Cumulative Scheduling

Peter Stuckey & Guido Tack

MONASH
University

Resource Constraint Project Scheduling (RCPSP)

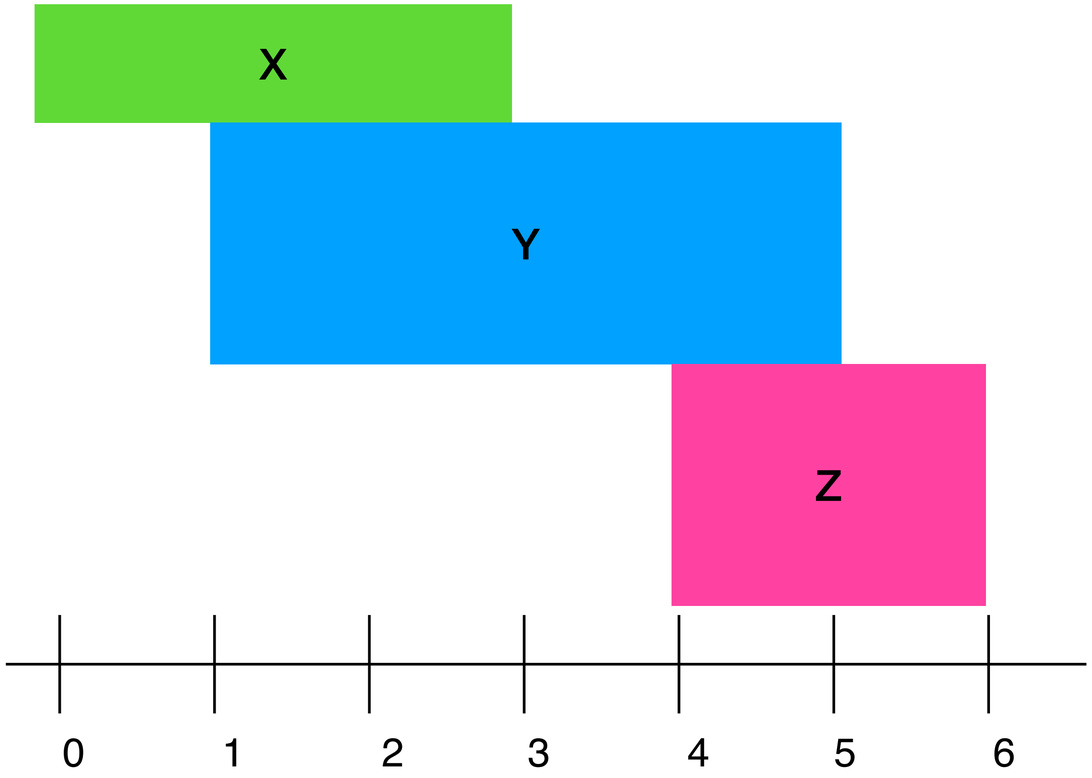
- Given tasks `t in TASK`
- Given precedences `p in PREC`
  - `pre[p,1]` precedes `pre[p,2]`
- Assume resources `r in RESOURCES`
- Each task `t` needs `res[r,t]` resources during its execution
- We have a limit `L[r]` for each resource
- Find the shortest schedule to run every task!
- Possibly the **most studied scheduling problem**

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**

Visualizing Resource Requirements

- A task `t` is a box of *length* `duration[t]` and *height* `res[t]`, starting at time `start[t]`



Modeling Resources: Time Decomposition

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

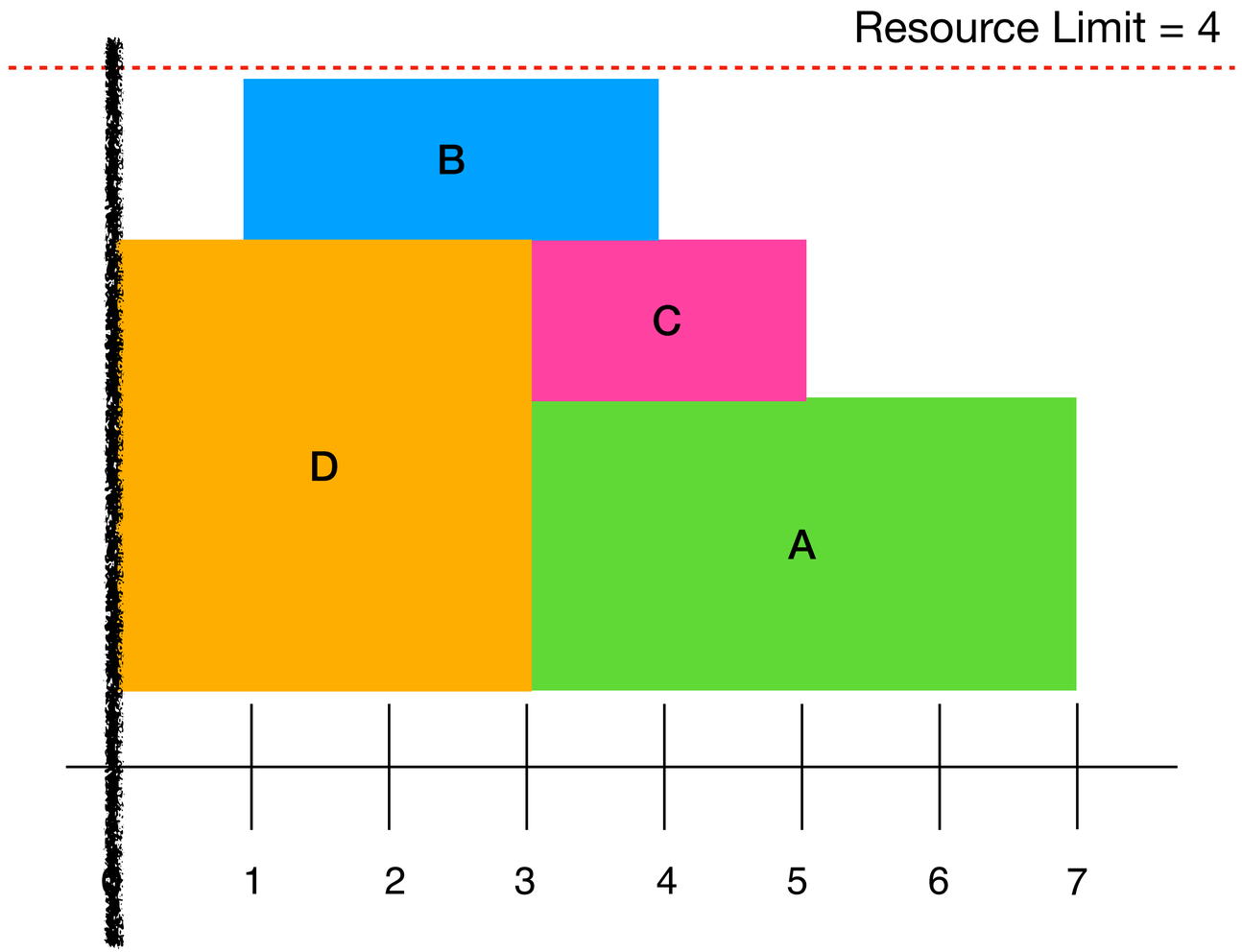
```
forall(i in TIME)(sum(t in TASK)(
   (start[t]<=i /\ start[t]+duration[t]>i)
    * res[t]
) <= L);
```

- Note the expression

```
start[t]<=i /\ start[t]+duration[t]>i
```

  represents whether task `t` runs at time `i`

Modeling Resources: Time Decomposition



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)
      ((start[t]<=i /\ start[t]+duration[t]>i)
        * res[t]) <= L);
```

- **Problem**: size is `card(TASK)*card(TIME)`
  - many time periods in `TIME`

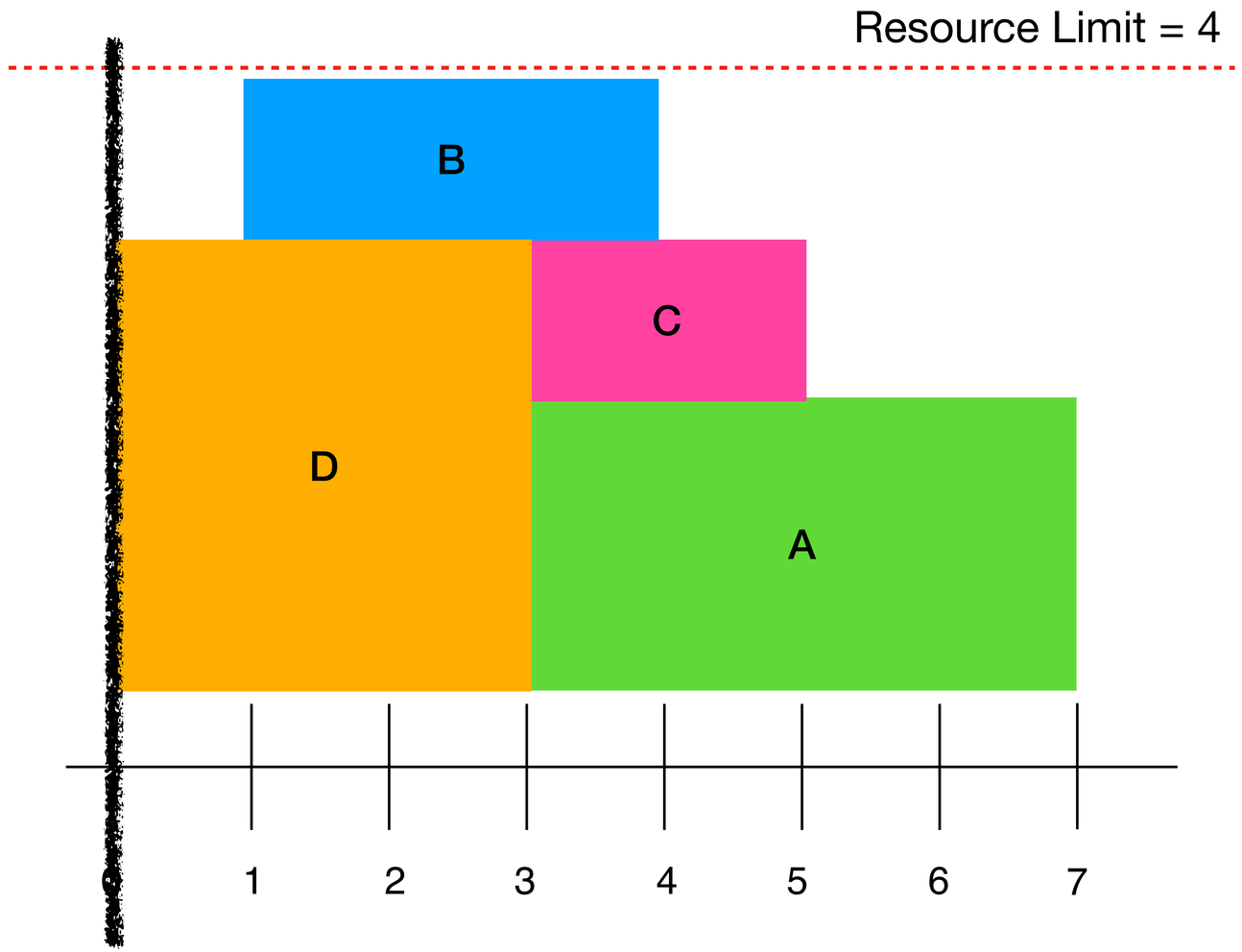
Modeling Resources: Task Decomposition

- Note we can overload a resource **only when** a task starts (otherwise no increase)

- Alternate model: only check start times

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

Modeling Resources: Task Decomposition



Modeling Resources: Task Decomposition

- A more explicit formulation

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

- Comparison with time decomposition
  - **Advantage**: much smaller than time decomposition with `card(TASK)`^{2}
  - **Problem**: not as much information (fewer constraints) to the solver

Cumulative Global Constraint

- The **cumulative** global constraint captures exactly a resource constraint

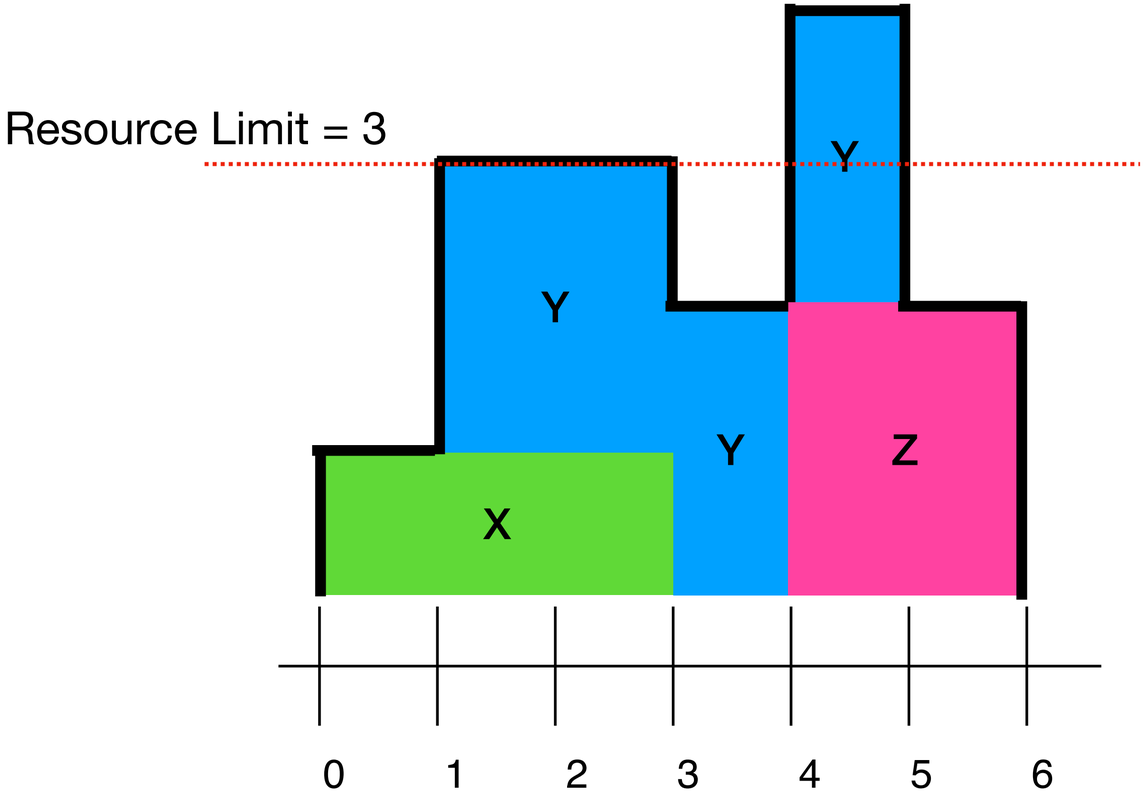
```
cumulative(<start time array>, <duration array>,
           <resource usage array>, <limit>)
```

  - 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

- They are not really boxes
- Timetable (black skyline) shows the usage



Summary

- 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

Summary

- 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