# عامل و سیستم‌های چندعاملی

دکتر علی اکرمی‌زاده

## معماری عامل ها

### ساختارهای کلی

# معماری عامل

- مدل عامل ساده
- معماری عامل شناختی
- معماری عامل واکنشی
- معماری چندلایه

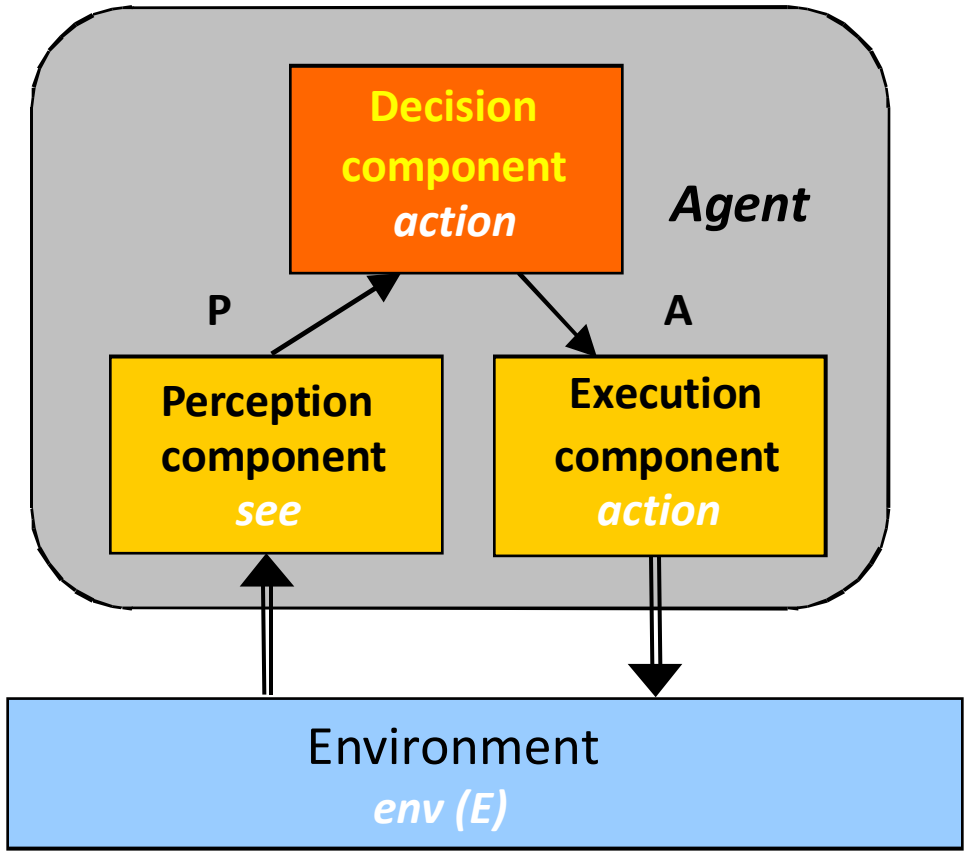
# فصل اول) مدل عامل ساده

- یک عامل محیط اطرافش را از طریق سنسورهایش درک کرده و از طریق عملگرهایش بر روی آن عمل انجام می دهد.
- هدف: طراحی یک عامل
  - طراحی تابعی که نگاشتی از دنیای ادراکات به عمل ها ایجاد کند.

$$\text{Agent} = \text{architecture} + \text{program}$$

- یک عامل منطقی
  - دارای یک سنجشگر عملکرد است که میزان موفقیت را تعیین می کند.
  - این عامل دارای یک توالی از ادراکات است که برای هر یک به دنبال یافتن عملی است که سنجشگر عملکرد را بیشینه کند.

# مدل عامل واکنشی




$E = \{e_1, .., e, ..\}$
$P = \{p_1, .., p, ..\}$
$A = \{a_1, .., a, ..\}$

**Reactive agent**

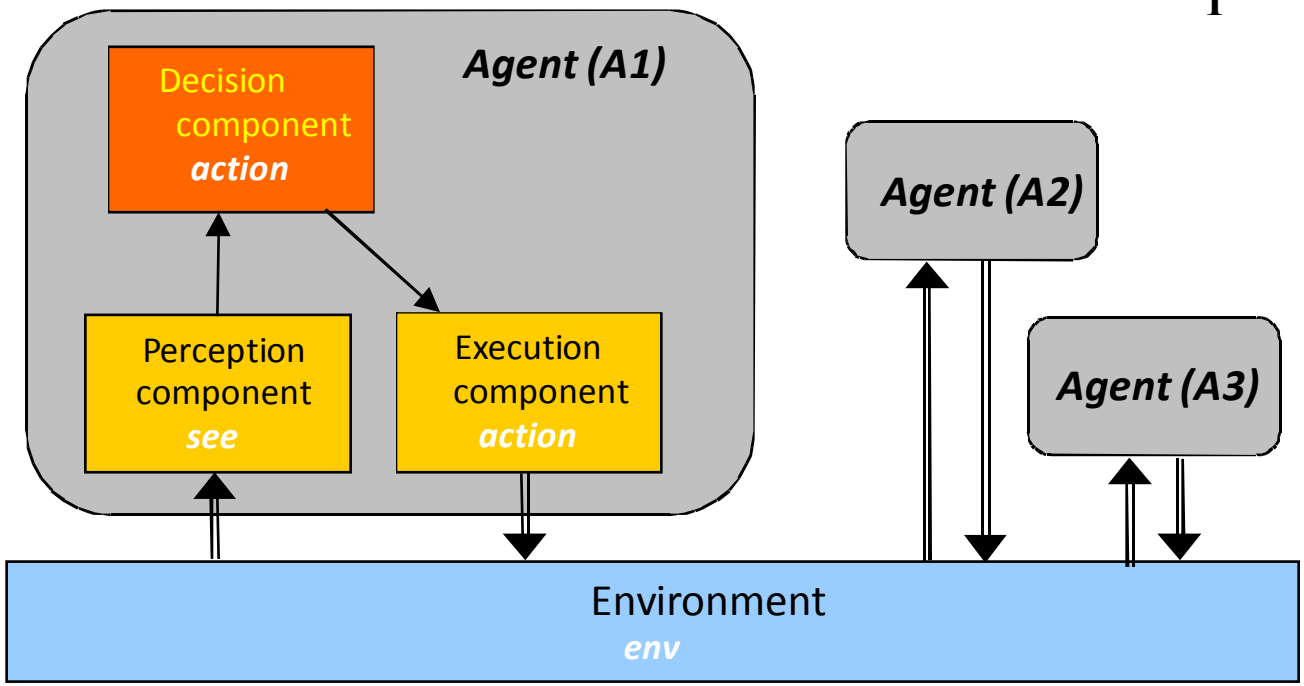
$see : E \rightarrow P$
$action : P \rightarrow A$
$env : E \times A \rightarrow E$
$(env : E \times A \rightarrow \mathcal{P}(E))$

# مدل عامل واکنشی در سیستم چندعاملی

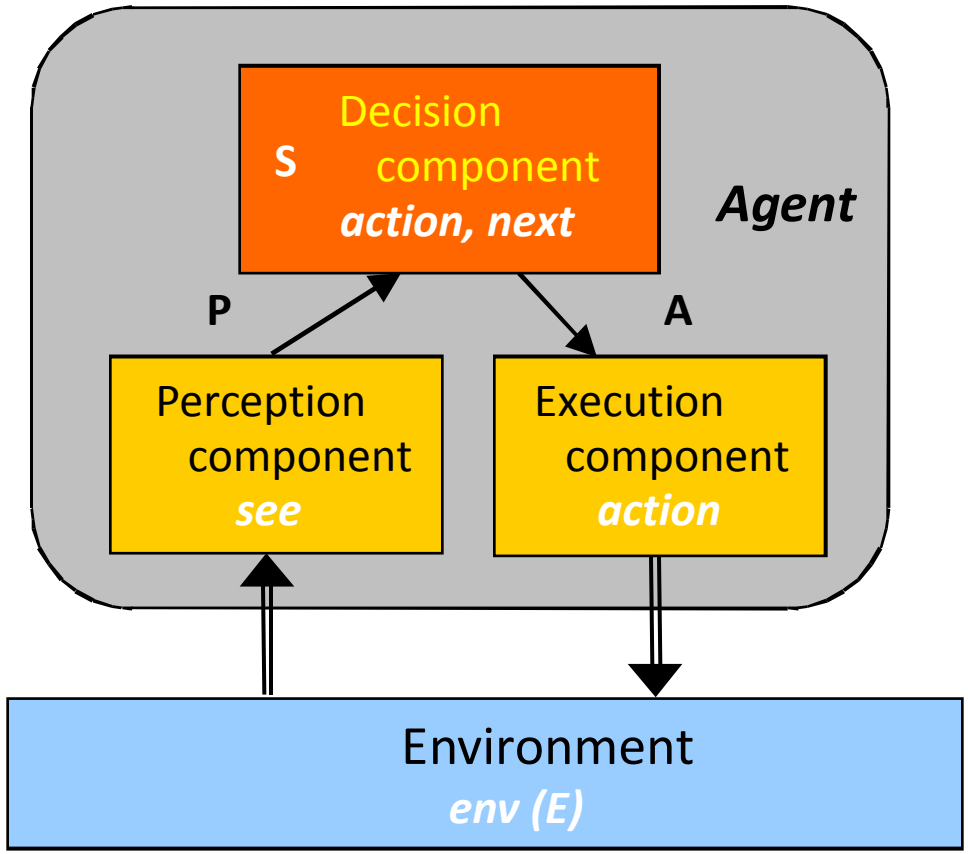
$A_1, \ldots, A_i, \ldots$
$P_1, \ldots, P_i, \ldots$
(usually the same)

**Several reactive agents**

$see_i : E \rightarrow P_i$
$action_i : P_i \rightarrow A_i$
$env : E \times A_1 \times \ldots A_n \rightarrow P(E)$



# مدل عامل حالت‌دار




$E = \{e_1, .., e, ..\}$
$P = \{p_1, .., p, ..\}$
$A = \{a_1, .., a, ..\}$
$S = \{s_1, .., s, ..\}$

**State Agent**

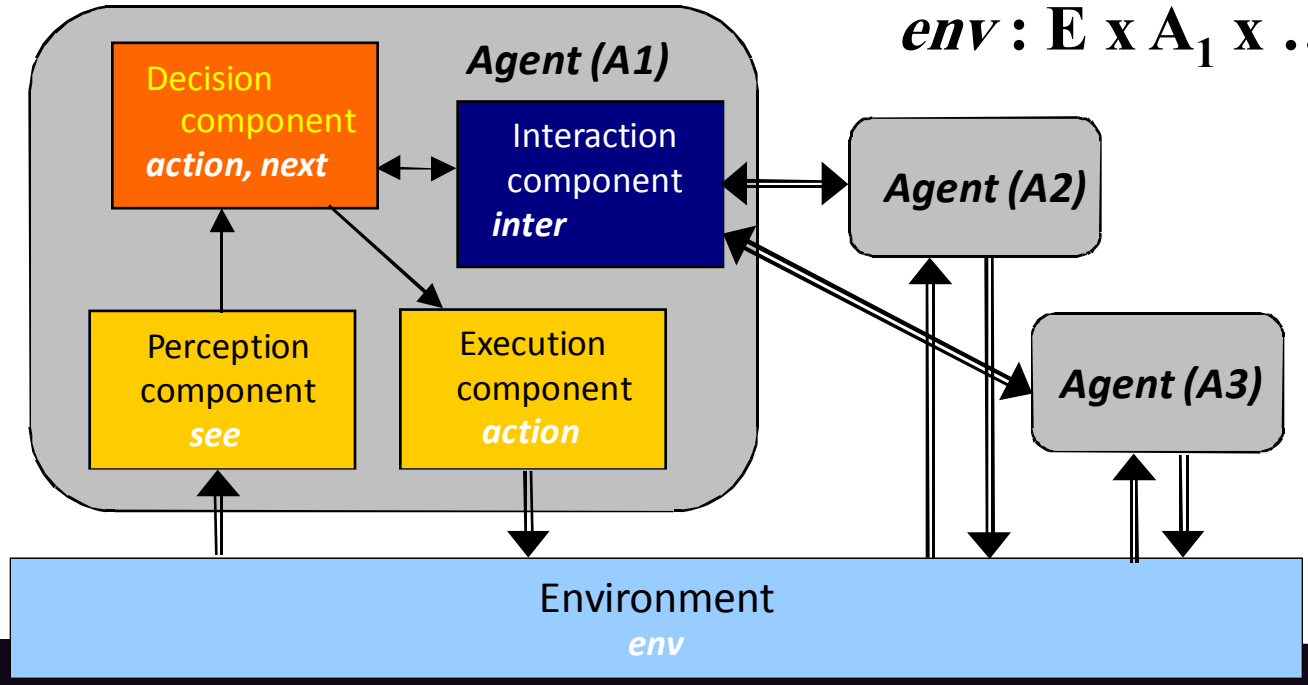
$see : E \rightarrow P$
$next : S \times P \rightarrow S$
$action : S \rightarrow A$
$env : E \times A \rightarrow P(E)$

# مدل عامل حالت‌دار در سیستم چندعاملی

**S1**,…, **Si**,..
$A_1$,…, $A_i$,..
$P_1$,…, $P_i$,..
(not always the same)
I = {$i_1$, .., $i_k$,…}

**Several cognitive agents**

$see_i : E \rightarrow P_i$
$next_i : S_i \times P \rightarrow S_i$
$action_i : S_i \times I \rightarrow A_i$
$inter_i : S_i \rightarrow I$
$env : E \times A_1 \times \ldots A_n \rightarrow P(E)$



*مدل عامل حالت‌دار*

- عامل دارای حالت و هدف

$$\text{goal} : \text{E} \rightarrow \{0, 1\}$$

- عامل دارای منفعت

$$\text{utility} : E \rightarrow R$$

- محیط نامعین

$$\text{env} : E \times A \rightarrow \text{P}(E)$$

- تئوری منفعت
  - هر حالت دارای درجه ای از قابلیت استفاده برای عامل است و آن عامل حالتی که دارای منفعت بیشتری است را ترجیح می دهد.
- تئوری تصمیم
  - یک عامل منطقی است، اگر و فقط اگر عملی را انتخاب کند که بیشترین منفعت قابل انتظار را نتیجه دهد. Maximum Expected Utility

*مدل عامل حالت‌دار ...*

- احتمال مورد انتظار اینکه انتخاب عمل a در حالت e منجر به حالت e' شود برابر است با:

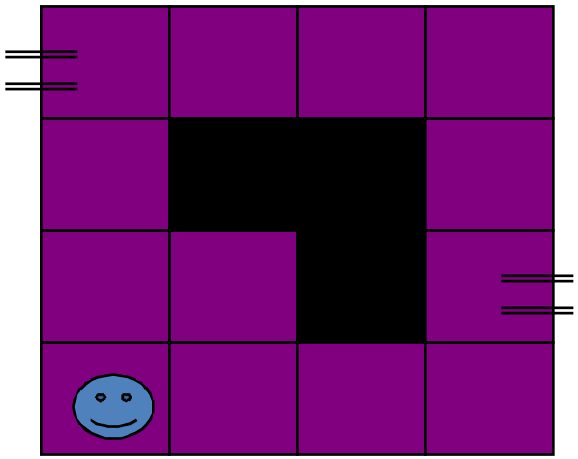
$$\sum_{e' \in env(e,a)} prob(ex(a,e) = e') = 1$$

- منفعت مورد انتظار حاصل از عمل a در حالت e از دیدگاه عامل برابر است با:

$$U(a,e) = \sum_{e' \in env(e,a)} prob(ex(a,e) = e') * utility(e')$$

Maximum Expected Utility (MEU)

# مثال خارج شدن از مارپیچ

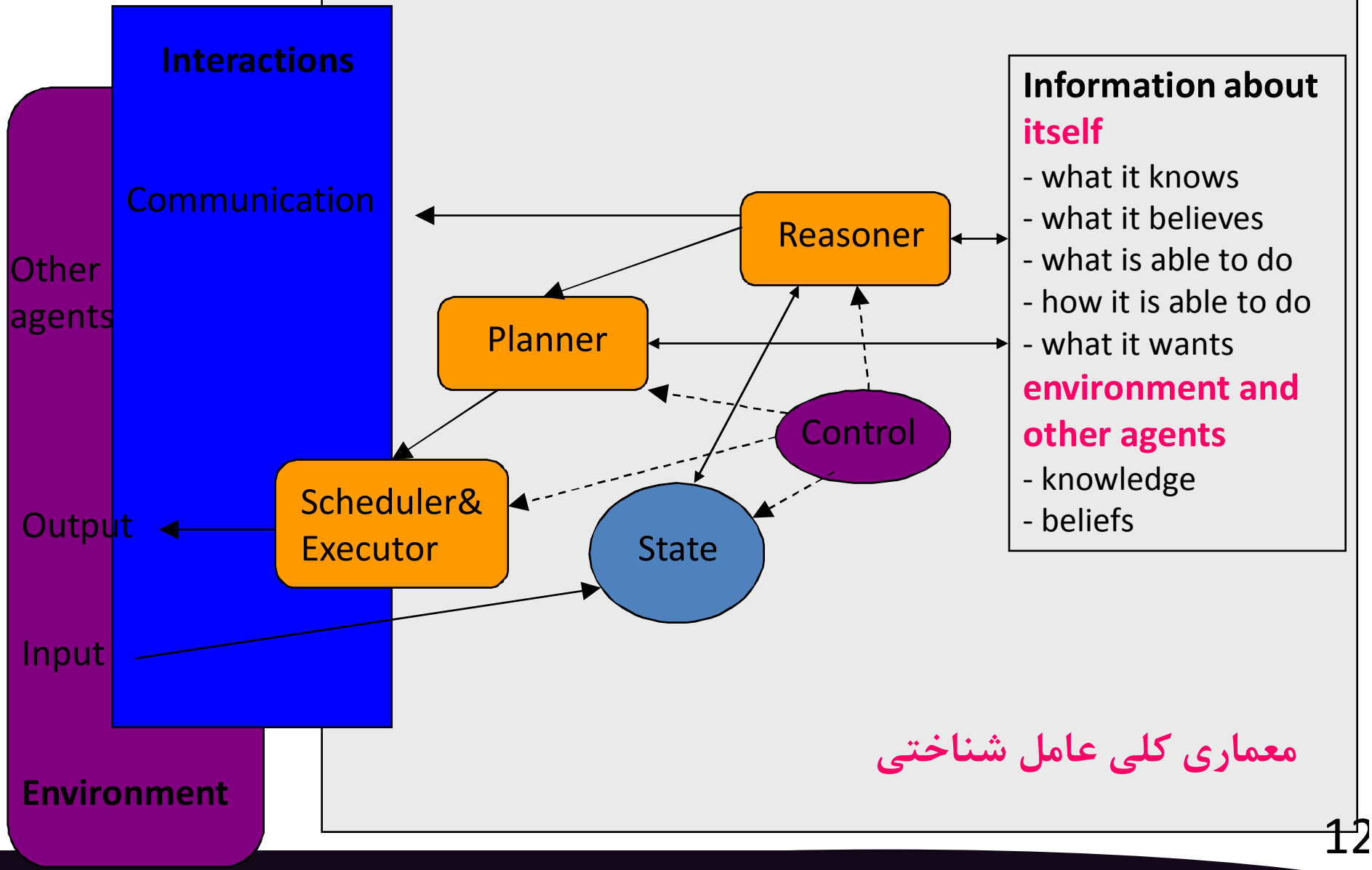


- عامل واکنشی
- عامل حالت‌دار
- عامل حالت‌دار با منفعت

- سه مسئله
  - اگر چند عمل قابل انتخاب باشد، کدامیک را باید انتخاب کرد؟
  - اگر نتیجه عملی شناخته نباشد، چه باید کرد؟
  - با تغییراتی که در محیط رخ می دهد چه باید کرد؟

# فصل دوم) معماری عامل شناختی

- رفتار منطقی از ترکیب روشهای هوش مصنوعی و تئوری تصمیم حاصل می شود.
  - در روشهای هوش مصنوعی، روشهای جستجوی فضای عملهای ممکن به منظور تکمیل دنباله ای که قادر باشد تحقق هدف را ممکن سازد.
  - در تئوری تصمیم، جایگزینهای مختلف شناسایی و اولویت بندی می شوند و یکی از آنها انتخاب می شود.
- صورت مسئله = اندیشیدن/تصمیم گرفتن
  - منابع عامل محدود است
  - مادامی که عامل تفکر می کند، ممکن است محیط تغییر کند.




معماری کلی عامل شناختی

# First Order Predicate Logic

- نمایش سمبولیک دانش + استفاده از مفسرهای منطقی ← استنتاج آنچه که باید انجام داد
- قواعد استنتاجی
  - $At(0,0) \wedge Free(0,1) \wedge Exit(east) \rightarrow Do(move\_east)$
  - Facts and rules about the environment
  - $At(0,0)$ $Wall(1,1)$ $\forall x \forall y \quad Wall(x,y) \rightarrow \neg Free(x,y)$
  - Automatically update current state and test for the goal state
  - $At(0,3)$ or $At(3,1)$
- استفاده از حساب وضعیت به منظور توصیف تغییراتی در FOPL
  - حساب تغییرات (Situation calculus) یک فرمول بندی منطقی است که برای نمایش و استدلال در محیطهای دینامیکی طراحی شده است. این رهیافت اولین بار توسط John McCarthy در سال ۱۹۶۳ معرفی شده است که بعداً در سال ۱۹۹۱ توسط Ray Reiter تکوین شده است.
  - Function Result(Action,State) = NewState
  - $At((0,0), S_0) \wedge Free(0,1) \wedge Exit(east) \rightarrow At((0,1), Result(move\_east,S_0))$
  - Try to prove the goal $At((0,3), \_)$ and determines actions that lead to it
  - means-end analysis

## *FOPL*

- مزایا
  - سادگی و زیبایی
  - قابلیت اجرا
- معایب
  - مشکل بودن نمایش تغییرات در طول زمان
    - شاید سایر روشها مناسبتر باشند
  - نیاز به انجام روشهای اسنتتاجی برای رسیدن به جواب
    - روشهای تعیین استراتژی مناسبتر و ساده تر هستند.
  - عدم انعطاف پذیری
  - semi-decidable
    - در تئوری محاسبات مجموعه S را Semi-decidable می نامند اگر وجود داشته باشد الگوریتمی که اگر ورودی به آن داده شود، فقط و فقط در صورتی به جواب می رسد که از همان مجموعه S ورودی اعمال شده باشد.

# معماری BDI (Belief-Desire-Intention)

- تئوری استدلال عملی که توسط Bratman در سال ۱۹۸۸ ارائه شده است.
- بیان سطح بالایی از معماری یک عامل با منابع محدود که
  - باور : اطلاعاتی که عامل از محیط دارد.
  - خواسته : وجهی از مسائل که عامل دوست دارد اتفاق بیفتد.
  - هدف : علایقی که عامل متعهد به دستیابی به آنها است.
- خواسته ها نقش مهمی در استدلال عملی بیان می کنند که باعث محدود شدن انتخابها می شود.

*BDI …*

- این رهیافت روش قابل قبول است چرا که:
  - پشتوانه روانشناسانه : بر مبنای تئوری تصمیم گیری منطقی در انسان بیان شده است.
  - معماری نرم افزاری : در تعدادی از مسائل پیچیده دنیای واقعی به خوبی مورد استفاده قرار گرفته اند.
    - IRMA (Intelligent Resource-bounded Machine Architecture)
    - PRS - Procedural Reasoning System
  - پشتوانه های منطقی : مدلها در تعدادی از منطقهای BDI به خوبی فرمول بندی شده است.
    - Rao & Georgeff, Wooldrige

# حلقه کنترلی عامل BDI

$\mathbf{B} = B_0$ $\mathbf{I} = I_0$ $\mathbf{D} = D_0$

**while** true **do**

get next percept p

**B** = **brf**(**B**,p)

**D** = **options**(**B**, **D**, **I**)

**I** = **filter**(**B**, **D**, **I**)

$\pi$ = **plan**(**B**, **I**)

execute($\pi$)

**end while**

# ساختار BDI


Belief revision

Beliefs
Knowledge

B = *brf*(B, p)

Opportunity analyzer

Deliberation process

Desires

D = *options*(B,D, I)

Intentions

Filter

Means-end reasonner

I = *filter*(B, D, I)

**Intentions structured in partial plans**

π = *plan*(B, I)

Library of plans

**Plans**

Executor

actions

# استراتژیهای تعهد

- گزینه ای که توسط عامل به عنوان یک هدف انتخاب شده است باعث ایجاد تعهد عامل نسبت به آن هدف می شود.
- تعهدات بیانگر پایبندی موقتی اهداف هستند.
- سئوال:
  - یک عامل تا چه زمان به یک تعهد پایبند است؟
- انواع تعهدات
  - تعهد کورکورانه ()
  - تعهد تک هدف ( Single minded)
  - تعهد آزاد ( Open minded)

# حلقه کنترلی عامل BDI

تعهد کورکورانه

```
B
I = I0  D = D0
while true do
        get next perceipt p
        B = brf(B,p)
        D = options(B, D, I)
        I = filter(B, D, I)
        π = plan(B, I)
        while not (empty(π) or succeeded (I, B)) do
                α = head(π)
                execute(α)
                π = tail(π)
                get next perceipt p
                B = brf(B,p)
                if not sound(π, I, B) then
                        π = plan(B, I)        ← Reactivity, replan
        end while
end while
```

# حلقه کنترلی عامل BDI

## تعهد تک هدف

```
B
I = I0  D = D0
while true do
        get next perceipt p
        B = brf(B,p)
        D = options(B, D, I)
        I = filter(B, D, I)
        π = plan(B, I)
        while not (empty(π) or succeeded (I, B) or impossible(I, B)) do
                α = head(π)
                execute(α)
                π = tail(π)
                get next perceipt p
                B = brf(B,p)
                if not sound(π, I, B) then
                        π = plan(B, I)
        end while
end while
```

*Dropping intentions that are impossible or have succeeded*

*Reactivity, replan*

# حلقه کنترلی عامل BDI

تعهد آزاد

```
B
I = I0  D = D0
while true do
        get next perceipt p
        B = brf(B,p)
        D = options(B, D, I)
        I = filter(B, D, I)
        π = plan(B, I)
        while not (empty(π) or succeeded (I, B) or impossible(I, B)) do
                α = head(π)
                execute(α)
                π = tail(π)
                get next perceipt p
                B = brf(B,p)
                if reconsider(I, B) then
                    D = options(B, D, I)
                    I = filter(B, D, I)
                π = plan(B, I)          ← Replan
        end while
end while
```

*BDI …*

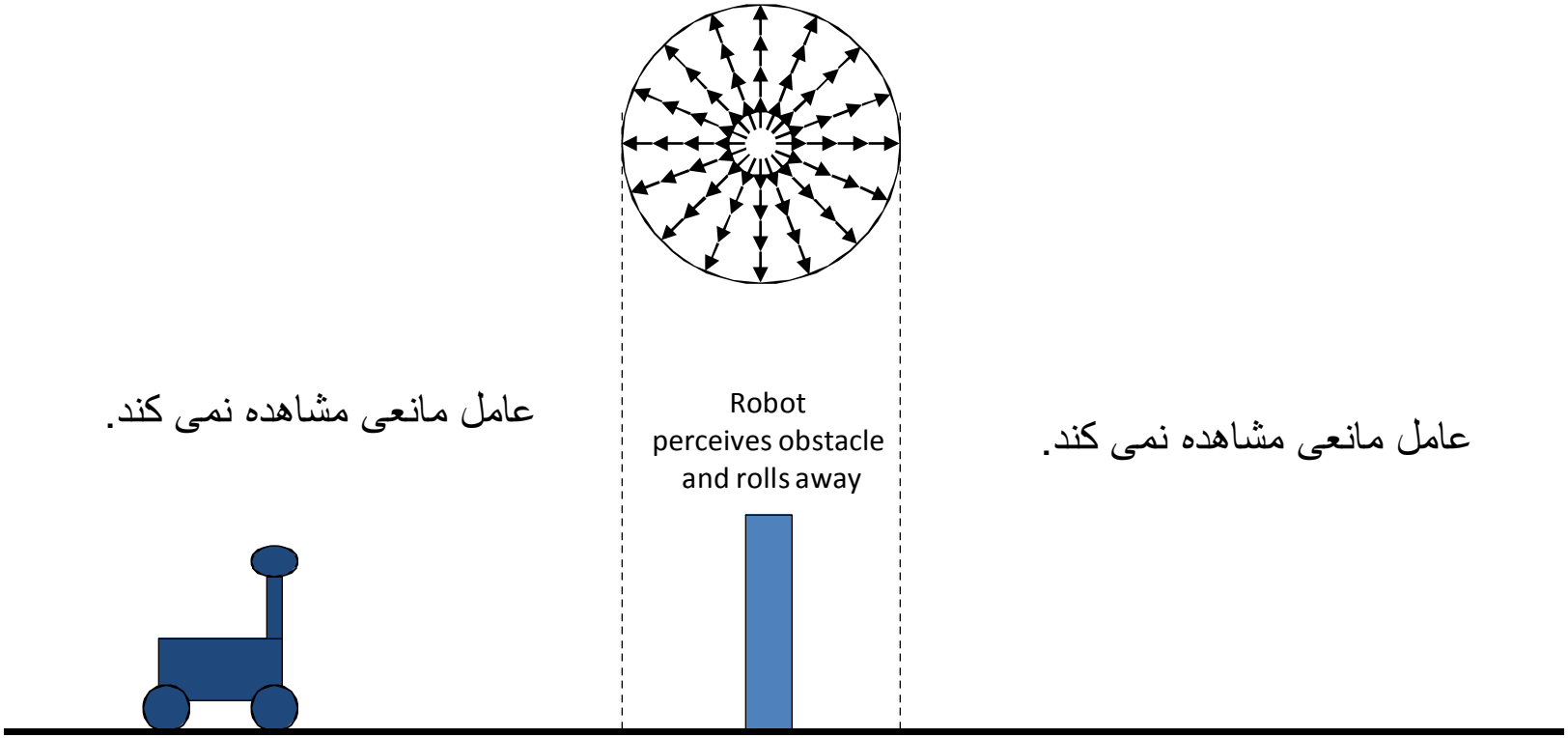
- این معماری یکی از محبوب ترینها است.
- البته معماری یکسانی وجود ندارد.
- تعدادی از معماریهای شناخته شده تر

- PRS - Procedural Reasoning System (Georgeff)
- dMARS
- UMPRS si JAM – C++
- JADE (http://jade.tilab.com/ )
- JACK (http://aosgrp.com/index.html)
- JADEX – XML si Java
- (http://vsis-www.informatik.uni-hamburg.de/projects/jadex/)
- JASON – Java (http://jason.sourceforge.net/)

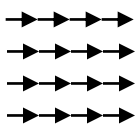
# فصل سوم) معماری عاملهای واکنشی

- معماریهای شناخته شده
  - حوزه پتانسیل
  - Subsumption architecture
- اگرچه در اواخر دهه ۸۰ مطرح شده اند ولی هنوز استفاده می شوند.
  - پروژه های عملی مانند رباتها به این معماری علاقه مند هستند.
  - پایه روشهای ترکیبی هستند.

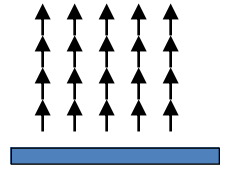
# مفهوم کلی


عامل مانعی مشاهده نمی کند.
Robot perceives obstacle and rolls away
عامل مانعی مشاهده نمی کند.

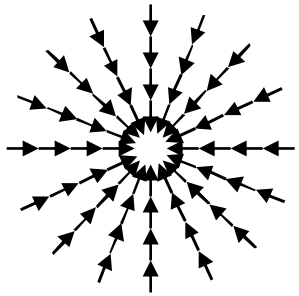
# میدانهای اولیه پتانسیلی



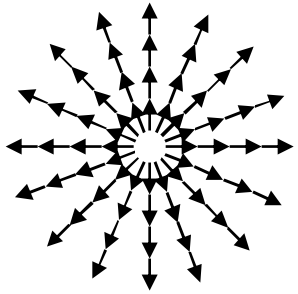
uniform



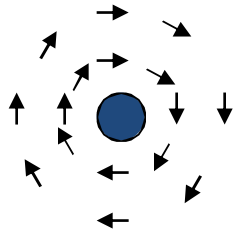
perpendicular



attractive

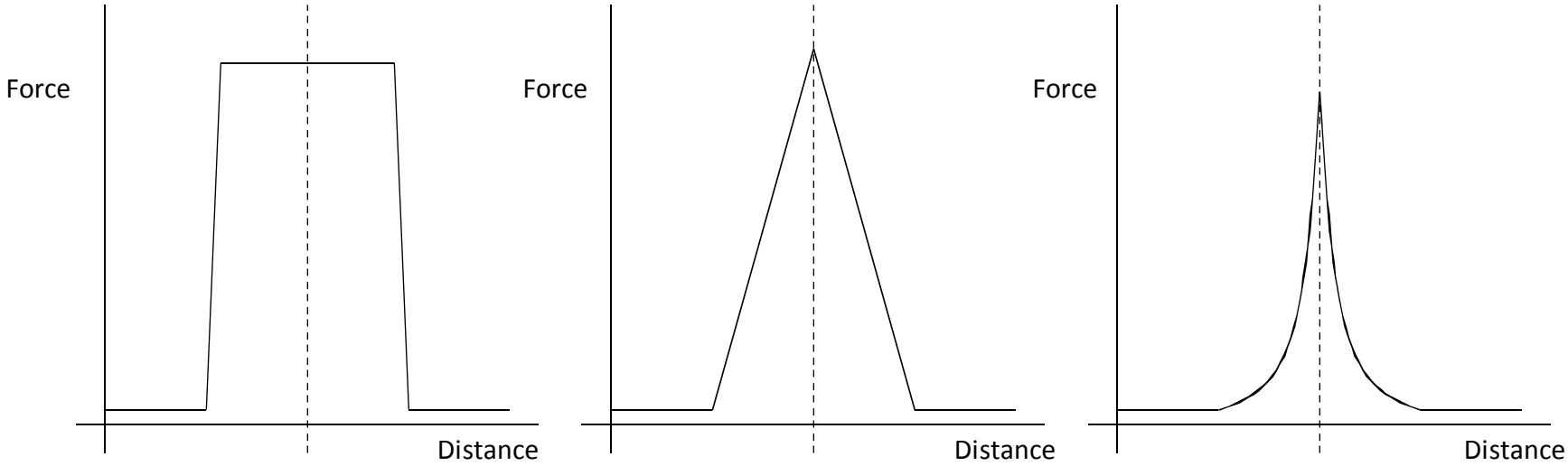


repulsion



Tangential/ rotational

# نمای مغناطیسی

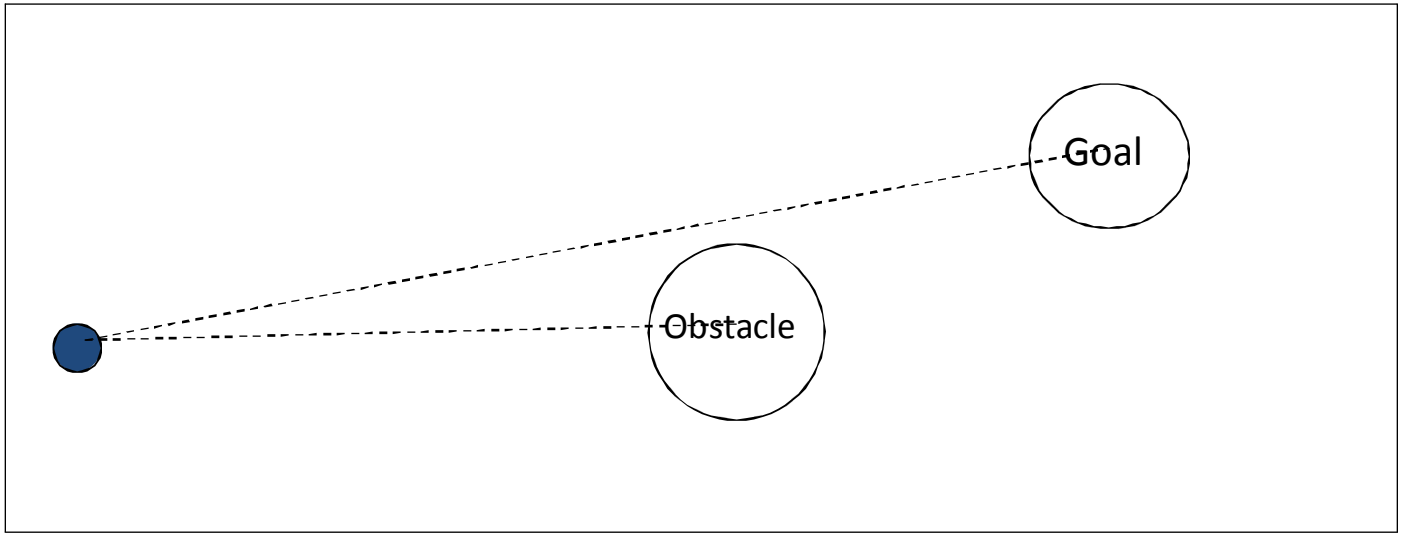


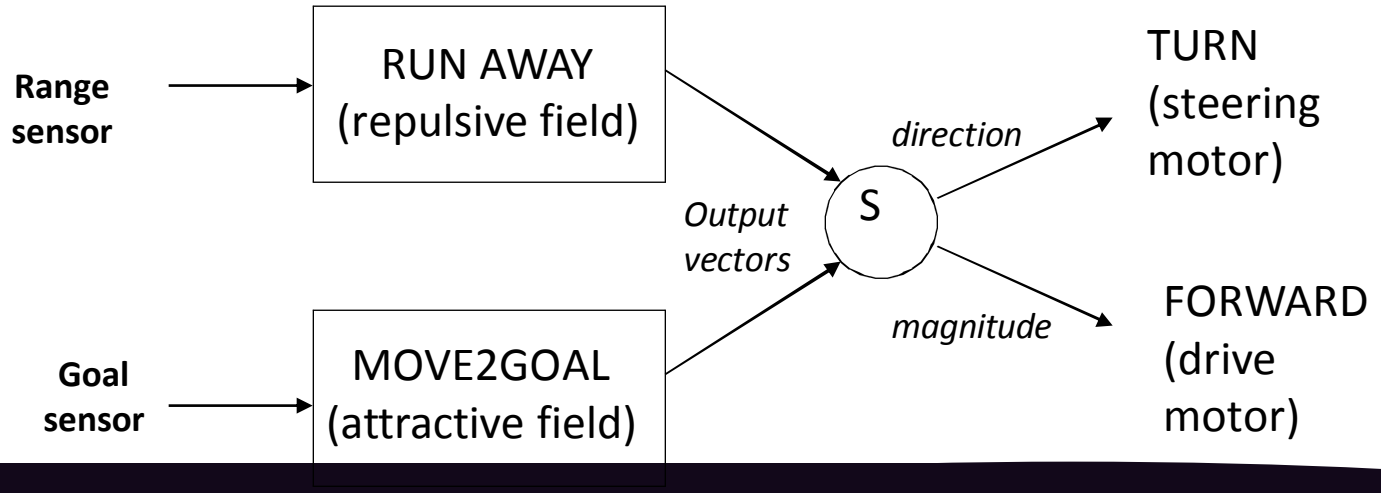

Constant magnitude

Linear drop off

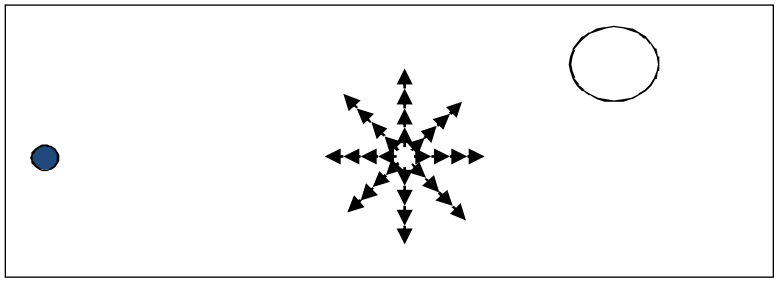
Exponential drop off

# ترکیب رفتارها

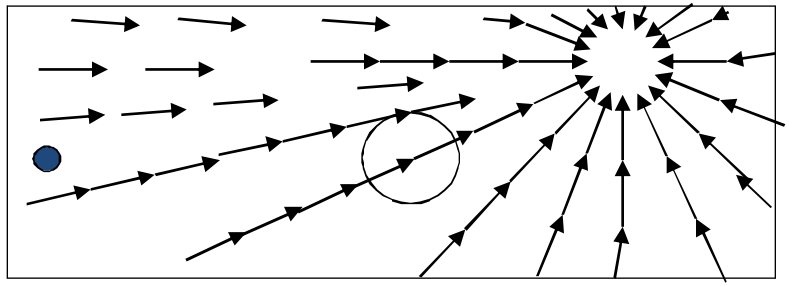


# مثال

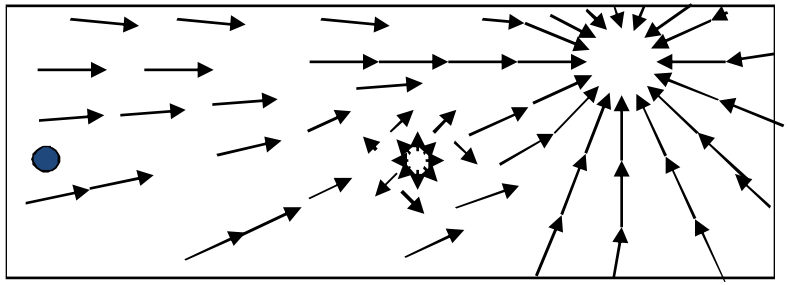


Obstacles repulsive field

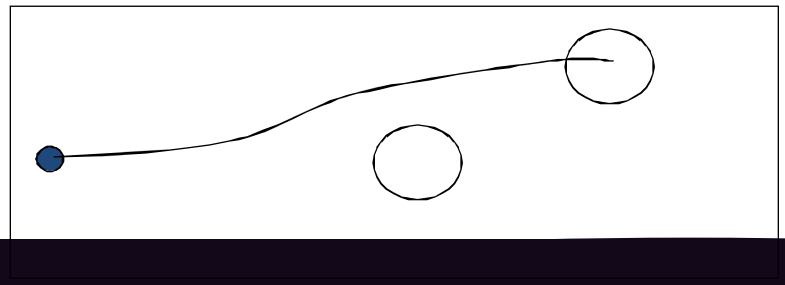


Goals attractive field

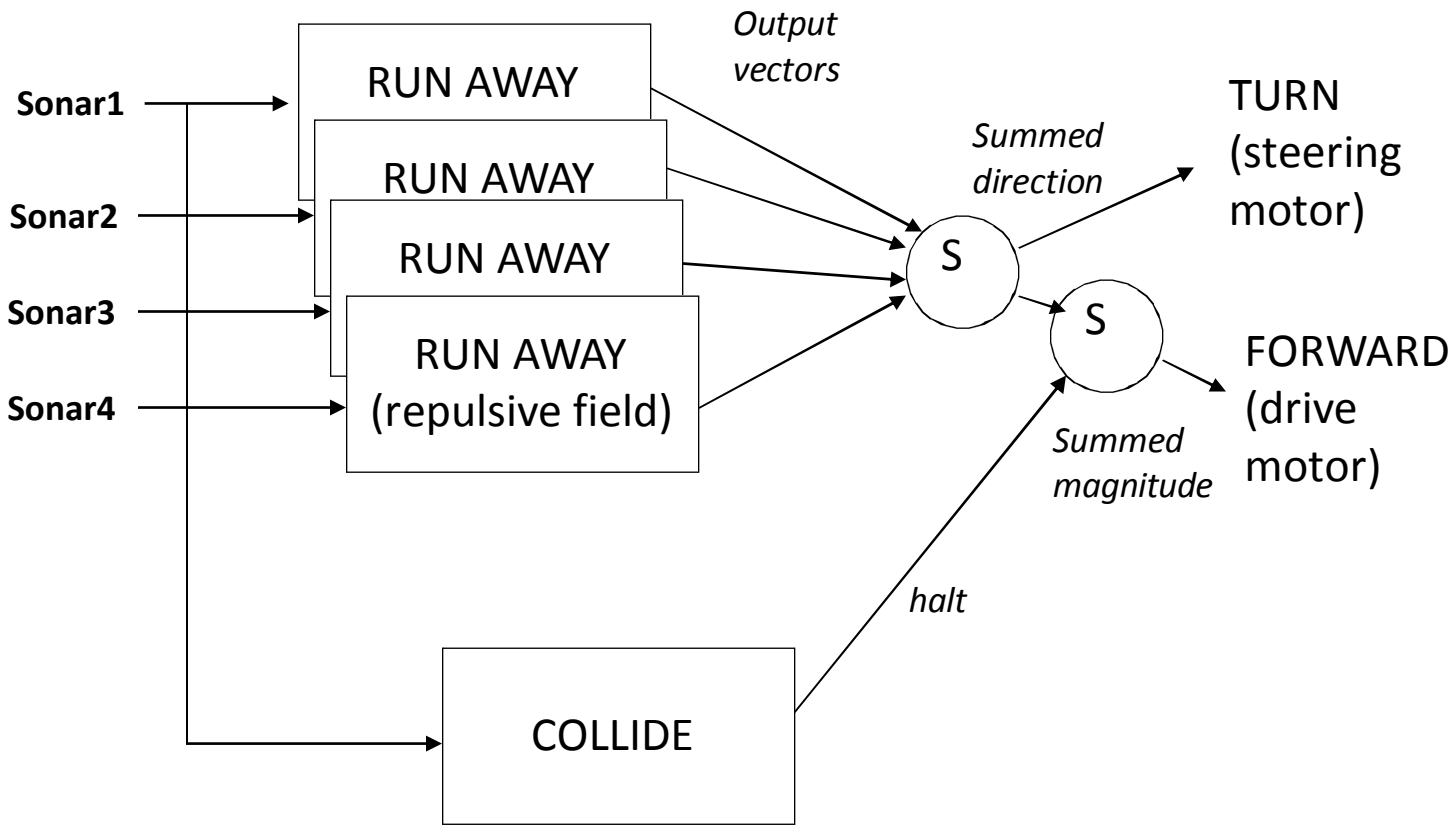
Combined vector field



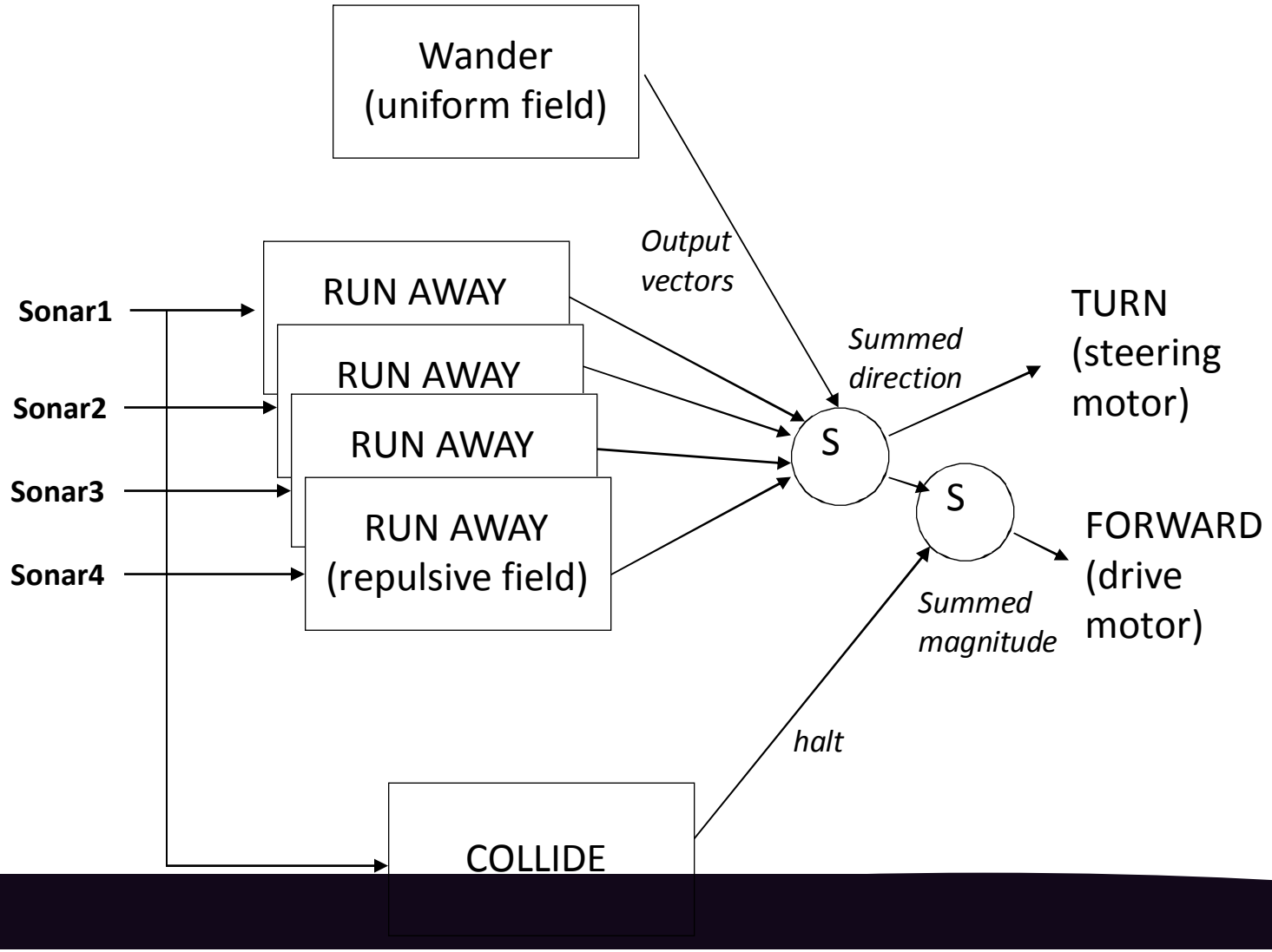
Robot path



# سطح صفر حوزه پتانسیلی



# سطح یک حوزه پتانسیلی



# سطح یک (جمع برداری)

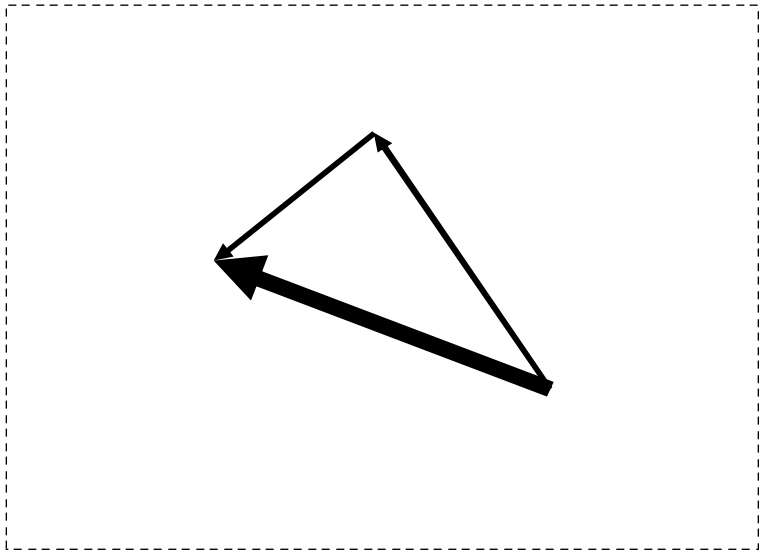
Obstacle

Vector from
WANDER

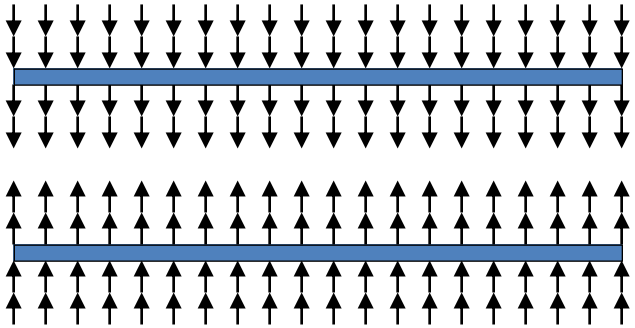
Resultant
vector

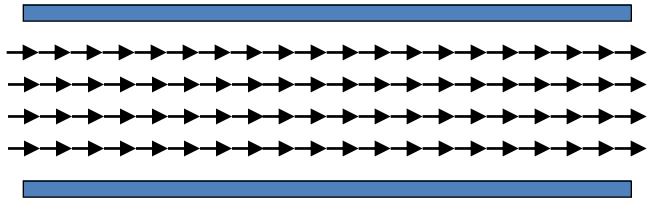
Robot

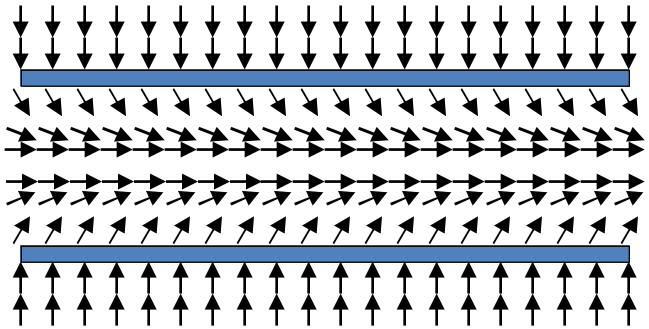
Vector from
RUNAWAY



# حرکت تعقیب راهرو در یک ربات



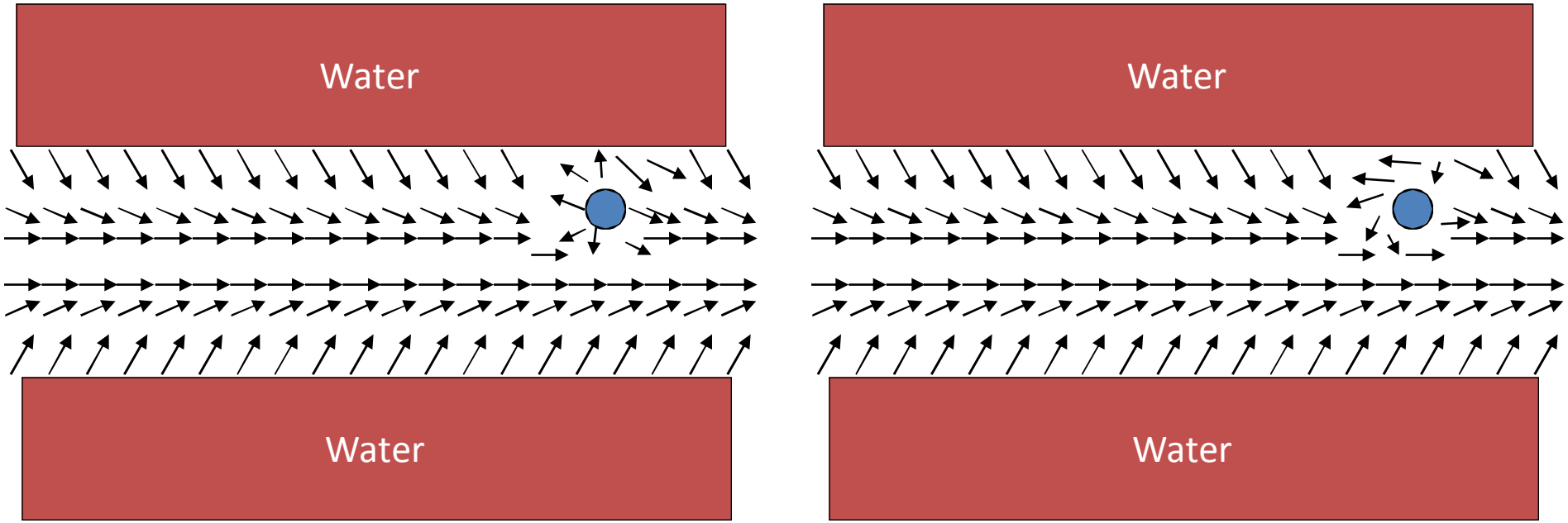




# مزایا و معایب

- ترکیب ساده رفتارها
- مینمم های محلی وجود دارند.

# مثال) حرکت بر روی پل همراه با مانع



# معماری Subsumption

- اولین بار توسط آقای Rodney Brooks
- به سادگی قابل پیاده سازی است.
- یک رفتار، یک ماژول مستقل با ورودی سنسوری از محیط و خروجی از نوع عمل قابل اجرای بر روی محیط است.
- رفتارها به صورت لایه های عمودی نوشته شده اند به گونه ایکه لایه پایین تر اولویت بالاتری در اجرا دارد.
- در حالت کلی نیازی به مدل داخلی از دنیا را ندارد.
- به منظور ساخت عامل های وظیفه گرا بسیار مناسب هستند.

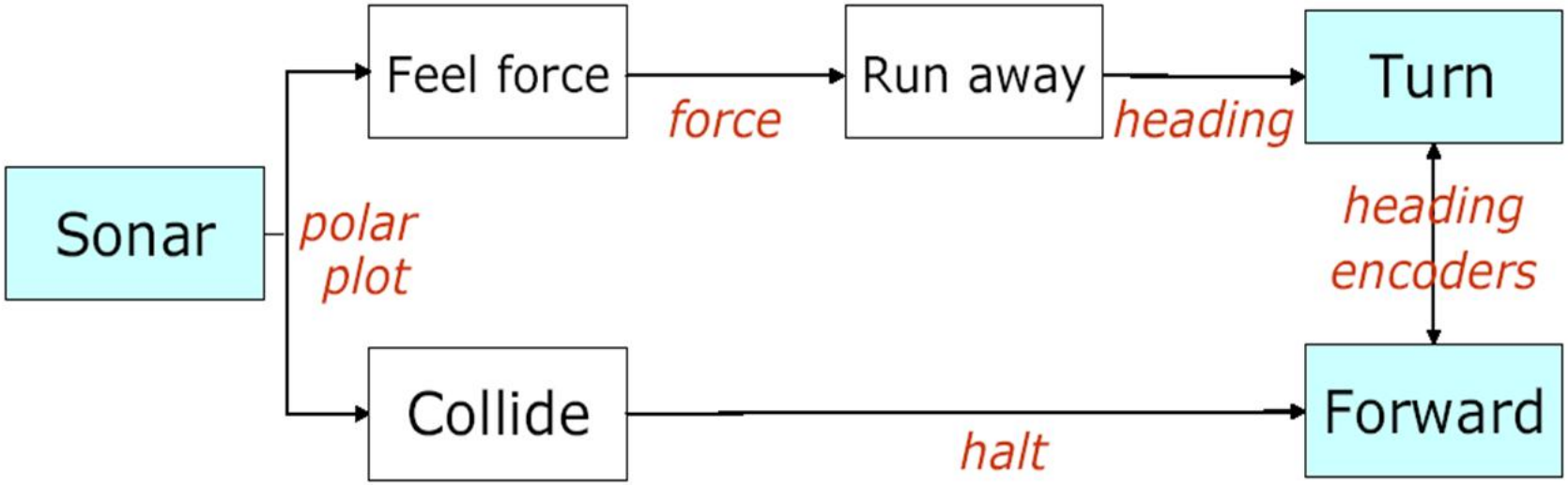
# مثال) عدم تصادم

- Sensor: A SONAR module that gives the distance to the objects in polar coordinates.

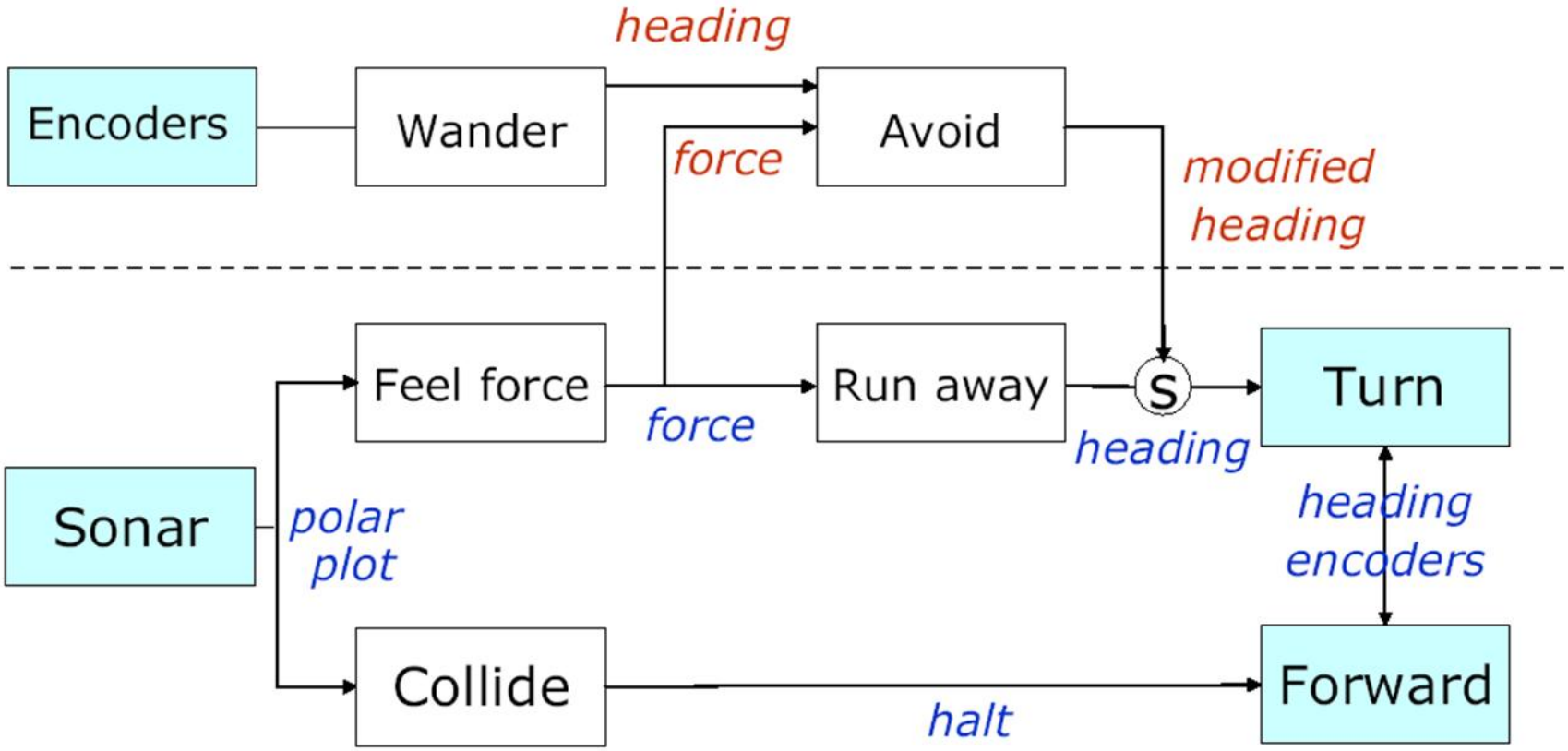
- Internal Modules
  - COLLIDE → detects if front obstacle is too close (i.e. halt)
  - FEELFORCE → sensor reading acts as repulsive force field
  - RUNAWAY → provides direction to move

- Actuators
  - TURN → provides motor output to turn robot
  - FORWARD → switches forward motion on or off

# سطح صفر) عدم تصادم

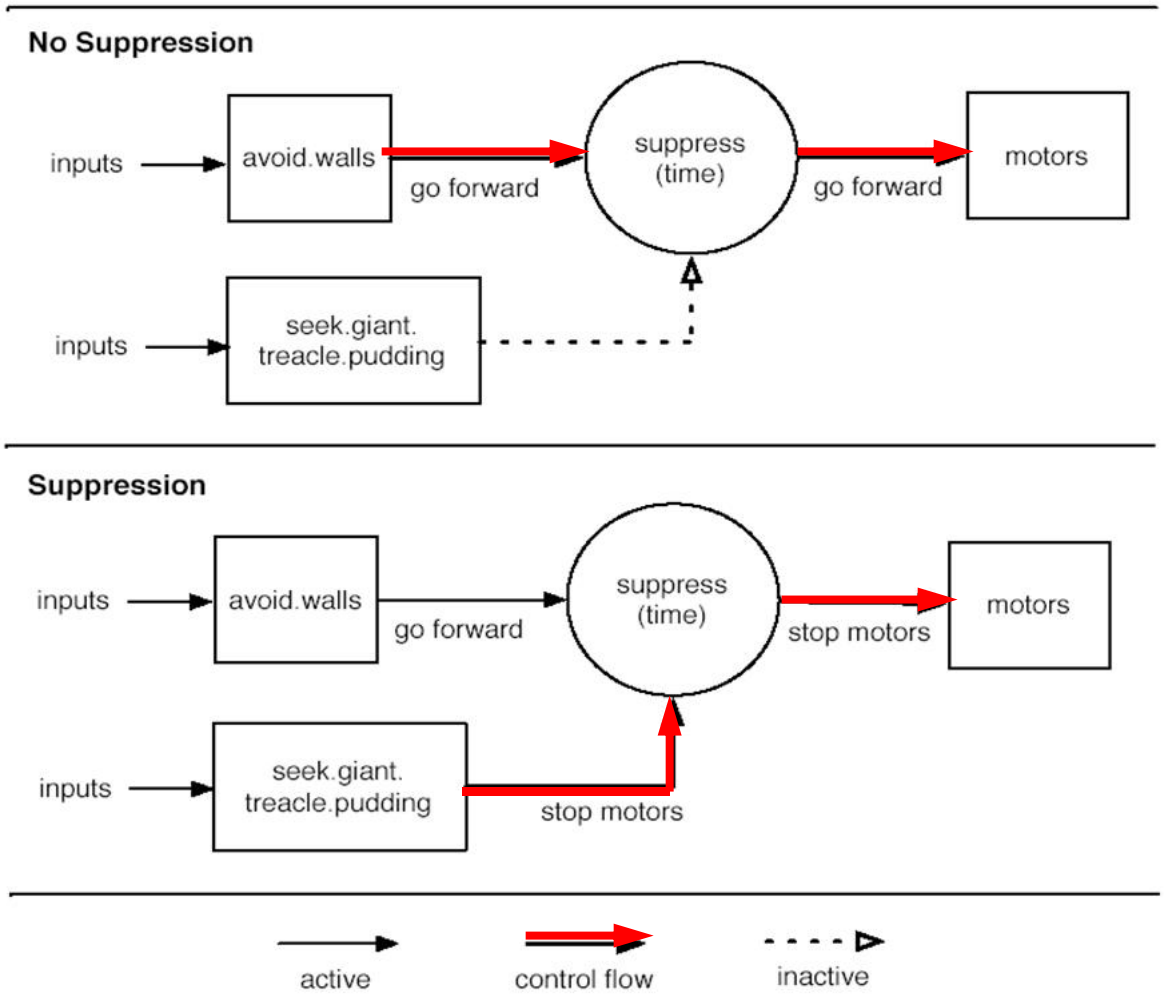


# سطح یک) پرسه زدن

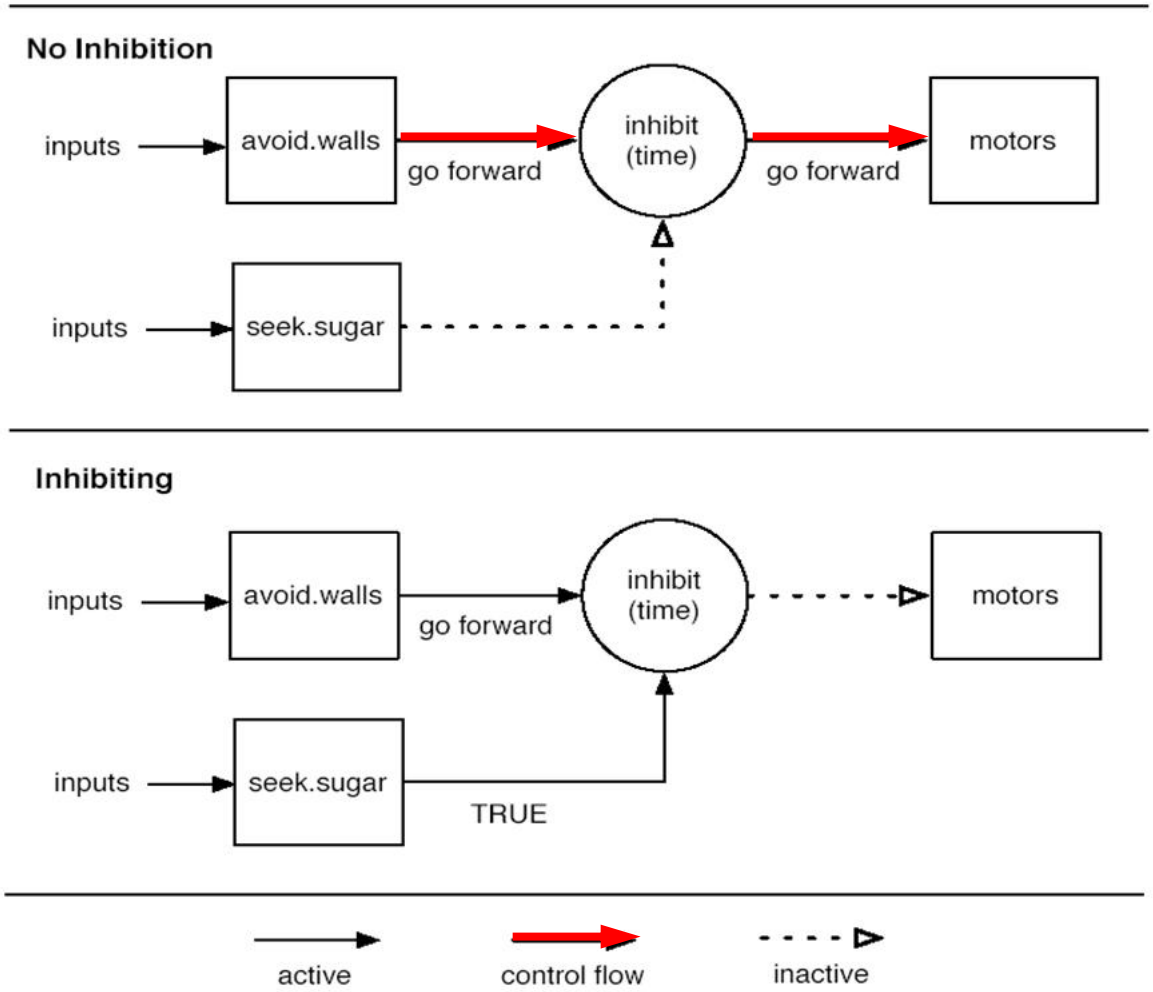



How does Turn know which module to take heading from?

**Suppression** – replaces all other inputs to the module with input coming from the "suppressing" module.



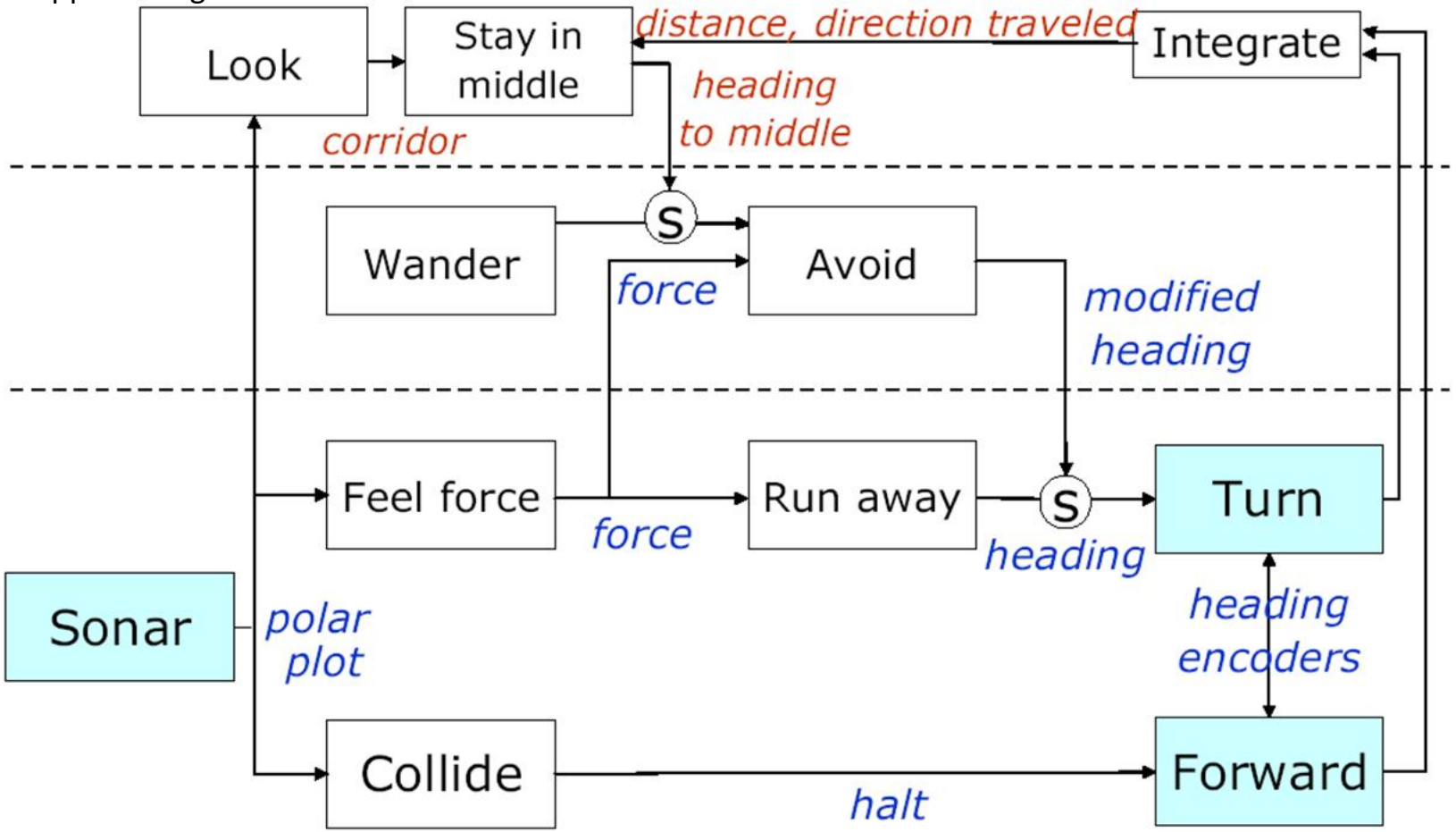
**Inhibition** – blocks output from the specified module for the defined time interval.



# سطح دو) تعقیب راهرو

Integrate - estimates how far robot has travelled off course

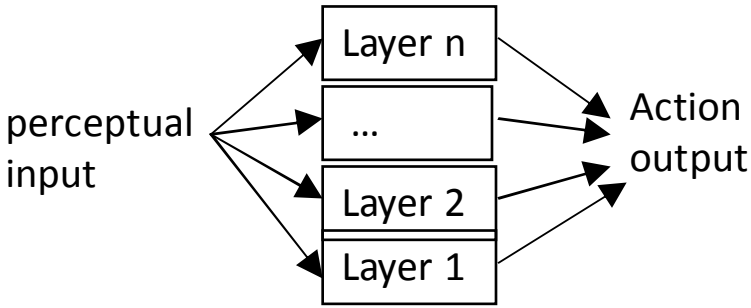
→ Supplies dangerous internal state



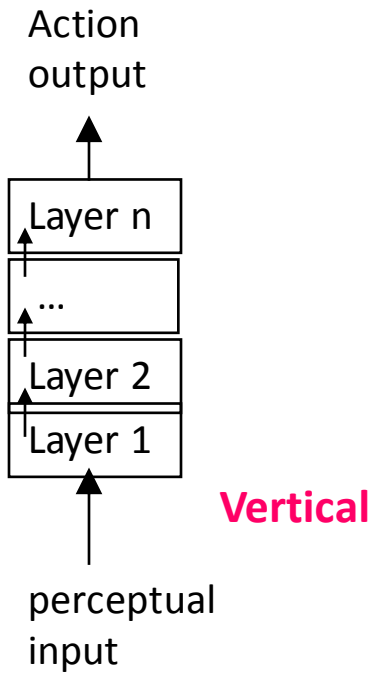
# فصل چهارم) معماری چندلایه عامل ها

- ترکیب رفتارهای واکنشی و پرواکتیو
  - رفتارهای پرواکتیو رفتارهایی است که به دنبال تولید یا کنترل حالتهای محیطی است و صرفاً قصد واکنش دادن نسبت به آنرا ندارد.
- حداقل دو لایه برای هر نوع رفتار وجود دارد.
- جریان اطلاعات
  - لایه بندی افقی
  - لایه بندی عمودی

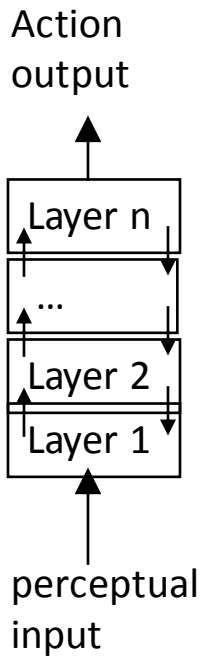
## معماری چندلایه عامل ها ...



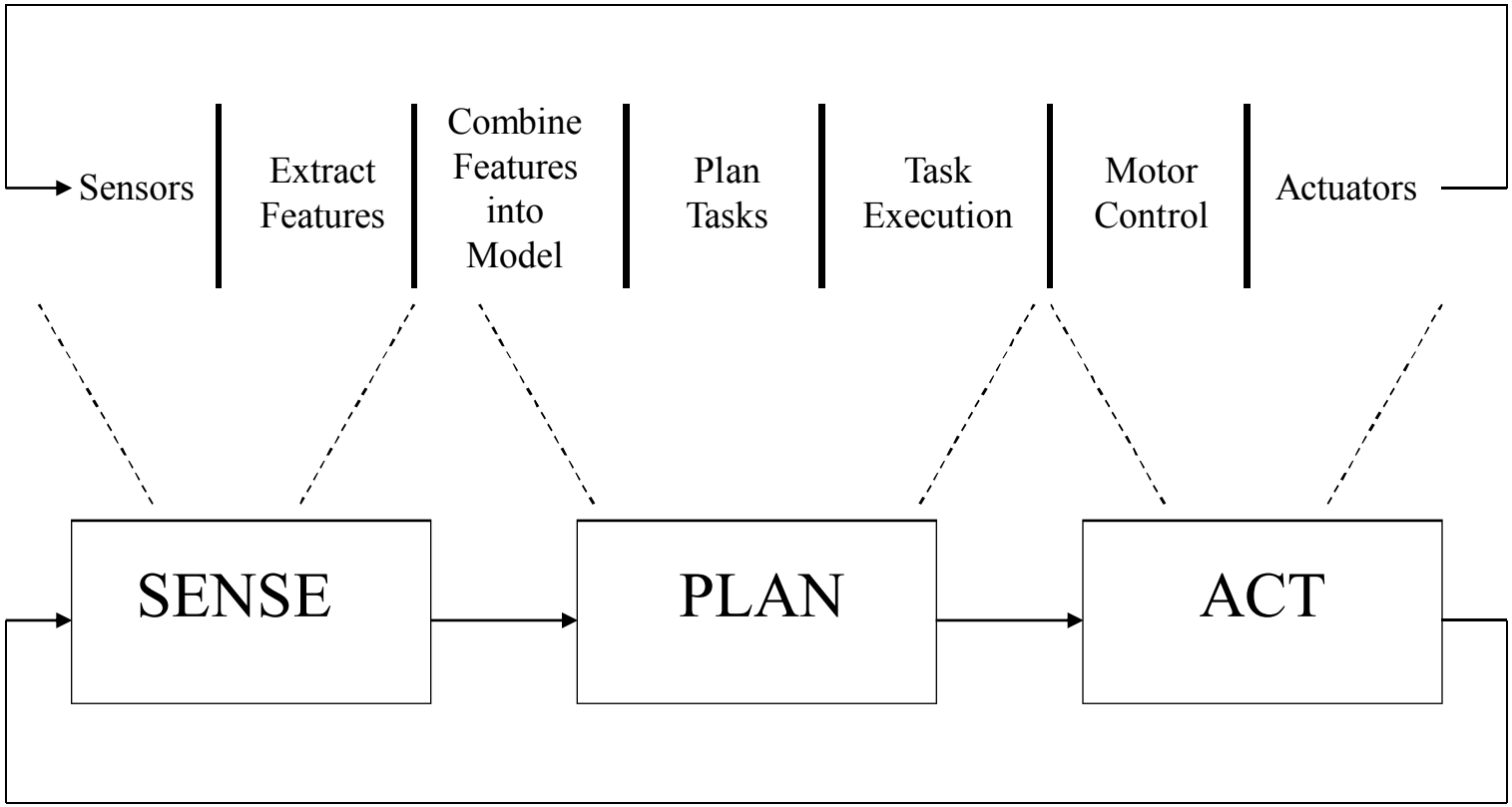

**Horizontal**




**Vertical**



# Horizontal Decomposition of Hierarchical Model

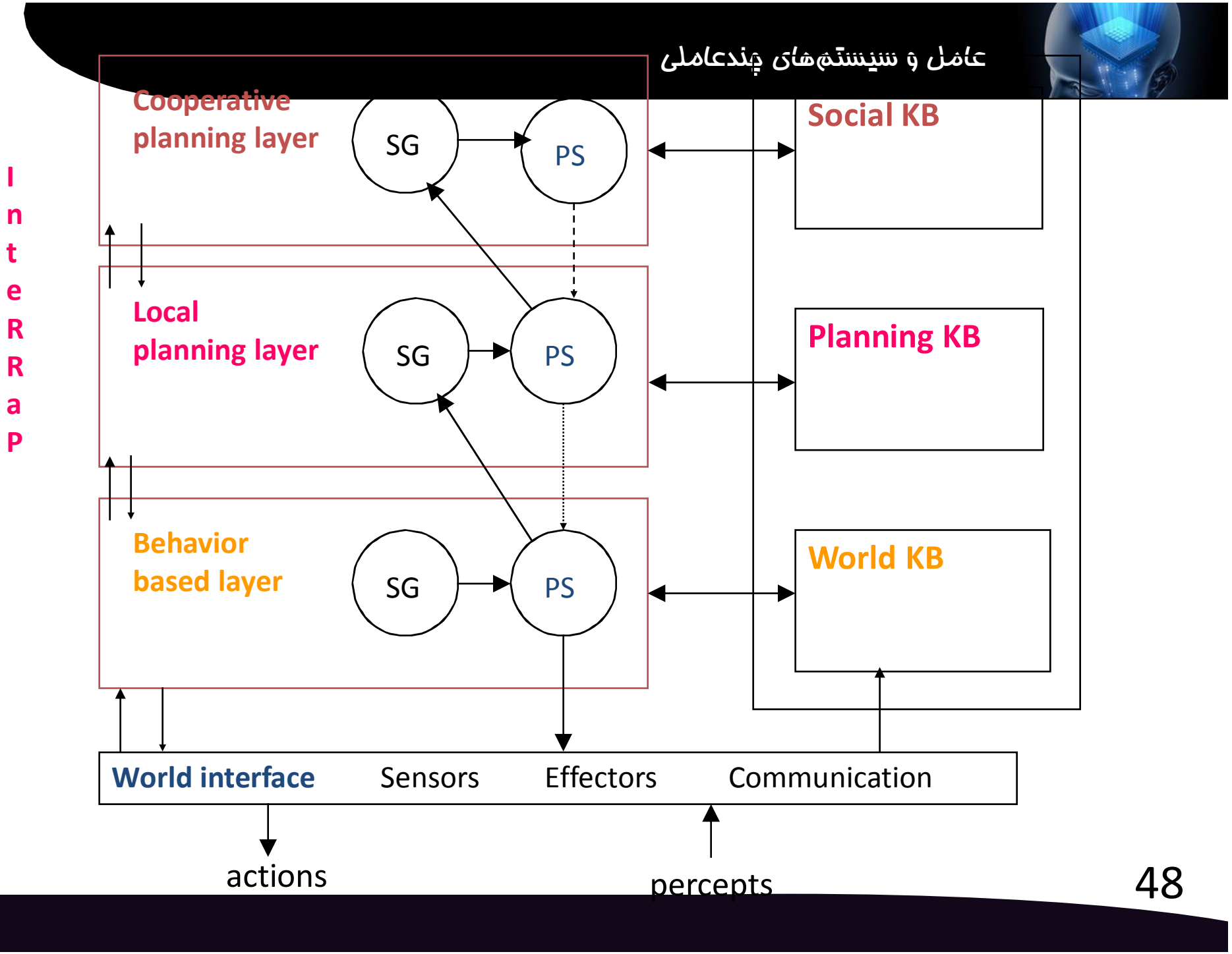


# ماشین تورینگ

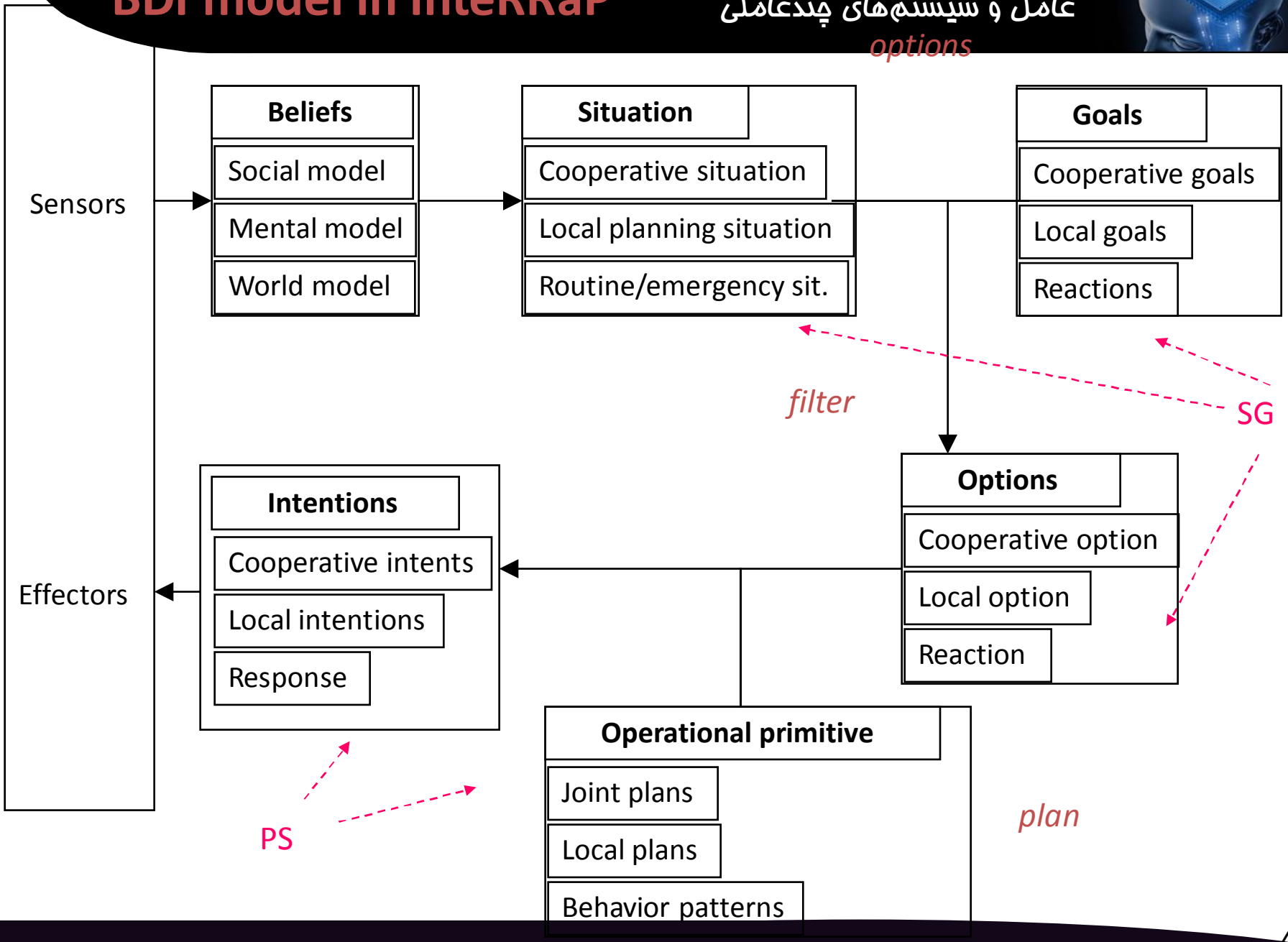
- یک معماری سه لایه افقی که هر لایه یک رفتار را برای اجرا کاندید می کند.
  - لایه واکنش
    - مجموعه ای از قوانین عمل–حالت که نسبت به اداراکات محیط واکنش نشان می دهند.
  - لایه برنامه ریزی
    - رفتارهای پرواکتیو
    - استفاده از کتابخانه ای از طرحها که تحت عنوان اسکیما شناخته می شوند. (schema)
    - طرحهای سلسله مراتبی در این لایه اصلاح و بازبینی می شوند.
  - لایه مدلسازی
    - توصیف محیط، خود عامل و سایر عاملها
    - تنظیم اهداف و پیش بینی تضادها در این لایه انجام می شود.
    - تهیه، تولید یا تحویل اهداف به لایه برنامه برزی
- ساختار کنترلی
  - به صورت متمرک است که مجموعه ای از قوانین کنترلی را در خود جای داده است.
  - قوانین وظیفه تحلیل اطلاعات لایه پایین تر و تولید کنترل به لایه بالاتر را دارند.
  - مدیریت لایه ها تا تعیین شود که عمل کدام لایه باید انجام شود.

# InteRRaP

- معماری دو ستونی عمودی است
  - ستون مربوط به کنترل و پایگاه دانش
- بر مبنای BDI بر روی کنترل دینامیک عامل تمرکز دارد.
- مبانی طراحی
  - معماری سه لایه که توصیفگر درجات مختلفی از فشرده سازی و پیچیدگی عامل ها است.
  - لایه های کنترلی به صورت پایین به بالا کار می کنند. به این صورت که اگر یک لایه از پس کاری برنیاید، انجام آنرا به لایه بالاتر محول می کند.
  - هر لایه از عملیات اولیه لایه پایین تر برای رسیدن به اهدافش استفاده می کند.
  - هر لایه کنترلی از دو ماژول بهره می برد.
    - ماژول شناسایی وضعیت/فعالسازی هدف (SG)
    - ماژول برنامه ریزی/زمانبندی


InterRRaP
Cooperative planning layer
Local planning layer
Behavior based layer
SG
PS
SG
PS
SG
PS
Social KB
Planning KB
World KB
World interface
Sensors
Effectors
Communication
actions
percepts

# BDI model in InteRRaP



# پایان

- مدل عامل ساده
- معماری عامل شناختی
- معماری عامل واکنشی
- معماری چندلایه