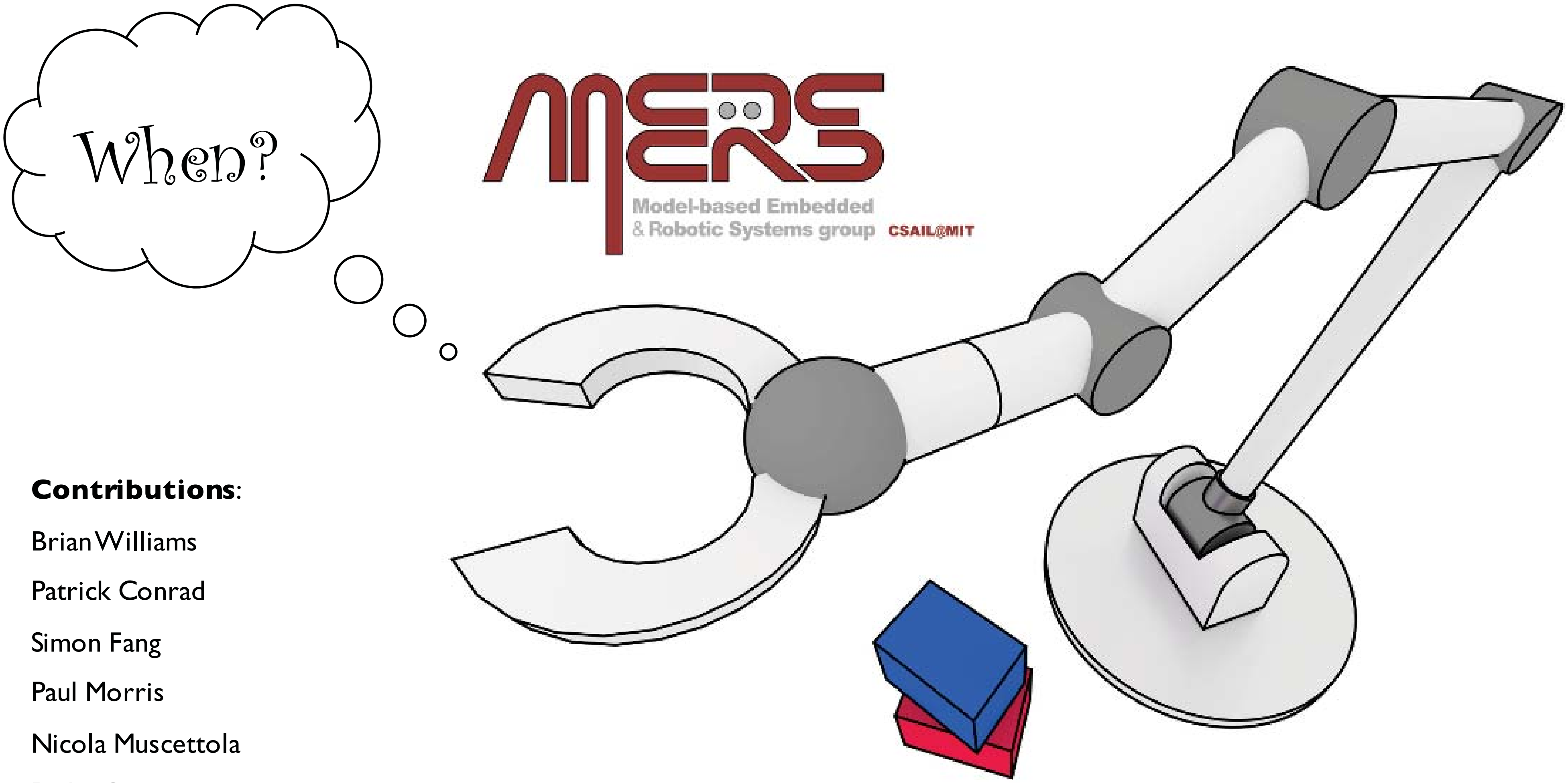
Programs with Flexible Time



Contributions:

Brian Williams

Patrick Conrad

Simon Fang

Paul Morris

Nicola Muscettola

Pedro Santana

Julie Shah

John Stedl

Andrew Wang

Steve Levine

Tuesday, Feb 16th

courtesy of JPL

Assignments

Problems Sets:

- Pset 1 due tomorrow (Wednesday) at 11:59pm

- Pset 2 released tomorrow

Interesting references:

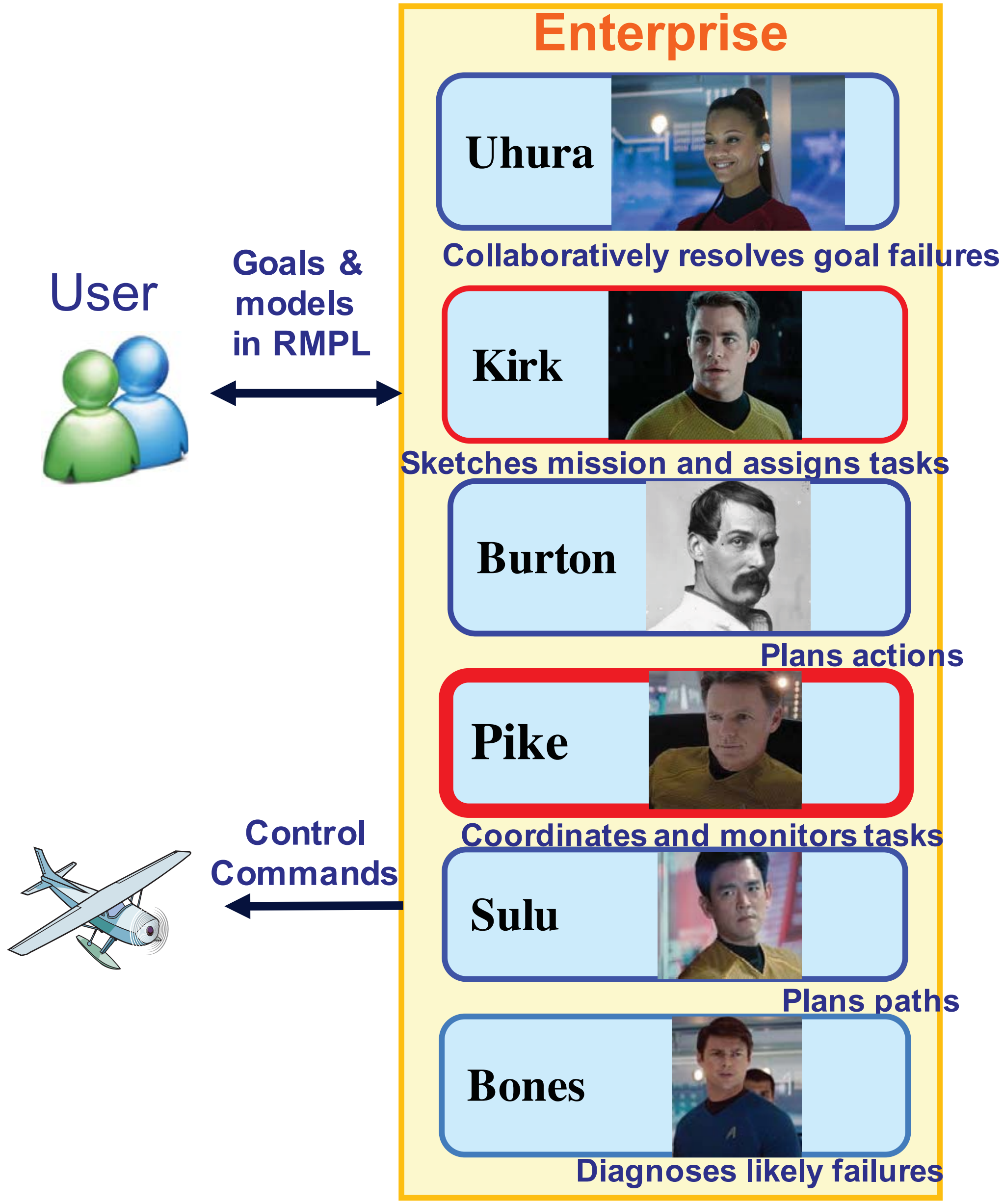
- Dechter, R., I. Meiri, J. Pearl, "Temporal Constraint Networks," Artificial Intelligence, 49, pp. 61-95,1991.

- Muscettola, N., P. Morris and I. Tsamardinos, "Reformulating Temporal Plans for Efficient Execution." Intl Conf. on Knowledge Representation and Reasoning (KRR), 1998.

Outline

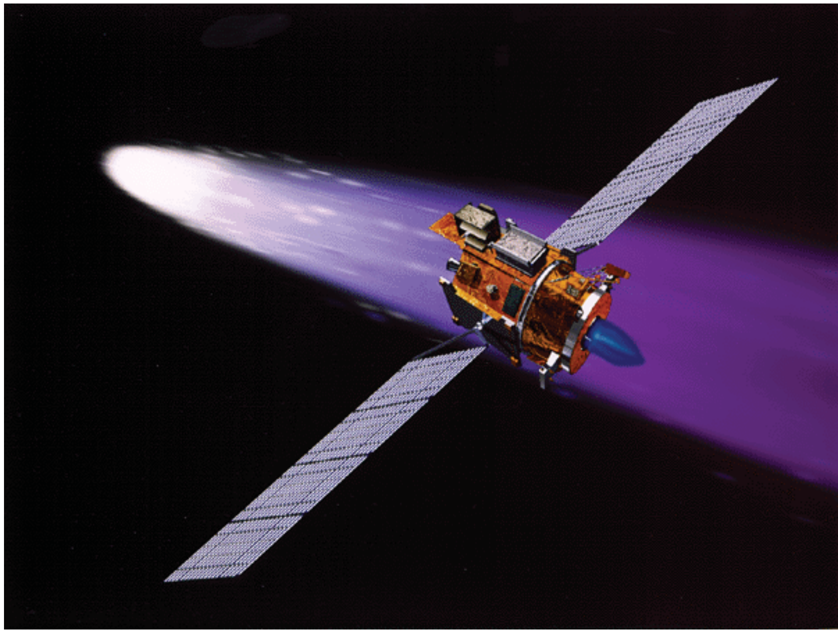
- Programs with Flexible Time
  - Intro
  - Describing temporal plans
  - Exposing implicit constraints
  - Consistency checking
  - Offline scheduling
  - Online execution
  - Reformulating for faster online execution

A single “cognitive system” language and executive.



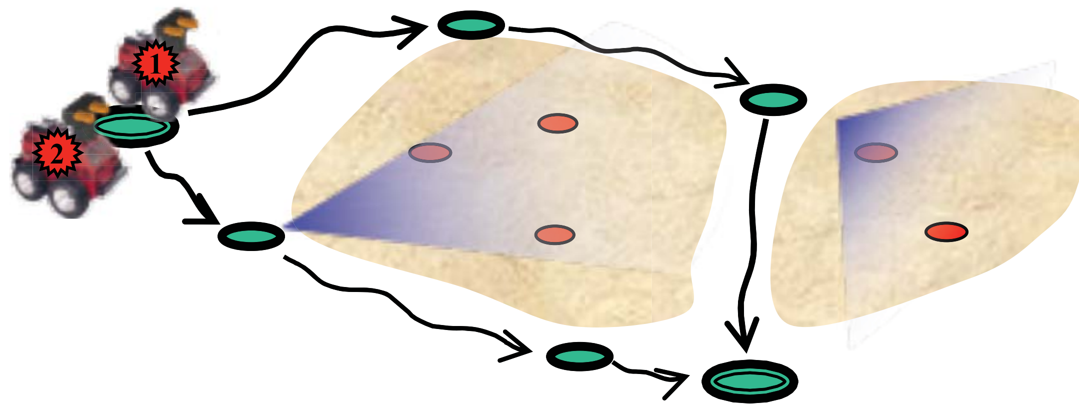
Flexible Time

- Flexible time = more robustness
- We tell cognitive robot:
  - Timing requirements ("engage boosters 2-4 minutes after launch but before reaching orbit")
  - Cognitive robot schedules autonomously.

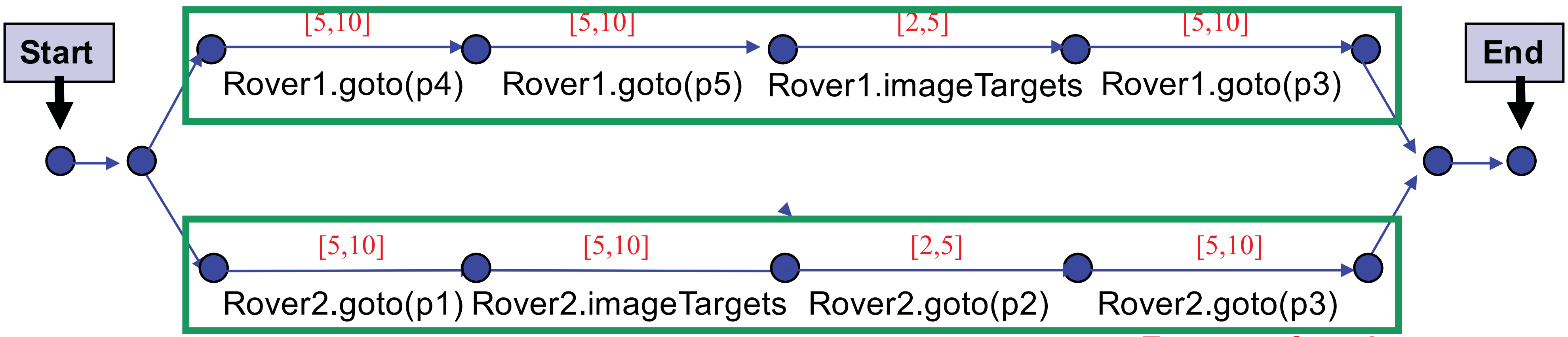


Execution of Timed Model-based Programs

```
imageScienceTargets(Rover1, Rover2)
{Parallel
  {Sequence
    [5,10] Rover1.goto(p4);
    [5,10] Rover1.goto(p5);
    [2,5] Rover1.imageTargets();
    [5,10] Rover1.goto(p3);
  },
  {Sequence
    [5,10] Rover2.goto(p1);
    [5,10] Rover2.imageTargets();
    [2,5] Rover2.goto(p2);
    [5,10] Rover2.goto(p3);
  }
}
```



in RMPL [williams et al 01]



Decisions include what, how, who and when. Focus of today

Agents adapt to temporal disturbances in a coordinated manner by scheduling the start of activities on the fly. [Muscettola, Morris, Tsamardinos, KR 98]

To Execute a Temporal Plan

Schedule Offline

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan

offline

online

4. Execute Plan

Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan

offline

online

4. Dynamically Schedule Plan

To Execute a Temporal Plan

Schedule Offline

Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan

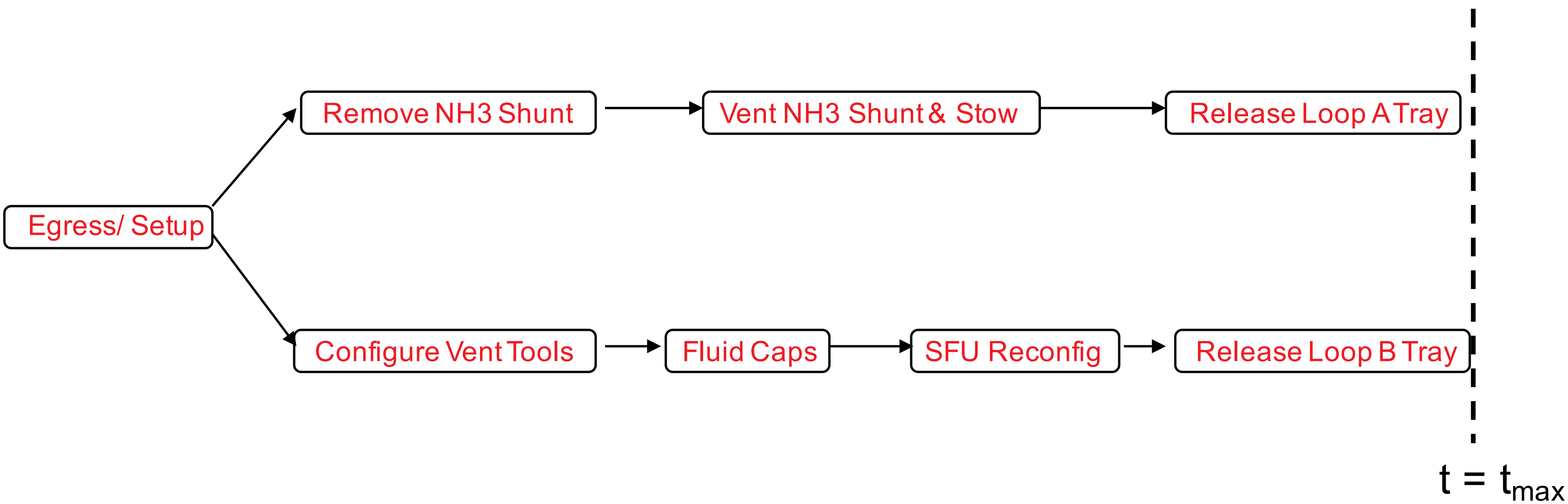
offline

online

4. Execute Plan

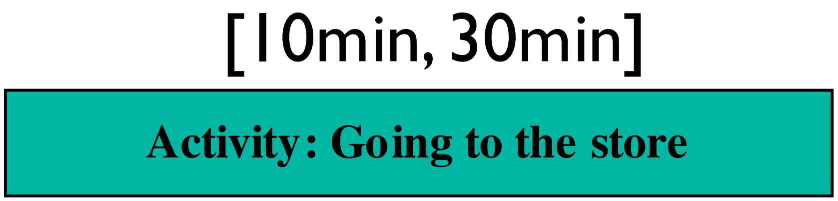
Describe Temporal Plan

- Activities to perform
- Relationships among activities

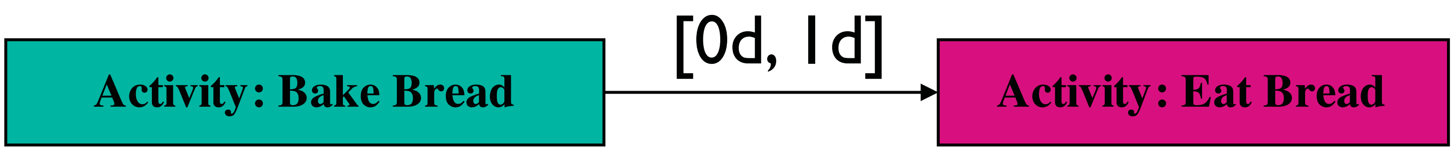


Metric Temporal Relations

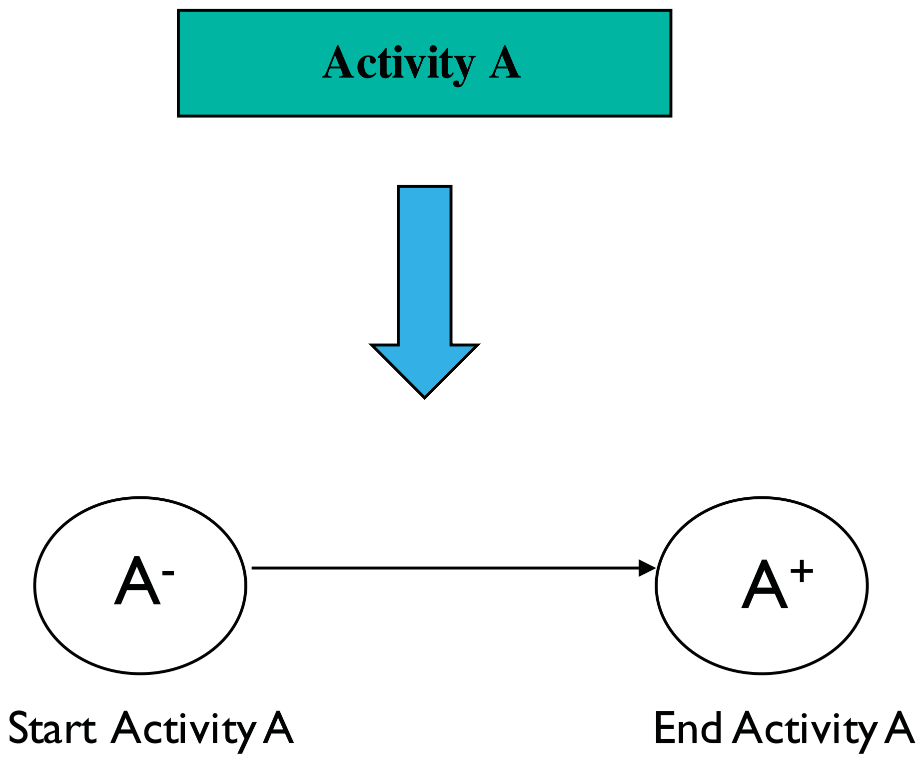
- Going to the store takes at least 10 min and at most 30 min.



- Bread should be eaten within one day of baking.

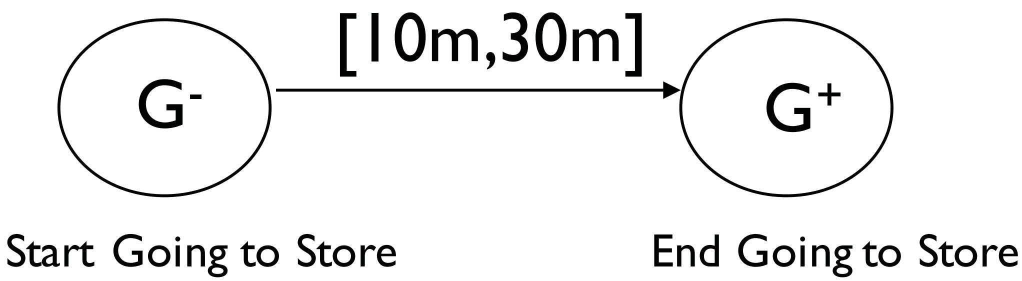


Simplify by reducing interval relations to relations on timepoints.



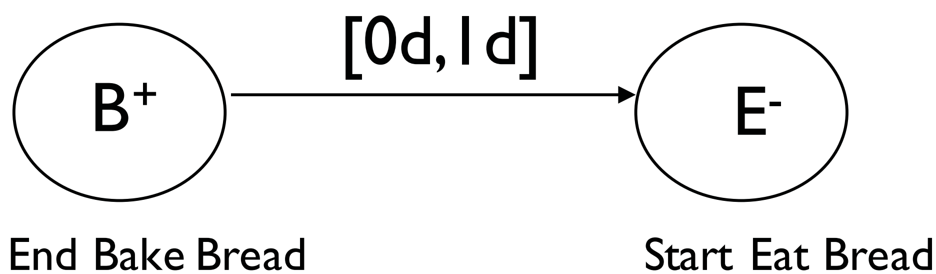
Metric Temporal Relations

- Going to the store takes at least 10 min and at most 30 min.



$10 \leq [G^{+} - G^{-}] \leq 30$

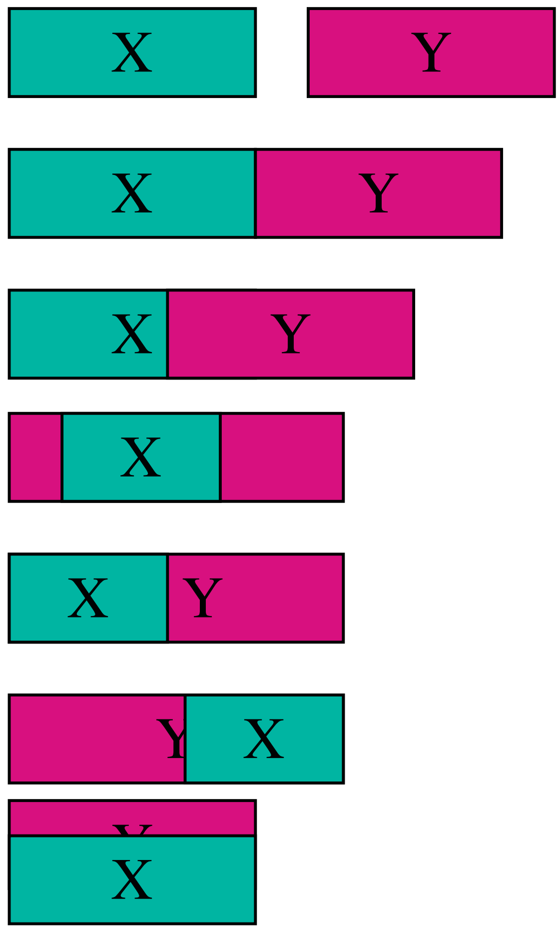
- Bread should be eaten within one day of baking.



$0 \leq [E^{-} - B^{+}] \leq 1$

Qualitative Temporal Relations

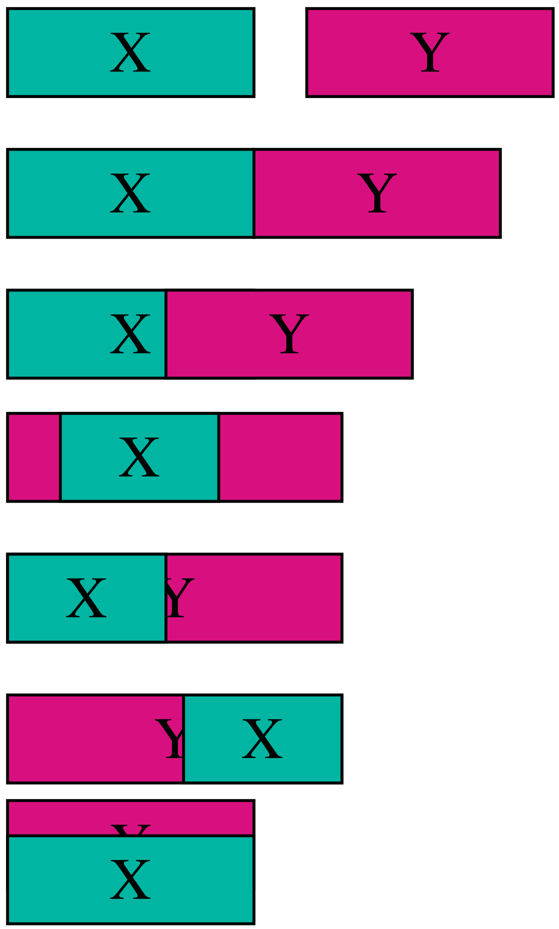
[Allen 83]

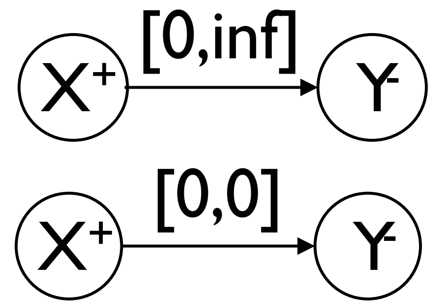


X before Y	Y after X
X meets Y	Y met-by X
X overlaps Y	Y overlapped-by X
X during Y	Y contains X
X starts Y	Y started-by X
X finishes Y	Y finished-by X
X equals Y	Y equals X
X disjoint Y	

Qualitative Temporal Relations

Expressed as timepoint inequalities:





X before Y	$X^+ < Y^-$
X meets Y	$X^+ = Y^-$
X overlaps Y	$Y^- < X^+$ and $X^- < Y^+$
X during Y	$Y^- < X^-$ and $X^+ < Y^+$
X starts Y	$X^- = Y^-$ and $X^+ < Y^+$
X finishes Y	$X^- < Y^-$ and $X^+ = Y^+$
X equals Y	$X^- = Y^-$ and $X^+ = Y^+$
X disjoint Y	$X^+ < Y^-$ or $Y^+ < X^-$

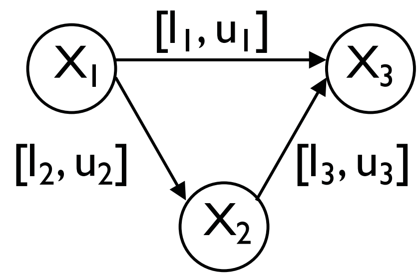
[Villain & Kautz; Simmons]

Temporal Relations Described by a Simple Temporal Network (STN)

- Simple Temporal Network [Dechter, Meiri, Pearl 91]
  - Tuple <X, C> where:
  - variables $X_1, \ldots X_n$, represent time points (real-valued domains)
  - binary constraints C of the form:

$$(X_k - X_i) \in [a_{ik}, b_{ik}]$$

    – called *links*.



Sufficient to represent:
• simple metric constraints
• all Allen relations but 1…

Can't represent:
• Allen's disjoint relation

Modeling Visualization

- http://bicycle.csail.mit.edu/stn/

To Execute a Temporal Plan

Schedule Offline

1. Describe Temporal Plan

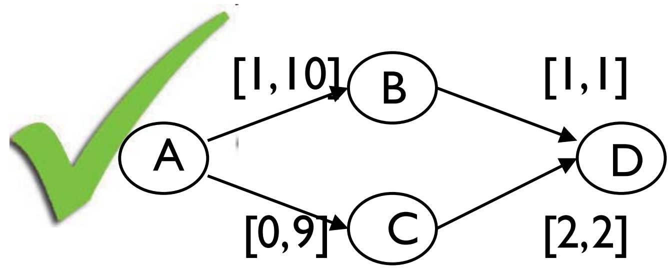
2. Test Consistency

3. Schedule Plan

offline

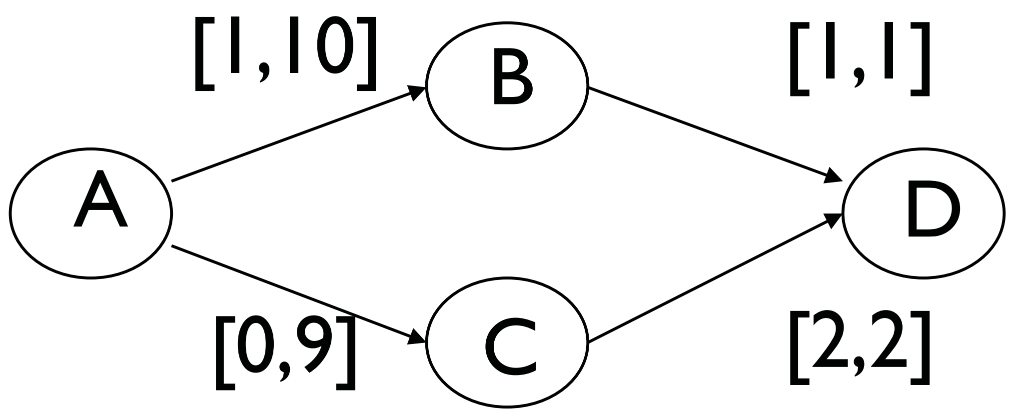
online

4. Execute Plan



Consistency of an STN

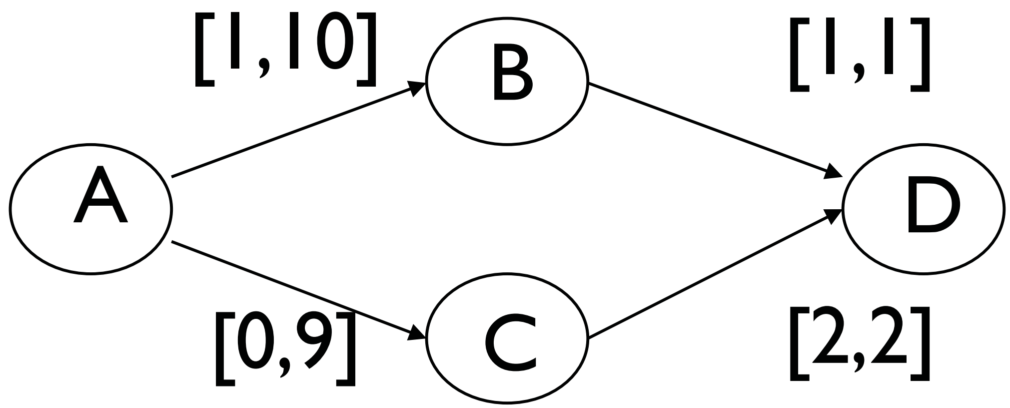
Input: STN <X, C> where $C_j = \Big< \langle X_k, X_i \rangle, \ \langle a_j, b_j \rangle \Big>$



STN is ***consistent*** iff there exists an assignment to times X satisfying C.

Schedule of an STN

Input: STN <X, C> where $C_j = \Big< \langle X_k, X_i \rangle, \langle a_j, b_j \rangle \Big>$



Schedule is assignment to all timepoints X consistent with constraints.

How to find schedule?

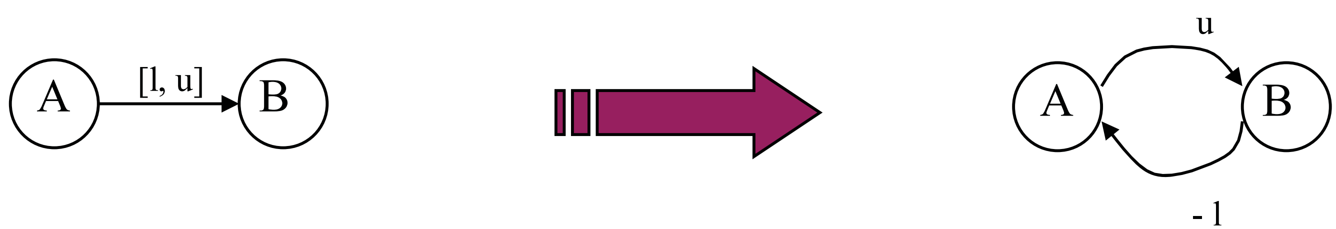
- Idea: Transform STN
  - Transform to ***distance-graph***
  - Common graph algorithms (i.e., shortest path) will apply

Transformation to distance graph

- (Board)

Map STN to Distance (D) Graph

Simple Temporal Network　　　　Distance Graph



$l \leq B - A \leq u$

$B - A \leq u$

$l \leq B - A$　　　　$A - B \leq -l$

- Upperbound mapped to outgoing, non-negative arc.
- Lowerbound mapped to incoming, negative arc.

[Dechter, Meiri, Pearl 91]

Algorithm: offline scheduling

Initialize execution window to [-∞, ∞] for each event

while unexecuted events:

    x_i = pick any unexecuted event

    t_i = pick any time in x_i's execution window

    Propagate to all x_i's neighbors & update their windows

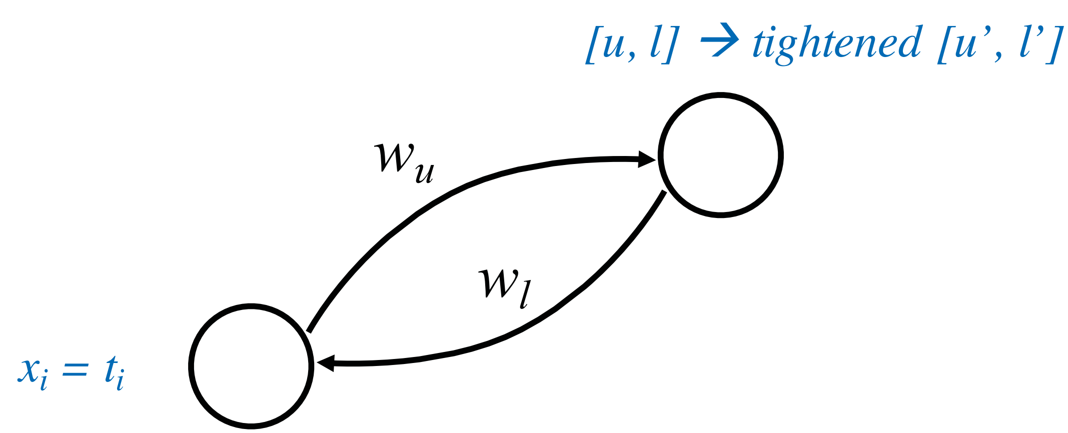
Naïve (and wrong) scheduling

- (Board)

Propagating to neighbors

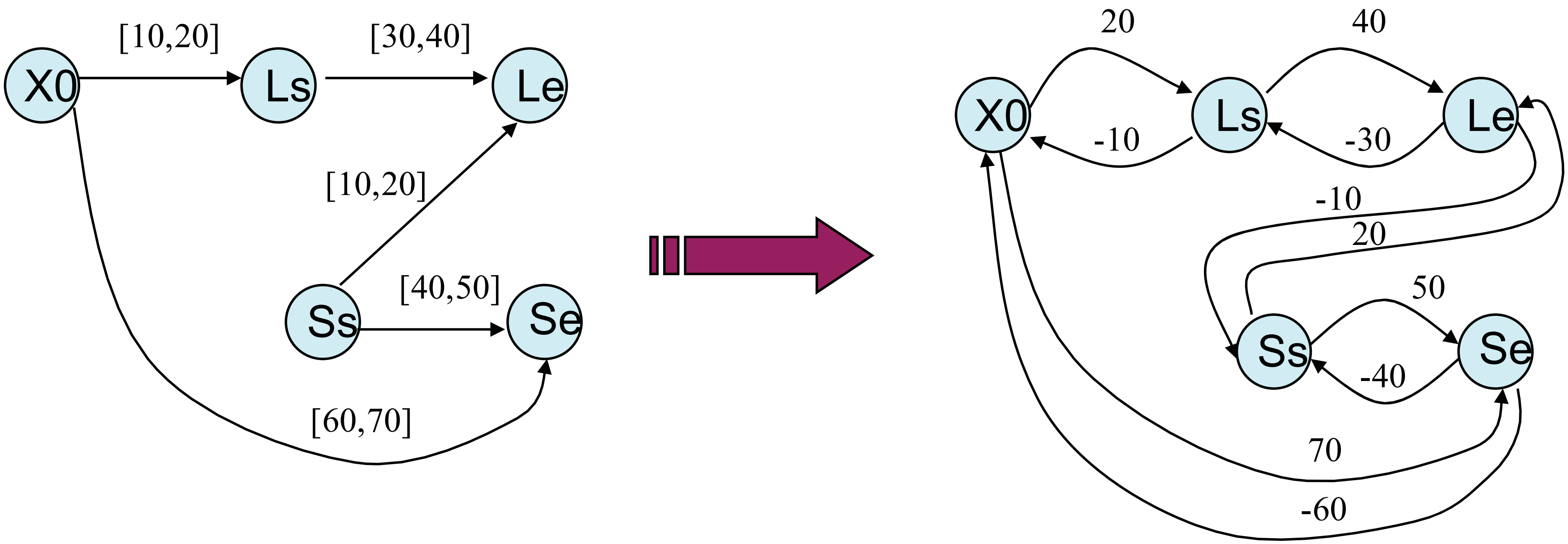
Tighten neighbor's execution windows:

- outgoing edges to neighbor: $u' = \min(u, t_i + w_u)$
- incoming edges from neighbor: $l' = \max(l, t_i - w_l)$



Exposing Implicit Constraints

- (Board)



[10,20]
[30,40]
X0
Ls
Le
[10,20]
[40,50]
Ss
Se
[60,70]
20
40
X0
Ls
Le
-10
-30
-10
20
50
Ss
Se
-40
70
-60

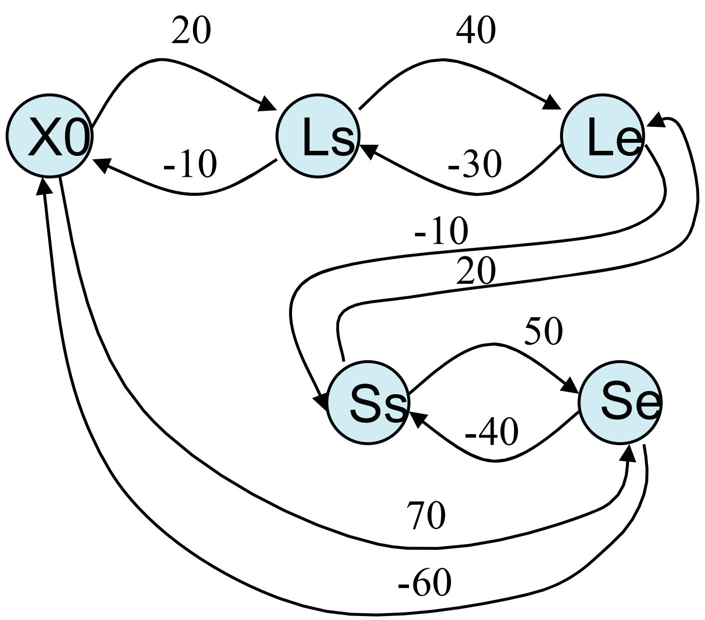
APSP Graph: Windows of Feasible Values

	X_0	Ls	Le	Ss	Se
X_0	0	20	50	30	70
Ls	-10	0	40	20	60
Le	-40	-30	0	-10	30
Ss	-20	-10	20	0	50
Se	-60	-50	-20	40	0

+ Latest Times

- Earliest Times

APSP d-graph



- Ls in [10, 20]
- Le in [40, 50]
- Ss in [20, 30]
- Se in [60, 70]

Dispatchable form

- An STN or distance graph is ***dispatchable*** if:
  - Can be properly scheduled via local propagations to neighbors only
- Requires all implicit constraints be explicit

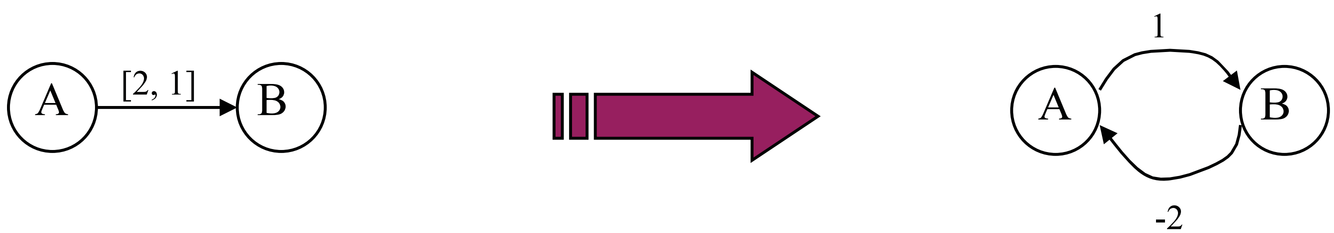
Checking Consistency

- (Board)

Check D-Graph Consistency

- Consistent iff D-graph has no negative cycles.
- Detect by computing shortest path from one event to all others.
  - Single Source Shortest Path (SSSP).
  - Event must reach all others.

Example of inconsistent constraint:



Simple Temporal Network

Distance Graph

Summary

- To schedule, want a simple, local-propagation algorithm
  - Requires exposing implicit constraints
- All-pairs shortest path (APSP) exposes all implicit constraints
  - Puts network in ***dispatchable form***
- Negative cycle in APSP: inconsistent.

To Execute a Temporal Plan

Schedule Offline

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan

offline

- - - - - - - - - - - - - - - - - - - -

online

4. Execute Plan

Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan

offline

- - - - - - - - - - - - - - - - - - - -

online

4. Dynamically Schedule Plan

Algorithm: offline scheduling

Compute dispatchable form (i.e., APSP)

Initialize execution window to [-∞, ∞] for each event

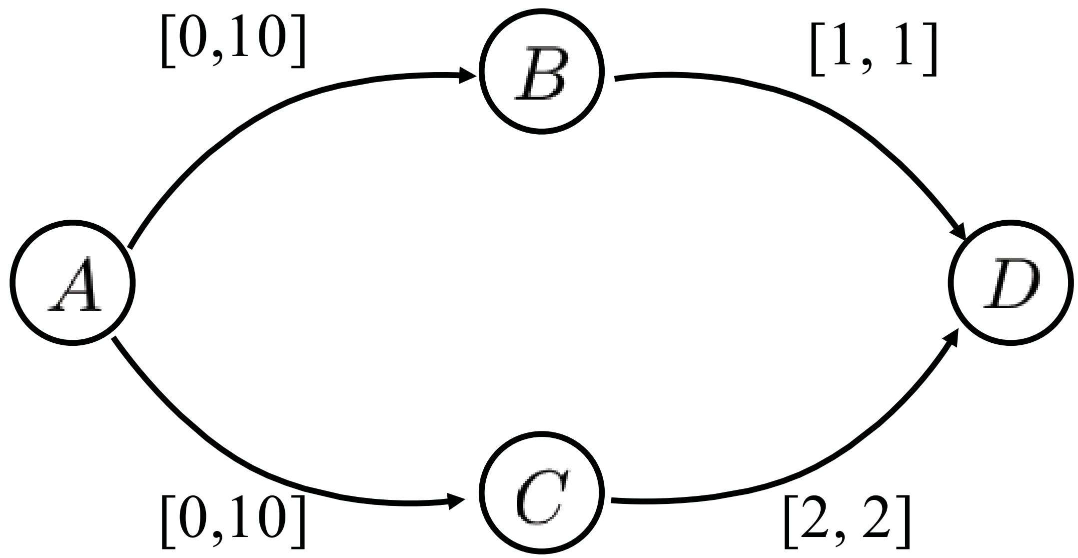
while unexecuted events:

  x_i = pick any unexecuted event

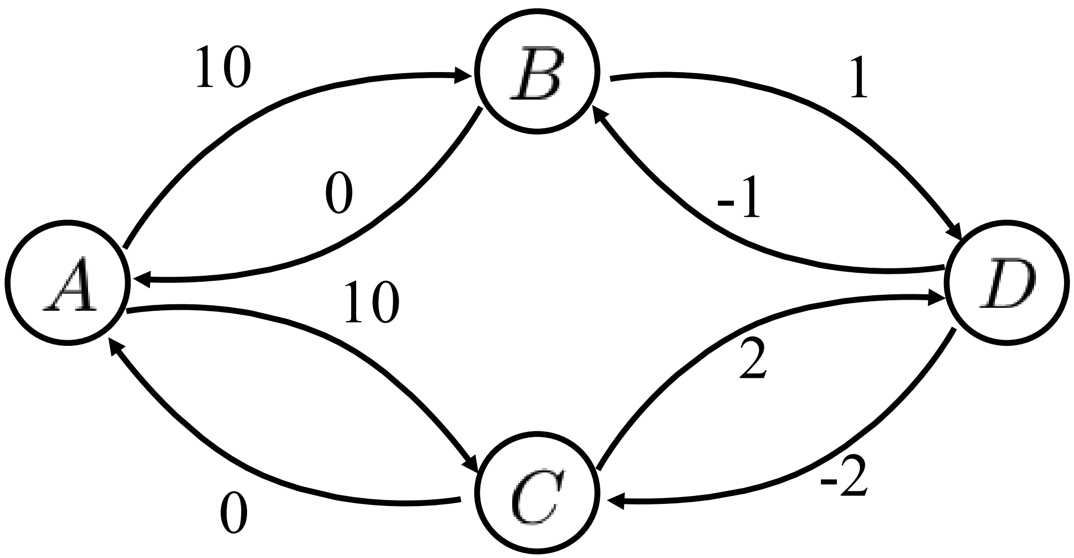
  t_i = pick any time in x_i's execution window

  Propagate to all x_i's neighbors & update their windows

The original STN

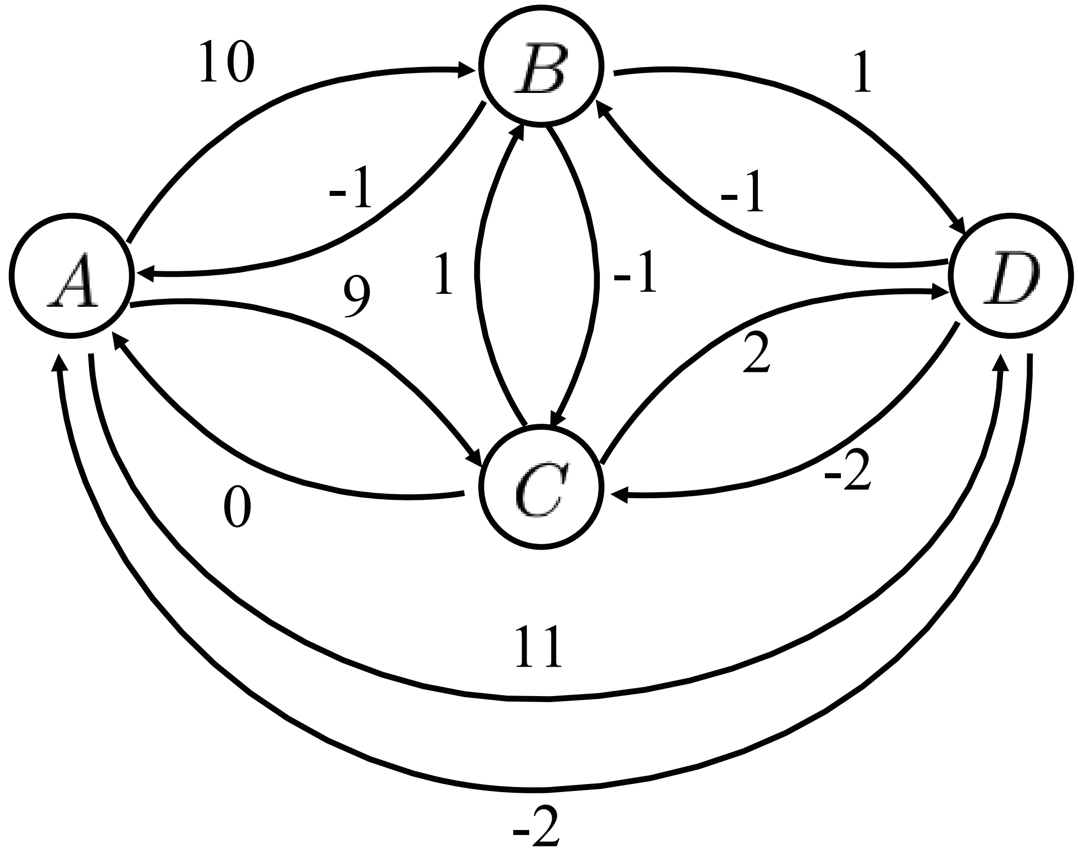


Distance graph transformation

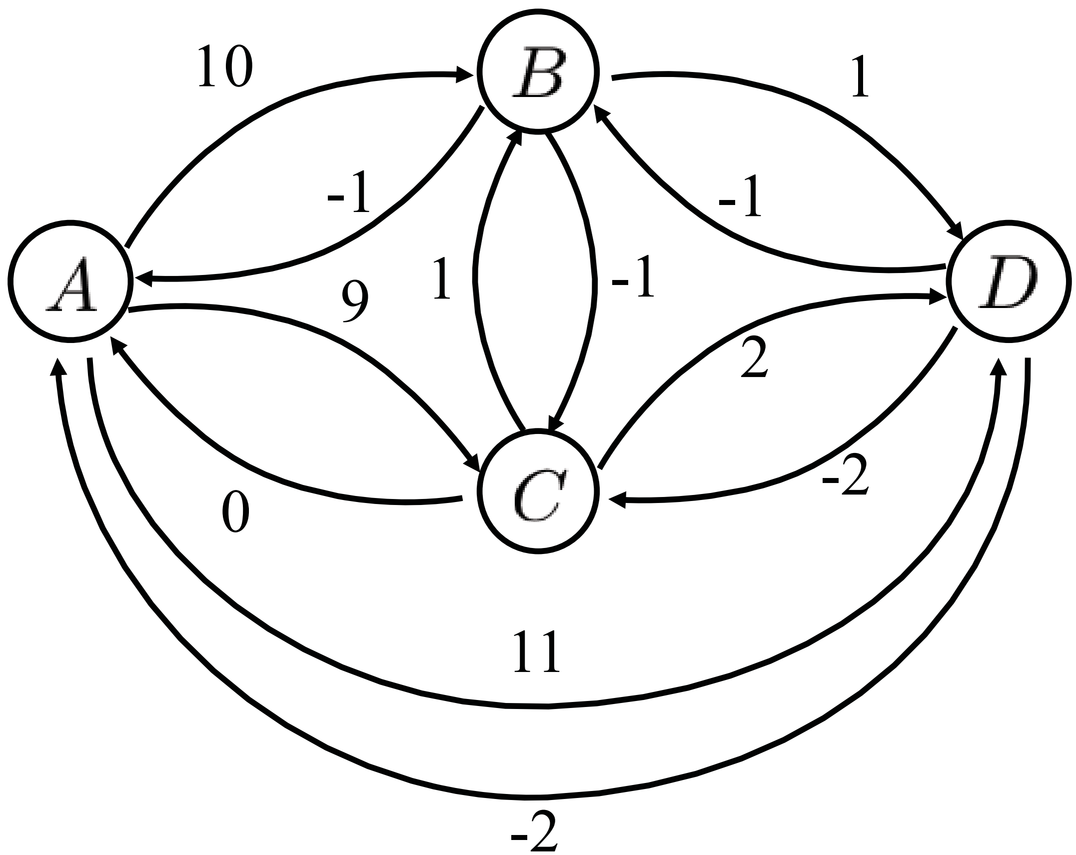


All pairs shortest path

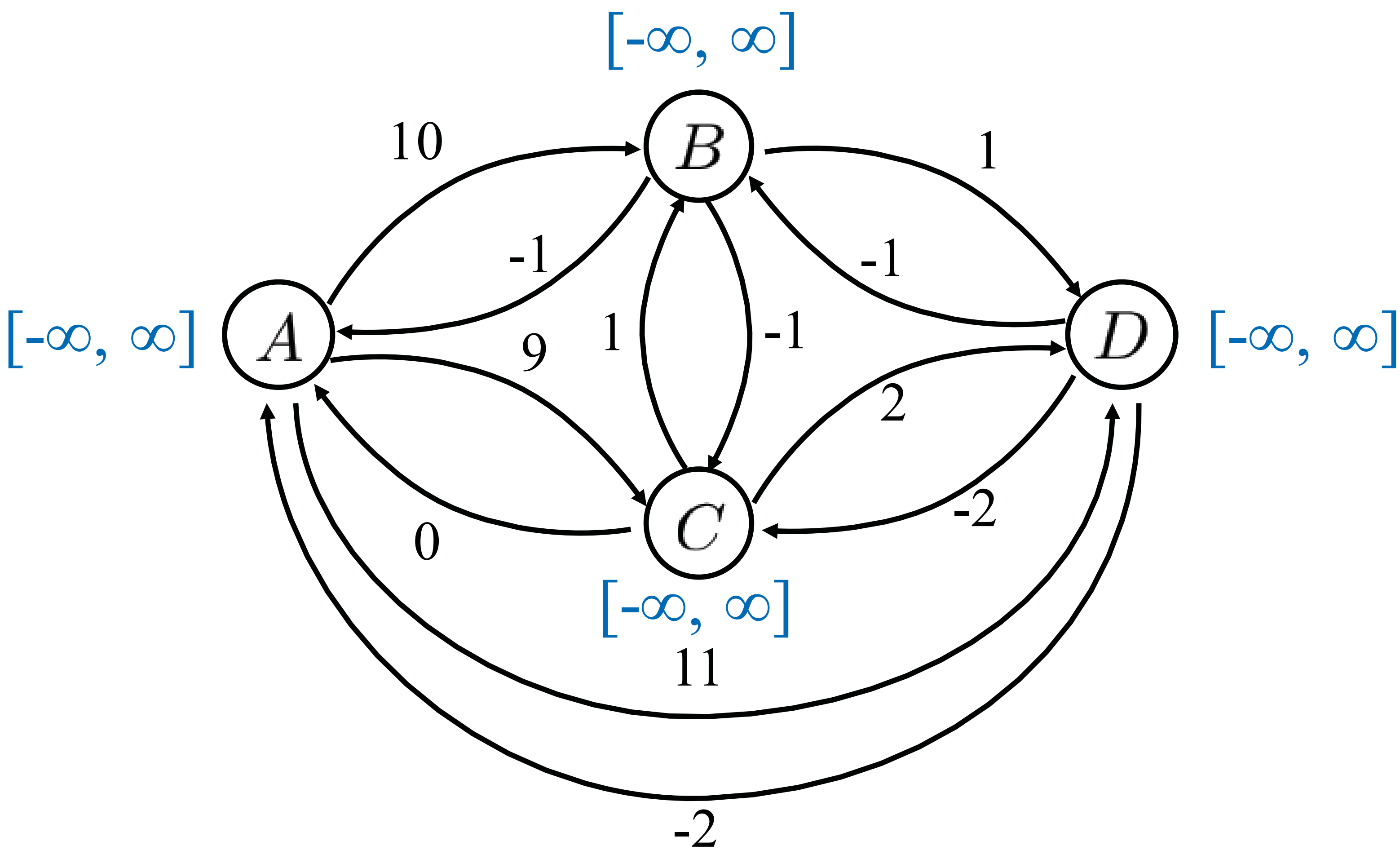
	A	B	C	D
A	0	10	9	11
B	-1	0	-1	1
C	0	1	0	2
D	-2	-1	-2	0



Computing a schedule

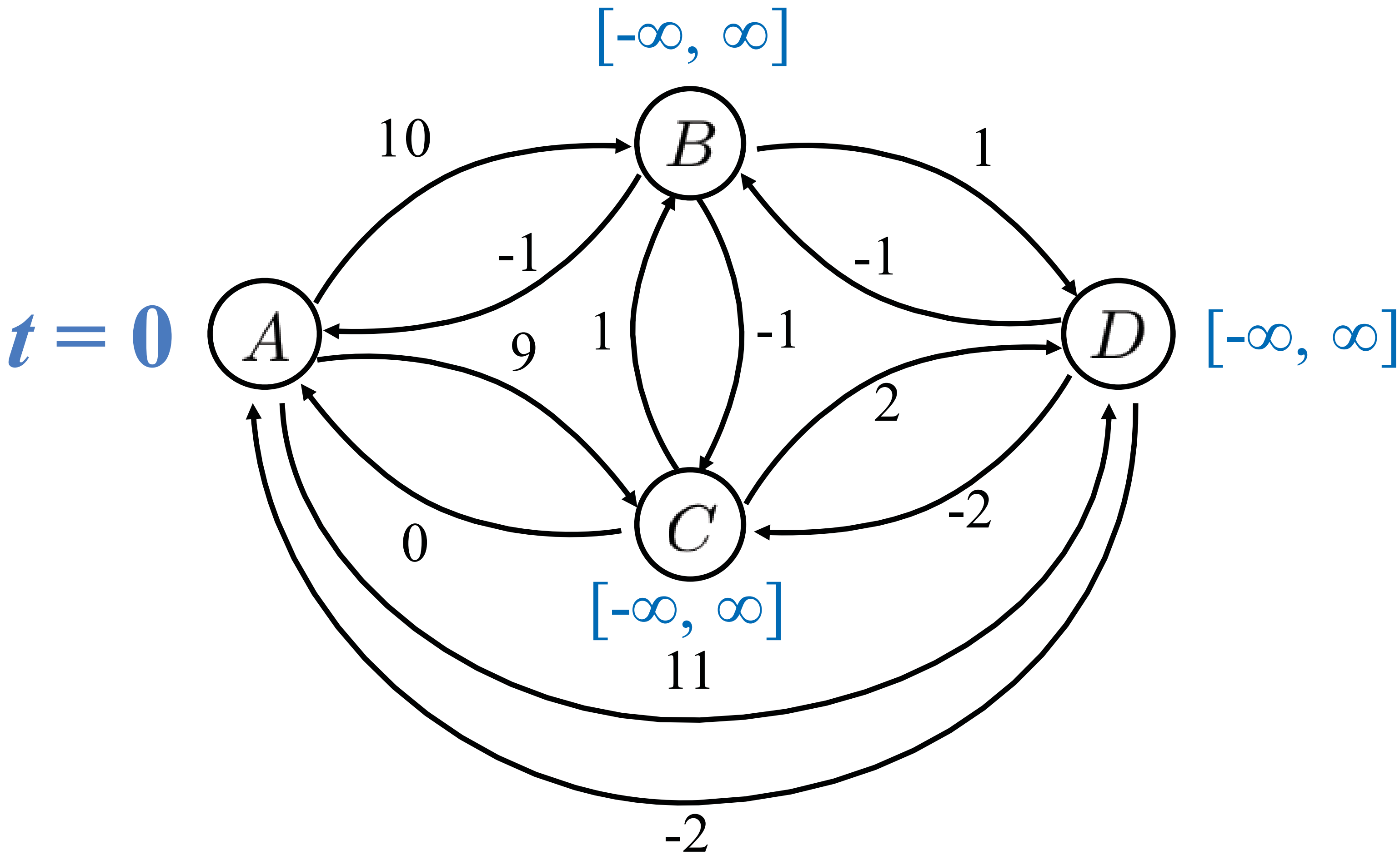


Computing a schedule



Initialize execution windows for each event in the plan

Computing a schedule

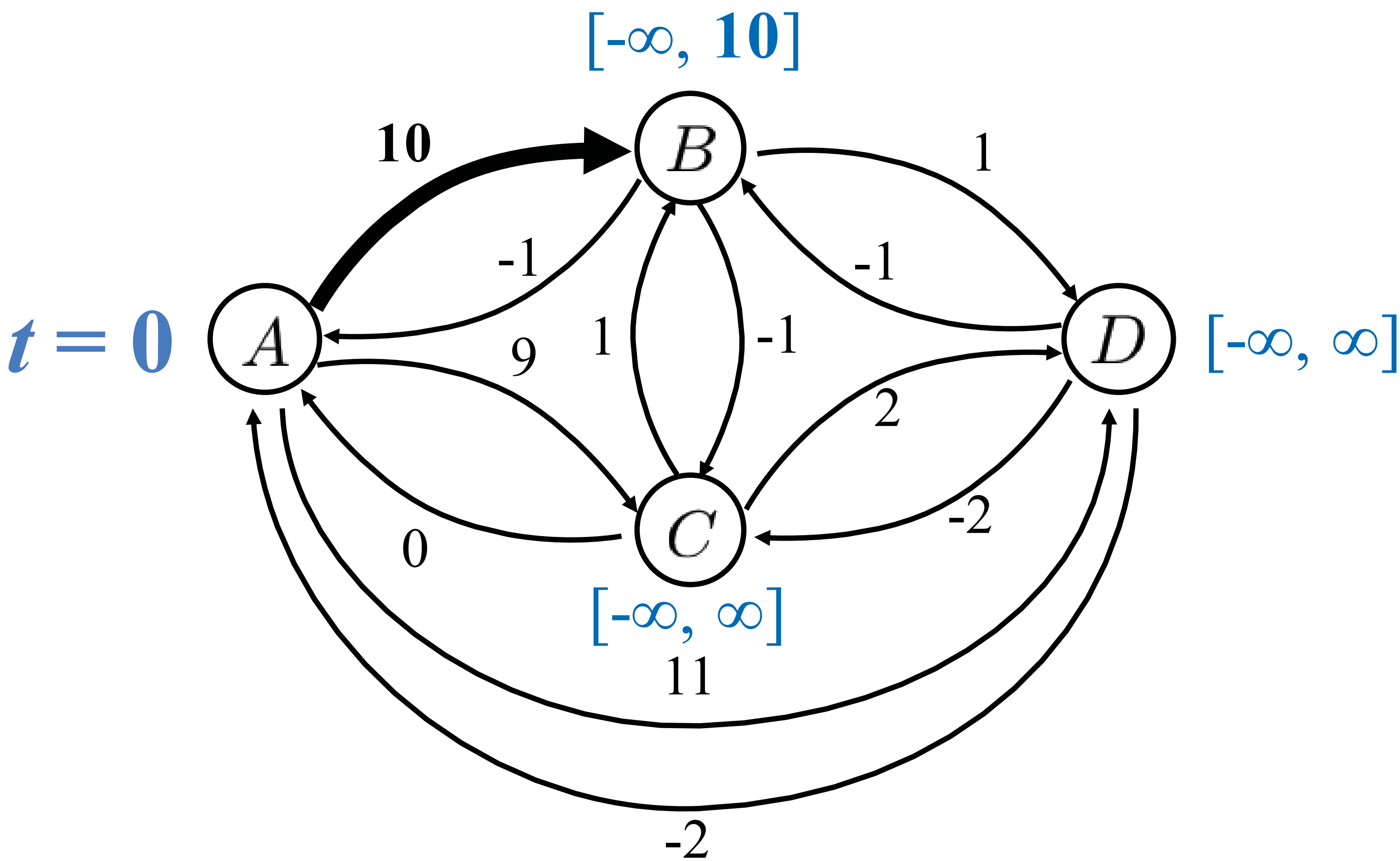


Assign the first event

Computing a schedule

outgoing edges to neighbor: $u' = \min(u, t_i + w_u)$

incoming edges from neighbor: $l' = \max(l, t_i - w_l)$

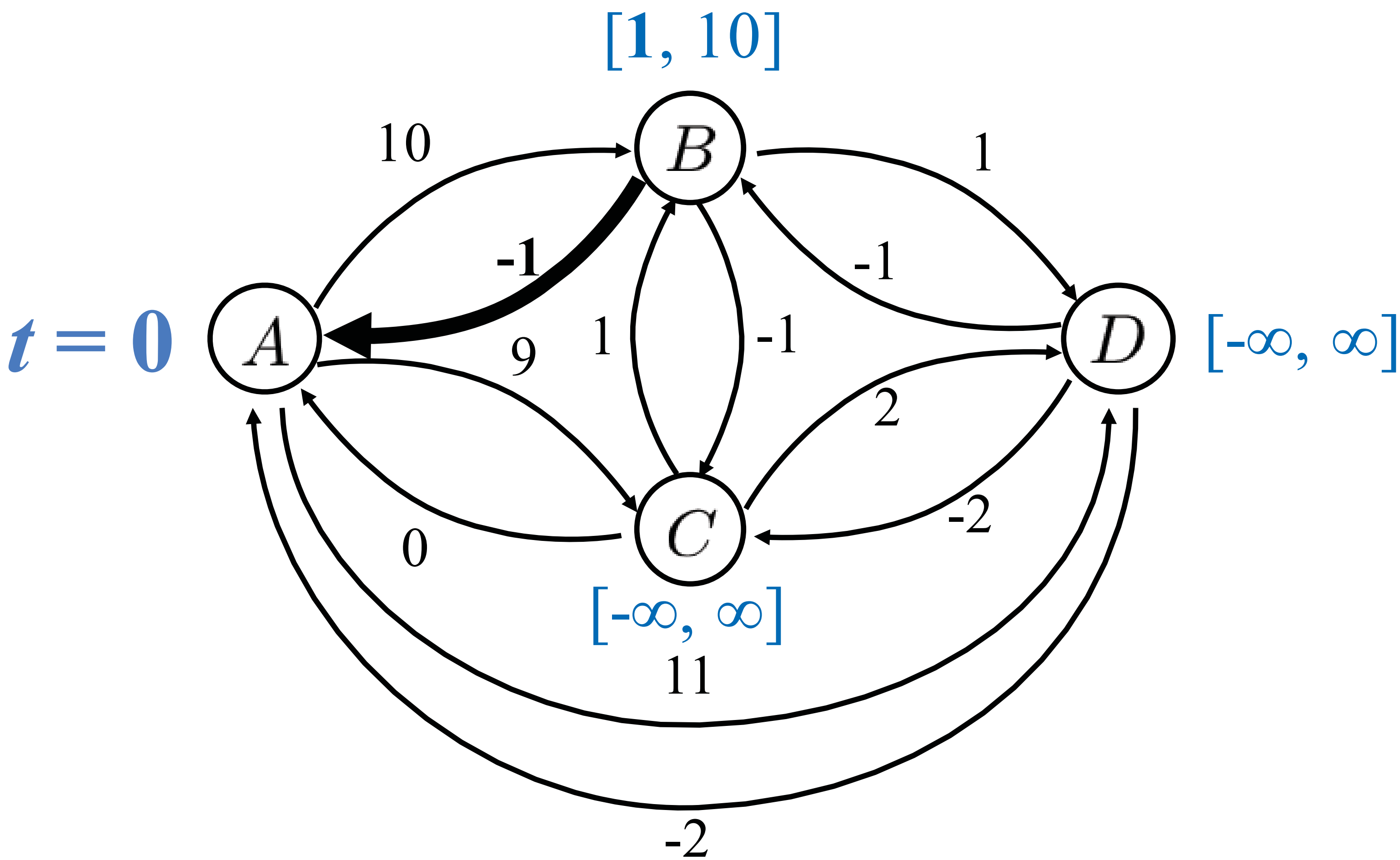


Propagate updated time bounds to neighbors

Computing a schedule

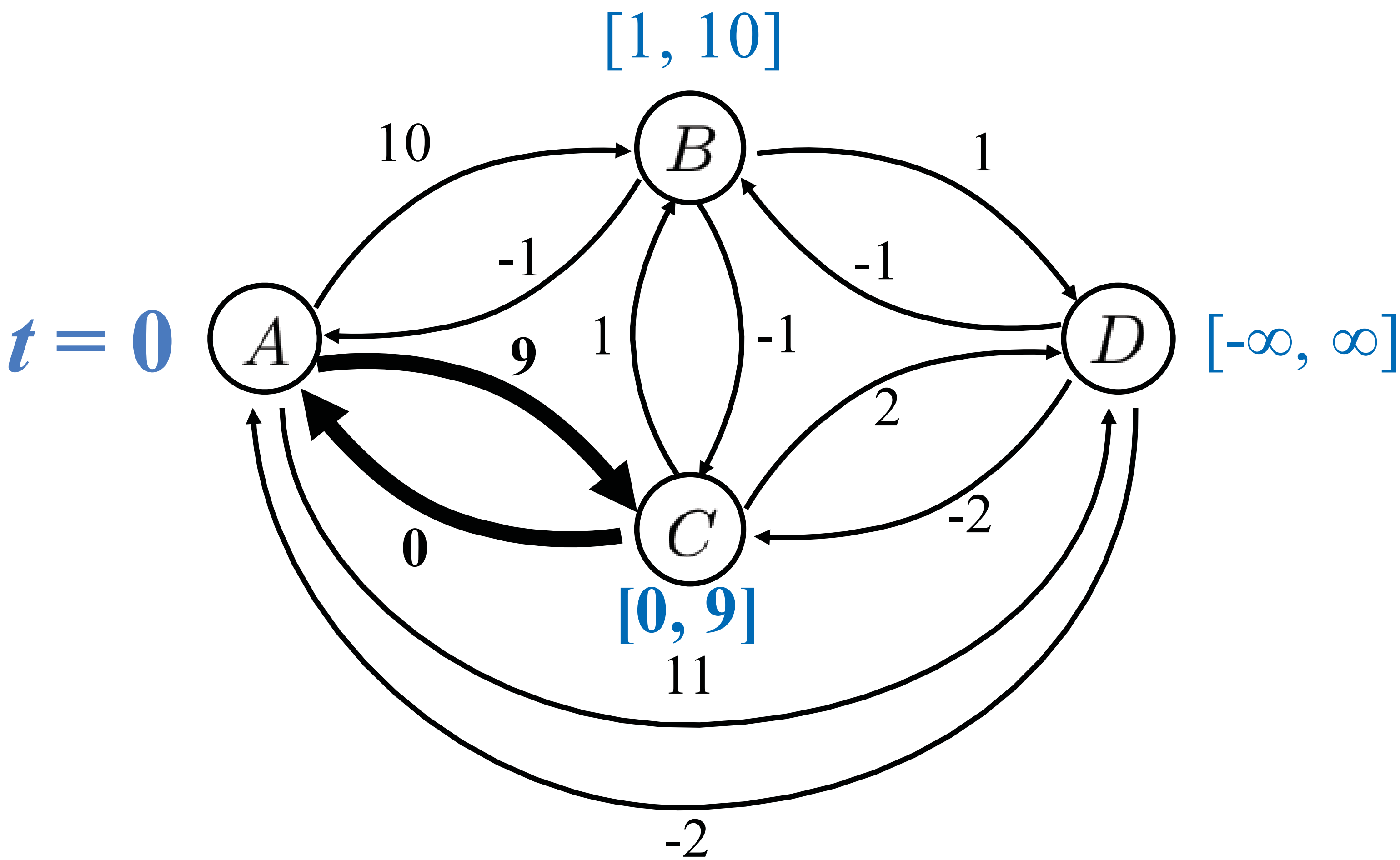
outgoing edges to neighbor: $u' = \min(u, t_i + w_u)$

incoming edges from neighbor: $\boldsymbol{l' = \max(l, t_i - w_l)}$



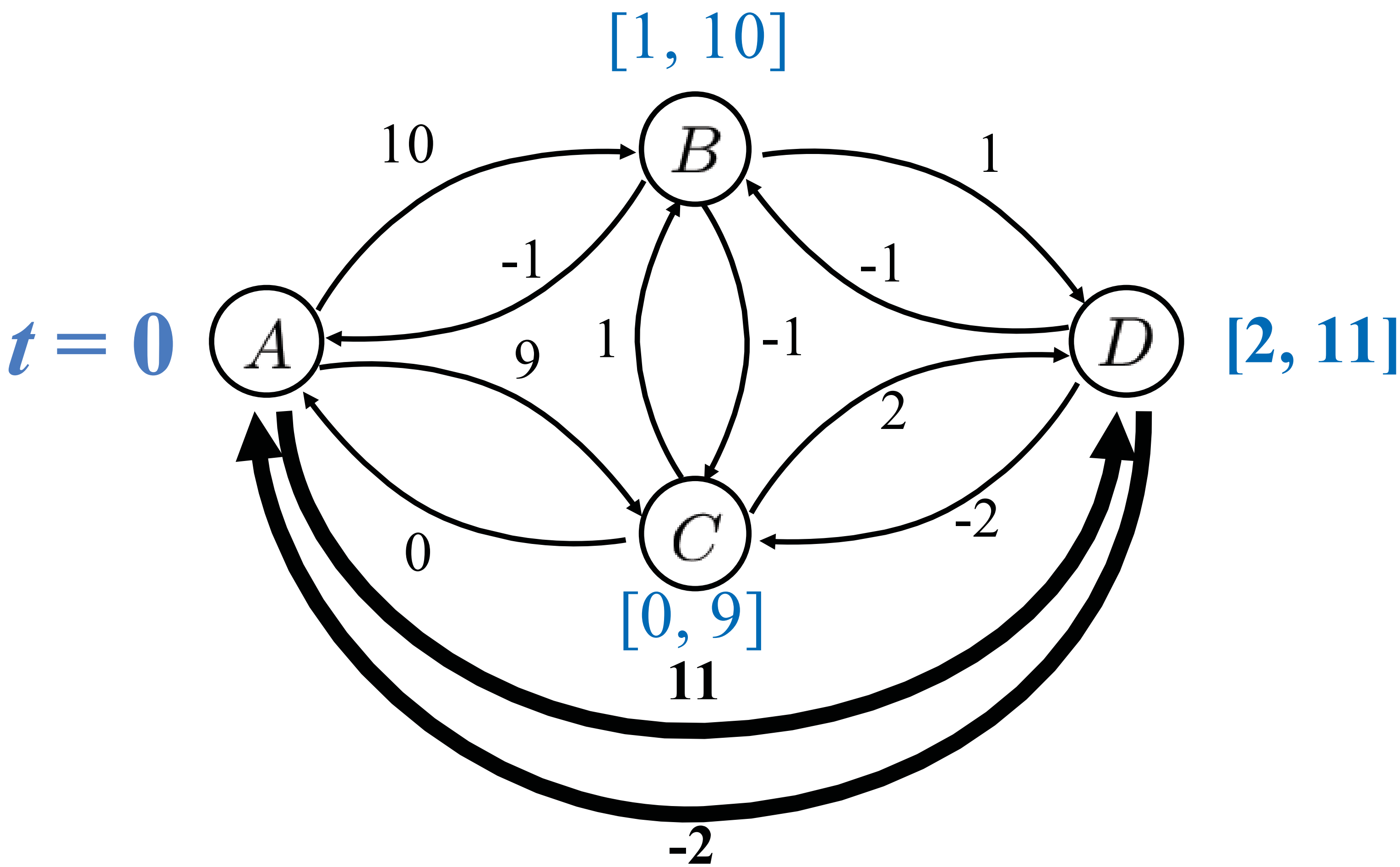
Propagate updated time bounds to neighbors

Computing a schedule



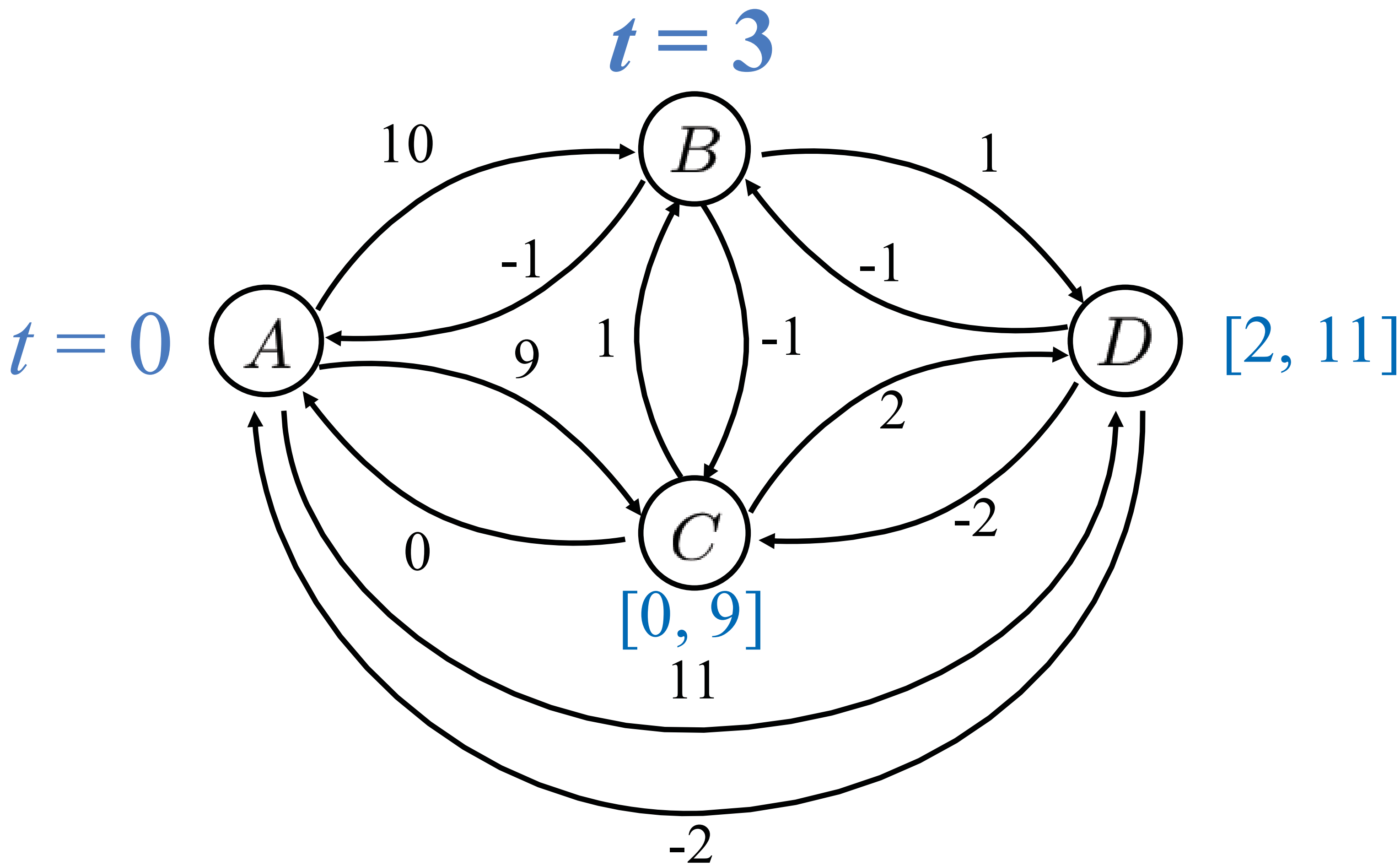
Propagate updated time bounds to neighbors

Computing a schedule



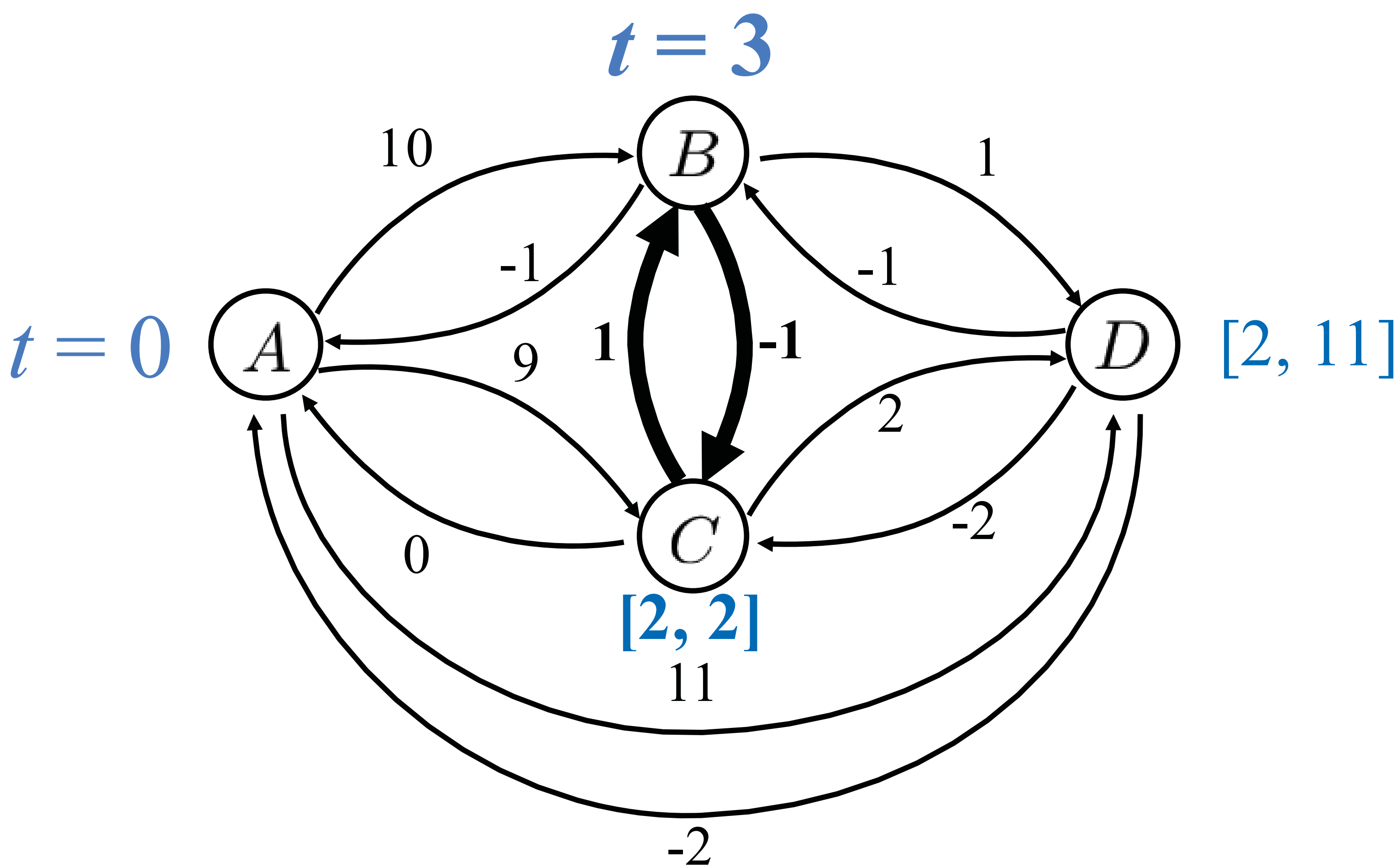
Propagate updated time bounds to neighbors

Computing a schedule



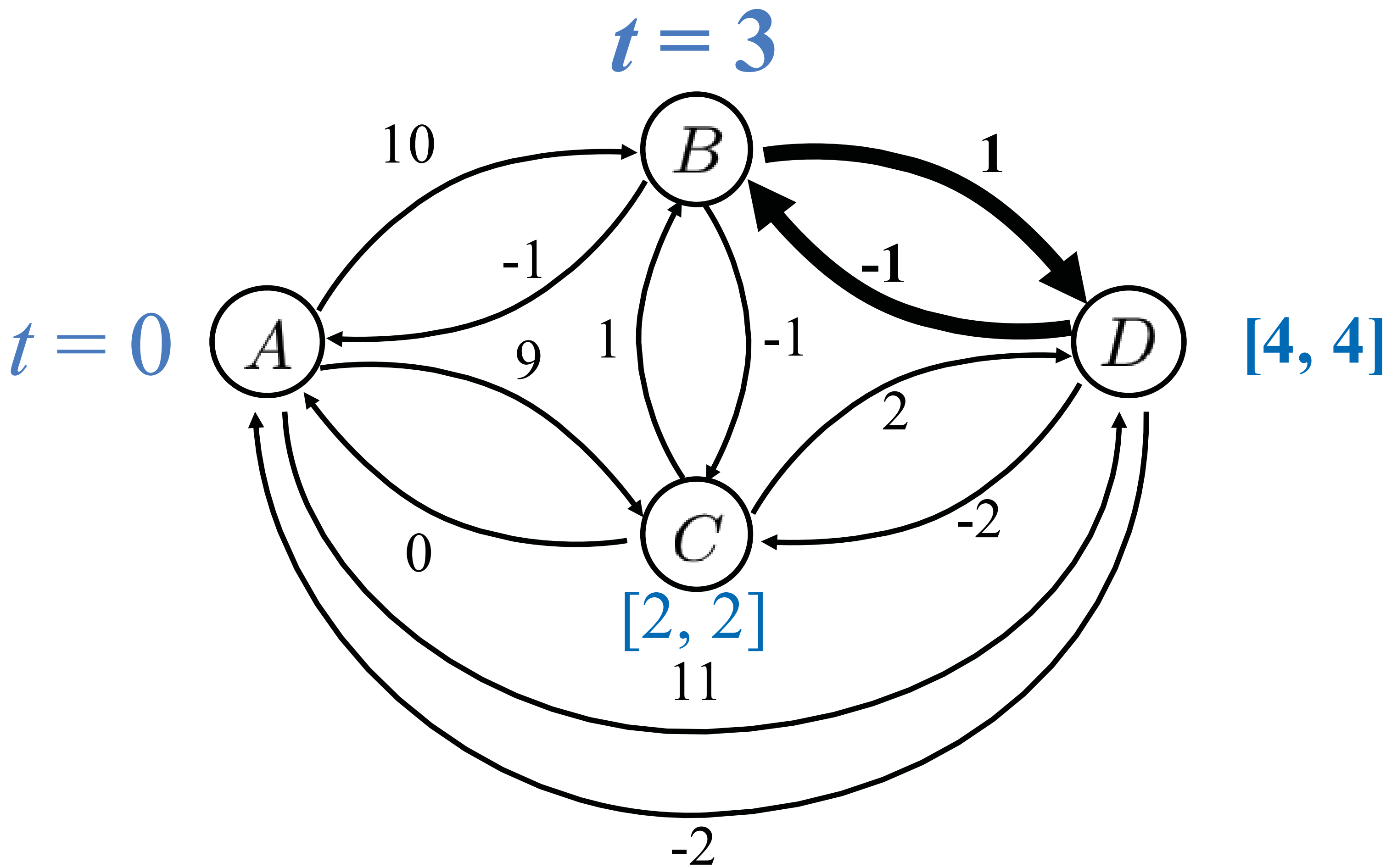
Arbitrarily pick another time point and assign it...

Computing a schedule



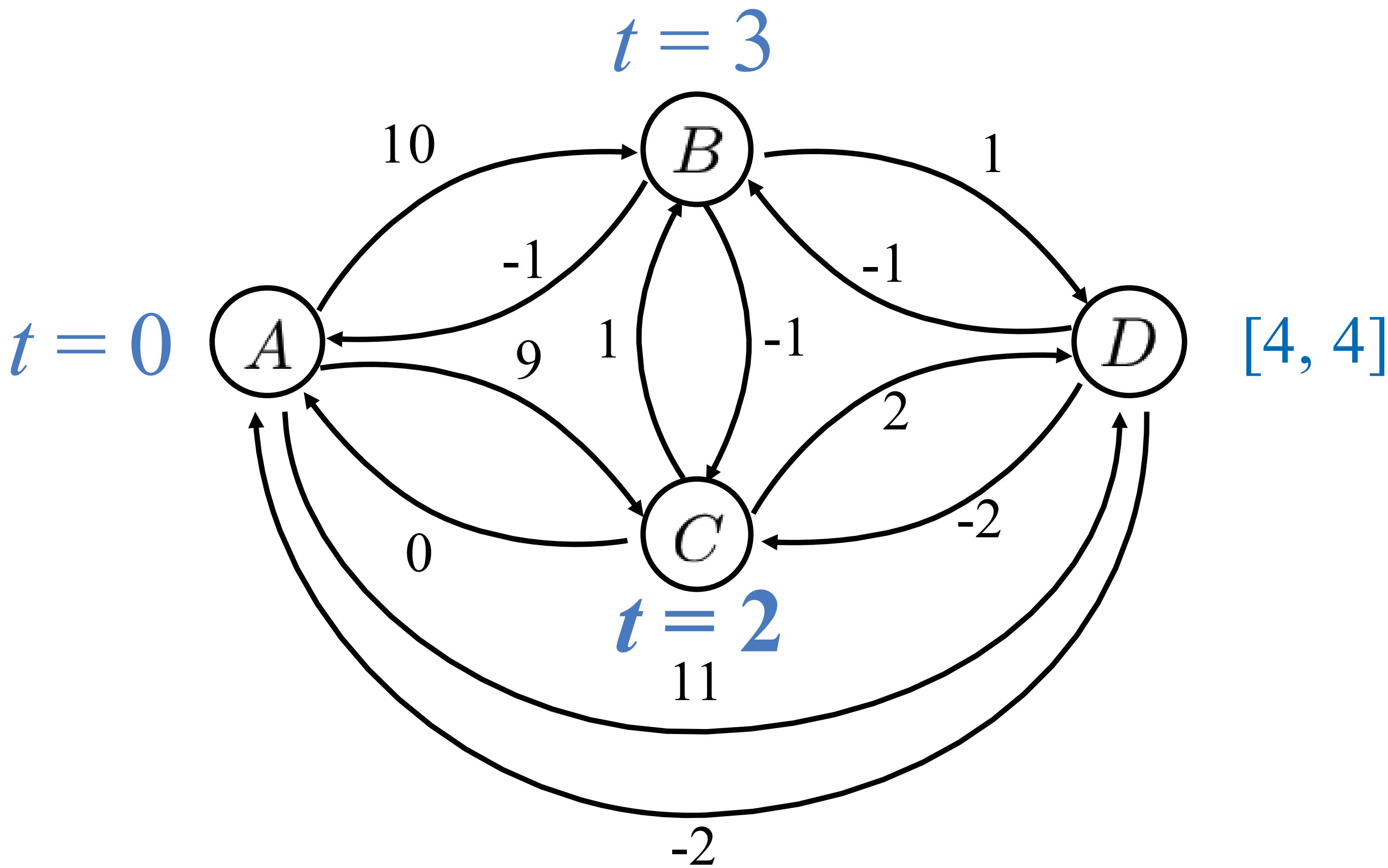
Propagate updated time bounds to neighbors

Computing a schedule



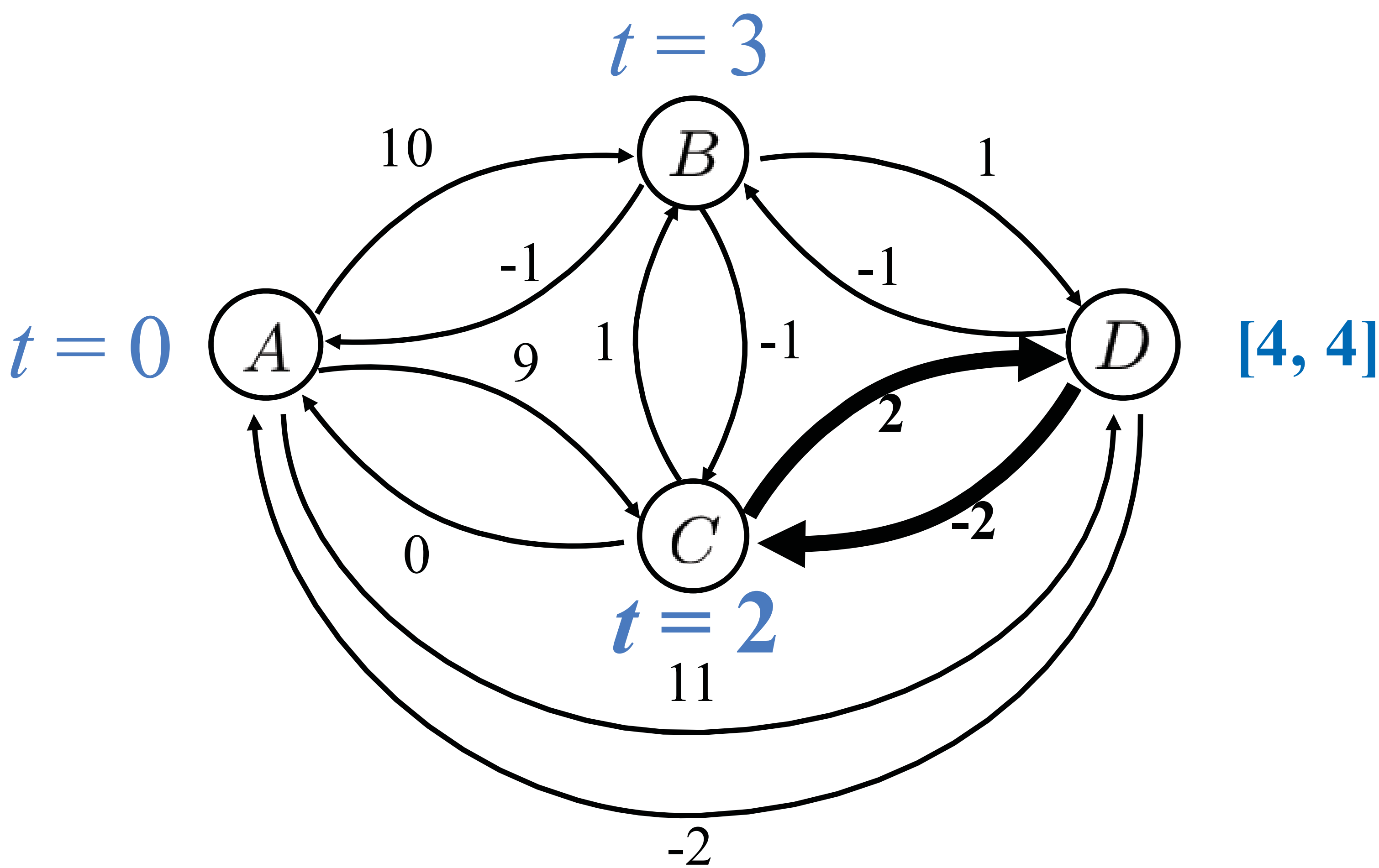
Propagate updated time bounds to neighbors

Computing a schedule



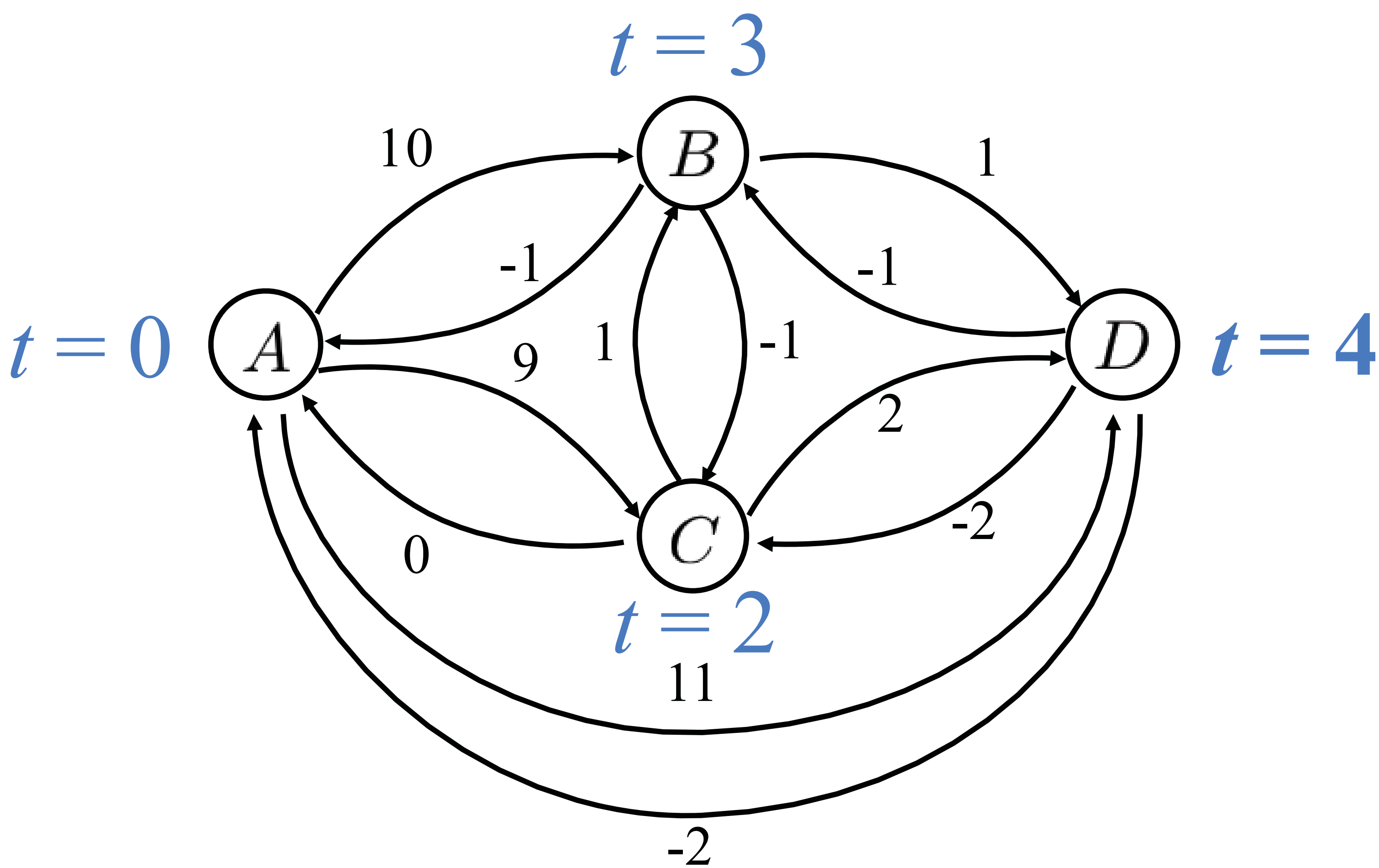
Pick another event and assign it

Computing a schedule



Propagate to neighbors

Computing a schedule



Assign the final event

Pre-computed schedules not robust against fluctuations

- We've just computed a schedule:

$$t_A = 0,\ t_B = 3,\ t_C = 2,\ t_D = 4$$

- But what if there's a disturbance?
  - i.e., what if $t_B = 3.1$?
  - i.e., what if $t_B = 4$?
  - i.e., what if $t_B = 100$?
- Pre-computed schedules not robust against fluctuations!
- **Solution:** Dispatch dynamically online.
  - Schedule events "on the fly," after observing past event times.
  - Increases robustness to many unanticipated fluctuations.
  - Flexible temporal constraints allow this!

To Execute a Temporal Plan

Schedule Offline

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan
4. Execute Plan

Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan
4. Dynamically Schedule Plan

offline

online

How do we schedule online?

- First, consider naive (incorrect!) approach.
- Similar to offline schedule algorithm, but now online:
  - **Wait** until current time in execution window ("active")

- (Still a problem though as we'll see shortly)

Naïve (wrong!) online dispatcher

Compute dispatchable form (i.e., APSP)

Initialize execution window to [-∞, ∞] for each event

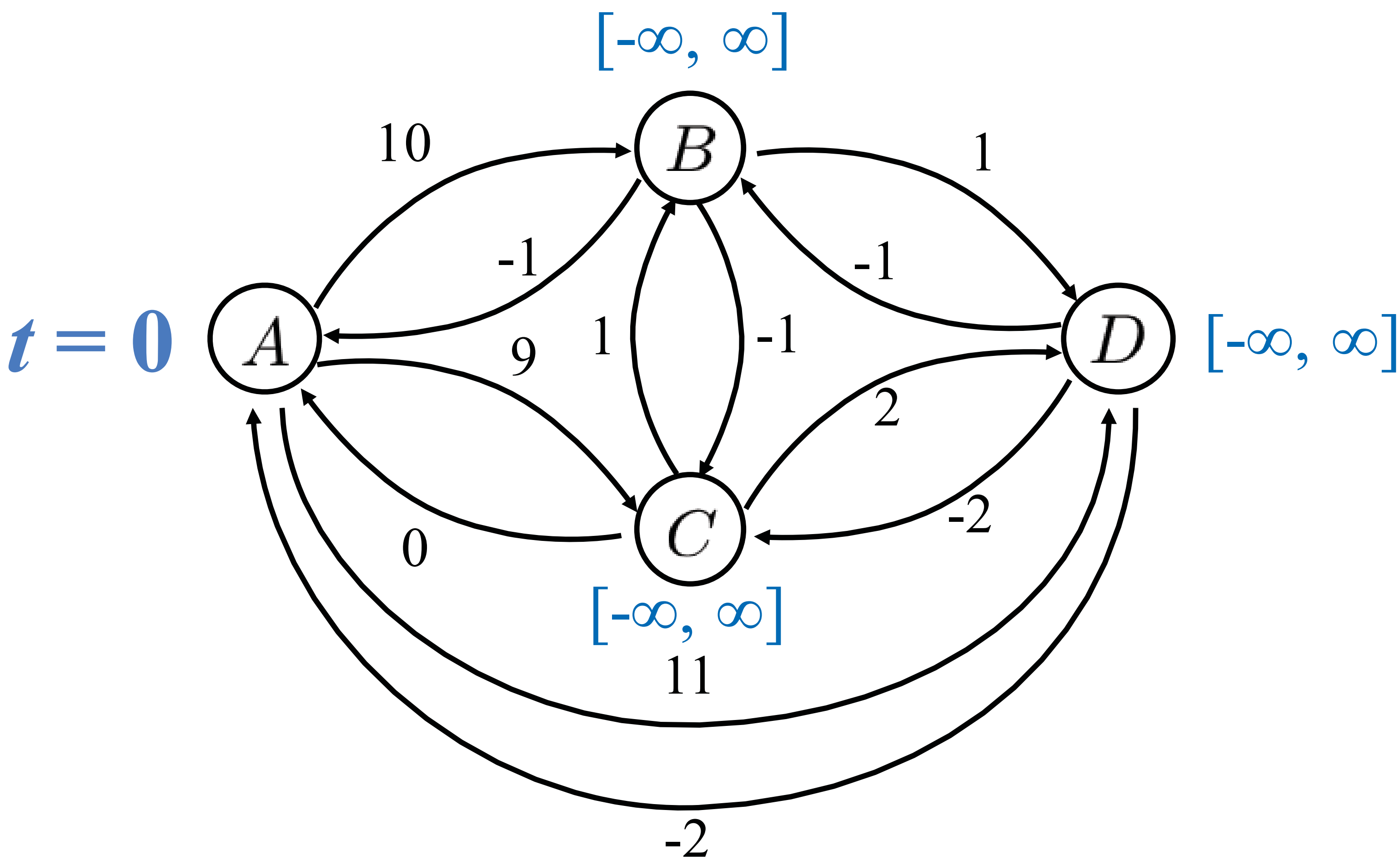
while unexecuted events:

    x_i = pick any unexecuted event **if current time in window**

    t_i = **now**

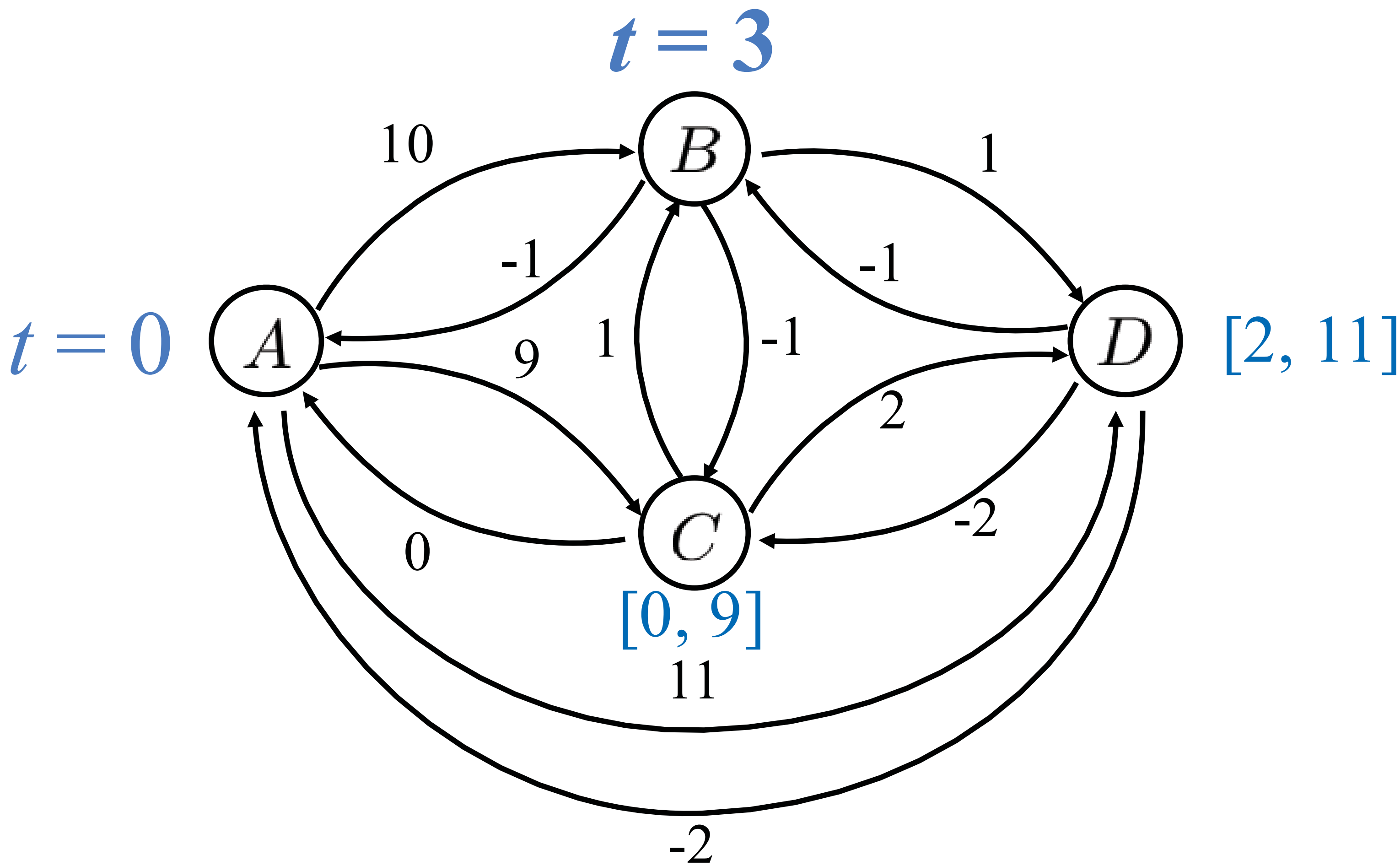
    Propagate to all x_i's neighbors & update their windows

Naïve (wrong!) online scheduling



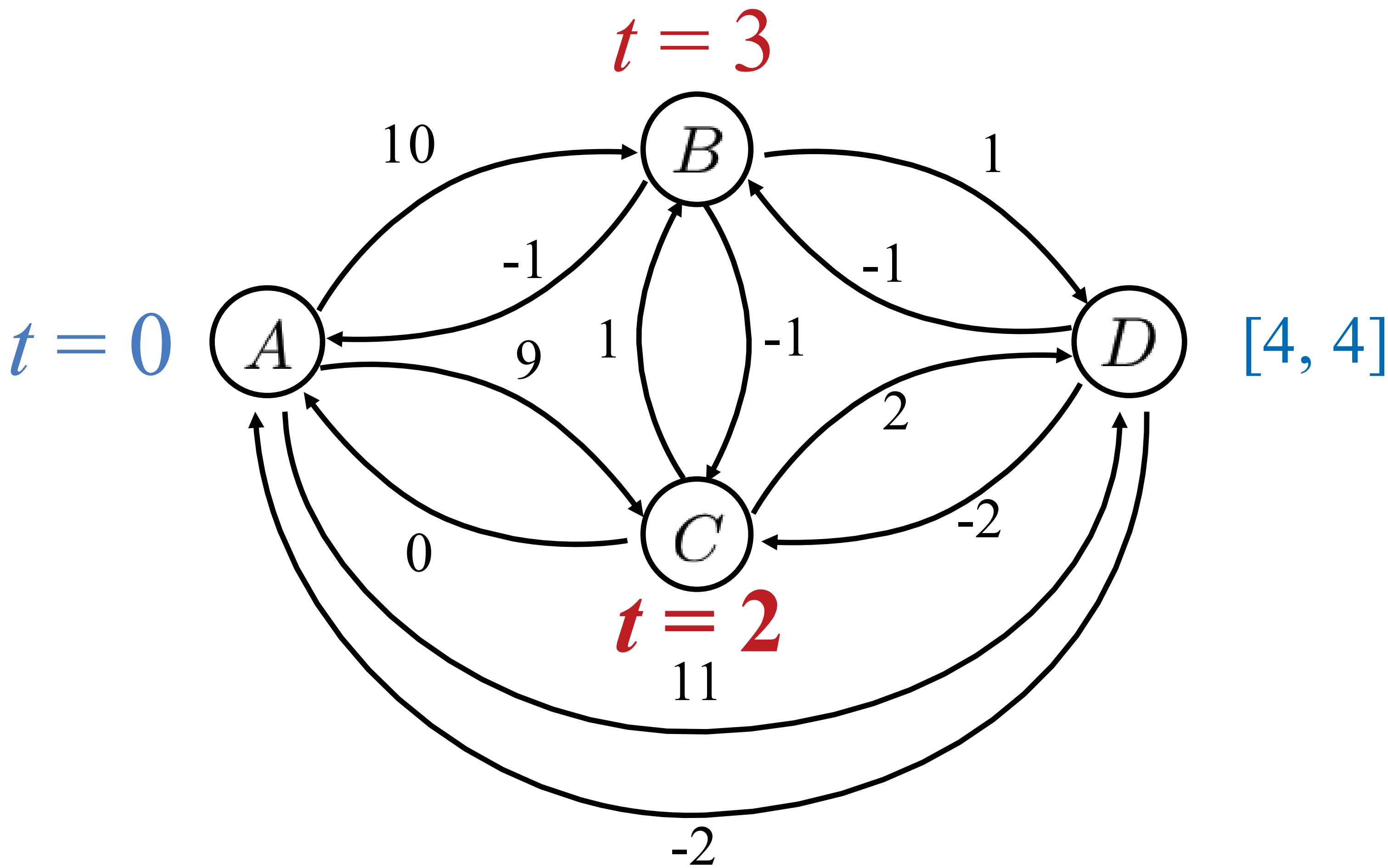
Arbitrarily picking next event…

Naïve (wrong!) online scheduling



Arbitrarily picking next event…

Naïve (wrong!) online scheduling



...but wait! We just ***assigned a past time!***

Enablement conditions dictate the ordering of dispatched events

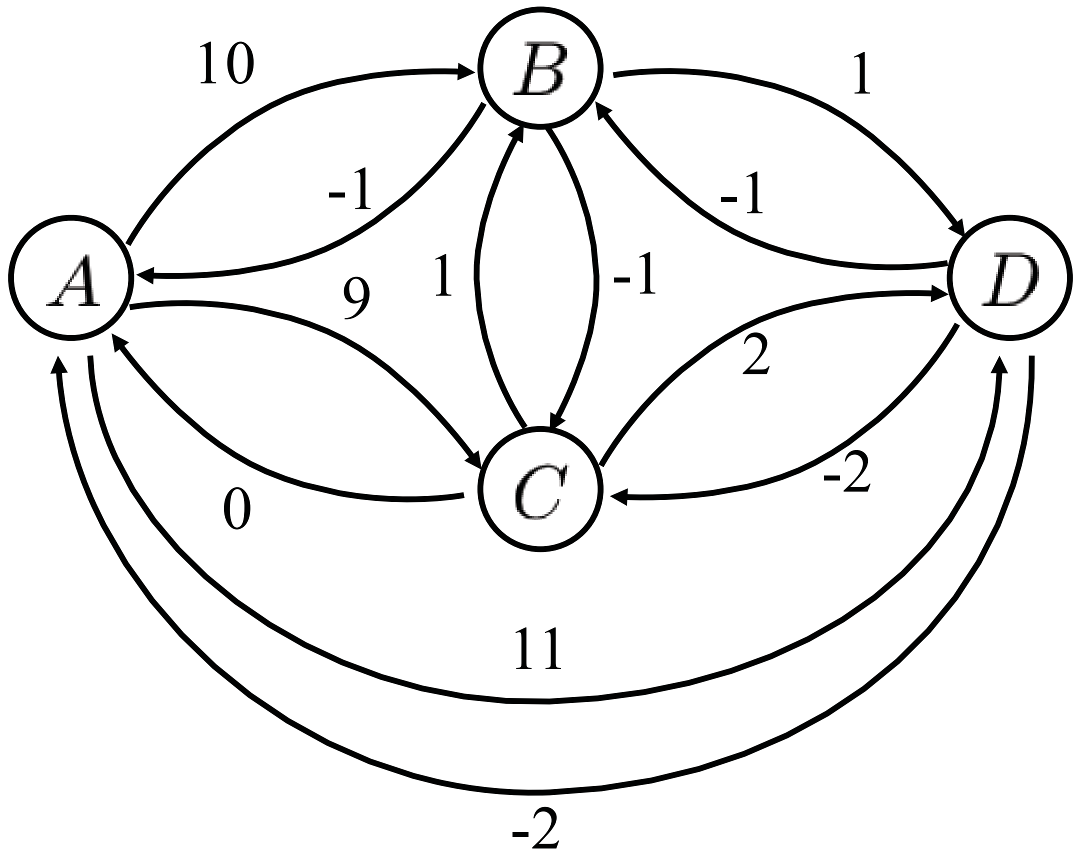
- Online: must assign monotonically increasing times
  - whereas offline algorithms may assign in any order.

- How can we constrain dispatcher to do this?

- **Solution:** determine "enablement conditions" by analyzing negative edges.
  - Allows us to infer if some edges must precede other edges

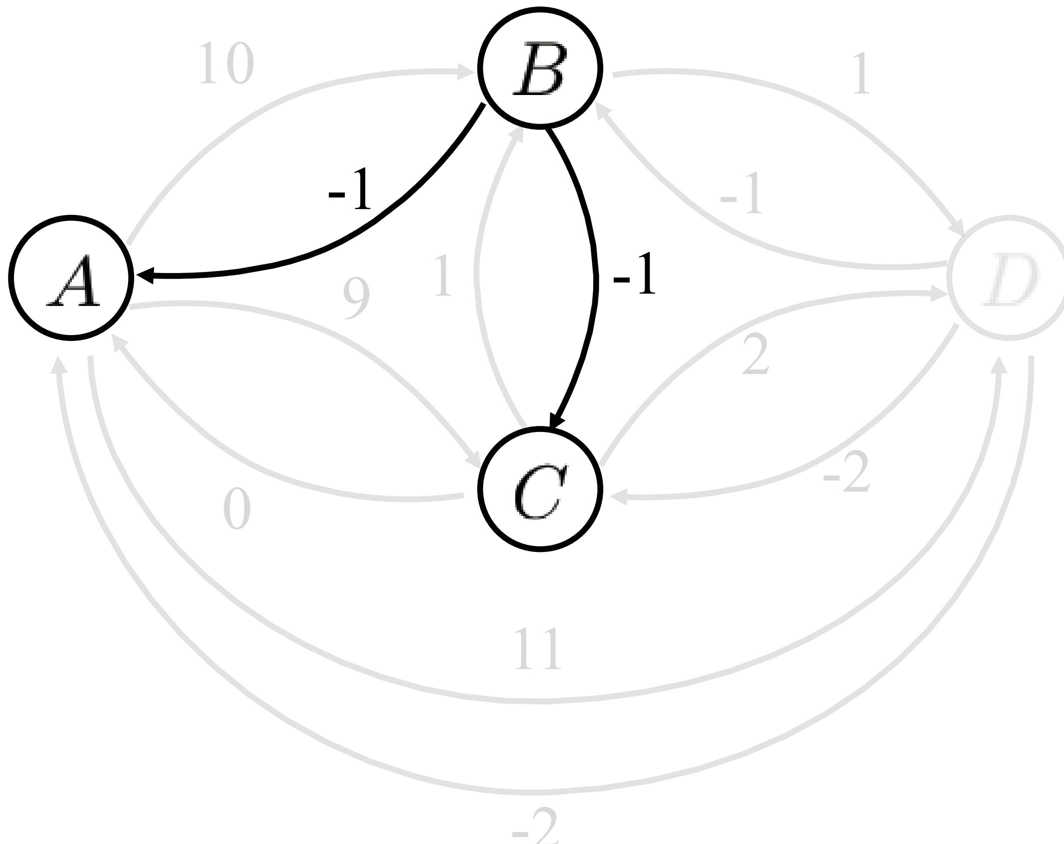
Enablement conditions dictate the ordering of dispatched events

- Negative edges from APSP dictate ordering constraints



Enablement conditions dictate the ordering of dispatched events

- Negative edges from APSP dictate ordering constraints



B must occur after both A and C!

Enablement conditions dictate the ordering of dispatched events

- An event is ***enabled*** if all its neighbors over negative edges have already been dispatched.
  – All "predecessors" have been dispatched.

- Modify online dispatching algorithm to only dispatch events if they are enabled.

- An event is ***active*** is the current time is within that event's execution window

Corrected online dispatcher

Compute dispatchable form (i.e., APSP)

Initialize execution window to [-∞, ∞] for each event

$E \leftarrow$ {events with no predecessors} # set of enabled events

$S \leftarrow$ {} # set of executed events

while unexecuted events:

    Wait until some event x_i in E is active

    t_i = now # dispatch x_i now at t_i

    Propagate to all x_i's neighbors & update their windows

    Add x_i to S

    Add to E any now-enabled events

Running online dispatcher

$E = \{A, C\}$

$S = \{\}$

Compute dispatchable form (i.e., APSP)

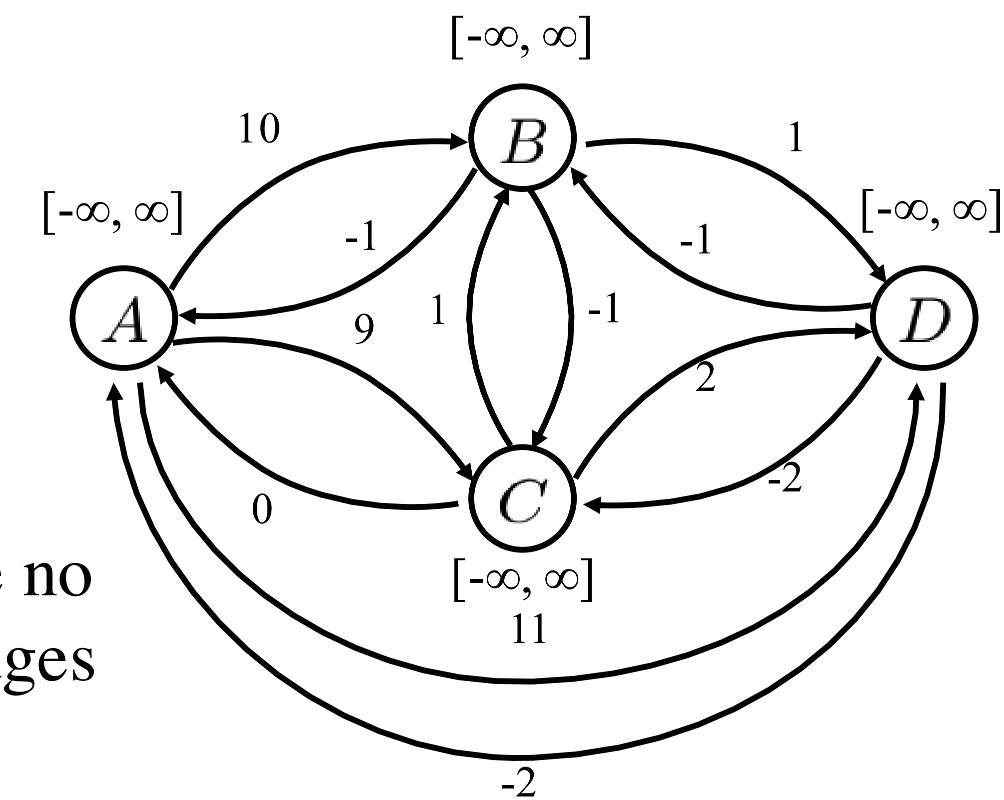
Initialize execution windows to [-∞, ∞]

$E \leftarrow$ {events with no predecessors}

$S \leftarrow \{\}$

while unexecuted events:

- Wait until some event x_i in E is active
- t_i = now
- Propagate to x_i's neighbors
- Add x_i to S
- Add to E any now-enabled events



A, C initially in E – have no negative, outgoing edges

Running online dispatcher

$E = \{A, C\}$

$S = \{\}$

Compute dispatchable form (i.e., APSP)

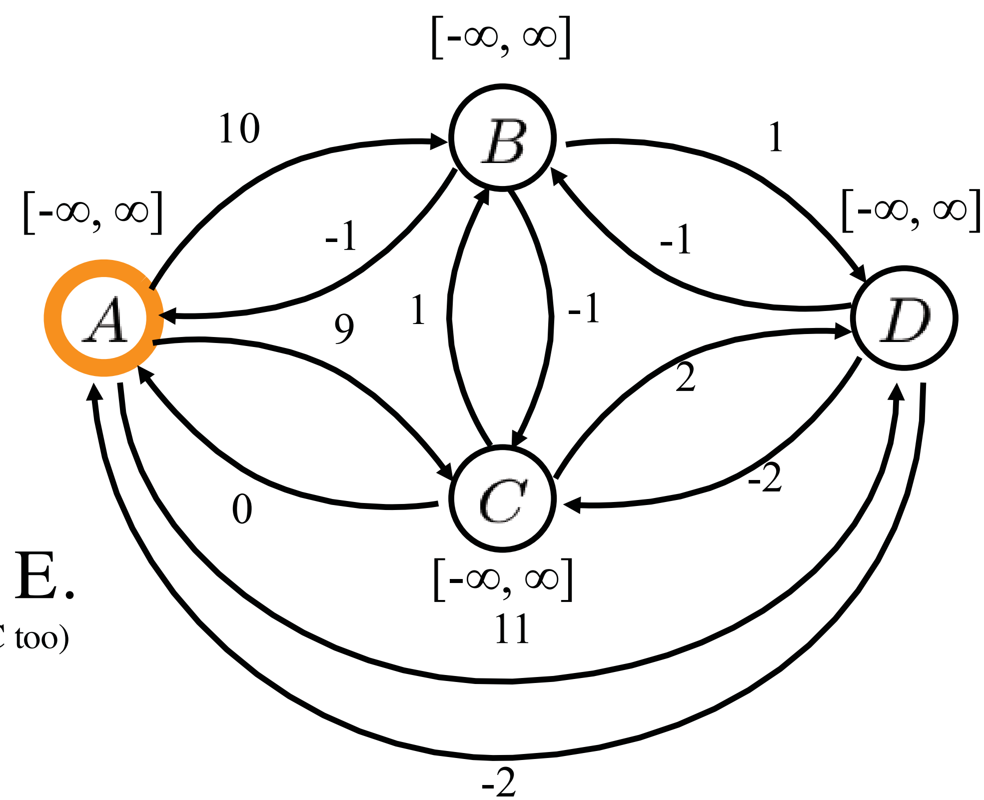
Initialize execution windows to [$-\infty$, ∞]

$E \leftarrow$ {events with no predecessors}

$S \leftarrow \{\}$

while unexecuted events:

- Wait until some event x_i in E is active
- t_i = now
- Propagate to x_i's neighbors
- Add x_i to S
- Add to E any now-enabled events



A is enabled and in E.

(could have chosen C too)

Running online dispatcher

Compute dispatchable form (i.e., APSP)

Initialize execution windows to [-∞, ∞]

$E \leftarrow$ {events with no predecessors}

$S \leftarrow$ {}

while unexecuted events:

Wait until some event x_i in E is active

t_i = now

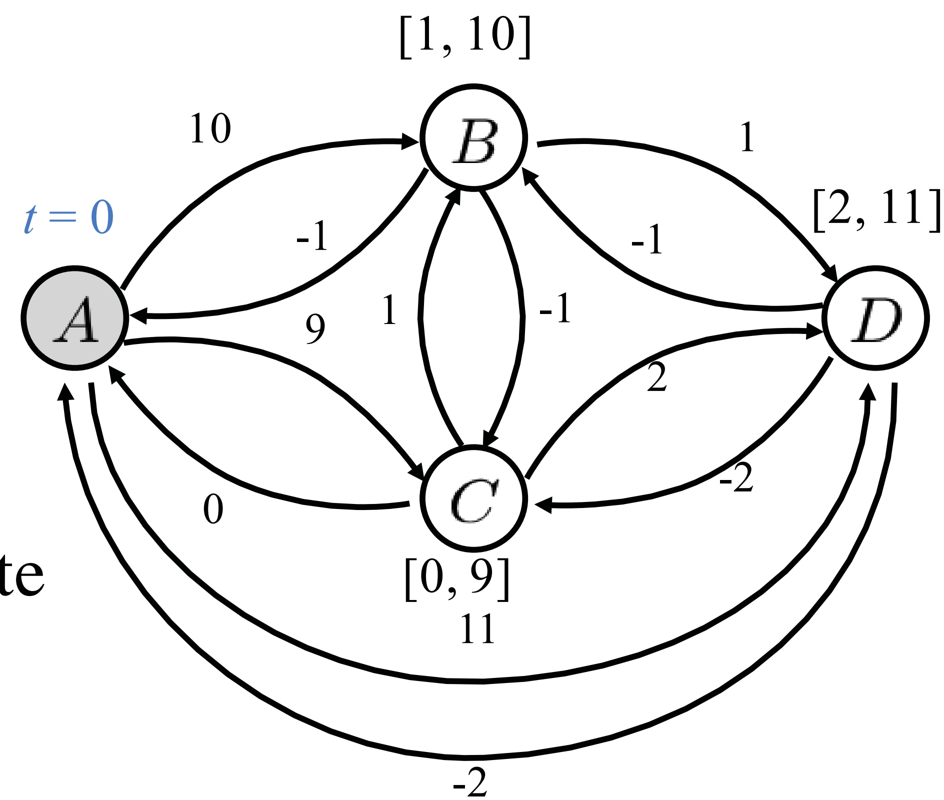
Propagate to x_i's neighbors

Add x_i to S

Add to E any now-enabled events

$E = \{C\}$

$S = \{A\}$



Dispatch A and propagate

Running online dispatcher

$E = \{C\}$
$S = \{A\}$

Compute dispatchable form (i.e., APSP)

Initialize execution windows to $[-\infty, \infty]$

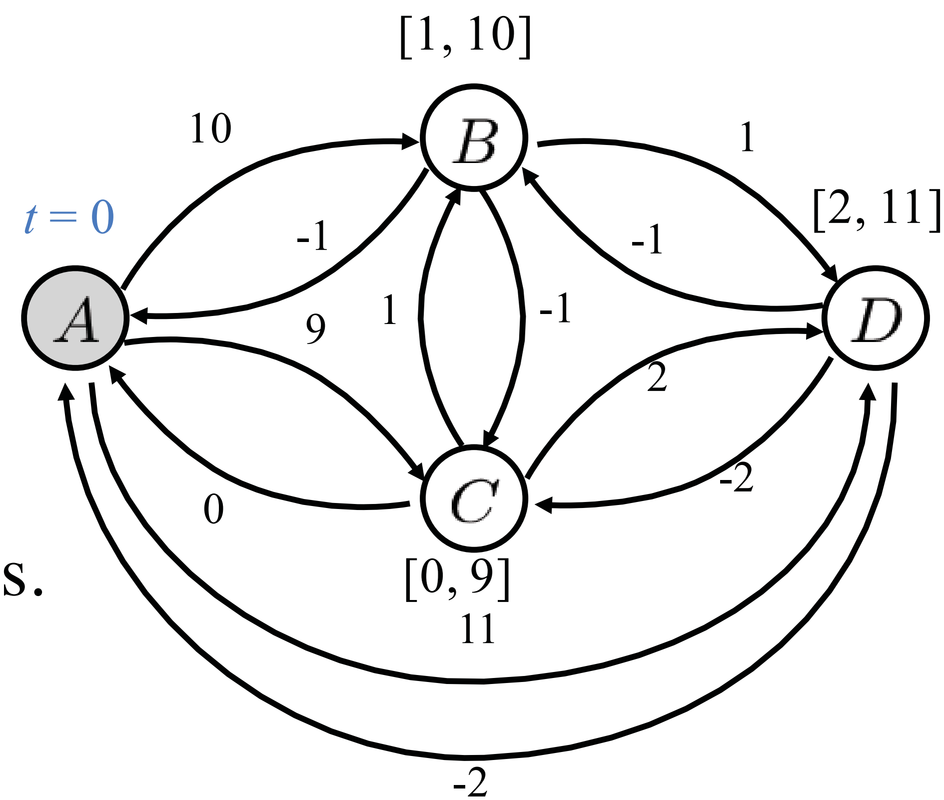
$E \leftarrow$ {events with no predecessors}

$S \leftarrow \{\}$

while unexecuted events:

- Wait until some event x_i in E is active
- t_i = now
- Propagate to x_i's neighbors
- Add x_i to S
- Add to E any now-enabled events

B, D not enabled! But C still is.



Running online dispatcher

Compute dispatchable form (i.e., APSP)

Initialize execution windows to [-∞, ∞]

$E \leftarrow$ {events with no predecessors}

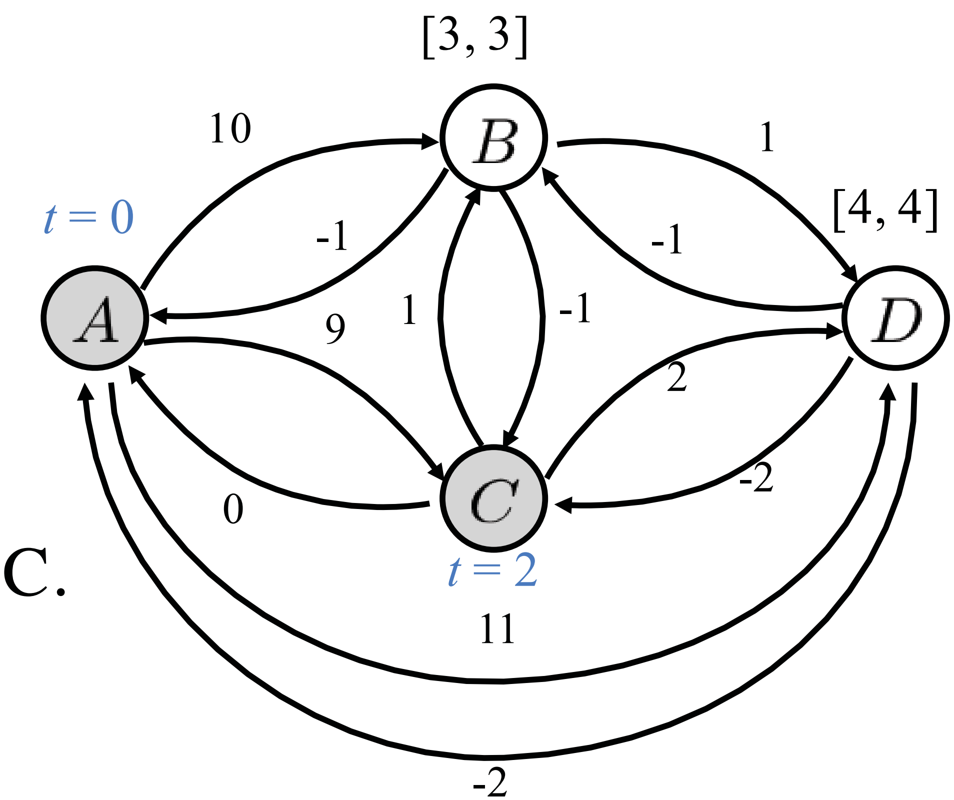
$S \leftarrow \{\}$

while unexecuted events:

- Wait until some event x_i in E is active
- t_i = now
- Propagate to x_i's neighbors
- Add x_i to S
- Add to E any now-enabled events

$E = \{\}$

$S = \{A, C\}$



Dispatch & propagate C.

Running online dispatcher

Compute dispatchable form (i.e., APSP)

Initialize execution windows to [-∞, ∞]

$E \leftarrow$ {events with no predecessors}

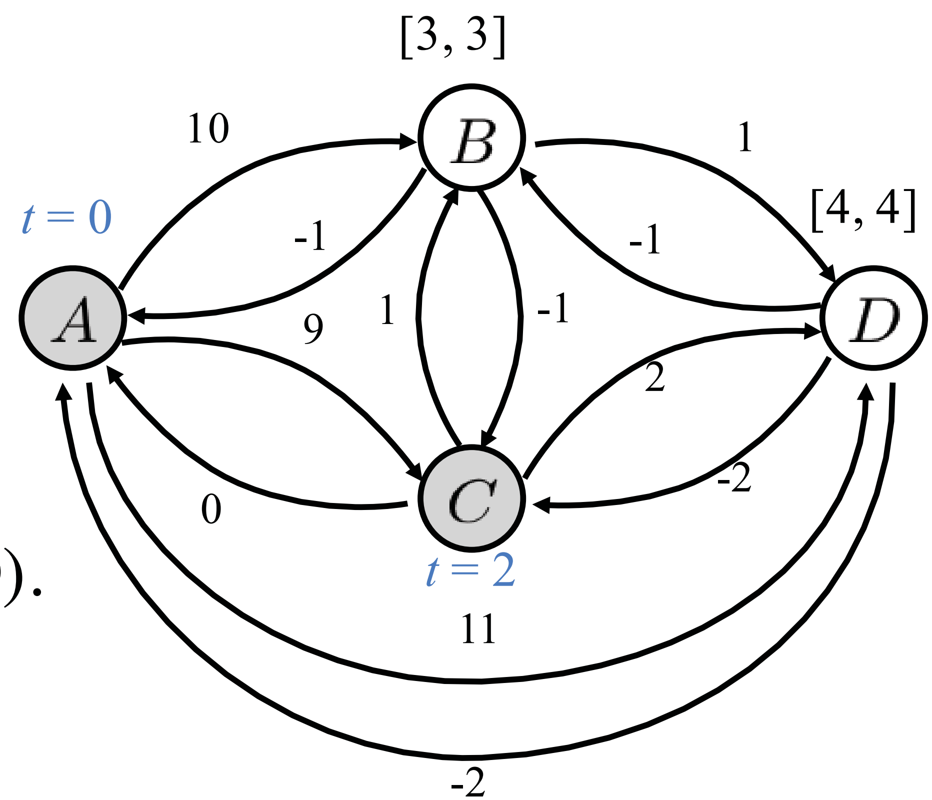
$S \leftarrow$ {}

while unexecuted events:

> Wait until some event x_i in E is active
>
> t_i = now
>
> Propagate to x_i's neighbors
>
> Add x_i to S
>
> Add to E any now-enabled events

E = {B}

S = {A, C}



B is now enabled (but still not D).

Running online dispatcher

Compute dispatchable form (i.e., APSP)

Initialize execution windows to [-∞, ∞]

$E \leftarrow$ {events with no predecessors}

$S \leftarrow$ {}

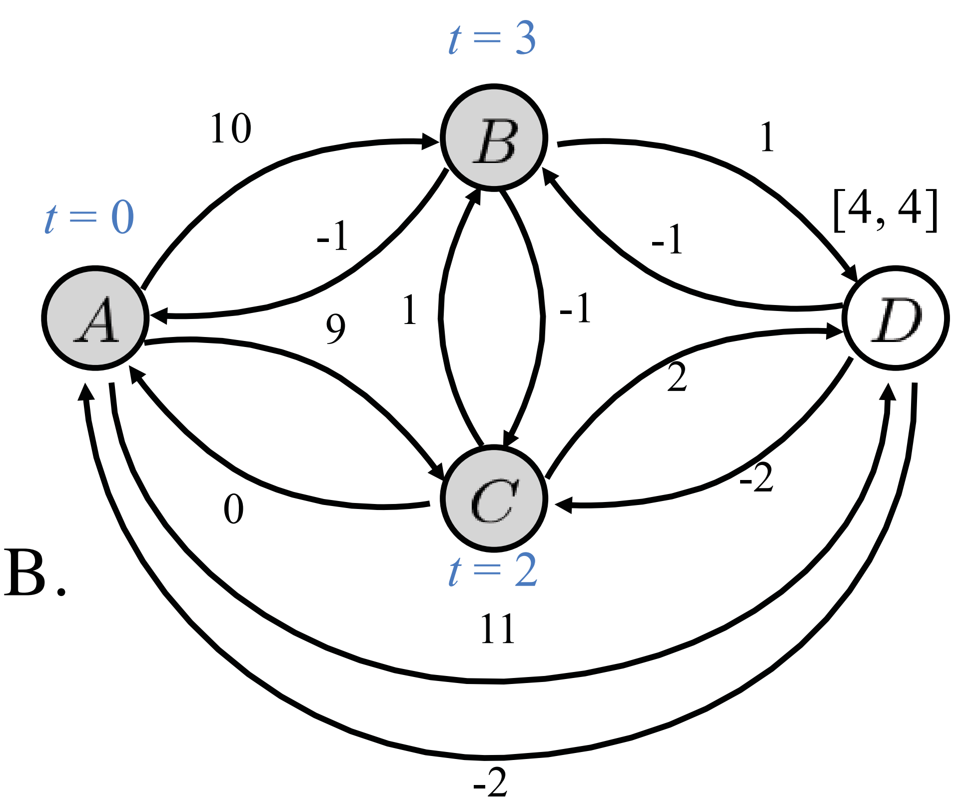
while unexecuted events:

- Wait until some event x_i in E is active
- t_i = now
- Propagate to x_i's neighbors
- Add x_i to S
- Add to E any now-enabled events

$E = \{\}$

$S = \{A, C, B\}$

Dispatch & propagate B.



Running online dispatcher

$E = \{\mathrm{D}\}$

$S = \{\mathrm{A, C, B}\}$

Compute dispatchable form (i.e., APSP)

Initialize execution windows to [-∞, ∞]

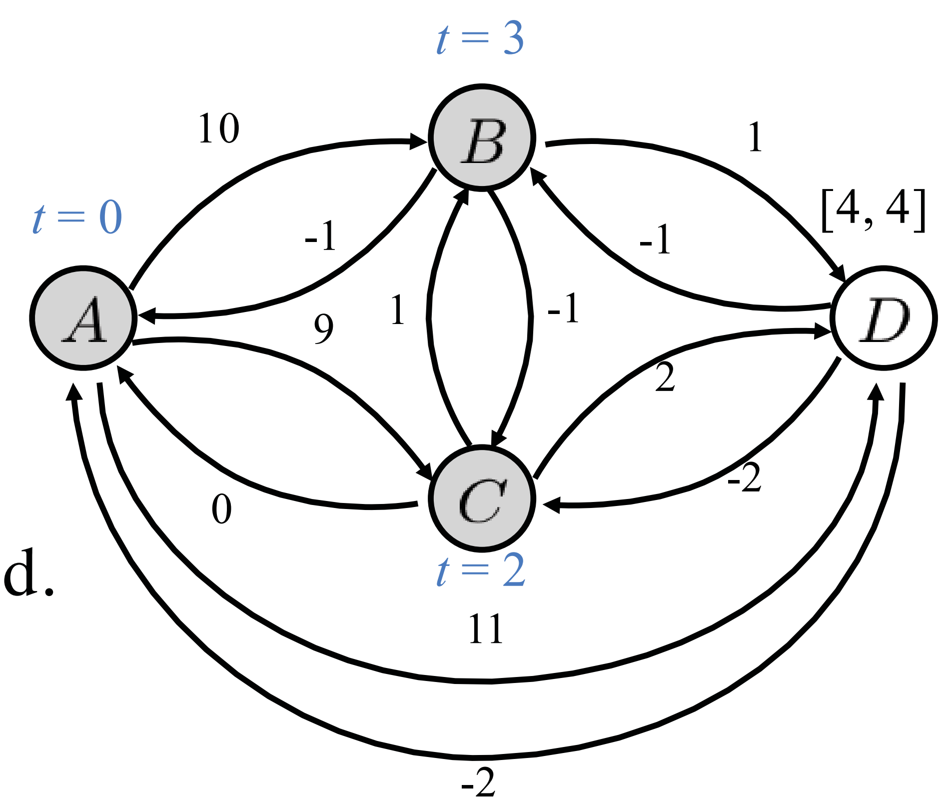
$E \leftarrow$ {events with no predecessors}

$S \leftarrow \{\}$

while unexecuted events:

- Wait until some event x_i in E is active
- t_i = now
- Propagate to x_i's neighbors
- Add x_i to S
- Add to E any now-enabled events

D is finally enabled.



Running online dispatcher

Compute dispatchable form (i.e., APSP)

Initialize execution windows to [-∞, ∞]

E ← {events with no predecessors}

S ← {}

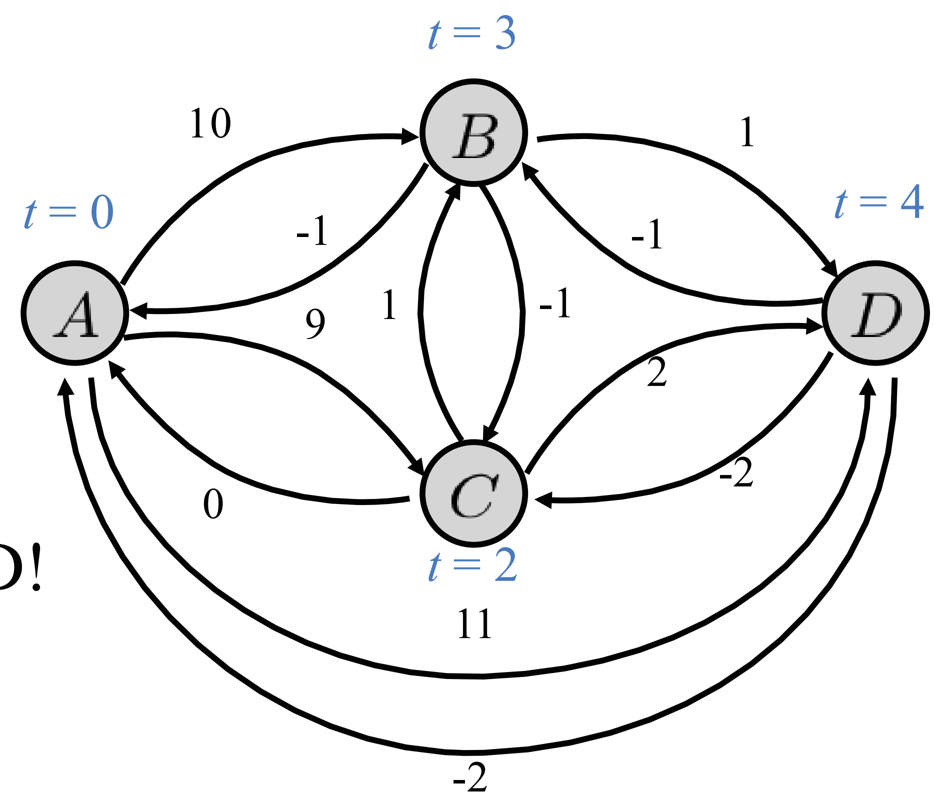
while unexecuted events:

- Wait until some event x_i in E is active
- t_i = now
- Propagate to x_i's neighbors
- Add x_i to S
- Add to E any now-enabled events

$E = \{\}$

$S = \{A, C, B, D\}$

Finish up by dispatching D!



73

Online dispatching algorithm remarks

- By considering predecessors, we guarantee that events assigned monotonically increasing times online.
- Capable of responding to fluctuations that do not affect overall temporal feasibility.
- (**Note:** must be run on an dispatchable / APSP graph!)

Online dispatcher efficiency

- Consider an STN with n edges.
- How many edges in APSP distance graph?

Compute dispatchable form (i.e., APSP)
Initialize execution windows to $[-\infty, \infty]$
$E \leftarrow$ {events with no predecessors}
$S \leftarrow \{\}$
while unexecuted events:
    Wait until some event x_i in E is active
    t_i = now
    Propagate to x_i's neighbors
    Add x_i to S
    Add to E any now-enabled events

Online dispatcher efficiency

- Consider an STN with n edges.
- How many edges in APSP distance graph? n^2.

Compute dispatchable form (i.e., APSP)
Initialize execution windows to [-∞, ∞]
$E \leftarrow$ {events with no predecessors}
$S \leftarrow$ {}
while unexecuted events:
    Wait until some event x_i in E is active
    t_i = now
    Propagate to x_i's neighbors
    Add x_i to S
    Add to E any now-enabled events

Online dispatcher efficiency

- Consider an STN with n edges.
- How many edges in APSP distance graph? n^2.
- How many neighbors to propagate to each step?

Compute dispatchable form (i.e., APSP)
Initialize execution windows to [-∞, ∞]
$E \leftarrow$ {events with no predecessors}
$S \leftarrow$ {}
while unexecuted events:
    Wait until some event x_i in E is active
    t_i = now
    Propagate to x_i's neighbors
    Add x_i to S
    Add to E any now-enabled events

Online dispatcher efficiency

- Consider an STN with n edges.
- How many edges in APSP distance graph? n^2.
- How many neighbors to propagate to each step? n.

Compute dispatchable form (i.e., APSP)
Initialize execution windows to $[-\infty, \infty]$
$E \leftarrow$ {events with no predecessors}
$S \leftarrow \{\}$
while unexecuted events:
    Wait until some event x_i in E is active
    t_i = now
    Propagate to x_i's neighbors
    Add x_i to S
    Add to E any now-enabled events

Online dispatcher efficiency

- Consider an STN with n edges.
- How many edges in APSP distance graph? n^2.
- How many neighbors to propagate to each step? n.
- Large STNs: propagation slow. Want to reduce this.

Compute dispatchable form (i.e., APSP)
Initialize execution windows to [-∞, ∞]
$E \leftarrow$ {events with no predecessors}
$S \leftarrow$ {}
while unexecuted events:
    Wait until some event x_i in E is active
    t_i = now
    Propagate to x_i's neighbors
    Add x_i to S
    Add to E any now-enabled events

To Execute a Temporal Plan

Schedule Offline

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan

offline

online

4. Execute Plan

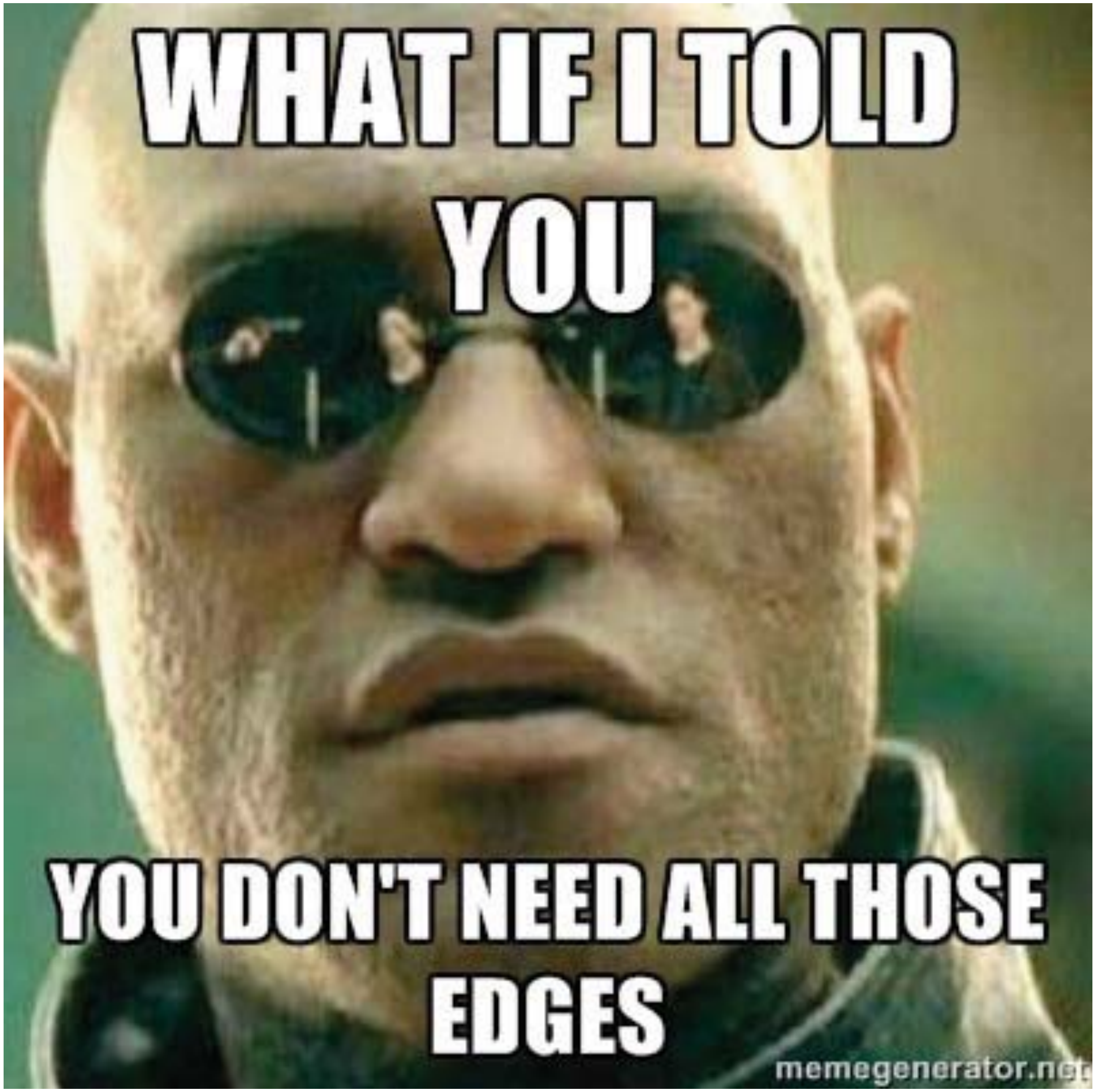
Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan

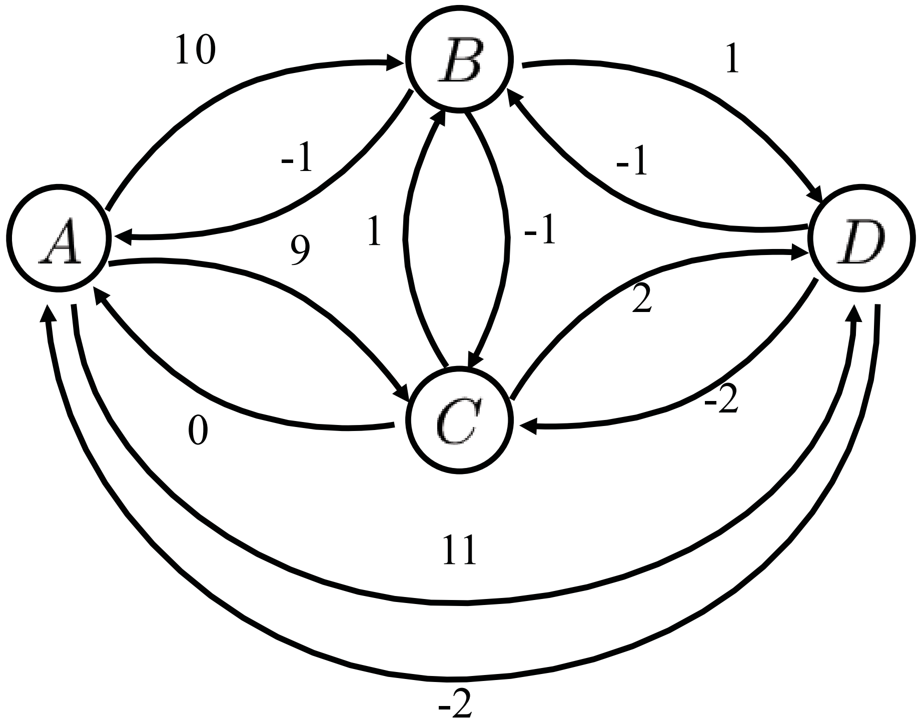
offline

online

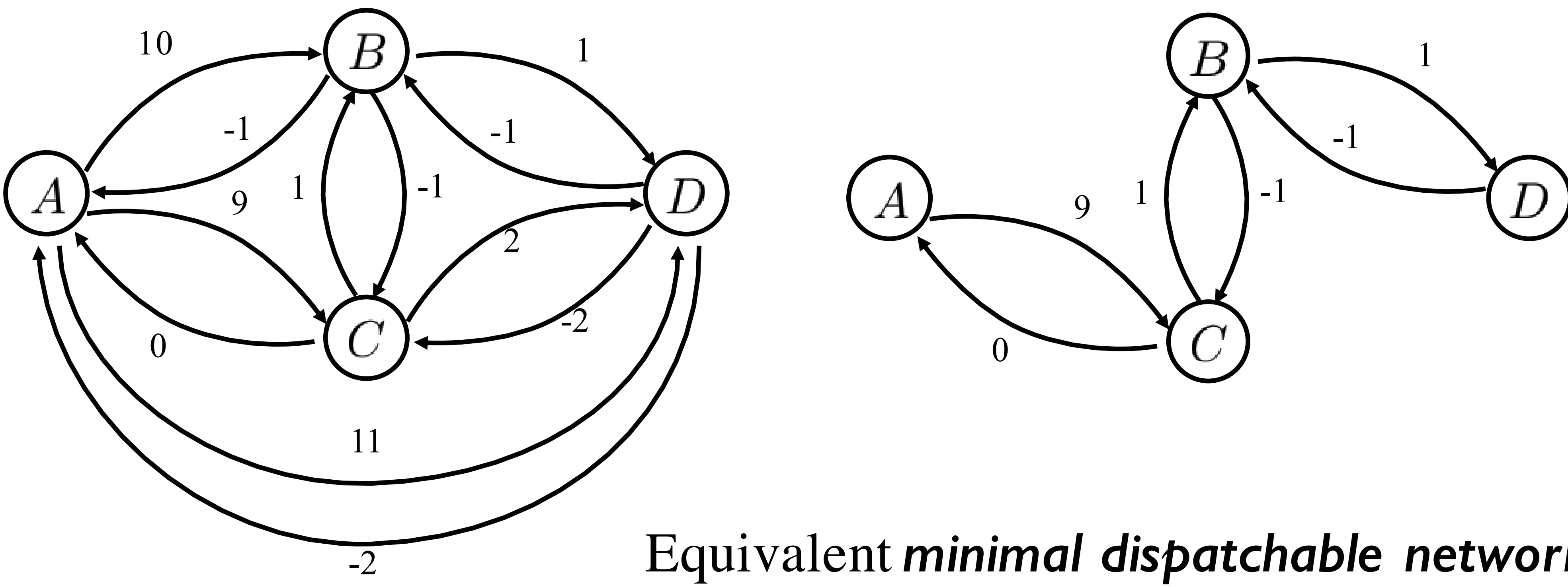
4. Dynamically Schedule Plan


WHAT IF I TOLD YOU
YOU DON'T NEED ALL THOSE EDGES
memegenerator.net

You don't need all those edges!



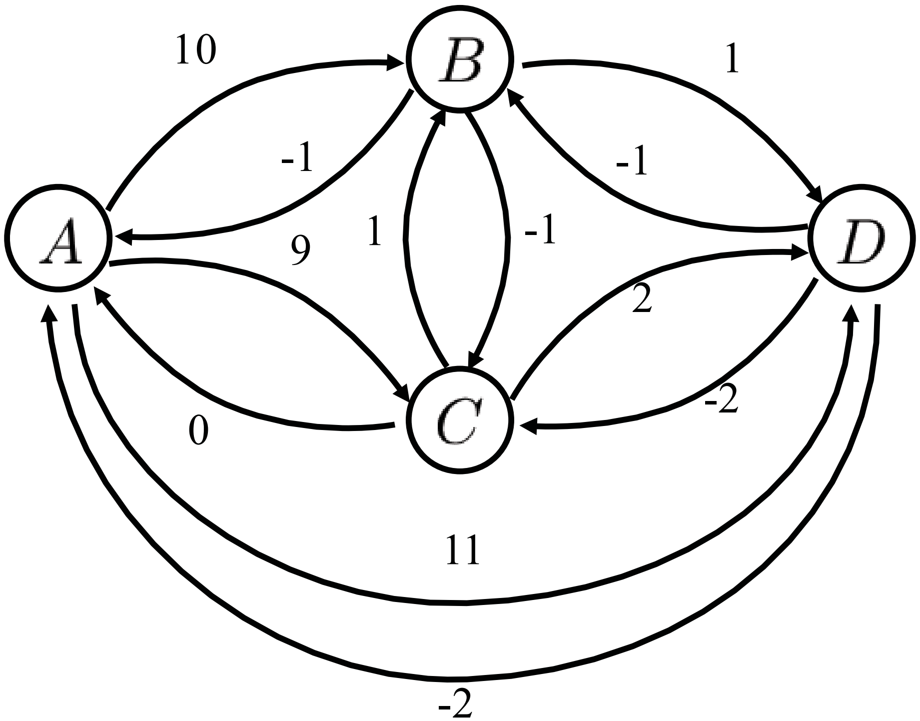
You don't need all those edges!



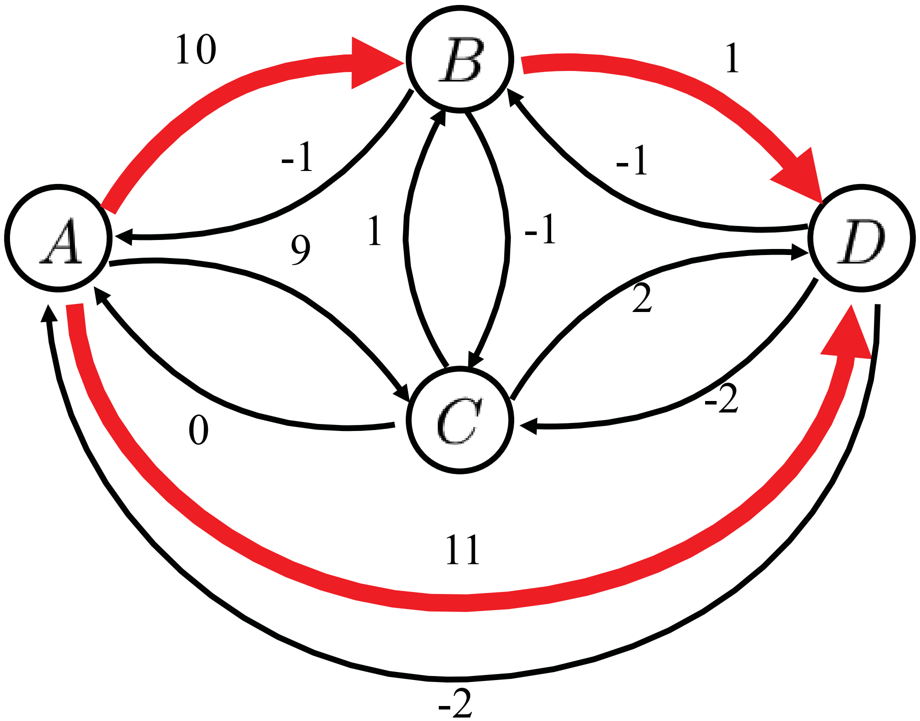
Equivalent ***minimal dispatchable network***

You don’t need all those edges!

Let’s consider a specific triangle of edges.



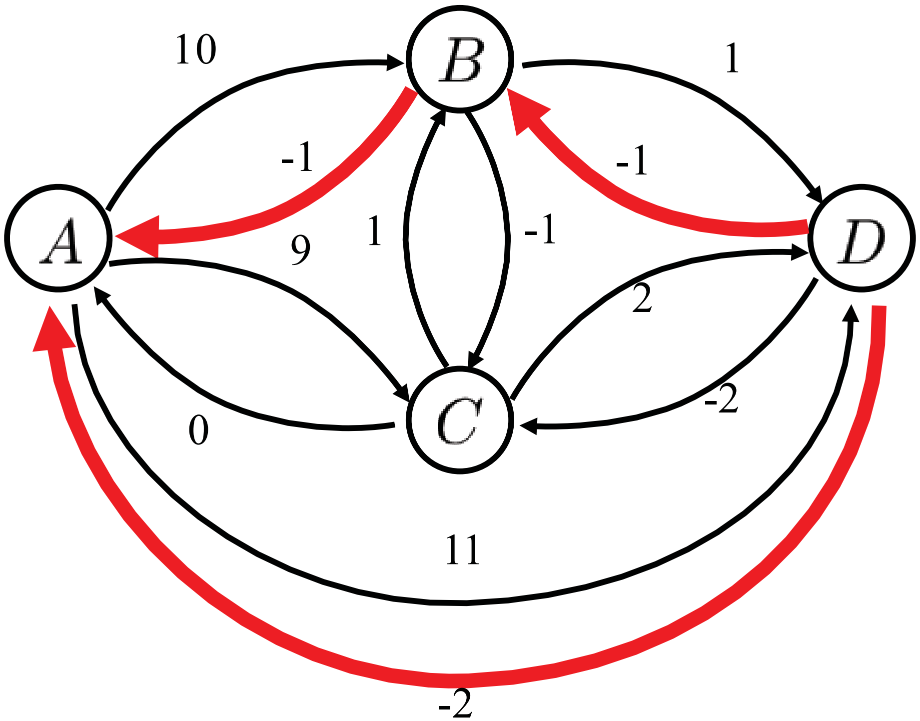
You don’t need all those edges!



Let’s consider a specific triangle of edges.

Do we really need the bottom edge?

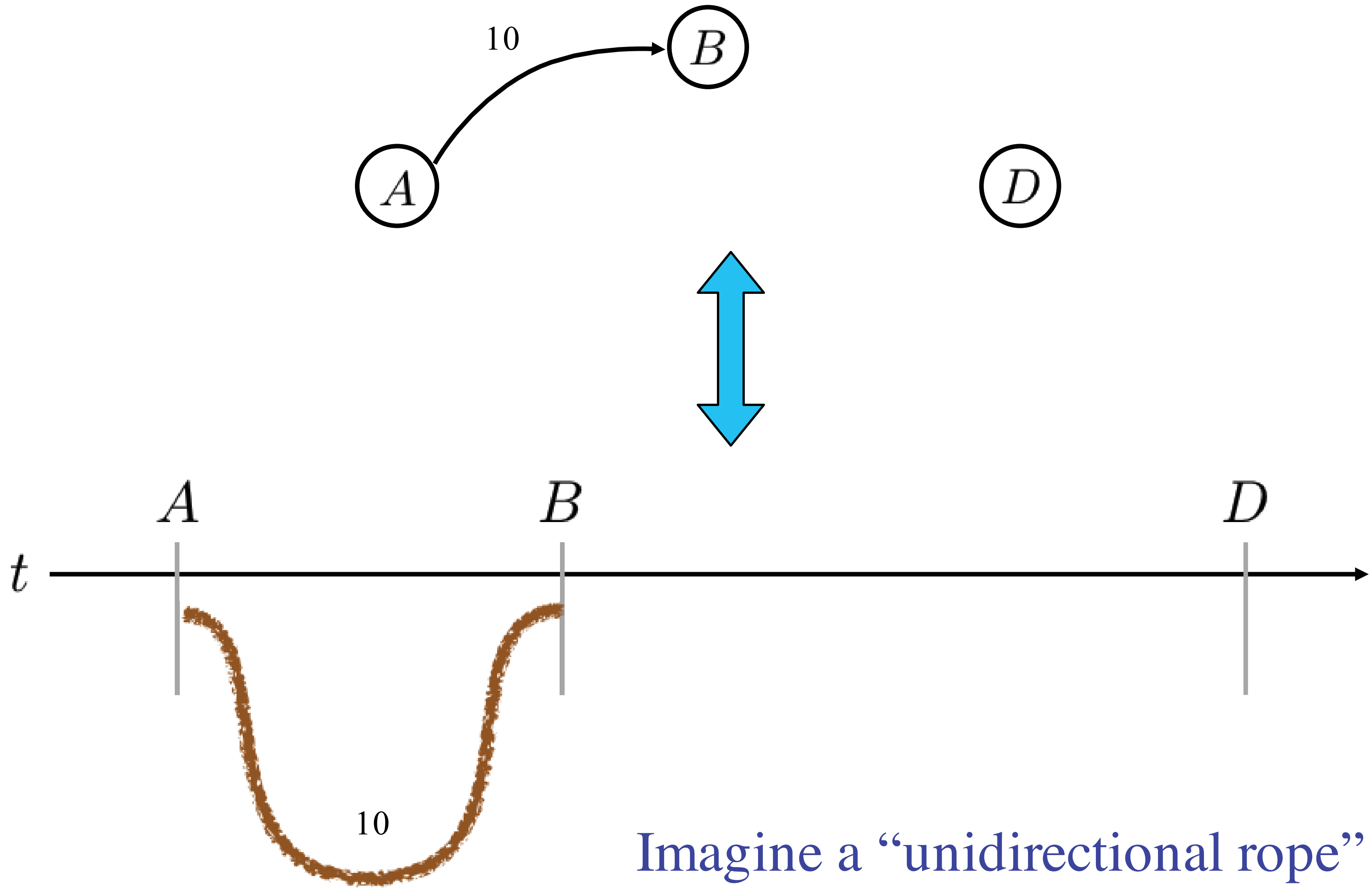
You don't need all those edges!



Let's consider a different triangle of edges.

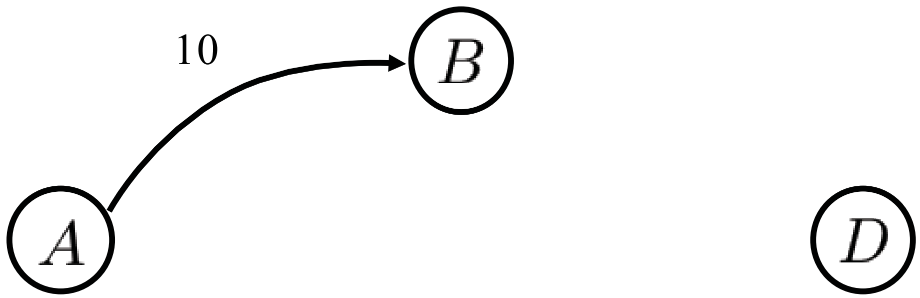
Do we really need the bottom edge?

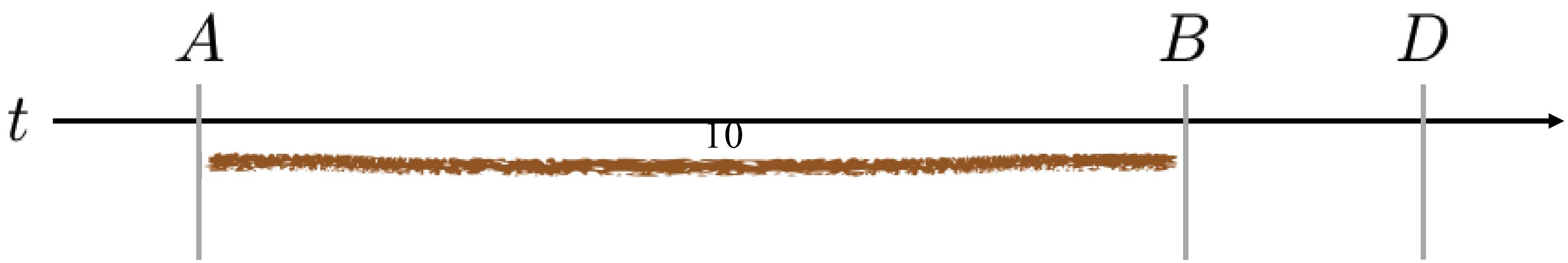
A ropes analogy



Imagine a "unidirectional rope" of length 10 constraining sliders on a track.

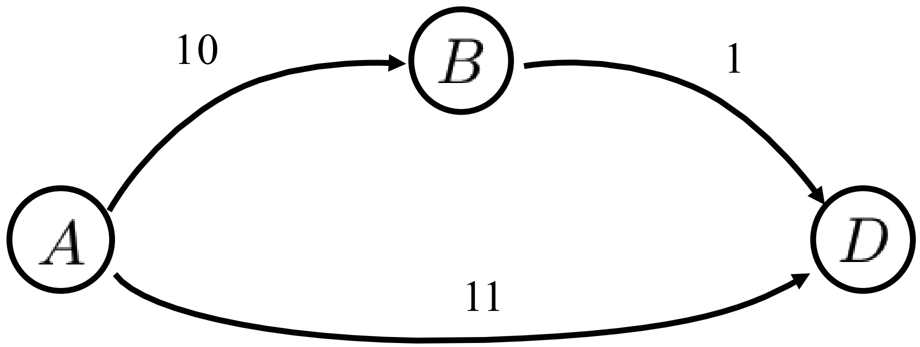
Rope analogy

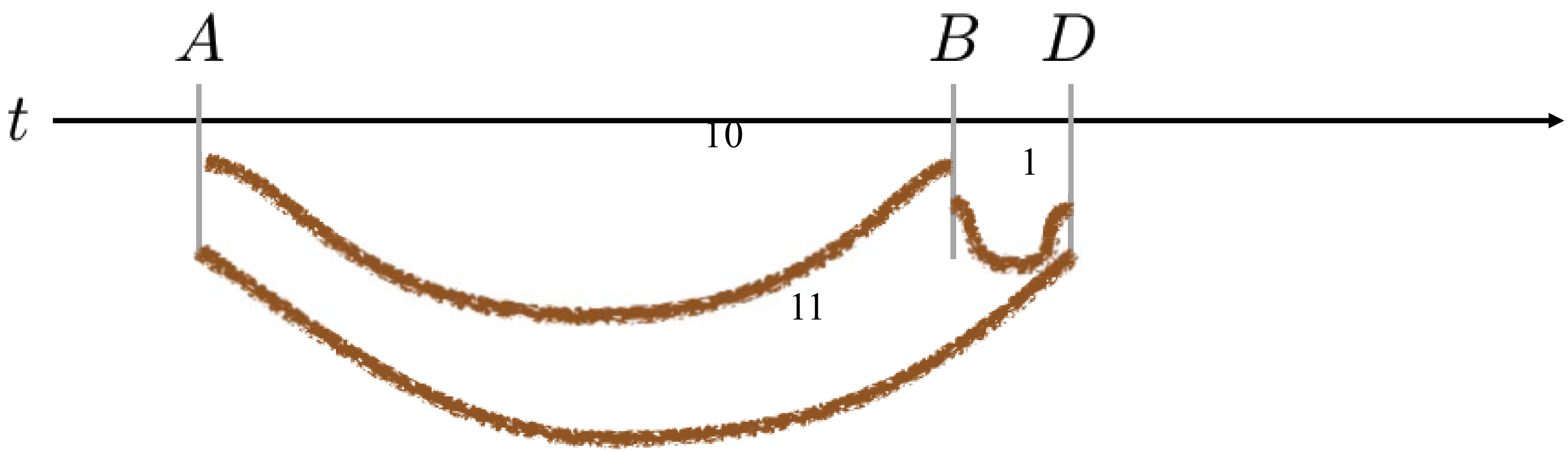




Imagine a "unidirectional rope" of length 10 constraining sliders on a track.

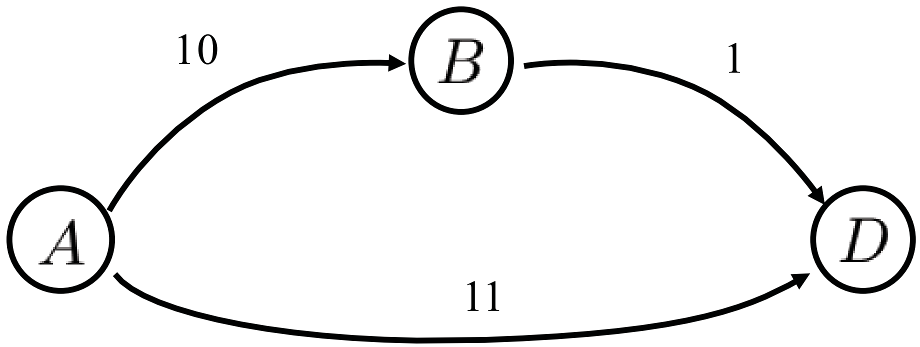
Rope analogy



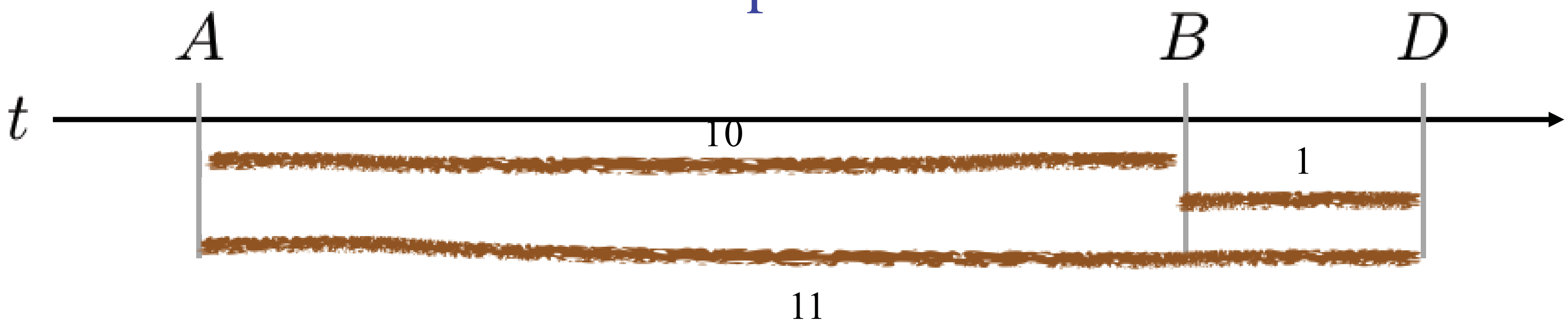


Now add in ropes for other constraints

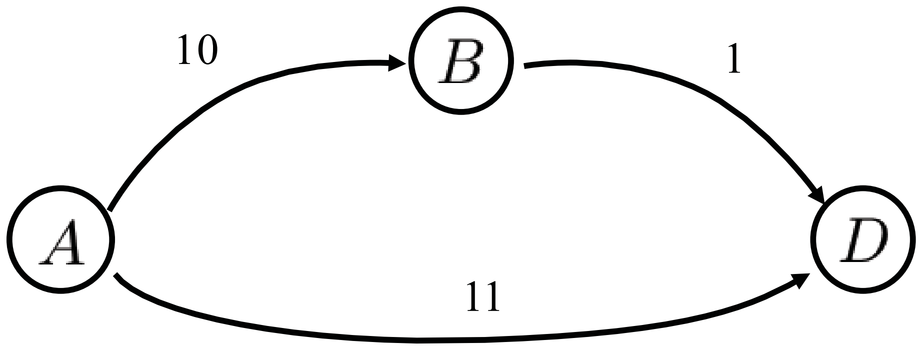
Rope analogy



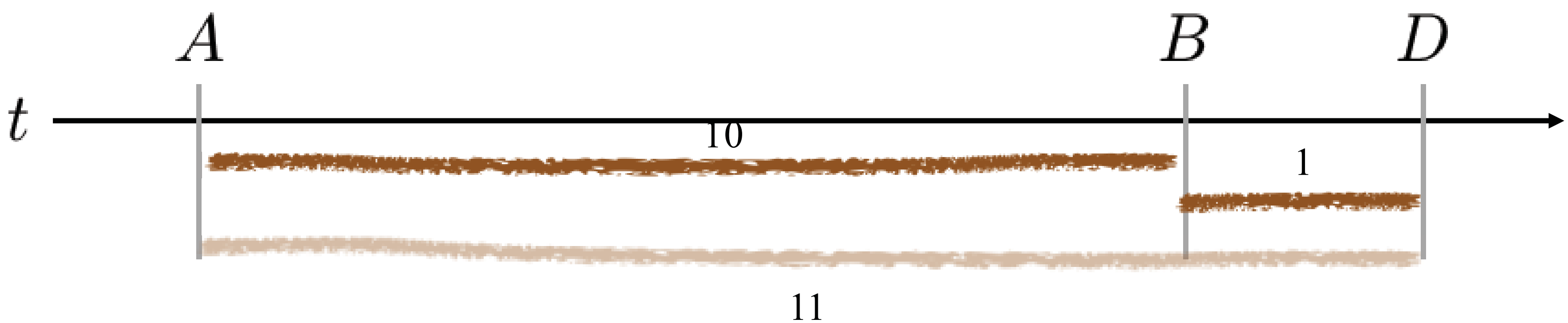
Imagine pulling A and D as tightly as possible.



Rope analogy

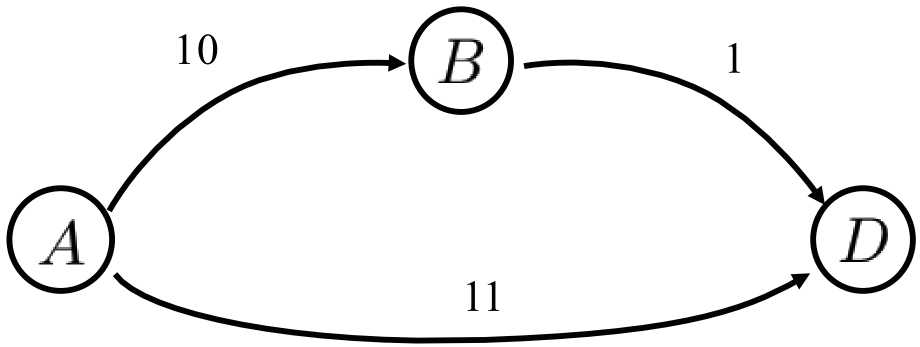


Can we remove rope AD without changing behavior?

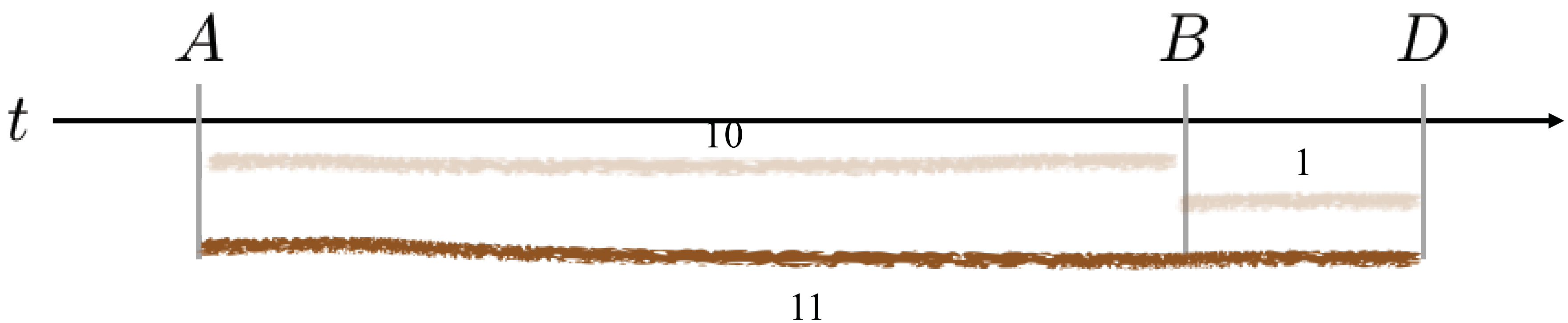


Yes! Same possible positions for A, B, D.

Rope analogy

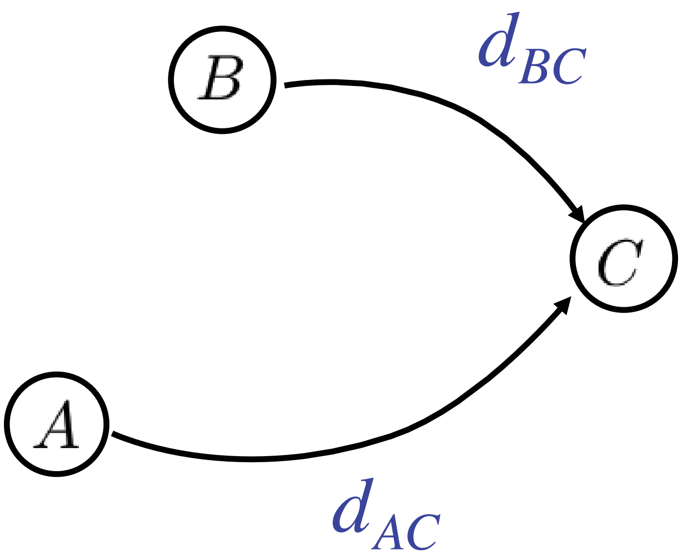


Can we remove ropes AB, BD without changing behavior?



No. AD still constrained, but B could slide freely! Not the same behavior. Collectively, AB and BD entail AD (but AD does not entail both AB and AD).

Upper dominating edges - detection from APSP



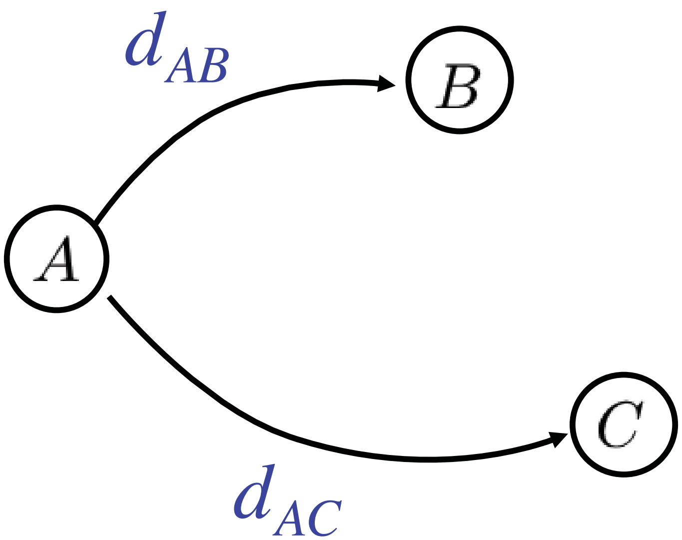
If d_{AC}, $d_{BC} \geq 0$ and

$d_{AB} + d_{BC} = d_{AC}$

then BC ***dominates*** AC

(Proof omitted - based on triangle rule property of APSP. Please see notes / reading for more info)

Lower dominating edges - detection from APSP



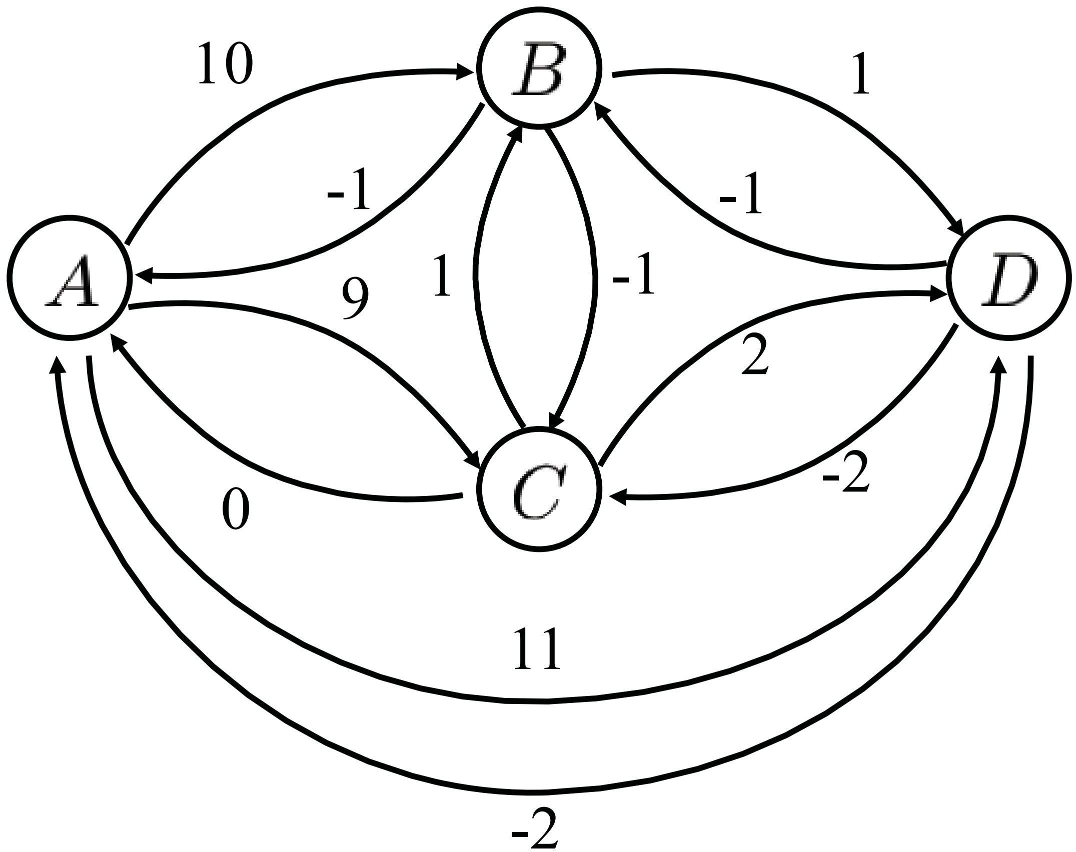
If $d_{AB}, d_{AC} < 0$ and

$d_{AB} + d_{BC} = d_{AC}$

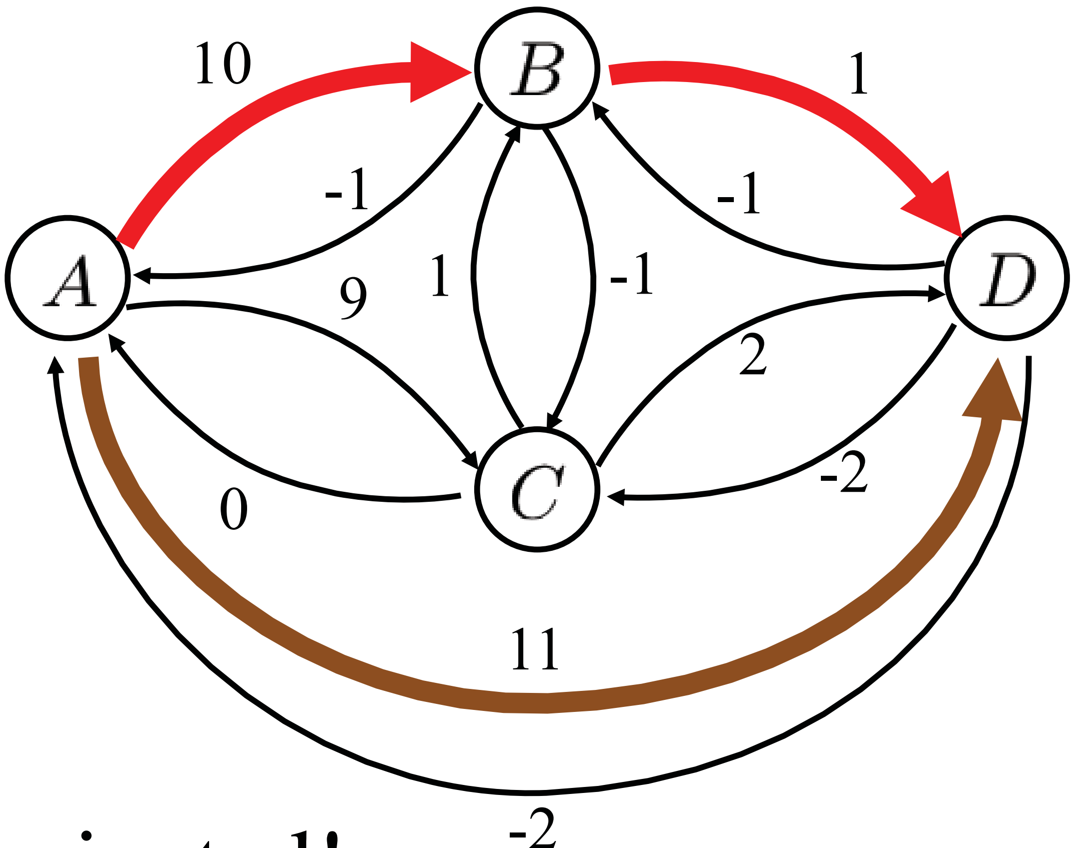
then AB ***dominates*** AC

(Proof omitted - based on triangle rule property of APSP. Please see notes / reading for more info)

Dominance example

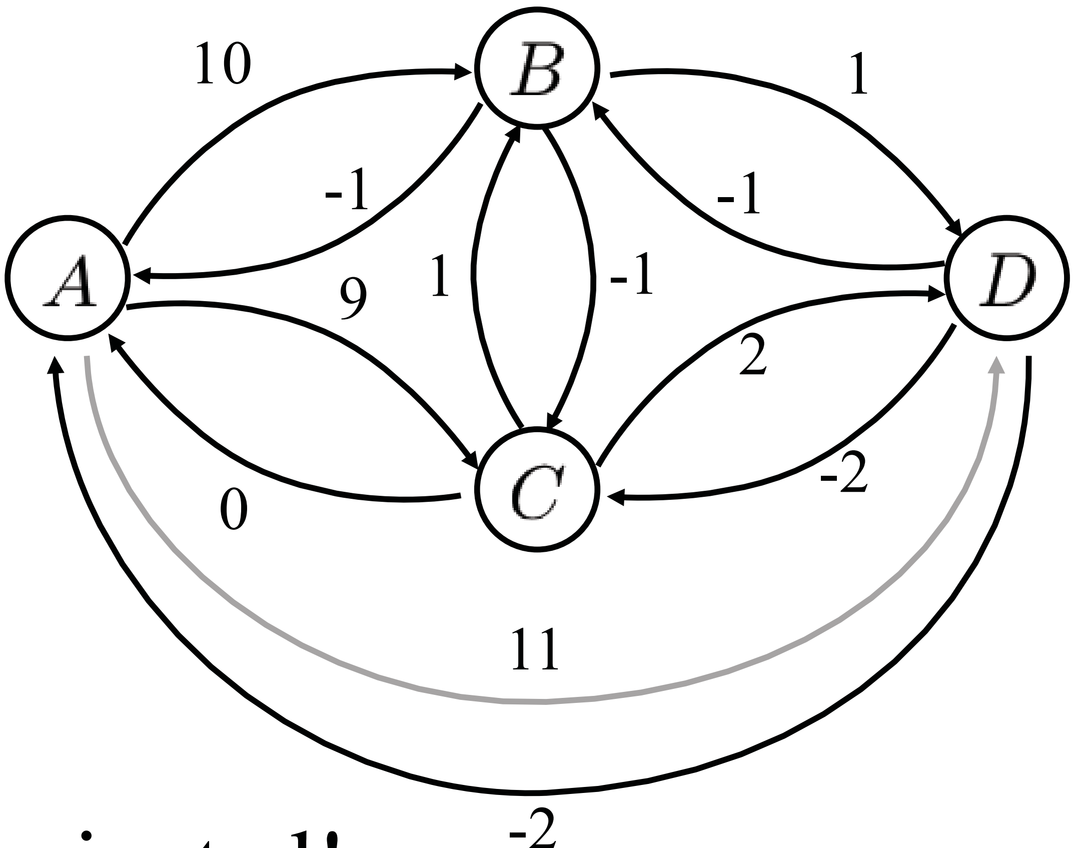


Dominance example



Upper dominated!

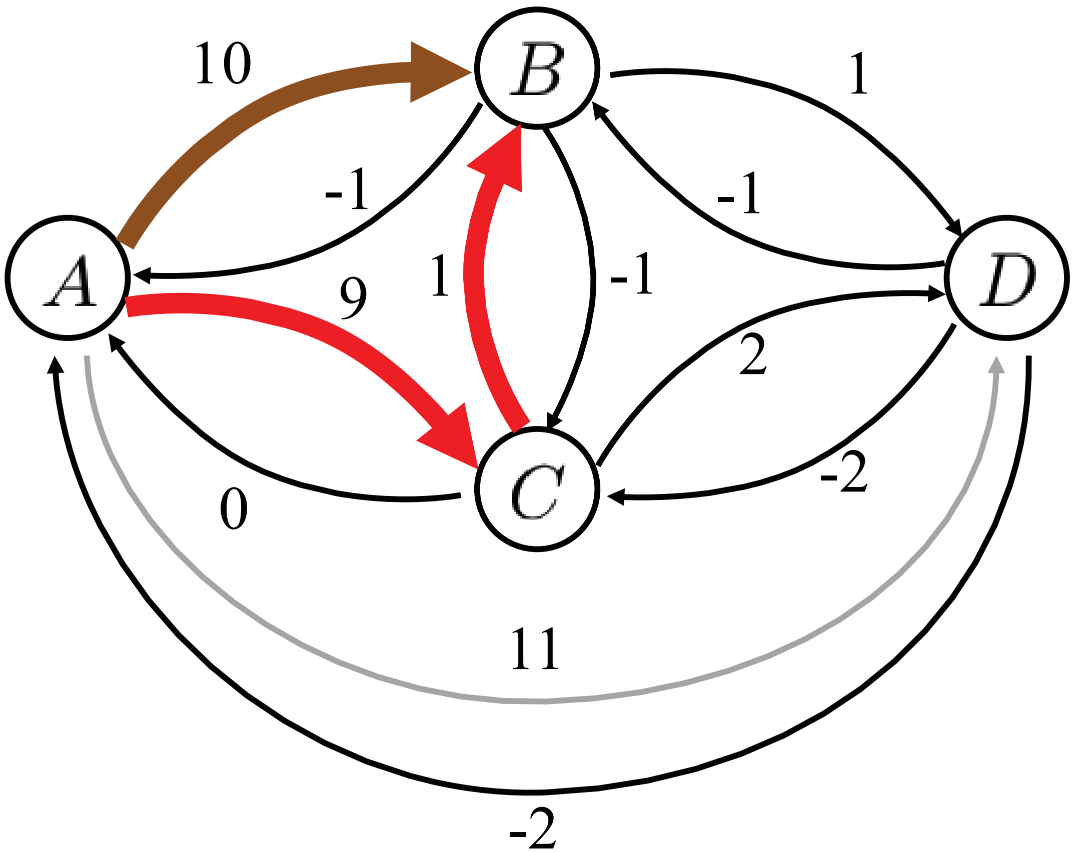
Dominance example



Upper dominated!

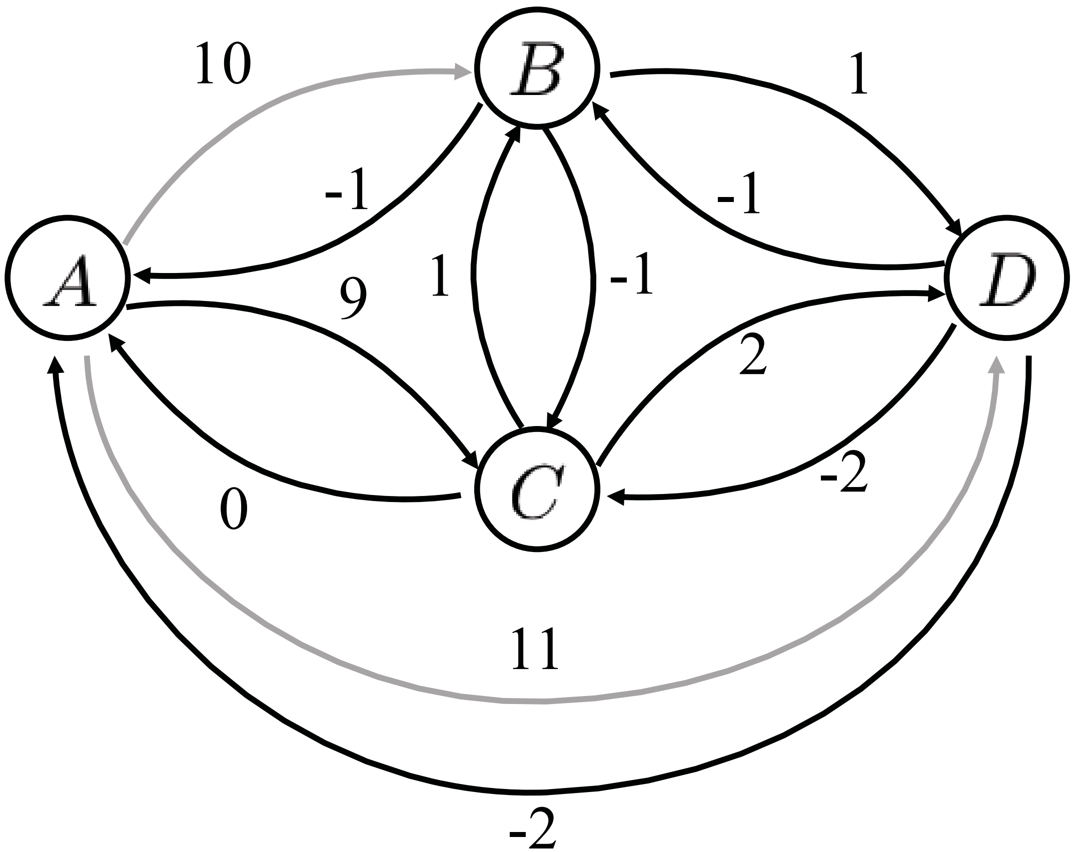
Dominance example

Upper dominated!

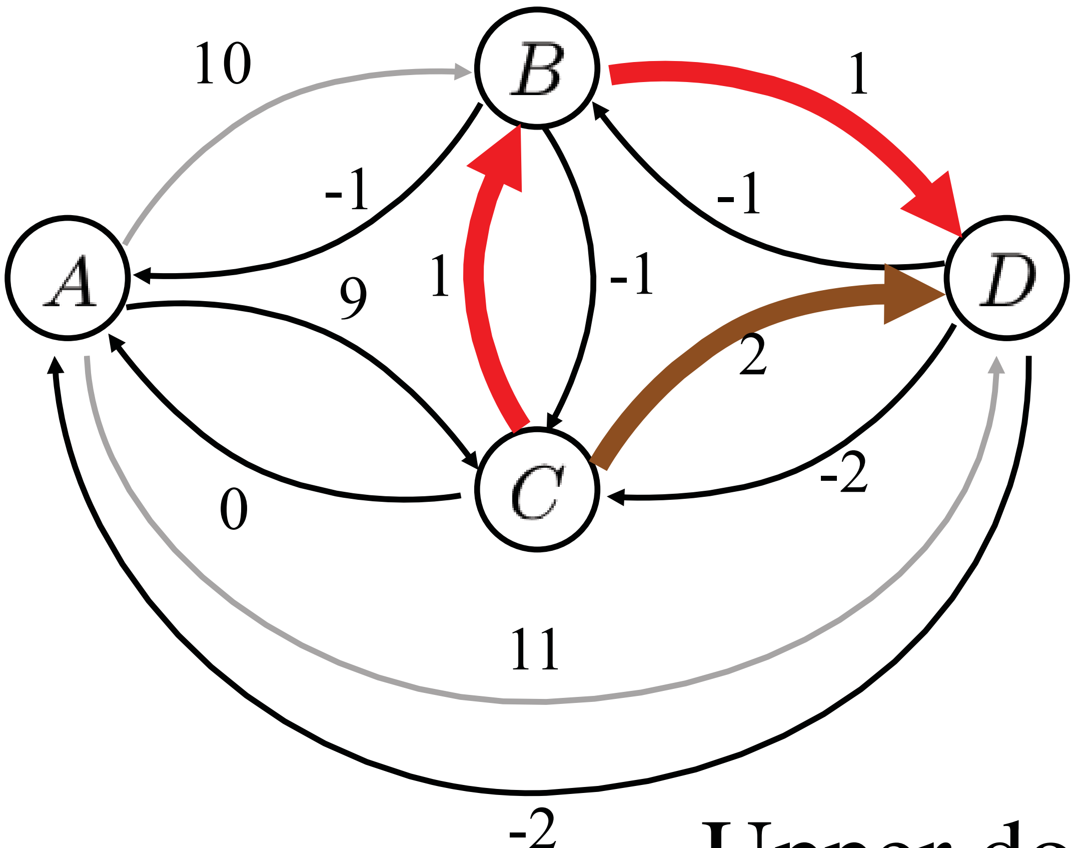


Dominance example

Upper dominated!

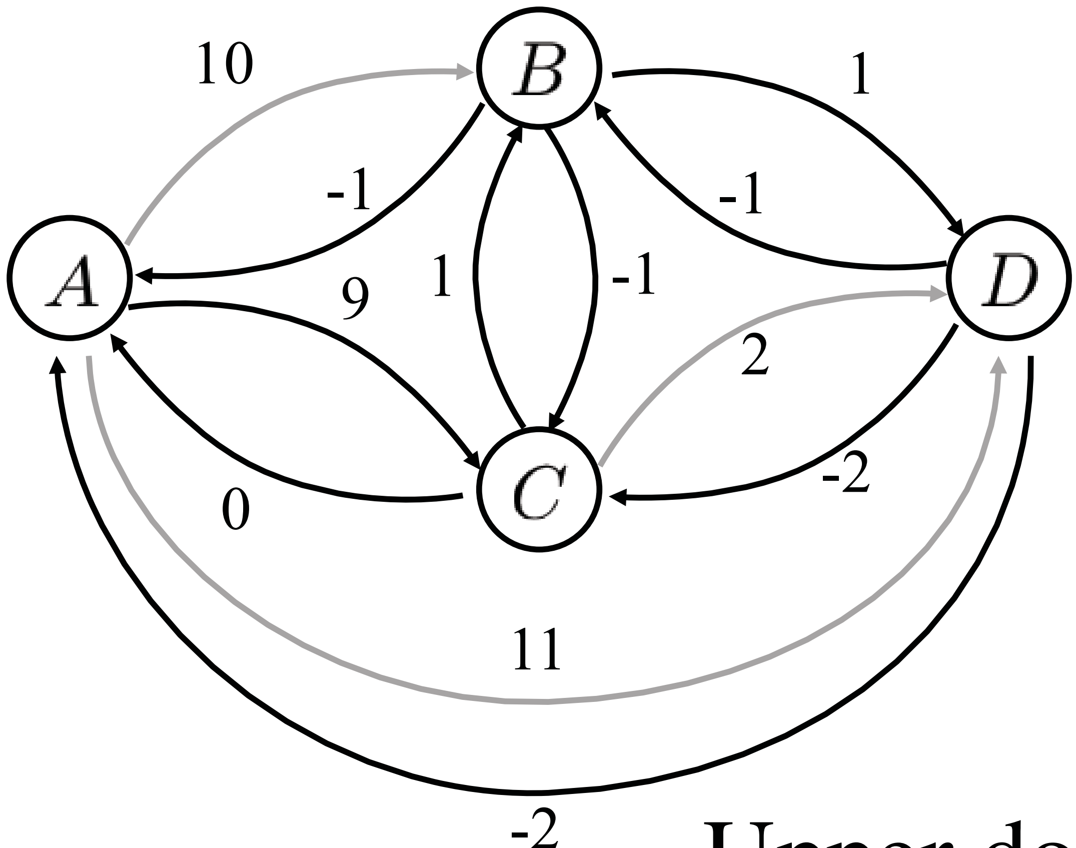


Dominance example



Upper dominated!

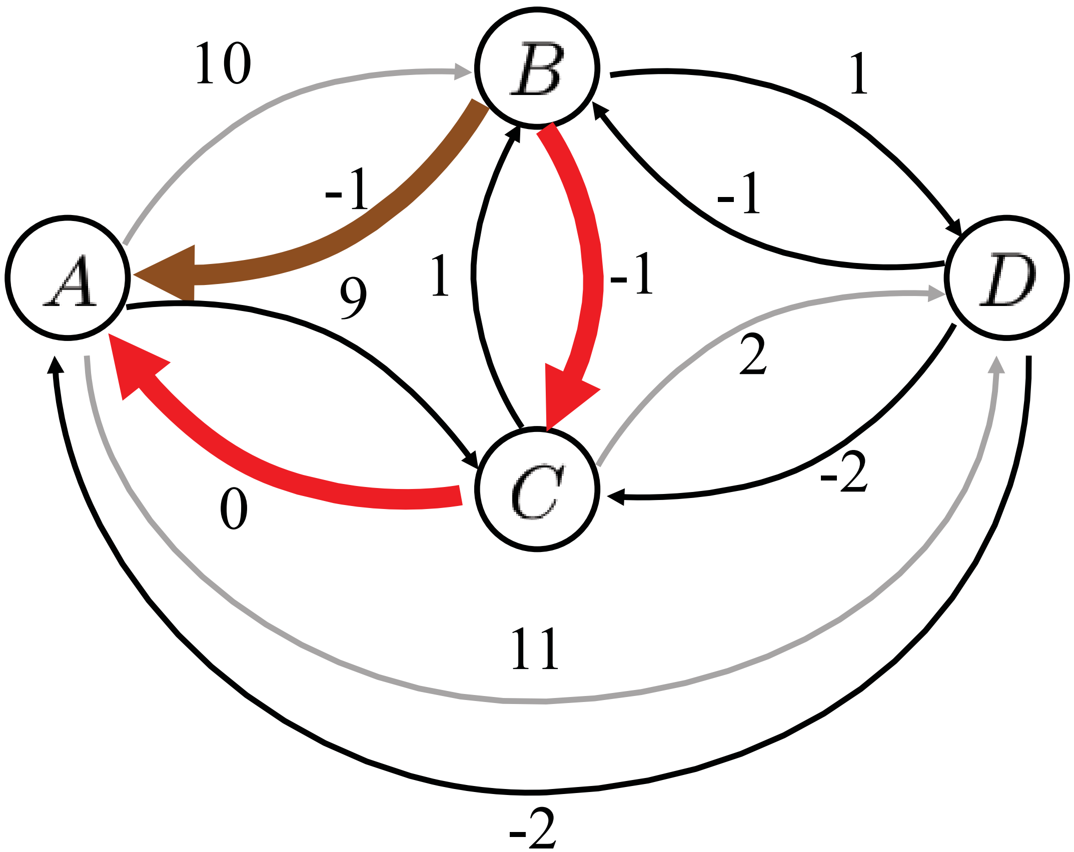
Dominance example



Upper dominated!

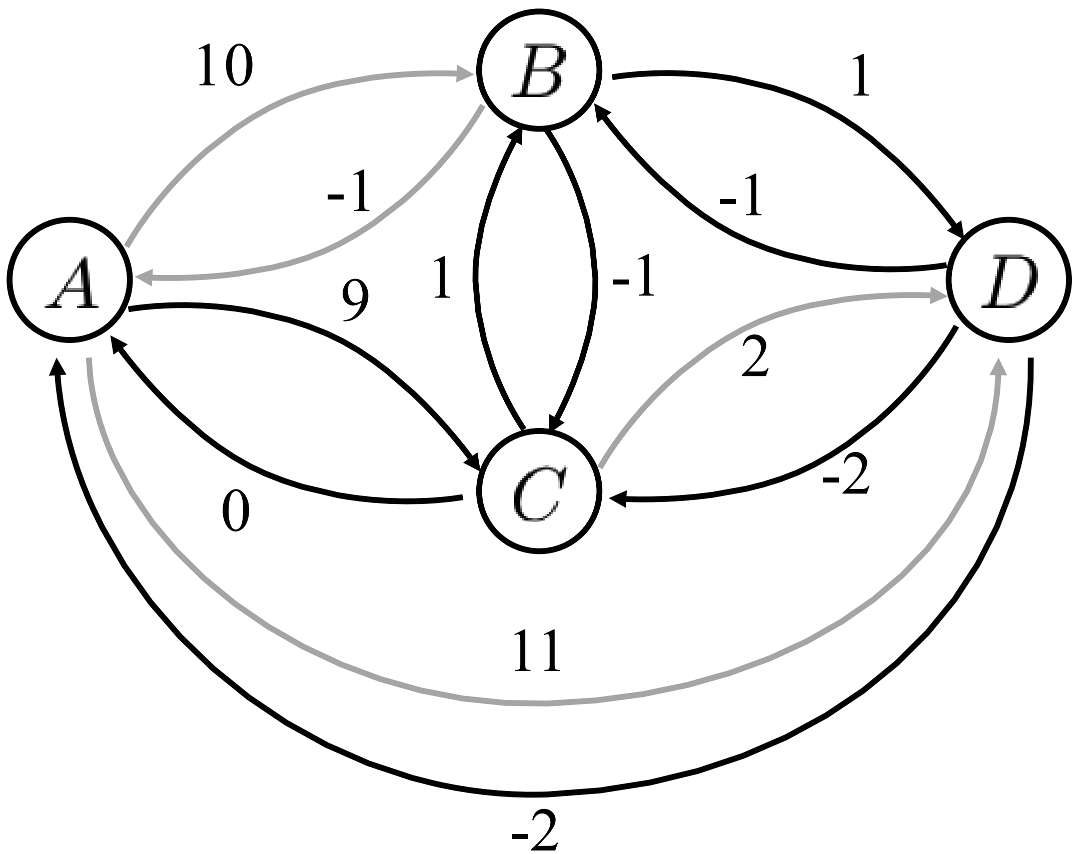
Dominance example

Lower dominated!

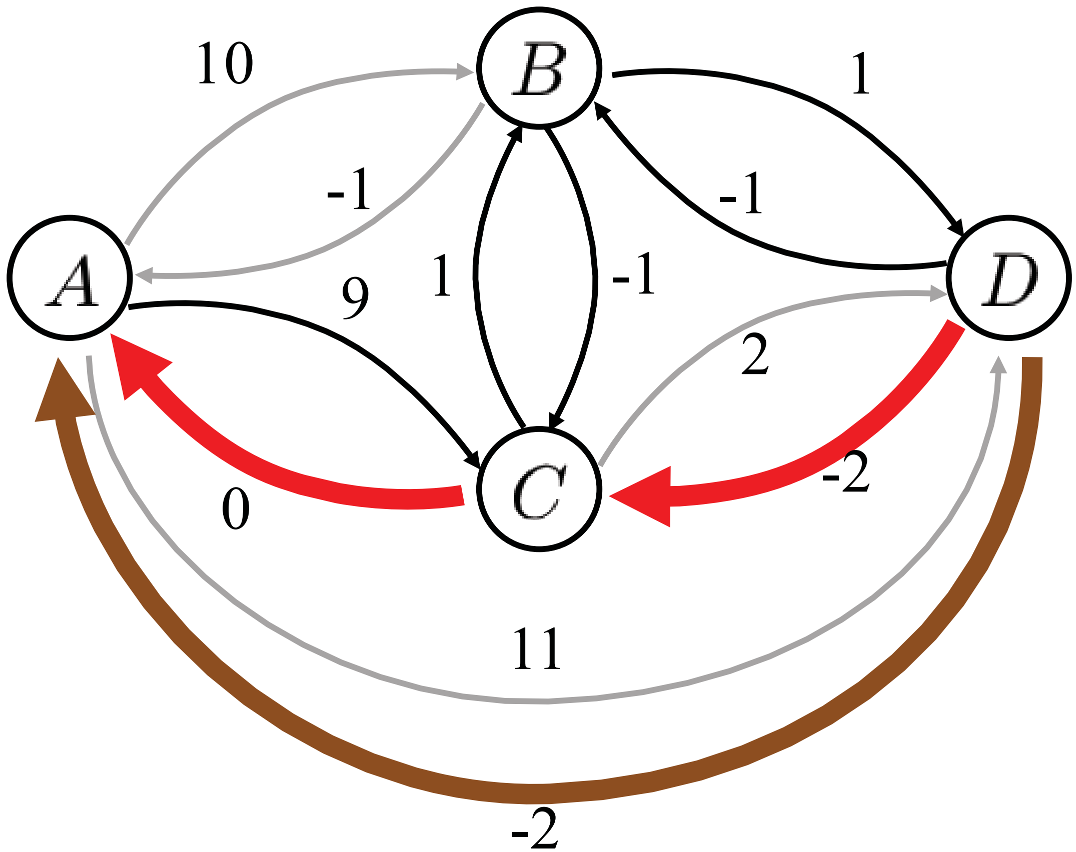


Dominance example

Lower dominated!

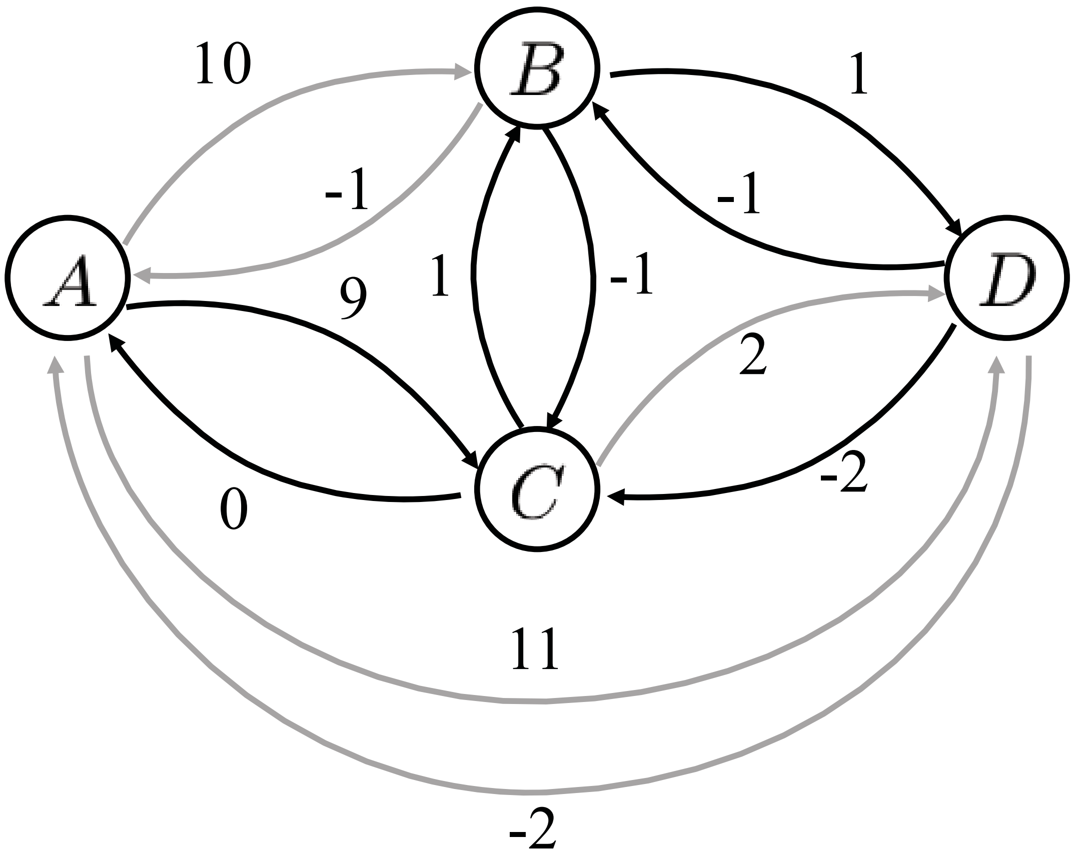


Dominance example



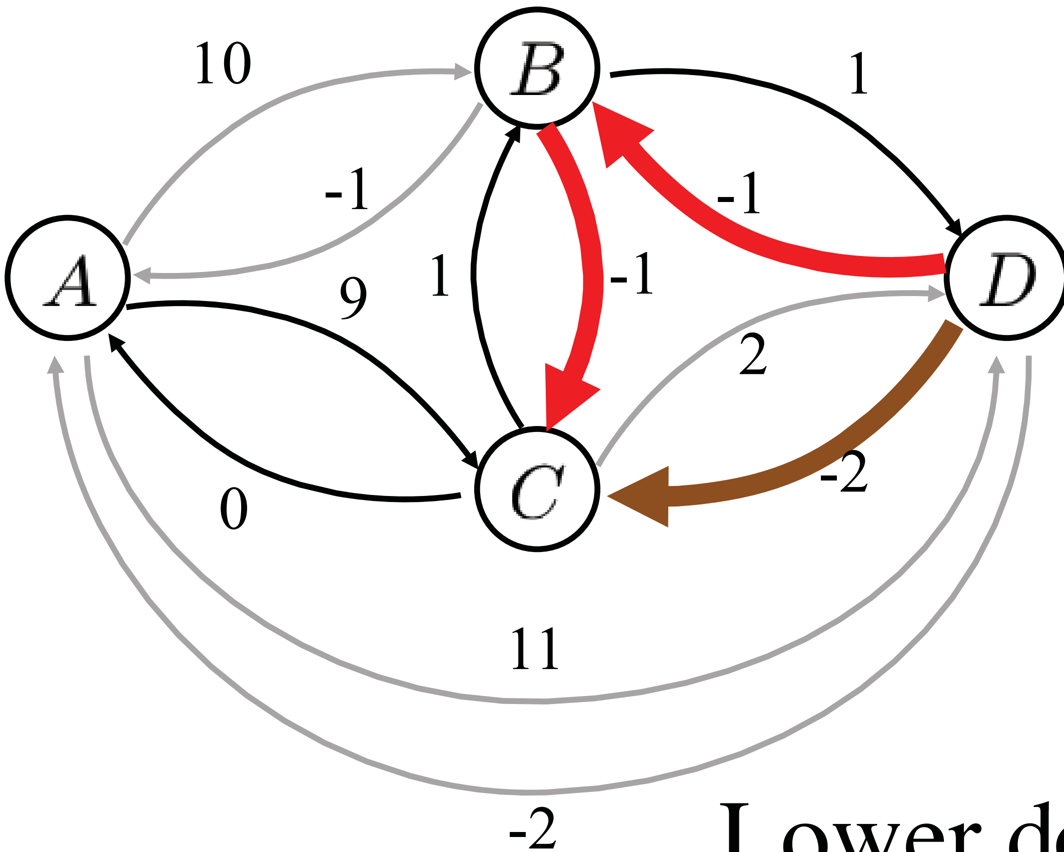
Lower dominated!

Dominance example



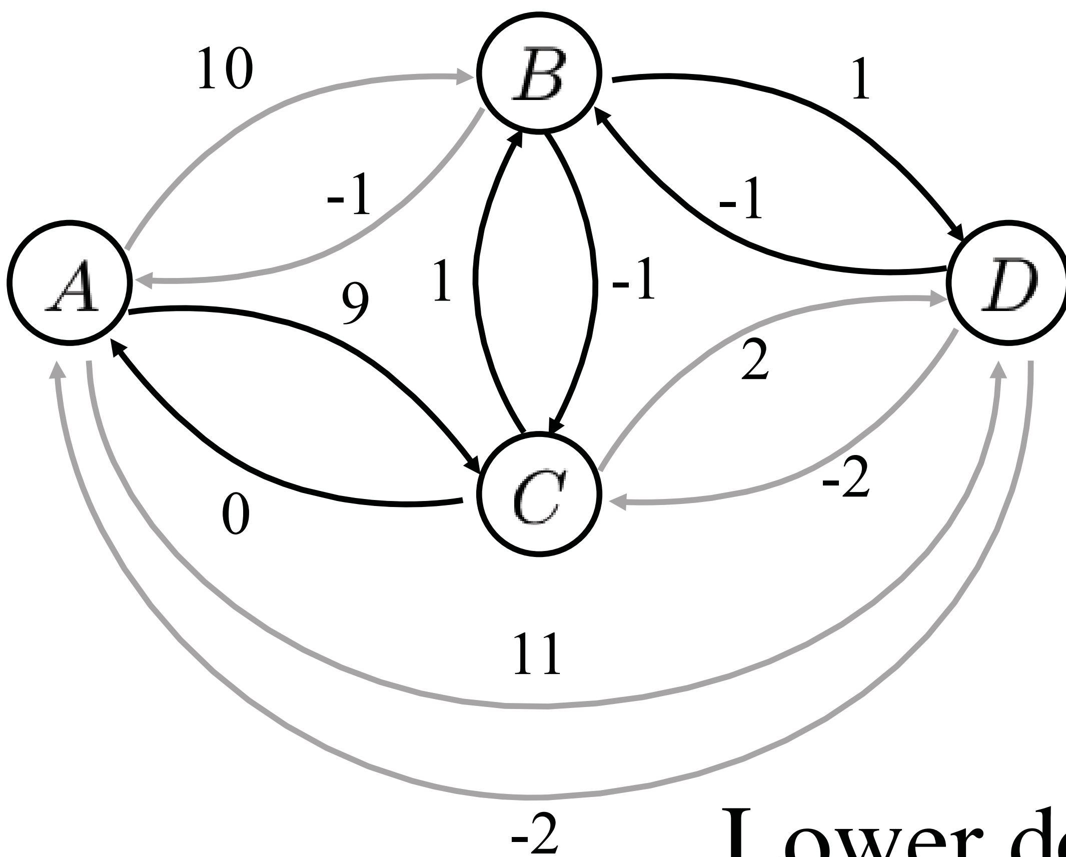
Lower dominated!

Dominance example



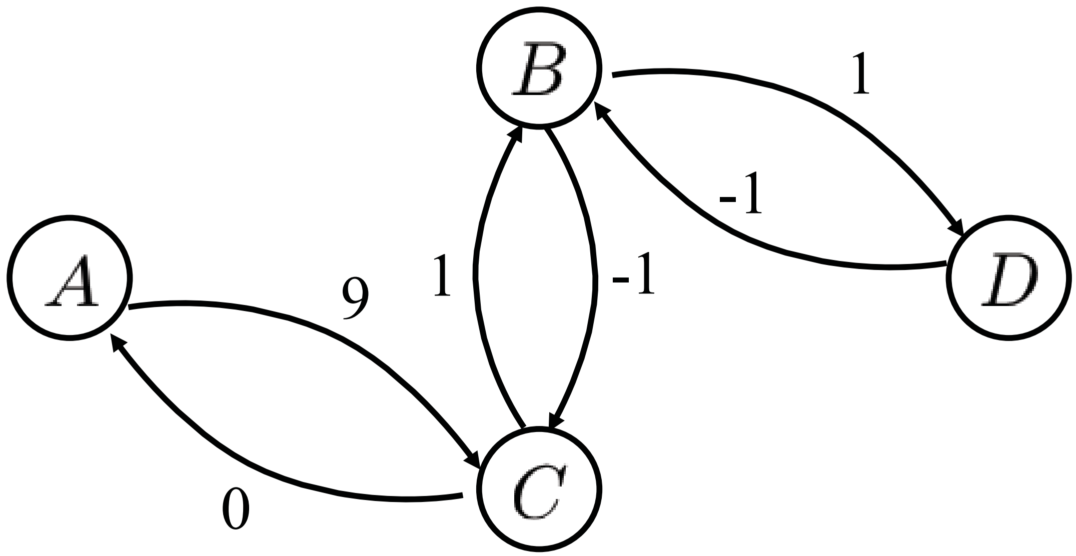
Lower dominated!

Dominance example

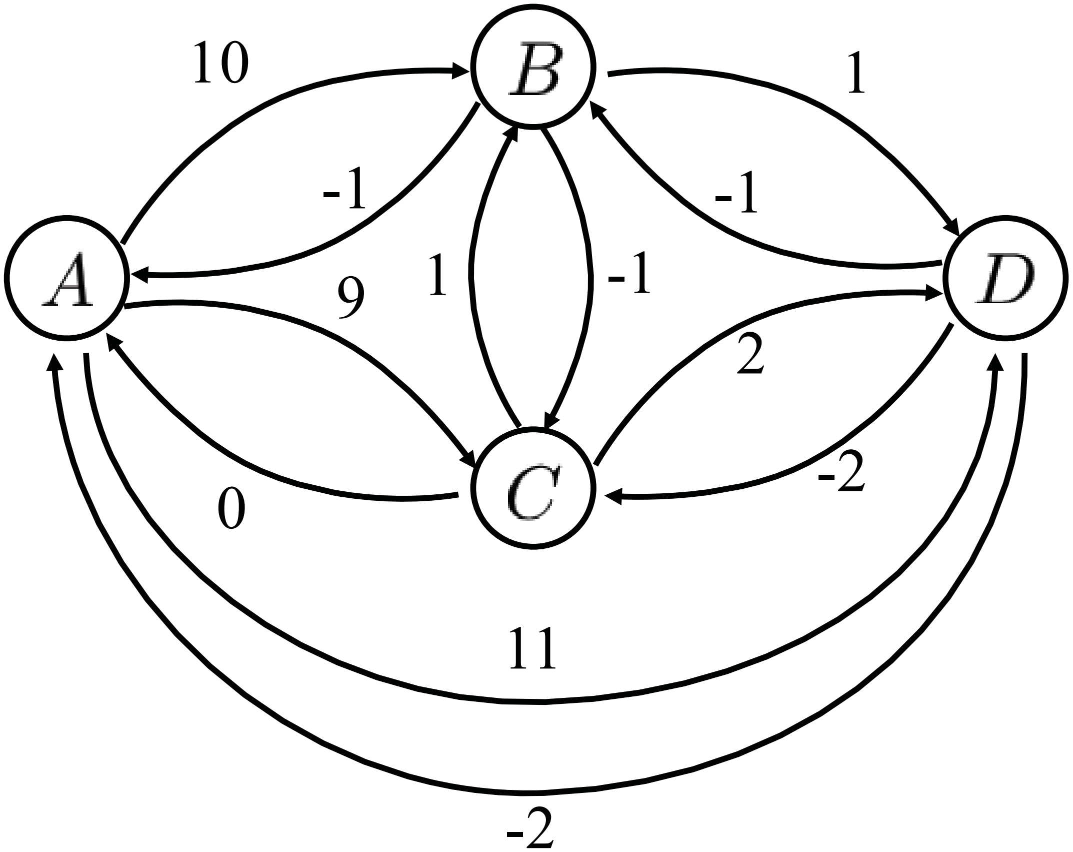


Lower dominated!

Dominance example

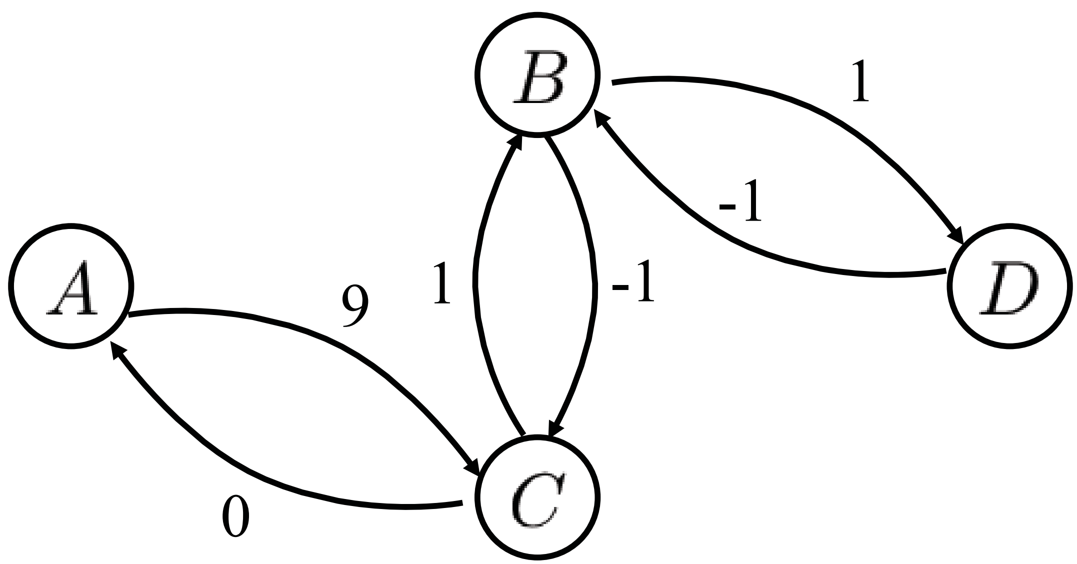


Dominance example

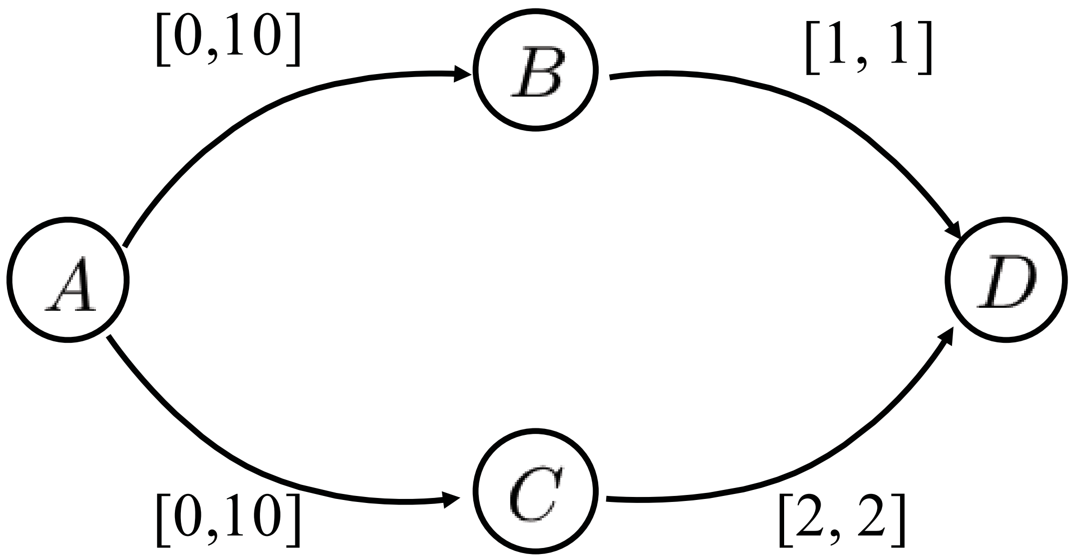


Original APSP distance graph

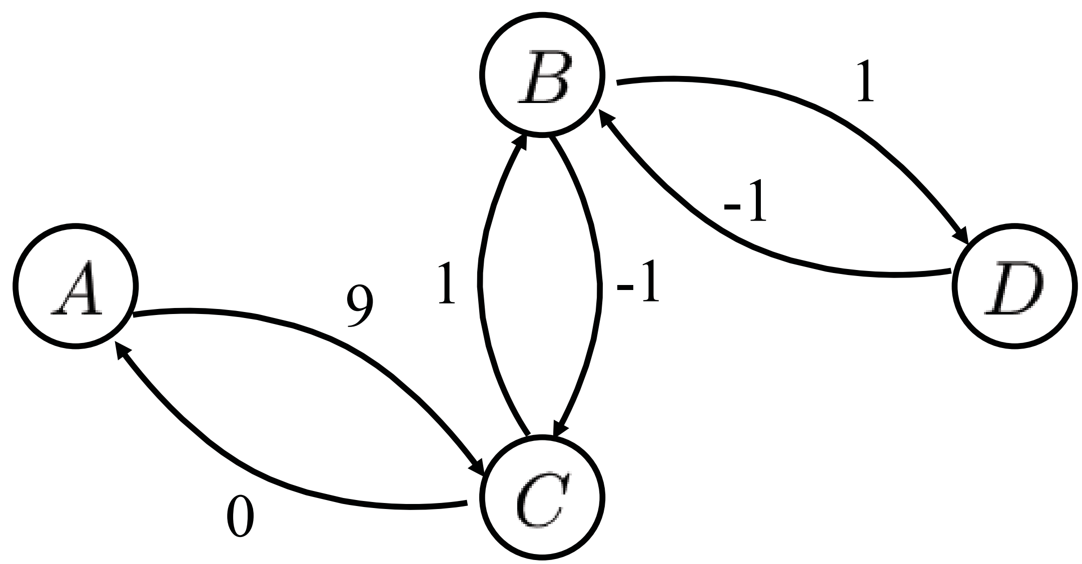
... now in ***minimal dispatchable form!***



Dominance example



Original STN



... now in ***minimal dispatchable form!***

FilteringAlgorithm(*G*)

Input: A dispatchable APSP-graph G

Output: A minimal dispatchable graph

for each pair of intersecting edges in G
    if both dominate each other
        if neither is marked
            arbitrarily mark one for elimination
        end if
    else if one dominates the other
        mark dominated edge for elimination
    end if
end for
remove all marked edges from graph
return *G*

Avoiding Intermediate Graph Explosion

- Problem:
  - All pairs shortest path table computation consumed $O(n^2)$ space
  - Only used as an intermediate - not needed after minimal dispatchable graph obtained.

- Solution:
  - Interleave process of APSP construction with edge elimination.
    - Never have to build whole APSP graph.

[Tsarmardinos 1998]

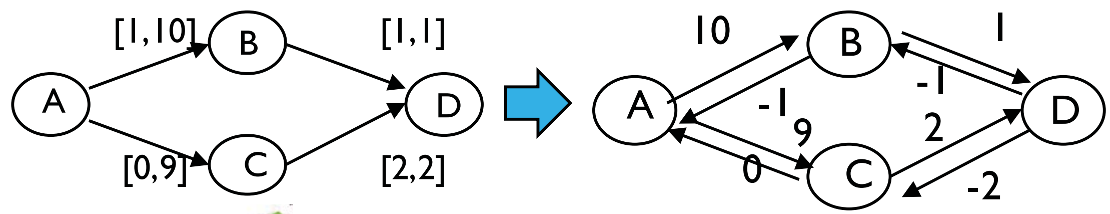
Recap

- Recap
  - To schedule online, times must monotonically increase - use enablement conditions
  - Running online allows greater flexibility to fluctuations
  - However, propagation costs can be large for large graphs
  - Can reduce edges by using domination to make graph smaller

To Execute a Temporal Plan

Schedule Offline

STN D Graph



1. Describe Temporal Plan

2. Test Consistency

Detect negative loops (SSSP).

3. Schedule Plan

APSP + Decomposition.

offline

online

4. Execute Plan

[Dechter, Meiri, Pearl 91]

To Execute a Temporal Plan

Schedule Offline

1. Describe Temporal Plan

2. Test Consistency

3. Schedule Plan

offline

online

4. Execute Plan

Problem: delays and fluctuations in task duration can cause plan failure.

Observation: temporal constraints leave room to adapt.

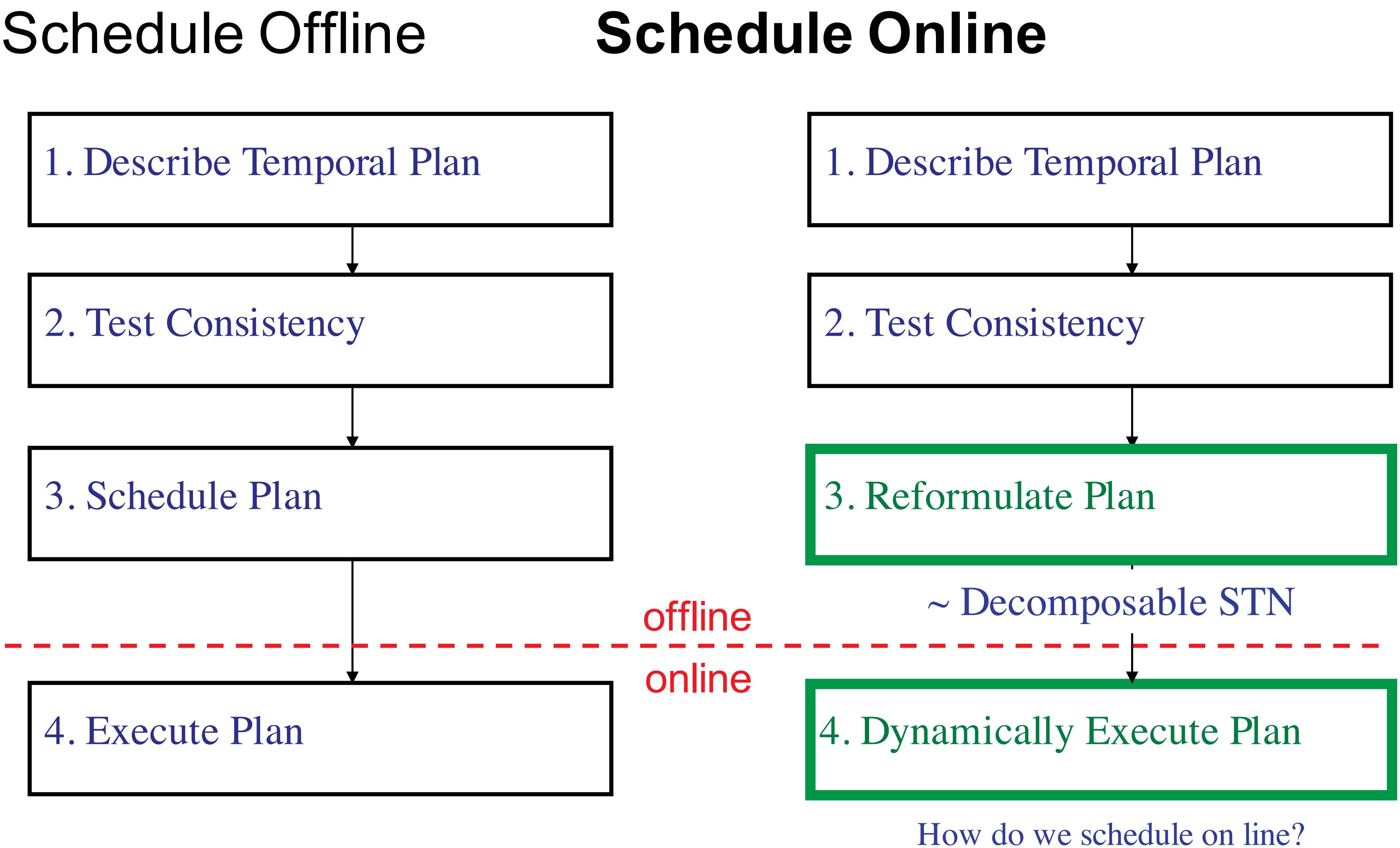
Flexible Execution adapts through dynamic scheduling:

Assign time to event when executed.

- Guarantee that all constraints will be satisfied.
- Schedule with low latency through pre-compilation.

[Muscettola, Morris, Tsmardinos KR98]

To Execute a Temporal Plan



Outline: To Execute a Temporal Plan

Schedule Online

1. Describe Temporal Plan

2. Test Consistency

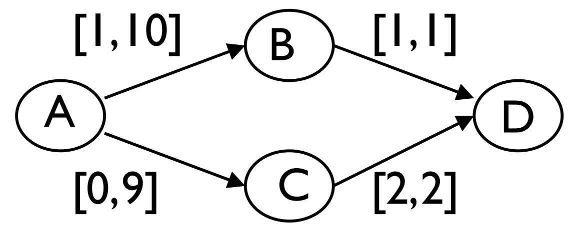
3. Reformulate Plan

offline

online

4. Dynamically Execute Plan

[Muscettola, Morris, Tsamardinos KR98]



To minimize latency remove redundant edges.