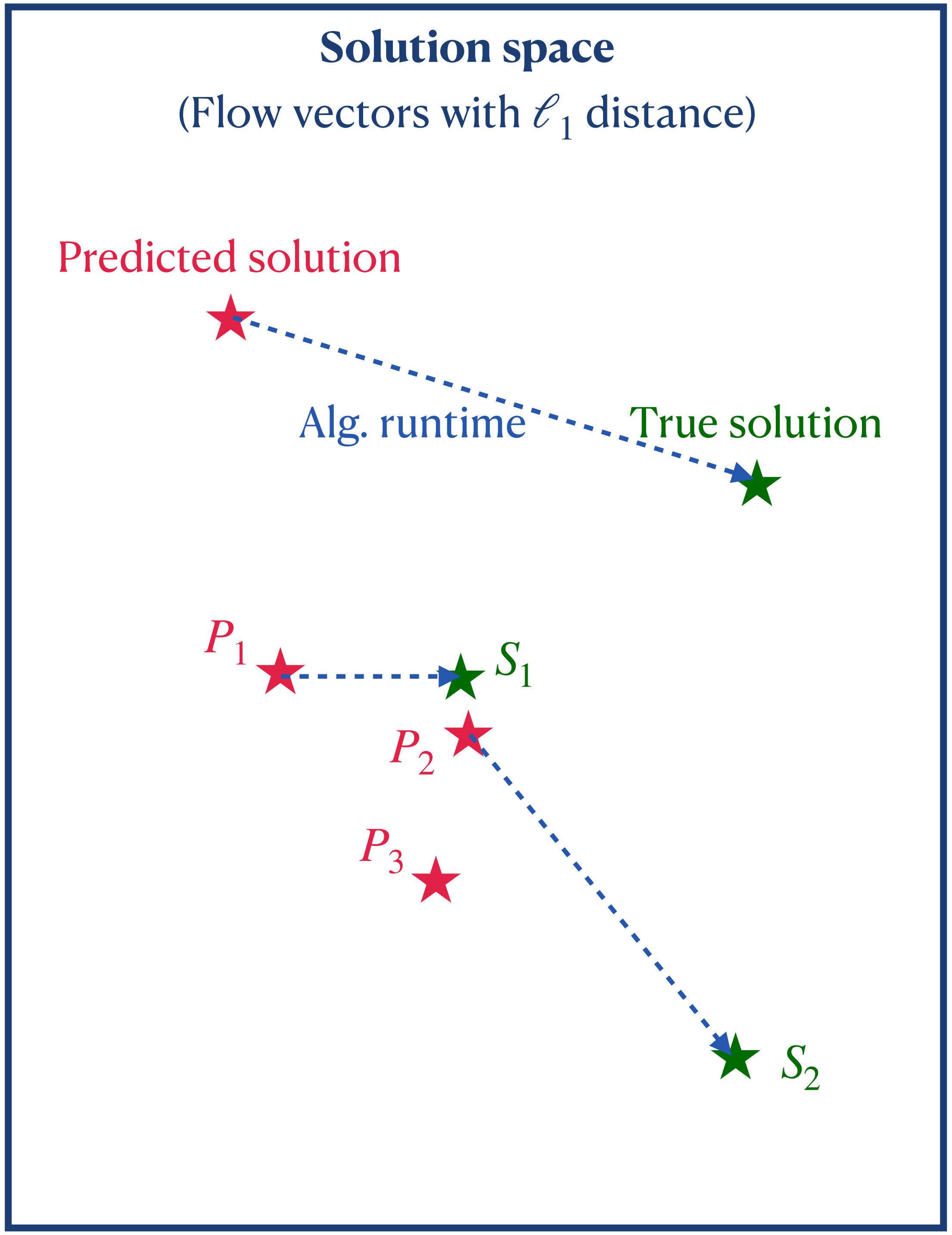
Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space
- True solution S_t is revealed
- Algorithm pays $d(P_t, S_t)$



Theoretical Interpretation

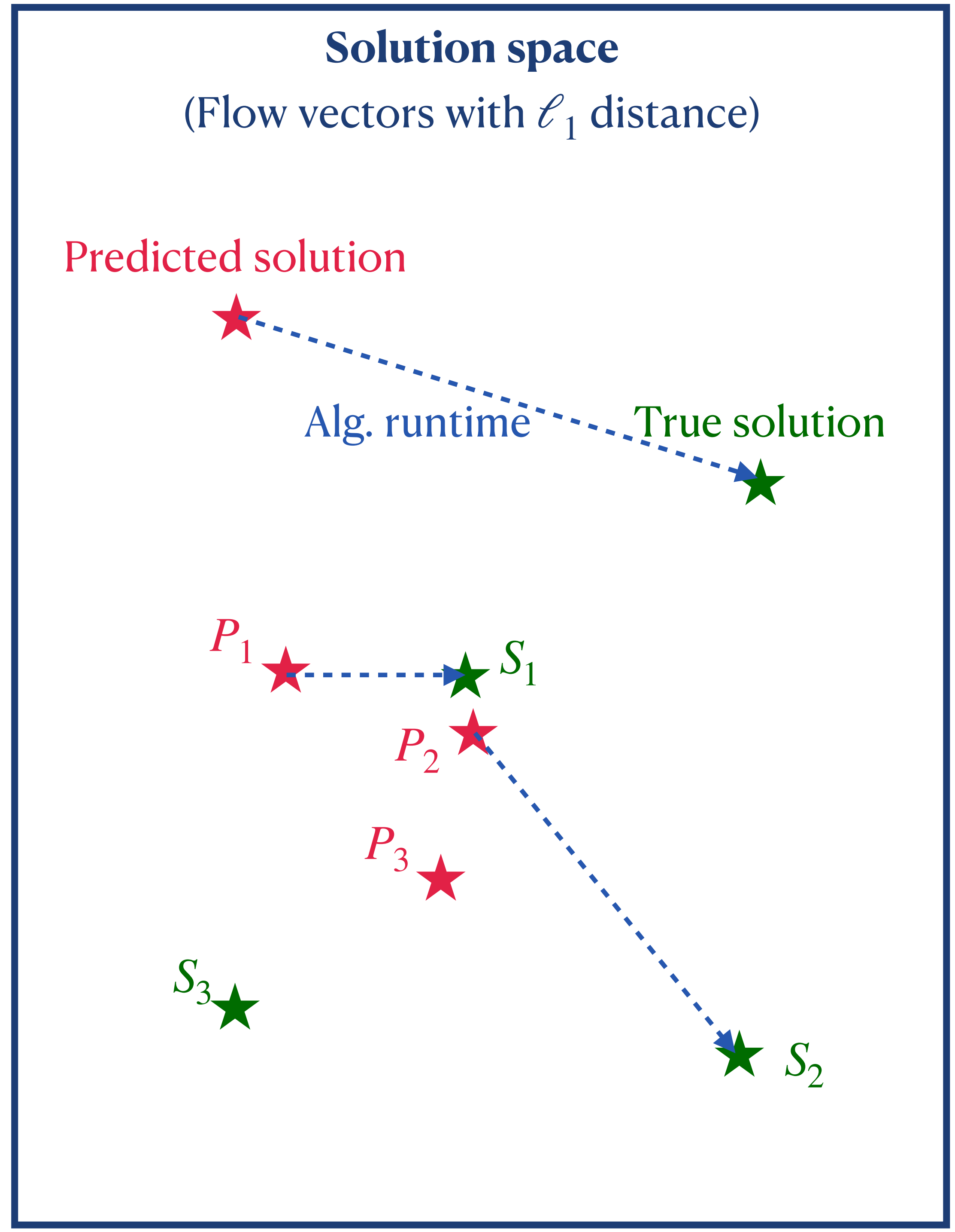
Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space
- True solution S_t is revealed
- Algorithm pays $d(P_t, S_t)$



Theoretical Interpretation

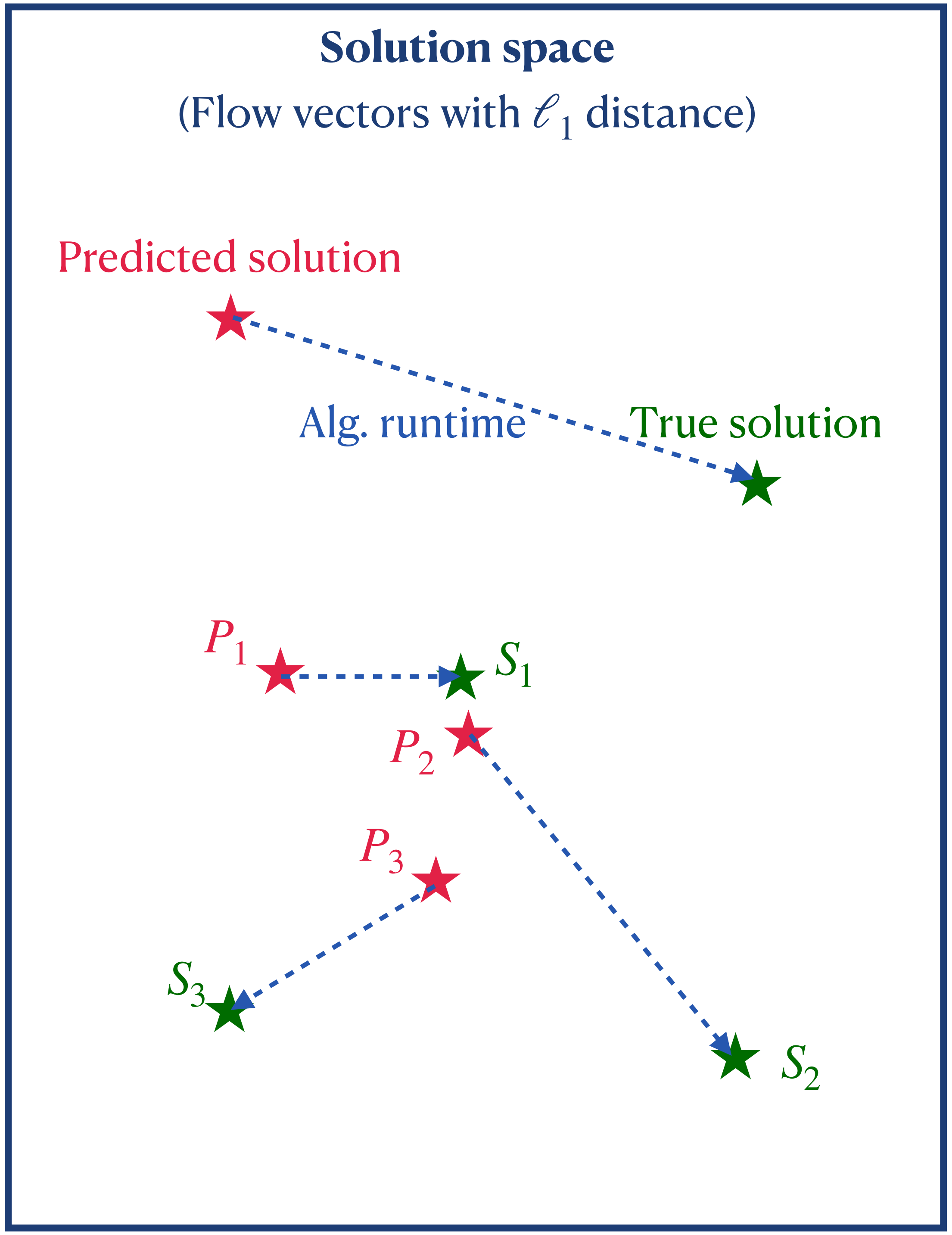
Is "yesterday's solution as today's prediction" theoretically principled?

Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space
- True solution S_t is revealed
- Algorithm pays $d(P_t, S_t)$



Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

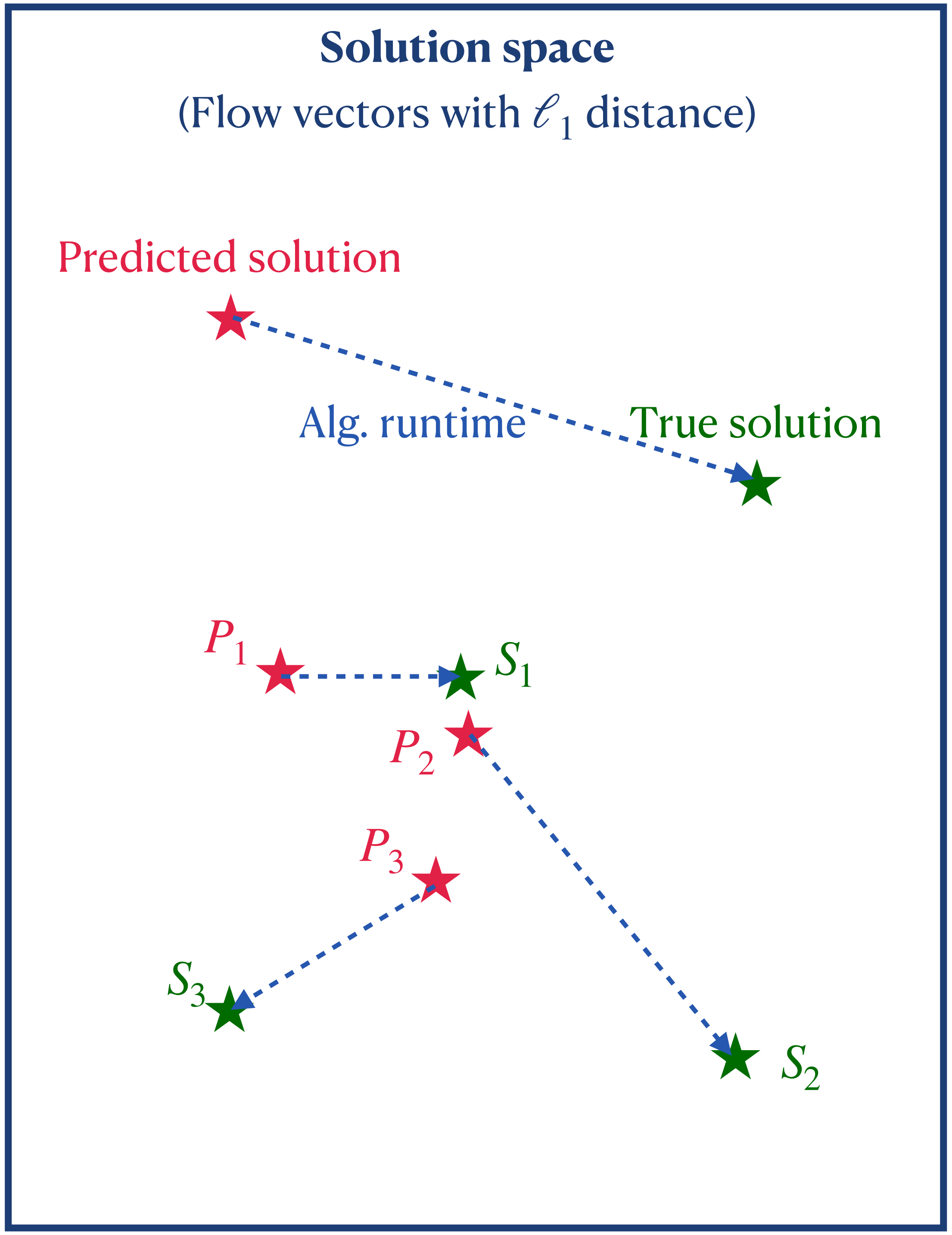
Ford-Fulkerson Warm Start algorithm [DMVW '23]:

- Flow solution is a vector indexed by edges
- Runtime of algorithm proportional to ℓ_1 **distance** between the predicted flow and the true solution

Meta problem: On each day t:

- Algorithm predicts a point P_t in the solution space
- True solution S_t is revealed
- Algorithm pays $d(P_t, S_t)$

Unlocks online algorithms toolkit!



Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

Solution space

(Flow vectors with ℓ_1 distance)

Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

- "Yesterday's solution as today's prediction" is competitive with the best fixed prediction in hindsight! [Khodak Balcan Talwalkar Vassilvitskii '22]

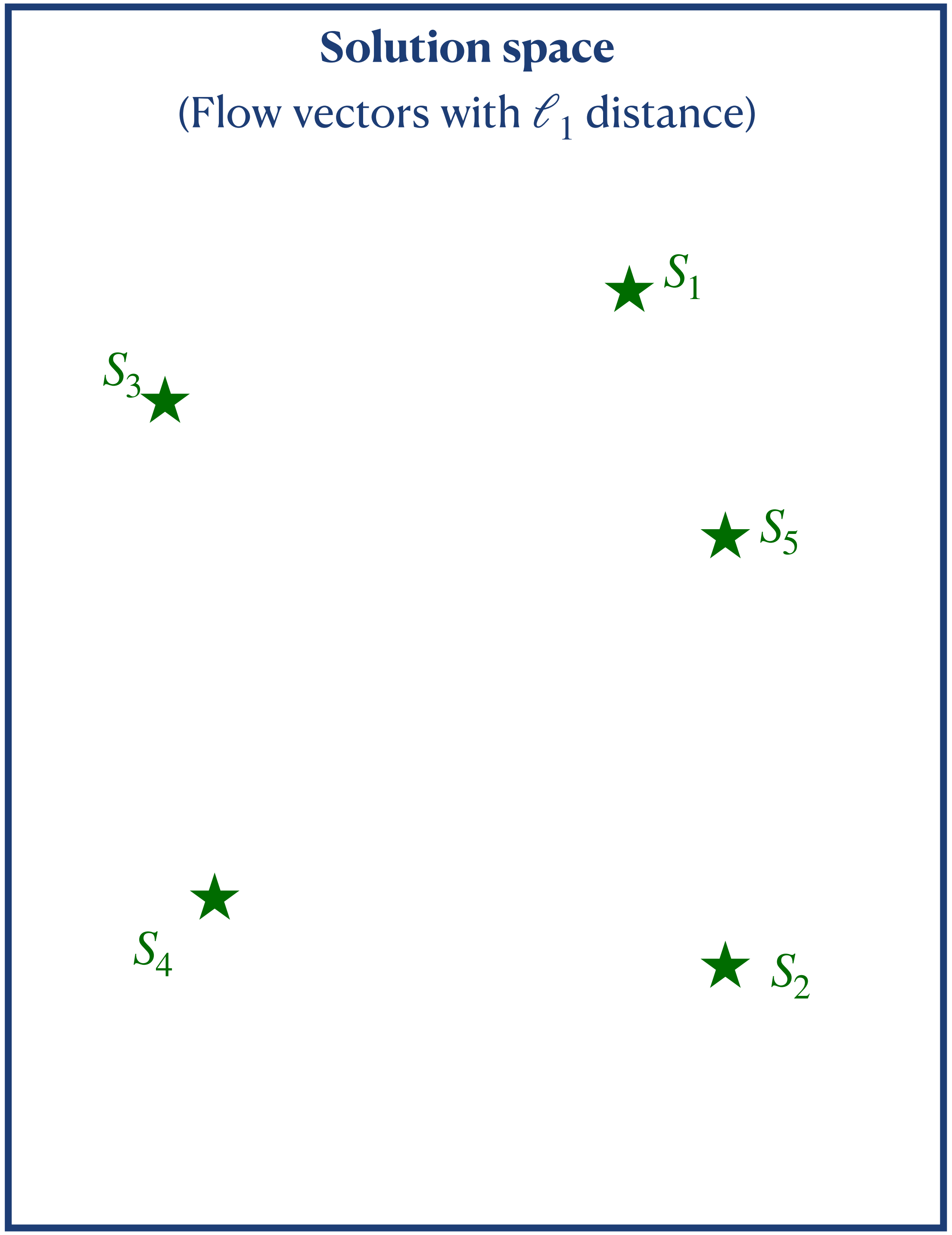
Solution space

(Flow vectors with ℓ_1 distance)

Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

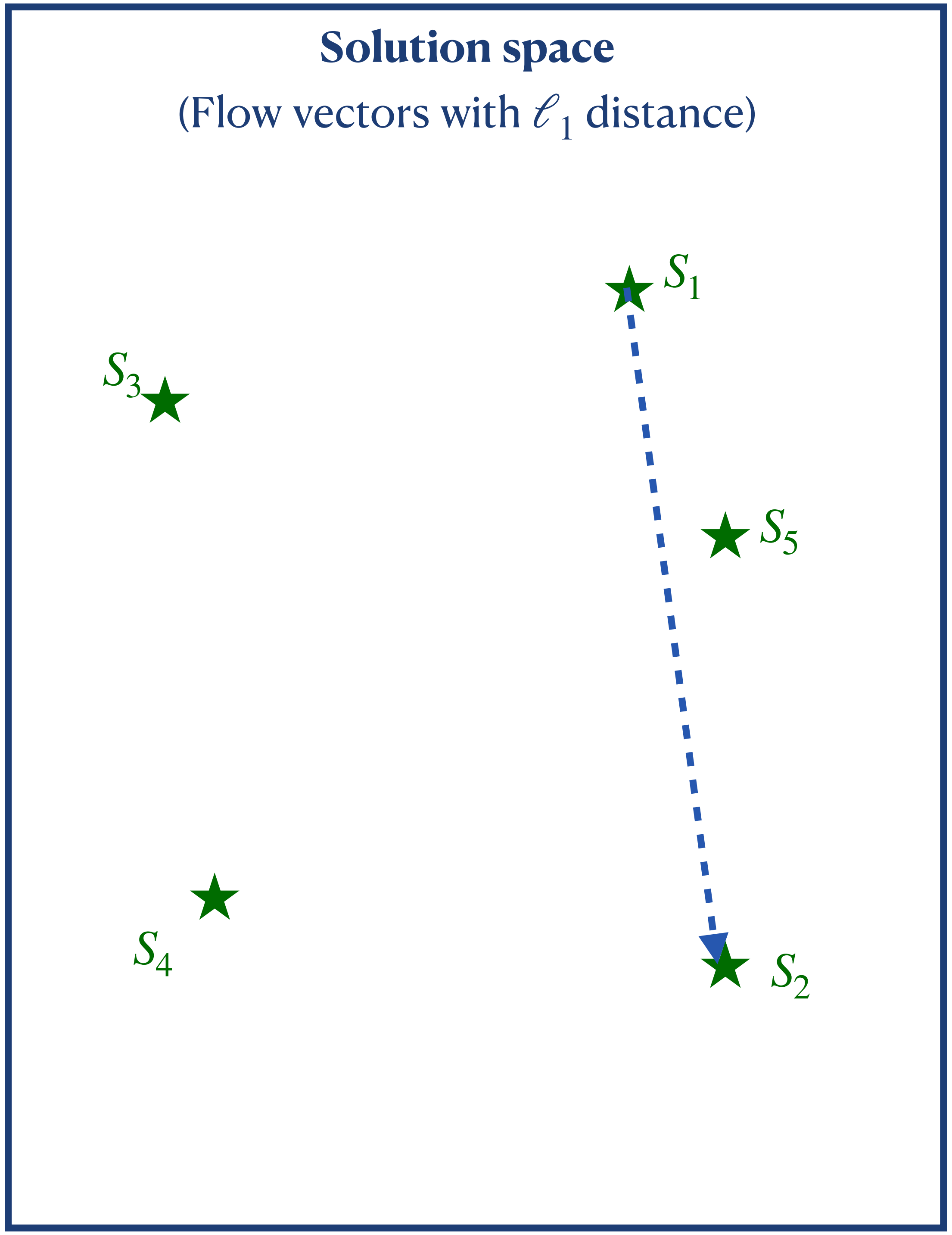
- "Yesterday's solution as today's prediction" is competitive with the best fixed prediction in hindsight! [Khodak Balcan Talwalkar Vassilvitskii '22]



Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

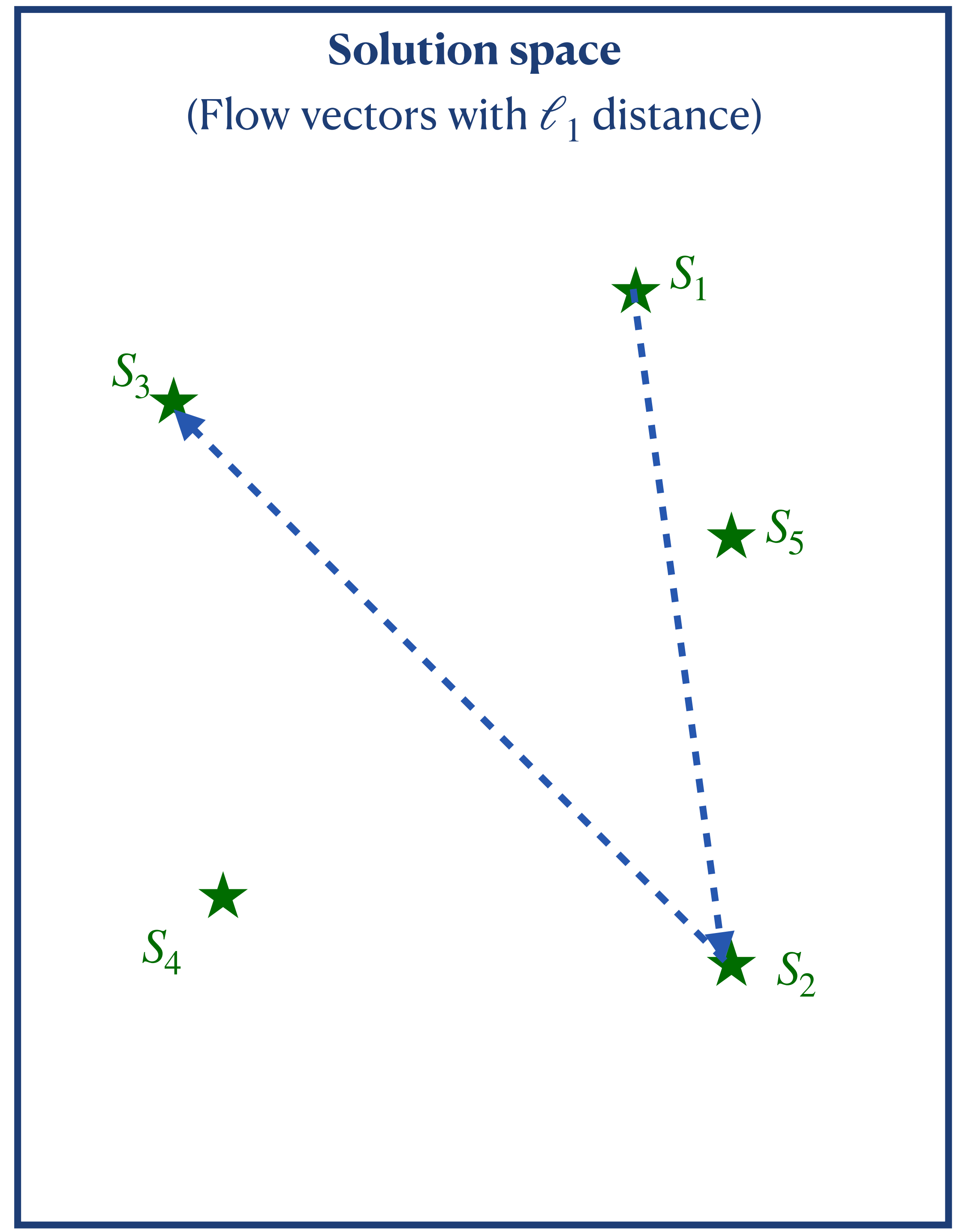
- "Yesterday's solution as today's prediction" is competitive with the best fixed prediction in hindsight! [Khodak Balcan Talwalkar Vassilvitskii '22]



Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

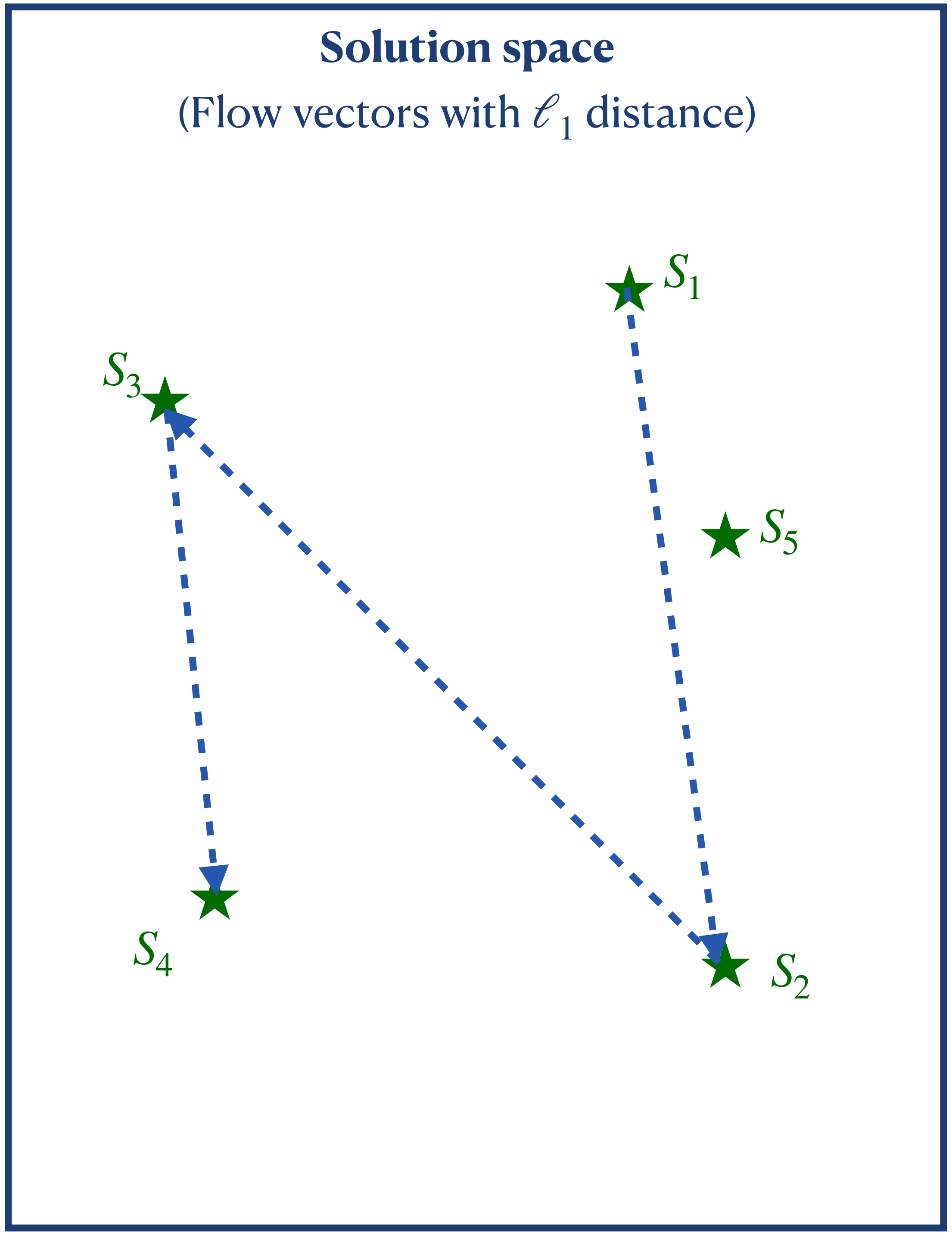
- "Yesterday's solution as today's prediction" is competitive with the best fixed prediction in hindsight! [Khodak Balcan Talwalkar Vassilvitskii '22]



Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

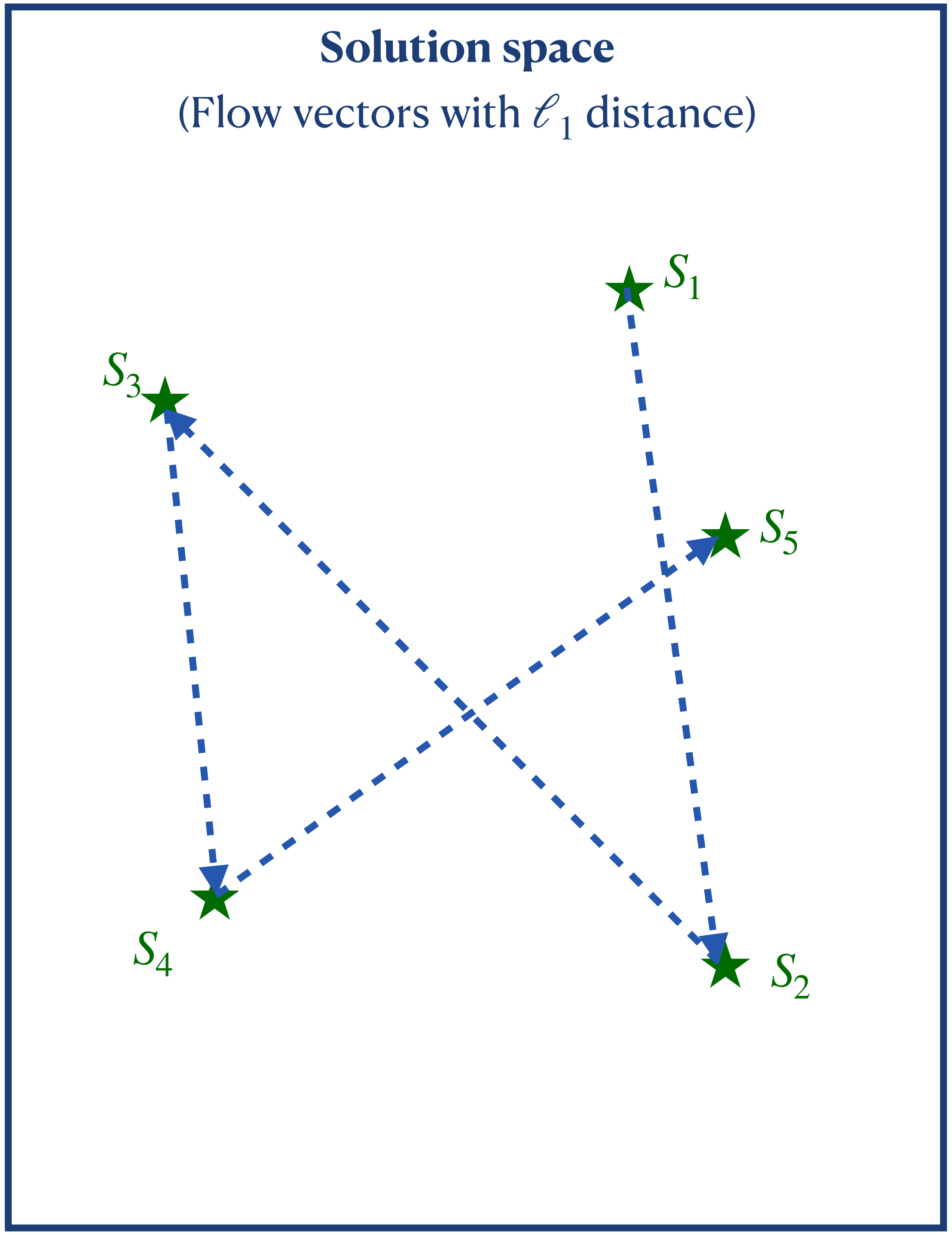
- "Yesterday's solution as today's prediction" is competitive with the best fixed prediction in hindsight! [Khodak Balcan Talwalkar Vassilvitskii '22]



Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

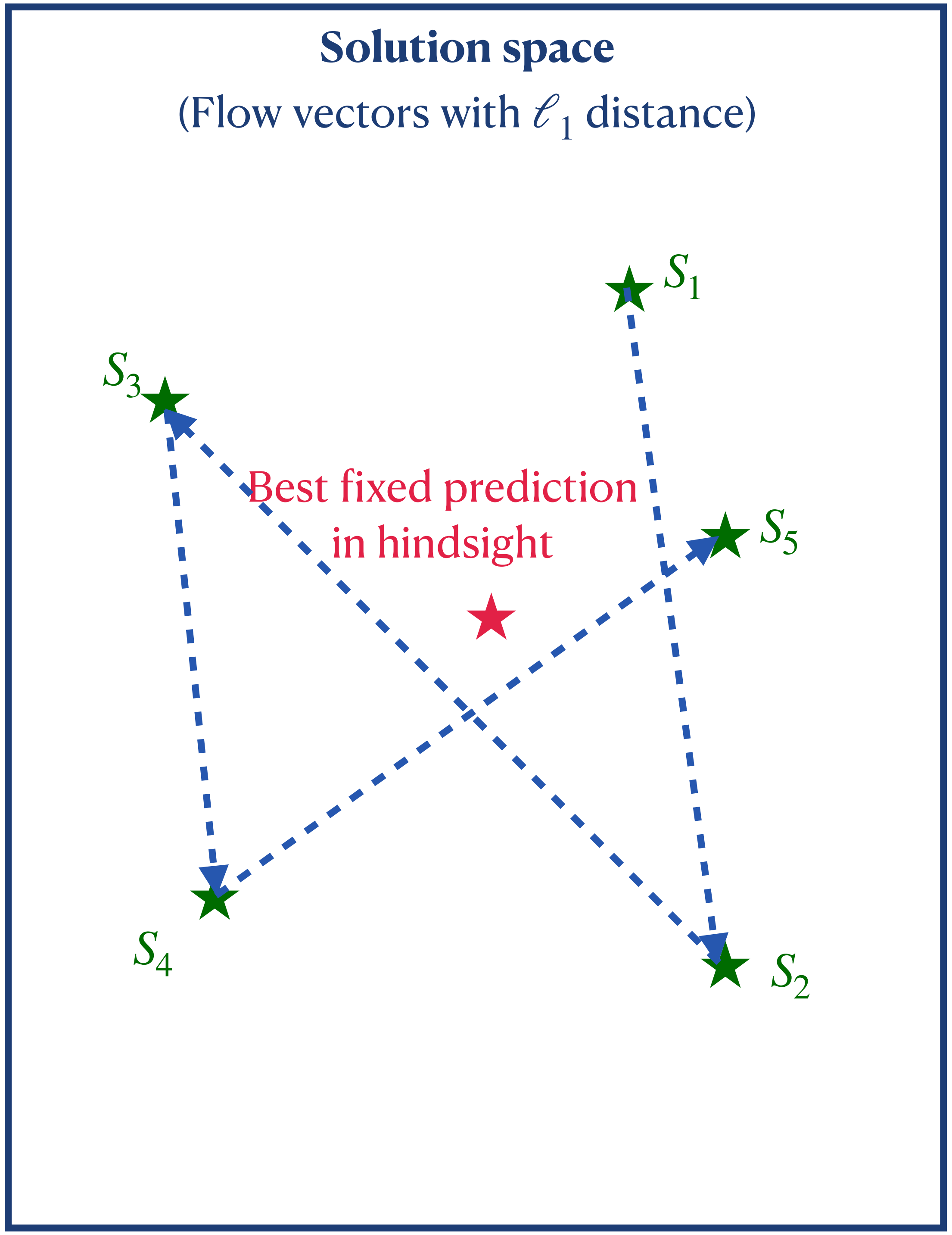
- "Yesterday's solution as today's prediction" is competitive with the best fixed prediction in hindsight! [Khodak Balcan Talwalkar Vassilvitskii '22]



Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

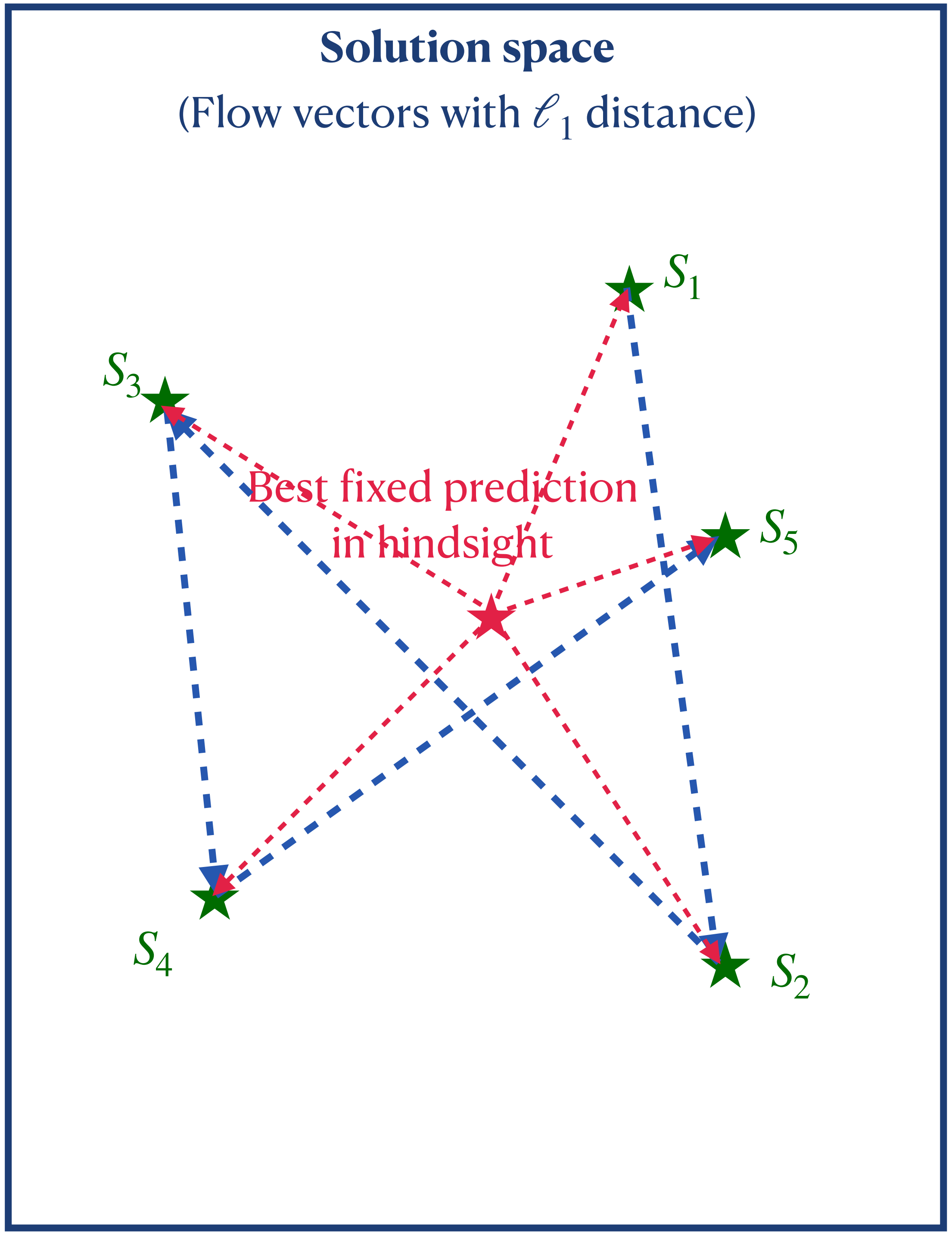
- "Yesterday's solution as today's prediction" is competitive with the best fixed prediction in hindsight! [Khodak Balcan Talwalkar Vassilvitskii '22]



Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

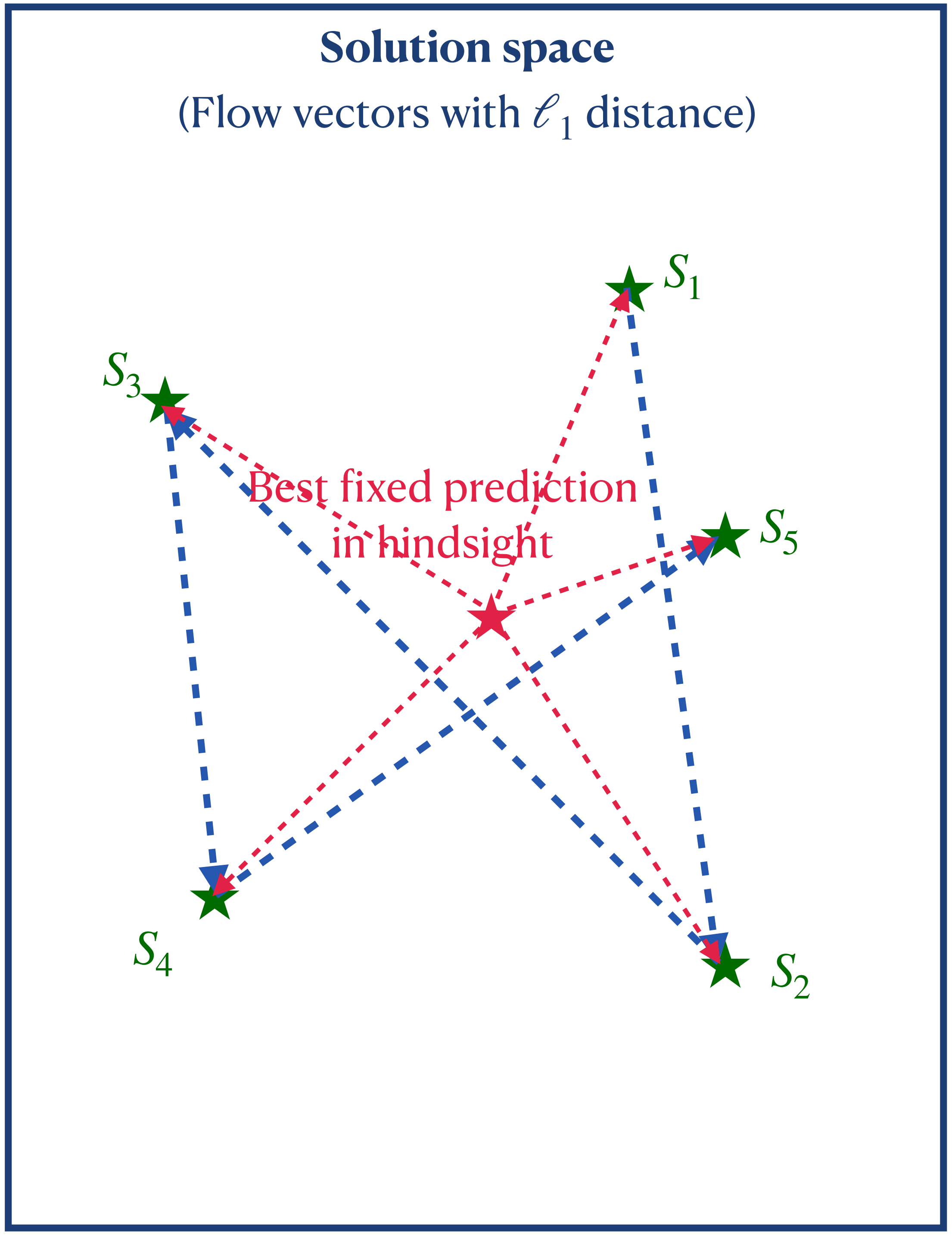
- "Yesterday's solution as today's prediction" is competitive with the best fixed prediction in hindsight! [Khodak Balcan Talwalkar Vassilvitskii '22]



Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

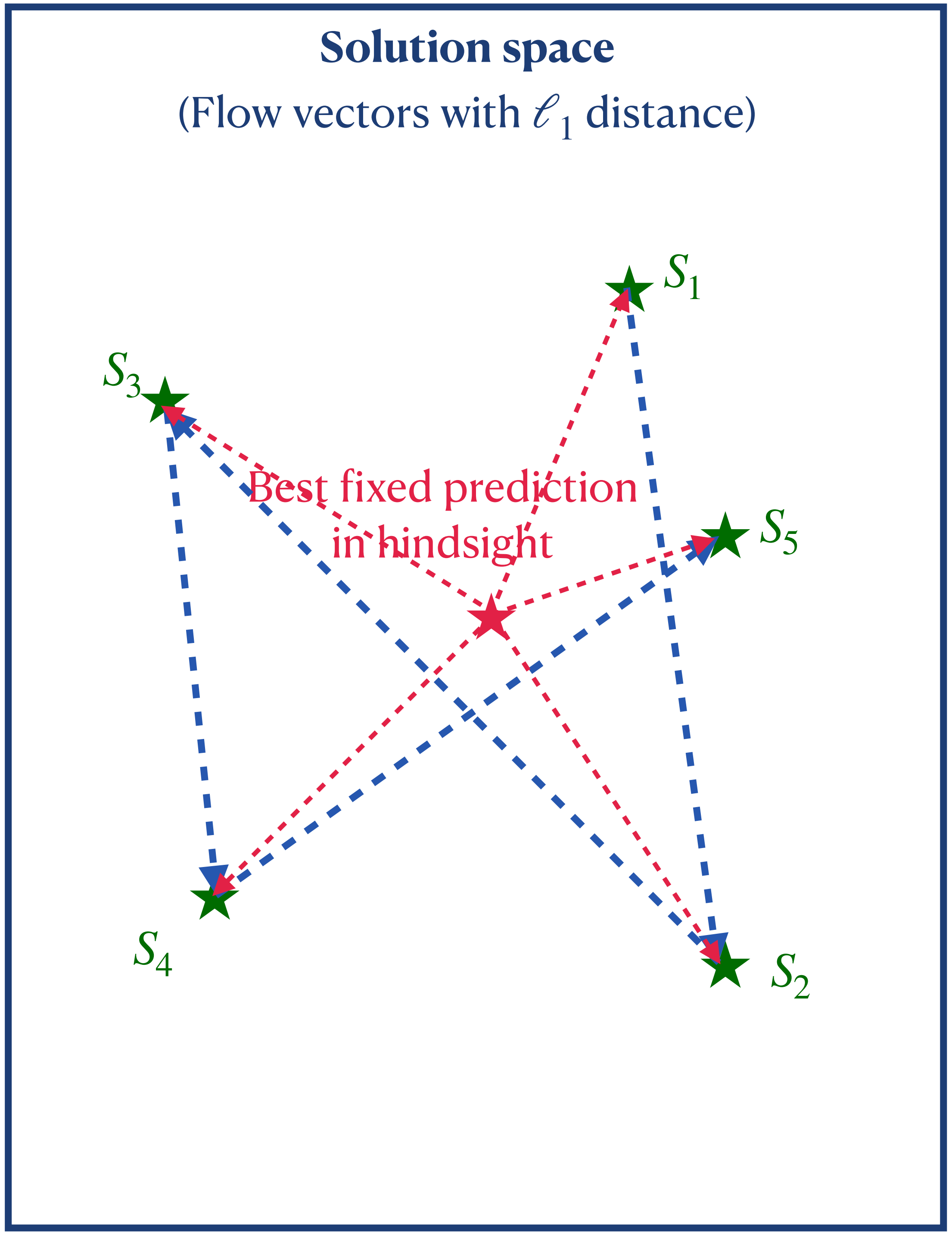
- "Yesterday's solution as today's prediction" is competitive with the best fixed prediction in hindsight! [Khodak Balcan Talwalkar Vassilvitskii '22]
  - Natural baseline from online algorithms literature



Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

- "Yesterday's solution as today's prediction" is competitive with the best fixed prediction in hindsight! [Khodak Balcan Talwalkar Vassilvitskii '22]
    - Natural baseline from online algorithms literature
    - When solutions in sequence form a tight **cluster**, faster than solving instances one at a time

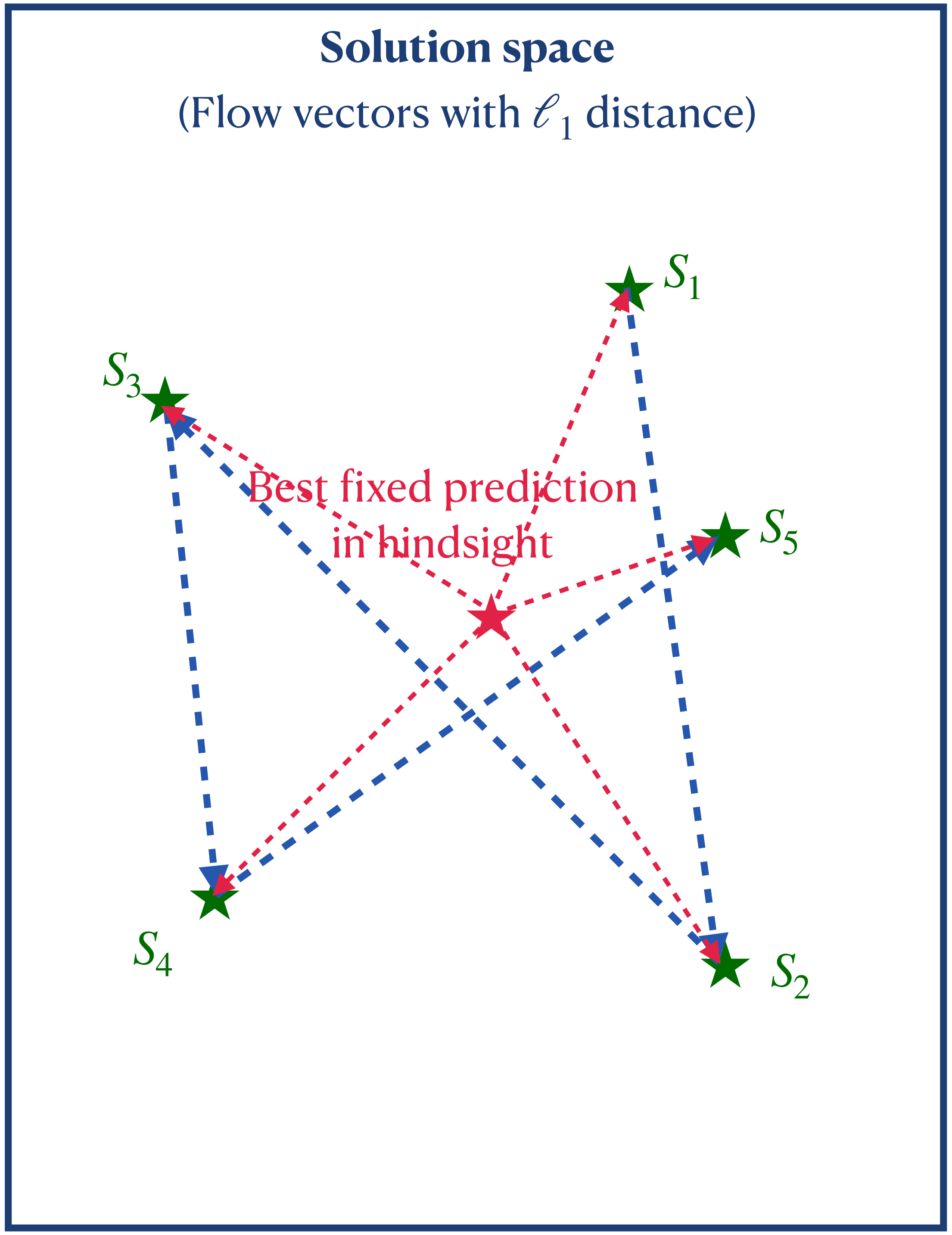


Theoretical Interpretation

Is "yesterday's solution as today's prediction" theoretically principled?

- "Yesterday's solution as today's prediction" is competitive with the best fixed prediction in hindsight! [Khodak Balcan Talwalkar Vassilvitskii '22]
  - Natural baseline from online algorithms literature
  - When solutions in sequence form a tight **cluster**, faster than solving instances one at a time

Can we do even better?



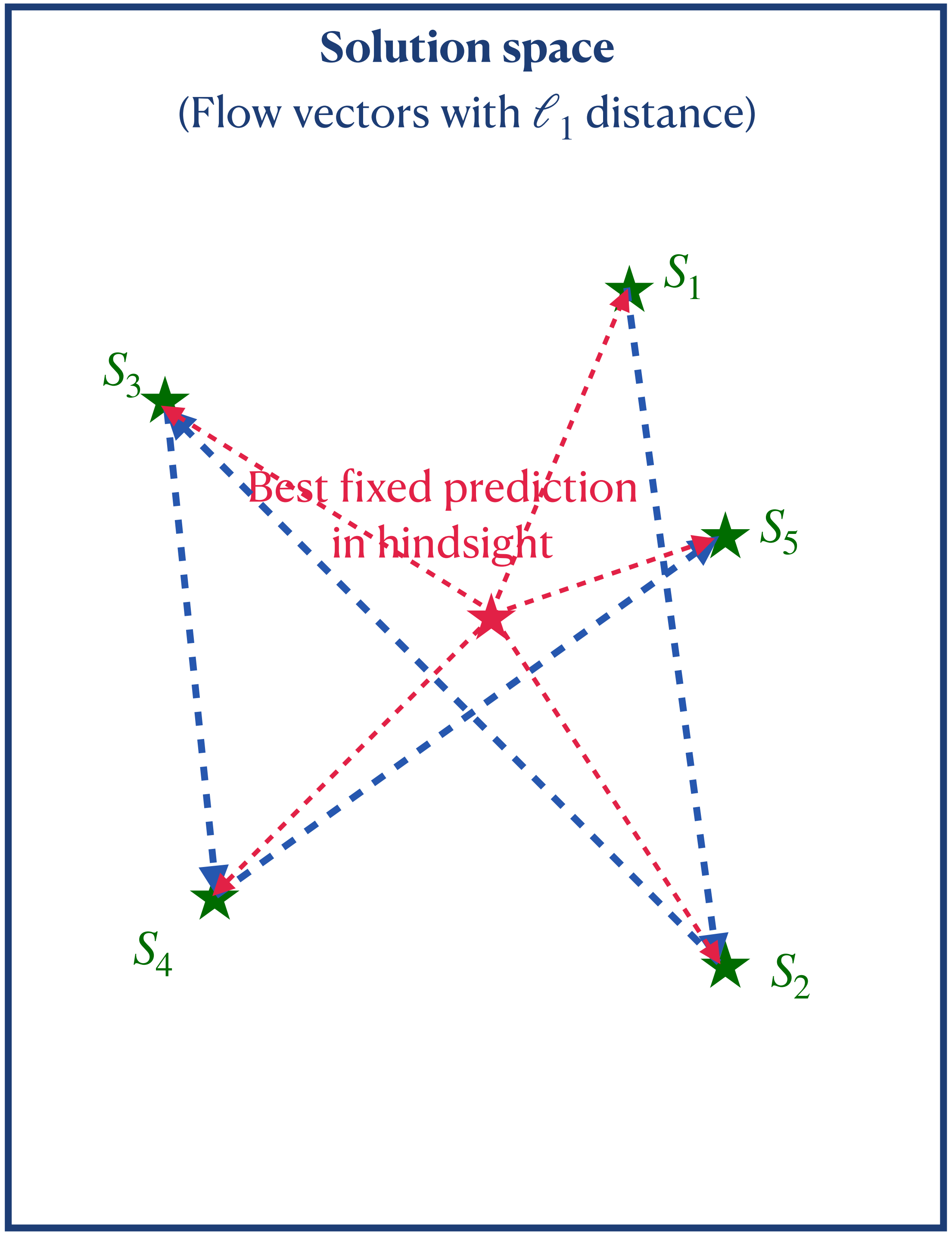
Theoretical Interpretation

Is “yesterday’s solution as today’s prediction” theoretically principled?

- “Yesterday’s solution as today’s prediction” is competitive with the best fixed prediction in hindsight! [Khodak Balcan Talwalkar Vassilvitskii ’22]
    - Natural baseline from online algorithms literature
    - When solutions in sequence form a tight **cluster**, faster than solving instances one at a time

Can we do even better?

- Yes! Can design algorithms to take advantage of other forms of structure (multiple clusters) and compete against adaptive baselines [Blum Srinivas ’25]



It's not just max-flow!

It's not just max-flow!

Warm start algorithms (with Learning-Augmented formalism) studied for:

It's not just max-flow!

Warm start algorithms (with Learning-Augmented formalism) studied for:

- Ford-Fulkerson [Polak Zub '22][Davies Moseley Vassilvitskii Wang '23]

It's not just max-flow!

Warm start algorithms (with Learning-Augmented formalism) studied for:

- Ford-Fulkerson [Polak Zub '22][Davies Moseley Vassilvitskii Wang '23]
- Bipartite matching [Dinitz Im Lavastida Moseley Vassilvitskii '21][Chen Silwal Vakilian Zhang '22]

It's not just max-flow!

Warm start algorithms (with Learning-Augmented formalism) studied for:

- Ford-Fulkerson [Polak Zub '22][Davies Moseley Vassilvitskii Wang '23]
- Bipartite matching [Dinitz Im Lavastida Moseley Vassilvitskii '21][Chen Silwal Vakilian Zhang '22]
- Max-Flow via Push-Relabel [Davies Vassilvitskii Wang '24]

It's not just max-flow!

Warm start algorithms (with Learning-Augmented formalism) studied for:

- Ford-Fulkerson [Polak Zub '22][Davies Moseley Vassilvitskii Wang '23]
- Bipartite matching [Dinitz Im Lavastida Moseley Vassilvitskii '21][Chen Silwal Vakilian Zhang '22]
- Max-Flow via Push-Relabel [Davies Vassilvitskii Wang '24]

Can hope to prove for other **local search** algorithms, main new contribution is dealing with infeasibility

It's not just max-flow!

Warm start algorithms (with Learning-Augmented formalism) studied for:

- Ford-Fulkerson [Polak Zub '22][Davies Moseley Vassilvitskii Wang '23]
- Bipartite matching [Dinitz Im Lavastida Moseley Vassilvitskii '21][Chen Silwal Vakilian Zhang '22]
- Max-Flow via Push-Relabel [Davies Vassilvitskii Wang '24]

Can hope to prove for other **local search** algorithms, main new contribution is dealing with infeasibility

In practice, warm starts are used for many optimization problems!

It’s not just max-flow!

Warm start algorithms (with Learning-Augmented formalism) studied for:

- Ford-Fulkerson [Polak Zub ’22][Davies Moseley Vassilvitskii Wang ’23]
- Bipartite matching [Dinitz Im Lavastida Moseley Vassilvitskii ’21][Chen Silwal Vakilian Zhang ’22]
- Max-Flow via Push-Relabel [Davies Vassilvitskii Wang ’24]

Can hope to prove for other **local search** algorithms, main new contribution is dealing with infeasibility

In practice, warm starts are used for many optimization problems!

Takeaways: Theoretical formalism of Learning-Augmented algorithms allows us to

It's not just max-flow!

Warm start algorithms (with Learning-Augmented formalism) studied for:

- Ford-Fulkerson [Polak Zub '22][Davies Moseley Vassilvitskii Wang '23]
- Bipartite matching [Dinitz Im Lavastida Moseley Vassilvitskii '21][Chen Silwal Vakilian Zhang '22]
- Max-Flow via Push-Relabel [Davies Vassilvitskii Wang '24]

Can hope to prove for other **local search** algorithms, main new contribution is dealing with infeasibility

In practice, warm starts are used for many optimization problems!

Takeaways: Theoretical formalism of Learning-Augmented algorithms allows us to

- Analyze and compare strategies for solving sequences of related instances

It’s not just max-flow!

Warm start algorithms (with Learning-Augmented formalism) studied for:

- Ford-Fulkerson [Polak Zub ’22][Davies Moseley Vassilvitskii Wang ’23]
- Bipartite matching [Dinitz Im Lavastida Moseley Vassilvitskii ’21][Chen Silwal Vakilian Zhang ’22]
- Max-Flow via Push-Relabel [Davies Vassilvitskii Wang ’24]

Can hope to prove for other **local search** algorithms, main new contribution is dealing with infeasibility

In practice, warm starts are used for many optimization problems!

Takeaways: Theoretical formalism of Learning-Augmented algorithms allows us to

- Analyze and compare strategies for solving sequences of related instances
- See new algorithmic opportunities

It's not just max-flow!

Warm start algorithms (with Learning-Augmented formalism) studied for:

- Ford-Fulkerson [Polak Zub '22][Davies Moseley Vassilvitskii Wang '23]
- Bipartite matching [Dinitz Im Lavastida Moseley Vassilvitskii '21][Chen Silwal Vakilian Zhang '22]
- Max-Flow via Push-Relabel [Davies Vassilvitskii Wang '24]

Can hope to prove for other **local search** algorithms, main new contribution is dealing with infeasibility

In practice, warm starts are used for many optimization problems!

Takeaways: Theoretical formalism of Learning-Augmented algorithms allows us to

- Analyze and compare strategies for solving sequences of related instances
- See new algorithmic opportunities
- **Modularize** (warm start vs. meta problem)

References

(1) Repeated Computations: Sequences of related instances of a problem can be solved faster than one at a time

Learning-augmented warm start algorithms:

- [Davies Moseley Vassilvitskii Wang '23][Polak Zub '22] Max-flow via Ford-Fulkerson
- [Dinitz Im Lavastida Moseley Vassilvitskii '21][Chen Silwal Vakilian Zhang '22] Primal dual bipartite matching
- [Davies Vassilvitskii Wang '24] Max-flow via Push Relabel

Prediction strategies for warm start algorithms:

- [Khodak Balcan Talwalkar Vassilvitskii '22] Compete with best fixed prediction in hindsight
- [Blum Srinivas '25] Take advantage of weaker structure in sequences

Three ideas

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

(3) **Randomized Algorithms:** Randomized algorithms and data structures can be hedged to take advantage of extra information by incorporating a prior

Dynamic Algorithms

Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

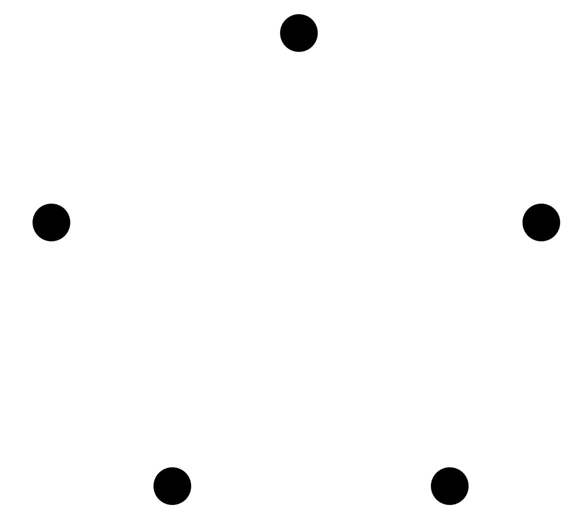
- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)

Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)

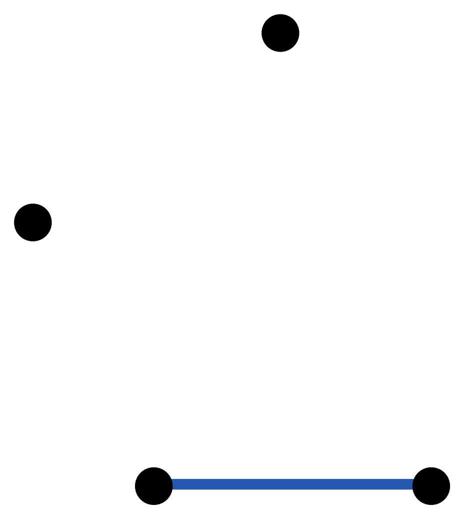


Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)

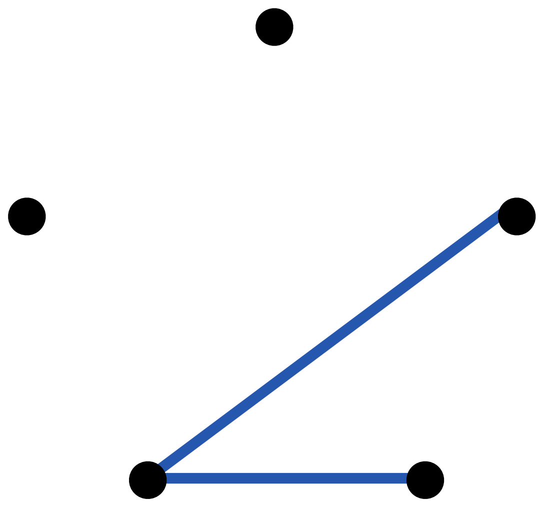


Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)

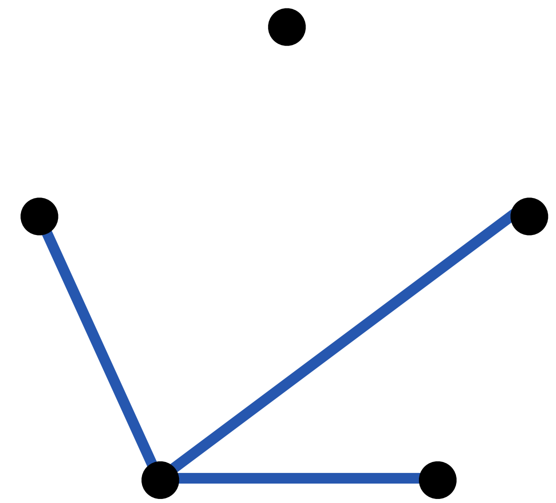


Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)

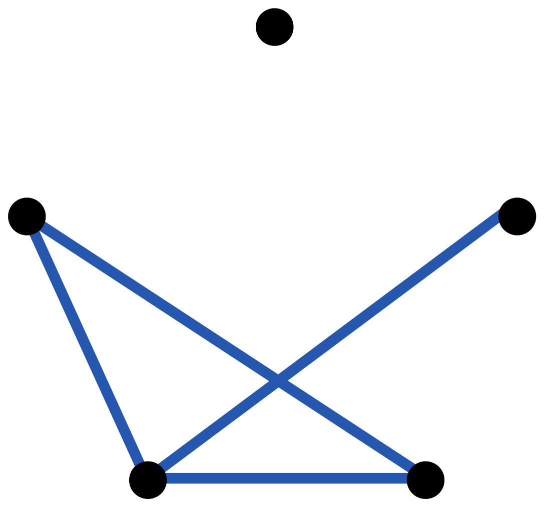


Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)

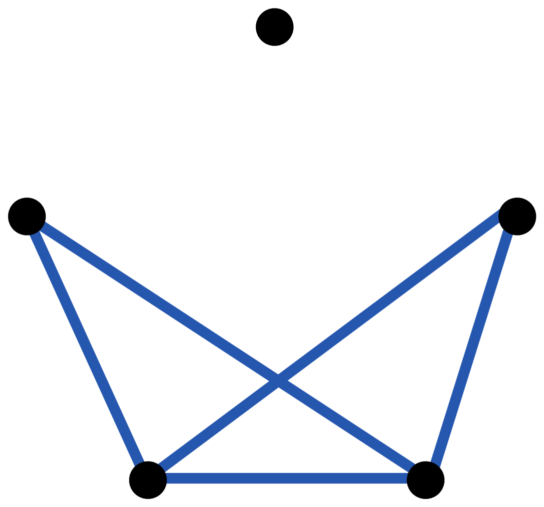


Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)

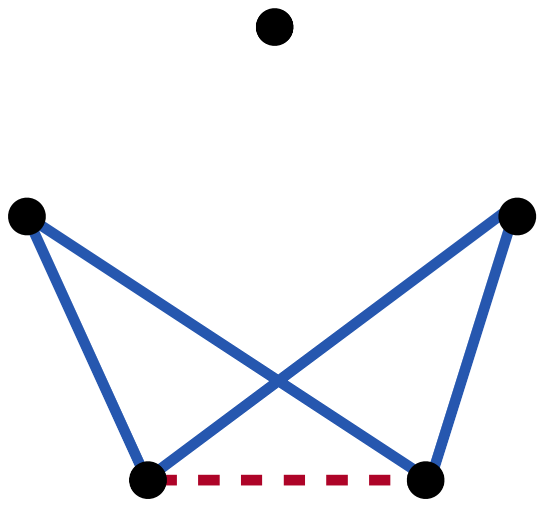


Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)

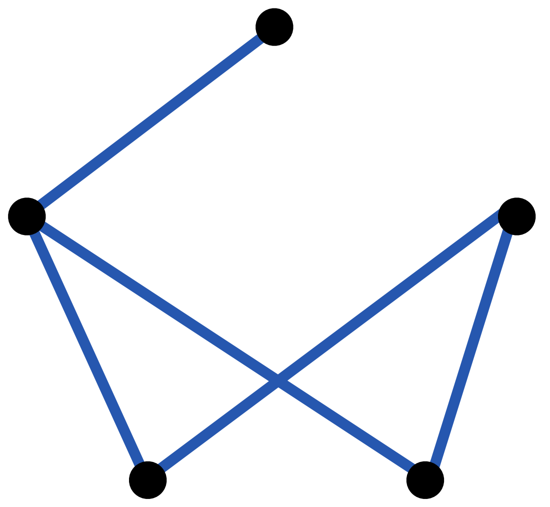


Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)

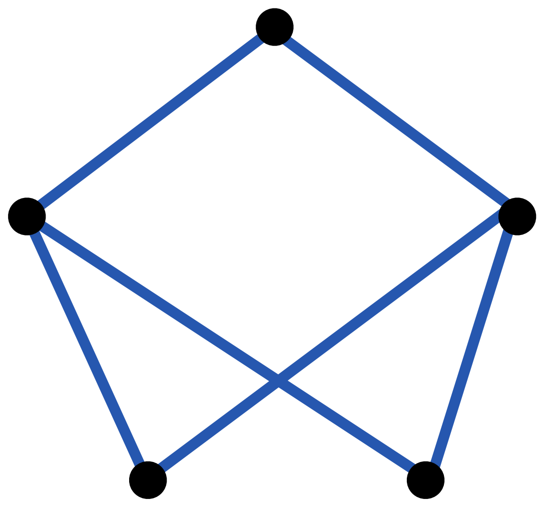


Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)

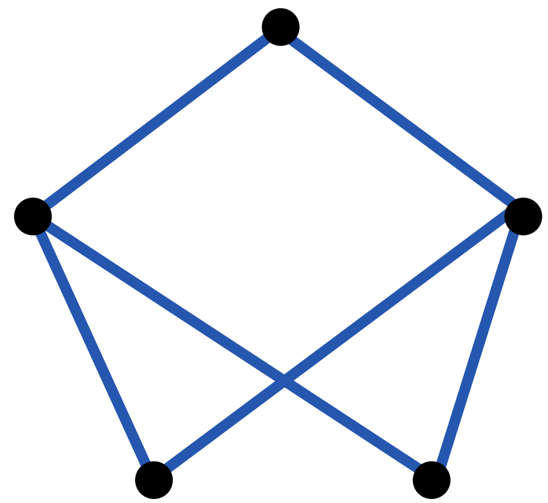


Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)
- Maintain data structure to efficiently query whether pairs of vertices are 3-vertex connected

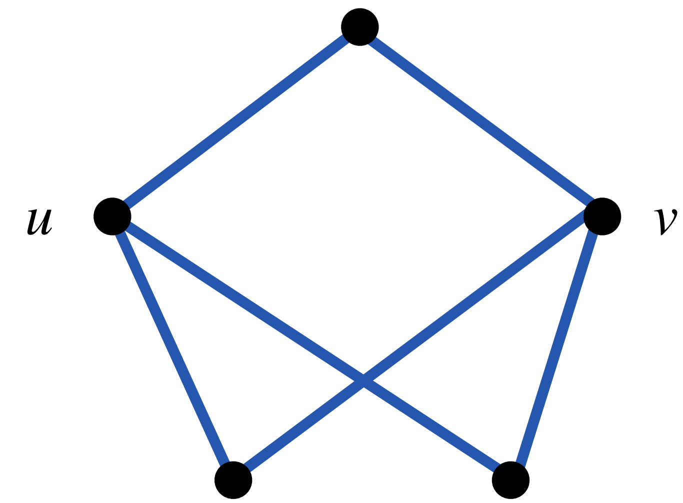


Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)
- Maintain data structure to efficiently query whether pairs of vertices are 3-vertex connected

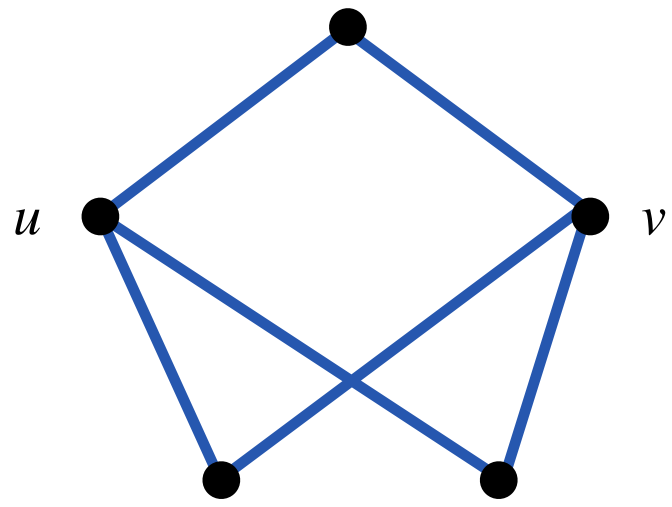


Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)
- Maintain data structure to efficiently query whether pairs of vertices are 3-vertex connected
- Minimize **amortized** time per update/query

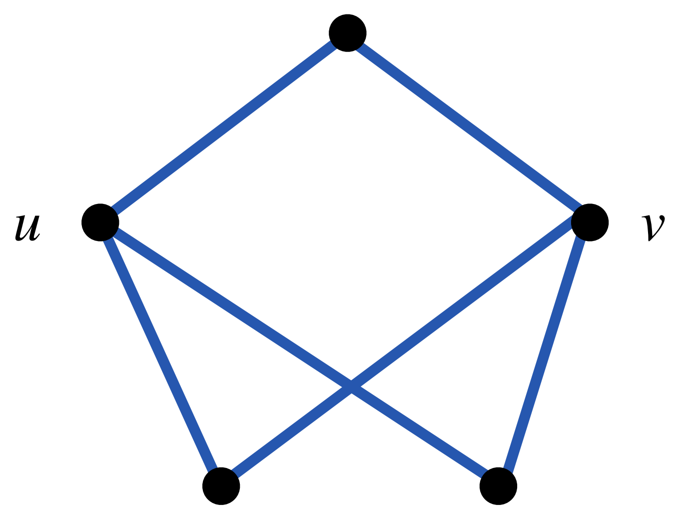


Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)
- Maintain data structure to efficiently query whether pairs of vertices are 3-vertex connected
- Minimize **amortized** time per update/query



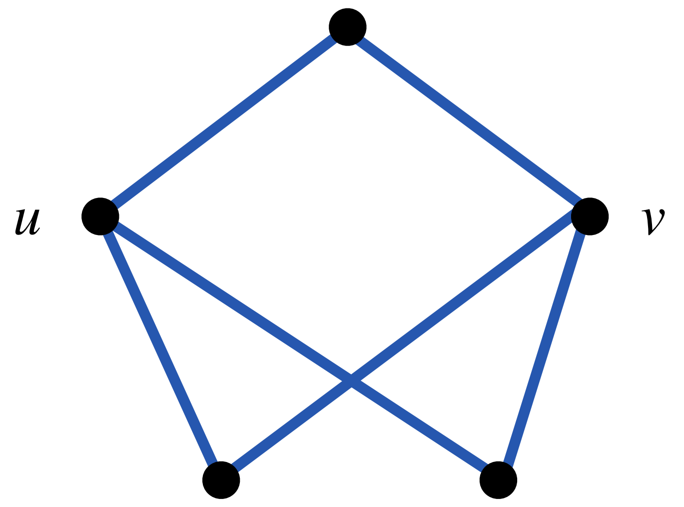
Challenge:

Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)
- Maintain data structure to efficiently query whether pairs of vertices are 3-vertex connected
- Minimize **amortized** time per update/query



Challenge:

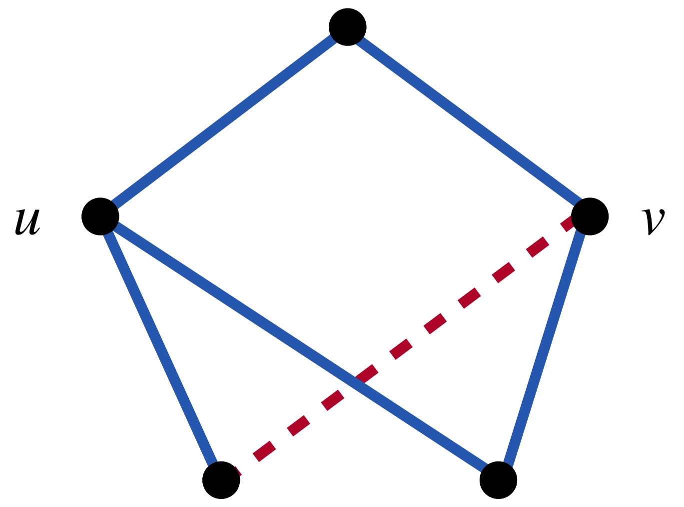
- Instance only changes a little, but solution can change a lot!

Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs

Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)
- Maintain data structure to efficiently query whether pairs of vertices are 3-vertex connected
- Minimize **amortized** time per update/query

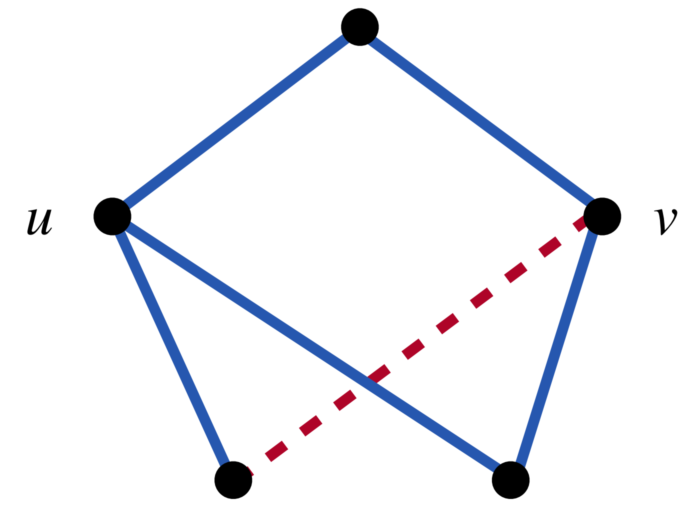


Challenge:

- Instance only changes a little, but solution can change a lot!

Dynamic Algorithms

Example: 3-vertex connectivity (triconnectivity) in dynamic graphs



Dynamic model:

- Graph on n vertices undergoes edge insertions and deletions (fully-dynamic)
- Maintain data structure to efficiently query whether pairs of vertices are 3-vertex connected
- Minimize **amortized** time per update/query

Challenge:

- Instance only changes a little, but solution can change a lot!
- Design data structures to "reuse" work to solve subproblems

Example: Triconnectivity

Example: Triconnectivity

Best known fully dynamic algorithm for triconnectivity:

$O(n^{2/3})$ (worst-case) update time [Galil Italiano Sarnak '99]

Example: Triconnectivity

Best known fully dynamic algorithm for triconnectivity:

$O(n^{2/3})$ (worst-case) update time [Galil Italiano Sarnak '99]

Baseline: better than recomputing solution from scratch every day

Example: Triconnectivity

Best known fully dynamic algorithm for triconnectivity:

$O(n^{2/3})$ (worst-case) update time [Galil Italiano Sarnak '99]

Baseline: better than recomputing solution from scratch every day

Good: update time sublinear in graph size

Example: Triconnectivity

Best known fully dynamic algorithm for triconnectivity:

$O(n^{2/3})$ (worst-case) update time [Galil Italiano Sarnak '99]

Baseline: better than recomputing solution from scratch every day

Good: update time sublinear in graph size

Gold standard: polylog(n) update time (exponential improvement over baseline)

Example: Triconnectivity

Best known fully dynamic algorithm for triconnectivity:

$O(n^{2/3})$ (worst-case) update time [Galil Italiano Sarnak '99]

Baseline: better than recomputing solution from scratch every day

Good: update time sublinear in graph size

Gold standard: polylog(n) update time (exponential improvement over baseline)

Compare to "offline dynamic" setting:

Example: Triconnectivity

Best known fully dynamic algorithm for triconnectivity:

$O(n^{2/3})$ (worst-case) update time [Galil Italiano Sarnak '99]

Baseline: better than recomputing solution from scratch every day

Good: update time sublinear in graph size

Gold standard: polylog(n) update time (exponential improvement over baseline)

Compare to "offline dynamic" setting:

- Sequence of updates and queries are given in one batch

Example: Triconnectivity

Best known fully dynamic algorithm for triconnectivity:

$O(n^{2/3})$ (worst-case) update time [Galil Italiano Sarnak '99]

Baseline: better than recomputing solution from scratch every day

Good: update time sublinear in graph size

Gold standard: polylog(n) update time (exponential improvement over baseline)

Compare to "offline dynamic" setting:

- Sequence of updates and queries are given in one batch
- Algorithm returns answers to all queries at once

Example: Triconnectivity

Best known fully dynamic algorithm for triconnectivity:

$O(n^{2/3})$ (worst-case) update time [Galil Italiano Sarnak '99]

Baseline: better than recomputing solution from scratch every day

Good: update time sublinear in graph size

Gold standard: polylog(n) update time (exponential improvement over baseline)

Compare to "offline dynamic" setting:

- Sequence of updates and queries are given in one batch
- Algorithm returns answers to all queries at once

Best known **offline dynamic** algorithm for triconnectivity:

polylog(n) (amortized) update time [Peng Sandlund Sleator '17]

with slick divide-and-conquer algorithm!

Example: Triconnectivity

Best known fully dynamic algorithm for triconnectivity:

$O(n^{2/3})$ (worst-case) update time [Galil Italiano Sarnak '99]

Baseline: better than recomputing solution from scratch every day

Good: update time sublinear in graph size

Gold standard: polylog(n) update time (exponential improvement over baseline)

Compare to "offline dynamic" setting:

- Sequence of updates and queries are given in one batch
- Algorithm returns answers to all queries at once

Best known **offline dynamic** algorithm for triconnectivity:

polylog(n) (amortized) update time [Peng Sandlund Sleator '17]

with slick divide-and-conquer algorithm!

Knowledge of future updates lets us reuse computation more efficiently!

Example: Triconnectivity

Best known fully dynamic algorithm for triconnectivity:

$O(n^{2/3})$ (worst-case) update time [Galil Italiano Sarnak '99]

Baseline: better than recomputing solution from scratch every day

Good: update time sublinear in graph size

Gold standard: polylog(n) update time (exponential improvement over baseline)

Compare to "offline dynamic" setting:

- Sequence of updates and queries are given in one batch
- Algorithm returns answers to all queries at once

Best known **offline dynamic** algorithm for triconnectivity:

polylog(n) (amortized) update time [Peng Sandlund Sleator '17]

with slick divide-and-conquer algorithm!

Knowledge of future updates lets us reuse computation more efficiently!

Gap between fully-dynamic and offline dynamic exists for many problems

Imperfect Information

Imperfect Information

Fully Dynamic Model

No information about future updates

Imperfect Information

Fully Dynamic Model
No information about future updates

Offline Dynamic Model
Perfect information about future updates

Imperfect Information

Fully Dynamic Model
No information about future updates

Offline Dynamic Model
Perfect information about future updates

Fully dynamic is **pessimistic**, but offline dynamic is too **optimistic**...

Imperfect Information

Fully Dynamic Model
No information about future updates

Offline Dynamic Model
Perfect information about future updates

Fully dynamic is **pessimistic**, but offline dynamic is too **optimistic**...

Can we instead use an **imperfect prediction** of future updates?

Imperfect Information

Fully Dynamic Model
No information about future updates

Offline Dynamic Model
Perfect information about future updates

Fully dynamic is **pessimistic**, but offline dynamic is too **optimistic**...

Can we instead use an **imperfect prediction** of future updates?

Learning-Augmented approach [Liu Srinivas '24]**:**

Imperfect Information

Fully Dynamic Model
No information about future updates

Offline Dynamic Model
Perfect information about future updates

Fully dynamic is **pessimistic**, but offline dynamic is too **optimistic**...

Can we instead use an **imperfect prediction** of future updates?

Learning-Augmented approach [Liu Srinivas '24]**:**

- Ask for a (potentially infeasible) prediction of the input sequence

Imperfect Information

Fully Dynamic Model
No information about future updates

Offline Dynamic Model
Perfect information about future updates

Fully dynamic is **pessimistic**, but offline dynamic is too **optimistic**…

Can we instead use an **imperfect prediction** of future updates?

Learning-Augmented approach [Liu Srinivas '24]**:**

- Ask for a (potentially infeasible) prediction of the input sequence
- Perform fully dynamic updates on the true sequence that may or may not be close to the predicted sequence

Imperfect Information

Fully Dynamic Model
No information about future updates

Offline Dynamic Model
Perfect information about future updates

Fully dynamic is **pessimistic**, but offline dynamic is too **optimistic**…

Can we instead use an **imperfect prediction** of future updates?

Learning-Augmented approach [Liu Srinivas '24]**:**

- Ask for a (potentially infeasible) prediction of the input sequence
- Perform fully dynamic updates on the true sequence that may or may not be close to the predicted sequence

Goal:

Imperfect Information

Fully Dynamic Model
No information about future updates

Offline Dynamic Model
Perfect information about future updates

Fully dynamic is **pessimistic**, but offline dynamic is too **optimistic**…

Can we instead use an **imperfect prediction** of future updates?

Learning-Augmented approach [Liu Srinivas '24]**:**

- Ask for a (potentially infeasible) prediction of the input sequence
- Perform fully dynamic updates on the true sequence that may or may not be close to the predicted sequence

Goal:

- **Consistency:** when predictions are good, recover offline dynamic performance

Imperfect Information

Fully Dynamic Model	Offline Dynamic Model
No information about future updates	Perfect information about future updates

Fully dynamic is **pessimistic**, but offline dynamic is too **optimistic**...

Can we instead use an **imperfect prediction** of future updates?

Learning-Augmented approach [Liu Srinivas '24]**:**

- Ask for a (potentially infeasible) prediction of the input sequence
- Perform fully dynamic updates on the true sequence that may or may not be close to the predicted sequence

Goal:

- **Consistency:** when predictions are good, recover offline dynamic performance
- **Robustness:** always do at least as well as fully dynamic performance

Imperfect Information

Fully Dynamic Model
No information about future updates

Offline Dynamic Model
Perfect information about future updates

Fully dynamic is **pessimistic**, but offline dynamic is too **optimistic**...

Can we instead use an **imperfect prediction** of future updates?

Learning-Augmented approach [Liu Srinivas '24]**:**

- Ask for a (potentially infeasible) prediction of the input sequence
- Perform fully dynamic updates on the true sequence that may or may not be close to the predicted sequence

Goal:

- **Consistency:** when predictions are good, recover offline dynamic performance
- **Robustness:** always do at least as well as fully dynamic performance
- **Graceful degradation:** smooth tradeoff in error of prediction

Imperfect Information

Fully Dynamic Model	Predicted-Updates Model	Offline Dynamic Model
No information about future updates	Imperfect information about future updates	Perfect information about future updates

Fully dynamic is **pessimistic**, but offline dynamic is too **optimistic**...

Can we instead use an **imperfect prediction** of future updates?

Learning-Augmented approach [Liu Srinivas '24]**:**

- Ask for a (potentially infeasible) prediction of the input sequence
- Perform fully dynamic updates on the true sequence that may or may not be close to the predicted sequence

Goal:

- **Consistency:** when predictions are good, recover offline dynamic performance
- **Robustness:** always do at least as well as fully dynamic performance
- **Graceful degradation:** smooth tradeoff in error of prediction

Under the Hood of Offline Dynamic Algorithms

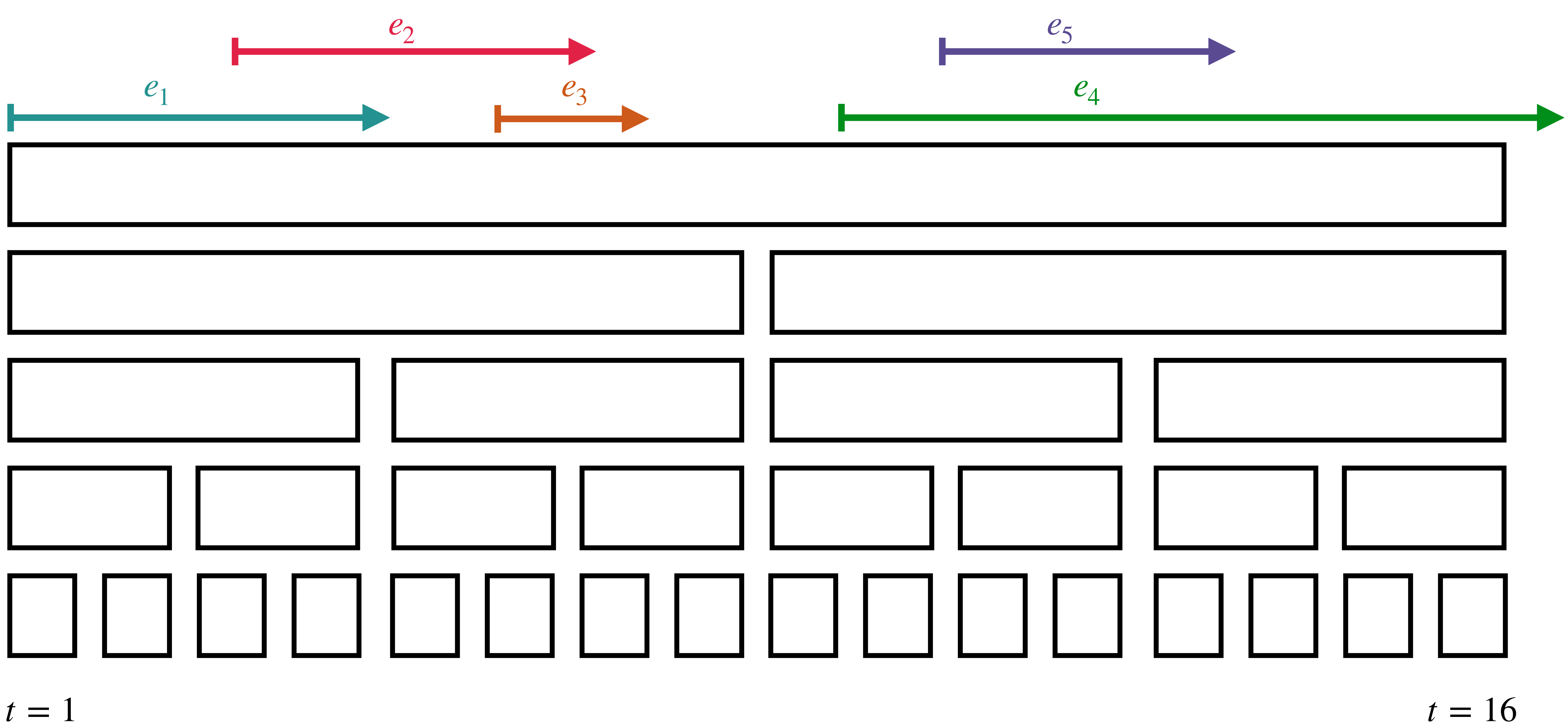
Under the Hood of Offline Dynamic Algorithms

Under the Hood of Offline Dynamic Algorithms

Divide-and-conquer approach:

Under the Hood of Offline Dynamic Algorithms

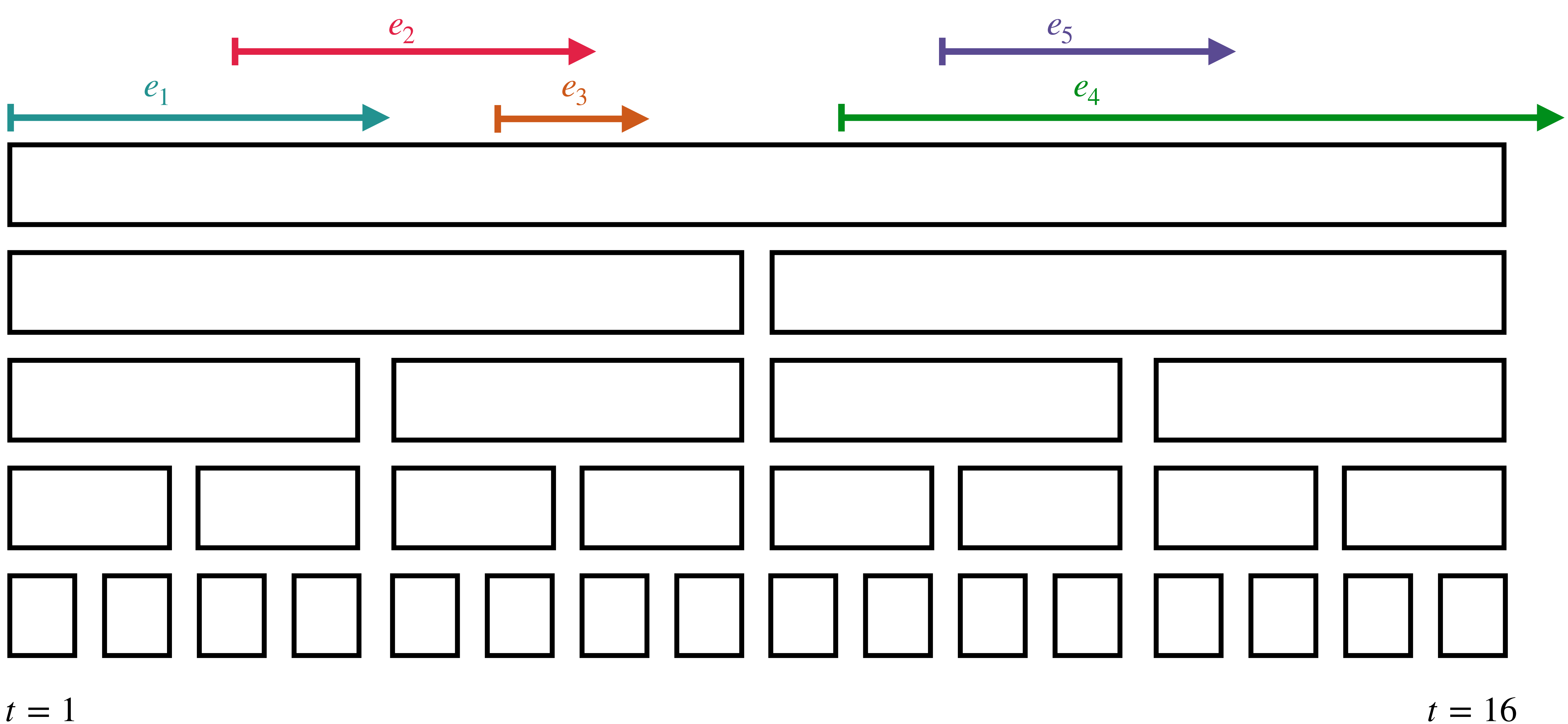
Divide-and-conquer approach:



Under the Hood of Offline Dynamic Algorithms

Divide-and-conquer approach:

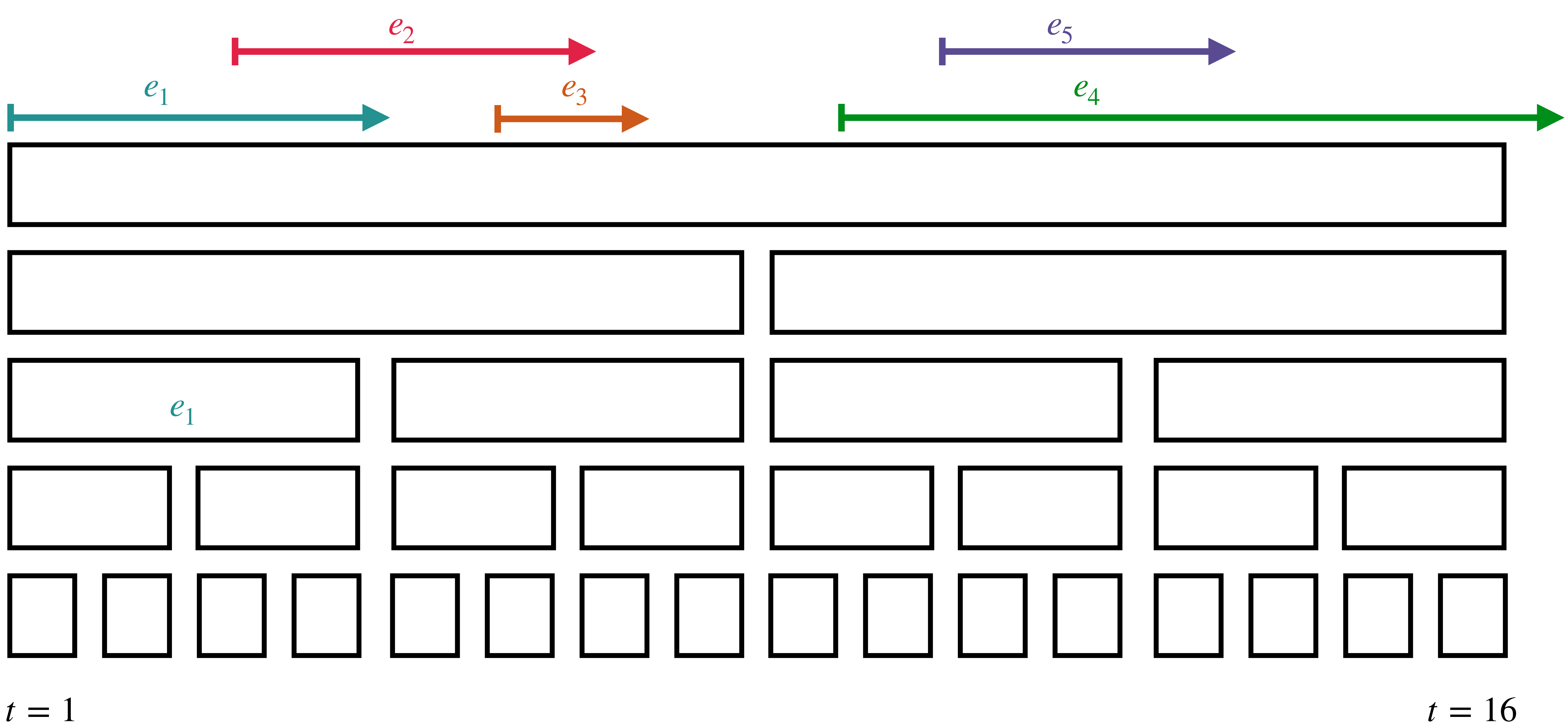
- Associate each edge with "windows" for which it is **permanent**



Under the Hood of Offline Dynamic Algorithms

Divide-and-conquer approach:

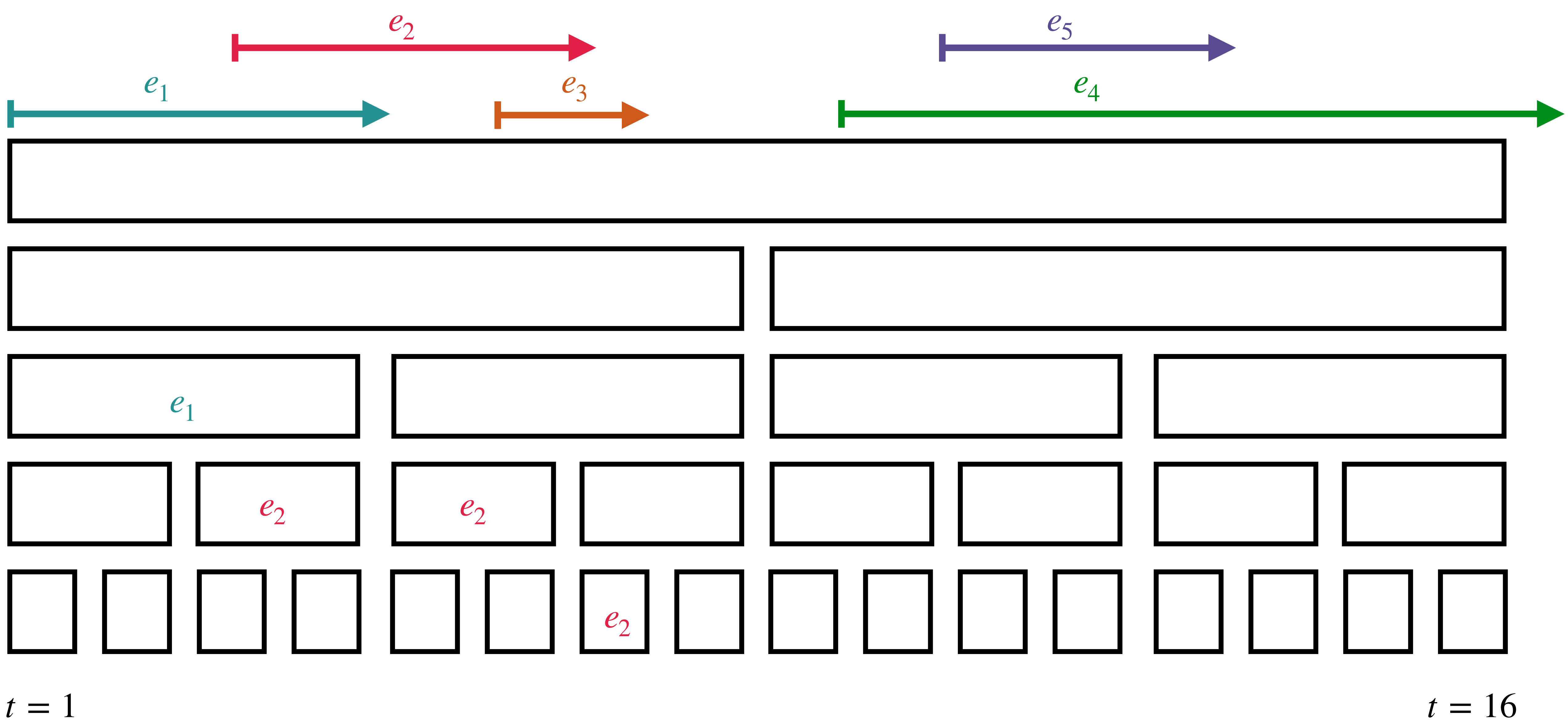
- Associate each edge with "windows" for which it is **permanent**



Under the Hood of Offline Dynamic Algorithms

Divide-and-conquer approach:

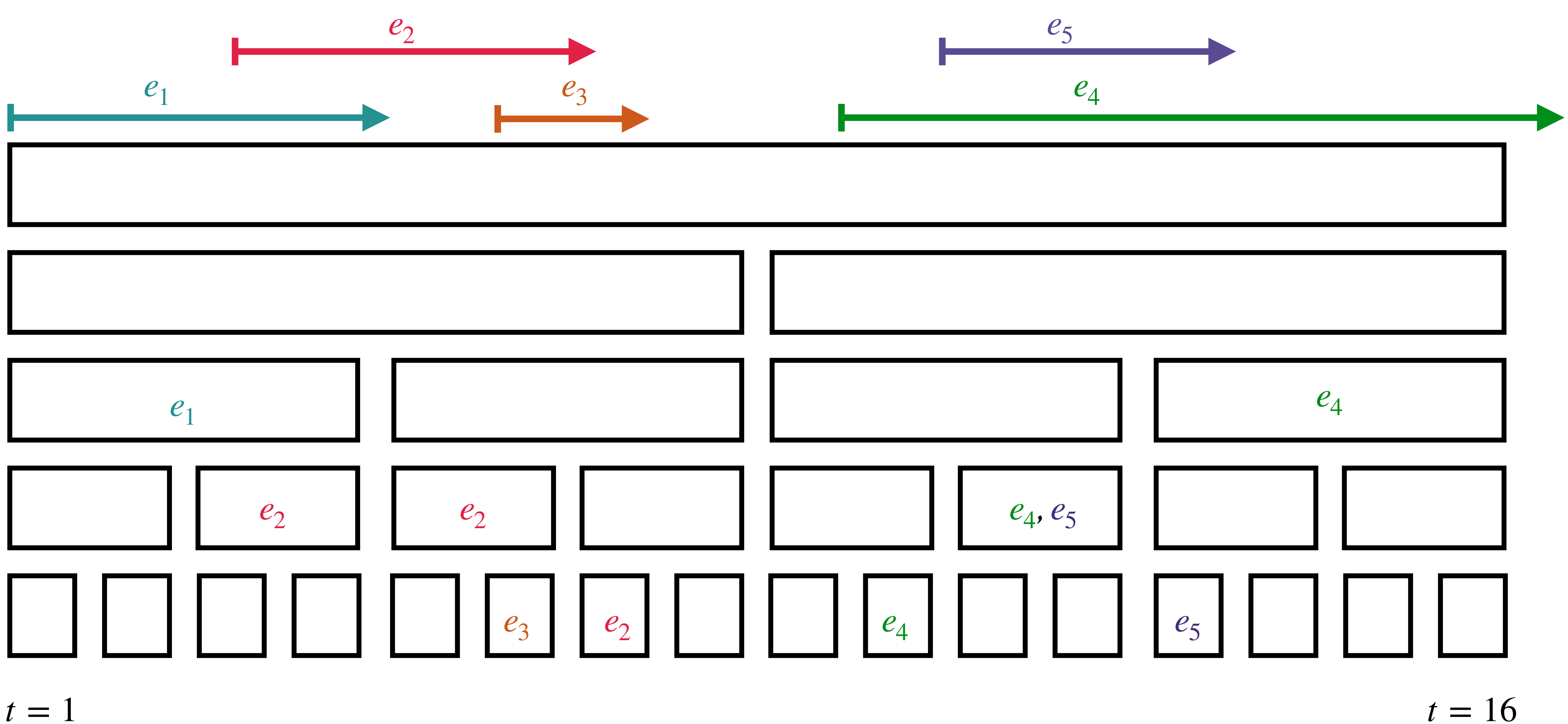
- Associate each edge with “windows” for which it is **permanent**



Under the Hood of Offline Dynamic Algorithms

Divide-and-conquer approach:

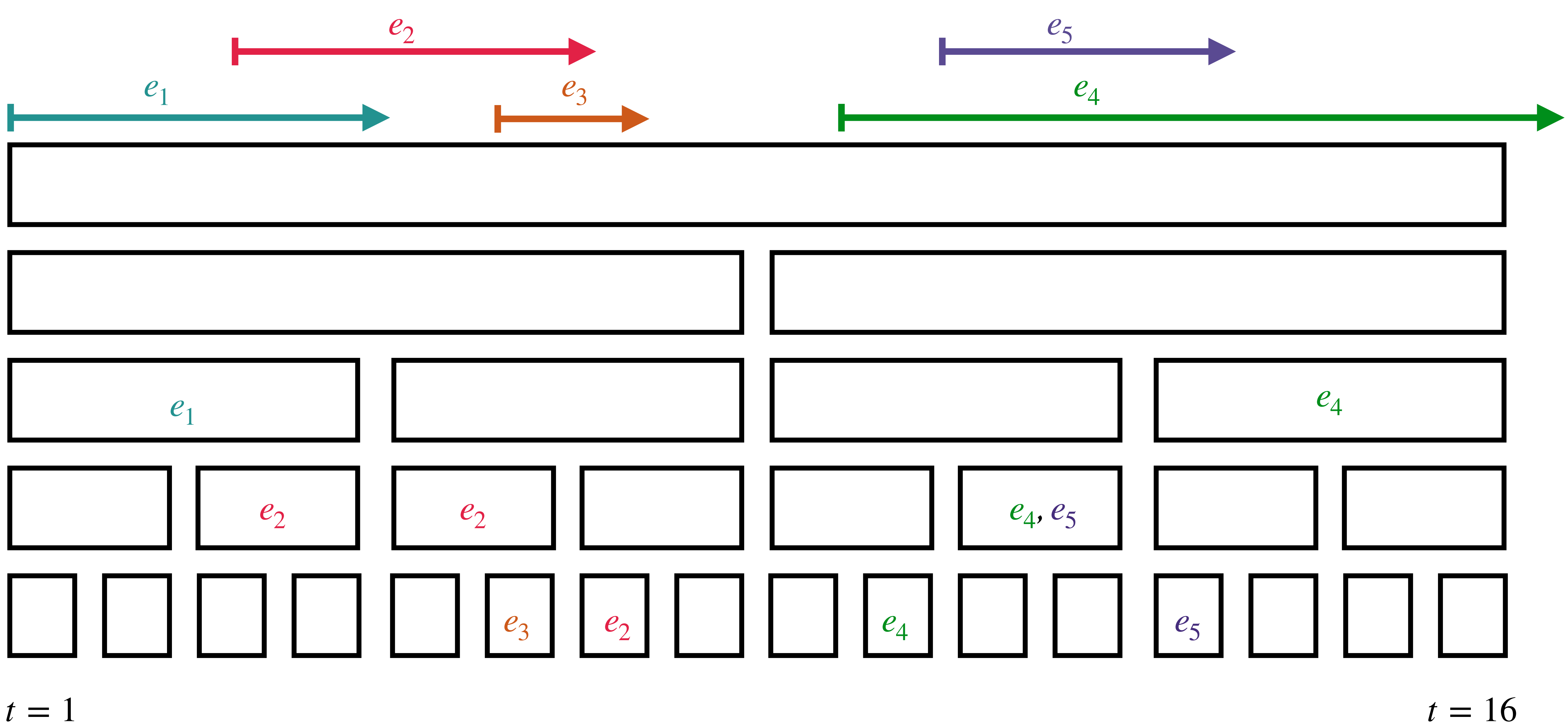
- Associate each edge with "windows" for which it is **permanent**



Under the Hood of Offline Dynamic Algorithms

Divide-and-conquer approach:

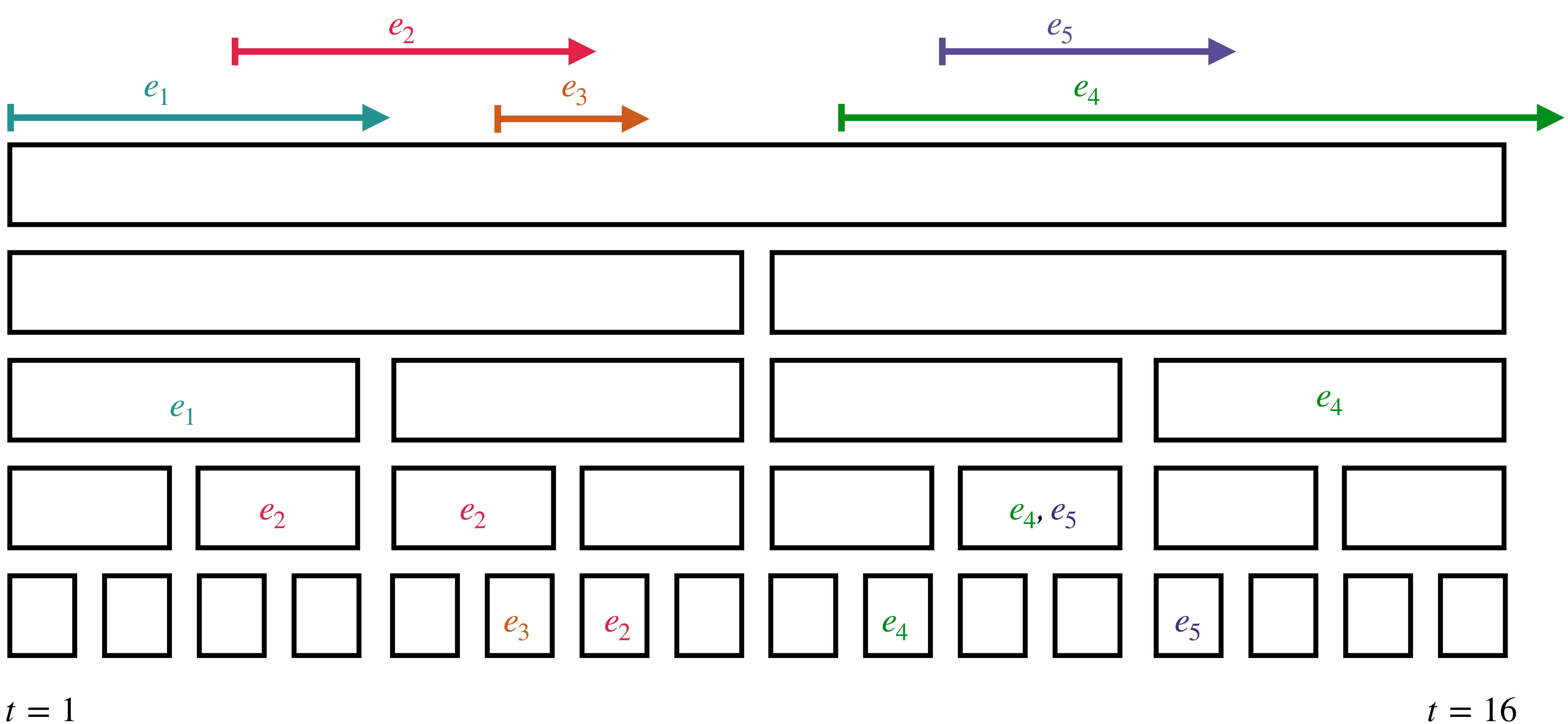
- Associate each edge with "windows" for which it is **permanent**
- Each window is associated with at most twice as many edges as its size



Under the Hood of Offline Dynamic Algorithms

Divide-and-conquer approach:

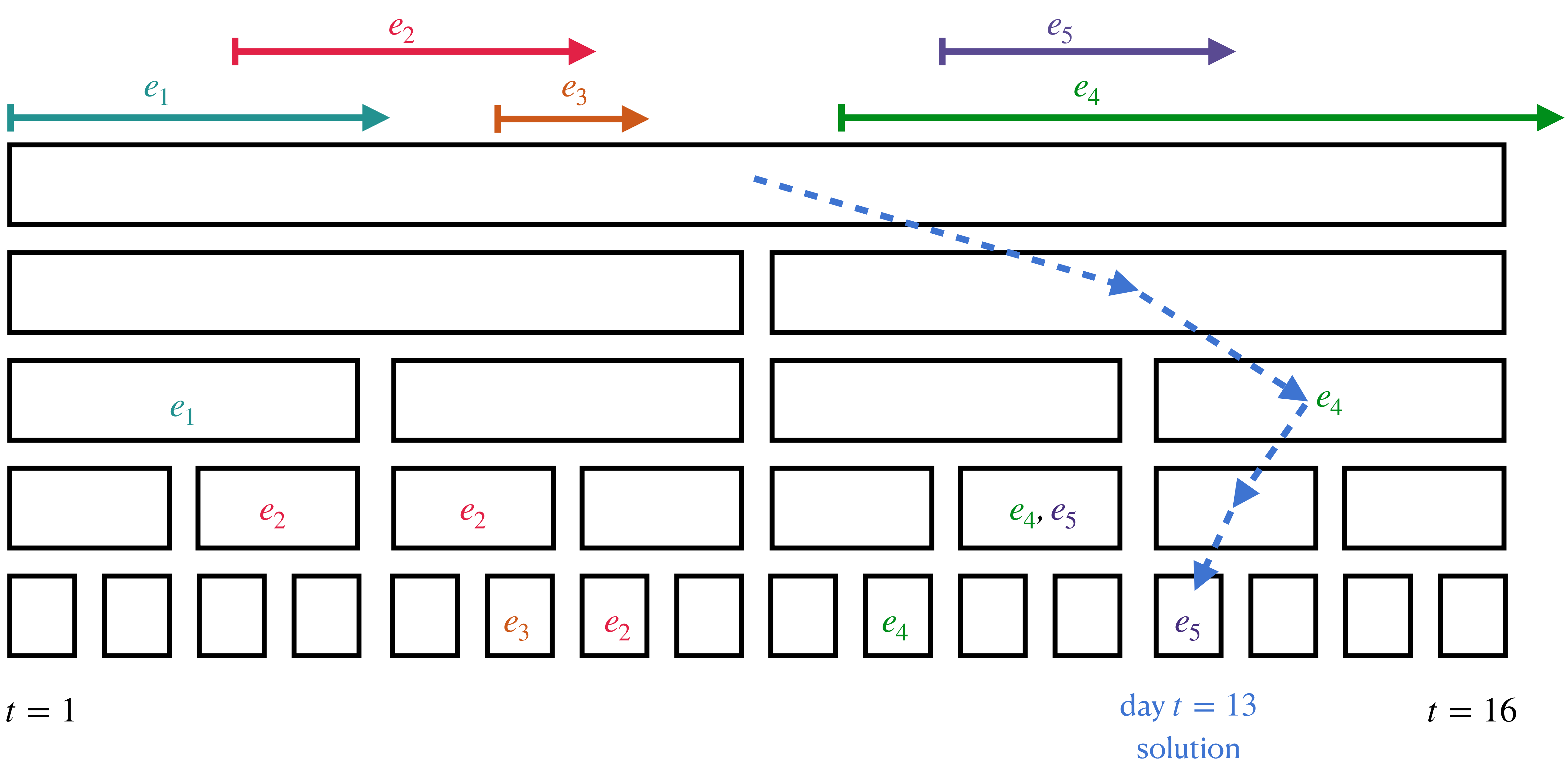
- Associate each edge with "windows" for which it is **permanent**
- Each window is associated with at most twice as many edges as its size
- Process edges in batches from root to leaf, using **sparsifier** for problem



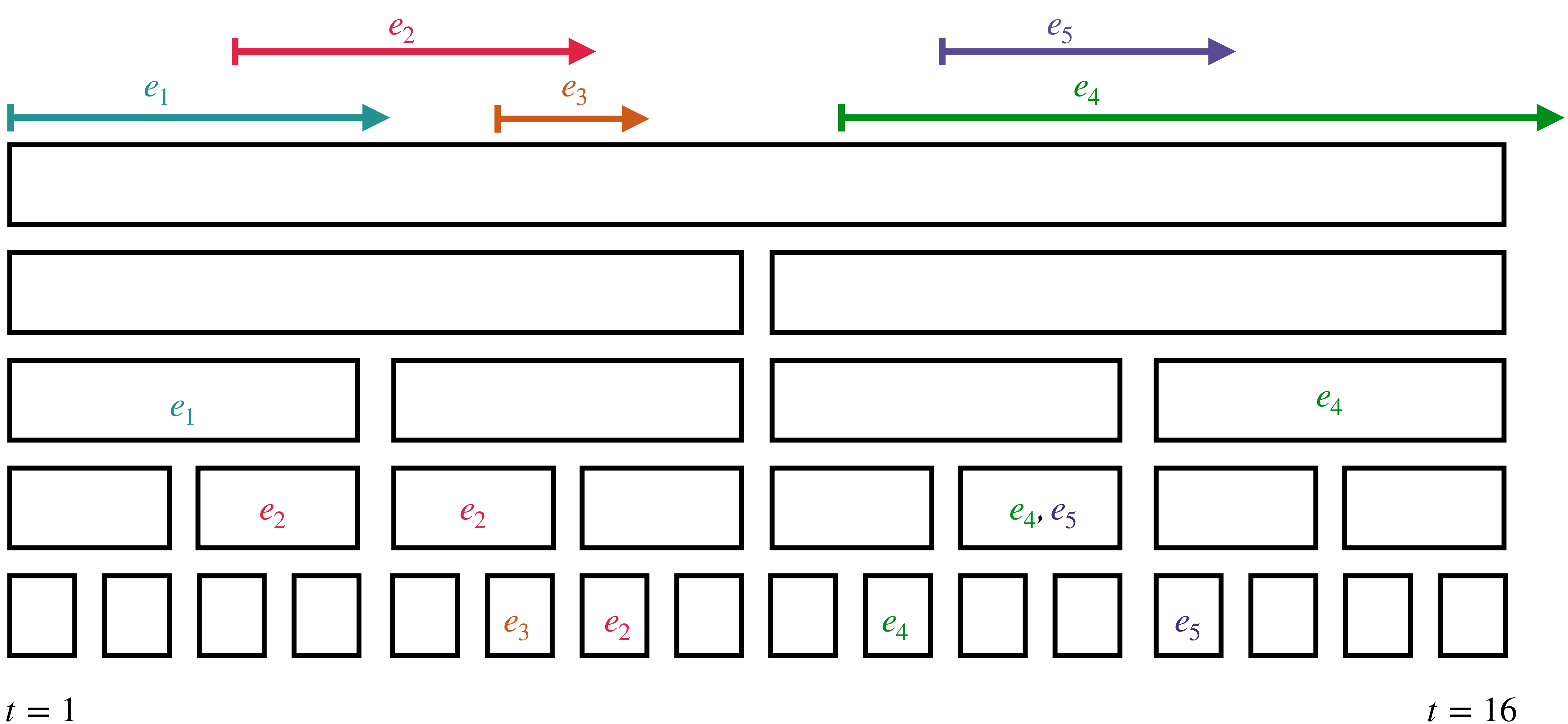
Under the Hood of Offline Dynamic Algorithms

Divide-and-conquer approach:

- Associate each edge with "windows" for which it is **permanent**
- Each window is associated with at most twice as many edges as its size
- Process edges in batches from root to leaf, using **sparsifier** for problem

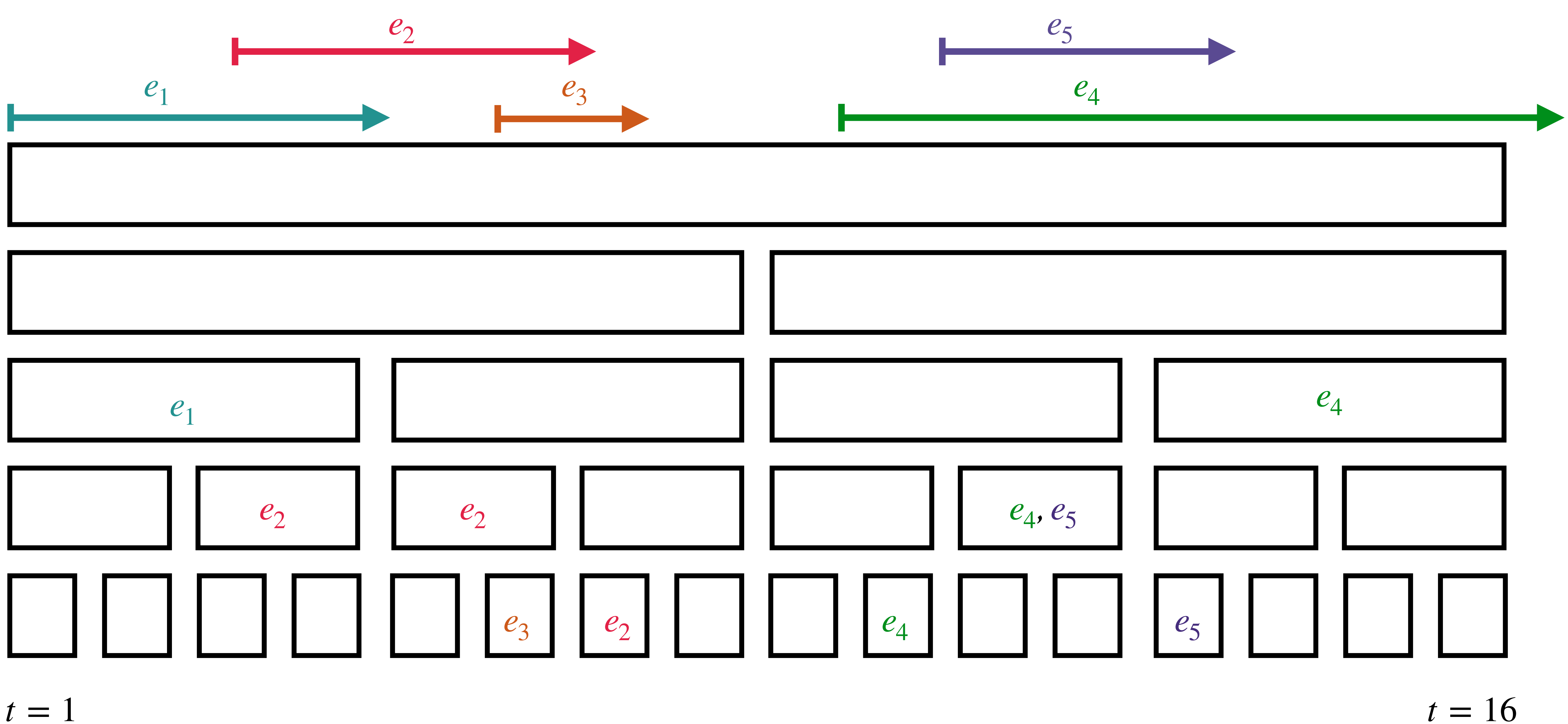


What if the prediction is wrong?



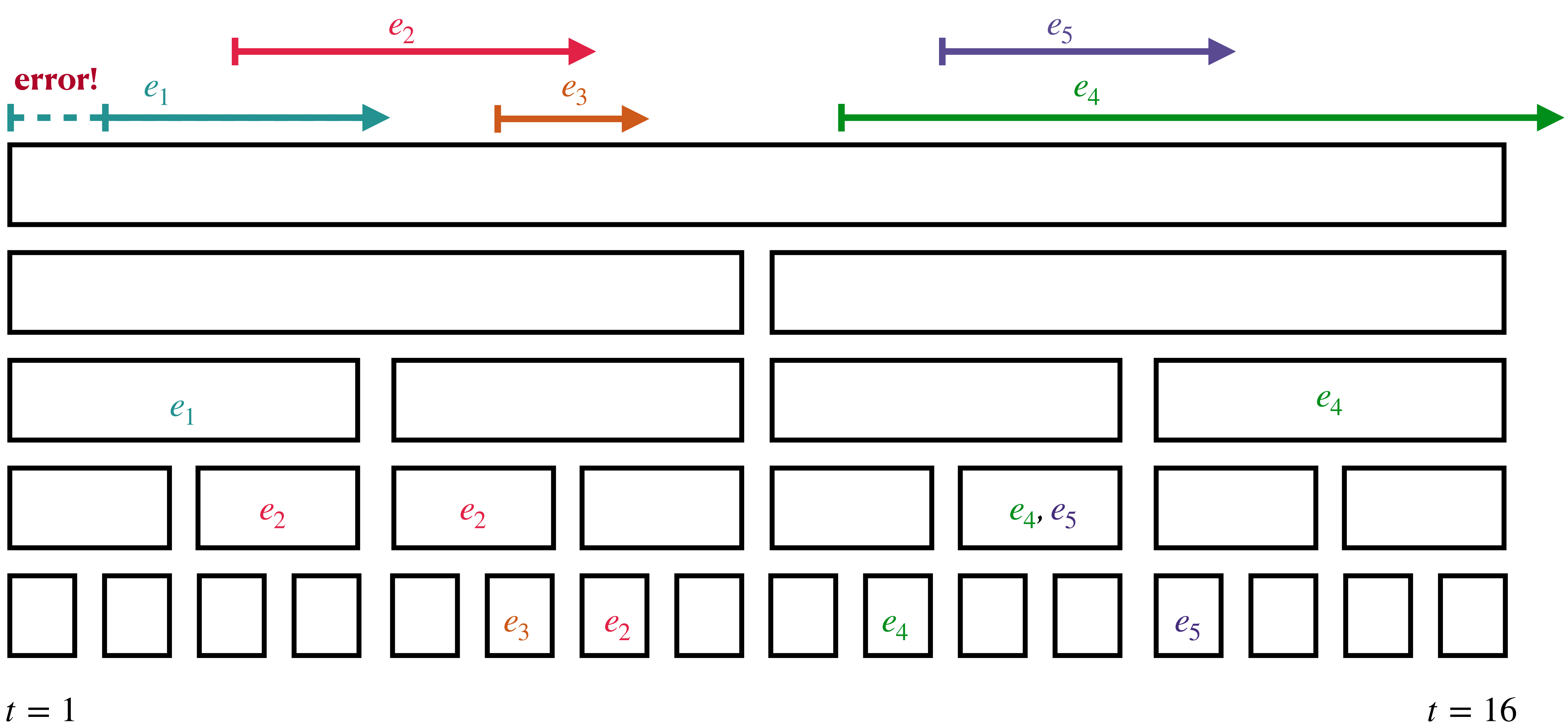
What if the prediction is wrong?

Predicted e_1 insertion on day 1, but happens on day 2 instead.



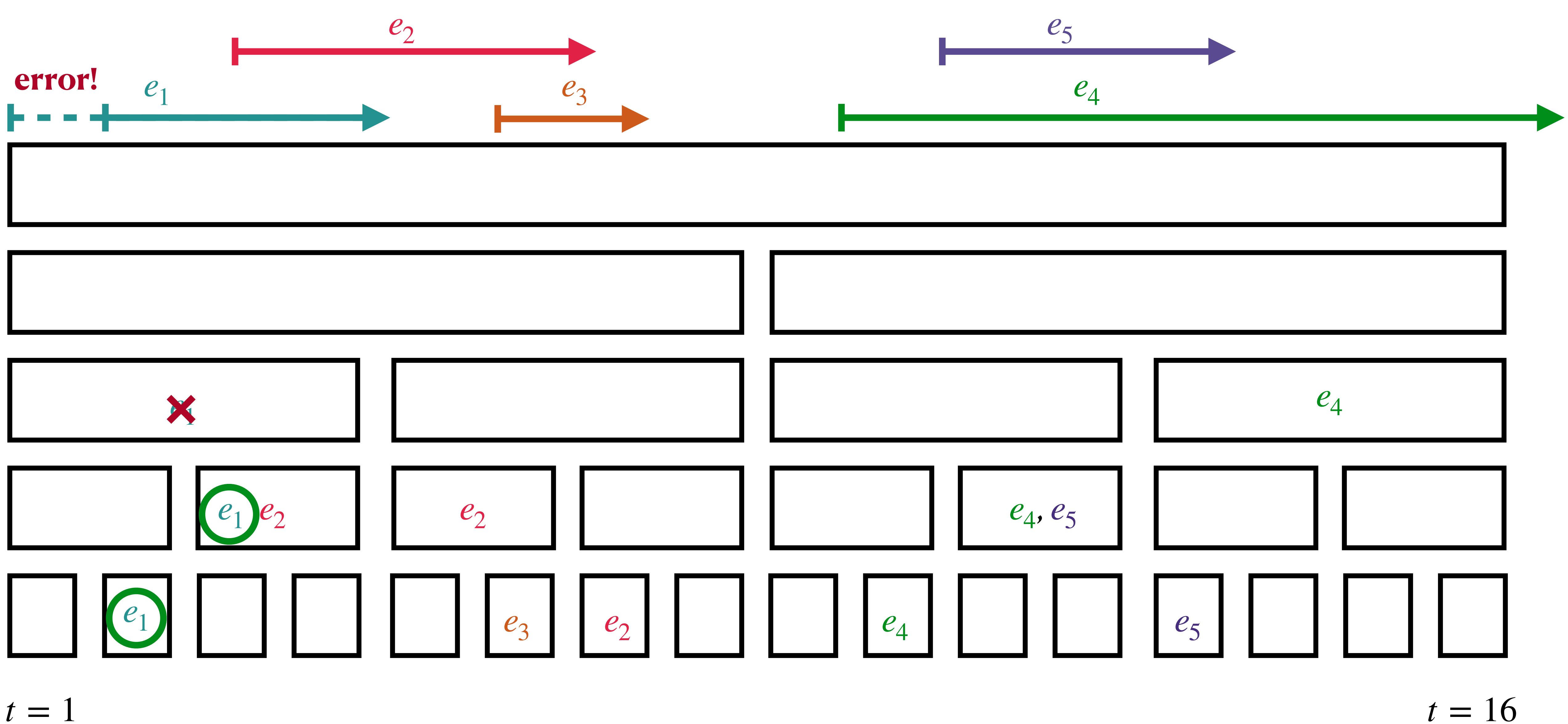
What if the prediction is wrong?

Predicted e_1 insertion on day 1, but happens on day 2 instead.



What if the prediction is wrong?

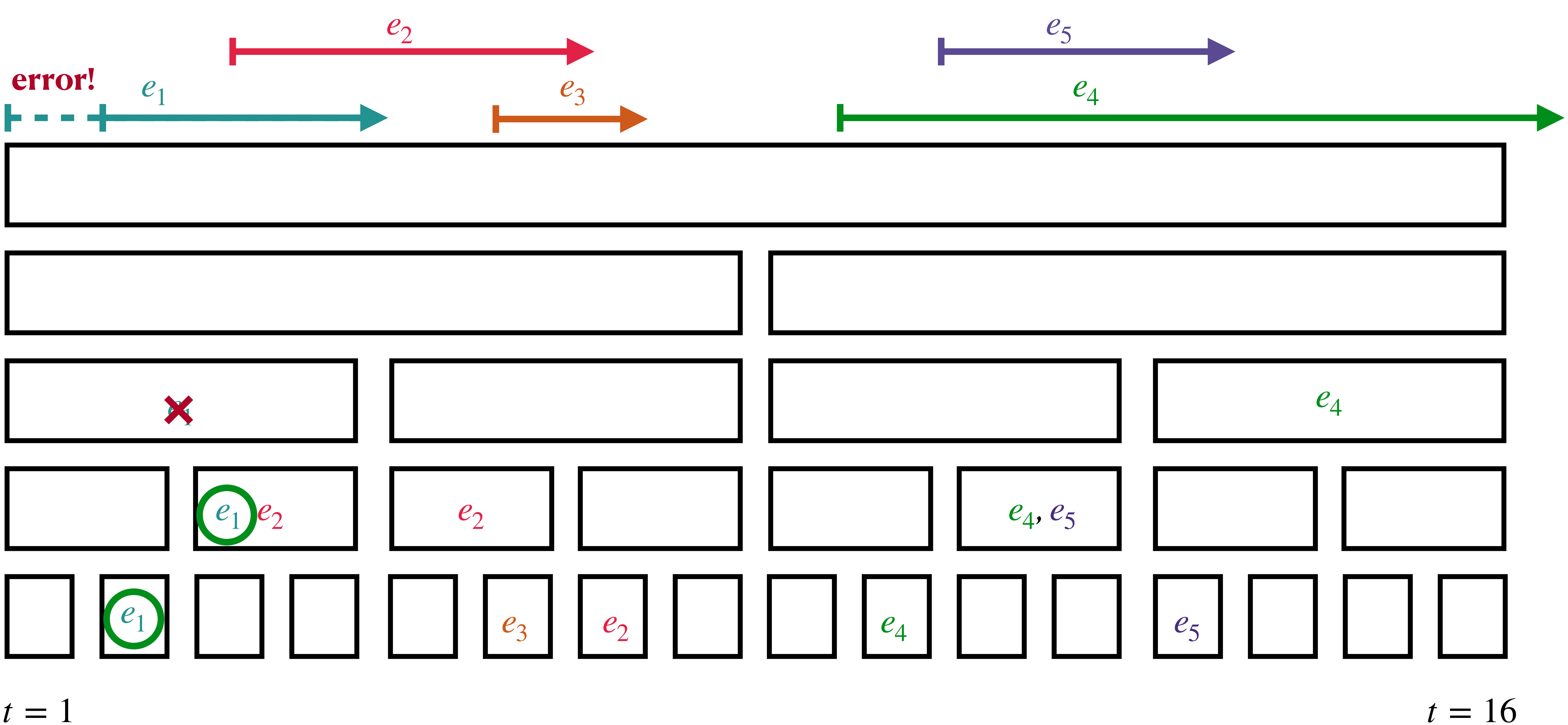
Predicted e_1 insertion on day 1, but happens on day 2 instead.



What if the prediction is wrong?

Predicted e_1 insertion on day 1, but happens on day 2 instead.

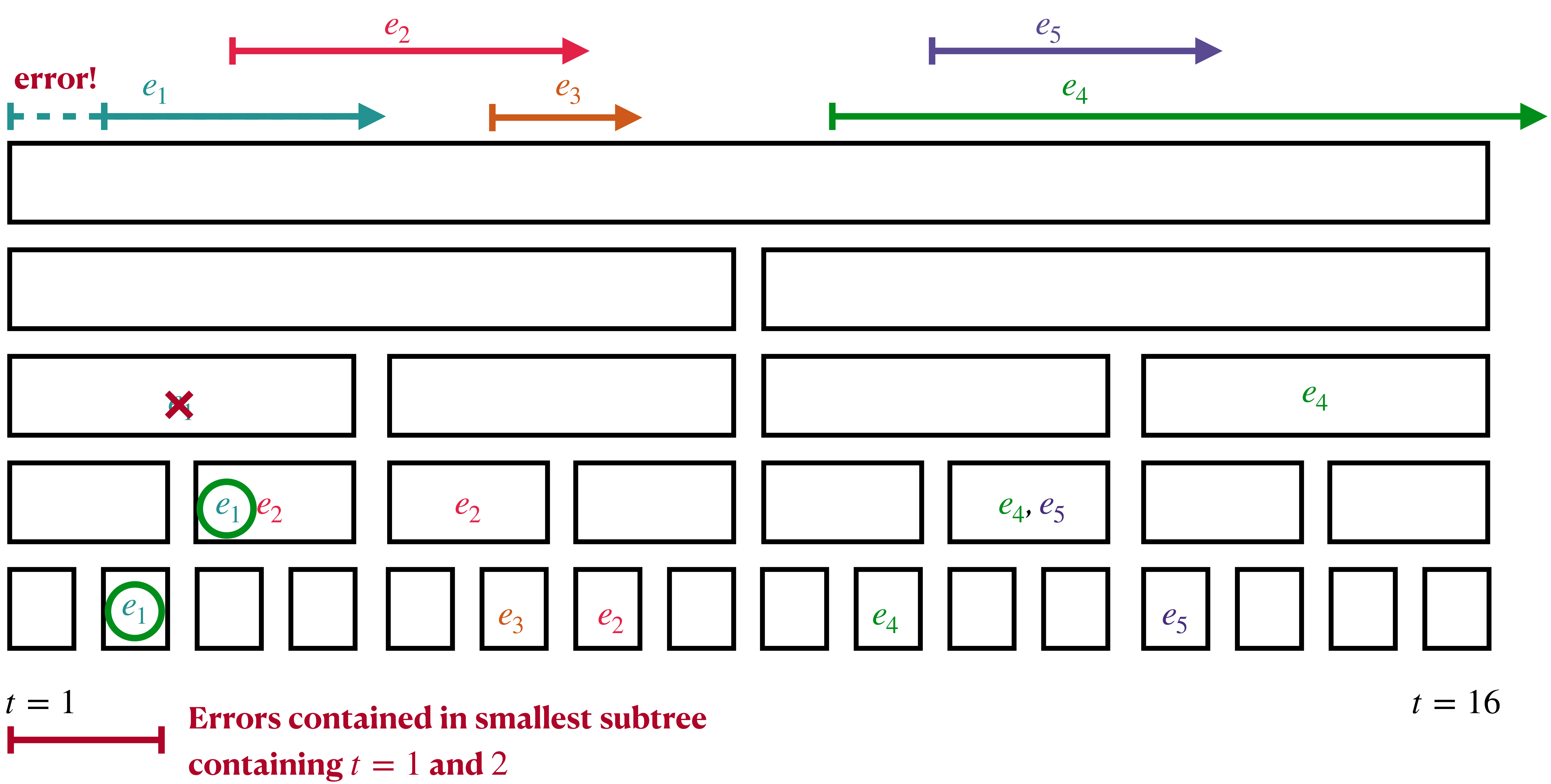
Observation: errors are localized!



What if the prediction is wrong?

Predicted e_1 insertion on day 1, but happens on day 2 instead.

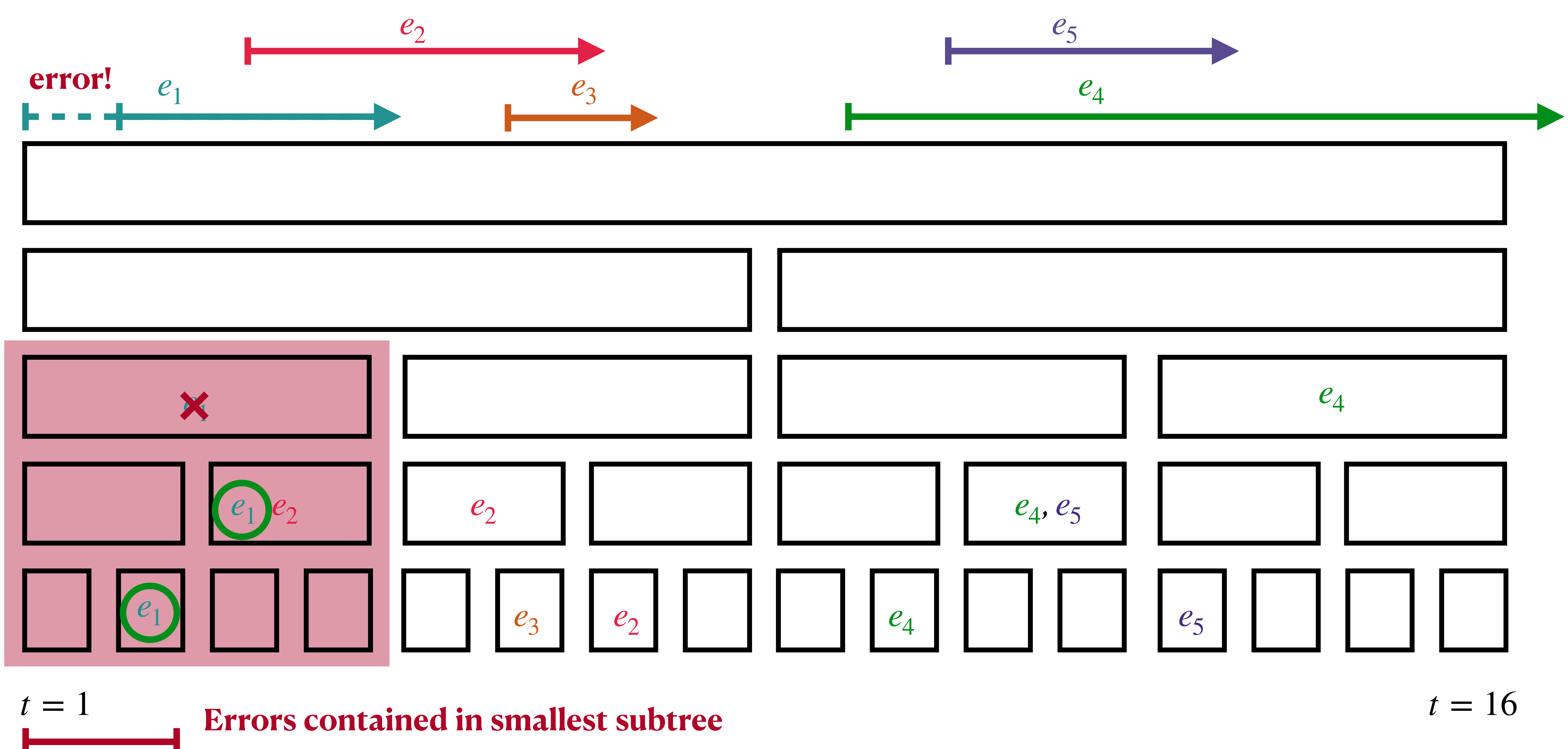
Observation: errors are localized!



What if the prediction is wrong?

Predicted e_1 insertion on day 1, but happens on day 2 instead.

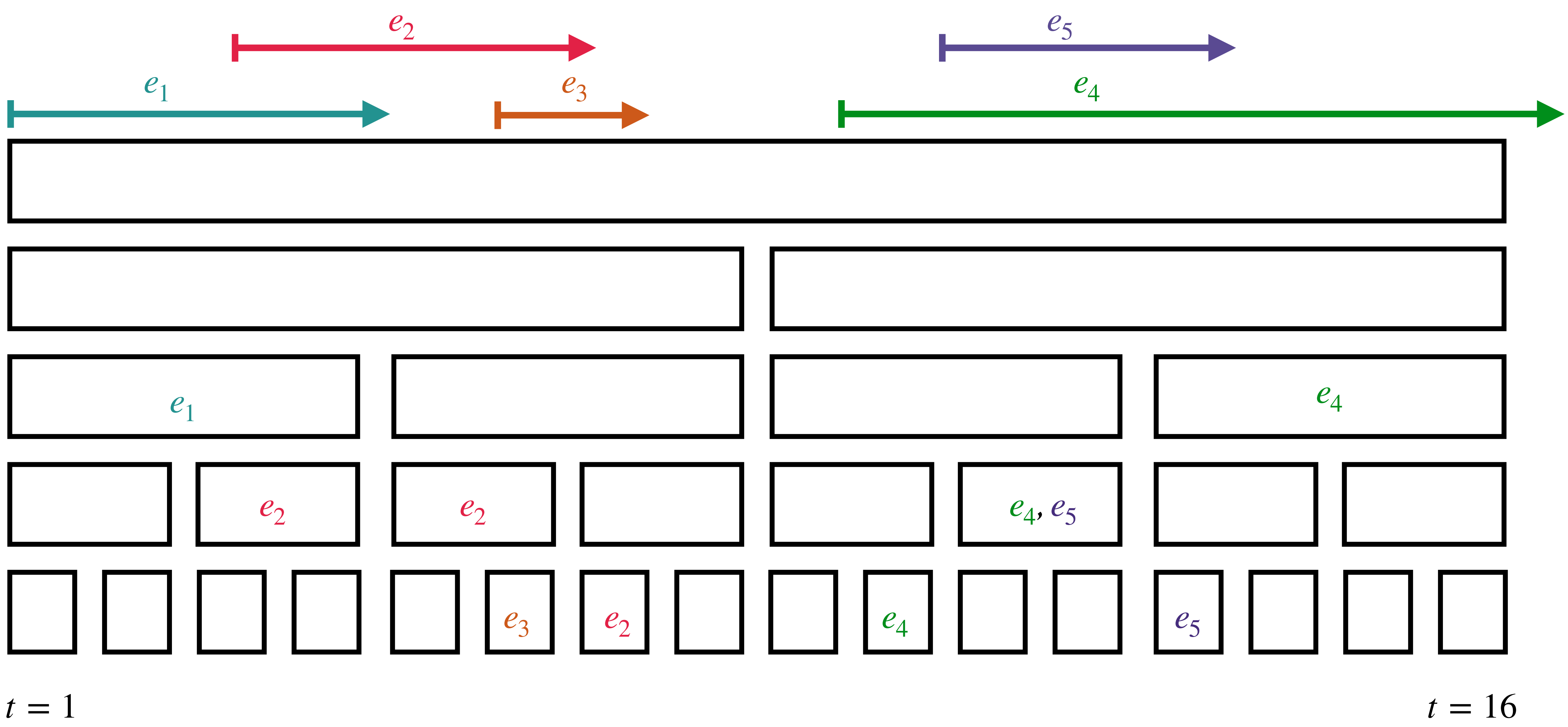
Observation: errors are localized!



Errors contained in smallest subtree containing $t = 1$ and 2

What if the prediction is wrong? (Part 2)

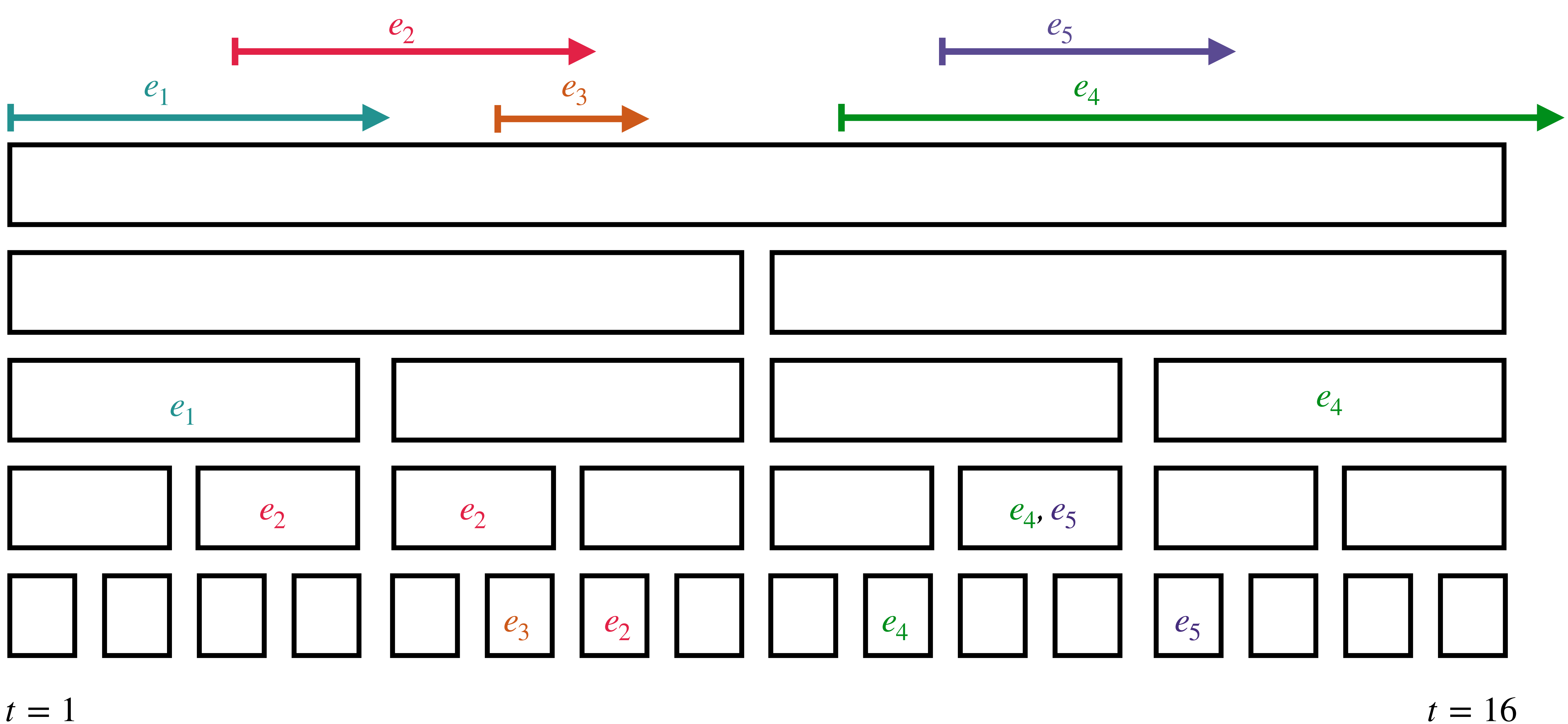
Observation: errors are localized!



What if the prediction is wrong? (Part 2)

Predicted e_4 insertion on day 10, but happens on day 8 instead.

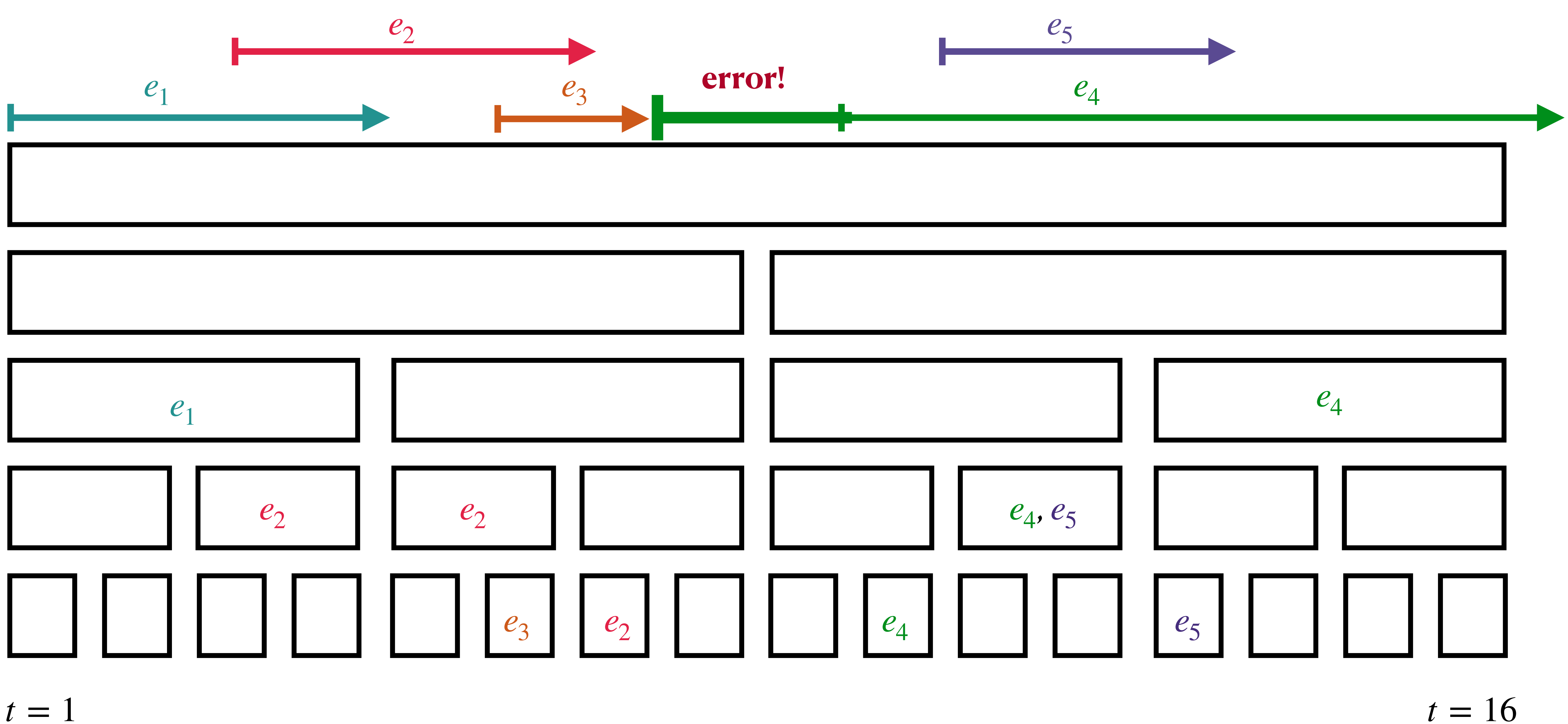
Observation: errors are localized!



What if the prediction is wrong? (Part 2)

Predicted e_4 insertion on day 10, but happens on day 8 instead.

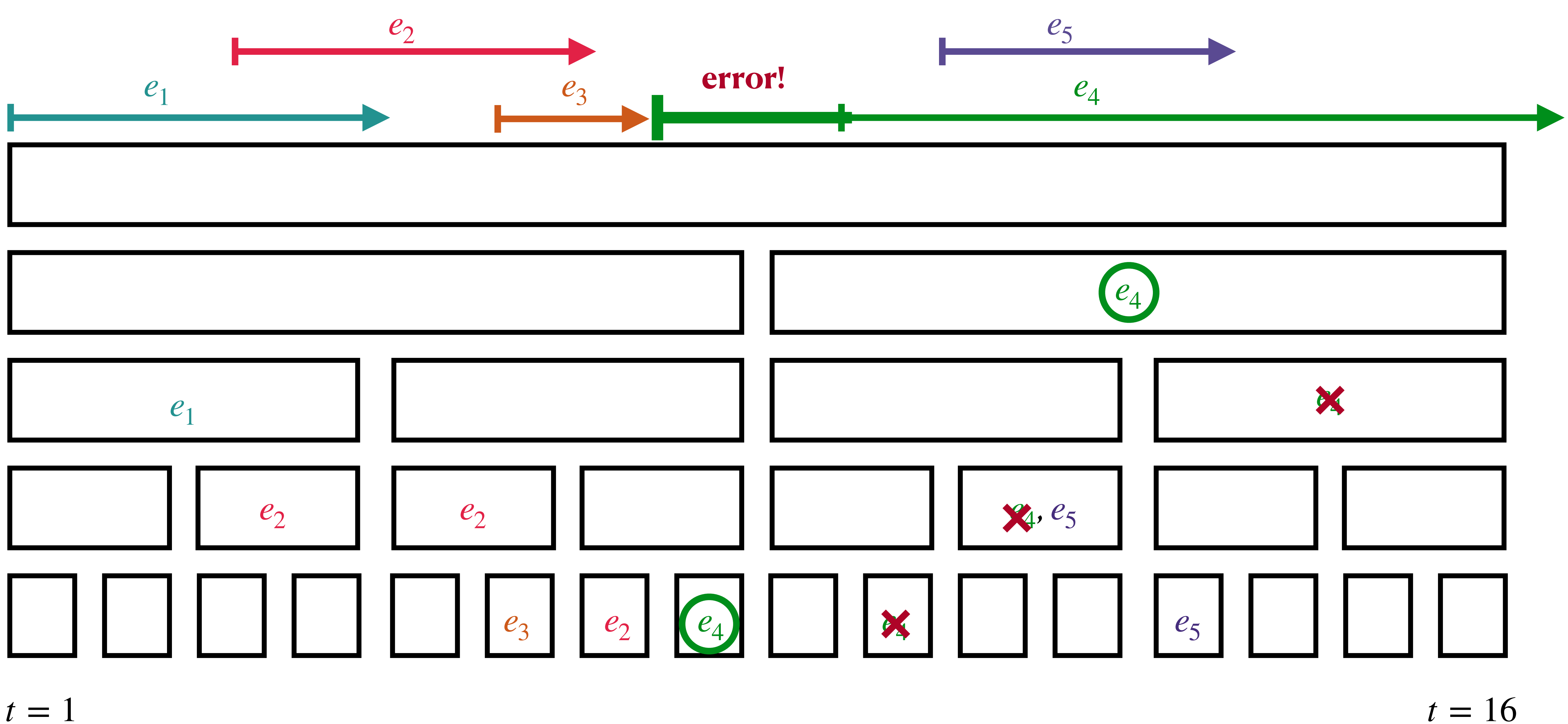
Observation: errors are localized!



What if the prediction is wrong? (Part 2)

Predicted e_4 insertion on day 10, but happens on day 8 instead.

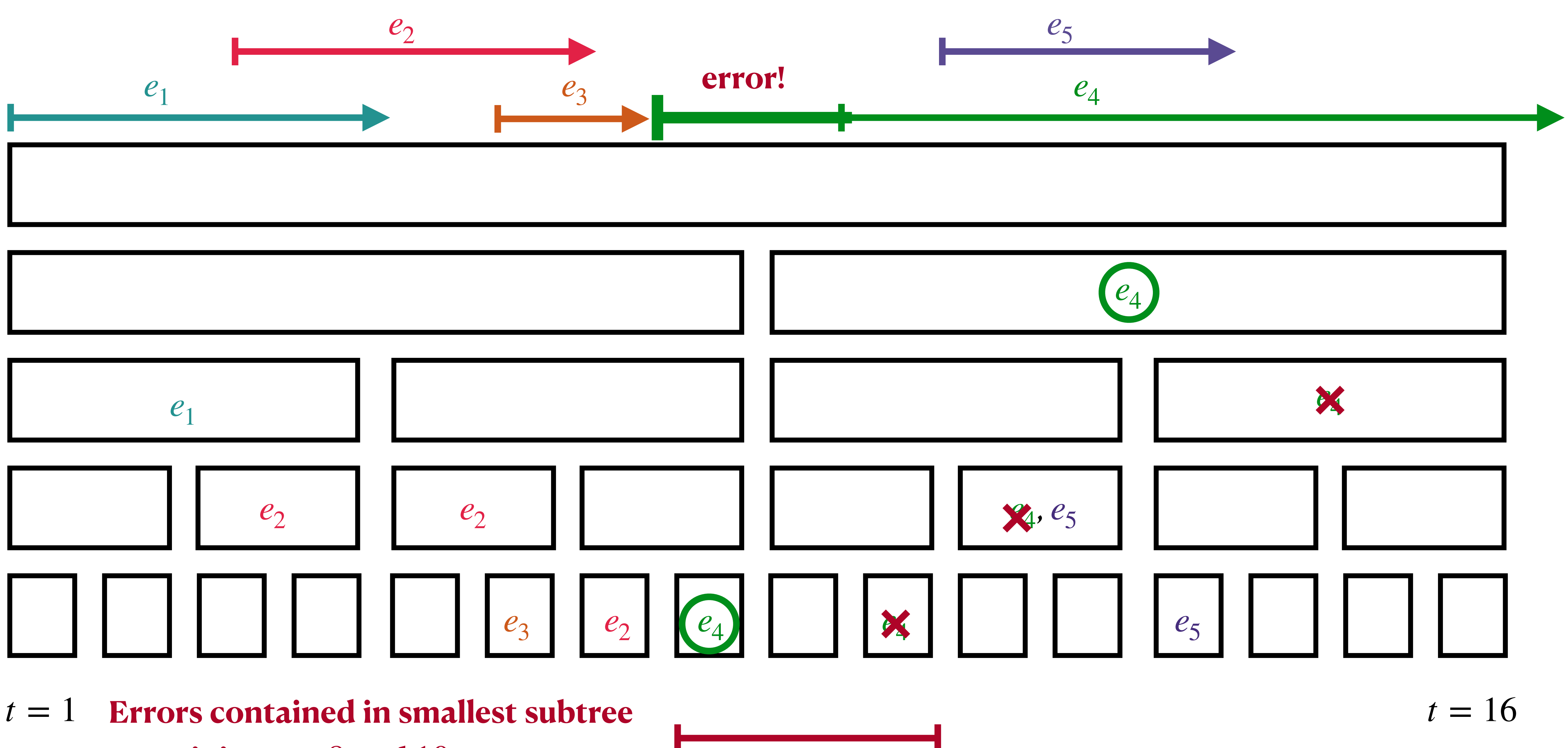
Observation: errors are localized!



What if the prediction is wrong? (Part 2)

Predicted e_4 insertion on day 10, but happens on day 8 instead.

Observation: errors are localized!

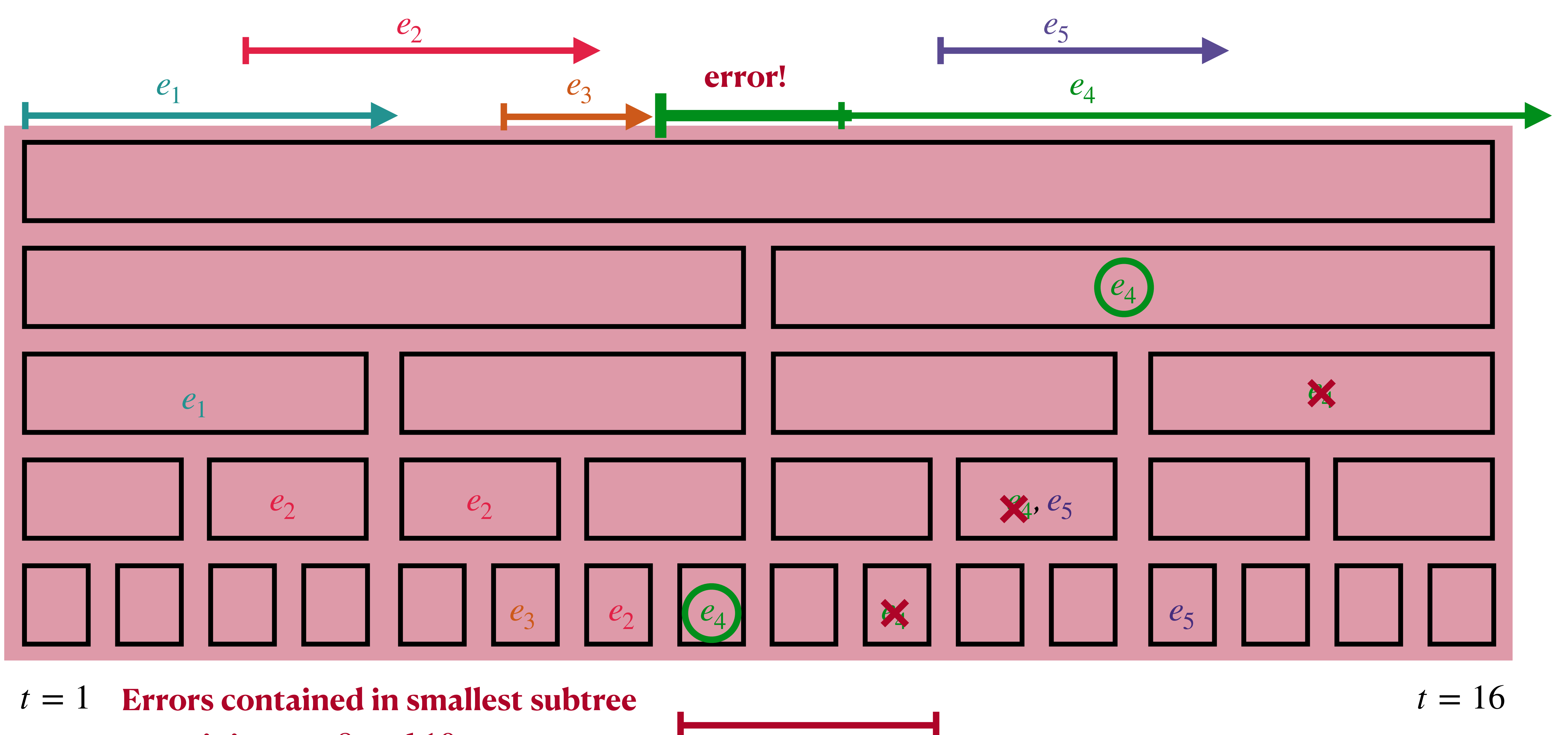


Errors contained in smallest subtree containing $t = 8$ and 10

What if the prediction is wrong? (Part 2)

Predicted e_4 insertion on day 10, but happens on day 8 instead.

Observation: errors are localized!



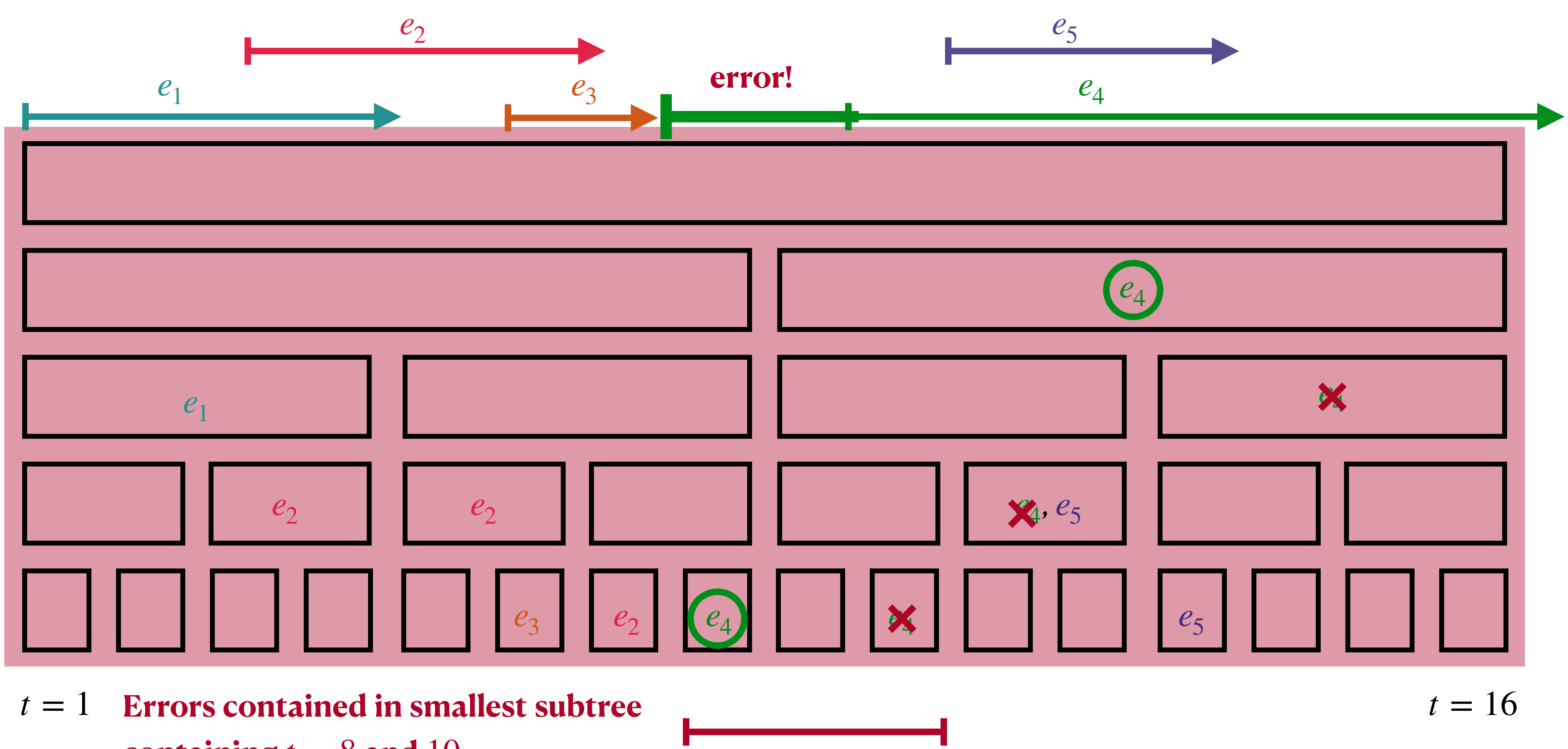
Errors contained in smallest subtree containing $t = 8$ and 10

What if the prediction is wrong? (Part 2)

Predicted e_4 insertion on day 10, but happens on day 8 instead.

Observation: errors are localized!

...but not very localized, small errors can trigger large recomputations



Workaround

Workaround

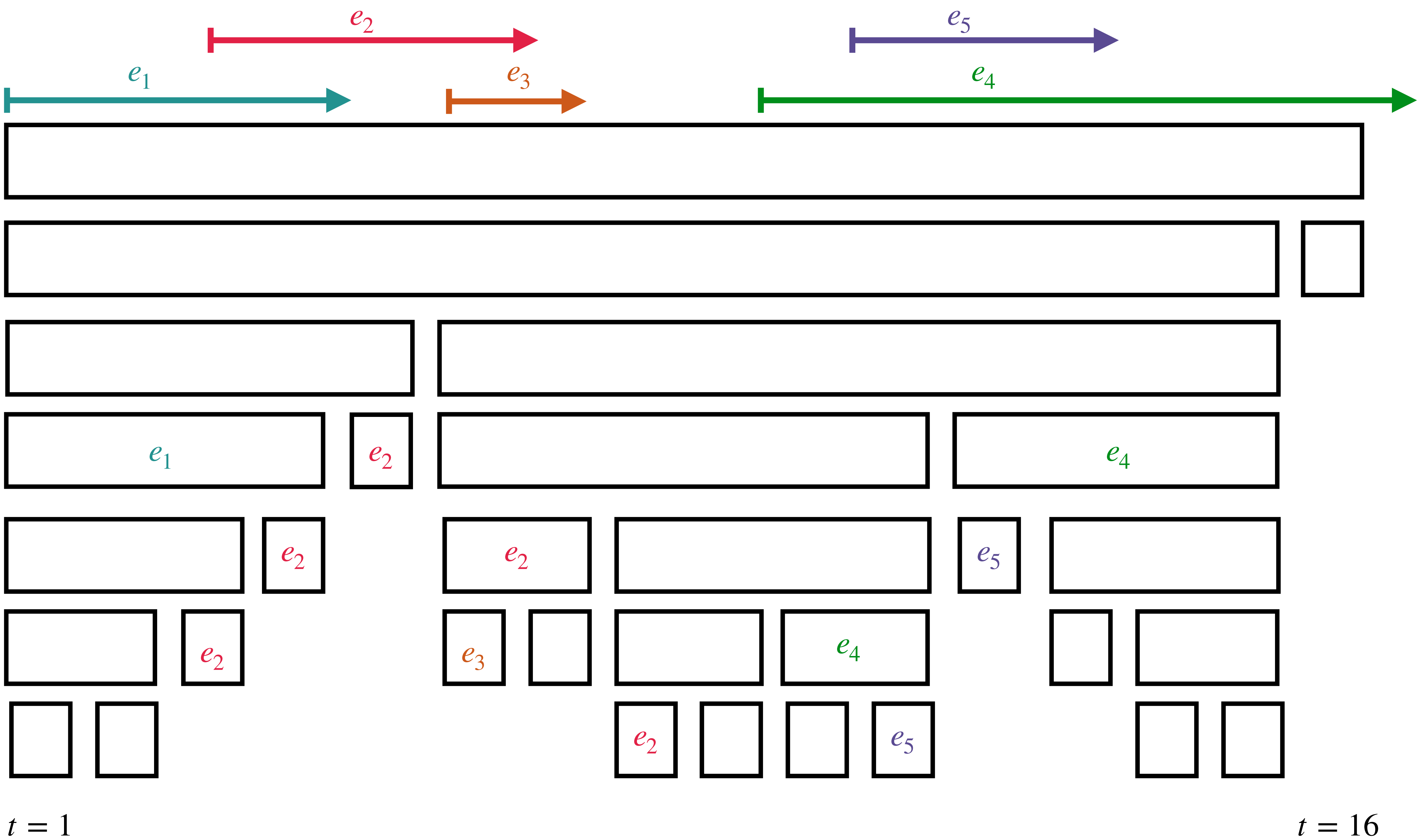
Use **random** divide-and-conquer tree

Workaround

Use **random** divide-and-conquer tree

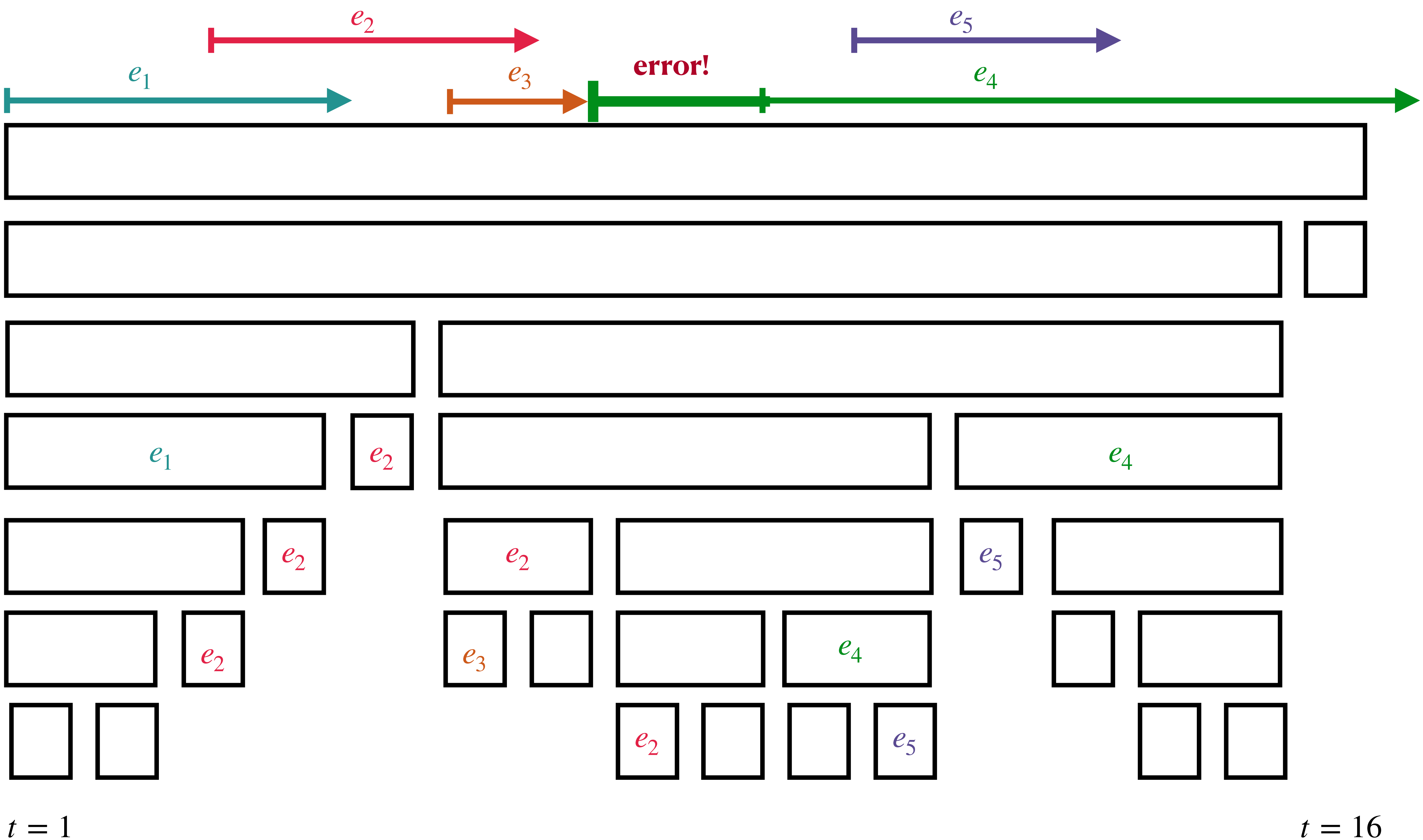
Workaround

Use **random** divide-and-conquer tree



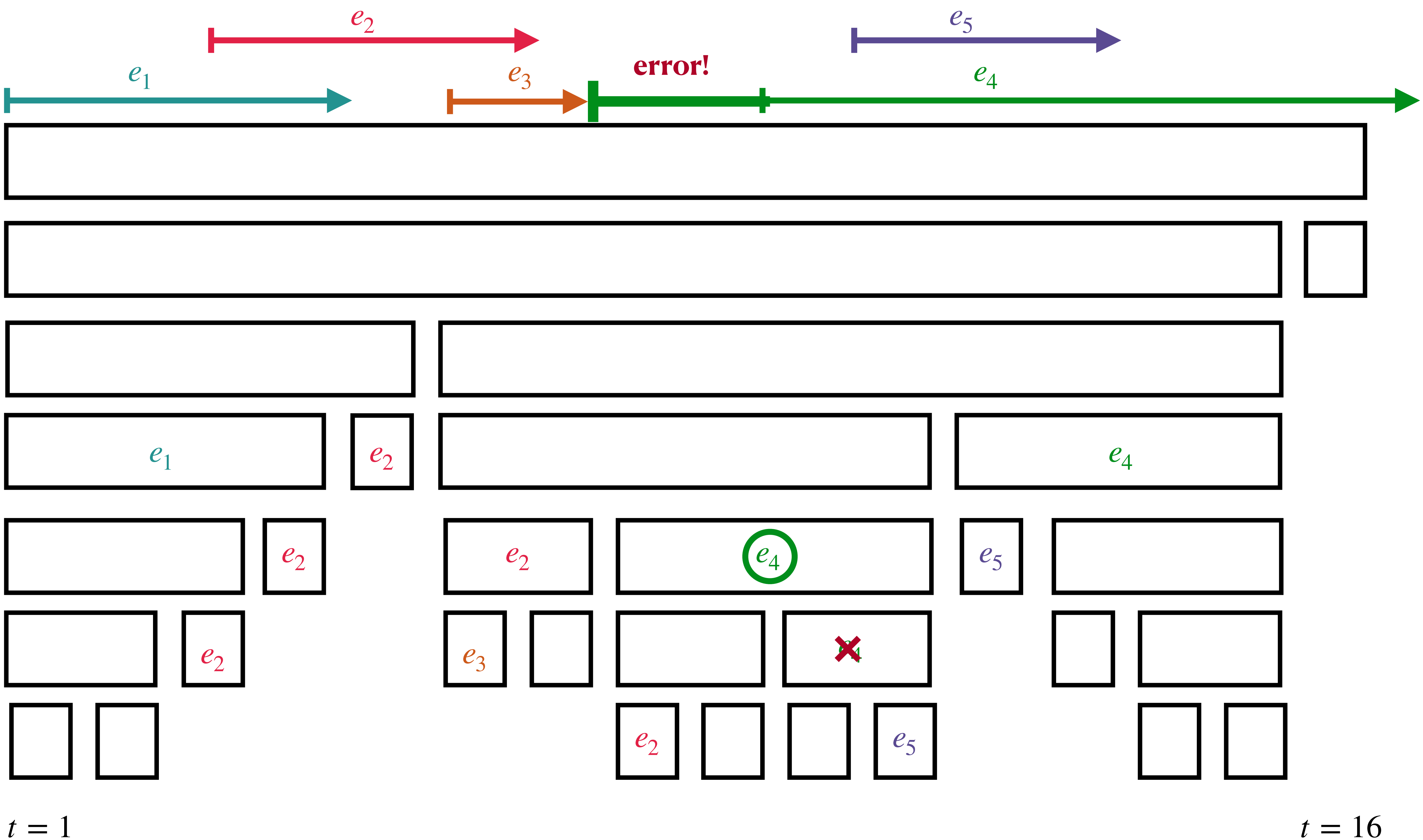
Workaround

Use **random** divide-and-conquer tree



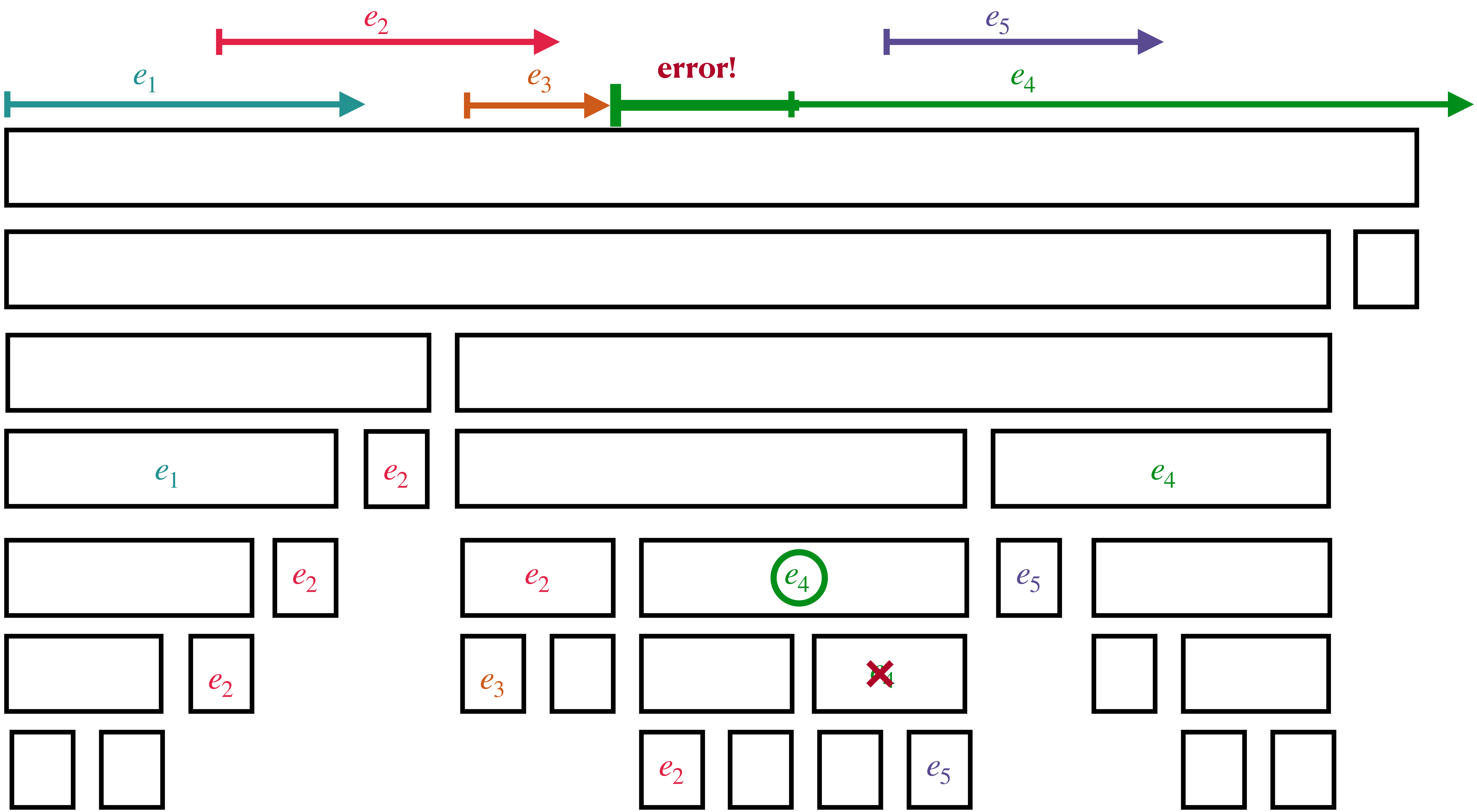
Workaround

Use **random** divide-and-conquer tree



Workaround

Use **random** divide-and-conquer tree



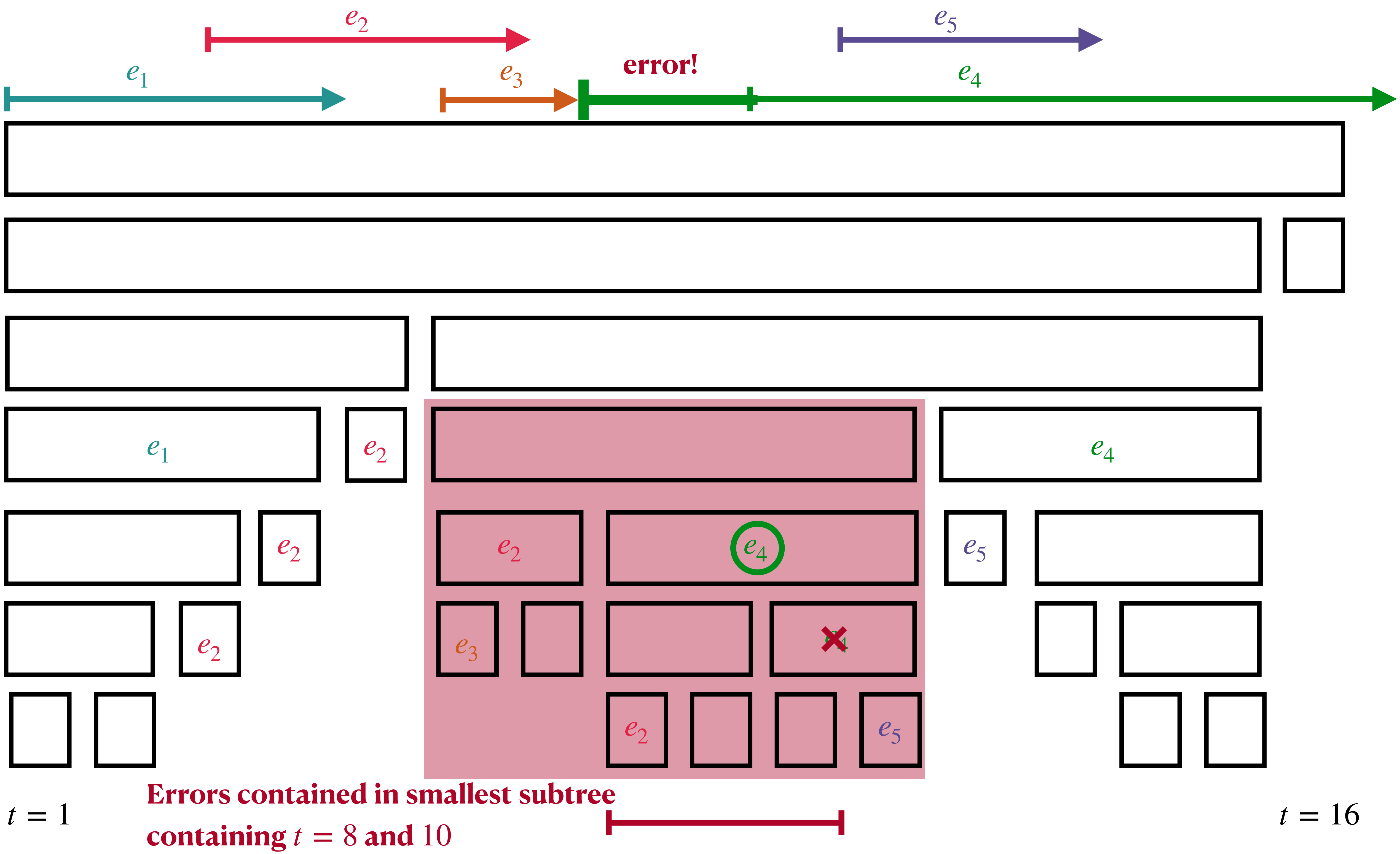
$t = 1$

Errors contained in smallest subtree containing $t = 8$ **and** 10

$t = 16$

Workaround

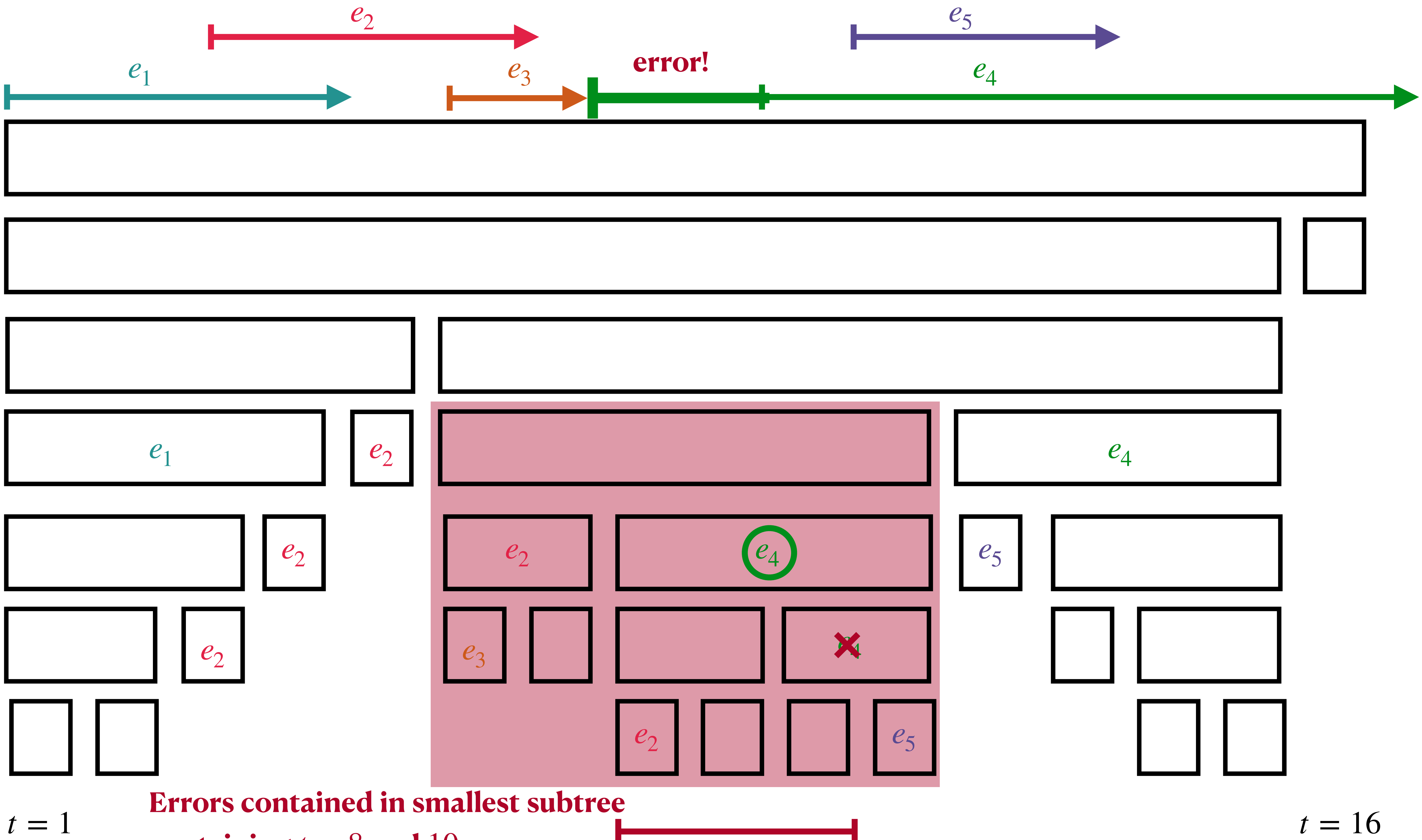
Use **random** divide-and-conquer tree



Workaround

Use **random** divide-and-conquer tree

$\Longrightarrow$ size of error subtree proportional to size of error



Errors contained in smallest subtree containing $t = 8$ and 10

Guarantees

Guarantees

Use this (and other) ideas to get an

fully-dynamic to offline reduction with predictions

Guarantees

Use this (and other) ideas to get an

fully-dynamic to offline reduction with predictions

Informal Theorem [Liu Srinivas '24]

Given a predicted sequence of events in advance, we can solve dynamic triconnectivity* in total time over T updates

$$\widetilde{O}(\min\{T + \eta,\ T \cdot n^{2/3}\}),$$

Where η is the ℓ_1 error of the prediction (sum over events of the absolute prediction error in time).

Guarantees

Use this (and other) ideas to get an

fully-dynamic to offline reduction with predictions

Informal Theorem [Liu Srinivas '24]

Given a predicted sequence of events in advance, we can solve dynamic triconnectivity* in total time over T updates

$$\widetilde{O}(\min\{T + \eta,\ T \cdot n^{2/3}\}),$$

Where η is the ℓ_1 error of the prediction (sum over events of the absolute prediction error in time).

*Can do reduction for many problems with similar offline dynamic algorithms!

Guarantees

Use this (and other) ideas to get an

fully-dynamic to offline reduction with predictions

Informal Theorem [Liu Srinivas ‘24]

Given a predicted sequence of events in advance, we can solve dynamic triconnectivity* in total time over T updates

$$\widetilde{O}(\min\{T+\eta,\ T\cdot n^{2/3}\}),$$

Where η is the ℓ_1 error of the prediction (sum over events of the absolute prediction error in time).

*Can do reduction for many problems with similar offline dynamic algorithms!

Takeaway: asymptotically lose **nothing** by taking advantage of predictions!

Big Picture

Many recent works use similar and different techniques to use predictions to get improved dynamic algorithms:

Big Picture

Many recent works use similar and different techniques to use predictions to get improved dynamic algorithms:

- [van den Brand Forster Nazari Polak '23] Other graph and matrix problems

Big Picture

Many recent works use similar and different techniques to use predictions to get improved dynamic algorithms:

- [van den Brand Forster Nazari Polak '23] Other graph and matrix problems
- [Agarwal Balkanski '24] Dynamic submodular maximization

Big Picture

Many recent works use similar and different techniques to use predictions to get improved dynamic algorithms:

- [van den Brand Forster Nazari Polak '23] Other graph and matrix problems
- [Agarwal Balkanski '24] Dynamic submodular maximization
- [McCauley Moseley Niaparast Singh '24] Incremental Topological Ordering

Big Picture

Many recent works use similar and different techniques to use predictions to get improved dynamic algorithms:

- [van den Brand Forster Nazari Polak '23] Other graph and matrix problems
- [Agarwal Balkanski '24] Dynamic submodular maximization
- [McCauley Moseley Niaparast Singh '24] Incremental Topological Ordering
- [McCauley Moseley Niaparast Niaparast Singh '25] Approximate SSSP

Big Picture

Many recent works use similar and different techniques to use predictions to get improved dynamic algorithms:

- [van den Brand Forster Nazari Polak '23] Other graph and matrix problems
- [Agarwal Balkanski '24] Dynamic submodular maximization
- [McCauley Moseley Niaparast Singh '24] Incremental Topological Ordering
- [McCauley Moseley Niaparast Niaparast Singh '25] Approximate SSSP

Spiritually similar but **incomparable** to warm starts

Big Picture

Many recent works use similar and different techniques to use predictions to get improved dynamic algorithms:

- [van den Brand Forster Nazari Polak '23] Other graph and matrix problems
- [Agarwal Balkanski '24] Dynamic submodular maximization
- [McCauley Moseley Niaparast Singh '24] Incremental Topological Ordering
- [McCauley Moseley Niaparast Niaparast Singh '25] Approximate SSSP

Spiritually similar but **incomparable** to warm starts

Dynamic Algorithms	Warm Starts

Big Picture

Many recent works use similar and different techniques to use predictions to get improved dynamic algorithms:

- [van den Brand Forster Nazari Polak '23] Other graph and matrix problems
- [Agarwal Balkanski '24] Dynamic submodular maximization
- [McCauley Moseley Niaparast Singh '24] Incremental Topological Ordering
- [McCauley Moseley Niaparast Niaparast Singh '25] Approximate SSSP

Spiritually similar but **incomparable** to warm starts

Dynamic Algorithms	Warm Starts
• Structured input sequence (edge insertion/deletions)	• Unstructured input sequence

Big Picture

Many recent works use similar and different techniques to use predictions to get improved dynamic algorithms:

- [van den Brand Forster Nazari Polak '23] Other graph and matrix problems
- [Agarwal Balkanski '24] Dynamic submodular maximization
- [McCauley Moseley Niaparast Singh '24] Incremental Topological Ordering
- [McCauley Moseley Niaparast Niaparast Singh '25] Approximate SSSP

Spiritually similar but **incomparable** to warm starts

Dynamic Algorithms	Warm Starts
• Structured input sequence (edge insertion/deletions) • Small changes in input, potentially large changes in solution	• Unstructured input sequence • Potentially large changes in input, take advantage of similarities in solutions

Big Picture

Many recent works use similar and different techniques to use predictions to get improved dynamic algorithms:

- [van den Brand Forster Nazari Polak '23] Other graph and matrix problems
- [Agarwal Balkanski '24] Dynamic submodular maximization
- [McCauley Moseley Niaparast Singh '24] Incremental Topological Ordering
- [McCauley Moseley Niaparast Niaparast Singh '25] Approximate SSSP

Spiritually similar but **incomparable** to warm starts

Dynamic Algorithms

- Structured input sequence (edge insertion/deletions)
- Small changes in input, potentially large changes in solution
- Can achieve sublinear update time

Warm Starts

- Unstructured input sequence
- Potentially large changes in input, take advantage of similarities in solutions
- At bare minimum, must read input on each day $\Rightarrow$ no sublinear update time

References

(2) Dynamic Algorithms/Data Structures: Dynamic problems are easier with information about future updates

Dynamic algorithms with predictions:

- [Liu Srinivas '24] Offline to online transformations
- [van den Brand Forster Nazari Polak '23] Graph and matrix problems
- [Agarwal Balkanski '24] Submodular maximization
- [McCauley Moseley Niaparast Singh '24] Incremental Topological Ordering
- [McCauley Moseley Niaparast Niaparast Singh '25] Incremental Approximate Single Source Shortest Paths

Related ideas:

- [Peng Sandlund Sleator '17] Designing offline-dynamic algorithms
- [Peng Rubinstein '22] Fully-dynamic to incremental reduction with known deletion error

Three ideas

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

(3) **Randomized Algorithms:** Randomized algorithms and data structures can be hedged to take advantage of extra information by incorporating a prior

Example: Treaps

Example: Treaps

Self-balancing tree data structure

Example: Treaps

Self-balancing tree data structure

Goal: Support element insertions, lookups, and deletions in $O(\log(n))$ time, n is # of elements in the tree at any given time

Example: Treaps

Self-balancing tree data structure

Goal: Support element insertions, lookups, and deletions in $O(\log(n))$ time, n is # of elements in the tree at any given time

Treap = **Tree** + **Heap** [Aragon Seidel '89]

Example: Treaps

Self-balancing tree data structure

Goal: Support element insertions, lookups, and deletions in $O(\log(n))$ time, n is # of elements in the tree at any given time

Rank $\in [0,1]$
unif. random!

Value $\in \mathbb{R}$

Treap = **Tree** + **Heap** [Aragon Seidel '89]

Example: Treaps

Self-balancing tree data structure

Goal: Support element insertions, lookups, and deletions in $O(\log(n))$ time, n is # of elements in the tree at any given time

Rank $\in [0,1]$
unif. random!

Value $\in \mathbb{R}$

Treap = **Tree** + **Heap** [Aragon Seidel '89]

Min-heap w.r.t. rank: parent always has lower rank than children

Example: Treaps

Self-balancing tree data structure

Goal: Support element insertions, lookups, and deletions in $O(\log(n))$ time, n is # of elements in the tree at any given time

Rank $\in [0,1]$
unif. random!

Value $\in \mathbb{R}$

Treap = **Tree** + **Heap** [Aragon Seidel '89]

Min-heap w.r.t. rank: parent always has lower rank than children

Search tree w.r.t. value: left subtree contains only elements of smaller value, right subtree contains only elements of larger value

Example: Treaps

Self-balancing tree data structure

Goal: Support element insertions, lookups, and deletions in $O(\log(n))$ time, n is # of elements in the tree at any given time

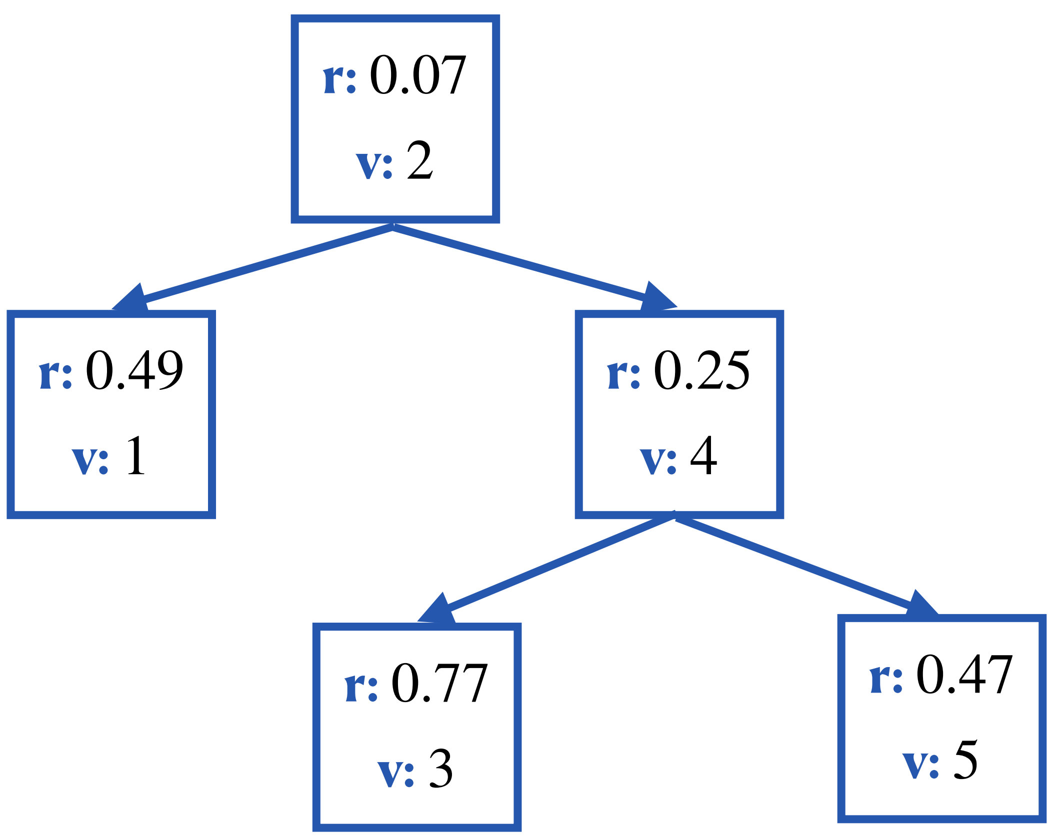
Rank $\in [0,1]$
unif. random!

Value $\in \mathbb{R}$

Treap = Tree + Heap [Aragon Seidel '89]

Min-heap w.r.t. rank: parent always has lower rank than children

Search tree w.r.t. value: left subtree contains only elements of smaller value, right subtree contains only elements of larger value



Example: Treaps

Self-balancing tree data structure

Goal: Support element insertions, lookups, and deletions in $O(\log(n))$ time, n is # of elements in the tree at any given time

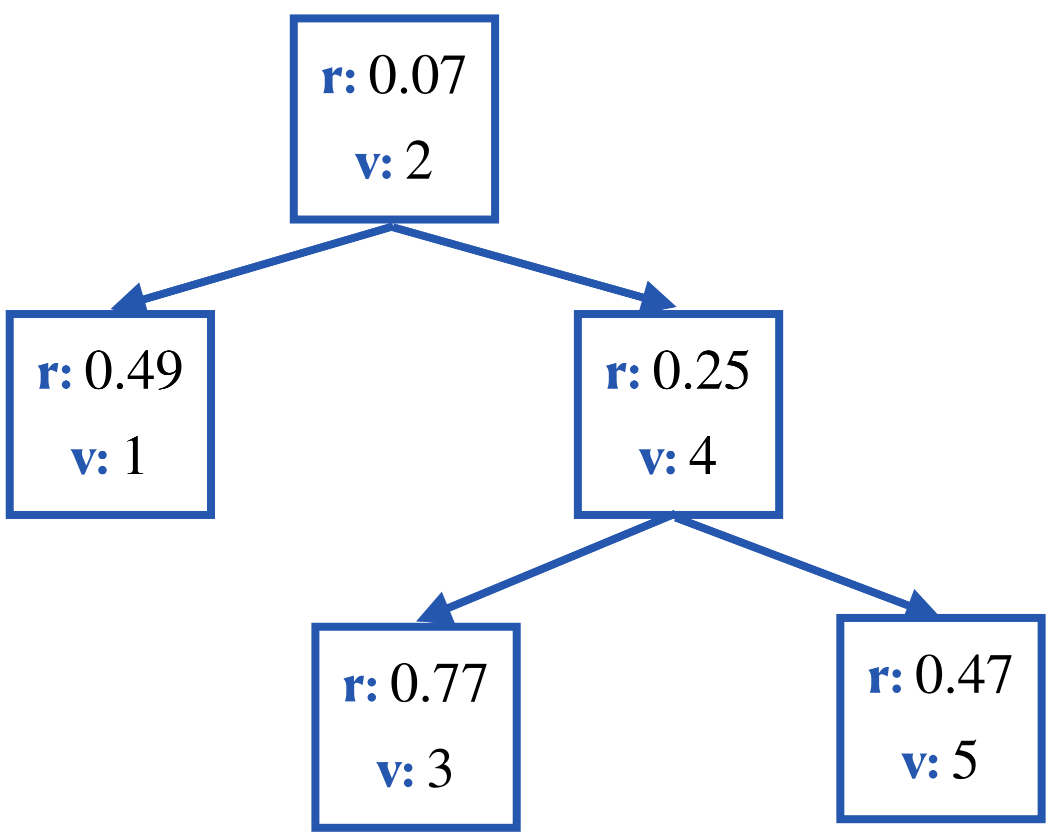
Rank $\in [0,1]$
unif. random!

Value $\in \mathbb{R}$

Treap = **Tree** + **Heap** [Aragon Seidel '89]

Min-heap w.r.t. rank: parent always has lower rank than children

Search tree w.r.t. value: left subtree contains only elements of smaller value, right subtree contains only elements of larger value



- $\log(n)$ depth with high probability (quicksort type analysis)

Example: Treaps

Self-balancing tree data structure

Goal: Support element insertions, lookups, and deletions in $O(\log(n))$ time, n is # of elements in the tree at any given time

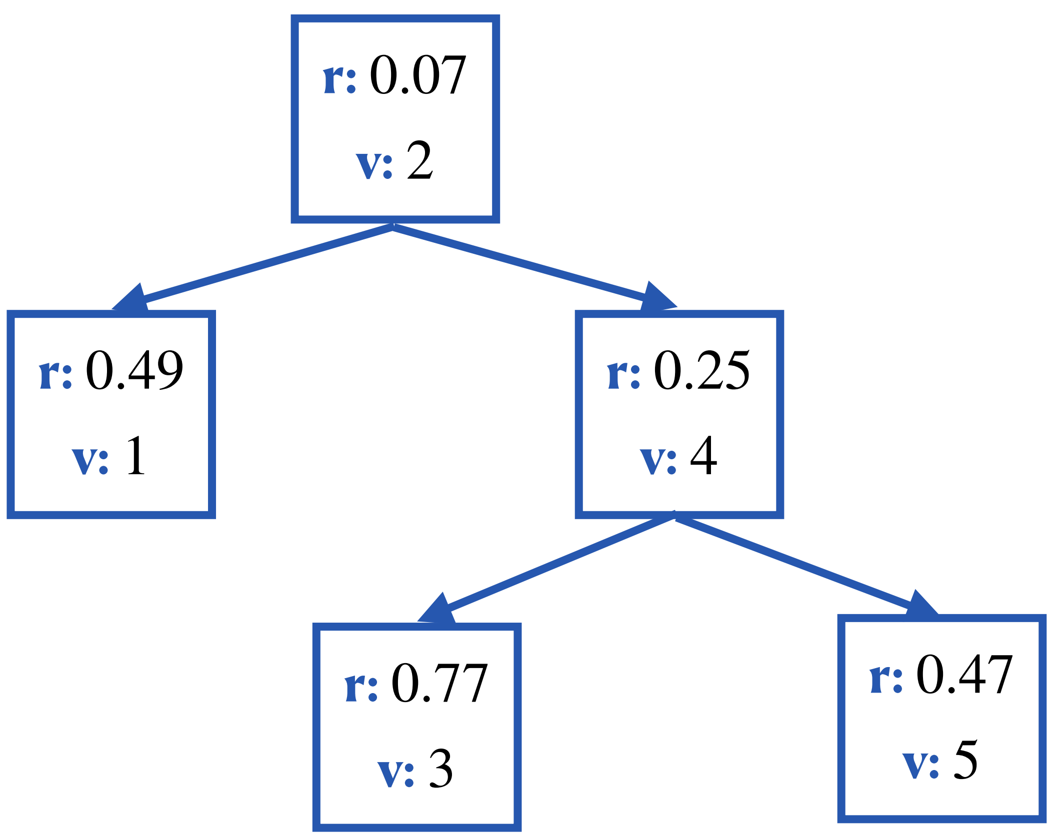
Rank $\in [0,1]$
unif. random!

Value $\in \mathbb{R}$

Treap = **Tree** + **Heap** [Aragon Seidel '89]

Min-heap w.r.t. rank: parent always has lower rank than children

Search tree w.r.t. value: left subtree contains only elements of smaller value, right subtree contains only elements of larger value



- $\log(n)$ depth with high probability (quicksort type analysis)
- Coupling between ranks and random tree allows dynamic updates

Room for improvement?

Room for improvement?

- $\log(n)$ update/query time optimal for a **worst-case** sequence

Room for improvement?

- $\log(n)$ update/query time optimal for a **worst-case** sequence
- For a given update sequence, could minimize lookup time by storing frequently accessed elements closer to the root

Room for improvement?

- $\log(n)$ update/query time optimal for a **worst-case** sequence
- For a given update sequence, could minimize lookup time by storing frequently accessed elements closer to the root
  - E.g., splay trees rebalance with accesses to do this dynamically

Room for improvement?

- $\log(n)$ update/query time optimal for a **worst-case** sequence
- For a given update sequence, could minimize lookup time by storing frequently accessed elements closer to the root
  - E.g., splay trees rebalance with accesses to do this dynamically
  - Splay trees have conjectured dynamic optimality (within a constant factor of best self-balancing tree in hindsight for every access sequence)

Room for improvement?

- $\log(n)$ update/query time optimal for a **worst-case** sequence
- For a given update sequence, could minimize lookup time by storing frequently accessed elements closer to the root
  - E.g., splay trees rebalance with accesses to do this dynamically
  - Splay trees have conjectured dynamic optimality (within a constant factor of best self-balancing tree in hindsight for every access sequence)
  - Dynamic optimality: long standing open problem

Room for improvement?

- $\log(n)$ update/query time optimal for a **worst-case** sequence
- For a given update sequence, could minimize lookup time by storing frequently accessed elements closer to the root
    - E.g., splay trees rebalance with accesses to do this dynamically
    - Splay trees have conjectured dynamic optimality (within a constant factor of best self-balancing tree in hindsight for every access sequence)
    - Dynamic optimality: long standing open problem
- What if you had **predictions** about the access sequence in advance? Can you get a **provable** optimality guarantee, without sacrificing worst-case update time?

Incorporate the prior

Incorporate the prior

Idea: incorporate frequency information into the random rank! [Lin Luo Woodruff '23]

Incorporate the prior

Idea: incorporate frequency information into the random rank! [Lin Luo Woodruff '23]

For an element x, let $p(x)$ be the **predicted** frequency with which x will be accessed

Incorporate the prior

Idea: incorporate frequency information into the random rank! [Lin Luo Woodruff '23]

For an element x, let $p(x)$ be the **predicted** frequency with which x will be accessed

Rank $\in [0,1]$

Value $\in \mathbb{R}$

Incorporate the prior

Idea: incorporate frequency information into the random rank! [Lin Luo Woodruff '23]

For an element x, let $p(x)$ be the **predicted** frequency with which x will be accessed

Rank $\in [0,1]$

Value $\in \mathbb{R}$

Standard Treap: Choose rank $\sim U[0,1]$

Incorporate the prior

Idea: incorporate frequency information into the random rank! [Lin Luo Woodruff '23]

For an element x, let $p(x)$ be the **predicted** frequency with which x will be accessed

Rank $\in [0,1]$

Value $\in \mathbb{R}$

Standard Treap: Choose rank $\sim U[0,1]$

Learning-Augmented Treap: $\text{rank}(x) \sim -\lfloor \log\log(p(x)) \rfloor + U[0,1]$

Incorporate the prior

Idea: incorporate frequency information into the random rank! [Lin Luo Woodruff '23]

For an element x, let $p(x)$ be the **predicted** frequency with which x will be accessed

Rank $\in [0,1]$

Value $\in \mathbb{R}$

Standard Treap: Choose rank $\sim U[0,1]$

Learning-Augmented Treap: $\text{rank}(x) \sim -\lfloor \log\log(p(x)) \rfloor + U[0,1]$

Hedge frequently accessed items to the top!

Incorporate the prior

Idea: incorporate frequency information into the random rank! [Lin Luo Woodruff '23]

For an element x, let $p(x)$ be the **predicted** frequency with which x will be accessed

Rank $\in [0,1]$

Value $\in \mathbb{R}$

Standard Treap: Choose rank $\sim U[0,1]$

Learning-Augmented Treap: $\text{rank}(x) \sim -\lfloor \log \log(p(x)) \rfloor + U[0,1]$

Hedge frequently accessed items to the top!

Provable guarantees [Chen Cao Stepin Chen '25]**:**

Incorporate the prior

Idea: incorporate frequency information into the random rank! [Lin Luo Woodruff '23]

For an element x, let $p(x)$ be the **predicted** frequency with which x will be accessed

Rank $\in [0,1]$

Value $\in \mathbb{R}$

Standard Treap: Choose rank $\sim U[0,1]$

Learning-Augmented Treap: $\text{rank}(x) \sim -\lfloor \log\log(p(x)) \rfloor + U[0,1]$

Hedge frequently accessed items to the top!

Provable guarantees [Chen Cao Stepin Chen '25]**:**

- Static optimality, when predictions are perfect

Incorporate the prior

Idea: incorporate frequency information into the random rank! [Lin Luo Woodruff '23]

For an element x, let $p(x)$ be the **predicted** frequency with which x will be accessed

$\boxed{\begin{array}{c}\textbf{Rank} \in [0,1] \\ \textbf{Value} \in \mathbb{R}\end{array}}$

Standard Treap: Choose rank $\sim U[0,1]$

Learning-Augmented Treap: $\text{rank}(x) \sim -\lfloor \log \log(p(x)) \rfloor + U[0,1]$

Hedge frequently accessed items to the top!

Provable guarantees [Chen Cao Stepin Chen '25]**:**

- Static optimality, when predictions are perfect
- Worst-case guarantees from Treap, with tradeoff

Incorporate the prior

Idea: incorporate frequency information into the random rank! [Lin Luo Woodruff '23]

For an element x, let $p(x)$ be the **predicted** frequency with which x will be accessed

Rank $\in [0,1]$

Value $\in \mathbb{R}$

Standard Treap: Choose rank $\sim U[0,1]$

Learning-Augmented Treap: $\text{rank}(x) \sim -\lfloor \log\log(p(x)) \rfloor + U[0,1]$

Hedge frequently accessed items to the top!

Provable guarantees [Chen Cao Stepin Chen '25]**:**

- Static optimality, when predictions are perfect
- Worst-case guarantees from Treap, with tradeoff
- (Meets provable benchmarks for splay trees, when frequencies are estimated on-the-fly)

Other Applications

Other Applications

Takeaway: Hedging is a natural idea. Learning-Augmented algorithms gives us the language to reason about it!

Other Applications

Takeaway: Hedging is a natural idea. Learning-Augmented algorithms gives us the language to reason about it!

Algorithmic challenge: How do you design the **right** distribution?

Other Applications

Takeaway: Hedging is a natural idea. Learning-Augmented algorithms gives us the language to reason about it!

Algorithmic challenge: How do you design the **right** distribution?

- Caching [Lykouris Vassilvitskii '18]

Other Applications

Takeaway: Hedging is a natural idea. Learning-Augmented algorithms gives us the language to reason about it!

Algorithmic challenge: How do you design the **right** distribution?

- Caching [Lykouris Vassilvitskii '18]
  - Use frequency information to inform randomized cache eviction decisions

Other Applications

Takeaway: Hedging is a natural idea. Learning-Augmented algorithms gives us the language to reason about it!

Algorithmic challenge: How do you design the **right** distribution?

- Caching [Lykouris Vassilvitskii '18]
  - Use frequency information to inform randomized cache eviction decisions
- Ski-Rental [Kumar Purohit Svitkina '18] ...

Other Applications

Takeaway: Hedging is a natural idea. Learning-Augmented algorithms gives us the language to reason about it!

Algorithmic challenge: How do you design the **right** distribution?

- Caching [Lykouris Vassilvitskii '18]
  - Use frequency information to inform randomized cache eviction decisions
- Ski-Rental [Kumar Purohit Svitkina '18] ...
  - Use information about the future to make decisions in the present for online algorithms

Other Applications

Takeaway: Hedging is a natural idea. Learning-Augmented algorithms gives us the language to reason about it!

Algorithmic challenge: How do you design the **right** distribution?

- Caching [Lykouris Vassilvitskii '18]
  - Use frequency information to inform randomized cache eviction decisions
- Ski-Rental [Kumar Purohit Svitkina '18] …
  - Use information about the future to make decisions in the present for online algorithms
- Min-cut [Moseley Niaparast Singh '25]

Other Applications

Takeaway: Hedging is a natural idea. Learning-Augmented algorithms gives us the language to reason about it!

Algorithmic challenge: How do you design the **right** distribution?

- Caching [Lykouris Vassilvitskii '18]
  - Use frequency information to inform randomized cache eviction decisions
- Ski-Rental [Kumar Purohit Svitkina '18] ...
  - Use information about the future to make decisions in the present for online algorithms
- Min-cut [Moseley Niaparast Singh '25]
  - Use predictions about the min-cut to inform what vertices to contract in Karger-Stein

References

(3) Randomized Algorithms: Randomized algorithms and data structures can be hedged to take advantage of extra information by incorporating a prior

Learning-Augmented Search Trees:

- [Lin Luo Woodruff '23] Learning-Augmented Treaps with guarantees for stochastic accesses
- [Chen Cao Stepin Chen '25] Guarantees for general access sequences

An incomplete list of other places to see this idea:

- [Lykouris Vassilvitskii '18] Caching
- [Kumar Purohit Svitkina '18], and follow up work, Ski-rental (proof of concept for online algorithms in general)
- [Moseley Niaparast Singh '25] Min-cut via Karger-Stein

Conclusion

Conclusion

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

Conclusion

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

Conclusion

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

(3) **Randomized Data Structures:** Randomized data structures can be hedged to take advantage of extra information by incorporating a prior

Conclusion

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

(3) **Randomized Data Structures:** Randomized data structures can be hedged to take advantage of extra information by incorporating a prior

...and so much more!

Conclusion

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

(3) **Randomized Data Structures:** Randomized data structures can be hedged to take advantage of extra information by incorporating a prior

...and so much more!

Takeaway: Learning-Augmented algorithms gives us tools and frameworks to reason about interesting and practical new problems

Conclusion

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

(3) **Randomized Data Structures:** Randomized data structures can be hedged to take advantage of extra information by incorporating a prior

...and so much more!

Takeaway: Learning-Augmented algorithms gives us tools and frameworks to reason about interesting and practical new problems

Exciting time to join the field!

Conclusion

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

(3) **Randomized Data Structures:** Randomized data structures can be hedged to take advantage of extra information by incorporating a prior

...and so much more!

Takeaway: Learning-Augmented algorithms gives us tools and frameworks to reason about interesting and practical new problems

Exciting time to join the field!

- Lots of great work over the last 5-10 years laying the groundwork

Conclusion

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

(3) **Randomized Data Structures:** Randomized data structures can be hedged to take advantage of extra information by incorporating a prior

...and so much more!

Takeaway: Learning-Augmented algorithms gives us tools and frameworks to reason about interesting and practical new problems

Exciting time to join the field!

- Lots of great work over the last 5-10 years laying the groundwork
- Seeing the payoff in new results that take advantage of a new ways of algorithmic thinking

Conclusion

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

(3) **Randomized Data Structures:** Randomized data structures can be hedged to take advantage of extra information by incorporating a prior

...and so much more!

Takeaway: Learning-Augmented algorithms gives us tools and frameworks to reason about interesting and practical new problems

Exciting time to join the field!

- Lots of great work over the last 5-10 years laying the groundwork
- Seeing the payoff in new results that take advantage of a new ways of algorithmic thinking
- Join us!

Conclusion

(1) **Repeated Computations:** Sequences of related instances of a problem can be solved faster than one at a time

(2) **Dynamic Algorithms/Data Structures:** Dynamic problems are easier with information about future updates

(3) **Randomized Data Structures:** Randomized data structures can be hedged to take advantage of extra information by incorporating a prior

...and so much more!

Takeaway: Learning-Augmented algorithms gives us tools and frameworks to reason about interesting and practical new problems

Exciting time to join the field!

- Lots of great work over the last 5-10 years laying the groundwork
- Seeing the payoff in new results that take advantage of a new ways of algorithmic thinking
- Join us!

THANKS!

vaidehi@u.northwestern.edu