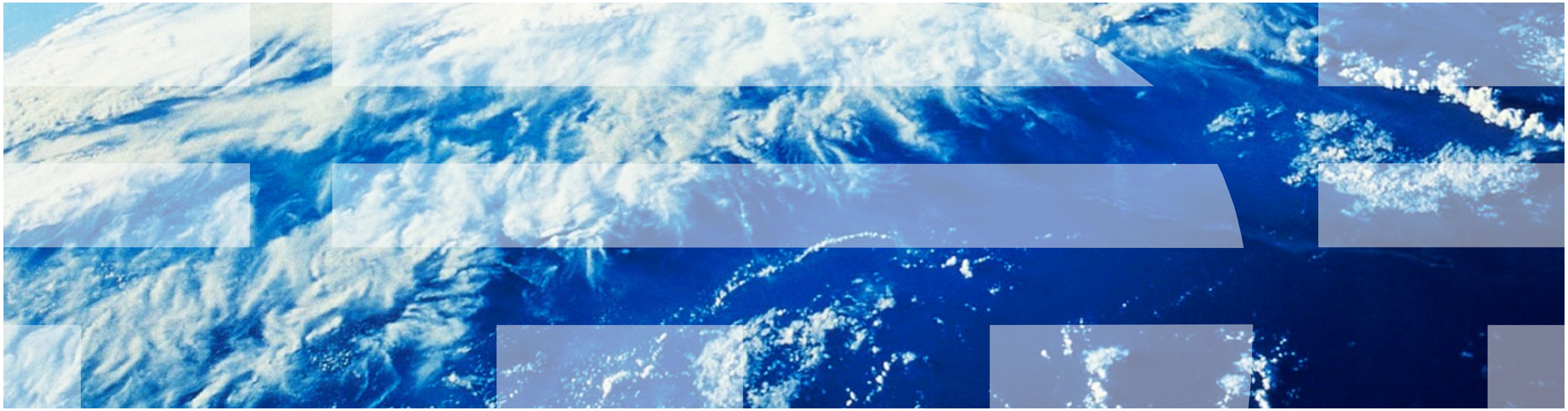
Scheduling Constraints, Propagation

CP Optimizer Development Team

Petr Vilím



Lazy clause generation and CP-based scheduling

- Lazy Clause Generation:
  - Analyze failures
  - Dynamically (lazily) add constraints (clauses) to avoid failing again for the same reason
  - Filtering algorithms not that important

[1] Schutt, Feydy, Stuckey, Wallace: Solving RCPSP/max by lazy clause generation
Journal of Scheduling 2012

noOverlap Constraint (unary/disjunctive resource)

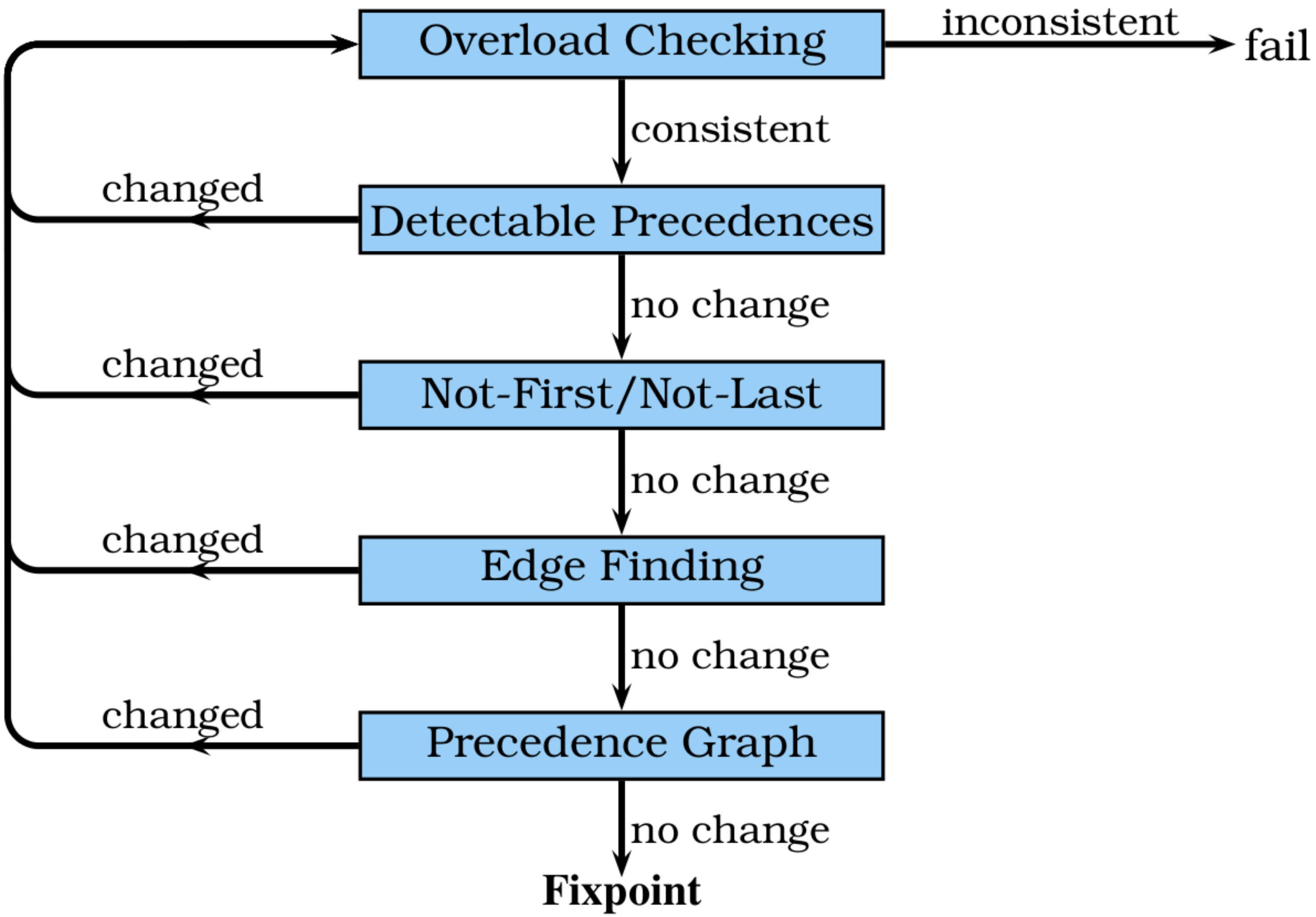
[1] Vilím: Global Constraints in Scheduling, PhD thesis, 2007

Propagation algorithms

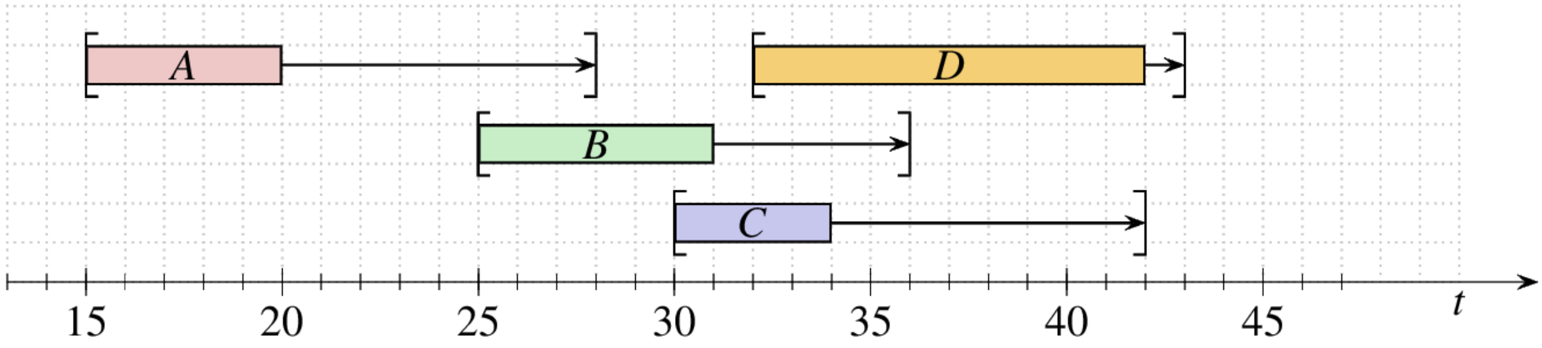
- *Overload Checking* (fail detection)
  - **O(n):** [Fahimi, Quimper]
- *Edge-Finding*
  - **O(n log n):** [Carlier & Pinson 1994], [Vilím]
  - **O(n2):** [Martin & Shmoys 96], [Wolf 2003], [Nuijten].
- *Not-Fist/Not-Last*
  - **O(n2):** [Baptiste & Le Pape 1996], [Torres & Lopez 1999], [Wolf 2003]
  - **O(n log n):** [Vilím]
- *Detectable Precedences*
  - **O(n log n):** [Vilím]
  - **O(n):** [Fahimi, Quimper]
- *…*

Each algorithm removes different type of inconsistent values, therefore they can be used together to achieve better pruning.

Fixpoint


Overload Checking
inconsistent
fail
consistent
Detectable Precedences
changed
no change
Not-First/Not-Last
changed
no change
Edge Finding
changed
no change
Precedence Graph
changed
no change
Fixpoint

Example: no solution (overload)

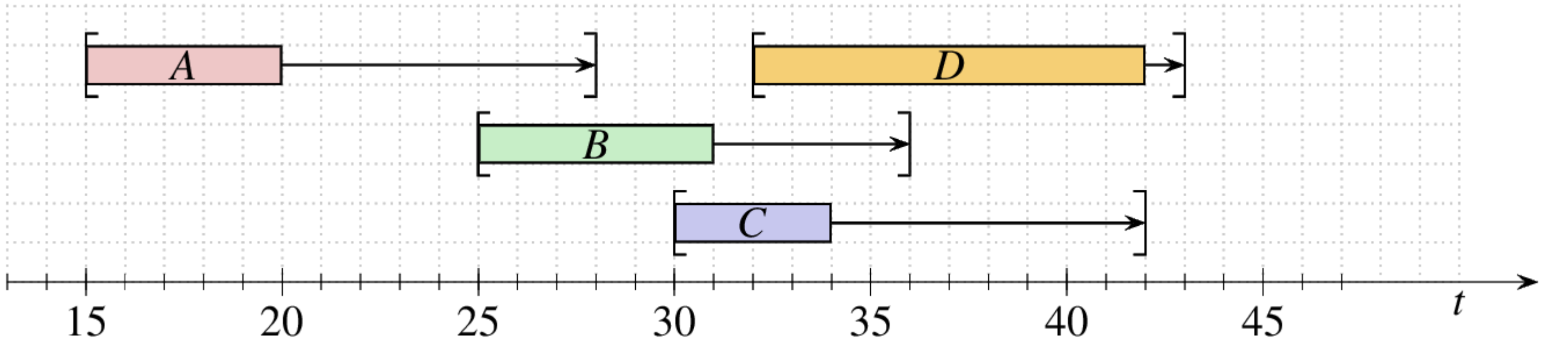


Traditional explanation:

- Union of time windows of {B, C, D} is [25, 43], its length is 18.
- Total duration of {B, C, D} is 6 + 4 + 10 = 20.
- 18 < 20 → no solution.

Leads to $O(n^2)$ algorithm.

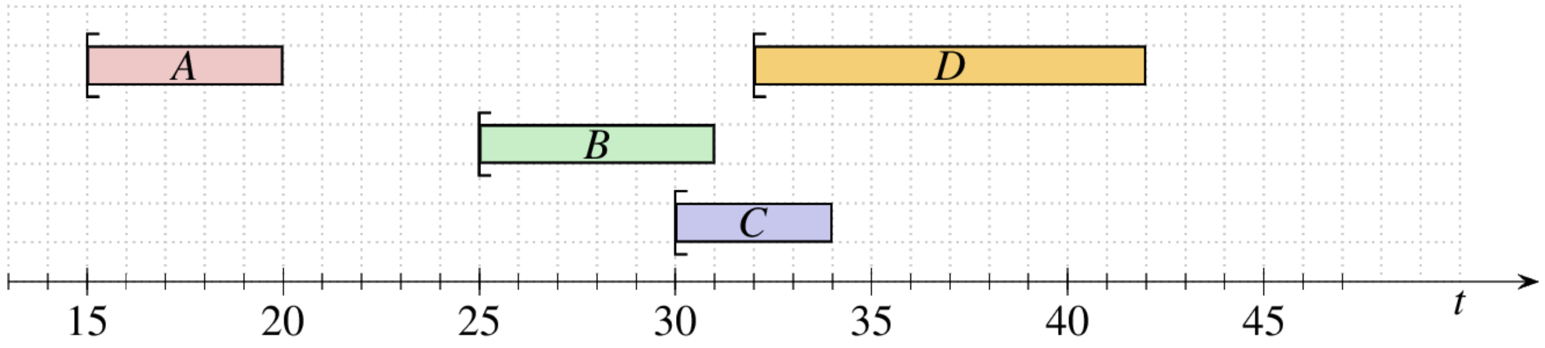
Example: no solution (overload)



Alternative explanation (leads to O(n log n) algorithm):

- Lets relax the problem by ignoring deadlines (all $\text{lct}_i = \infty$).

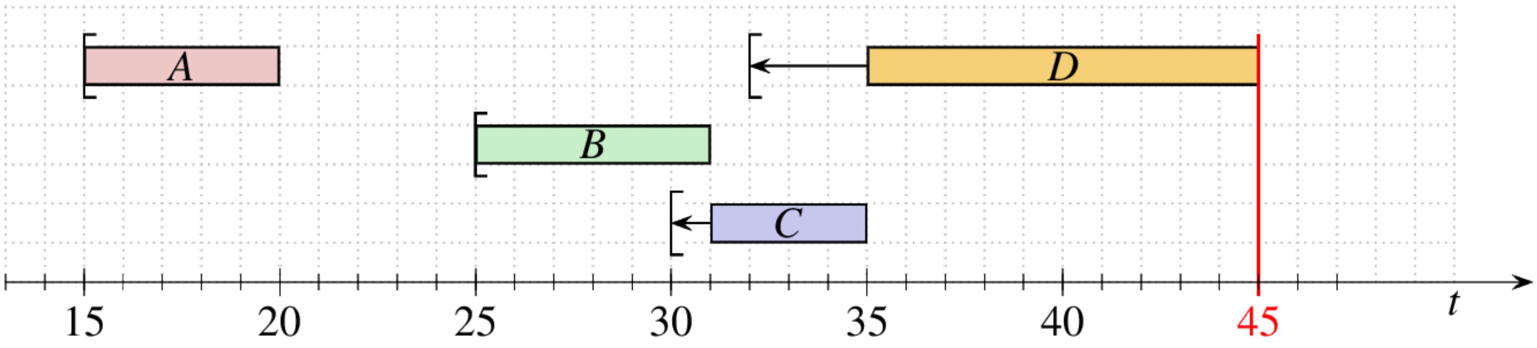
Example: no solution (overload)



Alternative explanation (leads to O(n log n) algorithm):

- Lets relax the problem by ignoring deadlines (all $\mathrm{lct}_i = \infty$).
- With this relaxation, what is *earliest completion time of set {A, B, C, D}*?

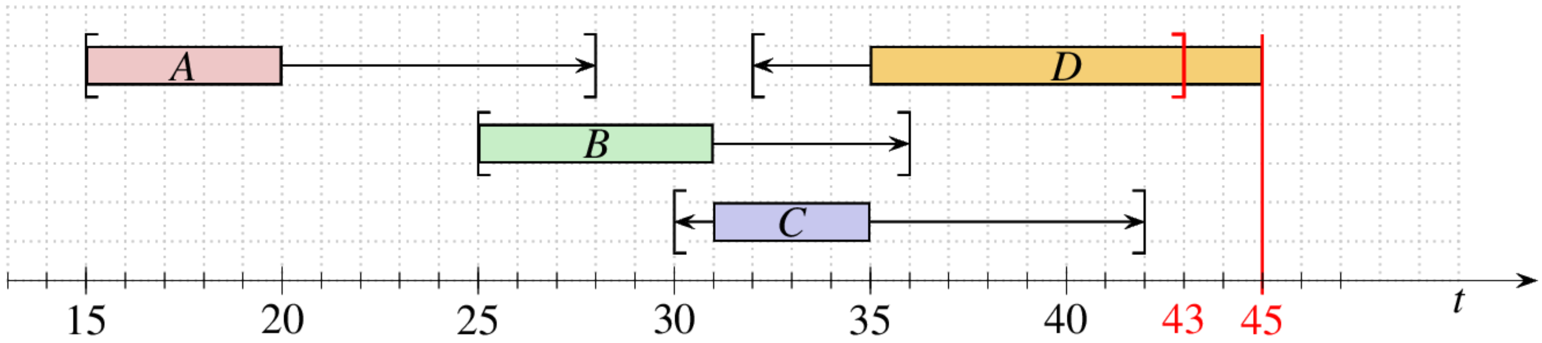
Example: no solution (overload)



Alternative explanation (leads to O(n log n) algorithm):

- Lets relax the problem by ignoring all deadlines (assuming all $\text{lct}_i = \infty$).
- With this relaxation, what is *earliest completion time of set {A, B, C, D}*?
  - $\text{est}_B + p_B + p_C + p_D = 25 + 6 + 4 + 10 = 45$

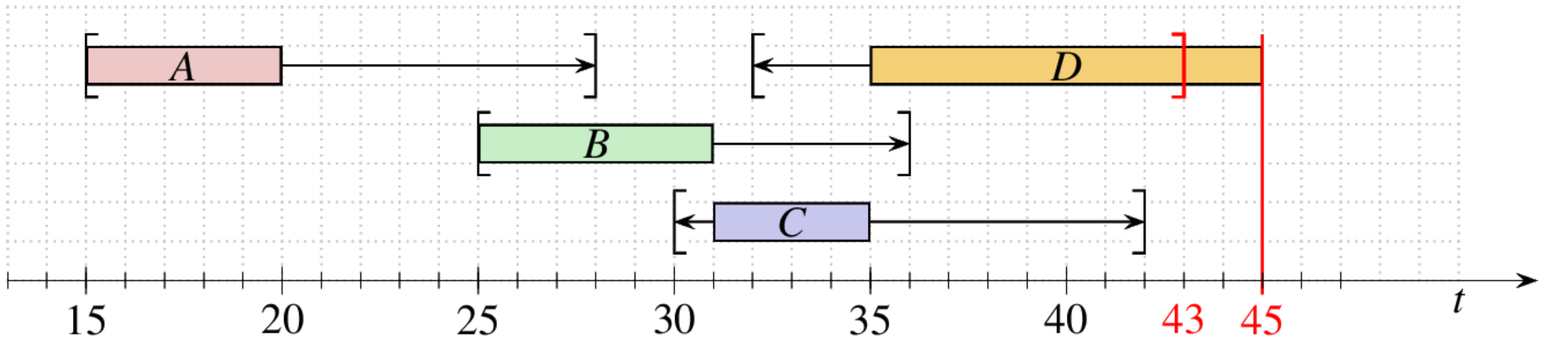
Example: no solution (overload)



Alternative explanation (leads to O(n log n) algorithm):

- Lets relax the problem by ignoring deadlines (all $\mathrm{lct}_i = \infty$).
- With this relaxation, what is *earliest completion time of set {A, B, C, D}*?
  - $\mathrm{est}_B + p_B + p_C + p_D = 25 + 6 + 4 + 10 = 45$
- But what is the deadline for {A, B, C, D}?
  - $\mathrm{lct}_{\{A,B,C,D\}} = \max\{\mathrm{lct}_A, \mathrm{lct}_B, \mathrm{lct}_C, \mathrm{lct}_D\} = \max\{28, 36, 42, 43\} = 43$
- 43 > 45 → no solution.

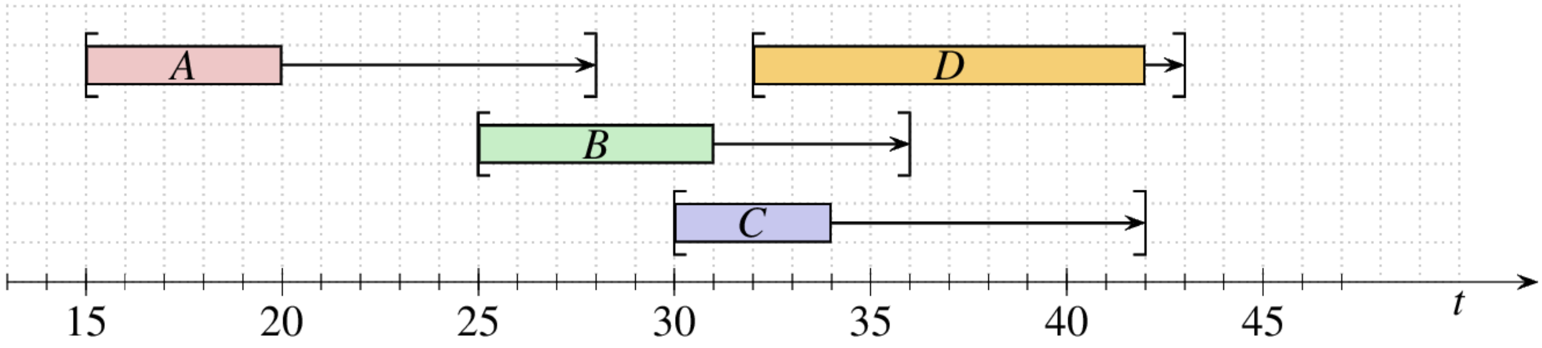
What is the difference?



- Classical explanation does not detect a problem for set {A, B, C, D}.
  - It have to check also subset {B, C, D} to recognize infeasibility.
  - There is O(n2) sets to check this way
    - One set for every combination of est_X and lct_Y.
- Alternative explanation correctly recognize problem for {A, B, C, D}.
  - There is O(n) sets to check this way
    - One for every lct_Y.

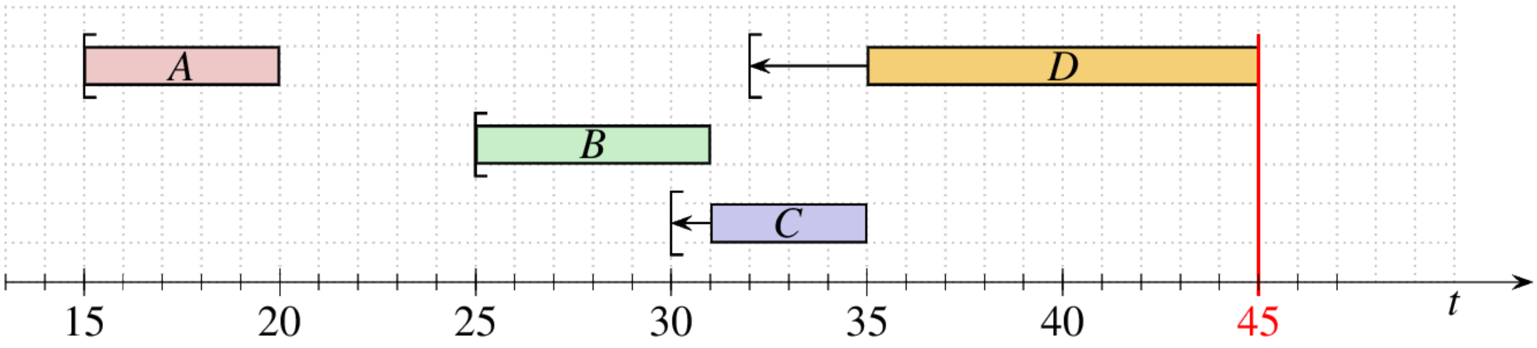
However, how to compute earliest completion times effectively?

Lets get more formal



- Let Ω is a set of activities.
  - Earliest start time of Ω is $est_\Omega = \min\{est_i, i \in \Omega\}$
  - Latest completion time of Ω is $lct_\Omega = \max\{lct_i, i \in \Omega\}$
  - Total duration of Ω is $p_\Omega = \text{sum}\{p_i, i \in \Omega\}$
- For Ω = {B, C, D}:
  - $est_\Omega = 25$
  - $lct_\Omega = 43$
  - $p_\Omega = 20$

Lets get more formal

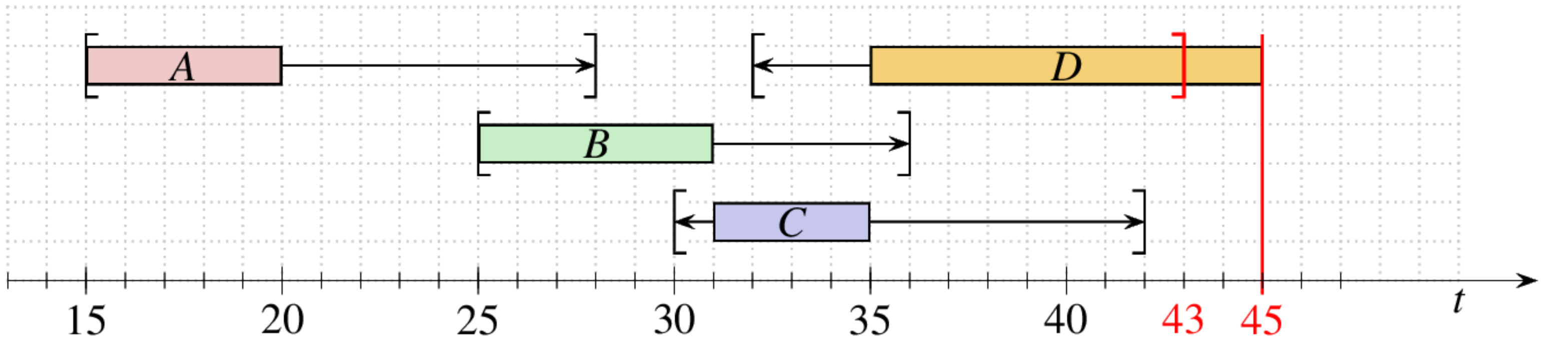


- Let Ω is a set of activities.
  - Earliest start time of Ω is $est_{\Omega} = \min\{est_i, i \in \Omega\}$
  - Latest completion time of Ω is $lct_{\Omega} = \max\{lct_i, i \in \Omega\}$
  - Total duration of Ω is $p_{\Omega} = \max\{lct_i, i \in \Omega\}$
- Earliest completion time of (another set of activities) Θ is:

$$\mathrm{ECT}_{\Theta} = \max\{\mathrm{est}_{\Omega} + \mathrm{p}_{\Omega},\ \Omega \subseteq \Theta\}$$

- For Θ = {A, B, C, D} the best Ω is {B, C, D} and $ECT_{\Theta} = 25 + 20 = 45$.

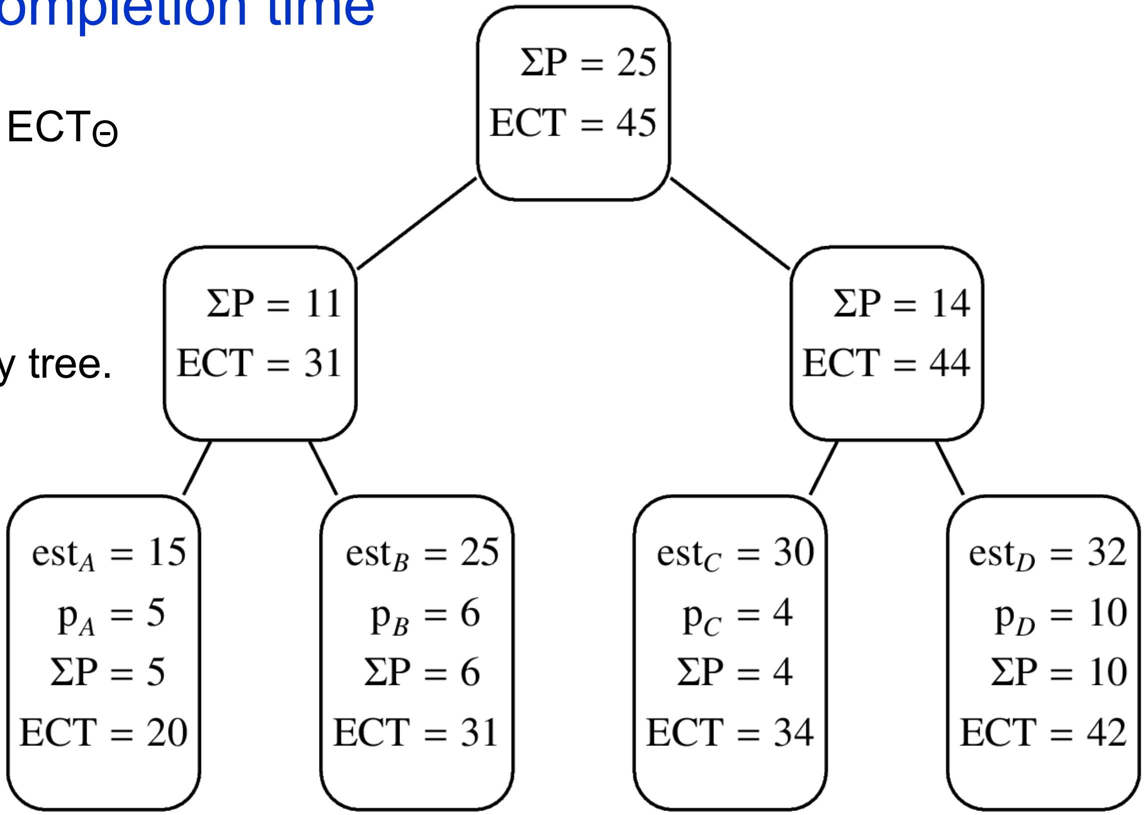
Overload rule



$$\underbrace{\mathrm{ECT}_{\Theta}}_{45} > \underbrace{\mathrm{lct}_{\Theta}}_{43} \Rightarrow \text{fail}$$

Computation of earliest completion time

- The goal is to quickly recompute ECT_Θ after a change of Θ such as:
  - addition of an activity into Θ
  - removal of an activity from Θ
- The idea: represent Θ by a binary tree.



$\Sigma P = 25$, $\mathrm{ECT} = 45$

$\Sigma P = 11$, $\mathrm{ECT} = 31$

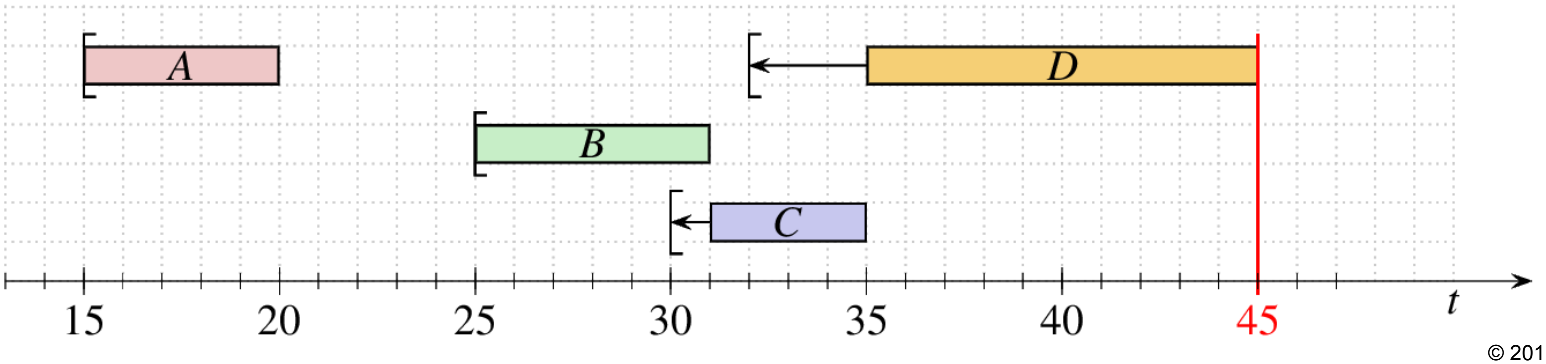
$\Sigma P = 14$, $\mathrm{ECT} = 44$

$\mathrm{est}_A = 15$, $p_A = 5$, $\Sigma P = 5$, $\mathrm{ECT} = 20$

$\mathrm{est}_B = 25$, $p_B = 6$, $\Sigma P = 6$, $\mathrm{ECT} = 31$

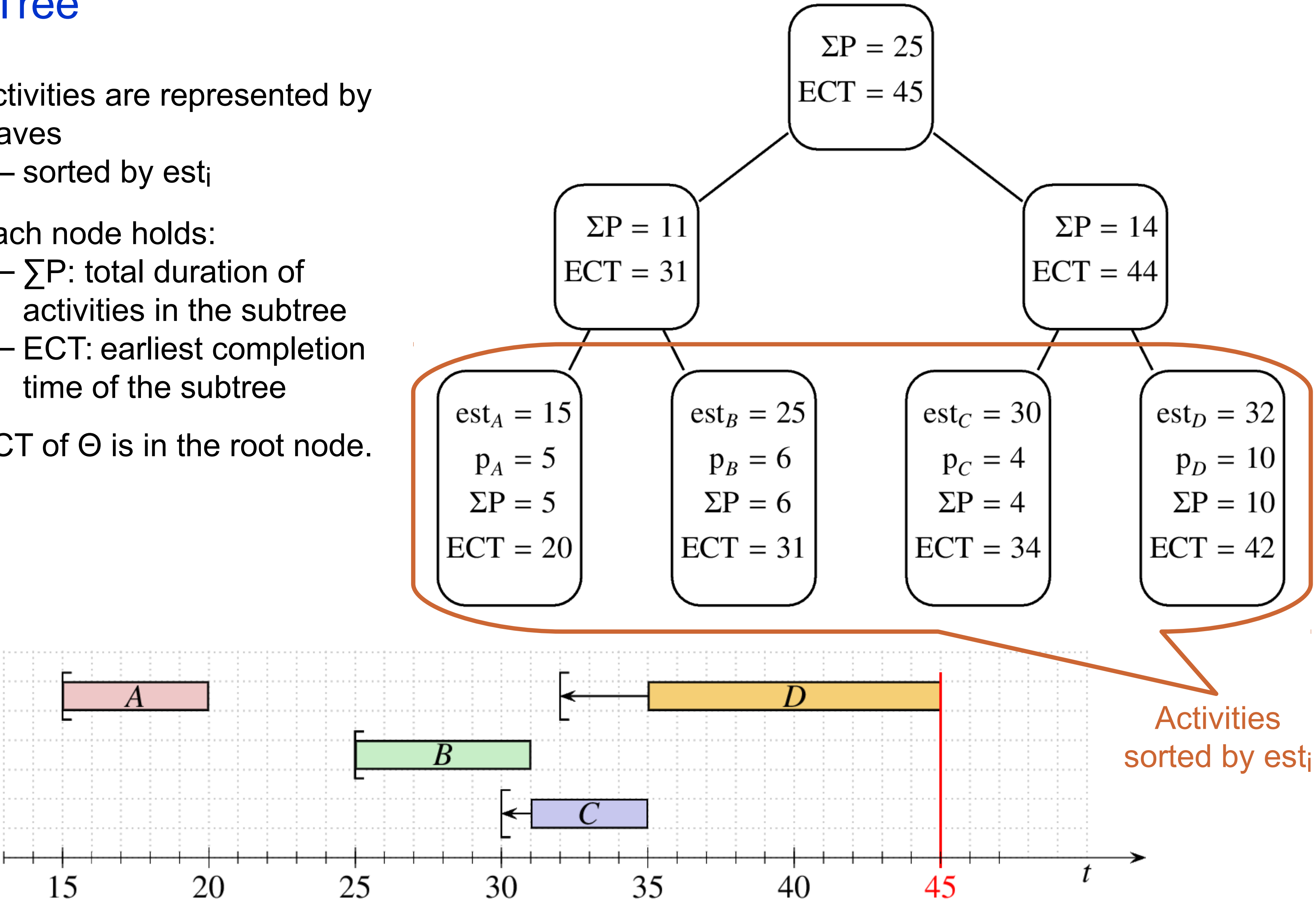
$\mathrm{est}_C = 30$, $p_C = 4$, $\Sigma P = 4$, $\mathrm{ECT} = 34$

$\mathrm{est}_D = 32$, $p_D = 10$, $\Sigma P = 10$, $\mathrm{ECT} = 42$



Θ-Tree

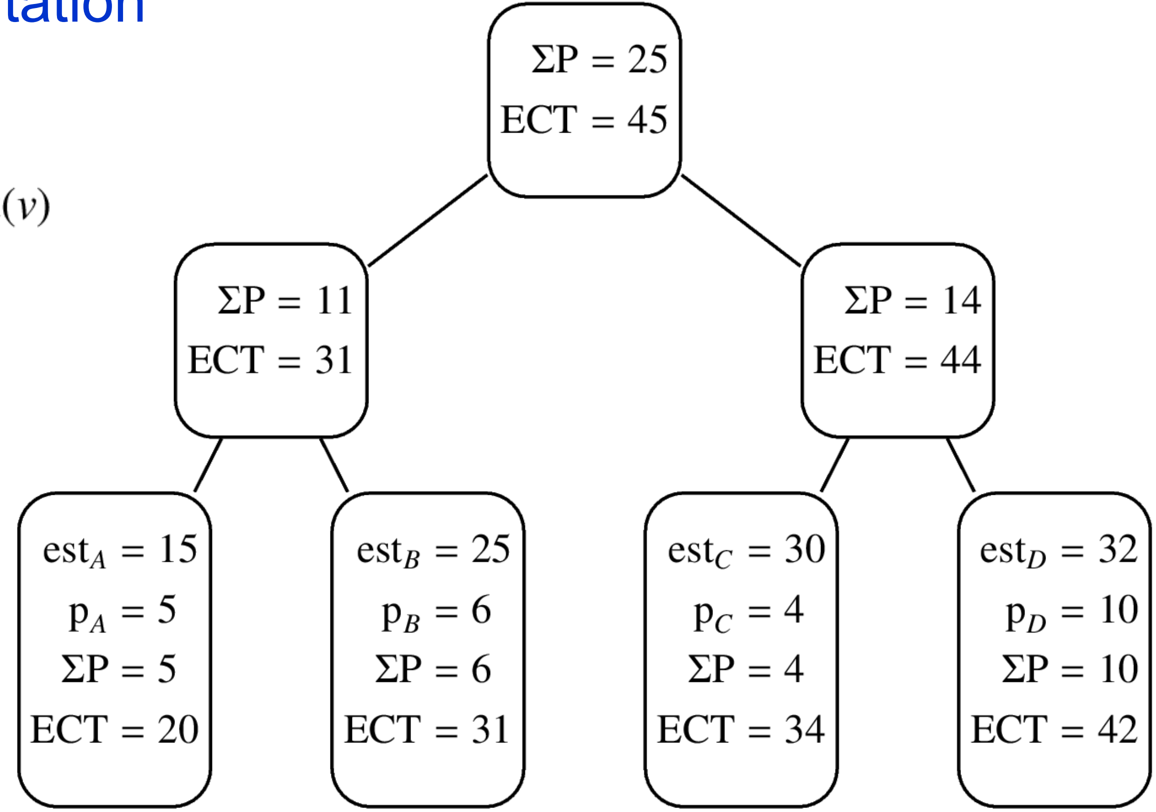
- Activities are represented by leaves
    - sorted by est_i
- Each node holds:
    - ∑P: total duration of activities in the subtree
    - ECT: earliest completion time of the subtree
- ECT of Θ is in the root node.



Activities sorted by est_i

Θ-Tree: recursive computation

$$\Sigma P_v = \Sigma P_{\mathrm{left}(v)} + \Sigma P_{\mathrm{right}(v)}$$



$\Sigma P = 25$
$ECT = 45$

$\Sigma P = 11$
$ECT = 31$

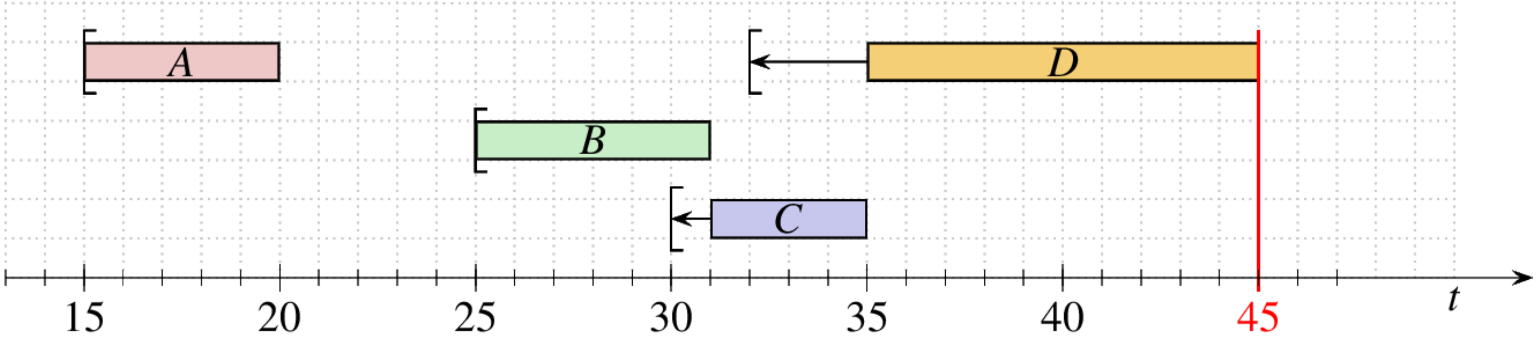
$\Sigma P = 14$
$ECT = 44$

$est_A = 15$
$p_A = 5$
$\Sigma P = 5$
$ECT = 20$

$est_B = 25$
$p_B = 6$
$\Sigma P = 6$
$ECT = 31$

$est_C = 30$
$p_C = 4$
$\Sigma P = 4$
$ECT = 34$

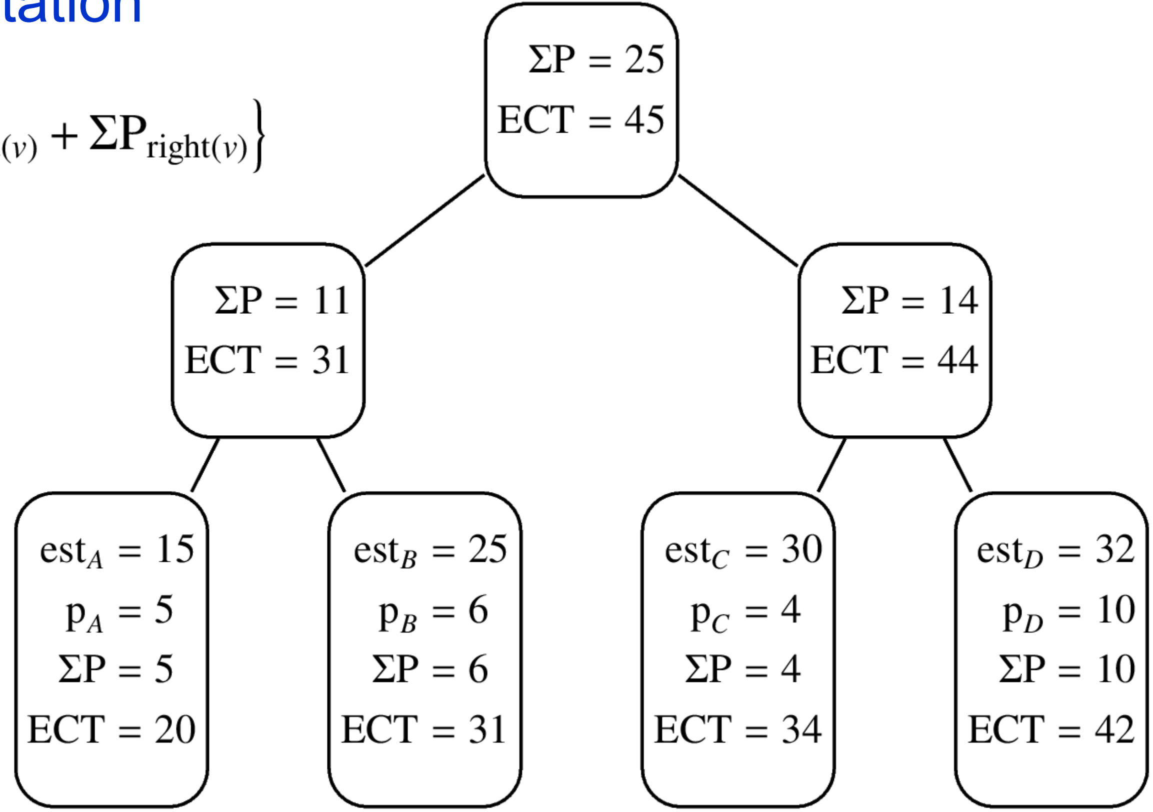
$est_D = 32$
$p_D = 10$
$\Sigma P = 10$
$ECT = 42$



Θ-Tree: recursive computation

$$\mathrm{ECT}_v = \max\left\{\mathrm{ECT}_{\mathrm{right}(v)},\ \mathrm{ECT}_{\mathrm{left}(v)} + \Sigma \mathrm{P}_{\mathrm{right}(v)}\right\}$$

$$\mathrm{ECT}_\Theta = \max\{\mathrm{est}_\Omega + \mathrm{p}_\Omega,\ \Omega \subseteq \Theta\}$$



$\Sigma \mathrm{P} = 25$
$\mathrm{ECT} = 45$

$\Sigma \mathrm{P} = 11$
$\mathrm{ECT} = 31$

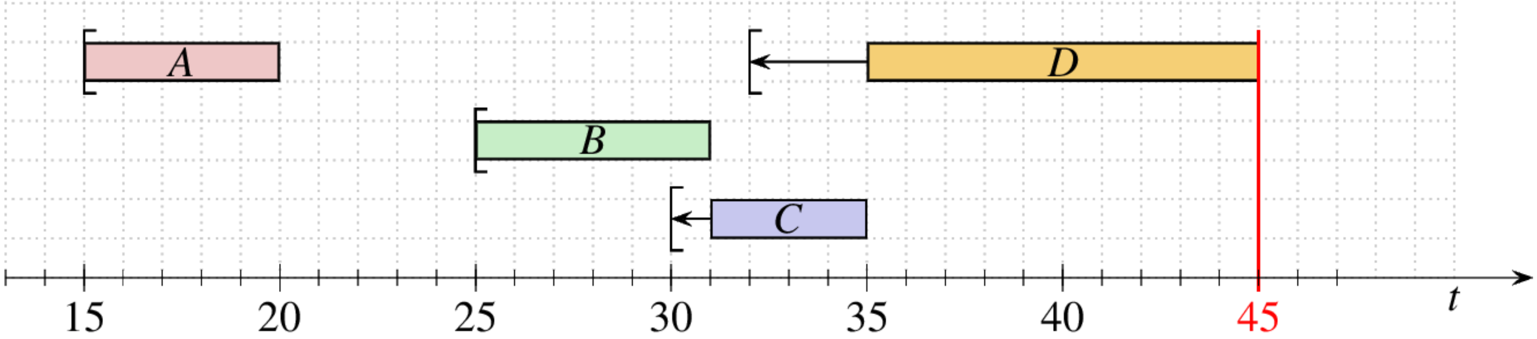
$\Sigma \mathrm{P} = 14$
$\mathrm{ECT} = 44$

$\mathrm{est}_A = 15$
$\mathrm{p}_A = 5$
$\Sigma \mathrm{P} = 5$
$\mathrm{ECT} = 20$

$\mathrm{est}_B = 25$
$\mathrm{p}_B = 6$
$\Sigma \mathrm{P} = 6$
$\mathrm{ECT} = 31$

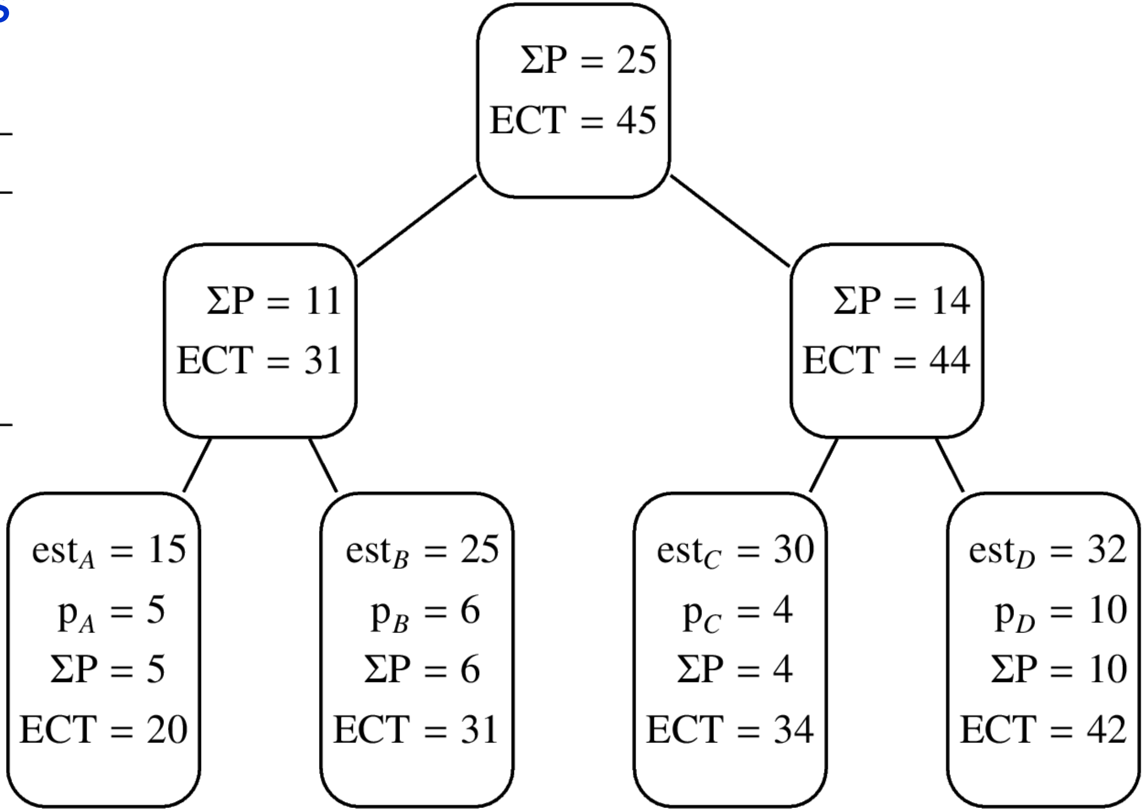
$\mathrm{est}_C = 30$
$\mathrm{p}_C = 4$
$\Sigma \mathrm{P} = 4$
$\mathrm{ECT} = 34$

$\mathrm{est}_D = 32$
$\mathrm{p}_D = 10$
$\Sigma \mathrm{P} = 10$
$\mathrm{ECT} = 42$



Θ-Tree: time complexities

Operation	**Time Complexity**
$\Theta := \emptyset$	$\mathcal{O}(1)$ or $\mathcal{O}(n \log n)$
$\Theta := \Theta \cup \{i\}$	$\mathcal{O}(\log n)$
$\Theta := \Theta \setminus \{i\}$	$\mathcal{O}(\log n)$
ECT_Θ	$\mathcal{O}(1)$



$\Sigma P = 25$
$\mathrm{ECT} = 45$

$\Sigma P = 11$
$\mathrm{ECT} = 31$

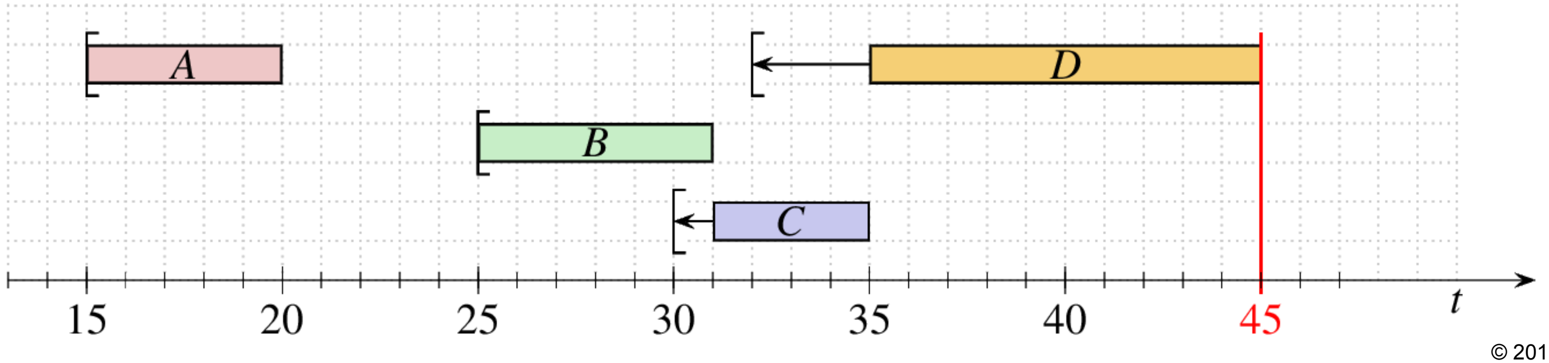
$\Sigma P = 14$
$\mathrm{ECT} = 44$

$\mathrm{est}_A = 15$
$p_A = 5$
$\Sigma P = 5$
$\mathrm{ECT} = 20$

$\mathrm{est}_B = 25$
$p_B = 6$
$\Sigma P = 6$
$\mathrm{ECT} = 31$

$\mathrm{est}_C = 30$
$p_C = 4$
$\Sigma P = 4$
$\mathrm{ECT} = 34$

$\mathrm{est}_D = 32$
$p_D = 10$
$\Sigma P = 10$
$\mathrm{ECT} = 42$



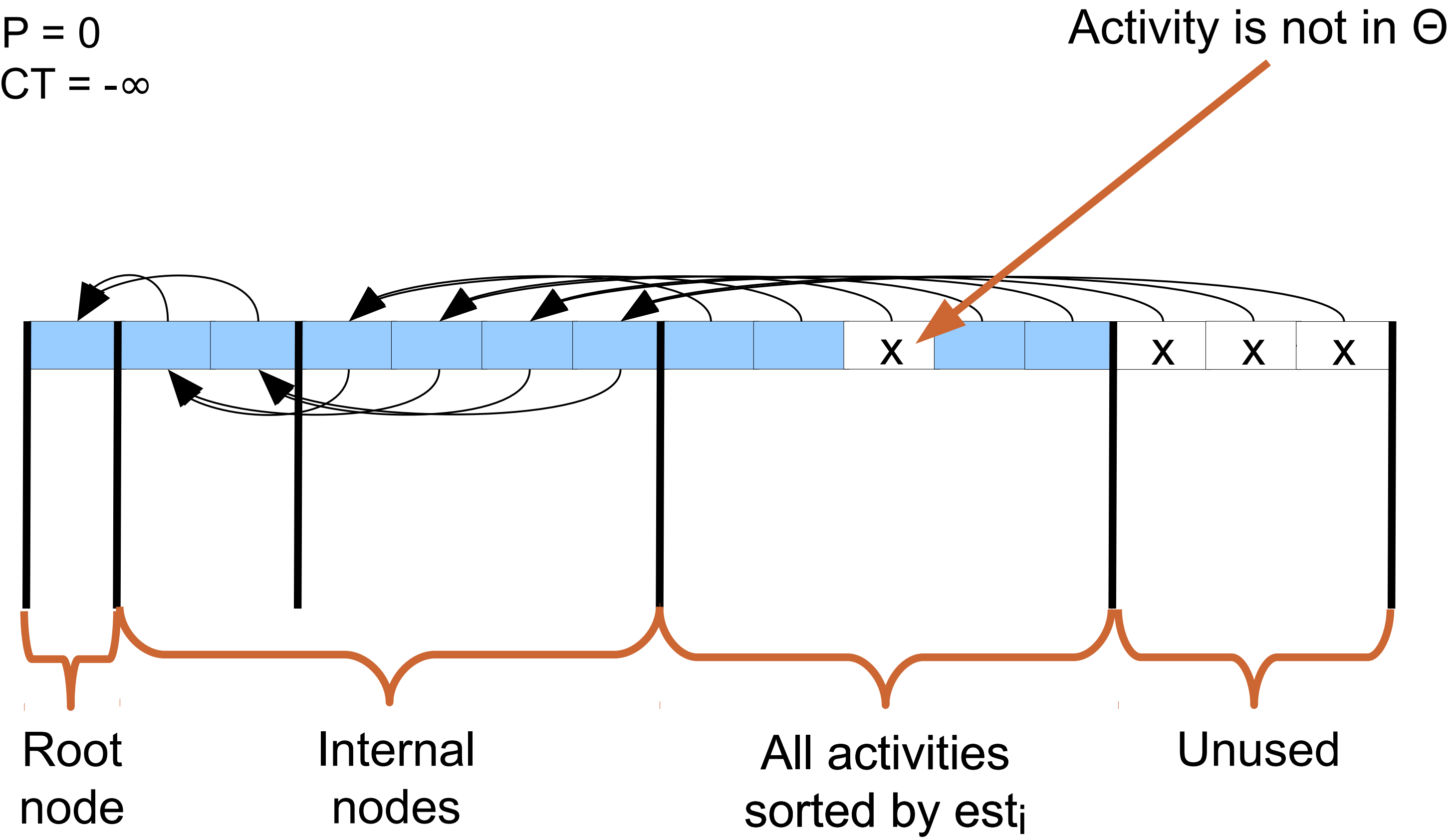
Overload checking algorithm

```
$\Theta := \emptyset$;
for $j \in T$ in non-decreasing order of $\mathrm{lct}_j$ do begin
    $\Theta := \Theta \cup \{j\}$;
    if $\mathrm{ECT}_\Theta > \mathrm{lct}_j$ then
        fail;   {No solution exists}
end;
```

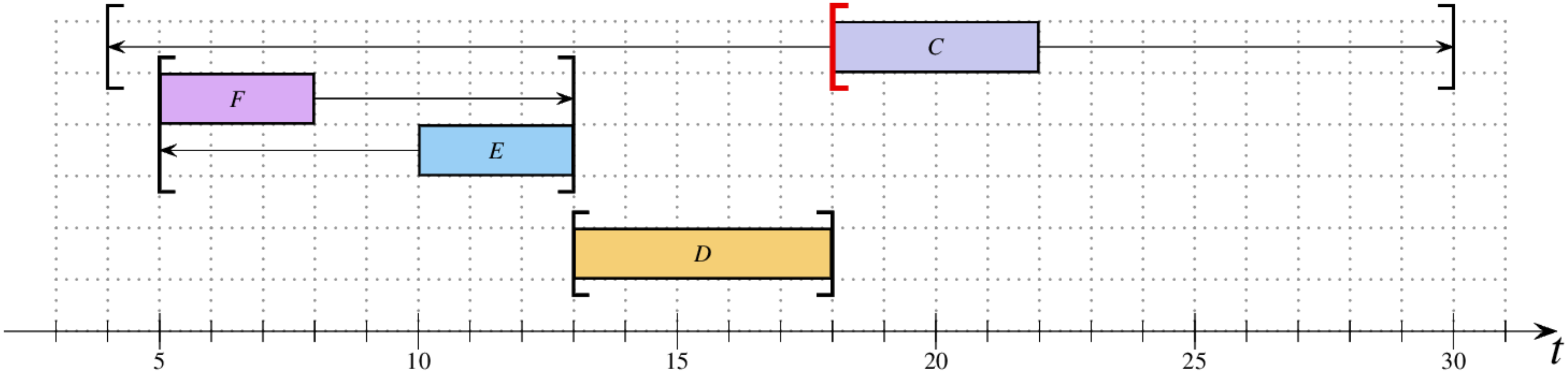
Time complexity is O(n log n).

Example of implementation of Θ-Tree

- Tree is stored in an array (similar to array representation of a heap).
- Tree doesn't change its shape. Instead of node addition/removal nodes are turned on/off.
- Node turned off:
  - $\sum P = 0$
  - $ECT = -\infty$

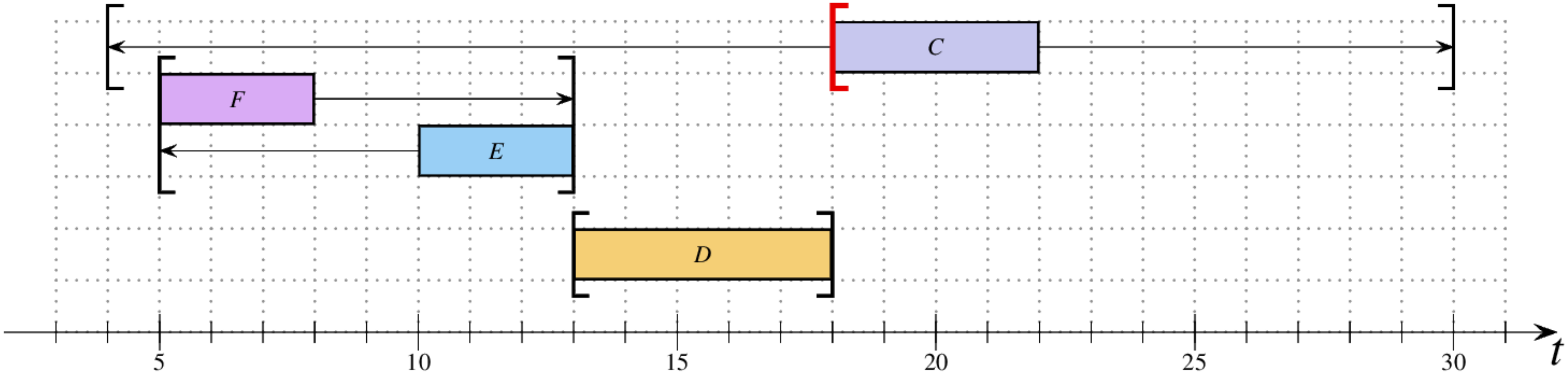


Edge Finding



- Edge finding improve bounds by removing values that would lead to overflow.
- Scheduling activity C before 18 would lead to overflow.
  - $est_C := 18$

Edge Finding



- Remember the overflow rule:

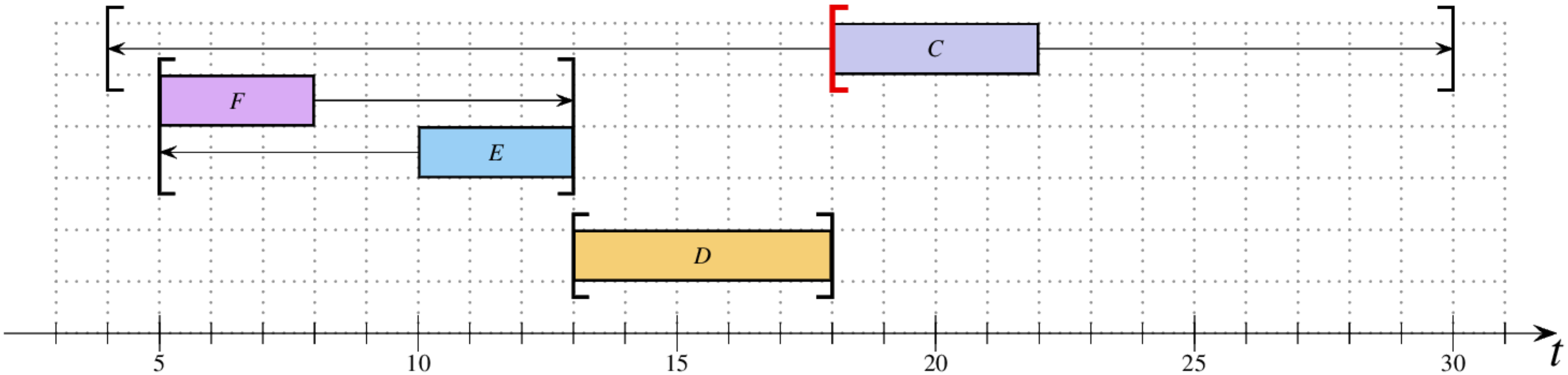
$$\mathrm{ECT}_\Theta > \mathrm{lct}_\Theta \quad \Rightarrow \quad \text{fail}$$

- Edge finding rule is:

$$\mathrm{ECT}_{\Theta \cup \{i\}} > \mathrm{lct}_\Theta \quad \Rightarrow \quad \Theta \ll i \quad \Rightarrow \quad \mathrm{est}_i := \max\{\mathrm{est}_i,\ \mathrm{ECT}_\Theta\}$$

- Setting lct_Θ as deadline for activity i would cause overflow.
  - Therefore i can start only after all activities from Θ finish.

Edge Finding: idea of the algorithm



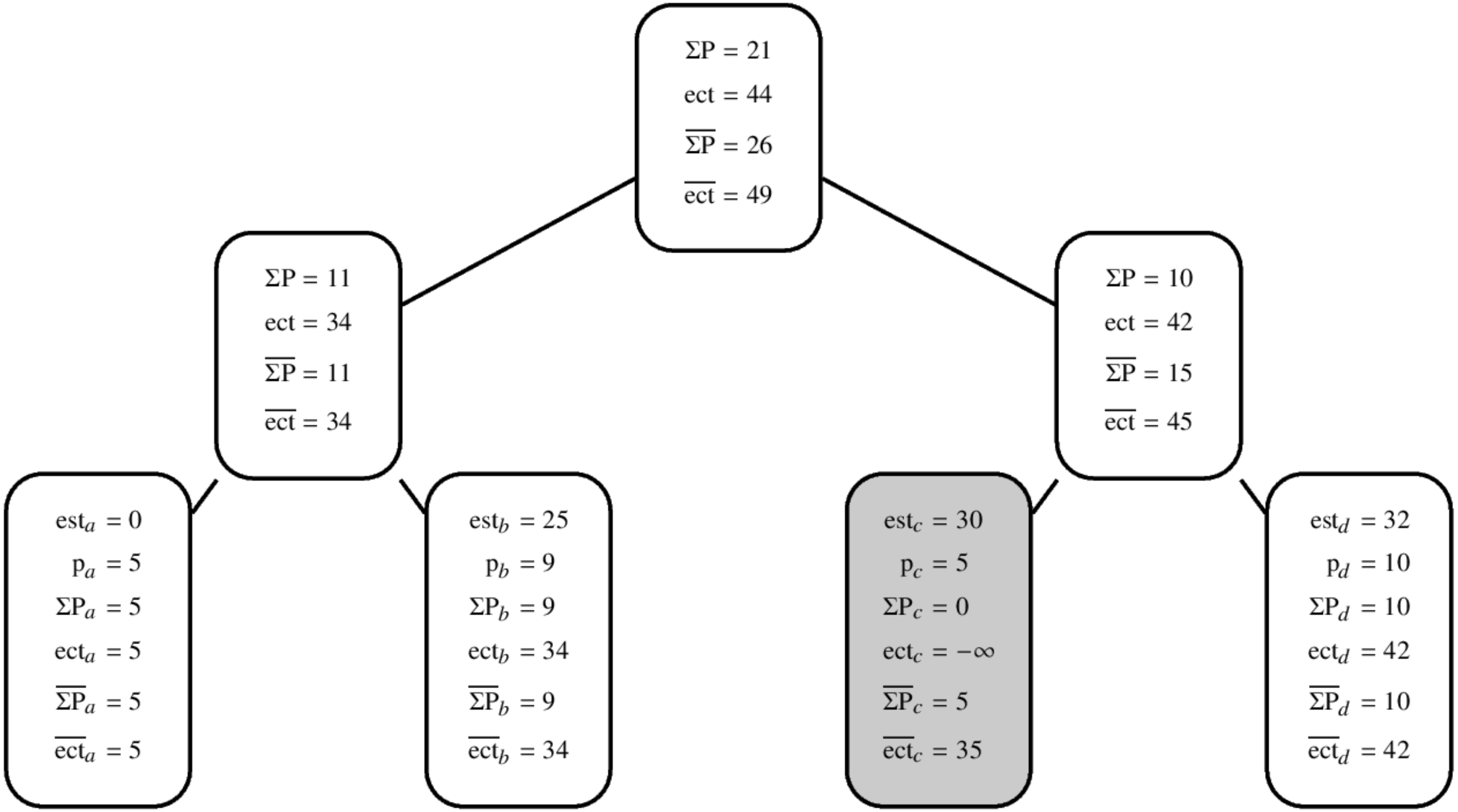
- Consider some deadline t.
- Θ = all activities that must finish before t.
- Λ = all activities that can start before t but can finish after t.
- If we can add one activity from Λ into Θ, how big earliest completion time we can make?
- Is it bigger than t?
- If yes, activity we used from Λ can be updated and removed from Λ.

- for example t = lct_D = 18
- Θ = {D, E, F}
- Λ = {C}
- $\mathrm{ECT}_{\{C,D,E,F\}}$ = 19
- Yes: 19 > 18
- est_C := 18

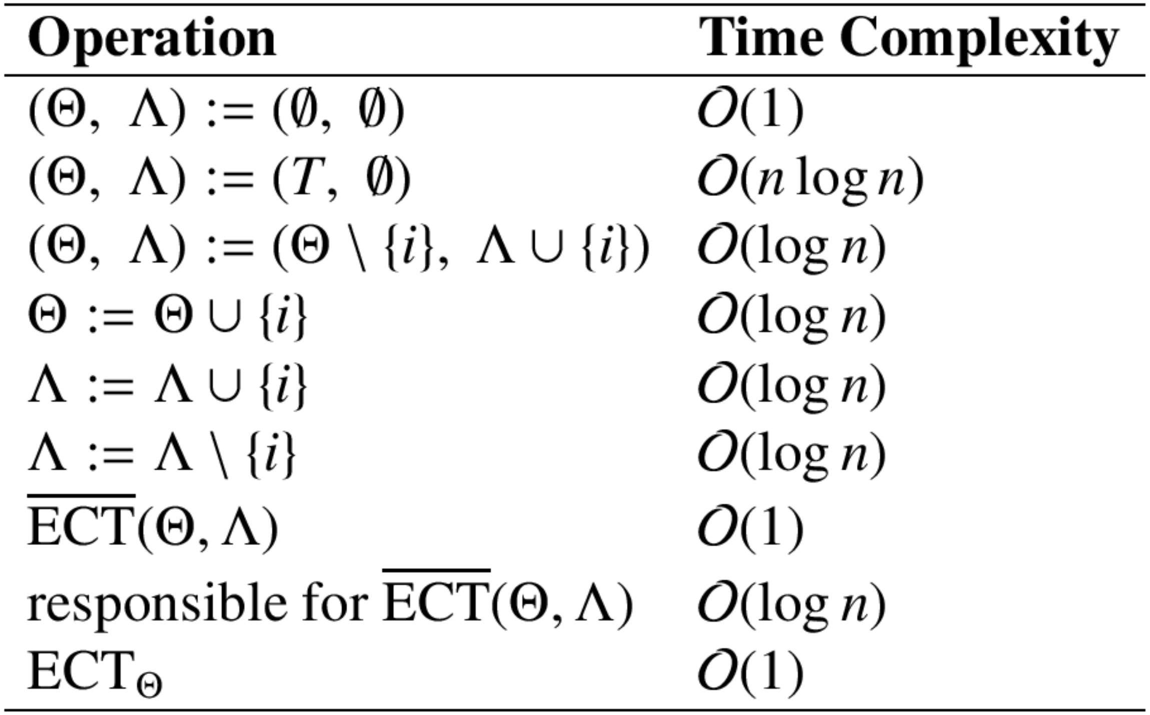
Θ-Λ-Tree

The concept of Θ-tree is extended to compute:

$$\overline{\mathrm{ECT}}(\Theta, \Lambda) = \max\left(\{\mathrm{ECT}_{\Theta}\} \cup \{\mathrm{ECT}_{\Theta \cup \{i\}},\ i \in \Lambda\}\right)$$



Θ-Λ-Tree: time complexities



Operation	**Time Complexity**
$(\Theta,\ \Lambda) := (\emptyset,\ \emptyset)$	$\mathcal{O}(1)$
$(\Theta,\ \Lambda) := (T,\ \emptyset)$	$\mathcal{O}(n \log n)$
$(\Theta,\ \Lambda) := (\Theta \setminus \{i\},\ \Lambda \cup \{i\})$	$\mathcal{O}(\log n)$
$\Theta := \Theta \cup \{i\}$	$\mathcal{O}(\log n)$
$\Lambda := \Lambda \cup \{i\}$	$\mathcal{O}(\log n)$
$\Lambda := \Lambda \setminus \{i\}$	$\mathcal{O}(\log n)$
$\overline{\mathrm{ECT}}(\Theta, \Lambda)$	$\mathcal{O}(1)$
responsible for $\overline{\mathrm{ECT}}(\Theta, \Lambda)$	$\mathcal{O}(\log n)$
ECT_Θ	$\mathcal{O}(1)$

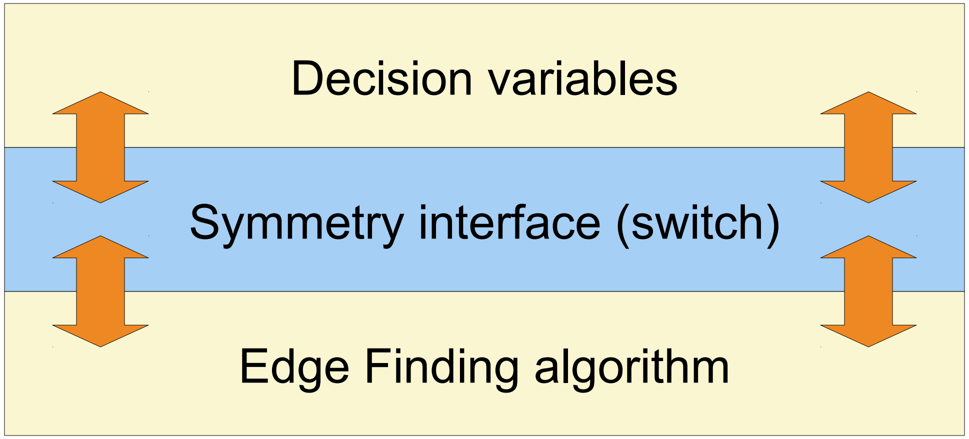
Edge Finding algorithm

```
$(\Theta, \Lambda) := (T, \emptyset)$;
Q := queue of all activities $j \in T$ in non-increasing order of $\mathrm{lct}_j$;
$j$ := Q.first;
while Q.size > 1 do begin
    if $\mathrm{ECT}_\Theta > \mathrm{lct}_j$ then
        fail; {Resource is overloaded}
    $(\Theta, \Lambda) := (\Theta \setminus \{j\}, \Lambda \cup \{j\})$;
    Q.dequeue;
    $j$ := Q.first;
    while $\overline{\mathrm{ECT}}(\Theta, \Lambda) > \mathrm{lct}_j$ do begin
        $i$ := gray activity responsible for $\overline{\mathrm{ECT}}(\Theta, \Lambda)$;
        $\mathrm{est}_i := \max\{\mathrm{est}_i, \mathrm{ECT}_\Theta\}$;
        $\Lambda := \Lambda \setminus \{i\}$;
    end;
end;
```

Time complexity is O(n log n).

Symmetry

- Just presented algorithm updates only est_i, not lct_i.
- Algorithm to update lct_i is symmetrical.
- There are two ways to implement it:
  - Write the algorithm twice ("forward" and "backward" versions).
  - Write the algorithm only once but feed it with symmetrical data.

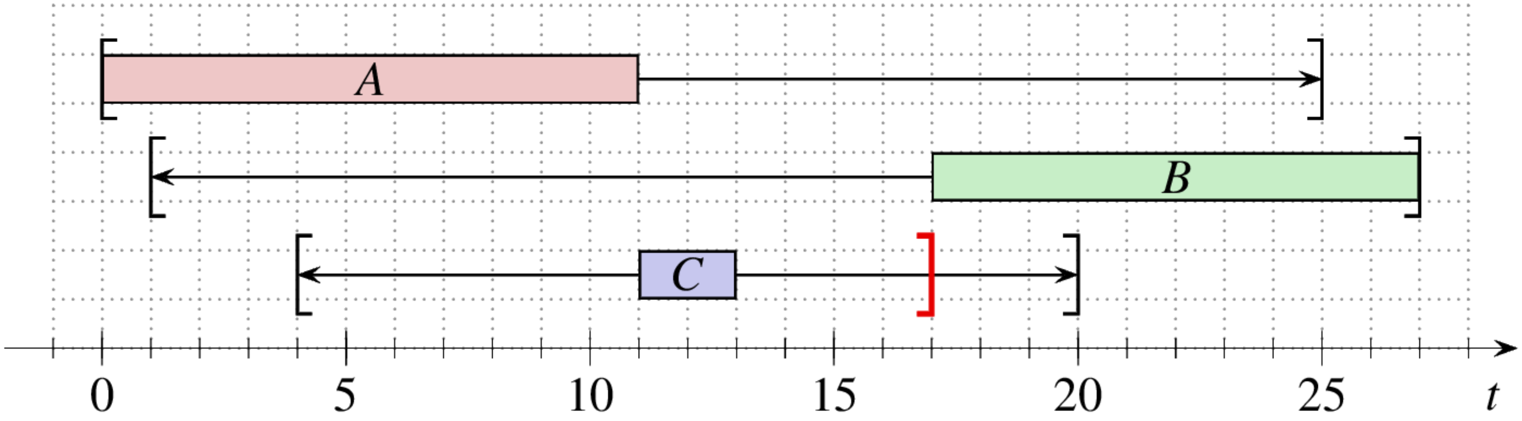


Decision variables

Symmetry interface (switch)

Edge Finding algorithm

Not-First / Not-Last



- Let Θ = {A, B}.
- $ECT_\Theta = ect_A + p_A + p_B = 0 + 11 + 10 = 21$
- If Θ is scheduled before C then Θ would have to end before $lct_C - p_C = 20 - 2 = 18$
  - This is not possible because 21 > 18
- At least one activity from Θ must be after C.
- $lct_C \le \max(lct_A - p_A, lct_B - p_B) = 17$

Propagation rule:

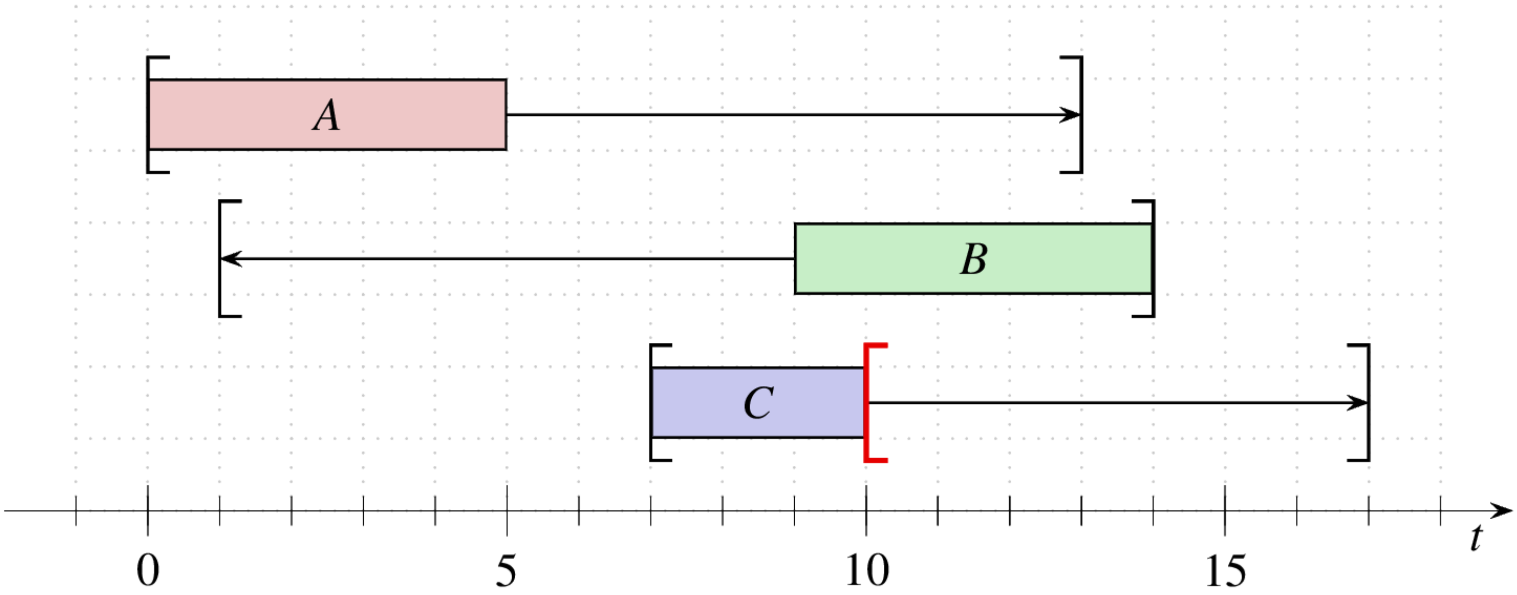
$$\mathrm{ECT}_\Theta > \mathrm{lct}_i - \mathrm{p}_i \quad \Rightarrow \quad i \text{ can't be last} \quad \Rightarrow \quad \mathrm{lct}_i := \max\left\{\mathrm{lct}_j - \mathrm{p}_j,\ j \in \Theta\right\}$$

Not-Last algorithm

```
for $i \in T$ do
    $\mathrm{lct}'_i := \mathrm{lct}_i$;
$\Theta := \emptyset$;
Q := queue of all activities $j \in T$ in non-decreasing order of $\mathrm{lct}_j - \mathrm{p}_j$;
for $i \in T$ in non-decreasing order of $\mathrm{lct}_i$ do begin
    while $\mathrm{lct}_i > \mathrm{lct}_{\mathrm{Q.first}} - \mathrm{p}_{\mathrm{Q.first}}$ do begin
        $j$ := Q.first;
        $\Theta := \Theta \cup \{j\}$;
        Q.dequeue;
    end;
    if $\mathrm{ECT}_{\Theta \setminus \{i\}} > \mathrm{lct}_i - \mathrm{p}_i$ then
        $\mathrm{lct}'_i := \min\left\{\mathrm{lct}_j - \mathrm{p}_j,\ \mathrm{lct}'_i\right\}$;
end;
for $i \in T$ do
    $\mathrm{lct}_i := \mathrm{lct}'_i$;
```

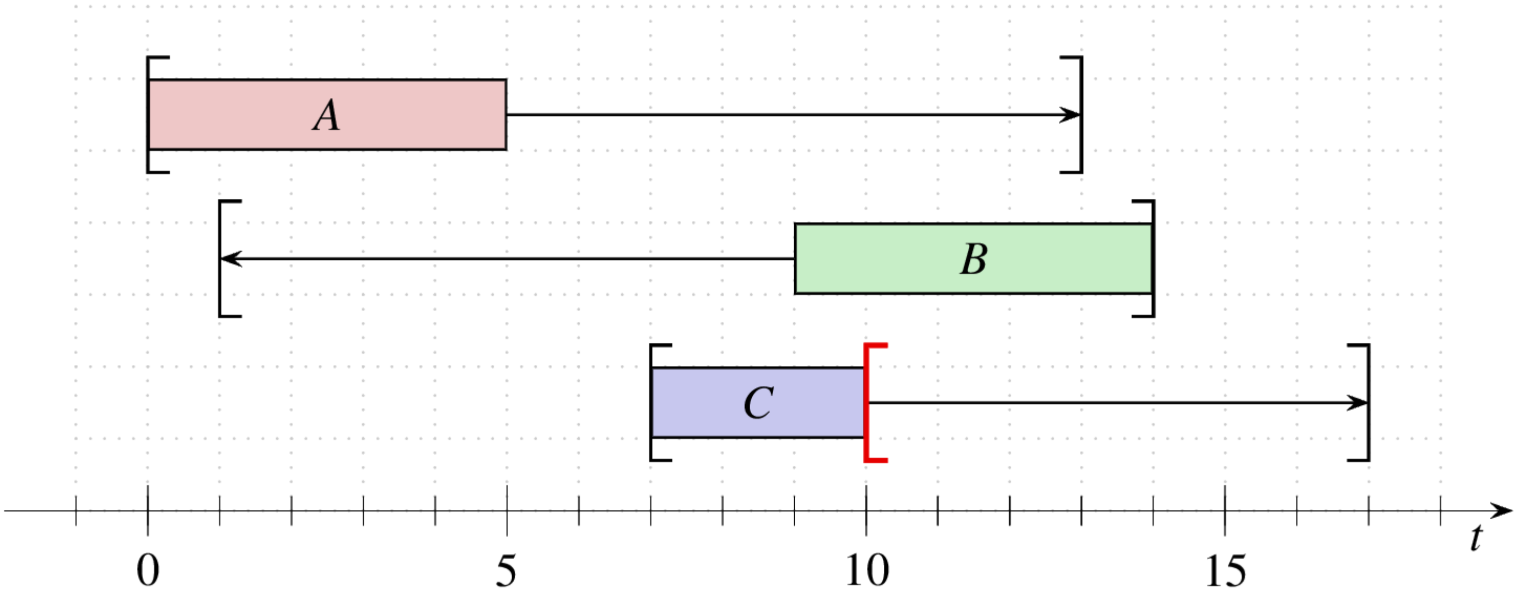
Time complexity is O(n log n).

Detectable precedences



- C doesn't fit before B. Therefore B is before C: B«C
- Similarly, C doesn't fit before A. Therefore A is before C: A«C
- {A, B}«C therefore C cannot start before $\text{ECT}_{\{A,B\}}$ = 10.

Detectable precedences



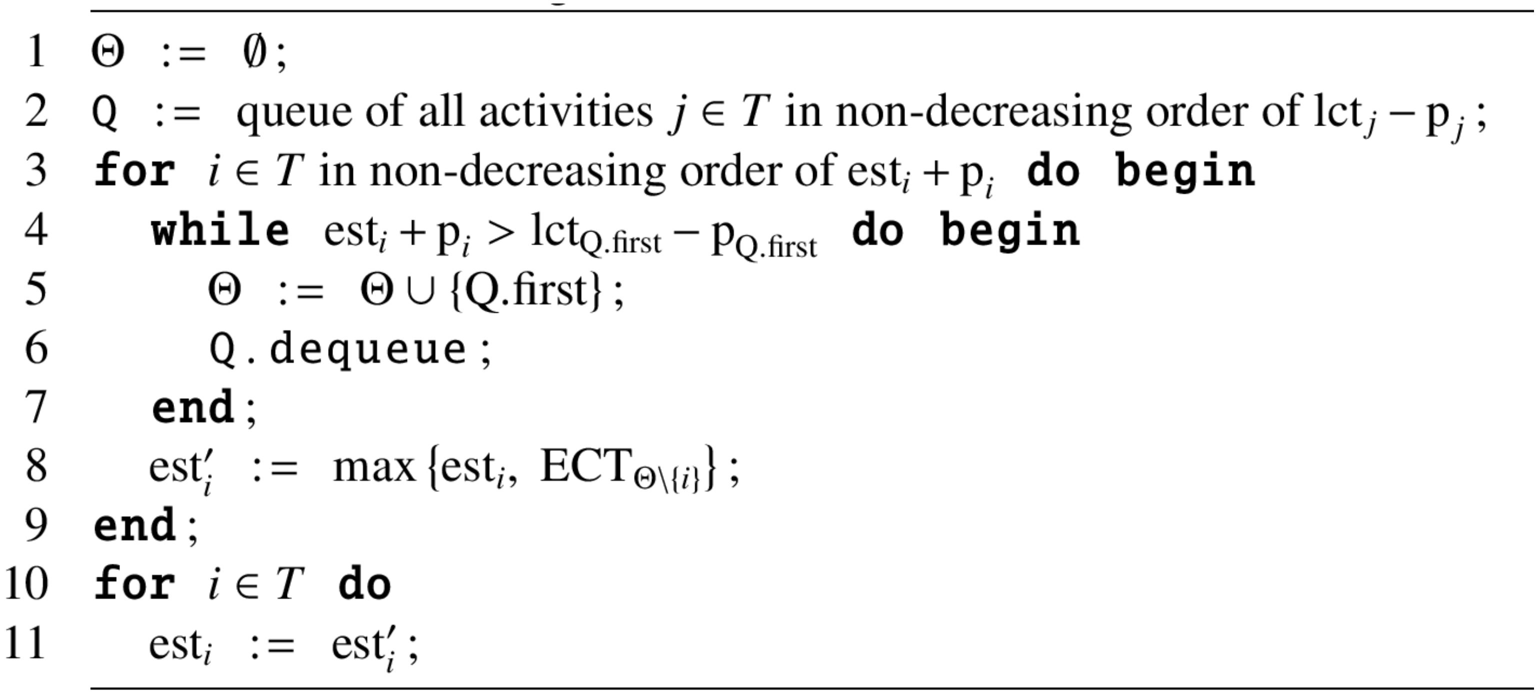
- Detectable precedence:

$$\mathrm{est}_i + \mathrm{p}_i > \mathrm{lct}_j - \mathrm{p}_j \quad \Rightarrow \quad j \ll i$$

The algorithm:

- Take an activity i
- Let Θ are detectable predecessors of i: Θ = {j, j«i}.
- Then i cannot start before ECT_{Θ}.

Detectable Precedences algorithm



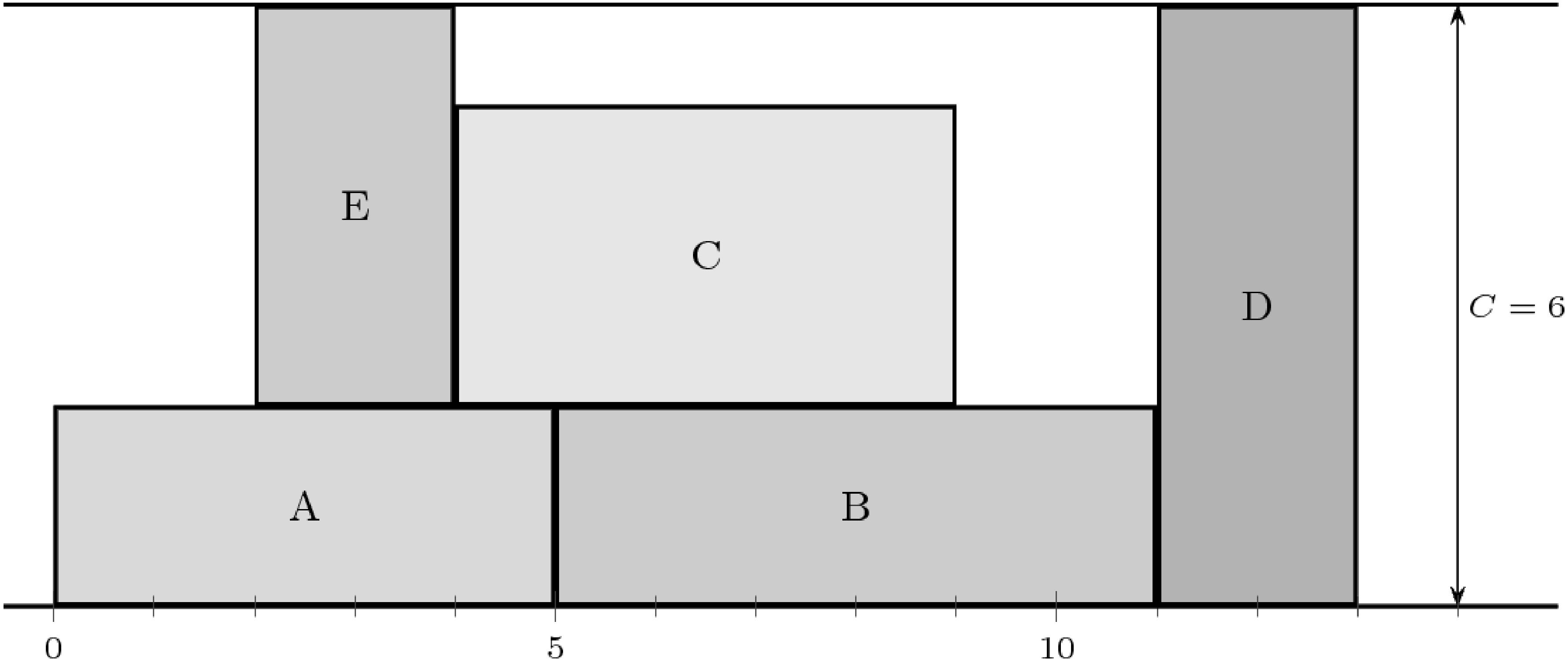
```
$\Theta := \emptyset$;
Q := queue of all activities $j \in T$ in non-decreasing order of $\mathrm{lct}_j - \mathrm{p}_j$;
for $i \in T$ in non-decreasing order of $\mathrm{est}_i + \mathrm{p}_i$ do begin
    while $\mathrm{est}_i + \mathrm{p}_i > \mathrm{lct}_{\mathrm{Q.first}} - \mathrm{p}_{\mathrm{Q.first}}$ do begin
        $\Theta := \Theta \cup \{\mathrm{Q.first}\}$;
        Q.dequeue;
    end;
    $\mathrm{est}'_i := \max\{\mathrm{est}_i, \mathrm{ECT}_{\Theta \setminus \{i\}}\}$;
end;
for $i \in T$ do
    $\mathrm{est}_i := \mathrm{est}'_i$;
```

Time complexity is O(n log n).

Cumulative Resources
Timetable Edge Finding

[1] Vilím: Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources, CPAIOR 2011

Cumulative Resource



Filtering Algorithms for Cumulative Resource

Classical Filtering Algorithms:

- Timetable propagation
- Edge Finding:
  - $O(kn^2)$
  - $O(kn \log n)$
- Extended Edge Finding
  - $O(kn^2)$
- Not-First / Not-Last
  - $O(n^2 \log n)$, lazy
- Energetic Reasoning
  - $O(n^3)$

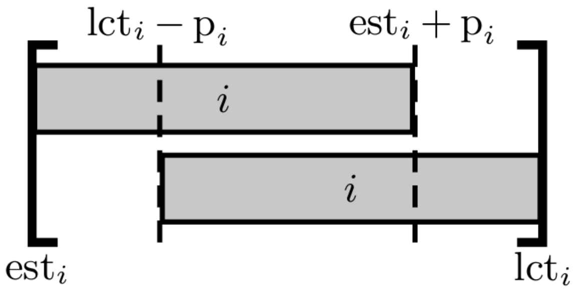
These algorithms are independent and could/should be used together.

Timetable Edge Finding:

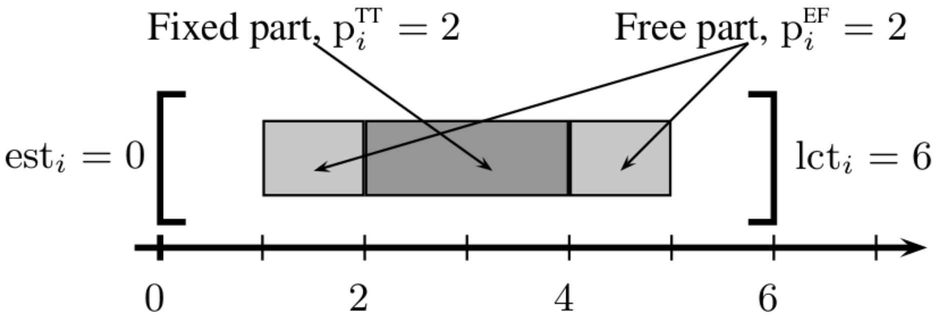
- Inspired by **all** the algorithms on the left.
- Meant to be used together with timetable propagation.
- Reuses/shares data structure with timetable propagation.
- Stronger propagation than both Edge Finding and Extended Edge Finding.
- Limited Not-First / Not-Last and Energetic Reasoning.
- $O(n^2)$
- Lazy propagation: may need more iterations to reach fixpoint.

Timetable Propagation

- If for activity i holds $lct_i - p_i < est_i + p_i$ then the activity necessarily use the resource during interval $[lct_i - p_i, est_i + p_i]$.
- In this case we split the interval into fixed and free parts:

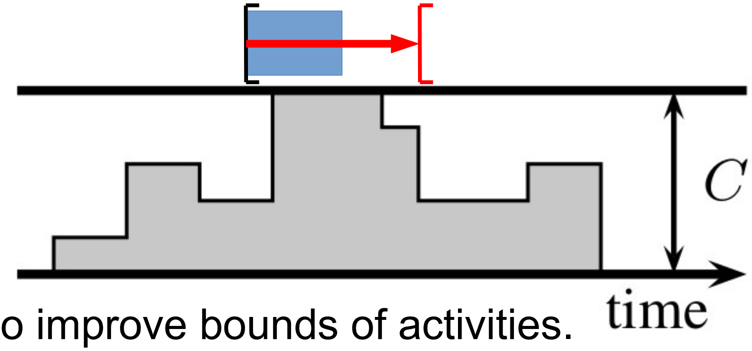


$\mathrm{lct}_i - \mathrm{p}_i$
$\mathrm{est}_i + \mathrm{p}_i$
i
i
est_i
lct_i



Fixed part, $\mathrm{p}_i^{\mathrm{TT}} = 2$
Free part, $\mathrm{p}_i^{\mathrm{EF}} = 2$
$\mathrm{est}_i = 0$
$\mathrm{lct}_i = 6$
0
2
4
6

- Fixed parts are aggregated into timetable (graph of minimum resource usage):

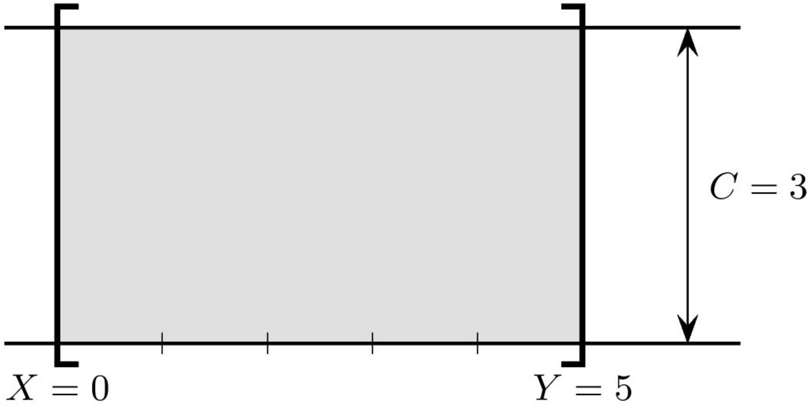


- The timetable is used to improve bounds of activities.

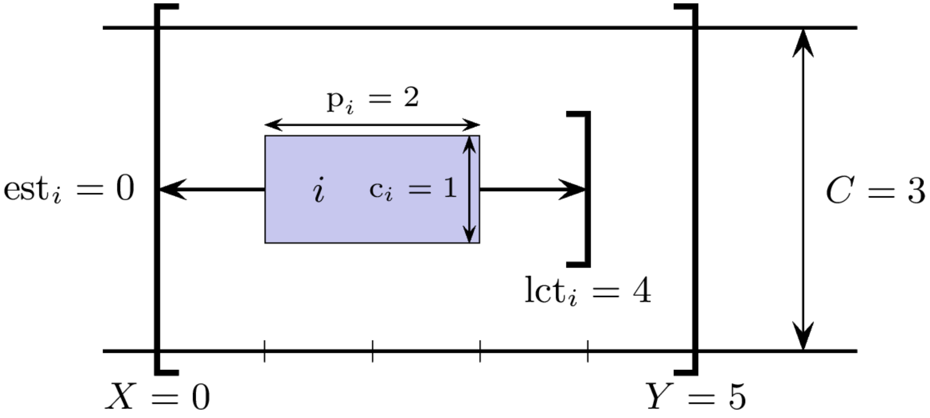
Overload Checking

- Similar to disjunctive case. O(n^2) and O(n log n) versions.
- It is the cornerstone of all Edge Finding algorithms.
- The idea is to chose an interval [X, Y] and compare:

Available energy (area) in interval [X, Y]:



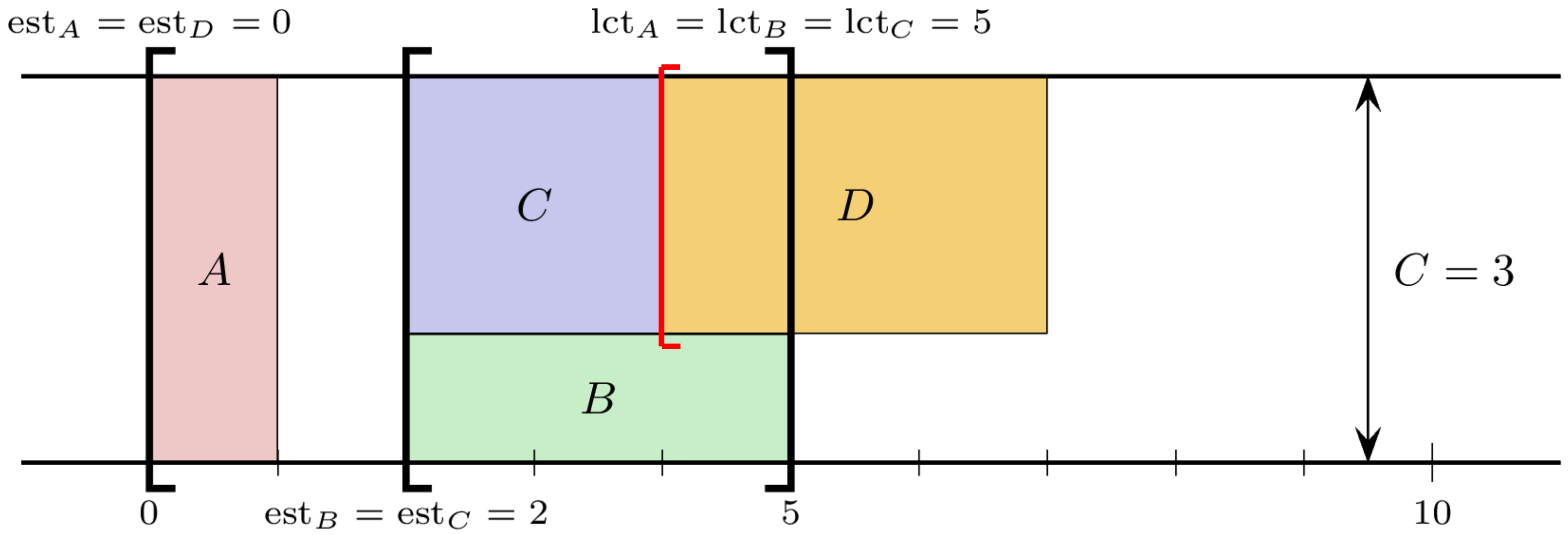
Total energy of activities which must be *completely inside* [X, Y]:



$$C(Y - X) < \sum_{\substack{\mathrm{est}_i \geq X \\ \mathrm{lct}_i \leq Y}} \mathrm{c}_i\, \mathrm{p}_i \quad \Rightarrow \quad \text{fail}$$

Standard and Extended Edge Finding Algorithms

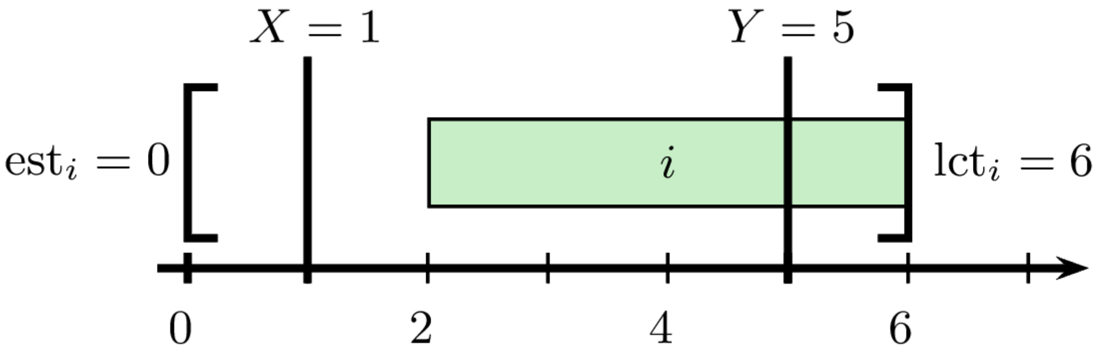
Informally speaking, these algorithms update time windows in such a way that scheduling any activity i on its earliest starting time est_i does not lead to immediate overload.


$\text{est}_A = \text{est}_D = 0$
$\text{lct}_A = \text{lct}_B = \text{lct}_C = 5$
C
D
A
B
$C = 3$
0
$\text{est}_B = \text{est}_C = 2$
5
10

- In this example, est_D can be updated from 0 to 4.
- Otherwise, either interval [0, 5] or [2, 5] would be overloaded.

Energetic Reasoning Algorithm

- Energy computation in Edge Finding takes into account only activities which are *completely inside* the interval [X,Y].
- Therefore it misses cases when only a part of the activity must be executed inside [X, Y]. For example, activity i in the following picture consumes at least 3 energy units during [1, 5]:



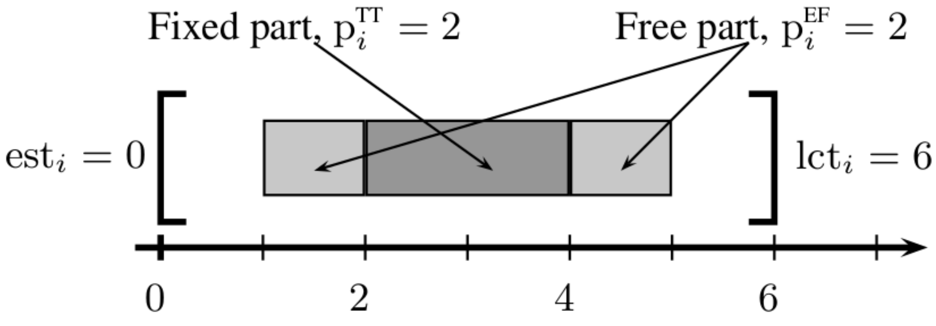
- There is Energetic Reasoning algorithm, which takes this energy into account, but it is $O(n^3)$.
- However there are some simple cases where we can improve energy computation without increasing time complexity.
- In particular, the idea is to take into account timetable.

Timetable Edge Finding

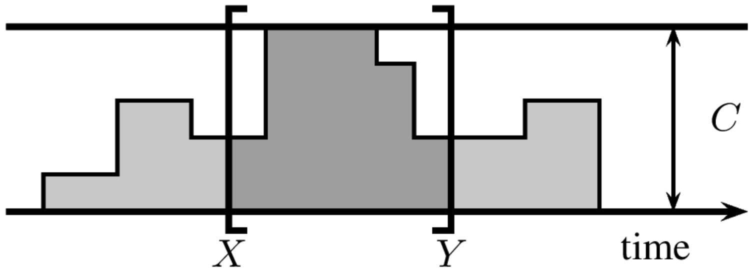
The idea is to split energy computation during [X, Y] into two parts:

energy from fixed parts

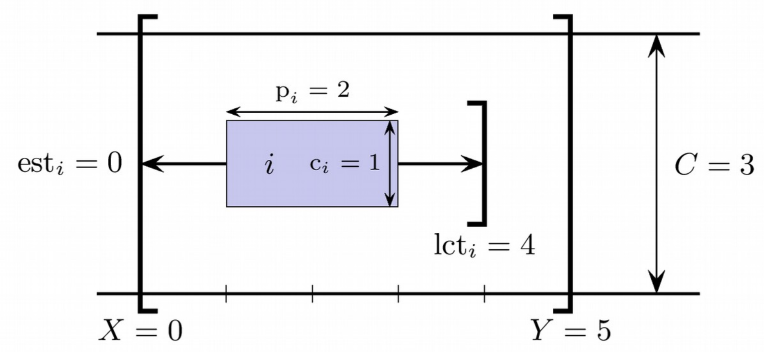
energy from from free parts



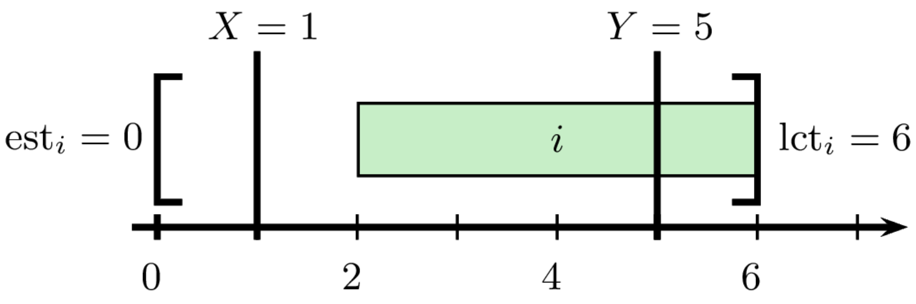
This energy can be easily computed from timetable:



Computed by standard Edge Finding way, but only from free parts:



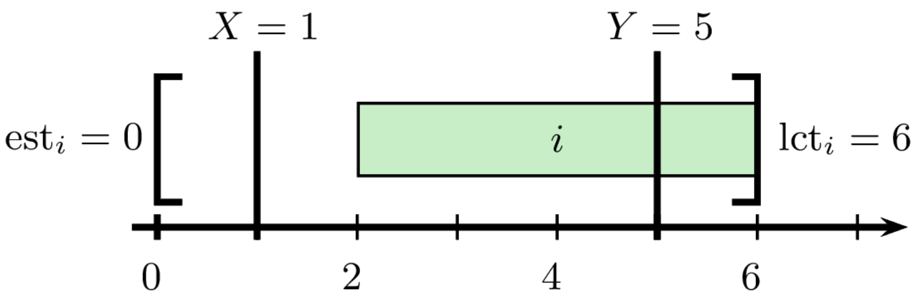
Example of energy computation



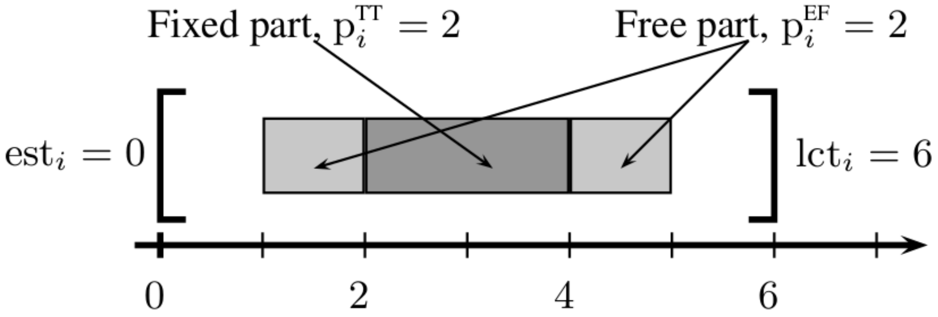
What is the minimal energy contribution of activity i to interval [1, 5]?

- Energetic reasoning: 3
  - Exact computation, but slow.
- Edge Finding: 0
  - Activity i is not *completely inside* [1, 5] therefore it is ignored.
- Timetable Edge Finding: 2 (from fixed part)
  - Fast, but not exact.

Example of energy computation



Timetable Edge Finding splits activity i into two fixed part (duration 2) and free part (also duration 2):


Fixed part, $\text{p}_i^{\text{TT}} = 2$
Free part, $\text{p}_i^{\text{EF}} = 2$
$\text{est}_i = 0$
$\text{lct}_i = 6$
0
2
4
6

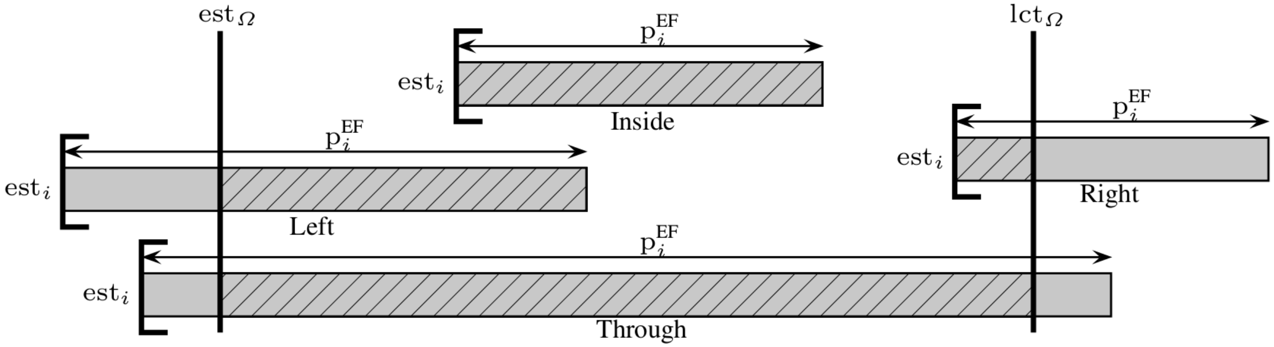
For interval [1,5], TTEF takes fixed part into account, but ignores free part (because it is not *completely inside* [1,5]). Total contribution counted is 2 energy units.

Note that for fixed activities, TTEF computes the same value as Energetic Reasoning.

Timetable Edge Finding algorithm

```
    $i \in T^{EF}$
  $est'_i := est_i$ ;
    $b \in T^{EF}$
  // Cases "Inside" and "Right"
  $eEF := 0$;
  $\iota := -1$;
        $a \in T^{EF}$ such that $est_a < lct_b$, in non-increasing order by $est_a$
          $lct_a \le lct_b$
        $eEF := eEF + e^{EF}_a$ ;
              $\iota = -1$        $\min(e^{EF}_a, c_a(lct_b - est_a)) > \min(e^{EF}_\iota, c_\iota(lct_b - est_\iota))$
             $\iota := a$;
      $reserve := C(lct_b - est_a) - eEF - (\texttt{ttAfterEst}[a] - \texttt{ttAfterLct}[b])$;
          $\iota \ne -1$        $reserve < \min(e^{EF}_\iota, c_\iota(lct_b - est_\iota))$
        $est'_\iota := \max(est'_\iota, lct_b - mandatoryIn(est_a, lct_b, \iota) - \lfloor reserve / c_\iota \rfloor)$ ;
        ;
  // Case "Through"
  $\iota := -1$;
        $a \in T^{EF}$ in non-decreasi
        break ties by non-increa

          $lct_a \le lct_b$
        $reserve := C(lct_b - est$
              $\iota \ne -1$        reserv
            $est'_\iota := \max(est'_\iota$, lc
        $eEF := eEF - e^{EF}_a$ ;
          ;
          $est_a + p^{EF}_a \ge lct_b$
        $\iota := a$;
        ;
      ;
// Case "Left"
      $a \in T^{EF}$
  $eEF := 0$;
  $\iota := -1$;
  $Q := $ queue of activities $i \in T^{EF}$ sorted by non-decreasing $est_i + p^{EF}_i$ ;
        $b \in T^{EF}$ in non-decreasing order by $est_b$
          $est_a \le est_b$
        $eEF := eEF + e^{EF}_b$ ;
              $est_{Q.top} + p^{EF}_{Q.top} < lct_b$
          $i := Q.dequeue$;
             $est_i < est_a$        $est_a < est_i + p^{EF}_i$
             ( $\iota = -1$        $c_i(est_i + p^{EF}_i - est_a) > c_\iota(est_\iota + p^{EF}_\iota - est_a)$ )
                $\iota := i$;
          ;
        $reserve := C(lct_b - est_a) - eEF - (\texttt{ttAfterEst}[a] - \texttt{ttAfterLct}[b])$;
            $\iota \ne -1$        $reserve < c_\iota(est_\iota + p^{EF}_\iota - est_a)$
            $est'_\iota := \max(est'_\iota, lct_b - mandatoryIn(est_a, lct_b, \iota) - \lfloor reserve / c_\iota \rfloor)$;
          ;
      ;
    $i \in T^{EF}$
  $est_i := est'_i$ ;
```



Time complexity is O(n^2).

Propagation with optional interval variables

[1]	Laborie, Rogerie: Reasoning with Conditional Time-intervals. FLAIRS-08.
[2]	Laborie, Rogerie: Reasoning with Conditional Time-intervals, Part II: an Algebraical Model for Resources. FLAIRS-09.

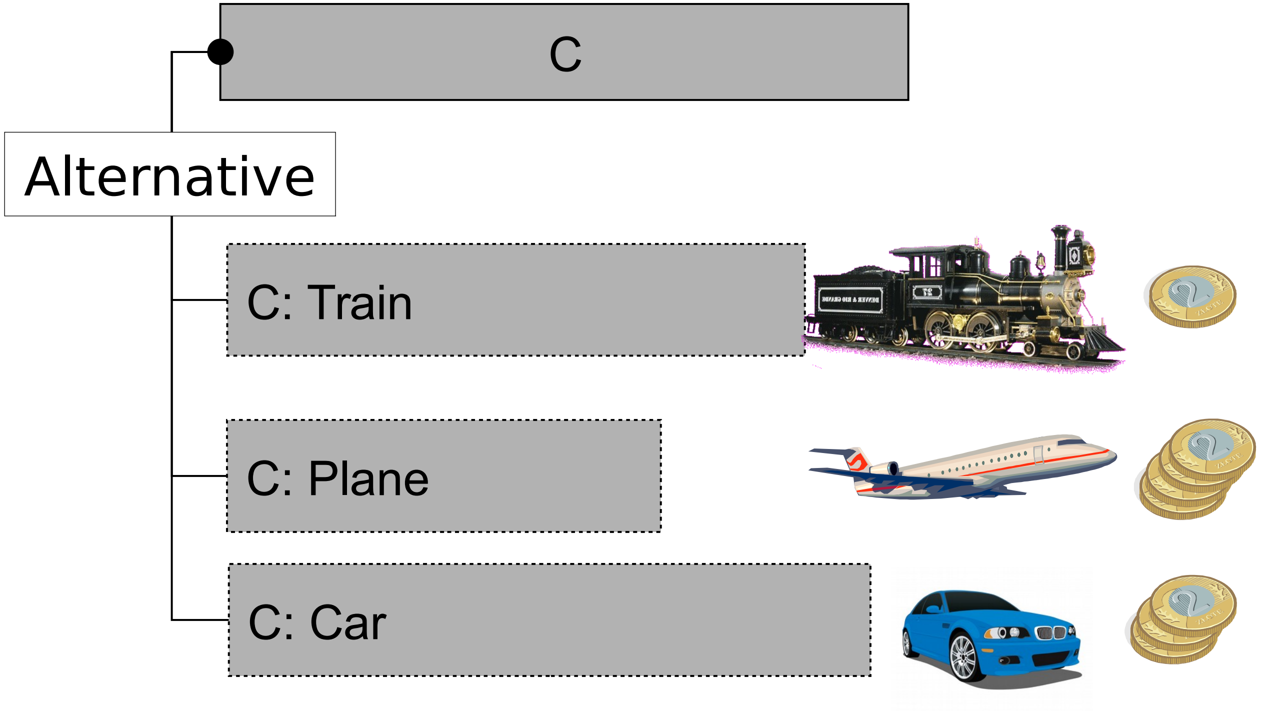
Alternatives

Let activity C represents my travel to visit a customer. I can travel by:

- train
- plane
- or car.

This decision affects:

- duration
- departure time
- cost
- resource usage



C

Alternative

C: Train

C: Plane

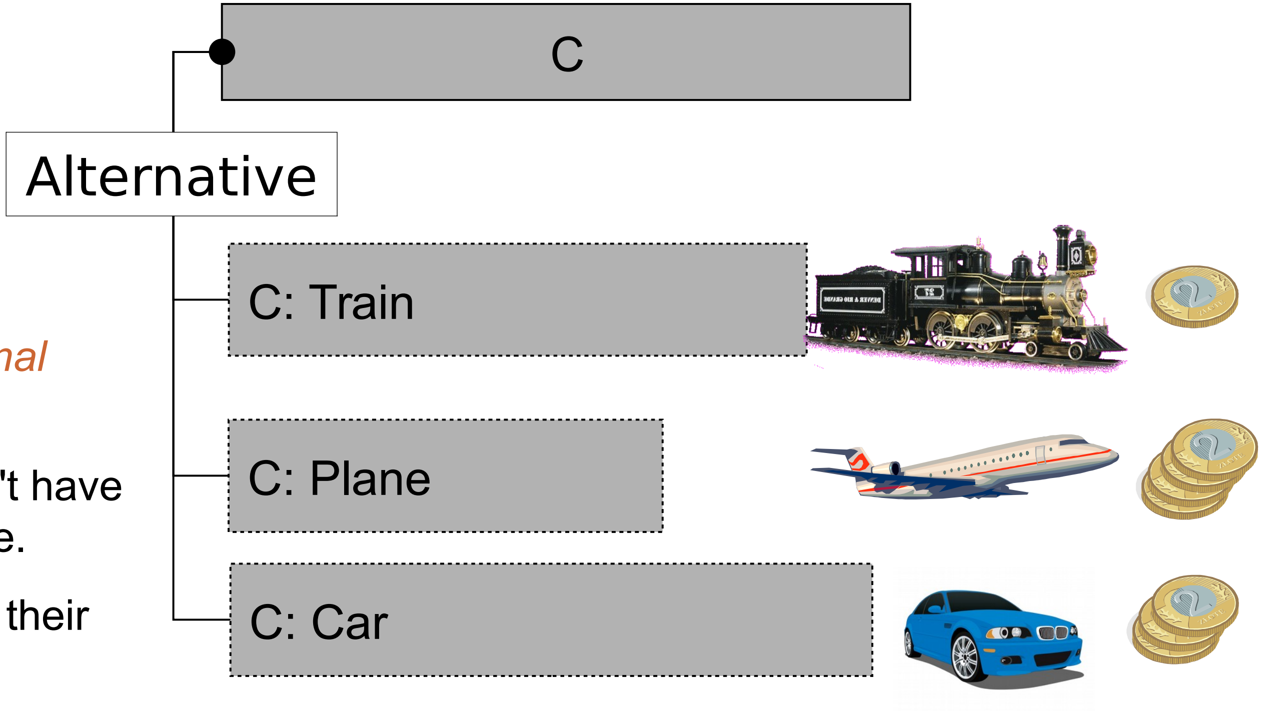
C: Car

Traditionally/historical way is to use meta-constraints to describe the problem:

- Either (train) duration = 8h and departure in {9:00, 13:40, .. } and cost = 170€
- Or (plane) duration = 3h and departure in { 9:20, 12:30, .. } and cost = 250€
- Or (car) duration = 11h and cost = 200€

Alternatives: new approach

The idea is to represent not only C as activity, but also its alternatives (modes).



C

Alternative

C: Train

C: Plane

C: Car

- C is *present* activity.
- Its alternatives are *optional* activities.
- Optional activities doesn't have to appear in the schedule.
- If they don't appear then their start is undefined.

The solver must make a decision which one of the activities C:Train, C:Plane and C:Car will be *present* in the solution. The remaining two activities will be *absent*.

Optional Interval Variable

Optional Interval Variable a:

$$\text{Domain}(a) \subseteq \{\perp\} \cup \{\, [s,e) \mid s,e \in \mathbb{Z},\ s \leq e \,\}$$

Absent interval

Interval of integers

In the model declaration, each interval variable must be either:

- present (mandatory, ⊥ is not in the domain)
- absent (domain is {⊥}).
- optional otherwise

In a solution, each interval variable must be either:

- present, then it starts at time s and ends at time e,
- or absent (⊥), and then it doesn't have any start or end.

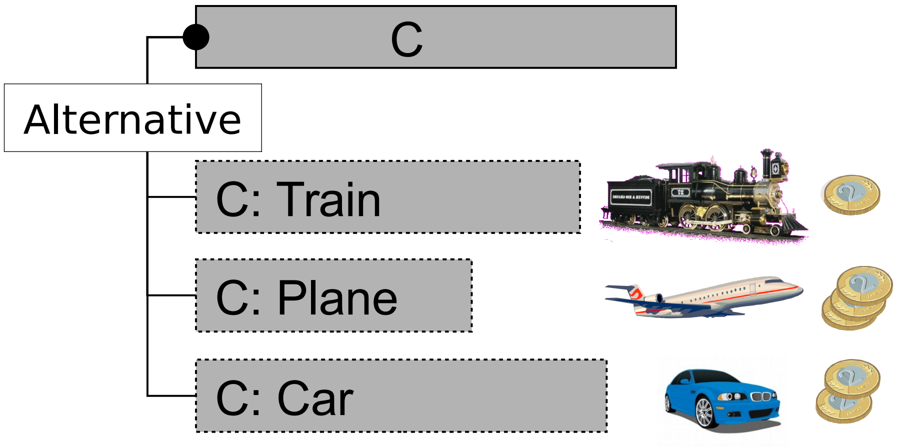
Notations: Let *a* be a **fixed** interval variable:

- If a={[s,e)} (***a* is present**) then we denote:
  - **x(a)=1** : presence status
  - **s(a)=s** : start of a
  - **e(a)=e** : end of a
- If a={⊥} (***a* is absent**), we denote:
  - **x(a)=0** (in this case, s(a) and e(a) are meaningless)

Semantics of the alternative constraint

alternative(C, {C:Train, C:Plane, C:Car})

- If C is present then:
  - Exactly one of C:Train, C:Plane, C:Car is present.
  - C and the chosen alternative start together and end together.
- If C is absent then C:Train, C:Plane and C:Car are also absent.



C

Alternative

C: Train

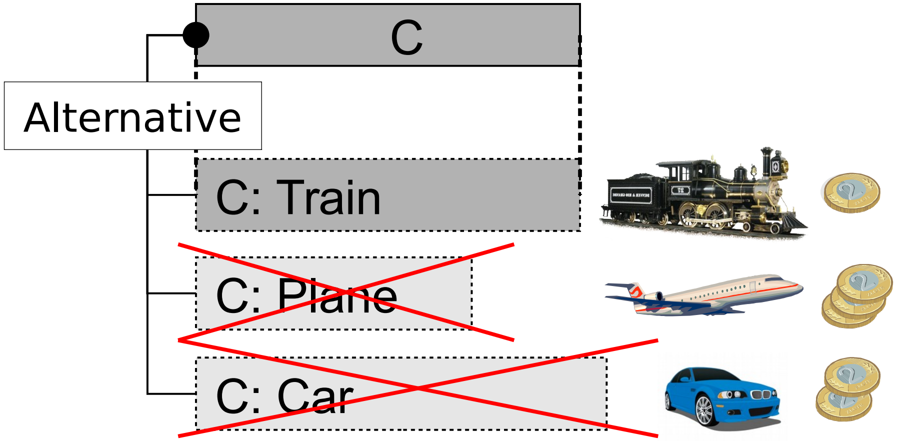
C: Plane

C: Car

Semantics of alternative constraint

alternative(C, {C:Train, C:Plane, C:Car})

- If C is present then:
  - Exactly one of C:Train, C:Plane, C:Car is present.
  - C and the chosen alternative starts together and end together.
- If C is absent then C:Train, C:Plane and C:Car are also absent.



This allows to easily constraints both on master interval C and its modes like C:Car.

After arrival, I'll check in to the hotel:

- endBeforeStart(C, HotelCheckin)

I have to be there by 14 o'clock:

- endOf(C) ≤ 14

If I use plane then I have to buy tickets at least 10 days ahead:

- presenceOf(BuyPlaneTickets) = presenceOf(C:Plane)
- endsBeforeStart(BuyPlaneTickets, C, 10)

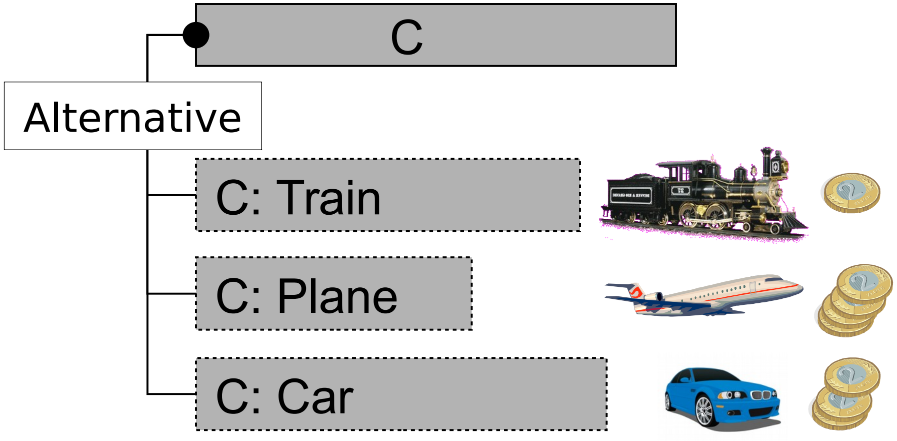
Car is a disjunctive resource that cannot be used by more than one driver at a time:

- noOverlap([C:Car, TravelOfMyWife1, TravelOfMyWife2, TravelOfMyWife3]);

Propagation of alternative constraint

alternative(C, {C:Train, C:Plane, C:Car})

- For optional activities, we maintain their time window [est_i, lct_i] for the case they will become present.
- For example:
  - – $est_{C:Train}$ = 9:00 (first train)
  - – $est_{C:Plane}$ = 9:20 (first plane)
  - – $est_{C:Car}$ = 8:00 (I refuse to get up early)

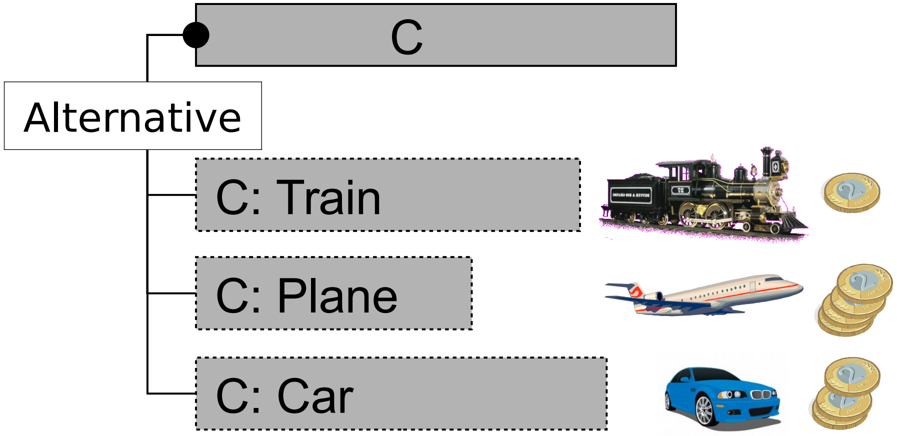


- Earliest starting time of master activity C is the minimum of available alternatives:
  - – est_C = 8:00

Propagation of alternative constraint

alternative(C, {C:Train, C:Plane, C:Car})

- For optional activities, we maintain their time window [est_i, lct_i] for the case they will become present.
- For example:
  - – $est_{C:Train}$ = 9:00 (first train)
  - – $est_{C:Plane}$ = 9:20 (first plane)
  - – $est_{C:Car}$ = 8:00 (I refuse to get up early)



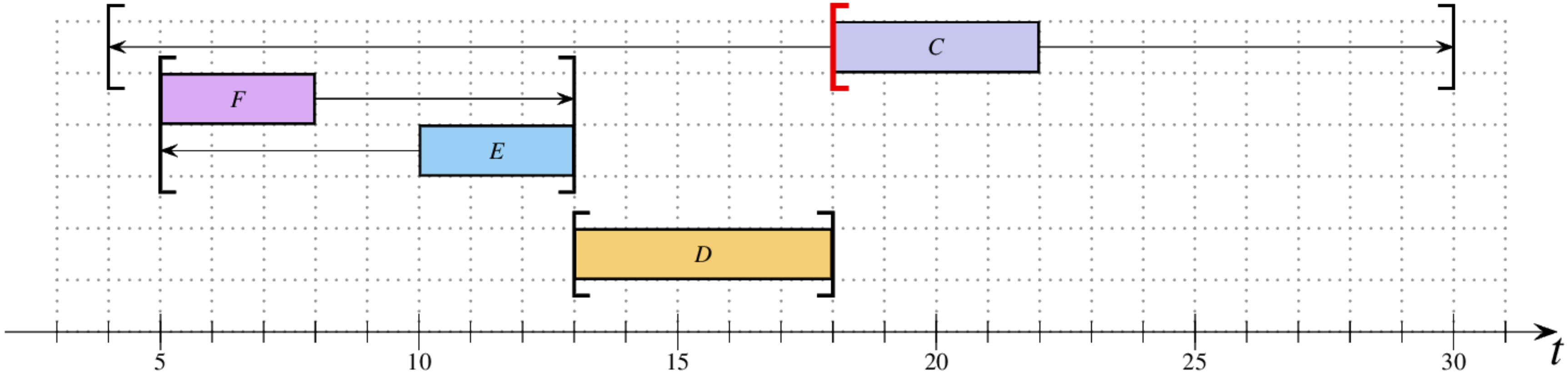
- Earliest starting time of master activity C is the minimum of available alternatives:
  - – est_C = 8:00
- My wife occupies the the car until 15:00 (present interval variable).
  - – noOverlap constraint propagates: $est_{C:Car}$ = 15.
- But that's too late (I have to be there by 14:00): $lct_{C:Car} \le lct_C = 14$.
  - – Therefore C:Car becomes *absent*.
  - – If C:Car wouldn't be optional then it would mean a fail.
- As a result, alternative constraint propagates est_C = 9:00.

How to handle optional activities in resource constraints?

The general rules are:

- Present activities influence all other activities on the resource including optional ones.
  - My wife blocked the car, C:Car was updated.
- Absent activities are ignored.
  - Once I decided not to use the car, car is not affected by my travel at all.
- Optional activities does not affect any other activity on the resource.
  - While I was only speculating about using the car, I couldn't postpone ride of my wife.

Disjunctive Edge Finding with optional activities



- Remember Edge Finding propagation rule:

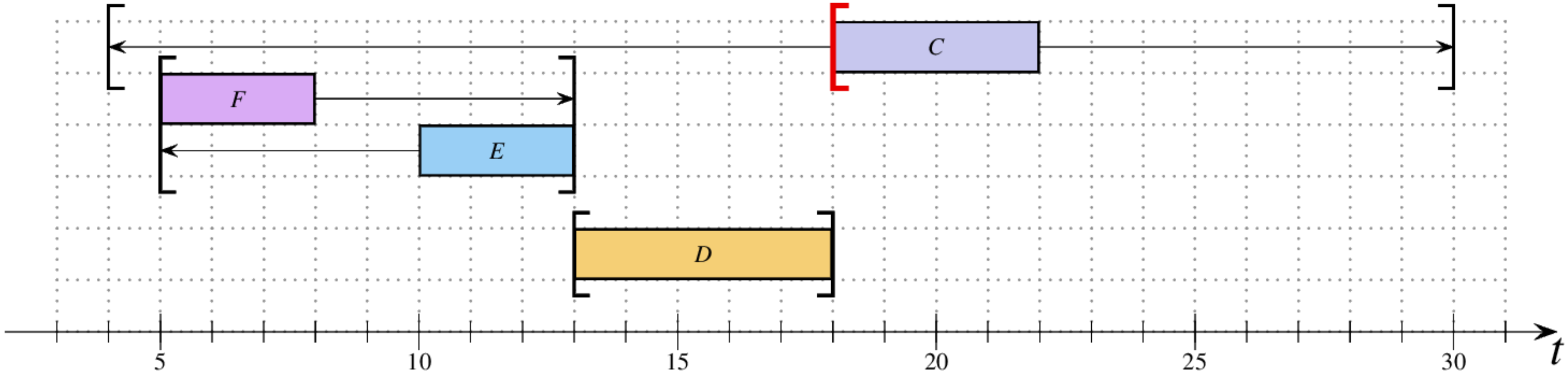
$$\mathrm{ECT}_{\Theta\cup\{i\}} > \mathrm{lct}_{\Theta} \quad\Rightarrow\quad \Theta \ll i \quad\Rightarrow\quad \mathrm{est}_i := \max\{\mathrm{est}_i,\ \mathrm{ECT}_{\Theta}\}$$

- Set Θ cannot contain any optional (or absent) interval.
  - Otherwise optional activity would affect activity i on the resource.

→ Never add optional activity into Θ.

- Note that i could be optional activity.

Disjunctive Edge Finding with optional activities



- Remember Edge Finding propagation rule:

$$\mathrm{ECT}_{\Theta\cup\{i\}} > \mathrm{lct}_\Theta \quad \Rightarrow \quad \Theta \ll i \quad \Rightarrow \quad \mathrm{est}_i := \max\{\mathrm{est}_i,\ \mathrm{ECT}_\Theta\}$$

Another approach:

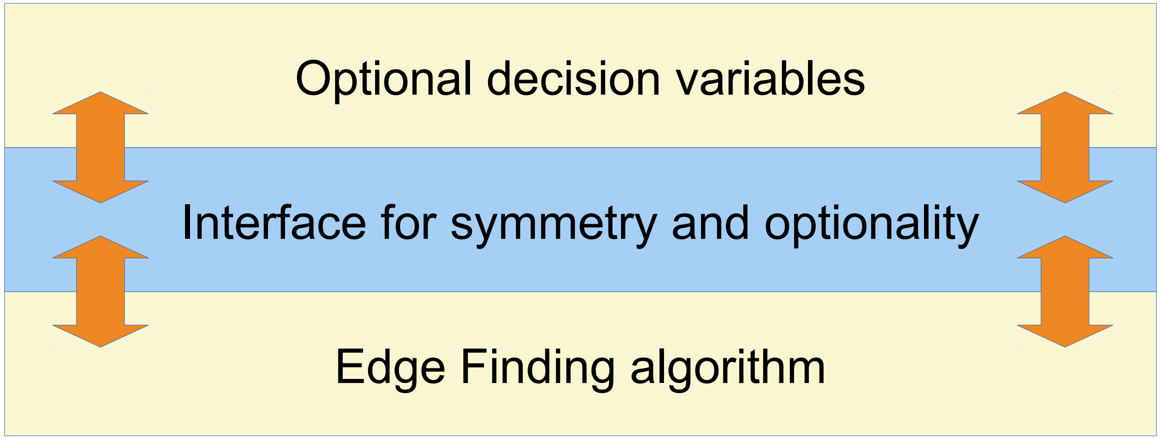
Use classical EF algorithm (unaware of optional activities) but pretend (just for the algorithm) that all optional activities have $\mathrm{lct}_i = \infty$.

– If optional activity I is in Θ then $\mathrm{lct}_\Theta = \infty$ and therefore the inequality doesn't hold.

→ It is not necessary to write new version of EF algorithm.

- It works for cumulative Edge Finding too.

Implementation of EF with optional activities



Optional decision variables

Interface for symmetry and optionality

Edge Finding algorithm

It works for Edge Finding, but not for (for example) Not-First / Not-Last.

Logical Network

[1] Laborie, Rogerie: Reasoning with Conditional Time-intervals. FLAIRS-08.
[2] Laborie, Rogerie: Reasoning with Conditional Time-intervals, Part II: an Algebraical Model for Resources. FLAIRS-09.

Logical constraints

Presence constraint presenceOf(a) means that *a* is present: x(a)=1

The constraint presenceOf(a) could be used in composed constraints (meta-constraints). For example:

- Same status: presenceOf(a) == presenceOf(b)
- Incompatibility: presenceOf(a) != presenceOf(b)
- Implication: presenceOf(a) $\leq$ presenceOf(b)
- At least 2 present: presenceOf(a) + presenceOf(b) + presenceOf(c) $\geq$ 2

Constraint Propagation: Logical network

- A **Logical network** is in charge of handling a set of binary logical constraints that can be inferred from the model:
- Those binary logical constraints are identified during presolve. For example:
  - presenceOf(a) ∨ presenceOf(b)
  - alternative(a, [b1,…,bn]) implies presenceOf(bi) ⇒ presenceOf(a)
- The binary logical constraints are translated as implications:

  [¬] presenceOf(a) ⇒ [¬] presenceOf(b)
- **Logical network** allows:
  - detecting infeasibilities
  - detecting new implications between intervals
  - fixing presence status of intervals
  - querying in O(1) whether presenceOf(a)⇒presenceOf(b) for any (a,b)
  - triggering events when the relation between two intervals changes

Constraint Propagation: Logical network

- Logical network = Implication graph (as in 2-SAT)
  - Nodes are literals representing the presence value of an interval or its negation (i.e. 2 nodes per interval variable).
  - Arcs are implications
- Literals with equivalent status are merged
- Fixed literals are removed from the graph
- The logical network maintains the transitive closure of implication relation between literals

Temporal Net

[1] Laborie, Rogerie: Reasoning with Conditional Time-intervals. FLAIRS-08.
[2] Laborie, Rogerie: Reasoning with Conditional Time-intervals,
Part II: an Algebraical Model for Resources. FLAIRS-09.

Precedence constraints

- Simple Precedence Constraints $t_i+z \le t_j$ reified by presence statuses

- Example: endBeforeStart(a,b,z) means

  $x(a) \wedge x(b) \Rightarrow e(a) + z \leq s(b)$

- Complete set of precedence constraints:

startBeforeStart,	startBeforeEnd
endBeforeStart,	endBeforeEnd
startAtStart,	startAtEnd
endAtStart,	endAtEnd

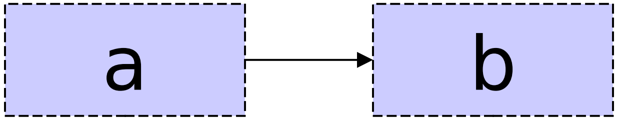
- Presolve recognizes other ways to model precedences, for example:
  endOf(a) <= startOf(b)

Constraint Propagation: Temporal network

- Precedence constraints are aggregated in Temporal network

- Conditional reasoning. Suppose that a and b are optional.

endBeforeStart(a,b):



From Logical network

$\text{presenceOf}(a) \Rightarrow \text{presenceOf}(b)$

- Propagation on the conditional bounds of *a* (would *a* be present) can assume that b will be present too, thus:

$$e_{max}(a) \leftarrow \min(e_{max}(a), s_{max}(b))$$

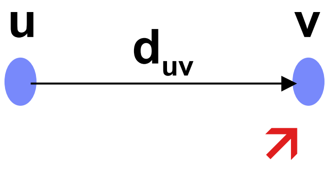
- Bounds are propagated even on interval variables with still undecided presence status.

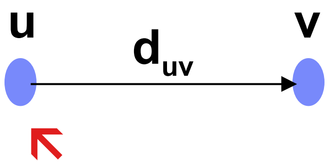
Constraint Propagation: Temporal network

- The temporal network is a directed graph where:
  - nodes are interval end points (start or end)
  - arcs are precedence constraints (with min delay)

- Let u and v be two interval end points and i(u),i(v) respectively denote the intervals of u and v

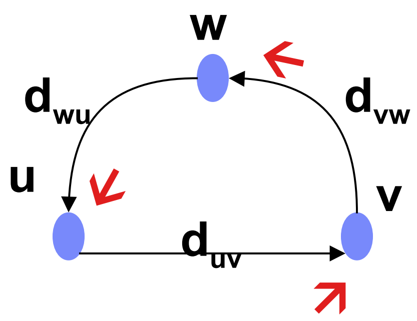
- An arc (u,v,d_{uv}) is said:
  - active on v iff it can propagate on v, that is $presenceOf(i(v)) \Rightarrow presenceOf(i(u))$
  - Active on u iff it can propagate on u, that is $presenceOf(i(u)) \Rightarrow presenceOf(i(v))$





Constraint Propagation: Temporal network

- At root node, an adapted Bellman-Ford algorithm is run:
  - Uses “active on source/target status” to propagate on interval conditional bounds
  - Detects positive cycles between nodes with equivalent presence status

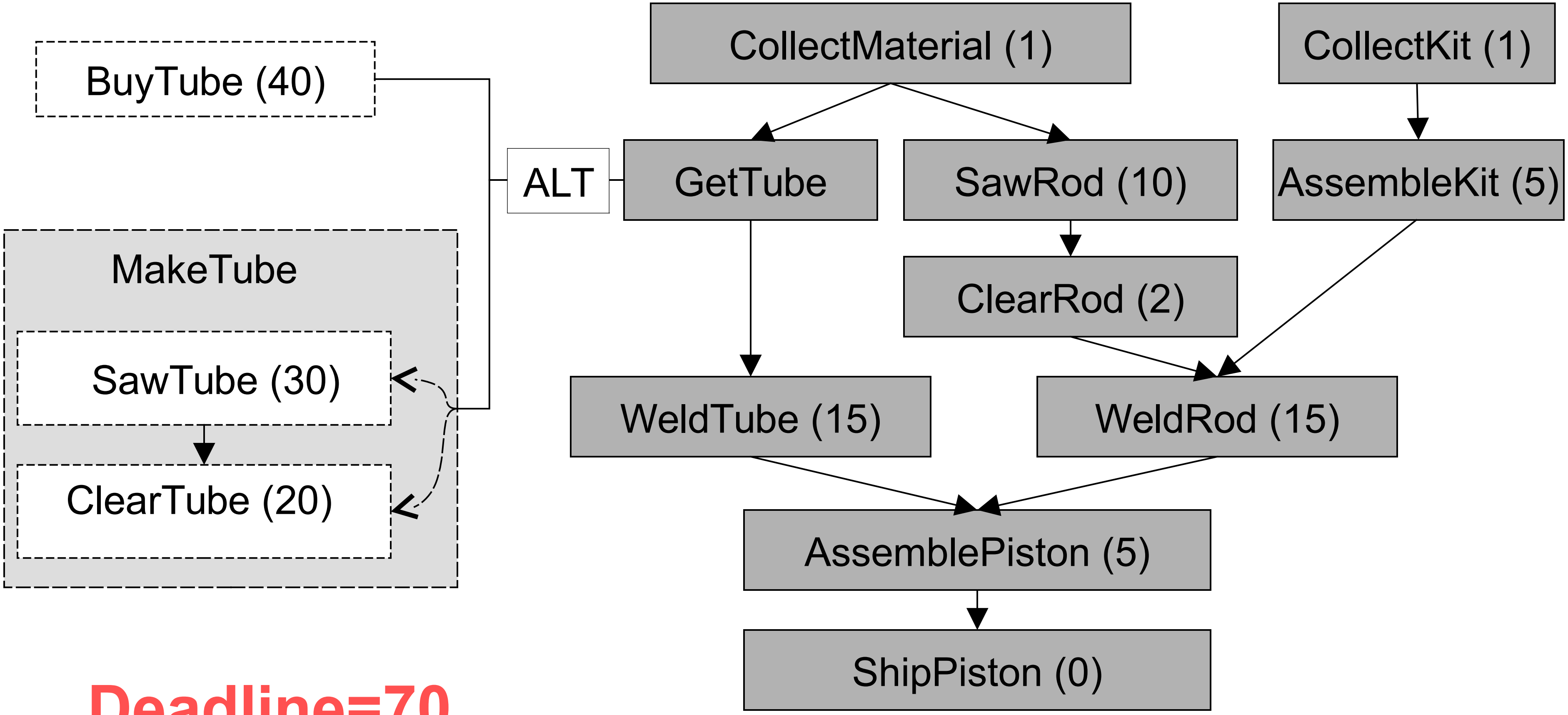


$d_{uv}+d_{vw}+d_{wu}>0 \Rightarrow$!presenceOf(i(u))

- Then, incremental propagation of each arc uses classical bound-consistency
- The temporal network also computes the connected and strongly connected components (useful for the search)

Constraint Propagation: Simple example

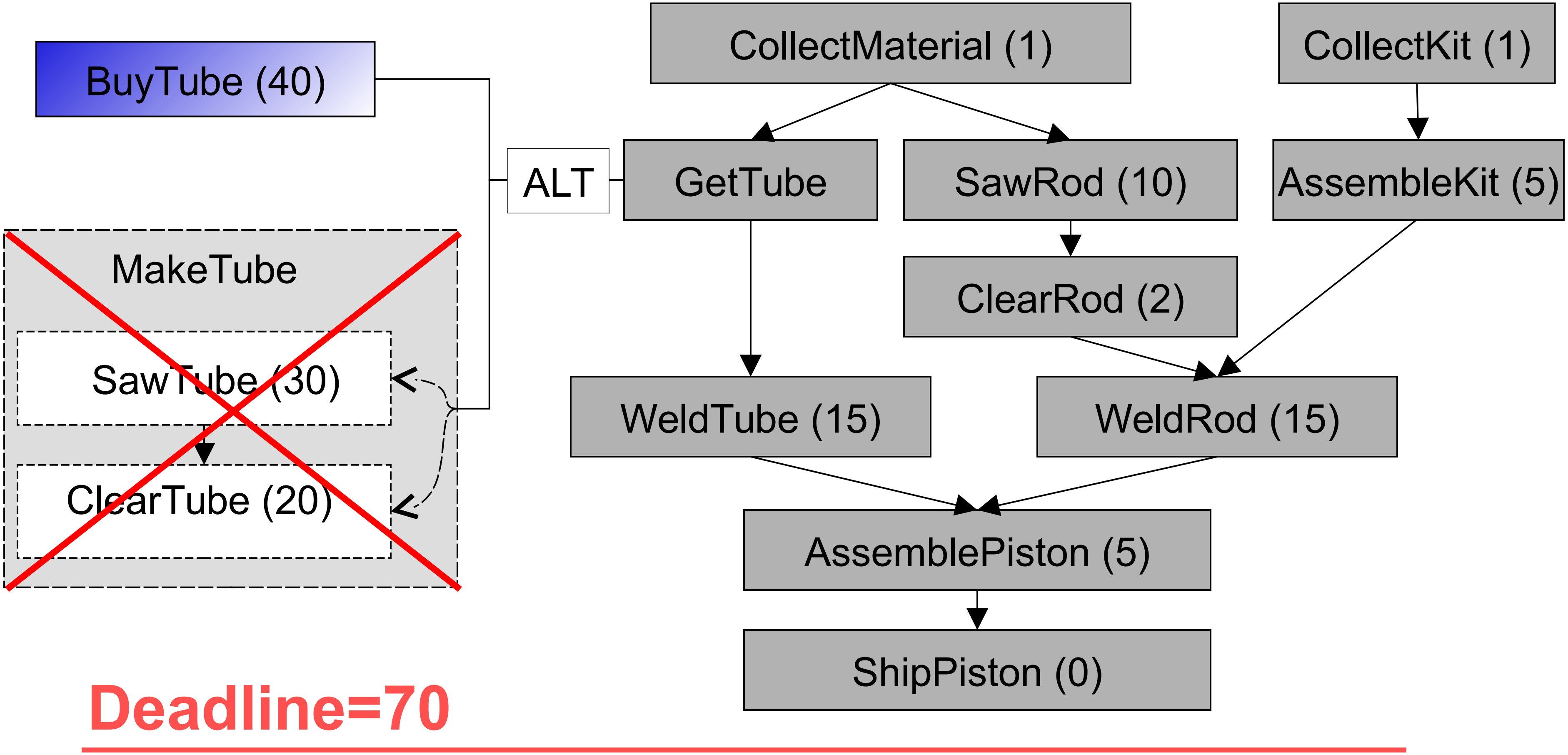
- Inspired from [Barták&Čepek 2007]



Deadline=70

Constraint Propagation: Simple example

- Inspired from [Barták&Čepek 2007]


BuyTube (40)
CollectMaterial (1)
CollectKit (1)
ALT
GetTube
SawRod (10)
AssembleKit (5)
MakeTube
ClearRod (2)
SawTube (30)
WeldTube (15)
WeldRod (15)
ClearTube (20)
AssemblePiston (5)
ShipPiston (0)
Deadline=70