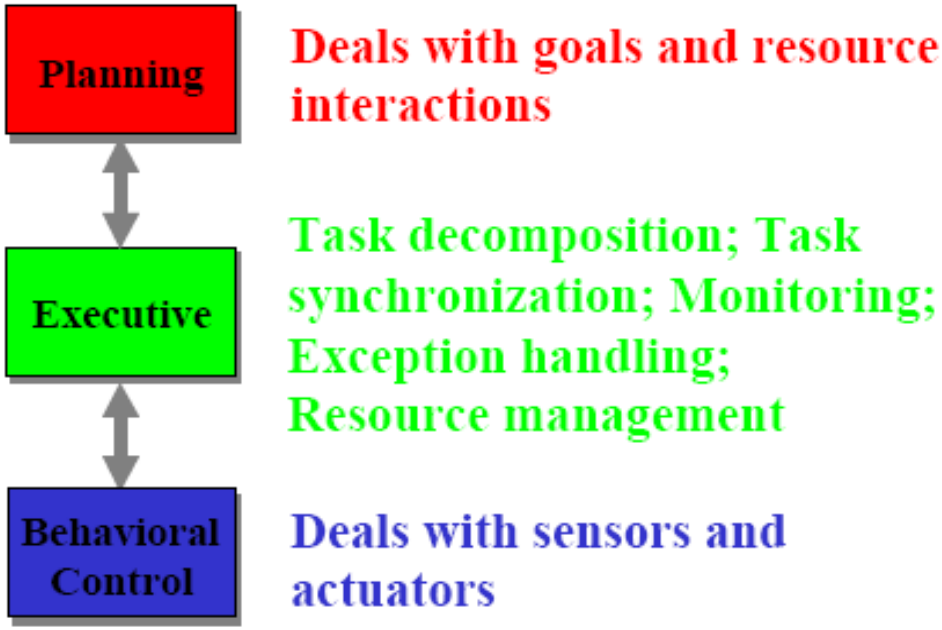
Task Planning

Robotic Architectures

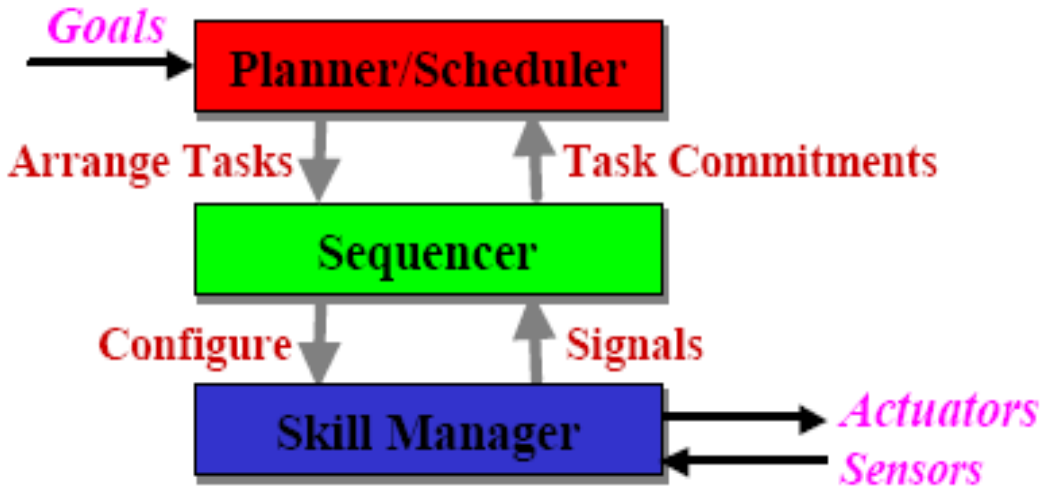
Architetture a 3 Livelli

- Deliberativo:
  pianificazione, ragionamento, decisione
- Esecutivo:
  monitoraggio dell'esecuzione,
  sequenziamento dei comandi
- Funzionale:
  funzionalità di controllo attuative e percettive



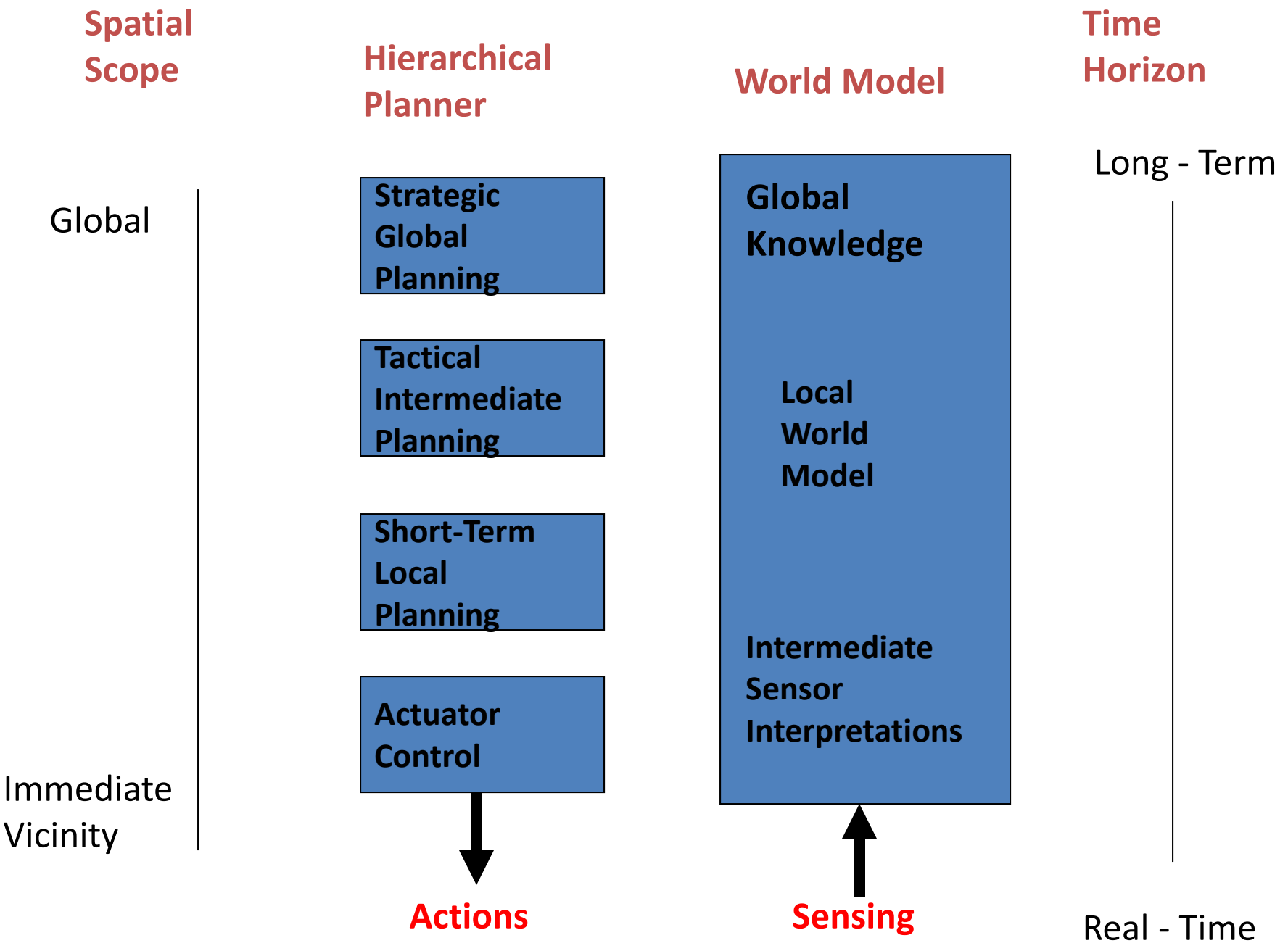
Architetture a 3 Livelli: ATLANTIS

- Explicit Separation of Planning, Sequencing, and Control
  - Upper layers provide *control flow* for lower layers
  - Lower layers provide *status* (state change) and *synchronization* (success/failure) for upper layers
- Heterogeneous Architecture
  - Each layer utilizes algorithms tuned for its particular role
  - Each layer has a representation to support its reasoning



Pianificazione Deliberativa

- Are often aligned with hierarchical control community within robotics.
- **Hierarchical planning** systems typically share a structured and clearly identifiable subdivision of functionality regarding to **distinct program modules** that communicate with each other in a **predictable and predetermined manner**.
- At a hierarchical planner's highest level, the **most global and least specific** plan is formulated.
- At the lowest levels, **rapid real-time** response is required, but **the planner is concerned only** with its immediate surroundings and has lost the sight of the big picture.


Spatial Scope
Hierarchical Planner
World Model
Time Horizon
Global
Immediate Vicinity
Strategic Global Planning
Tactical Intermediate Planning
Short-Term Local Planning
Actuator Control
Global Knowledge
Local World Model
Intermediate Sensor Interpretations
Long - Term
Real - Time
Actions
Sensing

Planning as Search

- Planning is looking ahead, **searching**
- The goal is a state.
- The robot's entire state space is enumerated, and searched, from the current state to the goal state.
- Different paths are tried until one is found that reaches the goal.
- If the optimal path is desired, **then all possible paths must be considered** in order to find the best one.

SPA = Planner-based

- Planner-based (deliberative) architectures typically involve three generic sequential steps or functional modules:

  1) sensing (S)

  2) planning (P)

  3) acting (A), executing the plan

- Thus, they are called SPA architectures.
- **SPA has serious drawbacks.**

Problem 1: Time Scale

- It takes a **very** (prohibitively) long time to search in a real robot's state space, as that space is typically very large.
- Real robots may **have collections of simple digital sensors** (e.g., switches, IRs), a **few more complex ones** (e.g., cameras), or **analog sensors** (e.g., encoders, gauges, etc.)
- => "too much information"
- => **Generating a plan is slow.**

SPA = Planner-based

Problem 2: Space

- It takes a **lot of** space (memory) **to represent** and manipulate the robot's state space representation.
- The representation must **contain all information needed for planning.**
- => Generating a plan can be large.
- Space is not nearly as much of a problem as time, in practice.

Problem 3: Information

- The **planner assumes** that the representation of the state space **is accurate and up-to-date**.
- => The representation must be **constantly updated and checked**
- The more information, the better.
- => "too little information"

SPA = Planner-based

Problem 4: Use of Plans

The resulting plan is only useful if:

a) the environment **does not change** during the **execution of a plan** in a way that affects the plan.

b) the representation **was accurate enough** to generate a correct plan.

c) the robot's effectors **are accurate enough** to perfectly execute each **step of the plan** in order to make the **next step possible**

Deliberation in Summary

- In short, deliberative (SPA, planner-based) approaches:
  - require **search and planning**, which are slow
  - encourage open-loop plan execution, which is limiting and dangerous
- Note that **if planning were not slow** (computationally expensive) then execution would not need to be open-loop, since re-planning could be done.

Hierarchical Planners vs. BBS

Hierarchical Planners

- Rely heavily on world models,
- Can readily integrate world knowledge,
- Have a broad perspective and scope.

BB Control Systems

- afford modular development,
- Real-time robust performance within a changing world,
- Incremental growth
- are tightly coupled with arriving sensory data.

Hybrid Control

- **The basic idea is simple**: we want the best of both worlds (if possible).
- The goal is to **combine** closed-loop and open-loop **execution.**
- That means to **combine reactive and deliberative control.**
- This implies combining the **different time-scales and representations.**
- This mix is called hybrid control.

Hybrid robotic architectures believe that a union of deliberative and behavior-based approaches can potentially yield the best of both worlds.

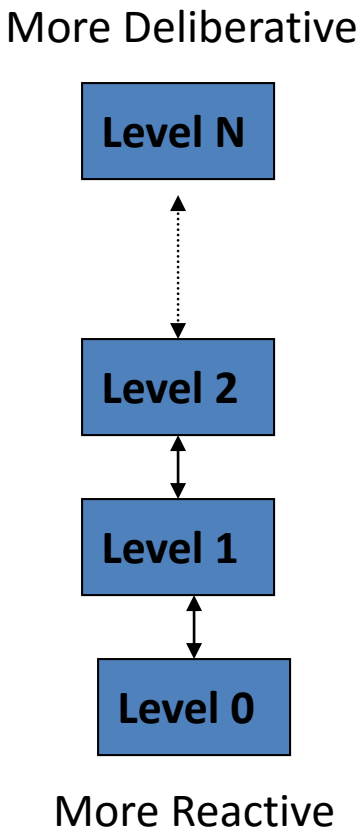
Organizing Hybrid Systems

Planning and reaction can be tied:

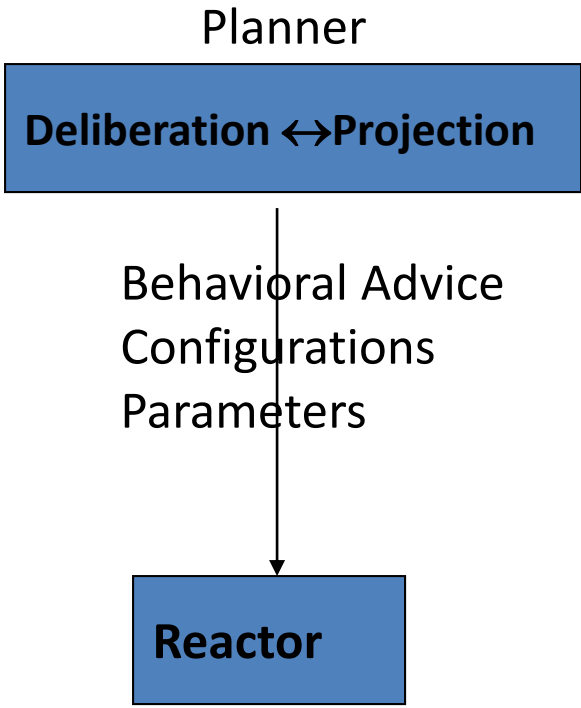
A: hierarchical integration - planning and reaction are involved with different activities, time scales

B: Planning to guide reaction - configure and set parameters for the reactive control system.

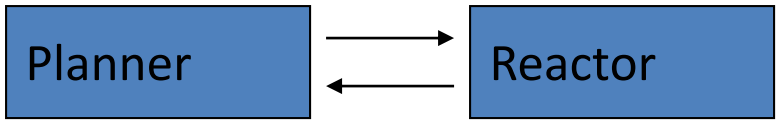
C: coupled - concurrent activities



A



B



C

Organizing Hybrid Systems

It was observed that the emerging architectural design of choice is:

- multi-layered hybrid comprising of
  - * a **top-down** planning system and
  - * a **lower-level** reactive system.

- **the interface** (middle layer between the two components) **design** is a central issue in differentiating different hybrid architectures.

In summary, a modern hybrid system typically consists of three components:

- ♦ **a reactive layer**
- ♦ **a planner**
- ♦ **a layer that puts the two together.**

=> Hybrid architectures are often called **three-layer architectures**.

The Magic Middle: Executive Control

- The middle layer has a hard job:

  1) compensate for the limitations of both the planner and the reactive system

  2) reconcile their different time-scales.

  3) deal with their different representations.

  4) reconcile any contradictory **commands** between the two.

- This is **the challenge** of hybrid systems

  => **achieving the right compromise between the two ends.**

The middle layer services.

Reusing Plans

- **Some frequently useful planned decisions may need to be reused**, so to avoid planning, an intermediate layer **may cache** and look those up. These can be:
  - **intermediate-level actions** (ILAs): stored in contingency tables.
  - **macro operators**: plans compiled into more general operators for future use.

Dynamic Re-planning

- Reaction **can influence** planning.
- Any **"important" changes** discovered by the low-level controller are passed back to the planner **in a way that** the planner can use to re-plan.
- The planner is interrupted when **even a partial answer** is needed in real-time.
- The reactive controller (and thus the robot) **is stopped** if it must wait for the planner to tell it *where to go*.

The middle layer services.

Planner - Driven Reaction

- Planning can also influence reaction.
- Any **"important" optimizations** the planner discovers are passed down to the reactive controller.
- The planner's suggestions **are used if they are** possible and safe.

  => Who has priority, planner or reactor? It depends, as we will see...

Types of "Reaction ↔ Planning" Interaction

- ♦ **Selection**: Planning is viewed as configuration.
- ♦ **Advising:** Planning is viewed as advice giving.
- ♦ **Adaptation**: Planning is viewed as adaptation of controller.
- ♦ **Postponing**: Planning is viewed as a least commitment process.

Universal Plans

- Suppose for a given problem, **all possible plans are generated for all possible situations in advance**, and stored.
- If for each situation a robot has a pre-existing optimal plan, **it can react optimally**, be reactive and optimal.
- It has a universal plan (These are complete reactive mappings).

Viability of Universal Plans

- A system with a universal plan **is reactive**; the planning is done at compile-time, **not at run-time**.
- Universal plans are **not viable in most domains**, because:
  - the **world** must be deterministic.
  - the **world** must not change.
  - the **goals** must not change.
  - the **world** is too complex (state space is too large).

Planning & Execution

- Planning
  - *Generate* a set of *actions* – **a plan** – that can transform an *initial state* of the world to a *goal state*

    [Newell and Simon, 1950s]
- Execution
  - Start at the initial state, and *perform* each action of a generated plan

Planning Problem

Newell and Simon 1956

- Given the *actions* available in a task domain.
- Given a problem specified as:
  - an initial *state* of the world,
  - a set of *goals* to be achieved.

- Find a *solution* to the problem, i.e., a *way* to transform the initial state into a new state of the world where the goal statement is true.

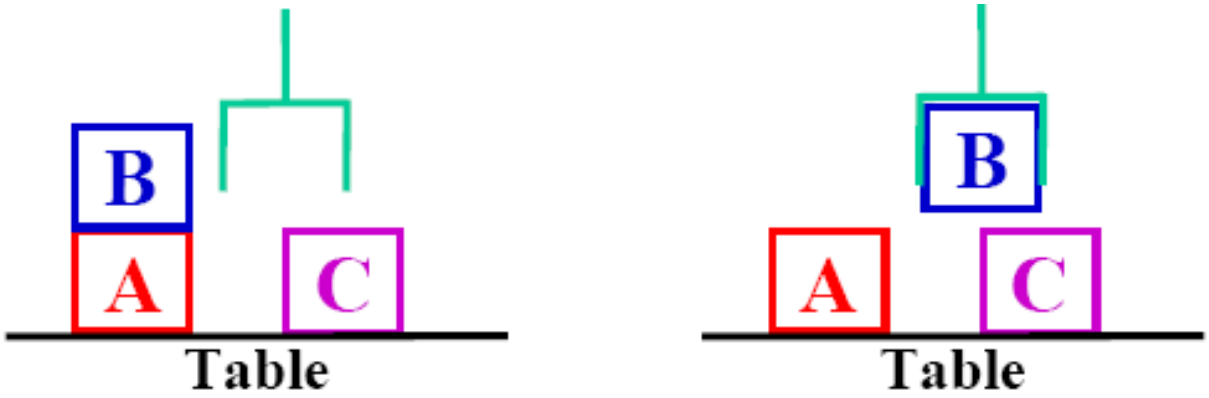
Action Model, State, Goals

Classical Planning

- Action Model: complete, deterministic, correct, rich representation
- State: single initial state, fully known
- Goals: complete satisfaction

Several different planning algorithms

Esempio: Blocks World



- Blocks are picked up and put down by the arm
- Blocks can be picked up only if they are clear, i.e., without any block on top
- The arm can pick up a block only if the arm is empty, i.e., if it is not holding another block, i.e., the arm can be pick up only one block at a time
- The arm can put down blocks on blocks or on the table

STRIPS Model

Pickup_from_table(b)

Pre: Block(b), Handempty
Clear(b), On(b, Table)

Add: Holding(b)

Delete: Handempty,
On(b, Table)

Pickup_from_block(b, c)

Pre: Block(b), Handempty
Clear(b), On(b, c), Block(c)

Add: Holding(b), Clear(c)

Delete: Handempty,
On(b, c)

Putdown_on_table(b)

Pre: Block(b), Holding(b)

Add: Handempty,
On(b, Table)

Delete: Holding(b)

Putdown_on_block(b, c)

Pre: Block(b), Holding(b)
Block(c), Clear(c), $b \neq c$

Add: Handempty, On(b, c)

Delete: Holding(b), Clear(c)

Init: On(a,Table), On(b,table), On(c,table)

Goal: On(a,table),On(b,a), On(c,b)

Spacecraft Domain

Observation-1
- target
- instruments

Observation-2

Observation-3

Observation-4

…

pointing

calibrated

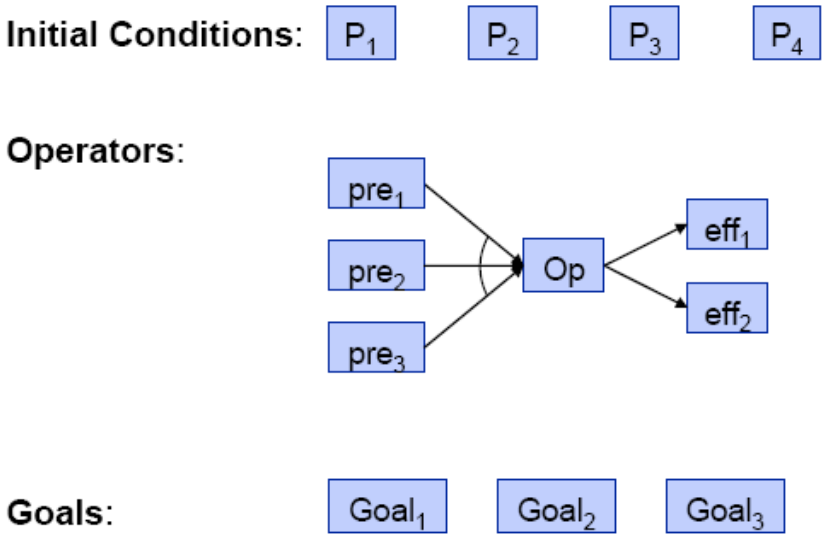
TakeImage (?target, ?instr):
- Pre: Status(?instr, Calibrated), Pointing(?target)
- Eff: Image(?target)

Calibrate (?instrument):
- Pre: Status(?instr, On), Calibration-Target(?target), Pointing(?target)
- Eff: ¬Status(?inst, On), Status(?instr, Calibrated)

Turn (?target):
- Pre: Pointing(?direction), ?direction ≠ ?target
- Eff: ¬Pointing(?direction), Pointing(?target)

Planning Problem

- **Planning Domain:** Descrizione degli operatori in termini di precondizioni ed effetti
- **Planning Problem:** Stato iniziale, Dominio, Goals



Tipi di Planning

- Classical Planning
- Temporal Planning
- Conditional Planning
- Decision Theoretic Planning
- …
- Least-Commitment Planning
- HTN planning
- …

Paradigms

Classical planning

(STRIPS, operator-based, first-principles)

"generative"

Hierarchical Task Network planning

"practical" planning

MDP & POMDP planning

planning under uncertainty

State Space vs. Plan Space

- Planning in the state space:
  - sequence of actions, from the initial state to the goal state
- Planning in the plan space:
  - Sequence of plan transformations, from an initial plan to the final one

Plan-State Search

- Search space is set of *partial plans*
- Plan is tuple $\langle A, O, B \rangle$
  - A: Set of ***actions***, of the form $(a_i : Op_j)$
  - O: Set of ***orderings***, of the form $(a_i < a_j)$
  - B: Set of ***bindings***, of the form $(v_i = C)$, $(v_i \neq C)$, $(v_i = v_j)$ or $(v_i \neq v_j)$
- Initial plan:
  - $\langle \{start, finish\}, \{start < finish\}, \{\} \rangle$
  - *start* has no preconditions; Its effects are the initial state
  - *finish* has no effects; Its preconditions are the goals

State-Space vs Plan-Space

Planning problem

Find a sequence of actions that make instance of the goal true

Nodes in search space

Standard search: node = concrete world state

Planning search: node = partial plan

(Partial) Plan consists of

- Set of operator applications S_i
- Partial (temporal) order constraints $S_i \prec S_j$
- Causal links $S_i \xrightarrow{c} S_j$

  Meaning: "S_i achieves $c \in precond(S_j)$" (record purpose of steps)

Search in the Plan-Space

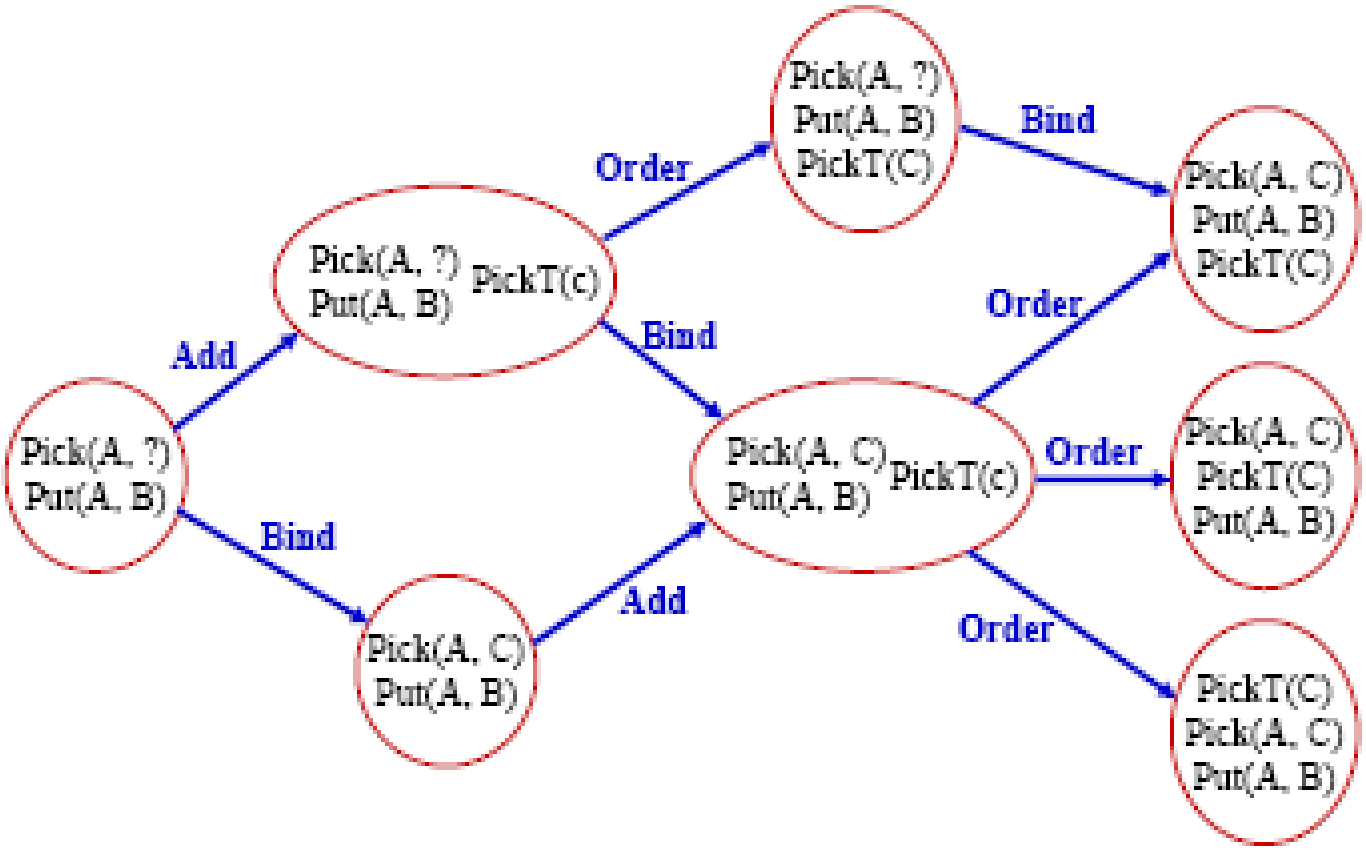
Operators on partial plans

- add an action and a causal link to achieve an open condition
- add a causal link from an existing action to an open condition
- add an order constraint to order one step w.r.t. another

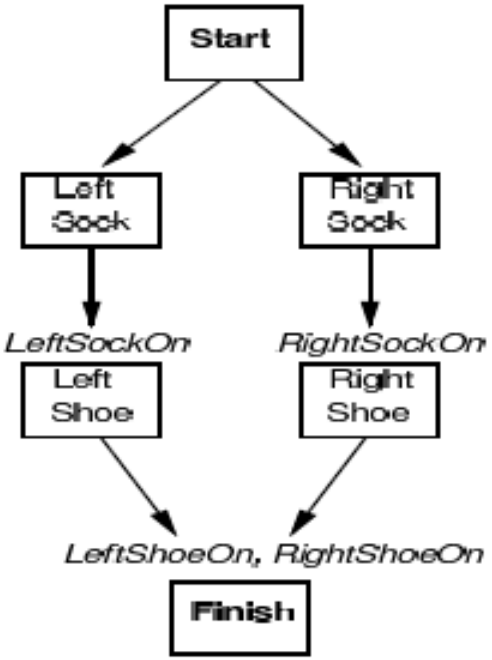
Open condition

A precondition of an action not yet causally linked

Plan-State Search



Partially-Ordered Plans



Special steps with empty action

Start no precond, initial assumptions as effect)

Finish goal as precond, no effect

Partial-Order Plans

Complete plan

A plan is complete iff every precondition is achieved

A precondition c of a step S_j is achieved (by S_i) if

- $S_i \prec S_j$
- $c \in effect(S_i)$
- **there is no S_k with $S_i \prec S_k \prec S_j$ and $\neg c \in effect(S_k)$**
  (otherwise S_k is called a clobberer or threat)

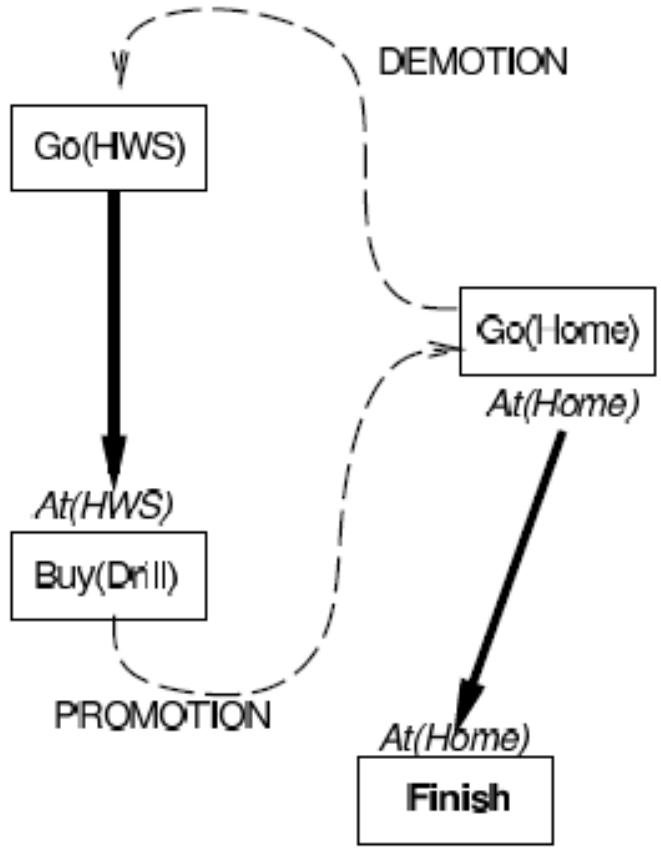
Clobberer / threat

A potentially intervening step that destroys the condition achieved by a causal link

Partial-Order Plans

Example

$Go(Home)$ **clobbers** $At(HWS)$



Demotion

Put before $Go(HWS)$

Promotion

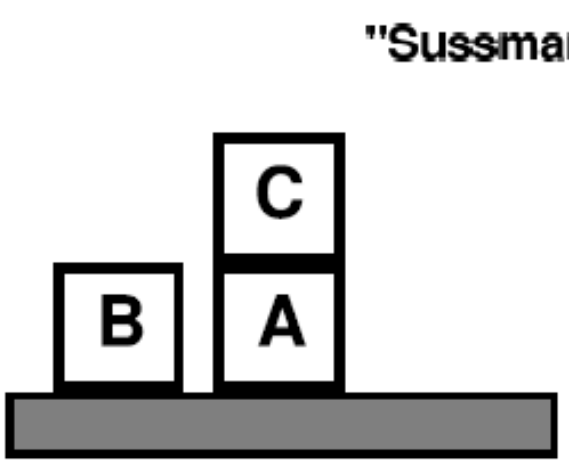
Put after $Buy(Drill)$

General Approach

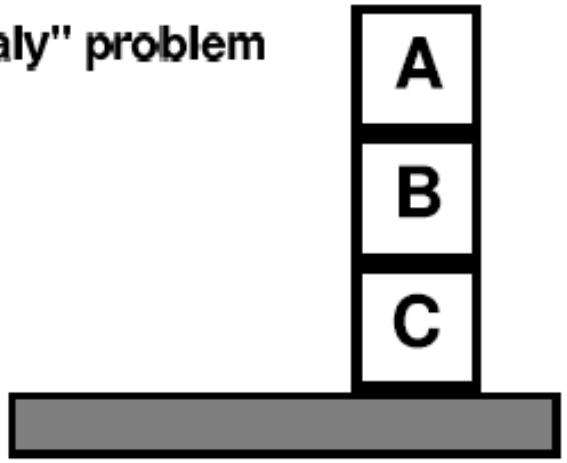
- General Approach
  - Find unachieved precondition
    - Add new action *or* link to existing action
  - Determine if conflicts occur
    - Previously achieved precondition is "clobbered"
    - Fix conflicts (reorder, bind, …)
- Partial-order planning can easily (and optimally) solve blocks world problems that involve goal interactions (e.g., the "Sussman Anomaly" problem)

Blocks World

"Sussman anomaly" problem



Start State



Goal State

Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)
Clear(z) On(x,y)

Clear(x) On(x,z)

PutOnTable(x)

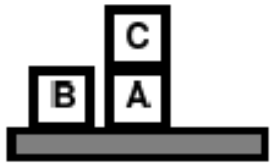
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Blocks World

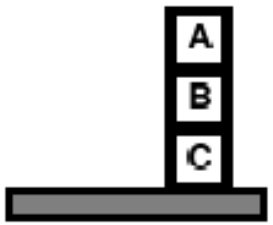
START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

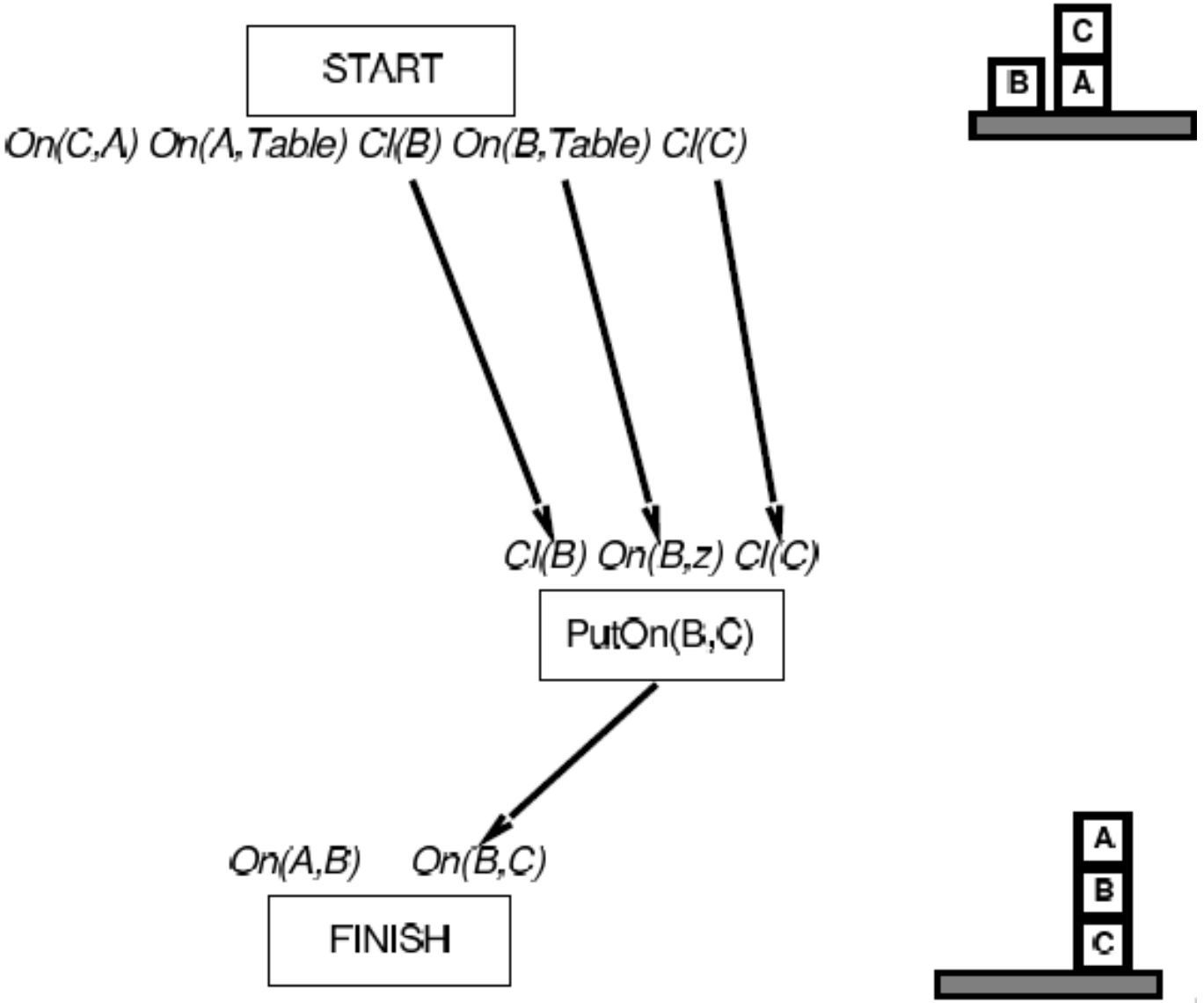


On(A,B) On(B,C)

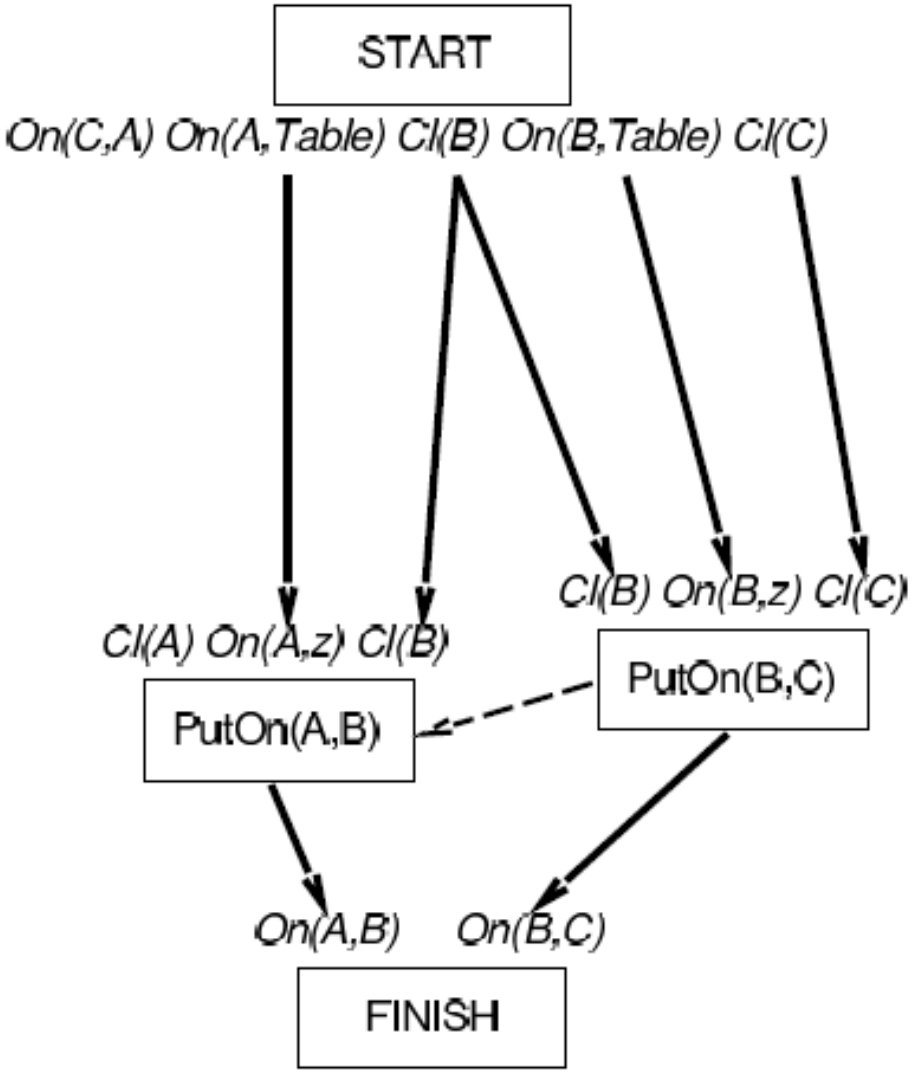
FINISH

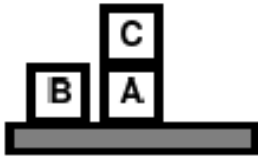


Blocks World

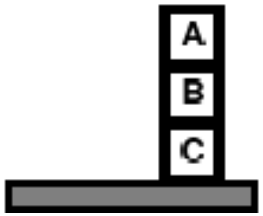


Blocks World

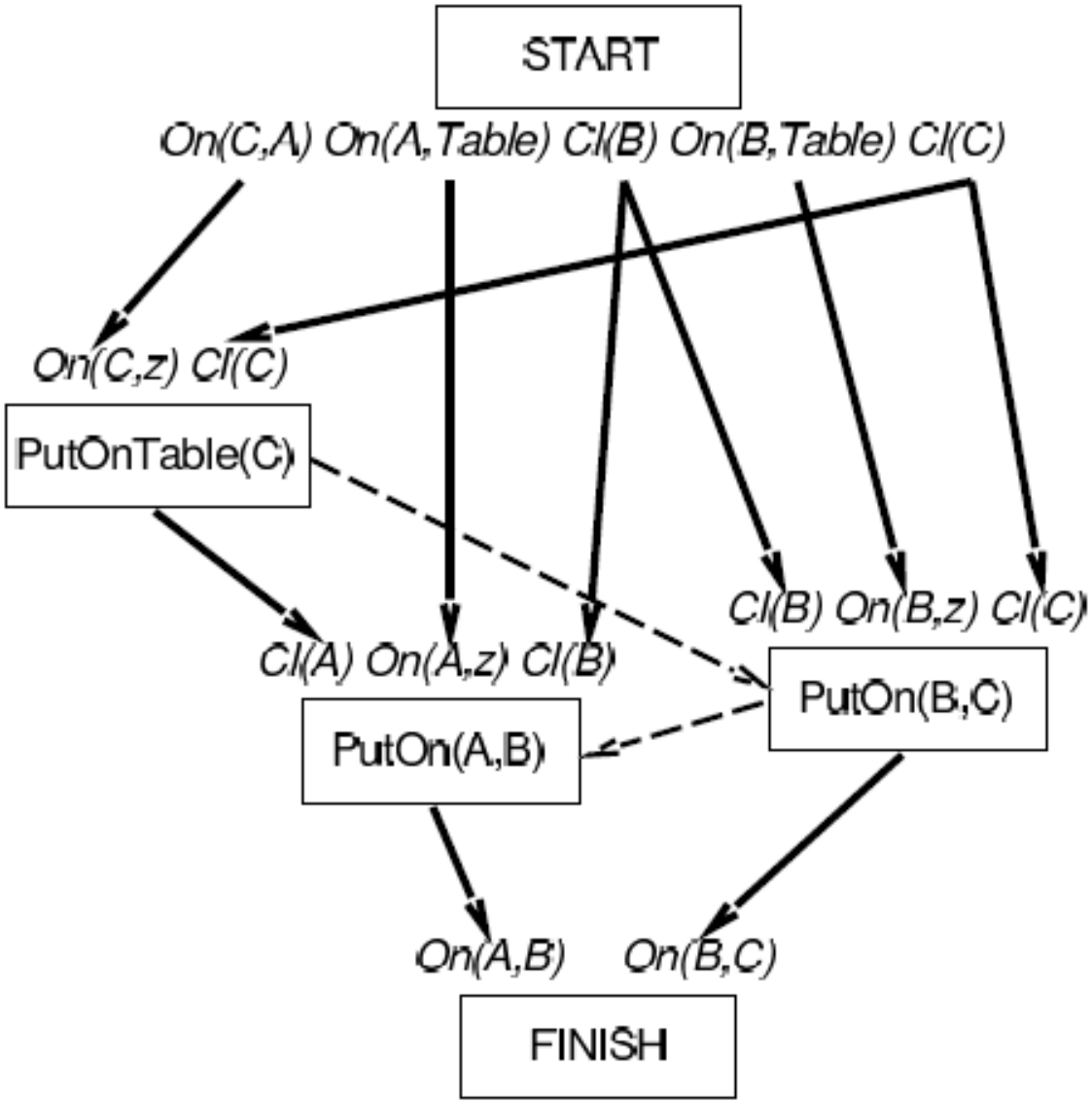


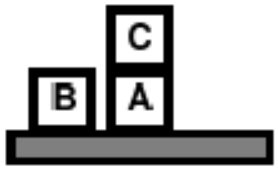


PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)



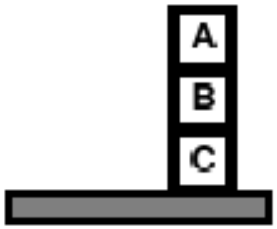
Blocks World



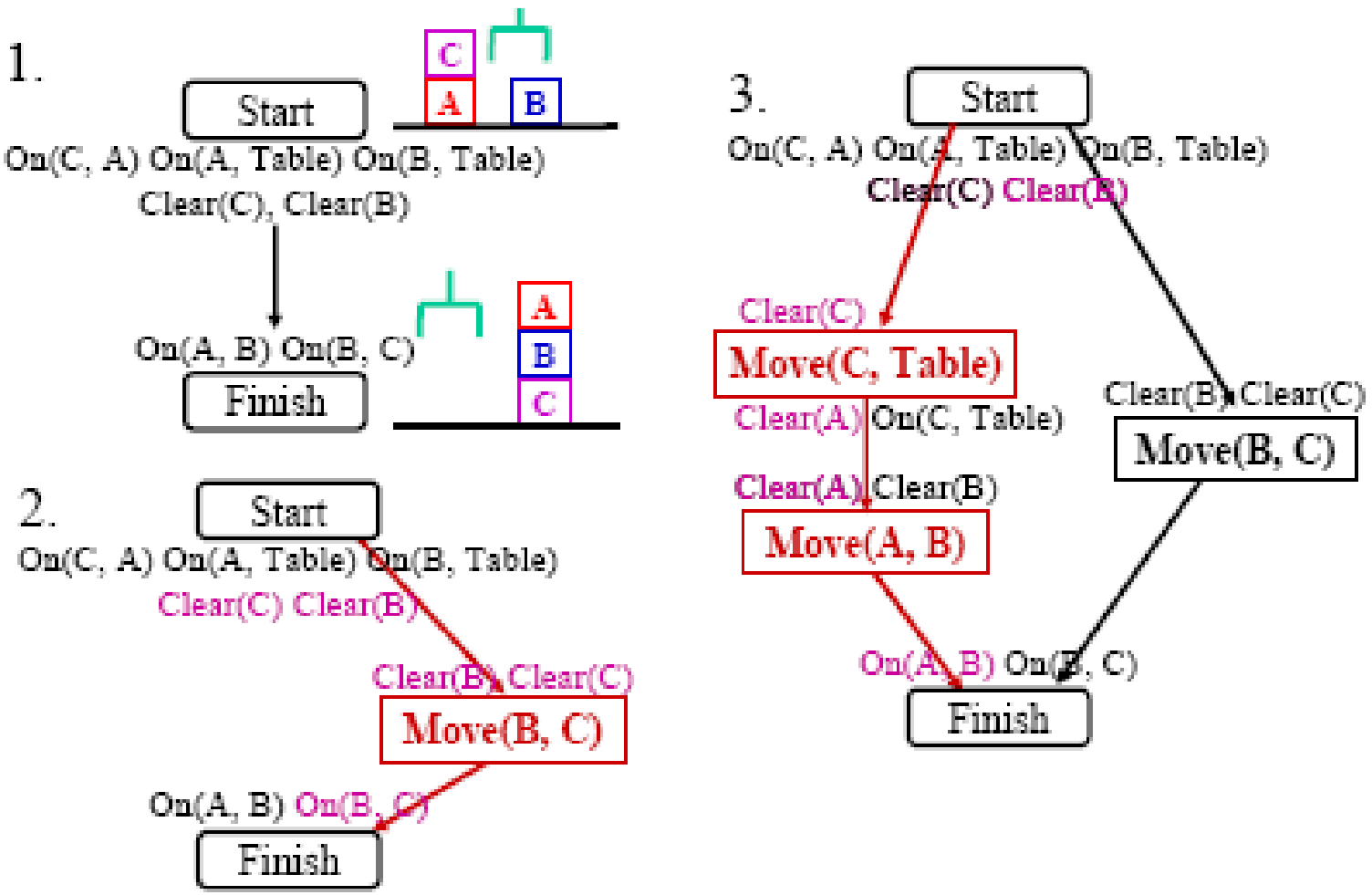


PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

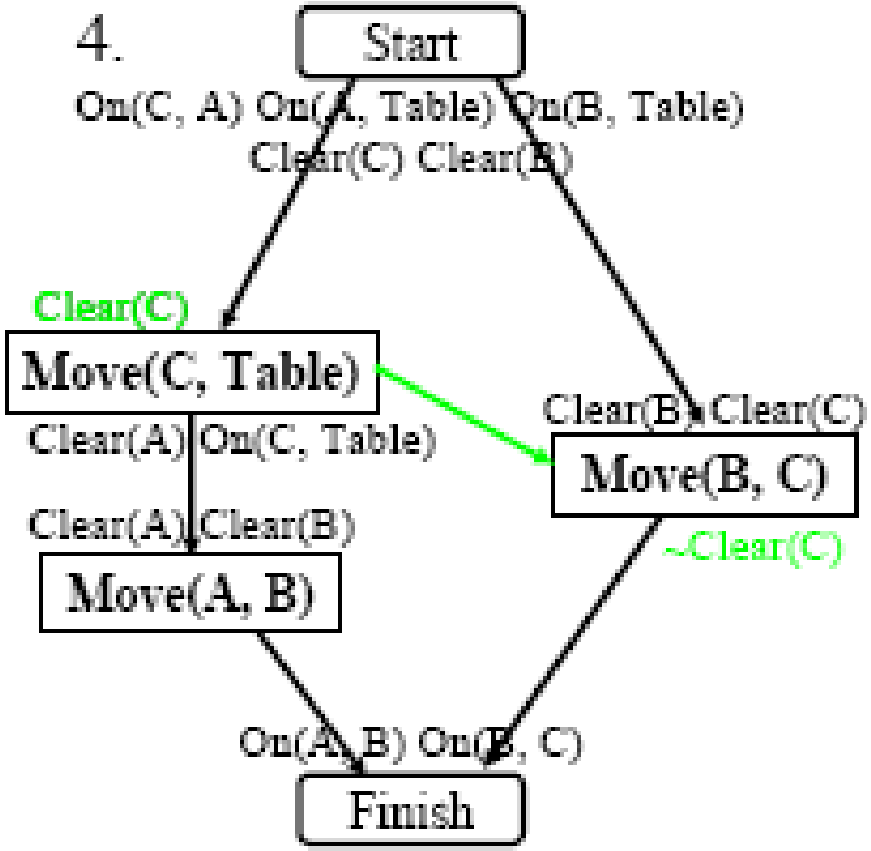
PutOn(B,C) clobbers Cl(C) => order after PutOnTable(C)

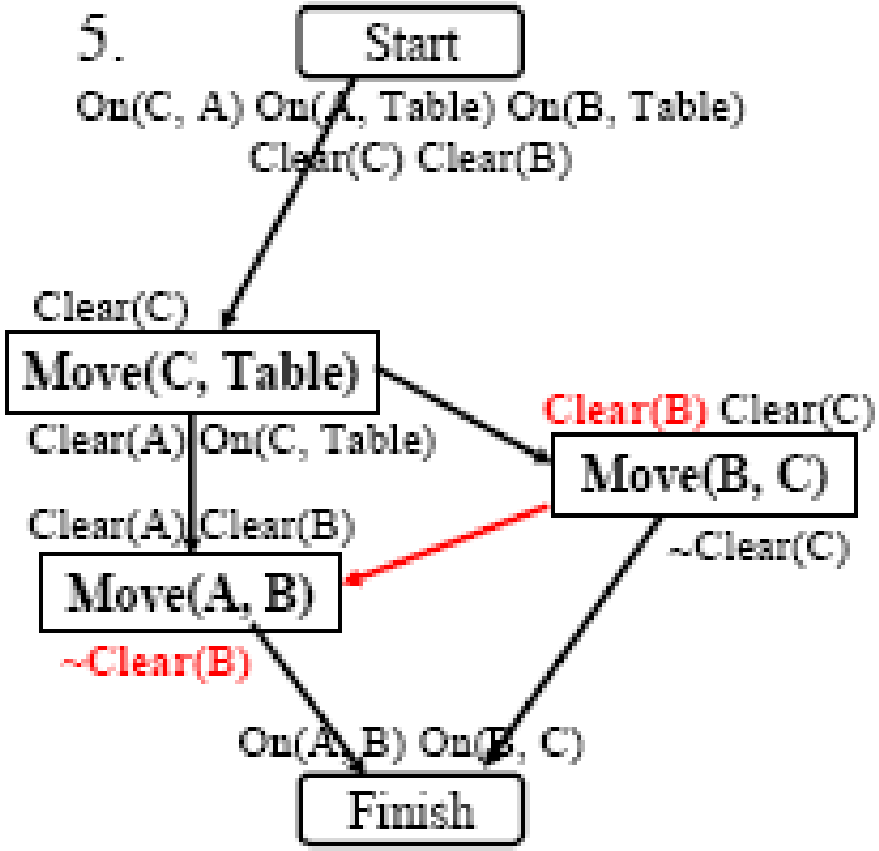


Blocks World



Blocks World





Least Commitment

- Basic Idea
  - *Make choices that are only relevant to solving the current part of the problem*

- Least Commitment Choices
  - **Orderings**: Leave actions unordered, unless they must be sequential
  - **Bindings**: Leave variables unbound, unless needed to unify with conditions being achieved
  - **Actions**: Usually not subject to "least commitment"
- Refinement
  - Only *add* information to the current plan
  - *Transformational* planning can remove choices

Terminology

- ***Totally Ordered*** Plan
  - There exists sufficient orderings O such that all actions in A are ordered with respect to each other
- ***Fully Instantiated*** Plan
  - There exists sufficient constraints in B such that all variables are constrained to be equal to some constant
- ***Consistent*** Plan
  - There are no contradictions in O or B
- ***Complete*** Plan
  - Every precondition p of every action a_i in A is ***achieved***: There exists an effect of an action a_j that comes before a_i and unifies with p, and no action a_k that deletes p comes between a_j and a_i

POP-Algorithm

function POP(*initial, goal, operators*) **returns** *plan*

```
plan ← MAKE-MINIMAL-PLAN(initial, goal)
loop do
    if SOLUTION?(plan) then return plan        % complete and consistent
    S_need, c ← SELECT-SUBGOAL(plan)
    CHOOSE-OPERATOR(plan, operators, S_need, c)
    RESOLVE-THREATS(plan)
end
```

---

function SELECT-SUBGOAL(*plan*) **returns** S_{need}, c

```
pick a plan step S_need from STEPS(plan)
    with a precondition c that has not been achieved
return S_need, c
```

POP-Algorithm

procedure CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, c)

choose a step S_{add} from *operators* or STEPS(*plan*) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step from *operators* **then**
    add S_{add} to STEPS(*plan*)
    add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

POP-Algorithm

procedure RESOLVE-THREATS(*plan*)

  for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**

    choose either

      Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)

      Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)

    if not CONSISTENT(*plan*) **then fail**

  end

POP-Algorithm

- Non-deterministic search for plan,
  backtracks over choicepoints on failure:
  - choice of S_{add} to achieve S_{need}
  - choice of promotion or demotion for clobberer
- Sound and complete
- There are extensions for:
  disjunction, universal quantification, negation, conditionals
- Efficient with good heuristics from problem description
  But: very sensitive to subgoal ordering
- Good for problems with loosely related subgoals

POP-Algorithm

- **Advantages**
  - Partial order planning is ***sound*** and ***complete***
  - Typically produces ***optimal*** solutions (plan length)
  - Least commitment may lead to shorter search times

- **Disadvantages**
  - Significantly more complex algorithms (higher *per-node* cost)
  - Hard to determine what is true in a state
  - Larger search space (**infinite**!)

Plan Monitoring

Execution monitoring

Failure: Preconditions of remaining plan not met

Action monitoring

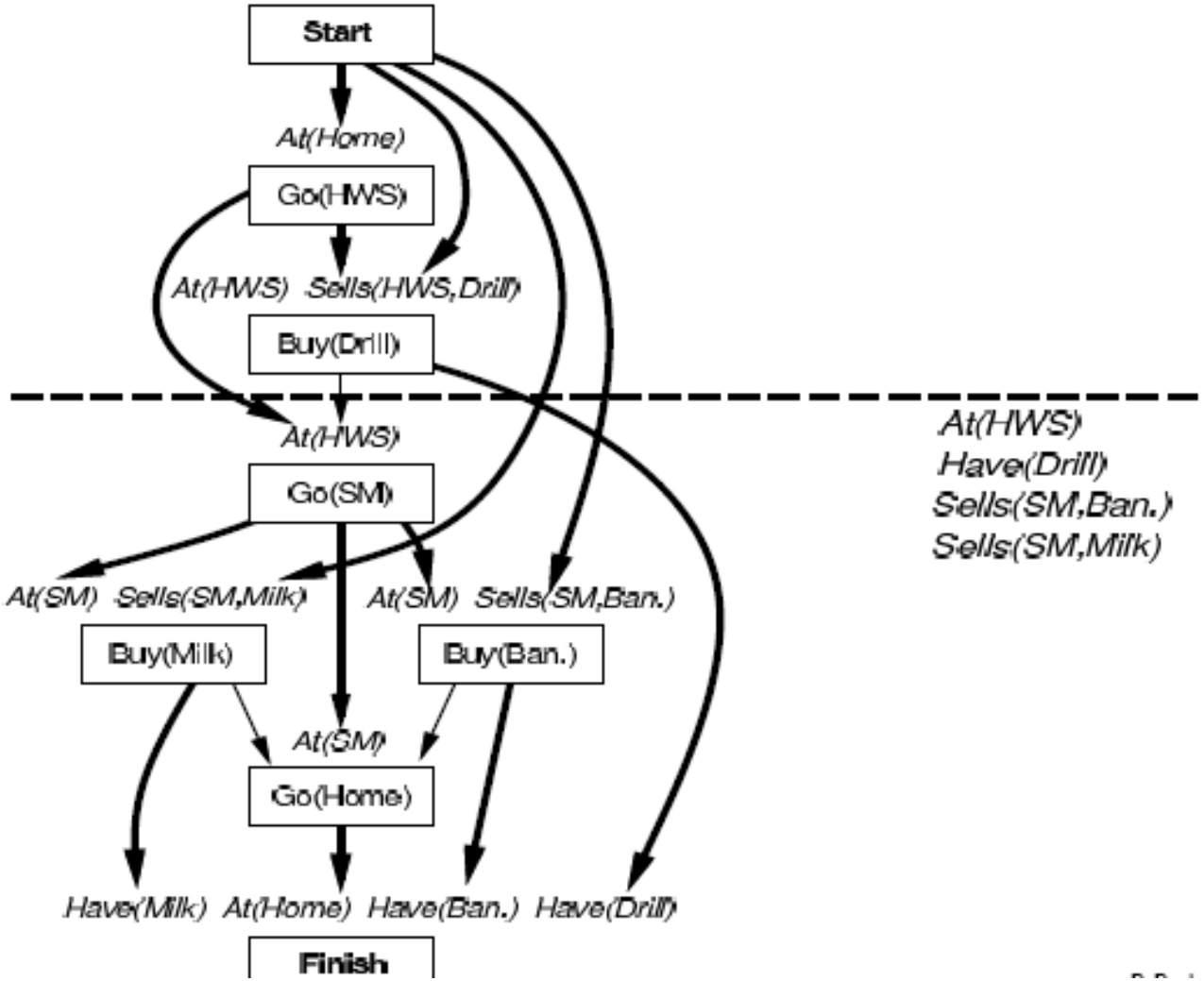
Failure: Preconditions of next action not met

(or action itself fails, e.g., robot bump sensor)

Consequence of failure

Need to replan

Preconditions for the rest of the plan



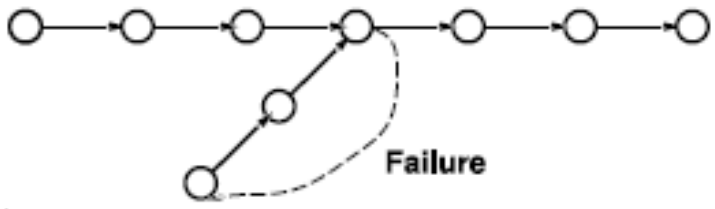
Replanning

Simplest

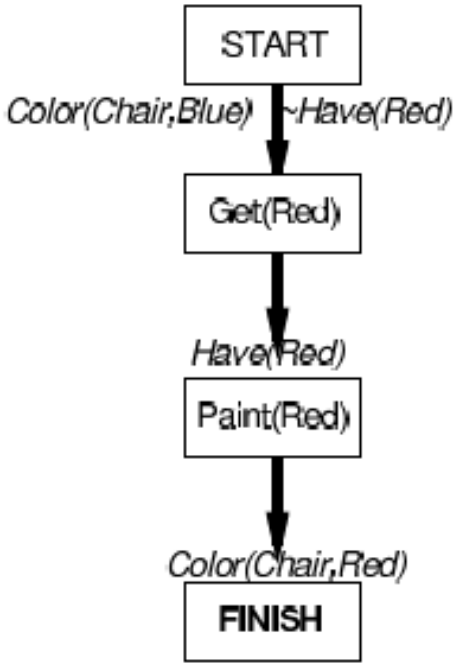
On failure, replan from scratch

Better

Plan to get back on track by reconnecting to best continuation



Replanning



PRECONDITIONS	FAILURE RESPONSE
none	N/A
Have(Red)	Fetch more red
Color(Chair,Red)	Repaint