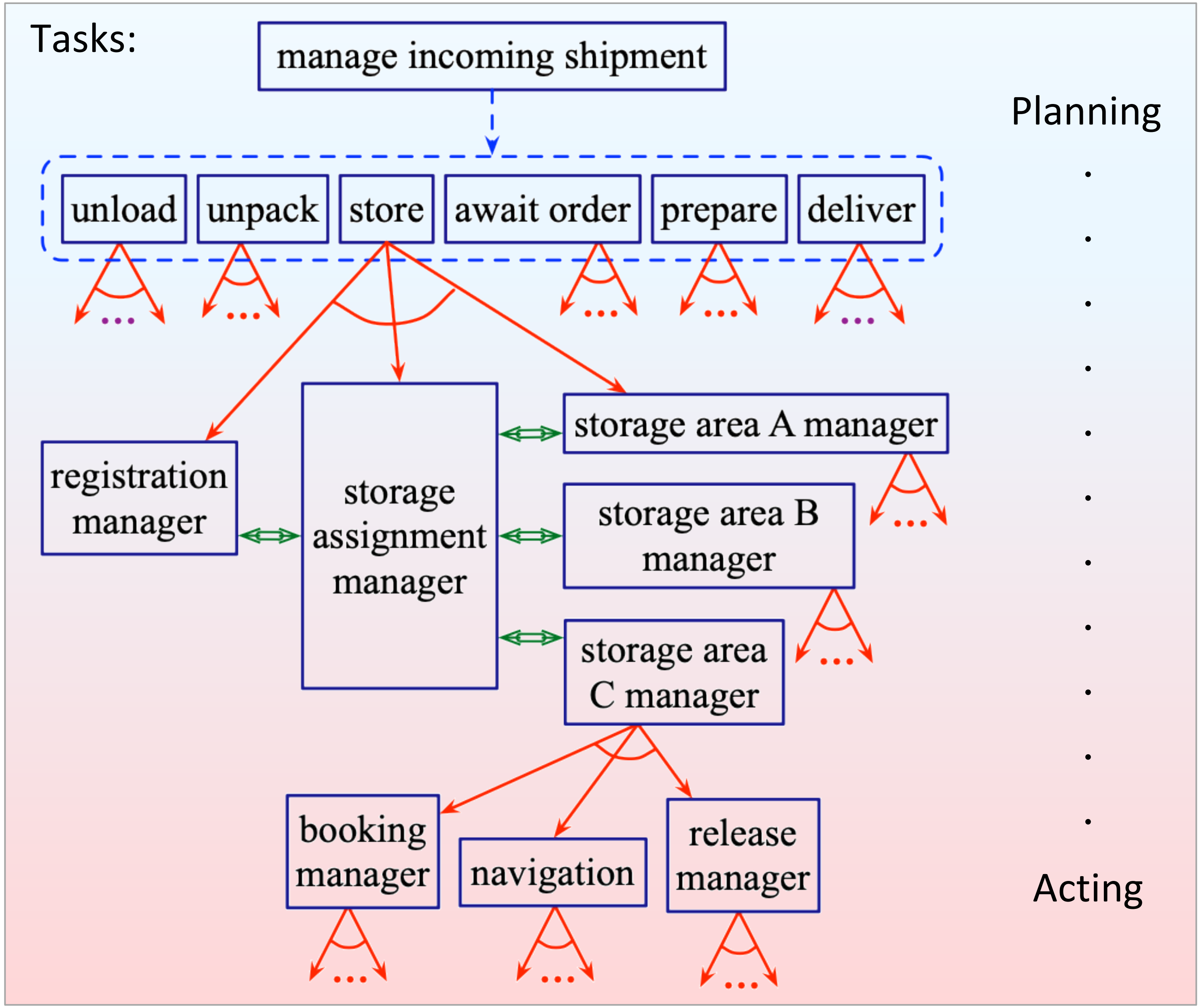
Integrated Planning and Acting Using Operational Models

Dana S. Nau and Sunandita Patra
University of Maryland

Motivation

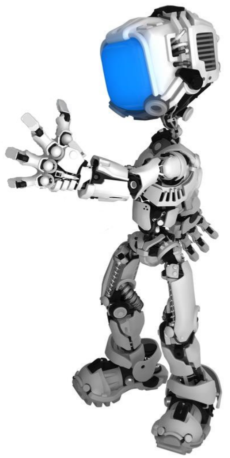
Harbor Management

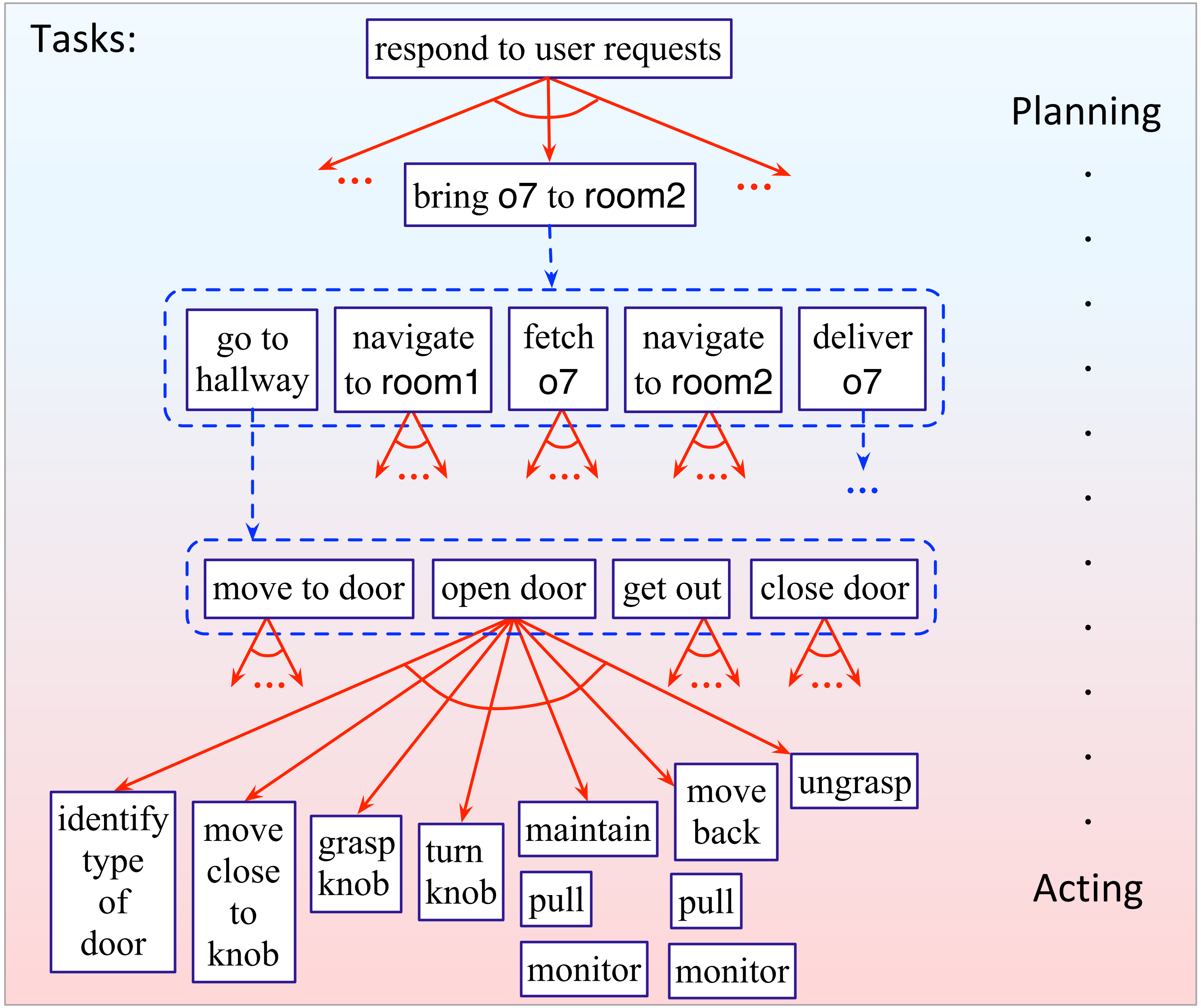
- Multiple levels of abstraction
  - Physical/managerial organization of harbor
- Higher levels:
  - Plan abstract tasks
- Lower levels:
  - Multiple agents, partial observability dynamic change
- Continual online planning
  - Plans are abstract and partial until more detail needed



Hypothetical Worker Robot

- Multiple levels of abstraction
- At higher levels:
  - Plan abstract tasks
- At lower levels:
  - Nondeterminism, partial observability dynamic change
- Continual online planning
  - Plans are abstract and partial until more detail needed

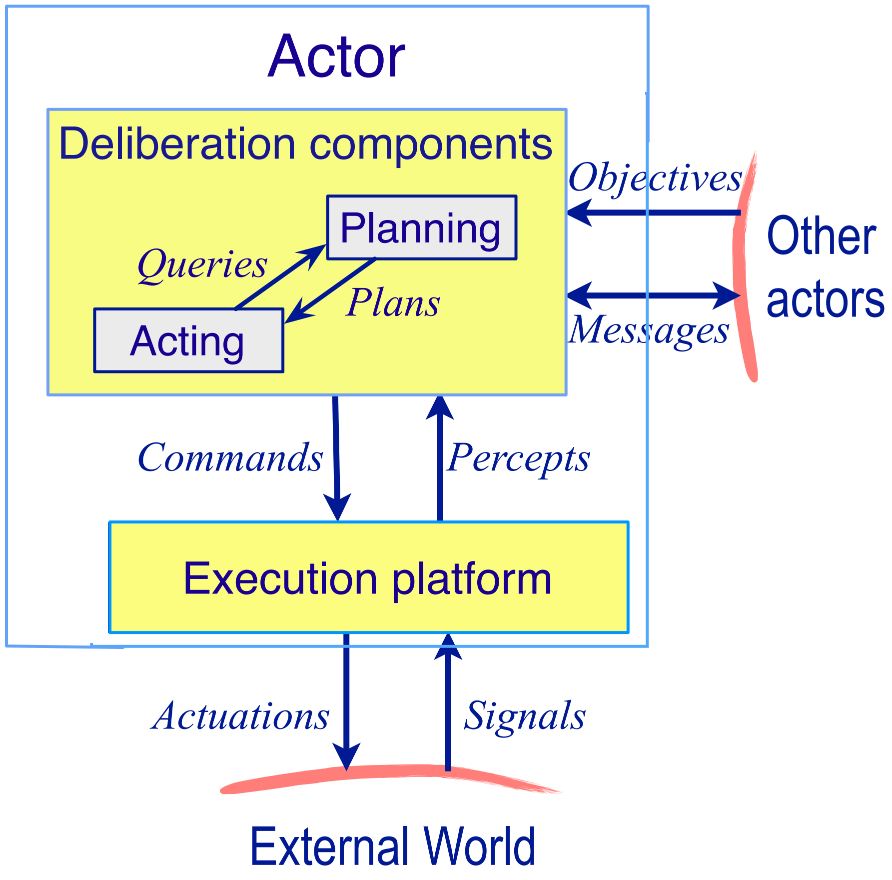




Planning and Acting

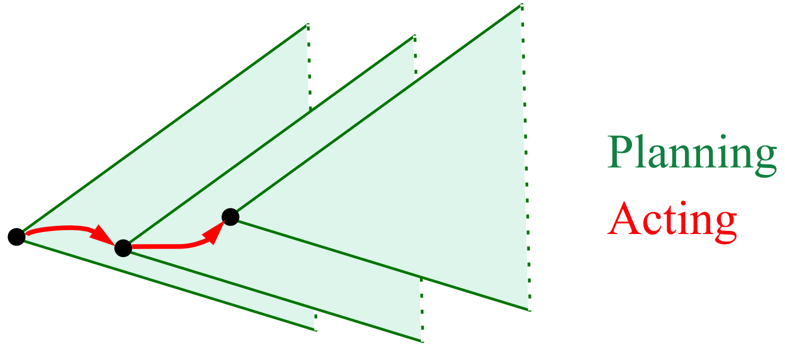
Planning

- *Prediction + search*
  - Search over predicted states, possible organizations of tasks and actions
- Uses *descriptive* models (e.g., PDDL)
  - predict *what* the actions will do
  - don't include instructions for performing it

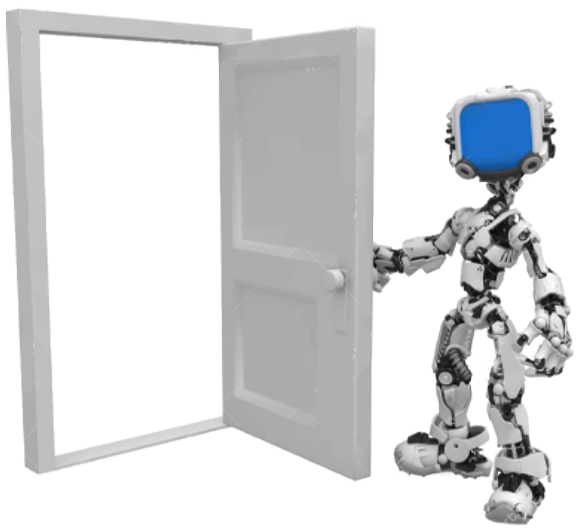


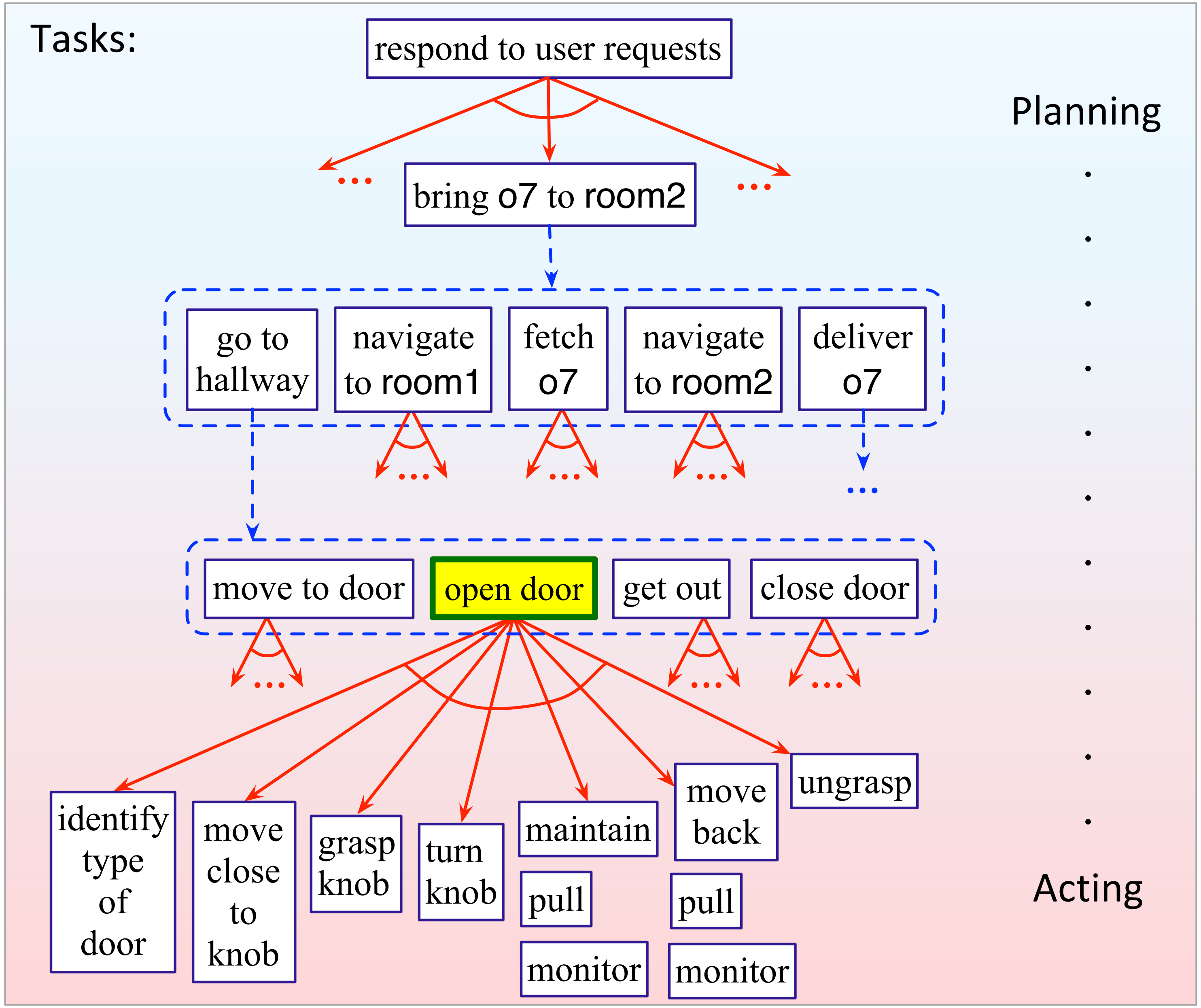
Acting

- *Performing* actions
  - Dynamic, unpredictable, partially observable environment
  - Adapt to context, react to events
- Uses *operational* models
  - instructions telling *how* to perform the actions



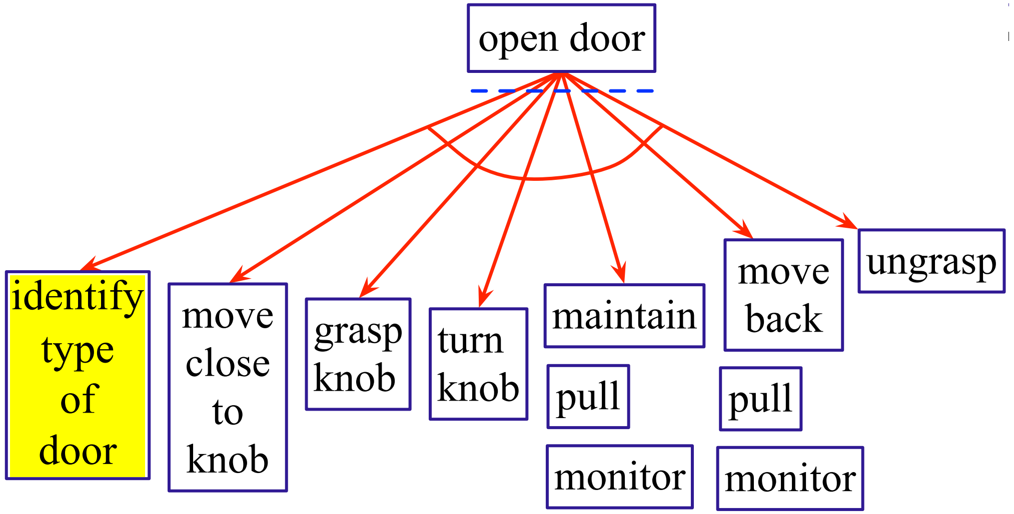
Opening a Door

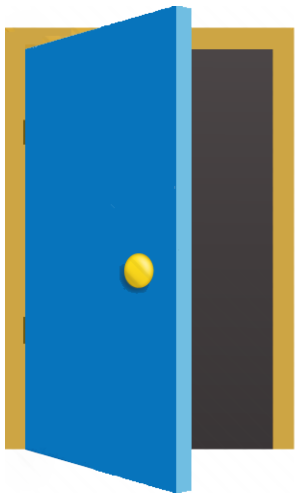




Opening a Door

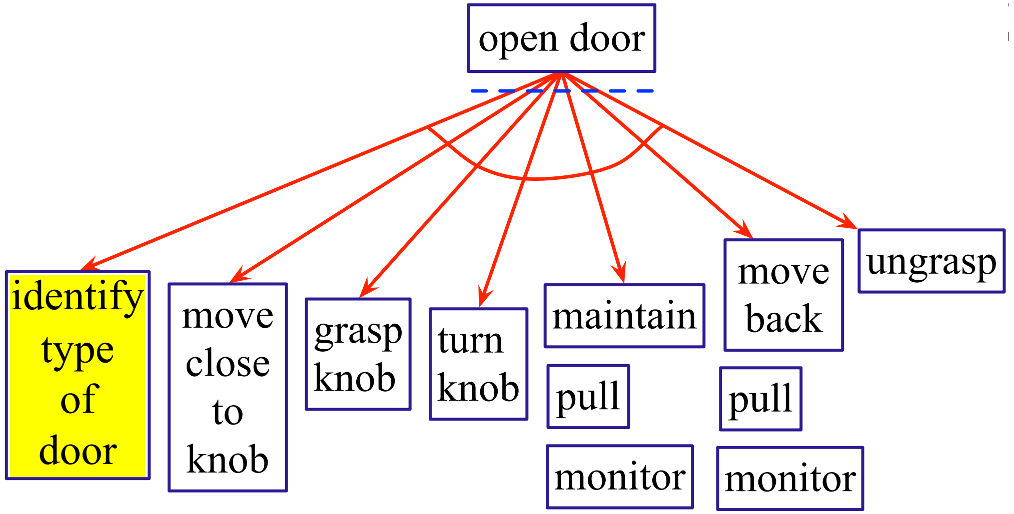
- Different methods, depending on what kind of door
  - Sliding or hinged?

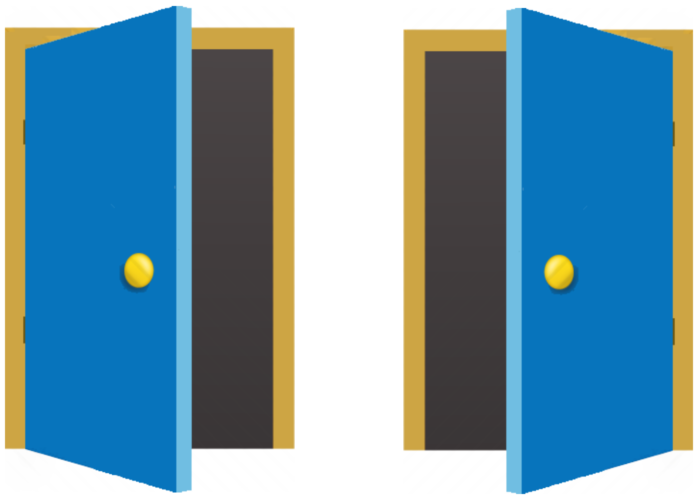




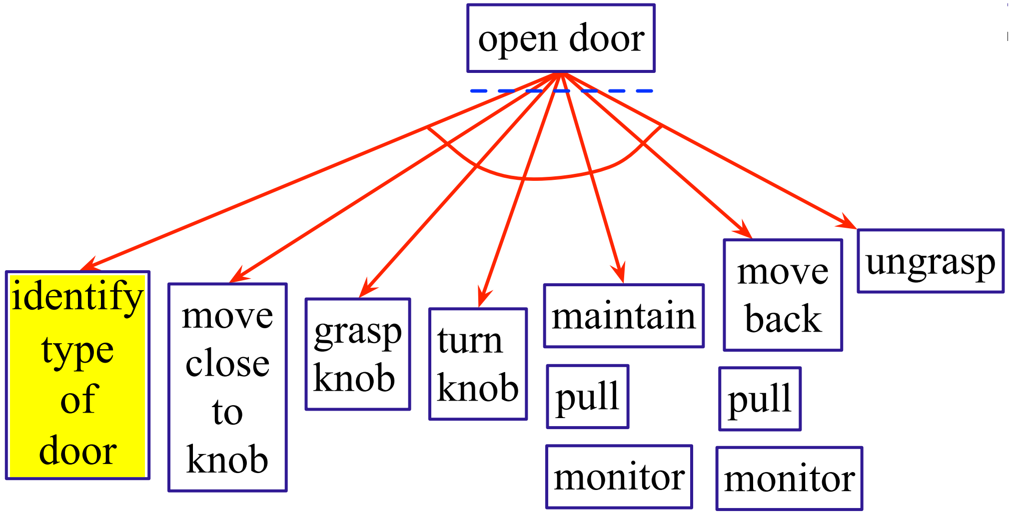
Opening a Door

- Different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?

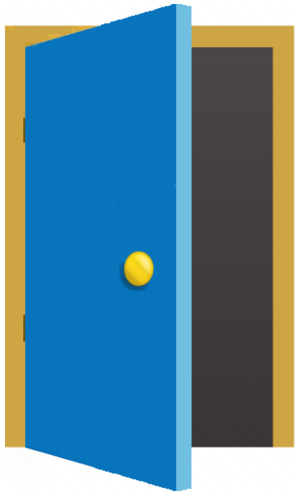


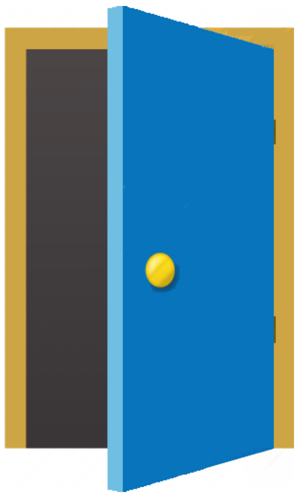


Opening a Door



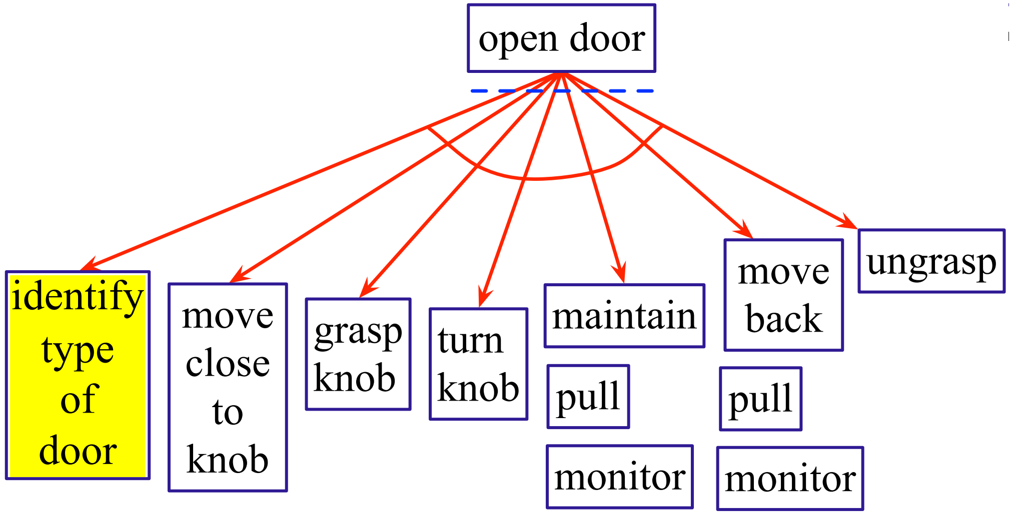
- Different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?

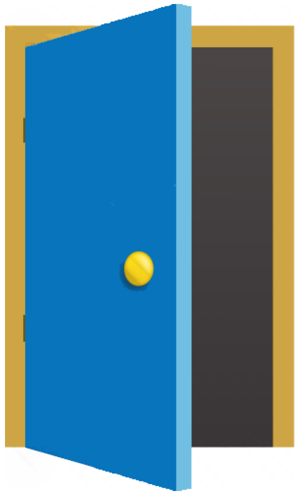


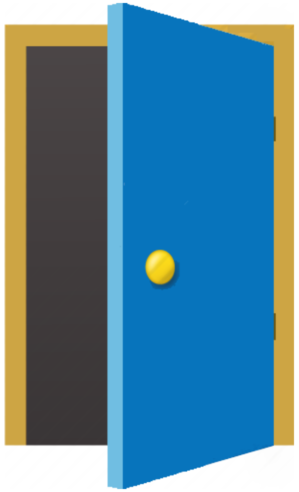


Opening a Door

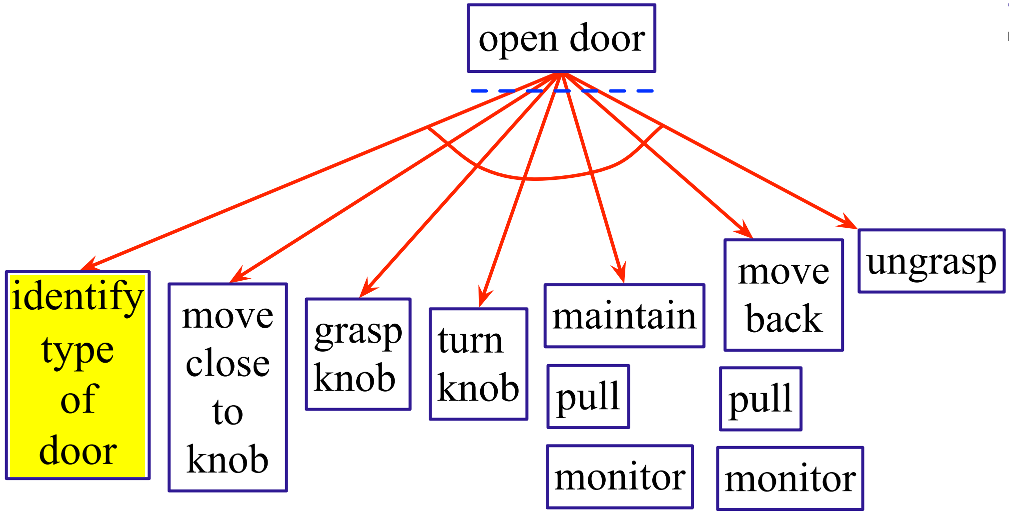
- Different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?
  - Knob, lever, push bar, …



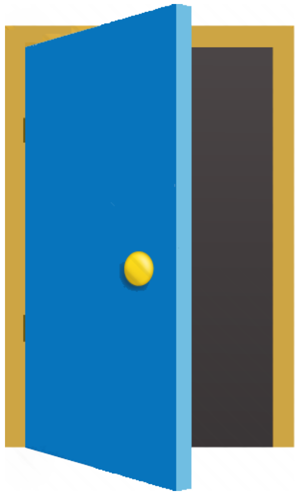


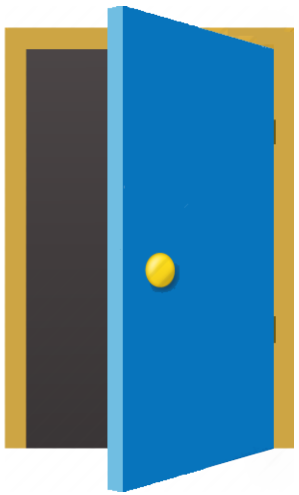


Opening a Door



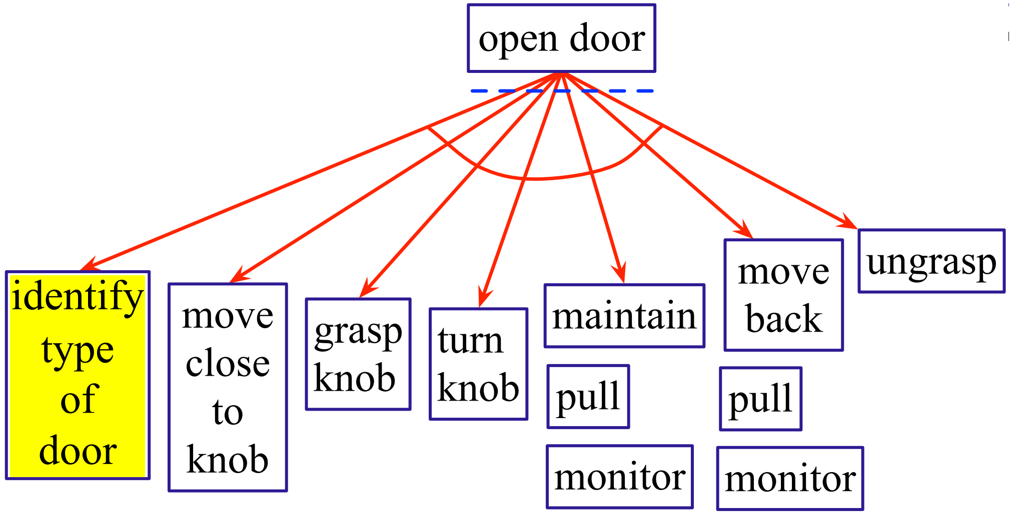
- Different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?
  - Knob, lever, push bar, pull handle, push plate, …

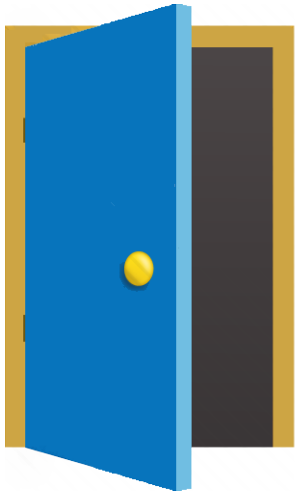


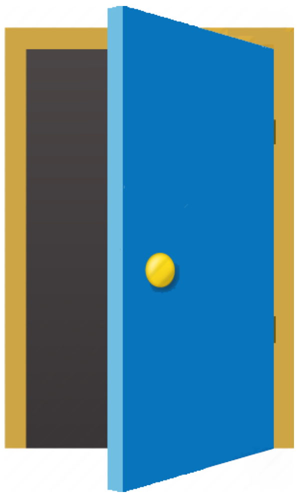


Opening a Door

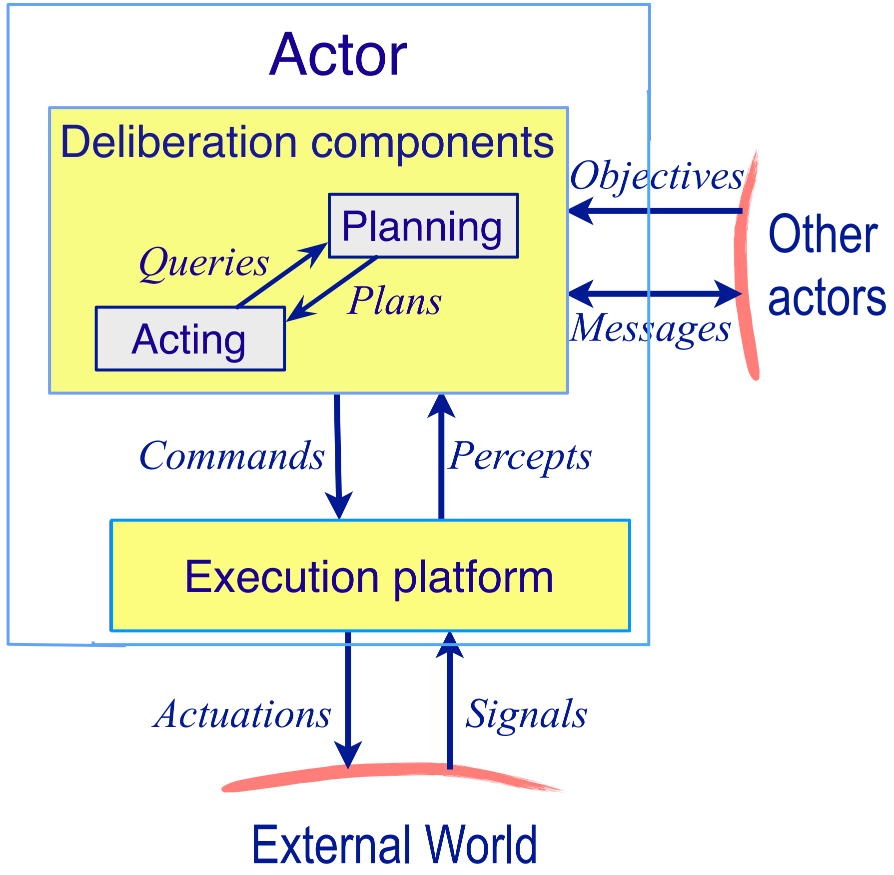
- Different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?
  - Knob, lever, push bar, pull handle, push plate, something else?







RAE and UPOM



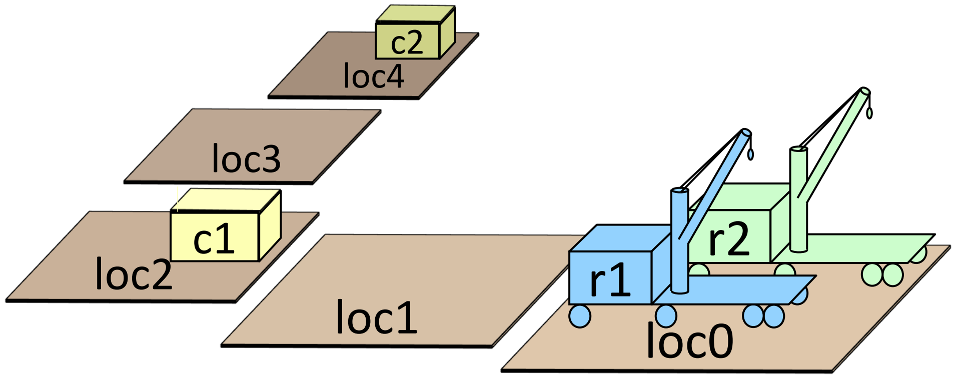
- Python implementation:
  - https://github.com/sunandita/ICAPS_Summer_School_RAE_2020
  - Full code: https://bitbucket.org/sunandita/rae/

- Related publications
  - Patra, Mason, Kumar, Ghallab, Traverso, and Nau (2020). Integrating Acting, Planning, and Learning in Hierarchical Operational Models.
    ICAPS-2020. Best student paper honorable mention award.
    https://www.aaai.org/ojs/index.php/ICAPS/article/view/6743/6597
  - Patra, Mason, Ghallab, Dana, and Traverso (2020). Deliberative Acting, Online Planning and Learning with Hierarchical Operational Models.
    Submitted for journal publication.
    Preprint at https://arxiv.org/abs/2010.01909
  - Ghallab, Nau, and Traverso (2016).
    Automated Planning and Acting.
    Cambridge University Press. Authors' final manuscript at
    http://projects.laas.fr/planning/

Outline

1. **Motivation**
2. **Representation** – state variables, commands, tasks, refinement methods
3. **Acting** – Rae (Refinement Acting Engine)
4. **Planning** – UPOM (UCT-like Planner for Operational Models)
5. **Acting with Planning** – Rae + UPOM
6. **Using the implementation** – Rae code, UPOM code, examples

Representation



- Objects
  - *Robots* = {r1, r2}
  - *Containers* = {c1, c2}
  - *Locations* = {loc1, loc2, loc3, loc4}

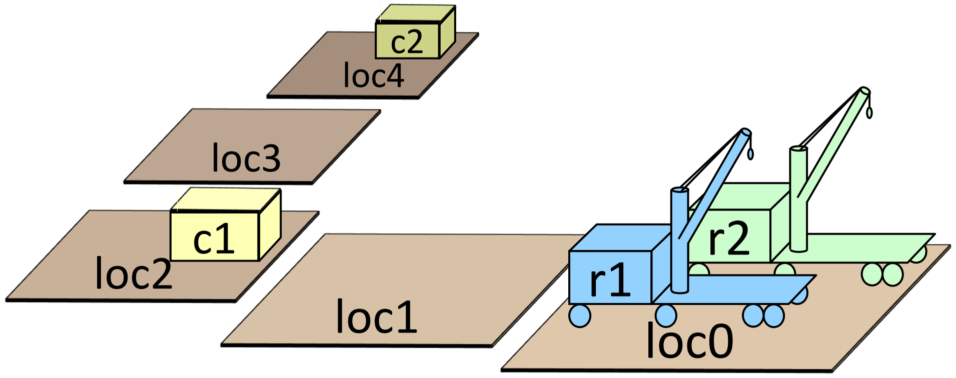
- Rigid relations (properties that won't change)
  - adjacent(loc0,loc1), adjacent(loc1,loc0), adjacent(loc1,loc2), adjacent(loc2,loc1), adjacent(loc2,loc3), adjacent(loc3,loc2), adjacent(loc3,loc4), adjacent(loc4,loc3)

- State variables (fluents)
  - where $r \in Robots$, $c \in Containers$, $l \in Locations$
  - loc(r) ∈ *Locations*
  - cargo(r) ∈ *Containers* ∪ {empty}
  - pos(c) ∈ *Locations* ∪ *Robots* ∪ {unknown}
  - view(l) ∈ {T, F}
    - Whether a robot has looked at location l
    - If view(l) = T then pos(c) = l for every container c at l

- Commands to the execution platform:
  - take(r,o,l): r takes object o at location l
  - put(r,o,l): r puts o at location l
  - perceive(r,l): robot r perceives what objects are at l
  - move-to(r,l): robot r moves to location l

Tasks and Methods

- *Task*: an activity for the actor to perform
  - taskname($arg_1, \ldots, arg_k$)
- For each task, one or more *refinement methods*
  - Operational models telling how to perform the task




```
method-name(arg1, …, argk)
    task:  task-identifier
    pre:   test
    body:
           a program
```

```
m-fetch1(r,c)
    task:  fetch(r,c)
    pre:   pos(c) = unknown
    body:
       if ∃l (view(l) = F) then
           move-to(r,l)
           perceive(r,l)
           if pos(c) = l then
command ——→   take(r,c,l)
           else fetch(r,c)  ←—— task
       else fail
```

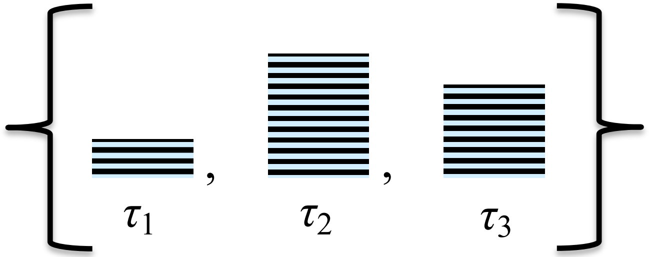
```
m-fetch2(r,c)
    task:  fetch(r,c)
    pre:   pos(c) ≠ unknown
    body:
       if loc(r) = pos(c) then
           take(r,c,pos(c))
       else do
           move-to(r,pos(c))
           take(r,c,pos(c))
```

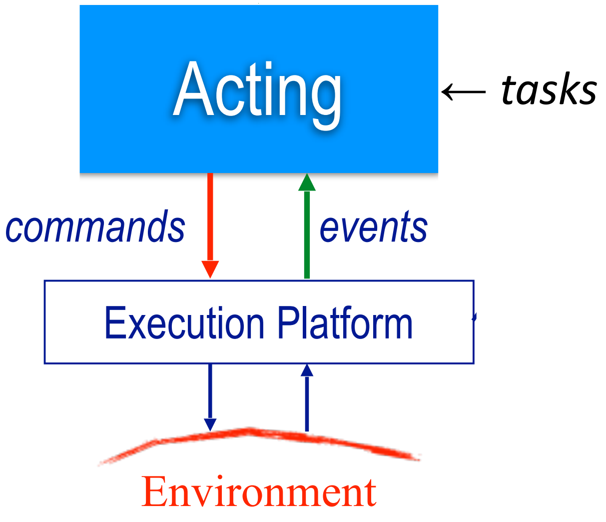
Outline

1. **Motivation**
2. **Representation** – state variables, commands, tasks, refinement methods
3. **Acting** – Rae (Refinement Acting Engine)
4. **Planning** – UPOM (UCT-like Planner for Operational Models)
5. **Acting with Planning** – Rae + UPOM
6. **Using the implementation** – Rae code, UPOM code, examples

Rae (Refinement Acting Engine)

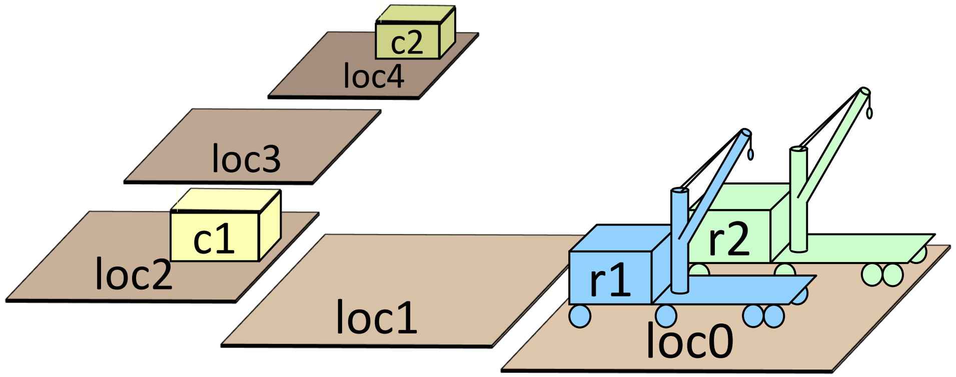
- Performs multiple tasks in parallel
  - Purely reactive, no lookahead
- For each task or event τ, a *refinement stack*
  - execution stack
  - corresponds to current path in Rae's search tree for τ
- *Agenda* = {all current refinement stacks}





procedure Rae:
    loop:
        for every new external task or event τ do
            choose a method instance m for τ
            create a refinement stack for τ, m
            add the stack to *Agenda*
        for each stack σ in *Agenda*
            Progress(σ)
            if σ is finished then remove it

Representation



- Objects
  - *Robots* = {r1, r2}
  - *Containers* = {c1, c2}
  - *Locations* = {loc1, loc2, loc3, loc4}
- Rigid relations (properties that won't change)
  - adjacent(loc0,loc1), adjacent(loc1,loc0), adjacent(loc1,loc2), adjacent(loc2,loc1), adjacent(loc2,loc3), adjacent(loc3,loc2), adjacent(loc3,loc4), adjacent(loc4,loc3)

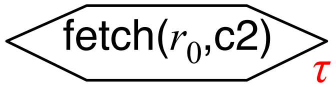
- State variables (fluents)
  - where $r \in Robots$, $c \in Containers$, $l \in Locations$
  - loc(r) $\in$ *Locations*
  - cargo(r) $\in$ *Containers* $\cup$ {nil}
  - pos(c) $\in$ *Locations* $\cup$ *Robots* $\cup$ {unknown}
  - view(l) $\in$ {T, F}
    - Whether a robot has looked at location l
    - If view(l) = T then pos(c) = l for every container c at l
- Commands to the execution platform:
  - take(r,o,l): r takes object o at location l
  - put(r,o,l): r puts o at location l
  - perceive(r,l): robot r perceives what objects are at l
  - move-to(r,l): robot r moves to location l

Example

m-fetch1(r,c)
    task: fetch(r,c)
    pre: pos(c) = unknown
    body:
        if $\exists l$ (view(l) = F) then
            move-to(r,l)
            perceive(r,l)
            if pos(c) = l then
                take(r,c,l)
            else fetch(r,c)
        else fail

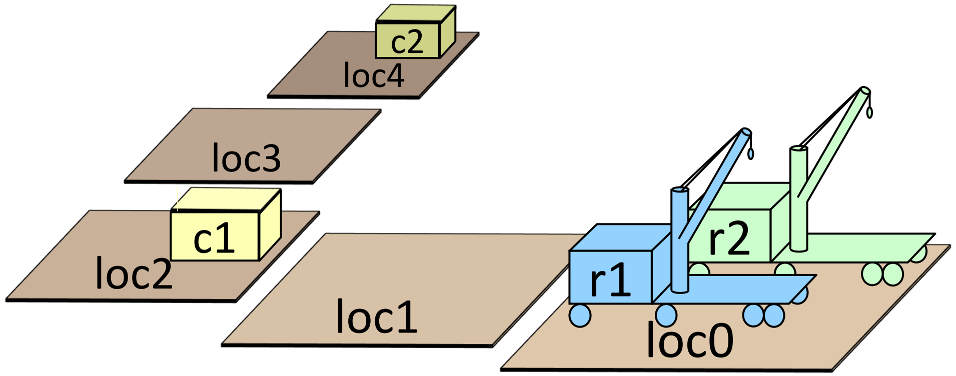
m-fetch2(r,c)
    task: fetch(r,c)
    pre: pos(c) $\neq$ unknown
    body:
        if loc(r) = pos(c) then
            take(r,c,pos(c))
        else do
            move-to(r,pos(c))
            take(r,c,pos(c))

Search tree



procedure Rae:
    loop:
        for $\boxed{\text{every new external task or event } \tau}$ do
            choose a method instance m for τ
            create a refinement stack for τ, m
            add the stack to *Agenda*
        for each stack σ in *Agenda*
          Progress(σ)
          if σ is finished then remove it

- Container locations unknown
- Partially observable
  - Robot only sees current location



Example

m-fetch1(r,c) $r = r_0$, $c =$ c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body:
  - if $\exists l$ (view(l) = F) then
    - move-to(r,l)
    - perceive(r,l)
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail

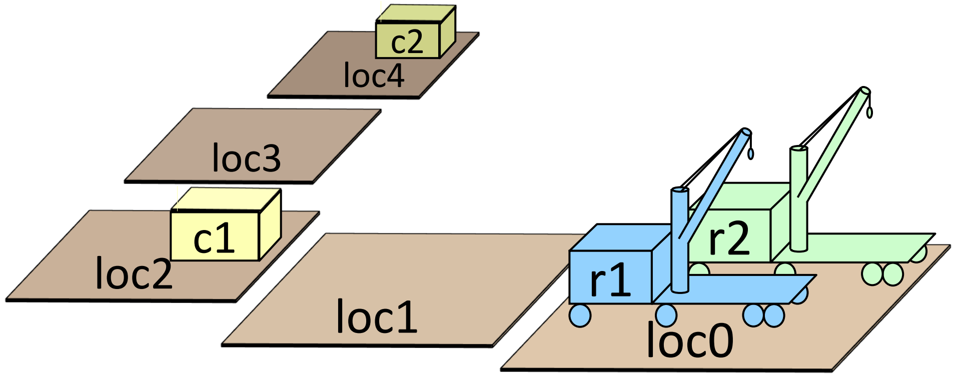
m-fetch2(r,c)
- task: fetch(r,c)
- pre: pos(c) $\neq$ unknown
- body:
  - if loc(r) = pos(c) then
    - take(r,c,pos(c))
  - else do
    - move-to(r,pos(c))
    - take(r,c,pos(c))

Search tree

fetch(r_0,c2) τ

Candidates
= {m-fetch(r1,c2),
m-fetch(r2,c2)}

- Container locations unknown
- Partially observable
  - Robot only sees current location



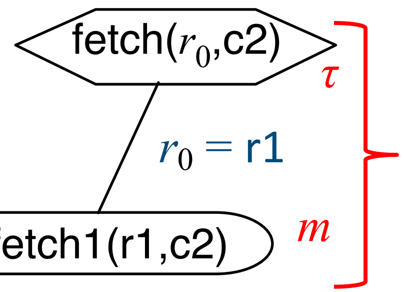
procedure Rae:
- loop:
  - for every new external task or event τ do
    - choose a method instance m for τ
    - create a refinement stack for τ, m
    - add the stack to *Agenda*
  - for each stack σ in *Agenda*
    - Progress(σ)
    - if σ is finished then remove it

Example

```
m-fetch1(r,c)   r = r1, c = c2
    task:  fetch(r,c)
    pre:   pos(c) = unknown
    body:
       if ∃l (view(l) = F) then
          move-to(r,l)
          perceive(r,l)
          if pos(c) = l then
               take(r,c,l)
          else fetch(r,c)
       else fail
```

Candidates
= {m-fetch1(r1,c2),
m-fetch2(r2,c2)}

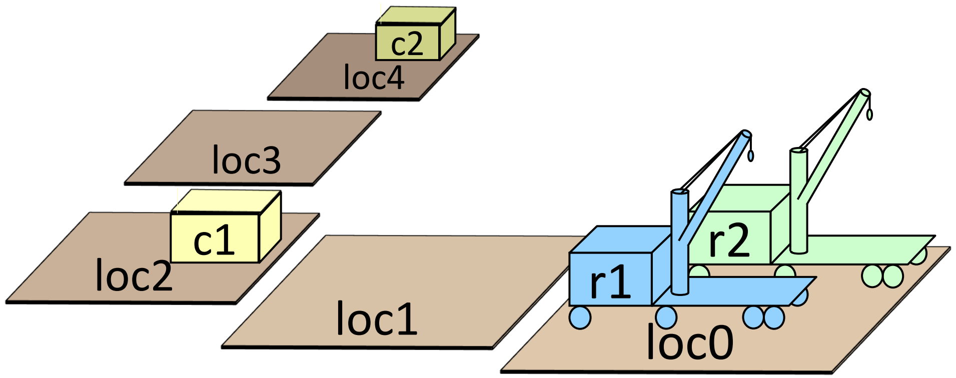
Search tree




```
procedure Rae:
   loop:
      for every new external task or event τ do
         choose a method instance m for τ
         create a refinement stack for τ, m
         add the stack to Agenda
      for each stack σ in Agenda
         Progress(σ)
         if σ is finished then remove it
```

```
m-fetch2(r,c)
    task:  fetch(r,c)
    pre:   pos(c) ≠ unknown
    body:
       if loc(r) = pos(c) then
          take(r,c,pos(c))
       else do
          move-to(r,pos(c))
          take(r,c,pos(c))
```

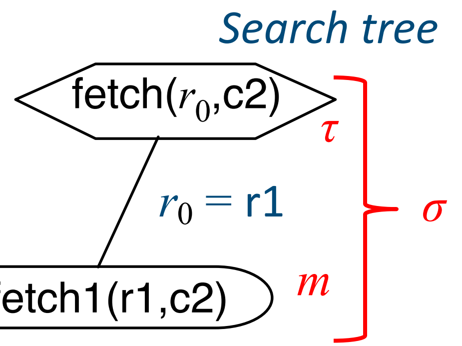
- Container locations unknown
- Partially observable
    - Robot only sees current location



Example

m-fetch1(r,c) r = r1, c = c2
  task: fetch(r,c)
  pre: pos(c) = unknown
  body:
    if $\exists l$ (view(l) = F) then
      move-to(r,l)
      perceive(r,l)
      if pos(c) = l then
        take(r,c,l)
      else search(r,c)
    else fail

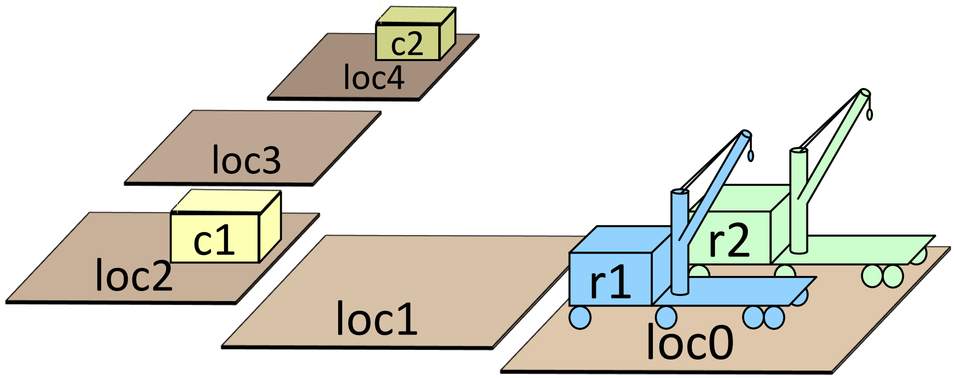
Candidates
= {m-fetch(r1,c2),
m-fetch(r2,c2)}



procedure Rae:
  loop:
    for every new external task or event τ do
      choose a method instance m for τ
      create a refinement stack for τ, m
      add the stack to *Agenda*
    for each stack σ in *Agenda*
      Progress(σ)
      if σ is finished then remove it

m-fetch2(r,c)
  task: fetch(r,c)
  pre: pos(c) ≠ unknown
  body:
    if loc(r) = pos(c) then
      take(r,c,pos(c))
    else do
      move-to(r,pos(c))
      take(r,c,pos(c))

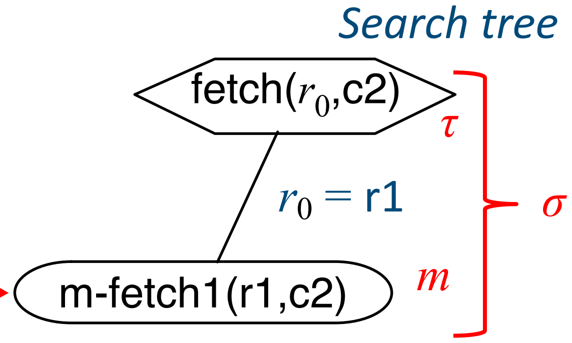
- Container locations unknown
- Partially observable
  - Robot only sees current location



Example

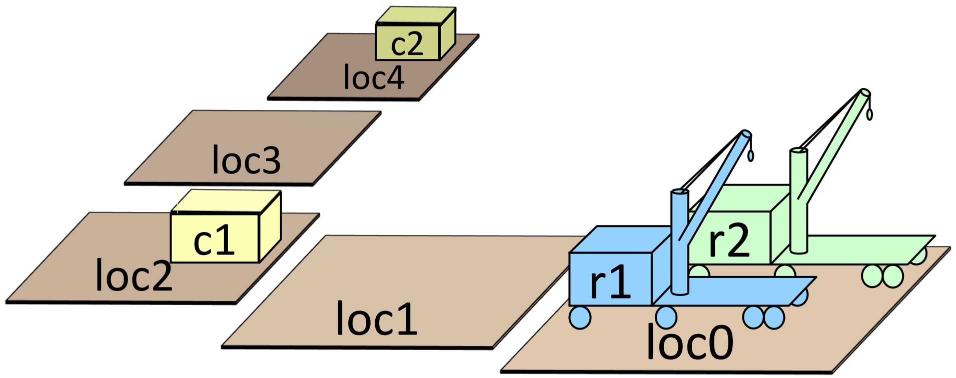
m-fetch1(r,c) r = r1, c = c2
task: fetch(r,c)
pre: pos(c) = unknown
body:
if ∃l (view(l) = F) then
move-to(r,l)
perceive(r,l)
if pos(c) = l then
take(r,c,l)
else fetch(r,c)
else fail

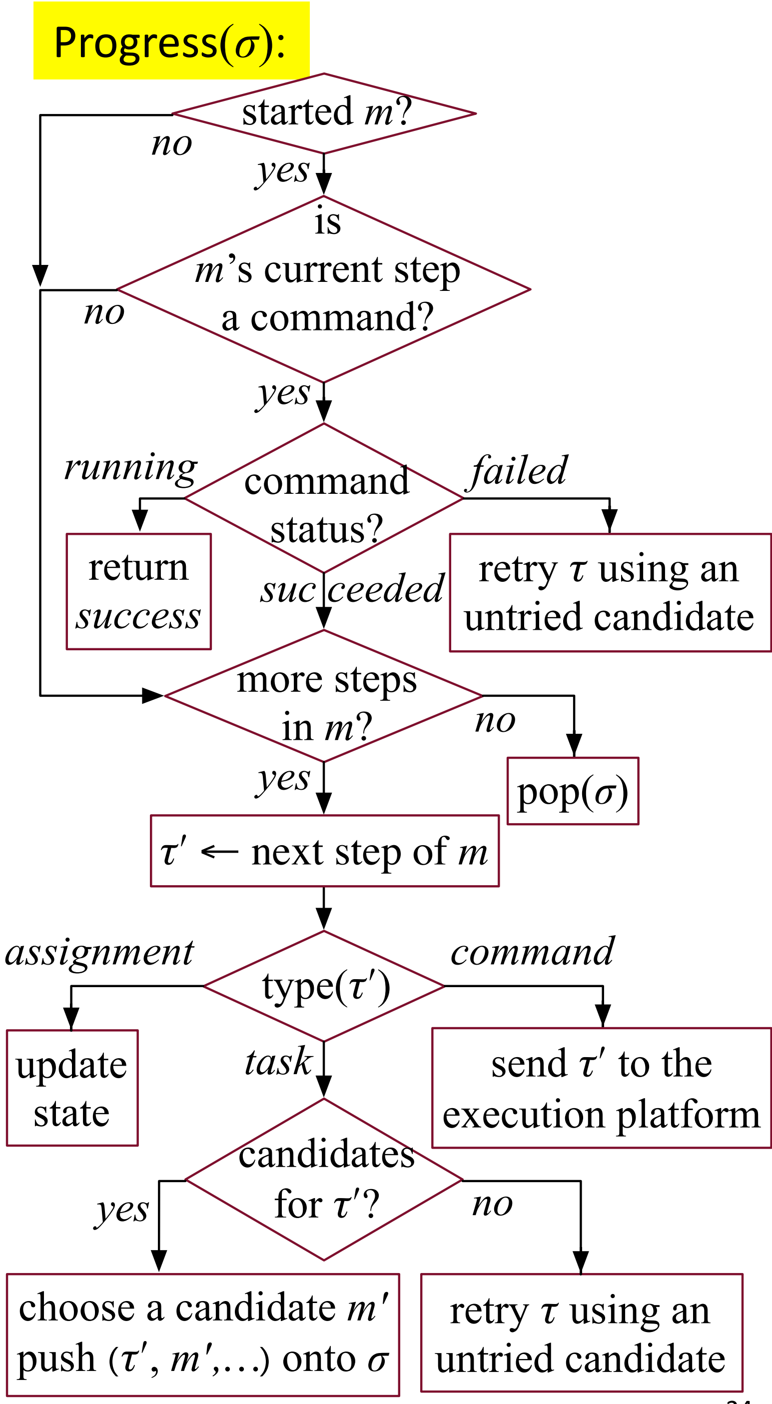
Candidates
= {m-fetch1(r1,c2),
m-fetch(r2,c2)}



m-fetch2(r,c)
task: fetch(r,c)
pre: pos(c) ≠ unknown
body:
if loc(r) = pos(c) then
take(r,c,pos(c))
else do
move-to(r,pos(c))
take(r,c,pos(c))

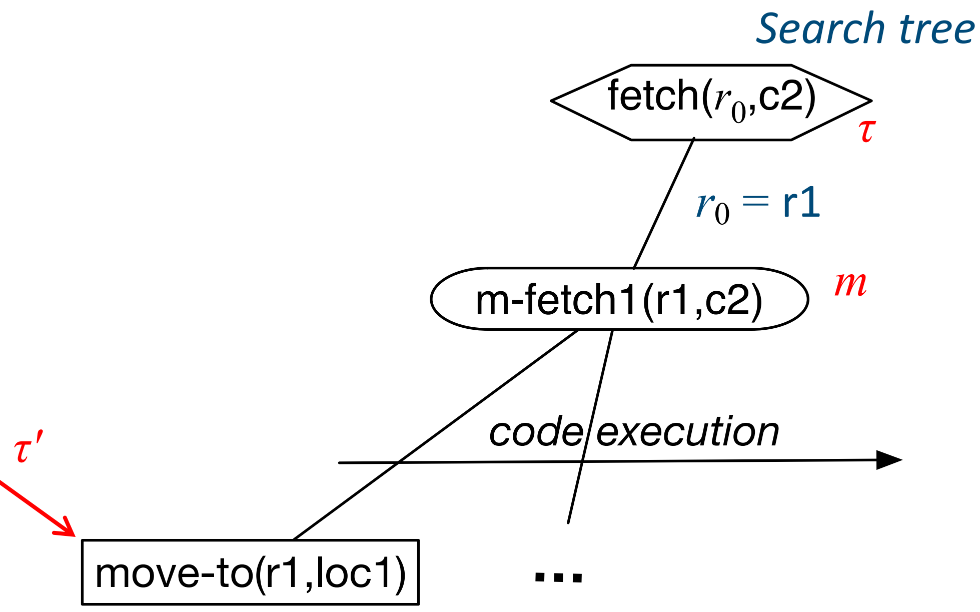
- Container locations unknown
- Partially observable
  - Robot only sees current location





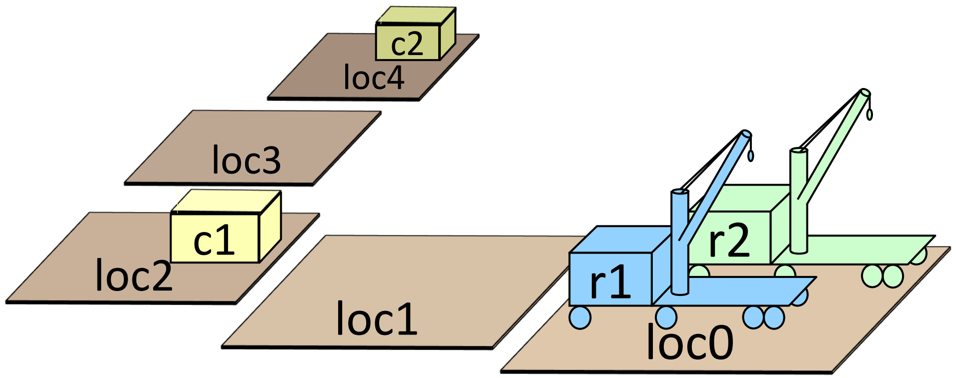
Example

m-fetch1(r,c) r = r1, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body: l = loc1
  - if $\exists l$ (view(l) = F) then
    - move-to(r,l)
    - perceive(r,l)
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail

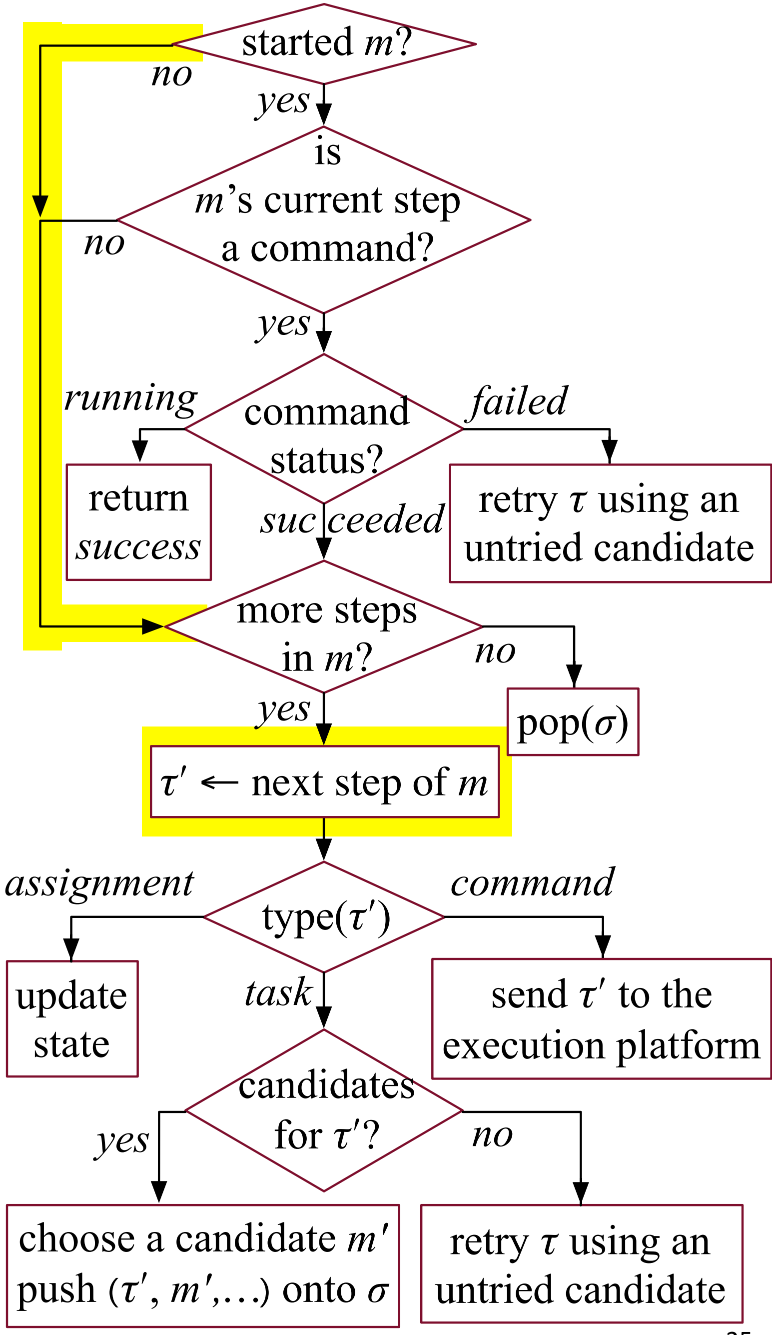


m-fetch2(r,c)
- task: fetch(r,c)
- pre: pos(c) $\neq$ unknown
- body:
  - if loc(r) = pos(c) then
    - take(r,c,pos(c))
  - else do
    - move-to(r,pos(c))
    - take(r,c,pos(c))

- Container locations unknown
- Partially observable
  - Robot only sees current location

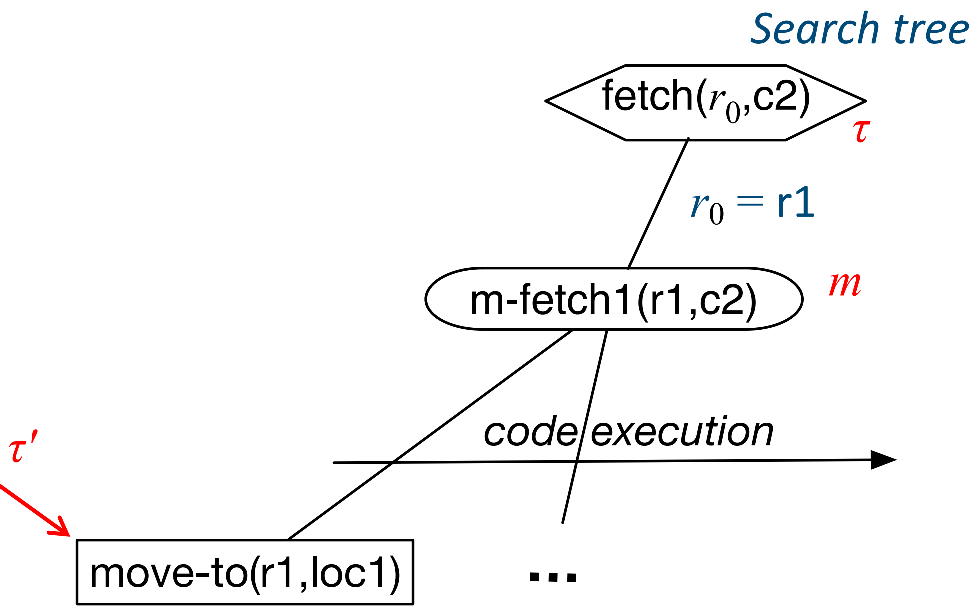


Progress(σ):



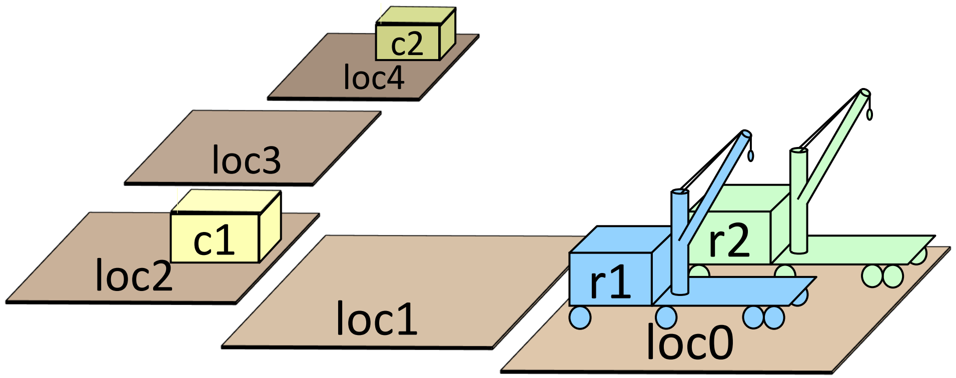
Example

m-fetch1(r,c) r = r1, c = c2
task: fetch(r,c)
pre: pos(c) = unknown
body: l = loc1
if ∃l (view(l) = F) then
move-to(r,l)
perceive(r,l)
if pos(c) = l then
take(r,c,l)
else fetch(r,c)
else fail

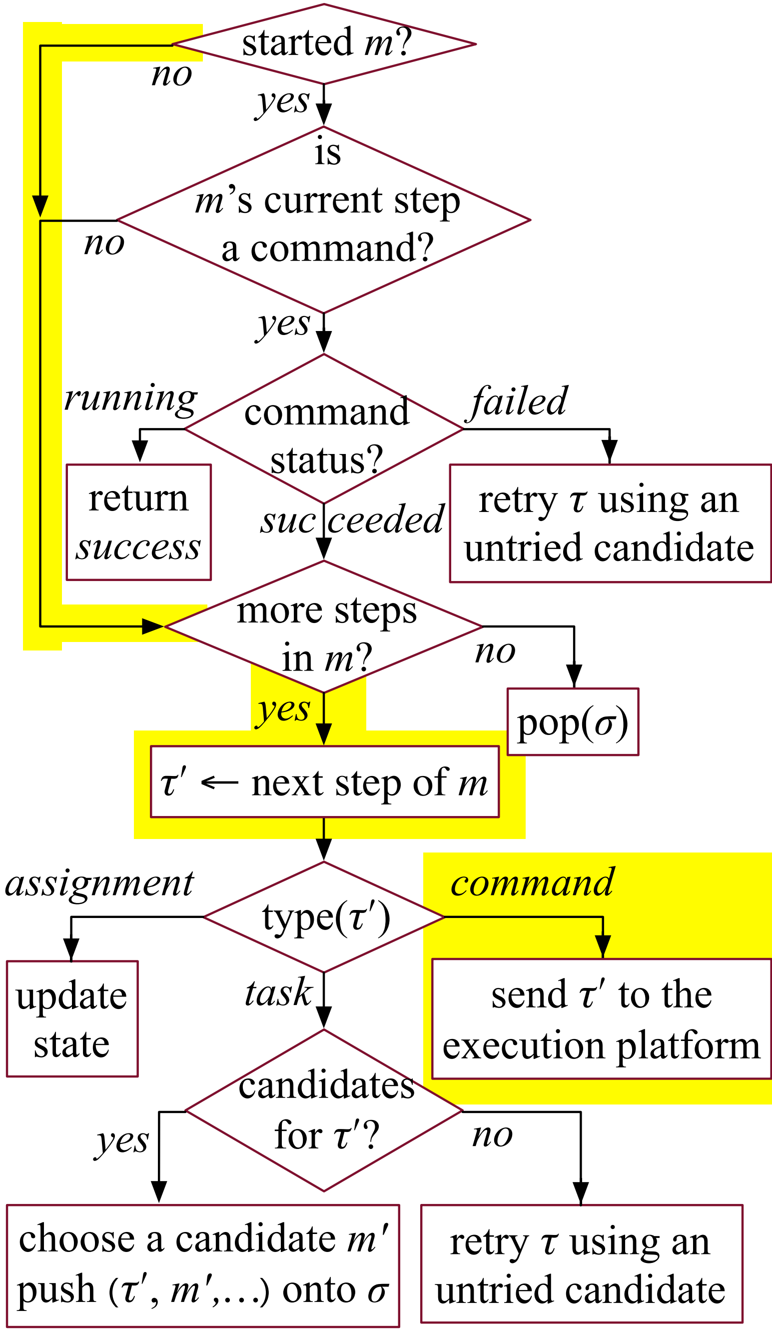


m-fetch2(r,c)
task: fetch(r,c)
pre: pos(c) ≠ unknown
body:
if loc(r) = pos(c) then
take(r,c,pos(c))
else do
move-to(r,pos(c))
take(r,c,pos(c))

- Container locations unknown
- Partially observable
  - Robot only sees current location

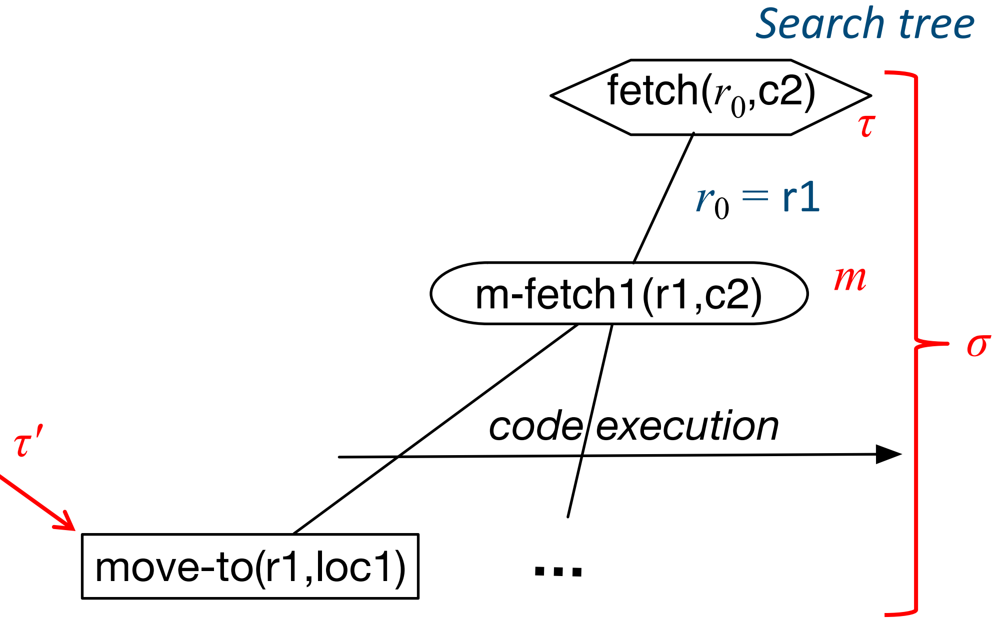


Progress(σ):



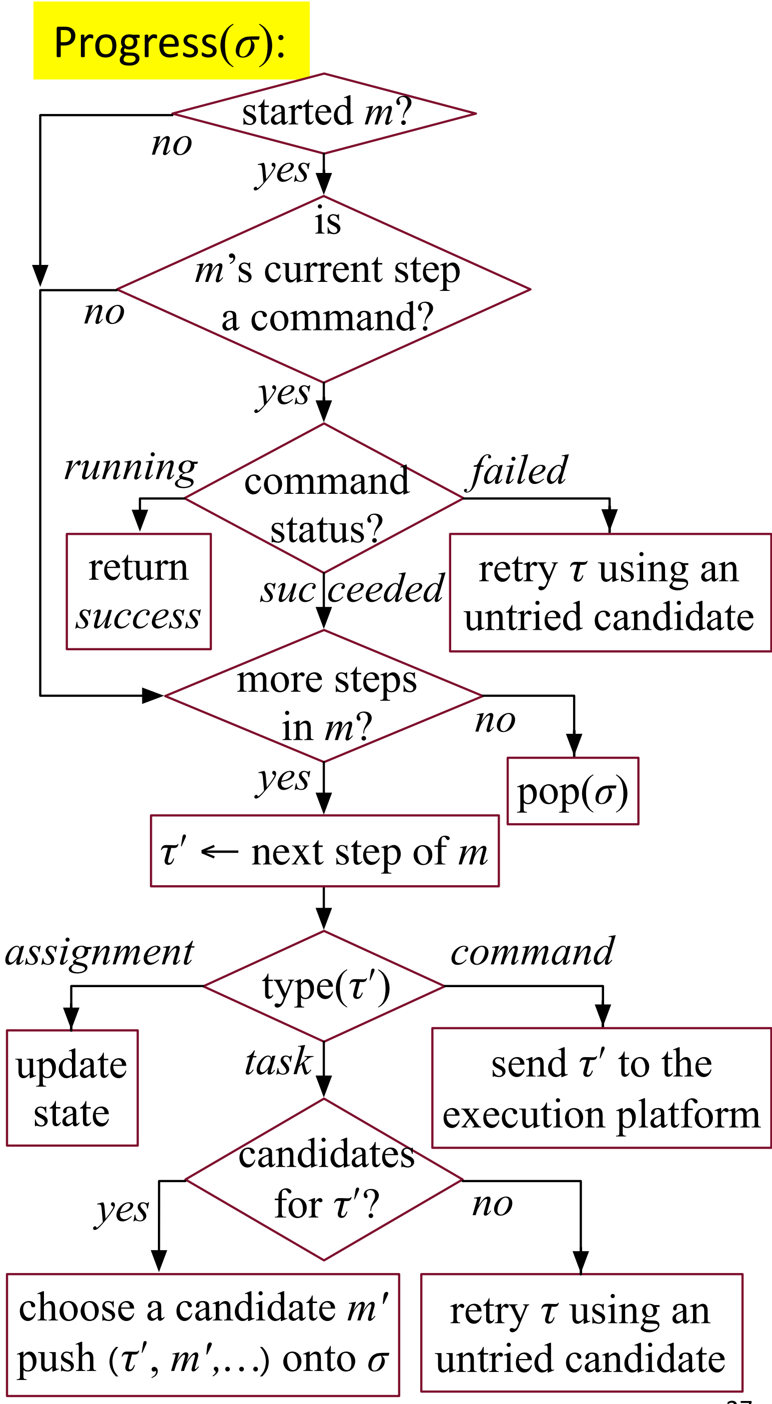
Example

m-fetch1(r,c) r = r1, c = c2
task: fetch(r,c)
pre: pos(c) = unknown
body: l = loc1
if $\exists l$ (view(l) = F) then
move-to(r,l)
perceive(r,l)
if pos(c) = l then
take(r,c,l)
else fetch(r,c)
else fail



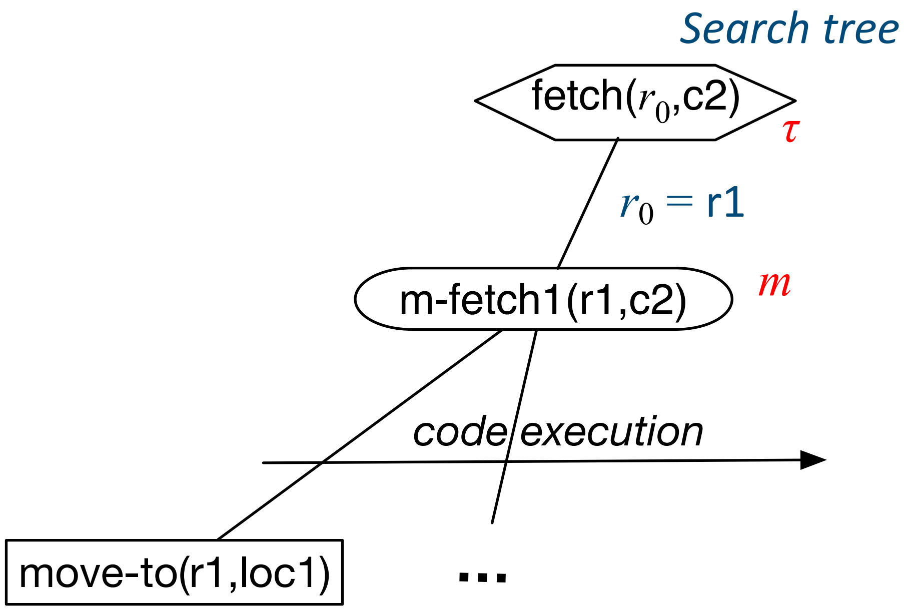
m-fetch2(r,c)
task: fetch(r,c)
pre: pos(c) $\neq$ unknown
body:
if loc(r) = pos(c) then
take(r,c,pos(c))
else do
move-to(r,pos(c))
take(r,c,pos(c))

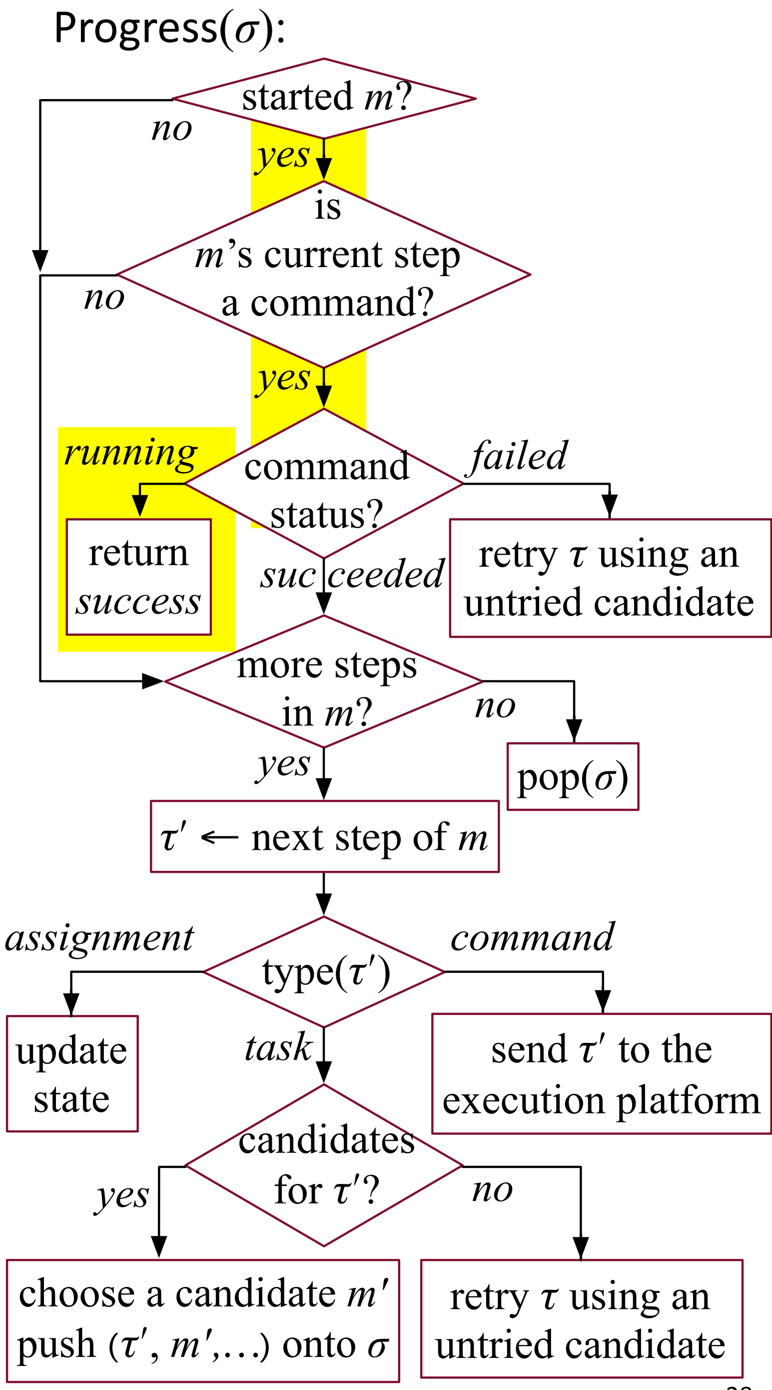
procedure Rae:
loop:
for every new external task or event τ do
choose a method instance m for τ
create a refinement stack for τ, m
add the stack to *Agenda*
for each stack σ in *Agenda*
Progress(σ)
if σ is finished then remove it



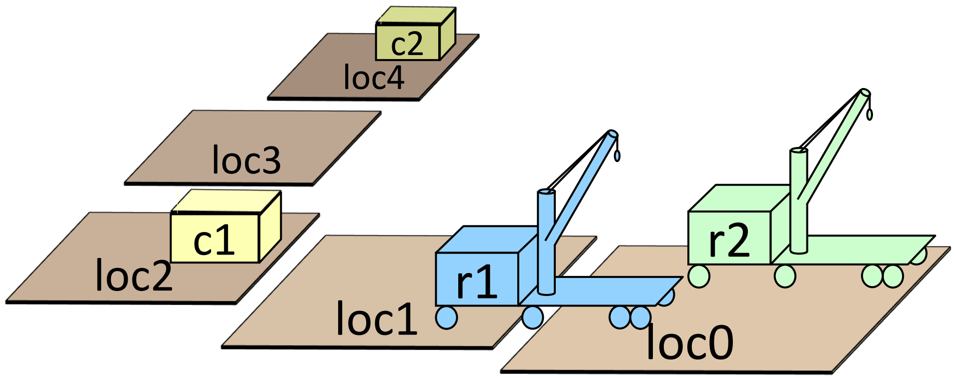
Example

m-fetch1(r,c) r = r1, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body: l = loc1
  - if $\exists l$ (view(l) = F) then
    - move-to(r,l) ← *running …*
    - perceive(r,l)
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail



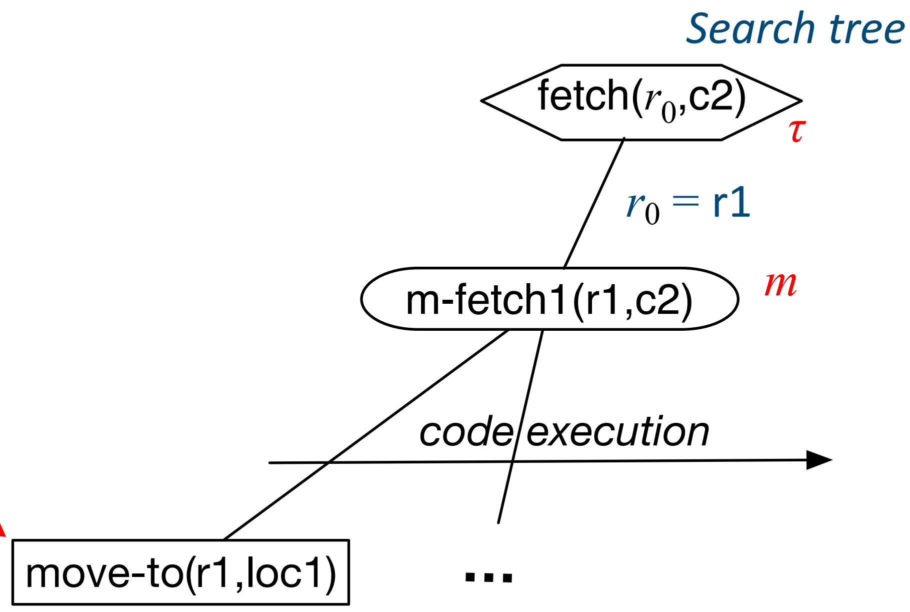


m-fetch2(r,c)
- task: fetch(r,c)
- pre: pos(c) ≠ unknown
- body:
  - if loc(r) = pos(c) then
    - take(r,c,pos(c))
  - else do
    - move-to(r,pos(c))
    - take(r,c,pos(c))

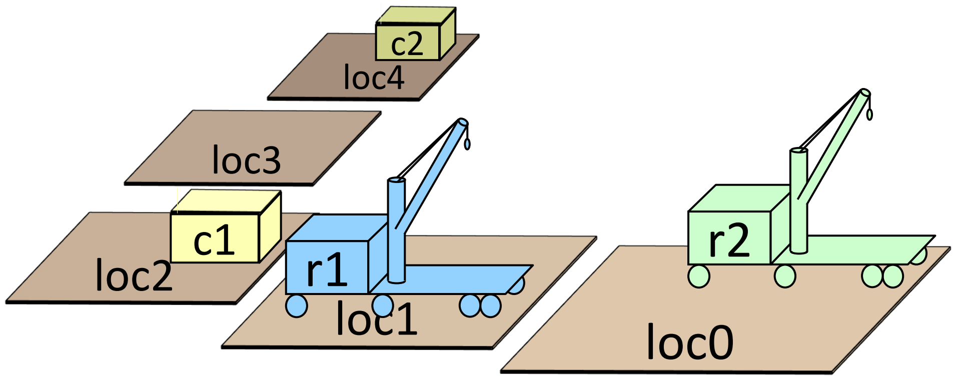


Example

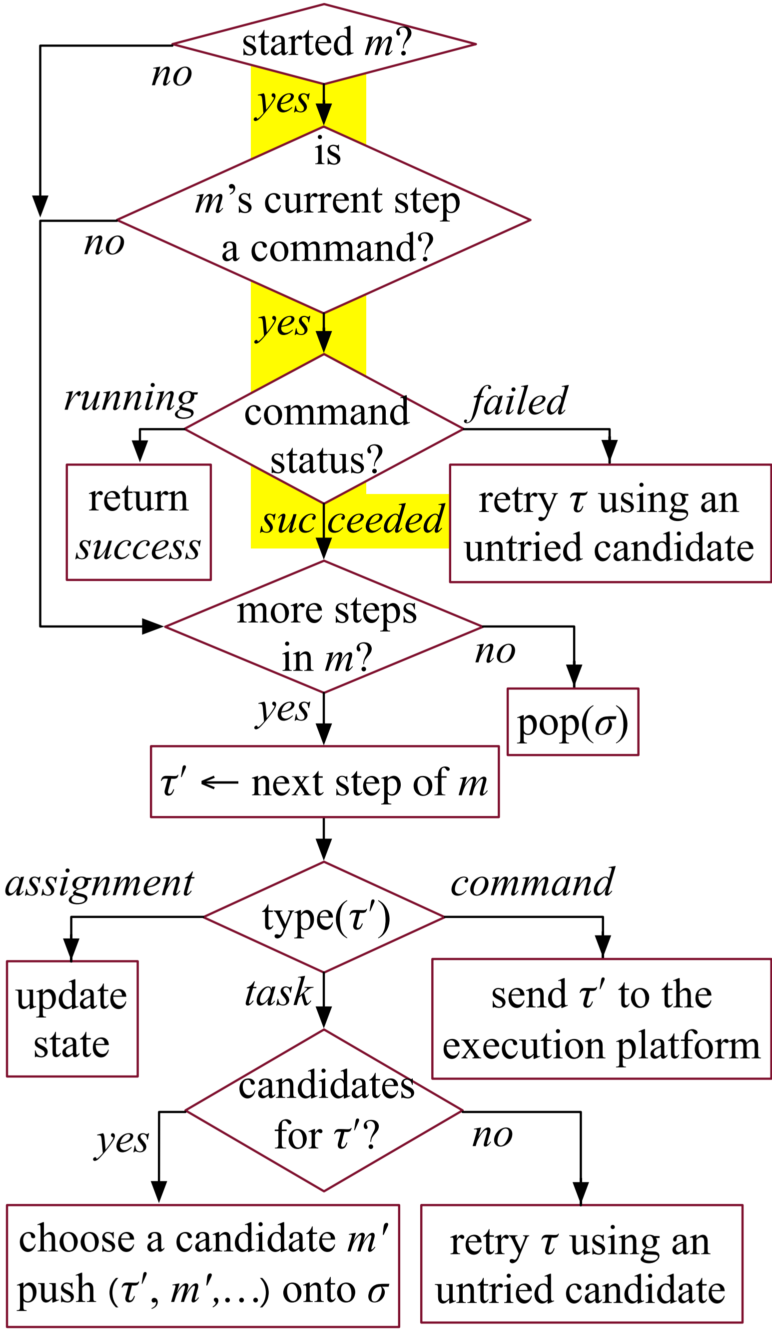
m-fetch1(r,c) r = r1, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body: l = loc1
  - if ∃l (view(l) = F) then
    - move-to(r,l) ← *succeeded*
    - perceive(r,l)
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail



m-fetch2(r,c)
- task: fetch(r,c)
- pre: pos(c) ≠ unknown
- body:
  - if loc(r) = pos(c) then
    - take(r,c,pos(c))
  - else do
    - move-to(r,pos(c))
    - take(r,c,pos(c))

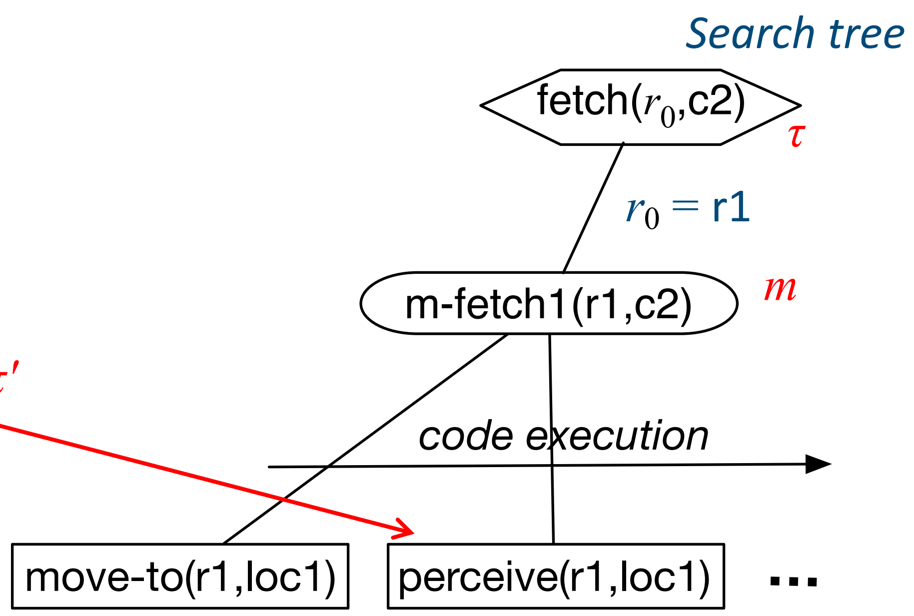


Progress(σ):

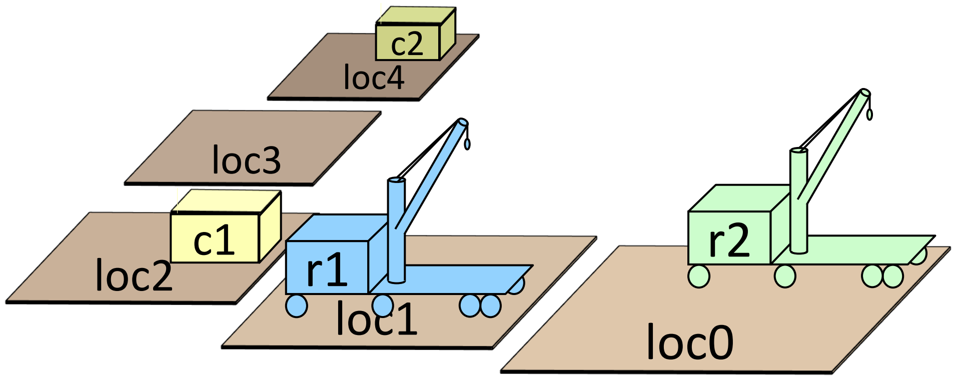


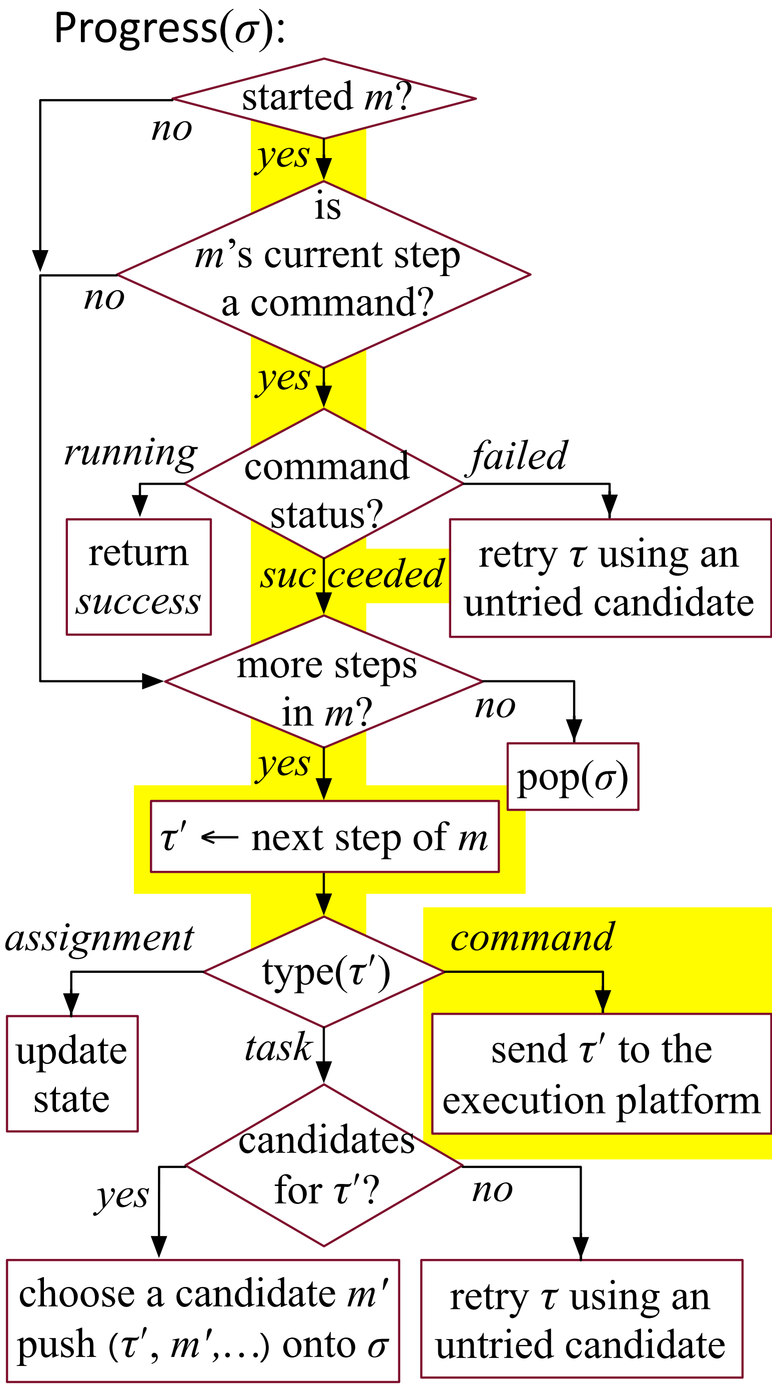
Example

m-fetch1(r,c) r = r1, c = c2
task: fetch(r,c)
pre: pos(c) = unknown
body: l = loc1
if ∃l (view(l) = F) then
move-to(r,l)
perceive(r,l)
if pos(c) = l then
take(r,c,l)
else fetch(r,c)
else fail



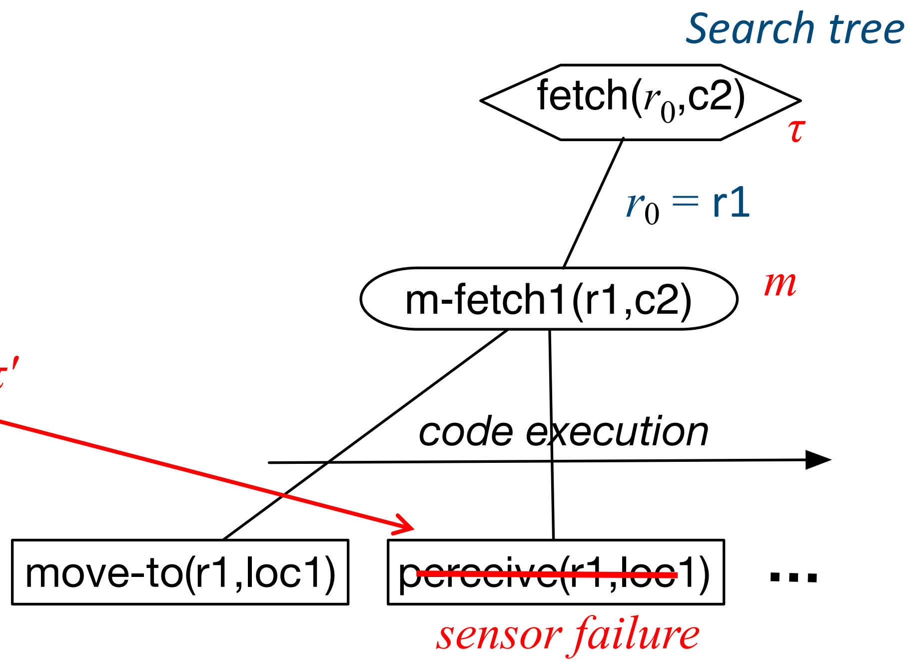
m-fetch2(r,c)
task: fetch(r,c)
pre: pos(c) ≠ unknown
body:
if loc(r) = pos(c) then
take(r,c,pos(c))
else do
move-to(r,pos(c))
take(r,c,pos(c))



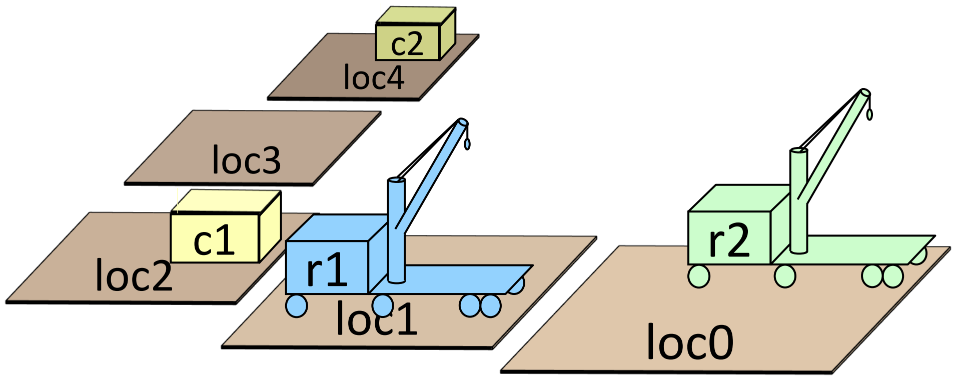


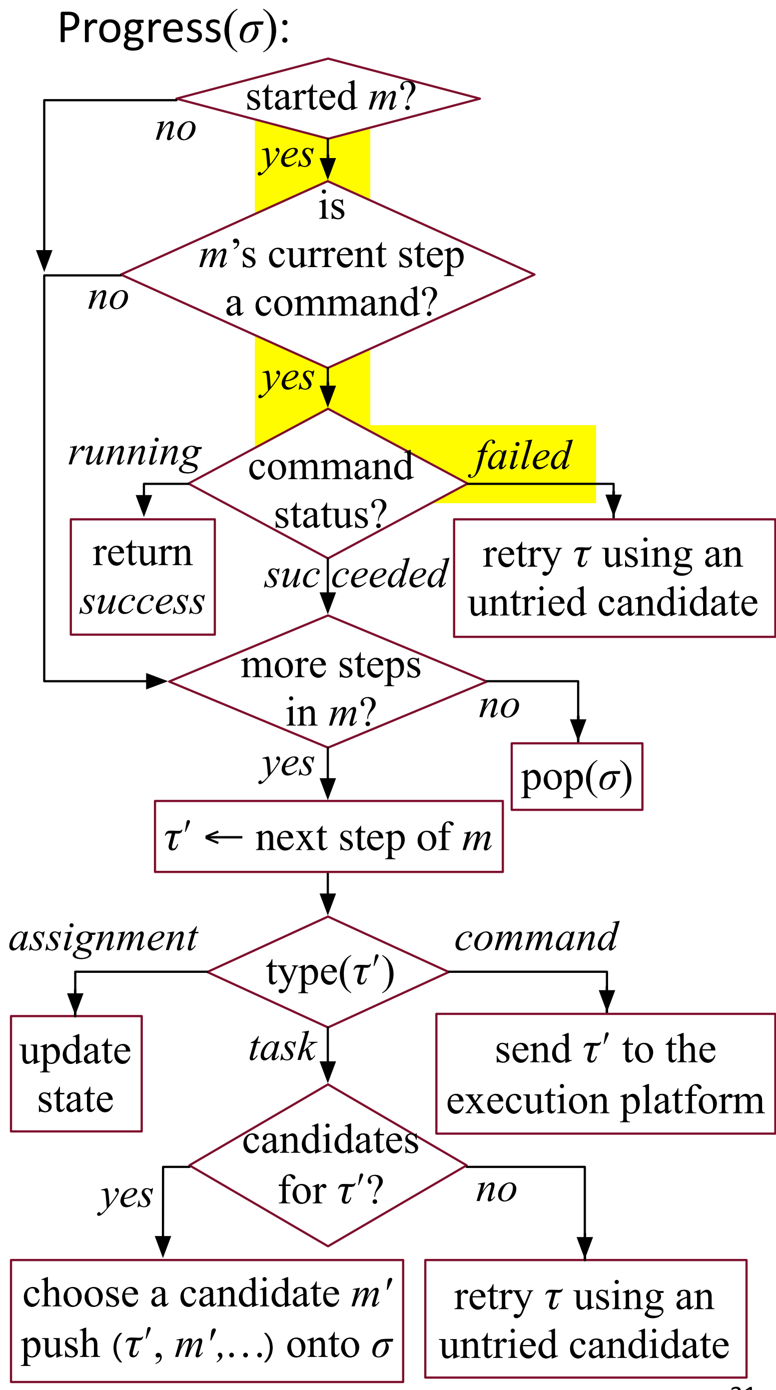
Example

m-fetch1(r,c) r = r1, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body: l = loc1
  - if $\exists l$ (view(l) = F) then
    - move-to(r,l)
    - perceive(r,l) ← *failed*
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail



m-fetch2(r,c)
- task: fetch(r,c)
- pre: pos(c) $\neq$ unknown
- body:
  - if loc(r) = pos(c) then
    - take(r,c,pos(c))
  - else do
    - move-to(r,pos(c))
    - take(r,c,pos(c))

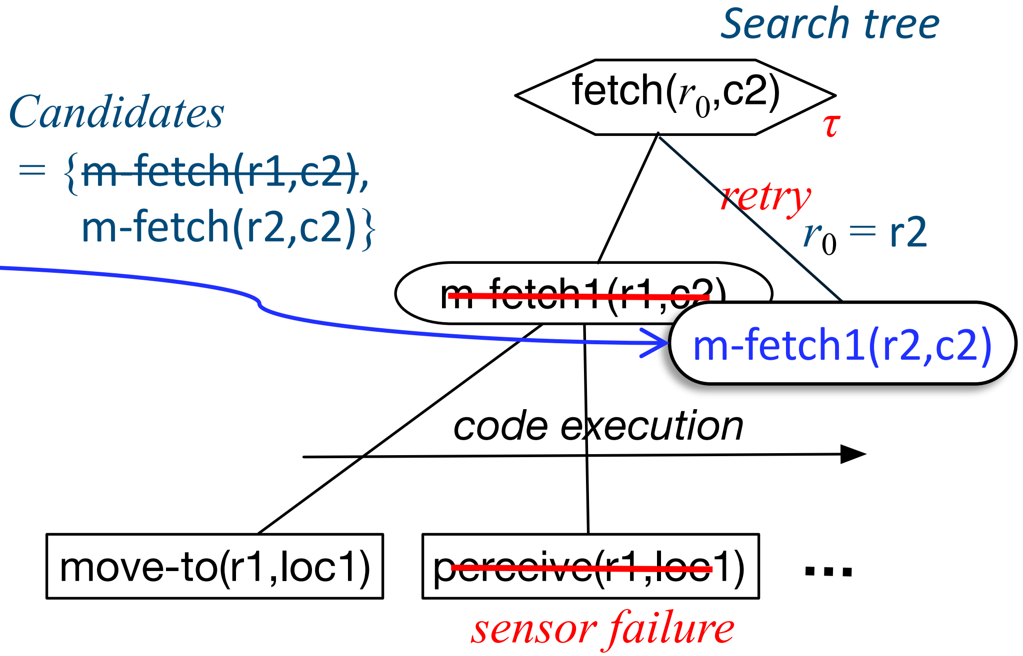


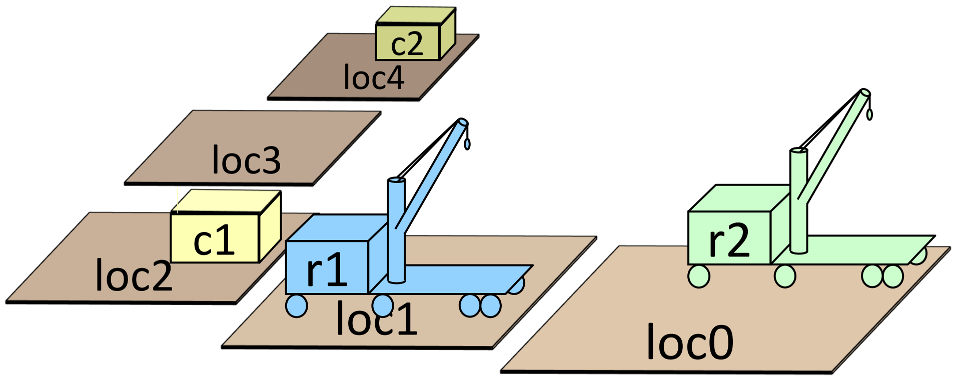


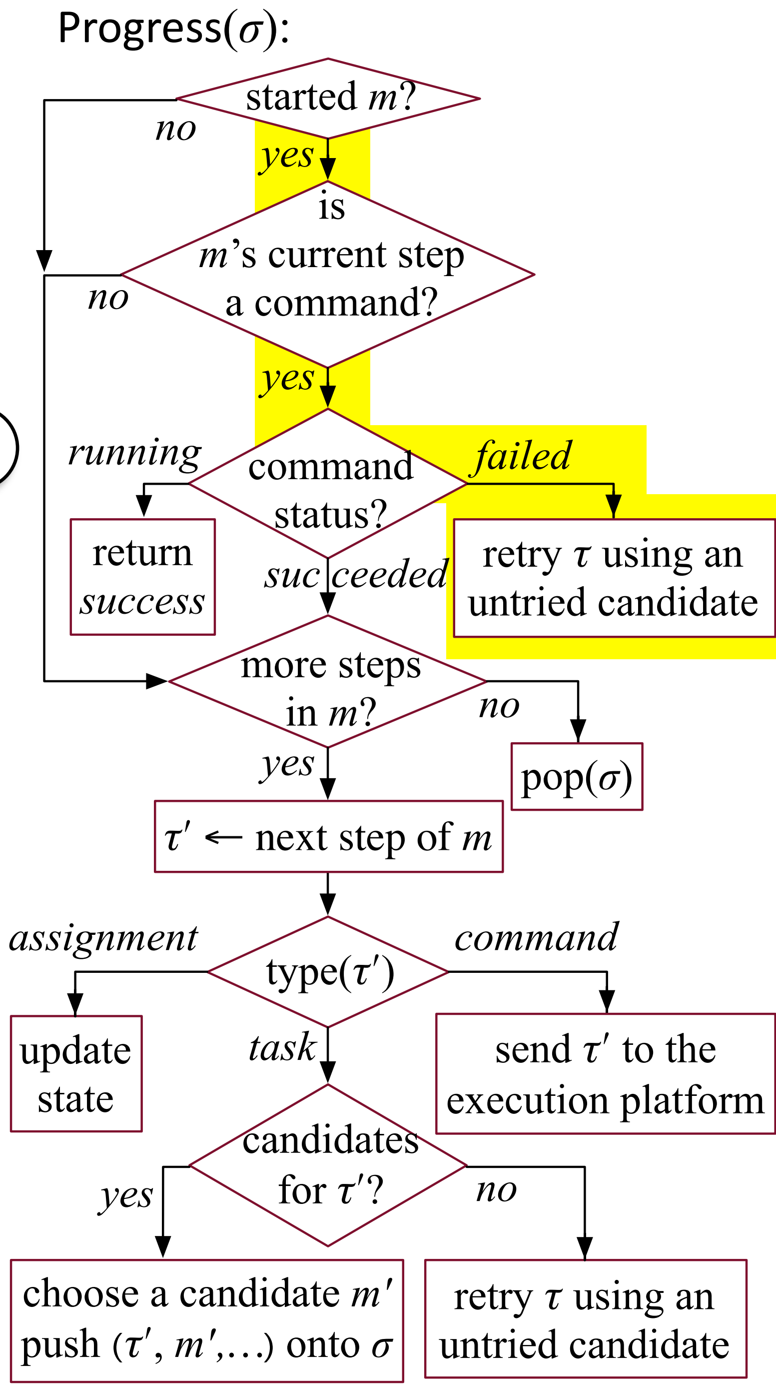
Example

```
m-fetch1(r,c)   r = r2, c = c2
    task:   fetch(r,c)
    pre:    pos(c) = unknown
    body:
       if ∃l (view(l) = F) then
           move-to(r,l)
           perceive(r,l)
           if pos(c) = l then
                 take(r,c,l)
           else fetch(r,c)
       else fail
```

```
m-fetch2(r,c)
    task:   fetch(r,c)
    pre:    pos(c) ≠ unknown
    body:
       if loc(r) = pos(c) then
           take(r,c,pos(c))
       else do
           move-to(r,pos(c))
           take(r,c,pos(c))
```



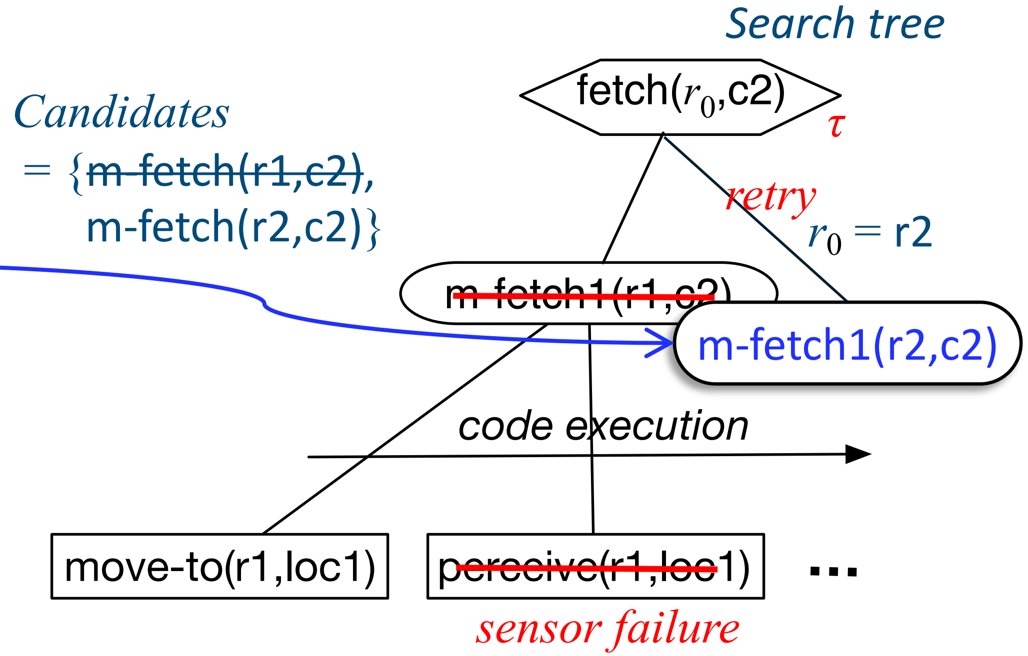




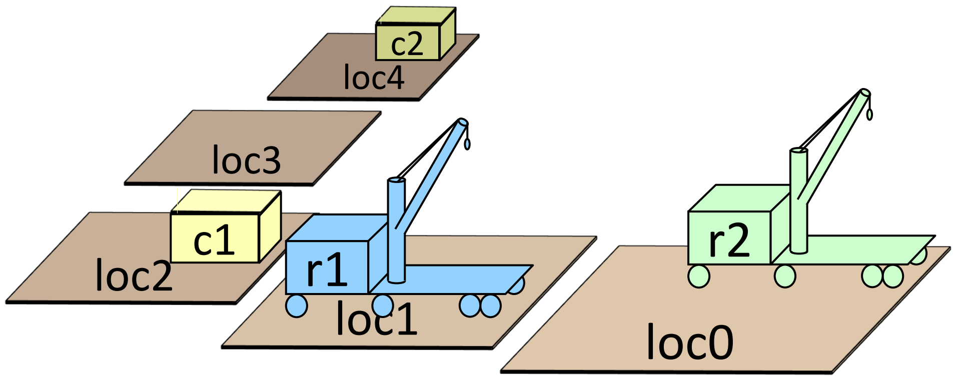
Example

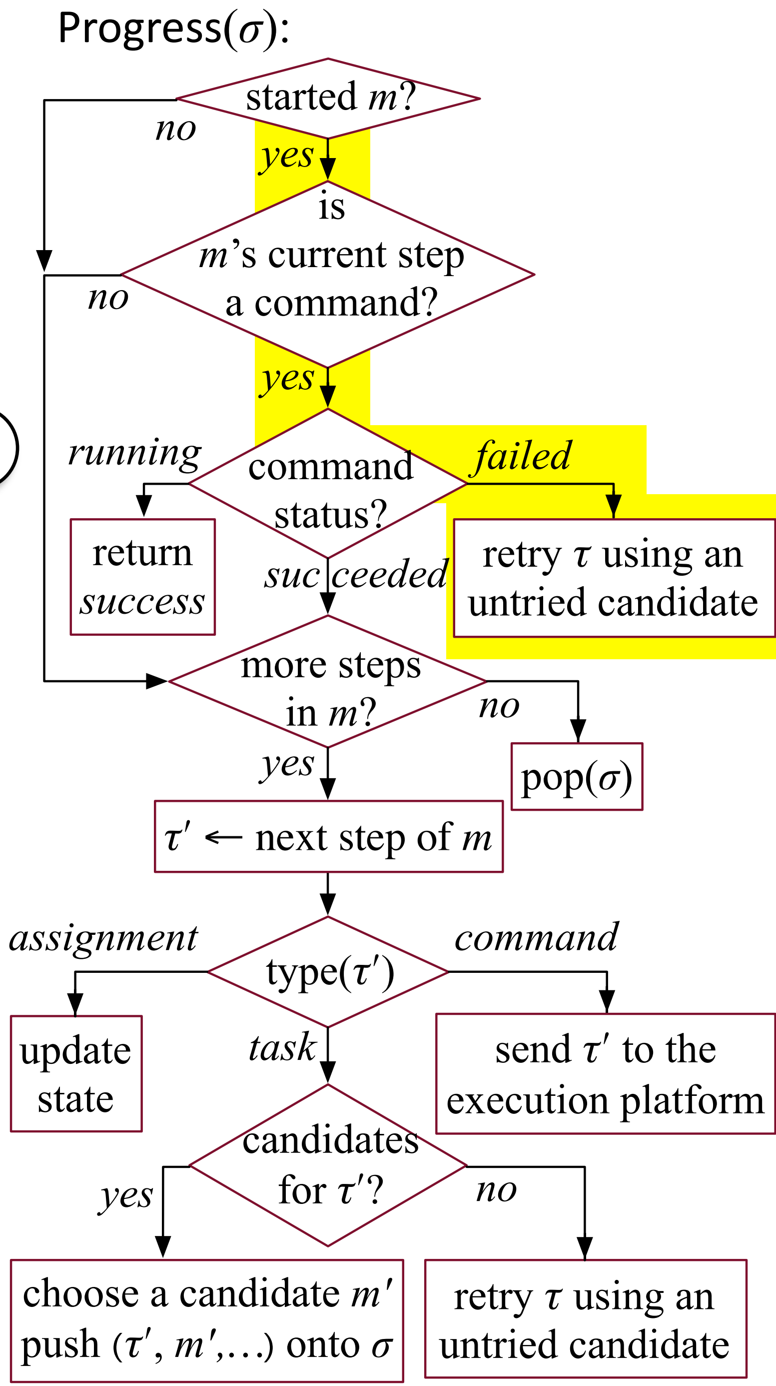
m-fetch1(r,c) r = r2, c = c2
task: fetch(r,c)
pre: pos(c) = unknown
body:
if ∃l (view(l) = F) then
move-to(r,l)
perceive(r,l)
if pos(c) = l then
take(r,c,l)
else fetch(r,c)
else fail

m-fetch2(r,c)
task: fetch(r,c)
pre: pos(c) ≠ unknown
body:
if loc(r) = pos(c) then
take(r,c,pos(c))
else do
move-to(r,pos(c))
take(r,c,pos(c))



Is this the same as a backtracking search?





Extensions to Rae

- Methods for events
  - e.g., an emergency
- Methods for goals
  - special kind of task: achieve(goal)
  - sets up a monitor to see if the goal has been achieved
- Concurrent subtasks

Outline

- **Motivation**
- **Representation** – state variables, commands, tasks, refinement methods
- **Acting** – Rae (Refinement Acting Engine)
- **Planning** – UPOM (UCT-like Planner for Operational Models)
- **Acting with Planning** – Rae + UPOM
- **Using the implementation** – Rae code, UPOM code, examples

Why Plan?

procedure Rae:

loop:

for every new external task or event τ do

choose a method instance m for τ

create a refinement stack for τ, m

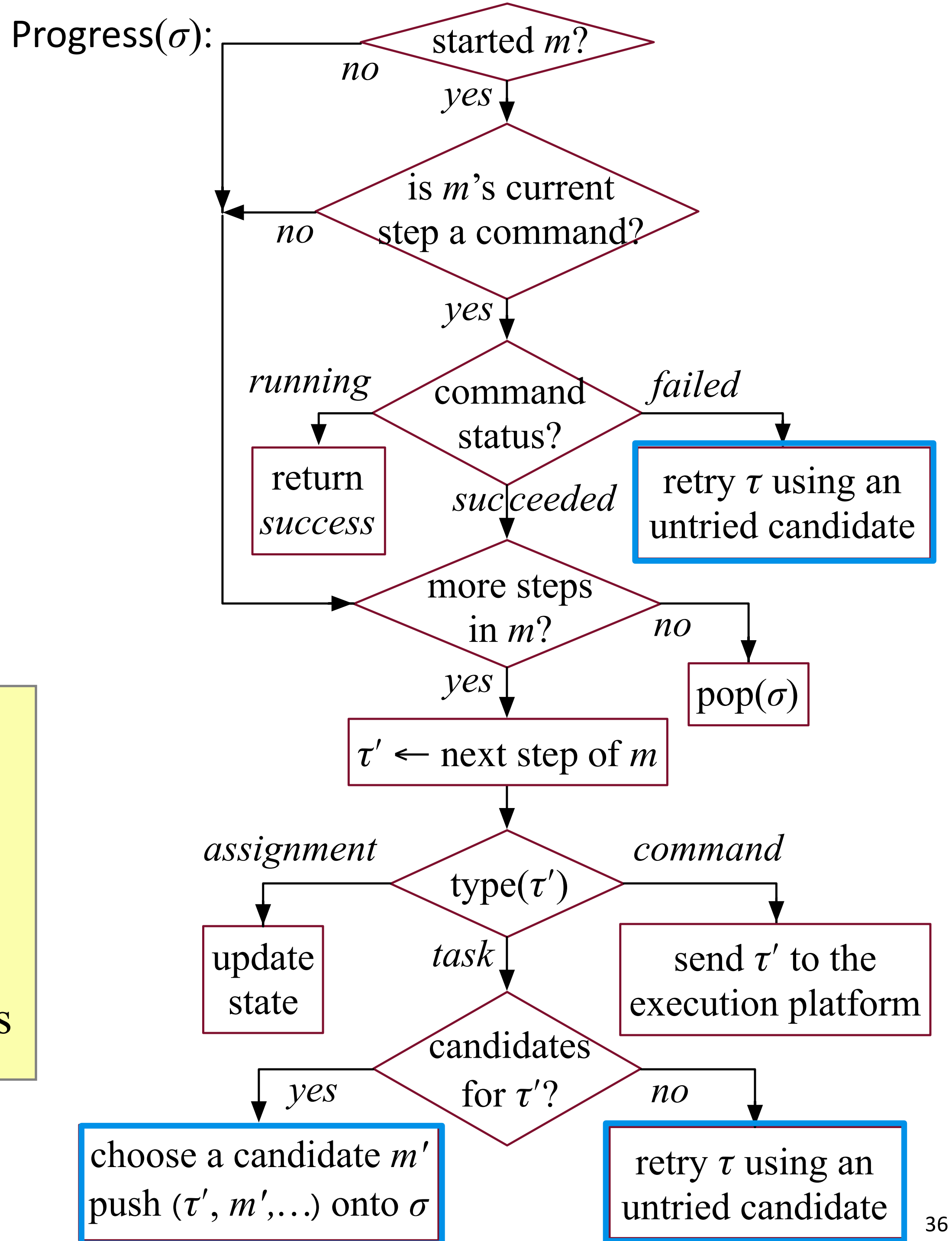
add the stack to *Agenda*

for each stack σ in *Agenda*

Progress(σ)

if σ is finished then remove it

- Bad choice may lead to
  - more costly solution
  - failure, need to recover
  - unrecoverable failure
- Idea: do simulations to predict outcomes

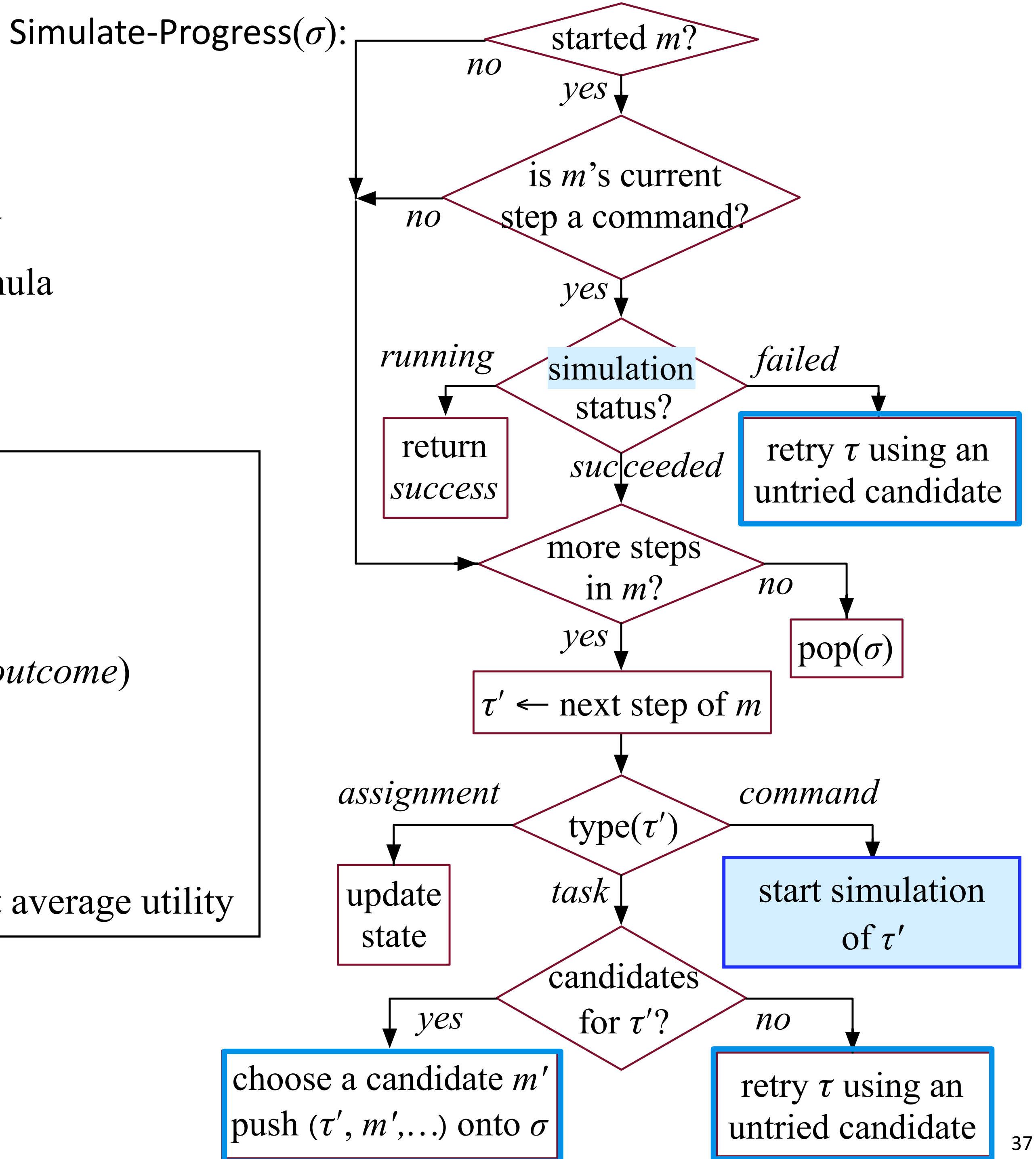


Planner

- Basic ideas
  - Repeated Monte Carlo rollouts on a single task *t*
  - Choose method instances using a UCT-like formula
  - Simulated execution of commands

UPOM(τ):
  choose a method instance m for τ
  create refinement stack σ for τ and m
  loop while Simulate-Progress(σ) $\neq$ *failure*
    if σ is completed then return (m, *utility of outcome*)
  return *failure*

UPOM-Lookahead (task τ):
  Call UPOM(τ) multiple times
  Return the $m \in$ *Candidates* that has the highest average utility



Simulating a command

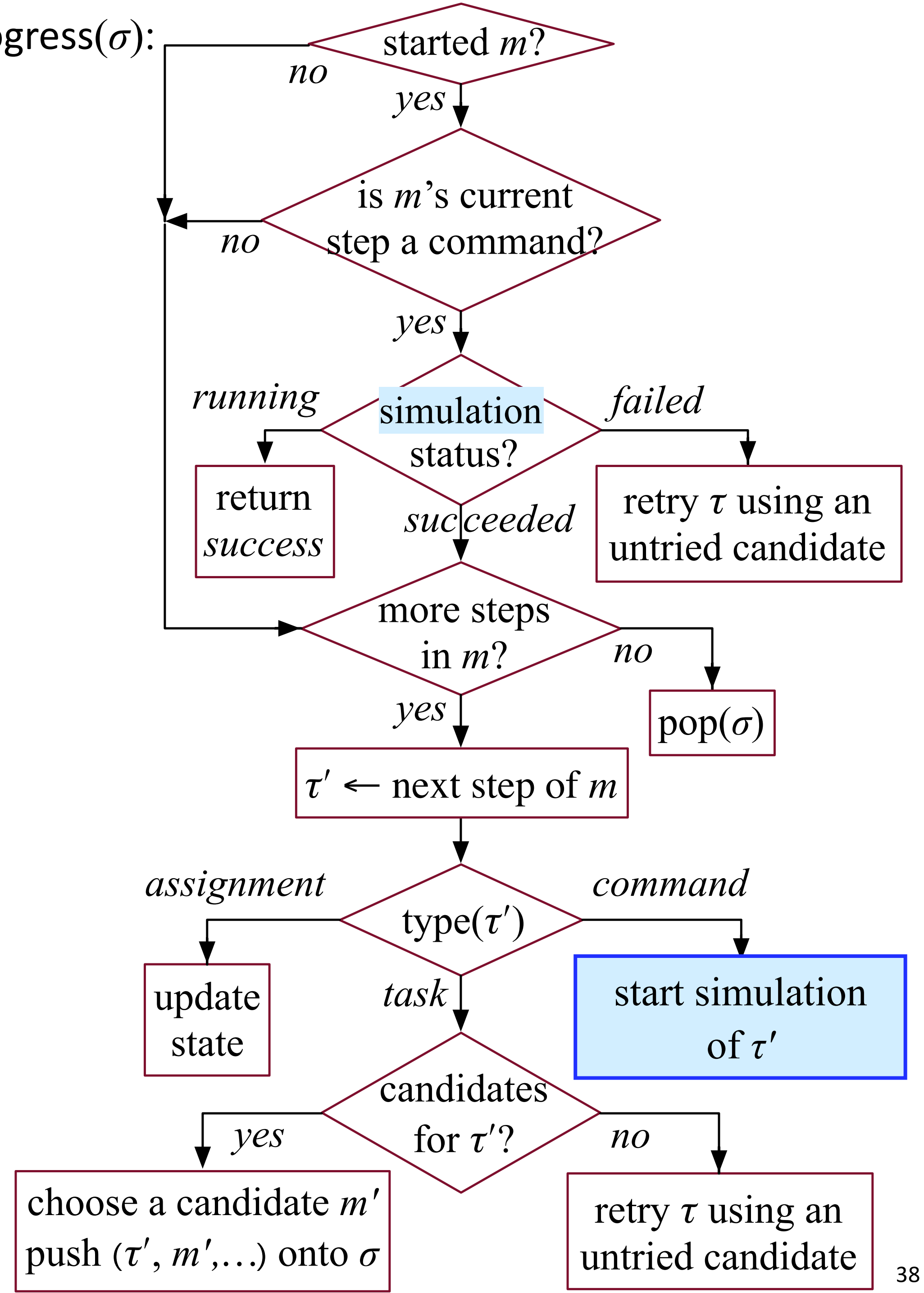
- Simplest case:
  - probabilistic action template

    $a(x_1, \ldots, x_k)$
    pre: …
    (p_1) eff_1: $e_{11}, e_{12}, \ldots$
    …
    (p_m) eff_m: $e_{m1}, e_{m2}, \ldots$

  - Choose randomly, each eff_i has probability p_i
  - Use eff_i to update the current state

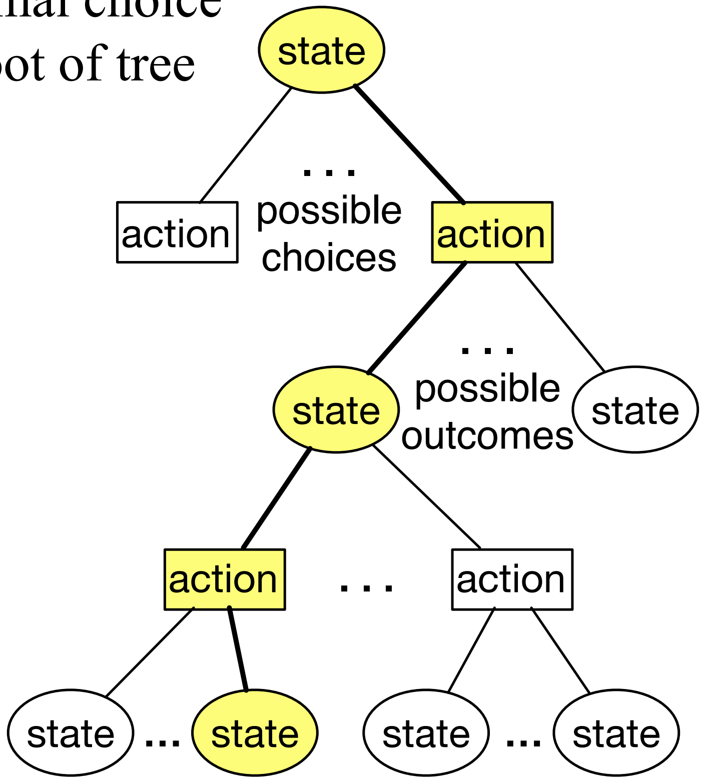
- More general:
  - Arbitrary computation, e.g., physics-based simulation
  - Run the code to get prediction of effects

Simulate-Progress(σ):



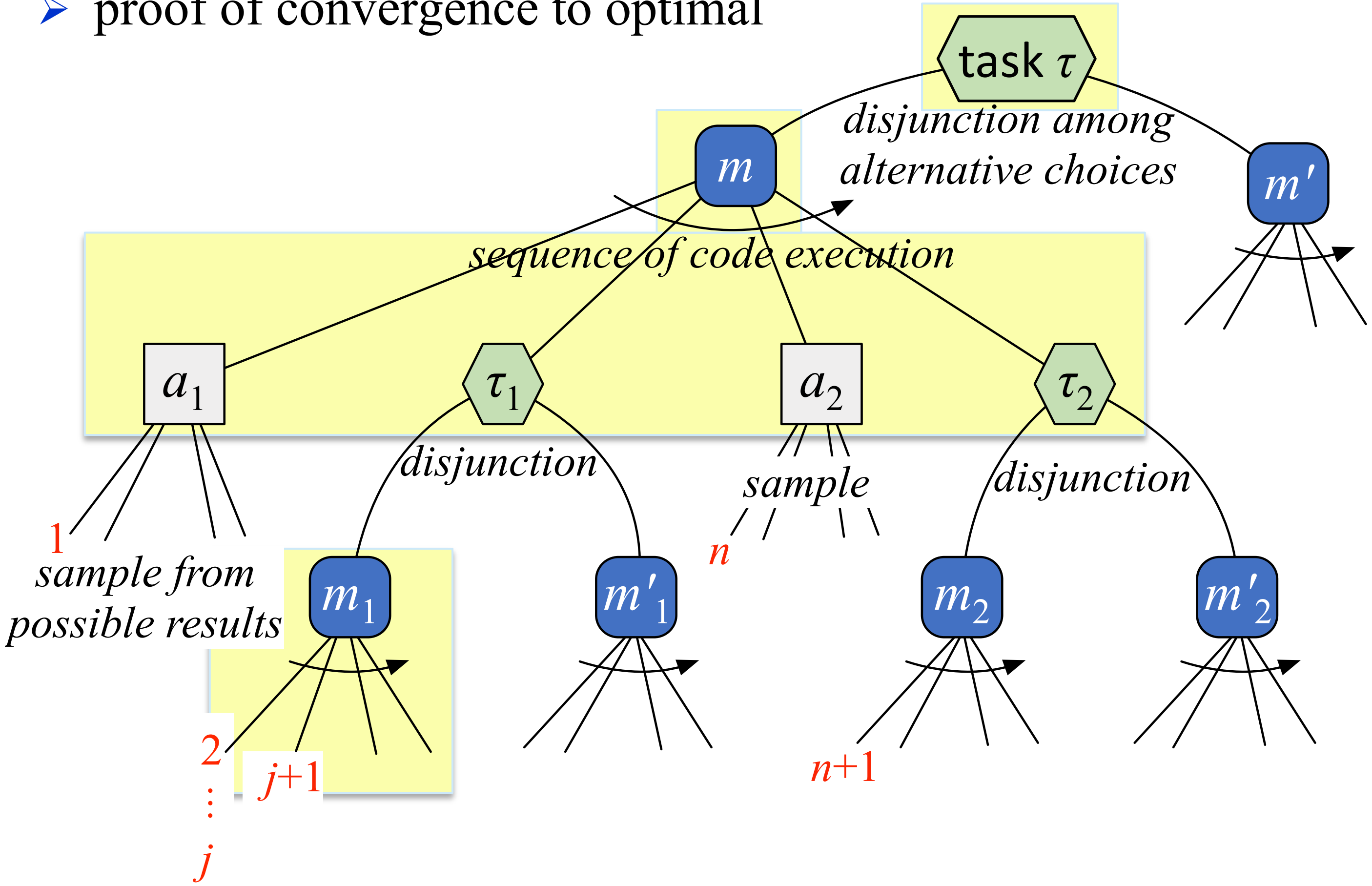
Monte Carlo Rollouts

- Rollouts on MDPs
  - At each state, choose action at random, get random outcome
- UCT algorithm
  - Choice of action balances exploration vs exploitation
  - Converges to optimal choice at root of tree



- UPOM search tree more complicated
  - tasks, methods, commands, code execution
- If no exogenous events, can map it into UCT on a complicated MDP
  - proof of convergence to optimal

method instance m
  task: τ
  pre: …
  body:
    action a_1
    task τ_1
    action a_2
    task τ_2



Outline

- **Motivation**
- **Representation** – state variables, commands, tasks, refinement methods
- **Acting** – Rae (Refinement Acting Engine)
- **Planning** – UPOM (UCT-like Planner for Operational Models)
- **Acting with Planning** – Rae + UPOM
- **Using the implementation** – Rae code, UPOM code, examples

RAE + UPOM

procedure Rae:

loop:

for every new external task or event τ do

choose a method instance m for τ

create a refinement stack for τ, m

add the stack to *Agenda*

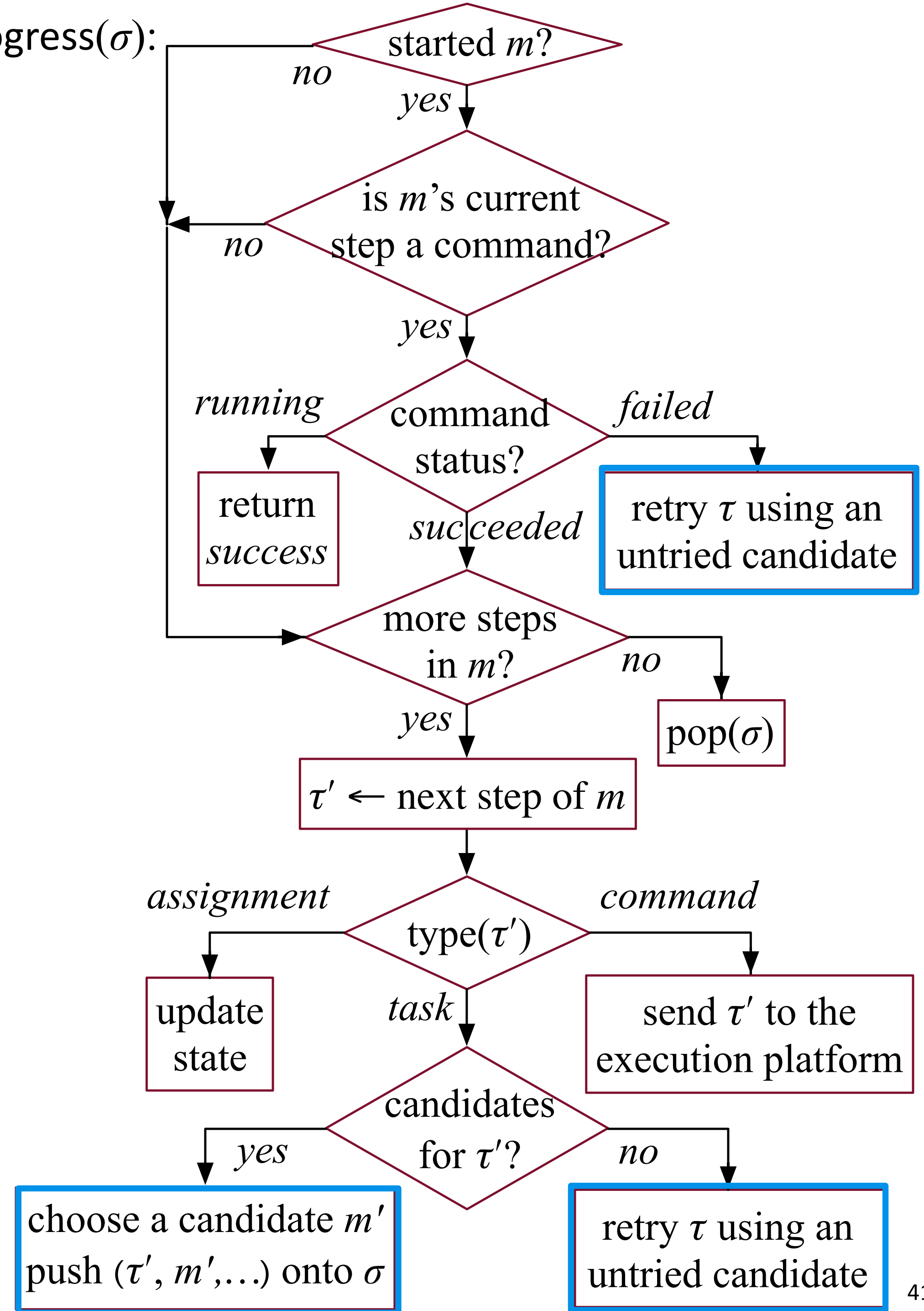
for each stack σ in *Agenda*

Progress(σ)

if σ is finished then remove it

- Whenever RAE needs to choose a method instance, use UPOM-Lookahead to make the choice

Progress(σ):



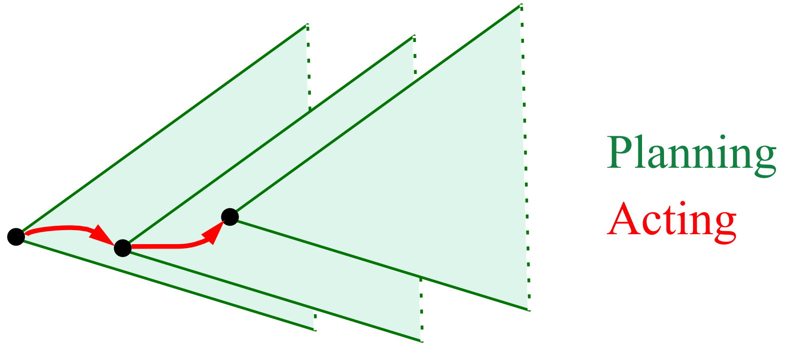
Summary of Experimental Results

Domain	$\|\mathcal{T}\|$	$\|\mathcal{M}\|$	$\|\overline{\mathcal{M}}\|$	$\|\mathcal{A}\|$	Dynamic events	Dead ends	Sensing	Robot collaboration	Concurrent tasks
S&R	8	16	16	14	✓	✓	✓	✓	✓
Explore	9	17	17	14	✓	✓	✓	✓	✓
Fetch	7	10	10	9	✓	✓	✓	–	✓
Nav	6	9	15	10	✓	–	✓	✓	✓
Deliver	6	6	50	9	✓	✓	–	✓	✓

- Five different domains, different combinations of characteristics
- Evaluation criteria:
  - Efficiency, successes vs failures, how many retries
- Result: planning helps
  - Rae operates better with UPOM than without
  - Rae operates better with more planning than with less planning

Other Details

- **Receding horizon**
  - Cut off search before accomplishing τ
    - e.g., depth d_{max} or when we run out of time
  - At leaf nodes, use heuristic function



- **Learning a heuristic function**
  - Supervised learning

Outline

1. **Motivation**
2. **Representation** – state variables, commands, tasks, refinement methods
3. **Acting** – Rae (Refinement Acting Engine)
4. **Planning** – UPOM (UCT-like Planner for Operational Models)
5. **Acting with Planning** – Rae + UPOM
6. **Using the implementation** – Rae code, UPOM code, examples

Code Demo

- **Github repository:** https://github.com/sunandita/ICAPS_Summer_School_RAE_2020

- System requirements:
  - Unix based operating system preferred
  - Have Docker or the Python Conda environment preinstalled
- Things to play with:
  - Domain file: ICAPS_Summer_School_RAE_2020/domains/domain_x.py
  - Problem file: ICAPS_Summer_School_RAE_2020 /problems/x/problemId_x.py
  - x ∊ [chargeableRobot, explorableEnv, searchAndRescue, springDoor, orderFulfillment]
- How to run?
  - cd ICAPS_Summer_School_RAE_2020/RAE_and_UPOM
  - python3 testRAEandUPOM.py –h