Intelligent Control
and Cognitive Systems
brings you...

Multiple, Conflicting Goals
Intro to Game AI

Joanna J. Bryson
University of Bath, United Kingdom

Outline

- How Game AI Is Hard in Special Ways
- How Game AI Approach Reality: Multiple Conflicting Goals
- One possible solution in detail

Introduction to Game AI

...an Example of Multiple, Conflicting Goals

- Make something smart and fun to interact with.
- Don't have it win (or lose) all the time.
- Don't use any CPU

Game AI – Problems

- Solving AI is hard.
- Game worlds are a total big fake.
  - No resources really needed:
    - space, time, energy, sleep, emotions, collisions all need to be faked…
    - probably not by simulation,
- …and Not using any CPU.

Collision Detection

- In the real world, can simulate very animal-like behaviour with very unanimal like componenents.
- e.g. one-legged “kangaroo”

because Laws of Physics

Christer Ericson
Sony Computer Entertainment

Slides @ http://realtimecollisiondetection.net/pubs/

THE PROBLEM

...of BEHAVIOUR in VIRTUAL REALTY is...

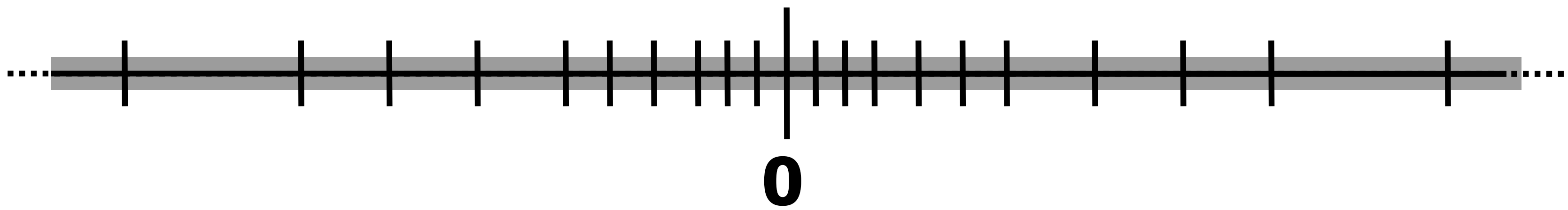
Floating-point arithmetic

Floating-point numbers

- Real numbers must be approximated
  - Floating-point numbers
  - Fixed-point numbers (integers)
  - Rational numbers
    - Homogeneous representation
- If we could work in real arithmetic, I wouldn't be having this talk!

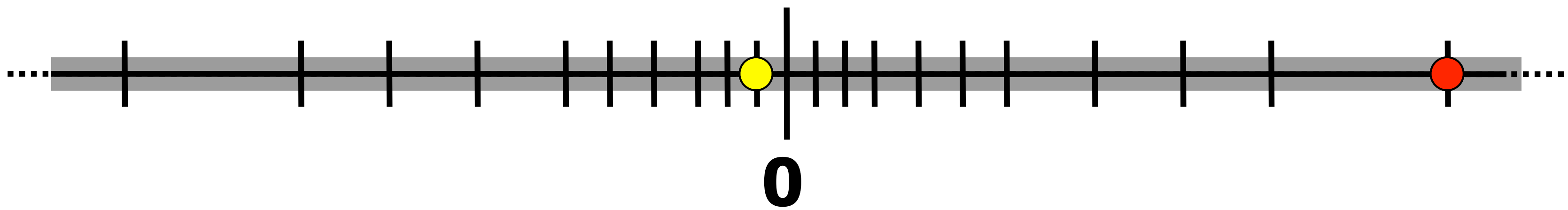
Floating-point numbers

- Irregular number line
  - Spacing increases the farther away from zero a number is located
  - Number range for exponent **k+1** has twice the spacing of the one for exponent **k**
  - Equally many representable numbers from one exponent to another



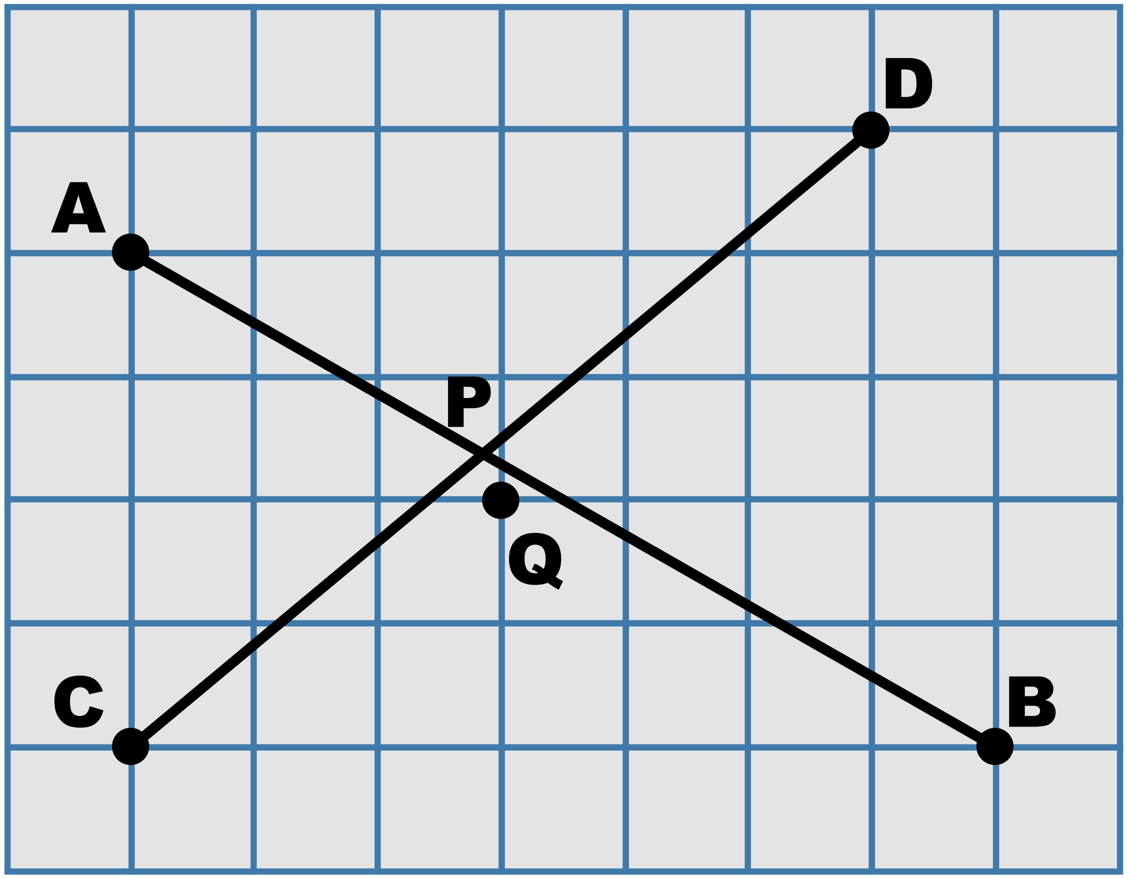
Floating-point numbers

- Consequence of irregular spacing:
  - $-10^{20} + (10^{20} + 1) = 0$
  - $(-10^{20} + 10^{20}) + 1 = 1$
- Thus, not associative (in general):
  - (a + b) + c != a + (b + c)
- Source of endless errors!



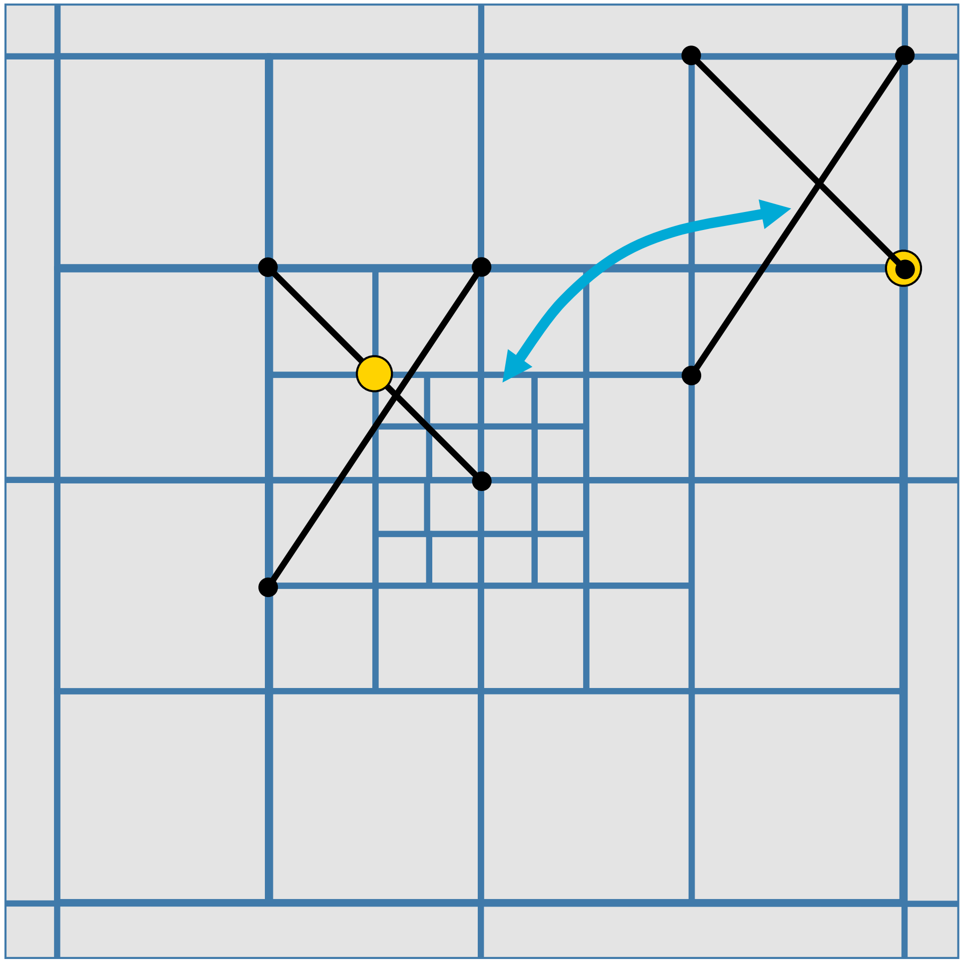
Floating-point numbers

- All discrete representations have non-representable points



The floating-point grid

- In floating-point, behavior changes based on position, due to the irregular spacing!



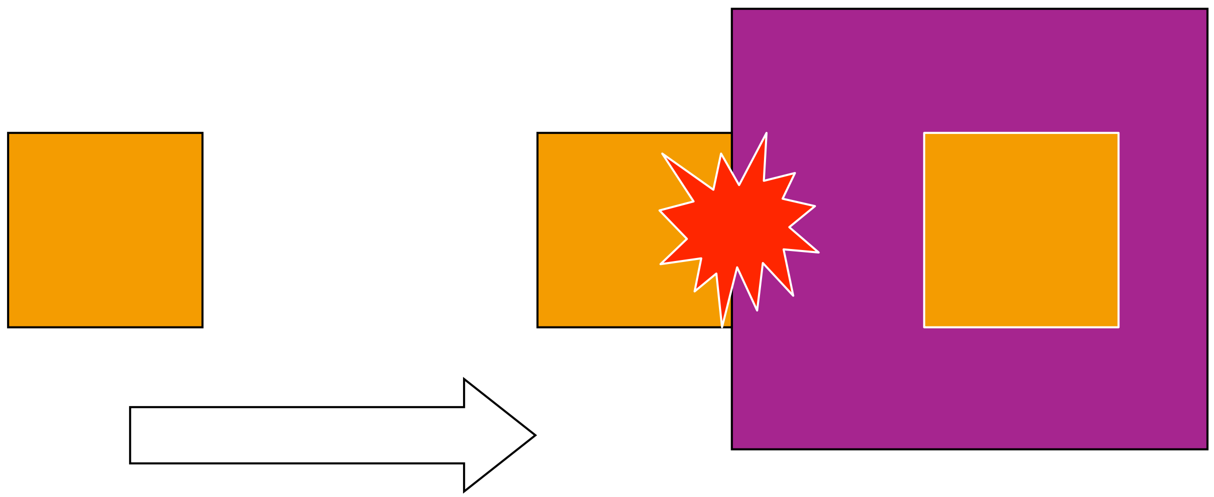
Continuous Collision Detection

David Knott

COMP 259 class presentation

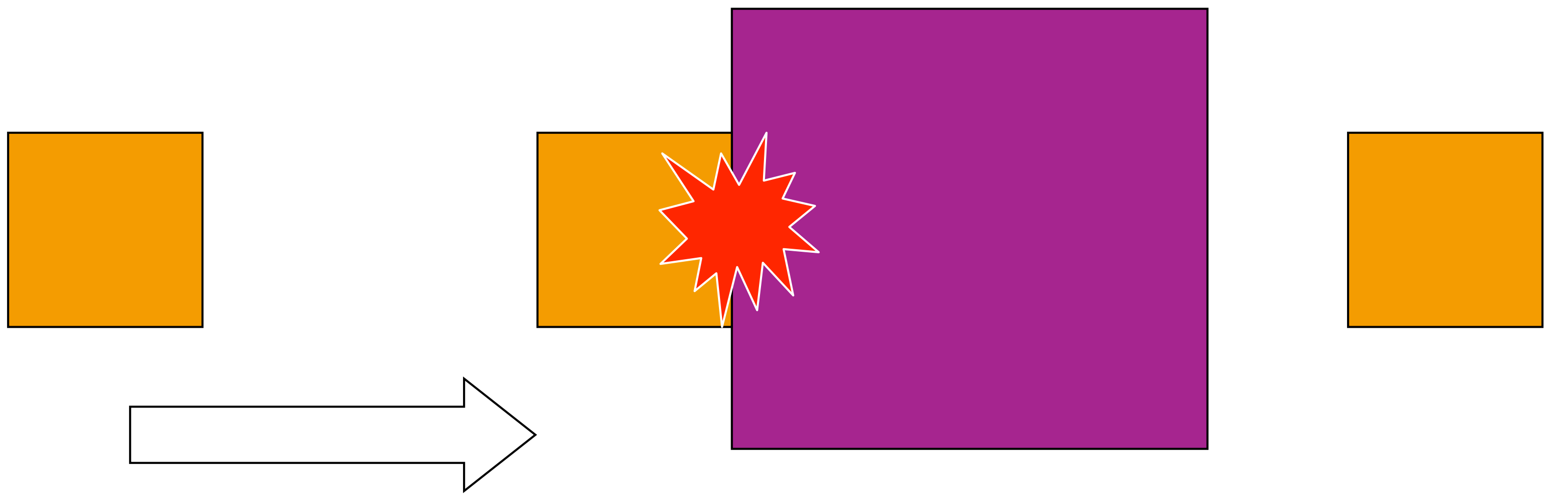
Why Perform Continuous CD?

- **The exact time and location of first contact may need to be found.**



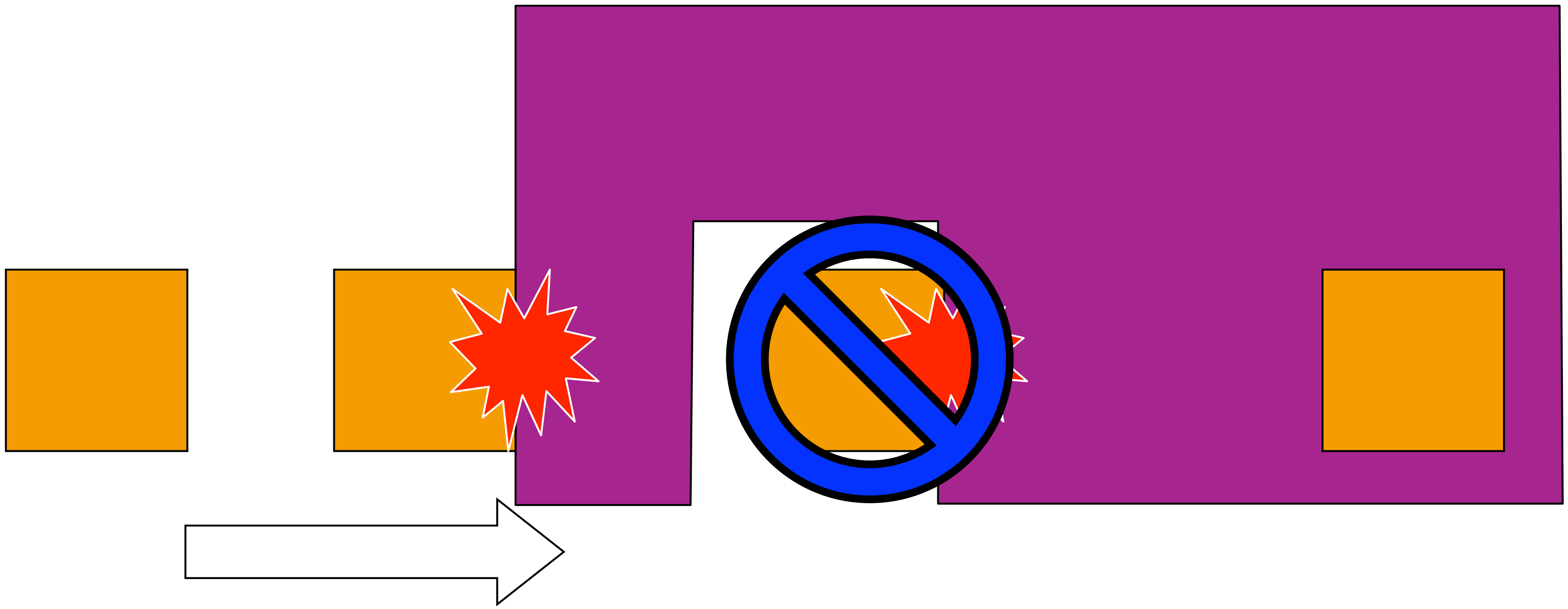
Why Perform Continuous CD?

- **Sampling at discrete intervals may miss a collision entirely.**



Why Perform Continuous CD?

- Sampling at discrete intervals may give the wrong collision!



Why Perform Continuous CD?

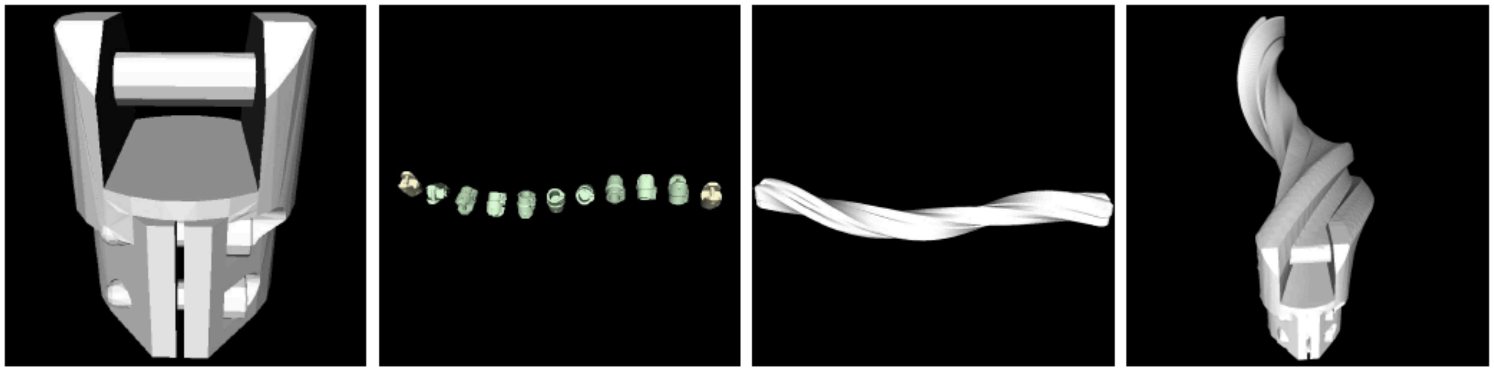
- **Most animation systems use backtracking methods**
  - Try to find point of first contact by binary search.
  - Subject to all problems from previous slides
  - Especially poor for non-solid objects (eg. cloth)

Types of motion

- Almost all CCD algorithms assume linear motion over a single time step
- Non-linear motion makes CCD computation much more expensive
  - True for both approximate and exact methods

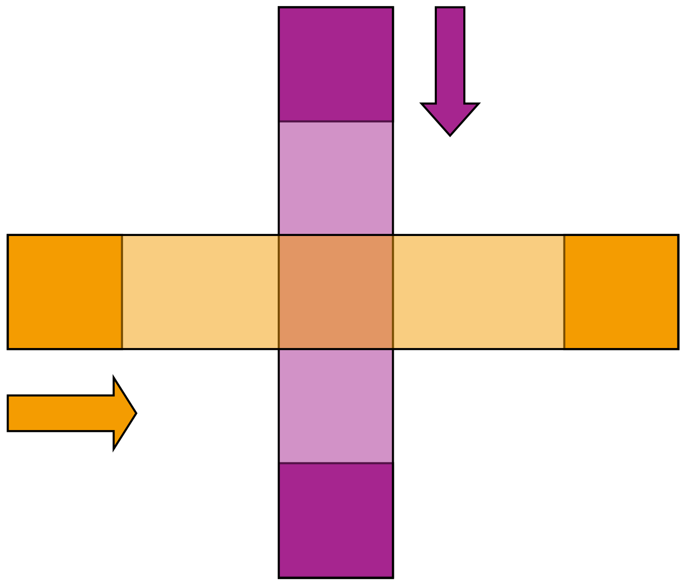
Swept-volumes

- **The motion of a primitive through space "sweeps out" a volume over a time interval**
  - **Similar to extrusion with an added rotational component.**



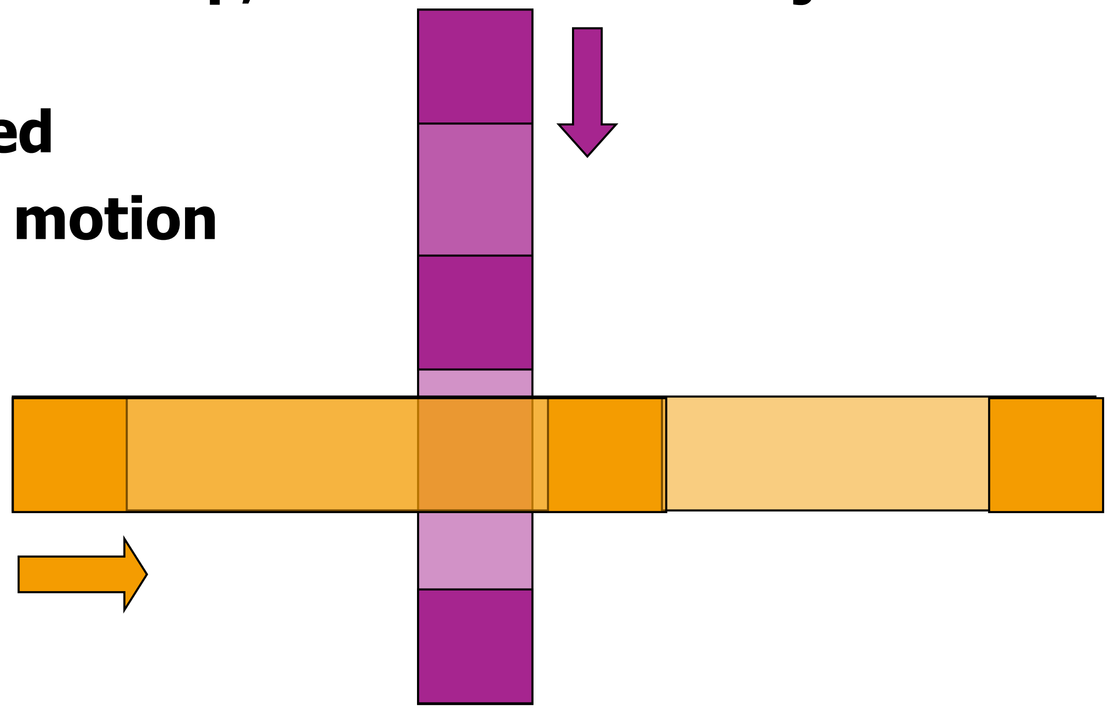
Swept Volumes

- Swept-Volumes of moving objects may be compared against each other
- This is a binary test for collision
  - Does not reveal when or where collision occurs



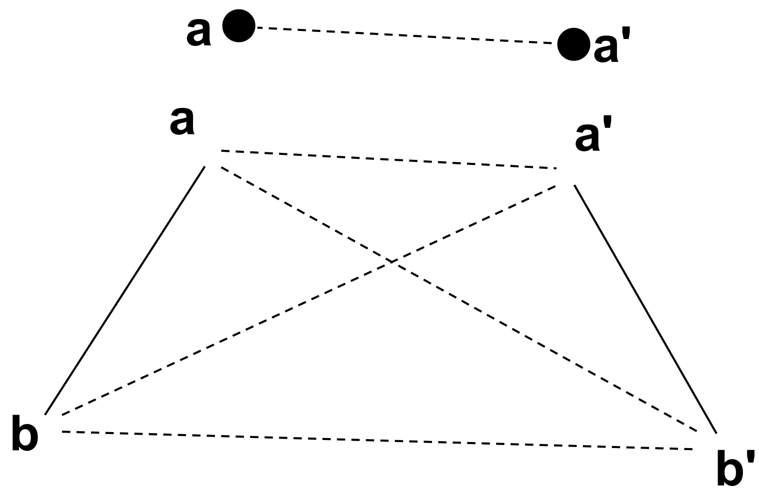
Swept Volumes

- Swept volumes are a sufficient but not necessary condition for determining if objects are collision-free
  - Swept volumes may overlap, even when the objects have not collided
  - Subdivision is needed
  - Or consider relative motion



Swept volumes in 3D space

- 1D - line
- 2D – prism

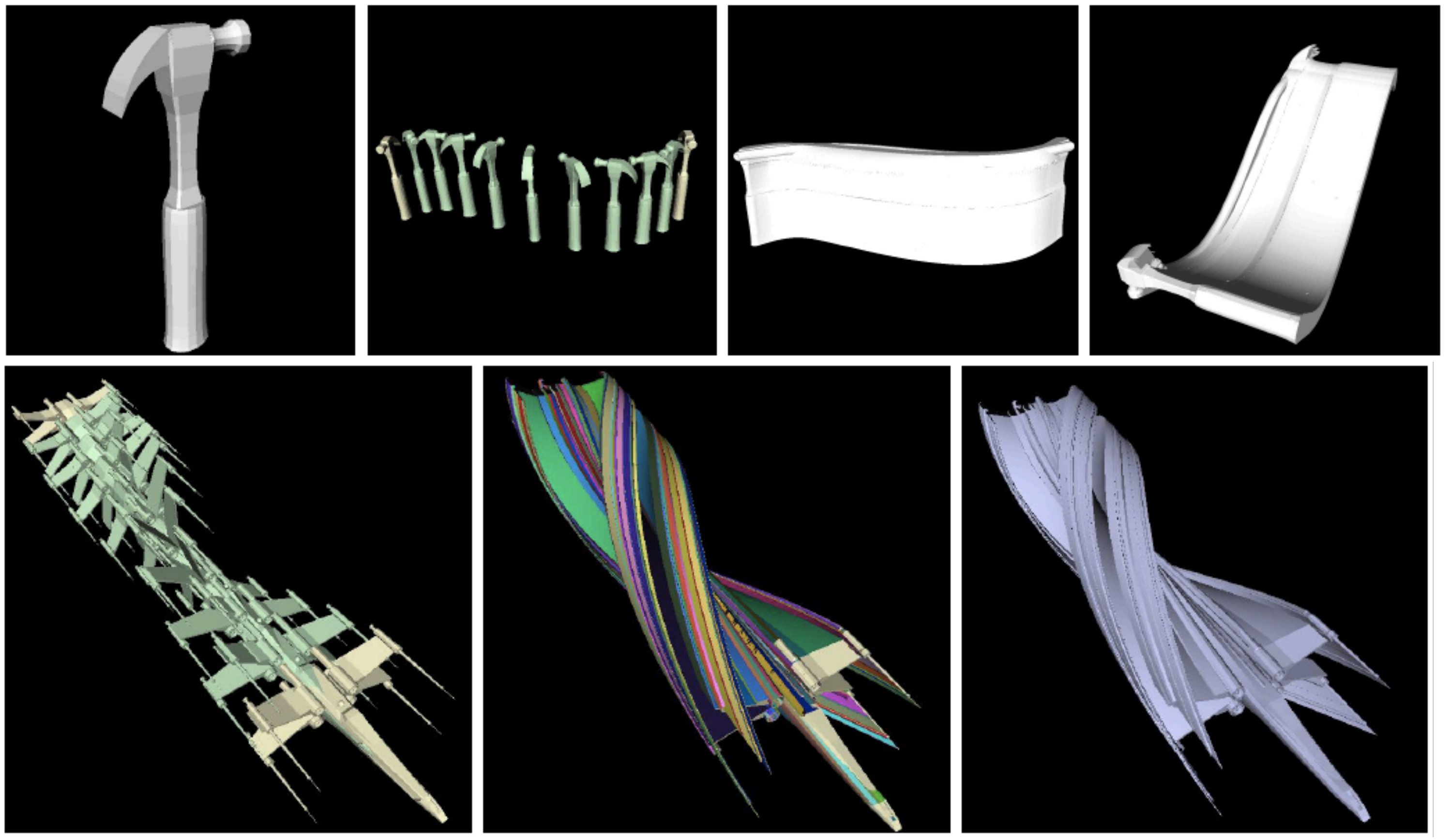


- 3D
  - becomes very complicated very quickly

Swept Volumes

- **An object in n dimensions sweeps out a volume in n+1 dimensions**
- **These volumes are very expensive to compute.**
  - **Even harder with arbitrary rotations.**

Swept Volumes



Source: "Fast Swept Volume Approximation…"

Y. Kim – ACM Solid Modelling 2003

Approximate CCD

- Rough (conservative) CCD tests can be performed via bounding volumes of the swept volumes
- Hierarchies of bounding volumes may be constructed

Convex Hulls of Swept Volumes

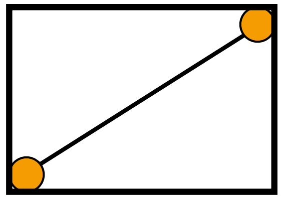
"A Safe Swept-Volume Approach to Collision Detection"

A. Foisy & V. Hayward

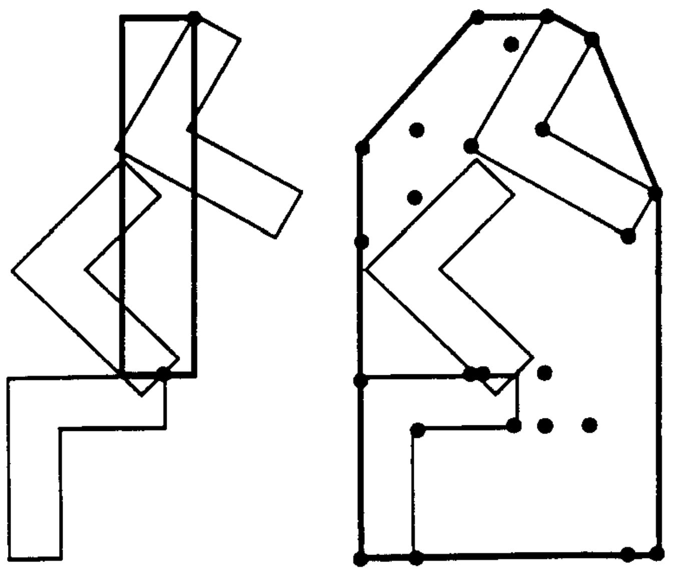
Int. Symp. on Robotics Research 1994

Convex Hulls of Swept Volumes

- The AABB of a moving vertex



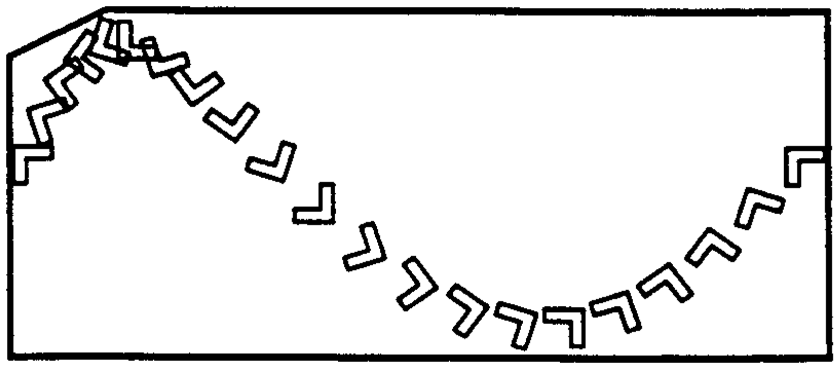
- Can find the convex hull of the AABBs of all vertices



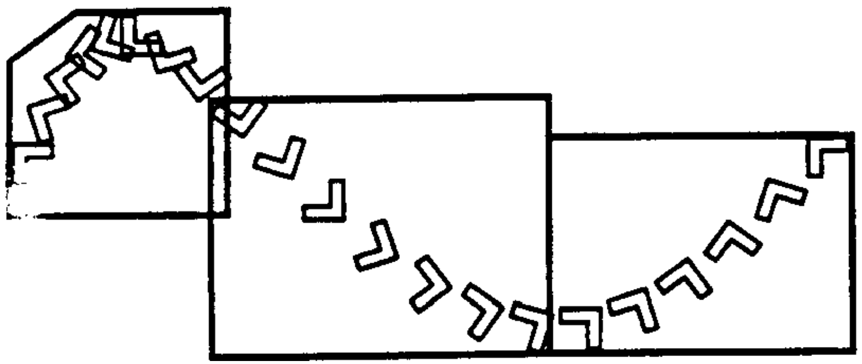
Convex Hulls of Swept Volumes

- **The convex hull of vertex AABBs is also a convex approximation to the swept volume of the moving object**

  But NOT the *convex hull* of the AABBs.



- **Can consider hierarchies of these**

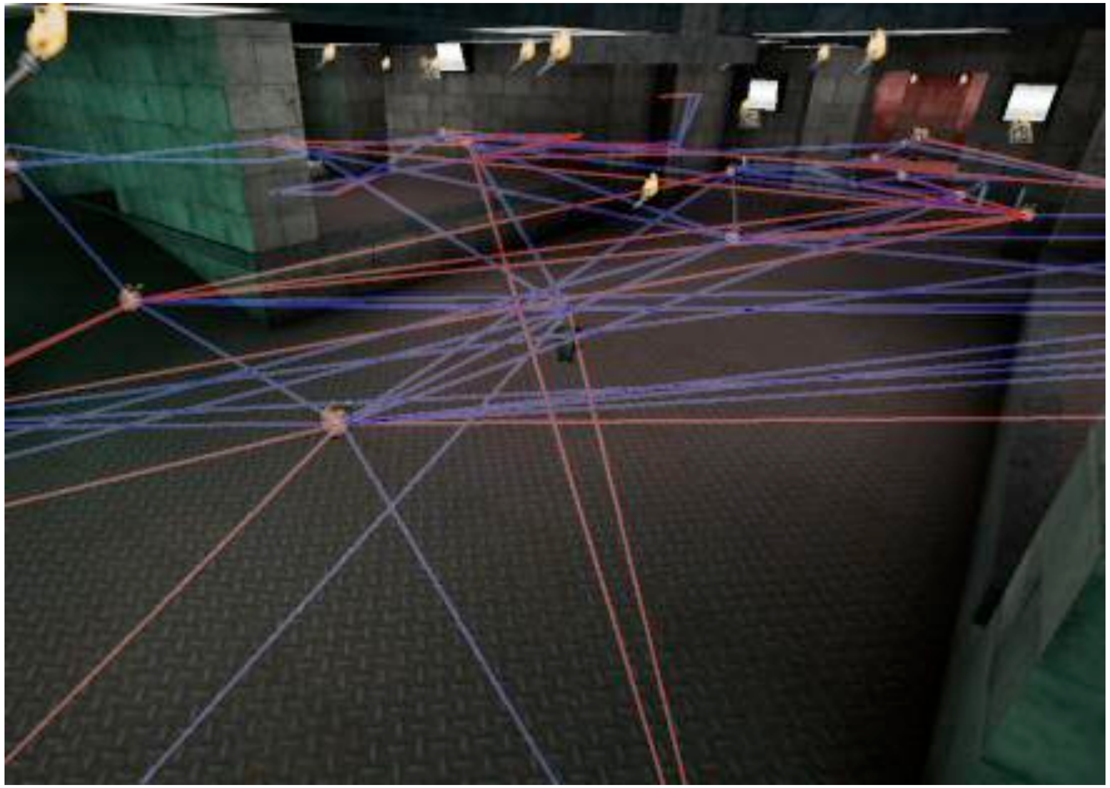


Game AI: More Problems

- What human users see is not rendered for AI.
  - No signs to read, these are just “textures”.
- Anything you aren’t intended to touch you will not be able to feel.
- There is generally no way to move to the majority of space.

Game AI: Solutions

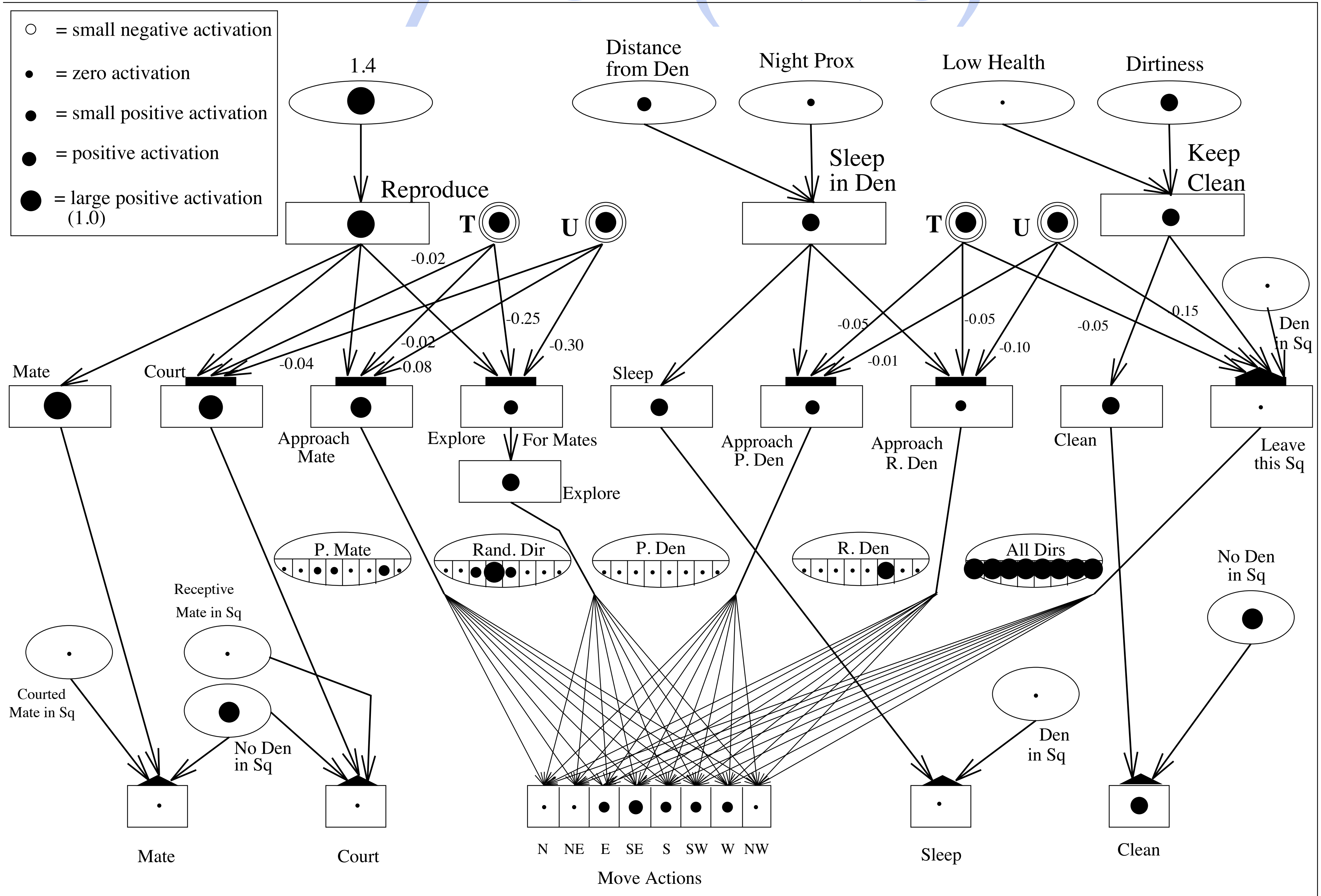
- Lots of hacks & abstractions.
  - Seldom really touch anything e.g. picking up when proximate.
  - Reduce path planning to using A* on way points.
- Still left with the minor problem of AI.



Outline

- How Game AI Is Hard in Special Ways
- How Game AI Approach Reality: Multiple Conflicting Goals
- One possible solution in detail

Tyrrell (1993)



Extended Rosenblatt and Payton Free-Flow Hierarchy

Multiple Conflicting Goals

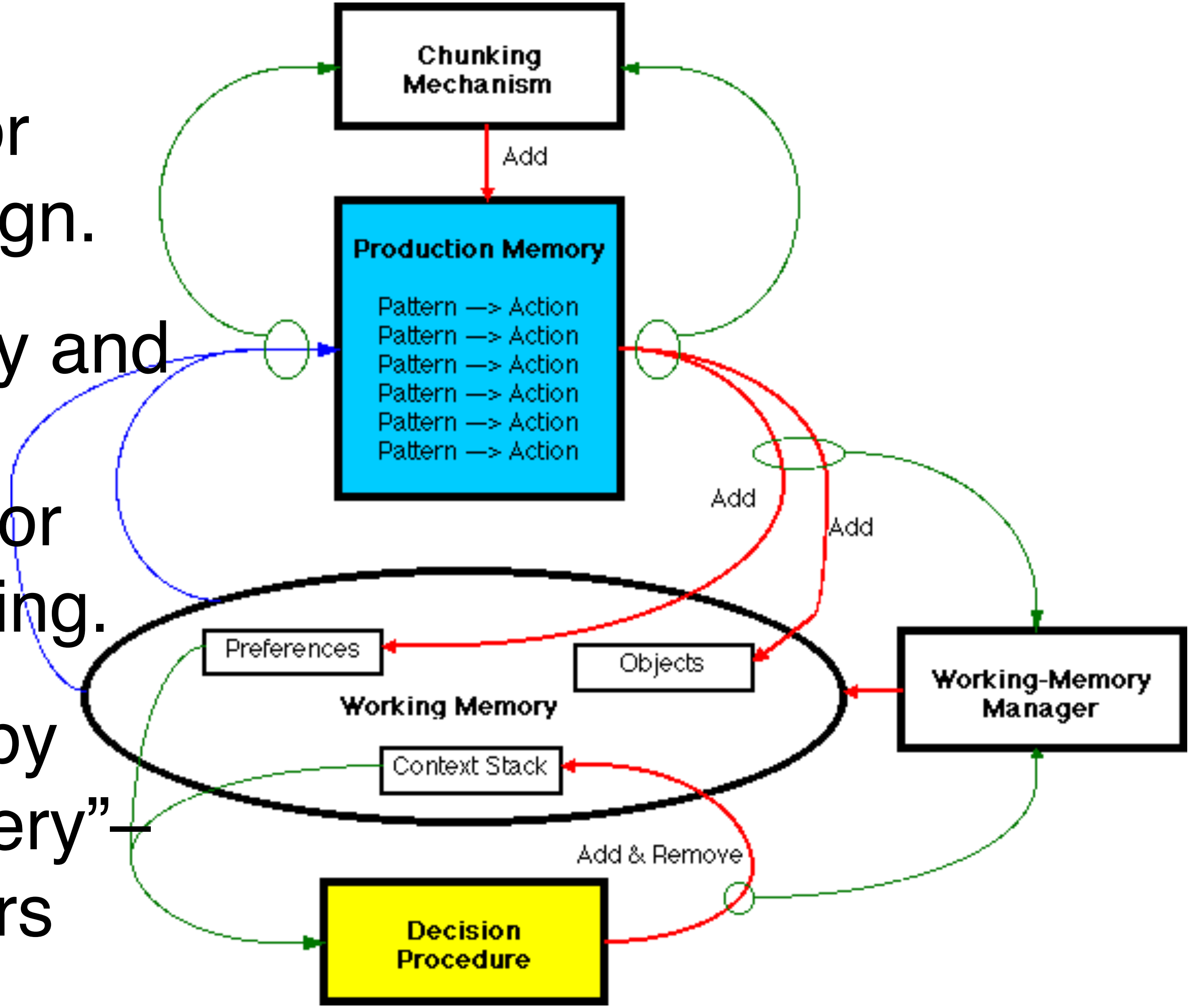
- Tyrrell's (1993) Simulated Environment (SE): eat 3 kinds of nutrients & water, avoid ungulates & predators, sleep, clean, mate.
- Robocup: stay in bounds, defend, score.
- Capture The Flag (CTF): Find enemy flag, don't get shot, defend own flag, find weapon, find health, help team mates.

Will Cognitive Architectures Help? (Revision of Lecture 4)

Imagine authoring in these…

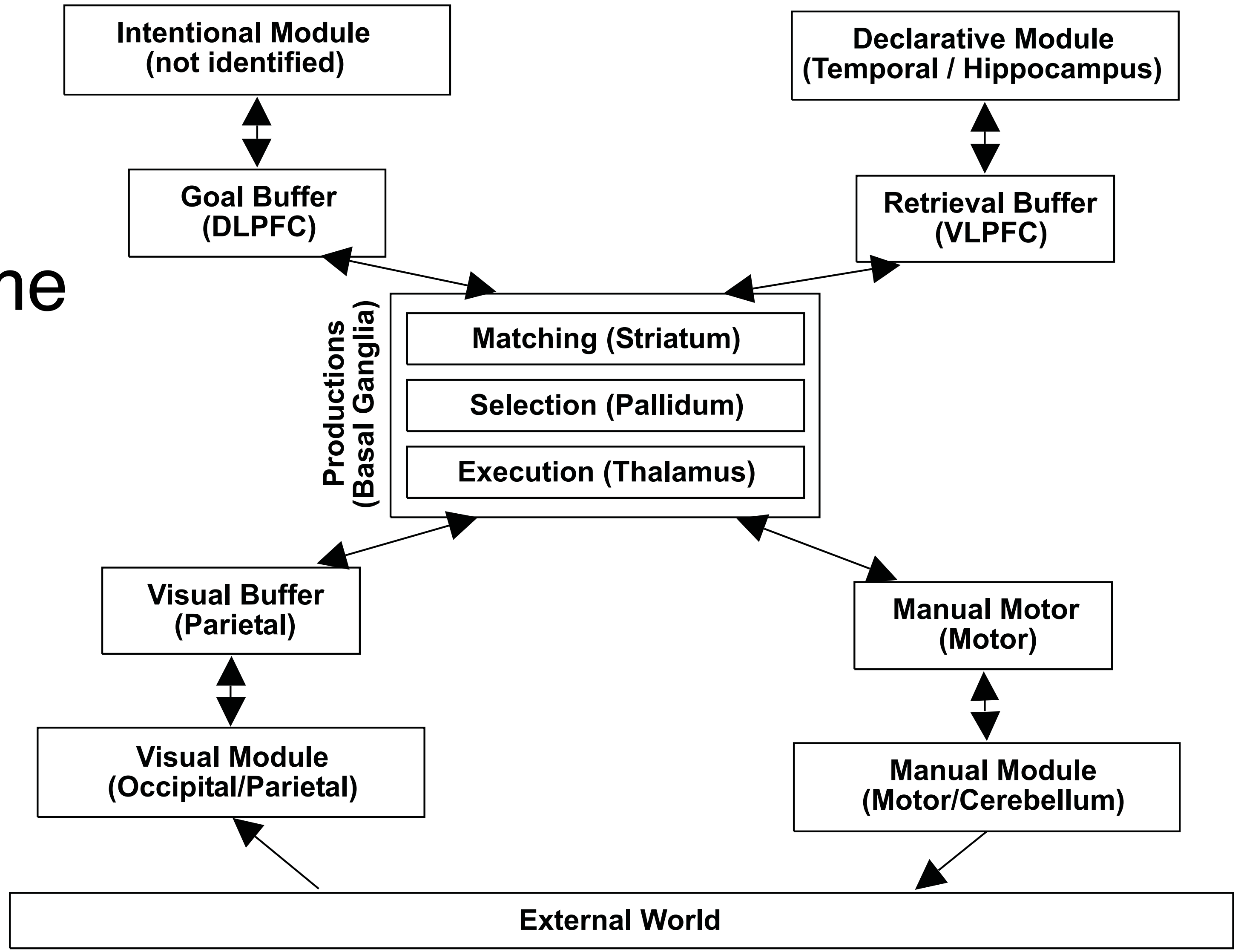
Soar

- Has been adopted for game design.
- CPU heavy and not really designed for programming.
- “teaching by brain surgery”– Olin Shivers



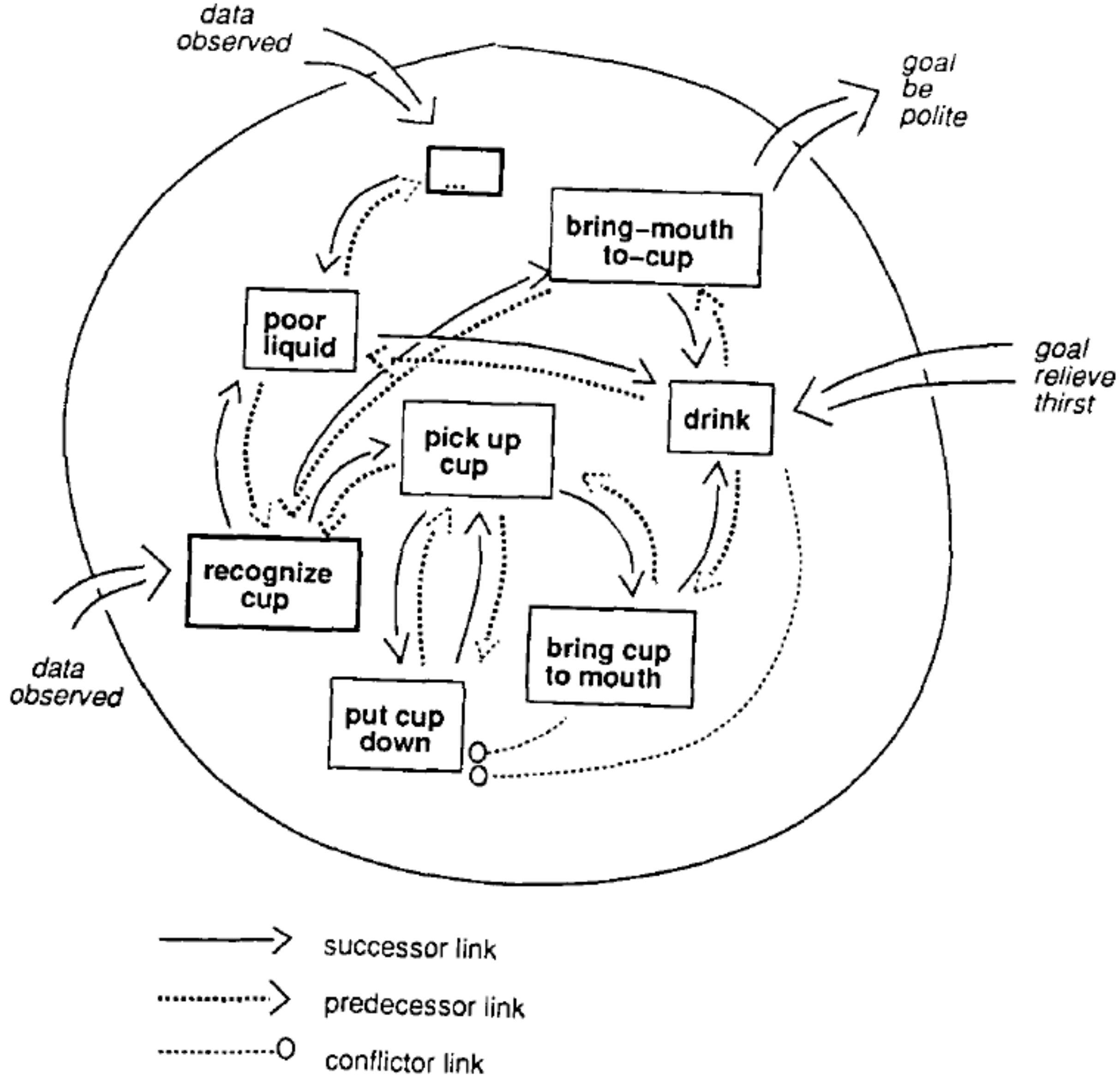
ACT-R Research Programme

- Similarly to Soar a lot of overhead for designing game AI, but has been demonstrated e.g. in RoboCup.

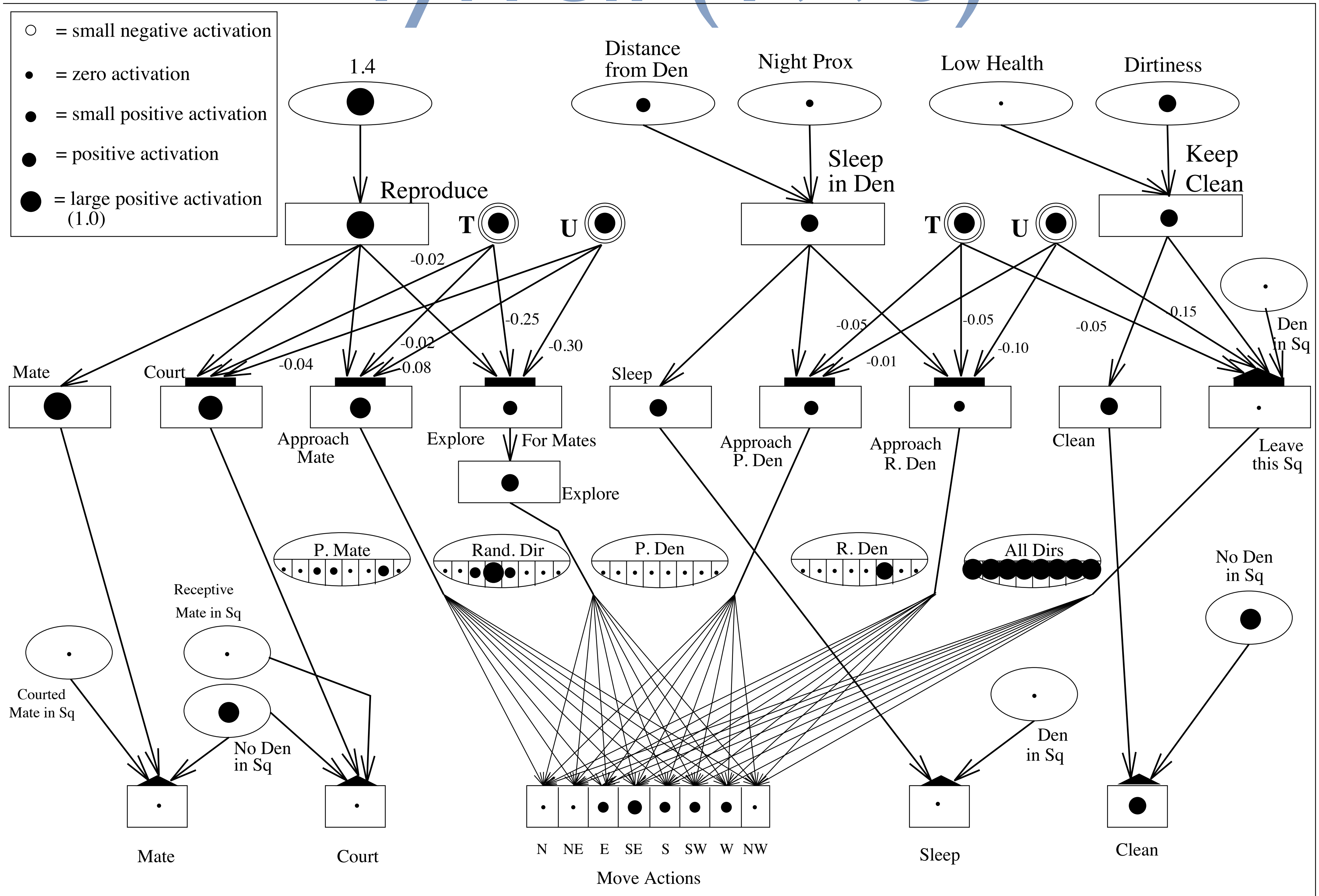


Spreading Activation Networks

- Designed for multiple goals.
- But doesn't converge to consumatory actions if too many steps in sequence required.



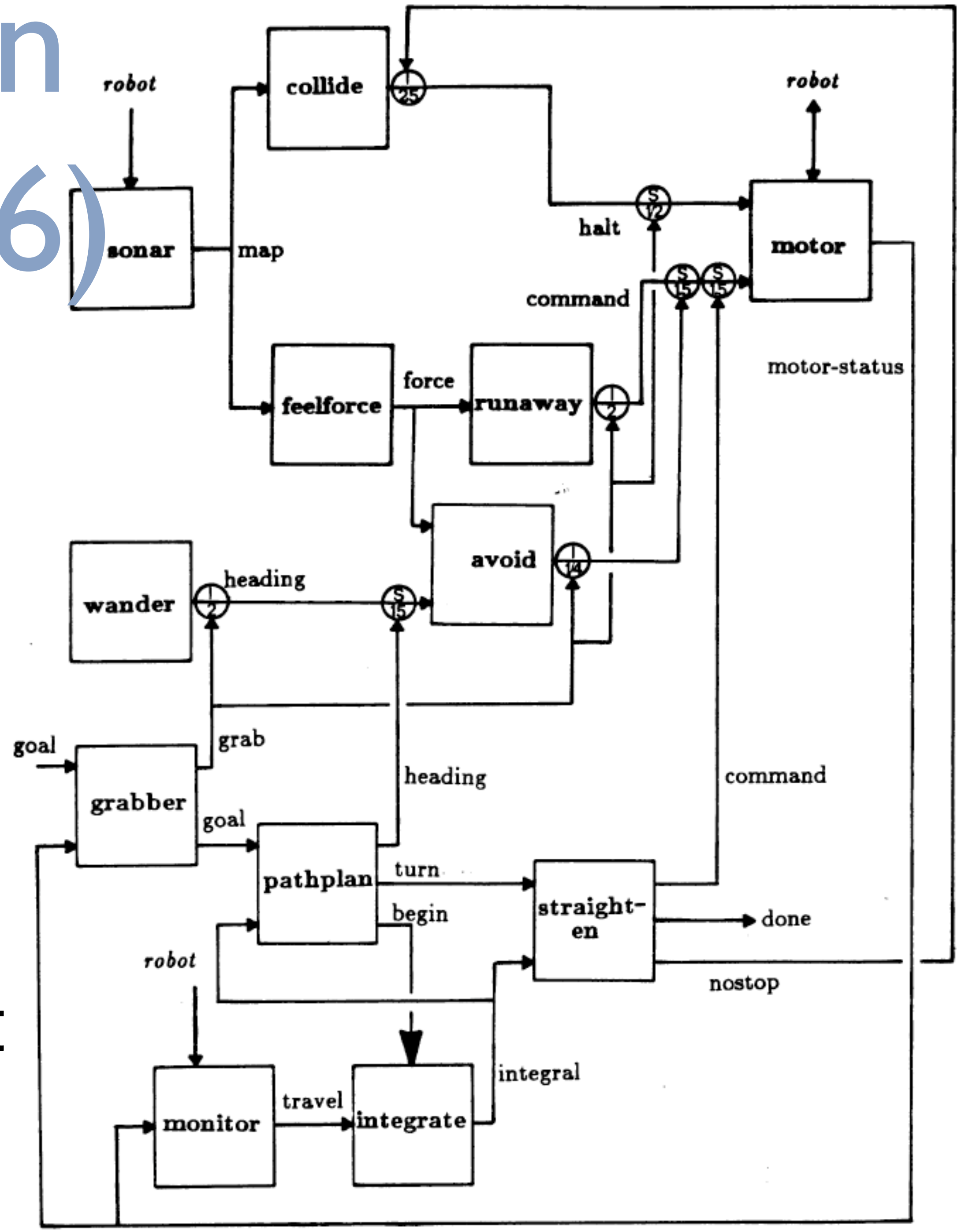
Tyrrell (1993)



Extended Rosenblatt and Payton Free-Flow Hierarchy

Subsumption (Brooks 1986)

- Multiple goals, one per level, but…
- Conflict only handled by subsumption, strict linear priority.
- Best for maintaining concurrent goals, not alternating between conflicting ones.

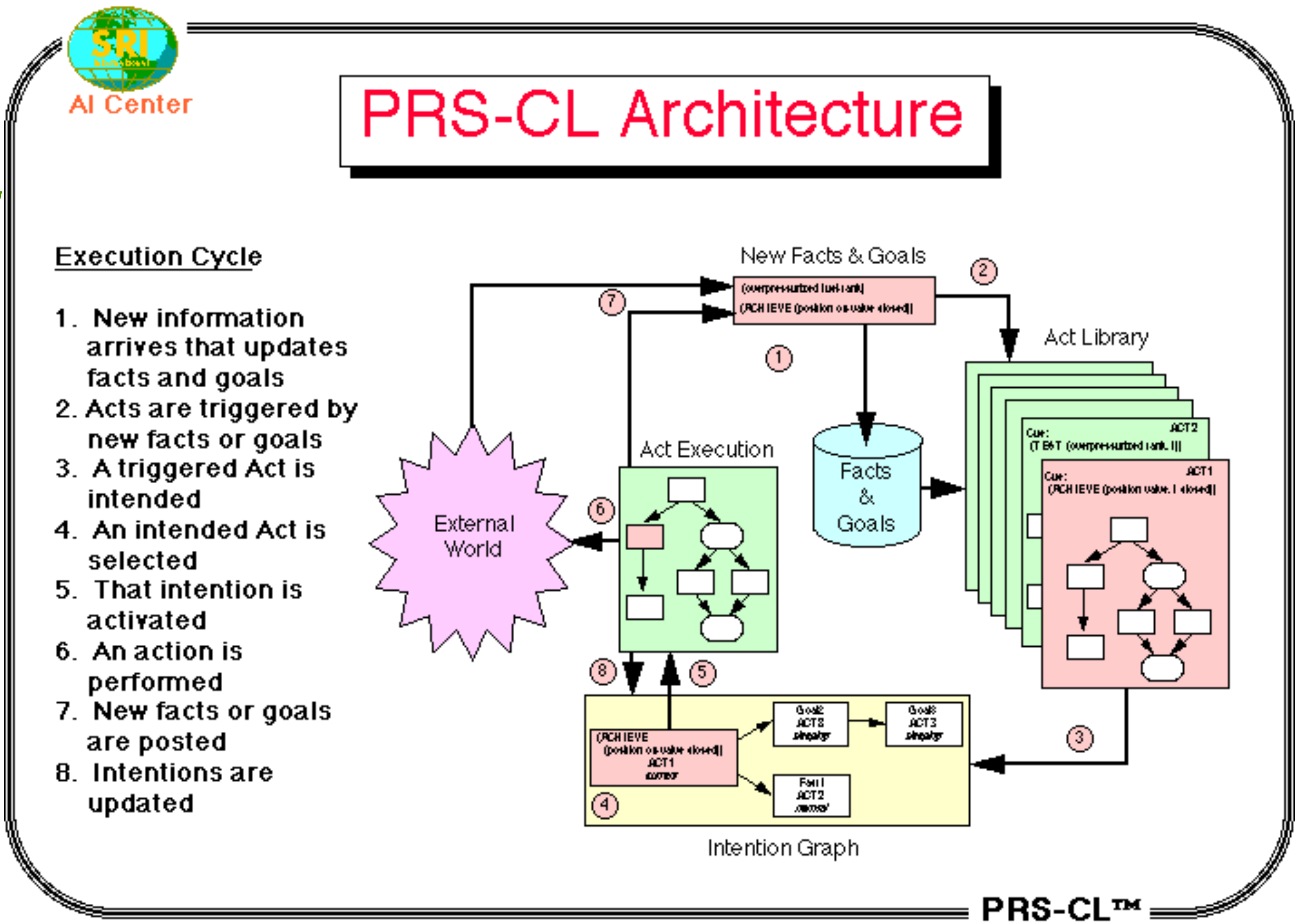


What Does Nature Do?

- Most animals allocate a packet of time to each goal (persistence vs. dithering, see emotion lecture next week.)
- E.g. (from McFarland 1981) even very hungry Siamese fighting fish will alternate time between eating and patrolling.
- Subsumption (and any architecture with perceptual aliasing issues) can't support this.

Belief, Desires, Intentions (BDI)

- *PRS really expected to pursue one goal at a time.*
- *Other BDI does get applied to games e.g. StarCraft competition, Robocup.*



Game AI as Art

- Cognitive architectures are designed for multiple conflicting goals.
  - OK for science, maybe domestic robotics.
- Game NPC ideally entertainment ∴ art.
  - Require authorship.
  - Want prestige (not just ££) of films.

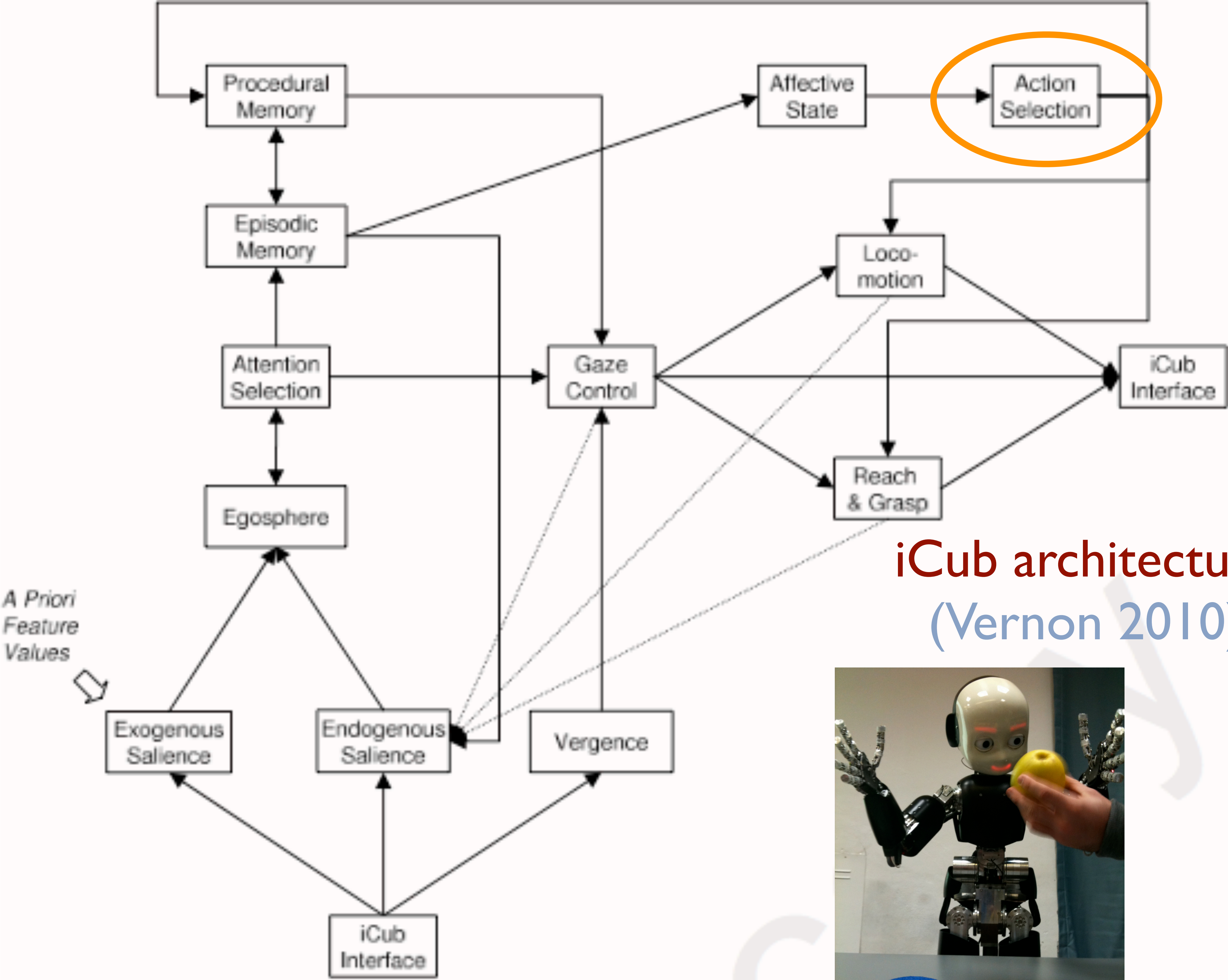
Systems AI: Motivation

- The more your AI does, the more CPU it uses.
- The less your AI knows in advance, the more things it has to try in order to learn.

$\Longrightarrow$ The more you can help programmers design / inform their AI, the better the AI will be.

What Do AI Architectures Provide?

- Search
  - May provide mechanisms for determining action selection and/or machine learning.
- Development methodologies:
  - Describe ontologies / representations;
  - Recommend development strategies.



iCub architecture

(Vernon 2010)

Outline

- How Game AI Is Hard in Special Ways
- How Game AI Approach Reality: Multiple Conflicting Goals
- One possible solution in detail

Game AI Today

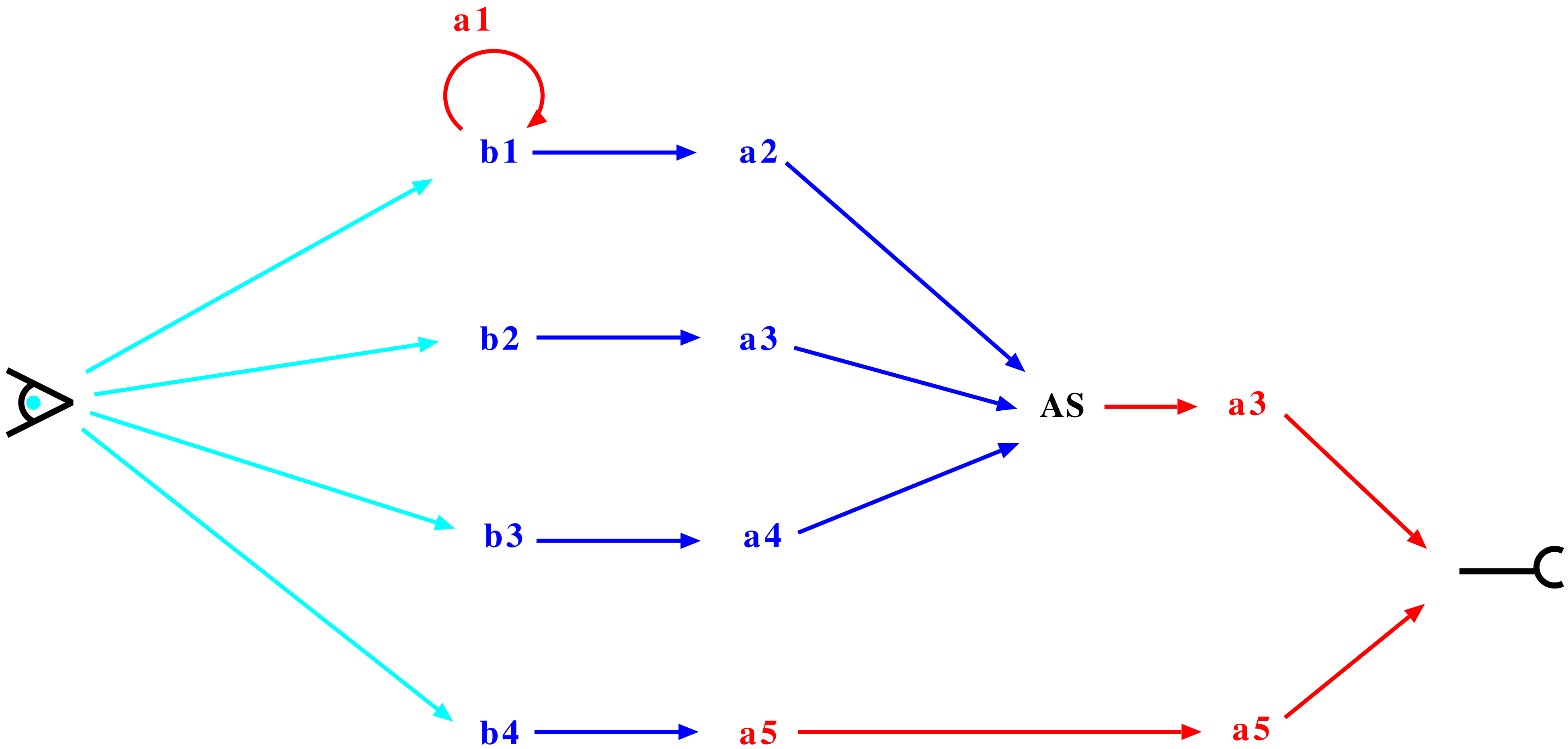
- Most popular: FSM + situation-specific learning & planning. E.g.
  - Learn human player's ability level
  - A* planning for navigating while watched.
- Up & Coming: Behaviour Trees
  - These are just action selection, more to systems AI than that.

Behavior Oriented Design

- All search (learning, planning) is done within modules with specialized representations.
- Specialized representations promote reliability of search; also determine decomposition.
- Modules provide perception, action, memory. Arbitration via hierarchical dynamic plans.
- Iterative / agile test & development cycle.

Bryson & Thórisson (2000), Bryson (2001, 2003), Grow et al (FDG 2014)

BOD Architecture



Modular behaviors generate actions, arbitrated where necessary by Action Selection based on hierarchical plans

BOD Features & Origins

- Differers from Subsumption Architecture by allowing 1) centralised, hierarchical action selection, 2) memory & 3) refactoring. Keeps lightweight perception specialised to action.
- Differs from conventional OOD by focussing on motivated action – hierarchical plans specify priorities for an agent. Keeps code reuse, module decomposition, SE focus.

Keeps modularity (key to both approaches.)

Hierarchical Action Selection

Parallel-rooted, Ordered, Slip-stack Hierarchical (POSH) action selection:

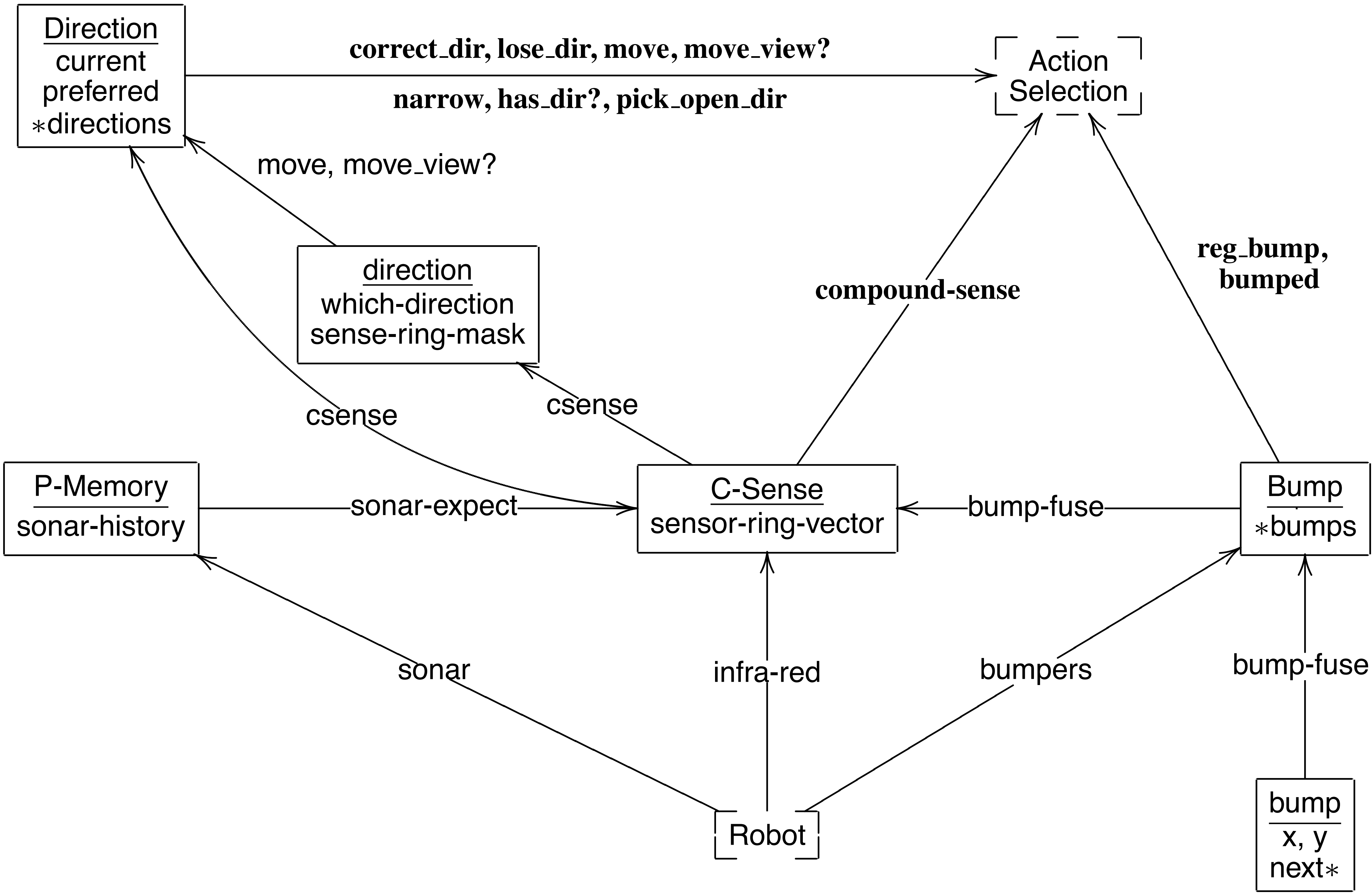
- Some things need to be checked at all times: drive collection.
- Some things only need considering in particular context: competences.
- Some things reliably follow from others: action patterns.

From Lecture 7 (Learning & Perception)

(Bryson 1997, 2001)

- Nomad 2000
- Sonars, IR, Bumpers, Odometry

Joanna J. Bryson “The Behavior-Oriented Design of Modular Agent Intelligence”, *Agent Technologies, Infrastructures, Tools, and Applications for e-Services*, R. Kowalszyk, J. P. Müller, H. Tianfield and R. Unland, eds., pp. 61–76, Springer, 2003.



Direction
current
preferred
*directions
correct_dir, lose_dir, move, move_view?
narrow, has_dir?, pick_open_dir
Action
Selection
move, move_view?
direction
which-direction
sense-ring-mask
compound-sense
reg_bump,
bumped
csense
csense
P-Memory
sonar-history
sonar-expect
C-Sense
sensor-ring-vector
bump-fuse
Bump
*bumps
sonar
infra-red
bumpers
bump-fuse
Robot
bump
x, y
next*

drive collection

life (D)

- talk [1/120 Hz] (worth_talking $\top$) → speak
- sense (C) [7 Hz]
  - bump (bumped $\top$) → yelp reg_bump back_off clear_bump lose_direction
  - look → compound_sense
- walk (C)
  - halt (has_direction $\top$) (move_view 'blocked) → lose_direction
  - start (has_direction $\bot$) → pick_open_dir
  - continue → move narrow (move_view 'clear) correct_dir
- wait → snore sleep

competence

life (D)

- talk [1/120 Hz] (worth_talking ⊤) → speak
- sense (C) [7 Hz]
  - bump (bumped ⊤) → yelp reg_bump back_off clear_bump lose_direction
  - look → compound_sense
- walk (C)
  - halt (has_direction ⊤) (move_view 'blocked) → lose_direction
  - start (has_direction ⊥) → pick_open_dir
  - continue → move narrow (move_view 'clear) correct_dir
- wait → snore sleep

action pattern

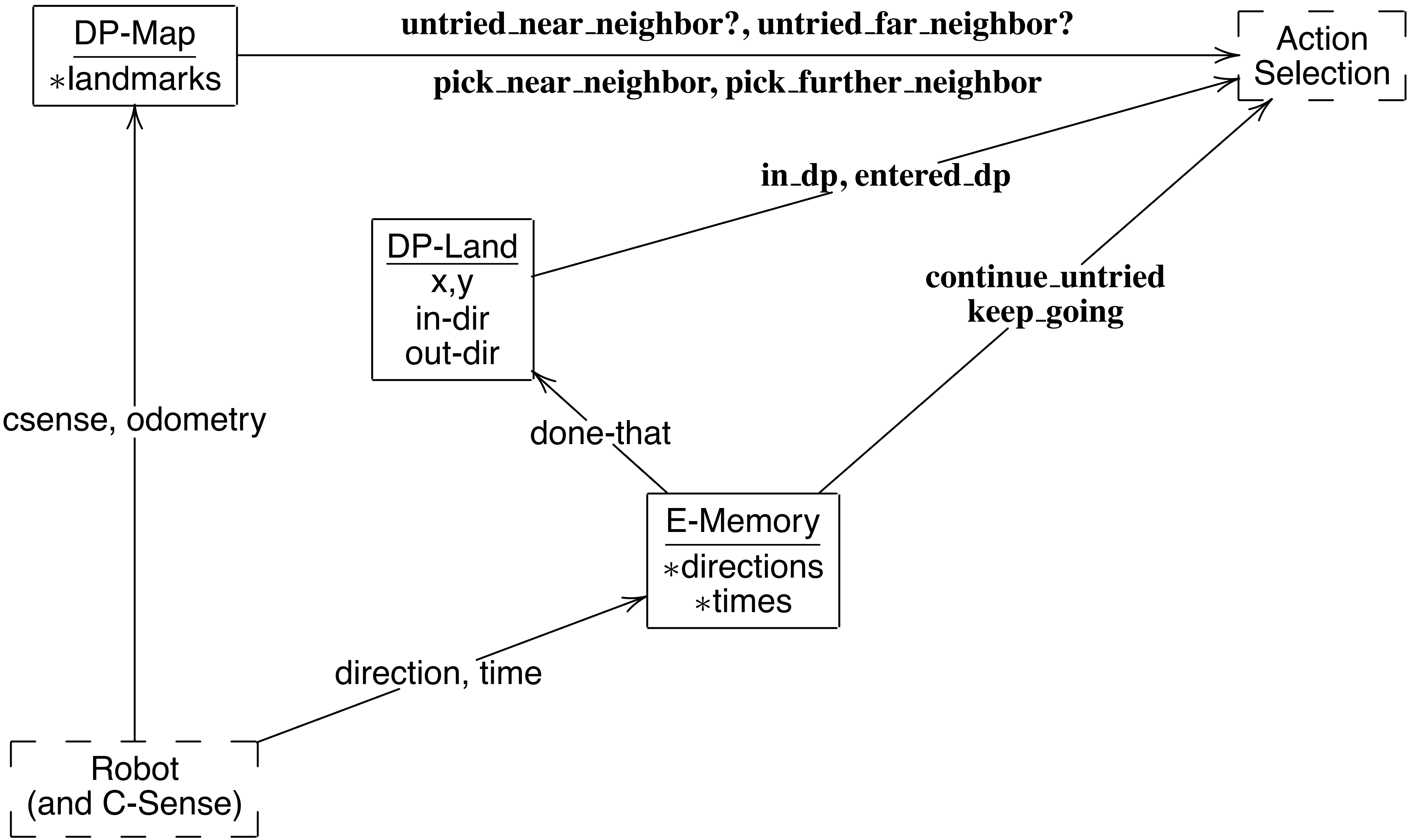
- *life* (D)
  - talk [1/120 Hz] (worth_talking ⊤)
    - speak
  - sense (C) [7 Hz]
    - bump (bumped ⊤)
      - yelp reg_bump back_off clear_bump lose_direction
    - look
      - compound_sense
  - walk (C)
    - halt (has_direction ⊤) (move_view 'blocked)
      - lose_direction
    - start (has_direction ⊥)
      - pick_open_dir
    - continue
      - move narrow (move_view 'clear) correct_dir
  - wait
    - snore sleep

sense primitives act

- *life* (D)
  - talk [1/120 Hz] (worth_talking $\top$)
    - speak
  - sense (C) [7 Hz]
    - bump (bumped $\top$)
      - yelp reg_bump back_off clear_bump lose_direction
    - look
      - compound_sense
  - walk (C)
    - halt (has_direction $\top$) (move_view 'blocked)
      - lose_direction
    - start (has_direction $\bot$)
      - pick_open_dir
    - continue
      - move narrow (move_view 'clear) correct_dir
  - wait
    - snore sleep

expanded competence

- walk (C)
  - halt (has_direction $\top$) (move_view 'blocked) — lose_direction
  - cogitate_route (C)
    - enter_dp (in_dp $\bot$) (entered_dp $\bot$) — lose_direction greet_dp
    - leave_dp (in_dp $\top$) (entered_dp $\top$) — dismiss_dp
  - pick_direction (C)
    - look_up (untried_near_neighbor $\top$) — pick_near_neighbor
    - keep_going (continue_untried $\top$) — pick_previous_direction
    - desperate_look_up (untried_far_neighbor $\top$) — pick_further_neighbor
  - start (has_direction $\bot$) — ask_directions
  - continue — move narrow (move_view 'clear) correct_dir


DP-Map
*landmarks
untried_near_neighbor?, untried_far_neighbor?
pick_near_neighbor, pick_further_neighbor
Action Selection
in_dp, entered_dp
DP-Land
x,y
in-dir
out-dir
continue_untried
keep_going
csense, odometry
done-that
E-Memory
*directions
*times
direction, time
Robot
(and C-Sense)

Things All Successful Real-Time AI has

1. Modularity
2. A means to redirect attention rapidly
3. A means to specify action selection for fiddly detailed subplans.

(Bryson 2000, JETAI)

Hierarchical Action Selection

Parallel-rooted, Ordered, Slip-stack Hierarchical (POSH) action selection:

- Some things need to be checked at all times: drive collection. Redirect Attention
- Some things only need considering in particular context: competences.
- Some things reliably follow from others: action patterns.

} Detailed AS

Modules not a part of AS

POSH & Behavior Trees

- Behavior Trees (Mateas et al 2003-*present*) currently a dominant approach to game AI, replacing FSM as leading form of control (AIGameDev.com).
- Like POSH: action≃act, condition≃sense, sequence≃action pattern, priority≃competence, parallel (supported throughout, no special name); temporal decorators: drive scheduling.
- Any hierarchical AS will work for BOD.

BOD Development Cycle

1. Initial decomposition ⇒ specification.

2. Scale the system.

   i. Code one behavior and/or plan.

   ii. Test and debug code (test earlier plans).

   iii. Simplify the design.

3. Revise the specification.

4. Iterate.

BOD Development Cycle

1. Initial decomposition ⇒ specification.
2. Scale the system.
   i. Code one behavior and/or plan.
   ii. Test and debug code (test earlier plans).
   iii. Simplify the design.
3. Revise the specification.
4. Iterate.

1. Specify (high-level) what the agent will do.
2. Describe activities as sequences of actions. competences and action patterns
3. Identify sensory and action primitives from these sequences.
4. Identify the state necessary to enable the primitives, cluster primitives by shared state. behavior modules
5. Identify and prioritize goals / drives. drive collection
6. Select a first (next) behavior to implement.

Simplify the Design

Trade off representations: plans vs. behaviors

- Use simplest plan structure unless redundancy (split primitives for sequence, add variable state in modules).
- If competences too complicated, introduce primitives or create more hierarchy.
- Split large behaviors, use plans to unify.
- All variable state in modules (deictic).

Simplify the Design

Use the simplest representations.

- Plans:
  - primitives, action patterns, competences.
  - drives only if need to always check.
- Behavior modules / memory:
  - none, deictic, specialized, general.

(Bryson, AgeS 2003)

BOD Arch. Lessons

- Modularity: problem spaces, combat combinatorics, allow locally-optimal representations.
  - Should use ordinary (OO) code (arbitrarily powerful but also access to primitives.)
- Hierarchical action selection for arbitration.
- Dedicated, high-frequency goal / attention switching, compensates for hierarchical AS.
- Agile development, refactoring (Beck 2000).

Modularity is Good, But Not Enough

Get Fuzzy (Conley 2006)