Optional Task Scheduling

Peter Stuckey & Guido Tack

MONASH University

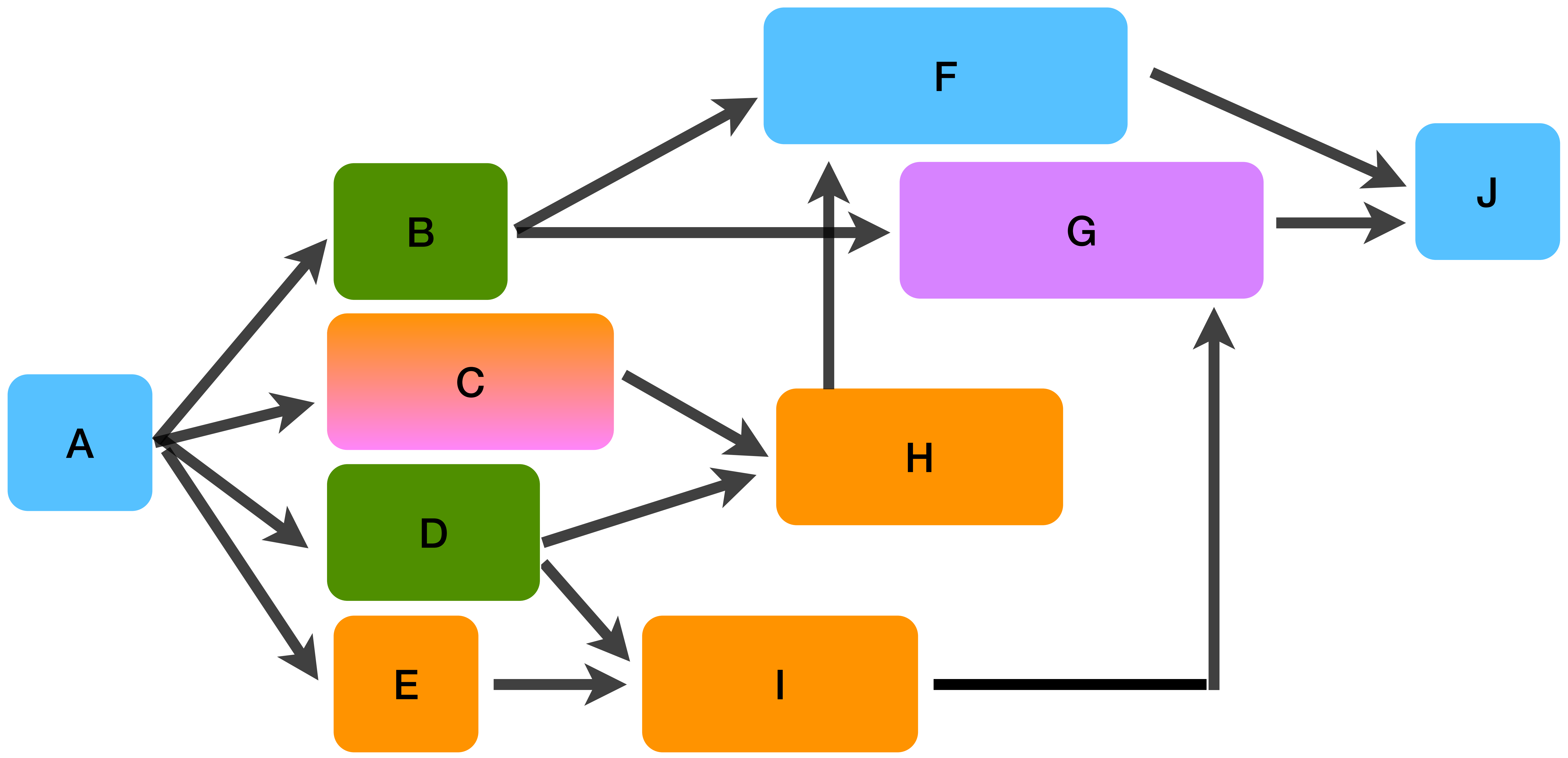
Optional Tasks

- Sometimes we have two or more ways to complete the same task
  - e.g. the same task could be performed on two different machines (but with different duration)
- The usual way of modelling this in CP is using **optional tasks**
  - e.g. create two tasks (one for each machine) but only require one to be executed.
- We can model optional execution by
  - **Optional start time**: <> if the task is not executed
  - **Variable duration**: 0 if not executed

Optional Tasks

- Optional tasks if not executed
  - **Act as if they dont exist**
- Care must be taken to model this correctly
  - e.g. precedences among optional tasks
    - Should always hold if one of the tasks is not executed.
  - e.g. if t1 before t2 and t2 before t3 and t2 doesnt run?
    - One interpretation: no constraints
    - Another interpretation: t1 before t3 (transitive closure)

Scheduling Instance Redux



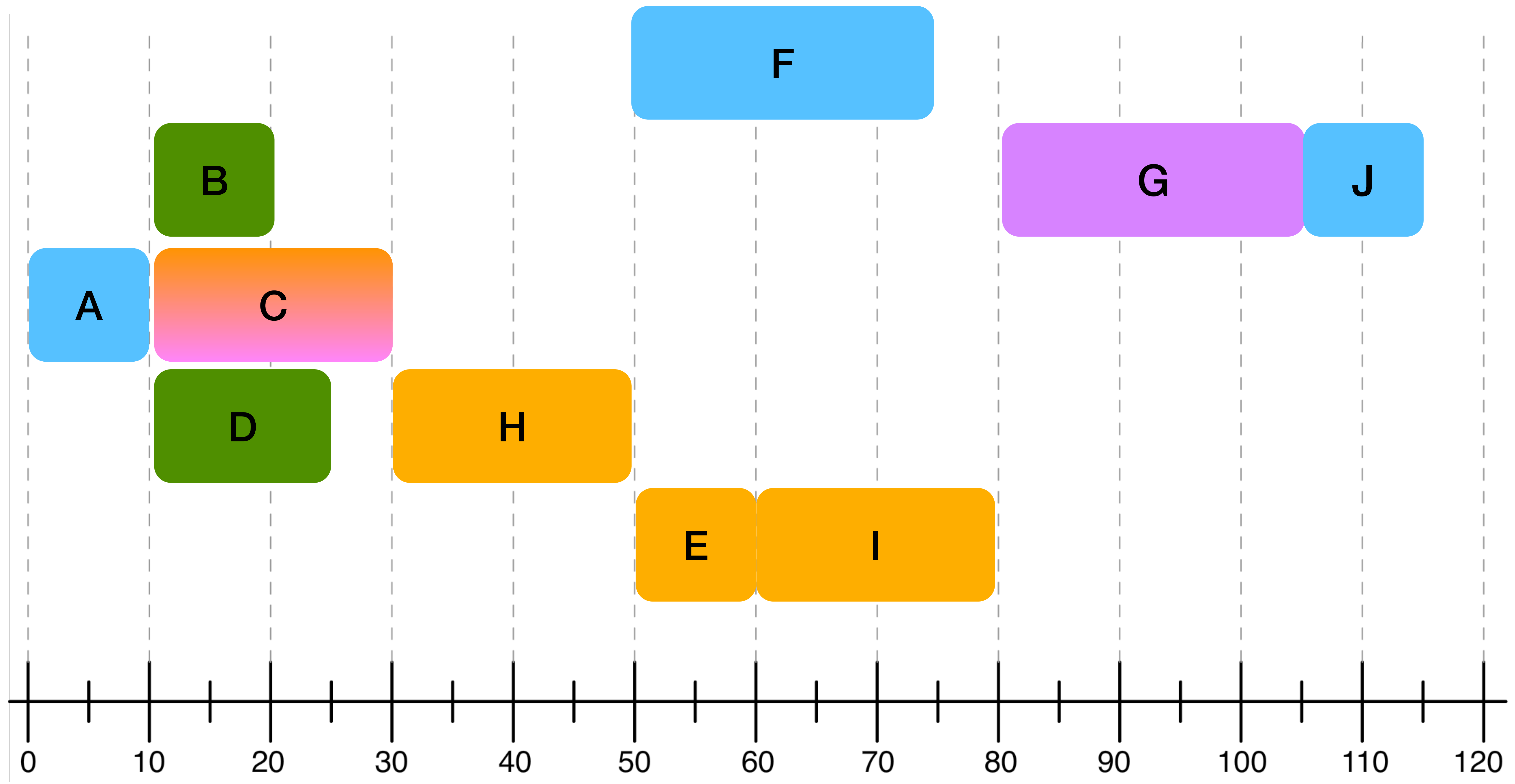
- Suppose only one of C or G is required
- And only one of B and D

Duration modelling of Optionality

- Optional tasks that don’t execute have 0 duration
    - Implicitly implements the transitive closure

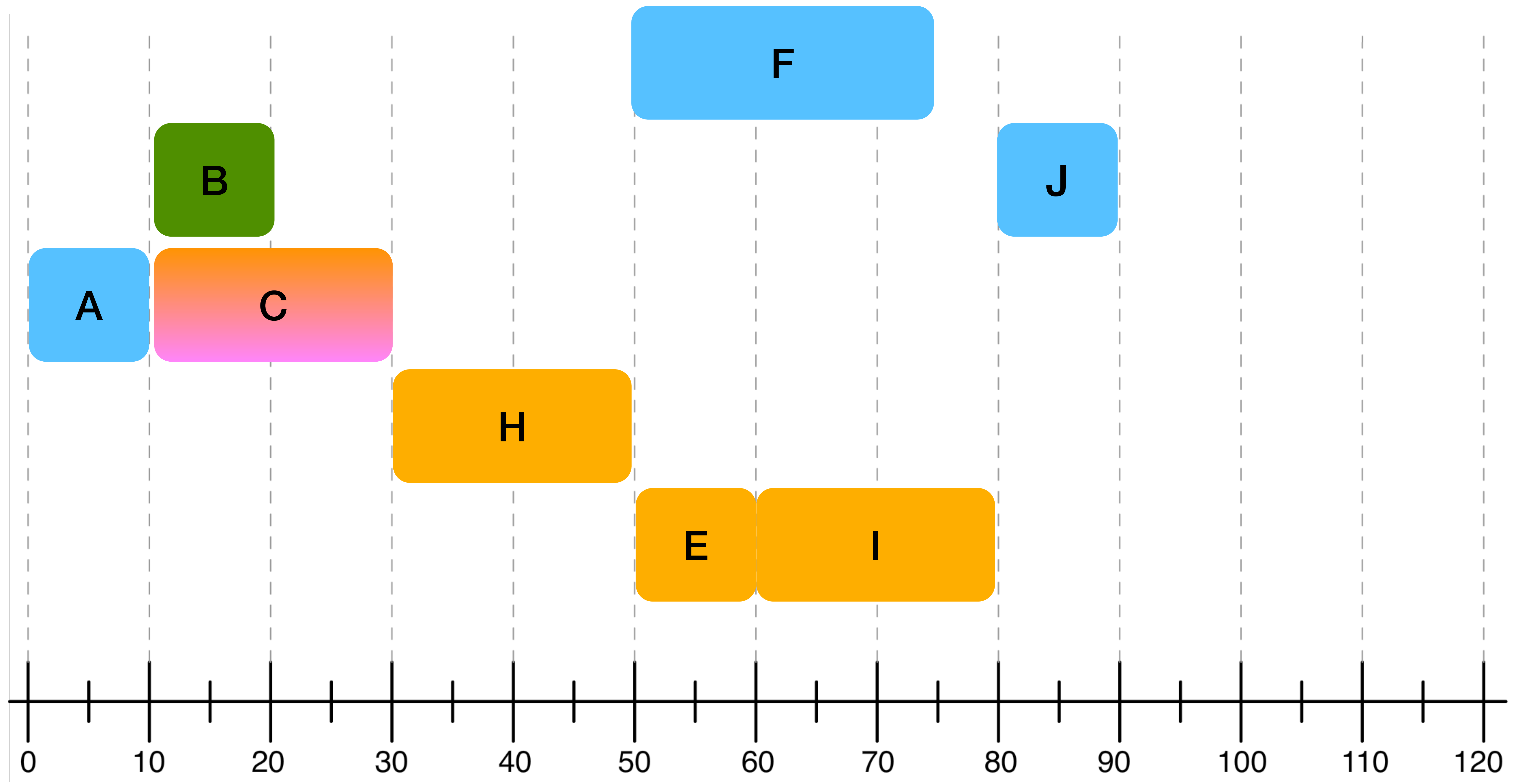
```
array[int] of set of TASK: options; % data

array[TASK] of var 0..e: start;
array[TASK] of var bool: runs; % which tasks run
array[TASK] of var 0..max(duration): aduration =
 [ runs[t]*duration[t] | t in TASK ];
  % replace duration by aduration in precs/disj

% every non optional task always runs
constraint forall(t in TASK diff array_union(options))
                 (runs[t]);
% (at least) one task of each option set runs
constraint forall(i in index_set(options))
           (exists(t in options[i])(runs[t]));
```

Schedule without Optionality



```
makespan      A    B    C    D    E    F    G    H    I    J
115         = [0,  10,  10,  10,  50,  50,  80,  30,  60,  105]
```

Scheduling with Optionality (0 duration)



```
makespan     A   B   C   D   E   F   G   H   I   J
90         = [0, 10, 10, 10, 50, 50, 80, 30, 60, 80]
```

Optional Start Time modelling of Optionality

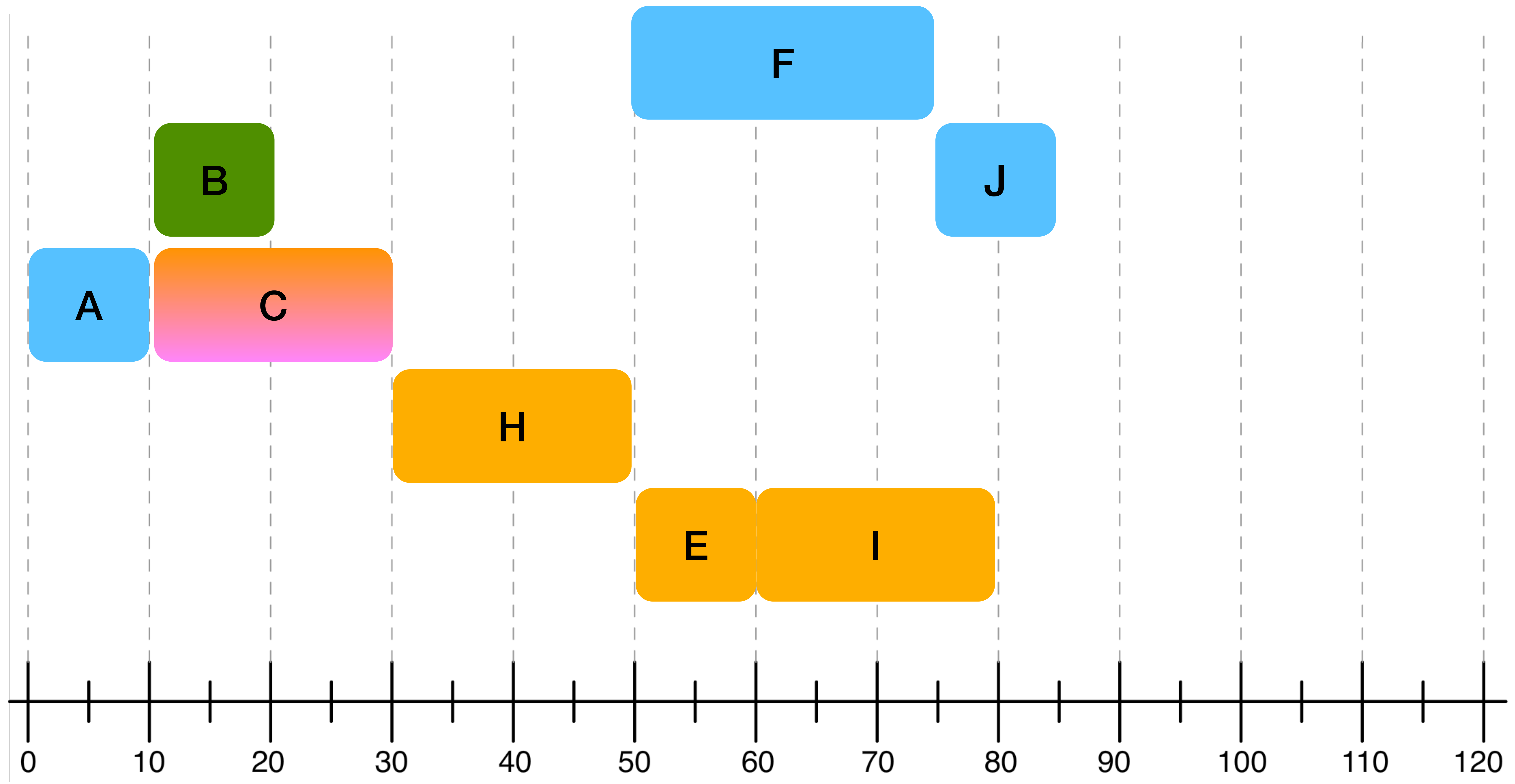
- MiniZinc supports optional variables
  - Extra value <> means "absent"
  - Variables taking value <> act as if "not there"
  - occurs(x) forces x to not take value <>
- The optional variable modelling simply changes

```
array[int] of set of TASK: options; % data

array[TASK] of var opt 0..e: start;
% every non optional task always occurs
constraint forall(t in TASK diff array_union(options))
                  (occurs(start[t]));
% (at least) one task of each option set occurs
constraint forall(i in index_set(options))
              (exists(t in options[I])(occurs(start[t])));
```

Optional keyword

occurs

Scheduling with Optionality (opt)



Lost the transitive constraint
I << G << J

```
makespan     A    B    C    D    E    F    G    H    I    J
85        = [0, 10, 10, <>, 50, 50, <>, 30, 60, 75]
```

Task Variables

- Many CP solvers support **task variables**
- Record of: (occurs, start, duration, end)
    - Optionality
    - end = start + duration
- They also have complex propagators to reason about them
    - Particular interaction of optionality + resources

Optional Task Scheduling vs Temporal Planning

- Suppose we generate k_a optional tasks for each possible action a
- The optional task scheduling problem can be used to represent a **(bounded) temporal planning problem**.
- Good idea:
  - Small bounds, and shared resources for tasks
- Bad idea:
  - Complex fluents, large numbers k_a

Summary

- Optional Tasks / Task Variables
  - Are a key to modelling complex scheduling problems
  - Commercial CP solvers spend a lot of effort to deal with them well
- Optional Task Scheduling ≈ Planning
- Task variables are coming to MiniZinc soonish!

EOF