

Introduction to Distributed and Embedded Multi-agent Systems

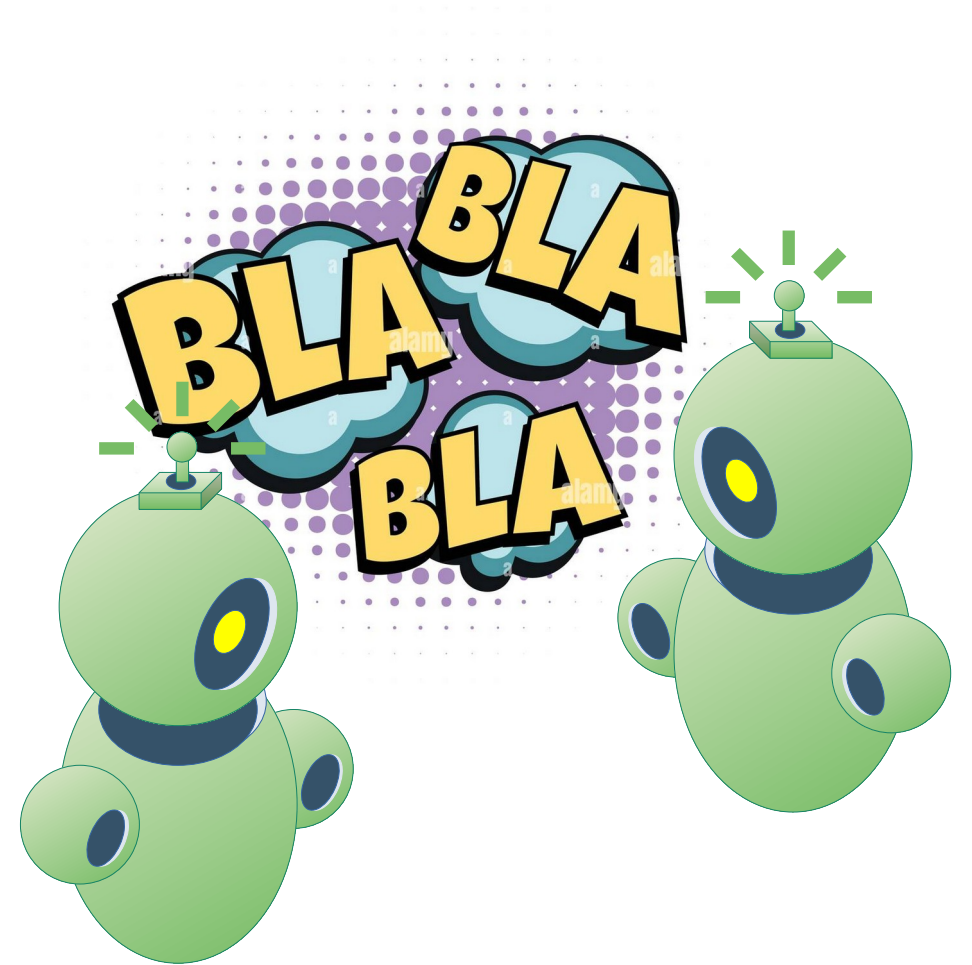
Carlos Eduardo Pantoja¹
Nilson Mori Lazarin^{1,2}

1. Centro Federal de Educação Tecnológica (CEFET/RJ) - 2. Universidade Federal Fluminense (UFF), Brasil



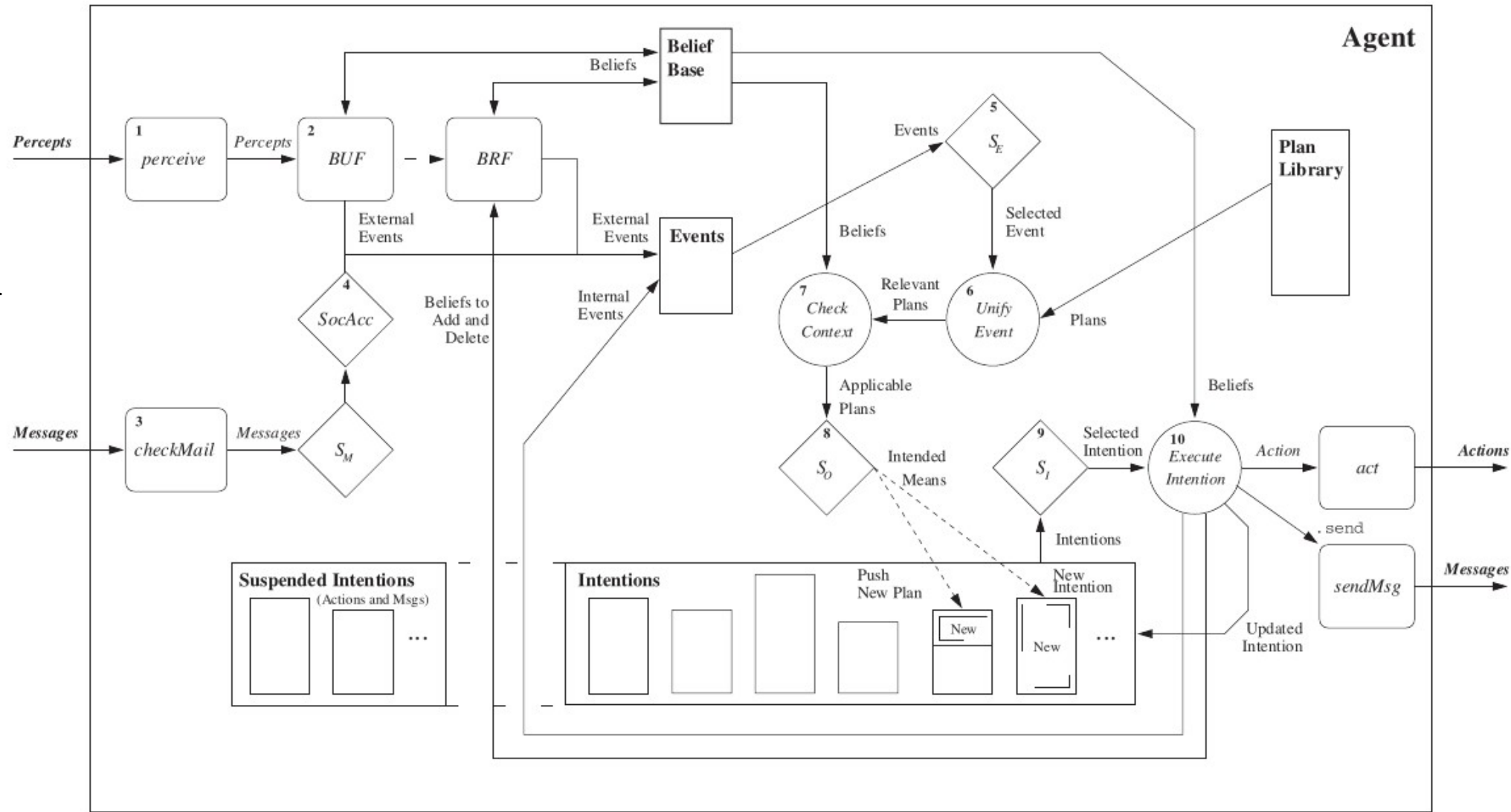
Março, 2024

COMMUNICATION IN MULTI-AGENT SYSTEMS



The Jason Reasoning Cycle

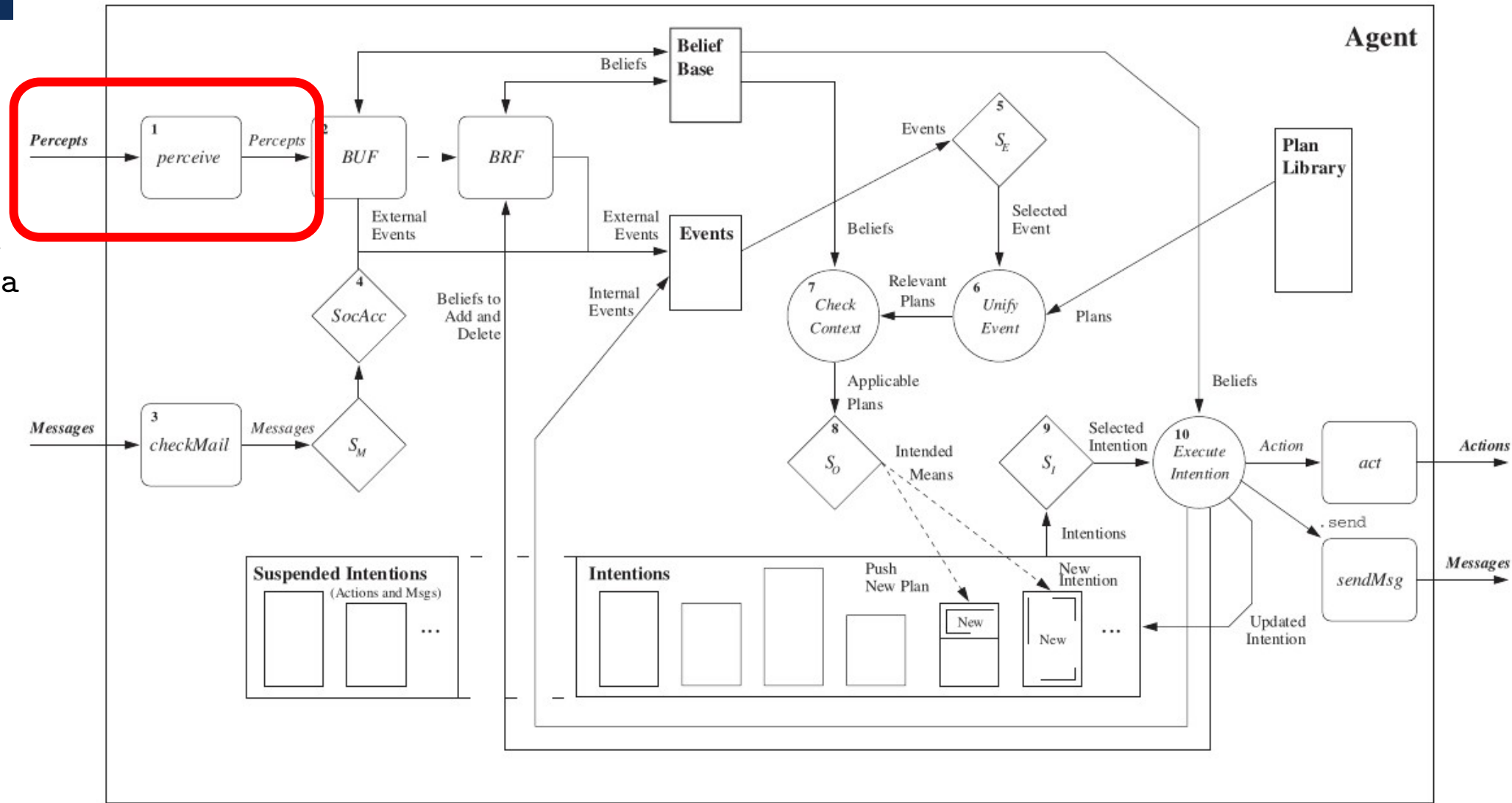
How the Jason Interpreter runs an agent program [1].



[1] R. H. Bordini, J. F. Hübner, e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason. em Wiley Series in Agent Technology. Wiley, 2007.

1. Perceiving the Environment

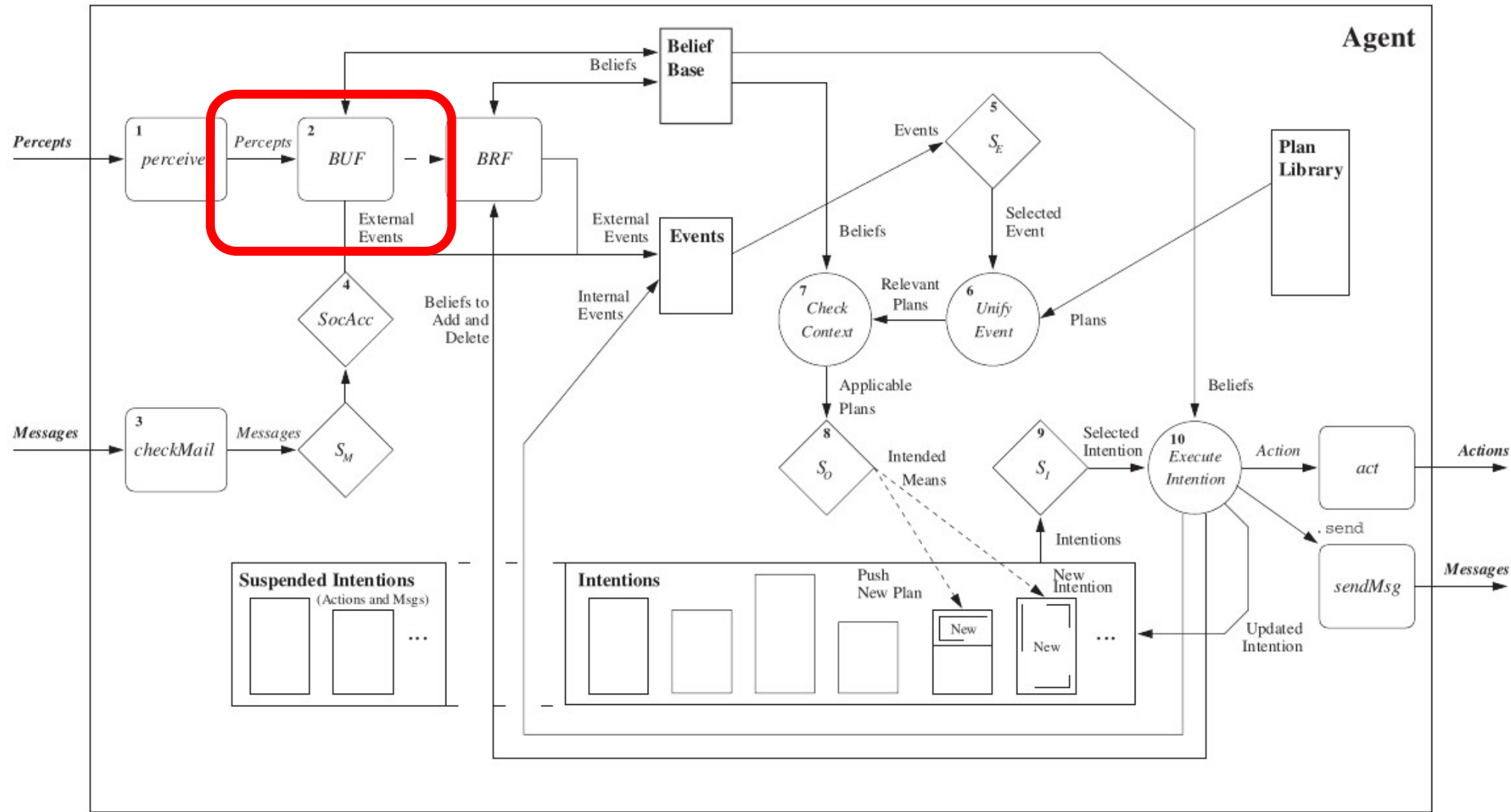
The first thing an agent does within a reasoning cycle is to sense the environment so as to update its beliefs about the state of environment [1].



[1] R. H. Bordini, J. F. Hübner, e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason. em Wiley Series in Agent Technology. Wiley, 2007.

2. Updating the Belief Base

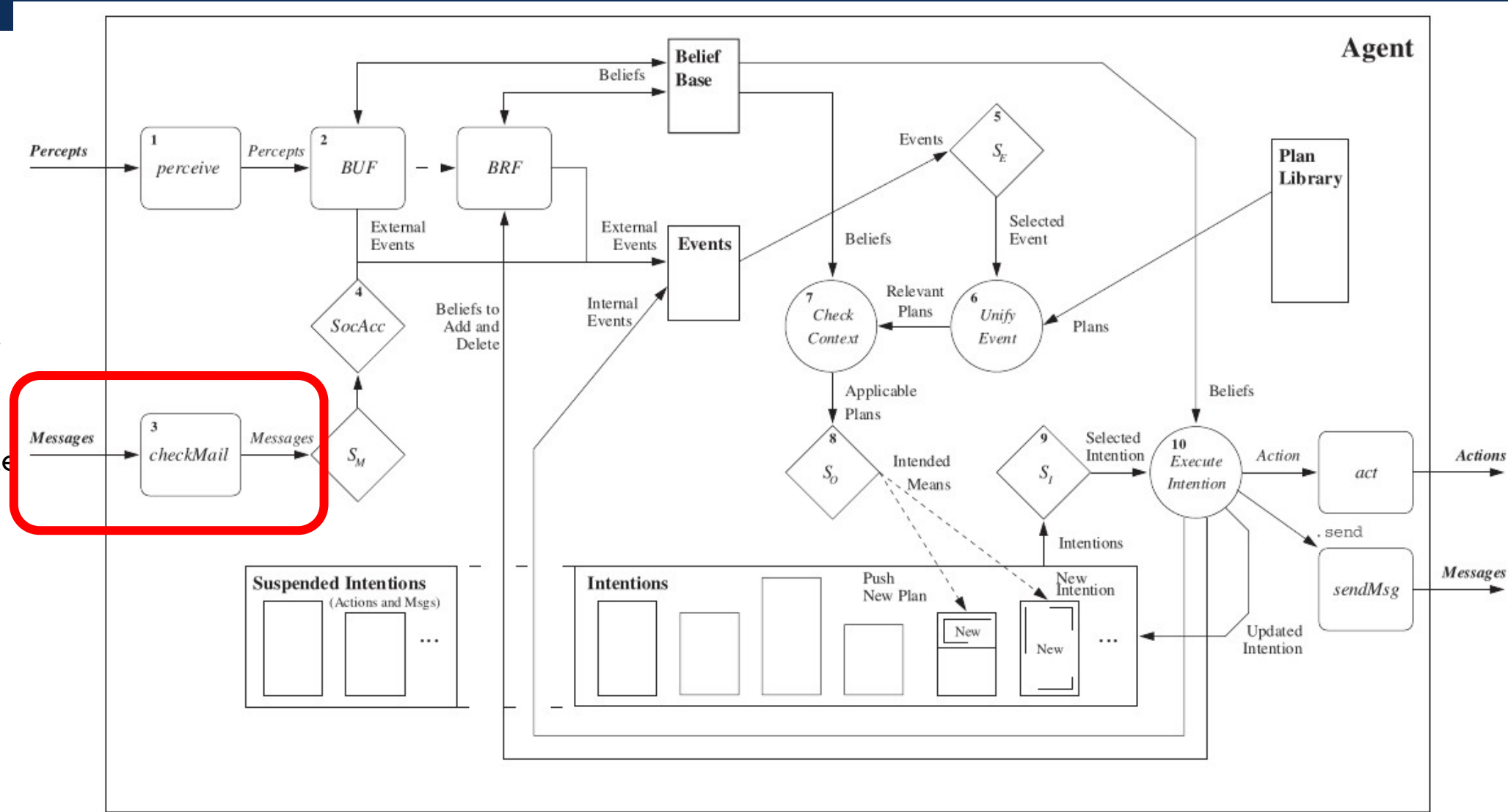
Once the list of percepts has been obtained, the belief base needs to be updated to reflect perceived changes to the environment [1].



[1] R. H. Bordini, J. F. Hübner, e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason. em Wiley Series in Agent Technology. Wiley, 2007.

3. Receiving Communication from Other Agents

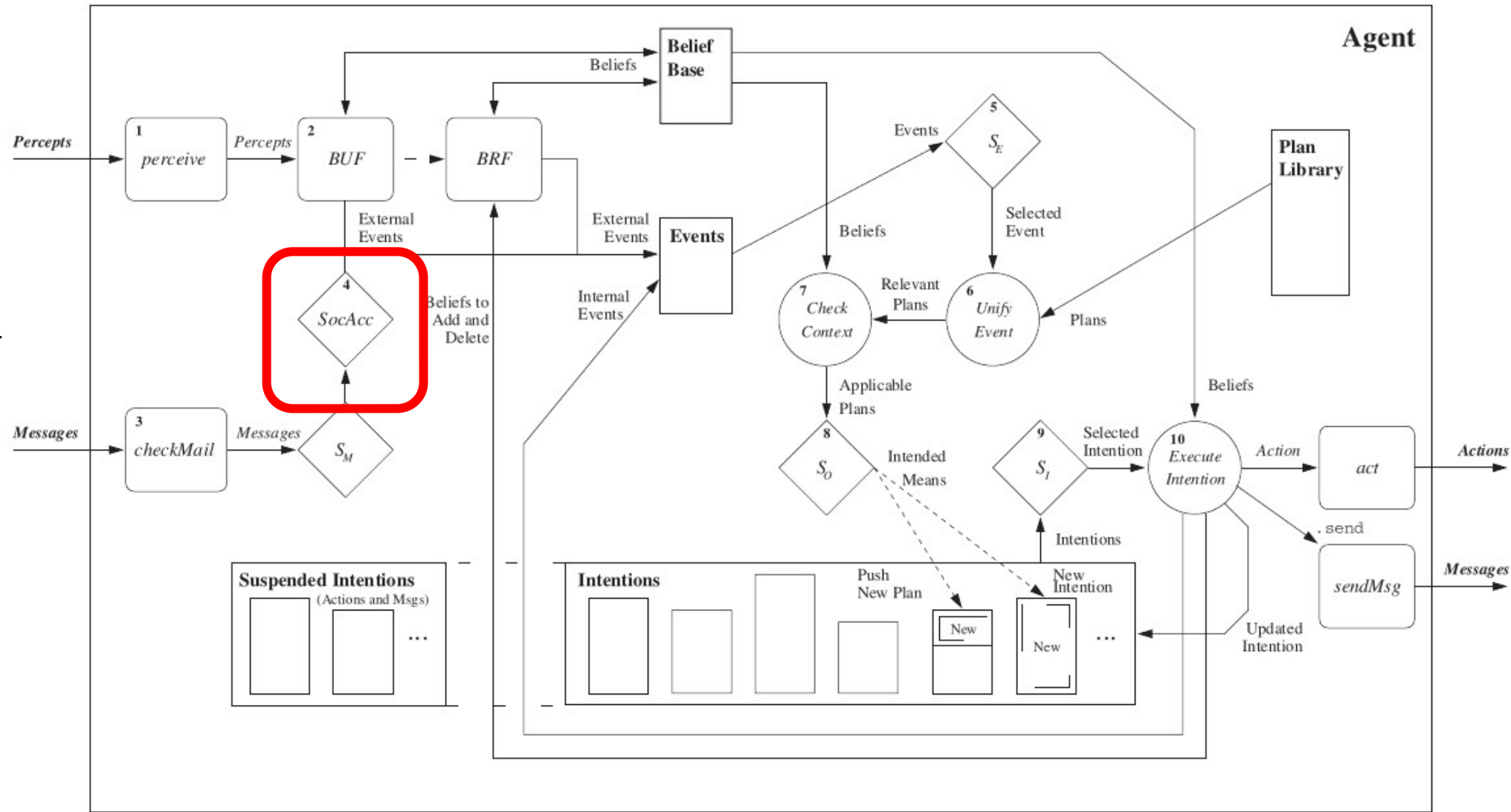
Other important sources of information for an agent in a multi-agent system are other agents in the same system [1].



[1] R. H. Bordini, J. F. Hübner, e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason. em Wiley Series in Agent Technology. Wiley, 2007.

4. Selecting 'Socially Acceptable' Messages

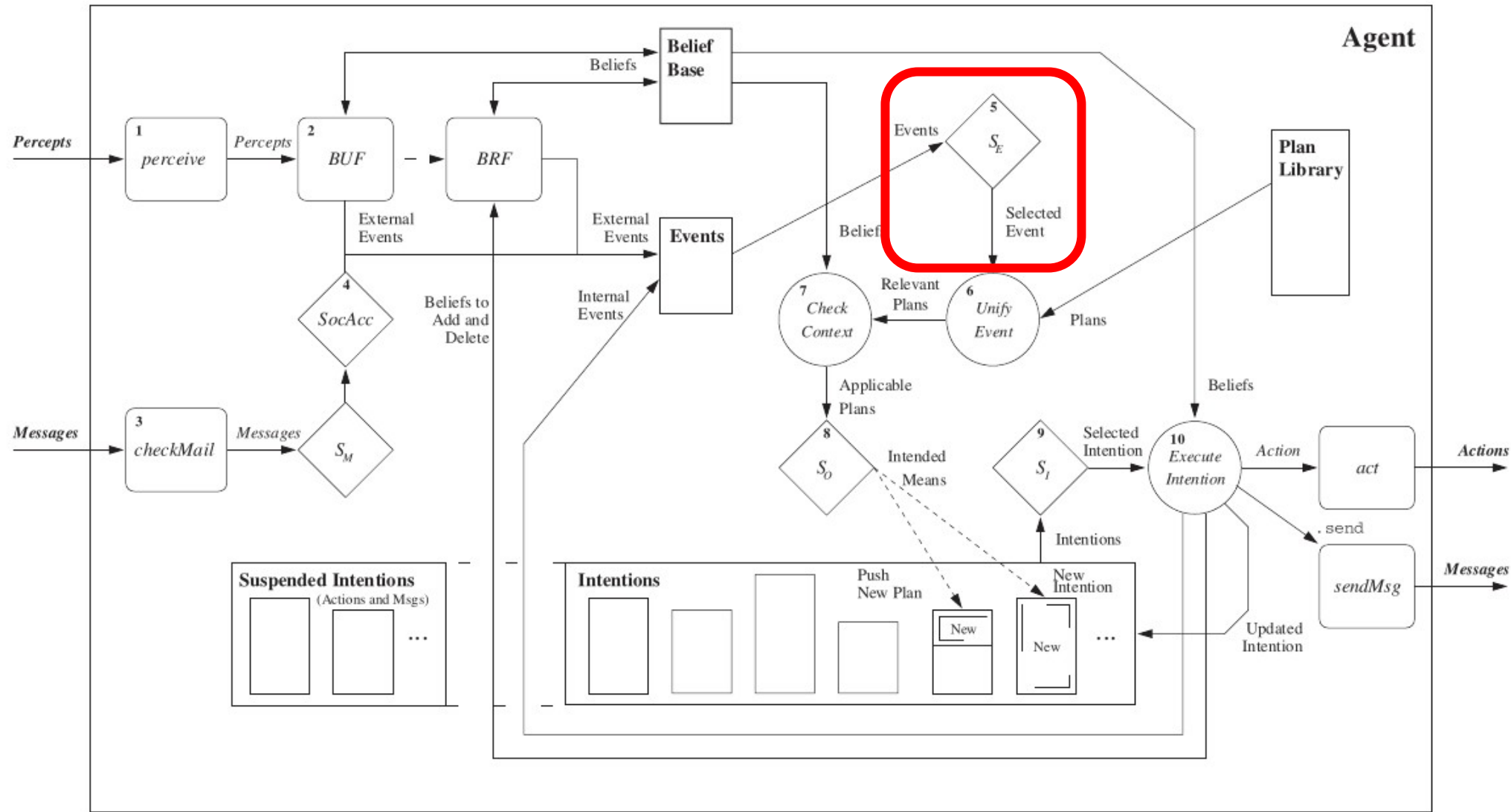
Before messages are processed, they go through a selection process to determine whether they can be accepted by the agent or not [1].



[1] R. H. Bordini, J. F. Hübner, e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason. em Wiley Series in Agent Technology. Wiley, 2007.

5. Selecting an Event

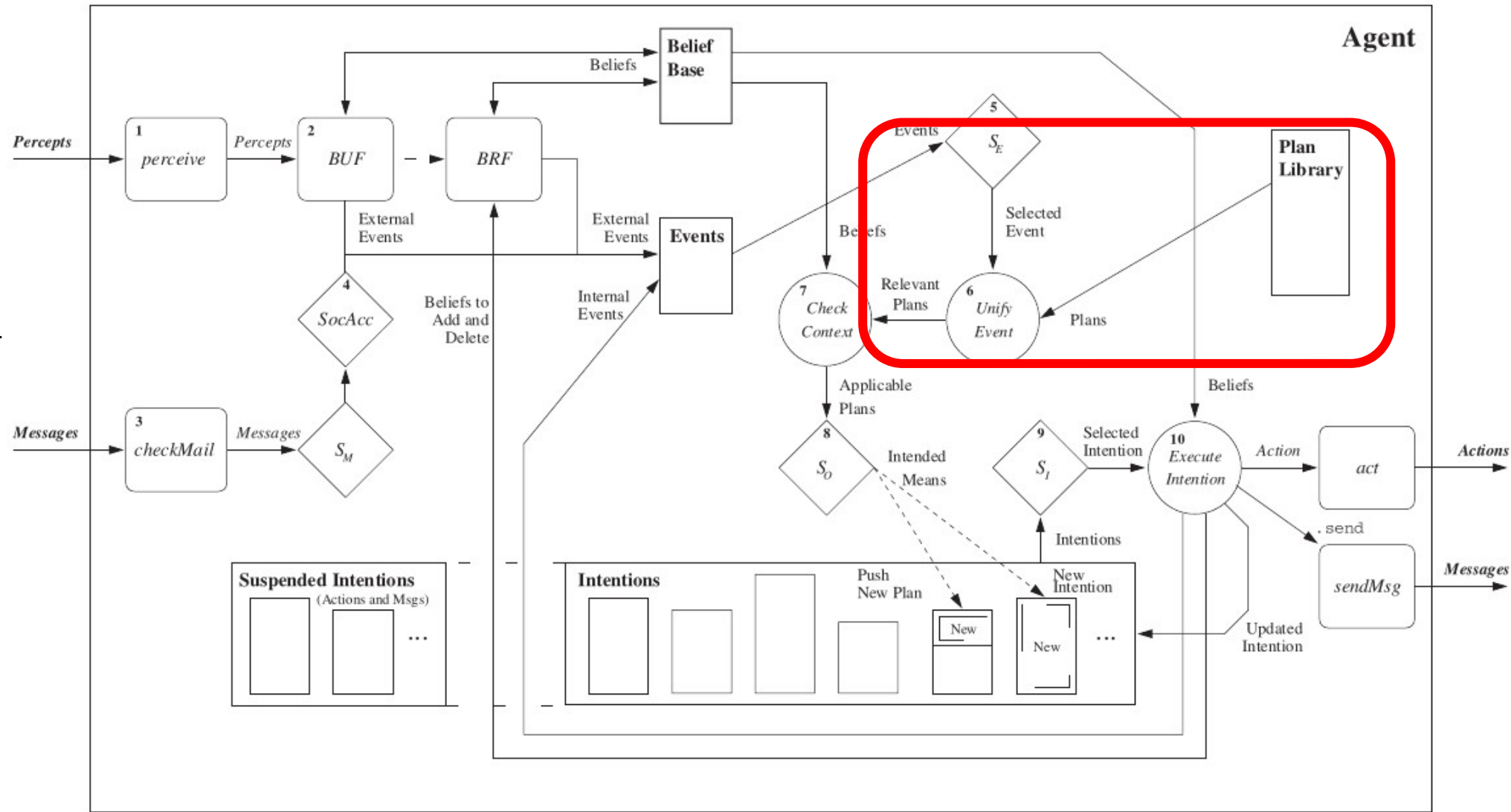
Practical BDI agents operate by continuously handling events, which represent either perceived changes in the environment or changes in the agent's own goals [1].



[1] R. H. Bordini, J. F. Hübner, e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason. em Wiley Series in Agent Technology. Wiley, 2007.

6. Retrieving all Relevant Plans

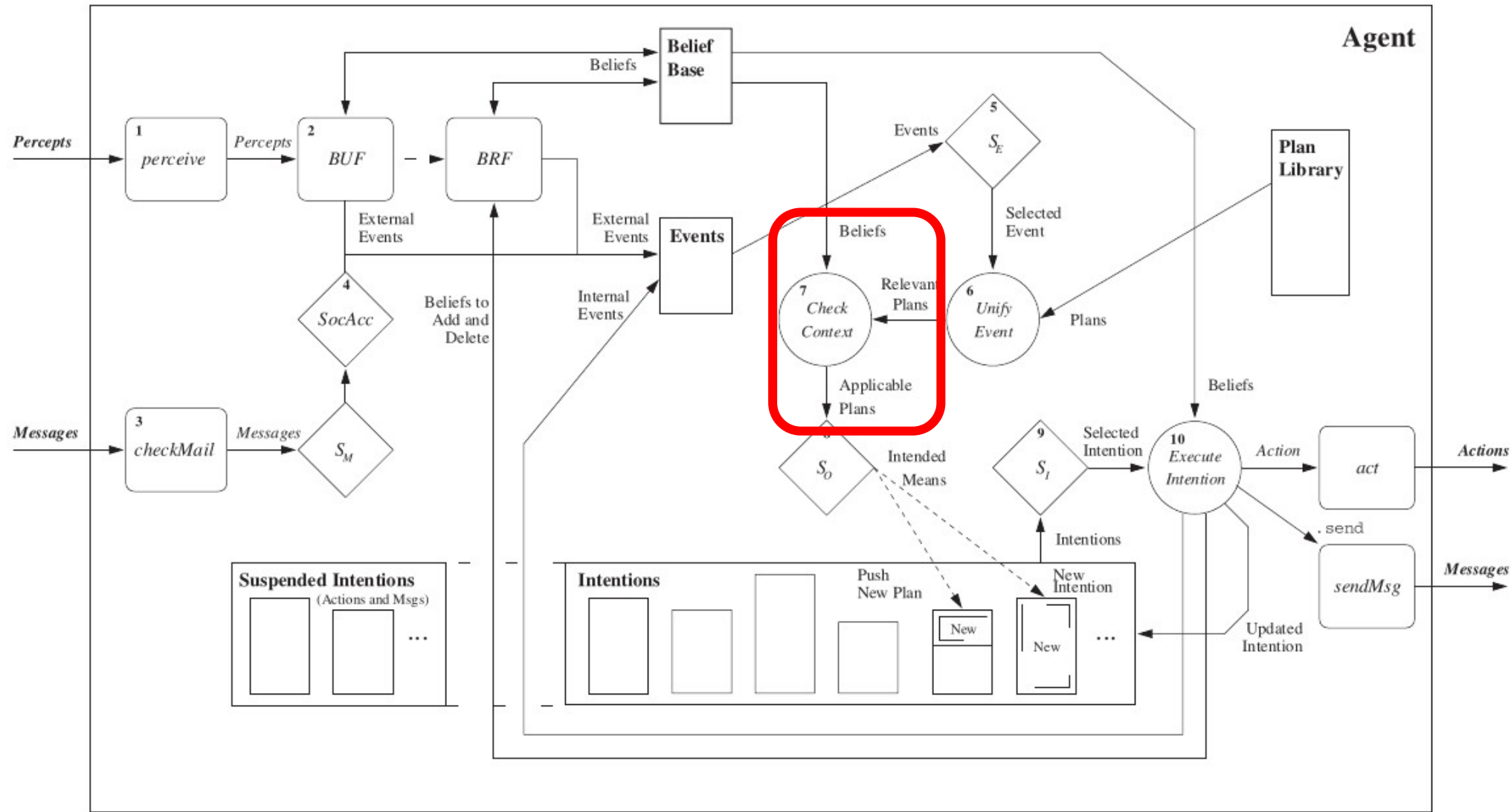
Now that we have a selected event, we need to find a plan that will allow the agent to act so as to handle that event [1].



[1] R. H. Bordini, J. F. Hübner, e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason. em Wiley Series in Agent Technology. Wiley, 2007.

7. Determining the Applicable Plans

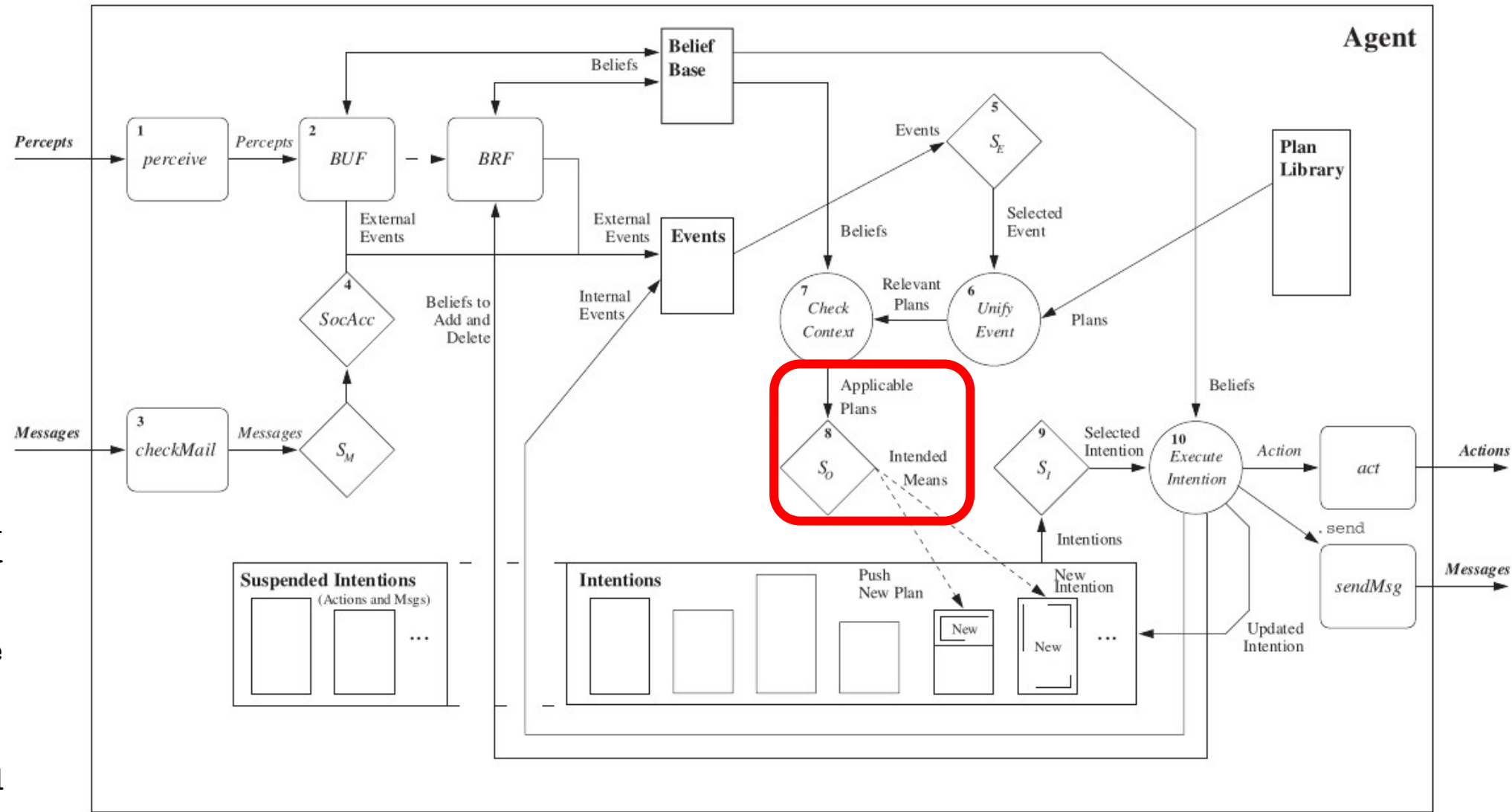
Plans have a context part which tells us whether a plan can be used at a particular moment in time, given the information the agent currently has [1].



[1] R. H. Bordini, J. F. Hübner, e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason. em Wiley Series in Agent Technology. Wiley, 2007.

8. Selecting One Applicable Plan

Given the agent's know-how as expressed by its plan library, and its current information about the world as expressed by its belief base, we have just determined that all the plans currently in the set of applicable plan are suitable alternatives to handle the selected event.

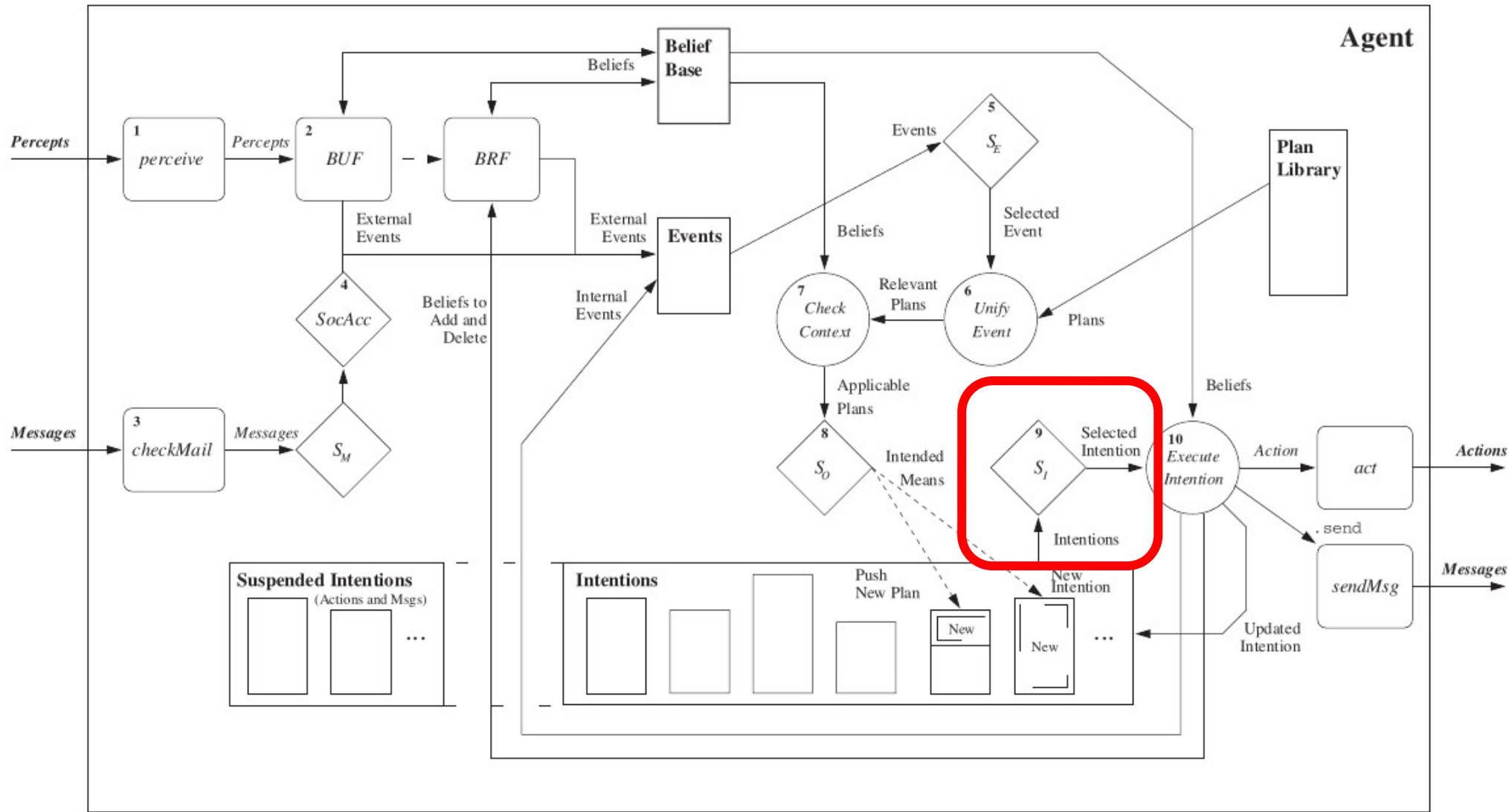


[1] R. Alami, J. F. Hübner, e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason. em Wiley Series in Agent Technology. Wiley, 2007.

9. Selecting an Intention for Further Execution

Assuming we had an event to handle, so far in the reasoning cycle we acquired a new intended means.

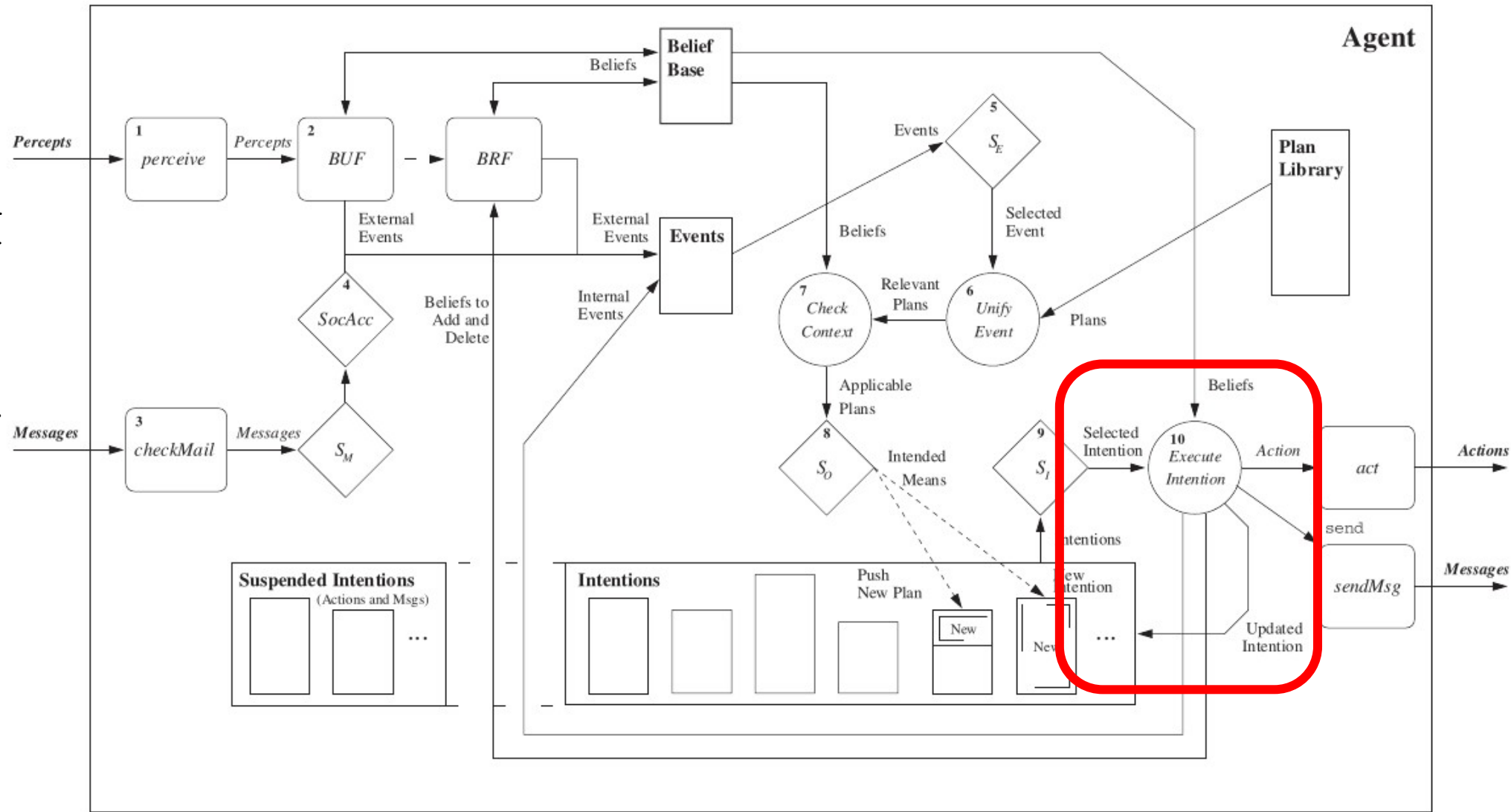
Typically an agent has more than one intention in the set of intentions, each representing a different focus of attention [1].



[1] R. H. Bordini, J. F. Hübner, e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason. em Wiley Series in Agent Technology. Wiley, 2007.

10. Executing One Step of an Intention

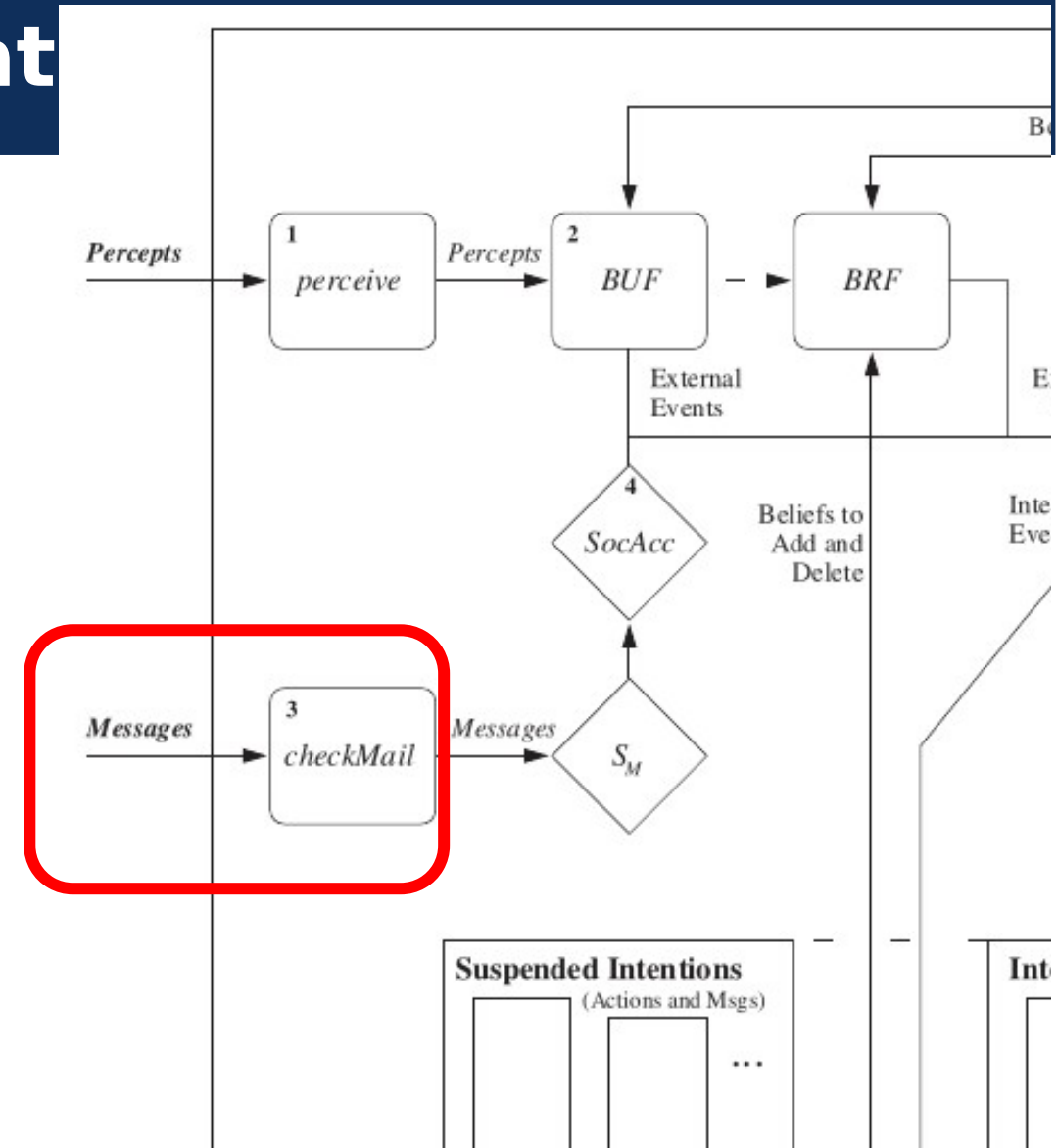
There are three main things that an agent does in every reasoning cycle: update its information about the world and other agents, handle one of the possibly generated events and then, of course, act upon the environment [1].



[1] R. H. Bordini, J. F. Hübner, e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason. em Wiley Series in Agent Technology. Wiley, 2007.

Jason Framework: Communicat

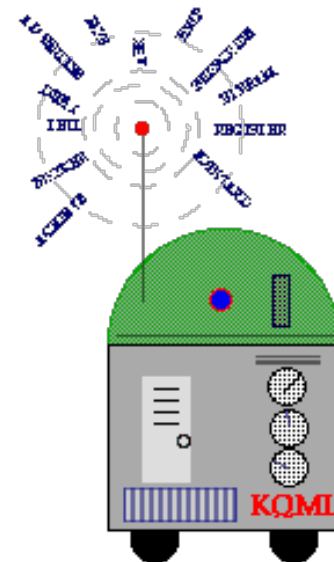
No início de cada ciclo de raciocínio, o agente verifica mensagens que ele possa ter recebido de outros agentes.



[1] R. H. Bordini, J. F. Hübner, e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason. em Wiley Series in Agent Technology. Wiley, 2007.

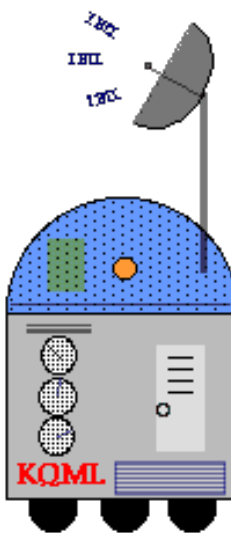
Jason Framework: Communication

Baseada na teoria dos Atos de Fala [1] e KQML [2] (Knowledge Query and Manipulation Language), um protocolo de comunicação para sistemas baseados em conhecimento.



KQML

Knowledge Query and Manipulation Language



- [1] M. Bierwisch, "Semantic Structure and Illocutionary Force", em *Speech Act Theory and Pragmatics*, J. R. Searle, F. Kiefer, e M. Bierwisch, Orgs., Dordrecht: Springer Netherlands, 1980, p. 1–35. doi: [10.1007/978-94-009-8964-1_1](https://doi.org/10.1007/978-94-009-8964-1_1).
[2] T. Finin, R. Fritzson, D. McKay, e R. McEntire, "KQML as an agent communication language", em *Proceedings of the third international conference on Information and knowledge management - CIKM '94*, Gaithersburg, Maryland, United States: ACM Press, 1994, p. 456–463. doi: [10.1145/191246.191322](https://doi.org/10.1145/191246.191322).

Jason Framework: Performatives

The performatives that are currently available for agent communication in Jason are largely inspired by KQML. We also include some new performatives, related to plan exchange rather than communication about propositions. The available performatives are briefly described below, where *s* denotes the agent that sends the message, and *r* denotes the agent that receives the message [1].

PERFORMATIVE	DESCRIPTION
tell	<i>s</i> intends <i>r</i> to believe (that <i>s</i> believes) the sentence in the message's content to be true;
untell	<i>s</i> intends <i>r</i> not to believe (that <i>s</i> believes) the sentence in the message's content to be true;
achieve	<i>s</i> requests that <i>r</i> try to achieve a state of the world where the message content is true;
unachieve	<i>s</i> requests that <i>r</i> try to drop the intention of achieving a state of the world where the message content is true;
tellHow	<i>s</i> informs <i>r</i> of a plan;
untellHow	<i>s</i> requests that <i>r</i> disregard a certain plan (i.e., delete that plan from its plan library);
askIf	<i>s</i> wants to know if the content of the message is true for <i>r</i> ;
askAll	<i>s</i> wants all of <i>r</i> 's answers to a question;
askHow	<i>s</i> wants all of <i>r</i> 's plans for a triggering event;

[1] R. H. Bordini e J. F. Hübner, "BDI Agent Programming in AgentSpeak Using Jason", em Computational Logic in Multi-Agent Systems, F. Toni e P. Torroni, Orgs., em Lecture Notes in Computer Science, vol. 3900. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, p. 143–164. doi: [10.1007/11750734_9](https://doi.org/10.1007/11750734_9).

Jason Framework: Message Structure

.send(receiver, ilf, message, answer, timeout)
.broadcast(ilf, message)

- **receiver**
 - Nome do agente destinatário da mensagem (Ou lista de destinatários)
- **ilf**
 - Força ilocucionária do ato de fala (KQML)
- **message**
 - Conteúdo da mensagem
- **answer**
 - Um termo qualquer que irá armazenar a resposta (campo opcional)
- **timeout**
 - Tempo limite em milisegundos para receber uma resposta (campo opcional)

Illocutionary Forces: tell

Illocutionary Forces: tell

O agente remetente pretende que o receptor **acredite** que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.

Illocutionary Forces: tell

O agente remetente pretende que o receptor **acredite** que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.

**agent
Kate**

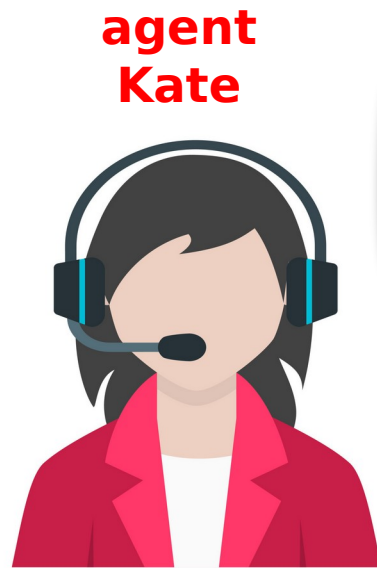


**agent
Bob**



Illocutionary Forces: tell

O agente remetente pretende que o receptor **acredite** que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.



Illocutionary Forces: tell

O agente remetente pretende que o receptor **acredite** que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.



Illocutionary Forces: tell

O agente remetente pretende que o receptor **acredite** que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.

**agent
Kate**



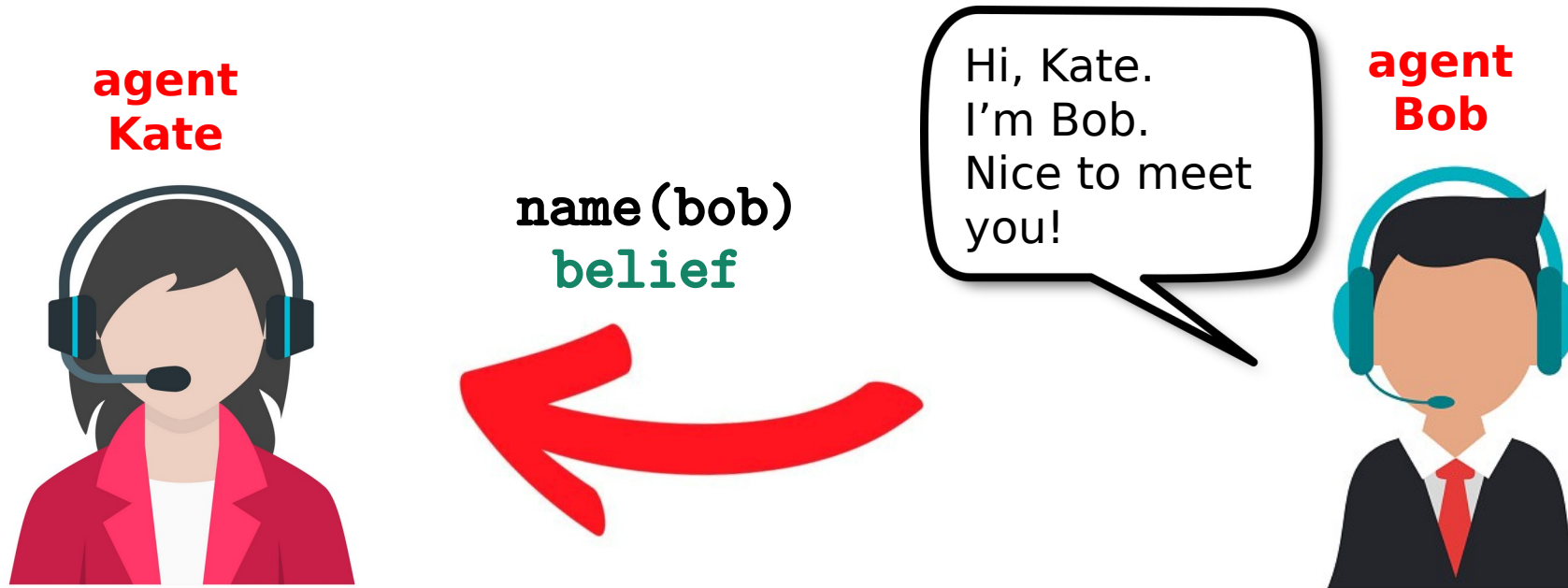
Hi, Kate.
I'm Bob.
Nice to meet
you!

**agent
Bob**



Illocutionary Forces: tell

O agente remetente pretende que o receptor **acredite** que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.



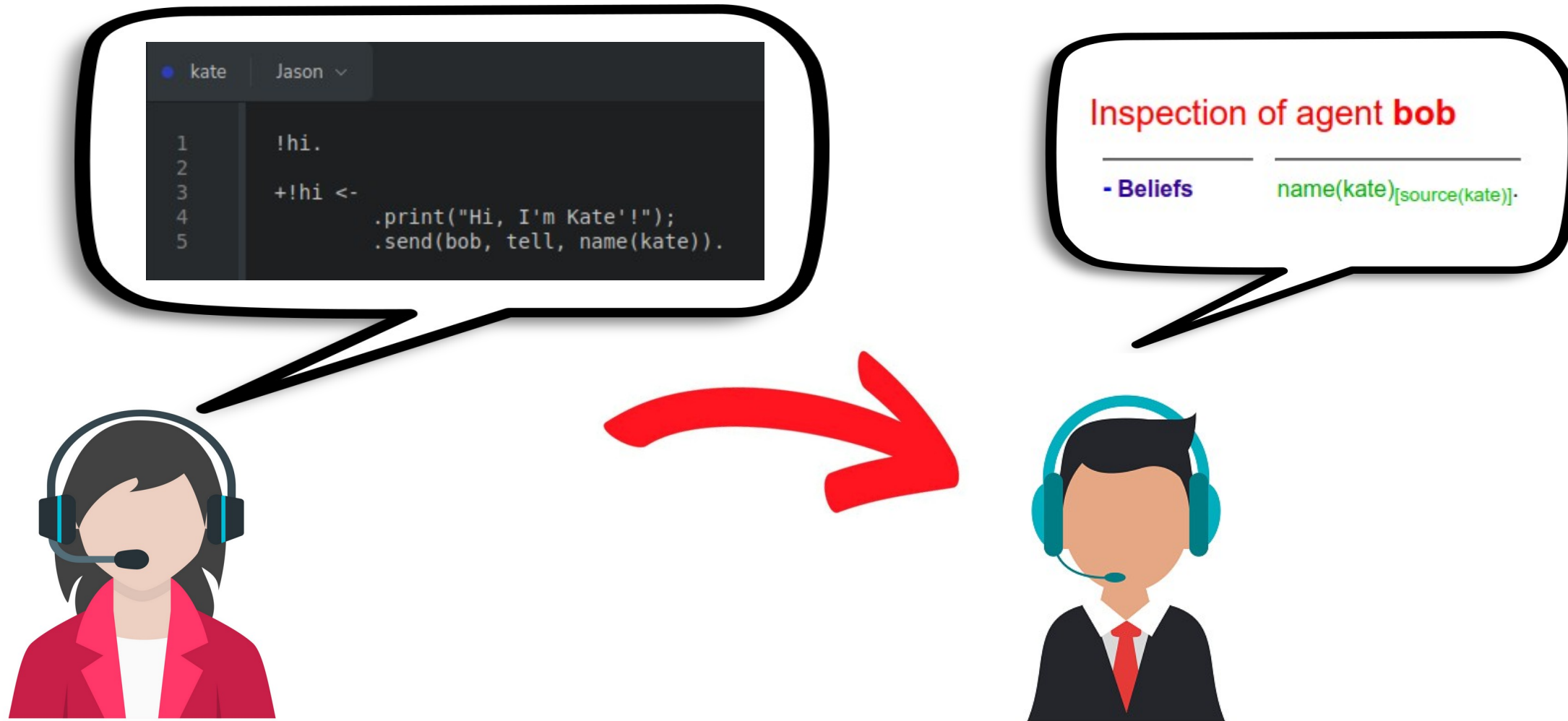
Illocutionary Forces: tell



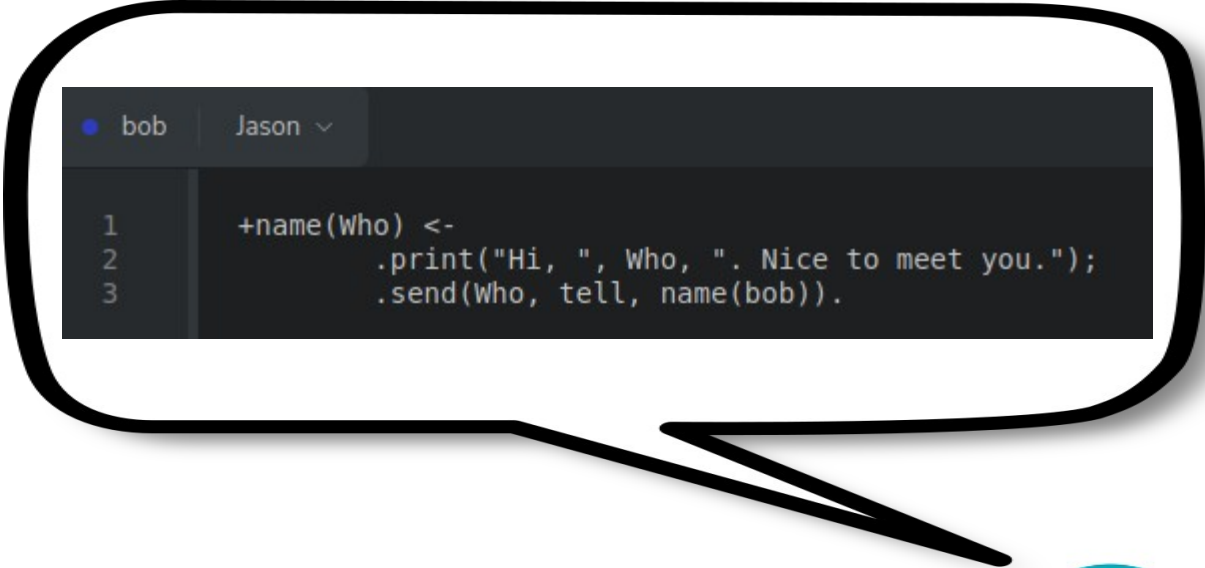
Illocutionary Forces: tell



Illocutionary Forces: tell



Illocutionary Forces: tell



```
1 +name(Who) <-  
2 .print("Hi, ", Who, ". Nice to meet you.");  
3 .send(Who, tell, name(bob)).
```



Illocutionary Forces: tell



Illocutionary Forces: tell

Inspection of agent **kate**

- Beliefs

`name(bob)[source(bob)].`

```
bob Jason v
1      +name(Who) <-
2      .print("Hi, ", Who, ". Nice to meet you.");
3      .send(Who, tell, name(bob)).
```



Illocutionary Forces: untell

Illocutionary Forces: untell

O agente remetente pretende que o receptor não acredite que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.

Illocutionary Forces: untell

O agente remetente pretende que o receptor não acredite que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.

**agent
Kate**

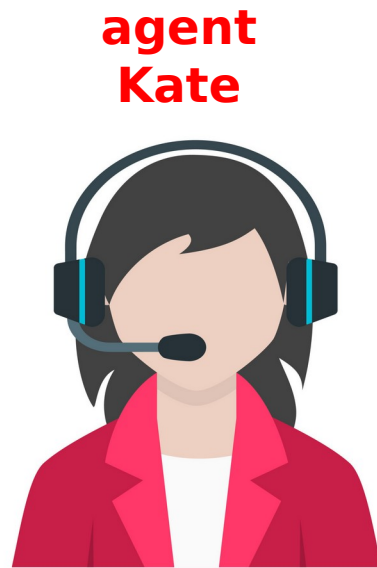


**agent
Bob**



Illocutionary Forces: untell

O agente remetente pretende que o receptor não acredite que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.



Are you free
this weekend?



Illocutionary Forces: untell

O agente remetente pretende que o receptor não acredite que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.



Illocutionary Forces: untell

O agente remetente pretende que o receptor não acredite que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.



Illocutionary Forces: untell

O agente remetente pretende que o receptor não acredite que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.



Illocutionary Forces: untell

O agente remetente pretende que o receptor não acredite que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.



Illocutionary Forces: untell

O agente remetente pretende que o receptor não acredite que o conteúdo enviado é verdadeiro de acordo com as crenças do remetente.

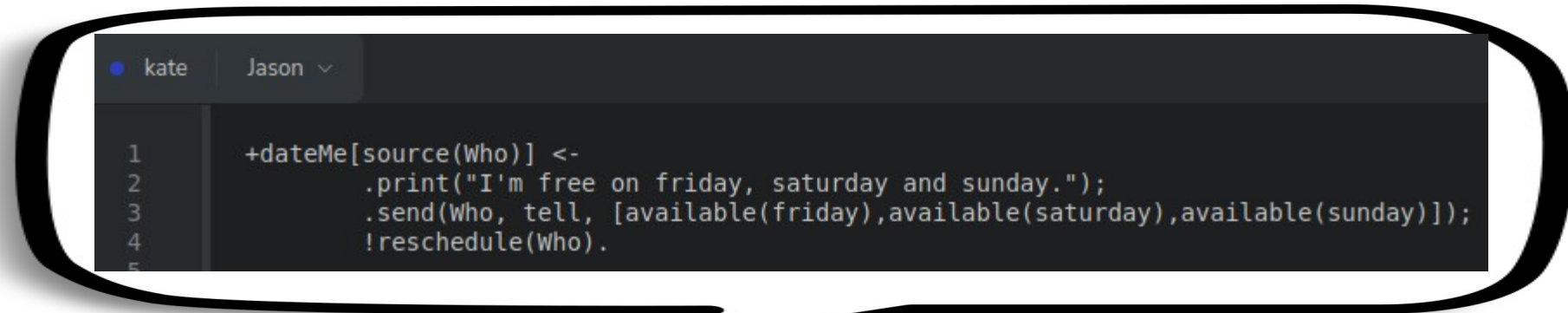


Illocutionary Forces: untell

```
bob Jason v
1 !hi.
2
3 +!hi <-
4 .print("Are you free this weekend?");
5 .send(kate, tell, dateMe).
```



Illocutionary Forces: untell



```
1 +dateMe[source(Who)] <-  
2 .print("I'm free on friday, saturday and sunday.");  
3 .send(Who, tell, [available(friday),available(saturday),available(sunday)]);  
4 !reschedule(Who).  
5
```



Illocutionary Forces: untell



Inspection of agent **bob**

- Beliefs

`available(sunday)[source(kate)]·`
`available(saturday)[source(kate)]·`
`available(friday)[source(kate)]·`

Illocutionary Forces: untell

```
6   +!reschedule(Who) <-  
7       .print("Sorry ", Who, ". I'm not free on sundays anymore.");  
8       .wait(1000);  
9       .send(Who, untell, available(sunday)).  
10
```



Illocutionary Forces: untell



Inspection of agent **bob**

- Beliefs

`available(saturday)[source(kate)]`

`available(friday)[source(kate)]`

Illocutionary Forces: achieve

Illocutionary Forces: achieve

O agente remetente pede que o receptor tente atingir um objetivo de estado verdadeiro de acordo com conteúdo enviado.

Illocutionary Forces: achieve

O agente remetente pede que o receptor tente atingir um objetivo de estado verdadeiro de acordo com conteúdo enviado.

**agent
Kate**



**agent
Bob**



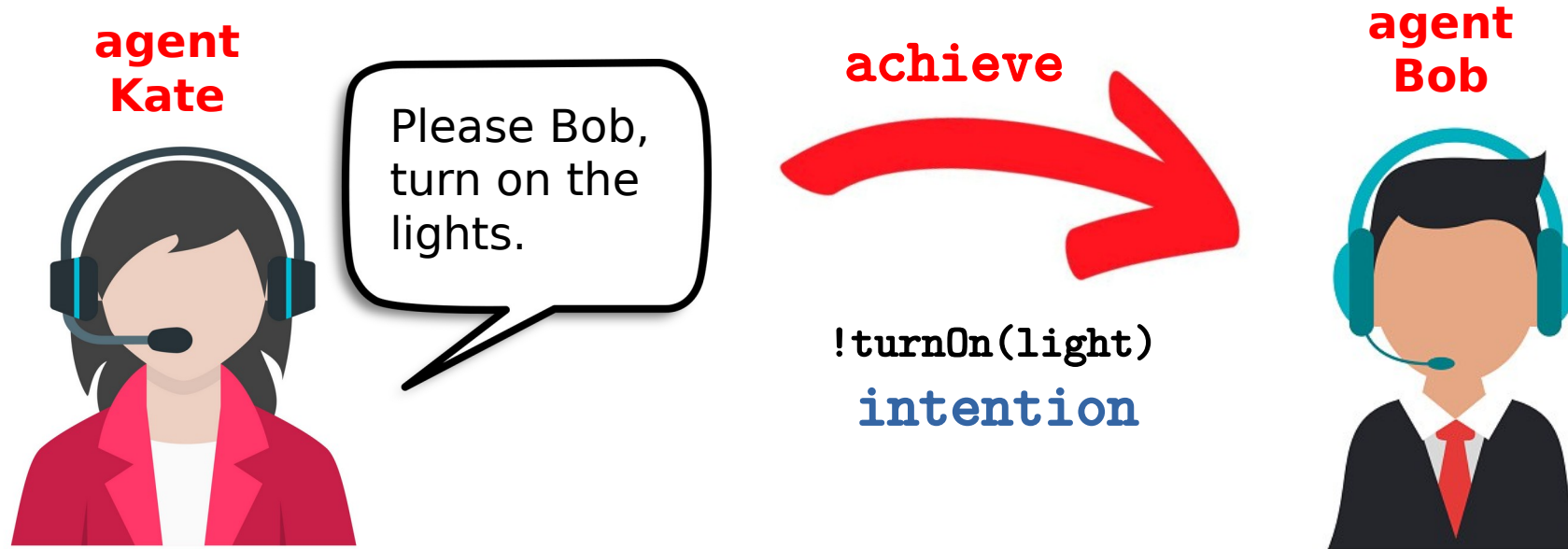
Illocutionary Forces: achieve

O agente remetente pede que o receptor tente atingir um objetivo de estado verdadeiro de acordo com conteúdo enviado.



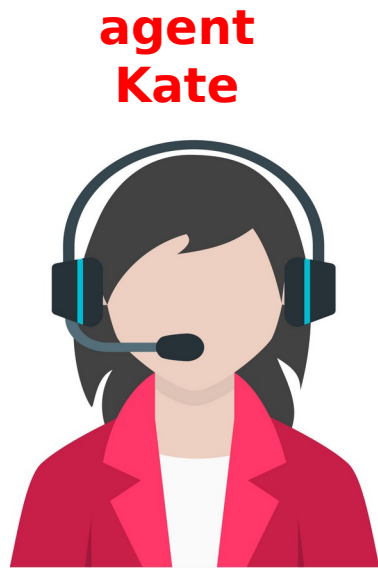
Illocutionary Forces: achieve

O agente remetente pede que o receptor tente atingir um objetivo de estado verdadeiro de acordo com conteúdo enviado.



Illocutionary Forces: achieve

O agente remetente pede que o receptor tente atingir um objetivo de estado verdadeiro de acordo com conteúdo enviado.



Illocutionary Forces: achieve



Illocutionary Forces: achieve



Illocutionary Forces: unachieve

Illocutionary Forces: unachieve

O agente remetente pede que o receptor deixe de tentar atingir um objetivo de estado verdadeiro de acordo com conteúdo enviado.

Illocutionary Forces: unachieve

O agente remetente pede que o receptor deixe de tentar atingir um objetivo de estado verdadeiro de acordo com conteúdo enviado.

**agent
Kate**



**agent
Bob**



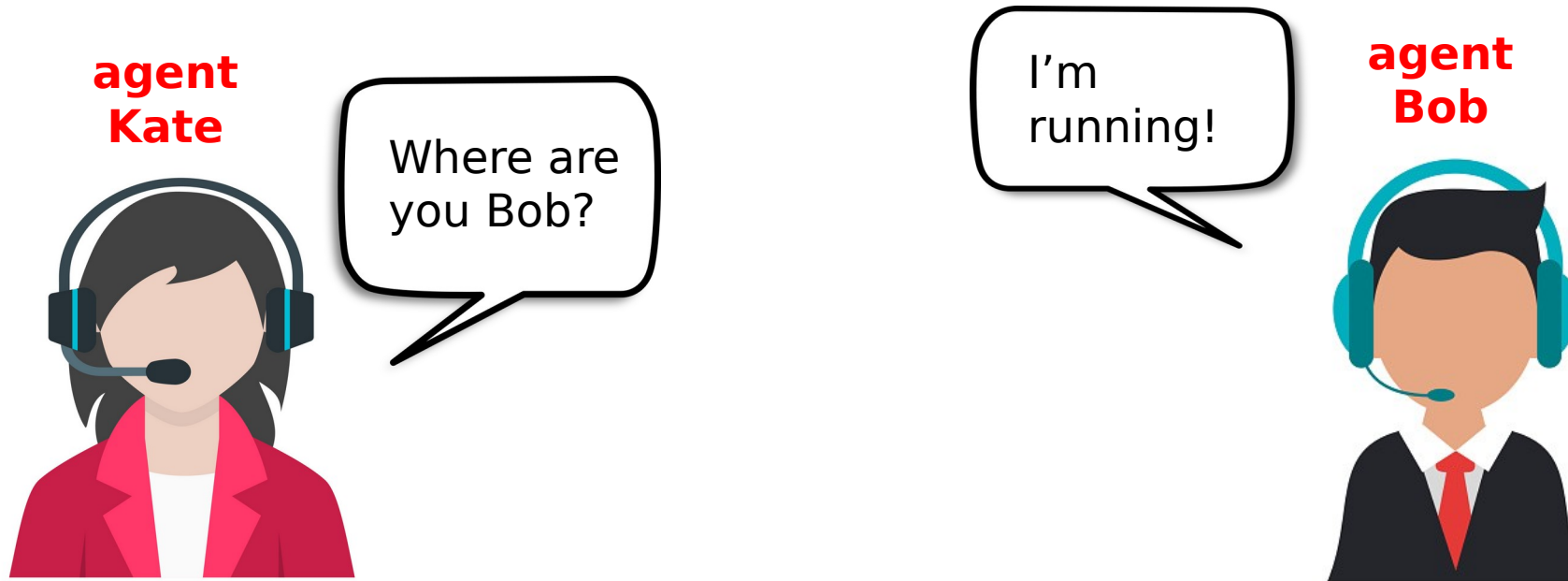
Illocutionary Forces: unachieve

O agente remetente pede que o receptor deixe de tentar atingir um objetivo de estado verdadeiro de acordo com conteúdo enviado.



Illocutionary Forces: unachieve

O agente remetente pede que o receptor deixe de tentar atingir um objetivo de estado verdadeiro de acordo com conteúdo enviado.



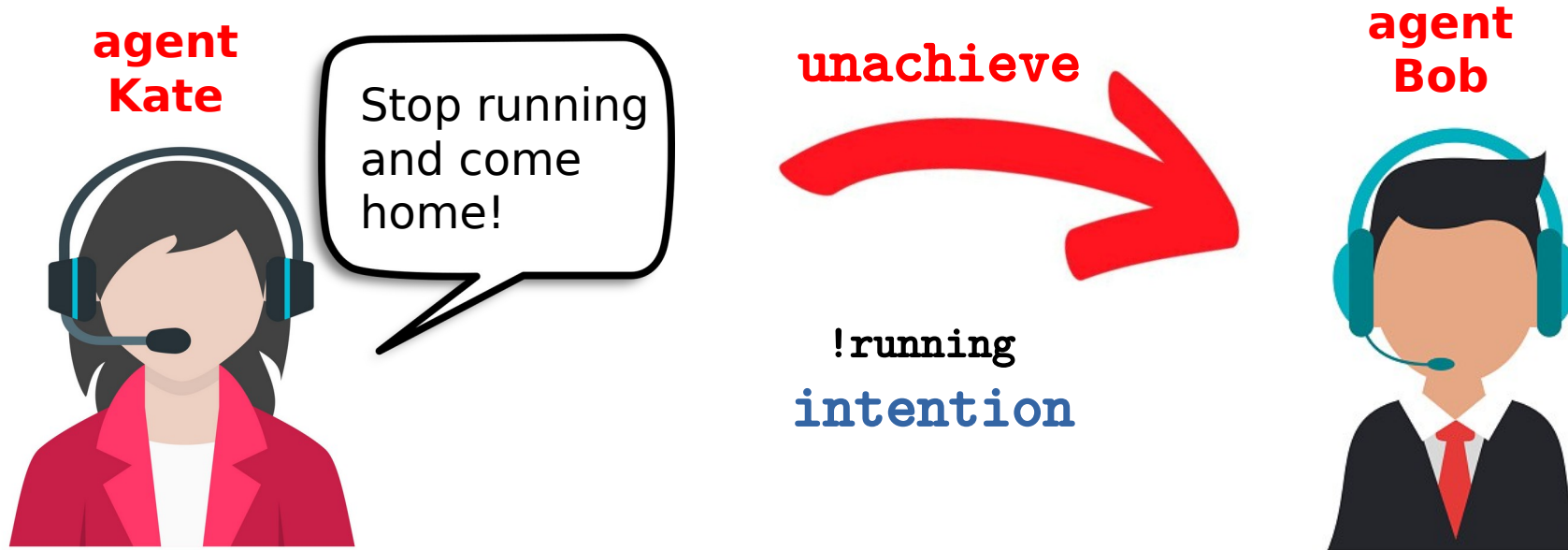
Illocutionary Forces: unachieve

O agente remetente pede que o receptor deixe de tentar atingir um objetivo de estado verdadeiro de acordo com conteúdo enviado.



Illocutionary Forces: unachieve

O agente remetente pede que o receptor deixe de tentar atingir um objetivo de estado verdadeiro de acordo com conteúdo enviado.



Illocutionary Forces: unachieve



Illocutionary Forces: unachieve



Illocutionary Forces: askOne

Illocutionary Forces: askOne

O agente remetente deseja saber se a resposta do receptor para determinada questão é verdadeira.

Illocutionary Forces: askOne

O agente remetente deseja saber se a resposta do receptor para determinada questão é verdadeira.

**agent
Kate**



**agent
Bob**



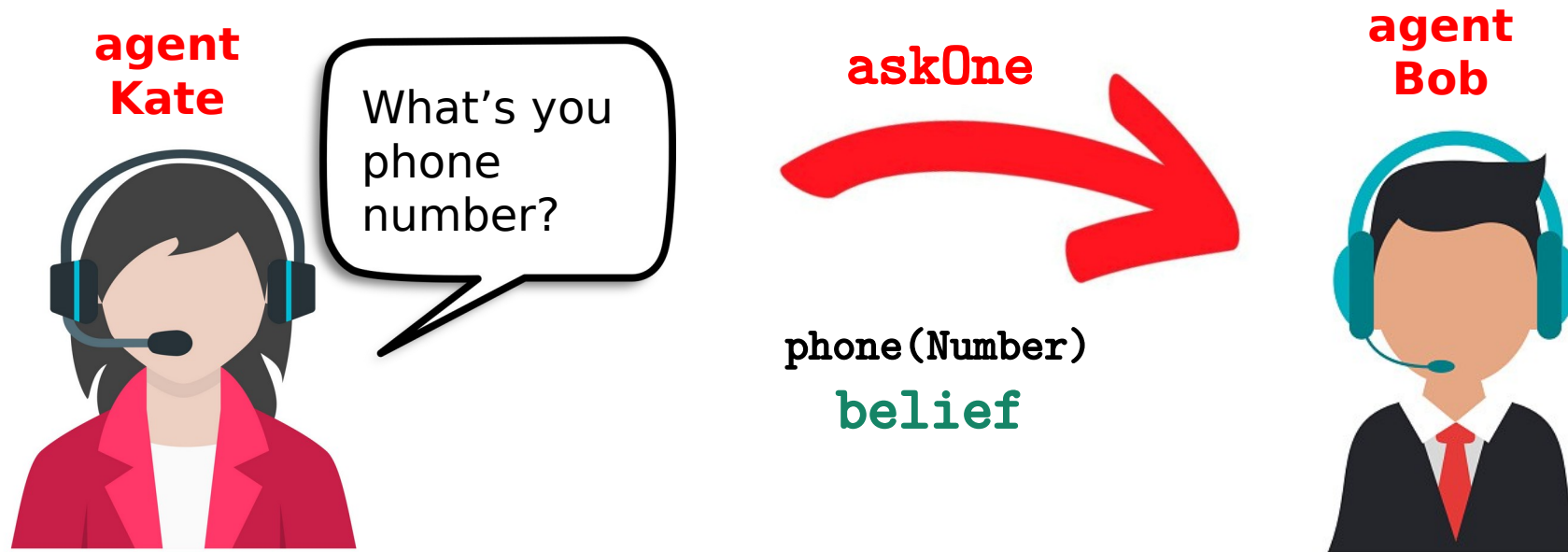
Illocutionary Forces: askOne

O agente remetente deseja saber se a resposta do receptor para determinada questão é verdadeira.



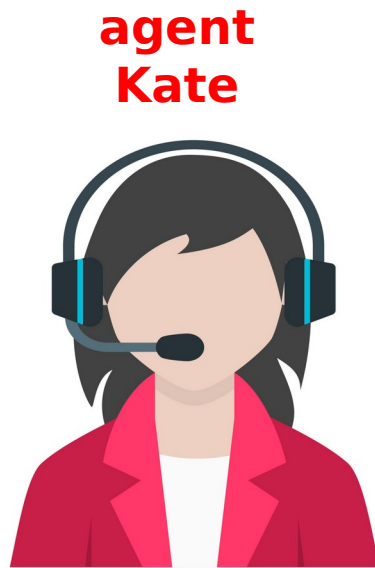
Illocutionary Forces: askOne

O agente remetente deseja saber se a resposta do receptor para determinada questão é verdadeira.



Illocutionary Forces: askOne

O agente remetente deseja saber se a resposta do receptor para determinada questão é verdadeira.

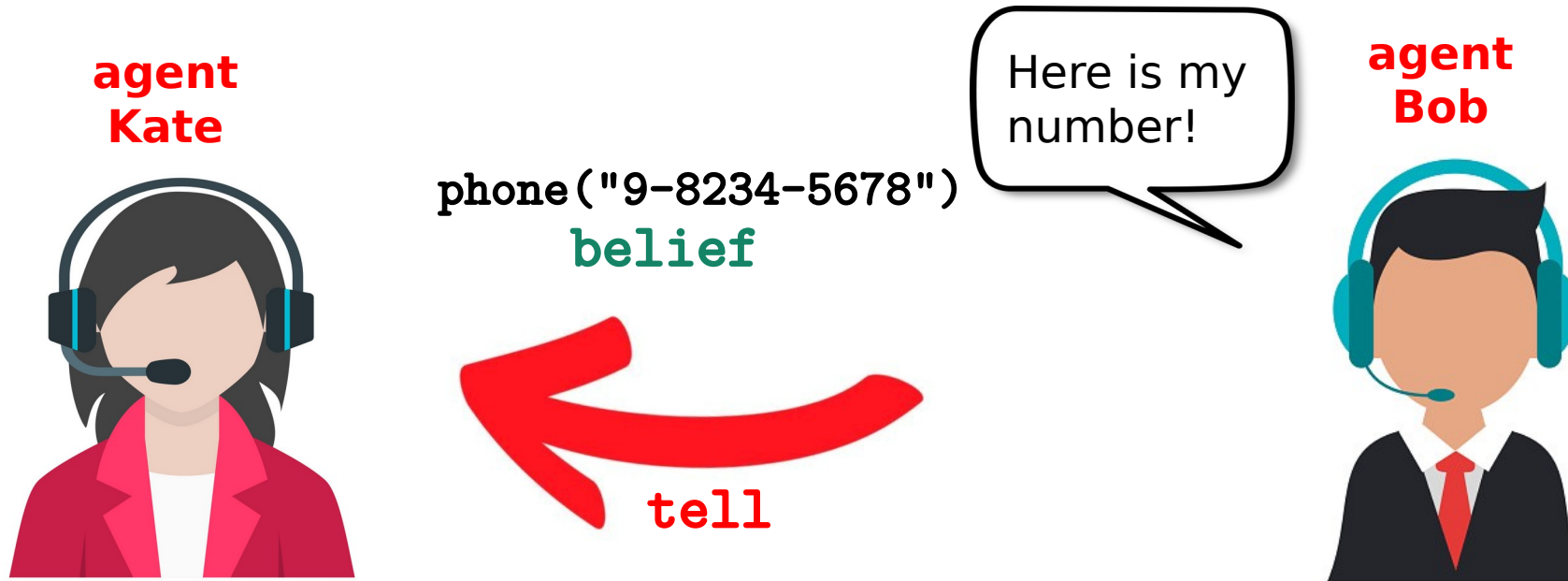


Here is my
number!



Illocutionary Forces: askOne

O agente remetente deseja saber se a resposta do receptor para determinada questão é verdadeira.



Illocutionary Forces: askOne



Illocutionary Forces: askOne

Inspection of agent **bob**

- Beliefs

`phone("9-8234-5678")[source(self)].`



Illocutionary Forces: askOne



Illocutionary Forces: askAll

Illocutionary Forces: askAll

0 agente remetente deseja saber todas as repostas do receptor sobre uma questão.

Illocutionary Forces: askAll

O agente remetente deseja saber todas as repostas do receptor sobre uma questão.

**agent
Kate**



**agent
Bob**



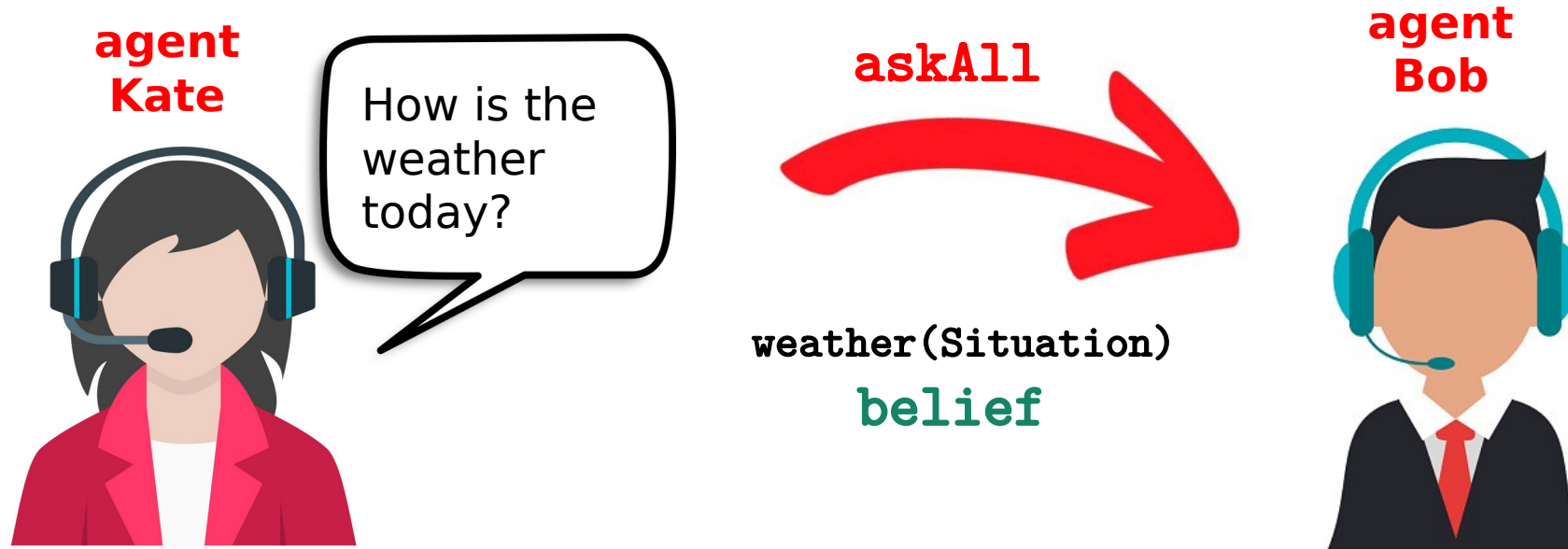
Illocutionary Forces: askAll

O agente remetente deseja saber todas as repostas do receptor sobre uma questão.



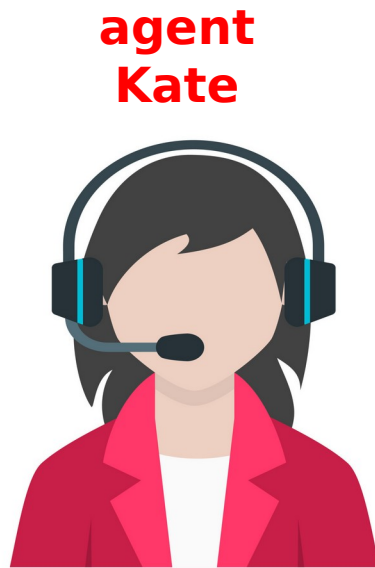
Illocutionary Forces: askAll

O agente remetente deseja saber todas as repostas do receptor sobre uma questão.



Illocutionary Forces: askAll

O agente remetente deseja saber todas as repostas do receptor sobre uma questão.



Here is the forecast!



Illocutionary Forces: askAll

O agente remetente deseja saber todas as repostas do receptor sobre uma questão.



Illocutionary Forces: askAll



Illocutionary Forces: askAll

Inspection of agent **bob**

- Beliefs

`weather(sunny)[source(self)]`

`weather(clean)[source(self)]`



Illocutionary Forces: askAll

Inspection of agent **kate**

- Beliefs

weather(clean)[source(bob)].
weather(sunny)[source(bob)].



Illocutionary Forces: tellHow

Illocutionary Forces: tellHow

O agente remetente informa ao agente receptor a implementação de um plano.

Illocutionary Forces: tellHow

O agente remetente informa ao agente receptor a implementação de um plano.

**agent
Kate**



**agent
Bob**



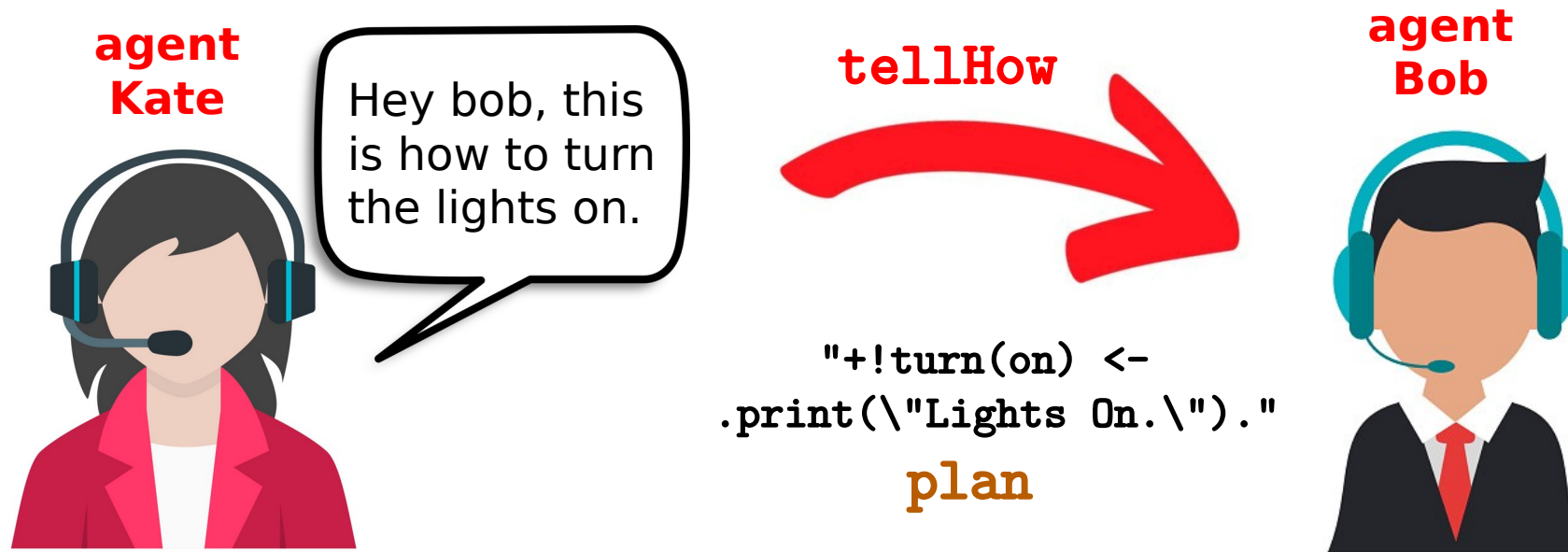
Illocutionary Forces: tellHow

O agente remetente informa ao agente receptor a implementação de um plano.



Illocutionary Forces: tellHow

O agente remetente informa ao agente receptor a implementação de um plano.



Illocutionary Forces: tellHow



Illocutionary Forces: tellHow

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.  
NOTE: Picked up JDK_JAVA_OPTIONS:  --add-opens=java.base/  
rmi/sun.rmi.transport=ALL-UNNAMED  
Jason Http Server running on http://127.0.1.1:3272  
[kate] Hey bob, this is how to turn the lights on.  
[bob] Lights On.
```



Illocutionary Forces: askHow

Illocutionary Forces: askHow

O agente remetente deseja saber todas as repostas do receptor sobre uma questão.

Illocutionary Forces: askHow

O agente remetente deseja saber todas as repostas do receptor sobre uma questão.

**agent
Kate**



**agent
Bob**



Illocutionary Forces: askHow

O agente remetente deseja saber todas as repostas do receptor sobre uma questão.

**agent
Kate**



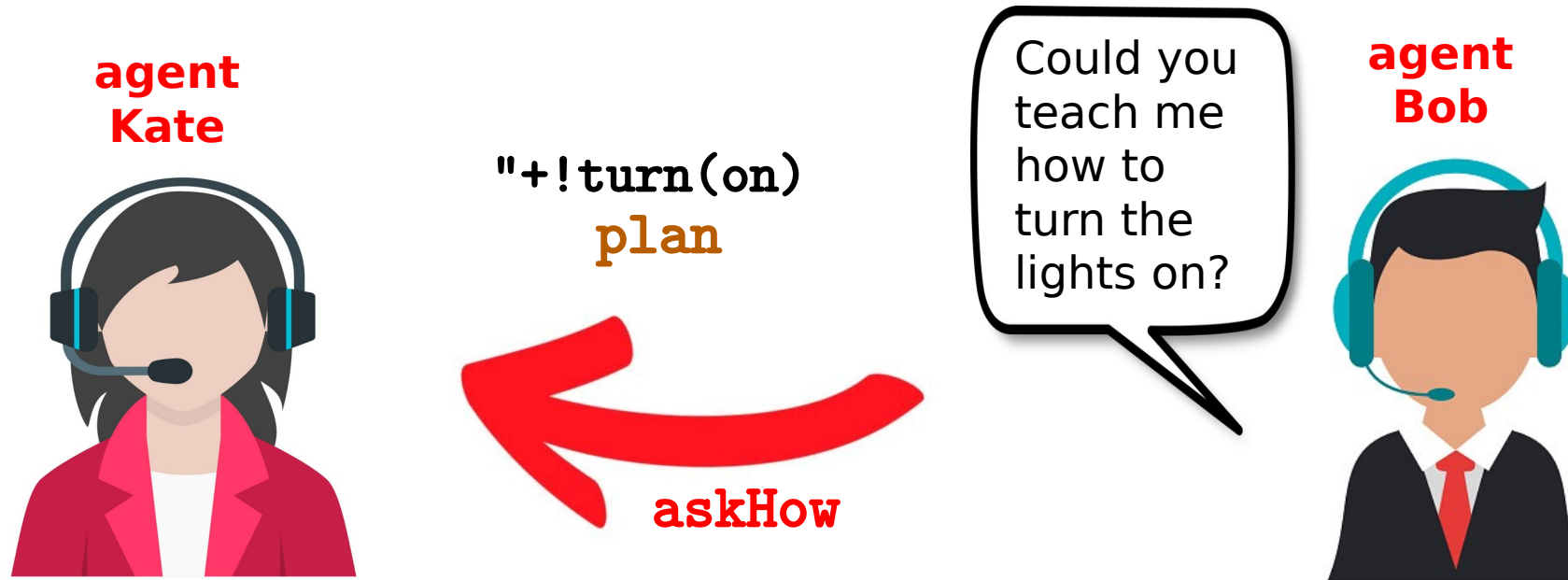
Could you
teach me
how to
turn the
lights on?

**agent
Bob**



Illocutionary Forces: askHow

O agente remetente deseja saber todas as repostas do receptor sobre uma questão.



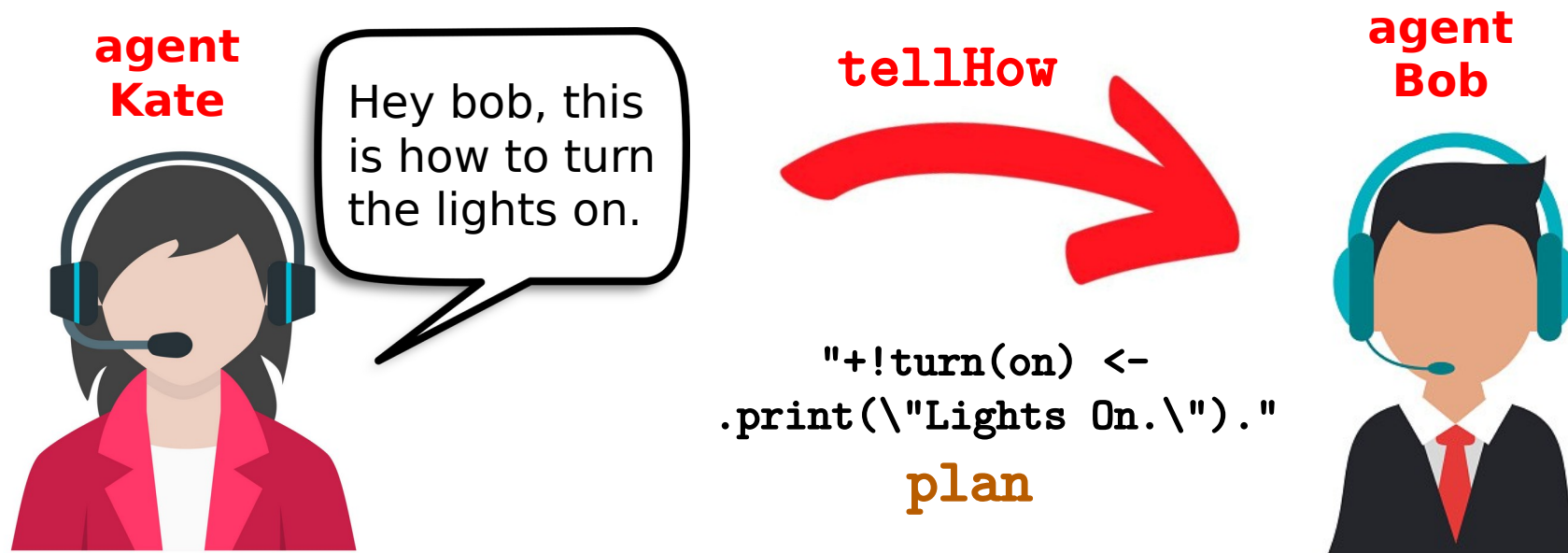
Illocutionary Forces: askHow

O agente remetente deseja saber todas as repostas do receptor sobre uma questão.

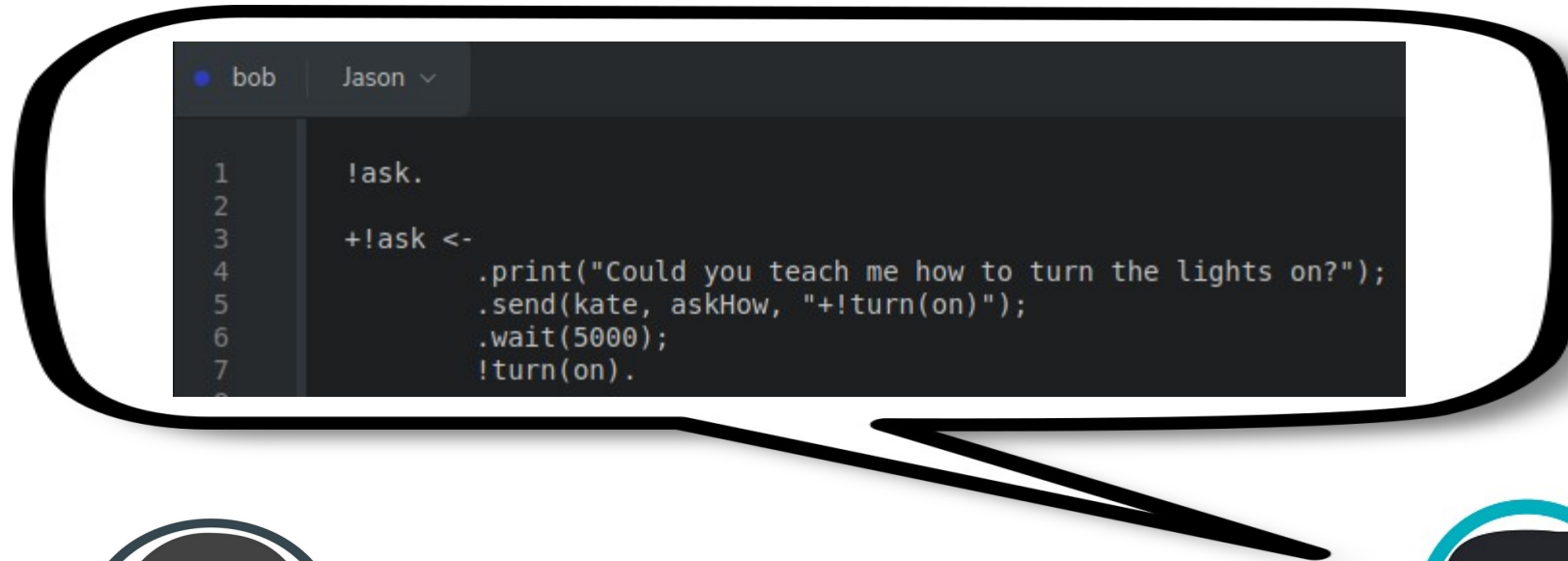


Illocutionary Forces: askHow

O agente remetente deseja saber todas as repostas do receptor sobre uma questão.



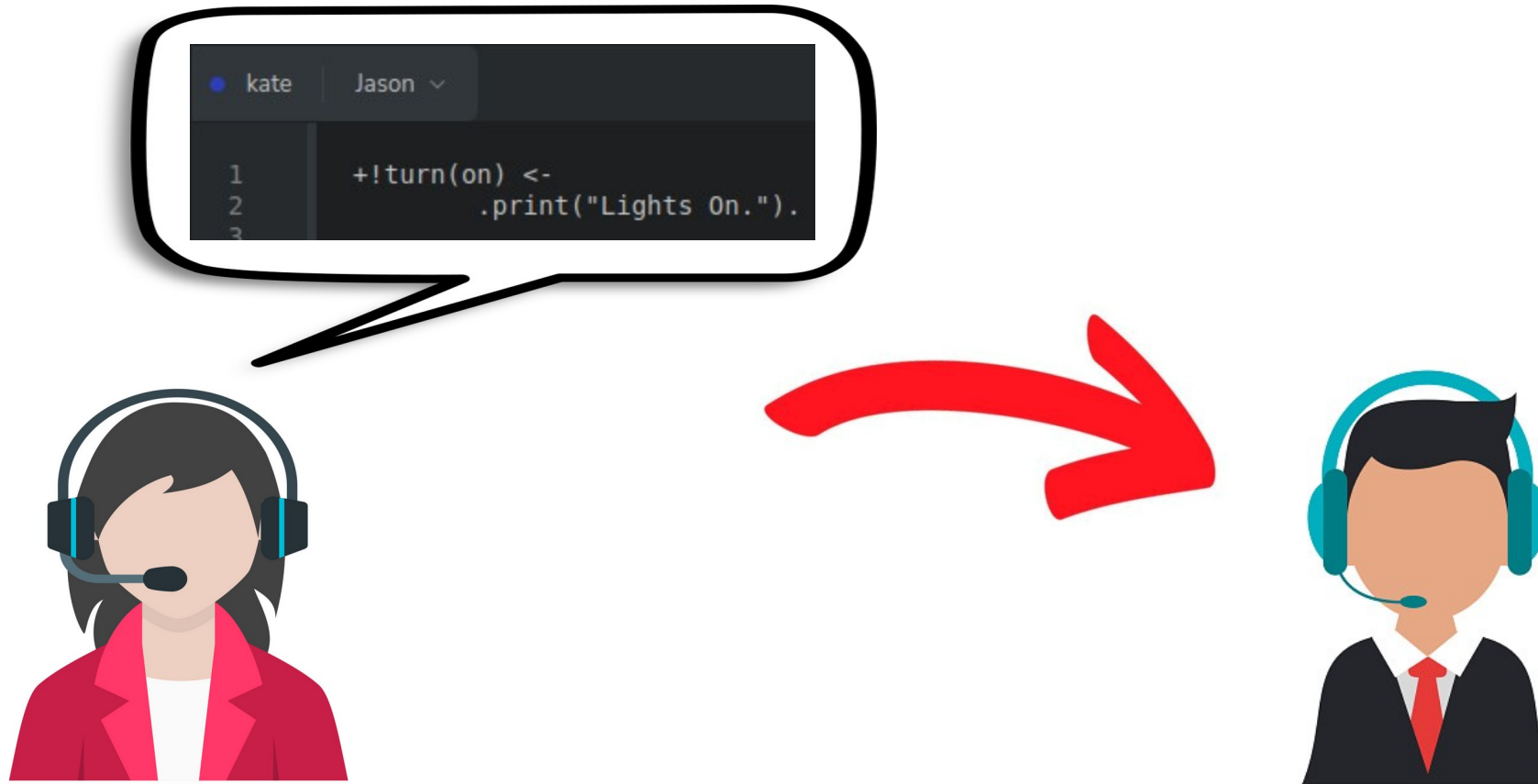
Illocutionary Forces: askHow



Illocutionary Forces: askHow



Illocutionary Forces: askHow



Illocutionary Forces: askHow

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.  
NOTE: Picked up JDK_JAVA_OPTIONS:  --add-opens=java.b  
rmi/sun.rmi.transport=ALL-UNNAMED  
Jason Http Server running on http://127.0.1.1:3272  
[bob] Could you teach me how to turn the lights on?  
[bob] Lights On.
```



MultiAgent System: Cozinheiro e Comilão



<https://github.com/chon-group/distributedAndEmbeddedAI/blob/main/02-Multi-agentSystem/examples/01-ProducerConsumer/agt/consumer.asl>



<https://github.com/chon-group/distributedAndEmbeddedAI/blob/main/02-Multi-agentSystem/examples/01-ProducerConsumer/agt/producer.asl>

Image credits:

https://smurfs.fandom.com/wiki/Chef_Smurf

<https://smurfs.fandom.com/pt/wiki/Fominha>

MultiAgent System: Exemplo

cozinheiro

```

1      /* Crenças Iniciais */
2      ultimoPedido(0).
3
4      /* Objetivos Iniciais */
5      !aguardarPedidos.
6
7      /* Planos */
8      +!aguardarPedidos: not preparandoPedido <-
9          .print("Aguardando Pedidos");
10         .wait(5000);
11         !aguardarPedidos.
12
13     +!aguardarPedidos: preparandoPedido<-
14         .wait(10000);
15         !aguardarPedidos.
16
17     +!pedido(Product,Qtd)[source(Cliente)] <-
18         +preparandoPedido;
19         ?ultimoPedido(N);
20         NrPedido=N+1;
21         +-ultimoPedido(NrPedido);
22         .print("Preparando o pedido Nr ",NrPedido);
23         .wait(2000);
24         .send(Cliente,tell,entregaComida(Product,Qtd));
25         -preparandoPedido.
26

```

comilao

```

1      /* Crenças Iniciais */
2      estoqueComida(0).    energia(0).
3
4      /* Objetivos Iniciais */
5      !curtir.
6
7      /* Planos */
8      +!comer: sintoFome & estoqueComida(C) & C>3 & energia(E) & E<=10 <-
9          +-estoqueComida(C-3);    +-energia(E+3);
10         .print("Comendo..... [Estoque de comida=",C-3,"] [Energia=",E+3,]");
11         .wait(1000);    !comer.
12
13     +!comer: energia(E) & E>10 <-
14         .print("Satisfeito..... [Energia=",E,]");
15         -sintoFome;    .wait(2000);    !curtir.
16
17     +!comer <-
18         ?estoqueComida(X);
19         .print("Preciso pedir comida..... [Estoque de comida=",X,]");
20         .wait(2000);    !pedirComida.
21
22     +!curtir: energia(E) & E>5 <-
23         .print("Curtindo....");    .wait(2000);    +-energia(E-3);
24         -sintoFome;    !curtir.
25
26     +!curtir: energia(E) & E<=5<-
27         ?energia(X);    .print("Sem Energia..... [Energia=",X,]");
28         +sintoFome;    .wait(2000);    !comer.
29
30     +!pedirComida <-
31         .print("Pedindo Comida");
32         .send(cozinheiro,achieve,pedido(lanche,5));
33         .wait(5000);    !comer.
34
35     +entregaComida(Product,Qtd)[source(Entregador)] <-
36         ?estoqueComida(X);
37         +-estoqueComida(X+Qtd);
38         .print("Oba! Chegou comida!!!!");
39         -entregaComida(Product,Qtd)[source(Entregador)].

```

ChonOS

A specific-purpose GNU/Linux
Status: **Alpha** Brought to you by

Git

SkyNet Server

Embedded MAS

RaspberryPi

SysConfig

Examples

DebianPackages

<https://sourceforge.net/p/chonos/examples/ci/master/tree/02-multiAgent/multiAgentExample01/>

MultiAgent System: Exemplo 2

banco	cozinheiro	comilao
<pre>1 /* Crenças Iniciais */ 2 saldo(comilao,20). 3 saldo(cozinheiro,10). 4 5 /* Objetivos Iniciais */ 6 7 /* Planos */ 8 +!pix(Destino,Valor)[source(Origem)]: saldo(Origem,Saldo) 9 & Saldo >= Valor <- 10 11 NovoSaldo = Saldo - Valor; 12 -saldo(Origem,Saldo); 13 +saldo(Origem,NovoSaldo); 14 15 ?saldo(Destino,SaldoDestino); 16 NovoSaldoDestino = SaldoDestino + Valor; 17 -saldo(Destino,SaldoDestino); 18 +saldo(Destino,NovoSaldoDestino); 19 20 .random(NrRegistro); 21 +operacao(NrRegistro,Origem,Destino,Valor); 22 .print("Pix realizado de [",Origem, 23 " para [",Destino, 24 " valor [",Valor, 25 " código [",NrRegistro,""]); 26 .send(Origem,tell,operacao(NrRegistro,Origem,Destino,Valor)). 27 28 +!pix(Destino,Valor)[source(Origem)]: saldo(Origem,Saldo) 29 & Saldo < Valor <- 30 .print("Saldo insuficiente... saldo=",Saldo); 31 .send(Origem,tell,semDinheiro).</pre>	<pre>1 /* Crenças Iniciais */ 2 ultimoPedido(0). 3 /* Objetivos Iniciais */ 4 !aguardarPedidos. 5 /* Planos */ 6 +!aguardarPedidos: not preparandoPedido <- 7 .print("Aguardando Pedidos"); 8 .wait(5000); 9 !aguardarPedidos. 10 11 +!aguardarPedidos: preparandoPedido<- 12 .wait(10000); 13 !aguardarPedidos. 14 15 +!pedido(Product,Qtd,Pix)[source(Cliente)] <- 16 .send(banco,askOne,operacao(Pix,Cliente,cozinheiro,Qtd),Reply); 17 +Reply; 18 ?operacao(Registro,Cliente,cozinheiro,Qtd); 19 +preparandoPedido; 20 !prepararPedido(Product,Qtd,Pix,Registro,Cliente); 21 +atendido(Pix); 22 -preparandoPedido. 23 24 +!prepararPedido(Product,Qtd,Pix,Validacao,Cliente): Pix=Validacao 25 & not atendido(Pix)<- 26 27 ?ultimoPedido(N); 28 NrPedido=N+1; 29 -+ultimoPedido(NrPedido); 30 .print("Preparando o pedido Nr ",NrPedido); 31 .wait(2000); 32 .send(Cliente,tell,entregaComida(Product,Qtd)). 33 34 -!prepararPedido(Product,Qtd,Pix,Validacao,Cliente) <- 35 .print("Pix inválido ou já atendido").</pre>	<pre>1 /* Crenças Iniciais */ 2 estoqueComida(0). 3 energia(0). 4 /* Objetivos Iniciais */ 5 !curtir. 6 /* Planos */ 7 +!comer: sintoFome & estoqueComida(C) & C>3 8 & energia(E) & E<=10 <- 9 -+estoqueComida(C-3); -+energia(E+3); 10 .print("Comendo..... [Geladeira=",C-3,"] [Energia=",E+3,"]"); 11 .wait(1000); !comer. 12 13 +!comer: energia(E) & E>10 <- 14 .print("Satisfeito..... [Energia=",E,"]"); 15 -sintoFome; .wait(2000); !curtir. 16 17 +!comer <- 18 ?estoqueComida(X); 19 .print("Preciso pedir comida..... [Geladeira=",X,"]"); 20 .wait(2000); !pedirComida. 21 22 +!curtir: energia(E) & E>5 <- 23 .print("Curtindo....."); .wait(2000); -+energia(E-3); 24 -sintoFome; !curtir. 25 26 +!curtir: energia(E) & E<=5<- 27 ?energia(X); .print("Sem Energia..... [Energia=",X,"]"); 28 +sintoFome; .wait(2000); !comer. 29 30 +!pedirComida <- 31 .print("Pedindo Comida"); 32 .send(banco,achieve,pix(cozinheiro,5)); 33 .wait(1000); 34 ?operacao(CodPix,comilao,cozinheiro,5); 35 .send(cozinheiro,achieve,pedido(lanche,5,CodPix)); 36 -operacao(CodPix,comilao,cozinheiro,5); 37 .wait(5000); !comer. 38 39 +entregaComida(Product,Qtd)[source(Entregador)] <- 40 ?estoqueComida(X); 41 -+estoqueComida(X+Qtd); 42 .print("Oba! Chegou comida!!!!"); 43 -entregaComida(Product,Qtd)[source(Entregador)]. 44 45 +semDinheiro[source(banco)] <- 46 .drop_desire(curtir); 47 .drop_desire(pedirComida); 48 .print("Ababou a festa :(").</pre>

<https://sourceforge.net/p/chonos/examples/ci/master/tree/02-multiAgent/multiAgentExample02/>

OBRIGADO!

pantoja@cefet-rj.br
nilson.lazarin@cefet-rj.br

