Modeling Time (Scheduling)

Peter Stuckey

Overview

- Scheduling problems
  - are one of the most common uses of CP in the real world
- Basic Scheduling
  - only precedence constraints
- Job Shop Scheduling
  - disjunctive global constraint
- Resource Constraint Project Scheduling
  - cumulative global constraint
- Sequence Dependent Setup Times
  - modeling order

Scheduling

- In discrete optimization
  - time is modeled by integers (not continuous)

- Time variables
  - tend to have VERY large ranges
    - e.g. start times on the minute for a 7 day schedule
  - typically only care about
    - earliest time, or
    - latest time
  - when reasoning (not about all possible times)

Basic Scheduling

- Scheduling is an important class of discrete optimisation problems

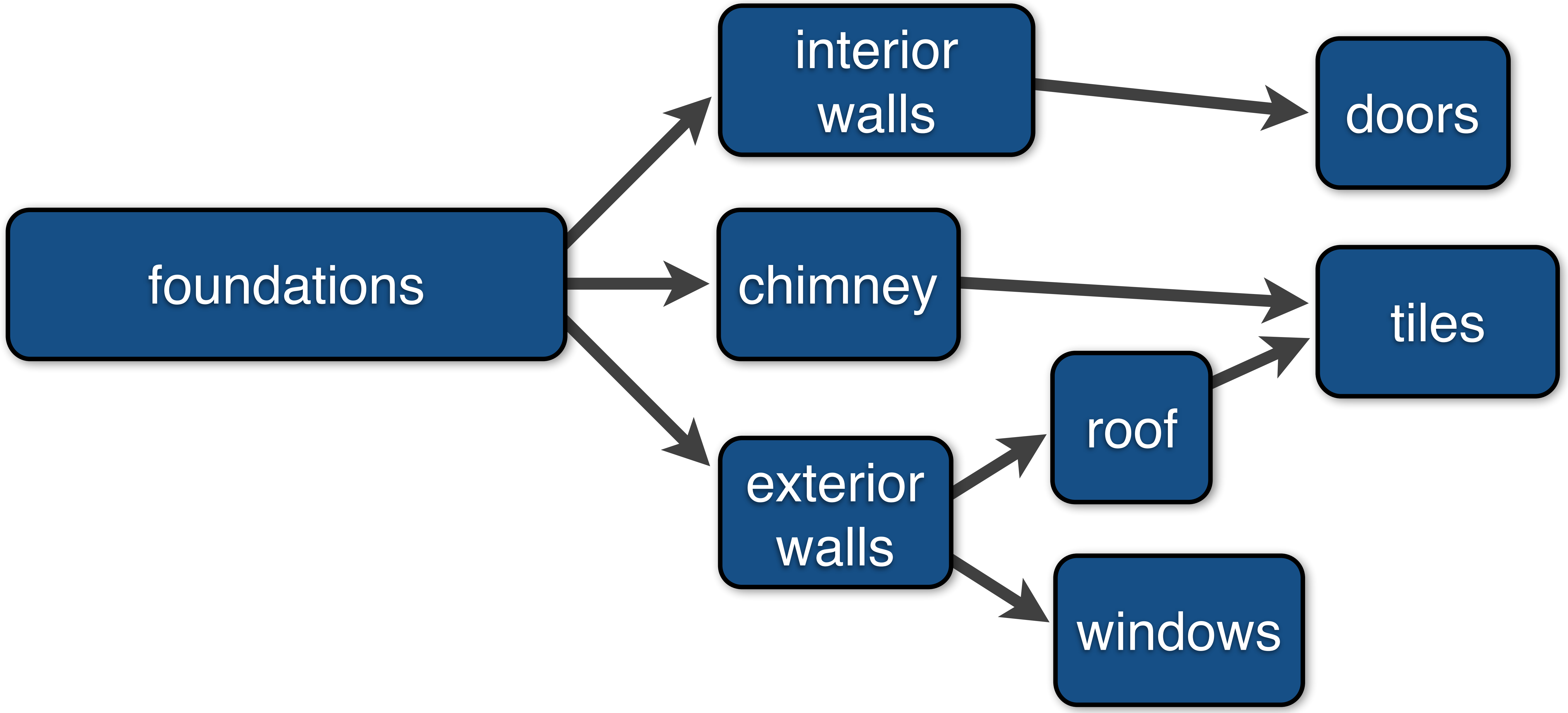
- Basic scheduling involves:
  - tasks with durations
  - precedences between tasks
    - one task must complete before another starts

- The aim is to schedule the tasks
  - usually to minimize the latest end time

Project Scheduling

- Building a house involves a number of tasks, and precedences where one task may not be started until another is completed. Each task has a duration. We need to determine the schedule that minimises the total time to build the house
  - Task (duration): foundations (7), interior walls (4), exterior walls (3), chimney (3), roof (2), doors (2), tiles (3), windows (3).
  - walls and chimney need foundations finished, roof and windows after exterior walls, doors after interior walls, tiles after chimney and roof

Project Scheduling



- Length indicates durations
- Arcs indicate precedences

Project Scheduling

- Data

```
int: n = 8; % no of tasks max
set of int: TASK = 1..n;
int: f = 1; int: iw =2; int: ew = 3;
int: c = 4; int: r = 5; int: d = 6;
int: t = 7; int: w = 8;
```

asignamos identificador a cada tarea
foundations ; interior walls i exterior walls
chimney i roof ; doors
tiles ; windows

```
array[TASK] of int: duration =
  [7,4,3,3,2,2,3,3];
int: p = 8; % number of precedences
set of int: PREC = 1..p;
array[PREC] of TASK: pre =
  [f,f,f,ew,ew,iw,c,r];
```

origen flechita

```
array[PREC] of TASK: post =
  [iw,ew,c,r,w,d,t,t];
```

destino flechita

Project Scheduling

- **Decisions**

```
int: s = sum(duration);
array[TASK] of var 0..s: start;
```

Cuándo empieza y acaba cada tarea. Se cuenta en días de manera relativa (día 1, 2, 3).

↳ cota superior, peor caso. Sirve para curar en salud qe seguro qe no se tardará más qe esto.

- **Constraints**

```
forall(i in PREC)
  (start[pre[i]] + duration[pre[i]]
   <= start[post[i]]);
```

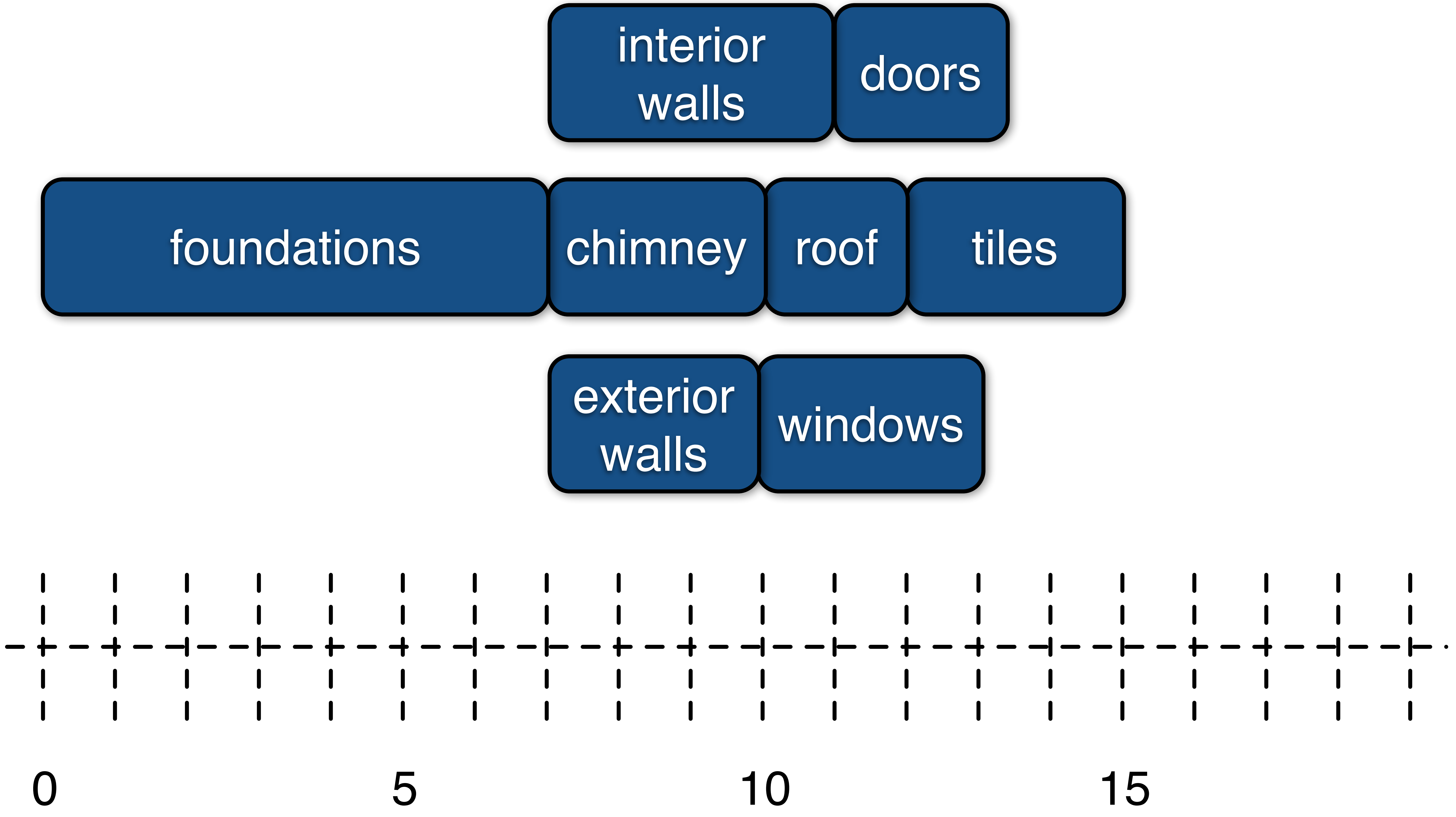
- **Objective**

```
var 0..s: makespan;
forall(t in TASK)
      (start[t] + duration[t] <= makespan);
solve minimize makespan;
```

Project Scheduling

- Constraints generated
  - s[f] + 7 <= s[iw]
  - s[f] + 7 <= s[ew]
  - s[f] + 7 <= s[c]
  - s[ew] + 3 <= s[r]
  - s[ew] + 3 <= s[w]
  - s[iw] + 4 <= s[d]
  - s[c] + 4 <= s[t]
  - s[r] + 2 <= s[t]
  - s[d] + 2 <= makespan
  - s[t] + 3 <= makespan
  - s[w] + 3 <= makespan

Project Scheduling Solution




```
makespan     f  iw  ew   c   r   d   t   w
15      = [0,  7,  7,  7, 10, 11, 12, 10]
```
→ tiempo de inicio

Difference logic constraints

- Difference logic constraints take the form
  - $x + d \leq y$ $\quad$ d is constant
- Note $x + d = y \leftrightarrow x + d \leq y \wedge y + (-d) \leq x$

  si se cumple esto significa que el problema es demasiado sencillo como para resolverlo con minizinc.

- A problem that is representable as a conjunction of difference logic constraints can be solved very rapidly
  - longest/shortest path problem
- But adding extra constraints means this advantage disappears
  - e.g. at most two tasks can run simultaneously

  en la realidad los problemas no son tan limpios, de modo que en la realidad sí que se usan algoritmos complejos.

Disjunctive Scheduling

Peter Stuckey

Scheduling Concepts (so far)

- Tasks
  - start time, duration, and end time
  - other attributes

```
array[TASK] of var int: s;
array[TASK] of var int: d;
array[TASK] of var int: e;
forall(t in TASK)(e[t] = s[t] + d[t]);
```

  - may omit end times, particular when d is fixed
- Precedences
  - one task can only start after another finishes
  - task t1 precedes t2

```
e[t1] <= s[t2]     (s[t1] + d[t1] <= s[t2])
```

Nonoverlap

- Consider the ProjectScheduling problem where we only have one carpenter who can undertake the walls and roof work
  - these tasks cannot overlap in duration

start time   duration

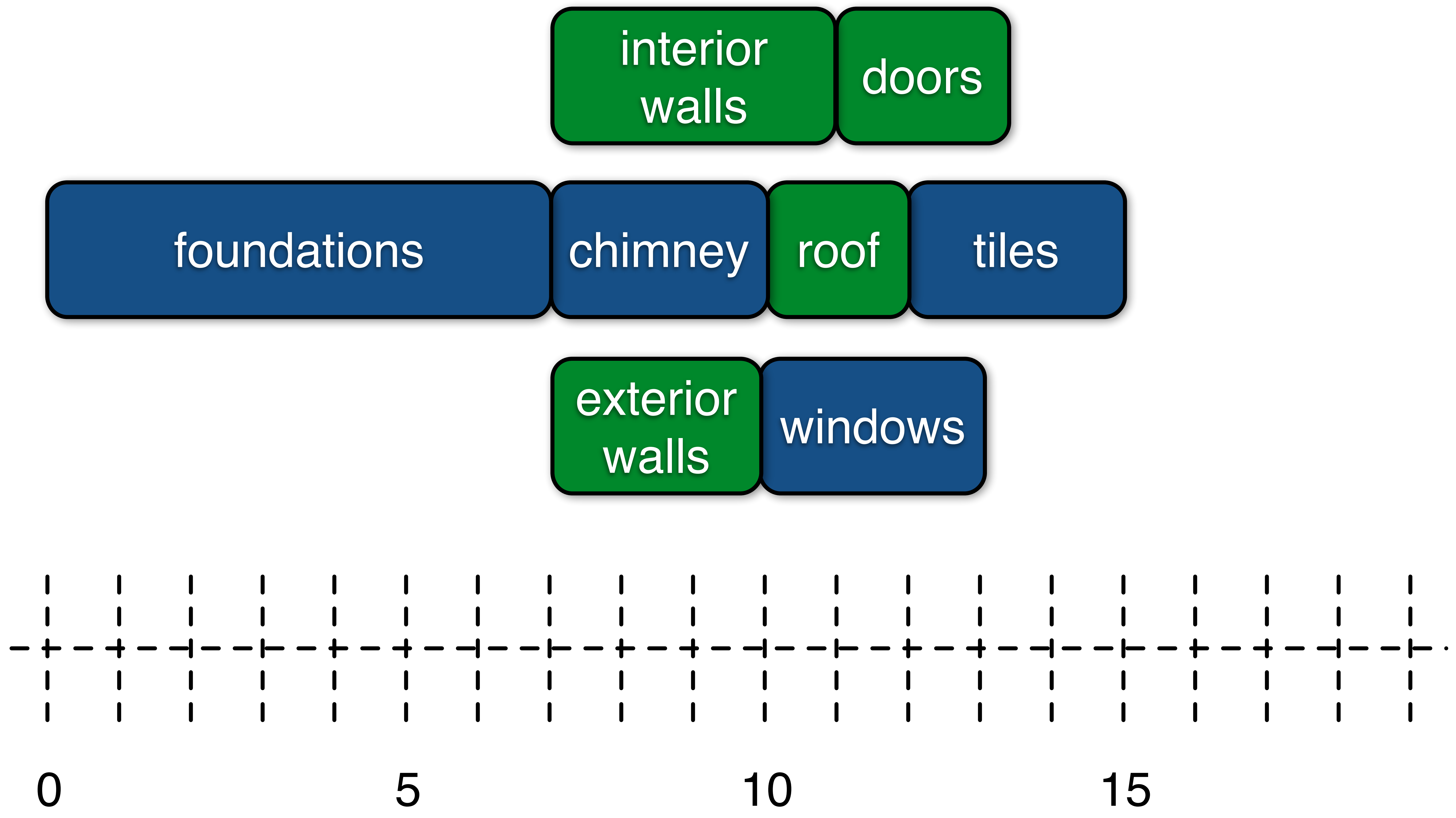
```
predicate nonoverlap(var int:s1, var int:d1,
                     var int:s2, var int:d2)=
          s1 + d1 <= s2 \/ s2 + d2 <= s1;
```

para definir que las tareas no se pueden solaparen.

```
set of TASK: CARPENTRY = { iw, ew, r, d };
forall(t1, t2 in CARPENTRY where t1 < t2)
      (nonoverlap(start[t1],duration[t1],
                  start[t2],duration[t2]));
```

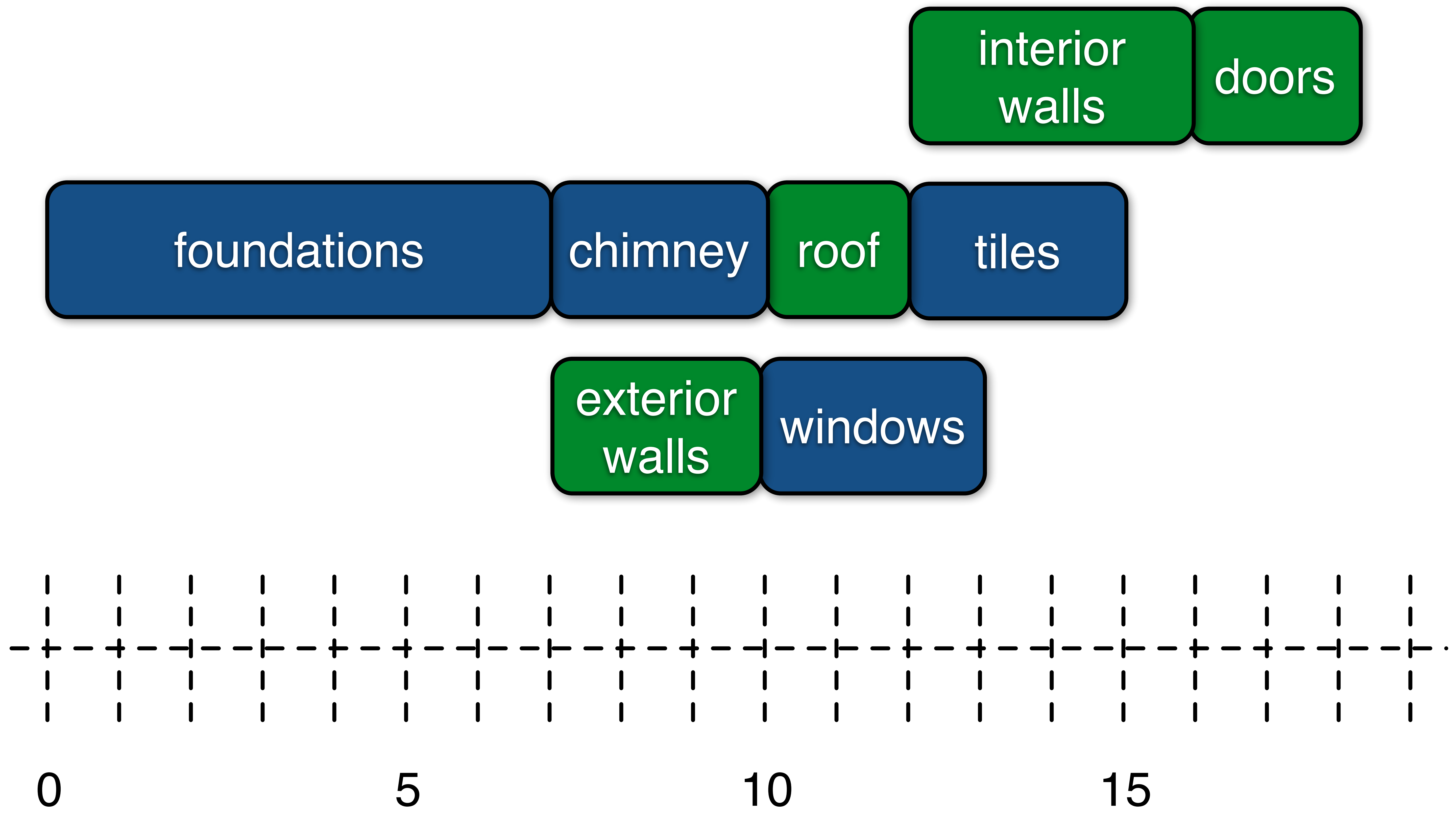
Lo mismo pero para más tareas, todas las definidas en el array.

ProjectScheduling with Carpentry




```
makespan      f   iw ew   c    r    d    t    w
15        = [0,   7, 7, 7, 10, 11, 12, 10]
```

ProjectScheduling with Carpentry




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

Resources

- Critical to most scheduling problems are limited resources
  - unary resource (at most one task at a time)
  - cumulative resource (a limit on the amount of resource used at any time)

Unary Resources

- The ProjectScheduling problem with non overlap involved a unary resource
  - number of tasks executing at one time

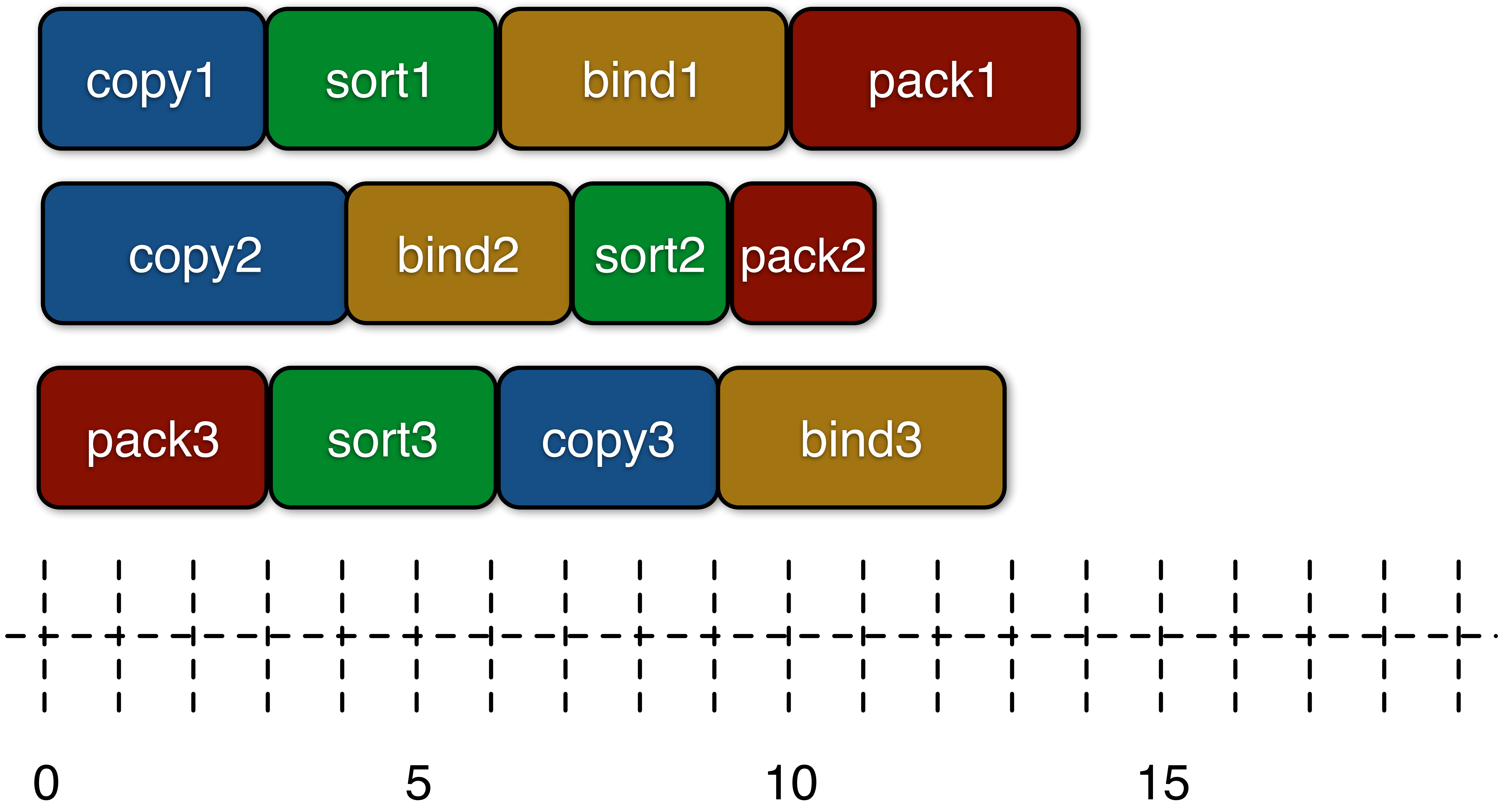
- Unary resources are common
  - machine
  - nurse, doctor, worker in a roster
  - track segment (one train at a time)
  - …

JobShop Scheduling

- JobShop: Given *n* jobs each made up of a sequence of *m* tasks, one each on each of *m* machines. Schedule the tasks to finish as early as possible where each machine can only run one task at a time
- Data

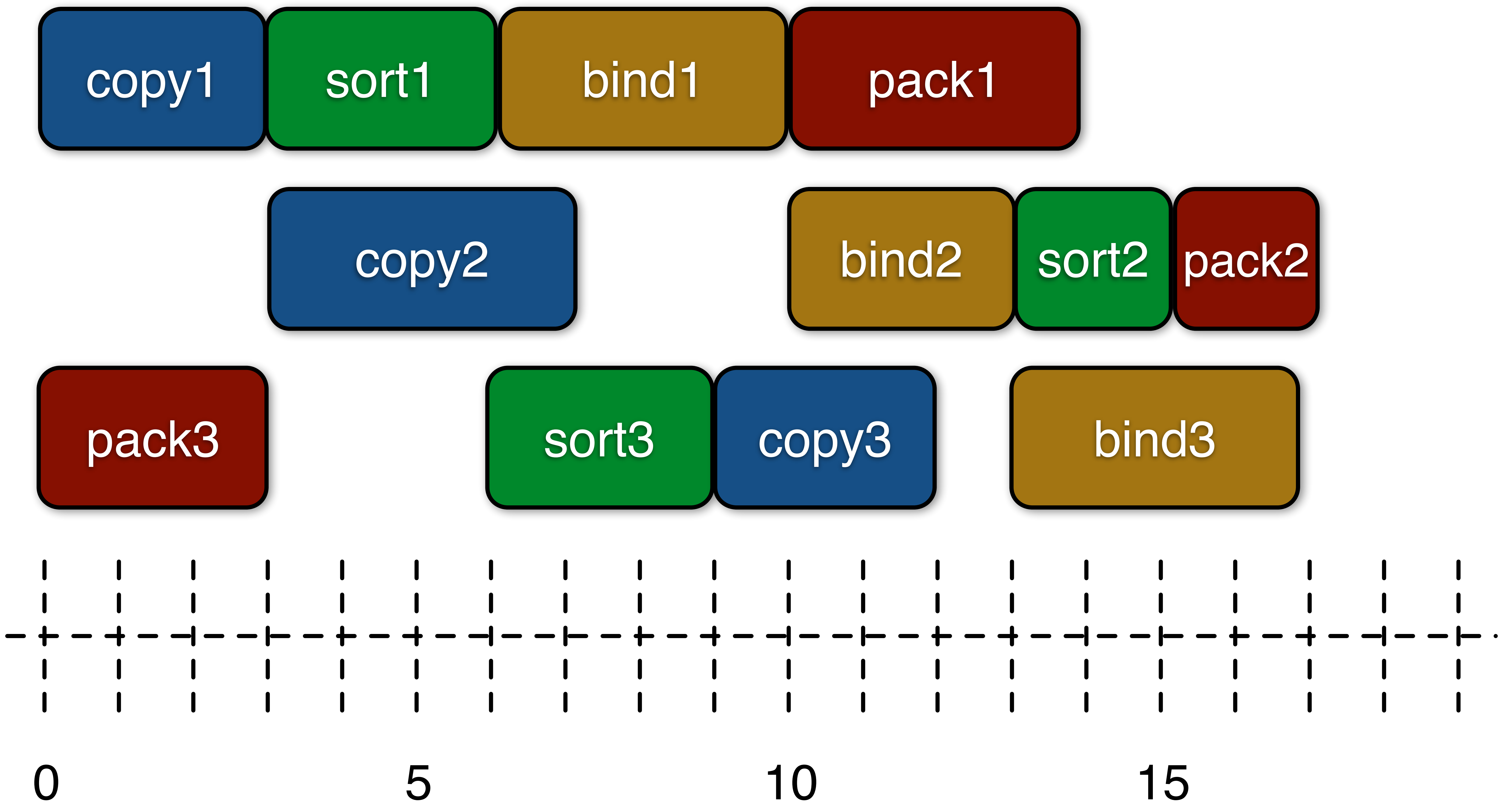
```
int: n;
set of int: JOB = 1..n;
int: m;
set of int: MACH = 1..m;
set of int: TASK = 1..m;
array[JOB,TASK] of int:  du; % length of task
array[JOB,TASK] of MACH: mc; % which machine
```

JobShop Example



- Rows indicate tasks in a Job
- Colors indicate different machine

JobShop Solution



The same machine can't do two different tasks at the same time.

- Tasks pushed later so no two of the same color are simultaneous

JobShop Variables + Constraints

- Variables

```
int: maxt = sum(j in JOB, t in TASK)(d[j,t]);
array[JOB,TASK] of var 0..maxt: s;
```

- Precedence Constraints

```
forall(j in JOB, t in 1..m-1)
      (s[j,t] + d[j,t] <= s[j,t+1]);
```

- Machine Constraints

```
forall(j1, j2 in JOB, t1, t2 in TASK where
       j1 < j2 /\ mc[j1,t1] = mc[j2,t2])
      (nonoverlap(s[j1,t1],d[j1,t1],
                  s[j2,t2],d[j2,t2]));
```

JobShop Objective

- Minimize the makespan (when the last job finishes)

```
var 0..maxt: makespan;
forall(j in JOB)
      (s[j,m] + d[j,m] <= makespan);
solve minimize makespan;
```

última tarea de cada trabajo