Hierarchical Refinement as a Generalization of HTN Planning

Dana Nau
University of Maryland

work performed with

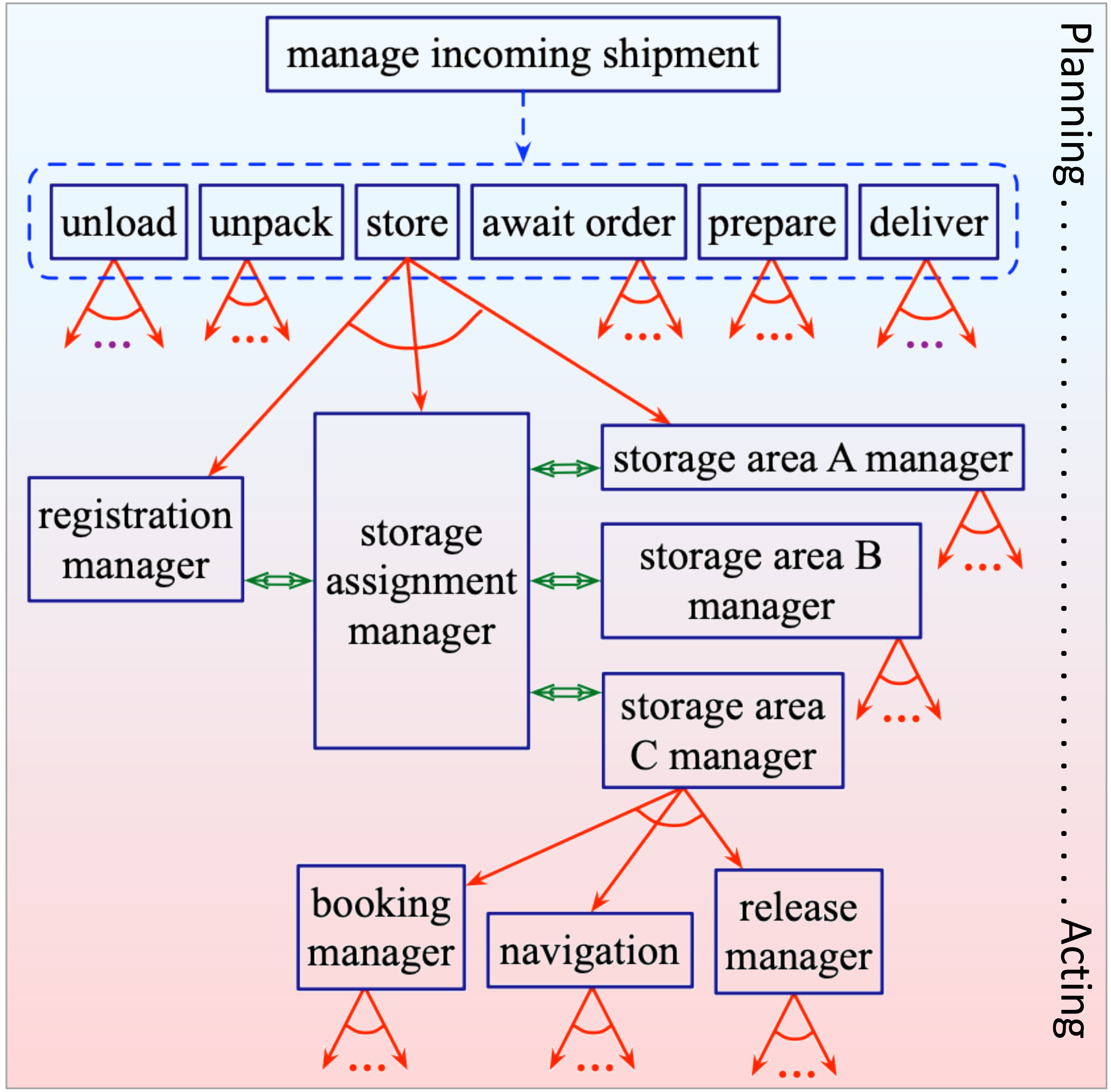
- Sunandita Patra University of Maryland
- Malik Ghallab LAAS/CNRS, University of Toulouse
- Paolo Traverso FBK ICT IRST, Trento, Italy
- James Mason University of Maryland

http://www.cs.umd.edu/~nau/papers/nau2019hierarchical.pdf

Bremen Harbor

Harbor Management Tasks

- Multiple levels of abstraction
  - Physical/managerial organization of harbor
- Upper levels:
  - Abstract tasks, can be planned in advance
- Lower levels:
  - Multiple agents
  - Partial observability
  - Dynamic change
- Continual online planning
  - Abstract and partial until more detail needed



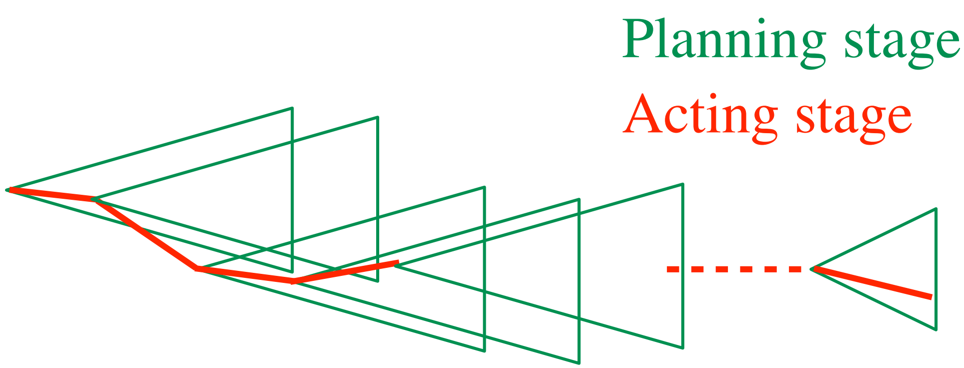
Acting and Planning

Acting

- *Performing* tasks and actions
- Use *operational models* that tell *how*
  - Dynamic, unpredictable environment
  - Adapt to context, react to events

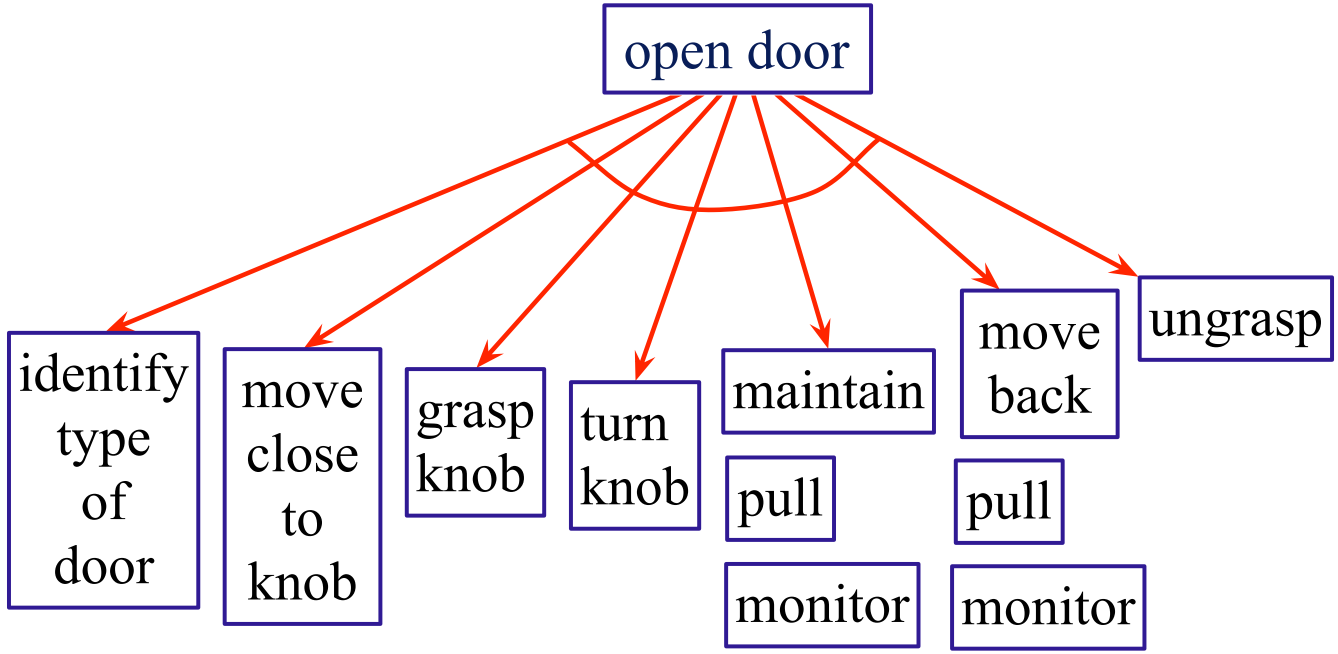
Planning

- *Prediction + search*
- Search over predicted states, possible organizations of tasks and actions
- Use *descriptive models* to predict *what*



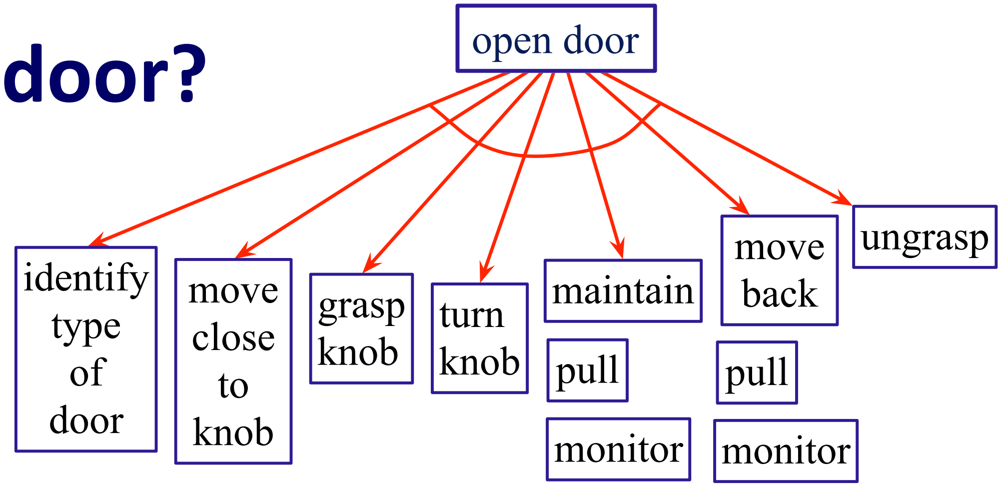
- Planning in service of acting
  - Actor asks planner for advice
- Planner runs online
  - e.g., receding horizon

Opening a Door

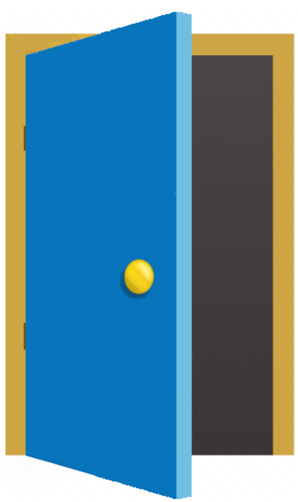


- Details depends on what kind of door
  - Might not be known until acting time

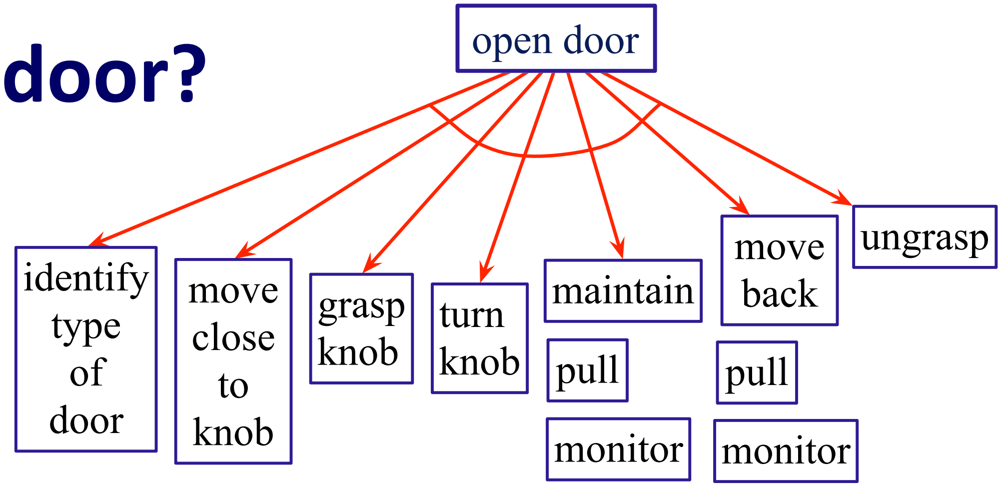
What kind of door?



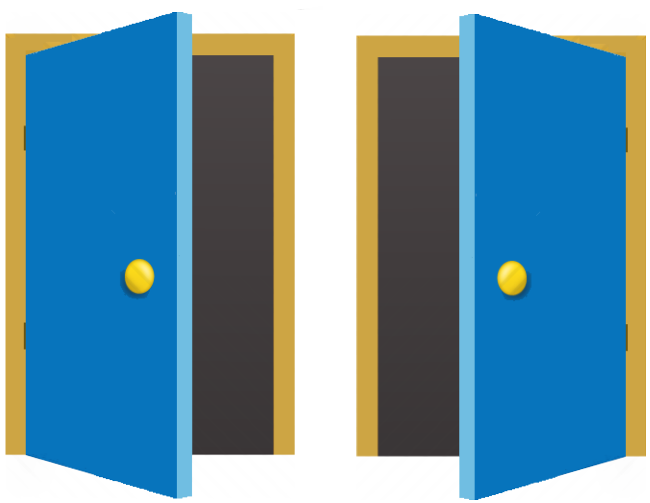
- Sliding or hinged



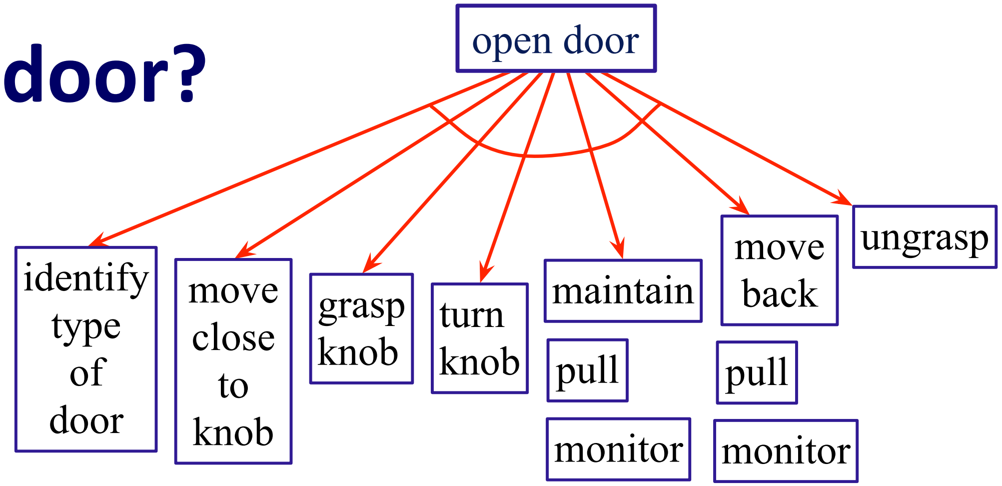
What kind of door?



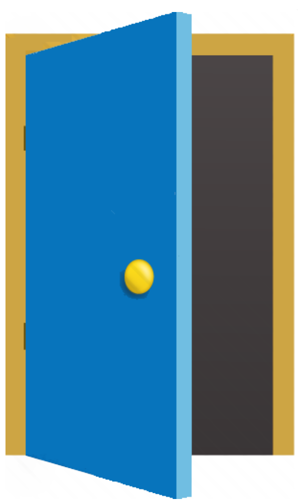
- Sliding or hinged
- Hinge on left or right

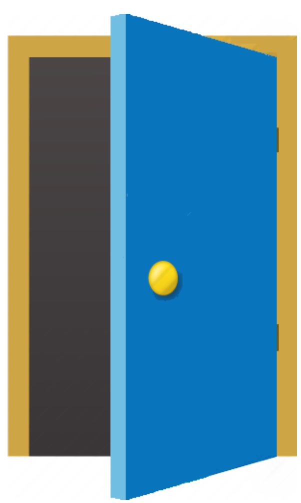


What kind of door?

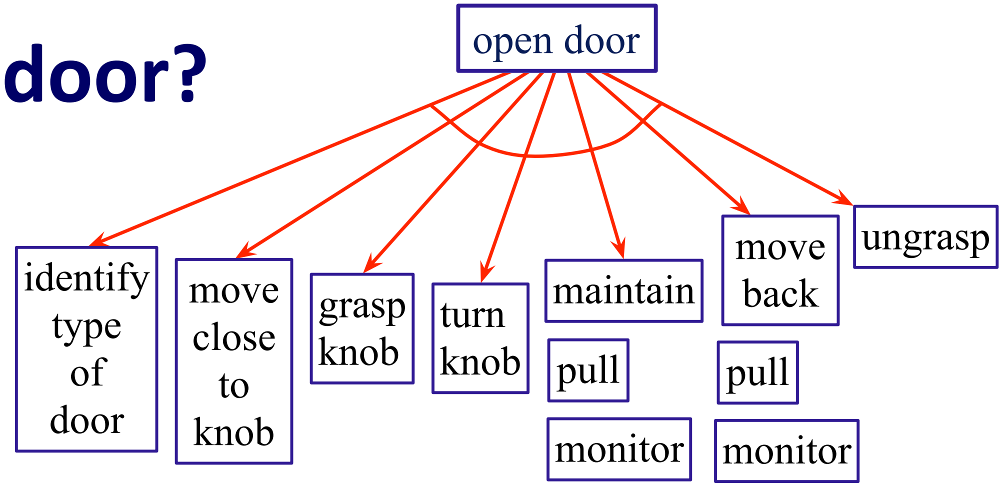


- Sliding or hinged
- Hinge on left or right
- Open toward or away

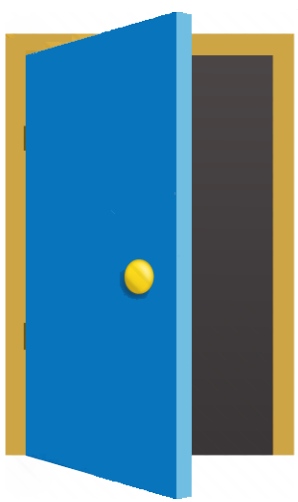


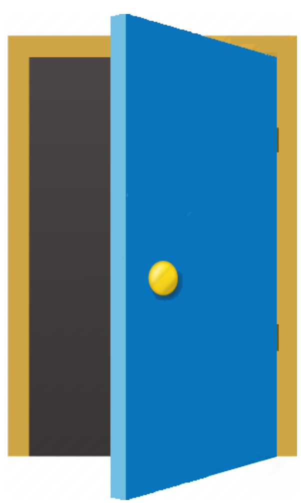


What kind of door?

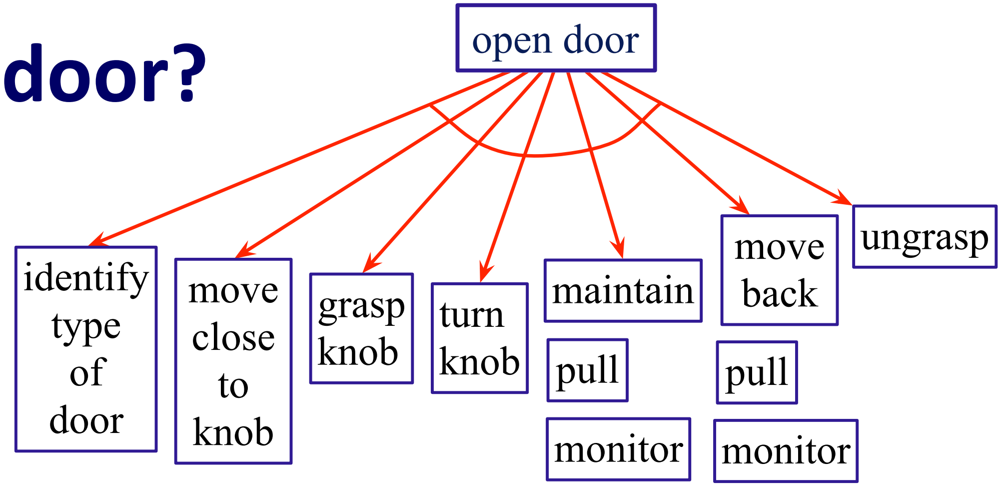


- Sliding or hinged
- Hinge on left or right
- Open toward or away
- Knob, lever,

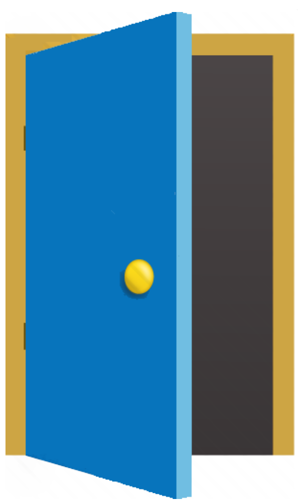


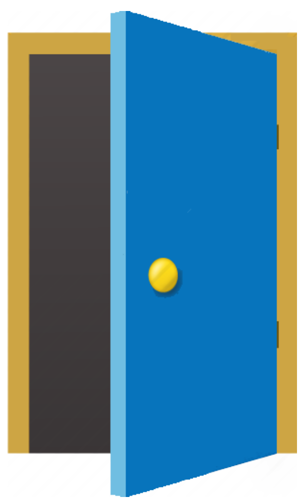


What kind of door?

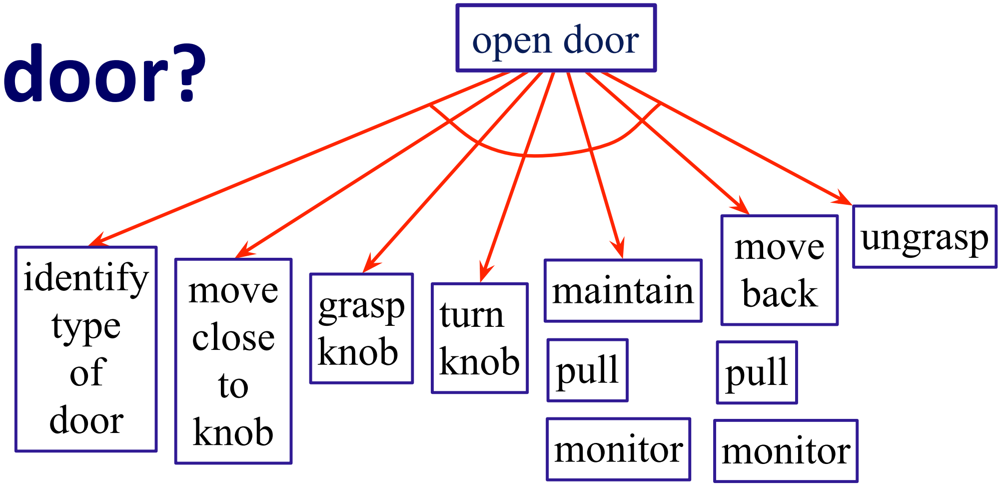


- Sliding or hinged
- Hinge on left or right
- Open toward or away
- Knob, lever,
  push bar, push plate,

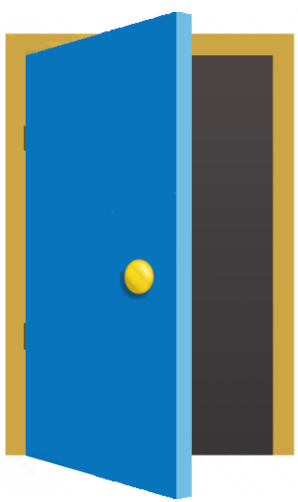


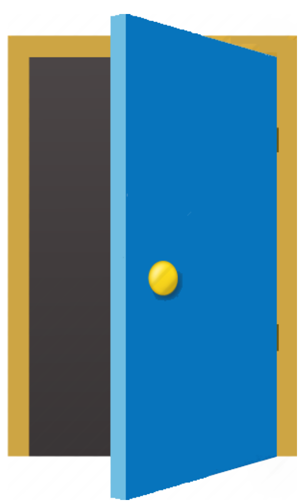


What kind of door?

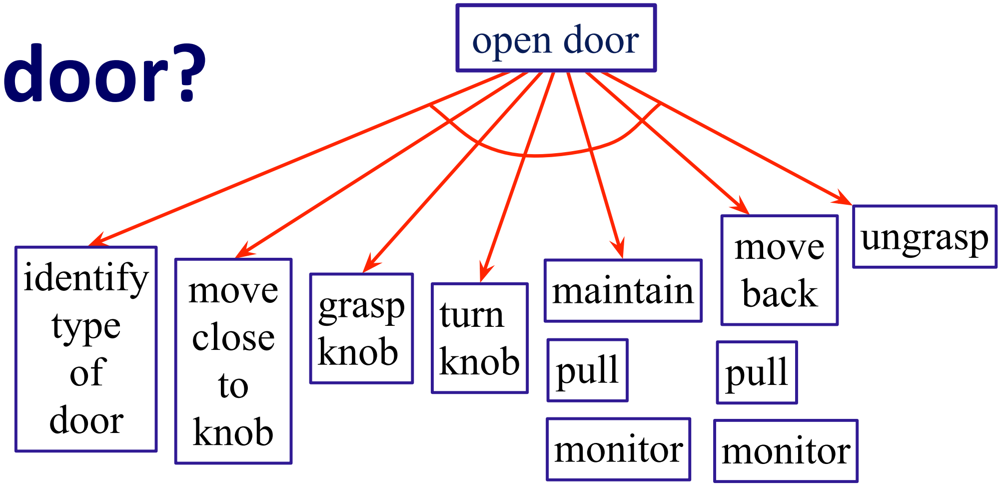


- Sliding or hinged
- Hinge on left or right
- Open toward or away
- Knob, lever,
  push bar, push plate,
  pull handle, thumb latch,

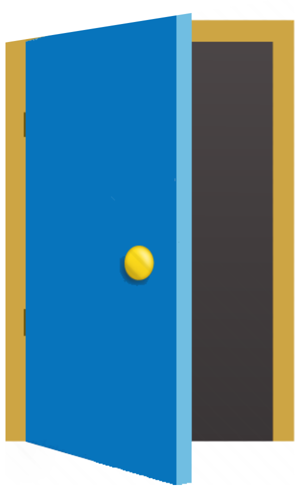


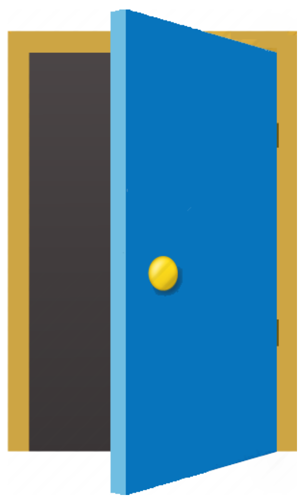


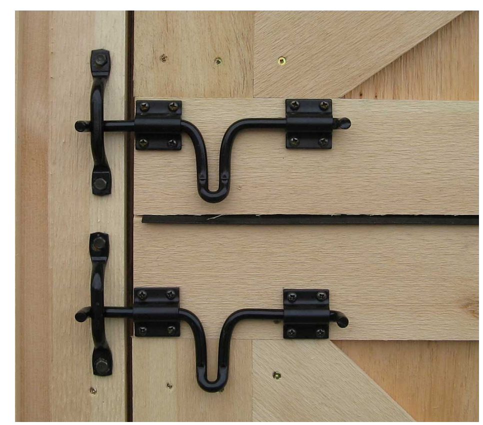
What kind of door?



- Sliding or hinged
- Hinge on left or right
- Open toward or away
- Knob, lever,
  push bar, push plate,
  pull handle, thumb latch,
  something else

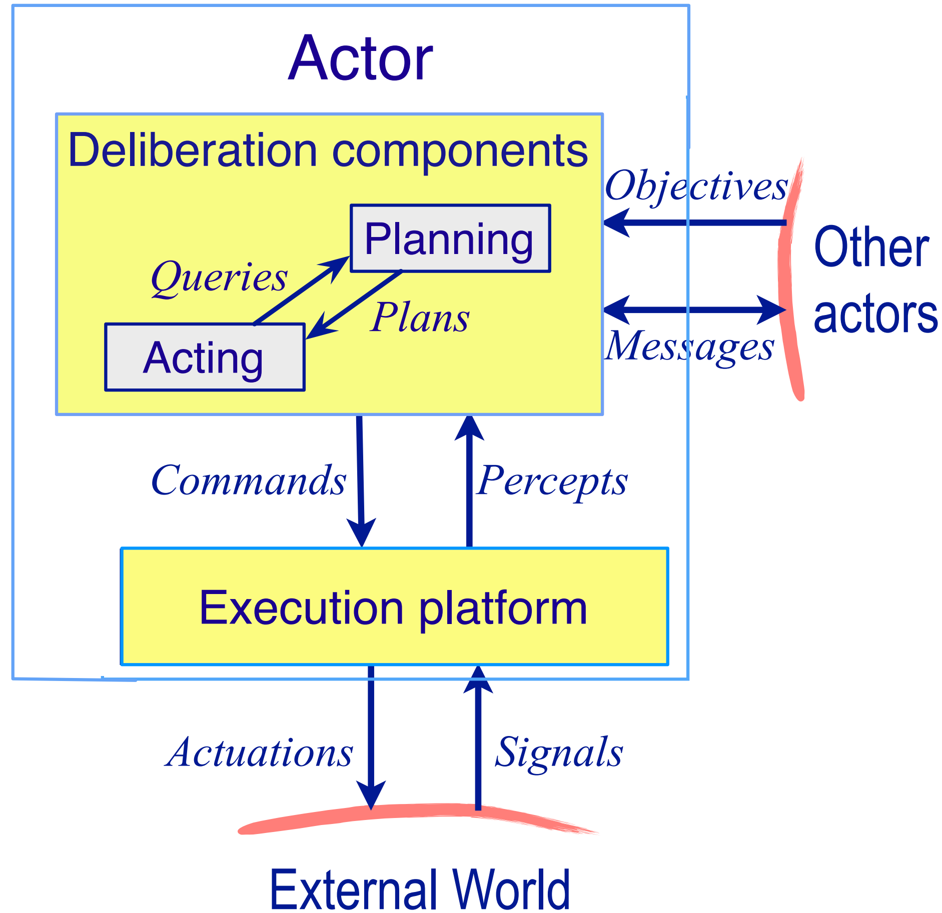






Refinement Acting

- *Task*:
  - ➢ activity for the actor to perform
- For each task, one or more *refinement methods*
  - ➢ Operational models telling how to perform the task



method-name(*arg*$_1$, …, *arg*$_k$)
    task: *task-identifier*
    pre: *test*
    body: *computer program that may include tasks and commands*

"primitive" functions that the actor can send to its execution platform

Refinement Acting

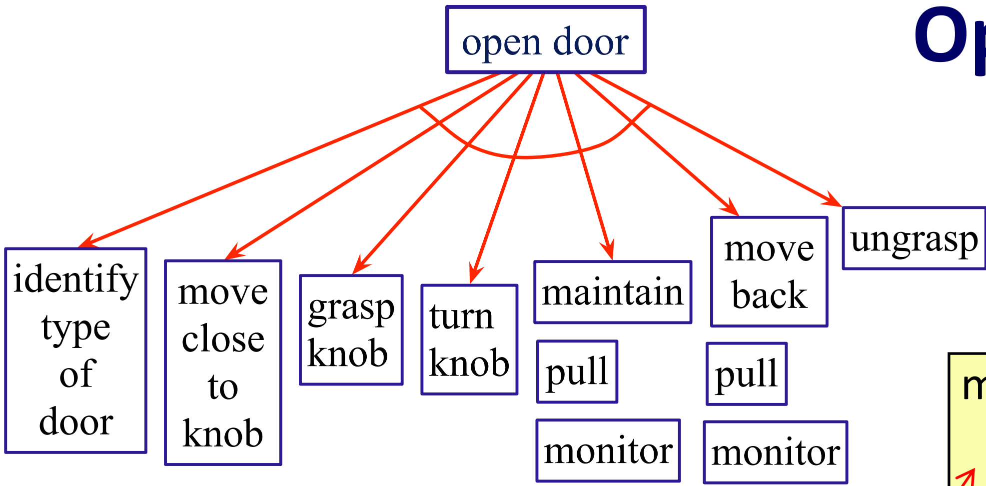
- *Task*:
  - activity for the actor to perform
- For each task, one or more *refinement methods*
  - Operational models telling how to perform the task

method-name($arg_1, \ldots, arg_k$)
task: *task-identifier*
pre: *test*
body: *computer program that may include tasks and commands*

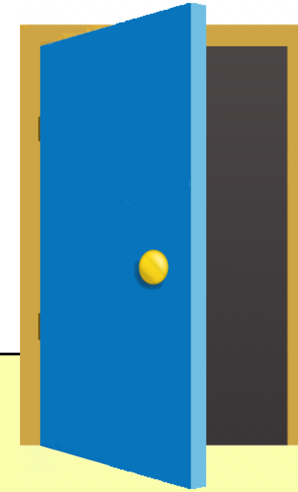
- Differences from HTN methods
  - Actor uses them reactively
  - Body is a computer program that invokes tasks, commands
  - Commands interact with external world
  - Outcomes not known in advance
  - Current state obtained using sensors, represented using state variables

"primitive" functions that the actor can send to its execution platform

Opening a Door



- What kind:
  - Hinged on left, opens toward us, knob



m-opendoor(*r*,*d*,*l*,*h*)
- task: opendoor(*r*,*d*)
- pre: loc(*r*) = *l* ∧ road(*l*,*d*) ∧ handle(*d*,*h*)
- body:
  - while ¬reachable(*r*,*h*) do
    - move-close(*r*,*h*)
  - monitor-status(*r*,*d*)
  - if door-status(*d*) = closed then
    - unlatch(*r*,*d*)
  - throw-wide(*r*,*d*)
  - end-monitor-status(*r*,*d*)

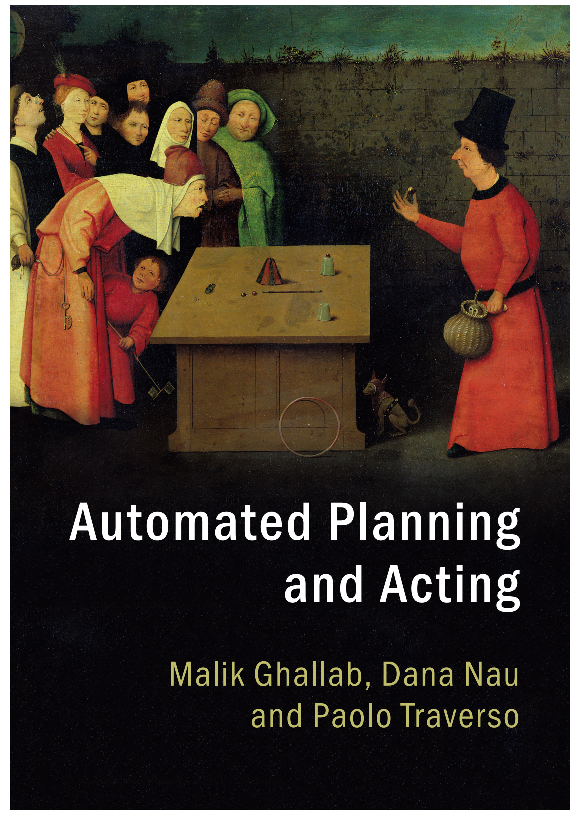
m1-unlatch(*r*,*d*,*l*,*o*)
- task: unlatch(*r*,*d*)
- pre: loc(*r*,*l*) ∧ toward-side(*l*,*d*) ∧ side(*d*,left) ∧ type(*d*,rotate) ∧ handle(*d*,*o*)
- body:
  - grasp(*r*,*o*)
  - turn(*r*,*o*,alpha1)
  - pull(*r*,val1)
  - if door-status(*d*) = cracked then ungrasp(*r*,*o*)
  - else fail

m1-throw-wide(*r*,*d*,*l*,*o*)
- task: throw-wide(r,*d*)
- pre: loc(*r*,*l*) ∧ toward-side(*l*,*d*) ∧ side(*d*,left) ∧ type(*d*,rotate) ∧ handle(*d*,*o*) ∧ door-status(*d*) = cracked
- body:
  - grasp(*r*,*o*)
  - pull(*r*,val1)
  - move-by(*r*,val2)

RAE (Reactive Acting Engine)

- Uses refinement methods to accomplish tasks
  - Based on OpenPRS robot control architecture

- I'll give a summary
- For details:
  - Ghallab, Nau, and Traverso, *Automated Planning and Acting,* Cambridge University Press, 2016
  - Final manuscript and lecture slides freely downloadable here



RAE (Reactive Acting Engine)

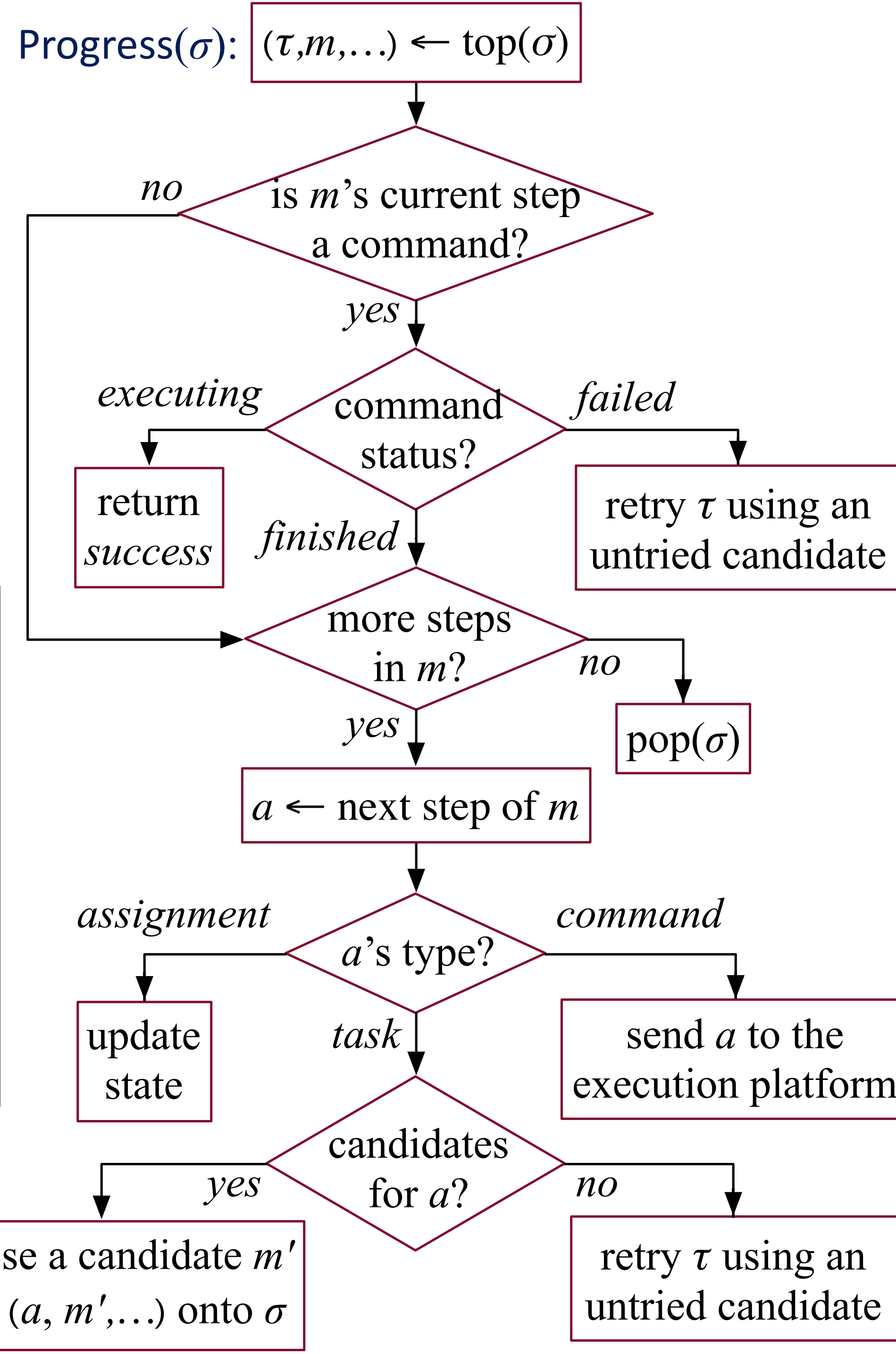
Agenda: $\{stack\ \sigma_1, \ldots, stack\ \sigma_n\}$

Like program execution stacks

stack =

(sub-subtask, method, …)
(subtask, method, …)
(task, method, …)

loop:
- for every new external task or event τ
  - *Candidates* = {applicable method instances}
  - choose $m \in$ *Candidates*
  - create refinement stack σ, *initially just* $\langle\tau,m\rangle$
  - add σ to *Agenda*
- for each stack σ in *Agenda*
  - Progress(σ)
  - if σ is finished, remove it from *Agenda*



RAE (Reactive Acting Engine)

Agenda: $\{stack\ \sigma_1, \ldots, stack\ \sigma_n\}$

Like program execution stacks

stack =	(sub-subtask, method, …)
	(subtask, method, …)
	(task, method, …)

loop:

  for every new external task or event τ

    Candidates = {applicable method instances}

    choose $m \in$ *Candidates*

    create refinement stack σ, *initially just* $\langle\tau,m\rangle$

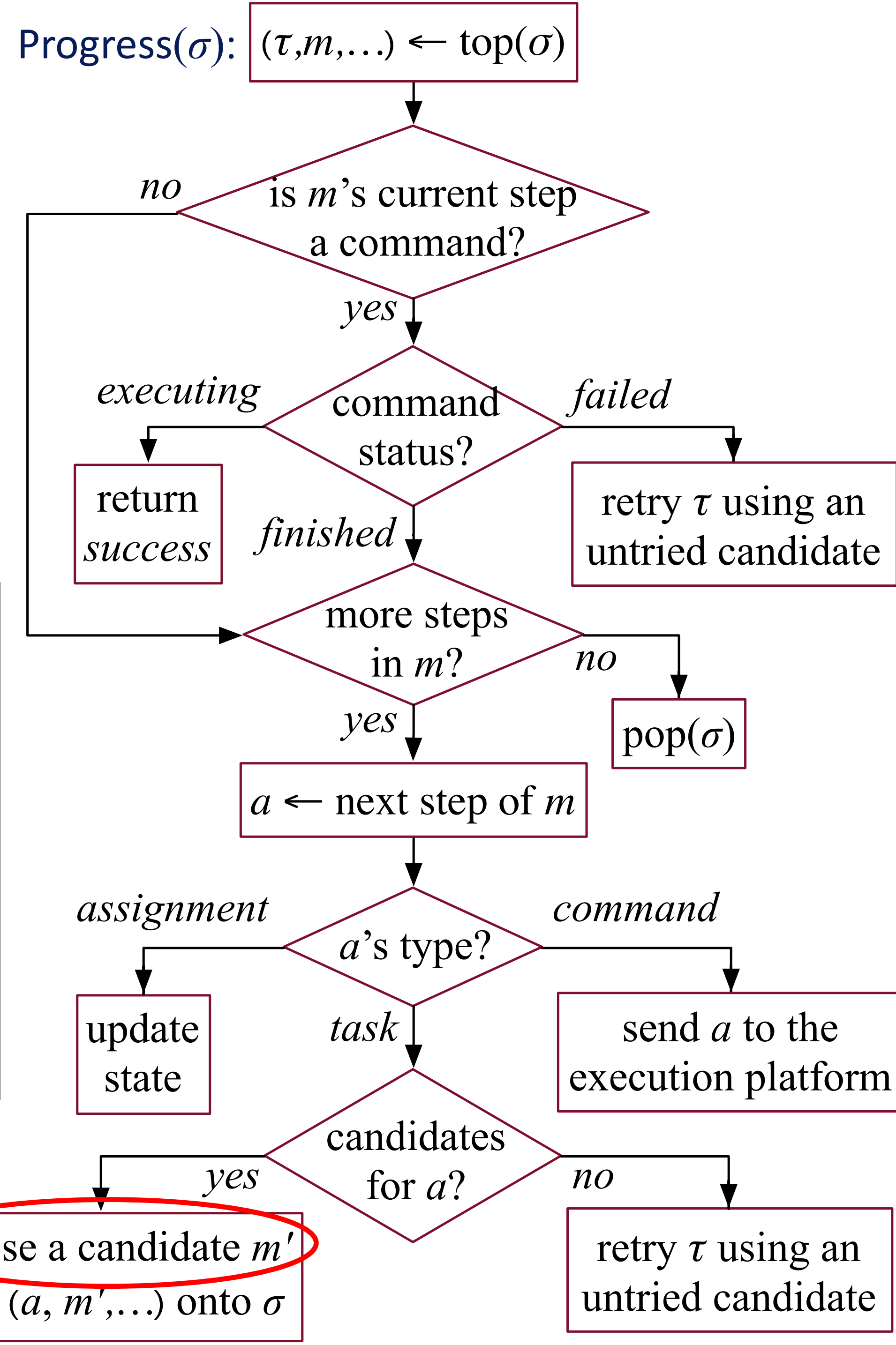
    add σ to *Agenda*

  for each stack σ in *Agenda*

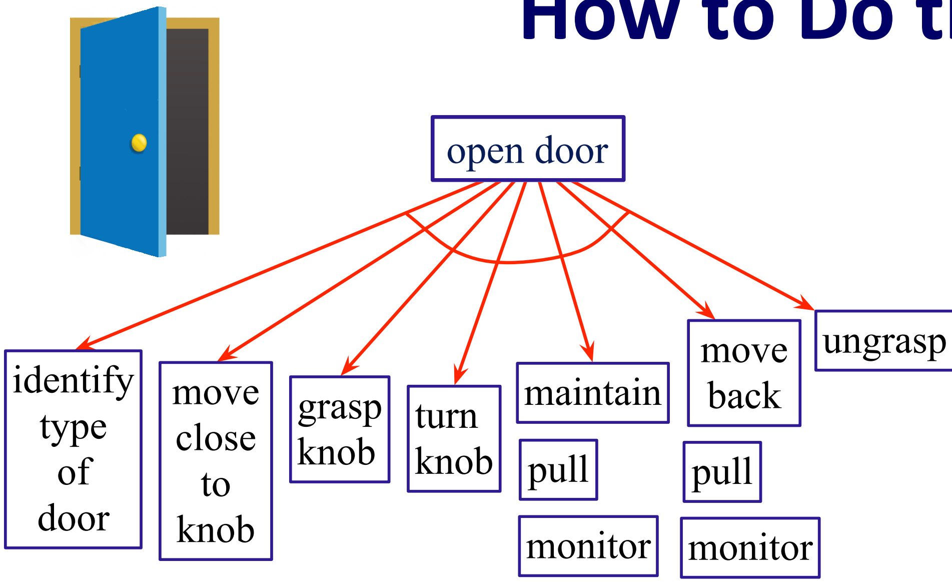
    Progress(σ)

    if σ is finished, remove it from *Agenda*

- Get advice from a planner



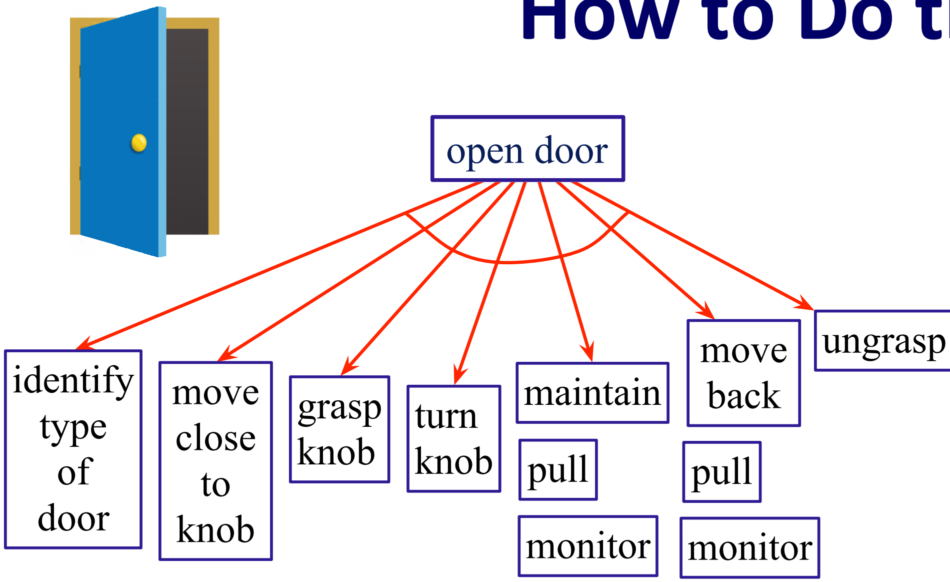
How to Do the Planning?



m-opendoor(r,d,l,h)
  task: opendoor(r,d)
  pre: loc(r) = l ∧ road(l,d)
    ∧ handle(d,h)
  body:
    while ¬reachable(r,h) do
      move-close(r,h)
    monitor-status(r,d)
    if door-status(d) = closed then
      unlatch(r,d)
    throw-wide(r,d)
    end-monitor-status(r,d)

- In the book: SeRPE planner
  - Extends the SHOP algorithm to reason about refinement methods
  - Executes code in the method's body, but
    - Descriptive models of commands
    - Classical actions, abstract state
  - To backtrack from a method *m*:
    - Revert to state when *m* was chosen

How to Do the Planning?




```
m-opendoor(r,d,l,h)
    task: opendoor(r,d)
    pre:  loc(r) = l ∧ road(l,d)
          ∧ handle(d,h)
    body:
        while ¬reachable(r,h) do
            move-close(r,h)
        monitor-status(r,d)
        if door-status(d) = closed then
            unlatch(r,d)
        throw-wide(r,d)
        end-monitor-status(r,d)
```

- ~~In the book: SeRPE planner~~
  - ~~Extends the SHOP algorithm to reason about refinement methods~~
  - ~~Executes code in the method's body, but~~
    - ~~Descriptive models of commands~~
    - ~~Classical actions, abstract state~~
  - ~~To backtrack from a method *m*:~~
    - ~~Revert to state when *m* was chosen~~
- Classical actions don't represent
  - Nondeterministic outcomes, partial observability, exogenous events, durations, predicted future events, overlapping actions and events, …
  - Sometimes OK, but often not

Acting and Planning

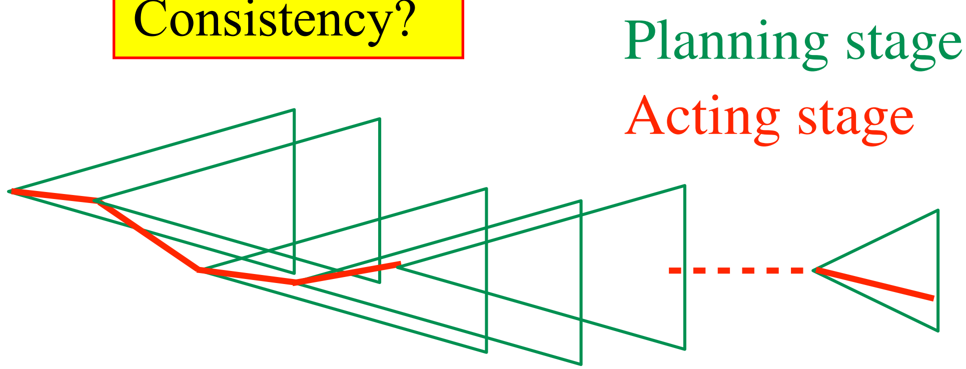
Acting

- *Performing* tasks and actions
- Use *operational models* that tell *how*
  - Dynamic, unpredictable environment
  - Adapt to context, react to events

Planning

- *Prediction + search*
- Search over predicted states, possible organizations of tasks and actions
- Use *descriptive models* to predict *what*

Consistency?



Acting and Planning

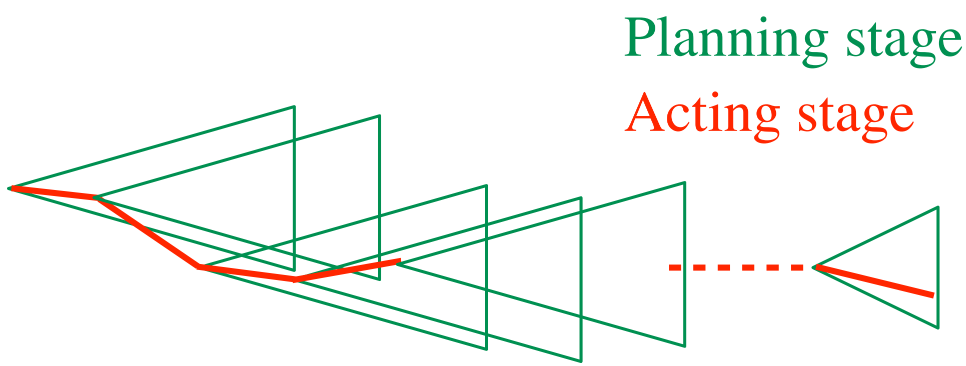
Acting

- *Performing* tasks and actions
- Use *operational models* that tell *how*
  - Dynamic, unpredictable environment
  - Adapt to context, react to events

Planning

- *Prediction + search*
- Search over predicted states, possible organizations of tasks and actions
- ~~Use *descriptive models*~~ to predict *what*

Simulated execution of actor's operational models



Why should we think this will work?

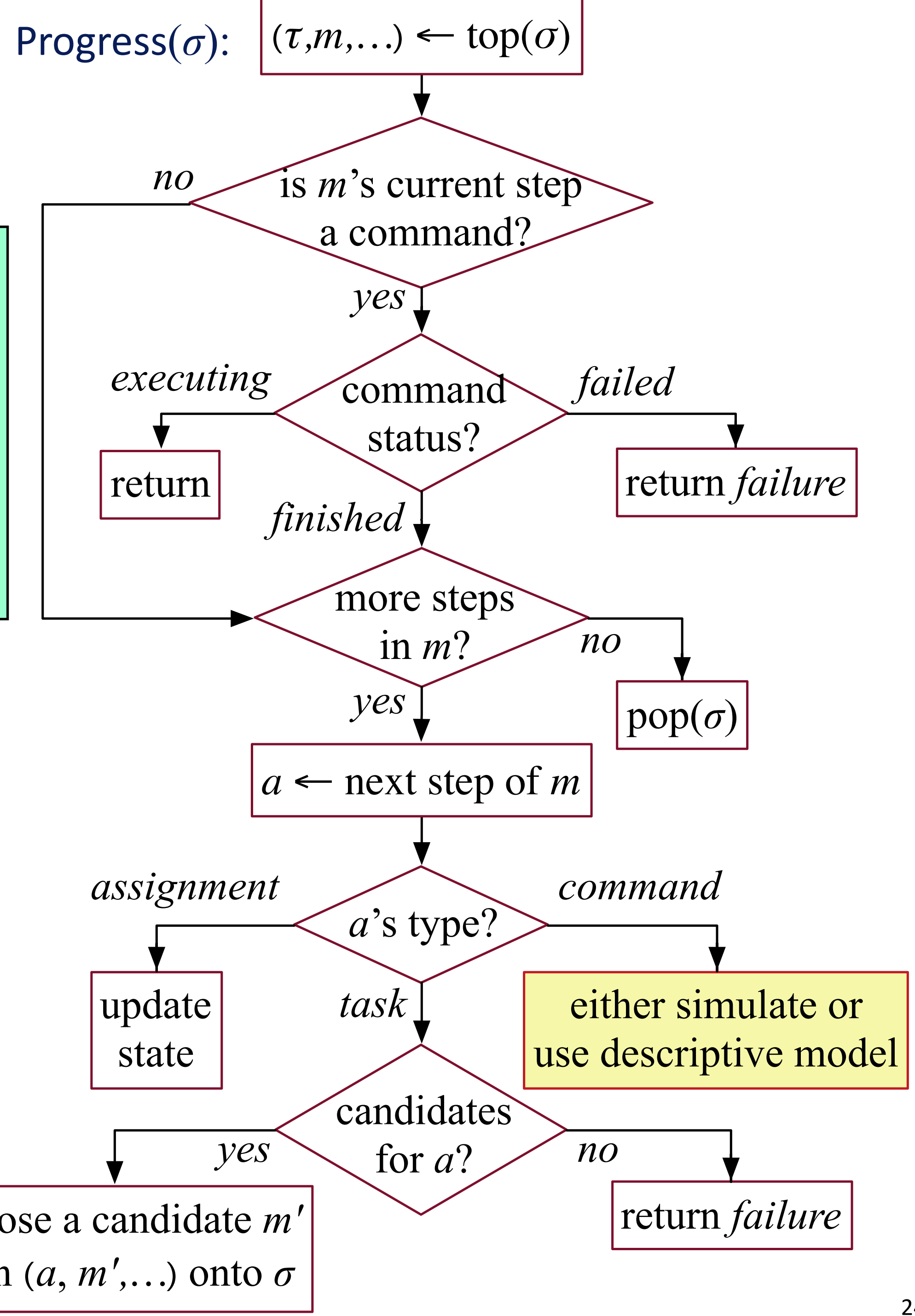
- Why does AI planning use descriptive models?
  (1) AI planners search a huge search space, need fast predictions
  (2) Effort required to create detailed operational models
    - Especially if you aren't a domain expert

- Problems aren't as bad as they used to be
  (1) Like HTN methods, the operational models focus the search
    Computers have gotten more powerful
    - Compute detailed simulations more quickly
      - e.g., fast physics-based simulations
  (2) Real-world actors will already include control software
    - May be able to use it as refinement methods

RAEplan

- Simulate RAE on a single task

RAEplan(task τ):
  Candidates = {applicable method instances}
  nondeterministically choose $m \in$ *Candidates*
  create refinement stack σ for τ and m
  loop while Progress(σ) $\neq$ *failure*
    if σ is completed then return m
  return *failure*



RAEplan

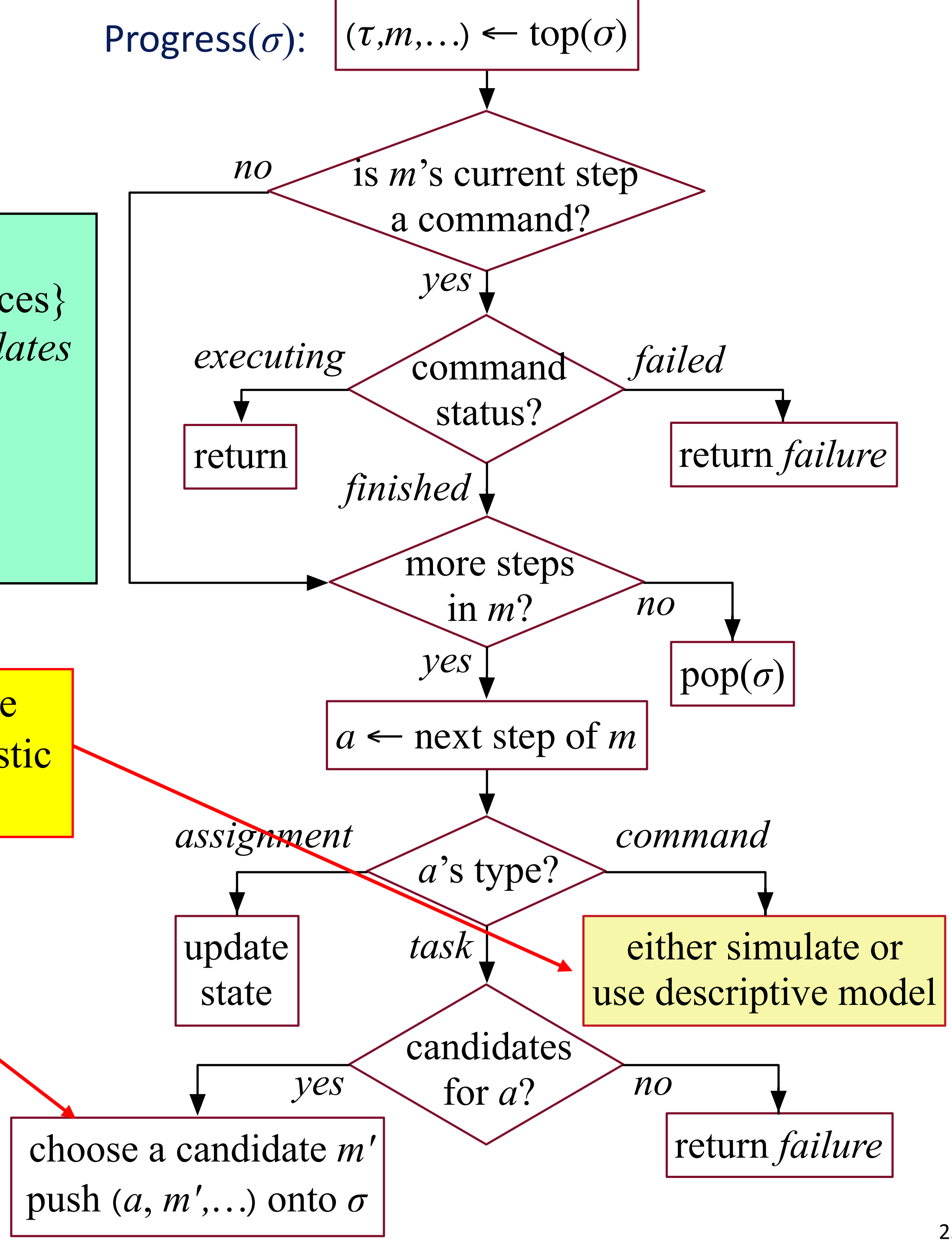
- Simulate RAE on a single task

RAEplan(task τ):
    Candidates = {applicable method instances}
    nondeterministically choose $m \in$ *Candidates*
    create refinement stack σ for τ and m
    loop while Progress(σ) $\neq$ *failure*
        if σ is completed then return m
    return *failure*

What control strategy?

How to handle nondeterministic outcomes?

Progress(σ):



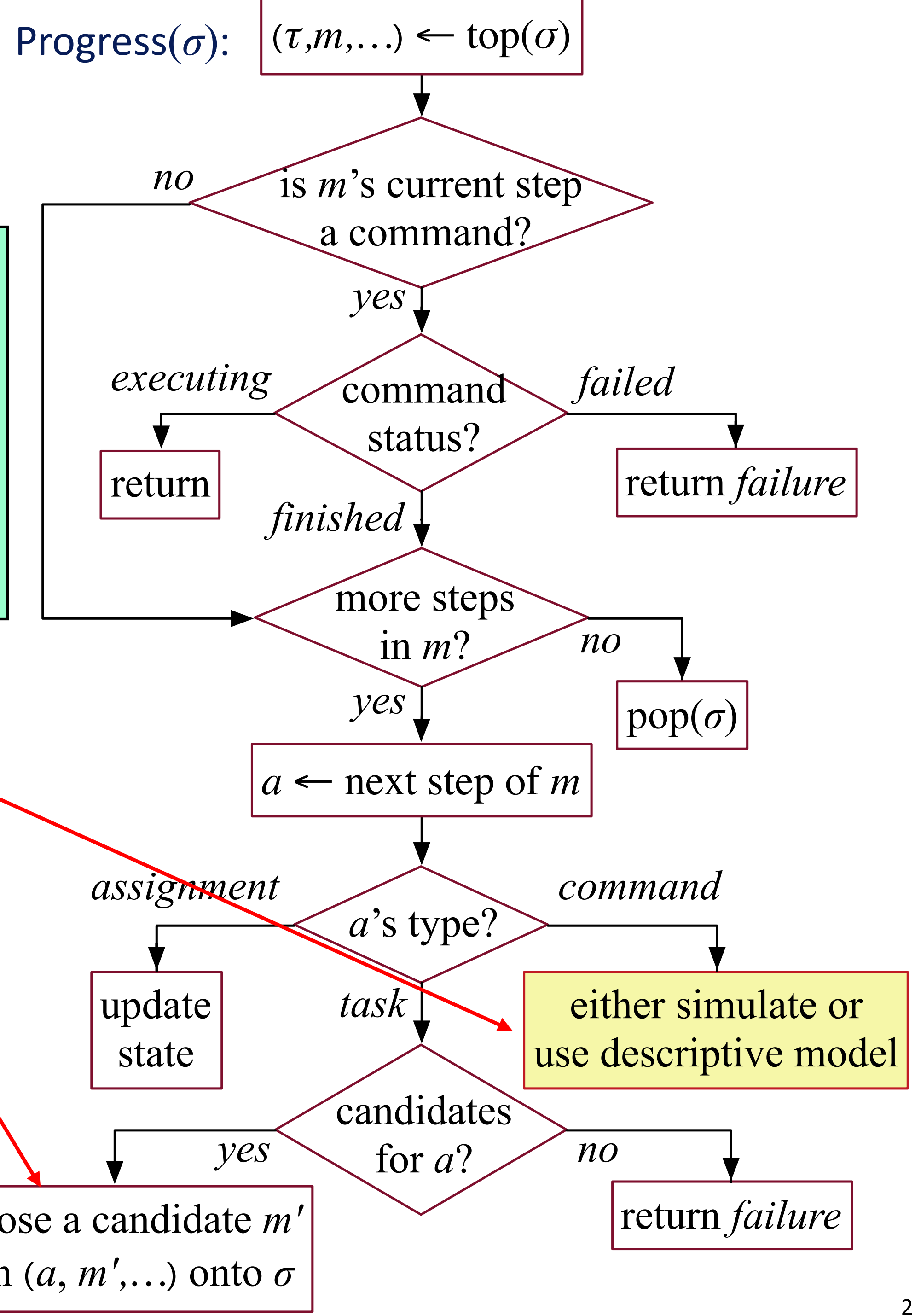
RAEplan

- Simulate RAE on a single task

```
RAEplan(task τ):
    Candidates = {applicable method instances}
    nondeterministically choose m ∈ Candidates
    create refinement stack σ for τ and m
    loop while Progress(σ) ≠ failure
        if σ is completed then return m
    return failure
```

Idea 1:
backtracking

- Backtrack over method's body
- Was easy in SHOP
  - backtrack over the task list
- In RAEplan, need to backtrack over code
- A group of students worked on this for several months, didn't get it to work



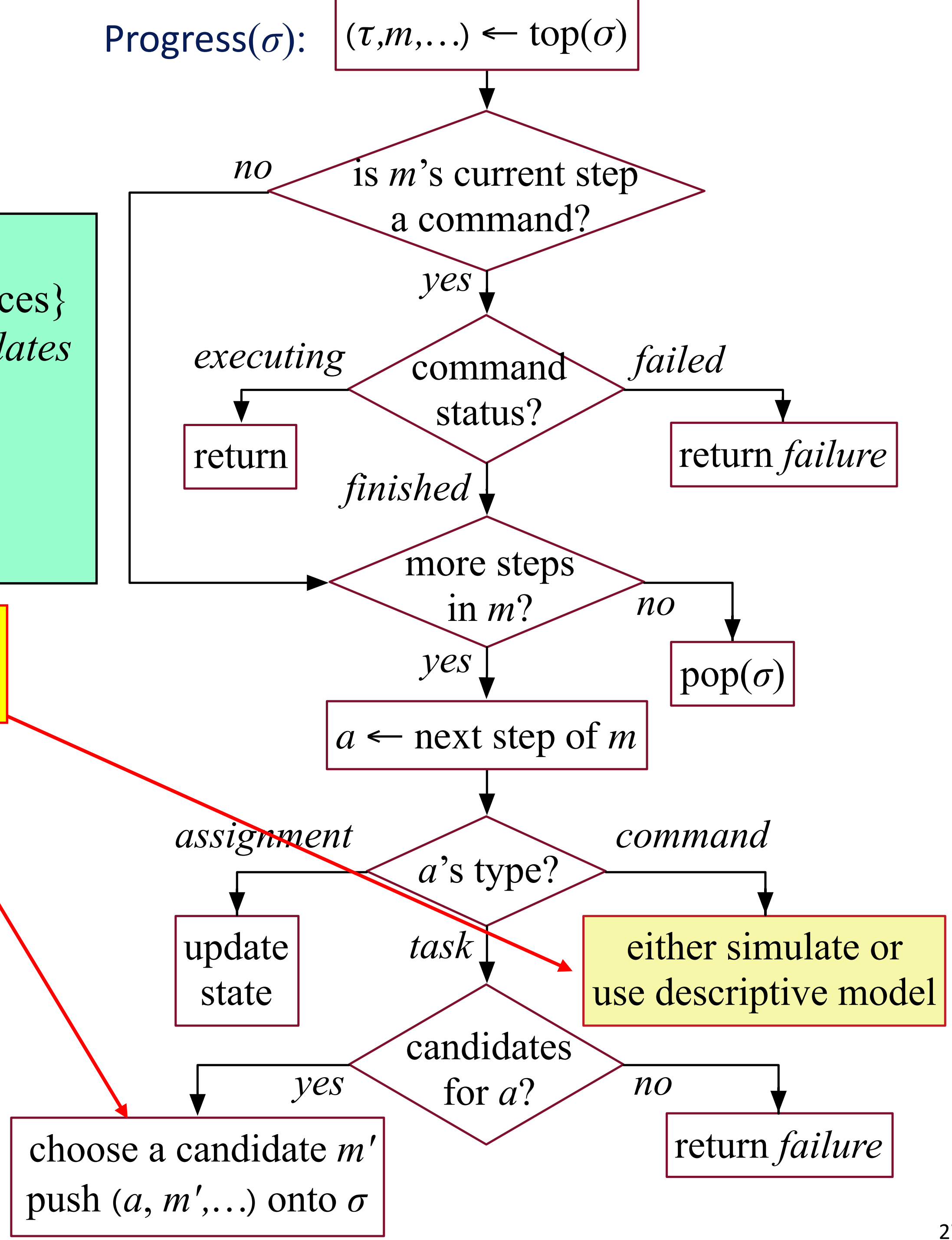
RAEplan

- Simulate RAE on a single task

RAEplan(task τ):
  Candidates = {applicable method instances}
  nondeterministically choose $m \in$ *Candidates*
  create refinement stack σ for τ and m
  loop while Progress(σ) $\neq$ *failure*
    if σ is completed then return m
  return *failure*

Idea 2:
multithreading

- Too many parallel processes

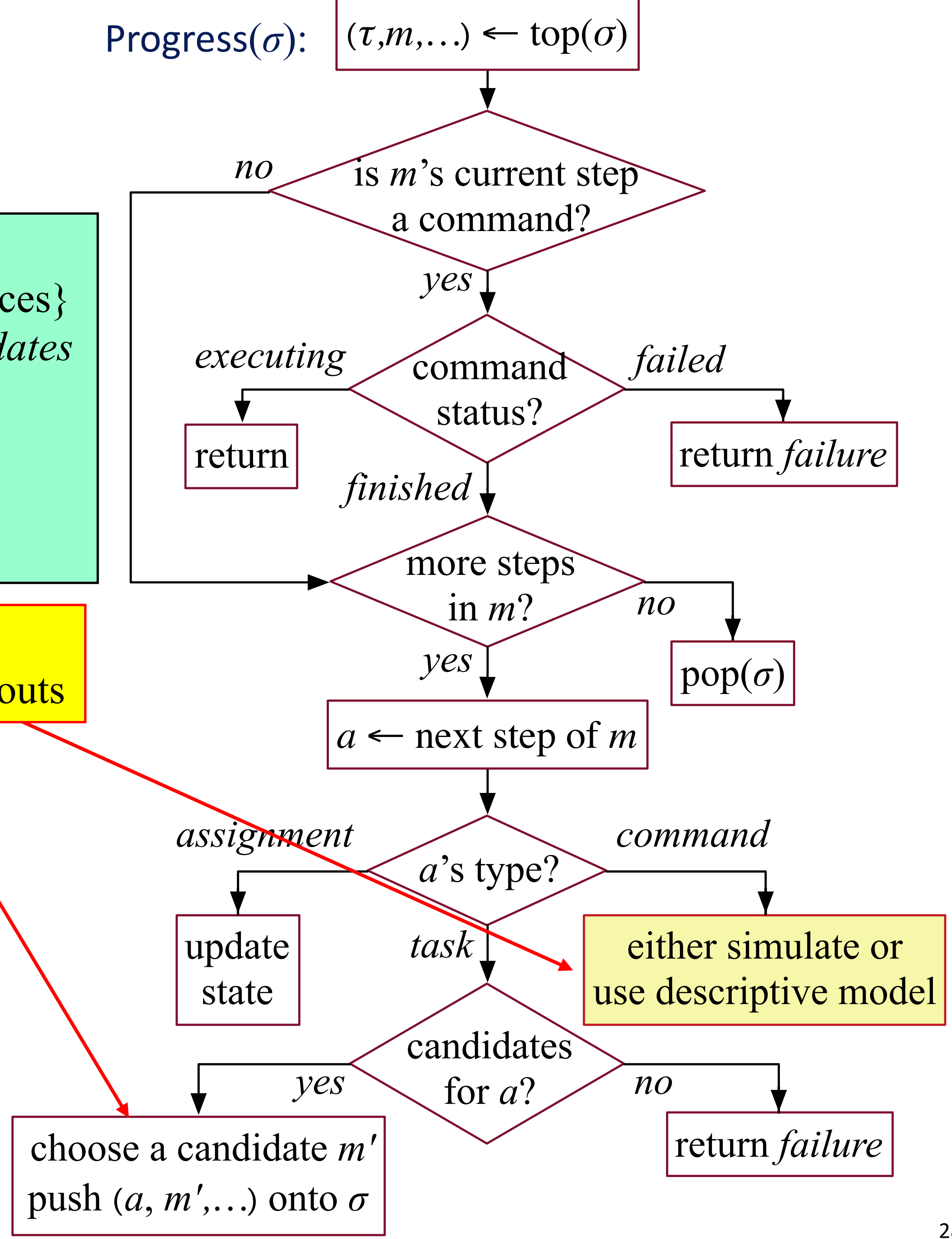


RAEplan

- Simulate RAE on a single task

RAEplan(task τ):
  Candidates = {applicable method instances}
  nondeterministically choose $m \in$ *Candidates*
  create refinement stack σ for τ and m
  loop while Progress(σ) $\neq$ *failure*
    if σ is completed then return m
  return *failure*

Idea 3:
Monte Carlo rollouts



- Multiple runs
  - Random outcomes in each run
- Return the method m that gives the highest expected utility

Patra, Traverso, Ghallab, and Nau. Acting and planning using operational models. *AAAI*, 2019

Summary of Experimental Results

Domain	Dynamic events	Dead ends	Sensing	Robot collaboration	Concurrent tasks
CR	✓	✓	✓	–	✓
EE	✓	✓	–	✓	✓
SD	✓	–	–	✓	✓
IP	✓	–	–	✓	✓

- Four different domains, different combinations of characteristics
- Evaluation criteria:
  - Efficiency, successes vs failures, how many retries
- Result: planning helps
  - RAE operated better with RAEplan than without
  - RAE operated better with more planning than with less planning

Patra, Traverso, Ghallab, and Nau. Acting and planning using operational models. *AAAI*, 2019

Summary

- Continual online planning, in service of acting
- Descriptive vs. operational models
- Refinement methods
  - Generalization of HTN methods
    - State variables
    - Body is a computer program that includes tasks, commands
    - Commands interact with environment
    - Outcomes not known in advance
  - Used in two ways
    - Reactively in RAE
    - Predictively in RAEplan
- Experimental results with RAE, RAEplan
  - More planning → better acting

Limitations and Future Work

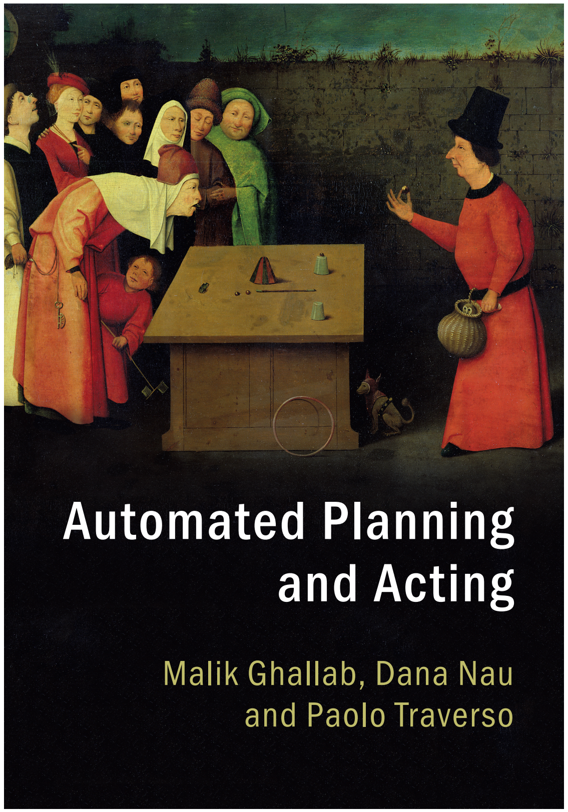
- How to get the operational models?
  - Earlier, I said, “Real-world actors may already include operational models”
    - OK if the actor is RAE, OpenPRS, …
    - Otherwise, might not be in a form we can use
  - Currently, only alternative is to write them ourselves
    - Develop learning algorithms to do this?

- Experiments were done in “toy domains”
  - Want to test the approach in real planning problems

- Ongoing project with US Naval Research Laboratory
  - Use RAE, RAEplan for recovery from attacks on software-defined networks
  - They’re writing the refinement methods
  - We plan to modify RAE, RAEplan to meet their needs

Links

- Ghallab, Nau, and Traverso, *Automated Planning and Acting,* Cambridge University Press, 2016
  - Final manuscript, lecture slides



- Patra, Traverso, Ghallab, and Nau. Acting and planning using operational models. *AAAI*, 2019

- RAE and RAEplan source code
  - 3-clause BSD license
  - Caveat: software still under development