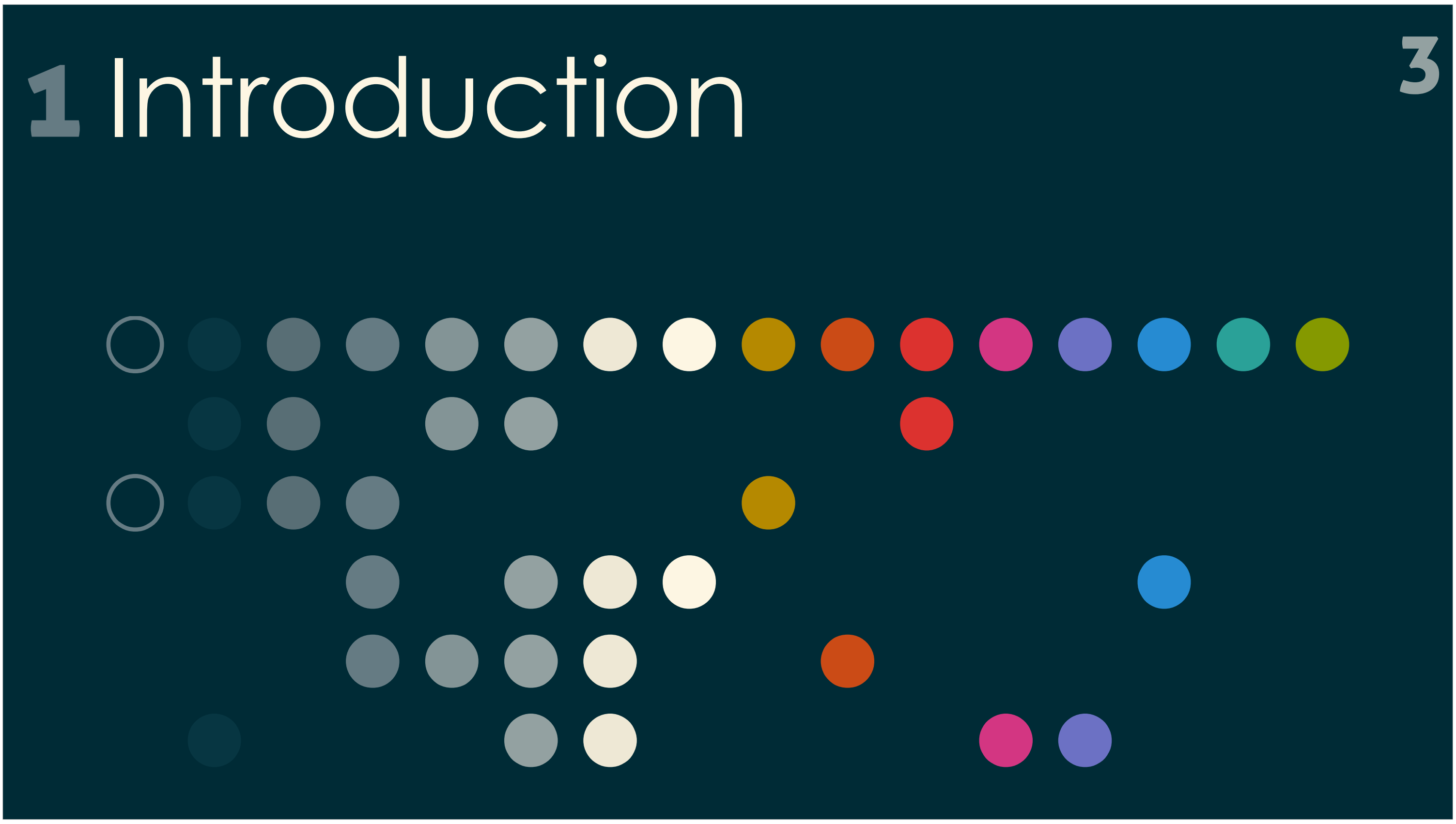
Présentation du jury

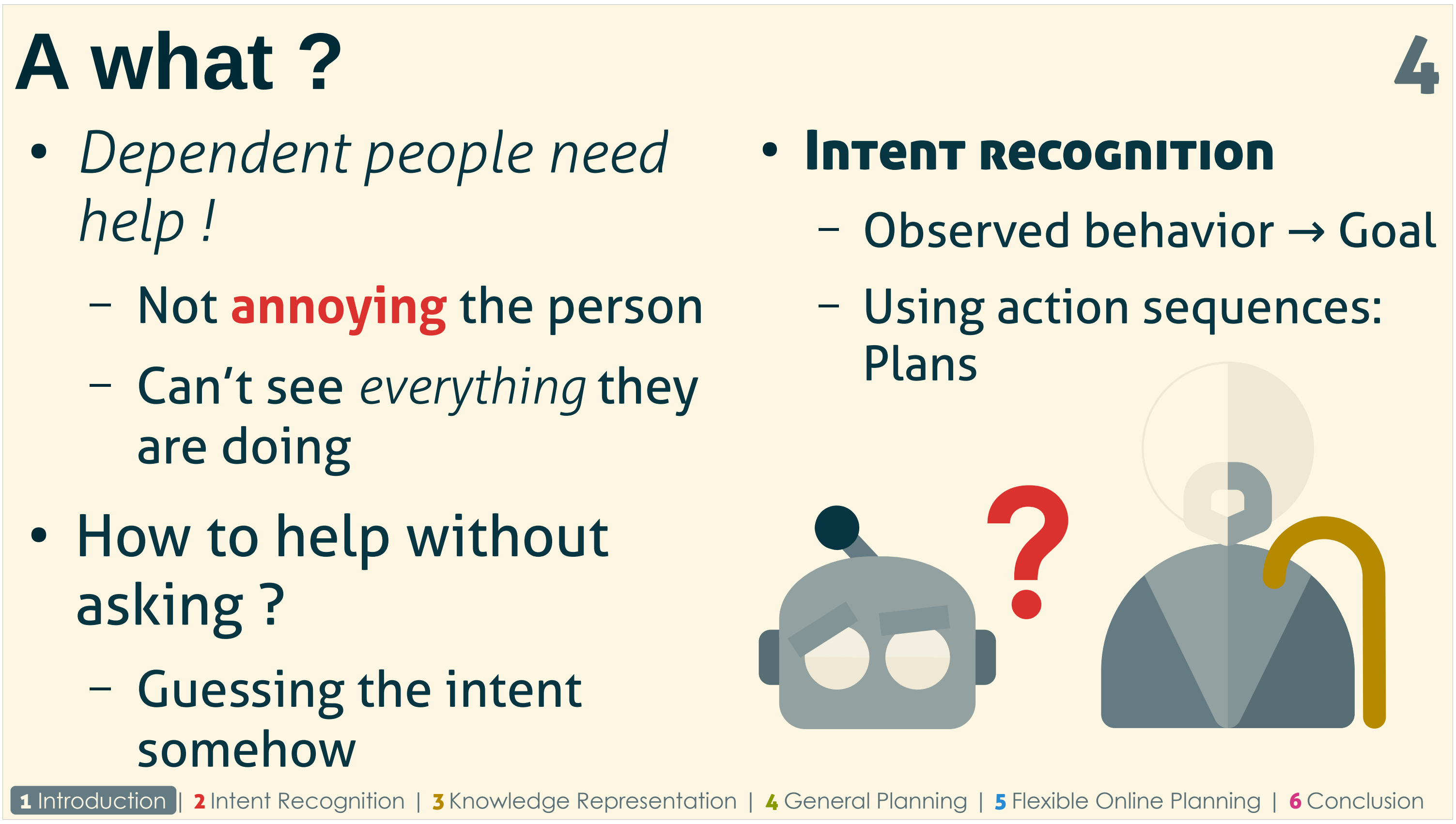
Bonjour à tous et merci d’assister à ma soutenance de thèse intitulée [titre].

Je suis Antoine Gréa, doctorant à l’université Lyon 1 et durant cette présentation je vais vous exposer mes diverses contributions.



1 Introduction

Tout d’abord on va présenter les base du problème



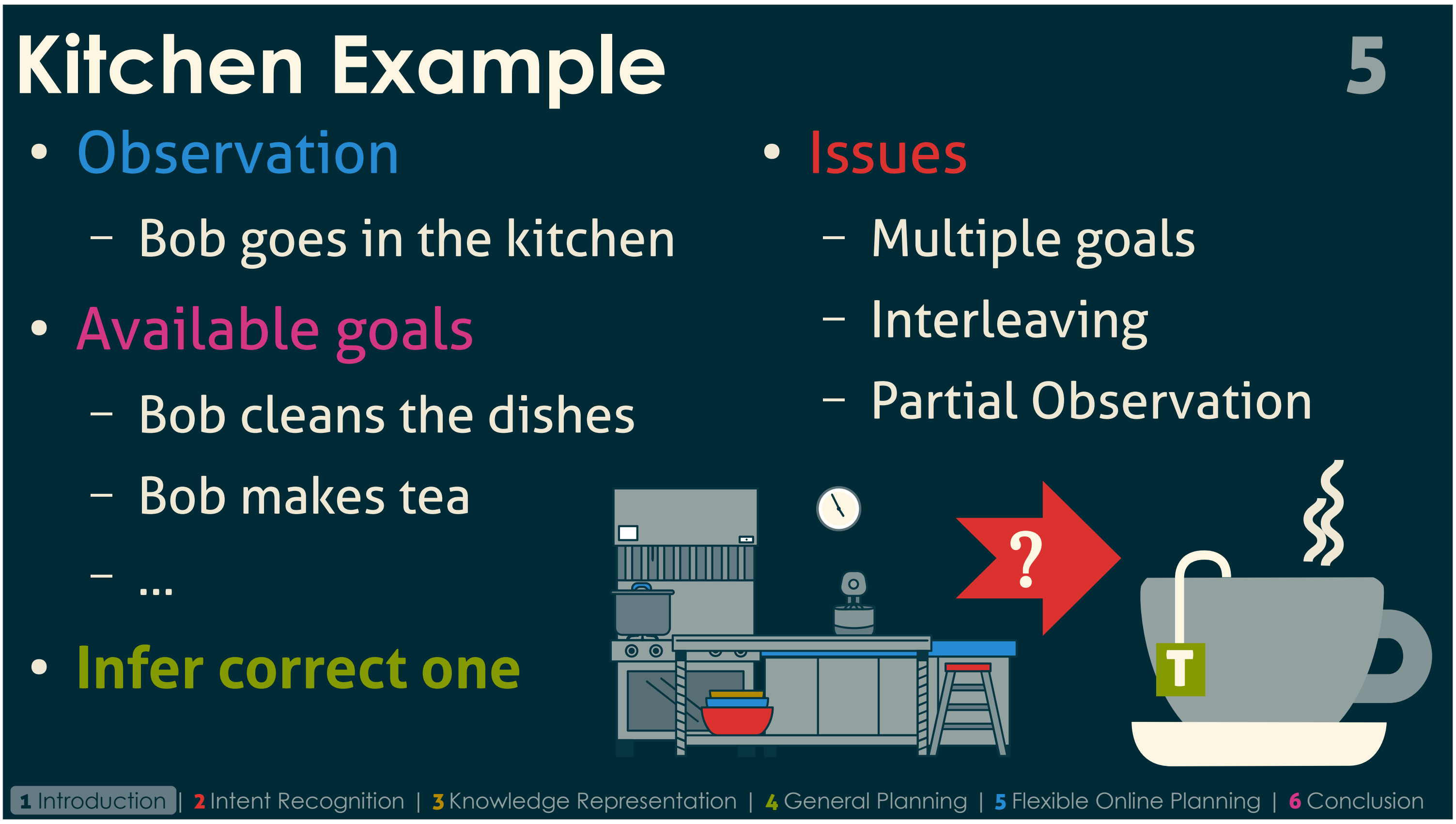
A what ?

- *Dependent people need help !*
  - Not **annoying** the person
  - Can’t see *everything* they are doing
- How to help without asking ?
  - Guessing the intent somehow

- **Intent recognition**
  - Observed behavior → Goal
  - Using action sequences: Plans

La reconnaissance d’intention c’est quoi ?

On ne doit pas déranger la personne et l’observation est toujours partielle.



Kitchen Example

- Observation
  - Bob goes in the kitchen
- Available goals
  - Bob cleans the dishes
  - Bob makes tea
  - …
- Infer correct one

- Issues
  - Multiple goals
  - Interleaving
  - Partial Observation

Voici l’example utilisé tout du long de cette présentation.

Plan

1 Introduction

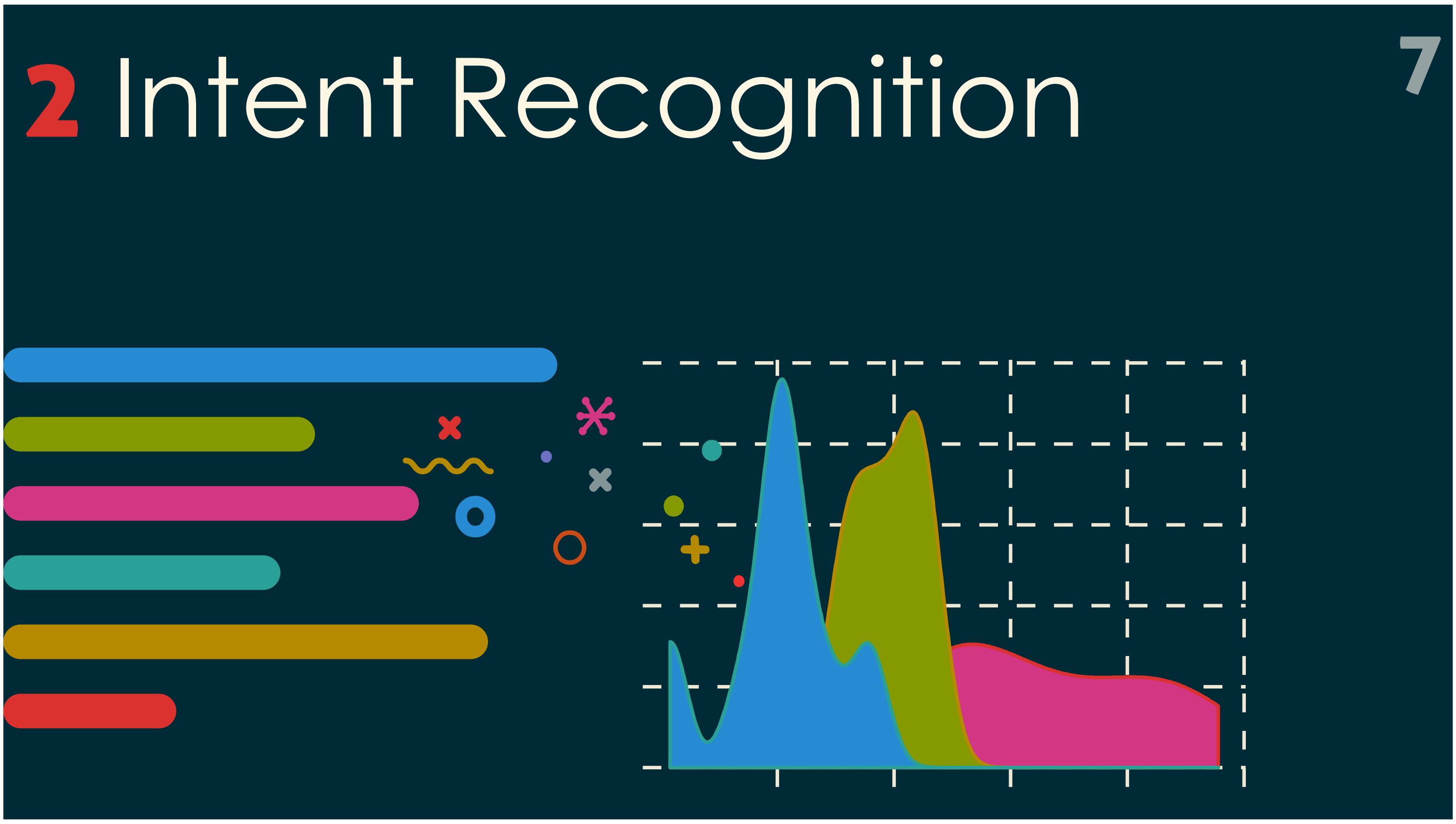
2 Intent Recognition

3 Knowledge Representation

4 General Planning

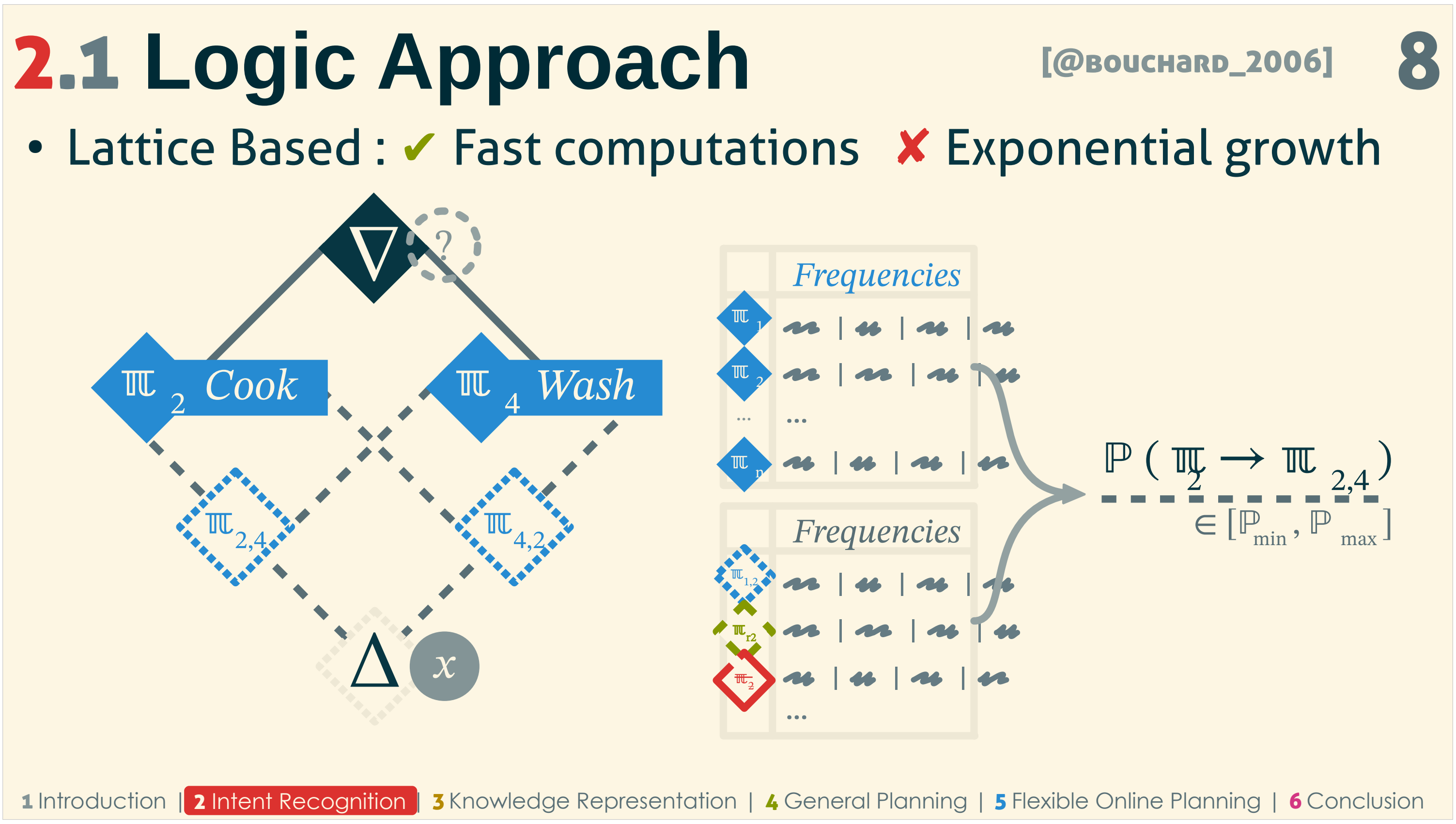
5 Flexible Online Planning

6 Conclusion



2 Intent Recognition

Comment faire de la reconnaissance d’intention



2.1 Logic Approach

[@bouchard_2006]

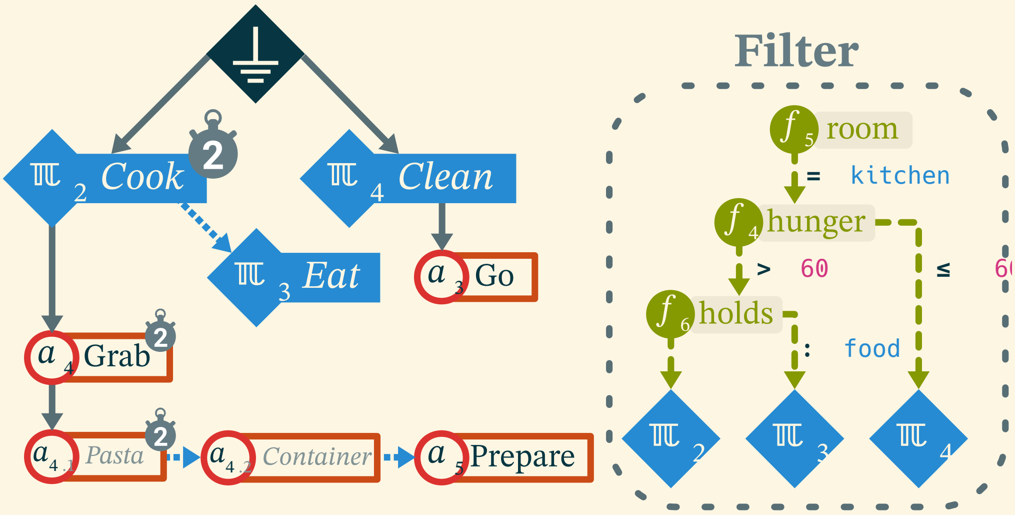
- Lattice Based : ✔ Fast computations ✘ Exponential growth

Approche logique Bouchard
Treilli : objet mathematique ordonné

2.2 Stochastic Approach

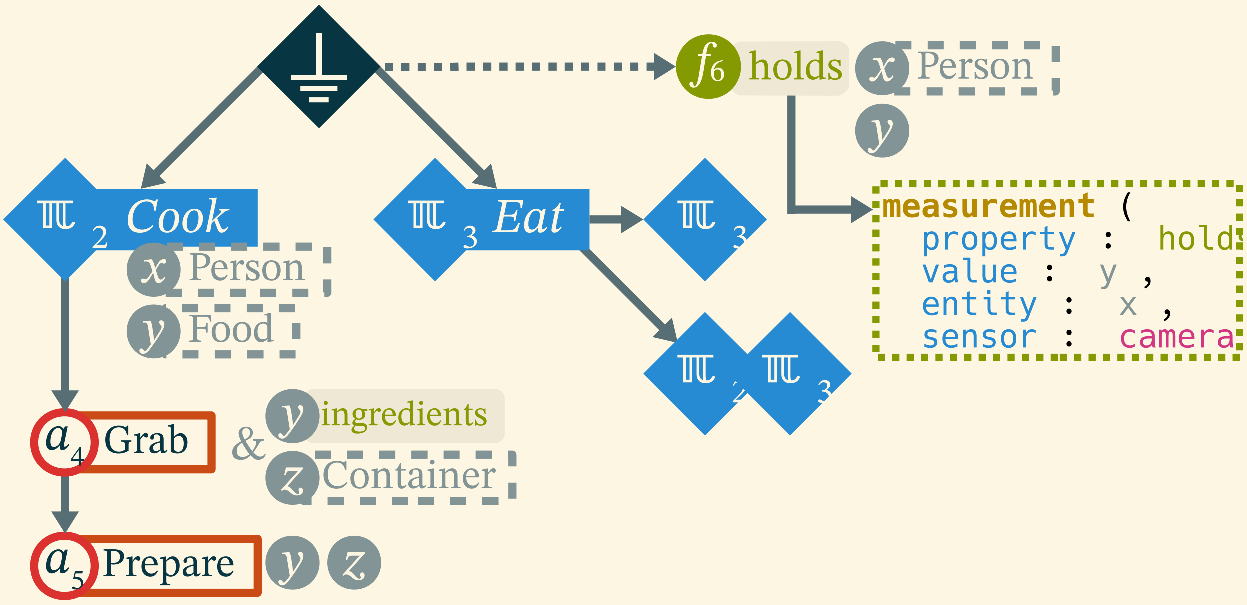
[@avrahami_2006]

- And/Or and decision tree :
  - ✔ Accurate & efficient
  - ✘ Handmade plan library & tree

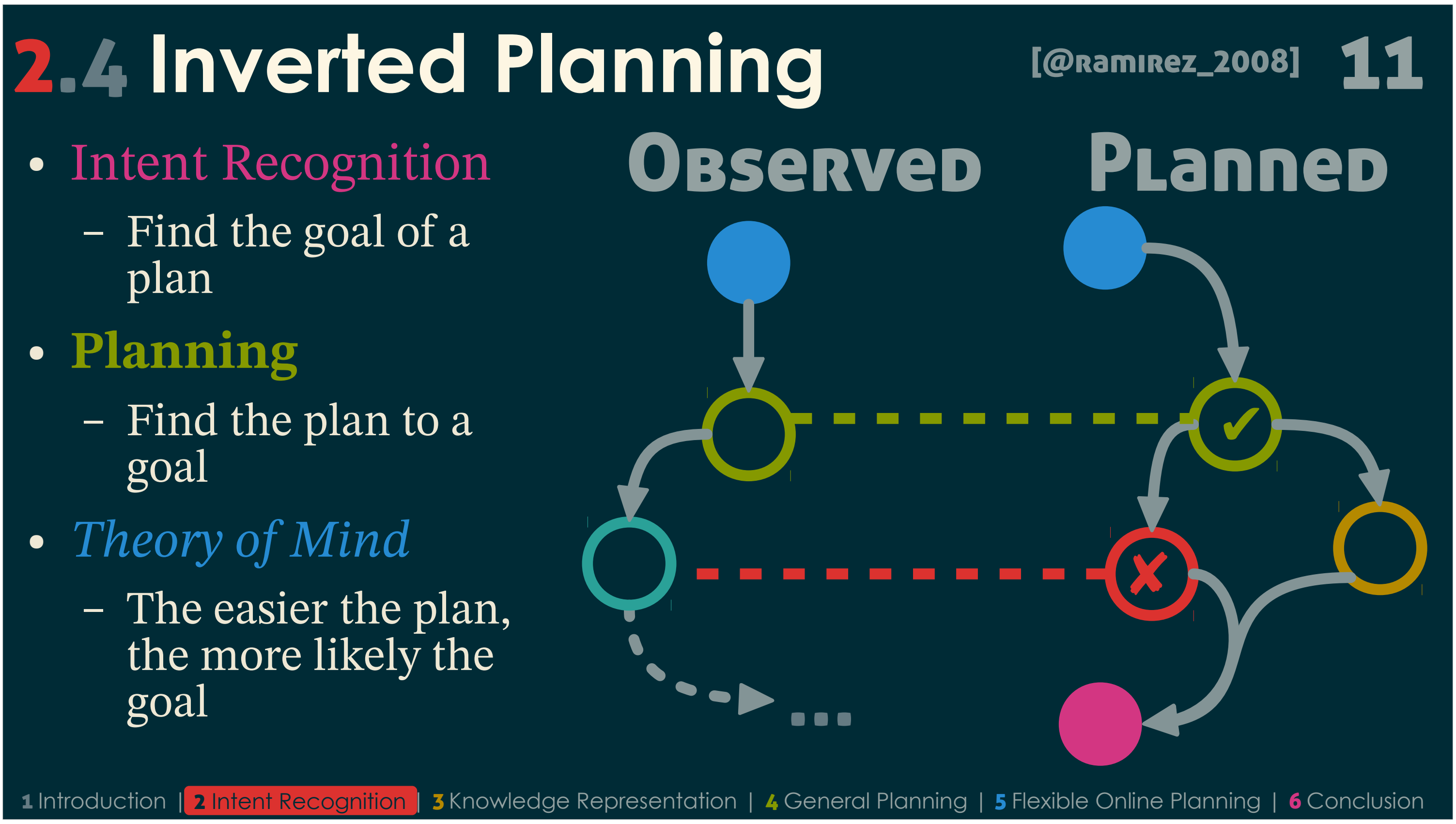


2.3 Grammatical Approach [@vidal_2010]

- Valued Grammar : ✔ Versatile ✘ Slow refresh rate (~40s)



```
measurement (
  property :  holds
  value :  y ,
  entity :  x ,
  sensor :  camera
```



2.4 Inverted Planning

[@ramirez_2008] 11

- Intent Recognition
  - Find the goal of a plan
- **Planning**
  - Find the plan to a goal
- *Theory of Mind*
  - The easier the plan, the more likely the goal

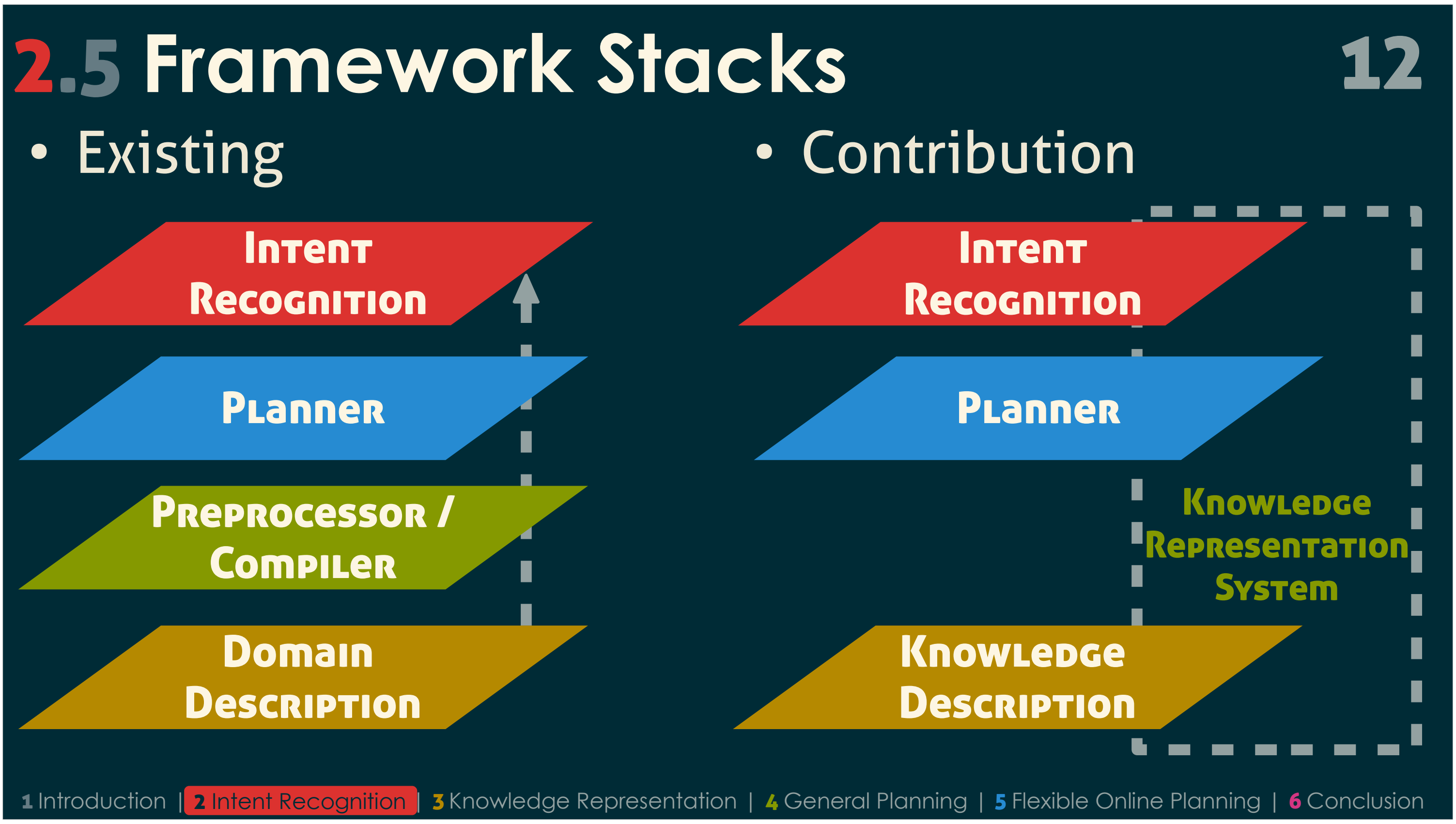
Observed

Planned

Planning: On a le but et on cherche le plan

ToM: On se met à la place de l’autre

On observe un plan en cours d’exécution donc partiel



2.5 Framework Stacks

- Existing
- Contribution

Existing:

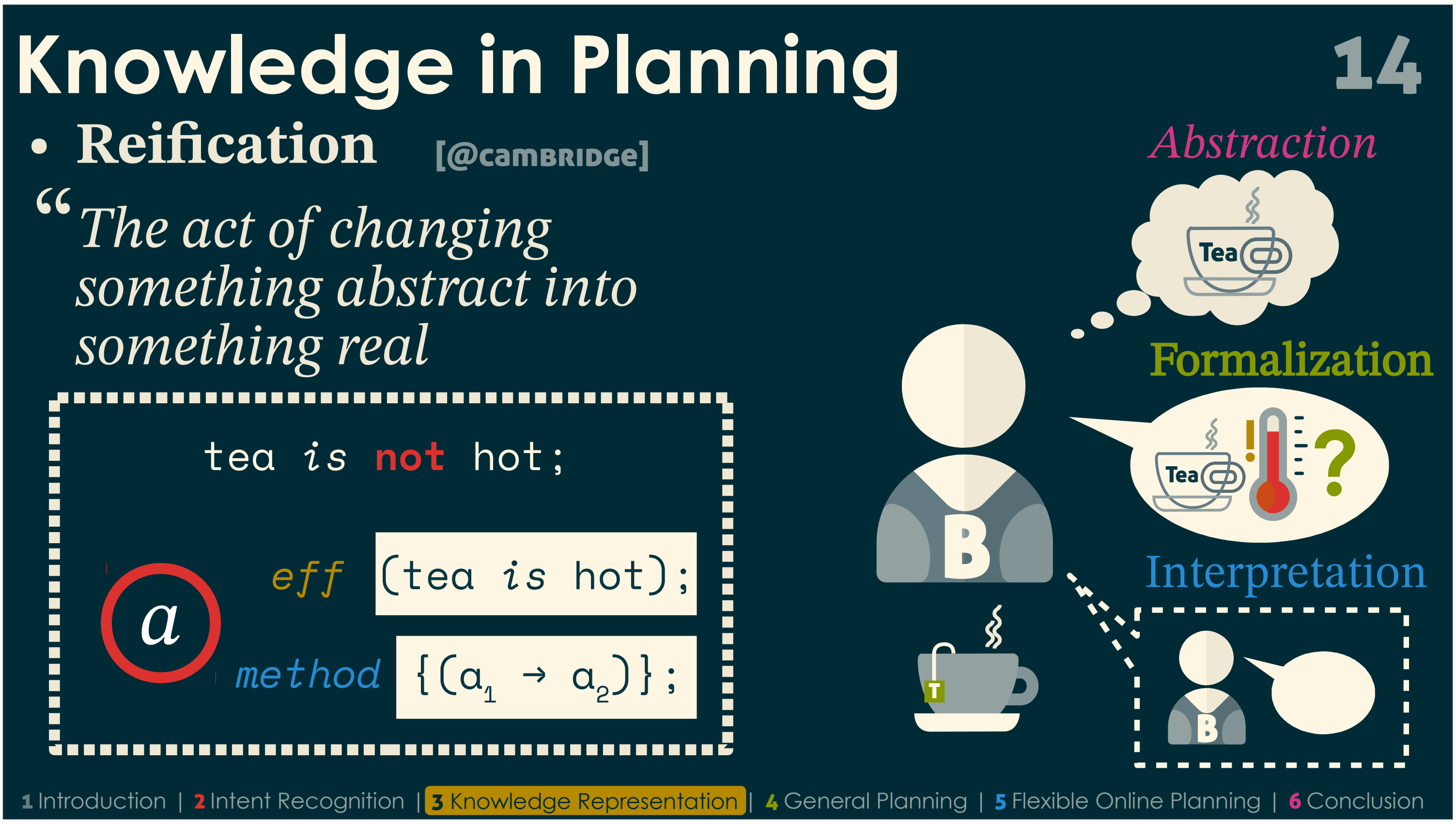
- Intent Recognition
- Planner
- Preprocessor / Compiler
- Domain Description

Contribution:

- Intent Recognition
- Planner
- Knowledge Representation System
- Knowledge Description

Le système de connaissance est utilisé à tout les niveau et c’est le même donc on rends la chose plus flexible

3 Knowledge Representation



Knowledge in Planning

- **Reification** [@cambridge]

> “*The act of changing something abstract into something real*

```
tea is not hot;
eff (tea is hot);
```

method $\{(a_1 \rightarrow a_2)\};$

Abstraction

Formalization

Interpretation

Abstraire c’est réduire une entité à ses characteristiques élémentaire pour plus facilement y référer (perte d’info)

Formalization: Exprimer les connaissances en utilisant un language sous-jacent (language hôte)

Inter : Connaître les relation entre entité et ainsi leur définition (circularité)

Existing Tools

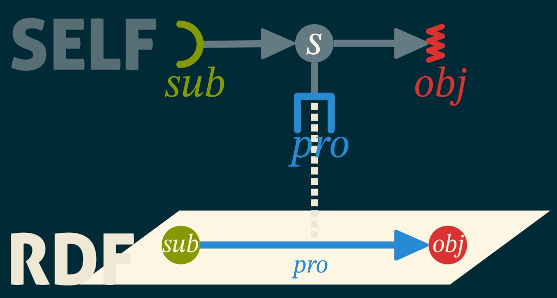
- Ontologies
  - Based on Description Logic

```
<?xml version="1.0"?>

<RDF>
  <Description about="Bob">
    <likes>Tea↙likes>
    <location>Kitchen↙location>
  ↙Description>
↙RDF>
```

- Languages
  - RDF
  - OWL-(Lite, DL, Full)
  - ...
- Issues
  - Reification inefficient
  - Higher order knowledge
  - Flexibility of the structure

SELF

- Self defined
  - Structure = meaning
  - **Ex:** `*x = x;`
- More expressive
- Native reification
  - Express fluents and states in higher order spaces
  - Methods for hierarchical planning



Examples:

```
s = (bob @ kitchen);
a pre s;
a methods
{go(kitchen) → take(cup)};
```

4 General Planning

Classical Planning

- Domain
  - Fluents
    - Formula over objects
  - States
    - Properties of the world
    - Formula over fluents
  - Actions
    - Precondition
    - Effects
- Problem
  - Initial state
  - Goal state
- Plan (solution)
  - Action sequence
  - Order
    - Total
    - Partial

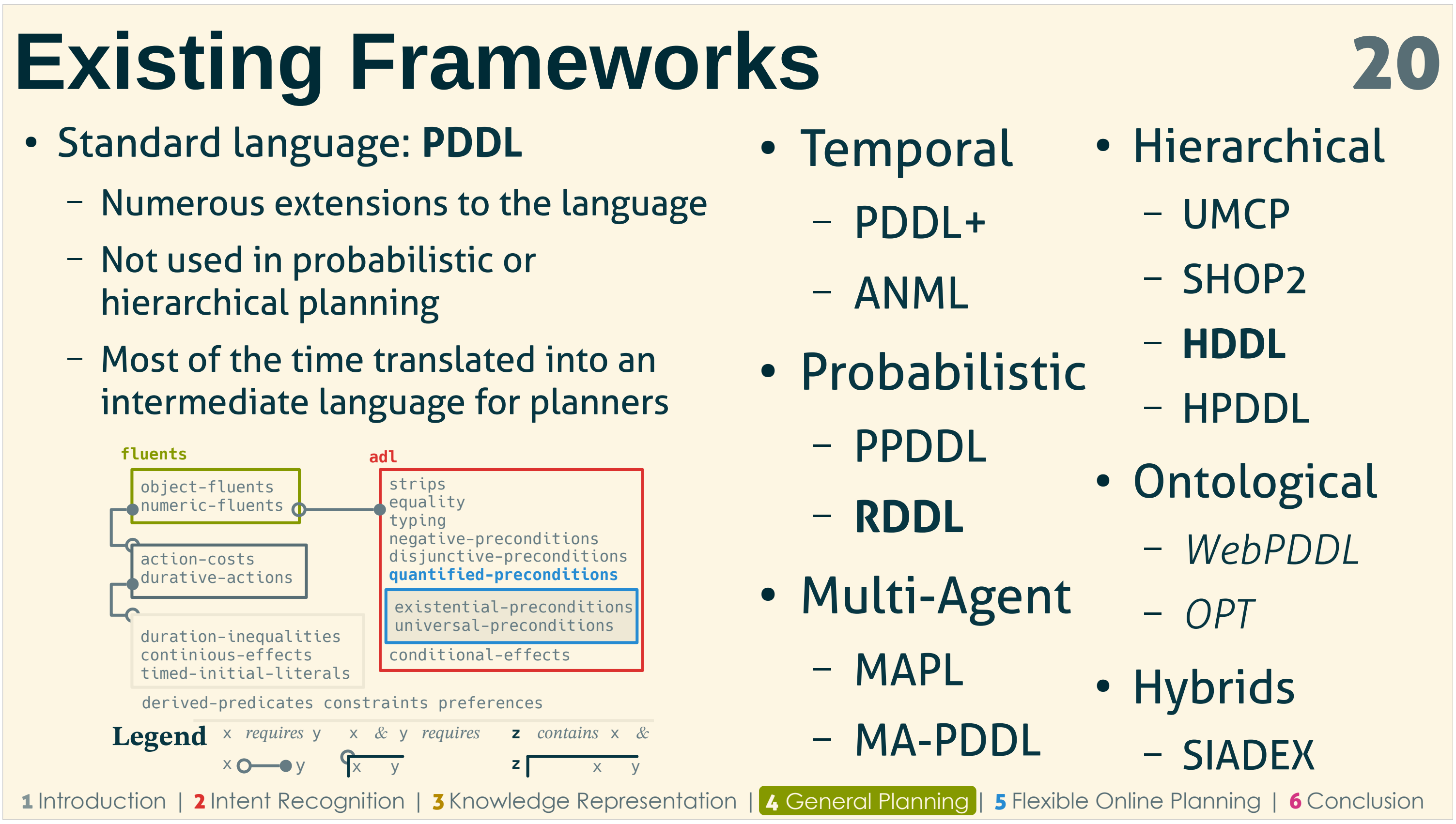
Example

- Fluents
  - thing *taken*
  - *hot* water, tea *ready*
- Actions
  - take, brew, boil, …

T
Sugar
Sugar
T

Initial State

Goal State



Existing Frameworks

- Standard language: **PDDL**
  - Numerous extensions to the language
  - Not used in probabilistic or hierarchical planning
  - Most of the time translated into an intermediate language for planners

fluents

object-fluents
numeric-fluents

action-costs
durative-actions

duration-inequalities
continious-effects
timed-initial-literals

adl

strips
equality
typing
negative-preconditions
disjunctive-preconditions
quantified-preconditions

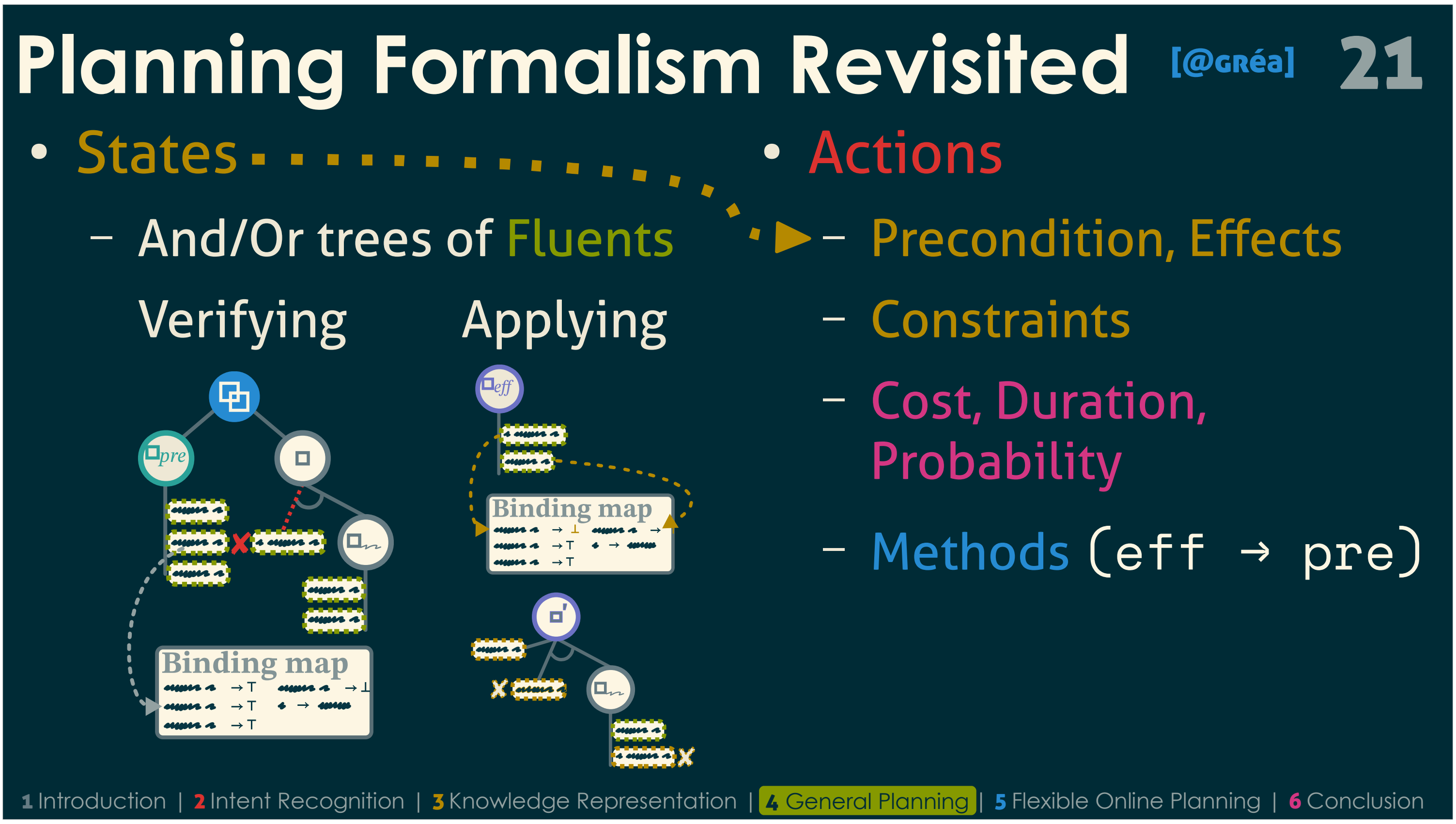
existential-preconditions
universal-preconditions

conditional-effects

derived-predicates constraints preferences

- Temporal
  - PDDL+
  - ANML
- Probabilistic
  - PPDDL
  - **RDDL**
- Multi-Agent
  - MAPL
  - MA-PDDL
- Hierarchical
  - UMCP
  - SHOP2
  - **HDDL**
  - HPDDL
- Ontological
  - *WebPDDL*
  - *OPT*
- Hybrids
  - SIADEX

HDDL pour HTN Track à l’IPC et converti vers SHOP2 parfois.



Planning Formalism Revisited [@Gréa] 21

- States
  - And/Or trees of Fluents
- Actions
  - Precondition, Effects
  - Constraints
  - Cost, Duration, Probability
  - Methods (eff → pre)

Verifying

Applying

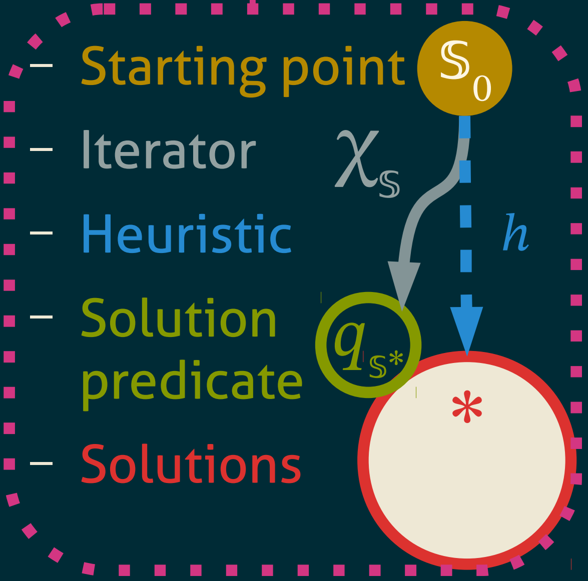
1 Introduction | 2 Intent Recognition | 3 Knowledge Representation | 4 General Planning | 5 Flexible Online Planning | 6 Conclusion

On construit un nouvel arbre avec les deux états et on férifie que la valeur ne soit pas “faux”

Application : On construit la conjonction et on enlève tout ce qui est dans l’état courant qui n’est pas compatible en utilisant la carte des correspondances.

General Planning Framework [@Gréa]

- Search Space
  - Starting point
  - Iterator
  - Heuristic
  - Solution predicate
  - Solutions



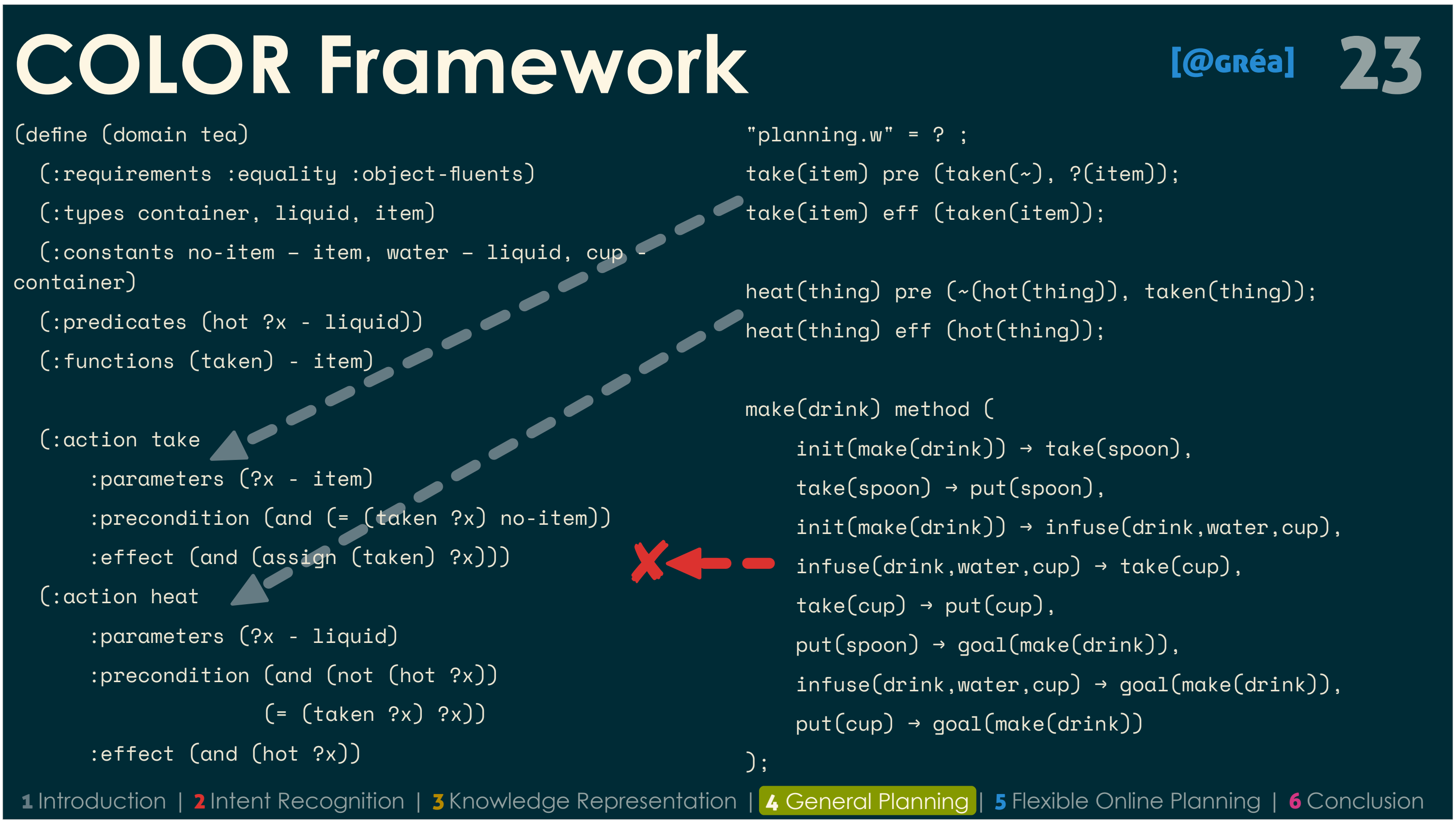
- Instances available for
  - **State-transition**
  - Plan space
  - Case based
  - Probabilistic
  - Hierarchical

$_0 = a^0$ (initial)

$\chi = a \in A$ (actions)

$q_* = (=a^*)$ (goal)

$= \square$ (states)



COLOR Framework

[@Gréa]

```
(define (domain tea)
  (:requirements :equality :object-fluents)
  (:types container, liquid, item)
  (:constants no-item – item, water – liquid, cup –
container)
  (:predicates (hot ?x - liquid))
  (:functions (taken) - item)

  (:action take
      :parameters (?x - item)
      :precondition (and (= (taken ?x) no-item))
      :effect (and (assign (taken) ?x)))
  (:action heat
      :parameters (?x - liquid)
      :precondition (and (not (hot ?x))
                         (= (taken ?x) ?x))
      :effect (and (hot ?x))
```

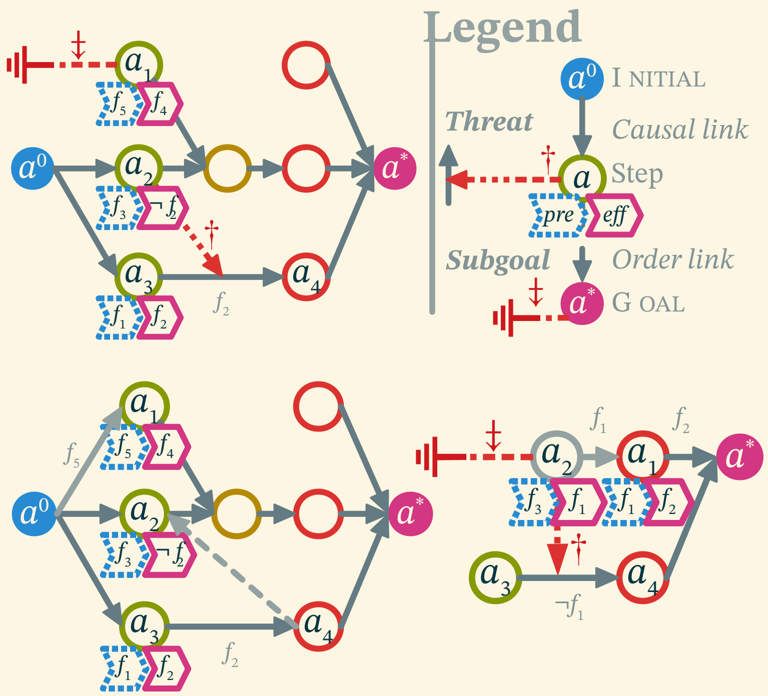
```
"planning.w" = ? ;
take(item) pre (taken(~), ?(item));
take(item) eff (taken(item));

heat(thing) pre (~(hot(thing)), taken(thing));
heat(thing) eff (hot(thing));

make(drink) method (
    init(make(drink)) → take(spoon),
    take(spoon) → put(spoon),
    init(make(drink)) → infuse(drink,water,cup),
    infuse(drink,water,cup) → take(cup),
    take(cup) → put(cup),
    put(spoon) → goal(make(drink)),
    infuse(drink,water,cup) → goal(make(drink)),
    put(cup) → goal(make(drink))
);
```

Pas de Plannification hierarchique en PDDL

5 Flexible Online Planning

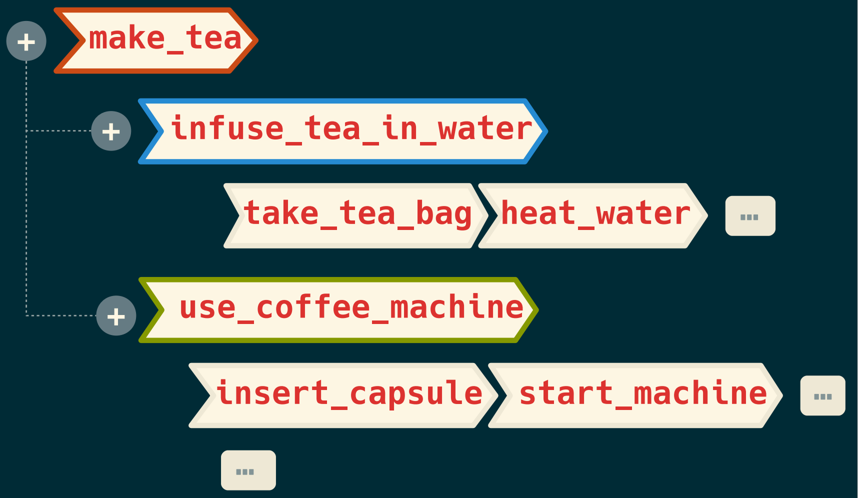
Plan Space Planning

- Exploration by refinements
- Flaws
  - Subgoals
  - Threats
- Resolvers
  - Side effects
- May need backtracking



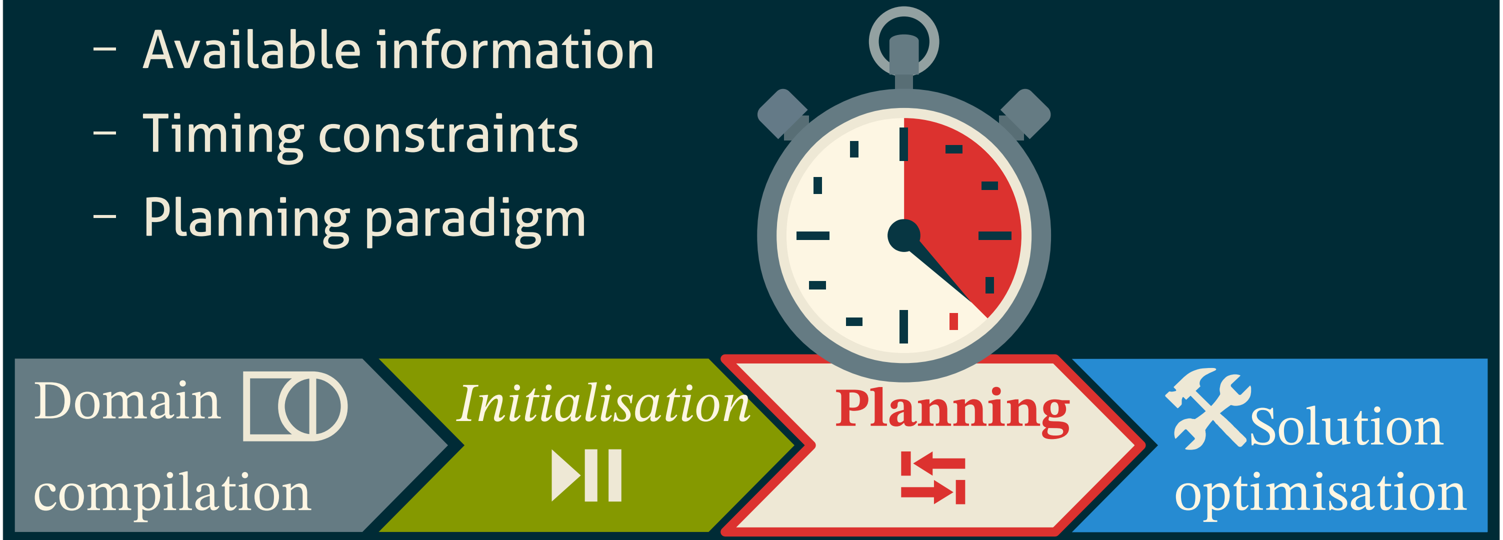
Hierarchical Task Networks

- Based on tasks decomposition
  - Replace task with method
- Lots of different approaches



Planning Phases

- Phases dependent on
  - Available information
  - Timing constraints
  - Planning paradigm



Domain compilation

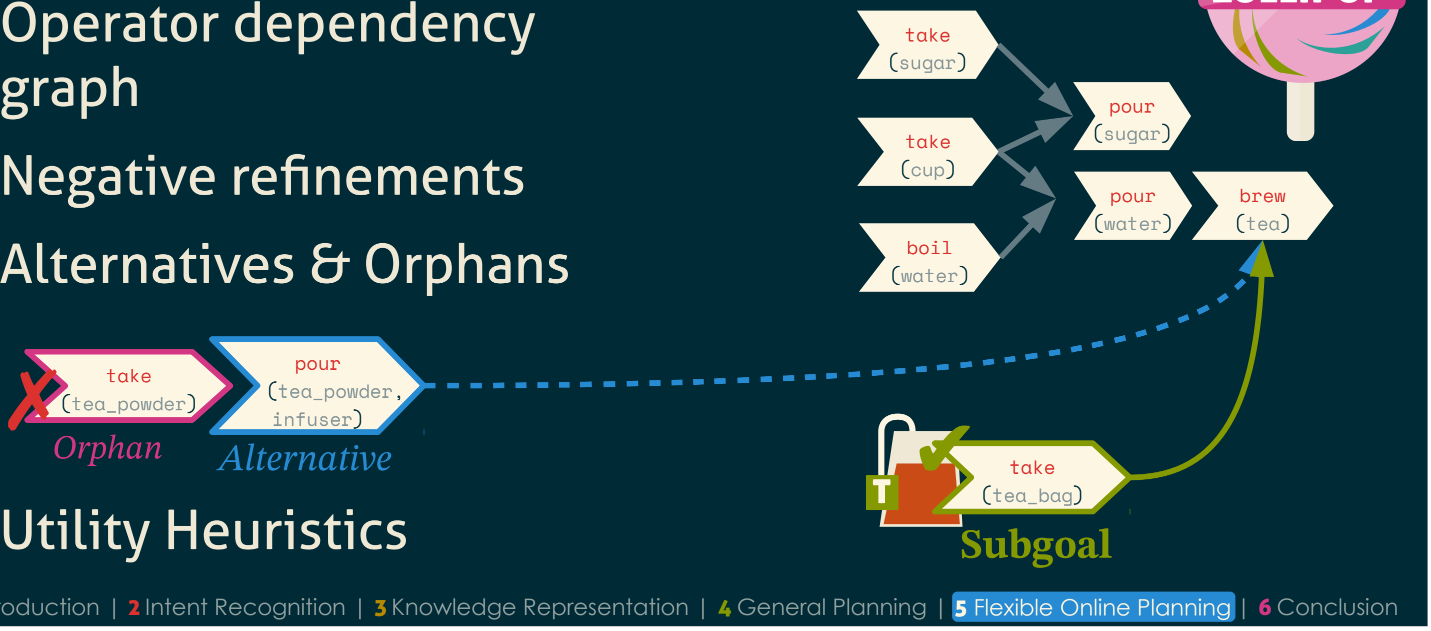
Initialisation

Planning

Solution optimisation

Plan Repair Prototype

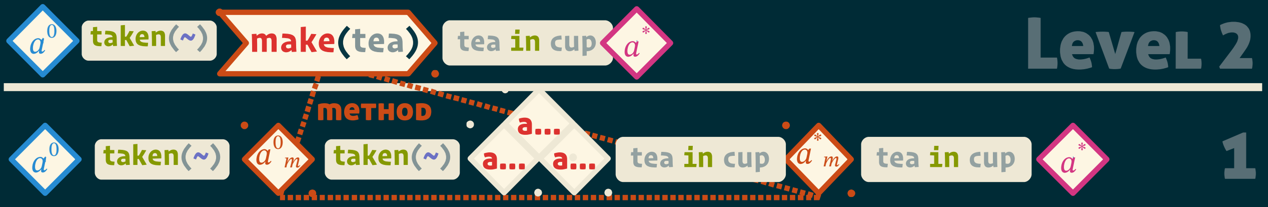
- Partial Order Planner (POP)
- Operator dependency graph
- Negative refinements
- Alternatives & Orphans



- Utility Heuristics

Abstract Planning

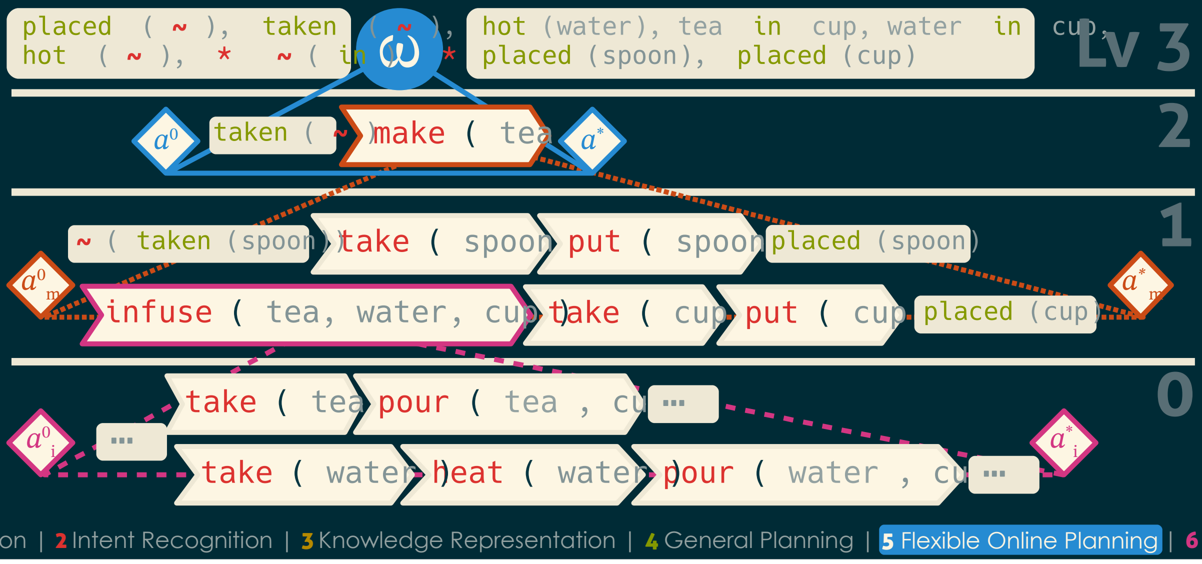
- HTN + POP planning
- Partial Resolution
  - An abstract solution at every level of abstraction
- Search by level
  - Expansion after completion :
- Decomposition flaw
  - Resolver : Decompose one composite action in the plan



HEART

- Low priority for expansion
- Each level is a plan (abstract solution)

- Change of level
  - Propagation of atomic actions
  - Expansion of Composite Equities



placed (~), taken (~), hot (~), * ~ (in) *

ω

hot (water), tea in cup, water in cup, placed (spoon), placed (cup)

Lv 3

a^0 taken (~) make (tea a^*

2

~ (taken (spoon)) take (spoon put (spoon placed (spoon)

a^0_m infuse (tea, water, cup take (cup put (cup placed (cup) a^*_m

1

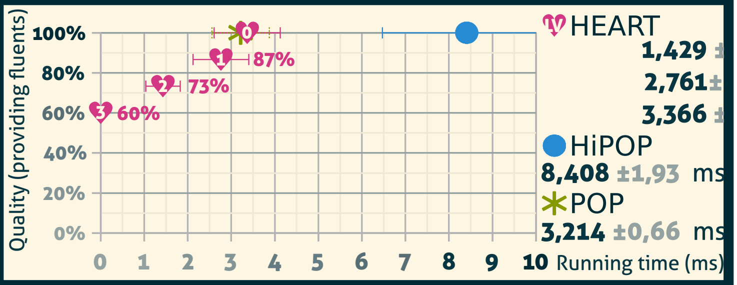
take (tea pour (tea , cu ⋯

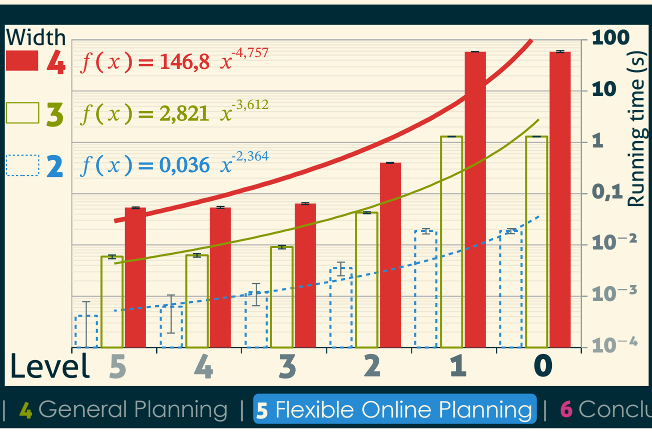
a^0_i ⋯ take (water heat (water pour (water , cu ⋯ a^*_i

0

Results

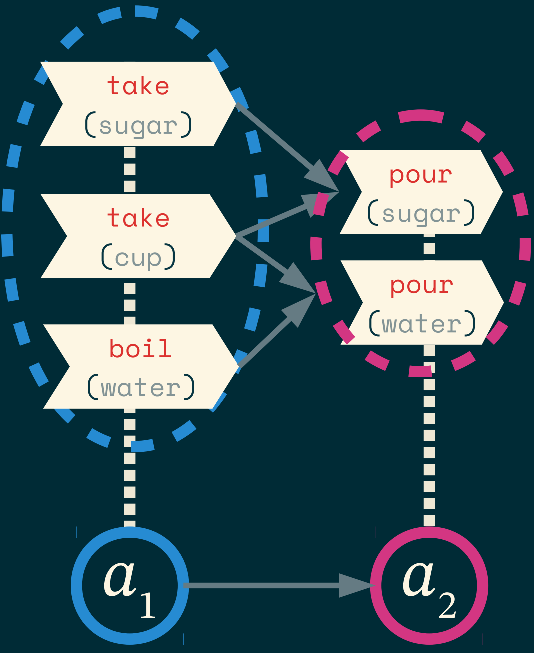
- 60% of the fluents before planning
- Exponentially faster at high abstraction levels
- Faster than HiPOP on some problems
- Common problems solved in milliseconds!





Toward Intent Recognition

- Linearized parallel actions using graph quotient
- Abstraction makes it easier (smaller plans)
- Backward chaining is inefficient



6 Conclusion

Contributions & Results

- SELF: A knowledge description language defined by structure
- COLOR: A general framework for planning with its formalization
- LOLLIPOP: A plan repair planner for online planning
- HEART: A flexible approach to real-time planning for abstract planning

Perspectives

- SELF Improvement
  - Improve the instantiation workflow
  - Parameterize flexibility performances
- Planning Colorized
  - Conversion tool from PDDL
  - Make a clean implementatio n for community use
- Fixing Planning Domains
  - Allow HEART to discover new HTN methods (macro-action learning)