Cumulative Scheduling

Peter Stuckey

Resources

- Unary resources are unique
- Often we have multiple identical copies of a resource
  - bulldozers
  - workers (of equal capability)
  - operating theaters
  - airplane gates
- How do we model multiple identical resources?
  - assume task `t` uses `res[t]`
  - assume a limit `L` of resource at all times

Modeling Resources: Time Decomposition

- The use of the resource at each time `i` is less that the limit `L`

```
forall(i in TIME)
       (sum(t in TASK)
           (bool2int(s[t]<=i /\ s[t]+d[t]>i)
            * res[t])
        <= L);
```

- Note the expression
  - `s[t]<=i /\ s[t]+d[t]>i`
  - represents whether task `t` runs at time `i`
- Problem: size is card(TASK)*card(TIME)
  - many time periods TIME

Modeling Resources: Task Decomposition

- Note we can only only overload a resource when a task starts (otherwise no increase)
- Alternate model: only check start times

```
forall(t2 in TASK)
      (sum(t in TASK)
          (bool2int(s[t]<=s[t2]
                   /\ s[t]+d[t]> s[t2])
           * res[t])
       <= L);
```

- Can we do improve this?

Modeling Resources: Task Decomposition

- Better model: we know t2 runs at time s[t2]

```
forall(t2 in TASK)
      (sum(t in TASK where t != t2)
          (bool2int(s[t]<=s[t2]
                    /\ s[t]+d[t]> s[t2])
           * res[t])
       + res[t2] <= L);
```

- Advantage: much smaller than time decomposition card(TASK)^2
- Problem: not as much information to the solver

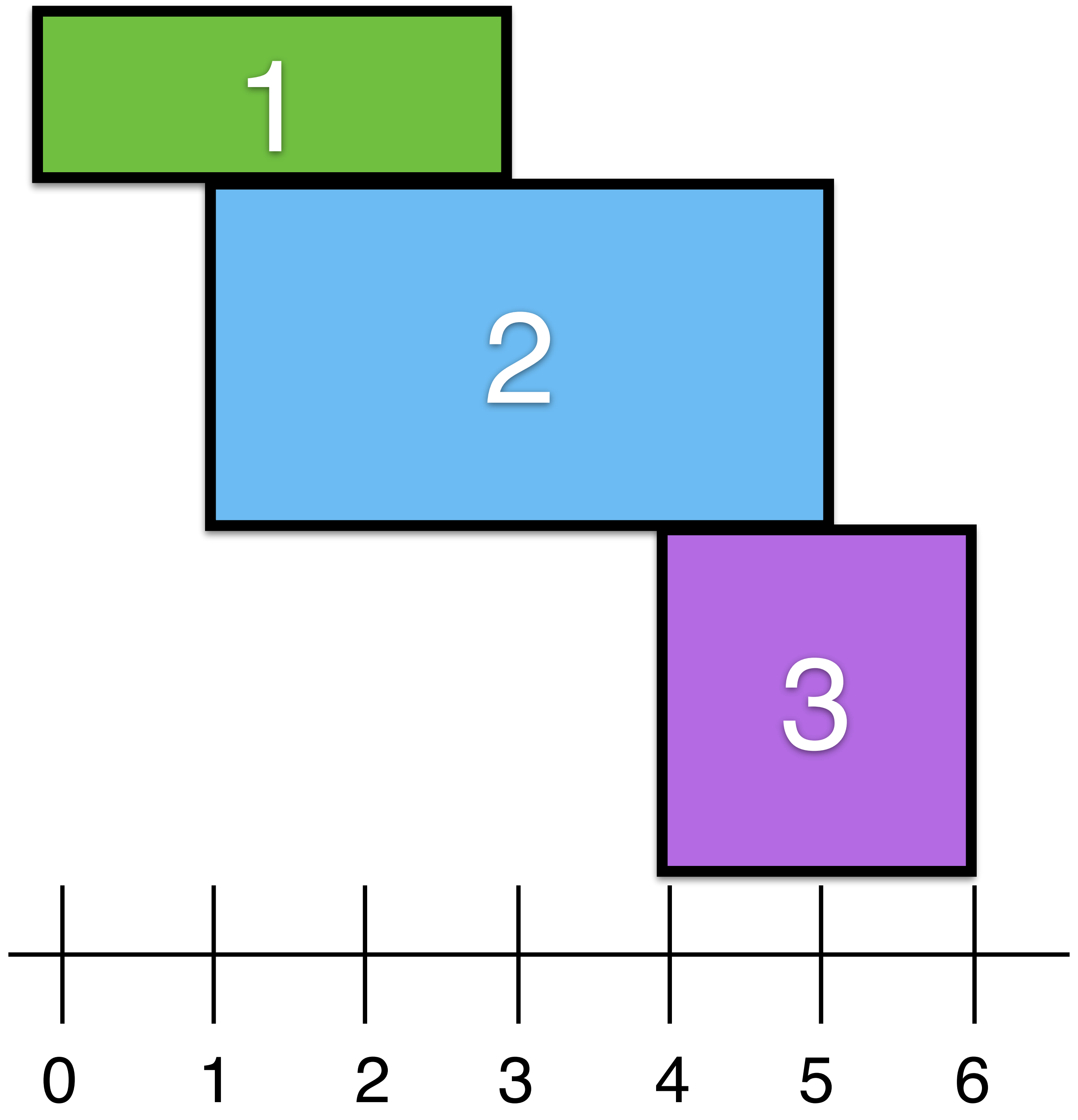
Cumulative

- The cumulative global constraint captures exactly a resource constraint
  - cumulative(<start time array>,<duration array>,
  - <resource usage array>, <limit>)
    - ensure no more than the limit of the resource is used at any time during the execution of tasks

```
predicate cumulative(array[int] of var int:s,
                     array[int] of var int:d,
                     array[int] of var int:r,
                     var int: L);
```

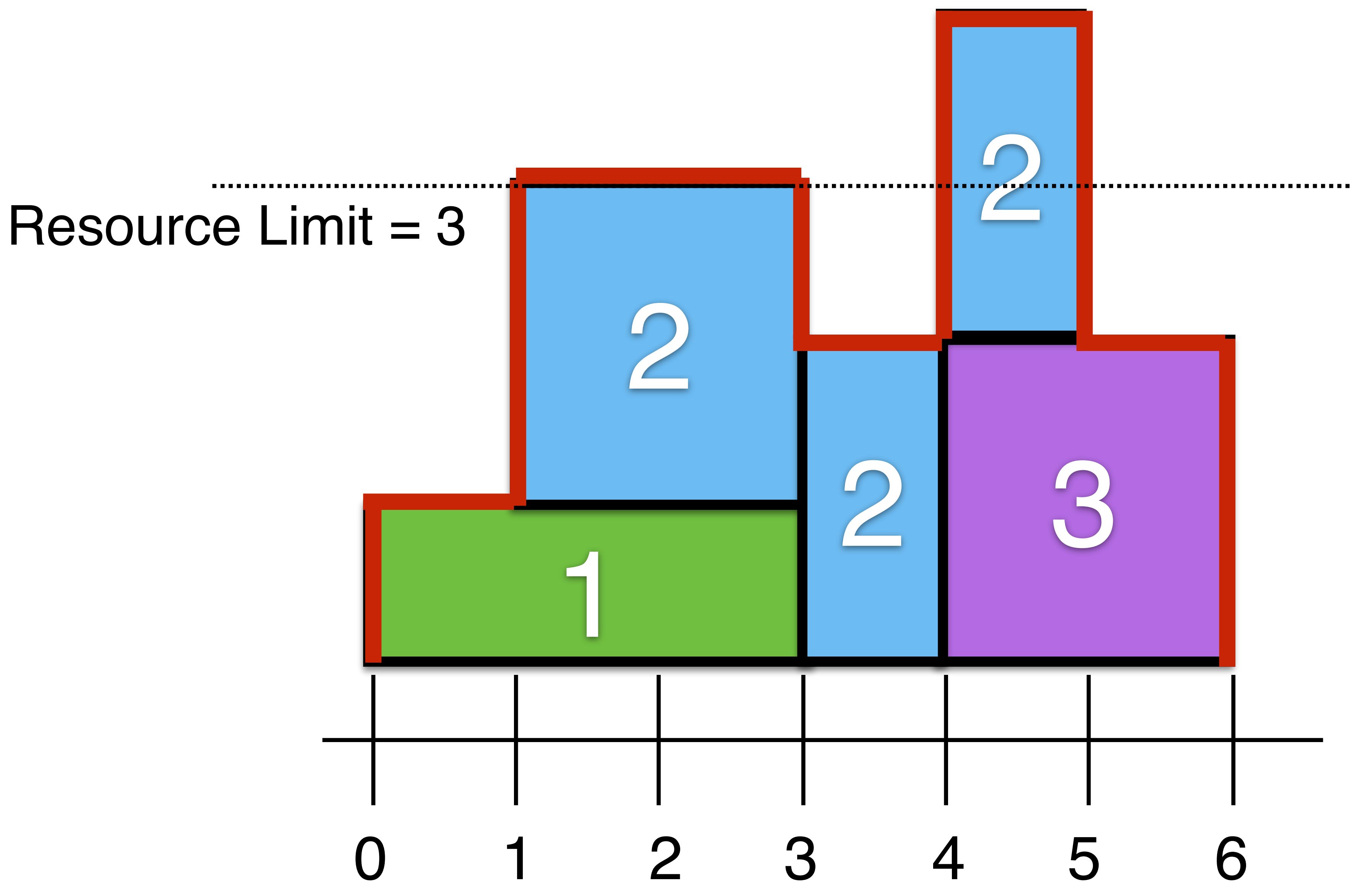
Visualizing Cumulative

- A task t is a box of length d[t] and height r[t] starting at time s[t]
  - e.g. cumulative([0,1,4],[3,4,2],[1,2,2],3)



Visualizing Cumulative

- A task t is a box of length d[t] and height r[t] starting at time s[t]
  - e.g. cumulative([0,1,4],[3,4,2],[1,2,2],3)
- They are not really boxes
- Timetable (red skyline) shows the usage



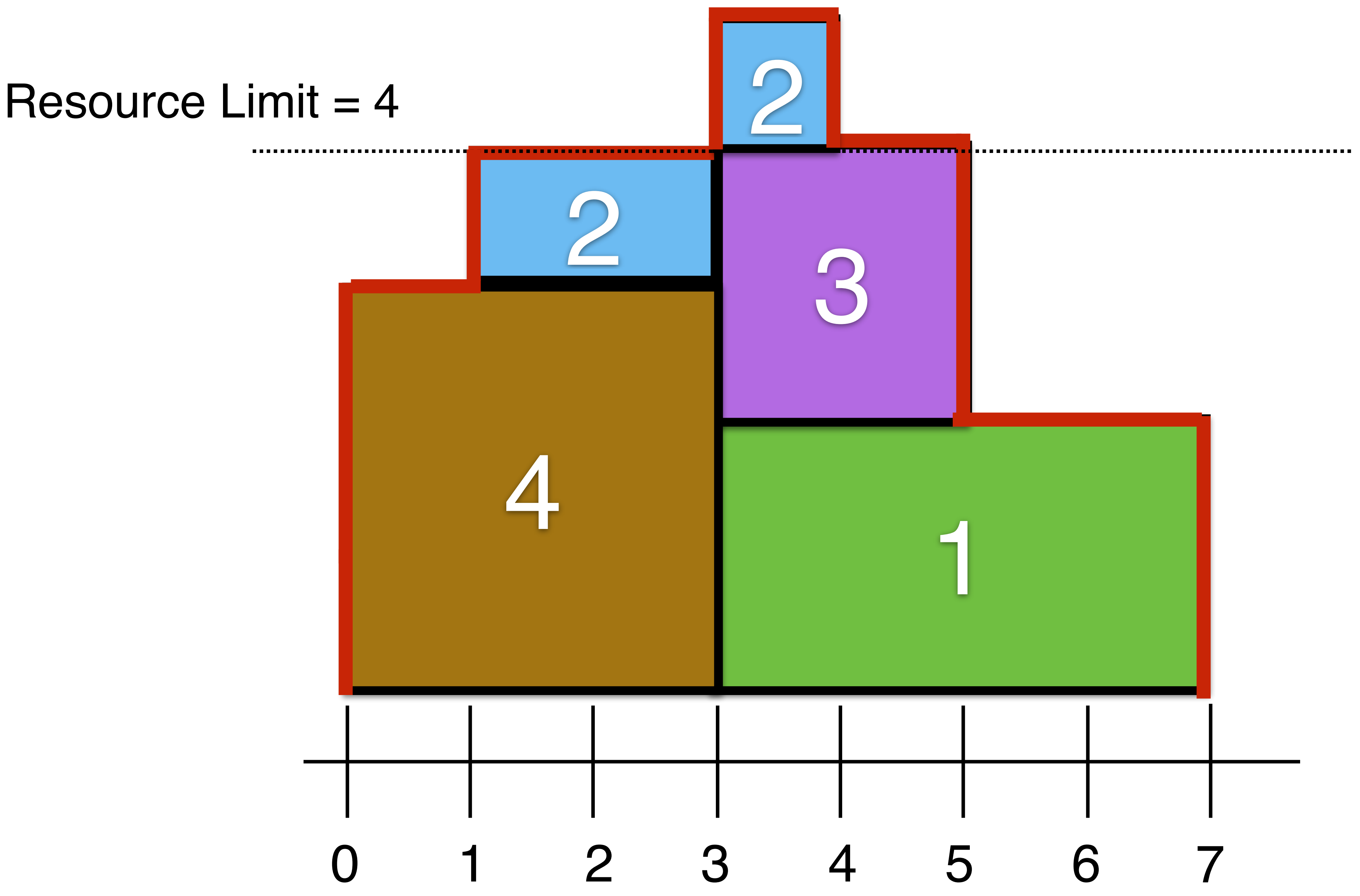
Cumulative Example

- Does the constraint below hold
  - cumulative([3,1,3,0],[4,3,2,3],[2,1,2,3],4)

- Given the cumulative constraint below does it have a solution
  - start time possibilities are given as ranges
  - cumulative([0..3,0..3,2..3,0..4],[4,3,2,3],[2,1,2,3],4)

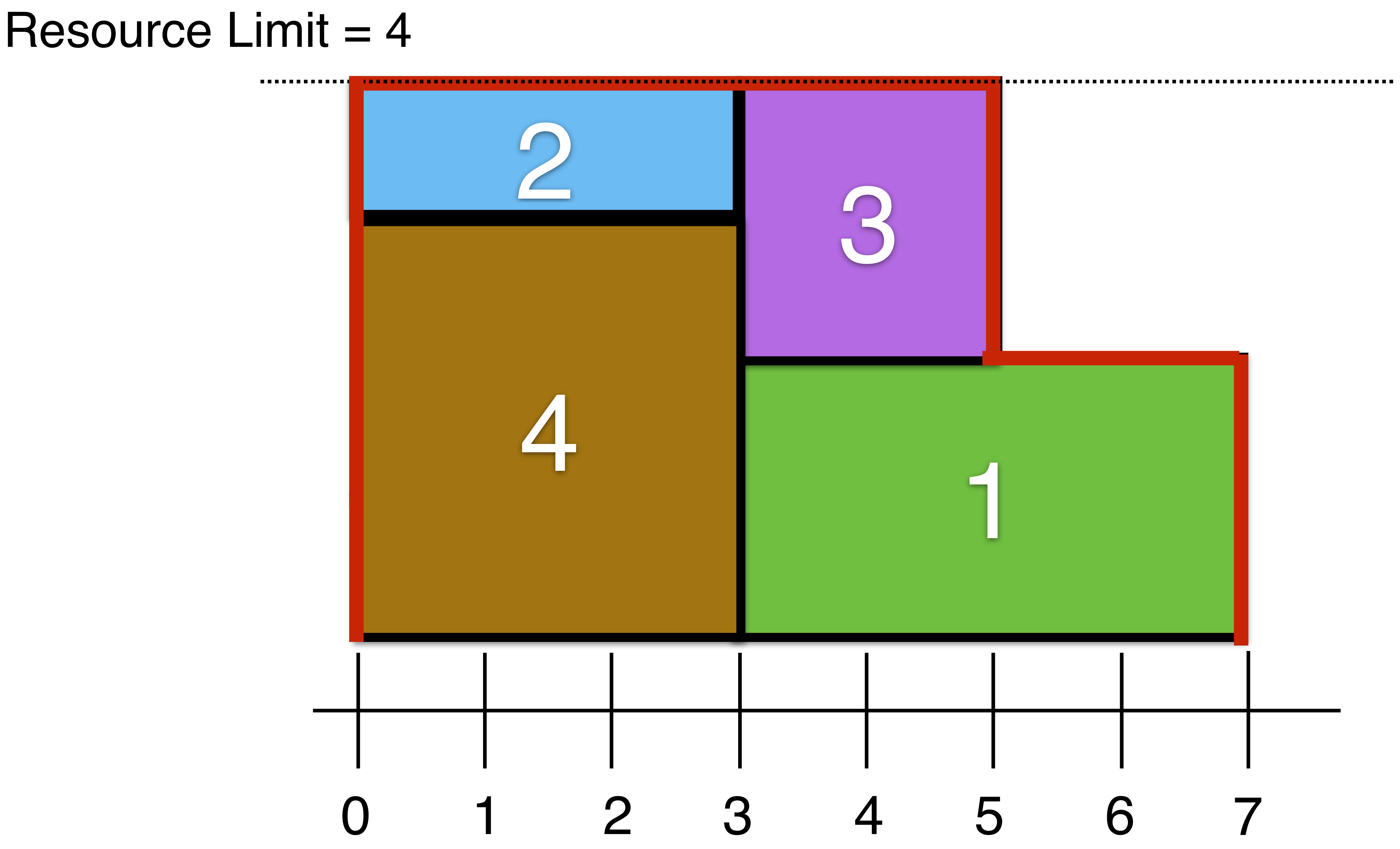
Visualizing Cumulative

- Does the constraint below hold
  - cumulative([3,1,3,0],[4,3,2,3],[2,1,2,3],4)



Visualizing Cumulative

- Given the cumulative constraint below does it have a solution
  - start time possibilities are given as ranges
  - cumulative([0..3,0..3,2..3,0..4],[4,3,2,3],[2,1,2,3],4)



Cumulative Propagators

- There is a lot of research in how to propagate cumulative constraints
  - timetable propagation
    - equivalent to the time decomposition
    - but faster than the task decomposition
  - edge finding
    - reasoning about time intervals rather than single times
  - energy based reasoning
    - more inference than edge finding, but slower
  - TTEF time table edge finding
    - a combination of timetable with some energy based reasoning
    - state of the art

Resource Constrained Project Scheduling (RCPSP)

- Given tasks $t \in TASK$
- Given precedences p ∈ PREC
  - pred[p] precedes succ[p]
- Assume resources $r \in RESOURCES$
- Each task t needs $req[r,t]$ resources during its execution
- We have a limit $L[r]$ for each resource
- Find the shortest schedule to run every task!

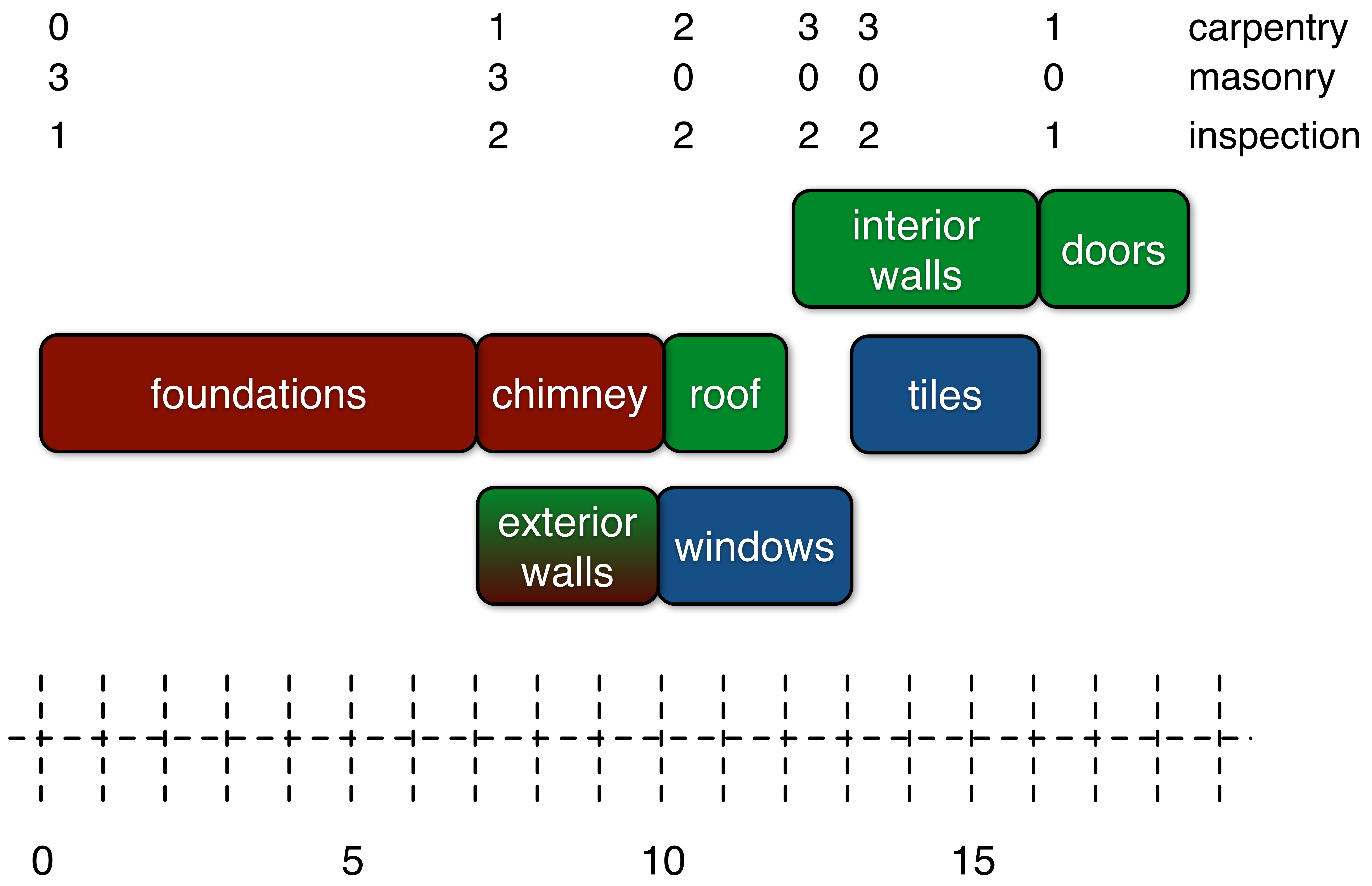
- Possibly the most studied scheduling problem

RCPSP House Building

- A more detailed version of the house building scheduling problem
- Three resources
  - carpentry
  - masonry
  - inspection

resource	f	iw	ew	c	r	d	t	w	limit
carpentry	0	3	1	0	2	1	0	0	3
masonry	3	0	2	1	0	0	0	0	3
inspection	1	1	1	1	1	1	1	1	2

RCPSP House Building Solution




```
makespan        f   iw  ew   c    r    d    t    w
18          = [0, 12, 7, 7, 10, 16, 13, 10]
```

RCPSP Data

- Data

```
int: n;                         % number of tasks
set of int: TASK = 1..n;
array[TASK] of int: d;          % duration
int: m;                         % no. of resources
set of int: RESOURCE = 1..m;
array[RESOURCE] of int: L;      % resource limit
array[RESOURCE,TASK] of int: res; % usage
int: l;                         % no. of precedences
set of int: PREC = 1..l;
array[PREC,1..2] of TASK: pre; i
      % predecessor/successor pairs
int: maxt; % maximum time
set of int: TIME = 0..maxt;
```

RCPSP House Data

- Example Data

```
n = 8;
d = [7,4,3,3,2,2,3,3];
m = 3;
L = [3,3,2];
res = [| 0,3,1,0,2,1,0,0
       | 3,0,2,1,0,0,0,0
       | 1,1,1,1,1,1,1,1 |];
l = 8;
pre = [| 1, 2
       | 1, 3
       | 1, 4
       | 3, 5
       | 3, 8
       | 2, 6
       | 4, 7
       | 5, 7 |];
```

RCPSP Model

- Decisions

```
array[TASK] of var TIME: s; % start time
```

- Constraints

```
forall(p in PREC)
        (s[pre[p,1]]+d[pre[p,1]] <= s[pre[p,2]]);

forall(r in RESOURCE)
   (cumulative(s,d,[res[r,t]|t in TASK],L[r]));
```

- Objective

```
solve minimize max(t in TASK)(s[t] + d[t]);
```

Overview

- Renewable capacitated resources
  - a resource capacity available over the schedule
- Time decomposition:
  - check resource usage at each time
- Task decomposition
  - check resource usage as each task starts
- `cumulative` global constraint

- RCPSP: a core scheduling problem

EOF