

# Multiagentové systémy

**Dr. Andrej Lúčný**

**KAI FMFI UK,**

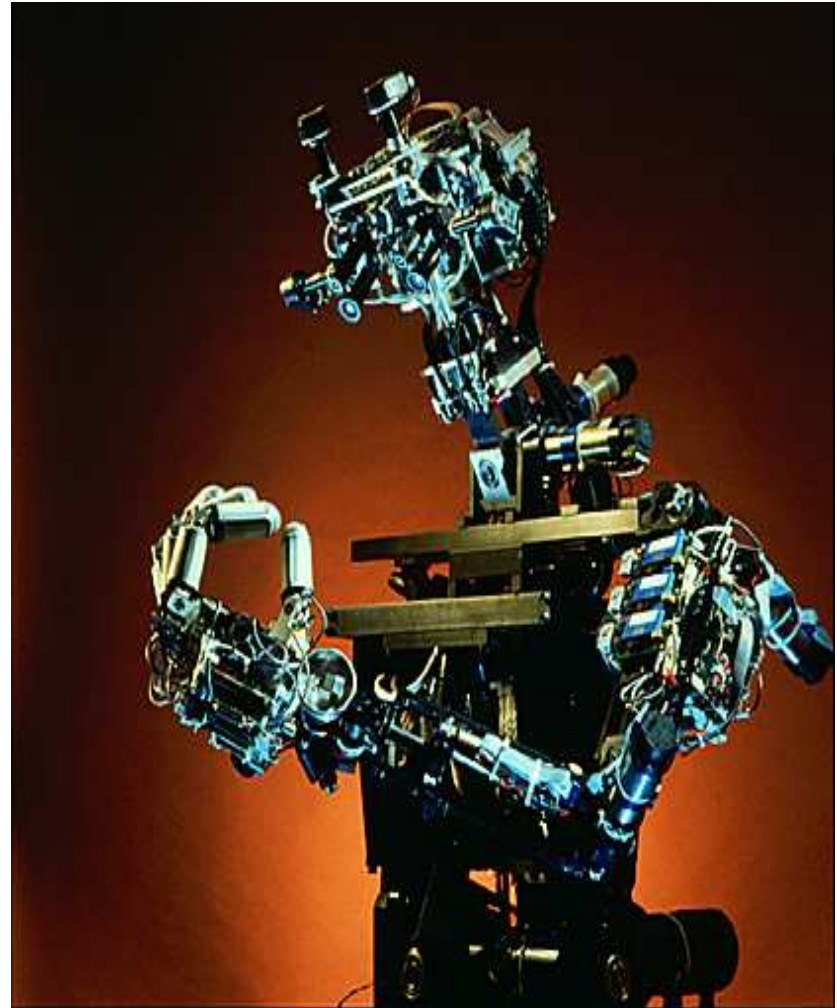
**andy@microstep-mis.com**

**<http://www.microstep-mis.sk/~andy>**

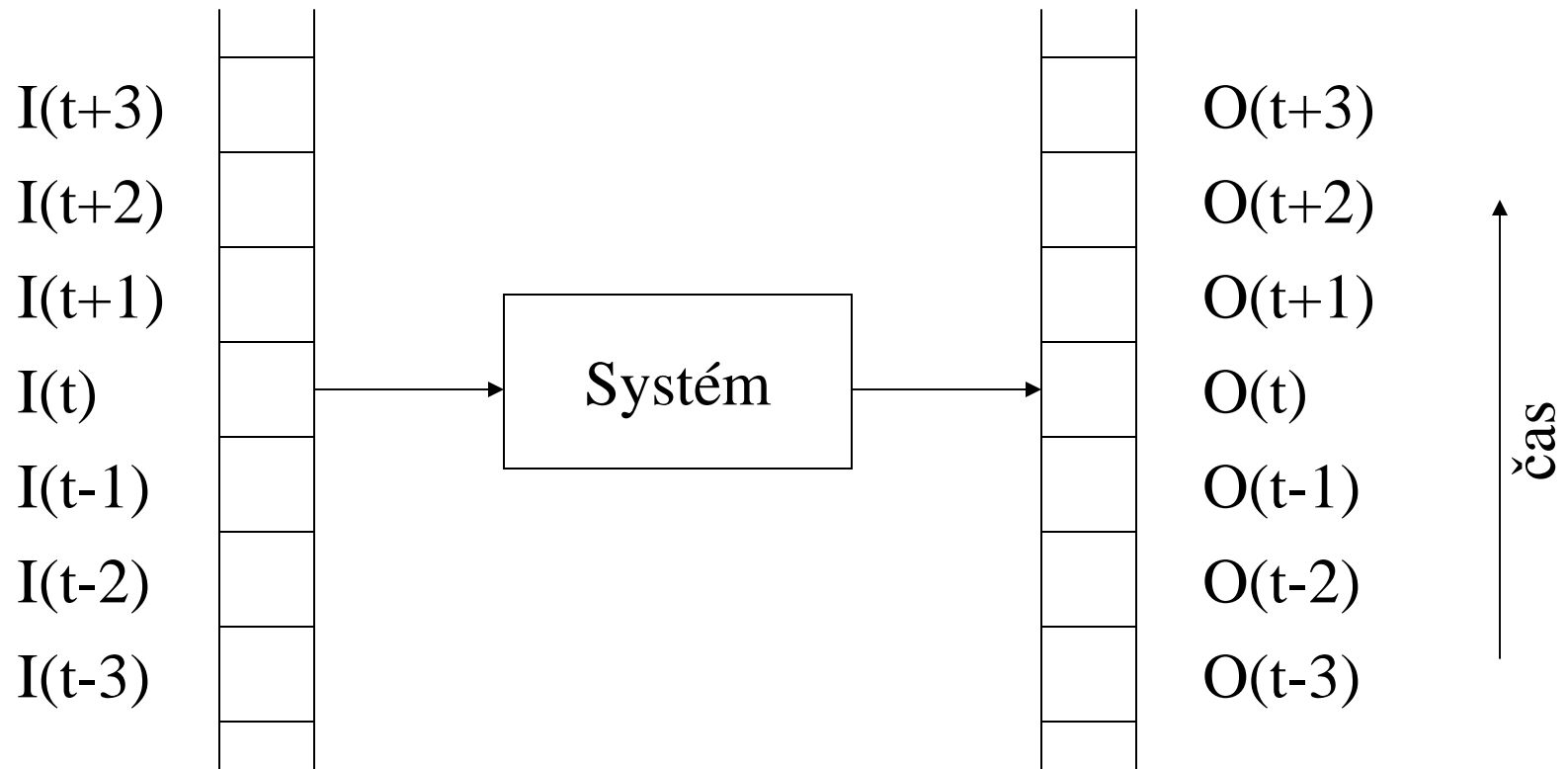
## Opakovanie

- ako funguje master – slave a čo ním riešime ?
  - ako funguje prievozník ?
  - aké primitívy SRR volá agent ?
  - aké primitívy SRR volá space ?
  - čo je implicitné vzorkovanie ?
- 
- prečo deti stále baví hrať sa s vodou a pieskom ?

# Multiagentový prístup k riadeniu



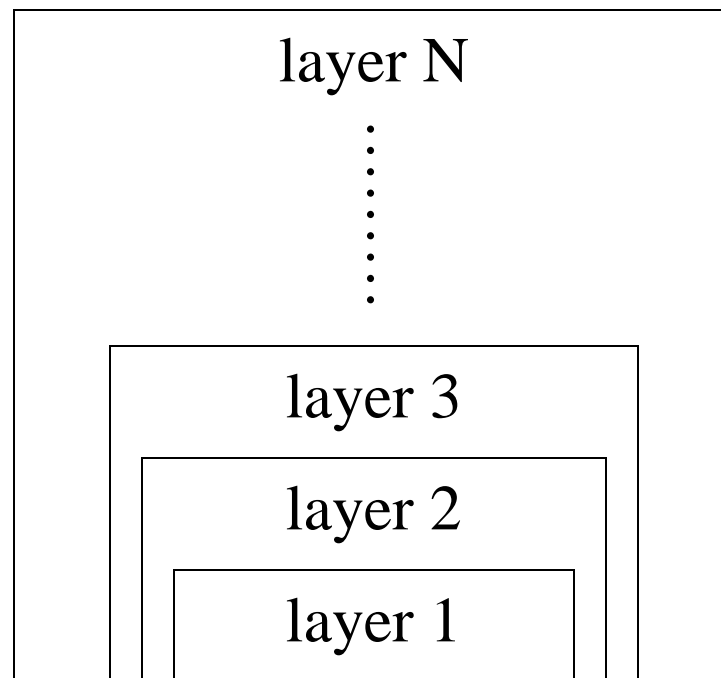
# Riadenie



# Subsumpčná architektúra

- Je architektúrou autonómneho systému, spravidla mobilného robota
- zjednodušene napodobňuje evolúciu v prírode tým, že nová verzia zahrňuje celú verziu starú (subsume = zahrňovať).
- jednotlivé evolučné kroky však vykonáva vývojár
- Vychádza z princípov tzv. kambrickej alebo „novej“ umelej inteligencie

# Biologická motivácia



# Kambrická inteligencia

## Základné pojmy

- Situovanosť
- Stelesnenosť
- Emergencia
- Interakcia
- Hierarchia (Inkrementálnosť)

# Situovanosť

- Pri tvorbe neuvažujeme, že sa vytváraný systém nachádza v akýchkoľvek podmienkach vyjadrených všeobecne, ale uvažujeme veľký súbor špecifických podmienok
- Môžeme najprv urobiť verziu, ktorá funguje len pre časť týchto podmienok a neskôr ju rozšíriť na ostatné
- Nebudeme podmienky ošetrovať jediným modulom, ale špecifické podmienky budú ošetrované špecifickými modulmi – „raz zaberie taký trik, inokedy onaký“
- Nesnažíme sa budovať abstraktnú reprezentáciu sveta. „Najlepšou reprezentáciou sveta je svet sám“.
- Systém vyvíjame pre konkrétne prostredie, v ktorom ho testujeme (ladíme)

# Stelesnenosť

- Neoddeľujeme návrh od implementácie
- Navrhujeme s prototypom v ruke
- Tým je umožnené po skončení každej fázy vývoja systém testovať a ladit'.
- Pracujeme so skutočnými vstupmi, spoliehame sa na ich reálny charakter.

# Emergencia

- Správanie systému je výsledkom interakcie jednotiek z ktorých sa skladá
- Súvislosť medzi správaním jednotiek a systému môže byť na prvý pohľad nejasná
- Pri vhodnej štruktúre systému môžeme explicitnou implementáciou určitých schopností implicitne implementovať schopnosť doteraz neuvažovanú. Vývojár môže byť neschopný tento jav predvídať, hoci jeho dodatočné vysvetlenie by aj nebolo zložité.

# Interakcia

- Systém môže inteligenciu čerpať z dynamiky prostredia. Často možno zložitý riadiaci systém nahradiť jednoduchým, ktorý vhodne stavia na dynamike prostredia
- Paradoxne (na rozdiel od systémov ktoré obsahujú kognitívny modul) v statickom prostredí sa takýto systém správa horšie než v dynamickom.
- Inteligencia sa prejavuje reakciami systému na stav jeho prostredia, teda v globálnom správaní, nemožno ju nájsť vnútri systému
- Reaktivita systému zabezpečuje jeho odolnosť voči rušivým zmenám a prechodným stavom v prostredí

# Hierarchia (Inkrementálnosť)

- Systém vyvíjame inkrementálne, po vrstvách, pričom každá implementuje určitú aktivitu
- Postupujeme pritom zdola nahor. Začínáme s hierarchicky nižšími vrstvami (biologická metafora: historicky staršími), teda s primitívnejšími aktivitami
- Postupne pridávame nové a nové vrstvy, pričom po každom inkremente, testujeme, ladíme a opravujeme, kým naozaj nezbehnú všetky testy určené pre danú fázu vývoja úspešne

# Hierarchia (Inkrementálnosť)

- Vďaka tomu sa nám nemôže stať že implementácia zlyhá na integrácii častí v celok. Hoci na druhej strane nám potenciálne hrozí, že systém dostaneme do stavu, že novú aktivitu už nemožno konzistentne pridať
- Hierarchia systému spočíva v tom, že vyššie vrstvy môžu využívať nižšie. Môžu ich pasívne monitorovať alebo aj aktívne na ne pôsobiť tým, že potláčajú alebo dokonca modifikujú ich činnosť

# Subsumpcia

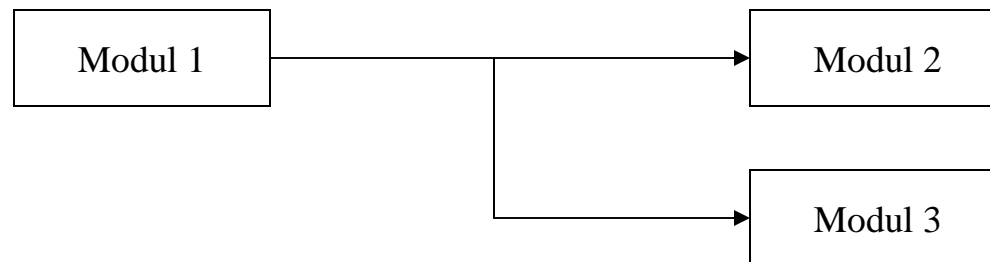
- Ako však môže hierarchicky nižšia – teda neskoršia - vrstva pôsobiť na hierarchicky vyššiu – teda skoršiu ?
- Ved' pri implementácii tej nižšej sme takúto možnosť neuvažovali a teda sme nevybudovali žiadne rozhranie, ktorým by mohla vyššia vrstva na nižšiu vplývať

# Subsumpcia

- Riešenie: systém musí mať takú modularitu, pri ktorej je toto rozhranie implicitne prítomné

# Subsumpčná architektúra

- Systém sa skladá z autonómnych modulov (Augmented Finite State Automata)
- Tie sú prepojené komunikačným vedením, po ktorom sú prenášané správy

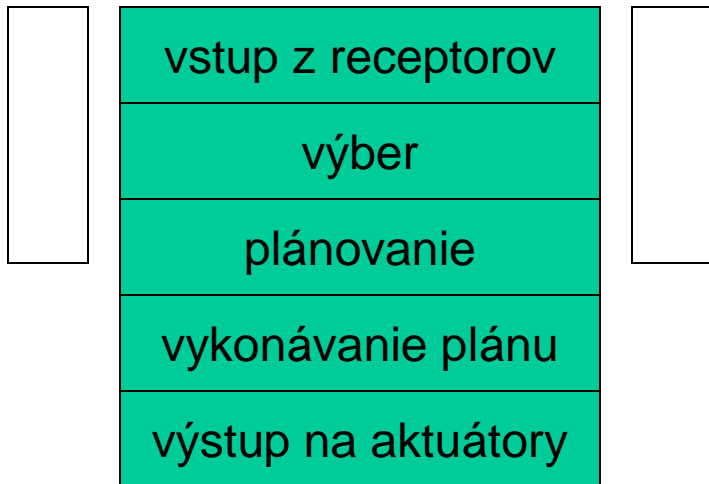


# Dekompozícia

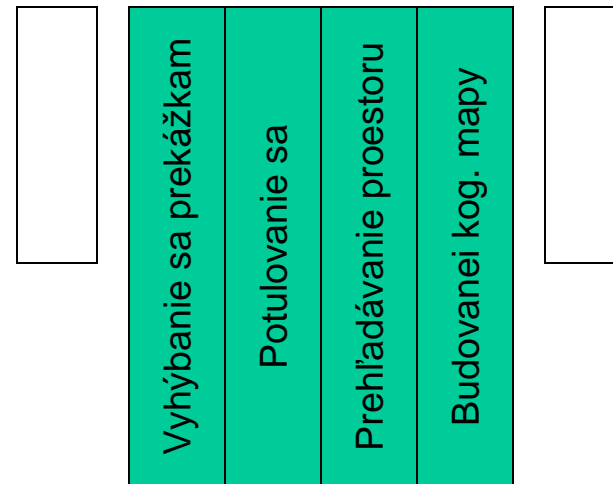
- Funkciou = spolu (z hľadiska hierarchických vrstiev) sú kódy realizujúce jednu funkciu, napr. spracovanie obrazu
- Aktivitou = spolu sú kódy realizujúce jednu aktivitu, napr. vyhýbanie sa prekážkam

Subsumčná architektúra je založená na dekompozícii aktivitou

funkciou  
horizontálna



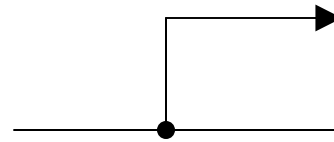
aktivitou  
vertikálna



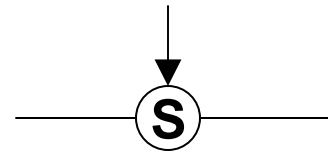
# Subsumpčná architektúra

- Vyššia vrstva môže s nižšou manipulovať pomocou:

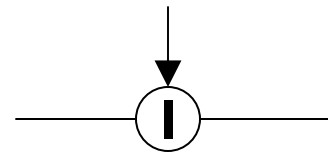
- odpočúvania



- supresie

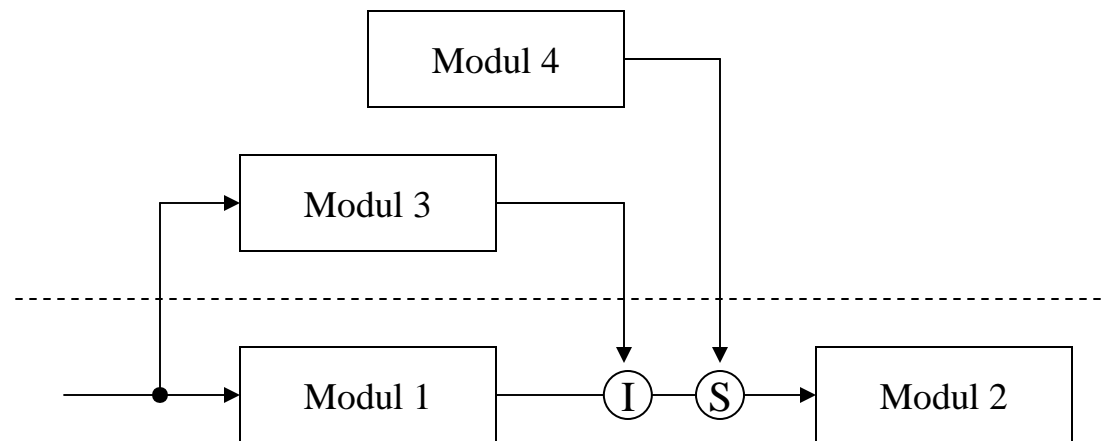


- inhibície

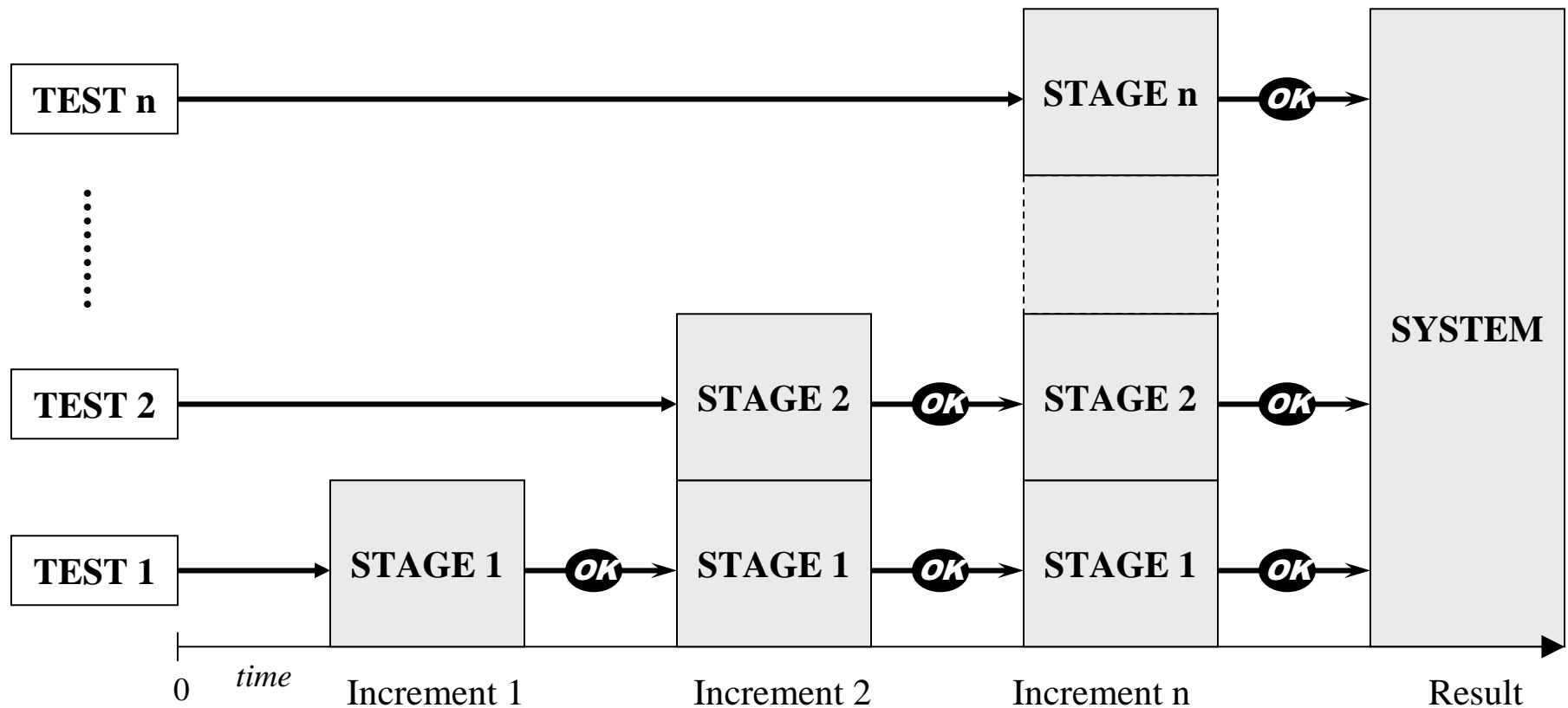


# Subsumpčná architektúra

- Pri pridaní novej vrstvy použijeme tieto mechanizmy na zavedenie kooperácie



# Inkrementálne, zdola - nahor



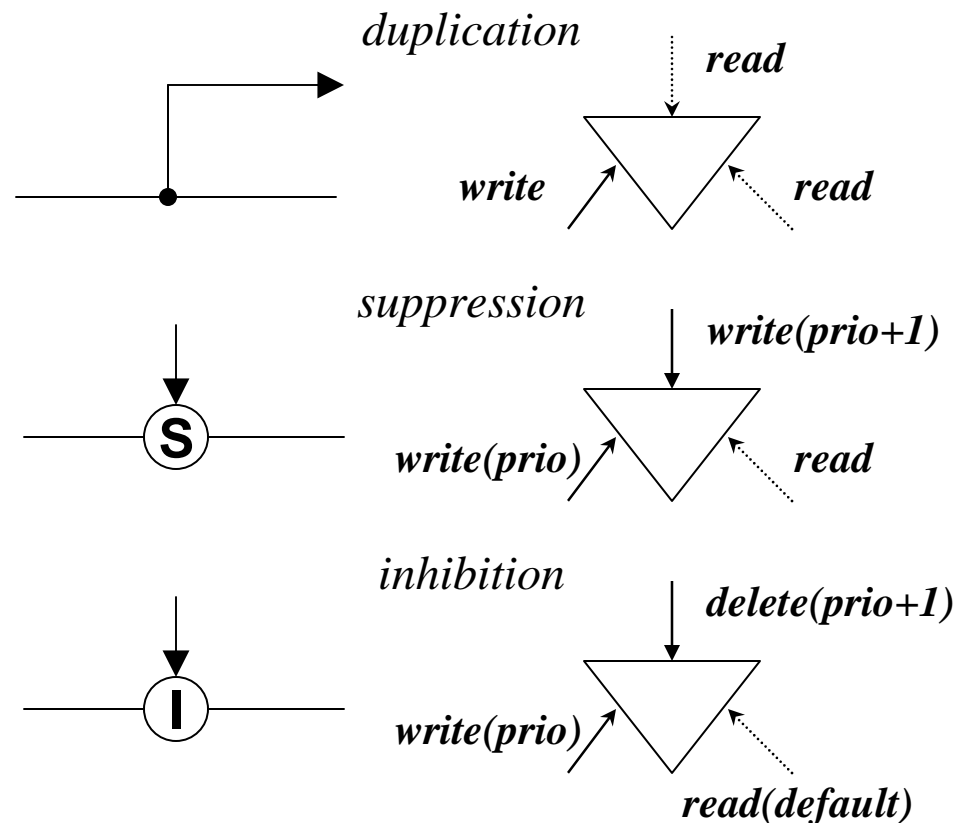
# Príklad:

- *For example, navigation of two wheeled robot in bureau can be developed by subsumption in the following way: We start with robot which just goes forward. Then we add a layer which recognizes obstacles and while they are detected, the layer replaces messages for one wheel to backward. As a result, the robot does not collide, but easily it can happen that it stays in the same region, moving in a cycle. Thus we add a layer which sometimes causes its random turn. However we perform such a turn only when no obstacles are detected and we implement it just by apparent detection of obstacles. Further we add another layer which provides an active search for suitable absolute directions for movement to another part of bureau. Once such direction is chosen, we implement its following by turns which are apparently random for the older layers, but in fact they keep the robot at the chosen trajectory. Other level can detects landmarks and having received a goal from user it can navigate to one of them by emulation of the chosen direction in the older level*



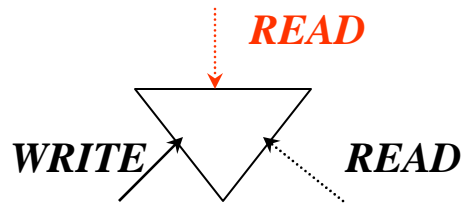
# Realizácia v Agent - Space

- Riešenie vyjadrené v subsumpčnej architektúre možno poľahky transformovať do agent-space



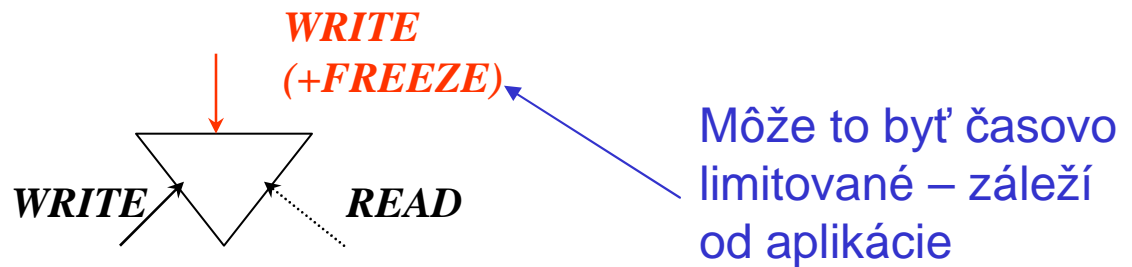
# Odpočúvanie

- Vyššia vrstva nedeštruktívne odoberá údaje z nižšej, sleduje čo sa tam deje



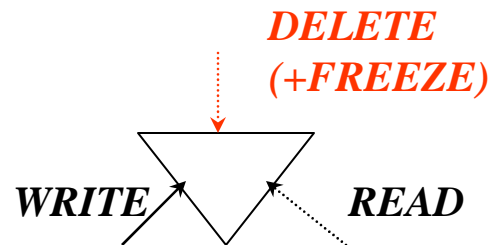
# Supresia

- Vyššia vrstva nahradí údaj plynúci v nižšej vlastnou hodnotou



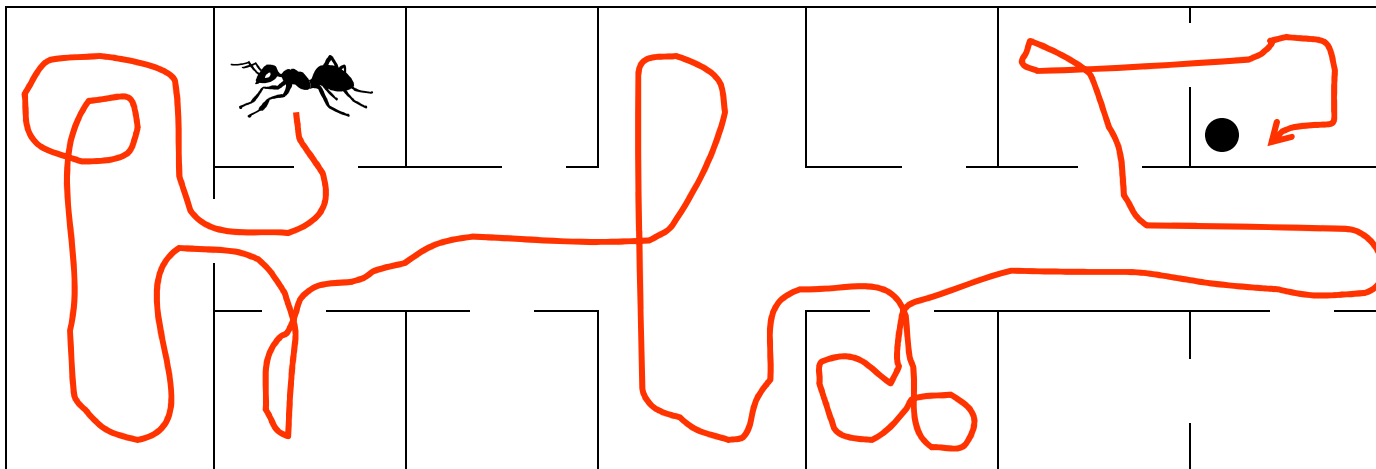
# Inhibícia

- Vyššia vrstva zmaže údaj v nižšej vrstve .  
Zastaví tam dátový tok

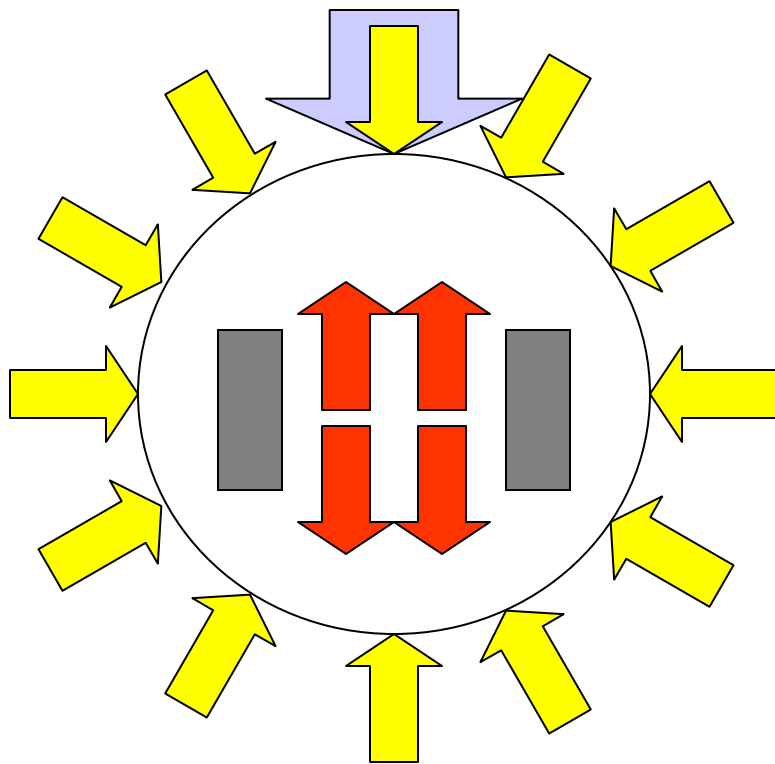


# Riadiaci systém robota

**Úloha: urobiť robota, ktorý dokáže nájsť na zem umiestnenú kovovú guľičku niekde na 4 poschodí bloku D.**



# Senzory a aktuátory



Sonary na detekciu  
najbližej prekážky

Kolesá na pohyb  
vpred, vzad a otáčanie

Detektor guľičky

# Návrh systému

Podľa princípu subsumpcie

---

**STOP – Zastavenie pri guľčke**

---

**EXPLORE – Prehľadávanie priestoru**

---

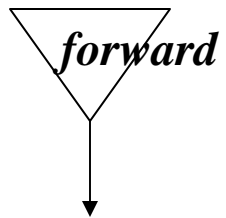
**WANDER - Potulovanie sa**

---

**AVOID - Vyhýbanie sa prekážkam**

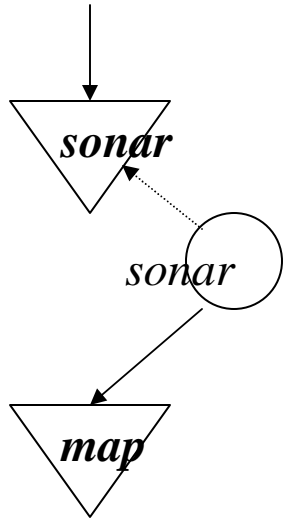
---

# AVOID

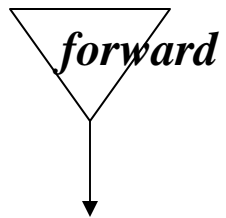
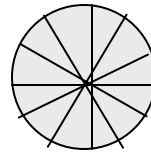


Za normálních okolností, ideme  
stále vpřed

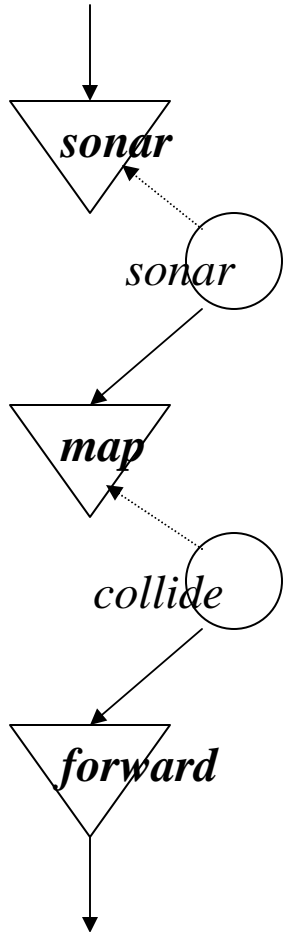
# AVOID



Sonar zostaví pole vzdialeností najbližších prekážok v rôznych absolútnych výsečiach



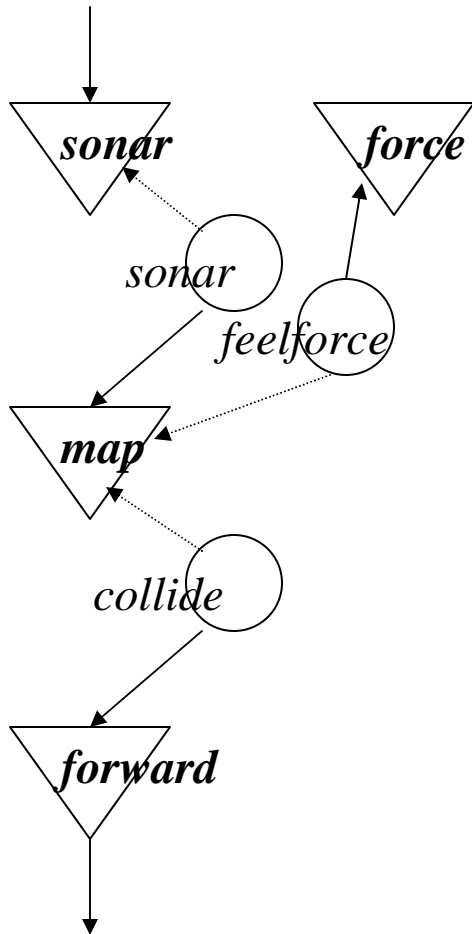
# AVOID



Ked hrozí náraz, collide zastaví forward. Zastaví ho na základe údajov z dopredného sonaru

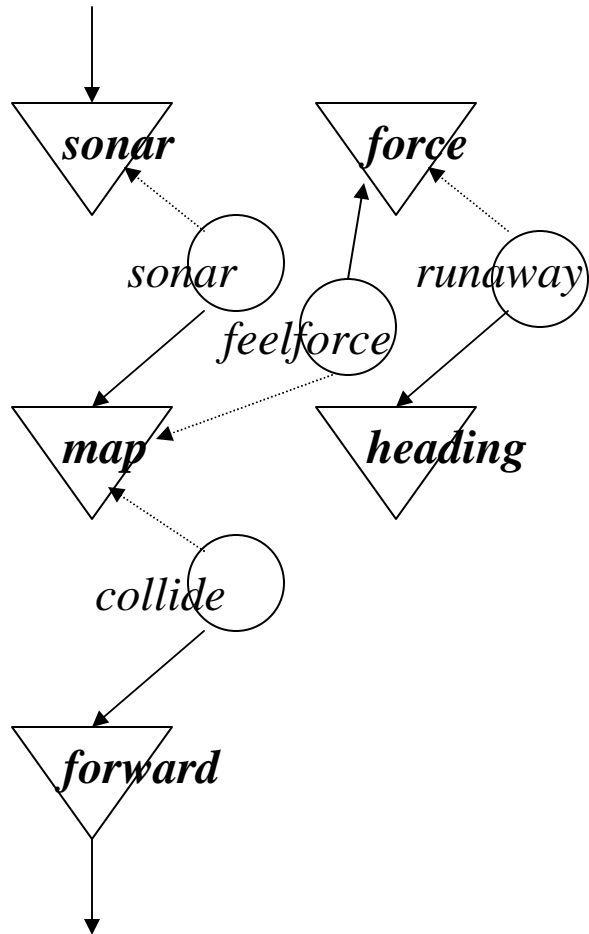
Tým pádom ideme rovno až kým nehrozí náraz, potom zastavíme

# AVOID



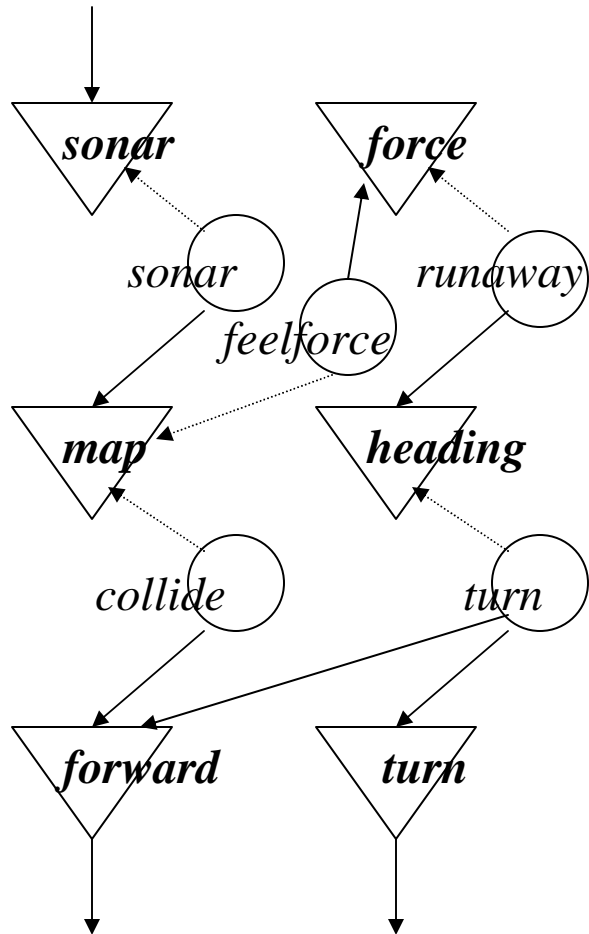
Feelforce vyhodnocuje smer v ktorom najviac hrozí zrážka

# AVOID



Runaway zavelí uberať sa opačným smerom

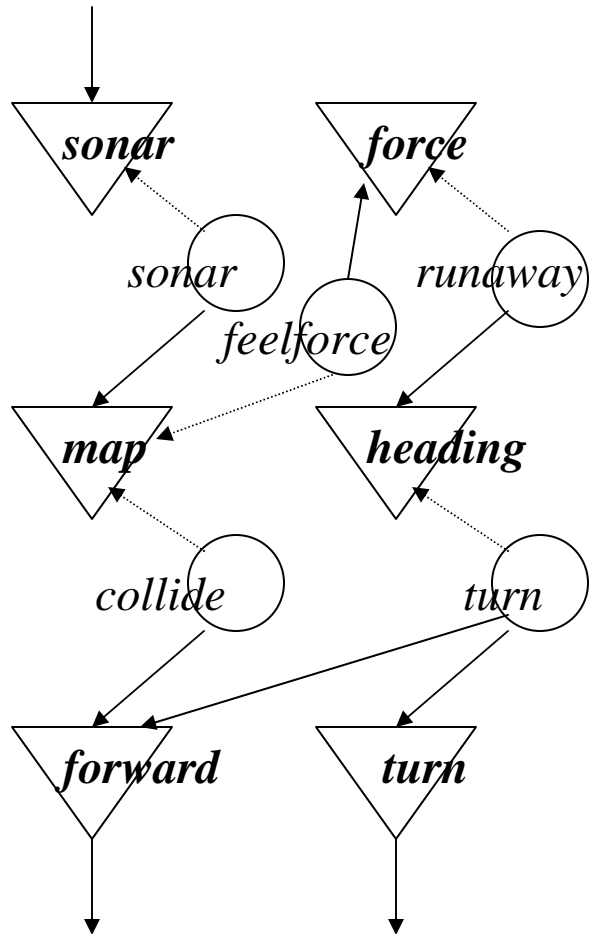
# AVOID



A turn podľa toho smeru riadi  
otáčanie kolies, prípadne aj cúvanie



# AVOID

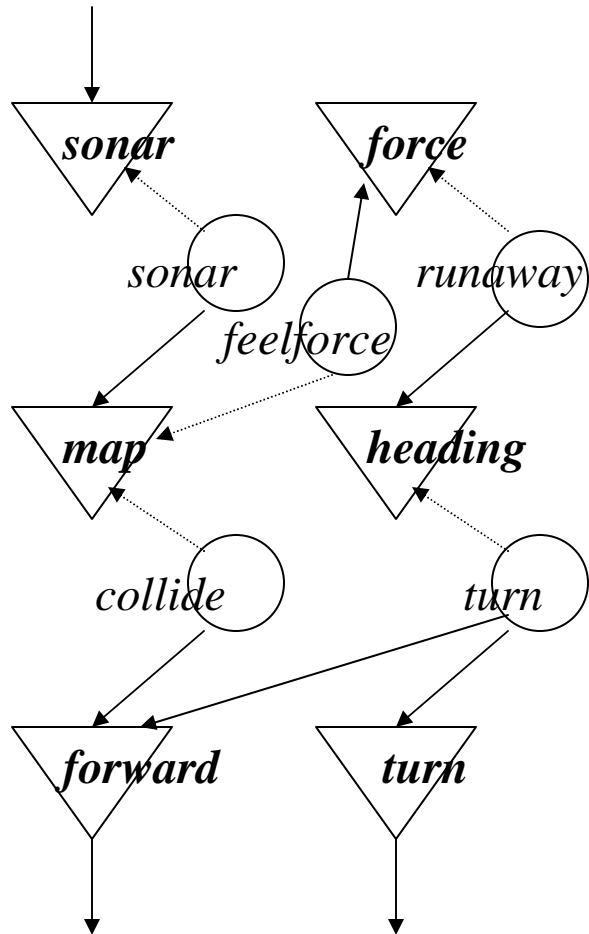


Tým pádom pojdeme rovno až kým nehrozí náraz. Potom sa začneme vyhýbať, a potom sa opäť pohybujeme rovno.

Pri tomto pohybe sa pomerne ľahko dostaneme do nejakého cyklu, ale na vrstvu AVOID to stačí.

AVOID je hotová.

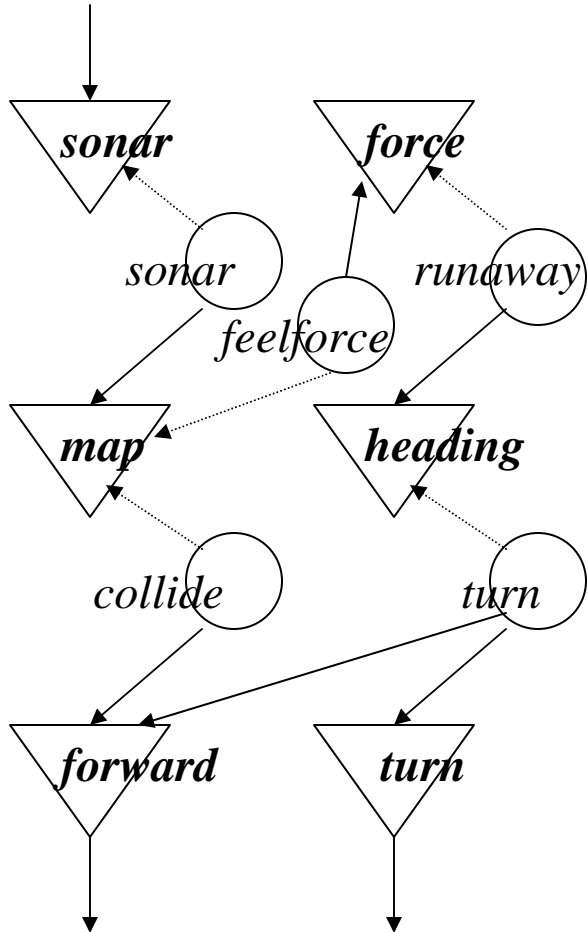
# AVOID



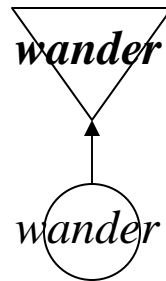
Všimnime si, že realizáciou vyhýbania sa statickým prekážkam, sme zároveň realizovali vyhýbanie sa pohybujúcim objektom, ktoré nebezpečne križujú smer nášho pohybu.

Podobné prípady sa niekedy označujú ako tzv. emergencia.

# AVOID



# WAN DER

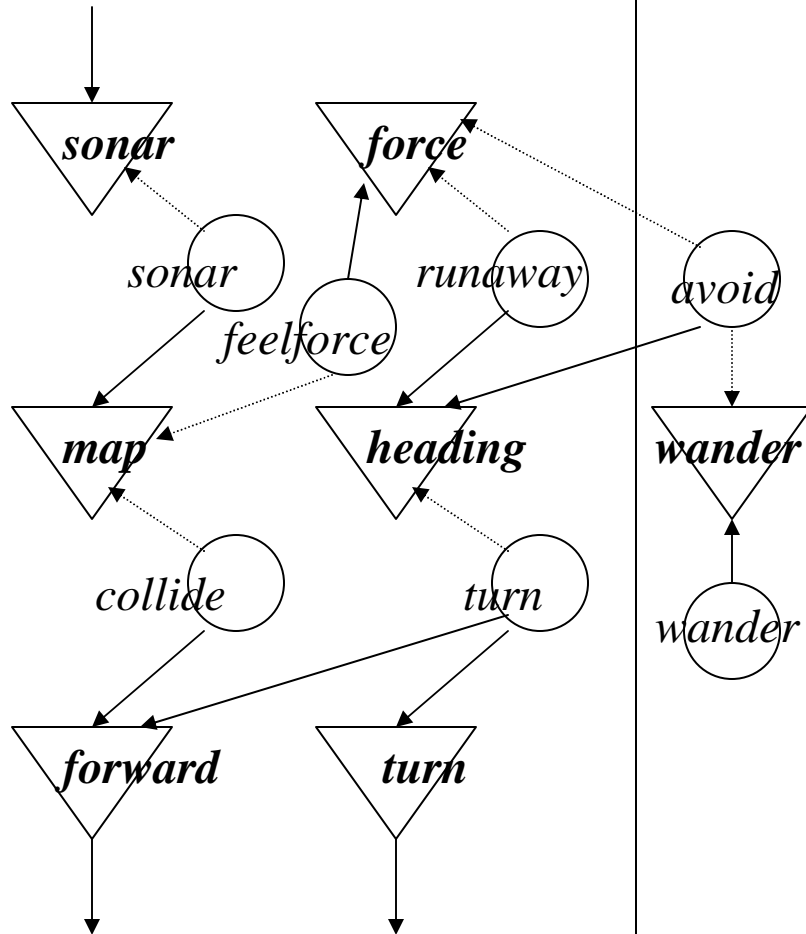


Teraz by sme chceli aby  
sem-tam porušil  
prípadné cyklenie  
náhodným pohybom

Preto agent wander s  
veľkým sleepom sem-  
navrhne určitý odklon  
smeru pohybu

# AVOID

# WAN DER

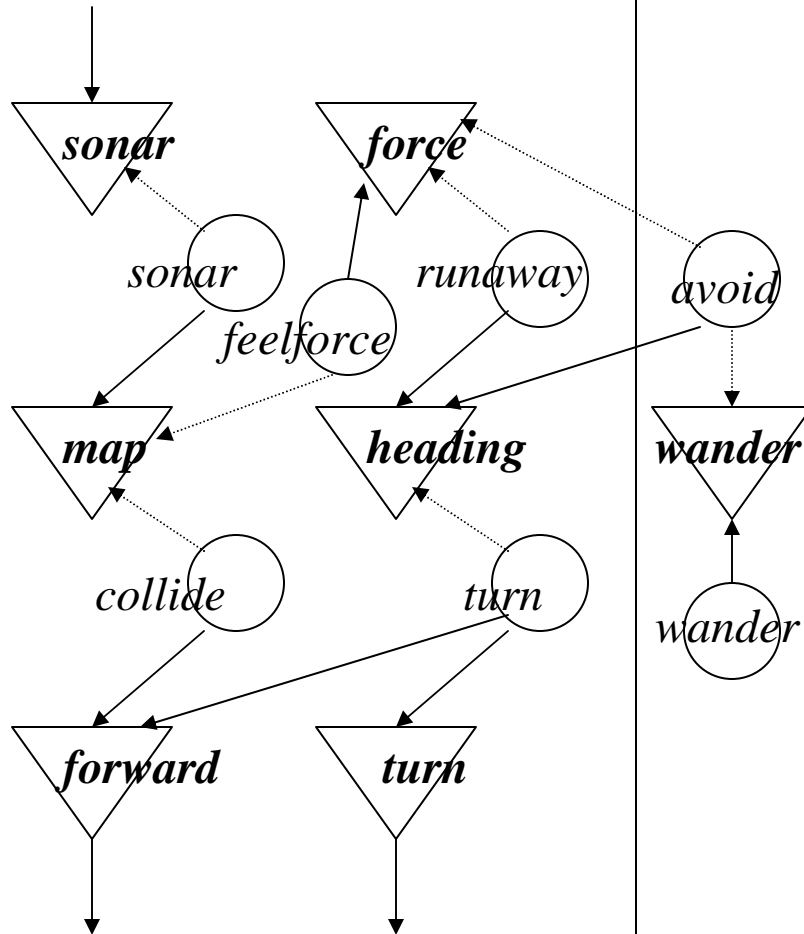


A agent avoid, ak sa cez force presvedčí, že momentálne nehrozí zrážka, prepíše navrhnutým smerom heading, tak ako keby nejaká zrážka hrozila.

Tým pádom využije vyhýbanie sa prekážkam na potulovanie sa

# AVOID

# WAN DER



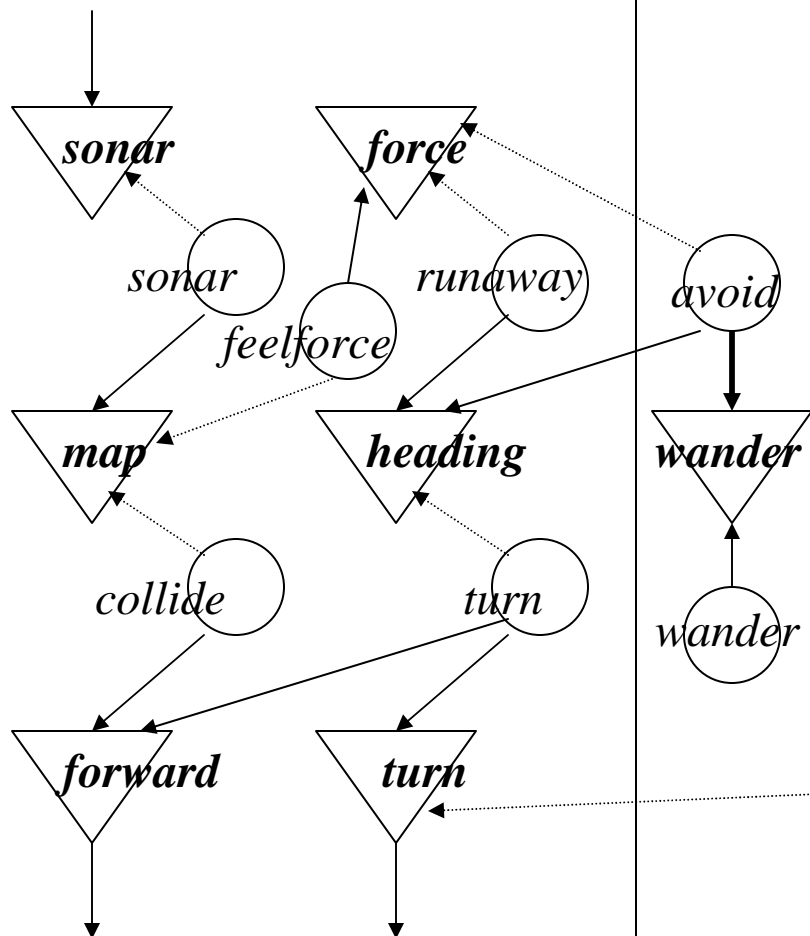
Tým pádom sa nielen vyhýbame prekážkam, ale sa aj potulujeme a pohyb nášho robota už nie je predvídateľný.

Ale zatiaľ sa dosť motá na mieste.

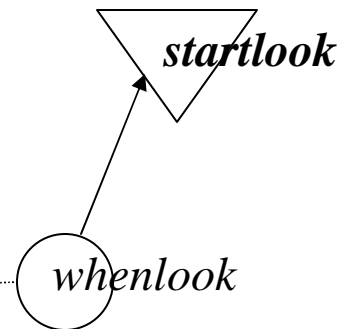
# AVOID

# WAN DER

# EXPLORE



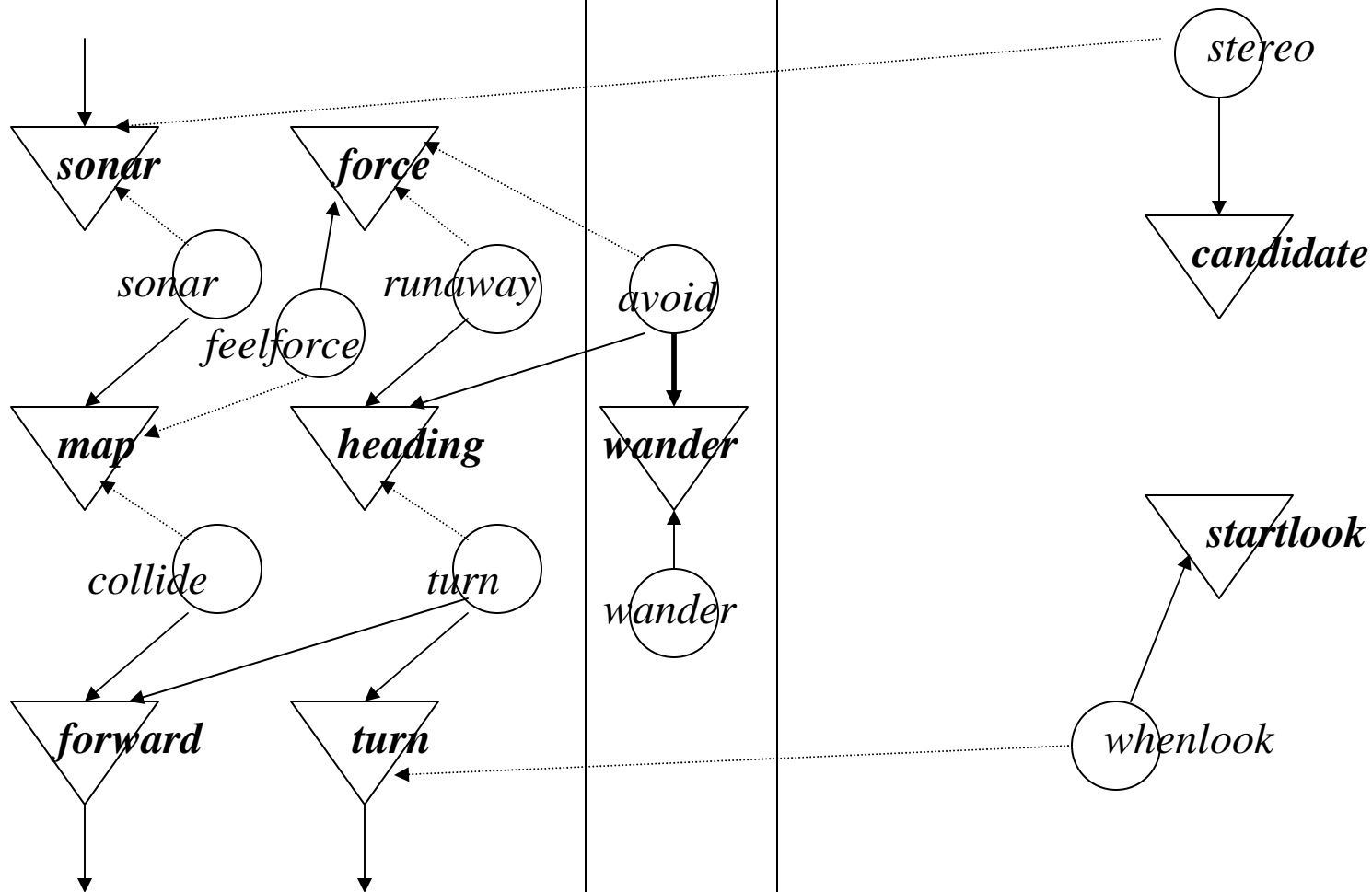
whenlook sleduje či sa nemotáme. Ak usúdi, že už je toho priveľa, dá povel na presun do inej oblasti



# AVOID

# WAN DER

# EXPLORE

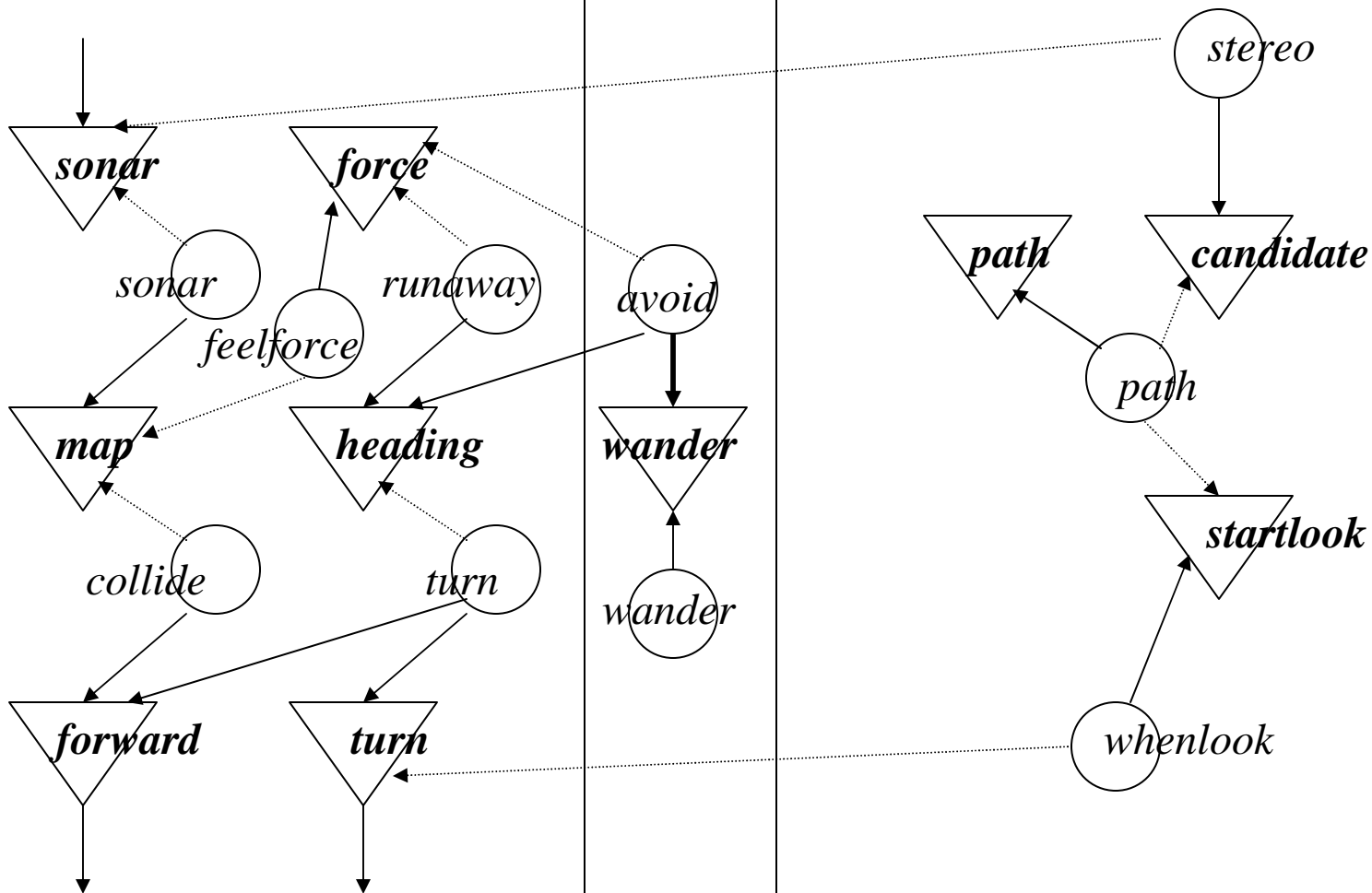


Stereo  
sleduje či  
sa na  
sonare  
objaví  
smer v  
ktorom je  
voľno

# AVOID

# WAN DER

# EXPLORE

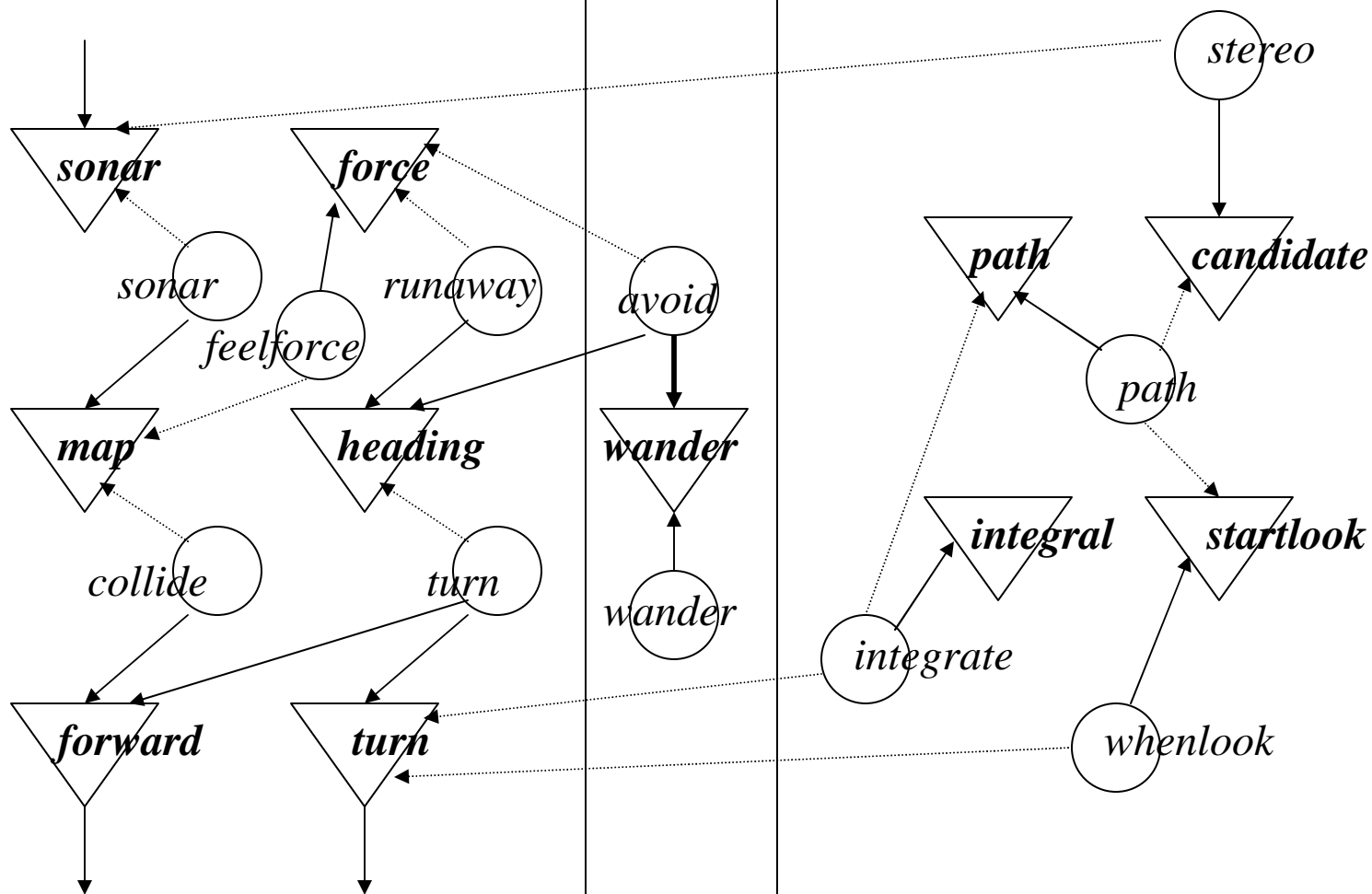


Path  
vytýči  
ktorým  
smerom  
sa  
presunie-  
me do  
inej  
oblasti

# AVOID

# WAN DER

# EXPLORE

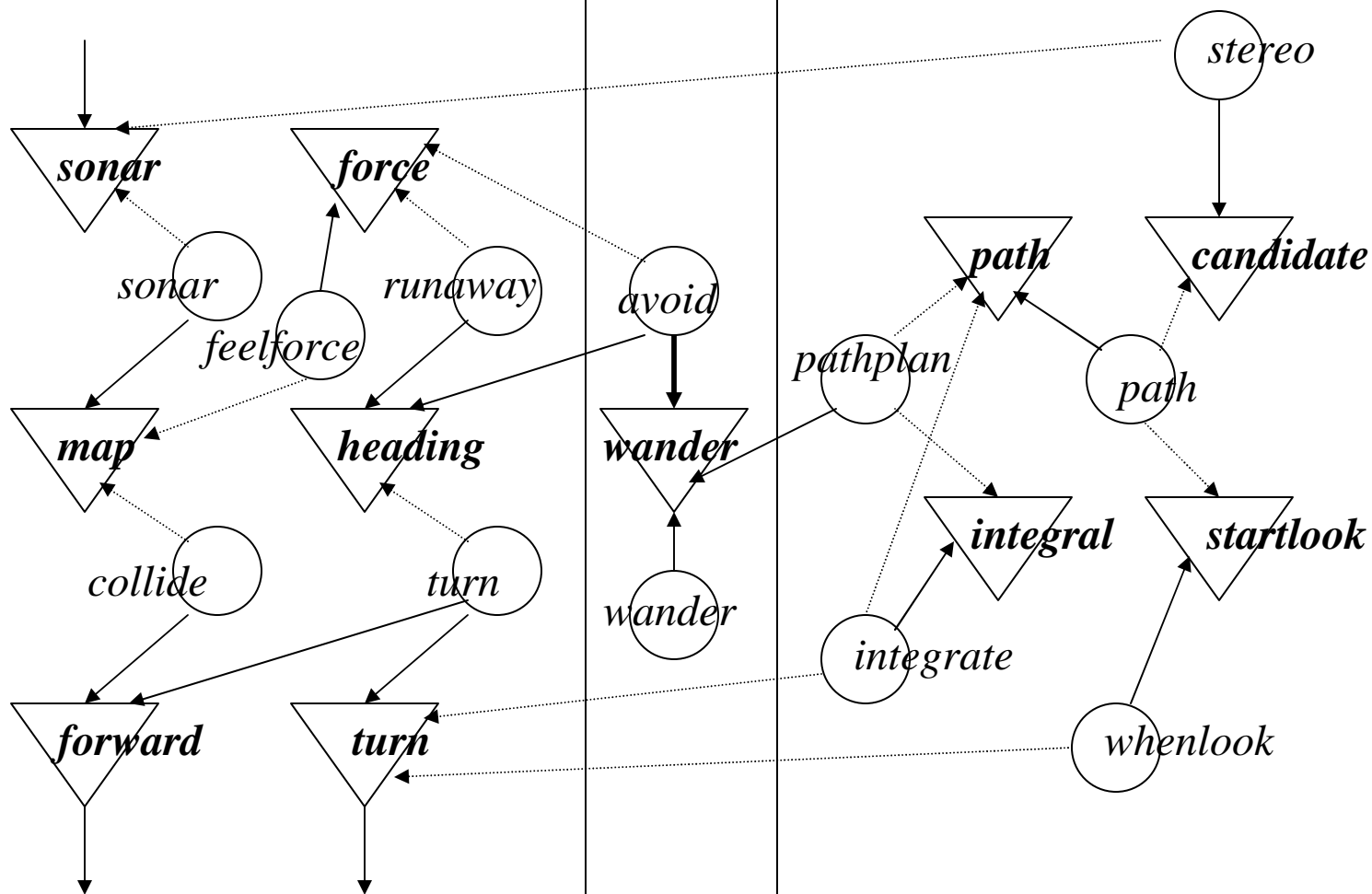


Je to ale  
relatívny  
smer,  
takže na  
to aby  
sme ho  
mohli  
absolutne  
chápať  
potrebujeme  
sumárnu  
odchýlku

# AVOID

# WAN DER

# EXPLORE

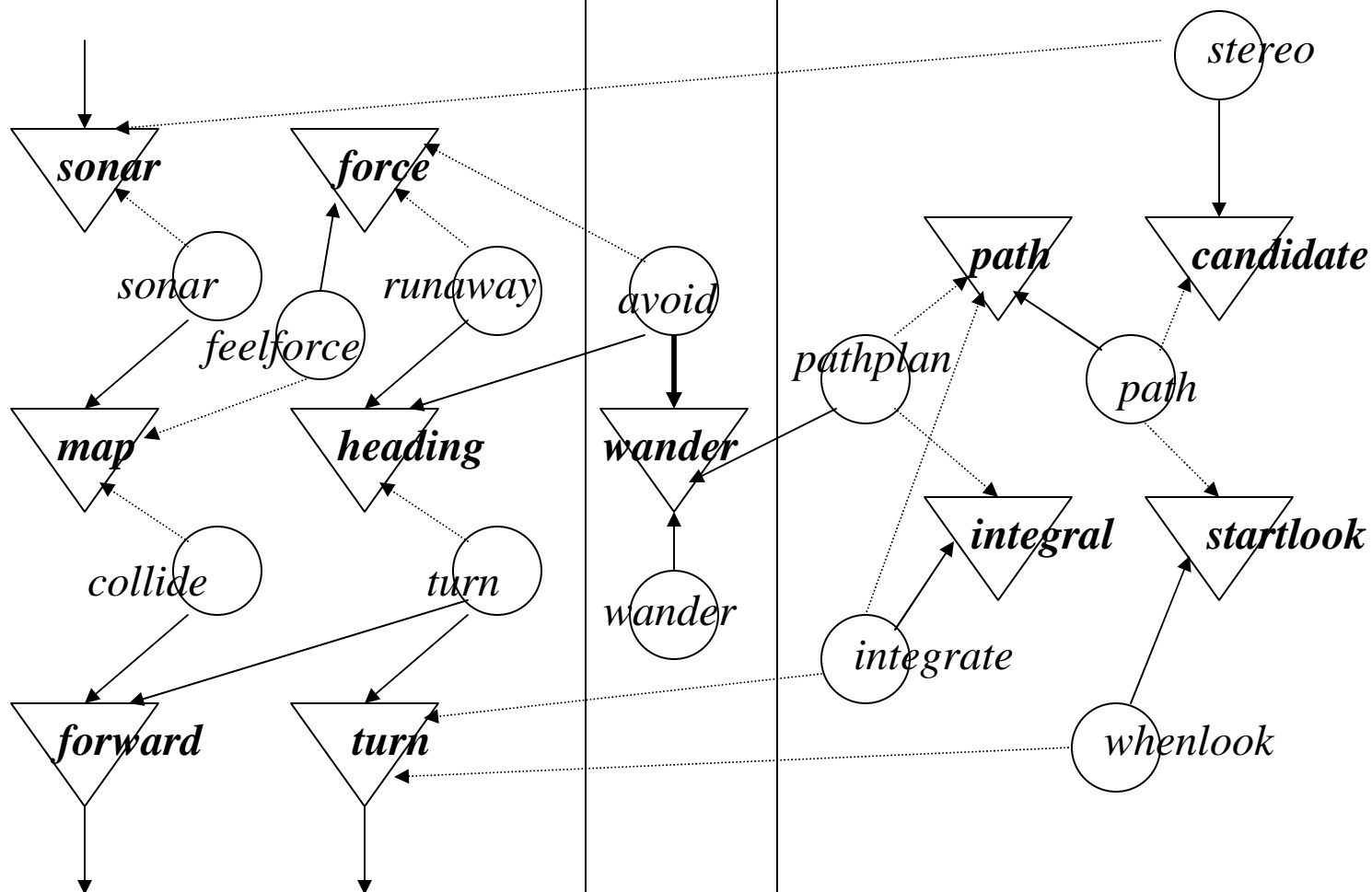


Pathplan  
v každej  
chvíli vie  
o aký  
uhol sa  
treba  
odchýli  
aby sme  
dodržali  
zvolený  
uhol na  
presun

# AVOID

# WAN DER

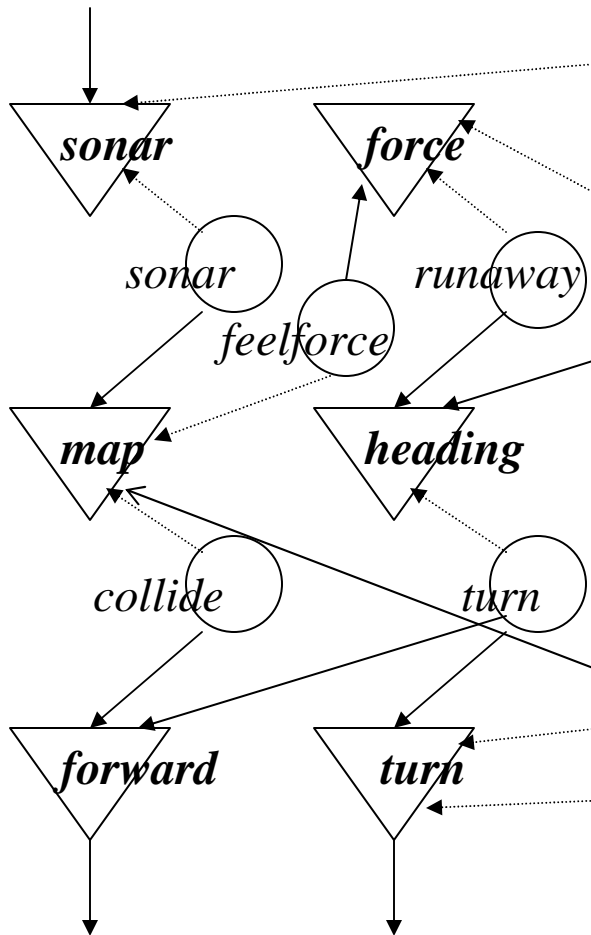
# EXPLORE



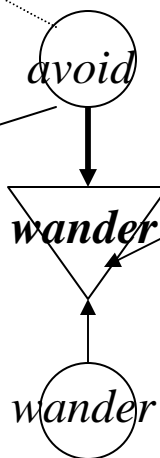
Už vieme  
prechádzať  
priestorom.

Hoci  
neuplatňu-  
jeme žiadne  
logicky  
správne  
prehľadáva-  
nie.

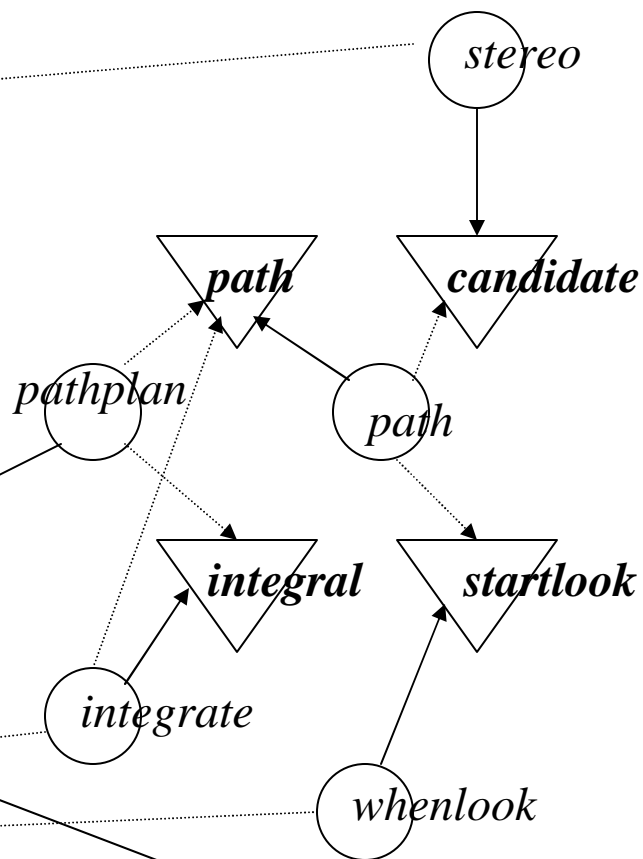
# AVOID



# WAN DER



# EXPLORE



# STOP

Keď  
nájdeme  
guličku,  
tvárimo sa,  
že všade  
vidíme  
prekážky

