Introduction to Artificial Intelligence

COMP307
Planning and Scheduling 3:
Dynamic Scheduling

Yi Mei

yi.mei@ecs.vuw.ac.nz

Outline

- Dynamic Scheduling
- Dispatching Rules
  - Generating schedules by rules
- Designing Dispatching Rules
  - Terminal set
  - Function set
  - Fitness function

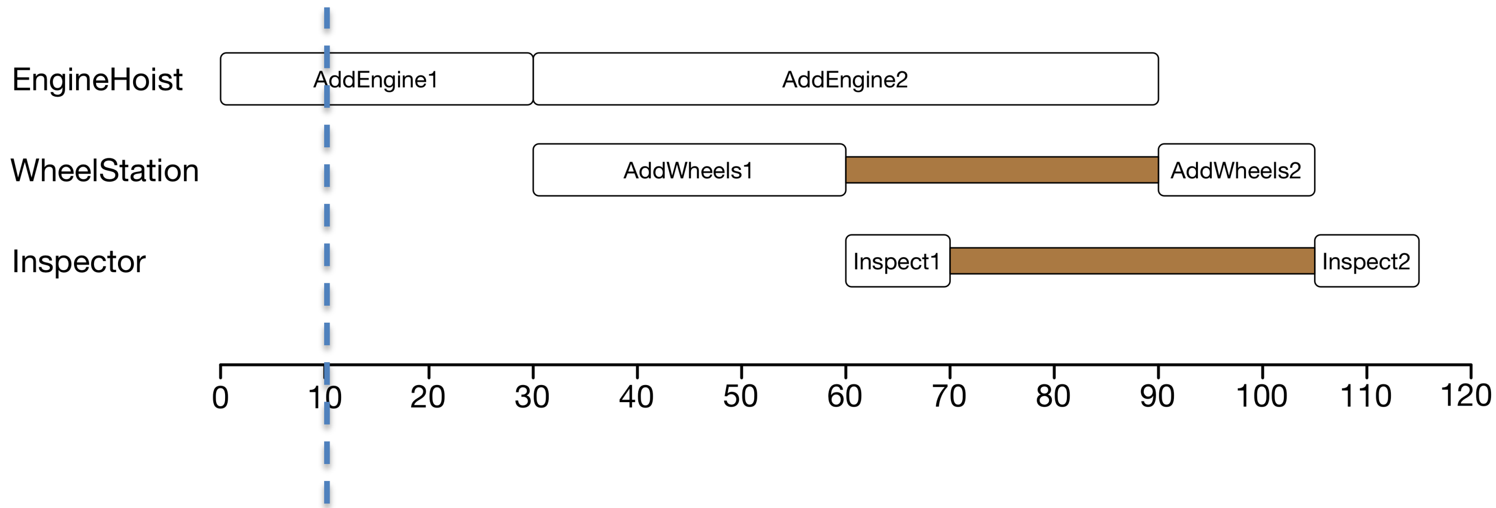
Dynamic Scheduling

- In static scheduling, it is assumed that all the information is known in advance and do not change over time
- In real life, usually not the case (dynamic environment)
  - The plan today won't work tomorrow

Dynamic Scheduling

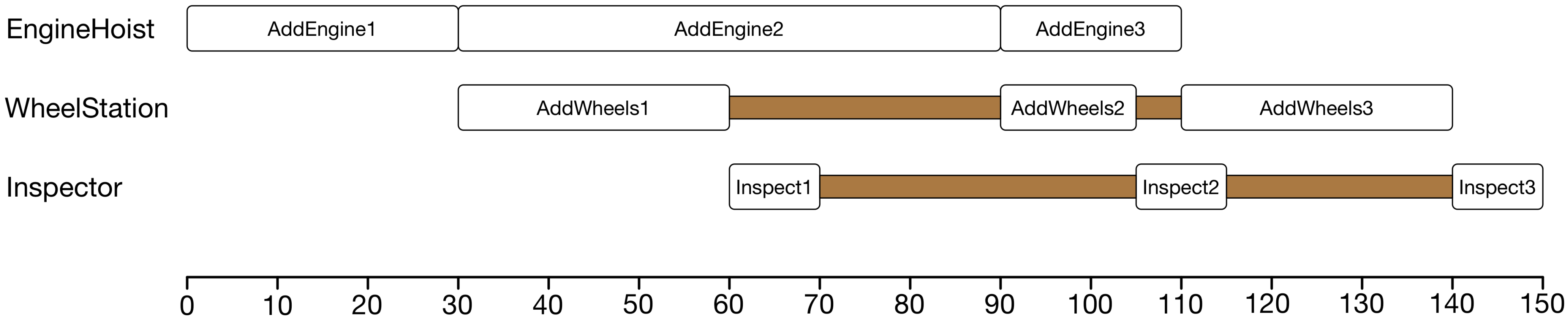
- Manufacture two cars
  - 2 jobs known in advance
  - Already made a plan: makespan = 115
  - A new job arrives at time 10

  - Job({AddEngine3 $\prec$ AddWheels3 $\prec$ Inspect3})
  - Operation(AddEngine3, ProcTime: 20, Use: EngineHoist)
  - Operation(AddWheels3, ProcTime: 30, Use: WheelStation)
  - Operation(Inspect3, ProcTime: 10, Use: Inspector)



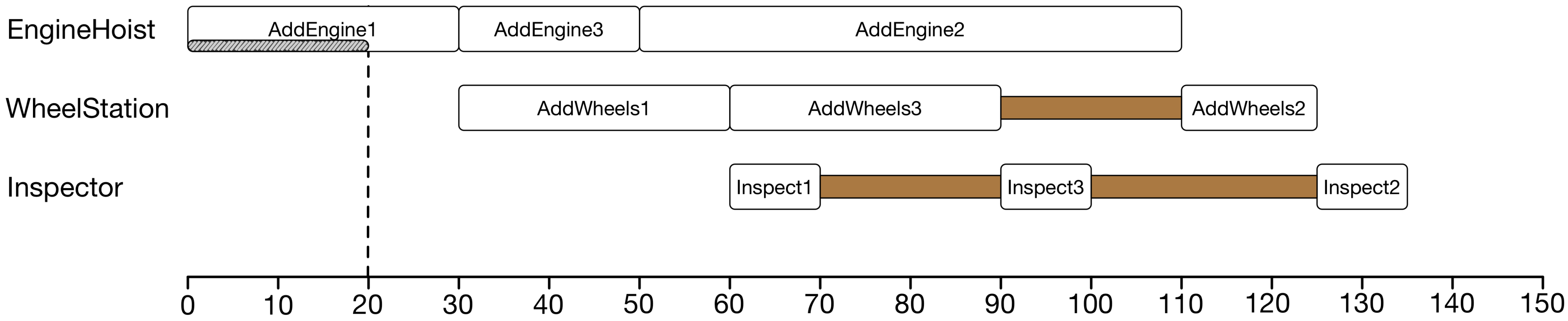
Dynamic Rescheduling

- Simply append to the end of the current schedule
  - Makespan = 150



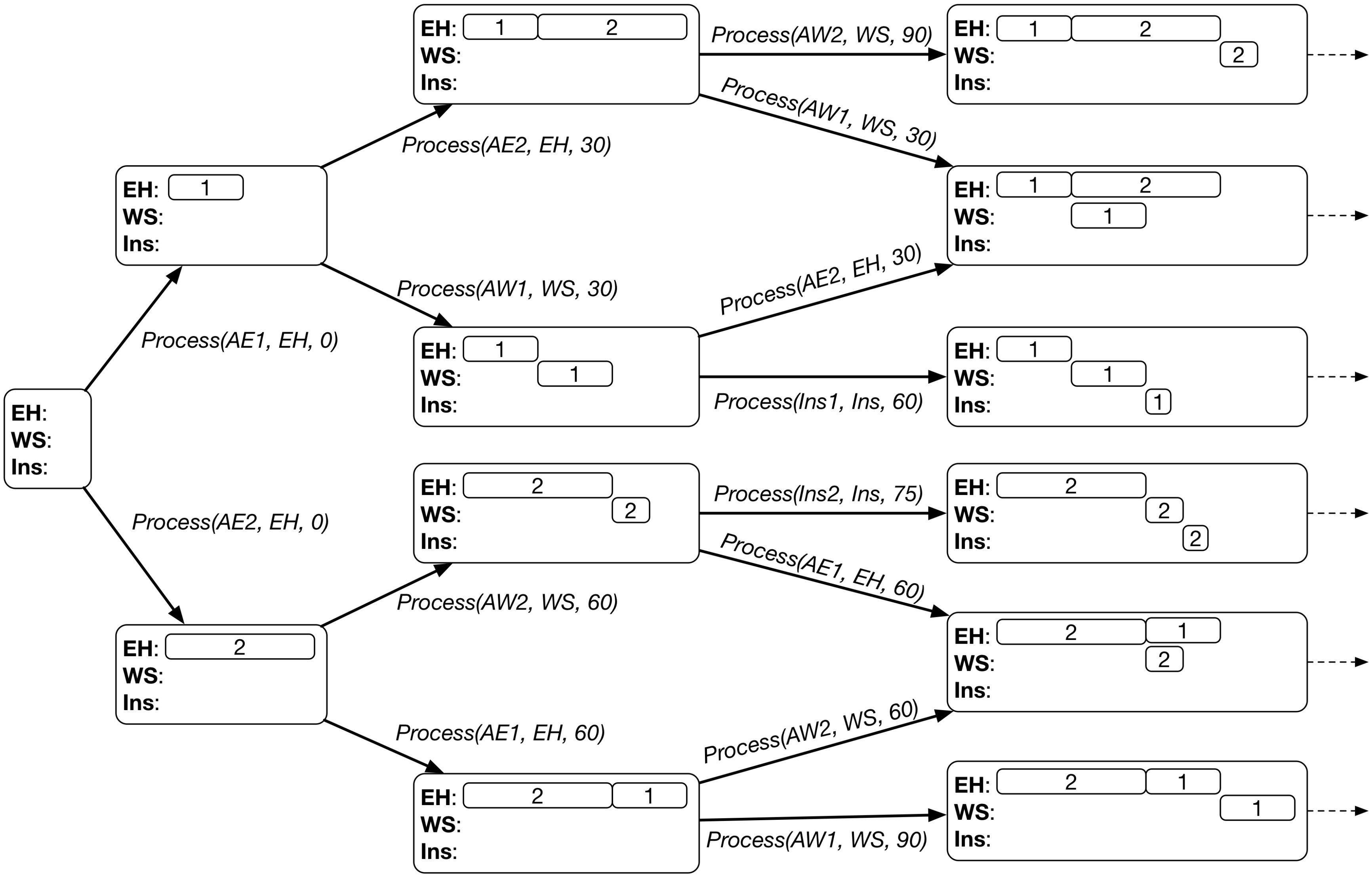
- Re-optimise the unexecuted schedule
  - Makespan = 135, but can be SLOW

10 jobs, 5 machines, 6.3×10^{32} solutions



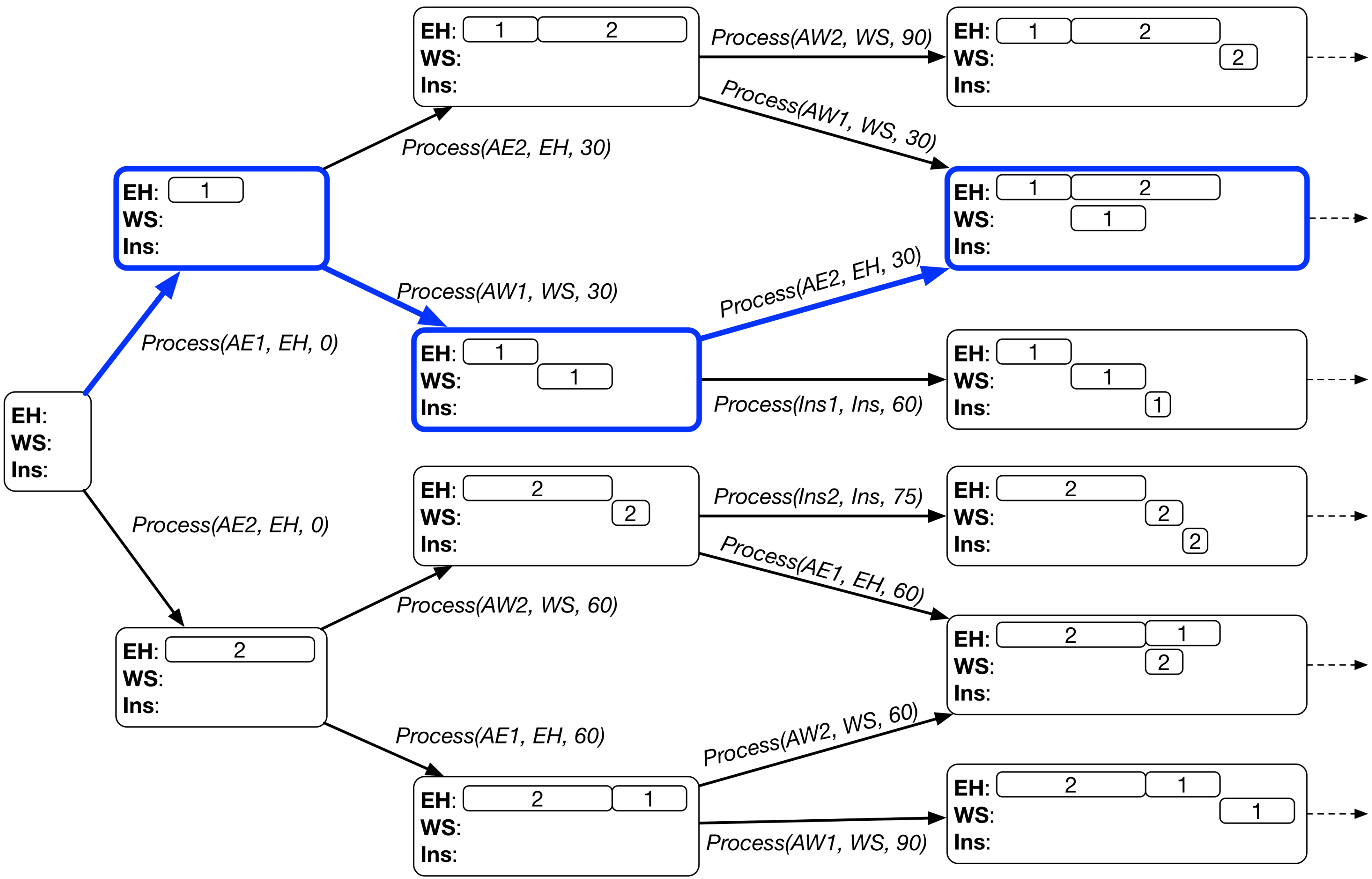
Dispatching Rule

- Forward search: expand all branches, **time consuming**



Dispatching Rule

- Intelligently select one branch at each point?



Dispatching Rule

- Dispatching Rule: a rule to select one action in each state
  - Considering ONLY the earliest applicable actions (non-delay)
  - Assigning a priority to each earliest action by a priority function
  - Selecting the action with the highest priority
- An example: Shortest Processing Time (SPT)
  - Always select the shortest processing
  - Priority of Process(o, m, t) is 0-ProcTime(o)

Dispatching Rule

- Which one is selected?

Action	Priority
Process(AddEngine2, EngineHoist, 30)	-60
Process(Inspect1, Inspector, 60)	-10

- Which one is selected?

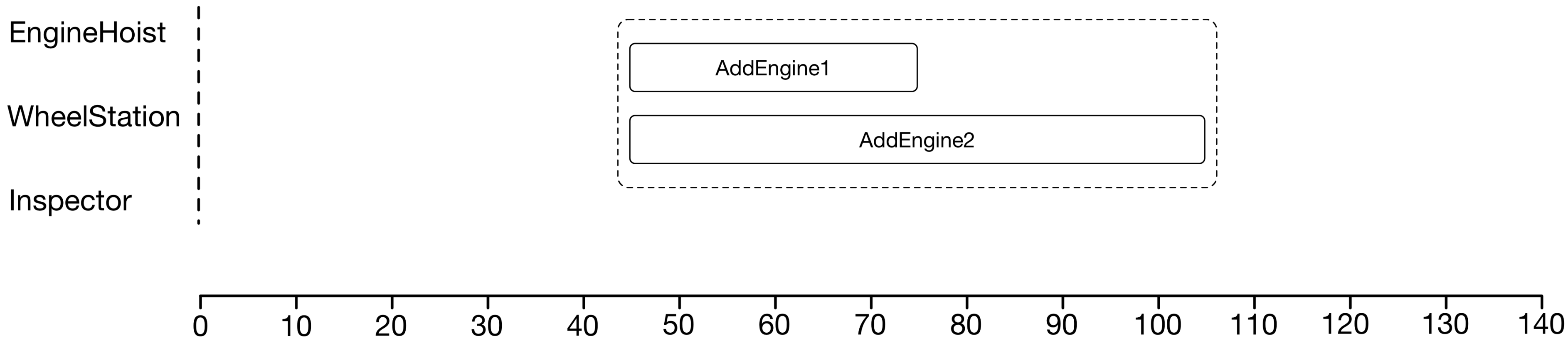
Action	Priority
Process(AddEngine1, EngineHoist, 0)	-30
Process(AddEngine2, EngineHoist, 0)	-60

Generate a Schedule by Dispatching Rule

- Step 1: Initialise state
  - empty schedule, all operations unprocessed, time = 0, machine idle time = 0, first operation ready time = **arrival time**, other operation ready time = ∞
- Step 2: Find the earliest applicable actions;
- Step 3: Select the next action by the dispatching rule
- Step 4: Add the selected action into the schedule, update the state
- Step 5: If all operations are processed, stop. Otherwise, go to step 2.

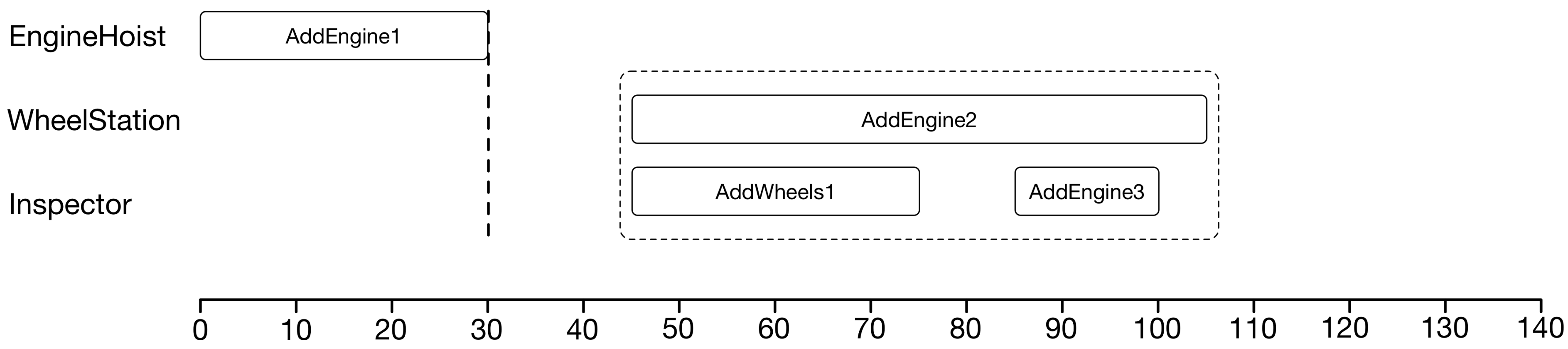
Generate Schedule by SPT

	Arrive	ProcTime		
		AddEngine	AddWheels	Inspect
Job 1	0	30	30	10
Job 2	0	60	15	10
Job 3	10	20	30	10



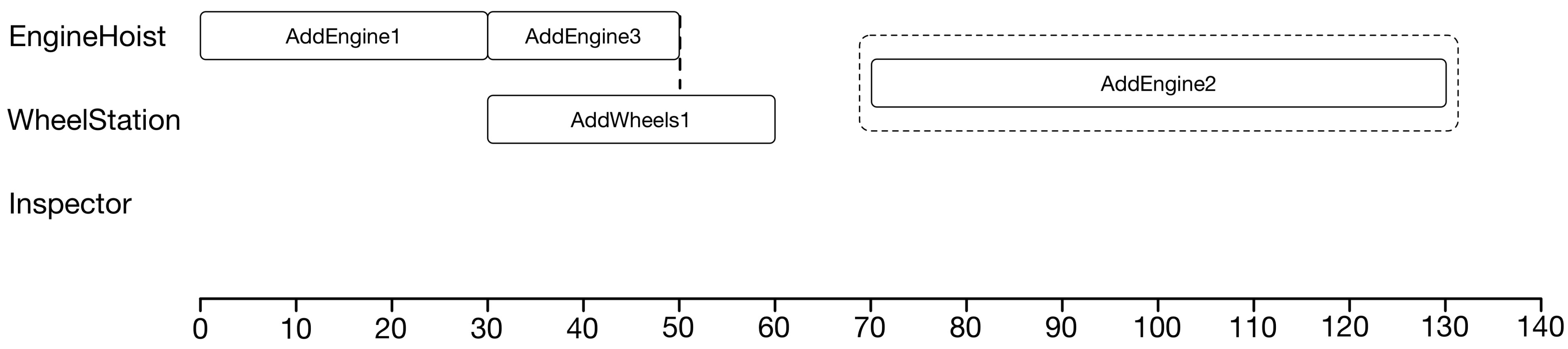
Generate Schedule by SPT

	Arrive	ProcTime		
		AddEngine	AddWheels	Inspect
Job 1	0	30	30	10
Job 2	0	60	15	10
Job 3	10	20	30	10



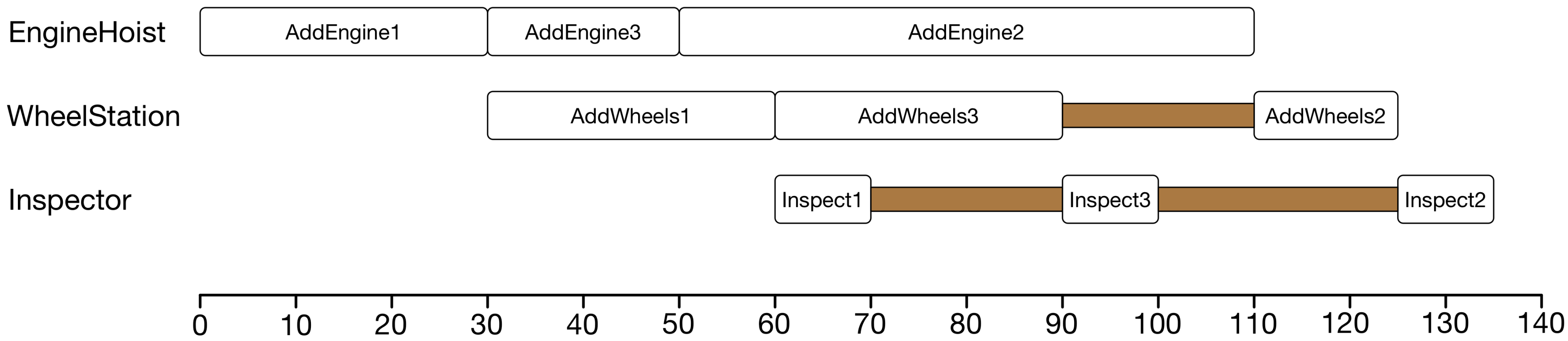
Generate Schedule by SPT

	Arrive	ProcTime		
		AddEngine	AddWheels	Inspect
Job 1	0	30	30	10
Job 2	0	60	15	10
Job 3	10	20	30	10



Generate Schedule by SPT

	Arrive	ProcTime		
		AddEngine	AddWheels	Inspect
Job 1	0	30	30	10
Job 2	0	60	15	10
Job 3	10	20	30	10



Advantage of Dispatching Rule

- Can be apply at ANY time point to change the remaining schedule
  - Initial state = current state
  - But only need at critical time point (a machine becomes idle, an operation becomes ready)
- Select ONLY the next action to be taken, NO need to generate the entire remaining schedule
- Very quick in real time, can handle dynamic environment very well
  - At each time point, complexity = #unprocessed ops * O(priority)

Design of Dispatching Rule

- Intuition
  - First-Come-First-Serve (Minimum Waiting Time)
  - Shortest Processing Time
  - Earliest Due Date
  - Maximum Work Remaining
  - …
- Look-Ahead
  - Work waiting on the next machine
  - Processing time of the next operation
- Composite rules
  - -(PT+WINQ)
  - -(2PT+WINQ+NPT)
  - …

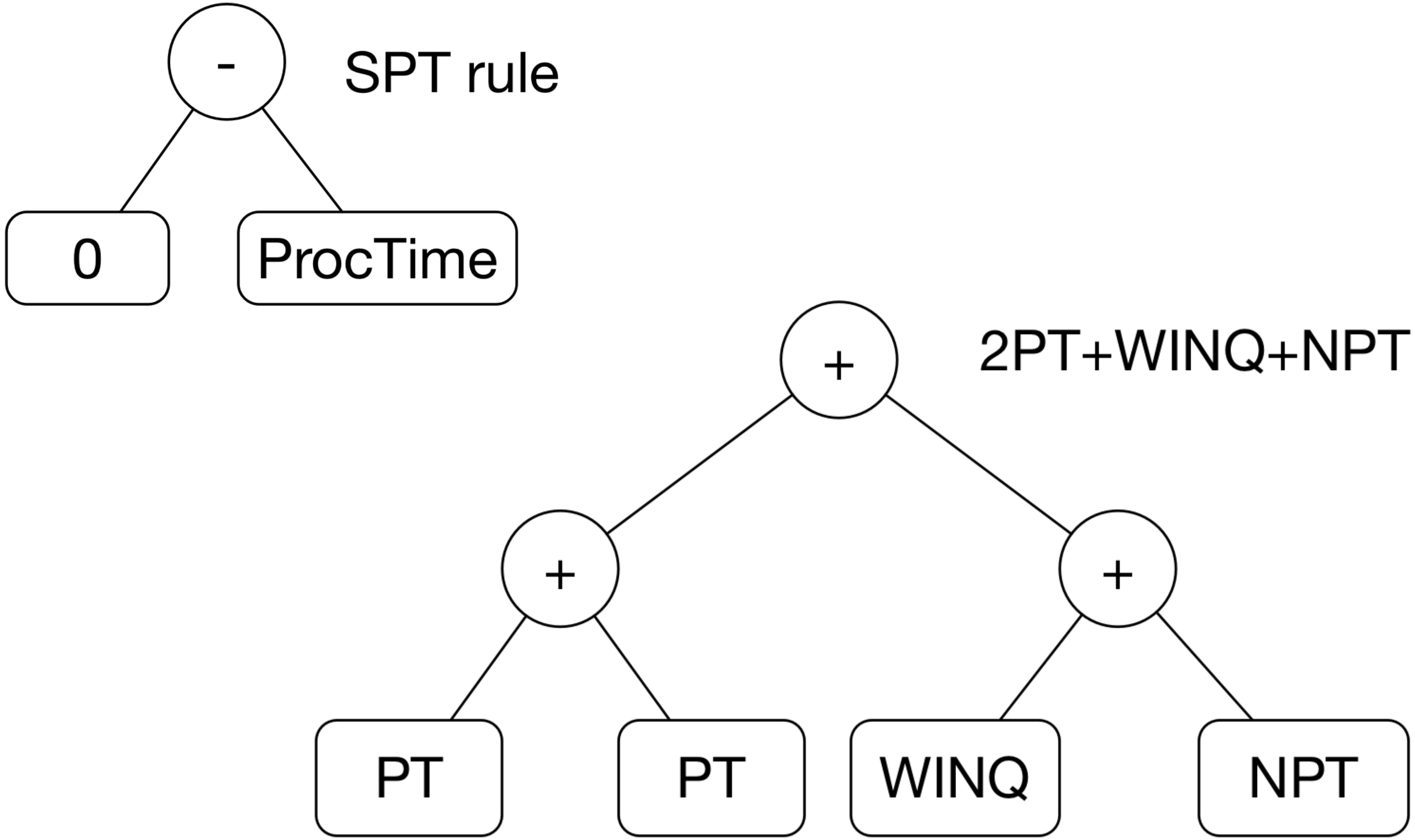
Design of Dispatching Rule

- Different scenarios need different rules
  - Ford car manufacturing factory in summer season
  - Samsung mobile production lines in spring season
- Hard to design effective rule for a particular given scenario

- Use Genetic Programming (GP) to learn/train dispatching rule based on historical data/simulation

Learning Dispatching Rule with GP

- Goal: find the best priority function (GP trees)



Learning Dispatching Rule with GP

- Terminal set: features/attributes of the state and the considered Process(o, m, t)
  - Processing time of o
  - Processing time of o’s next operation
  - Total processing time of all the subsequent operations after o (work remaining)
  - Constant coefficients
  - …
- Function set
  - {+, -, x, /}
  - {max, min}
  - …
- Fitness: average makespan (or any other objective) of the generated schedules for a set of training instances

Summary

- Simple (re-)search cannot handle dynamic scheduling
- Dispatching rule
- Generate a schedule by a dispatching rule
- Learning dispatching rules by GP