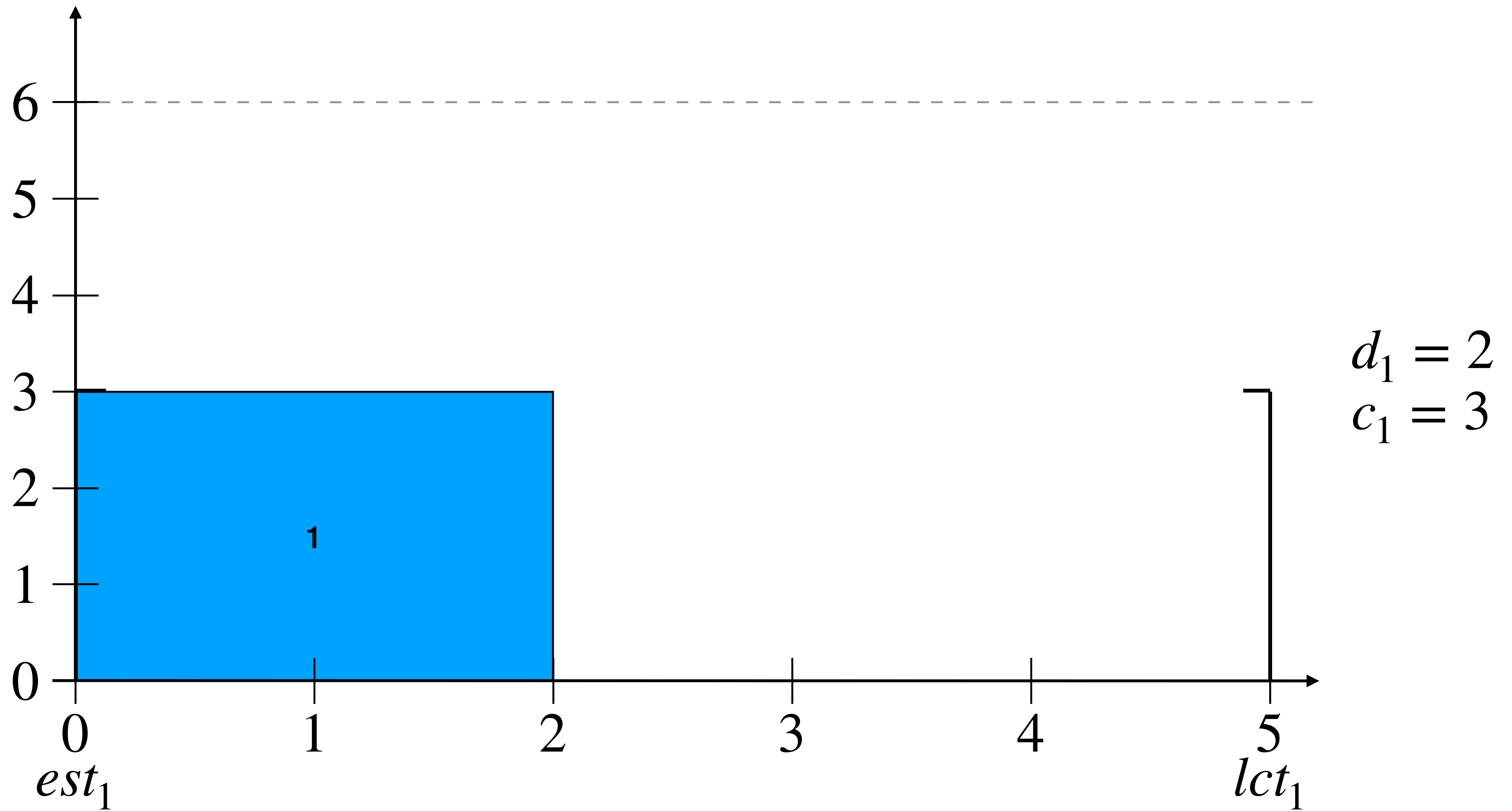


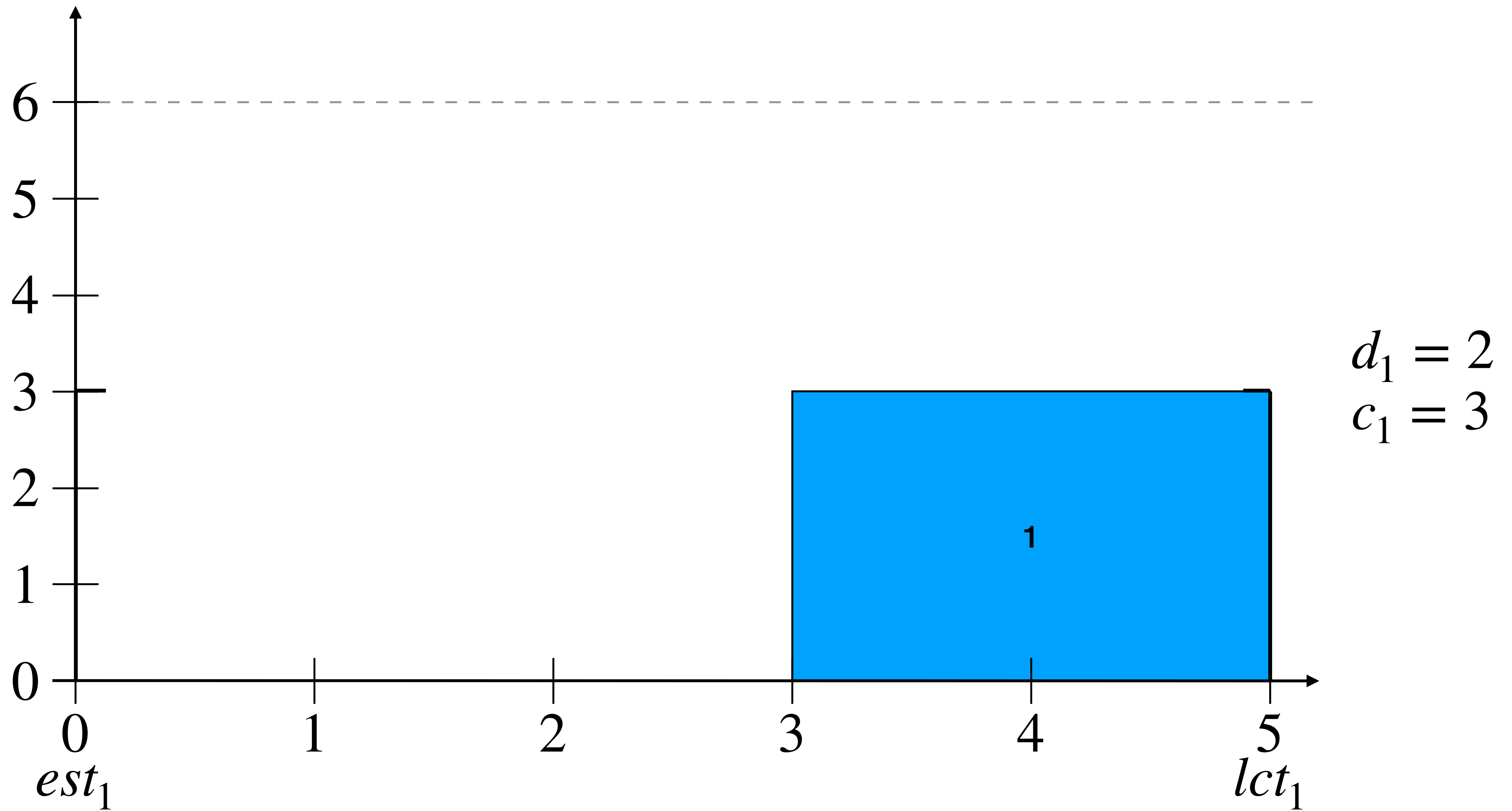
Augmenting the Cumulative Overload Check with Integral Resource Usage Reasoning

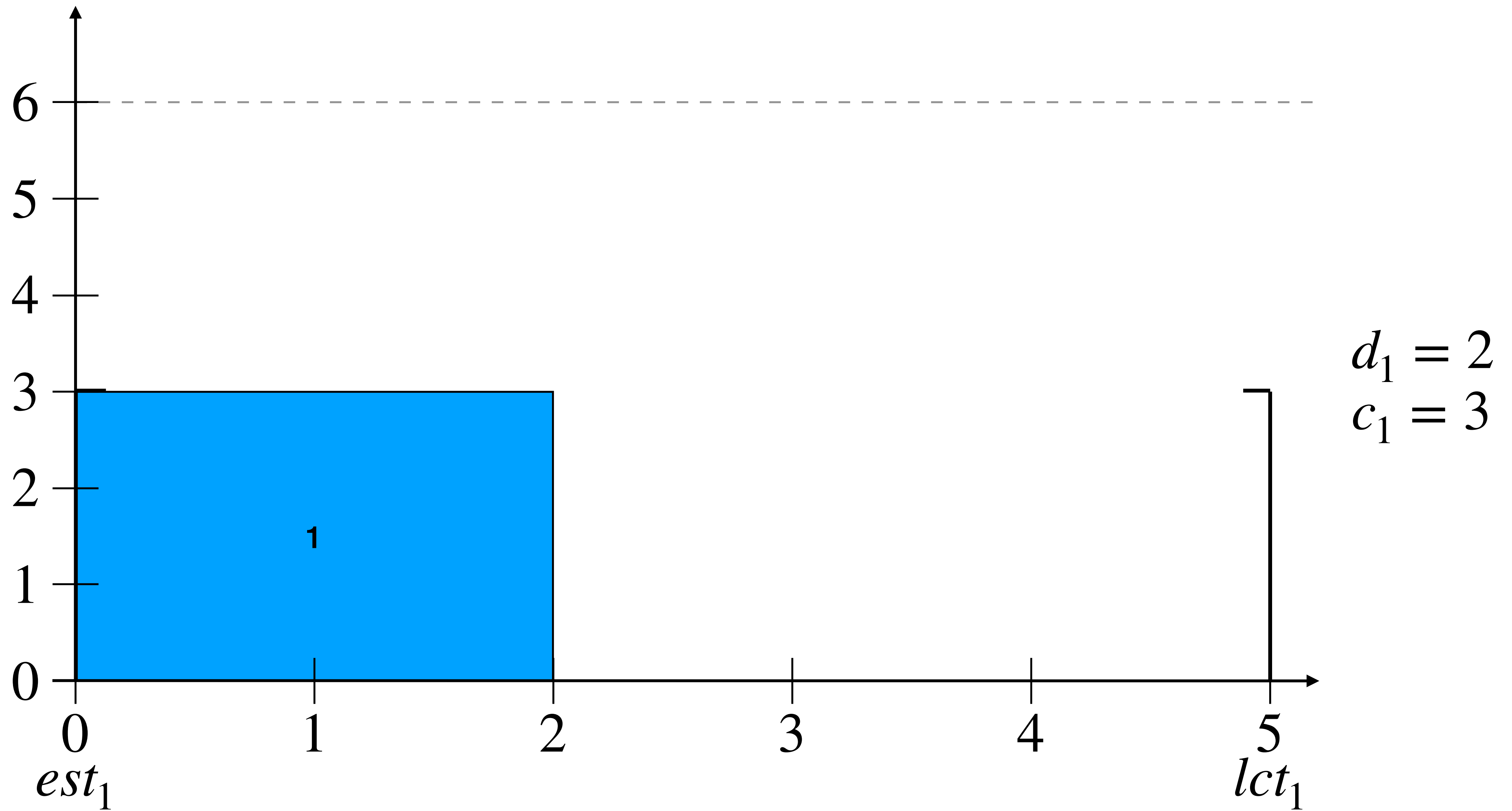
Samuel Cloutier* and Claude-Guy Quimper - Université Laval

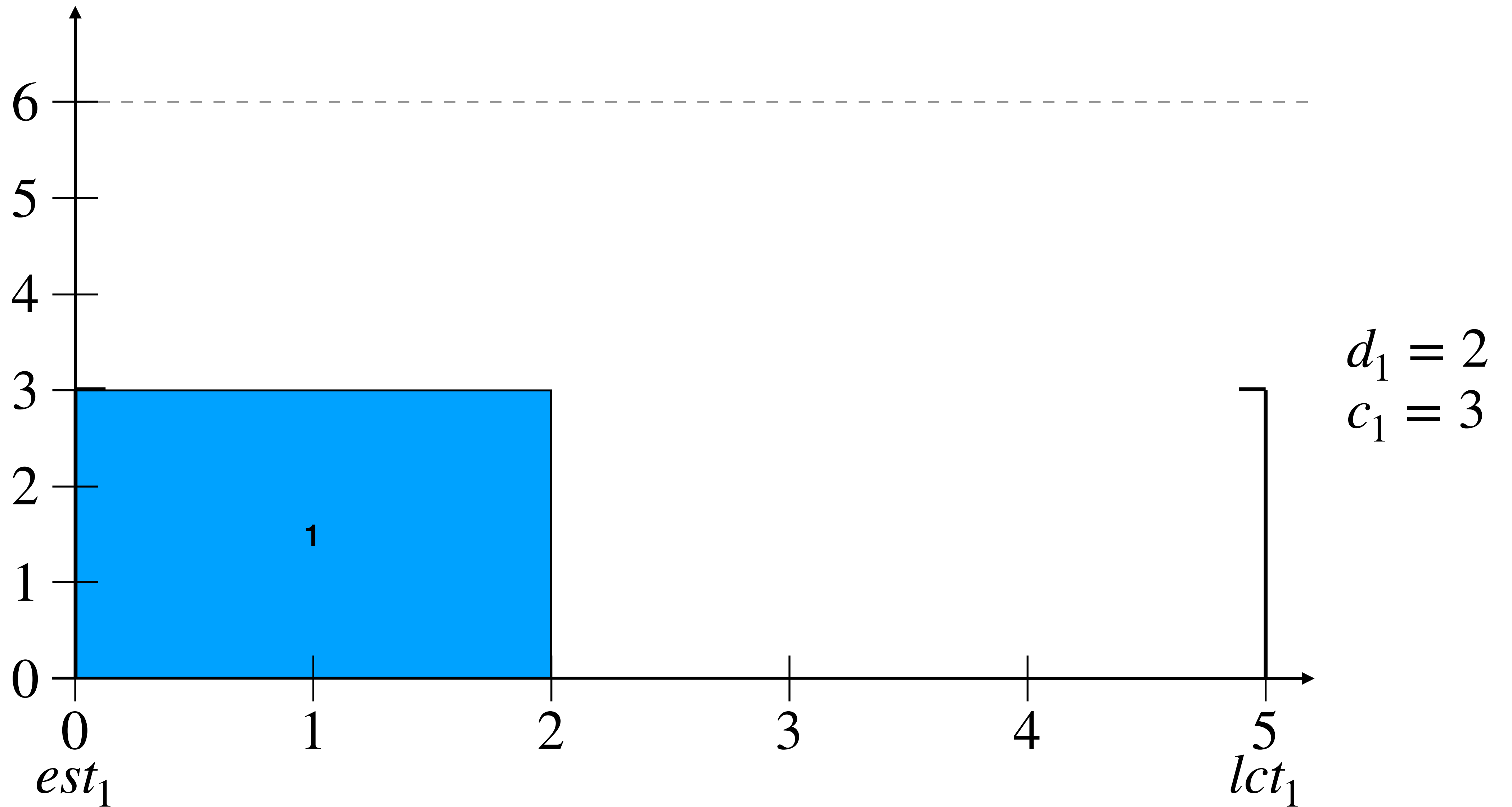
Cumulative Constraint

- We must schedule tasks in time so that the total resource consumption never overloads the capacity.
- A task has:
 1. A duration (d)
 2. A resource consumption (c)
 3. A starting time variable ($S \in [est, lst]$)







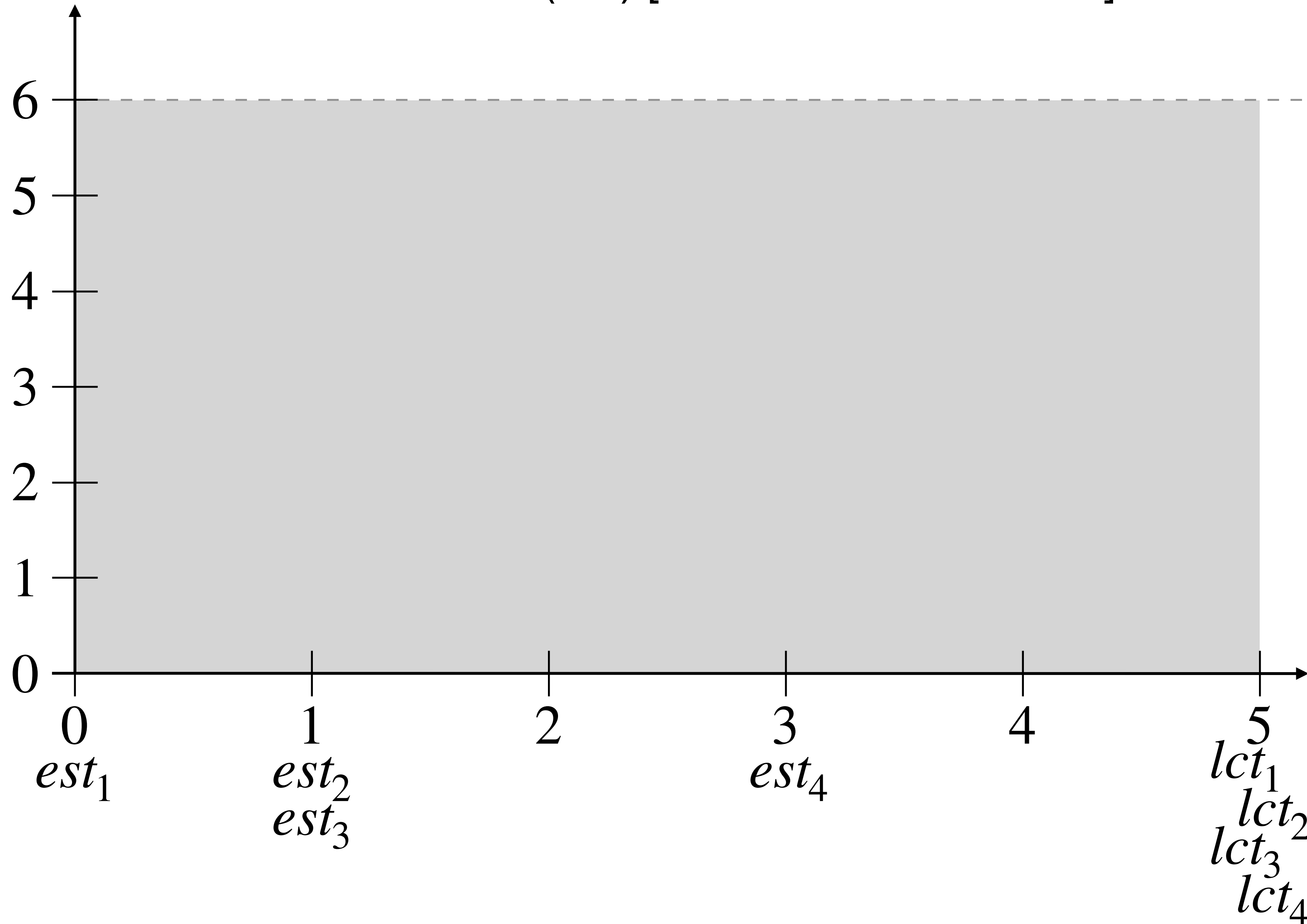


Resource constraint: CUMULATIVE(**S**, **d**, **c**, C)

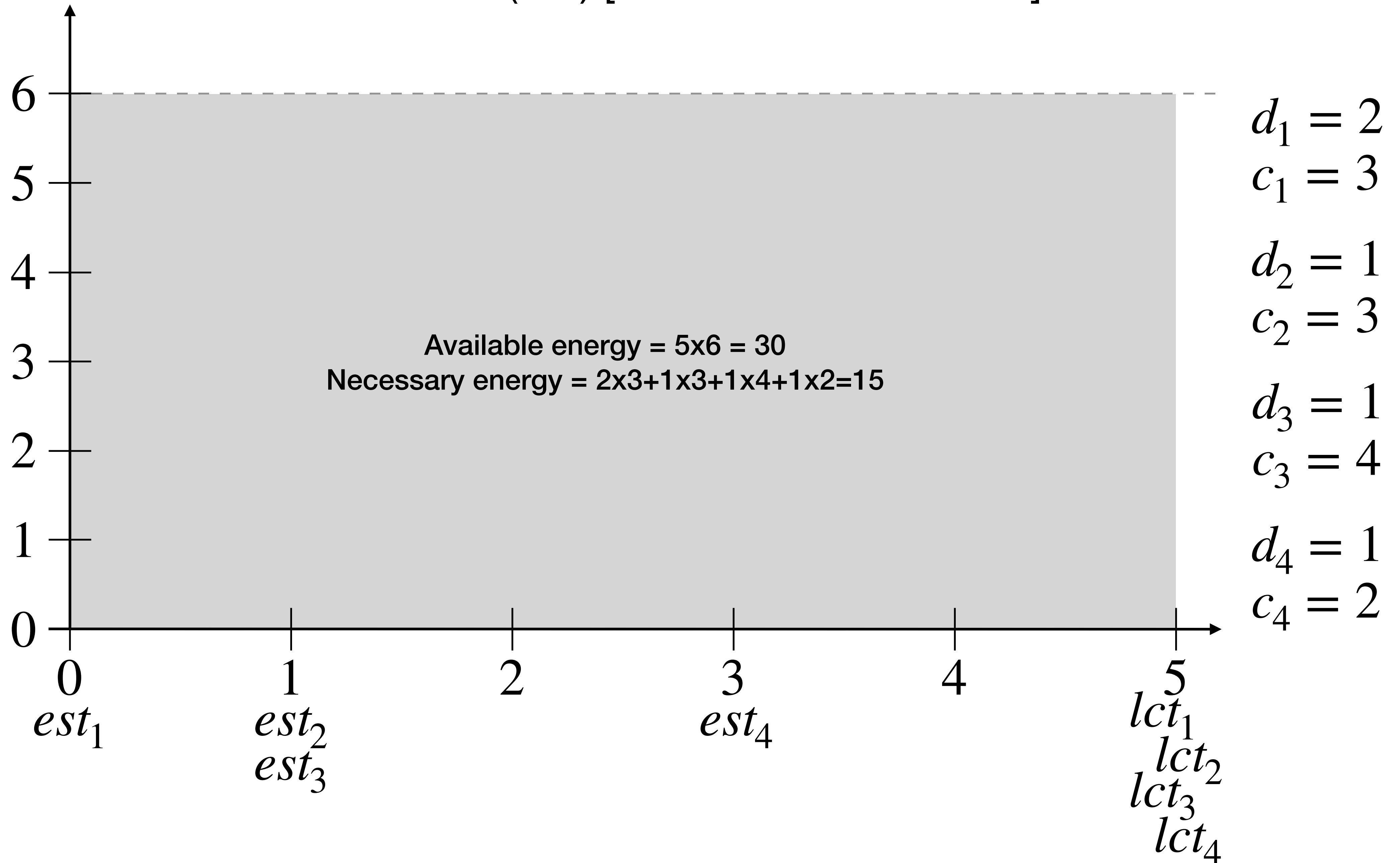
Checker for the Cumulative Constraint

- It is difficult to determine the satisfiability of the Cumulative constraint.
- Weaker rules are used in practice.

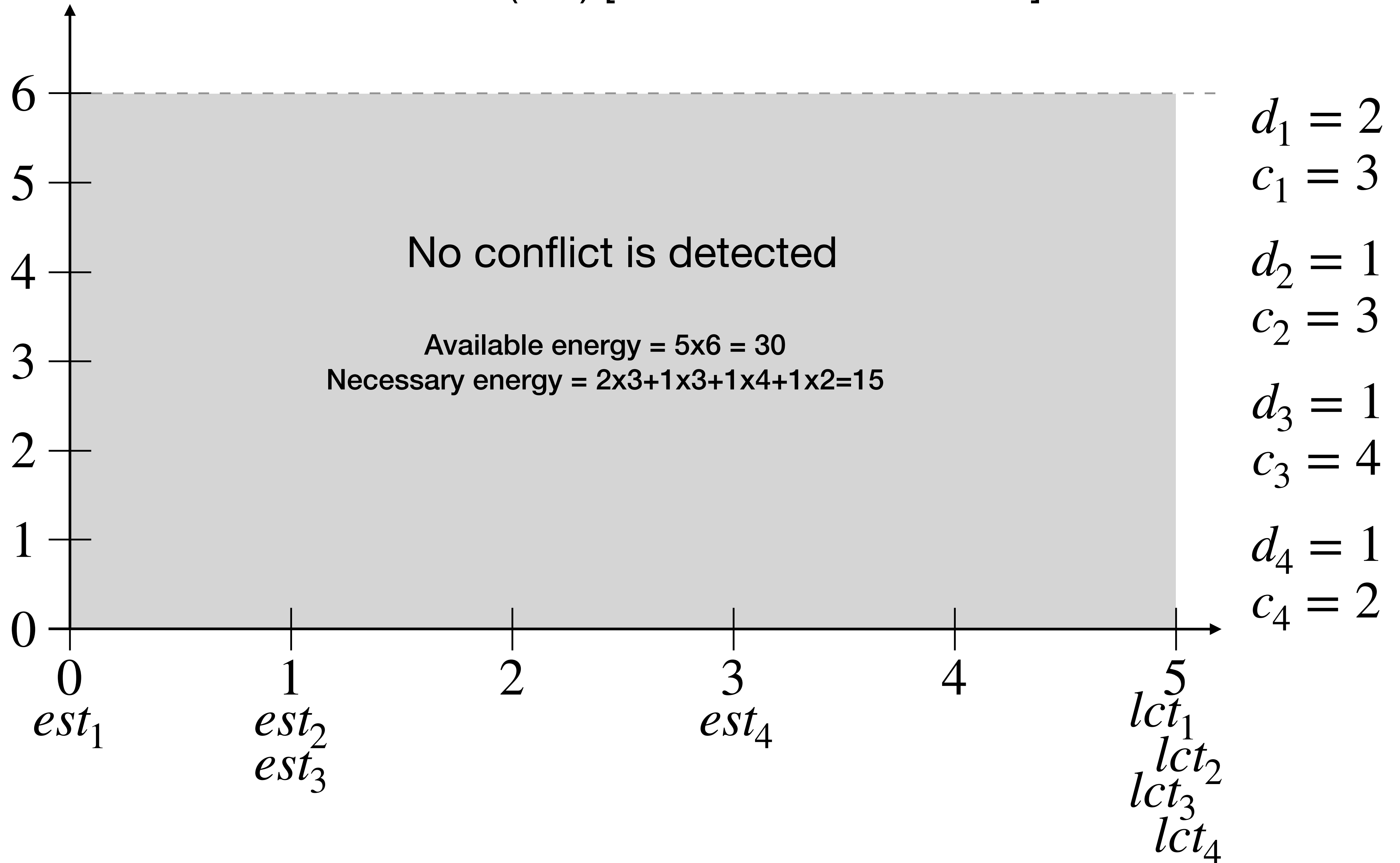
Overload Check (OC) [Wolf & Schrader 2006]



Overload Check (OC) [Wolf & Schrader 2006]



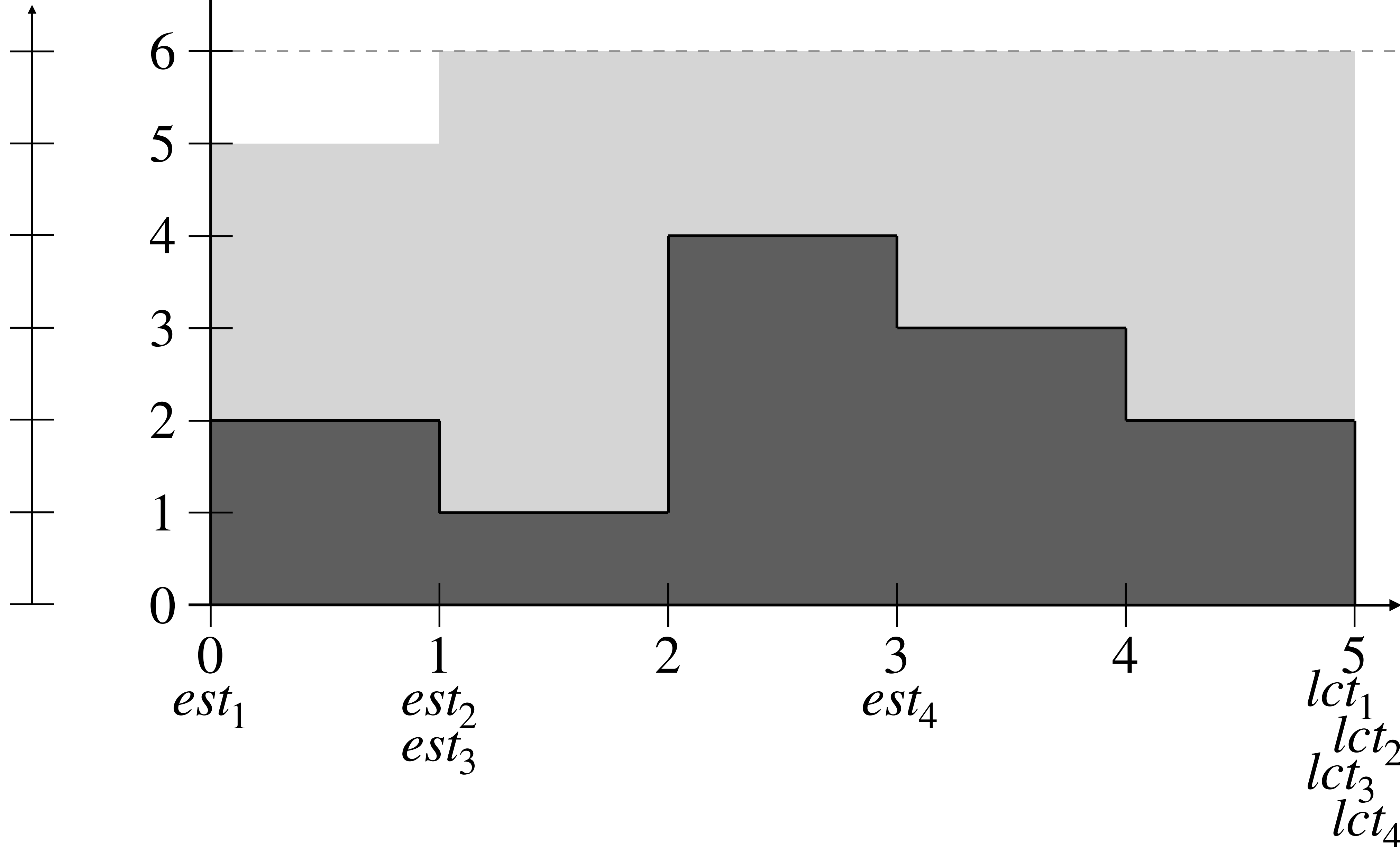
Overload Check (OC) [Wolf & Schrader 2006]



Time Table Horizontally Elastic Overload Check (TTHEOC)

[Kameugne et al. 2024]

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

$$d_4 = 1$$

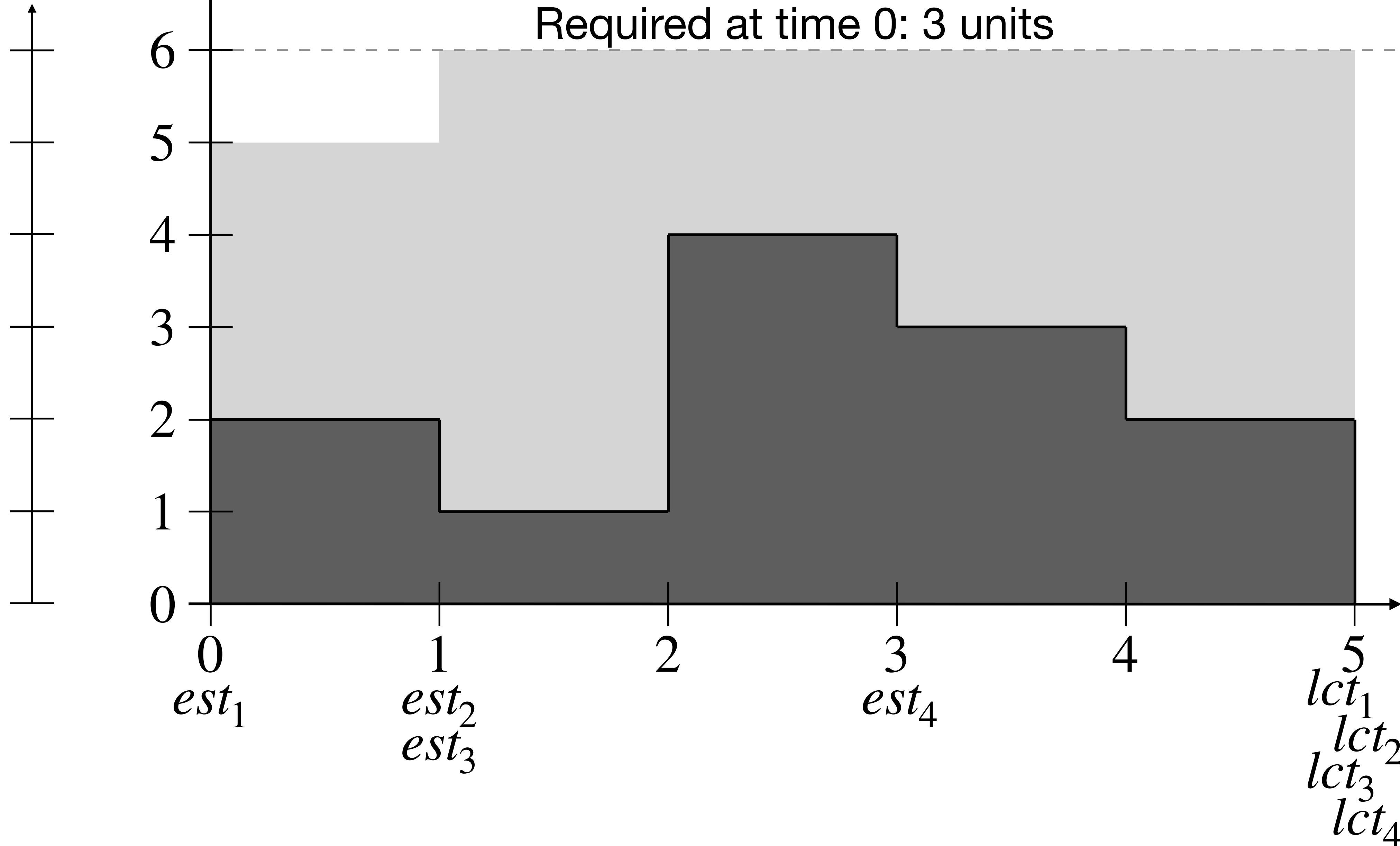
$$c_4 = 2$$

Time Table Horizontally Elastic Overload Check (TTHEOC)

[Kameugne et al. 2024]

Required at time 0: 3 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

$$d_4 = 1$$

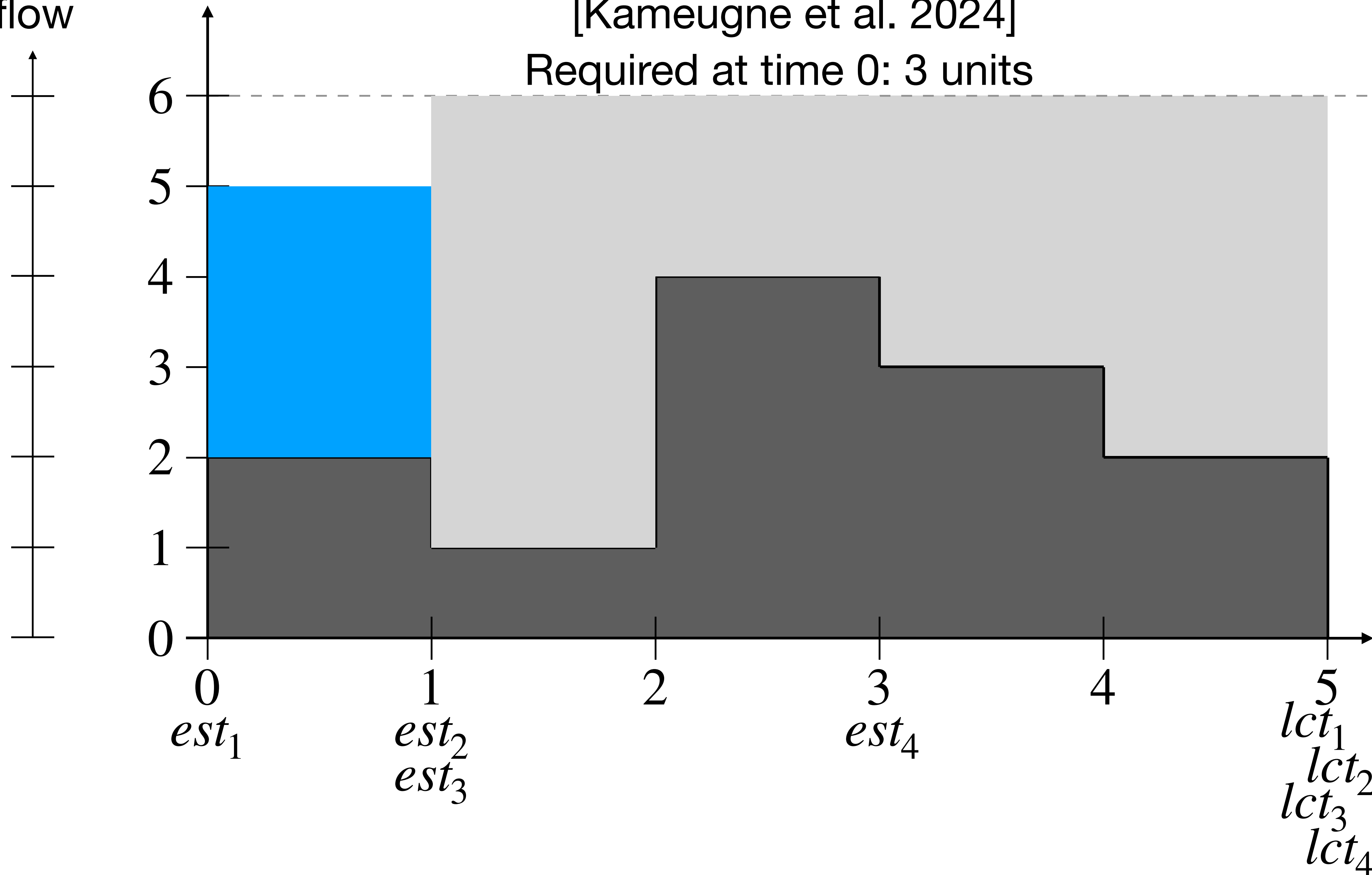
$$c_4 = 2$$

Time Table Horizontally Elastic Overload Check (TTHEOC)

[Kameugne et al. 2024]

Required at time 0: 3 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

$$d_4 = 1$$

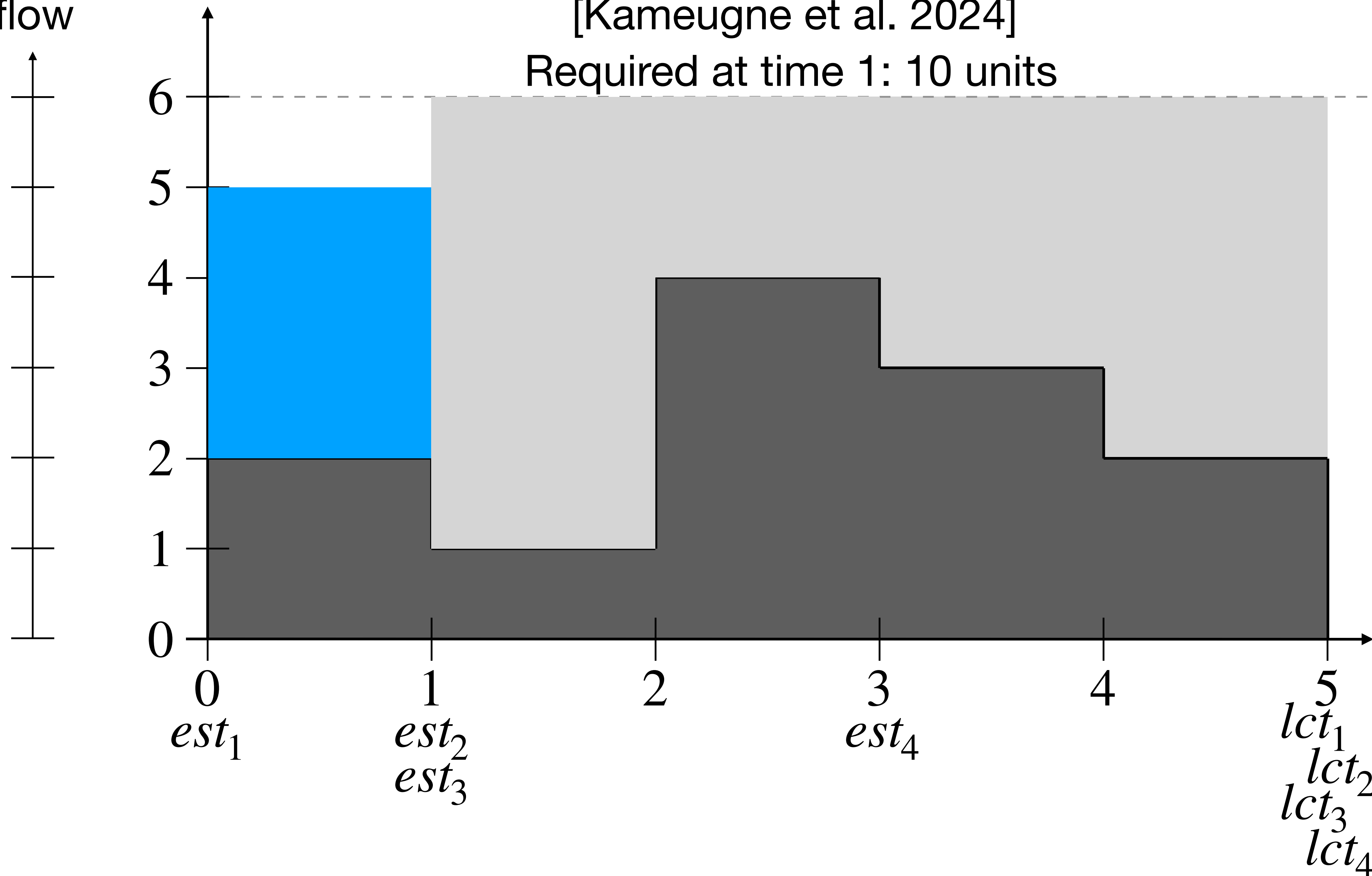
$$c_4 = 2$$

Time Table Horizontally Elastic Overload Check (TTHEOC)

[Kameugne et al. 2024]

Required at time 1: 10 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

$$d_4 = 1$$

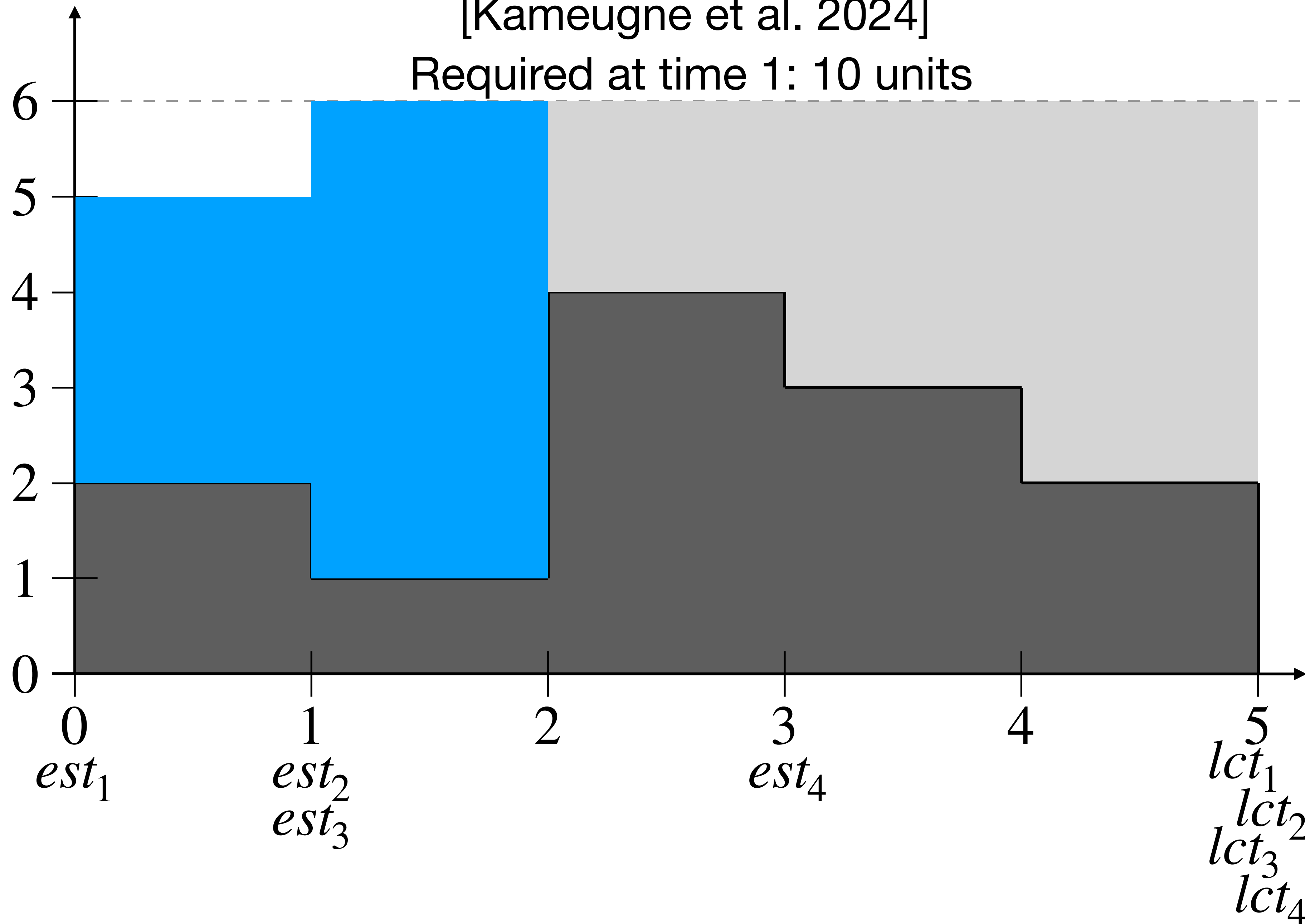
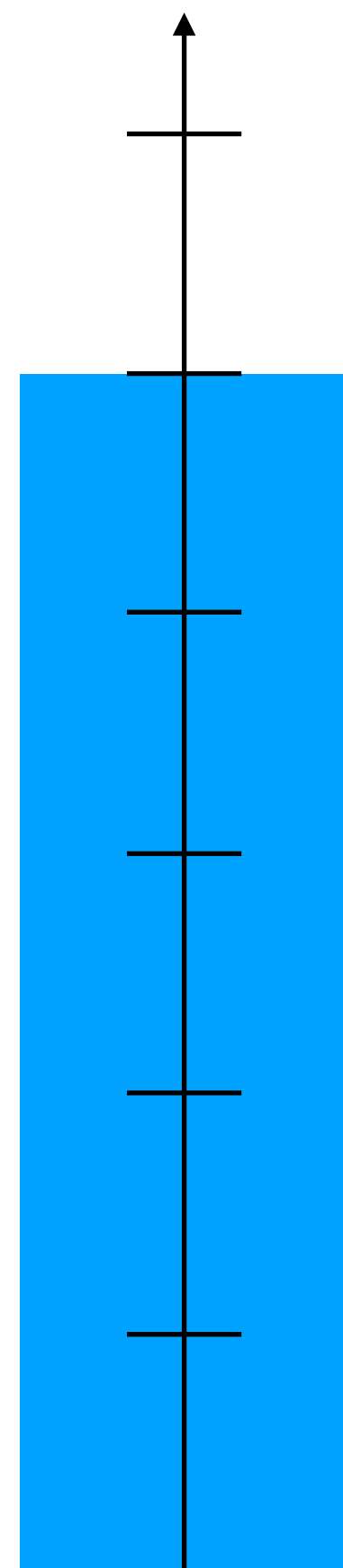
$$c_4 = 2$$

Time Table Horizontally Elastic Overload Check (TTHEOC)

[Kameugne et al. 2024]

Required at time 1: 10 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

$$d_4 = 1$$

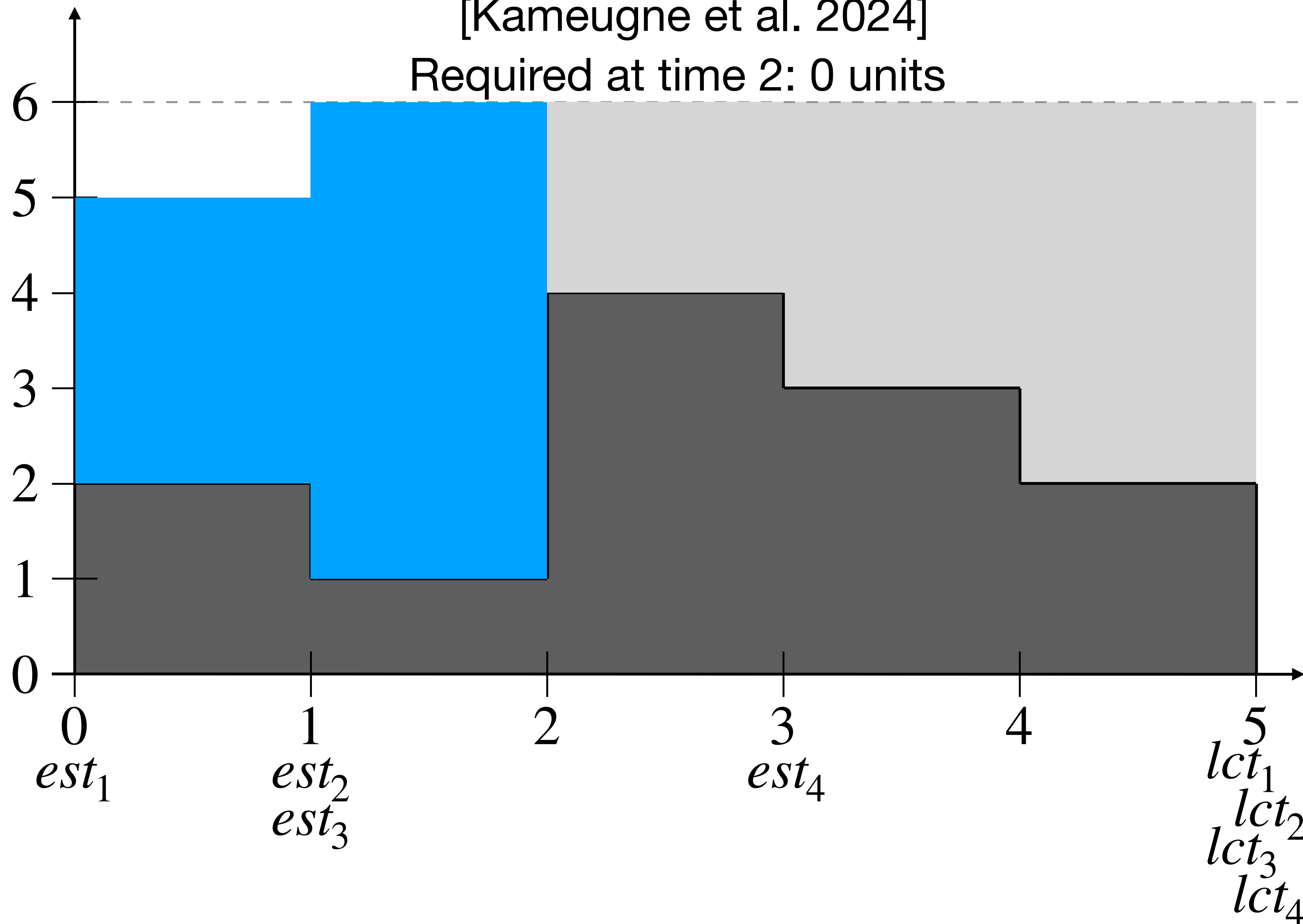
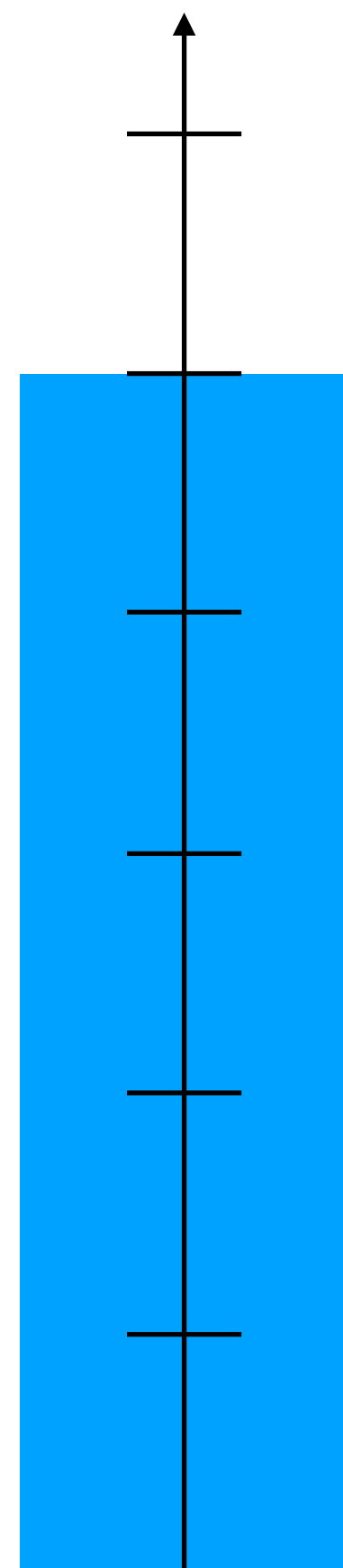
$$c_4 = 2$$

Time Table Horizontally Elastic Overload Check (TTHEOC)

[Kameugne et al. 2024]

Required at time 2: 0 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

$$d_4 = 1$$

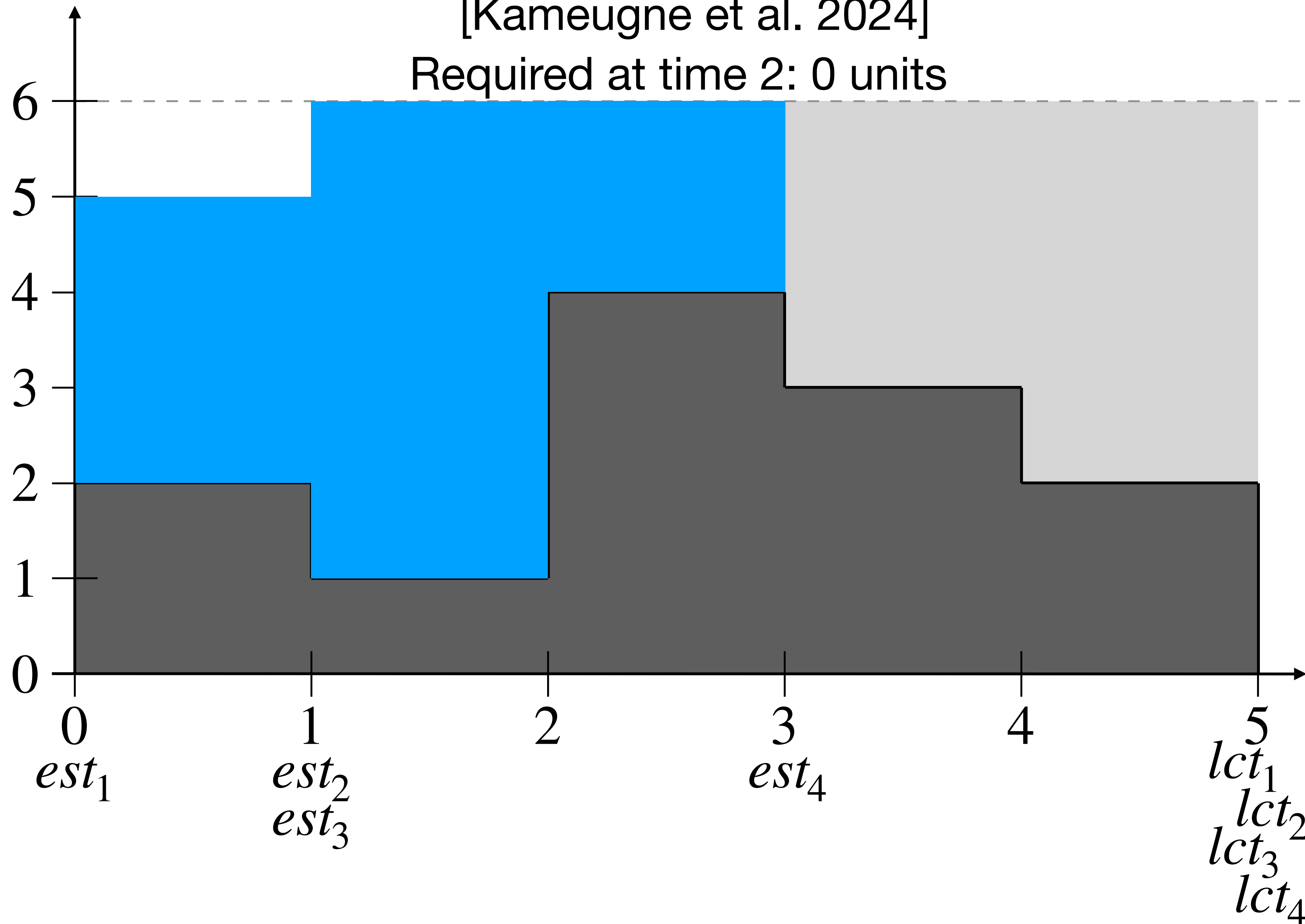
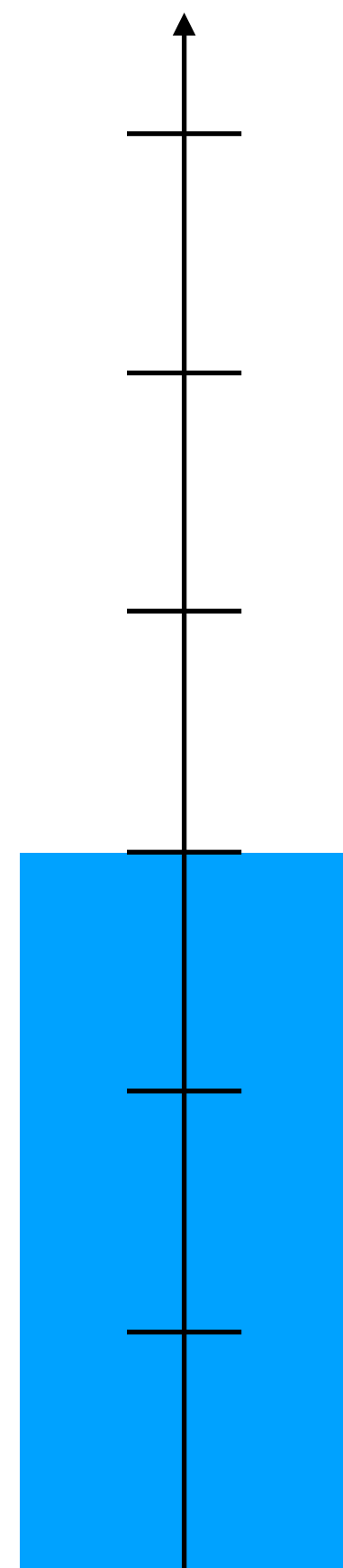
$$c_4 = 2$$

Time Table Horizontally Elastic Overload Check (TTHEOC)

[Kameugne et al. 2024]

Required at time 2: 0 units

Overflow

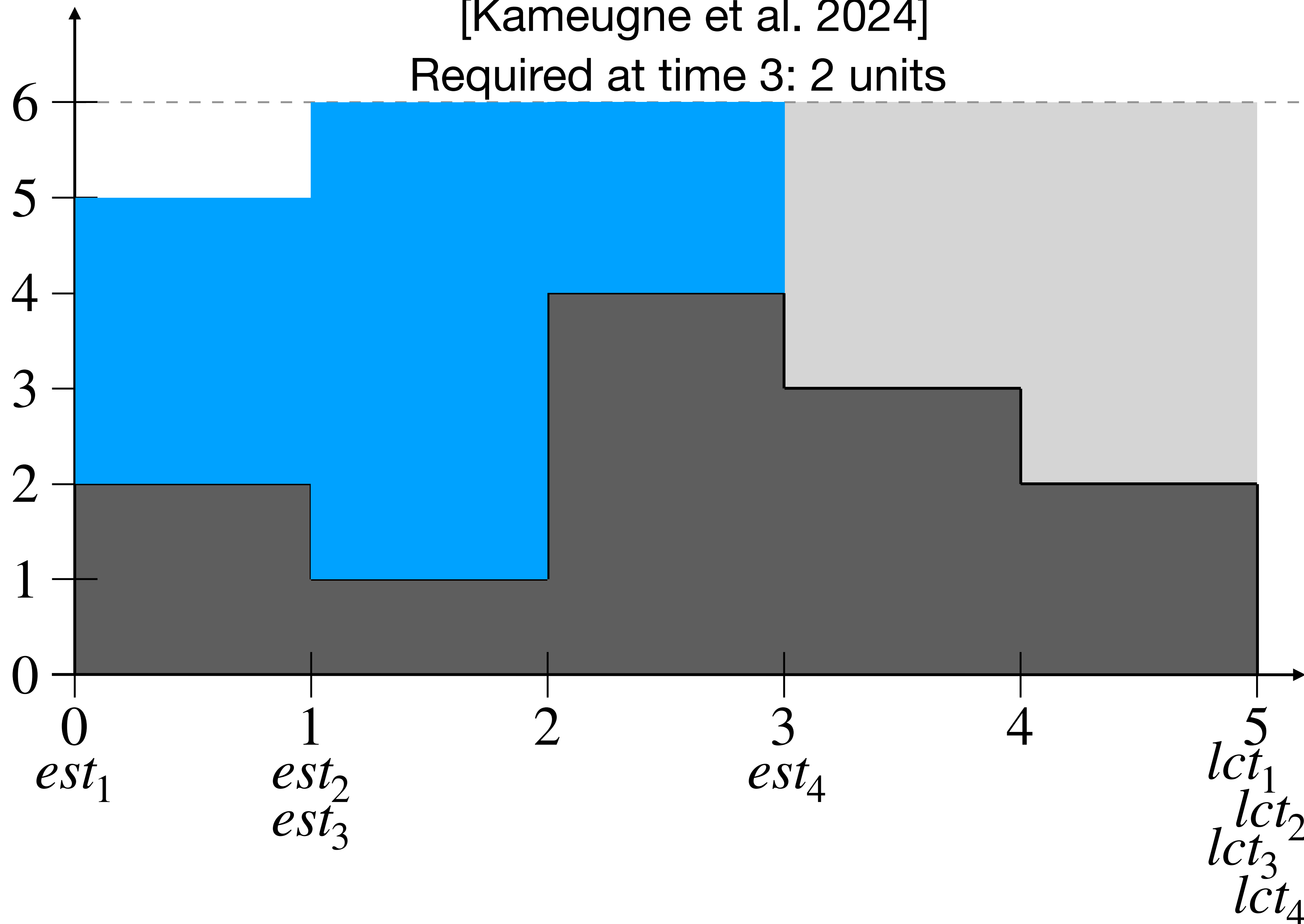
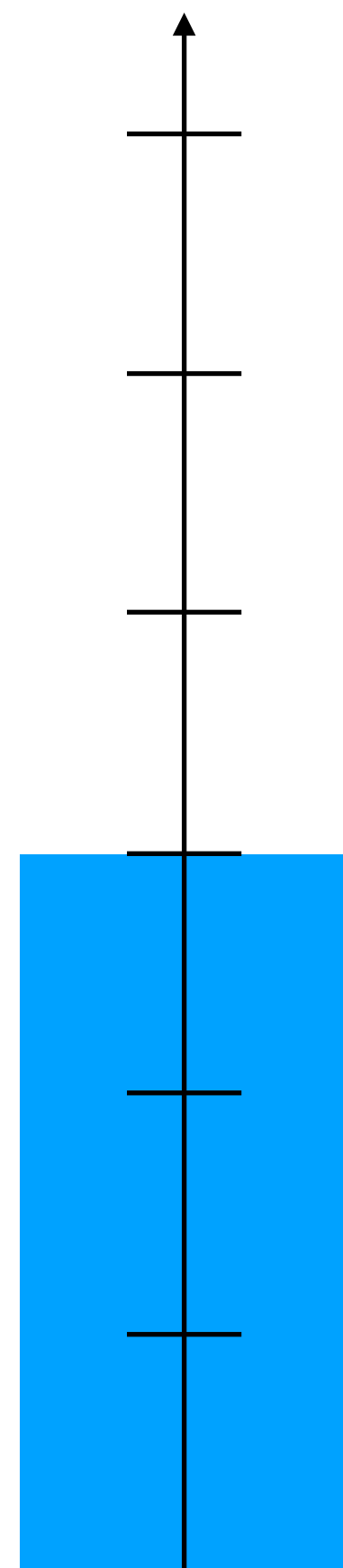


Time Table Horizontally Elastic Overload Check (TTHEOC)

[Kameugne et al. 2024]

Required at time 3: 2 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

$$d_4 = 1$$

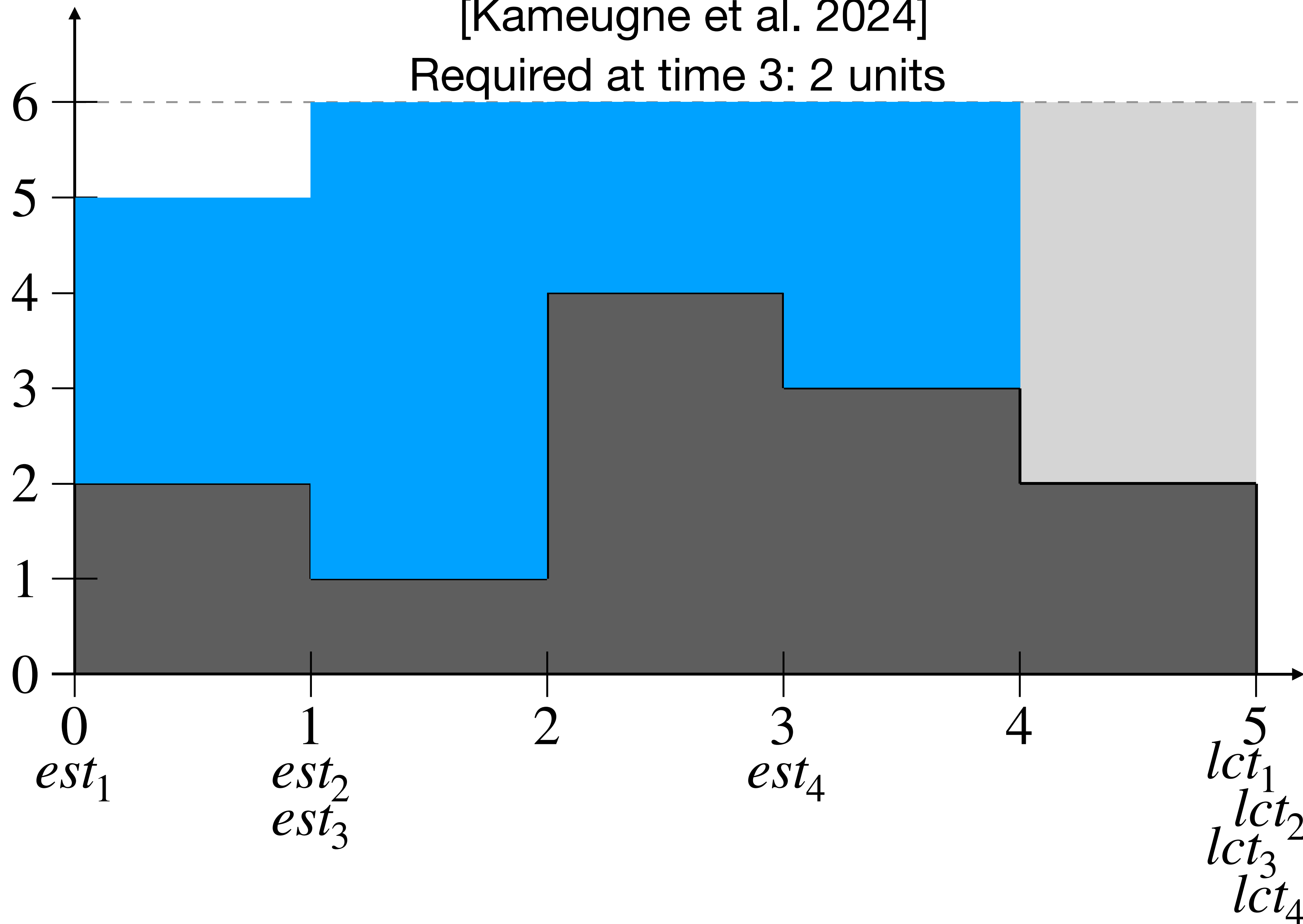
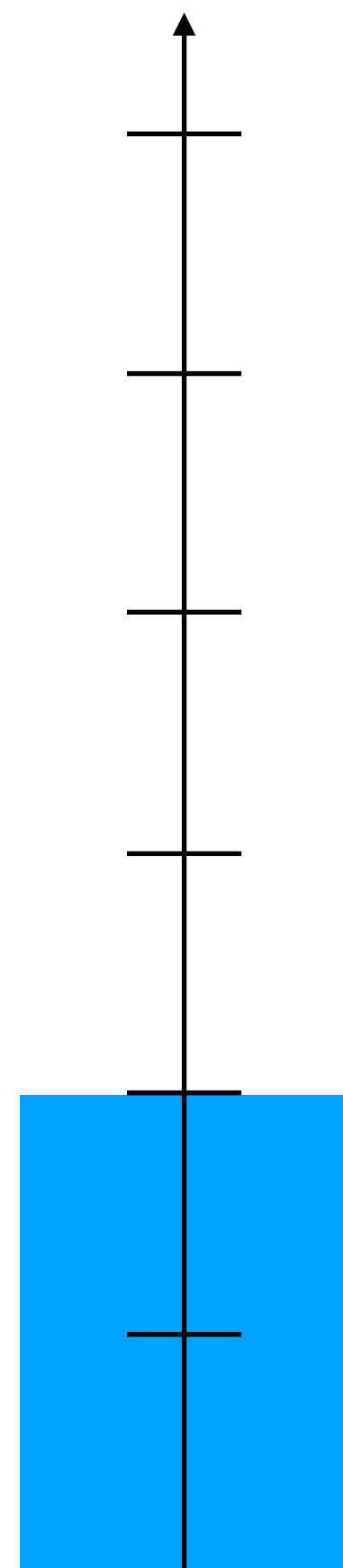
$$c_4 = 2$$

Time Table Horizontally Elastic Overload Check (TTHEOC)

[Kameugne et al. 2024]

Required at time 3: 2 units

Overflow

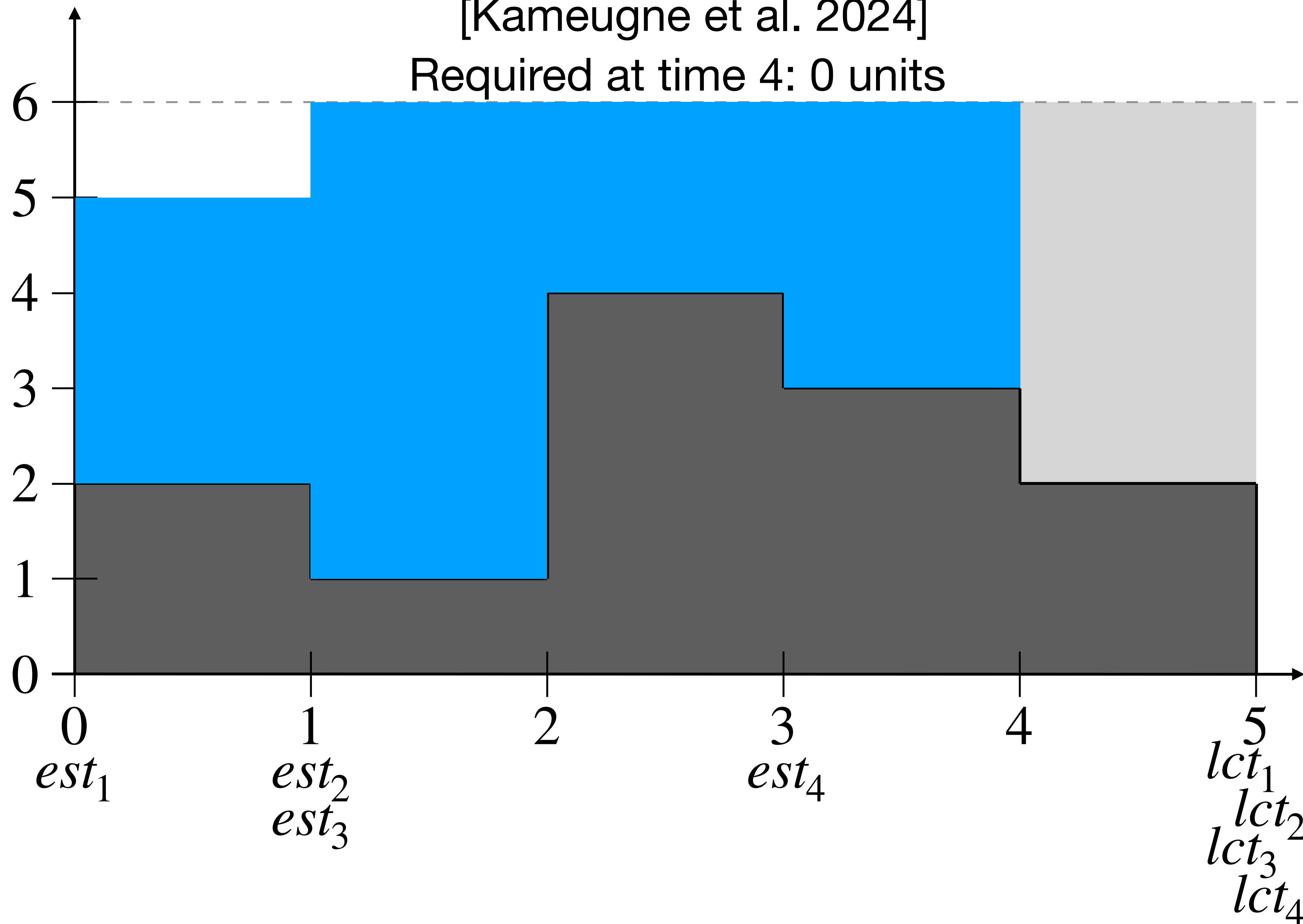
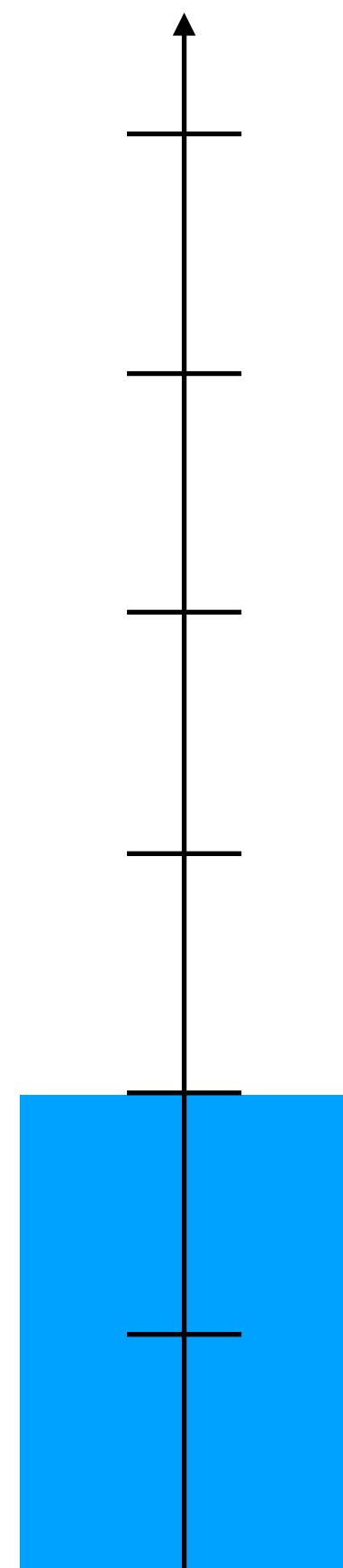


Time Table Horizontally Elastic Overload Check (TTHEOC)

[Kameugne et al. 2024]

Required at time 4: 0 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

$$d_4 = 1$$

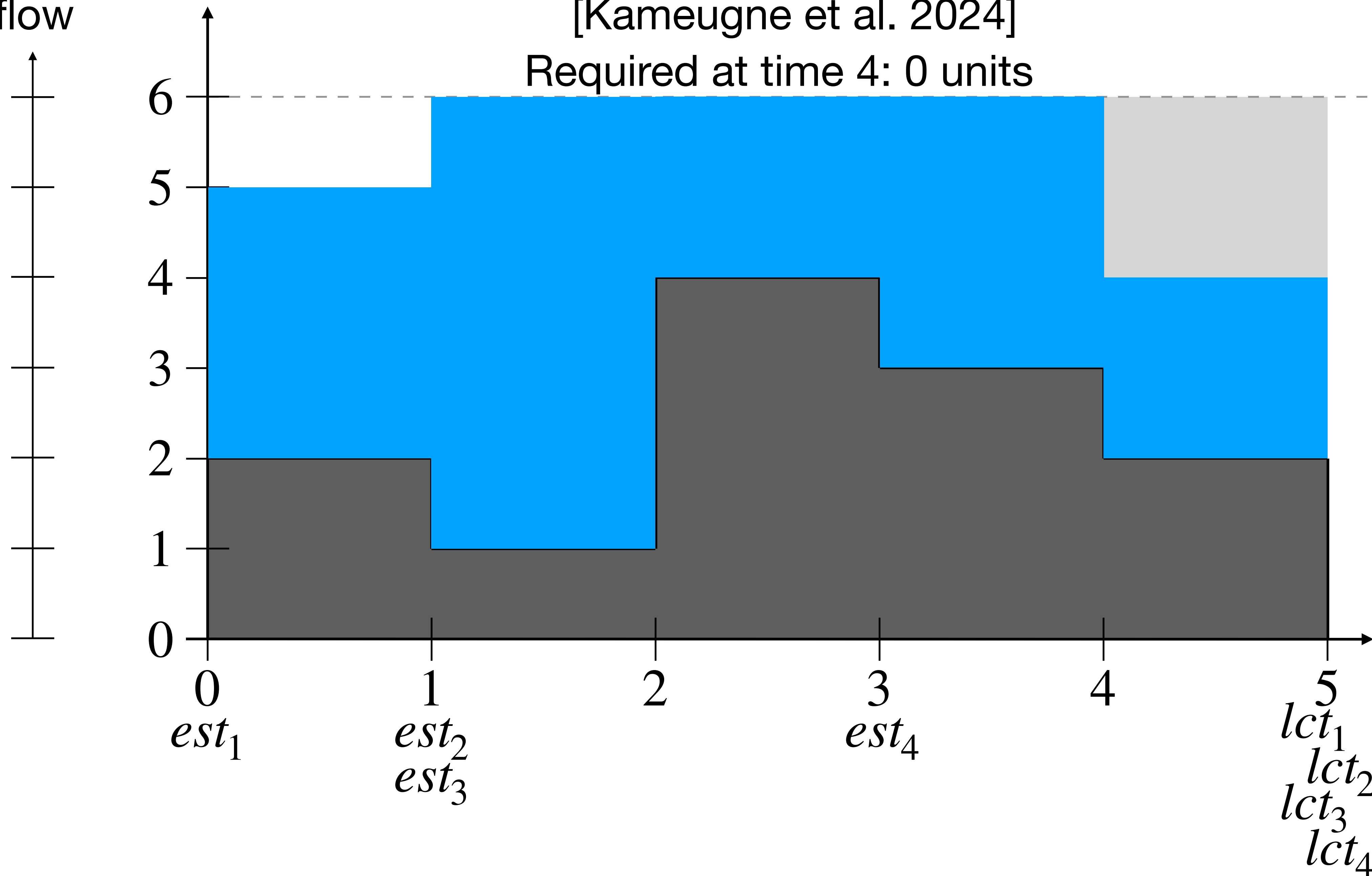
$$c_4 = 2$$

Time Table Horizontally Elastic Overload Check (TTHEOC)

[Kameugne et al. 2024]

Required at time 4: 0 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

$$d_4 = 1$$

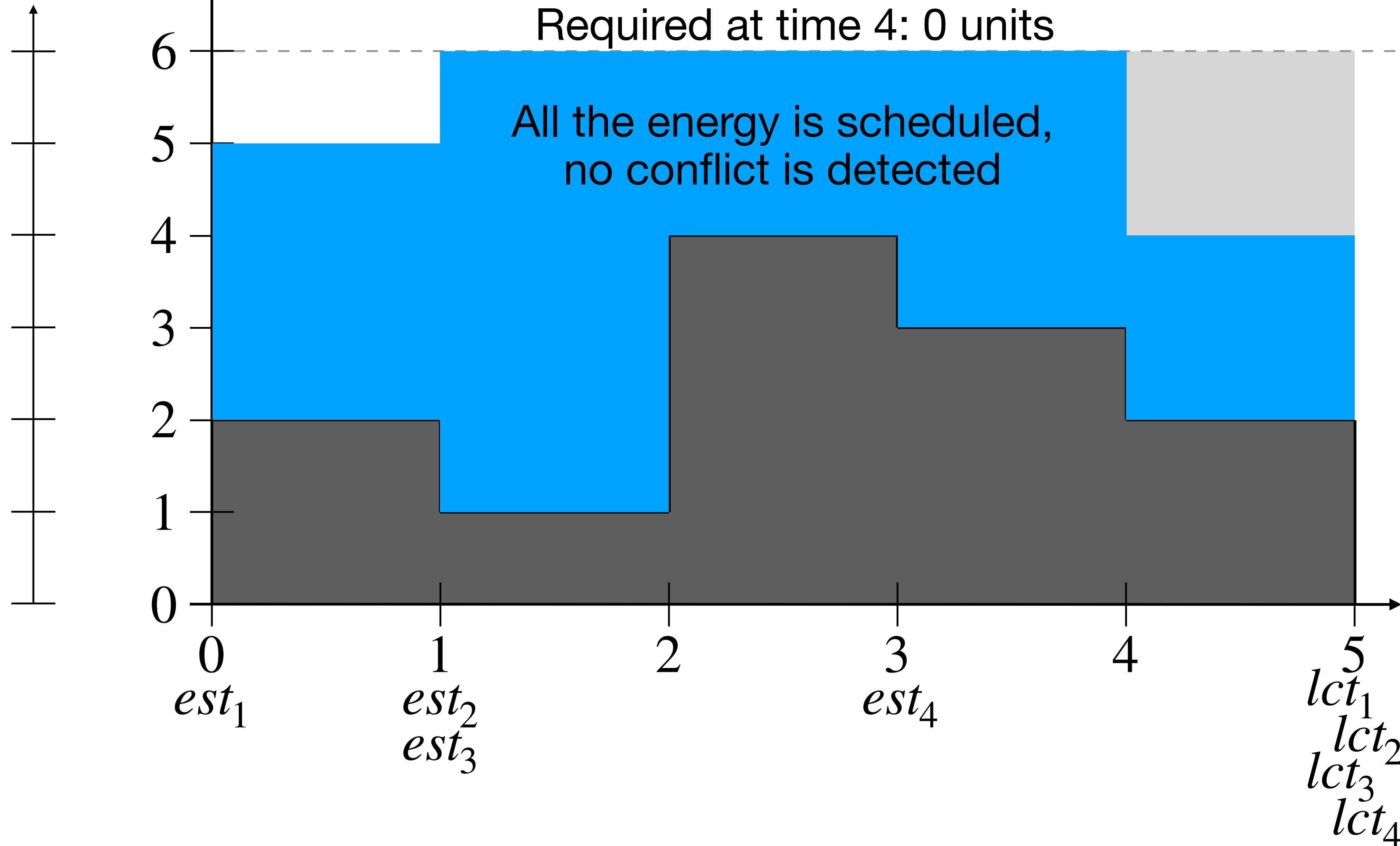
$$c_4 = 2$$

Time Table Horizontally Elastic Overload Check (TTHEOC)

[Kameugne et al. 2024]

Required at time 4: 0 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

$$d_4 = 1$$

$$c_4 = 2$$

Our Contribution

A New Rule Using Enhanced Bounds on the Resource Availability

- Resource consumptions are integral quantities, the task is either using the resource or not.
- To know the resource actually available at some time, we must find the greatest total resource consumption at that time.

Reasoning on Integral Consumptions

Applied to the previous exemple

- At time 1, 5 units of resource are free, and the tasks that may be running have consumptions of: 3, 3, and 4.

Reasoning on Integral Consumptions

Applied to the previous exemple

- At time 1, 5 units of resource are free, and the tasks that may be running have consumptions of: 3, 3, and 4.
- Can the 5 units of resource be used by these tasks?

Reasoning on Integral Consumptions

Applied to the previous exemple

- At time 1, 5 units of resource are free, and the tasks that may be running have consumptions of: 3, 3, and 4.
- Can the 5 units of resource be used by these tasks?
- At most one task can run at time 1, so at most 4 units of resource are actually available at time 1.

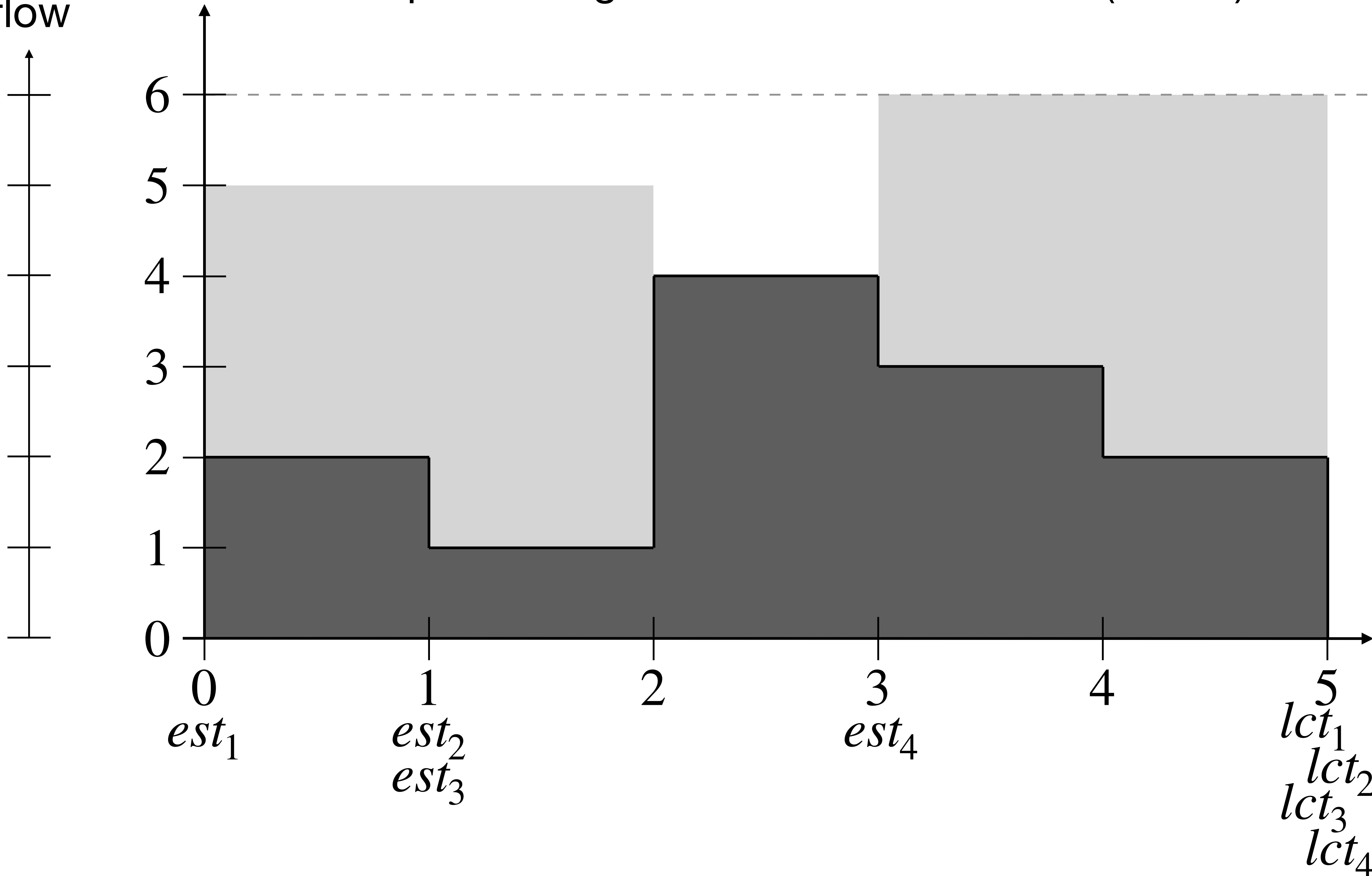
Reasoning on Integral Consumptions

Applied to the previous exemple

- At time 1, 5 units of resource are free, and the tasks that may be running have consumptions of: 3, 3, and 4.
- Can the 5 units of resource be used by these tasks?
- At most one task can run at time 1, so at most 4 units of resource are actually available at time 1.
- Applying such reasoning means solving a knapsack/subset-sum maximisation problem for each time point.

Knapsack-Augmented Overload Check (KAOC)

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

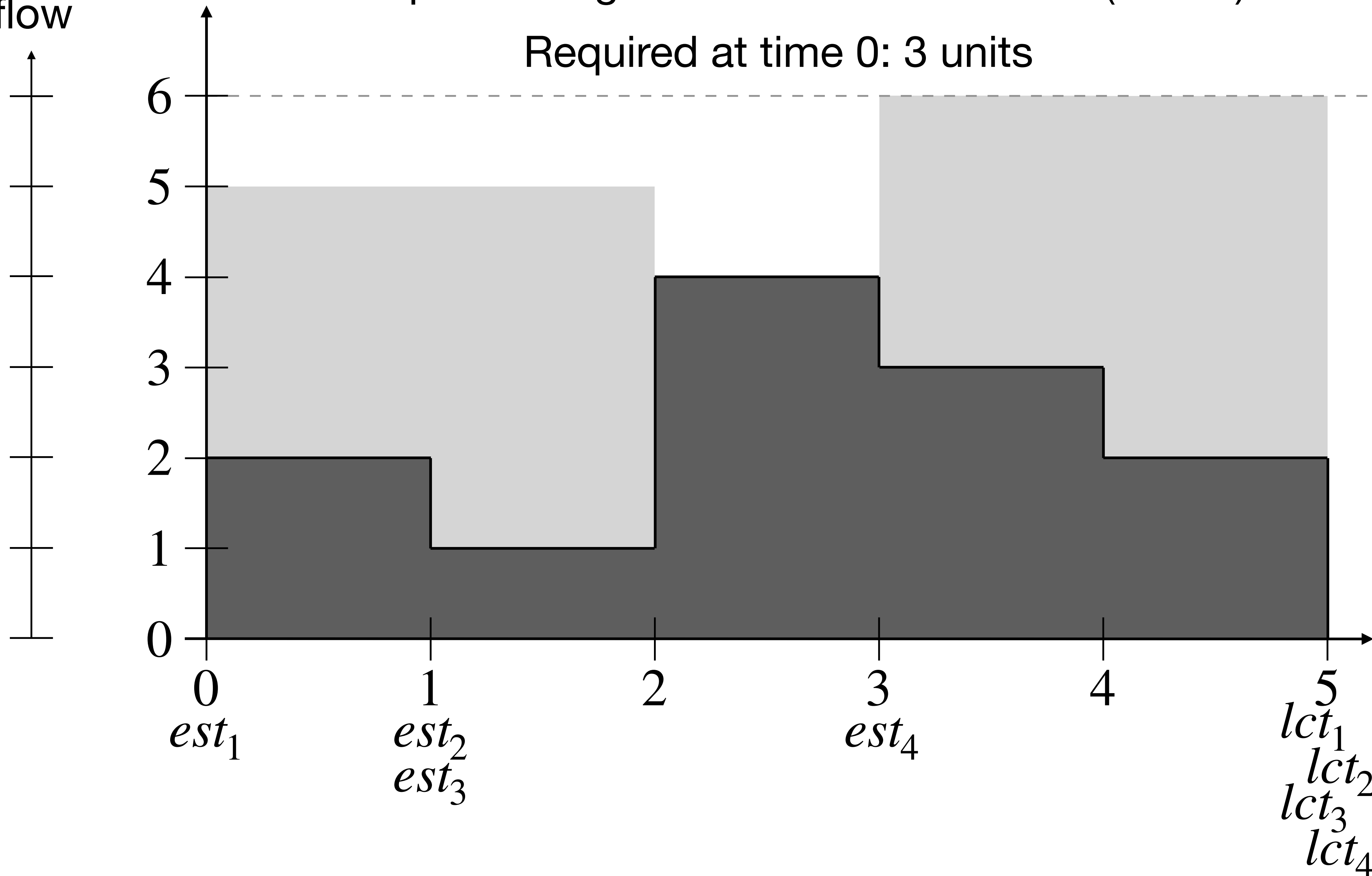
$$d_4 = 1$$

$$c_4 = 2$$

Knapsack-Augmented Overload Check (KAOC)

Required at time 0: 3 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

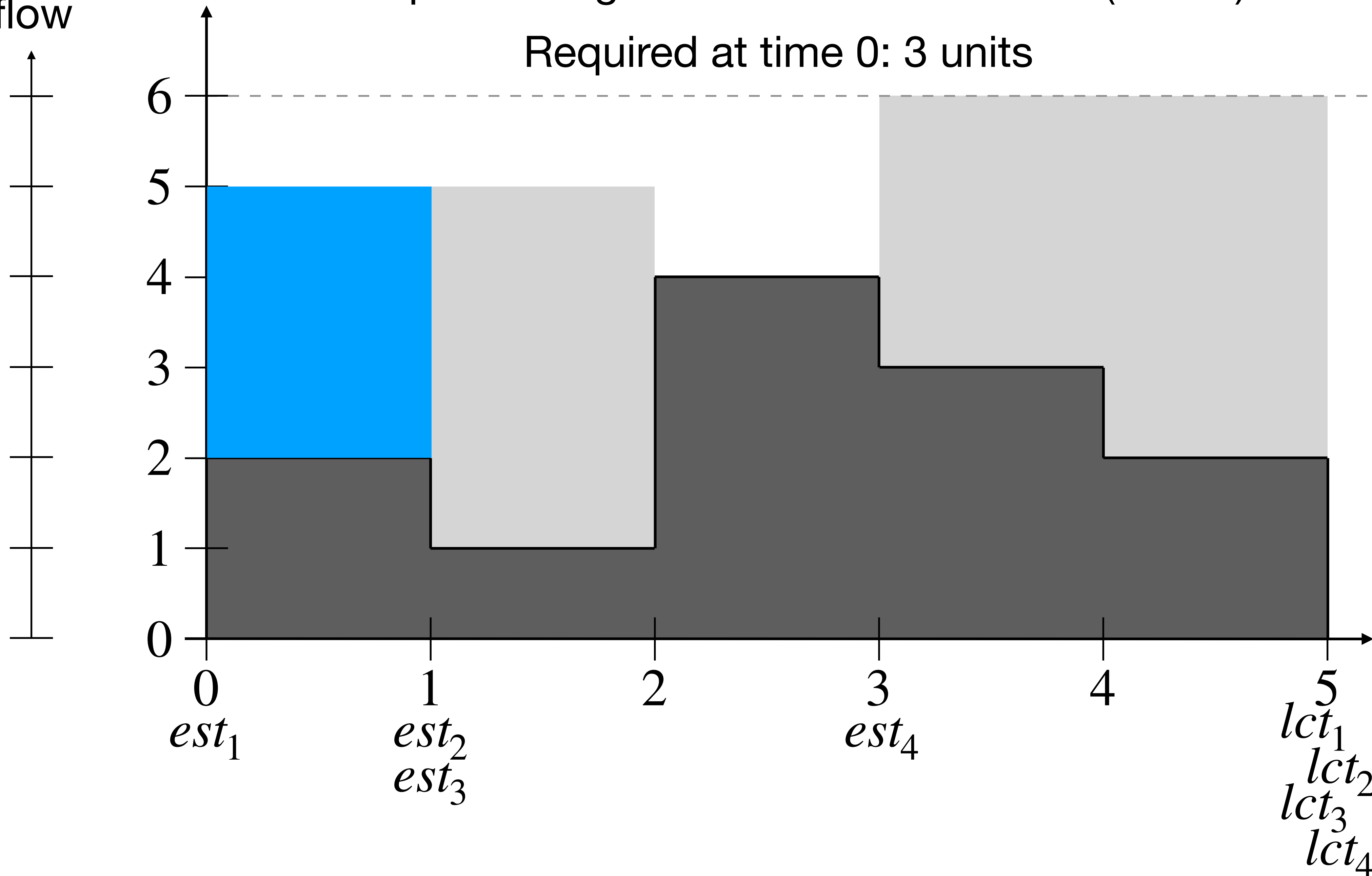
$$d_4 = 1$$

$$c_4 = 2$$

Knapsack-Augmented Overload Check (KAOC)

Required at time 0: 3 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

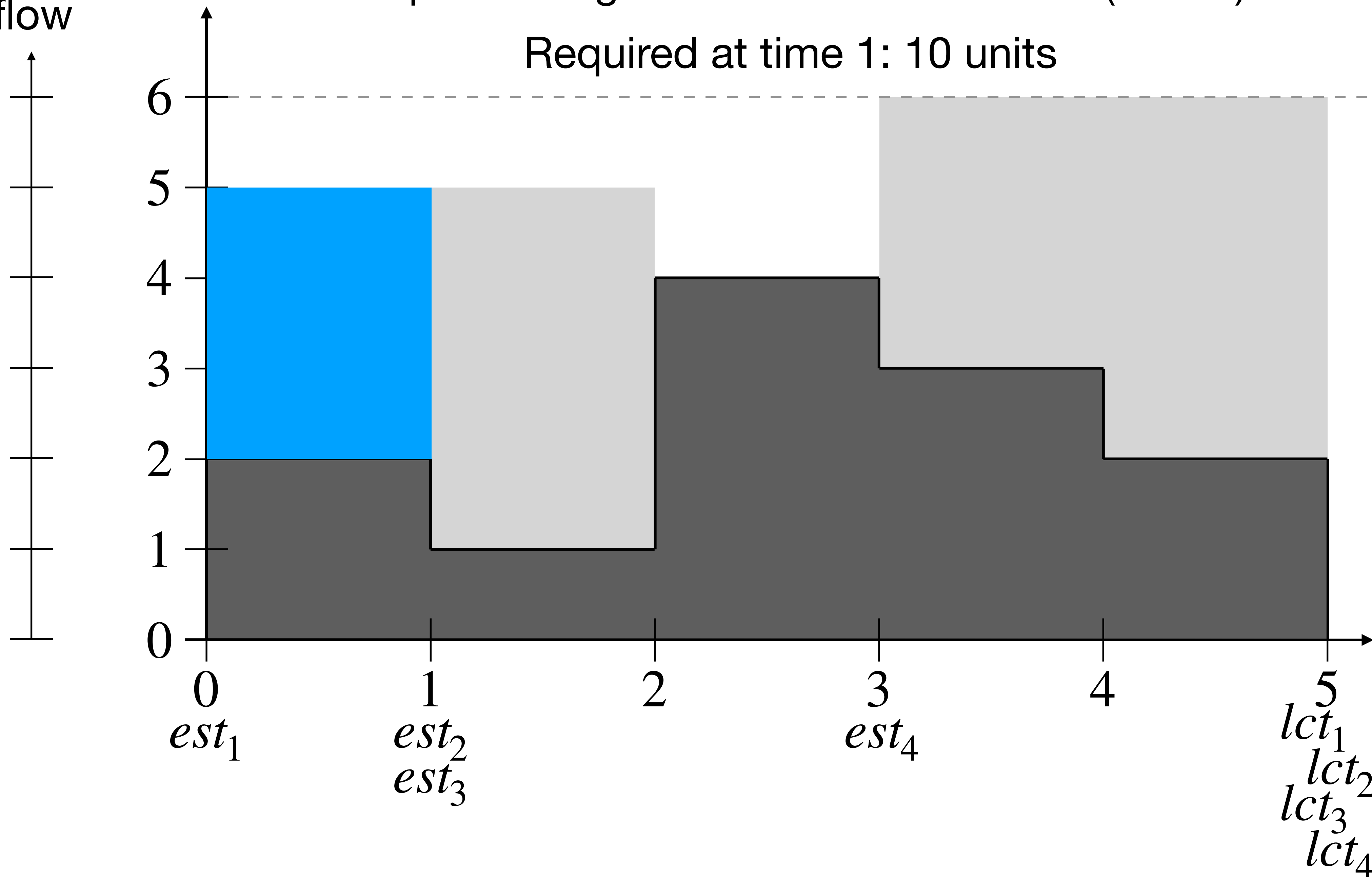
$$d_4 = 1$$

$$c_4 = 2$$

Knapsack-Augmented Overload Check (KAOC)

Required at time 1: 10 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

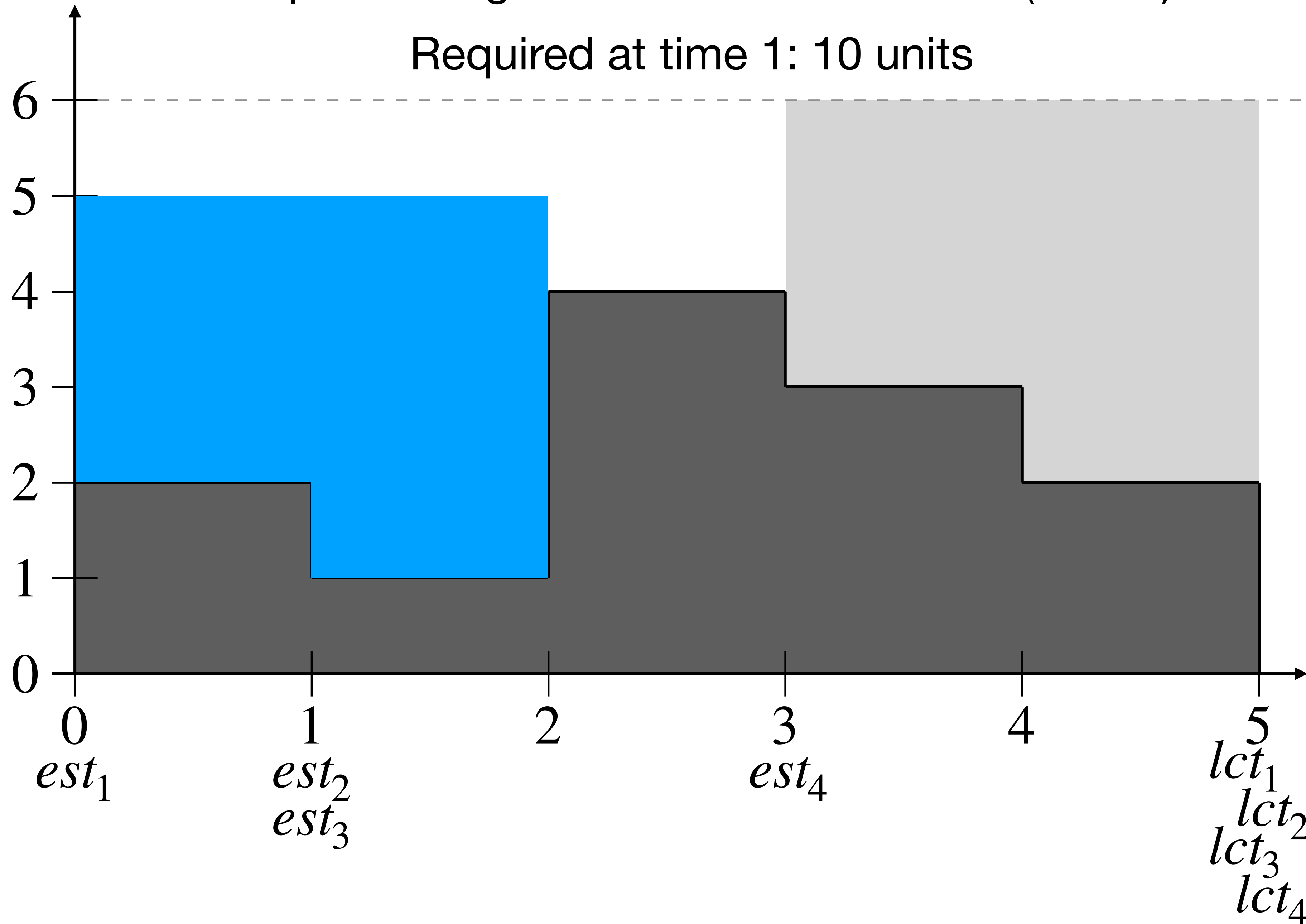
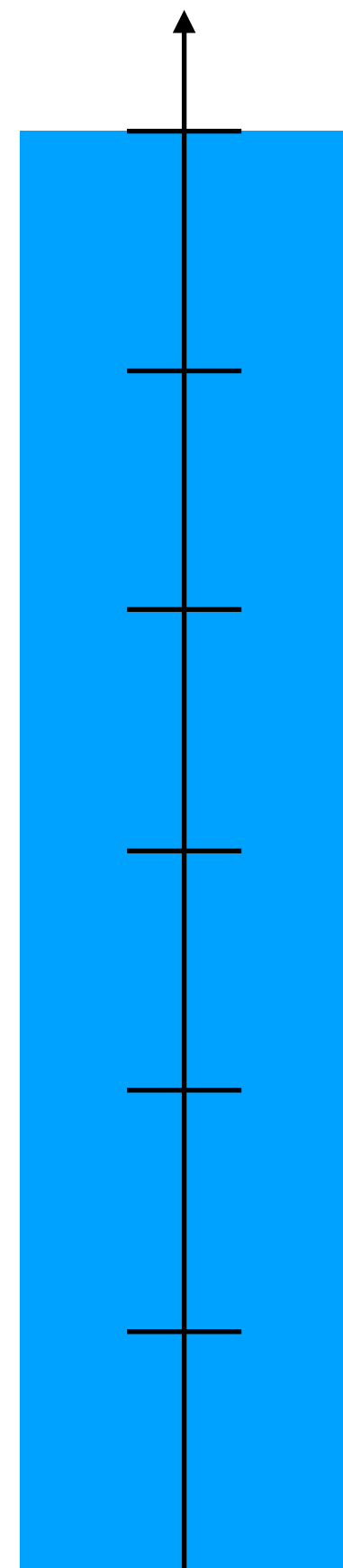
$$d_4 = 1$$

$$c_4 = 2$$

Knapsack-Augmented Overload Check (KAOC)

Required at time 1: 10 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

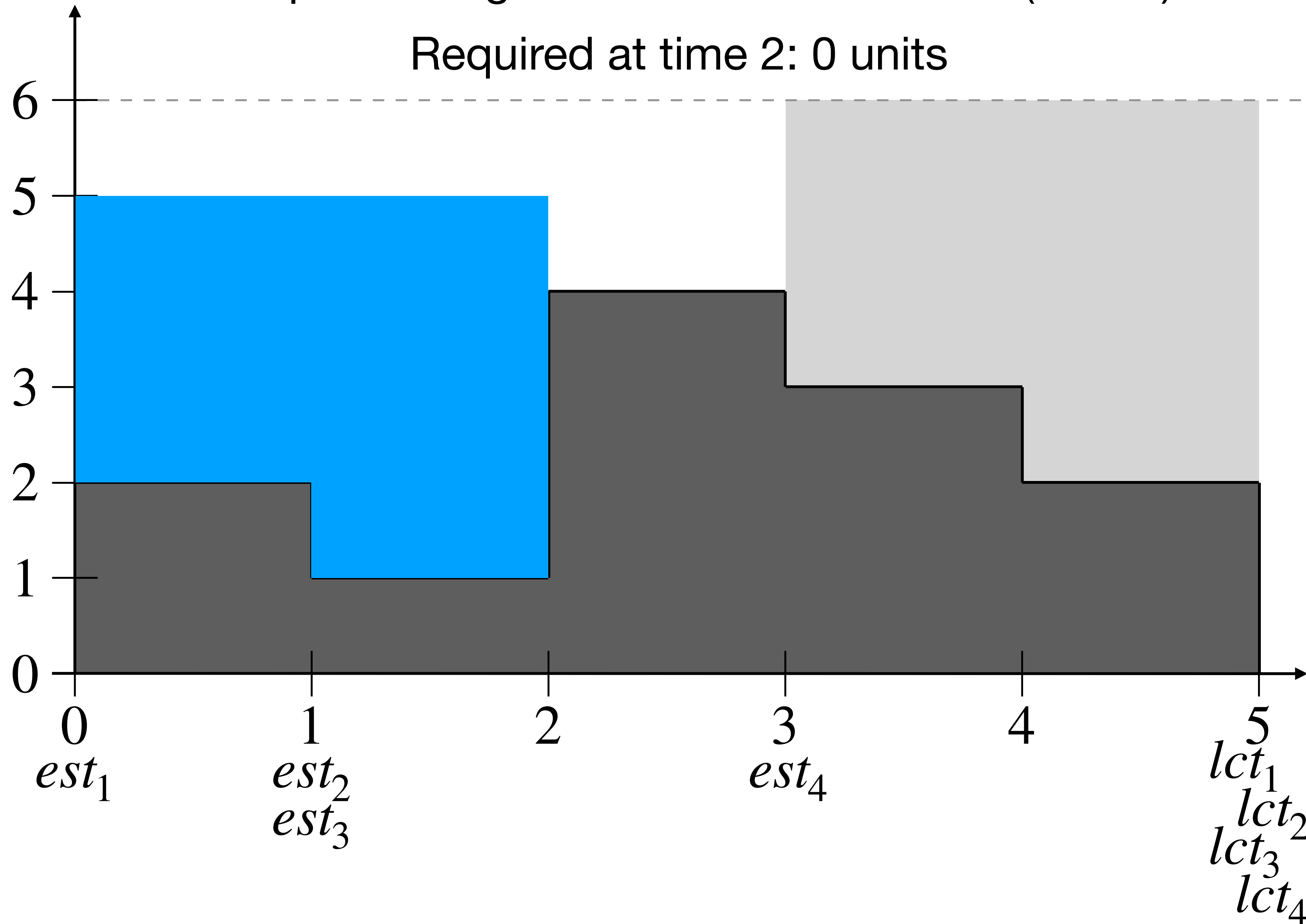
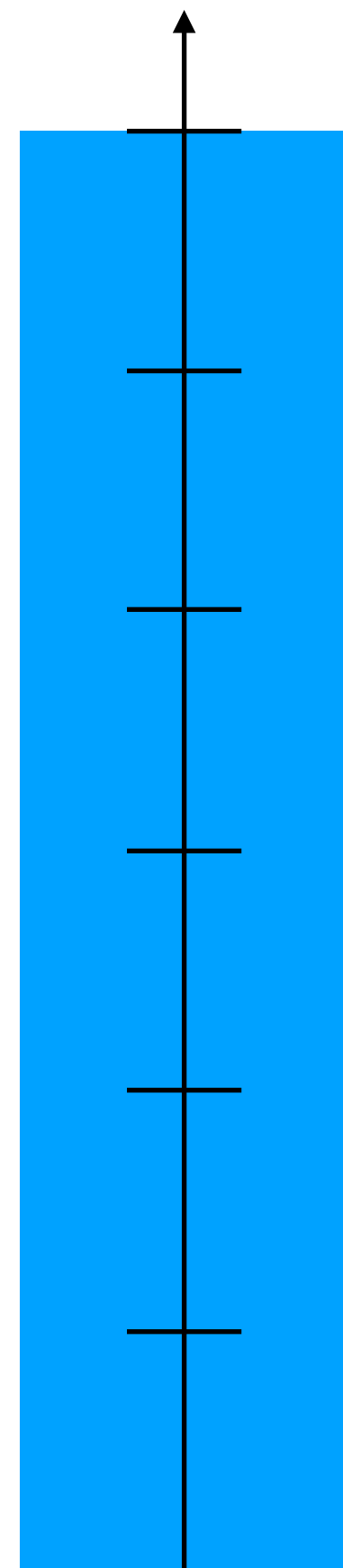
$$d_4 = 1$$

$$c_4 = 2$$

Knapsack-Augmented Overload Check (KAOC)

Required at time 2: 0 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

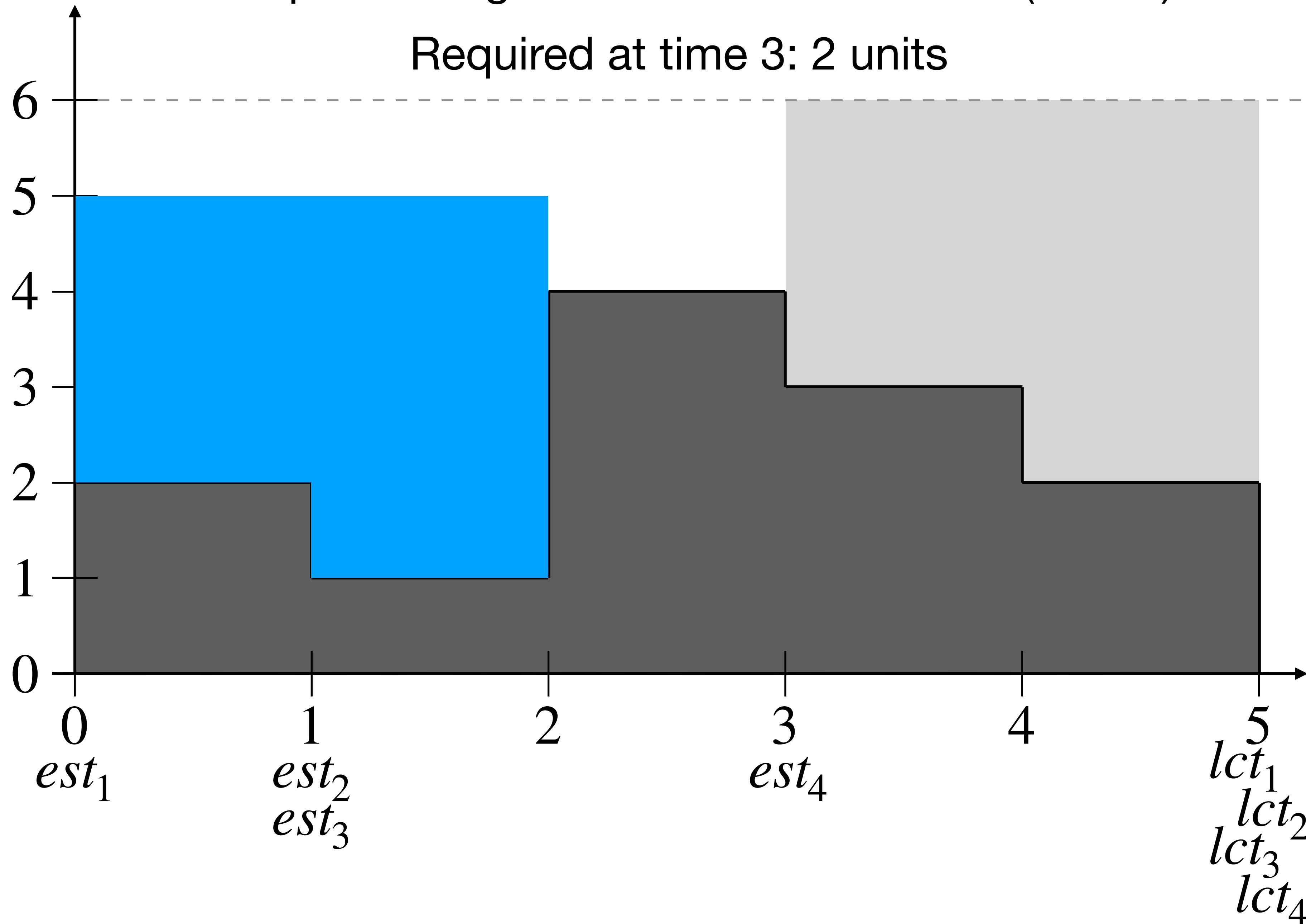
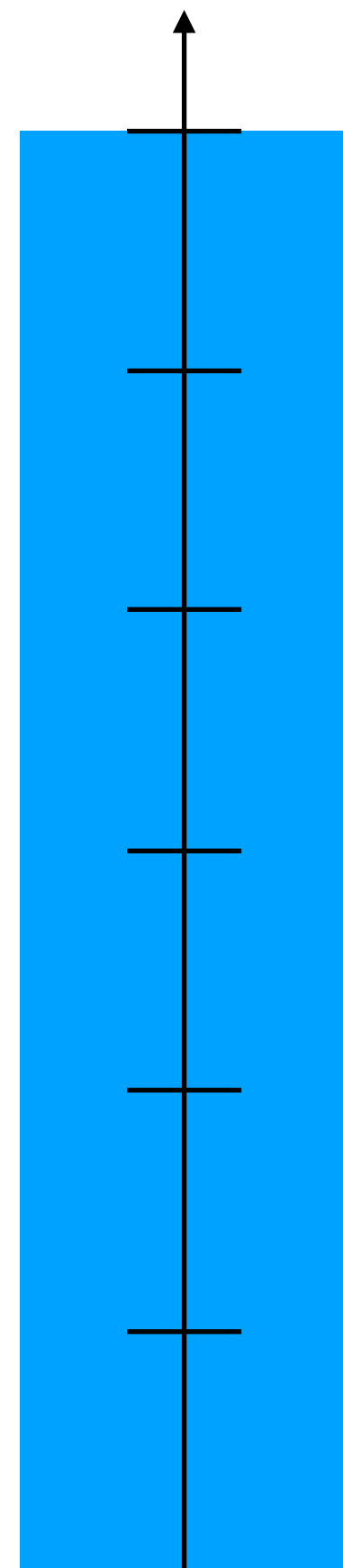
$$d_4 = 1$$

$$c_4 = 2$$

Knapsack-Augmented Overload Check (KAOC)

Required at time 3: 2 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

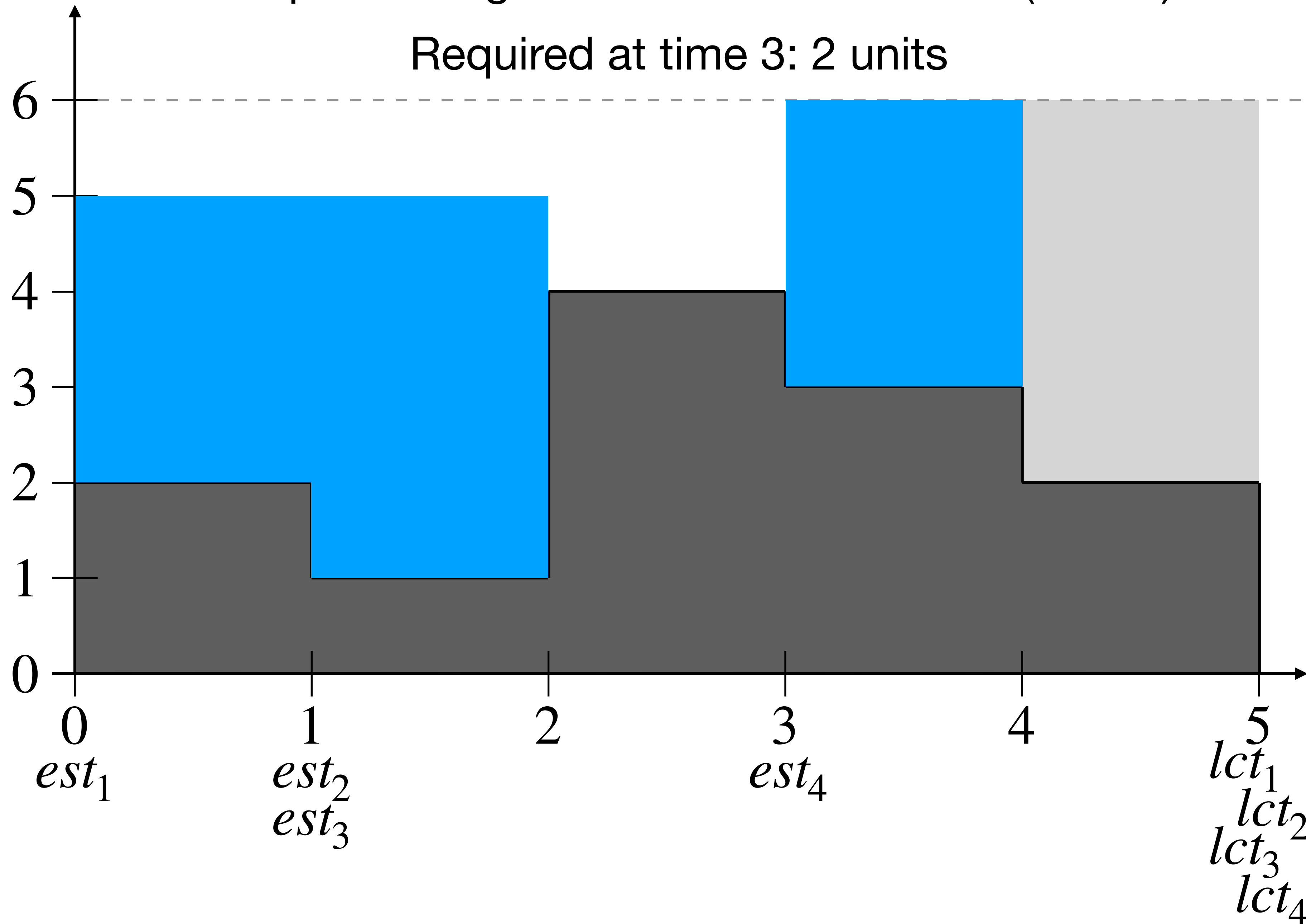
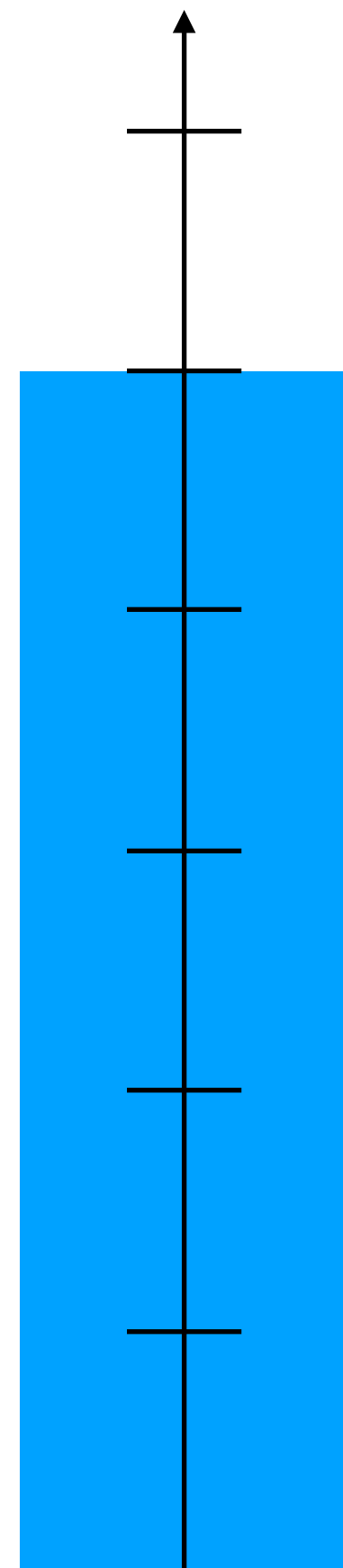
$$d_4 = 1$$

$$c_4 = 2$$

Knapsack-Augmented Overload Check (KAOC)

Required at time 3: 2 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

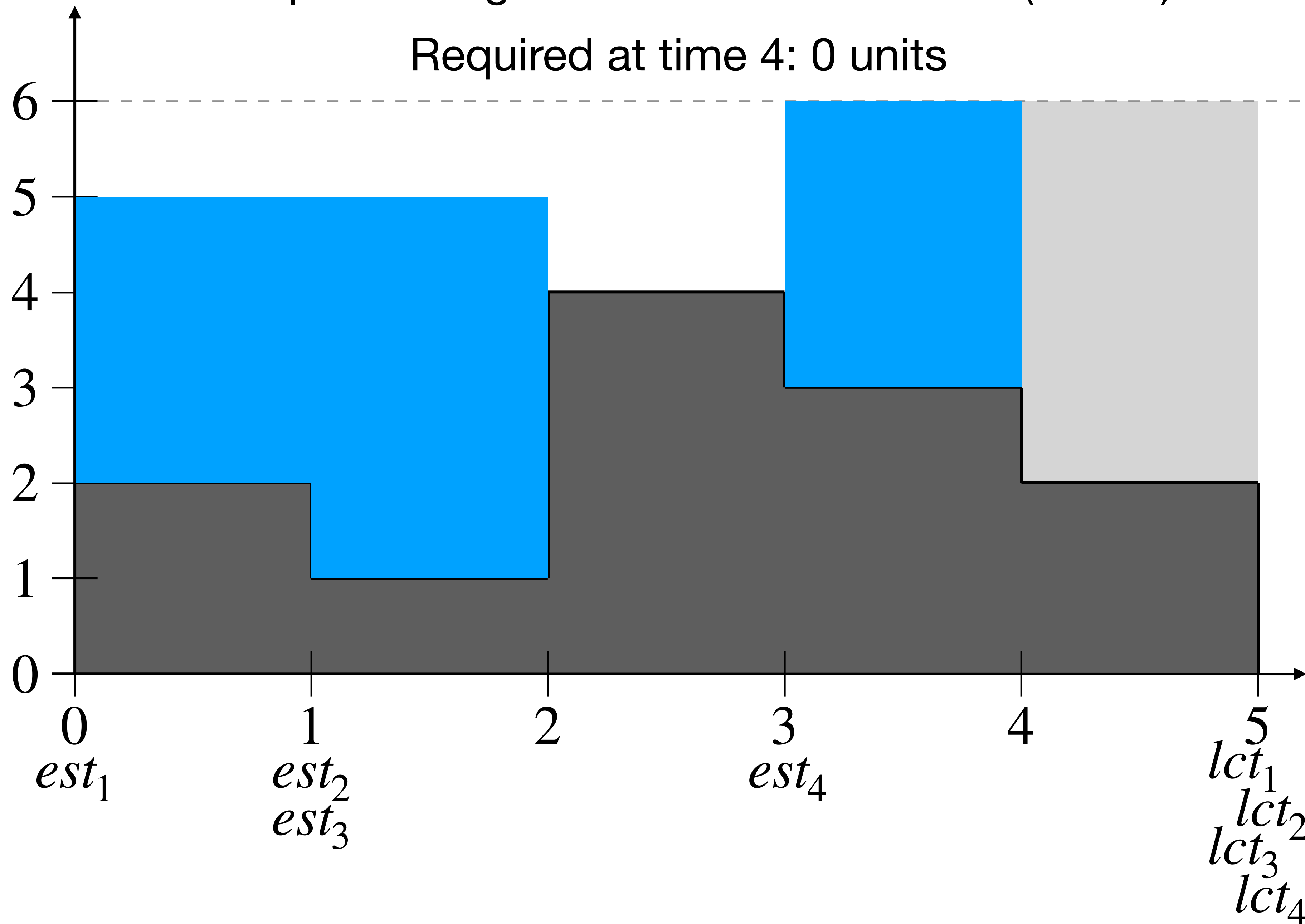
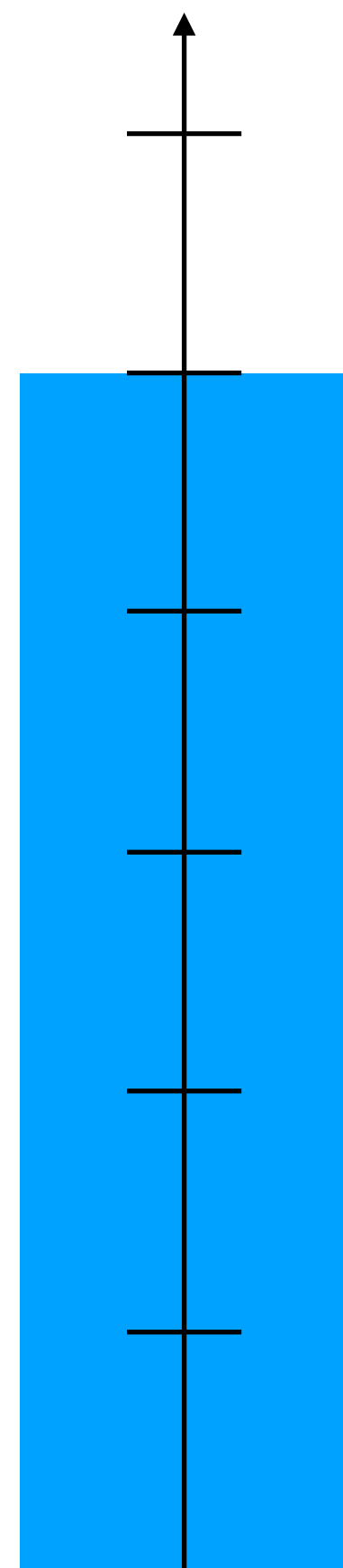
$$d_4 = 1$$

$$c_4 = 2$$

Knapsack-Augmented Overload Check (KAOC)

Required at time 4: 0 units

Overflow



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

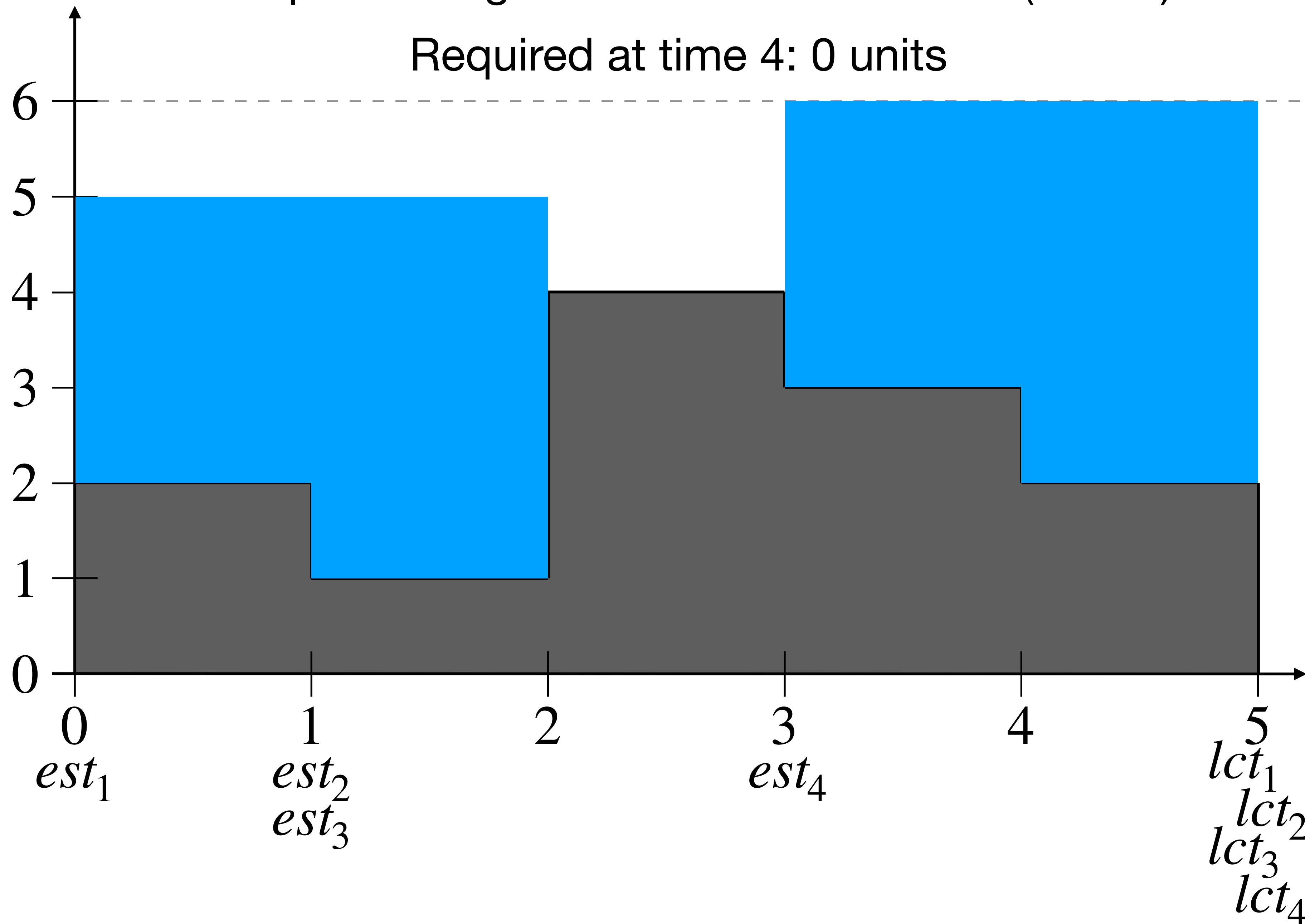
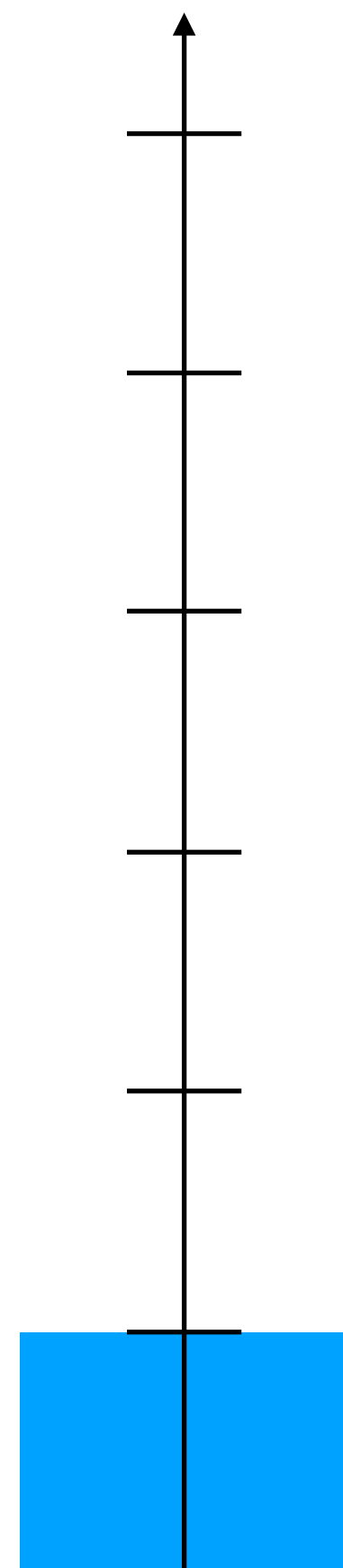
$$c_3 = 4$$

$$d_4 = 1$$

$$c_4 = 2$$

Knapsack-Augmented Overload Check (KAOC)

Required at time 4: 0 units



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

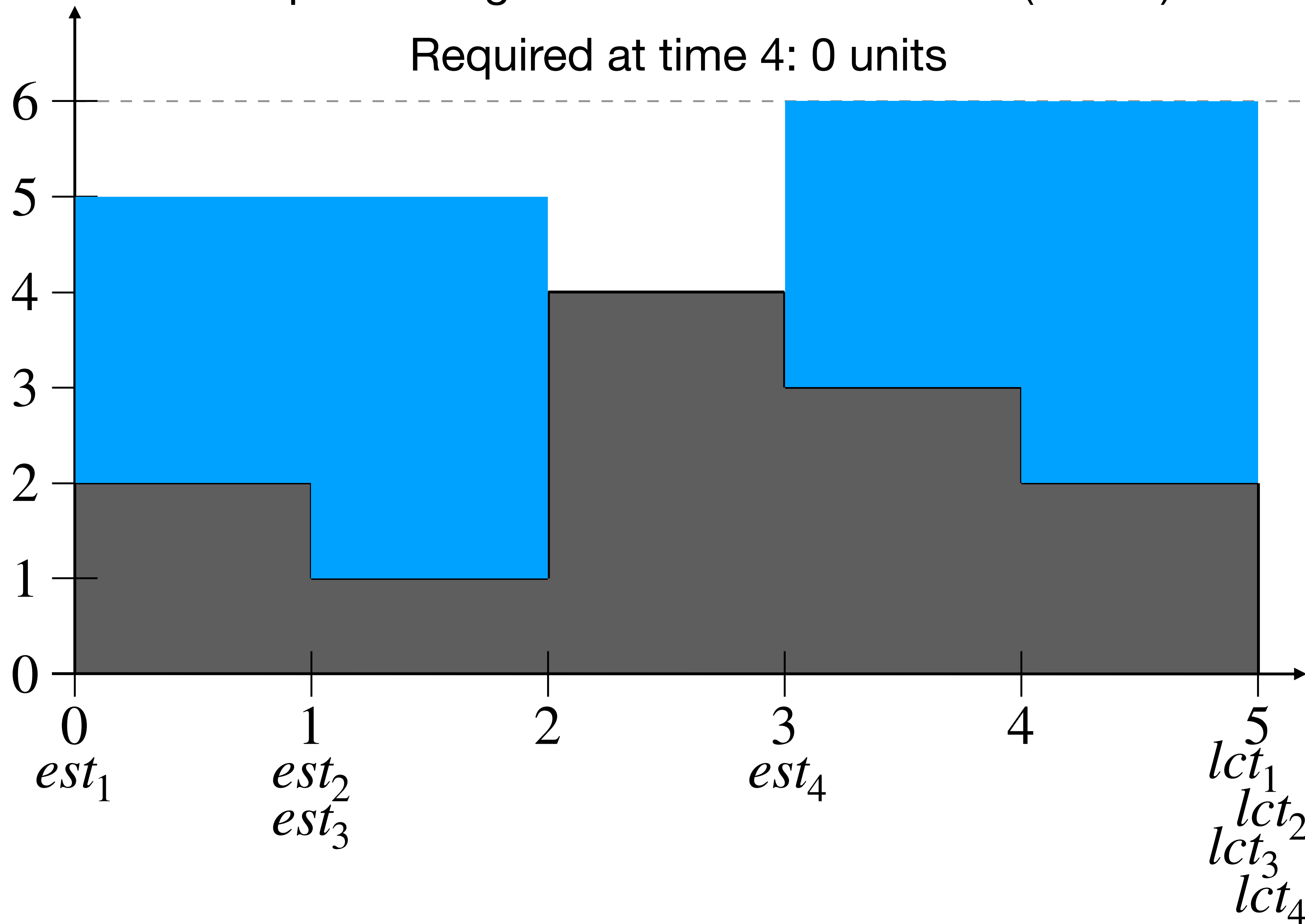
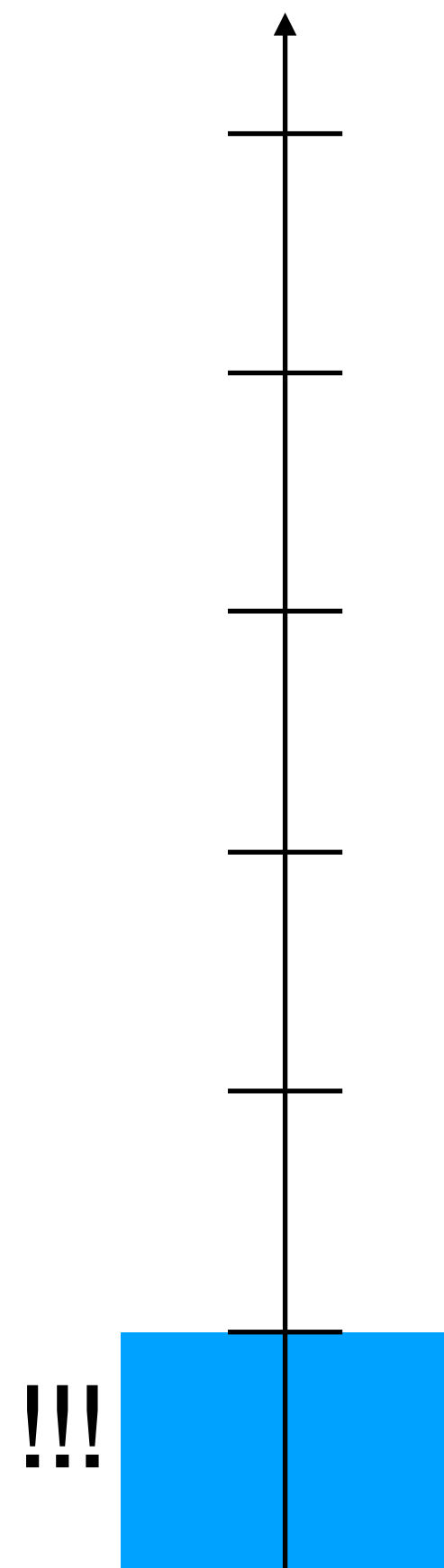
$$c_3 = 4$$

$$d_4 = 1$$

$$c_4 = 2$$

Knapsack-Augmented Overload Check (KAOC)

Required at time 4: 0 units



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

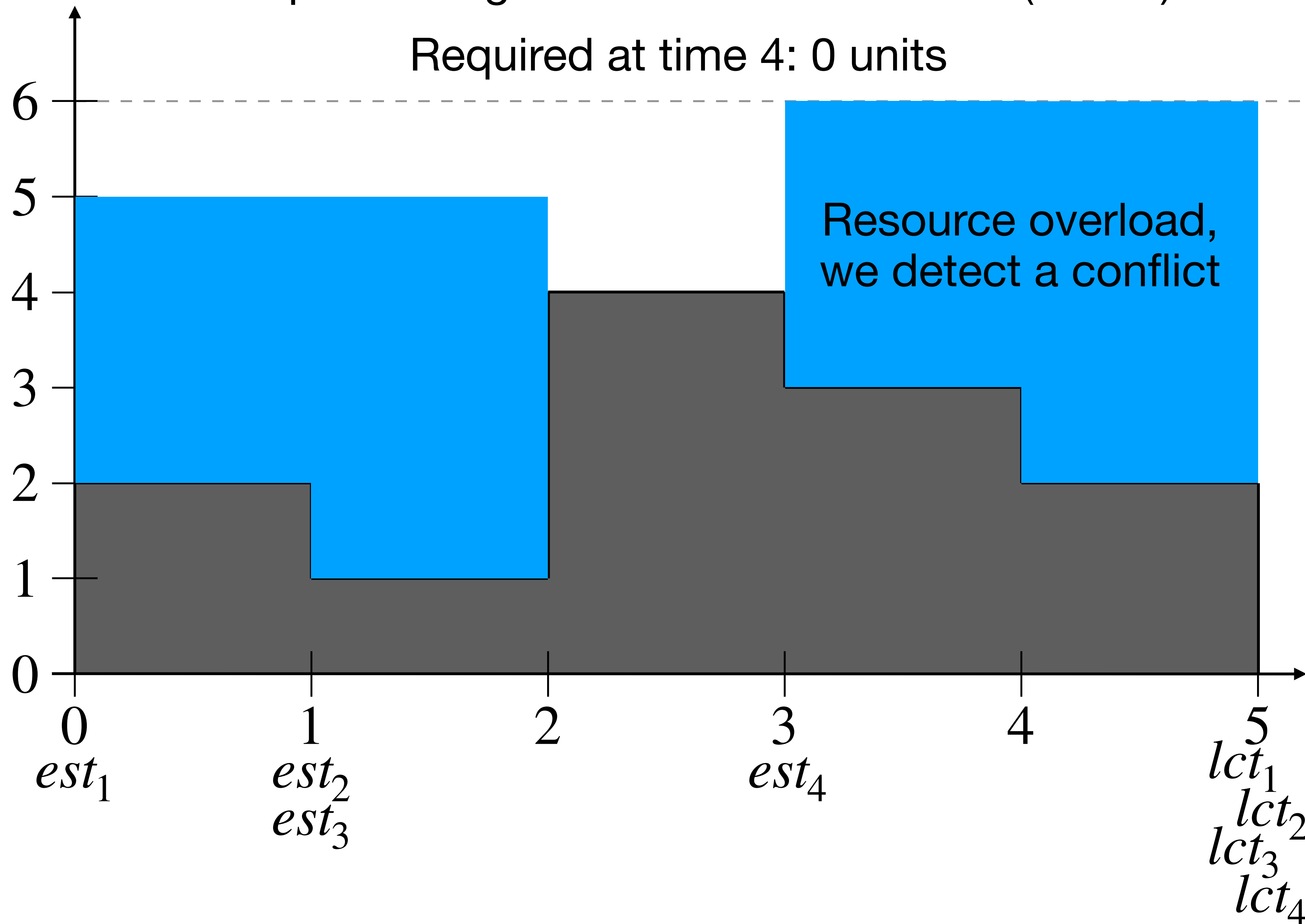
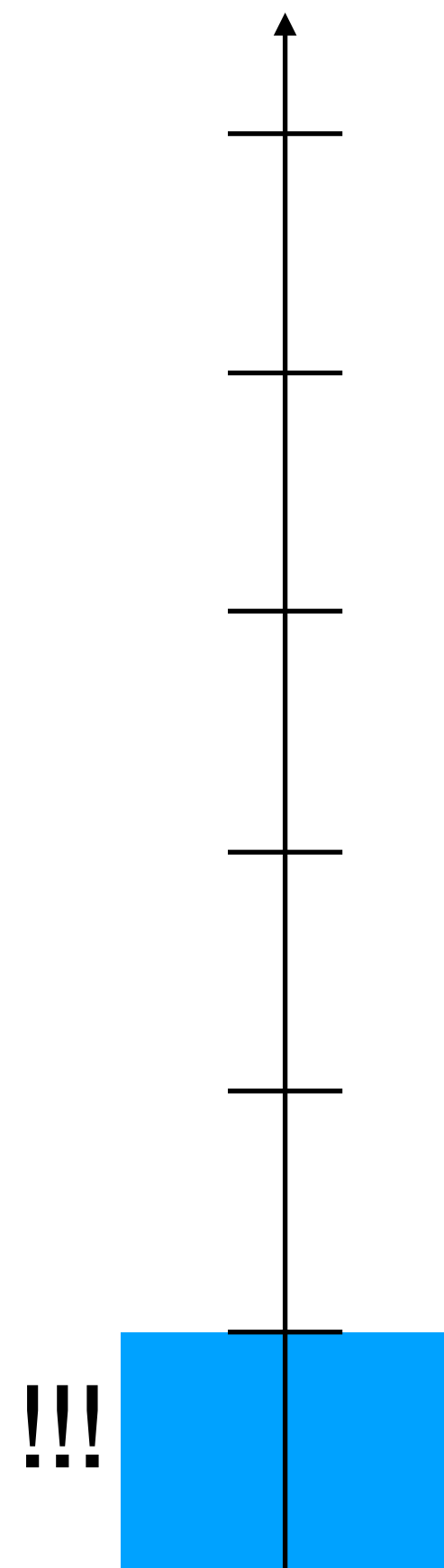
$$c_3 = 4$$

$$d_4 = 1$$

$$c_4 = 2$$

Knapsack-Augmented Overload Check (KAOC)

Required at time 4: 0 units



$$d_1 = 2$$

$$c_1 = 3$$

$$d_2 = 1$$

$$c_2 = 3$$

$$d_3 = 1$$

$$c_3 = 4$$

$$d_4 = 1$$

$$c_4 = 2$$

Solving the Knapsacks

Dynamic Programming - Bitsets

- A bitset of $C+1$ bits can encode all possible resource usage at some time.
- $b[j] = 1$ if and only if some tasks can use j units of resource together.
- Simple disjunction and shifting operations are sufficient to compute the bitset and solve our Knapsack problem.

Propagator Complexity

- TTHEOC [Kameugne et al. 2024] is applied in $\mathcal{O}(n^2)$ time.
- Our KAOOC is applied in $\mathcal{O}(Cn^2)$ time.
- In practical cases, our $\mathcal{O}(Cn^2)$ complexity collapses back to $\mathcal{O}(n^2)$ because C is small and the bitsets use constant time operations on 64-bit words.

Lazy Clause Generation

Creating General Explanations - Greedy algorithm

- Reducing the set of explained tasks
 - Remove the effect of the task and see if the conflict still holds.
- Generalizing the bounds
 - Modify the effect of the task to simulate it starting earlier/later, and see if the conflict still holds.

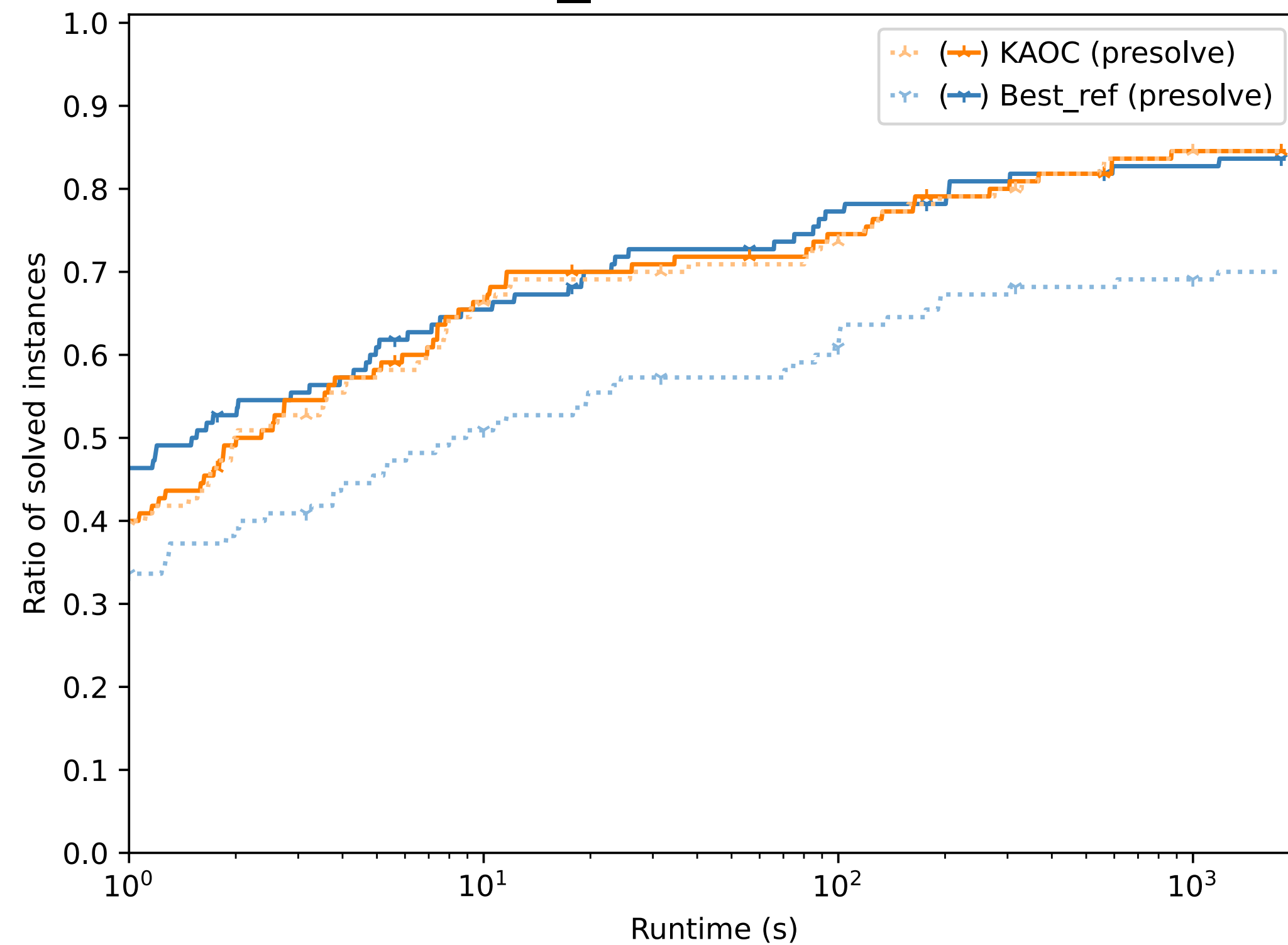
Pre-Solving

Coefficient Strengthening

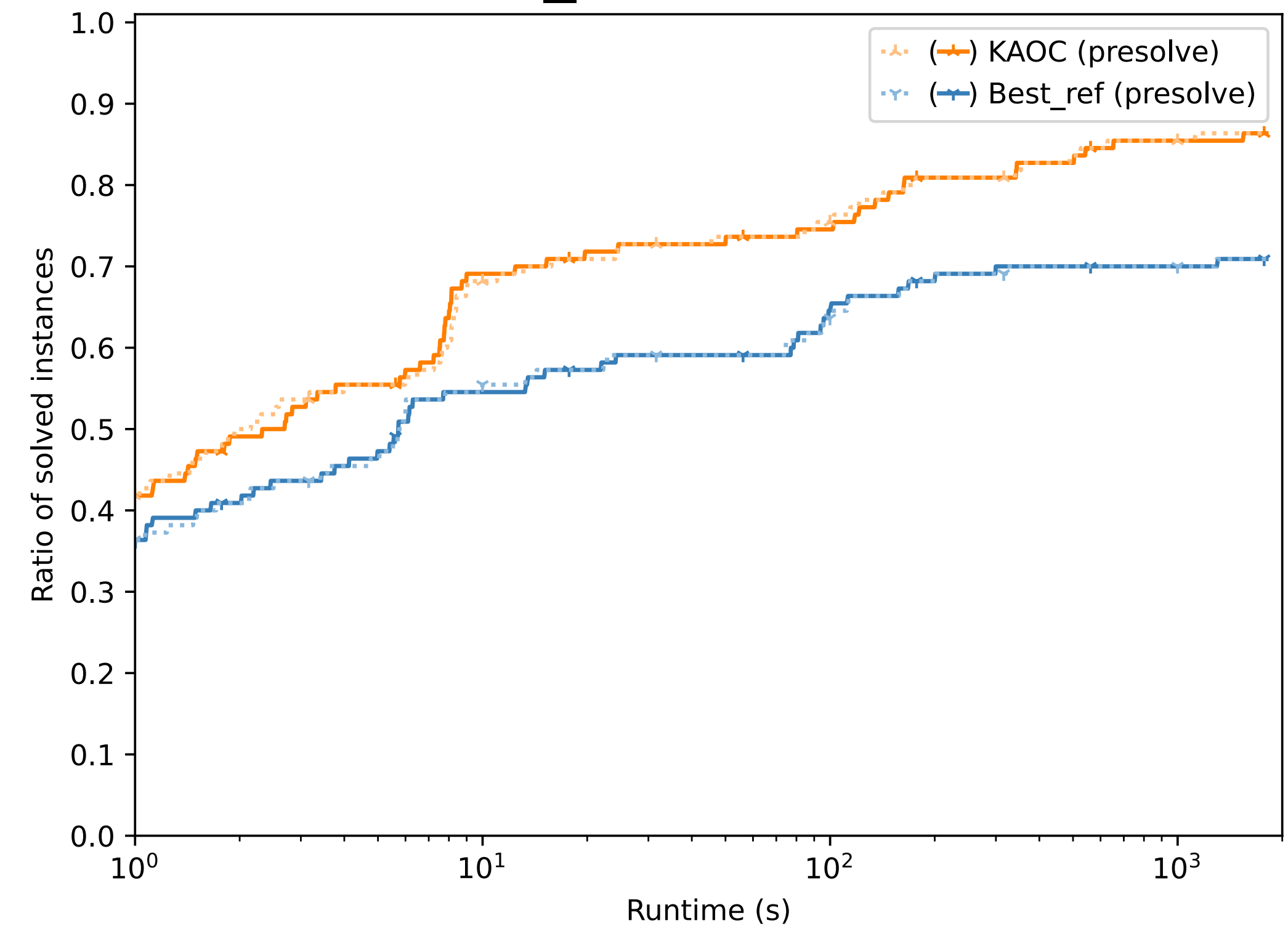
- Some pre-solving rules were proposed to help make energy based filtering more effective [Schulz 2013].
- Applied once to modify the instance, before the solver starts solving it.

Results

Ratio of solved instances from benchmarks
Pack/Pack_d in function of time



Ratio of solved instances from benchmarks
Pack+1/Pack_d+1 in function of time



What Can Follow

- The Overload Check is the basis for Edge-Finding filtering.
- Our enhancement should apply to Edge-Finding.

What Can Follow

- The Overload Check is the basis for Edge-Finding filtering.
- Our enhancement should apply to Edge-Finding.

Thank you for your attention!