

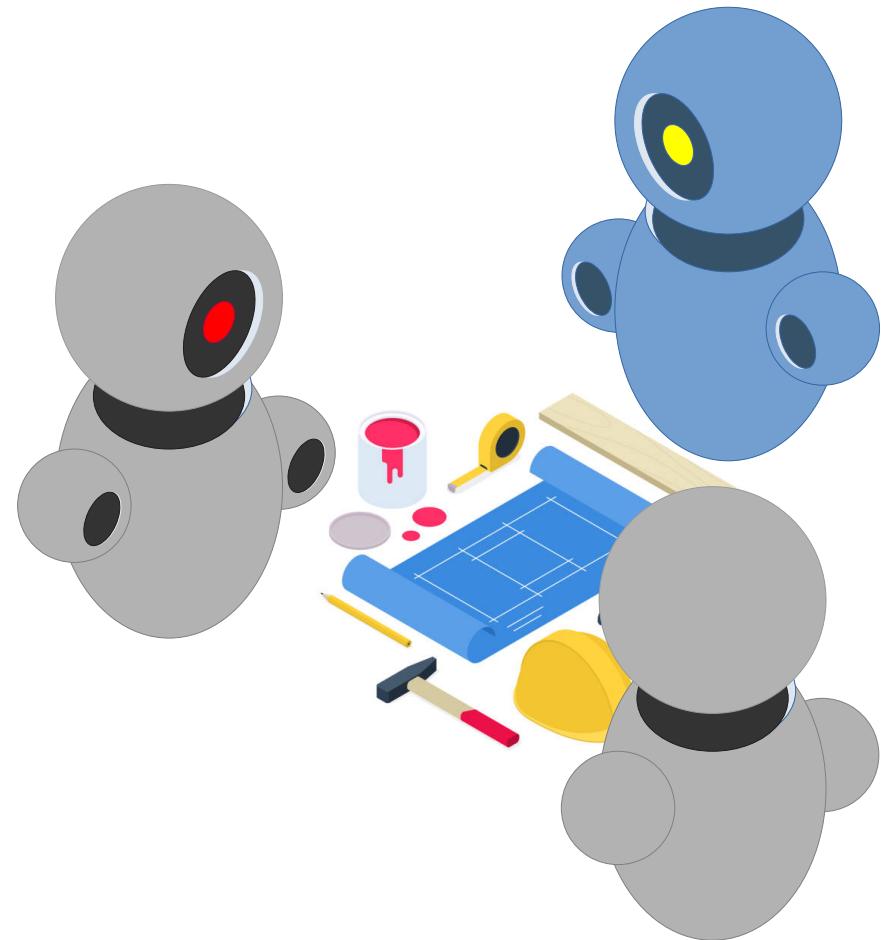
Introduction to Distributed and Embedded Multi-agent Systems

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

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



INTRODUCTION



Who we are?

The **Cognitive Hardware on Networks** group or
CHON Group.



Who we are?

The **Cognitive Hardware on Networks** group or
CHON Group.



Prof. NILSON MORI LAZARIN

CEFET/RJ and UFF



Who we are?

The **Cognitive Hardware on Networks** group or
CHON Group.



Prof. NILSON MORI LAZARIN
CEFET/RJ and UFF



Prof. CARLOS PANTOJA
CEFET/RJ



Who we are?

The **Cognitive Hardware on Networks** group or
CHON Group.



Prof. NILSON MORI LAZARIN
CEFET/RJ and UFF



Prof. CARLOS PANTOJA
CEFET/RJ



Where we have been?

This workshop is part of:

Where we have been?

This workshop is part of:

- **2 undergraduate courses** (CEFET/RJ Maria da Graça and Nova Friburgo);

Where we have been?

This workshop is part of:

- **2 undergraduate courses** (CEFET/RJ Maria da Graça and Nova Friburgo);
- **1 graduate course** (M.Sc. and Ph.D. at UFF);

Where we have been?

This workshop is part of:

- **2 undergraduate courses** (CEFET/RJ Maria da Graça and Nova Friburgo);
- **1 graduate course** (M.Sc. and Ph.D. at UFF);
- It was **presented at**: UTFPR, IFSC, IFC, CEFET/RJ, UFRJ, UFF, and Estácio;

Where we have been?

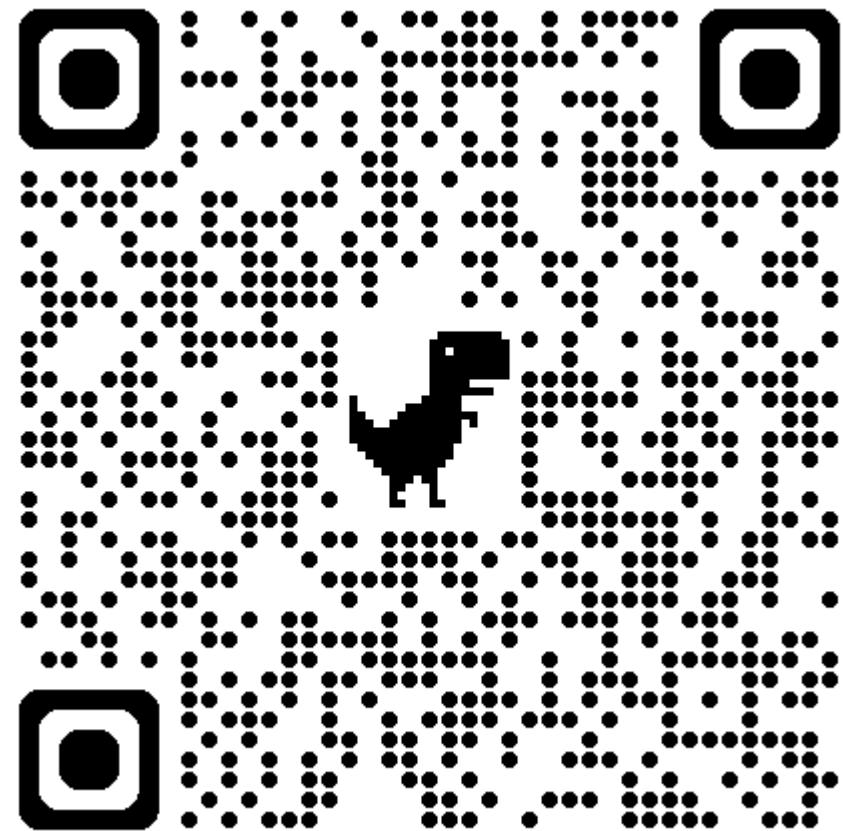
This workshop is part of:

- **2 undergraduate courses** (CEFET/RJ Maria da Graça and Nova Friburgo);
- **1 graduate course** (M.Sc. and Ph.D. at UFF);
- It was **presented at**: UTFPR, IFSC, IFC, CEFET/RJ, UFRJ, UFF, and Estácio;
- It was also **presented at**: WESAAC'22, WESAAC'23, and SEKE'23,

Material

All material are available on

[https://github.com/chon-group/distributedAndE
mbeddedAI/tree/main/workshops/wesaac/202
5](https://github.com/chon-group/distributedAndEmbeddedAI/tree/main/workshops/wesaac/2025)



Goals

This workshop goal is:

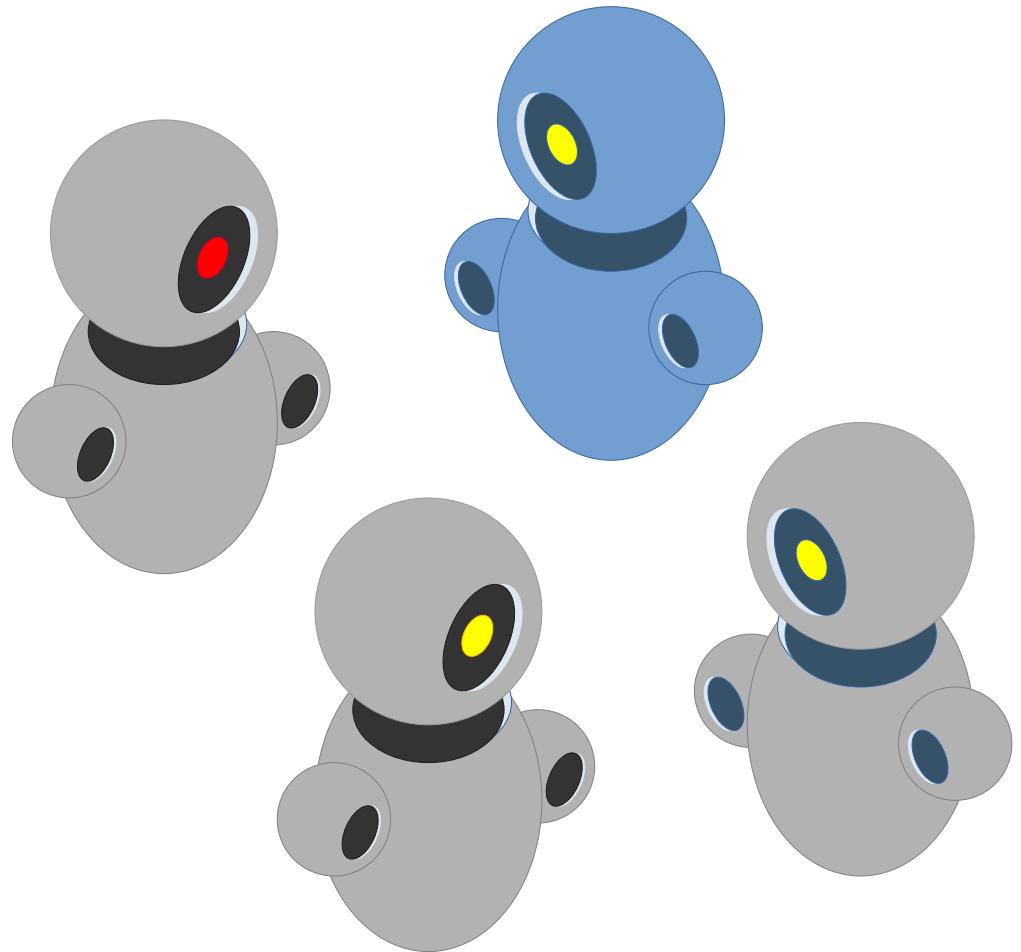
- To experience the development process of Embedded MAS using BDI Agents and Jason.

... and the **most important** of all...

Goals

HAVE FUN!

THE AGENT ORIENTED APPROACH

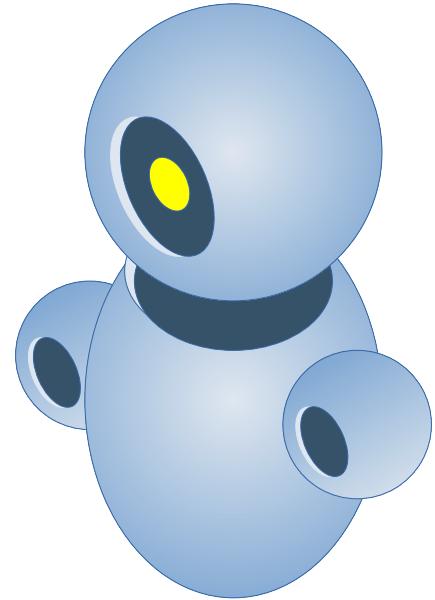


Agente

É um sistema computacional capaz de perceber e agir em um ambiente por meio de sua deliberação baseada em suas convicções e motivações.

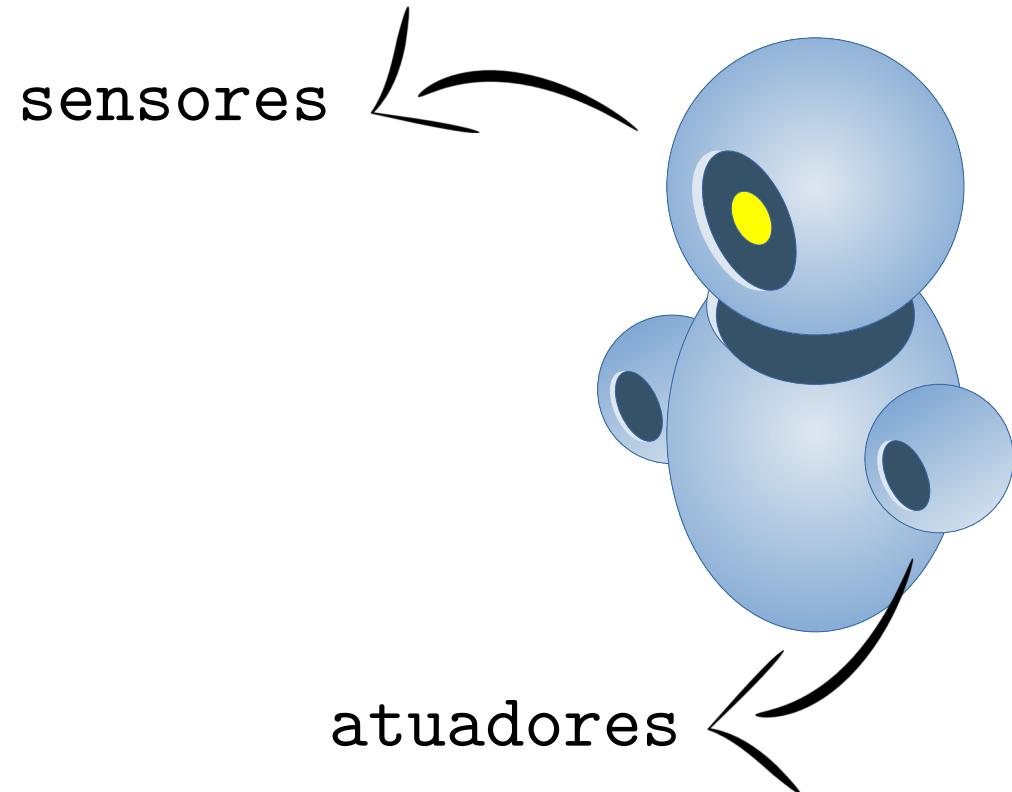
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



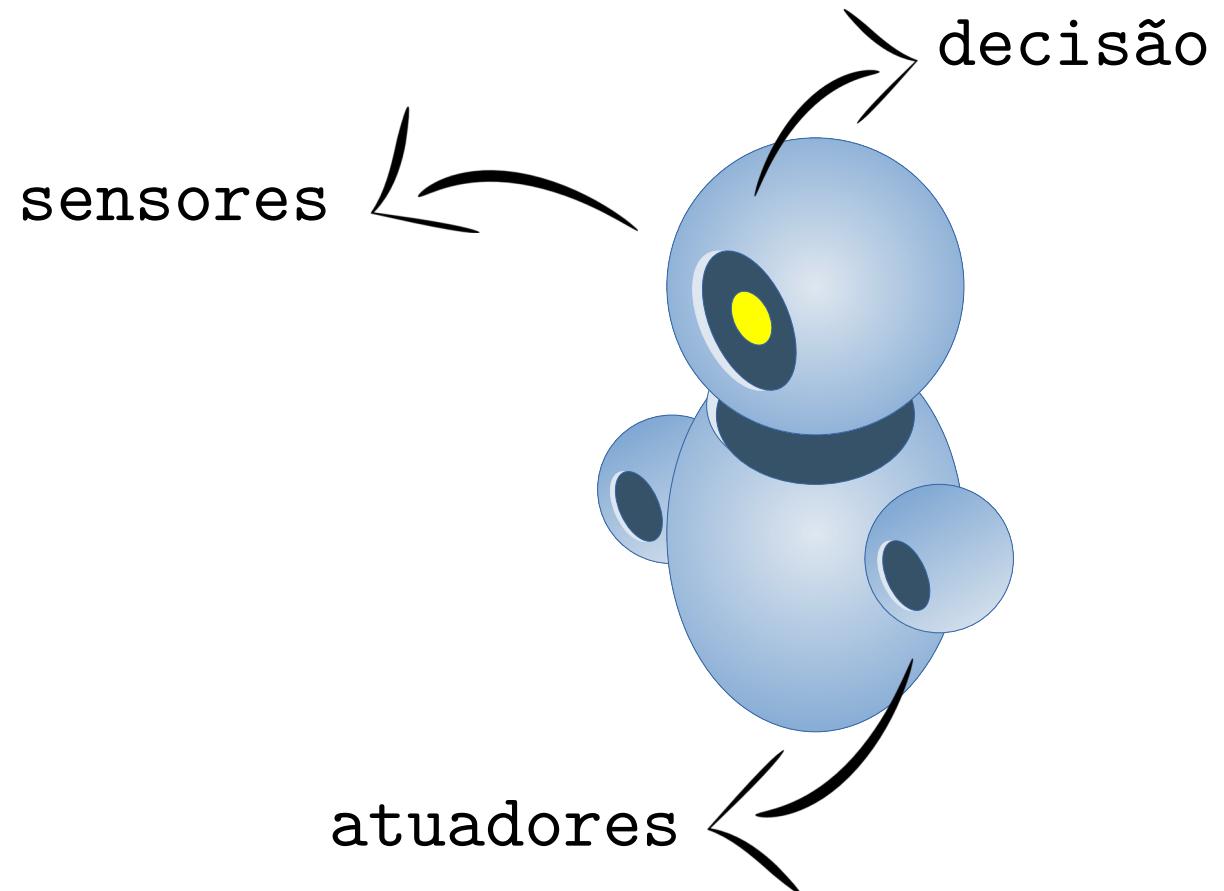
WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



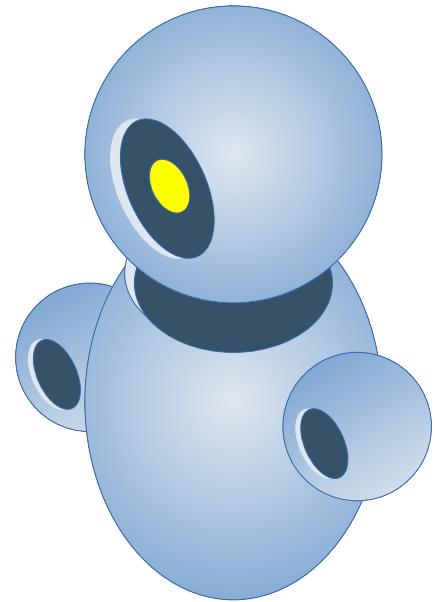
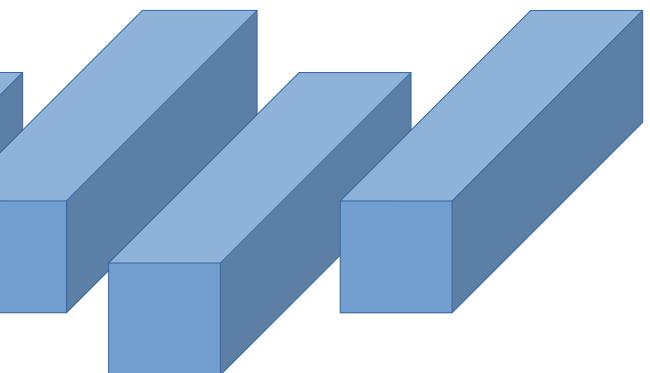
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



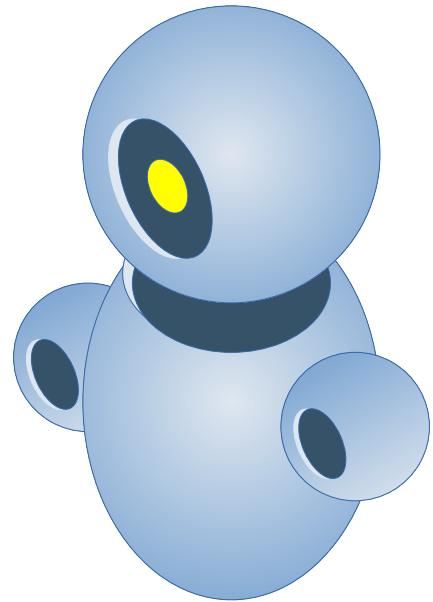
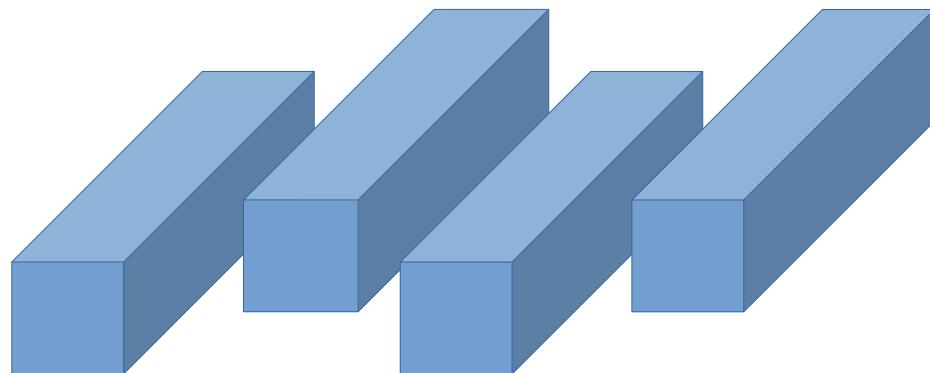
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



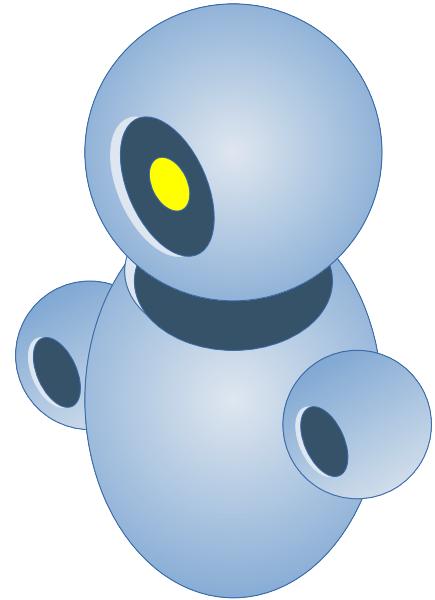
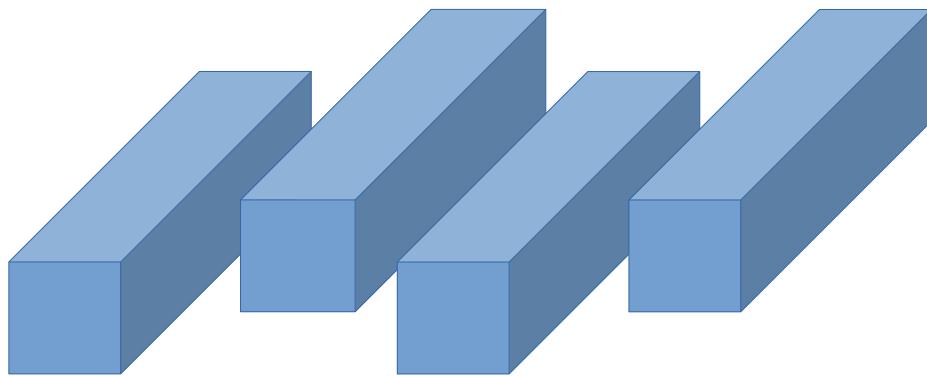
WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



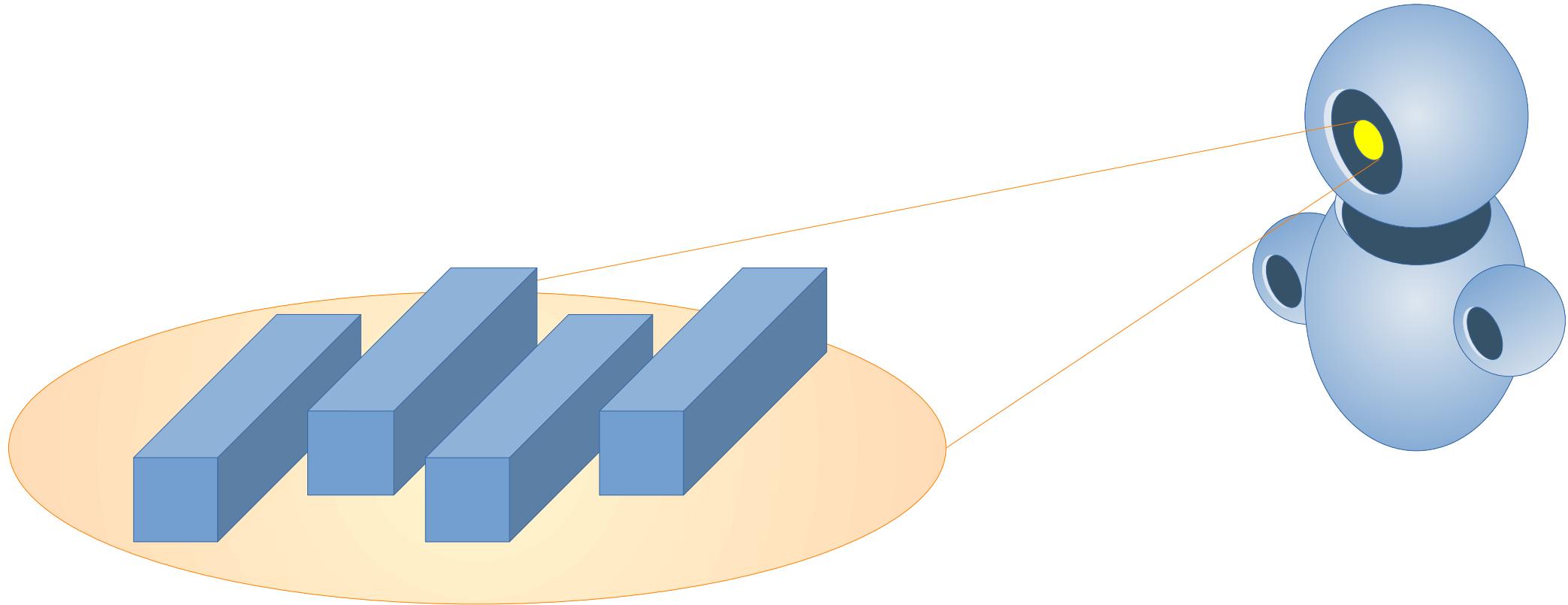
WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

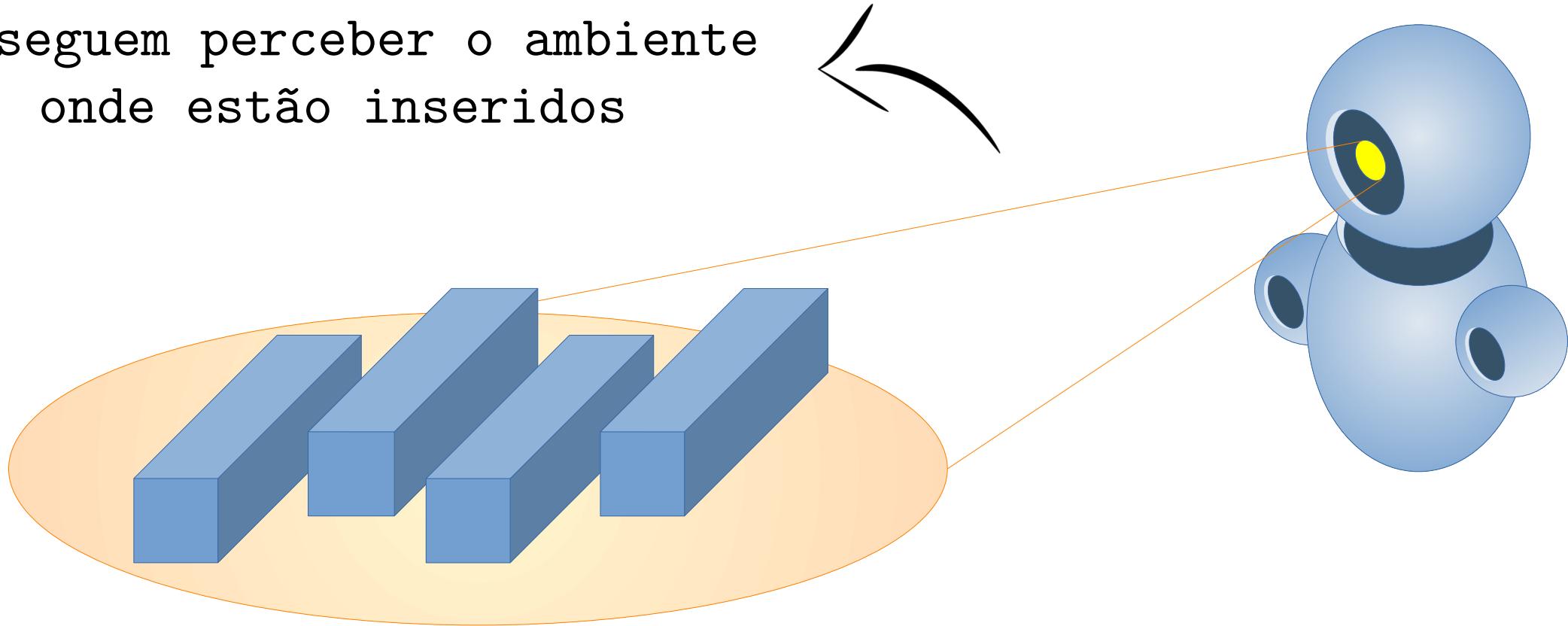
Agente



WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

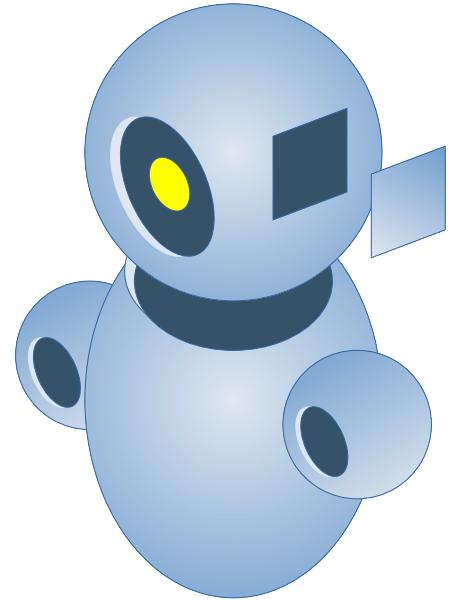
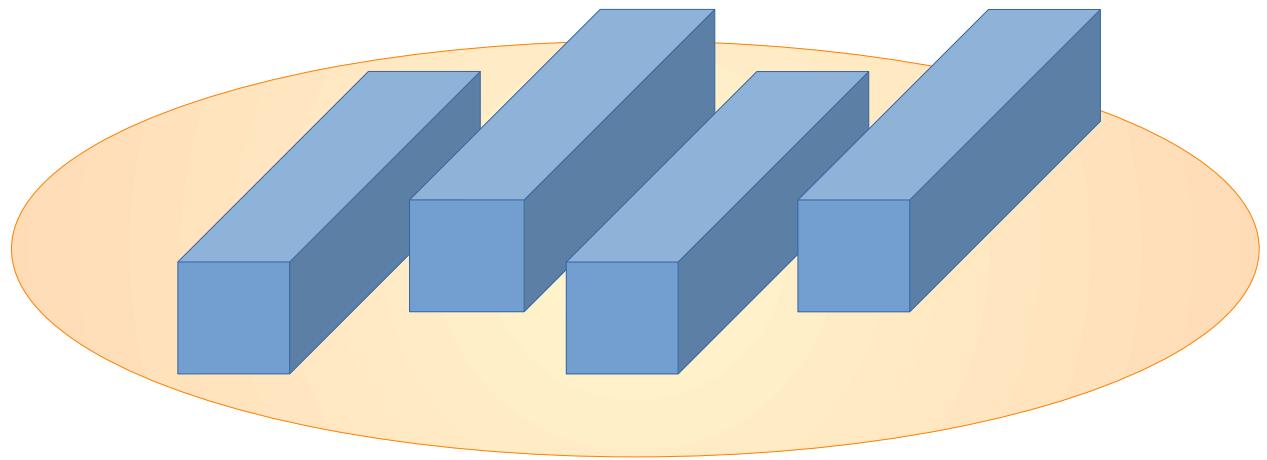
Agente

conseguem perceber o ambiente
onde estão inseridos



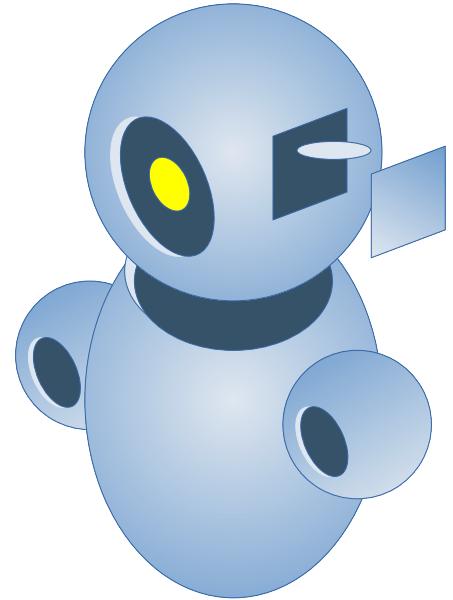
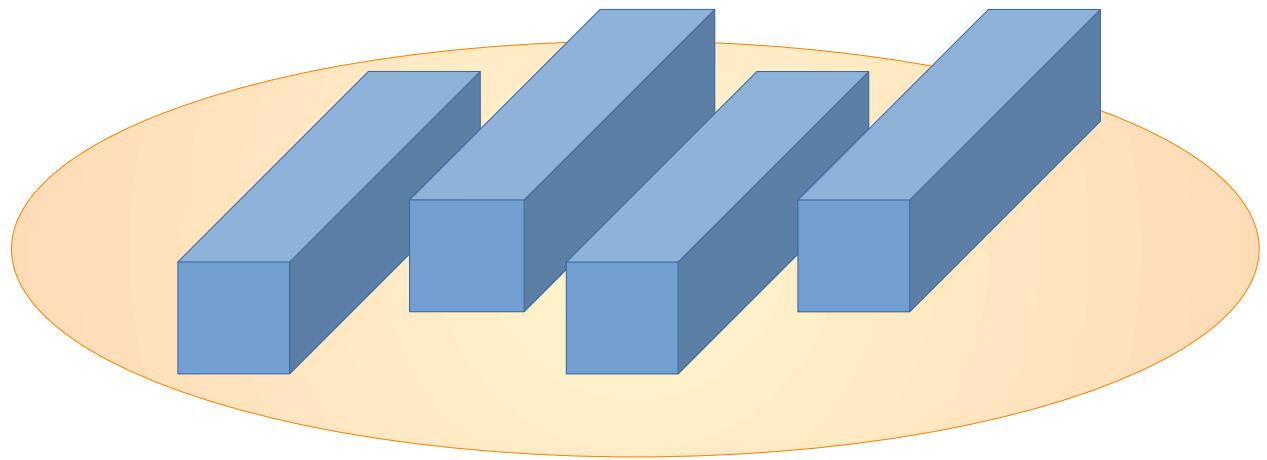
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



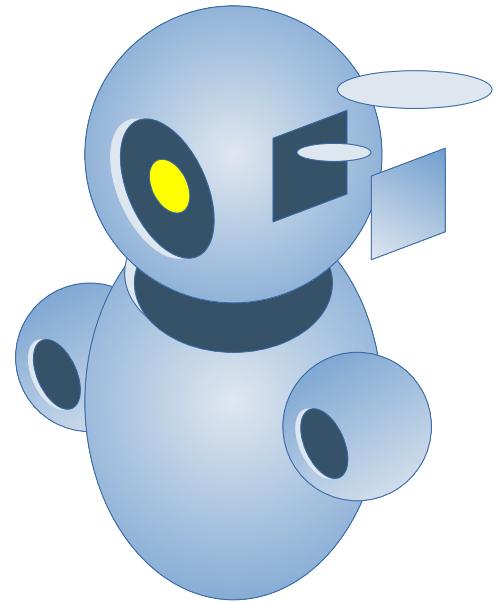
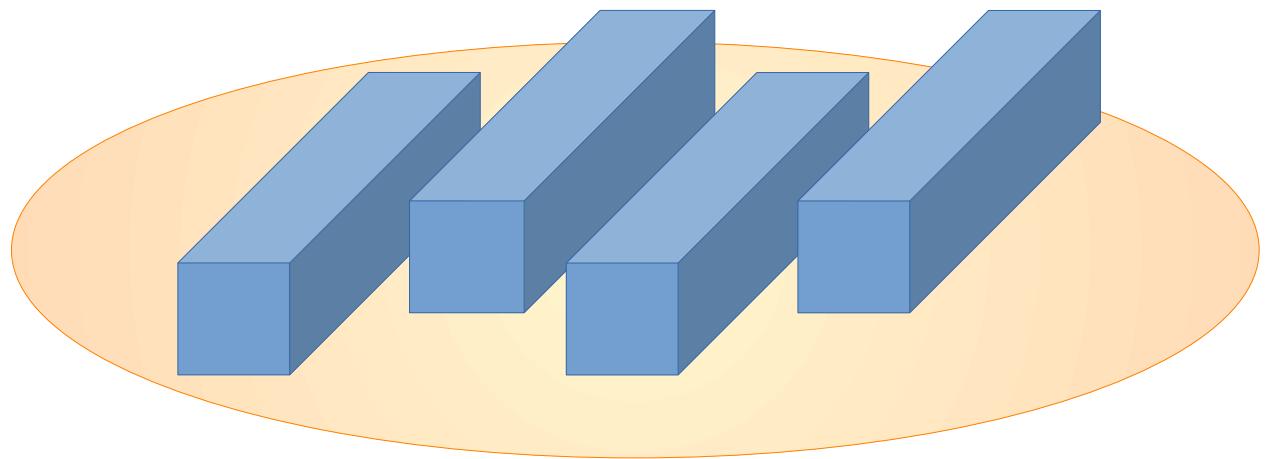
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



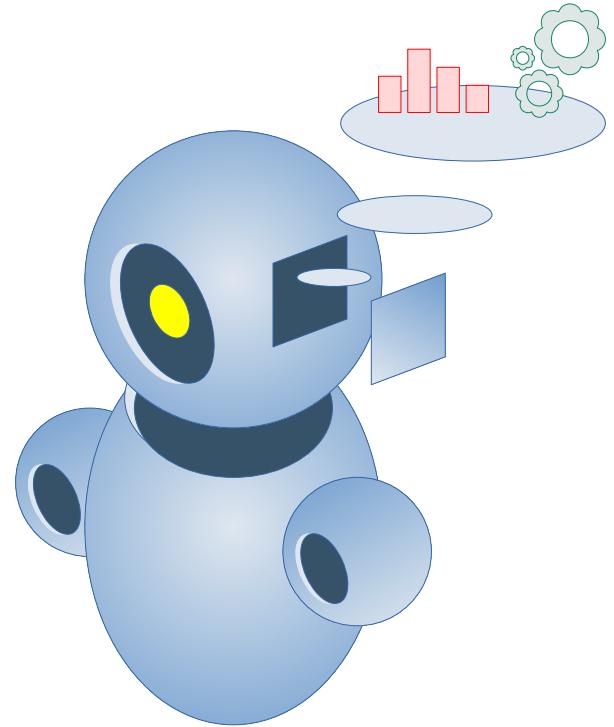
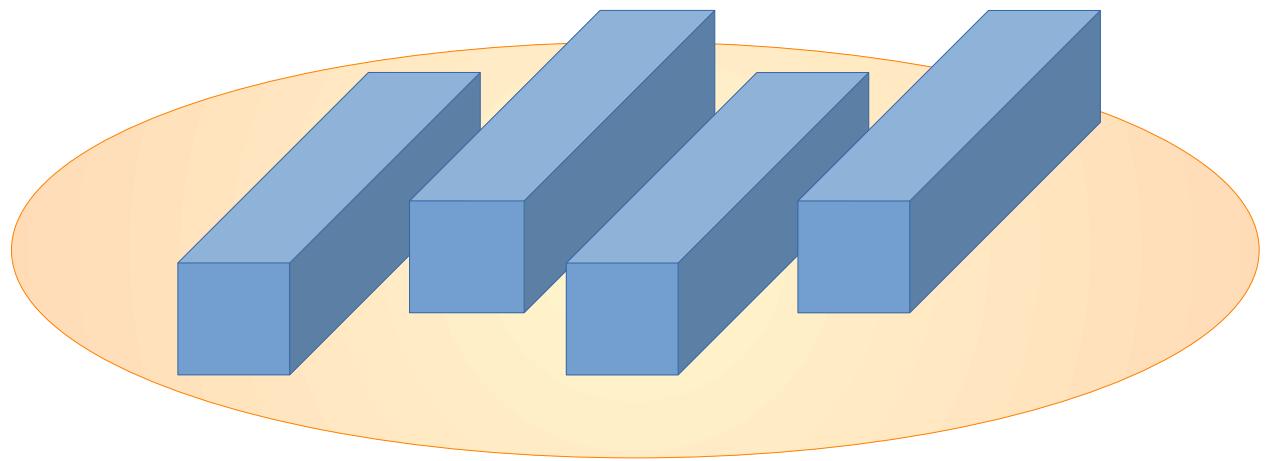
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

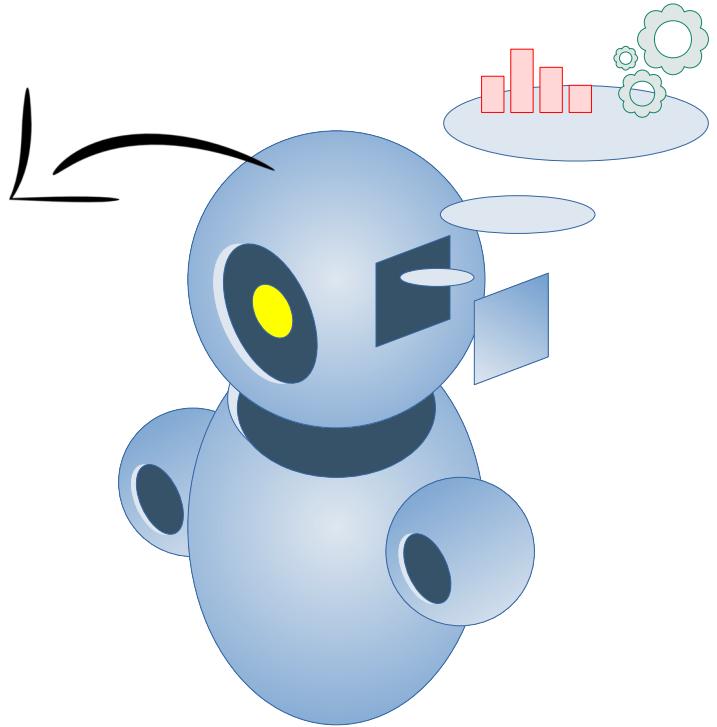
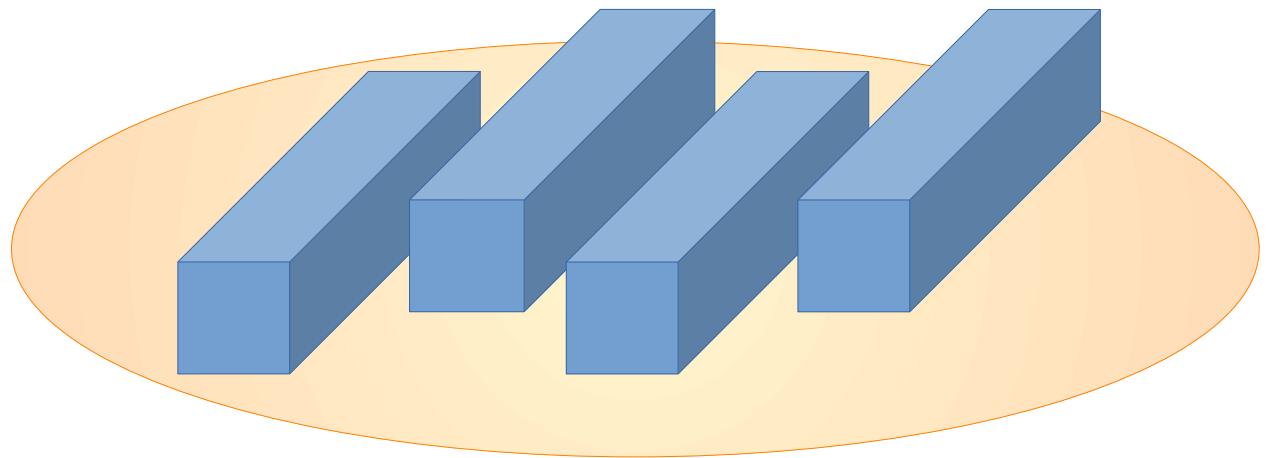
Agente



WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

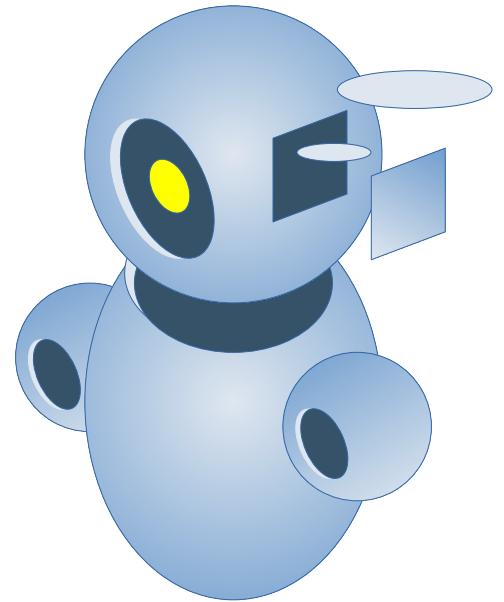
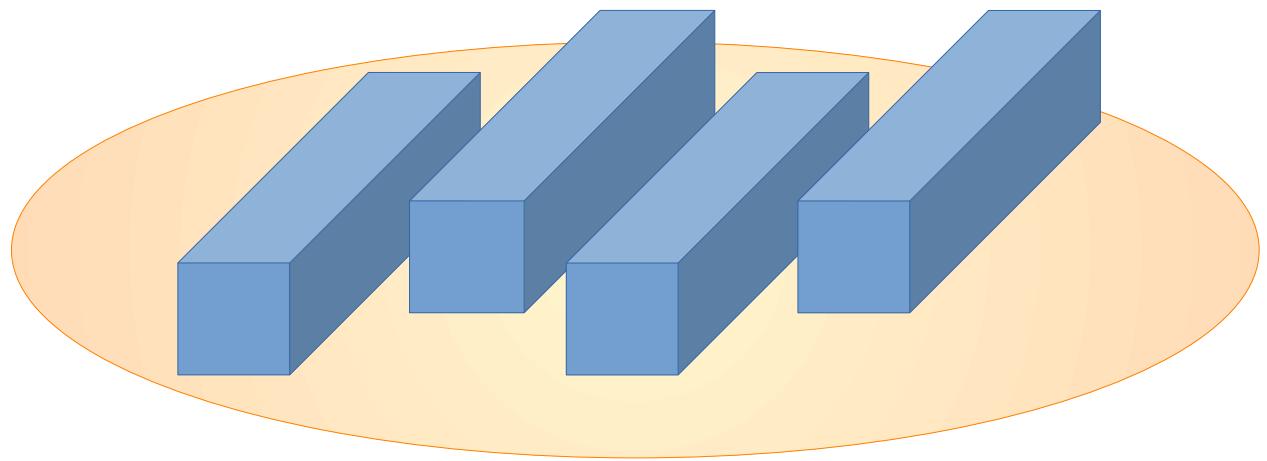
Agente

tomam decisões baseados em suas percepções e crenças



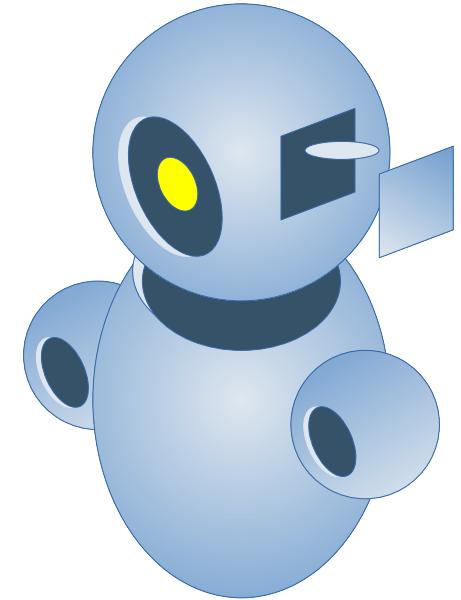
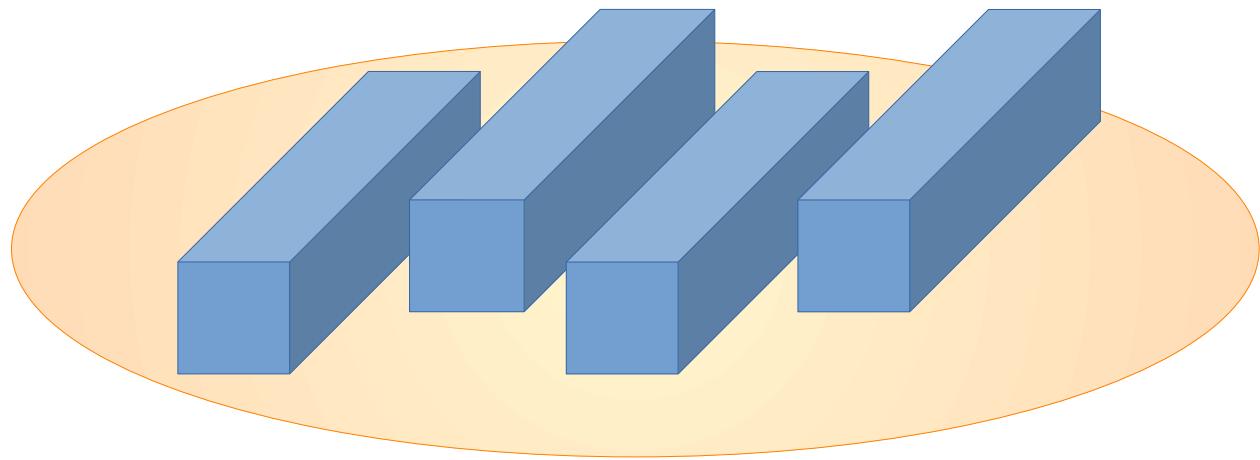
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



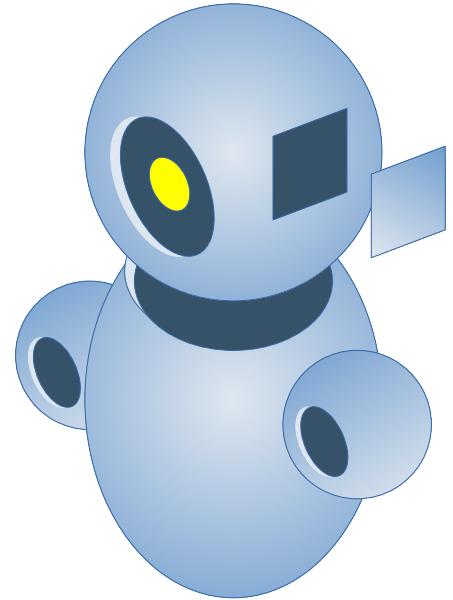
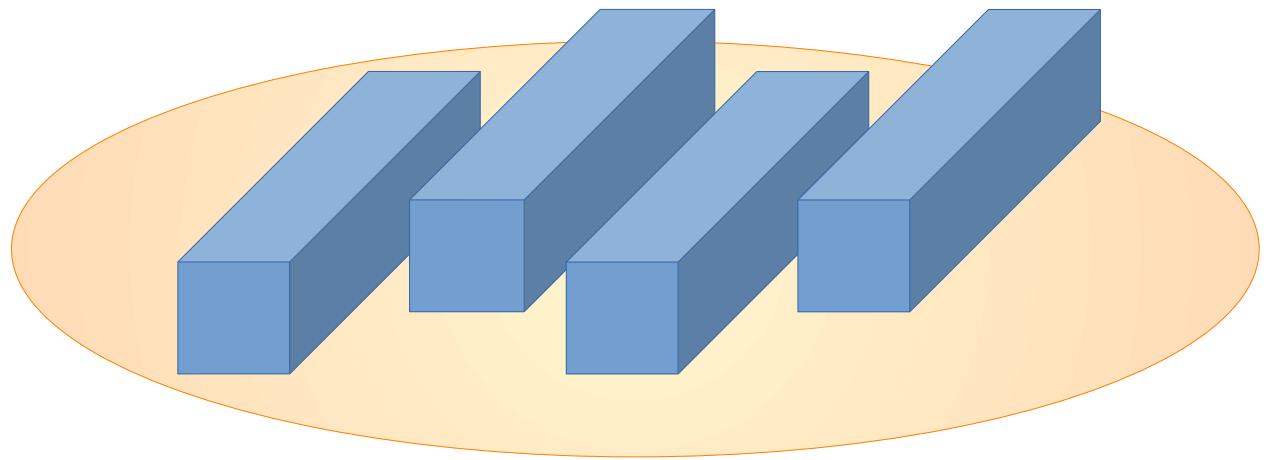
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



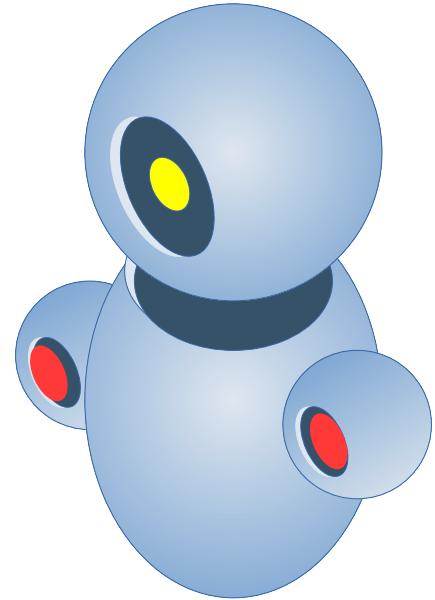
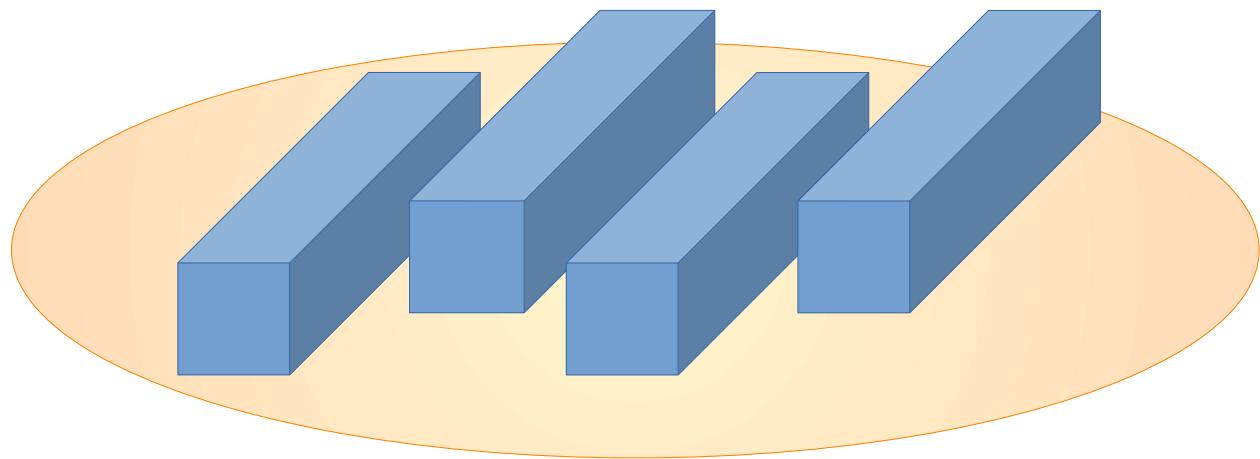
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



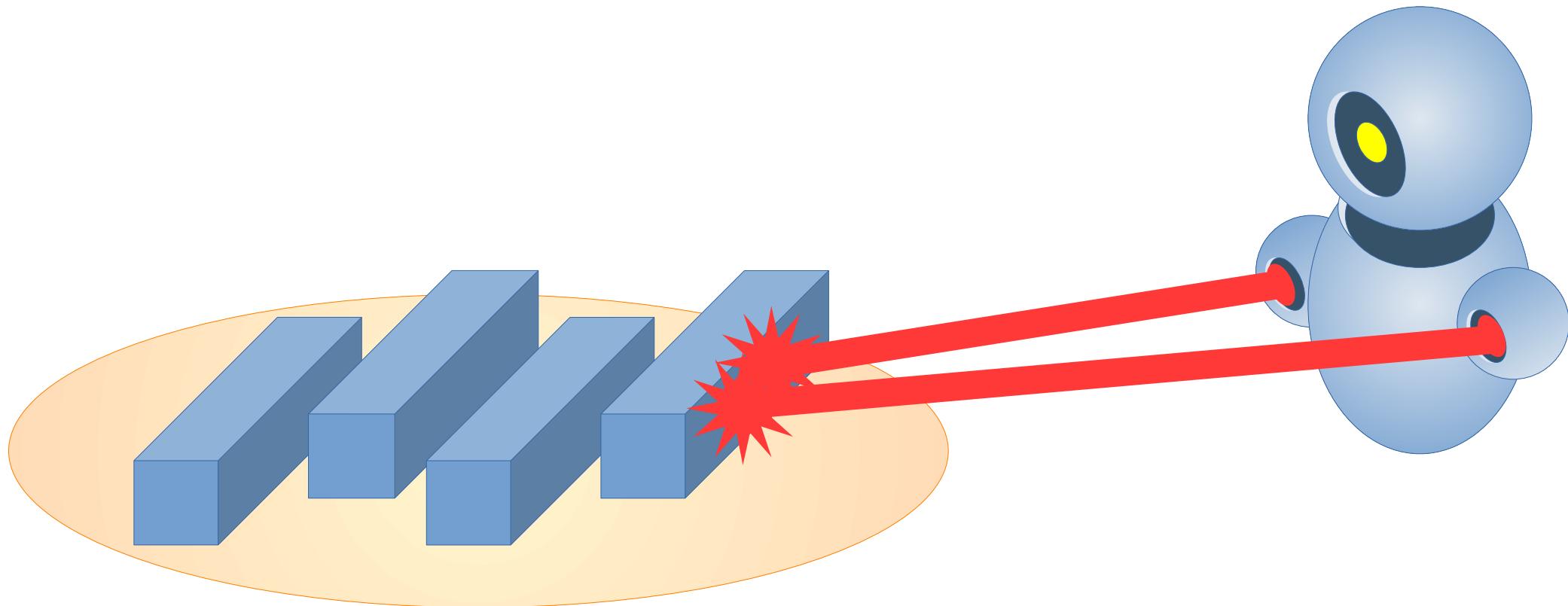
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



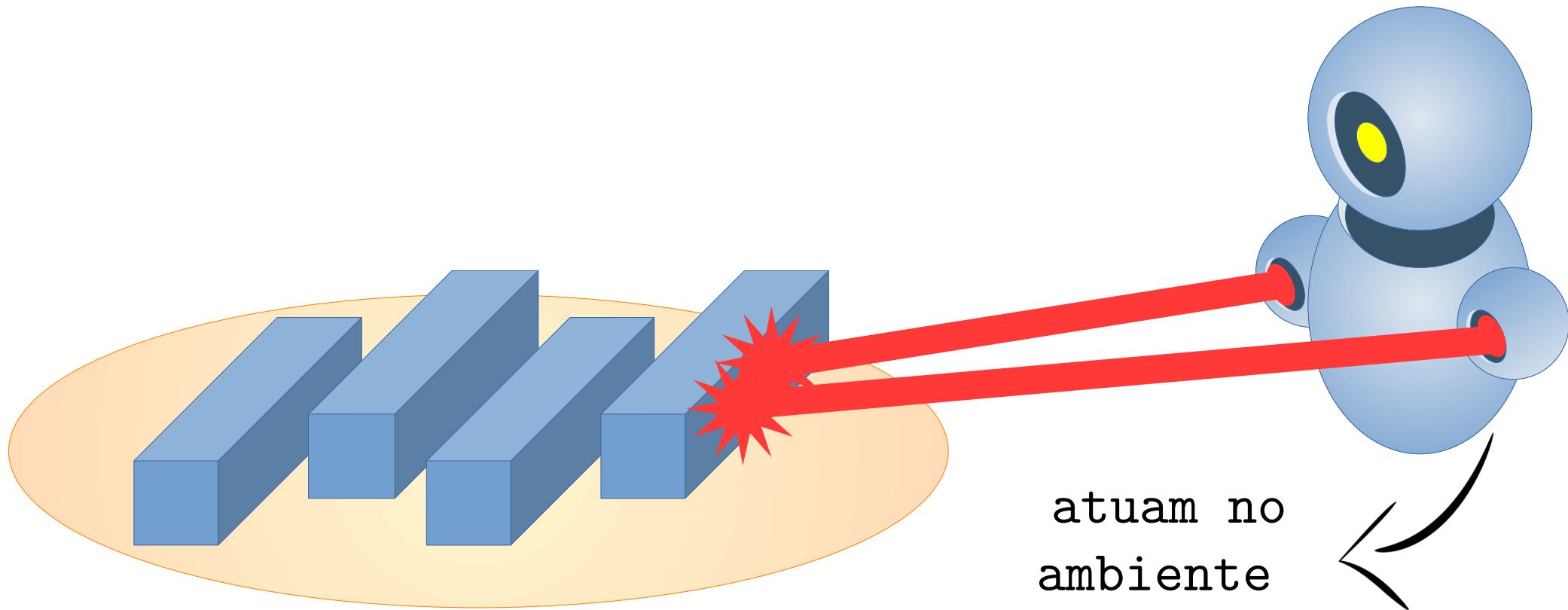
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



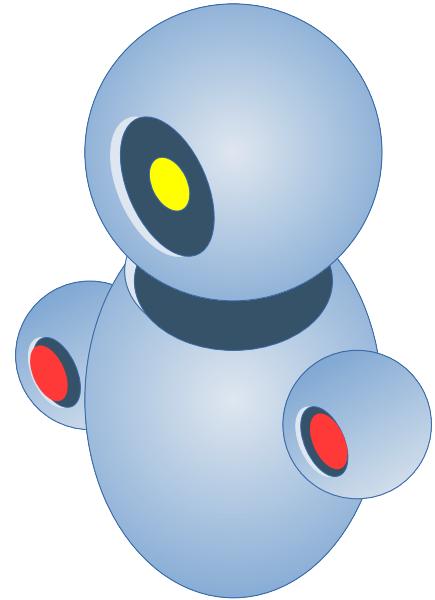
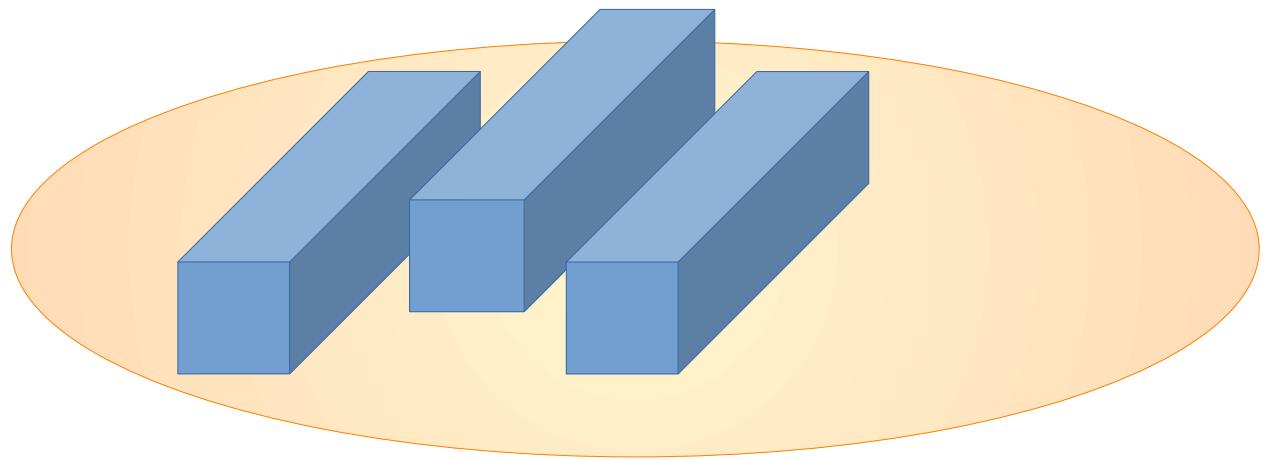
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



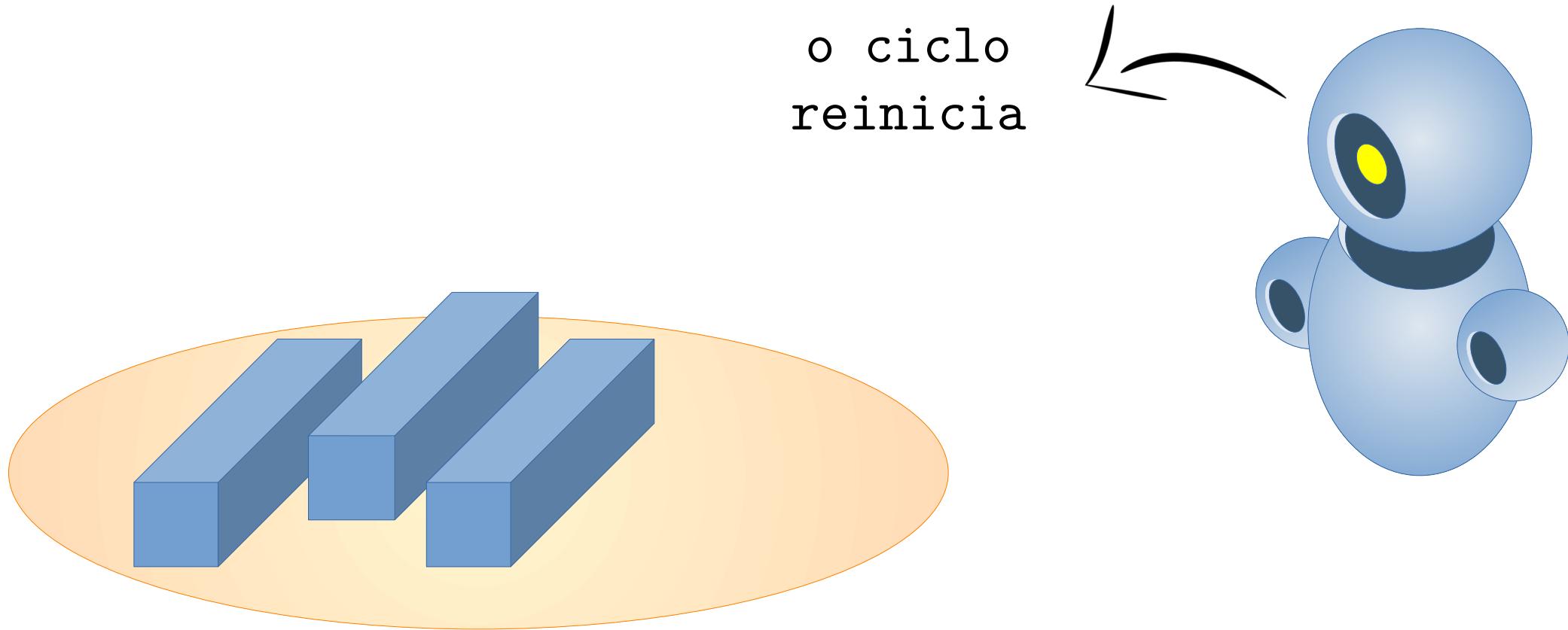
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Agente



WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Sistemas Multiagentes (SMA)

- Um SMA é um grupo de agentes autônomos fracamente acoplados trabalhando no mesmo ambiente.

WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

Sistemas Multiagentes (SMA)

- Um SMA é um grupo de agentes autônomos fracamente acoplados trabalhando no mesmo ambiente.
- Os agentes podem colaborar em objetivos comuns e competir em objetivos conflitantes.

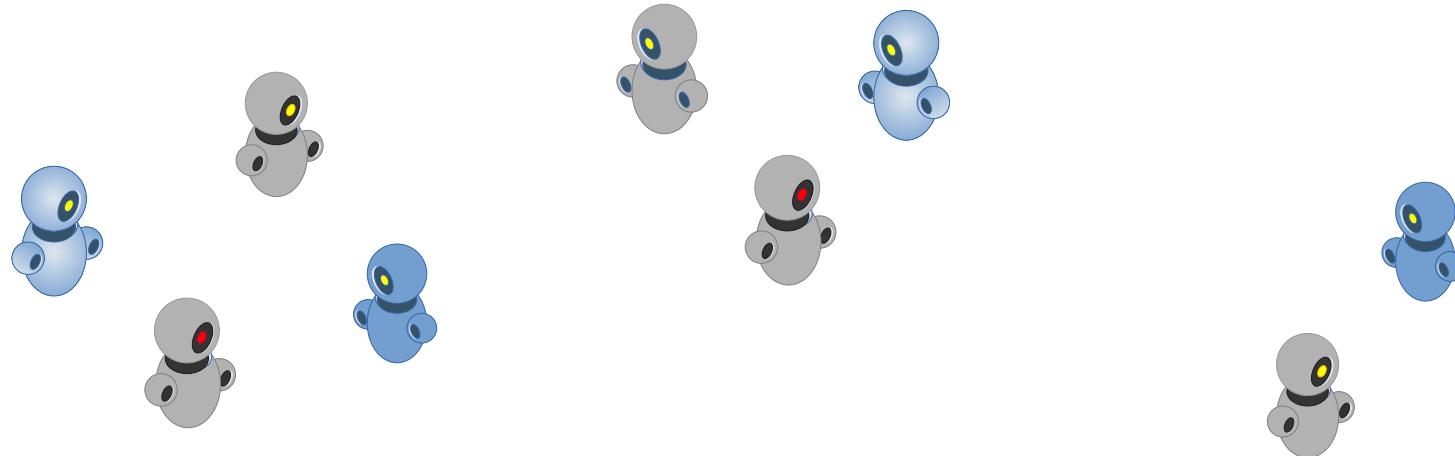
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

Sistemas Multiagentes (SMA)

- Um SMA é um grupo de agentes autônomos fracamente acoplados trabalhando no mesmo ambiente.
- Os agentes podem colaborar em objetivos comuns e competir em objetivos conflitantes.
- Atuam sob esferas de influências - limite de influência que um agente exerce em uma parte do ambiente

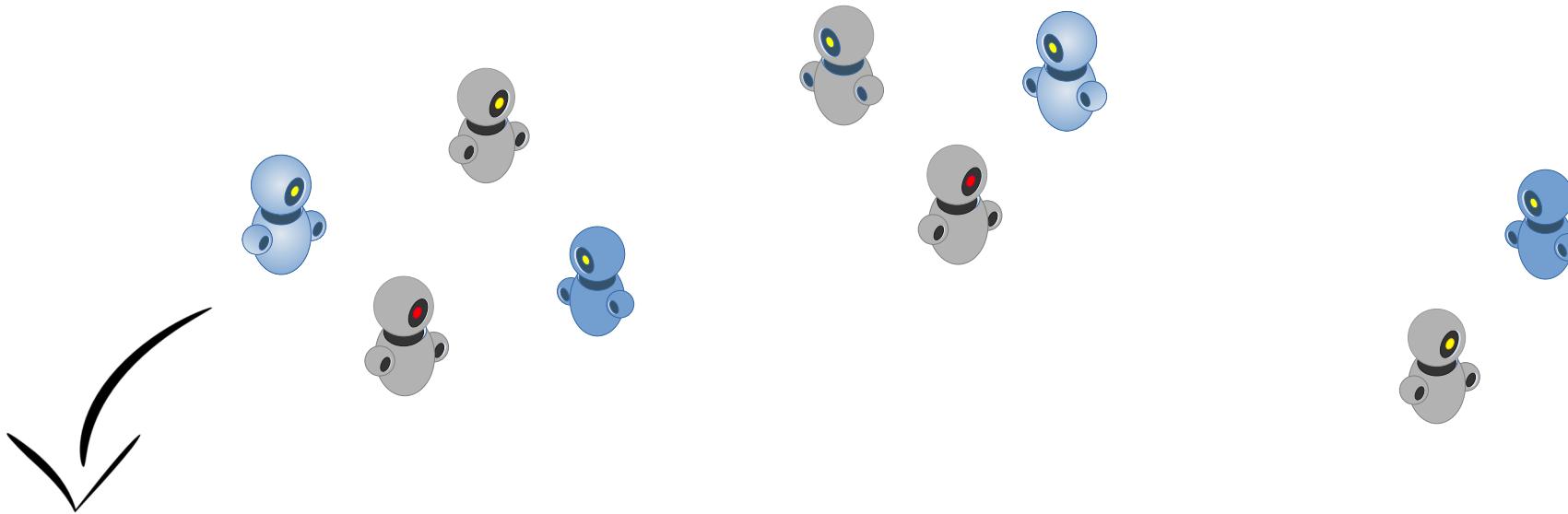
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

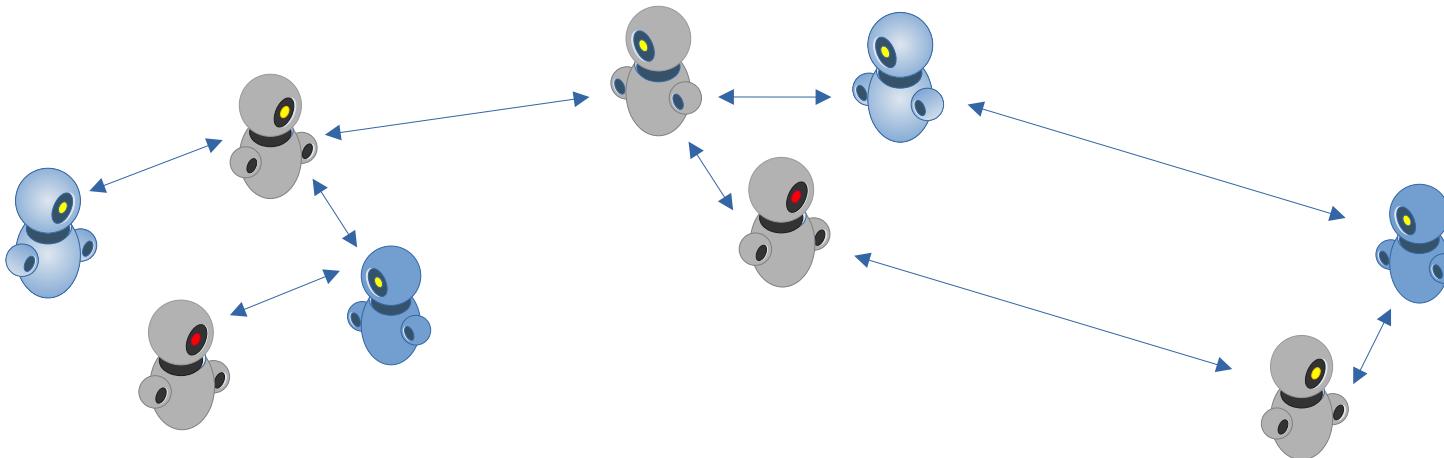
As Dimensões de Sistemas Multiagentes (SMA)



Agentes são autônomos
pró-ativos
reativos

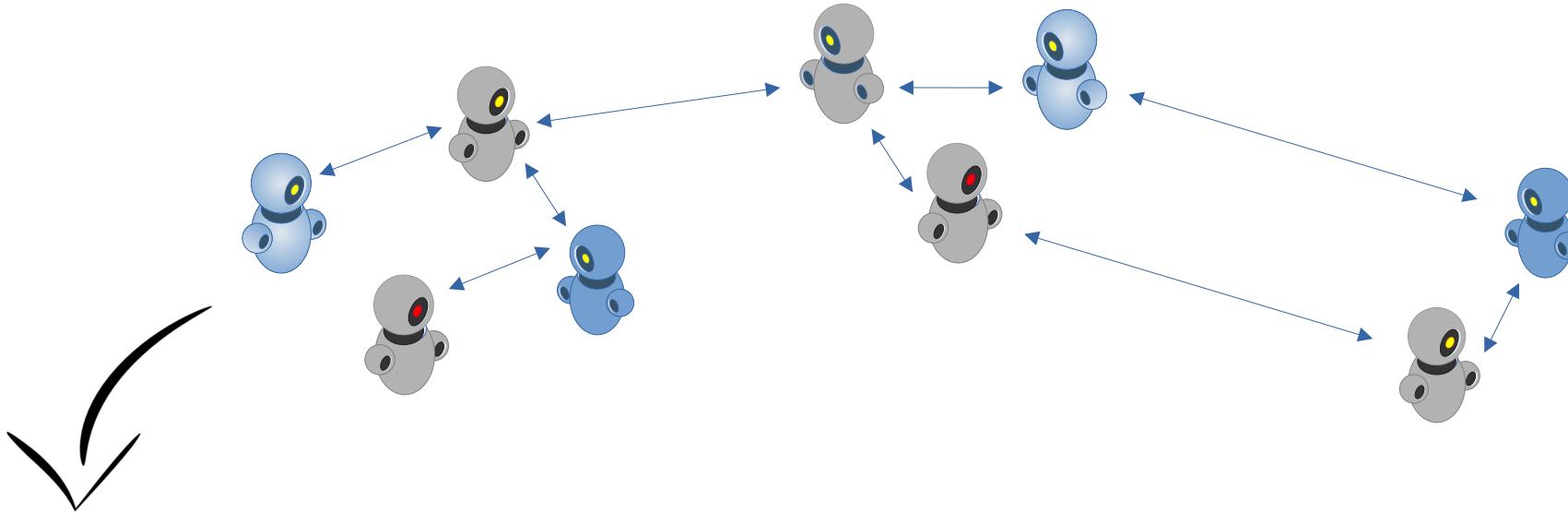
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

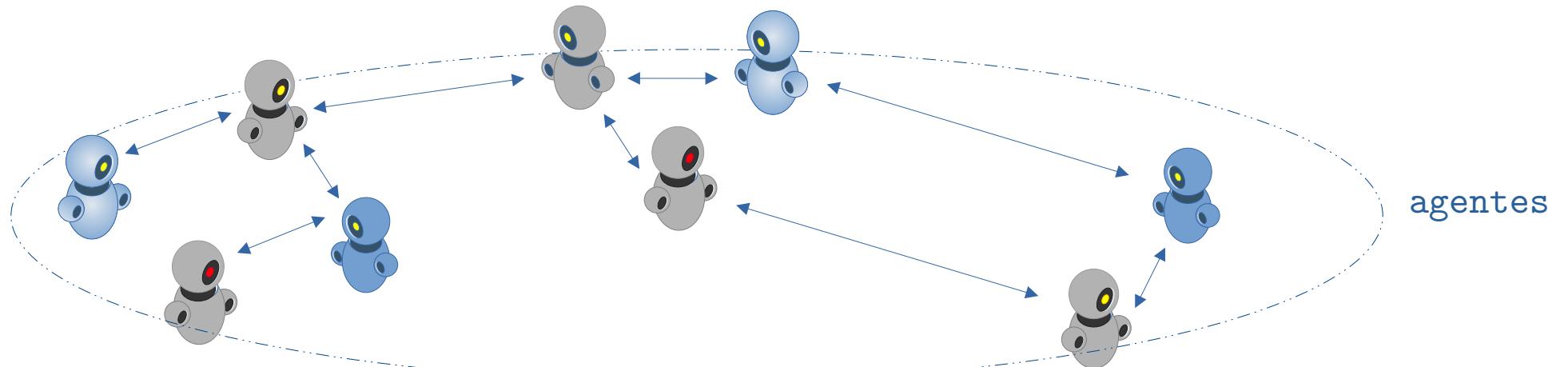
As Dimensões de Sistemas Multiagentes (SMA)



com habilidade
social

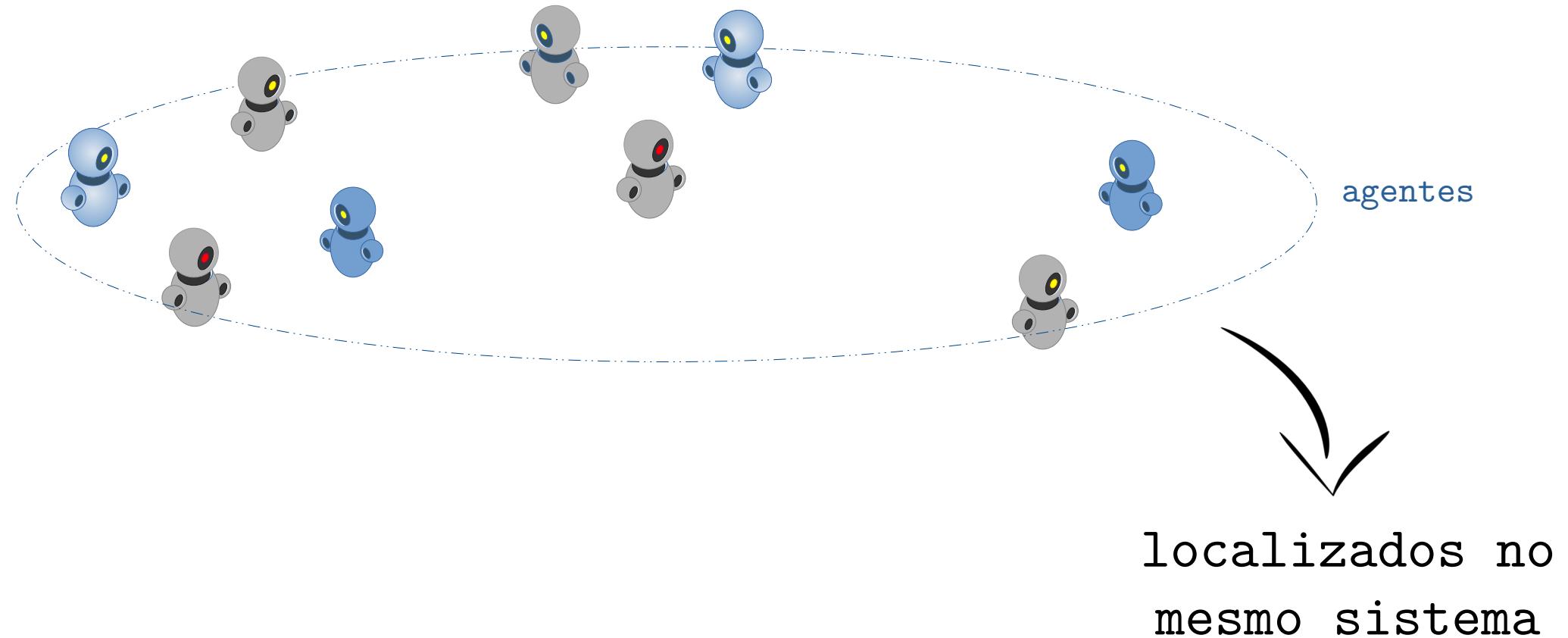
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



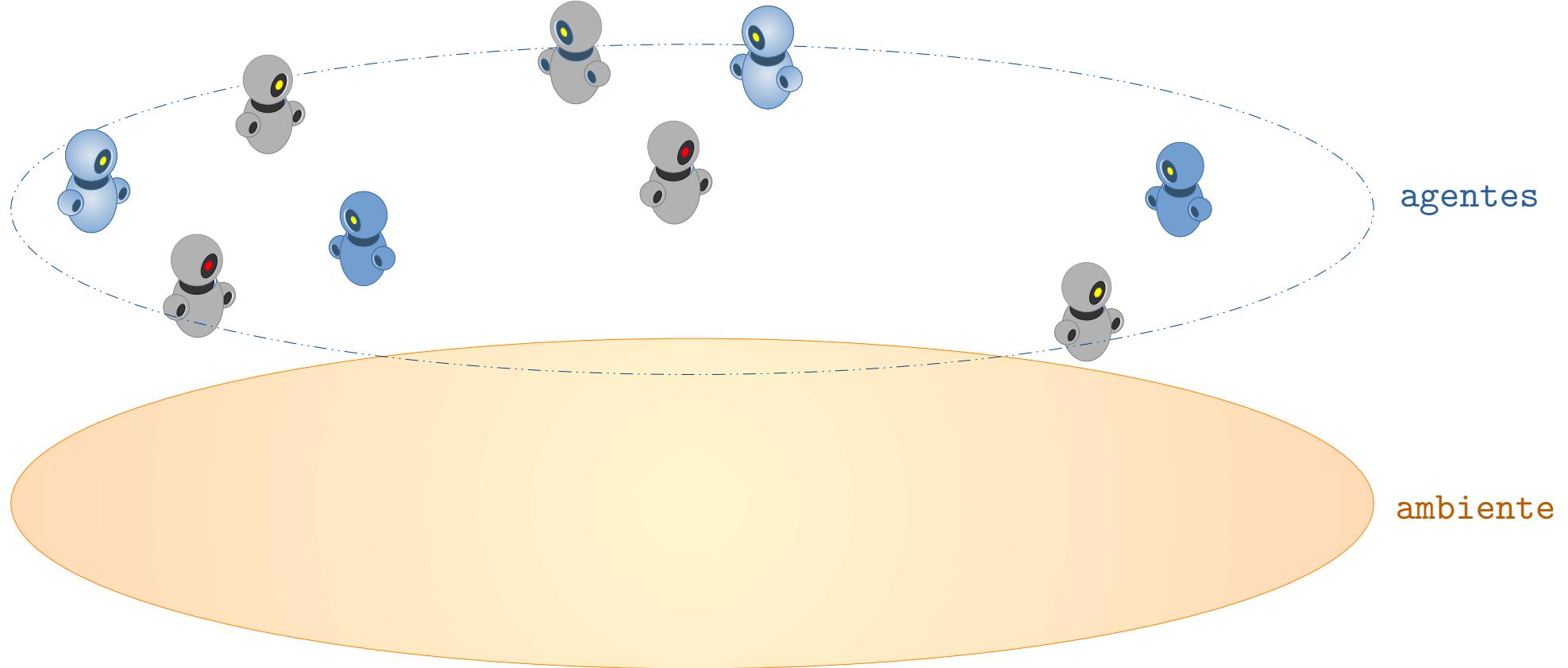
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



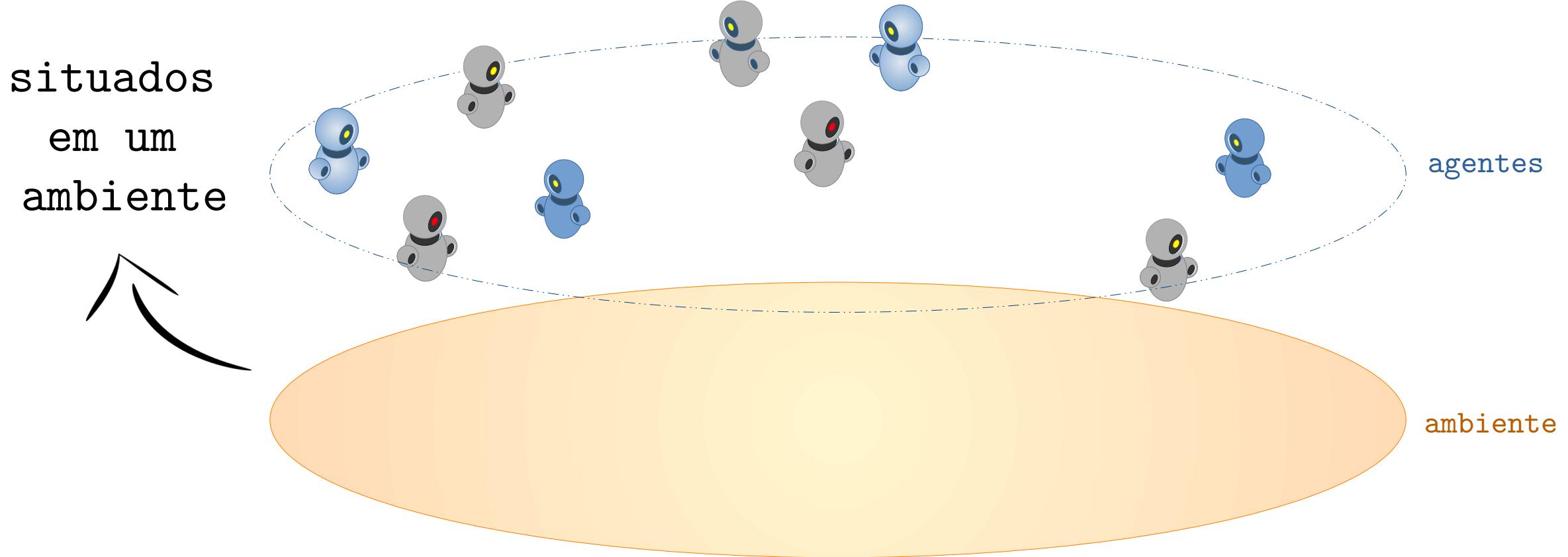
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



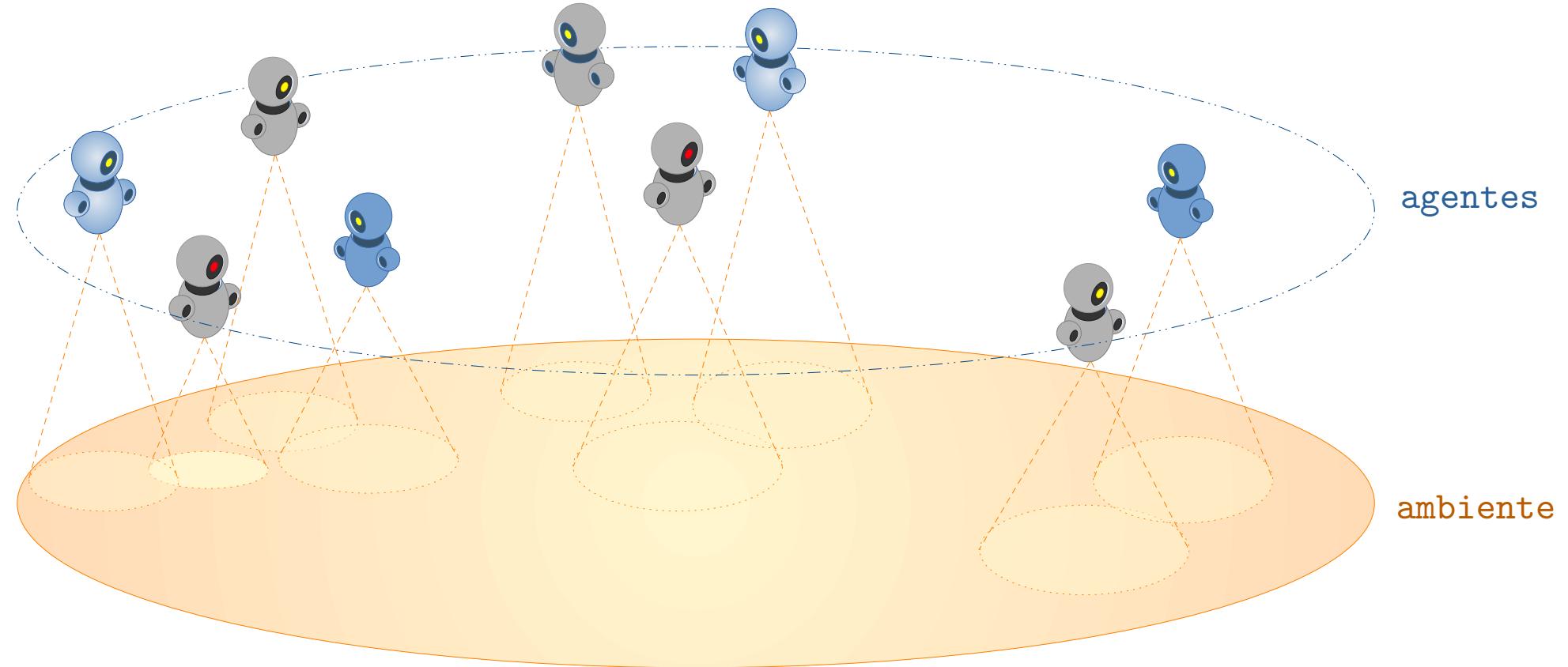
WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



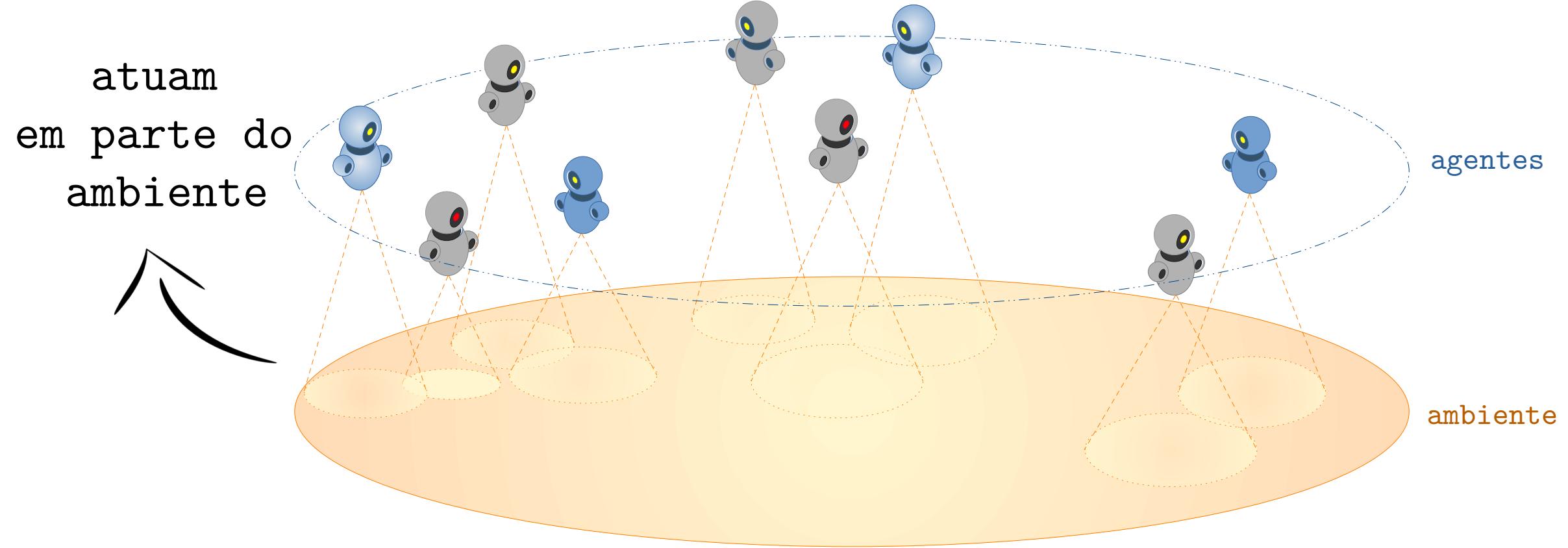
WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



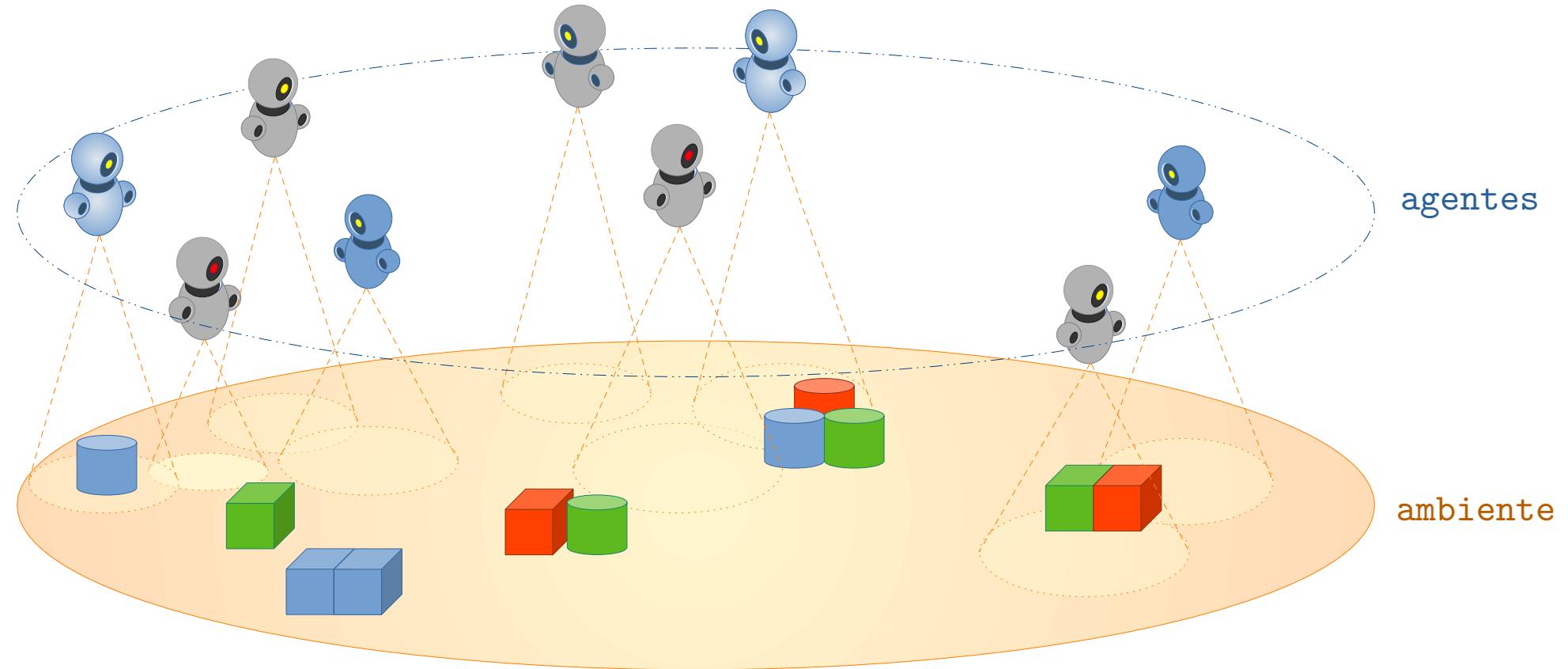
WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

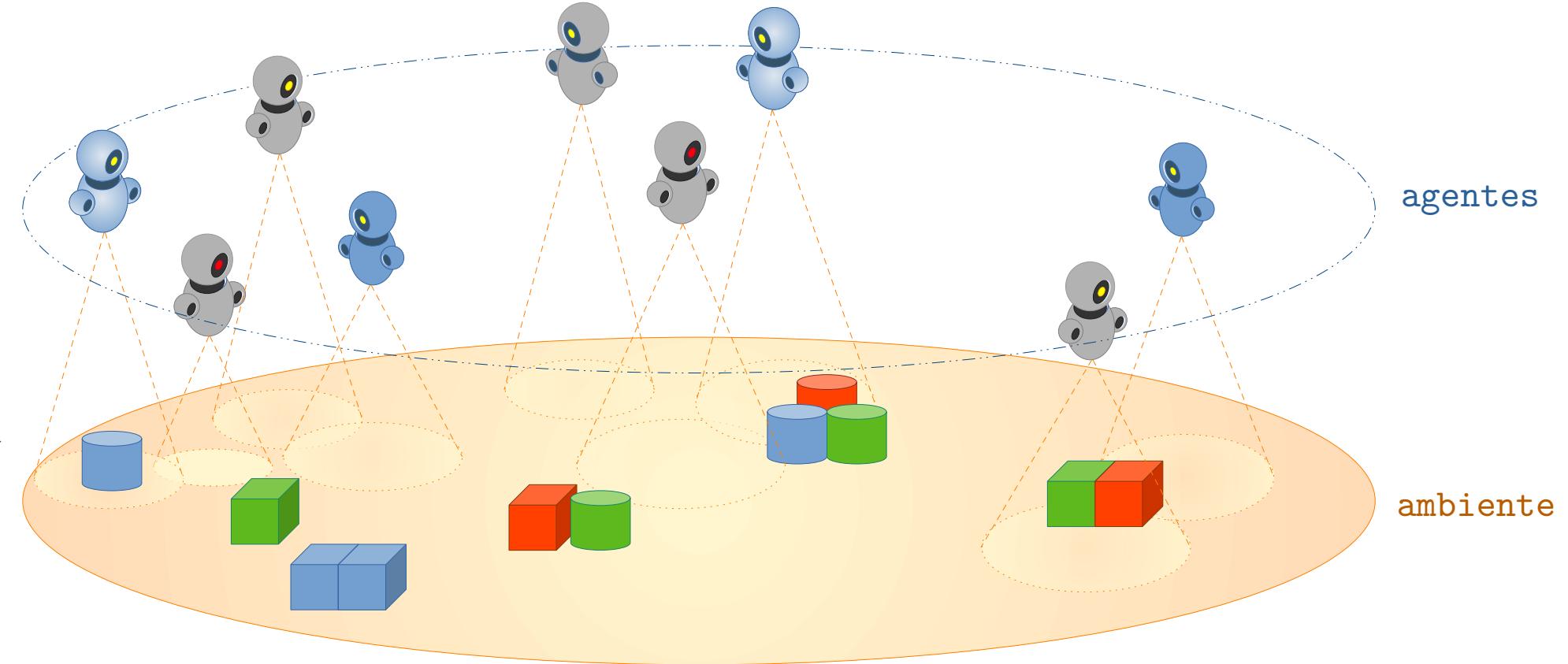
As Dimensões de Sistemas Multiagentes (SMA)



WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

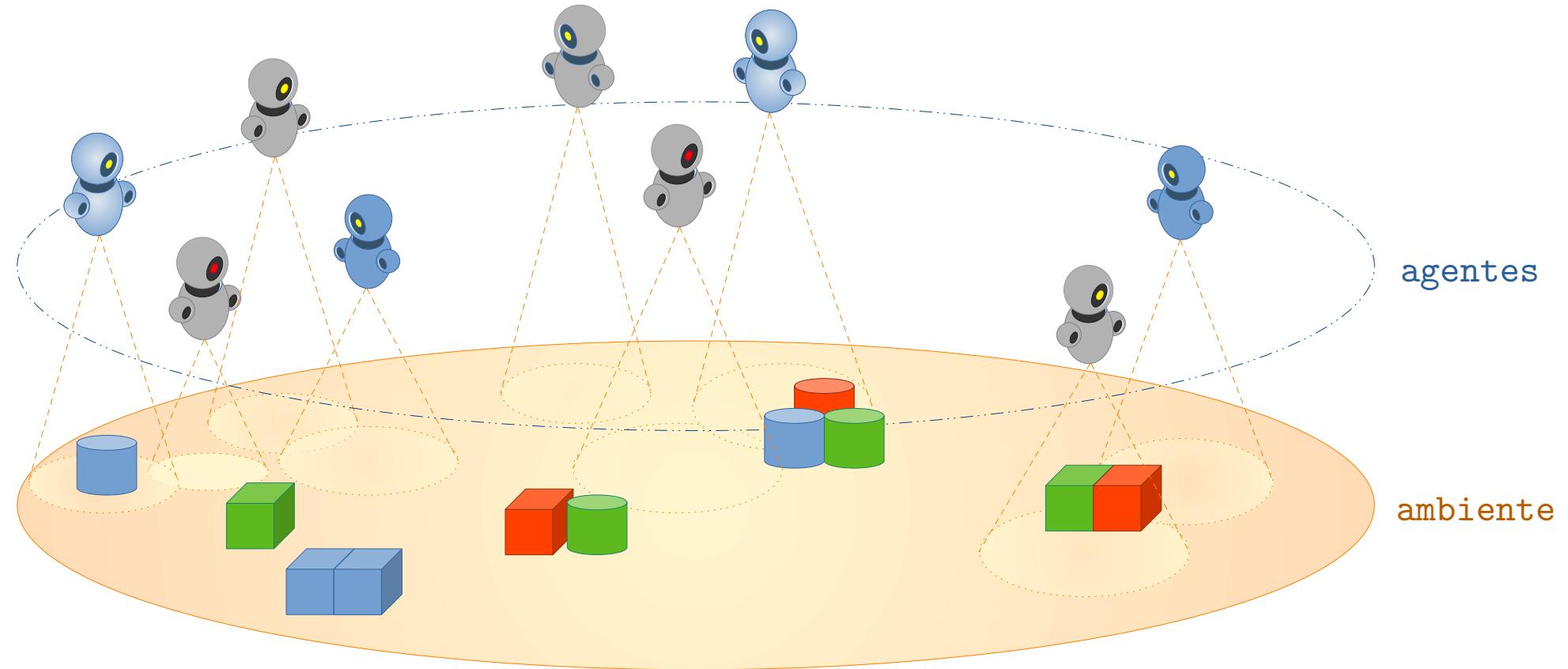
As Dimensões de Sistemas Multiagentes (SMA)

podem
acessar
artefatos



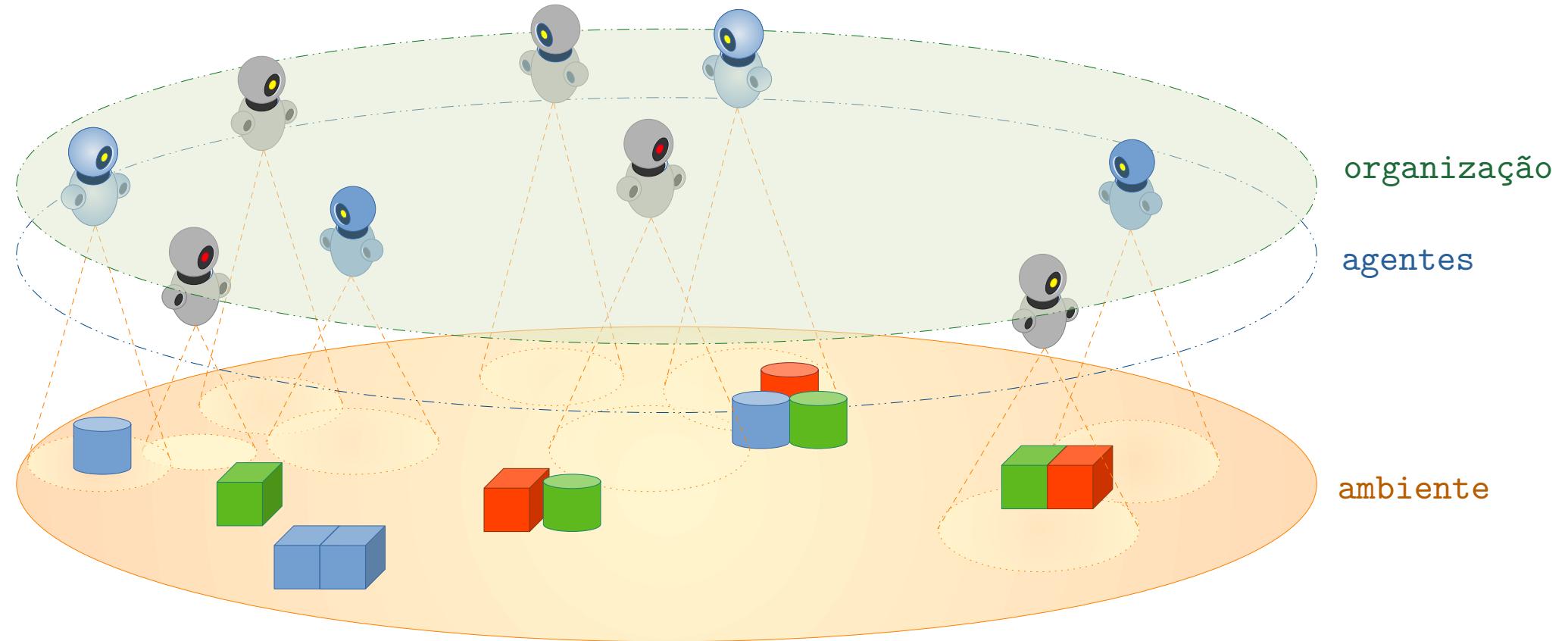
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

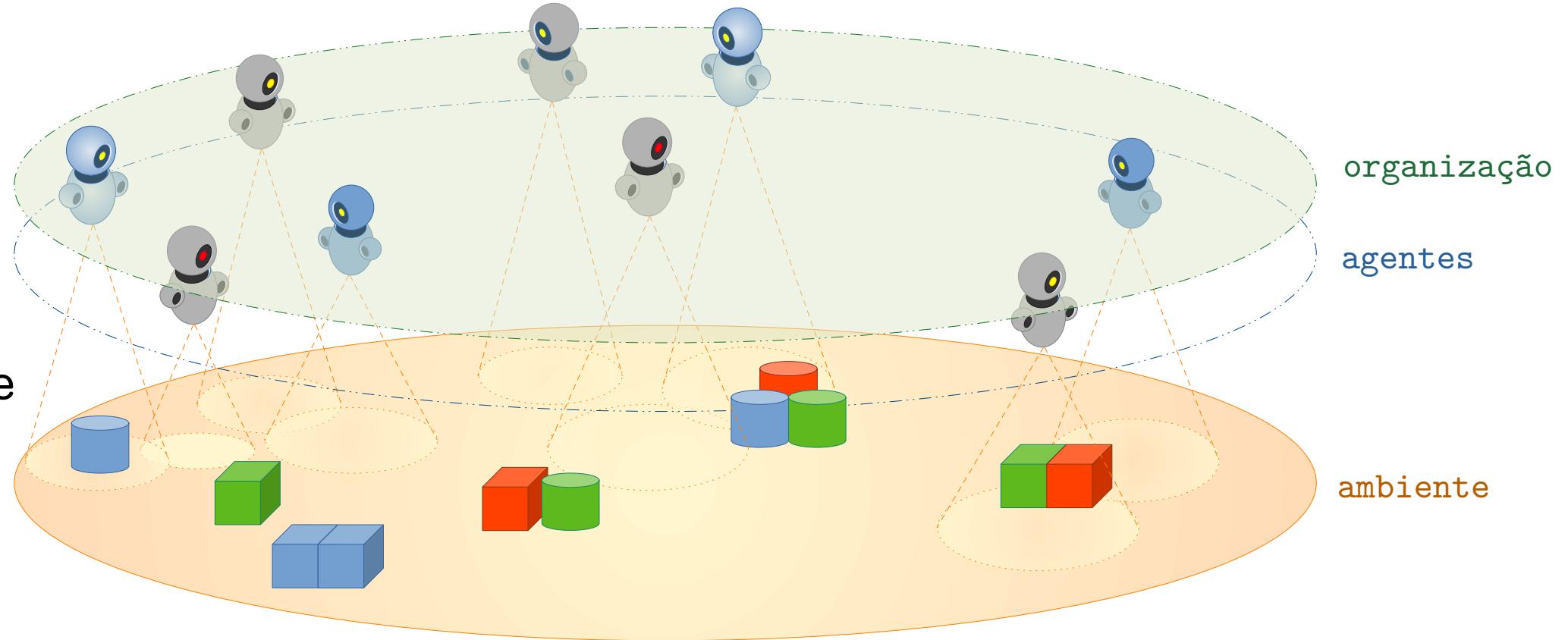
As Dimensões de Sistemas Multiagentes (SMA)



WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

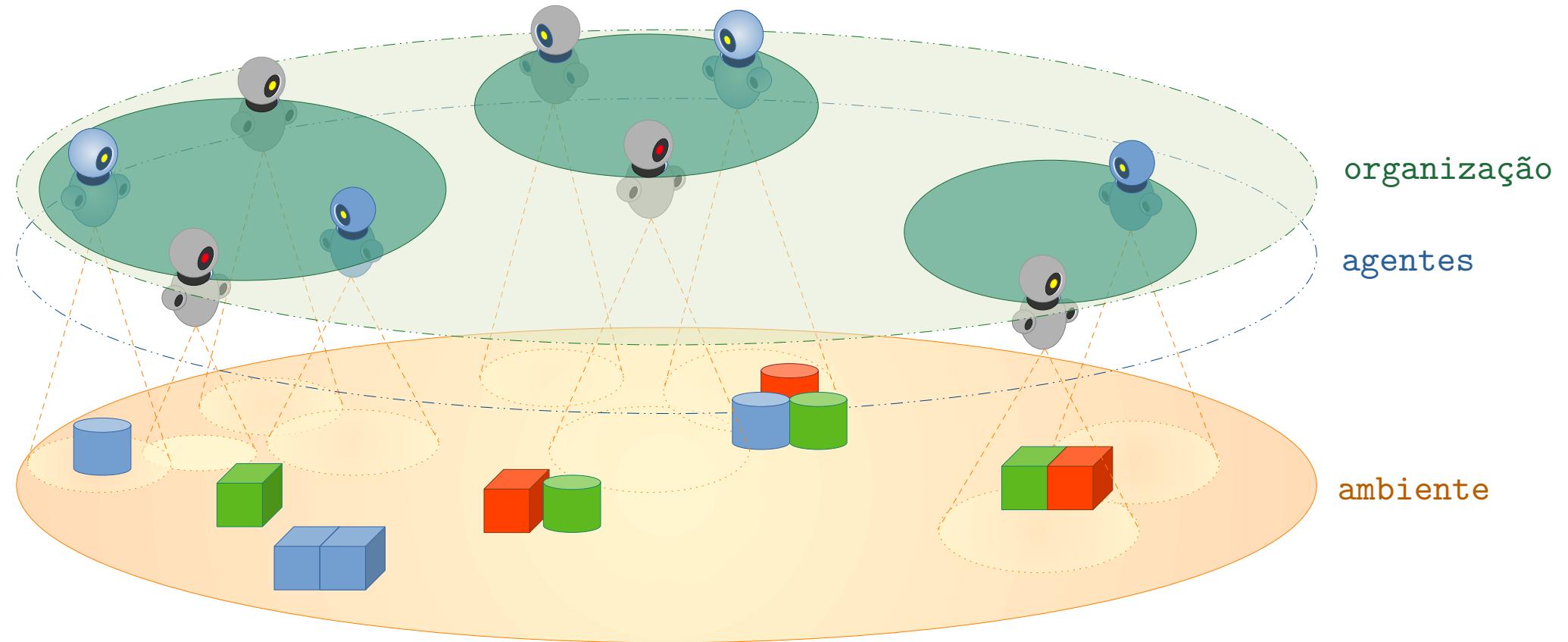
As Dimensões de Sistemas Multiagentes (SMA)

fazem parte
de uma
sociedade



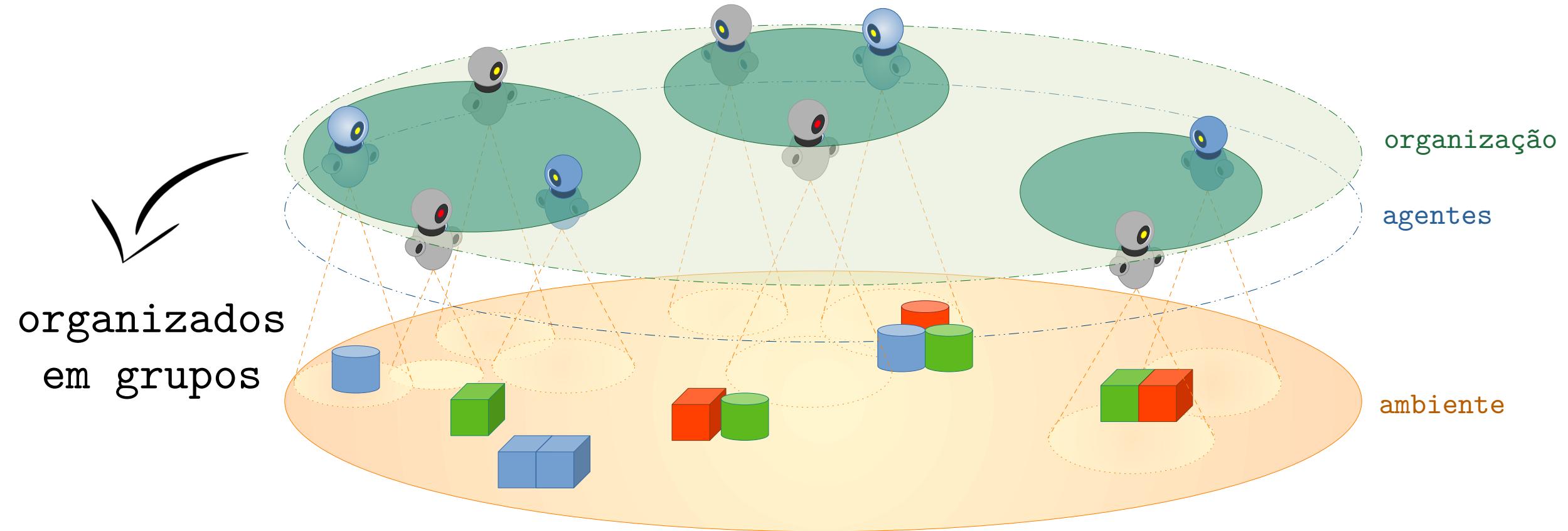
WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



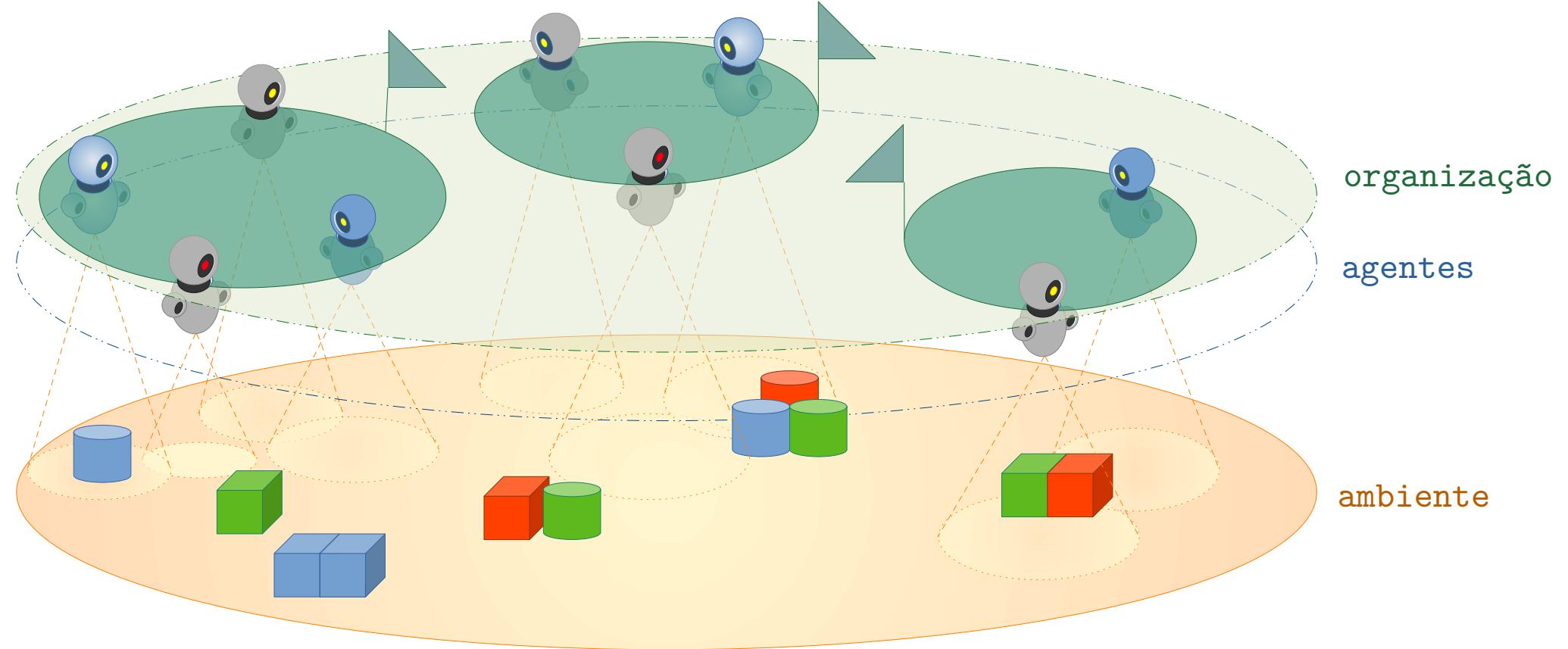
WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



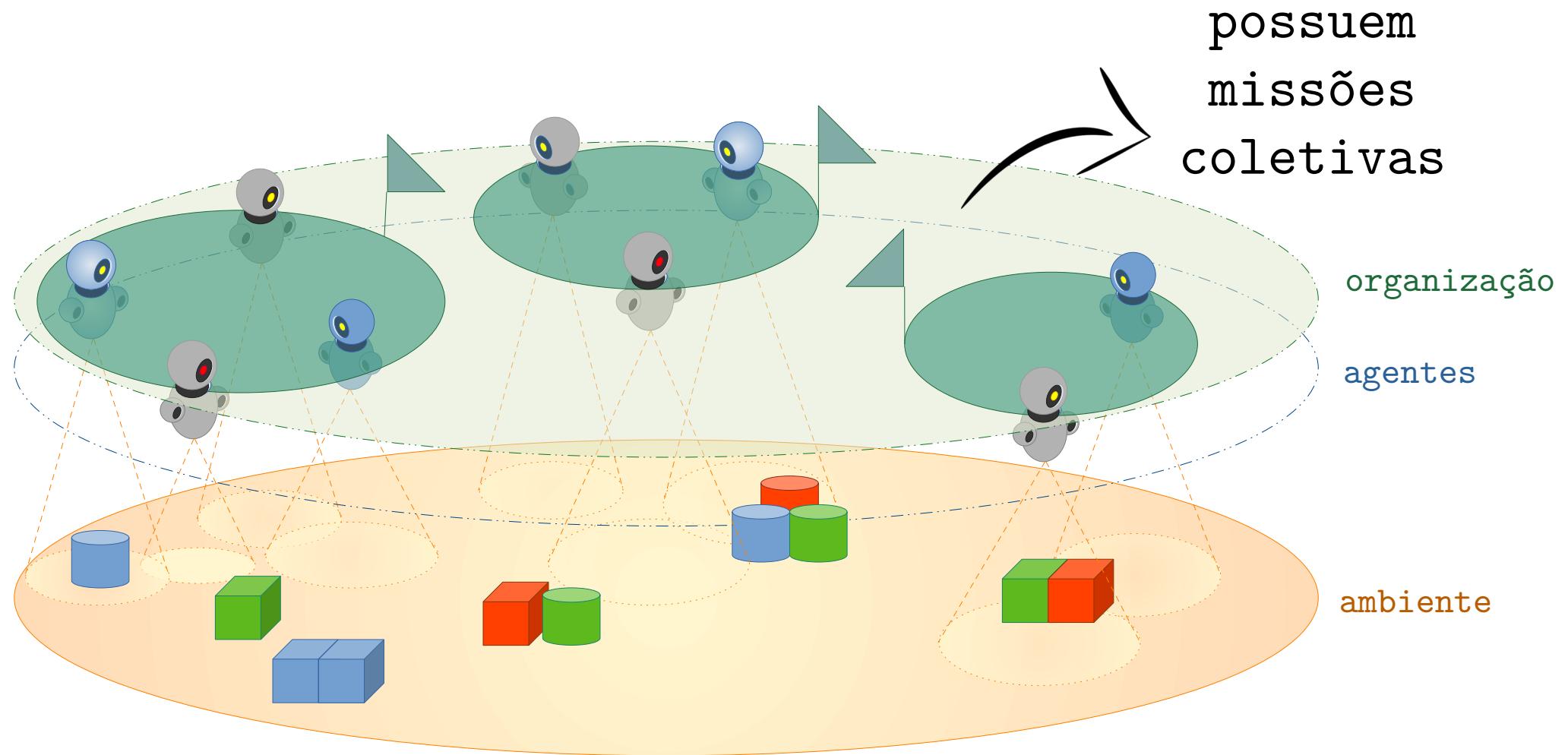
WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



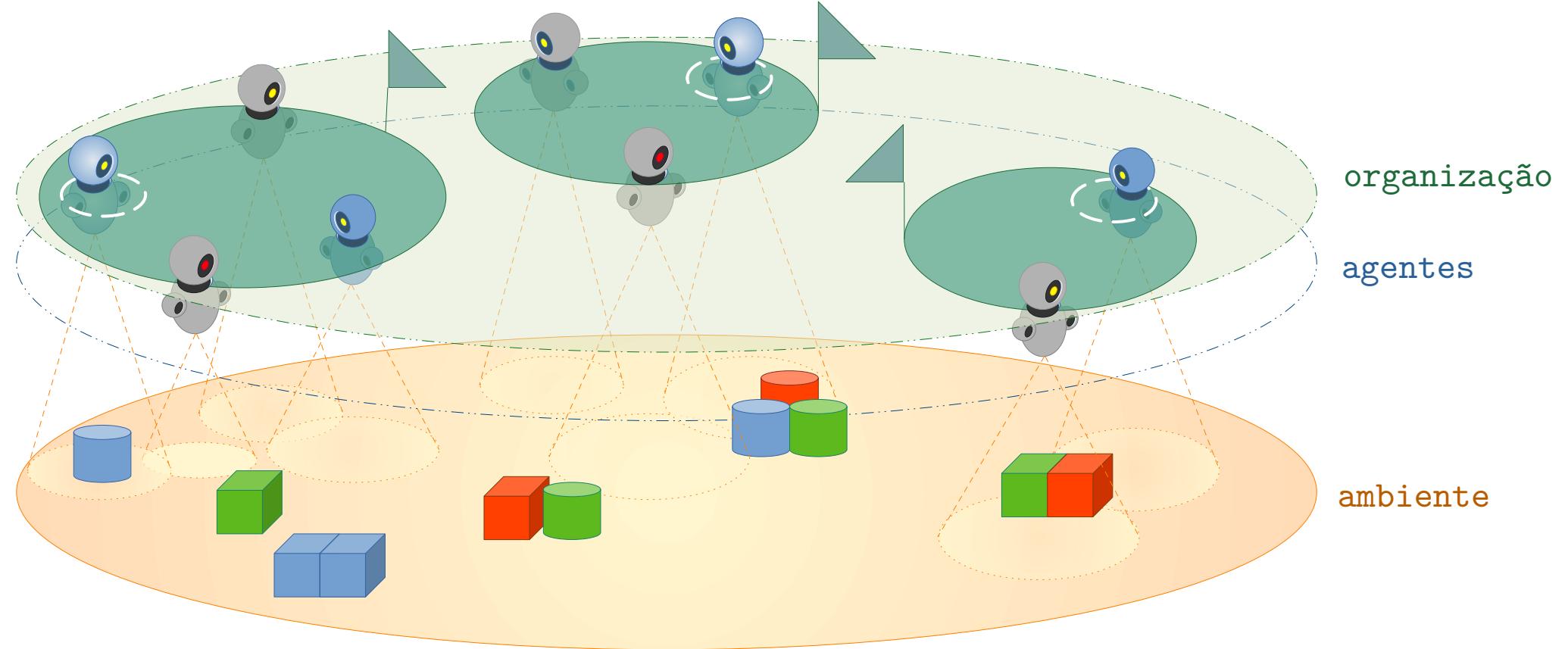
WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



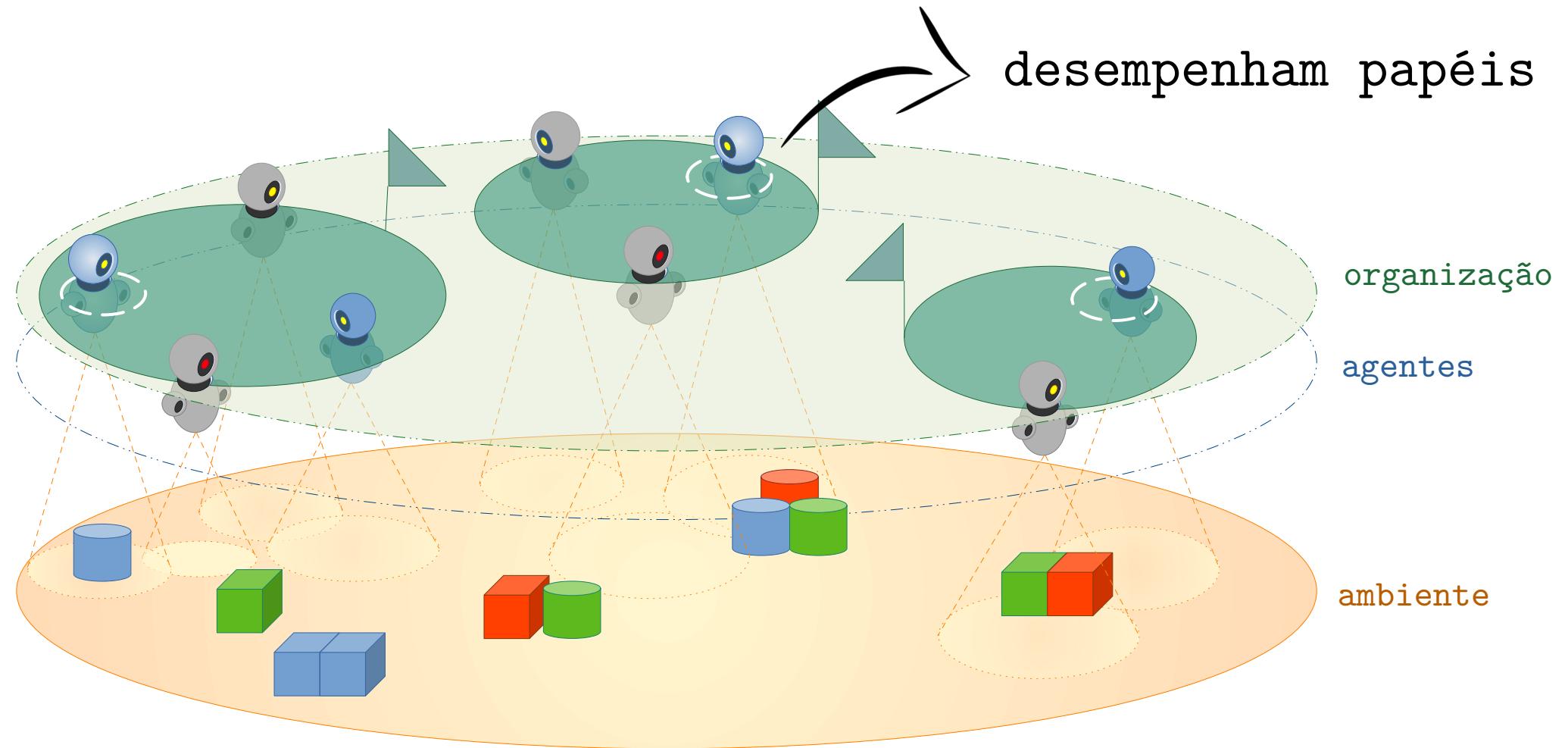
WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

As Dimensões de Sistemas Multiagentes (SMA)



WOOLDRIDGE, Michael. Intelligent Agents. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

Dimensão do Agente

Os agentes são componentes autônomos e cognitivos, originados da inteligência artificial, situados em um ambiente e possuem um conjunto de planos em resposta aos estímulos percebidos, para atingir seus objetivos de projeto e modificar o ambiente em que estão inseridos.

WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77

Características dos Agentes

Agentes são entidades autônomas da Inteligência Artificial

- **Autonomia:** trabalha de forma independente para atingir seus objetivos. Um agente toma decisões que estão sob seu próprio poder de controle, sem necessidade de intervenção externa para cumprir seu objetivo.

WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Características dos Agentes

Agentes são entidades autônomas da Inteligência Artificial

- **Autonomia:** trabalha de forma independente para atingir seus objetivos. Um agente toma decisões que estão sob seu próprio poder de controle, sem necessidade de intervenção externa para cumprir seu objetivo.
- **Pró-atividade:** se comporta direcionado a metas. Quando um agente tem um objetivo, ele próprio tentará executar planos para cumpri-lo de forma ativa. Ele não aguarda uma chamada para executar seus planos, porém os executa de forma proativa, à medida que seu estado mental é modificado.

WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Características dos Agentes

- **Reatividade:** responde às alterações do ambiente de maneira responsável. A modificação de estado em um ambiente provoca uma percepção ao agente, permitindo que este agente delibre em função desta alteração. Esta decisão pode ser totalmente reativa (equivalente aos reflexos humanos), ou mais elaborada (o que equivale a planos do cotidiano de um ser humano).

WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

Características dos Agentes

- **Reatividade:** responde às alterações do ambiente de maneira responsável. A modificação de estado em um ambiente provoca uma percepção ao agente, permitindo que este agente delibere em função desta alteração. Esta decisão pode ser totalmente reativa (equivalente aos reflexos humanos), ou mais elaborada (o que equivale a planos do cotidiano de um ser humano).
- **Habilidade Social:** interage com outros agentes para cooperar ou coordenar atividades que ajudem a atingir os objetivos no sistema. Além disso, habilidades sociais tratam de troca de conhecimento (crenças) entre agentes.

WOOLDRIDGE, Michael. Intelligent Agents. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1999. p. 27–77
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. 2009

A dimensão ambiental do SMA representa a noção de mundo dos agentes.

RICCI, A., VIROLI, M., OMICINI, A. (2007). CArtAgO: A Framework for Prototyping Artifact-Based Environments in MAS. In: Weyns, D., Parunak, H.V.D., Michel, F. (eds) Environments for Multi-Agent Systems III. E4MAS 2006. Lecture Notes in Computer Science(), vol 4389. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-71103-2_4

Dimensão do Ambiente

- um ambiente é composto de ferramentas que os agentes podem explorar em tempo de execução para realizar suas atividades;

RICCI, A., VIROLI, M., OMICINI, A. (2007). CArtAgO: A Framework for Prototyping Artifact-Based Environments in MAS. In: Weyns, D., Parunak, H.V.D., Michel, F. (eds) Environments for Multi-Agent Systems III. E4MAS 2006. Lecture Notes in Computer Science(), vol 4389. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-71103-2_4

Dimensão do Ambiente

- um ambiente é composto de ferramentas que os agentes podem explorar em tempo de execução para realizar suas atividades;
- agrupamento das ferramentas em grupos de trabalho.

RICCI, A., VIROLI, M., OMICINI, A. (2007). CArtAgO: A Framework for Prototyping Artifact-Based Environments in MAS. In: Weyns, D., Parunak, H.V.D., Michel, F. (eds) Environments for Multi-Agent Systems III. E4MAS 2006. Lecture Notes in Computer Science(), vol 4389. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-71103-2_4

Endógeno e Exógeno

- **Endógeno.** é aquele contido dentro do SMA e que modela as ferramentas que serão exploradas pelos agentes. Ele modela o ambiente exógeno dentro do contexto do SMA;

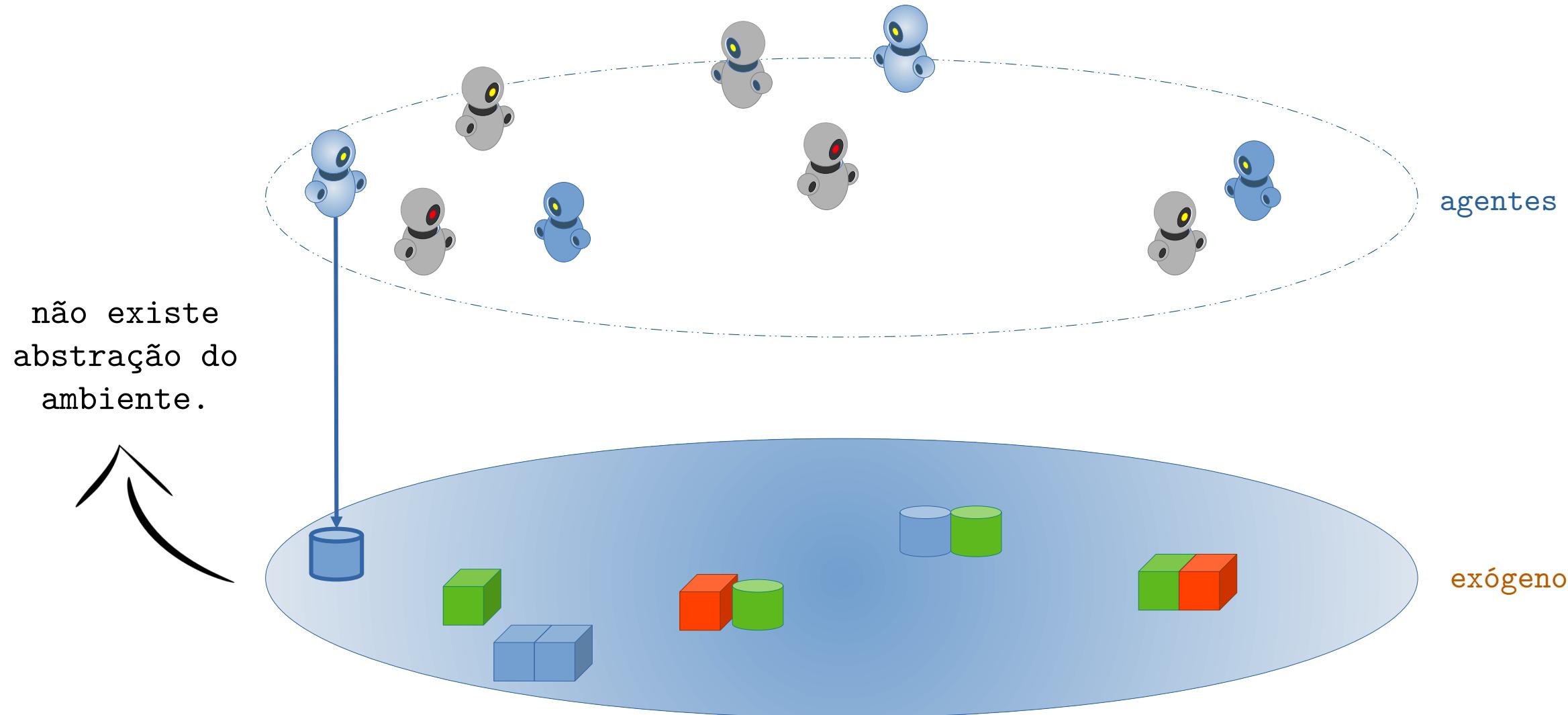
HUBNER, J.F., BOISSIER, O., KITIO, R. et al. Instrumenting multi-agent organisations with organisational artifacts and agents. *Auton Agent Multi-Agent Syst* 20, 369–400 (2010).
<https://doi.org/10.1007/s10458-009-9084-y>

Endógeno e Exógeno

- **Endógeno.** é aquele contido dentro do SMA e que modela as ferramentas que serão exploradas pelos agentes. Ele modela o ambiente exógeno dentro do contexto do SMA;
- **Exógeno.** aquele que é percebido e afetado pelo agente no contexto real. Representa o ambiente real (físico ou virtual) que impacta e é impactado pelo agente.

HUBNER, J.F., BOISSIER, O., KITIO, R. et al. Instrumenting multi-agent organisations with organisational artifacts and agents. *Auton Agent Multi-Agent Syst* 20, 369–400 (2010).
<https://doi.org/10.1007/s10458-009-9084-y>

Abordagem utilizada: Atuação Direta no Exógeno



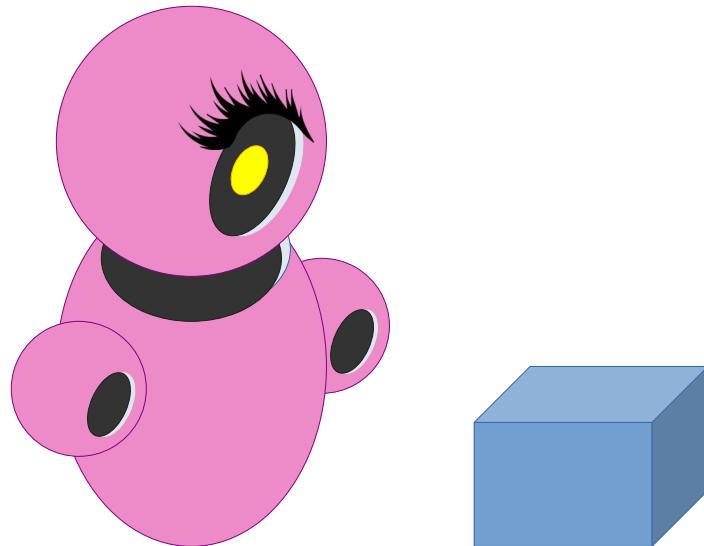
Artefatos

No ambiente endógeno, as ferramentas utilizadas pelos agentes são conhecidas como artefatos [Conte et al., 2016]

[Omicini et al., 2005].
[Conte et al., 2016]

Artefatos

No ambiente endógeno, as ferramentas utilizadas pelos agentes são conhecidas como artefatos [Conte et al., 2016]



[Conte et al., 2016]

Artefatos

No ambiente endógeno, as ferramentas utilizadas pelos agentes são conhecidas como artefatos [Conte et al., 2016]



[Conte et al., 2016]

Artefatos

No ambiente endógeno, as ferramentas utilizadas pelos agentes são conhecidas como artefatos [Conte et al., 2016]



[Conte et al., 2016]

Artefatos

Artefatos

No ambiente endógeno, as ferramentas utilizadas pelos agentes são conhecidas como artefatos [Conte et al., 2016]

- são dispositivos computacionais e não cognitivos;

Omicini, A., Ricci, A., Viroli, M. (2006). Coordination Artifacts as First-Class Abstractions for MAS Engineering: State of the Research. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds) Software Engineering for Multi-Agent Systems IV. SELMAS 2005. Lecture Notes in Computer Science, vol 3914. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11738817_5

[Conte et al., 2016]

Artefatos

No ambiente endógeno, as ferramentas utilizadas pelos agentes são conhecidas como artefatos [Conte et al., 2016]

- são dispositivos computacionais e não cognitivos;
- possuem determinada função ou serviço que podem ser explorados;

Omicini, A., Ricci, A., Viroli, M. (2006). Coordination Artifacts as First-Class Abstractions for MAS Engineering: State of the Research. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds) Software Engineering for Multi-Agent Systems IV. SELMAS 2005. Lecture Notes in Computer Science, vol 3914. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11738817_5

[Conte et al., 2016]

Artefatos

No ambiente endógeno, as ferramentas utilizadas pelos agentes são conhecidas como artefatos [Conte et al., 2016]

- são dispositivos computacionais e não cognitivos;
- possuem determinada função ou serviço que podem ser explorados;
- a exploração dos artefatos pode ser motivada pelos objetivos sociais ou individuais de um agente;

Omicini, A., Ricci, A., Viroli, M. (2006). Coordination Artifacts as First-Class Abstractions for MAS Engineering: State of the Research. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds) Software Engineering for Multi-Agent Systems IV. SELMAS 2005. Lecture Notes in Computer Science, vol 3914. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11738817_5

[Conte et al., 2016]

Artefatos

Artefatos

No ambiente endógeno, as ferramentas utilizadas pelos agentes são conhecidas como artefatos [Conte et al., 2016]

- são dispositivos computacionais e não cognitivos;
- possuem determinada função ou serviço que podem ser explorados;
- a exploração dos artefatos pode ser motivada pelos objetivos sociais ou individuais de um agente;
- podem ser selecionados, utilizados ou construídos, em casos que nenhum artefato útil foi encontrado no seu ambiente.

Omicini, A., Ricci, A., Viroli, M. (2006). Coordination Artifacts as First-Class Abstractions for MAS Engineering: State of the Research. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds) Software Engineering for Multi-Agent Systems IV. SELMAS 2005. Lecture Notes in Computer Science, vol 3914. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11738817_5

[Conte et al., 2016]

Artefatos

Os artefatos são compostos por quatro elementos [Ricci et al., 2005]:

Omicini, A., Ricci, A., Viroli, M. (2006). Coordination Artifacts as First-Class Abstractions for MAS Engineering: State of the Research. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds) Software Engineering for Multi-Agent Systems IV. SELMAS 2005. Lecture Notes in Computer Science, vol 3914. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11738817_5
[Ricci et al., 2005].

Artefatos

Os artefatos são compostos por quatro elementos [Ricci et al., 2005] :

- **interface de uso (UI)**. que é um conjunto das operações que os agentes podem realizar através dos artefatos;

Omicini, A., Ricci, A., Viroli, M. (2006). Coordination Artifacts as First-Class Abstractions for MAS Engineering: State of the Research. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds) Software Engineering for Multi-Agent Systems IV. SELMAS 2005. Lecture Notes in Computer Science, vol 3914. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11738817_5

[Ricci et al., 2005].

Artefatos

Os artefatos são compostos por quatro elementos [Ricci et al., 2005] :

- **interface de uso (UI)**. que é um conjunto das operações que os agentes podem realizar através dos artefatos;
- **instruções operacionais (OI) ou manuais**. que descrevem como o artefato deve ser usado para acessar suas funcionalidades;

Omicini, A., Ricci, A., Viroli, M. (2006). Coordination Artifacts as First-Class Abstractions for MAS Engineering: State of the Research. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds) Software Engineering for Multi-Agent Systems IV. SELMAS 2005. Lecture Notes in Computer Science, vol 3914. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11738817_5
[Ricci et al., 2005].

Artefatos

Os artefatos são compostos por quatro elementos [Ricci et al., 2005] :

- **interface de uso (UI)**. que é um conjunto das operações que os agentes podem realizar através dos artefatos;
- **instruções operacionais (OI) ou manuais**. que descrevem como o artefato deve ser usado para acessar suas funcionalidades;
- **função**. que é o objetivo da existência do artefato;

Omicini, A., Ricci, A., Viroli, M. (2006). Coordination Artifacts as First-Class Abstractions for MAS Engineering: State of the Research. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds) Software Engineering for Multi-Agent Systems IV. SELMAS 2005. Lecture Notes in Computer Science, vol 3914. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11738817_5

[Ricci et al., 2005].

Artefatos

Os artefatos são compostos por quatro elementos [Ricci et al., 2005] :

- **interface de uso (UI)**. que é um conjunto das operações que os agentes podem realizar através dos artefatos;
- **instruções operacionais (OI) ou manuais**. que descrevem como o artefato deve ser usado para acessar suas funcionalidades;
- **função**. que é o objetivo da existência do artefato;
- **estrutura/comportamento**. que são as características internas dos artefatos que definem como ele é implementado.

Omicini, A., Ricci, A., Viroli, M. (2006). Coordination Artifacts as First-Class Abstractions for MAS Engineering: State of the Research. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds) Software Engineering for Multi-Agent Systems IV. SELMAS 2005. Lecture Notes in Computer Science, vol 3914. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11738817_5

[Ricci et al., 2005].

Dimensão Organizacional

A dimensão de organização é responsável por alinhar os objetivos individuais dos agentes com os da organização na qual estão inseridos.

[Hannoun et al., 2000].

Dimensão Organizacional

O modelo organizacional se baseia:

[Hannoun et al., 2000].

Dimensão Organizacional

O modelo organizacional se baseia:

- na composição de um conjunto de regras que restringem o comportamento dos agentes;

[Hannoun et al., 2000].

Dimensão Organizacional

O modelo organizacional se baseia:

- na composição de um conjunto de regras que restringem o comportamento dos agentes;
- um agente se torna parte de uma organização com mútuas obrigações, proibições e permissões.

[Hannoun et al., 2000].

Componentes Organizacionais

Components

Componentes Organizacionais

Components

- **Scheme.** como um SMA atinge suas metas globais e como estão decompostos em planos e distribuídos em missões para os agentes.

Componentes Organizacionais

Components

- **Scheme.** como um SMA atinge suas metas globais e como estão decompostos em planos e distribuídos em missões para os agentes.
- **Mission.** que são um conjunto de metas globais e planos globais.

Componentes Organizacionais

Components

- **Scheme.** como um SMA atinge suas metas globais e como estão decompostos em planos e distribuídos em missões para os agentes.
- **Mission.** que são um conjunto de metas globais e planos globais.
- **Group.** Uma organização de agentes.

Componentes Organizacionais

Components

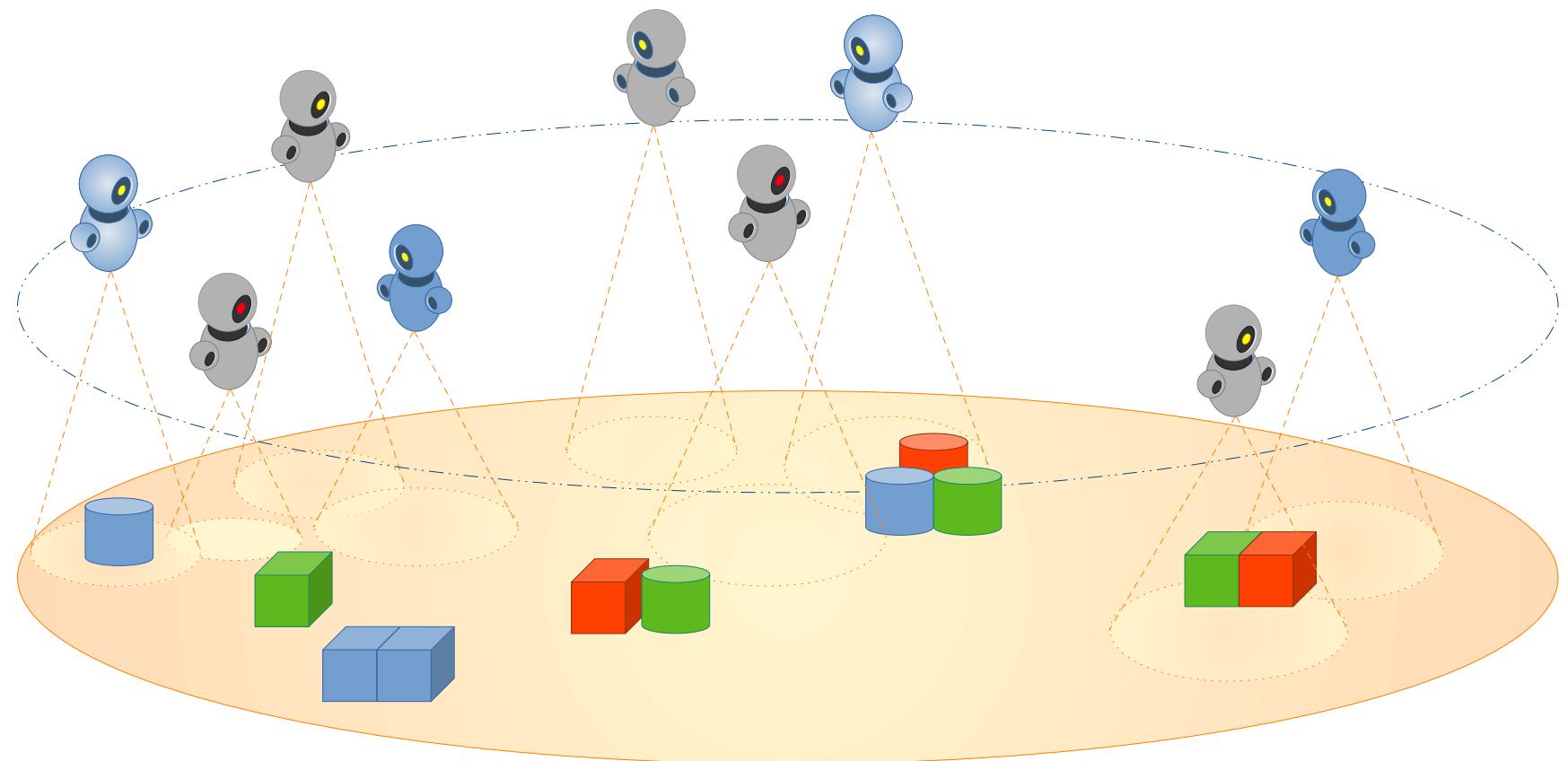
- **Scheme.** como um SMA atinge suas metas globais e como estão decompostos em planos e distribuídos em missões para os agentes.
- **Mission.** que são um conjunto de metas globais e planos globais.
- **Group.** Uma organização de agentes.
- **Role.** Papel que um agente pode assumir no sistema.

Componentes Organizacionais

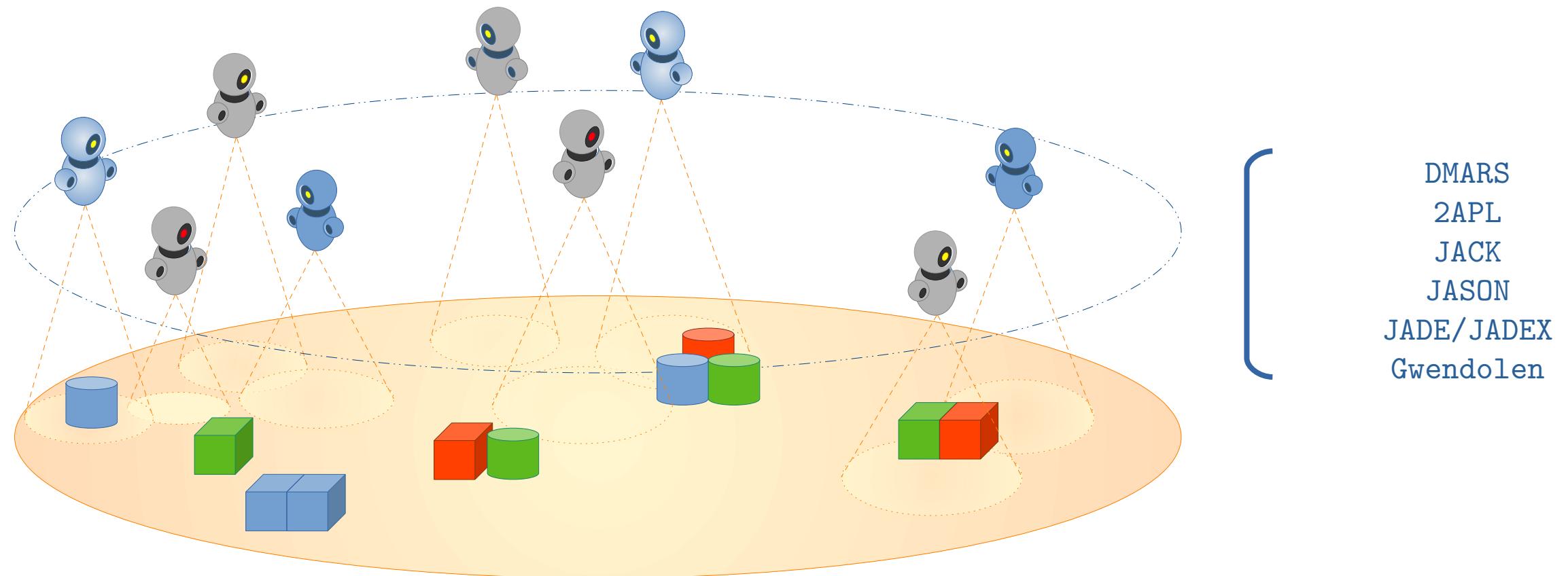
Components

- **Scheme.** como um SMA atinge suas metas globais e como estão decompostos em planos e distribuídos em missões para os agentes.
- **Mission.** que são um conjunto de metas globais e planos globais.
- **Group.** Uma organização de agentes.
- **Role.** Papel que um agente pode assumir no sistema.
- **Norm.** Norma que o agente deve estar de acordo a obedecer no sistema (permissões e obrigações).

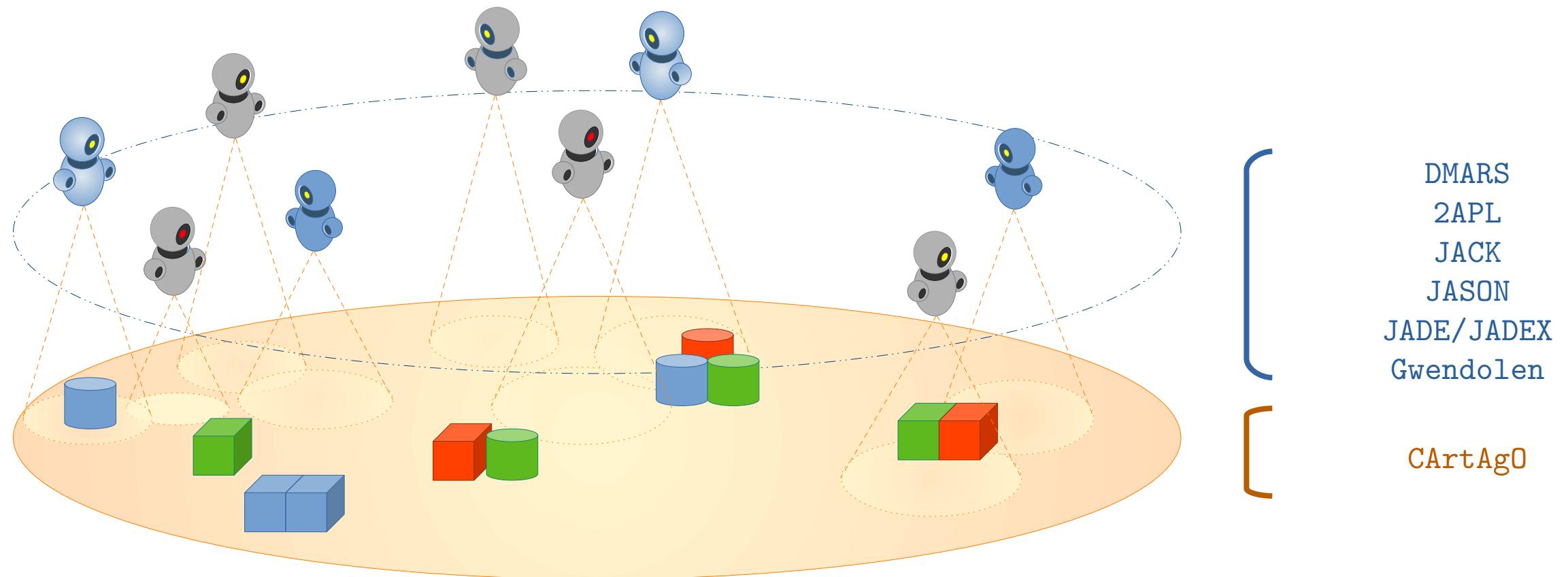
Overview Tecnológico



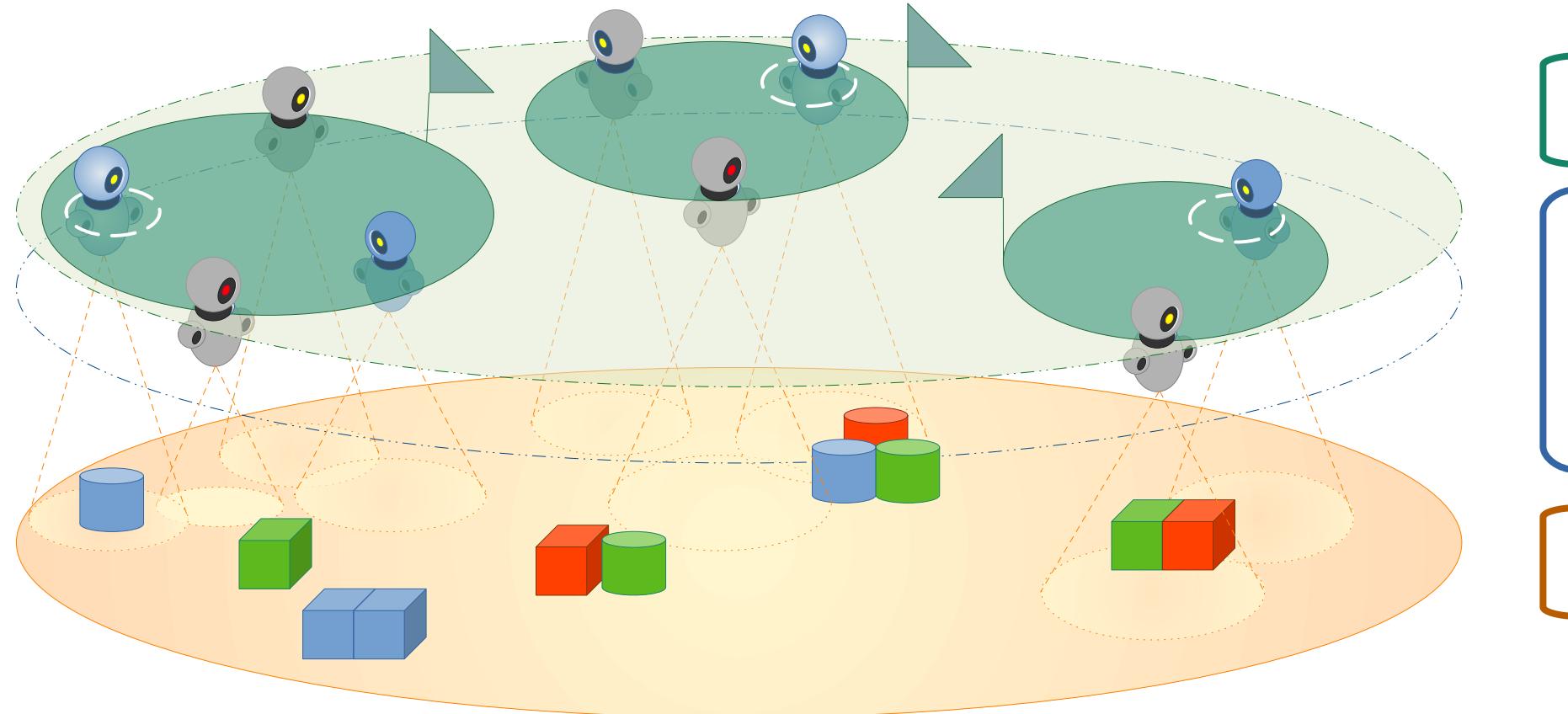
Overview Tecnológico



Overview Tecnológico



Overview Tecnológico

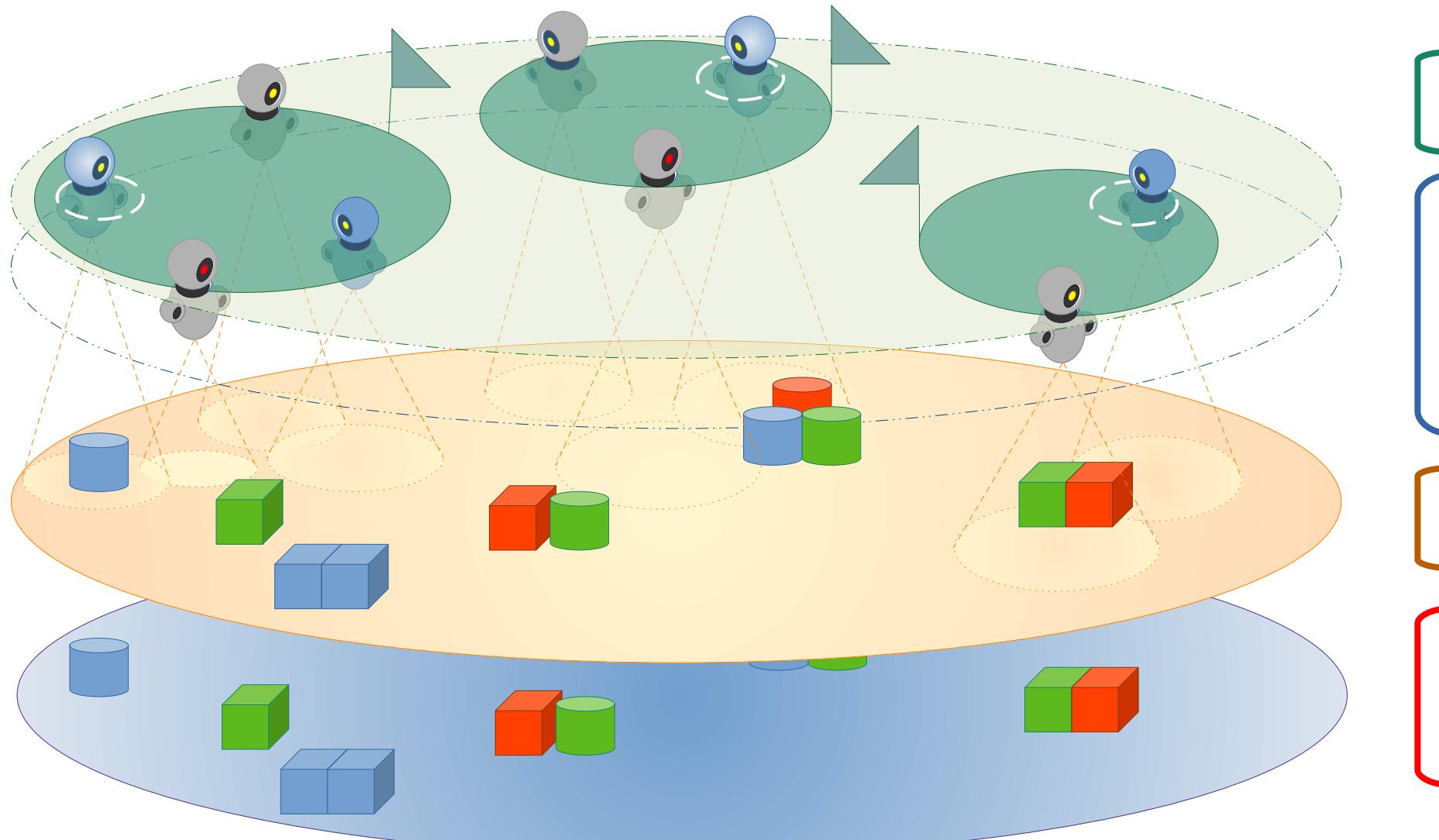


DEPINT
Moise+

DMARS
2APL
JACK
JASON
JADE/JADEX
Gwendolen

CArtAgO

Overview Tecnológico



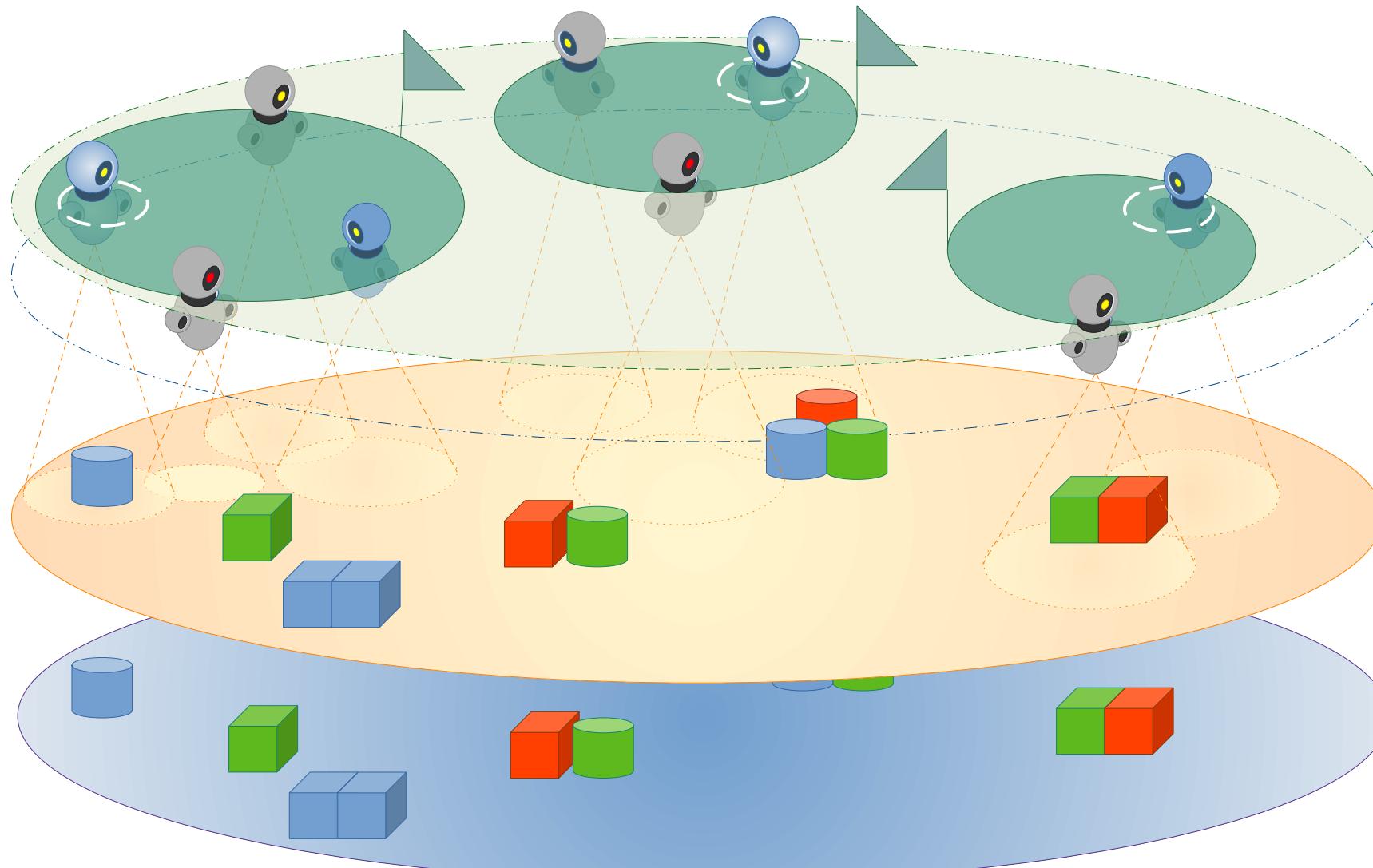
DEPINT
Moise+

DMARS
2APL
JACK
JASON
JADE/JADEX
Gwendolen

CArtAgO

Raspberry
Arduino
Gwendolen
ARGO

Overview Tecnológico



DEPINT
Moise+

DMARS
2APL

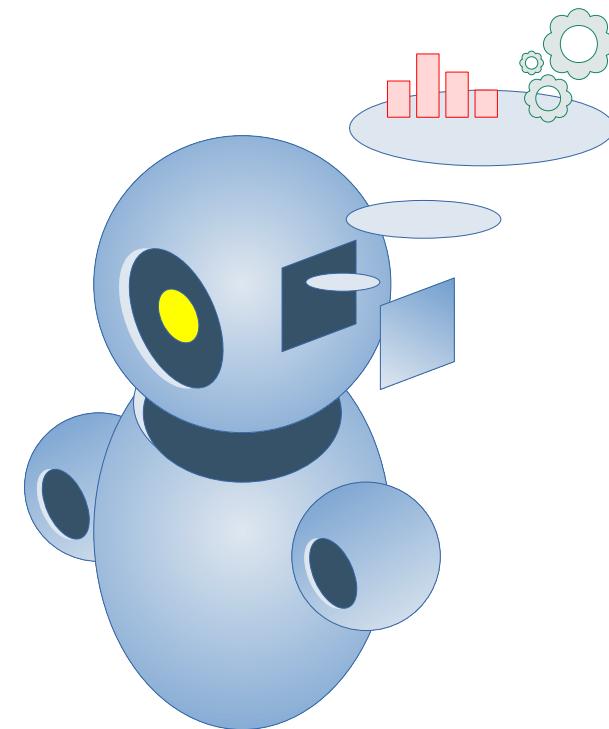
JACK
JASON

JADE/JADEX
Gwendolen

CArtAgO

Raspberry
Arduino
Gwendolen
ARGO

THE BELIEF- DESIRE- INTENTION



Modelos Cogniti

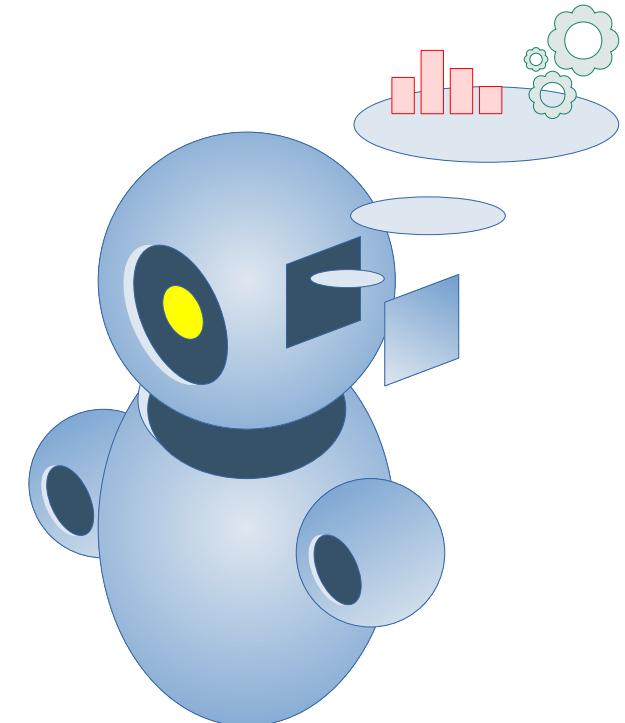
Um **modelo cognitivo** é uma representação simplificada dos processos e estruturas envolvidas nas funções cognitivas, como percepção, memória, aprendizagem, resolução de problemas e tomada de decisões.

MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

Um **modelo cognitivo** é uma representação simplificada dos processos e estruturas envolvidas nas funções cognitivas, como percepção, memória, aprendizagem, resolução de problemas e tomada de decisões.

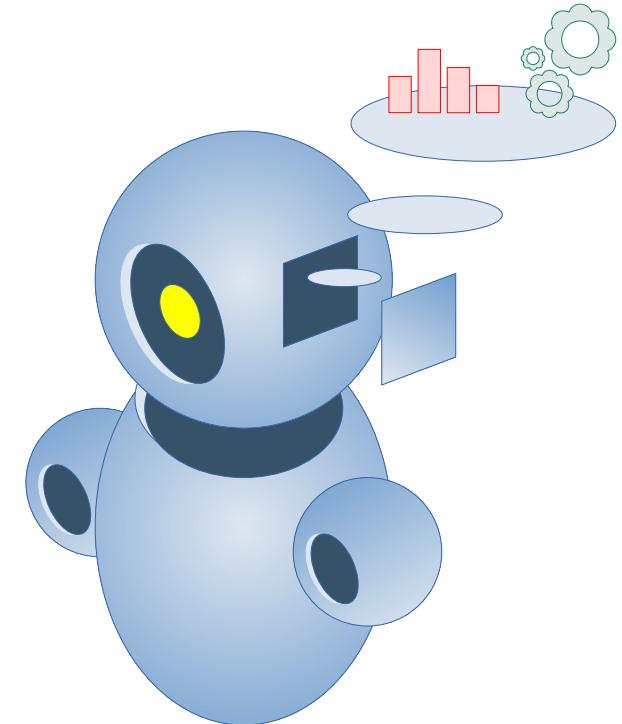


MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. I.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

- É um referencial teórico que ajuda pesquisadores e psicólogos a compreender e simular como funciona a mente humana.

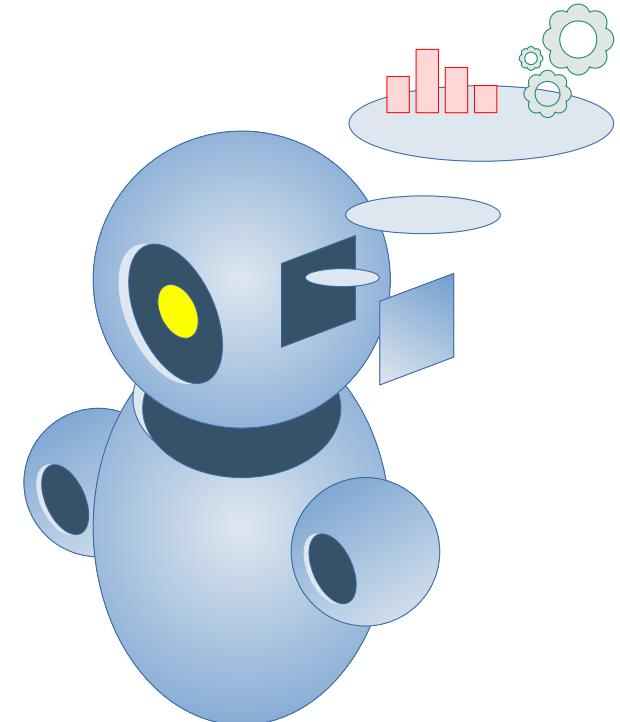


MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

- É um referencial teórico que ajuda pesquisadores e psicólogos a compreender e simular como funciona a mente humana.
- explicam os processos mentais e o comportamento, propondo mecanismos, regras ou algoritmos que fundamentam as atividades cognitivas.



MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

Existem diversos modelos cognitivos:

MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

Existem diversos modelos cognitivos:

- Adaptive Control of Thought - Rational (ACT-R);

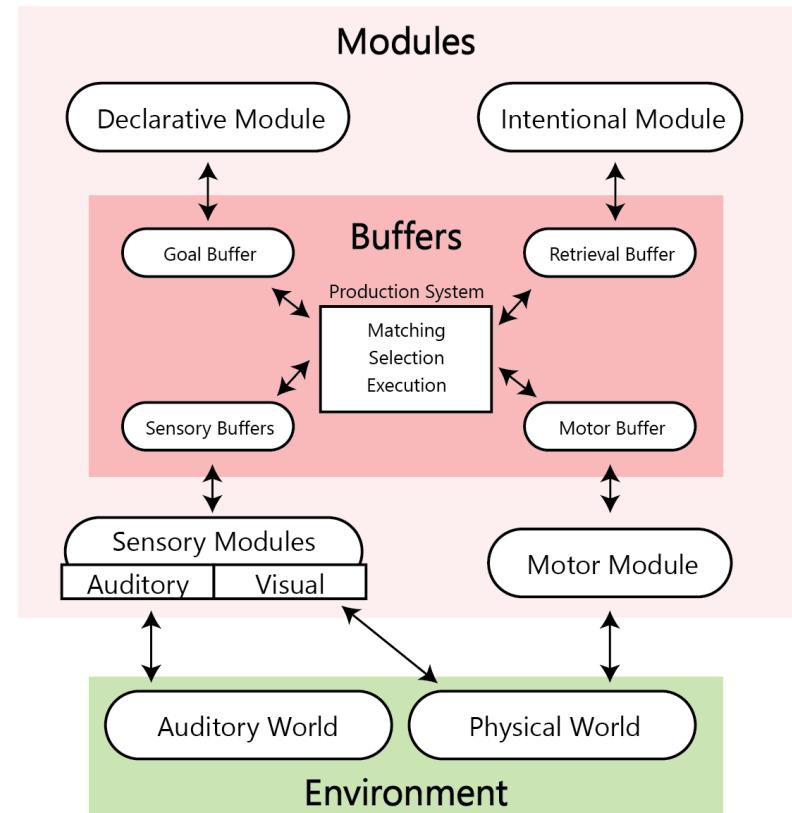
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

Existem diversos modelos cognitivos:

- Adaptive Control of Thought - Rational (ACT-R);



MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. I.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

Existem diversos modelos cognitivos:

- Adaptive Control of Thought - Rational;
- SOAR/Inteligências Multifocal;

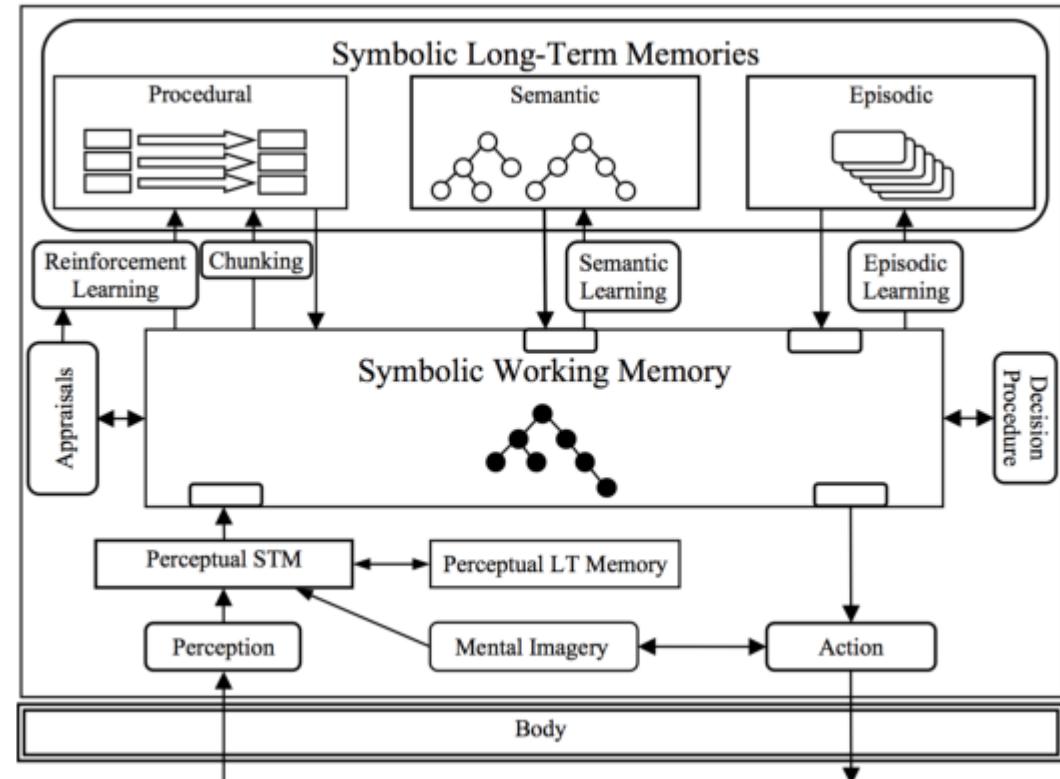
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

Existem diversos modelos cognitivos:

- Adaptive Control of Thought - Rational;
- SOAR/Inteligências Multifocal;



MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications.** [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

Existem diversos modelos cognitivos:

- Adaptive Control of Thought - Rational;
- SOAR/Inteligências Multifocal;
- Belief-Desire-Intention (BDI);

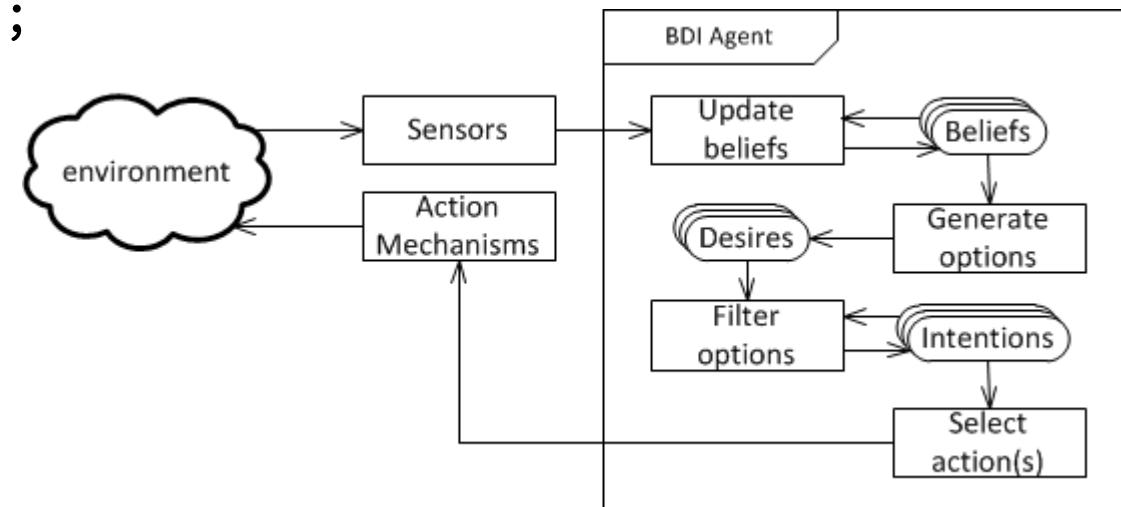
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

Existem diversos modelos cognitivos:

- Adaptive Control of Thought - Rational;
- SOAR/Inteligências Multifocal;
- Belief-Desire-Intention (BDI);



MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

Existem diversos modelos cognitivos:

- Adaptive Control of Thought - Rational;
- SOAR/Inteligências Multifocal;
- Belief-Desire-Intention (BDI);
- Practical Reasoning System (PRS);

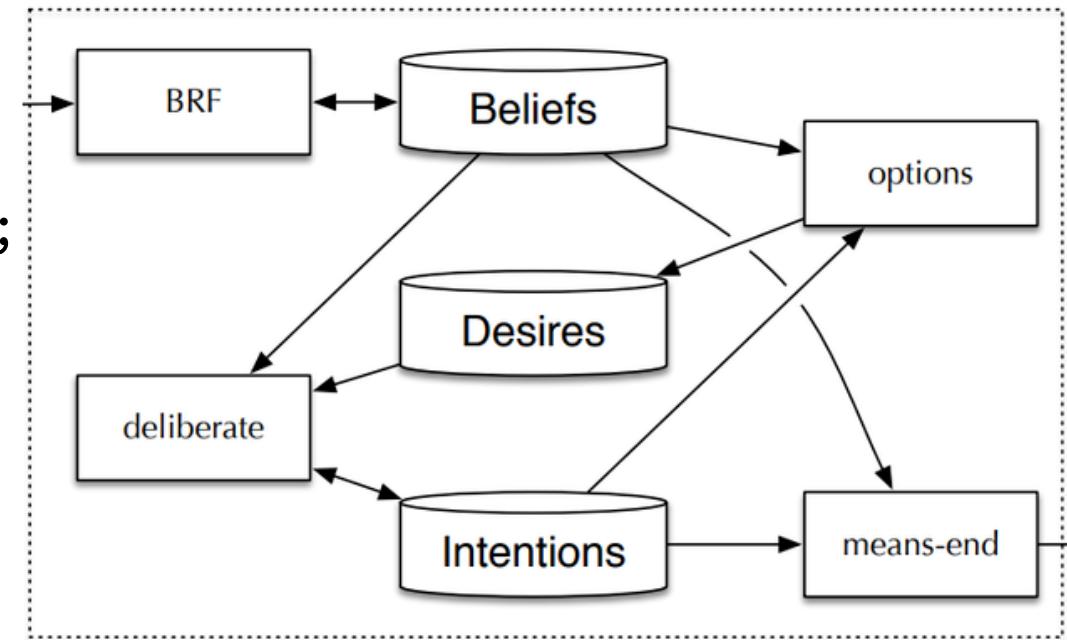
MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications.** [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.** Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

Existem diversos modelos cognitivos:

- Adaptive Control of Thought - Rational;
- SOAR/Inteligências Multifocal;
- Belief-Desire-Intention (BDI);
- Practical Reasoning System (PRS);



MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. I.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Modelos Cognitivos

Existem diversos modelos cognitivos:

- Adaptive Control of Thought - Rational;
- SOAR/Inteligências Multifocal;
- Belief-Desire-Intention (BDI);
- Practical Reasoning System (PRS);
- Many others.

MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Arquitetura BDI

Uma das principais arquiteturas para o desenvolvimento de agentes cognitivos é o modelo **belief-desire-intention (BDI)**.

MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Arquitetura BDI

Uma das principais arquiteturas para o desenvolvimento de agentes cognitivos é o modelo **belief-desire-intention** (BDI).

Este modelo está fundamentado no entendimento do raciocínio prático humano que decide, momento a momento, qual ação realizar para alcançar nossos objetivos.

MICHEL, Fabien; FERBER, Jacques; DROGOUL, Alexis. Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective. **Multi-Agent systems: Simulation and applications**. [S. l.]: CRC Press, 2009.

WOOLDRIDGE, Michael. Intelligent Agents. **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1999. p. 27–77.

Arquitetura BDI

O BDI possibilita a implementação de atitudes mentais em agentes cognitivos:

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

O BDI possibilita a implementação de atitudes mentais em agentes cognitivos:

- Crença é uma informação, ou seja, aquilo que o agente sabe sobre o ambiente, si mesmo ou outros agentes.

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

Arquitetura BDI

O BDI possibilita a implementação de atitudes mentais em agentes cognitivos:

- Crença é uma informação, ou seja, aquilo que o agente sabe sobre o ambiente, si mesmo ou outros agentes.
- Desejo é um propósito que o agente busca tornar realidade.

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

O BDI possibilita a implementação de atitudes mentais em agentes cognitivos:

- Crença é uma informação, ou seja, aquilo que o agente sabe sobre o ambiente, si mesmo ou outros agentes.
- Desejo é um propósito que o agente busca tornar realidade.
- Intenção é uma ação que o agente faz para alcançar um desejo.

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

AgentSpeak(L)

A linguagem de programação **AgentSpeak(L)** é uma extensão natural e elegante de programação em lógica para a arquitetura de agentes BDI, que representa um modelo abstrato para a programação de agentes cognitivos.

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

AgentSpeak(L)

Um agente AgentSpeak(L) possui minimamente:

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

AgentSpeak(L)

Um agente AgentSpeak(L) possui minimamente:

- Uma base de crenças;

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

AgentSpeak(L)

Um agente AgentSpeak(L) possui minimamente:

- Uma base de crenças;
- Uma lista de objetivos;

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

AgentSpeak(L)

Um agente AgentSpeak(L) possui minimamente:

- Uma base de crenças;
- Uma lista de objetivos;
- Uma biblioteca de planos.

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

BELIEFS AND RULES



Jason Framework



HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

Jason Framework



Jason | a Java-based interpreter for an extended version of AgentSpeak.

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

Jason Framework

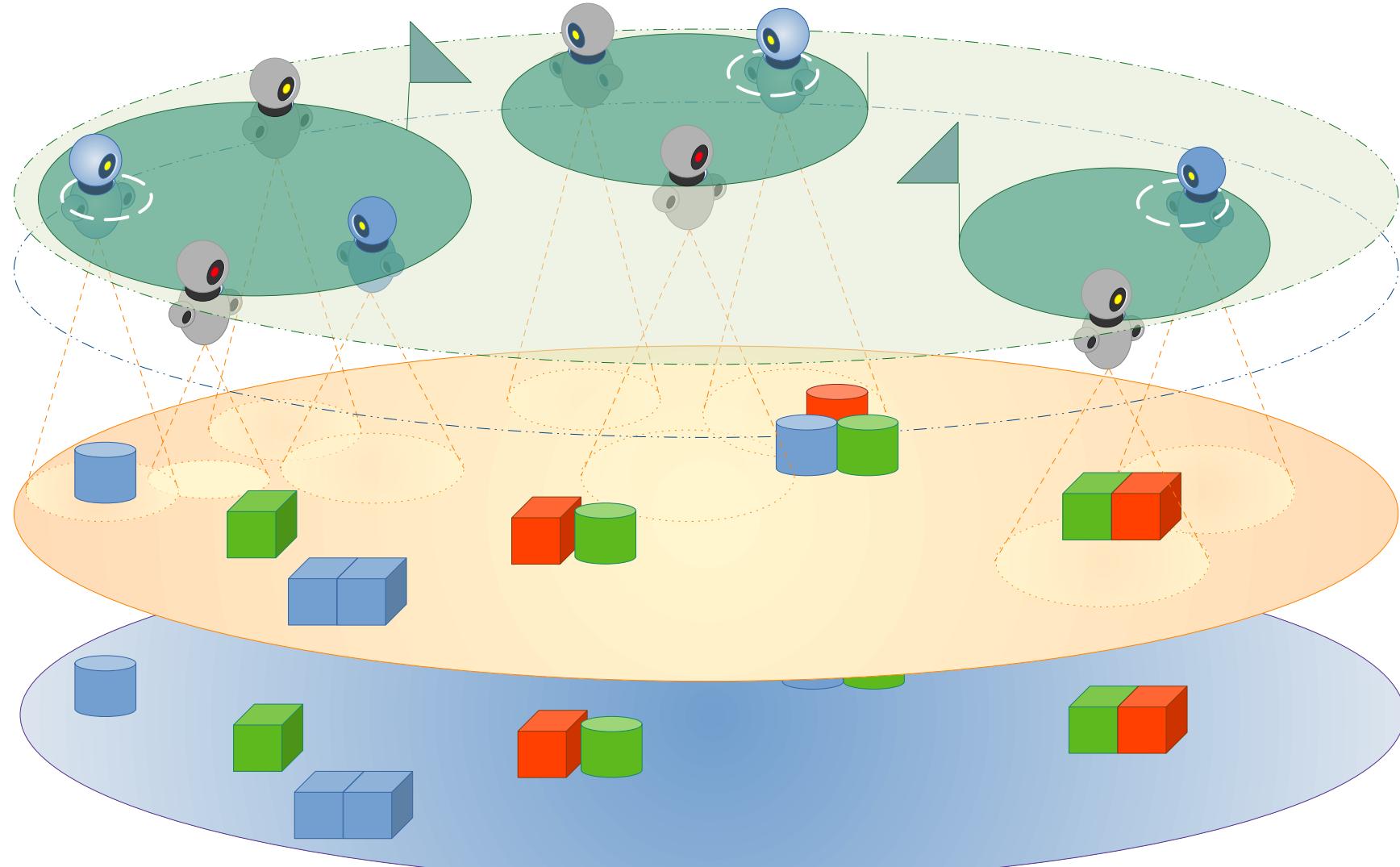


Jason | a Java-based interpreter for an extended version of AgentSpeak.

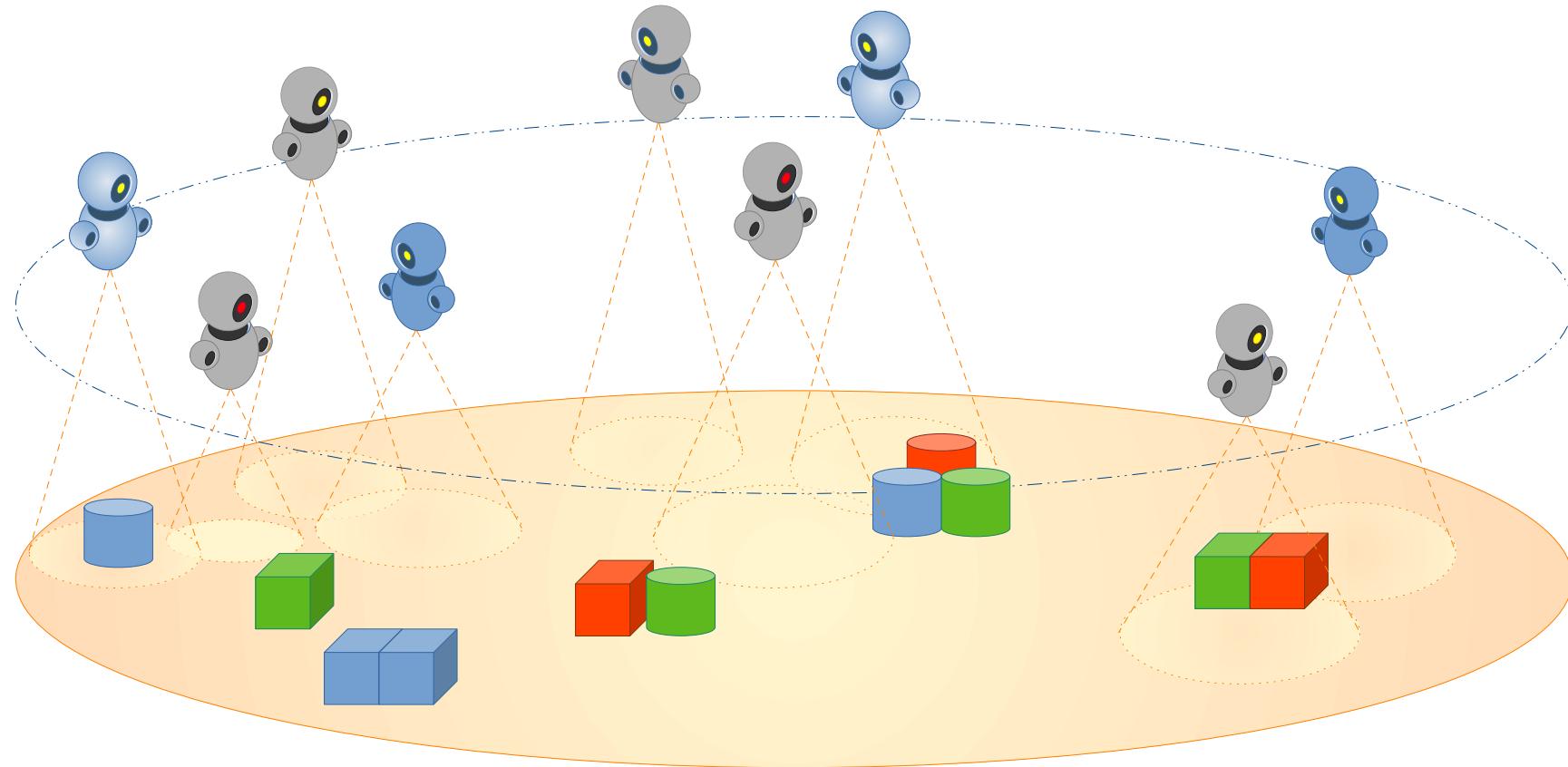
<http://jason.sf.net>

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

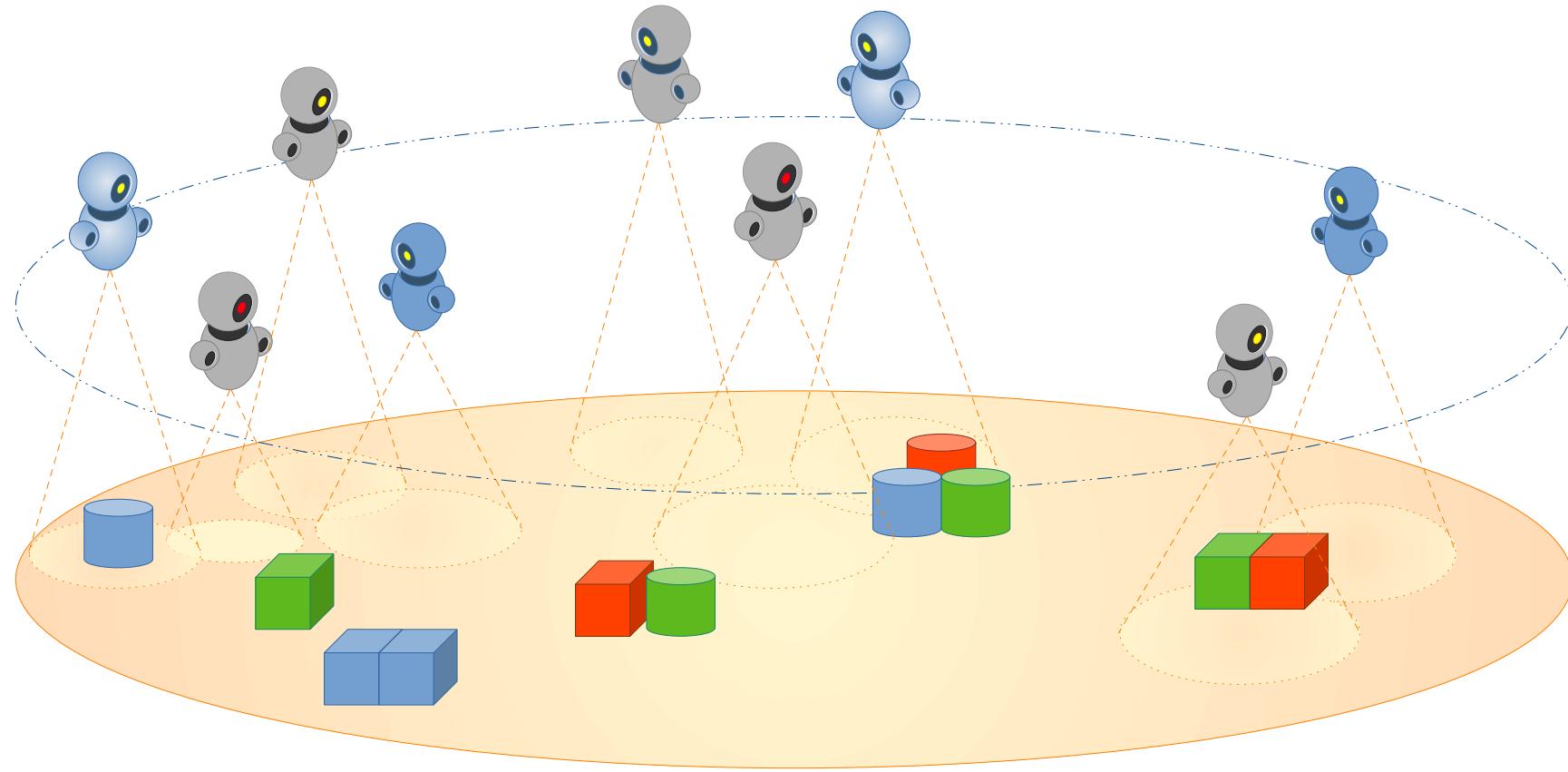
Jason Framework



Jason Framework



Jason Framework



Using Jason, it is possible to program the **agent** and the **endogenous environment** dimensions.

Jason Framework: Crenças

Em Jason, um agente armazena as **informações** percebidas do ambiente; as informações internas; e informações de comunicação através de crenças.

Jason Framework: Crenças

Em Jason, um agente armazena as **informações** percebidas do ambiente; as informações internas; e informações de comunicação através de crenças.

As **crenças** são armazenadas em uma **Base de Crenças** (Belief Base).

Jason Framework: Crenças

Em Jason, um agente armazena as **informações** percebidas do ambiente; as informações internas; e informações de comunicação através de crenças.

As **crenças** são armazenadas em uma **Base de Crenças** (Belief Base).

As crenças são representadas como predicados da **lógica tradicional**. Os **predicados** representam propriedades particulares.

Jason Framework: Beliefs

1. Mental Notes (self)

- Informações adicionadas na base de crenças pelo próprio agente.

Jason Framework: Beliefs Types

1. Mental Notes (self)

- Informações adicionadas na base de crenças pelo próprio agente.

2. Communication (agent)

- Informações obtidas pelo agente através da interação com outros agentes.

Jason Framework: Beliefs Types

1. Mental Notes (self)

- Informações adicionadas na base de crenças pelo próprio agente.

2. Communication (agent)

- Informações obtidas pelo agente através da interação com outros agentes.

3. Environmental Perceptions (percepts)

- Informações coletadas pelo agente que são relativas ao sensoriamento constante do ambiente.

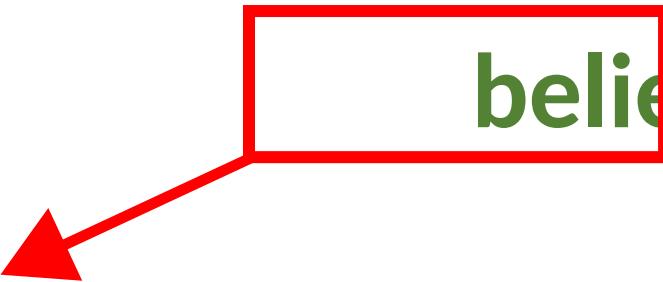
belief [source(type)]

Beliefs: Format

belief [source(type)]

Beliefs: Format

belief [source(type)]



a predicate from
logic.

Beliefs: Format

belief [source(type)]

Beliefs: Format

belief [source(type)]



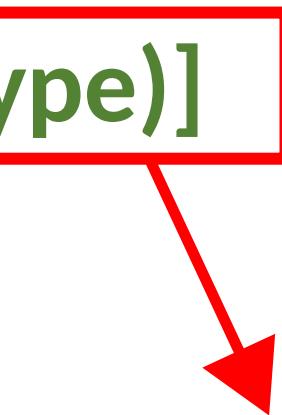
the source of the
belief.

Beliefs: Format

belief [source(type)]

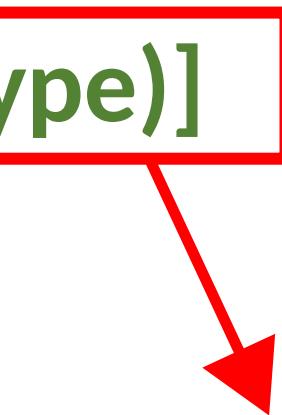
Beliefs: Format

belief [source(type)]



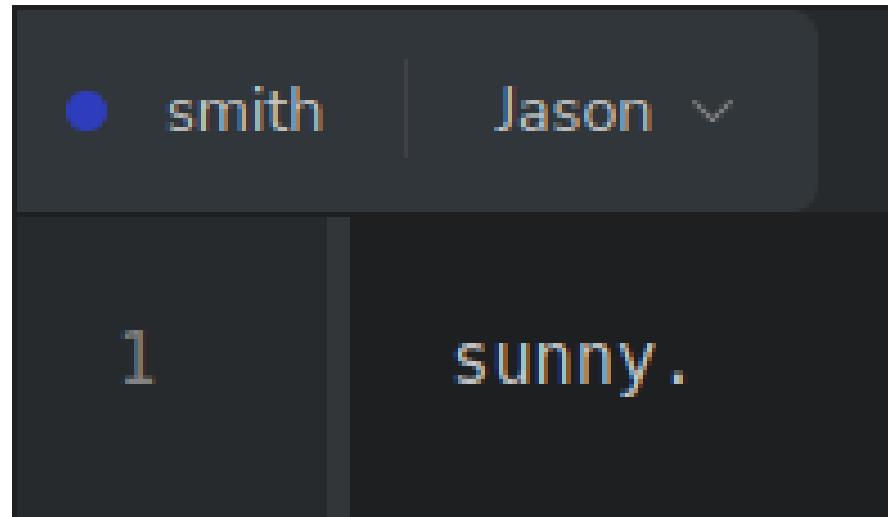
The belief's source
type .

belief [source(type)]

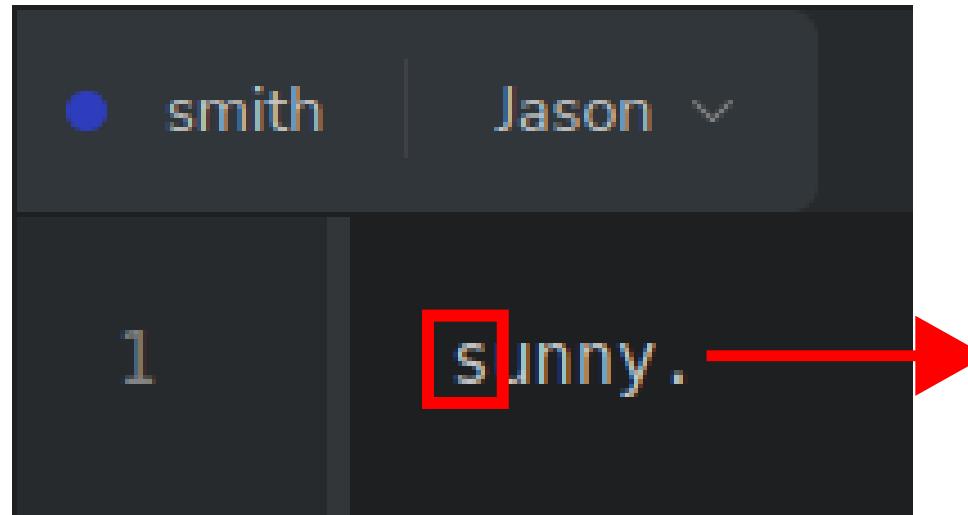


It can be self, percepts, or
from other agents.

Initial Beliefs: Ment

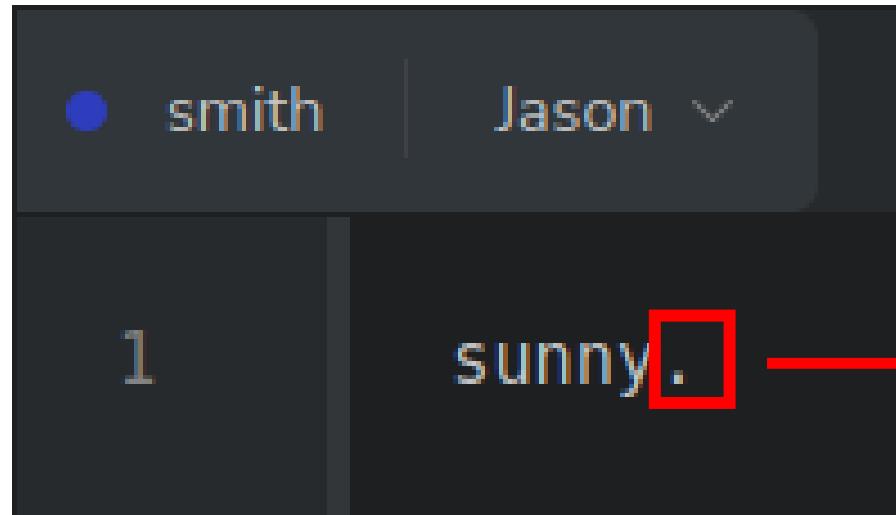


Initial Beliefs: Mental Notes



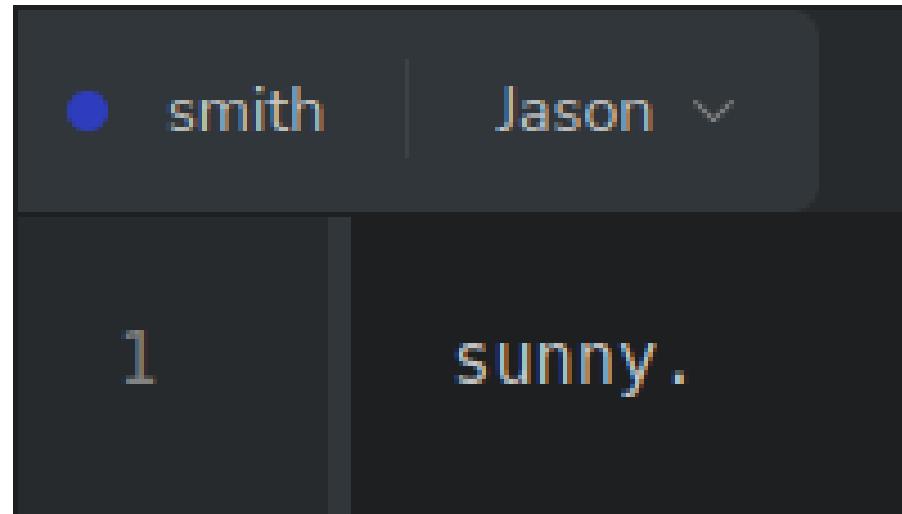
every belief must
start with a
lowercase letter.

Initial Beliefs: Mental Notes

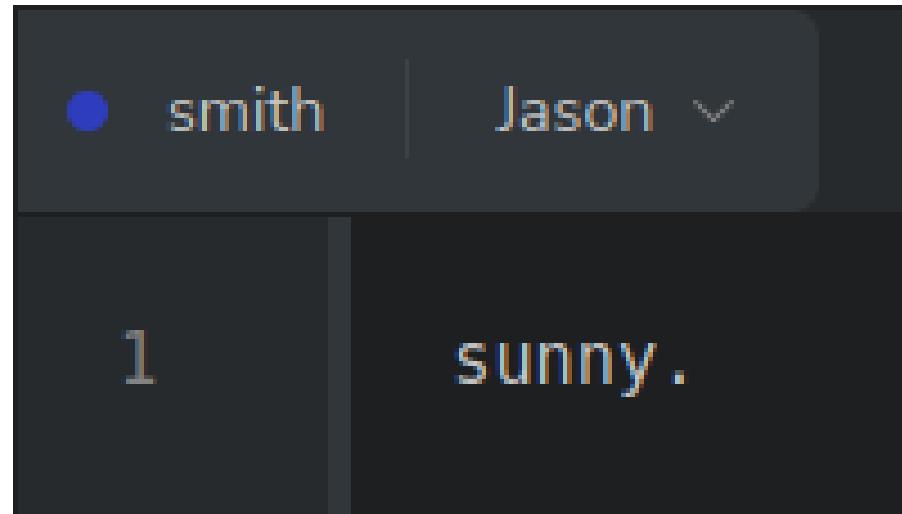


and it must end with
a period.

Initial Beliefs: Mental Notes



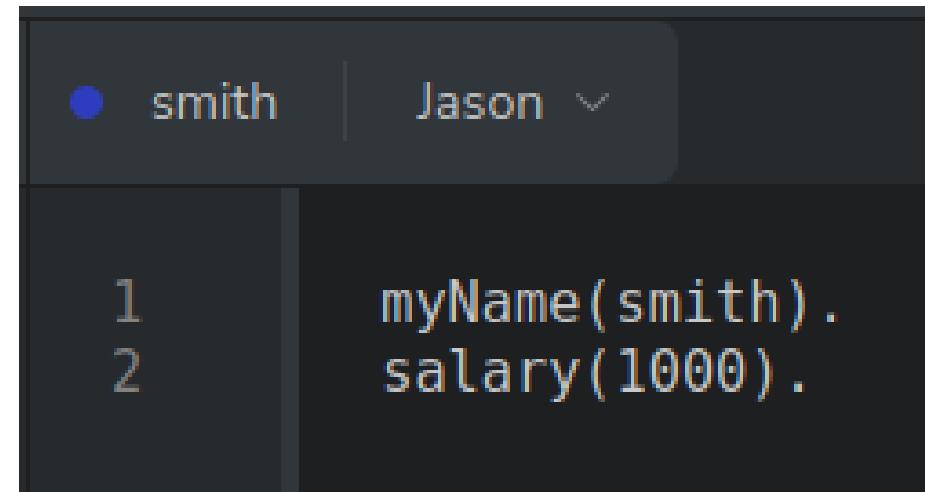
Initial Beliefs: Mental Notes



Inspection of agent **smith**

- **Beliefs** sunny[source(self)]
- **Annotations**

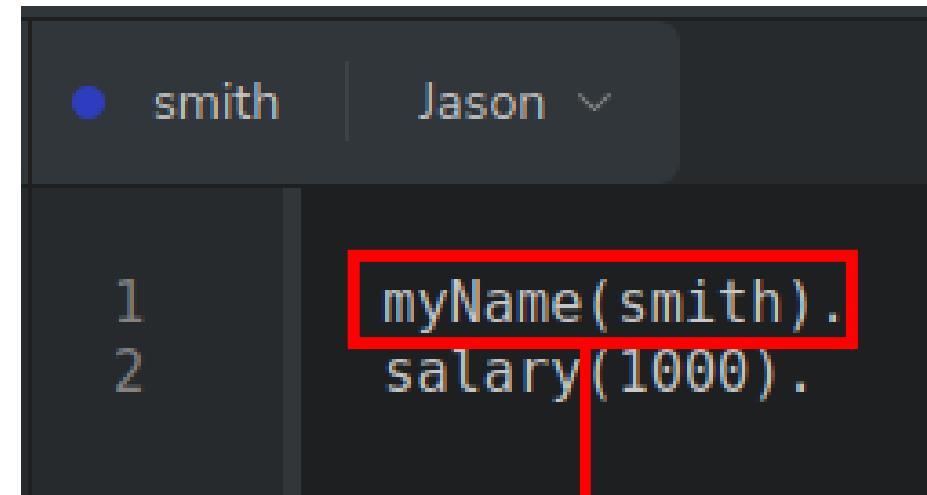
Initial Beliefs: Mental Notes



The screenshot shows a terminal window with a dark background. At the top, there is a header bar with a blue circular icon, the name "smith" in blue, and "Jason" in grey with a dropdown arrow. Below the header, the terminal window displays two numbered lines of code:

```
1 myName(smith).  
2 salary(1000).
```

Initial Beliefs: Mental Notes

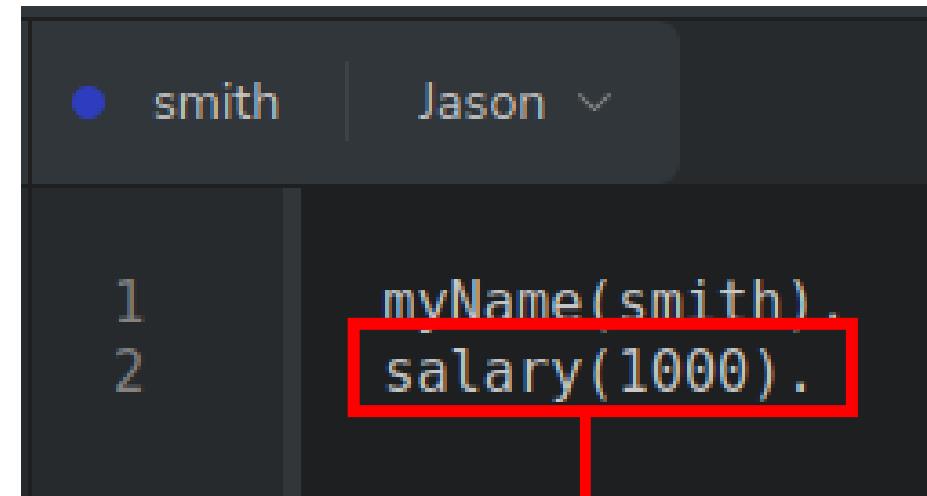


```
● smith | Jason ~
1 myName(smith).
2 salary(1000).
```



predicate(predicate)

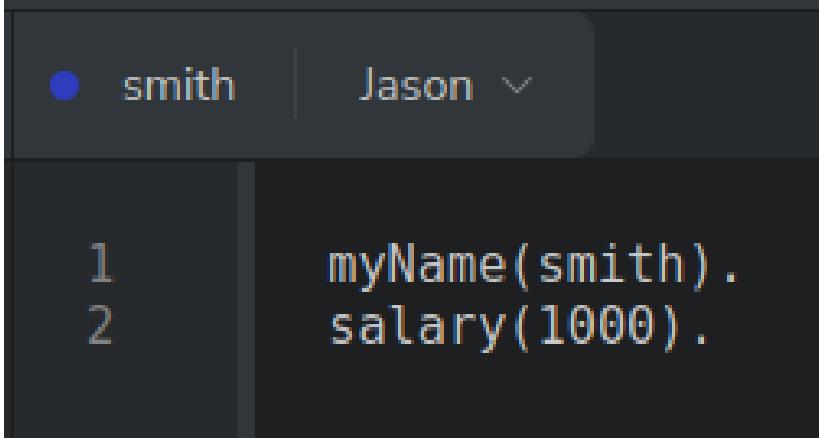
Initial Beliefs: Mental Notes



```
● smith | Jason ~
1 myName(smith).
2 salary(1000).
```

predicate(int)

Initial Beliefs: Mental Notes



The screenshot shows a terminal window with a dark background. At the top, there is a header bar with a blue dot icon, the name "smith" in blue, and a dropdown arrow. Below the header, the text "Jason" is partially visible. The main area of the terminal contains two numbered lines of code:

```
1 myName(smith).  
2 salary(1000).
```

Initial Beliefs: Mental Notes

```
● smith | Jason ▾  
1 myName(smith).  
2 salary(1000).
```



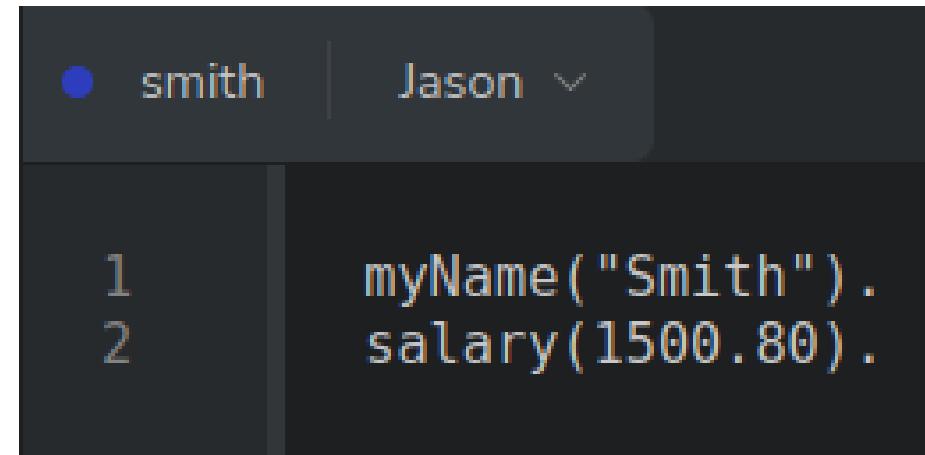
Inspection of agent **smith**

- Beliefs

myName(smith)[source(self)].
salary(1000)[source(self)].

- Annotations

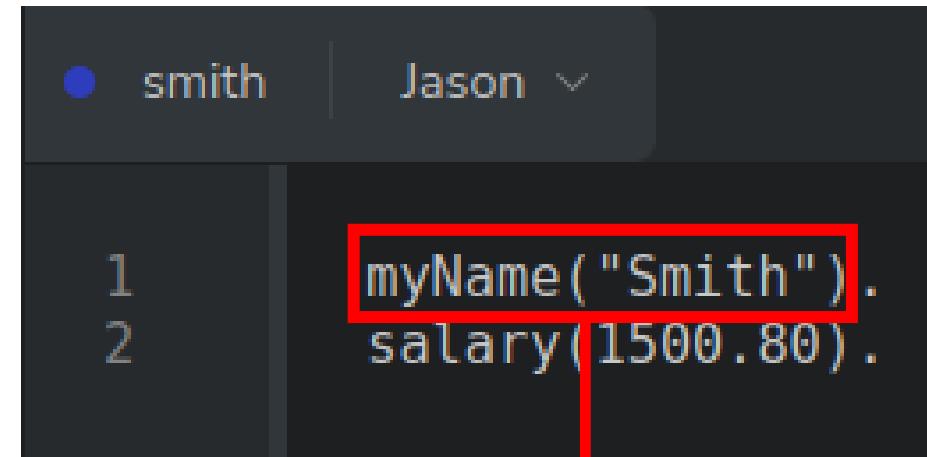
Initial Beliefs: Mental Notes



The screenshot shows a terminal window with a dark background. At the top, there is a header bar with a blue circular icon, the name "smith" in white, and a dropdown menu showing "Jason". Below the header, the terminal output is displayed in two columns. The left column contains two numbers: "1" and "2". The right column contains two lines of code: "myName("Smith")." and "salary(1500.80).".

```
● smith | Jason
1   myName("Smith").
2   salary(1500.80).
```

Initial Beliefs: Mental Notes

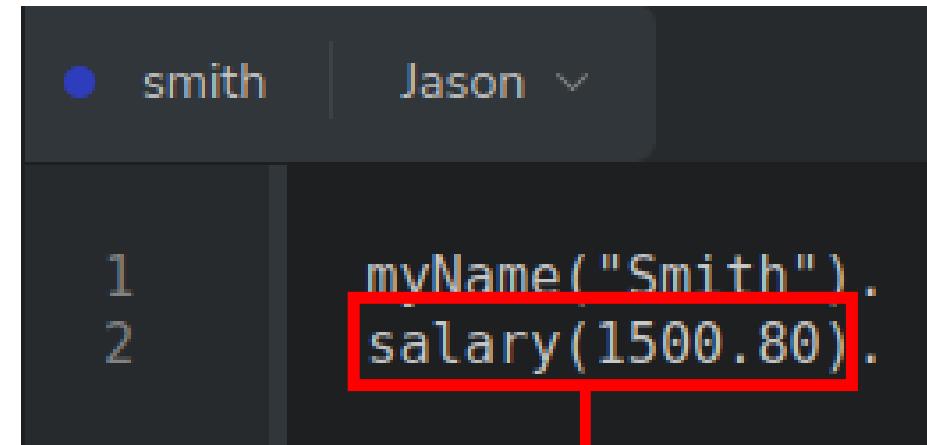


```
● smith | Jason
1 myName("Smith").
2 salary(1500.80).
```



predicate(String)

Initial Beliefs: Mental Notes

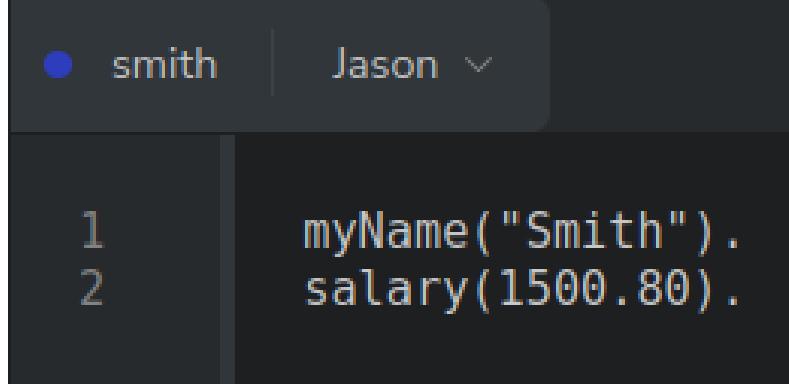


```
● smith | Jason
1 myName("Smith").
2 salary(1500.80).
```



predicate(float)

Initial Beliefs: Mental Notes



The screenshot shows a user interface for managing agent beliefs. At the top, there's a header with a blue dot icon, the name "smith", and a dropdown menu showing "Jason". Below this, a table lists two beliefs:

	Belief Content
1	myName("Smith").
2	salary(1500.80).

Initial Beliefs: Mental Notes

```
● smith      Jason ▾  
1   myName("Smith").  
2   salary(1500.80).
```



Inspection of agent smith

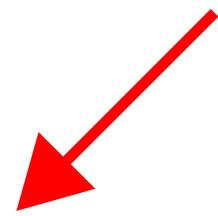
- Beliefs

myName("Smith")_[source(self)].
salary(1500.8)_[source(self)].

- Annotations

predicate(value)

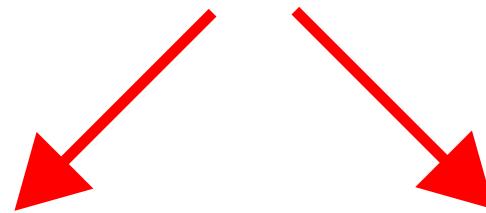
predicate(value)



predicate

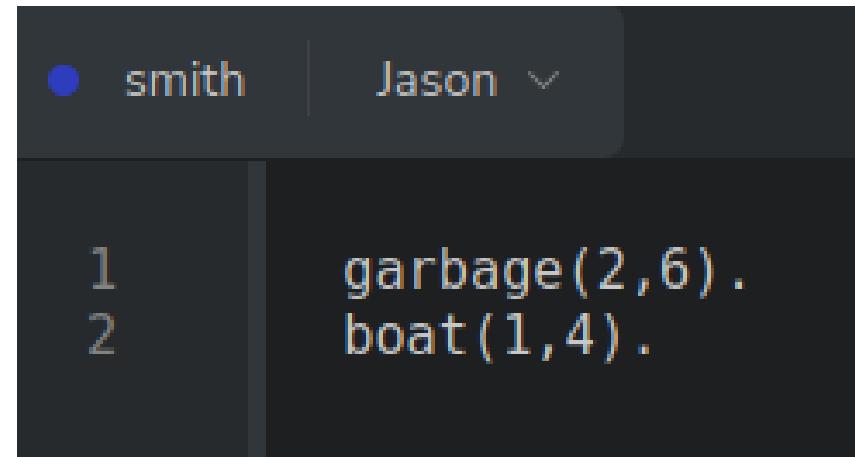
predicate(value)

predicate

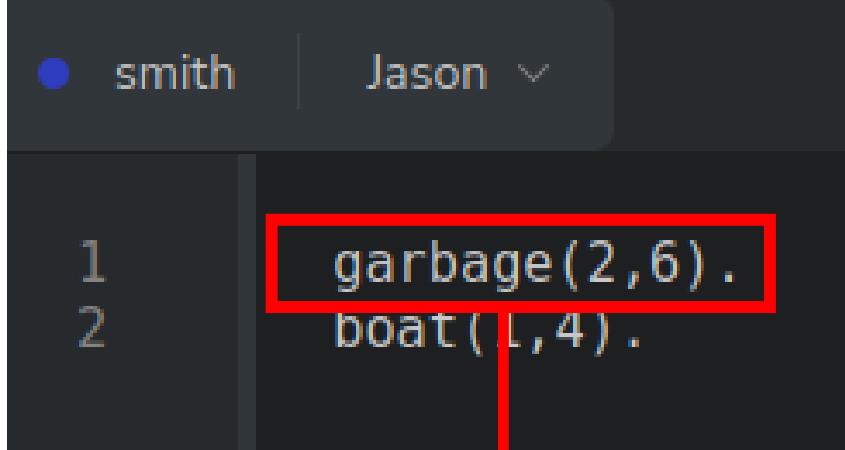


int, float, String, etc.

Initial Beliefs: Mental Notes



Initial Beliefs: Mental Notes

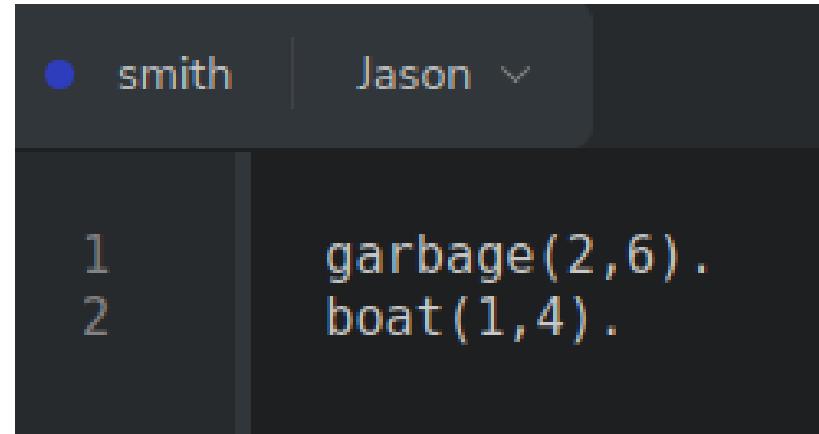


A screenshot of a mobile application interface. At the top, there is a header with a blue dot icon and the text "smith" followed by a vertical ellipsis and "Jason". Below the header, there are two numbered items: "1" and "2". Item "1" contains the text "garbage(2,6).", which is highlighted with a red rectangular box. Item "2" contains the text "boat(1,4).".



predicate(int, int)

Initial Beliefs: Mental Notes



A screenshot of a terminal window titled "smith" with the user "Jason". The terminal displays two numbered facts:

```
1 garbage(2,6).  
2 boat(1,4).
```

Initial Beliefs: Mental Notes

```
● smith    Jason ▾  
1      garbage(2,6).  
2      boat(1,4).
```



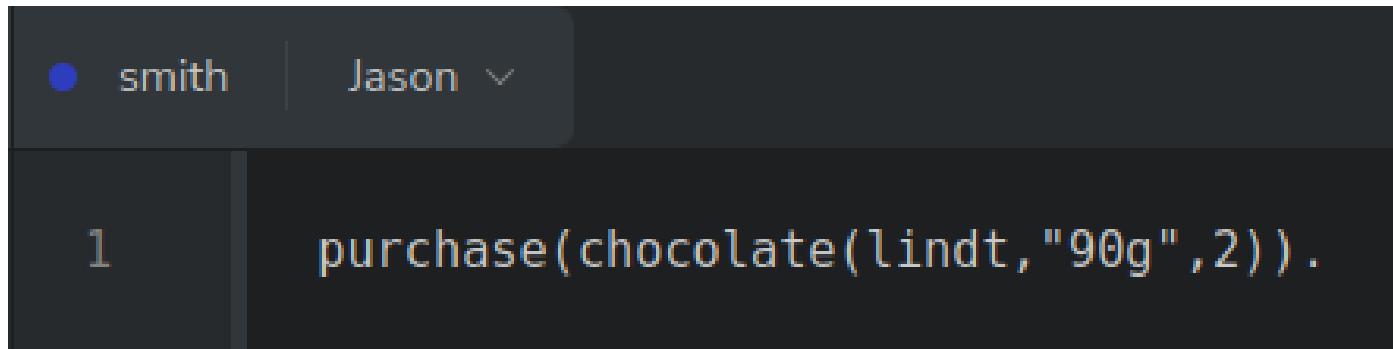
Inspection of agent smith

- Beliefs

boat(1,4)[source(self)].
garbage(2,6)[source(self)].

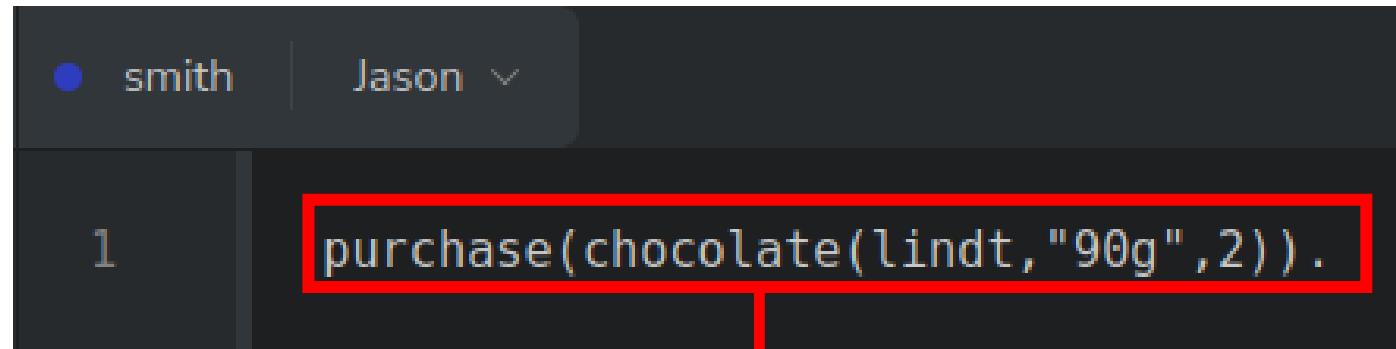
- Annotations

Initial Beliefs: Mental Notes

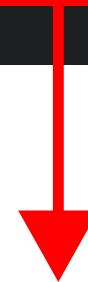


A screenshot of a terminal window with a dark background. At the top, there is a header bar with a blue dot icon, the name "smith", and a dropdown menu labeled "Jason". Below the header, the terminal prompt "1" is visible on the left, followed by the Prolog code "purchase(chocolate(lindt,"90g",2)).".

Initial Beliefs: Mental Notes

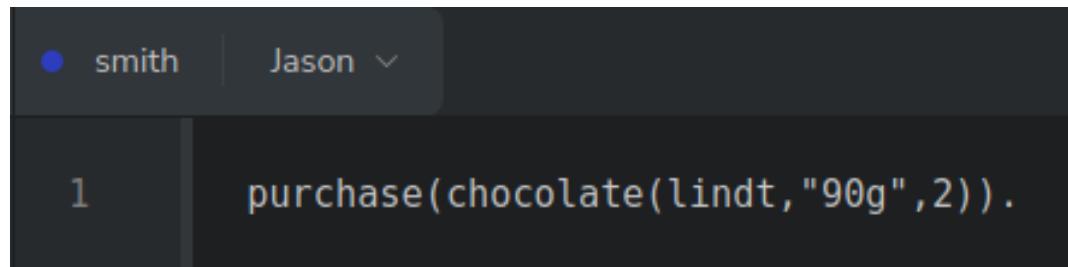


A screenshot of a mobile application interface. At the top, there is a dark header with a blue circular icon and the text "smith". To the right of the icon is the name "Jason" followed by a dropdown arrow. Below the header, the screen has a dark background. On the left side, there is a vertical light gray bar with the number "1" at its top. To the right of this bar, the text "purchase(chocolate(lindt, "90g"), 2)." is displayed in a white font. This text is enclosed in a red rectangular box. A red arrow points downwards from the bottom of this red box towards the explanatory text below.



predicate(predicate(predicate, String, int))

Initial Beliefs: Mental Notes



```
smith | Jason ▾
1 purchase(chocolate(lindt,"90g",2)).
```

Initial Beliefs: Mental Notes

```
● smith | Jason ▾  
1 purchase(chocolate(lindt,"90g",2)).
```



Inspection of agent **smith**

- Beliefs

```
purchase(chocolate(lindt,"90g",2))[source(self)].
```

- Annotations

predicate(predicate)

Beliefs: Format

predicate(predicate)

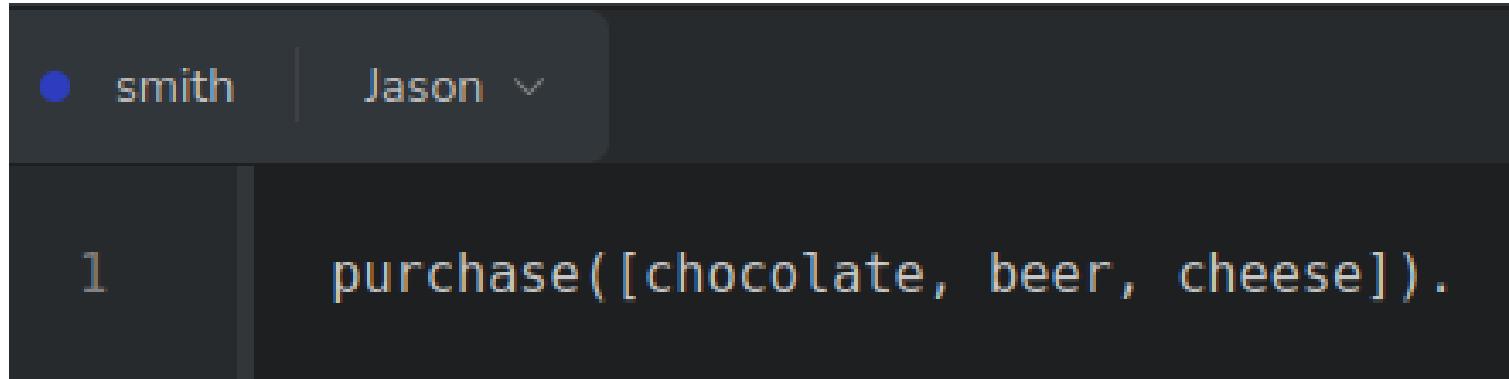
predicate(predicate(predicate))

predicate(predicate(predicate))

Beliefs: Format

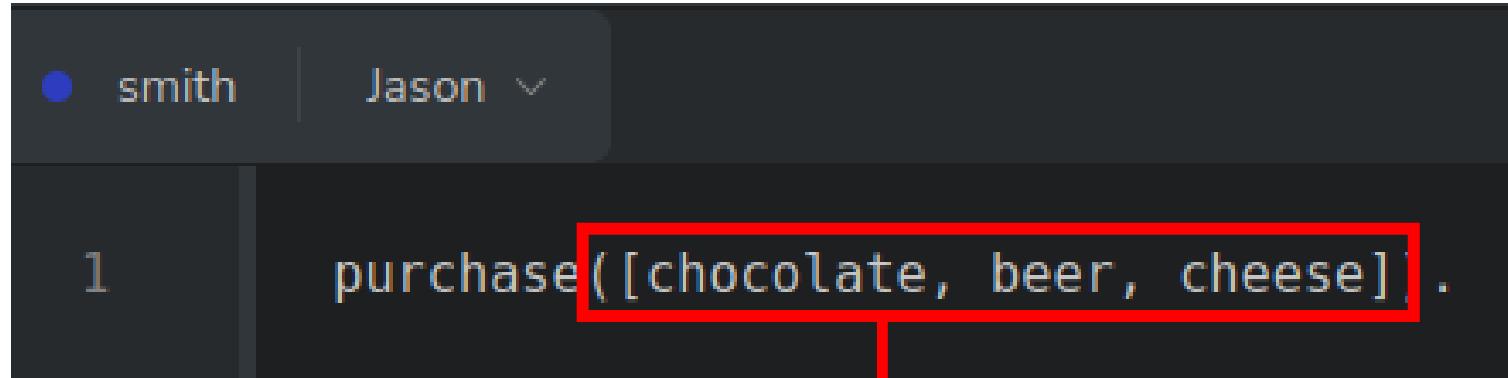
predicate(predicate(predicate(...))))

Initial Beliefs: Mental Notes

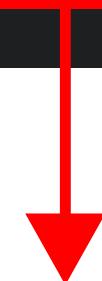


The screenshot shows a terminal window with a dark background. At the top left, there is a user interface element with a blue dot icon, the name "smith", and the name "Jason" followed by a dropdown arrow. Below this, the terminal prompt "1" is visible on the left. To its right, the command "purchase([chocolate, beer, cheese])." is displayed in white text.

Initial Beliefs: Mental Notes

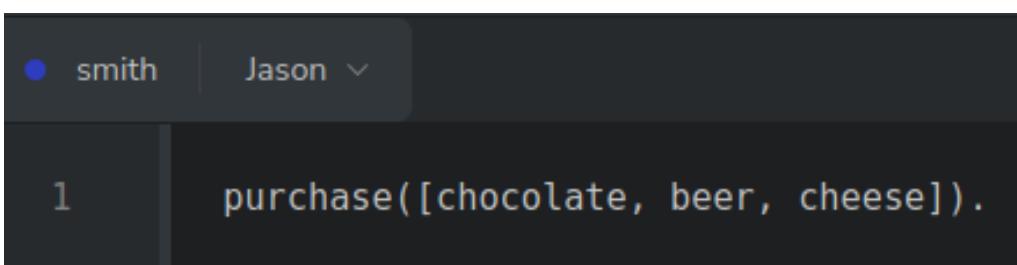


```
● smith | Jason ▾  
1 purchase([chocolate, beer, cheese]).
```



predicate(predicate, predicate, predicate)

Initial Beliefs: Mental Notes



The screenshot shows a user interface for the Jason agent programming language. At the top, there's a header bar with a blue dot icon, the name "smith", and a dropdown menu labeled "Jason". Below this is a dark grey panel containing a number "1" and the text "purchase([chocolate, beer, cheese]).". This represents a single belief or fact stored in the agent's memory.

Initial Beliefs: Mental Notes

```
● smith | Jason ▾  
1 purchase([chocolate, beer, cheese]).
```



Inspection of agent **smith**

- Beliefs

```
purchase([chocolate,beer,cheese])[source(self)].
```

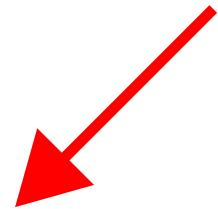
- Annotations

predicate([list])

predicate([list])

[value, value, ...]

[value, value, ...]



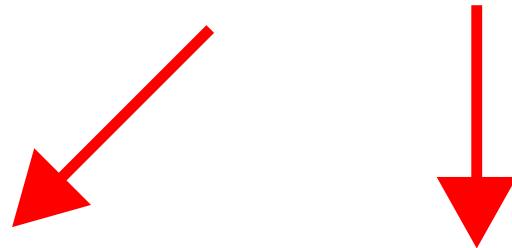
predicate

Beliefs: Format

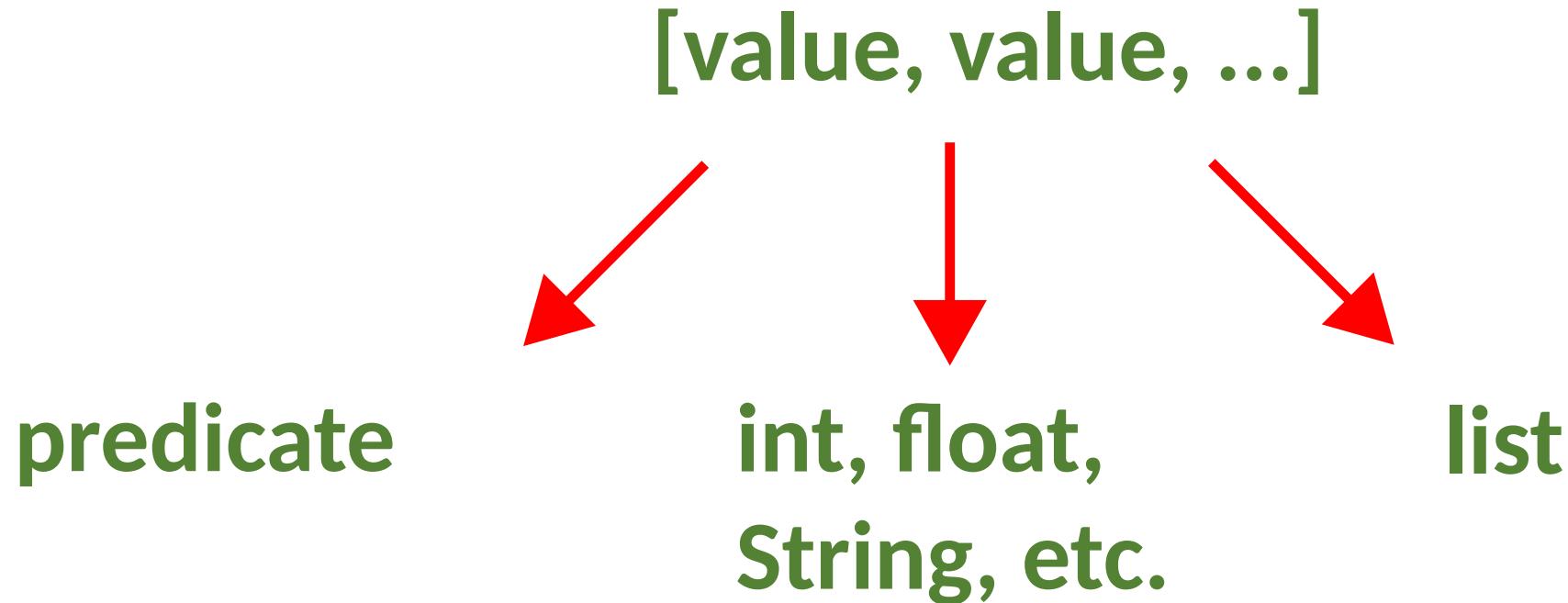
[value, value, ...]

predicate

int, float,
String, etc.



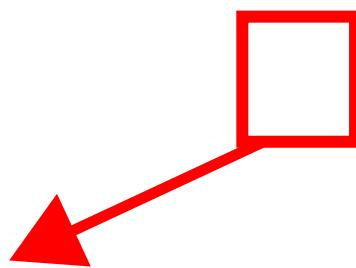
Beliefs: Format



Beliefs: Strong Negation

$\sim\text{belief} \text{ [source(type)]}$

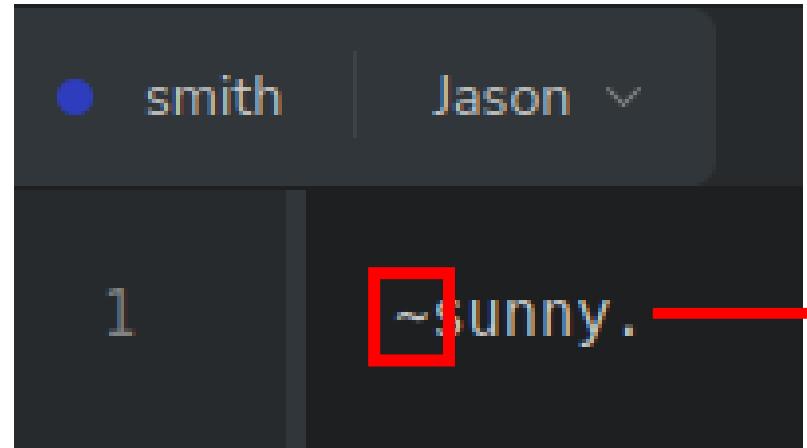
Beliefs: Strong Negation



$\text{~belief [source(type)]}$

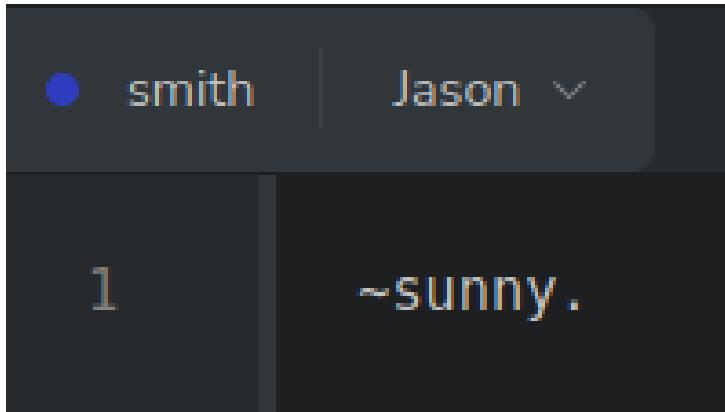
every strong negation
starts with ~.

Initial Beliefs

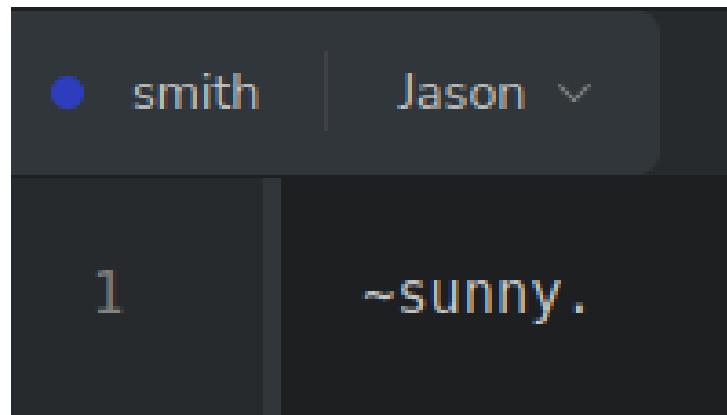


it negates the predicate.

Initial Beliefs: Mental Notes



Initial Beliefs: Mental Notes



Inspection of agent **smith**

- Beliefs

~sunny[source(self)].

- Annotations

Jason Framework: Rules

A **rule** is a statement that defines relationships between facts (beliefs).

Jason Framework: Rules

A **rule** is a statement that defines relationships between facts (beliefs).

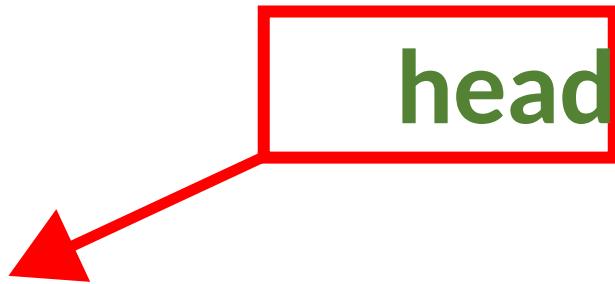
The rule operator can be understood as a **logical implication**. It reads as "**if**" or "**is true when**".

Rules: Format

head :- body.

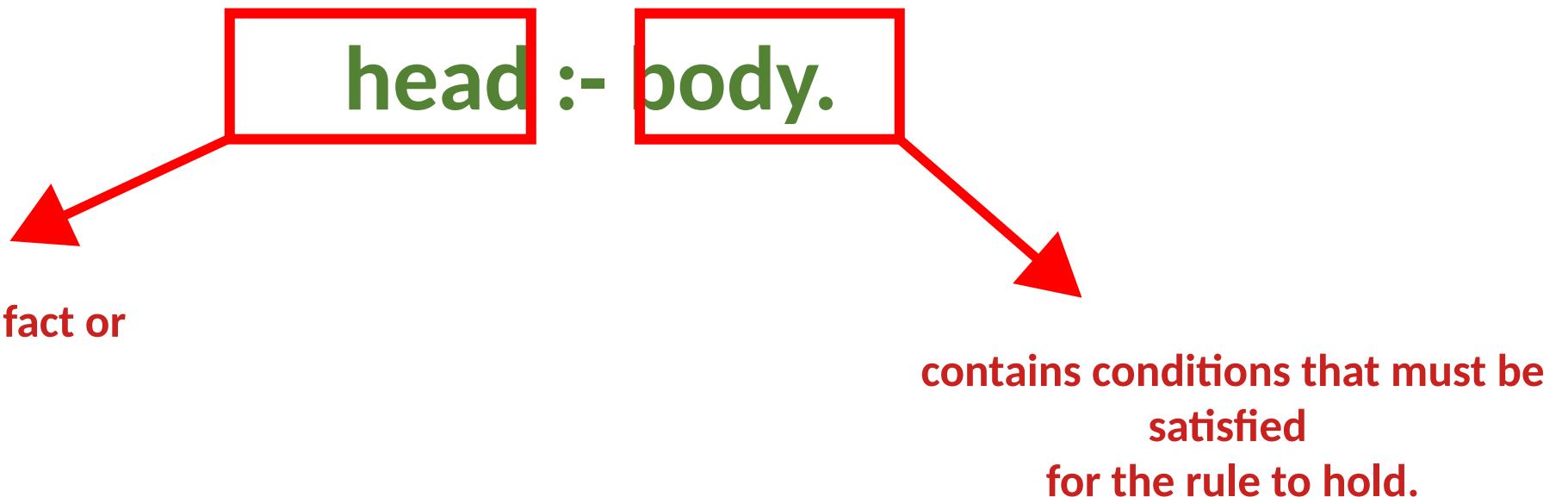
Rules: Format

head :- body.

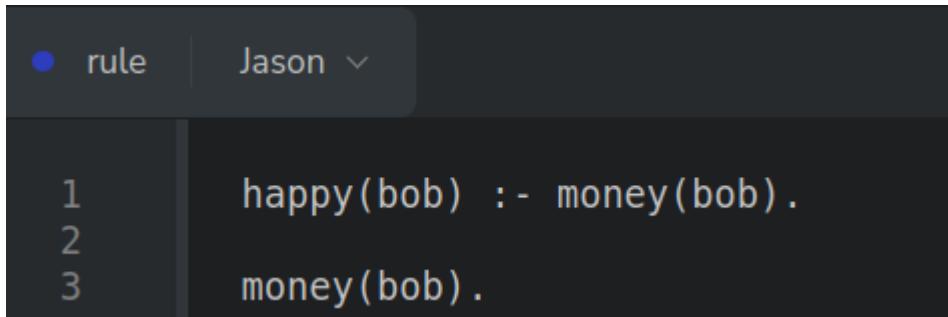


A consequent fact or
belief.

Rules: Format



Rules: Predicate

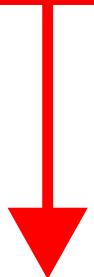


The screenshot shows a user interface for editing Prolog-like rules. At the top left is a blue circular icon with a white dot, followed by the word "rule". To its right is a dropdown menu set to "Jason". Below this is a list of numbered steps:

- 1 happy(bob) :- money(bob).
- 2
- 3 money(bob).

Rules: Predicate

```
● rule | Jason ▾  
1 happy(bob) :- money(bob).  
2  
3 money(bob).
```



When the agent believes...

Rules: Predicate

```
rule Jason
1 happy(bob) :- money(bob).
2
3 money(bob).
```

... this condition is satisfied.

Rules: Predicate

```
● rule | Jason ▾  
1 happy(bob) :- money(bob).  
2  
3 money(bob).
```



So, it holds ...

Rules: Predicate

```
● rule | Jason ▾  
1 happy(bob) :- money(bob).  
2  
3 money(bob).
```



... but it is NOT A BELIEF.

Rules: Predicate

```
● rule | Jason ▾  
1 happy(bob) :- money(bob).  
2  
3 money(bob).
```



Inspection of agent rule

- Beliefs	money(bob)[source(self)].
- Rules	happy(bob) :- money(bob).
- Annotations	

Jason Framework: Variables

In Jason, a variable is a **placeholder** for an **unknown value**.

Jason Framework: Variables

In Jason, a variable is a **placeholder** for an **unknown value**.

- denoted by a string consisting of an uppercase letter or an underscore;

Jason Framework: Variables

In Jason, a variable is a **placeholder** for an **unknown value**.

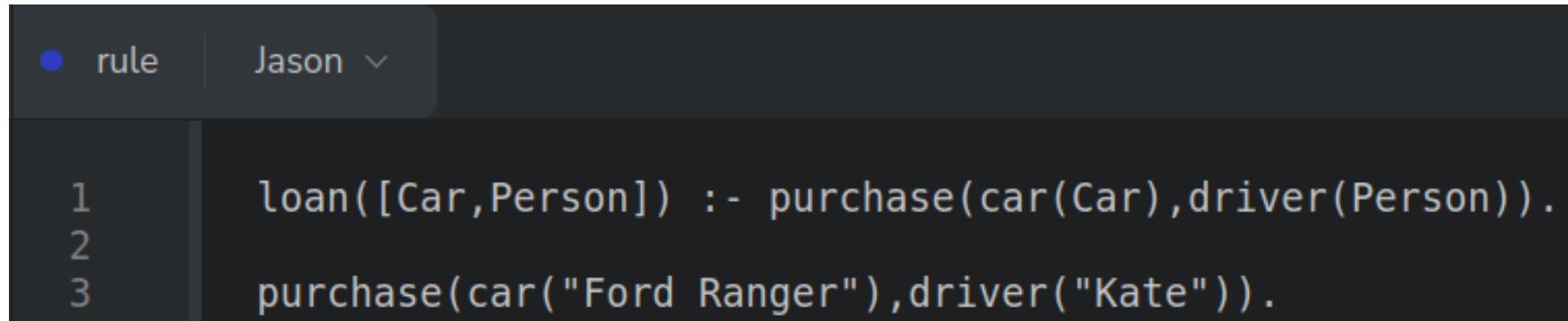
- denoted by a string consisting of an uppercase letter or an underscore;
- it must be instantiated with specific values while executing the agent's reasoning;

Jason Framework: Variables

In Jason, a variable is a **placeholder** for an **unknown value**.

- denoted by a string consisting of an uppercase letter or an underscore;
- it must be instantiated with specific values while executing the agent's reasoning;
- The underscore is a special variable that is often used when the value of a variable is not needed or not used. It is sometimes referred to as a "**don't care**" variable.

Rules: With Variables



```
rule Jason ▾
1 loan([Car,Person]) :- purchase(car(Car),driver(Person)).
2
3 purchase(car("Ford Ranger")), driver("Kate")).
```

Rules: With Variables

```
rule Jason
1 loan([Car,Person]) :- purchase(car(Car),driver(Person)).
2
3 purchase(car("Ford Ranger"),driver("Kate")).
```

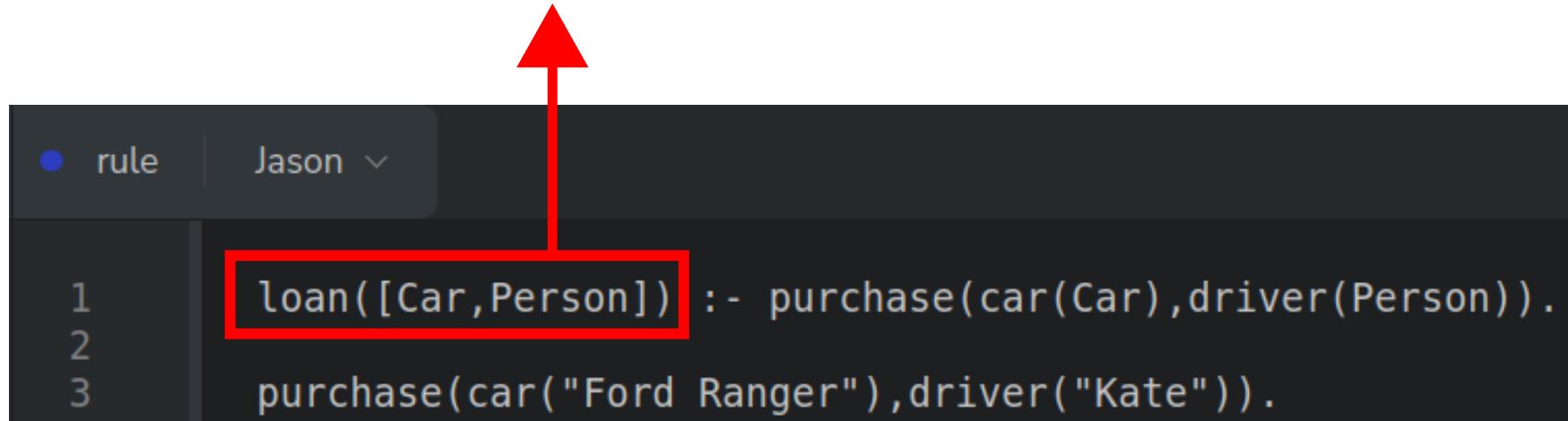
predicate(Variable)



predicate(Variable)

Rules: With Variables

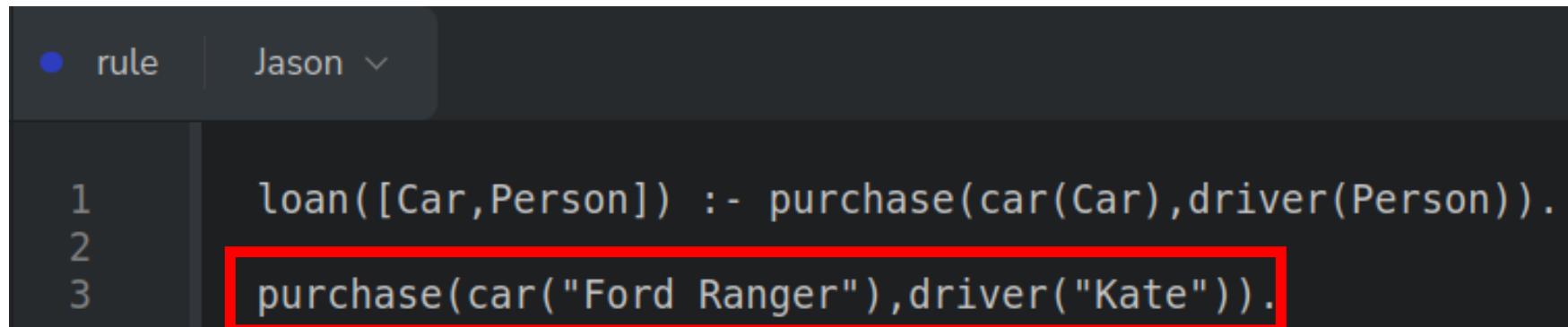
predicate([Variable, Variable])



A screenshot of a Prolog IDE interface. At the top, there is a toolbar with a 'rule' button and a dropdown menu set to 'Jason'. Below the toolbar, the code editor shows three numbered lines of Prolog code. The first line is '1 loan([Car,Person]):- purchase(car(Car),driver(Person)).'. The second line is '2'. The third line is '3 purchase(car("Ford Ranger"),driver("Kate")).' A red rectangular box highlights the term 'loan([Car,Person])' in the first line, and a red arrow points upwards from this box towards the title 'predicate([Variable, Variable])'.

```
1 loan([Car,Person]):- purchase(car(Car),driver(Person)).  
2  
3 purchase(car("Ford Ranger"),driver("Kate")).
```

Rules: With Variables

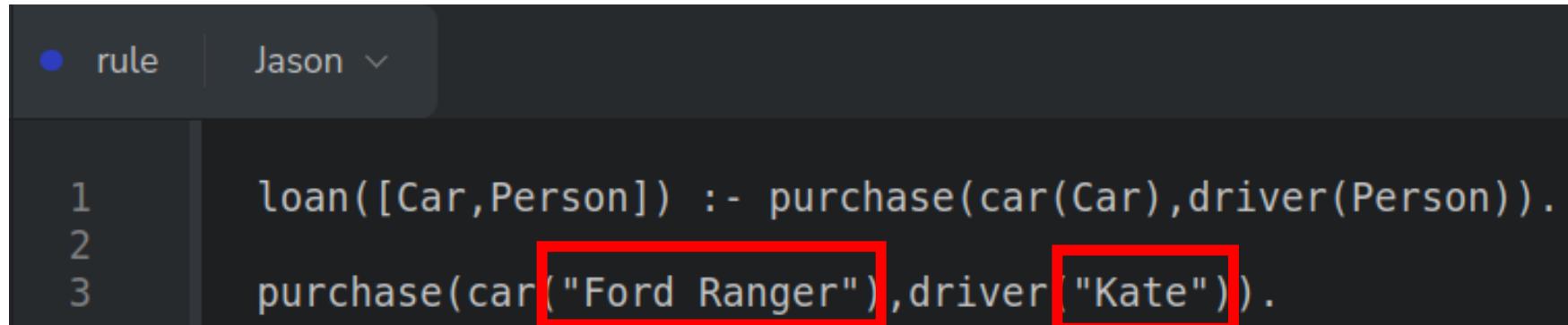


```
rule Jason
1 loan([Car,Person]) :- purchase(car(Car),driver(Person)).
2
3 purchase(car("Ford Ranger"),driver("Kate")).
```



If a belief exists...

Rules: With Variables



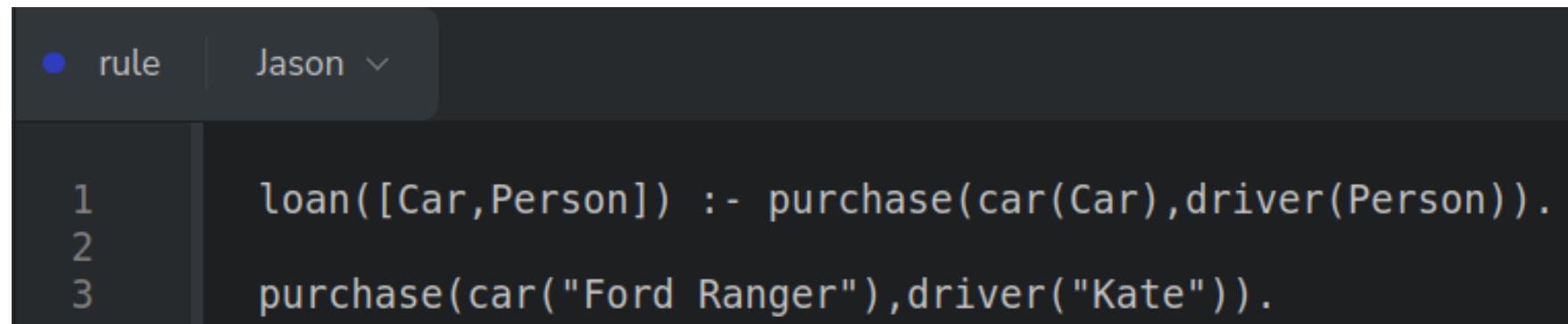
```
rule Jason ▾
1   loan([Car,Person]) :- purchase(car(Car),driver(Person)).
2
3   purchase(car("Ford Ranger"),driver("Kate")).
```

Rules: With Variables

```
rule Jason
1 loan([Car,Person])
2
3 purchase(car([Car]),driver([Person])).  
purchase(car("Ford Ranger"),driver("Kate")).
```

... it binds the value to a variable
(unification).

Rules: With Variables



The screenshot shows the Jason IDE interface. At the top, there is a navigation bar with a blue dot icon labeled "rule" and a dropdown menu labeled "Jason". Below the navigation bar, the code editor displays three numbered rules:

- 1 loan([Car,Person]) :- purchase(car(Car),driver(Person)).
- 2
- 3 purchase(car("Ford Ranger"),driver("Kate")).

Rules: With Variables

```
rule Jason
1 loan([Car,Person]) :- purchase(car(Car),driver(Person)).
2
3 purchase(car("Ford Ranger"),driver("Kate")).
```



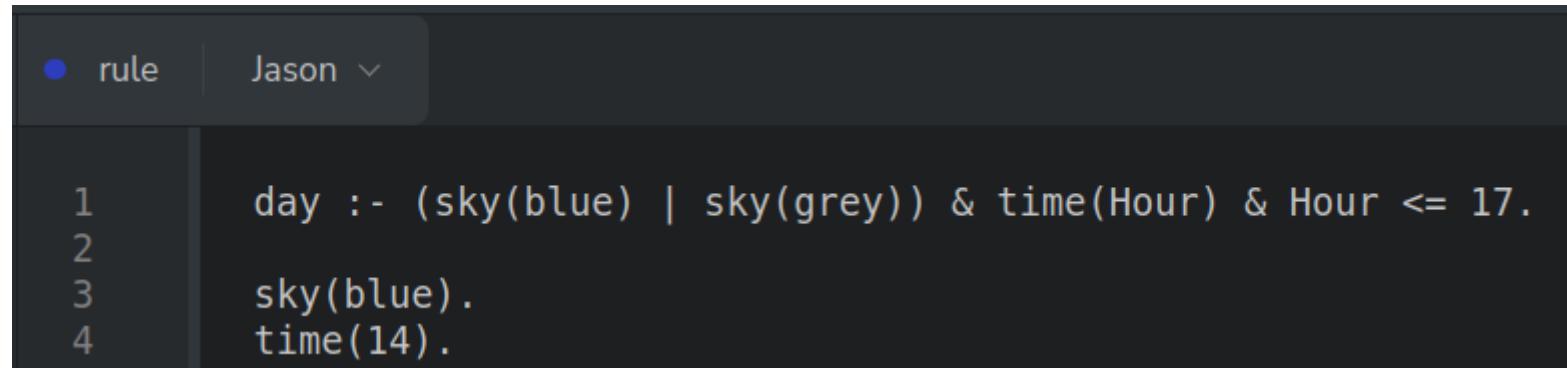
Inspection of agent rule

- Beliefs `purchase(car("Ford Ranger"),driver("Kate"))[source(self)]`

- Rules `loan([Car,Person]) :-
 purchase(car(Car),driver(Person)).`

- Annotations

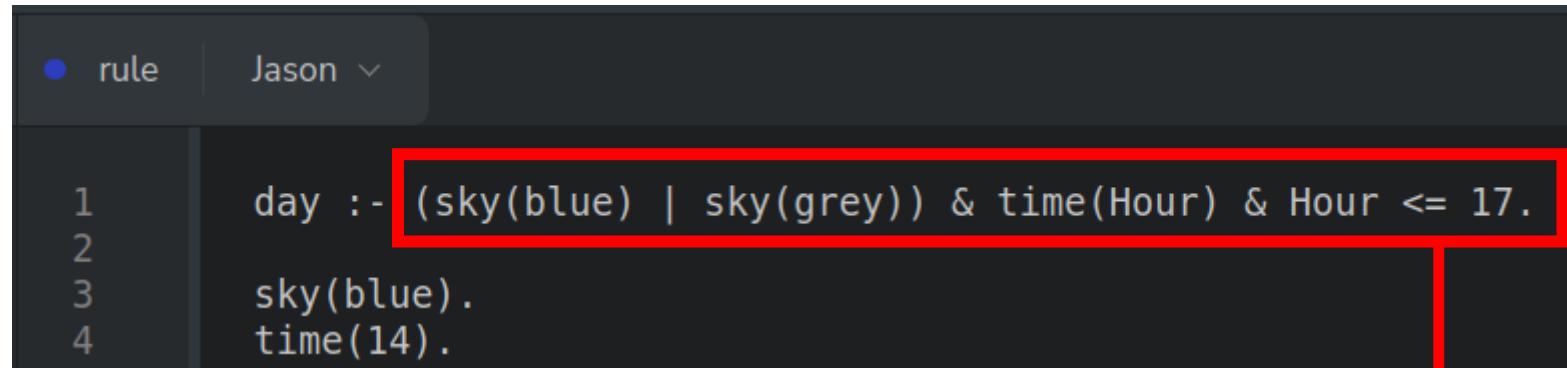
Rules: Predicate and Variable



The screenshot shows a Prolog IDE interface with a dark theme. At the top, there is a navigation bar with a blue dot icon followed by the word "rule" and a dropdown menu set to "Jason". Below the navigation bar, the code is displayed in a text area:

```
1 day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.  
2  
3 sky(blue).  
4 time(14).
```

Rules: Predicate and Variable

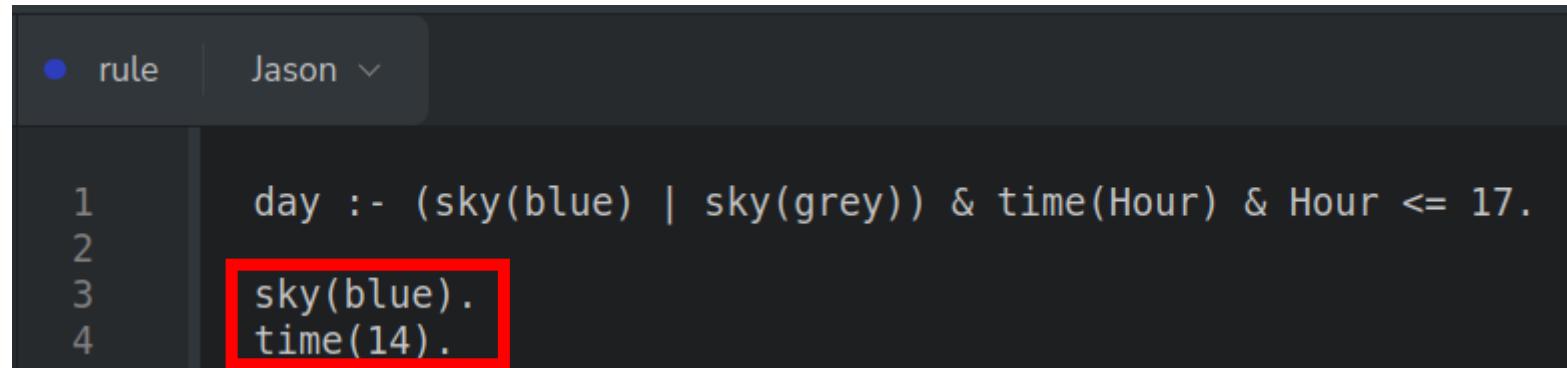


```
rule | Jason ▾  
1 day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.  
2  
3 sky(blue).  
4 time(14).
```



The verified conditions.

Rules: Predicate and Variable

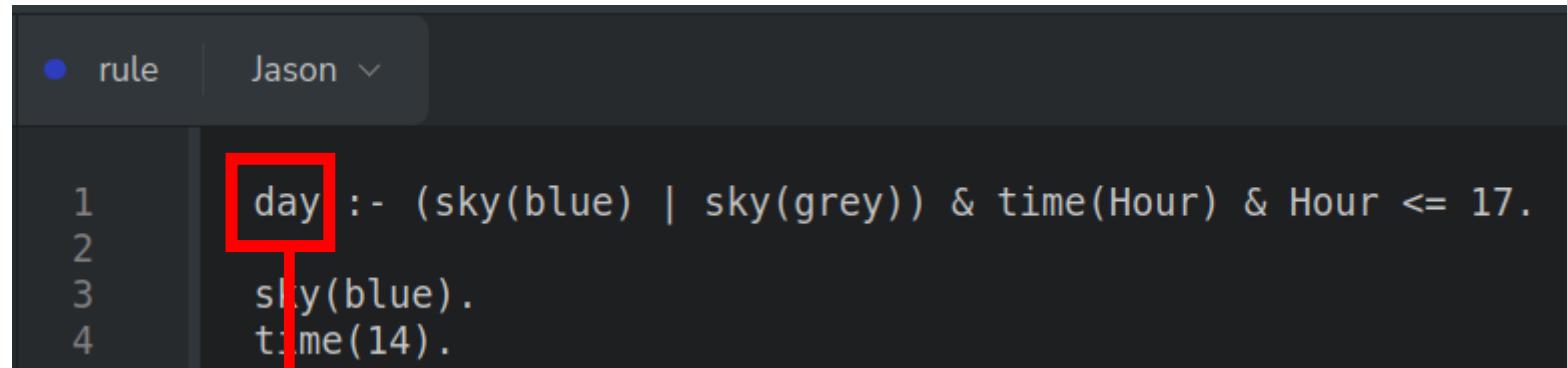


```
rule | Jason ▾  
1 day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.  
2  
3 sky(blue).  
4 time(14).
```



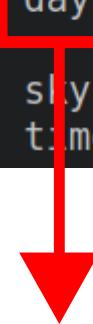
If the beliefs exist...

Rules: Predicate and Variable



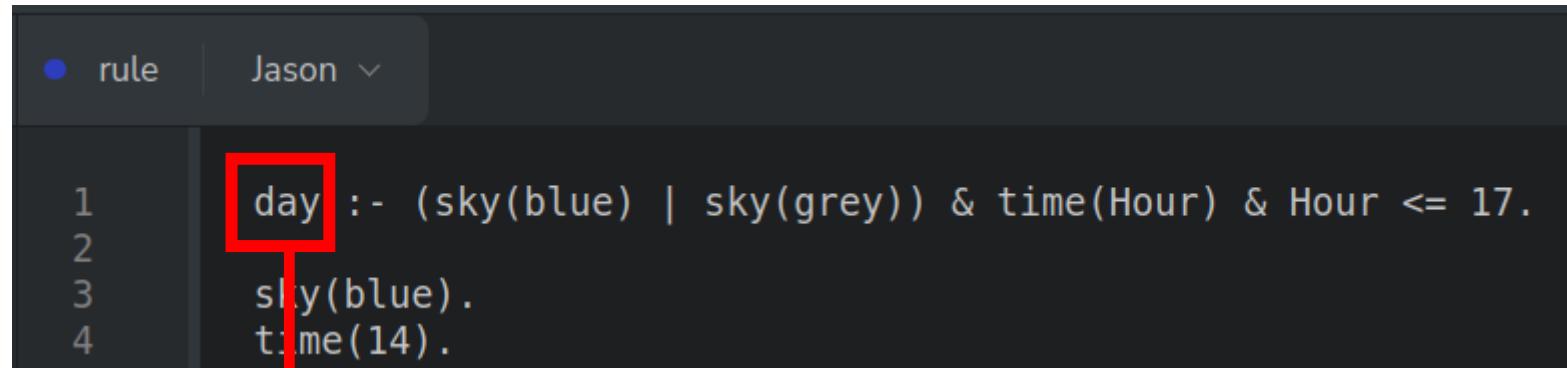
The screenshot shows a Prolog interface with the following code:

```
rule | Jason
1 day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.
2
3 sky(blue).
4 time(14).
```



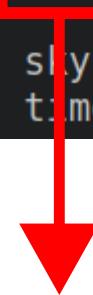
... the predicate holds, and agents can
use it ...

Rules: Predicate and Variable



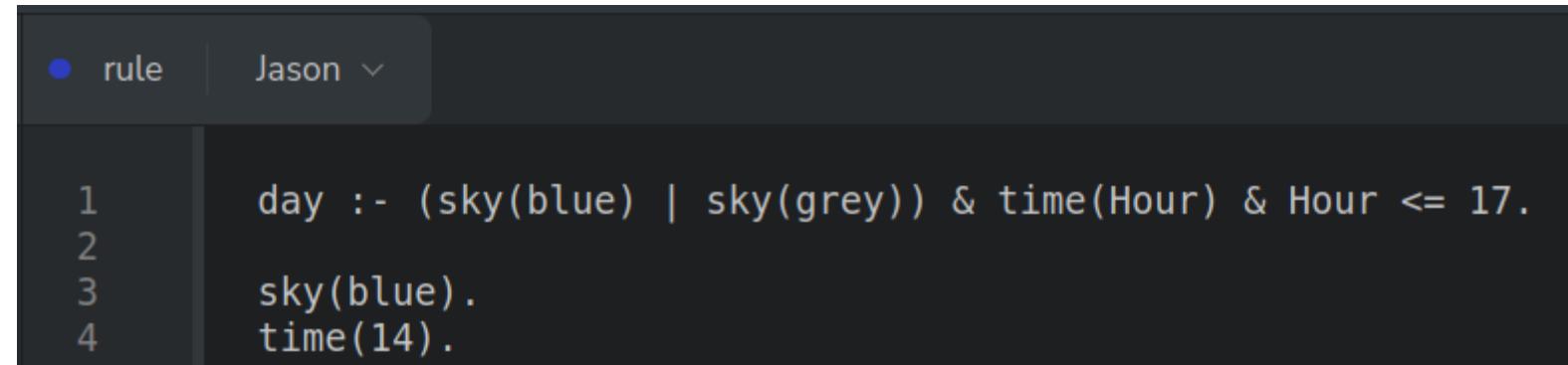
The screenshot shows a Prolog interface with the following code:

```
rule | Jason
1 day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.
2
3 sky(blue).
4 time(14).
```



... but it is NOT A BELIEF.

Rules: Predicate and Variable



```
rule Jason ▾  
1 day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.  
2  
3 sky(blue).  
4 time(14).
```

Rules: Predicate and Variable

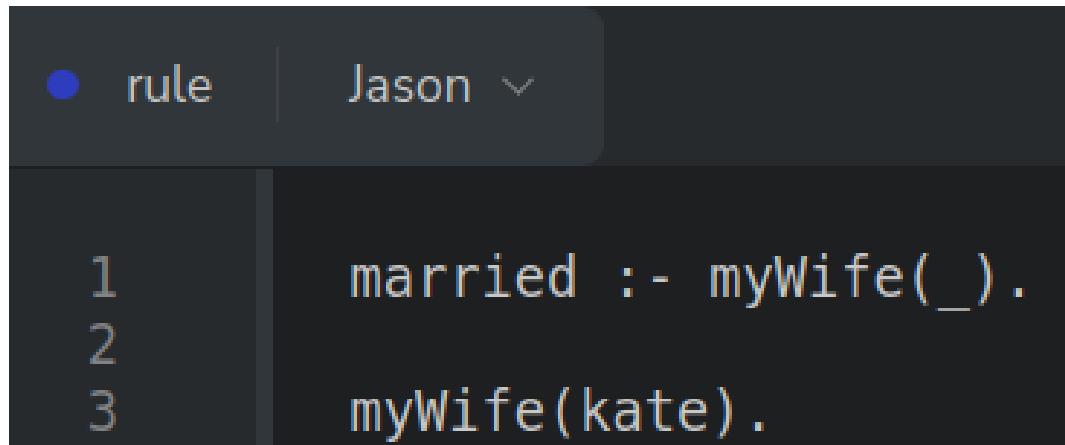
```
● rule Jason ▾  
1 day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.  
2  
3 sky(blue).  
4 time(14).
```



Inspection of agent rule

- Beliefs	sky(blue)[source(self)]. time(14)[source(self)].
- Rules	day :- ((sky(blue) sky(grey)) & (time(Hour) & (Hour <= 17))).
- Annotations	

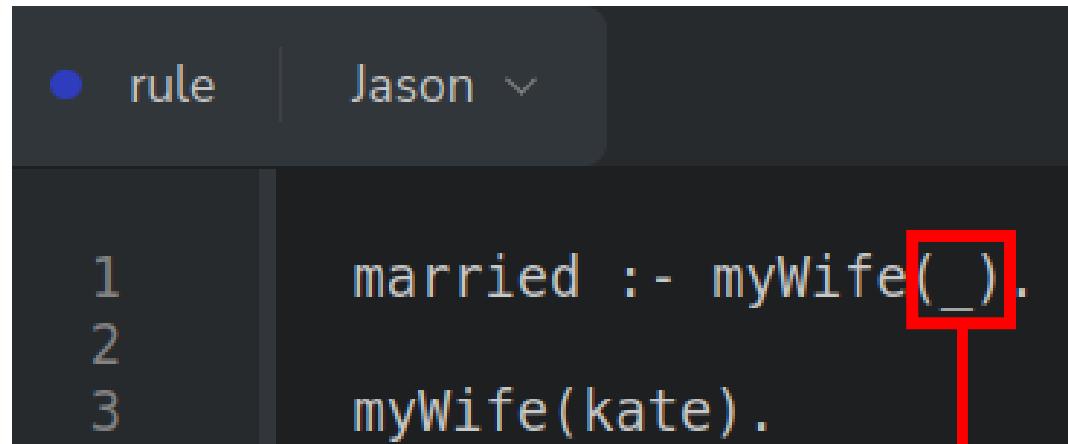
Rules: Underscore (Don't Care)



The screenshot shows the Jason rule editor interface. At the top, there is a navigation bar with a blue dot icon labeled "rule" and a dropdown menu labeled "Jason". Below the navigation bar, the code for two rules is displayed:

```
1 married :- myWife(_).
2
3 myWife(kate).
```

Rules: Underscore (Don't Care)



The screenshot shows a Prolog interface with the following code:

```
1 married :- myWife(_).  
2  
3 myWife(kate).
```

A red square box highlights the underscore character in the first line of code, and a red arrow points downwards from this box to the text "Don't care its value." located below the code.

Don't care its value.

Rules: Underscore (Don't Care)

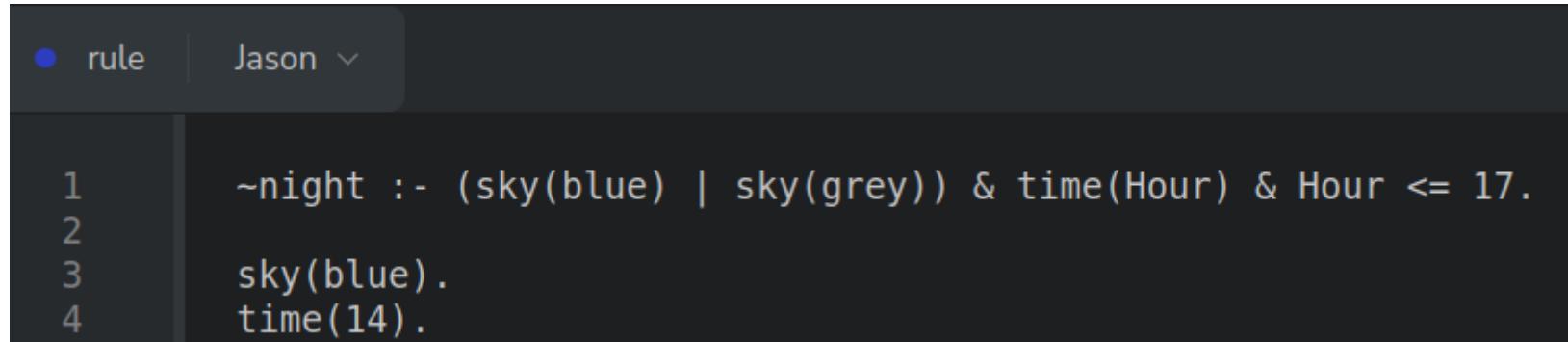
```
● rule | Jason ▾  
1 married :- myWife(_).  
2  
3 myWife(kate).
```



Inspection of agent rule

- Beliefs	myWife(kate)[source(self)].
- Rules	married :- myWife(_39).
- Annotations	

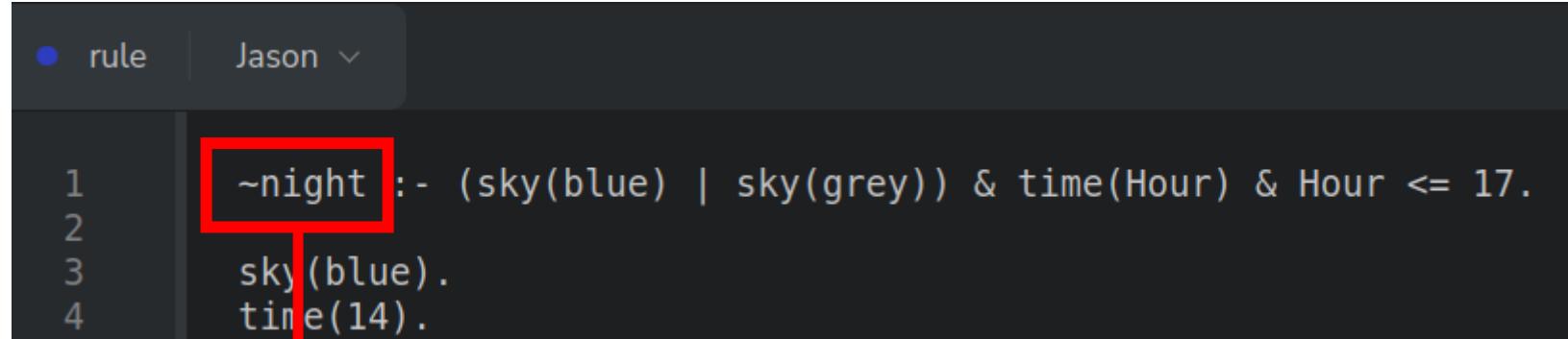
Rules: Strong Negation



The screenshot shows a user interface for editing Prolog-like rules. At the top, there's a navigation bar with a blue dot icon labeled "rule" and a dropdown menu labeled "Jason". Below this is a list of four numbered facts:

- 1 `~night :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.`
- 2 `sky(blue).`
- 3 `time(14).`

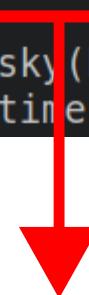
Rules: Strong Negation



The screenshot shows a Prolog interface with the following code:

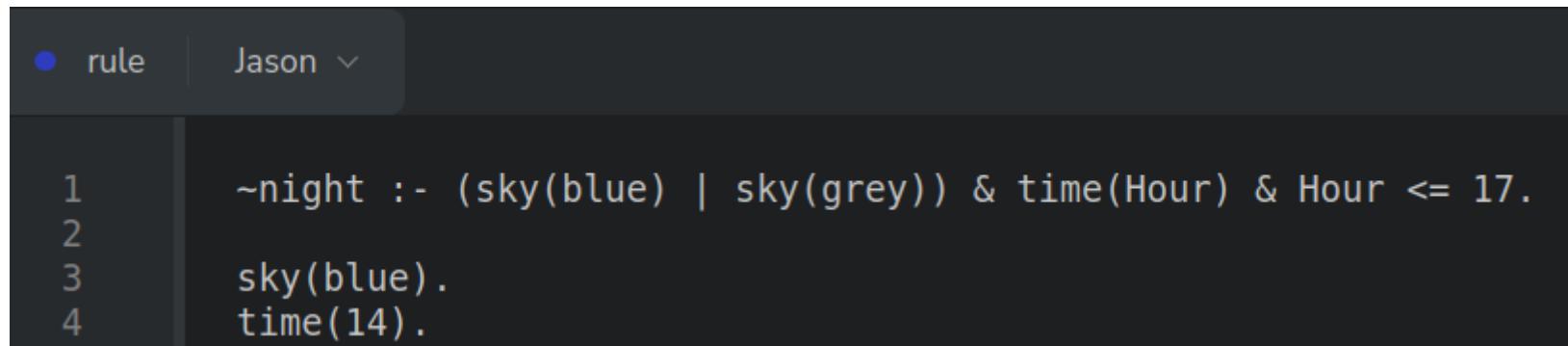
```
rule | Jason ▾
1  ~night :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.
2
3  sky(blue).
4  time(14).
```

The first rule, `~night :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.`, is highlighted with a red box around the predicate name `~night`.



~predicate

Rules: Strong Negation



The screenshot shows a Prolog IDE interface with a dark theme. At the top, there is a header bar with a blue dot icon labeled "rule" and a dropdown menu labeled "Jason". The main area displays four numbered facts:

- 1 `~night :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.`
- 2 `sky(blue).`
- 3 `time(14).`
- 4

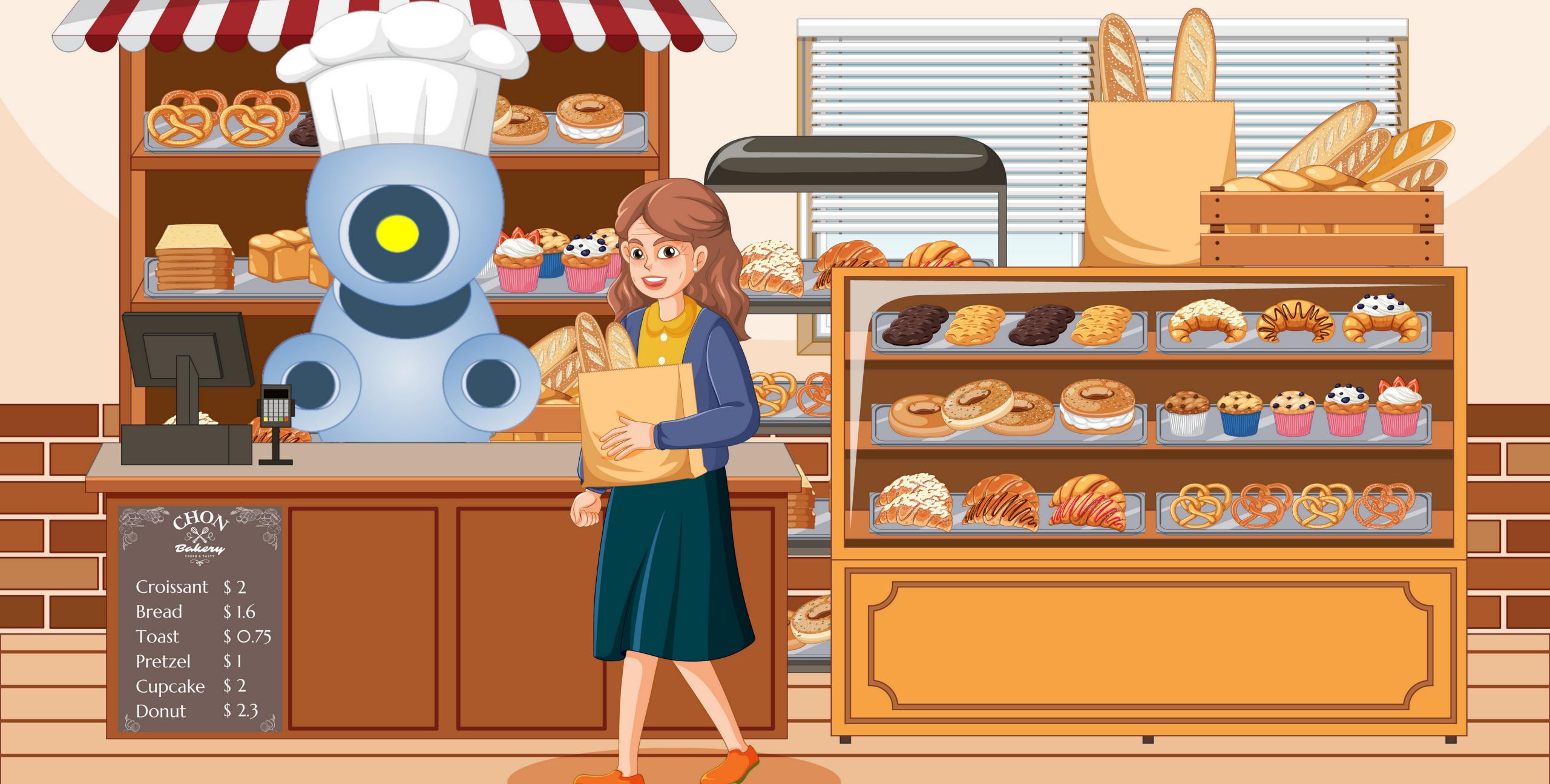
Rules: Strong Negation

```
rule Jason
1 ~night :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.
2
3 sky(blue).
4 time(14).
```

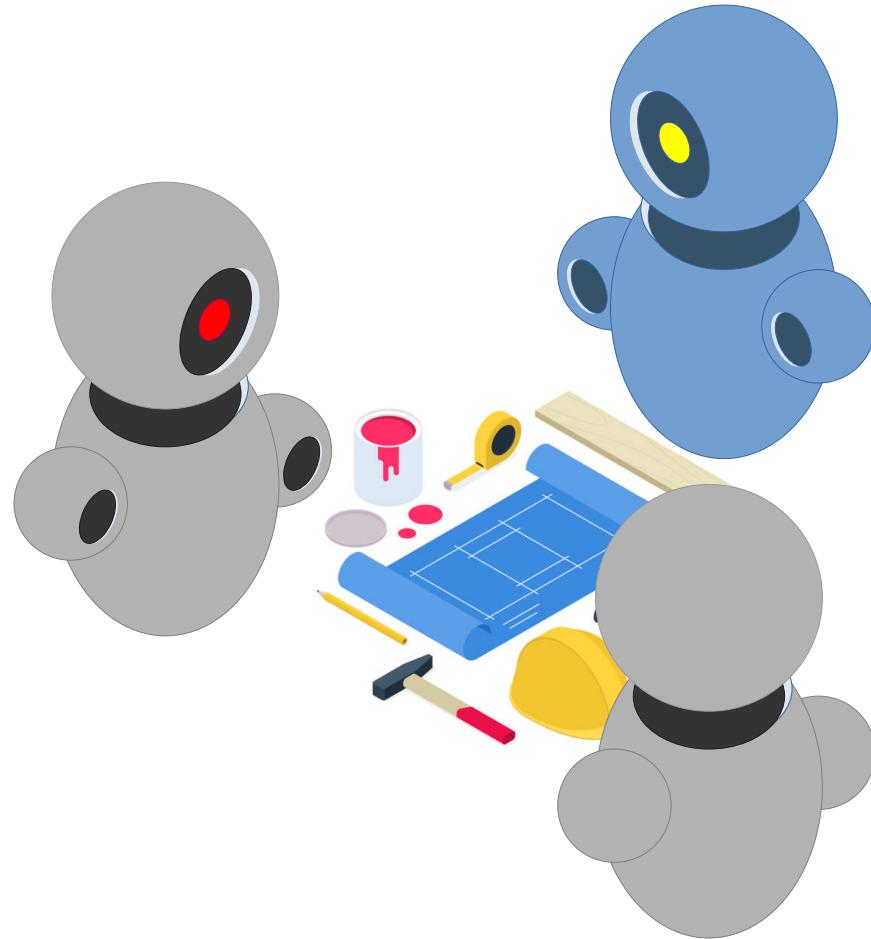


Inspection of agent rule

- Beliefs	sky(blue)[source(self)]. time(14)[source(self)].
- Rules	~night :- ((sky(blue) sky(grey)) & (time(Hour) & (Hour <= 17))).
- Annotations	



GOALS AND PLANS

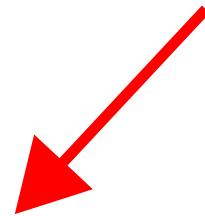


Jason Framework: Goals

Em Jason, os **goals** (objetivos) representam os estados do mundo em que o agente deseja atingir.

Jason Framework: Goals

Em Jason, os **goals** (objetivos) representam os estados do mundo em que o agente deseja atingir.

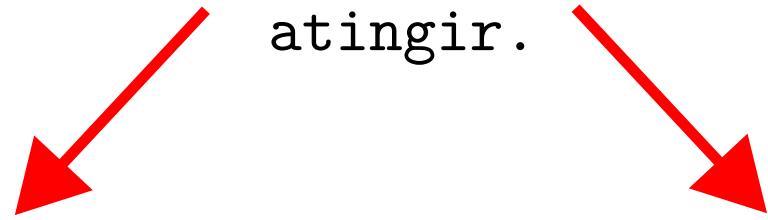


1. realização (!)

É um objetivo para atingir determinado estado desejado pelo agente.

Jason Framework: Goals

Em Jason, os **goals** (objetivos) representam os estados do mundo em que o agente deseja atingir.



1. realização (!)

É um objetivo para atingir determinado estado desejado pelo agente.

2. teste (?)

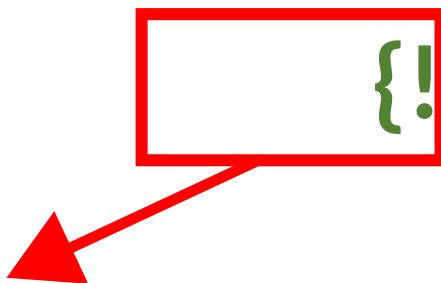
É um objetivo que tem a finalidade de resgatar informações da base de crenças do agente.

Goals: Format

{!|?}event [source(type)]

Goals: Format

{!|?}event [source(type)]

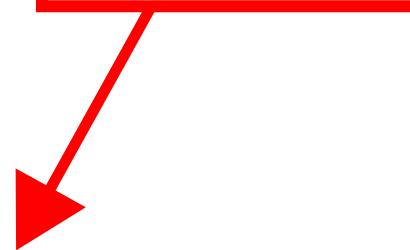


the goal
type



Goals: Format

{. | ?}event [source(type)]



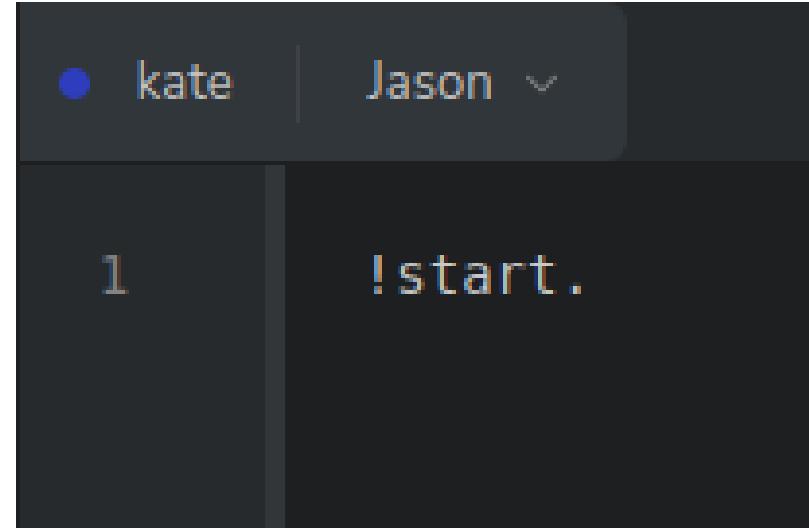
a predicate from
logic.

Goals: Format

{! | ?}event [source(type)]

the source of the
belief.

Goals: Initial Goal



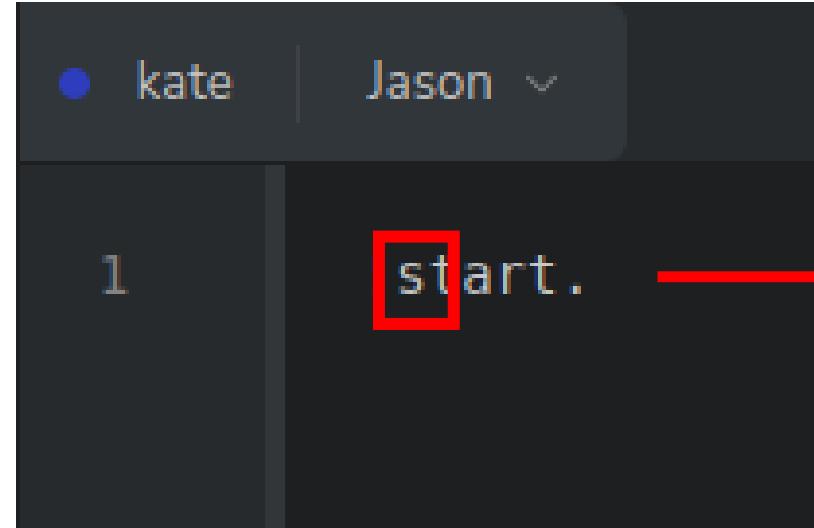
Goals: Initial Goal



A screenshot of a terminal window. At the top, there is a dark header bar with the text "kate" and "Jason". Below this, the terminal itself has a dark background. On the left side of the terminal, the number "1" is displayed. In the center, the text "!start." is visible, with the exclamation point "!start." highlighted by a red square box. A red arrow points from this box towards the explanatory text on the right.

every goal must
start with an
exclamation point.

Goals: Initial Goal

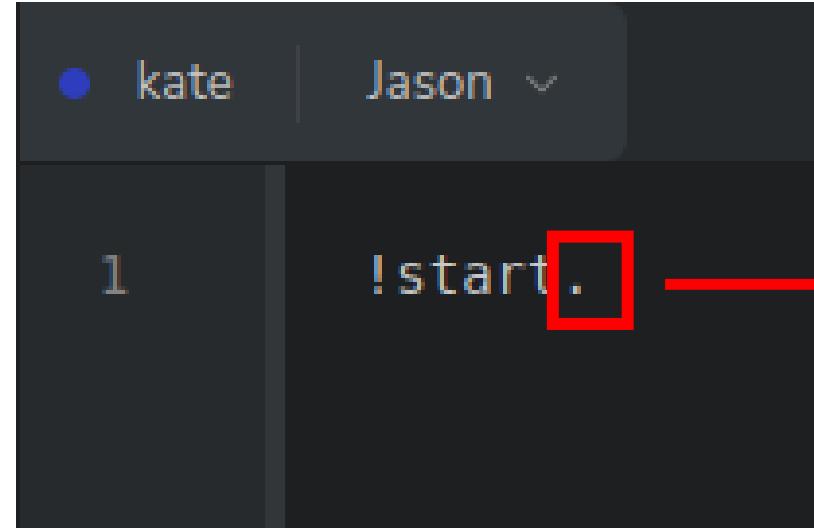


```
● kate | Jason ~
1
start.
```

A screenshot of a terminal window. At the top, there are user icons for 'kate' and 'Jason' with a dropdown arrow. Below this, the number '1' is displayed. In the main terminal area, the word 'start.' is typed, with the letter 's' highlighted by a red square box. A red arrow points from this highlighted 's' to the explanatory text on the right.

every goal's predicate must start with a lowercase letter.

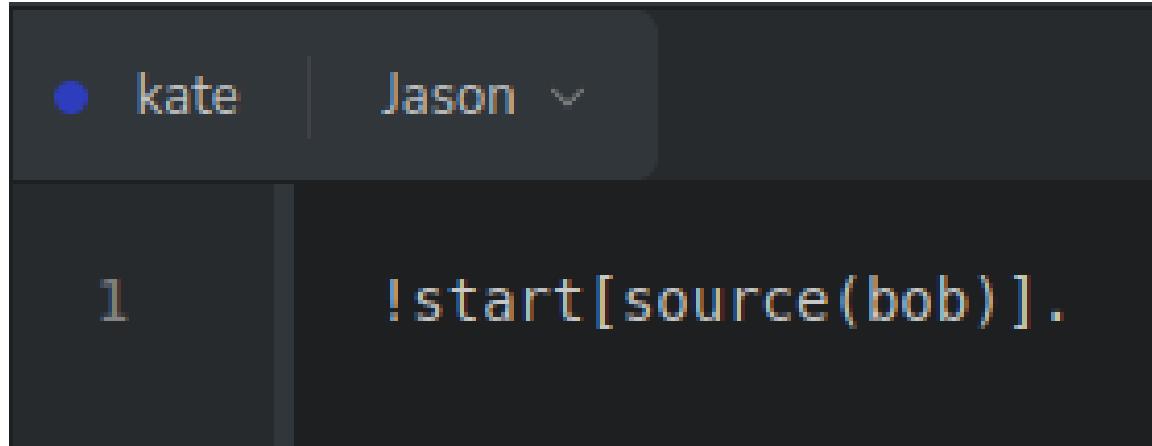
Goals: Initial Goal



A screenshot of a terminal window. At the top, there is a dark header bar with a blue dot icon, the name "kate", and a dropdown menu labeled "Jason". Below the header, the terminal window has a dark background. On the left side, there is a vertical dark sidebar with the number "1" at its top. The main area of the terminal contains the text "!start." where the period character "." is highlighted with a red square box. A red arrow points from this red box towards the explanatory text on the right.

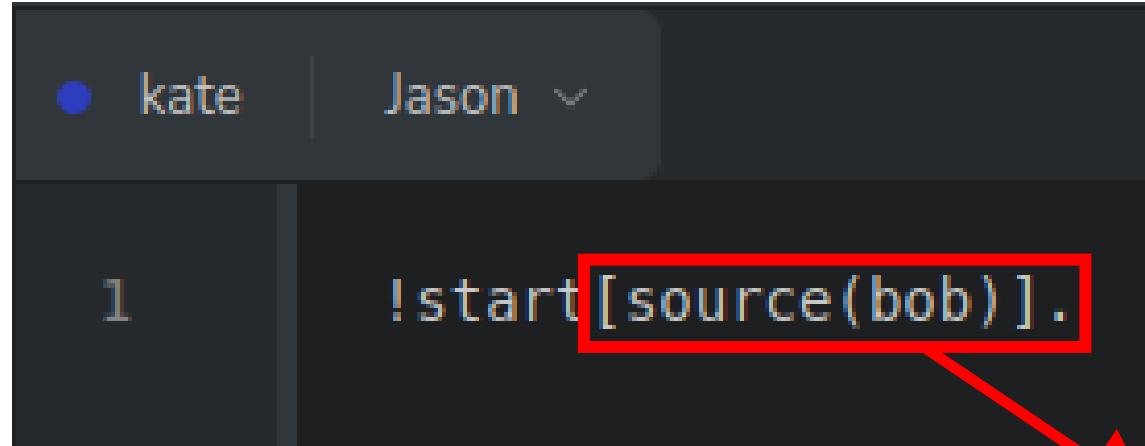
and it must end with
a period.

Goals: Initial Goal with a Source



The image shows a terminal window with a dark background. At the top, there is a header bar with a blue dot icon, the name "kate" in blue, and "Jason" in orange with a dropdown arrow. Below the header, the terminal window is divided into two vertical panes. The left pane contains the number "1". The right pane contains the command: `!start[source(bob)].`

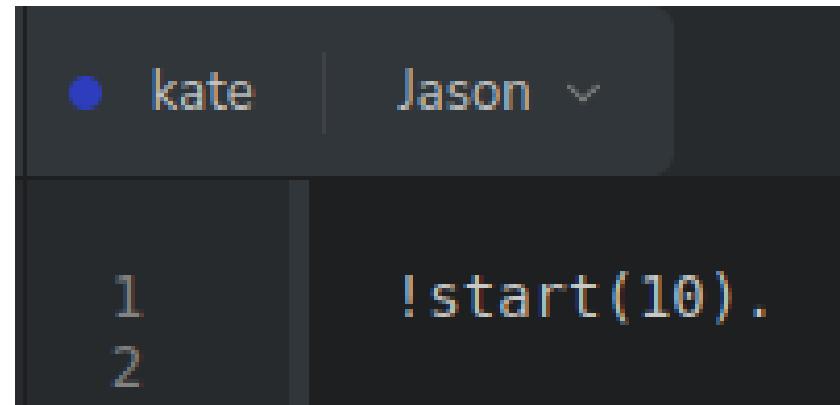
Goals: Initial Goal with a Source



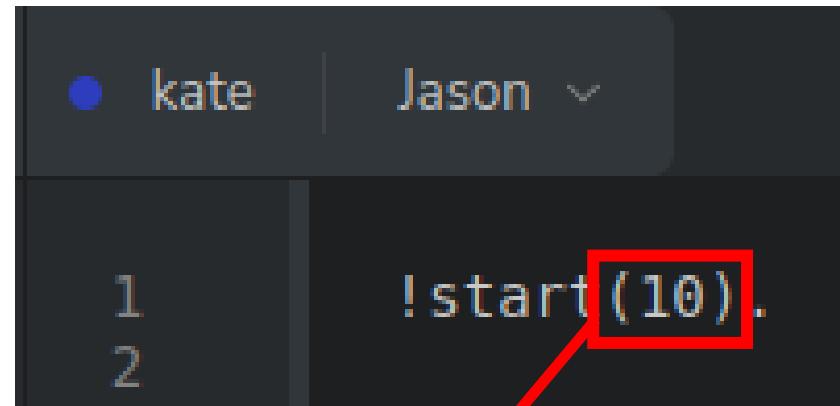
A screenshot of a terminal window with a dark background. At the top, there is a user interface element with a blue dot icon, the name "kate", and another name "Jason" followed by a dropdown arrow. Below this, the terminal prompt "1" is visible. To the right of the prompt, the command "!start [source(bob)]." is displayed. The entire command line is highlighted with a red rectangular box. A red arrow points from the bottom right towards the text "one can define a source." located below the terminal window.

one can define a
source.

Goals: Initial Goal with Predicate



Goals: Initial Goal with Predicate



predicate(int)

Goals: Initial Goal with Predicate

predicate(value)

Goals: Initial Goal with Predicate

predicate(value)

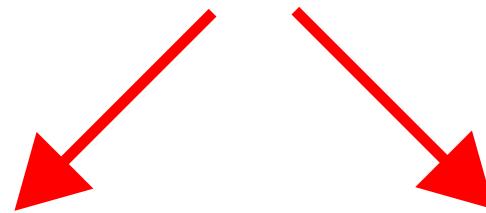


predicate

Goals: Initial Goal with Predicate

predicate(value)

predicate



int, float, String, etc.

Goals: Format

predicate(predicate)

Goals: Format

predicate(predicate)

Goals: Format

predicate(predicate(predicate))

Goals: Format

predicate(predicate(predicate))

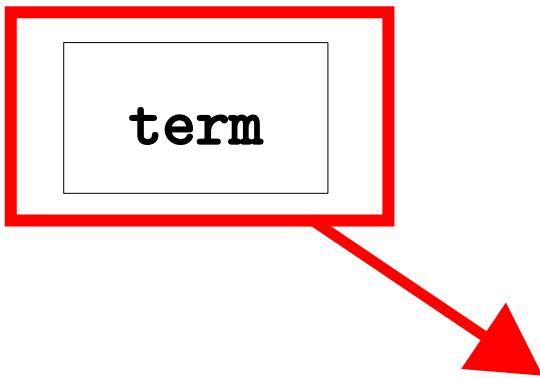
Goals: Format

predicate(predicate(predicate(...))))

Logic-Based Programming

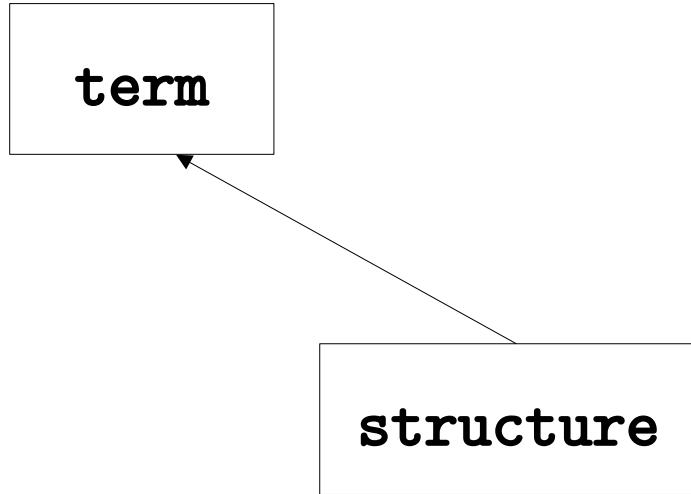
term

Logic-Based Programming

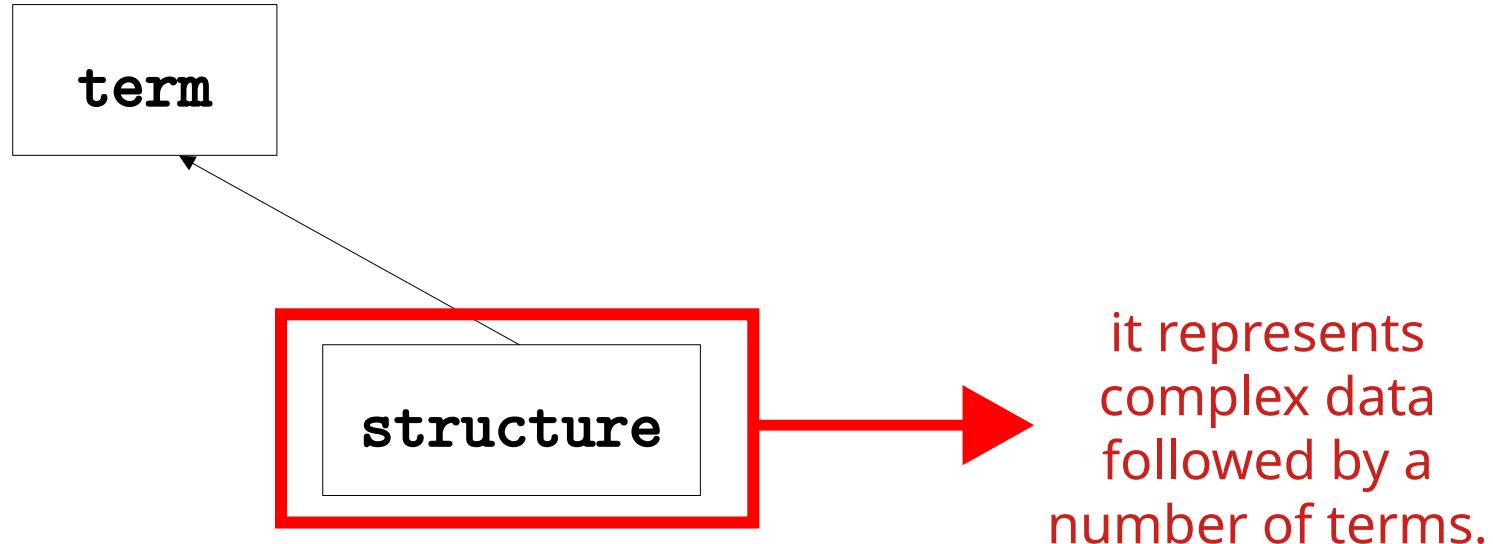


Every construction
in Jason is a term.

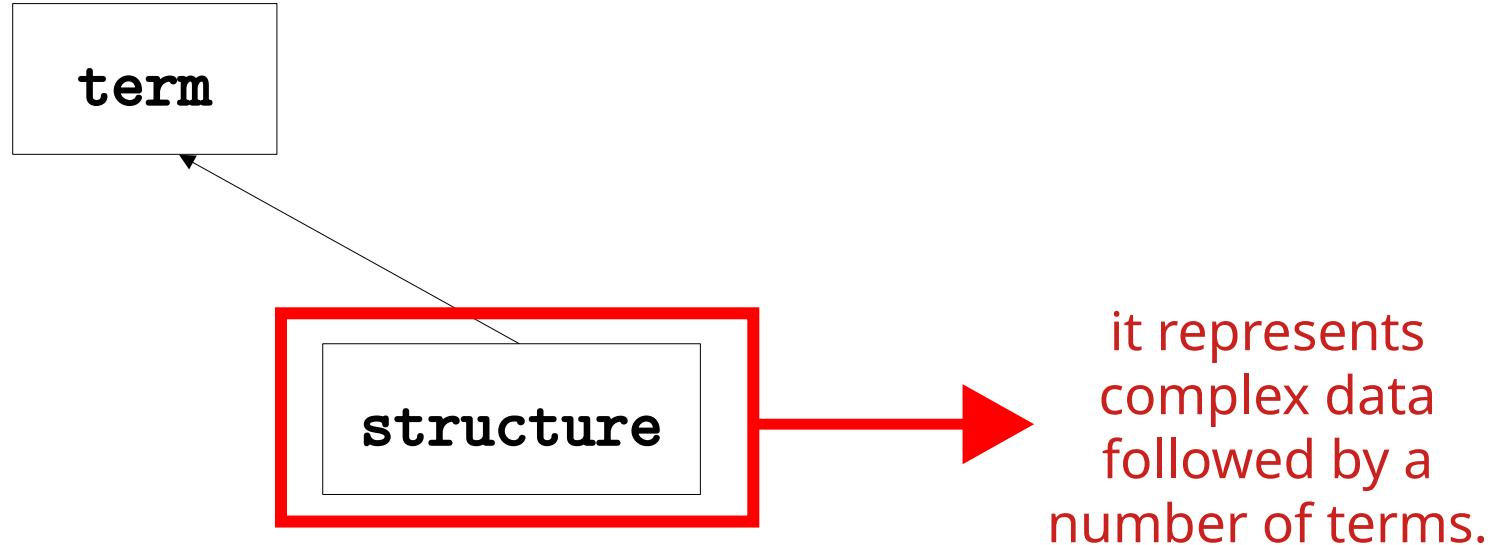
Logic-Based Programming



Logic-Based Programming



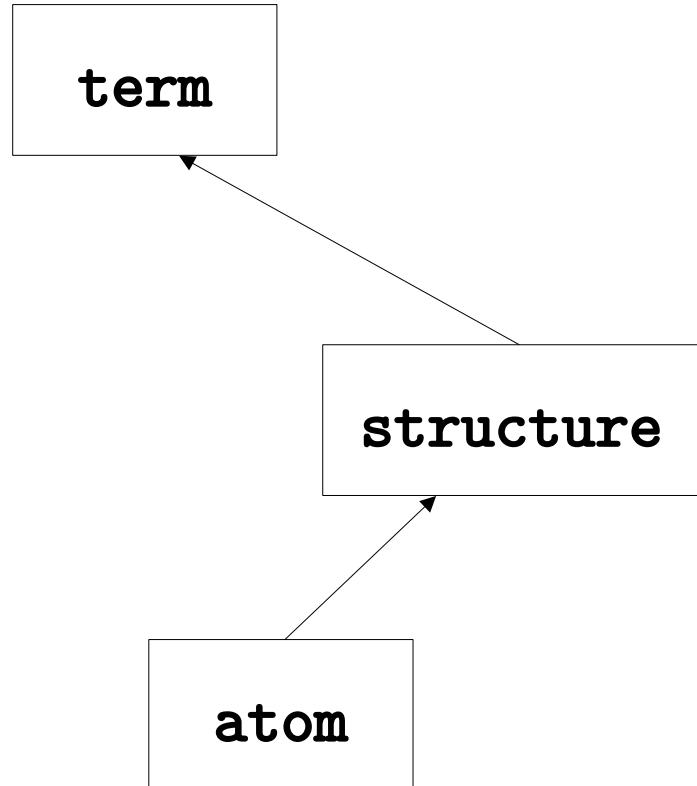
Logic-Based Programming



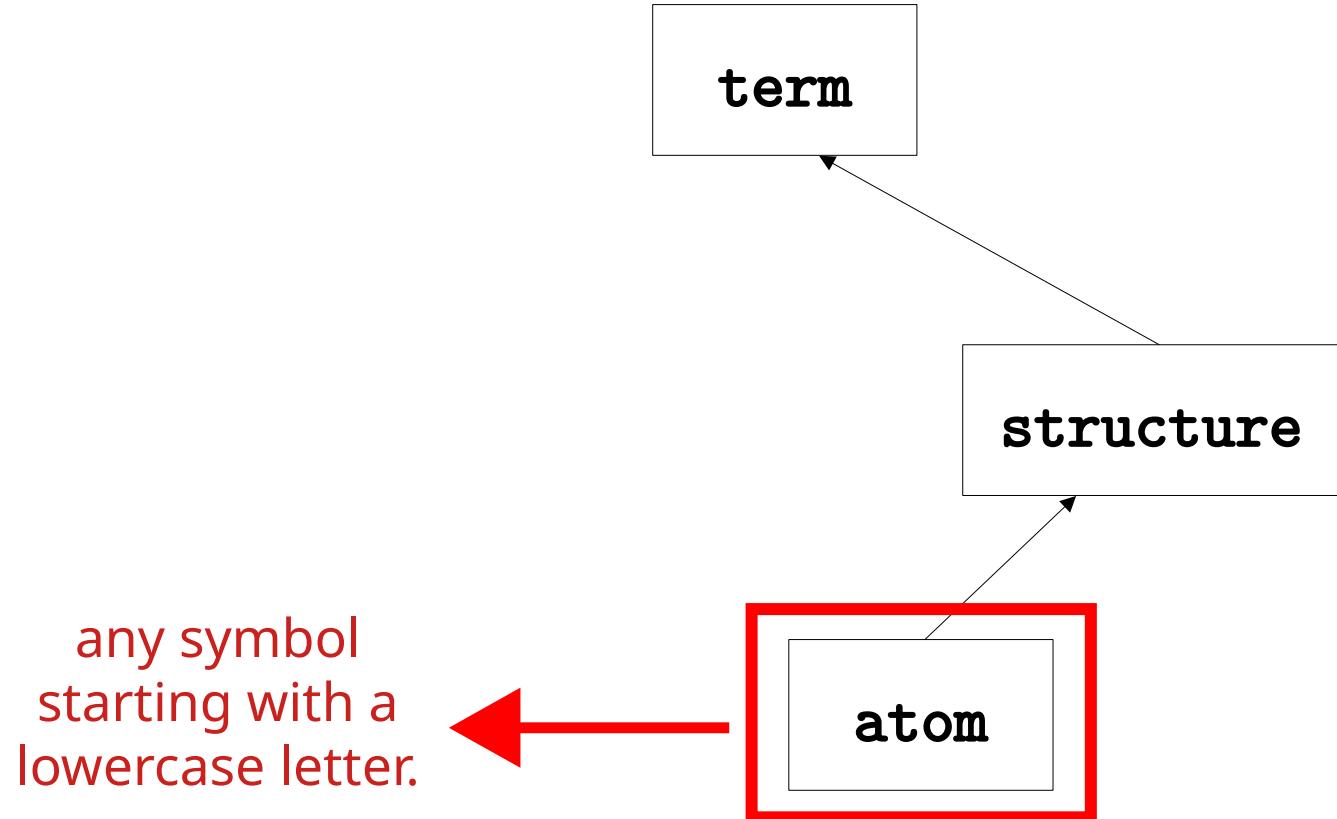
name(kate,18)



Logic-Based Programming



Logic-Based Programming

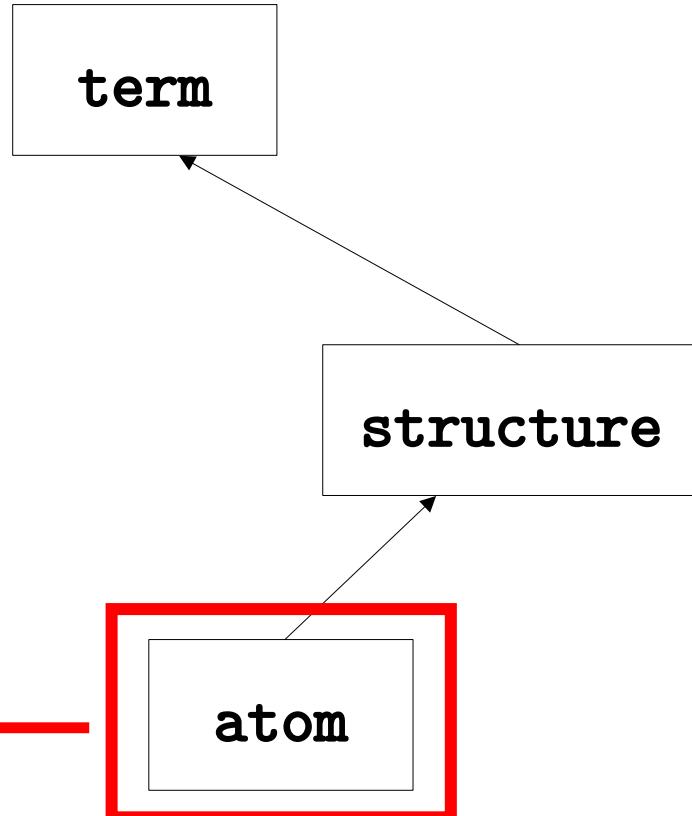


Logic-Based Programming

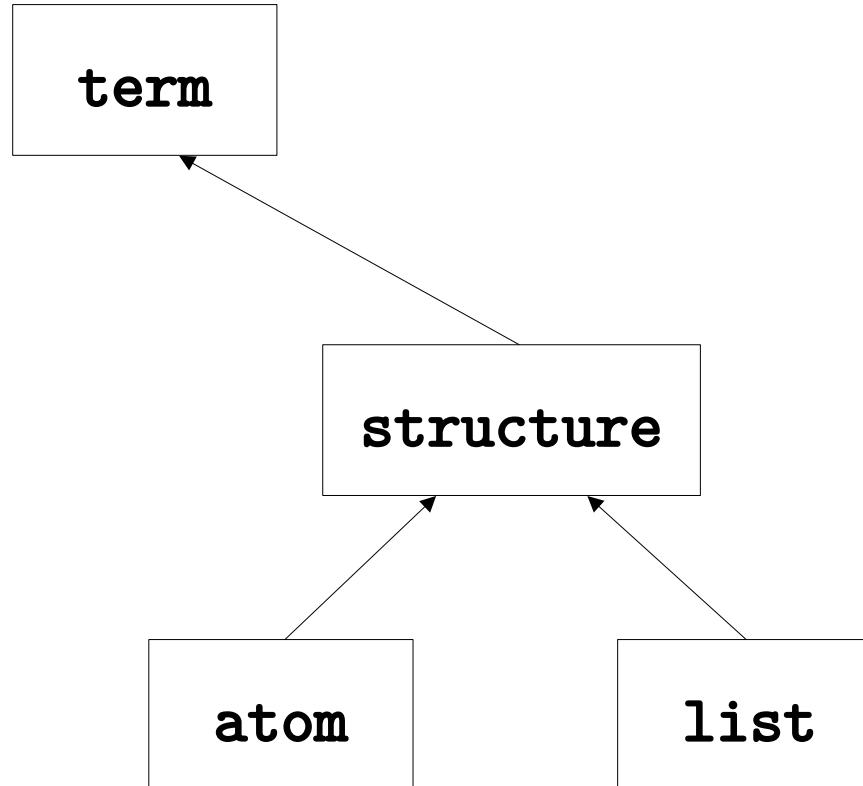
sunny



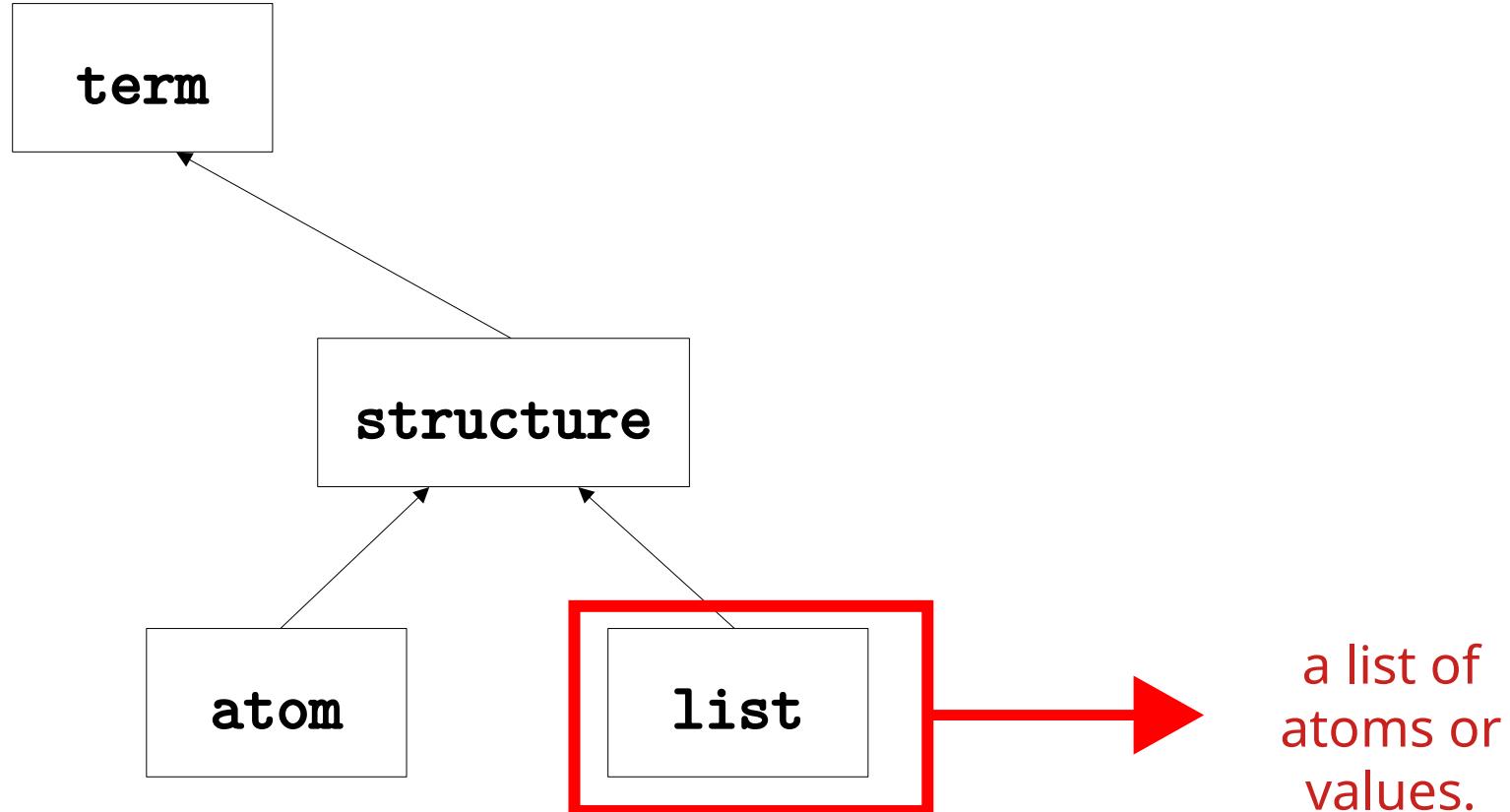
any symbol
starting with a
lowercase letter.



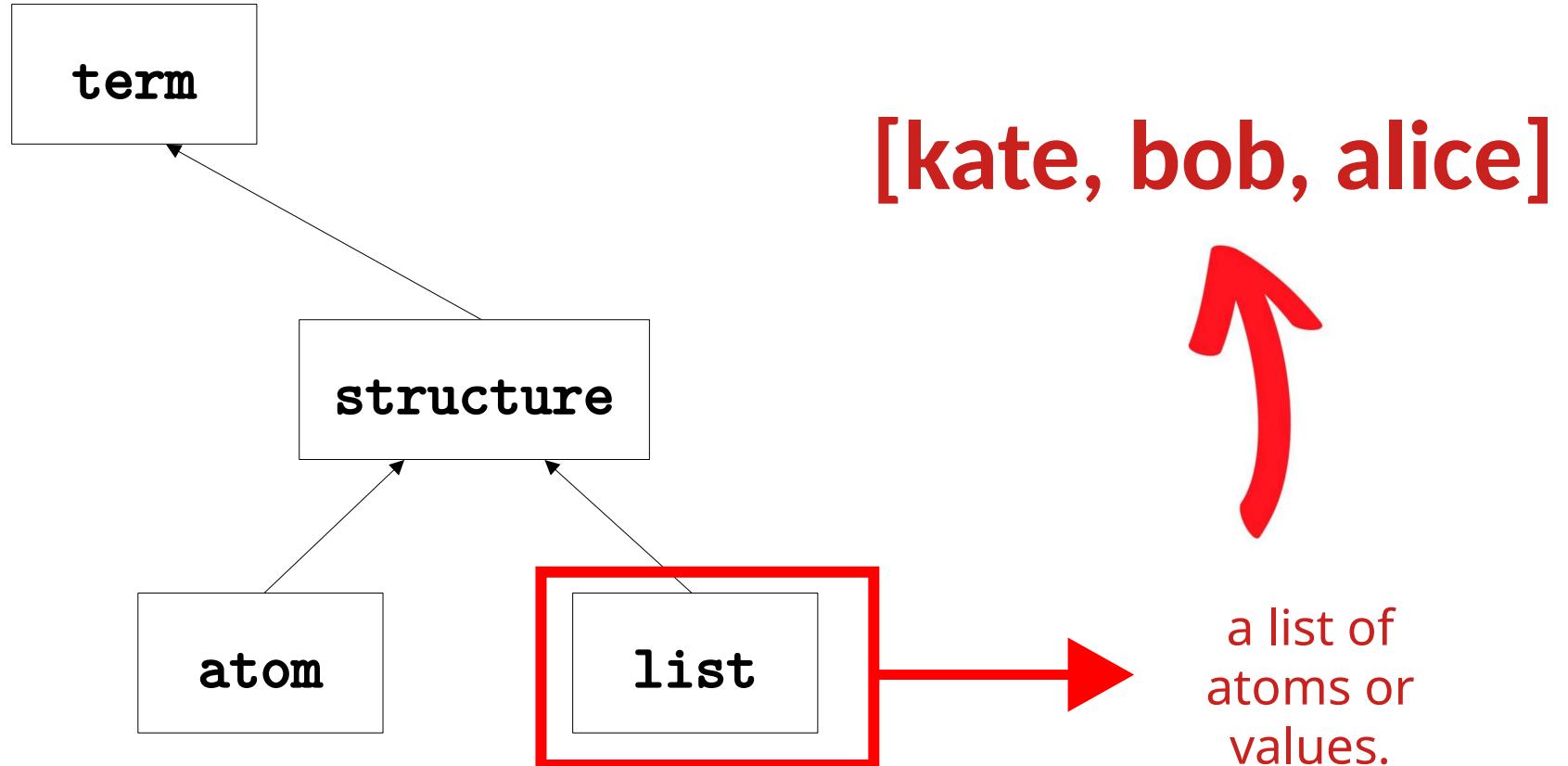
Logic-Based Programming



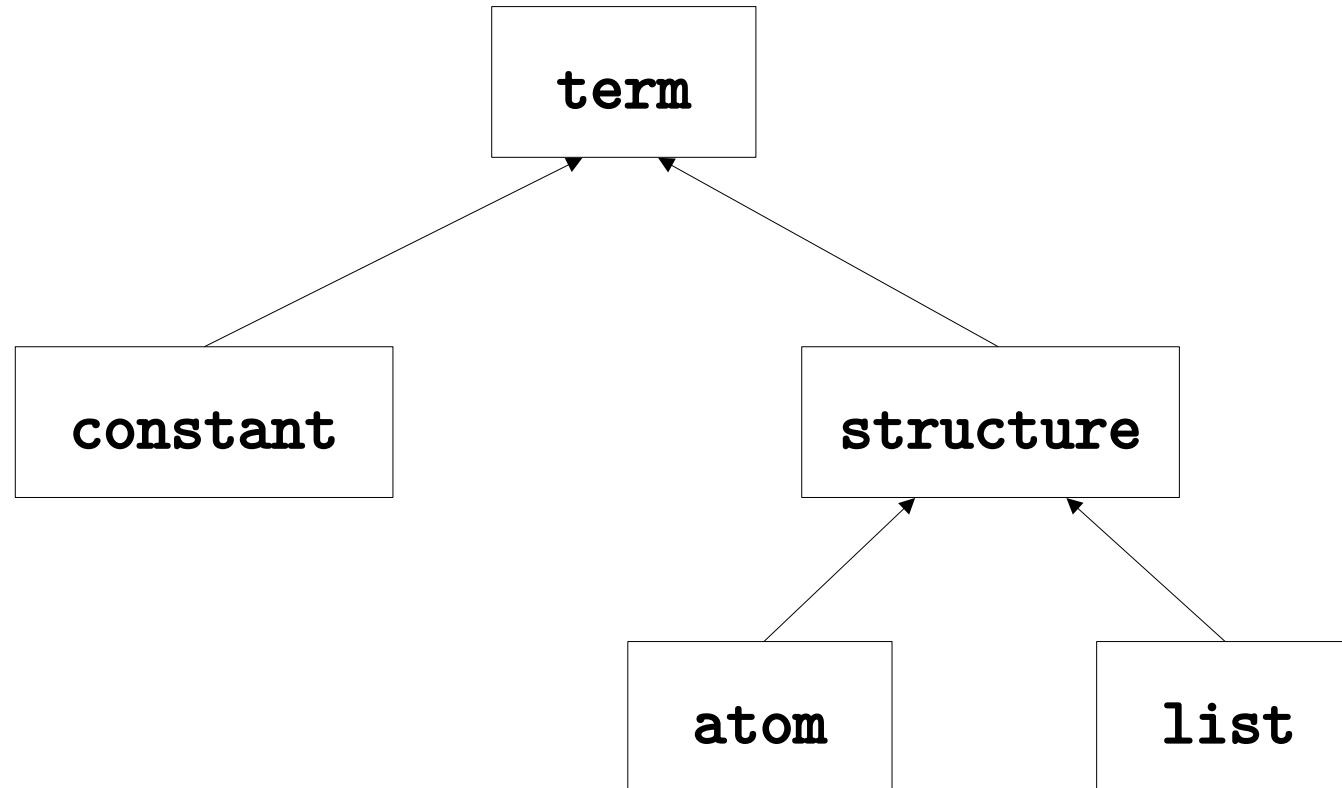
Logic-Based Programming



Logic-Based Programming

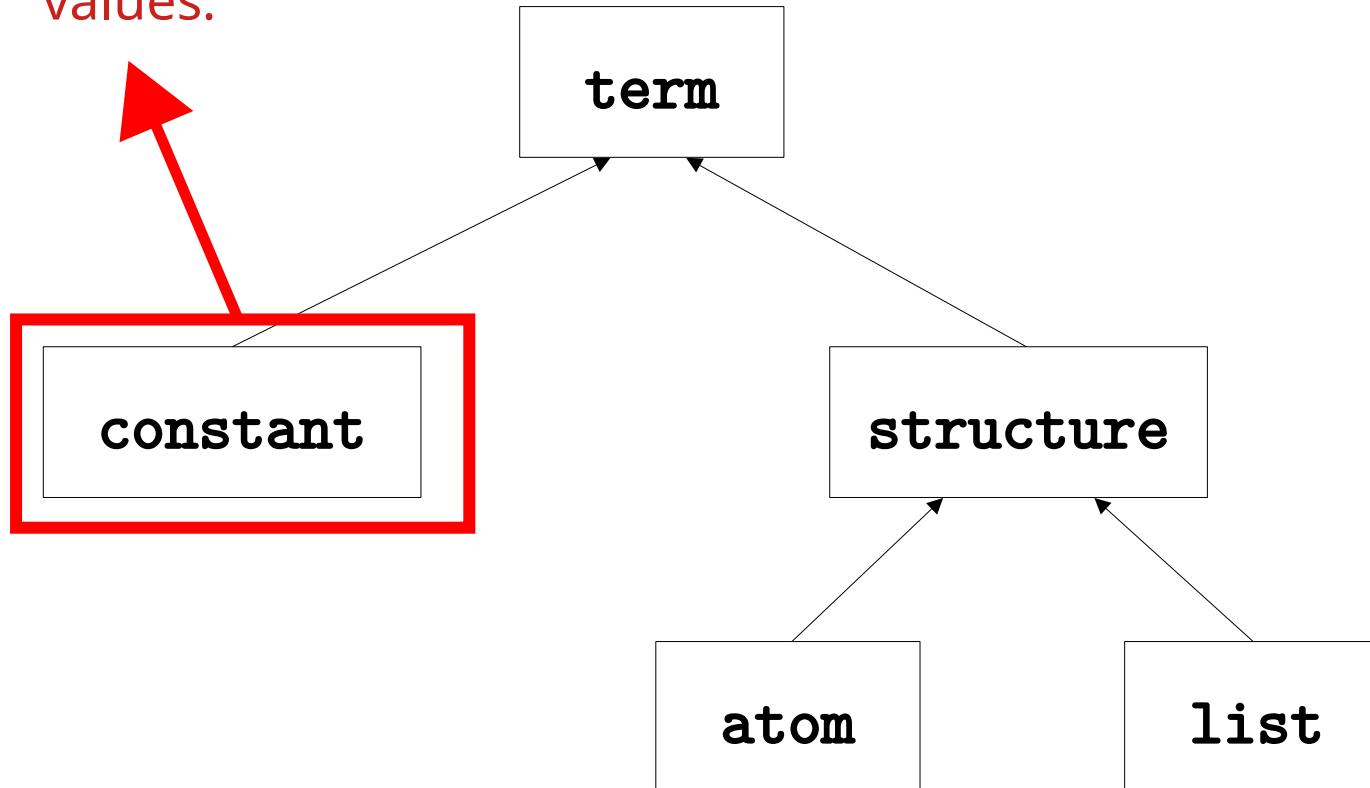


Logic-Based Programming



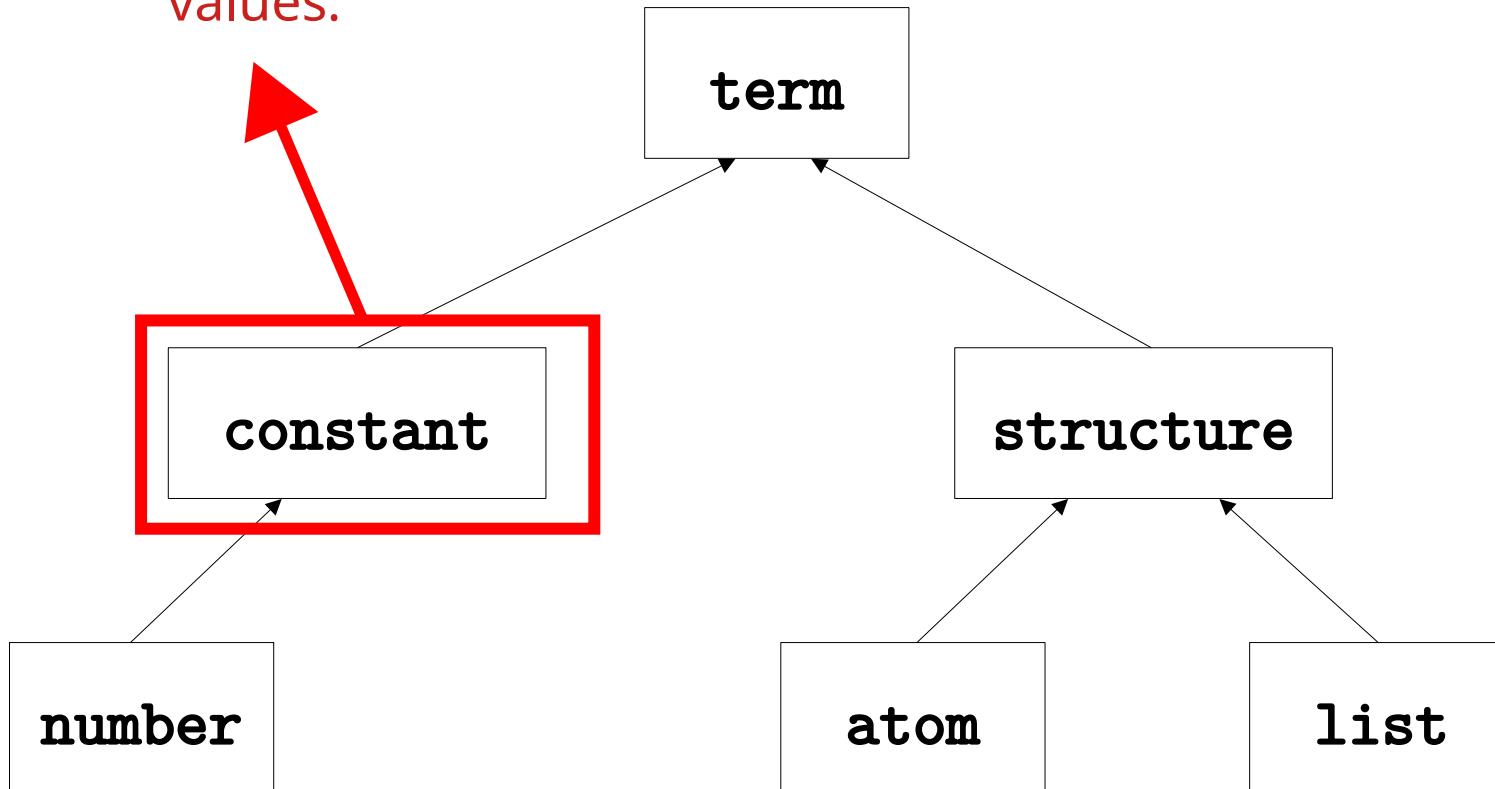
Logic-Based Programming

values.



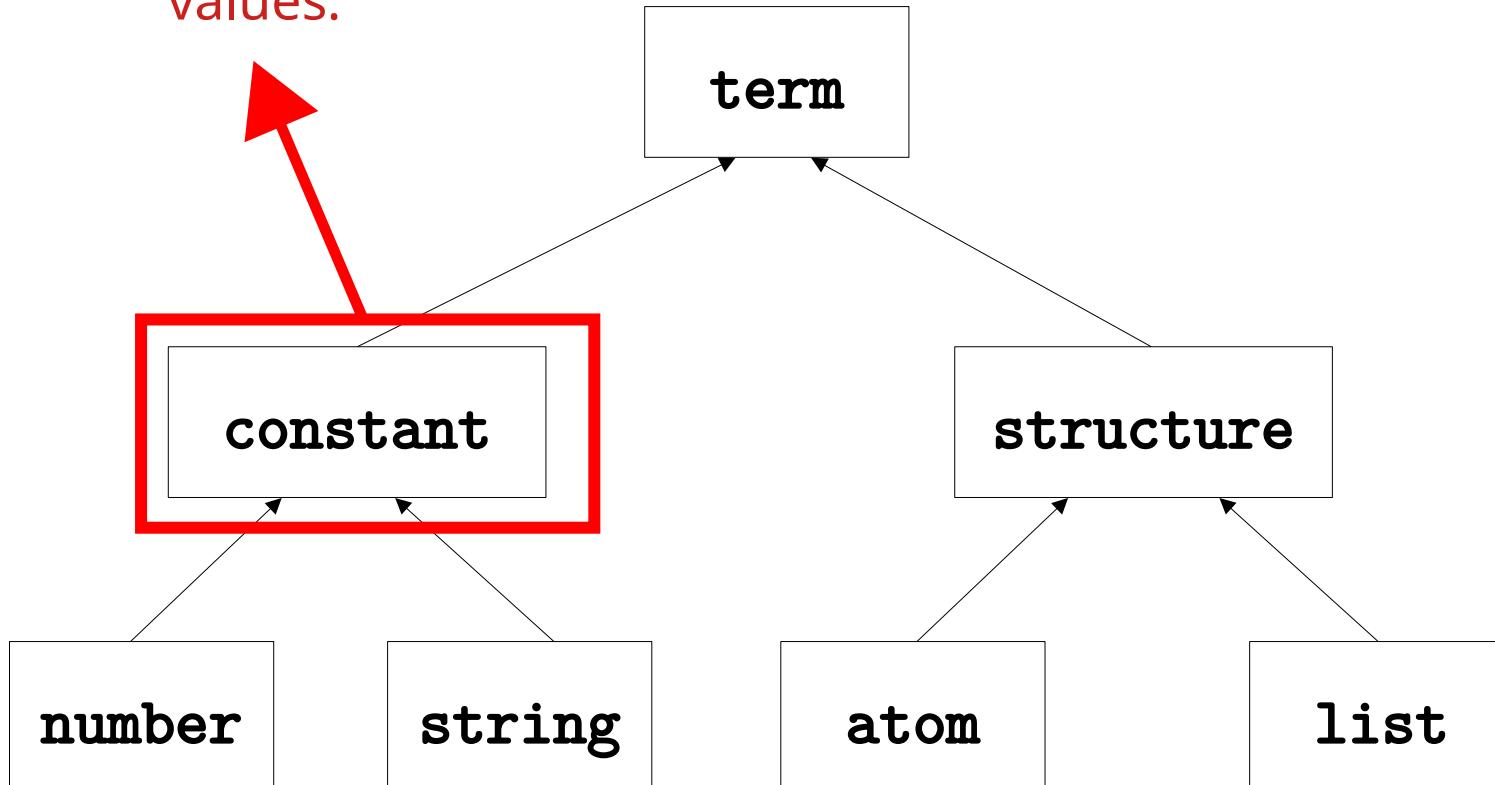
Logic-Based Programming

values.



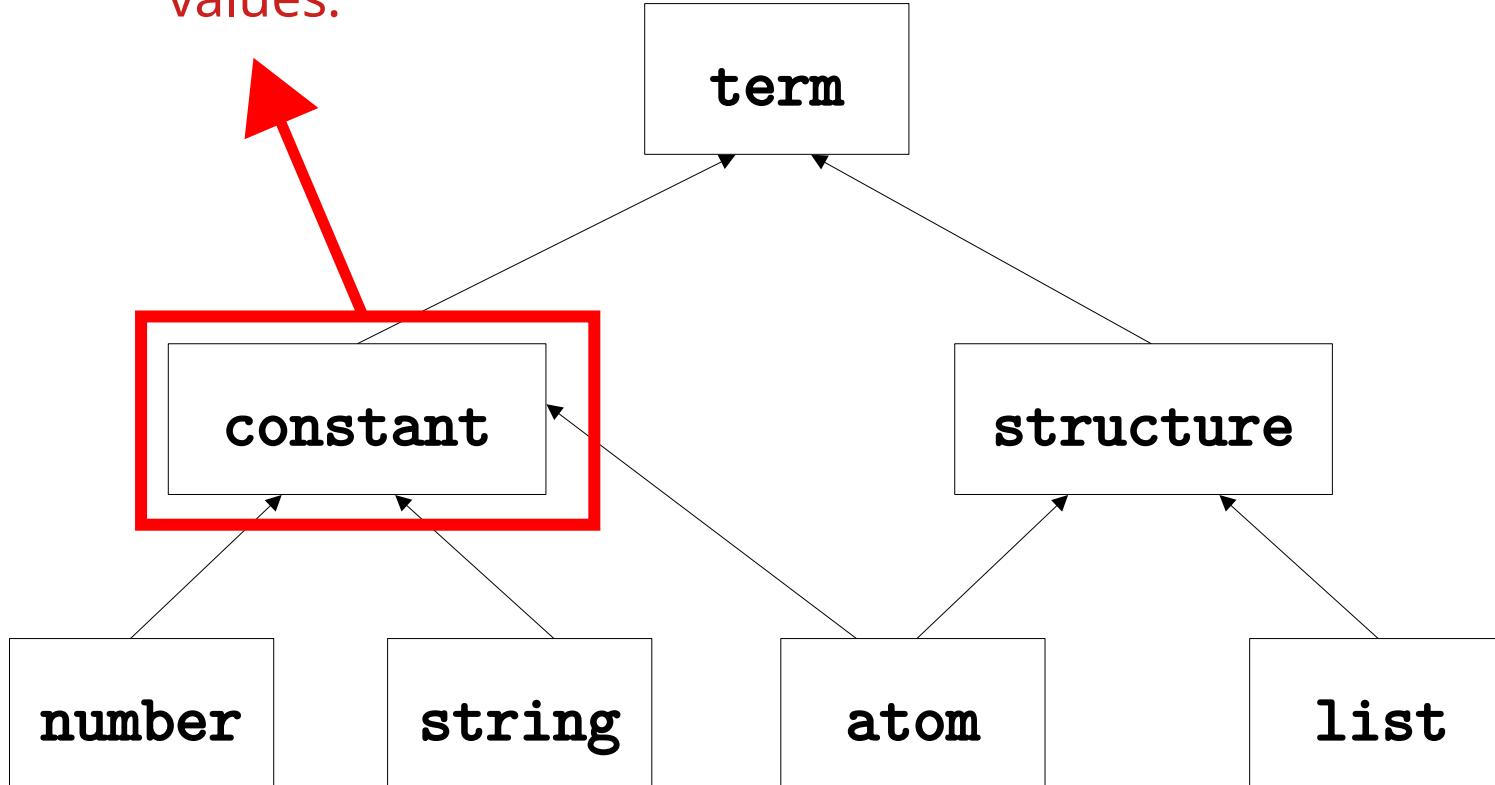
Logic-Based Programming

values.



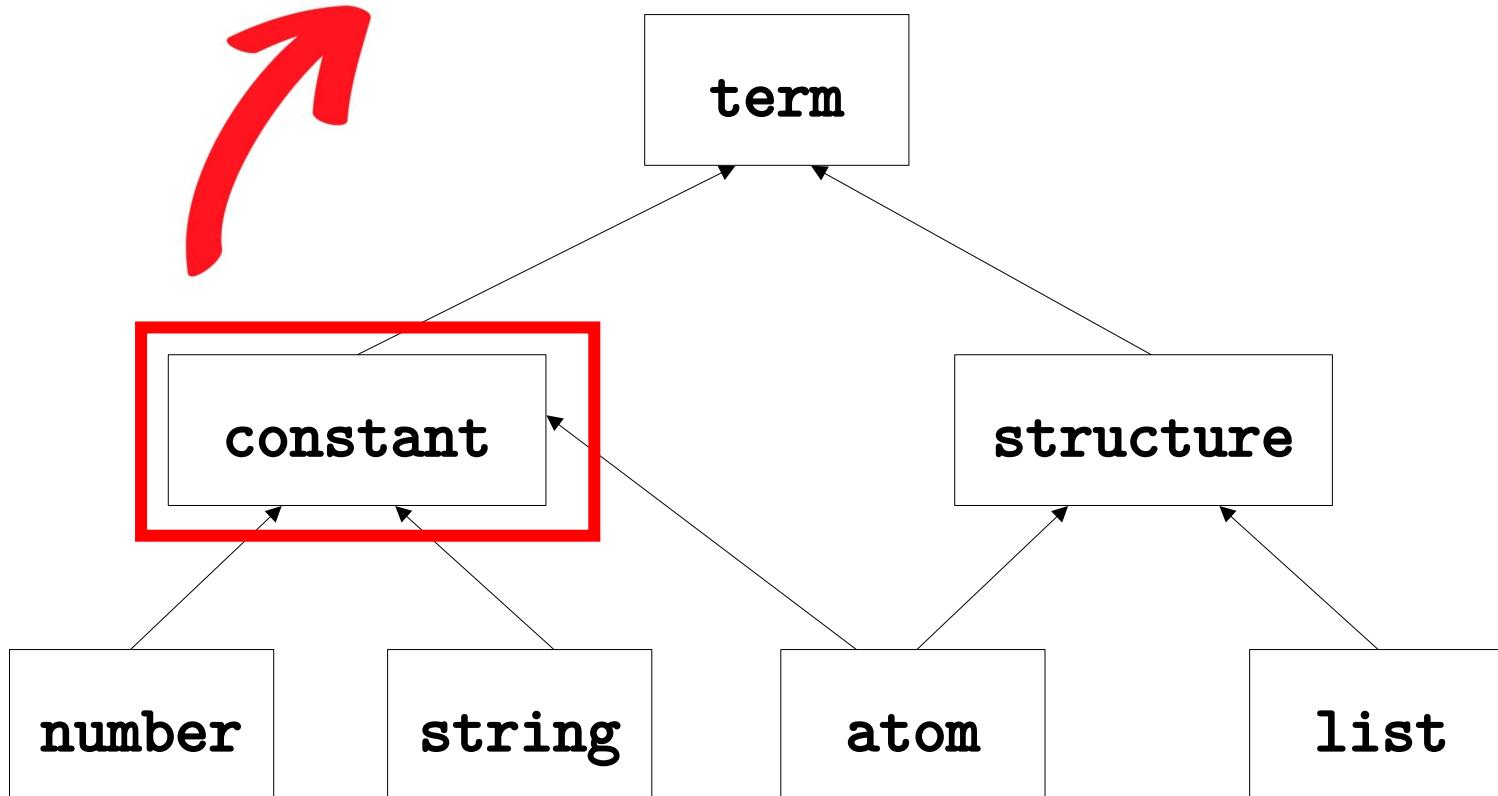
Logic-Based Programming

values.



Logic-Based Programming

employer(**kate**, 3582, “kate@kate.com”, full(40))



Jason Framework: Plans Format

triggering_event : context <- body.

triggering_event : context <- body.

1. triggering Event

- Um agente pode ter diversos objetivos. Os planos são ativados baseados nos eventos que podem ser ativados em determinado momento.

triggering_event : context <- body.

1. triggering Event

- Um agente pode ter diversos objetivos. Os planos são ativados baseados nos eventos que podem ser ativados em determinado momento.

2. context

- São as condições para a ativação de um plano dentro vários eventos.

triggering_event : context <- body.

1. triggering Event

- Um agente pode ter diversos objetivos. Os planos são ativados baseados nos eventos que podem ser ativados em determinado momento.

2. context

- São as condições para a ativação de um plano dentro vários eventos.

3. body.

- É o corpo do plano. Uma sequência de ações a ser executada pelo agente.

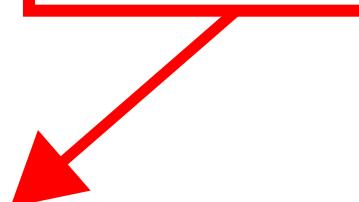
Plan: Format

{+|-}{!|?}event [source(type)]: context ←
action 1;
action 2;
action n.

Plan: Format

{+ | - }{!} ? }event [source(type)]:
action 1;
action 2;
action n.

The change type.



Plan: Format

{+ | -} {! | ?} event [source(type)]:
action 1;
action 2;
action n.

The type of the
goal.

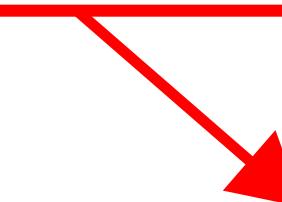
Plan: Format

```
{+|-}{!|?}event [source(type)]: context ←  
action 1;  
action 2;  
action n.
```

The triggering event.

Plan: Format

```
{+|-}{!|?}event [source(type)]: context ←  
action 1;  
action 2;  
action n.
```



the source of the
plan.

Plan: Format

```
{+|-}{!|?}event [source(type)]:  
    action 1;  
    action 2;  
    action n.
```

context :-



the beginning of the
conditions.

Plan: Format

{+|-}{!|?}event [source(type)]:
action 1;
action 2;
action n.



context ←

the beginning of the
conditions.

It is optional if there are
no condintions.

Plan: Format

{+|-}{!|?}event [source(type)]: context ←
 action 1;
 action 2;
 action n.

The activation
conditions.

Plan: Format

{+|-}{!|?}event [source(type)]:

context ←

action 1;

action 2;

action n.



The activation
conditions.

It is not mandatory.

Plan: Format

{+|-}{!|?}event [source(type)]: context ←
action 1;
action 2;
action n.



the beginning of the body.

Plan: Format

{+|-}{!|?}event [source(type)]: context ←

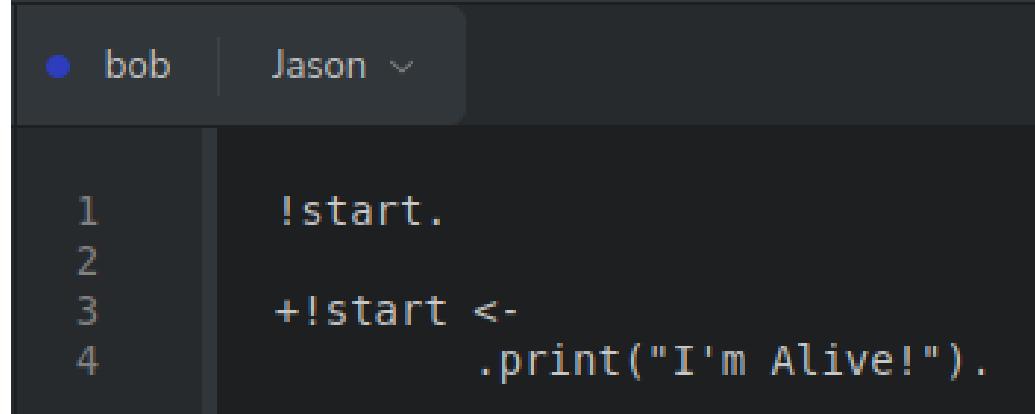
action 1;
action 2;
action n.

the actions of the body.
Separated by semi-colon.

Plan: Format

{+|-}{!|?}event [source(type)]: context ←
action 1;
action 2;
action n. the end of the plan.

Plan: Hello World



The screenshot shows a terminal window with a dark theme. At the top, there are two tabs: 'bob' (selected) and 'Jason'. The main area displays the following code:

```
1 !start.
2
3 +!start <-
4     .print("I'm Alive!").
```

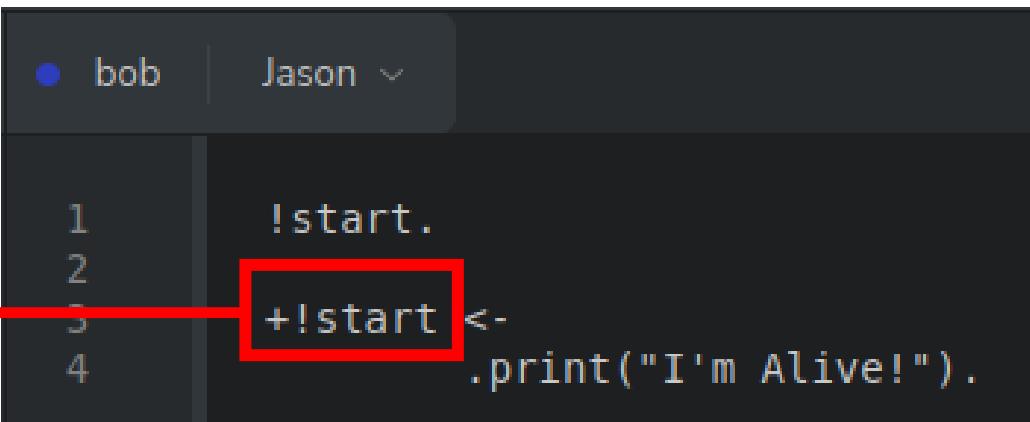
Plan: Hello World

An initial goal named
start.

```
bob Jason ~
1 !start.
2 +!start <-
3   .print("I'm Alive!").
4
```

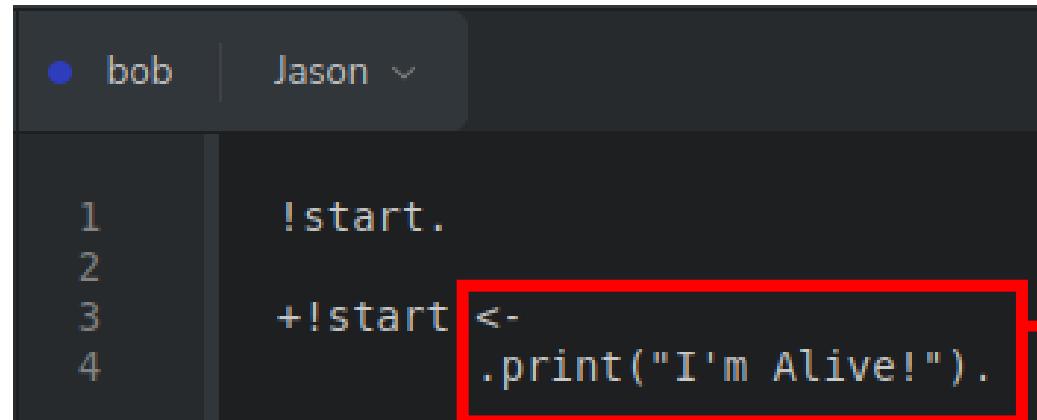
Plan: Hello World

An achievement plan
without triggering
event named start.



```
bob Jason ~
1 !start.
2 +!start <-
3 .print("I'm Alive!").
4
```

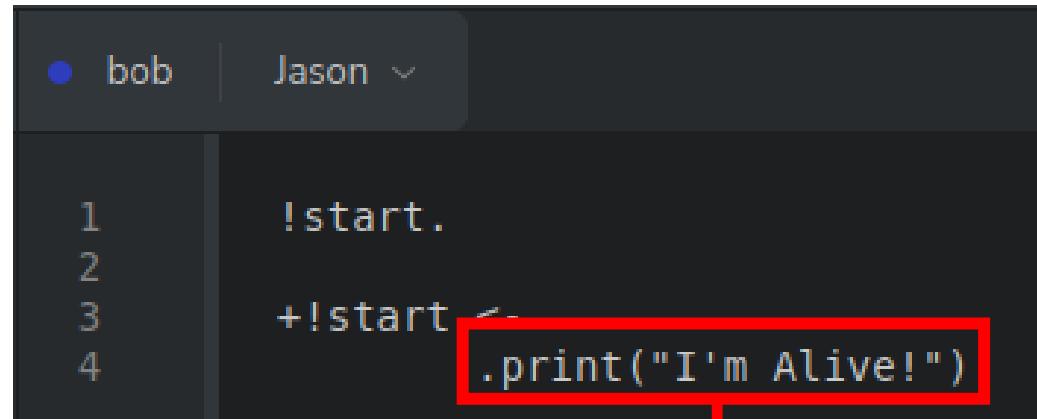
Plan: Hello World



```
bob      Jason ~
1       !start.
2
3       +!start <-
4       .print("I'm Alive!")
```

The body of the plan.

Plan: Hello World

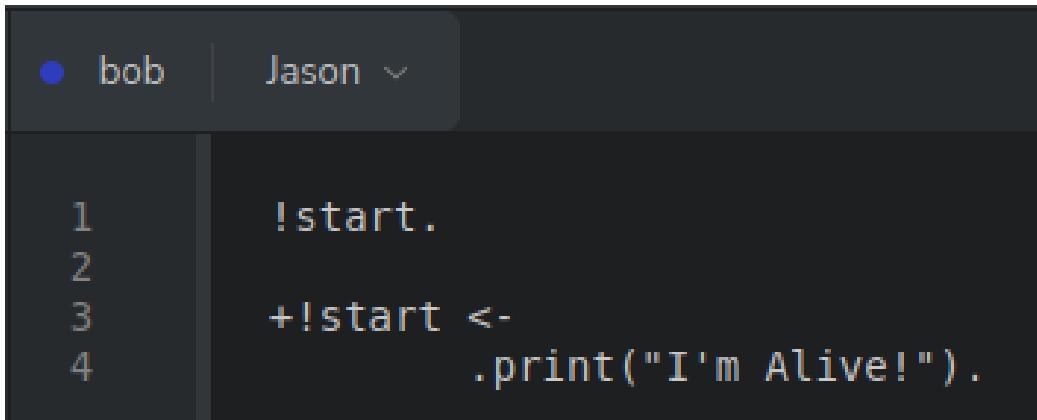


```
bob Jason >
1 !start.
2
3 +!start<
4 .print("I'm Alive!")
```



An action that prints in the agent tracer
(console).

Plan: Hello World



The screenshot shows a terminal window with a dark background. At the top, there is a header bar with a blue dot icon, the name "bob" in white, and a dropdown menu with "Jason" and a downward arrow. The main area of the terminal contains four numbered lines of code:

```
1 !start.  
2  
3 +!start <-  
4 .print("I'm Alive!").
```

Plan: Hello World

```
● bob | Jason ~  
1 !start.  
2  
3 +!start <-  
4 .print("I'm Alive!").
```



```
[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  
[bob] I'm Alive!
```

Plan: Types

1. Achievement Goal

- São ativados quando um plano é transformado de um desejo para uma intenção na mente do agente.

Plan: Types

1. Achievement Goal

- São ativados quando um plano é transformado de um desejo para uma intenção na mente do agente.

2. Test Goal

- São objetivos que recuperam informações da base de crenças.

Plan: Types

1. Achievement Goal

- São ativados quando um plano é transformado de um desejo para uma intenção na mente do agente.

2. Test Goal

- São objetivos que recuperam informações da base de crenças.

3. Belief Goal

- São planos ativados quando o agente adiciona ou remove uma crença da sua base de crenças.

Plans: Achievement Goal

{+|-}!event[source(type)]:



action 1;

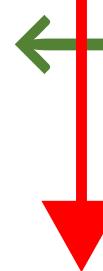
action 2;

action n.

context

Plans: Achievement Goal

{+ | -}!event[source(type)]:



defines if it is an
addition (+) or a
deletion (-) plan.

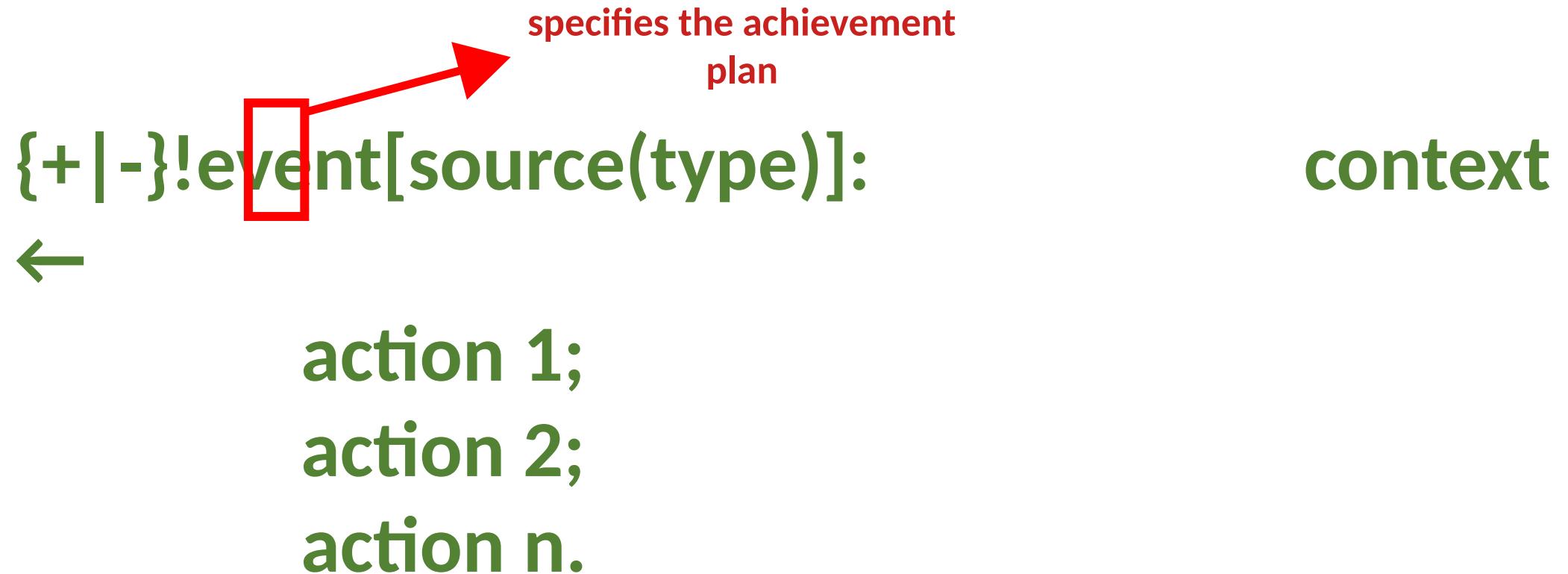
action 1;

action 2;

action n.

context

Plans: Achievement Goal

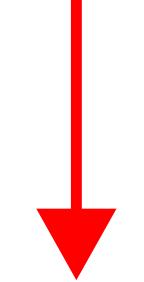


Plans: Addition Achievement Goal

```
+!event[source(type)]:  
    context ←  
        action 1;  
        action 2;  
        action n.
```

Plans: Addition Achievement Goal

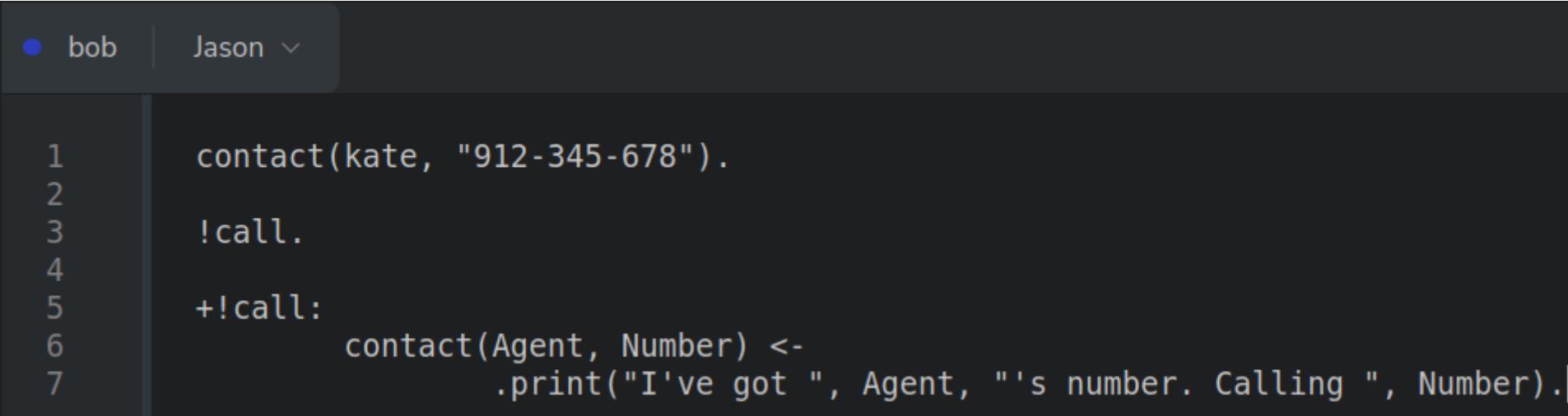
[+!]



defines an addition
plan.

event[source(type)]:
context ←
action 1;
action 2;
action n.

Addition Achievement Goal: Predicate



The screenshot shows a terminal window with a dark theme. At the top, it displays two agent names: 'bob' and 'Jason'. The main area contains the following Prolog code:

```
1 contact(kate, "912-345-678").  
2 !call.  
3  
4 +!call:  
5     contact(Agent, Number) <-  
6         .print("I've got ", Agent, "'s number. Calling ", Number).  
7
```

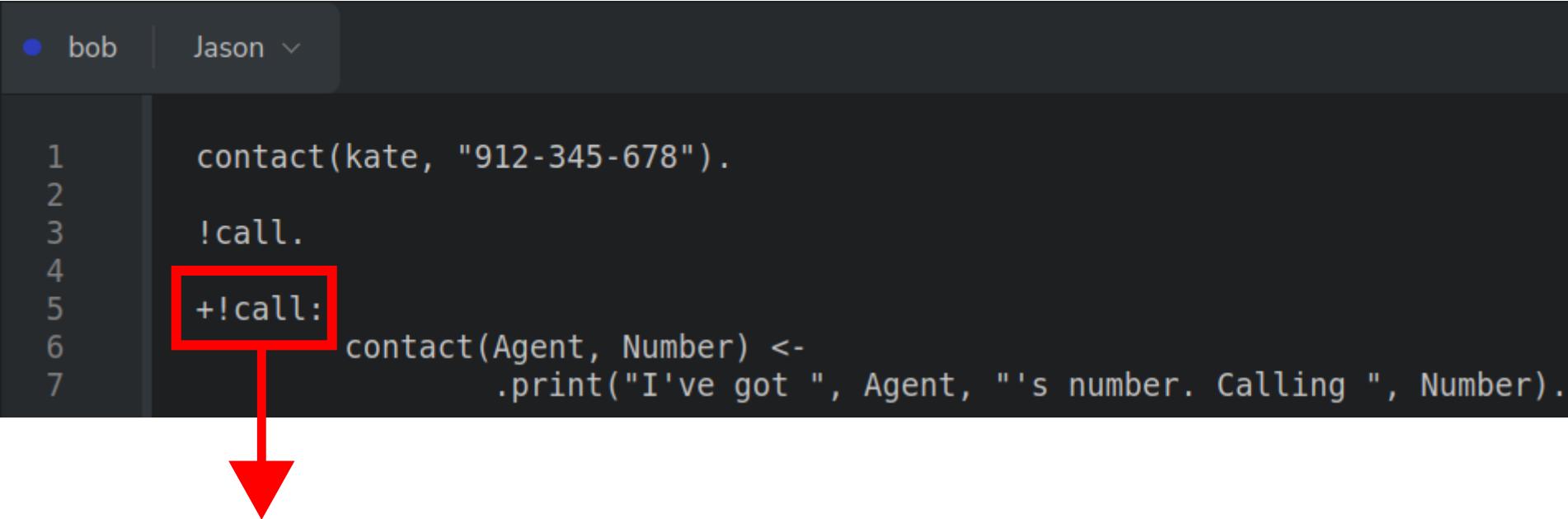
Addition Achievement Goal: Predicate

A predicate as a
goal...



```
bob | Jason ▾  
1 contact(kate, "912-345-678").  
2 !call.  
3  
4 +!call:  
5     contact(Agent, Number) <-  
6         .print("I've got ", Agent, "'s number. Calling ", Number).  
7
```

Addition Achievement Goal: Predicate



```
bob | Jason <v>
1 contact(kate, "912-345-678").
2
3 !call.
4
5 +!call:
6 contact(Agent, Number) <-
7     .print("I've got ", Agent, "'s number. Calling ", Number).
```

... which activates a plan with the same name...

Addition Achievement Goal: Predicate

```
• bob | Jason ▾  
1 contact(kate, "912-345-678").  
2 !call.  
3  
4  
5 +!call:  
6     contact(Agent, Number) <-  
7         .PRINT( I've got ", Agent, "'s number. Calling ", Number).  
8
```



... if the context is satisfied.

Addition Achievement Goal: One Plan

```
● bob | Jason ▾

1 !discount(phone,1200,10).
2
3 +!discount(Item,Value,Discount)
4      : Value > 100 <-
5          FinalValue = Value-(Value*Discount)/100;
6          .print("The final value is ", FinalValue, ".") .
```

Addition Achievement Goal: One Plan

An initial goal with predicate and values.



```
1 !discount(phone,1200,10).
2
3 +!discount(Item,Value,Discount)
4   : Value > 100 <-
5     FinalValue = Value-(Value*Discount)/100;
6     .print("The final value is ", FinalValue, ".") .
```

Addition Achievement Goal: One Plan

```
● bob | Jason ▾  
1      !discount(phone,1200,10).  
2  
3      +!discount(Item,Value,Discount)  
4          : value > 100 <-  
5              FinalValue = Value-(Value*Discount)/100;  
6              .print("The final value is ", FinalValue, ".") .
```



It activates this addition
achievement goal...

Addition Achievement Goal: One Plan

```
● bob | Jason ▾  
1 !discount(phone,1200,10).  
2  
3 +!discount(Item,Value,Discount)  
4     : Value > 100 <-  
5         finalValue = Value-(Value*Discount)/100;  
6         .print("The final value is ", FinalValue, ".") .
```



... if the context is satisfied.

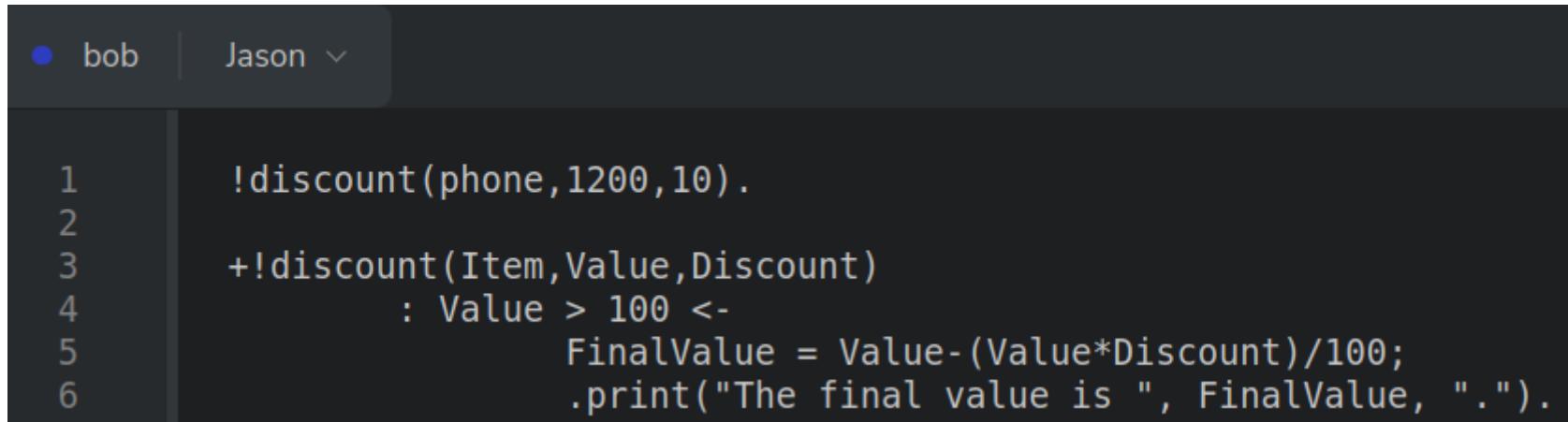
Addition Achievement Goal: One Plan

```
● bob | Jason ▾  
1 !discount(phone,1200,10).  
2  
3 +!discount(Item,Value,Discount)  
4      : Value > 100 <  
5          FinalValue = Value-(Value*Discount)/100;  
6          .print("The final value is ", FinalValue, ".") .
```



Then, the actions run.

Addition Achievement Goal: One Plan



The screenshot shows a terminal window with a dark background. At the top left, there is a user icon and the text "bob". To its right is a dropdown menu with "Jason" and a downward arrow. The main area of the terminal contains the following code:

```
1 !discount(phone,1200,10).
2
3 +!discount(Item,Value,Discount)
4   : Value > 100 <-
5     FinalValue = Value - (Value*Discount)/100;
6     .print("The final value is ", FinalValue, ".").
```

Addition Achievement Goal: One Plan

```
● bob | Jason ▾  
1 !discount(phone,1200,10).  
2  
3 +!discount(Item,Value,Discount)  
4 : Value > 100 <-  
5 FinalValue = Value-(Value*Discount)/100;  
6 .print("The final value is ", FinalValue, ".").
```



```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.  
Jason Http Server running on http://127.0.1.1:3272  
[bob] The final value is 1080.
```

Addition Achievement Goal: More Than One Plan

```
● bob | Jason ▾

1      !discount(phone,380).

2

3      +!discount(Item,Value)
4          : Value > 1000 <-
5              FinalValue = Value - (Value*10)/100;
6              .print("The final value of ", Item , " is ", FinalValue, ".").

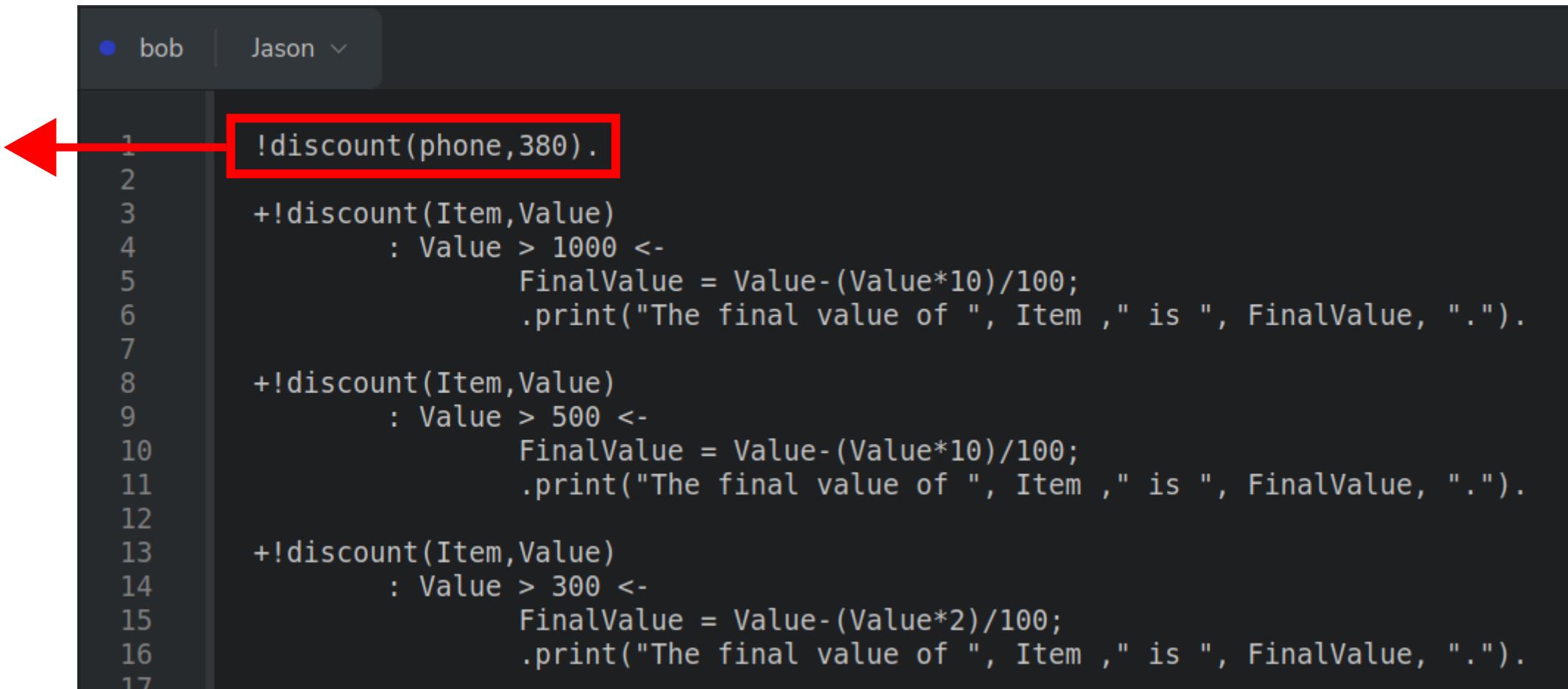
7      +!discount(Item,Value)
8          : Value > 500 <-
9              FinalValue = Value - (Value*10)/100;
10             .print("The final value of ", Item , " is ", FinalValue, ".").

11            +!discount(Item,Value)
12                : Value > 300 <-
13                    FinalValue = Value - (Value*2)/100;
14                    .print("The final value of ", Item , " is ", FinalValue, ".").

15
```

Addition Achievement Goal: More Than One Plan

The goal pursued.



```
bob | Jason ▾
1 !discount(phone,380).
2
3 +!discount(Item,Value)
4   : Value > 1000 <-
5     FinalValue = Value - (Value*10)/100;
6     .print("The final value of ", Item , " is ", FinalValue, ".").
7
8 +!discount(Item,Value)
9   : Value > 500 <-
10    FinalValue = Value - (Value*10)/100;
11    .print("The final value of ", Item , " is ", FinalValue, ".").
12
13 +!discount(Item,Value)
14   : Value > 300 <-
15     FinalValue = Value - (Value*2)/100;
16     .print("The final value of ", Item , " is ", FinalValue, ".").
17
```

Addition Achievement Goal: More Than One Plan

The possible
plans.

```
● bob | Jason ▾

1      !discount(phone,380).
2
3      +!discount(Item,Value)
4          : value > 1000 <-
5              FinalValue = Value - (Value*10)/100;
6              .print("The final value of ", Item , " is ", FinalValue, ".")
7
8      +!discount(Item,Value)
9          : value > 500 <-
10         FinalValue = Value - (Value*10)/100;
11         .print("The final value of ", Item , " is ", FinalValue, ".")
12
13     +!discount(Item,Value)
14         : value > 300 <-
15         FinalValue = Value - (Value*2)/100;
16         .print("The final value of ", Item , " is ", FinalValue, ".")
17
```

Addition Achievement Goal: More Than One Plan

```
● bob | Jason ▾

1 !discount(phone,380).
2
3 +!discount(Item,Value)
4   : Value > 1000 <-
5     FinalValue = Value - (Value*10)/100;
6     .print("The final value of ", Item , " is ", FinalValue, ".").
7
8 +!discount(Item,Value)
9   : Value > 500 <-
10    FinalValue = Value - (Value*10)/100;
11    .print("The final value of ", Item , " is ", FinalValue, ".").
12
13 +!discount(Item,Value)
14   : Value > 300 <-
15     FinalValue = Value - (Value*2)/100;
16     .print("The final value of ", Item , " is ", FinalValue, ".").
17
```

Addition Achievement Goal: More Than One Plan

The activated plan.

```
● bob | Jason ▾

1  !discount(phone, 380).
2
3  +!discount(Item,Value)
4    : Value > 1000 <-
5      FinalValue = Value - (Value*10)/100;
6      .print("The final value of ", Item , " is ", FinalValue, ".")
7
8  +!discount(Item,Value)
9    : Value > 500 <-
10   FinalValue = Value - (Value*10)/100;
11   .print("The final value of ", Item , " is ", FinalValue, ".")
12
13  +!discount(Item,Value)
14    : Value > 300 <-
15      FinalValue = Value - (Value*2)/100;
16      .print("The final value of ", Item , " is ", FinalValue, ".")
17
```

Addition Achievement Goal: More Than One Plan

bob Jason ▾

```
1      !discount(phone,380).
2
3      +!discount(Item,Value)
4          : Value > 1000 <-
5              FinalValue = Value-(Value*10)/100;
6              .print("The final value of ", Item , " is ", FinalValue, ".").
7
8      +!discount(Item,Value)
9          : Value > 500 <-
10             FinalValue = Value-(Value*10)/100;
11             .print("The final value of ", Item , " is ", FinalValue, ".").
12
13     +!discount(Item,Value)
14         : Value > 300 <-
15             FinalValue = Value-(Value*2)/100;
16             .print("The final value of ", Item , " is ", FinalValue, ".").
17
```

Addition Achievement Goal: More Than One Plan

bob Jason ▾

```
1      !discount(phone,380).
2
3      +!discount(Item,Value)
4          : Value > 1000 <-
5              FinalValue = Value-(Value*10)/100;
6              .print("The final value of ", Item , " is ", FinalValue, ".").
7
8      +!discount(Item,Value)
9          : Value > 500 <-
10             FinalValue = Value-(Value*10)/100;
11             .print("The final value of ", Item , " is ", FinalValue, ".").
12
13     +!discount(Item,Value)
14         : Value > 300 <-
15             FinalValue = Value-(Value*2)/100;
16             .print("The final value of ", Item , " is ", FinalValue, ".").
17
```

[ChonOS EmbeddedMAS] Starting the Multi-Agent System.
Jason Http Server running on http://127.0.1.1:3272
[bob] The final value of phone is 372.4.

Addition Achievement Goal: More Than One Plan

When there are more plans
with the **same event** name,
the agent will try them
one by one...

Addition Achievement Goal: More Than One Plan

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

.

.

.

+!event[source(type)]: context $\leftarrow \dots$.

Addition Achievement Goal: More Than One Plan

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

.

.

.

+!event[source(type)]: context $\leftarrow \dots$.

Addition Achievement Goal: More Than One Plan

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

.

.

.

+!event[source(type)]: context $\leftarrow \dots$.

Addition Achievement Goal: More Than One Plan

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

.

.

.

+!event[source(type)]: context $\leftarrow \dots$.

Addition Achievement Goal: More Than One Plan

... until it finds one that
satisfies the context.

Addition Achievement Goal: More Than One Plan

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

.

.

.

+!event[source(type)]: context $\leftarrow \dots$.

Addition Achievement Goal: More Than One Plan

When there are two or more
plans that could be
activated at the **same**
time . . .

Addition Achievement Goal: More Than One Plan

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

.

.

.

+!event[source(type)]: context $\leftarrow \dots$.

Addition Achievement Goal: More Than One Plan

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

.
. .

+!event[source(type)]: context $\leftarrow \dots$.

Addition Achievement Goal: More Than One Plan

... it chooses the first one
that **fits the context**.

Addition Achievement Goal: More Than One Plan

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

.

.

.

+!event[source(type)]: context $\leftarrow \dots$.

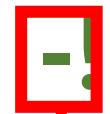
Addition Achievement Goal: More Than One Plan

However, when no one is activated, the agent finds a goal for which **no relevant plan** is available.

Plans: Deletion Achievement Goal

```
-!event[source(type)]:  
    context ←  
        action 1;  
        action 2;  
        action n.
```

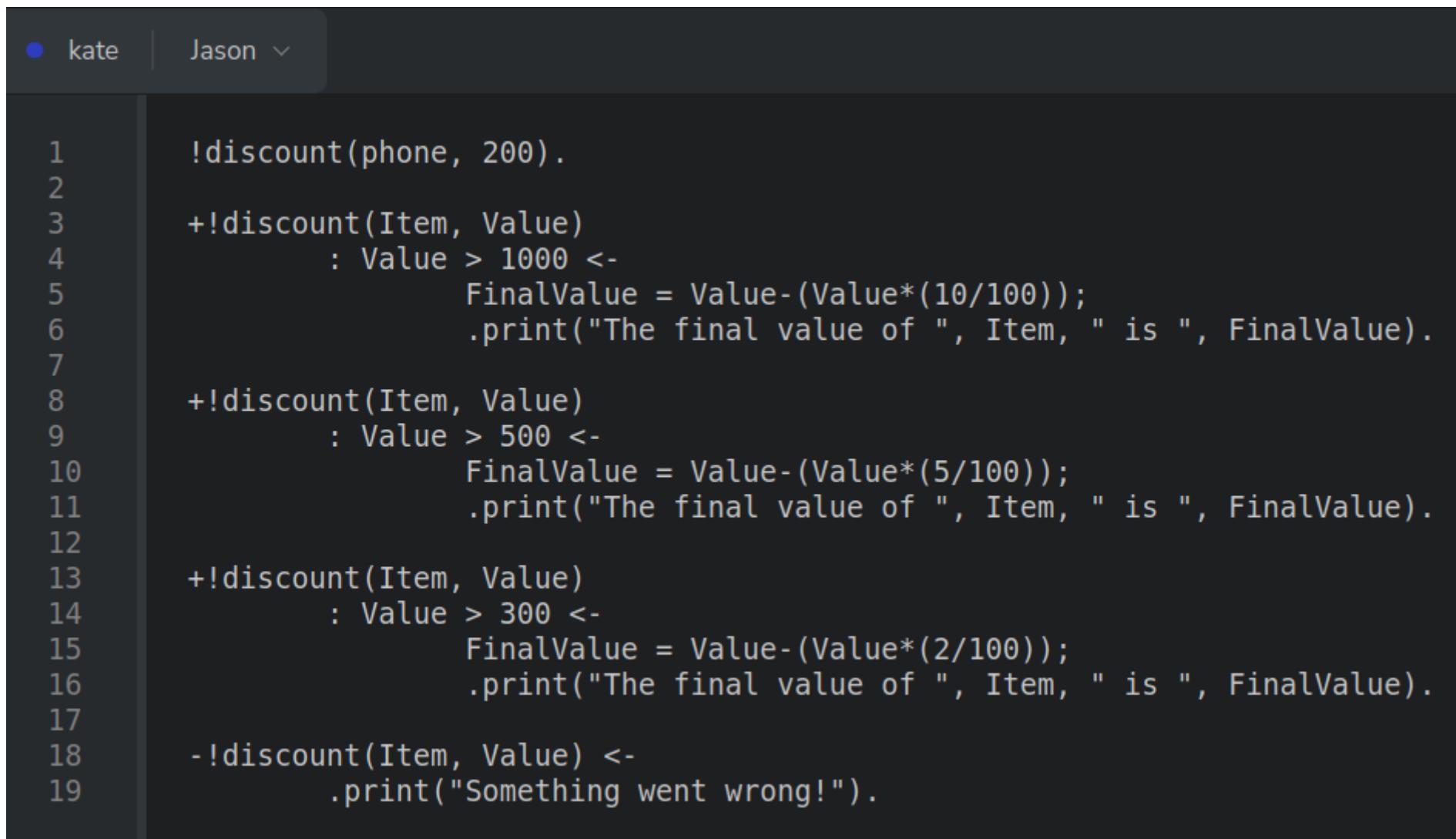
Plans: Deletion Achievement Goal



defines a deletion plan.

!event[source(type)]:
context ←
action 1;
action 2;
action n.

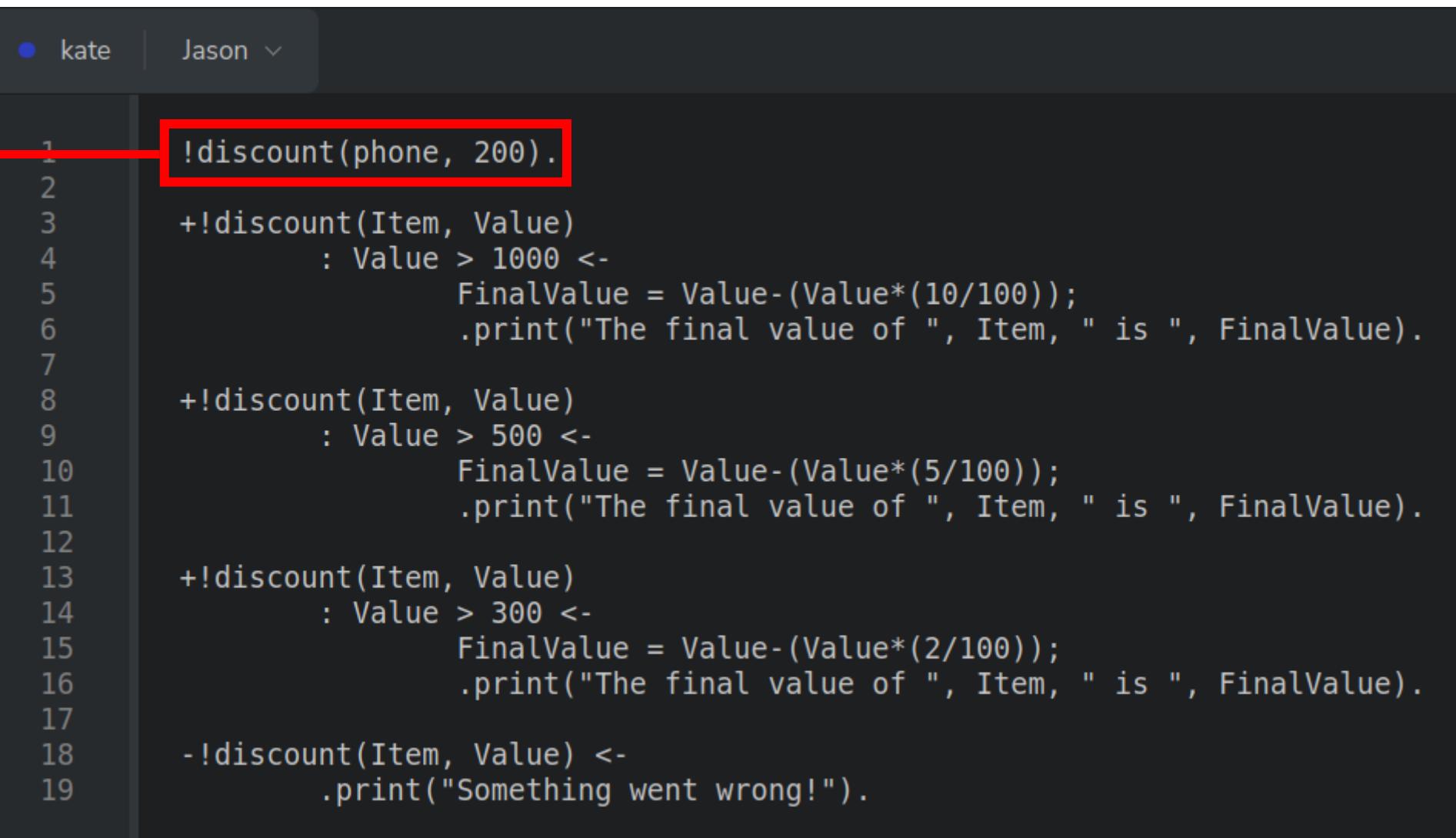
Plans: Deletion Achievement Goal



```
1      !discount(phone, 200).
2
3      +!discount(Item, Value)
4          : Value > 1000 <-
5              FinalValue = Value - (Value*(10/100));
6              .print("The final value of ", Item, " is ", FinalValue).
7
8      +!discount(Item, Value)
9          : Value > 500 <-
10             FinalValue = Value - (Value*(5/100));
11             .print("The final value of ", Item, " is ", FinalValue).
12
13     +!discount(Item, Value)
14         : Value > 300 <-
15             FinalValue = Value - (Value*(2/100));
16             .print("The final value of ", Item, " is ", FinalValue).
17
18     - !discount(Item, Value) <-
19         .print("Something went wrong!") .
```

Plans: Deletion Achievement Goal

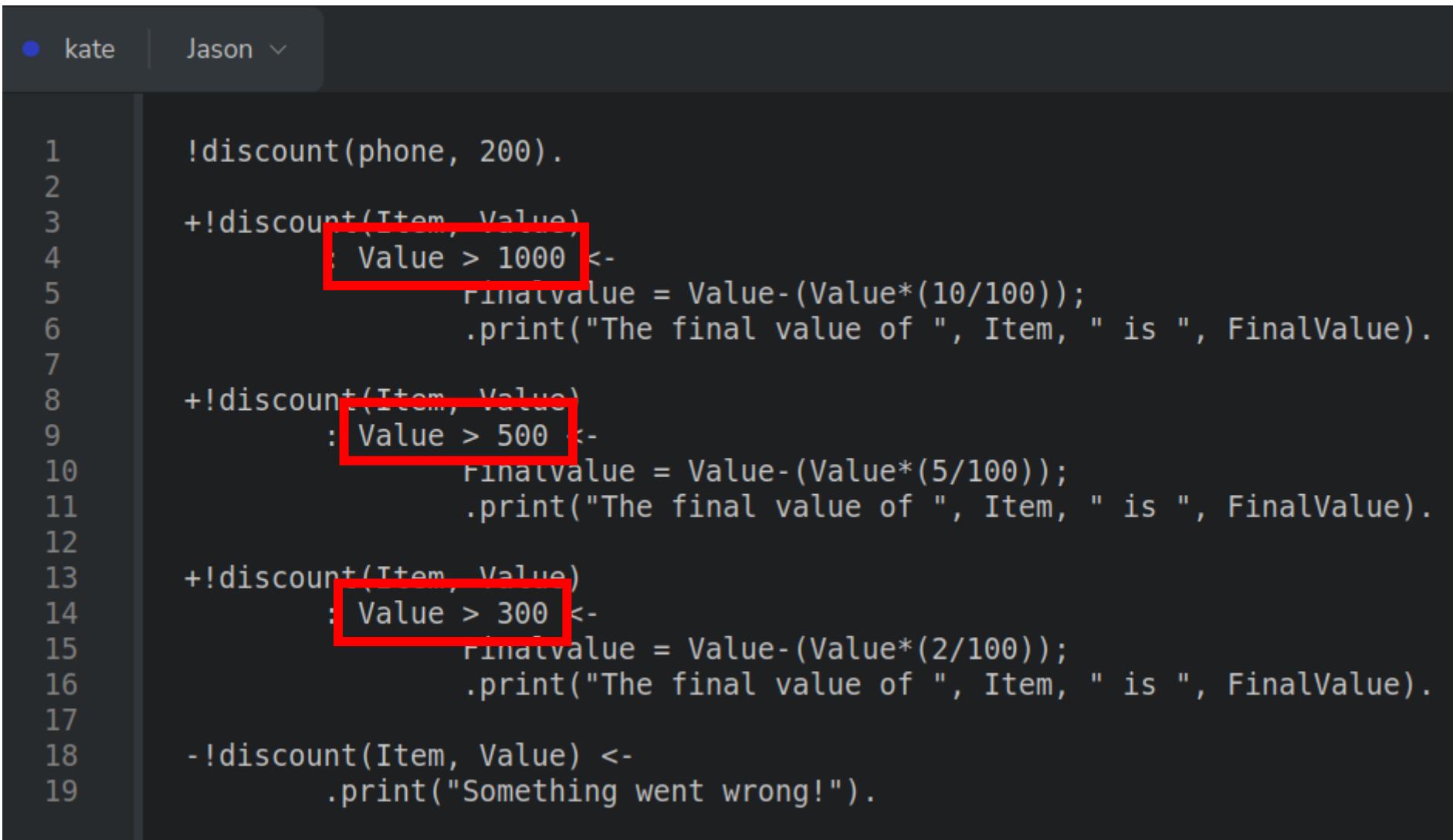
By using these
values...



```
● kate | Jason ▾
1 !discount(phone, 200).
2
3 +!discount(Item, Value)
4   : Value > 1000 <-
5     FinalValue = Value-(Value*(10/100));
6     .print("The final value of ", Item, " is ", FinalValue).
7
8 +!discount(Item, Value)
9   : Value > 500 <-
10    FinalValue = Value-(Value*(5/100));
11    .print("The final value of ", Item, " is ", FinalValue).
12
13 +!discount(Item, Value)
14   : Value > 300 <-
15    FinalValue = Value-(Value*(2/100));
16    .print("The final value of ", Item, " is ", FinalValue).
17
18 - !discount(Item, Value) <-
19   .print("Something went wrong!") .
```

Plans: Deletion Achievement Goal

... none of the available plans activate.



```
● kate | Jason ▾

1   !discount(phone, 200).

2
3   +!discount(Item, Value)
4     : Value > 1000 <-
5       FinalValue = Value - (Value*(10/100));
6       .print("The final value of ", Item, " is ", FinalValue).

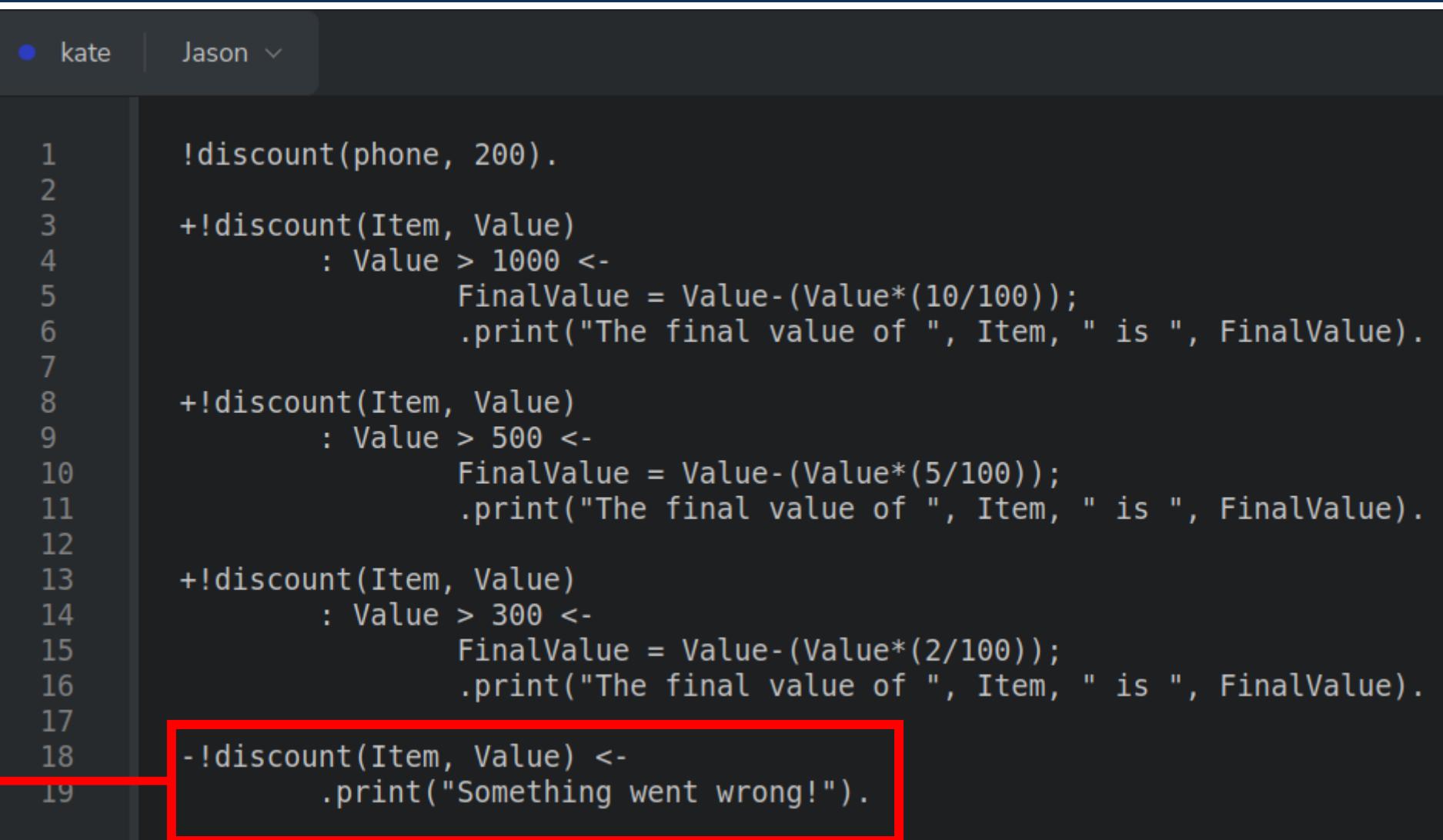
7
8   +!discount(Item, Value)
9     : Value > 500 <-
10    FinalValue = Value - (Value*(5/100));
11    .print("The final value of ", Item, " is ", FinalValue).

12
13  +!discount(Item, Value)
14    : Value > 300 <-
15    FinalValue = Value - (Value*(2/100));
16    .print("The final value of ", Item, " is ", FinalValue).

17
18  - !discount(Item, Value) <-
19    .print("Something went wrong!") .
```

Plans: Deletion Achievement Goal

Then, a contingency plan activates.



```
1      !discount(phone, 200).
2
3      +!discount(Item, Value)
4          : Value > 1000 <-
5              FinalValue = Value - (Value*(10/100));
6              .print("The final value of ", Item, " is ", FinalValue).
7
8      +!discount(Item, Value)
9          : Value > 500 <-
10             FinalValue = Value - (Value*(5/100));
11             .print("The final value of ", Item, " is ", FinalValue).
12
13     +!discount(Item, Value)
14         : Value > 300 <-
15             FinalValue = Value - (Value*(2/100));
16             .print("The final value of ", Item, " is ", FinalValue).
17
18     - !discount(Item, Value) <-
19         .print("Something went wrong!") .
```

Plans: Deletion Achievement Goal

● kate | Jason ▾

```
1   !discount(phone, 200).  
2  
3   +!discount(Item, Value)  
4     : Value > 1000 <-  
5       FinalValue = Value-(Value*(10/100));  
6       .print("The final value of ", Item, " is ", FinalValue).  
7  
8   +!discount(Item, Value)  
9     : Value > 500 <-  
10    FinalValue = Value-(Value*(5/100));  
11    .print("The final value of ", Item, " is ", FinalValue).  
12  
13  +!discount(Item, Value)  
14    : Value > 300 <-  
15    FinalValue = Value-(Value*(2/100));  
16    .print("The final value of ", Item, " is ", FinalValue).  
17  
18  -!discount(Item, Value) <-  
19    .print("Something went wrong!").
```

Plans: Deletion Achievement Goal

● kate | Jason ▾

```
1   !discount(phone, 200).  
2  
3   +!discount(Item, Value)  
4     : Value > 1000 <-  
5       FinalValue = Value-(Value*(10/100));  
6       .print("The final value of ", Item, " is ", FinalValue).  
7  
8   +!discount(Item, Value)  
9     : Value > 500 <-  
10    FinalValue = Value-(Value*(5/100));  
11    .print("The final value of ", Item, " is ", FinalValue).  
12  
13  +!discount(Item, Value)  
14    : Value > 300 <-  
15    FinalValue = Value-(Value*(2/100));  
16    .print("The final value of ", Item, " is ", FinalValue).  
17  
18  -!discount(Item, Value) <-  
19    .print("Something went wrong!").
```



```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.  
Jason Http Server running on http://127.0.1.1:3272  
[kate] Something went wrong!
```

Plans: Deletion Achievement Goal

In this case, another
addition achievement
plan can cover the
missing context.

Plans: Deletion Achievement Goal

```
● kate | Jason ▾

1   !discount(phone, 200).

2

3   +!discount(Item, Value)
4       : Value > 1000 <-
5           FinalValue = Value-(Value*(10/100));
6           .print("The final value of ", Item, " is ", FinalValue).

7

8   +!discount(Item, Value)
9       : Value > 500 <-
10          FinalValue = Value-(Value*(5/100));
11          .print("The final value of ", Item, " is ", FinalValue).

12

13  +!discount(Item, Value)
14      : Value > 300 <-
15          FinalValue = Value-(Value*(2/100));
16          .print("The final value of ", Item, " is ", FinalValue).

17

18  +!discount(Item, Value)
19      : Value <= 300 <-
20          .print("There is no discount on ", Item, ".").

21

22  -!discount(Item, Value) <-
23      .print("Something went wrong!").
```

Plans: Deletion Achievement Goal

So, this plan activates.



```
● kate | Jason ▾

1   !discount(phone, 200).

2

3   +!discount(Item, Value)
4       : Value > 1000 <-
5           FinalValue = Value-(Value*(10/100));
6           .print("The final value of ", Item, " is ", FinalValue).

7

8   +!discount(Item, Value)
9       : Value > 500 <-
10          FinalValue = Value-(Value*(5/100));
11          .print("The final value of ", Item, " is ", FinalValue).

12

13  +!discount(Item, Value)
14      : Value > 300 <-
15          FinalValue = Value-(Value*(2/100));
16          .print("The final value of ", Item, " is ", FinalValue).

17

18  +!discount(Item, Value)
19      Value <= 300 <-
20          .print("There is no discount on ", Item, ".").

21

22  -!discount(Item, Value) <-
23      .print("Something went wrong!").
```

Plans: Deletion Achievement Goal

```
● kate | Jason ▾  
1   !discount(phone, 200).  
2  
3   +!discount(Item, Value)  
4       : Value > 1000 <-  
5           FinalValue = Value-(Value*(10/100));  
6           .print("The final value of ", Item, " is ", FinalValue).  
7  
8   +!discount(Item, Value)  
9       : Value > 500 <-  
10      FinalValue = Value-(Value*(5/100));  
11      .print("The final value of ", Item, " is ", FinalValue).  
12  
13  +!discount(Item, Value)  
14      : Value > 300 <-  
15          FinalValue = Value-(Value*(2/100));  
16          .print("The final value of ", Item, " is ", FinalValue).  
17  
18  +!discount(Item, Value)  
19      : Value <= 300 <-  
20          .print("There is no discount on ", Item, ".").  
21  
22  !discount(Item, Value) <-  
23      .print("Something went wrong!").
```

It only activates if
the activated plan
fails.



Plans: Deletion Achievement Goal

kate | Jason ▾

```
1 !discount(phone,80).  
2  
3 +!discount(Item, Value)  
4     : Value > 1000 <-  
5         FinalValue = Value-(Value*(10/100));  
6         .print("The final value of ", Item, " is ", FinalValue).  
7  
8 +!discount(Item, Value)  
9     : Value > 500 <-  
10        FinalValue = Value-(Value*(5/100));  
11        .print("The final value of ", Item, " is ", FinalValue).  
12  
13 +!discount(Item, Value)  
14     : Value > 300 <-  
15        FinalValue = Value-(Value*(2/100));  
16        .print("The final value of ", Item, " is ", FinalValue).  
17  
18 +!discount(Item, Value)  
19     : Value <= 300 <-  
20         .print("There is no discount on ", Item, ".");  
21         !generateInvoice.  
22  
23 -!discount(Item, Value) <-  
24     .print("Something went wrong!").
```

Plans: Deletion Achievement Goal

kate | Jason ▾

```
1 !discount(phone,80).  
2  
3 +!discount(Item, Value)  
4     : Value > 1000 <-  
5         FinalValue = Value-(Value*(10/100));  
6         .print("The final value of ", Item, " is ", FinalValue).  
7  
8 +!discount(Item, Value)  
9     : Value > 500 <-  
10        FinalValue = Value-(Value*(5/100));  
11        .print("The final value of ", Item, " is ", FinalValue).  
12  
13 +!discount(Item, Value)  
14     : Value > 300 <-  
15        FinalValue = Value-(Value*(2/100));  
16        .print("The final value of ", Item, " is ", FinalValue).  
17  
18 +!discount(Item, Value)  
19     : Value <= 300 <-  
20         .print("There is no discount on ", Item, ".");  
21         !generateInvoice.  
22  
23 -!discount(Item, Value) <-  
24     .print("Something went wrong!").
```

The agent tries to commit with generateInvoice, but it does not have a relevant plan.

Plans: Deletion Achievement Goal

kate | Jason ▾

```
1 !discount(phone,80).  
2  
3 +!discount(Item, Value)  
4     : Value > 1000 <-  
5         FinalValue = Value-(Value*(10/100));  
6         .print("The final value of ", Item, " is ", FinalValue).  
7  
8 +!discount(Item, Value)  
9     : Value > 500 <-  
10        FinalValue = Value-(Value*(5/100));  
11        .print("The final value of ", Item, " is ", FinalValue).  
12  
13 +!discount(Item, Value)  
14     : Value > 300 <-  
15        FinalValue = Value-(Value*(2/100));  
16        .print("The final value of ", Item, " is ", FinalValue).  
17  
18 +!discount(Item, Value)  
19     : Value <= 300 <-  
20         .print("There is no discount on ", Item, ".");  
21         !generateInvoice.  
22  
23 -!discount(Item, Value) <-  
24     .print("Something went wrong!").
```

As the previous plan fails, the contingency plan is activated.

Plans: Deletion Achievement Goal

Deletion plans can also
have a **context**.

Plans: Deletion Achievement Goal

```
● kate | Jason ▾

1      stock(beer, 20).
2
3      !purchase(chocolate, 50).
4
5      +!purchase(Item, Amount):
6          stock(Item, Stock) &
7          Stock >= Amount <-
8              .print("Your product ", Item, " is available. We have ", Stock, " units.").
9
10     - !purchase(Item, Amount):
11         stock(Item, Stock) &
12         Stock < Amount <-
13             .print("Sorry, your product ", Item, " is unavailable. We have only ", Stock, " units.").
14
15     - !purchase(Item, _):
16         .findall(stock(Item,Value), stock(Item,Value), Result) &
17         Result == [] <-
18             .print("Sorry, we don't sell ", Item, ".").
19
20     - !purchase(_, _) <-
21         .print("Something went wrong!").
```

Plans: Deletion Achievement Goal

Deletion
plans with
context...

```
● kate Jason ▾

1   stock(beer, 20).
2
3   !purchase(chocolate, 50).
4
5   +!purchase(Item, Amount):
6       stock(Item, Stock) &
7       Stock >= Amount <-
8           .print("Your product ", Item, " is available. We have ", Stock, " units.").
9
10  - !purchase(Item, Amount):
11      stock(Item, Stock) &
12      Stock < Amount <-
13          .print("Sorry, your product ", Item, " is unavailable. We have only ", Stock, " units.").
14
15  - !purchase(Item, _):
16      .findall(stock(Item,Value), stock(Item,Value), Result) &
17      Result == [] <-
18          .print("Sorry, we don't sell ", Item, ".").
19
20  - !purchase(_,_) <-
21      .print("Something went wrong!").
```

Plans: Deletion Achievement Goal

... and one
without context.

```
● kate | Jason ▾

1      stock(beer, 20).
2
3      !purchase(chocolate, 50).
4
5      +!purchase(Item, Amount):
6          stock(Item, Stock) &
7          Stock >= Amount <-
8              .print("Your product ", Item, " is available. We have ", Stock, " units.").
9
10     - !purchase(Item, Amount):
11         stock(Item, Stock) &
12         Stock < Amount <-
13             .print("Sorry, your product ", Item, " is unavailable. We have only ", Stock, " units.").
14
15     - !purchase(Item, _):
16         .findall(stock(Item,Value), stock(Item,Value), Result) &
17         Result == [] <-
18             .print("Sorry, we don't sell ", Item, ".").
19
20     - !purchase(_,_) <-
21         .print("Something went wrong!")
```

Plans: Deletion Achievement Goal

+!event[source(type)]: context $\leftarrow \dots$.

Plans: Deletion Achievement Goal

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

Plans: Deletion Achievement Goal

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

:

:

:

+!event[source(type)]: context $\leftarrow \dots$.

Plans: Deletion Achievement Goal

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

⋮
⋮
⋮

+!event[source(type)]: context $\leftarrow \dots$.

-!event[source(type)]: context $\leftarrow \dots$.

Plans: Deletion Achievement Goal

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

⋮
⋮
⋮

+!event[source(type)]: context $\leftarrow \dots$.

-!event[source(type)]: context $\leftarrow \dots$.

-!event[source(type)]: context $\leftarrow \dots$.

Plans: Deletion Achievement Goal

+!event[source(type)]: context $\leftarrow \dots$.

+!event[source(type)]: context $\leftarrow \dots$.

⋮
⋮
⋮

+!event[source(type)]: context $\leftarrow \dots$.

-!event[source(type)]: context $\leftarrow \dots$.

-!event[source(type)]: context $\leftarrow \dots$.

⋮
⋮
⋮

-!event[source(type)]: context $\leftarrow \dots$.

Plans: Test Goal

{+|-}?event[source(type)]: context



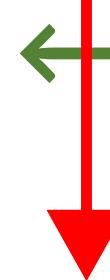
action 1;

action 2;

action n.

Plans: Test Goal

{+ | -}?



event[source(type)]:

context

action 1;

action 2;

action n.

defines if it is an addition (+) or a deletion (-) plan.

Plans: Test Goal

{+|-}?event[source(type)]: context
specifies the test goal plan.

←

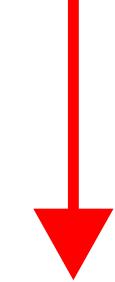
action 1;
action 2;
action n.

Plans: Addition Test Goal

```
+?event[source(type)]:  
    context ←  
        action 1;  
        action 2;  
        action n.
```

Plans: Addition Test Goal

[+?]



defines an addition
plan.

event[source(type)]:
context ←
action 1;
action 2;
action n.

Plans: Addition Test Goal

newAgent Jason ▾

```
1   agent(teddy).  
2  
3   !count.  
4  
5   +!count <-  
6       ?agent(Name);  
7       ?count(N);  
8       .print(N,". My name is ", Name,".");  
9       -+count(N+1);  
10      .wait(500);  
11      !count.  
12  
13      +?count(N) <-  
14          .print("I need to start counting...");  
15          N=1;  
16          +count(N+1).
```

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent S  
Jason Http Server running on http://127.0.1.1:3  
[newAgent] I need to start counting...  
[newAgent] 1. My name is teddy.  
[newAgent] 2. My name is teddy.  
[newAgent] 3. My name is teddy.  
[newAgent] 4. My name is teddy.  
[newAgent] 5. My name is teddy.  
[newAgent] 6. My name is teddy.  
[newAgent] 7. My name is teddy.  
[newAgent] 8. My name is teddy.  
[newAgent] 9. My name is teddy.  
[newAgent] 10. My name is teddy.  
[newAgent] 11. My name is teddy.  
[newAgent] 12. My name is teddy.  
[newAgent] 13. My name is teddy.  
[newAgent] 14. My name is teddy.  
[newAgent] 15. My name is teddy.  
[newAgent] 16. My name is teddy.  
[newAgent] 17. My name is teddy.  
[newAgent] 18. My name is teddy.  
[newAgent] 19. My name is teddy.  
[newAgent] 20. My name is teddy.
```

Plans: Addition Test Goal

newAgent Jason ▾

```
1   agent(teddy).  
2  
3   !count.  
4  
5   +!count <-  
6       ?agent(Name);  
7       ?count(N);  
8       .print(N,". My name is ", Name,".");  
9       -+count(N+1);  
10      .wait(500);  
11      !count.  
12  
13      +?count(N) <-  
14          .print("I need to start counting...");  
15          N=1;  
16          +count(N+1).
```

[ChonOS EmbeddedMAS] Starting the Multi-Agent S
Jason Http Server running on http://127.0.1.1:3
[newAgent] I need to start counting...
[newAgent] 1. My name is teddy.
[newAgent] 2. My name is teddy.
[newAgent] 3. My name is teddy.
[newAgent] 4. My name is teddy.
[newAgent] 5. My name is teddy.
[newAgent] 6. My name is teddy.
[newAgent] 7. My name is teddy.
[newAgent] 8. My name is teddy.
[newAgent] 9. My name is teddy.
[newAgent] 10. My name is teddy.
[newAgent] 11. My name is teddy.
[newAgent] 12. My name is teddy.
[newAgent] 13. My name is teddy.
[newAgent] 14. My name is teddy.
[newAgent] 15. My name is teddy.
[newAgent] 16. My name is teddy.
[newAgent] 17. My name is teddy.
[newAgent] 18. My name is teddy.
[newAgent] 19. My name is teddy.
[newAgent] 20. My name is teddy.

Plans: Addition Test Goal

newAgent Jason ▾

```
1 agent(teddy).  
2  
3 !count.  
4  
5 +!count <-  
6     ?agent(Name);  
7     ?count(N);  
8     .print(N,". My name is ", Name,".");  
9     -+count(N+1);  
10    .wait(500);  
11    !count.  
12  
13 +?count(N) <-  
14     .print("I need to start counting...");  
15     N=1;  
16     +count(N+1).
```

[ChonOS EmbeddedMAS] Starting the Multi-Agent S
Jason Http Server running on http://127.0.1.1:3
[newAgent] I need to start counting...
[newAgent] 1. My name is teddy.
[newAgent] 2. My name is teddy.
[newAgent] 3. My name is teddy.
[newAgent] 4. My name is teddy.
[newAgent] 5. My name is teddy.
[newAgent] 6. My name is teddy.
[newAgent] 7. My name is teddy.
[newAgent] 8. My name is teddy.
[newAgent] 9. My name is teddy.
[newAgent] 10. My name is teddy.
[newAgent] 11. My name is teddy.
[newAgent] 12. My name is teddy.
[newAgent] 13. My name is teddy.
[newAgent] 14. My name is teddy.
[newAgent] 15. My name is teddy.
[newAgent] 16. My name is teddy.
[newAgent] 17. My name is teddy.
[newAgent] 18. My name is teddy.
[newAgent] 19. My name is teddy.
[newAgent] 20. My name is teddy.

Plans: Addition Test Goal

newAgent Jason ▾

```
1   agent(teddy).  
2  
3   !count.  
4  
5   +!count <-  
6       ?agent(Name);  
7       ?count(N);  
8       .print(N,". My name is ", Name,".");  
9       -+count(N+1);  
10      .wait(500);  
11      !count.  
12  
13      +?count(N) <-  
14          .print("I need to start counting...");  
15          N=1;  
16          +count(N+1).
```

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent S  
Jason Http Server running on http://127.0.1.1:3  
[newAgent] I need to start counting...  
[newAgent] 1. My name is teddy.  
[newAgent] 2. My name is teddy.  
[newAgent] 3. My name is teddy.  
[newAgent] 4. My name is teddy.  
[newAgent] 5. My name is teddy.  
[newAgent] 6. My name is teddy.  
[newAgent] 7. My name is teddy.  
[newAgent] 8. My name is teddy.  
[newAgent] 9. My name is teddy.  
[newAgent] 10. My name is teddy.  
[newAgent] 11. My name is teddy.  
[newAgent] 12. My name is teddy.  
[newAgent] 13. My name is teddy.  
[newAgent] 14. My name is teddy.  
[newAgent] 15. My name is teddy.  
[newAgent] 16. My name is teddy.  
[newAgent] 17. My name is teddy.  
[newAgent] 18. My name is teddy.  
[newAgent] 19. My name is teddy.  
[newAgent] 20. My name is teddy.
```

Plans: Addition Test Goal

newAgent Jason ▾

```
1 agent(teddy).  
2  
3 !count.  
4  
5 +!count <-  
6     ?agent(Name);  
7     ?count(N);  
8     .print(N,". My name is ", Name,".");  
9     -+count(N+1);  
10    .wait(500);  
11    !count.  
12  
13 +?count(N) <-  
14     .print("I need to start counting...");  
15     N=1;  
16     +count(N+1).
```

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent S  
Jason Http Server running on http://127.0.1.1:3  
[newAgent] I need to start counting...  
[newAgent] 1. My name is teddy.  
[newAgent] 2. My name is teddy.  
[newAgent] 3. My name is teddy.  
[newAgent] 4. My name is teddy.  
[newAgent] 5. My name is teddy.  
[newAgent] 6. My name is teddy.  
[newAgent] 7. My name is teddy.  
[newAgent] 8. My name is teddy.  
[newAgent] 9. My name is teddy.  
[newAgent] 10. My name is teddy.  
[newAgent] 11. My name is teddy.  
[newAgent] 12. My name is teddy.  
[newAgent] 13. My name is teddy.  
[newAgent] 14. My name is teddy.  
[newAgent] 15. My name is teddy.  
[newAgent] 16. My name is teddy.  
[newAgent] 17. My name is teddy.  
[newAgent] 18. My name is teddy.  
[newAgent] 19. My name is teddy.  
[newAgent] 20. My name is teddy.
```

Plans: Deletion Test Goal

```
-?event[source(type)]:  
    context ←  
        action 1;  
        action 2;  
        action n.
```

Plans: Deletion Test Goal

-?event[source(type)]:

context ←

action 1;

action 2;

action n.

defines a deletion
plan.



Plans: Deletion Test Goal

```
● teddy | Jason ▾

1   agent(teddy).
2
3   !count.
4
5   +!count <-
6       ?agent(Name);
7       ?count(N);
8       .print(N, ". My name is ", Name, ".");
9       -+count(N+1);
10      .wait(2000);
11      !count.
12
13  +?count(N) <-
14      N=1;
15      .fail.
16
17  -?count(N): N \== 1 <-
18      N=1;
19      +count(N).
```

Plans: Belief

{+|-}event[source(type)]:
 action 1;
 action 2;
 action n.
 context ←

Plans: Belief

{+ | -}event



defines if it is an
addition (+) or a
deletion (-) plan.

[source(type)]:
action 1;
action 2;
action n.

context ←

```
+event[source(type)]:  
    context ←  
        action 1;  
        action 2;  
        action n.
```

Plans: Addition Belief Goal

+event[source(type)]:

context ←

action 1;

action 2;

action n.

defines an addition
plan.



Plans: Addition Belief Goal

alice Jason ▾

```
1 stock(beer, 20).
2
3 !purchase(beer, 10).
4
5 +!purchase(Item, Amount):
6     stock(Item, Stock) &
7     Stock >= Amount <-
8         .print("Your product ", Item, " is available. We have ", Stock, " units.");
9         -+stock(Item, Stock-Amount).
10
11 +stock(Item, NewStock) <-
12     .print("The stock amount for ", Item, " is now ", NewStock, ".").
13
14
15 -stock(Item, NewStock) <-
16     .print("The stock amount for ", Item, " is now decreasing from ", NewStock, ".").
```

Plans: Addition Belief Goal

```
● alice Jason ▾

1 stock(beer, 20).
2
3 !purchase(beer, 10).
4
5 +!purchase(Item, Amount):
6     stock(Item, Stock) &
7     Stock >= Amount <-
8         .print("Your product ", Item, " is available. We have ", Stock, " units.");
9         -+stock(Item, Stock-Amount).
10
11 +stock(Item, NewStock) <-
12     .print("The stock amount for ", Item, " is now ", NewStock, ".").
13
14
15 -stock(Item, NewStock) <-
16     .print("The stock amount for ", Item, " is now decreasing from ", NewStock, ".")
```

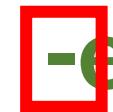
Plans: Addition Belief Goal

```
● alice | Jason ▾

1 stock(beer, 20).
2
3 !purchase(beer, 10).
4
5 +!purchase(Item, Amount):
6     stock(Item, Stock) &
7     Stock >= Amount <-
8         .print("Your product ", Item, " is available. We have ", Stock, " units.");
9         -+stock(Item, Stock - Amount).
10
11 +stock(Item, NewStock) <-
12     .print("The stock amount for ", Item, " is now ", NewStock, ".").
13
14
15 -stock(Item, NewStock) <-
16     .print("The stock amount for ", Item, " is now decreasing from ", NewStock, ".")
```

```
-event[source(type)]:  
    context ←  
        action 1;  
        action 2;  
        action n.
```

Plans: Deletion Belief Goal



defines a deletion plan.

-event[source(type)]:
context ←
action 1;
action 2;
action n.

Plans: Deletion Belief Goal

alice Jason ▾

```
1 stock(beer, 20).
2
3 !purchase(beer, 10).
4
5 +!purchase(Item, Amount):
6     stock(Item, Stock) &
7     Stock >= Amount <-
8         .print("Your product ", Item, " is available. We have ", Stock, " units.");
9         -+stock(Item, Stock-Amount).
10
11 +stock(Item, NewStock) <-
12     .print("The stock amount for ", Item, " is now ", NewStock, ".").
13
14
15 -stock(Item, NewStock) <-
16     .print("The stock amount for ", Item, " is now decreasing from ", NewStock, ".").
```

Plans: Deletion Belief Goal

```
● alice | Jason ▾

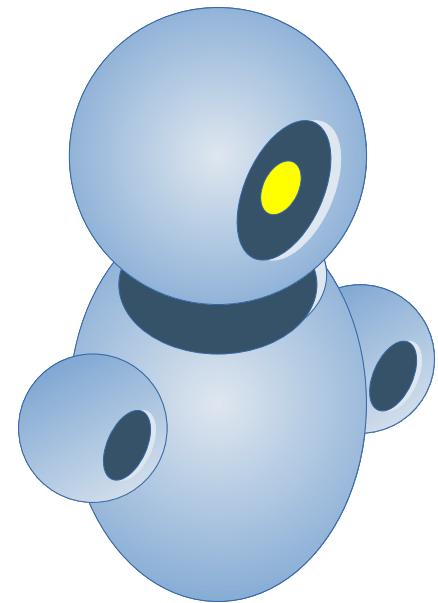
1 stock(beer, 20).
2
3 !purchase(beer, 10).
4
5 +!purchase(Item, Amount):
6     stock(Item, Stock) &
7     Stock >= Amount <-
8         .print("Your product ", Item, " is available. We have ", Stock, " units.");
9         -+stock(Item, Stock-Amount).
10
11 +stock(Item, NewStock) <-
12     .print("The stock amount for ", Item, " is now ", NewStock, ".").
13
14
15 -stock(Item, NewStock) <-
16     .print("The stock amount for ", Item, " is now decreasing from ", NewStock, ".") .
```

Plans: Deletion Belief Goal

```
● alice | Jason ▾

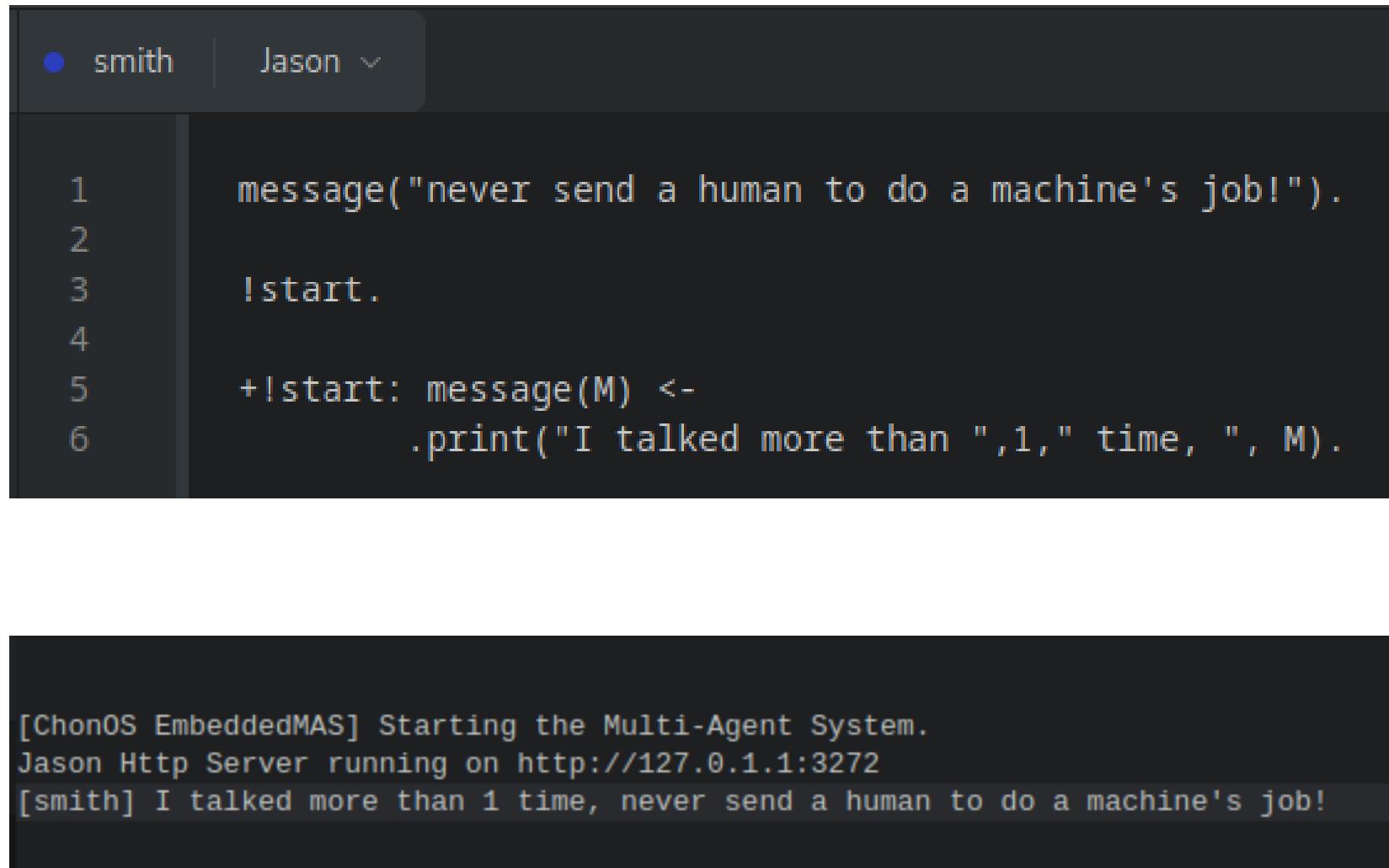
1 stock(beer, 20).
2
3 !purchase(beer, 10).
4
5 +!purchase(Item, Amount):
6     stock(Item, Stock) &
7     Stock >= Amount <-
8         print("Your product " Item, " is available. We have ", Stock, " units.");
9         -+stock(Item, Stock - Amount)
10
11 +stock(Item, NewStock) <-
12     .print("The stock amount for ", Item, " is now ", NewStock, ".")
13
14
15 -stock(Item, NewStock) <-
16     .print("The stock amount for ", Item, " is now decreasing from ", NewStock, ".")
```

Jason StdLib



.print(*parameters*)

- used for printing messages to the console where the system is running.
- It receives any number of parameters, which can be not only strings but also any AgentSpeak term (including variables).



The screenshot shows the Jason IDE interface. At the top, there are two tabs: 'smith' (selected) and 'Jason'. The code editor window contains the following code:

```
1 message("never send a human to do a machine's job!").  
2  
3 !start.  
4  
5 +!start: message(M) <-  
6     .print("I talked more than ", 1, " time, ", M).
```

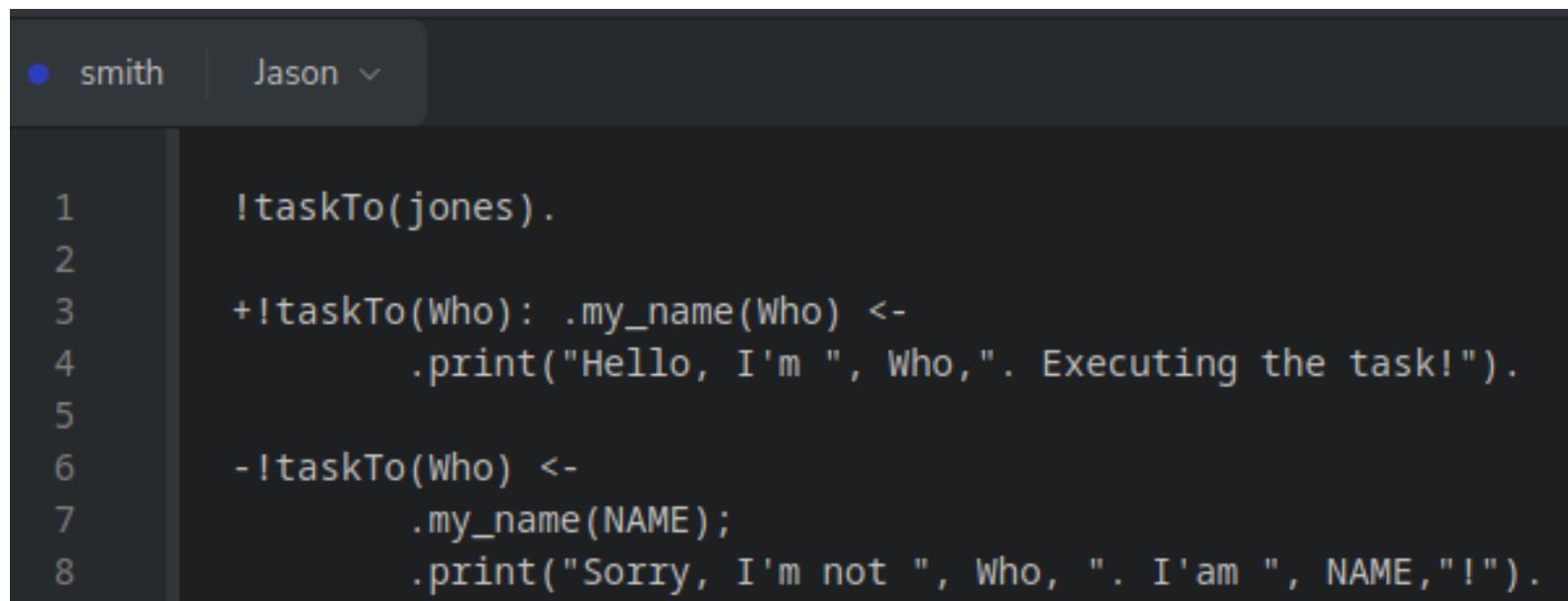
Below the code editor is a terminal window displaying the system's output:

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.  
Jason Http Server running on http://127.0.1.1:3272  
[smith] I talked more than 1 time, never send a human to do a machine's job!
```

<https://jason-lang.github.io/api/jason/stdlib/print.html>

.my_name(*atom* | *VAR*)

- gets the agent's unique identification in the multi-agent system.
- This identification is given by the runtime infrastructure of the system.



```
smith | Jason ▾

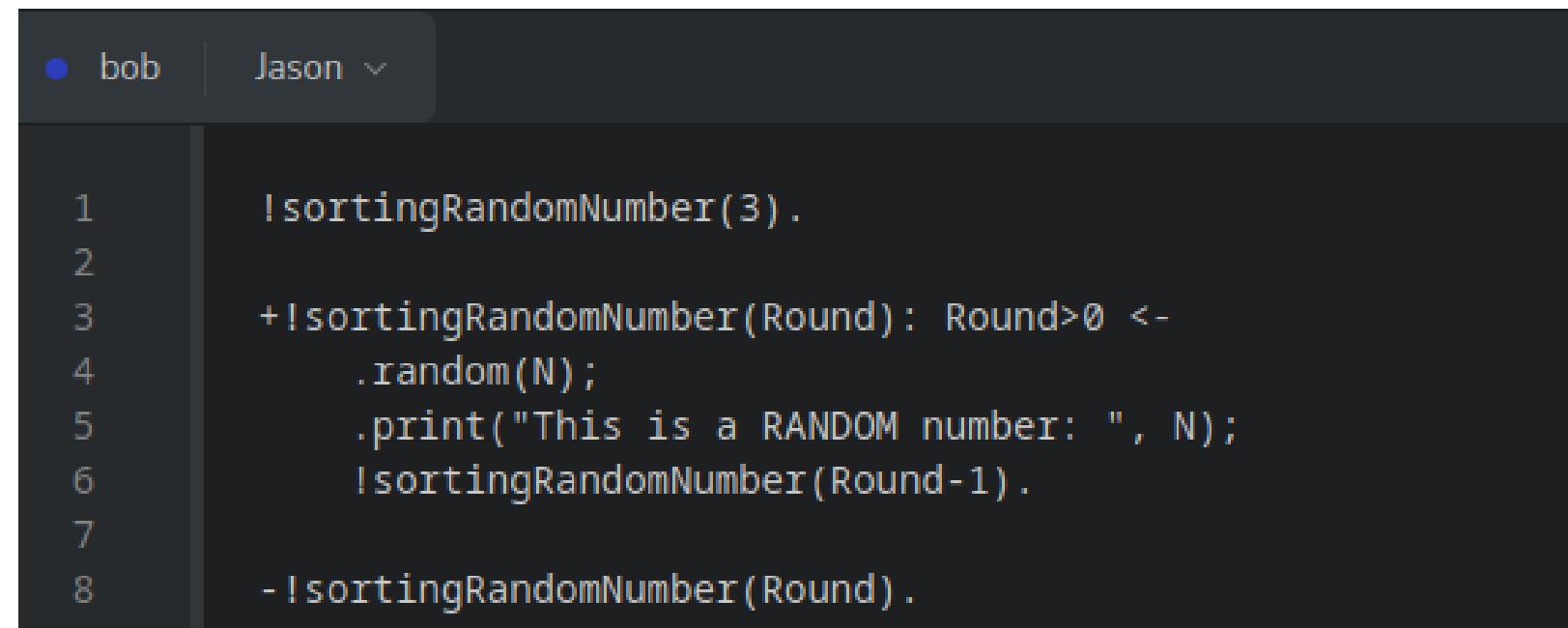
1 !taskTo(jones).
2
3 +!taskTo(Who): .my_name(Who) <-
4     .print("Hello, I'm ", Who, ". Executing the task!").
5
6 -!taskTo(Who) <-
7     .my_name(NAME);
8     .print("Sorry, I'm not ", Who, ". I'am ", NAME,"!") .
```

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.
Jason Http Server running on http://127.0.1.1:3272
[smith] Sorry, I'm not jones. I'am smith!
```

https://jason-lang.github.io/api/jason/stdlib/my_name.html

.random(VAR)

- unifies VAR with a random number between 0 and 1.



```
● bob | Jason ▾

1  !sortingRandomNumber(3) .
2
3  + !sortingRandomNumber(Round): Round>0 <-
4      .random(N);
5      .print("This is a RANDOM number: ", N);
6      !sortingRandomNumber(Round-1) .
7
8  - !sortingRandomNumber(Round) .
```

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.
Jason Http Server running on http://127.0.1.1:3272
[bob] This is a RANDOM number: 0.41921166808754884
[bob] This is a RANDOM number: 0.41014764451156094
[bob] This is a RANDOM number: 0.27646870349394936
```

<https://jason-lang.github.io/api/jason/stdlib/random.html>

.random([t1,t2, ...,tn],VAR)

- unifies VAR with a random value from the list.

```
alice.asl  x
alice.asl
1  listOfNumbers([1,2,3,4,5,6,7,8,9,0]).  

2  

3  !choosingNUmbersInAList(3).  

4  

5  +!choosingNUmbersInAList(Round): listOfNumbers(List) & Round > 0 <-
6    .random(List,N);
7    !choosed(Round,N).  

8  

9  +!choosed(Round,N): numberChoosed(N) <-
10   !choosingNUmbersInAList(Round).
11  +!choosed(Round,N): not numberChoosed(N) <-
12   .print("This is a choosed number of the list: ", N);
13   +numberChoosed(N);
14   !choosingNUmbersInAList(Round-1).  

15  

16 - !choosingNUmbersInAList(Round).
```

```
[alice] This is a choosed number of the list: 3
[alice] This is a choosed number of the list: 6
[alice] This is a choosed number of the list: 9
```

<https://jason-lang.github.io/api/jason/stdlib/random.html>

.stopMAS | .stopMAS(Delay) | .stopMAS(Delay,sucess | fail)

- aborts the execution of all agents in the multi-agent system.
- return 0 in the main function means that the program executed successfully.
- return 1 in the main function means that the program does not execute successfully and there is some error.

<https://jason-lang.github.io/api/jason/stdlib/stopMAS.html>

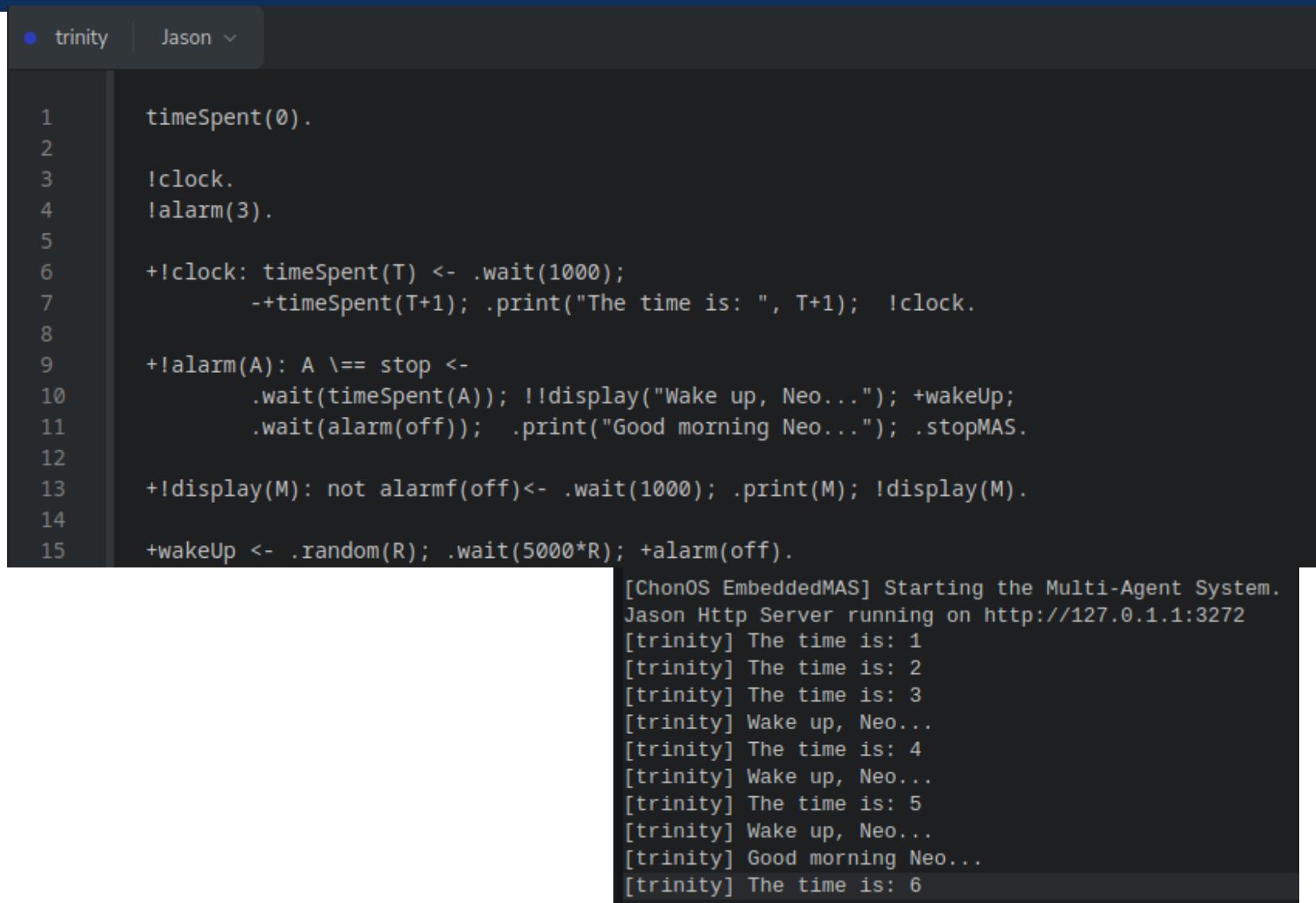
alice.asl

```
1  options([stopRightNow,stopAfter,stopWithError]).  
2  
3  !start.  
4  +!start: options(L) <- .random(L,Decision); !execute(Decision).  
5  +!show(M) <- .print(M); !show(M).  
6  
7  +!execute(stopWithError) <- !!show("Something is very wrong!!!!"); .stopMAS(0,1).  
8  +!execute(stopRightNow) <- !!show("Goodbye Cruel World!"); .stopMAS.  
9  +!execute(stopAfter) <- !!show("I still living!"); .stopMAS(250).
```

```
nilson@dell:~/chonGroup/distributedAndEmbeddedAI/course/08-GoalsAndPlans/Examples/internalActions/stopMAS/  
● stopMAS$ jason stopMAS.mas2j  
[alice] Something is very wrong!!!!  
  
FAILURE: Build failed with an exception.  
  
* What went wrong:  
Execution failed for task ':run'.  
> Process 'command '/usr/lib/jvm/java-17-openjdk-amd64/bin/java'' finished with non-zero exit value 1  
  
nilson@dell:~/chonGroup/distributedAndEmbeddedAI/course/08-GoalsAndPlans/Examples/internalActions/stopMAS/  
● stopMAS$ jason stopMAS.mas2j  
[alice] Goodbye Cruel World!  
nilson@dell:~/chonGroup/distributedAndEmbeddedAI/course/08-GoalsAndPlans/Examples/internalActions/stopMAS/  
● stopMAS$ jason stopMAS.mas2j  
[alice] I still living!  
[alice] I still living!  
[alice] I still living!
```

.wait(milliseconds) | .wait(Predicate)

- suspend the intention for the time specified by in milliseconds
- Suspend the intention until the *Predicate* is added in the belief base.



```
trinity | Jason ▾
1 timeSpent(0).
2
3 !clock.
4 !alarm(3).
5
6 +!clock: timeSpent(T) <- .wait(1000);
7     -+timeSpent(T+1); .print("The time is: ", T+1); !clock.
8
9 +!alarm(A): A \== stop <-
10      .wait(timeSpent(A)); !!display("Wake up, Neo..."); +wakeUp;
11      .wait(alarm(off)); .print("Good morning Neo..."); .stopMAS.
12
13 +!display(M): not alarmf(off)<- .wait(1000); .print(M); !display(M).
14
15 +wakeUp <- .random(R); .wait(5000*R); +alarm(off).
```

[ChonOS EmbeddedMAS] Starting the Multi-Agent System.
Jason Http Server running on http://127.0.1.1:3272
[trinity] The time is: 1
[trinity] The time is: 2
[trinity] The time is: 3
[trinity] Wake up, Neo...
[trinity] The time is: 4
[trinity] Wake up, Neo...
[trinity] The time is: 5
[trinity] Wake up, Neo...
[trinity] Good morning Neo...
[trinity] The time is: 6

<https://jason-lang.github.io/api/jason/stdlib/wait.html>

.count(predicate(term,_),Unify)

- counts the number of occurrences of a particular belief (pattern) in the agent's belief base.

```
bob | Jason ▾  
1  day(businessDay, monday).  
2  day(businessDay, tuesday).  
3  day(businessDay, wednesday).  
4  day(businessDay, thursday).  
5  day(businessDay, friday).  
6  day(weekend, saturday).  
7  day(weekend, sunday).  
8  
9  !start.  
10  
11  +!start <-  
12      .count(day(businessDay, _), Total1);  
13      .count(day(weekend, _), Total2);  
14      .print("My week has ", Total1, " business days and ", Total2, " in the weekend!").
```

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.  
Jason Http Server running on http://127.0.1.1:3272  
[bob] My week has 5 business days and 2 in the weekend!
```

<https://jason-lang.github.io/api/jason/stdlib/count.html>

.min(List,Unify) | .max(List,Unify)

```
bob Jason

1      listOfNumbers([1,2,3,4,5,6,7,8,9,0]).  
2  
3      !start.  
4      !anyNumber(8).  
5  
6      +!start: listOfNumbers(List) <-  
7          .min(List,Min); .max(List,Max);  
8          .print("The min is: ", Min , " and the max is: ", Max).  
9  
10     +!anyNumber(N): listOfNumbers(List) & .max(List,N) <- .print("The number ", N, " is the biggest in the list").  
11  
12     +!anyNumber(N): listOfNumbers(List) & .min(List,N) <- .print("The number ", N, " is the smallest in the list").  
13  
14     - !anyNumber(N) <- .print(N, " isn't nether biggest or smallest").
```

- gets the maximum value within a list of terms.

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.  
Jason Http Server running on http://127.0.1.1:3272  
[bob] 8 isn't nether biggest or smallest  
[bob] The min is: 0 and the max is: 9
```

<https://jason-lang.github.io/api/jason/stdlib/min.html>
<https://jason-lang.github.io/api/jason/stdlib/max.html>