CZECH
TECHNICAL
UNIVERSITY
IN PRAGUE

Belief-Desire-Intention (BDI) Architecture

MICHAL JAKOB
Agent Technology Center,
Dept. of Computer Science and Engineering,
FEE, Czech Technical University

AE4M36MAS Autumn 2013 - Lect. 2

Where are we?

Agent architectures (inc. BDI architecture)

Logics for MAS

Non-cooperative game theory

Coalition game theory

Mechanism design

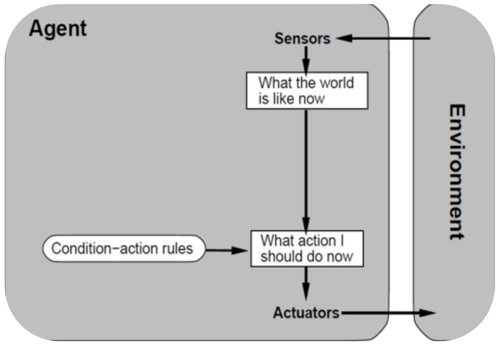
Auctions

Social choice

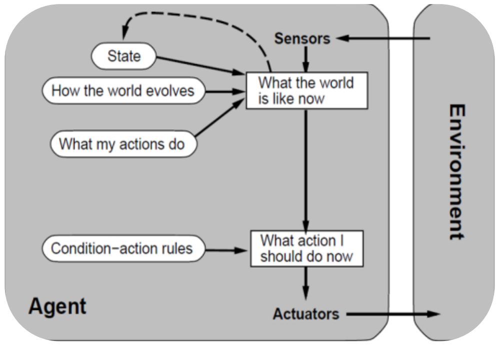
Distributed constraint reasoning (satisfaction and optimization)

Intro and Motivation

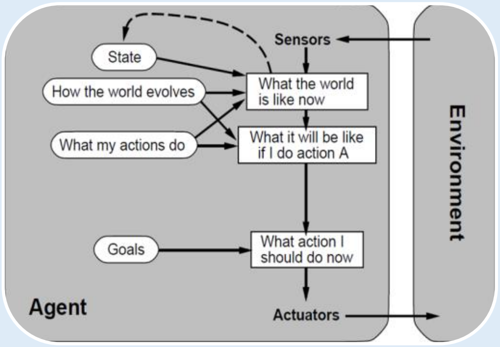
Basic Agent Architectures



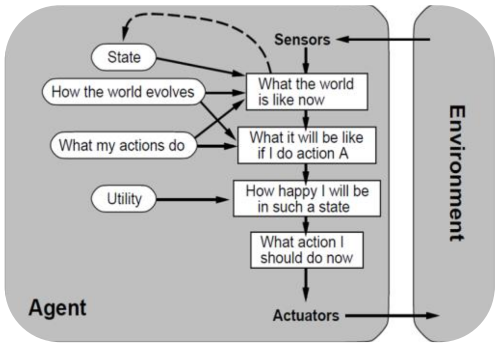
Reflex agent



Model-based agent

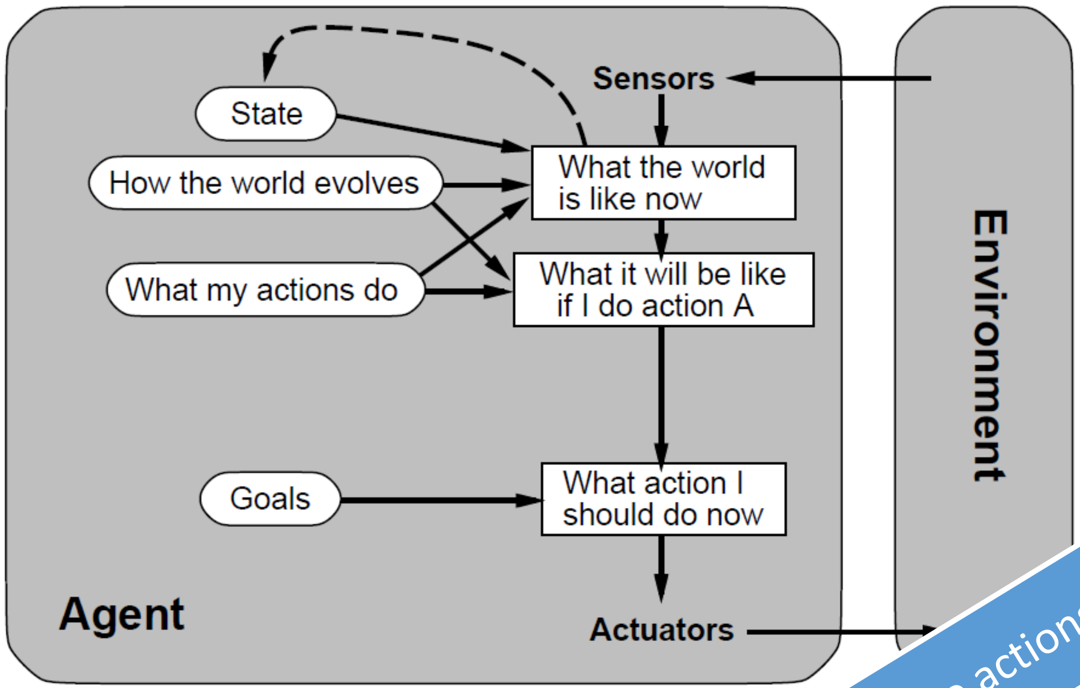

State
Sensors
How the world evolves
What the world is like now
What my actions do
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

Goal-based agent


State
Sensors
How the world evolves
What the world is like now
What my actions do
What it will be like if I do action A
Utility
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

Utility-based agent

Goal-based agents


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

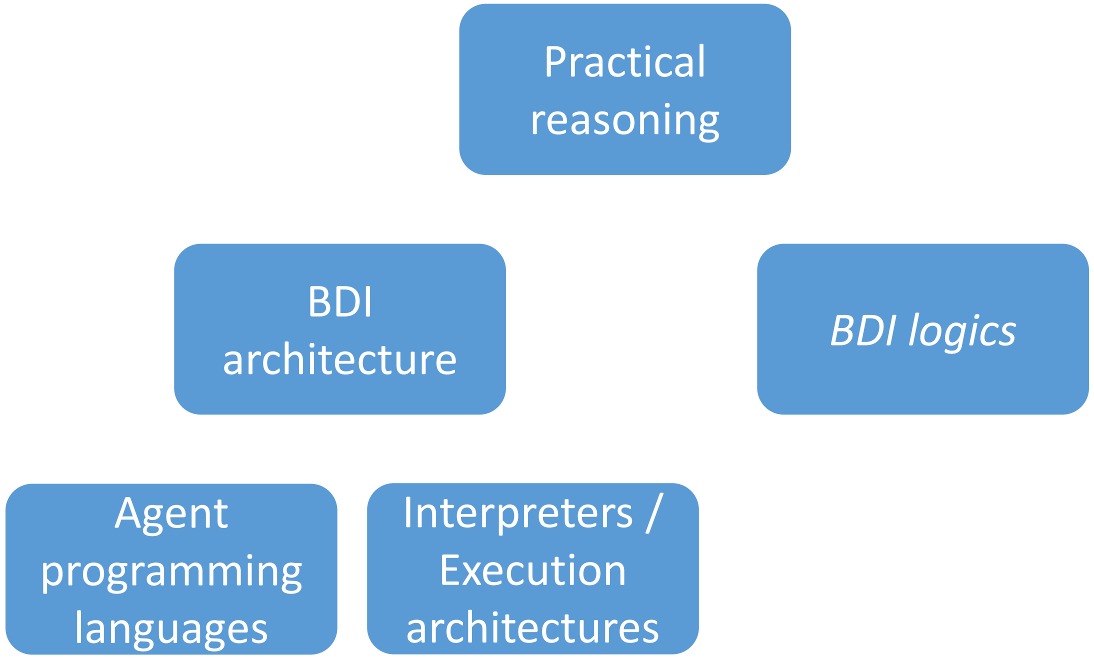
How to go from goals to actions effectively?

Big Picture

philosophical foundations

analysis and design

implementation


Practical reasoning
BDI architecture
BDI logics
Agent programming languages
Interpreters / Execution architectures

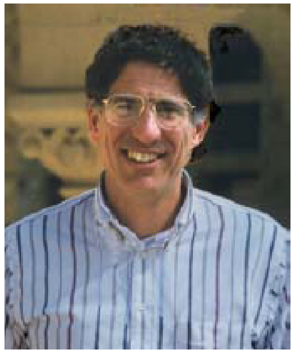
Practical Reasoning

CONCPTUALIZING RATIONAL ACTION

Practical Reasoning

- Practical reasoning is reasoning directed towards actions — the process of figuring out what to do.
- Principles of practical reasoning applied to agents largely derive from work of philosopher Michael Bratman (1990):

"Practical reasoning is a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes."



- Distinguish practical reasoning from theoretical reasoning.

Theoretical vs Practical Reasoning

"In theory, there is no difference between theory and practice. But, in practice, there is." – Jan L. A. van de Snepscheut

1. Theoretical reasoning is reasoning directed towards beliefs — concerned with deciding what to believe.
   - Tries to assess the way things are.
   - Process by which you change your beliefs and expectations;.
   - Example: you believe q if you believe p and you believe that if p then q.
2. Practical reasoning is reasoning directed towards actions — concerned with deciding what to do.
   - Decides how the world should be and what individuals should do.
   - Process by which you change your choices, plans, and intentions.
   - Example: you go to class, if you must go to class.

The Components of Practical Reasoning

Human practical reasoning consists of two activities:

strategic

1. Deliberation: deciding what state of affairs we want to achieve.
   - considering preferences, choosing goals, etc.;
   - balancing alternatives (decision-theory);
   - the outputs of deliberation are intentions;
   - interface between deliberation and means-end reasoning.

tactical

2. Means-ends reasoning: deciding how to achieve these states of affairs:
   - thinking about suitable actions, resources and how to "organize" activity;
   - building courses of action (planning);
   - the outputs of means-ends reasoning are plans.

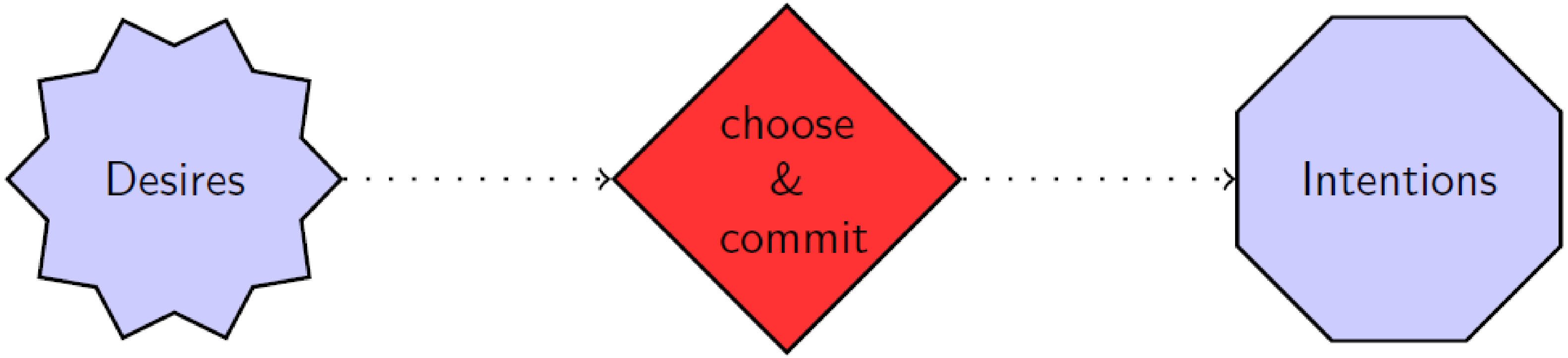
Fact: agents are resource-bounded & world is dynamic!

The key: To combine **deliberation** & **means-ends reasoning** appropriately.

Deliberation

How does an agent deliberate?

1. Begin by trying to understand what the options available to you are:
   - options available are desires.
2. Choose between them, and commit to some:
   - chosen options are then intentions.



Desires

- Desires describe the states of affairs that are considered for achievement, i.e., basic preferences of the agent.
- Desires are much weaker than intentions; not directly related to activity:

> *"My desire to play basketball this afternoon is merely a potential influence of my conduct this afternoon. It must vie with my other relevant desires [...] before it is settled what I will do. In contrast, once I intend to play basketball this afternoon, the matter is settled: I normally need not continue to weigh the pros and cons. When the afternoon arrives, I will normally just proceed to execute my intentions."*
> *(Bratman 1990)*

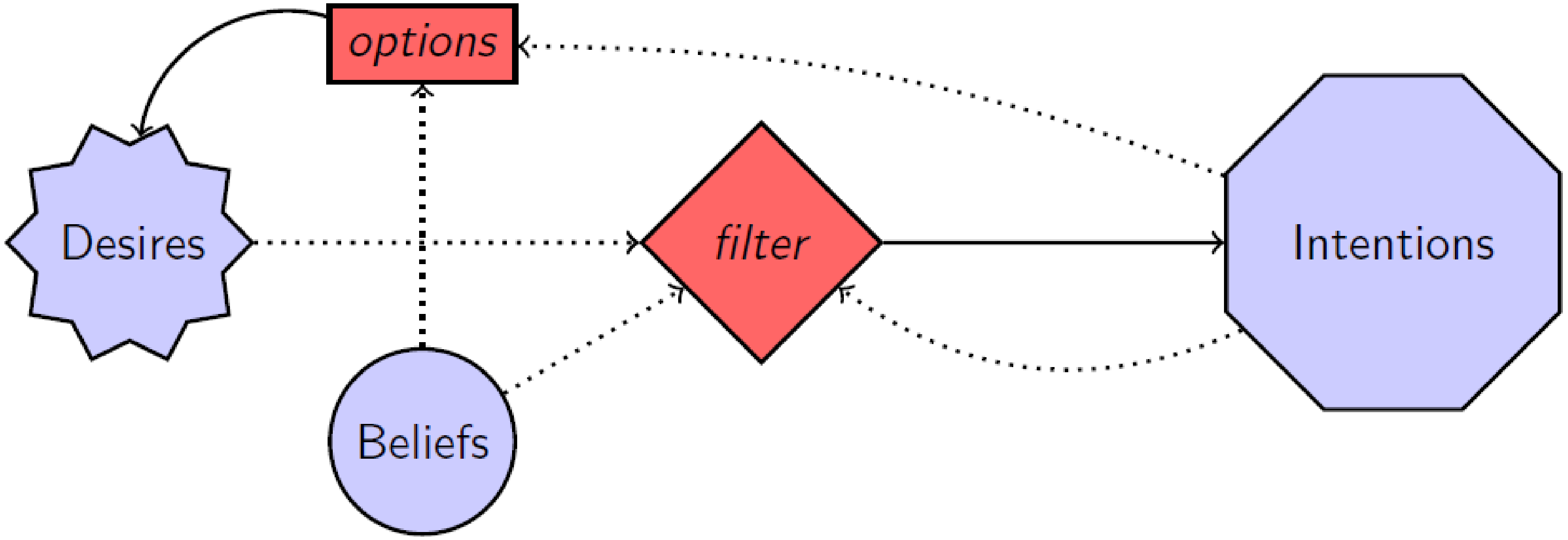
Intentions

- In ordinary speech: intentions refer to actions or to states of mind;
  - here we consider the latter!
  - E.g., I may adopt/have the intention to be an academic.
- Focus on future-directed intentions i.e. pro-attitudes leading to actions.
  - Intentions are about the (desired) future.
- We make reasonable attempts to fulfill intentions once we form them, but they may change if circumstances do.
  - Behavior arises to fulfill intentions.
  - Intentions affect action choice.

Functional Components of Deliberation

Option Generation agent generates a set of possible alternatives; via a function, *options*, which takes the agent's current beliefs and intentions, and from them determines a set of options/desires.

Filtering in which the agent chooses between competing alternatives, and commits to achieving them. In order to select between competing options, an agent uses a filter function.



Properties of Intentions

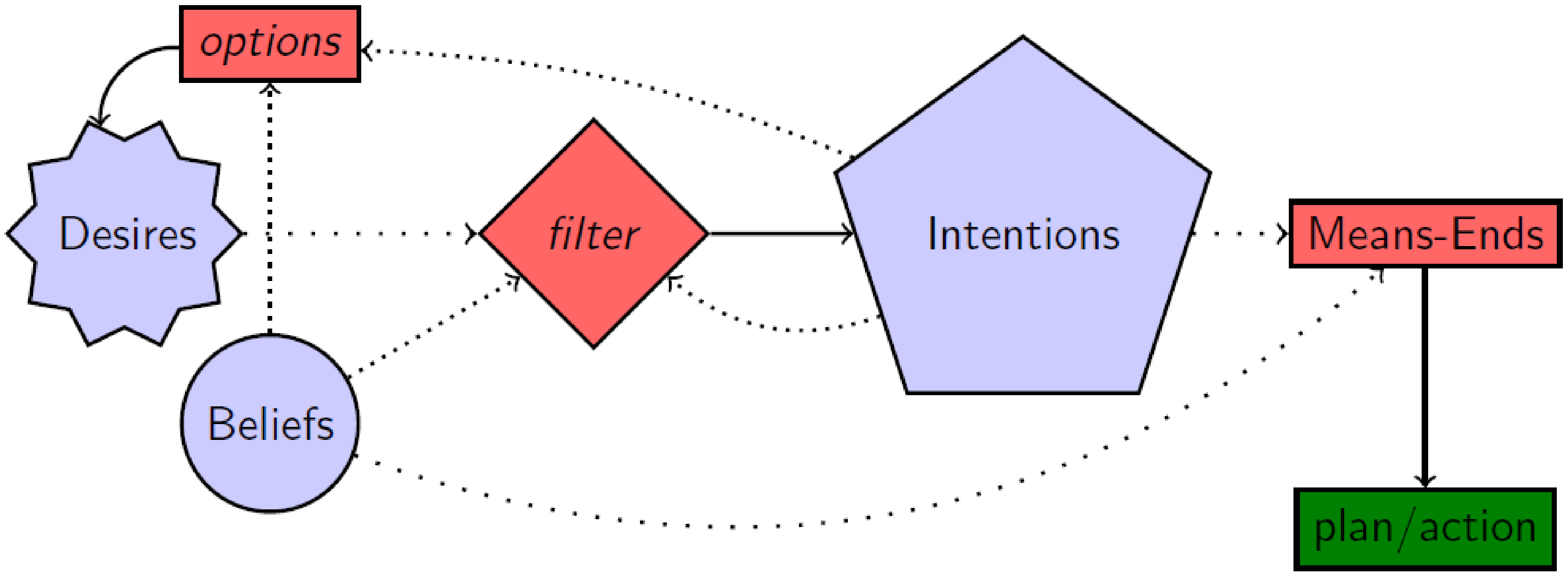
1. Intentions drive means-end reasoning.
2. Intentions constrain future deliberation (i.e., provide a “filter”).
3. Intentions persist.
4. Intentions influence beliefs concerning future practical reasoning.
5. Agents believe their intentions are possible.
6. Agents do not believe they will not bring about their intentions.
7. Under certain circumstances, agents believe they will bring about their intentions.
8. Agents need not intend all the expected side effects of their intentions.

Plans

Human practical reasoning consists of two activities:

1. Deliberation: deciding what to do. Forms intentions.
2. Means-ends reasoning: deciding how to do it. Forms plans. Forms plans.

Intentions drive means-ends reasoning: *If I adopt an intention, I will attempt to achieve it, this affects action choice.*



Commitments

We may think that deliberation and planning are sufficient to achieve desired behavior, unfortunately things are more complex...

Dynamic environment

After filter function, agent makes a commitment to chosen option:

- Commitment: *an agreement or pledge to do something in the future*;
- ∴ it implies temporal persistence.

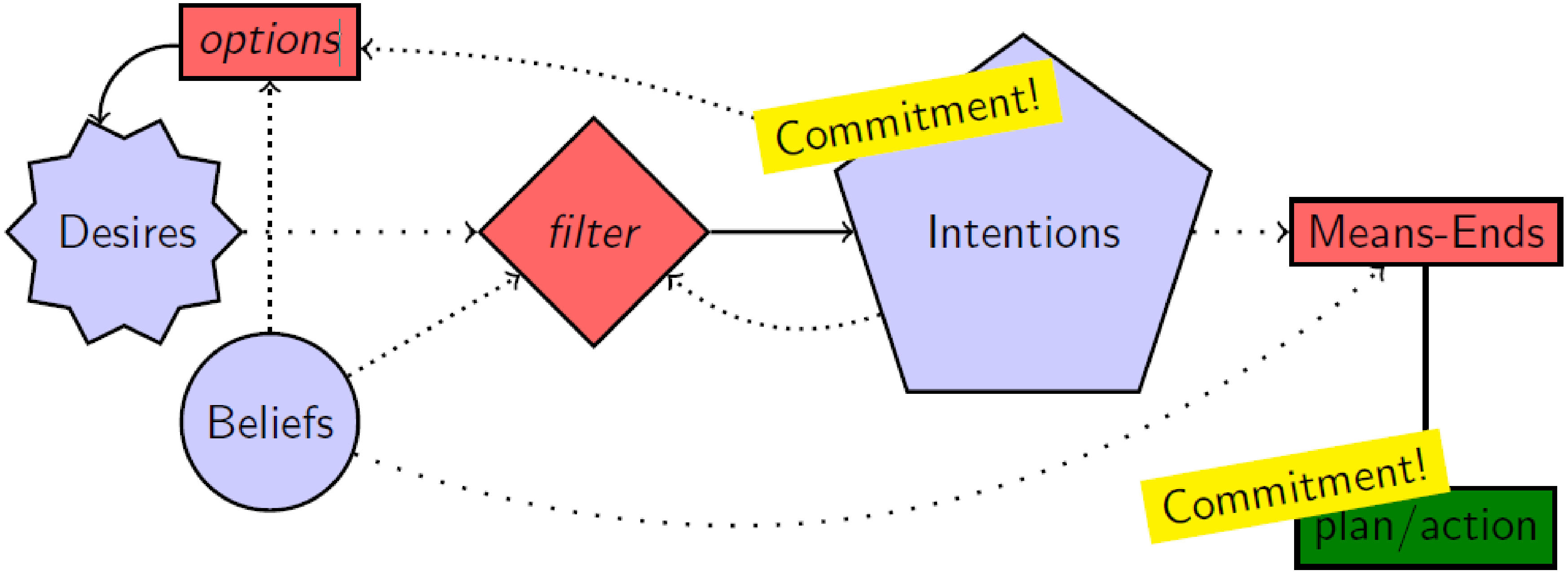
Questions:

1. how long should an intention persist?
2. what is the commitment on?

Commitments to Ends and Means

An agent has commitment both to ends (intentions), and means (plans).

- I am committed to meet/see my friend John this week (an intention);
- I am committed to drop-by John's place on Thursday afternoon (a mean).



Degrees of Commitments

Rao and Georgeff (1991) described the following commitment strategies:

Blind/Fanatical commitment A blindly committed agent will continue to maintain an intention until it believes the intention has actually been achieved.

Single-minded commitment A single-minded agent will continue to maintain an intention until it believes that either the intention has been achieved, or else that it is no longer possible to achieve the intention.

Open-minded commitment An open-minded agent to remains committed to its intentions until it succeeds or the goal is dropped.

Optimality of Intention Reconsideration (IR) Strategy?

- IR strategy is optimal if it would have changed intentions had he/she deliberated again (this assumes IR itself is cheap...)

Situation	IR?	Changed intentions?	Would have changed?	Optimal?
1	NO	–	NO	YES
2	NO	–	YES	NO
3	YES	NO	–	NO
4	YES	YES	–	YES

In situation (1), the agent did not choose to deliberate, and thus, did not change intentions. Still, if it had deliberated, it would not have changed.

- IR was not worth it! :-)

In situation (2), the agent did not choose to deliberate, but if it had done so, it would have changed intentions.

- IR must have been done! :-(

In situation (3), the agent chose to deliberate, but did not change intentions.

- IR was a waste of time! :-(

In situation (4), the agent chose to deliberate, and did change intentions.

- IR was worth it! :-)

When is an IR Strategy Optimal?

Kinny and Georgeff experimentally investigated effectiveness of intention reconsideration strategies:

Bold agents never pause to reconsider intentions.

Cautious agents stop to reconsider after every action.

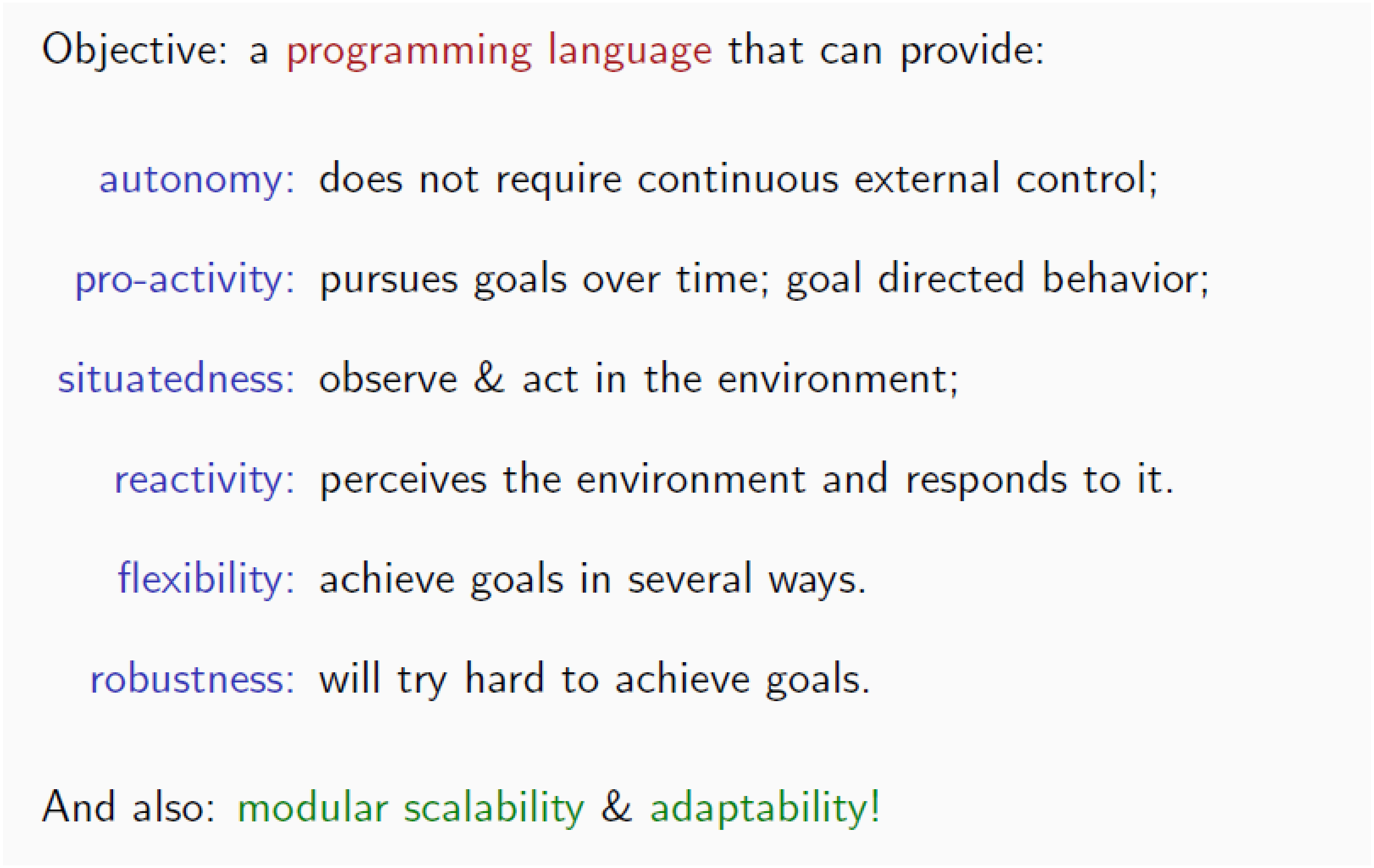
Dynamism in the environment is represented by the rate of world γ change:

- If γ is low (i.e., the environment does not change quickly), then bold agents do well compared to cautious ones.
  - cautious ones waste time reconsidering their commitments while bold agents are busy working towards—and achieving—their intentions.
- If γ is high (i.e., the environment changes frequently), then cautious agents tend to outperform bold agents.
  - cautious agents are able to recognize when intentions are doomed, and also to take advantage new opportunities when they arise.

BDI Architecture

OPERATIONALIZING PRACTICAL REASONING

BDI Programming



Objective: a programming language that can provide:

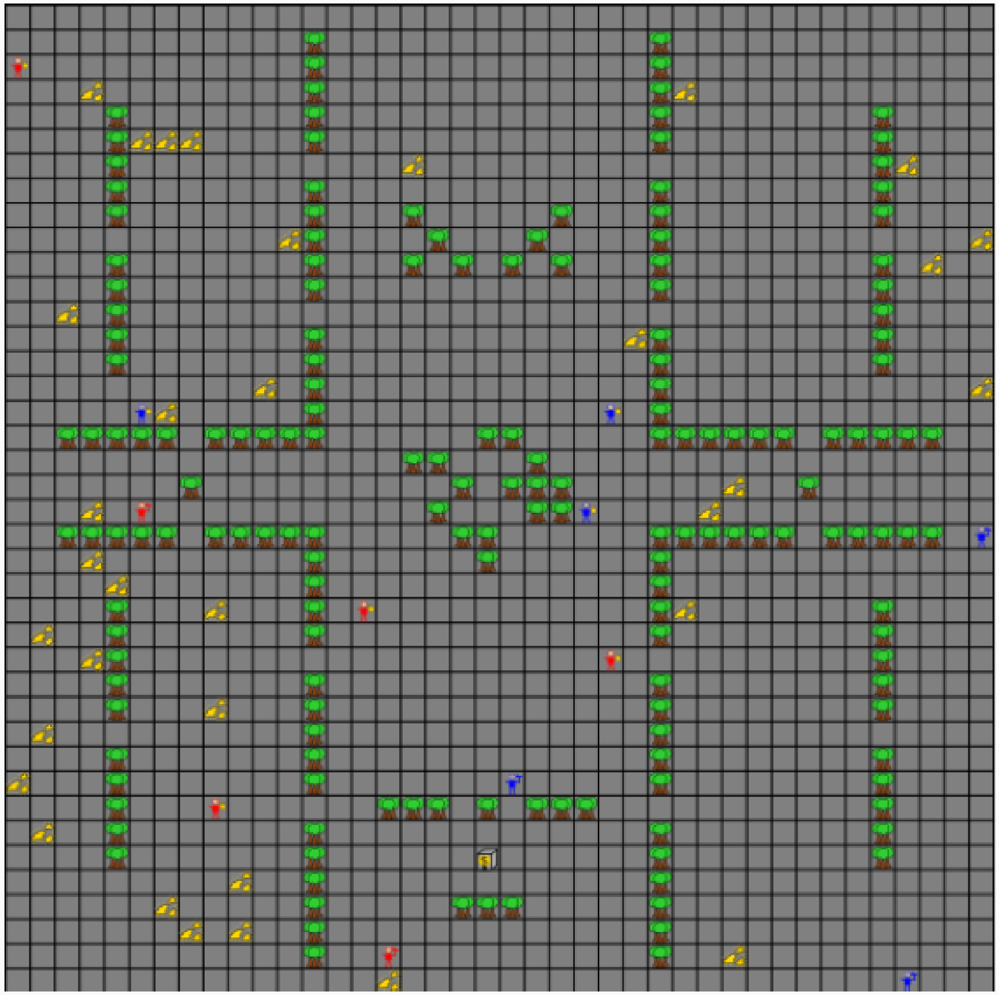
autonomy:	does not require continuous external control;
pro-activity:	pursues goals over time; goal directed behavior;
situatedness:	observe & act in the environment;
reactivity:	perceives the environment and responds to it.
flexibility:	achieve goals in several ways.
robustness:	will try hard to achieve goals.

And also: modular scalability & adaptability!

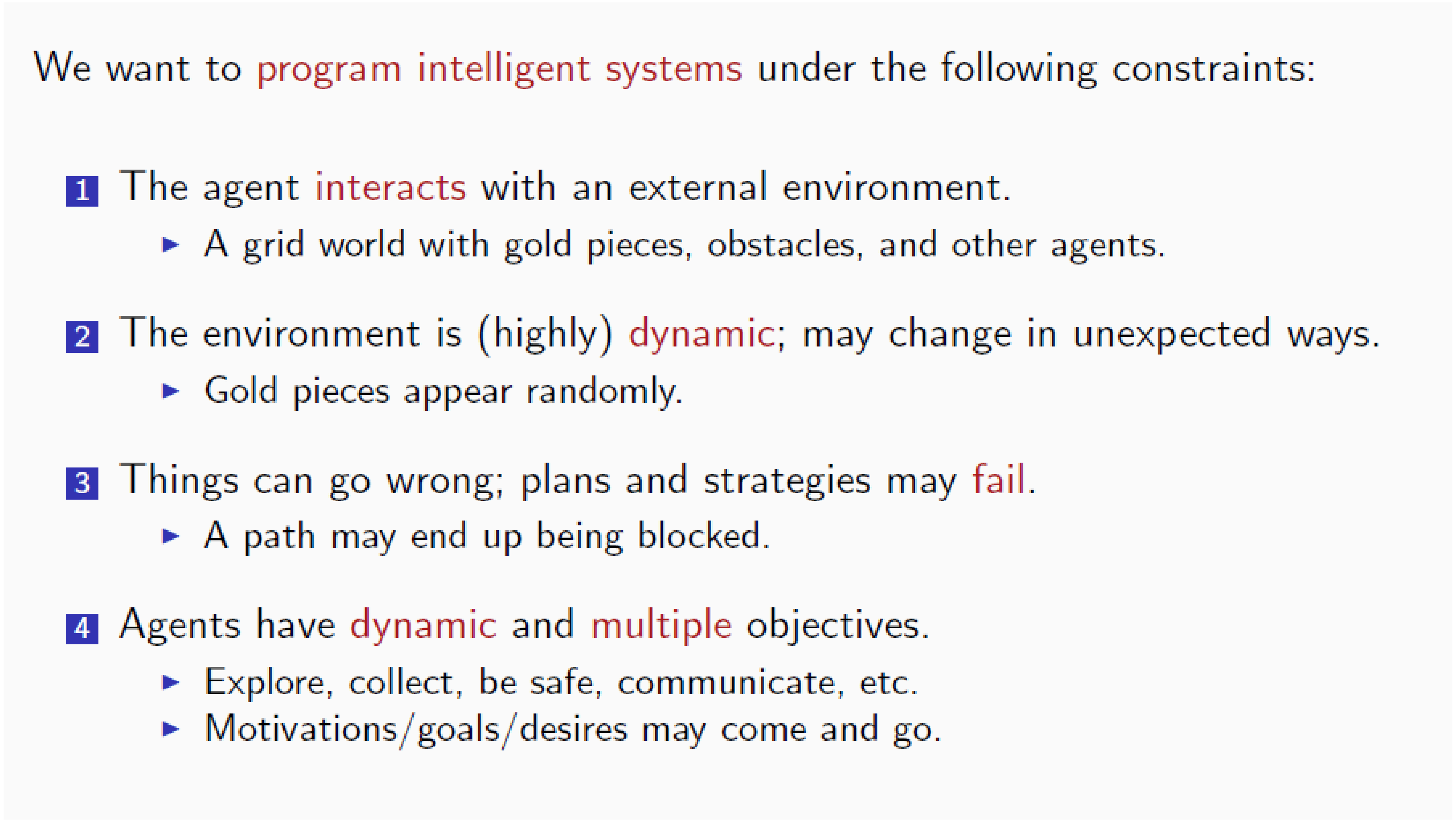
An Example: Gold Mining Game

2 teams competing to collect and drop gold in the depot

- dynamic
- complex
- unknown information
- failing actions
- failing sensors
- multi-agents



Some Constraints / Requirements



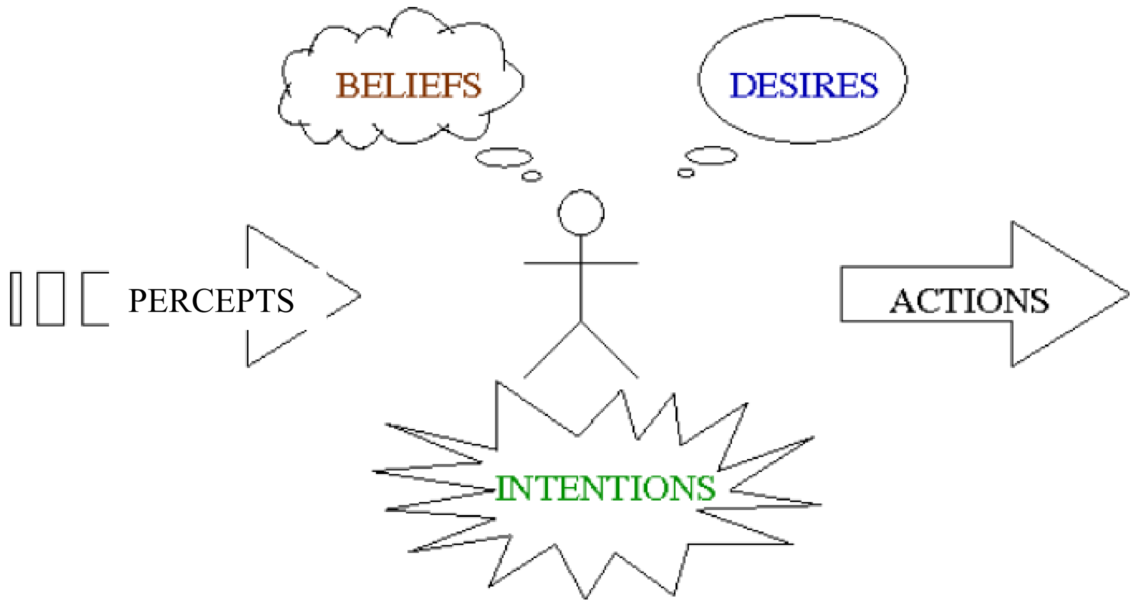
We want to program intelligent systems under the following constraints:

1. The agent interacts with an external environment.
   - A grid world with gold pieces, obstacles, and other agents.
2. The environment is (highly) dynamic; may change in unexpected ways.
   - Gold pieces appear randomly.
3. Things can go wrong; plans and strategies may fail.
   - A path may end up being blocked.
4. Agents have dynamic and multiple objectives.
   - Explore, collect, be safe, communicate, etc.
   - Motivations/goals/desires may come and go.

Belief-Desire-Intention (BDI) Model of Agency

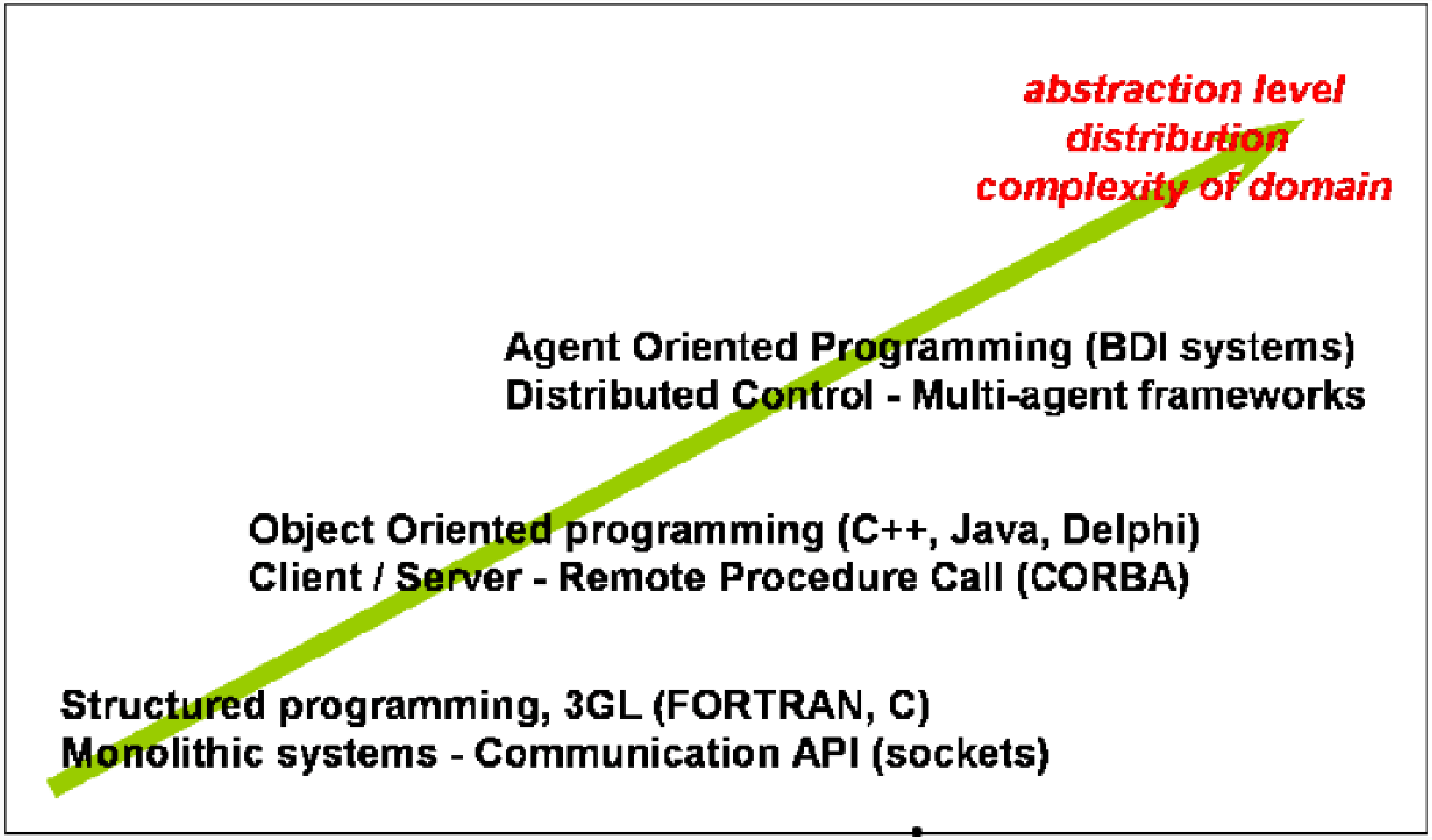
A model of agency with roots in practical reasoning and intentional system

∴ a set of concepts for thinking about and building agents.



Behavior arises due to the agent committing to some of its desires, and selecting actions that achieve its intentions given its beliefs.

Technology Development


abstraction level
distribution
complexity of domain
Agent Oriented Programming (BDI systems)
Distributed Control - Multi-agent frameworks
Object Oriented programming (C++, Java, Delphi)
Client / Server - Remote Procedure Call (CORBA)
Structured programming, 3GL (FORTRAN, C)
Monolithic systems - Communication API (sockets)

Building Agents: BDI Agent-oriented Programming

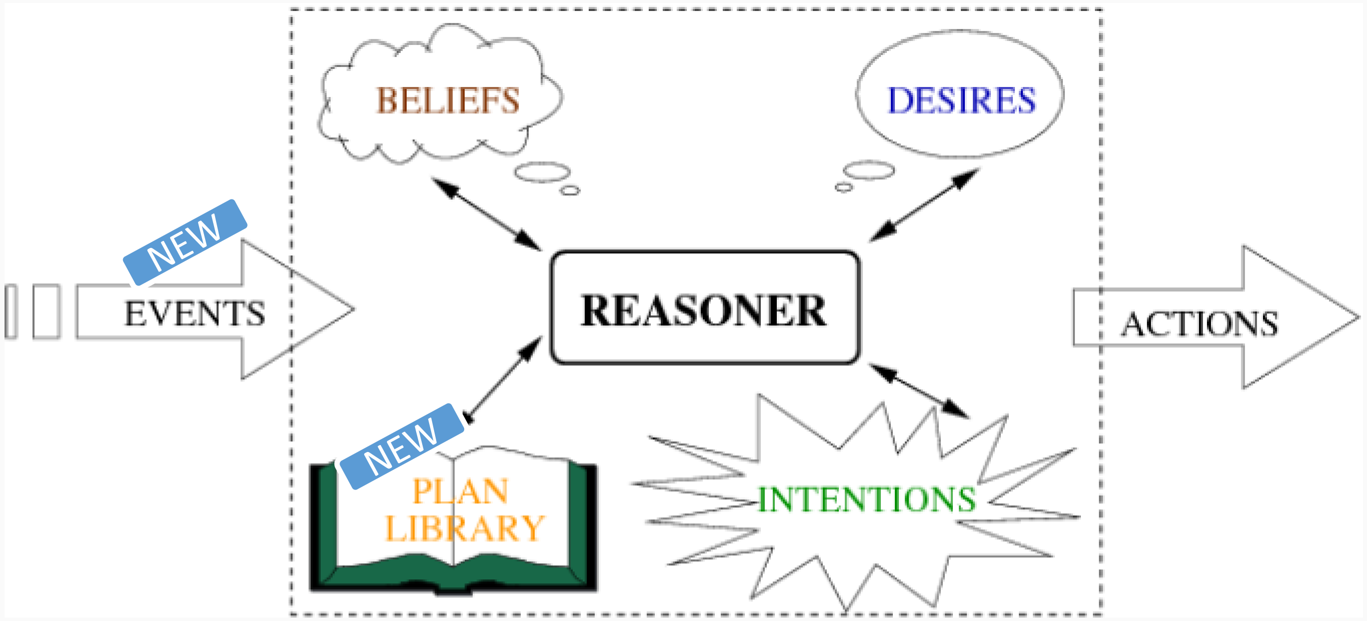
A new programming model and architecture to simplify the construction of todays large complex systems situated in dynamic environments:

1. View a system as composed of autonomous interacting entities (agents) which pursue their own goals and act in a rational manner.
2. Internal state and decision process of agents is modelled in an intuitive manner following the notion of mental attitudes.
3. Goal orientation: instead of directly requesting the agents to perform certain actions, the developer can define more abstract goals for the agents.
   - provides a certain degree of flexibility on how to achieve the goals.

Can be seen as a "successor" of object-oriented programming

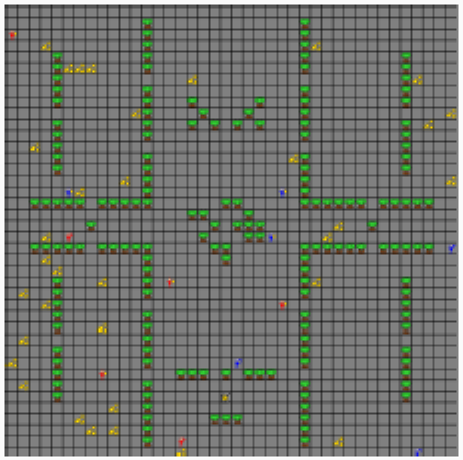
Key Features of BDI Agent-oriented Systems

Beliefs:	information about the world.
Events:	goals/desires to resolve; internal or external.
Plan library:	recipes for handling goals-events.
Intentions:	partially uninstantiated programs with commitment.



Key Features of BDI Agent-oriented Systems (cont.)

In the gold-mining game:



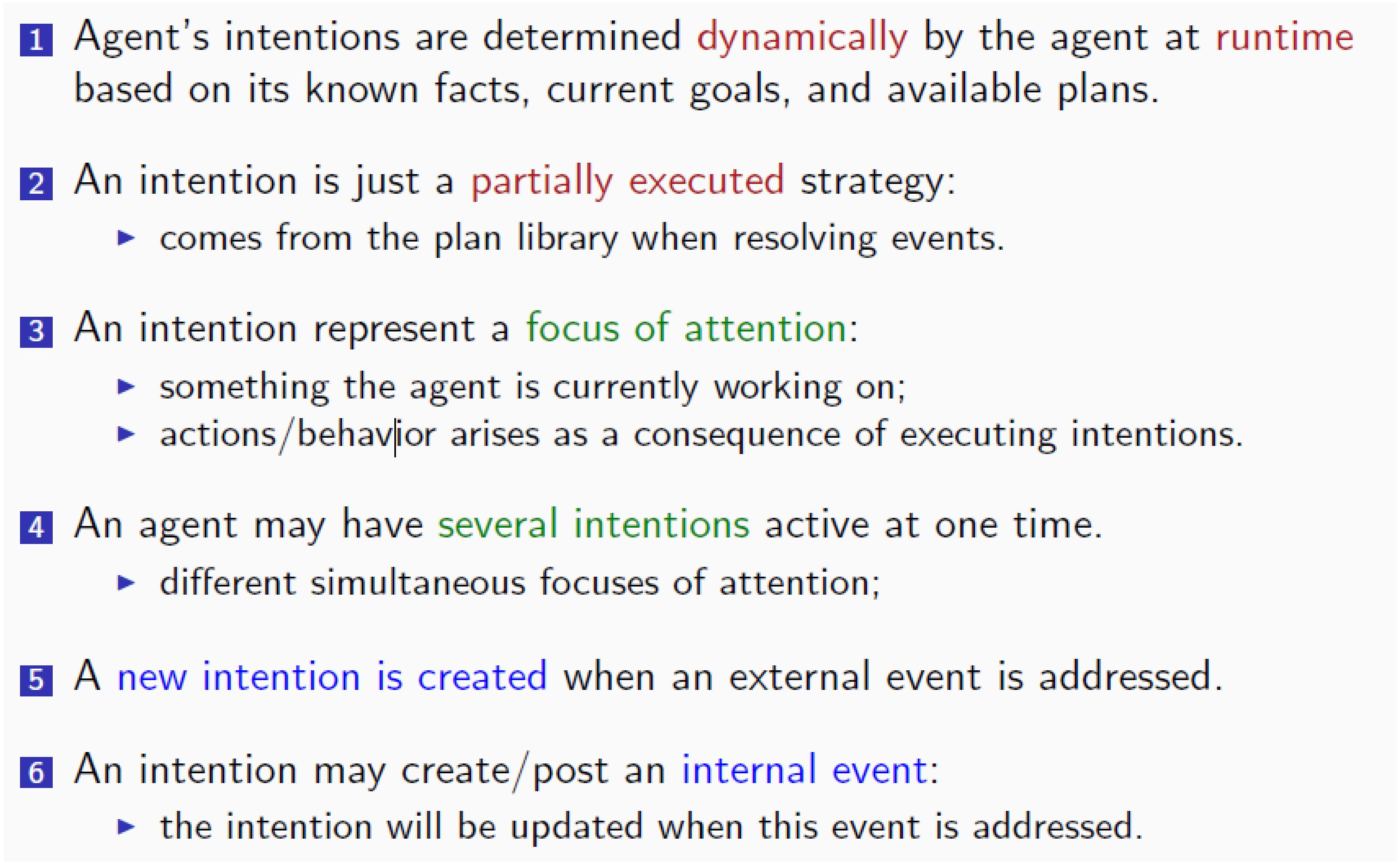
Beliefs: current location & location of depot.
static vs. dynamic size of grid, # of gold pieces carrying, etc.

Events: a gold piece is observed east;
external vs. internal player3 communicates its location;
the coordinator requests to explore the grid;
we formed the *internal goal* to travel to $loc(10, 22)$

Plan library: if I see gold here & I am not full, collect it.
if I hit an obstacle, go around it.
if I don't know of any gold, explore grid.
if I see gold around, move there and collect.

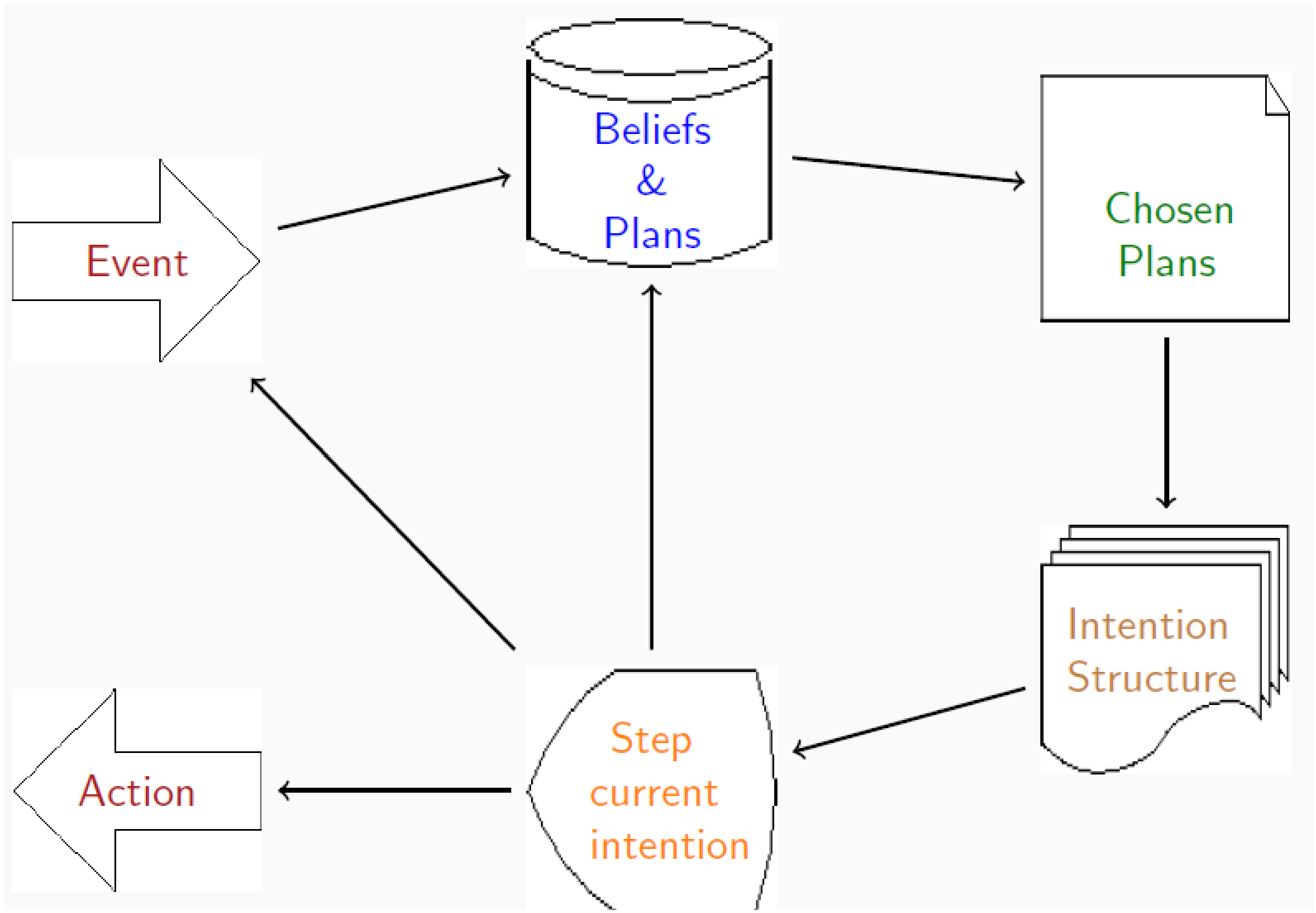
Intentions: I am currently traveling to the depot.
I am informing my team-mates of new obstacles I find.

Intentions



1. Agent's intentions are determined dynamically by the agent at runtime based on its known facts, current goals, and available plans.
2. An intention is just a partially executed strategy:
   - comes from the plan library when resolving events.
3. An intention represent a focus of attention:
   - something the agent is currently working on;
   - actions/behavior arises as a consequence of executing intentions.
4. An agent may have several intentions active at one time.
   - different simultaneous focuses of attention;
5. A new intention is created when an external event is addressed.
6. An intention may create/post an internal event:
   - the intention will be updated when this event is addressed.

The BDI Execution Cycle [Rao&Georgeff 92]



AgentSpeak

IMPLEMENTING BDI

Implementing BDI

We need:

1. Language for describing agent control programs
2. Runtime infrastructure (interpreter) to execute such programs

AgentSpeak (L)

Developed by A. S. Rao and has been influential in the design of other agent programming languages.

Programming language for **implementing BDI architectures**.

Extended to make it a practical agent programming language (R. Bordini).

AgentSpeak programs can be executed by the **Jason interpreter** (R. Bordini et al.).

- http://jason.sourceforge.net/

Based on **logic programming (Prolog)** using restricted first-order language with events and actions.

- There are also non-logic-based agent programming languages.

AgentSpeak: Beliefs

The **belief language** is based on first-order literals.

- $p(t_1, \ldots, t_n),\ \neg p(t_1, \ldots, t_n) \in beliefs$
- if $\phi, \psi \in beliefs$, then $\phi \wedge \psi \in beliefs$

The **belief base** of an AgentSpeak(L) program is a ground subset of the belief langauge

- e.g. $pos(r2,2,2) \wedge adjacent(a, b)$

AgentSpeak: Desires / Goals

Desires are based on **first order atomic formulae**.

- A **goal** is a desire that has been adopted for active pursuit by the agent.

Achievement goals: $!\, g(t_1, \ldots, t_n)$

- state that the agent wants to achieve a state of the world where the associated predicate is true.
- initiate the execution of *subplans*
- e.g. $!\, location(robot1, a)$

Test goals: $?\, g(t_1, \ldots t_n)$

- returns a unification for the associated predicate with one of the agent's beliefs; it fails if no unification is found.
- e.g. $?\, location(robot1, X)$

AgentSpeak: Events

Events initiate the execution of a plan.

Types of plan triggering events:

+b	(belief addition)
-b	(belief deletion)
+!g	(achievement-goal addition)
-!g	(achievement-goal deletion)
+?g	(test-goal addition)
-?g	(test-goal deletion)

External events generated from belief updates as a result of perceiving the environment or communication from other agents

Internal events generated from the agent’s own execution of a plan.

AgentSpeak: Plans

Plans are **context-sensitive** and **event-invoked recipes** to fulfil goals:

$$TriggeringEvent: Context \leftarrow PlanBody$$

- $TriggeringEvent$ denotes the purpose of the plan
- $Context$ is a conjunction of beliefs representing circumstances in which the plan can be used; context must be a logical consequence of that agent's current beliefs for the plan to be applicable.
- $PlanBody$ a sequence of basic actions or (sub)goals that the agent has to achieve (or test) when the plan, if applicable, is chosen for execution

Example

Triggering event

Context

```
+concert (A,V) : likes(A) <-
        !book_tickets(A,V).
```

Achievement goal added

```
+!book_tickets(A, V) :
        ¬busy(phone)
        <- call(V);
     …;
          !choose seats(A,V).
```

Basic action

Plan Example

```
+!at(Coords)
  : not at(Coords) & safe_path(Coords)
  ← move_towards(Coords); !at(Coords).
+!at(Coords)
  : not at(Coords) & no_safe_path(Coords) & not storm
  ← fly_towards(Coords); !at(Coords).
+!at(Coords)
  : not at(Coords) & very_bad_weather
  ← ask_for_teleport(Coords); ....
```

A plan failure triggers a goal-deletion event:

```
-!at(Coords)
  : very_bad_weather
  ← !wait_for_good_weather.
```

AgentSpeak: Intentions

Intention = plans the agent has **chosen for execution**.

Intentions are executed **one step at a time**.

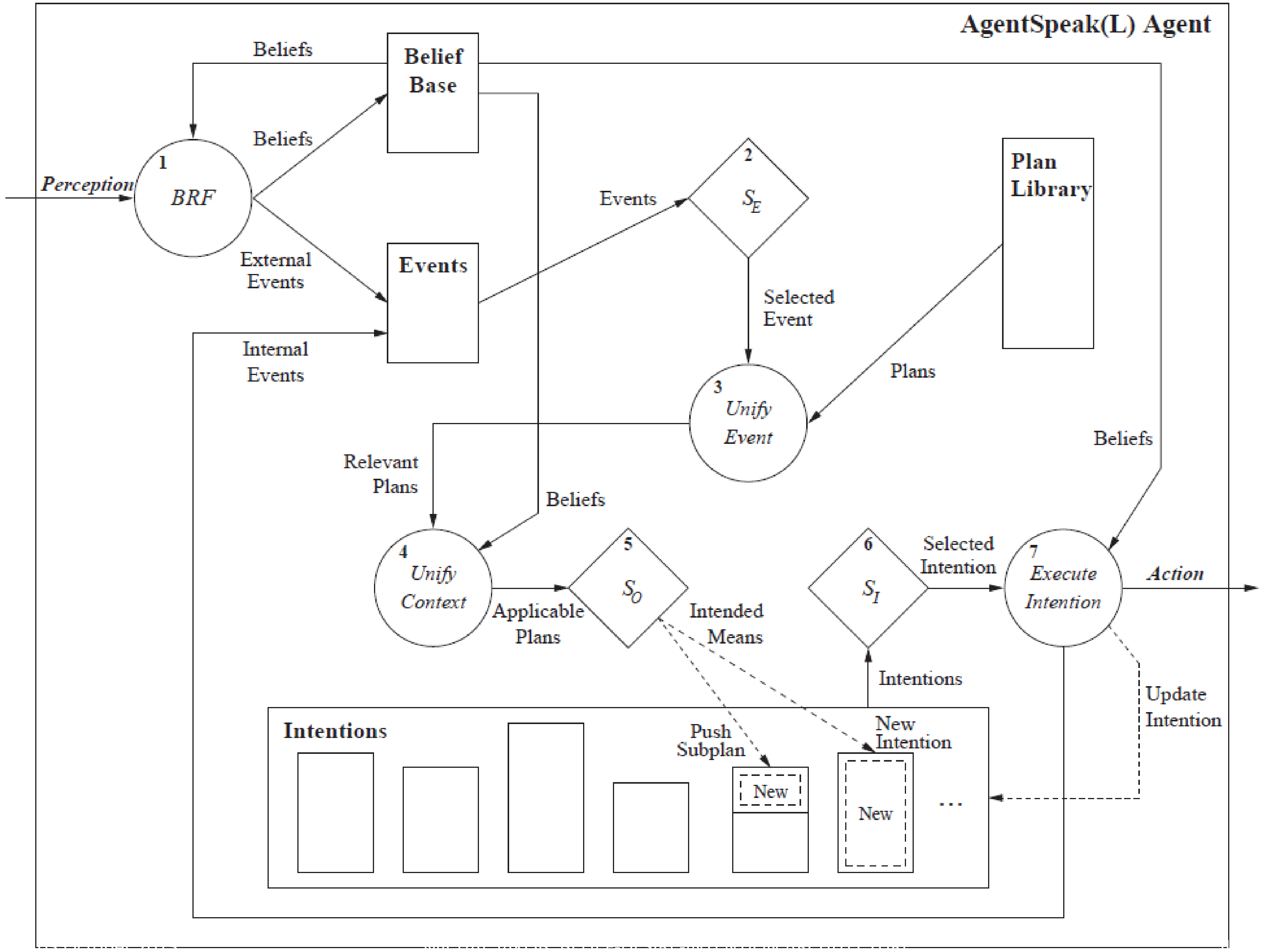
A step can

- query or change the beliefs
- perform actions on the external world
- suspend the execution until a certain condition is met
- submit new goals.

The operations performed by a step may generate new events, which, in turn, may start new intentions.

An intention succeeds when all its steps have been completed. It fails when certain conditions are not met or actions being performed report errors.

AgentSpeak Interpretation Cycle



AgentSpeak: Semantics

AgentSpeak(L) has an operational semantics defined in terms of agent configuration

$$\langle B, P, E, A, I, S_e, S_o, S_I \rangle$$

where

- B is a set of beliefs
- P is a set of plans
- E is a set of events (external and internal)
- A is a set of actions that can be performed in the environment
- I is a set of intentions each of which is a stack of partially instantiated plans
- S_e, S_o, S_I are selection functions for events, options, and intentions

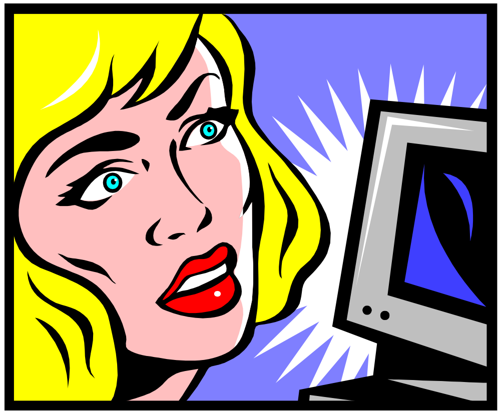
AgentSpeak: Selection functions

- S_e selects an events from E. The set of events is generated either by requests from users, from observing the environment, or by executing an intention
- S_o selects an option from P for a given event. An option is an applicable plan for an event, i.e. a plan whose triggering event is unifiable with event and whose condition is derivable from the belief base
- S_i selects an intention from I to execute.

Selection functions are agent-specific

- they should make selections based on an agent’s characteristics.

AgentSpeak: Example



ALICE

- During lunch time, forward all calls to Carla.
- When I am busy, incoming calls from colleagues should be forwarded to Denise.

AgentSpeak: Example Beliefs

```
user(alice).
user(bob).
user(carla).
user(denise).
~status(alice, idle).
status(bob, idle).
colleague(bob).
lunch_time("11:30").
```

AgentSpeak(L) Example Plans

```
user(alice).
user(bob).
user(carla).
user(denise).
~status(alice, idle).
status(bob, idle).
colleague(bob).
lunch_time("11:30").
```

"During lunch time, forward all calls to Carla".

```
+invite(X, alice) : lunch_time(t)    ←
   !call_forward(alice, X, carla).  (p1)
```

"When I am busy, incoming calls from colleagues should be forwarded to Denise".

```
+invite(X, alice) :
     colleague(X)  ←
   call_forward_busy(alice,X,denise).
                                     (p2)
```

```
+invite(X, Y): true   ←   connect(X,Y).
                                     (p3)
```

AgentSpeak Example Plans

```
user(alice).
user(bob).
user(carla).
user(denise).
~status(alice, idle).
status(bob, idle).
colleague(bob).
lunch_time("11:30").
+invite(X, alice) : lunch_time(t)   ←   !call_forward(alice, X, carla).   (p1)
+invite(X, alice) :     colleague(X)  ← call_forward_busy(alice,X,denise).(p2)
+invite(X, Y): true   ←  connect(X,Y).                                     (p3)
```

```
+!call_forward(X, From, To) : invite(From, X)
 ← +invite(From, To), - invite(From,X)                    (p4)
```

```
+!call_forvard_busy(Y, From, To) : invite(From, Y)&
   not(status(Y, idle)))

   ← +invite(From, To), - invite(From,Y).        (p5)
```

AgentSpeak Example

```
user(alice).
user(bob).
user(carla).
user(denise).
~status(alice, idle).
status(bob, idle).
colleague(bob).
lunch_time("11:30").

+invite(X, alice) : lunch_time(t)
            ← !call_forward(alice, X, carla).            (p1)
+invite(X, alice) :    colleague(X)
            ← call_forward_busy(alice,X,denise).         (p2)
+invite(X, Y): true   ←  connect(X,Y).                   (p3)
+!call_forward(X, From, To) : invite(From, X)
 ← +invite(From, To), - invite(From,X)                   (p4)
+!call_forward_busy(Y, From, To) : invite(From, Y)&
 not(status(Y, idle))
      ← +invite(From, To), - invite(From,Y).             (p5)
```

Execution - 1

A new event is sensed from the environment, **`+invite(Bob, Alice)`** (there is a call for Alice from Bob).

There are three *relevant* plans for this event (p1, p2 and p3)

- the event matches the triggering event of those three plans.

Relevant Plans	**Unifier**
`p1: +invite(X, alice) : lunch_time(t) ←!call_forward(alice, X, carla)`	
`p2: +invite(X, alice) : colleague(Bob) ← !call_forward_busy(alice, X, denise).`	{X=bob}
`p3 : +invite(X, Y): true ← connect(X,Y).`	{Y=alice, X=bob}

Execution - 2

Context of plan p2 is satisfied - `colleague(bob)` => p2 is *applicable*.

A new intention based on this plan is created in the set of intentions, because the event was external, generated from the perception of the environment.

The plan starts to be executed. It adds a new event, this time an internal event: `!call_forward_busy(alice,bob,denise)`.

Intention ID	Intension Stack	Unifier

1	`+invite(X,alice):colleague(X)`
	 `<- !call_forward_busy(alice,X,denise)`
		{X=bob}

Execution - 3

A plan relevant to this new event is found (p5):

Relevant Plans	**Unifier**
`p5: +!call_forward_busy(Y, From, To) : invite(From, Y) & not(status(Y, idle))) ← +invite(From, To), - invite(From,Y).`	{From=bob, Y=alice, To=denise}

- p5 has the context condition true, so it becomes an *applicable* plan and it is pushed on top of *intention 1 (*it was generated by an internal event)

Intention ID	**Intension Stack**	**Unifier**
1	`+!call_forward_busy(Y,From,To) : invite(From,Y) & not status(Y,idle) <- +invite(From,To); -invite(From,Y)`	{From=bob, Y=alice, To=denise}
	`+invite(X,alice) : colleague(X) <- !call_forward_busy(alice,X,denise)`	{X=bob}

Execution - 4

A new internal event is created, `+invite(bob, denise).`

three relevant plans for this event are found, p1, p2 and p3.

However, only plan p3 is applicable in this case, since the others don't have the context condition true.

The plan is pushed on top of the existing intention.

Intention ID	Intension Stack	Unifier
1	`+invite(X,Y) : <- connect(X,Y)`	{Y=denise, X=bob}
	`+!call_forward_busy(Y,From,To) : invite(From,Y) & not status(Y,idle) <- +invite(From,To); -invite(From,Y)`	{From=bob, Y=alice, To=denise}
	`+invite(X,alice) : colleague(X) <- !call_forward_busy(alice,X,denise)`	{X=bob}

Execution - 5

On top of the intention is a plan whose body contains an action.

The action is executed, **`connect(bob, denise)`** and is removed from the intention.

When all formulas in the body of a plan have been removed (i.e., have been executed), the whole plan is removed from the intention, and so is the achievement goal that generated it.

Intention ID	Intension Stack	Unifier
1	`+!call_forward_busy(Y,From,To) : invite(From,Y) & not status(Y,idle) <- -invite(From,Y)`	{From=bob, Y=alice, To=denise}
	`+invite(X,alice) : colleague(X) <- !call_forward_busy(alice,X,denise)`	{X=bob}

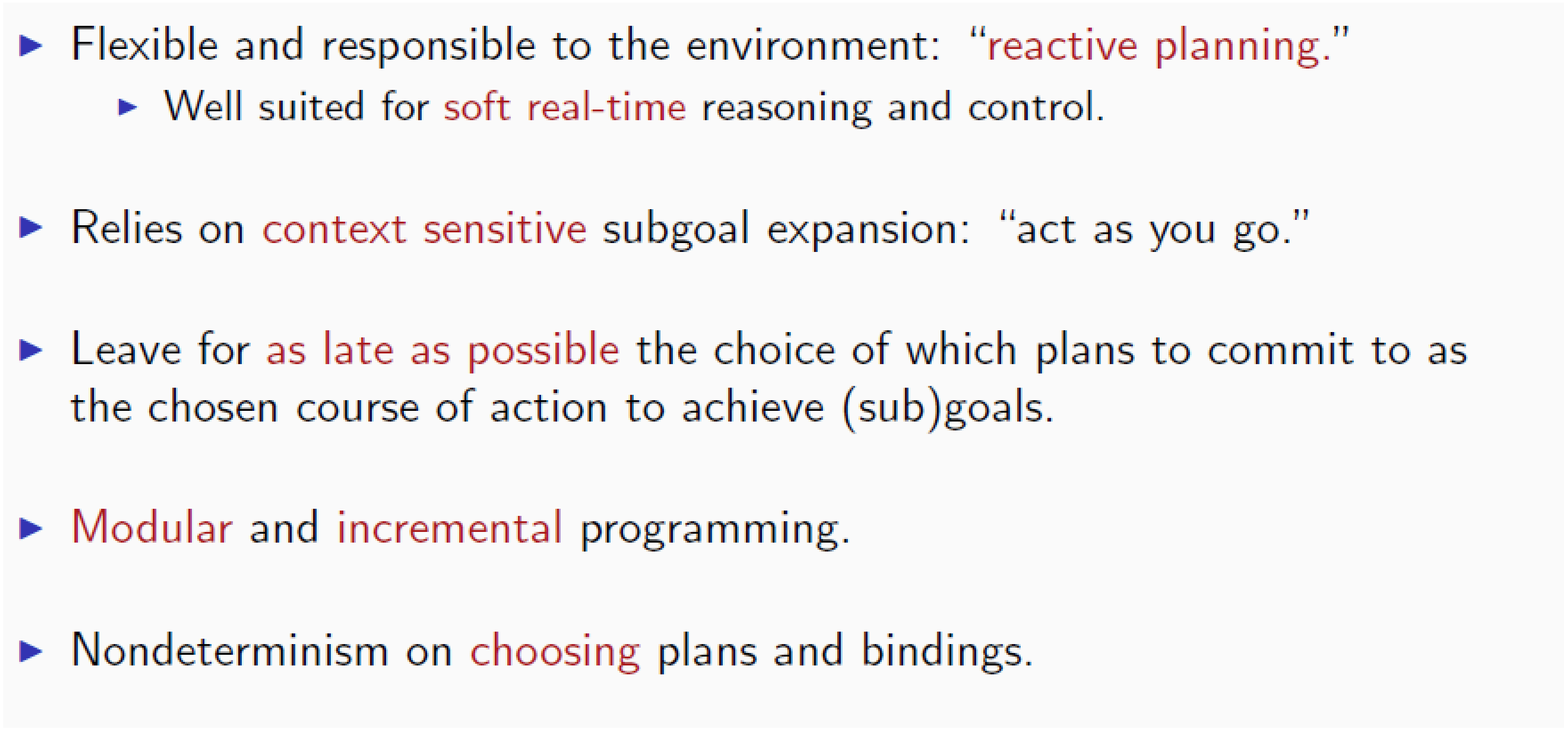
- The only thing that remains to be done is –invite(bob, alice) (this event is removed from the beliefs base).
- This ends a cycle of execution, and the process starts all over again, checking the state of the environment and reacting to events.

Time check!

BDI Programming Guidelines

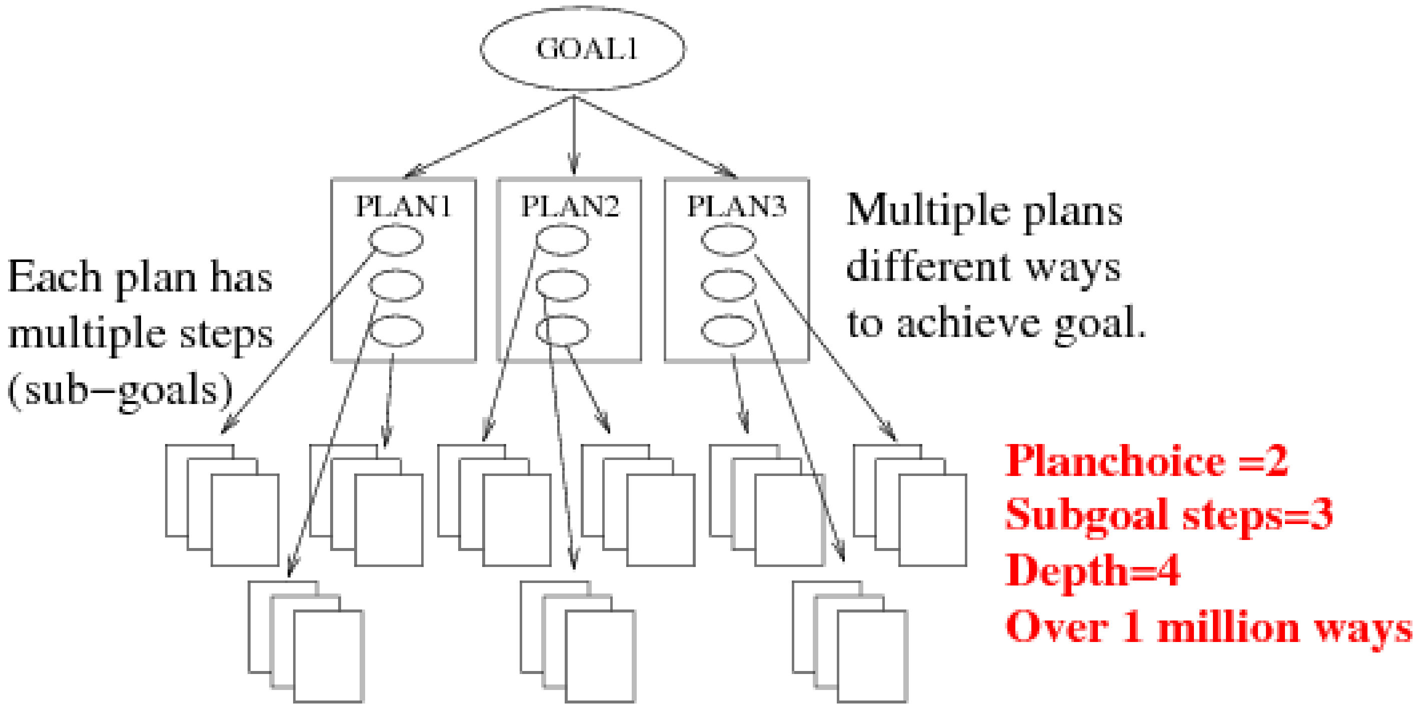
Key Points of BDI Programming

BDI Programming =
implicit goal-based programming + rational online executor



- Flexible and responsible to the environment: "reactive planning."
    - Well suited for soft real-time reasoning and control.
- Relies on context sensitive subgoal expansion: "act as you go."
- Leave for as late as possible the choice of which plans to commit to as the chosen course of action to achieve (sub)goals.
- Modular and incremental programming.
- Nondeterminism on choosing plans and bindings.

Possibility of Many Options

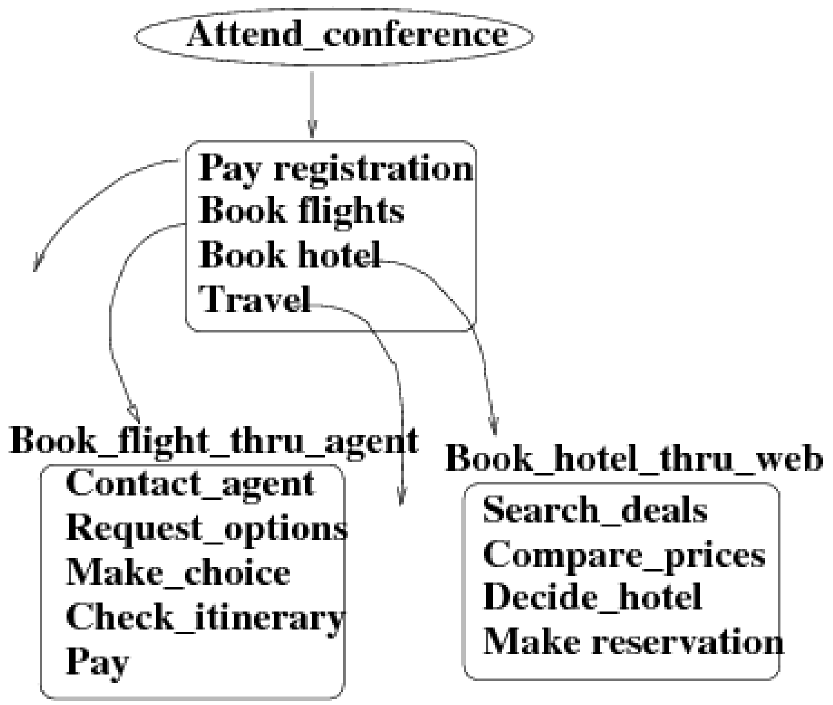

GOAL1
PLAN1
PLAN2
PLAN3
Each plan has multiple steps (sub–goals)
Multiple plans different ways to achieve goal.
Planchoice =2
Subgoal steps=3
Depth=4
Over 1 million ways

Here we have 30 plans, 81 way to achieve the goal. depends on choice of plans, number of steps, and depth of tree:

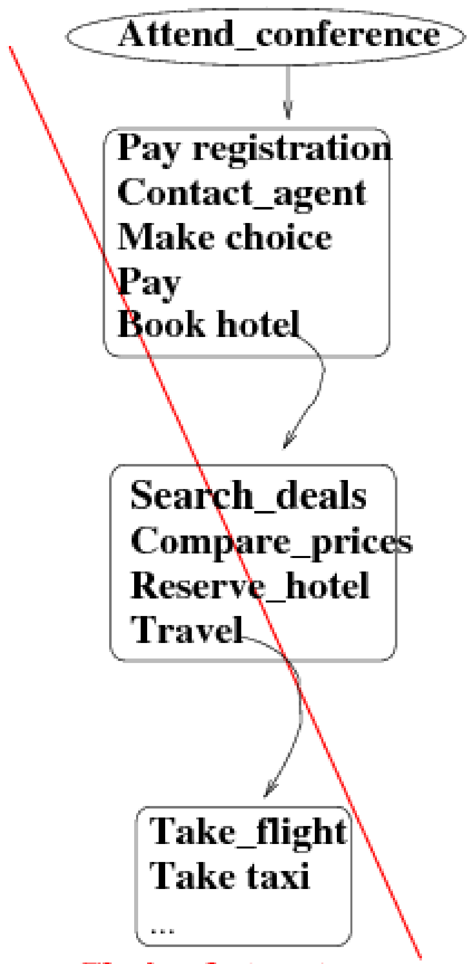
Making Use of the BDI Framework

1. Provide alternative plans where possible.
2. Break things down into subgoal steps.
3. Use subgoals and alternative plans rather than **if... then** in code.
4. Keep plans small and modular.
5. Plans are abstract modules - don’t chain them together like a flowchart.

Plan Structure



Hierarchical structure
– each plan is complete at its level of abstraction



Chained structure
– do some stuff then call next step...

~ HTN planning

Structuring Plans and Goals

Make each plan **complete** at a particular **abstraction level**.

- A high-level but complete plan for Attend Conference.

Use a **subgoal** - even if only one plan choice for now.

- Decouple a goal from its plans.

Modular and easy to **add other plan choices** later.

- Booking a flight can now be done with the Internet, if available!

Think in terms of **subgoals**, not function calls.

- What way-points do we need to achieve so as to realize a goal?

Learn to **pass information** between subgoals.

  - How are these way-points inter-related w.r.t. data?

BDI Summary

Practical way to implement **goal-oriented agents**.

Based on the theory of practical reasoning that human appear to use in daily lives.

Programming using mentalistic concepts of beliefs, desires and intentions.

BDI languages and executors/intepreters exist for implementation of BDI agents

- logic-based AgentSpeak language together with Jason interpreters probably the best known

Reading

- BDI agent programming in AgentSpeak using Jason Sections 1-3
- BDI lecture notes (Tambe/Greenstadt)