Robotics and Autonomous Systems

Lecture 24: Goal-based programming

Richard Williams

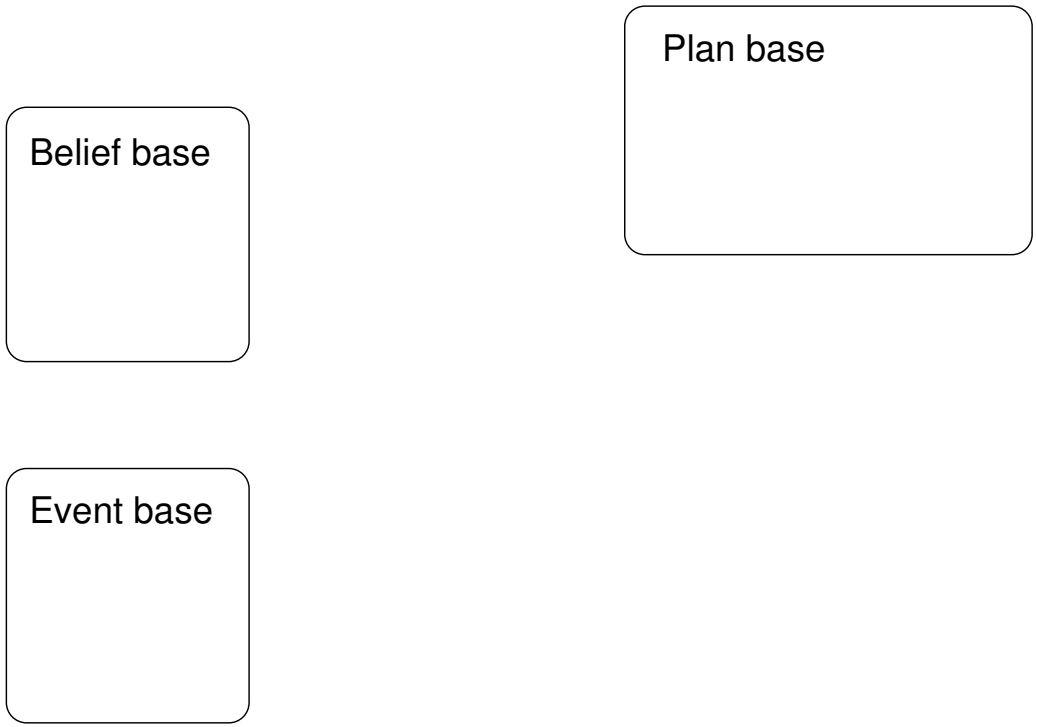
Department of Computer Science
University of Liverpool

UNIVERSITY OF LIVERPOOL

Today

- We will look at three things.
- We will recap the way that Jason supports BDI programming.
- We will look at some useful internal actions provided by Jason
- We will look at some common patterns of handling goals.

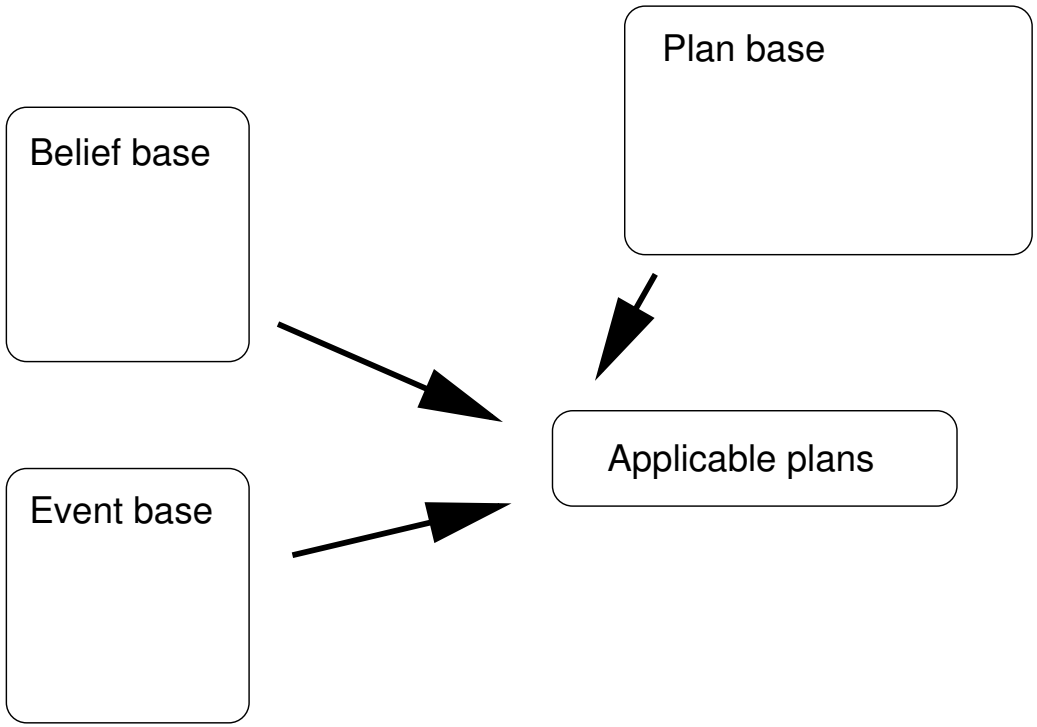
BDI



BDI

- The basic components are the belief base, the plan base, and the event base.

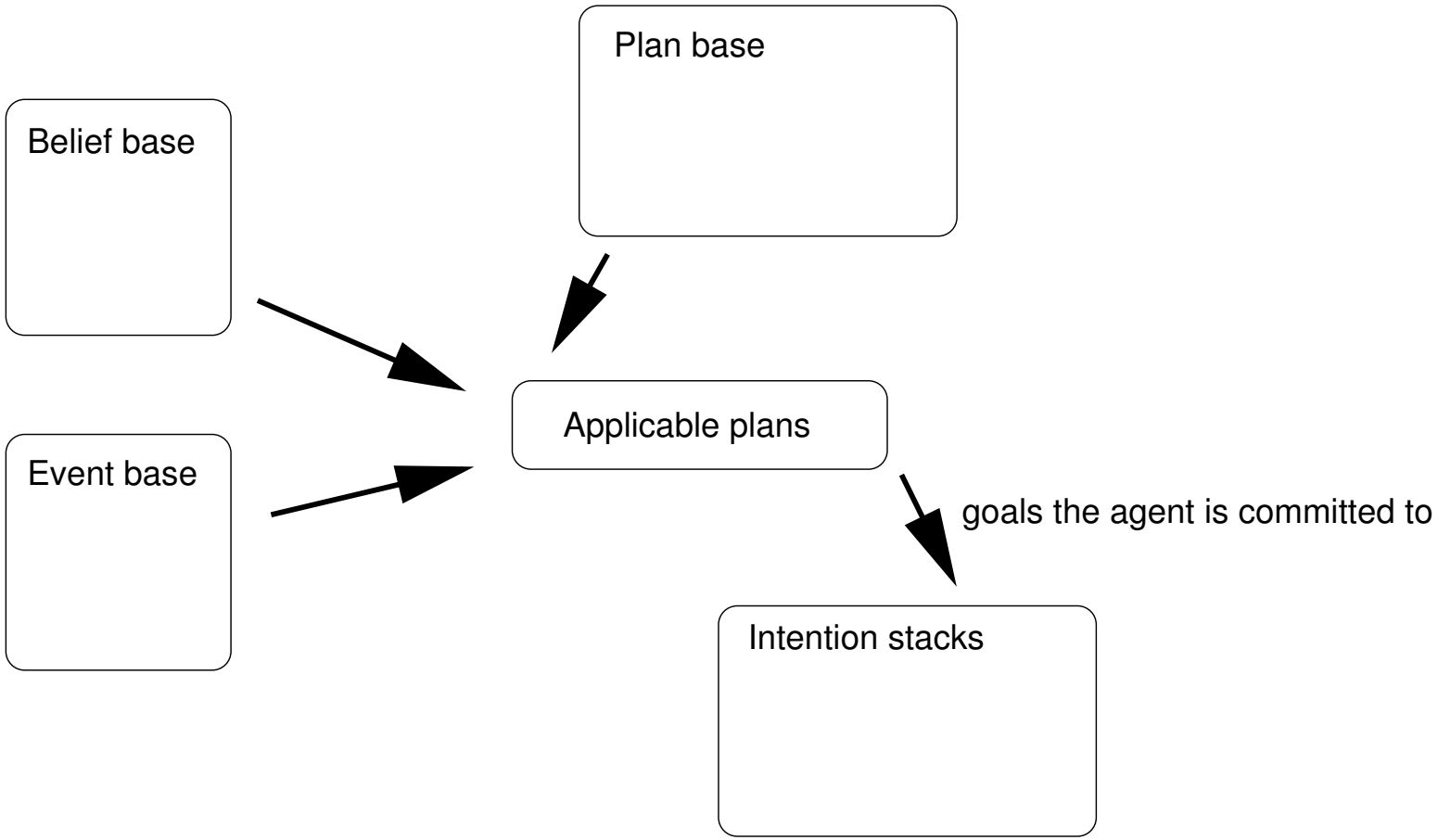
BDI



BDI

- Events and beliefs are matched against plans in the plan base to establish a set of applicable plans.

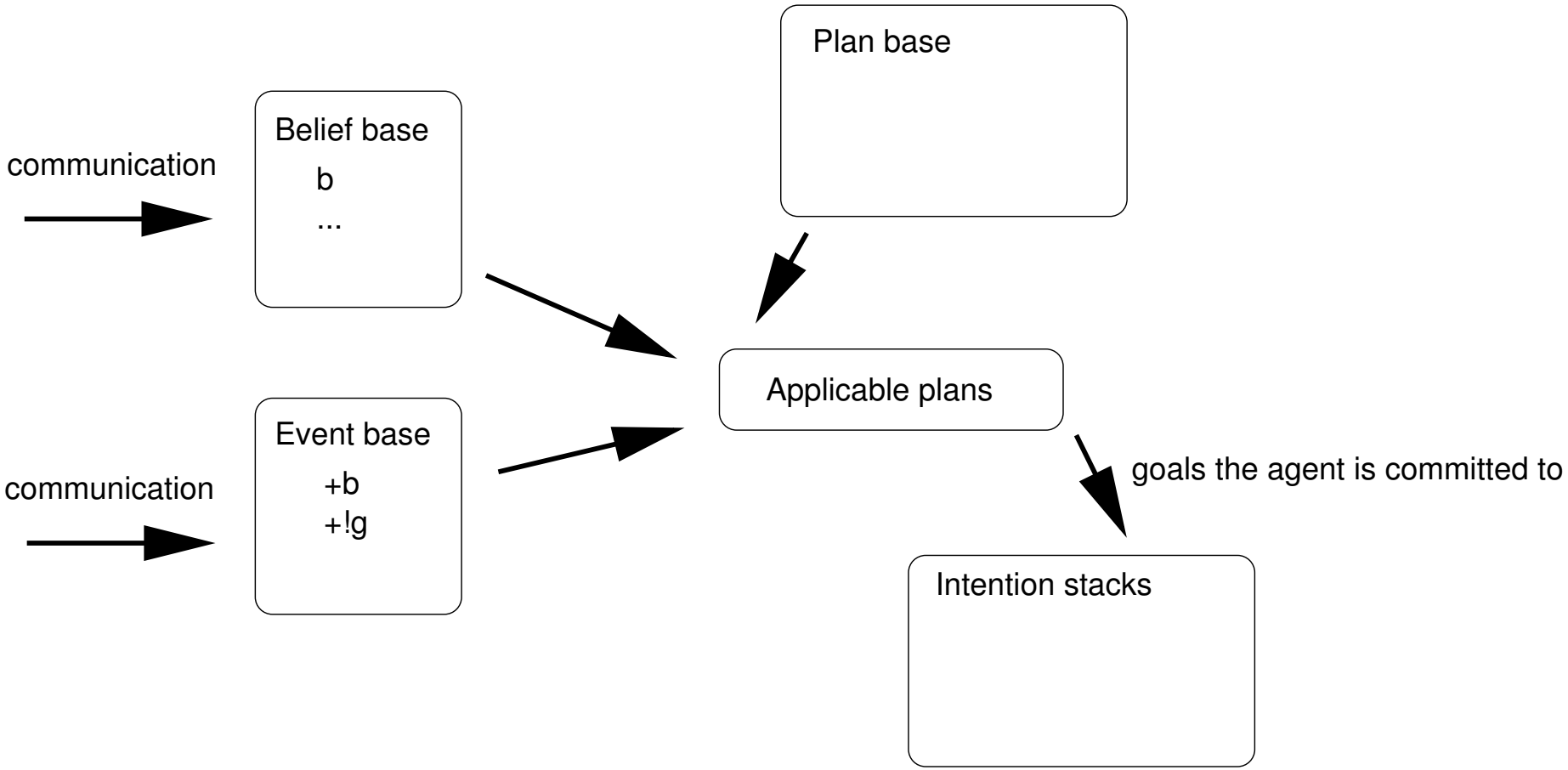
BDI


Plan base
Belief base
Applicable plans
Event base
goals the agent is committed to
Intention stacks

BDI

- A plan is selected, its goal is the current intention, and the plan is pushed onto the intention stack.

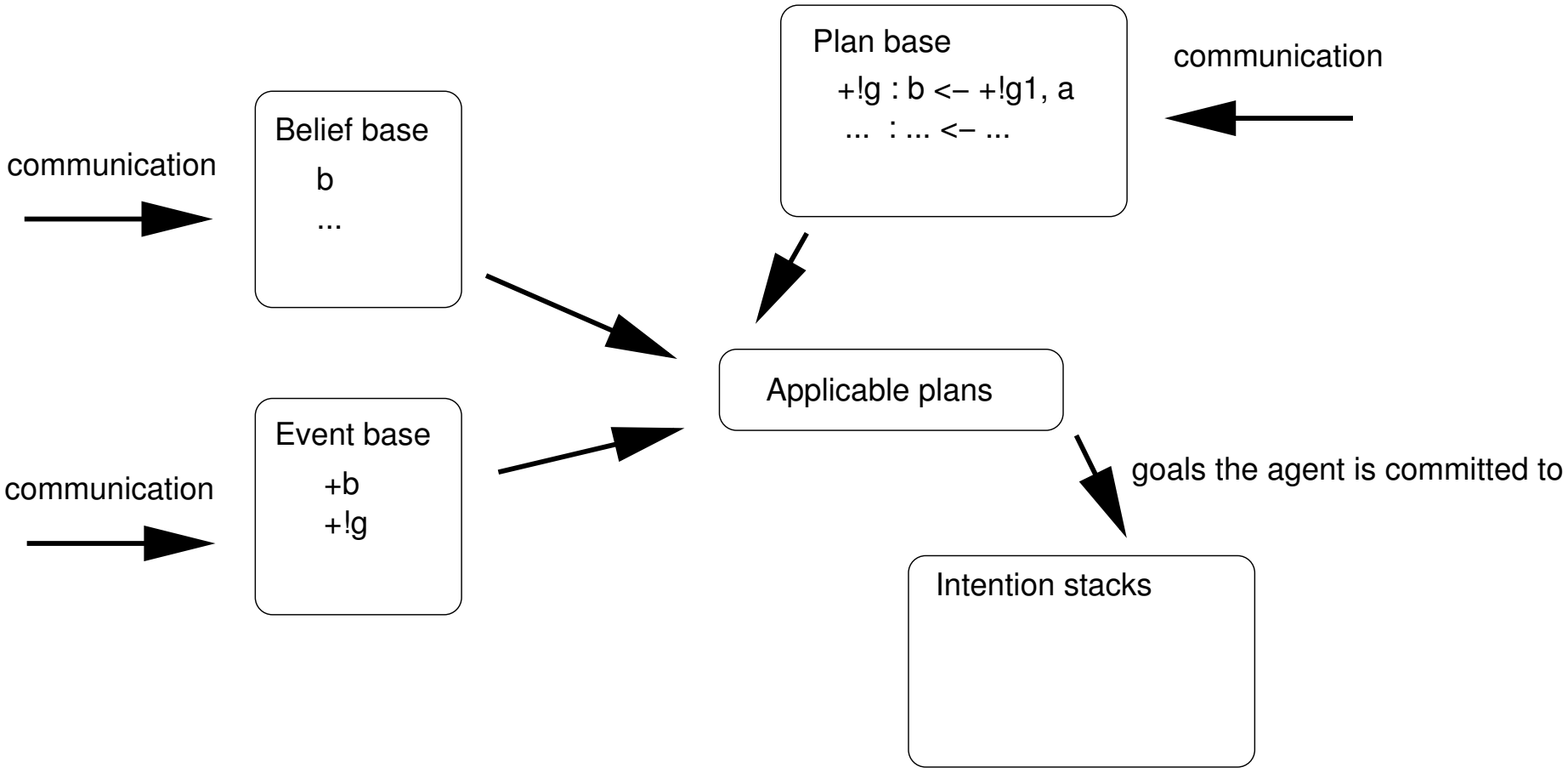
BDI


Plan base
communication
Belief base
b
...
Applicable plans
Event base
+b
+!g
communication
goals the agent is committed to
Intention stacks

BDI

- Communication adds to the belief and event bases.
- Here a belief b is added to the belief base, and so +b is added to the event base.
- We also see a goal added to the event base.

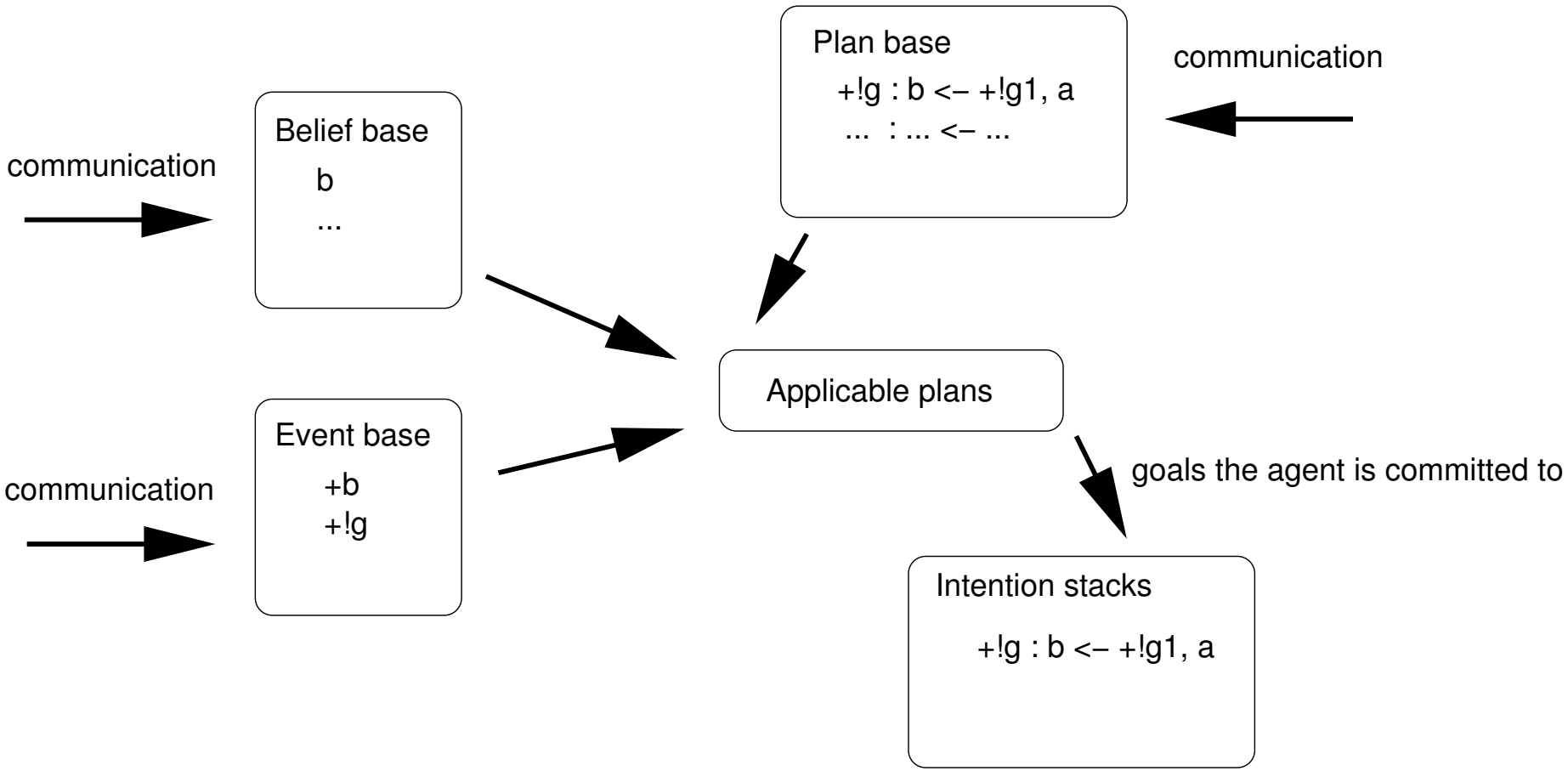
BDI


Plan base
+!g : b <– +!g1, a
... : ... <– ...
communication
communication
Belief base
b
...
Applicable plans
Event base
+b
+!g
communication
goals the agent is committed to
Intention stacks

BDI

- Plans can also be communicated.

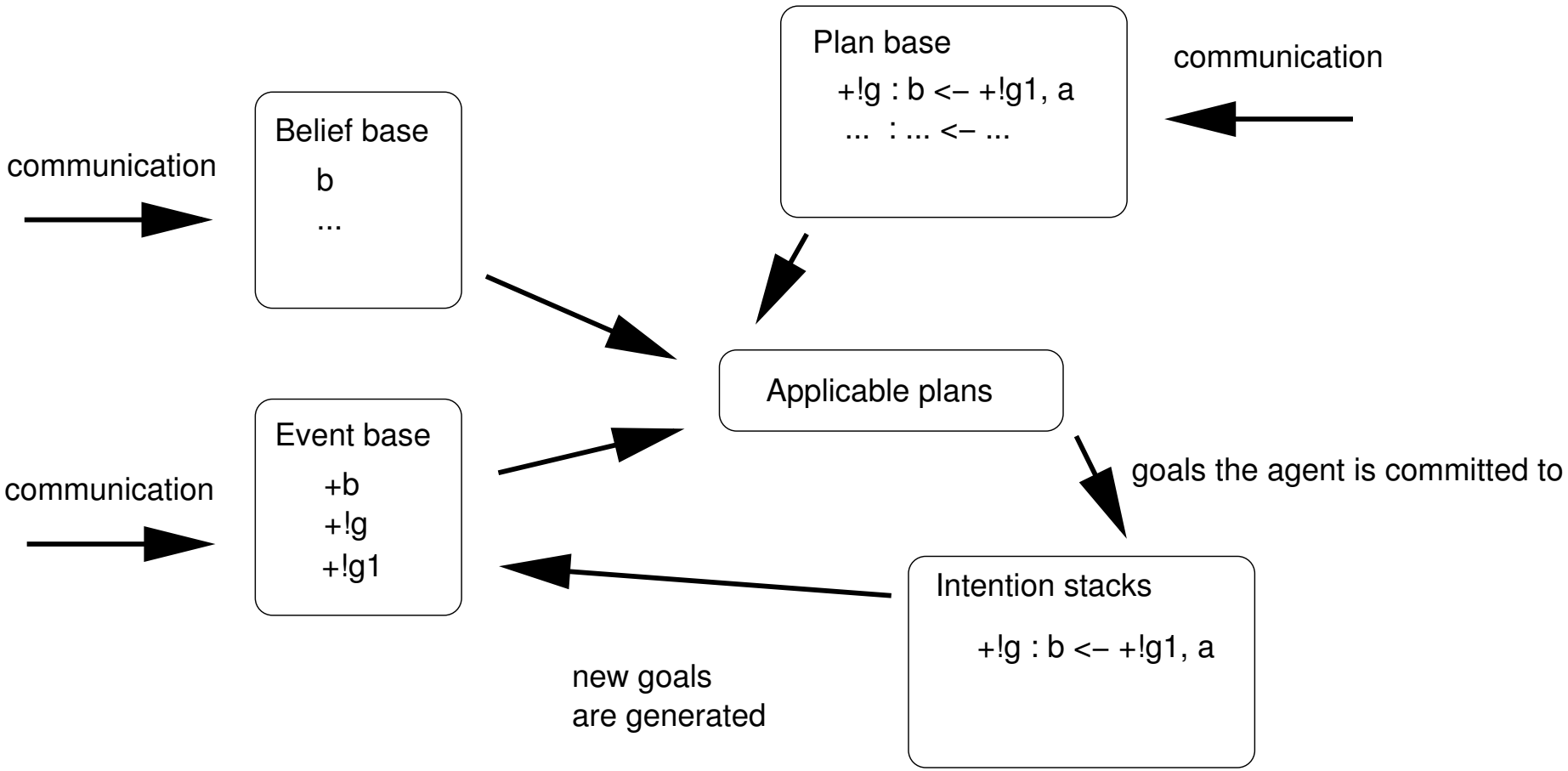
BDI


Plan base
+!g : b <– +!g1, a
... : ... <– ...
communication
communication
Belief base
b
...
Applicable plans
Event base
+b
+!g
communication
goals the agent is committed to
Intention stacks
+!g : b <– +!g1, a

BDI

- The event +!g matches with the plan in the plan base, is added to the set of applicable plans.
- The plan is then adopted, and pushed onto the intention stack.

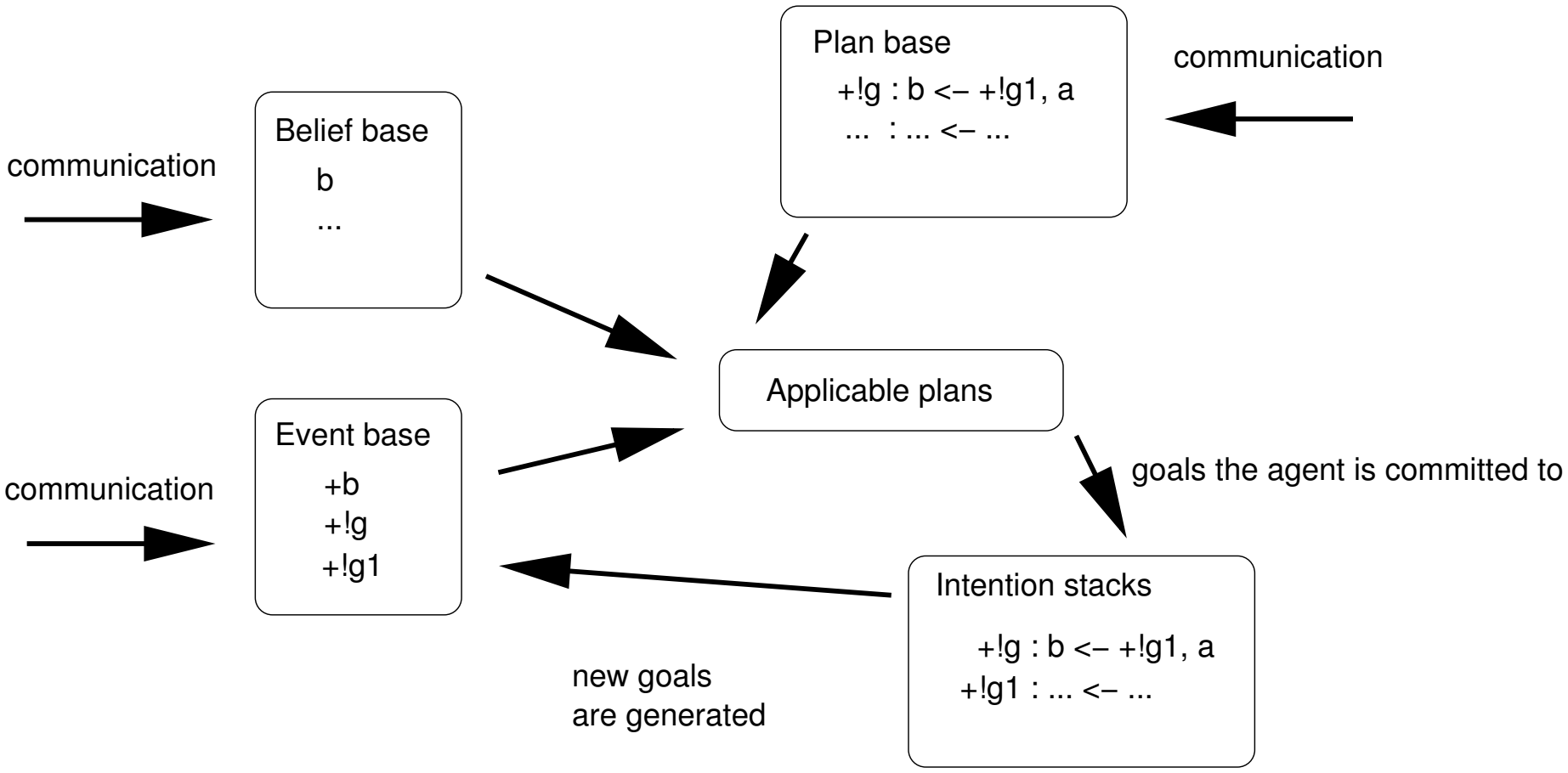
BDI


Plan base
+!g : b <– +!g1, a
... : ... <– ...
communication
communication
Belief base
b
...
Applicable plans
Event base
+b
+!g
+!g1
communication
goals the agent is committed to
Intention stacks
+!g : b <– +!g1, a
new goals
are generated

BDI

- Executing that plan invokes a new goal `g1`.
- This is added to the event base.

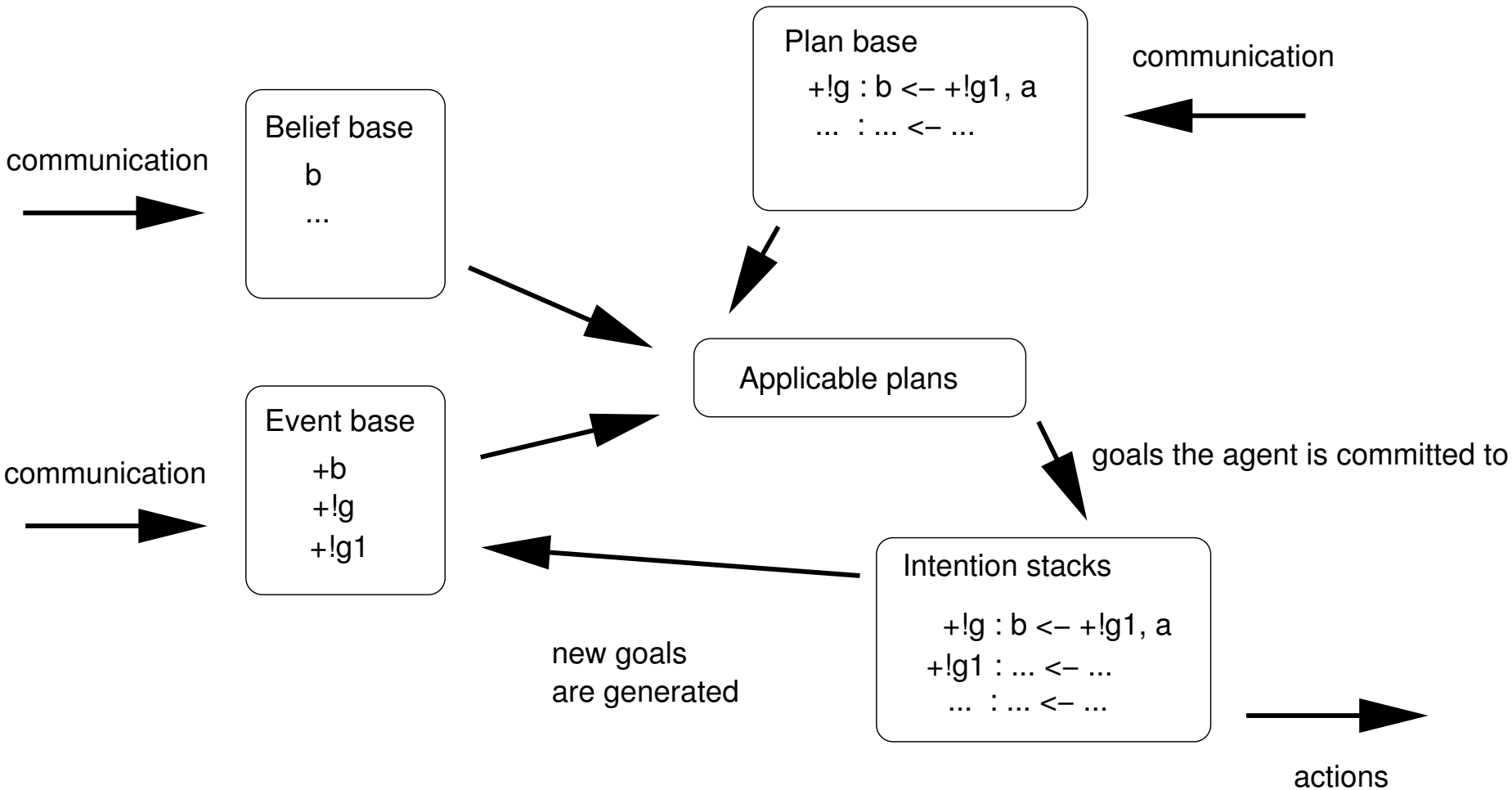
BDI


Plan base
+!g : b <– +!g1, a
... : ... <– ...
communication
communication
Belief base
b
...
Applicable plans
Event base
+b
+!g
+!g1
communication
goals the agent is committed to
Intention stacks
+!g : b <– +!g1, a
+!g1 : ... <– ...
new goals
are generated

BDI

- A new plan is adopted for the new goal, and this is pushed onto the intention stack.
- Note that the stack is not drawn like a stack.

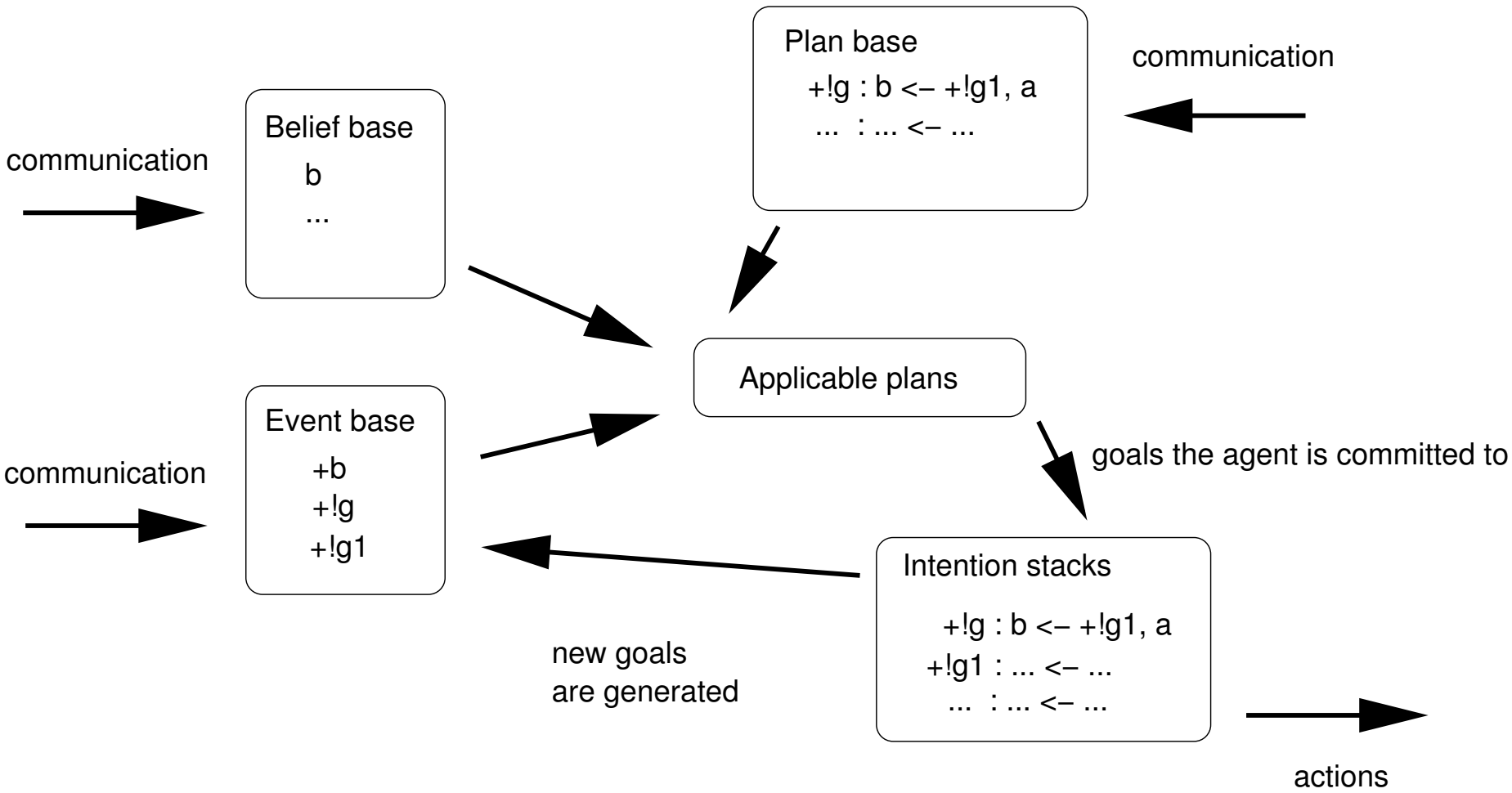
BDI


Plan base
+!g : b <– +!g1, a
... : ... <– ...
communication
communication
Belief base
b
...
Applicable plans
Event base
+b
+!g
+!g1
communication
goals the agent is committed to
Intention stacks
+!g : b <– +!g1, a
+!g1 : ... <– ...
... : ... <– ...
new goals
are generated
actions

BDI

- Another plan is required to achieve g1

BDI



- Desires: goals in the event base or in the head of some intention

Internal Actions

- Jason contains a number of internal actions that are useful for programming agents.
- Have already seen some:
  - `.print`
  - `.send`
- Here are more.
- More related to manipulating goals.

Internal Actions

- `.desire(literal p)`
  The action succeeds if the literal unifies with some goal (i.e. `+g!`) in the event base or in the head of some plan.
  `.desire(go(1, 3))` is true if `go(1, 3)` is a desire.
- `.intend(literal p)`
  The action succeeds if the literal unifies with some goal in the body of some intention.
  `.intend(go(1, 3))` is true if a plan with triggering event `+!(go(1, 3)` appears as an intention.

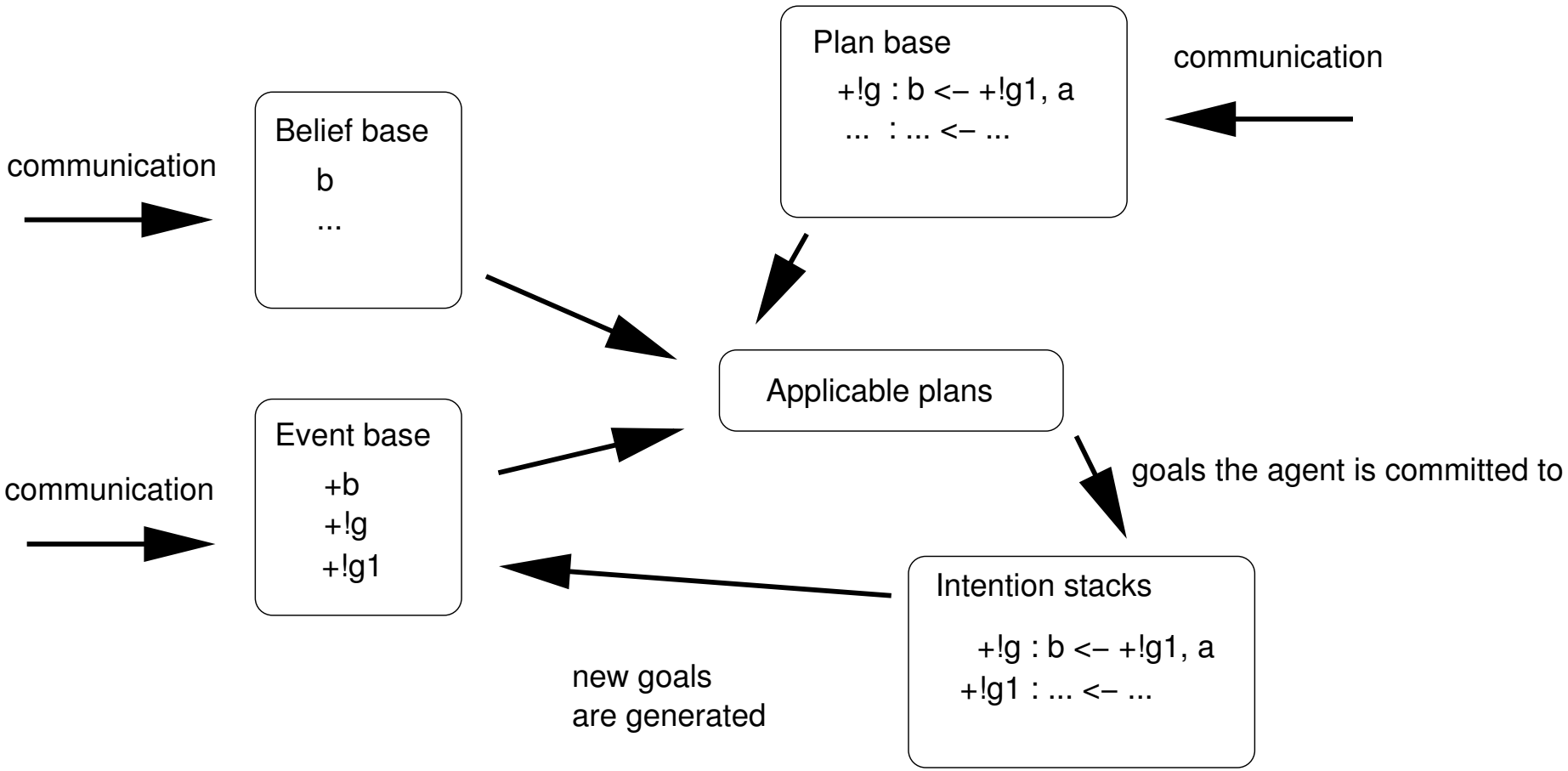
Internal Actions

- These are less “actions” in the conventional sense, than ways to test for the existence of a desire or intention and/or to capture its value in a variable.
- This value can then be used.
- For example, having detected `go(1, 3)`, above, we might then try to execute it.
- Or we might use the test to avoid re-creating an existing desire.

Internal Actions

- `.drop_desire(literal p)`
  Removes the argument desire. If the argument is `D`, then every entry `+!D` that occurs in the set of events or the set of intentions is deleted. No events are triggered.
- `.drop_intention(literal p)`
  Removes all the plans where the triggering condition unifies with the given literal
- `.drop_event(literal p)`
  Removes only goals in the event base that do not appear in any intention (plan head)

BDI


Plan base
+!g : b <– +!g1, a
... : ... <– ...
communication
communication
Belief base
b
...
Applicable plans
Event base
+b
+!g
+!g1
communication
goals the agent is committed to
Intention stacks
+!g : b <– +!g1, a
+!g1 : ... <– ...
new goals
are generated

Internal actions

- Note that these last three are complementary.

  `.drop_desire` removes its argument from
  - all events and
  - all intentions.

  `.drop_intention` removes its argument from
  - all intentions.

  `.drop_event` removes its argument from
  - all events.

Internal actions

- `.succeed_goal(literal p)`
  States the goal has succeeded.
  System behaves as if this has happened and the intentions are updated accordingly.

Internal actions

```
+!g2 : cxt2
  <- ... .

+!g1 : cxt1
  <- !g2
     ... .

+!g0 : cxt0
  <- !g1
     !g4
     ... .
```

```
+!g4 : cxt4
  <- ... .

+!g0: cxt0
  <- !g4
     ... .
```

- `.succeed_goal(!g1)`

Internal actions

- `.fail_goal(literal p)`
  States the goal has failed and is impossible to achieve.
  Plan is removed and a goal deletion event is generated for the plans requiring the goal.

Internal actions

```
+!g2 : cxt2
  <- ... .

+!g1 : cxt1
  <- !g2
     ... .

+!g0 : cxt0
  <- !g1
     !g4
     ... .
```

```
-!g0 : cxt0
  <- ... .

+!g0: cxt0
  <- !g1
     !g4
     ... .
```

- `.fail_goal(!g1)`

Declarative goals

- Up until now we have treated goals in a procedural fashion. Goals as triggers for plans

```
+!g:  c <- p
```

- Goals, however are often meant in a declarative way. For example, a robot that has to reach location (X,Y)
- Here what is meant is that the robot should reach the given location and expects the robot to believe it has reached it (i.e., believe the goal has been achieved)
- Procedural goals can be “made declarative” by the following transformation

Declarative goals

- ```
  +!l(X,Y) : bc(B) & B > 0.2
             <- go(X,Y).
  ```

  becomes

  ```
  +!l(X,Y) : bc(B) & B > 0.2
             <- go(X,Y); ?l(X,Y).
  ```

- `bc` is “battery charge”
- `l` is “at location”
- The test goal at the end forces the failure of the plan if the robot does not make it to the relevant location.
- Think of transformations such as these to be design patterns.
  ```

Declarative goal pattern

- A set of plans P for some goal:

```
+!g :  c1 <- p1
+!g :  c2 <- p2
⋮
+!g :  cn <- pn
```

- Is transformed into the set $DG(P)$ by:

```
+!g :  g <- true
+!g :  c1 <- p1; ?testg
+!g :  c2 <- p2; ?testg
⋮
+!g :  cn <- pn; ?testg
+g :  true <- .succeed_goal(g)
```

Declarative goal pattern

- The first line tests if the goal is true
  - Success means there is nothing to do.
- In the middle we have the plans with the test goals appended.
- The last plan is triggered if the agent realises that its goal has been achieved. Deletes any additional attempts to achieve it.

Goal programming patterns

- Programming with goals and plans offers a wide array of possibilities for structuring your code
- Different structures can serve different purposes and model different types of intelligent decision-making
- We will look now at a number of patterns that can be used to program different ways in which goals and plans interact generating different goal-driven behaviors

Backtracking declarative goals

- We transform the set of plans P into $DG(P)$ followed by:

```
-!g : true <- !!g
```

- If a plan from P fails, then try to achieve the goal again.
- This can loop if you aren't careful with the contexts.
- We call this transformation *BDG*.

Patterns for commitment

- We can use this kind of transformation to capture different kinds of commitment.
  - When to drop a goal

Remember?

Some time in the not-so-distant future, you are having trouble with your new household robot. You say "Willie, bring me a beer." The robot replies "OK boss." Twenty minutes later, you screech "Willie, why didn't you bring me that beer?" It answers "Well, I intended to get you the beer, but I decided to do something else." Miffed, you send the wise guy back to the manufacturer, complaining about a lack of commitment. After retrofitting, Willie is returned, marked "Model C: The Committed Assistant." Again, you ask Willie to bring you a beer. Again, it accedes, replying "Sure thing." Then you ask: "What kind of beer did you buy?" It answers: "Genessee." You say "Never mind."

One minute later, Willie trundles over with a Genessee in its gripper. [. . .] After still more tinkering, the manufacturer sends Willie back, promising no more problems with its commitments. So, being a somewhat trusting customer, you accept the rascal back into your household, but as a test, you ask it to bring you your last beer. [. . .] The robot gets the beer and starts towards you. As it approaches, it lifts its arm, wheels around, deliberately smashes the bottle, and trundles off. Back at the plant, when interrogated by customer service as to why it had abandoned its commitments, the robot replies that according to its specifications, it kept its commitments as long as required — commitments must be dropped when fulfilled or impossible to achieve. By smashing the bottle, the commitment became unachievable.

Blind committment

- We transform the set of plans P into declarative form followed by:

```
+!g :  true <- !!g
```

- (For this the book suggests using the declarative transformation $F(P)$, similar to $BDG(P)$.)
- "If g is added to the goal base, then create a new intention stack to handle g".
- Only give up on a goal when it is achieved.
- We call this transformation BC.

Single-minded commitment

- Blind commitment is usually too strong, so a weaker form is often useful.
- We transform the set of plans P into $BC(P)$ followed by:

```
+f :  true <- .fail_goal(g)
```

- Thus, in some situations, captured by `f`, we can drop the goal.
- If the agent comes to believe `f`, then it stops trying to achieve the goal.

Relativised commitment

- In single-minded commitment, goals are only dropped when unachievable.
- What if the reason for them goes away.
  - As in the Genesse example
- Relativised commitment
- We transform the set of plans P into $BC(P)$ followed by:

```
-m :  true <- .succeed_goal(g)
```

- If the motivation `m` disappears, then we have automatically succeeded in achieving the goal.

Open-minded commitment

- We combine the two previous patterns to get open-minded commitment:
- We transform the set of plans P into $BC(P)$ followed by:

```
+f :  true <- .fail_goal(g)
-m :  true <- .succeed_goal(g)
```

- A goal is pursued until achieved or until believed to be unachievable or until its motivation is no longer believed

Maintenance goals

- What we have looked at so far are achievement goals.
- Unlike achievement goals , maintenance goals are goals to maintain a given state-of-affairs, or to keep some aspect of the environment to have a certain value
  For example, bank balance above 0.
- Can make this goal into an achievement goal by a suitable transformation

Maintenance goals

- We transform the set of plans P into

```
g[source(percept)]
-g : true <- !g
```

  followed by $F(P)$.
- Where $F(P)$ is one of the transformations we saw before.
- Initally set g based on the value returned by some sensor.
- If g becomes false, make it true.

Summary

- We have looked at some more advanced features of goals as programming constructs
- In particular:
  - achievement vs. maintenance goals
  - procedural vs. declarative goals
  - types of commitments (blind, single-minded, relativized, open-minded)
- These might be of use for the second assignment